OPERATION AND MAINTENANCE INSTRUCTIONS

ASC-4X CENTRAL PROCESSOR (CP)
Volume 2

TEXAS INSTRUMENTS
INCORPORATED

ASC

OPERATION AND MAINTENANCE
INSTRUCTIONS

ASC-4X CENTRAL PROCESSOR (CP)
Volume 2

Equipment Group
P.O. Box 2909
Austin, Texas 78767

TEXAS INSTRUMENTS
INCORPORATED

934726-2
January 1976

INTRODUCTION

This manual is volume 2 of a two-volume set of operation and maintenance instructions for the 4-pipe Central Processor, which is used in the Advanced Scientific Computer (ASC) system, manufactured by Texas Instruments Incorporated.

This volume contains the following sections and appendixes:

Section 5 - Maintenance

Section 6 - Parts Listing

Section 7 - Diagrams

Appendix A - Details Maps

Appendix B - 4XIPU Listings and Circuit Board Descriptions

Appendix C - 4XCP Hazard Conditions

Appendix D - Hard Core

The part number for Volume 1 is 931443-2.

SECTION V
MAINTENANCE

5-1 GENERAL

The maintenance philosophy for the 4-pipe Central Processor consists of running a a series of diagnostic tests to fault isolate to a functional section and the use of AU Functional Logic Descriptions (FLDs), harness lists (in Fiche form), and flowcharts in conjunction with conventional test equipment to fault isolate down to the replaceable card level.

The FLDs available for the 4-pipe CP include:

- IPU4 FLD, part no. 931490
- AU4 FLD, part no. 931491
- MBU4 FLD, part no. 931492

The diagnostic tests available for maintenance of the 4-pipe CP are described in the multi-volume set of ASC System Diagnstics available at each ASC site.

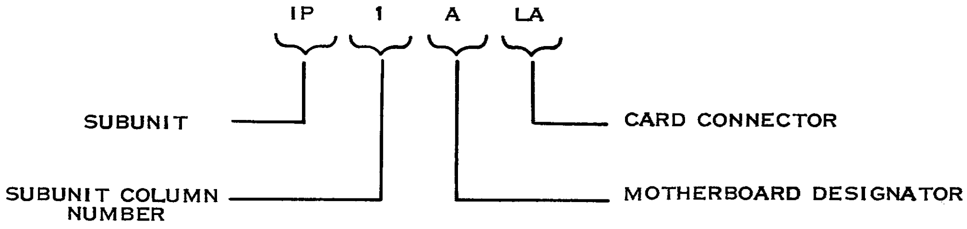
SECTION VI
PARTS LISTING

6-1 INTRODUCTION

This section provides a list of replaceable logic cards and their part numbers for the ASC 4X-Central Processor. The list is intended as a guide for ordering new cards and installing them in the CP. Items such as IC's, screws, washers, etc., are not included.

6-2 LOGIC CARDS

Logic cards for the CP are contained in ten chassis: two IPU's, four MBU's, and four AU's. Each AU and MBU chassis has three motherboards, and each IPU contains two motherboards which hold the logic cards. These motherboards are designated with the letters A, B, and C from top to bottom, respectively. Each card slot in a motherboard is designated with a two letter label, LA to LV. These designators are used to identify the particular card location in the CP. Figure 6-1 illustrates the information contained in a card location designator. The first two letters refer to the chassis. The next character refers to the column. The fourth character designates the motherboard in that column, and the last two letters identify the card slot on that motherboard. Table 6-1 lists all CP logic cards and are arranged by card location. Only one of the four identical MBU and AU pipes is listed.



(A) 115137

Figure 6-1. Card Location Information

Table 6-1. Central Processor Logic Cards

Card Location	Function	Part Number	Card Location	Function	Part Number
IP2ALA	TERMCRD	650296-1	IP2BLA	DUMMY	695011-1
IP2ALB	I4ZHAZ(1)	923558-1	LB	I4HDCORE	923570-1
LC	I4ZHAZ(0)	923558-1	LC	I4CMREQ	923573-1
LD	I4RHAZ(0)	923555-1	LD	I4INFACE(2)	923567-1
LE	I4ZHAZ(3)	923558-1	LE	I4INFACE(1)	923567-1
LF	I4HZAZ(2)	923558-1	LF	I4INFACE(0)	923567-1
LG	I4ZHAZ(5)	923558-1	LG	I4PIPTOP	923576-1
LH	I4ZHAZ(4)	923558-1	LH	I4MISC	923582-1
LI	I4ZHAZ(1)	923558-1	LI	I4VECLAS	923579-1
LJ	I4ZHAZ(7)	923558-1	LJ	I4LVL3	923585-1
LK	I4ZHAZ(6)	923558-1	LK	TERMCRD	650926-1
LL	I4ZHAZ(11)	923558-1	LL	I4ROUTE1	923588-1
LM	I4RHAZ(2)	923555-1	LM	I4ROUTE3	923594-1
LN	I4ZHAZ(8)	923558-1	LN	I4ROUTE2	923591-1
LO	I4ZHAZ(9)	923558-1	LO	I4INFACE(3)	923567-1
LP	I4ZHAZ(10)	923558-1	LP	BUCTL4-2	922700-2
LQ	I4ZHAZ(12)	923558-1	LQ	DUMMY	695011-1
LR	I4RHAZ(3)	923555-1	LR	DUMMY	695011-1
LS	I4ZHAZ(13)	923558-1	LS	DUMMY	695011-1
LT	I4ZHAZ(15)	923558-1	LT	DUMMY	695011-1
LU	I4ZHAZ(14)	923558-1	IP2BLU	LOGCLK-5	650356-5
IP2ALV	TERMCRD	650296-1			
IP3ALA	DUMMY	695011-1	IP3BLA	DUMMY	695011-1
IP3ALB	DUMMY	695011-1	LB	DUMMY	695011-1
LC	I4FILE(3)	923549-1	LC	DUMMY	695011-1
LD	I4FILE(7)	923549-1	LD	I4PIPE(4)	923561-1
LE	I4FILE(15)	923549-1	LE	I4PIPE(3)	923561-1
LF	I4FILE(14)	923549-1	LF	I4PIPE(2)	923561-1
LG	I4FILE(13)	923549-1	LG	I4STATUS	923564-1
LH	I4FILE(12)	923549-1	LH	I4PIPE(0)	923561-1
LI	I4FILE(11)	923549-1	LI	I4PIPE(1)	923561-1
LJ	TERMCRD	650296-1	LJ	TERMCRD	650296-1
LK	I4FILE(10)	923549-1	LK	ROMCRD L3	650299-228
LL	I4FILE(8)	923549-1	LL	BUCTL4-2	922700-2
LM	I4ADDR(1)	923552-1	LM	DUMMY	695011-1
LN	I4ADDR(0)	923552-1	LN	LOGCLK-7	650356-7
LO	I4FILE(6)	923549-1	LO	ROMCRD L2	650229-227
LP	I4FILE(5)	923549-1	LP	I4PIPE(5)	923561-1
LQ	I4FILE(4)	923549-1	LQ	I4PIPE(6)	923561-1
LR	I4FILE(9)	923549-1	LR	I4PIPE(7)	923561-1
LS	I4FILE(1)	923549-1	LS	I4MHCA	932005-1
LT	I4FILE(0)	923549-1	LT	DUMMY	695011-1
LU	I4FILE(2)	923549-1	LU	DUMMY	695011-1
IP3ALV	DUMMY	695011-1	IP3BLV	DUMMY	695011-1

Table 6-1. Central Processor Logic Cards (Continued)

Card Location	Function	Part Number	Card Location	Function	Part Number
MB1ALA	ROMCRD(0)	650299-207	MB1BLA	DUMMY	695011-1
B	ROMCRD(8)	650299-215	B	DUMMY	695011-1
C	BUROM(0)	686490-1	C	DUMMY	695011-1
D	ROMCRD(9)	650299-216	D	BUZAG	650362-1
E	ROMCRD(1)	650299-208	E	BUDATA(0)	650365-1
F	ROMCRD(2)	650299-209	F	BUDATA(1)	650365-1
G	ROMCRD(10)	650299-217	G	BUDATA(2)	650365-1
H	BUROM(1)	686490-1	H	BUDATA(3)	650365-1
I	ROMCRD(11)	650299-218	I	BUDATA(4)	650365-1
J	ROMCRD(3)	650299-210	J	BUDATA(5)	650365-1
K	BUCTL4-2	922700-2	K	BUDATA(6)	650365-1
L	ROMCRD(4)	650299-211	L	BUDATA(7)	650365-1
M	ROMCRD(12)	650299-219	M	BUDATA(8)	650365-1
N	BUROM(2)	686490-1	N	BUDATA(9)	650365-1
O	ROMCRD(13)	650299-220	O	BUDATA(10)	650365-1
P	ROMCRD(5)	650299-212	P	BUDATA(11)	650365-1
Q	ROMCRD(6)	650299-213	Q	BUDATA(12)	650365-1
R	ROMCRD(14)	650299-221	R	BUDATA(13)	650365-1
S	BUROM(3)	686490-1	S	BUDATA(14)	650365-1
T	ROMCRD(15)	650299-222	T	BUDATA(15)	650365-1
U	ROMCRD(7)	650299-214	U	TERMCRD	650296-1
MB1ALV	TERMCRD	650296-1	MB1BLV	DUMMY	695011-1
MB1CLA	BUCTL3A	931981-1	AU1ALA	TERMCRD	650296-1
B	BUCMR	932026-1	B	AU4XSELA(1)	931996-1
C	BUCTL1	922691-1	C	AU4XSELA(0)	931996-1
D	BUCTL4A	931984-1	D	AU4ADDA(0)	931936-1
E	TERMCRD	650296-1	E	AU4ADDA(1)	931936-1
F	BUCAFA(0)	931966-1	F	AU4ADDA(2)	931936-1
G	BUCAFA(1)	931966-1	G	AU4ADDA(3)	931936-1
H	BUADDRA(4)	931939-1	H	AU4ADDA(4)	931936-1
I	BULOOP(1)	686478-1	I	AU4ADDA(5)	931936-1
J	BULOOP(0)	686478-1	J	AU4ADDA(6)	931936-1
K	BULOOP(3)	686478-1	K	AU4ADDA(7)	931936-1
L	BULOOP(2)	686478-1	L	AU4CTLIA	929096-1
M	BUADDRA(3)	931939-1	M	AU4ADDA(8)	931936-1
N	BUADDRA(2)	931939-1	N	AU4ADDA(9)	931936-1
O	TERMCRD	650296-1	O	AU4ADDA(10)	931936-1
P	BUADDRA(1)	931939-1	P	AU4ADDA(11)	931936-1
Q	BUADDRA(0)	931939-1	Q	AU4ADDA(12)	931936-1
R	DUMMY	695011-1	R	AU4ADDA(13)	931936-1
S	LOGLK-7	650356-1	S	AU4ADDA(14)	931936-1
T	LOGCLK-3	650356-1	T	AU4ADDA(15)	931936-1
U	BUMHC1	710258-1	U	TERMCRD	650296-1
MB1CLV	DUMMY	695011-1	AU1ALV	LOGCLK-4	650356-4

Table 6-1. Central Processor Logic Cards (Continued)

Card Location	Function	Part Number	Card Location	Function	Part Number
AUIBLA	TERMCRD	650296-1	AUICLA	TERMCRD	650296-1
LB	AUOUTB(0)	931978-1	B	AUMULTA(0)	929335-1
LC	AUOUTB(1)	931978-1	C	AUSUMD(10)	686487-1
LD	AUOUTB(2)	931978-1	D	AUMULTA(1)	929335-1
LE	AUOUTB(3)	931978-1	E	AUSUMD(8)	686487-1
LF	AUOUTB(4)	931978-1	F	AUSUMD(4)	686487-1
LG	AUOUTB(5)	931978-1	G	AUMULTA(2)	929335-1
LH	AUOUTB(6)	931978-1	H	AUSUMD(2)	686487-1
LI	AUOUTB(7)	931978-1	I	AUSUMD(7)	686487-1
LJ	AUCTL3B	931969-1	J	AUMULTA(3)	929335-1
LK	AUROMFFB	931972-1	K	AUSUMD(6)	686487-1
LL	AUCTL2B	931954-1	L	AUSUMD(5)	686487-1
LM	AUCTL4A	931945-1	M	AUMULTA(4)	929335-1
LN	AUNORMA(0)	929338-1	N	AUSUMD(1)	686487-1
LO	AUNORMA(1)	929338-1	O	AUMULTA(5)	929335-1
LP	AUNORMA(2)	929338-1	P	AUSUMD(9)	686487-1
LQ	AUNORMA(3)	929338-1	Q	AUSUMD(3)	686487-1
LR	AUNORMA(4)	929338-1	R	AUMULTA(6)	929335-1
LS	AUNORMA(5)	929338-1	S	AUSUMD(0)	686487-1
LT	AUNORMA(6)	929338-1	T	AUMULTA(7)	929335-1
LU	AUNORMA(7)	929338-1	U	AUCTL5B	929093-1
AUIBLV	TERMCRD	650296-1	AUICLV	TERMCRD	650296-1

SECTION VII

DIAGRAMS

7-1 GENERAL

ASC Hardware Documentation Control at the TI Austin Facility provides each ASC site with the latest logic diagrams and engineer's lists. However, individual logic diagram sets, and the associated engineer's list in Fiche form, may be obtained from them by specifying only the card name. Drawing numbers are not required. Engineer's lists, in Fiche form, for motherboards and harnesses may also be obtained by name. In addition, logic diagrams may be ordered by drawing number from Drawing Control at the TI Austin Facility.

APPENDIX A
DETAILS MAPS

Page

- Dedicated CM Locations. A-1
- 4XCP Details Map Overview A-2
- 4XCP Status Map and PSW Register. A-3
- 4XCP Details Locations. A-4
- 4X IPU Register Stack A-6
- 4X IPU Details Map. A-7
- 4X MBU Details Map. A-15
- 4X AU Details Map A-23
- 4X Hard Core Unit Register Maps:
  - Master Hard Core A-31
  - IPU Hard Core. A-32
  - MBU Hard Core. A-33
  - AU Hard Core A-33

LOCATION (HFX)	FUNCTION
00	Default bootstrap buffer start
07	MCW address cell CP0
08	MCW address cell CP1
09	MCW address cell CP2
0A	MCW address cell CP3
14	Pointer to store Status area (single CP)
15	Pointer to load Status area (single CP)
16	Pointer to store Intermediate area (X1CP)
17	Pointer to load Intermediate area (X1CP)
18	Pointer to store Details area (single CP)
19	Pointer to load Details area (single CP)
20 - 27	BRSM augmented PC save words
28	Pointer to MCU Context switch area CP0
2A	Pointer to MCU Context switch area CP1
2C	Pointer to MCU Context switch area CP2
2E	Pointer to MCU Context switch area CP3
38	Pointer to store MCU Map and Protect area
39	Pointer to load MCU Map and Protect area
40 - 47	ROM R0 augmented save words
48 - 4F	ROM R1 augmented save words
50 - 57	ROM R2 augmented save words
58 - 5F	ROM R3 augmented save words
60 - 67	ROM Base augmented save words
68 - 6F	ROM PC augmented save words
78 - 79	Pointer to TCC2(0) High Priority queue headers
7C - 7D	Pointer to TCC2(0) Low Priority queue headers
80 - 84	DISC 0 CA address
88 - 8C	DISC 1 CA address
90 - 94	DISC 2 CA address
98 - 9C	DISC 3 CA address
A0 - A1	Pointer to TCC2(2) High Priority queue headers
A4 - A5	Pointer to TCC2(2) Low Priority queue headers
A8 - A9	Pointer to TCC2(3) High Priority queue headers
AC - AD	Pointer to TCC2(3) Low Priority queue headers
B0 - B1	Pointer to TCC2(4) High Priority queue headers
B4 - B5	Pointer to TCC2(4) Low Priority queue headers
B8 - B9	Pointer to TCC2(5) High Priority queue headers
BC - BD	Pointer to TCC2(5) Low Priority queue headers
C0 - C1	Pointer to TCC2(6) High Priority queue headers
C4 - C5	Pointer to TCC2(6) Low Priority queue headers
C8 - C9	Pointer to TCC2(7) High Priority queue headers
CC - CD	Pointer to TCC2(7) Low Priority queue headers
DD - D1	Pointer to TCC2(8) High Priority queue headers
D4 - D5	Pointer to TCC2(8) Low Priority queue headers
D8 - D9	Pointer to TCC2(9) High Priority queue headers
DC - DD	Pointer to TCC2(9) Low Priority queue headers
E8	MEM EXT
F0 - F1	Tape SCB 0
F2 - F3	Tape SCB 1
F4 - F5	Tape SCB 2
F6 - F7	Tape SCB 3
100	ROM dump return address to caller
101	ROM dump final octet address
110	Automatic interrupt ROM exit pointer
111	Software interrupt ROM exit pointer
113	ROM card boot convert pointer
118 - 119	Pointer to TCC2(1) High Priority queue headers
11C - 11D	Pointer to TCC2(1) Low Priority queue headers
120	Pointer to User Exit area for CP0
121	Pointer to Monitor Entry area for CP0
122	Pointer to User Exit area for CP1
123	Pointer to Monitor Entry area for CP1
124	Pointer to User Exit area for CP2
125	Pointer to Monitor Entry area for CP2
126	Pointer to User Exit area for CP3
127	Pointer to Monitor Entry area for CP3
128	Pointer to Monitor Exit area for CP0
129	Pointer to User Entry area for CP0
12A	Pointer to Monitor Exit area for CP1
12B	Pointer to User Entry area for CP1
12C	Pointer to Monitor Exit area for CP2
12D	Pointer to User Entry area for CP2
12E	Pointer to Monitor Exit area for CP3
12F	Pointer to User Entry area for CP3
130 - 133	ACC 0 Start Ca address
134 - 137	ACC 1 Start CA address
138 - 13B	ACC 2 Start CA address
13C - 13F	ACC 3 Start CA address

(B)132488

Figure A-1. Dedicated CM Locations

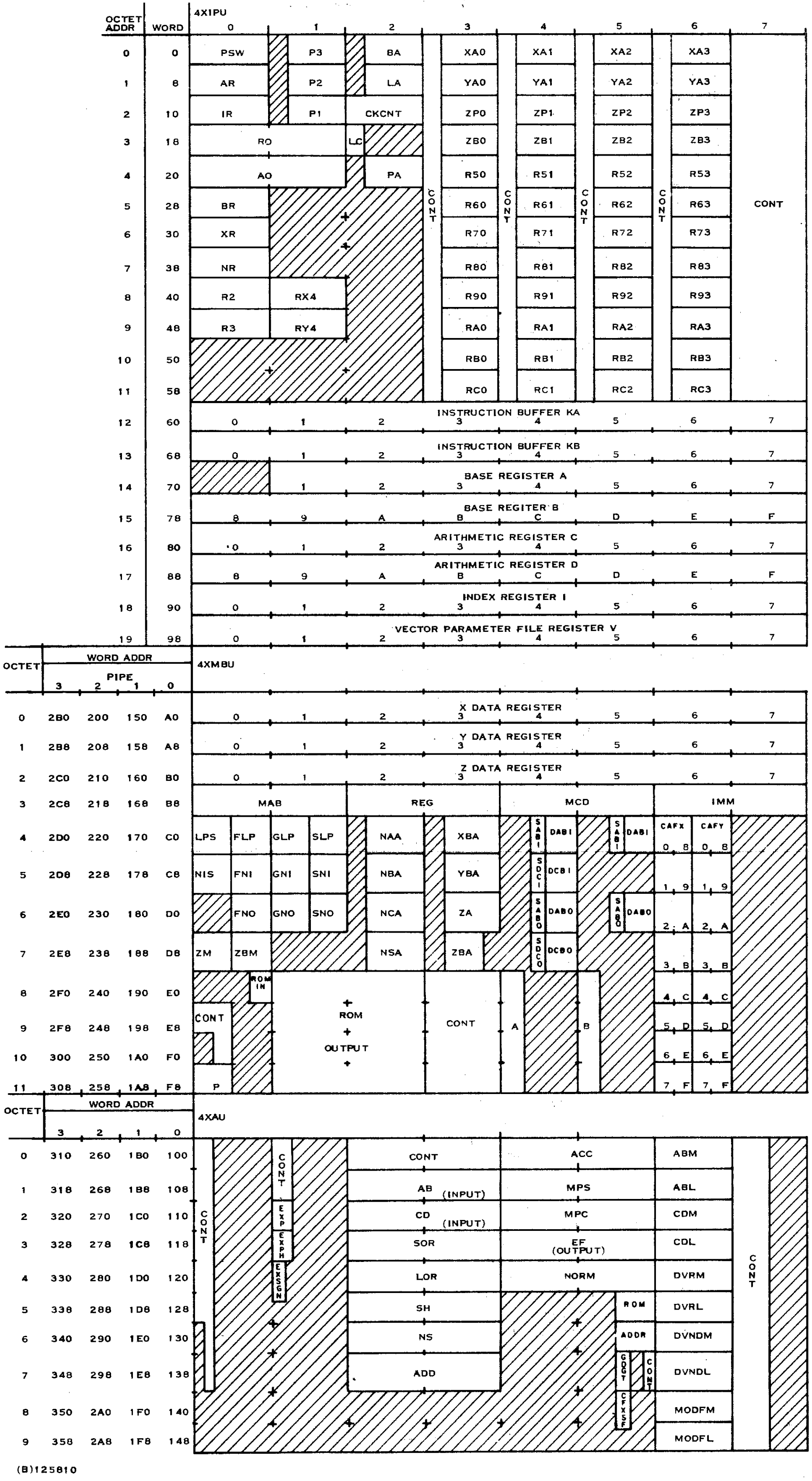
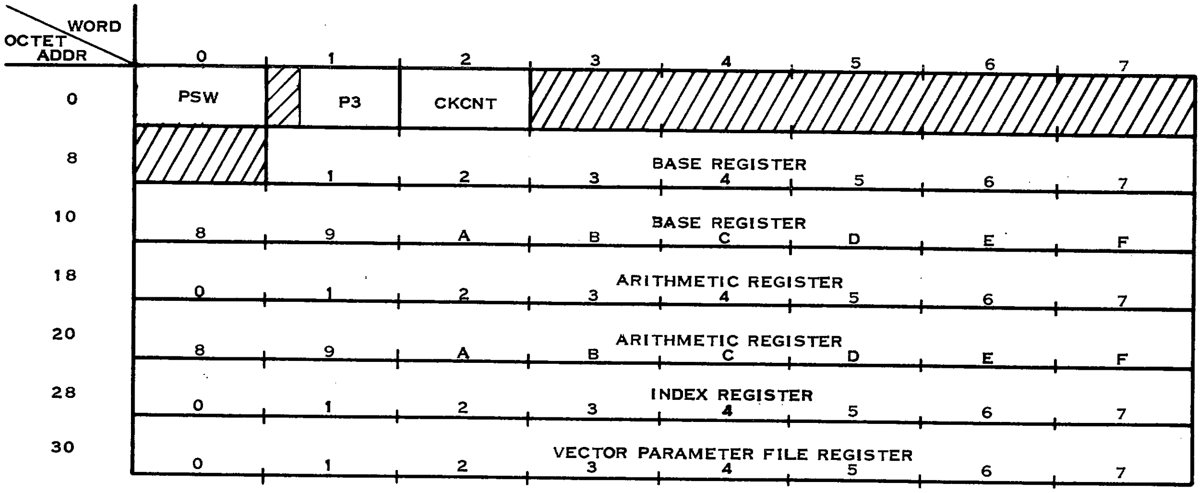
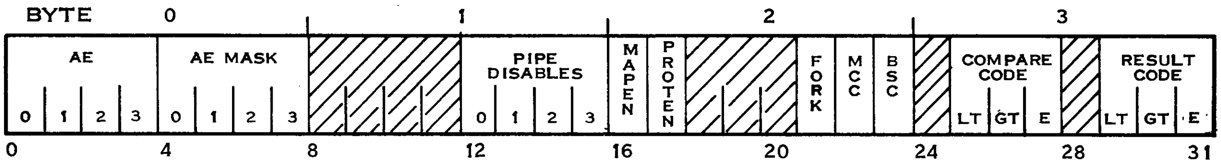


Figure A-2. 4XCP Details Map Overview



(B)125803

Figure A-3. 4XCP Status Map



ARITHMETIC EXCEPTION (AE) -
D - DIVIDE CHECK
1 - FX. PT. OVERFLOW
2 - FL. PT. OVERFLOW
3 - FL. PT. UNDERFLOW

PROTECT ENABLE (PROTEN) -
CM PROTECTED

MAP ENABLE (MAPEN) -
CM MAPPED

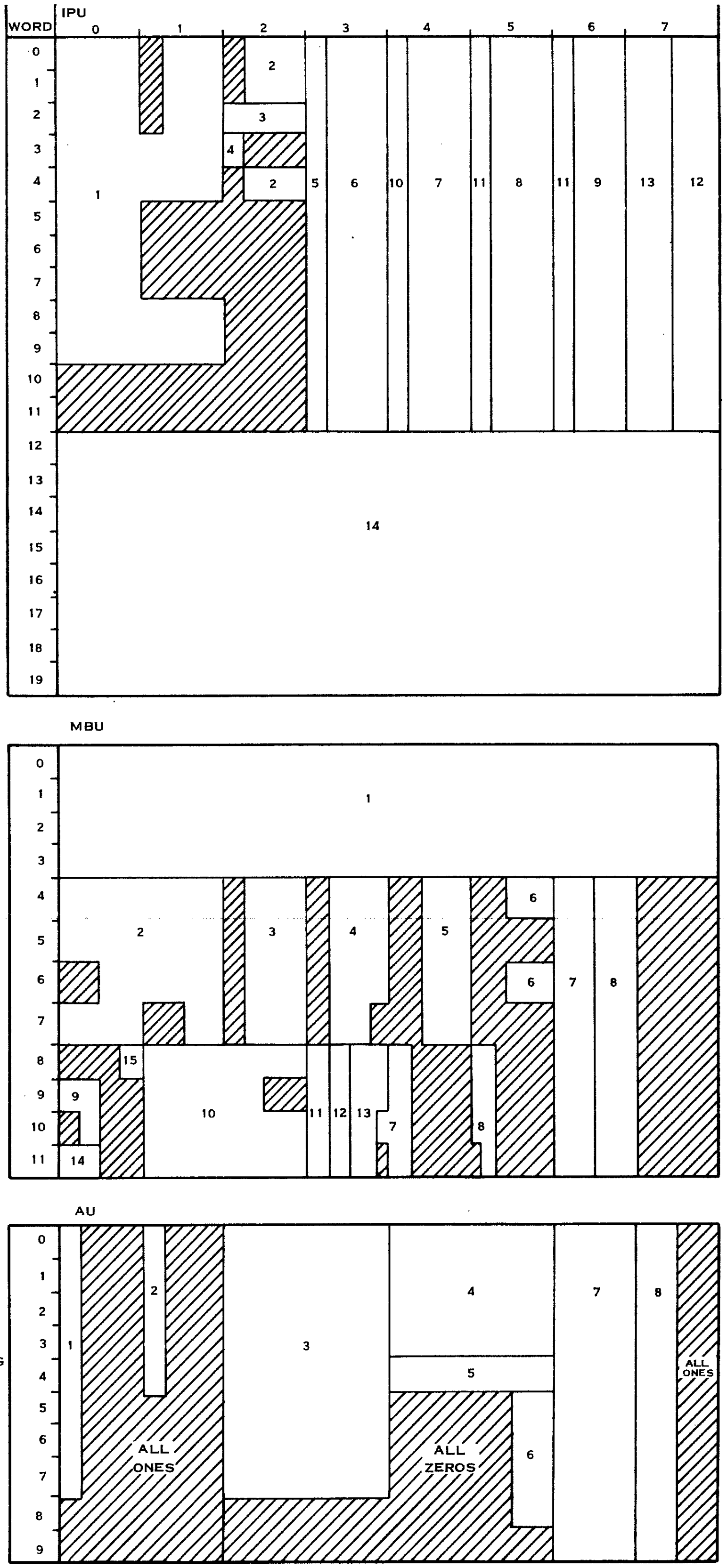
FORK MODE INDICATOR (FORK) -
CP IN FORK MODE

MONITOR CALL CONDITION (MCC) -
CP TRIED TO EXECUTE AN MCP/MCW

BRANCH OR SKIP CONDITION (BSC)
CP TRIED TO EXECUTE A BRANCH/SKIP FOR WHICH CONDITION WAS TRUE

(B)125809

Figure A-4. 4XCP Program Status Word (PSW)



NOTE: SEE SHEET 2 FOR BIT SLICING AND HARDWARE LOCATIONS OF THE NUMBERED AREAS.

(B)130856A(1 2)

Figure A-5. 4XCP Details Locations (Sheet 1 of 2)

4X DETAILS LOCATIONS

IPU DETAILS MAP

1.	I4PIPE(0–7):	4 BITS/CARD	10.	I4VECLAS:	BITS(0–2);
2.	I4ADDR(0–1):	12 BITS/CARDS		I4MISC:	BITS(3–5);
3.	I4ADDR(0–1):	16 BITS/CARD		I4CMREQ:	BIT(6); BIT(7) NOT USED
4.	I4ADDR(0–1):	4 BITS/CARD			
5.	I4RHAZ(0–3):	2 BITS/CARD	11.	I4INFACE(0–3):	1 BIT/HEX/CARD
6.	I4ZHAZ(0–3):	6 BITS/CARD	12.	I4INFACE(0–3):	4 BITS/CARD
7.	I4ZHAZ(4–7):	6 BITS/CARD	13.	I4STATUS:	BITS (0–3);
8.	I4ZHAZ(8–11):	6 BITS/CARD		I4CMREQ:	BITS (4–7);
9.	I4ZHAZ(12–15):	6 BITS/CARD		I4PIPTOP:	BITS (8–10);
				I4ROUTE3:	BIT (11);
				I4LVL3:	BITS (12–15)
			14.	I4FILE(0–15):	1 BIT/HALFWORD/ CARD

MBU DETAILS MAP

1.	BUDATA(0–15):	1 BIT/HALFWORD/ CARD	7.	BUCAFA(0):	COVERS 6L
			8.	BUCAFA(1):	COVERS 6R
2.	BULOOP(0–3):		9.	BUCTL3A	GATES OUT 0L, OCTETS 9–B
3.	BUADDRA(0–4):	5 BITS/CARD			
4.	BUCMR	1ST BIT	10.	BUROM(0–3):	1 HALFWORD/CARD
	BUADDR(0–4):	1 HEX/CARD	11.	BUCMR:	COVERS 3L, BITS 0–8
	BUZAG :	LAST HEX			
5.	BUCMR :	1ST 4 BITS;	12.	BUCTL4A:	COVERS 3L, BITS 9–15,
	BULOOP(0):	NEXT 7 BITS			
			13.	BUCTL3A:	COVERS 3R, BITS 0–14; BIT 15, SPARE
	BULOOP(1):	LAST 7 BITS			
6.	BUCMR:	1ST 4 BITS IN OCTETS 4 AND 6;	14.	BULOOP(0–3):	1 HEX/CARD
	BULOOP(2):	NEXT 7 BITS	15.	BUROM(0–3):	2 BITS/CARD
	BULOOP(3):	NEXT 7 BITS			

AU DETAILS MAP

1. AUCTL2B BITS (0–2); AUCTL3B BITS (3–7)
2. AUCTL5B BITS (0–7)
3. AU4ADDA(0–15): ONE HEX/CARD
4. AUOUTB(0–8): TWO HEX/CARD; ALSO FANS OUT AUNORM (0–7) AND AUCTL4A
5. AUNORMA(0–8): TWO HEX/CARD
6. AUCTL4A
7. AUMULTA(0–8): ONE HEX/CARD
8. AUROMFFB, BITS (0–12); AU4CLT1A, BITS (13–15)

(B)130856A (2/2)

Figure A-5. 4XCP Details Locations (Sheet 2 of 2)

X4 CP

REGISTER STACK FORMAT – I $\begin{Bmatrix}0\\1\\2\\3\end{Bmatrix}$ QR $\begin{Bmatrix}5\\6\\7\\8\\9\\A\\B\\C\end{Bmatrix}$ (8–31), IPU DETAILS MAP WORDS 3,4,5, AND 6

RSTACK BITS	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
DESCRIPTOR NAME	RA0	RA1	RA5	SCR	LAM	VHZ	RA2	RA3	ACT	OCK	CHZ	SP1 (UNUSED)	RA4	RA6	ZST	SCA	RHZ	LAC	DHW	DDW	RDT	SP2 (UNUSED)	AE4	SP3 (UNUSED)

REGISTER ADDRESS BITS – RA0–6
BITS 0–2 DECODED TO SELECT RF OCTET (A–V)
BITS 4–5 DECODED TO SELECT WORD (0–7) IN OCTET
BIT 6 SELECT LEFT (=0) OR RIGHT (=1) HW IN WORD

SCR – SHORT CIRCUIT REGISTER OPERAND
SCA – SHORT CIRCUIT ALPHA OPERAND
LAM – LOAD ARITHMETIC MASK
LAC – LOAD ARITHMETIC CONDITION
VHZ – VECTOR HAZARD (VPF BEING MODIFIED)
CHZ – COMPARE CODE HAZARD INSTRUCTION TYPE
RHZ – RESULT CODE HAZARD INSTRUCTION TYPE
AEH – ARITHMETIC EXCEPTION HAZARD INSTRUCTION TYPE
ACT – LEVEL 5-C ACTIVITY BIT
OCK – ONE CLOCK INSTRUCTION TYPE
ZST – Z STORE (CM DESTINATION) INSTRUCTION TYPE
DHW – DESTINATION HALFWORD
DDW – DESTINATION DOUBLEWORD
RDT – REGISTER FILE DESTINATION INSTRUCTION TYPE

SP1
SP2 } SPARE (UNUSED) RSTACK BITS
SP3

(B)131560

Figure A-6. 4X IPU Register Stack Format

IPU 4 DETAILS MAP
WORD 0

OCTET \ BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	IRQAE(0)	IRQAE(1)	IRQAE(2)	IRQAE(3)	IRQAEM(0)	IRQAEM(1)	IRQAEM(2)	IRQAEM(3)	NOT USED	NOT USED	NOT USED	NOT USED	PIPE DISABLE (1)	PIPE DISABLE (2)	PIPE DISABLE (3)	PIPE DISABLE (4)	IRQ MAPEN	IRQ PROEN	NOT USED	NOT USED	NOT USED	IRQFORK	IRQMCC	IRQBSC	NOT USED	IRQCL	IRQCG	IRQCE	NOT USED	IRQRL	IRQRG	IRQRE
1	IRQAR (0-31)																															
2	IIQIR (0-31)																															
3	IBQRO0 (0-31)																															
4	IBQAO0 (0-31)																															
5	IPQBR (0-31)																															
6	IPQXR (0-31)																															
7	IPQNR (0-31)																															
8	NOT USED			R2(0)	NOT USED			R2(1)	NOT USED			R2(2)	NOT USED			R2(3)	NOT USED			R2(4)	NOT USED			R2(5)	NOT USED			R2(6)	NOT USED			R2(7)
9	NOT USED			R3(0)	NOT USED			R3(1)	NOT USED			R3(2)	NOT USED			R3(3)	NOT USED			R3(4)	NOT USED			R3(5)	NOT USED			R3(6)	NOT USED			R3(7)
10	NOT USED																															
11	NOT USED																															
12	ILQKA0 (0-31)																															
13	ILQKB0 (0-31)																															
14	ALL ZEROS																															
15	IFQB0 (0-31)																															
16	IFQC0 (0-31)																															
17	IFQD0 (0-31)																															
18	IFQI0 (0-31)																															
19	IFQV0 (0-31)																															

IPU4 DETAILS MAP
WORD 1

OCTET \ BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	NOT USED								IRQP3 (8-31)																							
1	NOT USED								IPQP2 (8-31)																							
2	NOT USED								IIQP1 (8-31)																							
3	IBQRO1 (0-31)																															
4	IBQAO1 (0-31)																															
5	NOT USED																															
6	NOT USED																															
7	NOT USED																															
8	NOT USED			RX4(0)	NOT USED			RX4(1)	NOT USED			RX4(2)	NOT USED			RX4(3)	NOT USED			RX4(4)	NOT USED			RX4(5)	NOT USED			RX4(6)	NOT USED			RX4(7)
9	NOT USED			RY4(0)	NOT USED			RY4(1)	NOT USED			RY4(2)	NOT USED			RY4(3)	NOT USED			RY4(4)	NOT USED			RY4(5)	NOT USED			RY4(6)	NOT USED			RY4(7)
10	NOT USED																															
11	NOT USED																															
12	ILQKA1 (0-31)																															
13	ILQKB1 (0-31)																															
14	IFQA1 (0-31)																															
15	IFQB1 (0-31)																															
16	IFQC1 (0-31)																															
17	IFQD1 (0-31)																															
18	IFQI1 (0-31)																															
19	IFQV1 (0-31)																															

(D) 123682

Figure A-7. 4X IPU Details Map
(Sheet 1 of 4)

IPU4 DETAILS MAP
WORD 2

OCTET \ BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	NOT USED							ILQBA (8-31)																								
1	NOT USED							ILQLA (8-31)																								
2	CKCNTR (0-31)																															
3	ICQLC (0-7)							NOT USED																								
4	NOT USED							IIQPA (8-31)																								
5	NOT USED																															
6	NOT USED																															
7	NOT USED																															
8	NOT USED																															
9	NOT USED																															
10	NOT USED																															
11	NOT USED																															
12	ILQKA2 (0-31)																															
13	ILQKB2 (0-31)																															
14	IFQA2 (0-31)																															
15	IFQB2 (0-31)																															
16	IFQC2 (0-31)																															
17	IFQD2 (0-31)																															
18	IFQI2 (0-31)																															
19	IFQV2 (0-31)																															

IPU DETAILS MAP
WORD 3

OCTET \ BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	LD0ACT	R4ACT	LD0RA3	R4RA3	LD0RA4	R4RA4	LD0RA5	R4RA5	XA0(8-31)																							
1	LD0DHW	R4DHW	LD0RA2	R4RA2	LD0RA0	R4RA6	NOT USED	R4RHZ	YA0 (8-31)																							
2	LD0DDW	R4DDW	LD0RA6	R4RA6	LD0RA1	R4RA1	NOT USED	R4LAC	ZP0 (8-31)																							
3	LD1ACT	R4LAM	LD1RA3	R4CHZ	LD1RA4	R4OCK	LD1RA5	R4AEH	ZB0 8-31)																							
4	LD1DHW	NOT USED	LD1RA2	R4VHZ	LD1RA0	R4SCR	NOT USED	R4RDT	R50 (8-31)																							
5	LD1DDW	NOT USED	LD1RA6	R4ZST	LD1RA1	R4SCA	NOT USED	NOT USED	R60 (8-31)																							
6	LD2ACT	NOT USED	LD2RA3	NOT USED	LD2RA4	NOT USED	LD2RA5	NOT USED	R70 (8-31)																							
7	LD2DHW	NOT USED	LD2RA2	NOT USED	LD2RA0	NOT USED	NOT USED	NOT USED	R80 (8-31)																							
8	LD2DDW	NOT USED	LD2RA6	NOT USED	LD2RA1	NOT USED	NOT USED	NOT USED	R90 (8-31)																							
9	LD3ACT	NOT USED	LD3RA3	NOT USED	LD3RA4	NOT USED	LD3RA5	NOT USED	RA0 (8-31)																							
10	LD3DHW	NOT USED	LD3RA2	NOT USED	LD3RA0	NOT USED	NOT USED	NOT USED	RB0 (8-31)																							
11	LD3DDW	NOT USED	LD3RA6	NOT USED	LD3RA1	NOT USED	NOT USED	NOT USED	RC0 (8-31)																							
12	ILQKA3 (0-31)																															
13	ILQKB3 (0-31)																															
14	IFQA3 (0-31)																															
15	IFQB3 (0-31)																															
16	IFQC3 (0-31)																															
17	IFQD3 (0-31)																															
18	IFQI3 (0-31)																															
19	IFQV3 (0-31)																															

(D) 123683

Figure A-7. 4X IPU Details Map
(Sheet 2 of 4)

IPU4 DETAILS MAP
WORD 4

OCTET \ BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	VIP0	VISTR0	ARINC	ZJOIN0	Z0AGE1	BURHW	WNDER0	NOT USED	XA1(8-31) →																							
1	VIP1	VISTR1	ARHAZ	ZJOIN1	Z0AGE2	BURSW	WNDER1	NOT USED	YA1 (8-31) →																							
2	VIP2	VISTR2	RCTR0	ZJOIN2	Z0AGE3	BURDW	WNDER2	NOT USED	ZP1 (8-31) →																							
3	VIP3	VISTR3	RCTR1	ZJOIN3	Z1AGE0	BUAHW	WNDER3	NOT USED	ZB1 (8-31) →																							
4	SELPI0	VBAD0	RCTR2	RJOIN0	ZIAGE2	BUASW	WNDLT0	NOT USED	R51 (8-31) →																							
5	SELPI1	VBAD1	JOIN	RJOIN1	ZIAGE3	BUADW	WNDLT1	NOT USED	R61 (8-31) →																							
6	SELPI2	VBAD2	NOT USED	RJOIN2	Z2AGE0	VHW	WNDLT2	NOT USED	R71 (8-31) →																							
7	SELPI3	VBAD3	NOT USED	RJOIN3	Z2AGE1	VSW	WNDLT3	NOT USED	R81 (8-31) →																							
8	4GETI0	NOT USED	NOT USED	L4RJN	Z2AGE3	VDW	NOT USED	NOT USED	R91 (8-31) →																							
9	4GETI1	NOT USED	NOT USED	NOT USED	Z3AGE0	NOT USED	NOT USED	NOT USED	RA1 (8-31) →																							
10	4GETI2	NOT USED	NOT USED	NOT USED	Z3AGE1	NOT USED	NOT USED	NOT USED	RB1 (8 31) →																							
11	4GETI3	NOT USED	NOT USED	NOT USED	Z3AGE2	NOT USED	NOT USED	NOT USED	RC1 (8-31) →																							
12	ILQKA4 (0-31) →																															
13	ILQKB4 (0-31) →																															
14	IFQA4 (0-31) →																															
15	IFQB4 (0-31) →																															
16	IFQC4 (0-31) →																															
17	IFQD4 (0-31) →																															
18	IFQI4 (0-31) →																															
19	IFQV4 (0-31) →																															

IPU4 DETAILS MAP
WORD 5

OCTET \ BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	MBIAC0	MBIAC1	MBIAC2	MBIAC3	GP1L0	GP1L1	GPIL2	GP1L3	XA2 (8-31) →																							
1	DPMBI0	DPMBI1	DPMBI2	DPMBI3	GP2L0	GP2L1	GP2L2	GP2L3	YA2 (8-31) →																							
2	DPMBO0	DPMBO1	DPMBO2	DPMBO3	GP3L0	GP3L1	GP3L2	GP3L3	ZP2 (8-31) →																							
3	XAACT0	XAACT1	XAACT2	XAACT3	GP4L0	GP4L1	GP4L2	GP4L3	ZB2 (8-31) →																							
4	YAACT0	YAACT1	YAACT2	YAACT3	SMGP40	SMGP41	SMGP42	SMGP43	R52 (8-31) →																							
5	CUEQX0	CUEQX1	CUEQX2	CUEQX3	SMGP50	SMGP51	SMGP52	SMGP53	R62 (8-31) →																							
6	CUEQY0	CUEQY1	CUEQY2	CUEQY3	OCK40	OCK41	OCK42	OCK43	R72 (8-31) →																							
7	XAFUL0	XAFUL1	XAFUL2	XAFUL3	OCK50	OCK51	OCK52	OCK53	R82 (8-31) →																							
8	YAFUL0	YAFUL1	YAFUL2	YAFUL3	FCSGT0	FCSGT1	FCSGT2	FCSGT3	R92 (8-31) →																							
9	SCKT40	SCKT41	SCKT42	SCKT43	SLNXT0	SLNXT1	SLNXT2	SLNXT3	RA2 (8-31) →																							
10	SCKT50	SCKT51	SCKT52	SCKT53	PPMF0	PPMF1	PPMF2	PPMF3	RB2 (8-31) →																							
11	SCKT60	SCKT61	SCKT62	SCKT63	PACAO0	PACAO1	PACAO2	PACAO3	RC2 (8-31) →																							
12	ILQKA5 (0-31) →																															
13	ILQKB5 (0-31) →																															
14	IFQA5 (0-31) →																															
15	IFQB5 (0-31) →																															
16	IFQC5 (0-31) →																															
17	IFQD5 (0-31) →																															
18	IFQI5 (0-31) →																															
19	IFQV5 (0-31) →																															

(D) 123684

Figure A-7. 4X IPU Details Map
(Sheet 3 of 4)

IPU4 DETAILS MAP
WORD 6

BIT OCTET	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	ENAB0	ENAB1	ENAB2	ENAB3	QIRT0	QIRT1	QIRT2	QIRT3	XA3 (8-31)																							
1	ROM80	ROM81	ROM82	ROM83	MODE0	MODE1	MODE2	MODE3	YA3 (8-31)																							
2	ROM90	ROM91	ROM92	ROM93	LDXA0	LDXA1	LDXA2	LDXA3	ZP3 (8-31)																							
3	ROMA0	ROMA1	ROMA2	ROMA3	LDXBA0	LDXBA1	LDXBA2	LDXBA3	ZB3 (8-31)																							
4	ROMB0	ROMB1	ROMB2	ROMB3	LDYA0	LDYA1	LDYA2	LDYA3	R53 (8-31)																							
5	ROMC0	ROMC1	ROMC2	ROMC3	LDYBA0	LDYBA1	LDYBA2	LDYBA3	R63 (8-31)																							
6	REQFW0	REQFW1	REQFW2	REQFW3	XUP0	XUP1	XUP2	XUP3	R73 (8-31)																							
7	FWWT0	FWWT1	FWWT2	FWWT3	YUP0	YUP1	YUP2	YUP3	R83 (8-31)																							
8	ZPFUL0	ZPFUL1	ZPFUL2	ZPFUL3	ZTXU0	ZTXU1	ZTXU2	ZTXU3	R93 (8-31)																							
9	ZBFUL0	ZBFUL1	ZBFUL2	ZBFUL3	ZTYU0	ZTYU1	ZTYU2	ZTYU3	RA3 (8-31)																							
10	ZEX0	ZEX1	ZEX2	ZEX3	REGDP0	REGDP1	REGDP2	REGDP3	RB3 (8-31)																							
11	ZEY0	ZEY1	ZEY2	ZEY3	IMM0	IMM1	IMM2	IMM3	RC3 (8-31)																							
12	ILQKA6 (C-31)																															
13	ILQKB6 (0-31)																															
14	IFQA6 (0-31)																															
15	IFQB6 (0-31)																															
16	IFQC6 (0-31)																															
17	IFQD6 (0-31)																															
18	IFQI6 (0-31)																															
19	IFQV6 (0-31)																															

IPU4 DETAILS MAP
WORD 7

BIT OCTET	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	CCRT0	CCRT1	CCRT2	CCRT3	NOT USED	NOT USED	VAC0	VAC1	CCTR0	NOT USED	L1ACT	YNEXT0	L3ACT	L3PPL	L3CHK	L3PWT	PRMRA0	PRMRA1	PRMRA6	PRMBR	PRMIDW	PRMAEH	PRMEXN	PRMEXM	PRMAU4	PRMAU5	PRMAU6	PRMAU7	PRMGP1	PRMGP2	PRMGP3	PRMGP4
1	RCRT0	RCRT1	RCRT2	RCRT3	NOT USED	LAORD	DUAL	TFAIL	CCTR1	IIQS01	PRV1	YNEXT1	RIHIB	L3BWN	L3IWT	L3IHZ	PRMDHW	PRMDDW	PRMIHW	PRMISW	PRMM	PRMRHZ	PRMCAR	PRMCHZ	PRMRDT	PRMRSE	PRMHHW	PRMSDW	PRMMDV	PRMIOP	NOT USED	NOT USED
2	ADDRM4	ADDRM5	ADDRM6	ADDRM7	PBVLL	FLGFL	FLG12	FLG4	CCTR2	IIQS02	FRHAZ1	YNEXT2	XEC	HOLD	BRDN	OPDN	C3RA0	C3RA1	C3RA6	C3BR	C3IDW	C3AEH	C3EXN	C3FXM	C3AU4	C3AU5	C3AU6	C3AU7	C3GP1	C3GP2	C3GP3	C3GP4
3	T10	T11	T12	T13	QIP0	QIP1	QOP0	QOP1	L2ACT	IIQS03	TARGT1	YNEXT3	NOT USED	NOT USED	NOT USED	NOT USED	C3DHW	C3DDW	C3IHW	C3ISW	C3M	C3RHZ	C3CAR	C3CHZ	C3RDT	C3RSE	C3HHW	C3SDW	C3MDV	C3IOP	NOT USED	NOT USED
4	T20	T21	T22	T23	QBSY0*	QBSY1*	QBSY2*	QBSY3*	IND2	IIQS04	NOT USED	ZMAL10	FRHAZ3	ILOP	PRV3	TARGT3	RRMAHW	RRMADW	RRMSBL	RRMSGN	RRMBRH	RRMPNK	RRMBLU	RRMIDN	RRMSHW	NOT USED	RRMNSP	NOT USED	RRMSTR	RRMNSN	NOT USED	NOT USED
5	AROT26	AROT27	AROT28	AROT29	PRV0	ACT0*	CUE00	CUE10	FRHAZ2	IIQS05	NOT USED	ZMAL11	**CRSLT**	SKIND	TRMIN	POIND	RRMSPK	**RRMSYL**	RRMSOR	RRMGRY	RRMIDZ	RRMGRN	NOT USED	NOT USED	RRMSGT	RRMOCK	RRMRGS	NOT USED	NOT USED	NOT USED	NOT USED	NOT USED
6	AROT30	AROT31	AROT32	AR32	PRV1	ACT1*	CUE01	CUE11	TARGT2	IIQS06	NOT USED	ZMAL12	HCALI	PMOFF	MEQO	IND3	PDCMHF	NOT USED	NOT USED	NOT USED	PDCTHF	NOT USED										
7	M10	M11	M12	M13	PRV2	ACT2*	CUE02	CUE12	PRV2	IIQS07	NOT USED	ZMAL13	NOT USED	NOT USED	NOT USED	HDW	PDCLF	PDCNOP	NOT USED	NOT USED	PDCORG	PDCPB	NOT USED	NOT USED	PDCPP	PDCVCT	NOT USED	NOT USED	PDCXEC	NOT USED	NOT USED	NOT USED
8	NOT USED	NOT USED	NOT USED	NOT USED	PRV3	ACT3*	CUE03	CUE13	M20	IIQS08	NOT USED	PIRT0	L3VIN	L3FLW	L3VPI	L3VBR	RDCLF	RDCNOP	NOT USED	NOT USED	RDCORG	RDCPB	NOT USED	NOT USED	RDCPP	RDCVCT	NOT USED	NOT USED	RDCXEC	NOT USED	NOT USED	NOT USED
9	NOT USED	NOT USED	NOT USED	NOT USED	KRTAG	KAFUL	KAPRV	KAHAZ	M21	IIQS09	NOT USED	PIRT1	L3NIW	L3VGO	L3ORW	L3ORM	RDCBAE	RDCBBX	RDCBCC	RDCBCG	RDCBWN	RDCBXC	RDCFRK	RDCIN	RDCLLA	RDCMCP	RDCMLT	RDCPSH	RDCSTK	RDCXCH	RDCSTC	NOT USED
10	NOT USED	NOT USED	NOT USED	NOT USED	NOT USED	KBFUL	KBPRV	KBHAZ	M22	IIQS10	NOT USED	PIRT2	NOT USED	NOT USED	NOT USED	NOT USED	RDCBCL	RDCBRC	GAOSC0	NOT USED	RDCLAC	RDCLAM	GAOSC1	NOT USED	**RDCSKE**	RDCSPS	GAOSC2	NOT USED			GAOSC3	NOT USED
11	NOT USED	NOT USED	NOT USED	NOT USED	AREX *	IPPAE*	IPIOP*	IPPRV*	M23	STATE	NOT USED	PIRT3	NOT USED	NOT USED	NOT USED	NOT USED	AUIAC0	L7ACT0	DAVWT0	**NOT USED**	AUIAC1	L7ACT1	DAVWT1	**NOT USED**	AUIAC2	L7ACT2	DAVWT2	**NOT USED**	**AUIAC3**	L7ACT3	DAVWT3	**NOT USED**
12	ILQKA7 (0-31)																															
13	ILQKB7 (0-31)																															
14	IFQA7 (0-31)																															
15	IFQB7 (0-31)																															
16	IFQC7 (0-31)																															
17	IFQD7 (0-31)																															
18	IFQI7 (0-31)																															
19	IFQV7 (0-31)																															

(D) 123900

*NOTE DO NOT LOAD ACT0, ACT1, ACT2, ACT3, QBSY0, QBSY1, QBSY2, QBSY3, AREX, IPPAE, IPIOP, IPPRV

Figure A-7. 4X IPU Details Map
(Sheet 4 of 4)

LEFT HALFWORD 0

OCTET	BIT															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	BRQX0L(0-15)															
1	BRQY0L(0-15)															
2	BRQZ0L(0-15)															
3	BIQMAL(0-15)															
4	BLQLPS(0-15)															
5	BLQNIS(0-15)															
6	NOT USED															
7	BAQZM(0-15)															
8	NOT USED															
9	BCQ STAT(0)	BCQ STAT(1)	BCQ STAT(2)	BCQ STAT(3)	BCQ ACCT(0)	ACCT(1)	ACCT(2)	ACCT(3)	BCQ SV(0)	SV(1)	SV(2)	SV(3)	BCQ HS(0)	BCQ AVCAC(0)	BCQ AVCAC(1)	BCQ AVCAC(2)
10	NOT USED												BCQ VIS(0)	BCQ AAUAC(0)	BCQ AAUAC(1)	BCQ VCNOP
11	BFQP(0-15)															

RIGHT HALFWORD 0

OCTET	BIT															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	BRQX0R(0-15)															
1	BRQY0R(0-15)															
2	BRQZ0R(0-15)															
3	BIQMAR(0-15)															
4	BLQFLP(0-15)															
5	BLQFNI(0-15)															
6	BLQFNO(0-15)															
7	BAQZBM(0-15)															
8	NOT USED							BMQROMIS (0)	BMQROMIN(1-8)							
9	NOT USED															
10	NOT USED															
11	NOT USED															

LEFT HALFWORD 1

OCTET	BIT															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	BRQX1L(0-15)															
1	BRQY1L(0-15)															
2	BRQZ1L(0-15)															
3	BIQMBL(0-15)															
4	BLQGLP(0-15)															
5	BLQGNI(0-15)															
6	BLQGNO(0-15)															
7	NOT USED															
8	BMQROMOT(0-15)															
9	BMQROMOT(16-31)															
10	BMQROMOT(32-42)											NOT USED		BMQROMOT (45-47)		
11	NOT USED	BMQROMOT (49-53)					NOT USED					BMQROMOT (59-63)				

RIGHT HALFWORD 1

OCTET	BIT															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	BRQX1R(0-15)															
1	BRQY1R(0-15)															
2	BRQZ1R(0-15)															
3	BIQMBR(0-15)															
4	BLQSLP(0-15)															
5	BLQSNI(0-15)															
6	BLQSNO(0-15)															
7	BHQSPCFN(0-15)															
	BHSECFX	BHSECFXB	BHSECFXH	BHSECFY	BHSECFYB	BHSECFYH	BHSECFZ	BHSECFZB	**BHSECIN**	BHSECO	BHSECA	BHSECF	BHSECC	BHSECL	BHSECRI	BHSECRO
8	BMQROMOT (64-66)			NOT USED	BMQROMOT(68-79)											
9	BMQROMOT(80-95)															
10	BMQROMOT(96-111)															
11	BMQROMOT(112-118)							NOT USED				BMQROMOT(123-127)				

(D)127613 (1/4)

Figure A-8. 4X MBU Details Map
(Sheet 1 of 4)

LEFT HALFWORD 2

OCTET	BIT 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	BRQX2L(0-15)															
1	BRQY2L(0-15)															
2	BRQZ2L(0-15)															
3	BIQREG0L(0-15)															
4	NOT USED							BAQNAA(8-16)								
5	NOT USED							BAQNBA(8-16)								
6	NOT USED							BAQNCA(8-16)								
7	NOT USED							BAQNSA(8-16)								
8	BMQROMOT(128-143)															
9	BMQROMOT(144-159)															
10	BMQROMOT(160-173)														NOT USED	
11	BMQROMOT (176-178)			NOT USED					BMQROMOT 184	NOT USED		BMQROMOT (187-191)				

RIGHT HALFWORD 2

OCTET	BIT 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	BRQX2R(0-15)															
1	**BRQY2R**(0-15)															
2	BRQZ2R(0-15)															
3	BIQREG0R(0-15)															
4	BAQNAA(17-32)															
5	BAQNBA(17-32)															
6	BAQNCA(17-32)															
7	BAQNSA(17-32)															
8	BMQROMOT(192-206)															NOT USED
9	NOT USED															
10	BMQROMOT(224-239)															
11	BMQROMOT(240-249)										NOT USED	BMQROMOT(251-255)				

LEFT HALFWORD 3

OCTET	BIT 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	BRQX3L(0-15)															
1	BRQY3L(0-15)															
2	BRQZ3L(0-15)															
3	BIQREG1L(0-15)															
4	NOT USED								BAQXBA(8-16)							
5	NOT USED								BAQYBA(8-16)							
6	NOT USED								BAQZA(8-16)							
7	NOT USED								BAQZBA(8-16)							
8	BCQ CUEP0 (0)	BCQ CUEP0 (1)	BCQ CUE0 (0)	BCQ CUE0 (1)	BCQ CUE4 (0)	BCQ CUE4 (1)	BCQ **ZBREQ**	BCQ **XBREQ** (0)	BCQ **XBREQ** (1)	BCQ NSFUL	BCQ HWC	BCQ DWC	BCQ SWC	BCQ IMFUL	BCQ MBIAC	BCQ SRMST
9	BCQ CUEP1 (0)	BCQ CUEP1 (1)	BCQ CUE1 (0)	BCQ CUE1 (1)	BCQ CUE5 (0)	BCQ CUE5 (1)	BCQ ZBBSY	BCQ YNEXT	NOT USED	BCQ ZAFUL	BCQ HWZ	BCQ DWZ	BCQ VIC	BCQ RGFUL	BCQ DPMBI	BCQ **PVCAO**
10	BCQ CUEP2 (0)	BCQ CUEP2 (1)	BCQ CUE2 (0)	BCQ CUE2 (1)	BCQ CUE6 (0)	BCQ CUE6 (1)	BCQ IZBSY	BCQ ZFILN	NOT USED	BCQ ZFUL	BCQ SLPE1	BCQ SNIE1	BCQ SNOE1	BCQ YFRST	BCQ DPMBO	BCQ LSENC
11	BCQ PRVLT	BCQ PARER	BCQ CUE3 (0)	BCQ CUE3 (1)	BCQ CUE7 (0)	BCQ CUE7 (1)	BCQ ZWAIT	BCQ **VZAZB**	BCQ **VLSTW**	BCQ CVEND	BCQ SLPE0	BCQ SNIE0	BCQ SNOE0	BCQ **VORST**	BCQ AGRTR	BCQ TOGLE

RIGHT HALFWORD 3

OCTET	BIT 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	BRQX3R(0-15)															
1	BRQY3R(0-15)															
2	BRQZ3R(0-15)															
3	BIQREG1R(0-15)															
4	BAQXBA(17-28)												BAQXA(29-32)			
5	BAQYBA(17-28)												BAQYA(29-32)			
6	BAQZA(17-28)												BAQZEA(29-32)			
7	BAQZBA(17-28)												NOT USED			
8	BCQ PCAUR	BCQ GATAR	BCQ AURAC	BCQ SCKTS	BCQ NOPRI	BCQ CCKRI	BCQ ESLMO	BCQ AUL08 (0)	BCQ AUL08 (1)	BCQ RMOUT (0)	(4)	(8)	(12)	(16)	BCQ RMOUT (20)	NOT USED
9	NOT USED	BCQ GATAI	BCQ ACT12	BCQ SCR6	BCQ SMGRP	BCQ SGTRI	BCQ ESLAR	BCQ AUL09 (0)	BCQ AUL09 (1)	BCQ RMOUT (1)	(5)	(9)	(13)	(17)	BCQ RMOUT (21)	NOT USED
10	BCQ PCAUO	BCQ GATAO	BCQ DPAUO	BCQ RODO6	BCQ OCRMA	NOT USED	BCQ ESLAO	BCQ AUL10 (0)	BCQ AUL10 (1)	BCQ RMOUT (2)	(6)	(10)	(14)	BCQ RMOUT (18)	NOT USED	NOT USED
11	BCQ SLNXT	NOT USED	**NOT USED**	BCQ AREX	NOT USED	BCQ VRMGT	**NOT USED**	BCQ AUL11 (0)	BCQ AUL11 (1)	BCQ RMOUT (3)	(7)	(11)	(15)	BCQ RMOUT (19)	NOT USED	NOT USED

(D)127613 (2/4)

Figure A-8. 4X MBU Details Map
(Sheet 2 of 4)

LEFT HALFWORD 4

OCTET	BIT															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	BRQX4L(0-15)															
1	BRQY4L(0-15)															
2	BRQZ4L(0-15)															
3	BIQMCL(0-15)															
4	NOT USED														BDQSABI(0-1)	
5	NOT USED														BDQSDCI(0-1)	
6	NOT USED														BDQSABO(0-1)	
7	NOT USED														BDQSDCO(0-1)	
8	BCQDWA (0)	BCQVIA (0)	BLQFLPE0 (0)	BLQFLPE1 (0)	BCQFSTA0 (0)	BCQXFUL (0)	NOT USED									
9	BCQSWA (0)	BCQAVSV (0)	BLQFNIE0 (0)	BLQFNIE1 (0)	BCQFSTA1 (0)	BCQXHFUL (0)	NOT USED									
10	BCQHWA (0)	NOT USED	BLQFNOE0 (0)	BLQFNOE1 (0)	BCQXAENL (0)	BCQXBFUL (0)	NOT USED									
11	NOT USED				BCQXALSE (0)	BCQXAFUL (0)	NOT USED									

RIGHT HALFWORD 4

OCTET	BIT															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	BRQX4R(0-15)															
1	BRQY4R(0-15)															
2	BRQZ4R(0-15)															
3	BIQMCR(0-15)															
4	BDQSABI(2-3)		BDQDABI(1-7)							BDQDABI(11-17)						
5	BDQSDCI(2-3)		BDQDCI(1-7)							BDQDCI(11-17)						
6	BDQSABO(2-3)		BDQDABO(1-7)							BDQDABO(11-17)						
7	BDQSDCO(2-3)		BDQDCO(1-7)							BDQDCO(11-17)						
8	NOT USED															
9	NOT USED															
10	NOT USED															
11	NOT USED															

LEFT HALFWORD 5

OCTET	BIT															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	BRQX5L(0-15)															
1	BRQY5L(0-15)															
2	BRQZ5L(0-15)															
3	BIQMDL(0-15)															
4	NOT USED														BDQSABI(4-5)	
5	NOT USED															
6	NOT USED														BDQSABO(4-5)	
7	NOT USED															
8	BCQDWA (1)	BCQVIA (1)	BLQFLPE0 (1)	BLQFLPE1 (1)	BCQFSTA0 (1)	BCQXFUL (1)	NOT USED									
9	BCQSWA (1)	BCQAVSV (1)	BLQFNIE0 (1)	BLQFNIE1 (1)	BCQFSTA1 (1)	BCQXHFUL (1)	NOT USED									
10	BCQHWA (1)	NOT USED	BLQFNOE0 (1)	BLQFNOE1 (1)	BCQXAENL (1)	BCQXBFUL (1)	NOT USED									
11	NOT USED				BCQXALSE (1)	BCQXAFUL (1)	NOT USED									

RIGHT HALFWORD 5

OCTET	BIT															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	BRQX5R(0-15)															
1	BRQY5R(0-15)															
2	BRQZ5R(0-15)															
3	BIQMDR(0-15)															
4	BDQSABI(6-7)		BDQDABI(21-27)							BDQDABI(31-37)						
5	NOT USED															
6	BDQSABO(6-7)		BDQDABO(21-27)							BDQDABO(31-37)						
7	NOT USED															
8	NOT USED															
9	NOT USED															
10	NOT USED															
11	NOT USED															

(D)127613 (3/4)

Figure A-8. 4X MBU Details Map
(Sheet 3 of 4)

LEFT HALFWORD 6

OCTET	BIT															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	BRQX6L(0-15)															
1	BRQY6L(0-15)															
2	BRQZ6L(0-15)															
3	BIQIMM0L(0-15)															
4	BFQCAF0 (0)	BFQCAF1 (0)	BFQCAF2 (0)	BFQCAF3 (0)	BFQACTV (0)	BFQNEWO (0)	BFQENDL (0)	NOT USED	BFQCAF0 (8)	BFQCAF1 (8)	BFQCAF2 (8)	BFQCAF3 (8)	BFQACTV (8)	BFQNEWO (8)	BFOENDL (8)	NOT USED
5	BFQCAF0 (1)	BFQCAF1 (1)	BFQCAF2 (1)	BFQCAF3 (1)	BFQACTV (1)	BFQNEWO (1)	BFQENDL (1)	NOT USED	BFQCAF0 (9)	BFQCAF1 (9)	BFQCAF2 (9)	BFQCAF3 (9)	BFQACTV (9)	BFQNEWO (9)	BFQENDL (9)	NOT USED
6	BFQCAF0 (2)	BFQCAF1 (2)	BFQCAF2 (2)	BFQCAF3 (2)	BFQACTV (2)	BFQNEWO (2)	BFQENDL (2)	NOT USED	BFQCAF0 (10)	BFQCAF1 (10)	BFQCAF2 (10)	BFQCAF3 (10)	BFQACTV (10)	BFQNEWO (10)	BFQENDL (10)	NOT USED
7	BFQCAF0 (3)	BFQCAF1 (3)	BFQCAF2 (3)	BFQCAF3 (3)	BFQACTV (3)	BFQNEWO (3)	BFQENDL (3)	NOT USED	BFQCAF0 (11)	BFQCAF1 (11)	BFQCAF2 (11)	BFQCAF3 (11)	BFQACTV (11)	BFQNEWO (11)	BFQENDL (11)	NOT USED
8	BFQCAF0 (4)	BFQCAF1 (4)	BFQCAF2 (4)	BFQCAF3 (4)	BFQACTV (4)	BFQNEWO (4)	BFQENDL (4)	NOT USED	BFQCAF0 (12)	BFQCAF1 (12)	BFQCAF2 (12)	BFQCAF3 (12)	BFQACTV (12)	BFQNEWO (12)	BFQENDL (12)	NOT USED
9	BFQCAF0 (5)	BFQCAF1 (5)	BFQCAF2 (5)	BFQCAF3 (5)	BFQACTV (5)	BFQNEWO (5)	BFQENDL (5)	NOT USED	BFQCAF0 (13)	BFQCAF1 (13)	BFQCAF2 (13)	BFQCAF3 (13)	BFQACTV (13)	BFQNEWO (13)	BFQENDL (13)	NOT USED
10	BFQCAF0 (6)	BFQCAF1 (6)	BFQCAF2 (6)	BFQCAF3 (6)	BFQACTV (6)	BFQNEWO (6)	BFQENDL	NOT USED	BFQCAF0 (14)	BFQCAF1 (14)	BFQCAF2 (14)	BFQCAF3 (14)	BFQACTV (14)	BFQNEWO (14)	BFQENDL (14)	NOT USED
11	BFQCAF0 (7)	BFQCAF1 (7)	BFQCAF2 (7)	BFQCAF3 (7)	BFQACTV (7)	BFQNEWO (7)	BFQENDL (7)	NOT USED	BFQCAF0 (15)	BFQCAF1 (15)	BFQCAF2 (15)	BFQCAF3 (15)	BFQACTV (15)	BFQNEWO (15)	BFQENDL (15)	NOT USED

RIGHT HALFWORD 6

OCTET	BIT															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	BRQX6R(0-15)															
1	BRQY6R(0-15)															
2	BRQZ6R(0-15)															
3	BIQIMM0R(0-15)															
4	BFQCAF0 (16)	BFQCAF1 (16)	BFQCAF2 (16)	BFQCAF3 (16)	BFQACTV (16)	BFQNEWO (16)	BFQENDL (16)	NOT USED	BFQCAF0 (24)	BFQCAF1 (24)	BFQCAF2 (24)	BFQCAF3 (24)	BFQACTV (24)	BFQNEWO (24)	BFQENDL (24)	NOT USED
5	BFQCAF0 (17)	BFQCAF1 (17)	BFQCAF2 (17)	BFQCAF3 (17)	BFQACTV (17)	BFQNEWO (17)	BFQENDL (17)	NOT USED	BFQCAF0 (25)	BFQCAF1 (25)	BFQCAF2 (25)	BFQCAF3 (25)	BFQACTV (25)	BFQNEWO (25)	BFQENDL (25)	NOT USED
6	BFQCAF0 (18)	BFQCAF1 (18)	BFQCAF2 (18)	BFQCAF3 (18)	BFQACTV (18)	BFQNEWO (18)	BFQENDL (18)	NOT USED	BFQCAF0 (26)	BFQCAF1 (26)	BFQCAF2 (26)	BFQCAF3 (26)	BFQACTV (26)	BFQNEWO (26)	BFQENDL (26)	NOT USED
7	BFQCAF0 (19)	BFQCAF1 (19)	BFQCAF2 (19)	BFQCAF3 (19)	BFQACTV (19)	BFQNEWO (19)	BFQENDL (19)	NOT USED	BFQCAF0 (27)	BFQCAF1 (27)	BFQCAF2 (27)	BFQCAF3 (27)	BFQACTV (27)	BFQNEWO (27)	BFQENDL (27)	NOT USED
8	BFQCAF0 (20)	BFQCAF1 (20)	BFQCAF2 (20)	BFQCAF3 (20)	BFQACTV (20)	BFQNEWO (20)	BFQENDL (20)	NOT USED	BFQCAF0 (28)	BFQCAF1 (28)	BFQCAF2 (28)	BFQCAF3 (28)	BFQACTV (28)	BFQNEWO (28)	BFQENDL (28)	NOT USED
9	BFQCAF0 (21)	BFQCAF1 (21)	BFQCAF2 (21)	BFQCAF3 (21)	BFQACTV (21)	BFQNEWO (21)	BFQENDL (21)	NOT USED	BFQCAF0 (29)	BFQCAF1 (29)	BFQCAF2 (29)	BFQCAF3 (29)	BFQACTV (29)	BFQNEWO (29)	BFQENDL (29)	NOT USED
10	BFQCAF0 (22)	BFQCAF1 (22)	BFQCAF2 (22)	BFQCAF3 (22)	BFQACTV (22)	BFQNEWO (22)	BFQENDL (22)	NOT USED	BFQCAF0 (30)	BFQCAF1 (30)	BFQCAF2 (30)	BFQCAF3 (30)	BFQACTV (30)	BFQNEWO (30)	BFQENDL (30)	NOT USED
11	BFQCAF0 (23)	BFQCAF1 (23)	BFQCAF2 (23)	BFQCAF3 (23)	BFQACTV (23)	BFQNEWO (23)	BFQENDL (23)	NOT USED	BFQCAF0 (31)	BFQCAF1 (31)	BFQCAF2 (31)	BFQCAF3 (31)	BFQACTV (31)	BFQNEWO (31)	BFQENDL (31)	NOT USED

LEFT HALFWORD 7

OCTET	BIT															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	BRQX7L(0-15)															
1	BRQY7L(0-15)															
2	BRQZ7L(0-15)															
3	BIQIMM1L(0-15)															
4	NOT USED															
5	NOT USED															
6	NOT USED															
7	NOT USED															
8	NOT USED															
9	NOT USED															
10	NOT USED															
11	NOT USED															

RIGHT HALFWORD 7

OCTET	BIT															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	BRQX7R(0-15)															
1	BRQY7R(0-15)															
2	BRQZ7R(0-15)															
3	BIQIMM1R(0-15)															
4	NOT USED															
5	NOT USED															
6	NOT USED															
7	NOT USED															
8	NOT USED															
9	NOT USED															
10	NOT USED															
11	NOT USED															

(D)127613 (4/4)

Figure A-8. 4X MBU Details Map
(Sheet 4 of 4)

LEFT HALFWORD 0

OCTET	BIT															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	ALQSHENC(0-2)			AOQPPOFX	ASQIDXCT(0-3)											
1	ALQSHENC(3-4)		ALQADDCR (8)	AOQSGNCV (0)	AOQSGNCV (2)	AOQCVSGN	AOQLSHOF	AOQFLFXOF								
2	ALQFLUFN	ALQAC1 (25)	ALQFLOFN	AOQGINF(0-4)												
3	AXQUFFLT	AXQFLTUF	NOT USED	AOQGIND(0-4)												
4	NOT USED			ASQGZERO	AOQRCDT	ASQFXOF	AOQDVCHK	ASQITMCT								
5	NOT USED			AOQOFFX	AOQOFFL	AOQUFFL	AOQGZERO	AOQRL								
6	NOT USED			AOQRG	AOQRE	AOQCL	AOQCG	AOQCE								
7	NOT USED			AOQPPZER	ASQESL	ASQRFODD	AOQPPINL	ASQXOFSF								
8	NOT USED															
9	NOT USED															
	AUCTL2B 1ALL			AUCTL3B 1BLJ					NOT USED							

RIGHT HALFWORD 0

OCTET	BIT															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0																
1																
2																
3																
4																
5																
6																
7																
8																
9																
	NOT USED															

LEFT HALFWORD 1

OCTET	BIT															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	AXQCTL(0-7) [AXQROM(0-7)]															
1	AXQCTL (8-13) [AXQROM(8-12)]					AXQ-RECZER	AXQDIV	AXQFLT								
2	AXQEXP(0-7)															
3	AXQEXPH(0-7)															
4	AXQSGN	AXQSGNH	NOT USED													
5	NOT USED															
6	NOT USED															
7	NOT USED															
8	NOT USED															
9	NOT USED															
	AUCTL5B 1CLU								NOT USED							

RIGHT HALFWORD 1

OCTET	BIT															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0																
1																
2																
3																
4																
5																
6																
7																
8																
9																
	NOT USED															

(D)127614 (1/4)

Figure A-9. 4X AU Details Map
(Sheet 1 of 4)

LEFT HALFWORD 2

OCTET	BIT															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	AMQPKPK ROM(111)	AMQORD ROM(144)	AMQBITR ROM(128)	NOT USED	AMQHL	AMQDL	AMQ FLARC(2) ROM(93)	NOT USED		AMQPLFXC ROM(90)	AMQKG8EN ROM(135)	NOT USED		AMQFLCTL ROM(131)	AMQSTNFX ROM(154)	NOT USED
1	AIQAB(0-15)															
2	AIQCD(0-15															
3	AEQSOR(0-15															
4	AEQLOR(0-15)															
5	ARQSH(0-15															
6	ARQNS(0-15															
7	AAQADD(0-15															
8	NOT USED															
9	NOT USED															
	AU4ADDA(0) 1ALD				AU4ADDA(1) 1ALE				AU4ADDA(2) 1ALF				AU4ADDA(3) 1ALG			

RIGHT HALFWORD 2

OCTET	BIT															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	NOT USED	AMQFXFLC ROM(95	AMQFL ROM(83)	NOT USED		AMQODSRM ROM(143)	AMQSRM1L ROM(132)	NOT USED	AMQSRCH ROM(153)	AMQSRCK2 ROM(118)	AMQ FLARC(1) ROM(85)	NOT USED		AMQMYCK2 ROM(141)	AMQMYCK4 ROM(142)	NOT USED
1	AIQAB(16-31)															
2	AIQCD(16-31															
3	AEQSOR(16-31															
4	AEQLOR(16-31)															
5	ARQSH(16-31)															
6	ARQNS(16-31)															
7	AAQADD(16-31)															
8	NOT USED															
9	NOT USED															
	AU4ADDA(4)A 1ALH				AU4ADDA(5)A 1ALI				AU4ADDA(6)A 1ALJ				AU4ADDA(7)A 1ALK			

LEFT HALFWORD 3

OCTET	BIT															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	AMQSHIF ROM(116)	AMQFIRS ROM(82)	AMQ FLARC(0) ROM(84)	NOT USED	AMQCIRC ROM(129	AMQARSH ROM(74	NOT USED		AMQVDP ROM(51	NOT USED	AMQPSOPL ROM(145	NOT USED	AMQREP ROM(137	AMQSSMAL ROM(136	AMQVMAX ROM(149)	NOT USED
1	AIQAB(32-47)															
2	AIQCD(32-47															
3	AEQSOR(32-47															
4	AEQLOR(32-47															
5	ARQSH(32-47															
6	ARQNS(32-47															
7	ADQADD(32-47															
8	NOT USED															
9	NOT USED															
	AU4ADDA(8) 1ALM				AU4ADDA(9 1ALN				AU4ADDA(10) 1ALO				AU4ADDA(11) 1ALP			

RIGHT HALFWORD 3

OCTET	BIT															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	AMQSHMPG ROM(148	AMQSRM ROM(139)	AMQSRMB ROM(138	NOT USED	AMQ-HLDOR	NOT USED	AMQ1 ROM(155	NOT USED	AMQ2 ROM(156	AMQ3 ROM(157)	AMQ4 ROM(158	NOT USED	AMQSHXM ROM(150)	AMQFXAR ROM(133)	AMQFLAR ROM(86)	NOT USED
1	AIQAB(48-63)															
2	AIQCD(48-63)															
3	AEQSOR(48-63,															
4	AEQLOR(48-63															
5	ARQSH(48-63															
6	ARQNS(48-63,															
7	AAQADD(48-63,															
8	NOT USED															
9	NOT USED															
	AU4ADDA(12) 1ALQ				AU4ADDA(13) 1ALR				AU4ADDA(14) 1ALS				AU4ADDA(15) 1ALT			

(D)127614 (2/4)

Figure A-9. 4X AU Details Map
(Sheet 2 of 4)

LEFT HALFWORD 4

<table>
<tr><th rowspan="2">OCTET</th><th colspan="16">BIT</th></tr>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>11</th><th>12</th><th>13</th><th>14</th><th>15</th></tr>
<tr><td>0</td><td colspan="16">ASQACC(0-15)</td></tr>
<tr><td>1</td><td colspan="16">AXQMPS(0-15)</td></tr>
<tr><td>2</td><td colspan="16">AXQMPC(1-16)</td></tr>
<tr><td>3</td><td colspan="16">AOQEF(0-15)</td></tr>
<tr><td>4</td><td colspan="16">ALQNORM(0-15)</td></tr>
<tr><td>5</td><td colspan="16">NOT USED</td></tr>
<tr><td>6</td><td colspan="16">NOT USED</td></tr>
<tr><td>7</td><td colspan="16">NOT USED</td></tr>
<tr><td>8</td><td colspan="16">NOT USED</td></tr>
<tr><td>9</td><td colspan="16">NOT USED</td></tr>
<tr><td></td><td colspan="8">AUOUTB(0)
1BLB</td><td colspan="8">AUOUTB(1)
1BLC</td></tr>
</table>

RIGHT HALFWORD 4

<table>
<tr><th rowspan="2">OCTET</th><th colspan="16">BIT</th></tr>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>11</th><th>12</th><th>13</th><th>14</th><th>15</th></tr>
<tr><td rowspan="2">0</td><td colspan="16">ASQACC(16-31)</td></tr>
<tr><td>16</td><td>17</td><td>18</td><td>19</td><td>20</td><td>21</td><td>22</td><td>23</td><td>24</td><td>25</td><td>26</td><td>27</td><td>28</td><td>29</td><td>30</td><td>31</td></tr>
<tr><td>1</td><td colspan="16">AXQMPS(16-31)</td></tr>
<tr><td>2</td><td colspan="16">AXQMPC(17-32)</td></tr>
<tr><td>3</td><td colspan="16">AOQEF(16-31)</td></tr>
<tr><td>4</td><td colspan="16">ALQNOPM(16-31)</td></tr>
<tr><td>5</td><td colspan="16">NOT USED</td></tr>
<tr><td>6</td><td colspan="16">NOT USED</td></tr>
<tr><td>7</td><td colspan="16">NOT USED</td></tr>
<tr><td>8</td><td colspan="16">NOT USED</td></tr>
<tr><td>9</td><td colspan="16">NOT USED</td></tr>
<tr><td></td><td colspan="8">AUOUTB(2)
1BLD</td><td colspan="8">AUOUTB(3)
1BLE</td></tr>
</table>

LEFT HALFWORD 5

<table>
<tr><th rowspan="2">OCTET</th><th colspan="16">BIT</th></tr>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>11</th><th>12</th><th>13</th><th>14</th><th>15</th></tr>
<tr><td rowspan="2">0</td><td colspan="16">ASQACC(32-47)</td></tr>
<tr><td>32</td><td>33</td><td>34</td><td>35</td><td>36</td><td>37</td><td>38</td><td>39</td><td>40</td><td>41</td><td>42</td><td>43</td><td>44</td><td>45</td><td>46</td><td>47</td></tr>
<tr><td>1</td><td colspan="16">AXQMPS(32-47)</td></tr>
<tr><td>2</td><td colspan="16">AXQMPC(32-47)</td></tr>
<tr><td>3</td><td colspan="16">AOQEF(32-47)</td></tr>
<tr><td>4</td><td colspan="16">ALQNOPM(32-47)</td></tr>
<tr><td>5</td><td colspan="16">NOT USED</td></tr>
<tr><td>6</td><td colspan="16">NOT USED</td></tr>
<tr><td>7</td><td colspan="16">NOT USED</td></tr>
<tr><td>8</td><td colspan="16">NOT USED</td></tr>
<tr><td>9</td><td colspan="16">NOT USED</td></tr>
<tr><td></td><td colspan="8">AUOUTB(4)
1BLF</td><td colspan="8">AUOUTB(5)
1BLG</td></tr>
</table>

RIGHT HALFWORD 5

<table>
<tr><th rowspan="2">OCTET</th><th colspan="16">BIT</th></tr>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>11</th><th>12</th><th>13</th><th>14</th><th>15</th></tr>
<tr><td>0</td><td colspan="16">ASQACC(48-63)</td></tr>
<tr><td>1</td><td colspan="16">AXQMPS(48-63)</td></tr>
<tr><td>2</td><td colspan="16">AXQMPC(48-63)</td></tr>
<tr><td>3</td><td colspan="16">AOQEF(48-63)</td></tr>
<tr><td>4</td><td colspan="16">ALQNORM(48-63)</td></tr>
<tr><td rowspan="2">5</td><td colspan="16">ANQOGT(0-15)</td></tr>
<tr><td colspan="3"></td><td colspan="9">ANQADDR(0-8)</td><td>ALQDZNRM</td><td>ALQDZACC</td><td></td><td>AOQHWRES</td></tr>
<tr><td rowspan="2">6</td><td rowspan="2">AMQLNTCR</td><td colspan="15">AMQNGT(1-15)</td></tr>
<tr><td colspan="2"></td><td colspan="9">AMQADDR(0-8)</td><td>AMQ</td><td>AMQACTAC</td><td>ASQACCEX</td><td>AMQVHW</td></tr>
<tr><td>7</td><td colspan="4">ALQGDGT(0-3)</td><td colspan="9">AOQAE(0-8)</td><td>AAQADXOF</td><td>AOQPPZER</td><td>AOQPPINL</td></tr>
<tr><td>8</td><td colspan="4">ALQCFXSF(0-4)</td><td colspan="12">NOT USED</td></tr>
<tr><td>9</td><td colspan="16">NOT USED</td></tr>
<tr><td></td><td colspan="8">AUOUTB(6)
1BLH</td><td colspan="8">AUOUTB(7)
1BLI</td></tr>
</table>

(D)127614 (3/4)

Figure A-9. 4X AU Details Map
(Sheet 3 of 4)

LEFT HALFWORD 6

OCTET	BIT 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	AXQABM(0-15) 00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
1	AXQABL(32-47) 32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
2	AXQCDM(0-15)															
3	AXQCDL(32-47)															
4	AXQDVRM(0-15)															
5	AXQDVRL(32-47)															
6	AXQDVNDM(0-15)															
7	AXQDVNDL(32-47)															
8	AXQMODFM(0-15)															
9	AXQMODFL(32-47)															
	AUMULTA(0) 1CLB				AUMULTA(1) 1CLD				AUMULTA(2) 1CLG				AUMULTA(3) 1CLJ			

RIGHT HALFWORD 6

OCTET	BIT 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	AXQABM(16-31) 16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
1	AXQABL(48-63) 48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
2	AXQCDM(16-31)															
3	AXQCDL(48-63)															
4	AXQDVRM(16-31)															
5	AXQDVRL(48-63)															
6	AXQDVNDM(16-31)															
7	AXQDVNDL(48-63)															
8	AXQMODFM(16-31)															
9	AXQMODFL(48-63)															
	AUMULTA(4) 1CLM				AUMULTA(5) 1CLO				AUMULTA(6) 1CLR				AUMULTA(7) 1CLT			

LEFT HALFWORD 7

OCTET	BIT 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	FF(0) AMQACC(1) ROM(64)	FF(1) AMQACC(6) ROM(65)	FF(2) AMQACC(8) ROM(66)	FF(3) AMQSHFAC	FF(4) AMQAOFSL(0-3) ROM(69)	FF(5) ROM(70)	FF(6) ROM(71)	FF(7) ROM(72)	FF(8) AMQARCP ROM(73)	FF(9) AMQARSH ROM(74)	ANQOGT(0) ROM(0)	ANQOGT(11) ROM(11)	ANQOGT(12) ROM(12)	AEQED(1-3)		
1	FF(10) AMQBTNRC ROM(75)	FF(11) AMQCRCSH ROM(77)	FF(12) AMQDIV ROM(78)	FF(13) AMQDIVDP ROM(79)	FF(14) AMQDLMUL ROM(81)	FF(15) AMQFIRS ROM(82)	FF(16) AM- FLARC(0) ROM(84)	FF(17) AM- QFLARC(1) ROM(85)	FF(18) AMQFLAR ROM(86)	FF(19) AMQFLDAB ROM(87)	ANQOGT(14) ROM(15)	ANQOGT(15) ROM(16)	ANQOGT(18) ROM(19)	AEQED(4-6)		
2	FF(20) AMQFLDIV ROM(88)	FF(21) AMQFLDNM ROM(89)	FF(22) AMQFLMDP ROM(91)	FF(23) AMQFLMUL ROM(92)	FF(24) AM- QFLARC(2) ROM(93)	FF(25) AMQFLSUB ROM(94)	FF(26) AMQFXNOR ROM(96)	FF(27) AMQFXSH ROM(97)	FF(28) AMQGFD31 ROM(98)	FF(29) AMQG8DIS ROM(99)	ANQOGT(19) ROM(20 23)	ANQOGT(20) ROM(21 24)	ANQOGT(21) ROM(22 25)	AEQED(7)	AMQINC	AAQADCR8
3	FF(30) AM- QXOFCD(1) ROM(101)	FF(31) AM- QXOFCD(2) ROM(102)	FF(32) AMQLGARS ROM(103)	FF(33) AMQLOGCD(0-2) ROM(104)	FF(34) ROM(105)	FF(35) ROM(106)	FF(36) AMQLOGCP ROM(107)	FF(37) AMQNORM(0-2) ROM(108)	FF(38) ROM(109)	FF(39) ROM(110)	ANQOGT(22-24) ROM(26)	ROM(27)	ROM(28)	AEQET	AEQAGTC	AEQEDARX
4	FF(40) AMQPKPK ROM(111)	FF(41) AMQPPCMP ROM(112)	FF(42) AMQPPCPS ROM(103)	FF(43) AMQPPINL ROM(114)	FF(44) AM- QRCEN(5) ROM(32)	FF(45) AM- QRCEN(7) ROM(33)	FF(46) AMQSTSHAC ROM(34)	FF(47)AM- QRCENC(2) ROM(35)	FF(48)AM- QRCENC(1) ROM(36)	FF(49)AM- QRCENC(0) ROM(37)	ANQOGT(26-27) ROM(30)	ROM(31)	AMQNGT(21) ROM(22)	AEQFLCAR	AEQEDEX	AEQSCHIE
5	FF(50) AMQFXDIV ROM(38)	FF(51) AMQHLDSG ROM(39)	FF(52) AMQRCEN(17-18) ROM(40)	FF(53) ROM(41)	FF(54) AMQRCEN(20,17) ROM(42)	FF(55) ROM(43)	FF(56) AMQDVINC	FF(57) AN- QSOFSL(3)	FF(58) AMQSLMUL ROM(47)	FF(59) AN- QRCEN(18)	AMQNGT20	ANQSDVCK	AMQNGT(1) ROM(1)	ARQNSEXT	AOQSGDEL	AAQADDEX
6	FF(60) AMQSOFSL(1-3) ROM(45)	FF(61) ROM(46)	FF(62) ROM(47)	ANQSRCK2	FF(64) AXQMODLSB ROM(49)	FF(65) AM- QXOFCD(0) ROM(50)	FF(66) AMQSCCD ROM(51)	FF(67) AMQVDP ROM(52)	FF(68) AMQVECT ROM(53)	FF(69) AM- QXCTL(0)	AMQNGT(2) ROM(2)	AMQNGT(4-5) ROM(4)	ROM(5)	AAQFPS(0)	AAQFPS(2)	ARQDCD(0)
7	FF(70) AMQADTYP ROM(68)	FF(71) AMQDL	FF(72) AMQFL ROM(83)	FF(73) AMQFLFXC ROM(90)	FF(74) AMQFXFLC ROM(95)	FF(75) AMQHL	FF(76) AMQHWQOT ROM(100)	FF(77) AMQDIVFX	FF(78) AM- QCONT(11) ROM(76)	FF(79) ANQARCP	AMQNGT(6-8) ROM(6)	ROM(7)	ROM(8)	NOT USED		
8	AMQNGT(9-10) ROM(9)	ROM(10)	AM- QNGT(13) ROM(13)	AMQNGT(16-17) ROM(17)	ROM(18)	ANQXOFCD(1-2)		ANQOGT(1-2)		AMQNGT(3) ROM(3)	AN- QSOFSL(1)	ANQOGT(4-5)		NOT USED		
9	ANQOGT(6-10)					ANQOGT(13)	ANQOGT(16-17)		ANQOGT(25)	AM- QNGT(19)	ANQRCEN(5)	ANQRCEN(7)	AN- QXOFCD(0)	NOT USED		
	AUROMFFB 1BLK													AU4CTL1A 1ALL		

RIGHT HALFWORD 7

OCTET	BIT 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0																
1																
2																
3																
4																
5																
6																
7																
8																
9																
	NOT USED															

(D)127614 (4/4)

Figure A-9. 4X AU Details Map
(Sheet 4 of 4)

CP MASTER HARD CORE

4XCP UNIT REGISTER FORMAT

READ 440n WHEREn =BYTE NUMBER

BYTE #,n \ BIT	0	1	2	3	4	5	6	7
0	STATE (1,-8)							
1	CCRI(12-15)				CCR(12-15)			
2	RPF	RP(0)	RP(1)	GCC	GSE	GAT	GCB	GRZ
3	CCRI (II)	RPLY	HCINIT	ABORT	ZROPN	MHCCMP	MHCABT	CSR
4	ERR	ERRF	SYSERR	AUTO	RZF	RIPF	EXCHF	EXCMD
5	AT	AB	INTRP	VBG	MCP	MCW	MCWF	CLCMP
6	PRV	PAR	ILLO	AREXC	MIERR	AMERR	ICMP	IABT
7	MABT(0-3)				AABT(0-3)			
8	MCMP(0-3)				ACMP(0-3)			
9	PV	PE	IL	AE	ME	RZ(0-2)		
A	TR	AS	AC	SB	MC	SC	SS	RB

(B)125813 (1/3)

Figure A-10. Master Hardcore UR Format

READ 444n WHEREn =BYTE NUMBER

<table>
<tr><th>IPU HARD CORE
BYTE # .n \ BIT</th><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th></tr>
<tr><td>0</td><td>FREEZE</td><td colspan="7">STATE (0-6)</td></tr>
<tr><td>1</td><td colspan="4">LSD (0-3)</td><td>EXCH</td><td>HCREQ</td><td>HCINP</td><td>LDPTR</td></tr>
<tr><td>2</td><td>SPSDW</td><td>▨</td><td>▨</td><td colspan="5">OCTR (0-4)</td></tr>
<tr><td>3</td><td>▨</td><td>▨</td><td>PRM (0)</td><td>PRM (1)</td><td>▨</td><td colspan="3">RC (0-2)</td></tr>
<tr><td>4</td><td>RDA</td><td>RDS</td><td>RA</td><td>AR</td><td>DAV</td><td>DAV</td><td>PAR</td><td>PAR</td></tr>
<tr><td>5</td><td>RDA</td><td>WRITE</td><td>PRV</td><td>IPPRV</td><td>PAE</td><td>IPPAE</td><td>AREX</td><td>IPIOP</td></tr>
<tr><td>6</td><td colspan="8" rowspan="2">▨</td></tr>
<tr><td>7</td></tr>
<tr><td>8</td><td colspan="2">IP (0-1)</td><td colspan="2">OP (0-1)</td><td colspan="4">BSY (0-3)</td></tr>
<tr><td>9</td><td colspan="4">ACT (0-3)</td><td colspan="4">PRV (0-3)</td></tr>
<tr><td>A</td><td>CUE0 (0)</td><td>CUE1 (0)</td><td>CUE0 (1)</td><td>CUE1 (1)</td><td>CUE0 (2)</td><td>CUE1 (2)</td><td>CUE0 (3)</td><td>CUE1 (3)</td></tr>
<tr><td>B</td><td colspan="8">OA (8-15)</td></tr>
<tr><td>C</td><td colspan="8">OA (16-23)</td></tr>
<tr><td>D</td><td colspan="8">OA (24-31)</td></tr>
<tr><td>E</td><td>▨</td><td colspan="3">P3 (29-31)</td><td>▨</td><td colspan="3">PA (29-31)</td></tr>
</table>

(B)125813 (2/3)

Figure A-11. IPU Hardcore UR Format

MBU HARD CORE

READ 44mn, WHERE m = 2 FOR PIPE 0 MBU,
m = 3 FOR PIPE 1 MBU,
m = 6 FOR PIPE 2 MBU,
m = 7 FOR PIPE 3 MBU

BYTE # \ BIT	0	1	2	3	4	5	6	7
0	PM (0-1)		ZBR	AR	RA	RDA	RDS	OAFUL
1	OA (8-15)							
2	OA (16-23)							
3	OA (24-28)					RSTAT (0-2)		
4	OZC (0-7)							
5	LSD (0-3)				HCMPRV	HPARER	CCMPRV	CPARER
6	HSTATE (0-3)				HCREQ	HCINP	RNEQ0	EXCH
7	HCCNT (0-3)				UNCMP	ABTRM	ILOPR	▨
8	DSTATE (0-2)			DSCMP	WRAP (0-1)		AVDES (0-1)	

AU HARD CORE

Figure A-12. MBU Hardcore UR Format

READ 44mn , WHERE m = 8 FOR PIPE 0 AU
m = A FOR PIPE 1 AU
m = C FOR PIPE 2 AU
m = E FOR PIPE 3 AU

BYTE # \ BIT	0	1	2	3	4	5	6	7
0	ADDR (8-15)							
1	ADDR (16-23)							
2	ADDR (24-31)							
3	ZCB (0-7)							
4	PM (0-1)		AR	MBFUL	WRCMP	PR	PA	▨
5	STATE (0-7)							
6	LSD (0-3)				OCTCT (0-3)			

(B)125813 (3/3)

Figure A-13. AU Hardcore UR Format

APPENDIX B
X4 IPU LISTINGS, DIAGRAMS, AND DESCRIPTIONS

Page

- I4FILE Circuit Board Description. . . . . B-1 thru B-35
- I4CMREQ Circuit Board Description . . . . B-37 thru B-109
  - Look Ahead Controller Flowchart. B-56
  - Central Memory Requester Flowchart B-85
- I4ADDR Circuit Board Description. . . . . B-111 thru B-151
- X4 IPU Registers. B-153
- X4 IPU Block Diagram. B-155
- X4 IPU Controller Diagram B-157
- X4 IPU Controllers and Flip-flops B-158
- Program Status Word Registers B-163
- X4 IPU ROM 1 Listing B-164
- X4 IPU ROM 2 Listing B-166
- X4 IPU Interface Signals. B-167
- X4 IPU Glossary B-184

INTRODUCTION - BLOCK DIAGRAM SYMBOLS

The block diagram at the end of this section (figure B-6) condenses the information contained in the logic diagrams to a one sheet representation. The diagram contains all data lines and control signatures that are inputs to or outputs from the circuit board. In addition many of the key data and control lines internal to the board are illustrated. Other information contained on the block diagram is illustrated in figure B-1, and includes:

- Sheet Number - Location of the logic diagram for the depicted function within the logic set for the circuit board. Multiple sheets are referenced with a letter that is explained in the notes on the diagram.
- Pin Number - Circuit board input/output pin that carries the indicated signal. When more than one pin number appears, the pin for the most significant bit appears first (data), or the pin numbers appear in the order of the signatures listed on the line (control). Large quantities of pins are in tabular form in the notes on the diagram sheet.
- Bus size - Numbers on all lines indicate the number of bits represented by that particular line. A line without a number contains only one bit. The representation "8 x 32" indicates a multiple word transfer of eight, 32-bit words.

Data lines are heavy black lines. Control lines are single width lines.

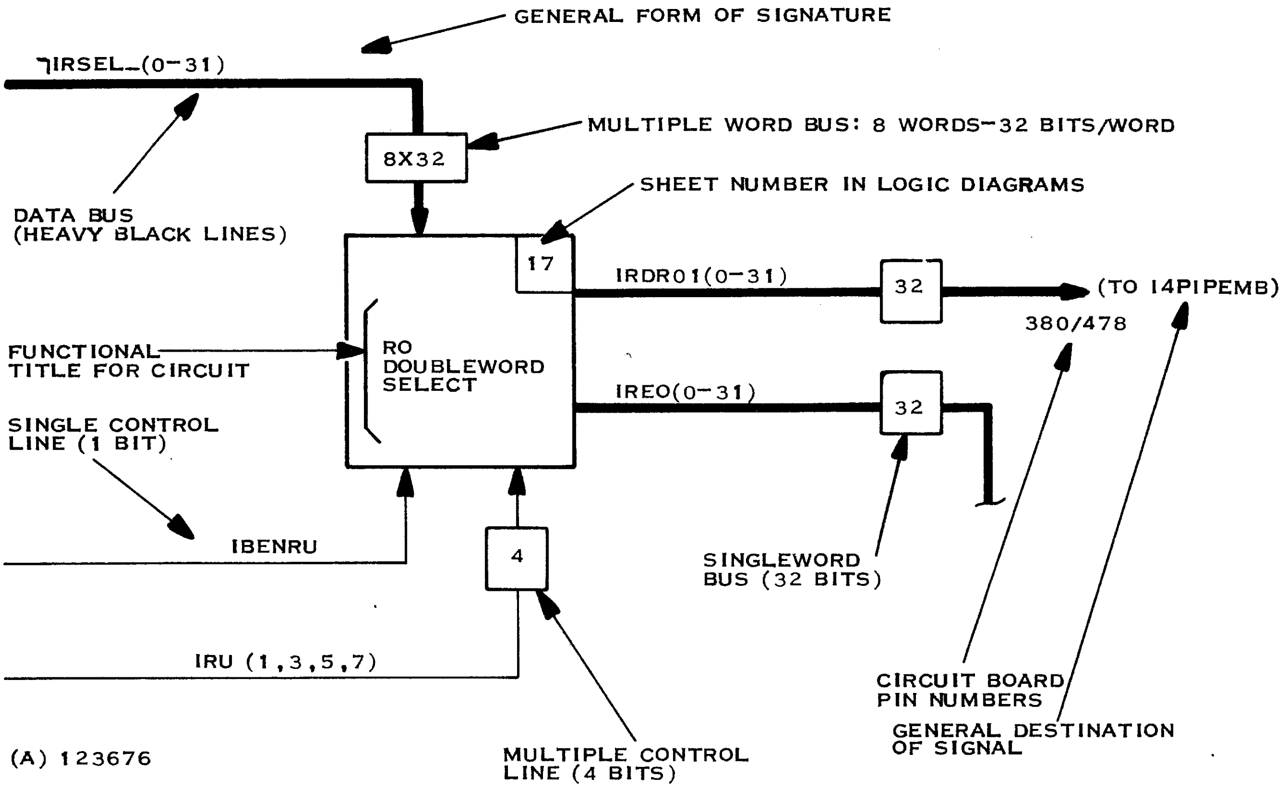

GENERAL FORM OF SIGNATURE
¬IRSEL_(0-31)
MULTIPLE WORD BUS: 8 WORDS-32 BITS/WORD
8X32
SHEET NUMBER IN LOGIC DIAGRAMS
DATA BUS
(HEAVY BLACK LINES)
17
IRDR01(0-31)
32
(TO 14PIPEMB)
380/478
FUNCTIONAL TITLE FOR CIRCUIT
RO DOUBLEWORD SELECT
IREO(0-31)
32
SINGLE CONTROL LINE (1 BIT)
IBENRU
4
SINGLEWORD BUS (32 BITS)
IRU (1,3,5,7)
CIRCUIT BOARD PIN NUMBERS
GENERAL DESTINATION OF SIGNAL
(A) 123676
MULTIPLE CONTROL LINE (4 BITS)

Figure B-1. Key to Symbols - Circuit Board Block Diagrams

I4FILE CIRCUIT BOARD

The I4FILEMB contains sixteen I4FILE circuit boards that comprise the following major components of the IPU:

KCM Memory Interface File	ILQKCMO-7(0-31)
KA Octet Buffer	ILQKAO-7(0-31)
KB Octet Buffer	ILQKBO-7(0-31)
Register File:	
Base Address File A	IFQA1-7(0-31)
Base Address File B	IFQB1-7(0-31)
General Arithmetic File C	IFQC1-7(0-31)
General Arithmetic File D	IFQD1-7(0-31)
Index Address File I	IFQI1-7(0-31)
Vector Parameter File V	IFQV1-7(0-31)

Each I4FILE circuit board contains two bits of each word in each octet of the above components, plus the required gating circuits to load each octet and retrieve information from each octet. The bits are distributed to the sixteen I4FILE circuit boards as listed in table B-1.

Table B-1. I4FILE Bit Slice Distribution

Octet Word Bits	Circuit Board	Location within I4FILEMB (3A)
0 and 16	I4FILE(O)	LT
1 and 17	I4FILE(1)	LS
2 and 18	I4FILE(2)	LU
3 and 19	I4FILE(3)	LC
4 and 20	I4FILE(4)	LQ
5 and 21	I4FILE(5)	LP
6 and 22	I4FILE(6)	LO
7 and 23	I4FILE(7)	LD
8 and 24	I4FILE(8)	LL
9 and 25	I4FILE(9)	LR
10 and 26	I4FILE(10)	LK
11 and 27	I4FILE(11)	LI
12 and 28	I4FILE(12)	LH
13 and 29	I4FILE(13)	LG
14 and 30	I4FILE(14)	LF
15 and 31	I4FILE(15)	LE

The enclosed fold-out block diagram illustrates all of the data paths and control signals that are implemented on the I4FILE circuit boards. The circuits include loading the register file, interface with the MCU, and gating circuits from the I4FILE octets to the AO, RO, XR, BR and IR registers. The following paragraphs describe the component circuits on the I4FILE circuit board and the required control signals to perform the transfers.

INTERFACE SIGNALS

Table B-2 defines the input signals to the I4FILE circuit board. Table B-3 defines the output signals from the I4FILE circuit board.

Table B-2. I4FILE Circuit Board Input Signals

Signature	Origin	Function

AOQEF00:2(0-31)
AOQEF01:2(0-31)
	AU Pipe 0	Doubleword (64-bit) input data from EF register. Appears as AOQEF0(0-63) at AU level.

AOQEF10:2(0-31)
AOQEF11:2(0-31)
	AU Pipe 1	Double word input data from EF register. Appears as AOQEF1(0-63) at AU level.

AOQEF20:2(0-31)
AOQEF21:2(0-31)
	AU Pipe 2	Doubleword input data from EF register. Appears as AOQEF2(0-63) at AU level.

AOQEF30:2(0-31)
AOQEF31:2(0-31)
	AU Pipe 3	Doubleword input data from EF register. Appears as AOQEF3(0-63) at AU level.

ICCMSTRB	I4HDCORE	Enables data input from MCU to KCM memory interface file.
ICLOCK:(0-15)	I4ADDR	System clock. Originates as $XCLOCK04:0(00-15). Clock pulses for register file and KA/KB buffers.
-ICWRITE	I4HDCORE	Enables data transfer from IPU to MCU.
-IIAUPP	I4PIPTOP	Selects AU word to BR for push, pull and modify instructions.
IIISEL(0-2)	I4PIPTOP	Selects word from IFQI_(0-31) in register file for transfer to XR register for indexing.
-IISELAB(0-3)	I4PIPTOP	Selects word from IFQA_(0-31) or IFQB_(0-31) in register file for transfer to BR register.
IIVTBR(0 and 1)	I4PIPTOP	Selects words 1, 2 or 3 from IFQV_(0-31) in register file for transfer to BR register during vector initialization.
-ILHAOIND	I4PIPTOP	Gates word from KCM to IR.
ILRFTIR	I4PIPTOP	When -ILHAOIND = "1", gates a word from KA or KB ("1"), or from register file ("0") to IR.

Table B-2. I4FILE Circuit Board Input Signals (Continued)

Signature	Origin	Function
ILSELK(29-31)	I4ADDR	Selects a word from KCM octet and from KA or KB octet to be subject to gating to IR.
-ILSELKA	I4PIPTOP	Enables output from KA to other IPU circuits.
-ILSELKB	I4PIPTOP	Enables output from KB to other IPU circuits.
IMCLEAR	I4HDCORE	Master clear to KCM.
IMCMTA	I4HDCORE	Gates octet from KCM to register file A.
IMCMTB	I4HDCORE	Gates octet from KCM to register file B.
IMCMTC	I4HDCORE	Gates octet from KCM to register file C.
IMCMTD	I4HDCORE	Gates octet from KCM to register file D.
IMCMTKA	I4CMREQ	Gates octet from KCM to KA buffer file.
IMCMTKB	I4CMREQ	Gates octet from KCM to KB buffer file.
IMCMTI	I4HDCORE	Gates octet from KCM to register file I.
IMCMTV	I4HDCORE	Gates octet from KCM to register file V.
-IMDTL0(0-31)	I4PIPE	Store details data lines to memory bus.
-IMDTL1(0-31)	I4PIPE	Store details data lines to memory bus.
-IMDTL2(8-31)	I4ADDR	Store details data lines to memory bus.
-IMDTL3(0-7)	I4RHAZ	Store details data lines to memory bus.
-IMDTL3(8-31)	I4ZHAZ	Store details data lines to memory bus.
-IMDTL4(0-2)	I4VECLAS	Store details data lines to memory bus.
-IMDTL4(3-5)	I4MISC	Store details data lines to memory bus.
-IMDTL4(6)	I4CMREQ	Store details data line to memory bus.
-IMDTL4(8-31)	I4ZHAZ	Store details data line to memory bus.

