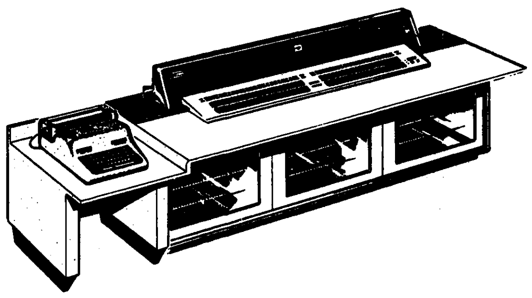
Serial No. 141

TECHNICAL MANUAL

transac®

S-2000



PHILCO CORPORATION
COMPUTER AND AUTOMATION DEPARTMENT
4700 WISSAHICKON AVENUE
PHILADELPHIA 44, PENNSYLVANIA

PREFACE

The S-2000 Technical Manual is primarily intended for training and reference use by TRANSAC Field Engineers. It can be used as well by other groups desiring a detailed description of the system.

The material is arranged to facilitate study and will serve as a comprehensive technical introduction to TRANSAC S-2000. The scope of this manual excludes those areas covered by other S-2000 publications, as the Programming Manual.

Inquiries concerning this manual should be directed to

Jerome Stone

Computer and Automation Department

TRANSAC S-2000 TECHNICAL MANUAL

PART 1 COMPUTER LOGICAL DESCRIPTION

TABLE OF CONTENTS

Table of Contents (continued)

Table of Contents (continued)

LIST OF ILLUSTRATIONS

LIST OF TABLES

1. ORGANIZATION OF THE COMPUTER

1.1 Block Diagram, Information Storage and Control

Figure 1.1-1 is a block diagram of the computer showing the major information storage registers, control and paths of information flow. The notes define some symbols which may be unfamiliar. This section describes the various units shown to give an overall view of the functions of the computer.

Control Information Storage

The organization of the control section of the central computer requires the following registers for storage of control information. The abbreviation follows the name of the register.

Program Register (PR)

The instruction about to be, or currently being, performed is located in the Program Register. PR is a 48-bit register, storing an instruction word (two instructions). The input to PR is usually from the internally stored program in memory. Direct access into PR is also available from the console with the "PR Keyboard" manually operated switches. The left instruction location of PR (bits 2_0 through 2_{-23}) is identified as PR_0. It contains the I_0 instruction. The other location, PR_1, contains the I_1 instruction.

Switch 2 (SW2)

The selection of which of the pair of instructions in PR is to be operative is made by SW2. SW2 contains a flip-flop, with that name. When SW2 = 0, I_0 is the operative instruction. Through SW2, the parts of the selected instruction are distributed among the various controls.

Switches

Functionally a switch as discussed here has two parts. The switch is a means of controlling a multitude of information signal lines that exist in a parallel computer. The control signal indicating the direction is a single line and it is usually desired to maintain the connection over a period of time.

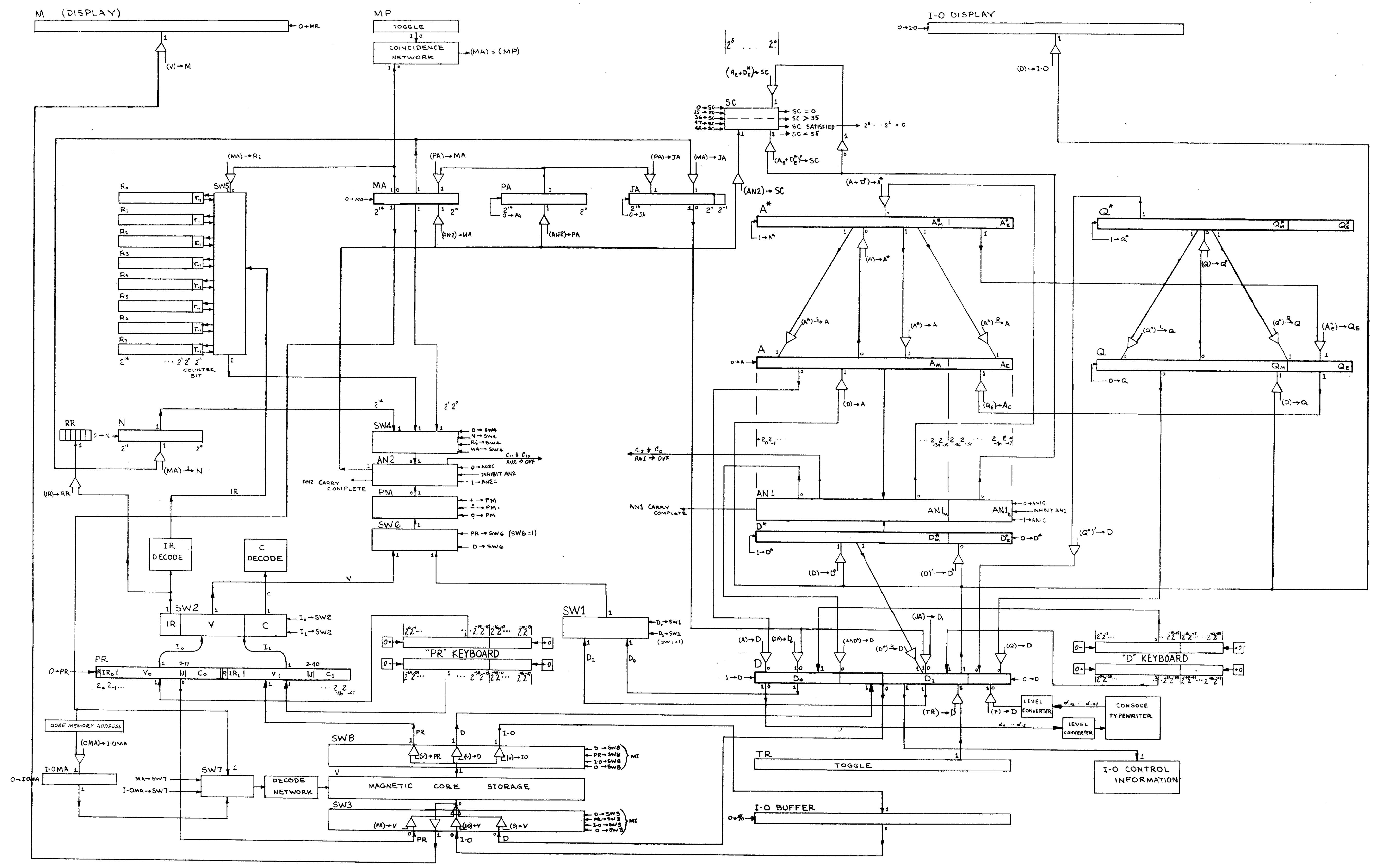


Figure 1.1-1 Block Diagram Central Computer

The switch usually has one or more flip-flops which are set to the required state by the control signal. The flip-flop output(s) go to a large number of AND gates where they permit or inhibit the transfer of information. The selected source or destination of the information will thus be determined by the state of the flip-flops. Some switches do not use flip-flops for selection control. In this case a decode network serves as selection control.

SW2, for example, is a single flip-flop as its order of selection is twofold. The flip-flop inputs are two. Each side of the SW2 flip-flop goes to 24 gates by which means I_0 or I_1 is transferred from PR.

Distribution of the Parts of the Current Instruction

The command part of the instruction, C, is routed to the command decoding network. The memory address part of the instruction, I_v, is routed to the Memory Address Register (MA) through the adder network, AN2 (a general purpose routing for this part of any instruction from PR to elsewhere in the control section). The index register address part of the instruction, IA, is routed to its decoding when an index register is involved in the instruction.

For the repeat instruction, the four leftmost bits, $\alpha\ \beta\ \gamma\ \delta$ (which indicate the type of repeat) are sent to the Repeat Register (RR) for control use. The I_v field of the repeat instruction, which indicates the number of times an instruction (or pair of instructions) is to be repeated, is sent to the Repeat Counter (N) by an indirect route via MA.

Adder Network for Control Information (AN2)

AN2 serves two basic functions. It is an adder network, one of the two in the central computer. The term, adder network, is used to signify a logical adder which has no registers, no storage capabilities. AN2 is used for addition or subtraction involving I_v and IA parts of the instruction. (IA, in this case, is part of a memory address, rather than an index register address.

This function of AN2 requires it to be connected to many registers of the control section. These existing information transfer paths can therefore be used for transfers involving these registers without arithmetic operations with a resulting saving in circuitry. Arithmetic is avoided during such a transfer by simply gating only one, instead of two register inputs to AN2. AN2 acts, for transfers, as a junction. Zeros are gated to the unused input.

AN2 Inputs

The inputs to AN2 are controlled by SW4 (Switch 4) and PM (Plus-Minus). SW4 is the means of selecting one of three sources as one input to AN2. They are N, MA, and X (selected index register). A fourth state of SW4 selects none of these; instead it gates zeros to AN2. SW4 contains two flip-flops.

PM also contains two flip-flops. It controls the other input to AN2 and can select either the number or its ones complement (for subtraction) as the quantity to be sent to AN2. If PM is " - " (one of the four states of the two flip-flops) the ones complement of the number is sent to AN2. If PM is "+ ", the number, uncomplemented, is transferred. If PM is "0", zeros are sent to this input

Address Register Size

In any individual S-2000 system the size of AN2, MA, PA., and X will be in accordance with the size of the memory in that system. As mentioned previously, X and JA each have an additional bit --x_c is the counter bit of an index register; ja_{-1} is the half-word identification in JA.

Index Registers (X = the selected index register)

The index registers store information which may be used to determine an effective address, in conjunction with the address part of the instruction. The effective address would be the actual one used in that particular performance of an instruction. At other times X, itself, can furnish the effective address. The index register can be incremented by 1, following each use, by means of the counter bit. When $x_c = 1$ (counter bit = 1) the register contents will be so incremented.

Under the repeat mode, in addition to the above use, the index register can be "repeat modified". The index register alone furnishes the address, in this case, and following each instruction performance the register is either incremented or decremented by the contents of the "address" part of the instruction being repeated.

An S-2000 system may contain up to 32 index registers, in groups of four. Information is transferred by means of SW5. SW5 has no flip-flops bearing that name. Selection is made by the Index Register Decode network to gate information to or from X. The input is always through MA; X's output goes only to AN2 via SW4.

Incrementing or decrementing the register contents is achieved by a pass through AN2. The register has no separate counting ability.

The number into the PM gating is either from PR or the D register. SW6 controls this selection. SW6 = 1 selects PR; SW6 = 0 selects D. SW2 selects one of the two halves of PR; I_0 or I_1 as the input to the SW6 gating. SW1 performs a similar function for the D register. SW1 = 0 selects the D_0 half; SW1 = 1 selects D_1. The D register has a connection to AN2 as it is used to handle control information (memory addresses) as well as its more common function of handling data.

Aside from the index registers, there are four other registers that store address information. MA contains the address currently used, or last used for memory access when the address was supplied by the central computer. Memory address can also be supplied by an input-output unit. In this case the I-O MA contains the current or last used address. The selection between MA and I-O MA is made at SW7.

PA (Program Address Register) stores the address of the next instruction word to be used. In the normal sequence this address is increased by one each time a new instruction word is transferred from memory. Jumps break this sequence; the address to which the program jumps is placed in PA during the execution of the jump instruction.

JA (Jump Address Register) might also be termed the "return address register". It is used to store the address of the next instruction in normal sequence that was not performed due to the program jump. This is required,as a common characteristic of programs is to leave the main part of the program, by means of a jump, to perform a sub-routine. Following the sub-routine, a return to the main program is desired. The contents of JA are the reference for the return. As JA refers to an instruction location, rather than a word location, the register contains in addition to the address, one more bit to indicate which half of the word contains the next instruction. (PA does not require this as there is a normal sequence of performance within the instruction word, I_0, I_1.)

An address is transferred to JA from either MA or PA. If the jump instruction is I_0 the instruction word containing it was just transferred to PR from memory. Its address is still in MA where it was used to access the memory. This address is transferred to JA as well as an indication that the next instruction in normal sequence is I_1 of that address.

$$(MA) \longrightarrow JA$$

$$1 \longrightarrow ja_{-1}$$

If the jump instruction were I_1, the address of the next instruction in normal sequence is stored in PA.

$$(PA) \longrightarrow JA$$

$$0 \longrightarrow ja_{-1}$$

JA is connected to D so that the address may be transferred to memory. The customary procedure in a sub-routine, immediately upon entering it, is to transfer the address in JA to the address part of a jump instruction at the end of the sub-routine. Thereupon, the exit from the sub-routine will be back to the point of departure from the main program.

Normal Sequencing of Address

The incrementing of the program address for normal sequence is done as follows:

$(PA) \longrightarrow MA$	address of next instruction word
$(MA) \longrightarrow SW7 \longrightarrow$ Decode	access memory to transfer next instruction word
$(MA) \longrightarrow AN2$)) $1 \longrightarrow AN2$)	add 1 to (MA)
$AN2 \longrightarrow PA$	store new address of next instruction word

In a shift operation, for example, the register cannot simultaneously store two different things, the original number and the shifted number. The A register is one of the adder network inputs. The sum is stored in A. A cannot simultaneously store both numbers. (The adder network, AN1, is inherently incapable of storage.)

Each of these data registers is therefore double ranked. Each consists of a pair of one word registers, A and A^*, D and D^*, Q and Q^*. Addition $(A) + (D) \longrightarrow A$ is performed by transferring D to D^*. $(A) + (D^*)$ are the inputs to AN1. AN1 output to A^*, A is cleared; and the sum in A^* is transferred to A. Shifting of A is performed by transferring A to A^*, clearing A, transferring A^* to A through the shift gates, right or left.

Table 1.1-1 lists the functions of these registers in arithmetic operations.

Repeat Register (RR)

The repeat register is a 6-bit register storing control information for operation in the repeat mode. Its contents are:

R	α	β	γ	δ	I

where:

R = 1	indicates repeat mode operation.
I	indicates whether one or a pair of instructions is being repeated. (I = 0 signifies that both instructions in the word should be performed).
$\alpha\ \beta\ \gamma\ \delta$	specify the type of repeat (modification or not of X, and manner of modification).

Repeat Counter (N)

N is a 12-bit register, enabling the repetitive performance of an instruction up to 4095 times. N does not possess counting ability.

TABLE 1.1-1

Operation	A	D	Q
Addition			
Before operation	Augend	Addend	
After operation	Sum	Addend	
Subtraction			
Before operation	Minuend	Subtrahend	
After operation	Difference	Subtrahend	
Multiplication Rounded Product			
Before operation		Multiplicand	Multiplier
After operation	Product	Multiplicand	Multiplier
Multiplication Double-length Product			
Before operation		Multiplicand	Multiplier
After operation	Major Product	Multiplicand	Minor Product
Division, Single-length dividend - Before operation	Dividend	Divisior	
Division, Double-length dividend -Before operation	Major dividend	Divisor	Minor dividend
Division, Both size dividends - After operation	Remainder	Divisor	Quotient

It stores a number (in twos complement form) which indicates how many times an instruction, or pair of instructions, is to be performed. The number is incremented by use of AN2 and returned to N via MA.

Memory Preset (MP)

Memory Preset consists of a bank of double throw toggle switches on the console provided primarily as a programming facility. The MP switch levels are continuously compared with the contents of MA and coincidence will be recognized to stop the computer, if desired.

Data Registers

Three registers are used to handle the data word in the computer, A, D and Q. (D is also used to handle an instruction word if part of the word is to be changed.) The nature of a parallel computer requires an additional storage media for these registers. During a shift operation, for example, a register cannot simultaneously store the original and shifted word.

Each of these registers has another 48-bit register associated with it, A^*, D^* and Q^*, to provide the required additional storage. These are spoken of as the "star" registers ("A Star", etc.).

CMA

The Core Memory Address register stores addresses for paper tape, magnetic drum and printer access to computer memory. The separate register is required as the input-output operation time-shares the computer with many internal instructions, the latter having other addresses.

The magnetic tape control also requires an individual memory address register. This is located in its control unit, outside the computer.

Switch 3 and Switch 8

These switches route information to and from memory. Switch 8 (SW8) routes the read-out from memory to the desired destination, PR, D, or IOB (Input-Output Buffer). Switch 3 (SW3) connects a register for writing-in from memony, PR, D, or IOB. The M register is for display only.

Other Controls

Several other controls are not illustrated. One group consists of the Fault flip-flops. When any of these are set to one, they will stop computer operation.

Command Fault (CF)	Command coding in PR which is not assigned to any instruction.
Memory Fault (MF)	Temperature of memory unit is not in operating range and operation is unreliable.
Exponent Fault (EF)	The result of an algorithm with floating point numbers has resulted in a quantity that cannot be represented by the computer.

The STOP flip-flop, when set, will halt the computer. The flip-flop is set either by the HALT instruction (HLT) or operator intervention (STOP switch in lower center of console)

The OVERFLOW (OVF) flip-flop is set by overflow conditions. The JOF instruction requires the computer to jump if overflow. The OVERFLOW console switch in the ON position will stop the computer, otherwise.

The JUMP flip-flop (JFF) is set when a jump is to be executed. It controls the details of the execution.

1.2 Control Organization

The performance of an instruction, or a series of instructions forming a program, consists of a larger sequence of what may be termed as "operations.". The basic program cycle operations would consist of selecting an instruction and performing it. The instructions themselves are a sequence of operations, perhaps involving transfer of data, followed by arithmetic or non-arithmetic manipulation of the data and possibly concluding with a transfer of the result from a register to memory or another register.

Among the operations of the instructions and the program cycle there are many identities and similarities. The complexity of the computer control logic can be minimized by an organization wherein the control specifies operations. An operation can be defined as some part of an instruction or part of the program cycle. The instruction calls for the proper sequence and number of operations. Some of the operations possible within an instruction have been previously mentioned. The operations control signal lines which cause groups of tasks to be performed.

The smallest piece of logical work performed by the computer can be termed a "task". The task is a change or transfer of information. The transfer changes the condition of some information storage unit. There are two types of information, data and control.

A criterion for determining the scope of a task, in the logical design of the computer, is similar to that of determining the least common denominator in arithmetic. The scope of a task should be chosen so that it can be used for operations in many instructions.

An instruction is performed as a group of operations subdivided into groups of tasks for two basic reasons. First, the instruction usually requires several operations that cannot be simultaneously performed. These operations must be sequenced in a pre-determined fashion. The program itself, is a sequence of instructions and timing therefore enters into program control.

Secondly, due to the similarities among many instructions, the sub-division into tasks can be organized so that tasks will be identical for similar instructions. The operation is then a group of performed tasks, some simultaneously, others in a predetermined sequence.

In the broadest sense there are six groups of operations. Listing them by their control names:

Program Control

Instruction Control

Algorithm Control

Floating Point Control

Memory Cycle Control

Input-Output Control

Program Control

This is the group name for the operations involved in maintaining the desired sequence of instructions. These include transfer of the instruction word from memory to the program register (where it is stored for execution), selection of the proper one of the two instructions within the instruction word, establishing the location of the next instruction, index register modification.

Instruction Control

This group of operations consists of those preparatory to the performance of any instruction as well as those necessary prior to specific instructions. In the case of non-arithmetic instructions, this control may cover the performance of the entire instruction.

Algorithm Control

The group of operations for performing arithmetic.

Floating Point Control

The manipulation of the operands prior to, and the result subsequent to, the arithmetic operations.

Memory Cycle Control

The transfer of information to and/or from core storage.

1.3 Control Registers

The computer is functionally divided into the first four groups of operations mentioned before. Only one control register of the four is active at a time. The last two, memory and input-output control, operate asynchronously and can be active simultaneously with the first four.

Each of the four groups of operations has a control register, consisting of the required number of flip-flops, to enable establishing the control for the activity within each operation. The control registers are identified as:

PI	Program Control Register
I I	Instruction Control Register
AI	Algorithm Control Register
FI	Floating Point Control Register

Each setting of a control register controls the execution of a specific variety of that species of operation.

Control Register Functions

PI

PI = 0	do next instruction
PI = 1	transfer next instruction word to PR
PI = 2	modify index register
PI = 3	count down repeat counter

AI

AI = 0	force add in multiplication (multiplier is minus one)
AI = 1	add or subtract
AI = 2	double-length multiplication

AI = 3	single-length multiplication, rounded
AI = 4	fixed point division, first cycle
AI = 5	fixed point division, other than first cycle, or floating point division
AI = 6	Shift
AI = 7	Q Jump

FI

FI = 0	arrange first cycle (beginning of add or subtract)
FI = 1	exponent addition or subtraction (beginning of multiply or divide)
FI = 2	shift D (arrangement of D since $D_E < A_E$)
FI = 3	shift A (arrangement of A since $D_E > A_E$)
FI = 4	normalize (following arithmetic operation)
FI = 5	correction (before algorithm if division, following if add, subtract or multiply)
FI = 6	Clear D ($D_E \ll A_E$)
FI = 7	clear A ($A_E \ll D_E$)

I I

I I = 0	control state used for most instructions
I I = 1	multiply cycle of multiply then add or subtract instructions

MI Memory Control

MI = 0	memory not being used by computer	
MI = 1	(M) $\longrightarrow$ PR $\longrightarrow$ M	(read and write PR)
MI = 2	(M) $\longrightarrow$ D	(read D)
MI = 3	(D) $\longrightarrow$ M	(write D)
MI = 4	(M) $\longrightarrow$ D $\longrightarrow$ M	(read and write D)
MI = 5	clear M, (D) $\longrightarrow$ M	(read 0 write D)
MI = 6	clear M, leave cleared	(read and write 0)
MI = 7	clear M	(read 0)

Note: Due to the nature of magnetic core use, the read-out from memory (sensing of information in the cores) is destructive. When the information must be maintained it is written back following the read, as in MI = 1 above.

The required sequence of operations is effected by having the instruction select the proper sequence of the control registers. Program control activity to select the next instruction to be performed, the PI operation, comes first. It is followed by the I I operation.

At the start of the I I operation, the next instruction is decoded and the sequence of operations arranged. Many instructions are completely performed in the I I operation. Arithmetic requires AI and possibly FI operations. The sequencing was predetermined by the logical design of the computer. The decoding of the command selects the control lines.

The various phases, or sequences within the operations are called "timings". The timings are the sequencing control for groups of tasks within the operation. These are consecutively numbered.

PT1 through PT4
IT1 through IT8
AT1 through AT4
FT1 through FT4

The progression of control through the timings for an operation is a function of the operation being performed. Some instructions require activity in all timings of a control, others do not. However, many similar sequences can be established. Generally, an operation is a sequence of four timings or a multiple thereof.

This can be illustrated by the general case. The purpose of the operation is to obtain new information (data or control) that is the resultant of the combination, or effect, of existing information. The operation can be symbolized as:

$$(X) \text{ and } (Y) \longrightarrow Z$$

where registers X and Y are the inputs to the processing network and Z is the register receiving the output of the network. This could represent an addition, for example. Usually the storage of the result in Z is temporary. It is transferred to another register for further use (in this case, back to X).

In the S-2000, information transfers are done in two steps. The receiving register is first cleared so that all the bits (binary digits) have the same value. The register may be cleared to all zeros or all ones dependent upon the operation. Then all bits having the opposite binary value are transferred to the register. By initially clearing the register to a known condition, the following transfer of information is a direct, simple process involving a minimum of logic.

The required tasks of this illustration would be as follows (where tasks listed horizontally can be simultaneously performed):

	clear X	clear Y	
1.	$\longrightarrow$ X	$\longrightarrow$ Y	clear Z
2.	$[(X) \text{ and } (Y)] \longrightarrow Z$		
3.	clear X		
4.	$(Z) \longrightarrow X$		

Usually one of the existing pieces of information has been transferred to X in some prior operation and Y is automatically cleared by program control at, say, the beginning of an instruction. This reduces the process to a four-step sequence, as numbered above, and is a reason for a module of four timings being used in the computer.

The timings are performed by using a group of flip-flops, one for each timing. For each control register, the timing flip-flops are interrelated to form a timing chain so that each is active only in its turn. The timing flip-flop outputs activate the gates to perform the selected tasks in proper sequence.

The performance of a simple, fixed point add instruction, $(A) + (V) \longrightarrow A$, can illustrate the sequence of control registers and their timings. The instruction is in the Program Register (PR) and has been selected as we start by entering the II operation at IT1.

In IT1, the following occurs (simultaneously):

1. A general clearing of controls preparatory to each instruction.

2. Prepare for transfer of the address of the operand in memory from PR, where the current instruction is stored, to the Memory Address Register (MA) where the address will be stored for decoding to select the desired memory location.

3. Clear MA

4. Clear the D register as it will receive the operand transferred from memory.

5. Go to IT2.

In IT2:

1. Transfer the address from PR to MA.

2. Set the algorithm control register to the state required for add, $1 \longrightarrow A1$. ($AI = 1$ indicates the operation is add or subtract). This is a preparatory

task, solely. AI will not take control until its timings have been initiated.

3. Set the memory control register to the state required for the transfer of the operand, 4 $\longrightarrow$ MI. The destructive nature of the memory read-out requires the information to be rewritten, following the transfer to D. The transfer to D will start at this time.

4. IT2 will remain on until the transfer, (V) $\longrightarrow$ D is completed, at which time the computer goes to IT3.

5. The memory timings continue, in parallel with the computer timings, to transfer (D) $\longrightarrow$ V.

In IT3:

1. Clear the D* register which is the input to the adder for one operand. (D and D* are separate one-word registers).

2. Go to IT4.

In IT4:

1. Transfer (D) $\longrightarrow$ D*

2. Go to AT1.

In AT1:

1. Clear the A* register which will receive the output of the adder.

2. A and D* registers are always connected to the adder inputs. The addition occurs during this timing.

3. Upon completion of the addition go to AT2.

In AT2:

1. Store knowledge of overflow if it occurred.

2. Connect the adder output to A* (in effect transfer the sum to A*).

3. Go to AT3.

In AT3:

1. Clear A.

2. Go to AT4.

In AT4:

1. Transfer the sum, (A*) $\longrightarrow$ A

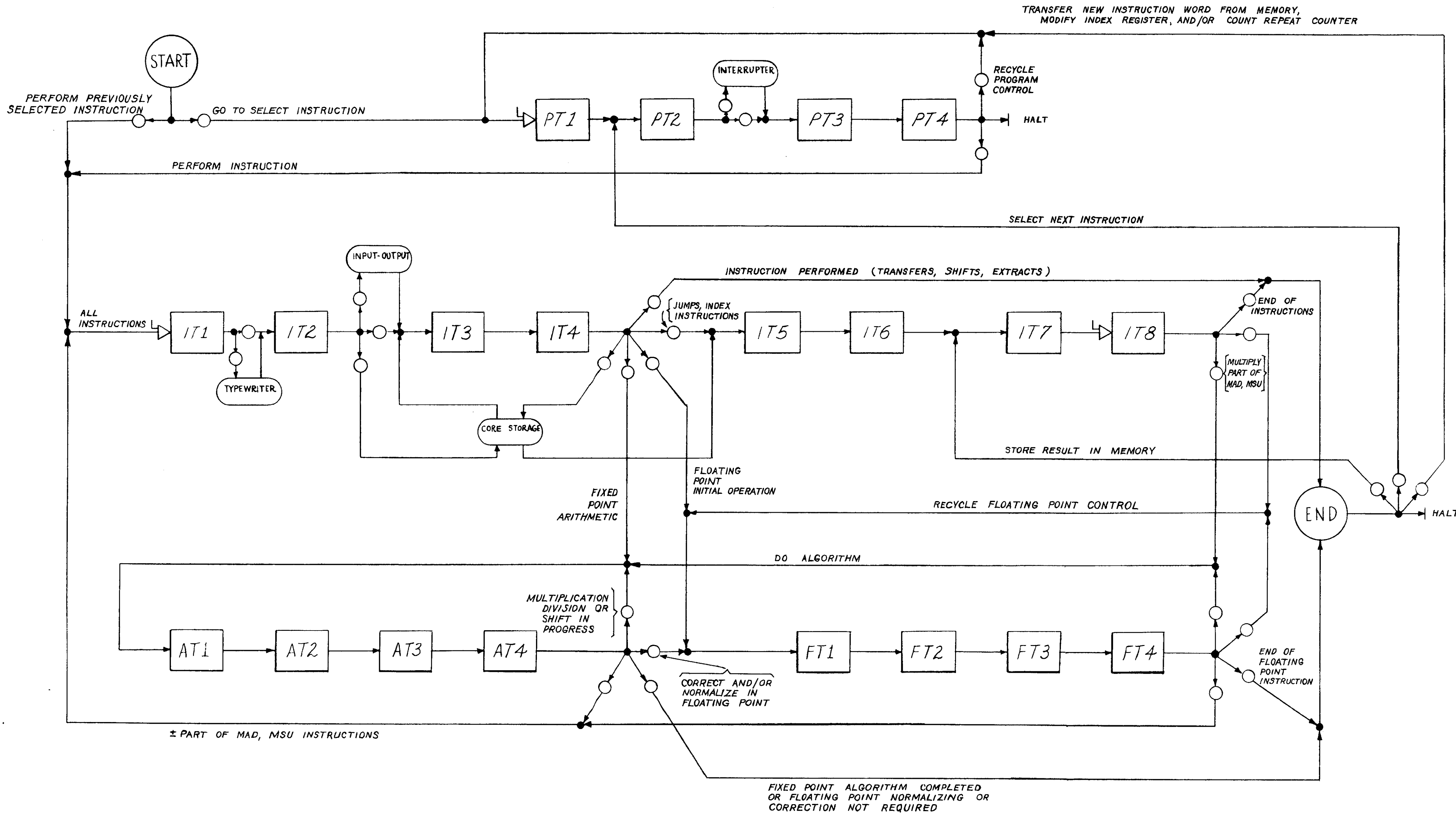
2. Go to "End". The instruction is completed and a program control decision is necessary to determine what follows this instruction.

Other instructions will go through a difference sequence of controls and timings due to the differences in operations. Floating point addition would require activity performed during FT's under FI control, in addition to the above described operations for fixed point addition. Transfer instructions are completely performed under II control from IT1 to IT4. It will be noted that during the add instruction the transfer of the operand from memory to D was done during an IT. A jump instruction would also be performed under II control, IT1 through IT8.

Figure 1.3-1 surveys the flow of control for the instructions.

Program Activity for Repeat and Index Register

An interesting use of control is that done with certain operations relating to index register modification (counting or repeat modification of the index register) or relating to the repeat mode control. These operations will be the last ones to be performed in an instruction and will therefore be followed by program control activity.



TRANSFER NEW INSTRUCTION WORD FROM MEMORY, MODIFY INDEX REGISTER, AND/OR COUNT REPEAT COUNTER
START
PERFORM PREVIOUSLY SELECTED INSTRUCTION
GO TO SELECT INSTRUCTION
INTERRUPTER
PT1
PT2
PT3
PT4
RECYCLE PROGRAM CONTROL
HALT
PERFORM INSTRUCTION
SELECT NEXT INSTRUCTION
INPUT-OUTPUT
INSTRUCTION PERFORMED (TRANSFERS, SHIFTS, EXTRACTS)
ALL INSTRUCTIONS
IT1
IT2
IT3
IT4
JUMPS, INDEX INSTRUCTIONS
IT5
IT6
IT7
IT8
END OF INSTRUCTIONS
TYPEWRITER
MULTIPLY PART OF MAD, MSU
CORE STORAGE
STORE RESULT IN MEMORY
FLOATING POINT INITIAL OPERATION
FIXED POINT ARITHMETIC
RECYCLE FLOATING POINT CONTROL
END
HALT
DO ALGORITHM
MULTIPLICATION DIVISION OR SHIFT IN PROGRESS
AT1
AT2
AT3
AT4
FT1
FT2
FT3
FT4
END OF FLOATING POINT INSTRUCTION
CORRECT AND/OR NORMALIZE IN FLOATING POINT
± PART OF MAD, MSU INSTRUCTIONS
FIXED POINT ALGORITHM COMPLETED OR FLOATING POINT NORMALIZING OR CORRECTION NOT REQUIRED

Figure 1.3-1 Flow Diagram of Computer Timing

In some cases the result of these operations will determine what type of program control action shall follow. For example, the result of one of these operations will determine whether or not the final performance of a repeated instruction was just done. If not, it is performed again. If its final performance was done, repeat is terminated and the succeeding instruction word must be obtained.

The nature of these operations lend themselves to performance under the program control. This also falls in with the general scheme of starting the basic program cycle with PI operations. At the end of the instruction performance, control is returned to PI. If further operations are required, they will be performed under control of PI, during the timings of PT. Following this the PT's may be reinitiated, and this time sequence will be for the program control of the next instruction.

Double Rank Control Registers

As PI may possibly be set to a different state during a set of PT's to control the next set of PT's immediately following, it is necessary to make PI a double rank register (PI and PI*). PI controls during PT1 and PT2, PI* controls during PT3 and PT4. PI can be changed to its new setting while PI* controls. The setting of PI is transferred to PI* in PT1.

Every control register with a recycling control in the sense that one set of its timings may immediately succeed another of its sets is double ranked. These registers are AI, FI and PI.

The characteristics of the activities during a timing should be noted. All are done in parallel. Most are accomplished simultaneously. The completion of addition is an example of one task that may take a longer time. The time required to complete an addition is variable and a function of the data. The computer does not allot a fixed time interval for a timing but proceeds from one timing to the next as soon as all tasks of a timing that are necessary for the next timing have been completed. Where the time required for a task is variable, it waits for a signal to indicate completion. If all task completion times of a timing never vary, the computer proceeds as rapidly as possible (the limiting factor is the switching time of the circuits) from one timing to the next. In this manner, the computer is operating as fast as data and circuitry permit, for each operation, rather than at a constant rate for all operations determined by the slowest possible variable case.

Timing Chain

The requirements for the organization of the timing chain are fourfold:

1. The proper sequence should be established.

2. No more than one timing should be active in any interval.

3. If the computer has been operating in parallel with memory it must wait, if necessary, before activating a timing requiring the memory contents.

4. A timing which acts upon a register that has been subject to a change of contents should not become active until the situation has stabilized.

To achieve these purposes requires a system of interlocks. A timing is active only under two types of conditions. Its associated timing flip-flop is set to one and several additional signals are permissive. These signals comprise the interlocks.

For example, one of the requirements for any timing to be active is the existence of the zero state of the previously active timing flip-flop (See Figure 1.3-2).

The proper sequence is assured by not turning a timing flip-flop off until the next timing flip-flop has turned on (set to one). Thus the set-to-one signal will continue until the flip-flop is set.

The third and fourth requirements are met by the third input of the three-input AND gate that activates the timing. This third input has been named the "trigger". Table 1.3-1 shows which timings require a trigger. MO, the PT1 trigger, signifies the completion of the memory operation of the previous instruction. MO $\equiv$ MI = 000; the memory control register has been reset to zero (or no memory operation was started at the end of the previous instruction). AN2CC (AN2 Carry Complete) indicates the stabilization of that adder network following a change of contents of a register that is an input to AN2. The D numbers that are used as triggers also incorporate the carry complete signal and note the exceptions, the instructions that need not

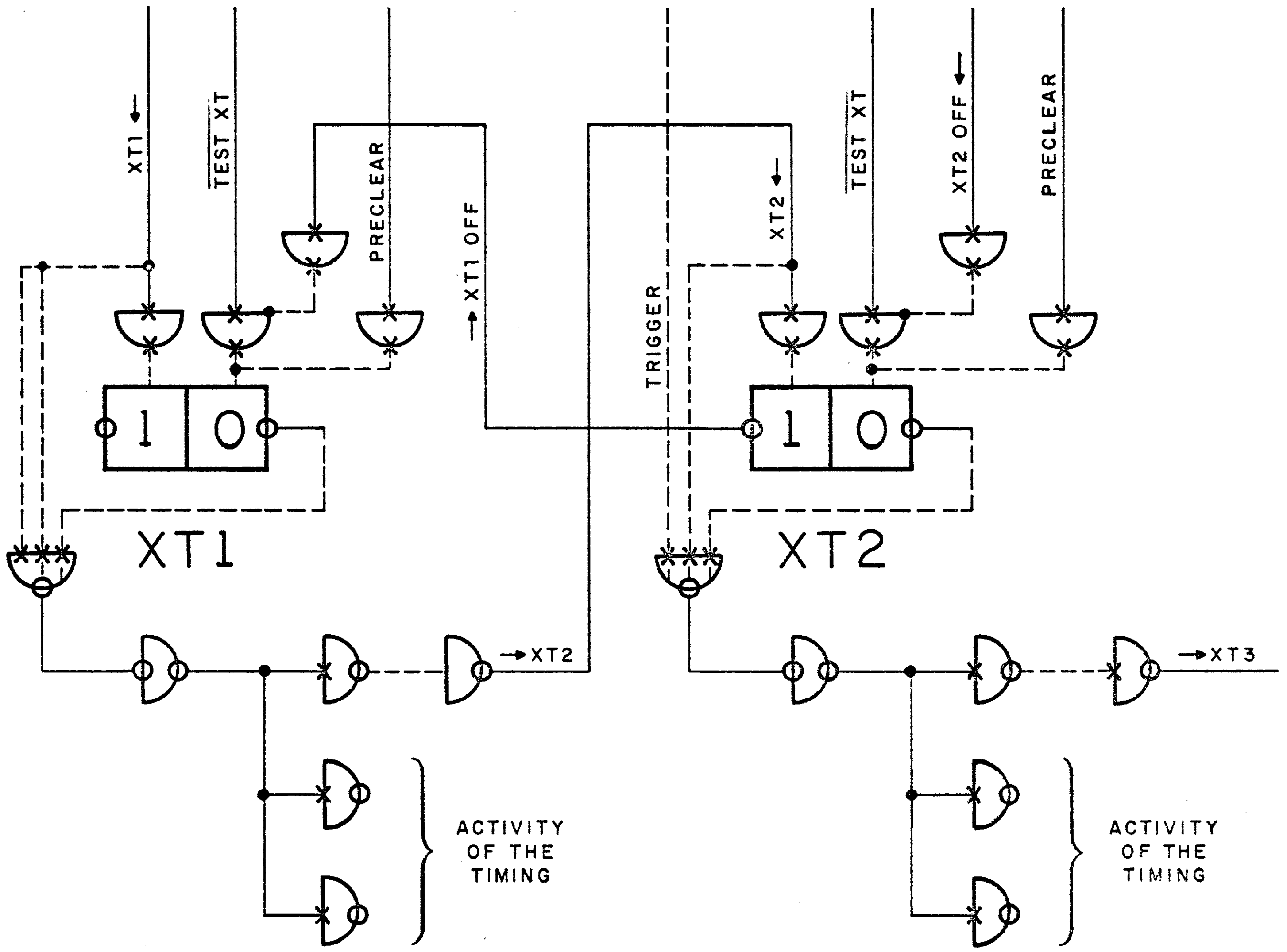


Figure 1.3-2 Timing Chain

await the carry complete signal. D67 is AN2CC or a jump instruction. D103 is AN1CC or instructions that need not await it. D117 is AN2CC or certain index register instructions. AN1ECC is the carry complete signal for the exponent part of AN1 in floating point number usage.

Referring to Figure 1.3-2, the "$\overline{\text{TEST XT}}$" signal is usually negative and is changed only for engineering tests of the computer. "Preclear" is part of a manual console control to restore program control to its starting point.

In Table 1.3-1, the AT4 off signal seems to present the paradox of the flip-flop attempting to set and reset simultaneously. This is not the case as the Off signal originates after the passage of the AT4 signal through several transistors with the attendant circuit delay. IT4, IT8, AT4, FT4, and PT4 are the possible end of the timing chain for their respective control registers. It is for this reason they are turned off in the manner indicated by Table 1.3-1.

Table 1.3-1

Timing	Trigger (if any)	Signal to Turn Timing Off
PT1	MO	PT2 = 1
PT2	AN2CC	PT3 = 1
PT3		PT4 = 1
PT4	AN2CC	PT4 · MO
IT1		IT2 = 1
IT2	D67	IT3 = 1
IT3		IT4 = 1
IT4	AN2CC	"IT4"
IT5	D103	IT6 = 1
IT6		IT7 = 1
IT7	D103	IT8 = 1
IT8	D117	"IT8"
AT1		AT2 = 1
AT2	D48	AT3 = 1
AT3		AT4 = 1
AT4		"AT4"
FT1	D49	FT2 = 1
FT2	AN1ECC	FT3 = 1
FT3		FT4 = 1
FT4		$\longrightarrow$ END

1.4 Commands

The rightmost eight bits of the instruction are assigned to the coding of the command part of the instruction. They are identified as:

	2_{-16}	2_{-17}	2_{-18}	2_{-19}	2_{-20}	2_{-21}	2_{-22}	2_{-23}
Bit Position								
	2_{-40}	2_{-41}	2_{-42}	2_{-43}	2_{-44}	2_{-45}	2_{-46}	2_{-47}
Identification	J	C_6	C_5	C_4	C_3	C_2	C_1	C_0

The scheme for command coding follows that of the computer control organization. Since many operations are the same in instructions, as far as is feasible specific meanings are given to each bit of the command coding. A binary bit will tend to specify whether or not a possible operation shall be performed. Two bits may be grouped to select one of four possible variations in an operation. This scheme cannot be used for every command coding due to the differences among the wide variety of TRANSAC instructions. However it is used extensively.

For example, C_6 is used to indicate whether or not an arithmetic instruction is involved:

$C_6 = 1$	arithmetic instruction
$C_6 = 0$	non-arithmetic instruction

The coding for arithmetic instructions follows. "X" is used to indicate that the value of the bit position is not significant. This is termed "bar coding". The value of bar coding is that it serves to indicate how to decode the command into individual control signal lines.

J	C_6	C_5	C_4	C_3	C_2	C_1	C_0	
X	1	X	X	X	X	X	X	Arithmetic with C_5 and C_4 indicating type of arithmetic operation.
X	1	0	0	X	X	X	X	Add
X	1	0	1	X	X	X	X	Subtract
X	1	1	0	X	X	X	X	Multiply
X	1	1	1	X	X	X	X	Divide (also some special instructions)

								C_3 indicates location of second operand (location of other operand not variable)	
X	1	X	0	0	X	X	X	operand in memory	
X	1	X	0	1	X	X	X	operand in register	
X	1	X	X	X	0	X	X	Use algebraic value of second operand	
X	1	X	X	X	1	X	X	Use absolute value of second operand (in add, subtract or multiply).	
X	1	0	X	X	X	0	X	Do not preclear A	
X	1	0	X	X	X	1	X	Preclear A	
X	1	1	X	X	X	0	X	Double length product or dividend	
X	1	1	X	X	X	1	X	Result rounded (single length) (except special arithmetic)	
X	1	X	X	X	X	X	0	Do not store result in M	) except) special
X	1	X	X	X	X	X	1	Store result in M	) arithmetic
0	1	X	X	X	X	X	X	Fixed point arithmetic	
1	1	X	X	X	X	X	X	Floating point arithmetic	

Shift Instructions 1 001 XXXX

C_3 C_2	code the register involved:
0 0	A and Q registers
0 1	A register
1 0	Q register
1 1	D register

C_1 codes the type of information in the word that is to be shifted.

$C_1 = 1$	means treat the leftmost bit (2_0) as the sign bit and the remainder as an admissible number. Shifting right is equivalent to dividing this number by 2; shifting left is equivalent to multiplying by 2.
$C_1 = 0$	means treat the word as some quantity. No sign is involved and all bits are treated alike.

C_0 codes the direction of the shift.

$C_0 = 0$	means shift left.
$C_0 = 1$	means shift right. (The D register can only shift right. $C_0 = 0$ causes a circular right shift in D.)

Jump Instructions X 010 XXXX

C_3 C_2 broadly defines the areas involved or types of jump instructions.

00XX defines a special group of jumps. (C_3 through C_0 incl.)

0000	unconditional jump
0001	jump if (A) = 0
0010	jump if OVF = 0 (overflow flip-flop is zero)
0011	jump if OVF = 1

01XX defines the A jumps

0100	jump if A is positive ($A \geqq 0$)
0101	jump if A is negative ($A < 0$)
0110	jump if (A) = (Q)
0111	jump if (A) = (D)

10XX defines the Q jumps

1000	jump if Q is positive (Q is also circular shifted left one place whether or not the control jumps.)

1001	jump if Q is negative. (Q is circular shifted left one place, as well)
1010	jump if Q is even ($2_{-47} = 0$) (Q is also circularly shifted right one place whether or not the control jumps).
1011	jump if Q is odd (Q is circular shifted right, one place, as well

11XX   refers to D and magnitude jumps

1100	jump if D is positive
1101	jump if (A) $\geqq$ (Q) in floating point sense
1110	jump if (A) $\geqq$ (Q) in the algebraic sense
1111	jump if (A) $\geqq$ (D) in the alphanumeric sense

In addition there is a group of six miscellaneous or special arithmetic instructions. The groups can be identified by the bar coding: X 111 1XXX.

00X	Multiply and add or subtract
01X	Logical product and add or subtract
10X	Add or subtract D

where $C_o = 0$ means add
$C_o = 1$ means subract

Non-Arithmetic Instructions

The non-arithmetic instructions are divided into five groups by the use of C_4 C_5 and J.

X	000	XXXX	special instructions
0	001	XXXX	transfers of data
1	001	XXXX	shifts
X	010	XXXX	jumps
X	011	XXXX	index register instructions

Transfers of Data   0 001   XXXX

Transfers of data are among four possible locations, indicating a two-bit coding to indentify the location.

00	location is memory
01	location is A register
10	location is Q register
11	location is D register

C_3 C_2 identify the sending location; C_1 C_0 identify the receiving location. If C_3 C_2 is the same as C_1 C_0 the register is cleared to zero. Otherwise the sending location is unaltered by the transfer.

The varieties of index register and special instructions are not suited for description by means of bar coding.

Command Decoding

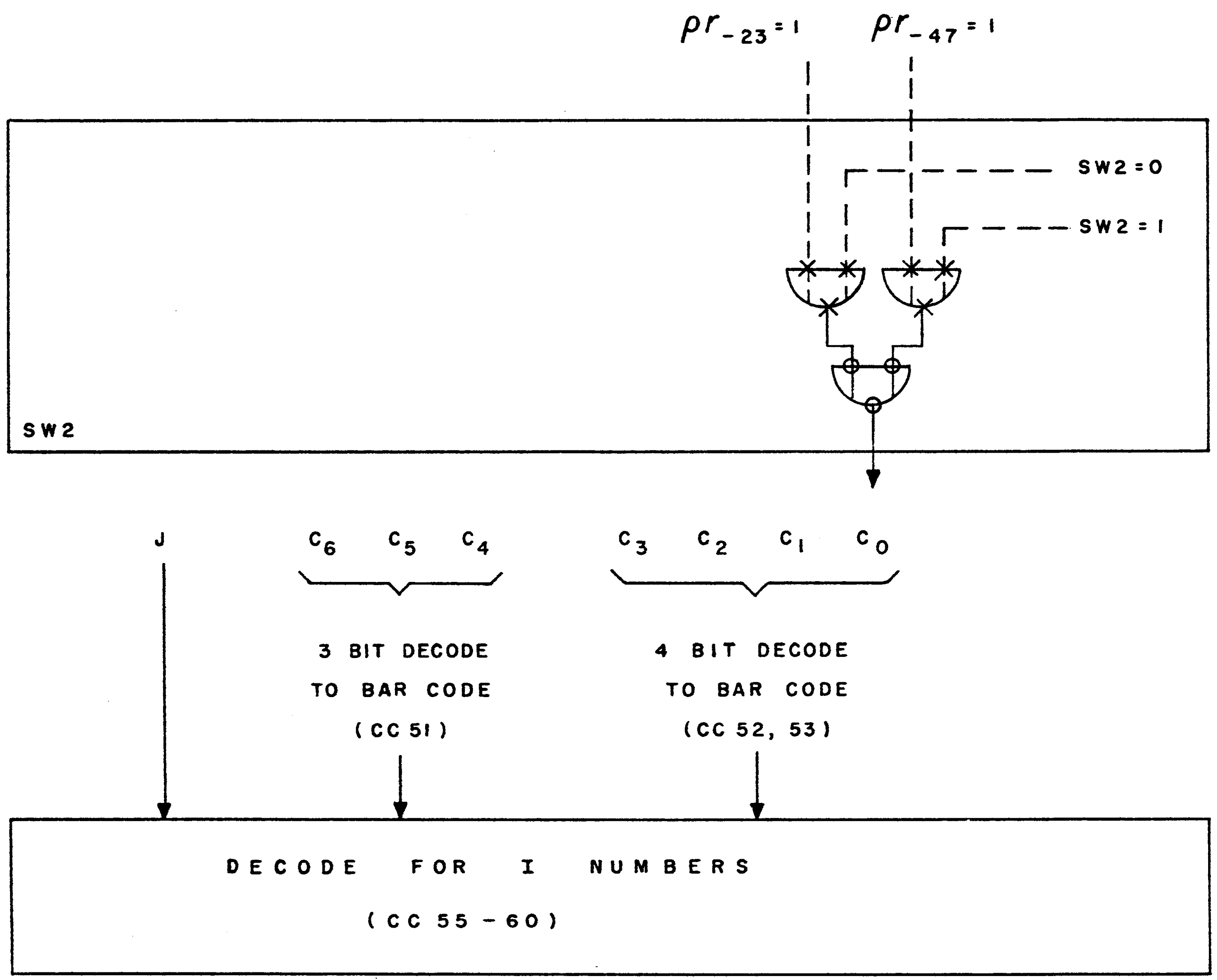
When the command bits of the instruction are decoded the bar codes are used, to a large extent, to determine the meaning of the decode outputs. The outputs are control signal lines and are called "I" numbers. Each is identified by the letter "I" followed by a number. The following is representative of the types of I numbers.

I1	010	0001	jump if A = 0 (a specific instruction)
I15	010		jump instructions (a type)
I39	1XX	0XX0	arithmetic instruction with (V) as one operand and result not stored (one variety of a type of instruction).

Decoding is done by sub-dividing the 8-bit command code into three groups as shown in Figure 1.4-1. Generally, the 3-bit and 4-bit decode results in bar codes which are combined to produce the "I number" control signal lines.

Translations

As an aid in translating between the machine command coding and English or the mnemonic code, the 8-bit code is converted to a quaternary type of code (quaternary number system having the numbers, 0 through 3). Reading from left to right, each pair of binary digits is converted to one quaternary number. Tables give the instruction and mnemonic code for each 4-digit quaternary number thus obtained. (See Appendix).



$pr_{-23}=1$
$pr_{-47}=1$
SW2=0
SW2=1
SW2
J
C_6 C_5 C_4
C_3 C_2 C_1 C_0
3 BIT DECODE TO BAR CODE (CC 51)
4 BIT DECODE TO BAR CODE (CC 52, 53)
DECODE FOR I NUMBERS (CC 55 - 60)

Figure 1.4-1 Command Decoding

1.5 Numbers - Control Signals

The problem of naming the hundreds of control signals in the computer is an appreciable one. The command part of the instruction, for example, is decoded to derive 138 control signals as the resultants of many combinations of the eight bit coding. If these signals were given descriptive names, the language on the schematic would tend to obscure the circuitry.

Abbreviations are used, instead. The signals are classified functionally. Each function is assigned a representative letter and the signals within the group are numbered consecutively. These abbreviated names are called "numbers".

The groups are decribed below. The numbers are listed in the appendix.

A Numbers	-	AI amd AI* signals. These will define the various choices of activity during algorithm operations. Other signals used to derive A numbers are some FI settings, SC = Sat., SC ≠ Sat. (Shift Counter not Satisfied).
F Numbers	-	Control signals for floating point operations derived from FI settings, the instructions and the data words being manipulated.
I Numbers	-	Signals derived from the command and occasionally data. Some I numbers represent the common tasks of several instructions to control their performance.
J Numbers	-	Jump activities derived from the command coding, the data words and their comparison.
M and DM Numbers	-	Signals derived from MI settings and other memory control signals.
P Numbers	-	Program Control signals from PI, PI* settings, RR, and other operating controls.
R Numbers	-	Repeat mode control signals.

S Numbers	- Skip control signals.
V Numbers	- Variables is the name given to activities dependent upon the data word, such as the sign of the word, overflow; or the settings of other program controls which vary periodically.
D Numbers	- When combinations of the above numbers are OR gated together, the resultant is called a "decision" and given a D Number. The D Number control signal will cause the specific tasks to be executed.

In effect then, the size and complexity of the computer requires three levels of control signals to instigate activities. There are activities done on the basis of the instruction, those that require several additional conditions to be done and those requiring many conditions to be performed.

2. INTRODUCTION TO TRANSAC LOGIC

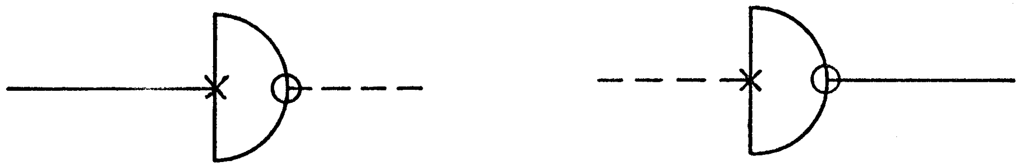
2.1 S-2000 Symbolism, Logical Circuits

A description of the symbolism used in the notation of the S-2000 logic and standard logical elements is attached as an Appendix for convenient reference. A familiarity with this material is assumed for the remainder of this section.

S-2000 Logical Circuits

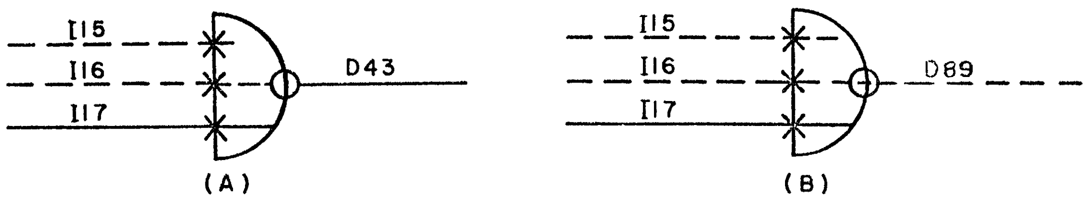
The basic logic circuits are formed by use of the standard transistor configurations.

Inversion (negation) is achieved with the common emitter (usually followed by an emitter follower for design reasons). See Figure 2.1-1.



Inverter
Figure 2.1-1

AND and OR circuits are formed by the use of either series (high) gates or parallel (wide) gates. Either gate can serve either the AND or the OR function.



Parallel RC Gate
Figure 2.1-2

The 3-wide gate of Figure 2.1-2 can be used as an AND gate (A) : $I15 \cdot I16 \cdot \overline{I17} \Rightarrow D43$ (The overscore denotes the inverse value, spoken of as NOT I17)

The gate can also be used for an OR function (B):

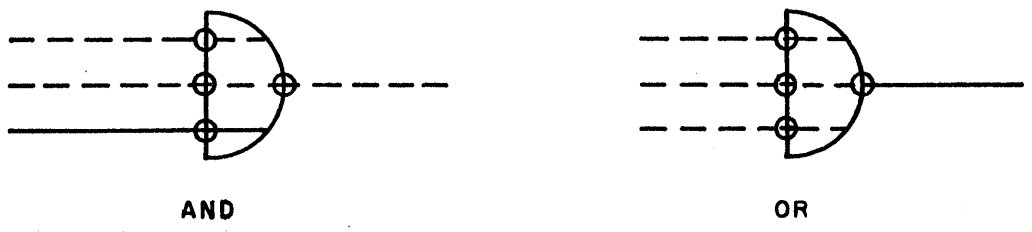
$$\overline{I15} \ \vee \ \overline{I16} \ \vee \ I17 \ \Rightarrow \ D89$$

(In our usage the term, OR, implies inclusive OR unless qualified.)

