NCR/CDC ADVANCED SYSTEMS LABORATORY

IPL-Software Engineering System

09/01/75
REV: B

NCR CONTROL DATA CORPORATION
ADVANCED SYSTEMS LABORATORY

CYBER 70/IPL ISWL

(ISWL/CI)

V2.0 ERS

Submitted: N. H. Peterson 8/29/75

Approved: J. P. Davis 9/4/75

K. White 9/5/75

Doc. No. ASL00344

Rev. B

RECORD of REVISIONS

REVISION	NOTES
A	ERS released. Titled "IPL ISWL {V1.0}".
{4/28/75}	
B	ERS completely updated, obsoleting all previous
{9/1/75}	editions. Title changed to "CYBER 70/IPL ISWL {V2.0}".

AA 3373

1.0 INTRODUCTION

The purpose of this document is to define the ISWL/CI V2.0 language to such a degree that the language can be understood and used by programmers experienced in the use of high-level block-structured languages.

The ISWL/CI V2.0 language is defined by the ISWL V2.0 (ISWL/CC) ERS and by this document.

Other appendices are included in the document to describe the following:

- LANGUAGE DISCREPANCIES
- IPL MACHINE BREAKOUT
- COMPILATION ERRORS
- IPL ISWL RESERVED WORDS
- ASSEMBLY LANGUAGE DESCRIPTION
- COMPILER OUTPUT
- COMPILING SOURCE MODULES
- OBJECT PROGRAM EXECUTION
- MACHINE BREAKOUT EXAMPLE

The reader is urged to read the appendices in the order as listed above from the top to the bottom.

2.0 APPLICABLE DOCUMENTS

The following document is required.

ISWL V2.0 ERS (Rev. A)

The following document references related material with which the reader's familiarity is assumed.

CYBER 70/IPL ISWL V2.0 Design Plan

The following documents reference related material which the reader is urged to read before he attempts to execute an IPL ISWL object program.

IPL Instruction Simulator V2 ERS

Software Engineering User's Handbook (Rev. K)

KRONOS Time-Sharing User's Manual

IPL Processor-Memory Model Independent GDS (Rev. D)

3.0 APPENDIX A - LANGUAGE DISCREPANCIES

This section describes the differences and restrictions of the ISWL/CI V2.0 Language from the ISWL V2.0 ERS and the "IPL Machine Breakout" definition (see appendix entitled, "IPL MACHINE BREAKOUT").

3.1 LANGUAGE DIFFERENCES

- No I/O of any kind is supported.
- Any feature requiring runtime support is not provided. This includes support of exponentiation and full runtime storage manipulation.
- Input will be the KRONOS Rev. C character set.
- Compiler usage, limitations, and error list are peculiar to the product.
- Machine breakout will be supported to the level defined by the proposal of November 15, 1974 by C. W. Schwarcz, entitled "IPL Machine Breakout" (see appendix entitled, "IPL Machine Breakout").
- The word "main" is an optional attribute of a procedure and indicates the first procedure of a program to be executed when so required by the system.
- Constant expressions can only be of type integer when used in constant declarations.
- Pointers to procedures aren't implemented.
- Forward pointer references to adaptable types are not allowed.
- The B comment toggle must occur syntactically first in the program before any non-ISWL text in the source program. Syntax scanning is left to right per source statement.

- Real data type is not supported.
- Sets with 256 elements are supported.
- Type identifier lists are supported in type and constant declarations.
- Substrings can be passed as parameters.
- Begin blocks with local declarations are supported.
- The pointer type test operator is supported.
- For statement increment and decrement is supported.
- The #eof function is not supported.

3.2 LANGUAGE RESTRICTIONS

- No parameter passing is allowed for code (IPL) procedures.
- Machine built-in functions with the exception of the address function must be evaluatable at compile time.
- #BIT type does not malign other fields in a record definition. The fields following a field of #BIT type are aligned to a byte boundary.
- #BIT and #BYTE types are conformable only to themselves.
- #FAST type is ignored and variables of this type are treated as automatic variables.
- #OFB can only be used in a machine statement.
- #OFB and #OFBBIT are not supported for arguments which are constants.
- LOCKREG and PROLOG directives are lexical.
- If a PROLOG directive is used, then it must be the first statement in the procedure.
- If a PROLOG directive is not used, the following restrictions apply.

- The following statements cannot be used:
  - CASE
  - GOTO EXIT
  - procedure/function calls
  - EXIT from procedure/function
  - RETURN
  - FOR
  - BEGIN
  - NEXT
  - RESET
  - CYCLE
  - ALLOCATE
  - VAR declarations for automatic variables
- No return code is generated for the procedure/function.
- #BIT type cannot span 64 bits.
- Locking a "locked" register and freeing a "free" register are illegal.
- In order to initialize types which are either or contain #BIT and #BYTE types, the #M function must be used. In this case, the left most n=(number of bits representing the type definition) bits in the hex character string are used for the initialization.
- If a variable is locked to a register and is one of the following types: #ADDRESS, #DESCRH, or #DESCRF, it is illegal to access a field of the variable.
- An * in a value constructor can only be used for uninitializing elements of a structured variable and not the entire structured variable.
- The fields of a machine data type are not addressable via the #LOC function and the unary pointer operator (^).
- The <bounds list> of an <allocation designator> cannot be of the form <scalar expression>...<scalar expression>.

4.0 APPENDIX B - IPL MACHINE BREAKOUT

The IPL Machine Breakout features of the language are defined by a memo entitled, "IPL Machine Breakout", written by C. W. Schwarcz on November 15, 1974. The contents of the memo are contained in the appendix.

4.1 IPL MACHINE BREAKOUT - (REVISION B)

SWL machine breakout provides facilities for defining both types and variables using machine-dependent types, facilities such as registers, and generating machine instructions through an assembly language facility.

By providing well-defined interfaces between SWL types and statements on the one hand, and machine types and statements (instructions) on the other hand, the amount of machine code can be reduced to the specific functions that the programmer wishes to define in a machine depenent manner. Housekeeping, input-output, memory management, addressing, and flow of control can all be written in SWL itself if the programmer desires.

4.1.1 CODE PROCEDURE

A procedure declared with the CODE attribute can make use of machine data types and machine statements. A CODE procedure does not automatically contain the standard compiler-generated prologue and epilogue. The standard prologue and epilogue can be generated through the use of the PROLOG and EPILOG machine directives.

While there are no restrictions as to the range of SWL statements and declarations in a code procedure, the programmer must ensure that the standard prologue or epilogue has been executed and that the necessary standard SWL environment is in fact if required by machine-independent SWL declarations or statements.

Syntax: add a new procedure attribute.

```
<proc_attribute> ::= <code_attribute>

<code_attribute> ::= CODE(IPL)
```

Note that the code attribute is not part of the procedure type. Thus callers of a procedure need not know whether the procedure being called is a code procedure or a normal procedure (unless, of course, the procedure requires machine-dependent parameters.

4.1.2 CODE BLOCK

A code block is a BEGIN block that can contain machine declarations or statements. It can also contain SHL declarations or statements. The same restrictions defined for code procedures apply to code blocks.

Syntax: replace the definition of begin statement.

```
<begin_statement> ::= BEGIN [[<code_attribute>]]
                      <declaration_list> <statement_list> END
```

Example:

```
nothing: BEGIN [ CODE(IPL) ]
          TYPE
            trans_vector =
              ARRAY [0..0ff(16)] OF #address,
            angus = #descrh;
          VAR trans : trans_vector,
            pu : ^ #address,
            i : 0..ff(16);
          FOR i := 0 TO ending DO
            pu := ^ trans[i]
          FOREND
          END nothing;
```

4.1.3 CODE MODULE

A code module is a module that contains machine declarations potentially including code procedures.

Syntax: replace module declaration.

```
<module_declaration> ::= MODULE [[<code_attribute>]]
                         [<module_identifier>] [(<prongs>)]
                         <declaration_list> MODEND
                         [<module_identifier>]
```

Example:

```
MODULE [CODE(IPL)] mover;
  PROC move (VAL from, to: #ADDRESS)
    VAR george : ARRAY [0..10] OF #BYTE(8,8);
    .....
  PROCEND move;
MODEND mover;
```

4.1.4 CODE ATTRIBUTE INHERITANCE

The code attribute follows normal SHL block structure. Thus any module, procedure, or block declared within a code module, procedure, or block also implicitly has the code attribute.

4.1.5 MACHINE DATA TYPES

Machine data types are provided to allow the use of machine dependent data within structured types, variable declaration, and most other nice places.

```
<machine_type> ::= <byte_type> | <bit_type> | <descriptor_type> |
                   <address_type>
```

Machine types are available in the form of built-in type declarations.

4.1.5.1 Byte_Type

A byte type defines a type that is allocated on a byte (eight bit) boundary and is an integral number of bytes long. The type definition can specify a more rigorous alignment (2, 4, or 8 bytes).

```
<byte_type> ::= #BYTE (<byte_length> [,<byte_alignment>] )
```

<byte_length> ::= <integer_expression>

<byte_alignment> ::= 1 | 2 | 4 | 8

Example:

```
TYPE
  one_word = #BYTE(8,8),        "8 bytes. Word aligned."
  decimal_string = #BYTE(256), "256 bytes. Byte aligned."
  half_word = #BYTE(4,4),       "4 bytes. Halfword aligned."
  character = #BYTE(1);         "1 byte. Byte aligned."
```

4.1.5.2 Bit Type

A bit type defines a fixed length bit string aligned on a bit boundary. The length of a bit string is a maximum of 64.

<bit_type> ::= #BIT (<bit_length>)

<bit_length> ::= <integer_constant>

4.1.5.3 Address Type

An address type is a full virtual address. It is defined as a record to allow the individual fields of a virtual address to be referenced.

<address_type> ::= #ADDRESS

#ADDRESS is equivalent to the following record:

```
TYPE
  #ADDRESS =
    RECORD
      ring_num  : #BIT(4),
      seg_num   : #BIT(12),
      byte_num  : #BYTE(4)
    RECEND;
```


4.1.5.4 Descriptor Type

Two descriptor types are provided to define the halfword and fullword bdp descriptors.

<descriptor_type> ::= #DESCRH | #DESCRF

The descriptor types are equivalent to the following record types.

```
TYPE
  #DESCRH =
    PACKED RECORD
      format : #BIT(1),
      dummy  : #BIT(2),
      length : #BIT(9),
      offset : #BYTE(2,2)
    RECEND

  #DESCRF =
    PACKED RECORD
      descr : #DESCRH,
      index : #BYTE(4,4)
    RECEND;
```

4.1.5.5 Machine Types In Expressions

The use of these types in SWL statements and expressions is limited to the assignment, equality, and inequality operators. Arithmetic expressions containing machine types are not allowed. Machine types can be addressed in the same way that normal SWL data types can be addressed (i.e. used as REF parameters, non-locally assigned to, etc.)