Table B-2. I4FILE Circuit Board Input Signals (Continued)

Signature	Origin	Function
-IMDTL5(0-7)	I4INFACE	Store details data lines to memory bus.
-IMDTL5(8-31)	I4ZHAZ	Store details data lines to memory bus.
-IMDTL6(0-7)	I4INFACE	Store details data lines to memory bus.
-IMDTL6(8-31)	I4ZHAZ	Store details data lines to memory bus.
-IMDTL7(0-3)	I4STATUS	Store details data lines to memory bus.
-IMDTL7(4-7)	I4CMREQ	Store details data lines to memory bus.
-IMDTL7(8-10)	I4PIPTOP	Store details data lines to memory bus.
-IMDTL7(11)	I4ROUTE3	Store details data line to memory bus.
-IMDTL7(12-15)	I4LVL3	Store details data lines to memory bus.
-IMDTL7(16-31)	I4INFACE	Store details data lines to memory bus.
IMFSLDIS	I4HDCORE	Disables selection of register file and memory interface files to MCU, register file or AO.
IMRE:4	I4HDCORE	Master reset to register file and KA/KB buffers.
IMSE:4	I4HDCORE	Preset to register file and KA/KB buffers.
IOCM0-7(0-31)	MCU	Bidirectional data bus to/from memory. Transfers octets only.
IRAOSEL(0-4)	I4MISC	Selects doubleword from register file, KCM, KA or KB for transfer to AO register. Also selects entry from register file to IR.
IRDISKSL	I4MISC	Disables selection of KCM, KA or KB to MCU, register file or AO.
IRELTAOR	I4MISC	Enables even left halfword to AO right half.

Table B-2. I4FILE Circuit Board Input Signals (Continued)

Signature	Origin	Function
IRELTROR	I4MISC	Enables even left halfword to RO right half.
IRENRU	I4MISC	Enables RO input selection.
IRLEMSEL(0-3)	I4STATUS	Gates one of four AU pipes' output to status register.
IROLTAOL	I4MISC	Enables odd left halfword to AO left half.
IROLTAOR	I4MISC	Enables odd left halfword to AO right half.
IROLTROL	I4MISC	Enables odd left halfword to RO left half.
IROLTROR	I4MISC	Enables odd left halfword to RO right half.
IRORTAOR	I4MISC	Enables odd right halfword to AO right half.
IRORTROR	I4MISC	Enables odd right halfword to RO right half.
IRROSEL(0-4)	I4MISC	Selects doubleword from register file for entry into RO .
-IVAUSEL(0-3)	I4MISC	Each bit selects one of the AU pipes for transferring data into the register file, or to the BR register for push, pull and modify.
IVREGDST	I4MISC	Enables input address decoding for the register file.
IVRSTACK (0-6, 8 and 9)	I4INFACE	Bits 0-6 are register file storage address; Bit 8 indicates a halfword store Bit 9 indicates a doubleword store (Bit 7 is not received by I4FILE).
-$XDTL2(0-7)	I4ADDR	Store details data lines to memory bus.

Table B-3. I4FILE Circuit Board Output Signals

Signature	Destination	Function
IFQV0(0-7)	I4MISC	Vector Op Code from register file V
IFQV0(8-11)	Not Used	
IFQV0(12)	I4MISC	Register file V output
IFQV0(13-15)	I4PIPTOP	Register file V output - Single valued vector
IFQV0(16-31)	I4INFACE	Register file V output - Vector length
IFQV1(0-4, 8-15)	Not used	
IFQV1(5-7)	I4PIPTOP	Register file V output - Index A vector
IFQV2 (0, 4, 8-15)	Not used	
IFQV2(1-3)	I4STATUS	Register file V output - halfword starting address
IFQV2(5-7)	I4PIPTOP	Register file V output - Index B vector
IFQV3(0-3)	I4MISC	Register file V output - Vector Increment direction
IFQV3(5-7)	I4PIPTOP	Register file V output - Index C vector
IFQV3(4, 8-15)	Not used	
IFQV5(16-31)	I4INFACE	Register file V output - Inner loop count
IFQV7(16-31)	I4INFACE	Register file V output - Outer loop count
IIDBR(0-31)	I4PIPE	Input word to BR register
-IIDXR(0-31)	I4PIPE	Input word to XR register
ILKTIR(0-31)	Not used	
-ILKTIR(0-31)	I4PIPE	Input word to IR register

Table B-3. I4FILE Circuit Board Output Signals (Continued)

Signature	Destination	Function
-ILQKCM0:2 (0-31)	I4PIPE	Load Details data lines from KCM
-ILQKCM1:2 (0-31)	I4PIPE	Load Details data lines from KCM
-ILQKCM2:2 (0-31)	I4ADDR	Load Details data lines from KCM
-ILQKCM3:2 (0-7)	I4RHAZ	Load Details data lines from KCM
-ILQKCM3:2 (8-31)	I4ZHAZ	Load Details data lines from KCM
-ILQKCM4:2 (0-2)	I4VECLAS	Load Details data lines from KCM
-ILQKCM4:2 (3-5)	I4MISC	Load Details data lines from KCM
-ILQKCM4:2(6)	I4CMREQ	Load Details data line from KCM
-ILQKCM4:2(7)	Not used	
-ILQKCM4:2 (8-31)	I4ZHAZ	Load Details data line from KCM
-ILQKCM5:2 (0-7)	I4INFACE	Load Details data lines from KCM
-ILQKCM5:2 (8-31)	I4ZHAZ	Load Details data lines from KCM
-ILQKCM6:2 (0-7)	I4INFACE	Load Details data lines from KCM
-ILQKCM6:2 (8-31)	I4ZHAZ	Load Details data lines from KCM
-ILQKCM7:2 (0-3)	I4STATUS	Load Details data lines from KCM

Table B-3. I4FILE Circuit Board Output Signals (Continued)

Signature	Destination	Function
-ILQKCM7:2 (4-7)	I4HDCORE/ I4CMREQ	Load Details data lines from KCM
-ILQKCM7:2 (8-10)	I4PIPTOP	Load Details data lines from KCM
-ILQKCM7:2(11)	I4ROUTE3	Load Details data line from KCM
-ILQKCM7:2 (12-15)	I4LVL3	Load Details data lines from KCM
-ILQKCM7:2 (16-31)	I4INFACE	Load Details data lines from KCM
IRDAO0(0-31)	I4PIPE	Input word to most significant 32 bits of AO.
IRDAO1(0-31)	I4PIPE	Input word to least significant 32 bits of AO.
IRDRO0(0-31)	I4PIPE	Input word to most significant 32 bits of RO.
IRDRO1(0-31)	I4PIPE	Input word to least significant 32 bits of RO.
IRLEMDAT(0-7)	I4STATUS	Input byte to status register from EF register of selected AU pipe.
IRLEMDAT(8-15)	Not used	

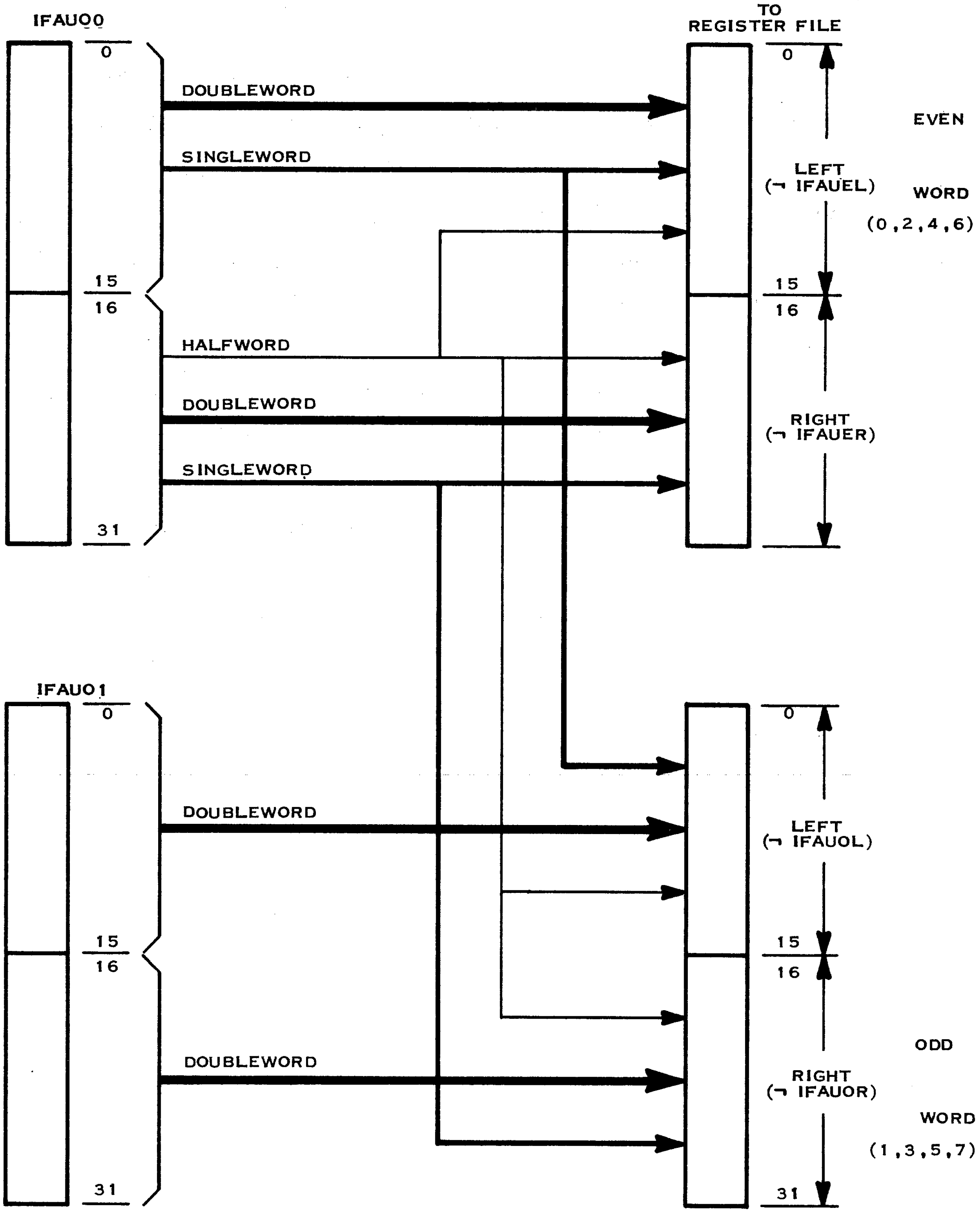
AU WORD SELECT

AU Word Select contains a four-input selectable multiplexer for each of the sixty-four bits of the incoming Arithmetic Unit (AU) word, plus eight four-input multiplexers that relay the eight most significant bits of the AU word to the status register on the I4PIPEMB. The inputs to each multiplexer originate in the EF output register of the four AU's. Each input is gated independently by a separate gate signal (- IVAUSEL(0-3)). However, only one gate signal can be active at any one time. For example, if -IVAUSEL(1) is active, it transfers the 64-bit input from AU1 (AOQEF10 (0-31) /AOQEF11(0-31)) into the Register File or to the BR register on the I4PIPEMB. Control gates on the I4CTLMB prevent the remaining IVAUSEL bits from becoming active. The data output (-IFAUO0/-IFAUO1 (0-31)) enters the register file for storage after control signals determine if it is a singleword, doubleword or halfword.

The eight most significant bits of the AU input (AOQEFX0:2(0-7)) may also transfer to the Program Status Doubleword register (IRQPSW) on the I4PIPEMB. Four select bits (IRLEMSEL(0-3)) from the Status Controller on the I4PIPEMB determine which AU pipe will be gated to the status register. The output from this selection (IRLEMDAT(0-7)) transfers to the I4PIPEMB.

WORD SIZE SELECT

Two control bits from the I4CTLMB (IVRSTACK 9 and 8) determine the word size of the incoming data from the selected AU pipe. If neither of these bits is set, a singleword size is selected (32 bits). The signal IFAUSW enables IFAUO0(0-15) to the left halfword inputs to the register file (IFAUEL/IFAUOL), and IFAUO0 (16-31) to the right halfword inputs to the register file (IFAUER/IFAUOR). Singleword inputs always originate from IFAUO0. If IVRSTACK(9) is set, a doubleword size is selected (64 bits). The signal IFAUDW gates IFAUO0 (0-31) to the left and right halves of the even word inputs to the register file (IFAUEL/IFAUER), and IFAUO1 (0-31) to the left and right halves of the odd word inputs to the register file (IFAUOL/IFAUOR). If IVRSTACK(8) is set, a halfword size is selected (16 bits). The signal IFAUHW transfers IFAUO0(16-31) to all four halfword inputs to the register file. A halfword input will always originate in the sixteen least significant bits of IFAUO0. Figure B-2 summarizes these transfer paths. IVRSTACK (8 and 9) cannot be set simultaneously.



(A)123678

Figure B-2. Word Size Select Paths

REGISTER FILE ADDRESS DETERMINATION

Seven control bits from I4CTLMB (IVRSTACK(0-6)) address the register file to the halfword level. The I4FILE circuit board decodes these bits to produce 94 halfword gate signals, one for each halfword in the register file. This circuit operates during an AU transfer into the register file.

IVRSTACK BITS. The IVRSTACK address originates in the register stack registers R5 through RC that correspond to each of the CP pipes (0-3). The register stack is located on I4HAZMB. When the operation in the AU pipe is complete, the address for storing the result transfers from register stack register C to I4CTLMB. If the address is less than or equal to 2F (the highest address in the register file), the IVRSTACK bits are sent to the I4FILE circuit boards to choose a register file location for storing the output from the AU pipe. The seven address bits allow the control circuit to designate any halfword in the register file. Bits 0, 1 and 2 select the octet within the file, bits 3, 4 and 5 isolate the word within the octet and bit 6 selects the right (bits 16-31) or left (bits 0-15) halfword. Refer to figure B-3 for an illustration of this bit breakdown.

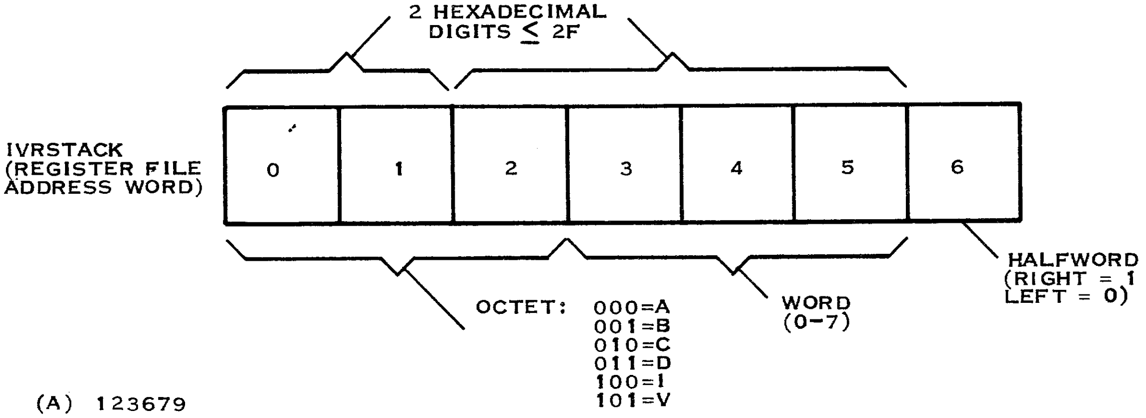


Figure B-3. IVRSTACK(0-6) Bit Assignments

IVRSTACK HARDWARE DECODE. Decode of the IVRSTACK bits to select a register file halfword is not a direct decode of the address bits. Since the control circuits may address either a halfword, a singleword, or a doubleword in the register file, I4FILE decodes the control bits in segments to allow for this capability. The decode circuit first examines bits 5 and 6 of the incoming address, plus IVRSTACK(8) (halfword select) and IVRSTACK (9) (doubleword select). The output from this examination designates the four possible combinations of half words represented by bits 5 and 6:

- Odd word, Right halfword (IFENGOR)
- Odd word, Left halfword (IFENGOL)
- Even word, Right halfword (IFENGER)
- Even word, Left halfword (IFENGEL)

If IVRSTACK(8) is set, the decode network generates one of these gating signals to select the proper halfword. During a singleword store, however, the differentiation between halfwords is not needed. The decode circuit produces two of the gating signals to select either the entire odd word, or the entire even word. If IVRSTACK(9) is set, this portion of the decode circuit is unnecessary. All four gating signals are generated.

The decode network network enabled by IVREGDST, then checks bits 0 and 1 of the incoming address to pick a group of two octets within the register file. If bit 0 is set, the address selects either the I or V files. If bit 1 is set, the address selects either the C or the D file. If neither of the two bits are set, the address selects either the A or B file. Since the control circuit has already determined that the address is less than or equal to 2F, bits 0 and 1 cannot be set simultaneously. This portion of the decode, combined with the first decode stage, isolates the halfword(s) within the two octets selected by bits 0 and 1.

The decode circuit then examines bits 2, 3 and 4 of the address. Bit 2 selects either the odd or the even octet of the two selected octets. Bits 3 and 4 determine the doubleword within that octet. When combined with the previously generated select bits, either one, two or four gate signals are produced to enter the incoming AU data word into the proper location in the register file. Figure B-4 illustrates the breakdown of the address word as viewed by the hardware decode circuit.

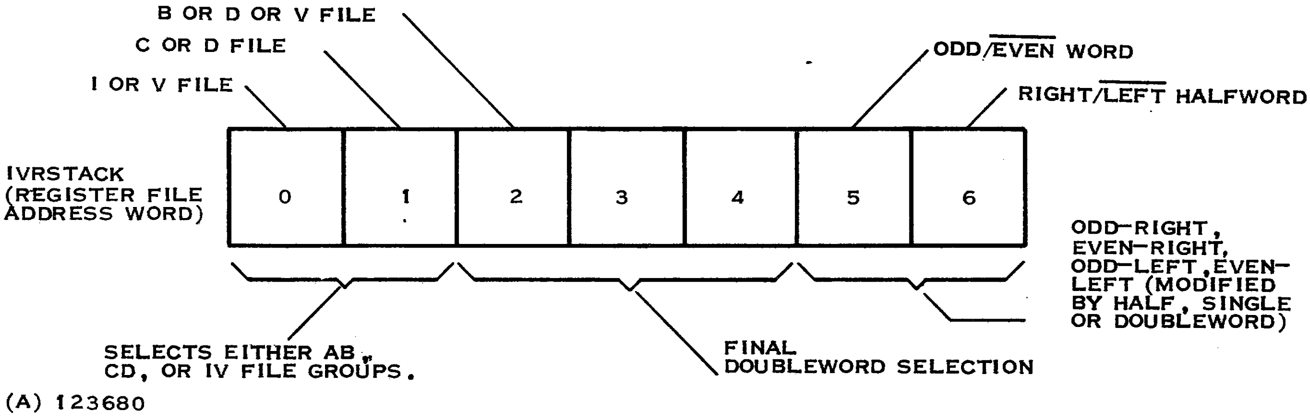

B OR D OR V FILE
C OR D FILE
I OR V FILE
ODD/$\overline{\text{EVEN}}$ WORD
RIGHT/$\overline{\text{LEFT}}$ HALFWORD
IVRSTACK (REGISTER FILE ADDRESS WORD)
0
1
2
3
4
5
6
ODD-RIGHT, EVEN-RIGHT, ODD-LEFT, EVEN-LEFT (MODIFIED BY HALF, SINGLE OR DOUBLEWORD)
SELECTS EITHER AB, CD, OR IV FILE GROUPS.
FINAL DOUBLEWORD SELECTION
(A) 123680

Figure B-4. IVRSTACK(0-6) Hardware Decode Bit Assignments

REGISTER FILE (IFQA - IFQV)

The register file is a storage area in the IPU that consists of forty-seven 32-bit registers. These registers are grouped into six groups of eight words (octets). The letters A, B, C, D, I and V differentiate the six octets. Octet A, however, contains only seven words, since the first word in that octet (address 00) cannot be addressed in the register file. Each I4FILE circuit board contains two bits of each word in the register file. These two bits are drawn from each halfword of the word so that bits 0 and 16 are on I4FILE(0), bits 1 and 17 are on I4FILE(1), etc. The primary block diagram description of the IPU describes the contents and functions of each register file octet.

INPUTS. Two input sources provide inputs to the register file: central memory through KCM, and any of the four AU pipes. The AU inputs (-IFAUEL, -IFAUER, -IFAUOL, -IFAUOR) allow storage of all process results into the register file so they may be retrieved without the need for a memory cycle to the MCU. The input from KCM (-IFSEL_(0-31)) loads file instructions, and is also used during a Load Details operation to load the register file. When the I4HDCORE circuit board on I4CTLMB is performing a CP Load Details, it generates each of the six input gates to the register file (IMCMTA, IMCMTB, IMCMTC, IMCMTD, IMCMTI, and IMCMTV) as its respective octet is read from memory into KCM and becomes available for input to the register file. For a Load File instruction, the IPU Level 3 Controller forces the I4CMREQ circuit board on I4CTLMB to fetch an octet from memory. Level 3 Controller generates input gates to the register file octets to load the memory octet.

STRUCTURE. The register file is composed of ECL type DF flip-flops. One input to each flip-flop is the -IFSEL bit from KCM. This input is enabled by the corresponding gate signal (IMCMTA - IMCMTV). The second input to the flip-flop is the -IFAUxx input. This input is gated by the respective IFAUG signal from the address decode. The system clock pulse (ICLOCK:1-15) supplies transfer pulses for entering data. In addition, master reset and preset functions are provided by the IMRE:4 and IMSE:4 signals, respectively.

OUTPUTS. The register file outputs are available to other select circuits within the I4FILE circuit board that choose the inputs to other registers in the IPU. In addition, the V octet can be directly transferred to I4CTLMB for vector parameter definition and hazard detection.

XR INPUT SELECT

The I octet of the register file contains seven words (1 through 7) that are used for instruction indexing. Three bits from the level 1 controller on I4CTLMB (IIISEL(0-2)) select the I octet word for transfer to the XR Register on I4PIPEMB. An octal decode circuit produces a separate gate signal for each of the seven words in the I octet. If all of the select bits are zeroes, no gate signal is produced so that only zeroes may be transferred to the XR register. (no indexing). The other seven combinations of the three bits gates produce for the respective words in the I octet (IFQI_(0-31)).

RO INPUT SELECT

The RO register is a 64-bit register located on the I4PIPE circuit boards. The input select to this register from the I4FILE circuit board provides either a halfword, a singleword, or a doubleword input to R0 from any octet in the register file. The selection of the proper input is performed in three steps: octet selection, doubleword selection, and singleword/halfword selection.

OCTET SELECT. Three control bits (IRROSEL(0-2)) from the level 3 controller on I4CTLMB determine which of the six register file octets are to be used for data to the R0 register. The three bits produce six gating signals (IRUA, IRUB, IRUC, IRUD, IRUI and IRUV) to choose one of the octets. The decode circuit views the three bits as a binary representation of an octal number. The output gating signals correspond to decodes 0 through 5 for octets A through V, respectively. Each of the six gating signals selects one of the octets from the register file. The output from this selection (-IRSEL_(0-31)) is then subjected to further refinement to the doubleword, singleword or halfword level.

DOUBLEWORD SELECTION. Two control bits (IRROSEL(3 and 4)) from I4CTLMB determine which doubleword from the selected register file octet contains data for entry data to the R0 register. In addition, an enable signal (IRENRU) allows the circuit to perform the selection and forward data to R0. If IRENRU is set, the following combinations of IRROSEL(3 and 4) select the indicated doubleword from the selected octet (-IRSEL_(0-31)):

IRROSEL(3)	(4)	Gate Signal	Doubleword Selected
0	0	IRU(1)	-IRSEL0(0-31) and -IRSEL1(0-31)
0	1	IRU(3)	-IRSEL2(0-31) and -IRSEL3(0-31)
1	0	IRU(5)	-IRSEL4(0-31) and -IRSEL5(0-31)
1	1	IRU(7)	-IRSEL6(0-31) and -IRSEL7(0-31)

The selected doubleword is then available to the singleword and halfword selection circuit as two singlewords. The singleword that originated from -IRSEL0, 2, 4, or 6 becomes the even output word, IRE0(0-31). The singleword that originated from -IRSEL1, 3, 5 or 7 becomes the odd output word, IRDRO1(0-31). IRDRO1(0-31) is also available to R0 to form the least significant singleword of a doubleword transfer.

SINGLEWORD/HALFWORD SELECTION. Four control signals from I4CTLMB determine the composition of the IRDRO0(0-31) output word to the RO register (refer to figure B-5). These control signals and their literal meaning are as follows:

• IROLTROR	Odd word, Left half To RO register Right word
• IRORTROR	Odd word, Right half To RO register Right word
• IRELTROR	Even word, Left half To RO register Right word
• IROLTROL	Odd word, Left half To RO register Left word

If the transfer to RO is a doubleword transfer, all of the above control signals will be zeroes. This condition transfers the input word IREO(0-31) directly to the IRDRO0(0-31) output word to become the most significant word input to the RO register (IRDRO1 (0-31) is the least significant word). For singleword entry into RO,, either the odd singleword or the even singleword may be transferred to the most significant word of RO. If the even word is to be transferred, (IREO(0-31)) the control signals remain at zeroes and the even word transfers directly to the output word as in a doubleword transfer. Gating signals to RO prevent IRDRO1(0-31) from transferring to RO during a singleword and halfword transfer. If the odd word is to be transferred to RO, IRORTROR and IROLTROL set. These signals gate the two halfwords of IRDRO1(0-31) to the corresponding halfword positions in the output word to RO.

All halfword transfers must be made through the right halfword (IRDRO0(16-31)) to RO. The singleword gating processes can also transfer the right half of either input word to the right half of the output word during halfword selections. The two remaining gate signals, IROLTROR and IRELTROR, transfer the left half of either the odd input word or the even input word, respectively, to the right halfword for output to RO.

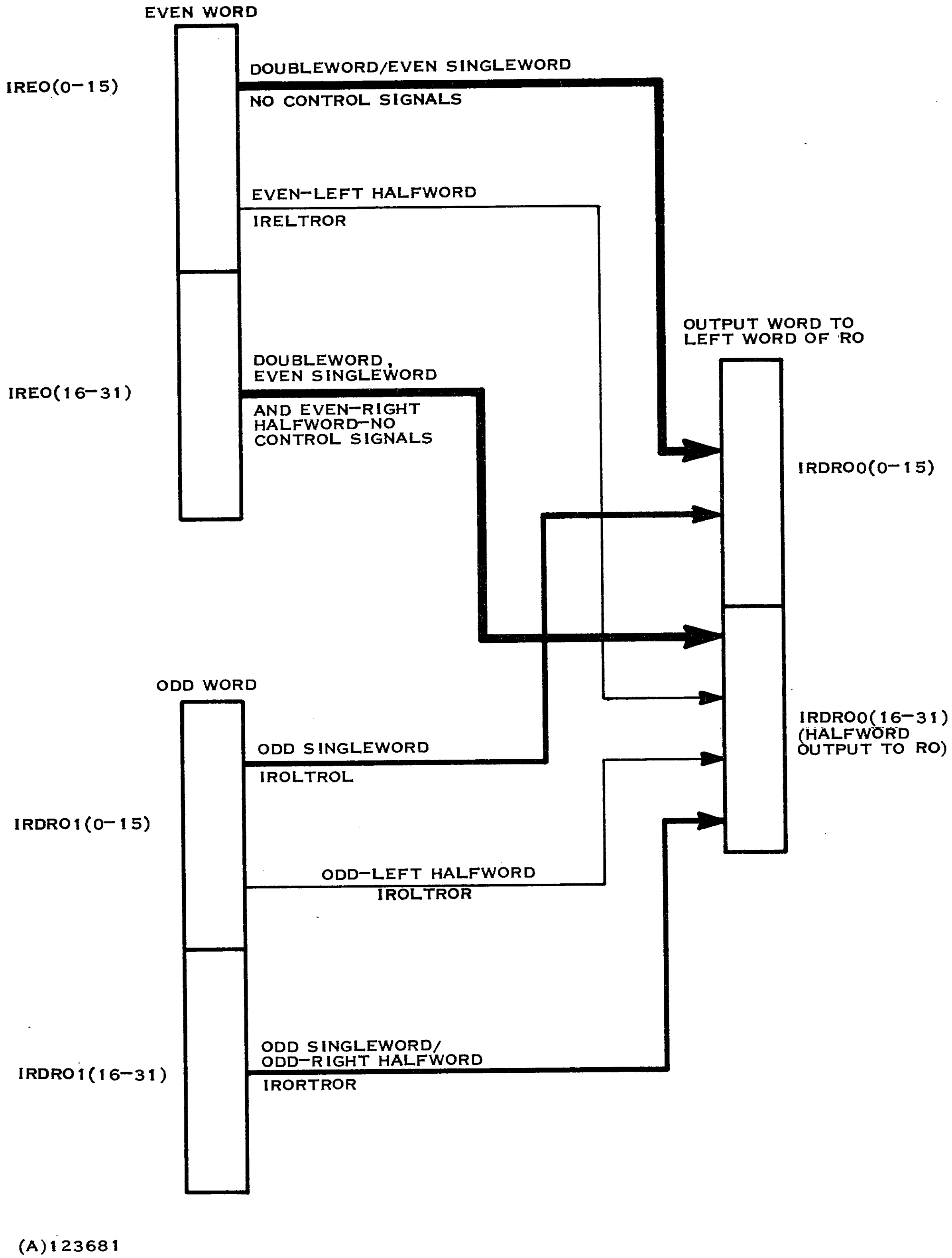


Figure B-5. Double/Single/Halfword Selection Paths

BR INPUT SELECT

The BR register is a 32-bit register, located on I4PIPEMB, that holds the base address for input to the address modification circuits. The input select gates transfer either the output from the A or B octets in the register file, the three vector address words from the V octet, or bits 32 through 63 from one of the AU pipes to the BR register. Seven control lines from I4CTLMB perform the transfer gating function.

A OR B OCTET SELECT. Four control bits (-IISELAB(0-3)) determine selection of either the A or the B octet in the register file for transfer to BR. Bit 0 of this group determines if the A or the B file will be used. When set it selects the A octet; when equal to zero, it selects the B octet. The remaining bits select one of the eight words in the selected octet. The signals are first inverted to produce "true" level signals before decoding. The decode circuit produces gate signals to the octet word that corresponds to the octal number represented by the three inverted control bits (ILSELAB(1-3)). The selected word is immediately available to the BR register.

V OCTET SELECTION. At the initiation of a vector instruction, words 1, 2 and 3 of the V octet (containing the starting addresses of the A, B and C vectors, respectively) are transferred to BR for possible address modification before being sent to the MBU. Two control signals (IIVTBR(0 and 1)) determine which of the three words will be transferred to BR at a particular moment. If both control bits are zero, no selection is made. Otherwise the bits select the following words:

01	IFQV1(0-31)
10	IFQV2(0-31)
11	IFQV3(0-31)

AU INPUT SELECT. During Push, Pull or Modify instructions, the AU modifies the memory pointer that is used for storage of results. In order to transfer the modified pointer to the MBU, a path through the IPU address development circuits is required. The BR input select provides this path. Only the right word (bits 32-63) of the AU output can be gated to the BR register. Four control bits (-IVAUSEL(0-3)) determine which AU pipe will supply the AU input (refer to the AU Word Select circuit description). To transfer the AU word to BR, a control signal (-IIAUPP) gates the word through the input select circuit to BR.

IF OCTET SELECT

The IF Octet Select circuit receives octet inputs from the register file, from KCM and from the two buffer files KA and KB. Five control signals from I4CTLMB determine which octet will be placed on the -IFSEL_(0-31) bus line. This bus line provides input data to the register file, to the memory storage select circuit and also to the AO and IR register input gating circuits. Control signals, derived from the type of instruction being processed, determine the destination of the -IFSEL_(0-31) data bits.

CONTROL SIGNAL DECODE. Five control signals, IRAOSEL(0-2), IMFSLDIS and IRDISKSL choose one of the input octets for output from the select circuit. IMFSLDIS, when high, disables the decode circuit. If this signal is low, IRAOSEL(0-2) selects one of the eight input octets as follows:

IRAOSEL(0-2)	Selected Octet	Gate Signal
000	IFQA_(0-31)	IBUA
001	IFQB_(0-31)	IBUB
010	IFQC_(0-31)	IBUC
011	IFQD_(0-31)	IBUD
100	IFQI_(0-31)	IBUI
101	IFQV_(0-31)	IBUV
110	ILK_(0-31)	ILSTORK
111	ILQKCM_(0-31)	ILSELKCM

The last two selections can be disabled by setting the remaining control signal, IRDISKSL, to a one level.

OUTPUT BUS PATHS. The -IFSEL_(0-31) bus provides inputs to other gating circuits on the I4FILE circuit board. Each of these inputs, however, has restricted use. The input path to the register file provides data from central memory through KCM, for storage in the register file. This path is used for Load File and Load File Multiple instructions, and a Load Details operation. The input path to the memory input select circuit may be used to store any of the input octets in memory. When storing maintenance details (-IMDTL_(0-31)), IMFSLDIS must disable selection of any octet to the -IFSEL_(0-31) lines. The bus path to the AO and IR register inputs may be used for octets from the register file, or from the memory interference files. This path for loading IR is normally only used for register file data.

CENTRAL MEMORY INTERFACE

The I4FILE circuit board contains the IPU's interface with central memory. This interface is asynchronous (not clocked). It reads octets from memory for instruction fetches and load details operations, and stores octets into memory for store file instructions and store details operations. The memory interface file, KCM, receives octets from memory; a memory storage gate circuit enables data octets to memory for storage.

MEMORY STORAGE GATE. Two source data signals may be relayed to central memory through the memory storage gate. The gate is enabled by -ICWRITE from I4CTLMB. When this signal is low, either of the two input data lines may transmit data to memory. Control signals from I4CTLMB ensure that when one of the data line inputs is active, the other input will be dormant. One input to this gate (-IMDTL_(0-31)) transmits details information from the other IPU circuit boards to central memory during a store details operation. The other input to this gate may be used for store details of the I4FILE circuit board registers, or for a store file or store file multiple instruction. Either case relays the contents of registers on the I4FILE circuit board to central memory for storage. When neither of these storage operations is being performed, the -ICWRITE signal goes high to enable the data bus to the MCU (IOCM_(0-31)) to be used for data transmission to KCM.

KCM MEMORY INTERFACE FILE. KCM is an octet buffer file that receives read data from the MCU through the bidirectional data bus (IOCM_(0-31)).

Addresses to the MCU to fetch data are produced on the I4ADDR circuit board. KCM has no clock input so that it may accept data from memory any time that it is enabled. One control signal (ICCMSTRB) from I4CTLMB enables all registers in the file. Inputs to the file are always octets. When ICCMSTRB is a "1", ILQKCM_(0-31) is loaded with the data on the memory data bus.

Three output buses from KCM channel the data to other components of the IPU. ILQKCM_(0-31) allows data transfer through the IF select circuit to the register file during store file and load details operations, and also allows storage of KCM contents into memory during a store details operation. ILQKCM_(0-31) also routes the contents of the file to the instruction register (IR) on the I4PIPEMB for indirect and execute instruction fetches. A second output from KCM (-ILQKCM_:2(0-31)) fans-out to other components of the IPU and is used exclusively for load details operations. The third output from KCM (-ILQKCM_:1(0-31)) transfers the contents of KCM to one of the two octet buffer files (KA or KB) so that KCM will be prepared to accept the next instruction octet from memory.

IMCLEAR from I4CTLMB initializes the file at the beginning of operations.

KA/KB OCTET BUFFER FILES. KA and KB are octet files that receive data input exclusively from KCM and hold that data for use by the IPU. During normal instruction processing, either KA or KB is initially loaded with an octet of instructions. While the instructions are being used from this first octet, the remaining file is loaded with the next instruction octet. The two files are used in this alternating fashion as long as instructions are in contiguous octet locations. I4CTLMB controls the input and output gating of KA and KB. IMCMTKA enables the next pulse from the ICLOCK to transfer the contents of KCM into the KA file. Similarly, IMCMTKB allows ICLOCK to transfer

KCM into KB. Two other control signals (-ILSELKA and -ILSELKB) gate the output from the corresponding file to the IPU circuit. The output from these files normally transfers to the instruction register (IR) on I4PIPEMB. However, during a store details the output is fed through the IF select circuit for transfer to memory.

AO INPUT SELECT

AO is a 64 bit register, located on I4PIPEMB, that relays operands to the MBU. The portion of the AO input select that is on the I4FILE circuit board supplies doubleword, singleword or halfword inputs to AO from the octets in the register file or from KCM, KA or KB. The select circuit supplies two 32-bit words to AO: one odd word (IRDAO1(0-31)) that becomes the least significant half of AO, and one even word (IRDAO0(0-31)) that becomes the most significant half of AO. Singlewords are loaded through the even word output; halfwords are loaded through the right half of the even word output (IRDAO0-(16-31)).

OCTET SELECT. Octet selection for AO is performed by the IFSEL octet select gate. Control bits IRA0SEL(0-2) choose the correct octet from the register file or from one of the memory interface files to supply input to AO.

DOUBLEWORD SELECTION. Two control bits from I4CTLMB (IRAOSEL-(3 and 4)) determine which doubleword from the selected octet will enter data to AO. The following combinations of these control bits select the indicated doubleword from the selected octet:

IRAOSEL(3)	(4)	Gate Signal	Doubleword selected
0	0	IBU(1)	-IFSEL0(0-31) and -IFSEL1(0-31)
0	1	IBU(3)	-IFSEL2(0-31) and -IFSEL3(0-31)
1	0	IBU(5)	-IFSEL4(0-31) and -IFSEL5(0-31)
1	1	IBU(7)	-IFSEL6(0-31) and -IFSEL7(0-31)

The selected doubleword is then available to the singleword and halfword selection circuit as two singlewords. The singleword that originated from -IFSEL0, 2, 4 or 6 becomes the even output word, IBEO(0-31). The singleword that originated from -IFSEL1,3,5 or 7 becomes the odd output word, IRDAO1(0-31). This odd word is also available to the AO register to form the least significant singleword of a doubleword entry.

SINGLEWORD/HALFWORD SELECTION. Four control signals from the I4CTLMB determine the composition of the IRDAO0(0-31) output word to the AO register. These control signals and their literal meaning are as follows:

●	IROLTAOR	Odd word, Left half To AO register, Right word
●	IRORTAOR	Odd word, Right half to AO register, Right word
●	IRELTAOR	Even word, Left half To AO register, Right word
●	IROLTAOL	Odd word, Left half To AO register, Left word

If the transfer to AO is a doubleword transfer, all of these control signals will be zeroes. This condition transfers the input word IBE0(0-31) directly to the output line to AO, while the odd word IRDAO1(0-31) transfers to the least significant half of AO.

For singleword entries into AO, either the odd singleword or the even singleword may be transferred to the most significant word of AO. If only the even word is to be transferred (IBEO(0-31)), the control signals remain at zeroes and the even word transfers to the output word as it does for a doubleword transfer. Gating signals prevent the other halfword from entering AO as part of a doubleword. If the odd word is to be transferred to AO, IRORTAOR and IROLTAOL set. These signals gate the two halfwords of IRDAO1(0-31) to the corresponding halfword positions in the output word to AO.

All halfword transfers to AO must be made through the right halfword (IRDAO0-(16-31). The singleword gating processes allow the right half of either input word to be transferred to the right half of the output word. The two remaining gate signals, IROLTAOR AND IRELTAOR, enable the transfer of the left half of either the odd word or the even word input, respectively, to the right halfword for output to AO.

In addition to supplying inputs to the left singleword of AO, this selection circuit also provides inputs from the register file to the instruction register (IR) on I4PIPEMB. The output to the IR input select (-IRDAO0(0-31)) is the complement of the word supplied to AO. Entries from the memory interface octets to IR are performed more efficiently by a different selection network, and are not loaded through this gate.

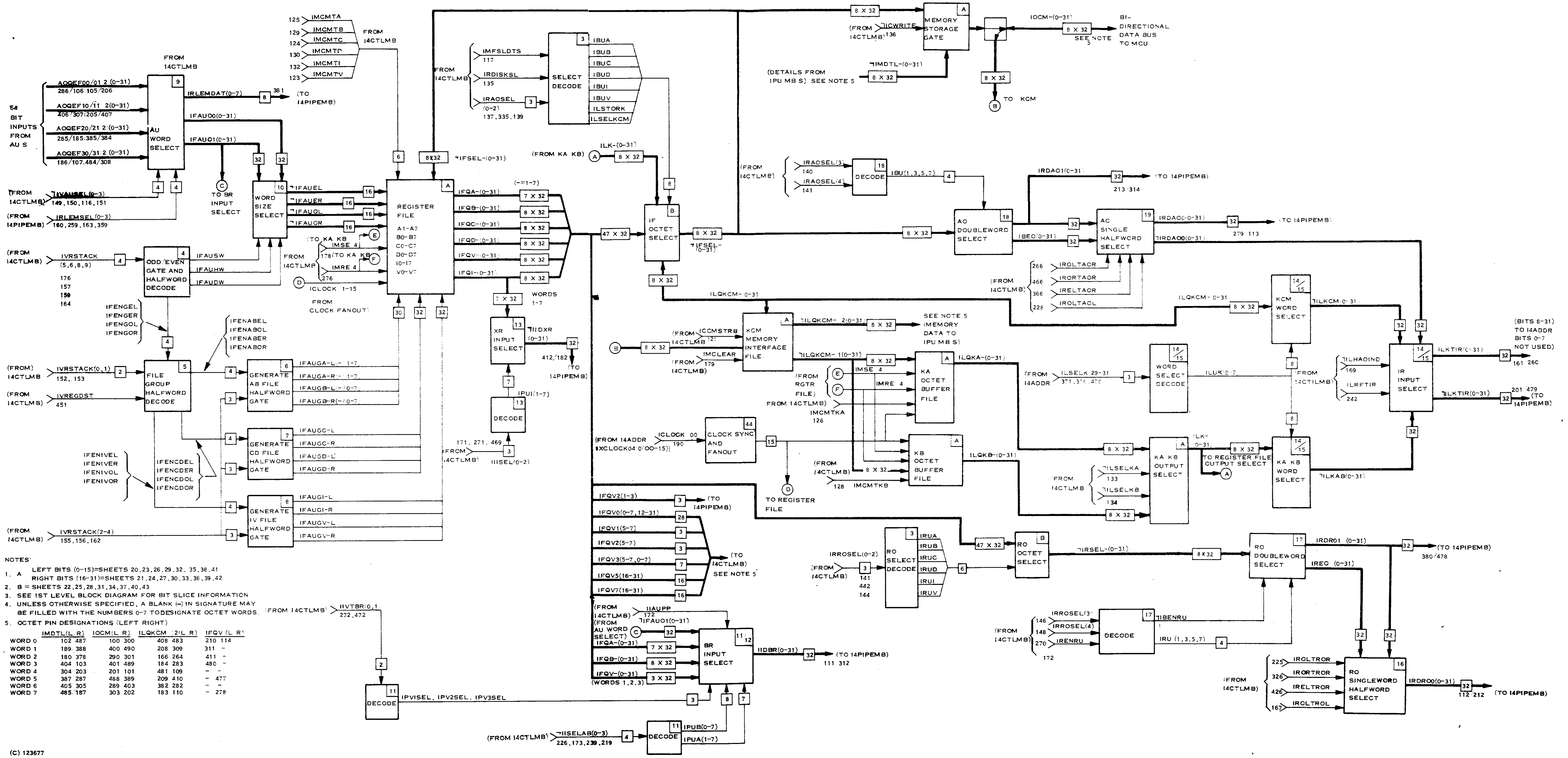
IR INPUT SELECT

The instruction register (IR) is a 32-bit register that receives input instruction words for processing by the IPU. The input selection to this register contained on the I4FILE circuit board chooses a word from KCM, KA, KB, or the register file for entry into IR. During normal processing, entry into IR is from either KA or KB. KCM transfers to IR when a hard core address is required from central memory or if an indirect address is contained in memory and is not resident in the IPU. The register file may also supply indirect address responses through the AO input selection, but only for the first indirect address. Subsequent indirect address fetches must be made to central memory.

KCM, KA AND KB WORD SELECT. Three control bits from the I4ADDR circuit board select a word from KCM, and from KA or KB for input to the final word selection gate. These three bits (ILSELK(29-31)) enter an octal decode circuit that produces one of eight possible input gates (ILUK(0-7)) depending upon the value of the input code. The ILUK(0-7) signal that is enabled transfers the corresponding word from ILQKCM_(0-31) and ILK_(0-31) to the final IR input gate.

FINAL WORD GATING. Three data words are available for final selection to IR: -IRDAO0(0-31) containing an input from the register file, -ILKCM(0-31) containing a word from central memory through KCM, and -ILKAB(0-31) that supplies inputs from either the KA or KB buffer files. Two control bits (-ILHAOIND and ILRFTIR) from I4CTLMB select which input word is to be transferred to IR. The first of these signals is dominant. Whenever -ILHAOIND is a zero level the input word from KCM will be transferred to IR. If this control line is high, ILRFTIR (register file to IR) transfers the register file input to IR when high and gates the KA/KB input word to IR when low.

Figure B-6 is the block diagram of the I4FILE circuit board.



NOTES:

1. A LEFT BITS (0-15)=SHEETS 20,23,26,29,32, 35,38,41
   RIGHT BITS (16-31)=SHEETS 21,24,27,30,33,36,39,42
2. B = SHEETS 22,25,28,31,34,37,40,43
3. SEE 1ST LEVEL BLOCK DIAGRAM FOR BIT SLICE INFORMATION
4. UNLESS OTHERWISE SPECIFIED, A BLANK (–) IN SIGNATURE MAY BE FILLED WITH THE NUMBERS 0-7 TODESIGNATE OCTET WORDS.
5. OCTET PIN DESIGNATIONS (LEFT RIGHT)

	IMDTL(L R)	IOCM(L R)	ILQKCM 2(L R)	IFQV (L R)
WORD 0	102 487	100 300	408 483	210 114
WORD 1	189 388	400 490	208 309	311 –
WORD 2	180 378	290 301	166 264	411 –
WORD 3	404 103	401 489	184 283	480 –
WORD 4	304 203	201 101	481 109	– –
WORD 5	387 287	488 389	209 410	– 477
WORD 6	405 305	289 403	382 282	– –
WORD 7	485 187	303 202	183 110	– 278

(C) 123677

Figure B-6. I4FILE Circuit Board Block Diagram

INTRODUCTION - BLOCK DIAGRAM SYMBOLS

The block diagram included in this circuit board description condenses the information contained in the logic diagrams to a one sheet representation. The diagram contains all control signatures that are inputs to or outputs from the circuit board. In addition many of the key control lines internal to the board are illustrated. Since the circuit board is a control board, no data lines are found on the board. Other information contained on the block diagram is illustrated in figure B-7, and includes:

- Sheet Number - Location of the logic diagram for the depicted function within the logic set for the circuit board. Multiple sheets are referenced with a letter that is explained in the notes on the diagram.
- Pin Number - Circuit board input/output pin that carries the indicated signal. When more than one pin number appears, the pin for the most significant bit appears first, or if different signals are included on one line, the pin numbers appear in the order of the signatures listed on the line. Large quantities of pins are in tabular form in the notes on the diagram sheet.
- Line size - Numbers on all lines indicate the number of bits represented by that particular line. A line without a number represents only one bit.

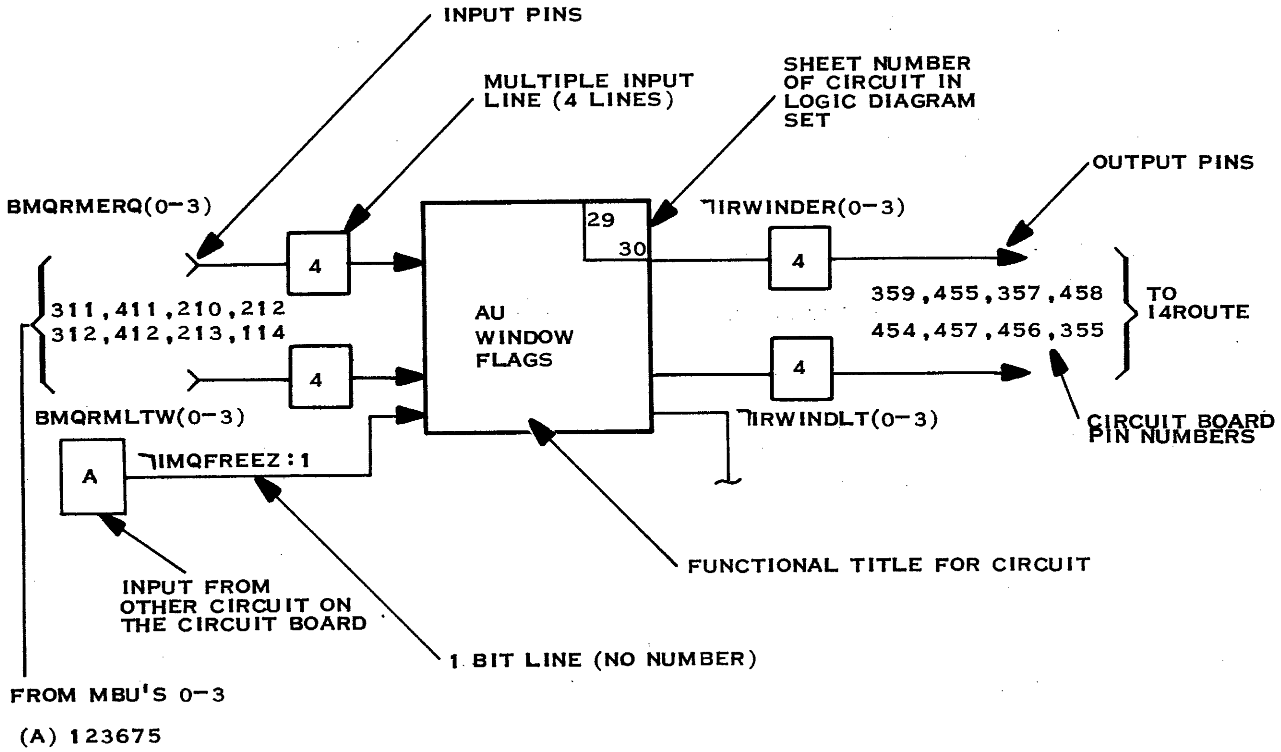


Figure B-7. Key to Symbols - I4CMREQ Circuit Board Block Diagrams

I4CMREQ CIRCUIT BOARD

The I4CMREQ circuit board is a control circuit board that contains the Look Ahead Controller and the Central Memory Requester for the IPU. In addition the circuit board contains a hazard determination for the KA and KB buffer files, flag flip-flops for the early and late window signals from the AU ROM, plus unit register fanin, and details fanin-fanout circuits. These circuits and their interrelations are illustrated on the block diagram at the end of this section (figure B-14). The following paragraphs describe the component circuits on the I4CMREQ circuit board and the control signals produced to coordinate instruction octet fetching for the IPU.

INTERFACE SIGNALS

Table B-4 defines the input signals to the I4CMREQ circuit board. Table B-5 defines the output signals from the I4CMREQ circuit board.

Table B-4. I4CMREQ Circuit Board Input Signals

Signature	Origin	Function
BMQRMERW(0-3)	AU ROM in MBU's 0-3	Indicates that the divide in pipe 0-3 is at a point so that a divide of the same group in level 3 could reach the AU in time to save divide initialization time by overlapping (Early Window). Includes memory fetch time.

Table B-4. I4CMREQ Circuit Board Input Signals (Continued)

Signature	Origin	Function
BMQRMLTW(0-3)	AU ROM in MBU's 0-3	Indicates that the divide in pipe 0-3 is at a point so that a divide of the same group in level 3 could reach the AU in time to save divide initialization time by overlapping (Late Window). Does not include memory fetch time.
ICCMFUL	I4HDCORE	Indicates to CMR that KCM (memory interface buffer) contains an octet of instructions for memory.
ICLOCK:00	CLOCKFAN	System clock. Originates as $XLOCK02:0(005). Clock pulses for control circuits.
-ICOABSY	I4HDCORE	Indicates to CMR that the OA address register to the MCU contains a valid instruction address that has not been transmitted.
ICQAREX	I4HDCORE	Arithmetic exception flag. Transferred during store details only.
ICQIPIOP	I4HDCORE	IPU illegal op code flag. Transferred during store details only.
ICQIPPAE	I4HDCORE	Parity error flag. Transferred during store details only.
ICQIPPRV	I4HDCORE	Memory protect violation flag. Transferred during store details only.
ICRCMPV	I4HDCORE	Read protect violation indication to CMR: last memory request produced a protect violation.
IHQOA:1 (8-31)	I4ADDR	Contents of the OA register for transfer as Unit Register data.

Table B-4. I4CMREQ Circuit Board Input Signals (Continued)

Signature	Origin	Function
IIQL1ACT	I4PIPTOP	Level 1 active flag: used by look ahead controller to determine if a PB target is in level 0.
IIQS(6)	I4PIPTOP	Level 1 controller state 6 flag; store file instruction at level 2.
-ILLAEQZB(0-3)	I4ZHAZ(0,4, 8,12)	Indicates that the look ahead octet is to be modified by an instruction in pipes 0 through 3.
-ILQKCM4(6)	I4FILE(6)	Load details input to early and late window flags.
-ILQKCM7(4-7)	I4FILE(4-7)	Load details inputs for I4CMREQ circuits.
ILQLC(0-7)	I4ADDR	Contents of the look ahead counter: used by the look ahead controller to determine the position of an upcoming branch.
ILQPA(29-31)	I4ADDR	Word address contained in the PA register. Used by look ahead controller to determine octet boundary (PA = 7). Also transferred as unit register data to the PP.
ILPAEBA(0,1)	I4ADDR	Indicates to the look ahead controller that the address in BA is equal to the present address octet in PA.
(-)ILPAEN	I4PIPTOP	Enables the look ahead controller to transfer a new instruction into the IR register.
ILPAENSB(1,2)	I4PIPTOP	Enables the look ahead controller to transfer a new instruction into the IR register.
ILZBVSPA	I4PIPTOP	Indicates that the present address octet will be modified by an instruction in one of the IPU pipes.
-IMHCREQ	I4HDCORE	Indicates to CMR that a maintenance command is being performed (hard core request).

Table B-4. I4CMREQ Circuit Board Input Signals (Continued)

Signature	Origin	Function
IMLDTL:2	I4HDCORE	Enables the details count decoder to generate the load details gates to the circuits on I4CMREQ.
IMOCTR:2(0-3)	I4HDCORE	Provides the sequencing count used during load and store details operations.
-IMQFREEZ:1	I4HDCORE	Disables IPU operation due to run bit being cleared or other abnormality.
IMRE:2	I4HDCORE	Master reset to I4CMREQ circuits.
IMSDTL:2	I4HDCORE	Enables the details count decoder to generate the store details gates to the circuits on I4CMREQ.
IMSE:2	I4HDCORE	Master preset to I4CMREQ circuits.
IMUREN(0-3)	I4HDCORE	Four bit code that selects one of seven inputs from I4CMREQ for transfer as unit register data to the PP.
-IPL2BLK	I4PIPTOP	Level 2 Block (not) - enables the look ahead controller to transfer a new instruction into the IR register.
IPQDCPB	I4INFACE	Prepare to Branch instruction at level 2: used by look ahead controller to locate the PB target.
IPQL2ACT	I4PIPTOP	Level 2 active flag: used by look ahead controller to determine if a PB target is in level 0.
-IPR2(4-7)	I4PIPE(4-7)	Contents of the level 2 R register (R2): used by look ahead controller to determine if a PB target is at level 0.
IRALINCM	I4ROUTE1	Alpha in Central Memory: Indicates that the desired operand is not in the register file.

Table B-4. I4CMREQ Circuit Board Input Signals (Continued)

Signature	Origin	Function
-IRARELA	I4LVL3	Address in the AR register in level 3 is contained in the look ahead octet.
-IRAREPA	I4LVL3	Address in the AR register in level 3 is contained in the presently executing octet.
IRBRHAZ	I4LVL3	A branch instruction at level 3 has encountered a hazard in the pipe and is waiting for it to clear.
-IRBRTLA	I4LVL3	Indicates to the look ahead controller that the target instruction of a branch at level 3 is contained in the look ahead octet.
-IRBRTOA	I4LVL3	Indicates to the look ahead controller that the target instruction of a branch at level 3 will have to be fetched from memory.
-IRBRTPA	I4LVL3	Indicates to the look ahead controller that the target instruction of a branch at level 3 is contained in the currently executing octet.
-IRBRTP1	I4LVL3	Indicates to the look ahead controller that the target instruction of a branch at level 3 is contained in level 1 of the IPU.
-IRBRTP2	I4LVL3	Indicates to the look ahead controller that the target instruction of a branch at level 3 is contained in level 2 of the IPU.
-IRDLBNT	I4LVL3	Dual and Branch not taken: While operating in dual mode, the expected branch was either skipped over, or not executed due to the lack of a required condition.

Table B-4. I4CMREQ Circuit Board Input Signals (Continued)

Signature	Origin	Function
-IREXIND	I4LVL3	An indirect address, or an Execute instruction is at level 3 and requires a new octet from memory to continue processing.
-IRINHAZ	I4LVL3	An instruction hazard has occurred and the corresponding instruction octet must be refetched to recover to changed information.
-IRLCLBR	I4LVL3	Local Branch: the branch instruction at level 3 references a target instruction that is resident in the IPU octets.
-IRLFREQ	I4VECLAS	Load file request: Data octets are to be transferred from memory to the register file.
IRLLXFER	I4LVL3	A Load Look Ahead instruction is at level 3.
-IRPBXFER	I4LVL3	A Prepare to Branch instruction is at level 3 and is enabled.
IRPAC3:1	I4ROUTE	Level 3 Path Ahead Clear - Enables the look ahead controller to transfer a new instruction into the IR register.
IRQDCPB	I4INFACE(1)	Prepare to Branch instruction at level 3: used by the look ahead controller to locate the PB target.
IRQP3(29-31)	I4PIPE(7)	Word address of the instruction in level 3: used in look ahead controller to determine position of PB target.
IRQTARGT	I4LVL3	PB target at level 3 flag.
-IRR3(4-7)	I4PIPE(4-7)	Contents of the level 3 R register. (R3): used by the look ahead controller to determine if a PB target is at level 0.

Table B-4. I4CMREQ Circuit Board Input Signals (Continued)

Signature	Origin	Function
-IRSFREQ	I4VECLAS	Store file request: Data octet(s) are to be transferred from the register file to memory.
-IRTGTFL	I4LVL3	Target fail: An expected branch was skipped over or failed to branch when it reached level 3.

Table B-5. I4CMREQ Circuit Board Output Signals

Signature	Destination	Function
ICCMTFIL	I4VECLAS/ I4MISC	Transfers an octet from KCM to the register file input routing circuit for storage in the register file.
ICCMTIR	I4PIPTOP	Enables transfer of a selected instruction word from KCM to the IR register.
ICCUEMPY	I4HDCORE	Indicates that the memory request queue is empty.
ICFL	I4HDCORE	Indicates that the returning octet is intended for the register file.
ICINCOP	I4HDCORE	Increment signal for the request queue output pointer. Enables protect violation flag due to ICFL or ICIR (on I4HDCORE).
ICIR	I4HDCORE	Indicates that the returning octet is intended to supply an instruction to the IR register.
ICLDO(11)	I4HDCORE	Load details count 11.
ICPRVO	I4HDCORE	The currently selected output queue location encountered a protect violation when trying to access data from memory. Enables protect violate flag on I4HDCORE due to ICFL or ICIR.
ICQKAFUL	I4PIPTOP/ I4LVL3	The KA buffer contains an instruction octet that can supply instructions to IR.

Table B-5. I4CMREQ Circuit Board Output Signals (Continued)

Signature	Destination	Function
ICQKAHAZ	I4PIPTOP/ I4LVL3	An instruction in the pipe(s) will modify the data contained in the KA buffer.
ICQKAPRV	I4PIPTOP	The data in the KA buffer resulted from a fetch that violated memory protect, and is therefore invalid data.
ICQKBFUL	I4PIPTOP/ I4LVL3	The KB buffer contains an instruction octet that can supply instructions to IR.
ICQKBHAZ	I4PIPTOP/ I4LVL3	An instruction in the pipe(s) will modify the data contained in the KB buffer.
ICQKBPRV	I4PIPTOP	The data in the KB buffer resulted from a fetch that violated memory protect, and is therefore invalid data.
ICQKRTAG	I4PIPTOP/ I4LVL3	Selects either the KA or the KB buffer to supply instructions to the IR register.
-ICRDACK	I4VECLAS/ I4LVL3	Indicates that CMR can accept a new read request to obtain data from memory.
-ICURDATA(0-7)	I4HDCORE	Unit register byte to be transferred to the PP.
ICWACK	I4VECLAS/ I4PIPTOP	Indicates that CMR can accept either a read or a write request, because the request queue is empty.
ICWACK1	I4HDCORE	Fanout of ICWACK.
ILARTBA(0,1)	I4ADDR(0,1)	Transfers the contents of the AR register to the BA register.
-ILARTLA	I4ADDR	Transfers the contents of the AR register to the LA register.

Table B-5. I4CMREQ Circuit Board Output Signals (Continued)

Signature	Destination	Function
-ILARTLC	I4ADDR	Transfers the contents of the AR register to the LC counter.
-ILARTOA	I4ADDR	Transfers the contents of the AR register to the OA register.
-ILARTPA	I4ADDR	Transfers the contents of the AR register to the PA register.
-ILBATLA	I4ADDR	Transfers the contents of the BA register to the LA register.
-ILBATOA	I4ADDR	Transfers the contents of the BA register to the OA register.
-ILBATPA	I4ADDR	Transfers the contents of the BA register to the PA register.
-ILCNTBA	I4ADDR	Transfers LA + 8 to the BA register.
ILDECLC(0,1)	I4ADDR	Decrement pulse to the LC counter.
-ILDTARGT	I4PIPTOP	Indicates that a PB target branch is in level 1.
ILDUAL	I4LVL3	Dual: A branch has encountered a hazard at level 3 and is holding for it to clear; therefore, the look ahead controller retains the current octet while fetching a new look ahead octet containing the target instruction of the branch.
-ILGBA	I4ADDR	Enables loading of the BA register.
-ILGLA	I4ADDR	Enables loading of the LA register.

Table 4-2. I4CMREQ CIRCUIT BOARD OUTPUT SIGNALS (Cont.)

Signature	Destination	Function
-ILGLC	I4ADDR	Enables loading of the LC counter.
-ILGOA	I4ADDR	Enables loading of the OA register.
-ILGPA	I4ADDR	Enables loading of the PA register.
-ININCLA	I4ADDR	Increment LA: Adds 8 to the address currently contained in the LA register.
-ININCPA	I4ADDR	Increment PA: Adds 1 to the address currently contained in the PA register.
-ILINHAZ	I4ADDR	Transfers the address in P3 into the OA, LA and PA registers to recover from an instruction hazard.
-ILLATBA	I4ADDR	Transfers the contents of the LA register to the BA register.
-ILLATOA	I4ADDR	Transfers the contents of the LA register to the OA register.
ILLOADIR:1 -ILLOADIR:1 -ILLOADIR:2	I4PIPTOP I4PIPTOP I4PIPEMB	Transfers a new instruction word into the IR register.
ILPATBA(0,1)	I4ADDR	Transfers the contents of the PA register to the BA register.
ILPBEN	I4LVL3	Prepare to Branch instruction enabled.
-ILP3TBA	I4ADDR	Transfers the contents of the P3 register to the BA register.
-ILP3TOA	I4ADDR	Transfers the contents of the P3 register to the OA register. (Not used)
-ILR3TLC	I4ADDR	Transfers the contents of the R3 register to the LC counter.
IMCMTKA	I4FILE	Transfers the octet in KCM to the KA buffer.

Table B-5. I4CMREQ Circuit Board Output Signals (Continued)

Signature	Destination	Function
IMCMTKB	I4FILE	Transfers the octet in KCM to the KB buffer.
-IMDTL4(6)	I4FILE(6)	Store details data lines to memory bus.
-IMDTL7(4-7)	I4FILE(4-7)	Store details data lines to memory bus.
-IRWINDER(0-3)	I4ROUTE2	Early window flag for AU(0-3). Indicates that if a divide at level 3 is in the same group as a divide that is currently in the pipe, the new divide can reach the AU in time to save divide initialization time by overlapping. This window allows for memory fetch time. This signal is delayed one clock from the input signal BMQRMERW(0-3) that produced it.
-IRWINDLT(0-3)	I4ROUTE2	Late window flag for AU(0-3). Indicates that if a divide at level 3 is in the same group as a divide that is currently in the pipe, the new divide can reach the AU in time to save divide initialization time by overlapping. This window does not allow for memory fetch time. This signal is delayed one clock from the input signal, BMQRMLTW(0-3) that produced it.
IVDMZERO(0-7)	not used	Fixed logic zero signals.

LOOK AHEAD CONTROLLER

The look ahead controller produces gating and control signals required to load each of the address registers on the I4ADDR circuit board and the IR register in level 1 of the IPU. The controller monitors the status of instruction octets in the IPU, and by loading the address registers at the proper time, assures that instructions will be available to IR with the minimum possible delay. During normal instruction sequencing, the controller loads the address of the look ahead octet (the next sequential octet after the current octet) into the OA register, so that the Central Memory Requester (CMR) may fetch that octet from memory and place it into the look ahead buffer (KA or KB). If a branch enters the pipe that has been preceded by a LLA or PB instruction, the controller fills the pipe following the branch instruction with instructions from the target octet of the branch. When the branch occurs, the instructions in the branch path will be immediately available. A branch that is not preceded by a PB or LLA instruction creates a delay by requiring a new memory fetch.

The following paragraphs describe the operation of the look ahead controller with reference to the flow chart of the controller logic that follows this description. The paragraphs follow the same order as the logic flow through the chart, and explain the major decision paths that are possible within the controller. Table B-6 lists the register transfers that the look ahead controller initiates.

CONTROLLER TIMING

The look ahead controller is composed of combinational logic, and as such, has no timing chain, sequence of events, or formal states. All of the question blocks illustrated in the flow chart are examined simultaneously during each control cycle to enable only one path through the controller. When the control clock pulse occurs, all of the action blocks on that enabled path are

Table B-6. Look Ahead Controller Transfers Index

TRANSFER	REFERENCE SIGNALS
AR→BA	¬ILBLK (18), (27)
AR→LA	¬ILBLK (0), (2), (4), (20), (29), (30), (45)
AR→LC	¬ILBLK (13)
AR→OA	¬ILBLK (0), (20), (30), (45) (also Load/Store File from CMR)
AR→PA	¬ILBLK (0), (2), (3), (4), (22)
BA→LA	¬ILBLK (7), (9), (10), (33), (34), (37), (41), (42)
BA→OA	¬ILBLK (9), (10), (33), (37), (41)
BA→PA	¬ILBLK (6), (7), (9), (38)
LA→BA	¬ILBLK (46)
LA+8→BA	¬ILBLK (47)
LA+8→LA	¬ILBLK (14), (15), (16), (19), (28), (31), (36), (39), (43), (44)
LA+8→OA	¬ILBLK (14), (15), (16), (19), (28), (31), (39), (43), (44)
LC-1→LC	¬ILBLK (38), (40), (48)
LOAD IR	¬ILBLK (11), (15), (17), (22), (25), (29), (30), (31), (38), (40), (48)
PA+1→BA	¬ILBLK (22), (38)
PA+1→PA	¬ILBLK (11), (15), (17), (25), (29), (30), (31), (40), (48)
P3→BA	¬ILBLK (13)
P3→LA	¬ILBLK (1)
P3→OA	¬ILBLK (1)
P3→PA	¬ILBLK (1)
R3→LC	¬ILBLK (18), (27)

executed simultaneously. This type of timing means that action blocks upstream from other decision blocks in the flow chart do not affect the decision block. Also, since all actions occur simultaneously, all action statements refer to conditions at the start of the control cycle. For example, ILBLK(22) (sheet 3 of the flow chart) transfers AR to PA and PA + 1 to BA. This statement does not mean that BA then contains AR + 1, but rather, it contains the address contained in PA at the beginning of the control cycle plus an increment.

LOOK AHEAD CONTROLLER TERMS

The following terms and their definitions are essential to understanding the flow charts and the look ahead controller discussion:

- Branch - An instruction that causes the next instruction to be accessed from a non-sequential location, creating the necessity for the controller to fetch a new look ahead octet.
- Buffer - The KA and KB octet buffers on the I4ADDR circuit board. The current buffer is that buffer from which instructions are currently being drawn. The look ahead buffer is the non-current buffer containing the next octet of instructions to be used.
- CMR - Central Memory Requester: The controller that initiates requests to memory for instruction octets.
- Dual - A mode of operation of the controller that is entered when a non-targeted branch encounters a hazard at level 3, and therefore, the terms of the branch cannot be determined. The controller saves the current octet, and fetches a new look ahead octet that contains the target instruction of the branch.
- Flag Full - An internal controller flag that indicates that the target instruction is in the current buffer and the target branch is in the

look ahead buffer. Therefore, the branch will be to an instruction that has previously been in the current buffer.

- Flag 4 - An internal controller flag that indicates that a target branch is in the IPU pipe (levels 1-3).

- Flag 12 - An internal controller flag that indicates that the target instruction of a branch either has been requested from memory or is resident in the IPU buffers.

- LA Ordered - An internal controller flag that indicates that a normal look ahead octet (present address +8) has been requested from memory, or is resident in the IPU buffers. The name is derived from the fact that the Look Ahead octet has been ORDERED from memory.

- LLA - Load Look Ahead instruction: An instruction that prepares the controller to branch back to a previous point in a program. When the LLA instruction reaches level 3, the address of the LLA contained in P3 is the target address of the branch and the developed portion of the LLA in the AR register designates the number of instructions between the LLA and the branch instruction.

- PB - Prepare to Branch instruction: An instruction that prepares the controller to branch to a predetermined address. When the PB instruction reaches level 3, the developed portion of the instruction in AR contains the target address of the branch, and the R3 register designates the number of instructions between the PB and the branch instruction (15 instructions maximum).

- Target Branch - The branch instruction that is the object of a PB or LLA instruction. May also be referred to as PB Target.

- Target Fail - An internal controller flag that indicates that when a targeted branch reached level 3, the branch was not taken or the instruction was skipped. The controller must then reconstruct the original instruction sequence.
- Target Instruction - The instruction that is the object of any branch instruction.

FLOW CHARTS

The look ahead controller flow charts that follow this description may be used as both a theory learning tool and a maintenance tool. Each question or action block contains a brief phrase that describes the function(s) of that particular block. Beside the block is the exact signature of the signal that is being examined (question block) or produced (action block). Along with the signature is a set of tagging information that designates the origin or monitoring point for that signal in the hardware, and the sheet in the logic diagrams where that signal originates. All sheet references are to sheets within the logic set for the 14CMREG circuit board. Figure B-8 illustrates the tagging information included with each signature on the flow charts. Figure B-9 is the Look Ahead Controller Flowchart.

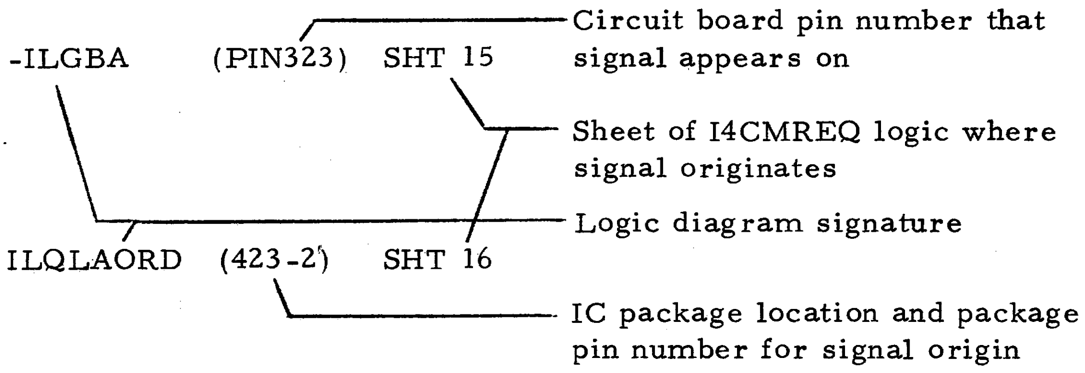
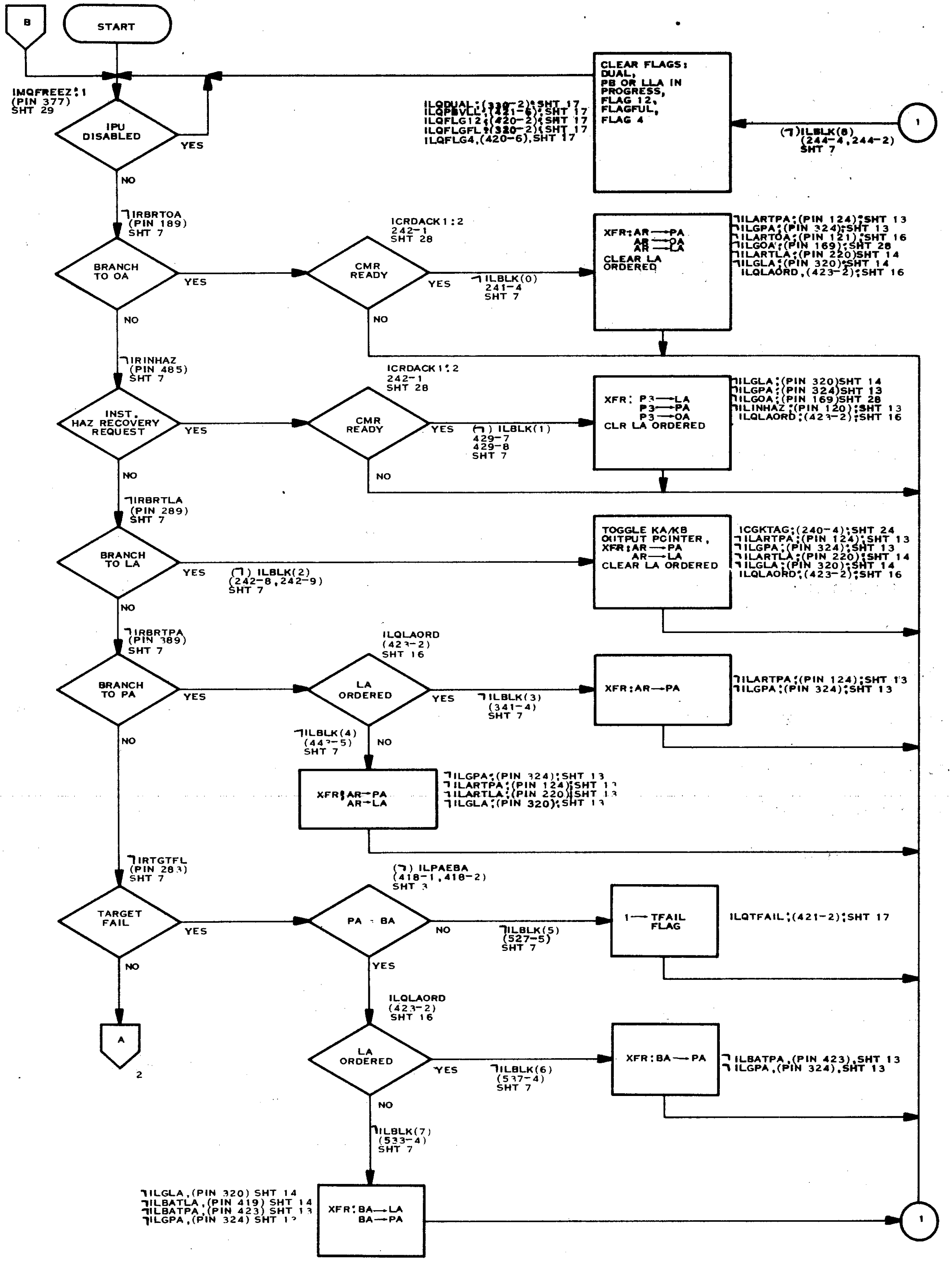
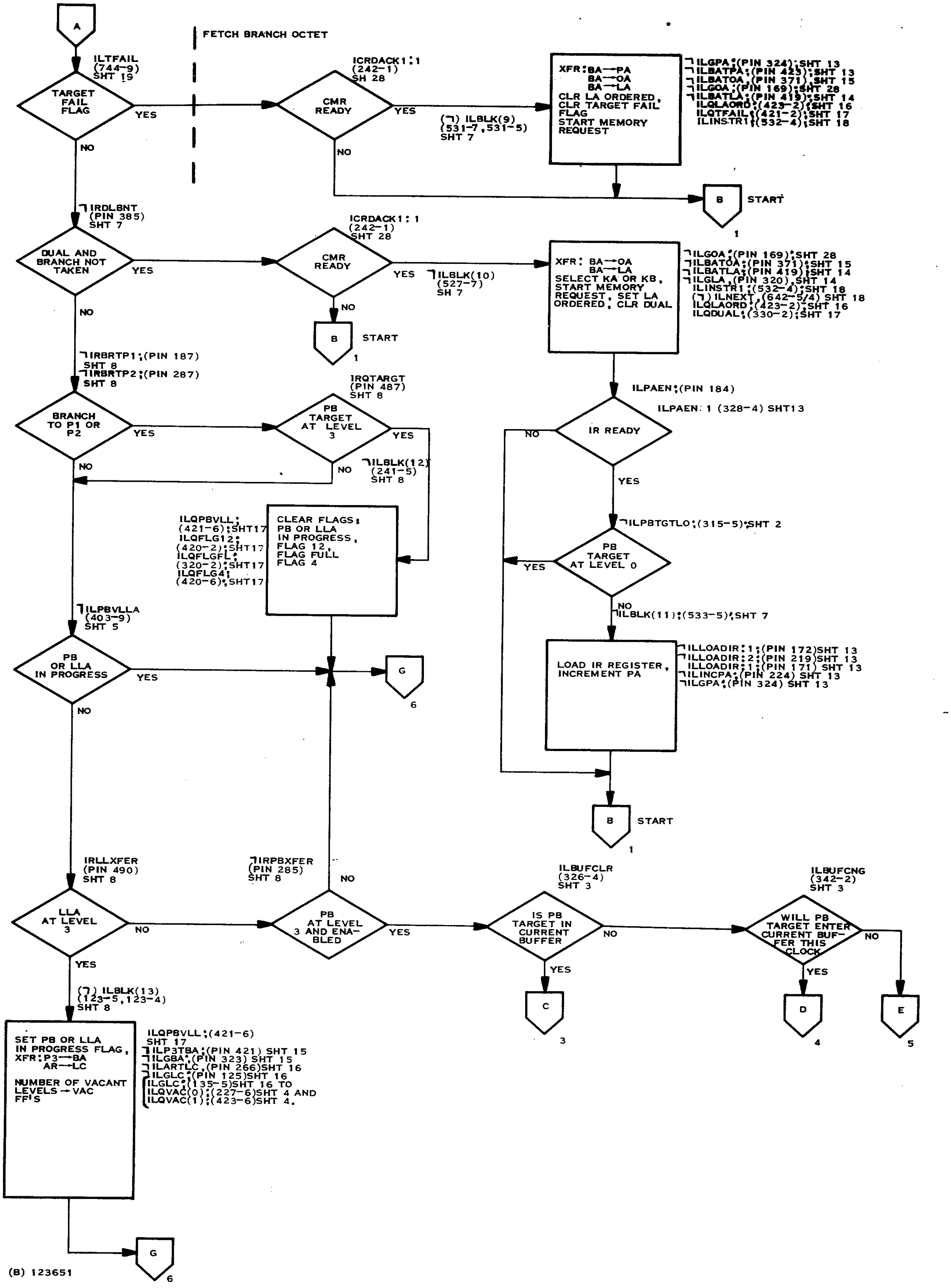

-ILGBA (PIN323) SHT 15
ILQLAORD (423-2) SHT 16
Circuit board pin number that signal appears on
Sheet of I4CMREQ logic where signal originates
Logic diagram signature
IC package location and package pin number for signal origin

Figure B-8. Flowchart Tagging Information



(B) 123650

Figure B-9. Look-Ahead Controller Flowchart (Sheet 1 of 8)



(B) 123651

Figure B-9. Look-Ahead Controller Flowchart (Sheet 2 of 8)

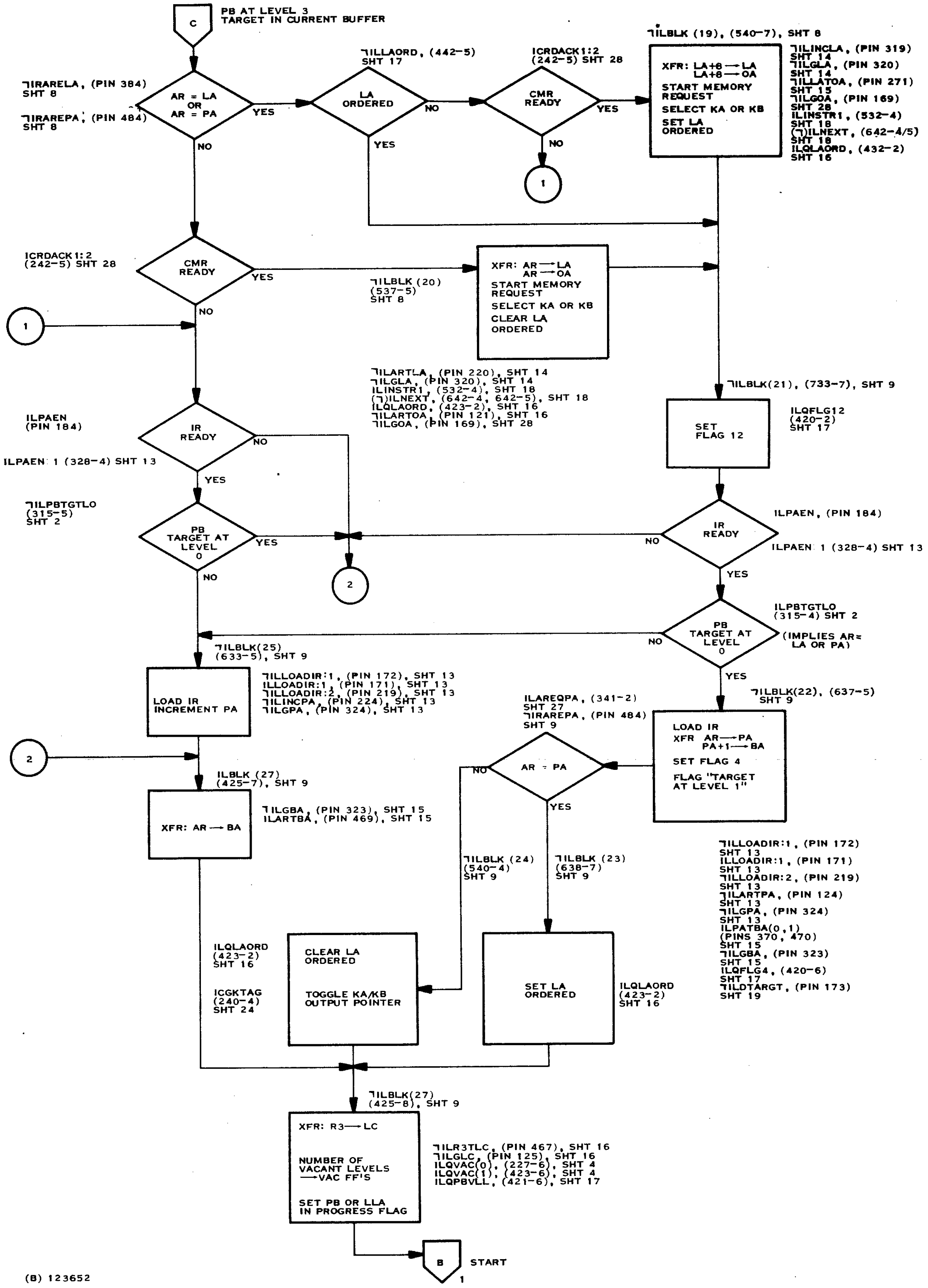
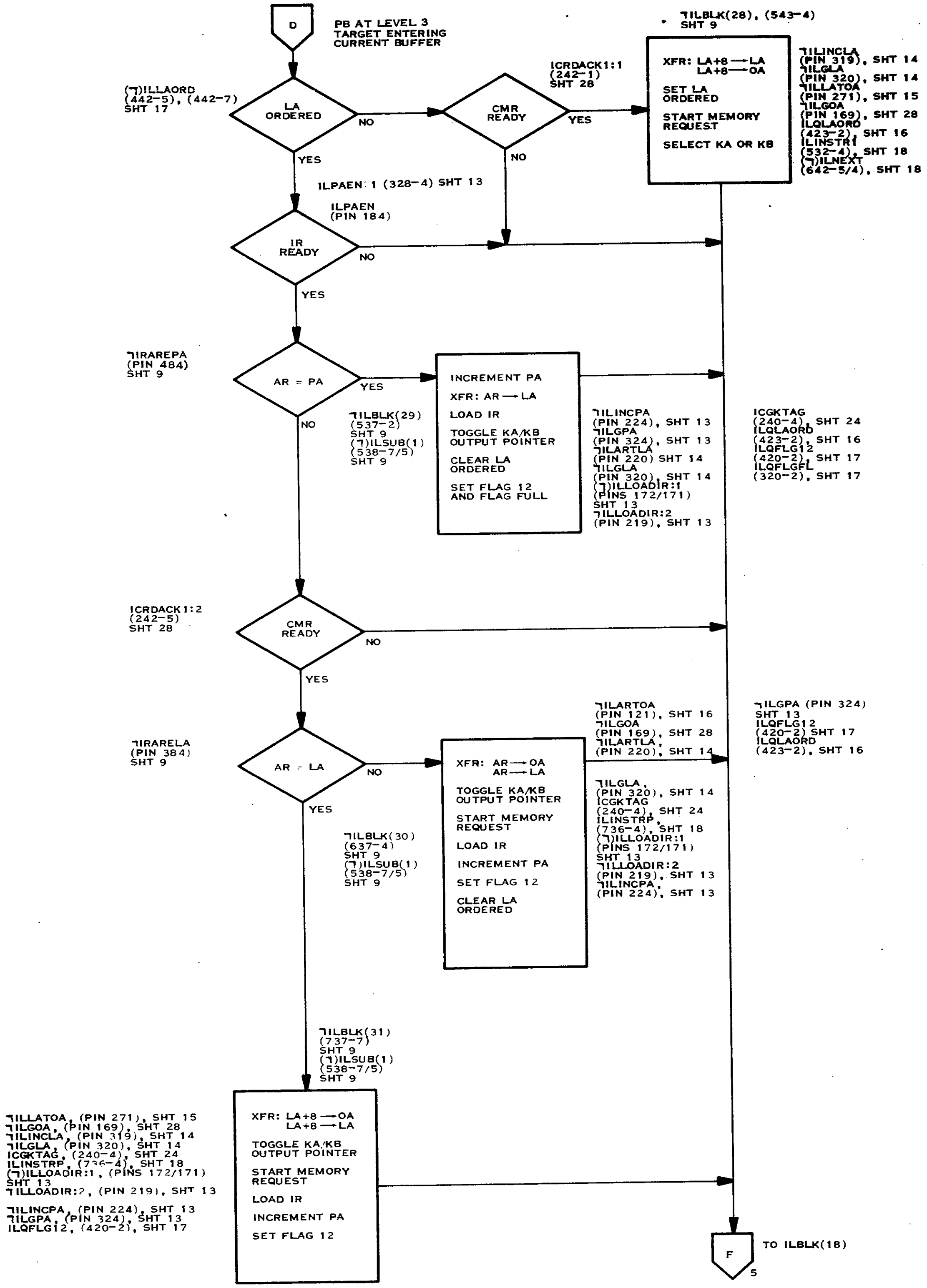


Figure B-9. Look-Ahead Controller Flowchart (Sheet 3 of 8)



(B) 123668

Figure B-9. Look-Ahead Controller Flowchart (Sheet 4 of 8)

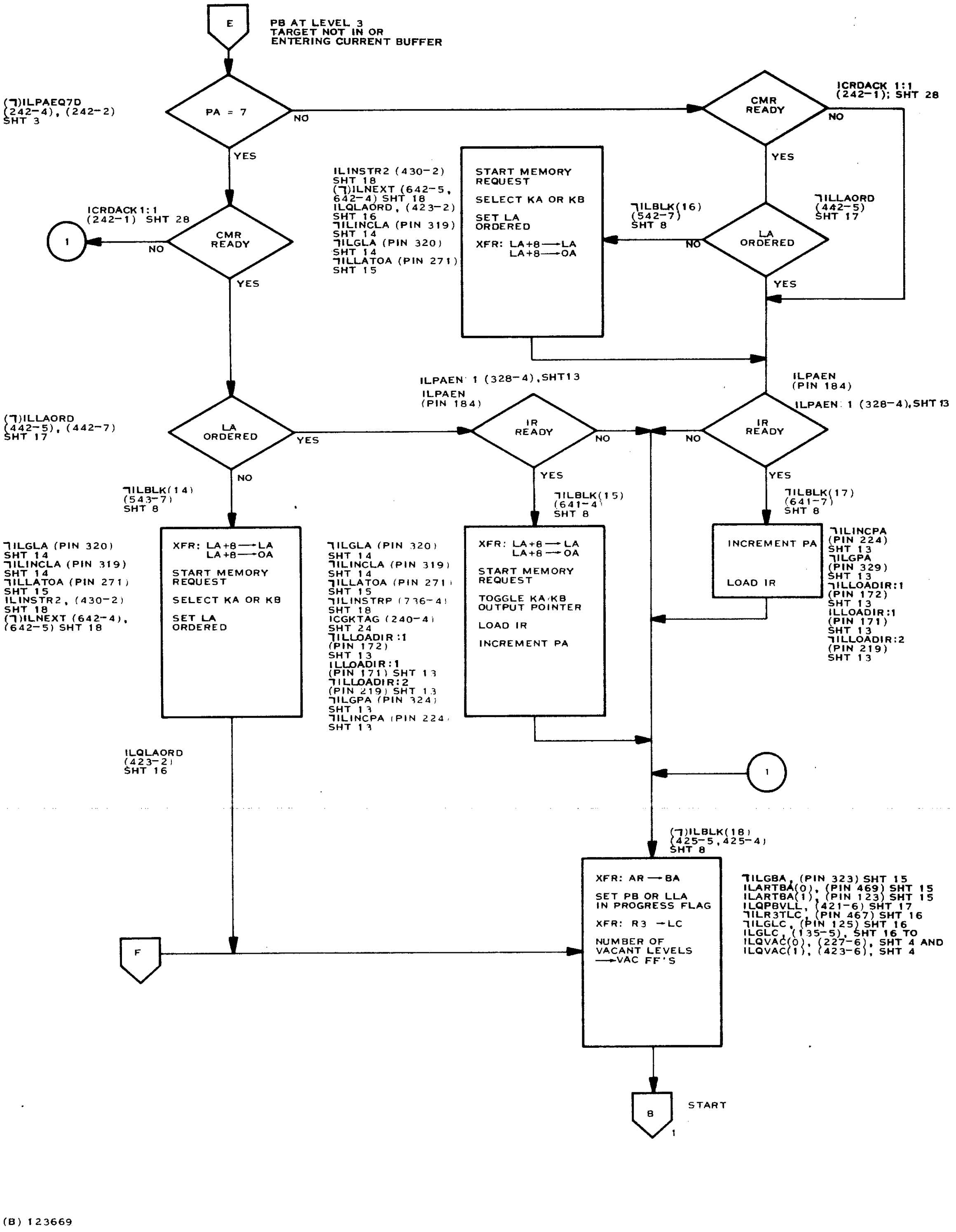


Figure B-9. Look-Ahead Controller Flowchart (Sheet 5 of 8)

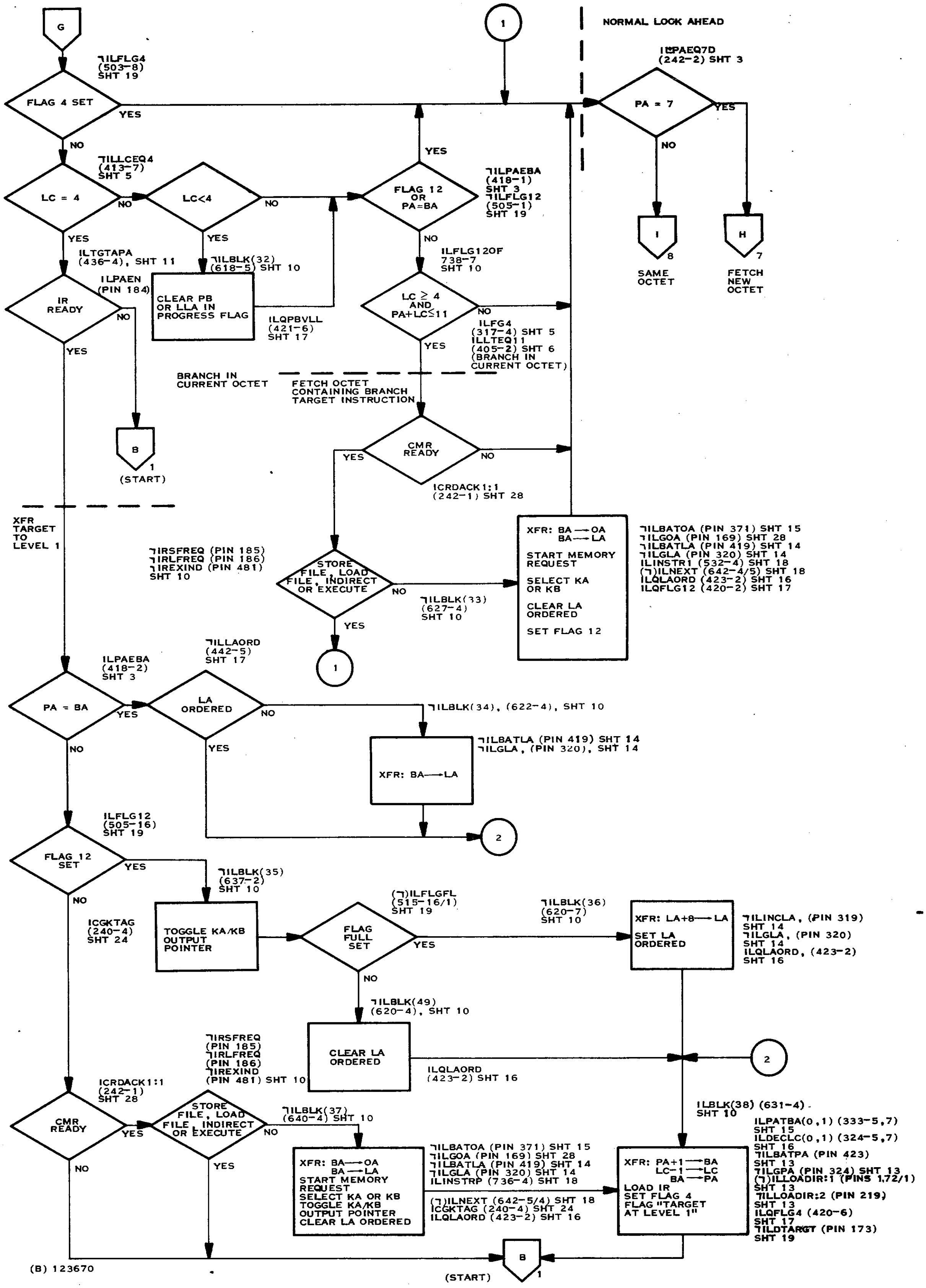


Figure B-9. Look-Ahead Controller Flowchart (Sheet 6 of 8)

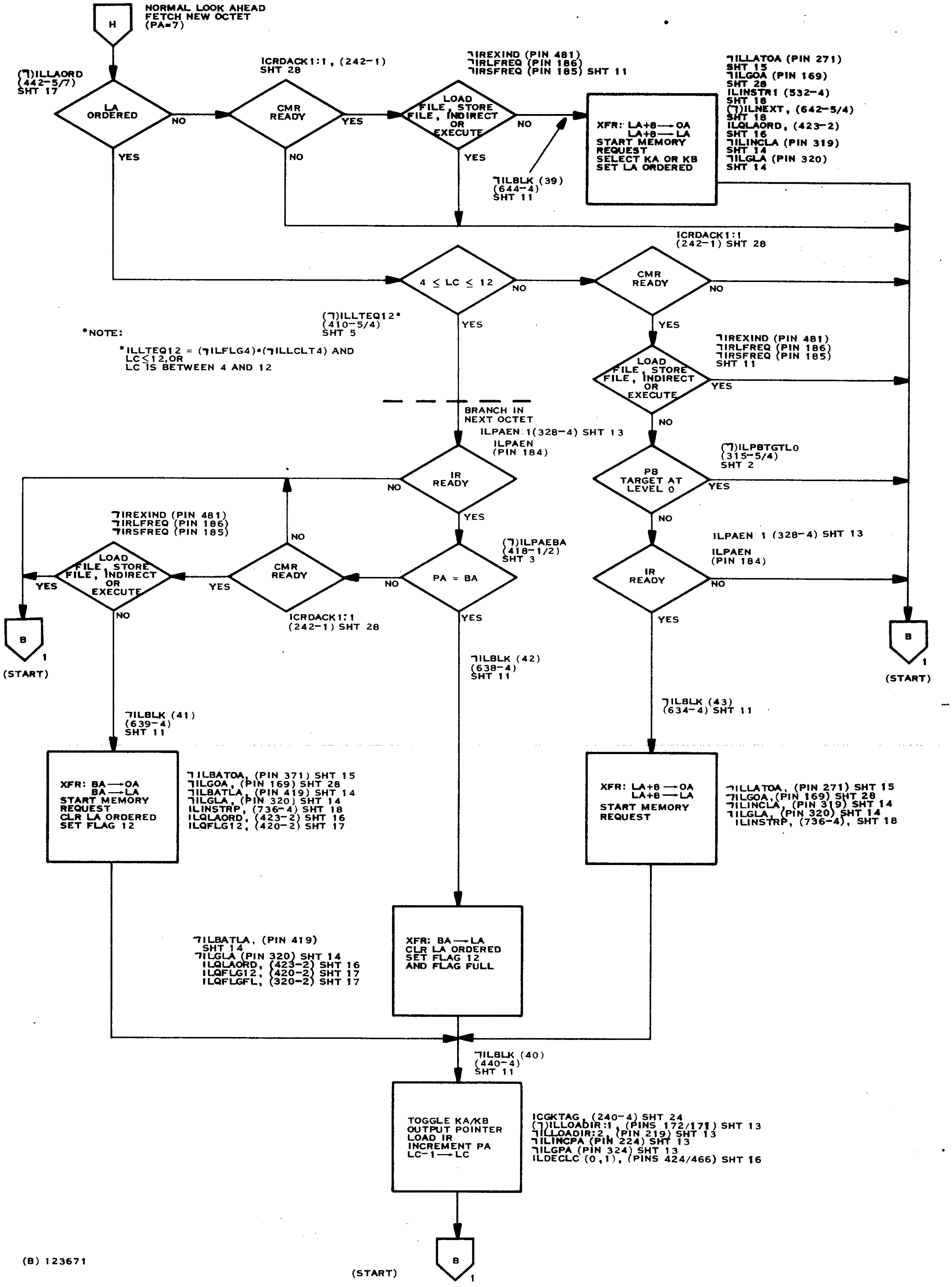


Figure B-9. Look-Ahead Controller Flowchart (Sheet 7 of 8)

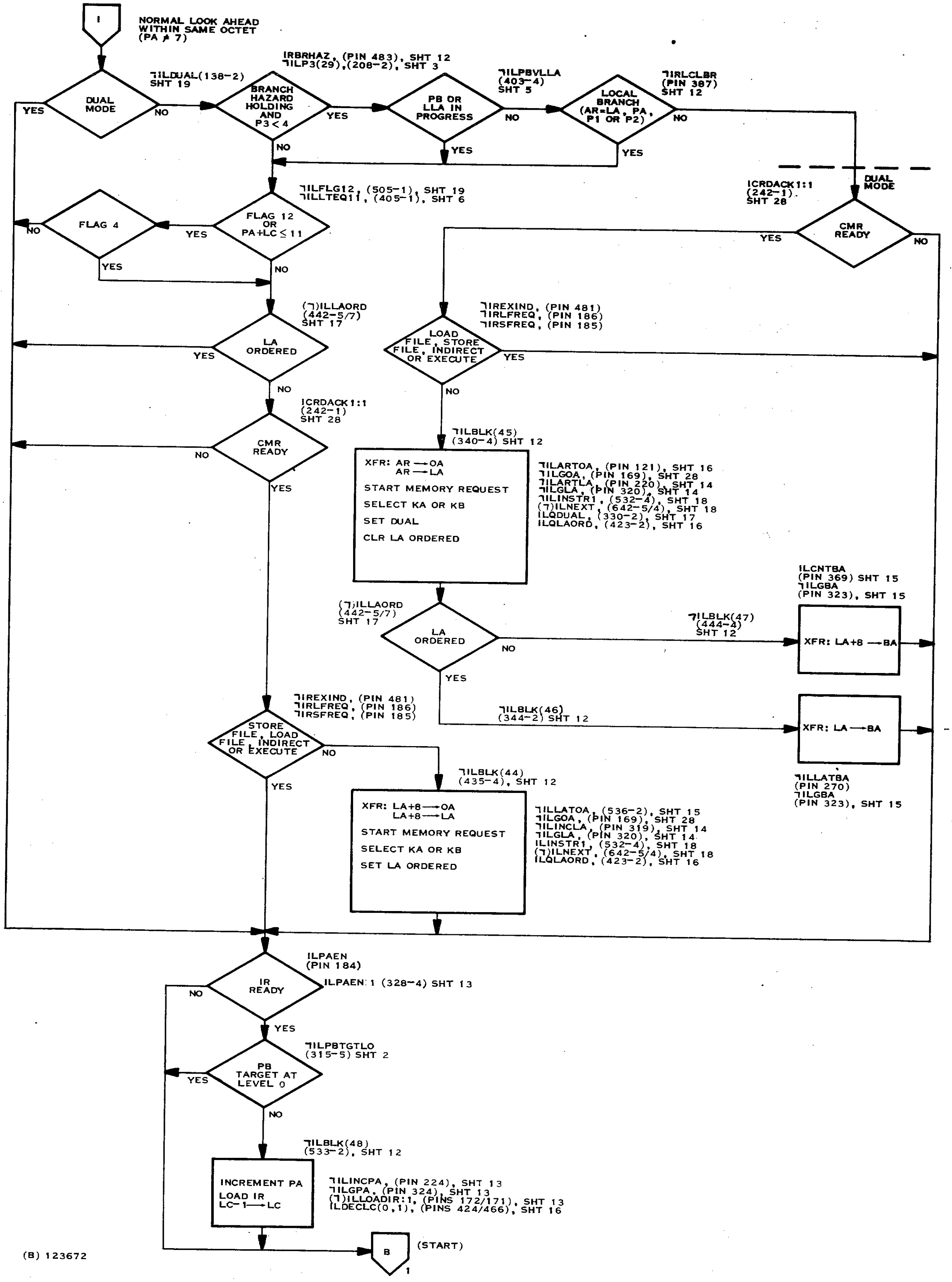

I
NORMAL LOOK AHEAD WITHIN SAME OCTET (PA ≠ 7)
¬ILDUAL(138-2) SHT 19
DUAL MODE
YES
NO
IRBRHAZ, (PIN 483), SHT 12
¬ILP3(29),(208-2), SHT 3
BRANCH HAZARD HOLDING AND P3 < 4
YES
NO
¬ILPBVLLA (403-4) SHT 5
PB OR LLA IN PROGRESS
NO
YES
¬IRLCLBR (PIN 387) SHT 12
LOCAL BRANCH (AR=LA, PA, P1 OR P2)
NO
YES
DUAL MODE
ICRDACK1:1 (242-1) SHT 28
CMR READY
YES
NO
¬ILFLG12, (505-1), SHT 19
¬ILLTEQ11, (405-1), SHT 6
FLAG 12 OR PA+LC ≤ 11
YES
NO
FLAG 4
NO
YES
(¬)ILLAORD (442-5/7) SHT 17
LA ORDERED
YES
NO
¬IREXIND, (PIN 481)
¬IRLFREQ, (PIN 186)
¬IRSFREQ, (PIN 185)
LOAD FILE, STORE FILE, INDIRECT OR EXECUTE
YES
NO
ICRDACK1:1 (242-1) SHT 28
CMR READY
NO
YES
¬ILBLK(45) (340-4) SHT 12
XFR: AR → OA
AR → LA
START MEMORY REQUEST
SELECT KA OR KB
SET DUAL
CLR LA ORDERED
¬ILARTOA, (PIN 121), SHT 16
¬ILGOA, (PIN 169), SHT 28
¬ILARTLA, (PIN 220), SHT 14
¬ILGLA, (PIN 320), SHT 14
¬ILINSTR1, (532-4), SHT 18
(¬)ILNEXT, (642-5/4), SHT 18
ILQDUAL, (330-2), SHT 17
ILQLAORD, (423-2), SHT 16
ILCNTBA (PIN 369) SHT 15
¬ILGBA (PIN 323), SHT 15
(¬)ILLAORD (442-5/7) SHT 17
LA ORDERED
NO
YES
¬ILBLK(47) (444-4) SHT 12
XFR: LA+8 → BA
¬ILBLK(46) (344-2) SHT 12
XFR: LA → BA
¬ILLATBA (PIN 270)
¬ILGBA (PIN 323), SHT 15
¬IREXIND, (PIN 481)
¬IRLFREQ, (PIN 186)
¬IRSFREQ, (PIN 185)
STORE FILE, LOAD FILE, INDIRECT OR EXECUTE
NO
YES
¬ILBLK(44) (435-4), SHT 12
XFR: LA+8 → OA
LA+8 → LA
START MEMORY REQUEST
SELECT KA OR KB
SET LA ORDERED
¬ILLATOA, (536-2), SHT 15
¬ILGOA, (PIN 169), SHT 28
¬ILINCLA, (PIN 319), SHT 14
¬ILGLA, (PIN 320), SHT 14
ILINSTR1, (532-4), SHT 18
(¬)ILNEXT, (642-5/4), SHT 18
ILQLAORD, (423-2), SHT 16
ILPAEN (PIN 184)
ILPAEN:1 (328-4) SHT 13
IR READY
NO
YES
¬ILPBTGTLO (315-5) SHT 2
PB TARGET AT LEVEL 0
YES
NO
¬ILBLK(48) (533-2), SHT 12
INCREMENT PA
LOAD IR
LC-1 → LC
¬ILINCPA, (PIN 224), SHT 13
¬ILGPA, (PIN 324), SHT 13
(¬)ILLOADIR:1, (PINS 172/171), SHT 13
ILDECLC(0,1), (PINS 424/466), SHT 16
B
1
(START)
(B) 123672

Figure B-9. Look-Ahead Controller Flowchart (Sheet 8 of 8)

BRANCH TO OA

A branch instruction at level 3 of the IPU that references an address that is not currently resident in the IPU registers requires a transfer of the branch address to OA so that the target instruction may be retrieved from memory. This condition is a branch to OA. The AR register at level 3 contains the branch address. Therefore, the controller transfers this address to OA for transmission to memory during the next memory request. In addition, the address is loaded into PA to select the target word from the new octet and into LA to be incremented and loaded into LA and OA as the next look-ahead address. At this point the address in LA is the same address as that in PA, indicating that the next octet has not yet been requested. Therefore, the "LA Ordered" flag, ILQLAORD, is cleared. The next control cycle will initiate the request to memory for the target octet.