The logical function of the parallel gate can usually be determined by the polarity of the active output as shown in the schematic logic adjacent to the output signal name. An active negative polarity (solid line) for the parallel gate output indicates the AND function. All input lines must be positive to result in a negative output.

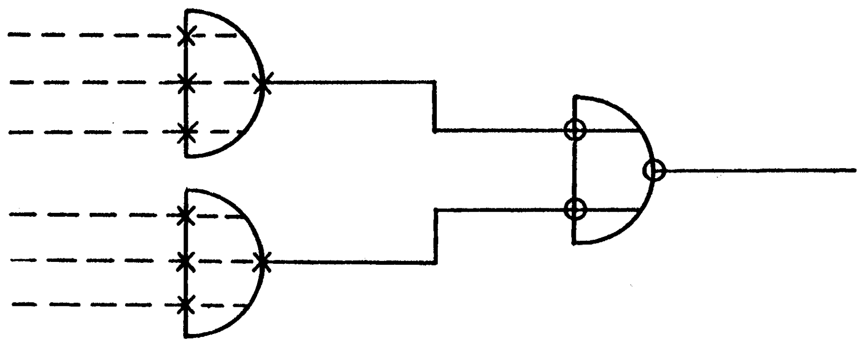
A positive active output polarity (dashed line) of the parallel gate indicates the OR function. Any combination of negative inputs will produce a positive output.

Figure 2.1-3 shows a parallel emitter follower configuration.



Parallel Emitter Follower Gate
Figure 2.1-3

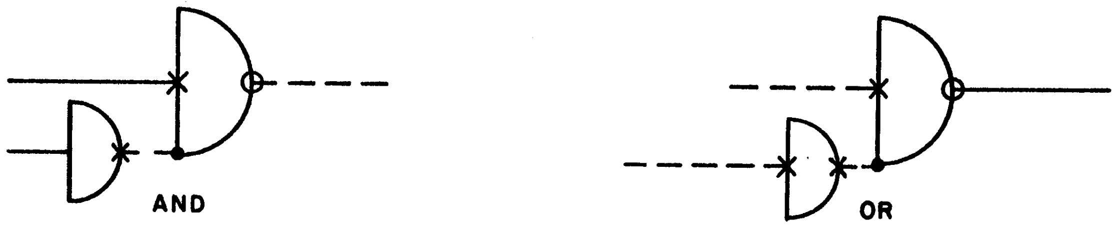
This configuration is customarily used for the OR function as illustrated in Figure 2.1-4.



Parallel Emitter Follower OR Gate
Figure 2.1-4

For the OR function the active output polarity is negative. The parallel emitter follower is occasionally used as an AND gate, active output polarity positive.

The series (high) gate can also serve either logical function. See Figure 2.1-5.



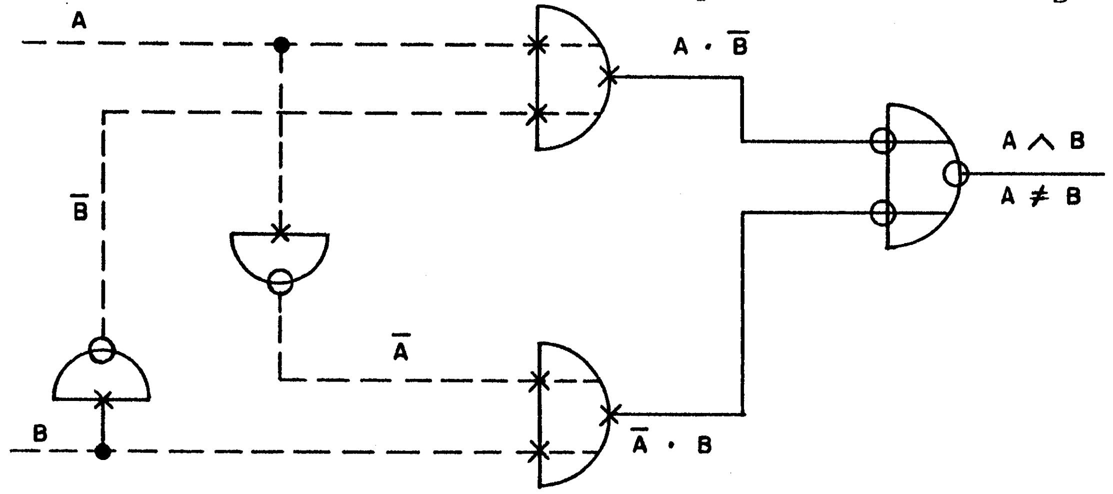
Series Gate
Figure 2.1-5

The series gate usually serves as an AND gate.

The Exclusive OR logical circuit may be better visualized by redefining Exclusive OR. It signifies inequality between two binary quantities:

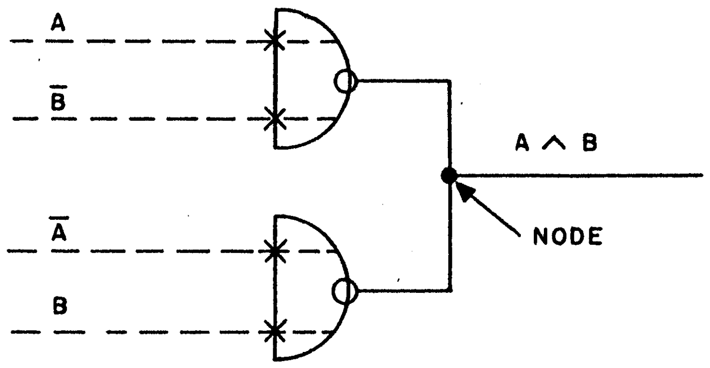
$$A \wedge B \equiv A \neq B$$

The logical circuit is devised upon this basis, See Figure 2.1-6.



Exclusive OR
Figure 2.1-6

The right half of Figure 2.1-6 may also be drawn as in Figure 2.1-7. The junction of the emitter follower bases is sometimes called a "node".



Exclusive OR
Figure 2.1-7

The customary use of the Exclusive OR is as part of a network for the comparison of two binary quantities. Figure 2.1-8 can be used to determine inequality or equality.

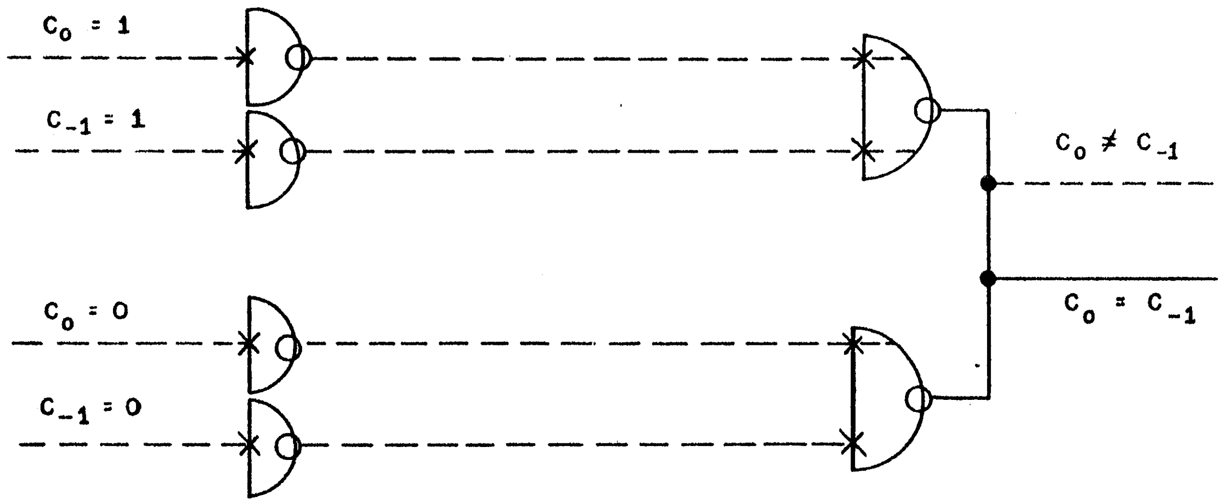


Figure 2.1-8

$$C_0 \neq C_{-1} \equiv (C_0 = 0 \cdot C_{-1} = 1) \vee (C_0 = 1 \cdot C_{-1} = 0)$$

The v symbol is customarily used, rather than $\wedge$, when the two terms are mutually exclusive, as the "V" in this example:

$$C_o = C_{-1} \equiv (C_o = 0 \cdot C_{-1} = 0) \vee (C_o = 1 \cdot C_{-1} = 1)$$

Both outputs are not necessarily derived from the same logical circuit. Figure 2.1-8 is an illustration of a method.

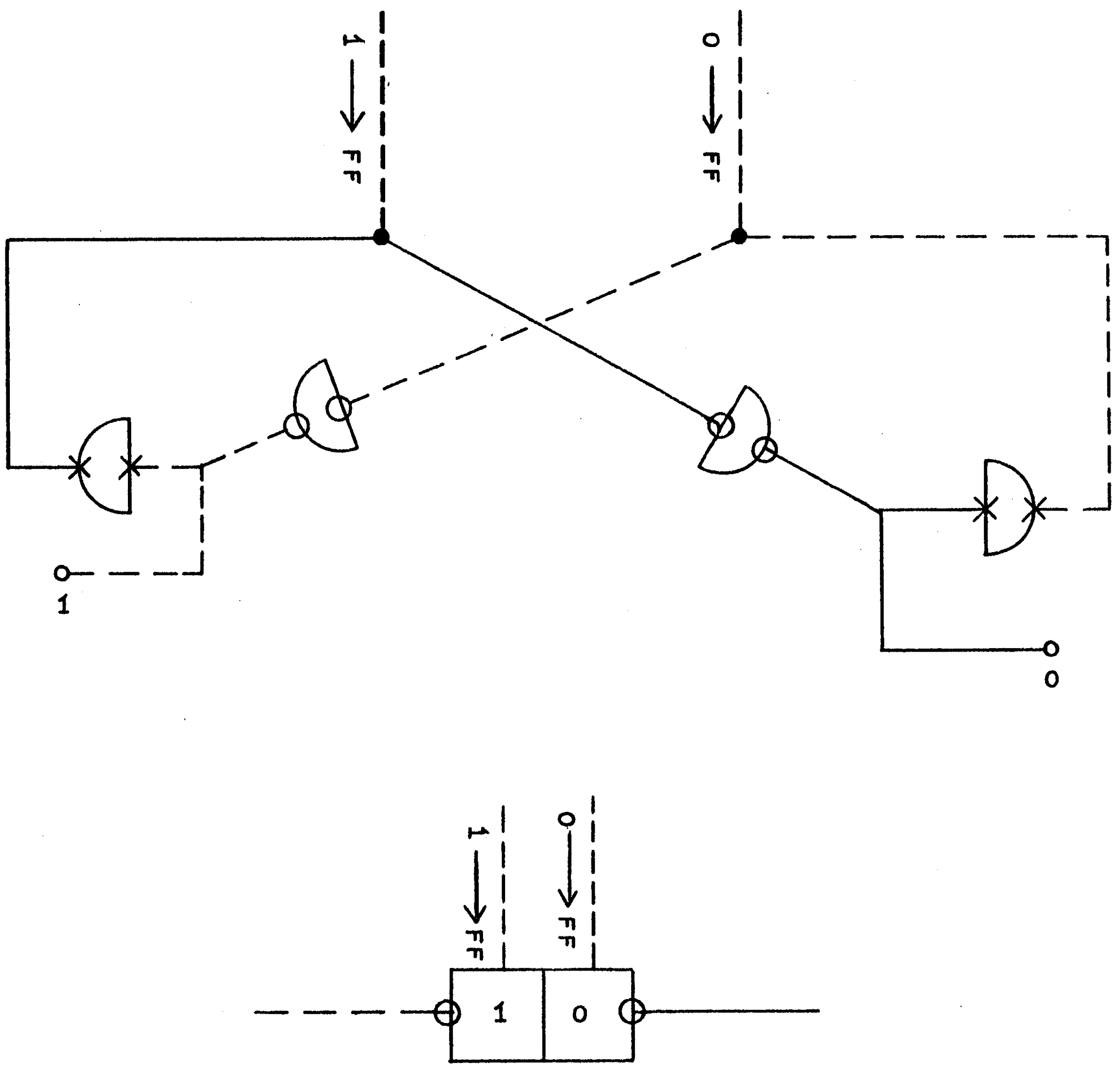
The standard RC Flip-flop is shown in Figure 2.1-9, logical circuit and symbol. The logical circuit is drawn here to emphasize the similarity to the crossover network of vacuum tube flip-flops.

The active input polarity is positive. The desired polarity of an active output line will determine from which of the two output sides the output signal is taken.

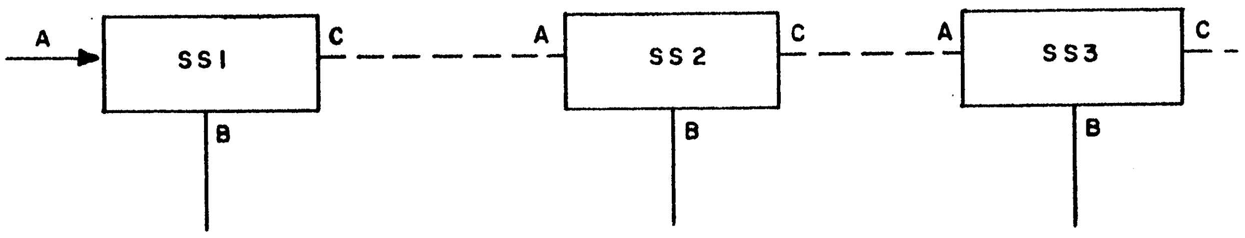
The Single Shot logical circuit is shown in Figure 2.1-10 The single shot can be triggered (changed to its unstable state) only by a negative-going signal at line A. The polarities shown for lines B and C indicate the polarities during the unstable state. The schematic logic shows a single shot timing chain in this manner.

The single shot is put to two uses, activity during its unstable state (usually the B line) and activity at the end of the unstable state (usually the C line). The A line determines when B starts to be active. The active period of B (unstable state) is of course determined by the circuit design of the single shot. C can trigger the next of a chain of single shots at the end of the unstable period. While the single shot is in its unstable state, the A line has no influence upon it. And the input that goes negative and remains negative can only trigger the single shot once. The A line must return to positive to be capable of retriggering the single shot.

The "C" output line of a single shot is normally negative. When connected as in Figure 2.1-10, the three single shots can and will operate only in sequence, each being activated by the deactivation of its predecessor. The length of the active period of the single shot is determined by the time constant of an RC network.



Standard Flip-Flop
Figure 2.1-9



Single Shot Timing Chain

Figure 2.1-10

2.2 Notation for Schematic Logic

This section describes the notation used in the schematic logic drawings of the S-2000 and this manual. The abbreviations, or symbols, used to name the registers and control devices are listed in the Glossary of Terms. An explanation of the notation method for signals follows here:

(A)	The parenthesis is used to indicate "the contents of" a storage device; in this case the contents of the A register. This distinguishes between the storage device itself, A, and its contents, (A).
D_1	The upper case letter is used to refer to the entire register or a major portion thereof; in this case the subscript number "1" indicates the right hand half of D.
d_1	The lower case letter indicates one bit of a data register; the subscript indicates which bit position. The positions are numbered from left to right, 0 to -47, corresponding to the negative exponent values for the binary fraction.
$AN2^0$	In the control information registers the binary numbers stored are integers such as a memory address, number of times an instruction is to be repeated, or number of places the word is to be shifted. The bit positions of the control information registers (and their adder, AN2) are therefore numbered right to left with a superscript corresponding to the positive exponent of the binary integer. The least significant bit position of AN2 is named $AN2^0$

as compared to the naming of the least significant bit of the A register, a_{-47}. (It was apparently considered confusing to apply the lower case letter notation to a mixture of alphabetic and numeric characters — $an2^{o}$ as against $AN2^{o}$.

$(SW2)'$	The prime symbol indicates the inverse, or for a binary quantity the ones complement. In this case, the ones complement of the contents of SW2.
$I67 \cdot V32$	Logical AND symbol; when I67 and V32 are both active.
$I67 \vee V32$	Logical inclusive OR symbol; either I67 OR V32 OR both are active.
$I67 \wedge V32$	Logical exclusive OR symbol, one OR the other but not both are active.
Control Signal Names	Two types of names are used. One type is an arbitrary or a mnemonic name such as I67, V32. The other type is a functional name, such as $1 \longrightarrow OVF$ (set the Overflow Flip-flop to its one state).
$\longrightarrow$	The action specified at the tail of the arrow is to be performed upon the device indicated by the head of the arrow. $0 \longrightarrow MA$, Clear the Memory Address register to zeros. $(V) \longrightarrow PR$, transfer the contents of the selected word in memory to the Program Register.

$\Longrightarrow$ Logical equations. When the signals shown at the arrow tail are active, the action listed at the arrowhead is to be performed.

$$D81 \cdot \overline{D79} \cdot \overline{D52} \Longrightarrow 2 \longrightarrow PI$$

When D81 and NOT D79 and NOT D52 are active, set the Program Control register to its "2" state. NOT D79 active is identical to D79 normal, logically.

$\longrightarrow$ AT1	Sometime the action can only be one thing and may be therefore implied. "Go to AT1 timing" is a signal named as shown. The circumstances under which the signal is active is shown by the schematic logic drawing for the signal's origin.
$(A) \xrightarrow{1} D$	When only a certain value of information is to be transferred, it is indicated by the superscript above the arrow. This control signal permits only the binary 1's in A to be transferred to corresponding bit positions in D. When prior to this action D is cleared ($0 \longrightarrow D$), the two result in the transfer of a word. The converse control signal is $\xrightarrow{0}$.
$\xrightarrow{L}$	Transfer, shifting left one place.
$\xrightarrow{R}$	Transfer, shifting right one place.

Data Transfer Control Signals

The distinction on the drawings between the data transfer signal lines and the control signals which permit the transfers is made in the following manner. The control signal will have a functional name describing the transfer of a register contents, or portion thereof.

$(D) \longrightarrow Q$ $d_{-23} = 1$	The signal lines transferring the contents will be named by bit position, due to the parallel nature of the computer. The control signal will go to 48 AND gates. In this instance the action is:

$$\left[(D) \longrightarrow Q\right] \cdot \left[d_{-23} = 1\right] \Longrightarrow \left[1 \longrightarrow q_{-23}\right]$$

Representative Register Schematics

The computer is parallel in operation and registers contain many bits. Since the logic is identical for many of these bits, the drawing can be representative and show the logic circuit for just one of the bits. A table accompanies the logic circuit drawing to list the connections to all of the circuits. The information signal line names will bear a subscript or superscript "x" instead of the specific bit position, for example, $MA^x = 0$ or $PA^x = 1$. When a circuit for a specific bit is different from the rest, the logic circuit for the bit is included in the drawing. Its signal lines will, of course, bear specific identification, e.g., $ja_{-1} = 0$.

Data Registers - Subdivision of Control Signals

The data registers are subdivided into the mantissa and exponent sections to store floating point numbers. Furthermore, the end bits of each section have different logic circuits from the rest of the bits. The logic schematic shows 6 bits of the register (4 specific and 2 representative) and tables:

a_0 a_1 a_{-35} a_{-47}

a_{-m}(bits a_{-2} through a_{-34} of the mantissa)

a_{-e} (bits a_{-36} through a_{-46} of the exponent)

Examples of control signals for the register are:

$(Q_E) \xrightarrow{1} A$	Transfer the ones in the exponent section of Q to A.
$q_{-1} \xrightarrow{1} a_{-47}$	Transfer q_{-1} bit to a_{-47} if it is a one: $(q_{-1} \longrightarrow a_{-47}) \cdot q_{-1} = 1 \Longrightarrow 1 \longrightarrow a_{-47}$. The parentheses is used here to simplify reading the terms of this logic.
$0 \longrightarrow A_M$	Clear the mantissa section of the A register to zero. This control signal will clear bits a_0 through a_{-35}.

3. INSTRUCTION SELECTION AND PERFORMANCE

3.1 Basic Program Cycle

The basic program cycle consists of the actions in selecting and performing an instruction. Unless altered by a jump instruction, the sequence of instructions performed is consecutively through the memory (by address). Each word contains two instructions, the instruction in the left half of the word (2_0 to 2_{-23} bits) is performed first.

Normally, therefore, in this sequence, every other basic program cycle starts with a transfer from (V) $\longrightarrow$ PR for a new instruction word. A control is required to specify on alternate cycles that a new instruction word is required. Another control is required to specify which half of the instruction word contains the instructions to be performed. Lastly, a control is required to alternate the first two controls between their two possible states. Flip-flops are used for these controls.

PI = 01	transfer an instruction word to PR from memory
PI = 00	no instruction word transfer required
SW2 = 0	perform I_0 the instruction in PR_0
SW2 = 1	perform I_1, the instruction in PR_1
MOD 2 = 0	the next instruction to be performed will be in PR_0
MOD 2 = 1	the next instruction to be performed will be in PR_1

SW2 as the control of the location in PR of the instruction currently being performed causes Mod 2 to alternate:

$$(SW2)' \longrightarrow \text{Mod } 2$$

The complement of SW 2 is used since the next location is the alternate:

Mod 2's condition determines the ensuing state of PI.

$$(\text{Mod } 2 = 0) \Longrightarrow (1 \longrightarrow \text{PI})$$

$$(\text{Mod } 2 = 1) \Longrightarrow (0 \longrightarrow \text{PI})$$

And, at the proper time, Mod 2 will cause SW2 to alternate:

$$\text{Mod } 2 \longrightarrow \text{SW2}$$

Mod 2 and PI which indicate actions to be performed following the performance of the current instruction can be switched during the performance of an instruction. SW2 must remain unchanged during the performance of an instruction as it is gating the command part of the instruction word to the command decoding network. SW2 is therefore switched between instructions.

The basic program includes timings of two control registers, PI and II. Program timings (PTs) can be considered as program cycle activities between instructions. This is the time SW2 is changed. PT2 is used as it is a timing common to both cases, transfer necessary or unnecessary from memory to PR.

All instructions go through the earlier timings of IT so it can be considered part of the basic program cycle. The complement of SW2 is transferred to Mod 2 in IT2. Following this, Mod 2's new state is transferred to PI in IT2 or IT3. The setting of PI in IT3 determines whether or not to get a new instruction word during the next set of PTs.

PI Time PI = 00

If the next instruction is already in PR, very little requires to be done during the PTs. To save time the computer in this case goes from "END" (a gating decision at the end of timings for AT, IT and FT) to PT2. In PT2, (Mod 2) $\longrightarrow$ SW2, to prepare to select the next instruction from PR. Then, unless the computer pauses due to a breakpoint or overflow or fault or operator intervention, it proceeds from PT4 to IT1 and the next instruction.

PI Time PI = 01

Three operations occur during these timings. A new instruction word is transferred from memory to PR (its address was stored in the PA register). Secondly, this address is increased by one and restored in PA to provide the next address of a normal sequence of instructions. Lastly, (Mod 2 $\longrightarrow$ SW2) to select the PR location. The activity during the four timings (PRl toPT4) is:

PT1

1.	Clear MA to receive address from PA	$0 \longrightarrow$ MA
2.	Connect AN2 inputs for incrementing the address	
	MA $\longrightarrow$ SW4 — address will enter adder via MA	
	1 $\longrightarrow$ AN2C — increment of one	
	0 $\longrightarrow$ PM — no input from SW6	
3.	Clear PR for new word	$0 \longrightarrow$ PR

PT2

1.	Transfer address to MA to decode for memory access and for input to AN2	(PA) $\longrightarrow$ MA
2.	Start memory cycle	$1 \longrightarrow$ MI
3.	Prepare to select next instruction	(Mod 2) $\longrightarrow$ SW2

PT3

1.	Clear PA for incremented address	$0 \longrightarrow$ PA

PT4

1.	Transfer incremented address to PA	(AN2) $\longrightarrow$ PA
2.	Proceed to next instruction unless control signals a pause	$\longrightarrow$ IT1

Computer Pause

The term "pause" is used to identify the cause of the stopping of computer operation. Pause indicates the computer stopped for a reason other than a programmed stop (the Halt instruction) or machine malfunction (usually termed "hanging up"). The pause can be due to a Fault, Overflow, MA = MP, Breakpoint, operating in the STEP or TEST modes, or operator intervention by depressing the STOP switch.

It is normally desirable to have the computer pause just after completing the instruction during which a pause was signalled. When a Fault develops (Command, Memory or Exponent Fault) it is convenient to learn by the console display what instruction and what part of the program was being performed. If it were the I_1 instruction, it is necessary to preserve (PR) by not reading in the next instruction word. The fault condition will therefore inhibit two signals, $1 \longrightarrow$ PT1 (avoid read-in new instruction word) and $1 \longrightarrow$ IT1 (do not go ahead to next instruction).

However, in the case of overflow, a unique situation exists due to the two jump instructions which are conditional upon overflow (JNO, JOF). In the case of JOF following an instruction whose performance generated an overflow, the computer should not pause, but proceed to jump.

This requires that program control must be always aware of the next instruction before a pause due to overflow. Therefore overflow will never stop the program cycle before PT1. The next instruction is an overflow jump. The timing logic of PT4 insures that the new word is in PR before the decision is made whether or not to pause. This situation also exists in the case of breakpoint, the JBT instruction.

STEP mode, MA = MP or operator intervention will cause the computer to pause in the same manner as a Fault, before PT1 or IT1. TEST will be discussed in detail in the section on Operating Modes.

First Basic Program Cycle

This first cycle must be manually selected. Console switches offer three choices:

I_0	Do the left instruction already in PR.
I_1	Do the right instruction already in PR.
I_v	Read next instruction word.

$I_0 \Longrightarrow 0 \longrightarrow SW2, \quad 0 \longrightarrow PI,\ PI^*$

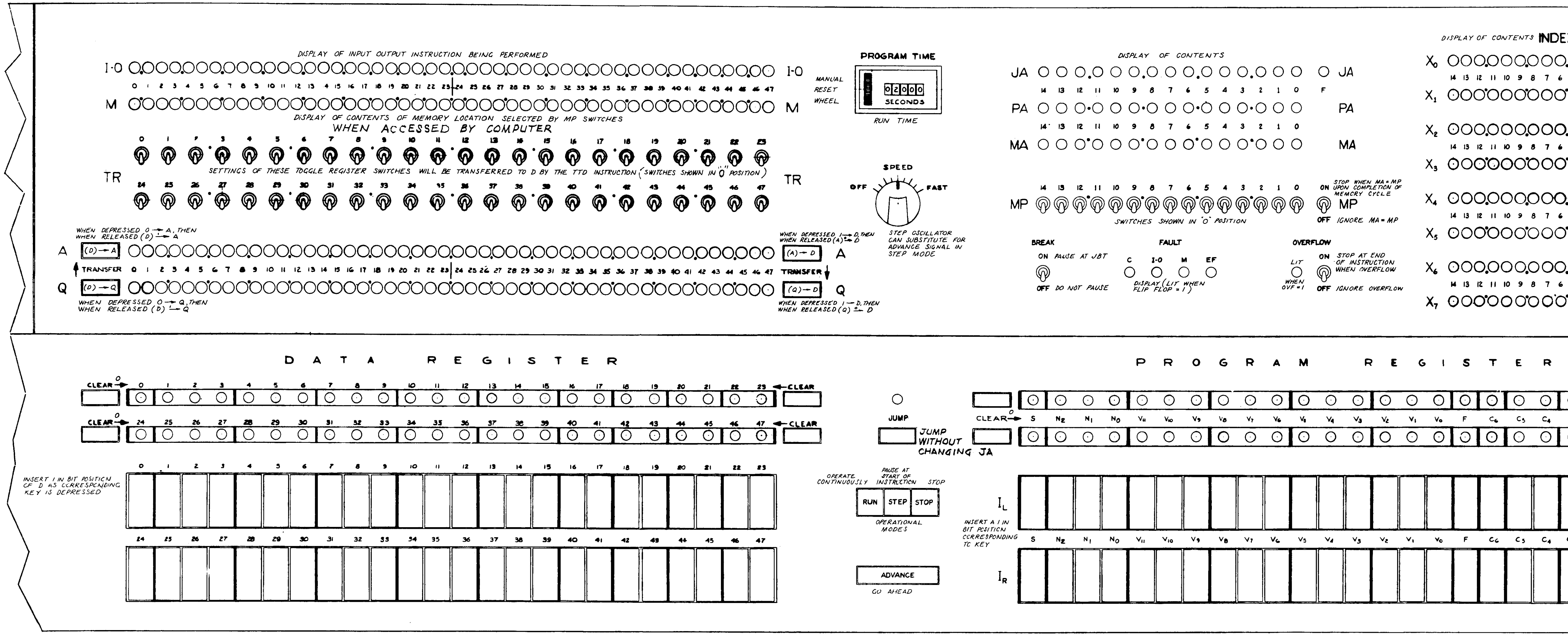
$I_1 \Longrightarrow 1 \longrightarrow SW2, \quad 0 \longrightarrow PI,\ PI^*$

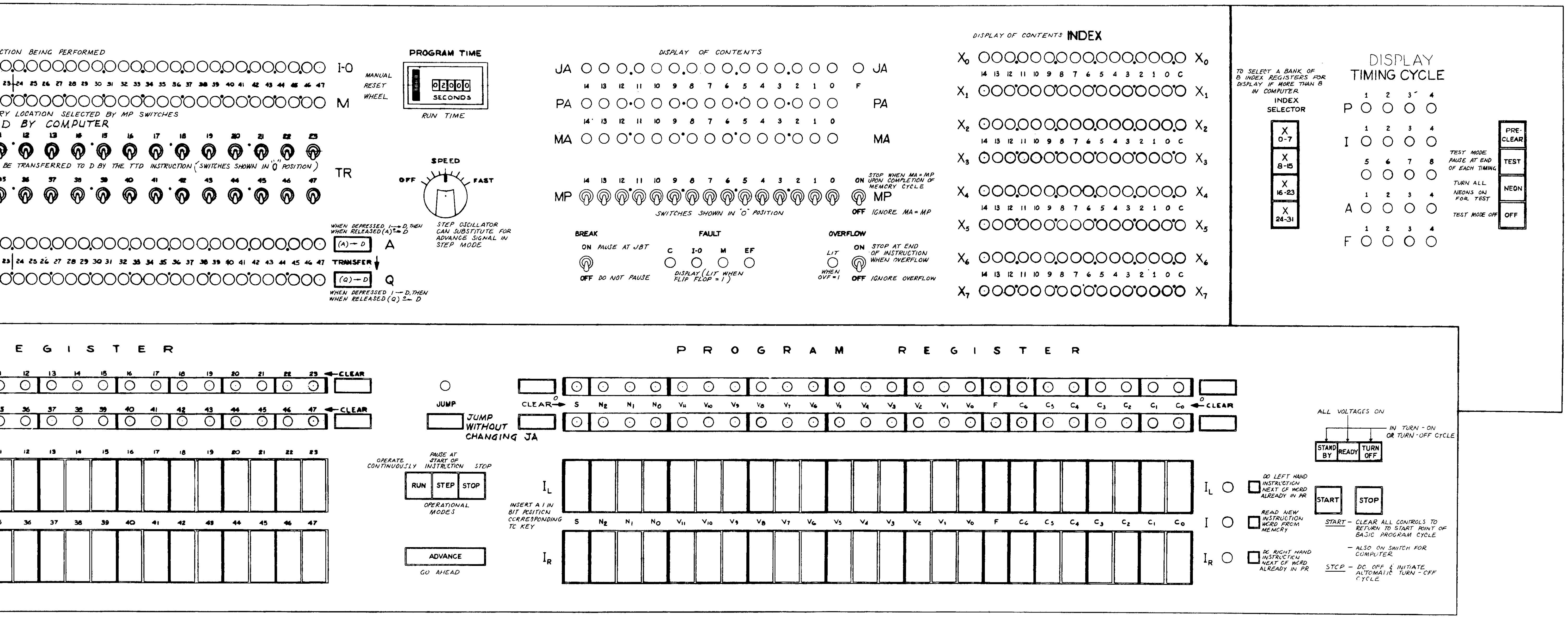
$I_v \Longrightarrow 0 \longrightarrow \text{Mod } 2 \quad 1 \longrightarrow PI$

4. COMPUTER CONSOLE AND OPERATING

4.1 Console and Operating Controls

Figure 4.1-1 shows the S-2000 console. A brief description of each operating control and visual display is noted alongside each part of the console in this drawing.


DISPLAY OF INPUT OUTPUT INSTRUCTION BEING PERFORMED
I-O
M
DISPLAY OF CONTENTS OF MEMORY LOCATION SELECTED BY MP SWITCHES WHEN ACCESSED BY COMPUTER
TR
SETTINGS OF THESE TOGGLE REGISTER SWITCHES WILL BE TRANSFERRED TO D BY THE TTD INSTRUCTION (SWITCHES SHOWN IN "0" POSITION)
WHEN DEPRESSED 0 → A, THEN WHEN RELEASED (D) → A
A
(D) → A
TRANSFER
Q
(D) → Q
WHEN DEPRESSED 0 → Q, THEN WHEN RELEASED (D) → Q
WHEN DEPRESSED 1 → D, THEN WHEN RELEASED (A) → D
(A) → D
TRANSFER
(Q) → D
WHEN DEPRESSED 1 → D, THEN WHEN RELEASED (Q) → D
PROGRAM TIME
MANUAL RESET WHEEL
SECONDS
RUN TIME
SPEED
OFF
FAST
STEP OSCILLATOR CAN SUBSTITUTE FOR ADVANCE SIGNAL IN STEP MODE
DISPLAY OF CONTENTS
JA
PA
MA
F
MP
SWITCHES SHOWN IN "0" POSITION
ON STOP WHEN MA = MP UPON COMPLETION OF MEMORY CYCLE
OFF IGNORE MA = MP
BREAK
ON PAUSE AT JBT
OFF DO NOT PAUSE
FAULT
C I-O M EF
DISPLAY (LIT WHEN FLIP FLOP = 1)
OVERFLOW
LIT WHEN OVF = 1
ON STOP AT END OF INSTRUCTION WHEN OVERFLOW
OFF IGNORE OVERFLOW
DISPLAY OF CONTENTS INDE
X₀ X₁ X₂ X₃ X₄ X₅ X₆ X₇
DATA REGISTER
CLEAR → 0
← CLEAR
INSERT 1 IN BIT POSITION OF D AS CORRESPONDING KEY IS DEPRESSED
JUMP
JUMP WITHOUT CHANGING JA
OPERATE CONTINUOUSLY
PAUSE AT START OF INSTRUCTION
STOP
RUN
STEP
OPERATIONAL MODES
ADVANCE
GO AHEAD
PROGRAM REGISTER
S N₂ N₁ N₀ V₁₁ V₁₀ V₉ V₈ V₇ V₆ V₅ V₄ V₃ V₂ V₁ V₀ F C₆ C₅ C₄
I_L
I_R
INSERT A 1 IN BIT POSITION CORRESPONDING TO KEY



PROGRAM TIME
MANUAL RESET WHEEL
SECONDS
RUN TIME
SPEED
OFF
FAST
STEP OSCILLATOR CAN SUBSTITUTE FOR ADVANCE SIGNAL IN STEP MODE
I-O
M
TR
A
Q
TRANSFER
WHEN DEPRESSED 1 → D, THEN WHEN RELEASED (A) → D
WHEN DEPRESSED 1 → D, THEN WHEN RELEASED (Q) → D
(A) → D
(Q) → D
DISPLAY OF CONTENTS
JA
PA
MA
MP
SWITCHES SHOWN IN "0" POSITION
ON STOP WHEN MA = MP UPON COMPLETION OF MEMORY CYCLE
OFF IGNORE MA = MP
BREAK
ON PAUSE AT JBT
OFF DO NOT PAUSE
FAULT
C I-O M EF
DISPLAY (LIT WHEN FLIP FLOP = 1)
OVERFLOW
LIT WHEN OVF = 1
ON STOP AT END OF INSTRUCTION WHEN OVERFLOW
OFF IGNORE OVERFLOW
DISPLAY OF CONTENTS INDEX
X0 X1 X2 X3 X4 X5 X6 X7
TO SELECT A BANK OF 8 INDEX REGISTERS FOR DISPLAY IF MORE THAN 8 IN COMPUTER
INDEX SELECTOR
X 0-7
X 8-15
X 16-23
X 24-31
DISPLAY TIMING CYCLE
P I A F
TEST MODE PAUSE AT END OF EACH TIMING
TURN ALL NEONS ON FOR TEST
TEST MODE OFF
PRE-CLEAR
TEST
NEON
OFF
REGISTER
CLEAR
JUMP
JUMP WITHOUT CHANGING JA
OPERATE CONTINUOUSLY
PAUSE AT START OF INSTRUCTION
STOP
RUN
STEP
STOP
OPERATIONAL MODES
ADVANCE
GO AHEAD
PROGRAM REGISTER
S N2 N1 N0 V11 V10 V9 V8 V7 V6 V5 V4 V3 V2 V1 V0 F C6 C5 C4 C3 C2 C1 C0
IL
IR
I
INSERT A 1 IN BIT POSITION CORRESPONDING TO KEY
DO LEFT HAND INSTRUCTION NEXT OF WORD ALREADY IN PR
READ NEW INSTRUCTION WORD FROM MEMORY
DO RIGHT HAND INSTRUCTION NEXT OF WORD ALREADY IN PR
ALL VOLTAGES ON
IN TURN-ON OR TURN-OFF CYCLE
STAND BY
READY
TURN OFF
START
STOP
START - CLEAR ALL CONTROLS TO RETURN TO START POINT OF BASIC PROGRAM CYCLE
- ALSO ON SWITCH FOR COMPUTER
STOP - DC OFF & INITIATE AUTOMATIC TURN-OFF CYCLE

Figure 4.1-1 S-2000 Console

Computer ON or OFF

A single switch initiates the automatic turn-on cycle of the computer. This is the START switch at the extreme right of the console. At the end of the turn-on cycle, the READY indication will light.

The STOP switch at the extreme right of the console will turn D. C. off and initiate the automatic turn-off cycle.

Power for the Magnetic Core Stack Heaters is independent of the computer on-off switches and can be discontinued only at the main circuit breaker.

This START switch also is used to establish the initial settings of the control registers and other control elements, effectively, clearing the controls to return the computer to the starting point of the basic program cycle.

Program Register

An instruction word can be manually entered directly into the Program Register by means of the key switches, numbered corresponding to bit positions.

PR is cleared to zero by the CLEAR switches alongside the display neons. Four switches enable changing information in only part of the register. The color of a switch and panel indicates the part of the register controlled by it.

D Register

Manual entry directly into the D Register is provided by CLEAR and key switches similar to those for PR.

A and Q Registers

The A and Q registers are manually accessible by transfers from D. Manual transfer switches (to the left of the display neons for the A and Q registers) control the transfer from D.

Manual transfers from A or Q to D are controlled by the switches to the right of the register display neons.

Toggle Register

This register, whose contents are manually selected, can be transferred to the D register under control of the program, namely the TTD instruction. With a suitably designed program, (TR) can be used to manually indicate which sections of a program are to be performed during the run. This method will be used in performing the maintenance routines. TR consists of 48 two-position toggle switches having a binary value of "0" in the down position.

M Register

The M register is used solely for display to monitor the contents of any single memory location. The location is manually selected by the MP toggle switches on the right side of the console.

I-O Register

This register also only has a display function. Its contents are the input-output instruction currently being performed. Or, the last I-O instruction if none is current.

JA, PA, MA, X Displays

These console neons are connected to the registers indicated and display their contents. The JA register has an F bit to indicate with which half of the instruction word the normal sequence of program control is to be resumed. The Index Registers each have a counter bit, C. When a system has more than eight Index Registers, they will be manually selected for display, in groups of eight, by the INDEX SELECTOR switches to the right of the neons.

MP Switches

These select a memory location for display in the M register. The down position of these two-position toggle switches is the binary "0" position. When the address selected by the MP switches coincides with the address used by the program, or program control, an option of stopping the computer is available. The coincidence produces an active MA = MP signal.

MA = MP

The switch immediately to the right of the MP_0 switch and labelled ON, OFF will cause the computer to halt if in the ON position, and MA = MP. The computer halts at the end of the memory cycle.

Overflow

The neon is lit when the Overflow flip-flop is in the one state, indicating overflow. If the switch is ON, the computer will stop at the end of the instruction during which overflow occurred, unless the following instruction is an overflow jump instruction.

Faults

The development of any of four faults (Command Fault, Input/Output Fault, Memory Fault, Exponent Fault) will be indicated by a lighted neon.

Command Fault is the result of an illegitimate command coding in the instruction. The eight-bit command code provides 256 possible coding combinations. Some of these are presently unused. If for some reason the combination appears in PR, the computer cannot proceed until the command coding is manually changed to that of an existing instruction.

Memory Fault exists when the core stack temperature is outside its operating range ($104^\circ F \pm 1\text{-}1/2^\circ F$). Memory operation is unreliable unless in range. The computer will cease to operate until the temperature is restored within range.

Exponent Fault represents the development of a quantity too large to represent in the computer as the result of floating point arithmetic.

Input/Output Fault is a type due to a control failure in any of the I-O equipment that prevents it from completing its operation. Generally, computer operation is not stopped. The faulty I-O sector cannot do succeeding operations having failed to complete one. However, other types of I-O equipment can be operated. Uniquely, an I-O Fault from a magnetic drum operation causes the computer to cease operating as the computer can be doing nothing else during a drum instruction performance.

Breakpoint Switch

When the BREAK switch is in the ON position, the computer will pause before performing the JBT instruction.

Stop Switch

When STOP is depressed, it latches itself and releases the RUN switch to deactivate P20. The computer will stop before performing the next instruction.

Advance

The ADVANCE switch signals the computer to "go ahead". The switch itself is merely used to set a flip-flop which provides a steady Advance signal. Otherwise switch contact bounce would result in a group of pulses. (This type of logic is called a "filter flip-flop".)

Jump

The JUMP switch will cause execution of a jump instruction that has been entered into PR without changing the contents of the JA register. This enables manual departure from a program without losing the point in the program run at which the intervention occurred.

This jump is executed by entering the address, to which to jump, in the I_V field of either half of PR by means of the manual switches. The PR_J bit is made the proper value to indicate whether to jump to the left or right. Either the I_L or the I_R manual switch is used to make this entered information the operative half of PR. Lastly the JUMP switch is depressed and released.

NOTE: If the R bit of the operative half of PR is equal 1, (X) will be added to (I_V) to form the effective address for the jump.

Preclear

The PRECLEAR switch, located in the upper right of the console, restores all controls which begin the basic program cycle in a fixed condition. Its action parallels that of the START switch except that it does not turn on the computer.

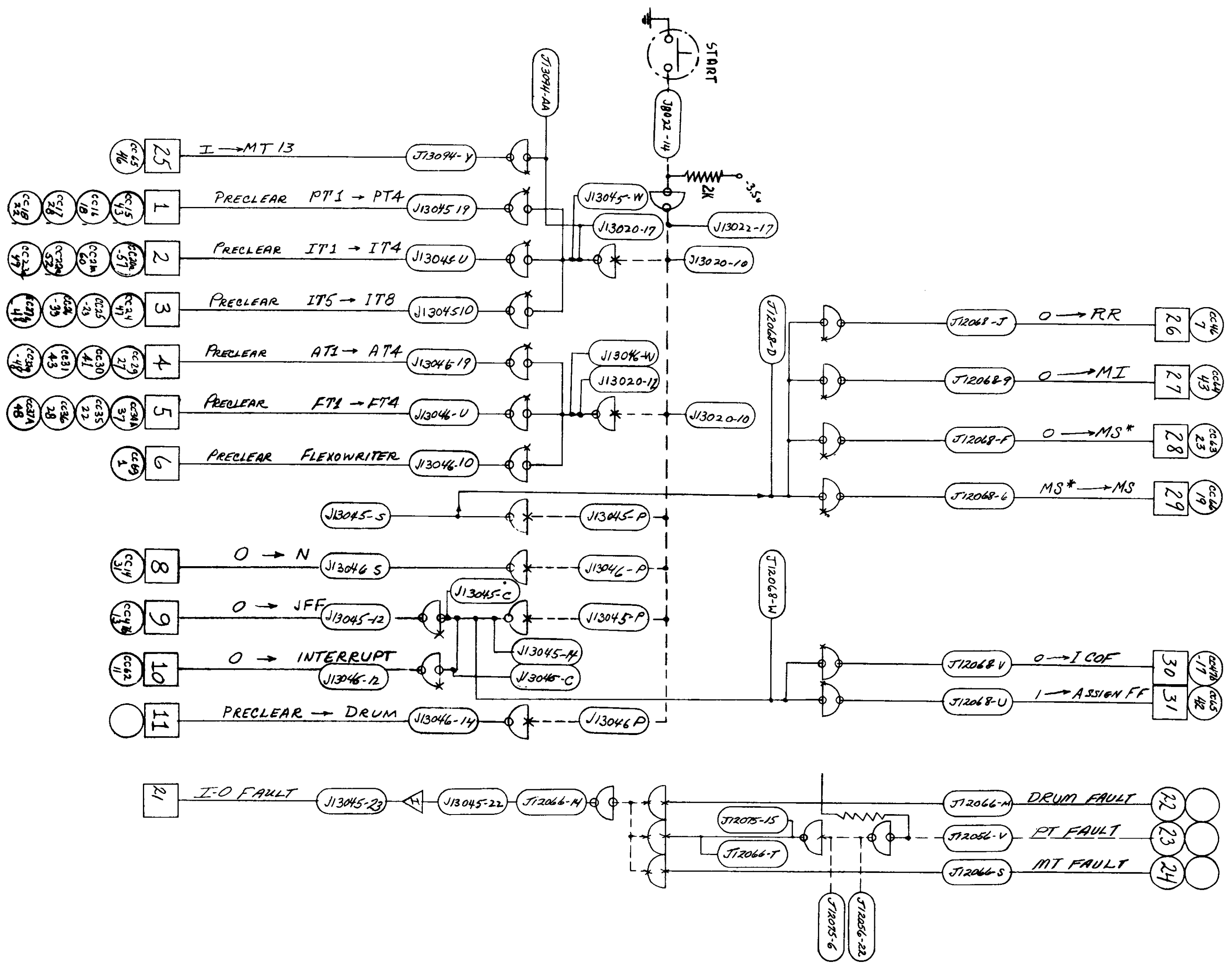


Figure 4.1-2 Console Controls (Partial)

4.2 Operational Modes

The mode selection switches, on the lower center of the console, determine the operating rate of the basic program cycle. In the RUN mode, the computer operates continuously, from one program cycle to the next in the absence of a fault. In the STEP mode, the computer pauses each cycle. The STOP switch is used to manually intervene in the operation. The computer is stopped at the completion of the current program cycle, when the stop switch is depressed, and cannot operate further.

The three switches are mechanically latching and interlocking so that only one can be kept closed at any time. Depressing one releases the other two. The ADVANCE switch, when depressed, will initiate the RUN mode of operation, if the RUN was previously depressed. The ADVANCE switch is a momentary switch. If the STEP is depressed, the ADVANCE is used to start the computer off on each program cycle. The ADVANCE switch is ineffective when STOP is depressed. In the STEP mode, the STEP OSCILLATOR can be substituted for the ADVANCE switch to make the program cycle repetition rate that of an oscillator whose frequency can be varied between one and ten pulses per second.

The point of the program cycle where the computer pauses before each instruction is the exit from PT4 to IT1. In addition the computer will pause before exiting to PT1. To summarize, in the STEP mode the computer pauses before every instruction and before transferring a new instruction word from memory. The pause is due to the lack of a permissive RUN signal, P20.

Referring to Figure 4.2-1, $P20 \equiv \text{RUN mode} \cdot \overline{\text{MEM FAULT}} \cdot (EF = 0) \cdot (CF = 0) \cdot (\text{Stop FF} = 0)$. EF and CF are the Exponent Fault and Command Fault flip-flops respectively. When set, they indicate the named error. Mem Fault indicates the core storage unit temperature is outside its operating range. The Stop FF is unrelated to the STOP mode switch. The Stop FF is set by a programmed halt, either the Halt instruction or when a selected memory address has been reached in the program.

P20 is one of the conditions required for timing to proceed from PT4 $\longrightarrow$ IT1. If it is inhibitory during PT4, PT4 will be turned off without producing a signal to start the next timing (IT1 in this case). The ADVANCE switch will reset the three flip-flops (EF, CF, STOP) and cause the computer to proceed to IT1.

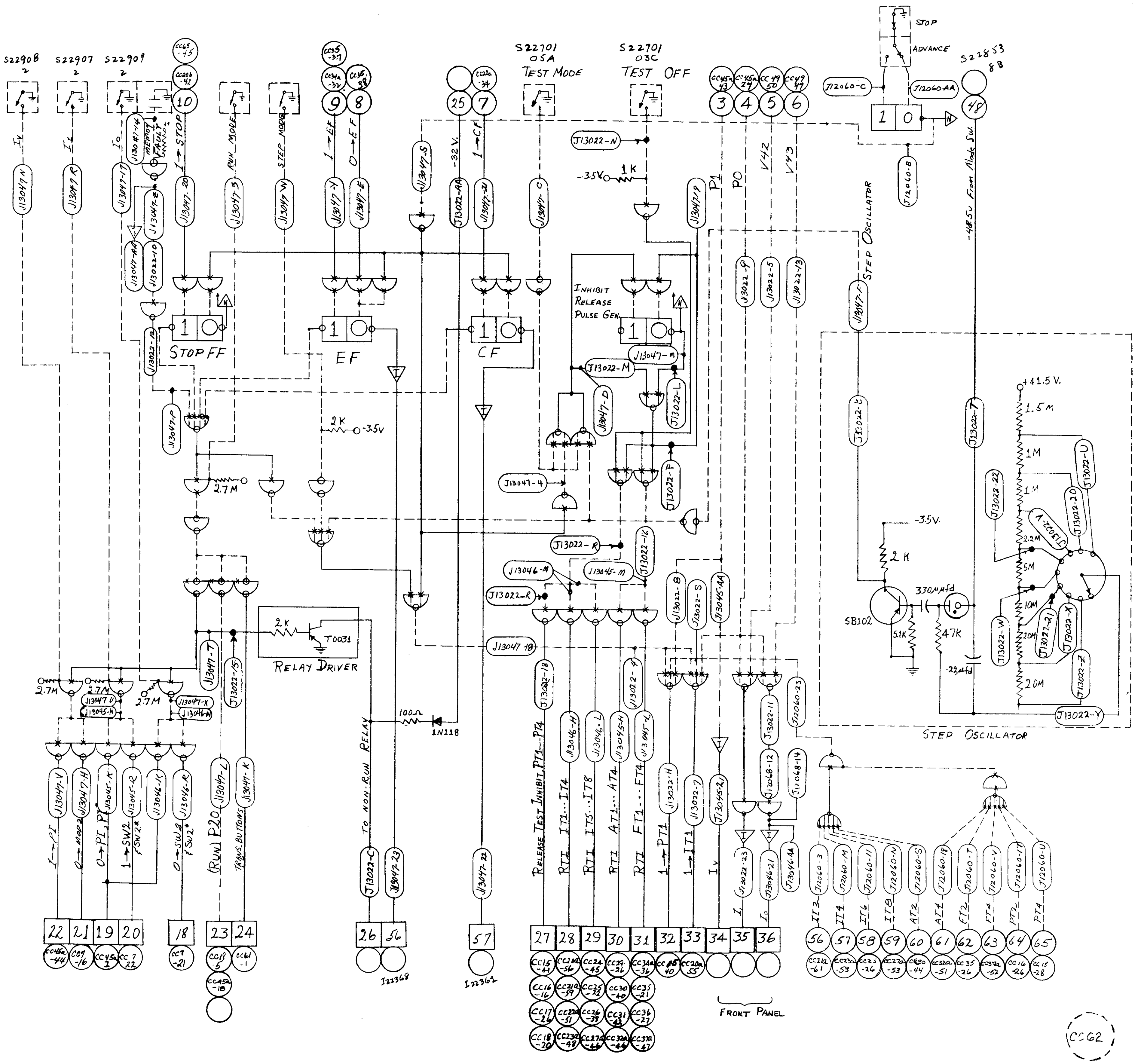


Figure 4.2-1 Front Panel Controls

ADVANCE $\cdot$ PO $\Longrightarrow$ IT1 (PO $\equiv$ (PI 0), do the next instruction). In PT3, before the pause, Mod 2 $\longrightarrow$ SW2 prepared the latter to indicate whether the next instruction was located in I_0 or I_1.

Test Mode

The Test mode of operation causes the computer to pause at the end of each timing. This is achieved by the following method of turning on a timing. "Turning on" can be defined as causing a permissive (active) signal on all lines carrying the timing signal to the gates involving actions to be performed during the timing.

The method requires two conditions to turn on a timing -- that the associated timing flip-flop be set to one and the subsequent removal of the setting signal.

$$xT_n \equiv (xT_n = 1) \cdot \overline{(1 \longrightarrow xT_n)}$$

where "xT_n" represents any control timing and its number. $(1 \longrightarrow xT_n)$ generally comes from the flip-flop of the preceding timing. The signal becomes inhibitory when the preceding timing flip-flop is reset to zero. If the resetting of the preceding flip-flop is prevented, the computer will pause as xT_n timing cannot be turned on. Two flip-flops will be set, xT_n and xT_{n-1}.

A two-high AND gate is at the reset input of each of the timing flip-flops. One of the signals to the gate is the signal that also goes to set the next timing flip-flop. The other signal at the reset gate is Release Test Inhibit. When the TEST mode is OFF, this signal is constantly active (permissive). In the TEST mode, this signal is inactive and prevents the resetting of the flip-flop. The Release Test Inhibit signal can be momentarily made active by either the manual ADVANCE switch or the STEP OSCILLATOR.

When the computer is paused in this fashion, both timing flip-flops are set, but only the earlier timing is on.

The computer is placed in the Test Mode by depressing the TEST switch in the upper right part of the console. TEST is removed by depressing the OFF switch. (The NEON switch turns on all neons in the computer, with the one exception of the neon on the card containing the Step Oscillator circuit, to enable a check of the visual indicators.)

The TIMING CYCLE display will indicate, during TEST mode which timing is on. Generally, two neons are lit as two timing flip-flops will be set. However only the earlier of the two timings is active.

Inhibit Release Pulse Generator

A method is required to insure that when the computer is released to perform the next timing, in the TEST mode, the release signal will be a single pulse that ends before the next timing is completed. Otherwise the computer might perform several timings per release. The pulse is provided by the Inhibit Release Pulse Generator (IRPG).

The signal which, when active, permits the timing flip-flops to be reset, is labelled "Release Test Inhibit" (RTI) on Figure 4.2-1.

The usual two-step logic is employed for an interlock:

$$\text{RTI} = \text{Test OFF} \ \vee \ (\text{IRPG} = 1 \ \cdot \ \overline{1 \longrightarrow \text{IRPG}})$$

$$1 \longrightarrow \text{IRPG} = \text{TEST} \ . \ (\text{ADVANCE} \vee \text{Step Oscillator})$$

$$\overline{(\text{TEST} \vee \text{ADVANCE} \vee \text{Step Osc.})} \ . \ \text{IRPG} = 1 \Longrightarrow \text{RTI}, \ 0 \longrightarrow \text{IRPG}$$

It is also necessary during TEST to inhibit the ADVANCE or Step Osc. active signals from sending 1 ⟶ PT1 or 1 ⟶ IT1 in the presence of P1 or P0. The logic "ORs" all of the other timings together to inhibit the above set signals if any other timing flip-flop is set.

5. LOGIC SECTIONS OF COMPUTER

5.1 Adder Networks

Both adder networks of the computer, AN1 and AN2, use the same principles of logical organization. The adder logic will be explained using AN1 as the illustration, see Figure 5.1-1.