An elementary machine type (#BIT or #BYTE) is conformable to either integer values or the results of the memory built-in function.

An element of type #ADDRESS is conformable to a direct pointer value.

An element of type #ADDRESS, #DESCRH, or #DESCRF can be conformable with an aggregate (value constructor) containing

values to set the #BIT and #BYTE type fields.

4.1.6 MACHINE STORAGE CLASSES

<machine_storage_class> ::= #FAST | <reg_class> [(<reg_number>)]

<reg_class> ::= #XREG | #AREG

<reg_number> ::= <int_con> | <int_con>..<int_con>

<int_con> ::= <integer_constant>

4.1.6.1 Fast Storage Class

A variable declared with the #FAST storage class resides in a hardware register whenever possible. It is loaded into a register whenever the variable is referenced and will stay in the register file until the register is required for another purpose by the compiler.

Since a #FAST variable resides in the register file, the data type of the variable must be such that the variable will fit in a register.

The type of register to which a #FAST variable is assigned is a function of the variable's data type. A direct pointer type or a #ADDRESS type will be assigned to an A register, while all other #FAST variables will be assigned to X registers.

Since registers are not addressable as virtual memory, a #FAST variable can neither be made the object of a pointer nor can it be passed by reference.

The #FAST storage class is similar to the automatic storage class in that a #FAST variable is initialized, and is allocated upon elaboration of the declaration. A fast variable can only be locally referenced.

example:

```
VAR   induction  : [#FAST] 0..255 := lexical_level,
      link       : [#FAST] ^stack_frame := current;
FOR   induction := induction DOWNTO 0 DO
      link := link^.old_link
FOREND
```

4.1.6.2 Register Storage Classes

A variable declaration containing a register storage class specification does not, in fact, result in storage allocation. It merely associates a variable name and data type with a particular register or class of registers. The actual allocation of a hardware register to the register variable is accomplished with the LOCKREG and FREEREG machine statements.

If a register number is provided in the register storage class specification then that particular register will be assigned to the variable when it is locked. If a subrange definition is provided as the register number then a register whose number is an element of that subrange will be assigned. Otherwise any available register of the correct class (#X or #A) will be allocated.

Since a register variable never resides in virtual memory, it can not be initialized, can not be made the object of a pointer, can not be passed by reference, and can only be locally referenced. The data type of the variable must, of course, be such that the variable will, in fact, fit in a register.

4.1.7 MACHINE STATEMENTS

Machine statements do not follow normal SWL syntax. A machine statement is an unlabeled statement that starts with an exclamation mark ('!') and ends in a semi-colon (';').

There are two classes of machine statements: directives and instructions. Directives are defined to coordinate the allocation of resources and implicit code generation between the programmer and the compiler. Instructions result in the generation of specific machine instructions.

A machine statement has the following syntax.

<machine_statement> ::= ! <operator_field> [<operand_list>]

<operator_field> ::= <operator> {,<qualifier>}

<operand_list> ::= <operands> {,<operands>}

<operands> ::= <operand> | [<operand>{,<operand>}]

<operand> ::= <constant_expression> | <register_variable>

<operator> ::= <instruction_mnemonic> | <qualifier_mnemonic>

4.1.7.1 Machine Directives

4.1.7.2 LOCKREG and FREEREG

The LOCKREG and FREEREG directives provide a mechanism for reservation and allocation of physical registers. The statements are the only method available for coordinating register reservation between the programmer and the compiler.

The LOCKREG statement reserves registers and ensures that the compiler does not use the locked registers for the generation of SWL code. The registers remain locked until the registers are freed with the FREEREG directive or when the compiler encounters the end of the statement in which the LOCKREG directive occured, whichever comes first. If no register list is provided then all the registers are locked. If insufficient registers are available to the compiler to generate code in the accustomed manner, then the programmer must ensure that no SWL constructs (other than machine instructions) requiring registers are to be compiled.

The FREEREG statement releases the registers identified in the register list. If no register list is present then all the registers are released.

<lockreg_directive> ::= LOCKREG [<register_list>]

<register_list> ::= <reg_elem> {,<reg_elem>}

<reg_elem> ::= <register_variable>[(<reg_number>)]

<freereg_directive> ::= FREEREG [<reg_variable_list>]

<reg_variable_list> ::= <reg_variable> {,<reg_variable>}

Example:

```
VAR
  x0  : [#XREG(0)] integer,
  ddf1, ddf2 : [#AREG(5..6)] #ADDRESS,
  bdp1, bdp2 : [#AREG(9..15)] #ADDRESS,
  index : [#XREG(2..15)] 0..256;

! LOCKREG x0, ddf1(5), index, bdp2(11..14);
    . . . .
! FREEREG x0, index;
```

4.1.7.2.1 PROLOGUE AND EPILOGUE

The PROLOG and EPILOG directives cause the generation of the correct SWL prologue and epilogue. The PROLOG directive must appear as the first executable statement of a code procedure or code block if the procedure or block requires the existence of the standard SWL environment.

4.1.7.3 Machine Built-In Functions

Machine built-in functions are used in machine declarations or statements as long as the arguments can be evaluated at compile time. Machine built-in functions can also be used in normal SWL statements.

4.1.7.3.1 OFFSET FUNCTIONS

The functions described below can be used in machine statements including instructions. The functions return an offset from a specified base to a specified variable.

#OFP(<argument>[,<base>]).

This function returns a signed integer parcel (two byte) offset from the base (default is the current instruction location) to the argument. Both the argument and the base (if present) must be either procedure or label identifiers.

#OFB(<argument>[,<base>]).

This function returns a signed integer byte offset from the base (defualts to the start of the static area, constants area, and stack frame for static variables, constants, and automatic variables, respectively) to the argument. The argument must be either a variable reference or a constant. Both the base and the argument must be aligned to byte boundaries.

#OFBIT(<argument>[,<base>]).

This function returns a signed integer bit offset in a fashion identical with the #OFB function. Neither the base nor the argument, however, is required to be on a byte boundary.

4.1.7.3.2 LENGTH FUNCTIONS

The functions described below return the length of the variable.

#LENG(<variable>).

This function returns the byte length of the variable.

#LENGBIT(<variable>).

This function returns the bit length of the variable.

4.1.7.3.3 ADDRESS FUNCTION

The address function returns the virtual address of the argument.

#ADDR (<variable>)

4.1.7.3.4 MEMORY FUNCTION

The memory function allows an arbitrary hexidecimal pattern to be represented as a constant character string containing hex digits and blanks. The result is left-aligned when assigned. Blanks in the character string are ignored.

#M(<constant_string>).

Example: VAR half: #BYTE(4) := #M('00 FA BCDE');

4.1.7.4 Instructions

4.1.7.5 Qualifier Definitions

NAME	Description
#BOOL,	A single bit.
#COND	The condition register.
#COLL	Collated.
#DR	Double precision real.
#EXTD	Extended
exception	#OVRFLO, #UNDFLO, or #INDEF.
#F	Full word.
field	#FORMAT,#KIND,#LENGTH,#OFSET,#INDEX,#F,#H
#H	Half word.
immed	#ALPHA, #ALPHAFILL, #BIN, #DEC
#MAINT	Maintainance register.
#BLOCK	Register Block.
#NUM	Numeric.
#PAGE	Page table.
#R	Real.
relation	#EQ, #GE, #GT, #LE, #LT, #NE.
#SUBSCR	Subscripted.
#STATE	State register.
#UN	Unnormalized.
#X0	Register X0.
chars	Integer constant: 1,2,3,4,5,6,7,8.

4.1.7.5.1 INSTRUCTION OPERAND SYNTAX

Xi, Xj, Xk ::= <#XREG_variables>

Ai, Aj, Ak ::= <#AREG_variables>

J, Q ::= <integer_constants>

Uj, Mk ::= [<A_table> [,<A_variable>] , Q]
<A_table>, <A_variable> ::= <#A_variable>

4.1.7.5.2 INSTRUCTION DEFINITIONS BY NUMBER

Note that the following instruction syntax does not follow normal SHL conventions in that the square brackets appearing in this section are required even though they are not underlined in this section.

REF. NO.	NAME AND QUALIFIERS	OPERANDS
1	LOAD,CHARS	Xk,[A],Xl,d]
2	LOAD,CHARS	Xk,[A],Q]
3	STORE,CHARS	Xk,[A],Xl,d]
4	STORE,CHARS	Xk,[A],C]
5	LOAD	Xk,[A],Xl,d]
6	LOAD	Xk,[A],Q]
7	STORE	Xk,[A],Xl,d]
8	STORE	Xk,[A],C]
9	LOAD,X0	Xk,[A],Xl,d]
10	LOAD,X0	Xk,[A],Q]
11	STORE,X0	Xk,[A],Xl,d]
12	STORE,X0	XK,[A],Q]
13	LOAD,CHARS	Xk,[#P,C]
14	LOAD,BOOL	Xk,[A],C]
15	STORE,BOOL	Xk,[A],Q]
16	LOAD	Ak,[A],Xl,d]
17	LOAD	Ak,[A],Q]
18	STORE	Ak,[A],Xl,d]
19	STORE	Ak,[A],C]
20	LOAD,MULT	[A],K]
21	STORE,MULT	[A],K]
22	ADD	Xk,X]

4.0 APPENDIX B - IPL MACHINE BREAKOUT
4.1.7.5.2 INSTRUCTION DEFINITIONS BY NUMBER

REF. NO.	NAME AND QUALIFIERS	OPERANDS
23	SUB	Xk,XI
24	MULT	Xk,Xj
25	DIV	Xk,Xj
26	ABS	Xk,Xj
27	ADD,H	Xk,Xj
28	ADD,H	Xk,[Xj],Q]
29	ADD,H	Xk,j
30	SUB,H	Xk,Xj
31	SUB,H	Xk,j
32	MULT,H	Xk,Xj
33	MULT,H	Xk,[Xj],Q]
34	DIV,H	Xk,Xj
35	COMP	Xk,Xj
36	COMP,H	Xk,Xj
37	BREQ	Xj,Xk,Q
38	BRNE	Xj,Xk,Q
39	BRGT	Xj,Xk,Q
40	BRLE	Xj,Xk,Q
41	BREQ,H	Xj,Xk,Q
42	BRNE,H	Xj,Xk,Q
43	BRGT,H	Xj,Xk,Q