INSTRUCTION HAZARD RECOVERY

If an instruction hazard has occurred at level 3, the controller must reaccess the octet from which the current instruction was drawn to obtain the new information. The address of the hazarded instruction is in the P3 register. To prepare for a memory fetch to access the octet, the address in P3 is transferred into the OA register. In addition, the controller loads the address into PA to select the target word from the new octet, and into LA to be incremented and loaded into LA and OA as the next look-ahead address. At this point the address in PA is the same as the address in LA, indicating that the next octet has not yet been requested. Therefore, the "LA Ordered" flag, ILQLAORD, is cleared. The next control cycle initiates the request to memory for the octet containing the hazarded instruction.

BRANCH TO LA

If a branch at level 3 references an address equal to the LA address, the target instruction will be in the look ahead buffer, KA or KB. (Since the level 3 controller checks the resident address registers in reverse order, i.e., P2, P1, PA and then LA, LA must be different from PA in order to produce a branch to LA. Therefore, LA has been ordered from memory). To access the word from LA, the target address in AR is transferred to PA and the KA/KB output selection pointer is toggled to choose the unused buffer. To ensure that LA contains the correct address and has not been altered since the branch determination, the address in AR is transferred to LA to be incremented for the next look-ahead octet. The LA Ordered flag clears to ensure that the look ahead octet will be ordered during the next control cycle.

BRANCH TO PA

If a branch at level 3 references an address equal to the PA address, the target instruction is in the current buffer. To access the target word from the current buffer, the branch address from AR is transferred to the PA register. In addition, if the LA Ordered flag is not set, the controller transfers the address in AR to LA to ensure that the next octet fetched from memory will be the next sequential octet following the target octet.

TARGET FAIL

If a branch instruction failed to branch when it reached level 3, or was skipped over in the program sequence, a target fail condition exists. The look ahead controller must then revert to the program sequence that was abandoned when the branch instruction entered the pipe. At that time the address of the instruction following the branch instruction was stored into the BA register, and the controller began loading instructions from the branch path. Since the branch will not be taken, the controller must use the address in BA to fetch the next instruction to continue the program. If the address in

BA is in the PA octet, an immediate recovery is possible without the delay of a memory fetch. The contents of BA are transferred to PA to select the proper word from the buffer to be loaded into the pipe, and if the LA octet has not been ordered from memory, BA also transfers to LA to ensure that the proper octet will be ordered for the next look-ahead octet. If the BA address is not equal to PA, a memory request cycle is required to recover from the target fail. The controller sets the TFAIL flag on this clock, so that the following clock will initiate a memory fetch for the correct octet (see Target Fail Flag).

TARGET FAIL FLAG

If a branch failed to branch and the address in BA (recovery address) is not in PA, the Target Fail Flag (TFAIL) has been set to indicate that a memory fetch is required to recover from the failure. To perform the memory fetch, the recovery address in BA is transferred to OA and a signal is sent to the central memory requester to begin a memory request. In addition, BA transfers to PA to select the word from the fetched octet and continue the program sequence. BA also transfers to LA to ensure that the proper look-ahead octet is fetched. At this point, LA is equal to PA, indicating that the look-ahead octet has not been fetched. The LA Ordered flag clears. This sequence of events corrects the target fail condition, so that the controller clears the Target Fail Flag.

DUAL AND BRANCH NOT TAKEN

If the look ahead controller is operating in the Dual mode and the branch at level 3 will not be taken, the look ahead octet that was discarded at the beginning of Dual mode must be retrieved. The address of this octet is stored in the BA register. The look ahead controller transfers the address in BA to OA and signals the central memory requester to initiate a memory request to fetch the look ahead octet. It also loads the address in BA into LA to ensure that the next look ahead octet will be in sequence with the octet that is being requested. Since BA contains the look ahead address of the current octet, the address in LA is eight greater than PA, indicating that the look ahead octet has been ordered. The LA Ordered flag sets. Fetching the new octet terminates Dual mode, so the controller clears the Dual flag. To prepare for loading IR with the next instruction, selection of a word from KA or KB is enabled. If IR can accept a new instruction and the next instruction is not the target of a PB in the pipe, the controller loads IR with a word from the current buffer and increments the PA address. A PB target must be held at level 0 until the PB reaches level 3.

PB TARGET AT LEVEL 3

When a targeted branch instruction reaches level 3, the look ahead controller determines if the branch is to a location that is currently in the IPU pipe. If the branch address is equal to either P2 or P1, the target instruction is in the pipe. The IPU needs only to clock the target instruction through the pipe to level 3. No additional memory fetches will be required to obtain the instruction. The look ahead controller clears the branch status flags (PB or LLA in Progress, Flag 12, Flag Full and Flag 4) to indicate that the branch has been satisfied, and continues with a normal look ahead cycle to put a new instruction into IR. The other pipe level controllers ensure that the pipe remains inactive until the target instruction reaches level 3.

PB OR LLA IN PROGRESS

If a PB or LLA instruction has prepared the look ahead controller for an upcoming branch instruction, the controller checks the look ahead counter (LC) and flag 4 (target in pipe) to determine if the branch is imminent. If the branch instruction is in the curretly executing octet, the controller transfers the address of the target instruction from BA into OA and LA (ILBLK(33) on sheet 6), initiates a request for that octet, sets Flag 12 to indicate that the target instruction has been ordered from memory, and enables a word selection from the KA or KB buffer files. Because at this point a new string of octets is to be begun, and LA is not the next sequential octet address from PA, the LA Ordered flag is cleared. The controller completes the cycle by loading a new instruction into IR and decrementing the look ahead counter.

LLA AT LEVEL 3

When a Load Look Ahead (LLA) instruction reaches level 3, the controller must establish conditions in the IPU address registers to that it can respond when the branch enters the pipe. To prepare for the branch, the controller sets the PB or LLA in Progress flag to indicate that the instruction has been recognized, transfers the contents of P3 (address of target instruction) to BA, loads the contents of AR (number of instructions until the branch occurs) into the look ahead counter, LC, and sets the VAC flip flops with the number of vacant levels in the pipe. These vacant levels must be considered when evaluating the LC count to determine the position of the branch instruction. After setting these conditions, the look ahead controller continues with the normal look ahead cycle to load a new instruction into IR.

PREPARE TO BRANCH AT LEVEL 3

When a PB instruction reaches level 3, the look ahead controller enters one of three paths to prepare the address registers for the branch instruction. The position of the target branch within the memory buffer file determines which path is followed. If the target branch has cleared the look ahead buffer and is in the current instruction octet, the controller follows the path illustrated on sheet 3 of the flow chart. If the next clock will use the last word in the current buffer and the look ahead buffer contains the target branch, the controller takes the path diagramed on sheet 4 of the flow chart. If neither of these conditions is true, the controller follows the path on sheet 5 of the flow chart. The following paragraphs describe the logical sequence of each of the prepare to branch paths.

PB TARGET IN CURRENT BUFFER

If the target branch is in the current octet and the instruction to be referenced by the upcoming branch instruction is resident in the IPU instruction octets (AR = LA or PA), the controller ensures that the look ahead octet has been requested (LA Ordered flag set) and sets Flag 12 to indicate that the target instruction is in the IPU instruction registers, or has been requested from memory. If the target instruction is not in the IPU, the controller must fetch the octet that contains that instruction. If a request is permitted (CMR Ready), the controller transfers the address of the target instruction to the OA and LA registers and signals the central memory requester to begin a memory request. This action sets LA equal to some other octet than the next sequential octet, so that the LA Ordered flag is cleared. The controller then sets Flag 12 to indicate that the target instruction has been requested from memory. After Flag 12 sets (through either of the above paths), if IR can accept a new instruction and the PB target is not at level 0, the controller loads IR with a new instruction, increments the instruction address in PA, stores the branch address from AR into BA and the look ahead count from R3 into LC, and then enables the number of vacant levels in the IPU into the VAC flip flops for use in tracking the position of the branch instruction. When these conditions have been established, the controller sets the PB or LLA in Progress flag to indicate recognition of the PB instruction, and awaits the next control clock.

If the target branch is at level 0 when Flag 12 sets, the controller loads the branch instruction into IR (if IR is ready) and transfers the address of the target instruction from AR into PA to begin drawing instructions from the branch path. In case the branch is not taken when the branch instruction reaches level 3, the address following the address of the branch instruction (P + 1) is loaded into BA. (If the branch is not taken, this address is then loaded into PA, LA and OA to retrieve the discarded program steps.) The controller then sets Flag 4 to indicate that the branch is in the pipe, and signals the level 1 controller that the branch is in level 1 (Target at level 1). Concurrent with these steps the controller examines the address of the target instruction contained in AR and compares it with the current PA octet. If AR is equal to PA, then the look ahead octet that was previously ordered is the proper look ahead octet. The controller therefore sets the LA Ordered flag. If AR is not equal to PA, then it must be in the LA octet to be at this point in the flow chart. Therefore, the controller toggles the KA/KB output pointer to select the look ahead octet for the next instruction, and clears the LA Ordered flag indicating the need for a look ahead octet.

At the end of all sequences through the PB portion of the flow chart, the look ahead count in the R3 register is loaded into LC, the number of vacant levels in the IPU is stored in the VAC flip flops and the PB or LLA in Progress flag sets. The controller then waits for the next control clock to begin another cycle.

PB TARGET ENTERING CURRENT BUFFER

When a PB reaches level 3, the next clock will use the last word in the current buffer and the target branch is contained in the upcoming buffer, the look ahead controller ensures that the look ahead octet has been ordered. If it has not, the controller transfers the look ahead address into OA and LA, sets the LA

Ordered flag, and signals CMR to initiate a memory request for the octet. The controller also enables selection of a word from the buffer for insertion into IR.

If the look ahead octet has been ordered from memory and IR can accept a new instruction, the controller determines if the address of the target instruction is equal to PA. If the target instruction is contained in PA, the controller loads the final word from PA octet into IR, increments PA and toggles the KA/KB output pointer to choose the next octet. It then loads the address of the target instruction from AR into LA and clears the LA Ordered flag so that the next octet following the target instruction will be ordered in the normal look ahead cycle following the branch. The controller then sets Flag 12 to indicate that the target instruction is present in the buffer, and Flag Full to indicate that both the branch instruction and the target instruction of the branch are in the two buffer files.

If AR is not in the PA octet, the controller determines if AR is contained in the look ahead octet that has been ordered. If it is not, the controller loads the target instruction address into OA and LA and signals the CMR to begin a memory request. It then loads the final word from the current buffer into IR, increments PA and toggles the KA/KB output pointer to the other octet buffer. Flag 12 sets to indicate that the target instruction octet has been ordered, and the LA Ordered flag clears to indicate that the octet that will enter the look ahead buffer is not the next sequential octet.

If AR is not in the PA octet, but is in the ordered LA octet, the controller loads the last word in the current octet into IR, increments PA and toggles the KA/KB output pointer to select the other octet buffer. It then transfers a new look ahead octet address (LA + 8) into OA and LA and signals to CMR to begin a request for that octet. Flag 12 sets to indicate that the octet containing the target instruction of the branch is entering the look ahead buffer.

Regardless of the path taken through this portion of the flow chart, the controller completes the control cycle by transferring the target instruction address from AR into BA for reference when the branch enters the pipe, loads the look ahead counter, LC, with the count from R3 that indicates the number of instructions between the PB and the branch instruction, and loads the number of vacant levels in the IPU into the VAC flip flops for use in tracking the branch instruction. The PB or LLA in Progress flag sets to indicate recognition of the PB instruction.

PB TARGET NOT IMMINENT

If the target branch is not in the current instruction octet and the IPU is not switching to the octet buffer that contains the target branch, two possible conditions exist. Either the target branch is in the look ahead octet and the present word address in PA is not equal to 7, or the target branch is not in the look ahead octet. If PA is not equal to 7, regardless of the position of the target branch, the controller determines if the look ahead octet has been requested from memory (LA Ordered). If it has not, the controller loads LA and OA with the look ahead address (LA + 8), signals the central memory requester to begin a memory fetch for that octet, and sets the LA Ordered flag to indicate that the octet has been requested. The controller then enables selection of a word from the current instruction buffer, and if IR can accept a new instruction, loads a word from the instruction buffer into IR and increments PA to the next word address.

If PA is equal to 7 and the target branch is not in the next look ahead octet, the controller ensures that the look ahead octet is available in the look ahead buffer by checking the LA Ordered flag. If the octet is not present, the controller loads the look ahead address (LA + 8) into LA and OA, initiates a memory request through the Central Memory requester, and sets the LA Ordered flag. The controller must then wait for the next control clock to load the final instruction of the current octet into IR. If the look ahead octet is present in the IPU, however, the controller loads the last instruction into IR, increments PA to the next instruction address and toggles the KA/KB output pointer to select the other instruction octet buffer. The controller also transfers the look ahead address of the next octet into LA and OA (the octet containing the target branch) and initiates a memory request for that octet. Since the LA Ordered flag is already set, it remains set to indicate that a new look ahead octet has been ordered.

Regardless of the path taken through this portion of the prepare to branch logic, the controller always ends the cycle by storing the branch parameters for use when the target branch reaches the pipe. It transfers the address of the target instruction from AR to BA, the number of instructions until the branch from R3 to LC, the number of vacant levels in the IPU into the VAC flip flops for use in tracking the target branch, and sets the PB or LLA in Progress flag. The controller will decrement the count in LC with each new instruction placed in IR until the branch arrives at level 3.

TRANSFER TARGET TO LEVEL 1

If Flag 4 (branch in pipe) is not set, and the count in LC is equal to 4, the target branch is at level 0 of the pipe and will be the next instruction transferred to IR. If IR can accept a new instruction, the controller compares the address of the target instruction in BA with the present instruction octet address. If the target instruction is in the current instruction octet (PA=BA), and the look ahead octet has not been requested (LA Ordered not set), the controller transfers the address in BA to LA so that the next octet in sequence with the target instruction octet will be ordered when a memory request is initiated. After preparing LA, or if LA Ordered was set, the controller loads the target branch into IR, signals the level 1 controller that the target is at level 1, and sets Flag 4 to indicate that the branch is in the pipe.

This transfer decrements the count in the LC counter. The controller then transfers the address of the target instruction to PA so that the instructions from the branch path will follow the branch instruction through the pipe to avoid the delay necessitated by having to fetch the new instructions when the branch reaches level 3. In case the branch is not taken when it reaches level 3, the controller transfers the address of the next instruction following the branch (P + 1) into BA. The controller can then reconstruct the instruction sequence if the branch is not taken.

If the target instruction was not in the current instruction octet (PA not equal to BA), the controller determines if the target instruction is in the look ahead octet (Flag 12 set). If Flag 12 is set, the controller loads the branch instruction into IR, decrements the LC counter, sets Flag 4 to indicate that the branch is in the pipe and notifies the level 1 controller that the target is at level 1. It then transfers BA to PA and saves P + 1 in BA for recovery in case the branch is not taken. The controller toggles the KA/KB output pointer to select the look ahead octet containing the target instruction, and examines the Flag Full flag. If this flag is clear, the controller clears the LA Ordered flag to ensure that a new look ahead octet will be requested on the next control cycle. If Flag Full is set, the current octet is also the octet that follows the look ahead octet. For this reason the controller sets the LA Ordered flag to prevent accessing a new octet, and places the address of the current octet (LA +8) into the look ahead address register. If the branch is taken, the IPU will draw from the octets in the following order:

1. Current Octet - until the branch instruction is reached.
2. Look Ahead Octet - containing the target instruction of the branch.
3. Current Octet
4. Next sequential octets (if branch not taken a second time).

If the target instruction is not in either PA or LA, the controller must request the octet from memory. It then transfers the address of the target instruction from BA to OA and LA, clears the LA Ordered flag and signals the Central Memory Requester to begin a memory fetch for the octet. The branch instruction is loaded into IR, Flag 4 sets, the LC counter decrements and the controller informs the Level 1 Controller that the target is at level 1. The address of the target instruction is then transferred from BA to PA and the KA/KB output pointer is toggled to select the look ahead buffer containing the requested target instruction octet for the next control cycle. The controller also stores PA +1 in BA for recovery in case the branch is not taken.

FETCH TARGET INSTRUCTION

If the target branch is not at level 0 and not in the pipe (LC greater than four and Flag 4 not set), and the target instruction is not in the IPU (Flag 12 not set and PA ≠ BA), the controller determines if the target branch is in the current octet. If it is in the current buffer, the controller must fetch the octet containing the target instruction so that the target instruction can be inserted into the pipe following the target branch. Therefore, the controller transfers the address of the target instruction into OA and LA, and signals the central memory requester to initiate a memory request for that octet. Flag 12 sets to indicate that the target instruction has been requested. Since this octet is a departure from the look ahead sequence, the LA Ordered flag clears.

The controller determines if the target branch is in the current buffer by examining the LC count, the number of vacant levels (VAC count) at the time the LC count was loaded, and the present address (PA). When the LC count was loaded, it indicated the number of instructions from the PB at level 3 until the branch instruction. If there were vacant levels in the pipe, however, these levels could have been filled from the current buffer (decrementing the LC count) without changing the number of instructions between the PB and the branch. Therefore, to gain an accurate position of the branch, the number of vacant levels must be added to the LC count, yielding the number of IPU levels from level 3 to the branch. The maximum number of levels in the current buffer and the IPU pipe is 11. This number decreases each time an instruction is used from the current buffer, so that the current number of levels is actually 11 minus the current word address. Therefore, if the target branch is in the current buffer, LC plus VAC must be less than or equal to 11 minus PA. By transposition, the controller performs the comparison;

$$LC + VAC + PA \leq 11,$$

to make the determination.

If Flag 4 was not set (branch not in pipe) before entering this decision sequence but LC was less than 4, the controller clears the PB or LLA in Progress flag. This set of conditions is not possible if a PB or LLA is in progress, because LC less than 4 indicates that the branch must be in the pipe.

NORMAL LOOK AHEAD

During normal instruction sequencing the look ahead controller inspects the address in PA to choose one of two logic paths. If the address in PA is equal to 7, a new octet must be fetched for the look ahead buffer. This logic path is diagramed on sheet 7 of the flow chart. If PA is not equal to 7, the next address will not exhaust the current buffer and no additional memory cycle is required at this point. The following paragraphs describe the logic steps for each of these paths.

FETCH NEW OCTET. If PA is equal to 7, the next instruction transfer to IR will access the last instruction in the current buffer. To ensure that a look ahead octet is available, the controller inspects the LA Ordered flag. If this flag is not set, the controller transfers the look ahead address (LA + 8) to OA and LA, initiates a memory request through the Central Memory Requester, enables a word selection from the current memory buffer, and sets the LA Ordered flag. No transfer to IR is made until the next control cycle, so that a look ahead octet will be available in the IPU.

If the LA Ordered flag is set, the controller checks the LC count. If this count is between 4 and 12, then a branch instruction will appear in the next octet that has been preceded by a Prepare to Branch instruction. The controller then checks the address of the target instruction contained in BA to determine if it is contained in the current octet (BA=PA). If the target instruction is contained in PA, the controller must save the PA octet for use after the branch instruction. The controller then loads IR with the last instruction in the current octet, decrements the LC count, increments PA and toggles the KA/KB output pointer to select the next octet buffer. To save the exhausted octet for the branch, the controller sets the Flag Full flag and Flag 12, transfers the target instruction address from BA to LA, and clears the LA Ordered flag to indicate that the look ahead octet is not a sequential look ahead address.

If the target instruction is not contained in PA (PA not equal to BA) then the controller must fetch a new look ahead octet. When the control clock enables transfers, the controller loads IR with the last instruction in the current octet, increments the address in PA, toggles the KA/KB output pointer to the other octet buffer, and decrements the LC count. To fetch the octet containing the target instruction, the controller transfers the address of the target instruction from BA to OA and LA, signals the Central Memory Requester to initiate a memory request, sets Flag 12 to indicate that the target instruction has been requested, and clears the LA Ordered flag to indicate that the look ahead octet is not a sequential octet address.

If the LC count is not between 4 and 12 (therefore it must be either greater than 12 or non-existent, since if it were less than 4 it would have triggered another flow chart path), the controller follows a simple look ahead procedure to fetch a new sequential octet for the look ahead buffer. When the control clock enables transfers, the controller loads the last instruction from the current buffer into IR, increments the address in PA, toggles the KA/KB output pointer to select the look ahead octet as the current octet for the next clock, and decrements the LC count. The controller then transfers the look ahead address (LA + 8) into OA and LA and initiates a memory request for that octet through the Central Memory Requester.

WITHIN SAME OCTET. If PA is not equal to 7, the next instruction transfer to IR will not exhaust the current instruction buffer. If the controller is not operating in the Dual mode, or will not enter the Dual mode, the controller checks to see if a targeted branch is within the current octet (PA + LC $\leq$ 11) or if its target instruction is in the look ahead octet (Flag 12). If a branch is present but is already in the pipe (Flag 4) or if no branch is imminent, the controller ensures that the look ahead octet has been requested. If it has not been requested, the controller transfers the look ahead address (LA + 8) to OA and LA, initiates a memory request for that octet, sets the LA Ordered flag and enables a word selection from KA or KB instruction buffers. In any case, if IR can accept an instruction transfer, and there is no PB target at level 0 when the PB is in the pipe, the controller loads a new instruction into IR, increments the address in PA and decrements any count contained in the LC counter.

DUAL MODE

The look ahead controller enters the Dual Mode when a branch instruction has reached level 3, but a previous instruction in the pipe may modify the conditions for the branch (branch hazard). Dual mode holds the current octet containing the branch instructions and fetches a new look ahead octet that contains the target instruction while holding the branch at level 3. This state is avoided if the branch was preceded by a LLA or PB instruction, since these instructions fetch the target instruction octet and put the target instructions in the pipe following the branch instruction. Similarly, if the branch is to a local octet or word (contained in LA, PA, P1 or P2) a simultaneous fetch to memory is not required. The address in P3 must be less than 4 in order for the controller to enter the Dual mode since a new octet will be required to supply instructions to the pipe if P3 is 4 or more. The buffers hold the current octet and the controller fetches the target octet. There can be no look ahead octet.

If the conditions for dual mode are met, the controller transfers the address of the target instruction from AR to OA and LA and initiates a fetch for that octet through the Central Memory Requester. The controller then enables selection of a word from KA or KB, sets the Dual flag to indicate the mode of operation, and clears the LA Ordered flag to indicate that the look ahead octet is not in the normal sequence. If the LA Ordered flag was set prior to the control clock pulse, then the address of that look ahead octet is stored in BA for retrieval in case the branch is not taken. If the LA Ordered flag was not set, the controller stores the address of the look ahead octet that should have been ordered (LA + 8) in BA for retrieval. If IR can accept a new instruction and the instruction at level 0 is not a PB target, the controller then loads IR with an instruction from the current octet, increments the address in PA and decrements the count in LC.

CENTRAL MEMORY REQUESTER (CMR)

Central Memory Requester (CMR) is the control circuit in the IPU that issues all normal operation memory requests to the Memory Control Unit (MCU), and accounts for all outstanding requests to memory. IPU Unit Hard Core has responsibility for memory requests during maintenance fetches and stores. As CMR issues each memory request to the MCU, it places a two bit code into the memory request queue. This code identifies the ultimate destination of the octet of data corresponding to that request. When the octet enters the IPU from memory, CMR retrieves the code from the queue to route the data to the chosen destination. CMR generates the gating signals required to perform the designated transfer, and increments the request queue input and output pointers with each request. If a memory protect violation occurs, CMR stores the violation as a flag. The flag accompanies the data generated by the violation as that data travels through the pipe. When the faulty data reaches level 3 the IPU recognizes the protect violation. Only then is the IPU disabled due to the violation.

CMR is physically distributed between I4CMREQ and I4HDCORE circuit boards. The majority of the control circuitry is on I4CMREQ and it is therefore included with the I4CMREQ circuit board discussion. However, the actual interface signals generated to the MCU are implemented on I4HDCORE.

The following paragraphs describe the operation of the CMR with reference to the flow chart of the controller logic that follows this description. The paragraphs follow the same order as the logic flow through the chart, and explain the major decision paths that are possible within the controller.

CONTROLLER TIMING

The Central Memory Requester is composed of combinational logic, and as such, has no timing chain, sequence of events, or formal states. All of the question blocks illustrated in the flow chart are examined simultaneously during each control cycle to enable only one path through the controller. When the control clock pulse occurs, all of the action blocks on that enabled path are executed simultaneously. This type of timing means that action blocks upstream from other decision blocks in the flow chart do not affect the decision block. Also, since all actions occur simultaneously, all action statements refer to conditions at the start of the control cycle.

CMR TERMS

The following terms and their definitions are essential to understanding the flow charts and the Central Memory Requester discussion:

- AR - Access Request: An interface signal to the Memory Control Unit (MCU) that, when toggled, indicates that the IPU requires access to some location in central memory. This signal is implemented on the I4HDCORE circuit board.
- Active Flag - a flag identified for each of the four request queue positions that indicates that the queue position contains a valid request to memory.
- BOGUSA/BOGUSB - internal controller signals that are generated when a request is made to memory for an octet that will supercede all other octets destined for the KA or KB buffer, respectively. These signals

prepare the target buffer to receive the new octet, and cancel the active flag for outstanding requests for their respective buffers.

- Busy Flag - a flag identified for each of the four request queue positions that indicates that the queue position is occupied by an outstanding memory request.

- CR File - Communications Register File: An array of registers in the Peripheral Processor (PP) that supplies maintenance commands and receives status and reply messages from major ASC units.

- IR - Instruction Register: the level 1 register of the IPU that receives instructions from KCM, KA, KB or the register file.

- KAFUL/KBFUL - flags that indicate that their respective buffer, KA or KB, contains an octet of instructions that can be transferred into the IPU pipe.

- KAPRV/KBPRV - Protect Violation - flags that indicate that their respective buffer, KA or KB, contains an octet of instructions that is not valid. This flag is passed through the pipe with the instruction until it reaches level 3. Level 3 rejects any instructions that have the PRV flag set.

- KRTAG - pointer that selects either KA or KB to supply instruction words to the IR register. When equal to "0", KRTAG selects outputs from KA; when equal to "1", KRTAG selects outputs from KB.

- NEXT - signal from the look ahead controller that is used with KRTAG to determine the destination buffer of a memory fetch. If NEXT and KRTAG are set to the same values (either 1's or 0's), KA is the destination buffer; if they are different values, KB is the destination buffer.

- OA - Output Address register: the register on the I4ADDR circuit board that transfers memory address to the MCU for memory accesses.

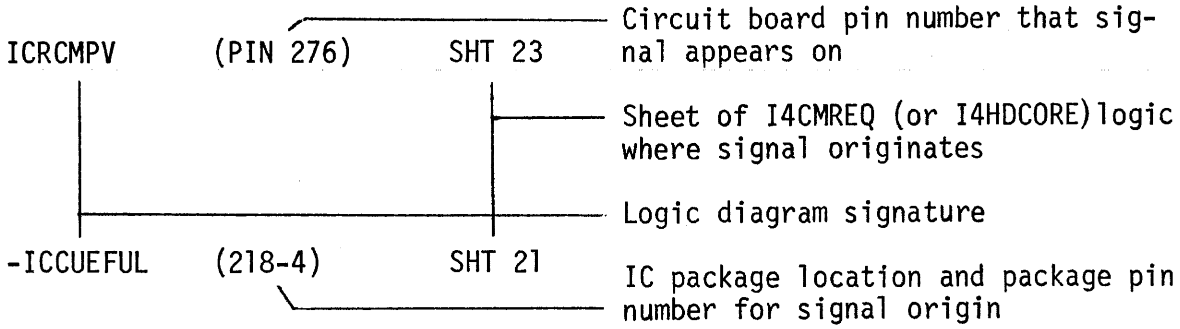
- PM(0,1) - Protect Mode bits: a two bit code to the MCU that identifies which set of protect parameters in the MCU protect registers will be used to define the permissible memory area for a particular operation. A code of "00" specifies no memory protect, "01" specifies write protect parameters, "10" specifies read protect parameters, and "11" specifies execute protect parameters.

- Protect Violation - A signal from the MCU (ICPRVO) that indicates that the last memory request attempted to access an area in memory that was outside of the permitted boundaries. The request is refused and no data will be transferred from the MCU as a result of that request.

- Queue - Request Queue: a four position array of two bit codes that identifies the intended destination of up to four outstanding memory requests. A code of "00" identifies the KA buffer, "01" identifies the KB buffer, "10" identifies the IR register, and "11" identifies the register file as the destination for the instruction octet contained in KCM when the queue position is referenced by the output pointer.

FLOW CHARTS

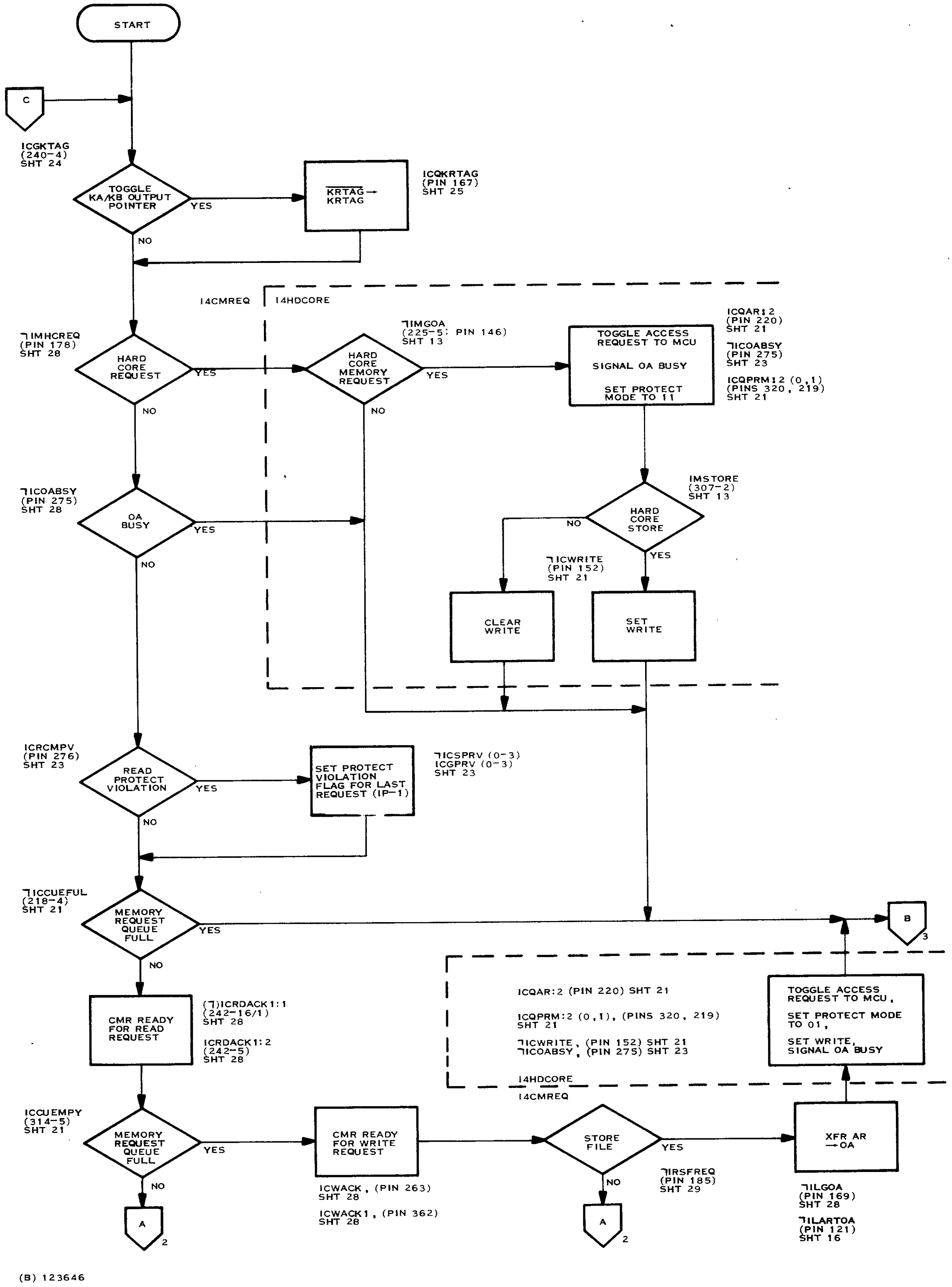
The CMR flow chart (figure B-10) that follows this description may be used as both a theory learning tool and a maintenance tool. Each question or action block contains a brief phrase that describes the function(s) of that particular block. Besides the block is the exact signature of the signal that is being examined (question block) or produced (action block). Along with the signature is a set of tagging information that designates the origin or monitoring point for that signal in the hardware, and the sheet in the logic diagrams where that signal originates. All sheet references are to sheets within the logic set for the I4CMREQ circuit board, except where dotted boxes on the flow charts indicate that the signals have been transferred to the I4HDCORE circuit board. The example below illustrates the tagging information included with each signature on the flow charts.


ICRCMPV
(PIN 276)
SHT 23
Circuit board pin number that signal appears on
Sheet of I4CMREQ (or I4HDCORE) logic where signal originates
Logic diagram signature
-ICCUEFUL
(218-4)
SHT 21
IC package location and package pin number for signal origin

Flow Chart Tagging Information - CMR Flow Charts

TOGGLE KA/KB OUTPUT POINTER

The KA/KB output pointer selects one of the two instruction buffers to supply instruction words to the IPU. When the look ahead controller requires a new



(B) 123646

Figure B-10. Central Memory Requester (Sheet 1 of 4)

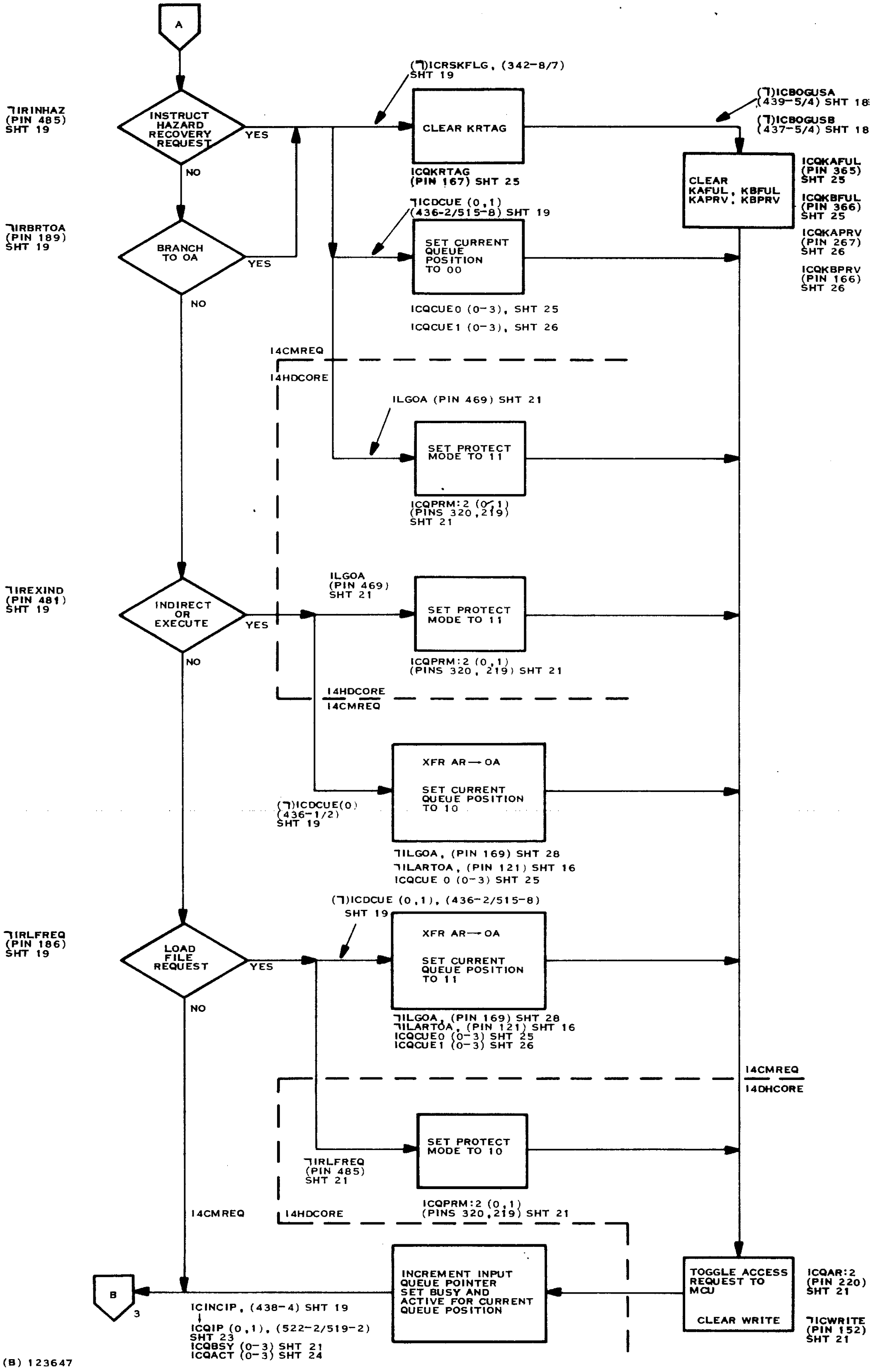


Figure B-10. Central Memory Requester (Sheet 2 of 4)

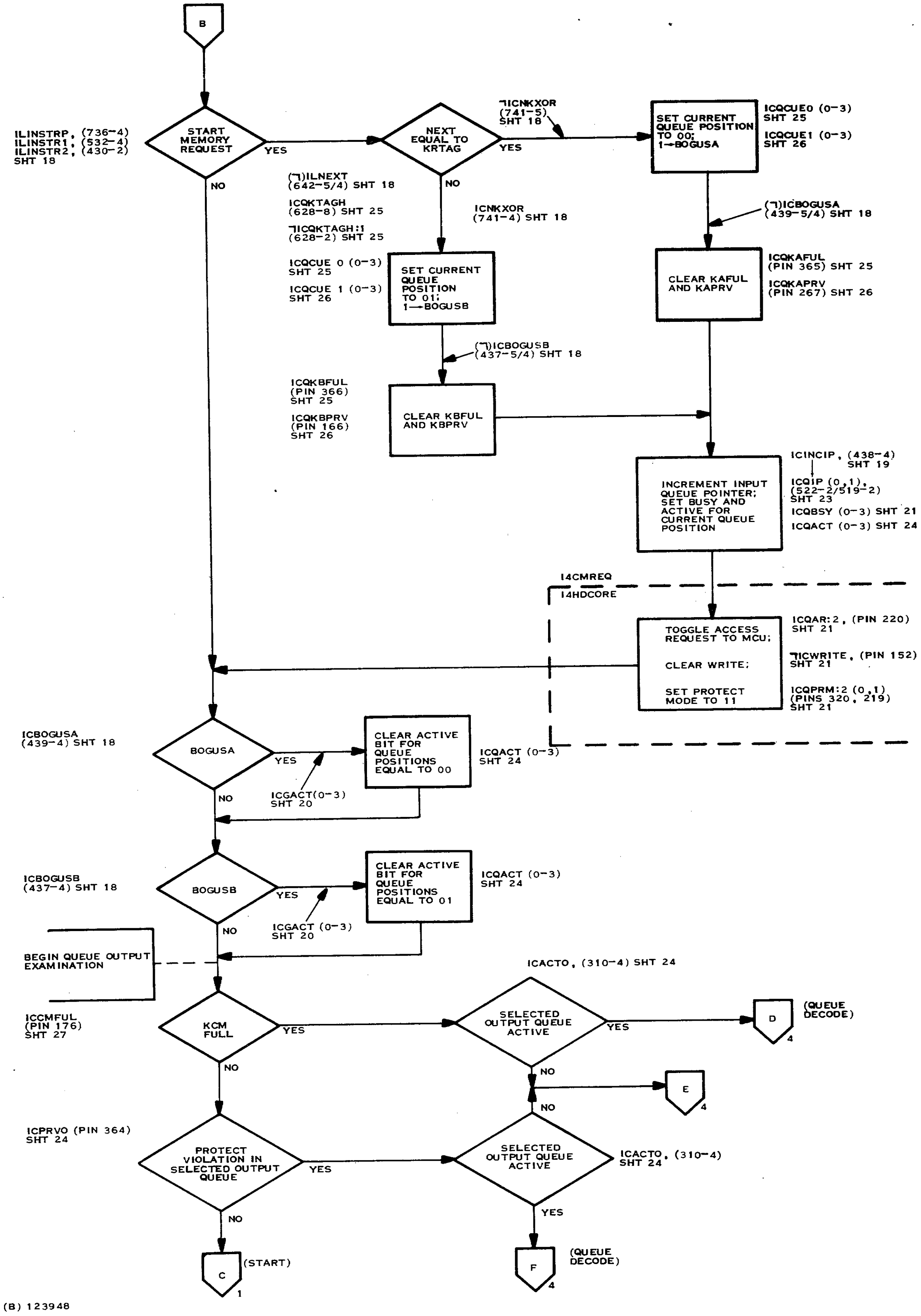

B
ILINSTRP, (736-4)
ILINSTR1, (532-4)
ILINSTR2, (430-2)
SHT 18
START MEMORY REQUEST
YES
NO
NEXT EQUAL TO KRTAG
YES
NO
¬ICNKXOR (741-5) SHT 18
SET CURRENT QUEUE POSITION TO 00; 1→BOGUSA
ICQCUE0 (0-3) SHT 25
ICQCUE1 (0-3) SHT 26
(¬)ILNEXT (642-5/4) SHT 18
ICQKTAGH (628-8) SHT 25
¬ICQKTAGH:1 (628-2) SHT 25
ICNKXOR (741-4) SHT 18
ICQCUE 0 (0-3) SHT 25
ICQCUE 1 (0-3) SHT 26
SET CURRENT QUEUE POSITION TO 01; 1→BOGUSB
(¬)ICBOGUSA (439-5/4) SHT 18
CLEAR KAFUL AND KAPRV
ICQKAFUL (PIN 365) SHT 25
ICQKAPRV (PIN 267) SHT 26
(¬)ICBOGUSB (437-5/4) SHT 18
ICQKBFUL (PIN 366) SHT 25
ICQKBPRV (PIN 166) SHT 26
CLEAR KBFUL AND KBPRV
ICINCIP, (438-4) SHT 19
ICQIP (0,1), (522-2/519-2) SHT 23
ICQBSY (0-3) SHT 21
ICQACT (0-3) SHT 24
INCREMENT INPUT QUEUE POINTER; SET BUSY AND ACTIVE FOR CURRENT QUEUE POSITION
I4CMREQ
I4HDCORE
TOGGLE ACCESS REQUEST TO MCU;
CLEAR WRITE;
SET PROTECT MODE TO 11
ICQAR:2, (PIN 220) SHT 21
¬ICWRITE, (PIN 152) SHT 21
ICQPRM:2 (0,1) (PINS 320, 219) SHT 21
ICBOGUSA (439-4) SHT 18
BOGUSA
YES
NO
ICGACT(0-3) SHT 20
CLEAR ACTIVE BIT FOR QUEUE POSITIONS EQUAL TO 00
ICQACT (0-3) SHT 24
ICBOGUSB (437-4) SHT 18
BOGUSB
YES
NO
ICGACT (0-3) SHT 20
CLEAR ACTIVE BIT FOR QUEUE POSITIONS EQUAL TO 01
ICQACT (0-3) SHT 24
BEGIN QUEUE OUTPUT EXAMINATION
ICACTO, (310-4) SHT 24
ICCMFUL (PIN 176) SHT 27
KCM FULL
YES
NO
SELECTED OUTPUT QUEUE ACTIVE
YES
NO
D
4
(QUEUE DECODE)
E
4
NO
ICPRVO (PIN 364) SHT 24
PROTECT VIOLATION IN SELECTED OUTPUT QUEUE
YES
NO
SELECTED OUTPUT QUEUE ACTIVE
ICACTO, (310-4) SHT 24
YES
C
1
(START)
F
4
(QUEUE DECODE)
(B) 123948

Figure B-10. Central Memory Requester (Sheet 3 of 4)

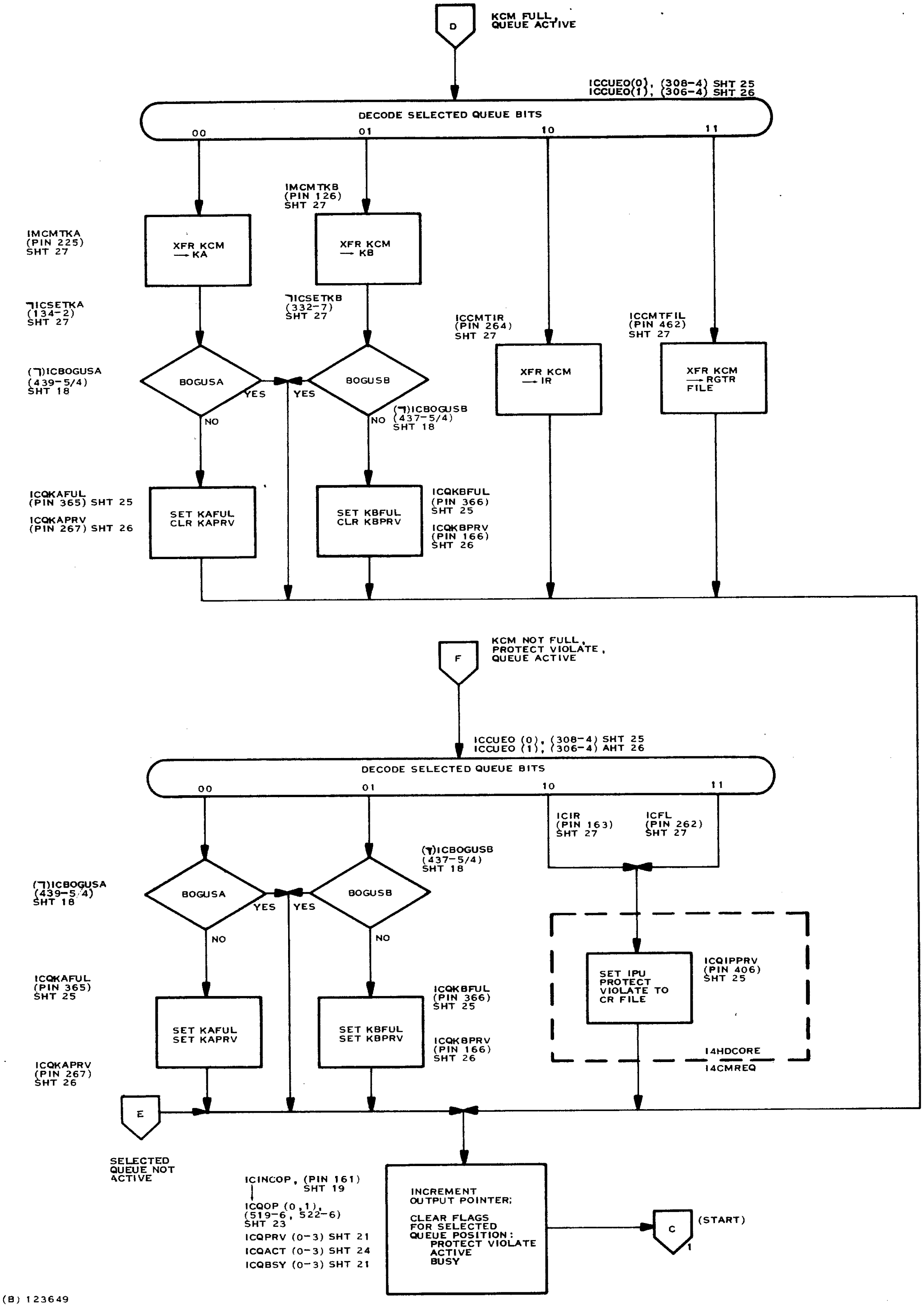

D
KCM FULL, QUEUE ACTIVE
ICCUEO(0), (308-4) SHT 25
ICCUEO(1), (306-4) SHT 26
DECODE SELECTED QUEUE BITS
00
01
10
11
IMCMTKB (PIN 126) SHT 27
IMCMTKA (PIN 225) SHT 27
XFR KCM → KA
XFR KCM → KB
¬ICSETKA (134-2) SHT 27
¬ICSETKB (332-7) SHT 27
ICCMTIR (PIN 264) SHT 27
ICCMTFIL (PIN 462) SHT 27
XFR KCM → IR
XFR KCM → RGTR FILE
(¬)ICBOGUSA (439-5/4) SHT 18
BOGUSA
BOGUSB
YES
YES
NO
NO
(¬)ICBOGUSB (437-5/4) SHT 18
ICQKAFUL (PIN 365) SHT 25
ICQKAPRV (PIN 267) SHT 26
SET KAFUL CLR KAPRV
SET KBFUL CLR KBPRV
ICQKBFUL (PIN 366) SHT 25
ICQKBPRV (PIN 166) SHT 26
F
KCM NOT FULL, PROTECT VIOLATE, QUEUE ACTIVE
ICCUEO (0), (308-4) SHT 25
ICCUEO (1), (306-4) AHT 26
DECODE SELECTED QUEUE BITS
00
01
10
11
ICIR (PIN 163) SHT 27
ICFL (PIN 262) SHT 27
(¬)ICBOGUSB (437-5/4) SHT 18
(¬)ICBOGUSA (439-5/4) SHT 18
BOGUSA
BOGUSB
YES
YES
NO
NO
ICQKAFUL (PIN 365) SHT 25
SET KAFUL SET KAPRV
SET KBFUL SET KBPRV
ICQKBFUL (PIN 366) SHT 25
ICQKBPRV (PIN 166) SHT 26
ICQKAPRV (PIN 267) SHT 26
SET IPU PROTECT VIOLATE TO CR FILE
ICQIPPRV (PIN 406) SHT 25
14HDCORE
14CMREQ
E
SELECTED QUEUE NOT ACTIVE
ICINCOP, (PIN 161) SHT 19
ICQOP (0,1), (519-6, 522-6) SHT 23
ICQPRV (0-3) SHT 21
ICQACT (0-3) SHT 24
ICQBSY (0-3) SHT 21
INCREMENT OUTPUT POINTER;
CLEAR FLAGS FOR SELECTED QUEUE POSITION: PROTECT VIOLATE ACTIVE BUSY
C
1
(START)
(B) 123649

Figure B-10. Central Memory Requester (Sheet 4 of 4)

octet to supply instructions, it signals CMR to toggle the output pointer to select the buffer that contains the look ahead octet. When CMR detects this signal, it inverts the output of the KRTAG flip flop and loads the flip flop with its inverted output. The output of the KRTAG flip flop then selects the other instruction buffer.

HARD CORE REQUEST

If a hard core operation is to be performed in the IPU, CMR determines if the operation will require a memory request. If no memory fetch or store is required, CMR performs no function in the hard core operation. CMR then continues with the control cycle. If the hard core operation involves an access to memory, CMR toggles the access request bit to the MCU indicating that an address is available for the MCU, generates an OA Busy signal to inhibit any further instructions from using the OA register, and sets the protect mode bits to a code of "11". This code designates to the MCU that the memory request will be restricted to the area in memory defined by the execute protect bits in the MCU protect registers. For a store operation CMR sets the Write flag to the MCU; for read operations, the Write flag is cleared.

OA BUSY

If the OA Busy flag is set, then CMR is currently processing a previous memory operation and cannot accept a new memory request. The control circuitry bypasses any new requests until the OA Busy flag clears.

READ PROTECT VIOLATION

If a memory read request initiated during the previous control cycle attempted to access an address in memory that was outside of the area defined by the read protect bits in the MCU, a read protect violation signal from the MCU informs CMR of the violation. So that the error can be recognized when the request is drawn from the memory queue, CMR sets a protect violation bit that corresponds to the queue position of the last request (current input queue count less one).

MEMORY REQUEST QUEUE FULL

If the memory request queue is full, CMR cannot accept further requests until a space opens in the queue. The control circuitry bypasses any new requests until the queue full indication is removed. If the queue is not full, CMR indicates to the look ahead controller that it can accept a new read request by producing the CMR ready indication (ICRDACK1).

MEMORY REQUEST QUEUE EMPTY

CMR cannot accept a write request to memory until all previous read requests have been satisfied. Therefore, if the queue is empty, CMR sets the write acknowledge signal (ICWACK1) to allow write requests. If the queue is not empty, CMR continues with the control cycle to check for memory read requests.

STORE FILE

The only IPU store operations that are not performed through the hard core circuits are the Store File operations that transfer the contents of the register file octets to an area in memory. For any of these instructions, CMR transfers the storage address from the AR register in level 3 to the OA register for transfer to the MCU. It then toggles the access request bit to the MCU, sets the Write flag and OA Busy, and transmits a protect mode of "01" to enable the write protect circuits in the MCU. If a store file operation is not indicated, CMR continues to inspect for other types of memory requests.

INSTRUCTION HAZARD RECOVERY OR BRANCH TO OA

CMR reacts identically to either a branch instruction to an address in central memory or an instruction hazard recovery. Both situations require the IPU to start a new chain of instructions and discard all instructions that reside in the IPU. CMR, therefore, reacts by reinitializing the control circuits and flags. It clears the KRTAG flip flop so that the new instruction sequence will be taken from the KA buffer, and also clears the protect violate and full flags associated with the buffers since the instructions in the buffers are to be disregarded. In addition CMR sets the code, "00" into the queue position indicated by the input pointer to indicate that the new octet will be loaded into the KA buffer. It then sets the protect mode to "11" to indicate to the MCU that the new octets are to be drawn from the area in memory defined by the execute protect registers. The look ahead controller ensures that the proper address has been loaded into OA, so that when CMR toggles AR to the MCU and clears the Write flag, the instruction octet will be fetched from the proper address. Concurrent with these actions, CMR increments the queue input pointer to prepare for the next request, and sets the Busy and Active bits that correspond to the queue position that was filled by the executed request. The controller then continues through the control cycle.

INDIRECT OR EXECUTE

An indirect address or an execute instruction require the IPU to fetch a new word from memory before proceeding with processing. CMR responds to either of these conditions by transferring the memory address from the AR register in level 3 to the OA register for transmission to the MCU, and by setting the protect mode bits to "11", indicating to the MCU that the request is to be subject to the confinements specified in the execute protect parameters. CMR sets the code "10" in the queue position corresponding to this request, so that when the fetch is complete and the instruction is in KCM, the word will transfer directly to IR to continue the program sequence. Having made these preparations, CMR toggles access request to the MCU and clears the Write flag to notify the MCU of a read request. Concurrent with the request, CMR increments the input queue pointer to prepare for the next request, and sets the Busy and Active flags that correspond to the queue location that was filled by the executed request. The controller then continues with the control cycle.

LOAD FILE REQUEST

A Load File instruction transfers the contents of a location in memory into one of the octets in the register file. When CMR detects a Load File at level 3, it transfers the address of the octet to be loaded from the AR register to the OA register for transmission to the MCU, and sets the protect mode bits to "10" to indicate that the fetch will be subject to the restrictions defined by the read protect registers. CMR then loads the code "11" into the request queue so that when the octet is read from memory into KCM, the octet will be routed directly into the register file. CMR then toggles access request and clears the Write flag to initiate the read cycle from memory. Concurrent with these steps, CMR increments the queue input pointer and sets the Active and Busy flags for the last queue position.

START MEMORY REQUEST

When the look ahead controller has loaded a new address into OA and is ready to send that address to the MCU, it activates one of three signals that indicate to CMR to initiate a memory request (ILINSTRP, ILINSTR1, ILINSTR2). When CMR receives any of these signals, it determines if the KRTAG flip flop is in the same state as the NEXT signal from the look ahead controller. If the two signals are equal, then the returning octet from the fetch will be placed in the KA buffer. CMR loads a code of "00" into the request queue to indicate the corresponding request is destined for the KA buffer, and generates the BOGUSA signal that clears the KAFUL and KAPRV flags so that the new data can enter the KA buffer. If KRTAG is not equal to NEXT, the instruction octet is destined for the KB buffer; CMR loads a code of "01" into the request queue to indicate that destination. It then generates BOGUSB that clears the KBFUL and KBPRV flags so that the new data may enter the KB buffer.

When these preparations are complete, CMR sets the Busy and Active flags for the current queue location, toggles access request to the MCU, clears the Write flag and sets the protect mode to "11" to indicate that the request will be subject to the restrictions of the Execute protect registers in the MCU. The input pointer for the request queue increments to prepare for the next control cycle.

BOGUSA

If the current control cycle produced a BOGUSA signal, CMR issued a request to memory for an octet that will be placed into the KA buffer, and will supercede previous memory requests for that buffer. CMR therefore disables other outstanding requests that are destined for the KA buffer by clearing the Active flag for all queue positions that contain a code of "00" ("00" indicates a destination of KA buffer).

BOGUSB

If the current control cycle produced a BOGUSB signal, CMR issued a request to memory for an octet that will be placed into the KB buffer, and will supercede previous memory requests for that buffer. CMR therefore disables other outstanding requests that are destined for the KB buffer by clearing the Active flag for all queue positions that contain a code of "01" ("01" indicates a destination of KB buffer).

KCM FULL

KCM Full indicates that a new octet of data from memory has entered the IPU and is residing in the memory interface file, KCM, on the I4FILEMB. When it receives the full indication, CMR then checks the queue location indicated by the output pointer to determine if the active bit is set. If it is not set, the queue position is not valid. CMR increments the output pointer and clears all flags associated with the queue location that was just checked. CMR will perform an inspection of the next output location on the next control clock. If the selected queue is active, the control circuits proceed to the queue decode circuit.

PROTECT VIOLATION IN SELECTED QUEUE LOCATION

If a new octet did not appear in KCM during the control cycle (KCM not full), CMR determines if a protect violation occurred during the memory fetch for the request that is represented by the current queue location. If no protect violation occurred, the control circuit returns to the beginning of the cycle and waits for the next control clock. If a protect violation did occur, but the current queue location is not active, CMR increments the output pointer and clears the flags associated with that queue location to prepare for the next control cycle. If the selected queue location is active, CMR proceeds to decode the bits in the queue to determine a course of action.

QUEUE DECODE - KCM FULL, QUEUE ACTIVE

The request queue contains a two-bit code to indicate the destination of the octet in KCM. This portion of the queue decode circuit examines the code and generates the gating signals to transfer a valid instruction octet from KCM to the indicated register in the IPU. The circuit produces the following actions under the indicated conditions:

queue = 00 - This indicates that the octet is to be transferred to the KA buffer. CMR generates IMCMTKA to the I4FILEMB to load the KCM octet into the KA buffer. If a BOGUSA signal has been produced during the control cycle, another octet of data will supercede the octet just loaded into KA. If BOGUSA was not produced, the transferred octet is valid. CMR then sets KAFUL and clears KAPRV flags to indicate the presence of a valid octet in the KA buffer.

queue = 01 - This indicates that the octet is to be transferred to the KB buffer. CMR generates IMCMTKB to the I4FILEMB to load the KCM octet into the KB buffer. If a BOGUSB signal has been produced during the control cycle, another octet of data will supercede the octet just loaded into KB. If BOGUSB was not produced, the transferred octet is valid. CMR then sets KBFUL and clears KBPRV flags to indicate the presence of a valid octet in the KB buffer.

queue = 10 - This indicates that a word from the KCM octet is to be transferred to the IR register. CMR generates ICCMTIR to the I4FILEMB to enable KCM to supply instructions to the IR selection circuits. These circuits on the I4FILE circuit board select the word from the KCM octet for input to the IR register.

queue = 11 - This indicates that the octet in KCM is to be stored in the register file on the I4FILE circuit boards. CMR generates ICCMTFIL to the I4FILEMB to enable the output from KCM to the register file input selection. Gating circuits on the I4FILE circuit boards determine which of the octets in the register file will be the final destination of the KCM octet.

Regardless of the code in the queue, CMR completes each control cycle by incrementing the output pointer to indicate the queue location for the next control cycle, and clears all flags associated with the processed queue location so that that queue location may be used by the input pointer to account for another request to memory.

QUEUE DECODE - KCM NOT FULL, PROTECT VIOLATE

If a protect violation occurred during the fetch cycle for the current output queue location, then no data will have been transferred from memory to KCM. This portion of the decode circuit examines the queue code and flags the protect violation as required by each circumstance so that the IPU may run efficiently:

queue = 00 - If BOGUSA has not been generated, the controller sets KAFUL flag to allow the IPU to continue to draw instructions from the buffer even though the data in the buffer is not valid. In addition, CMR sets the KAPRV flag, so that any instructions drawn from KA will be rejected when they reach level 3. If BOGUSA was generated during the control cycle, a replacement octet for the KA buffer has been requested. CMR ignores the protect violation and does not set the KAFUL flag so that the IPU will wait for the replacement octet to reach KA before drawing new instructions from that buffer.

queue = 01 - If BOGUSB has not been generated, the controller sets KBFUL flag to allow the IPU to continue to draw instructions from the buffer even though the data in the buffer is not valid. In addition, CMR sets the KBPRV flag, so that any instructions drawn from KB will be rejected when they reach level 3. If BOGUSB was generated during the control cycle, a replacement octet for the KB buffer has been requested. CMR ignores the protect violation and does not set the KBFUL flag so that the IPU will wait for the replacement octet to reach KB before drawing new instructions from that buffer.

queue = 10 - If the octet for which the protect violation occurred was intended to supply input directly to the IR register, CMR sets the IPU protect violation flag to the PP and does not enable the transfer into IR. Processing halts until the conflict is resolved.

queue = 11 - If the octet for which the protect violation occurred was intended to be stored into the register file, CMR sets the IPU protect violation flag to the PP and does not enable the transfer of data from KCM to the register file.

Regardless of the code in the queue, CMR completes each control cycle by incrementing the output pointer to indicate the queue location for the next control cycle, and clears all flags associated with the processed queue location so that that queue location may be used by the input pointer to account for another request to memory.

MEMORY REQUEST QUEUE AND POINTERS

The memory request queue is a four position queue that accounts for the intended destination(s) of up to four outstanding memory read requests. Each position in the queue contains a two bit code that represents the destinations of the memory requests as follows:

00	KA buffer
01	KB buffer
10	IR register
11	Register file

When CMR initiates a memory request to the MCU, it loads one position in the queue with the destination code for the octet to be fetched with that request. When the requested data enters KCM from the MCU, the associated destination code is drawn from the queue, and decoded to produce the proper gating signals to transfer the octet to its intended location. CMR loads the queue in sequential order, accesses codes from the queue in sequential order, and assumes that data will be returned from memory on a first-in, first-out basis. The request queue input and output pointers hold the two bit address of the input and output queue location to be accessed next. CMR increments the address in each pointer after using the corresponding address so that the sequential order of filling and emptying each position is maintained. When four memory requests are outstanding, a queue full indication prevents CMR from initiating further requests until receipt of data from the first outstanding request clears one of the queue positions.

KA/KB HAZARD DETERMINATION

A hazard to either the KA or KB octets exists when an instruction that has been previously processed alters the contents of the octet in one of the buffers after that octet has been loaded into its buffer. The data in the buffer is therefore not current and should not be used in processing. This circuit detects all such hazards and sets a flag to indicate that the hazard occurs. Because the instruction octet may not be used due to a branch, the hazard is not recognized until the first instruction from the faulty octet reaches level 3. The level 3 controller then recognizes the hazard flag and indicates to the look ahead controller to refetch the octet. The flow chart (figure B-11) illustrates the decisions involved in determining the buffer hazard conditions. All sheet references refer to sheets of the logic diagram set for the I4CMREQ circuit board. The following paragraphs describe the decision paths available in the determination.

ZB EQUAL TO LA. If an address in one of the ZB registers is equal to the octet address in the LA register, then an operation in one of the pipes will store its results into the look ahead octet. The logic then determines which buffer is currently supplying instructions to the pipe. If KA is active, then KB is the look ahead buffer; the controller sets the KB hazard flag. If KA is not active, then KA is the look ahead buffer; the controller sets the KA hazard flag.

STORE FILE AT LEVEL 2. If a store file instruction is at level 2 and the instruction does not transfer the contents of one file to another file (store is to CM), the controller examines the AR register to determine if the store file

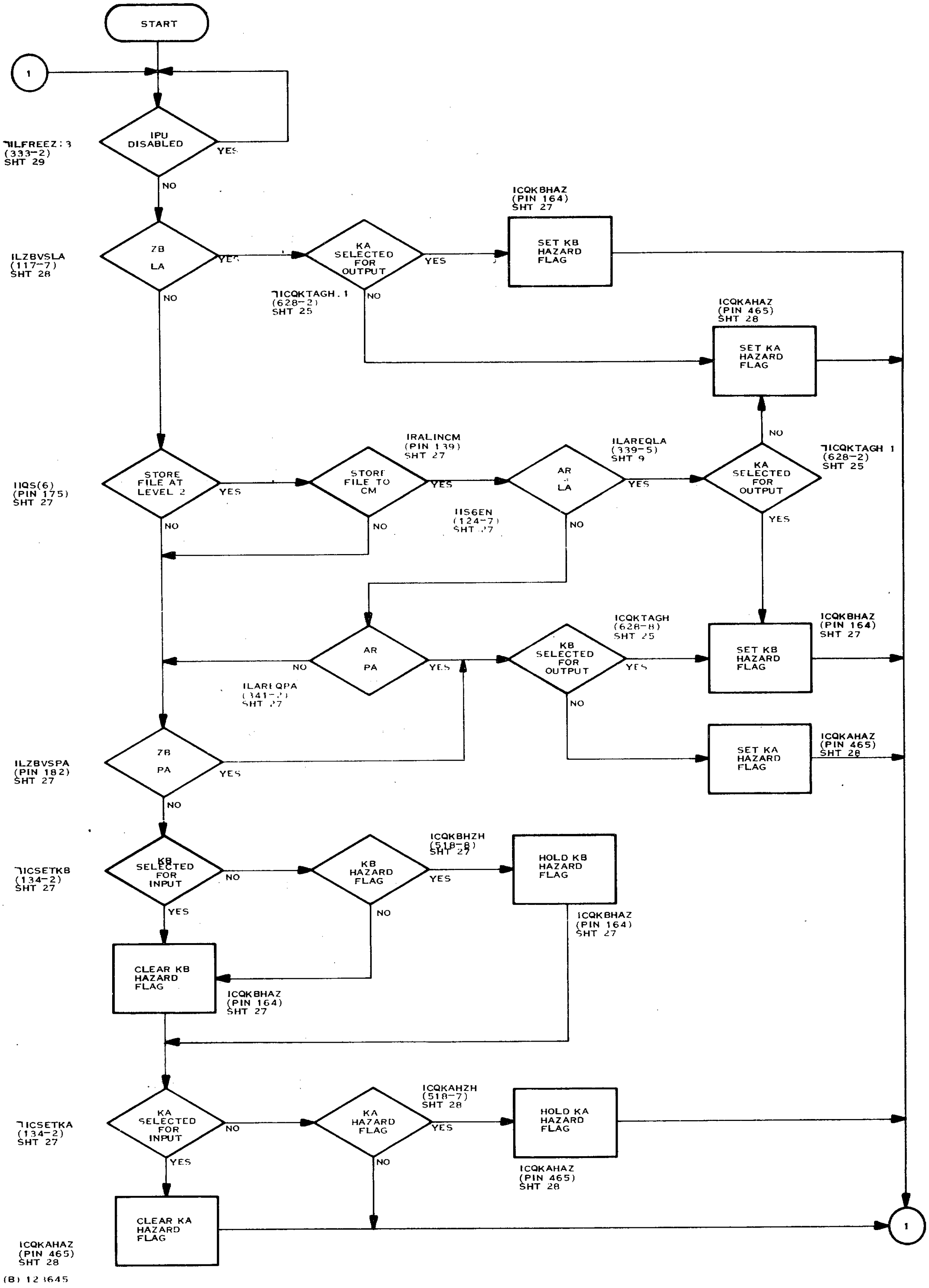


Figure B-11. KA/KB Hazard Determination Flowchart operation will affect one of the buffer files. If AR is contained in the LA octet, then a look ahead octet hazard exists. The controller then sets the appropriate hazard flag that corresponds to the look ahead octet. If the address in AR is equal to the octet address in PA, then a hazard exists for the current instruction buffer. The controller determines which buffer is currently supplying instructions to the pipe and sets the hazard flag that corresponds to that buffer.

ZB EQUAL TO PA. If one of the ZB addresses is contained in the PA register octet, then a current buffer hazard exists. The controller determines which buffer is currently active and sets the hazard flag for that buffer.

KB SELECTED FOR INPUT. If the KB buffer is not receiving a new octet of instructions during the current control clock, and the KB hazard flag has been set by a previously detected condition, the controller prevents the KB hazard flag from clearing. This gating path not only keeps the hazard flag set as long as the faulty octet is in the buffer, but allows the control clock to clear the flag when a new octet is loaded into the KB buffer.

KA SELECTED FOR INPUT. If the KA buffer is not receiving a new octet of instructions during the current control clock, and the KA hazard flag has been set by a previously detected condition, the controller prevents the KA hazard flag from clearing. This gating path not only keeps the hazard flag set as long as the faulty octet is in the buffer, but allows the control clock to clear the flag when a new octet is loaded into the KA buffer.

DETAILS COUNT DECODE

This circuit receives a four bit code (IMOCTR:2(1)) from I4HDCORE and decodes it to produce one of twelve store details select signals, or one of thirteen load details select signals. During a details operation, the code input to this circuit is incremented every clock. The output of the circuit, therefore, steps through counts 0 through 13 (ICLDO(0-13)) if the IMLDTL:2 signal indicates a load details operation, or through counts 0 through 11 (ICSDO(0-11))if the IMSDTL:2 signal indicates a store details operation. The value of the details count is equivalent to the octet number in the IPU details map corresponding to the byte that the count signal enables. The store details count is forwarded to the Store Details Gate circuit where each separate count signal selects five bits from the control circuits to be stored in memory. The load details count fans out to the control circuits on the I4CMREQ circuit board to enable data from the ILQKCM inputs to load parameters into the circuits. The decode of the details count for each of the operations is as follows:

IMOCTR:2(0-3)	IMLDTL:2	IMSDTL:2
0000	ICLDO(0)	ICSDO(0)
0001	ICLDO(1)	ICSDO(1)
0010	ICLDO(2)	ICSDO(2)
0011	ICLDO(3)	ICSDO(3)
0100	ICLDO(4)	ICSDO(4)
0101	ICLDO(5)	ICSDO(5)
0110	ICLDO(6)	ICSDO(6)
0111	ICLDO(7)	ICSDO(7)
1000	ICLDO(8)	ICSDO(8)
1001	ICLDO(9)	ICSDO(9)
1010	ICLDO(10)	ICSDO(10)
1011	ICLDO(11)	ICSDO(11)
1100	ICLDO(12)	
1101	ICLDO(13)	

STORE DETAILS GATE

This circuit receives twelve gating signals from the details count decode circuit during a store details operation. Each of the twelve input gating signals selects five bits from the I4CMREQ circuits and forwards them to I4FILE circuit boards to be stored into memory. The gating signals ICSDO(0-11) correspond to octets 0-11 in the memory block assigned to the IPU details. The output from the I4CMREQ store details gate supplies four bits to word 7 of octets 0-11, and one bit (bit 6) to word 4 of octets 0-11 in the IPU details area of memory. Figure B-12 illustrates the five bits transferred to memory during each details count cycle.

LOAD DETAILS

The load details operation is the reverse of the store details operation. The details count signals enable gates that load the details bits from memory into their respective flip-flops on I4CMREQ. Because of the gating signals for KA and KB buffers are produced on I4CMREQ, two additional counts (12 and 13) are generated during the load details. These two counts enable data from KCM into KA and KB, respectively. Count 11 is sent to I4HDCORE since those bits originate from that circuit board.

					STORE DETAILS COUNT (OCTET) ICSDO(X)						
¬IMDTL7				¬IMDTL4			¬IMDTL4	¬IMDTL7			
4	5	6	7	6			6	4	5	6	7
NOT USED	NOT USED	VACANT LEVELS BIT 0	VACANT LEVELS BIT 1	AU 0 EARLY WINDOW	0	6	AU 2 LATE WINDOW	QUEUE 1 PV	QUEUE 1 ACTIVE	QUEUE 1 BIT 0	QUEUE 1 BIT 1
NOT USED	LA ORDERED	DUAL	TARGET FAIL	AU 1 EARLY WINDOW	1	7	AU 3 LATE WINDOW	QUEUE 2 PV	QUEUE 2 ACTIVE	QUEUE 2 BIT 0	QUEUE 2 BIT 1
PB/LLA IN PROGRESS	FLAG FULL	FLAG 12	FLAG 4	AU 2 EARLY WINDOW	2	8	NOT USED	QUEUE 3 PV	QUEUE 3 ACTIVE	QUEUE 3 BIT 0	QUEUE 3 BIT 1
INPUT POINTER BIT 0	INPUT POINTER BIT 1	OUTPUT POINTER BIT 0	OUTPUT POINTER BIT 1	AU 3 EARLY WINDOW	3	9	NOT USED	KRTAG	KAFUL	KAPRV	KA HAZARD
QUEUE 0 BUSY	QUEUE 1 BUSY	QUEUE 2 BUSY	QUEUE 3 BUSY	AU 0 LATE WINDOW	4	10	NOT USED	NOT USED	KBFUL	KBPRV	KB HAZARD
QUEUE 0 PV	QUEUE 0 ACTIVE	QUEUE 0 BIT 0	QUEUE 0 BIT 1	AU 1 LATE WINDOW	5	11	NOT USED	ARITH. EXCEPT.	PARITY ERROR	ILL. OP. CODE	PROTECT VIOLATE

(A) 123674

Figure B-12. I4CMREQ Store Details Outputs

UNIT REGISTER (UR) DECODE

The UR decode circuit receives a four bit select code from the I4HDCORE circuit board (IMUREN 0-3) and generates seven gating signals to the Unit Register select circuit. Each gating signal enables one eight bit byte to the unit register fanin on the I4HDCORE circuit board. The most significant bit of the select code (IMUREN 0) is an enable bit for the decode circuit. The remaining bits are decoded to generate the gating signals as follows:

IMUREN 1,2,3	Output Signal
000	ICENDTUR(8)
001	ICENDTUR(9)
010	ICENDTUR(10)
011	ICENDTUR(11)
100	ICENDTUR(12)
101	ICENDTUR(13)
110	ICENDTUR(14)
111	No Op

UNIT REGISTER DATA SELECT

The unit register select circuit receives the signals generated by the UR Decode circuit and uses them to produce one of seven possible input bytes (8 bits) to be sent to the CP Unit Register in the PP. Figure B-13 illustrates the bit assignments for the seven unit register bytes that can be gated through this circuit.

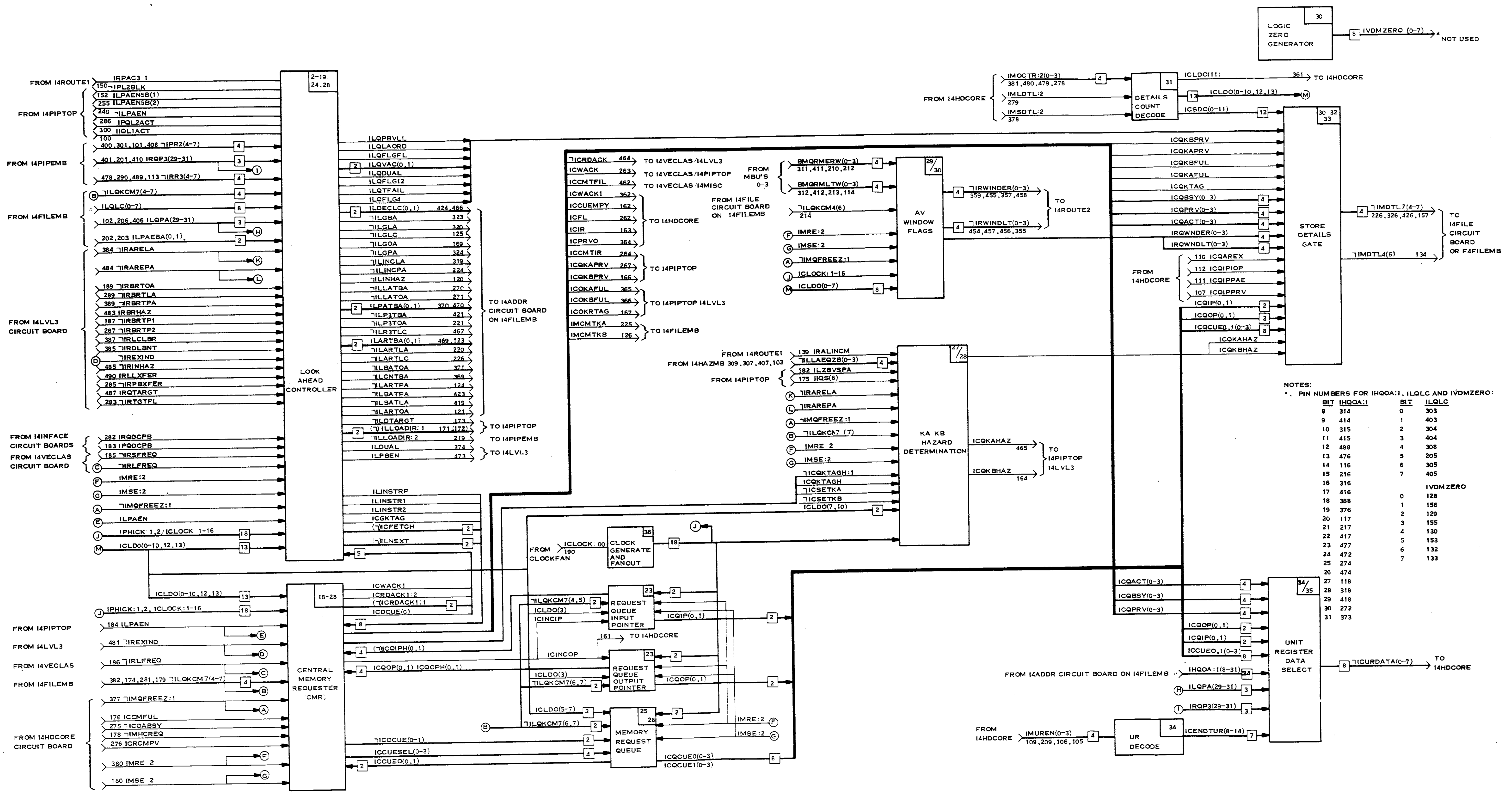
¬ICURDATA BITS

	0	1	2	3	4	5	6	7
ICENDTUR(8)	INPUT POINTER BIT 0	INPUT POINTER BIT 1	OUTPUT POINTER BIT 0	OUTPUT POINTER BIT 1	QUEUE 0 BUSY	QUEUE 1 BUSY	QUEUE 2 BUSY	QUEUE 3 BUSY
ICENDTUR(9)	QUEUE 0 ACTIVE	QUEUE 1 ACTIVE	QUEUE 2 ACTIVE	QUEUE 3 ACTIVE	QUEUE 0 PV	QUEUE 1 PV	QUEUE 2 PV	QUEUE 3 PV *
ICENDTUR(10)	QUEUE 0 BIT 0	QUEUE 0 BIT 1	QUEUE 1 BIT 0	QUEUE 1 BIT 1	QUEUE 2 BIT 0	QUEUE 2 BIT 1	QUEUE 3 BIT 0	QUEUE 3 BIT 1
ICENDTUR(11)	OA BIT 8	OA BIT 9	OA BIT 10	OA BIT 11	OA BIT 12	OA BIT 13	OA BIT 14	OA BIT 15
ICENDTUR(12)	OA BIT 16	OA BIT 17	OA BIT 18	OA BIT 19	OA BIT 20	OA BIT 21	OA BIT 22	OA BIT 23
ICENDTUR(13)	OA BIT 24	OA BIT 25	OA BIT 26	OA BIT 27	OA BIT 28	OA BIT 29	OA BIT 30	OA BIT 31
ICENDTUR(14)	NOT USED	P3 BIT 29	P3 BIT 30	P3 BIT 31	NOT USED	PA BIT 29	PA BIT 30	PA BIT 31

NOTES:

(A) 123673 * PV = PROTECT VIOLATION

Figure B-13. I4CMREQ Unit Register Bytes



NOTES:

*. PIN NUMBERS FOR IHQOA:1, ILQLC AND IVDMZERO:

BIT	IHQOA:1
8	314
9	414
10	315
11	415
12	488
13	476
14	116
15	216
16	316
17	416
18	388
19	376
20	117
21	217
22	417
23	477
24	472
25	274
26	474
27	118
28	318
29	418
30	272
31	373

BIT	ILQLC
0	303
1	403
2	304
3	404
4	308
5	205
6	305
7	405
	IVDMZERO
0	128
1	156
2	129
3	155
4	130
5	153
6	132
7	133

Figure B-14. I4CMREQ Circuit Board Block Diagram

AU WINDOW FLAGS

The I4CMREQ circuit board receives the early and late window bits for each of the four AU's, sets a flag when each window occurs and passes the indication to the I4ROUTE2 circuit board. The transfer results in a one clock delay from the time that the window pulse is produced from the AU ROM in one of the MBU's until the signal is available to the I4ROUTE2 circuit board. The window bits are generated during an instruction sequence when a divide instruction is in the corresponding AU. The early window flag indicates that if there is another divide instruction at level 3 of the same group as the divide in the AU, then the divide in level 3 may be executed (including a memory fetch for data) with an execution time saving. The time saving is due to overlapping of the processes in the AU to avoid the divide initialization time required to start a new divide instruction. The late window flag indicates the same condition as the early window flag, except that for the late window, no time is allowed for memory fetch. The operands must already be within the CP.

Figure B-14 is the I4CMREQ card block diagram.

INTRODUCTION - BLOCK DIAGRAM SYMBOLS

The block diagram included in this circuit board description (figure B-23) condenses the information contained in the logic diagrams to a one sheet representation. The diagram contains all data lines and control signatures that are inputs to or outputs from the circuit board. In addition many of the key data and control lines internal to the board are illustrated. Other information contained on the block diagram is illustrated in figure B-15, and includes:

- Sheet Number - Location of the logic diagram for the depicted function within the circuit board logic set. Multiple sheets are referenced with a letter that is explained in the notes on the diagram.
- Pin Number - Circuit board input/output pin that carries the indicated signal. When more than one pin number appears, the pin for the most significant bit appears first (address), or the pin numbers appear in the order of the signatures listed on the line (control). Large quantities of pins are in tabular form in the notes on the diagram sheet.
- Bus size - Numbers on all lines indicate the number of bits represented by that particular line. A line without a number contains only one bit.

Data and address lines are heavy black lines. Control lines are single width lines.

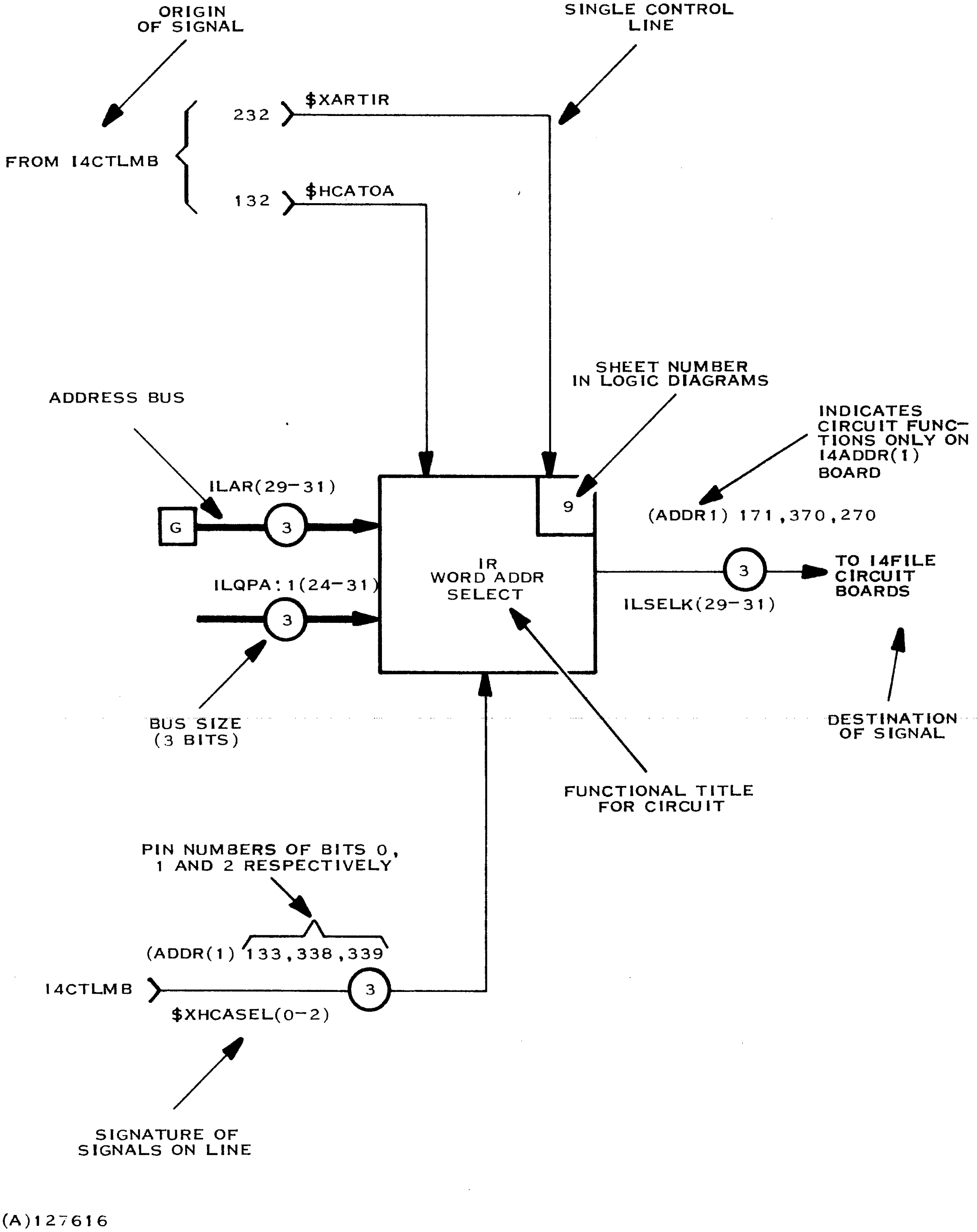


Figure B-15. Key to Symbols - I4ADDR Circuit Board Block Diagram

I4ADDR CIRCUIT BOARD

The I4FILEMB contains two I4ADDR circuit boards that comprise the following major components of the IPU:

OA Output Address Register	-	IHQOA(8-31)
LA Look Ahead Address Register	-	ILQLA(8-31)
PA Present Address Register	-	ILQPA(8-31)
BA Branch Address Register	-	ILQBA(8-31)
LC Look Ahead Counter (decrement)	-	ILQLC(0-7)
IPU Clock Counter	-	IMQCKCNT(8-31) and $XQCKCNT(0-7)
Clock fanout for I4FILEMB	-	$XLOCK04:0(0-19)

The I4ADDR(0) circuit board contains bits 8 through 19 for all registers and bits 0 through 3 for the 31 bit registers; I4ADDR(1) contains bits 20 through 31 for all registers and bits 4 through 7 for the 31 bit registers. Both circuit boards are physically and electrically identical. However, due to the bit division between the boards, a circuit that appears on one circuit board may not be used on the other circuit board.

The block diagram at the end of this section (figure B-23) illustrates the address paths and control signals that are implemented on the I4ADDR circuit boards. In addition to the major components, the circuits include: input gating networks for each register, a branch address comparator, an IR word selector, various holding registers, and the details sequencing and gating circuits required to perform maintenance operations on the IPU. The following paragraphs describe the component circuits on the I4ADDR circuit board and the required control signals to perform the transfers.

INTERFACE SIGNALS

Table B-7 defines the input signals to the I4ADDR circuit boards. Table B-8 defines the output signals from the I4ADDR circuit boards.

Table B-7. I4ADDR Circuit Board Input Signals

Signature	Origin	Function
ICLOCK:0	I4ADDR(1)	Originates as ICLOCK04:0(16 and 17). Control clock pulse for completing transfers between registers.
ILAR(08-31)	I4PIPE	Bits 8 through 31 of the AR register at level 3 of the IPU.
ILARTBA(0,1)	I4CMREQ	Transfers the contents of the AR register to the BA register for I4ADDR circuit boards 0 and 1.
-ILARTLA	I4CMREQ	Transfers the contents of the AR register to the LA register.
-ILARTLC	I4CMREQ	Transfers the contents of the AR register to the LC counter.
-ILARTOA	I4CMREQ	Transfers the contents of the AR register to the OA register.
-ILARTPA	I4CMREQ	Transfers the contents of the AR register to the PA register.
-ILBATLA	I4CMREQ	Transfers the contents of the BA register to the LA register.
-ILBATOA	I4CMREQ	Transfers the contents of the BA register to the OA register.
-ILBATPA	I4CMREQ	Transfers the contents of the BA register to the PA register.
-ILCNTBA	I4CMREQ	Transfers LA + 8 to the BA register.
ILDECLC(0,1)	I4CMREQ	Decrement pulse to the LC counter.
-ILGBA	I4CMREQ	Enables loading an address into the BA register.
-ILGLA	I4CMREQ	Enables loading an address into the LA register.

Table B-7. I4ADDR Circuit Board Input Signals (Continued)

Signature	Origin	Function
-ILGLC	I4CMREQ	Enables loading value into the LC counter.
-ILGOA	I4CMREQ	Enables loading an address into the OA register.
-ILGPA	I4CMREQ	Enables loading an address into the PA register.
-ILINCLA	I4CMREQ	Transfers LA + 8 to LA.
-ILINCPA	I4CMREQ	Enables the present address incrementer to add one to the address in the PA register.
-ILINHAZ	I4CMREQ	Loads the contents of P3 into LA, OA and PA to initialize the operation or recover from a hazard condition.
ILKTIR(8-31)	I4FILE	A word (pointer) from KCM that is loaded into OA at the start of a maintenance command.
-ILLATBA	I4CMREQ	Transfers the contents of the LA register to the BA register.
-ILLATOA	I4CMREQ	Adds eight to the contents of the LA register and transfers that address to the OA register.
ILPATBA(0,1)	I4CMREQ	Transfers the contents of the PA register to the BA register for I4ADDR circuit boards 0 and 1.
-ILP3TBA	I4CMREQ	Transfers the contents of the P3 register to the BA register.
-ILQKCM2:2 (8-31)	I4FILE	Load details inputs to the I4ADDR circuit boards.
-ILR3TLC	I4CMREQ	Transfers the contents of the R3 register to the LC counter.
ILTOGCKI(19)	I4ADDR	Provides interconnection between I4ADDR(0) and I4ADDR(1) within the clock counter circuit. Indicates that bits 20-31 of clock counter are ones.

Table B-7. I4ADDR Circuit Board Input Signals (Continued)

Signature	Origin	Function
ILTOGCKI(31)	I4ADDR	A constant logic "1" input to the least significant bit of the clock circuit.
IMCLEAR	I4HDCORE	Master hardcore clear to the OA register to initialize it for maintenance functions.
-IMCMTOA	I4HDCORE	Gates memory input to the OA register during the start of a maintenance command.
-IMCSTOA	I4HDCORE	Gates a predetermined constant address into the OA register to fetch a pointer during a CR command operation.
IMFREEZE:4	I4HDCORE	IPU disable signal due to a reset RUN bit or other abnormality.
-IMGOA	I4HDCORE	Enables loading of an address into the OA register.
-IMINCOA	I4HDCORE	Selects either OA or LA to supply the address to the octet incrementer to create the next octet address for transmission to the MCU.
IMLDPSDW:2	I4HDCORE	Load Program Status Doubleword. Transfers a word from memory into the clock counter.
IMLDTL:4	I4HDCORE	Load details: enables details sequencer circuit to produce gating signals to the I4ADDR registers to load the registers with memory data.
IMOCTR:4 (0-3)	I4HDCORE	A four bit code that is decoded to produce the gating signals for load and store details.
IMRE:4	I4HDCORE	Master reset for I4ADDR circuits.
IMSDTL:4	I4HDCORE	Store details: enables details sequencer circuit to produce gating signals to the store

Table B-7. I4ADDR Circuit Board Input Signals (Continued)

Signature	Origin	Function
IMSDTL:4 (continued)		details output select to transfer the contents of the I4ADDR registers to memory.
IMSE:4	I4HDCORE	Master Preset for I4ADDR circuits.
IMSTPSDW:2	I4HDCORE	Store Program Status Doubleword. Transfers the contents of the clock counter to a word in the details area of memory.
IRP3(8-31)	I4PIPE	Contents of the P3 register in level 3 representing the address of the instruction at level 3.
IXCLOCK:4	LOGCLK	Clock signal from I4CTLMB (ICCLKOUT:8) for fanout. Used on I4ADDR(1) only.
$XAR(24-31)	I4PIPE	Eight least significant bits of the AR register in level 3 that supply input to the look ahead counter during a Load Look Ahead instruction.
$XARTIR	I4PIPTOP	Originates as ILARTIR on I4CTLMB. This signal is sent to I4ADDR(1) to select the word address from the AR register that designates a word for the IR register. This input is a logic zero for the I4ADDR(0) circuit board.
$XEIGHT	I4HDCORE	Originates as IMEIGHT. This signal is sent to I4ADDR(1) to generate one of the pointer addresses needed to initiate any CCR command to the CP. This input is a logic zero for I4ADDR(0).
$XHCASEL(0-2)	I4HDCORE	Originates as IMHCASEL(0-2). This code is sent to I4ADDR(1) during hard core operations to select a word from KCM into the IR register. These inputs are logic zeroes for I4ADDR(0).

Table B-7. I4ADDR Circuit Board Input Signals (Continued)

Signature	Origin	Function
$HCATOA	I4HDCORE	Originates as IMHCATOA. This signal is sent to I4ADDR(1) to enable $XHCASEL(0-2) to select a word from IR during maintenance commands. This input is a logic zero for I4ADDR(0).
$XKARIN(16,19)	I4ADDR	Originates as ILKARIN(16,19). These two signals interconnect the two I4ADDR circuit boards within the LA incrementer circuit.
$XKARIN(28,31)	-	Constant logic one inputs to the least significant bits of the LA incrementer circuit.
$XKARPA(19)	I4ADDR	Originates as ILKARPA(19). Interconnects the two I4ADDR circuit boards within the PA incrementer circuit.
$XKARPA(31)	-	Constant logic one input to the least significant bit of the PA incrementer circuit.
$XONE	-	Constant logic inputs to the OA register: zero for I4ADDR(0) and one for I4ADDR(1). When enabled, these signals and $XEIGHT generate the pointer addresses needed to initiate any CCR command to the CP.
$XQKCM2:2 (0-7)	I4FILE	Load details inputs to the eight most significant bits of the clock counter circuit.
-$XR3(0-7)	I4PIPE	Bits 0-3 are fixed logic one inputs. Bits 4-7 are the contents of the R3 register at level 3 that load the LC counter during a PB instruction.
$XTOGCKI(3)	I4ADDR(1)	Originates as $XILTOGCKO(4). Connects I4ADDR(1) to I4ADDR(0) within the clock counter circuit. Indicates that bits 4-31 of the counter are ones.

Table B-7. I4ADDR Circuit Board Input Signals (Continued)

Signature	Origin	Function
$XTOGCKI(7)	I4ADDR(0)	Originates as ILTOGCKO(8). Connects I4ADDR(0) to I4ADDR(1) within the clock counter circuit. Indicates that bits 8-31 of the counter are ones.
$XTOGLCI	I4ADDR(1)	Originates as ILTOGLCO(1). Connects I4ADDR(1) to I4ADDR(0) within the Look Ahead Counter decrementing circuit. Indicates that bits 4-7 of the LCH register are equal to zero. This input to the I4ADDR(1) circuit board is a constant logic one.

Table B-8. I4ADDR Circuit Board Output Signals

Signature	Destination	Function
IHQOA:1(8-31)	I4CMREQ	The memory address contained in the OA register to be transferred to the PP as unit register data.
IHQOA:2(8-31)	MCU	Requested memory address to the memory control unit.
ILKARIN(7)	-	Not used.
ILKARIN(16)	I4ADDR(0)	Enters as $XKARIN(016). Provides interconnection within the LA incrementer circuit.
ILKARIN(19)	I4ADDR(0)	Enters as $XKARIN(019). Provides interconnection within the LA incrementer circuit.
ILKARIN(28)	-	Not used.
ILKARPA(7)	-	Not used.
ILKARPA(19)	I4ADDR(1)	Enters as $XKARPA(19). Interconnects the two I4ADDR circuit boards within the PA incrementer circuit.
ILPAEQBA (0,1)	I4CMREQ	Indicates to the Look Ahead Controller that the PA address is equal to the BA address for the portion of the address contained on I4ADDR(0,1). Only when both of these signals are true is the total address in PA equal to the total address in BA.
ILQLA:2(8-31)	I4ZHAZ	The look ahead address that is used by the hazard detection circuits to detect a look ahead octet hazard.
ILQLC(0-7)	I4CMREQ	The contents of the Look Ahead Counter that is used by the Look Ahead Controller to determine the position of an imminent branch that has been preceded by a PB or LLA instruction.

Table B-8. I4ADDR Circuit Board Output Signals (Continued)

Signature	Destination	Function
ILQPA:2(8-28)	I4ZHAZ	Present octet address used by the hazard detection circuits to detect a hazard to the current octet.
ILQPA:2 (29-31)	I4CMREQ	Present word address used by the Look Ahead Controller to detect octet boundaries (PA = 7).
ILQPAH:2 (8-31)	I4PIPE	Present instruction address transferred to the P1 register as the instruction moves to level 1.
ILSELK (29-31)	I4FILE	Selects a word from KCM, KA or KB for transfer to the IR register.
ILTOGCKO(8)	I4ADDR(1)	Enters as $XTOGCKI(7). Connects I4ADDR(1) to I4ADDR(0) within the clock counter circuit. Indicates that bits 8-31 are ones.
ILTOGCKO(20)	I4ADDR(0)	Enters as ILTOGCKI(19). Connects I4ADDR(1) to I4ADDR(0) within the clock counter circuit. Indicates that bits 20-31 are ones.
ILTOGLCO(0)	-	Not used
ILTOGLCO(1)	I4ADDR(1)	Enters as $XTOGLCI. Indicates that bits 4-7 of LCH are equal to zero.
-IMDTL2(8-31)	I4FILE	Store details data lines to memory bus.
IMQCCNTH(8-31)	I4PIPE	IPU clock count to the RO register.
-$XDTL2(0-7)	I4FILE	Store details data lines to memory bus.
$XILTOGCKO(00)	-	Not used.
$XILTOGCKO(04)	I4ADDR(0)	Enters as $XTOGCKI(3). Connects I4ADDR(1) to I4ADDR(0) within the clock counter circuit. Indicates that bits 4-31 are ones.

Table B-8. I4ADDR Circuit Board Output Signals (Continued)

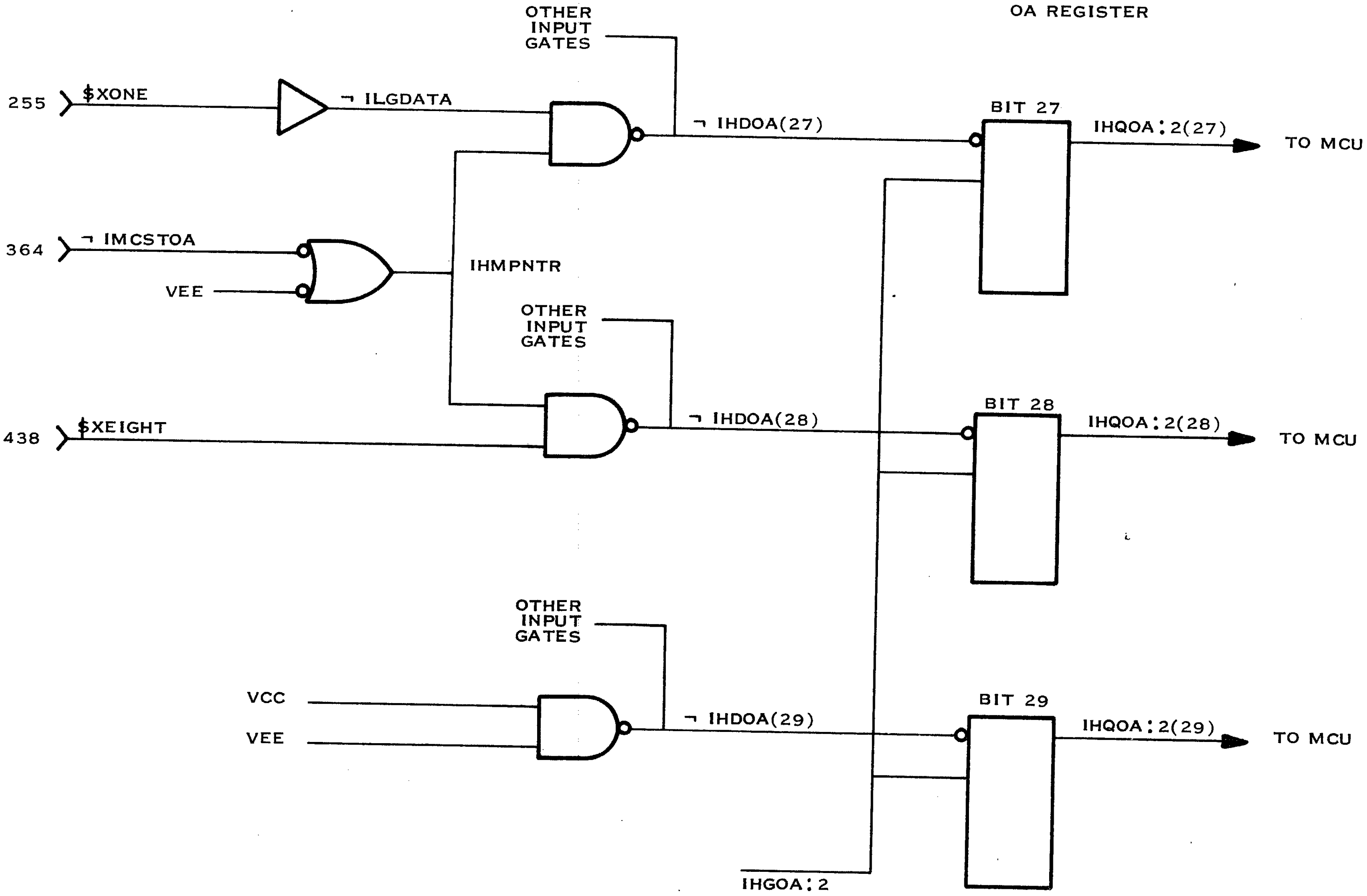
Signature	Destination	Function
$XLOCK04:0 (0-19)	I4FILEMB	Becomes clock input (ICLOCK:04:0(0-19)) to all circuit boards on the I4FILEMB. This output set is used only from the I4ADDR(1) circuit board. Corresponding outputs from I4ADDR(0) are not used.
$XQCCNTH(0-7)	I4PIPE	IPU clock count to the R0 register.

OA REGISTER

The OA (Output Address) register supplies octet addresses (21 bits) to the MCU to designate the area in memory for a store or fetch operation. The Look Ahead Controller (on the I4CMREQ circuit board) controls the gating of addresses into the OA register and initiates requests to the MCU that transfer addresses from OA to the MCU. The output from this register can be transferred to the PP as unit register data, and can also be incremented by eight to form the next sequential octet during load or store maintenance operations. The following paragraphs describe the six inputs to the OA register, their gating signals, and the uses for each input in the operation of the IPU. Either of two gating pulses, -ILGOA or -IMGOA, from I4CTLMB enable the selected input to OA. The CP clock pulse completes the transfer by clocking the selected address into OA. Although the register is implemented with 24 bits, only the 21 most significant bits are effective address bits.

MAINTENANCE COMMANDS. Load, Store and Exchange maintenance commands from the CCR file of the Peripheral Processor require the CP to access a pointer from a fixed location in central memory and use that pointer as the starting address of the data transfer to or from memory. If the maintenance command requires transfer of status, the pointer will be in locations 14 or 15; the pointer for CP or Maintenance details commands is in locations 18 or 19. To initiate the command, the IPU must first access the pointer location, transfer the contents of that location to the OA register, and initiate a request to memory to store or fetch from that location. Two inputs to the OA register enable the IPU to perform this function. The unit hard core controller on the I4HDCORE circuit board generates the gating signals to OA to effect the two transfers.

The octet address of the pointer for status maintenance commands is 10; the octet address of the pointer for CP or maintenance details commands is 18. When the hard core controller receives a maintenance command, it enables two inputs to the OA register bits 27 and 28 (figure B-16) by activating -IMCSTOA. This signal transfers a fixed logic one level ($XONE) to bit 27 and enables an input ($XEIGHT) to bit 28 of OA. If the maintenance operation is a CP or maintenance detail



(A)127617

Figure B-16. Load Pointer Address Gating command, the controller sets $XEIGHT to a logic one level so that both bits 27 and 28 are set. If the maintenance operation is a status command (4108 through 410A), the controller holds $XEIGHT at a zero so that only bit 27 is set. As a result, the controller loads a hexadecimal 10 into OA for status commands, and a hexadecimal 18 for details. When this operation is complete, the controller generates a request to the MCU to access the pointer from the location specified by the address that has been loaded into OA.

When the MCU returns the pointer octet to the KCM memory interface file on the I4FILE circuit boards, the controller selects the pointer word from the octet and transfers it to OA input gating circuit through the ILKTIR(8-31) address bus. The controller then enables that address into the OA register by activating -IMCMTOA. This operation places the starting address of the data transfer into the OA register. The controller then initiates a memory request to the MCU to either store or fetch from that location. If more than one octet is required in the operation, the area in memory will be in consecutive locations. To generate the addresses for the successive octets, the controller activates -IMINCOA to the octet incrementing circuit. This signal enables an output from the OA register (-IHQOA(8-31)) through the octet incrementing circuit and loads the incremented octet address back into the OA register. The controller performs this transfer for each new octet that is required to service the maintenance transfer.