AN1 Inputs

The data inputs to AN1 (A and D*) are always connected to the adder network. That is, no switching or gating is involved. The sum output of AN1 (A + D*) goes to A* through gates and a permissive control signal is required to transfer the adder sum to A*.

The Adder

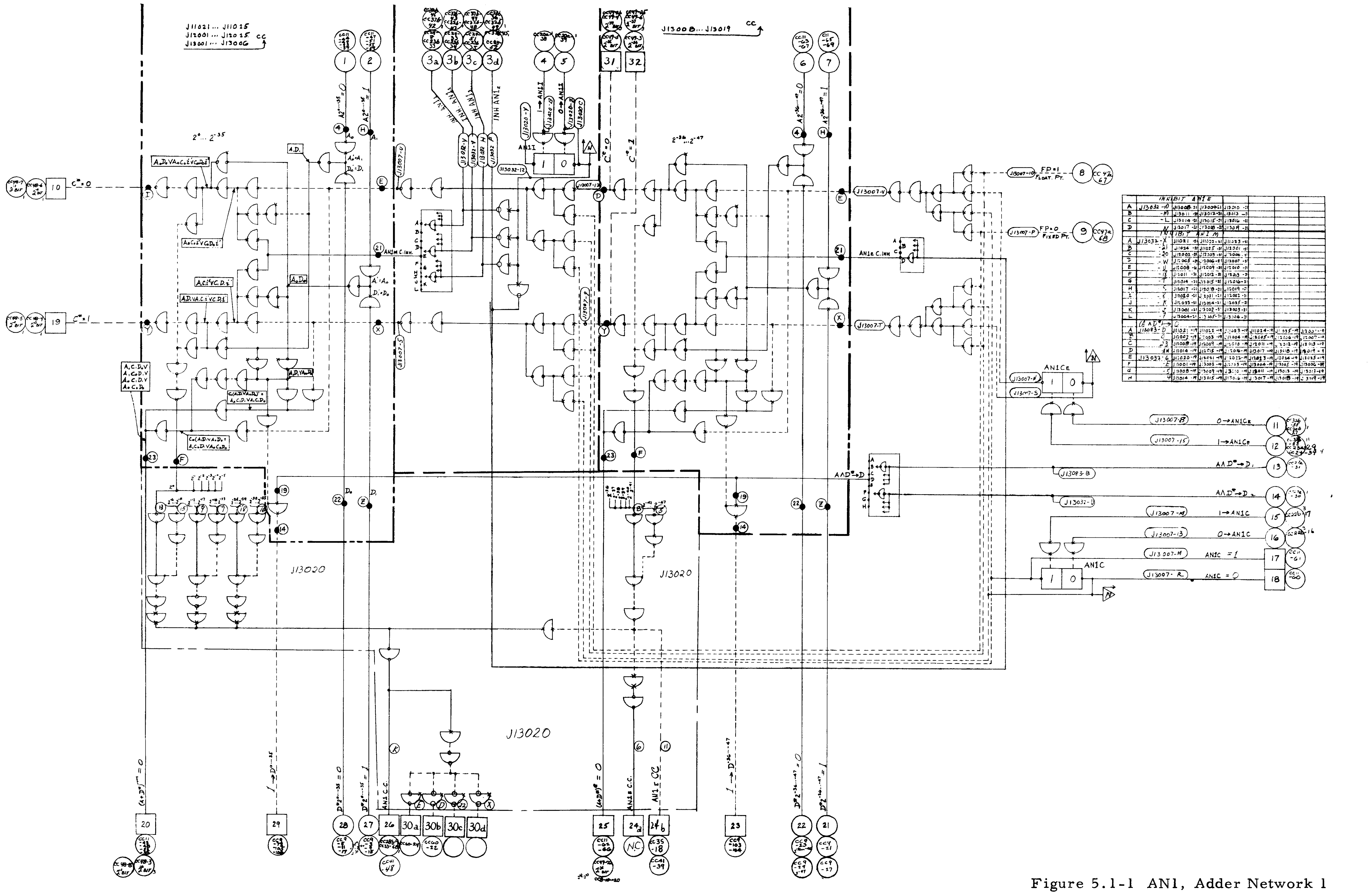
The adder is a logical network and its output is always available with one exception. The exception is the time required to respond to a change of information in one of its inputs, A or D*.

The result of the addition of two multi-order numbers (numbers having more than one digit) cannot be instantaneously determined. The determination of one of the three values, the carry-in, to be added per column has an unavoidable, finite time delay due to the adder circuitry of the adjacent lower order.

The carry-in will influence the sum of the column and may influence the carry-out from that column. The adder must be organized so that the sum for an order is not indicated until the carry-in exists to that order.

As the carry-in to an order is the carry-out from the adjacent lower order, the existence of a carry-out from every order of the number, can be used to indicate the presence of the sum for each order and the completion of the addition. For the binary adder this requires two carry-out signal lines, CO = 1 and CO = 0.

With the stipluation that the sum will not be determined until the three inputs to a column exist, the sum output can be obtained with just one signal line for each order. In AN1, the Sum = 0 line is the one



	INHIBIT AN1E						
A	J13032 -10	J13008-21	J13009-21	J13010 -21			
B	-M	J13011 -21	J13012-21	J13013 -21			
C	-L	J13014 -21	J13015-21	J13016 -21			
D	N	J13017 -21	J13018 -21	J13019 -21			
	INHIBIT AN1M						
A	J13032 -X	J11021 -21	J11022 -21	J11023 -21			
B	-21	J11024 -21	J11025 -21	J12001 -21			
C	-20	J12002 -21	J12003 -21	J12004 -21			
D	-W	J12005 -21	J12006 -21	J12007 -21			
E	-U	J12008 -21	J12009 -21	J12010 -21			
F	-18	J12011 -21	J12012 -21	J12013 -21			
G	-T	J12014 -21	J12015 -21	J12016 -21			
H	-S	J12017 -21	J12018 -21	J12019 -21			
I	-Z	J12020 -21	J12021 -21	J12022 -21			
J	-K	J12023 -21	J12024 -21	J12025 -21			
K	-5	J13001 -21	J13002 -21	J13003 -21			
L	-7	J13004 -21	J13005 -21	J13006 -21			
	(E AD*) → D						
A	J13083 - D	J11021 -19	J11022 -19	J11023 -19	J11024 -19	J11025 -19	J12001 -19
B	C	J12002 -19	J12003 -19	J12004 -19	J12005 -19	J12006 -19	J12007 -19
C	23	J12008 -19	J12009 -19	J12010 -19	J12011 -19	J12012 -19	J12013 -19
D	AH	J12014 -19	J12015 -19	J12016 -19	J13017 -19	J13018 -19	J13019 -19
E	J13032 - 6	J12020 -19	J12021 -19	J12022 -19	J12023 -19	J12024 -19	J12025 -19
F	-E	J13001 -19	J13002 -19	J13003 -19	J13004 -19	J13005 -19	J13006 -19
G	-5	J13008 -19	J13009 -19	J13010 -19	J13011 -19	J13012 -19	J13013 -19
H	4	J13014 -19	J13015 -19	J13016 -19	J13017 -19	J13018 -19	J13019 -19

Figure 5.1-1 AN1, Adder Network 1

used. The only other possible output is implied by the absence of an active S= 0 line. For example, the A* register is set to all ones before addition and the S = 0 lines from AN1 (Adder Network 1) will enter the sum in A*.

The inputs to each bit adder in AN1 consist of six lines, A_1 A_0 D_1 D_0 from the registers and C_1 C_0 from the adjacent lower order bit adder, where the subscripts indicate the binary value of the active signal lines.

The S-2000 adder logic for a bit is as follows:

1. If A = D, the carry-out is immediately determined as it will not be influenced by the carry-in.

$$A_1 D^*_1 \Rightarrow CO = 1$$

where CO is the carry-out

$$A_0 D^*_0 \Rightarrow CO = 0$$

2. Otherwise carry-out awaits the carry-in.

$$\overline{A_1 D^*_1} \cdot CI = 0 \Rightarrow CO = 0$$

where CI is the carry-in

$$\overline{A_0 D^*_0} \cdot CI = 1 \Rightarrow CO = 1$$

3. The sum also awaits the carry-in. The conditions for a sum of zero are then acted upon.

$$(A_1 D^*_1 \lor A_0 D^*_0) \cdot CI = 0 \Rightarrow S = 0$$

$$(\overline{A_1 D^*_1} \lor \overline{A_0 D^*_0}) \cdot CI = 1 \Rightarrow S = 0$$

4. The completion of the adder network operation for the addition is signaled by an active ANCC line (Adder Network Carry Complete). An active carry-out line from each bit adder is required for this signal:

$$(CO_0 = 1 \lor CO_0 = 0) \cdot (CO_{-1} = 1 \lor CO_{-1} = 0) \cdot$$

$$(CO_{-2}\ 1 \lor CO_{-2} = 0) \ldots\ldots \cdot (CO_{-47} = 1 \lor CO_{-47} = 0)$$

$$\Rightarrow AN1CC$$

for the case of fixed point addition in AN1, where the subscripts for "CO" indicate the bit position.

5. Thislogic requires carry-in lines to the least significant bit, 2_{-47}, (or bits in the case of floating point, 2_{-35} and 2_{-47}). These lines are energized by the flip-flops ANC (Adder Network Carry) and ANC_E (Adder Network Exponent Carry). These are established in the required condition prior to the addition.

Inhibiting the Adder

As the adder inputs are always connected to the adder, it is desirable to prevent the adder from being active whenever the contents of a register providing an adder input is being changed. One method of achieving this is to prevent the indication that a sum is available in the adder. This would mean preventing the existence of an active Carry Complete signal.

Whenever a transfer is made to a register providing an adder input one of the carry inhibit signals is activated. These signals prevent the carry-in to each bit position, thereby forestalling the carry-out and a Carry Complete signal, unless the two inputs are identical.

The carry inhibits are activated by two sources:

$1 \longrightarrow \text{AN1I}$ a flip-flop

$\text{AN1I} = 1 \Rightarrow \text{AN1 C. Inh.}$

and

$\text{Inh. AN1} \Rightarrow \text{AN1 C. Inh.}$

The Inh. AN1 signal (Inhibit AN1) is used to inhibit the adder for the brief intervals while a transfer to A or D* is being made. The AN1I flip-flop is used to enable the adder network to be used for comparison of A and D* instead of addition. A flip-flop is used as the carry inhibit signal is required to be active over a large part of the comparision instruction performance time.

Subtraction

Subtraction is done by adding the two's complement of D to A. For addition, D* is cleared to zeros and the ones in D are transferred to D*. For subtraction, the one's complement of D is transferred to D* by inversion. D* is cleared to ones and then the ones in D will be used to change the corresponding bits of D* to zeros. Furthermore, the carry-in to the least significant bit of AN1 is made a "one" to effectively provide the two's complement.

Floating Point

For floating point, AN1 is divided into two independent sections to handle the mantissa and exponent. This requires carry-in lines to each least significant digit of the two parts of the floating point number. AN1C flip-flop provides the carry-in to bit -35 and AN1CE flip-flop to bit -47. (In fixed point arithmetic AN1C provides the carry-in to bit -47. The use of these flip-flops is controlled by the FP = 0, Floating Point = 0, and FP = 1 signals).

AN1CC still represents carry complete for all 48 bits, but a separate signal line also exists, AN1ECC, to indicate carry completed for bits -36 through -47. The carry-outs from the sign and most significant bits of the exponent are required. These are four lines, two each from c_{-36} and c_{-37}.

AN2

AN2 operates similarly to AN1. Its differences are the following. The inputs to AN2 are selected among several sources. One input comes from SW4, the other from SW6. Only the ones from the register flip-flops are brought to SW4 and SW6. The outputs of the switches are inverted to derive the zeros input to AN2 for the bit, as well as the ones input. The SW6 outputs are gated with PM to control whether the information or its inverse (one's complement) is sent to AN2.

The logic of the adder network, AN2, is almost the same as for AN1. The sum outputs however are the ones bits ($AN2^n = 1$). The AN2 sum is transferred to MA through gates controlled by the "AN2 $\longrightarrow$ MA" signal.

The AN2I and AN2C flip-flops perform the carry inhibit and carry-in functions. The AN2CC signal is active when all bits have a carry-out.

5.2 Registers

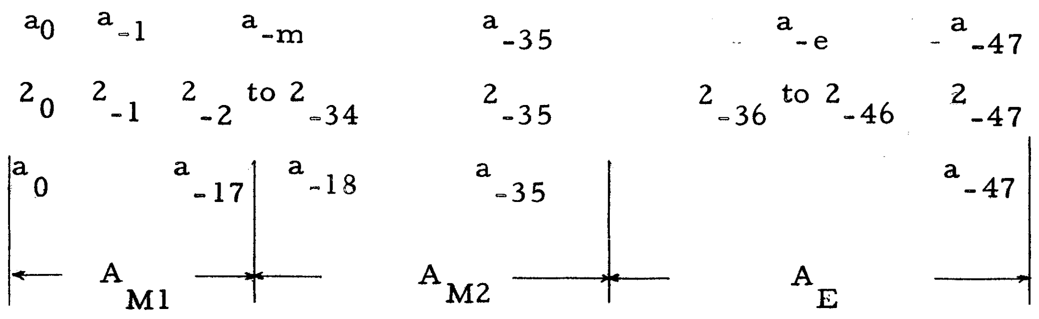
Register Sectors for Control

As previously mentioned, the logic diagram for a register, where parallel transfers occur, may be a condensed version of the actual situation. The schematic logic diagram of the A and A* registers can be illustrative of the notation method.

The 48 bits of the register have to be divided into groups. Circuit design considerations set a maximum number of bit circuits that may be driven by one driver circuit. In conforming to this requirement, the grouping is chosen so as to also meet the floating point word format requirement. This first division of the registers' bits forms three groups. M1 and M2 are the more and lesser significant parts, respectively, of the mantissa, (bits 2_0 through 2_{-35}). E is the exponent part, bits 2_{-36} through 2_{-47}. The clear register signal is made a trio of signal lines, each controlling 18 or 12 bits of the register.

For register transfers, shifts, arithmetic, etc., these parts must be further sub-divided in groups whose bit position treatment is similar or whose bit position treatment follows special, or additional considerations. The "special" bit positions are the sign, least and most significant bit positions.

The schematic logic of the A register shows it divided in six groups:



a_0 a_{-1} a_{-m} a_{-35} a_{-e} a_{-47}
2_0 2_{-1} 2_{-2} to 2_{-34} 2_{-35} 2_{-36} to 2_{-46} 2_{-47}
a_0 a_{-17} a_{-18} a_{-35} a_{-47}
A_{M1}
A_{M2}
A_E

Schematic Logic Diagrams

Figure 5.2-1 is the schematic logic drawing of A and A*. The card located in position J11021 contains the flip-flops storing the signs of A and A*. The a_{-1} bit is stored in card location J11022. The 33 bits of

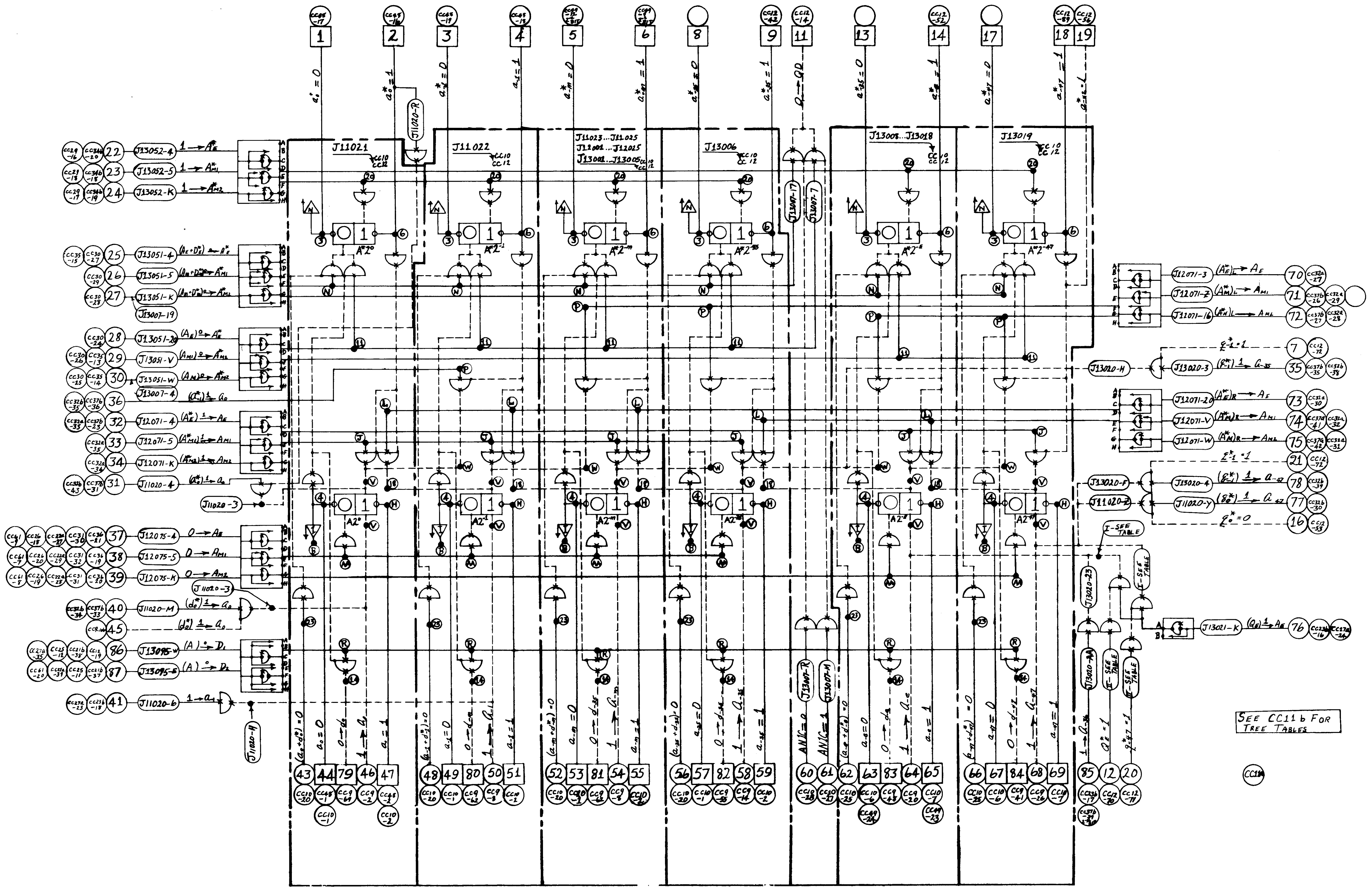


Figure 5.2-1 A Register Logical Diagram

a_{-m} are located in J11023 through J11025, J12001 through J12035, and J13001 through J13005. The remaining locations are as indicated. The locations were chosen so the register neons form a horizontal row in bit-positional order. The card locations are noted on the schematic logic at the top of the areas outlined by the heavy lines.

All physical terminal information is given either within the flattened ovals ⬭ or adjacent to a heavy, black dot, ●○ All other terminals are drawing reference symbols and refer to other drawings or tables.

Trees

Trees are shown in symbolic form. The actual wire routing is given in the tree tables associated with the schematic logic. Table 5.2-1 contains two of the 19 tree tables associated with Figure 5.2-1. The arrangement of the information within a table is in consecutive order; a_0 being on card 11021 and a_{-47} on card 13019.

Each of the three Clear A control signal lines, $0 \longrightarrow A_{M1}$, $0 \longrightarrow A_{M2}$, $0 \longrightarrow A_E$, drives tree circuitry. The development of $0 \longrightarrow A_{M1}$, for example, is through three emitter followers. Each emitter follower is connected to six inverters, each of which signals "Set 0" to a flip-flop of the register.

Figure 5.2-2 shows the circuitry symbolized by the first two illustrations. It is a composite drawing of elements from schematic logic, tree tables, and card schematics.

One of the control signals which clear A to zero is formed by the logic, IT3 $\cdot$ D84, on the card in position 13084. The numerals within the semicircles representing the transistor configuration give the physical location (on the card) of the transistor. The transistor forming the IT3 leg of the AND gate is transistor number 2 of module number 10. The type of module used is the C01 configuration. (Not shown is the logic that clears A_{M2} and A_E .)

The transfers between A and A* are clearly indicated on the schematic logic. The table for $(A)^* \longrightarrow A$ is given in Table 5.2-1.

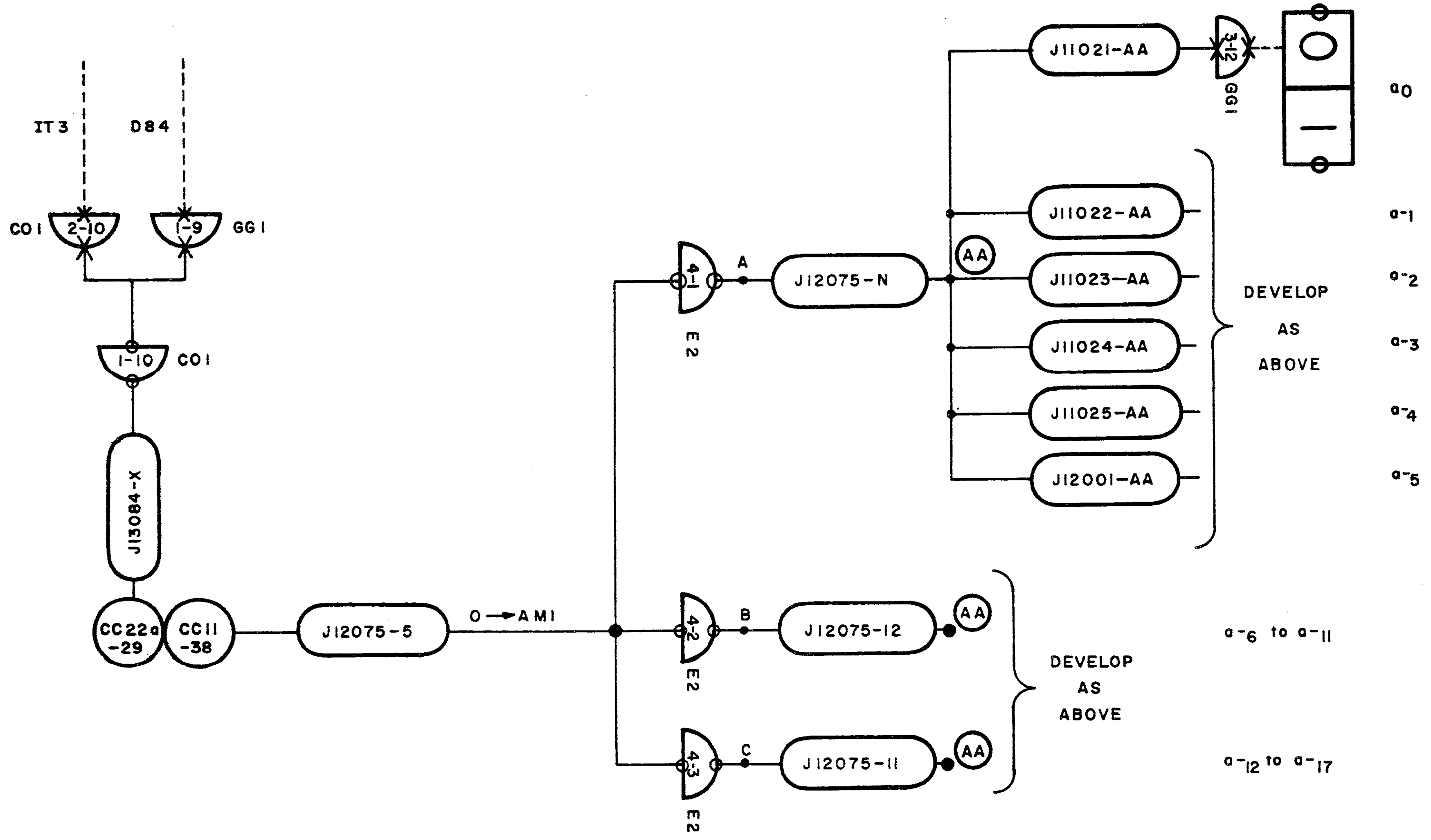


Figure 5.2-2 Tree (Partial) For 0 $\longrightarrow$ A

	$O \longrightarrow A$						
A	J12075-N	J11021-AA	J11022-AA	J11023-AA	J11024-AA	J11025-AA	J12001-AA
B	J12075-12	J12002-AA	J12003-AA	J12004-AA	J12005-AA	J12006-AA	J12007-AA
C	J12075-11	J12008-AA	J12009-AA	J12010-AA	J12011-AA	J12012-AA	J12013-AA
D	J12075-10	J12014-AA	J12015-AA	J12016-AA	J12017-AA	J12018-AA	J12019-AA
E	J12075-M	J12020-AA	J12021-AA	J12022-AA	J12023-AA	J12024-AA	J12025-AA
F	J12075-L	J13001-AA	J13002-AA	J13003-AA	J13004-AA	J13005-AA	J13006-AA
G	J12075-C	J13008-AA	J13009-AA	J13010-AA	J13011-AA	J13012-AA	J13013-AA
H	J12075-B	J13014-AA	J13015-AA	J13016-AA	J13017-AA	J13018-AA	J13019-AA

	$(A^*) \longrightarrow A$						
A	J12071-N	J11021-J	J11022-J	J11023-J	J11024-J	J11025-J	J12001-J
B	J12071-12	J12002-J	J12003-J	J12004-J	J12005-J	J12006-J	J12007-J
C	J12071-11	J12008-J	J12009-J	J12010-J	J12011-J	J12012-J	J12013-J
D	J12071-10	J12014-J	J12015-J	J12016-J	J12017-J	J12018-J	J12019-J
E	J12071-M	J12020-J	J12021-J	J12022-J	J12023-J	J12024-J	J12025-J
F	J12071-L	J13001-J	J13002-J	J13003-J	J13004-J	J13005-J	J13006-J
G	J12071-C	J13008-J	J13009-J	J13010-J	J13011-J	J13012-J	J13013-J
H	J12071-B	J13014-J	J13015-J	J13016-J	J13017-J	J13018-J	J13019-J

TABLE 5.2-1

Two-Step Transfer Method

Most inter-register transfers are performed by a two-step method to minimize logic. The receiving register is first cleared and then bits of opposite value to the clear condition are transferred. The register may be cleared to all zeros and ones transferred. Or it may be cleared to all ones and zeros transferred (see Figure 1.1-1):

$$0 \longrightarrow D^*; \quad (D) \xrightarrow{1} D^*$$

$$1 \longrightarrow A^*; \quad (A) \xrightarrow{0} A^*$$

The numeral above the arrow indicates the bit value transferred. An illustration of the transfer of a complement is:

$$1 \longrightarrow D^*; \quad (D)' \xrightarrow{0} D^* \ : \quad 1 \longrightarrow \text{ANIC}$$

Where d is equal to one, the corresponding bit position of d* is made zero. This transfer effects a ones complement and the carry-in flip-flop is set to 1 for the twos complement.

Jam Transfers

In some cases of transfer a one-step procedure is required. This is primarily due to time requirements, or transfers to only part of a register. Jam transfers will transfer the value of the bit position, either 0 or 1, to the corresponding position of the receiving register without prior clearing:

$$(SC) \longrightarrow SC^*$$

$$(MA) \longrightarrow X$$

One example of a jam transfer to part of a register is:

$$(JA) \longrightarrow D$$

This transfer involves only the address field, and possibly the J bit position, of half of the word in the D register. The remainder of the contents is not altered.

6. DETAILED LOGIC OF INSTRUCTIONS

6.1 Organization of Instruction Control

The initial activities of any instruction, as defined by the command coding are performed in the ITs. Therefore the ITs also perform general activities. All instructions start with the first four timings of IT. Some instructions are completed by IT4, others continue through IT8 and a third group goes from IT4 to the timings of another control register.

The following description of generalized activity of the ITs will indicate the manner of instruction performance control. The specific tasks for any instruction will be described under its family grouping.

<u>IT1</u>

Clear all storage units that need start the instruction in an initial condition:

$0 \longrightarrow$ AN2I

$0 \longrightarrow$ EO

$0 \longrightarrow$ FI

$0 \longrightarrow$ UF

$0 \longrightarrow$ AN1CE

$0 \longrightarrow$ SC

$0 \longrightarrow$ OVF (many instructions)

$0 \longrightarrow$ AN1I (except for equality jumps)

$0 \longrightarrow$ AN2C (most instructions)

Connect the inputs to AN2 so the effective address, when the instruction requires memory access, will be available in the adder network. Dependent upon the instruction coding and repeat control, the following are set to the desired states:

$PR \longrightarrow SW6$

$0 \longrightarrow AN2C$

Inh. AN2

if R = 0	if R = 1 and $\alpha\gamma = 0$	if repeat $\alpha\gamma = 1$
$0 \longrightarrow SW4$	$X \longrightarrow SW4$	$0 \longrightarrow PM$
$+ \longrightarrow PM$	$+ \longrightarrow PM$	$X \longrightarrow SW4$

Clear MA (except for jumps, Halt and CF)

$0 \longrightarrow MA$

Clear D

Clear JA (for jumps, and transfers through JA)

IT2

Program Control $(SW2)' \longrightarrow$ Mod 2

Transfer effective address to MA $AN2 \longrightarrow MA$

Set control registers to states required for the operations for the instruction (MI, AI , FI).

Set SC when involved.

IT3

Set PI for program operation to follow this instruction.

Clear receiving register for transfer instructions.

Clear D* for arithmetic and comparison operations.

Set JFF, if jump, for jump instructions not requiring a comparison as the jump condition.

IT4

Complete transfer for transfer instructions.

Transfer operand to D* for arithmetic or comparison exits:

IT4 $\longrightarrow$ END	for completed instructions as transfers, some special instructions, conditional jump instructions when not jumping, CF
IT4 $\longrightarrow$ AT1	fixed point arithmetic, shifts, Q jumps
IT4 $\longrightarrow$ FT1	arrangement of operands for floating point arithmetic

IT5 and IT6

Used by a few instructions only (certain index register, certain special and conditional jump instructions). No general activity is performed, only tasks specific to the above instructions.

IT7

Clear D for instructions which store result. Jump decision for jumps with comparison condition.

IT8

Transfer to D for store and access memory.

Set up address if jump, and program control.

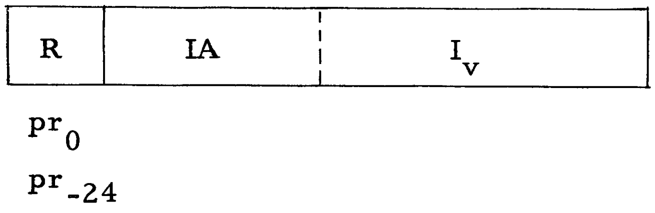
All instructions go from IT8 $\longrightarrow$ END except MAD and MSU.

6.2 Address Modification of the Index Registers

The effective address used in the performance of the instruction will be a choice of one of three numbers. They are addresses in the instruction, the contents of the selected index register, or the sum of the two. If the R bit of the instruction, pr_0 or pr_{-24}, (termed the "S bit" by programmers) is zero, the instruction alone furnishes the effective address. If the R bit is one, the effective address is the sum of the I_v field of the instruction and the contents of the selected index register, (X). In the repeat mode with repeat modification of the index register ($\alpha\gamma$ = 1), (X) alone is the effective address.

Instruction Word Address Field

The 16 leftmost bits of the instruction is divided into three fields, as indicated in Figure 6.2-1.



Address Fields

Figure 6.2-1

The size of the R field is fixed, one bit. The sizes of the other two are a function of the number of index registers in a specific S-2000 computer. For each computer the size of the IA and I_v fields are fixed by the wiring of the computer.

The size of the IA field is determined by the number of bits necessary to encode the selection of a particular index register. The number of index registers in a computer will generally be a power of two. The leftmost bit of the IA field will always be pr_{-1} or pr_{-25}. The field will extend to pr_{-5} or pr_{-30} for 32 index registers.

The I_v field size will be the remainder of the 16 bits. A computer with eight index registers, for example, will have a 3-bit IA field and a 12-bit I_v field. A 32 index register machine will have a 5-bit IA field and a 10-bit I_v field.

IA and I_v fields are known to programmers as the N and V fields, respectively, the R bit is also known as the S bit.

The transfer of these fields is determined by the R bit:

$R = 0 \Rightarrow (IA \text{ and } I_v) \longrightarrow MA$ via AN2

$R = 1 \Rightarrow (X) + I_v \longrightarrow MA$ via AN2

The logic of the transfer to MA is performed in IT1 and IT2.

IT1

Connect	$PR \longrightarrow SW6$
Not Subtract	$0 \longrightarrow AN2C$
	Inh. AN2
	and

If R bit = 0	If R bit = 1 and $a\gamma = 0$	If $a\gamma = 1$
$+ \longrightarrow PM$	$X \longrightarrow SW4$	$X \longrightarrow SW4$
$0 \longrightarrow SW4$	$+ \longrightarrow PM$	$0 \longrightarrow PM$

and

Clear MA $\quad 0 \longrightarrow MA$

IT2

$AN2 \longrightarrow MA$

Address or Parameter Modification

It can be seen that this logic permits modification of an address or the parameter of the shift instruction where the I_v field is the parameter or part of the parameter.

6.3 Memory Control and Use

6.3.1 Memory Cycle

The memory cycle consists of the operations necessary to transfer information to or from memory. The operations, in sequence, are read, write and post write disturb. The read operation is used to transfer from memory. Since the nature of the read-out process destroys the information in magnetic core storage, the read operation is also used to clear a memory location before transferring to memory. If it is desired not to lose the memory content due to a transfer from memory, the information is then transferred back (restored).

The write operation is used for restoring and transferring to memory. The post write disturb is required as part of the write operation due to the design of the memory. It has no logical significance.

Memory Timings

The sequence of tasks for memory operations are activated by the MTs. Unlike the rest of the computer control timings, these are generated by single shots. The single shots are connected to form a timing chain. With the exception of the interval between the end of the read and start of the write, timing proceeds uninterrupted through the sequence. As each single shot resets, it triggers the next in sequence.

Read Operation

The timings for the read are MT1 through MT5, in consecutive order. The read operation is initiated by:

$$\left[(\text{M10} \cdot \overline{\text{MI} = 3}) \vee \text{Begin IO Memory Cycle} \right] \cdot \text{Write Complete} \Longrightarrow \text{MT1}$$

MI = 3 is the write only operation for computer use of memory. The input-output use is always a full memory cycle. The Write Complete signal is normal only during a brief interval at the end of the write operation. It prevents the start of a new cycle before the memory drive currents of the previous cycle have ended. During MT4, the contents of V are transferred to D, PR or IOB. MT5 indicates the completion of the read.

Write Operation

Logic is required to initiate the write operation. There is an MT5 $\longrightarrow$ MT6 signal to immediately proceed to write, or the $\longrightarrow$ MT6 signal when an interval is required between read and write. During MT7's active period (D), (PR) or (IOB) are transferred to V. The method of writing requires zeros to be transferred to the inhibit drivers. Ones will be written in all cores of the word location except where inhibited by these drivers. The transfer is symbolically written as:

$$(D) \xrightarrow{0} V \text{ Inh.}$$

If all zeros are being written (clear V) the signal is $0 \longrightarrow V$ Inh.

6.3.2 Computer Use of Memory

Memory Control Register

Memory use by the central computer is organized by the Memory Control Register, MI. MI consists of three flip-flops. The decoded outputs of these in conjunction with the MTs (Memory Timings) result in the desired memory operations.

Memory Operations Classification

The required functions of memory use by the central computer can be collected into seven groups. Each group is represented by one state of the MI register and is given an MI number. The MI number corresponds to the binary value of the three flip-flops (MI = 1 ≡ MI = 001, etc.).

MI = 1	$(V) \rightarrow PR \rightarrow V$	Read a new instruction word to PR and restore in memory.
MI = 2	$(V) \rightarrow D$	Instructions with (V) as operand and storing result, or altering part of an instruction word in V. Memory will be cleared to 0 by this transfer and will be later written into with another state of MI.
MI = 3	$(D) \rightarrow V$	Replace in V. The results of the instruction replace the original contents of V by this write. The original contents were cleared out by the read during MI = 2.
MI = 4	$(V) \rightarrow D \rightarrow V$	Instructions involving (V) but causing no change in (V). V is restored by a write immediately following the read.

MI= 5 (register) $\longrightarrow$ V	Instructions that change (V) without use of the original contents. Read is used to clear V, immediately followed by the write to transfer (D)$\rightarrow$V.
MI= 6 CM	Clear Memory instruction only. Read and write 0.
MI= 7	Instructions not requiring (V) as an operand but storing result in V. The MI = 7 operation reads V to clear it. The subsequent write is under the control of MI = 3.

It will be noted that only one of the seven groups does not include a read operation, MI = 3.

MO

An eighth state of the Memory Control register, MI = 0 is used to signify whether or not the computer is using the memory. The MO number is active when the three flip-flops are reset to 0 and this situation occurs only when the computer is not using memory. The final timing, MT13, sends 0 $\rightarrow$MI, which indicates the completion of the memory operation for the computer. This is a common use of MO.

When possible, the computer operates in parallel with the memory operation. This is feasible for the MI = 7 state where a read operation is necessary preliminary to clear V prior to storing a new word. After sending 7$\rightarrow$MI in IT2 the computer continues to operate. However, before attempting to initiate the write, a test is made to insure the previous memory operation has ended, and MO is again active.

MO is strategically used in certain timings for this purpose. An example of the above-mentioned use would be the AQS instruction. The write memory operation to store the sum is initiated in IT8. However, IT7 $\cdot$ MO $\Rightarrow$ IT8 prevents a conflict.

The locations of MO are:

PT4 • MO • D122 • D123 • PO • P20 $\longrightarrow$ IT1
PT4 • MO $\Rightarrow$ PT4 off.

Wait until read completed, when transferring a new word to PR, before initiating the ITs.

IT7 • MO $\Rightarrow$ IT8

MO is the trigger for PT1 to prevent starting the read for a new instruction until the store operation of the previous instruction is completed.

Coordinating Memory and Computer Operations

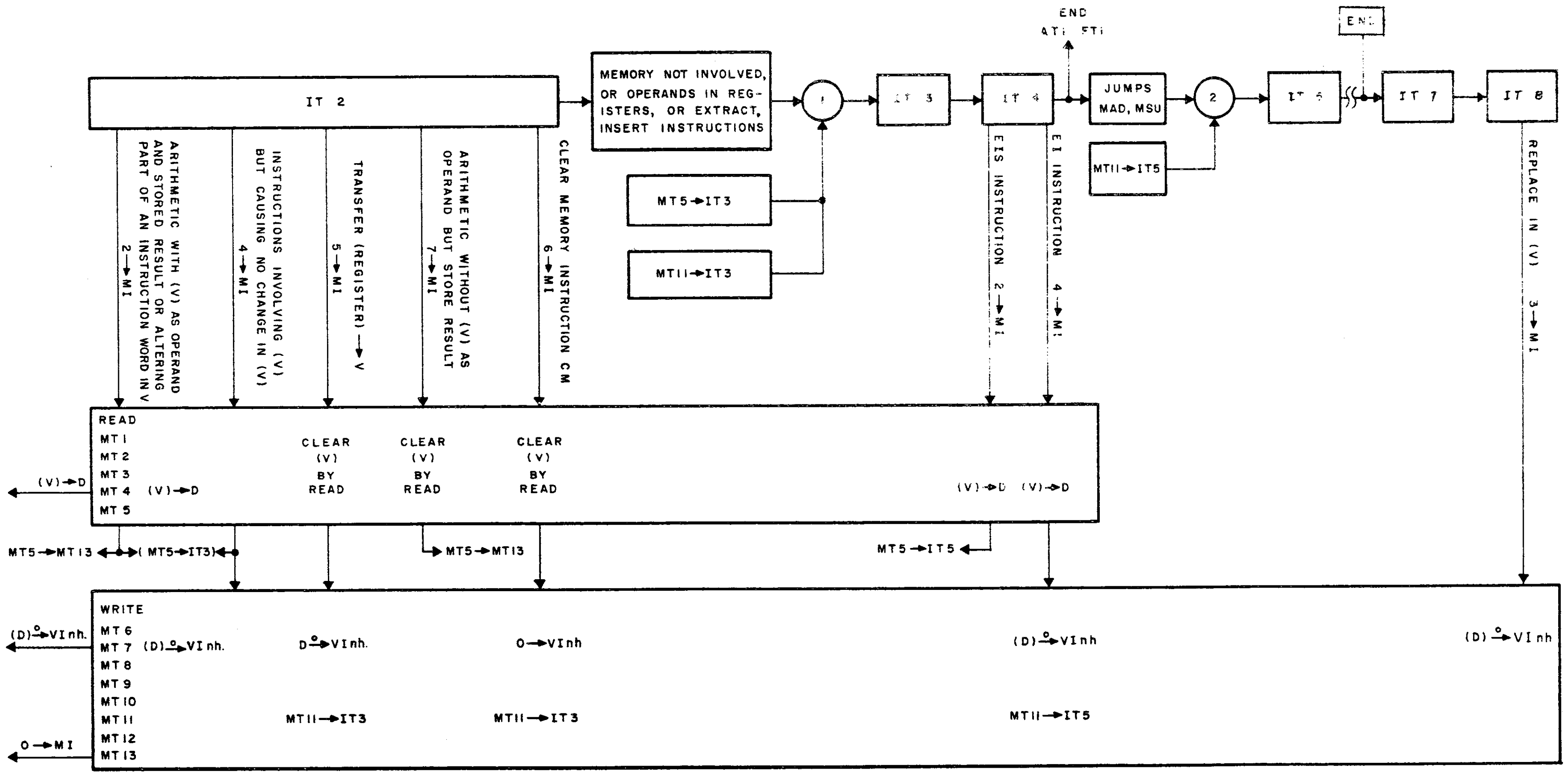
In most instances the computer requires a transfer from memory before proceeding. This is arranged as shown in Figure 6.3.2-I. IT2 is used by most instructions to initiate the read operations. If (V) are required before proceeding, the signal to activate IT3 will come from MT5 (Read Transfer Complete). If a write is required prior to proceeding, the signal will be MT11 $\Rightarrow$ IT3. MT11 denotes the point in the write operation when the computer can be released.

If the memory is not involved in the instruction, or not required at this time, IT2 $\longrightarrow$ IT3 directly.

The EI and EIS instructions are not prepared for the memory operations until IT4, unlike other instructions. Their memory action in IT4 is similar to the ones of IT2.

The replace operation, (D) $\longrightarrow$ V, is initiated in IT8. The computer proceeds without waiting to the end of the instruction. It will await the completion of the write operation at either PT1 or PT4.

The read-new-instruction-word operation (V) $\longrightarrow$ PR $\longrightarrow$ V is initiated in PT2 (unless the computer is to be interrupted by an input-output memory demand). The computer will await the completion of the transfer in PT4.


IT 2
MEMORY NOT INVOLVED, OR OPERANDS IN REGISTERS, OR EXTRACT, INSERT INSTRUCTIONS
1
IT 3
IT 4
END ATi FTi
JUMPS MAD, MSU
2
IT 5
END
IT 7
IT 8
MT5→IT3
MT11→IT3
MT11→IT5
ARITHMETIC WITH (V) AS OPERAND AND STORED RESULT OR ALTERING PART OF AN INSTRUCTION WORD IN V 2→MI
INSTRUCTIONS INVOLVING (V) BUT CAUSING NO CHANGE IN (V) 4→MI
TRANSFER (REGISTER) →V 5→MI
ARITHMETIC WITHOUT (V) AS OPERAND BUT STORE RESULT 7→MI
CLEAR MEMORY INSTRUCTION CM 6→MI
EIS INSTRUCTION 2→MI
EI INSTRUCTION 4→MI
REPLACE IN (V) 3→MI
READ
MT1
MT2
MT3
MT4
MT5
(V)→D
(V)→D
CLEAR (V) BY READ
CLEAR (V) BY READ
CLEAR (V) BY READ
(V)→D (V)→D
MT5→MT13
(MT5→IT3)
MT5→MT13
MT5→IT5
WRITE
MT6
MT7
MT8
MT9
MT10
MT11
MT12
MT13
(D)$\xrightarrow{o}$VInh.
(D)$\xrightarrow{o}$VInh.
D$\xrightarrow{o}$VInh.
0→VInh
(D)$\xrightarrow{o}$VInh
(D)$\xrightarrow{o}$VInh
MT11→IT3
MT11→IT3
MT11→IT5
0→MI

Computer Use of Memory

Figure 6.3.2-1

6.3.3 Memory Assignment

The usual amount of information transferred between magnetic tape and memory, or paper tape and memory, is a group of words. The tape word has a serial arrangement of eight characters within the word (or is treated as such) while a bit parallel storage of the word exists in the memory. The conversion is performed in a buffer register. In a transfer from tape to memory, for example, the characters are transferred serially from tape to a buffer storage. When a complete word is assembled it is transferred, bit parallel, to the memory. The tape control requests memory access for this purpose.

Memory access is required only for brief intervals during the transfer of a group of words. The transfer rate of characters between tape and buffer is much slower than the transfer rate of words between a tape unit and its buffer storage; it is feasible to allow the memory to be used by some other part of the system. Paper tape, magnetic tape and the central computer share memory access in this fashion during the execution of a tape transfer order. A transfer between memory and magnetic drum requires a continuous access to memory which will be described later.

The first three memory users operate asynchronously and require a memory assignment control system. The control assigns memory, as soon as it is available, to one requesting user in an established order of priorities. Simultaneous requests are responded to in this order:

1st - magnetic tape
2nd - paper tape
3rd - central computer

The control system is a series of interlocks to prevent access by more than one user at a time and establish the order of access. As the process of assignment takes a finite period of time (for switching) it is crucial to prevent the situation wherein the arrival of a higher-priority request during assignment to a lower-priority use results in assignment to two users simultaneously.

Input/Output Memory Assignment

To cope with this, the assignment process is performed in two steps. First, whenever memory is unused and available for access, all existing requests are noted. Secondly, while assignment is actually being made, no new requests are recognized. This two-part sequence is initiated by a request if memory is unused at that time, or at the end of each memory cycle.

The Assign Flip-flop recognizes requests only when it is in a reset condition (AFF = 0) and permits assignment of one recognized request when it is set. The logic is shown in Figure 6.3.3-1.

The requests, when recognized, are stored in flip-flops. The flip-flops remain set to one until the assignment has been made and memory access completed. The priorities are established by a group of AND gates. The signal from the 1's output of the Magnetic Tape FF will inhibit (when the flip-flop is set) the assign gates for paper tape and computer. The paper tape FF can inhibit the "Assign to C" gate.

The Magnetic Tape Request is a signal originating in magnetic tape control. With a multiplexing control for the magnetic tape units, the request signal exists as long as any of the tape units still require the memory.

The Paper Tape Request signal originates in paper tape control. It sets the MS* FF to 1, and must become inactive before action is taken on the request. This is required as the request signal is also used to perform some other logical function in paper tape that must be completed prior to the memory cycle.

The Computer Request signal is active while the MI Register is in any condition other than 0.

<u>Logic of Memory Assignment</u>

The Assign Flip-flop (AFF) tends to revert to the 1 state as:

$$AFF = 0 \cdot \overline{0 \longrightarrow AFF} \implies 1 \longrightarrow AFF$$

The customary practice is used of preventing the application of the output of a flip-flop until the switching signal has become normal.

Any request will initiate an assignment sequence, provided no current assignment exists (indicated by MTFF, PTFF, or CFF being

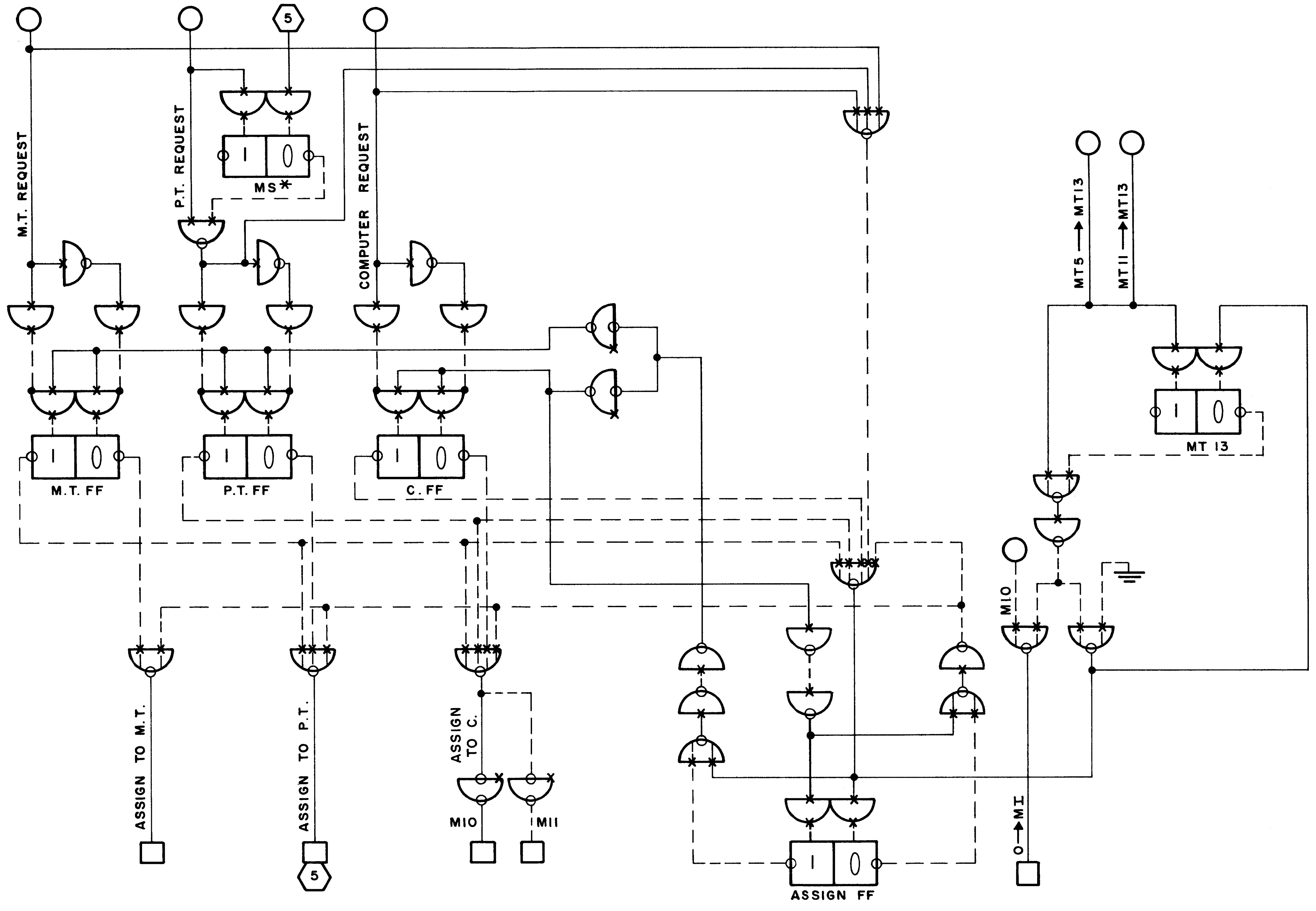


Figure 6.3.3-1 Memory Assignment Control

in the 0 state) nor an assignment sequence is in progress (AFF = 1).

$$\left[\text{MT Req. } v\ (\text{MS*} = 1 \cdot \overline{1 \longrightarrow \text{MS*}})\ v\ \text{ C. Req.}\right]$$

$$\cdot\ \text{MTFF} = 0 \cdot \text{PTFF} = 0 \cdot \text{CFF} = 0 \cdot \text{AFF} = 1 \cdot \overline{1 \longrightarrow \text{AFF}}$$

$$\Longrightarrow 0 \longrightarrow \text{AFF}$$

While AFF = 0 the requests are recognized and stored in the corresponding flip-flops:

$$\text{AFF} = 0 \cdot \overline{0 \longrightarrow \text{AFF}} \cdot \text{MT Req.} \Longrightarrow 1 \longrightarrow \text{MTFF}$$

$$\text{AFF} = 0 \cdot \overline{0 \longrightarrow \text{AFF}} \cdot \text{MS*} = 1 \cdot \overline{1 \longrightarrow \text{MS*}} \Longrightarrow 1 \longrightarrow \text{PTFF}$$

$$\text{AFF} \quad 0 \cdot \overline{0 \longrightarrow \text{AFF}} \cdot \text{C. Req.} \Longrightarrow 1 \longrightarrow \text{CFF}$$

and the second part of the assignment sequence initiated.

$$\text{AFF} = 0 \cdot \overline{0 \longrightarrow \text{AFF}} \Longrightarrow 1 \longrightarrow \text{AFF}$$

When AFF = 1 · $\overline{1 \longrightarrow \text{AFF}}$ the assignment will be based upon the order of priority as indicated as the three AND gates below the three flip-flops. The assign signals for tape are returned to input-output control for the generation of a signal to begin a memory cycle. M10 serves that purpose for the computer. M11 is used elsewhere to determine that the memory cycle in progress is not for the computer.

An assignment sequence is performed at the end of each memory cycle to clear the controls and recognize any requests that may have arisen. MT5 and MT11 are single shots that indicate the completion of read and write, respectively. In their active period they will set MT13 to 1.

$$\text{MT13} = 1 \cdot \overline{1 \longrightarrow \text{MT13}} \Longrightarrow 0 \longrightarrow \text{AFF},\ 0 \longrightarrow \text{MT13}$$

The MI register is cleared to 0 at this time, as well.

Magnetic Drum Access to Memory

The magnetic drum requires continuous access to memory during its transfers. Its consecutive word locations are available at the heads every 16 microseconds, less than two memory cycle periods. Insufficient time is available for another user to take memory between drum words. The drum is therefore given the lowest order of priority. It is not assigned the memory until all existing requests, from the other three users at the time of the drum request have been satisfied and their transfers completed. The computer is not released to perform other instructions in parallel with the drum transfer instruction. Thus no other requests can occur during the drum transfer period.

6.4 Transfers

This group of instructions provides for a transfer between any two data registers or between any data register and memory. The process of the transfer results in the transferred information existing in D as it is a junction for A, Q, and Memory transfers.

Due to the organizational similarities, the "clear register" instructions are made part of the transfer group. These clear the specified register or memory location to zero.

The instruction is performed in four timings, IT1 through IT4. The activities are listed below.

TRANSFER INSTRUCTIONS 001 J = 0

IT1

Start, Address $\longrightarrow$ MA PR $\longrightarrow$ SW6

0 $\longrightarrow$ AN2C

Inh. AN2

and

if R = 0	If R = 1	if $\alpha\ \gamma = 1$
0 $\longrightarrow$ SW4	X $\longrightarrow$ SW4	0 $\longrightarrow$ PM
+ $\longrightarrow$ PM	+ $\longrightarrow$ PM	X $\longrightarrow$ SW4

Clear D unless instruction is a 0 or 1 $\longrightarrow$ D

(D) $\longrightarrow$ elsewhere, or CA, CQ, CM

Clear MA 0 $\longrightarrow$ MA

$\longrightarrow$ IT2

IT2	trigger AN2CC	(D67)
	Program Control activity.	(SW2)' $\longrightarrow$ Mod 2
	Complete, Address $\longrightarrow$ MA.	AN2 $\longrightarrow$ MA
	Unless instruction is a (D) $\longrightarrow$ elsewhere, transfer from source to D (the junction point). This does not include CA, CQ and CM instructions.	(A) $\xrightarrow{0}$ D or (Q) $\xrightarrow{0}$ D or 4 $\longrightarrow$ MI [for (V) $\longrightarrow$]
	If transfer is $\longrightarrow$ V, start memory cycle to clear V then (D) $\longrightarrow$ V.	5 $\longrightarrow$ MI
	(Note: for CM instruction only use 6 $\longrightarrow$ MI for clearing.)	
	If memory is involved, memory control will provide the signal.	$\longrightarrow$ IT3
	Otherwise the computer proceeds without waiting.	$\longrightarrow$ IT3
IT3		
	Clear the receiving register	0 $\longrightarrow$ A or 0 $\longrightarrow$ Q
	Select next program control activity.	0, 1, 2 or 3 $\longrightarrow$ PI
		$\longrightarrow$ IT4
IT4		(Trigger AN2CC)
	Transfer to receiving register.	(D) $\longrightarrow$ A or (D) $\longrightarrow$ Q
		Inh. AN1
	End of instruction.	$\longrightarrow$ END

6.5 The Shift Instructions

This group of instructions enables A, D, or Q to be shifted right, A or Q to be shifted left, D to be circular right shifted, A and Q treated as a one register of double length to be shifted right or left.