4.0 APPENDIX B - IPL MACHINE BREAKOUT
4.1.7.5.2 INSTRUCTION DEFINITIONS BY NUMBER

REF. NO.	NAME AND QUALIFIERS	OPERANDS
44	BRLE,H	Xj,Xk,Q
45	DOLOOP	Xj,Xk,Q
46	BR	Aj,Ak,Q
47	BR	[#P,Xk]
48	BR	Aj,Xk
49	COPY	Xk,Xj
50	COPY	Xk,Aj
51	COPY	Ak,Aj
52	COPY	Ak,Aj
53	COPY,H	Xk,Xj
54	ADD	Ak,[A],Q]
55	ADD,P	Ak,[Xj],Q]
56	ADD	Ak,Q
57	COPY,H	Xk,j
58	COPY,H	Xk,j
59	COPY,EXTD	Xk,Q
60	COPY,XO	JK
61	COPY,SIGN	Xk,j
62	ROTATE	Xk,Xj,[XI,d]
63	SHIFT	Xk,Xj,[XI,d]
64	SHIFT,H	Xk,Xj,[XI,d]
65	OR	Xk,Xj

REF. NO.	NAME AND QUALIFIERS	OPERANDS
66	XOR	Xk,Xj
67	AND	Xk,Xi
68	NOT	Xk,Xj
69	INHIB	Xk,Xj
70	MASK	Xk,[Xi,d]
71	ISOLATE	Xk,Xj,[Xi,d]
72	INSERT	Xk,Xj,[Xi,d]
73	MOVE	Ak,Aj
74	ADD	Dk,Dj
75	SUB	Dk,Dj
76	MULT	Dk,Dj
77	DIV	Dk,Dj
78	SCALE	Dk,Dj
79	SCALERCUND	Dk,Dj
80	N/A	
81	N/A	
82	N/A	
83	COMP,DEC	Dk,Dj
84	COMP	Dk,Dj
85	COMP,COLL	Dk,Dj
86	SCAN	Dk,Dj
87	N/A	

REF. NO.	NAME AND QUALIFIERS	OPERANDS
88	TRANSLATE	Dk,Dj
89	MOVE	Dk,Dj
90	N/A	
91	EDIT	Dk,Dj
92	MOVE,NUM	Dk,Dj
93	N/A	
94	MOVE,field,field	Dk,Dj
95	ADD,field,field	Dk,Dj
96	CALCSUB	Dk,Dj
97	CONV,R	Xk,Xj
98	CONV	Xk,Xj
99	ADD,R	Xk,Xj
100	SUB,R	Xk,Xj
101	ADD,R,UN	Xk,Xj
102	SUB,R,UN	Xk,Xj
103	MULT,R	Xk,Xj
104	DIV,R	Xk,Xj
105	ADD,DR	Xk,Xj
106	SUB,DR	Xk,Xj
107	MULT,DR	Xk,Xj
108	DIV,DR	Xk,Xj
109	BREQ,R	Xj,Xk,Q

4.0 APPENDIX B - IPL MACHINE BREAKOUT
4.1.7.5.2 INSTRUCTION DEFINITIONS BY NUMBER

REF. NO.	NAME AND QUALIFIERS	OPERANDS
110	BRNE,R	Xj,Xk,Q
111	BRGT,R	Xj,Xk,Q
112	BRLE,R	Xj,Xk,Q
113	BR,exception	Xk,Q
114	COMP,R	Xj,Xk
115	CALL	[Ai,d],Aj,Ak
116	CALL	[#P,Q],Aj,Ak
117	RETURN	
118	POP	
119	N/A	
120	EXCHANGE	
121	ERROR	
122	INTRPT	j,Xk
123	INTRPTPROD	Xk
124	LOAD,CLEAR	?
125	COMP	Xk,Aj
126	COMP,PAGE	
127	LOAD,PAGE	Xj,Xk
128	N/A	
129	N/A	
130	LOAD,STATE	Xk,Xj
131	STORE,STATE	Xk,Xj

4.0 APPENDIX B - IPL MACHINE BREAKOUT
4.1.7.5.2 INSTRUCTION DEFINITIONS BY NUMBER

REF. NO.	NAME AND QUALIFIERS	OPERANDS
132	LOAD,MAINT	Xk,Xj
133	STORE,MAINT	Xk,Xj
134	BR,COND	jK,Q
135	N/A	
136	KEYPOINT	
137	N/A	
138	PURGE	
139	ALGORITHM	
140	N/A	
141	N/A	
142	N/A	
143	ADD,IMM	Xk,[Xj,Q]
144	COMP	Aj,Ak
145	CONV,relation	Xk
146	STORE,field	Dk,Xj
147	LOAD,field	Xk,Dj
148	ADD,field	Dk,Xj
149	SUB,field	Dk,Xj
150	AND	Dk,Dj
151	OR	Dk,Dj

REF. NO.	NAME AND QUALIFIERS	OPERANDS
152	XOR	Dk,DJ
153	NOT	Dk,DJ
154	COPY,IMMED	Dk,Q
155	COMP,IMMED	Dk,Q
156	ADD,IMMED	Dk,Q
157	LOAD	Ak,DJ
158	LOAD	Xk,DJ
159	STORE	Xk,D

4.2 IPL MACHINE BREAKOUT CORRECTIONS

This section describes corrections to the IPL Machine Breakout memo (see section entitled, "IPL MACHINE BREAKOUT" -- (REVISION B)).

4.2.1 MACHINE BREAKOUT CORRECTIONS

- <byte_length> ::= <positive_constant_integer_expression>

- <byte_alignment> ::= 2**n (where 0 ≤ n ≤ 8)
  (The ** operator is the exponentiation operator)

- <bit_type> ::= #BIT (<bit_length>[,bit_offset>])
  <bit_length> ::= <positive_integer_constant>
  <bit_offset> ::= <positive_integer_constant>
  (from byte boundary)
  (<bit_length> + <bit_offset> ≤ 64)

- TYPE
  #DESCR =
  PACKED RECORD
  format : #BIT(1),


```
    dummy : #BIT(2),
    tag : #BIT(4),
    length : #BIT(9),
    offset : #byte(2,2)
  RECEND
```

- Two functions have been added, #TOFB and #TOFBIT. These functions are analogous to the functions #OFB and #OFBIT, with the exception that the first argument is a type identifier of a record definition and the second argument is a field identifier of a field within the record.

```
#TOFB (<type_id>,<field_identifier>)
#TOFBIT (<type_id>,<field_identifier>)
<field_identifier> ::= <identifier>
<type_id> ::= <type identifier>
<type_identifier> ::= <identifier>
```

- The following corrections pertain to the section entitled "Instructions", within the "IPL Machine Breakout" appendix.

- The following qualifiers should be added:
  #MULT
  #IMM
  #P
  #SIGN
  #DEC
  #SET

- J,K,o,Q ::= <integer constant>

- All upper case qualifiers should be preceded by the character, number sign.

- All occurrences of the upper case string CHARS should be replaced by the lower case string chars.

- All occurrences of the string BRLE should be replaced by BRGE.

- For reference numbers 54 through 56, the name should be changed to ADDA.

- For reference number 56, the operands should be Ak,XJ.

- For reference numbers 57 through 61, the name should be

ENTER.

- For reference number 57, the qualifier should be POS.
- For reference number 58, the qualifier should be NEG.
- For reference number 52, the operands should be Ak,Xj.
- For reference number 67, the operands should be Xk,Xj.
- For reference number 95, the name and qualifiers should be ADD,field,field.
- For reference number 112, the name should be BRGE.
- For reference number 113, the qualifier should be exception.
- For reference number 124, the qualifier should be SET and the operands should be Xk,Aj.
- For reference number 122, the operands should be Xk.
- For reference number 136, the operands should be Xk,j,Q.
- For reference number 138, the operands should be Xk,Q.
- For reference number 139, the name and qualifiers should be ALGORITHM,chars, and the operands should be Aj,Ak,Q.
- For reference number 145, the qualifier does not allow all possible j specifications.
- For reference numbers 150 through 153, the operand field should be Aj,Ak.
- For reference numbers 154 through 156, the qualifier should be lower case immed.
- For reference number 159, the operands should be Xj,Dk.
- For reference number 153, the name should be MCC.

5.0 APPENDIX C - COMPILATION ERRORS

This section contains the definition of the compilation error codes output by the passes of the ISHL/CI V2.0 compiler.

5.1 PASS I ERROR CODE DEFINITIONS

The error messages generated by PASS I immediately follow the source line in which the error occurred on the output listing (see section entitled, "COMPILER OUTPUT") and are of the following form:

********** error code

The table below associates error codes with their definition. The total number of PASS I errors which is detected is written onto the listing file in the following form:

x ERROR(S) WERE DETECTED

1 : Scalar Type or Type Identifier expected
2 : Identifier expected
3 : Error in value punctuation
4 : ')' expected
5 : ':' expected
6 : Illegal/unexpected symbol
7 : Error in parameter list
8 : 'OF' expected
9 : '(' expected
10 : Illegal type
11 : '[' expected
12 : ']' expected
13 : 'END' expected
14 : ';' expected
15 : '=' expected
16 : 'BECOMES' expected
17 : Error in field list, Identifier or 'CASE' expected
20 : ',' expected
21 : 'MODULE' expected

22 : Illegal Symbol or Punctuation
23 : Semicolon may not appear in comment
24 : 'IFEND' expected
25 : 'WHILEND' expected
26 : "LOOPEND' expected
27 : 'FOREND' expected
28 : 'CASEND' expected
29 : '$' or '#' expected
30 : 'PROC(END)' expected
31 : CYCLE target not repetitive
32 : Invalid attribute
33 : EXIT not properly contained
34 : 'CRAMMED' expected
35 : Crammed structured must be Array or Record
36 : '..' expected
37 : '^' expected
38 : Type Identifier expected
39 : Conflicting Attributes
40 : Variable bounds in static domain illegal
41 : Adaptable or Bound Variant type in parameter list or adaptable pointer object only
42 : Feature illegal outside REPDEP procedures
43 : Illegal PRONG class
44 : Machine types illegal without CODE attribute
50 : Error in constant
51 : ':=' expected
52 : 'THEN' expected
53 : 'UNTIL' expected
54 : 'DO' expected
55 : 'TO/DOWNTO' expected
58 : Error in factor
59 : Dynamic variable reference illegal in this context
60 : Inconsistent module identification or nesting
61 : PRONG not declared
62 : Illegal variable reference
63 : Illegal label punctuations
99 : Value exceeds range
100 : Assignment/Index range error
101 : Multiple declaration of identifier
102 : Lower Bound must be less than Upper Bound
103 : Identifier not of appropriate class
104 : Undeclared identifier
105 : Sign not permitted
106 : Number expected
107 : Incompatible subrange types
108 : Invalid type for this structure
110 : Tagfield type must be Scalar or Subrange
111 : Selector incompatible with Tagfield type