IRP3(8-31). To initiate an operation after a context switch, or to recover from an instruction hazard, the level 3 instruction address from the P3 register (IRP3(8-31)) transfers to the OA register. One control signal (-ILINHAZ) from I4CTLMB gates the P3 address into OA for either of these cases. After a new job has been switched into the IPU, OA contains an address from the switching operation which is not the address of the first instruction in the new job. The context switch transfers the first instruction address into the P3 register. To initiate the new instruction sequence, the hard core controller gates the P3 address into OA (-ILINHAZ) and initiates a memory request for that octet from the MCU.

Similarly, if an instruction hazard occurs, the aborted instruction is allowed to pass to level 3 before the hazard decision is made. At that time P3 contains the address of the hazarded instruction. Therefore, gating the contents of P3 into OA allows the IPU to refetch that instruction from memory to recover from the hazard.

ILQBAH(8-31). This input to the OA register carries the contents of the BA (Branch Address) register. The look ahead controller on the I4CMREQ circuit board determines when this input will be enabled to the OA register by generating the -ILBATOA gating signal. The transfer of BA to OA is performed under these basic conditions:

1. PB or LLA - If a Prepare to Branch or Load Look Ahead instruction has loaded the BA register with the address of a target instruction, and the look ahead controller determines that the expected branch is imminent, the controller transfers the address in BA to OA to fetch the octet containing the branch instruction.

2. Target Fail - If the look ahead controller has fetched the octet containing the target instruction of a branch, but the branch is not taken when it reaches level 3, the controller transfers the contents of BA (address of executing instruction sequence before the target instruction octet was fetched) to recover the previous instruction sequence.

3. Dual/Branch not taken - If the look ahead controller is operating in the Dual mode, but the branch is not taken when the holding hazard is resolved, the controller must recover the look ahead octet that was discarded when entering the Dual mode. The address of the look ahead octet is stored in BA, so the controller transfers the contents of BA to OA to refetch the look ahead octet.

ILAR(8-31). The ILAR input carries the contents of the AR register at level 3 of the IPU. The look ahead controller on the I4CMREQ circuit board determines when this input will be enabled to the OA register by generating the -ILARTOA gating signal. The look ahead controller transfers BA to OA under five basic conditions:

1. Branch to Memory - If a branch instruction at level 3 references an address that is not in the IPU octets, the controller transfers the address of the target instruction from the AR register to the OA register to access the octet containing the instruction.

2. PB - If a Prepare to Branch instruction is at level 3, and the target branch of the PB is either in or is entering the current octet, the controller transfers the contents of the AR register (containing the address of the target instruction) to the OA register to fetch the target octet from memory.

3. Dual - When the look ahead controller enters the dual mode, it transfers the address of the target instruction from the AR register in level 3 to the OA register to access the target octet for use as the look ahead octet.

4. Load File - If a load file instruction is at level 3, the controller transfers the address of the area in memory from the AR register to the OA register.

5. Store File - If a store file instruction is at level 3, the controller transfers the address of the storage area in memory from the AR register to the OA register.

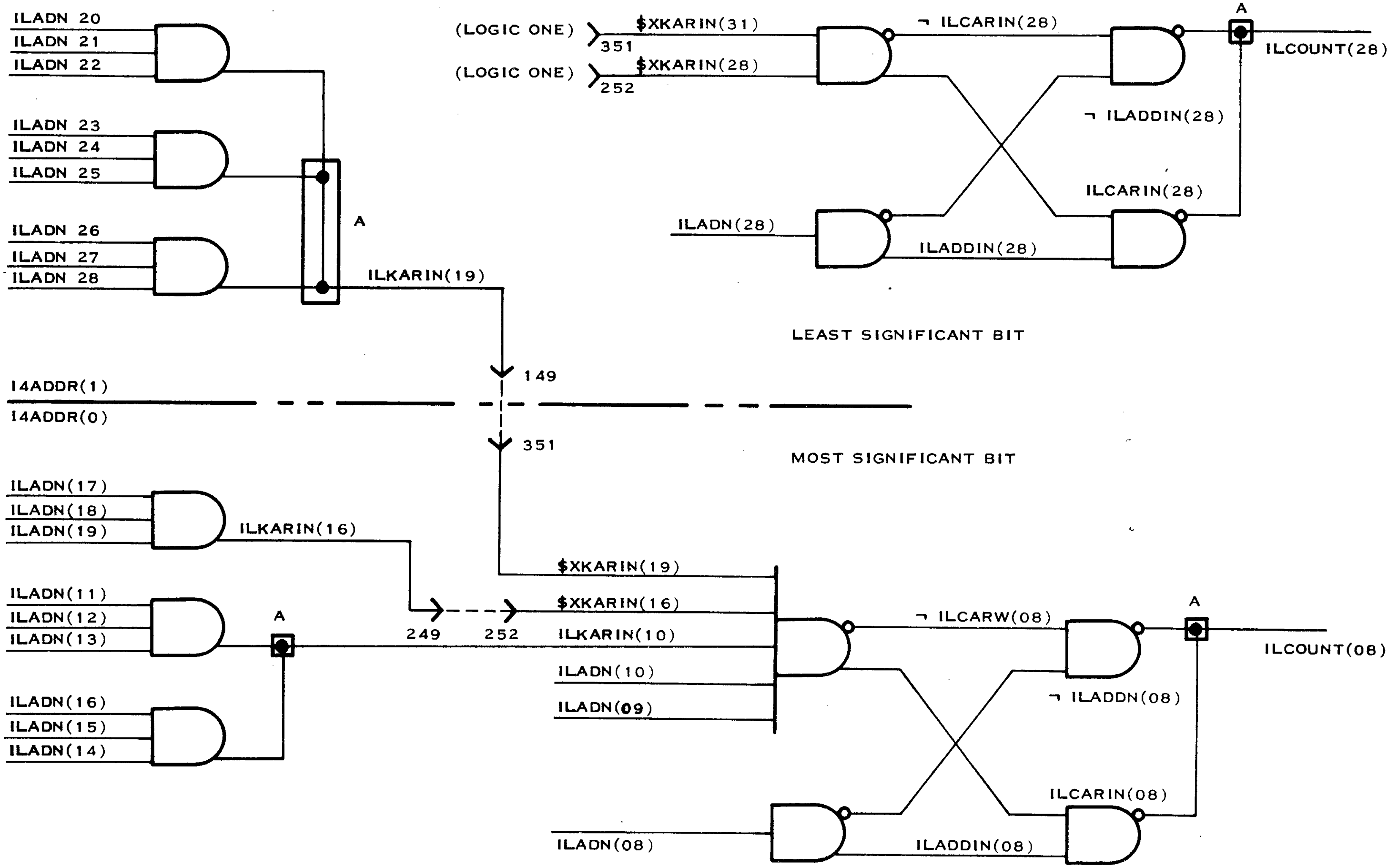
ILCOUNT (8-31). The ILCOUNT bus carries an address from the octet incrementing circuit to the OA register. The address may originate in either the LA or the OA register, but in either case the input represents the next sequential octet to be accessed from memory. During normal look ahead processing, the ILCOUNT lines originate in the LA register. The incrementing circuit adds eight to the word address in the LA register to form the ILCOUNT(8-31) signals. These address bits are normally loaded into LA and OA simultaneously so that when the look ahead address has been sent to the MCU, the address in OA equals the address in LA. The OA register supplies inputs to the octet incrementing circuit only during CCR maintenance command execution.

OCTET ADDRESS INCREMENT

This circuit and its associated input select circuit receive the contents of either the LA or the OA register, add eight to it, and forward the results to the BA LA and OA input select circuits. The control signal, -IMINCOA, determines whether the output from LA or from OA will be used in the incrementer. When this signal is low, OA is incremented; when -IMINCOA is high, LA is chosen, OA is incremented only during maintenance stores and fetches. The increment circuit examines all of the input bits in parallel. If all of the bits that are less significant than the bit under examination are ones, the circuit toggles that bit. Figure B-17 illustrates the circuit that performs this function for bit eight, the most complex, and for bit 28, the least complex. All other bits have similar circuits for examining the bits preceding them. All bits 9 through 28 must be ones before bit 8 toggles; bit 28 will toggle everytime due to the fixed one inputs on $XKARIN(31) and $XKARIN(28). The input bit $XONE, is a fixed "1" bit for I4ADDR(1) and a fixed "0" bit for I4ADDR(0). This signal disables bits 29, 30 and 31 on I4ADDR(1) so that the address from the incrementer is always an even octet boundary (29, 30 and 31 equal to "0"). This signal I4ADDR(0) enables bits 17, 18 and 19 to follow the incrementer circuit output.

LA REGISTER

The LA (Look Ahead) register is a 24 bit register that holds the octet address (21 most significant bits) for the look ahead octet. During normal processing, this address is equal to the address contained in OA that is being accessed from memory. When memory accepts the request, the look ahead controller selects the output from LA into the octet incrementer and enters the incremented address into LA and OA simultaneously. A second register associated with LA (LAH) receives the contents of LA one-half clock time after it enters LA. The output from this register is available for transfer to the BA register when the look ahead controller enters the Dual mode. The output from the LA register is also transferred to the hazard detecting circuits for determination of a look ahead hazard, and is available to the details gating circuit for storage into memory during a store details maintenance operation. -ILGLA from the look ahead controller transfers the inputs to the LA register from the gating circuit. The CP clock pulse completes the transfer. The following paragraphs describe the five inputs to the LA register, their gating signals, and the uses for each input in the operation of the IPU.



(A)127618

Figure B-17. Octet Address Incrementer, MSB and LSB

ENABLE LA INPUTS. Three transfer gating signals to the LA Input Select (-ILINCLA, -ILARTLA and -ILARTLA) cannot enable inputs to the LA register if the LA register is being loaded with data from memory during a context switch or other maintenance loading operation. Therefore, when LA is being loaded for a maintenance operation, IMCMTLA disables the three gating signals to prevent input conflicts. The input gates are enabled at all other times.

ILCOUNT (8-31). This input supplies addresses to LA during sequential octet addressing in normal look ahead mode. The input consists of the address that was previously in the LA register plus the octet increment value of eight. Whenever LA + 8 is loaded into the OA register, this path is also enabled to store the address sent to OA and provide an input to the octet incrementer to generate the next look ahead address.

ILQBAH(8-31). The ILQBAH lines transfer the contents of the Branch Address (BA) register to the LA register. The look ahead controller on the I4CMREQ circuit board determines when this input is enabled to LA by generating the -ILBATLA gating signal. The look ahead controller description explains in detail the conditions required to generate this signal. In general, the output of the BA register is transferred to the LA register under the following conditions:

1. Target Fail - When the look ahead controller has prepared the IPU to take a branch and the branch is not taken when it reaches level 3, the controller transfers the address in BA (recovery address) to the LA register. That address will be incremented and loaded into OA to form the next look ahead address after the recovery address has been sent to the MCU from OA.
2. Dual and Branch not taken - If the look ahead controller is in dual mode, but the branch at level 3 is not taken when the hazard is resolved, the controller transfers the recovery address in BA to the LA register. That address will be incremented and loaded into OA to form the next look ahead address after the recovery address is sent to the MCU from OA.
3. Branch - When a PB or LLA instruction has prepared the look ahead controller, and the controller detects the impending

branch instruction in the PA octet or in the pipe, the controller transfers the address of the target instruction from BA into LA (as well as OA). The address will be incremented and loaded into OA to form the next look ahead address after the address of the target instruction has been sent to the MCU.

ILAR(8-31). The ILAR bus carries output from the AR register at level 3 of the IPU. The AR register holds a developed address of a target instruction when either a branch or a prepare to branch instruction is at level 3. The look ahead controller determines when to transfer this address into the LA register. This determination is explained in detail in the discussion of the look ahead controller. In brief, the controller transfers the address in AR into the LA register under the following conditions:

1. Branch to OA - When a branch instruction at level 3 references an address that is not in the IPU, the controller transfers that address to the LA register (in addition to the OA register). This ensures that the proper address will be incremented and loaded into OA to fetch the next look ahead octet following the target octet.
2. Branch to LA - When a branch instruction at level 3 references an address in the look ahead octet, the controller transfers the address in AR to the LA register to ensure that the proper LA + 8 octet address will be loaded into OA to fetch the next look ahead octet.
3. Branch to PA - If a branch instruction at level 3 references an address in the currently executing octet, and the next look ahead octet has not been ordered from memory, the controller transfers the address in AR into the LA register to ensure that the proper look ahead address will be loaded into OA to access the next look ahead octet.
4. Prepare to Branch - If a PB instruction is at level 3 and its corresponding branch is in the pipe, the controller transfers the address in AR to the LA register (also to the OA register). This transfer ensures that after the target octet has been fetched, the proper address will be loaded into the OA register to access the next look ahead octet.
5. Initiate Dual - When the look ahead controller enters the dual mode, it transfers the AR address into the LA register to ensure that a sequential look ahead address will be loaded into the OA register after the target octet has been fetched from memory.

IRP3(8-31). These input lines carry the contents of the P3 register. P3 holds the address of the instruction at level 3 of the IPU. If an instruction hazard occurs that renders the instruction at level 3 unreliable, the look ahead controller transfers the address of that instruction from P3 to LA. This transfer ensures that after the instruction octet has been fetched from memory, the proper look ahead address will be created by the octet incrementer and loaded into the OA register to fetch the next look ahead octet.

-ILQKCM2:2(8-31). The contents of LA are loaded from and stored into word 2 of octet 1 in the IPU details map area of central memory. During load details, this input from word 2 of KCM is enabled to transfer the stored contents of LA into the LA register. The IMCMTLA signal from the details sequencer enables this transfer. This signal is produced when the details count from unit hard core is equal to 1, designating octet 1 of the details map. Therefore, this input and its gating signal transfer bits 8 through 31 of word 2 in octet 1 of the details map into the LA register.

PA REGISTER

The PA (Present Address) register contains the 24-bit word address of the next instruction that will be loaded into the IR register. The look ahead controller on the I4CMREQ circuit board controls the loading and incrementing of the address in the PA register. The output from this register selects a word from the current memory buffer for transfer to the IR register, and during a PB or LLA helps determine if the target instruction is in the current octet. The present address also transfers to the I4HAZMB for detecting instruction hazards in the current octet and to the I4CTLMB for determination of octet boundaries (PA=7). A third output from PA transfers to the PAH register. The PAH (PA Holding) register receives the contents of the PA register one-half clock after the address has entered the PA register. This buffering stage isolates the P1 register and the PA incrementer circuit from changes in the PA register during the control clock pulse. The output from the PAH register transfers to the P1 register on the I4PIPEMB when the corresponding instruction transfers to the IR register. The P1 register holds the address of the instruction at level 1 of the IPU. In addition, the output of the PAH register is used by the address incrementer circuit to supply sequential word addresses to the PA register. The gating signal, -ILGPA, from the look ahead controller enables one of five selected inputs to transfer an address into the PA register. The control clock pulse completes the transfer. The following paragraphs describe the five inputs to PA, their gating signals and their use within the IPU addressing circuits.

ILQBAH(8-31). These input lines supply an address from the BA register for input to the PA register. The look ahead controller on the I4CMREQ circuit board determines when to enable this input by generating the -ILBATPA gating signal. The look ahead controller description explains in detail the conditions required to produce this gating signal. Briefly, the BA register supplies inputs to the PA register under the following conditions:

1. Target Fail - If the look ahead controller has prepared the IPU for a branch but the branch is not taken, the controller must recover the old instruction sequence before any further processing is performed. The controller transfers the recovery address from BA to the PA register. When the recovery octet enters the IPU from memory, the PA register will be able to select a word with minimum delay.
2. Branch at Level 1- When a PB or LLA has prepared the look ahead controller to expect a branch instruction and the controller transfers that branch instruction into level 1, the controller loads the address of the target instruction from BA into PA. The next instruction sent to IR will then be accessed from the branch path.

ILAR(8-31). The ILAR lines carry an address from the AR register at level 3. This address is developed through indexing or modification and indicates the location of a target instruction when a branch or prepare to branch instruction is at level 3. The look ahead controller determines when to enable this input and issues the -ILARTPA gating signal. The controller generates this signal under the following conditions:

1. Branch to OA - If a branch instruction at level 3 references an address not in the IPU octets, the controller transfers

the target address to PA to select the target instruction from the octet as it returns from central memory.

2. Branch to LA - If a branch instruction at level 3 references an address in the look ahead octet, the controller transfers the target address from AR into PA to select the target instruction from the look ahead buffer when the KRTAG bit is toggled.

3. Branch to PA - If a branch instruction at level 3 references an address in the current octet, the controller transfers the target address from AR into PA to select the target instruction on the next control clock.

4. Prepare to Branch - If a prepare to branch instruction is at level 3 and its corresponding branch is at level 0, the controller transfers the target instruction address from AR to PA as the branch transfers to level 1. The instructions from the branch path will then follow immediately after the branch instruction in the pipe resulting in a minimum branch delay.

-ILQKCM2:2(8-31). Word 2 of octet 2 in the IPU details map area of central memory supplies and stores the contents of PA for maintenance exchanges. This input bus from the KCM memory interface file loads word 2 into the PA register when IMCMTPA enables the input. The details sequencer issues IMCMTPA when the details count, IMOCTR:4(0-3), decodes to a count of two. This decode indicates that octet 2 of the details map is in KCM so that enabling the load details input to the PA register at that time results in loading word 2 of octet 2 into the PA register.

IRP3(8-31). These input lines carry the contents of the P3 register. This register holds the address of the instruction at level 3 of the IPU. If an instruction hazard occurs that renders the instruction at level 3 unreliable, the look ahead controller transfers the address of that instruction from the P3 register to the PA register to select the hazarded instruction from the refetched octet when it enters the IPU from memory. The IPU then reprocesses the instruction stream from that point in the program.

IIPAINC(8-31). This address bus carries an address that is one greater than the address currently held in the PA register. The input is gated into the PA register after each new instruction transfer to IR so that the next instruction for IR will be from a sequential location in the current octet. The gating signal, IIINCPA, is generated by the look ahead controller on the I4CMREQ circuit board.

PA INCREMENTER

The PA incrementer circuit receives the contents of the PAH register, adds one to that address, and forwards it to the PA and BA registers. This input to PA selects the next instruction word to be loaded into IR. When transferred to BA, the incremented address represents a recovery address during a PB or LLA operation. The incrementer circuit requires no gating or control signals, so that when an address enters the PAH register, the incremented address is available from the incrementer circuit. The incrementer examines all of the input bits in parallel. If all of the bits that are less significant than the bit under examination are ones, the circuit toggles that bit, Figure B-18 illustrates the circuit that performs this function for bit eight, the most complex, and for bit 31, the least complex. All other bits have similar circuits for examining the bits preceding them. All bits 9 through 31 must be ones before bit 8 toggles; bit 31 toggles every time the input address changes due to the fixed one input on $XKARPA(31).

BA REGISTER

The BA (Branch Address) register is a 24-bit storage register that is used in conjunction with branch instruction handling in the IPU. This register does not always contain a branch address, however. The look ahead controller on I4CMREQ controls the choice of inputs to the register. In addition to holding a target instruction address of a branch under control of a PB or LLA instruction, the BA register may also contain a recovery address when the look ahead controller takes a branch path of instructions in preparation for the branch instruction. If the branch is not taken when it reaches level 3, the controller can access the recovery address in the BA register to reconstruct the instruction sequence that was in progress before the PB or LLA diverted the path. The BA register outputs are available to the branch address compare circuit for determining if the target is in the current octet, to the store details gate for transfer

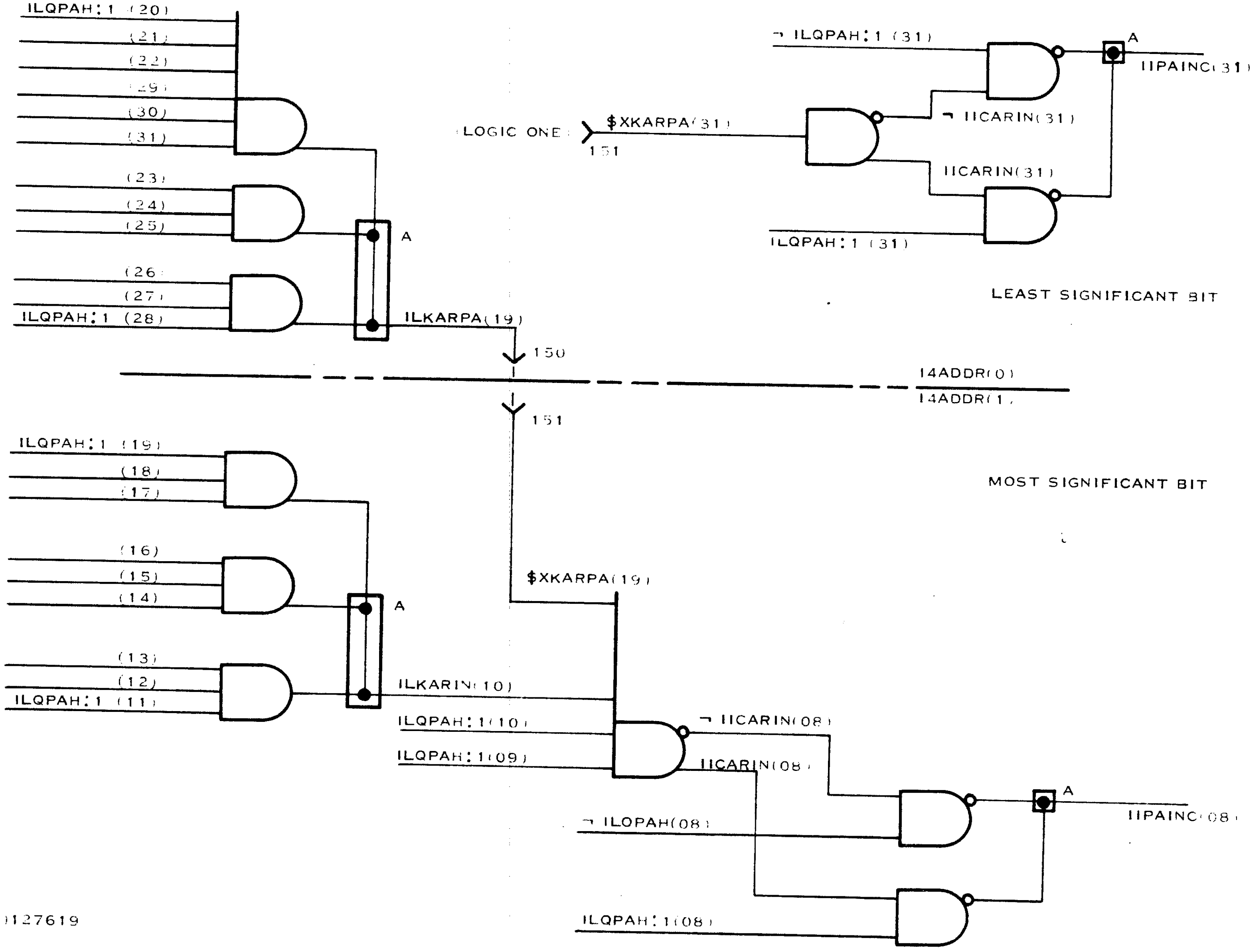


Figure B-18. PA Incrementer, MSB and LSB to memory, and to the BAH (BA Holding) register. BAH receives its address inputs from BA one-half clock time after the address enters the BA register. This buffering level isolates the BA register from the registers that receive inputs from it, so that the same clock pulse can transfer the contents of BA to another register while changing the contents of the BA register. The BAH register supplies outputs to the PA, LA and OA registers. Refer to the discussions of these registers for the conditions that prevail when these transfers occur. Six input busses may supply address inputs to the BA register. The choice of inputs is under control of the look ahead controller. Once the input is selected, -ILGBA and the IPU clock pulse enable the address transfer into the BA register. The following paragraphs describe the six input paths to the BA register, their gating signals, and the function of each input within the operation of the IPU.

IIPAINC(8-31). This input bus carries an address that is one greater than the current address in the PA register. This address is transferred to the BA register as a recovery address. When a PB or LLA instruction has prepared the controller for an upcoming branch instruction and the controller transfers that branch instruction from level 0 to level 1, it also transfers the next address in that instruction sequence (PA + 1) to the BA register. The controller will then access instructions from the branch path and insert them into the pipe following the branch instruction. If the branch instruction is skipped over, or is not taken when it reaches level 3, the recovery address in BA is retrieved to reconstruct the previous instruction path that would have been taken if the branch were not anticipated.

-ILQKCM2:2(8-31). Word 2 of octet 0 in the IPU details map area of central memory supplies and stores the contents of BA for maintenance exchanges. This input bus from the KCM memory interface file loads word 2 into the BA register when IMCMTBA enables the input. This gating signal is produced in the details sequencer circuit. The details sequencer issues IMCMTBA when the details count, IMOCTR:4(0-3), decodes to a count of zero. This decode indicates that octet 0 of the details map is in KCM so that enabling the load details input to the BA register at that time results in loading word 2 of octet 0 into the BA register.

IRP3(8-31). These input lines from the P3 register carry the address of the instruction that is currently at level 3. The look ahead controller transfers this input to the BA register as a branch address when a Load Look Ahead instruction is at level 3. An LLA instruction prepares the IPU to loop back to the LLA instruction location in the instruction sequence when the branch instruction is encountered. Therefore, when the LLA reaches level 3, the controller issues ILP3TBA to transfer the address of the LLA from P3 to BA for storage. When the branch instruction reaches level 1, the address in BA transfers to the addressing registers so that the instruction sequence will begin again with the LLA instruction.

ILAR(8-31). The ILAR lines carry the address contained in the AR register at level 3. This address is developed through indexing or modification, and indicates the location of a target instruction when a prepare to branch instruction is at level 3. The look ahead controller generates the gating signal ILARTBA to transfer this address to the BA register whenever a PB instruction reaches level 3 and is enabled. This results in storing the target instruction address. When the branch instruction is loaded into level 1, the controller accesses the target instruction address from BA to select instructions from the branch path.

ILCOUNT(8-31). The ILCOUNT bus supplies an address that is eight greater than the address contained in the LA register. If a look ahead octet has not yet been ordered from central memory, this address is eight greater than the currently executing address, or is equivalent to the address of the look ahead octet. When the look ahead controller enters the dual mode of operation, it retains the current instruction octet in the active instruction buffer (KA or KB) and fills the other instruction buffer with an octet from the branch path, so that the IPU will be ready to execute from either path. In doing this, the controller must discard or ignore the look ahead octet for the current instruction sequence. In order to reclaim the look ahead octet if the branch is not taken, the controller generates -ILCNTBA to store the LA + 8 address into the BA register if the look ahead octet has not been ordered from memory. If the branch is not taken, this address may be accessed from BA to fetch the look ahead octet from memory to reconstruct the instruction sequence.

ILQLAH(8-31). This input bus supplies the address that was in the LA register before the last control clock. If the look ahead octet has been ordered from memory, this address represents the address of the look ahead octet. When the look ahead controller enters the dual mode, it retains the current instruction octet but discards the look ahead octet. So that it may reconstruct the instruction path if the branch is not taken, the controller generates -ILLATBA to transfer the look ahead address in LAH to the BA register if the look ahead octet has been ordered. If the branch is not taken, then the controller can access the address in BA to continue the original instruction sequence.

BRANCH ADDRESS COMPARE

The branch address compare circuit examines the outputs from the BA register and the PA register to determine if the octet address in BA is equal to the octet address in PA. The circuit compares the true output of PA with the false output of BA, and the false output of PA with the true output of BA. If neither of these comparisons are true for each bit of the octet address, then the address in PA is equal to BA. Two separate signals, one for the address bits on I4ADDR(0) and one for the address bits on I4ADDR(1) are sent to the look ahead controller on the I4CMREQ circuit board. The look ahead controller combines the two signals to arrive at the final determination that BA is equal to PA. Although all 24 bits of each register enter the comparison circuit, the circuit compares only the 21 most significant bits (the octet address). A hard wired logic one signal ($XONE) to the I4ADDR(1) circuit board disables the comparison of bits 29, 30 and 31. This input is wired to a logic zero level for I4ADDR(0).

IR WORD ADDRESS SELECT

This circuit receives three word address inputs and selects one of the addresses for output to the I4FILE circuit boards. On the I4FILE circuit board this address (ILSELK(29-31)) chooses a word from KA, KB or KCM to be transferred to the IR register. Although this circuit is implemented on both I4ADDR(0) and I4ADDR(1) circuit boards, the selection for bits 17, 18 and 19 on I4ADDR(0) is not used. The selection of bits 29 through 31 on I4ADDR(1), therefore, is the only functional selection circuit. These bits form the word address portion of the address in their respective registers. The following paragraphs describe the inputs to the selection circuit and the conditions that exist when each input is enabled.

PRESENT ADDRESS. When both of the input gating signals are inactive ($XARTIR and $HCATOA), the present address register, PA, supplies word selection bits to the I4FILE circuit boards. The present address bits enter on the ILQPA:1(29-31) bus. This input supplies word addresses during normal octet processing, since PA normally contains the address of the currently used instruction.

AR REGISTER. When the $XARTIR gating signal becomes active, it gates the word address input from the AR register to the I4FILE circuit boards. The AR register contains an address that the IPU developed through modifications. If the address in AR is an indirect address, the level 3 controller generates $XARTIR to enable the word address field of the AR register to select an instruction word from the KCM, KA or KB octet. The IPU then waits for the new instruction to reach level 3 before continuing. By allowing the AR register to select the indirect word, the IPU is able to maintain the correct current address in the PA register.

HARD CORE WORD SELECT. At the beginning of a maintenance transfer command from the CCR file, the IPU must fetch a pointer from one of four fixed locations in memory. These locations are: 14 for status stores, 15 for status loads, 18 for CP and maintenance details stores, and 19 for CP and maintenance details loads. When the hard core controller detects one of these maintenance commands, it orders the corresponding octet (10 or 18) from memory (refer to OA register description). When this octet arrives in the KCM octet, the controller must then select the proper word from the octet and load that word into the OA register to initiate the transfer to or from memory. The $XHCASEL(0-2) word address input allows the controller to make this selection. Since the correct octet is already in the KCM file, these three input bits only designate the four values required to select the four pointer words from their respective octets. These values, for bits 0, 1, and 2 are:

- 000 - selects word zero from octet 18 for details stores.
- 001 - selects word one from octet 18 (location 19) for details loads.
- 100 - selects word four from octet 10 (location 14) for status stores.
- 101 - selects word five from octet 10 (location 15) for status loads.

Other values for these three bits are not possible, since bit 1 is fixed at a zero value on the I4HDCORE circuit board. To enable these bits to the IR selection circuit on the I4FILE circuit boards, the hard core controller produces the gating signal, $HCATOA. The selected pointer transfers only to the OA register and never to the IR register.

LOAD/STORE DETAILS SEQUENCER

This circuit receives a four bit code (IMOCTR:4(0-3)) from I4HDCORE and decodes it to produce one of five store details select signals, or one of six load details select signals. During a details operation, the code input to this circuit is incremented every clock. The output of the circuit, therefore, steps through the five store details gates (IMSTORBA, IMSTORLA, IMSTORCK, IMSTORLC and IMSTORPA) if the IMSDTL:4 signal indicates a store details operation, or through the six load details gates (IMCMTBA, IMCMTLA, IMCMTPA, IMCMTLC, IMCMTCK:1 and IMCMTCK:2) if the IMLDTL:4 signal indicates a load details operation. The decoded value of the input counter bits, IMOCTR:4(0-3), is equivalent to the octet number in the IPU details map corresponding to the byte that the count signal enables. The store details gating signals are forwarded to the store details output select circuit where each gating signal selects either an 8-bit, 24-bit or 32-bit details transfer to central memory. The load details gating signals fan out to the Circuit board registers to transfer data on the KCM input bus into the registers. The decode of the details count bits for each operation is as follows:

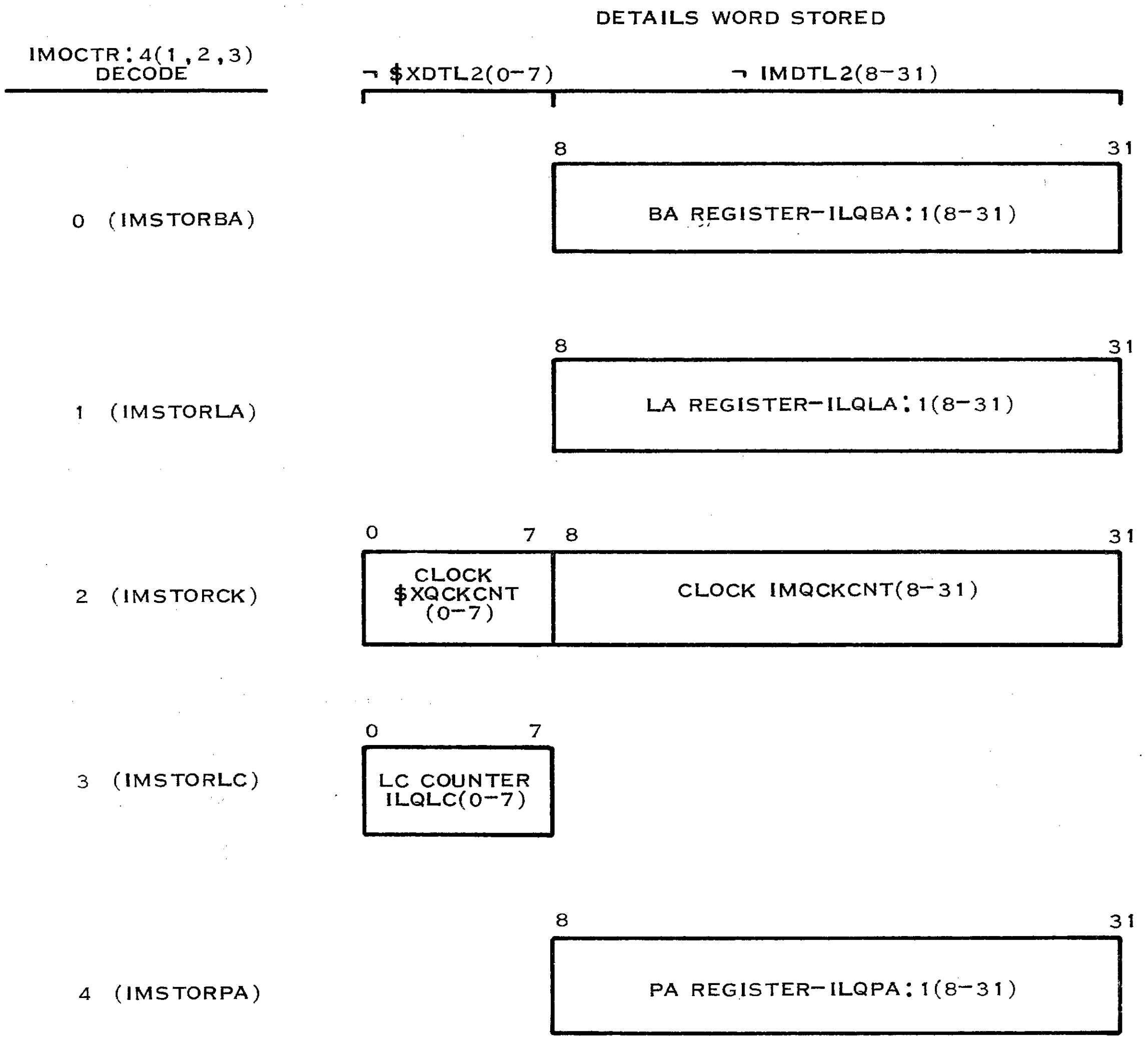
IMOCTR:4(0-3)	IMSDTL:4	IMLDTL:4
0000	IMSTORBA	IMCMTBA
0001	IMSTORLA	IMCMTLA
0010	IMSTORCK	IMCMTCK:1 IMCMTCK:2
0011	IMSTORLC	IMCMTLC
0100	IMSTORPA	IMCMTPA
0101 through 1111	No Op for I4ADDR circuit boards	

STATUS DOUBLEWORD

During a load or store status operation, the hard core controller generates a gating signal (IMLDPSDW:2 or IMSTPSDW:2) to the details sequencer circuit to produce either IMCMTCK:1 and 2, or IMSTORCK. These signals enable the controller to either load or store the contents of the clock counter on the I4ADDR circuit boards as part of the store status operation. Other information in the status doubleword includes the arithmetic exception and mask bits for each AU and the contents of the P3 register.

STORE DETAILS OUTPUT SELECT

This circuit receives five gating signals from the details sequencer circuit during a store details operation. Each of the gating signals selects the contents of one of the I4ADDR registers and places it on the store details bus (-IMDTL2(8-31) and -$XDTL2(0-7)) for storage in central memory. The gating signals, (IMSTORBA, IMSTORLA, IMSTORCK, IMSTORLC and IMSTORPA) correspond to octets 0 through 4, respectively, of the IPU details area in memory. Figure B-19 illustrates the details word segments that are enabled during each count of the details counter code (IMOCTR:4(1,2,3)).



(A)127620

Figure B-19. I4ADDR Details Words

LOAD DETAILS

The load details operation is the reverse of the store details operation. The details gating signals load the memory word segments from the KCM interface file to the registers within the I4ADDR circuit boards. The registers are loaded in the same order that they are stored during the store details operation. Two gating signals, one for each section of the clock circuit, are generated from the same load count.

R3 HOLDING REGISTER

The R3 holding register is an eight bit register that receives the contents of the R3 register with each half-phase clock pulse (IPHICK:4). The register buffers the R3 data so that the contents of R3 may be changed at the same time that the previous contents of R3 are loaded into the LC counter. Since the R3 register is only a four bit register, the inputs to bits 0-3 of the R3 holding register on I4ADDR(0) are wired to a logic one signal. The fixed one input ensures that the four most significant bits of the R3 holding register will always be zeroes. The R3 input loads the LC counter when a Prepare to Branch instruction is at level 3. At that time R3 contains the number of instructions between the PB and the branch instruction. Since R3 is only four bits, the largest number of instructions between the PB and the branch is fifteen.

LC COUNTER

The LC counter is an eight bit, decrementing counter that the look ahead controller uses to track branch instructions that have been preceded by an LLA or PB instruction. The controller loads the LC counter with a value specified by the LLA or PB instruction, and then decrements the count each time an instruction is transferred from one of the buffer files (KA or KB) into the IR register. When the count reaches zero (when adjusted for the number of instructions in levels 1 and 2 at the time of the PB or LLA), the branch instruction has been transferred to the IR register. The controller then fills the positions in the pipe behind the branch instruction with instructions from the branch path. The following paragraphs describe the four major circuits within the LC counter.

LC INPUT GATE. The input gate to the LC counter receives three, eight-bit inputs. Three corresponding control signals determine when each one of the three inputs will be enabled. The output of this circuit loads the eight bit positions in the LC counter. During count 3 of a load details operation, IMCMTLC enables the input from the KCM memory interface file (-$XQKCM2:2(0-7)) to load a value for the new program into the LC counter. If a Prepare to Branch instruction is at level 3, the look ahead controller transfers the look ahead count from the R3 register to the LC counter through the $XQR3H:1(0-7) input from the R3 holding register. If a load look ahead instruction is at level 3, the look ahead controller transfers the contents of the AR register to the LC counter by activating the -ILARTLC signal. Any of these input loads a value into the LC counter that specifies the number of instructions until a branch instruction occurs in the instruction sequence.

LC REGISTER. Eight, type DF flip-flops comprise the LC register. The output of the LC input gate drives one input to each flip-flop. This path loads the counter with an initial count-down value when the look ahead controller issues the -ILGLC control gate. The other data input to the register flip-flops is the output of the LCH register. LCH buffers the output of the LC register so that the contents of LC can be changed with respect to that output. Eight separate gating signals (ILTOGLC(0-7)) enable each individual input to the LC register as required to produce a decrementing count in the LC register. An enabled input to the register results in toggling the respective bit. The output of the LC register transfers to the LCH register, to the details output select, and to the look ahead controller for use in tracking an impending branch in the IPU.

LCH REGISTER. Eight type FF flip-flops comprise the LCH register, four flip-flops per I4ADDR circuit board. The LCH register isolates the output of the LC register from the input to the LC register, so that the output may change the contents of the LC register without the danger of a feed-through transfer causing two or more changes to the LC register during one clock time. The LCH register receives the contents of the LC register one-half clock time later than the data enters the LC register. When the LC contents have entered the LCH register, the output of this register is available to the decrement determination circuit, and to the input of the LC register. The decrement determination circuit examines the bits of the LCH register and decides which input bits to enable to the LC register to produce a decrement of one.

LOOK AHEAD DECREMENT. The look ahead decrement circuit receives a count from the LCH register and examines it to determine which bits to change to produce a decrement by one. When a new instruction enters the IR register, the look ahead controller generates ILDECLC(0,1) to the circuit and the decrementer toggles bits in the LC register to decrement the count. The decrement circuit determines which bits are to be toggled by applying an inspection algorithmn. It examines each bit of the LCH register in parallel. If all the bits that are less significant than the bit under inspection are equal to zero, then the decrementer toggles that bit in the LC register to decrement the count. Figure B-20 illustrates the circuit that performs this function. Bit 7 of the LC register will always be toggled due to the hard wired logic one input on the $XTOGLCI input. Similarly, bit 0 will toggle only if all bits in the LC register preceding it (bits 1-7) are zeroes.

CP CLOCK COUNTER

The CP clock counter circuit is an increment-by-one counter that consists of a register, a holding register and an incrementing gate circuit. The contents of the register are changed to the next higher value with each CP clock pulse, thus maintaining a running total of the number of clock pulses that have occurred since the clock circuit was initialized. The output of the clock circuit may be used by operating system software to determine the number of clock pulses that elapsed during the execution of a program sequence. This operation may be accomplished by storing the contents of the clock at the beginning and end of the sequence, and comparing the two values. The following paragraphs describe the operation and inputs for each of the three portions of the counter.

CLOCK COUNTER REGISTER. The clock counter register consists of 32, type DF flip-flops. One input to the register receives 32 bits from word 2 of the

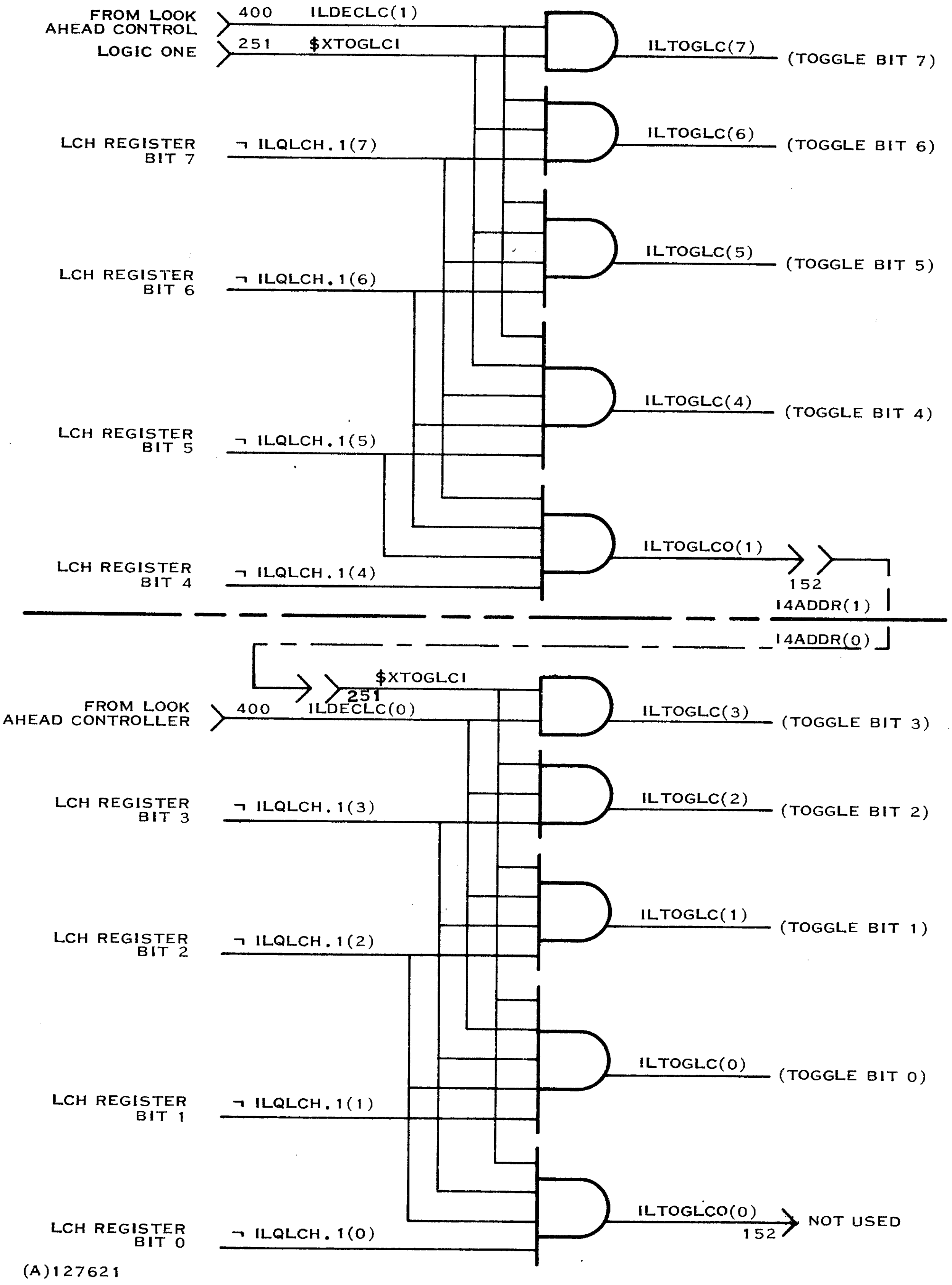


Figure B-20. Look Ahead Decrement Circuit

KCM interface file. This input is used during maintenance loads and is enabled by IMCMTCK:1, 2 from the details sequencer. These two gating signals are produced by either IMLDPSDW:2 for a load status, or during count 2 of a load details operation. The other input to the register is a feed-back loop from the CNTH register. The bits of this input are individually gated by control signals (ILTOGCK(8-31) and IXTOGCK(0-7)) from the clock increment circuit. If the increment circuit gates a bit into the register, that bit toggles. The increment circuit determines which bits to toggle so that an increment-by-one operation is performed each clock time. Clock pulses from the CP clock enable each input transfer to the CNT register. The output of CNT transfers to the CNTH register with each half phase clock pulse, and can be stored into memory through the details select during maintenance stores.

CNTH HOLDING REGISTER. The CNTH register receives the contents of the CNT register one-half clock pulse after the count has been entered into the CNT register. Half phase clock pulses (IPHICK:1 and 4) from the clock generation circuit transfer the contents of CNT to CNTH. The output of CNTH is available to the RO register on the I4PIPEMB for monitoring by the operating system, to the clock incrementer for determining increment gating signals, and to the CNT register to supply the toggle input data to increment the count in CNT.

CLOCK INCREMENTER. The clock incrementer circuit receives the contents of the CNTH register, examines that value, and generates gating signals to the feed back input of the CNT register to change the value in CNT to be one greater than the current value. The incrementer circuit requires no gating or control signals as long as IPU operations are enabled (IMQFREEZE:4 is low). Therefore, as soon as a new value is transferred into the CNTH, the incrementer circuit begins to produce the gating signals to increment the count in the CNT register. However, the CP clock pulse enables the actual incrementing operation in the CNT register. The incrementer examines all of the input bits in parallel. If all of the bits that are less significant than the bit under examination are ones, the circuit toggles that bit. The circuit that performs this function is illustrated in figure B-21 for bit 0, the most complex, and bit 31, the

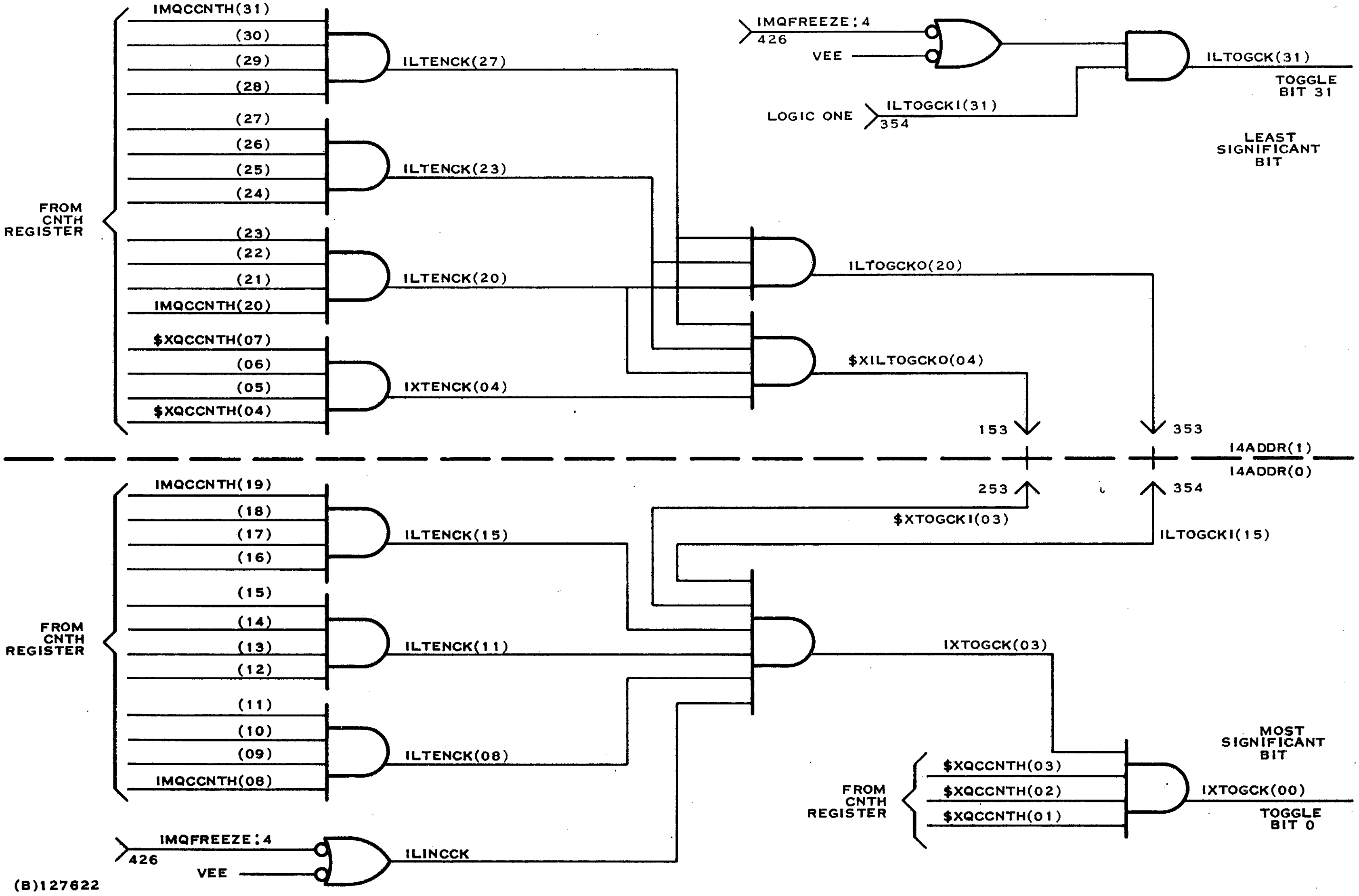


Figure B-21. Clock Incrementer

least complex. All other bits have similar circuits for examining the bit preceding them. All bits 1 through 31 must be ones before bit 0 toggles; bit 31 toggles every time the input value changes due to the fixed one input on ILTOGCKI(31). The incrementer circuit is divided into two segments on each I4ADDR circuit board. Because of this segmentation, the circuit requires interconnecting lines between the two circuit boards to maintain the continuity of the bit inspection. Figure B-22 illustrates the orientation of these interconnections.

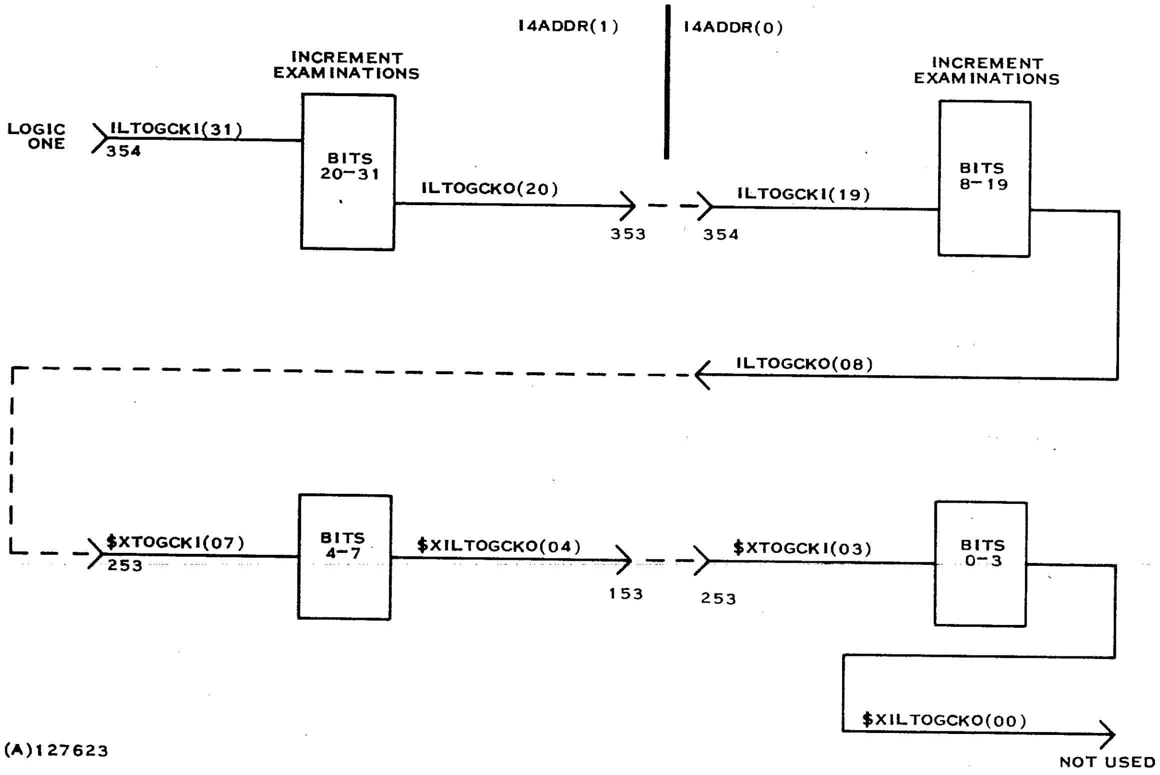
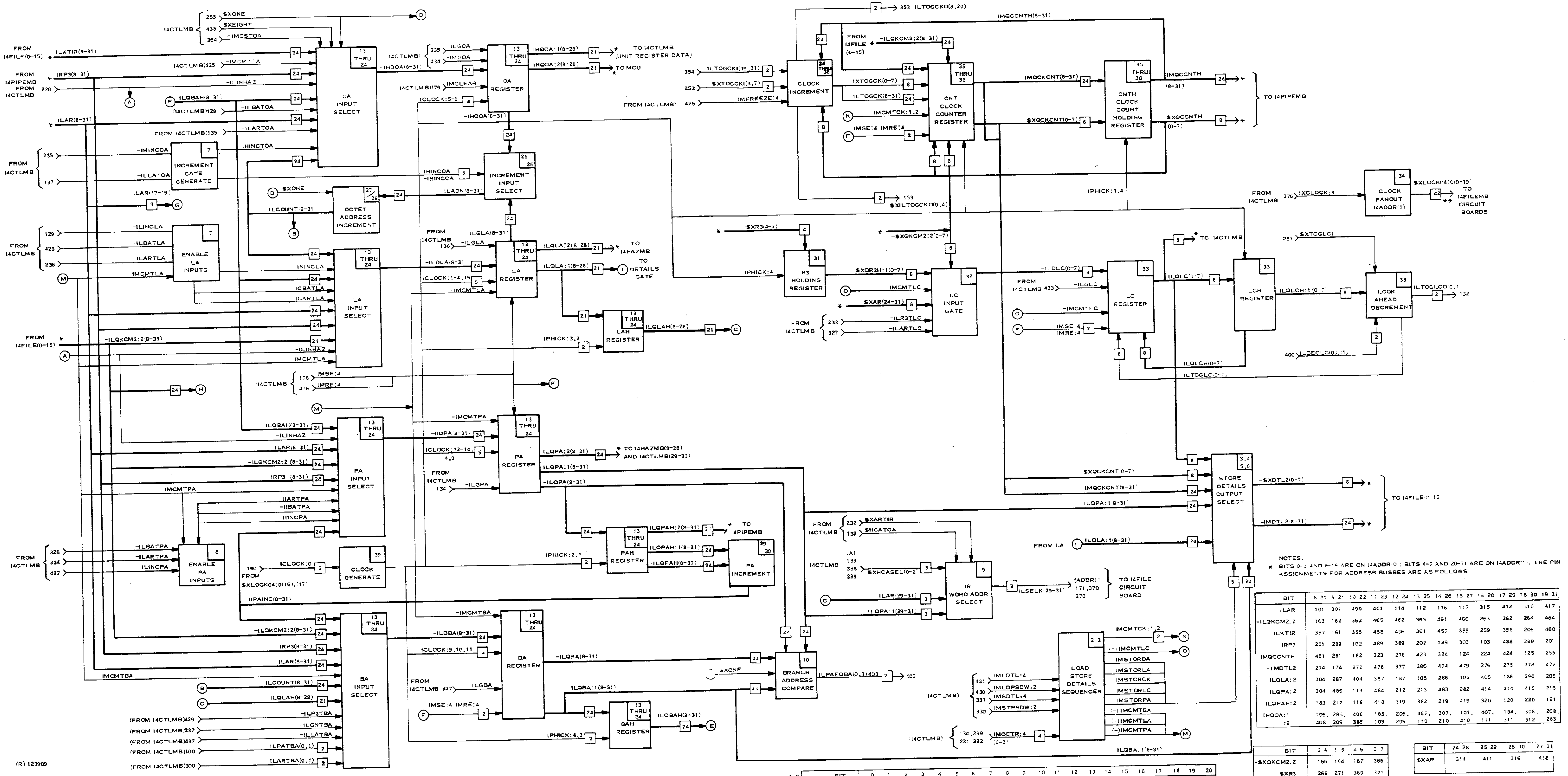


Figure B-22. Clock Incrementer Interconnections; I4ADDR(1) to I4ADDR(0)



NOTES.
* BITS 0-3 AND 8-19 ARE ON I4ADDR(0); BITS 4-7 AND 20-31 ARE ON I4ADDR(1). THE PIN ASSIGNMENTS FOR ADDRESS BUSSES ARE AS FOLLOWS:

BIT	8 20	9 21	10 22	11 23	12 24	13 25	14 26	15 27	16 28	17 29	18 30	19 31
ILAR	101	301	490	401	114	112	116	117	315	412	318	417
-ILQKCM2:2	163	162	362	465	462	365	461	466	263	262	264	464
ILKTIR	357	161	355	458	456	361	457	359	259	358	206	460
IRP3	201	289	102	489	389	202	189	303	103	488	368	201
IMQCCNTH	461	281	182	323	278	423	324	124	224	424	125	255
-IMDTL2	274	174	272	478	377	380	474	479	276	275	378	477
ILQLA:2	304	287	404	387	187	105	286	305	405	186	290	205
ILQPA:2	384	485	113	484	212	213	483	282	414	214	415	216
ILQPAH:2	183	217	118	418	319	382	219	419	320	120	220	121
IHQOA:1 :2	106, 408	285, 309	406, 385	185, 109	206, 209	487, 110	307, 210	107, 410	407, 111	184, 311	308, 312	208, 283

BIT	0 4	1 5	2 6	3 7
-$XQKCM2:2	166	164	167	366
-$XR3	266	271	369	371
-$XDTL	180	176	175	480
ILQLC	381	221	421	123
$XQCCNTH	326	126	279	226

BIT	24 28	25 29	26 30	27 31
$XAR	314	411	316	416

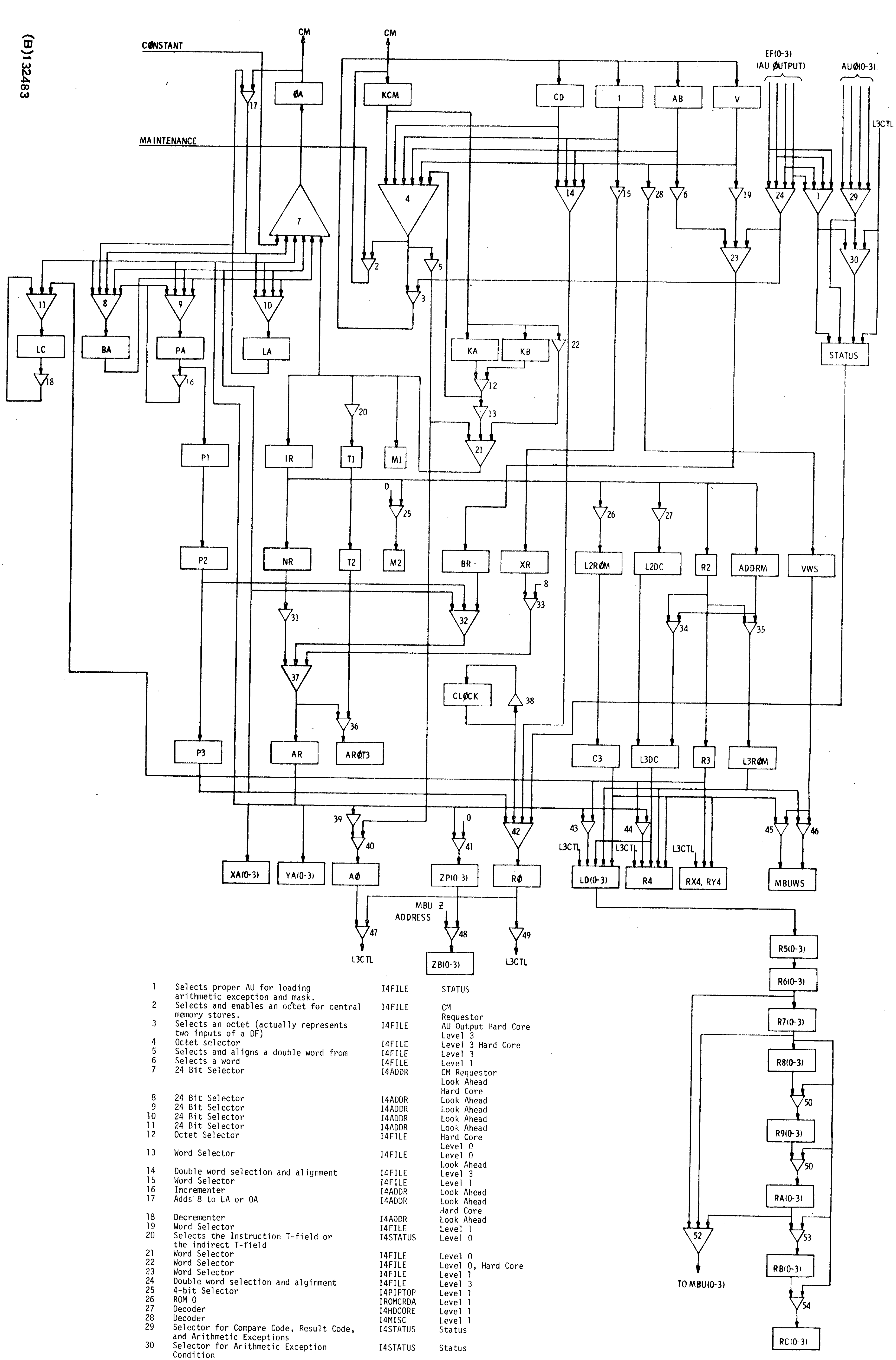
**

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
$XLOCK04:0	256	172	169	258	160	159	157	173	156	267	469	453	454	467	455	472	473	373	470	374	140

Figure B-23. I4ADDR Circuit Board Block Diagram

Table B-9. X4 IPU Registers

REGISTER	CONTENTS OR FUNCTION	LOCATION
ØA	Octet address for central memory requests.	I4ADDR
KCM	Octet data buffer for memory read requests.	I4FILE
A,B	Register file base registers	I4FILE
C,D	Register file arithmetic registers	I4FILE
I	Register file index registers	I4FILE
V	Vector parameter file	I4FILE
STATUS	Program status register	I4PIPE
CLOCK	Central Processor clock count	I4ADDR
BA	Branch address register	I4ADDR
PA	Address of the next sequential instruction to be executed.	I4ADDR
LA	Address of the next sequential octet of instructions to be executed.	I4ADDR
LC	Look ahead counter	I4ADDR
KA	Instruction buffer (1 octet)	I4FILE
KB	Instruction buffer (1 octet)	I4FILE
T1	T-field of instruction at level 1	I4STATUS
M1	M-field of instruction at level 1	I4STATUS
P1	Program counter at level 1	I4PIPE
IR	Instruction at level 1	I4PIPE
T2	T-field of the instruction at level 2	I4STATUS
M2	M-field of the instruction at level 2	I4PIPTOP
P2	Program Counter at level 2	I4PIPE
NR	Displacement of the operand address of the instruction at level 2	I4PIPE
BR	Selected base register (or vector parameter file register or stack pointer in special cases)	I4PIPE
XR	Selected index register	I4PIPE
L2RØM	Read only memory register at level 2	I4INFACE
L2DC	Instruction decode at level 2	I4INFACE
R2	4 most significant bits of the op-code and the R-field of the instruction at level 2	I4PIPE
ADDRM	4 least significant bits of the op-code of the instruction at level 2	I4STATUS
VWS	Vector word size	I4MISC
ARØT3	7 least significant bits of AR or T3	I4STATUS
P3	Program counter at level 3	I4PIPE
AR	Address register	I4PIPE
C3	Level 2 RØM transferred to level 3	I4INFACE
L3DC	Instruction decode at level 3	I4INFACE
L3RØM	RØM at level 3	I4INFACE
R3	4 most significant bits of the op-code and the R-field of the instruction at level 3.	I4PIPE
XA(0-3)	Address of the contents of the X register of each MBU	I4ZHAZ
YA(0-3)	Address of the contents of the Y register of each MBU	I4ZHAZ
AØ	A address of operand	I4PIPE
ZP(0-3)	Address of the contents of the Z register of each MBU	I4ZHAZ
RØ	Register Operand	I4PIPE
LD(0-3)	Last register address to enter the register stack	I4RHAZ
R4	Destination address and other bits at level 4	I4RHAZ



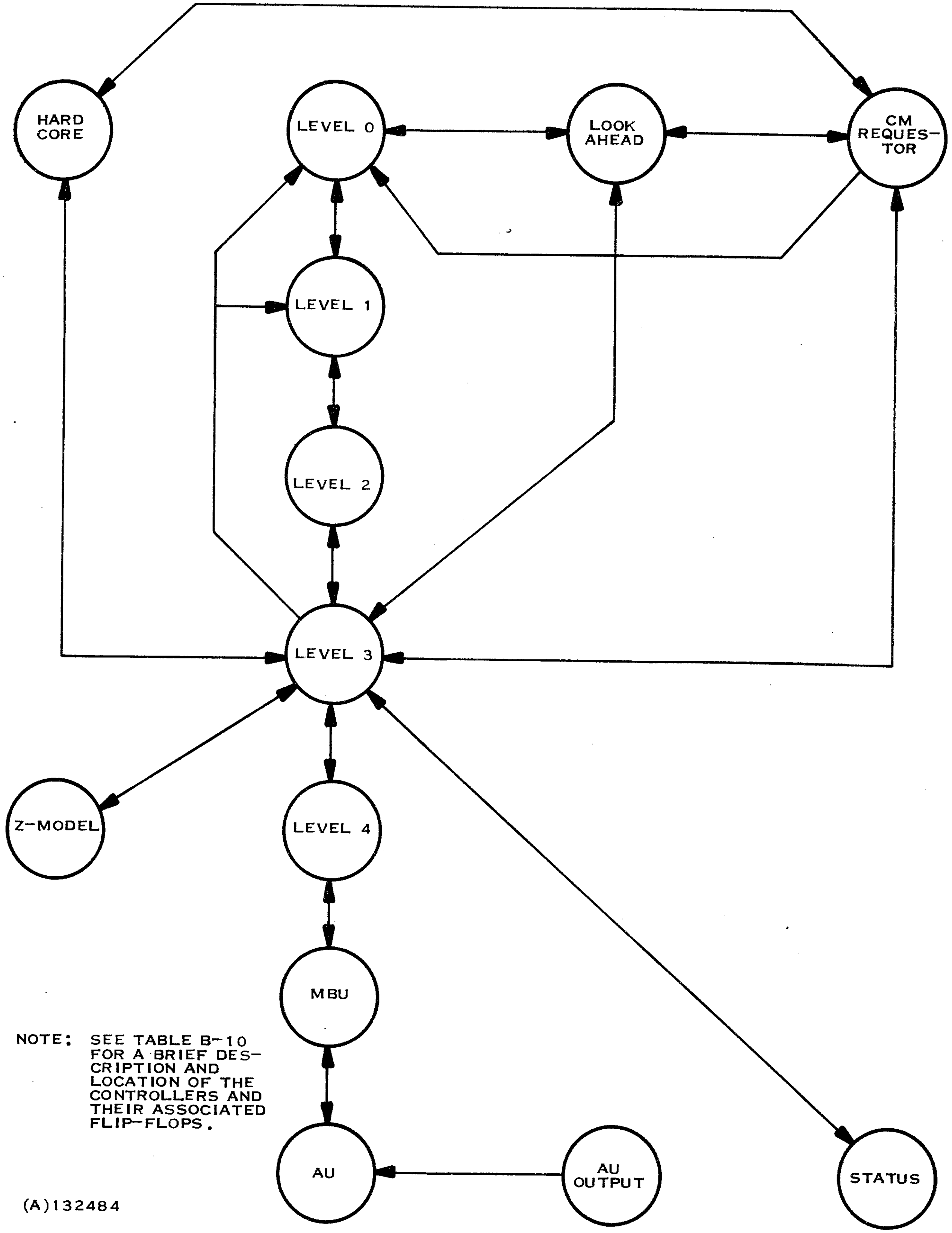
(B)132483

1	Selects proper AU for loading arithmetic exception and mask.	I4FILE	STATUS
2	Selects and enables an octet for central memory stores.	I4FILE	CM Requestor
3	Selects an octet (actually represents two inputs of a DF)	I4FILE	AU Output Hard Core Level 3
4	Octet selector	I4FILE	Level 3 Hard Core
5	Selects and aligns a double word from	I4FILE	Level 3
6	Selects a word	I4FILE	Level 1
7	24 Bit Selector	I4ADDR	CM Requestor Look Ahead Hard Core
8	24 Bit Selector	I4ADDR	Look Ahead
9	24 Bit Selector	I4ADDR	Look Ahead
10	24 Bit Selector	I4ADDR	Look Ahead
11	24 Bit Selector	I4ADDR	Look Ahead
12	Octet Selector	I4FILE	Hard Core Level 0
13	Word Selector	I4FILE	Level 0 Look Ahead
14	Double word selection and alignment	I4FILE	Level 3
15	Word Selector	I4FILE	Level 1
16	Incrementer	I4ADDR	Look Ahead
17	Adds 8 to LA or OA	I4ADDR	Look Ahead Hard Core
18	Decrementer	I4ADDR	Look Ahead
19	Word Selector	I4FILE	Level 1
20	Selects the Instruction T-field or the indirect T-field	I4STATUS	Level 0
21	Word Selector	I4FILE	Level 0
22	Word Selector	I4FILE	Level 0, Hard Core
23	Word Selector	I4FILE	Level 1
24	Double word selection and alginment	I4FILE	Level 3
25	4-bit Selector	I4PIPTOP	Level 1
26	ROM 0	IROMCRDA	Level 1
27	Decoder	I4HDCORE	Level 1
28	Decoder	I4MISC	Level 1
29	Selector for Compare Code, Result Code, and Arithmetic Exceptions	I4STATUS	Status
30	Selector for Arithmetic Exception Condition	I4STATUS	Status
31	Sign extender	I4PIPE	Level 2
32	Word Selector	I4PIPE	Level 2
33	Selector, shifter	I4PIPE	Level 2
34	Decoder	I4HDCORE	Level 2
35	ROM 1	IROMCRDB	Level 2
36	Selector (actually represents two inputs of a DF)	I4STATUS	Level 2
37	Adder	I4PIPE	Level 2
38	Incrementer	I4ADDR	Hard Core
39	Sign extender	I4PIPE	Level 3
40	Double-word selector	I4PIPE	Level 3
41	24-bit Selector	I4PIPE	Level 3
42	Double word selector	I4PIPE	Level 3
43	9-bit Selector	I4RHAZ	Level 3
44	9-bit Selector	I4RHAZ	Level 3
45	3-bit Selector	I4MISC	Level 3
46	3-bit Selector	I4MISC	Level 3
47	Adder (for BCLE and BCG)	I4PIPE	Level 3
48	24-bit Selector	I4MISC	Level 3
49	Incrementer-Decrementer (for Increment Decrement and test instructions)	I4PIPE	Level 3
50	24-bit Selector	I4ZHAZ	AU-Control
51	24-bit Selector	I4ZHAZ	AU-Control
52	Selector (Word Address Selector)	I4ZHAZ	Z-Model
53	24-bit Selector	I4ZHAZ	AU-Control
54	24-bit Selector	I4ZHAZ	AU-Control

Figure B-24. 4X IPU Block Diagram

Table B-9. X4 IPU Registers (Continued)

REGISTER	CONTENTS OR FUNCTION	LOCATION
RX4,RY4	Mapped op-code for the MBU	I4PIPE
MBUWS	MBU word size	I4MISC
ZB(0-3)	Address of the contents of the Z buffer of each MBU	I4ZHAZ
R5(0-3)	R4 copied into level 5	I4ZHAZ
R6(0-3)	R5 copied into level 6	I4ZHAZ
R7(0-3)	R6 copied into level 7	I4ZHAZ
R8(0-3)	R7 copied into level 8	I4ZHAZ
R9(0-3)	R7 or R8 copied into level 9	I4ZHAZ
RA(0-3)	R7 or R9 copied into level A	I4ZHAZ
RB(0-3)	R7 or RA copied into level B	I4ZHAZ
RC(0-3)	R7 or RB copied into level C	I4ZHAZ



NOTE: SEE TABLE B-10 FOR A BRIEF DESCRIPTION AND LOCATION OF THE CONTROLLERS AND THEIR ASSOCIATED FLIP-FLOPS.

(A)132484

Figure B-25. 4X IPU Controller State Diagram

Table B-10. Descriptions and Locations of 4X IPU Controllers and Flip-Flops

CONTROLLER	DESCRIPTION	PHYSICAL LOCATION
Hard Core	Controls all maintenance commands.	I4HDCORE
CM Requestor	Interfaces with central memory.	I4CMREQ, I4HDCORE
Look Ahead	Keeps appropriate data in the instruction buffers.	I4CMREQ
Level 0	Controls transfer of instructions and indirect calls to Level 1.	I4PIPTOP
Level 1	Controls transfer of Level 1 to Level 2.	I4PIPTOP
Level 2	Controls transfer of Level 2 to Level 3.	I4PIPTOP
Level 3	Controls transfer of Level 3 to Level 4	I4LVL3, I4VECLAS, I4MISC I4ROUTE 1, I4ROUTE 2, I4ROUTE
Level 4	Controls transfer of Level 4 to Level 5.	I4INFACE
MBU	Models activity of the MBU's.	I4INFACE
AU	Models activity of the AU/s.	I4INFACE
AU Output	Selects pipe whose results are to be accepted.	I4INFACE, I4MISC
Status	Controls modification of bits in the status register with the exception of IRQFORK, IROBSC, IRQMCC.	I4STATUS
Z-Model	Models the activity of stores to central memory.	I4INFACE

FLIP-FLOP	DESCRIPTION	CONTROLLER	LOCATION
ICQCUE0(0-3)	Destination address bit 0 of 4 read queue entires.	CM Requestor	I4CMREQ
ICQCUE1(0-3)	Destination address bit 1 of 4 read queue entries.	CM Requestor	I4CMREQ
ICQIP(0,1)	Read queue input pointer.	CM Requestor	I4CMREQ
ICQOP(0,1)	Read queue output pointer.	CM Requestor	I4CMREQ
ICQPRV(0-3)	Protect violate bit for each read queue entry.	CM Requestor	I4CMREQ
ICQACT(0-3)	Queue entry is active and will be used.	CM Requestor	I4CMREQ
ICQBSY(0-3)	Aueue entry is active and will be used.	CM Requestor	I4CMREQ
IMQRC(0-2)	Number of outstanding read requests.	CM Requestor	I4HDCORE
ICQPRM(0,1)	Protect Mode (type of request); Read = 11, Write = 01, Execute = 11.	CM Requestor	I4HDCORE
ICQAR	Requests central memory access.	CM Requestor	I4HDCORE
ICQRA	Central Memory request accepted.	CM Requestor	I4HDCORE
ICQRAO	ICQRA delayed 1 clock.	CM Requestor	I4HDCORE
IOQRDA	Latches read data available from central memory.	CM Requestor	I4HDCORE
ICQRDA	Synchronizes IOQRDA.	CM Requestor	I4HDCORE
ICQRDS	Read Data Sampled.	CM Requestor	I4HDCORE
IOQDAV	Latches data available from central memory.	CM Requestor	I4HDCORE
ICQDAV	Synchronizes IOQDAV.	CM Requestor	I4HDCORE
IOQPAR	Latches parity error from central memory.	CM Requestor	I4HDCORE
ICQPAR	Synchronizes IOQPAR.	CM Requestor	I4HDCORE
IMQPRV	Protection violation on a hard core command.	CM Requestor	I4HDCORE
IMQPAE	Parity error on a hard core command.	CM Requestor	I4HDCORE
ICQWRITE	Last request was a write request.	CM Requestor	I4HDCORE
ICQAREX	Arithmetic exception.	Hard Core	I4HDCORE
ICQIPPAE	Parity Error.	Hard Core	I4HDCORE
ICQIPIOP	Illegal op-code.	Hard Core	I4HDCORE
ICQIPPRV	Protect violate.	Hard Core	I4HDCORE

Table B-10. Descriptions and Locations of 4X IPU Controllers and Flip-Flops (Continued)

FLIP-FLOP	DESCRIPTION	CONTROLLER	LOCATION
IMQFREEZ	Disables normal instruction processing.	Hard Core	I4HDCORE
IMQOCTR(0-4)	Hard core octet counter.	Hard Core	I4HDCORE
IMQHCSTA(1-6)	Hard core state.	Hard Core	I4HDCORE
IMQLSD(0-3)	Hard core command.	Hard Core	I4HDCORE
IMQHCREQ	Hard core requirement (locks out further memory requests from other controllers).	Hard Core	I4HDCORE
IMQHCINP	Hard core in progress.	Hard Core	I4HDCORE
IMQEXCH	Indicates the load half of an exchange command must be done.	Hard Core	I4HDCORE
IMQLDPTR	Pointer octet is in KCM.	Hard Core	I4HDCORE
IMQSPSDW	Indicates a three word rather than octet store for store program status.	Hard Core	I4HDCORE
ICQKAFUL	KA instruction buffer is full.	Level 0	I4CMREQ
ICQKBFUL	KB instruction buffer is full.	Level 0	I4CMREQ
ICQKAHAZ	An instruction hazard exists in KA.	Level 0	I4CMREQ
ICQKBHAZ	An instruction hazard exists in KB.	Level 0	I4CMREQ
ICQKAPRV	The request for KA caused a protection violation.	Level 0	I4CMREQ
ICQKBPRV	The request for KB caused a protection violation.	Level 0	I4CMREQ
ICQKRTAG	Instructions are being fetched from KB.	Level 0	I4CMREQ
ILQSTATE	Level 0 state indicating that data from memory for an execute or indirect request is expected.	Level 0	I4PIPTOP
ILQTFAIL	The target branch has been skipped or not taken.	Look Ahead	I4CMREQ
ILQFLG12	Branch address has been requested.	Look Ahead	I4CMREQ
ILQFLG4	Target branch has entered the pipe.	Look Ahead	I4CMREQ
ILQFLGFL	Branch address + 8 is resident.	Look Ahead	I4CMREQ
ILQVAC(0,1)	Inactive levels when a PB or LLA is being executed.	Look Ahead	I4CMREQ
ILQLAORD	LA=PA+8.	Look Ahead	I4CMREQ
ILQPBVLL	PB or LLA, active in Look Ahead controller.	Look Ahead	I4CMREQ
ILQDUAL	Look Ahead controller in Dual mode.	Look Ahead	I4CMREQ
IIQL1ACT	Level 1 active.	Level 1	I4P1PTOP
IIQS(1)	Skip state.	Level 1	I4P1PTOP
IIQS(2)	Indirect or execute at level 2 state.	Level 1	I4P1PTOP
IIQS(3)	Indirect at level 3 state.	Level 1	I4P1PTOP
IIQS(4)	Execute at level 3 state.	Level 1	I4P1PTOP
IIQS(5)	Load file state.	Level 1	I4P1PTOP
IIQS(6)	Store file state.	Level 1	I4P1PTOP
IIQS(7)	Data available wait state.	Level 1	I4P1PTOP
IIQS(8)	Push, pull state.	Level 1	I4P1PTOP
IIQS(9)	Vector state.	Level 1	I4P1PTOP
IIQS(10)	Level 2 hazard state.	Level 1	I4P1PTOP
IIQPRV	Protect violate at level 1.	Level 1	I4P1PTOP
IIQFRHAZ	Instruction hazard at level 1	Level 1	I4P1PTOP
IIQTARGT	Target branch at level 1.	Level 1	I4P1PTOP
IPQL2ACT	Level 2 active.	Level 2	I4P1PTOP
IPQPRV	Protect violate at level 2.	Level 2	I4P1PTOP

Table B-10. Descriptions and Locations of 4X IPU Controllers and Flip-Flops (Continued)

FLIP-FLOP	DESCRIPTION	CONTROLLER	LOCATION
IPQFRHAZ	Instruction hazard at level 2.	Level 2	I4P1PTOP
IPQTARGT	Target branch at level 2.	Level 2	I4P1PTOP
IPQIND	Indirect instruction at level 2.	Level 2	I4P1PTOP
IRQL3ACT	Level 3 active.	Level 3	I4LVL3
IRQL3PPL	Push, pull state.	Level 3	I4LVL3
IRQL3CHK	Stack overflow check state.	Level 3	I4LVL3
IRQL3PWT	Stack pointer wait state.	Level 3	I4LVL3
IRQL3BWN	MCW, MCP state.	Level 3	I4LVL3
IRQL3IWT	Indirect wait state.	Level 3	I4LVL3
IRQL3IHZ	Instruction hazard state.	Level 3	I4LVL3
IRQL3VIN	Vector initiate state.	Level 3	I4LVL3
IRQL3FLW	File wait state.	Level 3	I4LVL3
IRQL3VP1	Vector plus 1 state.	Level 3	I4LVL3
IRQL3VBR	Vector burst state.	Level 3	I4LVL3
IRQL3NIW	New instruction wait state.	Level 3	I4LVL3
IRQL3ORW	Load and store file wait state.	Level 3	I4LVL3
IRQL3ORM	Load and store file multiple state.	Level 3	I4LVL3
IRQL3VGO	Vector go state.	Level 3	I4LVL3
IRQHOLD	General purpose flag.	Level 3	I4LVL3
IRQOPDN	Operand selection phase complete.	Level 3	I4LVL3
IRQBRDN	Branch phase complete.	Level 3	I4LVL3
IRQARINC	Increment AR for load or store file multiple.	Level 3	I4VECLAS
IRQXEC	Execute flag (don't branch skip or issue monitor call).	Level 3	I4LVL3
IRQRCTR(0-2)	Request counter.	Level 3	I4VECLAS
IRQCCTR(0-2)	Complete counter.	Level 3	I4PIPTOP
IRQJOIN	Join flag (process current vector or load file in join mode).	Level 3	I4VECLAS
IRQRIAIB	Forces central memory α operand.	Level 3	I4LVL3
IRQMEQO	M-field of instruction at level 3 was 0.	Level 3	I4LVL3
IRQPMOFF	Turns off protect and map enable.	Level 3	I4LVL3
IRQARHAZ	α hazard occured because of a vector.	Level 3	I4VECLAS
IRQFRHZ	Instruction hazard at level 3.	Level 3	I4LVL3
IRQPRV	Protect violate at level 3.	Level 3	I4LVL3
IRQTARGT	Target branch at level 3.	Level 3	I4LVL3
IRQIND	Indirect instruction at level 3.	Level 3	I4LVL3
IRQILOP	Illegal operation at level 3.	Level 3	I4LVL3
IRQHDW	Use double word for hazard detection.	Level 3	I4LVL3
IRQVIP(n)	n=0-3, vector in progress in pipe n.	Level 3	I4VECLAS
IRQVISTR(n)	n=0-3, vector initiate start in pipe n.	Level 3	I4VECLAS
IRQSELPI(n)	n=0-3, pipe n selected for vector at level 3.	Level 3	I4VECLAS
IRQVBAD(n)	n=0-3, vector bad guy in pipe n.	Level 3	I4VECLAS
IRQ4GETI(n)	n=0-3, do not execute vector waiting in MBU(n).	Level 3	I4VECLAS

Table B-10. Descriptions and Locations of 4X IPU Controllers and Flip-Flops (Continued)

FLIP-FLOP	DESCRIPTION	CONTROLLER	LOCATION
IRQPIRT(n)	n=0-3, previous instruction routed to pipe n.	Level 3	I4ROUTE3
IRQZMAL1(n)	n=0-3, all zone modification bits set in MBU(n).	Level 3	I4ROUTE3
IRQYNEXT(n)	n=0-3, Y buffer of MBU(n) to be loaded next.	Level 3	I4ROUTE3
IRQWNDER(n)	n=0-3, Early window for divide in pipe n is present.	Level 3	I4CMREQ
TRQWNDLT(n)	n=0-3, Late window for divide in pipe n is present.	Level 3	I4CMREQ
IRQZmAGE(n)	n=1-4, m≠n, determines pipe least recently used for stores.	Level 3	I4MISC
IRQPOIND	Stack pointer is at the AU output.	Level 3	I4LVL3
IRQTRMIN	Stack has overflowed.	Level 3	I4LVL3
IRQHCALI	Monitor call allowed.	Level 3	I4LVL3
IRQCRSLT	Result for BCLE, BCG operation (if set BCLE branch would be taken).	Level 3	I4LVL3
IRQSKIND	Skip indicator.	Level 3	I4LVL3
IBQL4RJN	Join mode read request at level 4.	Level 3	I4MISC
IBQRJOIN(n)	n=0-3, pipe n has an outstanding join mode read request.	Level 3	I4MISC
IBQZJOIN(n)	n=0-3, pipe n contains a store made in join mode.	Level 3	I4MISC
IBQIRT(n)	n=0-3, NEW INSTRUCTION AT LVL4 ROUTED TO PIPE n	LEVEL 4	I4INFACE(0-3)
IBQMODE(n)	n=0-3, PIPE n IS WAITING FOR AN UPDATE.	LEVEL 4	I4INFACE(0-3)
IBQLDXA(n)	n=0-3, INSTRUCTION AT LVL4 FOR PIPE n REQUIRES AN OPERAND FROM THE X=BUFFER	LEVEL 4	I4INFACE(0-3)
IBQLDXBA(n)	n=0-3, INITIATES X-BUFFER FETCH IN PIPE n	LEVEL 4	I4INFACE(0-3)
IBQLDYA(n)	n=0-3, INSTRUCTION AT LVL4 FOR PIPE n REQUIRES AN OPERAND FROM THE Y-BUFFER	LEVEL 4	I4INFACE(0-3)
IBQLDYBA(n)	n=0-3, INITIATES Y-BUFFER FETCH IN PIPE n	LEVEL 4	I4INFACE(0-3)
IBQXAACT(n)	n=0-3, VALID ADDRESS IN XA(n)	LEVEL 4	I4INFACE(0-3)
IBQYAACT(n)	n=0-3, VALID ADDRESS IN YA(n)	LEVEL 4	I4INFACE(0-3)
IBQXAFUL(n)	n=0-3, VALID DATA IN X-BUFFER FOR PIPE n	LEVEL 4	I4INFACE(0-3)
IBQYAFUL(n)	n=0-3, VALID DATA IN Y-BUFFER FOR PIPE n	LEVEL 4	I4INFACE(0-3)
IBQCUEQX(n)	n=0-3, BCCUEEQX(n) FROM MBU(n) DELAYED BY 1 CLOCK	LEVEL 4	I4INFACE(0-3)
IBQCUEQY(n)	n=0-3, BCCUEEQY(n) FROM MBU(n) DELAYED BY 1 CLOCK	LEVEL 4	I4INFACE(0-3)
IBQSMGP4(n)	n=0-3, INSTRUCTION AT LVL4 FOR PIPE n IS IN THE SAME GROUP AS THE LAST INSTRUCTION TO ENTER PIPE n.	LEVEL 4	I4INFACE(0-3)
IBQOCK4(n)	n=0-3, INSTRUCTION AT LVL4 FOR PIPE n IS A ONE CLOCK.	LEVEL 4	I4INFACE(0-3)
IBQSCKT4(n)	n=0-3, INSTRUCTION AT LVL4 FOR PIPE n WILL SHORT CIRCUIT.	LEVEL 4	I4INFACE(0-3)
IBQFCSGT(n)	n=0-3, INSTRUCTION AT LVL4 FOR PIPE n HAS SAME GROUP TIME ON ITS FIRST μ-SEQUENCE.	LEVEL 4	I4INFACE(0-3)
IBQREGDP(n)	n=0-3, INDICATES DATA IN RO FOR PIPE n.	LEVEL 4	I4INFACE(0-3)
IBQIMM(n)	n=0-3, INSTRUCTION AT LVL4 FOR PIPE n DOES NOT REQUIRE A MEMORY OPERAND.	LEVEL 4	I4INFACE(0-3)
IBQXUP(n)	n=0-3, INHIBITS THE SETTING OF DPMBI IN PIPE n PENDING AN X-UPDATE.	LEVEL 4	I4INFACE(0-3)
IBQYUP(n)	n=0-3, INHIBITS THE SETTING OF DPMBI IN PIPE n PENDING A Y-UPDATE.	LEVEL 4	I4INFACE(0-3)
IBQZTXU(n)	n=0-3, INITIATES A Z→X UPDATE IN PIPE n.	LEVEL 4	I4INFACE(0-3)
IBQZTYU(n)	n=0-3, INITIATES A Z→Y UPDATE IN PIPE n.	LEVEL 4	I4INFACE(0-3)
IBQZEX(n)	n=0-3, INDICATES PIPE n HAS SAME OCTET IN Z+X BUFFERS.	LEVEL 4	I4INFACE(0-3)

Table B-10. Descriptions and Locations of 4X IPU Controllers and Flip-Flops (Continued)

FLIP-FLOP	DESCRIPTION	CONTROLLER	LOCATION
IBQZEY(n)	n=0-3, INDICATES PIPE n HAS SAME OCTET IN Z+Y BUFFERS.	LEVEL 4	I4INFACE(0-3)
IBQGP1L(n)	n=0-3, INDICATES GROUP NUMBER OF LAST INSTRUCTION TO ENTER PIPE n.	LEVEL 4	I4INFACE(0-3)
IBQGP2L(n)			
IBQGP3L(n)			
IBQGP4L(n)			
IVQMBIAC(n)	n=0-3, INDICATES AN INSTRUCTION AT LVL5 IN PIPE n.	LEVEL 5	I4INFACE(0-3)
IVQDPMBI(n)	n=0-3, INDICATES INSTRUCTION AT LVL5 IN PIPE n HAS ITS DATA.	LEVEL 5	I4INFACE(0-3)
IVQSMGP5(n)	n=0-3, INDICATES INSTRUCTION AT LVL5 IN PIPE n IS IN THE SAME GROUP AS THE LAST INSTRUCTION TO ENTER THIS PIPE.	LEVEL 5	I4INFACE(0-3)
IVQOCK5(n)	n=0-3, INDICATES INSTRUCTION AT LVL5 IN PIPE n IS A ONE CLOCK.	LEVEL 5	I4INFACE(0-3)
IVQSLNXT(n)	n=0-3, INDICATES FOR PIPE n WHEN AN INSTRUCTION AT LVL5 CAN MOVE TO LVL6 BASED ON WHAT INSTRUCTIONS ARE IN THE AU.	LEVEL 5	I4INFACE(0-3)
IVQSCKT5(n)	n=0-3, INDICATES INSTRUCTION AT LVL5 OF PIPE n WILL SHORT CIRCUIT.	LEVEL 5	I4INFACE(0-3)
IVQDPMBO(n)	n=0-3, INDICATES AN INSTRUCTION AT LVL6 OF PIPE n.	LEVEL 6	I4INFACE(0-3)
IVQSCKT6(n)	n=0-3, INDICATES INSTRUCTION AT LVL6 OF PIPE n WILL SHORT CIRCUIT.	LEVEL 6	I4INFACE(0-3)
IVQENAB(n)	n=0-3, AU ROM BITS LATCHED IN THE IPU USED TO CONTROL MOVEMENT IN THE REGISTER STACK FOR PIPE n.	AU MODEL	I4INFACE(0-3)
IVQROM8(n)			
IVQROM9(n)			
IVQROMA(n)			
IVQROMB(n)			
IVQROMC(n)			
IVQPACAO(n)	n=0-3, PATH AHEAD CLEAR INTO LVL12 FOR PIPE n.	AU MODEL	I4INFACE(0-3)
IVQL7ACT(n)	n=0-3, INSTRUCTION AT LVL7 IN PIPE n.	AU MODEL	I4INFACE(0-3)
IVQAUIAC(n)	n=0-3, INSTRUCTION AT LVL8, 9, A, OR B IN PIPE n.	AU MODEL	I4INFACE(0-3)
IBQREQFW(n)	n=0-3, REQUEST FORCED WRITE STATE OF FORCED WRITE CONTROLLER FOR PIPE n.	Z-MODEL	I4INFACE(0-3)
IBQFWWT(n)	n=0-3, FORCED WRITE WAIT STATE OF FORCED WRITE CONTROLLER FOR PIPE n.	Z-MODEL	I4INFACE(0-3)
IBQZPFUL(n)	n=0-3, INDICATES VALID Z-BUFFER ADDRESS IN ZP FOR PIPE n.	Z-MODEL	I4INFACE(0-3)
IBQZBFUL(n)	n=0-3, INDICATES VALID ZB-BUFFER ADDRESS IN ZB FOR PIPE n.	Z-MODEL	I4INFACE(0-3)
IBQDAVWT(n)	n=0-3, DAV WAIT STATE IN ZB CONTROLLER FOR PIPE n.	Z-MODEL	I4INFACE(0-3)
IVQPPMF(n)	n=0-3, INDICATES 2ND PASS OF PUSH, PULL, OR MODIFY INSTRUCTION IN PIPE n.	Z-MODEL	I4INFACE(0-3)
IRQCCRT(n)	n=0-3, COMPARE CODE ROUTE FLIP-FLOPS INDICATING PIPE n WAS THE LAST TO RECEIVE AN INSTRUCTION WHICH CAN MODIFY THE CC REGISTER.	PROGRAM STATUS	I4STATUS
IRQRCRT(n)	n=0-3, RESULT CODE ROUTE FLIP-FLOPS INDICATING PIPE n WAS THE LAST TO RECEIVE AN INSTRUCTION WHICH CAN MODIFY THE RC REGISTER.	PROGRAM STATUS	I4STATUS

Table B-11. Program Status Word Registers

MOTHERBOARD SIGNATURE	CARD SIGNATURE IRQPSW( )	CARD LOC. I4PIPE( )	DESCRIPTION	
IRQAE(0)	0	0	DIVIDE CHECK	ARITHMETIC EXCEPTION REGISTER
IRQAE(1)	1	0	FIXED POINT OVERFLOW	
IRQAE(2)	2	0	FLOATING POINT OVERFLOW	
IRQAE(3)	3	0	FLOATING POINT UNDERFLOW	
IRQAEM(0-3)	4-7	1	ARITHMETIC EXCEPTION MASK REGISTER	
IRQPSW(8-11)	8-11	2	UNUSED	
IRQPIDIS(0-3)	12-15	3	PIPE DISABLE REGISTER	
IRQPROEN	16	4	PROTECT ENABLE BIT	CMU REGISTER
IRQMAPEN	17	4	MAP ENABLE BIT	
IRQPSW(18-19)	18-19	4	UNUSED	
IRQPSW(20)	20	5	UNUSED	BSR REGISTER
IRQFORK	21	5	FORK INDICATOR	
IRQMCC	22	5	MCC BIT	
IRQBSC	23	5	BSC BIT	
IRQPSW(24)	24	6	UNUSED	CC REGISTER
IRQCL	25	6	COMPARE LESS THAN	
IRQCG	26	6	COMPARE GREATER THAN	
IRQCE	27	6	COMPARE EQUAL	
IRQPSW(28)	28	7	UNUSED	RC REGISTER
IRQRL	29	7	RESULT LESS THAN	
IRQRG	30	7	RESULT GREATER THAN	
IRQRE	31	7	RESULT EQUAL	

Table B-12. X4 IPU ROM 1 Listing

MOTHERBOARD	CARD SIGNATURE		
SIGNATURE:	IPQROM(BIT)	CARD LOC:	
IPQRMXXX	IRQC3(BIT)	I4INFACE(n)	
IRQC3XXX	BIT=	n=	DESCRIPTION
AEH	9	1	ARITHMETIC EXCEPTION HAZARD, INSTRUCTION MAY MODIFY AE.
AU4	16	2	MERGED OP-CODE, BIT 4-7, FOR AU ROM ADDRESS.
AU5	17	2	
AU6	18	2	
AU7	19	2	
BR	3	0	INSTRUCTION CAN HAVE BASE ADDRESSING.
CAR	14	1	CARRY INTO BIT 32 OF AR REGISTER. FOR HALFWORD INSTRUCTIONS WHICH NORMALLY POINT TO THE RIGHT HALFWORD.
CHZ	15	1	COMPARE HAZARD. INSTRUCTION MAY MODIFY CC.
DHW	4	0	REGISTER DESTINATION HALF WORD.
DDW	5	0	REGISTER DESTINATION DOUBLE WORD.
EXM	11	1	EXTENDS SIGN FROM BIT 16 OF NR TO BIT 8 INTO OPA. EXTENDS SIGN FROM BIT 8 OF AR TO BIT 0 INTO ROO.
EXN	10	1	EXTENDS SIGN FROM BIT 20 OF NR TO BIT 8 INTO OPA.
GP1	24	3	INSTRUCTION GROUP NUMBER BITS.
GP2	25	3	
GP3	26	3	
GP3	27	3	
HHW	22	2	INDICATES WHEN HAZARD COMPARISONS SHOULD BE ON HALFWORD BASIS.
IDW	8	1	INDEXER DOUBLEWORD SIZE
IHW	6	0	INDEXER HALFWORD SIZE
IOP	29	3	ILLEGAL OP-CODE
ISW	7	0	INDEXER SINGLEWORD
M	12	1	SELECT M-FIELD FROM NR INTO OPA
MDV	28	3	MULTIPLY INSTR. WITH WORD SIZE OPTION
RAO	0	0	REGISTER ADDRESS BIT 0

Table B-12. X4 IPU ROM 1 Listing (Continued)

MOTHERBOARD
SIGNATURE:
IPQRMXXX
IRQC3XXX
	CARD SIGNATURE
	IPQROM(BIT)
	IRQC3(BIT)
	BIT=
		CARD LOC:
		I4INFACE(n)
		n=
			DESCRIPTION

RA1	1	0	REGISTER ADDRESS BIT 1
RA6	2	0	REGISTER ADDRESS BIT6, HALFWORD
RDT	20	2	REGISTER DESTINATION
RHZ	13	1	RESULT CODE HAZARD. INSTRUCTION CAN MODIFY RC.
RSE	21	2	REGISTER SPECIFICATION ERROR POSSIBLE
SDW	23	2	REGISTER SOURCE DOUBLEWORD SIZE

Table B-13. X4 IPU ROM 2 Listing

MOTHERBOARD SIGNATURE: IRQRMXXX	CARD SIGNATURE IRQROM(BIT) BIT=	CARD LOC: I4INFACE(n) n=	DESCRIPTION
ADW	1	0	ALPHA DOUBLEWORD SIZE
AHW	0	0	ALPHA HALFWORD SIZE
BLU	10	1	BLUE INSTRUCTION
BRH	8	1	BRANCH INSTRUCTION
GRN	13	1	GREEN INSTRUCTION
GRY	7	0	GRAY INSTRUCTION, i.e., ALL SUB-COLORS
IDN	11	1	INCREMENT OR DECREMENT AND BRANCH ON NON-ZERO INSTRUCTION
IDZ	12	1	INCREMENT OR DECREMENT AND BRANCH ON ZERO INSTRUCTION
NSN	25	3	STORE, BUT NOT STORE NEGATIVE INSTRUCTION
NSP	18	2	SUB-ORANGE, BUT NOT SPS INSTR.
OCK	21	2	ONE CLOCK INSTRUCTION
PNK	9	1	PINK INSTRUCTION
RGS	22	2	INSTR. WITH A REGISTER SOURCE
SBL	2	0	SUB-BLUE INSTRUCTION
SGN	3	0	SUB-GREEN INSTRUCTION
SGT	20	2	INSTRUCTION WHERE THE FIRST CLOCK OF THE AU ROM SEQUENCE IS THE SAME GROUP TIME
SHW	16	2	REGISTER SOURCE HALFWORD SIZE
SOR	6	0	SUB-ORANGE INSTRUCTION
SPK	4	0	SUB-PINK INSTRUCTION
STR	24	3	STORE INSTRUCTION
SYL	5	0	SUB-YELLOW INSTRUCTION

Table B-14. X4 IPU Interface Signals

IPU-MCU INTERFACE

IPU	MCU	DESCRIPTION
IOCMn(0-31)		n=0-7, Central memory data lines, two-way bus
IOPA		MCU to IPU, Parity error
IOPR		MCU to IPU, Protect violation
IOMIN		MCU to IPU, Memory inoperative
IOWG		MCU to IPU, Write gate (put data on the bus)
IORA		MCU to IPU, Request accepted
IORDA		MCU to IPU, Read data available
IODAV		MCU to IPU, Store data available
ICPIN		IPU to MCU, Processor inoperative
ICQAR:2		IPU to MCU, Access requested
ICRDS		IPU to MCU, Read data sampled
ICWGI		IPU to MCU, Write gate inverted (data is on the bus)
ICPRR		IPU to MCU, Parity error
ICZE(0-7)		IPU to MCU, Zone enables (indicates words to be written into)
IHQOA:2 (8-31)		IPU to MCU, Central memory address
ICQPRM:2 (0-1)		IPU to MCU, Protect mode
ICPE:9		IPU to MCU, Protect enable
ICME:9		IPU to MCU, Map enable

Table B-14. X4 IPU Interface Signals (Continued)

CP CLOCK - PPU INTERFACE

CLOCK-MHC	PPU	DESCRIPTION
CTB		PPU to Clock, Transfer bit
CCMDBT (0-8)		PPU to Clock, Common command register
CCMDBT (10-15)		PPU to Clock, Common command register
¬CEXTR		PPU to Clock, Run in external mode
¬CSLAVE		PPU to Clock, Run in slave mode
¬CACKCMD		Clock to PPU, Reset transfer bit
¬BHCLKEN:1		PPU to Clock, Clock enable for MBU (0)
2		PPU to Clock, Clock enable for MBU (1)
3		PPU to Clock, Clock enable for MBU (2)
4		PPU to Clock, Clock enable for MBU (3)
¬AHCLKEN:1		PPU to Clock, Clock enable for AU (0)
2		PPU to Clock, Clock enable for AU (1)
3		PPU to Clock, Clock enable for AU (2)
4		PPU to Clock, Clock enable for AU (3)
¬I4CLKEN		PPU to Clock, Clock enable for IPU

Table B-14. X4 IPU Interface Signals (Continued)

PPU - MASTER HARDCORE INTERFACE

MHC	UNIT-PPU	DESCRIPTION
¬I4CRSR		PPU to MHC, System reset } Reset CPU
I4STBY		PPU to MHC, System standby
I4CRTB		PPU to MHC, "Have CCR for some one in the system"
I4CRCCR(0-15)		PPU to MHC, Common command register buss
¬I4CRAC		PPU to MCH, "Allow automatic call" flag
¬I4CRAS		PPU to MHC, "Allow automatic switch" flag
¬I4CRTR		PPU to MHC, "Terminate CCR servicing abnormally"
I4CRTBRL		PPU to MHC, "I recognize you've sent I4TBRS."
¬I4CRMC		PPU to MHC, "I'm not finished servicing an automatic call"
¬I4CRSC		PPU to MHC, "I'm not finished servicing an automatic switch"
¬I4CRSB		PPU to MHC, "When doing an exchange, do a load status on the load cycle"
¬I4CRCSR		PPU to MHC, "The map and project registers have been loaded"
I4IGNPE		PPU to MHC, "Please ignor all parity errors"
U4EPRINT		PPU to MHC, If program errors occurs, interrupt PPU with I4QINTRP
I4MCPINT		PPU to MHC, If Monitor call and proceed, interrupt PPU with I4QINTRP
I4MCWINT		PPU to MHC, If Monitor call and wait, interrupt PPU with I4QINTRP
I4RERINT		PPU to MHC, If reason error occurs, interrupt PPU with I4QINTRP
I4SYSINT		PPU to MHC, If system error occurs, interrupt PPU with I4QINTRP
¬I4TBRS		MHC to PPU, "I've got the CCR command; reset I4CRTB so I can proceed"
¬I4QGSE		MHC to PPU, "I've encountered a system error that needs your attention"
¬I4QAT		MHC to PPU, "I've done a message call or program switch"
¬I4QMC		MHC to PPU, "Here's a message for you to look at"
¬I4QSC		MHC to PPU, "I've switched out one program, and switched in another"

Table B-14. X4 IPU Interface Signals (Continued)

PPU - MASTER HARDCORE INTERFACE

MHC	UNIT-PPU	DESCRIPTION
¬I4QGAT		MHC to PPU, The gate necessary to load at, MC and SC at the PPU end
¬I4QSS		MHC to PPU, "I've done a store status CCR command"
¬I4QRZ(0-2)		MHC to PPU, The reason error code buffer
¬I4QGRZ		MHC to PPU, The gate necessary to load RZ(0-2) at the PPU end
¬I4QGCC		MHC to PPU, "I've completed the CCR command" (also gates I4QAB to PPU)
¬I4QAB		MHC to PPU, "I've terminated due to an abnormal condition"
¬I4QME		MHC to PPU, "I've encountered a protect violate or parity error during servicing
¬I4QPE		MHC to PPU, Parity error flag, caused by program C.M. request (IPU or MBU's)
¬I4QIL		MHC to PPU, Illegal opcode flag (IPU or MBU's)
¬I4QAE		MHC to PPU, Arithmetic exception flag (IPU)
¬I4QPV		Protect violate flag, caused by program C. M. request (IPU or MBU's)
¬I4QRB		MHC to PPU; CPU run bit
¬I4QGCB		MHC to PPU, The gate necessary to load ME, DE, IL, AE, PV and RB at PPU end
¬I4QAVBG		MHC to PPU, "There's a vector bad guy running in at least one MBU-AU pipe
¬I4QINTR1:2		
¬BHUR *0-7)		MHC to PPU, Unit register data from MBU (0)
BHUR(10-17)		MHC to PPU, Unit register data from MBU (1)
BHUR (20-27)		MHC to PPU, Unit register data from MBU (2)
BHUR (30-37)		MHC to PPU, Unit register data from MBU (3)
¬AHUR (0-7)		MHC to PPU, Unit register data from AU (0)
AHUR (10-17)		MHC to PPU, Unit register data from AU (1)
AHUR (20-27)		MHC to PPU, Unit register data from AU (2)
AHUR (30-37)		MHC to PPU, Unit register data from AU (3)

Table B-14. X4 IPU Interface Signals (Continued)

PPU - MASTER HARDCORE INTERFACE

MHC	UNIT-PPU	DESCRIPTION
¬IMDTUR (0-7)		MHC to PPU, Unit register data from IPU
¬I4URDATA (0-7)		MHC to PPU, Unit register data from MHC
I4CSW (0-1)		MHC to PPU, Have the map and protect registers been loaded?