The register contents can be treated as two types. In the numeric shift the sign position value is not changed. In the ordinary shift the 2_0 bit position is treated as any other bit. The D register can only be shifted to the right. The circular shift of D (SCD) is an ordinary shift with the additional feature of d_{-47} being transferred to d_0. Figure 6.5-1 shows the changes effected by a one-place shift of the various types.

The logic of the shift instructions permits a shift of from none to 63 places per instruction. However, shifts of more than 48 places are meaningless. The clear register instructions are preferable to a 48 place shift.

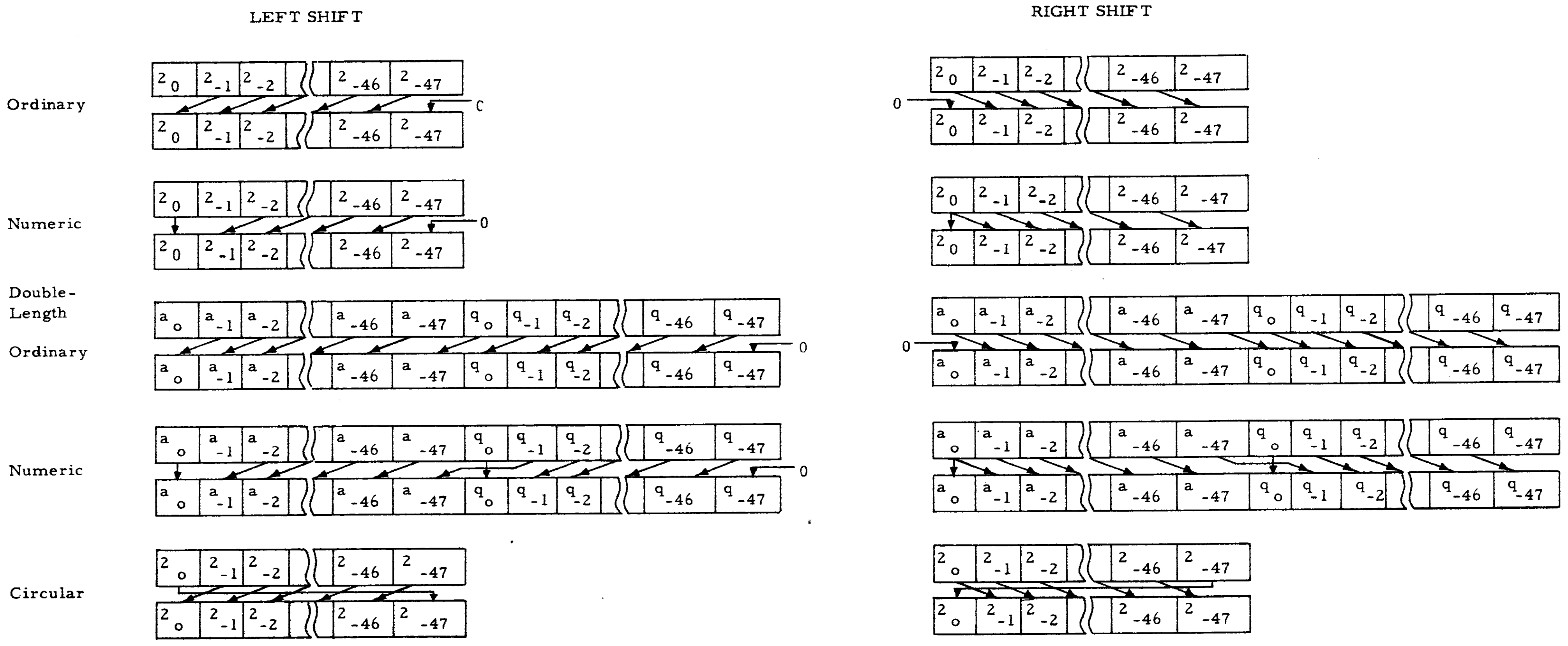
Organization of the Shift

The registers can be divided into two parts for the control of the shift. One part consists of those bits whose behavior for that direction of shift is standard. These are the bit positions not immediately adjacent to the sign position; 2_{-2} to 2_{-47} for a left shift and 2_0 to 2_{-46} for a right shift.

The other part consists of the variables. These are the bit position that is to be shifted into the 2_0 position; the use of 2_{-1} in left shift; the use of 2_{-47} in right shift. The variations are caused by the two treatments of the word, as ordinary or numeric; the choice of treating A and Q as one, double-length register; and the circular shift of D.

The logical details are developed in a straightforward manner from these considerations. The only exception is a minimizing of logic in the use of a_0 for a right shift. A moment's thought will reveal that the bit value of 2_{-1} after the shift is the same value as that of 2_0 before the shift regardless the type of shift.

The numeric right shift of a negative word in A should result in a bit value of 1 being placed in a_{-1}. For all right shifts of AR, a_0 is shifted to a_{-1} as a standard part.



S-2000 Shifts

Figure 6.5-1

The shift, per place, is performed by transferring the word to the star rank register, clearing the register to zero and then performing a shift transfer of ones from the star rank back to the register.

Using AR as an illustration of left shift, the standard part of the register is shifted left one place by the control signal $(A^*) \xrightarrow{L} A$, bit positions a^*_{-2} to a^*_{-47}. For the ordinary left shift, $a^*_{-1} \longrightarrow a^*_0$; a^*_0 is not transferred; a_{-47} having been cleared to zero will so remain. For the numeric left shift, $a^*_0 \longrightarrow a_0$; a^*_{-1} is not transferred. For the left shift of A, Q, either q^*_0 or q^*_{-1} is transferred to a_{-47} depending upon the type of shift. In the numeric shift, $q^*_{-1} \longrightarrow a_{-47}$, $q^*_0 \longrightarrow q_0$.

Table 6.5-1 lists the shift logic of the shift instructions. Most of this logic is used for the shifts during the arithmetic instructions as the desired activity in these cases is identical. The algorithm control setting for a shift operation is AI = 110. The activities listed in this table occur during AT4.

Shift Parameter

Shifting the desired number of places is performed by a series of one place shifts until the parameter has been attained. The parameter, stored in the Shift Counter is counted down, subtracted by one each shift until it reaches zero.

A Subtracting-one Binary Counter

When a subtracting binary counter has a fixed subtrahend of one, the logic of its operation can be based upon a few rules. The least significant bit is always complemented. If the least significant bit is one, before the subtraction, no other change is required. When the least significant bit is zero, before subtraction, then "one must be borrowed" from the next higher order. This borrowing complements the order. If it, itself, were zero before being complemented, a one must be borrowed from the next higher order, etc. Effectively, then, the rule becomes - if the least significant bit is zero, complement the least significant bit. And complement all consecutively higher orders up to and including the first order having a bit value of one prior to the subtraction.

	SLA	SRA	SLAN	SRAN	SLQ	SRQ	SLQN	SRQN	SLAQ	SRAQ	SLAQN	SRAQN
$(A^*) \xrightarrow{L} A$ $[a^*_{-2}$ through $a^*_{-47}]$	X		X						X		X	
$(A^*) \xrightarrow{R} A$ $[a^*_{0}$ through $a^*_{-46}]$		X		X						X		X
$(Q^*) \xrightarrow{L} Q$ $[q^*_{-2}$ through $q^*_{-47}]$					X		X		X		X	
$(Q^*) \xrightarrow{R} Q$ $[q^*_{-1}$ through $q^*_{-46}]$						X		X		X		X
$a^*_{-1} \longrightarrow a_0$	X								X			
$a^*_{0} \longrightarrow a_0$			X	X							X	X
$q^*_{-1} \longrightarrow q_0$					X				X			
$q^*_{0} \longrightarrow q_0$							X	X			X	X
$q^*_{0} \longrightarrow q_{-1}$						X		X		X		
$q^*_{0} \longrightarrow a_{-47}$									X			
$q^*_{-1} \longrightarrow a_{-47}$											X	
$a^*_{-47} \longrightarrow q_0$										X		
$a^*_{-47} \longrightarrow q_{-1}$												X

	SCD	SRD	SRDN
$(D^*) \xrightarrow{R} D$ $[d^*_{0}$ through $d^*_{-46}]$	X	X	X
$d^*_{-47} \longrightarrow d_0$	X		
$d^*_{0} \longrightarrow d_0$			X

Shift Logic of Shift Instructions

Table 6.5-1

If the counter has a star rank register associated with it, it is capable of extremely fast operation. The subtraction (in the sense of value changes in the bit positions) can be performed simultaneously. No "borrow propagation time" is required. The number is transferred to the star rank. The counter is not cleared. Then, only when a bit value changes, is the change transferred back to the counter. The subtraction can be performed in two timings: (the values of the numbers in the counter $\geqq$ zero).

1. $(SC) \longrightarrow SC^*$

2. $(SC^* - 1) \Longrightarrow \left[(SC^*_0)' \longrightarrow SC_0\right]$

$(SC^* - 1) \;.\; (SC^*_0 = 0) \Longrightarrow \left[(SC^*_1)' \longrightarrow SC_1\right]$

$(SC^* - 1) \;.\; (SC^*_0 = 0) \;.\; (SC^*_1 = 0)$
$\Longrightarrow (SC^*_2)' \longrightarrow SC_2$

and so on for each order of the shift counter.

The Shift Counter

The shift counter is a 6-bit counter, the smallest number of binary bits required to represent the value, 48. Figure 6.5-2 is the logic schematic of SC. This discussion is concerned with the features required for the shift instructions, the upper half of the diagram.

The Shift Counter has two ranks, SC and SC*. The value representing the number of places to be shifted is initially transferred from PR to SC. SC is counted down one, each shift of one place. The shift, per place, is done in the set of four ATs. In AT1, $(SC) \longrightarrow SC^*$. In AT3, $(SC^* - 1) \longrightarrow SC$. In AT4, the value of SC represents the number of places remaining to be shifted. In AT4 SC* contains SC's value plus one. When SC = 000000, SC* = 000001. The actual shift, per place, is of course completed in AT4.

The decision whether to shift again or end the instruction ($AT4 \longrightarrow AT1$, $AT4 \longrightarrow END$, respectively) is determined by whether SC = 000000. As a transfer into SC occurs during AT4, it is not feasible to sense SC at that time. SC* is sensed; does SC* = 000001? As SC*

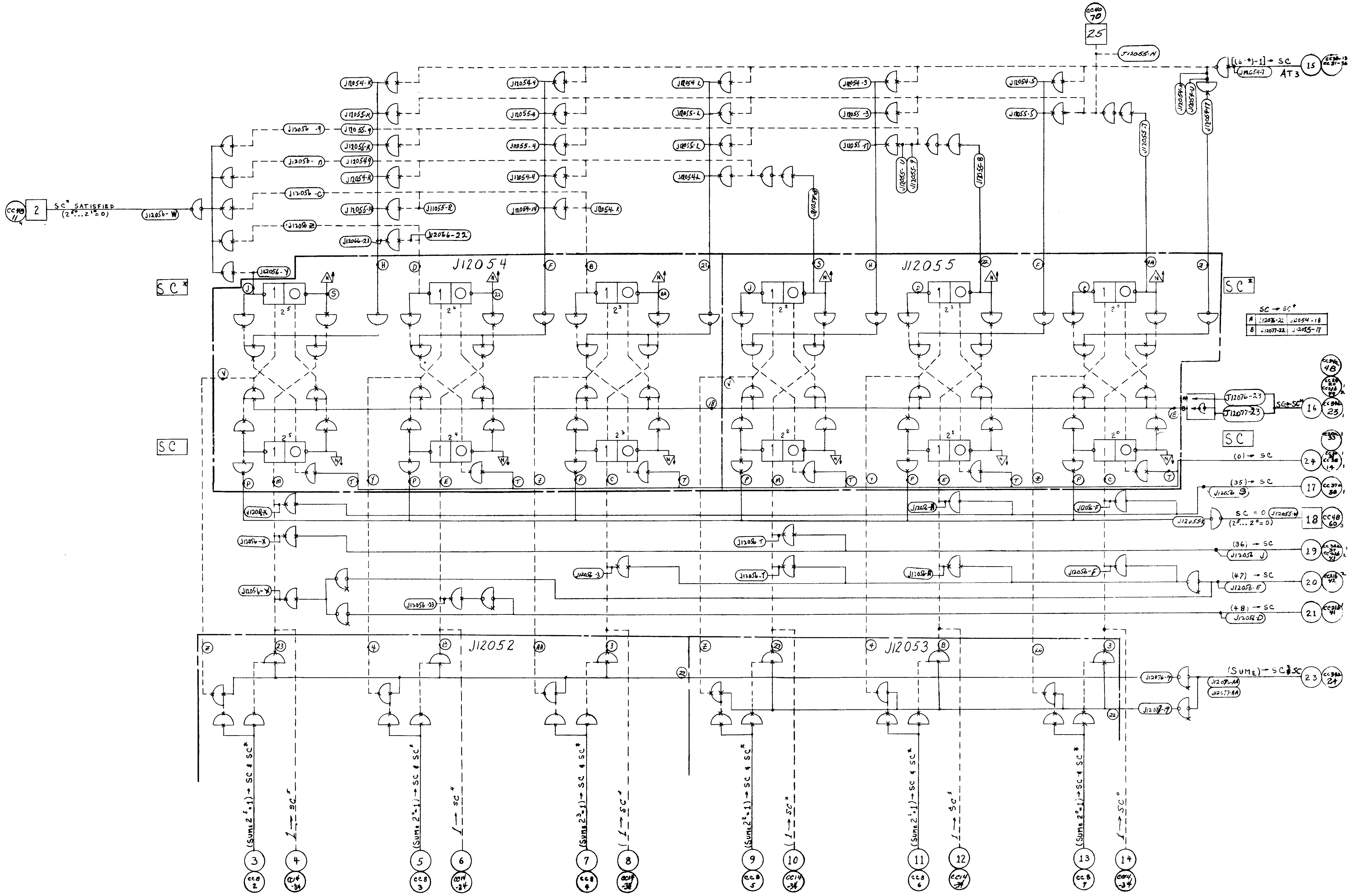


Figure 6.5-2 Shift Counter

is being counted down by a known decrement, the logic can be $(2^5, 2^4, 2^3, 2^2, 2^1, = 0)$. The value of 2^0 is insignificant as the first instance the above condition is realized during the count down is when the contents of SC* are 000001. The condition is named SC = SAT. (SC Satisified).

The SC = SAT. and SC $\neq$ SAT. signals control whether the computer proceeds to END or AT1, from AT4. There is another similar control signal that should not be confused with SC = SAT. This is the SC = 0 signal, the output of an AND gate that is active only when all six bits of SC are zero.

Shift Instructions, Sequence of Activity

IT1

clearings	$0 \longrightarrow$ AN1I
Prepare transfer to MA	$0 \longrightarrow$ OVF
	$0 \longrightarrow$ MA (not required for this instruction)
Prepare transfer of (I_V)	
and/or $(X) \longrightarrow$ SC	Inh. AN2
	$0 \longrightarrow$ AN2C

<u>S = 0</u>	<u>S = 1</u>	<u>RPT = 1</u>
$0 \longrightarrow$ SW4	$X \longrightarrow$ SW4	$X \longrightarrow$ SW4
PR $\longrightarrow$ SW6	PR $\longrightarrow$ SW6	PR $\longrightarrow$ SW6
$+ \longrightarrow$ PM	$+ \longrightarrow$ PM	$\underline{\alpha\gamma = 0}$ / $+ \longrightarrow$ PM ; $\underline{\alpha\gamma \quad = 0}$ / $0 \longrightarrow$ PM

IT2		trigger AN2CC
	Program control	$(SW2)' \longrightarrow$ Mod 2
	Set AI to shift operation	$6 \longrightarrow AI$
	Transfer shift parameter to counter.	$AN2 \longrightarrow SC$
	Transfer to MA (not required)	$AN2 \longrightarrow MA$
IT3		
	Program Control	Set PI state
IT4		Trigger AN2CC
	If shift parameter is zero end instruction.	$\longrightarrow$ END
	Otherwise begin shift operation.	$\longrightarrow$ AT1
AT1		
	Operational control.	$AI \longrightarrow AI*$
	Prepare to count down SC	$SC \longrightarrow SC*$
	Clear star rank (s) of involved register (s).	$(1 \longrightarrow A*) \vee (1 \longrightarrow Q*) \vee (0 \longrightarrow D*)$
AT2		trigger D48
	Transfer word to involved star rank (s).	$(A \xrightarrow{0} A*) \vee (Q \xrightarrow{0} Q*) \vee (D \xrightarrow{1} D*)$
AT3		
	Count down SC.	$(SC* - 1) \longrightarrow SC$
	Clear involved register (s)	$(0 \longrightarrow A) \vee (0 \longrightarrow Q) \vee (0 \longrightarrow D)$

AT4	Perform Shift	see Table 6.5-1
	If Shift Overflow is detected.	$1 \longrightarrow$ OVF
	Shift incomplete, recycle.	$\longrightarrow$ AT1
	Shift complete.	$\longrightarrow$ END

<u>Shift Overflow</u>

During a numeric shift of (A), or a numeric shift of (Q), it is important to detect a loss of a significant value bit in the highest order of the number. This value would be a one for positive numbers and a zero for negative numbers in the 2_{-1} bit position. In effect, the capacity of the register has been exceeded. This loss is termed Shift Overflow and can be predicted by the condition $2_0 \neq 2_{-1}$ in the star register.

The logic for Shift overflow is:

$$D46 \equiv AI = 110 \cdot \left[(0XX0 \; . \; a^*_0 \neq a^*_{-1}) \vee (10X0 \; . \; q^*_0 \neq q^*_{-1})\right]$$

where the bar code is from the command bits C_3 through C_0.

The existence of Shift Overflow will cause OVF to be set to 1 during the shift instruction.

The loss of bits from the 2_{-47} position during a right shift is inconsequential.

Q Jump - Circular Shift of Q

The Q jumps contain a one-place circular shift of that register, either left or right depending upon the instruction.

The shift is performed under the control of AI = 111 ($7 \longrightarrow$ AI in IT2 of the instruction). The control is straightforward and only one set of ATs is required.

From AT4 control passes to IT7, if jumping. Otherwise, control is returned to PI for the next instruction, index register modification or repeat register count-down.

AT1			
	Clear Q* for transfer	AI = 111	$1 \longrightarrow Q^*_M$
		AI = 111 · FP = 0	$1 \longrightarrow Q^*_M$
AT2			Trigger D48
	Transfer	AI = 111	$(Q_M) \xrightarrow{0} Q^*_M$
		AI = 111 · FP = 0	$(Q_E) \xrightarrow{0} Q^*_E$
AT3			
	Clear Q	AI* = 1X1	$0 \longrightarrow Q_M$
		AI* = 1X1 · FP = 0	$0 \longrightarrow Q_E$
AT4			
	Circular shift right	AI* = 111 · XX1X	$(Q^*_M) \xrightarrow{R} Q_M$
		AI* = 111 · XX1X · FP = 0	$(Q^*_E) \xrightarrow{R} Q_E$
		AI* = 111 · XX1X	$q^*_{-47} \longrightarrow q_o$
		AI* = 111 · XX1X	$q^*_0 \longrightarrow q_{-1}$

AT4 (Continued)

Circular shift left	AI* = 111 · XX0X	$(Q^*_M) \xrightarrow{L} Q_M$
	AI* = 111 · XX0X · FP = 0	$(Q^*_E) \xrightarrow{L} Q_E$
	AI* = 111 · XX0X	$q^*_{-1} \longrightarrow q_0$
	AI* = 111 · XX0X	$q^*_\theta \longrightarrow q_{-47}$
Exit	AI = 111 · FI = 000 · II = 0	AT4 $\longrightarrow$ END
If jumping	(Q jump) · JFF = 1	END $\longrightarrow$ IT7
If not jumping	(Q jump) · JFF = 0	END $\longrightarrow$ PT1 or PT2

6.6.1 The Add Instructions

With the A register as the augend, the add instructions offer a variety of options.

The A register always contains the augend. To it may be added the contents of a memory location or the Q register. The addition can be algebraic or the absolute value of (V) or (Q) may be used instead. The A register can be cleared to zero before the addition. The sum is always developed in A and may or may not be stored. Fixed or floating point addition can be performed.

The D register can also be added algebraically to A, but the only option is a choice of either fixed or floating point addition. The compound arithmetic instructions augment A by the product of (M) x (Q). (This instruction is described at the end of the multiply section, 6.6.3).

The above comprises a total of thirty-six instructions.

The logic of the basic, fixed point add, $(A) + (V) \longrightarrow A$, will be detailed first. Following this, the logic variations for the various options will be described.

The operations of the basic add instruction are twofold. The memory contents are transferred to D* during the ITs and then added to A during the ATs.

Logic of the Basic Add Instruction

$(A) + (V) \longrightarrow A$

IT1		
Clear controls	(timing alone)	$0 \longrightarrow$ E0, UF, SC, FI, AN2I, AN1CE
	Not equality jump nor JAZ.	$0 \longrightarrow$ AN1I
	ICOF = 0	$0 \longrightarrow$ OF
	Not repeat nor skip nor certain index instructions.	$0 \longrightarrow$ AN2C
SW4 input	Arithmetic instruction and R bit = 0.	$0 \longrightarrow$ SW4
	R bit = 1 and neither repeat nor skip instructions.	$X \longrightarrow$ SW4
SW6 input	$\overline{I20}$	PR $\longrightarrow$ SW6
PM input	Arithmetic instruction and neither repeat nor skip instructions and:	
	$\alpha\gamma = 0$	$+ \longrightarrow$ PM
	$\alpha\gamma = 1$	$0 \longrightarrow$ PM
Clear D	Memory operand and not CF (command fault).	$0 \longrightarrow$ D
Clear MA	Arithmetic instruction.	$0 \longrightarrow$ MA Inh. AN2
IT2		Trigger AN2CC
Set algorithm control for add.	Addition	$1 \longrightarrow$ AI
Clear SC*.	Addition	SC $\longrightarrow$ SC*
Transfer address.	Arithmetic instruction	AN2 $\longrightarrow$ MA
Start Memory Cycle.	Memory operand and result not stored.	$4 \longrightarrow$ MI

	Program control	I bit = 0 and not CF and not repeat instruction.	$(SW2)' \longrightarrow$ Mod 2 $MT11 \longrightarrow IT3$
IT3			
	Program control		0, 1, 2 or 3 $\longrightarrow$ PI
	Clear D* for transfer	Algebraic add:	$0 \longrightarrow D^*_M$, AN1C
		and fixed point	$0 \longrightarrow D^*_E$
		and floating point	$1 \longrightarrow D^*_E$
IT4			
			Trigger AN2CC
	Transfer	Algebraic add:	$(D_M) \longrightarrow D^*_M$
		and fixed point	$(D_E) \longrightarrow D^*_E$
		and floating point	$(D_E)' \longrightarrow D^*_E$, $1 \longrightarrow$ AN1CE
	Fixed point		$\longrightarrow$ AT1
	Floating point		$\longrightarrow$ FT1
AT1			
	Clear A* for sum	$\overline{AI = 1XX}$	$1 \longrightarrow A^*_M$
		$\overline{AI = 1XX} \cdot FP = 0$	$1 \longrightarrow A^*_E$
	Control	$AI = 001$	$(AI) \longrightarrow AI^*$
AT2			Trigger AN1CC
	Transfer sum	$AI = 00X$	$(A_M + D^*_M) \longrightarrow A^*_M$
		$AI = 00X \cdot FP = 0$	$(A_E + D^*_E) \longrightarrow A^*_E$
	If overflow	$AI = 00X \cdot AN1C_0 \neq AN1C_{-1} \cdot FP = 0$	$1 \longrightarrow 0F$

IT8 Trigger D 117

Transfer Sum. $(A) \xrightarrow{0} D$

Start Write. $3 \longrightarrow MI$

$\longrightarrow END$

Clear Add

This requires the A register to be cleared to zero before the algorithm.

$$IT3 \bullet D84 \Longrightarrow 0 \longrightarrow A$$

Absolute Value Add

The absolute value of the addend is used to increase the augend for a sum. This is done by arranging to have a positive number as the quantity transferred to D*, complementing if necessary.

IT3

Clear D*	Absolute value add and $d_o = 0$ and fixed point.	$0 \longrightarrow D^*_M$, $0 \longrightarrow AN1C$ $0 \longrightarrow D^*_E$
	Absolute value add and $d_o = 1$ and fixed point.	$1 \longrightarrow D^*_M$, $1 \longrightarrow AN1C$ $1 \longrightarrow D^*_E$

IT4

Transfer	D89	$(D_M) \longrightarrow D^*_M$
	D89 • I69	$(D_E) \longrightarrow D^*_E$
	D85	$(D_M)' \longrightarrow D^*_M$
	D85 • I69	$(D_E)' \longrightarrow D^*_E$

The balance of the logic is the same as the basic add instruction. It may be noted that when Q contains the addend for absolute add, the $(Q) \longrightarrow D$ transfer is completed by IT3.

Floating point add and subtract is described in sub-section 6.6.6.

6.6.2 The Subtract Instructions

The subtract instructions are identical in type to the add instructions. They also total thirty-six in number.

The essential difference between add and subtract is that in the latter case the twos complement of the subtrahend is transferred to D*. Subtraction is performed as complementary addition.

The difference in the subtract logic lies in IT3 and IT4 where $1 \longrightarrow$ D*, AN1C and $(D)' \longrightarrow$ D*.

An exception is absolute value subtract. Here a negative number must be transferred to D* in order to decrease the number in A.

D89 controls the transfer of the number unchanged; D85 causes the twos complement to be transferred to D*. The D and I number definitions in the Appendix explain this logic.

6.6.3 The Multiply Instructions

For multiplication, the multiplier must be located in the Q register prior to any multiply instruction. The various multiply instructions afford these options:

1. Fixed or floating point number multiplication.

2. The multiplicand may be located in either memory or the A register. It is transferred by the instruction to D.

3. Either the algebraic or the absolute value of the multiplicand may be used.

4. The product may be single-length, rounded or double length with the major half in A and the minor half in Q. The multiplier is presered in rounded multiplication and necessarily replace in double length.

5. The product may or may not be stored in memory. In the memory store, only (A) → V.

6. Two arithmetic operations may be combined in one instruction:

   MAD: (V) X (Q) (A) A

   MSU: (V) X (Q) (A) A

   Both of these instructions perform single length, rounded multiplication. Their logic is described in the section on special arithmetic.

Multiplication Method

Multiplication is achieved by the right shifting and accumulation of partial products.

```
    .101                                          .101
x    .01              is equivalent to           x.010
 -------                                          ----
     101                                          000
   000                                              ↘
 -------                                           000
  .00101                                          1010
                                                  ----
                                                  1010
                                                     ↘
                                                   1010
                                                  000
                                                  -----
                                                  01010
                                                     ↘
                                                 .00101
```

With binary numbers the decision for obtaining a partial product is whether or not to add the multiplicand to the esisting partial product (at the start the partial product is zero). For each order of a positive multiplier, 47 in the S-2000, the multiplican in D* is added to the partial product in A if the value of the bit of the multiplier (in Q) is 1. If the bit value is 0, $(A) + 0 \longrightarrow A^*$. The new partial product is shifted right one place, each time: $(A^*) \xrightarrow{R} A$

To facilitate sensing the value of the multiplier bit, the Q register is also shifted right one place, each time. The bit of the order used will then always be in the q_{-47} position. The shifting of Q also provides storage space for the minor part of the product in double length multiplication. In rounded multiplication, (Q) are circular shifted right.

Negative Multiplier

Due to the twos complement method of representing negative numbers, this multiplication process would not result in meaningful representation of the product if a negative multiplier were used. A positive multiplier is always used, deriving it, if necessary, by complementing:

$$(+D) \quad X \quad (-Q) \;=\; (-D) \quad X \quad (+Q)$$

The twos complement of D is readily obtained by $(D)' \longrightarrow D^*$, $1 \longrightarrow$ AN1C. The effective complementing of Q is done bit by bit, however, during multiplication to eliminate the time that would be required to add 1 to the ones complement of Q.

One rule for determining the bit by bit relationship between a number and its twos complement is:

> Beginning at the least significant bit and proceeding consecutively to the higher orders, the bit values of a number and its twos complement are the same up to and including the first bit with a value of 1. Thereafter, the bit values are opposite:

0.10101010000

1.01010110000

The bit by bit complementing of a negative multiplier is used to determine whether or not add (D*) to (A). This determination can be done by comparing q_{-47} (Q is shifted right one place each time) with a flipflop (named "QC"). Until the first "1" has been reached, the value of q_{-47} should be "1" for $(A) + (D^*) \longrightarrow A^*$. Thereafter, the value of q_{-47} should be "0" for that addition. The outline of the method (for a negative multiplier) is:

(a) Initially set QC to 1

(b) $(q_{-47} = QC) \Longrightarrow (A) + (D^*) \longrightarrow A^*$

(c) $(q_{-47} = 1) \Longrightarrow (0 \longrightarrow QC)$

(The twos complement of (D) is added to (A) if the multiplier is originally negative).

The actual logic used for this process in fixed point multiplication produces the " $|q| = 1$" signal which will cause D* to be added to A.

$$(|q| = 1) = \left[(q_0 = 0 \ \mathrm{v} \ QC = 1) \cdot (FP = 0 \cdot q_{-47} = 1)\right]$$
$$\mathrm{v} \ \left[FP = 0 \cdot q_0 = 1 \cdot q_{-47} = 0 \cdot QC = 0\right]$$

(Note: $FP = 0$ indicates fixed point arithmetic.)

If the multiplier is negative, $(D)' \longrightarrow D^*$ during IT2 and IT3.

Shift Counter Use

The 47 addition cycles for fixed point multiplication are established by the use of SC which is set to 47 in IT2 and counted down each AT3. The sets of AT timings will be repeated until SC SAT. SC SAT is active when SC* reaches 000001, but is not used until (SC* -1) SC and therefore SC 000000.

Product Sign

There are two exceptions in the S-2000 to the customary rule that the product sign is determined by the signs of the operands.

One exception is when the multiplier is zero and the multiplicand negative. The rule that unlike operand signs cause a negative product does not apply. Zero is defined in the S-2000 as a positive number. This case of unlike signs (-N) x (+0) results in a positive product, namely zero.

The other exception develops from rounding in single length multiplication. If the negative product is sufficiently large (close to zero), rounding will make it zero. Here again the sign changes. For example:

$$-2^{-24} \times 2^{-24} = -2^{-48}$$

The largest, representable, fixed-point negative number is -2^{-47} (forty-eight ones). If rounding, -2^{-48} is increased to the next larger, representable number. This is zero, which of course is positive by definition.

The S-2000 rule for product sign can be summarized by stating that the product is positive unless the effective multiplicand is negative AND the effective multiplier is not zero AND rounding does not make the product zero.

The effective multiplier is always positive, as previously explained. The effective multiplicand is (D*). (+ D) x (-Q) is changed to effectively become (-D*) x (+Q). (-D) x (-Q) is likewise changed to effectively become (+ D*) x (+ Q). A negative (D*) therefore implies a negative product, unless either of the two exceptions, described above, exists.

The logic for forming the positive sign is :

$$AT2 \cdot a_o = 0 \cdot \overline{\left[q = 1 \cdot d^*_o = 1 \cdot c_o = 0 \right]} \Longrightarrow \text{positive sign}$$

If the effective multiplier is negative ($d^*_0 = 1$), AND not zero ($|q| = 1$) AND rounding does not make the product zero ($c_0 = 0$), the sign is negative.

If the multiplicand is negative and the multiplier not zero, $AN1C_0 = 1$ indicates the rounded product has become zero instead of a negative number.

The quantity, -2^{-48}, would be represented by the binary number: $2^1 - 2^{-48}$. To this the round factor is added, effectively 2^{-48}. The sum is 2^1, indicated by $AN1C_0 = 1$.

The positive sign, where the multiplier is zero, would be indicated by the absence of an active $|q| = 1$ signal during the algorithm.

The a^*_{+1} flipflop is used to prepare the sign bit for a_0. In every AT1: $1 \longrightarrow a^*_{+1}$

In AT2, $0 \longrightarrow a^*_{+1}$, by the logic previously cited. And in AT4, $(a^*_{+1}) \longrightarrow a_0$. During the last AT4, also $(a^*_{+1}) \longrightarrow q_0$ during double-length multiplication.

Multiplier is Zero

The exception is when the multiplier is zero, which by definition is a positive number in the S-2000. The logic for the usual sign transfer in multiplication is:

$$AT4 \cdot QC = 0 \longrightarrow (a^*_{+1} \longrightarrow a_0)$$

Q will not be reset unless one of the bits of the multiplier has a value of one. The product will be zero, and $a_0 = 0$ if $(Q) = 0$, as A is initially cleared and no additions of $(A + D^*)$ subsequently occur.

Multiplier is Minus One

Here, similarly to $(Q) = 0$, the 47 add cycles will leave $(A) = 0$. One additional add cycle is required, known as "force add", which effectively will transfer $(D)'' \longrightarrow A$.

The definition of $(Q) = -1$ is that in the 47th add cycle ($SC = SAT$), the sign is negative ($q_0 = 1$), all previous multiplier bits have been zero ($QC = 1$), and the most significant bit, which has been shifted to q_{-47} by now, is also zero ($q_{-47} = 0$):

$$AT3 \cdot A2b \cdot V8 \cdot V17 \cdot FP = 0 \Longrightarrow (0 \rightarrow AI)$$

The exit from AT4 $\rightarrow$ END is prevented when (Q) = -1. Instead the force add cycle, AI = 000, is performed before the instruction ends.

Both Operands are -1

During the force add cycle, (D*) = 0.111 ..... 1 and ANIC = 1. This results in overflow as $ANIC_0 \neq ANIC_{-1}$. The only time multiplication in the S-2000 can result in overflow is for (-1) X (-1) = 1.

$$AT2 \cdot AI = 00X \cdot ANIC_0 \neq ANIC_{-1} \Longrightarrow (1 \rightarrow OVF)$$

AI Control

The states of AI used for multiplication are AI = 010 for double length and AI = 011 for single length multiplication. AI is changed to AI = 000 for the force add cycle.

Double Length Multiplication

The A register is initially cleared to zero; then (D) X (Q) $\rightarrow$ A, Q with the major or more significant half of the product appearing in A. The multiplier is of course lost. The sign of the product is stored in both a_0 and q_0.

Single Length Multiplication

This arithmetic produces a 47-bit, rounded product, with sign in the A register. The multiplier in Q is circular shifted right, in the numeric sense, during multiplication and is not lost.

Rounding off is a process used when the accuracy of the result is limited by reducing the number of digits allotted to represent the answer. The product is available as a 94-bit number. In single length, only 47 place accuracy is maintained. The rule for rounding is that the least significant bit of the shortened number should be a value closest to the unreduced size number. The choice is between two consecutive values of the L.S.D. of the shortened number (0 and 1 in the binary system). If the difference between the lower value and the actual number is greater than the difference between the higher value and the actual value, the higher value is used. If not, the lower value is used. To demonstrate with 3 and 6 digit binary numbers:

Difference between .100 and .101 is .001000

.100100 is rounded to .101

.100011 is rounded to .100

Rounding is achieved by adding a one to the part of the full length number that is subsequently dropped. The one is added to the most significant digit of the dropped part, in the S-2000, the 2_{-48} bit of the product.

If a one is added to the proper bit position prior to, rather than after, the multiplication, there is no necessity to temporarily store the double length product in A and Q. The minor half of the product can be dropped out of A as it is shifted right. The multiplier in Q can be preserved.

Prior to the multipication, the bit position that will eventually contain the 2^{-48} bit of the product is the a_{-1} position. In single length, rounded multiplication:

$$IT2 \Rightarrow (0 \longrightarrow A)$$

$$IT3 \Rightarrow (1 \longrightarrow a_{-1})$$

The register contains 0.1 rather than zero as the multiplication algorithm starts.

Absolute Value (of Multiplicand)

The sign of the product is the sign of the multiplier. It may be necessary to complement D to obtain the required representation of the product.

If (+Q) and (-D), or (-Q) and (+D) it is necessary to complement by $(D)' \longrightarrow D$ and $1 \longrightarrow AN1C$. This will result in a product of the same sign as Q, with the number in the proper form.

If the signs of (Q) and (D) are alike, the multiplicand already has the proper form and is transferred, unaltered, to D*.

If (-Q), the multiplier is effectively bit-by-bit complemented during the multiplication.

<u>Logic for Fixed Point Multiply Instructions</u>

IT1

General clearing of controls:	$0 \rightarrow$ AN1I, AN2I, EO, UF, SC
	$0 \rightarrow$ OVF (if ICOF not effective)
	$0 \rightarrow$ MA, AN2C, FI, AN1CE
Address modification, if any, and connect inputs to AN2.	
Clear D, if memory operand:	$0 \rightarrow$ D
if (A) operand:	$1 \rightarrow$ D

IT2

	(SW2)' $\rightarrow$ Mod 2
	AN2 $\rightarrow$ MA
Memory access, if required,	
if product stored:	$7 \rightarrow$ MI
if product not stored:	$4 \rightarrow$ MI
If operand in A:	A $\xrightarrow{o}$ D
Set Algorithm Control,	
double length product:	$2 \rightarrow$ AI
rounded product:	$3 \rightarrow$ AI
Set SC,	
fixed point:	$47 \rightarrow$ SC
Set QC	$1 \rightarrow$ QC

IT3

Set PI for program activity following instruction:	1, 2, or 3 $\rightarrow$ PI
Clear A to zero for initial partial product:	0 $\rightarrow$ A
Clear D* for transfer from D	
for (D) $\rightarrow$ D*	0 $\rightarrow$ D* and 0 $\rightarrow$ AN1C
for (D)' $\rightarrow$ D*	1 $\rightarrow$ D* and 1 $\rightarrow$ AN1C

IT4

	Inh. AN1
Transfer to D*	
not complementing:	(D) $\rightarrow$ D*
complementing:	(D)' $\rightarrow$ D*
If rounding, enter factor:	1 $\rightarrow$ a_{-1}
Fixed point	$\rightarrow$ AT1
(Floating point	$\rightarrow$ FT1)

AT1

Control	(AI) $\rightarrow$ AI*
Prepare count-down	(SC) $\rightarrow$ SC*
Clear A*	1 $\rightarrow$ A*, a^*_{+1}
Clear Q*	1 $\rightarrow$ Q*

AT2

	Trigger D48
Add on this cycle,	
(AI 01X · $\|q\|$ = 1) or force add, AI = 00X:	(A D*) $\xrightarrow{o}$ A*

Do not add this cycle,

$(AI = 01X \cdot |q| = 0)$ $\qquad (A) \xrightarrow{o} A^*$

Prepare Shift $\qquad (Q) \xrightarrow{o} Q^*$

If overflow during force add, (-1) X (-1) : $\qquad 1 \rightarrow OVF$

Positive sign $\qquad a_o = 0 \cdot \overline{\left[A11 \cdot d^*_o = 1 \cdot c_o = 0 \right]} \Rightarrow 0 \rightarrow a^*_{+1}$

AT3

Change QC when reaching first multiplier bit equal to 1;

$(q^*_{-47} = 1 \cdot FP = 0) \quad v$

$(q^*_{-35} = 1 \cdot FP = 1)$ $\qquad 0 \rightarrow QC$

Count down SC $\qquad (SC^* -1) \rightarrow SC$

Clear for shift $\qquad 0 \rightarrow A, \quad 0 \rightarrow Q$

Last normal cycle, set controls for further operations, if necessary:

Force add next, multiplier is minus one; $\qquad 0 \rightarrow AI$

$\left[SC = SAT \cdot QC = 1 \cdot q_o = 1 \right]$

$\left[(FP = 0 \cdot q^*_{-47} = 0) \quad v \quad (FP = 1 \cdot q^*_{-35} = 0) \right]$

AT4

Shift A right $(AI^* = 01X)$, $\qquad (A^*) \xrightarrow{R} A$

Double length, fixed point $\qquad a^*_{-47} \rightarrow q_{-1}$

Force add cycle $(AI^* = 00X)$, do not shift $\qquad A^* \rightarrow A$

Shift Q right	$(Q^*) \xrightarrow{R} Q$
Rounded, or double length and SC$\neq$SAT.	$q^*_0 \longrightarrow q_0$
Rounded (circular shift of Q) if $q^*_{-47} = 1$	$1 \longrightarrow q_{-1}$
Transfer product sign to A.	$a^*_{+1} \longrightarrow a_0$
Double length, last cycle, transfer product sign to Q.	$a^*_{+1} \longrightarrow q_0$
Perform another add cycle, (SC$\neq$SAT or Q = -1)	$\longrightarrow$ AT1
Recycling (SC $\neq$ SAT)	$\longrightarrow$ AT1
Last cycle (SC = SAT) and multiplier is neither 0 nor 1 $\overline{(QC = 1 \cdot q_0 = 1)}$	$\longrightarrow$ END
Force add $(QC = 1 \cdot q_0 = 1)$	$\longrightarrow$ AT1

(It may be noted that the force add logic can be redundantly active before the 47th cycle with the other "$\longrightarrow$ AT1 " signal.)

Multiply and Add, Subtract Instructions

These two instructions, MAD and MSU are a combination of MMR $[(V) \times Q \rightarrow A$, rounded$]$ followed either by AD or SD, $[(A) \pm (D) \rightarrow A]$. The logic is essentially the same. However, the MAD, MSU instructions require additional logic in IT4 through IT8 for control purposes and the extra operation that is required.

The extra operation is necessary as the number that will be added to, or subtracted from, the product in the A register at the start of the instruction. It must be relocated before the multiply algorithm as the A register is required for the product.

The procedure used is to store this number in D. This can be done after the multiplicand has been transferred from memory, through D to D*. IT4-8 timings are used for the $(A) \rightarrow D$ transfer.

Identification is necessary to distinguish the two instructions from the other individual multiply, add, or subtract instructions. The various phases within the instruction must also be distinguished, primarily the ITs.

The II register is used for this purpose:

II = 0	ITs for activity prior to multiplication. Go from IT4 to IT5, $1 \rightarrow$ II in IT5.
II = 1	Go from AT4 or FT4 (end of mutliplication) to IT1 for activities prior to add, subtract. Go from IT4 to AT1 or FT1.
AT1	When in add, subtract algorithm (AI= 001), $0 \rightarrow$ II. This will cause exit from AT4 or FT4 to END.

Logic of MAD, MSU

The logic listed here is that which is used in addition to the existing logic for the rounded multiplication and the add, subtract A to D instructions. Aside from the additions and exceptions noted below, the logic is identical. The listing below should be read in conjunction with the logic for multiply previously given in this sub-section, and the logic for add, subtract in sub-sections 6.6.1 and 6.6.2.

IT1	II= 0 Same as MMR.		
IT2	Same as MMR except (SW2)' $\longrightarrow$ Mod 2 is NOT done at this time.		
IT3	Same as MMR except A is not cleared at this time.		
IT4	Same as MMR except the rounding factor ($1 \longrightarrow a_{-1}$) is not transferred at this time.		
	(MAD v MSU) • II =0		IT4 $\longrightarrow$ IT5
IT5			Trigger AN1CC
	Clear D for transfer.	I9	$1 \longrightarrow D$
	Control	I9	$1 \longrightarrow II$
	(Note: MAD, MSU are the only arithmetic instructions using IT5-6. Therefore I9, which is the 3-bit command coding for **arithmetic instructions IXX, identifies them in these ITs.)**		
IT6			
	Transfer operand for second part of instruction.	I9	$(A) \longrightarrow D$
IT7			
	Clear A as initial partial product.	D108 •V32	$0 \longrightarrow A$
IT8			Trigger D117
	Rounding factor.	I60	$1 \longrightarrow a_{-1}$
	Transfer multiplier exponent for addition.	I60 • I70	$(Q_E) \longrightarrow A_E$
	Fixed point, algorithm	I60 • I69	$\longrightarrow$ AT1
	Floating point, exponent addition.	I60 • I70	$\longrightarrow$ FT1

AT4

Fixed point algorithm is completed:
AI* = 01X • SC = SAT • $\overline{V17}$ • FI =000 • II = 1 $\Longrightarrow$ AT4 $\longrightarrow$ IT1

FT4

Floating point is completed as:

Product is normalized.	FI* =100 • $a^*_0 \neq a^*_{-2}$ • II =1	$\Longrightarrow$ FT4 $\longrightarrow$ IT1
Product is zero	FI* =100 • SC= SAT • II=1	$\Longrightarrow$ FT4 $\longrightarrow$ IT1
Product overflow corrected.	FI* =101 • II= 1	$\Longrightarrow$ FT4 $\longrightarrow$ IT1
Product is effectively zero as exponent underflows. Do not do multiply algorithm.	FI* =001 • UF = 1 • II= 1	$\Longrightarrow$ FT4 $\longrightarrow$ IT1

Note: An exponent fault will not be acted upon by the computer until the end of the instruction, subsequent to the add or subtract. The IT, AT and FT operations with II = 1 use the same logic as that of the AD and SD instructions.

6.6.4 Divide Instructions

Transac S-2000 can perform eight divide instructions, four each with fixed and floating point numbers. The four types are single or double length dividends; the quotient resulting from either may or may not be stored in memory. The dividend must be located in A or A, Q prior to the divide instruction. The divide instruction always transfers the divisor from memory to D. If the quotient is stored, it replaces the divisor in the memory location. The remainder is located in A.

The results of division are cogently explained by a Note from the Programming R and D Department which follows.

TRANSAC S-2000 DIVISION ALGORITHM

The TRANSAC S-2000 division algorithm divides a number x in A or in A and Q by a number y in memory and produces a quotient in Q and a remainder in A. The sign of the remainder is always the same as the sign of x. For reasons of speed and consistency of computer logic, the quotient when negative is a 1's complement rather than a 2's complement number. In most calculations this is simply equivalent to stating that the number in the Q register is always the quotient rounded down by truncation of the remainder. In some cases it is necessary to know exactly what is in Q and what is in A after a division and this Note attempts to clarify this point.

Consider the division $x \div y$. We can write

$$\left|\frac{x}{y}\right| = q + \frac{r \times 2^{-47}}{y}$$

This equation defines q and r as positive numbers. Let us assume that $r \neq 0$, i.e., the division is not exact. After the division is complete the contents of Q and A will be as follows:

x	y	(Q)	(A)
+	+	q	r
-	-	q	2-r
+	-	$(2-2^{-47})-q$	r
-	+	$(2-2^{-47})-q$	2-r

The number $(2-2^{-47})-q$ is the one's complement of q and can be obtained by simply changing all 0's to 1 's and all 1's to 0's in the binary representation of q.

The case $r = 0$ is special because $2-0$, i.e., negative zero, is not a valid negative number. When $r = 0$ we therefore write

$$\frac{|x|}{|y|} = q \quad \text{if } x \text{ is positive}$$

$$\frac{|x|}{|y|} = q - 2^{-47} + \frac{d \times 2^{-47}}{d} \quad \text{if } x \text{ is negative(where } d = -|y|).$$

Thus when division is exact and the remainder is 0, the Q and A registers will appear after division as follows:

x	y	(Q)	A
+	+	q	0
-	-	$q-2^{-47}$	y
+	-	$(2-2^{-47})-q$	0
-	+	$(2-2^{47})-(q-2^{-47})$	-y
		$= 2-q$	

The above statements are also true in floating point division. The constant 2^{-47} must, of course, be replaced by 2^{-35}. However, in case of floating point division the quotient is normalized by continuing the division process and adjusting the exponent. When floating point division is completed the exponent bits of A will contain the same exponent as the 12 exponent bits of Q.

Examples

Consider the division $\frac{1}{4} \div \frac{3}{4} = \frac{1}{3}$ Here

$|x| = 0.0100\ldots$
$|y| = 0.1100\ldots$
$q = 0.010101\ldots010$
$r = 0.1000\ldots$

$\underline{x}$	$\underline{y}$	$\underline{(Q)}$	$\underline{(A)}$
$+\frac{1}{4}$	$+\frac{3}{4}$	0.0101...010	0.1000...
$+\frac{1}{4}$	$-\frac{3}{4}$	0.0101...010	0.1000...
$+\frac{1}{4}$	$-\frac{3}{4}$	1.1010...101	0.1000...
$-\frac{1}{4}$	$+\frac{3}{4}$	1.1010...101	1.1000...

Now consider the exact division $\frac{1}{4} \div \frac{1}{2} = \frac{1}{2}$

$|x| = 0.0100...$

$|y| = 0.1000...$

$q = 0.1000...$

$\underline{x}$	$\underline{y}$	$\underline{(Q)}$	$\underline{(A)}$
$\frac{1}{4}$	$\frac{1}{2}$	0.1000...000	0
$-\frac{1}{4}$	$-\frac{1}{2}$	0.0111...111	1.100...000
$+\frac{1}{4}$	$-\frac{1}{2}$	1.0111...111	0
$-\frac{1}{4}$	$+\frac{1}{2}$	1.1000...000	1.100...000

Division Method

Division is achieved by a series of subtractions of the divisor from the dividend accompanied by left shifts of the difference. The quotient is developed bit-by-bit, as the following example demonstrates. Positive four-bit numbers and sign are used for simplicity.

If the dividend is smaller than the divisor, the subtraction is not done, and the quotient bit is zero. Otherwise the quotient bit is 1; the difference (which is actually the remainder to this point) is shifted left one place.

```
                9/16  ÷  12/16  =  3/4
                         _______
                                  quotient bit 0.1100
                0.1100) 0.1001
                      - 0.1100     0

                        1.0010
                      - 0.1100     1      ( The computer will subtract
                        0.0110              by addition of twos complement)

                        0.1100
                      - 0.1100     1
                        0.0000

                        0.0000
                      - 0.1100     0

                        0.0000
                      - 0.1100     0

           Remainder    0.0000
```

The quotient bit is transferred to q_{47} so Q is also shifted left as division progresses. This neatly fits the scheme for a double length dividend with the minor half initially in Q.

It will be noted that three types of operations are required --- comparison: is $A < D$?; subtraction: if $A \geq D$; and shift, in all cases.

A familiar comparison-by-addition technique is used. The form of the numbers is arranged so that if equal the sum is a fixed value (2^1). If unequal, the sum is greater or less than this value depending upon which number is greater. To achieve this, one number should be in positive form and the other in negative form:

$$|N_1| + 10.0 - |N_2|$$

Initially, if the signs of A and D are alike, the twos complement of D is used, $(D') \longrightarrow D^*$, $1 \longrightarrow$ AN1C.
The sum will have this range.

$$\begin{array}{c} 0 \leqq |N| < 1.0 \\ \underline{1.0 \leqq 10.0 - |N| < 10.0} \\ \hline 1.0 \leqq \text{SUM} < 11.0 \end{array}$$

If $A = D$ $\quad (|A| + 10.0 - |D|) = 10.0$
When $|A| < |D|$ these will be the sums for the various cases of signs.

$+A+D$	$(|A| + 10.0 - |D|) < 10.0$	$a_o = 0$	$d_o^* = 1$	$s_o = 1$
$+A-D$	$(|A| + 10.0 - |D|) < 10.0$	$a_o = 0$	$d_o^* = 1$	$s_o = 1$
$-A-D$	$(10.0 - |A| + |D|) > 10.0$	$a_o = 1$	$d_o^* = 0$	$s_o = 0$
$-A+D$	$(10.0 - |A| + |D|) > 10.0$	$a_o = 1$	$d_o^* = 0$	$s_o = 0$

Therefore the active signal $d_o^* \neq s_o^*$ indicates $(A) \geqq (D)$ and that the subtraction can be performed.

Absolute value subtraction is the operation required by this method of performing division. This will be achieved if one input to AN1 is in positive form. (A) and (D*) have opposite signs and the AN1 output is the result of a subtraction.

If the subtraction can occur the quotient bit will be given a significant value (1 for a positive quotient, 0 for a negative quotient). This is the procedure for the last 47 cycles of the division algorithm which are done under the control of AI = 101.

QD Quotient Digit

Two timings intervene between the comparison of the numbers, each cycle, and the storing of the quotient bit (digit) in the Q register. The bit is therefore temporarily stored in the QD flipflop.

It is necessary to provide for the development of the quotient as either a positive or negative number. If the signs of the dividend and divisor are the same, the quotient is of course positive. In this case the significant value of the quotient bit is "1" when subtraction can occur.

With unlike signs, when subtraction occurs the significant value of the quotient bit is "0".

The state of AN1C can indicate whether the quotient is to be positive or negative. With like signs, the twos complement of (D) is required:

AN1C = 1	Indicates positive quotient
AN1C = 0	Indicates negative quotient

The method used to develop the quotient bit is as follows: the signal that clears Q*, (1 ⟶ Q*) also sets 1 ⟶ QD. The quotient bit should be zero under two conditions - positive quotient, not subtracting; negative quotient, subtracting.

$$\left[AN1C = 1 \cdot (A) \xrightarrow{o} A^*\right] \vee \left[AN1C = 0 \cdot (A + D^*) \longrightarrow A^*\right] \Longrightarrow 0 \longrightarrow QD$$

This is done during AT2. The least significant bit of Q is cleared to zero and if QD = 1, it is transferred to q_{-47} or q_{-35}, as the case may be.

Quotient Magnitude

The quotient, as developed by the algorithm, must be capable of being stored in a 48-bit register. This requires that the dividend be smaller than the divisor, generally, as the quotient must be in the range $-1 \geqq Q > +1$ to be stored.

Quotient overflow can be detected without performing the division since the relative magnitudes of the divisior and dividend can be compared in the first cycle of the division algorithm. This first cycle seeks a different result of the comparison than do the succeeding 47 cycles. In the first cycle of division, unless $|A| < |D|$ the division cannot be performed and the overflow will be set instead. The algorithm control is AI = 100 and the logic used is $a_o \neq s_o$?

Referring to the table listing a_o, d_o^*, and s_o, for the four cases of signs, it will be noted that $a_o \neq s_o$ when $|A| < |D|$. If $|A| > |D|$, $a_o = s_o$. In the first case, division can proceed and this is effected by changing AI* from 100 to 101, under whose control the algorithm can recycle. In the case of overflow, "OF" is set to 1 and the division is not attempted. Instead the contents of A and Q are shifted right one place and the instruction is ended.

The limits of acceptable quotients, i.e., conditions that do not produce overflow are set out in the table below (Table 6.6.4-1). Note that positive operands with equal absolute magnitudes cause overflow as the quotient developed would be +1. However, the quotient developed for negative operands with equal absolute magnitudes is $1-2^{-47}$. This number is representable in the S-2000. The quotient of -1 can be achieved by the computer only when A is negative and D positive.