113 : Index type must be Non-Real Scalar or Subrange
114 : Scalar type must not be Real
116 : Error in type of standard procedure parameter
117 : Unsatisfied forward reference
118 : Pointer with forward REF type must be fixed
120 : Function result must be Scalar, Subrange or Pointer
123 : Equivalent types in UNION declaration
125 : Error in type of standard function parameter
126 : Number of values does not match declaration
129 : Type conflict of operands
130 : Expression is not of set type
131 : Only tests on equality allowed
132 : Strict inclusion not allowed
134 : Illegal type of operand(s)
135 : Type of operand must be Boolean or Set
136 : Set element must be Non-Real Scalar or Subrange
137 : Value incompatible with set element
138 : Variable must be of array type
139 : Index incompatible with declarations
140 : Variable must be of record type
141 : Type of variable must be Pointer
142 : Illegal parameter substitution
143 : Illegal type of loop control variable
144 : Expression type illegal
145 : Type conflict
147 : Type incompatible with selection expression
148 : Subrange bounds must be Scalar
150 : Assignment to standard function illegal
151 : Assignment to formal function illegal
152 : No such field in this record
156 : Multidefined case selector
160 : Previous declarations was not forward
161 : Procedure previously forward declared
165 : Multidefined label
166 : Undeclared exit label
170 : Variant tag must be initialized
171 : Pack/Cram attribute illegal on this type
172 : Initialization illegal in this context/on this type
173 : Value constructor may contain '*' only in initialization
174 : Value type incompatible with declaration
175 : Base specification required
176 : Crammed record must be fixed bound
177 : Width specification required
178 : Object size exceeds width specifications
179 : Element must be fixed size
180 : Invalid adaptable subrange
181 : Subrange must be fixed bound for crammed
182 : Multiple/Illegal variant, variable bound, adaptable fields

5.0 APPENDIX C - COMPILATION ERRORS
5.1 PASS I ERROR CODE DEFINITIONS

183 : Selector must be constant
184 : Invalid parental type for relative pointer
185 : Invalid pointer object type
186 : Integer expression expected
187 : Invalid string length
188 : String object type must be CHAR
189 : Object must be data type
190 : Value list does not match declaration
191 : Tag value must be constant
192 : Tag value not in selector range
193 : Static initialization by constants only
194 : Value out of set range (0..255)
195 : Assignment to external function illegal
196 : Type of variable must be string
197 : Substring length exceeds source length
198 : Assignment expression not UNION member
199 : Invalid UNION member type
201 : Error in constant - digit expected
202 : String in constant contains invalid delimeter (EOL)
203 : Integer constant too large
205 : Constant expected
206 : Exponent too large/out of range
207 : Symbol too long
208 : Radix not numeric
209 : Radix out of range
210 : Invalid radix
211 : Invalid digit in radix
213 : Positive integer expected
219 : Unknown machine type
220 : Machine dependent construct illegal
221 : Machine type cannot be object of pointer
222 : Value exceeds register range for IPL
223 : Machine class variables cannot be PRONG or STATIC
224 : Type incompatible with register class
225 : Register class may not be initialized
226 : Field reference to AREG variable illegal
227 : Invalid declaration/statement without prolog
228 : Type must be UNION
229 : Conformity operator expected
230 : Illegal qualifier
231 : Illegal operation code
232 : Illegal instruction format
233 : Illegal use of two qualifiers
234 : Variable unknown locally
235 : Variable must be register class
236 : Register descriptor on lock only
237 : Register out of range
238 : All registers locked (overflow)

5.0 APPENDIX C - COMPILATION ERRORS
5.1 PASS I ERROR CODE DEFINITIONS

239 : Register not locked
240 : Illegal Operand field in this position
241 : Unrecognizable operand field
242 : Global reference to register variable illegal
243 : Machine type constructors may only contain constants
244 : Only variant records may be 'bound'
250 : Too many nested SCOPE environments
251 : Compile time display vector overflow
298 : Relevent symbol found
399 : Feature not supported
400*: Compiler error in LOADVALU/LOADATTRID/SELECTOR
401*: Compiler error in LOAD
402*: Compiler error in LOADADDRESS
403*: Compiler error in STORE
404*: Compiler error in BOATALOC/LOADADDRESS/SELECTOR
regarding machine classes
405 : Only machine types here
407*: Compiler error

NOTE: The error codes marked by an asterisk indicate either a programming error or a compiler failure. If you obtain one of these error codes, first peruse your program for programming errors; and if no errors are discovered, you should use the PSR Bug Reporting mechanism described in the SES User's Handbook to report the problem.

5.2 PASS II ERROR CODE DEFINITION

The error messages generated by PASS II immediately follow either the "source form" on the compiler listing file or follow the message: "x ERROR(S) WERE DETECTED", which also is on the listing file. The messages have the following generic form:

```
"message text"
PCODE = PPP      COUNT  yyy  zzz
```

"message text" is the error message and zzz is the "LINE" field on the compiler listing file. (See section entitled, "Compiler Output".) The other information should be ignored.

The various forms of "message text" are listed below. The colons are not included in "message text" and serve the function of "message text" separators.

**: OUT OF AREGS
: A REGISTER LOCKED TABLE OVERFLOW

```
    : A REGISTER LOCKED TABLE OVERFLOW
   *: RE-ASSIGN XREG FAILED, CLEARXREG CALLED
   *: PUSHSTACK FOUND NO REGS
   *: RTS IS UNAVAILABLE BECAUSE PROLOG INHIBITED
   *: NO REG FOR POPSTACK
   *: POPSTACK FAILED IRRECOVERABLY
   *: NO REG FOR PUSH
   *: PUSHREG - REG MODEL FAILURE
   *: OPERAND STACK OVERFLOW
   *: OPERAND STACK UNDERFLOW
   *: POPPING A MARK
  **: LOCKXREG - NO HOLD OR PREASSIGNED REG
  **: NO REGISTERS FOR LOCKAREG
    : SYMBOL FOR X REGISTER ALREADY LOCKED
    : FREE FOR UNLOCKED X REGISTER
    : FREE FOR UNKNOWN X REGISTER SYMBOL
    : SYMBOL FOR A REGISTER ALREADY LOCKED
    : FREE OF UNLOCKED A REGISTER
    : FREE OF UNKNOWN A REGISTER SYMBOL
  **: ASSIGNMENT OF X2 FAILED
  **: GENCUP - FUNC REG UNAVAIL
    : REPLACED BY -
  **: UNKNOWN X REG ID
  **: UNKNOWN A REG ID
   *: MORE THAN ONE OPERAND FOR FUNCTION
    : DESC. VAR REG NOT 8-F
    : DESC. TABLE REG NOT 4-5
    : UNLOCKED REFERENCE TO REG ID
  **: SOME FORWARD LABEL REFERENCES UNSATISFIED
   *: BAD MODIFIER
    : ADDRESS IN RTS
  **: GENCXP - FUNC REG UNAVAIL
    : GENCXP - FUNC REG LOCKED
   *: UNKNOWN XREF NAME
    : GENCUP - FUNC REG LOCKED
   *: QUALIFIER INCORRECT
   *: MAKEXAVAIL STACK FAILED
   *: MAKEXAVAIL FAILED
   *: UNDEFINED CONSTRUCT
  **: OPERAND OVERFLOW
  **: NEG. VALUE FOR UNSIGNED FIELD
   *: DUPLICATE LABELS
    : UNIMPLEMENTED
```

NOTE: The forms of "message text" which are preceded by an asterisk indicate compiler failure. If you obtain one of these error messages, you should use the PSR Bug Reporting mechanism described in the SES User's Handbook. The forms

of "message text" which are preceded by two asterisks indicate programmer error or compiler. If you obtain one of these error messages, you should peruse your program for programming errors on line ,ZZZ. If no programming error is detected, you should use the PSR Bug Reporting mechanism.

6.0 APPENDIX D - IPL ISWL RESERVED WORDS

A list of reserved words is given below. Words preceded by a double asterisk, **, are reserved words in the ISWL/CC and are not reserved words in ISWL/CI. Words preceded by an asterisk, *, are reserved words in ISWL/CI and are not reserved words in ISWL/CC. Words which are not preceded by one or two asterisks are reserved words in both ISWL/CC and ISWL/CI.

*ALIAS
ALIGNED
ALLOCATE
AND
ARRAY
BEGIN
*BOUND
BY
*CAND
CASE
CASEND
CAT
**CLOSE
CODE
CONST
COPROC
CRAMMED
CREATE
CYCLE
**DEFINE
**DEQUEUE
DESTROY
DO
DOWNTO
ELSE
END
**ENQUEUE
*EPILOG
**EXECUTE
EXIT
EXTERNAL
**FILE
FOR
**NIL
*NIN
NOT
OF
*ON
**OPEN
OR
ORIF
PACKED
POP
PROC
PROCEND
*PROLOG
PUSH
**QUEUE
RECEND
RECORD
REF
REL
REP
REPDEP
REPEAT
RESET
RESUME
RETURN
**REWIND
SEGMENT
SEQ
SET
STACK
STATIC
STRING
**TAG

FOREND
*FORWARD
FREE
*FRELREG
*GET
GOTO
HEAP
IF
IFEND
IN
LABEL
*LOCKREG
LOOP
LOOPEND
MACRO
MACROEND
*MAIN
*MALIGNED
MOD
**MODE
MODEND
MODULE
NEXT
THEN
TO
TYPE
UNION
UNPACKED
UNTIL
*UOR
VAL
VAR
*VSTRING
**WEOF
WHEN
WHILE
WHILEND
XDCL
XOR
XREF

7.0 APPENDIX E - ASSEMBLY LANGUAGE DESCRIPTION

The purpose of this section is to describe the IPL assembler mneumonics output by the compiler when the "C" comment toggle is selected. (See section entitled, COMPILER OUTPUT.)

The compiler identifies each symbolic instruction mneumonic according to its syntax and generates a 16- or 32-bit instruction. The table below indicates the translation of assembler mneumonics and operand fields to the IPL machine instructions (see IPL Processor-Memory Model Independent GDS). The first and second columns of the table indicate, by reference number and opcode respectively, the instruction generated for the assembler menumonic in the third column. The forth column is the operand field of the instruction and specifies the order from left to right of the operands.