Table B-14. X4 IPU Interface Signals (Continued)

MBU, AU - MASTER HARDCORE INTERFACE

MHC	UNIT - MUB(0-3)	DESCRIPTION	
I4RUNBIT:1	BHRUNBIT	MHC to MBU(0)	Indicated value of CPU run bit (I4QRB)
I4RUNBIT:2	BHRUNBIT	MHC to MBU(1)	
I4RUNBIT:3	BHRUNBIT	MHC to MBU(2)	
I4RUNBIT:4	BHRUNBIT	MHC to MBU(3)	
I4CLEAR:1	BICLEAR	MHC to MBU(0)	"Master clear of CPU"
I4CLEAR:2	BICLEAR	MHC to MBU(1)	
I4CLEAR:3	BICLEAR	MHC to MBU(2)	
I4CLEAR:4	BICLEAR	MHC to MBU(3)	
¬I4HCCCR:1(12-15)	¬BHCCR(12-15)	MHC to MBU(0)	Least significant hex of buffered common command register (I4QCCR(12-15))
I4HCCCR:2(12-15)	BHCCR(12-15)	MHC to MBU(1)	
I4HCCCR:3(12-15)	BHCCR(12-15)	MHC to MBU(2)	
I4HCCCR:4(12-15)	BHCCR(12-15)	MHC to MBU(3)	
I4HCINIT:1	BHHCINIT	MHC to MBU(0)	"Initiate CCR servicing"
I4HCINIT:2	BHHCINIT	MHC to MBU(1)	
I4HCINIT:3	BHHCINIT	MHC to MBU(2)	
I4HCINIT:4	BHHCINIT	MHC to MBU(3)	
I4URSEL:1(0-3)	BHURSEL(0-3)	MHC to MBU(0)	Least significant hex of PPU common command register (I4CRCCR(12-15)) used for selecting data to BHUR0-7, 10-17, 20-27, 30-37
I4URSEL:2(0-3)	BHURSEL(0-3)	MHC to MBU(1)	
I4URSEL:3(0-3)	BHURSEL(0-3)	MHC to MBU(2)	
I4URSEL:4(0-3)	BHURSEL(0-3)	MHC to MBU(3)	
¬I4ABORT:1	¬BHABORT	MHC to MBU(0)	Indicates "ABORT CCCR servicing <u>now</u>"
I4ABORT:2	BHABORT	MHC to MBU(1)	
I4ABORT:3	BHABORT	MHC to MBU(2)	
I4ABORT:4	BHABORT	MHC to MBU(3)	
I4HCRINP:1	BHHCRINP	MHC to MBU(0)	Indicates "Proceed with CCR servicing"
I4HCRINP:2	BHHCRINP	MHC to MBU(1)	
I4HCRINP:3	BHHCRINP	MHC to MBU(2)	
I4HCRINP:4	BHHCRINP	MHC to MBU(3)	

Table B-14. X4 IPU Interface Signals (Continued)

MBU, AU - MASTER HARDCORE INTERFACE

MHC	UNIT - MBU(0-3)	DESCRIPTION	
¬I4CLRREQ:1	¬BHCLRREQ	MHC to MBU(0)	Reset MBU's hard core requirement flag (BHQHCREQ)
I4CLRREQ:2	BHCLRREQ	MHC to MBU(1)	
I4CLRREQ:3	BHCLRREQ	MHC to MBU(2)	
I4CLRREQ:4	BHCLRREQ	MHC to MBU(3)	
I4SETREQ:1	BHSETREQ	MHC to MBU(0)	Set MBU's BHQHCREQ
I4SETREQ:2	BHSETREQ	MHC to MBU(1)	
I4SETREQ:3	BHSETREQ	MHC to MBU(2)	
I4SETREQ:4	BHSETREQ	MHC to MBU(3)	
BHQUNCMP(0)	BHQUNCMP	MBU(0) to MHC	MBU's unit hard core "I've completed CCR servicing"
BHQUNCMP(1)	BHQUNCMP	MBU(1) to MHC	
BHQUNCMP(2)	BHQUNCMP	MBU(2) to MHC	
BHQUNCMP(3)	BHQUNCMP	MBU(3) to MHC	
BHQABTRM(0)	BHQABTRM	MBU(0) to MHC	"MBUHC's 'I've terminated due to an abnormal condition'"
BHQABTRM(1)	BHQABTRM	MBU(1) to MHC	
BHQABTRM(2)	BHQABTRM	MBU(2) to MHC	
BHQABTRM(3)	BHQABTRM	MBU(3) to MHC	
BCQPRVLT(0)	BCQPRVLT	MBU(0) to MHC	Protection violation due to normal program CM request
BCQPRVLT(1)	BCQPRVLT	MBU(1) to MHC	
BCQPRVLT(2)	BCQPRVLT	MBU(2) to MHC	
BCQPRVLT(3)	BCQPRVLT	MBU(3) to MHC	
BCQPAPER(0)	BCQPAPER	MBU(0) to MHC	Parity error due to normal program CM request
BCQPAPER(1)	BCQPAPER	MBU(1) to MHC	
BCQPAPER(2)	BCQPAPER	MBU(2) to MHC	
BCQPAPER(3)	BCQPAPER	MBU(3) to MHC	
BCQILOPR(0)	BCQILOPR	MBU(0) to MHC	Illegal ϕ prode encountered during normal program running
BCQILOPR(1)	BCQILOPR	MBU(1) to MHC	
BCQILOPR(2)	BCQILOPR	MBU(2) to MHC	
BCQILOPR(3)	BCQILOPR	MBU(3) to MHC	

Table B-14. X4 IPU Interface Signals (Continued)

MBU, AU - MASTER HARDCORE INTERFACE

MHC	UNIT - MBU(0-3)		DESCRIPTION
BHCMERR(0)	BHCMERR	MBU(0) to MHC	MBU's parity error or protection violations due to MBUHC CM request
BHCMERR(1)	BHCMERR	MBU(1) to MHC	
BHCMERR(2)	BHCMERR	MBU(2) to MHC	
BHCMERR(3)	BHCMERR	MBU(3) to MHC	
BHMBUZP(0)	BHMBUZP	MBU(0) to MHC	MBU's "Ive reached a zero pending CM request condition"
BHMBUZP(1)	BHMBUZP	MBU(1) to MHC	
BHMBUZP(2)	BHMBUZP	MBU(2) to MHC	
BHMBUZP(3)	BHMBUZP	MBU(3) to MHC	
¬BHUR0(0-7)	BHUR(0-7)	MBU(0) to PPU	MBUHC's unit register data
BHUR1(0-7)	BHUR(0-7)	MBU(1) to PPU	
BHUR2(0-7)	BHUR(0-7)	MBU(2) to PPU	
BHUR3(0-7)	BHUR(0-7)	MBU(3) to PPU	
I4IGNPAR:1	BIPAESTP	MHC to MBU(0)	Ignor parity errors (prevents loading of flags)
I4IGNPAR:2	BIPAESTP	MHC to MBU(1)	
I4IGNPAR:3	BIPAESTP	MHC to MBU(2)	
I4IGNPAR:4	BIPAESTP	MHC to MBU(3)	
I4CLEAR:5	AHMRCLR	MHC to AU(0)	"Master clear of CPU"
I4CLEAR:6	AHMRCLR	MHC to AU(1)	
I4CLEAR:7	AHMRCLR	MHC to AU(2)	
I4CLEAR:8	AHMRCLR	MHC to AU(3)	
¬I4HCCRR:5(12-15)	¬AHCCR(12-15)	MHC to AU(0)	Least significant hex of buffered common command register (I4QCCR(12-15))
I4HCCRR:6(12-15)	AHCCR(12-15)	MHC to AU(1)	
I4HCCRR:7(12-15)	AHCCR(12-15)	MHC to AU(2)	
I4HCCRR:8(12-15)	AHCCR(12-15)	MHC to AU(3)	
I4HCINIT:5	AHHCINIT	MHC to AU(0)	"Initiate CCR servicing"
I4HCINIT:6	AHHCINIT	MHC to AU(1)	
I4HCINIT:7	AHHCINIT	MHC to AU(2)	
I4HCINIT:8	AHHCINIT	MHC to AU(3)	

Table B-14. X4 IPU Interface Signals (Continued)

MBU, AU - MASTER HARDCORE INTERFACE

MHC	UNIT - MBU(0-3)	DESCRIPTION	
¬I4ABORT:5	¬AHABRT	MHC to AU(0)	"ABORT CCR servicing now"
I4ABORT:6	AHABRT	MHC to AU(1)	
I4ABORT:7	AHABRT	MHC to AU(2)	
I4ABORT:8	AHABRT	MHC to AU(3)	
I4HCWAIT:5	AHWA	MHC to AU(0)	"Don't proceed with CCR servicing yet"
I4HCWAIT:6	AHWA	MHC to AU(1)	
I4HCWAIT:6	AHWA	MHC to AU(2)	
I4HCWAIT:7	AHWA	MHC to AU(3)	
I4URSEL:5(0-3)	AHCCR(0-3)	MHC to AU(0)	Least significant hex of PPU Common Command Register (I4CRCCR(12-15)) used for selecting data to AHUR 0-7, 10-17, 20-27, 30-37
I4URSEL:6(0-3)	AHCCR(0-3)	MHC to AU(1)	
I4URSEL:7(0-3)	AHCCR(0-3)	MHC to AU(2)	
I4URSEL:8(0-3)	AHCCR(0-3)	MHC to AU(3)	
I4IGNPAR:5	AHIGNPA	MHC to AU(0)	Ignor all parity errors (prevents loading of AHQPA)
I4IGNPAR:6	AHIGNPA	MHC to AU(1)	
I4IGNPAR:7	AHIGNPA	MHC to AU(2)	
I4IGNPAR:8	AHIGNPA	MHC to AU(3)	
¬AHUNITCP(0)	¬AHUNITCP	AU(0) to MHC	AU's "I've completed CCR servicing"
AHUNITCP(1)	AHUNITCP	AU(1) to MHC	
AHUNITCP(2)	AHUNITCP	AU(2) to MHC	
AHUNITCP(3)	AHUNITCP	AU(3) to MHC	
¬AHABNTRM(0)	¬AHABNTRM	AU(0) to MHC	AU's "I've terminated due to an abnormal condition"
AHABNTRM(1)	AHABNTRM	AU(1) to MHC	
AHABNTRM(2)	AHABNTRM	AU(2) to MHC	
AHABNTRM(3)	AHABNTRM	AU(3) to MHC	
AHQPR(0)	AHQPR	AU(0) to MHC	AU's protection violation falg, caused by AUHC CM request
AHQPR(1)	AHQPR	AU(1) to MHC	
AHQPR(2)	AHQPR	AU(2) to MHC	
AHQPR(3)	AHQPR	AU(3) to MHC	

Table B-14. X4 IPU Interface Signals (Continued)

MBU, AU - MASTER HARDCORE INTERFACE

MHC	UNIT - MBU(0-3)	DESCRIPTION	
¬AHQPA(0)	¬AHQPA	AU(0) to MHC	AU's parity error flag, caused by AUHC CM request
AHQPA(1)	AHQPA	AU(1) to MHC	
AHQPA(2)	AHQPA	AU(2) to MHC	
AHQPA(3)	AHQPA	AU(3) to MHC	
¬AHUR0(0-7)	¬AHUR(0-7)	AU(0) to PPU	AU's unit register data
AHUR1(0-7)	AHUR(0-7)	AU(1) to PPU	
AHUR2(0-7)	AHUR(0-7)	AU(2) to PPU	
AHUR3(0-7)	AHUR(0-7)	AU(3) to PPU	

Table B-14. X4 IPU Interface Signals (Continued)

IPU - MASTER HARDCORE INTERFACE

MHC	UNIT - IPU	DESCRIPTION
I4RUNBIT:5	¬I4RUNEQO	MHC to IPUHC signal; indicating value of CPU run bit, I4QRB
I4CLEAR:9	I4CLEAR	MHC to IPUHC signal; indicating "master clear of CPU"
¬I4HCCCR:9(12-15)	¬I4CCR(12-15)	MHC to IPUHC signals; LS hex of buffered common command register
I4HCINIT:9	I4HCINIT	MHC to IPUHC signal; indicates "initiate CCR command"
I4URSEL:9(0-3)	I4UREN(0-3)	MHC to IPUHC signals; LS hex of PPU CCR, used to select data to Imurdata
I4ABORT9	I4ABORT	MHC to IPUHC signal; indicates "ABORT CCR servicing <u>now</u>"
¬I4HCWAIT:8	I4ZROPEN	MHC to IPUHC signal; indicates "Proceed with CCR servicing"
¬I4CLRREQ:5	¬I4SETREQ	MHC to IPUHC signal; set IMQHCREQ so that IMQFREEZ ← 1
¬I4HCCALL:9	¬I4HCCALL	MHC to IPU level 3; "Go ahead and write your message for the PPU."
¬I4PIPOFF(0-3)	¬IRPIPOFF(0-3)	IPU to MHC signals; indicate which MBU-AU pipes are not operational
¬I4HCABNT	¬IMHCABNT	IPUHC to MHC signal; indicates "I've terminated CCR servicing due to abnormal conditions
I4QIPPRV	ICQIPPRV	IPUHC to MHC flag; protection violation occurred during normal program run
I4QIPIOP	ICQIPIOP	IPUHC to MHC flag; illegal opcode encountered during normal program run
I4QIPPAE	ICQIPPAE	IPUHC to MHC flag; parity error occurred during normal program run
I4QAREX	ICQAREX	IPUHC to MHC flag; arithmetic exception occurred during normal program run
¬I4HCCOMP	¬IMHCCOMP	IPUHC to MHC signal; indicates "I've completed CCR servicing"
IrZRPEND	IPZROPEN	IPUHC to MHC signal; IPU has reached a "No CM requests pending" condition
I4CALCMP	IRCALCMP	IPU level 3 to MHC; "I've written my message to the PPU"
I4MCPREQ	IRMCPREQ	IPU level 3 to MHC; "I wish to write a message to the PPU"
I4MCWREQ	IRMCWREQ	IPU level 3 to MHC; "Iwish to write a message to PPU and do a program switch"
I4MEMERR	IMMEMERR	IPUHC to MHC flag; parity error or protection violation due to IPUHC CM request
I4ANYVBG	IMANYVBG	IPUHC to MHC flag; "There's at least one vector bad guy being processed"
¬IMDTUR(0-7)	¬IMURDATA(0-7)	IPUHC to PPU; unit register data from IPUHC and IPU CM requestor

Table B-14. X4 IPU Interface Signals (Continued)

SIGNALS FROM IPU TO MBU(n); n=0, 1, 2, 3

IPU	MBU	DESCRIPTION
¬IBnA00(0-15) ¬IBnA00(16-31) ¬IBnA01(0-15) ¬IBnA01(16-31)	¬BIMBN0L(0-15) ¬BIMBN0R(0-15) ¬BIMBN1L(0-15) ¬BIMBN1R(0-15)	Immediate operands and VPF data path into the IMM register.
¬IBnR00(0-15) ¬IBnR01(16-31) ¬IBnR01(0-15) ¬IBnR01(16-31)	¬BIMBR0L(0-15) ¬BIMBR0R(0-15) ¬BIMBR1L(0-15) ¬BIMBR1R(0-15)	Register operands to REG register data path
IBnA00(8-28)	BIAR(8-28)	Operand octet address to XBA or YBA register
IBnA00(29-31)	BARXA(29-31)	Operand word address to XA or YA register
IBnA01(0)	BARXA(32)	Operand halfword address bit to XA or YA register
InZP:1(8-28)	BIZP(8-28)	Scalar store octet address to NSA register
InZLSB(29-32)	BIZP(29-32)	Scalar store element address to ZA register
¬IFnV3(0) ¬IFnV3(1-3)	¬BIVIS(0) ¬BIVI(1-3)	VI field from VPF word 3 for vector initialization
IBQLDXA(n)*	BIAOTXA	Gates IBnA00(29-31) +IBnA01(0) into XA register
IBQLDXBA(n)*	BIAOTXBA	Gates IBnA00(8-28) into XBA register
IBQLDYA(n)*	BIAOTYA	Gates IBnA00(29-31) + IBnA01(0) into YA register
IBQIMM(n)*	BIIMMED	Gates¬IBnA00(0-31) +¬IBnA01(0-31) into IMM register
IBQREGDP(n)*	BIREGDP	Gates¬IBnR00(0-31) +¬IBnR01(0-31) into REG register
IBQZTXU(n)	BIZTXUDT	Causes Z X update under control of the zone modification bits
IBQZTYU(n)	BIZTYUDT	Causes Z Y update under control of the zone modification bits
IBQZEX(n)	BIZAEQZA	Indicates X address = Z address. Causes a Z→X update if a forced write request (IBQREQFW(n)) occurs.

*→Must be = 0 during vectors

Table B-14. X4 IPU Interface Signals (Continued)

SIGNALS FROM IPU TO MBU (n); n=0, 1, 2, 3

IPU	MBU	DESCRIPTION
IBQZEY(n)	BIYAEQZA	Indicates Y address = Z address. Causes a Z→Y update if a forced write request (IBQREGFW(n)) occurs.
IBQZUP(n)	BIXUPDT	Inhibits the setting of DPMBI until the Z→X update has occurred.
¬IBQYUP(n)	¬BIYUPDT	Inhibits the setting of DPMBI until the Z→Y update has occurred.
¬IRQVISTR(n)	¬BIVCINIT	Starts vector initialization in the MBU and causes the transfer of the VPF to the MBU.
¬IRVECWTC(n)	¬BIZBBUSY	Prevents a vector from generating a Z write request
IBBUADW(n)	BIDWA	IMM or B Doubleword size for scalars or vectors
IBBURDW(n)	BIDWB	REG or A
IPVDW(n)	BIDWC	C Doubleword size for C vector
IBBUASW(n)	BISWA	IMM or B Singleword size for scalars or vectors
IBBURSW(n)	BISWB	REG or A
IPVSW(n)	BISWC	C Singleword size for C vector
IBBUAHW(n)	BIHWA	IMM or B Halfword size for scalars or vectors
IBBURHW(n)	BIHWB	REG or A
IPVHW(n)	BIHWC	C Halfword size for C vector
IVZPTNS(n)	BIZPTNSA	Gates ZP register to NSA register
IBQSCKT4(n)	¬BISSCKT4	Indicates α or REG short circuit at LVL4.
IRAE(n)	BIAE	Indicates arithmetic exception has occurred
IBQFCSGT(n)	BIFCKSGT	The first clock of the μ sequence for the instruction at LVL4 is the same group time, i.e., the time at which an overlap can occur.
IBQSMGP4(n)	BISMGRP	Indicates that the instruction at LVL4 is in the same instruction group as the last instruction to enter MBU(n).

Table B-14. X4 IPU Interface Signals (Continued)

SIGNALS FROM IPU TO MBU(n); n=0, 1, 2, 3

IPU	MBU	DESCRIPTION
¬IBQOCK4(n)	¬BIOCKRMA	Indicates that the instruction at LVL4 is a one clock
IBQREQRW(n)	BIFRCDWR	Causes MBU(n) to initiate a forced write
IRQVISTR(n)	BIVINIT	Causes MBU(n) to begin vector initialization
¬IVnERDTC(0-3)	¬BIERDTC(0-3)	Indicates when data destined for the register file will be at LVL12 on the next clock
ICME:n, n=1,2,3,4	BIME	Map enable to MBU(0-3) memory port
ICPE:n, n=1,2,3,4	BIPE	Protect enable to MBU(0-3) memory port
¬InZHW	¬BIHWZ	Scalar Z buffer halfword size
¬InZDW	¬BIDWZ	Scalar Z buffer doubleword size
IBnRX4(0-3)	BIOPM(1-4)	MBU ROM address for scalar instructions
IBnRY4(0-3)	BIOPM(5-8)	
IBnRY4(4)	BIOPM(0)	Extended MBU ROM address for special instructions
¬IRVECWTA(n)	¬BIVECWTA	Prevents a vector from issuing read requests
¬IR4GETIT(n)	¬BI4GETIT	Causes MBU(n) to return to normal after a vector initialization
¬IMSUPRUN	¬BISSUPRN	Allows MBU to run with run bit off (hard core)
IVRFODL5(n)	BIR5(5)	R-field odd at LVL5
¬IMBGABRT(n)	¬BIBGABRT	Bad guy abort signal
IFnV2(0)	BIHS(0)	Option bit on peak pick, etc., to suppress item counts at end of self loops.

Table B-14. X4 IPU Interface Signals (Continued)

SIGNALS FROM IPU TO MBU (n); n=0, 1, 2, 3

MUB(n)	IPU	DESCRIPTION
BAQZA:1(8-28)	BAQZAn:1(8-28)	Data path for Z-buffer octet address (ZA) to the ZB register for hazard detection during vectors
BCQSYNC(4)	BCQSYNC4(n)	Indicate SC buffer in MBU(n) is full
BCCUEEQX(0)	BCCUEEQX(n)	Cue is equal to X, i.e., data in SC buffer is destined for the X buffer
BCCUEEQX(0)	BCCUEEQY(n)	Cue is equal to Y, i.e., data in SC buffer is destined for the Y buffer
BCVECEND	BCVECEND(n)	Indicates normal vector end
BCZMAL1	BCZMAL1(n)	Indicates zone modification bits all "1"
BHQDSCMP:1	BHQDSCMP:1(n)	De-escalate complete indicates all requests in CAF have been serviced
BMQROMOT:1(192)	BMQNCLCK(n)	Next clock is the last clock in the AU ROM seq
BMQROMOT:1(193)	BMQNCNOP(n)	Next clock is the NOP seq in the AU ROM
BMQROMOT:1(195)	BMQRMENB(n)	AU ROM enable
BMQROMOT:1(202)	BMQRMRM9(n)	AU ROM RM9
BMQROMOT:1(198)	BMQRMRMA(n)	AU ROM RMA Used for stack control
BMQROMOT:1(199	BMQRMRM8(n)	AU ROM RMB
BMQROMOT:1(200	BMQRMRMC(n)	AU ROM RMC
BMQROMOT:1(203)	BMQNCSGT(n)	Next clock is the same group time in the AU seq
BMQROMOT:1(205)	BMQAUPO(n)	Indicates, during the exeuction of a stack instruction, that the terminate signal from the AU can be sampled and that the stack pointer will be at LVL12 on the next clock
BMQROMOT:1(201)	BMQRMRM8(n)	AU ROM RM8
BCQDAV	BCQDAV(n)	DAV from MBU memory port
BCFWRTDN	BCFWRTDN(n)	Forced write done signal from MBU's CMR
BMQROMOT:1(207)	BMQRMERW(n)	Divide early window
BMQROMOT:1(208)	BMQRMLTW(n)	Divide late window

Table B-14. X4 IPU Interface Signals (Continued)

SIGNALS FROM IPU TO AU(n); n=0, 1, 2, 3

IPU	AU(n)	DESCRIPTION
IVSCRL6(n)	AIPUSCAB	Instruction at LVL6 has short circuit reg, i.e., EF→AB
IVSCAL6(n)	AIPUSCCD	Instruction at LVL6 has short circuit x, i.e., EF→CD
IVDHWL6(n)	AMHL	Instruction at LVL6 has halfword select
IVDDWL6(n)	AMDL	Instruction at LVL6 has doubleword select
ICME:n, n=5,6,7,8	AHMPEN	Map enable to AU(0-3) memory port
ICPE:n, n=5,6,7,8	AHPREN	Protect enable to AU(0-3) memory port

Table B-14. X4 IPU Interface Signals (Continued)

SIGNALS FROM AU(n) TO IPU; n=0, 1, 2, 3

AU(n)	IPU	DESCRIPTION
AOQOFFX:1	AOQOFFX:1(n)	Fixed point overflow
AOQDVCHK:1	AOQDVCHK:1(n)	Divide check
AOQOFFL:1	AOQOFFL:1(n)	Floating point overflow
AOQUFFL:1	AOQUFFL:1(n)	Floating point underflow
AOQRL:1	AOQRL:1(n)	RL Result code bits
AOQRE:1	AOQRE:1(n)	RE
AOQRG:L	AOQRG:1(n)	RG
AOQCL	AOQCL(n)	CL Compare code bits
AOQCE	AOQCE(n)	CE
AOQCG	AOQCG(n)	CG
¬AOTRMSTK	¬AOTRMSTK(n)	Stack instruction terminate signal
AVQET:1	AEQET:1(n)	Skip indicator
AOQEF:2(0-63)	AOQEFn:2(0-63)	AU output register doubleword

Table B-15. X4 IPU Glossary

TERM	SIG	LOC	DESCR
ACT(OP)	ICACTO	I4CMREQ	Signal indicating that the active bit pointed to by the cue output pointer is true.
ACTIVITY LVL6-LVL11(0-3)	IVL6TLBA	I4INFACE (0-3)	Signal indicating activity at levels 6-11 for pipe(0-3).
ACTIVITY LVL7-LVL11(0-3)	NONE	I4INFACE (0-3)	Signal indicating there is at least one instruct in levels 7-11.
ACTIVITY LVL8-LVL11(0-3)	IVQAUIAC	I4INFACE (0-3)	Flag indicating there is an instruction at levels 8-11 of pipe(0-3).
ADW	IRQRMADW	I4INFACE (0)	ROM bit indicating that AR contains a doubleword address.
AE HAZ(0-3)	IRAEH(0-3)	I4ZHAZ (3,7,11, 15)	Signal indicating the presence of at least one AEH bit in the register stack (levels 5-12) for pipe (0-3) or at lvl4.
AE POSSIBLE	IRQC3AEH	I4INFACE (1)	ROM bit indicating that the instruction at lvl3 could cause an arithmetic exception.
ALLOW CURRENT	IRQR3(5)	I4PIPE (5)	Flag in the R-field of the vector instruction indicating (IF FORK IND,=1) when the current vector instruction can proceed independently.
ALLOW FOLLOWING	IRQR3(6)	I4PIPE (6)	Flag in the R-field of the vector instruction which, if it is a one(zero), sets (resets) the fork ind.
ALL PIPES EMPTY	IRAPIPMT	I4ROUTE2	Signal indicating there is no activity in any of the pipes levels 4-12.
ALL ZMB(0-3)	IRQZMAL1(0-3)	I4ROUTE3	This is the MBU signal BCZMAL1(0-3) delayed by one clock. It indicates that all of the zone modification bits for the Z-buffer in pipe(0-3) are ones, i.e., that every halfword in the Z-buffer has new data.

Table B-15. X4 IPU Glossary (Continued)

TERM	SIG	LOC	DESCR
ANY JOINED REQUESTS	IRANYRJN	I4MIXC	Signal in lvl3 controller indicating at least one join request flag set.
ANY NEAR RANGE HAZARD	IRANRHAZ	I4LVL3	Indicates comparison of P3 and any ZP (IRP3EQZP(0-3)) or ZB (IRP3EQZB(0-3)) address at the octet level.
ANY REG. DEST.	IRARGDST	I4ROUTE1	Signal in the lvl3 controller indicating at least one RDT bit in one of the register stacks for pipes(0-3).
ANY RHAZ	IRNORHAZ	I4ROUTE1	Signal from lvl3 controller indicating that none of the RHAZ(0-3) is true.
ANY R-OCTET HAZ	IRANYROH	I4VECLAS	Signal in the lvl3 controller indicating at least one R-octet HAZ(0-3).
ANY SP	IRANYSP	I4VECLAS	Signal in lvl3 controller indicating that a pipe has been selected, i.e., one of the SP(0-3) flipflops has been set. See SP(0-3).
ANY STF HAZ	IRANYSFH	I4ROUTE2	Signal indicating a store file hazard consisting of: AR=XA and ¬XAFUL or AR=YA and ¬YAFUL for at least one pipe(0-3).
ANY T3 HAZ	IRNOARHZ	I4ROUTE1	See "ANY α-RHAZ." The register used for this comparison is IRAROT3(26-32). Except for a BCLE or BCG instruction, AR data is selected into this register and used for the α-RHAZ comparison. For a BCLE or BCG instruction T3 is selected and the comparison becomes "ANY T3 HAZ."
ANY VHAZ	IRANYVHZ	I4VECLAS	Vector hazard signal indicating at least one VHZ bit is on in one of the register stacks for pipes(0-3). This indicates when the index or vector registers are going to be modified by an instruction in one of the pipes.

Table B-15. X4 IPU Glossary (Continued)

TERM	SIG	LOC	DESCR
ALL ZPFUL	NONE	I4ROUTE2	Signal indicating all ZPFUL(0-3) are true.
α HAZ(0-3)	NONE	I4ROUTE2	Signal from lvl3 controller indicating AR=ZP (IRAREQZP(0-3) or AR=ZB (IRAREQZB(0-3) for pipe(0-3) at the octet level.
α HAZ FLAG	IRQARHAZ	I4VECLAS	Flag indicating that during a vector running in the joined mode with the "Allow Following" bit on, the vector modified the region from which ~~one~~ of the following instructions was taken.
α-RHAZ(0-3)	IRARRHZ(0-3)	I4RHAZ (0-3)	Signal indicating AR compares with some register stack address (levels 4-12) for pipe(0-3).
α-REG. OCTET HAZ(0-3)	IRAROHAZ(0-3)	I4RHAZ (0-3)	Signal indicating AR compare with some register stack address (levels 4-12) at the octet level for pipe(0-3).
ANY AE HAZ	IRAAEHZ	I4ROUTE1	Signal in lvl3 controller indicating the presence of at least one AE HAZ(0-3).
ANY α HAZ	IRAREZPB	I4ROUTE2	Signal indicating at least one ZP(0-3) or ZB(0-3) equals AR.
ANY α-RHAZ	IRNOARHAZ	I4ROUTE1	Signal indicating no α-RHAZ(0-3) is true.
ANY α-REG.OCTET HAZ	IRAARROH	I4VECLAS	Signal in the lvl3 controller indicating at least one α-REG. OCTET HAZ(0-3).
ANY DIVIDE CHECK	IRDVCHK	I4STATUS	Signal indicating a divide check has occurred in at least one of the AU(0-3).
ANY FIXED POINT OVERFLOW	IROFFX	I4STATUS	Signal indicating a fixed point overflow has occurred iin at least one of the AU(0-3).
ANY FLOATING POINT OVERFLOW	IROFFL	I4STATUS	Signal indicating a floating point overflow has occurred in at least one of the AU(0-3).
ANY FLOATING POINT UNDERFLOW	IRUFFL	I4STATUS	Signal indicating a floating point underflow has occurred in at least one of the AU(0-3).

Table B-15. X4 IPU Glossary (Continued)

TERM	SIG	LOC	DESCR
ANY VIP	IRANYVIP	I4VECLAS	From lvl3 controller indicating at least one vector in progress FLAG (IRQVIP(0-3)) is set.
ANY Z JOIN	IREMJNPI	I4VECLAS	Signal indicating at least one Z-JOIN(0-3) (IBQZJOIN(0-3)) set.
AORO. AVAIL.	IRAOROA	I4ROUTE1	This signal in the lvl3 controller indicates that the AO and RO registers are free.
ARITHMETIC EXCEPTION	ICQAREX	I4HDCORE	Flag indicating an arithmetic exception has occurred.
ARTZP(0-3)	IRARTZP(0-3)	I4ROUTE3	Signal from lvl3 controller indicating a store is going from lvl3 to lvl4.
AR=LA	IRAREQLA	I4ZHAZ(3)	Signal generated on I4ZHAZ indicating AR=LA on the octet level.
AR=LA or AR=PA	ILARELVP	I4CMREQ	Signal indicating AR=LA or AR=PA at the octet level.
AR=LA, PA, P1, or P2	IRLCLBR	I4LVL3	Signal from lvl3 controller indicating AR=P1 or P2 at the word level, or AR=PA or LA at the octet level.
AR=LD(0-3)	IRARVSLD(0-3)	I4ZHAZ (0,4,8, 12)	Signal indicating AR=LD for pipe(0-3).
AR=PA	IRAREQPA	I4ZHAZ(2)	Signal generated on I4ZHAZ indicating AR=PA at the octet level.
AR=P1	IRARVSP1	I4ZHAZ(1)	Signal indicating AR=P1 at the word level.
AR=P2	IRARVSP2	I4ZHAZ(2)	Signal indicating AR=P2 at the word level.
AR=XA(0-3)	IRAREQX(0-3)	I4ZHAZ (0,4,8, 12)	Signal indicating AR=XA at the octet level fo‸ pipe(0-3).
AR=YA(0-3)	IRAREQY(0-3)	I4ZHAZ (1,5,9, 13)	Signal indicating AR=YA at the octet level for pipe(0-3).

Table B-15. X4 IPU Glossary (Continued)

TERM	SIG	LOC	DESCR
AR=ZP(0-3)	IRAREQZP(0-3)	I4ZHAZ(3, 7,11,15)	Signal indicating AR=ZP at the octet level for pipe(0-3).
AR=ZB(0-3)	IRAREQZB(0-3)	I4ZHAZ(1, 5,9,13)	Signal indicating AR=ZB at the octet level for pipe(0-3).
AR=0	IRAREQ0	I4ROUTE1	Signal from lvl3 controller indicating AR(8-31)=0.
AR$\leq$2F	NONE	I4ROUTE1	Composed of signals from I4PIPE indicating AR(8-23) =0 and AR(24-27) $\leq$ 2.
AVAIL(0-3)	IBPAVAIL(0-3)	I4INFACE (0-3)	This signal indicates that pipe(0-3) is available, i.e., that pipe(0-3) is turned on, no vector is in pipe(0-3), and lvl5 of pipe(0-3) will be empty on the next clock.

Table B-15. X4 IPU Glossary (Continued)

TERM	SIG	LOC	DESCR
BAD GUY ABORT(0-3)	IMBGABRT(0-3)	I4HDCORE	Signal indicating a non-recoverable switch is about to take place while a vector bad guy is running in pipe(0-3).
BAE	IRQDCBAE	I4INFACE (0)	Flag decoded from the instruction Op code indicating a BAE instruction.
BCC	IRQDCBCC	I4INFACE (0)	Flag decoded from the instruction Op code indicating a BCC instruction.
BCLE,BCG	IRBCGBCL	I4ROUTE1	Signal indicating a BCLE or BCG instruction at lvl3.
BLB,BLX	IRQDCBBX	I4INFACE (0)	Flag decoded from the instruction Op code indicating a BLB or BLX instruction.
BLUE INSTR. A1 LVL3	IRQRMBLU	I4INFACE (1)	ROM bit indicating a BLUE instruction at lvl3.
BOGUSA	ICBOGUSA	I4CMREQ	Signal used to cancel any requests in the cue for the KA buffer by turning off the active bit for that request.
BOGUSB	ICBOGUSB	I4CMREQ	Signal used to cancel any requests in the cue for the KB buffer by turning off the active bit for that request.
BRANCH DONE FLAG	IRQBRDN	I4LVL3	Flag controlled by the lvl3 controller, used to delay by one clock the testing of the branch results for the BCLE and BCG instructions. It is also used to indicate when the branch portion of the BLB, BLX instructions has been finished.
BRANCH INST.	IRQRMBRH	None	ROM bit indentifying all branch instructions.
BRANCH NOT TAKEN	IRBRNTRN	I4LVL3	Signal from lvl3 controller indicating the branch at lvl3 was not taken.
BRANCH TAKEN	IRBRTKN	I4ROUTE1	Signal indicating for all conditional branches that a branch will be taken.

Table B-15. X4 IPU Glossary (Continued)

TERM	SIG	LOC	DESCR
BRANLA	IRBRTLA	I4LVL3	Signal from lvl3 controller indicating a branch to LA, i.e., to the instruction octet pointed to by LA.
BRANOA	IRBRTOA	I4LVL3	Signal from lvl3 controller indicating a branch to OA, i.e., to CM.
BRANPA	IRBRTPA	I4LVL3	Signal from lvl3 controller indicating a branch to PA, i.e., to the current instruction octet.
BRANP1	IRBRTP1	I4LVL3	Signal from lvl3 controller indicating a branch to lvl1.
BRANP2	IRBRTP2	I4LVL3	Signal from lvl3 controller indicating a branch to lvl2.
BRC	IRQDCBRC	I4INFACE (0)	Flag decoded from the instruction Op code indicating a BRC instruction.
BRHAZ	IRBRHAZ	I4LVL3	Lvl3 signal indicating that a branch at lvl3 must wait for a hazard to clear.
BROWN INSTR. AT LVL3	IRQDCBWN	I4I4FACE (1)	Flag decoded from instruction Op code indicating a BROWN (MCP,MCW) instruction at lvl3.
BXEC	IRQDCBXC	I4INFACE (1)	Flag decoded from the instruction Op code indicating a BXEC instruction.

Table B-15. X4 IPU Glossary (Continued)

TERM	SIG	LOC	DESCR
CALL PERMISSION IND.	IRQHCALI	I4LVL3	This flag is the I4HCCALL signal from master hard core (I4MHC) delayed by one clock.
CC ROUTE(0-3)	IRQCCRT(0-3)	I4STATUS	Flag which indicates that pipe(0-3) was the last to receive an instruction that could modify the compare code register.
CHZ BIT AT LVLR (0-3)	IVCHZLC(0-3)	I4ZHAZ (1,5,9, 13)	Flag from the register stack for pipe (0-3) indicating an instruction is at lvl12 that could modify the compare code register.
CMTFILE	ICCMTFIL	I4CMREQ	Signal enabling the transfer of KCM to one of the register file octets. This signal indicates that the transfer will occur on the next clock.
CMTIR	ICCMTIR	I4CMREQ	Signal from CMR indicating that an instruction or an indirect cell will be gated into IR from KCM on the next clock.
COMP. CTR.=0-7	IRCCTREQ(0-7)	I4VECLAS	Signal in the lvl3 controller indicating that the complete counter (IRQCCTR(0-2) contains 0-7. This counter is used by both the LF,LFM, and VECT instructions.
CUE EMPTY	ICCUEMPY	I4CMREQ	Signal indicating that none of the busy bits in the cue is on, i.e., there are no outstanding memory requests.
CUE FULL	ICCUEFUL	I4CMREQ	Signal indicating that all 4 busy bits in the CUE are on, i.e., that no more memory requests can be made.

Table B-15. X4 IPU Glossary (Continued)

TERM	SIG	LOC	DESCR
DAV(0-)FROM MBU(0-3)	BCQDAV(0-3)	MBU(0-3)	When this signal goes false it indicates the write in progress has reached memory.
DAV FROM IPU	ICDAV	I4HDCORE	When this signal goes false it indicates the write in progress has reached memory.
DIVIDE(0-3)	IBLSTDIV(0-3)	I4INFACE (0-3)	This signal is decoded from the same group flipflops at lvl4 and indicates the last instruction down pipe(0-3) was a divide.
DIVIDE FLOATING (0-3)	IBDF(0-3)	I4INFACE (0-3)	This signal is decoded from the same group flipflops at lvl4 and indicates the last instruction down pipe (0-3) was a divide floating.
DPMBI(0-3)	IVQDPMBI(0-3)	I4INFACE (0-3)	Lvl4 flag which indicates when the data required by an instruction at lvl5 is available.
DPMBO(0-3)	IVQDPMBO(0-3)	I4INFACE (0-3)	Flag indicating an instruction is at lvl6 of pipe(0-3).
DUAL&BRNTKN	IRDLBNT	I4LVL3	Signal from lvl3 controller indicating the lookahead controller in the dual mode and the branch will not be taken.
DUAL MODE	ILQDUAL	I4CMREQ	Flag indicating the lookahead controller is in the dual mode, i.e., that the branch octet is being fetched while a branch is waiting at lvl3.

Table B-15. X4 IPU Glossary (Continued)

TERM	SIG	LOC	DESCR
EARLY WINDOW(0-3)	IRQWNDER(0-3)	I4CMREQ	Signal from AU ROM for pipe(0-3) (BMQRMERW(0-3)) delayed by one clock. It indicates that the divide in pipe(0-3) is at a point such that the divide at lvl3 in the same group could reach the AU in time to save the divide initialization time by overlapping. This window includes memory fetch time.
EXPECT REG.DEST. AT LVL12(0-3)	IVnERDTC(0-3), n=1,2,3	I4INFACE (0-3)	Signal sent to MBU(n) indicating that on the next clock an instruction with a register destination will be at lvl12 of pipe (0-3).

Table B-15. X4 IPU Glossary (Continued)

TERM	SIG	LOC	DESCR
FAR RANGE HAZ.FLAG AT LVL3	IRQFRHZ	I4LVL3	Flag carried with the instruction set by a comparison of LA, PA, P1 or P2 with any ZB address, or set by lvl3 controller by comparison of P3 with any ZB address during execution of a joined vector.
FILE→FILE	IRFILTFL	I4VECLAS	Signal from lvl3 controller used to effect a transfer of one file to another in the register file.
FIRST CLOCK SAME GROUP	IRBQFCSGT(0-3)	I4INFACE (0-3)	Lvl4 flag indicating that the instructions in the same group as the one at lvl4 have a same group time on their first μ-sequence. This bit is the ROM-2 IRQRMSGT bit latched at lvl4.
FLAGFUL	ILQFLGFL	I4CMREQ	Flag indicating the target branch is in the lookahead buffer and that the current buffer is the one pointed to by the target branch, i.e., both octets needed after the branch is taken are resident.
FLAG4	ILQFLG4	I4CMREQ	Flag used by lookahead controller to indicate that the target branch is in levels 1-3.
FLAG12	ILQFLG12	I4CMREQ	Flag indicating that the octet pointed to by the target branch has been requested or is resident.
FORCED WRITE COMPLETE(0-3)	IBFWCOMP(0-3)	I4INFACE (0-3)	Signal from the ZBFUL controller indicating the forced write in pipe(0-3) is complete.
FORCED WRITE WAIT (0-3)	IBFWRWT(0-3)	I4INFACE (0-3)	Signal from the forced write controller on I4INFACE indicating a forced write in progress.
FORCED WRITE WAIT (SP)	NONE	I4VECLAS	This is the normal forced write wait signal with the array signal being supplied by the selected pipe flipflops. See (0-3).
FORK	IRQDCFORK	I4INFACE (1)	Flag decoded from instruction Op code indicating a FORK instruction.

Table B-15. X4 IPU Glossary (Continued)

TERM	SIG	LOC	DESCR
FORK IND.	IRQFORK	I4PIPE(5)	Flag set by the FORK instruction or by the allow following bit (IRQR3(6)=1) of a vector. Reset by the join instruction or by the allow following bit (IRQR3(6)=0) of a vector. This flag=1 allows subsequent instructions to proceed independently (FORK MODE).
GETOUT	IMGETOUT	I4HDCORE	Signal used to reset the lvl3 controller during a joined vector bad guy when an error condition occurs in the unit hardcore.
GRAY INSTR. AT LVL3	IRQRMGRY	I4INFACE (0)	ROM bit indicating a gray (all sub-colors) instruction at lvl3.
GREEN INSTR. AT LVL3	IRQRMGRN	I4INFACE (1)	ROM bit indicating a green instruction at lvl3.

Table B-15. X4 IPU Glossary (Continued)

TERM	SIG	LOC	DESCR
HANDLE JN	IRHNDLJN	I4LVL3	Signal from the lvl3 controller indicating there is a JOIN or FORK instruction at lvl3 which needs servicing by the R/Z join controller.
HC GATE OA	IMGOA	I4HDCORE	Signal from the unit hardcore used to initiate a hardcore memory request.
HCREQ	IMHCREQ	I4HDCORE	Signal from unit hardcore indicating that hardcore requies the use of the CMR.
HC STORE	IMSTORE	I4HDCORE	Signal from the unit hardcore indicating that the current memory request from the hardcore is a store.
HDW	IRQHDW	I4LVL3	Lvl3 flag indicating when R3 should be taken as a doubleword address in the hazard comparisons.
HEX REG. HAZ.	NONE	I4ROUTE1	Signal in lvl3 controller indicating that there exists an instruction in one of the pipes that can modify one of the hex registers: AE, CC, RC.
HOLD FLAG	IRQHOLD	I4LVL3	This flag is used as a steering mechanism in the lvl3 controller for the green, pink, brown, and orange instructions.
HOLD PTR.	IRHLDPTR	I4LVL3	Signal from lvl3 controller indicating that the second pass of A PSH, PUL, or MOD instruction must hold at lvl3. This signal prevents an other instruction from moving into lvl2 on top of the stack pointer.

Table B-15. X4 IPU Glossary (Continued)

TERM	SIG	LOC	DESCR
IHAZ	IRQL3IHZ	I4LVL3	Flag indicating that the lvl3 controller is in the instruction hazard state.
ILLEGAL OP CODE	IPQRMIOP	I4INFACE (3)	ROM bit indicating an unused op code that does not have a first hex of zero.
ILOP FLAG	IRQILOP	I4LVL3	Set by lvl2 controller when the current instruction has an illegal op code, the current indirect cell does have zeros in the most significant hex, or there is a spec. error at lvl2.
IMM(0-3)	IBQIMM(0-3)	I4INFACE (0-3)	This flag is sent to MBU(0-3) to turn on the input controller for all instructions that do not require a memory operand.
INC AR	IRINCAR	I4VECLAS	Signal from lvl3 controller used to increment AR by 8 during LFM and STFM instructions.
INDIRECT INSTR.	IRQIND	I4LVL3	Set by lvl2 controller as indirect instr. is passed to lvl3. Reset when terminal indirect cell is sent to lvl3 or branch is not taken.
IND.INSTR.AT LVL1	IIINDAT1	I4PIPTOP	Signal indicating that an indirect instruction is at lvl1.
IND.INSTR.AT LVL2	IPQIND	I4PIPTOP	Lvl2 flag indicating that an indirect instruction is at lvl2.
INDIRECT CELL AT LVL2	IRQIND	I4LVL3	When lvl2 is active, this indicates cell is at lvl2.
INDIRECT REQ.	IREXIND	I4LVL3	Signal from lvl3 controller indicating that an indirect cell or the object of an execute instruction is being requested from CM.
INSTR	ILINSTRP* ILINSTR1* ILINSTR2	I4CMREQ	Signal from the lookahead controller initiating a memory request for an instruction octet.

Table B-15. X4 IPU Glossary (Continued)

TERM	SIG	LOC	DESCR
INSTR.AT LVL4 IS IN SAME GROUP AS LAST INSTR.(0-3)	IBQSMGP4(0-3)	I4INFACE (0-3)	Lvl4 flag indicating that the instruction at lvl4 going to pipe(0-3) is in the same group as the last instruction to enter that pipe.
INSTR. AT LVL5 IS IN SAME GROUP AS LAST INSTR. (0-3)	IVQSMGP5(0-3)	I4INFACE (0-3)	Flag indicating the instruction at lvl5 in pipe(0-3) is in the same group as the last instruction down that pipe.
INSTR.HAZ.RECOVERY REQ	IRINHAZ	I4LVL3	Signal from lvl3 IHAZ state indicating all near range HAZ. have cleared and the instruction octet can be refetched.
INSTR.PV.	IRQPRV	I4LVL3	Carried with the instruction to lvl3 to indicate that the CM acquisition of the octet containing this instruction resulted in a memory protect violation.

Table B-15. X4 IPU Glossary (Continued)

TERM	SIG	LOC	DESCR
JOIN	IRQDCJN	I4INFACE (1)	Flag decoded from the instruction op code indicating a join instruction.
JOIN FLAG	IRQJOIN	I4VECLAS	Flag controlled by lvl3 indicating when a joined vector is at lvl3.
KAFUL	ICQKAFUL	I4CMREQ	Flag controlled by the CMR which indicates when KA has valid data.
KA INSTR. HAZ.	ICQKAHAZ	I4CMREQ	Flag controlled by the CMR which indicates that LA=ZB for the octet currently in KA.
KBFUL	ICQKBFUL	I4CMREQ	Flag controlled by the CMR indicating when KB has valid data.
KB INSTR. HAZ	ICQKBHAZ	I4CMREQ	Flag controlled by the CMR indicating that LA=ZB for the octet currently in KB.
KCM FULL	ICCMFUL	I4HDCORE	Signal which is true for one clock after ICQRDA has been toggled indicating that read data has been put into KCM. KCM full and PRV(OP) cannot both be true at the same time.
KRTAG	ICQKRTAG	I4CMREQ	Flag controlled by the CMR which points to the current instruction buffer, i.e., if KRTAG is true KB is the current instruction buffer, otherwise KA is.

Table B-15. X4 IPU Glossary (Continued)

TERM	SIG	LOC	DESCR
LAC BIT AT LVL12 (0-3)	IVLACLC(0-3)	I4ZHAZ (0-3)	Flag in the register stack for pipe(0-3) indicating a LAC or LEM instruction is at lvl12.
LAORD	ILQLAORD	I4CMREQ	Flag indicating LA=PA+8.
LAM BIT AT LVL12 (0-3)	IVLAMLC(0-3)	I4ZHAZ (0,4,8, 12)	Flag in the register stack for pipe(0-3) indicating a LAM or LEM instruction is at lvl12.
LAM,LAC BITS	IRLAEM	I4ROUTE1	Signal in the lvl3 controller indicating at least one LAM or LAC bit in the register stacks for pipes(0-3).
LAM,LAC,LEM	IRLAEMIN	I4ROUTE1	Signal in lvl3 controller indicating the instr. at lvl3 is LAM,LAC, or LEM, i.e., IRQDCLAM or IRQDCLAC is true.
LAST INSTR. AT LVL5 A ONE CLOCK(0-3)	IVQOCKL5(0-3)	I4INFACE (0-3)	This flag is set when a one clock instruction enters lvl5. Since it is not reset when the instruction goes to lvl6, if MBIACT(0-3) is not true, it indicates whether the last instruction at lvl5 was a one clock.
LAST WRITE SIGNAL FROM MUB(0-3)	BCFWRTDN(0-3)	MBU(0-3)	Signal from MBU(0-3) indicating either that the last write of a vector has occurred or that the forced write in progress has finished.
LATE WINDOW(0-3)	IRQWNDLT(0-3)	I4CMREQ	Signal from AU ROM for pipe(0-3) (BMQRMLTW(0-3)) delayed by one clock. It indicates that the divide in pipe(0-3) is at a point such that the divide at lvl3 in the same group could reach the AU in time to save the divide initialization time by overlapping.
LC=4	ILLCEQ4	I4CMREQ	Signal indicating that PBACT or LLAACT is true and the loop counter (ILQLC(0-7)) plus the vacany flipflops (ILQVAC(0-1)) equals 4.

Table B-15. X4 IPU Glossary (Continued)

TERM	SIG	LOC	DESCR
LC<4	ILLCLT4	I4CMREQ	Signal indicating that PBACT or LLAACT is true and the loop counter (ILQLC(0-7)) plus the vacancy flipflops (ILQVAC (0-1)) is less than 4.
LC≤12	ILLTEQ12	I4CMREQ	Signal indicating that PBACT or LLAACT is true and the loop counter (ILQ2C(0-7)) is ≤ 12.
LDXA(0-3)	IBQLDXA(0-3)	I4INFACE (0-3)	Lvl4 flag sent to MBU(0-3) to indicate a memory operand is to be taken from the X-buffer by the instruction at lvl4.
LDXBA(0-3)	IBQLDXBA(0-3)	I4INFACE (0-3)	Lvl4 flag sent to MBU(0-3) to initiate an X-buffer fetch from CM.
LDYA(0-3)	IBQLDYA(0-3)	I4INFACE (0-3)	Lvl4 flag sent to MBU(0-3) to indicate a memory operand is to be taken from the Y-buffer by the instruction at lvl4.
LDYBA(0-3)	IBQLDYBA(0-3)	I4INFACE (0-3)	Lvl4 flag sent to MBU(0-3) to initiate a Y-buffer fetch from CM.
LEADING ZEROS	IPQDCNOP	I4INFACE (0)	Lvl2 flag decoded from the instruction op code indicating the hex of the instruction is zero.
LEVELS 5 or 6 ACTIVE(SP)	IR5O6SPA	I4VECLAS	Signal in lvl3 controller indicating that lvl 5 or 6 of the pipe pointed to by (0-3) is active. See SP(0-3).
LF REQ	IRLFREQ	I4VECLAS	Signal used to make a memory request during a LF or LFM instruction.
LLA	IRQDCLLA	I4INFACE (2)	Flag decoded from the instruction op code indicating a load lookahead instruction.
LLAXFER	IRLLXFER	I4LVL3	Signal from lvl3 controller indicating an LLA instruction at lvl3.
L,NI,OR NO=0	IRVCTNOP	I4VECLAC	Signal in the lvl3 controller indicating that the self-loop, inner-loop, or outer-loop count in the VPF is equal to zero. This is a vector NOP.

Table B-15. X4 IPU Glossary (Continued)

TERM	SIG	LOC	DESCR
LOADIR	ILLOADIR	I4CMREQ	Signal from CMR used to gate an instruction or indirect cell into the IR register.
LOCAL INDIRECTO TO IR	IRLCLIND	I4LVL3	Signal from lvl3 controller indicating that an indirect cell or the object of an execute instruction is to be taken from KA, KB, or the register file.
LOOK AHEAD BUFFER CLEAR	ILBUFCLR	I4CMREQ	Signal indicating that the target branch of a PB instruction is in the current, i.e., the buffer pointed to by PA.
LOOKAHEAD BUFFER CLEARING	ILBUFCNG	I4CMREQ	Signal indicating that the target branch of a PB instruction is in the buffer pointed to by LA, but will be in the buffer pointed to by PA on the next clock.
LVL1 ACTIVE	IIQL1ACT	I4PIPTOP	Set and reset by the lvl0 controller to indicate the presence of an instruction or data at lvl1.
LVL1 ADV	IIL1ADV	I4PIPTOP	Signal from lvl1 controller indicating that the data in lvl1 will advance to lvl2.
LVL1 BLOCK	IIL1BLK	I4PIPTOP	Signal from lvl1 controller indicating that nother should move into lvl1.
LVL1 HAZ STATE	IIQS(10)	I4PIPTOP	Lvl1 HAZ state flipflop.
LVL1 IND. AT 3 STATE	IIQS(3)	I4PIPTOP	Lvl1 flag indicating the lvl1 controller is in the indirect at 3 state.
LVL1 INSTR. M HAZ FREE	IIFDMHF	I4HDCORE	Signal decoded from the instruction op code indicating that the instruction at lvl1 cannot be base relative.
LVL1 INSTR. T HAZ FREE	IIFDTHF	I4HDCORE	Signal decoded from the instruction op code indicating that the instruction at lvl1 cannot be indexed.

Table B-15. X4 IPU Glossary (Continued)

TERM	SIG	LOC	DESCR
LVL1 INSTR. T HAZ FREE	IIFDTHF	I4HDCORE	Signal decoded from the instruction op code indicating that the instruction at lvl1 cannot be indexed.
LVL1 XEC AT 3 STATE	IIQS(4)	I4PIPTOP	Lvl1 flag indicating that the lvl1 controller is in the execute at 3 state.
LVL2 ACTIVE	IPQL2ACT	I4PIPTOP	Set and reset by the lvl1 controller to indicate the presence of an instruction or data at lvl1.
LVL2 BLOCK	IPL2BLK	I4PIPTOP	Signal from lvl1 and lvl2 controllers indicating that nothing should move into lvl2.
VLV3 ACTIVE	IRQL3ACT	I4LVL3	Set and reset by lvl2 controller to indicate the presence of an instruction or data at lvl1.
LVL3 CHECK STATE	IRSTK2ND	I4ROUTE3	Signal indicating that lvl3 controller is in the result check state.
LVL3 IN PROGRESS	IRLVL3IP	I4LVL3	Signal indicating that IRQHOLD, IRQXEC, IRQBRDN, IRQOPDN, or IRQRIHIB is set.
LVL3 PUSH PULL STATE	IRQL3PPL	I4LVL3	Flag indicating the lvl3 controller is making the 3rd pass of a push or pull instruction.
LVL3→LVL4(0-3)	IRIRT4P(0-3)	I4ROUTE2	Signal indicating that the data at lvl3 is going to the lvl4 controller for pipe(0-3).
LVL4→LVL5	IRL4TL5	I4MISC	Signal indicating any lvl4 to lvl5 transfer for pipes(0-3).
LVL7 ACTIVE AND NOT A ONE CLOCK(0-3)	NONE	I4INFACE (0-3)	Signal indicating IVQL7ACT(0-3) is true and IVQOCKL7(0-3) is not true for pipe(0-3).
LVL11 ACTIVE(0-3)	IVQACTLB(0-3)	I4ZHAZ (1,5,9, 13)	Register stack bit indicating an instruction at lvl11 in pipe(0-3).
L4RJN	IBQL4RJN	I4MISC	Flag indicating the instruction is a joined read.

Table B-15. X4 IPU Glossary (Continued)

TERM	SIG	LOC	DESCR
MBIACT(0-3)	IVQMBIAC(0-3)	I4INFACE (0-3)	Flag to indicate the presence of an instruction at lvl5 in IPU's MBU model.
MCP INSTR.	IRQDCMCP	I4INFACE (2)	Flag decoded from instruction op code.
M HAZ FREE	IPQDCMHF	I4INFACE (0)	Flag at lvl2 decoded from the instruction op code indicating that the instruction at lvl2 cannot be base relative.
MODE(0-3)	IBQMODE(0-3)	I4INFACE (0-3)	Flag at lvl4 indicating the lvl4 controller is holding off a Z→X/Y update until either the X/Y-buffer has data or until all Z-stores clear ptpe(0-3).
MODIFY CC	IRQC3CHZ	I4INFACE (1)	ROM bit indicating that the instruction at lvl3 could modify the compare code register.
MODIFY RC	IRQC3RHZ	I4INFACE (1)	ROM bit indicating that the instruction at lvl3 could modify the result code register.
MULTIPLE AR=ZB	IRZBPEN	I4ROUTE2	Signal indicating that more than one ZB(0-3) compares with AR. This can occur during certain pairs of vectors.
MULTIPLE INSTR.	IRQDCMLT	I4INFACE (2)	Flag decoded from the instruction op code indicating a LFM or STFM instruction.
M=0	IRQMEQO	I4LVL3	Set by lvl] controller to indicate instruction in lvl3 has M-FIELD=0.
M1 HAZ	NONE	I4PIPTOP	Signal indicating the comparison of M1 with R2, AR, or an address in any of the register stacks.
M1=0	IIM1EQO	I4PIPTOP	Signal indicating the M-FIELD (IIQIR(16-19)) at lvl1 is zero.
M2 HAZ	IPARVM2	I4RHAZ(0)	Signal indicating the comparison os M2 and AR.
M2=0	IPM2EO	I4PIPTOP	Signal in lvl2 controller indicating that the M-FIELD (IPQM2(0-3)) is zero.

Table B-15. X4 IPU Glossary (Continued)

TERM	SIG	LOC	DESCR
NEAR RANGE HAZ(0-3)	IRNRIHAZ(0-3)	I4LVL3	Indicates comparison of P3 with ZP (IRP3EQZP(0-3)) or ZB(IRP3EQZB(0-3)) for pipe (0-3).
NEXT	ILNEXT	I4CMREQ	Signal from the lookahead controller used in conjunction with KRTAG to indicate whether KA or KB is to receive the next instruction octet. With instr true, next false indicates the buffer pointed to by KRTAG will be fetched. Next true indicates the buffer not pointed to by KRTAG will be fetched.
NEXT CLOCK IS LAST CLOCK OF ROM SEQ (0-3)	BMQNCLCK(0-3)	MBU(0-3)	AU ROM bit which indicates that the second to last μ-sequence of an instruction is at the lvl6 ROM output register, i.e., the next μ-sequence will be the last μ-sequence for the instruction.
NEXT CLOCK IS SAME GROUP TIME IN ROM SEQ.(0-3)	BMQNCSGT(0-3)	MBU(0-3)	AU ROM bit indicating the next μ-sequence is the same group time in the AU ROM sequence.
NEXT ROM CODE NOP (0-3)	BMQNCNOP(0-3)	MBU(0-3)	AU ROM bit which indicates that the last μ-sequence for an instruction is at the lvl6 ROM output register, i.e., the next μ-sequence will be the NOP (or IDLE) seqence.
NIFRZ	IRNIFRZ	I4LVL3	Signal from lvl3 controller used to inactivate the lvl0, lvl1, and lvl2 controllers during the execution of a status command. It also tells the unit hardcore that a new instruction is at lvl3 and lvlS4-12 are empty.
NOP INSTR. AT LVL3	IRQDCNOP	I4INFACE (0)	Flag decoded from instruction op code indicating a NOP at lvl3 (i.e., first hex of op code=0).

Table B-15. X4 IPU Glossary (Continued)

TERM	SIG	LOC	DESCR
OABSY	ICOABSY	I4HDCORE	Signal indicating that the AR and RA flipflops are in opposite states, i.e., that a memory request is in progress and the OA register cannot be changed.
ONE CLOCK AT LVL4 (0-3)	IBQOCKL4(0-3)	I4INFACE (0-3)	Lvl4 flag indicating a one clock instruction is at lvl4 going to pipe(0-3).
ONE CLOCK AT LVL5 (0-3	IVQOCKL5(0-3)	I4INFACE (0-3)	This flag is set when a one clock instruction goes to lvl5, but it is not reset when the instruction goes to lvl6 unless a non-one clock follows into lvl5. Thus, when MBIACT(0-3) is true, this flag indicates a one clock instruction at lvl5.
ONE CLOCK AT LVL6 (0-3)	IVQOCKL6(0-3)	I4ZHAZ (1,5,9, 13)	Register stack bit indicating a one clock instruction at lvl6 of pipe(0-3).
ONE CLOCK AT LVL7 (0-3)	IVQOCKL7(0-3)	I4ZHAZ (1,5,9, 13)	Register stack bit indicating a one clock at lvl7.
ONLY ONE AE HAZ	IRONEAEH	I4ROUTE1	Signal in lvl3 controller indicating only one AE HAZ(0-3) is true.
OP DONE	IRQOPDN	I4LVL3	This flag is used by the lvl3 controller to indicate when the operand load portion of the instruction has been finished.
ORANGE INSTR. AT LVL3	IRQDCORG	I4INFACE (1)	Flag decoded from instruction op code indicating an orange (LF, LFM, STF, STFM) instrution at lvl3.
ORANGE LOAD AT LVL22	IPQDCLF	I4INFACE (0)	Lvl2 flag decoded from the instruction op code indicating that a LF or LFM instruction is at lvl2.
ORANGE LOAD AT LVL3	IRQDCLF	I4INFACE (0)	Flag decoded from the instruction op code indicating an LF or LFM instruction at lvl3.
ORDER,SELECT, REPLACE,MAP	IRBADGUY	I4MISC	Signal decoded from VPF op code lines indicating a vector bad guy.

Table B-15. X4 IPU Glossary (Continued)

TERM	SIG	LOC	DESCR
PACAUI(0-3)	IVPACAUI(0-3)	I4INFACE (0-3)	Signal indicating that an instruction at lvls 8-11 can advance and that an instruction at lvl7 can go to lvl8, 9, 10 or 11.
PACAUO(0-3)	IVQPACAO(0-3)	I4INFACE (0-3)	This flag indicates when an instruction can be moved into lvl12.
PACAUR(0-3)	IVPACAUR(0-3)	I4INFACE (0-3)	Signal indicating that an instruction can move from lvl6 to lvl7.
PACMBI(0-3)	IVPACMBI(0-3)	I4INFACE (0-3)	Signal indicating that an instruction can be moved into lvl5 of pipe(0-3).
PACMBO(0-3)	IVPACMBO(0-3)	I4INFACE (0-3)	Signal which indicates when an instruction at lvl5 can move to lvl6 based on the movement of any instructions in levels 6-12. This is not the usual definition, i.e.: PACMBO(0-3)= DPMBO(0-3)*PACAUR(0-3)+¬DPMBO (0-3)*GATAUI+¬AUIACT(0-3).
PAC2	IPPAC2	I4PIPTOP	Signal from lvl2 controller indicating that new data can move into lvl2.
PAC3	IRPAC3	I4ROUTE1	Signal from lvl3 controller indicating new data can move into lvl3.
PAC4(0-3)	IBPAC4(0-3)	I4INFACE (0-3)	Signal from lvl4 controller(0-3) indicating that it can accept an instruction bound for pipe (0-3).
PAC4(PIRT)	NONE	I4ROUTE2	This is the normal PAC4 signal with the array number (0-3) being determined by which of the past instruction route flipflops is set. See PIRT (0-3).
PAENAB	ILPAEN	I4PIPTOP	Signal from lvl0 controller indicating that a new instruction can be put into IR.
PA=BA	ILPAEBA	I4CMREQ	Signal indicating PA=BA at the octet level.
PA=7	ILPAEQ7	I4CMREQ	Signal indicating the 3 LSB's of PA (ILQPA(29-31)) are = 7.

Table B-15. X4 IPU Glossary (Continued)

TERM	SIG	LOC	DESCR
PA+LC ≤ 11	ILLTEQ11	I4CMREQ	Signal indicating that PBACT or LLAACT is true and the 3 LSB's of PA(ILQPA(29-31)) plus the loop counter (ILQL6(0-7)) plus the vacancy flipflops (ILQVAC (0-1)) is less than 11.
PB	IRQDCPB	I4INFACE (1)	Flag decoded from the instruction op code indicating a prepare to branch instruction.
PBACT+LLAACT	ILQPBVLLA	I4CMREQ	Flag indicating a PB or an LLA is in progress.
PB TARGET AT LVLO	ILPBTGTLO	I4CMREQ	Signal indicating that a PB instruction is at lvl2 or lvl3 and its target is the next instruction into IR.
PBXFER	IRPBXFER	I4LVL3	Signal from lvl3 controller indicating a PB instruction at lvl3 and PBENAB is true.
PINK INSTR. AT LVL3	IRQRMPNK	I4INFACE (1)	ROM bit indicating at pink (skips) instruction at lvl3.
PIPCLR(0-3)	IRPIPCLR(0-3)	I4ROUTE2	Levels 4-C not active.
PIRT(0-3)	IRQPIRT(0-3)	I4ROUTE3	These flipflops are used to indicate which of the pipes received the last instruction. The array number of the flipflop set indicates the pipe number.
PPO	IRPPO	I4LVL3	Signal from lvl3 controller to gate the stack pointer into BR for a PSH or PUL instruction.
PP1	IRPP1	I4LVL3	Signal from lvl3 controller indicating that the PSH or PUL instruction at lvl3 is going to execute its third pass. This signal gates the pointer into AR.
PRV(OP)	ICPRVO	I4CMREQ	Signal indicating that the protect violation bit pointed is true. PRV(OP) and KCM full cannot both be true.
PUSH INSTR.	IRQDCPSH	I4INFACE (2)	Flag decoded from instruction op code.
PUSH, PULL AT LVL2	IPQDCPP	I4INFACE (2)	Lvl2 flag decoded from instruction op code indicating that a PUSH or PULL instruction is at lvl2.

Table B-15. X4 IPU Glossary (Continued)

TERM	SIG	LOC	DESCR
PUSH OR PULL INSTR. AT LVL3	IRQDCPP	I4INFACE (2)	Flag decoded from instruction op code indicating PSH or PUL instruction.
PUSH, PULL, OR MODIFY 2ND PASS (0-3)	IVQPPMF(0-3)	I4INFACE (0-3)	Flag set on the lvl3→lvl4 transfer by the lvl3 controller to indicate that the second pass of a PUSH, PULL, or MODIFY instruction is in progress. It is reset when the data get to lvl12.
PO INDICATOR	IRQPOIND	I4LVL3	This flag is BMQAUPO (0-3) delayed by one clock. It indicates the stack pointer is in the AU output and ready to be gated into BR.
PO SIGNAL	BMQAUPO(0-3)	MBU(0-3)	This is an AU ROM bit indicating when the terminate stack signal from the AU can be sampled. This signal is latched in the PO IND. flipflop (IRQPOIND)
P3<4	ILP3(29)	I4CMREQ	Signal indicating the 3 LSB's of P3 are <4, i.e., bit 29 is false.
QIRT(0-3)	IBQIRT(0-3)	I4INFACE (0-3)	Flag at lvl4 indicating the presence of a new instruction at lvl4. This flag is set for all instructions as they go to lvl4 and is reset one clock after PACMBI(0-3) is true.

Table B-15. X4 IPU Glossary (Continued)

TERM	SIG	LOC	DESCR
RC HAZ	IRLATRCH	I4ROUTE1	Signal in the lvl3 controller indicating the presence of at least one RHZ bit in the register stack (levels 4-12) pointed to by the RC route (0-3) flipflops.
RC ROUTE(0-3)	IRQRCRT(0-3)	I4STATUS	Flag which indicates that (0-3) was the last to receive an instruction that could modify the compare code register.
RDACK	ICRDACK	I4CMREQ	Signal indicating the CMR will accept a read request if one is made.
RDACK1	NONE	I4CMREQ	This signal is cimplemented where used, and it consists of the normal RDACK and no indirect REQ., SF REQ, or LF REQ.
RUT AT LVL12(0-3)	IVQRDTLC(0-3)	I4ZHAZ (3,7,11, 15)	Flag in register stack for pipe(0) indicating that an instruction with a register destination is at lvl12.
READ AT LVL3 GOING TO LVL4	IRNEWRD3	I4ROUTE3	Signal indicating a read at lvl3 going to lvl4.
READ PROTECT	ICRCMPV	I4HDCORE	Signal indicating a protect violation has occurred during a read request. This signal can be true only during the clock following the toggling of the RA flipflop.
RECOVER LVL2 HAZ	IPRL2HAZ	I4PIPTOP	Signal from lvl2 controller indicating a T2 or M2 hazard exists.
REG.DEST. AT LVL3	IRQC3RDT	I4INFACE (2)	ROM bit indicating that the instruction at lvl3 has a register destination.
REG.DEST. AT LVL7 (0-3)	IVQRDTL7(0-3)	I4ZHAZ (3,7,11, 15)	Register stack bit indicating an instruction with a register destination is at lvl7 in pipe(0-3).
REG.DEST. AT LVL11 (0-3)	IRQRDTL(0-3)	I4ZHAZ (3,7,11, 15)	Register stack bit indicating an instruction with a register destination is at lvl11 in pipe(0-3).

Table B-15. X4 IPU Glossary (Continued)

TERM	SIG	LOC	DESCR
REG.INHIBIT FLAG	IRQRIHIB	I4LVL3	Set by lvl3 controller to indicate the first level of indirect addressing is complete. Reset by PAC3.
REG.SOURCE AT LVL3	IRQRMRGS	I4INFACE (2)	ROM bit indicating that the instruction at lvl3 has a register source.
REQ.CTR.=0-7	IRRCTREQ(0-7)	I4VECLAS	Signal in lvl3 controller indicating that the request counter (IRQRCTR(0-2)) contains 0-7.
RHAZ(0-3)	IRR3HAZ(0-3)	I4RHAZ (0-3)	Signal indicating R3 compares with some register stack address (levels 4-12) in pipe(0-3).
RHZ BIT AT LVL12 (0-3)	IVRHZLC(0-3)	I4ZHAZ (2,6,10, 14)	Flag in the register stack for pipe(0-3) indicating an instruction is at lvl12 that could modify the result code register.
R-OCTET HAZ(0-3)	IRR3OHAZ(0-3)	I4RHAZ (0-3)	Signal indicating the 3 LSB's of R3 (IRQR3(5-7)) compare with some register stack address (levels 4-12) at the octet level in pipe(0-3).

RXAACT(0-3)	IRRXAACT(0-3)
	IRZTXACT(0-3)
		I4ROUTE3
		I4VECLAS
			Signals from lvl3 controller to reset XAACT(0-3).

RXAFUL(0-3)	IRRXAFUL(0-3)	I4ROUTE3	Signal from lvl3 controller to reset XAFUL(0-3).

RYAACT(0-3)	IRRXAACT(0-3)
	IRZTXACT(0-3)
		I4ROUTE3
		I4VECLAS
			Signals from lvl3 controller to reset YAACT(0-3).

RYAFUL(0-3)	IRRYAFUL(0-3)	I4ROUTE3	Signal from lvl3 controller to reset YAFUL(0-3).
R3=LD(0-3)	IRR3VSLD(0-3)	I4ZHAZ (1,5,9, 13)	Signal indicating R3=LD for pipe(0-3).
R3=0	IRR3EQO	I4ROUTE2	Signal in lvl3 controller indicating that the R-field(IRQR34-7)) and the ROM bits IRQC3RAO, IRQC3RA1 ARE=0, for the BAE and BXEC instructions this indicates a NOP.

Table B-15. X4 IPU Glossary (Continued)

TERM	SIG	LOC	DESCR
R3=7	IRR3EQ7	I4ROUTE1	Signal in the lvl3 controller indicating that R-field IRQR3(4-7)=0. For the BRC and BCC instructions this indicates an unconditional branch.
R3>1 AND NO LLA OR PB ACTIVE	ILPBEN	I4CMREQ	Signal from lookahead controller indicating that no PB or LLA is in progress and R3>1, i.e., a PBXFER can be sent if a PB is at lvl3.

Table B-15. X4 IPU Glossary (Continued)

TERM	SIG	LOC	DESCR
SAME GROUP(0-3)	IBSMGRP(0)	I4INFACE (0-3)	Signal from lvl4 indicating that the instruction at lvl3 is in the same group as the last instruction to enter pipe(0-3). The instruction group is determined from ROM-1 bits: GP1, GP2, GP3, GP4.
SCLK INSTR.	IRQDCSTC	I4INFACE (3)	Flag decoded from instruction op code indicating a store clock instruction.
SF REQ	IRSFREQ	I4VECLAS	Signal used to make a memory request during a STF or STFM instruction.
SHORT CIRCUIT AT LVL4(0-3)	IBQSCKT4(0-3)	I4INFACE (0-3)	Lvl4 flag indicating the instruction at lvl4 is short circuiting on an instruction in pipe(0-3).
SHORT CIRCUIT AT LVL5	IVQSCKT5(0-3)	I4INFACE (0-3)	Flag indicating that the instruction at lvl5 is going to short circuit on the previous instruction in pipe(0-3).
SHORT CIRCUIT AT LVL6(0-3)	IVQSCKT6(0-3)	I4INFACE (0-3)	Flag indicating an instruction at lvl6 in pipe(0-3) which is going to short circuit on the previous instruction in that pipe.
SKIP	IRSKIP	I4LVL3	Signal from lvl3 controller indicating a skip is to be taken.
SKIP TAKEN INDICATOR	IRQSKIND	I4LVL3	This flag indicates that the conditions in the AU necessary for a skip to take place have been met. It is essentially the AU signal AEQET:1(PIRT) or its complement delayed by one clock.
SLNXT(0-3)	IVQSLNXT(0-3)	I4INFACE (0-3)	Lvl4 flag which indicates when an instruction can move to lvl6 based on the types of instructions in levels 6-12 of pipe(0-3).

Table B-15. X4 IPU Glossary (Continued)

TERM	SIG	LOC	DESCR
SLNXT PACMBO(0-3)	IVSLPMBO(0-3)	I4INFACE (0-3)	Signal indicating that DPMBO (0-3) is not true or PACAUR (0-3) is true, i.e., this is the usual definition for PACMBO. Since SLNXT(0-3) is a flipflop, this signal was used instead of PACMBO(0-3) to avoid introducing a bubble into the pipe under certain conditions.
SP(0-3)	IRQSELPI(0-3)	I4VECLAS	These flipflops indicate which pipe, if any, has been selected by a vector instruction.
SPEC.ERR.	NONE	I4PIPTOP	Signal indicating: 1) a BCLE or BCG instruction with odd T-field; 2) an instruction with an odd R-field when the ROM-1 RSE bit is on; or 3) the ROM-1 IOP bit on (ILLEGAL OP).
SPEC.ERR AT LVL3	IRORSPEC	I4ROUTE1	Signal from lvl3 controller indicating STFM is destined for the register file or a LFM is coming from the register file.
SPS INSTR.	IRQDCSPS	I4INFACE (2)	Flag decoded from instruction op code indicating an SPS instruction.
STACK INSTR.AT LVL3	IRQDCSTK	I4INFACE (3)	Flag decoded from instruction op code indicating a stack (PSH, PUL, MOD) instruction at lvl3.
STACK INSTR. IN AU(PIRT)	IRSTKIAU	I4ROUTE1	Signal indicating that there is no activity in levels 4-6 of the pipe pointed to by PIRT(0-3) and so the first pass of the current stack instruction must be in the AU (levels 7-12).

START FORCED WRITE (0-3)	IRSTFWRT(0-3)
	IRSTRFRW(0-3)
	IRSFWIHZ(0-3)
		I4ROUTE2
		I4VECLAS
		I4LVL3
			Signals from the lvl3 controller to initiate a forced write in pipe(0-3).

STATUS FREEZE	IMSTAFRZ	I4HDCORE	Signal used by lvl3 controller to empty the pipes so that a status command can be executed.

Table B-15. X4 IPU Glossary (Continued)

TERM	SIG	LOC	DESCR
STORE NEGATIVE	IRQRMNSN	I4INFACE (3)	ROM bit indicating a store that is not a store negative.
STL AT LVL3	IRSTREG	I4ROUTE1	Signal from lvl3 controller indicating a store instruction at lvl3 going to the register file (a store "little").
SUBTYPE	SUB YEL	I4INFACE (0)	ROM bits indicating instruction sub colars.
	SUB OR	I4INFACE (0)	
	SUB BLUE	I4INFACE (0)	
	SUB GRN	I4INFACE (0)	
	SUB PINK	I4INFACE (0)	
SXAFUL(0-3)	IRSXAFUL(0-3)	I4ROUTE3	Signal from lvl3 controller to set XAFUL(0-3).
SYAFUL(0-3)	IRSYAFUL(0-3)	I4ROUTE3	Signal from lvl3 controller to set YAFUL(0-3).

Table B-15. X4 IPU Glossary (Continued)

TERM	SIG	LOC	DESCR
TARGET AT LVL1	IIQTARGT	I4PIPTOP	Flag indicating that the target of a PB or LLA instruction is at lvl1.
TARGET AT LVL2	IPQTARGT	I4PIPTOP	Flag indicating that the target of a PB or LLA instruction is at lvl2.
TARGET AT LVL3	IRQTARGT	I4LVL3	Carried with the instruction to lvl3 to indicate that the current instruction came from the target location of a PB or LLA instruction.
TARGET FAIL	IRTGTFL	I4LVL3	Signal indicating that a target branch has failed to branch or has been skipped or branched around.
TERMINAL IND. AT LVL1	IITIAT1	I4PIPTOP	Signal indicating that the indirect cell's indirect bit (IIQIR(4)) is not set.
TERMINATE(PIRT)	AOTRMSTK(0-3)	AU(0-3)	This signal indicates when a stack instruction should be terminated because the test pass fail. It is latched in the TRMIN.IND (IRQTRMIN). The array number is provided by the PIRT(0-3) flipflops.
TERMIN.IND.	IRQTRMIN	I4LVL3	This flag is the latched version of the AU signal AOTRMSTK(0-3). It indicates when a stack instruction should be terminated after the first pass.
TFAIL	ILQTFAIL	I4CMREQ	Flag used by the lookahead controller to remember that the octet containing the TARGET+1 instruction must be refetched after a TARGET FAIL.
T HAZ FREE	IPQDCTHF	I4INFACE (1)	Flag at lvl2 decoded from instruction op code indicating that the instruction at lvl2 cannot be indexed.
T IND HAZ	IITHAZS1	I4PIPTOP	Signal in lvl1 controller indicating a comparison between T1 and R2 or T1 and an address in any of the register stacks.

Table B-15. X4 IPU Glossary (Continued)

TERM	SIG	LOC	DESCR
TOGGLE	NONE	I4CMREQ	Signal from lookahead controller to complement the KRTAG flipflop.
T1 HAZ	NONE	I4PIPTOP	Signal indicating the comparison of T1 with $2, AR, or an address in any of the register stacks.
T1=0	IIT1EQ0	I4PIPTOP	Signal indicating the T-field (IIQT1(1-3)) is zero.
T2 HAZ	IPARVT2	I4RHAZ(0)	Signal indicating the comparison of T2 and AR.
T2=0	IPT2E0	I4PIPTOP	Signal in lvl2 controller indicating that the T-field (IPQT2(1-3)) is zero.

Table B-15. X4 IPU Glossary (Continued)

TERM	SIG	LOC	DESTR
VACANT(0-3)	IRPVAC(0-3)	I4ROUTE2	This signal indicates no activity in levels 5-12 of pipe(0-3) and no instruction in level 4 destined for that pipe, and that the pipe is on with no vector in progress. If only one pipe is on, this signal indicates only that no vector is in progress.
VACANT(PIRT)	IRPRPVAC	I4ROUTE2	This is the pipe [illegible] ~~VACANT~~(0-3) signal with the PIRT(0-3) flipflops providing the array number.
VBAD(SP)	IRQVBAD(0-3)	I4VECLAS	This flag indicates a vector bad guy in progress in pipe (0-3) with the array number being supplied by SP(0-3). See SP(0-3).
VECT.AT LVL2	IPQDCVCT	I4INFACE (2)	Lvl2 flag decoded from instruction op code indicating that a vector is at lvl2.
VECTL R(LSB)=0	IRQR3(7)	I4PIPE(7)	LSB of the R-field of a vector instruction, where=0 indicates the VPF must be loaded before proceeding.
VECTOR INSTR. AT LVL3	IRQDCVCT	I4INFACE (2)	Flag decoded from instruction op code indicating a vector (VECT) instruction at lvl3.
VECTOR,PUSH,PULL, OR ORANGE INSTR. AT LVL2	IPSBBLK	I4PIPTOP	Signal indicating that one of these instructions is at lvl2.
VIP(0-3)	IRQVIP(0-3)	I4VECLAS	Set by lvl3 vector controller to indicate when a vector is in progress in pipe(0-3). Reset by MBU with vector end signal.
VIP(SP)	IRQVIP(0-3)	I4VECLAS	This is the normal VIP(0-3) flag with the array number supplied by SP(0-3). See SP(0-3).
VI0, VI1, VI2, VI3	IRVI0,IRVI1 IRVI2,IRVI3	I4VECLAS	Timing signals from lvl3 controller during vector initialization used to control the gating of the VPF.

Table B-15. X4 IPU Glossary (Continued)

TERM	SIG	LOC	DESCR
WACK	ICWACK	I4CMREQ	Signal indicating the CMR will accept a write request if one is made.
XAACT(0-3)	IBQYAACT(0-3)	I4INFACE (0-3)	Flag indicating the X-buffer address in XA is valid for pipe(0-3).
XAFUL(0-3)	IBQZAFUL(0-3)	I4INFACE (0-3)	Flag indicating the X-buffer in MBU(0-3) pointed to by the address in XA has valid data.
XCH INSTR.	IRQDCXCH	I4INFACE (3)	This flag is decoded from the exchange instruction op code.
XEC AT LVL1	IISDXEC	I4HDCORE	Signal decoded from instruction op code indicating an execute instruction at lvl1.
XEC AT LVL2	IPQDCXEC	I4INFACE (3)	Lvl2 flag decoded from instruction op code indicating the instruction at lvl2 is an execute.
XEC FLAG	IRQXEC	I4LVL3	Flag set by execute instruction at lvl3 to steer the lvl3 controller while it is executing the target instruction.
XFSET(0-3)	IBXFSET(0-3)	I4INFACE (0-3)	Signal indicating that on the next clock, data from CM will be gated into the X-buffer for pipe(0-3).
XUP(0-3)	IBQXUP(0-3)	I4INFACE (0-3)	Lvl4 flag sent to MBU(0-3) to inhibit the setting of DPMBI (0-3) until an update can take place.

Table B-15. X4 IPU Glossary (Continued)

TERM	SIG	LOC	DESCR
YAACT(0-3)	IBQYAACT(0-3)	I4INFACE (0-3)	Flag indicating the Y-buffer address in YA is valid for pipe(0-3).
YAFUL(0-3)	IBQYAFUL(0-3)	I4INFACE (0-3)	Flag indicating the Y-buffer in MBU(0-3) pointed to by the address in YA has valid data.
YELLOW INSTRUCTION AT LVL3	IRQDCXEC	I4INFACE (3)	Flag decoded from instruction op code indicating a yellow (execute) instruction at lvl3.
YFSET(0-3)	IBYFSET(0-3)	I4INFACE (0-3)	Signal indicating that on the next clock data from CM will be gated into the Y-buffer for pipe(0-3).
YNEXT(0-3)	IRQYNEXT(0-3)	I4ROUTE3	When this flipflop is set, it indicates that the Y-buffer in MBU(G-3) is the next buffer to receive a read request.
YNEXT(PIRT)	NONE	I4ROUTE3	This is the normal YNEXT flag with the array number (0-3) being determined by which of the PIRT flipflops is set. See PIRT.
YUP(0-3)	IBQYUP(0-3)	I4INFACE (0-3)	Lvl4 flag sent to MBU(0-3) to inhibit the setting of DPMBI(0-3) until an update can take place.

Table B-15. X4 IPU Glossary (Continued)

TERM	SIG	LOC	DESCR
ZAGE0(0-3)	IRZAGED0(0-3)	I4MISC	Signal indicating that none of the 3 Z-AGE flipflops for pipe (0-3) is set, making this the youngest of the 4 Z-buffers. When all ZPFUL(0-3) are set, only one pipe can have this AGE.
ZAGE1(0-3)	IRZAGED1(0-3)	I4MISC	Signal indicating that 1 of the 3 Z-AGE flipflops for pipe(0-3) is set, making this the second youngest of the 4 Z-buffers. When all ZPFUL(0-3) are set, only one pipe can have this AGE.
ZAGE2(0-3)	IRZAGED2(0-3)	I4MISC	Signal indicating that 2 out of the 3 Z-AGE flipflops for pipe(0-3) are set, making this the second oldest of the 4 Z-buffers. When all ZPFUL (0-3) are set, only one pipe can have this AGE.
ZAGE3(0-3)	IRZAGED3(0-3)	I4MISC	Signal indicating that all 3 of the Z-AGE flipflops for pipe(0-3) are set, making this the oldest of the 4 Z-buffers. When all ZPFUL (0-3) are set, only one pipe can have this AGE.
BFUL(0-3) Z	IBQZBFUL(0-3)	I4INFACE (0-3)	Flag controlled by ZBFUL controller to indicate that the address in ZB is valid for pipe(0-3).
ZEX(0-3)	IBQZEX(0-3)	I4INFACE (0-3)	Lvl4 flag indicating that the Z-buffer and X-buffer in pipe (0-3) have the same address. It is used by the forced write controller to initiate an automatic Z→X update during a forced write.
ZEY(0-3)	IBQZEY(0-3)	I4INFACE (0-3)	Lvl4 flag indicating that the Z-buffer and Y-buffer in pipe(0-3) have the same address. It is used by the forced write controller to initiate an automatic Z→Y update during a forced write.

Table B-15. X4 IPU Glossary (Continued)

TERM	SIG	LOC	DESCR
Z-JOIN(0-3)	IBQZJOIN(0-3)	I4MISC	Flag indicating that at least one store was made into the Z-buffer of pipe(0-3) while in the joined mode. The flag is set IF FORK IND.=0 when ZPFUL(0-3) is set, and reset when pipe(0-3) is force written.
ZPFUL(0-3)	IBQZPFUL(0-3)	I4INFACE (0-3)	Flag controlled by ZPFUL controller to indicate the address in ZP is valid for pipe(0-3).
ZPFUL(SP)	NONE	I4VECLAS	This is the normal ZPFUL flag with the array number determined by the SP(0-3) flipflops. · See SP(0-3).
ZPTZB(0-3)	IPZPTZB(0-3)	I4INFACE (0-3)	Signal from forced write controller used to gate ZP→ZB and set ZBFUL for pipe(0-3).
Z-STORE AT LVL6(0-3)	IVQZSTL6(0-3)	I4ZHAZ (2,6,10, 14)	Register stack bit indicating a Z-store at lvl6 of pipe(0-3).
Z-STORE AT LVL7(0-3)	IVQZSTL7(0-3)	I4ZHAZ (2,6,10 14)	Register stack bit indicating a Z-store at lvl7 of pipe(0-3).
Z-STORE AT LVL10 (0-3)	IVQZSTLA(0-3)	I4ZHAZ (2,6,10, 14)	Register stack bit indicating a Z-store at lvl10 of pipe(0-3). This can happen only during the second pass of a PUSH, PULL, or MODIFY instruction.
ZSTP(0-3)	IRZST(0-3)	I4ZHAZ (2,6,10, 14)	Indicates at least one Z-store in levels 5-11 of pipe(0-3) or a Z-store at lvl4 destined for pipe(0-3) generated on I4RHAZ.
ZTXU(0-3)	IBQZTXU(0-3)	I4INFACE (0-3)	Lvl4 flag sent to MBU(0-3) to cause a Z→X update on the next clock.
ZTYU(0-3)	IBQZTYU(0-3)	I4INFACE (0-3)	Lvl4 flag sent to MBU(0-3) to cause a Z→Y update on the next clock.

APPENDIX C
4XCP HAZARD CONDITIONS

APPENDIX C
4XCP HAZARD CONDITIONS

This appendix lists the types of hazard conditions that occur in the times-four CP. These hazards are divided into three main categories: (1) those involving only scalar instructions, (2) those in which both scalars and vectors are in progress at the same time, and (3) those involving only vectors. These categories are further subdivided in the outline on the following page.

I. Scalar Hazards

A. Register Hazards

1. Register Operand Hazards

2. Alpha Register Operand Hazards

3. Index or Base Hazards

4.* Destination Hazards

5.* Largest Word Size Hazard

B. Central Memory Address Hazards

1. Store-Read Hazards

2. Store-Load File Hazards

3. Store-Store File Hazards

4. Store-File and Load-File R-Octet Hazards

5. Store File Hazard

6. Store File-Read Hazard

C. Instruction Hazards

1. Storing Over Instructions

2. Store File Over Instructions

3. Vectors Storing Over Instructions

D. Arithmetic Exception Hazards

1. Load Arithmetic Exception Mask or Condition Hazards

2. Store Program Status Hazards

E. Branch Hazards

1. Result Code Hazard

2. Condition Code Hazard

3. Arithmetic Exception Branch Hazard

*New 2X, 3X, and 4X Hazards

II. Scalar-Vector Hazards

   A. VPF Modification

   B. Scalars Writing Over Vector Input Arrays

   C. Vectors Writing Over Scalar Read Data

   D. Alpha Hazards During Vectors in Fork and Join Mode

III. Vector Hazards

   A Vectors Storing Over Their Own Input Data Arrays

   B. Addressing Conflicts Between Two Vectors Executing Simultaneously in Parallel Pipes

   C. Halfword Z-fill-in Hazard

Throughout this description, operand and instruction conflicts are referred to as hazards. The intended meaning of "hazard" as used here should be "something to avoid if possible." In most cases not staying away from hazards will result only in a slower execution rate due to instruction delays while waiting for the hazard to clear, but the hazard will not affect the logical results of a program. In other cases, such as those involving the "vector hazard rule," a hazard is quite critical and will result in numerical program errors or loss of program control if the hazard rule is not taken into account. The difference in terminology between these two uses of the word "hazard" is distinguishable in this description by a block to the right of each hazard heading which indicates the following:

DELAY for delay-producing hazards which do not affect program results.

FUNCTIONAL for error-producing hazards.

I. Scalar Hazards

A. Register Hazards

1. Register Operand Hazards

DELAY

Two conditions are necessary for a register operand hazard to exist. These conditions are:

(a) A first instruction of an instruction stream having a register destination aimed at one of the registers of the register file. This instruction is located below level 3 of the CP pipeline but has not yet entered its result into the register file.

(b) A second instruction, occurring at a later point in the instruction stream, having arrived at level 3 of the CP pipeline, and having a register source requirement to read a portion or all of the register presently destined to be modified by the first instruction.

<u>Hardware solution:</u> Logic is provided to detect this hazard. Register operand hazards are cleared when the first instruction has modified its destination register.

<u>Delay avoidance</u>: It is possible to avoid some of the delay due to register operand hazards. Two methods exist.

(a) Using the short-circuit path -

Two short-circuit data paths exist from the output to the input of the arithmetic unit. One is for Register Short Circuits, and the other is for Alpha Register Short Circuits.

The Register Short-Circuit path is used when a second instruction experiencing a register hazard determines that a first instruction causing the hazard is still the last instruction to have entered a given pipe. Other instructions may have occurred between the "first" and "second" instructions, but none of these have taken the same pipe as the first instruction.

For this short circuit to take place, the word sizes of the source and destination registers of the two instructions must be the same. An exception to this rule is that halfword-to-halfword short circuits are not provided. In addition to the same word size short circuits, there also exists short circuits for doubleword results feeding back to second instruction singleword register sources with even register addresses.

(b) Instruction insertion -

In some cases it may be possible to insert other instructions between the twc that cause a register operand hazard. The hazard is not actually avoided by this method; it is just postponed to the point where it does not exist any more. The method does, however, allow the CP to perform other useful computations during time that the CP would normally be waiting for the hazard to clear. Of course, the other work performed could not use the register which is causing the hazard.

Delay time: Register operand hazard delay is dependent upon the pipe time of instruction I1 which is 4 clocks + AU time + memory time, providing the AU output does not become blocked due to multiple AU outputs destined for the register file. AU time is found listed in Table 1 of the CP timing section (B2) of the CP Hardware Volume. Memory time is eight clocks if instruction I1 makes a memory read request. This time delay equation assumes an initially empty pipe. For accurate timing estimates, all pipe conditions prior to instruction I1 must be taken into account.

Example of register operand hazard:

Inst #	Inst R, α	Pipe	Comment
I1	LOAD A1, CM1	(0)	
I2	LOAD A2, CM2	(0)	Assume AR=ZP(0)
I3	ADD A1, CM3		

For this hazard to appear in the 4X CP, operation of the AU short-circuit path must be blocked by one or more other instructions using a different register between the two instructions that cause the hazard. The other instruction(s) must use the same pipe in order to block the short-circuit path. For purposes of this example, it is assumed that a prior store instruction has left the Z-buffer of pipe 0 with data destined for octet CM2 in central memory. This condition forces instruction I2 down pipe 0, thereby blocking I3 from picking up its register operand (A1) over the AU register short-circuit path from the prior result of instruction I1. Without the short circuit, I3 must wait in level 3 until A1 has been modified by instruction I1.

2. Alpha Register Operand Hazards

DELAY

Two conditions are necessary for an alpha register operand hazard to exist. These conditions are:

(a) A first instruction of an instruction stream having a register destination aimed at one of the registers of the register file. This instruction is located in the CP pipeline between levels 4 through 12 inclusive.

(b) A second instruction, occurring at a later point in the instruction stream, having arrived at level 3 of the CP pipeline and having an effective address $\alpha \leq 2F$ and an M-field of zero, such that an alpha register source requirement exists to read a portion or all all of the register presently destined to be modified by the first instruction.

Hardware solution: Logic is provided to detect this hazard. Alpha register operand hazards are cleared when the first instruction has modified its destination register.

Delay avoidance: The same two methods that were used to avoid delays due to register operand hazards are also useful for avoiding delays due to alpha register operand hazards.

Delay time: Alpha register hazard delay is dependent upon the pipe time of instruction I1 which is 4 clocks + AU time + memory time, providing the AU output does not become blocked due to multiple AU outputs destined for the register file. AU time is from Table 1, and memory time is eight clocks if instruction I1 makes a memory read request.

Inst #	Inst R, α	Pipe	Comment
I1	LOAD A1, CM1	(0)	
I2	LOAD A2, CM2	(0)	Assume AR=ZP(0)
I3	ADD A3, A1	(0)	

In this example instruction I2 takes pipe 0 because its central memory read data, CM2, is assumed to be resident in the Z-buffer of pipe 0. I2 blocks the possibility of instruction I3 picking up its alpha register operand (A1) via the AU alpha register short-circuit path from the output of instruction I1. Without the short circuit, I3 must wait in level 3 until A1 has been modified by I1.

3. Index or Base Hazards

DELAY

Two conditions necessary for an index or base hazard are:

(a) A first instruction of an instruction stream having a register destination aimed at one of the index or base registers of the register file. This instruction is located in the CP pipeline between levels 2 through 12 inclusive.

(b) A second instruction, occurring at a later point in the instruction stream, having the requirement to use the index or base register presently destined to be modified by the first instruction.

Hardware solution: Logic is provided to detect this hazard. Index or base hazards are cleared when the first instruction has modified its destination register. Index or base hazards hold the second instruction at level 1 until the hazard is cleared. However, late index or base hazards occur when a Store instruction into a base or index register immediately precedes a second instruction requiring the index or base register at level 2. For late hazards of this type, the second instruction will refetch its index or base register at level 2 when the hazard has cleared.

Delay avoidance: There is no hardware short circuit to decrease the delay due to index or base hazards. The method of instruction insertion can be used to perform other work while waiting for the index or base hazard to clear.

Delay time: Index or base hazard delay is dependent upon the pipe time of instruction I1 which is 6 clocks + AU time + memory time, providing the AU output does not become blocked due to multiple AU outputs destined for the register file. AU time is from Table 1, and memory time is eight clocks if instruction I1 makes a memory read request.

Example of Index hazard:

Instruction #	Inst R, α
I1	LOAD X1, CM1
I2	ADD A1, CM2, X1

In this example, instruction I1 modified index register X1; then the next instruction, I2, uses index register X1 to develop its effective address, CM2 + (X1). The second instruction must wait in level 1 until instruction I1 has modified index register X1.

4. Destination Hazards

DELAY

Destination hazards exist in a 2X, 3X, or 4X CP, but not in a 1X CP. This hazard is due to a Load, Interpret, or Normalize instruction having the same register destination address as that of some prior instruction which has not yet entered its result into the register file. The prior instruction is any type that has a register destination.

If this hazard were not detected by the 2X, 3X, and 4X machines, then it would be possible for a second instruction to overtake a first instruction, via another MBU-AU pair, and place its result in the register file prior to the result of the first instruction. Results occurring out of sequence in this manner would not leave the latest value in the register file at the common register address of the two instructions. Hardware protects against this type of hazard.

Hardware solution: Logic is enabled in a 2X, 3X, or 4X configuration for detecting destination hazards. This logic is disabled in a 1X configuration, since one instruction cannot pass another instruction in a one-pipe machine. Also, hardware is provided in the 2X, 3X, and 4X machines to force Load type instructions down the pipe in which the destination hazard exists for cases when the effective address is selecting a register of the register file ($\alpha \leq$ 2F and M = 0). Forcing the instruction into the pipe behind the destination hazard has the effect of preventing the instruction race condition, while at the same time allowing the Load type instruction to proceed without experiencing a destination hazard delay.

For the case when the effective address of a Load type instruction is selecting a central memory operand, a destination hazard causes further checking by hardware to determine whether the Load address (in register AR of level 3) is equal to the Z-octet address contained in pipes other than the one causing the destination hazard. If there is address agreement, then the destination hazard causes a delay until the hazard clears. This delay prevents a forced write of Z in the other pipe and a read request for the Load address from the destination hazard pipe. If there is no address agreement, then the Load takes the pipe causing the destination hazard, and no delay occurs.

Delay avoidance: Delays due to destination hazards are avoided by hardware for the cases when: the Load address is selecting a register of the register file; or when the Load address is selecting central memory and the address is also resident in the Z-buffer of the destination pipe; or when the Load address is not resident in the Z-buffer of any of the pipes.

The method of instruction insertion can be used to fill in the dead time waiting for a destination hazard to clear.

Delay time: A destination hazard clears when the register destination instruction causing the hazard has placed its result in the register file. For meaningful code, the register destination instruction (I1) should be followed by a Store, then by the Load type instruction (I3) experiencing the delay. Otherwise, I3 will simply write over data entered into the destination register by I1. If, for example, I1 were an ADD, then the result of the addition would be lost because of the Load into the same register.

With the Store instruction between I1 and I3, the destination hazard delay due to I1 is 3 clocks + AU time + memory time, providing the AU output does not become blocked due to multiple AU outputs destined for the register file.

Example of destination hazard is as follows:

Instruction #	Inst R, α
I1	DIVIDE A1, CM1
I2	LOAD A1, CM2

In this example, the divide instruction could be replaced by any instruction with a register destination of A1. The Load does not see a source register operand hazard because its source operand is from central memory. It would be all right for the Load to take a different pipe if it were not possible for the Load to overtake and pass the divide. To guard against this possibility, the Load instruction is routed down the same pipe taken by the divide. The divide pipe is taken assuming that CM2 is not resident in the Z-buffer of any other pipe. If residency exists, then the Load waits at level 3 until the destination hazard due to the divide has cleared.

Note that this example shows meaningless code, since the result of the division is destroyed by the Load.

5. Largest Word Size Hazard

DELAY

Four conditions must exist for a largest word size hazard to occur. These conditions are:

(a) Only two types of second instructions can cause this hazard. They are Multiply instructions (Op codes 6C and 7C) with an even addressed arithmetic register specification and fixed to floating point conversion instructions (Op codes AA, A9, and AB; mnemonic codes FXFD, FHFL, and FHFD, respectively). These two types of instructions are the only ones which use a larger register destination word size than their own source word size.

(b) The second instruction must be preceded by a first instruction with a register destination of smaller word size than that of the second instruction's register destination.

(c) The register destination of the first instruction must be contained within the register modification space of the destination register of the second instruction.

(d) The register destination of the first instruction does not have the same address as the source register of the second instruction. Otherwise, if the addresses were the same, then the hazard would be classified as a register operand hazard.

Hardware solution: Hardware detects largest word size hazards and holds the second instruction in level 3 until the hazard clears.

Delay avoidance: The method of instruction insertion can be used to fill in the dead time waiting for a largest word size hazard to clear.

Delay time: Largest word size hazard delay time is dependent upon the time for instruction I1 to clear the pipe. This time is 4 clocks + AU time + memory time, providing the AU output does not become blocked due to multiple AU outputs destined for the register file. AU time is from Table 1, memory time is eight clocks if instruction I1 makes a memory read request.

Example of largest word size hazard:

An example of an instruction sequence with a largest word size hazard is given following in which a Load to an arithmetic register is followed by a multiply using an adjacent, evenly addressed arithmetic register.

Instruction #	Inst R, α
I1	LOAD A1, CM1
I2	MULTIPLY A0, CM2

The source register of the multiply does not show up as a register operand hazard with the destination register of the Load instruction. The multiply instruction, using even address register A0, will place its result into the even-odd register address pair A0-A1. If, for example, the read octet of the multiply was resident in one of the four pipes and the read octet of the Load was not resident, then it would be possible for the multiply to place its doubleword result into registers A0 and A1 prior to A1 being entered by the Load instruction. If A1 were entered late by the Load, then the expected final doubleword product of the multiply would be overwritten in its least significant half. Largest word size hazard logic in the CP does not allow this to happen.

Another example is presented to show that the hardware does not call out a largest word size hazard unnecessarily. In this example, an evenly addressed register (A0) is used by the Load, while an odd register (A1) is used by the multiply. This is just opposite to the register usage of the previous example.

Instruction #	Inst R, α
I1	LOAD A0, CM1
I2	MULTIPLY A1, CM2

The multiply instruction does not have a source register operand hazard or a largest word size hazard. The multiply selects its pipe according to the scalar routing rules. There is no hazard delay for this instruction sequence.

B. Central Memory Address Hazards

1. Store-Read Hazards

DELAY

A Store instruction followed by a read from the same octet of memory causes store-read hazards. Several store-read sequences are considered in order to see how timing between instructions influences whether this type of hazard appears or not.

Example B.1.1

Instruction	Inst R, α	Pipe
I1	STORE A1, CM1	(0)
I2	ADD A2, CM1	(X) $\begin{cases} X=0 \text{ if no SC} \\ X\neq 0 \text{ if SC} \end{cases}$

For this example, instruction I2 at level 3 determines that there is address agreement between address register AR and the Z-pipe register, ZP(p), for one of the pipes (p). The ZP registers cover Store instructions from levels 4 through 12 and the Z-buffer. Another set of four registers, ZB(p), holds the Store address of data being written into memory.

At this point in the determination of pipe selection, it is possible for instruction I2 to have a register operand hazard and find that it can pick up its register operand (A2) by using the AU short-circuit path. If an AU short circuit is found, then instruction I2 will initially try to take a pipe other than 0 because pipe 0 contains a last destination register address of A1. The register source address of an instruction is placed in the last destination register for Store instructions if they do not modify data when passing through the arithmetic unit. Store Negative is an example of a Store type instruction that modifies its register source data in the AU.

If the Add initially tries to take a different pipe because of both a register hazard and a short-circuit condition, then a forced write to central memory will be started for octet CM1 in pipe 0. This forced write occurs so that the data from location CM1 can be read by another pipe. The only data path from the output of one pipe to the input of another pipe is by means of central memory. Before the read request can be issued by another pipe, the complete write cycle for that memory location must have finished. This is because the four pipes are connected to memory over separate memory ports, and the Memory Control Unit (MCU) does not guarantee that a write to memory will be processed first for the case of a write then a read to the same octet on two different ports.

The timing delay for both a write and a read cycle is shown as case 1 in the store-read timing diagram of Figure 1. The ADD is completed on clock 30 for case 1.

It is highly likely that the register will clear before the forced write is complete; in which case, the Add instruction may choose any other pipe for its read from location CM1. This change in routing does not affect the delay time if the forced write has been initiated. However, if the register hazard clears before the forced write is initiated, then the read from location CM1 can be done from pipe 0. If octet CM1 is not resident in the X- or Y-buffers of pipe 0, then a memory read request is required before the Z-to-X update can occur. This is shown as case 2 of Figure 1, in which the Add is completed on clock 17. This is thirteen clocks earlier than the time for completion in case 1.

Case 3 occurs when the store immediately precedes the add, there is no short-circuit condition to cause the Add to select another pipe; and the X- or Y-buffer has a resident octet containing location CM1. The resident X or Y is the result of a prior memory read request for data within the octet containing word CM1. The Add is allowed to move into level 6 when there are no Z-stores in the pipe and the update from Z to X has taken place. The Add is completed on clock 12, some eighteen clocks earlier than case 1.