Signs		Magnitude	Sum	a_o	d^*_o	s_o	Result
A	D						
+	+	$\|A\| = \|D\|$	=10.0	0	1	0	OF
+	+	$\|A\| = \|D\|$	>10.0	0	1	0	OF
-	-	$\|A\| = \|D\|$	=10.0	1	0	0	$1 - 2^{-47}$ $(q - 2^{-47})$
-	-	$\|A\| > \|D\|$	<10.0	1	0	1	OF
+	-	$\|A\| = \|D\|$	=10.0	0	1	0	OF
+	-	$\|A\| > \|D\|$	>10.0	0	1	0	OF
-	+	$\|A\| = \|D\|$	=10.0	1	0	0	2 - 1 (2 - q)
-	+	$\|A\| > \|D\|$	<10.0	1	0	1	OF

Overflow for Fixed Point Division

Table 6.6.4-1

The dividend is always treated as double-length by the algorithm logic for simplicity of organization. The Q register is always shifted left along with the A register. However, in single-length division, Q is cleared to zero prior to the algorithm.

The following lists the logic and activity for the instruction and algorithm timings. Floating point will be subsequently discussed.

IT1

The usual control clearings and preparation to transfer the address to MA.

IT2

	Trigger AN2CC
Transfer address.	$AN2 \longrightarrow MA$
Transfer divisor storing quotient	$2 \longrightarrow MI$
not storing quotient	$4 \longrightarrow MI$
Set algorithm control for fixed point	$4 \longrightarrow AI$
Set controls for floating point	$5 \longrightarrow AI$, $1 \longrightarrow FI$
Set SC: fixed point	$48 \longrightarrow SC$
floating point	$36 \longrightarrow SC$
Program control	$(SW2)' \longrightarrow$ Mod 2
Read transfer complete	$MT5 \longrightarrow IT3$

IT3

Unlike signs, prepare transfer (D)	$0 \longrightarrow D^*_M$, $0 \longrightarrow AN1C$
fixed point	$0 \longrightarrow D^*_E$
Like signs, prepare transfer (D)''	$1 \longrightarrow D^*_M$, $1 \longrightarrow AN1C$
fixed point	$1 \longrightarrow D^*_E$

IT3(continued)

Single-length dividend, clear Q	$0 \longrightarrow Q_M$
fixed point	$0 \longrightarrow Q_E$
Floating point, prepare transfer $(D_E)'$	$1 \longrightarrow D^*_E$
Program control	0, 1, 2, or 3 $\longrightarrow$ PI

IT4

AN2CC trigger
Inhibit AN1

Unlike signs	$(D_M) \longrightarrow D^*_M$
fixed point	$(D_E) \longrightarrow D^*_E$
Like signs	$(D_M)' \longrightarrow D^*_M$
fixed point	$(D_E)' \longrightarrow D^*_E$
Floating point	$(D_E)' \longrightarrow D^*_E$, $1 \longrightarrow$ AN1CE
Fixed point	$\longrightarrow$ AT1
Floating point	$\longrightarrow$ FT1

AT1

Clear A*, Q*, QD AI = 10X	$1 \longrightarrow A^*_M$, $1 \longrightarrow Q^*_M$, QD
AI = 10X · FP = 0	$1 \longrightarrow A^*_E$, Q^*_E
Prepare SC count down AI = 10X	(SC) $\longrightarrow$ SC*
Control timing alone	(AI) $\longrightarrow$ AI*
Floating point, clear FI FP = 1	$0 \longrightarrow$ FI

AT2 Trigger AN1CC

1st cycle:

Proceed with division,

$\lvert A \rvert < \lvert D \rvert$	$AI = 100 \cdot a_o \neq s_o$	$5 \longrightarrow AI^*$

Do not proceed,

$\lvert A \rvert > \lvert D \rvert$	$AI = 100 \cdot a_o = s_o$	$1 \longrightarrow OVF$

Prepare shift remainder	$AI = 100$	$(A_M) \xrightarrow{o} A^*_M$
	$AI = 100 \cdot FP = 0$	$(A_E) \xrightarrow{o} A^*_E$
	$AI = 10X$	$(Q_M) \xrightarrow{o} Q^*_M$
	$AI = 10X \cdot FP = 0$	$(Q_E) \xrightarrow{o} Q^*_E$
Positive quotient	$(A_M) \longrightarrow A^*_M \cdot AN1C = 1$	$0 \longrightarrow QD$
After 1st cycle: Subtract $A_M \geqq D^*_M$	$AI = 101 \cdot d^*_o \neq s_o$	$(A_M + D^*_M) \xrightarrow{o} A^*_M$
	$AI = 101 \cdot d^*_o \neq s_o \cdot FP = 0$	$(A_E + D^*_E) \xrightarrow{o} A^*_E$
	$(A_M + D^*_M) \longrightarrow A^*_M \cdot AN1C = 0$	$0 \xrightarrow{o} QD$
Transfer unaltered	$AI = 101 \cdot d^*_o = s_o$	$(A_M) \xrightarrow{o} A^*_M$
remainder, $A_M < D_M$	$AI = 101 \cdot d^*_o = s_o \cdot$	
	$\cdot FP = 0$	$(A_E) \xrightarrow{o} A^*_E$
	$(A_M) \longrightarrow A^*_M \cdot AN1C = 1$	$0 \longrightarrow QD$
Prepare shift quotient		
(and double-length	$AI = 10X$	$(Q_M) \xrightarrow{o} Q^*_M$
dividend)	$AI = 10X \cdot FP = 0$	$(Q_E) \xrightarrow{o} Q^*_E$

AT3

Clear A, Q,	$AI^* = 10X$	$0 \longrightarrow A_M,\ Q_M$
	$AI^* = 10X \cdot FP = 0$	$0 \longrightarrow A_E,\ Q_E$
Count down SC	$AI^* = 10X$	$(SC^* - 1) \longrightarrow SC$
Continue divison	$AI^* = 101$	$5 \longrightarrow AI$
Floating point,	$FP = 1 \cdot q^*_{-1} = q^*_{-2} \cdot$	
Normalizing Required	$\left[(AI^* = 101 \cdot SC = SAT) \vee (AI^* = 101 \cdot FI^* = 100 \cdot FP = 1)\right]$	$4 \longrightarrow FI$

AT4

Shift remainder left for next cycle of algorithm	$AI^* = 101 \cdot SC \neq SAT \cdot \overline{(FI^* = 4 \cdot FP = 1)}$	$(A^*_M) \xrightarrow{L} A_M$, $a^*_{-1} \xrightarrow{L} a_o$
	(above conditions) $\cdot$ $FP = 0$	$(A^*_E) \xrightarrow{L} A_E$, $q^*_{-1} \longrightarrow a_{-47}$
	(above conditions) $\cdot$ $FP = 1$	$q^*_{-1} \longrightarrow a_{-35}$
Shift quotient (and	$AI^* = 101$	$(Q^*_M) \xrightarrow{L} Q_M$
double-length dividend) left	$AI^* = 101 \cdot FP = 0$	$(Q^*_E) \xrightarrow{L} Q_E$
Transfer QD $\longrightarrow$ Q	$AI^* = 101 \cdot FP = 0$	$(QD) \longrightarrow q_{-47}$
	$AI^* = 101 \cdot FP = 1$	$(QD) \longrightarrow q_{-35}$
Last cycle, do not shift remainder	$(AI^* = 101 \cdot SC = SAT) \vee (AI^* = 101 \cdot FI^* = 100 \cdot FP = 1)$	$(A^*_M \longrightarrow A_M$, $q^*_{-1} \rightarrow q_o$
	$(AI^* = 101 \cdot SC = SAT \cdot FP = 0)$	$(A^*_E) \longrightarrow A_E$

Fixed point division not possible, shift dividend right	$AI^* = 100$	$(A^*_M) \xrightarrow{R} A_M$ $(Q^*_M) \xrightarrow{R} Q_M$; $q^*_o \longrightarrow q_o$; $a^*_o \longrightarrow a_o$
	$AI^* = 100 \cdot FP = 0$	$(A^*_E) \xrightarrow{R} A_E$; $a^*_{-47} \longrightarrow q_{-1}$; $(Q^*_E) \xrightarrow{R} Q_E$
	$AI^* = 100 \cdot FP = 1$	$a^*_{-35} \longrightarrow q_{-1}$
Recycle	$AI^* = 101 \cdot SC \neq SAT \cdot \overline{(FI^* = 100 \cdot FP = 1)}$	$\longrightarrow AT1$
Exit, fixed point	$AI^* = 101 \cdot SC = SAT \cdot FI = 000 \cdot II = 0$	$\longrightarrow END$
Exit, floating point	$AI^* = 101 \cdot FI^* = 100 \cdot FP = 1$	$\longrightarrow FT1$
Exit, overflow	$AI^* = 100 \cdot FI = 000 \cdot II = 0$	$\longrightarrow END$
Store Quotient		$END \longrightarrow IT7$
IT7		
Clear D		$1 \longrightarrow D$
IT8		Trigger D117
Transfer quotient		$(Q) \xrightarrow{o} D$
Start write operation		$3 \longrightarrow MI$
		$\longrightarrow END$

(If overflow during the divide-and-store instructions, zero or the minor half of the dividend, shifted right, will be written into memory.)

6.6.5 Floating Point Numbers

To briefly review, the floating point number in the S-2000 is composed of two parts, (2^{-M} X $2^{\pm E}$) mantissa and exponent. The mantissa contains the sign of the number (bit 2^0) and a binary fraction of 35 place accuracy. The positive mantissa has a range of:

$$0 \leqq +M < 2^0$$

The negative mantissa is represented by the range:

$$2^0 \leqq -M < 2^1$$

The negative value is represented as ($2^1 - |M|$) and ($2^{12} - |E|$). As in fixed point numbers, the larger (more positive) negative number is represented by a larger value binary number than a smaller negative number.

The exponent part of the floating point number consists of a sign (bit 2_{-36}) and an 11-bit binary integer. The positive integer has a range:

$$0 \leqq +E < 2^{11}$$

$$000\ 000\ 000\ 000 \leqq +E < 100\ 000\ 000\ 000$$

The negative exponent is represented in the usual complementary method by: $-E = \left[2^{12} - |E| \right]$ with the range:

$$2^{12} < -E \leqq 2^{11}$$

$$1\ 000\ 000\ 000\ 000 < -E \leqq 100\ 000\ 000\ 000$$

The greater the negative exponent, the smaller its absolute value.

The range of exponent number values is $-2048 \leqq E \geqq 2047$

Normalizing

To facilitate floating point arithmetic, the numbers are "normalized". The exponent is reduced to the smallest value that can be used to represent the number. The mantissa is accordingly shifted left during this process. The exponent is reduced by one and the mantissa shifted left one place until a significant bit appears in the 2_{-1} position (detected as $2_0 \neq 2_{-1}$) or 36 shifts have been performed. 36 shifts will result in a mantissa of zero. The purpose of normalizing is to allow the maximum degree of accuracy in representing a quantity within the capacity of the 35-bit part of the register. This is achieved when the most significant bit position has a significant value. For example, a quantity in the form $(2^{-35} + 2^{-37}) \times 2^{10}$ could only be contained in the register to the accuracy of $2^{-35} \times 2^{10}$. Normalizing will permit greater accuracy: $(2^{-1} + 2^{-3}) \times 2^{-24}$.

The general practice is to normalize numbers as they are entered into the computer. Most conversion routines, as TAC, provide this feature.

Most results of floating point arithmetic are normalized as part of the arithmetic instruction. The exception is when normalization would require the exponent to be smaller than -2048 (exponent underflow). Since it is considered best to preserve the accuracy, normalization is stopped at this lower exponent limit and the number is left non-normalized. The accuracy may or may not be reduced in subsequent arithmetic operations. The possibility of preserving this extended accuracy is maintained as long as is permissible.

If unnormalized numbers are used, the sole potential loss is in the accuracy of the result. When two non-normalized quantities are entered into the computer and used, this may occur. When $(2^{-30} \times 2^{-5}) + 2^{-20} \times 2^{-45})$ is performed, the sum will be $(2^{-30} \times 2^{-5})$ which will be normalized to $(2^{-1} \times 2^{-34})$. The addend was made zero as its exponent was smaller than the augend exponent by more than 35. The same values normalized $(2^{-1} \times 2^{-34}) + (2^{-30} \times 2^{-64})$, would result in the sum $(2^{-1} \times 2^{-34}) + (2^{-30} \times 2^{-34})$ or $\left[(2^{-1} + 2^{-30}) \times 2^{-34}\right]$. The sum is more accurate.

It is the responsibility of the programmer to scale the numbers to achieve the desired accuracy of the result.

A non-normalized number that has been entered into the computer can usually be normalized by addition to zero (Clear Add).

Zero could be represented by a mantissa value of zero and any of the 4095 possible exponent values. Normalizing, however, results in one representation which is considered the standard form of zero.

$$A_M = 0.00 \ldots . 0 \qquad A_E = 1.00 \ldots . 0$$

This conforms with the rule of making the exponent as small as possible.

Normalizing Procedure

The control state is FI = 100. This basic illustration of normalizing would be for add or subtract. At the end of the preceding algorithm, $36 \longrightarrow SC$ to set the shift limit.

Preparation is also made to subtract one from the exponent (A_E) by $1 \longrightarrow D^*_E$, $0 \longrightarrow$ AN1CE.

FT1		trigger D49
	Control activity	$(FI) \longrightarrow FI^*$
	Prepare count down SC	$(SC) \longrightarrow SC^*$
	Clear A^*_M	$1 \longrightarrow A^*_M$
	Clear A^*_E	$1 \longrightarrow A^*_E$
	If $(A_E - 1) < -2048$, underflow (this simplification will be later qualified)	$1 \longrightarrow UF$
FT2		trigger AN1ECC
	Prepare to shift mantissa	$(A_M) \longrightarrow A^*_M$
	Transfer difference $(A_E - 1)$	$(A_E + D^*_E) \longrightarrow A^*_E$
FT3		
	Count down SC	$(SC^* - 1) \longrightarrow SC$
	Control activity	$(FI^*) \longrightarrow FI$
	Clear A_M, A_E is no underflow	$0 \longrightarrow A_M, A_E$

FT4

If no underflow:

Transfer new exponent value	$(A^*_E) \longrightarrow A_E$
And if SC $\neq$ SAT, shift mantissa left	$(A^*_M) \xrightarrow{L} A_M$
	$a^*_0 \longrightarrow a_0$
And if number now normalized	$\longrightarrow$ END
And if number not normalized	$\longrightarrow$ FT1

If underflow, end instruction	$\longrightarrow$ END
If 36th shift (SC = SAT) generate standard form of zero (A_M was cleared in FT3).	$1 \longrightarrow a_{-36}$
And end.	$\longrightarrow$ END

Normalization is detected by $a^*_0 \neq a^*_{-2}$. The bit a^*_{-2} is the same as bit a_{-1} of the shifted number.

Underflow will leave the non-normalized number in A_M with $(A_E) = -2048$ as A was not altered in this event.

Overflow and Underflow

Each of the two parts of the floating point number that is the result of an arithmetic operation can independently exceed the capacity of the part of the register allotted to store it.

The mantissa exceeds the storage capacity when:

$$2^{-1} < M \geq 2^0$$

Excess in either direction is a mantissa overflow. The exponent exceeds its storage capacity when:

$$\underbrace{\longleftarrow\!\!|}_{\text{Exponent underflow}} \; -2048 > E > +2047 \; \underbrace{|\!\!\longrightarrow}_{\text{Exponent overflow}}$$

Several control flip-flops are set to note these out-of-bounds developments in the exponents.

UF = 1	The exponent is now smaller than -2048.
E0 = 1	Exponent now out-of bounds, either exponent underflow or overflow.
EF = 1	Exponent Fault. The entire number is now too large to be represented if this condition exists or will exist at the end of the instruction.

In some cases it is possible to adjust for the excess in one part of the number by changing the other part of the number.

Correction

If addition or subtraction results in mantissa overflow ($AN1C_{-0} \neq AN1C_{-1}$), it can be accommodated thusly:

$$(+2^{0} \times 2^{-1021}) = (+2^{-1} \times 2^{-1020})$$

$$(-2^{1} \times 2^{2046}) = (-2^{0} \times 2^{2047})$$

The exponent is increased by one and the mantissa shifted right one place (in effect, divided by two). This can be done as long as the increased exponent does not cause exponent overflow. The upper limit is always $E = 2^{2047}$ as the exponent is being increased.

This process is known as "correction" and the state of FI that controls the process is FI = 101.

When correction is not possible, due to the exponent already being equal to +2047, the entire number has overflowed, or the mantissa is too small to be corrected without exponent overflow. Because the exponent limits the correction, this condition is termed "Exponent Fault" (EF) and the associated flip-flop will be set ($1 \longrightarrow EF$).

Another point should be noted about mantissa overflow. The capacity of the register is never exceeded by more than one (2^0) when the mantissa overflows in add, subtract or multiply. The maximum sum is with the positive mantissas,

$$(2^0 - 2^{-35}) + (2^0 - 2^{-35}) = (2^1 - 2^{-34})$$

The difference between this sum and the register is less than one,

$$(2^1 - 2^{-34}) - (2^0 - 2^{-35}) < 2^0$$

The minimum mantissa sum occurs when adding (-1) + (-1), represented as $2^0 + 2^0 = 2^1$. This exceeds the capacity by 2^0.

An increase of one in the exponent is equivalent to a decrease of 2^0 in the mantissa, which is equal to or greater than the potential overflow excess. Correction can be done by a change of one mantissa place and an exponent increase of one.

Correction, Floating Point

In AT2 if overflow exists, $5 \longrightarrow$ FI. This illustration is for add or subtract. Under these conditions, in AT4:

AT4		
	Add one to A_E	$1 \longrightarrow$ AN1CE
		$0 \longrightarrow D^*_E$
		$\longrightarrow$ FT1
FT1		
		$(FI) \longrightarrow FI^*$
	Clear A^*_M, A^*_E	$1 \longrightarrow A^*_M, A^*_E$
	If exponent increase caused overflow	$1 \longrightarrow$ EF

FT2

Prepare shift of (A_M)	$(A_M) \longrightarrow A^*_M$
Transfer exponent sum	$(A_E + D_E) \xrightarrow{M} A^*_E$

FT3

Clear A	$0 \longrightarrow A_M, A_E$

FT4

Shift mantissa	$(A^*_M) \xrightarrow{R} A_M$
	$d^*_0 \longrightarrow a_0$
Transfer new exponent	$(A^*_E) \longrightarrow A_E$
End instruction	$\longrightarrow$ END

As mantissa overflow is produced only when numbers have the same sign, d^*_0 is used to provide the sign of the corrected number..

When correction resutls in exponent overflow, the fault is indicated. The register will contain a mantissa of either 0.1 or 1.0. The exponent of +2048 will have a "negative-appearance", 1.00 000 000 000, as +2048 is beyond its capacity.

When exponent fault exists at the end of the instruction performance, the computer control will jump to the I_0 half of location 00000. From this point corrective action may be programmed.

Before jumping, (PA) are stored in JA to indicate the approximate program location of the instruction that developed the exponent fault. PA at this time does not contain the address of the fault-causing instruction, but rather, that-address-plus-one. The correction routine must reduce (JA) by 1 to learn the actual memory location involved.

Setting up this jump requires two timings. In the first $0 \longrightarrow$ JA, (PA) $\longrightarrow$ JA, and $1 \longrightarrow ja_{-1}$, if SW2 = 1. Then $0 \longrightarrow$ PA, $0 \longrightarrow$ Mod2, $1 \longrightarrow$ PI. And then proceed to PT1. EF will be cleared to 0 during PT3 of PI = 01.

6.6.6 Floating Point Addition, Subtraction

Prior to addition or subtraction of floating point numbers, their exponents have to be made equal so the mantissas bear the proper relation to each other.

$$\begin{array}{r} 2^{-4} \times 2^{6} \\ + \quad 2^{-4} \times 2^{3} \\ \hline \end{array} \longrightarrow \begin{array}{r} 2^{-4} \times 2^{6} \\ + \quad 2^{-4} \times 2^{6} \\ \hline 2^{-3} \times 2^{6} \end{array}$$

The general method of arrangement used is to change the smaller, (less positive) exponent to the value of the larger exponent. Then the mantissa of the number with the changed exponent is shifted right the number of places equal to the difference between the exponents, for example:

If $A_E < D_E$; $D_E \longrightarrow A_E$ and shift right (A_M)

$(D_E - A_E)$ places.

Actually the exponent transfer need only occur if the exponent in A is the smaller. During addition of the numbers, only the mantissas are sent through AN1. A_E is not disturbed during the ATs for floating point numbers. Therefore with the sum to be in A, there is no need to transfer $(A_E) \longrightarrow D_E$, if the latter were the smaller number.

As the mantissa can only be represented to an accuracy of 35 places, when the difference between exponents is greater than 35, the mantissa of the smaller number is made equal to zero. For the closest the machine can represent the sum of $(2^{-1} \times 2^{100}) + (2^{-1} \times 2^{50})$ is $(2^{-1} \times 2^{100})$. The increase is $(2^{-51} \times 2^{100})$ and cannot be represented with the 35 place accuracy of the mantissa.

The logic simultaneously tests to detect if $A_E \leqq (D_E + 35)$, if $A_E > (35 + D_E)$ or if $A < (D_E - 35)$. If the first condition is detected, D_M is shifted right. If the second condition is detected, D_M is cleared to zero. If the third condition is detected, A_M is cleared to zero. If none is detected, A_M is shifted right.

The exponents are compared by adding the twos complement of D_E to A_E. Table 6.6.6-1 lists the logic used. The initial state of FI for this arrangement is FI = 000. During the ITs $(D_E)' \longrightarrow D^*_E$ and $1 \longrightarrow$ AN1CE. The result is examined in FT1.

Comparison of Exponents with Like Signs

$0 \leqq +E < 2^{11}$	decimal
$000\ 000\ 000\ 000 \leqq +E < 100\ 000\ 000\ 000$	binary
$0 \leqq +E \leqq +2047$	number range
$-2048 \leqq -E \leqq -1$	number range
$2^{11} \leqq -E > 2^{12}$	decimal
$100\ 000\ 000\ 000 \leqq -E \leqq 111\ 111\ 111\ 111$	binary

The two exponents are compared by subtraction which is performed as: $A_E + (2^{12} - |D_E|)$. Like signs will be indicated by $C_{-36} = C_{-37}$. If $A_E \geqq D_E$, S_{-36} will be 0; if $A_E < D_E$, S_{-36} will be 1.

$+A_E \geqq +D_E$

If they are both positive and $A_E \geqq D_E$, the range of the sum, S, is:

$$(2^{12} + 2^{11}) > S \geqq 2^{12}$$

$$1\ 011\ 111\ 111\ 111 \geqq S \geqq 1\ 000\ 000\ 000\ 000$$

This range is characterized by S_{-36} (sum bit for AN1 bit position $_{-36}$) being 0, like signs is indicated by $C_{-36} = C_{-37}$. (Since $a_{-36} = 0$, $d^*_{-36} = 1$ and $S \geqq 2^{12}$, C_{-36} and C_{-37} will equal 1.

If the difference between these exponents permits arrangement of the mantissa of the smaller (difference is equal to or less than 35) the range of the sum for positive exponents is:

$$2^{12} \leqq S \leqq 2^{12} + 35$$

Relative Exponent Values	Logic of Determination FI = 000	Action	Method
$+A_E \geqq (-D_E + 2048)$	$C_{-36} = 0 \cdot C_{-37} = 1$	Clear D_M^*	$6 \longrightarrow FI^*$
$-A_E < (+D_E - 2048)$	$C_{-36} = 1 \cdot C_{-37} = 0$	Clear A_M, $(D_E) \rightarrow A_E$	$7 \longrightarrow FI^*$
$\pm D_E \leqq \pm A_E \leqq (\pm D_E + 35)$	$C_{-36} = C_{-37} \cdot S_{-36} = 0$ $\lvert S \rvert < 35$	Shift (D^*_M) Right	$2 \longrightarrow FI^*$
$(\pm D_E + 35) < \pm A < (-D_E + 2048)$	$C_{-36} = C_{-37} \cdot S_{-36} = 0$ $\lvert S \rvert > 35$	Clear D*	$6 \longrightarrow FI^*$
$(\pm A_E + 35) < \pm D_E \leqq (-A_E + 2048)$	$C_{-36} = C_{-37} \cdot S_{-36} = 1$ $\lvert S \rvert > 35$	Clear A_M, (D_E) A_E	$7 \longrightarrow FI^*$

If None of the Above Conditions Existed During FTI Timing, The Remaining Possibility is Acted Upon in FT3.

$\pm A_E < \pm D_E \leqq (\pm A_E + 35)$	$FI^* = 000$	Shift (A_M) Right, $(D_E) \longrightarrow A_E$

Note C_{-36} = ANIC $_{-36}$

C_{-37} = ANIC $_{-37}$

S_{-36} = Sum Bit ANI $_{-36}$

$\lvert S \rvert$ Absolute Value of Difference Between A_E and D_E

Table 6.6.6-1 Exponent Comparison for Add or Subtract

If the difference is greater than 35, the range of the sum is:

$$1\ 100\ 000\ 000\ 000 > S > 100\ 000\ 100\ 011$$

The symbols of the following table are used to indicate a sum bit value of 1 for the twelve AN1E bit positions:

S_{-36}	S_{-37}	S_{-38}	S_{-39}	S_{-40}	S_{-41}	S_{-42}	S_{-43}	S_{-44}	S_{-45}	S_{-46}	S_{-47}
A	B	C	D	E	F	G	H	I	J	K	L

The positive number, 35 would be symbolized by:

$$\begin{array}{cccc} 000 & 000 & 100 & 011 \\ \overline{A}\,\overline{B}\,\overline{C} & \overline{D}\,\overline{E}\,\overline{F} & G\overline{H}\,\overline{I} & \overline{J}KL \end{array}$$

And the logic for determining a difference greater than 35 for this case is:

$$N > 35 = \overline{A}\ \left[(B \vee C \vee D \vee E \vee F) \vee G\ (H \vee I \vee J) \right]$$

If these conditions do not exist, the $N \leqq 35$ signal line will be active. (On some logic schematics the term "$|S|$" is used instead of the term "N".) See Figure 6.6.6-1.

With N or $|S| \leqq 35$, the value of the six least significant bits of the sum is the number of places D^*_M must be shifted right. These six bits are transferred to the shift counter as the shift parameter.

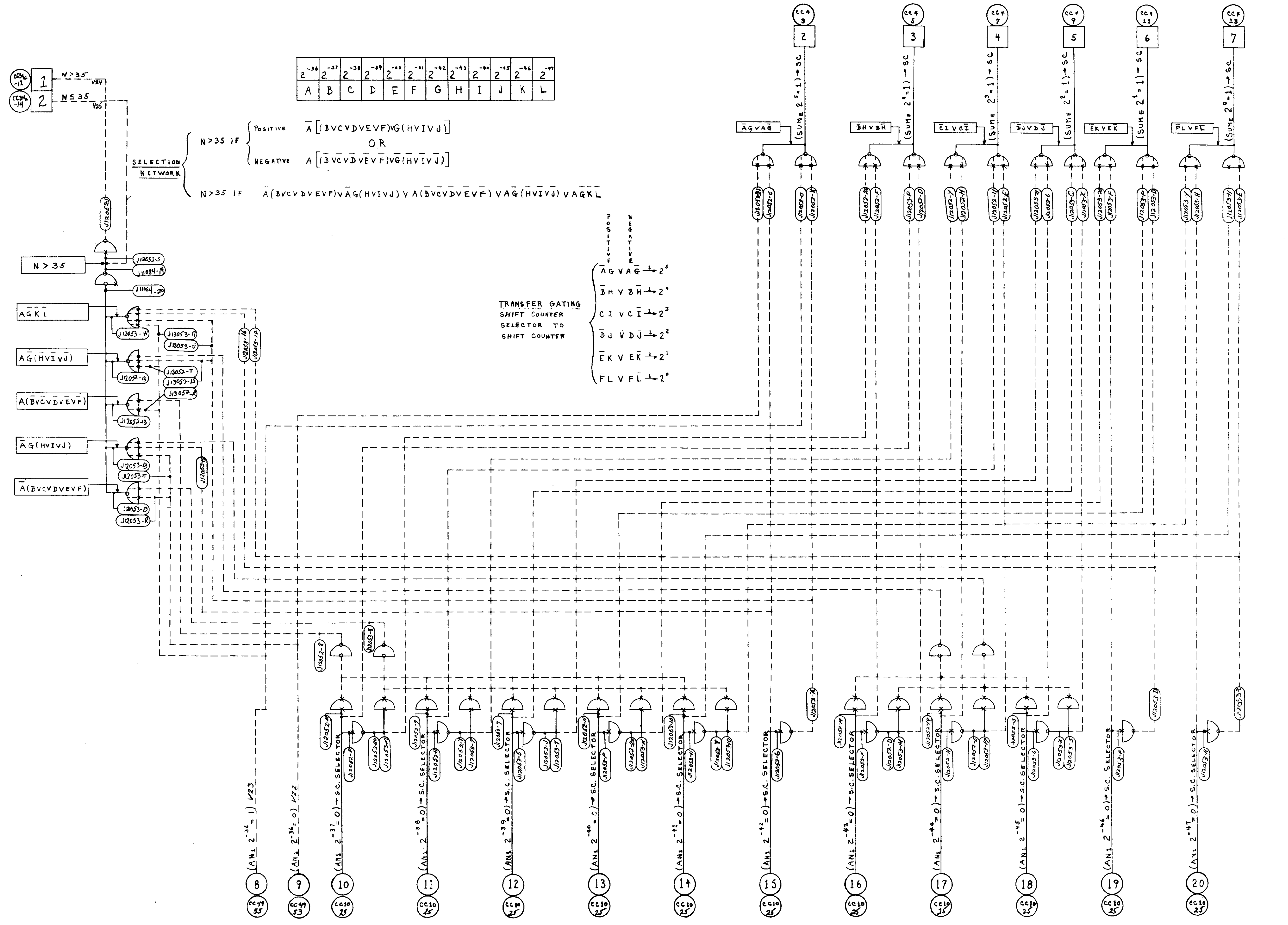
<u>$+A_E < +D_E$</u>

When $+A_E < +D_E$, the range of the sum

$$\left[A_E + (2^{12} - |D_E|)\right] \text{ is} \left[2^{12} - (|D_E| - |A_E|\right]$$

$$2^{12} > S > 2^{11}$$

$$1\ 000\ 000\ 000\ 000 > S > 100\ 000\ 000\ 000$$



N > 35
N ≤ 35
SELECTION NETWORK
N>35 IF Positive $\bar{A}[(BVCVDVEVF)VG(HVIVJ)]$ OR Negative $A[(\overline{BVCVDVEVF})VG(\overline{HVIVJ})]$
N>35 IF $\bar{A}(BVCVDVEVF)V\bar{A}G(HVIVJ)VA(\overline{BVCVDVEVF})VAG(\overline{HVIVJ})VAGKL$
TRANSFER GATING SHIFT COUNTER SELECTOR TO SHIFT COUNTER
$\bar{A}GVA\bar{G} \xrightarrow{1} 2^5$
$\bar{B}HVB\bar{H} \xrightarrow{1} 2^4$
$\bar{C}IVC\bar{I} \xrightarrow{1} 2^3$
$\bar{D}JVD\bar{J} \xrightarrow{1} 2^2$
$\bar{E}KVE\bar{K} \xrightarrow{1} 2^1$
$\bar{F}LVF\bar{L} \xrightarrow{1} 2^0$
AGKL
$A\bar{G}(\overline{HVIVJ})$
$A(\overline{BVCVDVEVF})$
$\bar{A}G(HVIVJ)$
$\bar{A}(BVCVDVEVF)$
(Sum $2^5=1$) → SC
(Sum $2^4=1$) → SC
(Sum $2^3=1$) → SC
(Sum $2^2=1$) → SC
(Sum $2^1=1$) → SC
(Sum $2^0=1$) → SC
(AN1 $2^{-36}=1$) V23
(AN1 $2^{-36}=0$) V22
(AN1 $2^{-37}=0$) → S.C. SELECTOR
(AN1 $2^{-38}=0$) → S.C. SELECTOR
(AN1 $2^{-39}=0$) → S.C. SELECTOR
(AN1 $2^{-40}=0$) → S.C. SELECTOR
(AN1 $2^{-41}=0$) → S.C. SELECTOR
(AN1 $2^{-42}=0$) → S.C. SELECTOR
(AN1 $2^{-43}=0$) → S.C. SELECTOR
(AN1 $2^{-44}=0$) → S.C. SELECTOR
(AN1 $2^{-45}=0$) → S.C. SELECTOR
(AN1 $2^{-46}=0$) → S.C. SELECTOR
(AN1 $2^{-47}=0$) → S.C. SELECTOR

2^{-36}	2^{-37}	2^{-38}	2^{-39}	2^{-40}	2^{-41}	2^{-42}	2^{-43}	2^{-44}	2^{-45}	2^{-46}	2^{-47}
A	B	C	D	E	F	G	H	I	J	K	L

Figure 6.6.6-1 Shift Counter Selector

This sum is characterized by $S_{-36} = 1$, and $C_{-36} = C_{-37} = 0$. ($a_{-36} = 0$, $d^*_{-36} = 1$, $C_{-36} = 0$ and C_{-37} must equal 0 to have a sum in this range.)

If $(+A_E + 35) < +D_E$, the range of the sum becomes:

$$(2^{12} - 35) > S > 2^{11}$$

$$111\ 111\ 011\ 101 > S > 100\ 000\ 000\ 000$$

A value in this range is identified as being smaller than $(2^{12} - 35)$ by this logic:

$$\begin{array}{ll} (2^{12} - 35) & 1\ 1\ 1 \quad 1\ 1\ 1 \quad 0\ 1\ 1\ 1 \quad 0\ 1 \\ \text{(Logic)} & A\left[(\bar{B} \vee \bar{C} \vee \bar{D} \vee \bar{E} \vee \bar{F}) \vee (\bar{G}\ \bar{H} \vee \bar{I} \vee \bar{J}) \vee \bar{G}\ \bar{K}\ \bar{L}\right] \end{array}$$

In this case $N > 35$ becomes active and A_M will be cleared.

If $N \leq 35$, then the ones complement of the value of the six least significant bits is one less than the number of places A_M is to be shifted. For example, inverting the six bits of the number above $(2^{12} - 35)$ will result in 100 010 or 34. This is transferred to the shift counter as the shift parameter. The logic compensates by not counting down SC the first time A_M is shifted. (A_M) will be shifted in this instance, 34 + 1 places.

$-A_E \geqq -D_E$

When both exponents are negative they are compared by:

$$(2^{12} - |A_E|) + \left[2^{12} - (2^{12} - |D_E|)\right] \quad \text{or } 2^{12} - |A_E| + |D_E|$$

If equal the sum is 2^{12}. If $A_E \geqq D_E$, the absolute value of A_E is smaller than that of D_E (e.g. $2^{-4} > 2^{-8}$) and the range of the sum is:

$$(2^{12} + 2^{11}) > S \geq 2^{12}$$

$$1\ 011\ 111\ 111\ 111 \geqq S \geqq 1\ 000\ 000\ 000\ 000$$

with $S_{-36} = 0$ and $C_{-36} = C_{-37}$.

If the difference is greater than 35, $|S| >$ 000 000 100 011.

<u>$-A_E < -D_E$</u>

The sum in this instance will be:

$$2^{12} > S > 2^{11}$$

$$111\ 111\ 111\ 111 \geqq S \geqq 100\ 000\ 000\ 001$$

with $S_{-36} = 1$ and $C_{-36} = C_{-37}$. The right hand binary number is the sum when $A_E = -2048$ and $D_E = -1$. If the difference is more than 35, the range is:

$$(2^{12} - 35) > S > 2^{11}$$

$$111\ 111\ 011\ 101 > S > 100\ 000\ 000\ 000$$

If $A_E < D_E$ the value transferred to the shift counter is always one less than the actual number of places to be shifted. This is compensated for as previously mentioned.

<u>Exponents with Unlike Signs</u>

Additional logic is required for these cases as some of the sums lie outside the ranges covered by the logic for exponents with like signs.

In negative numbers the smaller absolute value is the greater quantity. When $-A_E > -D_E$, $|A_E| < |D_E|$. Adding A_E and the twos complement of D_E resulted in a sum that exceeded 2^{12}.

The closest any two exponent numbers of unlike signs can approach each other are difference of -1 and 0. The greatest difference is for -2048 and +2047. In looking at the absolute values of numbers with unlike signs, as either or both absolute values increase in magnitude the difference between the numbers becomes greater.

If the values of two numbers with unlike signs are close to each other, the sum of their absolute values will be small. This sum (of absolute values) will increase as the difference between the numbers increases. The range is:

$$|0| + |1| \leqq S \geqq |2047| + |2048|$$

$$2^{0} \leqq S \geqq 2^{12} - 1$$

This sum of absolute values actually is the difference between the numbers when dealing with numbers of unlike signs.

$+A_E$, $-D_E$

$+ \left[(2^{12} - (2^{12} - |D_E|)\right]$ Obviously A_E is greater. The operation is A or $|A_E| + |D_E|$. The sum ranges:

$$0 < S < 2^{12}$$

$$000\ 000\ 000\ 001 \leqq S \leqq 111\ 111\ 111\ 111$$

which can be divided into two parts:

$$\text{(I) } 0 < S < 2^{11} \qquad \text{(II) } 2^{11} \leqq S < 2^{12}$$

Part I has $S_{-36} = 0$ and $C_{-36} = C_{-37}$. This is as $a_{-36} = 0$, $d^*_{-36} = 0$ and $C_{-36} = 0$ since the sum is less than 2^{11} . Therefore C_{-37} must also be 0. The existing logic will shift or clear D, depending upon the magnitude of the difference. D can be shifted if $|A_E| + |D_E| \leqq 000\ 000\ 100\ 011$.

If the sum is in part II the difference is equal to or greater than 2^{11} . This means the difference between these exponents is equal to or greater than 2048, a difference completely beyond the adjustment capacity of the mantissa (35 places). D is effectively zero in relation to the number in A and the D register should be cleared to zero.

This range, $2^{11} \leqq S < 2^{12}$ can be distinguished by $C_{-36} = 0 \cdot C_{-37} = 1$ as $a_{-36} = 0$ and $d^*_{-36} = 0$, C_{-37} must equal 1 to make the sum equal or exceed 2^{11} . If the sum is to be less than 2^{12} , C_{-36} must equal zero.

$\underline{-A_E \text{ and } +D_E}$

If the difference is 35 or less, A can be shifted. If greater than 35, A is cleared to zero. The logic detects the latter condition:

$$(2^{12} - |A_E|) + (2^{12} - |D_E|) = 2^{13} - (|A_E| + |D_E|)$$

$$(2^{13} - 1) \geqq S \geqq (2^{12} + 1)$$

$$1\ 111\ 111\ 111\ 111 \geqq S \geqq 1\ 000\ 000\ 000\ 001$$

While the computer cannot store 2^{12}, $C_{-36} = 1$ indicates that value. Here again if the absolute values are small in magnitude the exponents are close to each other and the sum is near 2^{13}. The range of the sum for which the number in A is sufficiently close to the number in D is when the sum of the exponent is

$$111\ 111\ 111\ 111 \geqq S \geqq 111\ 111\ 011\ 101$$

If the sum is less, A is cleared to zero.

The logic covers the sum in two parts.

(Part I) $$(2^{13} - 1) \geqq S \geqq 2\,(2^{12} + 2^{11})$$

$$1\ 111\ 111\ 111\ 111 > S \geq 1\ 100\ 000\ 000\ 000$$

This part I is covered by the same logic that serves $(-A_E + 35) < -D_E$

$$S_{-36} = 1 \bullet C_{-36} = C_{-37} \bullet |S| > 35$$

(Part II) $$(2^{12} + 2^{11} - 1) \geqq S \geqq (2^{12} + 1)$$

$$1\ 011\ 111\ 111\ 111 > S > 1\ 000\ 000\ 000\ 001$$

For this range the minimum difference between exponents is 2049. The A register should be cleared to zero. This range is identified by:

$$C_{-36} = 1 \bullet C_{-37} = 0$$

This will be since $a_{-36} = 1$, $d^*_{-36} = 1$. For S_{-36} to be 0, C_{-37} must be zero.

FT Action

Table 6.6.6-2 lists the activities during the FTs to shift or clear the smaller number. The "C" and "S" bits refer to AN1. The SC and SC* signals must be precisely understood to follow the action.

SC = SAT means SC* = 000 001 or 000 000. It will, of course, reach 1 first when counting down.

SC $\neq$ SAT means SC* > 000 001.

SC* = 0 means SC = SAT $\cdot$ SC* 2^0 = 0
or SC* = 000 000.

SC = 0 means SC = 000 000.

(See Figure 6.5-1)

Shifting D^*_M

As the original mantissa value has been transferred to D^*_M during IT4, the shift cycle starts with the number in D^*_M. The process for each set of FTs except the last is: clear D_M, (D^*_M) to D_M shifted right, clear D^*_M, (D_M) to D^*_M. The shift counter is decremented and examined. During the last cycle (D_M) are not transferred to D^*_M. This is to cover the eventuality of the numbers originally being equal. If equal, D^*_M should not be shifted. By inhibiting the $(D_M) \longrightarrow D^*_M$ transfer when SC* = 0, the activity for equal numbers will result in one FT cycle with the contents of D^*_M remaining unaltered.

During the first cycle, FI = 000 and FI* = 010 control; during any succeeding cycles FI, FI* = 010 control.

Shifting A_M

When (A_M), as the smaller number, is to be shifted, it is necessary to compensate for the existing situation that the number in SC, SC* is one less than the actual number of places to be shifted.

	Shift D_M	Clear D_M		Shift A_M	Clear A_M	
First Cycle	$\pm D_E \leqq \pm A_E \leqq (\pm D_E + 35)$	$\pm A_E > (\pm D_E + 35)$ $+A_E < (-D_E + 2048)$	$+A_E \geqq (-D_E + 2048)$	$\pm A_E < \pm D_E \leqq (\pm A_E + 35)$	$\pm D_E > (\pm A_E + 35)$ $\pm D_E \leqq (-A_E + 2048)$	$+D_E > (-A_E + 2048)$
FT1 (Trigger D49)	$FI = 000 \cdot C_{-36} = C_{-37} \cdot S_{-36} = 0 \cdot \lvert S \rvert \leqq 35 \Rightarrow (2 \rightarrow FI^*)$	$FI = 000 \cdot C_{-36} = C_{-37} \cdot S_{-36} = 0 \cdot \lvert S \rvert > 35 \Rightarrow (6 \rightarrow FI^*)$	$FI = 000 \cdot C_{-36} = 0 \cdot C_{-37} = 1 \Rightarrow (6 \rightarrow FI^*)$	$FI = 000 \cdot C_{-36} = C_{-37} \cdot S_{-36} = 1 \Rightarrow (1 \rightarrow A^*_M)$ $FI = 000 \cdot C_{-36} = C_{-37} = S_{-36} = 1 \Rightarrow [(FI) \rightarrow FI^*]$	$FI = 000 \cdot C_{-36} = C_{-37} \cdot S_{-36} = 1 \cdot \lvert S \rvert > 35 \Rightarrow (7 \rightarrow FI^*)$	$FI = 000 \cdot C_{-36} = 1 \cdot C_{-37} = 0 \Rightarrow (7 \rightarrow FI^*)$
	$FI = 000 \cdot C_{-36} = C_{-37} \cdot S_{-36} = 0 \Rightarrow (0 \rightarrow D_M)$					
	$FI = 000 \Rightarrow (\lvert S \rvert \rightarrow SC, SC^*)$					
FT2 Trig. AN1ECC	$FI = 000 \cdot C_{-36} = C_{-37} \cdot S_{-36} = 0 \Rightarrow [(D^*_M) \xrightarrow{R} D_M], [d^*_o \rightarrow d_o]$			$FI = 000 \cdot C_{-36} = C_{-37} \cdot S_{-36} = 1 \Rightarrow [(A_M) \rightarrow A^*_M]$		
FT3	$FI^* = 01X \Rightarrow [(SC^* - 1) \rightarrow SC]$ $FI^* \neq 000 \Rightarrow [(FI^*) \rightarrow FI]$ $FI^* = 010 \cdot SC^* \neq 0 \Rightarrow (0 \rightarrow D^*_M)$	$FI^* = 110 \Rightarrow 0 \rightarrow D^*_M$ $FI^* = 110 \Rightarrow 0 \rightarrow AN1C$		$FI^* = 000 \Rightarrow (3 \rightarrow FI)$ $FI^* = 000 \Rightarrow (0 \rightarrow A_M)$ $FI^* = 000 \Rightarrow (0 \rightarrow A_E)$	$(FI^*) \rightarrow FI$ $FI^* = 1X1 \Rightarrow (0 \rightarrow A_M, A_E)$	
FT4	$FI^* = 010 \cdot SC^* \neq 0 \Rightarrow [(D_M) \rightarrow D^*_M]$ $FI^* = 01X \cdot SC \neq SAT \rightarrow FT1$ $FI^* = 01X \cdot SC = SAT \rightarrow AT1$	$FI^* = 110 \Rightarrow AT1$		$FI^* = 000 \Rightarrow [(D_E) \rightarrow A_E]$ $FI^* = 000 \Rightarrow [(A^*_M) \xrightarrow{R} A_M]$ $FI^* = 000 \Rightarrow (a^*_o \rightarrow a_o)$ $FI^* = 000 \cdot SC \neq 0 \rightarrow FT1$ $FI^* = 000 \cdot SC = 0 \rightarrow AT1$	$FI^* = 111 \Rightarrow [(D_E) \rightarrow A_E]$ $FI^* = 111 \Rightarrow AT1$	
Other Cycles FT1	$\overline{FI = X0X} \Rightarrow [(FI) \rightarrow FI^*]$ $FI = 01X \Rightarrow [(SC) \rightarrow SC]$ $FI = 010 \Rightarrow (0 \rightarrow D_M)$			$\overline{FI = X0X} \Rightarrow [(FI) \rightarrow FI^*]$ $FI = 01X \Rightarrow [(SC) \rightarrow SC^*]$ $FI = 011 \Rightarrow (1 \rightarrow A^*_M)$		
FT2	$FI = 010 \Rightarrow [(D^*_M) \xrightarrow{R} D_M], [d^*_o \rightarrow d_o]$			$FI = 011 \Rightarrow [(A_M) \rightarrow A^*_M]$		
FT3	$FI^* = 01X \Rightarrow [(SC^* = 1) \rightarrow SC]$ $FI^* \neq 000 \Rightarrow [(FI^*) \rightarrow FI]$ $FI^* \neq 010 \cdot SC^* \neq 0 \Rightarrow (0 \rightarrow D^*_M)$			$FI^* = 01X \Rightarrow [(SC^* - 1) \rightarrow SC]$ $FI^* \neq 000 \Rightarrow [(FI^*) \rightarrow FI]$ $FI^* = 011 \Rightarrow (0 \rightarrow A_M, A_E)$		
FT4	$FI^* = 010 \cdot SC^* \neq 0 \Rightarrow [(D_M) \rightarrow D^*_M]$			$FI^* = 011 \Rightarrow [(A^*_M) \xrightarrow{R} A_E]$ $FI^* = 011 \Rightarrow (a^*_o \rightarrow a_o)$ $FI^* = 011 \Rightarrow [(D_E) \rightarrow A_E]$		

$FI^* = 01X \cdot SC = SAT \Rightarrow FT1$
$FI^* = 01X \cdot SC = SAT \Rightarrow AT1$

Table 6.6.6-2 Arranging Smaller Floating Point Number for Add or Subtract

SC is not decremented during the first FT cycle, only during the succeeding ones.

(A_M) require to be shifted at least once under this control. If the numbers were equal, the control would be FI = 2; (A_M) is shifted only when smaller.

A_E is cleared and $(D_E) \longrightarrow A_E$

<u>Clearing A_M or D_M</u>

In either case only one set of FTs is required. The logic is straightforward as is shown in Table 6.6.6-2.

If $D_E \gg A_E$, the A register is cleared and information must be transferred to A from D as the resulting sum or difference. $(D_E) \longrightarrow A_E$ in FT4 and then control is passed to the algorithm timings for the proper transfer of the mantissa to A_M through AN2.

If $A_E \gg D_E$, the result is already in A. However, to cope with the eventuality that the augend, or minuend, may not be in normalized form, the control also proceeds from FT4 $\longrightarrow$ AT1.

During the algorithm timings, no change of (A_M) is effected as (D^*_M) was cleared to zero in FT3. However, the test for a normalized number will be made during the algorithm timings. And if necessary (A) will be normalized.

Algorithms, F.P. Addition, Subtraction

These are similar to fixed point add or subtract with two exceptions. Only the mantissas are added. If overflow occurs, it may be corrected by changing the exponent of the sum.

$$\begin{array}{rccl}
 & 2^{-1} & \text{X} & 2^{10} \\
+ & 2^{-1} & \text{X} & 2^{10} \\
\hline
 & 2^{0} & \text{X} & 2^{10} \text{ correct to } 2^{-1} \text{ X } 2^{11} \\
 & -2^{0} & \text{X} & 2^{10} \\
+ & -2^{0} & \text{X} & 2^{10} \\
\hline
 & -2^{1} & \text{X} & 2^{10} \text{ correct to } -2^{0} \text{ x } 2^{11}
\end{array}$$

The correction can be performed only if it does not cause the capacity of the 11-bit exponent to be exceeded. If correction develops an exponent greater than 2047, the exponent overflow will send 1 $\longrightarrow$ EF (Exponent Fault Flipflop) and the computer will jump to a correction routine.

AN1 is separated by floating point control into two parts, AN1$_M$ and AN1$_E$. AN1C = 1 will carry-in a one to bit $_{-35}$; AN1CE = 1 to bit $_{-47}$. There are two carry-complete signals, AN1CC and AN1ECC.

6.6.7 Floating Point Multiplication

The algorithm of these instructions is generally the same as fixed point multiplication. Floating point control covers the three additional types of operations required for floating point multiplication: addition of exponents, normalizing the product, and dealing with the various cases of unrepresentable products. The sequence of operations for the instruction is add exponents, multiply mantissas, normalize or correct product if necessary.

Table 6.6.7-1 lists the various possible cases of products. (In double-length multiplication, the comments for A apply to Q.) Case 1 is the type of product resulting from most instructions. The product is representable by the register capacity.

Case 2 is the product whose value is too small to be represented by the computer as a normalized number, $P < 2^{-1} X 2^{-2048}$. The quantity is close enough to be considered as zero. The A register is made zero, the instruction ends immediately after the exponent addition with the (A) = 0 and UF = 1 and EO = 1. The computer does not halt, the two flip-flops are reset to zero in IT1 of the next instruction. This product is not a fault, merely small enough to be made equal to zero.

In Case 3, addition of exponents resulted in exponent overflow. The product may, however, still be representable in the computer after normalizing, for example:

$$(2^{-1} X 2^{1024}) X (2^{-1} X 2^{1024}) = 2^{-1} X 2^{2047}$$

The fact that exponent overflow occurred is stored by $1 \longrightarrow EO$, $1 \longrightarrow EF$; the mantissas are multiplied and normalization is attempted. To successfully normalize, the exponent must be reduced below +2048. This can be detected by the "underflow" indication $c_{-36} = 1$, $c_{-37} = 0$.

Case	Logic of Detection	Procedure	Product Value	Indications at End
1. Sum of exponents is within capacity of computer $-2048 \leq S_E \leq +2047$	$C_{-36} = C_{-37}$	Do multiplication, possibly followed by normalizing or correction.		
2. Sum of exponents produces exponent underflow $S_E < -2048$	$C_{-36} = 1 \ . \ C_{-37} = 0$	Product too small, make it zero. ($A_M = 0.00\ldots0$ $A_E = 1.00\ldots.0$) Do not multiply, end instruction, no halt.	$A_M = 0$ $A_E = 2^{11}$	UF = 1 EO = 1
3. Sum of exponents produces exponent overflow $S_E > +2047$	$C_{-36} = 0 \ . \ C_{-37} = 1$	May be compensated during normalizing or correction. Store knowledge of OVF, $1 \longrightarrow$ EO, EF and proceed to multiply.		

4. Multiplication where multiplier = -1	QC = 0 At End	Normalize if necessary, otherwise end instruction. If ending here and previous exponent overflow, jump*
		If normalizing does not reduce exponent below +2048, jump*
		If normalizing results in zero mantissa and still $A_E > +2047$,
		If no previous fault but normalizing produces underflow, end instruction at this point with unnormalized number,
		If normalizing, previous exponent overflow and now underflow, remove fault. Number normalized and in range.
			$A_M \neq 0$ $A_E \geq 2^{11}$
			$A_M \neq 0$ $A_E \geq 2^{11}$
			$A_M = 0$ $A_E \geq 2^{11}$
			$-1/2 < A_M < +1/2$ $A_E = 2^{11}$
				EO = 1 EF = 1
				EO = 1 EF = 1
				EO = 1 EF = 1
				UF = 1 EO = 0
				EO = 1

5A. N x -1 where $N \neq -1$	In 35th add cycle $QC = 1 \ . \ q_o = 1 \ . \ q^*_{-35} = 0$	Do force add cycle, then normalize if necessary. Possible results as in 4 above.		
5B. -1 x -1	During force add $C_o \neq C_{-1}$	Do correction (FI = 101). If exponent overflow now or previously, jump*	$A_M = +1/2$ $A_E > +2047$	EO = 0 or 1 EF = 1

* Jump to memory location 00000 to permit entering a programmed correction routine.

Cases of Floating Point Products

Table 6.6.7-1

The process of subtracting one from A_E is:

$$\begin{array}{llll} & A_E & = +2048 & = 100\ 000\ 000\ 000 \\ + & & & + \\ & D^*_E & = -1 & = \underline{111\ 111\ 111.111} \end{array}$$

with the underflow type of carries from $ANIC_{-36}$ and $ANIC_{-37}$.

If this occurs during the 36 shifts of normalizing, the exponent fault is cancelled, $0 \longrightarrow EF$. Otherwise the instruction ends with an indicated fault, a zero mantissa, and a "negative - appearance" exponent.

Case 4 is the algorithm for all mantissas except when the multiplier is -1. After 35 add cycles, control goes to FI = 100 for normalizing. A previous exponent overflow may or may not be compensated. Exponent underflow may result during normalizing.

In case 5A a multiplier of -1 requires the force add cycle. This ends the instruction if normalized operands were used. A previous exponent overflow would still be indicated by the exponent fault.

The floating point logic accommodates the possibility that non-normalized numbers may be used as arithmetic operands. After the force add cycle control will go to FI = 100, if the product is not in normalized form ($a^*_0 = a^*_{-1}$).

Case 5B is -1 X -1. The mantissa overflow can be corrected (FI = 101), if there was no previous exponent overflow and correction does not produce exponent overflow. Otherwise the instruction ends with EF = 1, a mantissa of 0.100 .... 0 and the "negative-appearing" exponent.

<u>Logic of Exponent Addition (FI = 001)</u>

During the ITs the two exponents are transferred to A_E and D^*_E for their addition. To simplify SC inputs the $36 \longrightarrow SC$ signal is used both for normalizing and algorithm recycling.

The floating point multiplication algorithm only requires 35 cycles, so SC is counted down one during these FTs and the algorithm is begun with SC = 35.