GDS REF NO.	OP CODE	IPL ASSEMBLER MNEUMONIC	OPERAND FIELD
001	D0	LBYXIS	S,J,K,I,D
002	C0	LBYXS	S,J,K,Q
003	D8	SBYXIS	S,J,K,I,D
004	C8	SBYXS	S,J,K,Q
005	A2	LWXI	J,K,I,D
006	82	LWX	J,K,Q
007	A3	SWXI	J,K,I,D
008	83	SWX	J,K,Q
009	A4	LBYXIX	J,K,I,D
010	84	LBYXX	J,K,Q
011	A5	SBYXIX	J,K,I,D
012	85	SBYXX	J,K,Q

7.0 APPENDIX E - ASSEMBLY LANGUAGE DESCRIPTION

GDS REF NO.	OP CODE	IPL ASSEMBLER MNEUMONIC	OPERAND FIELD
053	0C	CPHXX	J,K
054	8E	CPADD	J,K,Q
055	8F	CP	J,K,Q
056	2A	AIND	J,K
057	3D	EXPJ	J,K
058	3E	EXMJ	J,K
059	8D	EXQ	K,Q
060	3F	EHLG	J,K
061	1F	EXL	J,K
062	A8	SHXC	J,K,I,D
063	A9	SHX	J,K,I,D
064	AA	SHHX	J,K,I,D
065	18	LS	J,K
066	19	LD	J,K
067	1A	LP	J,K
068	1B	LC	J,K
069	1C	LINH	J,K
070	AC	ISOM	K,I,D
071	AD	ISOX	J,K,I,D
072	AE	INSX	J,K,I,D

7.0 APPENDIX L - ASSEMBLY LANGUAGE DESCRIPTION

GDS REF NO.	OP CODE	IPL ASSEMBLER MNEUMONIC	OPERAND FIELD
073	12	MC	J,K
074	E0	DS	J0,J1,K0 K1,Q0,Q1
075	E1	DD	J0,J1,K0 K1,Q0,Q1
076	E2	DP	J0,J1,K0 K1,Q0,Q1
077	E3	DQ	J0,J1,K0 K1,Q0,Q1
078	E4	DSCAL	J0,J1,K0 K1,Q0,Q1
079	E5	DSCALR	J0,J1,K0 K1,Q0,Q1
083	E6	DFCOM	J0,J1,K0 K1,Q0,Q1
084	E8	DBCOM	J0,J1,K0 K1,Q0,Q1
085	E9	DBCOL	J0,J1,K0 K1,Q0,Q1
086	EA	DBSCAN	J0,J1,K0 K1,Q0,Q1
088	EB	DTRAN	J0,J1,K0 K1,Q0,Q1
089	EC	DBMV	J0,J1,K0 K1,Q0,Q1

GOS REF NO.	OP CODE	IPL ASSEMBLER MNEUMONIC	OPERAND FIELD
091	ED	DEDIT	J0,J1,K0 K1,Q0,Q1
092	E7	DNMV	J0,J1,K0 K1,Q0,Q1
094	EE	DTABMV	J0,J1,K0 K1,Q0,Q1
095	EF	DOFI	J0,J1,K0 K1,Q0,Q1
096	F4	DCALSB	J0,J1,K0 K1,Q0,Q1
097	3A	FCFP	J,K
098	38	FCI	J,K
099	30	FS	J,K
100	31	FD	J,K
101	38	FSU	J,K
102	39	FDU	J,K
103	32	FP	J,K
104	33	FQ	J,K
105	34	FSDP	J,K
106	35	FDDP	J,K
107	36	FPDP	J,K
108	37	FQDP	J,K
109	98	FDEQ	J,K,Q

GOS REF NO.	OP CODE	IPL ASSEMBLER MNEUMONIC	OPERAND FIELD
110	99	FBNE	J,K,Q
111	9A	FBGT	J,K,Q
112	9B	FBNLN	J,K,Q
113	9E	FBEX	J,K,Q
114	3C	FCOM	J,K
115	A6	CALLJ	J,K,I,D
116	B0	CALLP	J,K,Q
117	04	RET	
118	06	POP	
120	02	EXCH	
121	00	PE	
122	03	INPC	J,K
123	10	INPD	K
124	14	LSBT	J,K
125	15	COMX	J,K
126	16	TPG	J,K
127	17	LPGT	J,K
130	0E	CSRX	J,K
131	0F	CXSR	J,K
132	08	CCMRX	J,K

7.0 APPENDIX E - ASSEMBLY LANGUAGE DESCRIPTION

GDS REF NO.	OP CODE	IPL ASSEMBLER MNEUMONIC	OPERAND FIELD
133	07	CXCMR	J,K
134	9F	BCR	J,K,Q
136	B1	KEY	J,K,Q
138	05	PB	J,K
139	B8	ALG	S,J,K,Q
143	8B	ISQ	J,K,Q
144	13	COMA	J,K
145	1E	MRK	J,K
146	F0	DDFINS	J,K0,K1 Q1
147	F1	DDFEX	J0,J1 K,Q0
148	F2	DDFIX	J,K0 K1,Q1
149	F3	DDFDX	J,K0 K1,Q1
150	F5	LPMEM	J,K
151	F6	LSMEM	J,K
152	F7	LDMEM	J,K
153	F8	MCC	J,K

7.0 APPENDIX E - ASSEMBLY LANGUAGE DESCRIPTION

GDS REF NO.	OP CODE	IPL ASSEMBLER MNEUMONIC	OPERAND FIELD
154	F9	DMI	J,K0,K1 Q0,Q1
155	FA	DCI	J,K0,K1 Q0,Q1
156	FB	DAI	J,K0,K1 Q0,Q1
157	FC	DLA	J0,J1 K,Q0
158	FD	DLX	J0,J1 K,Q0
159	FE	DSX	J,K0 K1,Q1
160	FF	IO	J,K,Q

8.0 APPENDIX F - COMPILER OUTPUT

If the compilation does not abort, the compiler outputs data onto two files, a listing file and optionally, an object file (see section entitled, "COMPILING SOURCE MODULES").

8.1 LISTING FILE

The listing file contains two text forms of the source input file. The first form, the "source form", is an exact copy of the source input file concatenated with information provided by the compiler. The second form, the "assembly form", is an IPL assembly language listing of the generated object code for the source input file and is produced when the C comment toggle is selected.

Example of Listing File:

Given the following source lines to compile.

```
MODULE ERS#EXAMPLE;
VAR
  V1 : INTEGER,
  V2 : [STATIC] INTEGER := 2;
PROC [MAIN] ERS#P;
VAR
  V3 : [STATIC] INTEGER := 4,
  V4 : INTEGER,
  V5,V6 : INTEGER;
IF V2 = 3 THEN
  V4 := 6 ELSE
  V5 := 7
IFEND;
  V3 := V2;
PROCEND ERS#P;
MODEND ERS#EXAMPLE
```


The "source form" is

```
PG 1  STAT  AUTO  PR  NS  1  ISWL/IPL  VERSION 2.0, LEVEL 0

   1     0     0   0   0  MODULE ERS#EXAMPLE;
   2     0     0   0   0  VAR
   3     0     0   0   0    V1 : INTEGER,
   4     4     0   0   0    V2 : [STATIC] INTEGER := 2;
   5     8     0   0   0  PROC [MAIN] ERS#P;
   6     8     6   1   0  VAR
   7     8     6   1   0    V3 : [STATIC] INTEGER := 4,
   8    12     6   1   0    V4 : INTEGER,
   9    12    10   1   0    V5,V6 : INTEGER;
  10    12    18   1   0  IF V2 = V3 THEN
  11    12    18   1   1    V4 := 6 ELSE
  12    12    18   1   1    V5 := 7
  13    12    18   1   1  IFEND;
  14    12    18   1   0    V3 := V2;
  15    12    18   1   0  PROCEND ERS#P;
  16    12     0   0   0  MODEND ERS#EXAMPLE
YY/MM/DD   HH.MM.SS
```

and the "assembly form" is

```
S.LN OPC  SECT/OFFSET --TEXT-- IPL OP REF

  10 LOD      CS 000000 C3720004 LBYXS    2
     LOC      CS 000004 80030003 EXQ     59
     EQU      CS 000008 2C23     ICH     36
              CS 00000A 1E82     MRK    145
     FJP      CS 00000C*90020000 BXHEQ   41
  11 LOC      CS 000010 80020006 EXQ     59
     STR      CS 000014 CB120006 SBYXS    4
     UJP      CS 000018*94000000 BXEQ    37
              CS 00001E 0008
  13 LOC      CS 00001C 80020007 EXQ     59
     STR      CS 000020 CB12000E SBYXS    4
              CS 00001A 0006
  14 LOD      CS 000024 C3720004 LBYXS    2
     STR      CS 000028 CB720008 SBYXS    4
  15 RET      CS 00002C 0400     RET    117   EPT CS 00002E R ERS#P
     DEF      CS 00002E 8F0F0004 CP      55
              CS 000032*90000000 BXHEQ   41
     ENT      CS 000036 81010000 SA      19
              CS 00003A 3E100018 CPADD   54
     UJP      CS 00003E 9400FFE1 BXEQ    37 CS+000000
  16 DAT      ST 000004 00000002
     DAT      ST 000008 00000004
     FIN      CS 000034 0008
              CS 000042 80370000 LA      17   RIF CS 000044 Q BS C
                                              AIN BS 000000 P A(ST)
              CS 000046 2FF0     JPR     48

              SECTION MAP OF MODULE: ERS#EXAMPLE

             0 CS 000048 CODE SECTION        READ     EXECUTE
             2 BS 000008 BINDING SECTION     READ     BINDING
             1 ST 00000C WORKING SECTION     READ     WRITE
          TRANSFER NAME: ERS#P
```


With regard to the "source form", the fields concatenated to the source input are described below. The compiler fields concatenated with the source input lines are called output lines.

FIELD	DEFINITION
PG	Decimal number indicating the page number of the "source form".
LINE	Decimal number assigned to each source line in the input file. This field appears immediately below the PG field in the "source form".
STAT	Decimal number which signifies a relative byte offset from the beginning of the STATIC area. The field only has meaning if the associated source input line on the source ouput line is either a variable declaration which has a "static" or "xdcl" attribute or is a variable declaration which is immediately contained within a module declaration. In this case, the byte offset is the location allocated to the last variable (as scanned from left to right) appearing in the associated source line.
AUTO	Decimal number which signifies a relative byte offset from the beginning of a STACK frame. The field has meaning only if the associated source input line on the same output line is a variable declaration which is immediately contained in a procedure. In this case, the byte offset is the location allocated to the variable appearing in the associated source line.
PR	Decimal number which indicates the procedure nesting level.
NS	Decimal number which indicates the structured statement nesting level.
DATE/TIME	The date/time stamp immediately follows the

last output line of the compilation unit.