Case 4 is one in which there are no Z-stores between levels 4 through 12 of pipe 0, but pipe 0 contains a resident Z-octet of address CM1. This implies that the Store into CM1 occurred prior to the arrival of the Add at level 3 and that there were other instructions between the Store and the Add which allowed time for the Store to pass through the pipeline into the Z-buffer. Also, there is assumed to be a resident X as a result of a prior memory read request. The Add is completed on clock 9 for this case. This is twenty-one clocks earlier than case 1.

The Store-read example used for the purpose of this description is functionally the same as

```
    I1        STORE         A1, CM1
    I2        ADD           A2, A1
```

Using this code, the Add takes the same pipe as the Store because of the alpha short circuit on register A1. There is no long write then read delay as previously described. Therefore, it is apparent from what has gone before that code should be written as in the preceding whenever possible in place of the Store-add example given earlier.

CASE 1. STORE INTO CM1, THEN ADD WITH SHORT CIRCUIT TO ANOTHER PIPE, NONRESIDENT X.

CASE 2. STORE INTO CM1, THEN ADD WITH UPDATE FROM Z TO X, NONRESIDENT X.

CASE 3. STORE INTO CM1, THEN ADD WITH UPDATE FROM Z TO X, RESIDENT X.

CASE 4. RESIDENT Z OF OCTET CM1, THEN ADD WITH UPDATE FROM Z TO X, RESIDENT X.

TIME

LEVEL	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
3	S	A	→	→	→	→	→	→	→	→	→	→	→	→	→	→															
4		S	A														A														
5			S	A	→	→	→	→	→	→	→	→	→					A	→	→	→	→	→	→	→	→					
6				S		A			A					A													A				
7					S		A			A					A													A			
8								A			A					A													A		
12						S			A			A					A													A	
Z							S																								
RF										A			A					A													A
										CASE 4			CASE 3					CASE 2													CASE 1

STORE-READ TIMING

2. Store-Load File Hazards

DELAY

This hazard is caused by CP code of the form :

Instruction #	Inst R, α
I1	STORE A1, CM1
I2	LOAD FILE R, CM1

Store-Load File hazards are the result of a Load File instruction requesting the same octet from memory as was written into by a previous Store instruction. The Store could have been the immediately preceding inst. or one which occurred some time ago but which left its Z-octet resident in the pipe that was taken; i.e., no other Stores to different octets have taken that pipe since the Store to CM1.

Since the Store octet is resident in one of the pipes, its address, CM1, is in some ZP (p) register. The Load-File address, CM1, is compared against all ZP addresses when the Load File reaches level 3 of the IPU. Address agreement from this comparison is called an alpha hazard.

At this point the LF instruction at level 3 finds the pipe (p) in which the resident Z-octet of address CM1 is located. A check is made to see if the hazard is due to a vector in progress rather than a scalar Store to memory. If the hazard is due to a vector in progress, then the alpha comparison is ignored; and the Load File proceeds to execution. The alpha comparison is ignored in this case because compiled code will not contain vectors that write over scalar data areas when the CP is placed in the FORK mode. If the CP is placed in the JOIN mode, then it is not possible for subsequent scalar instructions to proceed to execution until the vector has completed.

If a vector is not in progress in the pipe (p) in which the alpha hazard exists, then the LF instruction in level 3 waits until there are no Z-stores in pipe p. This wait insures that the Z-octet buffer has all its data before a "start forced write" command is issued to the MBU.

The LF instruction continues to wait at level 3 until the storage of octet CM1 is completed to central memory. It then proceeds to execution at level 3, making its load file request via the IPU4 memory port.

Hardware solution: Logic is provided to detect this hazard. Store-Load File hazards are cleared when the Z-octet containing the Load-File data has been written into memory. The Load File is held in level 3 until the hazard is cleared; then it is executed at level 3. A Load-File instruction does not use the CP pipeline below level 3.

Delay avoidance: Delay cannot be avoided for adjacent Store-Load File instructions. The method of instruction insertion can be used to make the instructions nonadjacent. If this method is used, then a second Store to a different octet should be forced down the pipe containing CM1 by means of a short circuit to the register address of the Store. However, be careful selecting a different octet for storage because, if the different octet happens to be resident in some other pipe, then the Store will take that other resident pipe. Also, the X- and Y-buffers must not contain the address of the second Store (ST2) because then ST2 would have to wait on outstanding reads before the forced write of ST1 could be issued. Also, the second Store must not occur before the first Store has had time to pass through the pipe; otherwise, the second Store would have to wait in level 3; and that is what is to be avoided. The list of contingencies on the second Store would seem to preclude its use as an effective means to avoid a Store-Load File hazard; however, this method is given in hopes that it can be used to advantage.

Delay time: Store-Load File delay depends on the time for the first Store to pass through the pipe into Z, then for the Z-octet to be written into memory. This time is 5 clocks + memory write time if the Store immediately precedes the Load File. The five clocks are the pipe time of the Store and can be deleted if it is known that there are no Stores in the pipe; i.e., that the resident Z-octet has all its data. Also, this timing equation assumes that the AU output does not become blocked due to multiple AU outputs destined for the register file. This would be due to instructions prior to the Store. Memory write time in this equation is normally equal to eight clocks.

3. Store-Store File Hazards

DELAY

This hazard is essentially the same as the Store-Load File hazard just described. In fact, the same descriptions can be used for the hardware solution, delay avoidance, and delay time sections by replacing the words "Load File" with "Store File."

In order for this hazard to exist, the Store instruction's address must be contained within the octet address space of the Store File instruction. This is an unlikely condition for reasonable code, since data stored is immediately written over by the Store File instruction.

Prior to execution of the Store File, the Store instruction is forced out of the CP pipeline into memory so that the Store File is certain to place its data into memory after the store. This is done to preserve the order of instructions.

4. Store File and Load File R-Octet Hazards

DELAY

This hazard is common to both Load File and Store File instructions. The delay occurs when the octet selected by the Load File or Store File instruction at level 3 (as specified by the R-field) covers a register being modified by an instruction below level 3.

In the case of a Load File, the purpose of the delay is to insure that all registers have been modified for all instructions below level 3 that have register destinations targeted for the same octet as that specified by the Load File. Otherwise, if the delay were not applied, a late arriving register destination could modify a register within the octet entered by the Load File after the Load File was completed. The order of instructions would not be preserved if the Load File were executed without testing for an R-octet hazard.

For a Store File instruction, the purpose of the delay is to insure that the register file octet to be stored has been modified by all instructions below level 3 that have register destinations targeted for the same octet as that to be stored. This is so the Store File will have all its data to be stored.

5. Store File Hazard

DELAY

A Store File hazard is due to a memory read instruction followed shortly by a Store File to the same memory octet. The memory read is positioned in time such that it has requested memory but has not yet received its data from memory. With this condition true, the Store File cannot proceed to execution because, if it did, then it is possible for the Store File data to be written into memory prior to data being read from memory by the instruction using a memory operand. Since the read and write operations take place on different memory ports, the Memory Control Unit (MCU) cannot guarantee that a read arriving first will be processed first. For this reason, the Store File must wait until the memory data has been received.

Hardware solution: Hardware detects this hazard. Store File hazards are cleared when the X-or Y-buffer of the MBU, which had an outstanding request for an octet of memory of the same address as the Store File, has received its data. The Store File waits at level 3 until the hazard has cleared.

Delay avoidance: This delay can be avoided by separating memory reads to a given octet from Store Files to that same octet. The separation can be done by instruction insertion. Enough time must be allowed for the read data to be received before the Store File arrives at level 3 for execution.

Delay time: Store-File hazard delay depends on the memory time of the instruction causing the hazard. If the memory read instruction is immediately ahead of the Store File, then the Store-File wait time can be as high as ten clocks, assuming no memory interference.

Example: An example of CP code producing this hazard is as follows:

Instruction #	Inst R, α
I1	ADD A1, CM1
I2	STORE FILE R, CM1

6. Store File-Read Hazards

DELAY

This hazard is caused by CP code of the form:

Instruction #	Inst R, α
I1	STORE FILE R, CM1
I2	ADD A1, CM1

which is just the reverse of the code given in the previous example for a Store File hazard.

To be sure of getting the latest data, the Add must wait for the Store File to complete its write to memory before the Add can request memory. This hazard delay occurs as a normal part of a Store File instruction. That is, the second instruction is delayed regardless of its instruction type or memory address.

Hardware solution: This hazard is protected by the level 1 controller of the IPU. When a Store File instruction is detected at level 2, the level 1 controller goes into the "Store File state" and blocks any subsequent instructions from reaching level 2. Subsequent instructions are held at level 1 until the "write acknowledge" signal is received from the MCU and the "data available" signal has returned to "zero." These conditions indicate that the Store File data has been received by the MCU and that the process of writing data into the memory module has been initiated. At this point the instruction following the Store File is released from level 1.

Delay avoidance: This delay cannot be avoided; it is inherent in the operation of the Store File instruction.

Delay time: Store File read hazards cause a delay of eight clocks in the execution of the "read" instruction at level 3, assuming that there is no wait for "write acknowledge" or for "data available" due to memory interference. These eight clocks are to be considered as the time to execute the Store File and not as any additional delay caused by the fictitious Store File read hazard.

C. Instruction Hazards

1. Storing Over Instructions

DELAY

Scalar Store instructions that modify other instructions may cause an instruction hazard delay. In particular, when the address being stored into by a Store instruction (below level 3) is equal to the instruction address of an instruction at levels 1, 2, or 3 or is contained within the octet of instructions requested or resident in the KA or KB buffers, then an instruction hazard flag will be set.

Store instructions below level 3 include Stores resident in the Z-buffers of each of the four pipes. These Stores have not yet been written into memory. Stores in the pipeline or in the Z-buffers hold their storage addresses in the ZP registers. Stores being transferred to memory from the Z-buffers through the ZB-buffers hold their storage addresses in the ZB-registers during the transfer. The ZP- and ZB-registers are compared with the program counter values from level 3 upwards through the present address (PA) and look-ahead address (LA) registers. A true comparison indicates an instruction hazard.

Instruction hazard recovery takes place if the instruction so marked as having an instruction hazard reaches level 3 for execution. Level 3 is the point at which a flagged instruction hazard becomes a real instruction hazard. This distinction is made because it is possible for an instruction to be marked as having a potential instruction hazard but to never require the instruction hazard recovery process as a result of a Branch instruction taking the branch prior to the flagged instruction's arrival at level 3. In this case the flagged instruction is not executed; and, so, there is no need to recover the modified instruction.

The process of instruction hazard recovery involves performing a forced write operation on the pipe that contains the Store which caused the instruction hazard. Once the Store octet has been written into memory, the IPU may refetch the instruction address that was marked as having an instruction hazard.

Hardware solution: Instruction hazards are divided into two classes for the purposes of hardware implementation. These classes are: near-range hazards and far-range hazards. A near-range hazard occurs when there is a true comparison between the program counter at level 3 (P3) and the ZP- or ZB-registers of any of the four pipes. This condition causes a forced write and then an immediate instruction hazard recovery. It is not a potential but a real instruction hazard when it occurs at level 3.

A far-range hazard is when there is a true comparison between the following pairs of registers:

Look-ahead	LA vs. ZP for all pipes
Present address	PA vs. ZP 〃
Level 1	P1 vs. ZP 〃
Level 2	P2 vs. ZP 〃

These comparisons are flagged as potential hazards at the appropriate level of the pipeline. A far-range hazard may never reach level 3 if a branch is taken before the flagged instruction arrives at level 3. However, if it arrives at level 3, then a forced write is sent to the pipe containing the Store that caused the hazard. Instruction hazard recovery starts after the forced write is complete. Recovery is accomplished by fetching the modified instruction from memory.

<u>Delay avoidance</u>: The simplest and most effective way to avoid instruction hazards is to abide by the rule "never modify instructions."

<u>Delay time</u>: For the case of a Store Negative (STN) instruction directly ahead of an instruction which the STN modifies, it takes six clocks for the Store to reach the Z-buffer from level 4, another six clocks to complete the forced write to memory, eight clocks to fetch the modified instruction into the KA- or KB-buffer, and three more clocks to get it down to level 3 where it was when the instruction hazard was detected. This totals twenty-three clocks for a worst case instruction hazard recovery.

2. Store File Over Instructions

DELAY

This hazard is basically the same as storing over instructions (C.1). The main difference is that the comparisons with program counter addresses P1, PA, and LA are made against AR instead of ZP and ZB. The octet address being stored into is in the AR register at the time the Store File is executed. This address does not pass through the ZP- and ZB-registers as does the address of a Store instruction. Also, there is no need for AR to be compared against the program counter registers at levels 2 and 3 (P2 and P3) because level 2 is blocked from holding any instructions during a Store File and level 3 is where the Store File is being executed; i.e., an instruction cannot modify itself.

No forced write is necessary as a part of Store File instruction hazards. Completion of execution of the Store File implies that the Store File octet is in memory. A refetch of the instruction address occurs when the instruction marked with an instruction hazard reaches level 3.

Hardware solution: Store File instruction hazards are detected by the hardware. AR is compared with P1, PA, and LA on an octet level. If the hazard reaches level 3, instruction hazard recovery starts immediately by fetching the modified instruction octet from memory.

Delay avoidance: Do not modify instructions.

<u>Delay time</u>: Since the instruction hazard recovery process does not start until the flagged instruction reaches level 3, the delay is equivalent to a Branch to a nonresident octet at that point in the program. This delay is eleven clocks.

3. Vectors Storing Over Instructions FUNCTIONAL

Several possibilities exist for this hazard; these may be divided into two classes: (1) a vector storing over subsequent vector instructions and (2) a vector storing over subsequent scalar instructions. For both classes, the FORK/JOIN mode determines the way in which hardware deals with the hazard.

Consider a first vector instruction which has its "Allow Following" bit set to "zero" so that subsequent scalar or vector instructions are not allowed to start execution until all vectors in progress have completed. If the next (second) instruction is a vector, then it is allowed to request the vector parameter file and initialize the Memory Buffer Unit (MBU) of the selected pipe. Or, if the second instruction is a Load File (LF), then the operation of loading the register file can be completed; but that is all; no other instructions can be executed in the JOIN mode.

In either case, the hardware monitors for instruction hazards. This is done by comparing the $\vec{C}$ vector addresses of all vectors in progress with the program counter of the instruction at level 3. If a true comparison is found, then the level 3 far-range hazard flag is set. This flag causes an instruction refetch of the modified second instruction after all vectors have completed.

If the second instruction is changed to something other than the instruction it was, then there is no way the register file can be reinstated to what it was before execution of the second instruction (the VECTL or LF executed while the first vector or vectors were running). If the conditions of this hazard have occurred as stated, then the program will lose control, or produce incorrect answers, as a result of the register file being modified by an instruction that was modified.

If the second instruction is anything other than a Vector or Load File, the hardware monitors for an instruction hazard. This is done by comparing the $\vec{C}$ vector addresses of all vectors in progress with the program counter of the second instruction at level 3. A true comparison will set the level 3 far-range hazard flag, which - in turn - will cause an instruction hazard recovery request after all vectors in progress have completed. In this case no damage has been done since the second instruction at level 3 was never executed before its modification (as was the case with a VECTL or LF instruction). The modified instruction will be executed after the instruction recovery process has been completed.

Consider now a first vector instruction which has its "Allow Following" bit set to "one" so that subsequent scalar or vector instructions are allowed to proceed in parallel with the first vector. If the $\vec{C}$ vector addresses of the first vector happen to write over instruction addresses of subsequent instructions in or above level 3 (but within the IPU), then the near-range hazard signal or the far-range hazard flag will become true, resulting in a refetch of the modified instruction when the hazardous instruction reaches level 3.

This hazard has the potential of causing intermittent problems during checkout because a hazard may show up or not depending upon the phasing of C vectors with respect to instructions in the upper half of the pipe. However, to isolate the problem to some extent, the "Allow Following" bit can be set to zero and the level 3 far-range hazard flag checked at the completion of the vector to see if the next instruction address at level 3 has been overwritten.

Hardware solution: Instruction hazards are marked in the JOIN mode when vectors write over subsequent instructions in the IPU. These hazards may or may not occur, depending on vector-scalar phasing in the FORK mode.

Marked instructions make an instruction hazard recovery request to obtain the modified instruction.

Delay avoidance: Do not allow vectors to write over instruction areas of memory.

Delay time: Instruction hazard recovery time is eleven clocks. Recovery occurs after all vectors have completed for the case in which the last vector had the "Allow Following" bit set to zero.

If the original and modified instruction was a VECTL or LF, then the file load time is lost. Also, the second vector initialization time is lost if a VECTL or VECT is modified.

D. Arithmetic Exception Hazards

1. Load Arithmetic Exception Mask or AE Condition Hazards

DELAY

Three instructions exist which load the arithmetic exception condition or mask registers. These are:

LAM	Load arithmetic exception mask
LAC	Load arithmetic exception condition
LEM	Load arithmetic exception mask and condition

These three instructions must wait until none or only one pipe contains instructions that may modify the arithmetic exception (AE) condition or mask registers. Instructions that modify the AE condition register are an LAC, LAM, or any arithmetic operation that has the potential of setting any of the AE condition bits. These bits are:

Divide check

Fixed-point overflow

Floating-point exponent overflow

Floating-point exponent underflow

Instructions that modify the AE mask register are an LAM or LEM.

If only one pipe contains instructions that may modify the AE condition or mask registers (these instructions generate an AE hazard signal in the pipeline below level 3), then that pipe will be selected, providing the effective address is big (to central memory, $\alpha > 2F$). If the address is small ($\alpha \leq 2F$ and $M = 0$), then the addressed register must not currently be undergoing modification by another pipe. If no modification is taking place, then the pipe containing the AE hazard will be selected. If modification is taking place,and the current modification is by the selected pipe, then the instruction performing the modification must be the last instruction to have entered the pipe containing the AE hazard. In other words, if alpha is small and an alpha register hazard exists, then the short circuit on alpha must occur in the pipe containing the singular AE hazard. If these conditions exist, the AE hazard will not cause a delay.

The AE hazard just described is possible on the LAM, LAC, and LEM instructions. Another delay, closely associated with the AE hazard delay, is encountered when an arithmetic instruction, çapable of producing an arithmetic exception condition, finds an LAM, LAC, or LEM instruction at a lower level of any of the four pipes. If found, the arithmetic instruction must wait at level 3 until the LAM, LAC, or LEM instruction has completed its operation (the pipes are clear of any of these three instructions).

Hardware solution: Any LAM, LAC, LEM, or arithmetic instruction capable of producing an arithmetic exception condition will insert an AE hazard bit into the pipe selected by the said instruction. This bit travels with the instruction as it moves down the pipeline. Any subsequent LAM, LAC, or LEM instruction waits at level 3 until only one or none of the AE hazard bits exist in the four pipes. If only one pipe contains an AE hazard, then that pipe is selected according to the previously stated conditions. If no AE hazards exist, then the pipe is selected according to the known scalar routing rules based on register hazards, short circuits, and X, Y, Z buffer activity.

Also, an instruction which has the potential of an arithmetic exception condition will make a test to see if any LAM or LAC indicator bits are below level 3. If so, then the AE possible instruction waits at level 3 until the LAM or LAC instruction has been cleared from the pipe.

Delay avoidance: Insert other non-AE hazard instructions in front of LAM, LAC, or LEM instructions to allow time for the AE hazard to clear. Also, perform nonarithmetic operations after LAM or LAC instructions to allow time for the LAM or LAC to clear the pipe.

Delay time: AE hazard delay time amounts to four clocks plus AU time plus memory time if the AE hazard producing instruction immediately preceded the LAM, LAC, or LEM instruction.

2. Store Program Status Hazard DELAY

Before a Store Program Status (SPS) instruction can leave level 3, it must have the final result code and condition code for instructions prior to the SPS. Of the four fields stored by the SPS (CP memory usage, BSR, CC, and RC), only the CC and RC can be modified while the SPS is in level 3. CP memory usage does not change during program execution, and the BSR field can only change while an Execute instruction is in level 3. The SPS waits until all result code or condition code modifying instructions have cleared all four pipes. It then proceeds down the pipe selected according to the SPS storage address.

Hardware solution: An SPS instruction makes a test for a signal called hex register hazard in the level 3 controller. This signal will be true if any result code or condition code modifying instructions are in any of the pipes below level 3. When the last instruction to set the result code or to set the condition code (prior to the SPS) has cleared its selected pipe, then the SPS is released from level 3.

Delay avoidance: There is no easy way to avoid a hex register hazard since nearly all instructions either modify the result code or the condition code. The instructions remaining after the RC and CC modifying types are removed constitute only the special operation type instructions: LEM, LAM, LAC, LLA, XCH, LF, LFM, STF, STFM, SPS, LEA, MCP, MCW, INT, PSH, PUL, MOD, BLB, BLX, FORK, JOIN, BCC, BRC, and BAE. Nearly every one of these has its own special delay except for LLA, LEA, and INT. Very little programming can be done using these remaining three instructions.

Delay time: SPS hazard delay time is dependent upon the time for the last result code or condition code modifying instruction to clear the pipe. If this instruction immediately precedes the SPS, then the delay time is four clocks plus AU time plus memory time, providing the AU output does not become blocked due to multiple AU outputs destined for the register file. AU time is from Table 1. Memory time is eight clocks if the hex register hazard producing instruction makes a memory request. It is zero if no request is made.

the latest result code value has not been set yet. The latest result code modifying instruction is below level 3 but has not cleared the pipe. The result code will be set simultaneously with the AU result being placed in the register file. A BRC instruction waits in level 3 until the last instruction to modify the result code has cleared the pipe. There may, at this time, still be other prior result code setting instructions in other pipes, but these result codes were evidently not needed and, in fact, will never affect the result code if some other result code modifying instruction came later and was the last one before a BRC instruction.

Unconditional branch instructions (branches with an R-field of 7) do not experience this delay; they simply make their branch address request upon arrival at level 3. A conditional branch instruction within the top four words of an octet will make a request for its branch address on the assumption that the branch will be taken. This feature employs the "dual look-ahead" hardware of the IPU which is based on the concept that the branch octet of instructions will be available in one of the instruction buffers (either KA or KB) by the time the branch decision is made.

In the event that the branch is not taken, then the fact that at least four instructions along the downstream path still reside in the current instruction buffer allows the normal look-ahead octet of instructions to be refetched while the four remaining downstream instructions are being processed.

Hardware solution: Branch hazards are monitored by examining one of three signals, depending upon the type of branch instruction. The three types of branch instructions are:

BRC	Branch on result code
BCC	Branch on condition code
BAE	Branch on arithmetic exception

These three branch instructions each check for their own type of hazard:

BRC	checks for result code hazards
BCC	checks for condition code hazards
BAE	checks for arithmetic exception hazards

The register stack contains, as one of its components, three columns of bits, one for each type of hazard. These bits track the instructions down the pipe. When all bits in a particular column have cleared, then that hazard associated with that column has cleared. The branch decision can be made when the hazard associated with that branch has cleared.

Delay avoidance: For branch on compare code instructions, result code setting instructions can be inserted between the branch (BCC) and the compare code setting instruction. In this instance, the inserted instructions may be selected from a wide variety of the CP instruction set. This is not quite so true with BRC or BAE instructions since they have the characteristic of waiting on result producing instructions. It is hard to find more than one compare code setting instruction that can be inserted between a BRC or BAE and the last result code or AE setting instruction.

Delay time: Branch hazard delay time is dependent upon the time for the last RC , CC, or AE setting instruction to clear the pipe for a BRC, BCC, or BAE instruction, respectively. If the hazard causing instruction immediately precedes the BRC, BCC, or BAE instruction, then the delay time is four clocks plus AU time plus memory time, providing the AU output does not become blocked due to multiple AU results destined for the register file. AU time is from Table 1. Memory time is eight clocks if the hazard-causing instruction makes a memory read request. It is zero if no request is made.

2. Condition Code Hazard

DELAY

This hazard occurs when a "Branch on Condition Code" (BCC) instruction arrives at level 3 for execution and the latest condition code value has not been set yet. This hazard is similar to the Result Code Hazard just described in section E.1.

3. Arithmetic Exception Branch Hazard

DELAY

This hazard occurs when a "Branch on Arithmetic Exception" (BAE) instruction arrives at level 3 for execution and the latest arithmetic exception condition code has not been set yet. This hazard is similar to the Result Code Hazard just described in section E.1.

II. Scalar-Vector Hazards

A. Vector Parameter File Modification

DELAY

Before a vector instruction can transmit the vector parameters to the Memory Buffer Unit (MBU), it must have the final values in the vector parameter file (VPF). The VPF consists of the lower eight registers of the forty-eight-word register file (words 27 through 2F hex). This file may be acquired from memory (VECTL), or it may be the current contents of the VPF. In either case there must not be any scalar instructions currently in the CP pipeline that will modify the vector parameter file registers. If any such VPF modifying instruction exists below level 3, then the vector instruction at level 3 will wait until the VPF hazard clears.

Hardware solution: A vector instruction at level 3 tests a signal called "any V haz" to determine whether any register of the vector parameter file will be modified by an instruction below level 3. The "any V haz" signal is made from an ORing of four columns of bits in the register stack. These bits track the instruction as it moves down the CP pipeline. The top bit of this column is set when an instruction with an index or vector register destination moves from level 3 to 4. The column splits into four columns at level 5, at the point where four pipes begin.

From the statement describing the setting of the top most bit, it is apparent that the signal "any V haz" also detects any index register modifying instructions. This protects against picking up a wrong index value when the starting indices are added to the vector starting addresses as part of the vector initialization process.

Delay avoidance: Avoid modifying index registers or vector parameter registers immediately before a vector. Insert other arithmetic or base register modifying instructions before vectors.

Delay time: Vector parameter file hazards are dependent upon the pipe time of the index or vector modifying instruction. If the VPF destination instruction immediately precedes the vector, then the delay time is four clocks plus AU time plus memory time, providing the AU output does not become blocked due to multiple AU results destined for the register file. AU time is from Table 1, and memory time is eight clocks if the scalar instruction makes a memory read request.

B. Scalars Writing Over Vector Input Arrays FUNCTIONAL

There is no hardware checking to guard against scalar store instructions writing over vector read data arrays. For this functional hazard to exist, the following conditions must be true:

(a) The CP must have been in the fork mode when the scalar store was executed and must remain in that mode until a first vector after the scalar store is executed with the "allow current" bit set to "one."

(b) The first vector must closely follow the scalar store type instruction. Store types are all stores, push, pull, modify, or exchange instructions.

(c) The scalar store type instruction must write into an area of memory that is used for the initial input data by the vector.

Therefore, in order to not encounter this type of scalar-vector hazard, complement the sense of any condition a, b, or c above. In particular, the simplest method of preventing the acquisition of old data (if condition C is a requirement of the program) is to turn "off" (zero) the "allow current" bit of the first vector after the scalar store.

Hardware solution: Turning "off" (zeroing) the "allow current" bit causes the vector to wait until all pipes have been emptied and all stores currently residing in the Z buffers of the MBU's to be forced into memory. The vector is allowed to make its vector parameter file request while the Z buffers are being emptied.

It is also possible to place the CP in the join mode during execution of the scalar store instruction and then return to the fork mode prior to the vector. If the vector has "allow current" on, then only those pipes executing stores in the join mode will perform a forced write operation to purge their Z buffers of write data. Other non-Z-join pipes are not required to force their data into memory.

Delay avoidance: Negate any of the three conditions a, b, or c above. Preferably, use the join mode to prevent vector read data from being picked up from a given memory area before scalar stores have had time to write into that same memory area.

Delay time: Assume that conditions b and c are program requirements. If this is so, then it is necessary to invoke the join mode to prevent the acquisition of old rather than new data. Now, if the first vector after a scalar store is a VECTL, then the delay time for emptying the joined pipes is overlapped with the load file fetch time for obtaining the vector parameters of the VECTL instruction. The emptying of the joined pipes may also be overlapped with the transmittal of the vector parameters to the MBU. If the joined pipes are not emptied by the time that the vector is ready to request its first read data from memory, then the read requests are held up until the joined pipes are cleared.

If the first vector is a VECT, then the time for emptying the joined pipes can only be overlapped with vector initialization to the MBU. For this case, vector read requests may be delayed more often, depending upon the time required to clear the joined pipes.

C. Vectors Writing Over Scalar Read Data FUNCTIONAL

This hazard is essentially the reverse of hazard II.B just described. In this case a similar set of conditions must be true for the functional hazard to exist:

(a) The vector must have the "allow following" bit set to "one."

(b) The first scalar read must closely follow the vector, and there must not have been any intervening join instructions.

(c) The vector must write into an area of memory from which the scalar read data is obtained.

Therefore, in order to not encounter vector-scalar hazards, complement the sense of any condition a, b, or c above. In particular, the easiest method of preventing the acquisition of old data (if condition c is a requirement of the program) is to zero the "allow following" bit of the vector.

Hardware solution: Turning "off" (zeroing) the "allow following" bit causes the vector to enter a state where it waits for any vectors in progress to complete before executing the next instruction at level 3. Any vector in progress includes the current vector, so it is not possible for a subsequent scalar (except for a load file instruction) to begin execution until the current vector and all preceding vectors have completed. This also includes the completion of all outstanding scalar reads that may be in progress in other pipes during the current vector execution.

Preventing the subsequent scalar from beginning execution guárantees that vector output data will have been written into memory before a scalar read request is issued for the same area of central memory.

Notice that load file scalar requests are allowed in the join mode. This is for the purpose of obtaining a new vector parameter file and then making singleword or halfword modifications before executing a VECT instruction. The memory address of the load file octet is monitored throughout the duration of the vector with "allow following off." Should the vector write over the load file octet location, a flag (alpha hazard flag) will be set to "one." This flag causes the load file octet to be refetched at the completion of the vector.

A hazard that is noncorrectable, along this same line of thought, is when the vector writes over the instruction location of the load file instruction. Should this happen, it is not possible to reconstruct the state of the register file prior to execution of the load file instruction. If the load file instruction location is modified to anything other than a load file to the same register file, then the program will most likely produce erroneous results. Again, the rule "never modify instructions" should prevent most individuals from making this mistake.

Delay avoidance: Negate any of the three conditions a, b, or c preceding. Preferably, turn off the "allow following" bit to prevent reading scalar data before the preceding vector has written into a given area of memory.

Delay time: There is no overlapping of subsequent scalars (except for load file instructions) if the "allow following" bit is "zero." Prior vectors may continue processing in parallel with the last vector if the prior vectors had their "allow following" set to "one." However, the next scalar after the last vector must wait until the longest vector has completed since it waits for all vectors to complete. The longest vector may not necessarily be the last vector. Therefore, the actual delay may exceed the expected delay unless the length of all vectors are taken into account.

If the next instruction after the vector with "allow following" set to "zero" is another vector, then the next vector is allowed to request its vector parameter file and to initialize the MBU but to go no further.

D. Alpha Hazards During Vectors in the Fork and Join Mode

FUNCTIONAL

DELAY

To begin with, scalar instructions by themselves cannot cause a functional alpha operand hazard. Emphasis here should be placed on the word "functional" as that means a possible error-producing hazard. Alpha operand delays do occur among purely scalar instructions (scalars executed by themselves, clear of vectors), but these delays do not produce errors in results. These delay-generating conditions were described in section B under Central Memory Address Hazards.

Scalar instruction hazards were discussed in sections I.C.1 and I.C.2 for scalars operating clear of vectors and in section I.C.3 for scalars and vectors operating in parallel.

What is of concern in this section is scalar alpha hazards caused by a prior vector. That is, a vector writing over the memory location of a subsequent vector parameter file of a VECTL instruction or a subsequent Load File from memory. Vectors writing over scalar read data was discussed in Section II.C.

Assume that the join mode has been established within some list of scalar instructions prior to the arrival of a first vector at level 3. If the vector is a VECTL, which receives its vector parameter file (VPF) from memory, a test is made for an alpha hazard before issuing the load file request. Since being in the join mode prior to the arrival of this vector guarantees that there are currently no other vectors in progress, the alpha hazard detection hardware initiates a forced write signal to the MBU containing the hazard-producing store; i.e., the store whose address agrees with the alpha address of the vector instruction. The VPF load request is issued after the forced write has been completed. Delay, in this case, is due to waiting for write data to arrive in memory prior to issuing the read request. Had the alpha hazard not existed, the forced write operation would have taken place as a normal part of the preparation for the vector in the join mode; but the VPF load would not have had to wait on the write to complete.

Now, consider the case where a first vector is executed in the fork mode but has its "allow following" bit set to "zero"; then, suppose a second vector follows immediately. This second vector makes its test for alpha hazards in the "vector plus one" state. Here the situation is different because now it is possible for both vectors and scalars to be in progress simultaneously in any of the four pipes. The alpha hazard logic cannot issue a forced write command to a pipe that is executing a vector because (fortunately) that pipe will simply ignore the command. The logic will, however, issue a forced write to the pipes containing scalar stores; and, in addition, the alpha hazard logic will cause the VPF load request to be delayed until the scalar forced writes are completed. The unprotected case is seen to be when the first vector, with "allow following" off, writes over the vector parameter file (in memory) of the succeeding vector. It is difficult to protect against this case with hardware since the vector parameter file address that is contained in the AR register of level 3 is erased during vector initialization. That is, register AR becomes involved in the process of transmitting vector parameters to the MBU and cannot continue the function of monitoring a VPF address in AR against all output addresses of the first $\vec{C}$ vector. It is more important, in terms of time that would otherwise be lost on all joined vector initializations, for the initialization to proceed and to sacrifice the alpha hazard hardware checking for this case. Protection of a VPF in memory can be insured by software by using the sequence:

```
     SEQ1   VECTL     "AF" = 0
            LF        (load VPF)
            VECT
```

instead of,

```
     SEQ2   VECTL     "AF" = 0
            VECTL
```

The VPF of the second VECTL of sequence 2 is unprotected since the second VECTL proceeds through initialization while the first VECTL is still executing. In sequence 1 the alpha address of the LF instruction is checked against all $\vec{C}$ vector addresses of the first VECTL, so the VPF for VECT is protected against modification by VECTL.

Hardware solution: A test is made for scalar alpha hazards prior to making the load file request for a VECTL instruction. This is done regardless of the fork or join mode but is intended for the join mode. If an alpha hazard, due to a prior scalar storing over a VECTL's alpha address, is seen during the fork mode, then the forced write takes place the same as in the join mode. However, if a prior vector writes over a VECTL's alpha address and the VECTL is executed, all the while remaining in the fork mode, then the alpha hazard will not be seen and the VECTL can pick up its VPF prior to modification by the first vector.

Delay avoidance: Avoid modifying vector parameter files (VPF) of subsequent vectors by means of scalar stores or vector writes (to memory) which are in the immediate vicinity of a vector using the VPF being modified. If separation of modification from use by the method of instruction insertion is not feasible, then at least insert a join instruction between modification and use to prevent functional errors.

Delay time: Needs FUSS prediction.

III. Vector Hazards

A. Vectors Storing Over Their Own Input Data Arrays $\boxed{\text{FUNCTIONAL}}$

This hazard results from a violation of the familiar "Vector Hazard Rule," which states that:

> A "hazard condition" occurs whenever the present octet address of input vector $\vec{A}$ or $\vec{B}$ or the next four octet addresses for each of vectors $\vec{A}$ or $\vec{B}$ is the same as the present result octet address or the eight past result octet addresses of output vector $\vec{C}$.

If the Vector Hazard Rule is violated, the "old" rather than the "new" (updated) information is used as the operand. For example, a vector will use the "old" values for the $\vec{B}$ vector operands if the element address of c_i is one greater than the element address of b_i and all vectors are assigned a positive increment direction during the self-loop. Hardware is not built in to detect this hazard. Also, delay avoidance and delay time descriptions do not apply to functional hazards.

B. Addressing Conflicts Between Two or More Vectors Executing Simultaneously in Parallel Pipes

FUNCTIONAL

This addressing conflict is caused by the independence of separate pipes executing in parallel. Unless one is careful, two vectors started in separate but parallel pipes can have their data paths (in memory) cross one another. Input paths crossing input paths cause no trouble. However, let an input data path cross behind an output data path of an earlier vector; and program errors are almost certain. If the memory paths cross like an "X," then the time and phase (ahead or behind) relationship of reaching the intercept point is important. Since vector rates are influenced by memory interference, it is difficult to predict the time of reaching the intercept point for either vector. Therefore, rate control is impossible. So, to prevent memory read-write collisions between vectors, these intercepting vectors must be executed independently of one another.

Another type of intercept would be the parallel path type, particularly when the parallel paths form a single line tracing the same area of memory. In this case the vector rates are also quite critical to program execution. For example, if the output vector were trailing an input vector through contiguous locations, all would be fine (assuming the output vector was the second vector to start execution). Suppose that, subsequently, the output vector of pipe 1 caught up with and passed the input vector of pipe 2. Now, the first vector would be reading output data from the second vector, a difficult condition to contend with, especially when trying to interpret program results following execution; i.e., man, take a look at that dump! These vectors must also be executed independently óf one another.

Hardware is not implemented to detect this hazard. Also, delay avoidance and delay time descriptions do not apply to functional hazards.

C. Halfword Z-fill-in Hazards

FUNCTIONAL

This hazard is similar to the one just described in section III.B since it is caused by two or more vectors executing simultaneously in parallel pipes. An unusual characteristic of this hazard is that it is due to two or more halfword output vectors writing over the same area of memory. It is also quite difficult to predict the hazard based on Fortran compiler algorithms for finding two vectors writing into the same memory space because the errors may arise in halfword locations within the octet being modified and not necessarily at the halfword location common to both vectors.

The hazard is due to the fact that halfword output vectors, which do not fill up an entire octet with halfwords, require a halfword Z-fill-in operation. A Z-fill-in involves both a read and a write cycle, controlled by the Memory Buffer Unit. Since memory only accepts singleword stores (there are eight zone enable lines controlling the word of an octet to be stored), the MBU must recognize halfword stores which do not fill both halves of a singleword location. This is done by examining the sixteen halfword zone bits of the Z buffer of the MBU. Any even-odd bit pair forming a true exclusive or logical combination indicates the need for a Z-fill-in operation.

The operation is as follows:

(1) The address of the octet to be stored into memory is first sent to memory as a read request for that octet.

(2) The requested octet is loaded into the ZB buffer of the MBU.

(3) The "filled" halfwords of the Z buffer are transferred to the ZB buffer. "Unfilled" halfwords are not transferred.

(4) The "now updated" ZB buffer is written into memory at the storage octet address. Zone enable lines are a logical "one" for those singlewords that have been updated.

A halfword Z-fill-in hazard occurs when the following sequence occurs:

(1) A first pipe requests an octet for the purpose of Z-fill-in.

(2) A second pipe requests the same octet for the purpose of Z-fill-in.

(3) The first pipe modifies one halfword of the octet and then stores the octet back into memory.

(4) The second pipe modifies some other halfword of the same octet and then stores it back into memory.

Operation 4 above has just erased the work done by the first pipe. Results in the common octet which were to have been stored by the first pipe are lost, and recovery is impossible.

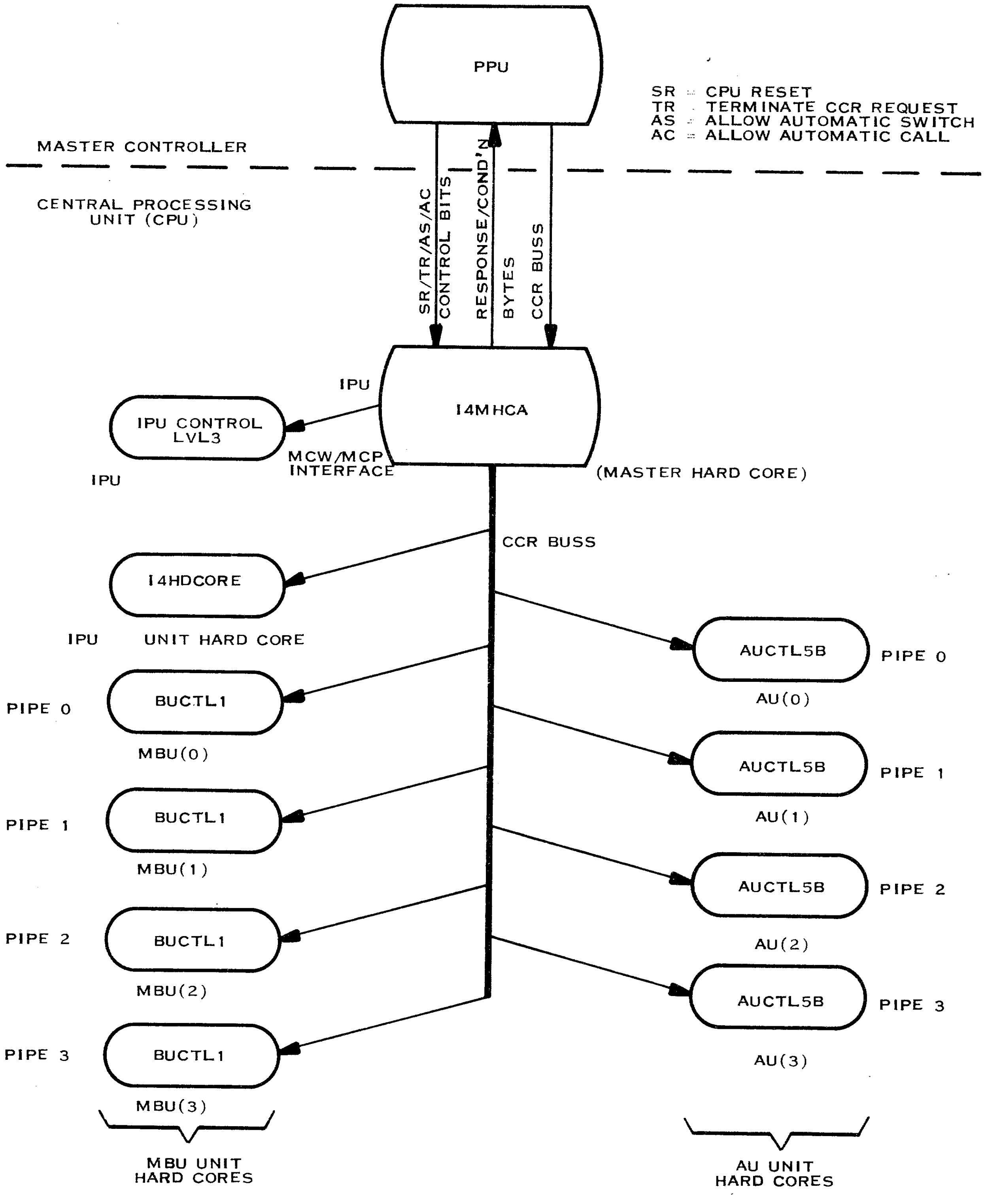
In order to protect against this functional hazard, one must make a complete examination of halfword $\vec{C}$ output vectors, that are running simultaneously in the fork mode, to be sure that memory octets common to both vectors are not being used. If this is not possible, then vectors with halfword outputs should be run with "allow following" turned off, set to zero.

Hardware is not implemented to detect this hazard. Also, delay avoidance and delay time descriptions do not apply to functional hazards.

APPENDIX D
HARD CORE

Page

- Master Controller/Master Hardcore/Unit Hardcore Overview D-1
- Master Hardcore D-7
- IPU4 Unit Hardcore D-28
- MBU4 Unit Hardcore D-63
- AU4 Unit Hardcore D-68
- BUMHC and LOGCLK-8 Logic Cards D-75
- CP-4X UR Dump Interpretation D-91



(A)132485

Figure D-1. Master Controller/Master Hardcore/Unit Hardcore Overview

D-1 PP RESPONSE POLLING OF THE CP

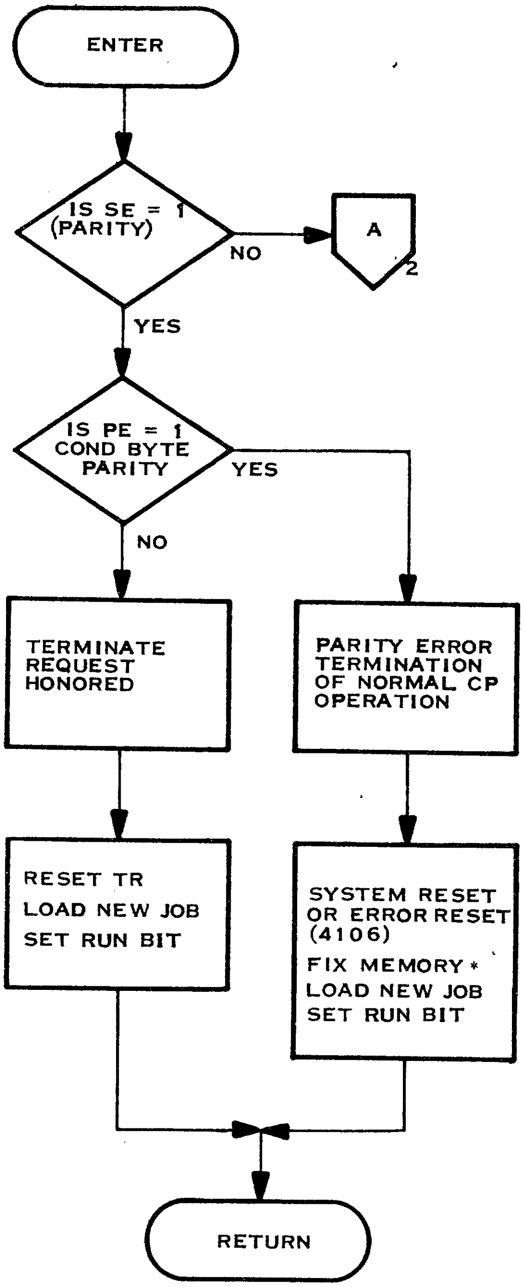
The Peripheral Processor (PP) monitors the CP response bits in its CR File, Register 12 (hexadecimal). These response bits are System Error (SE), Abnormal Termination (AT), Message Complete (MC) and Switch Complete (SC). If the CP sets one of these bits, the PP performs a set of actions to recover from the condition that produced the response bit. Figure D-2 illustrates the polling loop that the PP follows. The following paragraphs describe the PP response to each of the conditions.

D-2 SYSTEM ERROR. If the CP detects a parity error during Hardcore operation, it sets the SE response bit. The PP then checks the PE bit in the condition byte to determine if the parity error occurred during a memory fetch. If PE is set, the PP indicates a parity error termination, issues a reset command to the CP, loads a new job into the CP via an exchange intermediate CCR command, and sets the CP run bit to start the new job into operation. If PE is not set, the PP resets the terminate request bit in the control byte, loads a new job into the CP and sets the Run bit to start the new job into operation.

D-3 ABNORMAL TERMINATION. If a maintenance command executing in the CP encountered a memory error condition and terminated abnormally, the CP will set the AT bit in the CP response byte of the PP CP File. If this bit sets, the PP inspects the SC and MC bits to determine the condition prior to termination.

If SC and MC are both clear, the CP is inactive between jobs and is ready to receive new commands. If MC is clear but SC is set, and error switch was in progress when the CP terminated the sequence. If MC is set, a monitor call resulted in program termination. The condition of the SC bit indicates which call was present: SC = 1 indicates a MCW operation; SC = 0 indicates a MCP operation. The PP then determines what conditions caused the termination by inspecting the PE and PV condition byte bits. If PE is set, then a memory parity error or other system error caused the termination. The condition in memory must be corrected before the job can run in that area of memory. The PP loads a new job into the CP and sets the Run bit to start the CP on the new job. If the PV bit is set, then a memory operation encountered a memory protect violation causing the CP program to terminate. The memory protect parameters in the MCU must be adjusted to allow the program to access the required area in memory. The PP loads a new job into the CP and sets the Run bit to start the operation in the CP. If neither of the two condition byte bits are set, the termination resulted from a termination request issued by the PP. The PP resets the Termination Request bit (TR), loads a new job into the PP and sets the Run bit to start the new job into the CP.

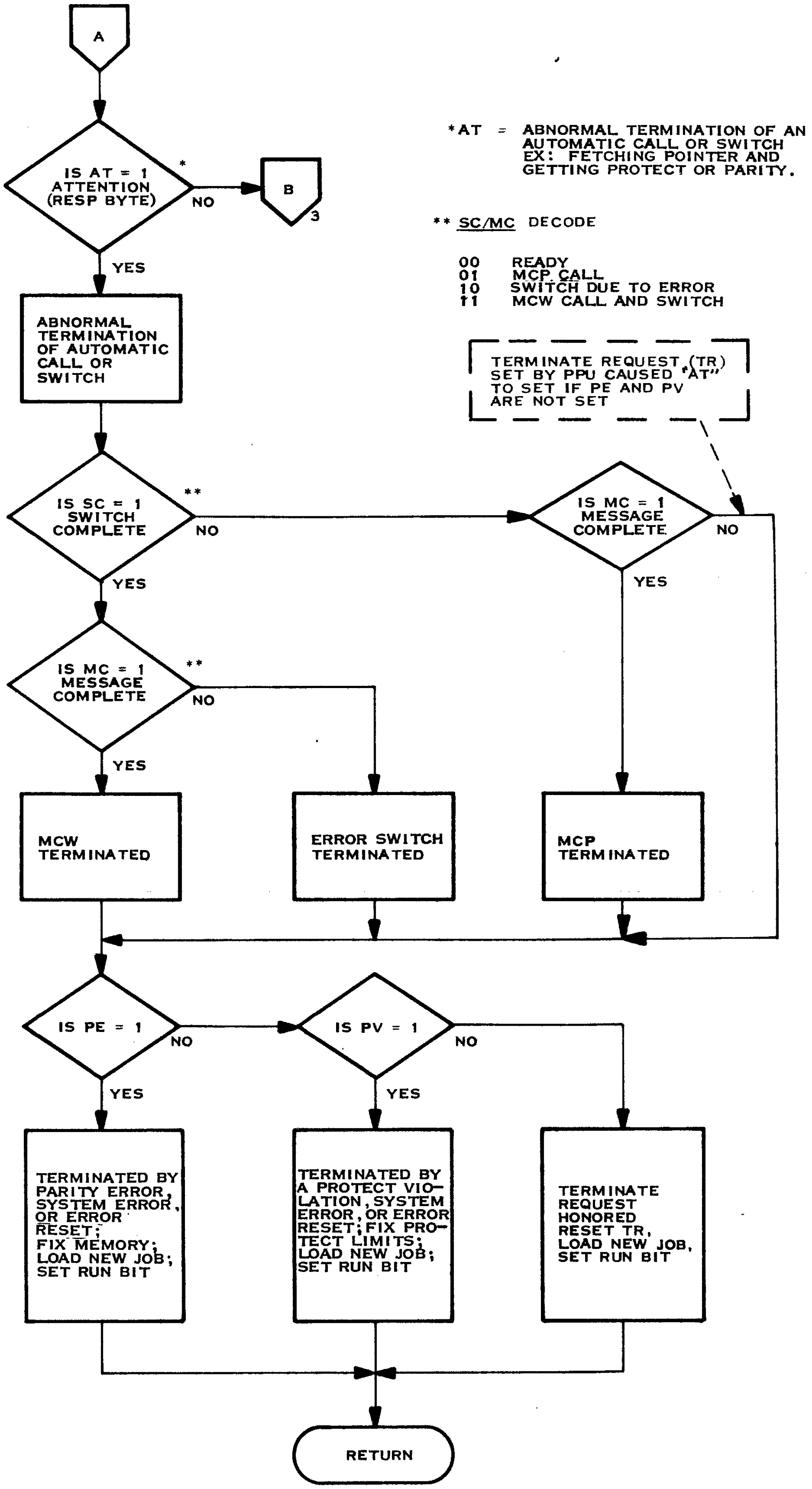
D-4 NORMAL TERMINATION. If no system error or abnormal termination occurs, the PP inspects the SC and MC response bits to determine if a switch operation or call instruction executed properly. If neither of these two bits is set, some condition has prevented completion of the operation. The PP inspects the reason code bits to determine the cause. If SC is set, but MC is not, then the CP has completed an error switch operation. The PP must examine the condition byte to determine the nature of the error, prepare a new job for exchange operations in case the current job in the CP requires PP intervention, and


ENTER
IS SE = 1 (PARITY)
NO
A
2
YES
IS PE = 1 COND BYTE PARITY
YES
NO
TERMINATE REQUEST HONORED
PARITY ERROR TERMINATION OF NORMAL CP OPERATION
RESET TR
LOAD NEW JOB
SET RUN BIT
SYSTEM RESET OR ERROR RESET (4106)
FIX MEMORY*
LOAD NEW JOB
SET RUN BIT
RETURN

*FIX MEMORY—IF PARITY OCCURS, IT IS ASSUMED THAT MEMORY IS BROKEN AND REQUIRES REPAIR AND THEREFORE NO ATTEMPT TO SWITCH IS MADE.

(A)115842

Figure D-2. PP Automatic Interrupt or Polling Loop of CP Status (Sheet 1 of 4)


A
IS AT = 1 ATTENTION (RESP BYTE)
*
NO
B
3
YES
ABNORMAL TERMINATION OF AUTOMATIC CALL OR SWITCH
IS SC = 1 SWITCH COMPLETE
**
NO
YES
IS MC = 1 MESSAGE COMPLETE
NO
YES
TERMINATE REQUEST (TR) SET BY PPU CAUSED "AT" TO SET IF PE AND PV ARE NOT SET
IS MC = 1 MESSAGE COMPLETE
**
NO
YES
MCW TERMINATED
ERROR SWITCH TERMINATED
MCP TERMINATED
IS PE = 1
NO
YES
IS PV = 1
NO
YES
TERMINATED BY PARITY ERROR, SYSTEM ERROR, OR ERROR RESET; FIX MEMORY; LOAD NEW JOB; SET RUN BIT
TERMINATED BY A PROTECT VIOLATION, SYSTEM ERROR, OR ERROR RESET; FIX PROTECT LIMITS; LOAD NEW JOB; SET RUN BIT
TERMINATE REQUEST HONORED RESET TR, LOAD NEW JOB, SET RUN BIT
RETURN

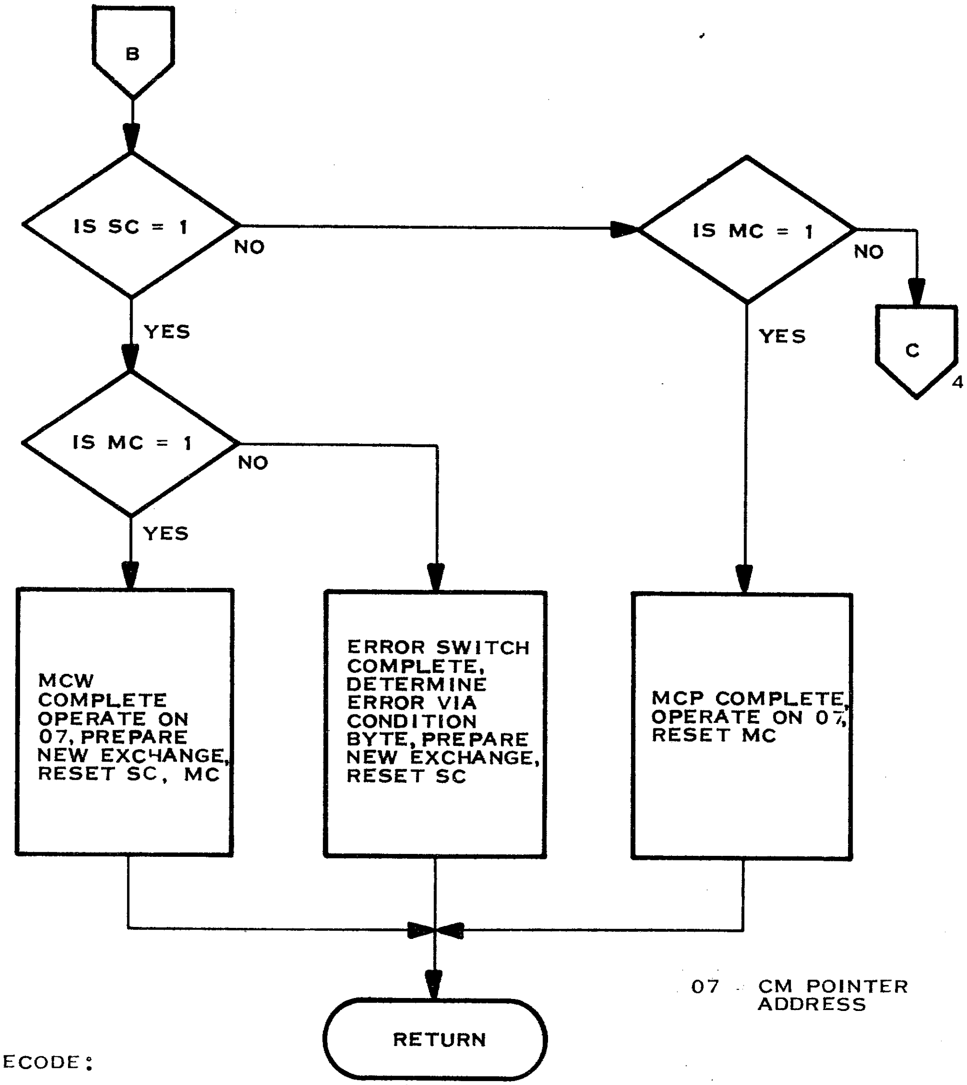
*AT = ABNORMAL TERMINATION OF AN AUTOMATIC CALL OR SWITCH EX: FETCHING POINTER AND GETTING PROTECT OR PARITY.

** SC/MC DECODE

00 READY
01 MCP CALL
10 SWITCH DUE TO ERROR
11 MCW CALL AND SWITCH

(A)115843

Figure D-2. PP Automatic Interrupt or Polling Loop of CP Status (Sheet 2 of 4)


B
IS SC = 1
NO
YES
IS MC = 1
NO
YES
C
4
IS MC = 1
NO
YES
MCW COMPLETE OPERATE ON 07, PREPARE NEW EXCHANGE, RESET SC, MC
ERROR SWITCH COMPLETE, DETERMINE ERROR VIA CONDITION BYTE, PREPARE NEW EXCHANGE, RESET SC
MCP COMPLETE, OPERATE ON 07, RESET MC
RETURN

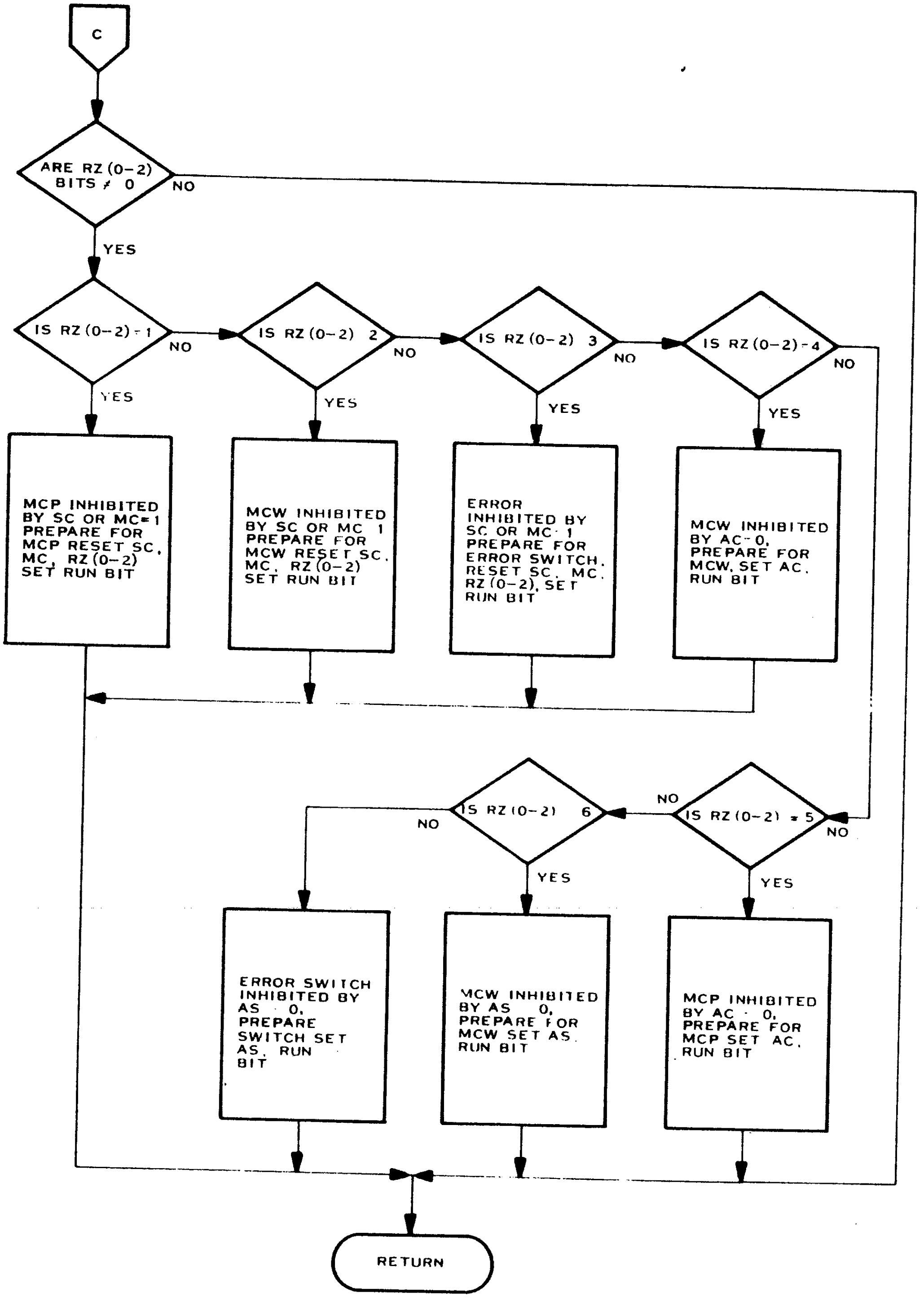
07 - CM POINTER ADDRESS

SC/MC DECODE:

00	READY
01	MCP CALL
10	SWITCH DUE TO ERROR
11	MCW CALL AND SWITCH

(A)115844

Figure D-2. PP Automatic Interrupt or Polling Loop of CP Status (Sheet 3 of 4)


C
ARE RZ (0-2) BITS ≠ 0
NO
YES
IS RZ (0-2) = 1
NO
YES
IS RZ (0-2) 2
NO
YES
IS RZ (0-2) 3
NO
YES
IS RZ (0-2) = 4
NO
YES
MCP INHIBITED BY SC OR MC=1 PREPARE FOR MCP RESET SC, MC, RZ (0-2) SET RUN BIT
MCW INHIBITED BY SC OR MC 1 PREPARE FOR MCW RESET SC, MC, RZ (0-2) SET RUN BIT
ERROR INHIBITED BY SC OR MC 1 PREPARE FOR ERROR SWITCH, RESET SC, MC, RZ (0-2), SET RUN BIT
MCW INHIBITED BY AC = 0, PREPARE FOR MCW, SET AC, RUN BIT
IS RZ (0-2) 6
NO
YES
NO
IS RZ (0-2) = 5
NO
YES
ERROR SWITCH INHIBITED BY AS 0, PREPARE SWITCH SET AS, RUN BIT
MCW INHIBITED BY AS 0, PREPARE FOR MCW SET AS, RUN BIT
MCP INHIBITED BY AC 0, PREPARE FOR MCP SET AC, RUN BIT
RETURN

(A)115845

Figure D-2. PP Automatic Interrupt or Polling Loop of CP Status (Sheet 4 of 4)

reset the SC bit. If MC is set, then a monitor call operation has successfully completed. The condition of the SC bit indicates which of the two monitor calls was completed. In either case, the PP pulls the call pointer from memory location 07 and performs the required operation to fulfill the CP's requirements. The PP also resets the MC bit, and, if the operation was an MCU, prepares a new exchange to be ready in case the current program in the CP completes or requests context switching via another MCW operation.

D-5 IPU4 MASTER HARDCORE (I4MHCA)

The Master Hardcore Controller card is located on the Pipe Motherboard in the IPU Column of the X4 CPU. It responds to three types of CPU actions:

(1) PPU Common Command Register (CCR) Servicing

(2) CPU Error Mode Servicing

(3) CPU Monitor Call Servicing.

These actions are further discussed in the following paragraphs.

D-6 PPU COMMON COMMAND REGISTER SERVICING. The following CCR Commands are serviced by the X4 Master Hardcore (CP CCR Commands CR#OC, Bytes 0 and 1):

4100 THRU 4103	NOP
4104	RESET ALL CP REGISTERS
4105	SET ALL CP REGISTERS
4106	RESET CP ERROR CELLS (AE(0-3),PROTECT,PARITY,ILLEGAL OPCODE)
4107	NOP
4108	STORE CP STATUS MAP USING POINTER AT LOCATION #14.
4109	RESET CP PIPES AND LOAD STATUS MAP USING POINTER AT LOCATION #15.
410A*	EXCHANGE CP STATUS MAPS USING POINTERS AT LOCATIONS #14 AND #15 OR #19, AND LOAD MAP AND PROTECT REGISTERS USING POINTER AT LOCATION #28.
410B	STORE CP DETAILS MAP USING POINTER AT LOCATION #18.
410C	LOAD CP DETAILS MAP USING POINTER AT LOCATION #19.
410D*	EXCHANGE CP DETAILS MAP USING POINTERS AT LOCATION #18 AND #15 OR #19, AND LOAD MAP AND PROTECT REGISTERS USING POINTER AT LOCATION #28.
410E	RESET CP RUN BIT AND STORE CP DETAILS MAP USING POINTER AT LOCATION #18.
410F	RESET CP RUN BIT AND LOAD CP DETAILS MAP USING POINTER AT LOCATION #19.

*THE LOAD CYCLE OF 410A AND 410D DEPENDS UPON LOAD STATUS BIT (CR #14, BYTE #3, BIT #1)

4110 RESET CP RUN BIT

4111 SET CP RUN BIT

42mn TURN ON CP CLOCK ACCORDING TO m =

- 0 - CONTINUOUS NORMAL CLOCK
- 1 - CONTINUOUS MARGINAL CLOCK
- 2 - CONTINUOUS SLOW CLOCK
- 8 - BURST OF n NORMAL CLOCK PULSES
- 9 - BURST OF n MARGINAL CLOCK PULSES
- A - BURST OF n SLOW CLOCK PULSES

44mn STORE BYTE n OF CP HARDCORE INTO THE UNIT REGISTER ACCORDING TO m =

- 0 - CP MASTER HARDCORE
- 4 - IPU HARDCORE
- 2 - MBU PIPE 0 HARDCORE
- 3 - MBU PIPE 1 HARDCORE
- 6 - MBU PIPE 2 HARDCORE
- 7 - MBU PIPE 3 HARDCORE
- 8 - AU PIPE 0 HARDCORE
- A - AU PIPE 1 HARDCORE
- C - AU PIPE 2 HARDCORE
- E - AU PIPE 3 HARDCORE

The two exchange commands are recorded into a "Store" command followed by a "Load" command by the Master Hardcore. Thus if a 410A is received from the PPU, a 4108 is sent to the Unit Hardcores; if a 410D, a 410B is sent. The second pass of an Exchange command, the "Load" cycle, is controlled by I4CRSB (status bit), which is sent by the PPU. If I4CRSB = 0, on the load cycle, a 410D is sent to the Unit Hardcores; if I4CRSB = 1, a 4109 is sent to the Unit Hardcore.

In the Master Hardcore flowcharts and logic diagrams, the commands have been grouped as follows:

Definition	Mnemonic	CCR Code
1.) Simple Commands	I4SIMCMD	4100 thru 4107
2.) Checked Commands	I4CHKCMD	4108 thru 410D
3.) Memory Commands	I4MEMCMD	410E and 410F
4.) Set Run bit Command	I4SRBCMD	4111
5.) Reset Run bit Command	I4RRBCMD	4110

Within the I4CHKCMD group are the two exchange commands discussed above, and this sub-group is called I4EXCCMD.

D-7 CAPTURE CCR. The capture CCR logic is part of the master hardcore circuitry that monitors the transfer bit (TB) and unit code of the Command Command Register (CCR) in the Peripheral Processor Communications Register File. The flowchart in figure D-3 illustrates the control cycle. If TB sets, the PP is issuing a command to one of the system devices. The controller then inspects the unit code of the command to determine if the command is intended for the CP. A code of 41 (in hexadecimal) specifies a CP command. The controller then inspects the Request Present flag (QRPF) to determine if another request is currently being processed by the hardcore logic. If this flag is set, the controller must wait until it clears before proceeding. When the flag clears, the controller activates the Reset TB (RSTB) and Gate CCR (GCCR) lines to the CR File to transfer the new command into the CP. When the PP returns a recognition of the transfer (TBRL), the controller deactivates the two signals and activates the Request Present indicator (RP) to indicate to the hardcore logic that a new command is resident. The controller then ensures that the TB has not yet reset.

NOTE

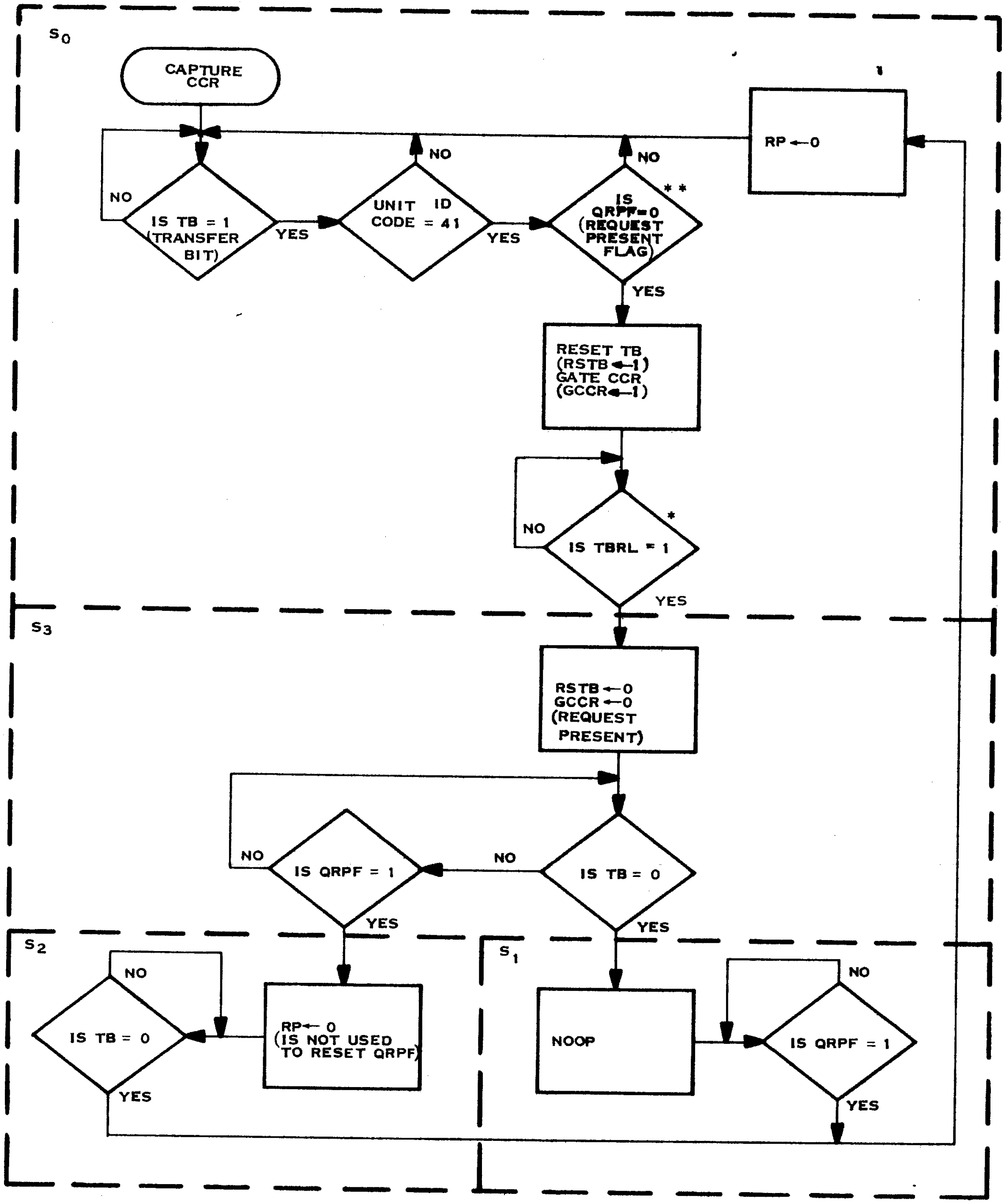
Due to the long clock period of the CR File with respect to the CP, the CP has several clock periods before TB resets. Therefore, if TB is reset at this point, it should not have been set. This feature also provides a timeout that negates the CCR command if QRPF does not set within a reasonable time.

If TB is still set, the controller waits for the hardcore logic to set QRPF as a result of RP being active. QRPF indicates an active command in the hardcore logic. When QRPF sets, the controller drops RP and waits for TB to clear. When TB clears, the cycle is complete and the controller is ready to begin the cycle again.

D-8 CPU ERROR MODE SERVICING (See Figures D-4 through D-13)

Two modes of error can occur in the CPU that are monitored and handled by the Master Hardcore. They are: (1) Program Errors, and (2) System Errors. Program Errors are defined as errors that occur while a program is running in the CPU. They are further broken down into Protection Violations, Parity Errors, Illegal Opcodes Encountered (these three types may be generated by either the IPU or one of the MBU's) and Arithmetic Exceptions (developed by the AU and buffered in the IPU). System Errors are defined as errors that occur while a hardcore command is being processed. They may occur because of a Protection Violation or Parity Error occurrance in either the IPU, the MBU's or the AU's during the hardcore command, or because of a "Terminate Hardcore Command Request" signal from the PPU, which is buffered as I4QTR in the Master Hardcore.

Program errors are buffered individually in the Master Hardcore in I4QPV, I4QPE, I4QIL, and I4QAE, respectively for Protection Violation, Parity Error, Illegal Opcode and Arithmetic Exception; the System Error, Protection Violation or Parity Error, condition is buffered as I4QME. These are gated to the PPU by I4QGCB.



NOTE: THIS IS ALL ASYNCHRONOUS LOGIC. (1→ XXX REQUIRES NO CLOCK)

*TBRL –LATCH OUTPUT ON CCR COOKIE SHEET THAT LOOKS AT ALL TB RESET LINES AND WHEN TBR CLK OCCURS IT GETS SET SO THAT THE MHC KNOWS HOW LONG TO HOLD RSTB = 1 DUE TO CCR 500 NS CLOCK

**QRPF – GETS RESET VIA COMMAND COMPLETE IN SYNCHRONOUS LOGIC STATE 7.

ASYNCHRONOUS CCR LOADER FLOWCHART SYMBOL	14QRP (0)(1)
C0	00
C1	01
C2	10
C3	11

HC)

(A)115837

Figure D-3. CP Master Hardcore (MHC) Capture CCR Logic Flowchart

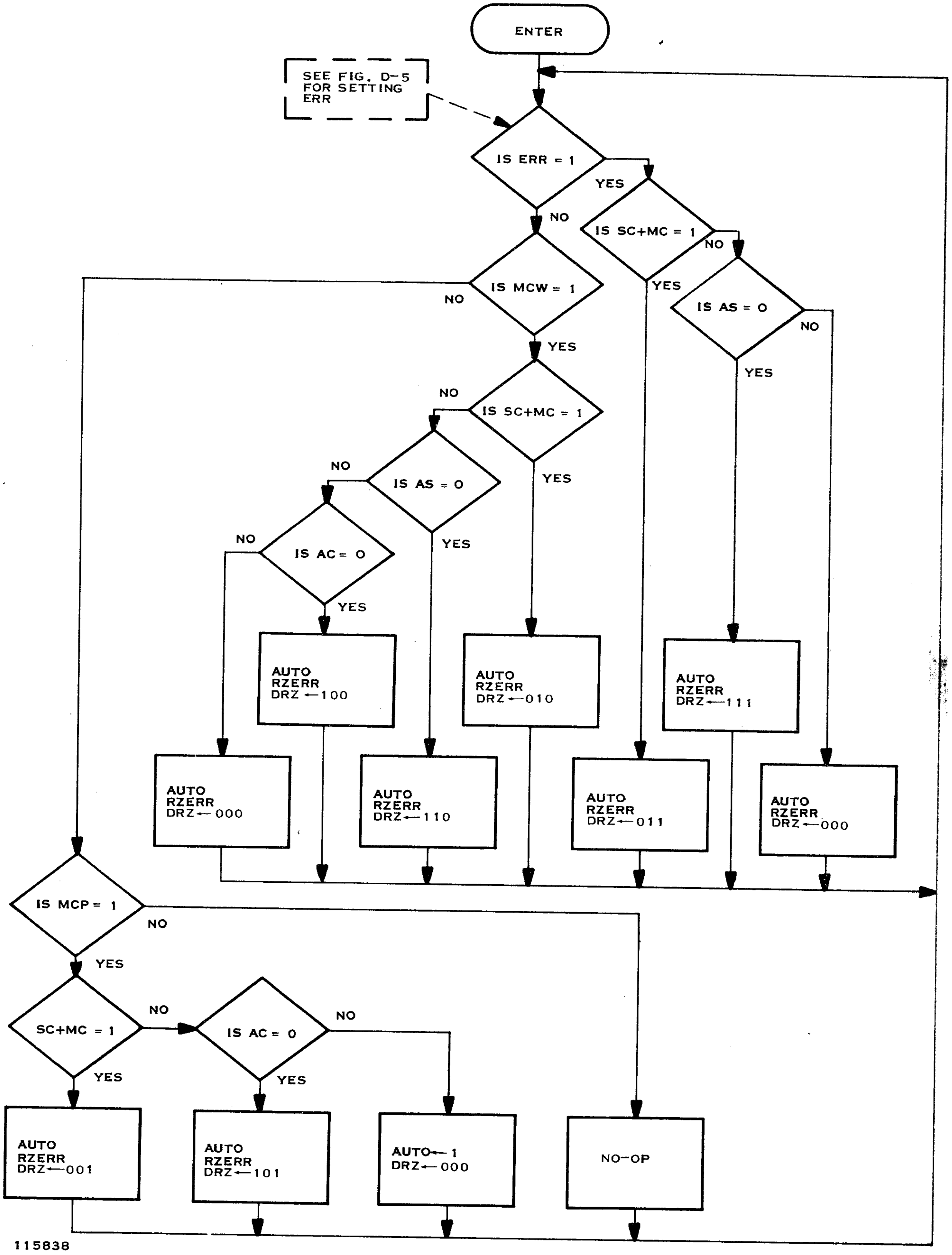


Figure D-4. MHC Monitor Flowchart

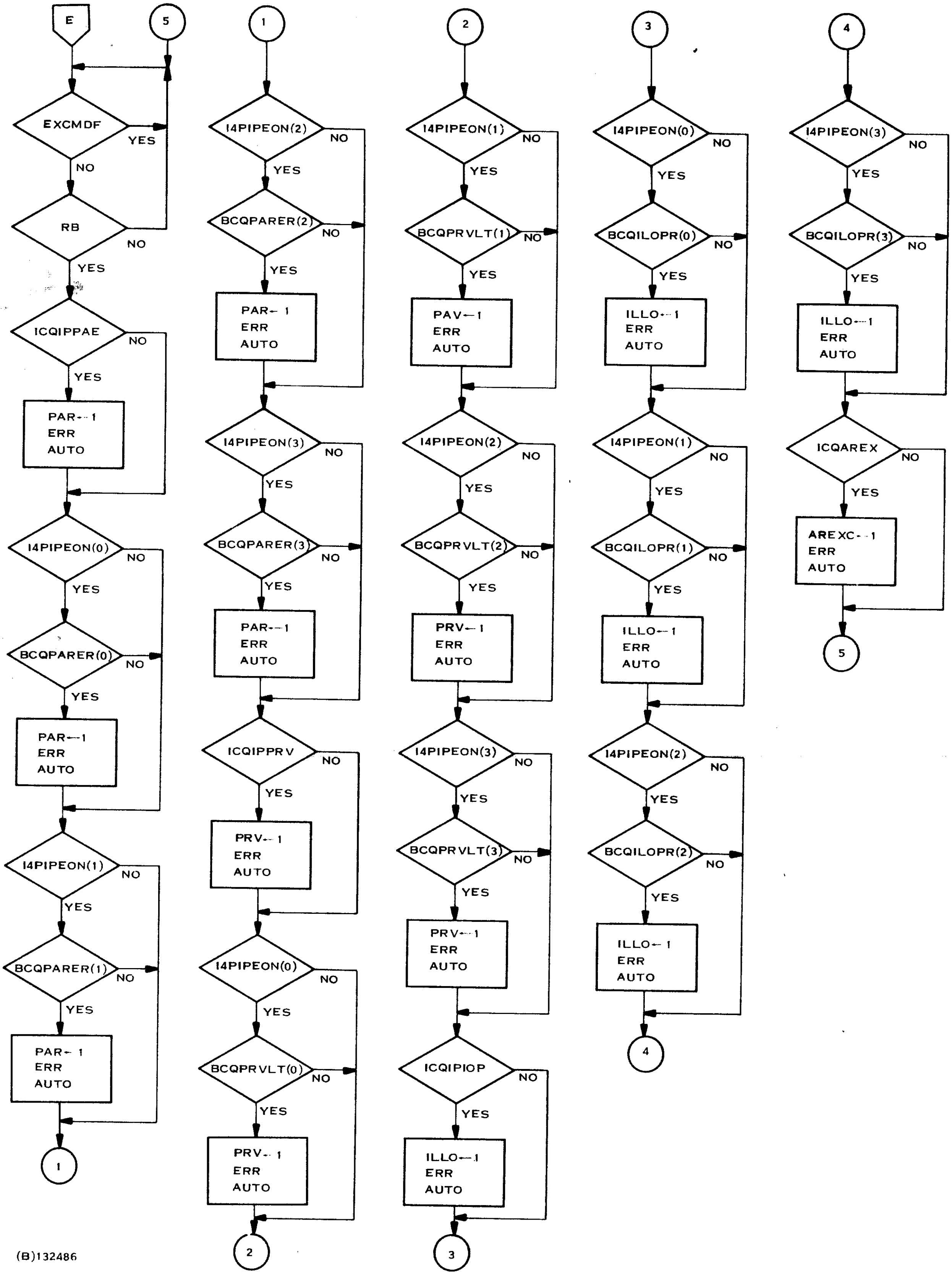
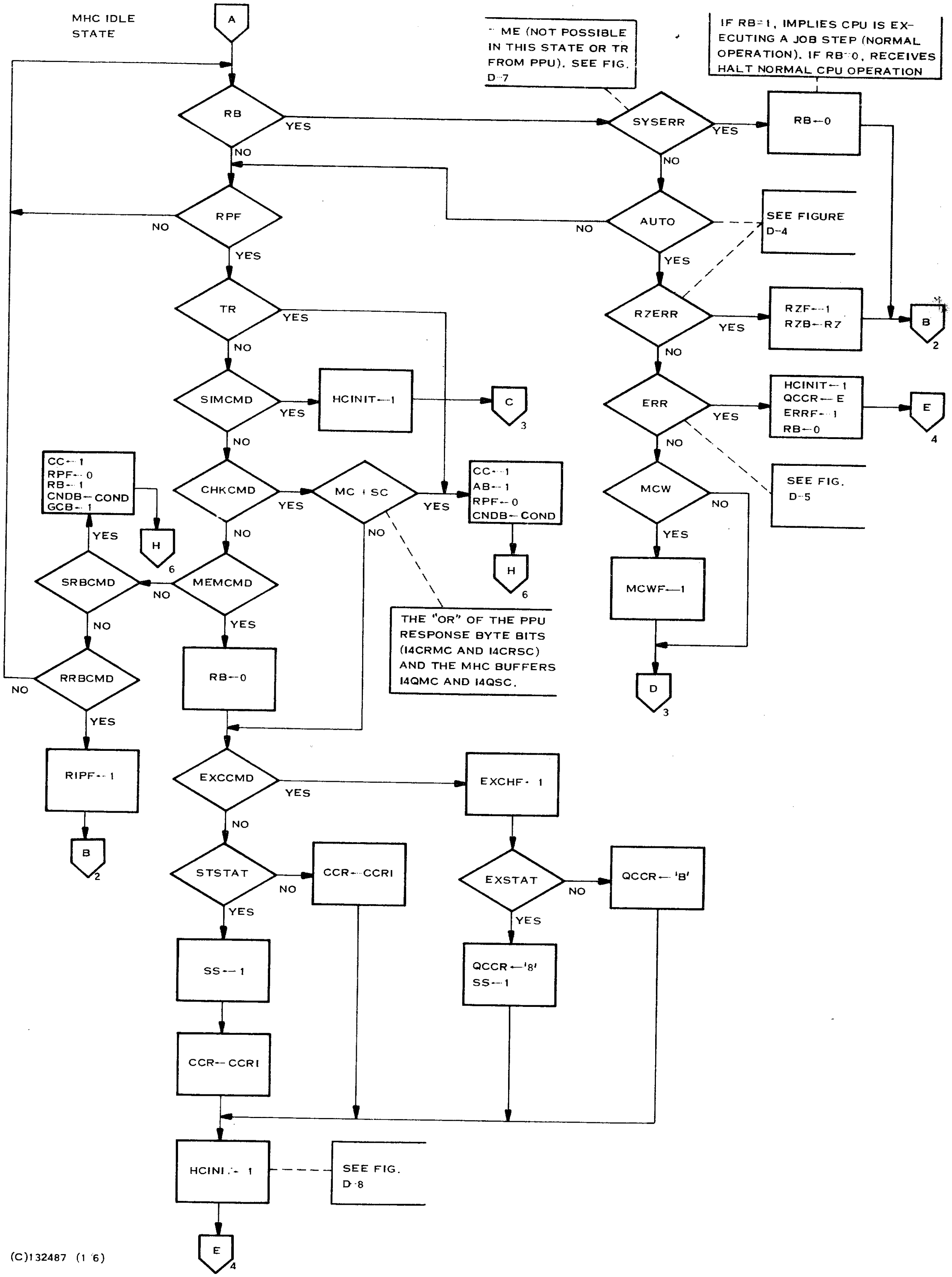
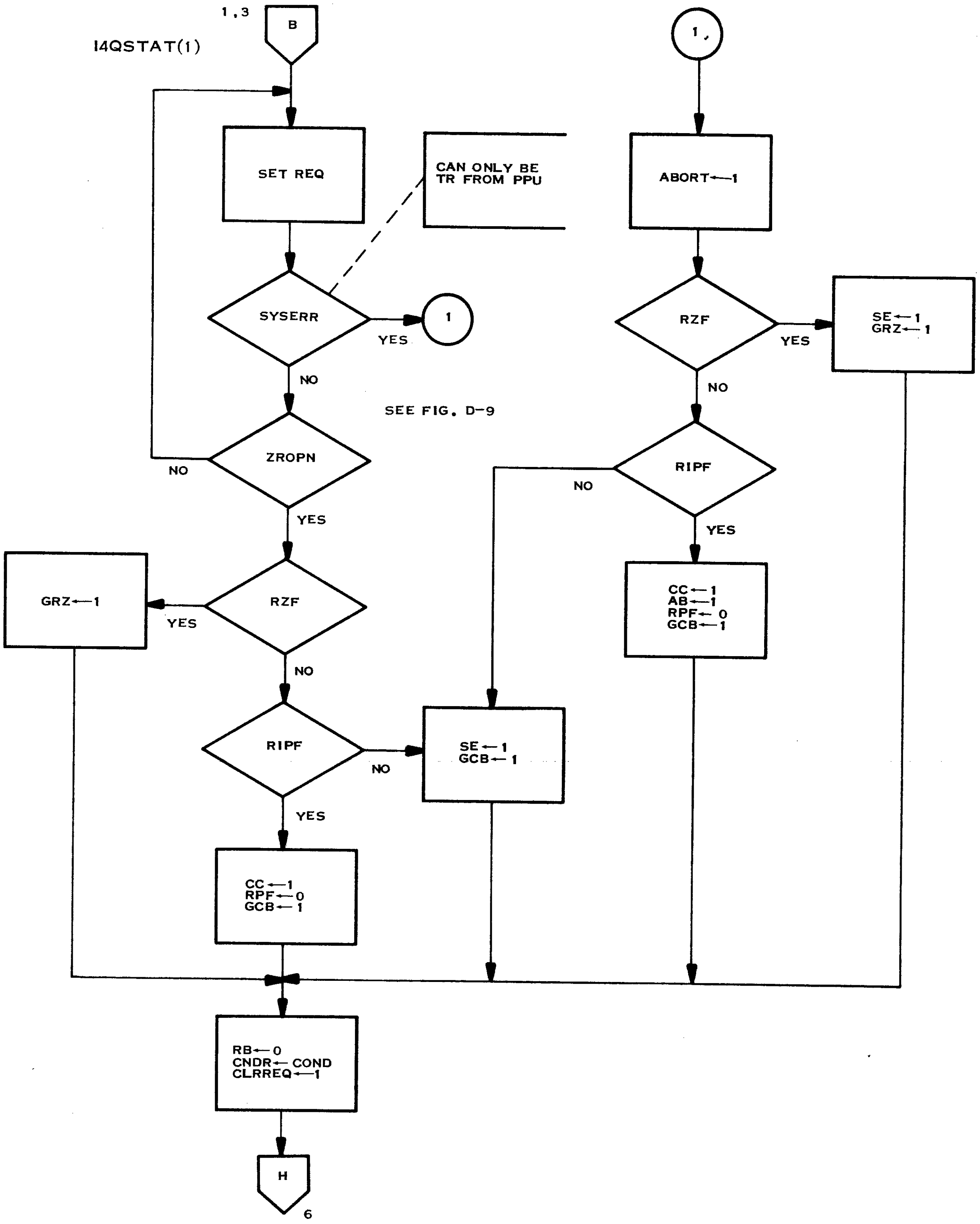


Figure D-5. MHC Program Error Cell Detection



(C)132487 (1/6)

Figure D-6. Master Hardcore Flowchart (Sheet 1 of 6)



(B)132487 (2/6)

Figure D-6. Master Hardcore Flowchart (Sheet 2 of 6)

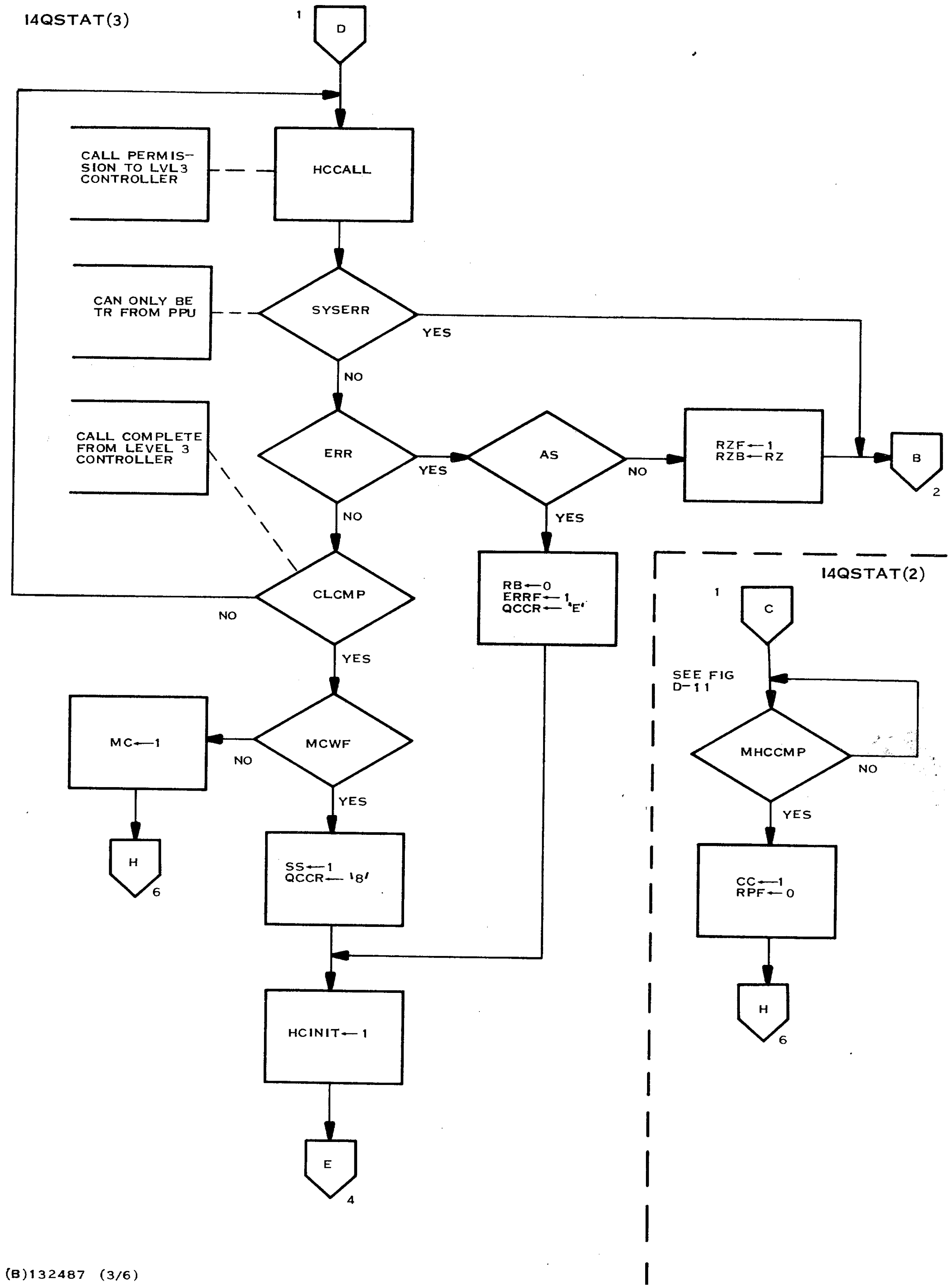
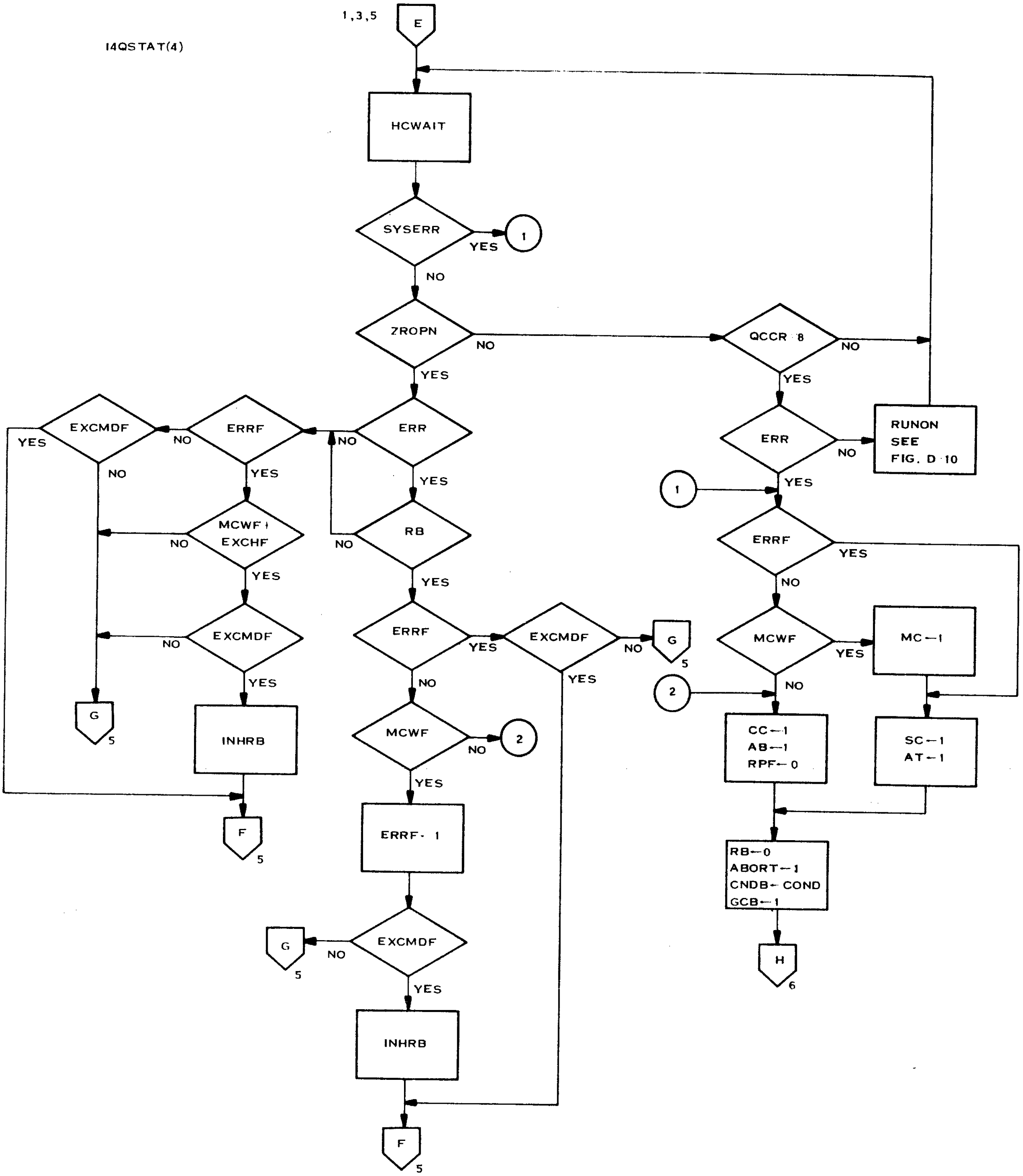


Figure D-6. Master Hardcore Flowchart (Sheet 3 of 6)



(C)132487 (4 6)

Figure D-6. Master Hardcore Flowchart (Sheet 4 of 6)

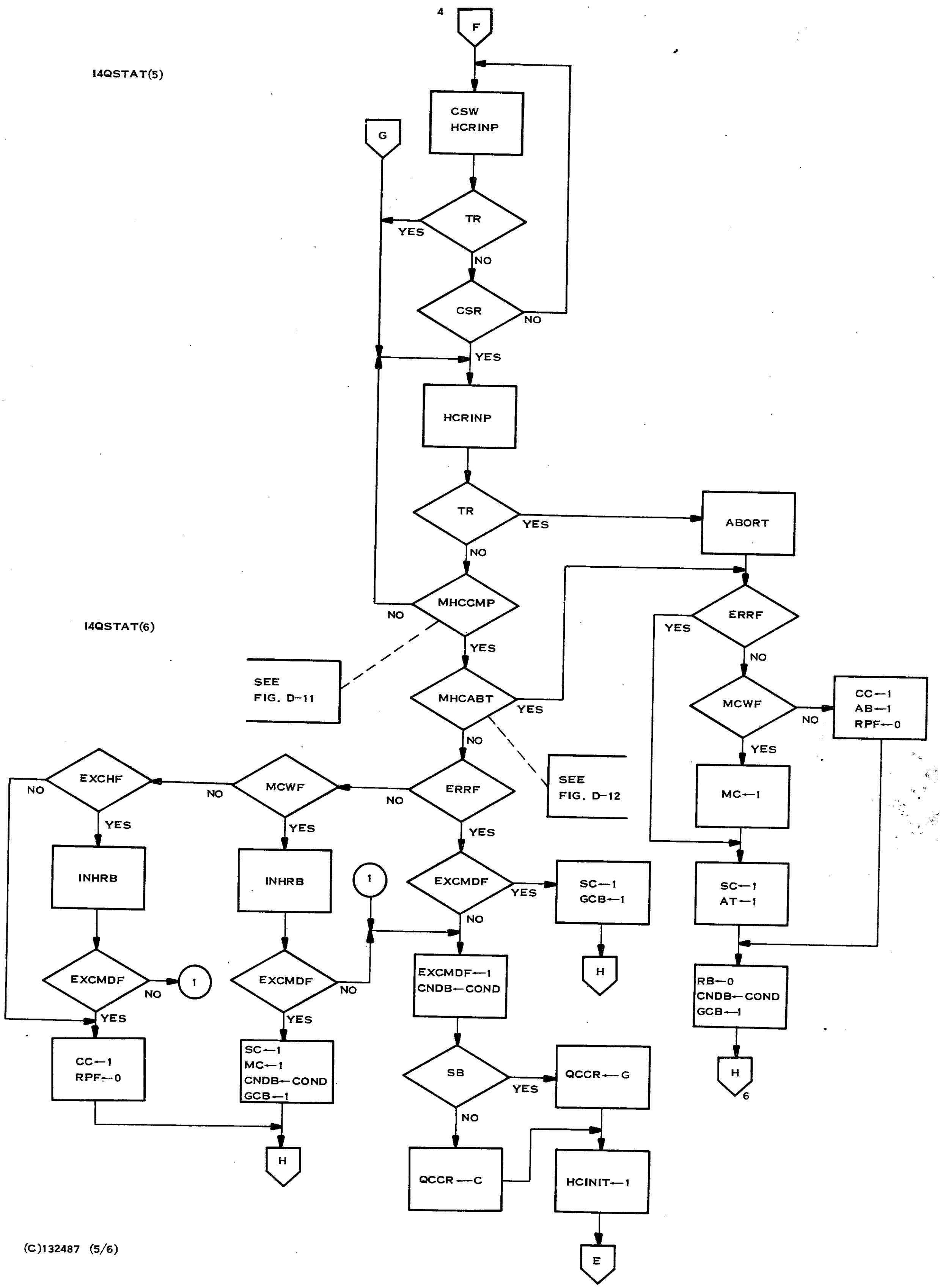


Figure D-6. Master Hardcore Flowchart (Sheet 5 of 6)

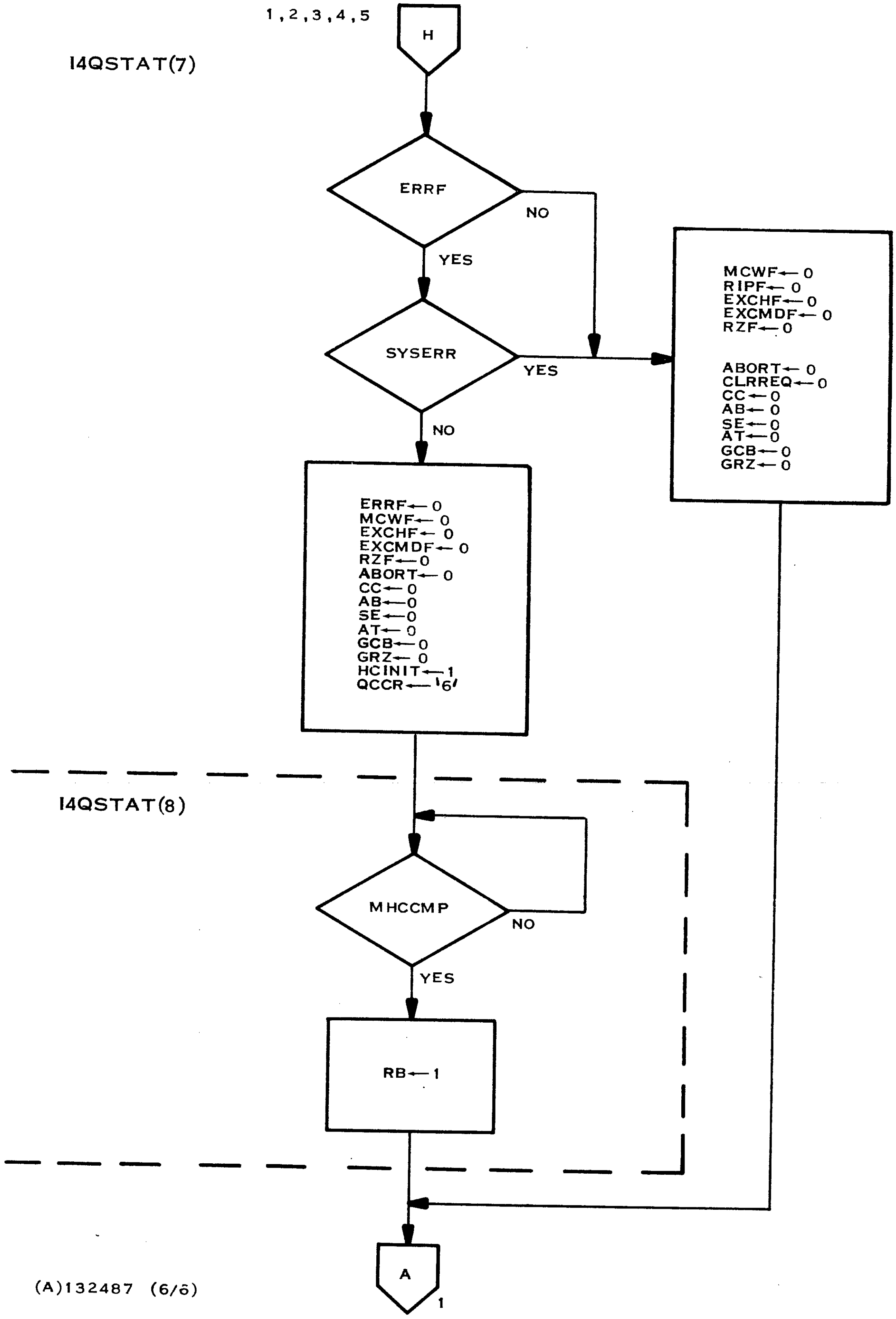
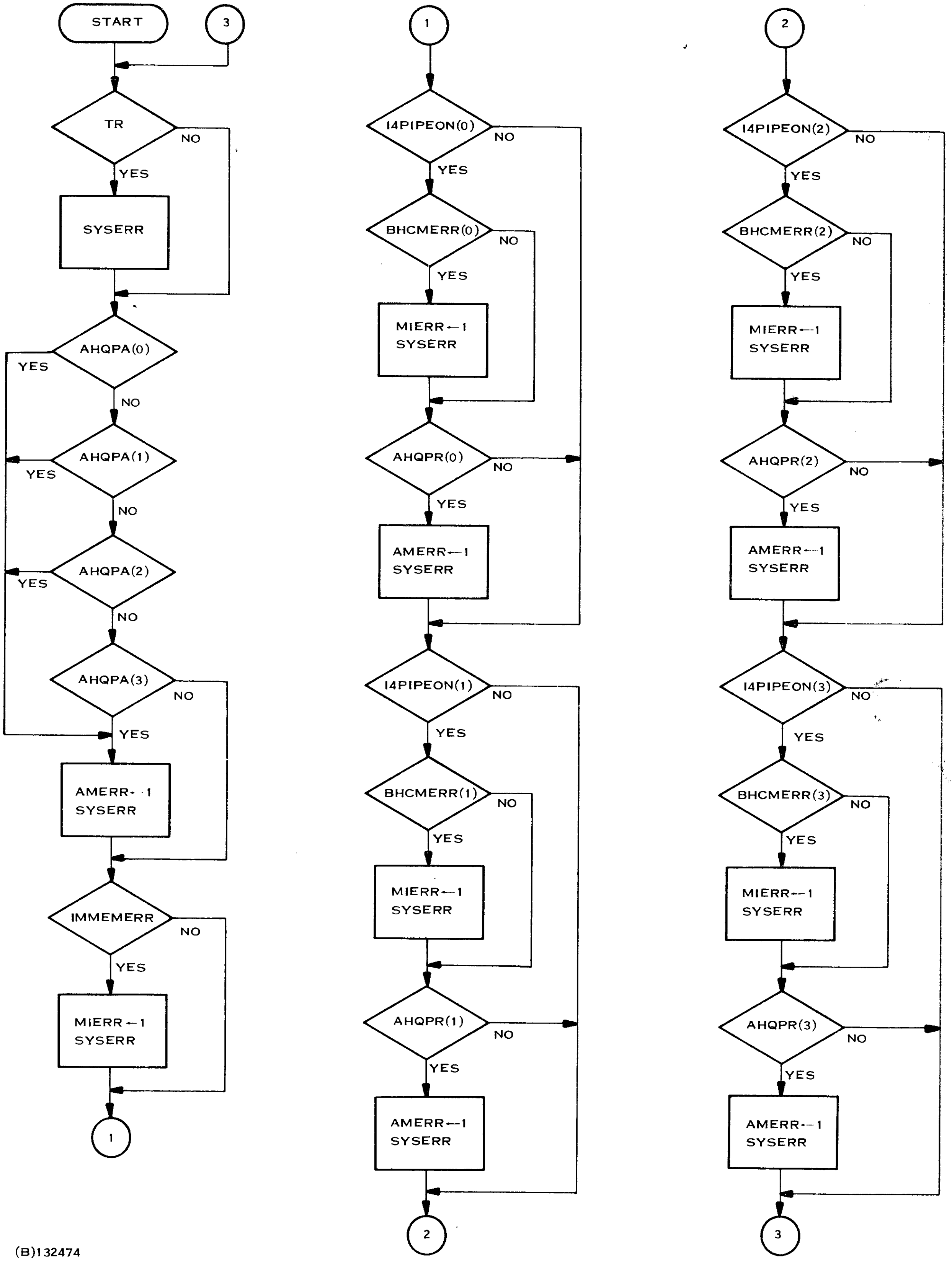