FT1		trigger D49
	Clear A^*_E	$1 \longrightarrow A^*_E$
	If exponent sum underflows, prepare to end	$1 \longrightarrow$ UF, EO
	If exponent sum overflows, may be compensated, note overflow	$1 \longrightarrow$ EF, EO
	Control activity	$(FI) \longrightarrow FI^*$
	Control activity	$(SC) \longrightarrow SC^*$
FT2		trigger AN1ECC
	Transfer exponent sum	$(A_E + D^*_E) \longrightarrow A^*_E$
FT3		
	If underflow, clear A	$0 \longrightarrow A_M, A_E$
	Control activity	$(FI^*) \longrightarrow FI$
	Count down SC to 35	$(SC^* - 1) \longrightarrow SC$
FT4		
	If no underflow, transfer exponent sum	$(A^*_E) \longrightarrow A_E$
	If no underflow, and double-length	$(A^*_E) \longrightarrow Q_E$
	If no underflow, go to algorithm	$\longrightarrow$ AT1
	If underflow, set exponent to zero	$1 \longrightarrow a_{-36}$
	If underflow, end instruction	$\longrightarrow$ END

Logic of Multiply Algorithm (AI = 01X)

The operations are very similar to fixed point multiplication. Only the mantissas of A and D* are connected to AN1. The other differences are noted below:

AT1

reset FI (FP = 1) $0 \longrightarrow FI$

AT2

If overflow during force add, do correction

$$AI = 00X \;\cdot\; FP = 1 \;\cdot\; C_o \neq C_{-1} \Longrightarrow 5 \longrightarrow FI$$

AT3

If force add required after 35th cycle:

$$SC = SAT \;\cdot\; QC = 1 \;\cdot\; q_o = 1 \;\cdot\; q_{-35} = 0 \;\cdot\; FP = 1$$
$$\Longrightarrow 0 \longrightarrow AI$$

Algorithm completed, normalization required:

$$SC = SAT \;\cdot\; QC = 0 \;\cdot\; d^*_o = a^*_o \;\cdot\; FP = 1$$
$$\Longrightarrow 4 \longrightarrow FI$$

(a^*_o at this time is the MSD of the product;
d^*_o is equal to the product sign.

Normalization required next, multiplier is 0, or + 1/2:

$$SC = SAT \;\cdot\; d^*_o = a^*_o \;\cdot\; (q^*_o = 0 \vee q^*_{-35} = 1)$$
$$\cdot\; FP = 1 \Longrightarrow 4 \longrightarrow FI$$

(This extra logic is required for (Q) = ± 1/2 as QC still is 1 in this timing.

$0 \longrightarrow$ QC occurs during this AT3 timing. When (Q) = 0, an operation under normalization control is required to develop the standard form of zero.)

Multiplier bit is 1, reset QC

$$AI^* = 01X \cdot FP = 1 \cdot q^*_{-35} = 1 \Longrightarrow 0 \longrightarrow QC$$

AT4

Do force add next, $(QC = 1 \cdot q_o = 1) \Longrightarrow \longrightarrow AT1$

Do normalization or correction next.

$FI \neq 000 \Longrightarrow \longrightarrow FT1$

*Add 1 to A_E for correction. $0 \longrightarrow D^*_E$, $1 \longrightarrow ANICE$

*Subtract 1 from A_E for normalizing.
$1 \longrightarrow D^*_E$, $0 \longrightarrow ANICE$

*(No timing signal at these gates. The earliest they can be done is AT4.)

Shift limit for normalizing which follows

$$(AI^* = 00X \vee SC = SAT) \cdot FI = 100 \cdot \overline{(QC = 1 \cdot q^*_o = 0)} \Longrightarrow 36 \longrightarrow SC$$

Product already in normalized form, end instruction (FI = 000) $\longrightarrow$ END

Logic of Normalizing after Multiply

The logic is the same as for normalizing after add or subtract with the additional logic for previous exponent overflow (during addition of the exponents). This additional logic is:

FT1

Reducing exponent now (A_E - 1) causes exponent to pass below +2048 which compensates for previous overflow:

$$FI = 100 \cdot EO = 1 \cdot c_{-36} = 1 \cdot c_{-37} = 0 \Longrightarrow 0 \longrightarrow EF$$

Exponent underflow without previous overflow. Stop normalizing and end instruction:

$$FI = 100 \cdot EO = 0 \cdot c_{-36} = 1 \cdot c_{-37} = 0 \Longrightarrow 1 \longrightarrow UF$$

FT2

(No additional logic.)

FT3

If no underflow now, clear A

$$FI^* = 100 \cdot UF = 0 \Longrightarrow 0 \longrightarrow A_M, A_E$$

FT4

If no underflow now and less than 36 cycles

($FI^* = 100 \cdot UF = 0 \cdot SC \neq SAT$)

1. Transfer decreased exponent $(A^*_E) \longrightarrow A_E$
2. Shift A_M $(A^*_M) \xrightarrow{L} A_M$

$$a^*_o \longrightarrow a_o$$

If 36th cycle, standard zero

$$FI^* = 100 \cdot SC = SAT \Longrightarrow 1 \longrightarrow a_{-36}$$

If number now normalized ($a^*_o \neq a^*_{-2}$) $\longrightarrow$ END

(a^*_{-2} will be MSD of the shifted mantissa)

If number still not normalized, no underflow
and less than 36 shifts, recycle

$AI = 01X \cdot FI^* = 100 \cdot a^*_o = a^*_{-2} \cdot UF = 0 \cdot SC \neq SAT$

$\Longrightarrow \longrightarrow$ FT1

If 36th shift, end instruction

$FI^* = 100 \cdot SC = SAT \cdot II = 0 \Longrightarrow \longrightarrow$ END

(II = 0 is used to distinguish this from
the MAD, MSU instructions)

If underflow, end instruction

$FI^* = 100 \cdot UF = 1 \cdot II = 0 \Longrightarrow \longrightarrow$ END

Logic for Correction, Multiply

The logic is the same as for add, subtract. Correction can be accomplished in one cycle. If exponent overflow occurs, EF is set to 1. In the overflow case, the corrected mantissa and overflowed exponent will be in the register.

6.6.8 Divide, Floating Point

The algorithm logic of floating point division is almost identical to that of fixed point division. The floating Point control for division is separated into its various parts for the introductory description below. This is followed by a line-by-line explanation of the floating point logic.

Floating point control initially causes subtraction of the exponents. The FI = 001 state is used. Then, if the mantissas have the proper magnitude relationship:

i.e. $|A_M| < |D_M|$, the division algorithm is performed.

Normalizing is not required in these instances as the quotient lies in the range of 1/2 to 1. With normalized operands used for the algorithm, the smallest quotient would be produced by $1/2 \div 1$.

FI = 001

The set of FTs prior to the algorithm, during which the exponents are subtracted, is also used to perform the magnitude comparison similar to the first cycle of fixed point division. During the FTs the mantissas are compared and if $|A_M| < |D_M|$, the former is shifted left (as is the Q register) and the first quotient digit transferred to q_{-35}.

The control state is FI = 001 (addition or subtraction of exponents). There is no need for AI = 100 action. During IT2, AI is set to 5 instead.

In addition to the above activity, the SC is counted down one, to 35, and the control passes from FT4 to AT1 for the 35 cycles of the algorithm.

Dividend Correction

When operating with normalized numbers there is as great a probability the dividend will be larger than the divisor as the probability it will be smaller. This is also the case if the operands are not normalized.

Unlike fixed point division, the instances of a floating point dividend being the larger may be corrected prior to the algorithm and then division can be performed. The dividend mantissa is shifted right one place at a time, and compared, until it is smaller than the divisor. The quotient exponent (subtraction of exponents having already occurred) is increased in a corresponding manner.

With normalized numbers, one right shift will make the dividend smaller than the divisor. Unnormalized numbers may require a greater number of shifts.

A limit of 36 shifts is set for correction. If the dividend must be shifted clear to zero to have it "smaller" than the divisor, the divisor itself is zero. The instruction can be ended without the algorithm.

The quotient resulting from the algorithm using a corrected dividend will be in normalized form.

<u>FI = 101</u>

The FI control used for correction is, of course, FI = 101 as the action is similar to correction for overflow in other arithmetic. Involved is the right shifting of a mantissa and adding 1 to an exponent.

FI = 001 in the first FT1 after the ITs. If $d^*_o \neq s_o$, the dividend is the larger and $5 \longrightarrow$ FI*. The dividend mantissa will be shifted right during the remainder of this set of FTs.

The AN1 exponent inputs, at this time, are those for subtraction of the operand exponents. This is done and the difference transferred to A^*_E. Thereafter the contents of D* and AN1CE may be changed to provide for the addition of 1 to the quotient exponent.

To recapitulate, if correction is necessary, during the first FT cycle, FT1 and FT2 are under the control of FI = 001. Then FT3 and FT4 are under the control of FI* = 101.

In all succeeding FT1's, FI = 101 controls. One is added to (A^*_E). If the dividend is now smaller, $1 \longrightarrow$ FI* and during FT3 and FT4 preparations are completed to proceed to the algorithm. These are the same as is done for a dividend that did not require correction (see FI = 001). Otherwise another right shift is performed and SC counted down and recycle.

As the number of correction shifts is not predetermined, the logic of FI = 101 resets SC to 35 prior to the algorithm.

When the divisor is zero, the active SC = SAT signal sets the EF flip-flop to 1 and provides the exit to END.

Normalizing

A normalizing procedure does exist to permit manipulation of quotients resulting from the division of unnormalized numbers that may be stored in the computer and some special cases of quotients in a ones complement form.

Normalizing subsequent to division is unique as both a quotient and a remainder must be treated. A method which will preserve the accuracy of both parts of the answer is to continue the division process until the quotient assumes a normalized form. For this method of normalizing, after 35 cycles of ATs for the standard length of division, a sequence of both FTs and ATs are used until the number is normalized. During each set of the FTs the quotient exponent is decreased by 1 and the remainder is shifted left. This is followed by a set of ATs, where division is extended one place and the quotient again judged. Normalizing is controlled by FI = 100. The signal (FI* = 100 • AI = 101) identifies division for normalizing during the algorithm. ($D*_E$) is changed to 1's and AN1CE to 0 for decreasing the quotient exponent.

Overflow and Underflow

When the difference of the operand exponents exceeds +2047, the algorithm is performed with the assumption that normalizing may subsequently reduce the quotient exponent to a representable quantity. Under FI = 001 or FI = 101, $1 \longrightarrow$ EF if overflow. This can be removed ($0 \longrightarrow$ EF) if during FI = 100 operation, the exponent again passes +2048 ($c_{-36} = 1 \cdot c_{-37} = 0$) in a decreasing direction.

A corrected dividend will produce a normalized quotient. Overflow in this case cannot be compensated. The algorithm is however performed to produce the quotient. EF remains set to 1 at the end of the instruction to indicate the fault.

When the quotient exponent is smaller than -2048, the quotient is considered effectively zero. The division algorithm is not necessary as this is determined in FI = 001 or FI = 101 operations prior to the AI = 101 algorithm. The increase of the quotient exponent during correction may bring it in range again.

Underflow is not considered a fault, merely a quantity close enough to zero to be actually made zero.

Table 6.6.8-1 summarizes the register contents and visual indications resulting from division faults.

Case	Result	Registers at End	Visual Indications
Fixed Point DD > DV	Do not divide. Shift DD right. Set OVF For divide & store instructions, (Q) $\longrightarrow$ V and DV is lost.	(A) = DD x 2^{-1} (Q) = 0 or minor half of DD x 2^{-1}	OF = 1
Floating Point			
$(QT_E) < -2048$ after any correction if necessary.	Do not divide. Make QT & R zero. Set UF	$(A_M) = 0$ $(A_E) = -1$ $(Q_M) = 0$ $(Q_E) = -1$	UF = 1
DV = 0	Do not divide. End instruction and jump to memory location zero.	(A_M) = all bits have same value as original value of a_o $(A_E) = DD_E - DV_E + 35$ (Q_M) = all bits have same value as original value of q_o $(Q_E) = 0$	EF = 1
$(QT_E) > +2047$ when QT is in normalized form	Do division. After End, jump to memory location zero.	(A) = R (Q) = QT	EF = 1

DD	= Dividend
DV	= Divisor
QT	= Quotient
R	= Remainder

Division Faults

Table 6.6.8-1

Logic of Floating Point Division

In addition to the logic for all division instructions, the following occurs for floating point during the ITs. (See fixed point division logic for remaining activity.)

IT1	Clear FI		$0 \longrightarrow FI$
IT2	Set algorithm control for division and floating point control for exponent subtraction.		$5 \longrightarrow AI$ $1 \longrightarrow FI$
IT4	Twos complement of D_E to $D*_E$ for subtraction		$(D_E)' \longrightarrow D*_E$, $1 \longrightarrow AN1CE$ $IT4 \longrightarrow FT1$

FT1 Trigger: AN1CC · A10a

Prepare shift of remainder.	$FI = 001 \cdot AI = 001$	$1 \longrightarrow A*_M$
Prepare shift of remainder.	$FI = X01 \cdot AI = 001$	$1 \longrightarrow Q*_M$
Prepare to transfer exponent difference.	$FI = 001$	$1 \longrightarrow A*_E$
Prepare to count down SC.	$FI = 001$	$(SC) \longrightarrow SC*$
$\lvert A_M \rvert < \lvert D_M \rvert$ do division.	$FI = 001 \cdot AI = 101 \cdot d*_o = s_o$	$(FI) \longrightarrow FI*$
$\lvert A_M \rvert > \lvert D_M \rvert$ correct (A_M) before dividing.	$FI = 001 \cdot AI = 101 \cdot d*_o \neq s_o$	$5 \longrightarrow FI*$

FT2 Trigger: AN1ECC

Prepare to shift remainder	$(FI = 001 \cdot AI = 101) \vee FI = 10X$	$(A_M) \longrightarrow A*_M$
Prepare to shift remainder	$FI = X01 \cdot AI = 101$	$(Q_M) \longrightarrow Q*_M$

FT2 (Continued)

Transfer quotient exponent.	$FI = 001 \vee FI = 10X$	$(A_E + D^*_E) \longrightarrow A^*_E$
Exponent subtraction produces underflow. If dividend is not to be corrected, make quotient zero and end instruction.	$FI = 001 \cdot c_{-36} = 1 \cdot c_{-37} = 0$	$1 \longrightarrow EO, UF$
Exponent subtraction produces overflow. Note fault and continue with division. (If normalizing later brings exponent into range, remove fault).	$FI = 001 \cdot c_{-36} = 0 \cdot c_{-37} = 1$	$1 \longrightarrow EO, EF$

FT3

Prepare to shift remainder.	$(FI^* = 00X \cdot AI = 101) \vee FI^* = 1X1$	$0 \longrightarrow A_M$
Prepare to shift remainder.	$(FI^* = 00X \cdot AI = 101) \vee (FI^* = 101 \cdot AI = 101)$	$0 \longrightarrow Q_M$
Prepare for quotient exponent.	$FI^* = 001 \vee FI^* = 1X1$	$0 \longrightarrow A_E$
Prepare for quotient exponent.	$FI^* = 001 \vee FI^* = 101$	$0 \longrightarrow Q_E$
Correcting A_M, count down SC.	$FI^* = 101 \cdot AI = 101$	$(SC^*_{-1}) \longrightarrow SC$
Division algorithm next, prepare to set SC to 35.	$FI^* = 00X \cdot AI = 101$	$0 \longrightarrow SC$
Floating Point Control	$FI^* \neq 000$	$(FI^*) \longrightarrow FI$

FT4

Division algorithm next:

1st cycle shift of remainder.		$(A^*_M) \xrightarrow{L} A_M$, $(Q^*_M) \xrightarrow{L} Q_M$, $a^*_{-1} \longrightarrow a_o$, $q^*_o \longrightarrow q_o$, $q^*_{-1} \longrightarrow a_{-35}$
Transfer quotient exponent	FI* = 001 • AI = 101 • UF = 0	$(A^*_E) \longrightarrow Q_E$
Transfer first quotient bit.		$QD \longrightarrow q_{-35}$
Set SC for algorithm.		$35 \longrightarrow SC$
Proceed to algorithm	FI* = 001 • UF = 0	$\longrightarrow$ AT1
Transfer quotient exponent.	(FI* = 001 • UF = 0) v FI* = 101	$(A^*_E) \longrightarrow A_E$
Correct (decrease) dividend	FI* = 101	$(A^*_M) \xrightarrow{R} A_M$
Correct (decrease) dividend	FI* = 101 • AI = 101	$(Q^*_M) \xrightarrow{R} Q_M$, $q^*_o \longrightarrow q_o$, $a^*_o \longrightarrow a_o$, $a^*_{-35} \longrightarrow q_{-1}$
Prepare to increase quotient exponent.	FI* = 101	$(A^*_E) \longrightarrow A_E$
Prepare to add 1 to exponent.	FI = 101 • C_6 = 1 (see Note 1)	$0 \longrightarrow D^*_E$, $1 \longrightarrow$ AN1CE
Recycle to increase exponent and test corrected dividend.	FI* = 101 • AI = 101 • SC $\neq$ SAT	$\longrightarrow$ FT1

Note 1. No timing on these gates. As soon as FI* = 101 has been transferred to FI (see FT3), D^*_E and AN1CE will start to change. Its trigger will prevent the turn-on of the following FT1 timing until the register has changed. C_6 is the command bit coding for arithmetic. Similarly for 4 $\longrightarrow$ FI in AT3.

FT4 (Continued)

Quotient too small (exponent underflow) and not (or no longer) correcting dividend.		
Form standard zero for quotient and remainder and end instruction.	$FI^* = 00X \cdot AI = 101 \cdot UF = 1$	$1 \longrightarrow q_{-36}$
	$FI^* = 001 \cdot UF = 1$	$1 \longrightarrow a_{-36}$
	$FI^* = 001 \cdot UF = 1 \cdot II = 0$	$\longrightarrow$ END

When the dividend is being corrected, another set of FTs follow:

FT1

Prepare to shift dividend.	$FI = 10X$	$1 \longrightarrow A^*_M$
	$FI = X01 \cdot AI = 101$	$1 \longrightarrow Q^*_M$
Prepare to increase quotient exponent.	$FI = 10X$	$1 \longrightarrow A^*_E$
Prepare to count down SC.	$FI = 101 \cdot AI = 101$	$(SC) \longrightarrow SC^*$
If now $\lvert A_M \rvert < \lvert D_M \rvert$. Treat (A_M) as just prior to algorithm.	$FI = 101 \cdot AI = 101 \cdot d^*_o = s_o$	$1 \longrightarrow FI^*$

FT2

Increasing quotient exponent now brings exponent that underflowed on subtraction into range again, remove fault.	$FI = 101 \cdot C_{-36} = 0 \cdot C_{-37} = 1$	$0 \longrightarrow UF$

FT2 (Continued)

Increasing quotient exponent causes overflow, note fault. $FI = 101 \cdot C_{-36} = 0 \cdot C_{-37} = 1 \cdot UF = 0$ $1 \longrightarrow EF$

The remaining activity of FT2, FT3 and FT4 is the same as the logic listed for the first set of FT's. If the dividend can be made smaller than the divisor, the algorithm is then performed. Note that SC is set to 35 in the FT4 just prior to AT1. If the divisor is zero, 36 FT cycles will occur and at FT4:

$FI^* = 101 \cdot AI = 101 \cdot SC = SAT$ $1 \longrightarrow EF$

$FI^* = 101 \cdot AI = 101 \cdot SC = SAT$ $\longrightarrow END$

Normalizing the Quotient

AT3

Set normalizing control	$[(AI^* = 101 \cdot SC = SAT) \vee (AI^* = 101 \cdot FI^* = 100 \cdot FP = 1)] \cdot q^*_{-1} = q^*_{-2}$	$4 \longrightarrow FI$

AT4

Set SC for normalizing.	$FI = 100 \cdot SC = SAT \cdot \overline{(QC = 1 \cdot q^*_o = 0)}$	$36 \longrightarrow SC$
Proceed to FT's.	$AI^* = 101 \cdot SC = SAT \cdot FI \neq 000$	$AT4 \longrightarrow FT1$
Prepare to decrease quotient exponent.	$FI = 100 \cdot C_6 = 1$ (see Note 1)	$1 \longrightarrow D^*_E$, $0 \longrightarrow AN1CE$

FT1

		Trigger $AN1CC \cdot AI = 101$
Prepare to shift remainder.	$FI = 10X$	$1 \longrightarrow A^*_M$
Prepare to decrease quotient exponent (which was left in A_E prior to algorithm).	$FI = 10X$	$1 \longrightarrow A^*_E$
Prepare count down of SC.	$FI = 100 \cdot AI = 101$	$(SC) \longrightarrow SC^*$
Control	$FI = 100$	$(FI) \longrightarrow FI^*$

FT2

Transfer reduced quotient exponent.	$FI = 10X$	$(A_E + D^*_E) \longrightarrow A^*_E$
Transfer remainder.	$FI = 10X$	$(A_M) \longrightarrow A^*_M$
If decreasing exponent will cause underflow, note and prevent subsequent changes of quotient and remainder.	$FI = 100 \cdot C_{-36} = 1 \cdot C_{-37} = 0 \cdot E0 = 0$	$1 \longrightarrow UF$
Normalizing eliminates previous exponent overflow, remove fault.	$FI = 100 \cdot C_{-36} = 1 \cdot C_{-37} = 0 \cdot E0 = 1$	$0 \longrightarrow EF$

FT3

Prepare shift of remainder	$FI^* = 100 \cdot UF = 0$	$0 \rightarrow A_M$
and transfer of exponent.	$FI^* = 100 \cdot UF = 0$	$0 \rightarrow A_E$
	$FI^* = 100 \cdot AI = 101 \cdot UF = 0$	$0 \rightarrow Q_E$
If quotient too small, make it zero.	$FI^* = 100 \cdot AI = 101 \cdot SC = SAT$	$0 \rightarrow Q_M$
Control	$FI^* \neq 000$	$(FI^*) \longrightarrow FI$

FT4

Transfer quotient exponent	$FI^* = 100 \cdot UF = 0 \cdot SC \neq SAT$	$(A^*_E) \longrightarrow A_E$
Transfer quotient exponent	$FI^* = 100 \cdot AI = 101 \cdot UF = 0 \cdot SC \neq SAT$	$(A^*_E) \longrightarrow Q_E$
Shift remainder.	$FI^* = 100 \cdot UF = 0 \cdot SC \neq SAT$	$(A^*_M) \xrightarrow{L} A_M$
Shift remainder.	$FI^* = 100 \cdot AI = 101 \cdot UF = 0 \cdot SC \neq SAT$	$a^*_{-1} \longrightarrow a_o$
Do another cycle of division.	$FI^* = 100 \cdot AI = 101 \cdot UF = 0 \cdot SC \neq SAT$	$\longrightarrow AT1$
Quotient mantissa too small at	$FI^* = 100 \cdot SC = SAT$	$1 \longrightarrow a_{-36}$
end of normalizing, make it zero.	$FI^* = 100 \cdot AI = 101 \cdot SC = SAT$	$1 \longrightarrow q_{-36}$
	$FI^* = 100 \cdot SC = SAT \cdot II = 0$	$\longrightarrow END$
Quotient exponent underflow during normalizing, leave as is.	$FI^* = 100 \cdot UF = 1 \cdot II = 0$	$\longrightarrow END$

6.6.9 Inhibit Clear of Overflow

The design of the S-2000 includes several facilities to provide methods of handling the occasions when the register capacity is exceeded. This overflow ($AN1C_0 \neq AN1C_{-1}$) can develop as a result of both arithmetic, and non-arithmetic instructions. It may be of significance in arithmetic, shift and certain index register instructions. To permit direction of overflow for these instructions, OF is cleared to zero before these instructions.

The optional procedures in the event of overflow are:

1. Ignore overflow and proceed to the next instruction. (OVERFLOW switch in OFF position.)
2. Halt computer unless the succeeding instruction is a jump conditional upon the state of the Overflow flip-flop. (OVERFLOW switch is ON.)

This clearing of OF at the start of these instructions can be inhibited to enable sensing of the development of overflow as a result of any of a group of instructions, such as a sub-routine, and postpone action thereon until the end of the sub-routine. The action will be done by the JNO or JOF instructions. The ICO instruction will inhibit the normal clearing of OF. (While using this method, the OVERFLOW switch is set to the OFF position.)

The ICO instruction sets the ICOF flip-flop to the "one" state. While ICOF = 1, it prevents an active $0 \longrightarrow$ OF signal from resetting the Overflow flip-flop. ICOF will be reset to zero during the JNO and JOF instructions, as will OF.

The logic of the ICO instruction is performed in four timings, IT1-IT4. The $1 \longrightarrow$ ICOF signal is active in IT2. The logic enables counting of an index register if a count of the performances of the instruction is desired. The J-bit coding of this instruction is not significant. In IT1 of ICO, itself, $0 \rightarrow$ OF. This enables a program to test for overflow as specifically occuring at some time subsequent to the performance of the ICO instruction.

6.7.1 Jump Instructions

A jump is effected by transferring to PA the address to which the program should jump for the next instruction, and then calling for a new instruction word immediately following the jump instruction by sending 1 to PI. The location of the next instruction in the new word is determined by the J bit of the jump instruction, which is sent to Mod 2.

As an aid in returning to the point of the program from which the jump was made, the adress of that point is stored in JA. It is the address of the instruction following in normal, consecutive sequence the address in which the jump instruction is located. A typical program method when jumping to a sub-routine would be to transfer the return address in JA to a jump instruction which is the exit from the sub-routine. The TJM instruction could be used. This is done at the start of the sub-routine.

During a conditional jump instruction, the return address is stored in JA regardless of whether the jump is executed. Therefore it cannot be said that the contents of JA are the return address for the last jump made. They are the return address for the last jump instruction.

The intent of the unconditional jump instruction, JMP, requires no explanation. The JBT instruction is also unconditional in nature. However, if the BREAKPOINT console switch is in the ON position, the computer will pause at the JBT instruction until the ADVANCE bar is depressed. If the BREAKPOINT switch is OFF, action of JBT and JMP is identical.

The conditional jump instructions base a decision to jump on either the value of one binary bit (sign of the number, odd or even, overflow) or on the result of a comparison of two numbers or words (equal to, equal to or greater than).

Classification of Jump Instructions

To illustrate how the logic of the jump instructions is organized, they can be divided into four groups, functionally. The three bit code, (C_4 C_5 C_6) is 010. The four bit code (C_0 C_1 C_2 C_3) is listed below. If $J = 0$, jump to I_0 and conversely.

Group 1 Unconditional Jumps

JMP 0000

JBT 000	0001	if BREAKPOINT switch is ON, wait until ADVANCE bar depressed before jumping.

Group 2. Conditional jumps requiring no register change operation.

JNO	0010	jump if no overflow
JOF	0011	jump if overflow
JAP	0100	jump if (A) are positive
JAN	0101	jump if (A) are negative
JDP	1100	jump if (D) are positive

Group 3. Conditional jumps requiring a register change operation.

JAZ	0001	jump if $(A) = 0$
JAEQ	0110	jump if $(A) = (Q)$
JAED	0111	jump if $(A) = (D)$
JAGD	1111	jump if $(A) \geq (D)$ alphanumeric sense
JAGQ	1110	jump if $(A) \geq (Q)$ algebraic sense
JAGQF	1101	jump if $(A) \geq (Q)$ floating point sense

Group 4. Conditional jumps including a circular shift of (Q).

JQP	1000	jump if (Q) are positive, unconditionally circular shift (Q) left one place.
JQN	1001	Jump if (Q) are negative, unconditionally circular shift (Q) left one place.

JQE	1010	jump if (Q) are even, unconditionally circular shift (Q) right one place.
JQO	1011	jump if (Q) are odd, unconditionally circular shift (Q) right one place.

Group 3 requires a transfer operation prior to the comparison for the jump decision as one of the numbers being compared must be placed in D*. Group 4 requires an algorithm operation after the jump decision and prior to jumping. Control during these instructions must include an AI operation for the circular shift.

Groups 1 and 2 require no transfers prior to the jump decision. The decision for the conditional jumps is based upon the value of one binary bit. The required activities could be accomplished within four timings of IT. If a jump is not to be executed these instructions end after IT4.

Groups 3 and 4 require more than four ITs due to the transfers or shifts. Therefore the general organization is to have all jump instructions proceed through IT8 when a jump is to be executed. During IT1 through IT4, the return address is placed in JA; groups 1, 2 and 4 jump decisions are made; group 3 performs its prior transfers of the word. During IT5 through IT7, group 3 makes its comparison for the jump decision; groups 1 and 2 perform no tasks. Group 4 proceeds from IT4 to a set of ATs for the circular shift and, if a jump is to be executed, returns from AT4 to IT7. In IT8 final preparations are made to execute the jump.

It is apparent that the jump decision must be stored for an interval during the instruction. JFF (Jump Flip Flop) is used for this purpose; JFF = 1 indicates that a jump is to be executed.

Table 6.7.1-1, "Organization of Jump Instructions" provides a survey of the activities for all of these instructions.

Breakpoint Jump

The BREAKPOINT switch, at the present time, is a two-position switch, ON and OFF. With the switch ON the computer will pause before entering IT1 of the JBT instruction. This is the sole unique part of the JBT logic.

	Tasks Common to All Jumps	JMP JBT	JNO	JOF	JAP	JAN	JDP	JAZ	JAED	JAEQ	JAGD	JAGQ	JAGQF	JQP	JQN	JQE	JQO
IT1	Prepare to place address to which to jump in PA Inh. AN2 0 → AN2C If $(I_V)'$ alone supplies address PR → SW6 + → PM 0 → SW4 If address is modified by (X) X → SW4 If in repeat mode and (X) alone is address 0 → PM Clear JA for return address 0 → JA	Clear AN1I 0 → AN1I	Clear ICOF 0 → ICOF					Inhibit AN1 for entire instruction 1 → AN1I		Prepare Transfer to D 1 → D	Prepare Transfer to D 1 → D			Clear AN1I 0 → AN1I			
IT2	Transfer to JA If SW2 = 0, an instruction word with the jump instruction as I_0 was just transferred (unless in repeat). Therefore address of the I_1 instruction is still in MA as it was not cleared in IT1 (MA) → JA 1 → ja_{-1} If SW2 = 1 (ja_{-1} was cleared to 0 in IT1) PA → JA Program control (SW2)' → Mod 2									Transfer $(Q) \xrightarrow{0} D$	Transfer $(Q) \xrightarrow{0} (D)$			7 → AI			
IT3	Prepare end-of-instruction operations If X modification involved 2 → PI Otherwise the existing logic will send 0, 1 or 3 to PI. This sets up program control if the conditional jump instruction does not jump. If a jump, in IT8 a 1 → PI will be generated	1 → JFF	If OVF = 0 1 → JFF	If OVF = 1 1 → JFF	If a_0 = 0 1 → JFF	If a_0 = 1 1 → JFF	If d_0 = 0 1 → JFF	Clear D* for comparison with (A) 0 → D* 0 → AN1C	Clear D* for comparison 0 → D* 0 → AN1C	Clear D* for comparison 0 → D* 0 → AN1C	Clear D* for a complement 1 → D* 1 → AN1C	Clear D* for a complement 1 → D* 1 → AN1C	Clear D* for a complement 1 → D* 1 → AN1C	If q_o = 0 1 → JFF	If q_o = 1 1 → JFF	If q_{-47} = 0 1 → JFF	If q_{-47} = 1 1 → JFF
IT4	Where a transfer was not required before the jump decision, the instruction can now be ended if no jump will occur. If a jump will occur or prior transfers necessary, the instruction continues to IT5	→ IT5	Decision made, clear OVF 0 → OVF Will Jump IT4 → IT5 Will not jump IT4 → END					→ IT5	Transfer to AN1 input Inh. AN1 (D) → D*	Transfer to AN1 input Inh. AN1 (D) → D*	Transfer complement to AN1 input Inh. AN1 (D)' → D*			Proceed to AI control for circular shift of Q IT4 → AT1 After shifting - AT4 → END Will jump END → IT7 Will not jump END → PT1			
IT5	(Action in JAGQF only)												Compare mantissa only Inh. AN1E and compensate exponents if necessary 1 → AN1CE				
IT6	No Action																
IT7	If jumping, Clear PA 0 → PA							If A = D* 1 → JFF		If A = D* 1 → JFF	If A ≥ D 1 → JFF						
IT8	Jumping. Identify which instruction of new word is to be performed next J → Mod 2 Transfer address of jump (AN2) → PA Transfer new word next 1 → PI → END																

Table 6.7.1-1 Organization of Jump Instructions

P21 is a signal active when the switch is OFF. If the current instruction is JBT, P21 active is required for the gating, PT4 $\longrightarrow$ IT1.

$$P21 \vee \overline{I81} = D122$$

$$I81 = \overline{P23} \cdot JBT$$

The P23 signal was provided for the eventuality of a three-position Breakpoint switch. At the present time it is constantly normal. Therefore I81 is active during the JBT instruction.

If the ADVANCE bar is depressed when the computer pauses, the computer will proceed to perform the unconditional jump. However while the computer is paused, an option exists of either skipping over or changing the instruction.

Jump Instructions in Repeat Mode

These instructions follow the rules of operation in the repeat mode that apply to all instructions. In addition, however, one circumstance will result in the contents of JA being something other than the return address.

When the jump instruction is I_0 :

$$IT2 \cdot D76 \cdot V43 \Longrightarrow [(MA) \longrightarrow JA]$$

to transfer the return address for I_1 to JA. The presumption is that the word containing the jump instruction was just transferred to PR and its address is still in MA. The action in the PTs preceeding these ITs was (V) $\longrightarrow$ PR.

However, when in the repeat mode, the second and all succeding times the jump instruction is performed, the action in the PTs preceeding the ITs was the counting of N. MA will contain the Repeat Counter information in IT2 as MA is the transfer route from AN2 to N. Therefore in IT2, effectively $(MA) \equiv (N) \longrightarrow JA$.

The programmer may use the conditional jump as the left hand instruction of a repeated pair if the desired procedure is to test (conditional jump) before changing (right hand instruction). In this situation, a return to this point of the program will not be possible following the jump as the return address has been lost. These situations will be used when a return is not necessary.

6.7.2 Comparison Logic for Conditional Jumps

The S-2000 can perform two types of comparison of quantities for conditional jump instructions.

1. Equality, for example (A) = (D)

2. Equal to or greater than, for example $(A) \geq (D)$

In the second type of comparison the quantities can be viewed in either of three senses.

1. Alphanumeric sense where the numbers or quantities representing characters are always considered as positive binary numbers, in the range:

   $$0.0 \leq N < 10.0$$

   No sign consideration is therefore necessary. A quantity greater than 1.0 is greater than any quantity smaller than 1.0.

2. Algebraic sense where the sign must be considered.

   The number range is:

   $$-1.0 \geq N < +1.0$$

3. Floating point sense where the represented number is algebraic, consisting of a mantissa and exponent $(N \times 2^n)$.

<u>Equality Comparison</u>

The nature of AN1 readily permits a determination whether the contents of A and D* are identical. One rule of the adder logic is that from each bit position a carry-out exists only under two conditions. Either the bits are equal, or if the bits are unequal then only when there is a carry-in to that bit. Another rule is that AN1CC exists only if there is a carry-out from each bit.

If the existence of the carry-in to each bit is prevented, by some means, then AN1CC can exist only if $a_n = d_n$ for every bit. If the carry-in is inhibited, the existence of AN1CC is proof that (A) = (D). This is the method of comparison for equality.

The carry-in signal to a bit must pass through an "and gate" where it is gated with the "Carry Inhibit" (AN1 C. Inh.) signal. If AN1 C. Inh. is active, no active gate output will exist and there will be no carry-in to the bit.

For equality comparisons, the AN1 C. Inh. signal is provided by a flip-flop called AN1I (AN1 Inhibit FF)

$$(AN1I = 1) \Rightarrow \text{AN1 C. Inh.}$$

Comparison of A Register with Zero

The JAZ instruction (jump if (A) = 0) is an equality comparison. The customary method of equality comparison is used. In this case, $0 \longrightarrow D^*$ and $1 \longrightarrow AN1I$. With the carry-in inhibited, an active AN1CC indicates equality with zero.

The JAP instruction may sometimes be referred to as "jump if (A) $\geqq$ 0". The more literal meaning of JAP is "jump if (A) are positive". This test is based upon the value of a_0 .

Comparison for "Equal to or Greater than"

"Greater than" is defined as being more positive. For two positive quantities, the larger is the greater. A positive quantity is greater than any negative quantity, with zero being considered as positive. When comparing negative numbers, the one closer to zero is the greater.

Alphanumeric Sense

The operation used to compare quantities in this sense is to add (A) and the two's complement of (D).

$A + (10.0 - |D|)$ where 10.0 represents binary two

Since $(10.0 - |D|) + D = 10.0$ by definition of a two's complement, then if $A \geqq D$, $A + (10.0 - |D|) \geqq 10.0$. A sum $\geqq 10.0$ will be evidenced by $c_0 = 1$, as against a sum < 10.0. The carry-out of AN1 C_0 id called "c_0".

Algebraic Sense

The same operation is used as above, adding (A) to the two's complement of (D). The number in Q is always transferred to D for the comparison. A review of some of the characteristics of the addition of binary numbers may aid in understanding the rules for deciding the results of the comparison. Based upon these characteristics, a determination can be made of whether $(A) \geq (D)$ by the examination of only S_o, C_o, and C_{-1}. Some of these characteristics are (although specific bits of AN1 are used in these illustrations, the cases are general):

$$\text{if } a_o = d^*_o \text{ , then } S_o = C_{-1}$$

$$\text{if } a_o \neq d^*_o \text{ , then } S_o \neq C_{-1}$$

$$\text{if } a_o = d^*_o = 1\text{, then } C_o = 1$$

In the S-2000, positive quantities are represented as a range of binary numbers: $1.0 > N \geq 0.0$. Negative numbers are represented as $(10.0 - |N|)$ and lie in the range of the binary numbers: $10.0 > -N \geq 1.0$. When quantities of oposite sign and equal magnitude are added, the sum is: $N + (10.0 - |N|) = 10.0$.

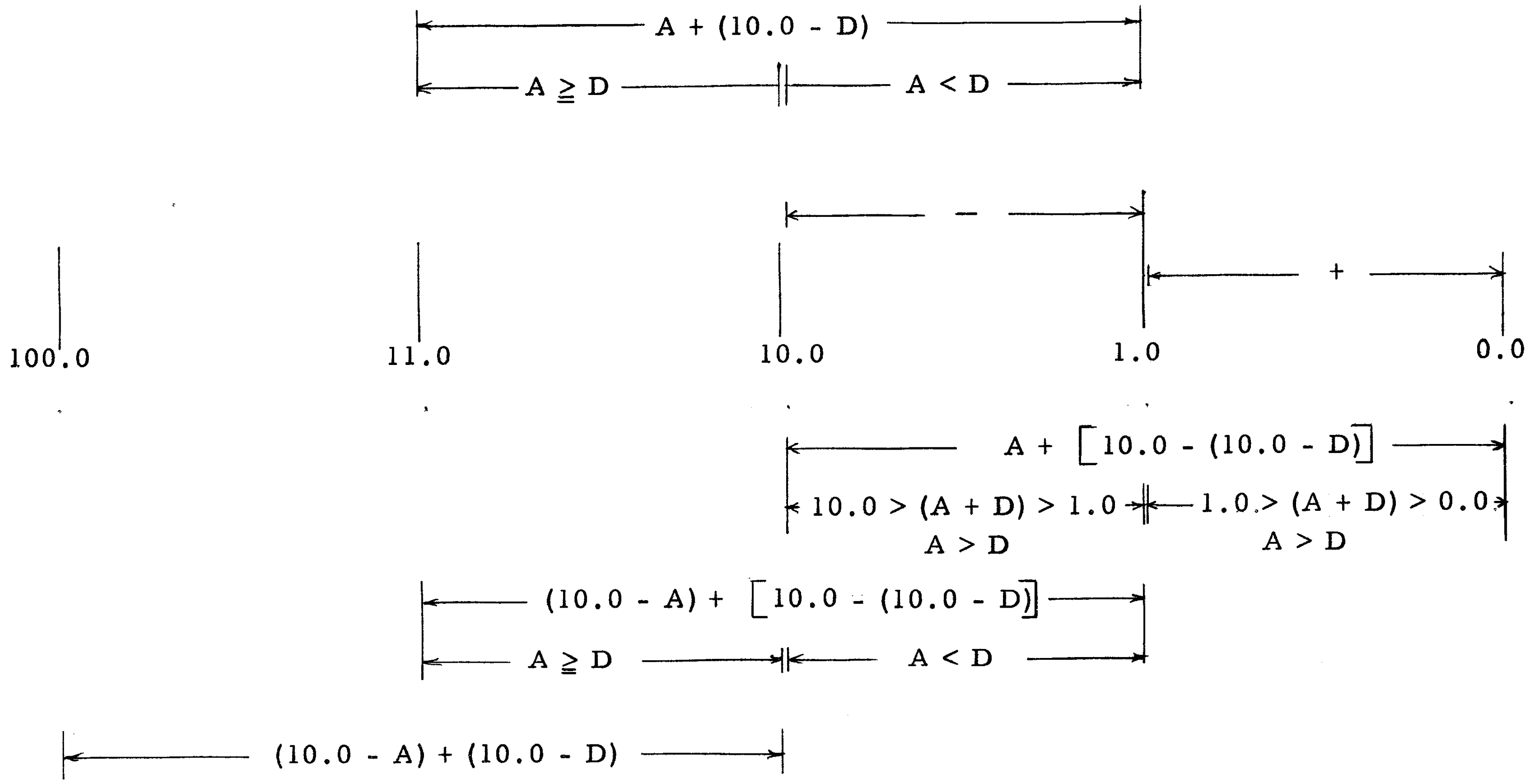
However the 2^1 bit is not stored and the result is the actual sum deducted by 10.0 or 0.0. This is a customary method of subtraction by mechanical or electronic devices.

When comparison is made, in the algebraic sense, the sums for the various possible combinations of numbers have the ranges shown in Figure 6.7.2-1. "A" and "D" are positive quantities of the range $1.0 > N \geq 0.0$; a negative quantity in the register is shown as $(10.0 - |A|)$ or $(10.0 - |D|)$.

<u>Contents of Both Registers Positive</u>

When the operation $A + (10.0 - |D|)$ is performed, the sum will be greater than 1.0 and be in the range:

$$\begin{array}{lcccc} & 1.0 & > & A & \geq 0.0 \\ + & 10.0 & \geq & (10.0 - |D|) & > 1.0 \\ \hline & 11.0 & > & \text{Sum} & > 1.0 \end{array}$$



Ranges of Sums Algebraic Comparison

Figure 6.7.2-1

If $A \geqq D$ the range of the sum is $11.0 > \text{Sum} \geqq 10.0$

If $A < D$ the range of the sum is $10.0 > \text{Sum} = 1.0$

Now, $a_o = 0$ and $d^*_o = 1$ as the two's complement of (D) is being sent to AN1. Since $a_o \neq d^*_o$, the $S_o \neq C_{-1}$.

If $A \geqq D$, to have a number in the range shown above, $S_o = 0$, which means $C_{-1} = 1$. This distinguishes it from $A < D$ where $S_o = 1$ and $C_{-1} = 0$. The logic used for this comparison by the computer is $(S_o = 0) \cdot (C_{-1} = 1)$.

The computer uses three logical conditions to decide that $(A) \geqq (D)$ in the algebraic sense. Table 6.7.2-1 shows the eight possible combinations of a_o d^*_o S_o C_o and C_{-1} and the logical condition satified when $(A) \geq (D)$. The conditions existing when $(A) < (D)$ are also listed to demonstrate the validity of the logical conditions used. Table 6.7.2-2 shows the values of the bits involved for the ranges of sums.

<u>A Register Positive, D Register Negative</u>

(A) is greater than (D) regardless of absolute magnitudes.

This operation is: $A + [10.0 - (10.0 - |D|)]$ or $A + D$

The range of the sum: $10.0 > \text{Sum} \geqq 0.0$

$$a_o = d^*_o = 0$$

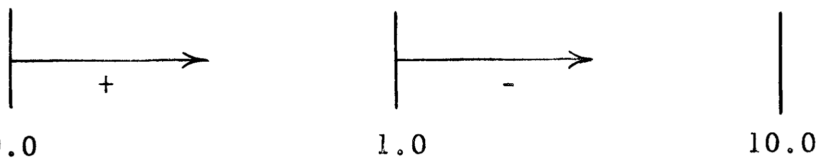
When the sum < 1.0, then $(S_o = 0) \cdot (C_o = 0)$.

When the sum $\geqq 1.0$ then $(C_o = 0) \cdot (C_{-1} = 1)$ is satisfied.

<u>Contents of Both Registers Negative</u>

The operation is: $(10.0 - |A|) + [10.0 - (10.0 - |D|)]$ or $10.0 + [(10.0 - |A|) - (10.0 - |D|)]$. The range of the sum is: $11.0 > \text{Sum} \geqq 1.0$.

The two's complement method of representing negative numbers adheres to the general rule whereby a more positive (a larger) number is represented by larger binary value. This is shown on the scale.



(A)	(D)	(A) : (D)	a_0	d^*_0	S_0	C_0	C_{-1}	Logical Condition Satisfied
+	+	$\geqq$	0	1	0	1	1	$(S_0 = 0) \cdot (C_{-1} = 1)$
+	+	$<$	0	1	1	0	0	
+	-		0	0	0	0	0	$(S_0 = 0) \cdot (C_0 = 0)$
+	-		0	0	1	0	1	$(C_0 = 0) \cdot (C_{-1} = 1)$
-	-	$\geqq$	1	0	0	1	1	$(S_0 = 0) \cdot (C_{-1} = 1)$
-	-	$<$	1	0	1	0	0	
-	+		1	1	1	1	1	
-	+		1	1	0	1	0	

Table 6.7.2-1

Ranges of Sums for A + D*

$1.0 > S_1 \geqq 0.0$ $a_0 = d^*_0 = S_0 = C_{-1} = C_0 = 0$ $A^+ > D^-$

$10.0 > S_2 \geqq 1.0$ $C_0 = 0$, $S_0 = 1$ and only one of the remaining three $(a_0\ d_0\ c_{-1})$ is equal to 1

C_0	S_0	a_0	d^*_0	C_{-1}	
0	1	0	1	0	$A^+ < D^+$
0	1	0	0	1	$A^+ > D^-$
0	1	1	0	0	$A^- < D^-$

$11.0 > S_3 \geqq 10.0$ $C_0 = 1$ $S_0 = 0$

C_0	S_0	a_0	d^*_0	C_{-1}	
1	0	1	1	0	$A^- < D^+$
1	0	0	1	1	$A^+ \geq D^+$
1	0	1	0	1	$A^- \geqq D^-$

$100.0 > S_4 \geqq 11.0$ $a_0 = d^*_0 = C_{-1} = S_0 = C_0 = 1$ $A^- < D^+$

Note: Subscripts "0" and "-1" Mean bit positions 0 and -1, or bit positions -36 and -37, respectively.

Table 6.7.2-2

where the arrows indicate the direction of representing larger numbers. Minus one is the smallest negative number.

When comparing two negative numbers, if $(A) \geqq (D)$, in the sense that (A) is more positive, the sum is in the range: $11.0 > \text{Sum} \geqq 10.0$ as indicated by the formula in the first paragraph. Otherwise the sum range will be $10.0 > \text{Sum} \geqq 1.0$.

When $(A) \geqq (D)$ $\qquad 11.0 > \text{Sum} \geqq 10.0$

$S_0 = 0$ and since $a_0 = 1$, $d^*_0 = 0$ $\qquad C_{-1} \neq S_0$ and $C_{-1} = 1$

If (A) is more negative than (D) the sum is in the range:

$$10.0 > \text{Sum} < 1.0$$

$S_0 = 1$, $C_{-1} = 0$ and $c_0 = 0$

(A) Negative and (D) Positive

For the logic to be valid it must decide that (A) is not equal to nor greater than (D) for this condition. The operation is $(10.0 - |A|) + (10.0 - |D|)$ or $100.0 - (|A| + |D|)$.

The range of the sum: $\qquad 100.0 \geqq \text{Sum} > 10.0$

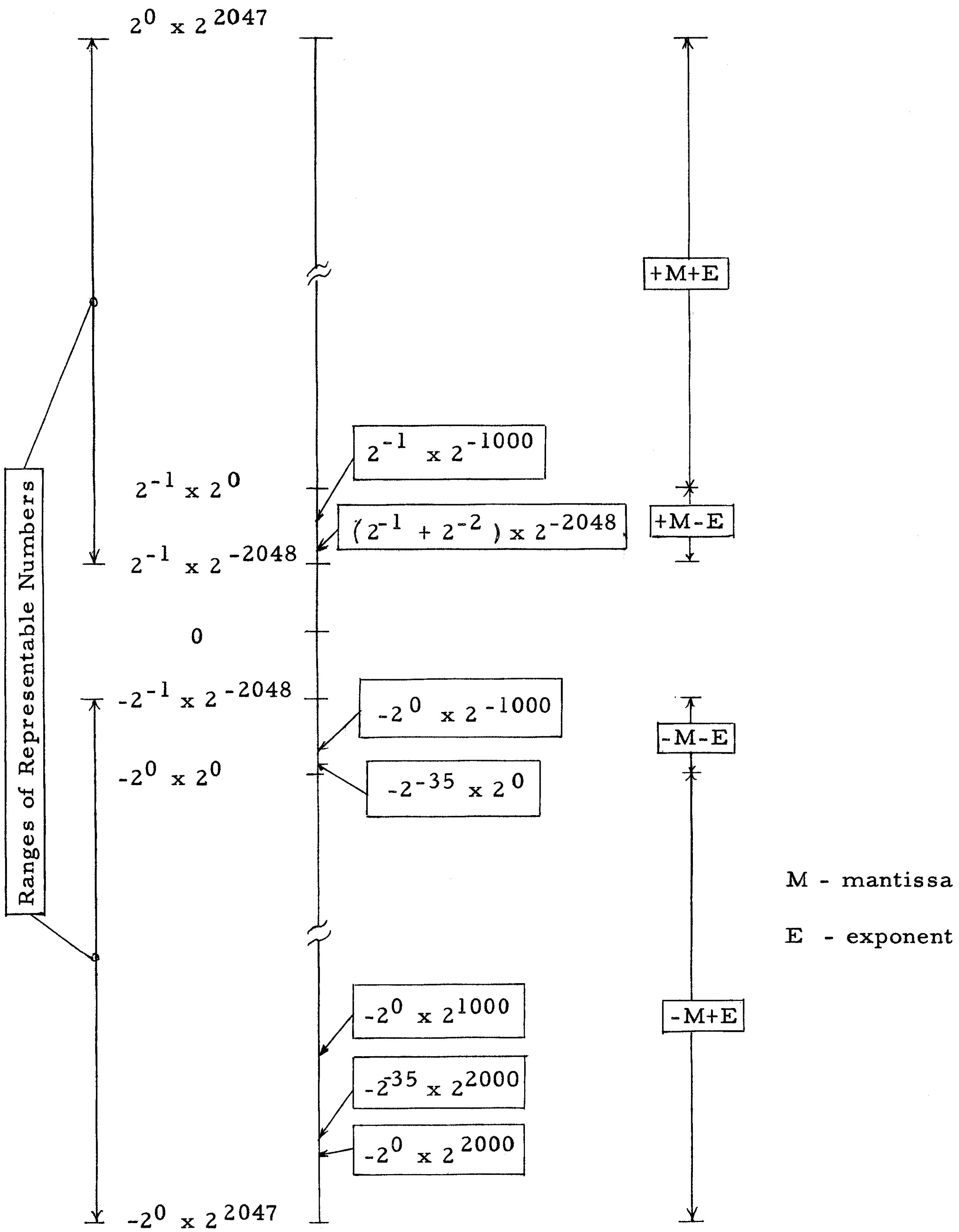
$a_0 = d^*_0 = 1$, therefore $C_0 = 1$ and $S_0 = C_{-1}$.

This will not satisfy any of the three logical conditions used for comparison.

Floating Point Numbers

The ranges of floating point numbers in S-2000 are shown in Figure 6.7.2-2. Floating point numbers have the following characteristics of change in mantissa (M) and exponent (E) as the numbers (N) grow larger (become more positive).

+M	+E	as N increases, + M or + E or both increase
+M	- E	as N increases, + M or - E or both increase
- M	- E	as N increases, - M increases or - E decreases or both
- M	+ E	as N increases, - M increases or + E decreases or both



S-2000 Floating Point Numbers

Figure 6.7.2-2

To summarize, as the mantissa becomes larger, the number becomes larger. The effort of the exponent, as it becomes larger, is to increase the distance of the number from zero. A larger exponent of a positive number makes the number larger. A larger exponent for a negative exponent makes the number smaller (less positive).

Table 6.7.2-2, summarizes the 16 possible cases of mantissas and exponents for the comparison of the two floating point numbers. In ten cases $A \geqq D$. The logic for the decision listed does not decide if $A_M \geqq D_M$. This is decided separately.

Table 6.7.2-3, shows that in the case where both floating point numbers are negative, the decision for the comparison of the exponents is: $A_E \leqq D_E$? This decision requires some adjustment of the exponent values before the comparison because the comparison logic cannot make such a decision directly. The comparison logic can make one of two decisions: $A_E \geqq D_E$? $A_E < D_E$? The adjustment is to increase D_E by one for comparison purposes. Then the decision for the exponents of two negative numbers becomes: $A_E < (D_E + 1)$? This is feasible for the comparison logic.

As the comparison is based upon the result of an arithmetic process, $A_E + (10.0 - |D_E|)$, where 10.0 is a binary two, it is possible to combine the addition of D_E and 1 with the subtraction of A_E and D_E. It will be noted, for example, that the subtraction of 10.0 and D_E was combined with the addition of A_E and $(10.0 - |D_E|)$. The combination is achieved by sending D_E' and AN1CE = 1 to the adder, AN1.

$D_E + 1$ can also be written as $D_E + 1.0$. The latter form indicates we are representing the value of one with the binary number of one, 1.0.

When comparing A_E and $D_E + 1.0$, the computer does so by the addition of $A_E + \left[10.0 - (|D_E| + 1.0)\right]^E$

With factoring this becomes: $A_E + \left[(10.0 - |D_E| + 1.0)\right]$

or: $A_E + (1.0 - |D_E|)$

$(1.0 - |D_E|)$ is the one's complement of D_E by definition. The effect of sending the one's complement of D_E alone to the adder is the same as sending the two's complement of $D_E + 1$: $D_E' \cdot (\text{AN1CE} = 0) \equiv (D_E + 1)' \cdot (\text{AN1CE} = 1)$. Therefore the usual case for comparison of negative numbers is to send $0 \longrightarrow$ AN1CE in order to have the one's complement of D alone connected to AN1.

	SIGN				RELATIVE MAGNITUDES FOR A > D	LOGIC FOR DECISION
	A_M	D_M	A_E	D_E		
J21	+	+	+	+	$A_E > D_E$ or $A_E = D_E \cdot A_M \geqq D_M$	$\overline{(C_{-36} = 1 \cdot C_{-37} = 0)} \cdot S_{-36} = 0 \cdot a_0 = 0 \cdot d_0 = 0$
J18	+	+	+	+	$A > D$	$C_{-36} = 0 \cdot C_{-37} = 1 \cdot a_0 = 0 \cdot d_0 = 0$
J21	+	+	-	-	$A_E > D_E$ or $A_E = D_E \cdot A_M \geqq D_M$	$\overline{(C_{-36} = 1 \cdot C_{-37} = 0)} \cdot S_{-36} = 0 \cdot a_0 = 0 \cdot d_0 = 0$
J20	+	-			$A > D$	$a_0 = 0 \cdot d_0 = 1$
J22	-	-	+	+	$A_E < D_E$ or $A_E = D_E \cdot A_M \geqq D_M$	$\overline{(C_{-36} = 0 \cdot C_{-37} = 1)} \cdot S_{-36} = 1 \cdot a_0 = 1 \cdot d_0 = 1$
J22 J19	-	-	-	+	$A > D$	Same as Line Above or $C_{-36} = 1 \cdot C_{-37} = 0 \cdot a_0 = 1 \cdot d_0 = 1$
J22	-	-	-	-	$A_E < D_E$ or $A_E = D_E \cdot A_M \geqq D_M$	$\overline{(C_{-36} = 0 \cdot C_{-37} = 1)} \cdot S_{-36} = 1 \cdot a_0 = 1 \cdot d_0 = 1$
	+	+	-	+	$A < D$	
	-	+			$A < D$	
	-	-	+	-	$A < D$	

JAQF Exponent Comparison

Table 6.7.2-3

Mantissas

When comparing two floating point numbers, the decision that $A \geqq D$ cannot be made solely by separate comparison of the respective parts of the compound numbers to determine the relations of $A_M : D_M$ and $A_E : D_E$ (where the subscripts indicate the mantissa and exponent). The difference between mantissas has to be considered when comparing exponents as that difference may outweigh the difference between the exponents. For example, if $A = .1 \times 2^4$ and $D = .0001 \times 2^6$ then A is greater than D.