The "assembly form" contains a description of the SES/IPL LINKER records which are output by the compiler for the compilation unit.

For each level zero module within a compilation unit, the "assembly form" description includes a "section definition" description, a "section summary" description, and "other SES/IPL LINKER record" descriptions.

The "section definition" description is preceeded by the header line

S.LN OPC SECT/OFFSET --TEXT-- IPL OP REF

Each line appearing under this header line in the section description is referred to as an entry in the "section definition" description. The meaning of each of the fields in the header line is:

FIELD	DEFINITION
S.LN	Same definition as the LINE field in the "source form" of the listing file.
OPC	Used for compiler debugging and is to be ignored.
SECT/OFFSET	Defines the SES/IPL LINKER section and offset (in hexadecimal bytes from the beginning of the section) into which the --TEXT-- field is placed. The compiler uses five mneumonic names to classify the different sections it generates: CS - code section BS - binding section SI - static (working storage) section LT - literal section HP - heap section
--TEXT--	Hexadecimal data which is inserted at the location specified by the SECT/OFFSET field.
IPL OP	IPL assembler mneumonic for the instruction in the --TEXT-- field. This field is blank if the --TEXT-- field is not an IPL instruction.
REF	IPL reference number for the instruction in the --TEXT-- field. This field is blank if

the --TEXT-- field is not an IPL instruction.

Other items to note within the "section definition" description are the following:

- If an asterisk appears between the SECT/OFFSET and --TEXT-- field, the instruction in the --TEXT-- field contains a forward reference to a "target location" in the code section. Another entry, ENTRY N, in the section definition description is generated by the compiler to resolve the forward reference. ENTRY N's SECT/OFFSET field will be:

  CS XXXXXX where

  XXXXXX = YYYYYY + 2
  and
  YYYYYY = hexadecimal offset of the "section definition" entry which contained the forward reference.

  ENTRY N immediately precedes the entry in "section definition" description whose --TEXT-- field contains the IPL instruction which is the object of the forward reference.

- If an entry in the section definition description is for an instruction and the text string, 'CS + ZZZZZZ' (where ZZZZZZ is a hexadecimal number) appears after the REF field, then this instruction contains a backward reference to the code section and ZZZZZZ is the offset in the code section to which the reference is made.

- One can determine the IPL instructions which are generated for a source line in the following manner. The entry with a non-blank S.LN field associates a group of "section definition" entries which are generated for source line, S.LN. The group of entries include the entry with the non-blank S.LN field and all entries with blank S.LN fields which follow, up to the next entry with a non-blank S.LN field. All entries in this group which are instruction entries (i.e., ones with a non-blank REF field) are the instructions generated for the source line.

The "section summary" description begins with a header line of

SECTION MAP OF MODULE : module identifier

where module identifier is the identifier that was on the module declaration statement.

The entries which follow the header line are of the following generic form:

ORD1	SECTION1	LENGTH1	SEC DESC1	ATTRIBUTES1
ORD2	SECTION2	LENGTH2	SEC DESC2	ATTRIBUTES2
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
ORDN	SECTIONN	LENGTHN	SEC DESCN	ATTRIBUTESN

TRANSFER NAME : XXX

FIELD	DEFINITION
ORD	SES/IPL LINKER section ordinal number for section.
SECTION	Either CS, BS, ST, LT, HP (see SECT/OFFSET field definition).
LENGTH	Hexadecimal byte length of section.
SEC_DESC	Either CODE SECTION BINDING SECTION WORKING SECTION EXTENSIBLE COMMON SECTION WORKING SECTION
ATTRIBUTES	SES/IPL LINKER section attributes. Either READ, WRITE, EXECUTE, BINDING, or a combination of these.
TRANSFER NAME	XXX is either blank or is the procedure identifier which had the main attribute.

The "other SES/IPL LINKER record" descriptions are intermixed with the "section definition" descriptions, appear to the right of the REF field on the output listing, and are used for debugging the compiler.

In general, these descriptions describe the following type of SES/IPL LINKER records output by the compiler:

entry point definition (EPT)
relocation information (RIF)
address insertion (AIN)
external reference linkage (XRL)
bit string insertion (BIT)

8.0 APPENDIX F - COMPILER OUTPUT
8.1 LISTING FILE

address offset insertion (AOI)

8.2 OBJECT FILE

The object file is in a format which can be linked by the SES/IPL LINKER V1.0 (see SES User's Handbook section entitled, SES/IPL LINKER).

9.0 APPENDIX H - COMPILING SOURCE MODULES

9.0 APPENDIX H - COMPILING SOURCE MODULES

The ISWL/CI V2.0 compiler is executed as a procedure and the following command readies the compilation procedure:

```
GET,IPLISWL/UN=ALL
```

After the IPLISWL compilation procedure is obtained, it may be executed with the following control statement:

```
-IPLISWL[,I=<source file>][,O=<listing file>]
        [,B=<object file>][,PASS=<1|2>][,FL=<fl>]
```

The default parameter values are:

```
I = I
O = O
B = B
PASS = 2 (complete compilation)
FL = 140000
```

The user can increase the efficiency of his IPLISWL compilations by using the PASS option on the procedure call. The IPLISWL compiler has two distinct passes and they may be optionally executed. If PASS=1, the syntax errors are checked. If PASS=2 or the parameter is omitted, the syntax errors are checked and the code generation is done and an object deck is generated.

If the compiler aborts during a compilation, the message, "COMPILATION ABORTED", is output to the KRONOS standard output file. If you obtain this message while compiling, you should use the PSR Bug Reporting mechanism described in the SES User Handbook. It is recommended that you attach the dayfile of the aborted compilation with the PSR.

10.0 APPENDIX G - OBJECT PROGRAM EXECUTION

This section describes ISWL/CI V2.0 runtime support procedures, and initiation and termination of an IPL ISWL object program.

10.1 IPL ISWL RUNTIME SUPPORT PROCEDURES

There exists ISWL/CI V2.0 runtime support procedures for processing NEXT and RESET statements, for processing the standard procedure, #STRINGREP, for processing the standard function, $STRING, and for processing execution time diagnostics (see section entitled "ABNORMAL TERMINATION"). If these runtime support procedures are referenced, they must be linked with the binary file produced by the compiler via the SES/IPL LINKER. The table below indicates the indirect files in the ALL catalog which contain the runtime procedures:

File Name	File Contents
IPLRT1	runtime error support
IPLRT2	#STRINGREP and $STRING support
IPLRT3	NEXT,RESET support

10.2 INITIATION

This section contains an example of KRONOS and SES Subsystem commands necessary to compile an ISWL source program, link and load the object file produced by the compiler, and execute the program on the simulator (see the SES User's Handbook for a description of the IPL SIMULATOR, SES/IPL LINKER, and the SES/IPL REAL MEMORY LOADER).

ISWL/CI Version 2.0 compilation and execution

The user should be familiar with the SES Subsystem, especially the LINK and RML commands, and the IPL Simulator.

```
batch
$RFL,20000.
/get,source
/get,ipliswl/un=all
/-ipliswl(i=source,b=binary,o=listing,fl=134000)
/get,utl/un=all
/utl,1010,listing          "use 10xx for listing RevC files"
/get,sesins1=yourpr1       "get user's LNS environment"
/get,sesins2=yourpr2
/get,sesins3=yourpr3
/get,runses/un=all
/ascii
/-runses(ins=1)            "enter subsystem to execute LINK and RML"
**MSG1: SES V1.1 75/09/02.  08.00.00.  - PLEASE LOGIN
? login you
                           "LINK_CB and LIF have been previously
                           declared and saved as part of the LNS
                           environment."
? link_cb.name_seed        "check on prefix for subsequent simulator
                           command file"
lxxx
? lif(1).kfn='binary'
     .
     .                     "files to be linked"
     .
? link,lcb=link_cb,lpd=to_rml
? rml,lpd=to_rml           "use LPD name quoted on LINK command"
? logoff                   "exit subsystem to run simulator"
**MSG 84: CONNECT TIME = 00.03.01.
**END RUNSES
/normal                    "simulator does not accept lower case"
/get,runsim/un=all
/-runsim
IPL SIMULATOR VERSION 2.1
 MEMORY=00001FFF
? enter lxxxent            "RML generated command file"
ENTER LXXXENT
MEMORY 11000
IPL SIMULATOR VERSION 2.1
 MEMORY=10006
LOAD LXXXZ01
LOAD LXXXZ02
LOAD LXXXZ03
```

```
LOAD LXXXZ04
LOAD LXXXZ05
ECHO OFF
          00002600  02 EXCH    02000000       P = F0F0F0F0
      P  F0F0 F0F0 F0F0 F0F0       MCR  F0F0          UCR  F0F0
    BKP  0000 0000 0000          STATE  MONITOR
 TRACE LL 0000 0000 0000        TRACE UL 0000 0000 0000
CR A=3,AV=800000005002
CR S=P,SV=800000004044
EXIT
     .
     .                  "Simulator commands/Program Execution"
     .
? end
END
 SIMULATION COMPLETE.
/
```

10.3 NORMAL TERMINATION

A normal procedure epilog (see object code generated for a procend) is generated for procedures with the "main" or "xdcl" attributes. Therefore, if the environment was established as it is described in the section entitled "INITIATION", and the program executed correctly, an environment specification error will occur (MCR=0100) and the P address should be the address of the "return" instruction of the procedure to which control was transferred. The IPL Simulator will then prompt the user for a command.

10.4 ABNORMAL TERMINATION

Through the use of comment toggles (see section entitled "IPLISWL Compiler", in the SES User's Handbook) the compiler generates code to check the following conditions:

CONDITION
Assignment subrange error
Divide by zero

CONDITION
Index out of range
Case index out of range

If one of the above conditions is detected, the compiler assigned source line number at which the error occurred is entered into register X1 and a call is made to a runtime procedure which outputs an error message onto the file, IPLOUT, and executes a program error instruction. Execution aborts and the user is then prompted for a command by the IPL Simulator.