Figure D-6. Master Hardcore Flowchart (Sheet 6 of 6)



(B)132474

Figure D-7. MHC System Error Cell Detection Flowchart

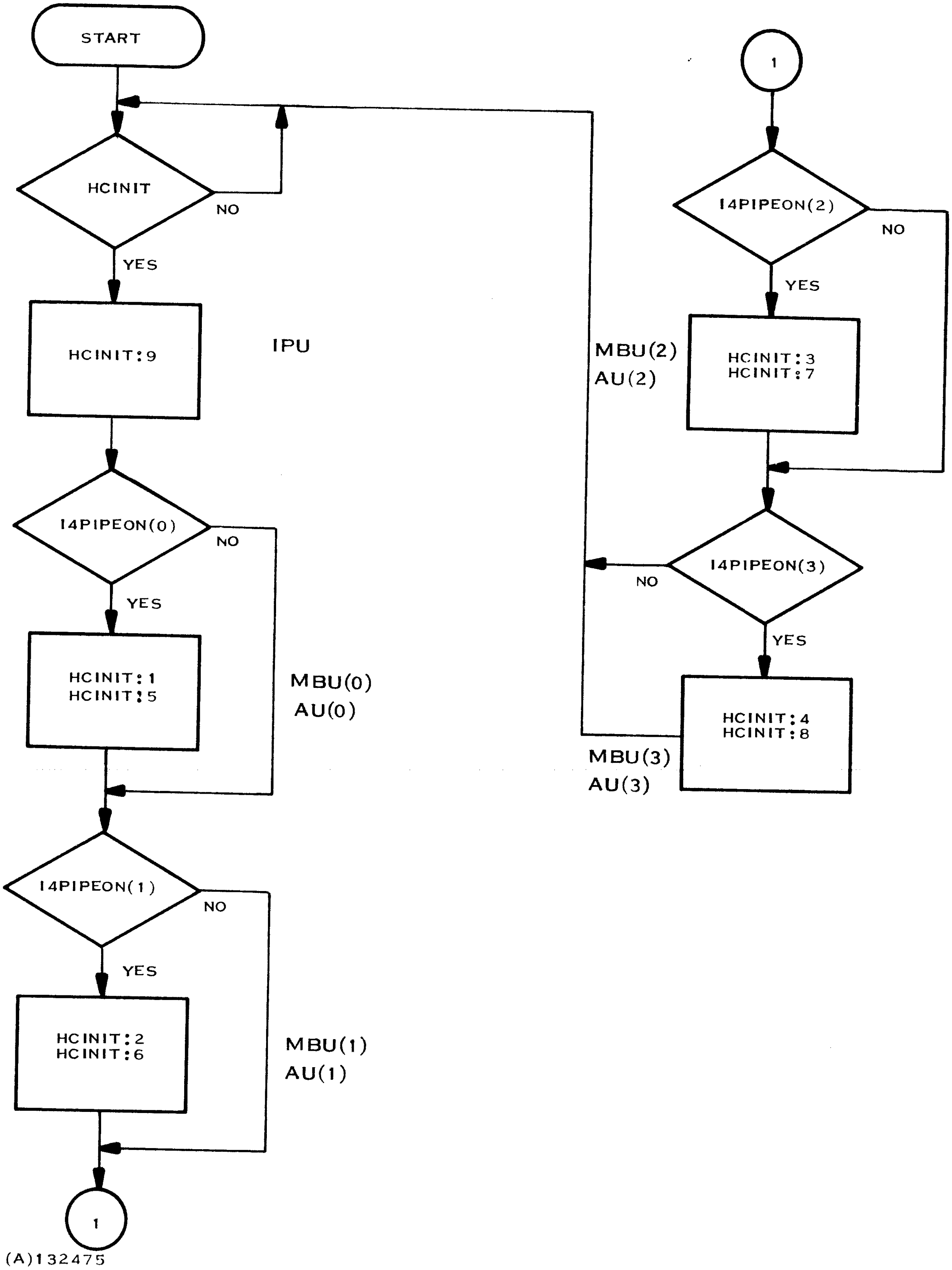


Figure D-8. MHC Initiating Unit Hardcores Flowchart

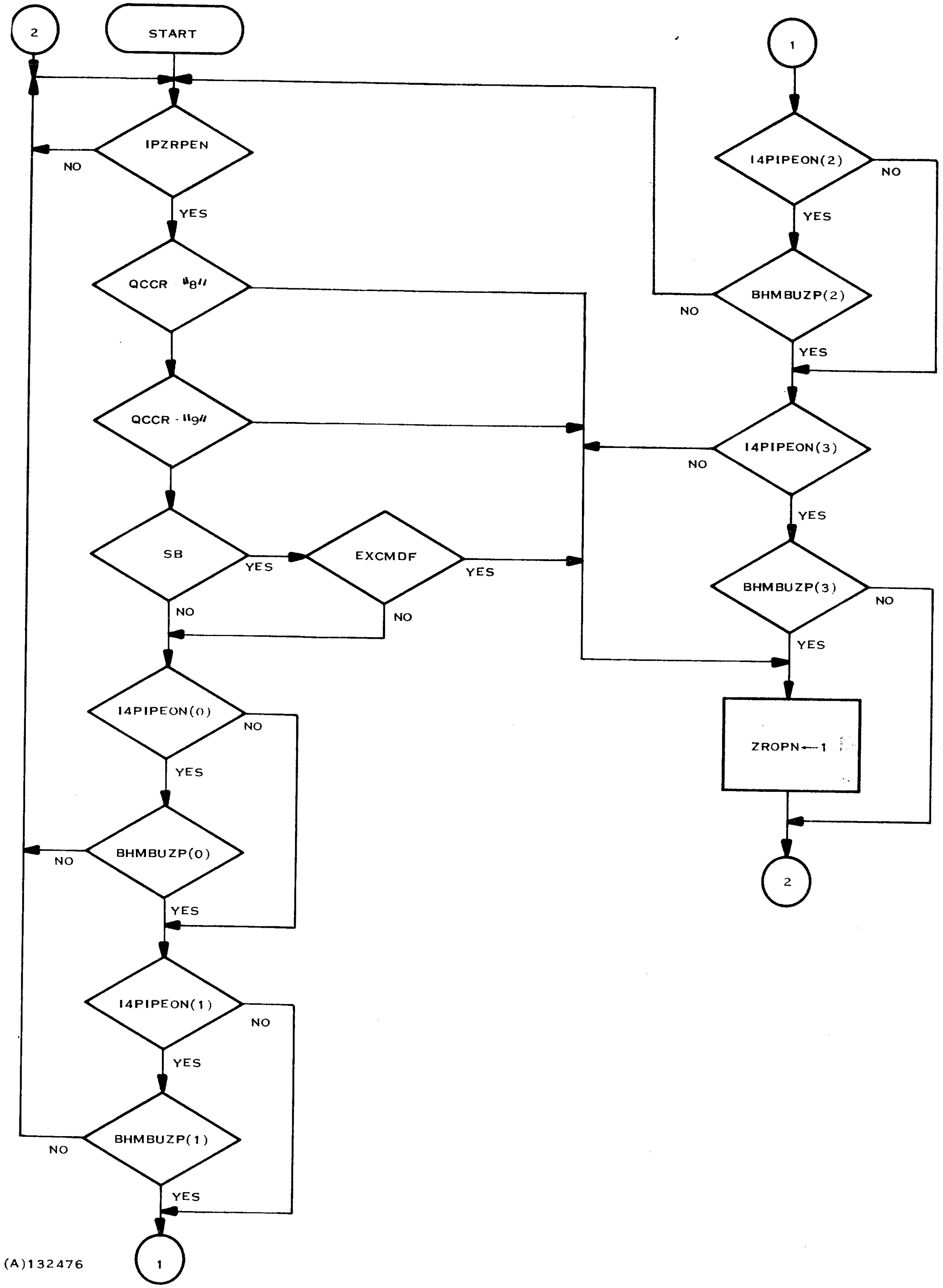
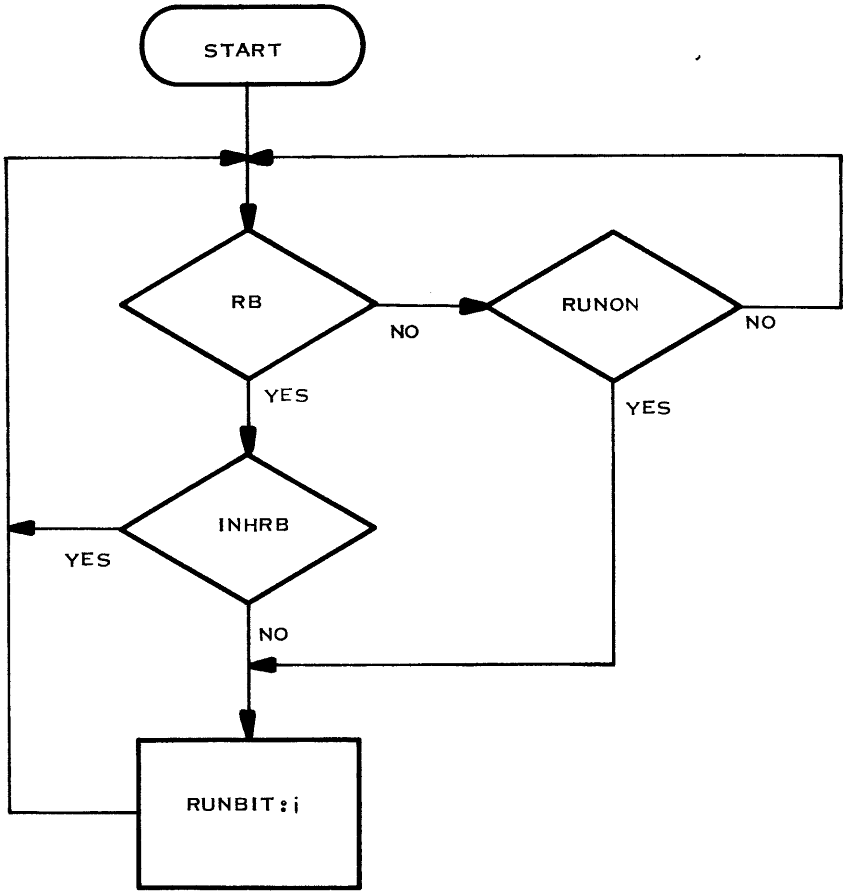


Figure D-9. MHC Zero Pending Memory Requests Detection Flowchart



14RUNBIT:i = COPY OF 14QRB SENT TO UNIT HARD CORES [UHC'S]
[14QRB=1 > UHCS' QFREEZ=0]

14INHRB = INHIBIT RUNBIT TO UHC'S
= [14QMCWF + 14QEXCHF} * [14QSTAT(6) + 14QEXCMDF]

14RUNON = FORCE 14RUNBIT:i = 1 FOR A 4108 CCR COMMAND UNTIL IPU-UHC RESPONDS WITH "ZERO PENDING"
= [14QSTAT(4)* {14QCCR(8-12) = "8"} * ¬ 14QZROPN]

(A)132477

Figure D-10. MHC Run Bit Control Flowchart

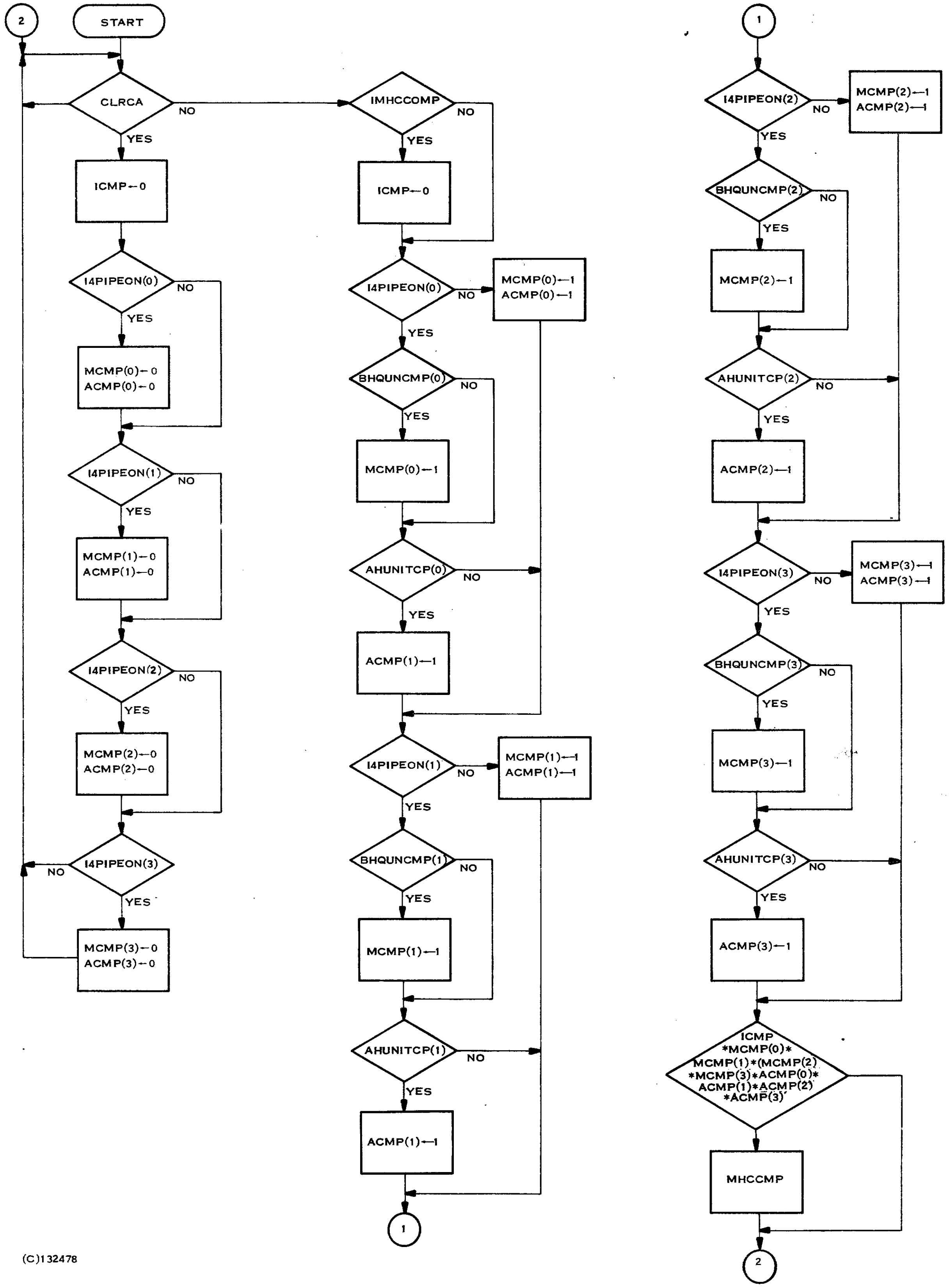


Figure D-11. MHC Unit Hardcore Complete Detect Flowchart

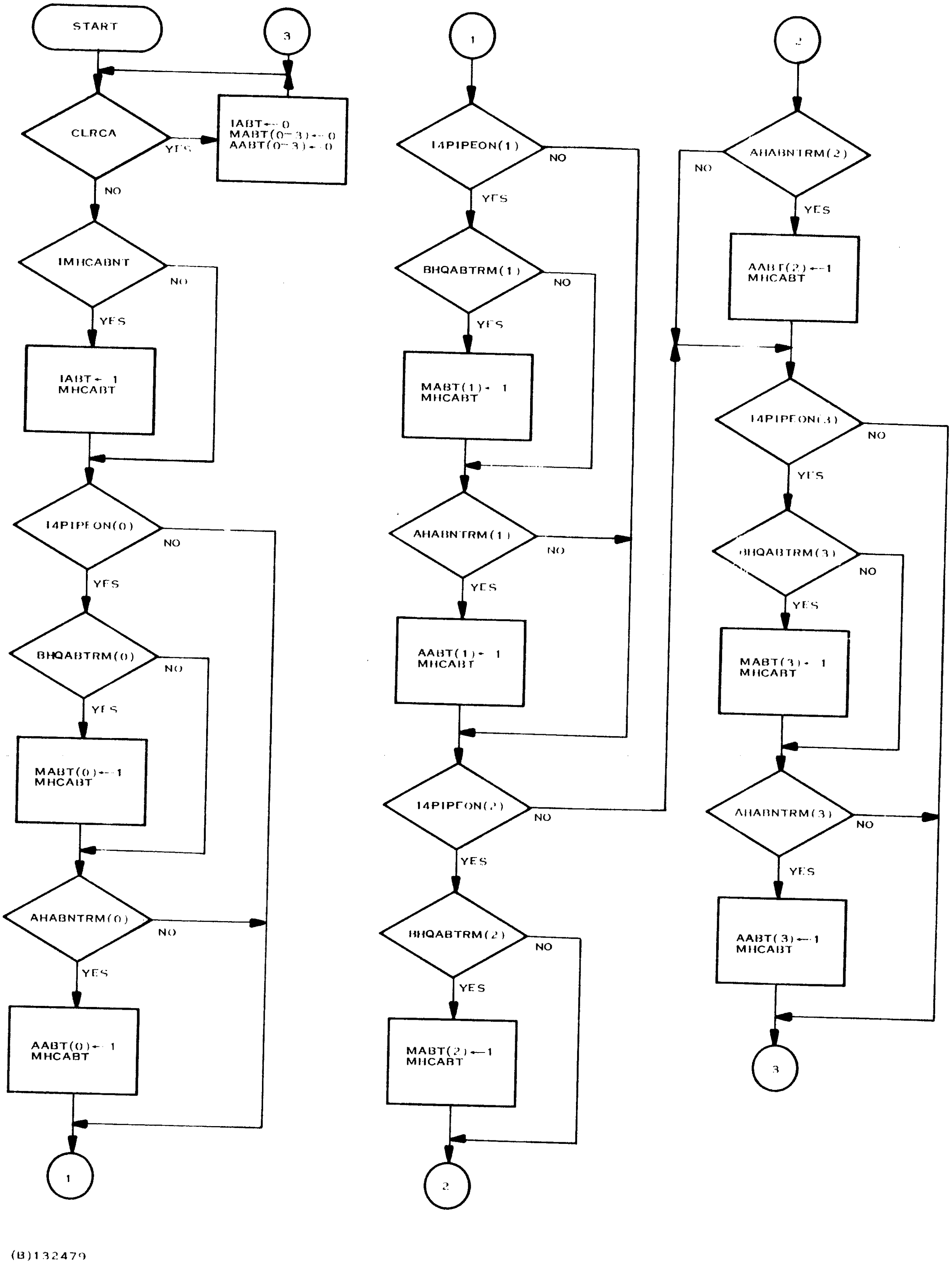
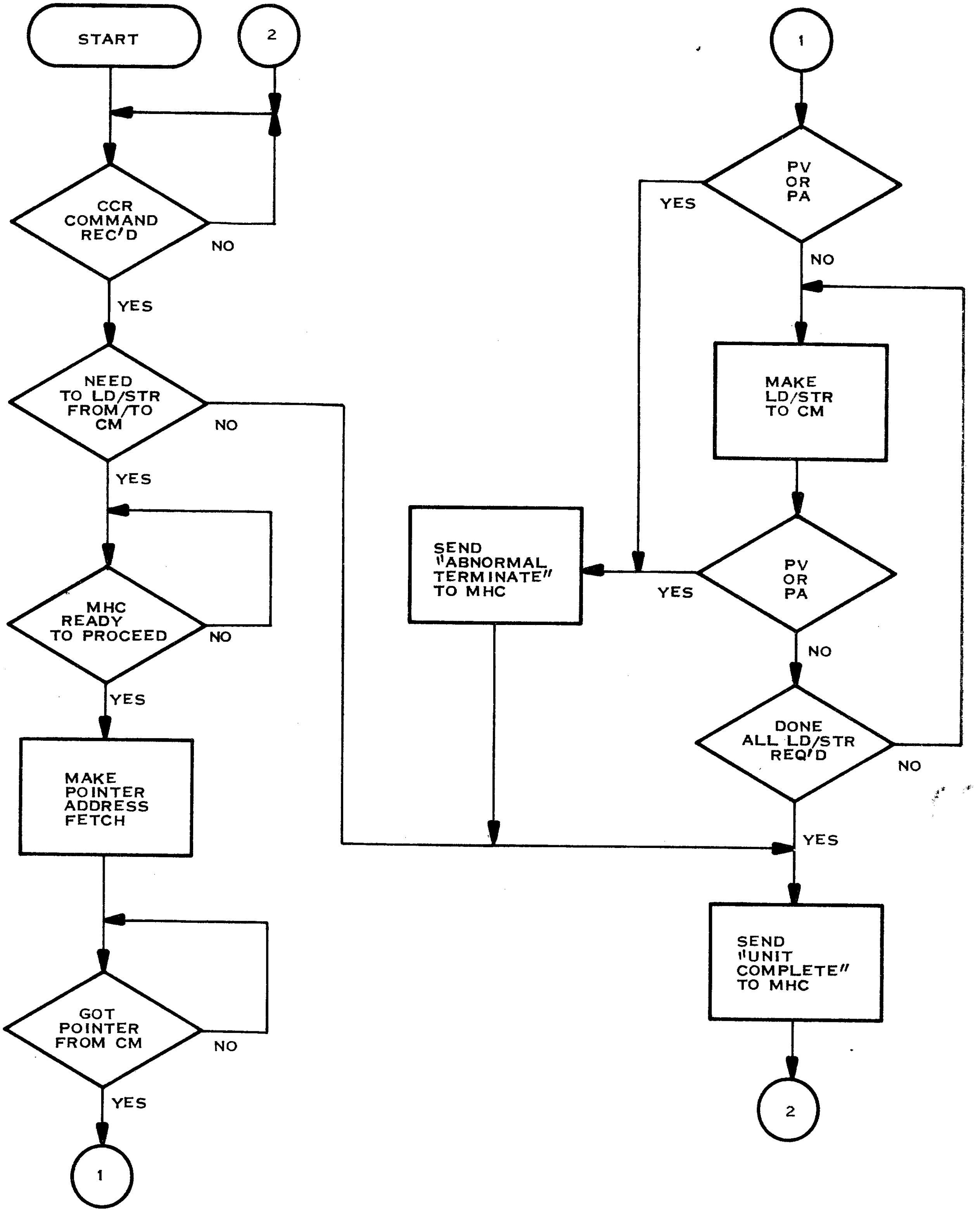


Figure D-12. MHC Unit Hardcore Abnormal Exit Detection



(A)132480

Figure D-13. Typical Unit Hardcore Operation Flowchart

The Master Hardcore may respond in two ways to the Program errors: (1) it may merely tell the PPU that an error has occurred and then wait for the PPU to indicate what is to follow (manual mode) or (2) it may automatically switch out the offending program and switch in a new program to be run (automatic mode).

Manual mode occurs when:

1.) The PPU has not responded to and cleared a previous code or switch condition (I4CRMC + I4CRSC = 1); or,

2.) The PPU has not set I4CRAS, the "Allow Automatic Switch" control bit.

In either case, the error condition is recoded into a reason error condition via the Reason Encoder logic, and the Reason code buffer (I4QRZ(0-2)) is gated to the PPU by I4QGRZ.

Automatic mode occurs when I4CRMC + I4CRSC = 0 and I4CRAS = 1. In this mode we turn off the machine "Run" mode bit (I4QRB) to halt the machine, do a Store Maintenance Details (410E CCR command) to swtich out the offending program, do a Load Status command (if I4CRSB = 1) or a Load CP Details to switch in the next job the PPU has cue'd up, send a "Clear Abnormal Flags" command to the Unit Hardcores to clear the error condition (since it has been serviced), and finally, turn the run bit back on (I4QRB←1), so that the new program may begin.

As may be seen from a casual glance at the flowcharts, system errors are serviced differently, depending upon when they occur. The overriding concern, however, is that the machine be halted (I4QRB←0), and the PPU notified that such a condition exists, so that the PPU may assume control and issue such commands as necessary to restore the CPU to a running condition again.

D-9 CPU MONITOR CALL SERVICING

There are two IPU Instructions that issue a "Call" to the PPU; that is, they halt the IPU for as long a time as necessary to either write a message only or to write a message and switch out the present program. These instructions are "Monitor Call and Proceed" and "Monitor Call and Wait" abbreviated MCP and MCW. The "Call" is sent to the Master Hardcore, which in turn determines how it will be serviced.

As in the case of Program error servicing, MCP/MCW instructions may be handled in either the manual mode or the automatic mode depending upon control conditions from the PPU.

The manual mode will occur if:

1. I4CRMC + I4CRSC = 1; or

2. I4CRAC = 0 (the PPU has not set the "Allow Automatic Call" control bit; or,

3. the instruction is MCW and I4CRAS = 1.

If the manual mode occurs, the Reason encoder logic recodes the MCP/MCW into a Reason error, and handles the situation exactly as described in the Program error - manual mode section.

The automatic mode occurs when all of the above manual mode conditions are not true; i.e., I4CRMC + I4CRSC + I4CRAC + I4CRAS = 0 for MCW instructions or I4CPMC + I4CRSC + I4CRAC = 0 for MCP instructions.

If the instruction is MCP, in the automatic mode, the Master Hardcore will send I4HCCALL to the IPU Level 3 Controller to allow it to write its message for the PPU. When the level 3 Controller has finished, it will send IRCALCMP to the Master Hardcore, when it is buffered in I4QCLCMP. The Master Hardcore then sets I4QMC to tell the PPU that a message is ready for reading. The PPU must then respond by sending I4CRMC to the Master Hardocre, so that I4QMC may be cleared. The Master Hardcore will not consider any further MCW/MCP requests or Error servicing requests to be in the Automatic mode until it has followed a complete cycle of changes on I4CRMC; i.e., I4CRMC = 0, then I4CRMC = 1 then I4CRMC = 0. If the instruction is MCP in the automatic mode, the Master Hardcore will send I4HCCALL to the IPU Level 3 Controller and wait for IRCALCMP. It will then issue a Store Status command to the Unit Hardcores to switch out the present program. When it completes normally, it will issue either a Load Program Status (if I4CRSB = 1) or a Load CP Detail to switch in the new program. When it completes normally, it will send both I4QMC and I4QSC to the PPU to tell it that a message is ready, and that a program switch has completed. The PPU must respond with a complete cycle on I4CRMC + I4CRSC before the Master Hardcore will consider any further MCP/MCW/Error service requests to be in the automatic mode.

If after an MCP/MCW Automatic mode service request begins another error occurs, the Master Hardcore will terminate the servicing at that point and tell the PPU what was in progress and why it failed via the CP Condition and CP Response bytes.

IPU4 UNIT HARD CORE CONTROLLER

The central processor Master Hard Core (MHC) logic transmits the four least significant bits of CCR commands to the IPU4 Unit Hard Core controller (controller). The controller then decodes those bits of the command and, if the command is pertinent for the IPU4, generates the gating and control signals required to perform the maintenance transfer. The controller is implemented entirely on the I4HDCORE circuit board in location LB of the I4CTLMB. One initial waiting state plus six clocked states, 1 through 6, comprise the functional divisions of the controller. The state diagram and flow charts included with this description illustrate the inter-relationship of these states. The following paragraphs describe the operation and function of the controller states for each maintenance command. The discussion follows the logic flow depicted in the flow charts.

STATE 0. This state is an asynchronous waiting state of the controller. Whenever the controller is not processing a CCR command for MHC, the controller returns to state 0 to await the next command. MHC activates Hard Core Initiate (I4HCINIT) to inform the controller of a command to be performed. When the controller detects this signal, it transfers the four least significant bits of the CCR code (I4CCR(12-15)) from MHC to the controller's command register, IMQLSD(0-3). When this transfer is complete, the controller enters state 1 following the control clock pulse.

COMMAND DECODE. When the controller has recognized and captured a CCR command, it enters state 1 to begin decoding and processing the indicated operation. Table D-1 lists the codes that the controller recognizes and responds to, and the operations that the codes initiate in the IPU4. Execution of No Op conditions and basic, one-action commands is handled entirely while the controller is in state 1. No Op conditions produce a command complete signal from the controller (-IMHCCOMP). The controller returns immediately to state 0 to await a new command. The basic commands of master reset, master preset, and clear abnormal flags produce their respective clear or set signals from the controller in state 1. The controller

Table D-1. IPU4 CCR Commands

I4CCR(12-15), or IMQLSD(0-3)	Command	Required Action
0-3	No Op	Return Command Complete to MHC
4	Reset Details Cells	Master Clear IPU4 registers and flags.
5	Set Details Cells	Master Preset IPU4 registers and flags.
6	Reset Error Cells	Clear IPU4 abnormal flags.
7	Illegal Op	Return Command Complete to MHC
8	Store Status	Fetch pointer from location 14, store program status word, P3, clock counter and Register File in pointer octet.
9	Load Status	Fetch pointer from location 15; load contents of pointer octet into program status word, P3, clock counter and Register File.
10 (A)	Exchange Status	Fetch pointer from location 14; store status in pointer octet. Fetch pointer from location 15; load status from pointer octet.
11 (B)	Store CP Details	Fetch pointer from location 18; store details parameters in octets beginning with pointer octet. Wind down operations in pipe before store.
12 (C)	Load CP Details	Fetch pointer from location 19; load details parameters from octets beginning with pointer octet.
13 (D)	Exchange CP Details	Fetch pointer from location 18; store details beginning with pointer octet. Select pointer from location 19; load details beginning with pointer octet. Wind down pipes before store.

Table D-1. IPU4 CCR Commands (Continued)

I4CCR(12-15), or IMQLSD(0-3)	Command	Required Action
14 (E)	Store Maint. Details	Fetch pointer from location 18; store details parameters in octets beginning with power octet. Freeze pipes before store.
15 (F)	Load Maint. Details	Fetch pointer from location 19; load details parameters from octets beginning with pointer octet.

then issues command complete to MHC and returns to state 0. One other condition, an abort from MHC, causes the controller to return to state 0 from state 1. This condition indicates that one of the other CP unit hard core controllers has failed to perform its function in the transfer command. Completion of the command by the IPU4 would be fruitless. The controller then sets abnormal termination to MHC, reinitializes the octet counter, clears the hard core request indicator to CMR, and signals command complete to MHC. The controller then returns to state 0. All other commands involve more complex control operations. To handle these operations, the controller enters the other states. The remaining paragraphs describe the control paths of each of these complex transfer operations.

STATUS OPERATIONS. A status transfer (load, store or exchange) is the minimum maintenance transfer that the CP performs. A status command transfers six complete octets and one partial octet either to or from an area in memory from or to the register file, the clock counter, the P3 register, and the status word register in the IPU4. Figure D-14 illustrates the octet and word allocations for each of the data quantities in the status transfer. A counter circuit (IMQOCTR(0-4)) in the unit hard core controller determines which octet is to be involved in the transfer during any particular clock cycle. This counter is initially loaded with a value of 20 to select the first octet in the status map, and then with a value of 14 to select the second octet in the status map. Subsequent memory cycles increment the counter from 14 through 19 so that all of the octets in the status map are selected. The location of the status map in memory for a particular transfer is indicated by the contents of another memory word in either location 14 (store) or 15 (load). The actual contents (pointer) of either of these memory locations is under control of the operating system, so that the location of the map in memory may be changed for different jobs that are running in the CP. To access the pointer from memory, the controller loads the octet address of the pointer (10) into the 0A register and requests that octet from memory. The controller sets three selection bits, IMHCASEL(0-2), to the values that gate the

OCTET	IMQOCTR 0 1 2 3 4	WORD 0	WORD 1	WORD 2	WORD 3	WORD 4	WORD 5	WORD 6	WORD 7
0	(20) 10100	IRQPSW (0–31)	IRQP3 (8–31)	CLOCK COUNTER					
1	(14) 01110	ZEROES	REGISTER FILE A IFQA 1–7 (0–31)						
2	(15) 01111	REGISTER FILE B IFQB 0–7 (0–31)							
3	(16) 10000	REGISTER FILE C IFQC 0–7 (0–31)							
4	(17) 10001	REGISTER FILE D IFQD 0–7 (0–31)							
5	(18) 10010	REGISTER FILE I IFQI 0–7 (0–31)							
6	(19) 10011	REGISTER FILE V IFQV 0–7 (0–31)							

(A)127624

Figure D-14. IPU4 Status Map

proper pointer from the KCM octet to the 0A register to begin the status transfer. Store status transfers travel through the IMDTL bus lines to the I4FILE circuit boards to be placed on the two way data bus to memory. Load details transfers enter the IPU at the KCM interface file and are transferred directly from KCM to their respective registers over the ILQKCM data bus lines.

STORE STATUS. A code of eight indicates a store status operation. When the controller detects this code in IMQLSD(0-3), it sets the octet counter to a value of 20 (10100). When enabled, this count transfers the program status word, P3 and the clock counter contents to word 0 of the status map area in memory. The controller then sends Status Freeze to the level 3 controller, so that the level 3 controller will hold the next instruction at level 3 and empty the pipes. When all pipes are empty and an unprocessed instruction is at level 3, the level 3 controller generates -IRNIFRZ to the hard core controller. When the controller receives this signal, and if all other unit hard core controllers in the CP have not encountered complications, the IPU4 is prepared to perform a store status operation. The controller then sets -IMHCREQ to central memory requester (CMR) indicating that the controller will be using CMR to initiate requests to memory. The controller then enters state 2.

In state 2 during a store status, the controller checks each CP pipe to determine if a vector is running in that pipe that cannot be wound down without further memory requests (vector bad guy). If a vector bad guy is in any of the pipes, the controller signals the level 3 controller that the vector will be aborted, and then forces the vector out of the pipe without an orderly de-escalation. The controller also ensures that de-escalation is complete in each pipe. If not and a vector is still in progress, the controller activates -IMSUPRUN(X) so that the current buffered data in the MBU will run to completion and stop. The controller waits until all pipes have cleared. In state 2 the controller also ensures that no memory requests are outstanding or expected, so that the outstanding requests will not be lost when the status store is performed. The controller indicates to MHC that it is ready to perform the command by issuing zero pending (IPZROPEN). When the MHC responds with I4ZROPEN, the controller locks the IPU4 in its present state and sets the hard core in progress flag (IMQHCINP). The controller enters state 3.

During a store status in state 3, the controller sets bit 27 of the OA register. Setting this bit loads octet address 10 into the OA register. This octet is the octet that contains the pointer for status operations. The controller initiates a memory request for this octet through CMR, and enters state 4.

In state 4 the controller waits for the requested octet to return from memory. It then checks for a parity error or protect violation during the fetch for the pointer octet. If either of these conditions occur, the controller terminates the operation and indicates to the MHC that the operation is complete and abnormally terminated. If no errors occur, the controller sets the Load Pointer flag to indicate that the pointer must be selected from the octet in KCM, and advances to state 5.

When the controller enters state 5 from state 4, the write flag sets. This flag indicates to the memory bus selection circuit on the I4FILE circuit boards that the operation will be a store into memory. The selection circuit then enables data from the store details lines to the MCU, The controller also activates the store program status doubleword flag that indicates the first status octet is being stored. The controller then sets pointer select bit 0 of IMHCASEL(0-2), so that word four of the pointer octet (location 14) is transferred into OA to designate the starting octet for the status store into memory. The controller toggles access request (ICQAR:2) to the MCU to begin the store operation for the first octet, and clears the load pointer flag so that subsequent passes through state 5 will not reselect a pointer from KCM. The controller enters state 6.

During the initial pass through state 6, the octet counter has been set to a value of 20. The controller enables status inputs to the memory bus by producing IMSTPSDW:1 and IMSTPSDW:2. When the store has been made, the controller ensures that no memory errors have occurred. If an error occurs, the controller performs the abnormal termination cycle. If no errors occur, the controller clears the store status flag, and the controller loads a value of 14 into the octet counter. Loading 14 into the octet counter begins a cycle for storing the contents of the register file into memory. To initiate the memory cycle for the first octet of the register file, the controller returns to state 5.

After the memory cycle for the first register file octet has been initiated, the controller returns to state 6, checks for a memory error, and if no errors have occurred, increments the octet counter and returns to state 5 to initiate another store operation for the next register file octet. This cycle repeats until the octet counter reaches 19 and the octet corresponding to that count (V file) has been stored. At that point the store status operation is complete. The controller clears the octet counter and the hard core request indication to CMR, and terminates the operation by sending-IMHCCOMP to MHC. The controller returns to state 0 to await another command.

LOAD STATUS. A code of 9 in IMQLSD(0-3) indicates a load status operation. When the controller detects this code, it sets the details octet counter to a value of 20 (10100) so that the status word, P3 and the clock counter will be selected first during the load operation. The controller then sets -IMHCREQ to CMR indicating that the controller will be using the memory access paths for a maintenance transfer. The controller then enters state 2.

For a load status, state 2 of the controller does not need to check the contents of the CP pipes as in the store status, since nothing in the pipes will be saved. Instead the controller ensures that the IPU has no outstanding requests to memory and expects no responses from the MCU. Then, if the other unit hard core controllers have not encountered difficulty, the controller indicates to MHC that it is ready to perform the transfer. When MHC responds with I4ZROPEN, the controller locks the IPU4 in its present state, sets the hard core in progress flag, and enters state 3.

In state 3 the controller generates a master clear to the IPU4 registers and flags to prepare the circuits for the load operation and ensure that all previous information is erased. The controller then loads a value of 10 into the OA register by setting bit 27, and initiates a memory request for the status pointer octet.

When the octet enters KCM, the controller checks for memory errors, and if none have occurred, sets the load pointer flag to indicate that the pointer must be selected from the KCM octet and loaded into OA to fetch the status parameters from memory. The controller enters state 5.

The pointer must be selected from KCM to continue with the load status operation. State 5 sets both bits 0 and 2 of IMHCASEL(0-2) so that word 5 of the octet in KCM (location 15) will be selected and transferred to OA. The controller generates the required transfer signals, clears the load pointer flag, and then initiates a memory request for the first status octet by toggling AR to the MCU. The controller then moves to state 6.

In state 6 the controller checks the value of the octet counter. Since at the start of the load status operation (state 1) the counter was loaded with a value of 20, the controller generates IMLDPSDW:1 and IMLDPSDW:2 to enable the inputs to the status word register, P3 and the clock counter from their respective words in the KCM file. When the octet enters KCM from memory, the controller ensures that no memory errors have been encountered, and loads a new value of 14 into the counter so that the first octet of the register file will be loaded during the next cycle. In order to fetch the next octet from memory, the controller adds eight to the address in OA, and toggles AR to initiate a memory request for that next octet. When each successive octet returns from memory into KCM, the new value in the octet counter gates that octet to one of the octets in the register file. Concurrently, the controller checks for memory errors, increments the count in the octet counter, and initiates a new memory request for the next octet (OA + 8). This cycle repeats until the octet counter reaches 19 and the octet corresponding to that count (V file) has been loaded. At that point the load status operation is complete. The controller clears the octet counter and the hard core request indication to CMR, and terminates the operation by sending -IMHCCOMP to MHC. The controller returns to state 0 to await a new command.

EXCHANGE STATUS. A code of 10 in the IMQLSD(0-3) register indicates an exchange status operation. The exchange status command produces a store status operation followed by a load status operation, and follows the paths through the controller logic described for those operations. The exchange flag, IMQEXCH, indicates which cycle is being performed during the exhange. This flag sets at the start of the exchange operation to indicate to the controller that the store status portion of the exchange operation is in progress. When the store status portion is complete, controller state 6 clears the exchange flag so that the controller will follow the load status paths.

DETAILS OPERATIONS. A details transfer (load, store or exchange) switches the contents of all the vital registers in the IPU to preserve the parameters of a program executing in the IPU, or enter new program parameters into the IPU. All details operations transfer twenty octets of information between memory and the registers and flags of the IPU. Appendix A to this description illustrates the bit, word and octet assignments of the critical registers and flags of the IPU. A counter circuit (IMQOCTR(0-4)) in the unit hard core controller determines which octet is to be involved in the transfer during any particular clock cycle. Since twenty octets are involved, the counter must increment from an initial count of zero through a count of nineteen to enable transfer of all octets in the details map. The location of the details map in memory for a particular transfer is indicated by the contents of another memory word in either location 18 (store) or 19 (load). The actual contents (pointer) of either of these memory locations is under control of the operating system, so that the location of the map in memory may be changed for different jobs that are running in the CP. To access the pointer from memory, the controller loads the octet address of the pointer (18) into the OA register and requests that octet from memory. The controller sets three selection bits, IMHCASEL(0-2), to the values that gate the pointer from KCM to the OA register to begin the details transfer. Store cycle data travels through the IMDTL bus lines to the I4FILE circuit boards to be placed on the two way data bus to memory. Load cycle data travels through KCM to the respective registers and flags in the IPU.

STORE CP DETAILS. A code of 11 in IMQLSD(0-3) indicates a store CP details command. When the controller detects this code, it clears the octet counter to zero to ensure that the transfer starts with octet zero, and indicates to CMR that it will be using the memory access paths to perform a maintenance transfer. The controller then enters state 2.

During a store CP details, state 2 of the controller checks each CP pipe to determine if a vector is in any of the pipes that cannot be wound down without further memory fetches (vector bad guy). If a vector bad guy is in any of the pipes, the controller indicates to the level 3 controller that the vector will be forced from the pipe without an orderly wind-up, and sends an abort signal (IMBGABRT) to the respective MBU to irradicate the vector. If no vector bad guy is present, but another vector is in the pipe that hasn't been de-escalated, the controller issues IMSUPRUN to the respective MBU to de-escalate the vector regardless of the condition of the CP Run bit. When de-escalation is complete in all pipes, the controller ensures that the IPU has no outstanding requests to memory, and that a request has not been issued during the current clock. When the controller is sure that no memory requests will be lost, it informs MHC that zero requests are pending (IPZROPEN) so that the IPU is prepared to execute the command. When MHC returns I4ZROPEN, the controller locks the IPU in its current state, sets the hard core in progress flag, and transfers control to state 3.

In state 3 the controller loads a value of 18 into OA by setting bits 27 and 28 of that register. The controller then initiates a memory request for the pointer octet, and enters state 4.

When the octet enters KCM, the controller checks for memory errors, and if none have occurred, sets the load pointer flag to indicate that the pointer must be selected from the KCM octet and loaded into OA to provide the starting address of the store operation in memory. The controller enters state 5.

When the controller enters state 5 from state 4, the write flag sets. This flag indicates to the memory bus selection circuit on the I4FILE circuit boards that the operation will be a store into memory. That selection circuit then enables data from the store details lines to the MCU. The controller leaves all of the IMHCASEL(0-2) bits clear to select word zero from the octet in KCM (location 18). This pointer word transfers to OA and the controller toggles AR to the MCU to initiate the first store operation of the transfer. As the controller exits from state 5 to state 6, it clears the load pointer flag so that subsequent passes through state 5 will not reselect a pointer from KCM.

When the controller enters state 6, the octet counter is at count zero since the counter was cleared at the beginning of the store CP details operation (state 1). The controller disables the output selection from the register file to avoid storing those parameters at this time, transmits the octet count to the individual details gating circuits on the IPU circuit boards, and enables those gating circuits to supply inputs to the data bus to the MCU. Decode of the octet count and selection of the corresponding input bits to the data bus is performed on each IPU4 circuit board. When the store cycle is complete, the controller checks for memory errors during the cycle, and if none have occurred, it increments the count in the octet counter and returns to state 5 to initiate another store cycle to memory. When the octet count reaches 12 or 13 (KA or KB Buffer files), IMFSLDIS deactivates since the storage path for KA and KB is through the register file output selection gate. When the octet counter reaches 19 and the octet corresponding to that count has been stored (register file V), the store CP details operation is complete. The controller clears the count in the octet counter and the hard core request flag, and transmits a command complete signal to MHC. The controller then returns to state 0 to await another command from MHC.

LOAD CP DETAILS. A code of 12 in IMQLSD(0-3) indicates a load CP details command to the IPU. When the controller detects this code, it clears the octet counter to an initial value of zero and sets the hard core request flag to indicate to CMR that unit hard core will be using the memory access paths. The controller then enters state 2.

During a load CP details operation, the controller does not need to check operational status in the pipes as it does during a store operation. The contents of any pipe will be destroyed by the load process if the pipes are not empty. Therefore, when the controller enters state 2 it checks only to see if the IPU has any outstanding memory requests that will return before the operation begins. Such data may be confused with the pointer octet, and produce errors. When the controller is sure that no requests have been sent to memory that have not returned, it transmits IPZROPEN to MHC to indicate "zero pending requests". When MHC responds with I4ZROPEN, the controller locks the IPU in its current state and sets the hard core in progress flag. Control transfers to state 3.

In state 3 of the controller loads a value of 18 into OA by setting bits 27 and 28 of that register. The controller then initiates a memory request for the pointer octet, and enters status 4.

When the octet enters KCM, the controller checks for memory errors, and if none have occurred, sets the load pointer flag to indicate that the pointer must be selected from the KCM octet and loaded into OA to provide the starting address of the store operation in memory. The controller enters state 5.

In state 5 the controller sets bit 2 of IMHCASEL(0-2) to select word one from the octet in KCM (location 19), and transfers that word to OA fetch the first octet of data from memory. The controller toggles AR to the MCU to initiate the memory request for that data, and clears the load pointer flag so that the pointer will not be reselected during repeated cycles through state 5. Control transfers to state 6.

Since the octet counter was cleared at the start of the load operation, the value in the counter is zero. Therefore, when the controller enters state 6, it transmits the contents of the counter to the separate IPU details gating circuits and enables those circuits to produce selection signals that route the incoming data from KCM to the proper IPU registers and flags. When the octet enters KCM from memory, the controller checks for any memory errors. If none have occurred during the memory cycle, the controller increments the count in the octet counter, adds eight to the address in OA, and toggles AR to the MCU to initiate a new memory fetch. The controller continues to cycle in state 6 until the octet counter reaches 19. At that point the load operation is complete. The controller clears the octet counter and the hard core request flag, and indicates to MHC that the command is complete. The controller returns to state 0 to await a new command.

EXCHANGE CP DETAILS. A code of 13 in IMQLSD(0-3) indicates an exchange CP details operation. The exchange CP details command produces a store CP details followed by a load CP details operation, and follows the paths through the controller logic described for those operations. The exchange flag, IMQEXCH, indicates which cycle is being performed during the exchange. This flag sets at the start of the exchange operation to indicate to the controller that the store CP details portion of the exchange operation is in progress. When the store portion is complete, controller state 6 clears the exchange flag so that the controller will follow the load status paths.

STORE MAINTENANCE DETAILS. A code of 14 in IMQLSD(0-3) indicates a store maintenance details command. The execution of this command is identical to the store CP details command, except that the maintenance details operation does not check the status of the pipes before storing the IPU parameters. Instead it freezes the IPU in its current state as long as no requests are outstanding to the MCU. This difference occurs in state 2 of the controller. All oth϶r controller paths are identical to the CP details controller paths. This command is used strictly for maintenance, and assumes that due to some IPU difficulty, the data in the pipes should not be wound to completion. Therefore, the current status is stored of each of the IPU parameters so that the maintenance technician may examine the parameters to determine the source of the error.

LOAD MAINTENANCE DETAILS. A code of 15 in IMQLSD(0-3) indicates a load maintenance details command. Execution of this command in the unit hard core controller is identical to execution of the load CP details command.

OUTSTANDING REQUEST COUNTER

This counter circuit keeps track of the number of outstanding read requests to memory for use by the unit hard core controller. When the controller is performing a read operation (-ICWRIT) and the MCU has returned the RA (request accepted) signal (ICRQCMP), this circuit checks to see if data is available from the MCU.

ABNORMAL FLAGS

The abnormal flags are four flip-flops. Each flip-flop sets whenever its particular abnormal condition occurs and is detected by the IPU circuits. The output from the circuit is available to the PP for inspection through the UR fanin circuit, and can also be stored into memory during a maintenance operation. The maintenance path to memory is enabled on the I4CMREQ circuit board. Three of the flags respond to condition detection circuit that are not on the I4HDCORE circuit board. These flags are: arithmetic exception (ICQAREX) set by -IRARTHEX from the I4STATUS circuit board, IPU illegal op code (ICQIPIOP) set by IRSETIOP from the I4LVL3 circuit board, and IPU parity error (ICQIPPAE) which responds to IOPA from the MCU through the MCU interface circuit.

The memory protect violation flag (ICQIPPRV) also responds to a signal from an error detecting circuit not contained on I4HDCORE (IRSETPRV from I4LVL3). However, an additional circuit on the I4HDCORE circuit board detects another protect violation that occurs during instruction fetching for the

IR register or the register file. The four signals that comprise the two error conditions originate on the I4CMREQ circuit board. ICIR indicates an octet in KCM is intended for IR; ICFL indicates an octet in KCM is intended for the register file. If either of these signals is active and the protect violation bit is set for the corresponding queue location (ICPRVO), the protect violation flag sets when the output pointer is incremented to the next value (ICINCOP).

One additional input to the flag circuit allows the flags to be loaded with a value from the details map in memory during a maintenance operation. This path from KCM (ILQKCM7(4-7)) is enabled during the maintenance operation when the octet counter indicates that the 11th octet of the details map is in KCM (ICLD0 (11)). Since this is the only circuit on I4HDCORE that is part of the details map, the actual decode of the octet counter count is performed on the I4CMREQ circuit board and the gating signal is forwarded to I4HDCORE. Refer to the description of the I4CMREQ circuit board for a description of the details gating circuit.

UNIT REGISTER (UR) DECODE

The UR decode circuit receives a four bit select code (I4UREN(0-3)) from the master hard core controller on the I4MHC circuit board. The circuit decodes the four bits to produce five gating signals to the UR Data Selection and Fanin circuit. In addition, the circuit forwards the selection code bits to the I4CMREQ circuit board to select the UR data from that board (IMUREN(0-3)). The four bit select code decodes as follows to produce five selection bits that return the indicated UR data to master hard core:

I4UREN(0-3)	Output Signal
1000	IHENDTUR(0)
1001	IHENDTUR(1)
1010	IHENDTUR(2)
1011	IHENDTUR(3)
1100	IHENDTUR(4)
1101	IHENDTUR(5)

UNIT REGISTER DATA SELECTION AND FANIN

This selection circuit receives the signals generated by the UR Decode circuit and uses them to produce one of six possible input bytes (8 bits) from the registers and flags on the I4HDCORE circuit board. If none of the input selection bits is active, the UR data input from I4CMREQ is enabled to place data on the UR data bus (-IMURDATA(0-7)) for transfer to the CP Unit Register in the PP. Figure D-15 illustrates the bit assignments for the six unit register bytes that originate on this circuit board. Refer to the description of the I4CMREQ circuit board for the bit assignments on the -ICURDATA(0-7) bus.

LEVEL 1 INSTRUCTION DECODE

The level 1 instruction decode circuit receives the operation code (op code) portion of the instruction in the IR register at level 1 (IIQIR(0-7)). The circuit decodes this eight bit number into two hexadecimal digits and uses the combination of digits to specify whether or not certain instructions or types of instructions are currently at level 1 of the IPU. The output of this decode circuit is not used by level 1, but transfers to DCL2 register at level 2 when the instruction in IR transfers to level 2. This decode circuit is selective. It inspects only for those instructions that require IPU attention at level 2 before the instruction reaches level 3. The instructions, op codes and resulting output signals that this circuit produces are listed in tables D-2 and D-3.

-IIFDTHF. This output from the decoding circuit indicates that the instruction at level 1 is "T Hazard Free". That is, the instruction type cannot be modified by indexing, so that the circuit that checks for an indexing hazard should be disabled. The op codes for the instructions in this category that produce -IIFDTHF are listed in table B-10, along with the component op codes for the other output signals from this circuit. The No Op input that produces -IIFDTHF is conditioned by -II2S(3) from the level 1 controller. This signal indicates that an indirect instruction is not at the

¬ IMURDATA (0-7)

INPUT SELECT SIGNAL	0	1	2	3	4	5	6	7
NO ACTIVE GATE	¬ ICURDATA (0-7) FROM I4CMREQ							
IHENDTUR(0)	IMQFREEZ	HARD CORE STATE 0	HARD CORE STATE 1	HARD CORE STATE 2	HARD CORE STATE 3	HARD CORE STATE 4	HARD CORE STATE 5	HARD CORE STATE 6
		IMQHCSTA(0-6)						
IHENDTUR(1)	HARD CORE COMMAND REGISTER IMQLSD(0-3)				IMQEXCH	IMQHCREQ	IMQHCINP	IMQLDPTR
IHENDTUR(2)	IMQSPSDW	NOT USED		OCTET COUNTER IMQOCTR(0-4)				
IHENDTUR(3)	NOT USED		PROTECT MODE ID ICQPRM:1(0,1)		NOT USED	OUTSTANDING REQUEST COUNTER-IMQRC(0-2)		
IHENDTUR(4)	ICQRDA:1	ICQRDS:1	ICQRA:1	ICQAR:1	ICQDAV:1	IOQDAV	ICQPAR:1	IOQPAR
IHENDTUR(5)	IOQRDA	ICQWRITE:1	IMQPRV:1	ICQIPPRV	IMQPAE:1	ICQIPPAE	ICQAREX	ICQIPIOP

(B)127625

Figure D-15. I4HDCORE UR Data Byte Assignments

Table D-2. Level 1 Instruction Decode Outputs

Op Code	Instruction	Output Signal to DCL2
0X	No Op	-IISDNOP -IIFDTHF -IIFDMHF
16	LLA	-IIFDTHF -IIFDMHF
1B	Load File	-IIFDLF -IIFDORG
1F	Load File Multiple	-IIFDLF -IIFDORG
2B	Store File	-IIFDORG
2F	Store File Multiple	-IIFDORG
50	Add immediate to AR	-IIFDMHF
51	Add immediate to AR, halfword	-IIFDMHF
54	Load immediate to AR	-IIFDMHF
55	Load immediate to AR, halfword	-IIFDMHF
58	Subtract immediate from AR	-IIFDMHF
59	Subtract immediate from AR, halfword	-IIFDMHF
7X	Add, divide or multiply immediate operations	-IIFDMHF
84 - 87	Conditional Branches	-IIFDTHF
90	Monitor call and proceed	-IIFDMHF
93	Push	-IIFDPP
94	Monitor call and wait	-IIFDMHF
96	Execute	IISDXEC -IISDXEC
97	Pull	-IIFDPP
9A	Fork	-IIFDTHF -IIFDMHF
9B	Join	-IIFDTHF -IIFDMHF

Table D-2. Level 1 Instruction Decode Outputs (Continued)

Op Code	Instruction	Output Signal to DCL2
9E	Prepare to Branch	-IISDPB
BO	Execute Vector Parameter File	-IISDVECT
CC, CD, CF	Circular shifts	-IIFDMHF
Dx	Compare immediates	-IIFDMHF
Fx	Logical operations on immediates	-IIFDMHF

Table D-3. Output Signal Components

Signal	Op Codes that produce it
IIFDTHF	Ox, 16, 84, 85, 86, 87, 9A or 9B
-IIFDLF	1B or 1F
-IIFDORG	1B, 1F, 2B or 2F
-IIFDPP	93 or 97
-IISDNOP	Ox
-IISDPB	9E
-IISDVECT	BO
-IISDXEC	96
-IIFDMHF	Ox, 16, 50, 51, 54, 55, 58, 59, 90, 94, 9A, 9B, CC, CD, CF, Dx, or Fx.

level 1 controller is not in the level 3 indirect at level 3 state. If the level 1 controller is in that state, the No Op input to the T Hazard Free signal is disabled, so that the T Hazard Free flag is not set by the blocked state of the pipe while the indirect instruction is being processed. The remaining op codes that produce this signal are not qualified.

-IIFDMHF. This output from the decoding circuit indicates that the instruction at level 1 is "M Hazard Free". That is, the instruction cannot be modified using the base relative modification circuits, so that the circuit that checks for a base modification hazard should be disabled. Refer to table D-2 for the component op codes that are included in the M Hazard Free category.

LEVEL 2 INSTRUCTION DECODE

The level 2 instruction decode receives the op code portion of the instruction at level 2 from the R2 register (IPQR2(0-3)) and from the ADDRM register (IPQADDRM(4-7)) on the I4PIPE circuit boards. It decodes the eight bit input into two hexadecimal digits, and then combines the hexadecimal digits to determine if the instruction is one of a set of instructions that requires attention from the level 3 controller. The output of this circuit is not used by level 2, but transfers to the DCL3 register at level 3 when the instruction transfers to level 3. The circuit inspects only for the set of instructions outlined in table D-4.

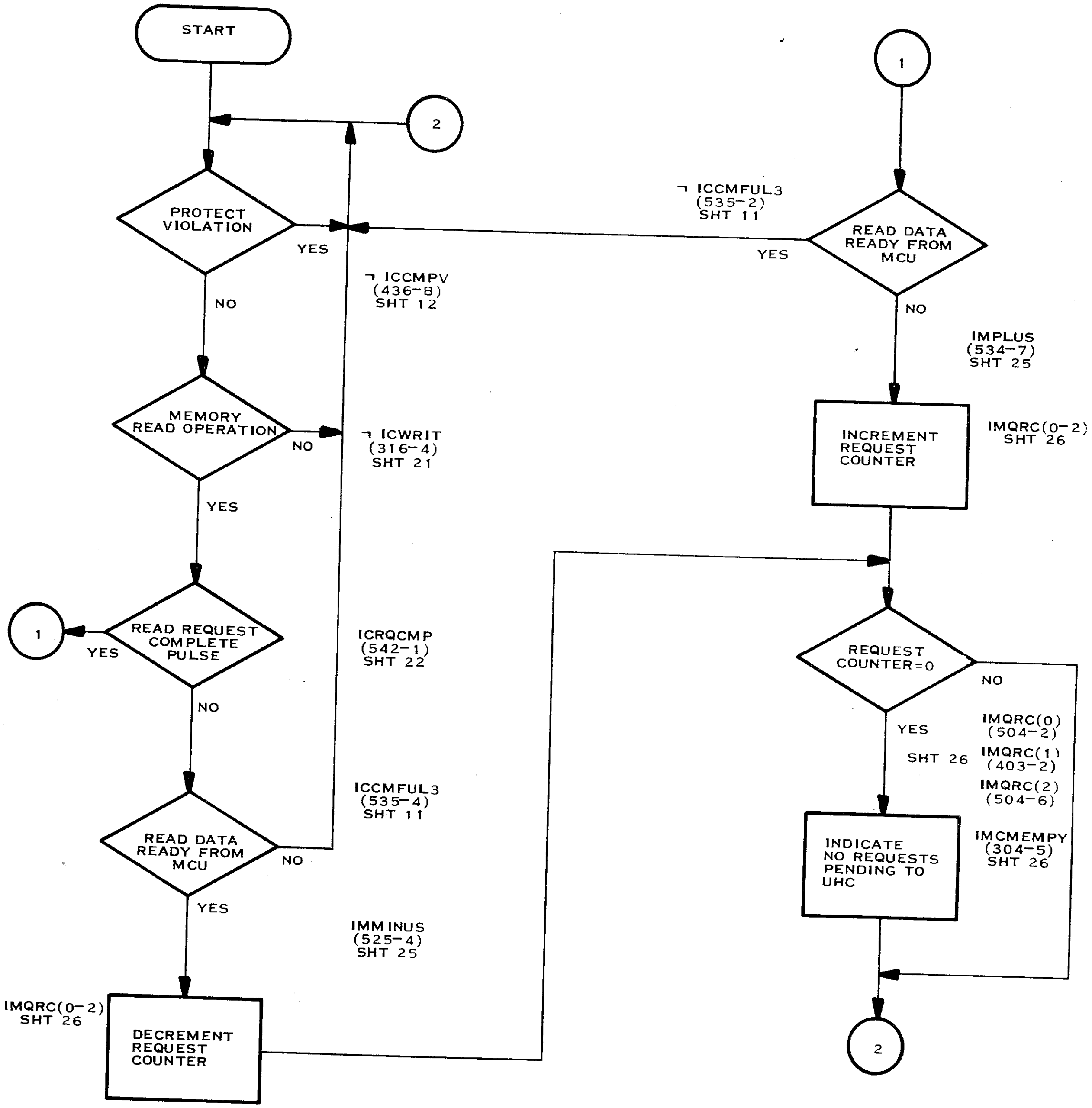
Figures D-16 and D-17 are flowcharts for the I4HDCORE circuit board and figure D-18 is the card block diagram.

Table D-4. Level 2 Instruction Decode Outputs

Hexadecimal Op Code	Instruction	Signal Generated
11	Load Arithmetic mask and condition	-IPFDLAC -IPFDLAM
12	Load arithmetic mask	-IPFDLAM
13	Load arithmetic exception condition	-IPFDLAC
16	Load look ahead	-IPSDLLA
1A	Exchange	-IPSDXCH
1F	Load File Multiple	-IPFDMLT
22	Store Program Status Word	-IPSDSPS
2F	Store file multiple	-IPFDMLT
80	Increment, test and skip on equal	-IPFDSKE
82	Decrement, test and skip on equal	-IPFDSKE
84	Branch on AR less than or equal	-IPFDBLCE IPBCLBCG
85	Branch on AR greater than	-IPFDBDG IPBCLBCG
86	Branch on index less than or equal	-IPFDBCLE IPBCLBCG
87	Branch on index greater than	-IPFDBCG IPBCLBCG
90	Monitor call and proceed	-IPSDMCP -IPFDBWN
91	Branch on compare code	-IPSDBCC
93	Push	-IPSDPSH -IPFDSTK
94	Monitor call and wait	-IPFDBWN
95	Branch on result code	-IPSDBRC
97	Pull	-IPFDSTK
98	Branch and load base register with P3	-IPFDBBX
99	Branch and load index or vector registers with P3	-IPFDBBX
9A	Fork	-IPSDFORK

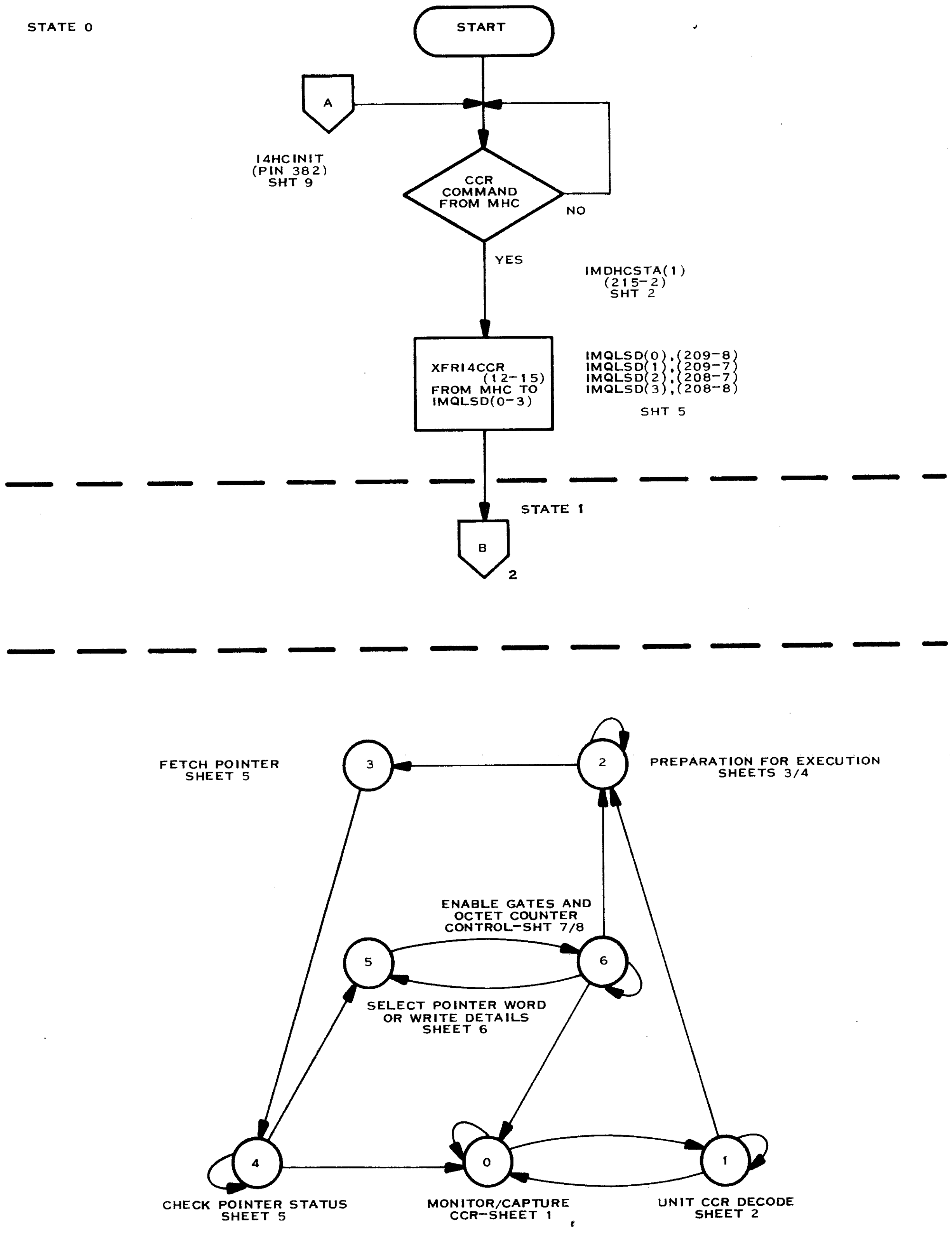
Table D-4. Level 2 Instruction Decode Outputs (Continued)

Hexadecimal Op Code	Instruction	Signal Generated
9B	Join	-IPSDJOIN
9C	Branch on execute condition true	-IPSDBXEC
9D	Branch on arithmetic exception true	-IPSDBAE
AE	Store clock	-IPSDSTC



(B)127626

Figure D-16. I4HDCORE, Outstanding Request Counter Flowchart



(B)127627 (1/8)

Figure D-17. I4HDCORE, IPU-4 Unit Hardcore Flowchart (Sheet 1 of 8)

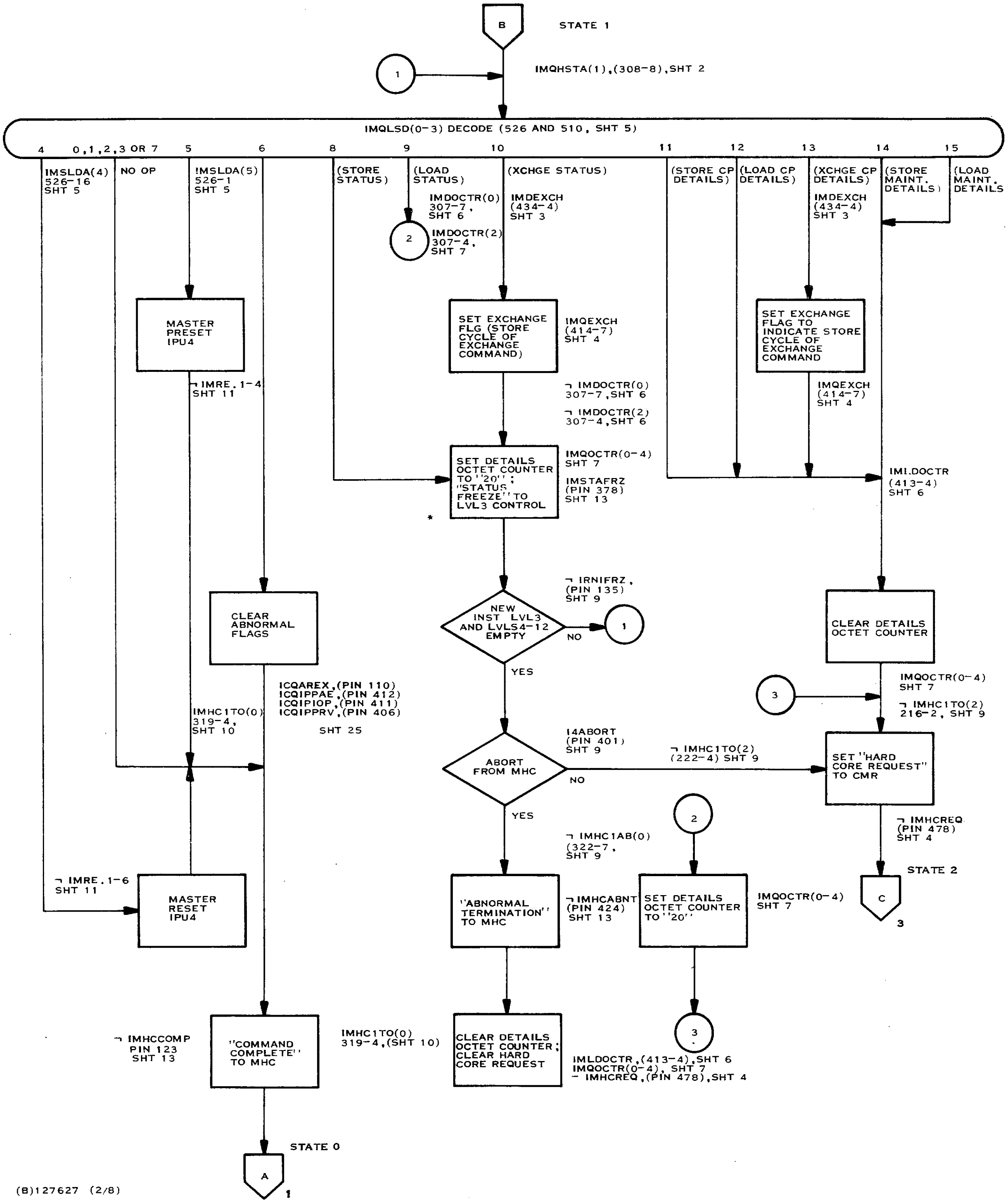


Figure D-17. I4HDCORE, IPU-4 Unit Hardcore Flowchart (Sheet 2 of 8)

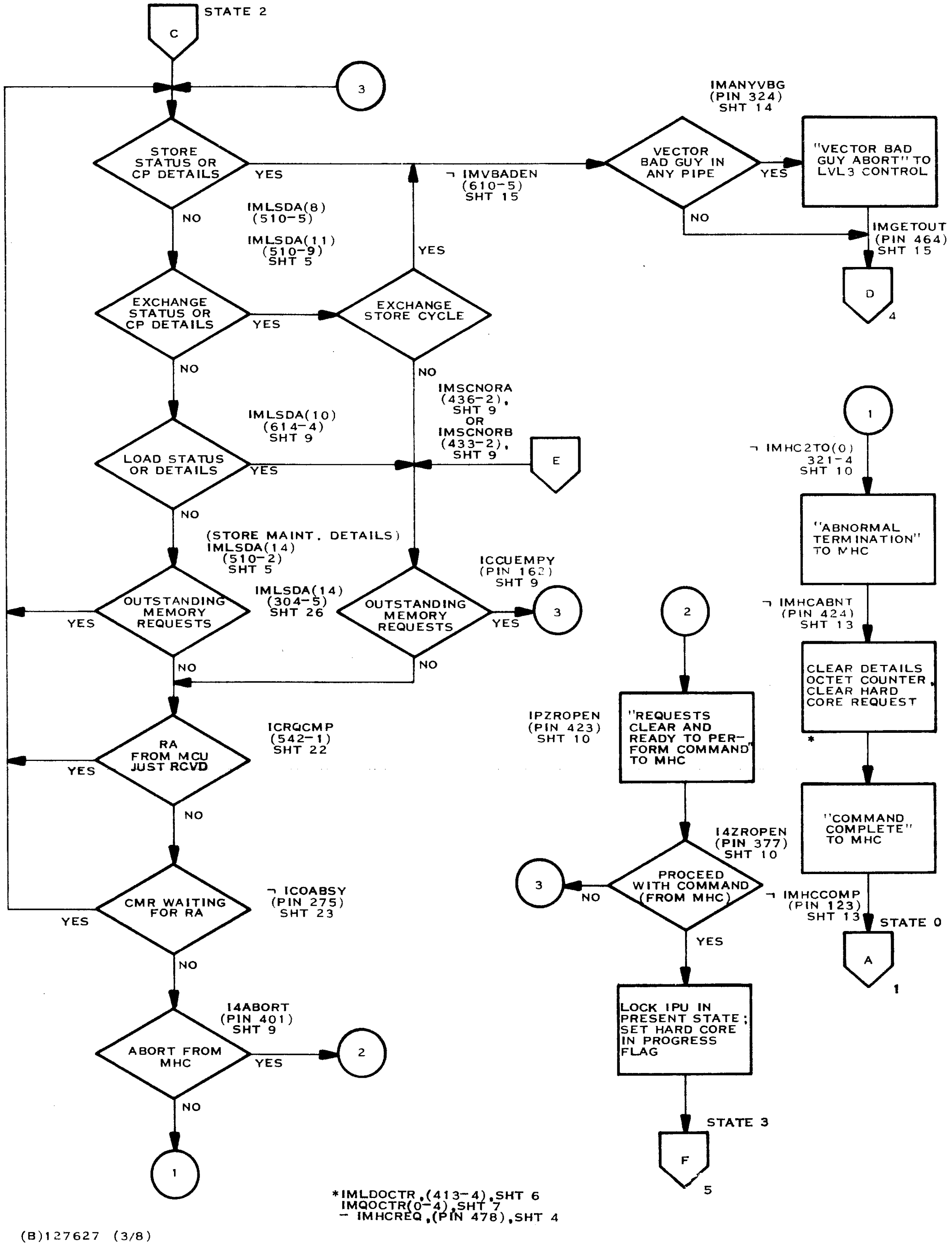
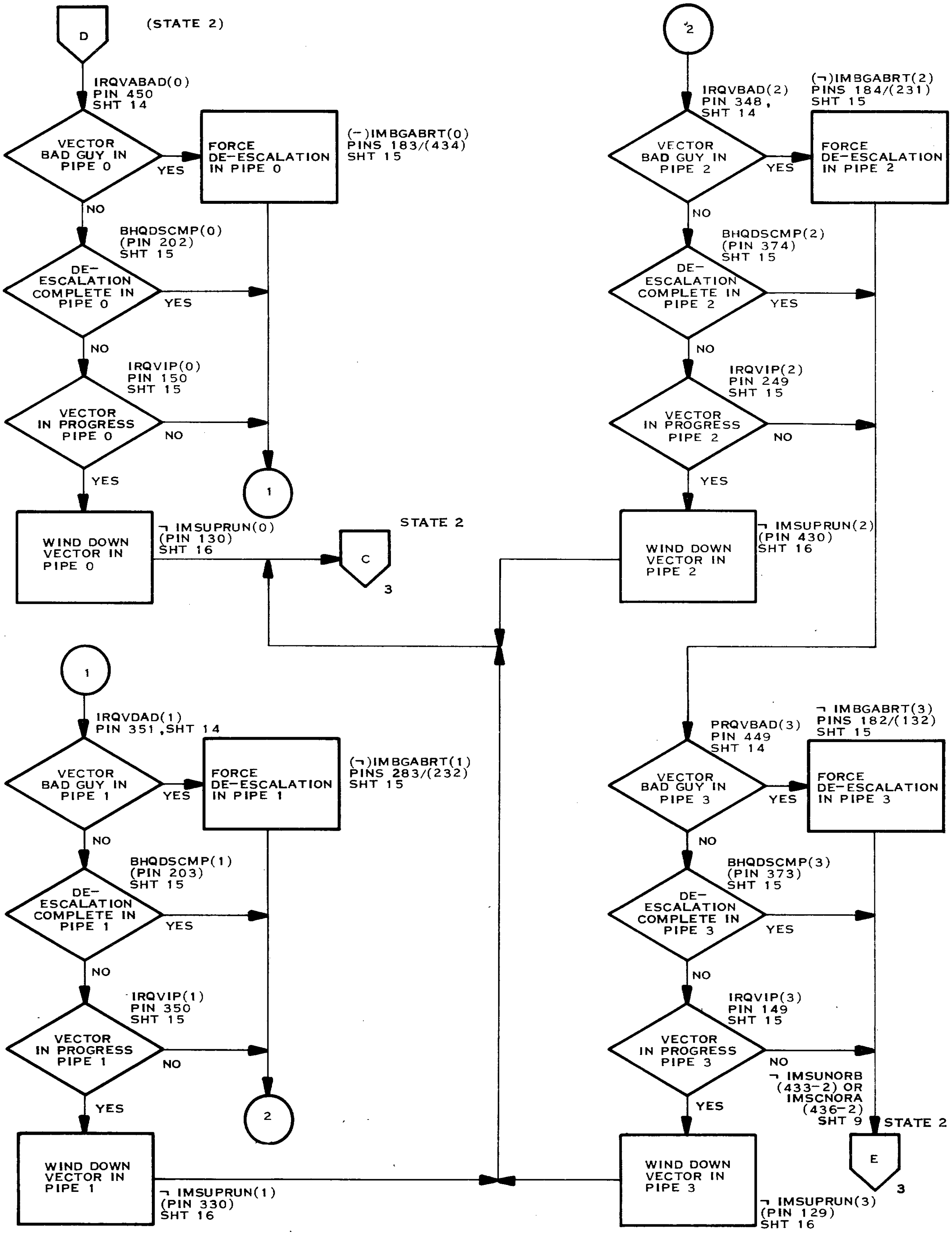
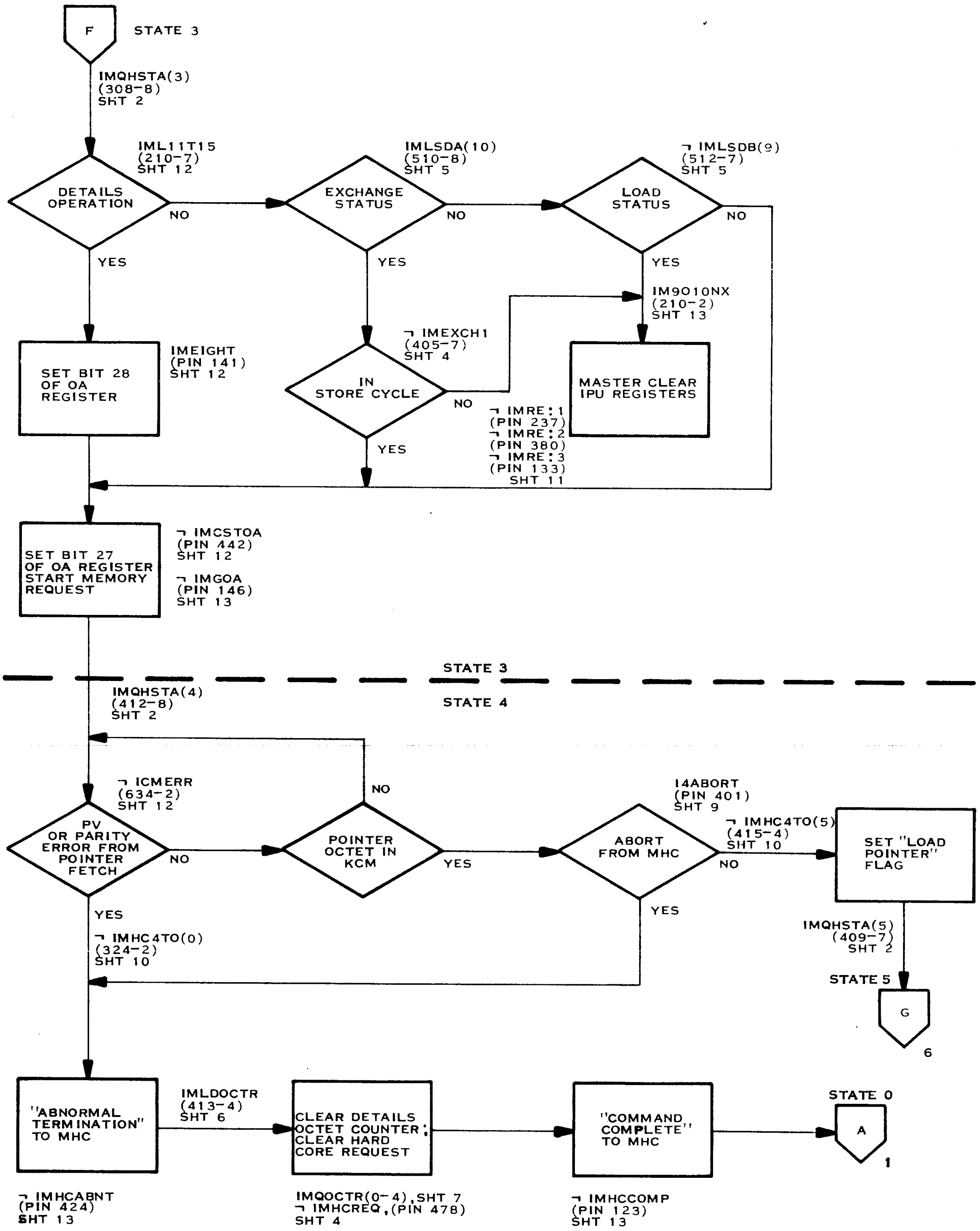

STATE 2
C
3
STORE STATUS OR CP DETAILS
YES
NO
IMLSDA(8) (510-5)
IMLSDA(11) (510-9) SHT 5
¬ IMVBADEN (610-5) SHT 15
YES
IMANYVBG (PIN 324) SHT 14
VECTOR BAD GUY IN ANY PIPE
YES
"VECTOR BAD GUY ABORT" TO LVL3 CONTROL
NO
IMGETOUT (PIN 464) SHT 15
D
4
EXCHANGE STATUS OR CP DETAILS
YES
EXCHANGE STORE CYCLE
NO
NO
IMSCNORA (436-2), SHT 9 OR IMSCNORB (433-2), SHT 9
IMLSDA(10) (614-4) SHT 9
LOAD STATUS OR DETAILS
YES
E
NO
1
¬ IMHC2TO(0) 321-4 SHT 10
"ABNORMAL TERMINATION" TO MHC
(STORE MAINT. DETAILS) IMLSDA(14) (510-2) SHT 5
ICCUEMPY (PIN 162) SHT 9
OUTSTANDING MEMORY REQUESTS
YES
IMLSDA(14) (304-5) SHT 26
OUTSTANDING MEMORY REQUESTS
YES
3
2
¬ IMHCABNT (PIN 424) SHT 13
NO
NO
CLEAR DETAILS OCTET COUNTER, CLEAR HARD CORE REQUEST
*
IPZROPEN (PIN 423) SHT 10
"REQUESTS CLEAR AND READY TO PERFORM COMMAND" TO MHC
RA FROM MCU JUST RCVD
YES
ICRQCMP (542-1) SHT 22
NO
"COMMAND COMPLETE" TO MHC
I4ZROPEN (PIN 377) SHT 10
3
NO
PROCEED WITH COMMAND (FROM MHC)
¬ IMHCCOMP (PIN 123) SHT 13
STATE 0
CMR WAITING FOR RA
YES
¬ ICOABSY (PIN 275) SHT 23
YES
A
1
NO
I4ABORT (PIN 401) SHT 9
LOCK IPU IN PRESENT STATE; SET HARD CORE IN PROGRESS FLAG
ABORT FROM MHC
YES
2
NO
STATE 3
1
F
5
*IMLDOCTR,(413-4),SHT 6
IMQOCTR(0-4),SHT 7
¬ IMHCREQ,(PIN 478),SHT 4
(B)127627 (3/8)

Figure D-17. I4HDCORE, IPU-4 Unit Hardcore Flowchart (Sheet 3 of 8)



(B)127627 (4/8)

Figure D-17. I4HDCORE, IPU-4 Unit Hardcore Flowchart (Sheet 4 of 8)



(B)127627 (5/8)

Figure D-17. I4HDCORE, IPU-4 Unit Hardcore Flowchart (Sheet 5 of 8)

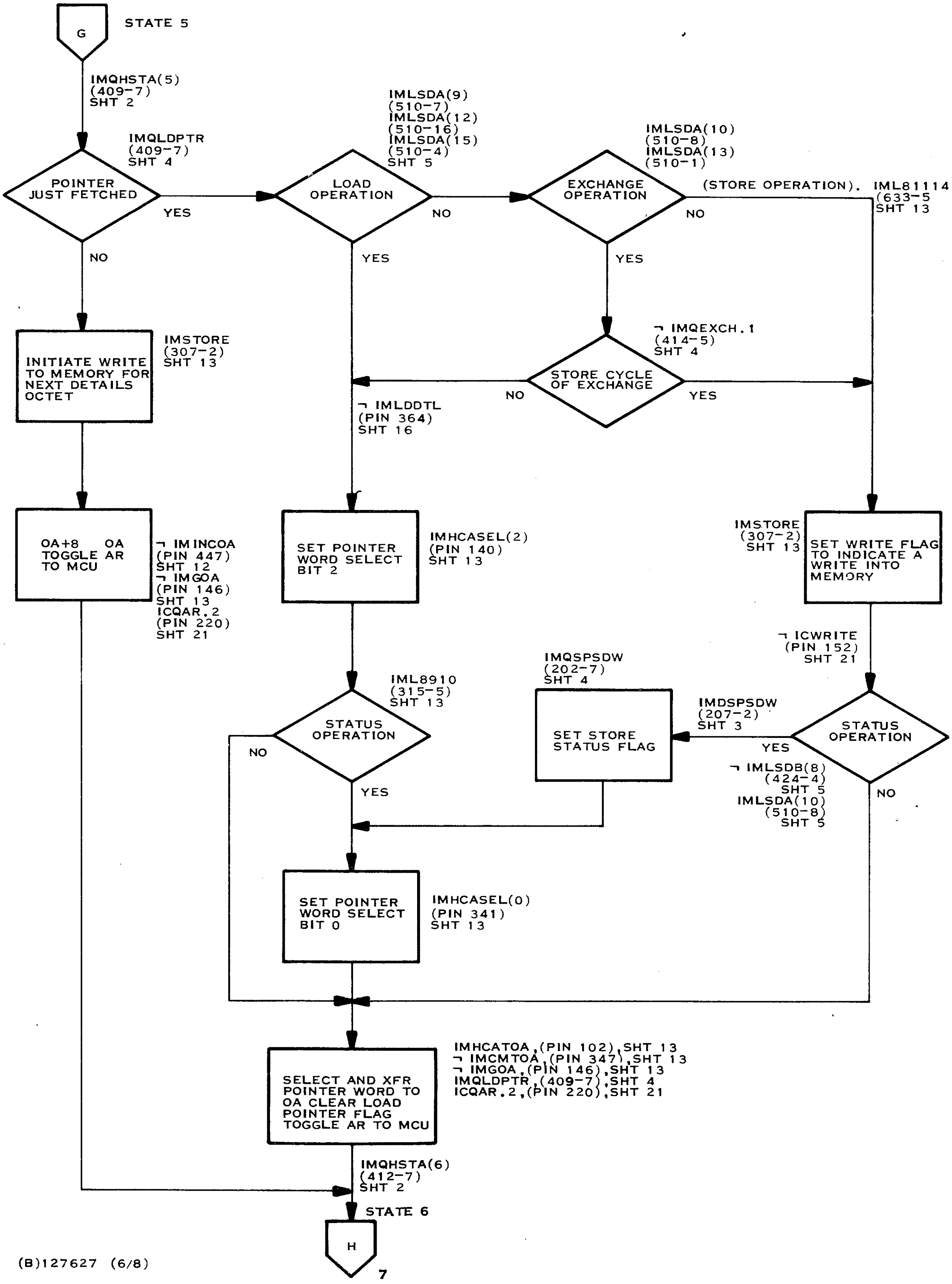


Figure D-17. I4HDCORE, IPU-4 Unit Hardcore Flowchart (Sheet 6 of 8)

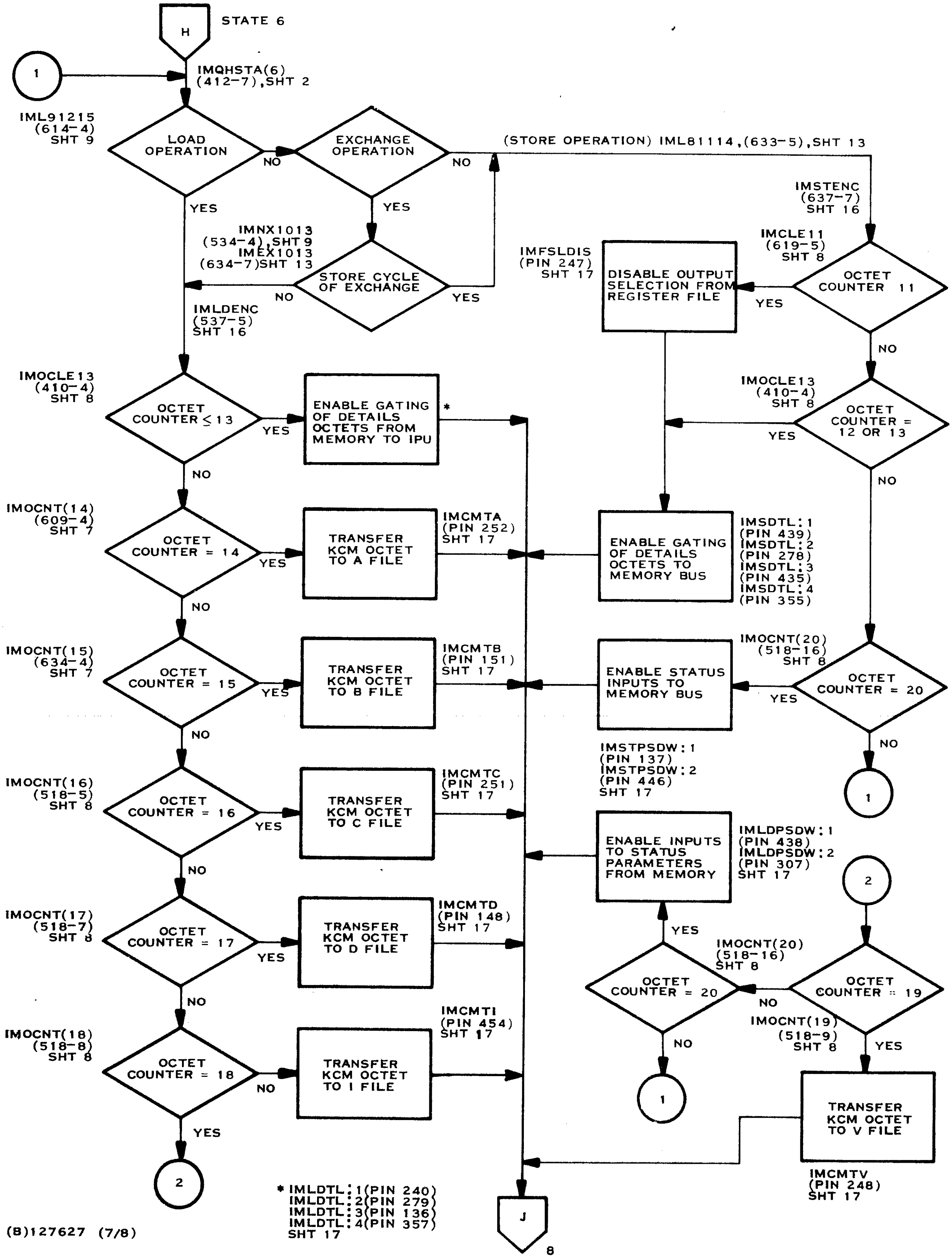
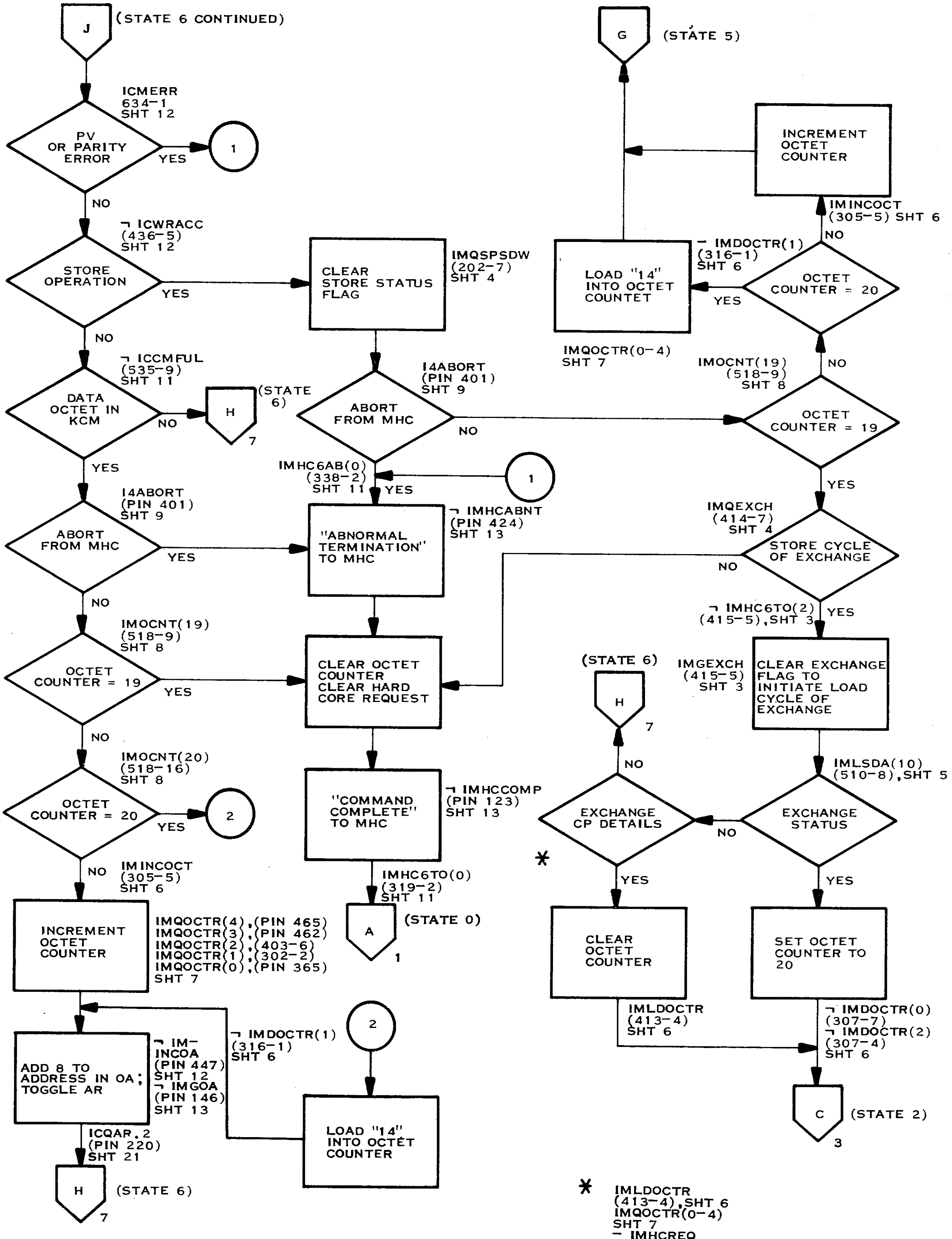


Figure D-17. I4HDCORE, IPU-4 Unit Hardcore Flowchart (Sheet 7 of 8)



(B)127627 (8/8)

Figure D-17. I4HDCORE, IPU-4 Unit Hardcore Flowchart (Sheet 8 of 8)

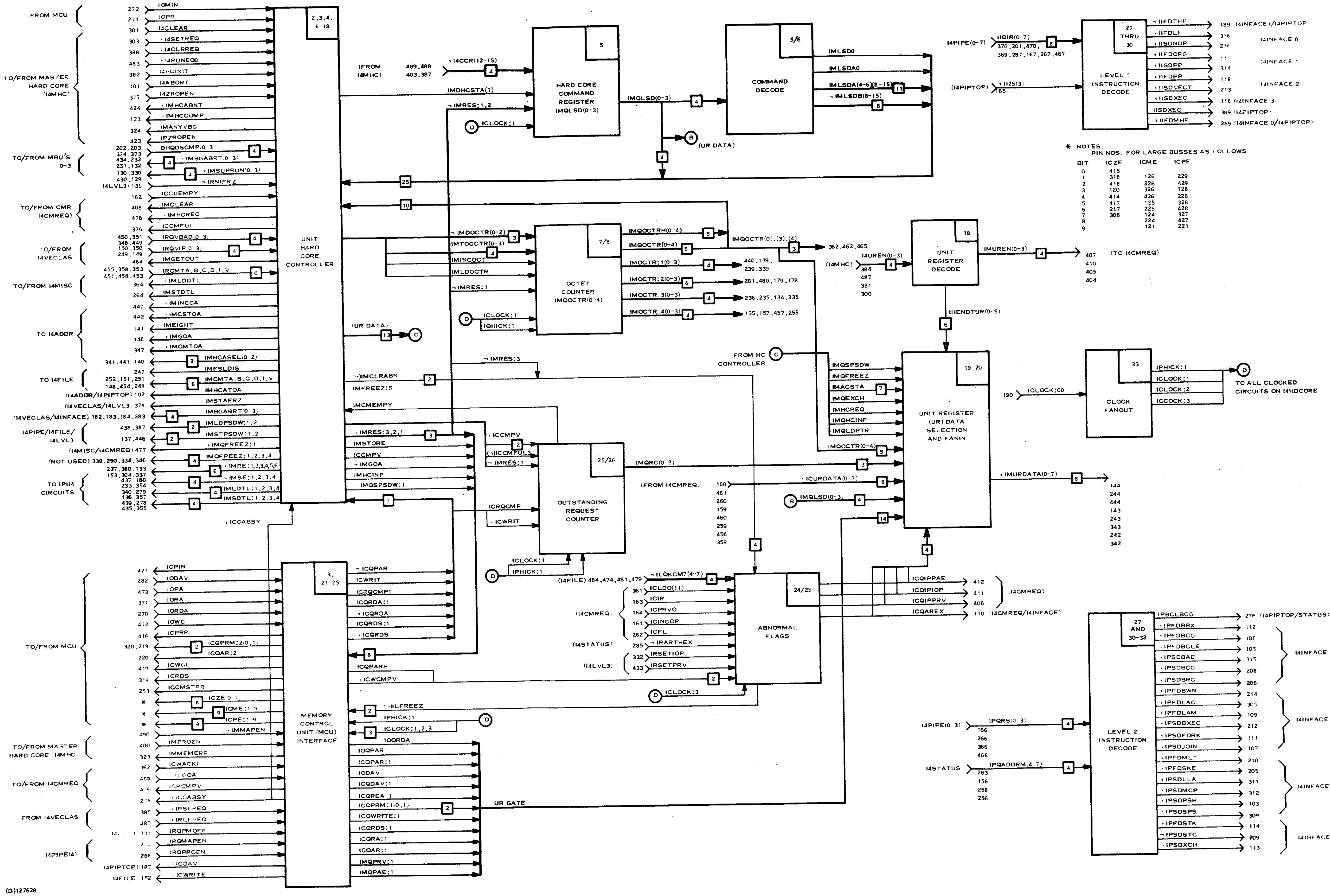
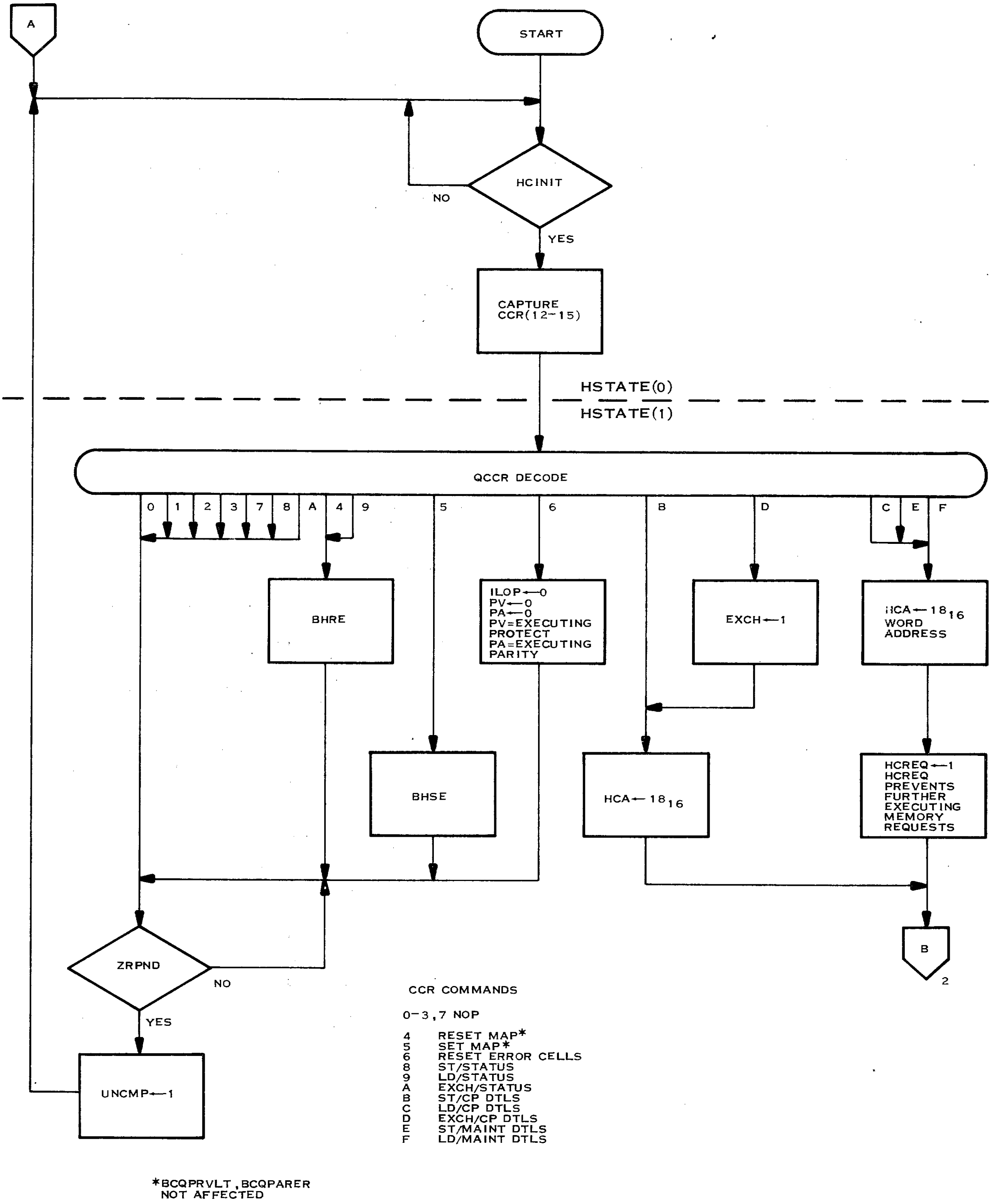


Figure D-18. I4HDCORE Circuit Board Block Diagram



CCR COMMANDS

0–3,7	NOP
4	RESET MAP*
5	SET MAP*
6	RESET ERROR CELLS
8	ST/STATUS
9	LD/STATUS
A	EXCH/STATUS
B	ST/CP DTLS
C	LD/CP DTLS
D	EXCH/CP DTLS
E	ST/MAINT DTLS
F	LD/MAINT DTLS

*BCQPRVLT, BCQPARER NOT AFFECTED

(B)132481 (1/3)

Figure D-19. 4X MBU Unit Hardcore Flowchart (Sheet 1 of 3)

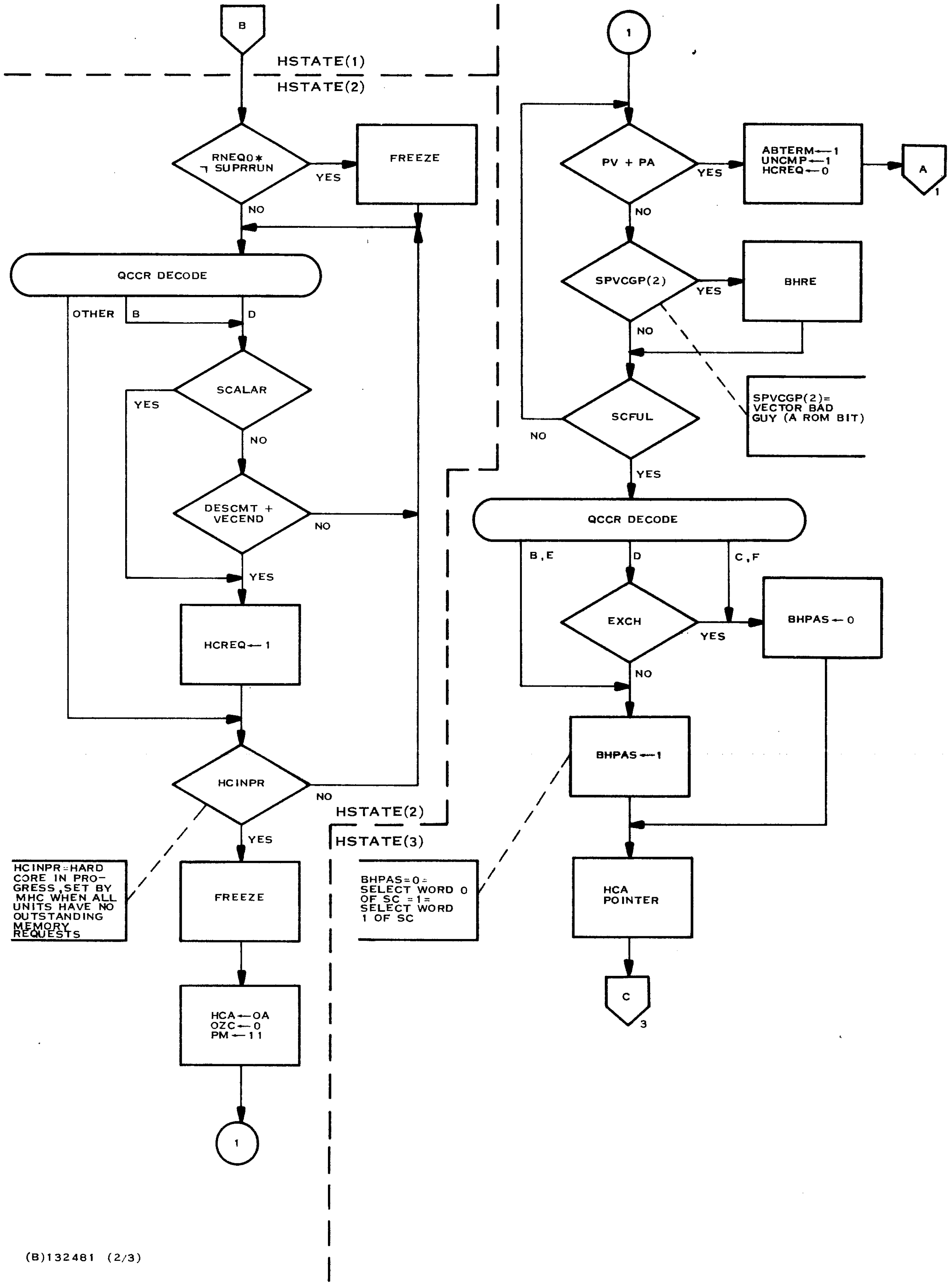


Figure D-19. 4X MBU Unit Hardcore Flowchart (Sheet 2 of 3)