Normalizing the floating point numbers enables the computer to avoid this situation. The mantissas of normalized numbers with like signs are in the range from .1000....0 to .111....1, inclusive. The difference between the two mantissas will therefore be less than a binary .1 or less than 2^{-1}.

With normalized numbers the comparison of magnitudes can be based, with one exception, solely upon the exponents. The minimum differnce between exponents is greater than the maximum difference between mantissas. With positive numbers, for example, $A_E > D_E$ alone will determine that $A > D$. The exception is when $A_E = D_E$. In this case the determination that $A \geqq D$ is also based upon a comparison of A_M and D_M. In this case A_M must be equal to or greater than D_M for $A \geqq D$.

The method used to handle the case of $A_E = D_E$ was chosen to enable a decision at the time the exponents are compared. The mantissas are compared first and one exponent adjusted, as a result if necessary. The reasoning is: if $A_M < D_M$ then $A \geqq D$ only if $A_E > D_E$. The comparison logic can only judge $[A_E \geq D_E?]$ So for the case of $A_M < D_M$ the exponent of D is adjusted (increased by one). In this case, when A_E and D_E are equal, the decision that $A \geqq D$ is prevented as the actual comparison is being made between A_E and $(D_E + 1.0)$. If $A_M < D_M$, then for $A \geqq D$, $A_E \geqq D_E + 1.0$

$D_E + 1.0$ is derived by sending the one's complement, instead of the two's complement, to AN1 for the arithmetic:

$(D)' \longrightarrow D^*$ and
only if $A_M \geqq D_M$, $1 \longrightarrow$ AN1CE

Table 6.7.2-4 summarizes the logic for the various cases.

If Result of Magnitude Comparison is	Exponent Comparison Should be	Thus	Logic for Mantissa Decision	V. Numbers
$^{+}A_M \geqq {}^{+}D_M$	$A_E \geqq D_E$	$1 \longrightarrow$ ANICE	$\overline{(C_0 = 1 \cdot C_{-1} = 0)} \cdot a_0 = 0 \cdot S_0 = 0$	$V47 \cdot V63 \cdot \overline{V61}$
$^{+}A_M < {}^{+}D_M$	$A_E \geqq D_E + 1.0$	$0 \longrightarrow$ ANICE	Note: ANICE was precleared to zero. No action necessary at this time.	
$^{-}A_M \geqq {}^{-}D_M$	$A_E < D_E + 1.0$	$0 \longrightarrow$ ANICE		
$^{-}A_M < {}^{-}D_M$	$A_E < D_E$	$1 \longrightarrow$ ANICE	$\overline{(C_0 = 0 \cdot C_{-1} = 1)} \cdot a_0 = 1 \cdot S_0 = 1$	$\overline{V60} \cdot V48 \cdot V62$
$^{+}A_M > {}^{-}D_M$	See Note 1	$1 \longrightarrow$ ANICE	$C_0 = 0 \cdot C_{-1} = 1 \cdot a_0 = 0$	$V60 \cdot V47$
$^{-}A_M < {}^{+}D_M$	See Note 1	$1 \longrightarrow$ ANICE	$C_0 = 1 \cdot C_{-1} = 0 \cdot a_0 = 1$	$V48 \cdot V61$

Note 1. Redundant activity as later comparison is based upon signs of numbers.

JAQF Mantissa Comparison

Table 6.7.2-4

6.8 Repeat

6.8.1 Repeat Instruction

The Repeat Instruction causes repetitive performance of the next instruction (if the Repeat Instruction is in PR_0) or the next pair of instructions (if the Repeat Instruction is in PR_1). The Repeat Instruction establishes the parameters of the repetitive performance with this format:

$\alpha\ \beta\ \gamma\ \delta$	I_V	C
4 bits	12 bits	8 bits
Address modification of repeated instructions	Number of times the following instruction/s is/are to be performed	Command coding of repeat

Address Modification

$\alpha\ \beta$	refer to I_0 being repeated.
$\gamma\ \delta$	refer to I_1 being repeated.
$\alpha\ \beta$ or $\gamma\ \delta$ = 00	Address modification (and any index register counting) are as if instruction were not being repeated.
$\alpha\ \beta$ or $\gamma\ \delta$ = 10	Effective address of repeated instruction is the contents of the selected index register, (selected by the S and I_A bits of the instruction being repeated). After each performance of the repeated instruction (X) is increased by (I_V) of the repeated instruction.
$\alpha\ \beta$ or $\gamma\ \delta$ = 11	Same as for 10 except $(X) - (I_V) \longrightarrow X$

In 10 and 11, the value of X_c does not affect (X).

$\alpha\ \beta\ \gamma\ \delta$ are stored in the Repeat Register (RR). SW2 selects the proper pair of bits for control during the performance of the instruction in the repeat mode. $(\alpha = 0 \cdot SW2 = 0) \vee (\gamma = 0 \cdot SW2 = 1)$ is referred to as $\alpha\gamma = 0$. $\alpha\gamma = 1$ is devised in similar fashion.

Repeat Register

RR

RPT	α	β	γ	δ	I

RR is cleared to zero at the end of the repeat mode and so remains until the next repeat mode. The command coding of the Repeat Instruction (000 1011) sets the RPT bit of RR to 1. If the Repeat instruction is I_0 then only I_1 of that instruction word is to be performed in the repeat mode. In this case, SW2 = 0 will set the I bit of RR to 1. The other four bits of RR store the information in the four most significant bits of the Repeat Instruction for address and index register modification. The output signals of RR are designated as "R" numbers.

Amount of Performance

The 12-bit field, I_V , indicates the number of times the following instruction (or pair) is to be performed. The range of I_V is from 0 to 4095, inclusive. If it is 0, the following instruction (or pair) is skipped, not performed at all.

The number in this I_V field is transferred, in two's complement form, to the Repeat Counter, N. Following each performance of the I_1 instruction in the repeat mode, one is added to N. N is connected to the 2_0 through 2_{11} bits of AN2 and AN2C is set to 1. The sum is returned to N through MA.

When N has been incremented after the desired number of instruction performances, it will be equal to zero. N = 0 is sensed as the first time a carry-out of one exists from bit 2_{11} ($AN2c_{11} = 1$). This ends the repeat mode by clearing RR to zero and calling for a new instruction word.

Repeat Mode Termination by Jump

A jump (usually a conditional jump instruction) provides another exit from the repeat mode. If JFF is set to 1 during the instruction, regardless of the Repeat Counter setting, the program jump is executed and the repeat mode is terminated.

Skip

To allow skipping the instruction performance when I_V of the Repeat instruction is zero, the quantity in I_V is sensed by sending it to the adder in complementary form. If $AN2c_{11} = 0$ the repeat mode operational controls are then set up and (I_V) is transferred to N. If $AN2c_{11} = 1$ arrangements to skip are made.

To skip, if the Repeat instruction were I_0, a new instruction word is called for by sending $1 \longrightarrow$ PI. If the repeat instruction were I_1, (PA) are incremented by one and then a new instruction word is called for by sending $1 \longrightarrow$ PI.

Repeat Instruction Performance

The sequence of tasks for this instruction follows. The instruction either sets up RR to control the repeat mode of operation or adjusts the program control to effect a skip.

Repeat Instruction

	Purpose	Activity	Logic
IT1			
	Clear RR	$0 \longrightarrow$ PR	I12
	$(I_V)'' + 0 \longrightarrow$ AN2	PR $\longrightarrow$ SW6	$\overline{I20}$
	to transfer $I_V \longrightarrow N$	$- \longrightarrow$ PM	D59
	and sense if $I_V = 0$.	$1 \longrightarrow$ AN2C	D59
		$0 \longrightarrow$ SW4	D55
		Inh. AN2	timing alone

	Purpose	Activity	Logic
	Prepare transfer route $I_v \longrightarrow$ N.	$0 \longrightarrow$ MA	D65 · $\overline{\text{I41}}$ · $\overline{\text{I81}}$ · $\overline{\text{D52}}$
		IT1 $\longrightarrow$ IT2	$\overline{\text{I84}}$
IT2			trigger D67
	Prepare to locate next instruction. If RPT instruction is I_1, next instruction will be I_0 (regardless of skip).	(SW2)' $\longrightarrow$ Mod 2	I12 · V42
	If RPT is I_0 next instruction will be I_1 if $I_v \neq 0$ (carry $AN2c_{11} = 0$)	(SW2)' $\longrightarrow$ Mod 2	I12 · V41
	If skip ($AN2c_{11} = 1$) the next instruction will be I_0 and Mod 2 already is 0.		
	Transfer $1 \longrightarrow$ RR if $AN2c_{11} = 0$. The $1 \longrightarrow$ RR signal will set RPT bit to 1, and transfer $\alpha\beta\gamma\delta$ from PR to RR.	$1 \longrightarrow$ RR	I12 · V41
	Start Transfer (AN) $\longrightarrow$ N	AN2 $\longrightarrow$ MA	D65 · $\overline{\text{I41}}$ · $\overline{\text{I81}}$ · $\overline{\text{D52}}$
		IT2 $\longrightarrow$ IT3	D 73 · $\overline{\text{I44}}$ · $\overline{\text{I128}}$ · $\overline{\text{I17}}$
IT3			
	Clear N for new number	$0 \longrightarrow$ N	I12
	Establish whether instruction to be repeated is already in PR or has to be transferred from memory	$0 \longrightarrow$ PI, PI*	I12 · V 43 · V41
		$1 \longrightarrow$ PI	D 79 · V42

	Purpose	Activity	Logic
	If skip ($AN2c_{11}$ = 1)	1 $\longrightarrow$ PI	I12 • V56
		IT3 $\longrightarrow$ IT4	
IT4			trigger AN2CC
	Complete transfer (AN2) $\longrightarrow$ N if not skipping ($AN2c_{11}$ = 0).	MA $\longrightarrow$ N	I12 • V41
	If no skip end instruction	IT4 $\longrightarrow$ END	I12 • V41
	If skip and RPT was I_0 end instruction as PI and Mod 2 have been previously set to the proper states.	IT4 $\longrightarrow$ END	I12 • V43 • V56
	If skip and RPT was I_1 continue IT to increment PA. PI and Mod 2 have been previously set to the proper states.	IT4 $\longrightarrow$ IT5	I12 • V42 • V56
IT5			trigger D103
	Connect AN2 inputs for adding 1 to PA.	MA $\longrightarrow$ SW4	I12
	(MA = 0 due to logic) (AN2c was set to 1 in IT1)	0 $\longrightarrow$ PM	I12
		IT5 $\longrightarrow$ IT6	
IT6			
	Transfer (PA) $\longrightarrow$ AN2.	PA $\longrightarrow$ MA	I12
		Inh. AN2	I12
		IT6 $\longrightarrow$ IT7	

	Purpose	Activity	Logic
IT7			
	Clear PA for new address.	$0 \longrightarrow$ PA	D113
		IT7 $\longrightarrow$ IT8	
IT8			trigger V66 (AN2CC)
	Transfer new address.	AN2 $\longrightarrow$ PA	D113
	End instruction.	IT8 $\longrightarrow$ END	

6.8.2 Repeat Mode of Operation

As indicated previously, the performance of an instruction in the repeat mode differs in only two ways from performance in the non-repeat mode. The instruction may be either skipped or repeated, and the effective address may be different from that in the non-repeat mode of performance.

The controls for repeat mode operation are described in the following section to show the relations between counting the index register and repeat modification of the index register.

6.8.3 Index Register Modification

The controls for the counting of an index register and operating in the repeat mode are organized in similar fashion. For this reason they are described in one section.

Non-routine Program Control Activity

During index register modification and repeat mode operations a departure must be made from the routine sequence of program control activity. The routine activity is the case when instructions follow each other in consecutive memory address sequence and the action of SW2, Mod 2 and PI is shown in Figure 6.8.3-1.

These activities are inserted between the end of an instruction performance and the routine program control activity. PI = 10 is used for the modification of X and PI = 11 is used to count N and test N = 0? If a new instruction word is not required for the next instruction, the activities usually performed during the PTs with PI = 00 are incorporated with these end-of-instruction tasks. However, if a new word is required, a separate set of PTs with PI = 01 is necessary. This will follow the timings under PI = 10 and/or PI = 11.

For example, if the counter bit of the selected index register were 1, at the end of the instruction, the contents of X are to be incremented by 1. The counting is done during a set of PTs with PI = 10. The sequence of controls are shown in Figure 6.8.3-2

During the timings under PI = 01, 10 or 11 the program control register must be capable of two different settings simultaneously to indicate the current and succeeding program control states. For this reason two program control registers exist, PI and PI*. The setting of PI is transferred to PI* in PT1 by timing alone. PI controls activity during PT1 and PT2. PI* controls activity during PT3 and PT4. During PT3, PI is set to its new state in preparation for the next set of PTs. With PI equal to 01 or 11 the PTs are reinitiated:

$$PT4 \cdot PI = 10 \Longrightarrow PT1$$

During the last set of PTs, PI is set to 00 in PT3. This state of PI will provide the condition for exiting from PT4 to IT1.

Program Control for Repeat Mode

Following each performance of the I_1 instruction in the repeat mode of operation, (N) are incremented and the N = 0? test is made. In this case during IT3 of the repeated instruction,

SW2	I_0 or I_1 is currently being performed or about to start.
PI = 0	Next instruction is in PR.
PI = 1	Get new instruction word for next instruction.
Mod 2	Next instruction will be I_0 or I_1.

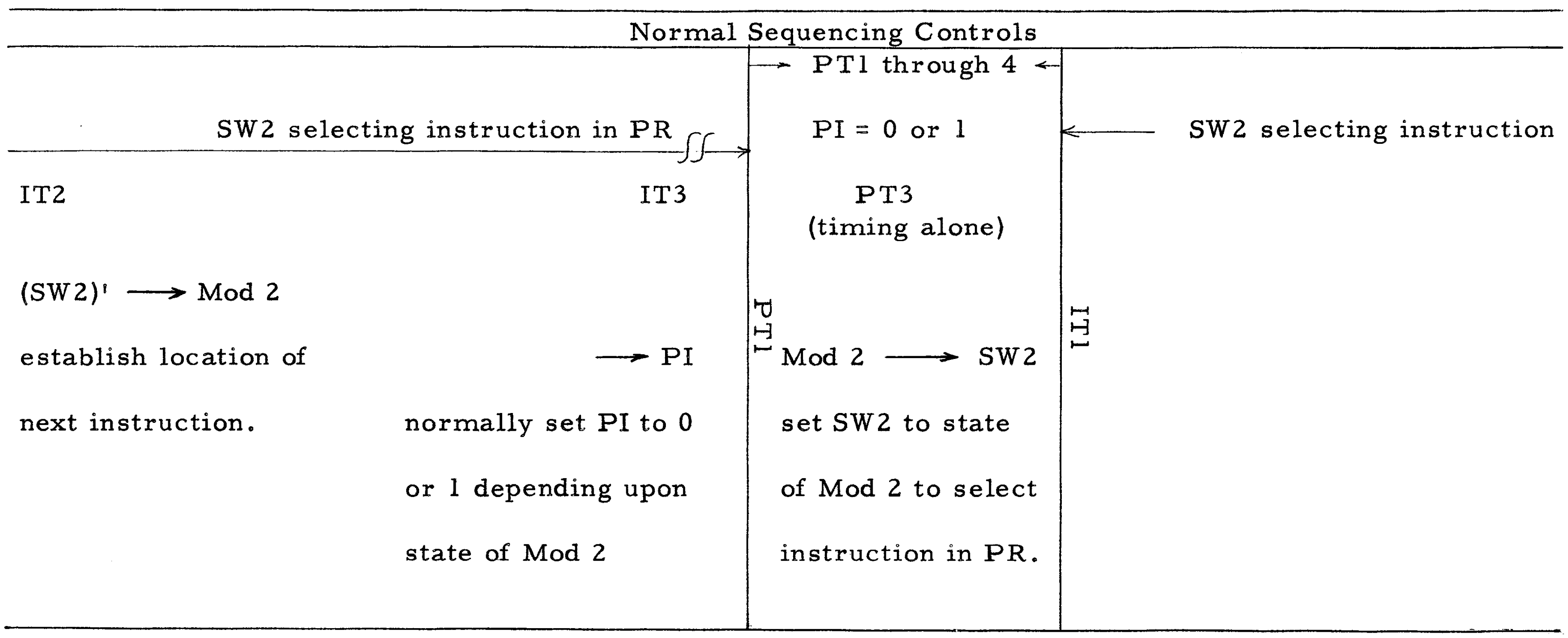

Normal Sequencing Controls
PT1 through 4
SW2 selecting instruction in PR
PI = 0 or 1
SW2 selecting instruction
IT2
IT3
PT3 (timing alone)
(SW2)' → Mod 2
establish location of next instruction.
→ PI
normally set PI to 0 or 1 depending upon state of Mod 2
PT1
Mod 2 → SW2
set SW2 to state of Mod 2 to select instruction in PR.
IT1

Figure 6.8.3-1 Normal Program Control Sequencing

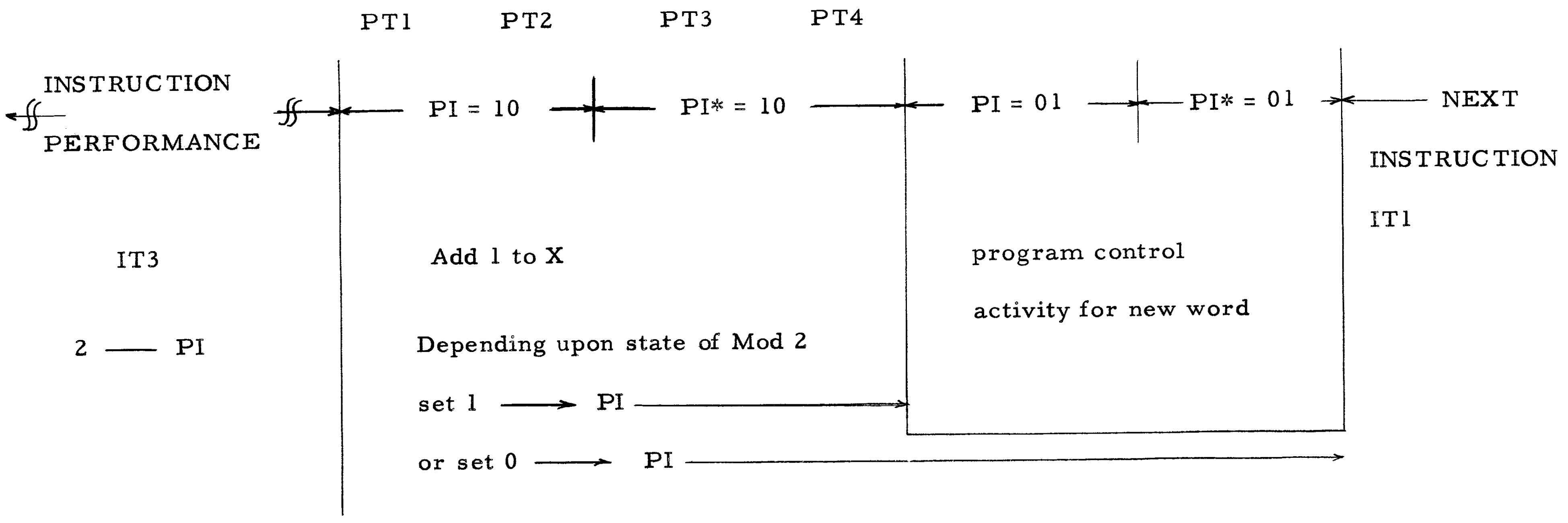


Figure 6.8.3-2 Index/Repeat Program

$3 \longrightarrow$ PI. During this set of PTs the above-mentioned tasks are performed and PI is set to 00 or 01 dependent upon the N = 0? test. If $N \neq 0$, then PI = 00 will cause control to pass to II after the one set of PTs.

If a repeat modification of (X) is also involved, it is done prior to the incrementing of (N). Two and possibly three sets of PTs occur:

	during IT3	$2 \longrightarrow$ PI	prepare for modifying X
1.	during PT3	$3 \longrightarrow$ PI	prepare for incrementing N
2.	during PT3	0 or $1 \longrightarrow$ PI	depending upon N = 0? test
3.	if ending repeat mode, last of PTs with PI = 01	$0 \longrightarrow$ PI	program control action for new word

When a repeated jump instruction causes a jump there is no need to increment N. The PTs under PI = 11 are bypassed. Following the instruction, if no X modification is involved, control passes directly to PI = 01 for the new word. If X is to be modified, following the instruction control will pass to PI = 10 and then to PI = 01. N and RR are cleared to zero in PT1 of PI = 01.

Repeat mode operation requires one or two departures from the usual sequencing of instruction -- no new instruction word and possibly perform I_1 only. The first is achieved by avoiding $1 \longrightarrow$ PI until the exit from repeat is reached. If, in addition, only I_1 is to be performed it is necessary to prevent the alternation of SW2 and Mod 2. This chain is broken by inhibiting the routine action during IT2;

$$IT2 \cdot R6 \cdot \overline{D52} \cdot \overline{I122} \cdot \overline{I12} \Longrightarrow (SW2)' \longrightarrow \text{Mod } 2$$

where R6 means I = 0. The I bit of the Repeat Register (RR) will be I = 1 only if I_1 alone is to be repeated. Note that I = 0 when not in the repeat mode as RR is cleared to zero at the exit from the repeat mode.

<u>Logic of Routine Program Control Activity</u>

Mod 2

$$IT2 \cdot R6 \cdot \overline{D52} \cdot \overline{I122} \cdot \overline{I12} \Longrightarrow (SW2)' \longrightarrow \text{Mod } 2$$

$$R6 = I = 0$$

$\overline{D52}$ = not a CF (Command Fault)

$\overline{D122}$ = not M x Q ±A $\Longrightarrow$ A rounded and II = 0

$\overline{I12}$ = not the RPT instruction

PI

$IT3 \cdot R6 \cdot R2 \cdot \overline{D79} \cdot P6 \cdot V55 \Longrightarrow 0 \longrightarrow PI, PI^*$

$IT3 \cdot R0 \cdot V54 \cdot P6 \cdot \overline{D79} \cdot \overline{D52} \Longrightarrow 1 \longrightarrow PI$

R6 = I = 0 performing both instructions in word.

R2 = $\alpha\gamma$ = 0 not in repeat mode or no repeat modification of X.

$\overline{D79} = \overline{I12 \vee I17}$ neither Repeat nor IOC instructions.

P6 = (X_c = 0) no counting modification of X.

V55 = (Mod 2 = 1) next instruction is I_1.

R0 = (RPT = 0) not in repeat mode

V54 = (Mod 2 = 0)

$\overline{D52}$ = not CF

SW2

PT3 $\Longrightarrow$ (Mod 2) $\longrightarrow$ SW2 (timing alone)

<u>Logic for Counting Modification of X</u>

During the instruction using an index register with counter bit of 1, $IT3 \cdot D81 \cdot \overline{I123} \cdot \overline{D52} \cdot \overline{D79} \Longrightarrow (2 \longrightarrow PI)$

D81 = $\overline{R2}$ v P7 X modification involved

$\overline{R2}$ = ($\alpha\ \gamma$ = 1)

P7 = (X_c = 1)

$\overline{I123}$ = not an index register instruction

$\overline{D52}$ = not CF

$\overline{D79}$ = neither RPT nor IOC instruction

At the end of the instruction control is returned to the program control register with PI = 10.

<u>Count Modification of Index Register</u>

PT1 PI = 10

connect operands to AN2	Inh. AN2
for counting of X	X $\longrightarrow$ SW4
	1 $\longrightarrow$ AN2C
	0 $\longrightarrow$ PM
Prepare transfer route back to X.	0 $\longrightarrow$ MA
	PR $\longrightarrow$ SW6
	PI $\longrightarrow$ PI*

PT2

begin transfer of sum to X	AN2 $\longrightarrow$ MA

PT3

complete transfer AN2 $\longrightarrow$ X	MA $\longrightarrow$ X
arrange control of selection of next instruction	Mod 2 $\longrightarrow$ SW2
Select next program control operation	
next instruction is to be I_1 (Mod 2 = 1)	0 $\longrightarrow$ PI
new instruction word required as:	
next instruction is to be I_0 in normal sequence (Mod 2 = 0) or a jump is to be executed.)	1 $\longrightarrow$ PI

PT4

If new word required	$\longrightarrow$ PT1
If last instruction was I_0 and no new word required	$\longrightarrow$ IT1

<u>Repeat Modification of Index Register</u>

In IT3 of the instruction being repeated, 2 $\longrightarrow$ PI, with the identical logic used if the index register were to be counted. Then in the first set of PT timings that follow the instruction.

PT1 PI = 10

Connect inputs to AN2	X $\longrightarrow$ SW4
	PR $\longrightarrow$ SW6
	Inh. AN2
if (X) + (I_V)	+ $\longrightarrow$ PM
	0 $\longrightarrow$ AN2C
if (X) - (I_V)	- $\longrightarrow$ PM
	1 $\longrightarrow$ AN2C
Prepare (AN2) $\longrightarrow$ X	0 $\longrightarrow$ MA
Program Control	PI $\longrightarrow$ PI*

PT2

Start (AN2) $\longrightarrow$ X	(AN2) $\longrightarrow$ MA
Prepare to select next instruction	(Mod 2) $\longrightarrow$ SW2

PT3

Complete (AN2) $\longrightarrow$ X	(MA) $\longrightarrow$ X

Select next control activity

Do I_1 next	$0 \longrightarrow PI$
No jump, last instruction was I_1, count X	$3 \longrightarrow PI$
Jump	$1 \longrightarrow PI$
	$0 \longrightarrow RR$

PT4

Reinitiate PTs for new word program control activity or for counting N	$\longrightarrow PT1$
Do next instruction, last instruction was I_0	$\longrightarrow IT_1$

Counting N

The method of incrementing N is similar to counting X. PI = 11 is the control of counting N. AN2 is used for the addition. N=0 is sensed by $AN2C_{11} = 1$. The details are listed in the "PI States Table".

An interesting development exists as a by-product of the incrementing of N. This may be used by programmers, see Programming R & D Note No. 11. The incremented number is returned to the Repeat Counter via MA. If the I_0 instruction of a repeated pair is a conditional jump, the computer will return to the I_0 instruction each time with the new N number in MA. The logic of conditional jump does not recognize the existence of the repeat mode. If the jump instruction is I_0, it operates on the assumption the instruction word was just transferred to PR from memory and its address is still in MA. This is transferred to JA as the return address.

In the repeat mode the Repeat Counter setting is placed in JA by the (MA) $\longrightarrow$ JA transfer. If repeat is terminated by the jump (in I_0) the contents of JA can be used to determine the number of times the instruction pair was performed.

The jump as the I_1 instruction will not produce this action since (PA) $\longrightarrow$ JA for the return address. If repeat is terminated the first time the jump in I_0 is performed, (JA) will not refer to (N), as the counter is incremented after the performance of the I_1 instruction.

PI States Table

Table 6.8.3-1 is a compilation of all the logic of PI activity. It contains listing of purpose, action and logic for each of the four states of the PI registers. The table indicates how the logic is minimized for the various functions.

	PI = 00
	do next instruction
		PI = 01
		get next instruction word
			PI = 10
			modify index register
				PI = 11
				increment Repeat Counter

PT 1		PI → PI* timing Incrementing PA 0 → PM P8 Prepare AN2 inputs 1 → AN2C P8 MA → SW4 P1 Prepare MA for new address 0 → MA timing Prepare PR for new word 0 → PR P1 Ending repeat mode by a jump 0 → RR P1 . V67 0 → N P1 . V67	PI → PI* Counting X, prepare AN2 inputs 0 → PM P2 . R2 1 → AN2C P2 . R9 X → SW4 P2 Inh. AN2 P2 Repeat modify X PR → SW6 P2 Prepare AN2 inputs Inh. AN2 P2 if adding + → PM P2.R3.R4 0 → AN2C P2.R3.R4 if subtracting - → PM P2.R3.R5 1 → AN2C P2.R9 Prepare AN2 → X 0 → MA timing	PI → PI* Incrementing N 0 → PM P8 Prepare AN2 inputs 1 → AN2C P8 N → SW4 P3 Inh. AN2 P3 Prepare AN2 → N 0 → MA timing
PT 2	Prepare to select next instruction (Mod 2) → SW2 timing	Address of next word for decoding (PA) → MA P25 and incrementing Inh. AN2 P1 (V) → PR 1 → MI P25 Prepare to select next instruction (Mod 2) → SW2 timing	Start AN2 → X AN2 → MA P9 Prepare to select next instruction (Mod 2) → SW2	Start AN2 → N AN2 → MA P9 Prepare to select next instruction (Mod 2) → SW2 timing
PT 3	PI* = 00	PI* = 01 Prepare AN2 → PA 0 → PA P11 Do instruction next 0 → PI D120	PI* = 10 Complete AN2 → X (MA) → X P12 After counting X get new instruction word for normal sequence 1 → PI D121 After counting X do I_1 which is already in PR 0 → PI D120 Jump after modifying X 1 → PI D121 0 → RR P12.V67 No jump; after modifying X count N 3 → PI P12. V59 . [R7v(R1.V54)]	PI* = 11 Clear N for new count 0 → N P13 Continue repeat mode N ≠ 0 0 → PI D120 End repeat mode 1 → PI D121 N = 0, next 0 → Mod 2 P13.V56 instruction is I_0 0 → RR P13.V56
PT 4	(SW2) → SW2* 0 → II Proceed to next instruction as memory not in use, no pause for BLPT or OVF → IT1 D122.D123 .PO.MO.P20	Transfer next address of sequence AN2 → PA P11 If new word was read due to jump clear JFF 0 → JFF P11 Do next instruction → IT1 D122.D123 .PO.MO.P20	Reinitiate PT for new word program control activity or for counting N → PT1 D124 Do next instruction if just counted X or if modified X and last instruction was I_0 → IT1 D122.D123 .PO.MO.P20	Complete transfer of new count to N (MA) → N P13 Reinitiate PT for next instruction word → PT1 D124 Do next instruction continuing repeat → IT1 D122.D123 .PO.MO.P20

PI States Table

Table 6.8.3-1

6.8.4 No Op Instruction NOP 000 0011

It is not always possible, when coding for repeated performance of a single instruction, to arrange the Repeat instruction and the one to be repeated in the same word. The NOP instruction is available, as a filler, for these cases. The RPT instruction is the right hand instruction of the preceding word. The instruction to be repeated and the NOP instruction form the next word.

The logic of NOP is devised to enable it to be used as the left or right hand instruction of the repeated pair. This requires the ability to increment and test the Repeat Counter.

NOP proceeds through the first four timings of IT and goes to END from IT4. The only activities performed are the standard control clearings in IT1, transfer (SW2)' in IT2 and set PI in IT3.

6.9 Index Register Instructions

Scope of Instructions

The index register instructions are those involving a transfer from an index register or some change in the contents of an index register other than counting. These instructions can be classified in three groups:

1. Transfers to or from index register.
2. Add to or subtract from index register.
3. Add to or subtract from index register, then perform some additional operation conditional upon the comparison of the new quantity in the index register.

The following subscripts will be used to describe the index register instructions.

0	left half of an instruction word
1	right half of an instruction word
C	counter bit of the index register
J	J bit of an instruction
R	R bit of an instruction
I_V	memory address field of an instruction
- 1	bit of JA corresponding to X_C and I_R and D_R bits

Transfers

This group of instructions concerns transfers between a selected index register and either the D register or PR.

In transfers between X and D, the contents of the latter is usually an instruction word. The transfer involves the memory address, and possibly the J bit, of one of the two instructions of the word in D. When transferred, the J bit of the instruction in D goes to the counter bit of the index register, or vice versa.

The memory address size is variable and dependent upon whether or not the instruction in D contains an index register address. The R bit of the instruction in D can denote the number of bits to be transferred between X and D.

The transfer instruction itself will specify which of the two halves in D is to be referenced. If the J bit of the transfer command is 0, D_0 is involved. If $I_J = 1$, D_1 is the reference.

011	0000	TDX	$(D_V) \longrightarrow X$ I_J selects either D_0 or D_1 D_{0R} or D_{1R} determines number of bits transferred.
011	0001	TDXC	$(D_V) \longrightarrow X$, $(D_J) \longrightarrow X_C$
011	0010	TXD	$(X) \longrightarrow D_V$
011	0011	TXDC	$(X) \longrightarrow D_V$, $(X_C) \longrightarrow D_J$
011	1001	TIX	$(I_V) \longrightarrow X$, $(I_J) \longrightarrow X_C$
000	0111	TCX	$(I_J) \longrightarrow X_C$

The transfer of $(D_V) \longrightarrow X$ is via SW1, SW6, PM, AN2 and MA to the index register. (See figure 1.1-1.) MA is cleared to zero for the transfer. When $D_R = 1$, zeros will be transferred to the higher order bit positions of the index register. For example, with an eight index register computer, when $D_R = 1$ the twelve least significant bits of D_V will be transferred (d_{-4} to d_{-15} or d_{-28} to d_{-39}). These would be transferred to bit positions 2^{11} to 2^{0} of the index register. If the memory size were 8096 words, bit position 2^{12} of X would be made zero.

The TDXC instructions transfer (D_J) to X_C as the new counter bit, as well as transferring (D_V). With the one exception to be described later, none of the index register instructions cause counting action of the index register when they are performed. The only logic that sets 2 to PI is:

$$D81 \cdot \overline{D79} \cdot \overline{D52} \cdot \overline{I23} \implies 2 \longrightarrow PI$$

and $I23 \equiv 011$, of the 3-bit command coding, which identifies all but one type of the index register instruction.

The transfer logic is straightforward. In IT1 the controls are cleared and the proper inputs corrected to AN2.

$0 \longrightarrow$ AN1I, AN2I, EO, UF, SC, FI, AN1CE
Inhibit AN2
$0 \longrightarrow$ SW4, $+ \longrightarrow$ PM, $0 \longrightarrow$ AN2C, D $\longrightarrow$ SW6, $0 \longrightarrow$ MA
Either 0 or 1 $\longrightarrow$ SW1, dependent upon the J bit of transfer instruction.

In IT2 the number is transferred to MA. In IT5, it is transferred from MA to X. If the counter bit is to be transferred, it is done in this timing. This completes the significant part of the instruction which will go to END through IT8.

The transfers into the index registers are "jam transfers", the registers do not require a prior clearing.

TXD, TXDC

The transfer from an index register to D is via the JA register. The route is X, AN2, MA, JA to D. The counter bit, if transferred, goes $(X_C) \longrightarrow ja_{-1}$ and then to d_{-16} or d_{-40} .

The route utilizes the existing transfer paths. The transfer from JA to D was required to enable storing the return address in memory after a jump. (JA) $\longrightarrow$ D is a jam transfer as only part of D is to be changed.

The setting of SW1 is used to control into which half of D the transfer is to be made. The value of the J bit of the TXD or TXDC instruction is transferred to SW1.

The outline of the logic of these transfers is as follows. In IT1 the controls are cleared, the MA and JA registers are cleared and AN2 inputs connected (X $\longrightarrow$ SW4, $0 \longrightarrow$ PM, $0 \longrightarrow$ AN2C). PR is connected to SW6 but is of no significance. SW1 is set by the I_J bit.

In IT2, AN2 is connected to MA. In IT7, (MA) are transferred to JA and in IT8, (JA) are transferred to D. For the TXDC instruction, $(X_C) \longrightarrow ja_{-1}$ in IT5 and ja_{-1} is transferred with the rest of JA in IT8.

The number of bits transferred from JA to D is determined by this logic (for an eight index register system):

$011 \cdot 001X \cdot SW1 = 0 \Longrightarrow (ja_{11} \ldots ja_0) \longrightarrow d_{-4} \ldots d_{-15}$
$011 \cdot 001X \cdot SW1 = 0 \cdot d_0 = 0 \Longrightarrow (ja_{12}) \longrightarrow d_{-3}$
$011 \cdot 001X \cdot SW1 = 1 \Longrightarrow (ja_{11}) \ldots ja_0) \longrightarrow d_{-28} \ldots d_{-39}$
$011 \cdot 001X \cdot SW1 = 1 \cdot d_{-24} = 0 \Longrightarrow (ja_{12}) \longrightarrow d_{-27}$

The instructions involving D and X can be used to handle information other than the address field of an instruction. The information, such as a test quantity or a counter number, is kept in the suitable part of the word for such transfers. This is, of course, the bit positions equivalent to those comprising the address field of an instruction.

Transfer Counter Bit - TCX

These two instructions (TCXS, TCXZ) will transfer the value of the J bit in their command coding to the specified index register. TCXS means "Set counter to one"; TCXZ means "Make counter bit Zero". The D register is not involved. The 3-bit command coding, 000, is different from the other index register instructions.

The instruction is performed in four ITs (IT1 - IT4). In IT2, $(I_J) \longrightarrow X_C$ for the transfer of the new counter bit.

If TCXS is performed, and the R bit of the TCXS instruction is equal to 1, the index register will count the performance of the TCXS instruction as one of the times the specified index register was referenced.

$$IT3 \cdot X_C = 1 \cdot R \text{ bit} = 1 \cdot \overline{SKIP} \cdot \overline{REPEAT} \cdot \overline{CF} \Longrightarrow 2 \longrightarrow PI$$

This occurs after IT2, when X_C was set to 1.

Add, Subtract Index

011	0100	ADX	$(X) + (D_V) \longrightarrow X$
011	0101	SDX	$(X) - (D_V) \longrightarrow X$

The outline of these four instructions should be apparent. It is similar to TDX, but for the addition or subtraction the index register is connected to AN2. The I_J bit determines which half of D is used as an operand. The D_R bit of that determines the number of bits used for D_V. AN2 is used for the addition. The sum is returned to X via MA. If the counter bit were one, the register would not be counted (after the add or subtract) as the 3-bit command coding, 011, prevents $2 \longrightarrow PI$ action.

The instruction is performed during IT1 to IT8.

Index Arithmetic - Conditional Jump (AIXJ, SIXJ)

These two instructions are used in a program to change the contents of an index register, by a fixed amount each time, until a specified limit is reached.

011	1100	AIXJ	$(X) + (I_V) \longrightarrow X$; then jump if $X \neq D_{OV}$ to the address given by D_{1V} and D_{1J}
011	1101	SIXJ	Same except $(X) - (I_V) \longrightarrow X$.

The increment, or decrement, is I_V. The instruction has to specify the index register and I_V will be less than 15 bits. The specified limit is the address of the left half of D, and d_0 will determine whether 15 bits of D_0, or less, are compared with the contents of the index register. The jump address is given by the right half of the D register.

AN2 is used for two purposes. First to add or subtract, then for the equality comparison. The return address is transferred to JA regardless of whether the jump occurs.

Logic of AIXJ

IT1

Clear controls	timing alone	$0 \longrightarrow$ EF, UF, SC, AN2I, FI, ANICE
" "	$\overline{D51}$	$0 \longrightarrow$ AN1I
Set AN2 inputs	$\overline{I18}$	$X \longrightarrow$ SW4
	$\overline{I20}$	PR $\longrightarrow$ SW6
	$\overline{I26} \cdot \overline{I29}$	$+ \longrightarrow$ PM
	$\overline{(I12 \vee I17 \vee I27 \vee I28)}$	$0 \longrightarrow$ AN2C
	timing alone	Inh. AN2
Clear register		$0 \longrightarrow$ JA

IT2

Program control	$R6 \cdot \overline{D52} \cdot \overline{I12}$	$(SW2)' \longrightarrow$ Mod 2
Transfer return address	$I53 \cdot SW2 = 1$	$(PA) \longrightarrow$ JA
Transfer return address	$I53 \cdot SW2 = 0$	$(MA) \longrightarrow$ JA, $1 \longrightarrow ja_{-1}$

IT3

Clear MA for sum	I53	$0 \longrightarrow$ MA
Program control	$R2 \cdot R6 \cdot (P6 \vee I123) \cdot \overline{V55} \cdot \overline{D79}$	$0 \longrightarrow$ PI
" "	$R0 \cdot (P6 \vee I123) \cdot V54 \cdot \overline{D79} \cdot \overline{D52}$	$1 \longrightarrow$ PI
" "	$R10 \cdot (P6 \vee I123) \cdot V42 \cdot \overline{D79} \cdot \overline{D52}$	$3 \longrightarrow$ PI

IT4			Trigger AN2CC
	Transfer sum to MA	I53	AN2 $\longrightarrow$ MA
IT5			Trigger AN2CC
	Transfer sum to X	I130	(MA) $\longrightarrow$ X
	Prepare AN2 for equality comparison	I129	1 $\longrightarrow$ AN2I
	AN2 inputs for	I29	D $\longrightarrow$ SW6
	comparison	I53	0 $\longrightarrow$ SW1
		I29	+ $\longrightarrow$ PM

(Note: X is already connected to SW4)

IT6			
	No activity		
IT7			
	$(X) \neq (D_0)$, jump	I53 · $\overline{\text{AN2CC}}$	1 $\longrightarrow$ JFF
	Preparations for jump	" "	1 $\longrightarrow$ SW1
	address to PA	" "	0 $\longrightarrow$ SW4
		" "	0 $\longrightarrow$ AN2I
		" "	0 $\longrightarrow$ PA
	Program control for jump	" "	1 $\longrightarrow$ PI
IT8			Trigger AN2CC
	If jumping	JFF = 1	AN2 $\longrightarrow$ PA
		"	J $\longrightarrow$ Mod 2
	End	$\overline{\text{I63}}$	IT8 $\longrightarrow$ END

The SIXJ instruction is the same except the twos complement of (I_V) is sent to AN2 in IT1 by:

$$- \longrightarrow \text{PM}, \quad 1 \longrightarrow \text{AN2C}$$

AIXO, SIXO

The remainder of the third group of index register instructions (Add or subtract, then compare) will set the Overflow flip-flop if $(X) = (D_V)$.

011	1110	AIXO	$(X) + (I_V) \longrightarrow X$; then $1 \longrightarrow OF$ if $(X) = (D_V)$ where I_J specifies which half of D and D_R specifies the size of D_V.
011	1111	SIXO	Same except $(X) - (I_V) \longrightarrow X$.

In IT1, the controls are cleared, the AN2 inputs connected ($X \longrightarrow SW4$, $PR \longrightarrow SW6$, + or - $\longrightarrow$ PM, 0 or 1 $\longrightarrow$ AN2C), MA is cleared and $I_J \longrightarrow SW1$.

In IT2, the sum is transferred to MA and in IT5 to X. In IT5, preparations are made for the comparison ($D \longrightarrow SW6$, + $\longrightarrow$ PM, 1 $\longrightarrow$ AN2I). If (X) = (D) then in IT7, the Overflow flip-flop will be set

$$IT7 \cdot 011\ 111X \cdot AN2CC \Longrightarrow 1 \longrightarrow OF$$

The instruction goes to END through IT8.

6.10 Address Substitution and Increase

<u>Address Substitution TJM</u> 000 1000

The purpose of the TJM instruction is to replace the address field of an instruction, stored in memory, with the contents of JA. I_V of the TJM instruction, possibly modified by an index register, and the J bit of TJM give the memory location of the instruction to be altered.

The sequence of operations is to transfer memory to D, transfer (JA) to the appropriate D_V, then transfer (D) back to the same memory location. The ja_{-1} bit is transferred to D_j.

<u>Logic of TJM</u>

IT1

Clear controls	$0 \longrightarrow$ AN21, EO, UF, SC, FI, AN1CE
	Inh. AN2
Connect AN2 inputs for memory address:	$0 \longrightarrow$ AN2C
I_V alone	$0 \longrightarrow$ SW4, PR $\longrightarrow$ SW6, + $\longrightarrow$ PM
Index modified address	$X \longrightarrow$ SW4, PR $\longrightarrow$ SW6, + $\longrightarrow$ PM
In repeat mode, $\alpha\gamma = 1$	$X \longrightarrow$ SW4, PR $\longrightarrow$ SW6, 0 $\longrightarrow$ PM
Clear for transfers	$0 \longrightarrow$ MA, D
Select half of D for (JA) $\longrightarrow$ D	0 or 1 $\longrightarrow$ SW1

IT2 Trigger AN2CC

Program control	(SW2)' $\longrightarrow$ Mod 2
Transfer address	AN2 $\longrightarrow$ MA
Read Memory	$2 \longrightarrow$ MI
	MT5 $\longrightarrow$ IT3

IT3

Program Control	0, 1, 2 or 3 $\longrightarrow$ PI

IT4 - IT7 no significant activity

IT8 (Illustration for 32 index registers)

$(JA) \longrightarrow D$	$I35 \cdot SW1 = 0$	$(ja_9 \cdots ja_0) \longrightarrow d_{-6} \cdots\cdots d_{-15}$
	$I35 \cdot SW1 = 0 \cdot d_0 = 0$	$(ja_{14} \cdots ja_{10}) \longrightarrow d_{-1} \cdots\cdots d_{-5}$
	$I35 \cdot SW1 = 1$	$(ja_9 \cdots ja_0) \longrightarrow d_{-30} \cdots\cdots d_{-39}$
	$I35 \cdot SW1 = 1 \cdot d_{-24} = 0$	$(ja_{14} \cdots ja_{10}) \longrightarrow d_{-25} \cdots d_{-29}$
	$I137 \cdot SW1 = 0$	$(ja_{-1}) \longrightarrow d_{-16}$
	$I137 \cdot SW1 = 1$	$(ja_{-1}) \longrightarrow d_{-40}$
End		IT8 --- END

Transfer Instruction Operand Address   TIJ   000   1010

The TIJ instruction is usually used in conjunction with the TJM instruction as one type of address substitution procedure.

TIJ cause the contents of I_V to be transferred to JA. The contents of I_V may be increased by (X) with the standard "effective address" logic of IT1. The ja_{-1} bit is given the value of the J bit of the instruction.

The essential logic details are that in IT1 and IT2 the substitute address is transferred to MA, via AN2, from PR, X or a combination of the two.

JA is cleared to 0 in IT1 and in IT2; if I83 is active (1 000 1010), $1 \longrightarrow ja_{-1}$. $(MA) \longrightarrow$ JA in IT7 and the instruction goes to End from IT8.

Increase Address Instruction INCA 000 1001

INCA will increase, by one, the address field of an instruction stored in memory. The instruction word that will undergo the increase is read from memory. Then the selected D_V is connected to AN2 where a one is added to it ($1 \longrightarrow$ AN2C). The increased address is returned to memory via MA, JA, and D.

The usual considerations of the I_J and D_R bits apply, wherein the J bit of INCA specifies the half-word to be referenced at the D register and the R bit of that half-word indicates the size of the address field.

In IT1 and IT2 the effective address is placed in MA to access the memory for the operand word. D and JA are cleared to zero and a read memory operation is initiated.

In IT3, D is connected as the sole register input to AN2. An increment is formed by $1 \longrightarrow$ AN2C. The sum is transferred to MA in IT4 and on to JA in IT7. The address for memory access is again transferred to MA in IT7 and IT8. The (JA) $\longrightarrow$ D transfer occurs in IT8 using the same logic as the TJM instruction, (the ja_{-1} bit is, of course, not transferred). Lastly a write memory operation is called for by $3 \longrightarrow$ MI. The instruction goes to End from IT8.

The method of increasing the address means it can pass through zero without overflowing into the bits on the left (the R bit or index register address, as the case may be).

6.11 Reference Change LWD SWD

Larger Word	1 000	1111	LWD	If $(V) > (A)$, alphanumeric sense, $(V) \longrightarrow A$, address of $V \longrightarrow JA$.
Smaller Word	1 000	1111	SWD	If $(V) < (A)$, alphanumeric sense, $(V) \longrightarrow A$, address of $V \longrightarrow JA$.

These two instructions enable the sorting and merging of records.

The comparison method is similar to that used for some of the conditional jump instructions. The LWD instruction makes the test $(D) > (A)$? This is so if (D) are neither equal to nor smaller than (A).

The procedure:

$$(D) > (A) \text{ if } \left[(A) + 2 - (D)\right] < 2$$

The absence of a carry from the most significant sum bit position ($ANIC_0 = 0$) indicates a sum smaller than 2.

However, the test for the SWD instruction must exclude the case of equality of the two words. The method is revised in this fashion:

$$\text{If } (D) = (A), \quad \left[(A) + 2 - (D)\right] = 2$$

$$\text{If } (D) \leqq (A), \quad \left[(A) + 2 - (D)\right] \geqq 2$$

To eliminate the case of equality, the addend is initially decreased by the smallest possible amount. If the sum then is equal to or greater than 2, $(D) < (A)$.

$$(D) < (A) \text{ if } \left[(A) + 2 - 2^{-47} - (D)\right] \geqq 2$$

$2-2^{-47} - (D)$ is the ones complement, derived by $(D)' \longrightarrow D$, $0 \longrightarrow ANIC$.

A sum equal to or greater than 2 (10.0 in binary form) is indicated by a carry from the most significant sum bit position ($ANIC_0 = 1$).

If the tests for either instruction indicate (A) are to be replaced, II is set to 1 for the subsequent activity.

Logic Summary

In IT1 the usual control clearings and preparations to transfer the effective address to MA are performed. D is cleared for the read-out from memory.

The read and restore operation ($4 \longrightarrow$ MI) is begin in IT2. Upon completion, MT11 $\longrightarrow$ IT3.

During IT3, PI is set to the required state and D* is cleared for transfer of the complement of (D). ANIC is set to 1 for the LWD instruction or set to 0 for SWD.

$$IT3 \cdot D87 \cdot \overline{I118} \Longrightarrow 1 \longrightarrow ANIC$$

$$IT3 \cdot D93 \Longrightarrow 0 \longrightarrow ANIC$$

The complement is transferred to D* in IT4 and the test made in IT5.

IT5		Trigger ANICC
(LWD)	$137 \cdot 169 \cdot V64$	$1 \longrightarrow$ II
(SWD)	$137 \cdot 170 \cdot V65$	$1 \longrightarrow$ II

$I37 \equiv$ LWD v SWD

$I69 \equiv$ J bit = 0 $\qquad$ $I70 \equiv$ J bit = 1

$V64 \equiv ANIC_0 = 0$ $\qquad$ $V\ 65 \equiv ANIC_0 = 1$

The above active conditions will clear JA in IT6.

In IT7 with II = 1 (for these instructions), A is cleared and (MA) transferred to JA. MA, undisturbed since IT2, contains the address of the word compared with (A). In IT8, (D) is transferred to A and the instruction goes to END.

6.12 Bit-by-Bit Boolean

The following instructions are grouped by a close functional relationship. For while they may be only loosely related from a programming viewpoint, their methods of performance are very similar. The word contents are processed, bit-by-bit, in accordance with some of the rules of Boolean algebra.

The register symbols used in this subsection will be lower case letters to indicate the reference to any corresponding single bit positions of registers or memory. The letter, "m" shall be used to denote memory bit positions to avoid confusion with the inclusive OR symbol.

The instructions will be first described from the viewpoint of the results they produce. Following this, logic will be discussed.

Extract Instructions

The extract instructions enable the isolation of any desired portion or portions (fields) of a word and their transfer into a pre-cleared register. The selection is controlled by the bit pattern of a word in another register. The word is often termed the "mask". As the extract instruction is usually applied, field(s) of a word in memory are transferred to D under control of the bit pattern of a word in Q.

$$\text{when } q_i = 0, \quad 0 \longrightarrow d_i$$

$$\text{when } q_i = 1, \quad m_i \longrightarrow d_i$$

The ETD instruction accomplishes this extraction. The process is a bit-by-bit binary multiplication, which is also termed "logical multiply".

$$(m_i) \cdot (q_i) \longrightarrow d_i \qquad (1 \cdot 0) = (0 \cdot 1) = (0 \cdot 0) = 0; \quad (1 \cdot 1) = 1$$

(The algebraic symbol for multiply, " • ", is undoubtedly the source of the AND symbol.)

It might be noted that as far as Q and M are concerned, the roles of the mask word and extracted word are interchangeable.

Variations of the extract instructions are listed below:

0	000	1100	ETD	$m_i \cdot q_i \longrightarrow d_i$
1	000	1100	ETA	$m_i \cdot q_i \longrightarrow d_i, a_i$
	111	1010	EA	$m_i \cdot q_i \longrightarrow d_i$; $(A) + (D) \longrightarrow A$
	111	1011	ES	$m_i \cdot q_i \longrightarrow d_i$; $(A) - (D) \longrightarrow A$

The latter two, EA and ES, are two-part instructions. Following the extract operation, numerical arithmetic is performed with the contents of A and D. This can be fixed or floating point arithmetic.

Insert Instructions

The insert instructions are a further variation of extract. The extracted field(s) are used as a replacement for corresponding bit positions of an uncleared register. The extracted bits are inserted into a word in the A register. This is not a general logical multiply. The location of the mask is fixed, the Q register, and the extraction is from memory:

when $q_i = 1$, $m_i \longrightarrow a_i$

when $q_i = 0$, a_i is left unaltered

The S-2000 has two insert instructions.

0	000	1110	EI	Extract M per Q and insert in A.
1	000	1110	EIS	Do above and store (A).

Word Merging

The inclusive OR merging instruction is DORMS, which combines all the 1 bits of the word in D and in memory. This word replaces the original contents of the memory location.

0	000	1101	DORMS	$d_i \vee m_i \longrightarrow m_i$

The exclusive OR merging instruction is:

1 000 1101 AWCS $a_i \wedge m_i \longrightarrow m_i$

It combines the words in A and D, bit-by-bit, by the exclusive OR rule. Effectively this is "add without carry", hence the mnemonic title of the instruction. The formed word is stored in memory.

$$(a_i = 1 \cdot d_i = 0) \ v \ (a_i = 0 \cdot d_i = 1) \Longrightarrow 1 \longrightarrow m_i$$

$$a_i = d_i \Longrightarrow 0 \longrightarrow m_i$$

Methods

The Boolean algebra can be simply accomplished by means of transfers between registers. The sequence of the standard method used for extract is:

$$\left.\begin{array}{ll} 1. & 0 \longrightarrow D \\ 2. & M \xrightarrow{1} D \\ 3. & Q \xrightarrow{0} D \end{array}\right\} \quad m_i = 1 \cdot \overline{q_i = 0} \Longrightarrow 1 \longrightarrow d_i$$

First the 1 bits in memory are transferred to the cleared D register. All other bit positions will be 0. If any bit postion of Q is 0, the corresponding bit position of D should be zero. Secondly, therefore, the 0 bits of Q are transferred to D to change back to 0 any of these bits that the memory read had changed to 1.

The above is the essence of the ETD instruction. ETA will perform the extract and then transfer (D) to the pre-cleared A register. EA and ES do the extract and then add or subtract (D) from (A).

Table 6.12-1 compiles the logic details of these instructions.

Insert performs the operation:

$$q_i = 1 \Longrightarrow (m_i = 1 \ v \ m_i = 0) \longrightarrow d_i, \ a_i$$

The process may be reasoned thus ——

The information in a_i is to be changed only when q_i is 1. Where $q_i = 1$, delete the existing information by $0 \longrightarrow a_i$. Then extract all the m_i bits with the mask of $q_i = 1$. And lastly transfer all the extracted m_i bits to a_i. The $m_i = 0$ bits need not be transferred as the corresponding a_i were previously cleared to 0.