The runtime error messages are the following:

```
ASSIGNMENT SUBRANGE ERROR AT LINE XXXXX
CASE INDEX ERROR AT LINE XXXXX
INDEX OUT OF RANGE ERROR AT LINE XXXXX
DIVIDE BY ZERO ERROR AT LINE XXXXX
```

The field, XXXXX, is the LINE field on the "source form" of the listing file (see section entitled, "LISTING FILE").

11.0 APPENDIX H - MACHINE BREAKOUT EXAMPLE

11.0 APPENDIX H - MACHINE BREAKOUT EXAMPLE

This appendix contains an ISWL/CI "listing file" of a source program which exemplifies the usage of machine breakout.

11.0 APPENDIX H - MACHINE BREAKOUT EXAMPLE

```
PG  1  STAT   AUTO PR NS 1  ISWL/IPL  VERSION 2.0, LEVEL 0

    1     0      0  0  0 MODULE [CODE(IPL)] MBEXAMP;
    2     0      0  0  0   "$M80"
    3     0      0  0  0 TYPE
    4     0      0  0  0  REC1 = RECORD
    5     0      0  0  0            BIT1,BIT2 : #BIT(5,2),
    6     0      0  0  0            BYTE1 : #BYTE(5,4),
    7     0      0  0  0            ADR1 : #ADDRESS,
    8     0      0  0  0            HDESC : #DESCRH,
    9     0      0  0  0            FDESC : #DESCRF
   10     0      0  0  0           RECEND;
   11     0      0  0  0 VAR
   12     0      0  0  0   VREC : REC1 := [#M('5'),#M('5'),#M('1 2 3 4 5 6 7 8 9 A'),
   13    32      0  0  0                      [#M('4'),#M('8'),#M('E')],
   14    32      0  0  0                      [#M('0'),#M('8'),#M('4'),#M('8'),#M('1 7')],
   15    32      0  0  0                      [[#M('0'),#M('4'),#M('4'),#M('8'),#M('E')],#M('2 3 4 5')]];
   16    32      0  0  0  VBIT : #BIT(31,33),
   17    40      0  0  0  VBYTE  : #BYTE(7,4),
   18    47      0  0  0  VADR : #ADDRESS,
   19    53      0  0  0  VHDESC : #DESCRH,
   20    60      0  0  0  VFDESC : #DESCRF;
   21    72      0  0  0
   22    72      0  0  0 PROC [CODE(IPL),MAIN] MBEXAMP;
   23    72      6  1  0 !PROLOG;
   24    72      6  1  0 VAR
   25    72      6  1  0  X4 : [#XREG(4)] INTEGER,
   26    72      6  1  0  X3 : [#XREG(3)] REAL,
   27    72      6  1  0  X2 : [#XREG(2)] CHAR,
   28    72      6  1  0  A4 : [#AREG(4)] #ADDRESS,
   29    72      6  1  0  A5 : [#AREG(5)] #ADDRESS,
   30    72      6  1  0  AA : [#AREG(0A(16))] #ADDRESS,
   31    72      6  1  0  AB : [#AREG(0B(16))] #ADDRESS,
   32    72      6  1  0  AC : [#AREG(0C(16))] #ADDRESS,
   33    72      6  1  0  AD : [#AREG(0D(16))] #ADDRESS,
   34    72      6  1  0  A8,A9 : [#AREG(8..9)] #ADDRESS,
   35    72      6  1  0  VBOOL : BOOLEAN,
   36    72      7  1  0  I : INTEGER,
   37    72     11  1  0  PI1 : ^INTEGER;
   38    72     17  1  0
   39    72     17  1  0  VREC.BYTE1 := #M('A B C D E A B C D E');
   40    72     17  1  0  PI1 := VADR;
   41    72     17  1  0  VADR := $#ADDRESS[#M('0'),#M('0'),#M('3 2 1 0')];
   42    72     23  1  0  VHDESC := $#DESCRH[#M('8'),#M('4'),#M('F'),#M('2 5 8'),#M('2 4')];
   43    72     24  1  0  VFDESC := $#DESCRF[[#M('0'),#M('C'),#M('0'),#M('250'),#M('24')],#M('4096')];
   44    72     32  1  0  LBL1 : VBOOL := VREC = VREC;
   45    72     17  1  0  LBL2 : VBOOL := VFDESC /= VFDESC;
   46    72     17  1  0  I := #OFB(VREC.ADR1,VREC);
   47    72     17  1  0  I := #OFB(PI1);
   48    72     17  1  0  I := #OFB(VADR);
   49    72     17  1  0  I := #OFBIT(VREC.ADR1,VREC);
   50    72     17  1  0  I := #OFBIT(PI1);
   51    72     17  1  0  I := #OFBIT(VADR);
   52    72     17  1  0  I := #LENG(VREC);
   53    72     17  1  0  I := #LENGBIT(VREC);
   54    72     17  1  0  PI1 := #ADDR(I);
```

11.0 APPENDIX H - MACHINE BREAKOUT EXAMPLE

```
PG  2  STAT  AUTO PR NS 1  ISWL/IPL  VERSION 2.0, LEVEL 0

   55    72    17  1  0  !LOCKREG;
   56    72    17  1  0  !LCAD,7                    X4,[AC,X3,1024];
   57    72    17  1  0  !LOAD,7                    X4,[AC,4096];
   58    72    17  1  0  !STORE,7                   X4,[AC,X3,1024];
   59    72    17  1  0  !STORE,7                   X4,[AC,4096];
   60    72    17  1  0  !LOAD                      X4,[AC,X3,1024];
   61    72    17  1  0  !LOAD                      X4,[AC,4096];
   62    72    17  1  0  !STORE                     X4,[AC,X3,1024];
   63    72    17  1  0  !STORE                     X4,[AC,4096];
   64    72    17  1  0  !LCAD,#X0                  X3,[AC,X2,1024];
   65    72    17  1  0  !LOAD,#X0                  X3,[AC,4096];
   66    72    17  1  0  !STORE,#X0                 X3,[AC,X3,1024];
   67    72    17  1  0  !STORE,#X0                 X3,[AC,4096];
   68    72    17  1  0  !LCAD,7                    X3,[#P,4096];
   69    72    17  1  0  !LOAD,#BOOL               X3,[AC,4096];
   70    72    17  1  0  !STORE,#BOOL              X3,[AC,4096];
   71    72    17  1  0  !LOAD                      AC,[AD,X4,1024];
   72    72    17  1  0  !LOAD                      AC,[AD,4096];
   73    72    17  1  0  !STORE                     AC,[AD,X3,1024];
   74    72    17  1  0  !STORE                     AC,[AD,4096];
   75    72    17  1  0  !LOAD,#MULT               [AC,5];
   76    72    17  1  0  !STORE,#MULT              [AC,5];
   77    72    17  1  0  !ADD                       X4,X3;
   78    72    17  1  0  !SUB                       X4,X3;
   79    72    17  1  0  !MULT                      X4,X3;
   80    72    17  1  0  !DIV                       X4,X3;
   81    72    17  1  0  !ABS                       X4,X3;
   82    72    17  1  0  !ADD,#H                    X4,X3;
   83    72    17  1  0  !ADD,#H                    X4,[X3,4096];
   84    72    17  1  0  !ADD,#H                    X4,5;
   85    72    17  1  0  !SUB,#H                    X4,X3;
   86    72    17  1  0  !SUB,#H                    X4,5;
   87    72    17  1  0  !MULT,#H                   X4,X3;
   88    72    17  1  0  !MULT,#H                   X4,[X3,4096];
   89    72    17  1  0  !DIV,#H                    X4,X3;
   90    72    17  1  0  !COMP                      X4,X3;
   91    72    17  1  0  !COMP,#H                   X4,X3;
   92    72    17  1  0  !BREQ                      X4,X3,4096;
   93    72    17  1  0  !BRNE                      X4,X3,#OFP(LBL1);
   94    72    17  1  0  !BRGT                      X4,X3,4096;
   95    72    17  1  0  !BRGE                      X4,X3,#OFP(LBL2,LBL1);
   96    72    17  1  0  !BREQ,#H                   X4,X3,4096;
   97    72    17  1  0  !BRNE,#H                   X4,X3,4096;
   98    72    17  1  0  !BRGT,#H                   X4,X3,4096;
   99    72    17  1  0  !BRGE,#H                   X4,X3,4096;
  100    72    17  1  0  !DOLOOP                    X4,X3,4096;
  101    72    17  1  0  !BR                        AC,AD,4096;
  102    72    17  1  0  !BR                        [#P,X4];
  103    72    17  1  0  !BR                        AC,X4;
  104    72    17  1  0  !COPY                      X4,X3;
  105    72    17  1  0  !COPY                      X4,AC;
  106    72    17  1  0  !COPY                      AC,AD;
  107    72    17  1  0  !COPY                      AC,X4;
  108    72    17  1  0  !COPY,#H                   X4,X3;
  109    72    17  1  0  !AOCA                      AC,[AD,4096];
```