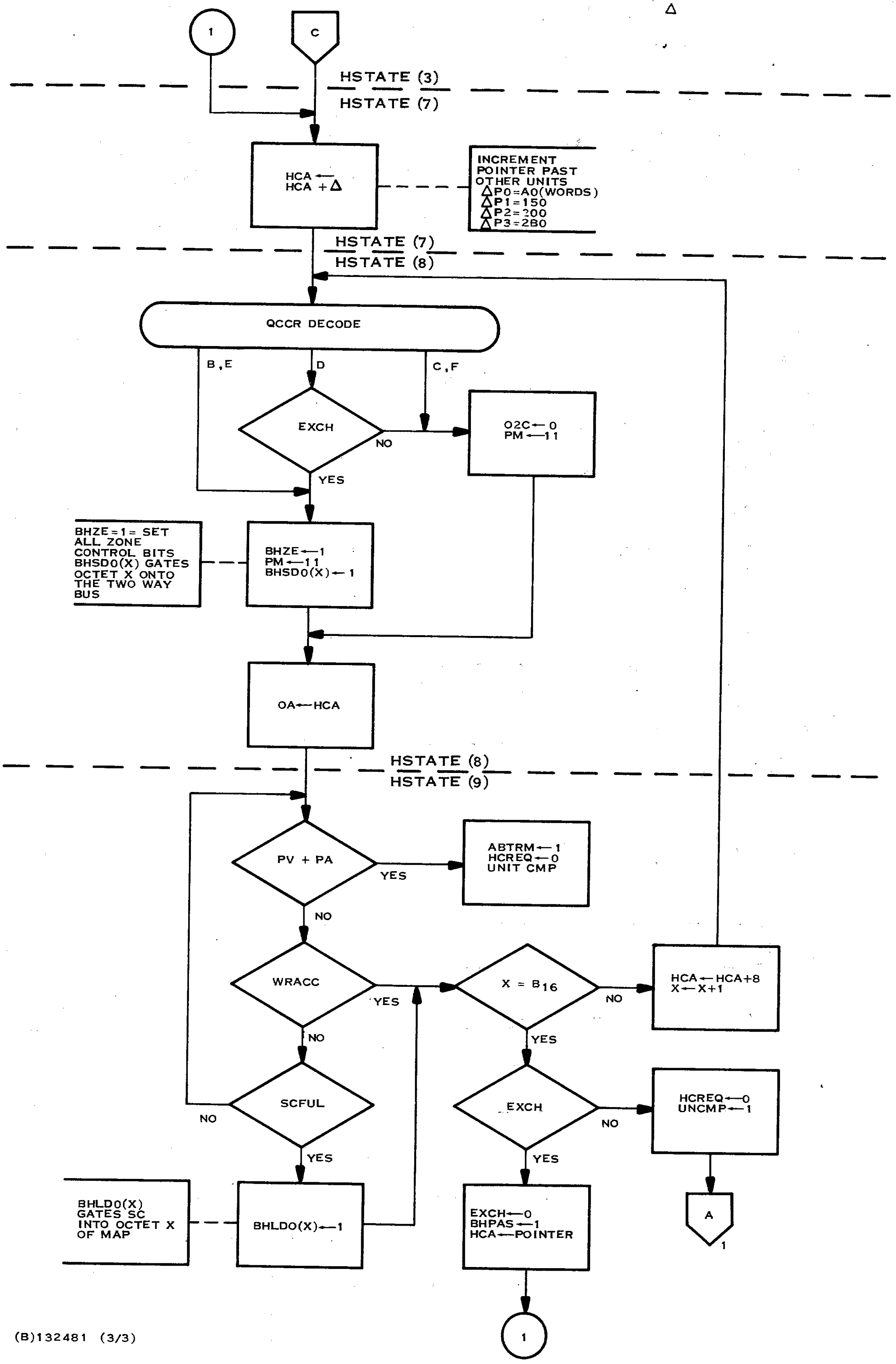
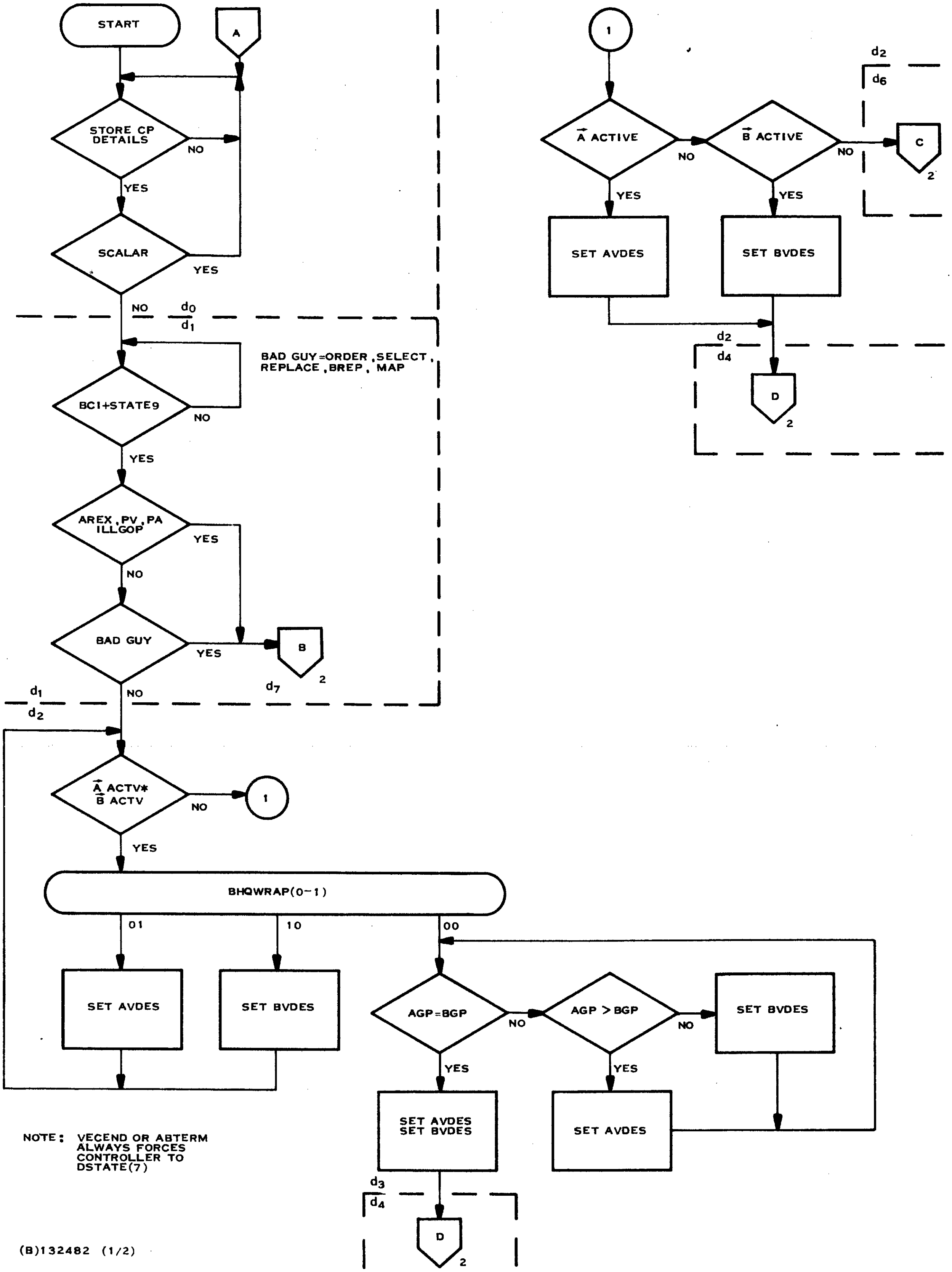


Figure D-19. 4X MBU Unit Hardcore Flowchart (Sheet 3 of 3)



NOTE: VECEND OR ABTERM ALWAYS FORCES CONTROLLER TO DSTATE(7)

(B)132482 (1/2)

Figure D-20. 4X MBU UHC De-escalate Controller Flowchart (Sheet 1 of 2)

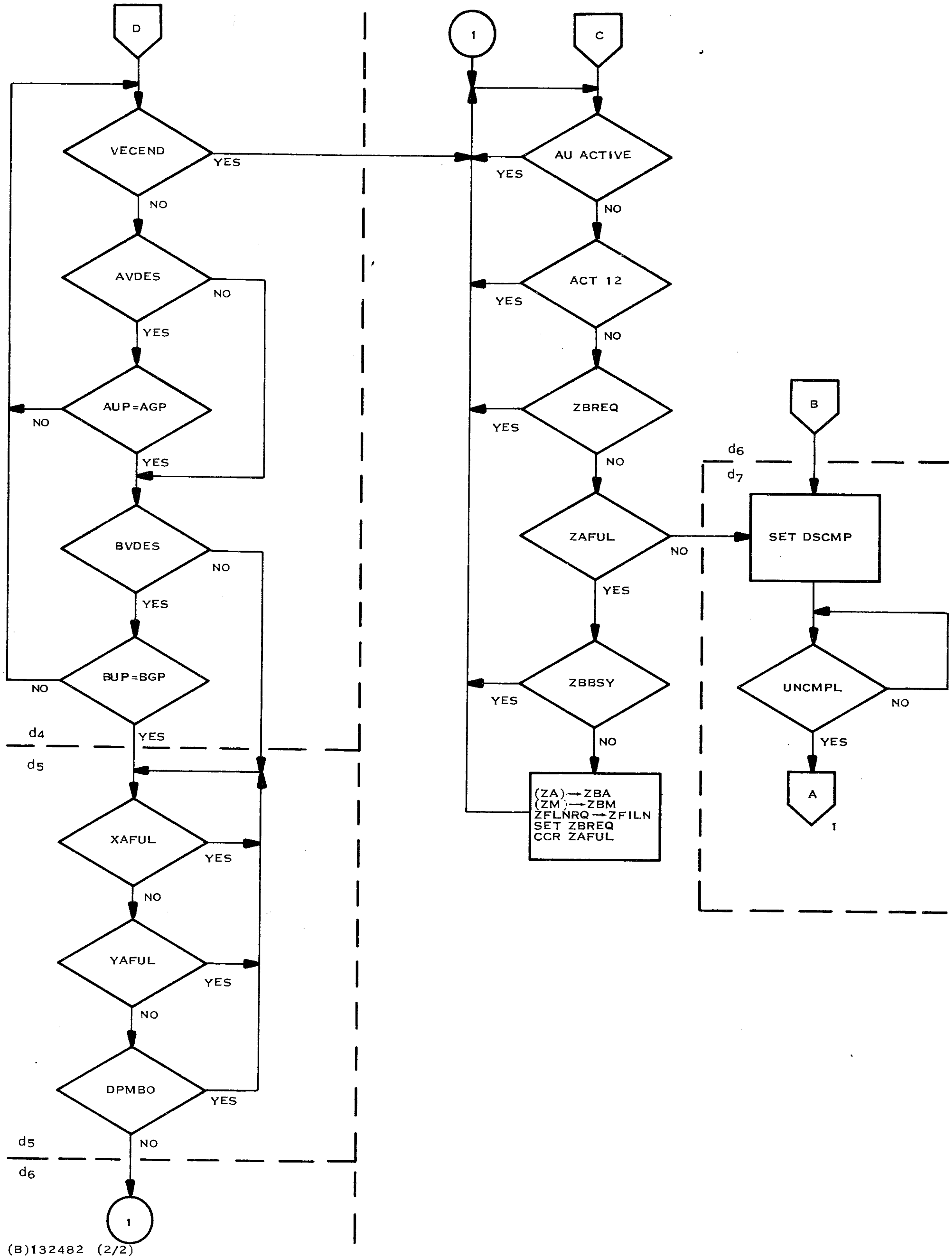
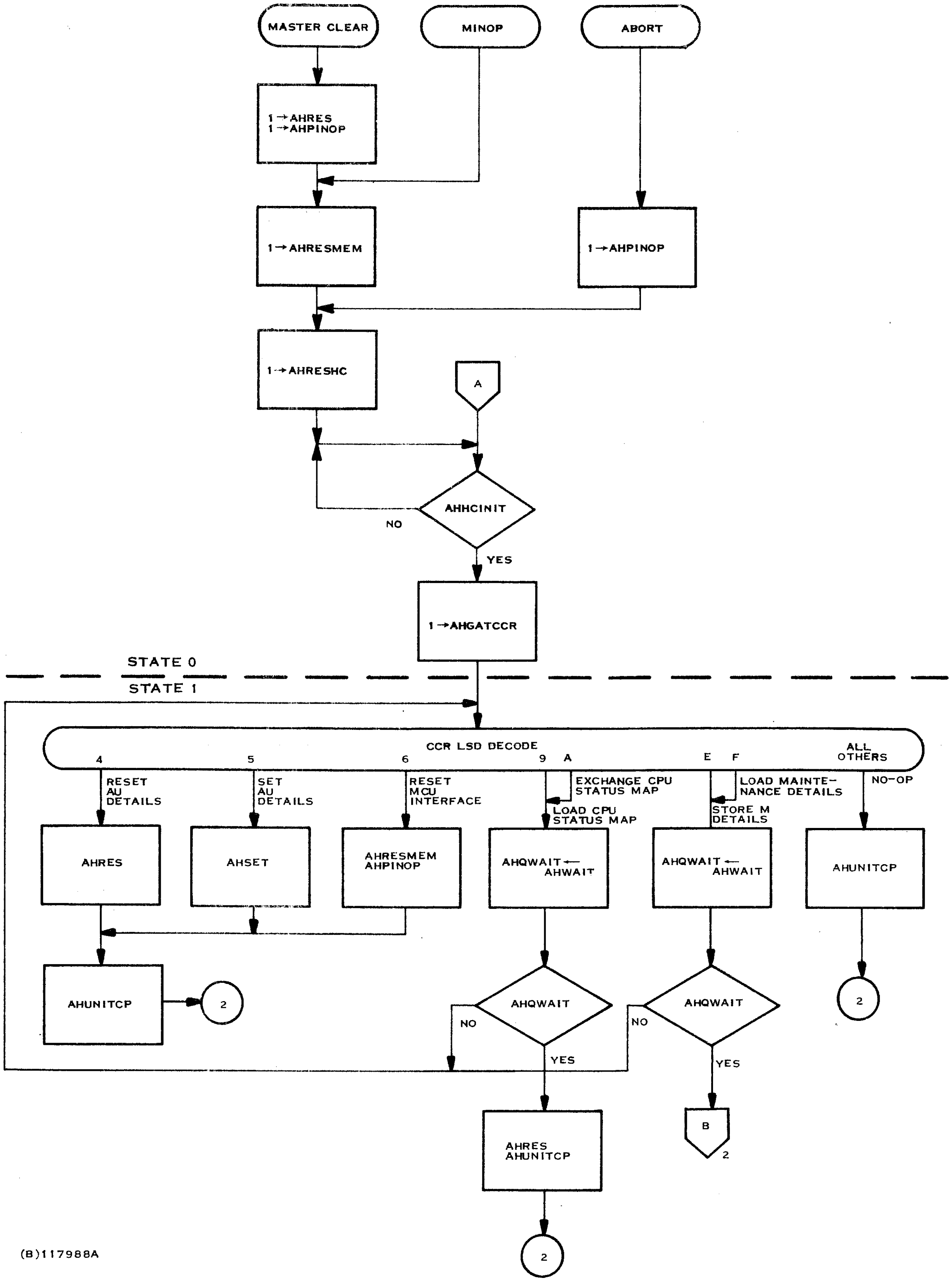


Figure D-20. 4X MBU UHC De-escalate Controller Flowchart (Sheet 2 of 2)



(B)117988A

Figure D-21. 4X AU Unit Hardcore Flowchart (Sheet 1 of 7)

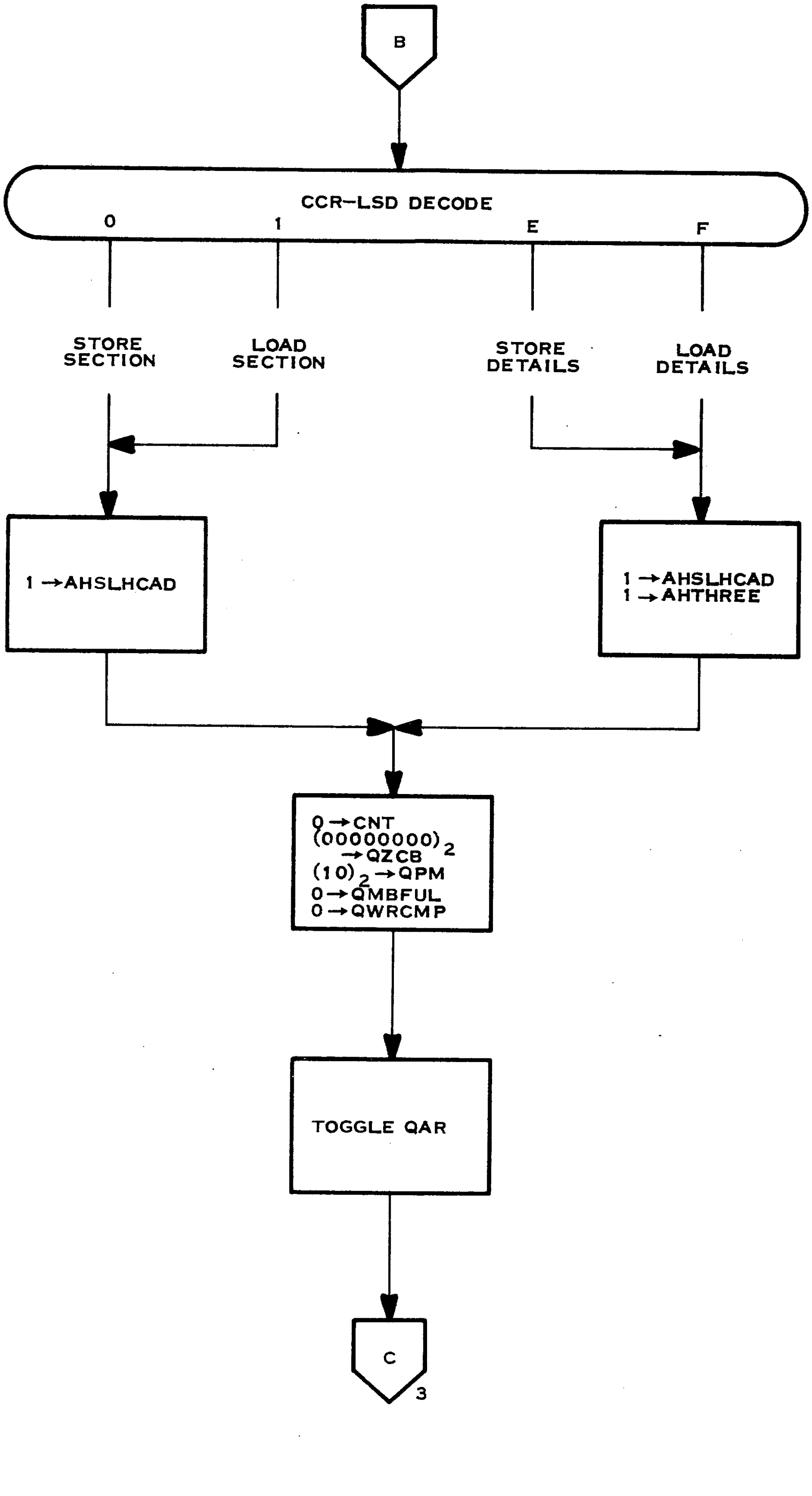


Figure D-21. 4X AU Unit Hardcore Flowchart (Sheet 2 of 7)

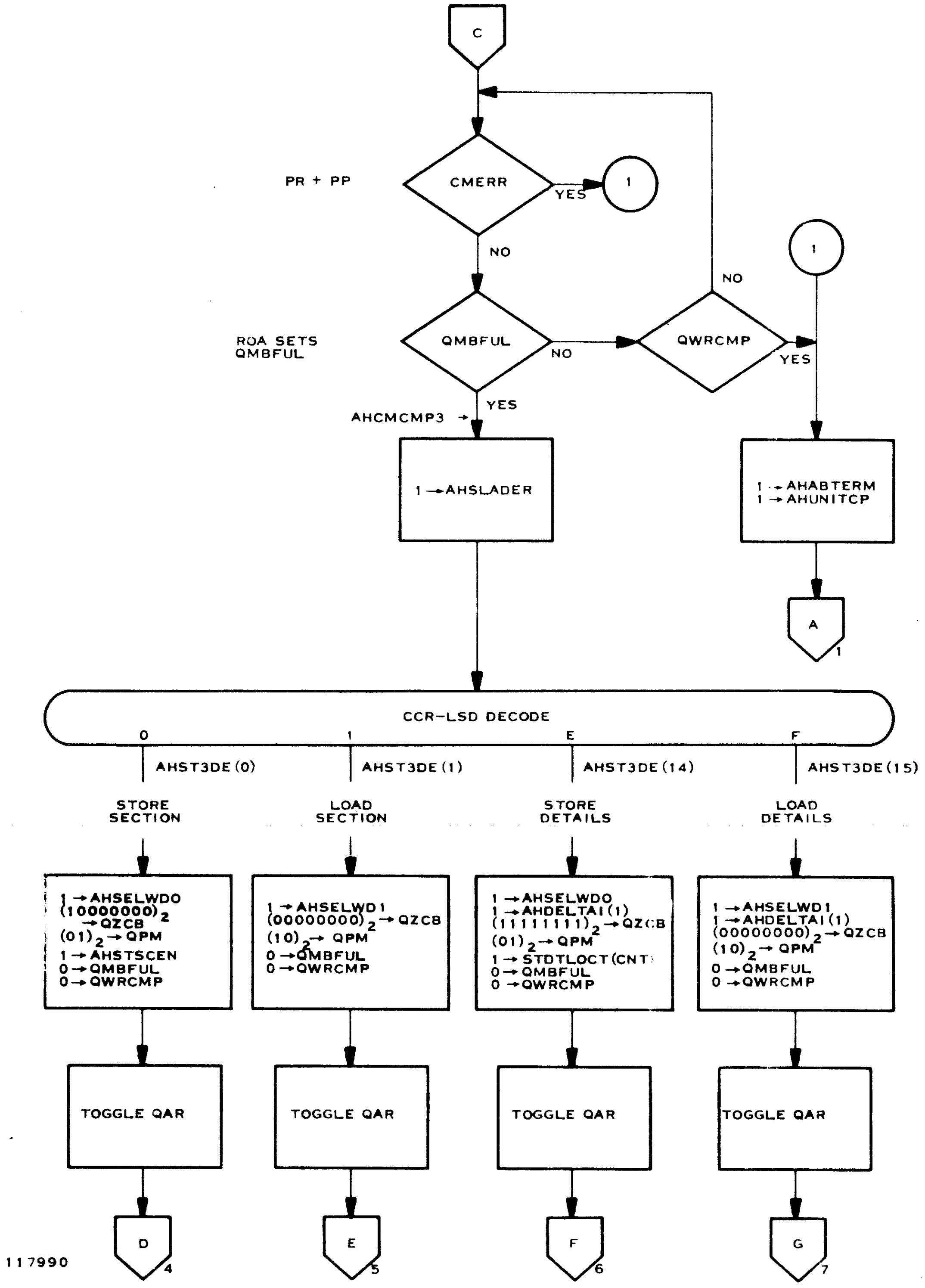
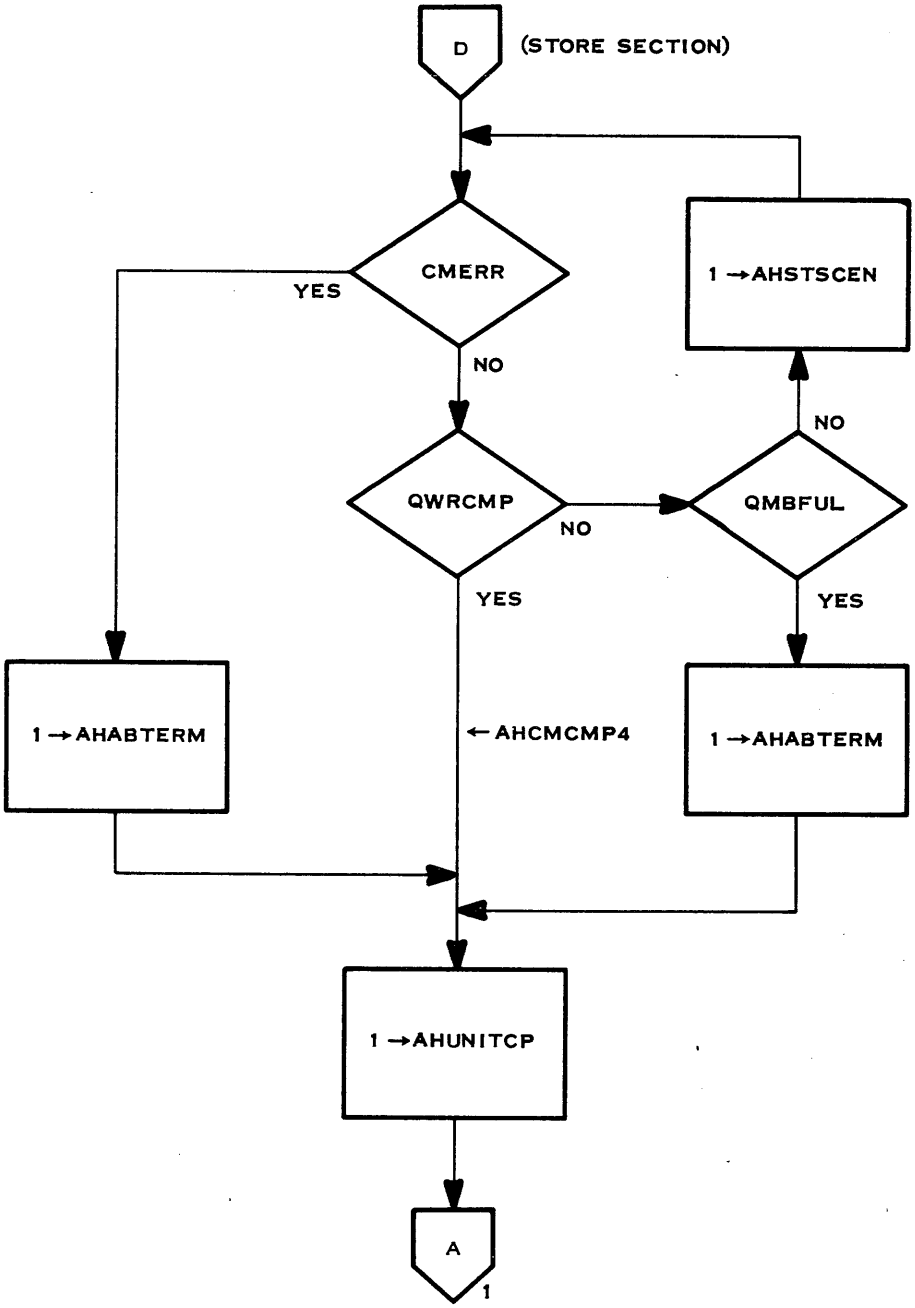
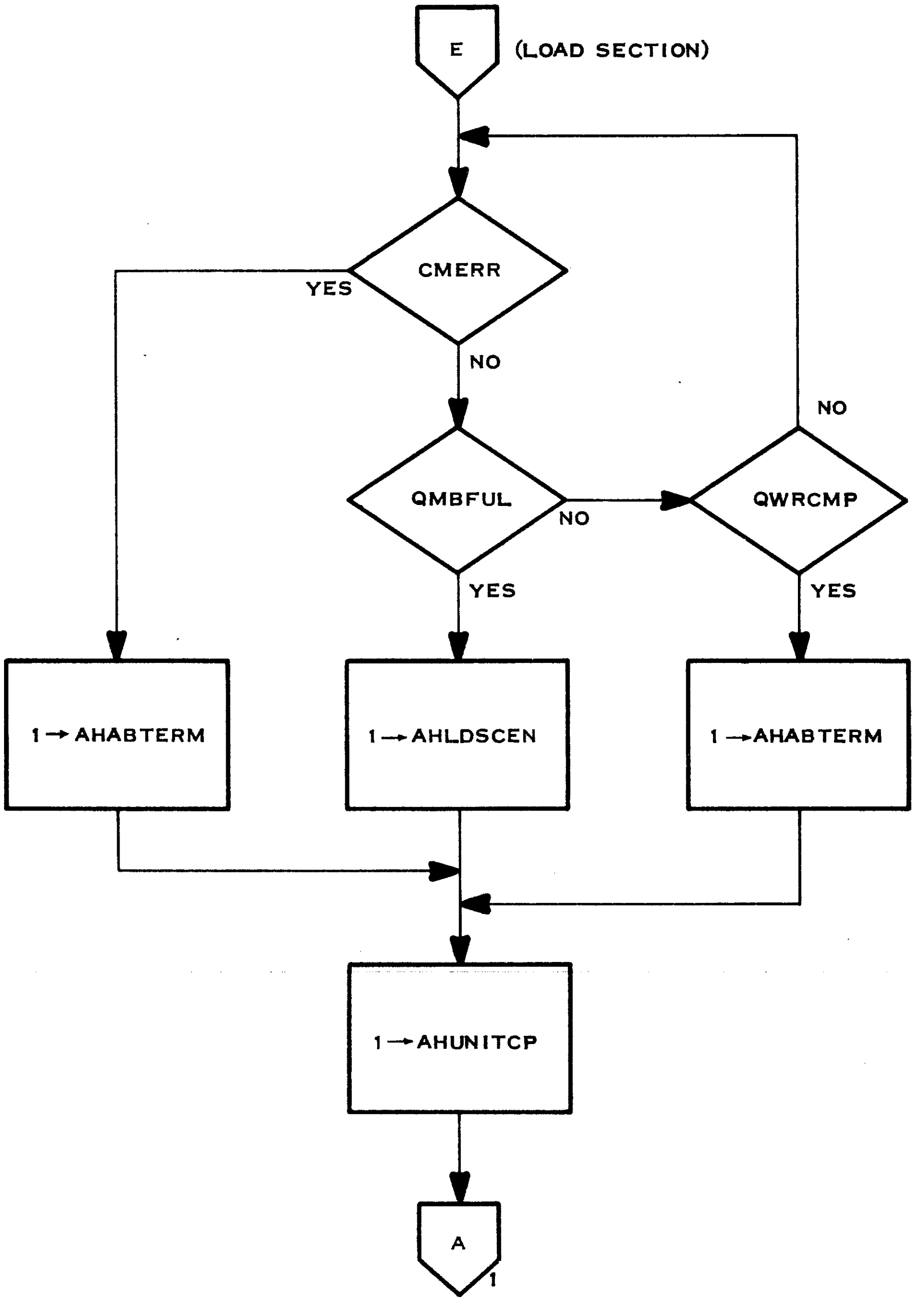


Figure D-2l. 4X AU Unit Hardcore Flowchart (Sheet 3 of 7)



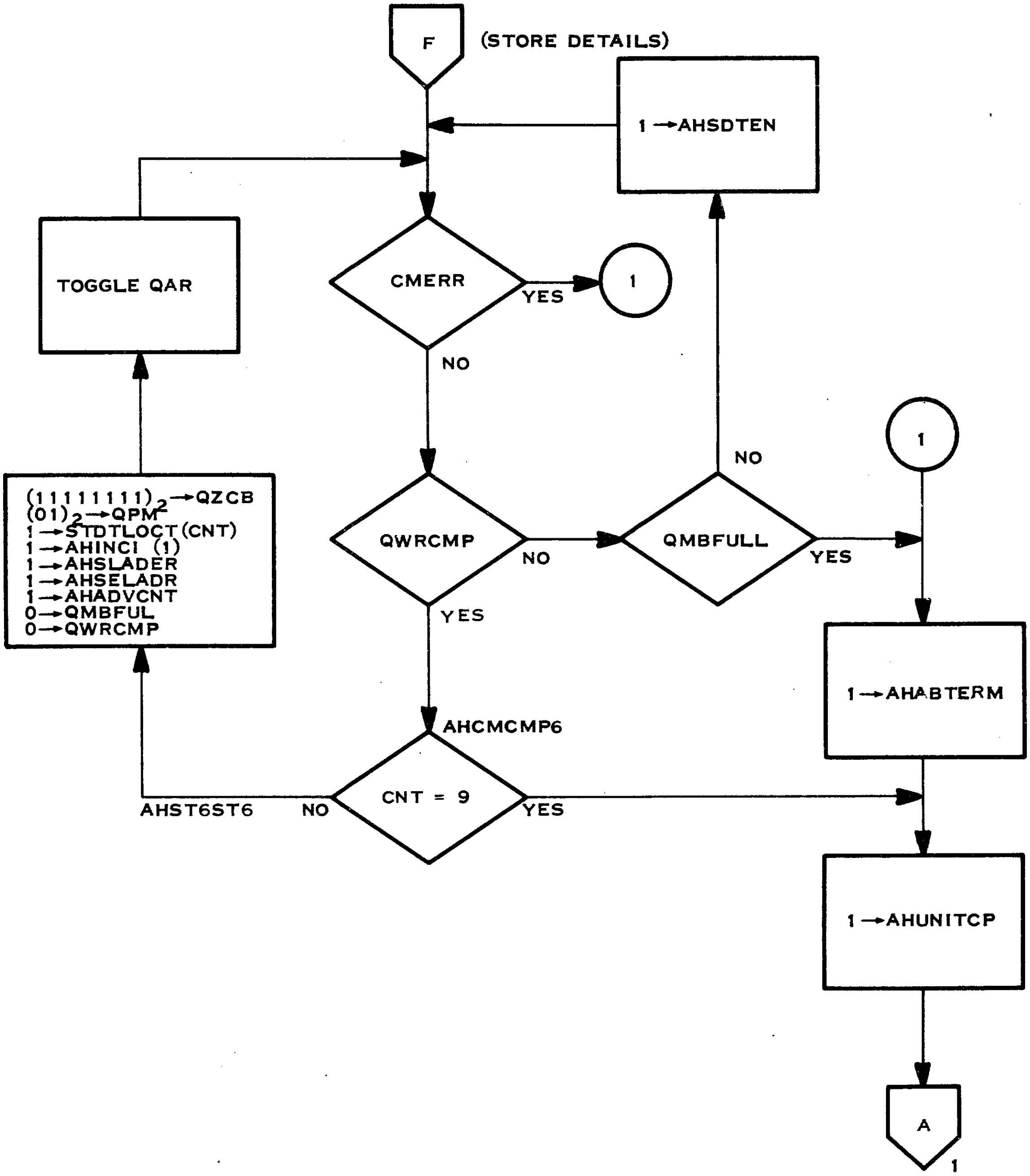
117991

Figure D-21. 4X AU Unit Hardcore Flowchart (Sheet 4 of 7)



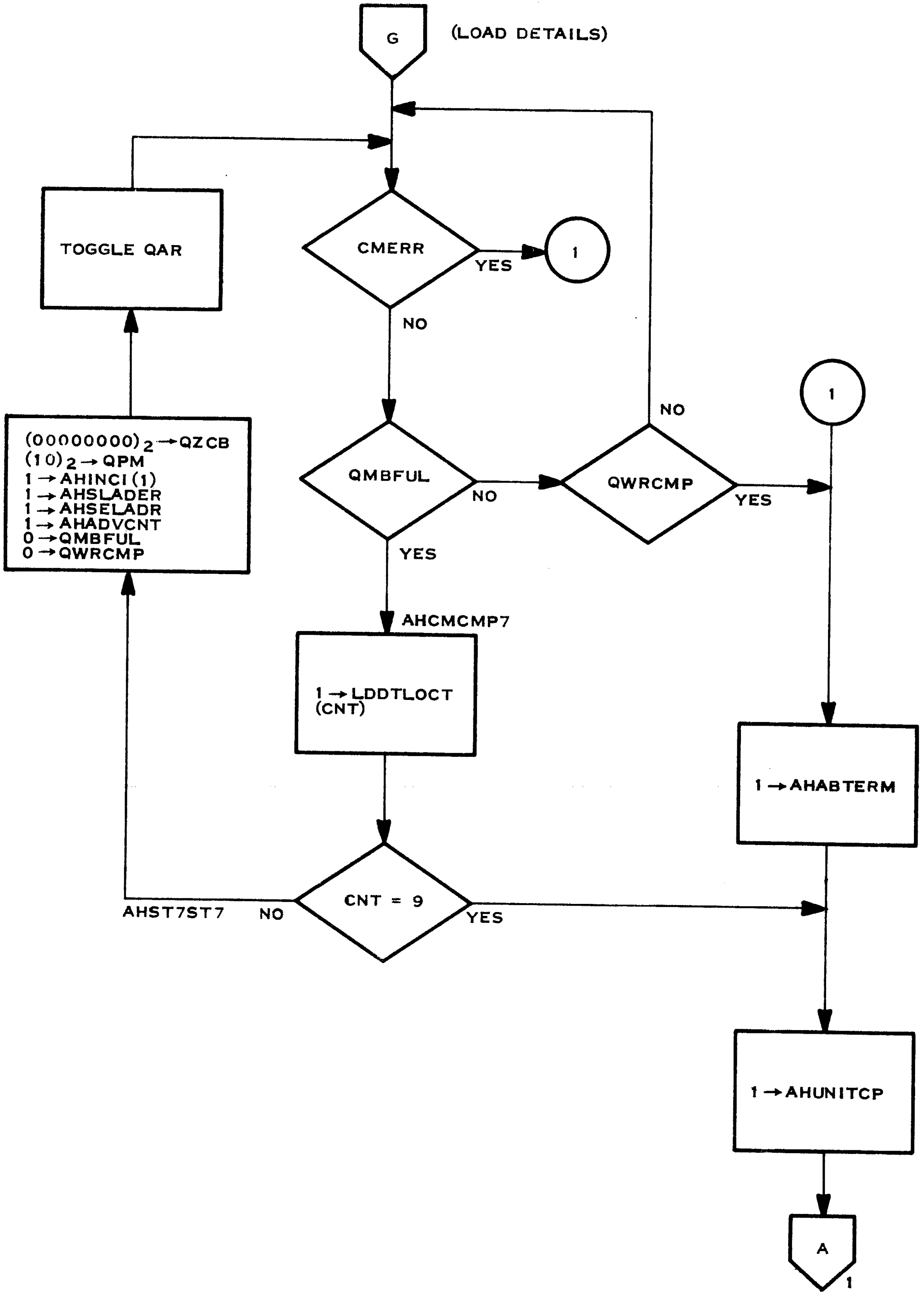
117992

Figure D-21. 4X AU Unit Hardcore Flowchart (Sheet 5 of 7)



117993

Figure D-21. 4X AU Unit Hardcore Flowchart (Sheet 6 of 7)



117994

Figure D-21. 4X AU Unit Hardcore Flowchart (Sheet 7 of 7)

BUMHC AND LOGCLK-8 LOGIC CARDS

During early stages of CPU Check-Out, sub-units of the CPU can be checked "off line". For instance the IPU can be checked alone, as can any pipe (MBU/AU pair). In this case, since the function pipe station tester requires a master hardcore interface, the BUMHC card (a X1 Master Hardcore card) is used to provide this interface.

When pipe check-out is complete, the BUMHC card can be removed from the machine: it is no longer required for proper pipe operation.

Similarly, LOGCLK-8 logic card provides "MASTER CPU Clock" for the pipe during pipe check-out and may also be removed after off line pipe check-out.

The following paragraphs and flowcharts are provided for pipe check-out, hardcore understanding.

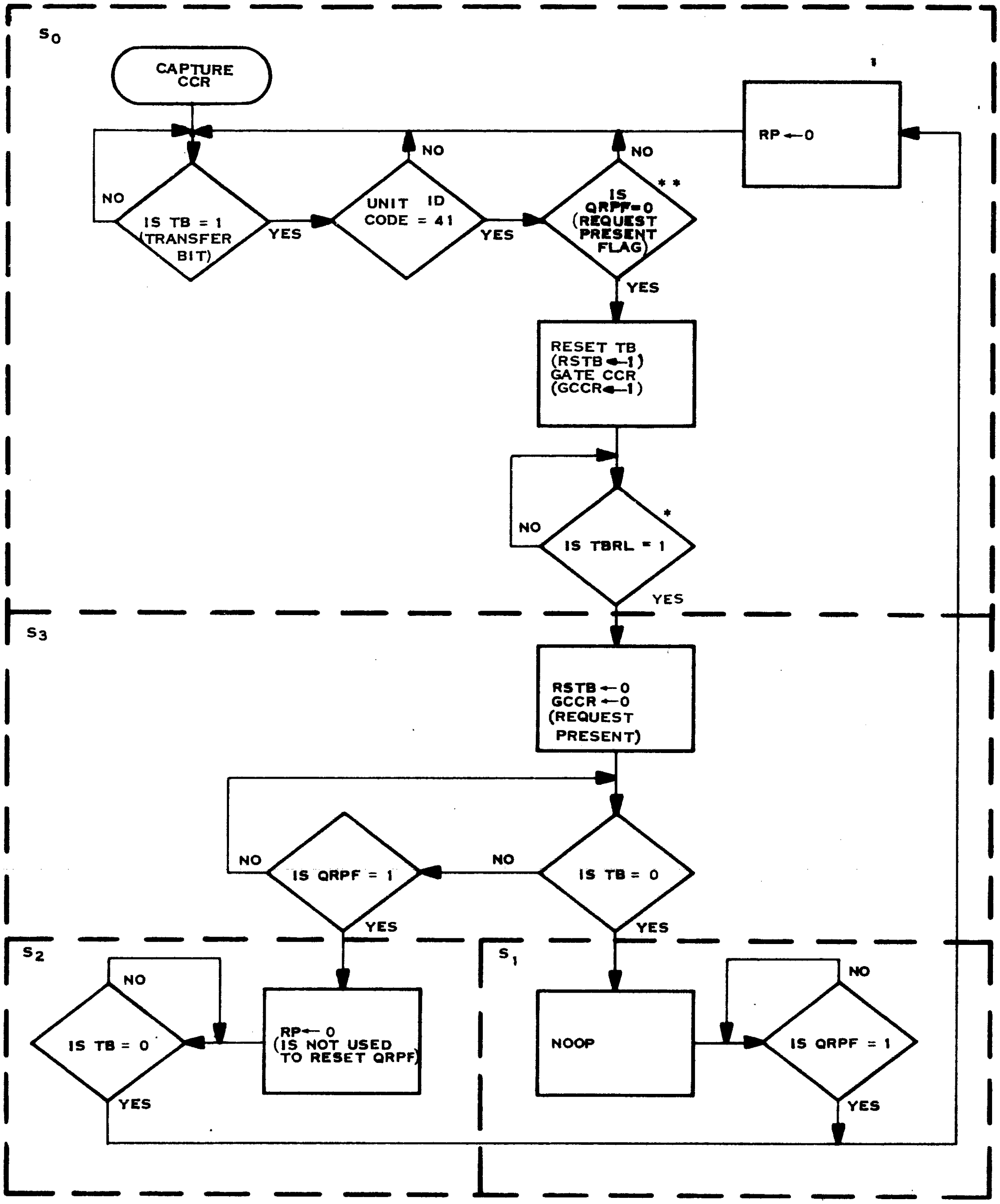
CAPTURE CCR

The capture CCR logic is part of the master hard core circuitry that monitors the transfer bit (TB) and unit code of the Common Command Register (CCR) in the Peripheral Processor Communications Register File. The flowchart in figure D-22 illustrates the control cycle. If TB sets, the PP is issuing a command to one of the system devices. The controller then inspects the unit code of the command to determine if the command is intended for the CP. A code of 41 (in hexadecimal) specifies a CP command. The controller then inspects the Request Present flag (QRPF) to determine if another request is currently being processed by the hard core logic. If this flag is set, the controller must wait until it clears before proceeding. When the flag clears, the controller activates the Reset TB (RSTB) and Gate CCR (GCCR) lines to the CR File to transfer the new command into the CP. When the PP returns a recognition of the transfer (TBRL), the controller deactivates the two signals and activates the Request Present indicator (RP) to indicate to the hard core logic that a new command is resident. The controller then ensures that the TB has not yet reset.

NOTE

> Due to the long clock period of the CR File with respect to the CP, the CP has several clock periods before TB resets. Therefore, if TB is reset at this point, it should not have been set. This feature also provides a time-out that negates the CCR command if QRPF does not set within a reasonable time.

If TB is still set, the controller waits for the hard core logic to set QRPF as a result of RP being active. QRPF indicates an active command in the hard core logic. When QRPF sets, the controller drops RP and waits for TB to clear. When TB clears, the cycle is complete and the controller is ready to begin the cycle again.



NOTE: THIS IS ALL ASYNCHRONOUS LOGIC. (1→ XXX REQUIRES NO CLOCK)

*TBRL –LATCH OUTPUT ON CCR COOKIE SHEET THAT LOOKS AT ALL TB RESET LINES AND WHEN TBR CLK OCCURS IT GETS SET SO THAT THE MHC KNOWS HOW LONG TO HOLD RSTB = 1 DUE TO CCR 500 NS CLOCK

* *QRPF – GETS RESET VIA COMMAND COMPLETE IN SYNCHRONOUS LOGIC STATE 7.

(A)115837

Figure D-22. Capture CCR Logic Flowchart

ERROR MONITOR

The error monitor logic, illustrated in figure D-23, determines the conditions in the CP hard core interface and sets the reason code bits to the PP to indicate the status of a context switch in the CP. If an error occurs in the program currently executing in the CP, the program should be switched out of the CP and a new program loaded into the CP to conserve processor time. The monitor, therefore, checks the SC and MC bits from the PP. If either of these bits is set, then the PP has not as yet recovered from the last program switch to ready a new program for the current switch. The controller sets the reason code bits to "011" to indicate this condition. If however, the PP has prepared a new program, the controller examines the AS control bit from the PP. If Allow Switch is a zero, the PP is prohibiting a context switch. The controller sets the reason code bits to "111" to indicate that an error has occurred, but has not been switched out of the CP. If AS is not zero, the controller clears the reason code bits and allows hard core logic to initiate the context switch.

The decision paths are similar for MCW or MCP instructions in the program sequence. These instructions are also dependent upon Allow Call (AC) being set, so an inspection of that bit is also made. Since an MCP does not initiate an immediate switch of programs, the AS bit does not have to be active for successful completion of the instruction. If no error, MCW or MCP is encountered during the control cycle, the controller clears the reason code bits and returns to the start of the control cycle for the next clock.

BUMHC SEQUENCE CONTROL

Sequence control monitors the program progress in the CP, initiates context switches when errors occur, decodes CCR commands from the CR file and issues commands to the unit hard core controllers to execute the commands. Before initiating any operation, sequence control examines the control byte from the CR file to determine if that operation is permitted. The sequence control flowchart appears in figure D-24 and table D-5 defines the acronyms used in the flowchart.

STATE 0

State 0 of sequence control monitors the conditions in the CP during normal processing and waits for a CCR command from the PP. If a program error, MCW, MCP or system error occurs during normal processing, the controller exits to the state that performs the required steps for that condition. If a CCR command enters from the PP, the controller decodes the command and exits to the state required to initiate that command. During normal operation, the controller monitors the Run Bit to ensure that normal operation is in progress. It then checks the System Error indication and exits to state 1 if a system error has occurred. If no system error occurs, the controller checks AUTO from the Error Monitor circuit. If this signal is active, the controller determines which condition caused AUTO (by examining the other indicator bits from the Error Monitor) and exits to the proper state. If AUTO is not set, the controller examines QRPF. If this flag is set, then the Capture CCR logic has detected and captured a CCR command from the PP. The controller determines

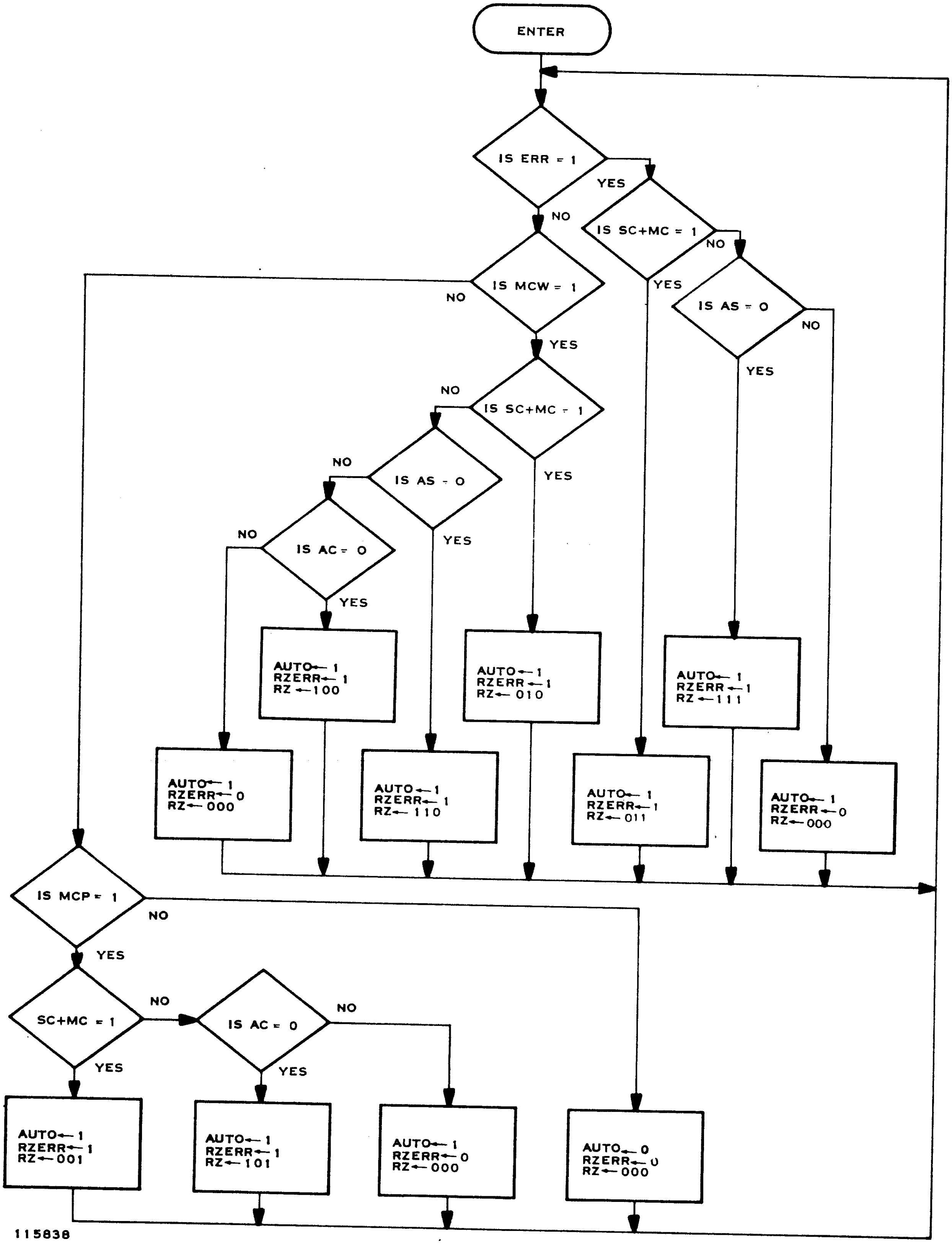


Figure D-23. Monitor Flowchart

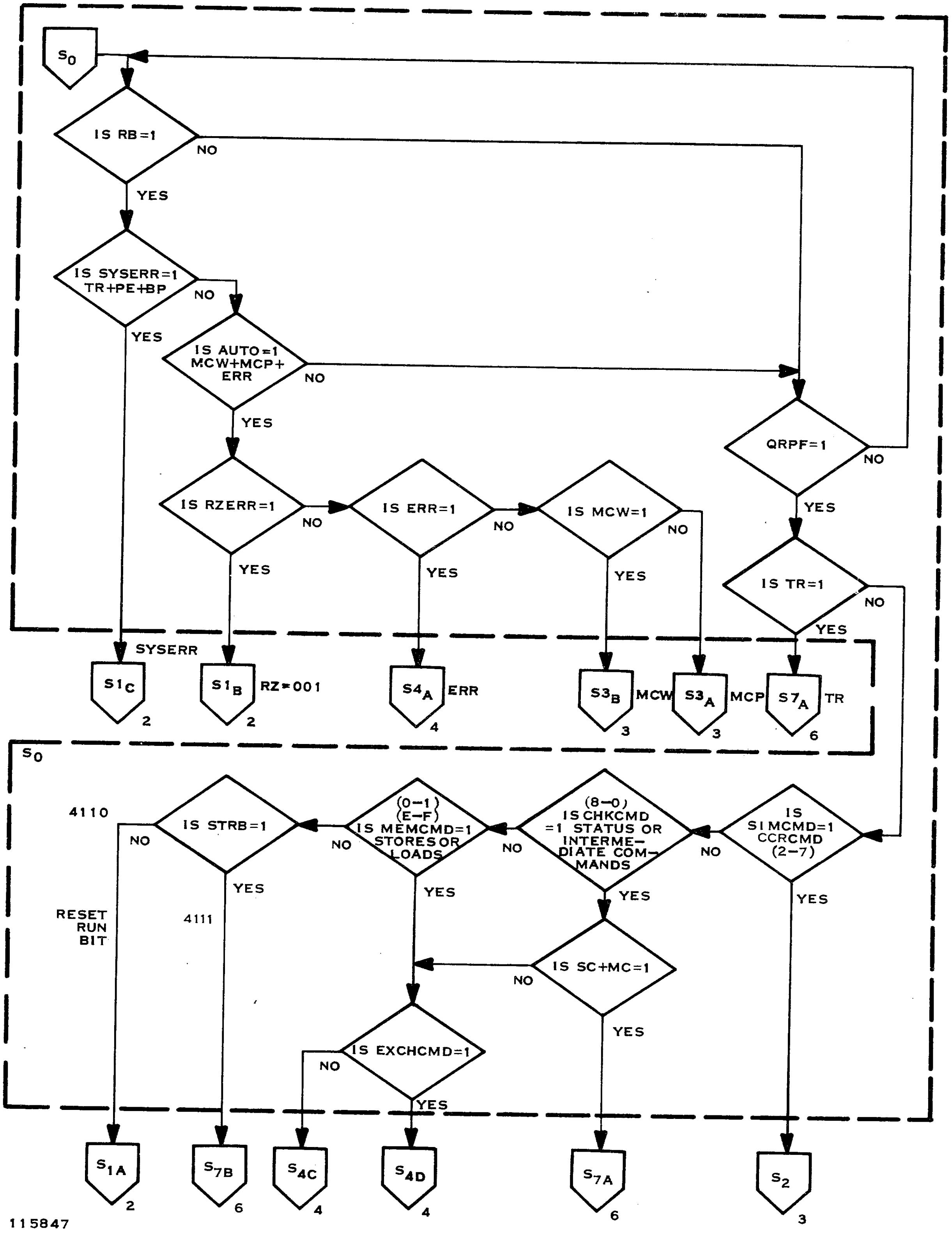
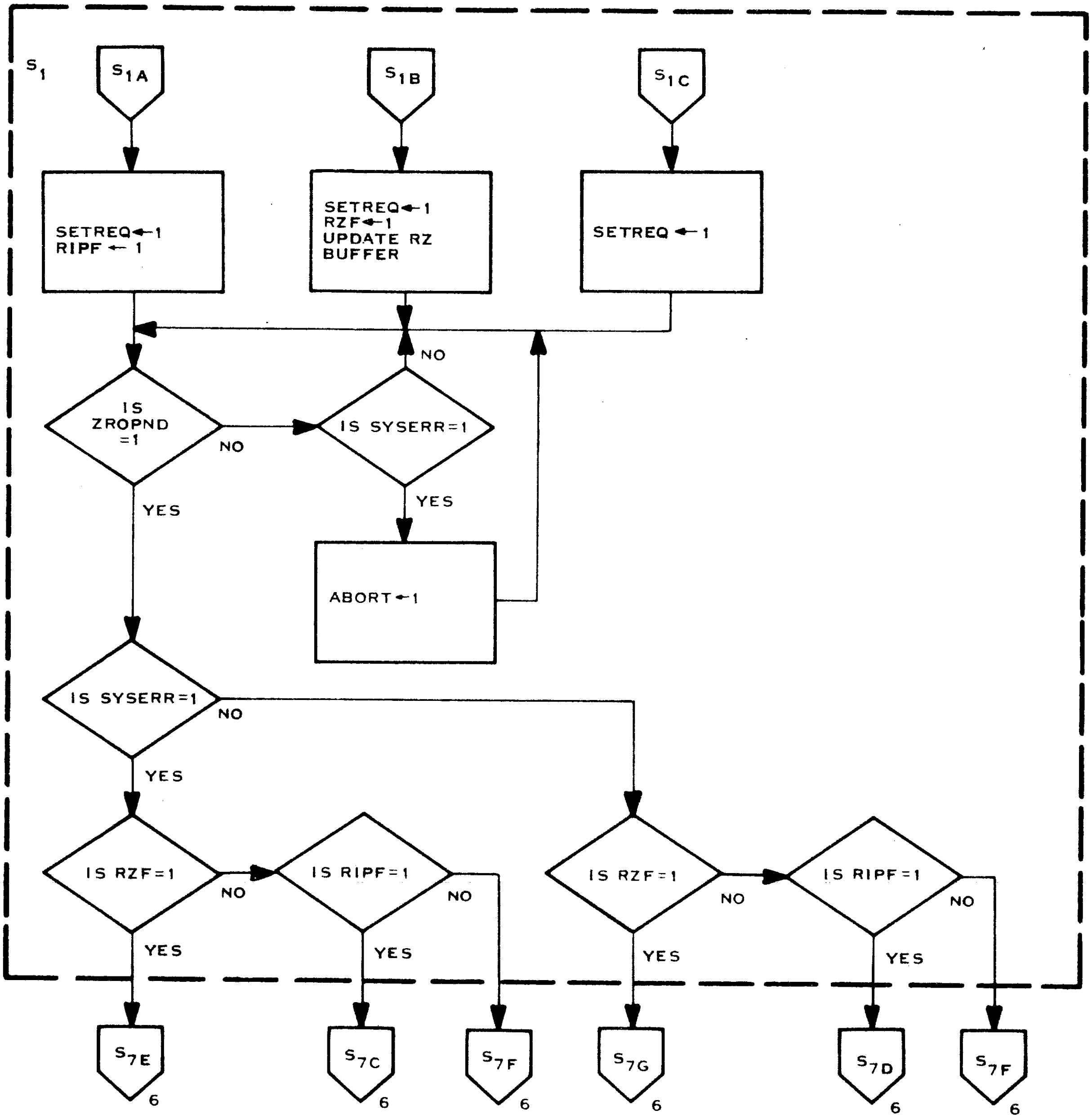
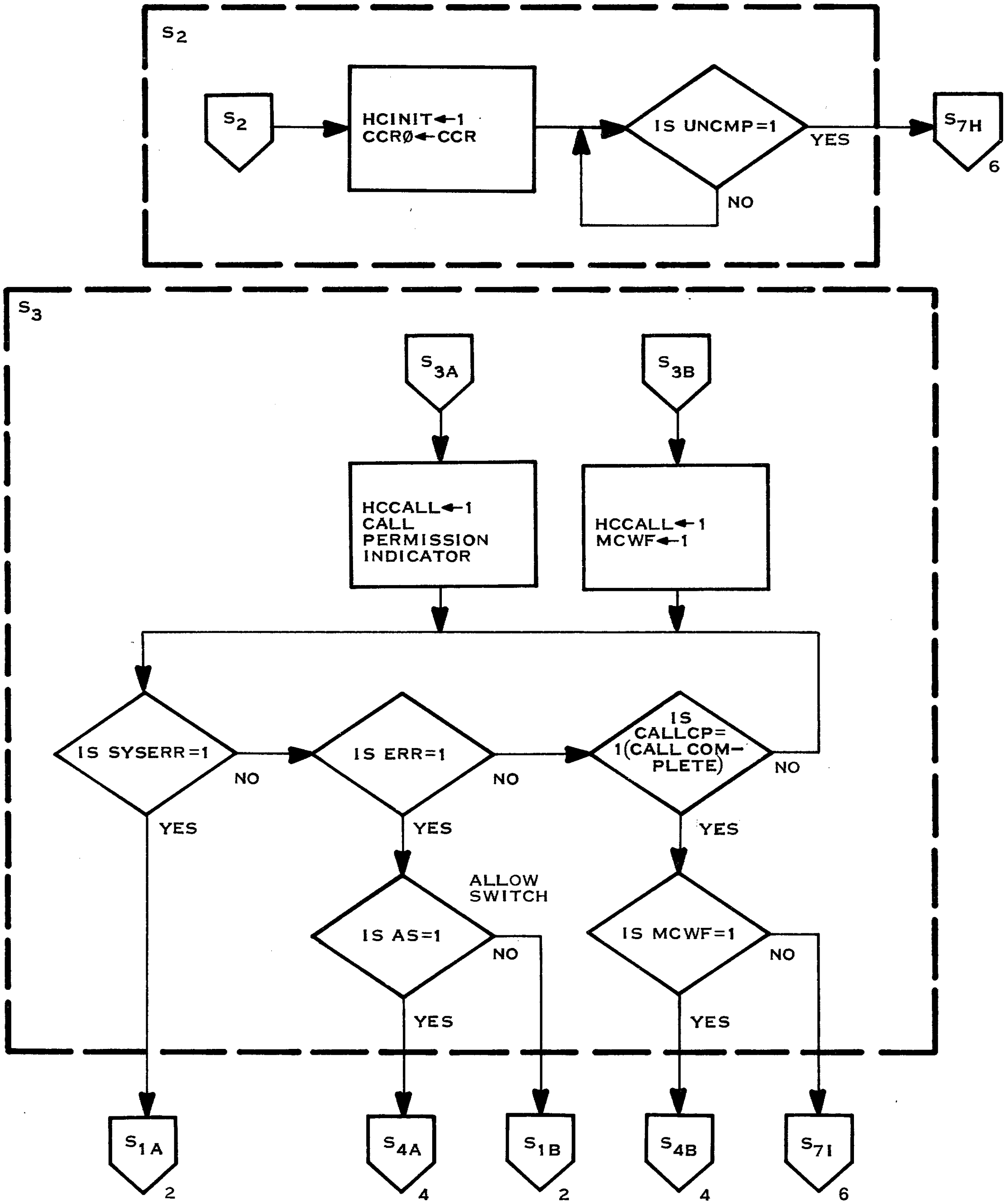


Figure D-24. BUMHC Sequence Control Flowchart (Sheet 1 of 7)



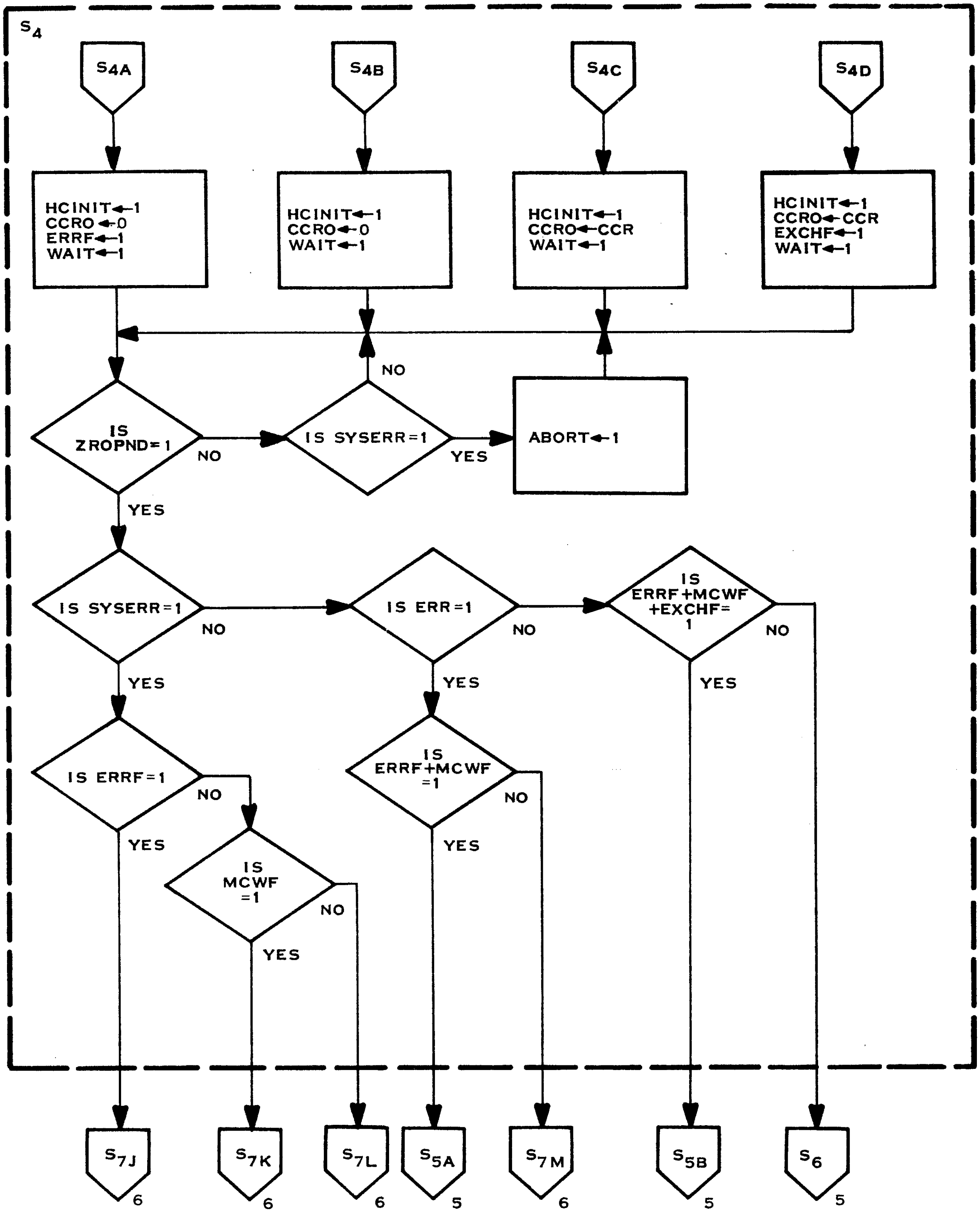
115848

Figure D-24. BUMHC Sequence Control Flowchart (Sheet 2 of 7)



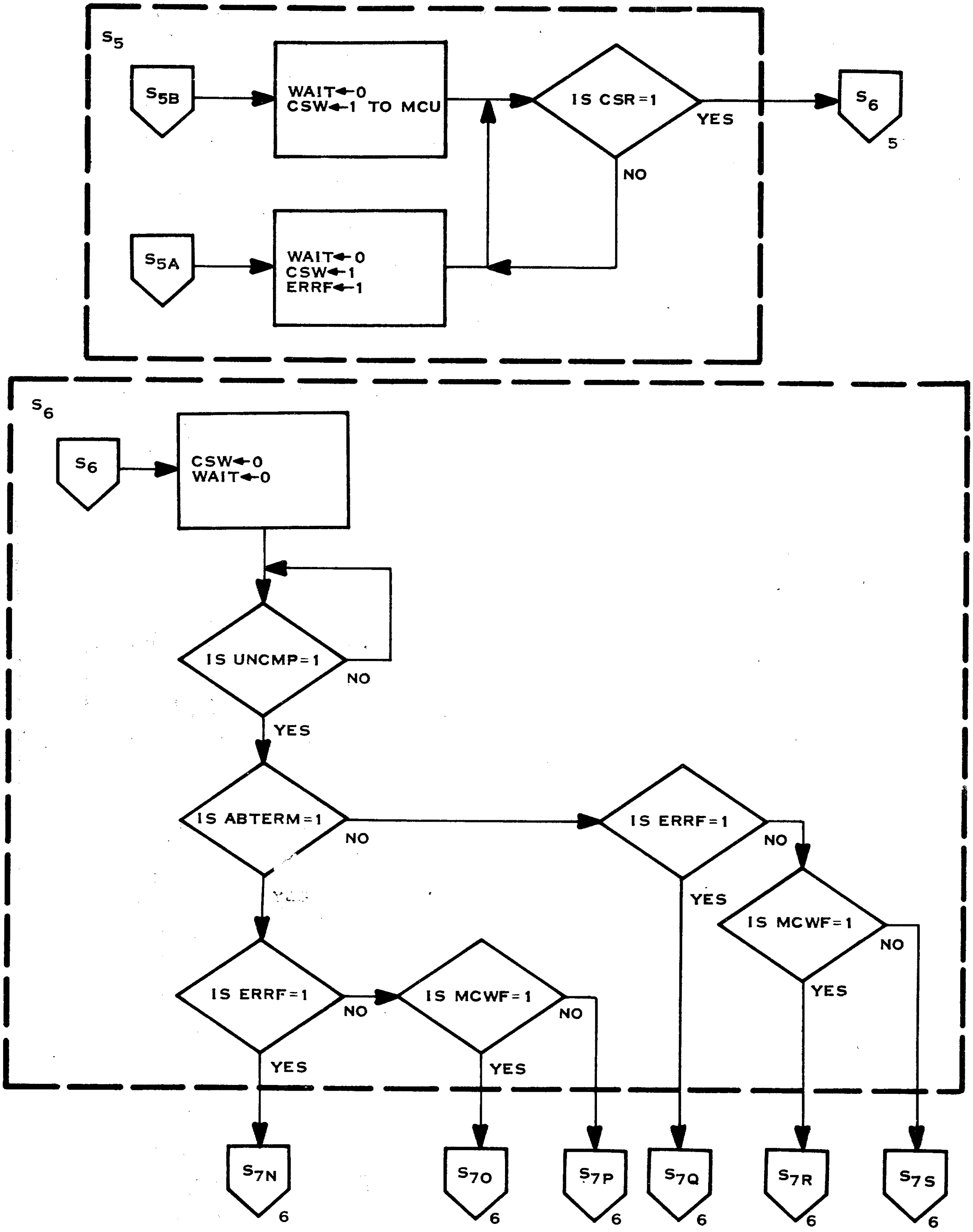
115849

Figure D-24. BUMHC Sequence Control Flowchart (Sheet 3 of 7)



115850

Figure D-24. BUMHC Sequence Control Flowchart (Sheet 4 of 7)



115851

Figure D-24. BUMHC Sequence Control Flowchart (Sheet 5 of 7)

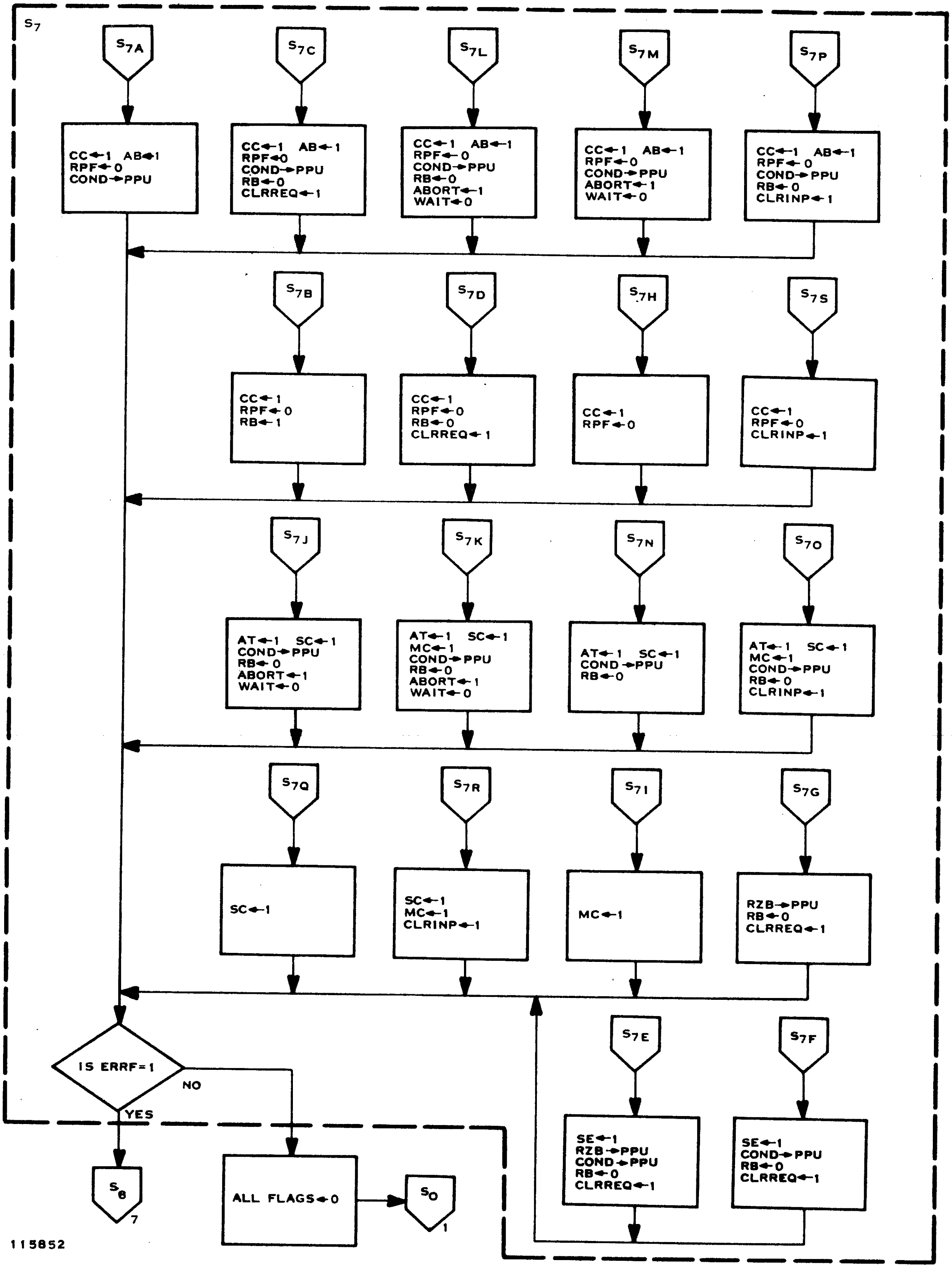
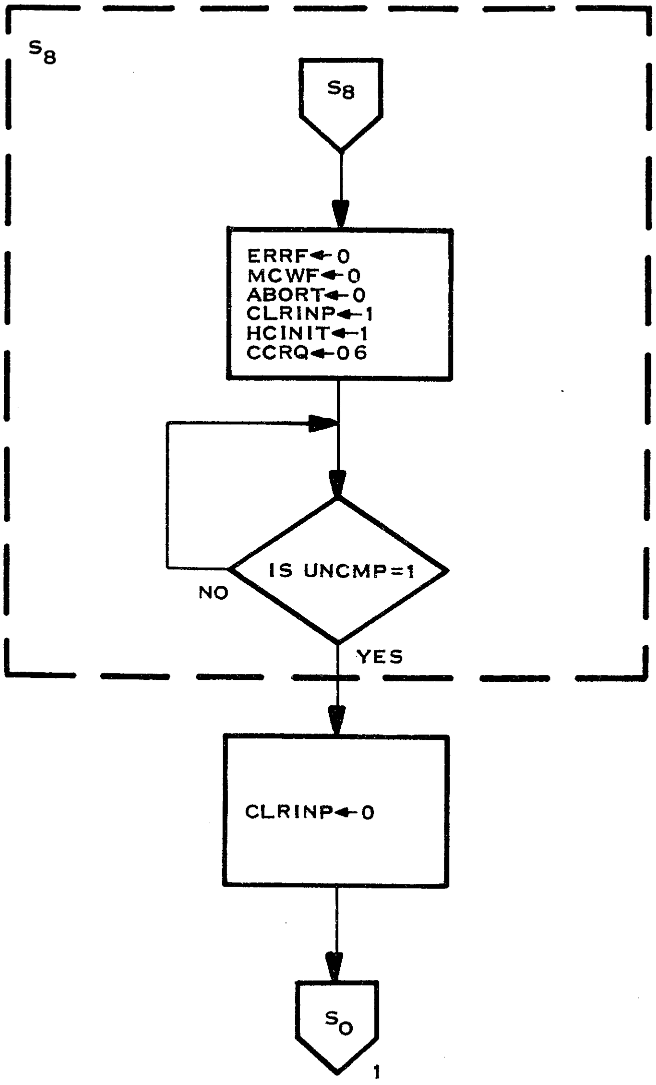


Figure D-24. BUMHC Sequence Control Flowchart (Sheet 6 of 7)



115871

Figure D-24. BUMHC Sequence Control Flowchart (Sheet 7 of 7)

Table D-5. Sequence Control Acronyms

Term	Function
AB	Abnormal - Condition byte bit to PP that indicates that the last CCR command terminated abnormally.
ABORT	Signal to unit hard core controllers that prevents them from further processing of any CCR command.
ABTERM	Signal from the unit hard core controllers that indicates an abnormal termination of a unit command.
AS	Allow Switch - CP Control Byte bit from PP that enables automatic switching for MCW and errors.
AT	Abnormal Termination - Response Byte bit that indicates to the PP that a switch or call terminated abnormally.
AUTO	Signal from Error Monitor that indicates an MCW, MCP or an error condition in the CP.
CC	Command Complete - Condition Byte bit that indicates the completion of the last CCR command.
CCRO	CCR Output Register - Register in master hard core that supplies instruction codes to unit hard core controllers.
CHKCMD	Check Command - Internal signal indicating that the CCR command is either a status or intermediate maintenance transfer (CCR codes 4108 through 410D).
CLRINP	Signal that clears the Hard Core In Progress flag in the unit hard core controllers.
CLRREQ	Clears the unit hard cores Hard Core Requirement flag.
CSR	Context Switch Response - signal from PP indicating that map and protect parameters are set up in MCU for a switch.
CSW	Context Switch - signal to PP indicating a requirement for new map and protect parameters in the MCU for a context switch.
ERRF	Internal flag that indicates a program error switch is being performed.
EXCHCMD	Exchange Command - internal signal that indicates that the CCR command involves both a load and a store (exchange) - CCR codes 410A and 410D.
EXCHF	Internal flag that indicates that the hard core logic is processing an exchange command from the PP.

Table D-5. Sequence Control Acronyms (Continued)

Term	Function
HCCALL	Call permission indicator to level 3 controller to enable IPU to write the call pointer.
HCINIT	Hard Core Initiate - Starts hard core operation in the unit hard core controllers.
MC	Message Complete - Response Byte bit that indicates that the CP has completed an MCW or MCP.
MCWF	Internal flag indicating that the controller is performing an MCW operation.
MEMCMD	Internal signal that indicates that the CCR command is either a load or store operation - CCR codes 4100, 4101, 410E and 410F.
QRPF	Request in Progress flag - Set by the Capture CCR logic indicating that a CCR command is active in the CP. Cleared at the completion of command by Sequence control.
RB	Run Bit - When set, enables CP processing.
RIPF	Internal flag indicating that the command in progress will reset the run bit.
RZF	Internal flag indicating that an error reason code will be sent to the PP.
SC	Switch Complete - Response Byte bit indicating that the CP has completed an MCW or error switch.
SE	System Error - Response Byte bit indicating that a parity error occurred during normal processing.
SETREQ	Set Hard Core Requirement - Sets the Hard Core Requirement flags in the unit hard core controllers so that the units will wind down operations in preparation for a maintenance command.
SIMCMD	Simple Command - Internal signal indicating that the CCR command is a basic command requiring no complex operations - CCR codes 4102 through 4107.
STRB	Set Run Bit - Internal signal indicating a Set Run Bit CCR command.
SYSERR	System Error - Internal signal indicating that a parity error has occurred during CP processing or the PP has set TR.
TR	Terminate Request - Control Byte bit that instructs the CP to cease processing the current CCR command.

Table D-5. Sequence Control Acronyms (Continued)

Term	Function
UNCMP	Unit Complete - Internal signal that indicates that all CP unit hard core controllers have completed their operations for the current CCR command.
WAIT	Prevents memory requests to keep the CP in a zero request pending state.
ZROPND	Zero Pending - Indicates that all memory requests have been satisfied by returning data from memory (no outstanding requests).

that the PP has not terminated the request, and then decodes the command. If the command is a simple one-step command (lock or unlock PC, set or reset all registers, or reset error cells), the controller exits to state 2 to enable the unit hard core controllers to perform their functions. If the command is an intermediate or status command, the controller ensures that the CP has not recently completed a context switch that the PP has not recognized (SC or MC set). If this is the case, the program that the PP requested status or intermediate information for is no longer in the CP. The controller, therefore, sets command complete, abnormal termination, clears the request present flag and transfers the condition byte to the PP. If the CCR command is a load or store operation, the controller exits to state 4 to process the request. If the command is a Set Run Bit, the controller sets the Run Bit and command complete, and clears the Request Present flag. If none of the above commands are present, the controller defaults to a clear run bit command which is performed in state 1.

STATE 1

Four conditions cause the controller to enter state 1. These conditions are:

1. System error during normal operation.
2. Error reason code to be sent to PP.
3. Reset Run Bit command from PP.
4. System error during a call operation.

For each of the above causes, the controller sets the Hard Core Requirement flag in each of the unit hard core controllers so that they will conclude the operations that are currently being processed. If a reason code is to be sent to the PP, the controller also updates the reason code in the reason buffer so that the correct information will be sent to the PP, and sets the RZF flag to indicate that the code will be sent. The RIPF flag is set if the Run Bit will be cleared by the command in progress. The controller then waits for ZROPND to indicate that all outstanding memory requests have returned from memory. During the waiting period, the controller monitors the system error indicator and disables the unit hard core controllers if a system error occurs. The controller then decodes the states of the three indicators; SYSERR, RZF and RIPF, to determine what actions to perform. RZF and RIPF are mutually exclusive flags, whereas, SYSERR could have occurred while waiting for ZROPND. The decode and the resulting actions are listed in the flowchart (figure 4-45). The actions are enabled for the next clock following the decode (state 7).

STATE 2

The controller enters State 2 when it decodes the CCR command from the PP to be a simple operation requiring no complex transfers in the CP. The controller then issues HCINIT to the unit hard core controllers to start each of the units into their respective sequences, and gates the CCR code into the master hard core's CCR Output register. The controller then waits until each unit hard core controller returns an operation complete indication before it exits to state 7 to issue Command Complete to the PP and clear the Request Present flag (QRPF).

STATE 3

The controller enters state 3 during call commands. In this state the controller waits while the level 3 controller writes the call message into the designated location (location 07) in memory. When the controller enters state 3, it sets HCCALL to the level 3 controller to enable it to write the message into memory. If the command is an MCW, the controller also sets the MCWF flag. The controller then waits for Call Complete indicating that the message has been written. During the waiting period, the controller monitors the error indicators to ensure that no system or program errors occur. A system error causes the controller to exit to state 1 to terminate the operation. If a program error occurs, the controller may terminate the operation in state 1 if the Allow Switch bit is not set. If AS is set, the controller exits to state 4 to begin a context switch that loads a new program into the CP. If the message is written into memory without error, the controller examines the MCWF flag to determine if it should perform a context switch (MCW), or if it should continue to process the same program (MCP). For an MCP, the controller exits to state 7 to set Message Complete to the PP. For an MCW the controller exits to state 4 to begin the context switch.

STATE 4

The controller enters state four under four conditions, each of which requires some type of memory transfer (load, store or exchange). When the controller enters state 4, it sets HCINIT prepare the unit hard core controller for the transfer operation, sets WAIT to prevent them from initiating any memory requests, and transfers a code into the CCR Output register for transfer to the UHC's ("0" for context switches, or CCR code for the direct CCR commands). The Exchange flag sets if the operation is an exchange. The controller then waits for all outstanding memory requests to return from memory. During the waiting period, if a system error condition occurs, the controller sends Abort to the unit hard core controllers to halt performance of the CCR command. When ZROPND becomes active, the controller ensures that no system errors have occurred, that a program error has not occurred, or if ERR is set, that the operation is a context switch (ERRF or MCWF). The controller then exits to state 5 for context switch or exchange operations and to state 6 for load or store operations.

STATE 5

The controller enters state 5 to perform an exchange of CP contents. In this state, the controller sends CSW to the PP to ensure that the map and protect parameters have been established in the MCU. When the PP responds with CSR, indicating that the parameters for the new program are ready, the controller exits to state 6. When the controller enters state 5, it also clears the WAIT flag so that the unit hard core controllers can begin the store portion of the exchange while the sequence controller is waiting for CSR.

STATE 6

When the controller enters state 6, it clears the WAIT flag and CSW to the PP if CSW was enabled in state 5. Clearing WAIT allows the unit hard core controllers to perform the designated memory transfer operation. For context switches, the new program is loaded while the controller is in state 6; for loads or stores, that operation is performed in state 6. When each of the unit hard core controllers has completed its transfer operation, the controller determines if any of the units terminated abnormally and issues the status and transfer commands required for each of the conditions. These commands are illustrated in the flowchart for sequence control (figure D-24).

STATE 7

State 7 enables the status and transfer commands that the controller has determined are required (through examination of the operations in the other states of the controller). These transfer commands and status reports are illustrated in the sequence control flowchart. After enabling these commands, the controller determines if the operation completed by hard core was an error context switch (ERRF). If not, the controller clears all control flags and returns to the initial monitor cycle in state 0. If ERRF is set, the controller exits to state 8 to reinitialize the CP hardware flags.

STATE 8

The controller enters state 8 from state 7 after completion of an error switch operation. Since an error produced the switch operation and a new program has entered the CP, the system error cells in each of the CP units must be cleared so that they will not affect the operation of the new program. To produce this effect, the controller loads a code of "06" (Reset system error cells) into the CCR Output register, and enables the unit hard cores by setting Hard Core Initiate (HCINIT). The controller remains in state 8 until the unit hard core controllers have completed the operation (UNCMP). The controller then returns to the initial monitor cycle in state 0.

Table D-6. 4XCP UR Dump Interpretation

Byte	Bit	Signature	Description
		MASTER HARDCORE	

0	0-7	STATE(1-8)
		I4STAT(1-8)
			One bit each for states 1 through 8 of the sequence controller. State 0 is not represented in UR Dump.

1	0-3	CCR(12-15)
		I4QCCRI(12-15)
			Last hex of last CCR command sent to the CP.

1	4-7	CCR(12-15)
		I4QCCRB(12-15)
			Last hex of last CCR command sent to unit hardcores.

2	0	RPF I4QRPF	Request present flag is a bit set by the CCR loader to inform the sequence controller a new CCR command is to be performed. Reset by the sequence controller when it completes the command.

2	1-2	RP(0, 1)
		I4QRP(0, 1)
			State flip-flops of the CCR asynchronous loader. Responsible for setting RPF and resetting the transfer bit in the CR file.

2	3	GCC I4QGCC:1	Gate command complete. Set by the sequence controller to set the command complete bit in the CR file to indicate completion of a CCR command. Also gates the AB bit (CR 12 Bits 0 + 1).
2	4	GSE I4QGSE:1	Gate system error. Set by the sequence controller to set the system error bit in the CR file to indicate terminate request, memory protect violation or memory parity error.
2	5	GAT I4QGAT:1	Gate attention set by the sequence controller to gate the attention, switch complete and message complete bits into CR 12 byte 0, bits 1-3.

MHC consists of the Sequence Controller, the Reason Error Encoder, and the CCR Asynchronous Loader.

Table D-6. 4XCP UR Dump Interpretation (Continued)

Byte	Bit	Signature	Description
2	6	GCB I4QGCB:1	Gate condition byte set by the sequence controller to gate the CP condition bits into CR 12 byte 2 bits 2-6; ME, PE, IL, AE, and PV.
2	7	GRZ I4QGRZ:1	Gate reason code set by the sequence controller to gate the reason code into CR 12 byte 0 bits 5-7 as set up by the reason error encoder.

3	0	CCRI(11)
		I4QCCRI(11)
			CCR input register bit 11, buffers fact that a set or reset run bit CCR command was sent to CPU (the MHC).

3	1	RPLY I4QRPLY:1	Reply set to indicate message complete or switch complete, or both, has been set in CR file 1 bits used to develop attention.
3	2	HCINIT I4QHCINIT	Hardcore initiate used to start hardcore operation in unit hardcore controllers.
3	3	ABORT I4QABORT	Abort signal to unit hardcores to stop CCR processing by cleaning UHC's.
3	4	ZROPN I4QZROPN	Zero pending indicates no outstanding memory requests.
3	5	MHCCMP I4MHCCMP	Master hardcore complete. An "AND" of all unit hardcore complete signals.
3	6	MHCABT I4MHCABT	Master hardcore abnormal termination, an "OR" of all unit hardcore abnormal termination signals.

Table D-6. 4XCP UR Dump Interpretation (Continued)

Byte	Bit	Signature	Description
3	7	CSR I4QCSR	Context switch reply signal sent from the MCU to indicate the map and protect registers are valid and it is alright to proceed with the load half of an exchange command.
4	0	ERR I4ERR	Error indicates an illegal op code, memory parity error, memory protect violation, or arithmetic exception and the run bit was on.
4	1	ERRF I4QERRF	Seb by sequence controller to retain fact that ERR was true and no condition exists that requires software or manual intervention.
4	2	SYSERR I4SYSERR	Master hardcore system error, "OR" of memory parity and protect error signals from unit hardcores and terminate request.
4	3	AUTO I4AUTO	Signal generated by ERR or a monitor call used to interrupt normal operation.
4	4	RZF I4QRZF	Reason flag set by the sequence controller to indicate that a reason code other than zero has been generated.
4	5	RIPF I4QRIPF	Reset in progress flag indicates the command in progress will reset the run bit.
4	6	EXCHF I4QEXCHF	Exchange flag set by sequence controller to retain the fact that it is processing an exchange CCR command.

Table D-6. 4XCP UR Dump Interpretation (Continued)

Byte	Bit	Signature	Description
4	7	EXCMD I4QEXCMD	Exchange command set by the sequence controller after the store portion of an exchange command to retain the fact that it has started the load operation and will not reinitiate it in state 6.
5	0	AT I4QAT:1	Abnormal termination indicates an error condition occurred during a switch or call operation.
5	1	AB I4QAB:1	Abnormal set by sequence controller to indicate CCR command terminated abnormally sets CR 12 byte 2 bit 1.
5	2	INTRP I4QINTRP	Interrupt generated by sequence controller when an error or monitor call condition exists and can be handled without software or manual intervention.
5	3	VBG I4QAVBG	Any vector bad buy signal generated by the level 3 controller to indicate that a vector is being executed that cannot be restarted except as a new instruction.
5	4	MCP I4QMCP	Monitor call and proceed indicates an MCP instruction is being processed.
5	5	MCW I4QMCN	Monitor call and wait indicates that an MCW is being processed
5	6	MCWF I4QMCWF	MCW flag - a flag set in sequence controller state 3 to retain the fact that an MCW is in progress.
5	7	CLCMP I4QCLCMP	Call complete set by the level 3 controller to inform master hardcore that the message has been stored on a monitor call operation.

Table D-6. 4XCP UR Dump Interpretation (Continued)

Byte	Bit	Signature	Description
6	0	PRV I4QPRV	Protect violation. "OR" of unit protect violations from IPU and MBU's.
6	1	PAR I4QPAR	Memory parity error. "OR" of unit parity errors from IPU and MBU's.
6	2	ILLO I4QILLO	Illegal operand "OR" of I4Q1PIOP from IPU and BHQLLOPR(N) from MBU's.
6	3	AREXC I4QAREXC	Arithmetic exception from IPU.
6	4	MIERR I4QMIERR	MBU or IPU memory error "OR" of I4MEMERR from the IPU and BHCMERR (0-3) from the MBU's.
6	5	AMERR I4QAMERR	AU memory error "OR" of AHQPR(0-3) signals from AU's.
6	6	1CMP I4Q1CMP	IPU's unit complete bit.
6	7	IABT I4QIABT	IPU's abnormal termination bit.

7	0-3	MABT(0-3)
		I4QMABT(0-3)
			MBUS' abnormal termination bits.

7	4-7	AABT(0-3)
		I4QAABT(0-3)
			AUS' abnormal termination bits.

8	0-3	MCMP(0-3)
		I4QMCMP(0-3)
			MBUS' unit complete bits.

8	4-7	ACMP(0-3)
		I4QACMP(0-3)
			AUS' unit complete bits.

Table D-6. 4XCP UR Dump Interpretation (Continued)

Byte	Bit	Signature	Description
9*	0	PV I4QPV	Protect violation CR 12, byte 2, bit 6.
	1	PE I4QPE	Parity error CR12 byte 2, bit 3.
	2	1L I4Q1L	Illegal op code CR 12, byte 2, bit 4.
	3	AE I4QAE	Arithmetic exception CR 12, byte 2, bit 5.
	4	ME I4QME	Memory error CR12, byte 2, bit 2.

	5-7	RZ(0-2)
		I4QRZ(0-2)
			Reason code CR 12, byte 0, bits 5-7.

A*	0	TR I4QTR	Terminate request CR A, byte 3, bit 5.
	1	AS I4QA-S	Allow switch CR A, byte 3, bit 7.
	2	AC I4QAC	Allow call CR A, byte 3, bit 6.
	3	SB I4QSB	Load status bit CR 14, byte 3, bit 1.
	4	MC I4QMC	Message complete CR 12, byte 0, bit 2.
	5	SC I4QSC	Switch complete CR 12, byte 0, bit 3.
	6	SS I4QSS	Status stored CR 12, byte 0, bit 4.
	7	RB I4QRB	Run bit CR 12, byte 2, bit 7.

*Bytes 9 and A are identical to the bits in the CR file for each condition.

Table D-6. 4XCP UR Dump Interpretation (Continued)

Byte	Bit	Signature	Description
			IPU HARDCORE
0	0	FREEZE IMQFREEZ	IPU's equivalent of run bit if run bit = 1, freeze = 0.

0	1-7	STATE(0-6)
		IMHCSTA(0-6)
			State flip-flops for IPU hardcore.

1	0-3	LSD(0-3)
		IMQLSD(0-3)
			Low order hex of last CCR command sent to the IPU by master hardcore.

1	4	EXCH IMQEXCH	Exchange indicates that the IPU hardcore has decoded an exchange CCR command.
1	5	HCREQ IMQHCREQ	Hardcore requirement signal to the CM requestor to stop normal request processing.
1	6	HCINP IMQHCINP	Hardcore in progress signal from the master hardcore to indicate it is OK to proceed with CCR command processing.
1	7	LDPTR IMQLDPTR	Load pointer set on transition from State 4 to State 5 to retain the fact that the pointer octet has been loaded reset on transition from State 5 to State 6.
2	0	SPSDW IMQSPSDW	Store program status doubleword. Set to A 1 to gate the program status to IOCM.

2	3-7	OCTR(0-4)
		IMQOCTR(0-4)
			Octet counter. Counter used to indicate what data in the status, details or PSDW is involved in a memory transfer.

3	2	PRM(0)
		ICQPRM:(0)
			Signal developed from IMSFREQ which is used to make a memory request during a STF or STFM instruction.

Table D-6. 4XCP UR Dump Interpretation (Continued)

Byte	Bit	Signature	Description

3	3	PRM(1)
		ICQPRM:(1)
			Signal developed from IMLFREQ which is used to make a memory request during a LF or LFM instruction.

3	5-7	RC(0-2)
		IMQRC(0-2)
			Read counter indicates the number of outstanding read requests to central memory.

4	0	RDA ICQRDA:1*	Q output of 2nd level F/F set by read data available from the MCU.
4	1	RDS ICQRDS:1	Read data sampled to the MCU.
4	2	RA ICQRA:1	Request accepted from MCU.
4	3	AR ICQAR:1	Access request to MCU.
4	4	DAV ICQDAV:1*	Q output of 2nd level F/F set by data available signal from the MCU.
4	5	DAV IOQDAV*	$\overline{Q}$ output of 1st level F/F set by data available signal from the MCU.
4	6	PAR ICQPAR:1*	Q output of second level F/F set by parity error signal from MCU.
4	7	PAR IOQPAR*	$\overline{Q}$ output of first level F/F set by parity error signal from MCU.
5	0	RDA IOQRDA*	$\overline{Q}$ output of F/F set by read data available from the MCU (IORDA).
5	1	WRITE ICQWRITE:1	Write. Indicates a write to memory operation. F/F is clocked by gate signal to OA register.

*Bits from 1st level are asynchronously loaded, 2nd level are synchronized.

Table D-6. 4XCP UR Dump Interpretation (Continued)

Byte	Bit	Signature	Description
5	2	PRV IMQPRV:1	Protect violation which occurred while the freeze bit was set (i.e., a hardcore operation).
5	3	IPPRV ICQIPPRV**	Protect violation which occurred during normal processing.
5	4	PAE IMQPAE:1	Memory parity error which occurred while the freeze bit was set (i.e., during a hardcore operation).
5	5	IPPAE ICQIPPAE**	Memory parity error during normal instruction processing.
5	6	AREX ICQAREX**	Arithmetic exception. A divide check, FX. PT. overflow, FL. PT. overflow, or FL. PT. underflow has occurred and its corresponding mask bit was set.
5	7	IPIDP ICIPIOP**	Illegal op code.

8	0-1	IP(0-1)
		ICQIF(0-1)
			Input pointer for CM request que.

8	2-3	OP(0-1)
		ICQOP(0-1)
			Output pointer for CM request que.

8	4-7	BSY(0-3)
		ICQBSY(0-3)
			CM request que busy bits if set. Each bit indicates that the request it is associated with has been made.

9	0-3	ACT(0-3)
		ICQACT(0-3)
			CM request que active bits if set. Each bit indicates the request it is associated with will be used when it returns from memory.

**Set by lines from Level 3 Controller.

Table D-6. 4XCP UR Dump Interpretation (Continued)

<table>
<tr><th>Byte</th><th>Bit</th><th>Signature</th><th>Description</th></tr>
<tr><td>9</td><td>4-7</td><td>PRV(0-3)
ICQPRV(0-3)</td><td>CM request que protect violation bits.</td></tr>
<tr><td>A</td><td>0-7</td><td>CUE0(M),
CUE1(M)
ICQCUEN(M)</td><td>Encoded bits which give the destination of the data for CM. request queue position M where M = 0, 1, 2, 3 and N = 0, 1. The encoding is done in the following manner:
<table>
<tr><th>CUE0(M)</th><th>CUE1(M)</th><th>Destination</th><th>Operation</th></tr>
<tr><td>0</td><td>0</td><td>KA</td><td>(Load Instructions)</td></tr>
<tr><td>0</td><td>1</td><td>KB</td><td>(Load Instructions)</td></tr>
<tr><td>1</td><td>0</td><td>IR</td><td>(Indirect Address)</td></tr>
<tr><td>1</td><td>1</td><td>Register File</td><td>(Load File)</td></tr>
</table>
</td></tr>
<tr><td>B</td><td>0-7</td><td>OA(8-15)
IHQOA(8-15)</td><td>Contents of OA register bits 8-31, last CM address accessed by the IPU.</td></tr>
<tr><td>C</td><td>0-7</td><td>OA(16-23)
IHQOA(16-23)</td><td></td></tr>
<tr><td>D</td><td>0-7</td><td>OA(24-31)
IHQOA(24-31)</td><td></td></tr>
<tr><td>E</td><td>1-3</td><td>P3(29-31)
IRQP3(29-31)</td><td>Low order 3 bits of P3 register. (The program counter at Level 3).</td></tr>
<tr><td>E</td><td>5-7</td><td>PA(29-31)
ILQPA(29-31)</td><td>Low order 3 bits of PA register. (The present address or program count of the next instruction that will be loaded into the pipe).</td></tr>
<tr><td>0</td><td>0-1</td><td>PM(0-1)
BHQPM:1(0-1)</td><td>Protect mode bits sent to MCU</td></tr>
</table>

Table D-6. 4XCP UR Dump Interpretation (Continued)

Byte*	Bit	Signature	Description
0	2	ZBR BCQZBR:2	Z buffer release - indicates write data has been latched up in MCU and ZB data no longer need to be kept.
0	3	AR BCQAR:1	Access request to the MCU.
0	4	RA BCRA	Request accepted from the MCU.
0	5	RDA BCRDA	Read data available from the MCU.
0	6	RDS BCQRDS:1	Read data sampled to the MCU.
0	7	OAFUL BCQOAFUL:2	OA Full indicates that the MBU's CM address register contains a valid address.

1	0-7	OA(8-15)
		BHQOA:1(8-15)
			Address of last MBU CM request.

2	0-7	OA(16-23)
		BHQOA:1(16-23)	

3	0-4	OA(24-28)
		BHQOA:1(24-28)	

3	5-7	RSTAT(0-2)
		BCQRSTAT(0-2)
			Requestor State 0-2. Outputs of 3 flip-f flops used to encode CM requestor States 0-7 which equals the number of outstanding requests.

4	0-7	OZC(0-7) BHQOZC:1(0-7	Zone control bits associated with MBU requester.

5	0-3	LSD(0-3)
		BHQLSD:2(0-3)
			Low order hex of last CCR command sent to the MBU.

5	4	HCMPRV BHQCMPRV	Protect violation which occurred during a hardcore operation.

Table D-6. 4XCP UR Dump Interpretation (Continued)

Byte*	Bit	Signature	Description
5	5	HPARER BHQPARER	Parity error which occurred during a hardcore operation.
5	6	COMPRV BCCMPRV	Protect violation which occurred during instruction processing.
5	7	CPARER BCPARER	Parity error which occurred during instruction processing

6	0-3	HSTATE(0-3)
		BHQSTATE:1(0-3)
			Output of MBU hardcore state flip-flops used to encode States 0-9.

6	4	HCREQ BHQHCREQ:1	Hardcore requirement. Signal to the MBU CM requester to stop normal request processing and proceed to a zero pending state.
6	5	HCINP BHQHCINP:1	Hardcore in progress signal from the master hardcore indicating that all of the conditions exist that will allow a CCR command to be processed.
6	6	RNEQ0 BHRNEQO	Run bit equal to zero.
6	7	EXCH BHQEXCH:1	Exchange. Indicates an exchange command.

7	0-3	HCCNT(0-3)
		BHQHCCNT:1(0-3)
			Hardcore count 0-3. Outputs of counter used for load and store details.

7	4	UNCMP BHQUNCMP	Unit complete sent to master hardcore.

*Data for bytes 0 through 5 originate on BUCMR or BUCTL2 through a selector located on the BUCAF cards. Bits 0-3 of the selector are on BUCAF(0) and 4-7 are on BUCAF(1). Signature given is on the BUCTLMB as input to the selector.

Table D-6. 4XCP UR Dump Interpretation (Continued)

Byte	Bit	Signature	Description
7	5	ABTRM BHQABTRM:1	Abnormal termination indicates a memory error during a hardcore operation.
7	6	ILOPR BHQILOPR	Illegal operand indicates an illegal operand in the last attempt to interpret the vector parameter file.

8	0-2	DSTATE(0-2)
		BHQDSTAT:1(0-2)
			De-escalate state flip-flops used to decode States 0-7.

8	3	DSCMP BHQDSCMP:1	De-escalate complete set in State 7 of de-escalate controller indicates no activity in the pipe.

8	4-5	WRAP(0-1)
		BHQWRAP:1(0-1)
			Wrap bits used to indicate if a CAF generator pointer has wrapped around ahead of the other pointer necessary for de-escalation. Bit 4 indicates the "A" vector has wrapped around ahead of the "B" vector.

8	6	AVDES(0)
		BCQAVDES:1(0)
			"A" vector de-escalate. Signature used to halt the "A" vector input pointer for de-escalation of vector operation.

8	7	AVDES(1)
		BCQAVDES:1(1)
			"B" vector de-escalate. Signature used to half the "B" vector input pointer for de-escalation of a vector operation.

		AU HARDCORE	

0	0-7	ADDR(8-15)
		AHQADDR:1(8-15)
			Address bits 8-15 from AU4XSEL(0) card for last hardcore address.

1	0-7	ADDR(16-23)
		AHQADDR:1(16-19)
			Address bits 16-19 from AU4XSEL(0).

		AHQADDR:1(8-11)	Address bits 20-23 from AU4SEL(1) for last hardcore address.

Table D-6. 4XCP UR Dump Interpretation (Continued)

Byte	Bit	Signature	Description

2	0-7	ADDR(24-31)
		AHQADDR:1(12-19)
			Address bits 24-31 from AU4XSEL(1) for last hardcore address.

3	0-7	ZCB(0-7)
		AHZCB:1(0-7)
			Zone enable bits for the MBU with current design should all be in the same state since we only store octets.

4	0-1	PM(0-1)
		AHQPM:1(0-1)
			Protect mode bits sent to the MCU.

4	2	AR AHQAR:1	Access request. One level removed from signal sent to the MCU.
4	3	MBFUL AHQMBFUL:1	Memory Buffer Full. Synchronized RDA signal from MCU.
4	4	WRCMP AHQWRCMP:1	Write complete. Synchronized write gate signal from the MCU indicates two way bus is turned and the MCU can accept write data.
4 4	5	PR AHQPR	Project response. Synchronized protect response from the MCU.
4	6	PA AHQPA:1	Parity error. Synchronized parity error from the MCU.
5	0-7	STATE0-7 AHQSTATE:1(0-7)	Output of AU hardcore state flip-flops used one per state.

6	0-3	LSD(0-3)
		AHQCRLSD:1(0-3)
			Last significant digit of last CCR command sent to the AU hardcore.

6	4-7	OCTET(0-3)
		AHQOCTCT:1(0-3)
			Output of octet counter used in load and store details operations.

*CCR commands for CP details (410B-D) operations are recoded to maintenance details commands (410E, F) for the AU only.

TI-22006