11.0 APPENDIX H - MACHINE BREAKOUT EXAMPLE

```
PG  3  STAT  AUTO PR NS 1  ISWL/IPL  VERSION 2.0, LEVEL 0

  110    72    17  1  0  !ADDA,#P                    AC,[X2,#OFP(LBL1)];
  111    72    17  1  0  !ADDA                       AC,X3;
  112    72    17  1  0  !ENTER,#POS                 X4,15;
  113    72    17  1  0  !ENTER,#NEG                 X4,15;
  114    72    17  1  0  !ENTER,#EXTD                X4,4096;
  115    72    17  1  0  !ENTER,#X0                  78;
  116    72    17  1  0  !ENTER,#SIGN                X4,15;
  117    72    17  1  0  !ROTATE                     X4,X3,[X2,10];
  118    72    17  1  0  !SHIFT                      X4,X3,[X2,10];
  119    72    17  1  0  !SHIFT,#H                   X4,X3,[X2,10];
  120    72    17  1  0  !OR                         X4,X3;
  121    72    17  1  0  !XOR                        X4,X3;
  122    72    17  1  0  !AND                        X4,X3;
  123    72    17  1  0  !NOT                        X4,X3;
  124    72    17  1  0  !INHIB                      X4,X3;
  125    72    17  1  0  !MASK                       X4,[X3,10];
  126    72    17  1  0  !ISOLATE                    X4,X3,[X2,10];
  127    72    17  1  0  !INSERT                     X4,X3,[X2,10];
  128    72    17  1  0  !MOVE                       AA,AB;
  129    72    17  1  0  !ADD                        [A4,A8,100],[A5,A9,100];
  130    72    17  1  0  !SUB                        [A4,A8,100],[A5,A9,100];
  131    72    17  1  0  !MULT                       [A4,A8,100],[A5,A9,100];
  132    72    17  1  0  !DIV                        [A4,A8,100],[A5,A9,100];
  133    72    17  1  0  !SCALE                      [A4,A8,100],[A5,A9,100];
  134    72    17  1  0  !SCALEROUND                 [A4,A8,100],[A5,A9,100];
  135    72    17  1  0  !COMP,#DEC                  [A4,A8,100],[A5,A9,100];
  136    72    17  1  0  !COMP                       [A4,A8,100],[A5,A9,100];
  137    72    17  1  0  !COMP,#COLL                 [A4,A8,100],[A5,A9,100];
  138    72    17  1  0  !SCAN                       [A4,A8,100],[A5,A9,100];
  139    72    17  1  0  !TRANSLATE                  [A4,A8,100],[A5,A9,100];
  140    72    17  1  0  !MOVE                       [A4,A8,100],[A5,A9,100];
  141    72    17  1  0  !EDIT                       [A4,A8,100],[A5,A9,100];
  142    72    17  1  0  !MOVE,#NUM                  [A4,A8,100],[A5,A9,100];
  143    72    17  1  0  !MOVE,#FORMAT,#FORMAT       [A4,A8,100],[A5,A9,100];
  144    72    17  1  0  !ADD,#F,#F                  [A4,A8,100],[A5,A9,100];
  145    72    17  1  0  !CALCSUB                    [A4,A8,100],[A5,A9,100];
  146    72    17  1  0  !CONV,#R                    X4,X3;
  147    72    17  1  0  !CONV                       X4,X3;
  148    72    17  1  0  !ADD,#R                     X4,X3;
  149    72    17  1  0  !SUB,#R                     X4,X3;
  150    72    17  1  0  !ADD,#R,#UN                 X4,X3;
  151    72    17  1  0  !SUB,#R,#UN                 X4,X3;
  152    72    17  1  0  !MULT,#R                    X4,X3;
  153    72    17  1  0  !DIV,#R                     X4,X3;
  154    72    17  1  0  !ADD,#DR                    X4,X3;
  155    72    17  1  0  !SUB,#DR                    X4,X3;
  156    72    17  1  0  !MULT,#DR                   X4,X3;
  157    72    17  1  0  !DIV,#DR                    X4,X3;
  158    72    17  1  0  !BREQ,#R                    X4,X3,4096;
  159    72    17  1  0  !BRNE,#R                    X4,X3,4096;
  160    72    17  1  0  !BRGT,#R                    X4,X3,4096;
  161    72    17  1  0  !BRGE,#R                    X4,X3,4096;
  162    72    17  1  0  !BR,#UNDFLO                 X3,4096;
  163    72    17  1  0  !COMP,#R                    X4,X3;
  164    72    17  1  0  !CALL                       [AA,15],AB,AC;
```

11.0 APPENDIX H - MACHINE BREAKOUT EXAMPLE

```
PG  4  STAT  AUTO PR NS 1  ISWL/IPL  VERSION 2.0, LEVEL 0

  165    72    17  1  0  !CALL                          [#P,4096],AB,AC;
  166    72    17  1  0  !RETURN;
  167    72    17  1  0  !POP;
  168    72    17  1  0  !EXCHANGE;
  169    72    17  1  0  !ERROR;
  170    72    17  1  0  !INTRPT                        X3;
  171    72    17  1  0  !INTRPTPROD                    X3;
  172    72    17  1  0  !LOAD,#SET                     X4,AA;
  173    72    17  1  0  !COMP                          X4,AA;
  174    72    17  1  0  !COMP,#PAGE;
  175    72    17  1  0  !LOAD,#PAGE                    X4,X3;
  176    72    17  1  0  !LOAD,#STATE                   X4,X3;
  177    72    17  1  0  !STORE,#STATE                  X4,X3;
  178    72    17  1  0  !LOAD,#MAINT                   X4,X3;
  179    72    17  1  0  !STORE,#MAINT                  X4,X3;
  180    72    17  1  0  !BR,#COND                      61,4096;
  181    72    17  1  0  !KEYPOINT                     X3,2,4;
  182    72    17  1  0  !PURGE                        X4,3;
  183    72    17  1  0  !ALGORITHM,7                  A8,A9,16;
  184    72    17  1  0  !ADD,#IMM                     X4,[X3,4096];
  185    72    17  1  0  !COMP                          AA,AB;
  186    72    17  1  0  !CONV,#EQ                      X4;
  187    72    17  1  0  !STORE,#KIND                  [A4,A8,100],X4;
  188    72    17  1  0  !LOAD,#OFFSET                 X4,[A4,A8,100];
  189    72    17  1  0  !ADD,#INDEX                   [A4,A8,100],X4;
  190    72    17  1  0  !SUB,#INDEX                   [A4,A8,100],X4;
  191    72    17  1  0  !AND                          AC,A8;
  192    72    17  1  0  !OR                           AC,A8;
  193    72    17  1  0  !XOR                          AC,A8;
  194    72    17  1  0  !MCC                          AC,A8;
  195    72    17  1  0  !COPY,#ALPHA                  [A4,A8,100],69;
  196    72    17  1  0  !COMP,#ALPHAFILL              [A4,A8,100],68;
  197    72    17  1  0  !ADD,#BIN                     [A4,A8,100],67;
  198    72    17  1  0  !LOAD                         AD,[A4,A8,100];
  199    72    17  1  0  !LOAD                         X4,[A4,A8,100];
  200    72    17  1  0  !STORE                        X4,[A4,A8,100];
  201    72    17  1  0  !IO                           A8,1,2;
  202    72    17  1  0 "END TESTING MACHINE CODE"
  203    72    17  1  0  !EPILOG;
  204    72    17  1  0 PROCEND MBEXAMP;
  205    72     0  0  0 MODEND MBEXAMP
75/08/29.  09.52.04
```

11.0 APPENDIX H - MACHINE BREAKOUT EXAMPLE

```
S.LN OPC  SECT/OFFSET --TEXT-- IPL OP REF

 204 DEF                                  EPT CS 00032A R MBEXAMP
 205 FIN                                  AIN BS 000000 P A(ST)
                                          AIN BS 000008 P A(LT)


            SECTION MAP OF MODULE: MBEXAMP

              0 CS 000348 CODE SECTION          READ      EXECUTE
              3 BS 000010 BINDING SECTION       READ      BINDING
              2 LT 000017 WORKING SECTION       READ
              1 ST 000048 WORKING SECTION       READ      WRITE
          TRANSFER NAME: MBEXAMP
```

11.0 APPENDIX H - MACHINE BREAKOUT EXAMPLE

LINE	OFFSET	LINE	OFFSET	LINE	OFFSET	LINE	OFFSET
1	CS+000000	92	CS+0001D8	147	CS+00028E	204	CS+000328
23	CS+000080	93	CS+0001DC	148	CS+000290	205	CS+00033E
39	CS+000000	94	CS+0001E0	149	CS+000292		
40	CS+00001A	95	CS+0001E4	150	CS+000294		
41	CS+000026	96	CS+0001E8	151	CS+000296		
42	CS+000068	97	CS+0001EC	152	CS+000298		
43	CS+00004A	98	CS+0001F0	153	CS+00029A		
44	CS+0000EC	99	CS+0001F4	154	CS+00029C		
45	CS+00010A	100	CS+0001F8	155	CS+00029E		
46	CS+000122	101	CS+0001FC	156	CS+0002A0		
47	CS+000128	102	CS+000200	157	CS+0002A2		
48	CS+00012E	103	CS+000202	158	CS+0002A4		
49	CS+000134	104	CS+000204	159	CS+0002A8		
50	CS+00013C	105	CS+000206	160	CS+0002AC		
51	CS+000144	106	CS+000208	161	CS+0002B0		
52	CS+00014C	107	CS+00020A	162	CS+000294		
53	CS+000154	108	CS+00020C	163	CS+0002B8		
54	CS+00015C	109	CS+00020E	164	CS+0002BA		
55	CS+000166	110	CS+000212	165	CS+0002BE		
56	CS+000166	111	CS+000216	166	CS+0002C2		
57	CS+00016A	112	CS+000218	167	CS+0002C4		
58	CS+00016E	113	CS+00021A	168	CS+0002C6		
59	CS+000172	114	CS+00021C	169	CS+0002C8		
60	CS+000176	115	CS+000220	170	CS+0002CA		
61	CS+00017A	116	CS+000222	171	CS+0002CC		
62	CS+00017E	117	CS+000224	172	CS+0002CE		
63	CS+000182	118	CS+000228	173	CS+0002D0		
64	CS+000186	119	CS+00022C	174	CS+0002D2		
65	CS+00018A	120	CS+000230	175	CS+0002D4		
66	CS+00018E	121	CS+000232	176	CS+0002D6		
67	CS+000192	122	CS+000234	177	CS+0002D8		
68	CS+000196	123	CS+000236	178	CS+0002DA		
69	CS+00019A	124	CS+000238	179	CS+0002DC		
70	CS+00019E	125	CS+00023A	180	CS+0002DE		
71	CS+0001A2	126	CS+00023E	181	CS+0002E2		
72	CS+0001A6	127	CS+000242	182	CS+0002E6		
73	CS+0001AA	128	CS+000246	183	CS+0002E8		
74	CS+0001AE	129	CS+000248	184	CS+0002EC		
75	CS+0001B2	130	CS+00024C	185	CS+0002F0		
76	CS+0001B4	131	CS+000250	186	CS+0002F2		
77	CS+0001B6	132	CS+000254	187	CS+0002F4		
78	CS+0001B8	133	CS+000258	188	CS+0002F8		
79	CS+0001BA	134	CS+00025C	189	CS+0002FC		
80	CS+0001BC	135	CS+000260	190	CS+000300		
81	CS+0001BE	136	CS+000264	191	CS+000304		
82	CS+0001C0	137	CS+000268	192	CS+000306		
83	CS+0001C2	138	CS+00026C	193	CS+000308		
84	CS+0001C6	139	CS+000270	194	CS+00030A		
85	CS+0001C8	140	CS+000274	195	CS+00030C		
86	CS+0001CA	141	CS+000278	196	CS+000310		
87	CS+0001CC	142	CS+00027C	197	CS+000314		
88	CS+0001CE	143	CS+000280	198	CS+000318		
89	CS+0001D2	144	CS+000284	199	CS+00031C		
90	CS+0001D4	145	CS+000288	200	CS+000320		
91	CS+0001D6	146	CS+00028C	201	CS+000324		