The sequence becomes the following:

1.	$1 \longrightarrow D$	
2.	$(A) \xrightarrow{0} D$	Transfer $(A) \longrightarrow D$
3.	$(Q)' \xrightarrow{0} D$	Where $q_i = 1$, send a $0 \longrightarrow d_i$
4.	$0 \longrightarrow A$	Transfer back to A with deletes
5.	$(D) \xrightarrow{1} A$	
6.	$(M) \xrightarrow{1} D$	Standard extract process
7.	$(Q) \xrightarrow{0} D$	
8.	$(D) \longrightarrow A$	

(To utilize existing circuitry, the transfer of the Q complement to A is via Q* .)

The logic details are listed in Table 6.12-1. The preparations for the subsequent numerical arithmetic can be performed during the same timings as the extract. (SC) $\longrightarrow$ SC* is done in IT2 to clear the latter, as IT1 $\Longrightarrow 0 \longrightarrow$ SC.)

Word Merge

The DORMS instruction is the simplest of the group:

$$(d_i = 1) \quad v \quad (m_i = 1) \quad \Longrightarrow \quad 1 \longrightarrow m_i$$

The ones of the selected memory location are transferred to D by a read. The restore (part of $4 \longrightarrow$ MI) will then transfer the merged word to memory. The instruction has been achieved.

AWCS

AWCS could be accomplished solely by means of transfers, as the other instructions. However, this would require more than eight timings. An alternate method was chosen to perform it more quickly.

The adder network permits the logical operation:

$$a_i \wedge d^*_i \longrightarrow$$

This is part of the development of the sum bit. The inverse of $a_i \wedge d^*_i$ is $a_i = d^*_i$. The latter is developed for each bit in AN1, and is titled "$A_1\ D_1$ v $A_0\ D_0$". See Figure 5.1-1. For addition, this signal is used for further logic with the carry-in.

The process for AWCS is to transfer (M) $\longrightarrow$ D*; clear D to ones: and then set d_i to 0 where $a_i = d^*_1$.

$$(A \wedge D^* \longrightarrow D) \cdot a_i = d^* \Longrightarrow 0 \longrightarrow d_i$$

Using AN1, the instruction requires six timings, however IT1 through IT8 is taken for its performance.

	ETD 0 000 1100	ETA 1 000 1100	EA 111 1010	ES 111 1011	EI 0 000 1100	EIS 1 000 1100	DORMS 0 000 1101	AWCS 1 000 1101
IT1 ①	$0 \rightarrow D$	$0 \rightarrow D$	$0 \rightarrow D$	$0 \rightarrow D$	$1 \rightarrow D$ $1 \rightarrow Q^*$	$1 \rightarrow D$ $1 \rightarrow Q^*$		$0 \rightarrow D$
IT2	$AN2 \rightarrow MA$ $4 \rightarrow MI$ $(SW2)' \rightarrow \text{Mod } 2$	$AN2 \rightarrow MA$ $4 \rightarrow MI$ $(SW2)' \rightarrow \text{Mod } 2$	$AN2 \rightarrow MA$ $4 \rightarrow MI$ $1 \rightarrow AI$ $(SC) \rightarrow SC^*$ $(SW2)' \rightarrow \text{Mod } 2$	$AN2 \rightarrow MA$ $4 \rightarrow MI$ $1 \rightarrow AI$ $(SC) \rightarrow SC^*$ $(SW2)' \rightarrow \text{Mod } 2$	$AN2 \rightarrow MA$ $(A) \xrightarrow{0} D$ $(Q) \xrightarrow{0} Q^*$ $(SW2)' \rightarrow \text{Mod } 2$	$AN2 \rightarrow MA$ $(A) \xrightarrow{0} D$ $(Q) \xrightarrow{0} Q^*$ $(SW2)' \rightarrow \text{Mod } 2$	$AN2 \rightarrow MA$ $4 \rightarrow MI$ $(SW2)' \rightarrow \text{Mod } 2$	$AN2 \rightarrow MA$ $2 \rightarrow MI$ $(SW2)' \rightarrow \text{Mod } 2$
IT3	⑥ $(Q) \xrightarrow{0} D$	$(Q) \xrightarrow{0} D$ $0 \rightarrow D$	$(Q) \xrightarrow{0} D$ $0 \rightarrow D^*_M$ $0 \rightarrow AN1C$ ② $0 \rightarrow D^*_E$ ③ $1 \rightarrow D^*_E$	$(Q) \xrightarrow{0} D$ $1 \rightarrow D^*_M$ $1 \rightarrow AN1C$ $1 \rightarrow D^*_E$	⑤ $(Q^*)' \xrightarrow{0} D$ $0 \rightarrow A$	$(Q^*)' \xrightarrow{0} D$ $0 \rightarrow A$		$0 \rightarrow D^*$ $0 \rightarrow AN1C$
IT4	$\rightarrow END$	$(D) \xrightarrow{1} A$ $\rightarrow END$	$(D_M) \xrightarrow{1} D^*_M$ ② $(D_E) \xrightarrow{1} D^*_E$ ③ $(D_E)' \xrightarrow{0} D^*_E$ ③ $1 \rightarrow AN1CE$ $\rightarrow$ AT1 or FT1	④ $(D)' \xrightarrow{0} D^*$ $\rightarrow$ AT1 or FT1	$(D) \xrightarrow{1} A$ $4 \rightarrow MI$	$(D) \xrightarrow{1} A$ $2 \rightarrow MI$	$\rightarrow END$	$(D) \xrightarrow{1} D^*$ $\rightarrow IT5$
IT5					⑥ $(Q) \xrightarrow{0} D$	$(Q) \xrightarrow{0} D$		
IT6					⑦ $(D) \xrightarrow{1} A$	$(D) \xrightarrow{1} A$		
IT7						$1 \rightarrow D$		$1 \rightarrow D$
IT8					$\rightarrow END$	$(A) \xrightarrow{0} D$ $3 \rightarrow MI$ $\rightarrow END$		⑧ $(A \wedge D^*) \xrightarrow{0} D$ $3 \rightarrow MI$ $\rightarrow END$

Notes

① Also control clearings, effective address to AN2

② Fixed Point

③ Floating Point

④ $[(D)' \rightarrow D^*] \cdot d_i = 1 \Longrightarrow 0 \rightarrow d^*_i$

⑤ $[(Q^*)' \xrightarrow{0} D] \cdot q^*_i = 1 \Longrightarrow 0 \rightarrow d_i$

⑥ $[(Q) \xrightarrow{0} D] \cdot q_i = 0 \Longrightarrow 0 \rightarrow d_i$

⑦ $[(D) \xrightarrow{1} A] \cdot d_i = 1 \Longrightarrow 1 \rightarrow a_i$

⑧ $[(A \wedge D^*) \xrightarrow{0} D] \cdot a_i = d^*_i \Longrightarrow 0 \rightarrow d_i$

Figure 6.12-1 Logic of Bit-by-Bit Boolean

6.13 Computer Stop HLT CF

Halt Instruction HLT 000 0000

The computer response is as the instruction title indicates. The stop is achieved by setting the Stop FF to 1 during the performance of HLT. This, in turn, causes the P20 signal to become normal. P20 active is required for either exit from PT4, PT4 $\longrightarrow$ IT1 or PT4 $\longrightarrow$ PT1.

Precisely defined, the computer will stop after the first set of PTs following the halt instruction unless PI = 01. Either of several activities can occur during the PTs dependent upon the coding of the Halt Instruction.

If the R bit is 0 and HLT is the I_0 instruction, the standard PI = 00 activity will occur. The stop will be before IT1 of the I_1 instruction.

If R = 1 and $X_C = 1$, the index register will be counted and then the computer will stop.

If, for no discernible program purpose, HLT is performed as the I_1 instruction in the repeat mode, the computer will stop after the index register is being counted or repeat modified. If X is not being changed, the stop will be after the Repeat counter is counted down.

If PI = 01, the computer will stop before transferring the next instruction word.

Logic of Halt

IT1

Standard control clearings and AN2 input connections, but MA is not cleared:

$$D65 \cdot \overline{D52} \cdot \overline{I41} \cdot \overline{I81} \Longrightarrow 0 \longrightarrow MA$$

where I41 $\equiv$ 000 0000.

Set Stop FF	I41	$1 \longrightarrow$ Stop

IT2

Program control I41 $(SW2)' \longrightarrow$ Mod 2

The "address" is not transferred to MA.

$$D65 \cdot \overline{I41} \cdot \overline{I81} \cdot \overline{D52} \Longrightarrow AN2 \longrightarrow MA$$

IT3

$$\text{Mod } 2 = 1 \cdot \overline{D79} \cdot I = 0 \cdot \alpha\gamma = 0 \cdot \left[(R = 1 \cdot X_C = 0) \vee I123\right]$$
$$\Longrightarrow 0 \longrightarrow PI, PI^*$$

$$\text{Mod } 2 = 0 \cdot \overline{D79} \cdot \overline{D52} \cdot RPT = 0 \quad \left[(R = 1 \cdot X_C = 0) \vee I123\right]$$
$$\Longrightarrow 1 \longrightarrow PI$$

$$\left[(R = 1 \cdot X_C = 1) \vee \alpha\gamma = 1\right] \cdot \overline{D79} \cdot \overline{D52} \cdot \overline{I123} \Longrightarrow 2 \longrightarrow PI$$

$$RPT = 1 \cdot \alpha\gamma = 0 \cdot SW2 = 1 \cdot \overline{D79} \cdot \overline{D52} \cdot \left[(R = 1 \cdot X_C = 0 \vee I123\right]$$
$$\Longrightarrow 3 \longrightarrow PI$$

IT4

End $(I41 \vee I13 \vee I125 \vee I126) \cdot JFF = 0 \longrightarrow$ END

END

$E1 \cdot PI = 1X \longrightarrow PT1$

$E1 \cdot PO \longrightarrow PT3$

Visual Indication

The stop due to HLT can readily be identified by the Stop FF neon on the console. Only two conditions set the flipflop to 1.

$$IT\,2 \cdot I41 \Longrightarrow 1 \longrightarrow \text{Stop}$$

$$MA = MP \cdot MA\ \text{Stop} \cdot M10 \cdot MT10 \Longrightarrow 1 \longrightarrow \text{Stop}$$

Command Fault CF

A command fault is a command coding pattern which has not been assigned to any instruction. Of the 256 possible codings, 22 are unassigned. If, regardless of the cause, any of these appear in the operative half of PR, the computer is stopped and the CF flip-flop is set to 1.

These illegitimate codings are called "Command Faults" when they are decoded.

Coding	Decoded As
111 01XX	I3
111 111X	I4
011 011X	I6
011 1000	I7
011 101X	I8

The above I numbers are combined in an OR gate to form the D52 signal.

D52, when active, causes certain logical activity. Its purpose is to return the computer to the point in program control just prior to the "instruction", namely PT4. Here the computer is stopped to permit examination and manual intervention to correct the command coding in PR.

With D52 active, the computer goes through IT1 to IT4, End and PT2 through PT4. The only activity is to set the CF flipflop (which makes P20 normal); and the control clearings performed by IT1 for all instructions.

D52, when active, prevents any other changes in data or control information. The nature of these changes is determined by the instruction replacement coding for CF, which is not known to the computer.

The computer stops after PT4 since P20 is normal.

Logic of CF

IT1		
		$0 \longrightarrow \text{EO, UF, SC, AN1I, AN2I}$
		Inh AN2
	D52	$1 \longrightarrow \text{CF}$

The following are inhibited by D52 active:

		$0 \longrightarrow \text{OF}$
		$0 \longrightarrow D_{M1}, D_{M2}, D_{M3}$
		$0 \longrightarrow D_{E1}, D_{E2}$
		$0 \longrightarrow \text{MA}$
IT2	D52	$\longrightarrow \text{IT3}$

The following are inhibited by D52 active:

		$\text{AN2} \longrightarrow \text{MA}$
		$(\text{SW2})' \longrightarrow \text{Mod2}$
		$2 \longrightarrow \text{MI}$
		$4 \longrightarrow \text{M1}$
IT3	D52	$0 \longrightarrow \text{PI, PI*}$

The following are inhibited by D52 active:

		$1 \longrightarrow \text{PI}$
		$2 \longrightarrow \text{P1}$
		$3 \longrightarrow \text{P1}$

IT4

D52 $\longrightarrow$ END

The following are inhibited by D52 active:

$\longrightarrow$ AT1

$\longrightarrow$ IT5

$\longrightarrow$ FT1

END

PI = 00 · E1 $\Longrightarrow$ $\longrightarrow$ PT2

[Note: E1 $\equiv$ (IT4 v IT8) $\longrightarrow$ END]

PT4

Computer stops as P20 active is required for exit to IT1 or PT1.

6.14 Console Transfer Instructions TTD TCM TDC

In addition to the manual console switches, three instructions provide means of transfers between computer and console.

1	000	0100	TTD	(TR) $\longrightarrow$ D
0	000	0101	TCM	Console Typewriter $\longrightarrow$ V
1	000	0101	TDC	(D) $\longrightarrow$ Console Typewriter

Toggle Register

The bank of 48 miniature, bat-handled toggle switches is manually preset for the desired word. The "up" position of the handle produces a bit value of 1.

The instruction is performed during IT1 to IT4. The D register is cleared to zero in IT1; the transfer is done in IT2; the instruction goes to End from IT4.

The instruction incorporates the standard index register activity which may be used as a performance counter.

TABLE OF POWERS OF 2

2^n	n	2^{-n}
1	0	1.0
2	1	0.5
4	2	0.25
8	3	0.125
16	4	0.062 5
32	5	0.031 25
64	6	0.015 625
128	7	0.007 812 5
256	8	0.003 906 25
512	9	0.001 953 125
1 024	10	0.000 976 562 5
2 048	11	0.000 488 281 25
4 096	12	0.000 244 140 625
8 192	13	0.000 122 070 312 5
16 384	14	0.000 061 035 156 25
32 768	15	0.000 030 517 578 125
65 536	16	0.000 015 258 789 062 5
131 072	17	0.000 007 629 394 531 25
262 144	18	0.000 003 814 697 265 625
524 288	19	0.000 001 907 348 632 812 5
1 048 576	20	0.000 000 953 674 316 406 25
2 097 152	21	0.000 000 476 837 158 203 125
4 194 304	22	0.000 000 238 418 579 101 562 5
8 388 608	23	0.000 000 119 209 289 550 781 25
16 777 216	24	0.000 000 059 604 644 775 390 625
33 554 432	25	0.000 000 029 802 322 387 695 312 5
67 108 864	26	0.000 000 014 901 161 193 847 656 25
134 217 728	27	0.000 000 007 450 580 596 923 828 125
268 435 456	28	0.000 000 003 725 290 298 461 914 062 5
536 870 912	29	0.000 000 001 862 645 149 230 957 031 25
1 073 741 824	30	0.000 000 000 931 322 574 615 478 515 625
2 147 483 648	31	0.000 000 000 465 661 287 307 739 257 812 5
4 294 967 296	32	0.000 000 000 232 830 643 653 869 628 906 25
8 589 934 592	33	0.000 000 000 116 415 321 826 934 814 453 125
17 179 869 184	34	0.000 000 000 058 207 660 913 467 407 226 562 5
34 359 738 368	35	0.000 000 000 029 103 830 456 733 703 613 281 25
68 719 476 736	36	0.000 000 000 014 551 915 228 366 851 806 640 625
137 438 953 472	37	0.000 000 000 007 275 957 614 183 425 903 320 312 5
274 877 906 944	38	0.000 000 000 003 637 978 807 091 712 951 660 156 25
549 755 813 888	39	0.000 000 000 001 818 989 403 545 856 475 830 078 125
1 099 511 627 776	40	0.000 000 000 000 909 494 701 772 928 237 915 039 062 5
2 199 023 255 552	41	0.000 000 000 000 454 747 350 886 464 118 957 519 531 25
4 398 046 511 104	42	0.000 000 000 000 227 373 675 443 232 059 478 759 765 625
8 796 093 022 208	43	0.000 000 000 000 113 686 837 721 616 029 739 379 882 812 5
17 592 186 044 416	44	0.000 000 000 000 056 843 418 860 808 014 869 689 941 406 25
35 184 372 088 832	45	0.000 000 000 000 028 421 709 430 404 007 434 844 970 703 125
70 368 744 177 664	46	0.000 000 000 000 014 210 854 715 202 003 717 422 485 351 562 5
140 737 488 355 328	47	0.000 000 000 000 007 105 427 357 601 001 858 711 242 675 781 25

ALPHANUMERIC CODING

	00			01			10			11		
	Line Printer	Console Typewriter		Line Printer	Console Typewriter		Line Printer	Console Typewriter		Line Printer	Console Typewriter	
		Upper Case	Lower Case		Upper Case	Lower Case		Upper Case	Lower Case		Upper Case	Lower Case
0000	0	0	)	&	&	*	-	-	/	△ (Space)	Space	Space
0001	1	1	"	A	A	A	J	J	J	/	None	None
0010	2	2	@	B	B	B	K	K	K	S	S	S
0011	3	3	Σ	C	C	C	L	L	L	T	T	T
0100	4	4	$	D	D	D	M	M	M	U	U	U
0101	5	5	∈	E	E	E	N	N	N	V	V	V
0110	6	6	%	F	F	F	O	O	O	W	W	W
0111	7	7	>	G	G	G	P	P	P	X	X	X
1000	8	8	;	H	H	H	Q	Q	Q	Y	Y	Y
1001	9	9	(	I	I	I	R	R	R	Z	Z	Z
1010	∈	None	None	r ("ignore" char.)	carriage return		t (cond. stop)	Tab	TAB	i (abs. stop)	Stop	Stop
1011	#	#	?	.	.	→	$	None	None	,	,	:
1100	@	None	None	⊣	None	None	*	None	None	%	None	None
1101	=	None	None	'	'	|	"	None	None	>	None	None
1110	+	+	=	)	None	None	(	None	None	→	None	None
1111	;	None	None	?	Upper Case Shift		:	Lower Case Shift		Σ (line marker)	Code Delete	

NOTES:

1. The symbols r, △, and Σ are printed only in "memory dump" mode. Otherwise the corresponding codes are used for control characters.
2. An editing program is normally used to convert between the console typewriter code and the line printer code.
3. The paper tape control recognizes the Code 111010 as a stop instruction.

SYMBOLS

( )	Contents of
$\| \|$	Absolute value of quantity.
A	A register.
D	D register.
D_0, D_1	Left and right halves, respectively of D.
D_J, I_J	Most significant bit of 8-bit command code, bit 2_{-16} or 2_{-40}
;	Indicates a sequence. Operation at left of semicolon first, followed by operation to right of semicolon.
I	Part of Program Register currently operative. (I) will be the instruction currently being performed.
I_V, D_{0V}, D_{1V}	The "V" indicates address field of the 24 bit instruction (or half-word).
JA	Jump Address register.
ja_{-1}	The bit in JA corresponding to D_{0J} or D_{1J} of D, I_J of I, and X_C of X.
Q	Q register.
V	Selected memory location.
X	Selected index register.
X_C	Counter bit of selected index register.

	Quaternary Code	Mnemonic Code	Special
	00	HLTL	Halt, Stop computer
	01	JBTL	Stop if BREAKPOINT switch ON. If ADVANCE then given, jump to instruction in location given by (I_V) and (I_J). If BREAKPOINT switch originally OFF, jump without stopping.
	02	ICOL	Inhibit clearing of Overflow flipflop.
	03	NOPL	No operation (filler). Continue to next instruction.
	10	TIO	Skip next instruction if I-0 order in D is accepted by I-0 control.
	11	TCM	Transfer one alphanumeric character from console typewriter to rightmost part of D (d_{-40} ... d_{-47}). Remainder of D unchanged. Then (D) V.
00	12	SKC	Skip next instruction if specified I-0 transmission is completed.
	13	TCXZ	0 X_C
	20	TJML	(V) D; (JA) D_{0V}, (ja_{-1}) D_{0J}; (D) V.
	21	INCAL	(V) D; (D_{0V}) + 1 D; (D) V.
	22	TIJL	(I_V) JA, 0 ja_{-1}
	23	RPT	Establish repeat mode where I_V is number of times instruction(s) are to be performed. If RPT is I_0, repeat I_1. If RPT is I_1, repeat the two instructions of the next word.
	30	ETD	Extract to D; bit-by-bit logical multiply: (V) (Q) D where 0 0 = 0 1 = 1 0 = 0; 1 1 = 1

Quaternary Code	Mnemonic Code	Special
31	DORMS	Word merge, bit-by-bit combining of ones: (V) $\circ$ (D) $\longrightarrow$ D, V where $0 \cdot 1 = 1 \cdot 0 = 1 \quad 1 = 1, \ 0 \cdot 0 = 0$
00 32	EI	Insert in A; where any bit positions of Q have a value of 1, insert the value of the corresponding bit positions of V in A. Leave the balance of A unchanged.
33	LWD	If (V) > (A), Alphanumeric sense, (V) $\longrightarrow$ A and address of V $\longrightarrow$ JA.

20

		Special
00	HLTR	Same as HLTL
01	JBTR	Same as JBTL, except jump to right.
02	ICOR	Same as ICOL
03	NOPR	Same as NOPL
10	TTD	(TR) $\longrightarrow$ D
11	TDC	(d_0 ... d_{-5}) $\longrightarrow$ console typewriter (One alphanumeric character).
12	SKF	Skip next instruction if specified I-0 Faults have not occurred.
13	TCXS	1 $\longrightarrow$ X_C
20	TJMR	Same as TJML, except transfer to D_1 .
21	INCAR	Same as INCAL, except increase D_{1V}.
22	TIJR	Same as TIJL, except 1 $\longrightarrow$ ja_{-1}.
23	RPT	
30	ETA	Same as ETD, followed by (D) $\longrightarrow$ A.
31	AWCS	Add without carry and store; bit-by-bit logical add of (V) + (A) $\longrightarrow$ D, V. where $0 + 1 = 1 + 0 = 1$; $0 + 0 = 1 + 1 = 0$
32	EIS	Same as EI, followed by (A) $\longrightarrow$ D, V.
33	SWD	If (V) < (A), alphanumeric sense, (V) $\longrightarrow$ A and address of V $\longrightarrow$ JA.

Transfers

	00	CM	$0 \longrightarrow V$
	01	TMA	$(V) \longrightarrow A$
	02	TMQ	$(V) \longrightarrow Q$
	03	TMD	$(V) \longrightarrow D$
	10	TAM	$(A) \longrightarrow V$
	11	CA	$0 \longrightarrow A$
	12	TAQ	$(A) \longrightarrow Q$
01	13	TAD	$(A) \longrightarrow D$
	20	TQM	$(Q) \longrightarrow V$
	21	TQA	$(Q) \longrightarrow A$
	22	CQ	$0 \longrightarrow Q$
	23	TQD	$(Q) \longrightarrow D$
	30	TDM	$(D) \longrightarrow V$
	31	TDA	$(D) \longrightarrow A$
	32	TDQ	$(D) \longrightarrow Q$
	33	CD	$0 \longrightarrow D$

Shifts

	00	SLAQ	Shift left, treating A, Q as one double-length word.
	01	SRAQ	Shift right, treating A, Q as one double-length word.
	02	SLAQN	Shift left, treating A, Q as one double-length number and sign.
	03	SRAQN	Shift right, treating A, Q as one double-length number and sign.
	10	SLA	
	11	SRA	
	12	SRAN	
	13	SRAN	
21	20	SLQ	
	21	SRQ	
	22	SLQN	
	23	SRQN	
	30	SCD	Circular right shift (D).
	31	SRD	
	32	SCD	
	33	SRDN	

Note: See Figure 6.5-1 for illustration of shifts.

Jumps - Left

(Note: Jump to instruction whose address is given by (I_V, I_J). If $I_J = 0$, instruction is left half-word of (V).)

02	00	JMPL	Unconditional jump
	01	JAZL	Jump if (A) = 0
	02	JNOL	Jump if OF = 0
	03	JOFL	Jump if OF = 1
	10	JAPL	Jump if positive, $(A) \geqq 0$
	11	JANL	Jump if negative, $(A) < 0$
	12	JAEQL	Jump if (A) = (Q)
	13	JAEDL	Jump if (A) = (D)
	20	JQPL	Jump if $(Q) \geqq 0$, unconditionally circular shift (Q) left one place.
	21	JQNL	Jump if $(Q) < 0$, unconditionally circular shift (Q) left one place.
	22	JQEL	Jump if Q is even, unconditionally circular shift (Q) right one place.
	23	JQOL	Jump if Q is odd, unconditionally circular shift (Q) right one place.
	30	JDPL	Jump if $(D) \geqq 0$
	31	JAGQFL	Jump if $(A) \geqq (Q)$, floating point sense.
	32	JAGQL	Jump if $(A) \geqq (Q)$, algebraic sense.
	33	JAGDL	Jump if $(A) \geqq (D)$, alphanumeric sense.

Jumps - Right

	00	JMPR	conditions same as for left jumps
	01	JAZR	
	02	JNOR	
	03	JOFR	
	10	JAPR	
	11	JANR	
	12	JAEQR	
	13	JAEDR	
22	20	JQPR	
	21	JQNR	
	22	JQER	
	23	JQOR	
	30	JDPR	
	31	JAGQFR	
	32	JAGQR	
	33	JAGDR	

Index - Left

	00	TDXL	$(D_{0V}) \longrightarrow X$
	01	TDXLC	$(D_{0V}) \longrightarrow X,\ (D_{0J}) \longrightarrow X_C$
	02	TXDL	$(X) \longrightarrow D_{0V}$
	03	TXDLC	$(X) \longrightarrow D_{0V},\ (X_C) \longrightarrow D_{0J}$
	10	ADXL	$(X) + (D_{0V}) \longrightarrow X$
	11	SDXL	$(X) - (D_{0V}) \longrightarrow X$
	12		CF
03	13		CF
	20		CF
	21	TIXZ	$(I_V) \longrightarrow X,\ 0 \longrightarrow X_C$
	22		CF
	23		CF
	30	AIXJ	$(X) + (I_V) \longrightarrow X$; jump to address in D_{0V}, D_{0J} if $(X) \neq (D_{0V})$
	31	SIXJ	$(X) - (I_V) \longrightarrow X$; jump to address in D_{0V}, D_{0J} if $(X) \neq (D_{0V})$
	32	AIXOL	$(X) + (I_V) \longrightarrow X$; $1 \longrightarrow 0F$ if $(X) = (D_{0V})$
	33	SIXOL	$(X) - (I_V) \longrightarrow X$; $1 \longrightarrow 0F$ if $(X) = (D_{1V})$

Index - Right

	00	TDXR	Same as TDXL, except use D_1
	01	TDXRC	Same as TDXLC, except use D_1
	02	TXDR	Same as TXDL, " " "
	03	TXDRC	Same as TXDLC, " " "
	10	ADXR	Same as ADXL, " " "
	11	SDXR	Same as SDXL, " " "
	12		CF
	13		CF
23	20		CF
	21	TIXS	Same as TIXZ, except $1 \longrightarrow X_C$
	22		CF
	23		CF
	30	AIXJ	
	31	SIXJ	
	32	AIXOR	Same as AIXOL, except use D_1
	33	SIXOR	Same as SIXOL, " " "

Addition

00	AM	$(A) + (V) \longrightarrow A$
01	AMS	$(A) + (V) \longrightarrow A, D, V$
02	CAM	$0 \longrightarrow A; (A) + (V) \longrightarrow A$
03	CAMS	$0 \longrightarrow A; (A) + (V) \longrightarrow A, D, V$
10	AMA	$(A) + V \longrightarrow A$
11	AMAS	$(A) + V \longrightarrow A, D, V$
12	CAMA	$0 \longrightarrow A; (A) + \lvert V\rvert \longrightarrow A$
13	CAMAS	$0 \longrightarrow A; (A) + \lvert V\rvert \longrightarrow A, D, V$
20	AQ	$(A) + (Q) \longrightarrow A$
21	AQS	$(A) + (Q) \longrightarrow A, D, V$
22	CAQ	$0 \longrightarrow A; (A) + (Q) \longrightarrow A$
23	CAQS	$0 \longrightarrow A; (A) + (Q) \longrightarrow A, D, V$
30	AQA	$(A) + \lvert Q\rvert \longrightarrow A$
31	AQAS	$(A) + \lvert Q\rvert \longrightarrow A, D, V$
32	CAQA	$0 \longrightarrow A; (A) + \lvert Q\rvert \longrightarrow A$
33	CAQAS	$0 \longrightarrow A; (A) + \lvert Q\rvert \longrightarrow A, D, V.$

Subtraction

	00	SM	$(A) - (V) \longrightarrow A$
	01	SMS	$(A) - (V) \longrightarrow A, D, V$
	02	CSM	$0 \longrightarrow A; (A) - (V) \longrightarrow A$
	03	CSMS	$0 \longrightarrow A; (A) - (V) \longrightarrow A, D, V$
	10	SMA	$(A) - \lvert V \rvert \longrightarrow A$
	11	SMAS	$(A) - \lvert V \rvert \longrightarrow A, D, V$
	12	CSMA	$0 \longrightarrow A; (A) - \lvert V \rvert \longrightarrow A$
	13	CSMAS	$0 \longrightarrow A; (A) - \lvert V \rvert \longrightarrow A, D, V$
11	20	SQ	$(A) - (Q) \longrightarrow A$
	21	SQS	$(A) - (Q) \longrightarrow A, D, V$
	22	CSQ	$0 \longrightarrow A; (A) - (Q) \longrightarrow A$
	23	CSQS	$0 \longrightarrow A; (A) - (Q) \longrightarrow A, D, V$
	30	SQA	$(A) - \lvert Q \rvert \longrightarrow A$
	31	SQAS	$(A) - \lvert Q \rvert \longrightarrow A, D, V$
	32	CSQA	$0 \longrightarrow A; (A) - \lvert Q \rvert \longrightarrow A$
	33	CSQAS	$0 \longrightarrow A; (A) - \lvert Q \rvert \longrightarrow A, D, V$

Multiplication

	00	MM	$(V) \times (Q) \longrightarrow A, Q$
	01	MMS	$(V) \times (Q) \longrightarrow A, Q; (A) \longrightarrow D, V$
	02	MMR	$(V) \times (Q) \longrightarrow A$
	03	MMRS	$(V) \times (Q) \longrightarrow A; (A) \longrightarrow D, V$
	10	MMA	$\lvert V \rvert \times (Q) \longrightarrow A, Q$
	11	MMAS	$\lvert V \rvert \times (Q) \longrightarrow A, Q; (A) \longrightarrow D, V$
	12	MMAR	$\lvert V \rvert \times (Q) \longrightarrow A$
	13	MMARS	$\lvert V \rvert \times (Q) \longrightarrow A; (A) \longrightarrow D, V$
12	20	MA	$(A) \times (Q) \longrightarrow A, Q$
	21	MAS	$(A) \times (Q) \longrightarrow Q, Q; (A) \longrightarrow D, V$
	22	MAR	$(A) \times (Q) \longrightarrow A$
	23	MARS	$(A) \times (Q) \longrightarrow A; (A) \longrightarrow D, V$
	30	MAA	$\lvert A \rvert \times (Q) \longrightarrow A, Q$
	31	MAAS	$\lvert A \rvert \times (Q) \longrightarrow A, Q; (A) \longrightarrow D, V$
	32	MAAR	$\lvert A \rvert \times (Q) \longrightarrow A$
	33	MAARS	$\lvert A \rvert \times (Q) \longrightarrow A; (A) \longrightarrow D, V$

Note: "$\longrightarrow$ A, Q" implies double-length product with the more significant half in A. Signs of both (A) and (Q) will be product sign,

"$\longrightarrow$ A" implies single-length, worded product. Multiplier remains in Q after instruction.

Division and Special

	00	DAQ	$(A, Q) \div (V) \longrightarrow Q$, remainder $\longrightarrow A$
	01	DAQS	$(A, Q) \div (V) \longrightarrow Q$, $(Q) \longrightarrow D, V$
	02	DA	$(A) \div (V) \longrightarrow Q$, remainder $\longrightarrow A$
	03	DAS	$(A) \div (V) \longrightarrow Q$, $(Q) \longrightarrow D, V$
	10		CF
	11		CF
	12		CF
	13		CF
	20	MAD	$(V) \times (Q) \longrightarrow D$; $(A) + (D) \longrightarrow A$
11	21	MSU	$(V) \times (Q) \longrightarrow D$; $(A) - (D) \longrightarrow A$
	22	EA	Extract and add: $(V) \cdot (Q) \longrightarrow D$; $(A) + (D) \longrightarrow A$
	23	ES	Extract and subtract: $(V) \cdot (Q) \longrightarrow D$; $(A) - (D) \longrightarrow A$
	30	AD	$(A) + (D) \longrightarrow A$
	31	SD	$(A) - (D) \longrightarrow A$
	32		CF
	33		CF

Note: "$(A, Q) \div$ " - implies double-length dividend with the more significant half in A.

"$(A) \div$ " - implies single-length dividend.

Remainder of division is always in A.
Remainder of division is always in A.

CONTROL REGISTER STATES

AI

000	Force add in multiplication.
001	Add or subtract.
010	Multiply, double length product.
011	Multiply, single length product.
100	Fixed point division, first cycle.
101	Division, all other than above.
110	Shift
111	Q jump

FI

000	Exponent comparison for add or subtract.
001	Exponent addition or subtraction (for multiply or divide).
010	Shift (D_M); arrangement of (D_M) as $(D_E) < (A_E)$.
011	Shift (A_M); arrangement of (A_M) as $(D_E) > (A_E)$.
100	Normalize, following arithmetic operation.
101	Correction, following add, subtract, or multiply, before divide.
110	Clear D, $(D_E) \ll (A_E)$.
111	Clear A, $(D_E) \gg (A_E)$.

MI

000	Memory not being used by computer.	
001	$(V) \longrightarrow PR \longrightarrow V$	
010	$(V) \longrightarrow D$	
011	$(D) \longrightarrow V$	
100	$(V) \longrightarrow D \longrightarrow V$	
101	Clear V, $(D) \longrightarrow V$	
110	Clear V, leave cleared	(read and write 0).
111	Clear V	(read 0).

PI

00	Do next instruction.
01	Transfer next instruction word from memory to **PR**.
10	Modify index register.
11	Count down repeat counter.

SYMBOLS USED IN DEFINITIONS OF NUMBERS

D_V	"Address field" portion of half-word in D.
E	Subscript. Exponent part of floating point number.
I_V	Memory Address field of the instruction.
M	Subscript. Mantissa part of floating point number.
V	The selected memory location
X	The selected index register.

A NUMBERS

Note: The lower case "a" of an A number refers to an AI setting. This "a" listing also implies the existence of a lower case "b" number which refers to a setting of AI*. (These are not listed here to conserve space.

Number	Conditions	Definition
A1	AI = 001	Add or subtract
A2a	AI = 01X	Multiplication
A3a	AI - 10X	Division
A4	AI = 110	Shift
A5a	$\overline{\text{AI = 1XX}}$	Not division, shift or Q jump
A6	AI = 110 • 0XXX	Shift A or A, Q
A7	AI = 110 • 11XX	Shift D
A8a	AI = 100	Fixed point division, first cycle
A9a	AI = 00X	Add, subtract or force add
A10a	AI = 101	Floating point division, or fixed point division not first cycle
A11	AI = 01X • |q| = 1	Multiplication, multiplier bit is 1
A12	AI = 01X • |q| = 0	" , " " " 0
A13	AI = 110 • X0XX	Shift Q or A, Q
A14	AI = 111	Q jump
A15	AI* = 000	Force add
A16	AI* = 1X1	Q jump, or floating point division or fixed point division after first cycle
A17	AI* = 110 • 0XX1	Involves SRA - (SRAQ, SRAQN, SRA, SRAN)
A18	AI* = 110 • 0XX0	Involves SLA -
A19	AI* = 110 • X0X1	Involves SR - Q
A20	AI* = 111 • XX1X	Q jump right
A21	AI* = 110 • X0X0	Involves SL - Q
A22	AI* = 111 • XX0X	Q jump left
A23	AI* = 110 • 0X1X	Involves S - A - N
A24	AI* = 010	Double length multiplication
A25	AI* = 011	Rounded multiplication
A26	AI* = 110 • X01X	Involves S - Q · N
A27	AI* = 110 • 0010	SLAQN

Number	Conditions	Definition
A28	AI* = 110 • 0011	SRAQN
A29	AI* = 110 • 1111	SRDN
A30	AI* = 110 • 11X0	SCD
A31	AI* = 110 • 0000	SLAQ
A32	AI* = 110 • 0001	SRAQ
A33	(AI* = 101 • FI* = 100 FP = 1) V (AI* = 101 • SC = SAT)	Last cycle of division
A34	AI* = 101 • FI* = 100 • FP = 1 • SC ≠ SAT	Division, not last cycle
A35	AI* = 010 • SC ≠ SAT	Double length multiplication, recycling
A36	(AI* = 01X • SC ≠ SAT) V (AI* = 110 • SC ≠ SAT)	Multiplication, recycling or shift recycling
A37	AI* = 110 • SC = SAT	Shift, last cycle
A38	AI* = 01X • SC = SAT • $\overline{\text{V17}}$	Multiplication, last cycle, Q ≠ -1
A39	AI* = 110 • 0X00	Involves SLA –
A40	AI* = 110 • X000	Involves SL – Q
A41	AI* = 110 • 10X1	Involves SRQ –

D NUMBERS

D Number	Conditions	D Number	Conditions
D1A	A2a v A3a v A4	D19	A34 v A36
D1B	A2b v A3b v A4	D20	A8b v A9b v A14 vA33 v A37 v A38
D2A	A3a v A5a v A6	D21	F4a v F8 v F9
D2B	A3b v A5b v A6	D22	F10 v F11 v F12 F13
D3	A2a v A3a v A13 v A14	D23	F3a v F14 vF15
D4	A9a v A11	D24	F1A v F15
D5A	A6 v A8a v A12	D25	F16 vF17
D5B	A34 v A39	D26	F18 vF19 vF20 vF21
D7	A2b v A3b v A13 v A16	D28	F6b vF24
D8	A9a v V8	D29	F0b vF3b vF22 v F25 vF26 vF27
D9	A9b v A33	D30	F0b vF1B vF3b vF25 v F27
D10	A2b v A8b v A17	D31	F19 v F22 vF29 vF30 vF32
D11	A18 v A34	D32	F1b vF5b vF25
D12	A2b v A8b v A19 vA20	D33	F4b vF19
D13	A10b v A21 v A22	D34	F0b vF3b vF7b
D14	A8b v A23	D35	F0b vF3b vF5b
D15	A8b v A25 v A26 vA35	D36	F5b vF34
D16	A27 v A34	D37	F30 vF31
D17	A8b v A24 v A28	D38	F22 vF33
D18	A22 v A33 A40	D39	F0b vF3b vF19

D Number	Conditions	D Number	Conditions
D40	F19 vF21	D62	I25 vI142
D41	F21 vF36	D63	I31 vI32 vI36 vI37 vI39 vI62
D43	A2a vA9a	D64	I33 vI34 vI35 vI40 vI46 vI122
D44	F4a vF5a	D65	I9 vI14 vI19 vI21 vI25
D45	F29 vF32 vF38	D66	I 15 vI29 vI42 v I 52 vI53 vI81
D46	SHIFT OVF	D67	AN2CC vI15 vI81
D47	A20 vA32 vA41	D69	I34 vI35 vI40
D48	AN1CC vA4 vA12 vA14	D70	I58 vI59
D49	F39 vF40 vF41 vF42	D71	I61 vI62 vI63
D51	I1 vI2	D72	I66-1 vI68
D52	I3 vI4 vI6 vI7 vI8	D73	I74 vI75 vI76 vI77 vI78 vI79
D53	I9 vI10 vI11	D74	I10 vI43 vI50 vI63 vI72 vI80
D54	I9 vI14 vI15	D75	I43 vI44 vI45 vI72
D55	I12 vI16 vI17	D76	I15 vI53 vI81
D56	I18 vI19	D77	J2 vJ3 vJ4 vJ5 vJ10
D57	I11 vI21 vI35	D78	J6 vJ7 vJ8 vJ9 vI81 vI85
D58	I23 vI24 vI25 vI26	D79	I12 vI17
D59	I12 vI17 vI27 vI28	D80	S1 v S2 v S3 v S4 vS5 v S6
		D81	R2#1 v P7
D60	I47 vI48 vI49 vI50 vI51	D82	I52 v I53
D61	143 v I44 vI45 v172	D83	I86 v I87

D Number	Conditions
D84	I43 v I68 vI88 I89 v I90
D85	I91 v I92 vI95 vI96 vI99 v I100
D86	I93 v I94 vI97 vI98
D87	I37 v I48 vI101 vI102 vI103
D88	I61 v I66 v I104 I62 v I63
D89	I105 v I106 v I107 vI108 v I113
D90	I109 vI110vI111vI112
D91	I2 v I34 vI114vI115
D92	I68 v I98 v I112
D93	I2 v I34 vI114 v I115 v I116 vI118
D94	I36 v I62
D95	I43 v I90 v I119 v I143
D96	I120 v I121
D97	I65 v I122
D98	I1 v I2 v I48 v I85 I101 v I122
D99	I34 v I35 v I37 v I42 I81 v I123
D100	I13 v I41 v I125 vI126
D101	I36 v I46 v I54 I84 v I127 v I139
D102	I11 #1 v I130
D103	AN1CC v D51 vD126
D104	J11 v J12 v J13 v J14 v J15

D Number	Conditions
D105	J18 v J19 v J20 v J21 v J22
D106	I29 v I42
D107	I34 v I134 v I135
D108	I37 v I60
D109	I9 v I34 v I35 v I135
D110	I61 v I135 v I136
D111	I29 v I35
D112	I137 v I138
D113	I12 v V67
D114	I54 v I122
D115	I44 v I56 v I57
D117	AN2CC v I11 vI21 v I16
D118	S7 v S8
D119	J4 v J5 v J10
D120	P11 vP14 vP15
D121	P16 vP17 vP18
D122	I81 vP21
D123	V45 vI13 vP22
D124	P9 vP24
D125	I132 vI140 vI10 vI15
D126	I34 v I35 v I81 v I60 v I123 v I79

F NUMBERS

Note Numbers with suffix "a" refer to FI settings, those with suffix "b" to FI* settings.

Number	Conditions	Definition
F0	FI = 000	Compare numbers for add, subtract.
F1	FI = 001	Exponent addition, subtraction for multiplication, division.
F2	FI = 010	Shift D_M $(D_E \leq A_E)$
F3	FI = 011	Shift A_M $(D_E > A_E)$
F4	FI = 100	Normalize result
F5	FI = 101	Correct result for overflow.
F6	FI = 110	Clear D_M $(D_E \ll A_E)$
F7	FI = 111	Clear A_M $(D_E \gg A_E)$
F8	$\text{FI} = \text{X01} \cdot \overline{\text{A10a}}$	Exponent addition or correction for multiplication.
F9	$\overline{\text{FI} = \text{X0X}}$	Shift or clear A_M or D_M
F10	$\text{FI} = 100 \cdot \overline{\text{A10a}}$	Normalize for multiplication.
F11	$\text{FI} = 101 \cdot \text{A10a}$	Correction for division.
F12	FI = 001	
F13	FI = 01X	Shift A_M or D_M
F14	$\text{FI} = 001 \cdot \text{A10a}$	Division, subtract exponents
F15	FI = 10X	Normalize or correction
F16	$\text{FI} = \text{X01} \cdot \text{A10a}$	Division, subtract exponents or correction
F17	$\text{FI} = 100 \cdot \text{A24}$	Double length multiplication, normalize
F18	$\text{FI*} = 100 \cdot \overline{\text{A10a}}$	Multiplication, normalize
F19	$\text{FI*} = 101 \cdot \text{A10a}$	Division, correction
F20	$\text{FI*} = 001 \cdot \text{A2a}$	Multiplication, add exponents.
F21	FI* = 01X	Shift D_M or A_M
F22	$\text{FI*} = \text{00X} \cdot \text{A10a}$	Division, subtract exponents
F24	$\text{FI*} = 010 \cdot \text{SC*} = 0$	Shifting D_M, recycling.
F25	$\text{FI*} = 100 \cdot \text{UF} = 0$	Normalizing, no underflow.
F26	$\text{FI*} = 001 \cdot \text{A2a} \cdot \text{V30}$	Underflow when exponents added
F27	FI* = 1X1	Correction or clear A_M

Number	Conditions	Definition
F28	$\overline{\text{FI* = 000}}$	Not exponent comparison
F29	FI* = 001 · I64 · V30	Double length multiplication, exponent addition underflows.
F30	FI* = 100 · I64 · UF = 0	Double length multiplication, normalizing does not cause underflow.
F31	FI* = 100 · A10a · UF = 0	Division, normalizing does not cause underflow.
F32	FI* = 100 · A10a · SC = SAT	Division, normalizing completed, number is zero.
F33	FI* = 001 · I64	Add exponents, double length multiplication.
F34	FI* = 001 · UF = 0	Addition or subtraction of exponents did not underflow.
F35	FI* = 100 · A10a	Division, normalizing
F36	FI* = 000 · SC = 0	Shifting of A_M completed
F37	FI* = 000 · SC $\neq$ 0	Shifting A_M, recycling
F38	FI* = 100 · I64 · SC = SAT	Double length multiplication, normalizing, number is zero.
F39	AN1ECC · $\overline{\text{V19}}$ · V23	
F40	AN1ECC · V18	
F41	MO · $\overline{\text{A10a}}$ · AN1ECC	
F42	(FOa · A10a) v (AN1ECC · A10a)	

I NUMBERS

Number	Coding	Definition
I1	010 0001	Jump if A is zero
I2	010 011X	Jump if A = Q, A = D
I3	111 01XX	
I4	111 111X	
I6	011 011X	CF (Command Fault)
I7	011 1000	
I8	011 101X	
I9	1XX	Arithmetic instruction
I10	1 001	Shift instruction
I11	011 111X	AIXO, SIXO $(X) \pm I_V \rightarrow X$; then $1 \rightarrow$ OVF if $(X) = (D_V)$.
I12	000 1011	RPT
I13	010 001X	JNO, JOF
I14	00X	Special, transfer or shift instructions.
I15	010	Jump instruction
I16	011 X00X	Transfer to X
I17	000 0110	Skip instructions SKC, SKF
I18	011 X1XX	Instructions which add or subtract (X) and I_V, then compare with D_O. AIXJ, SIXJ, AIXO, SIXO; or add or subtract (X) and (D). ADX, SDX
I19	011 XX1X	(not readily definable)
I20	011 0X0X	Index instructions where D affects X.
I21	011 0XXX	Index instructions involving D and not PR.
I23	011 00XX	Transfers between D and X.
I24	011 0100	ADX $(X) + (D_V) \rightarrow X$
I25	011 1001	TIX $(I_V, J) \rightarrow X, X_c$
I26	011 11X0	AIXJ, AIXO
I27	011 0101	SDX $(X) - (D_V) \rightarrow X$
I28	011 11X1	SIXJ, SIXO
I29	011 001X	TXD, TXDC transfers of X to D
I30	000 0010	ICO Inhibit clearing of overflow
I31	0 001 0001	TMA $(V) \rightarrow A$
I32	0 001 001X	TMQ, TMD $(V) \rightarrow$ Q or D

Number	Coding			Definition
I33	0	001	1111	CD $0 \longrightarrow D$
I34	1	000	1101	AWCS
I35		000	100X	TJM, INCA change in address field of V.
I36		000	1100	ETA, ETD
I37		000	1111	SWD, LWD
I39		1XX	0XX0	Arithmetic instruction with (V) operand and result not stored.
I40		1XX	0XX1	Arithmetic instruction with (V) operand and stored result.
I41		000	0000	HLT
I42		000	1010	TIJ
I43		000	1110	EI, EIS
I44	0	001	0100	TAM $(A) \longrightarrow V$
I45	0	001	011X	TAQ, TAD $(A) \longrightarrow Q$, $(A) \longrightarrow D$
I46	1	000	0100	TTM $(TR) \longrightarrow V$
I47	0	001	1011	TQD $(Q) \longrightarrow D$
I48		010	1101	JAGQF jump if $(A) \geq (D)$ floating point
I49		010	X110	JAEQ, JAGQ jump based on A & Q comparison
I50		10X	1XXX	Add, subtract with Q operand
I51	0	001	100X	TQM, TQA
I52		000	1001	INCA
I53		011	110X	AIXJ, SIXJ
I54	0	000	1101	DORMS
I55	0	001	0000	CM
I56	0	001	1000	TQM
I57	0	001	1100	TDM
I58		10X	1XX1	Add, subtract with Q operand and store result
I59		1X0	1XX1	Add, multiply, both operands in registers, store result.
I60		111	100X	MAD, MSU
I61		10X		Add, subtract
I62		111	101X	EA, ES
I63		111	110X	AD, SD
I64		110	XX0X	Double length multiplication.
I65		110	XX1X	Rounded multiplication
I66		111	00XX	Division

Number		Coding				Definition
I67		010	10XX			Jump conditional upon Q.
I68		110				Multiplication
I69	0					J = 0
I70	1					J = 1
I72		110	1XXX			Multiplication, operand in A
I74	0	001	1X1X			
I75	0	001	1XX1			
I76	0	001	X1X1			
I77	0	001	X11X			Does not involve memory
I78		000	0XXX			(used in IT2 $\longrightarrow$ IT3 gating)
I79		000	101X			
I80		01X				
I81		000	0001	$\overline{P23}$		JBT
I83	1	000	1010			TIJR
I84		000	0101			TCM, TDC
I85		010	0000			JMP
I86		111	001X			Division, single length
I87	0	001	XX10			Transfer to Q
I88		10X	XX1X			Add, subtract, preclear A
I89	0	001	XX01			Transfer to A
I90	1	000	1100			ETA
I91		100	X1XX	V57		Add, absolute value, and d_o = 1
I92		101	X1XX	V53		Subtract, absolute value, and d_o = 0
I93		110	X1XX	V53	V50	Multiply, absolute value, d_o = 0 and q_o = 1
I94		110	X1XX	V57	V49	Multiply absolute value, d_o = 1 and q_o = 0
I95		111	00XX	V53	V47	Divide, d_o = 0 and a_o = 0
I96		111	00XX	V57	V48	Divide, d_o = 1 and a_o = 1
I97		110	X0XX	V50		Multiply, algebraic value, q_o = 1
I98		111	100X	V50	V33	Multiply cycle of MAD, MSU and q_o = 1
I99		111	1001	V32		Subtract cycle of MSU
I100		101	X0XX			Subtract algebraic value
I101		010	111X			JAGQ, JAGD
I102		111	1011			ES
I103		111	1101			SD
I104		111	100X	V32		Add, subtract cycle of MAD, MSU
I105		100	X1XX	V53		Add, absolute value, d_o = 0
I106		101	X1XX	V57		Subtract, absolute value, d_o = 1

Number		Coding				Definition
I107		111	00XX	· V53	· V48	Divide, $d_o = 0$ and $a_o = 1$
I108		111	00XX	· V57	· V47	Divide, $d_o = 1$ and $a_o = 0$
I109		110	X0XX	· V49		Multiply, algebraic value, $q_o = 0$
I110		110	X1XX	· V49	· V53	Multiply, algebraic value, $q_o = 0$ and $d_o = 0$
I111		110	X1XX	· V57	· V50	Multiply, algebraic value, $q_o = 1$ and $d_o = 1$
I112		111	100X	· V33	· V49	Multiply cycle of MAD, MSU and $q_o = 0$
I113		100	X0XX			Add, algebraic value
I114		111	1000	· V32		Add cycle of MAD
I115		111	1010			EA
I116		111	1100			AD
I118	1	000	1111			SWD
I119	0	001	X001			TMA, TQA
I120	0	001	1110			TDQ
I121	0	001	0X10			TMQ, TAQ
I122		111	10XX	· V33		Multiply cycle of MAD
I123		011				Index register instruction
I124		001				Transfer or shift instructions
I125		010	010X			Jump conditional upon sign of A
I126		010	1100			JDP
I127		000	0X1X			Not readily definable used for IT4 $\longrightarrow$ END
I128	0	000	0100			TIO
I129		011	11XX			Index register instructions with conditional jump or overflow test
I130		011	XX0X			Index register instructions that do not change (X).
I131		011	0001			TDXC
I132		111	1XXX			Special arithmetic instructions
I134		1XX	XXX1			Arithmetic instruction, store result
I135	1	000	1110			EIS
I136		1X0				Add or Multiply
I137		000	1000			TJM
I138		011	0011			TXDC
I139		000	0001	P23		JBT and Breakpoint switch to Ignore.
I140			XXX0	I9		Arithmetic instructions not storing result.
I141		111	1001			MSU
I142		000	0111			TCX
I143	0	001	1101			TDA
I144	0	000	0110			SKC
I145	1	000	0110			SKF

J NUMBERS

Number	Conditions	Definitions
J2	010 0010 · V45	JNO · OVF = 0
J3	010 0011 · V46	JOF · OVF = 1
J4	010 0100 · V47	JAP · $a_o = 0$
J5	010 0101 · V48	JAN · $a_o = 1$
J6	010 1000 · V49	JQP · $q_o = 0$
J7	010 1001 · V50	JQN · $q_o = 1$
J8	010 1010 · V51	JQE · $q_{-47} = 0$
J9	010 1011 · V52	JQO · $q_{-47} = 1$
J10	010 1100 · V53	JDP · $d_o = 0$
J11	010 0001 · AN1CC	JAZ · (A) = 0
J12	010 011X · AN1CC	Jump, A is equal to Q or D
J13	010 1111 · V65	JAGD · $A \geq D$ alphanumeric sense
J14	010 1110 · $\underline{V60}$	JAGQ · $A \geq Q$ algebraic sense
J15	010 1110 · $\overline{V61}$ · V63	
J18	010 1101 · V19 · V47 · V53	JAGQF · $A > Q$ $+A_M$ $+A_E$ $+Q_M$ $+Q_E$
J19	010 1101 · V18 · V48 · V57	JAGQF · $A > Q$ $-A_M$ $-A_E$ $-Q_M$ $+Q_E$
J20	010 1101 · V47 · V57	JAGQF · $A > Q$ $+A_M$ $-Q_M$
J21	010 1101 · V18 · V22 · V47 · V53	" · $A \geq Q$ $+A_M$ $+A_E$ $+Q_M$ $+Q_E$
		or $+A_M$ $-A_E$ $+Q_M$ $-Q_E$
J22	010 1101 · V23 · V48 · V57	" " $-A_M$ $+A_E$ $-Q_M$ $+Q_E$
		or $-A_M$ $-A_E$ $-Q_M$ $+Q_E$
		or $-A_M$ $-A_E$ $-Q_M$ $-Q_E$

M AND DM NUMBERS

Number	Conditions	Definition
DM1	MI = 010 v 100	$(V) \longrightarrow D$
DM2	(MI = 001 v 100 v 101 v 110) v M11	Write immediately after read
DM3	MI = 010 v M13	Involves $(V) \longrightarrow D$ operation
DM4	MI = 011 v 100 v 101	Involves write with $(D) \longrightarrow V$
DM5	MI = 100 v 101 v 110	
DM6	MI = 010 v 111	Read only operation
MO	MI = 000	Memory not in use or requested by computer.
M1	MI = 001 • M10	Computer assigned memory for $(V) \rightarrow PR \rightarrow V$
M2	MI = 010 • M10	" " " " $(V) \rightarrow D$
M3	MI = 011 • M10	" " " " $(D) \rightarrow V$
M4	MI = 100 • M10	" " " " $(V) \rightarrow D \rightarrow V$
M6	MI = 110 • M10	" " " " $0 \rightarrow V$
M8	IOMI = 0 (From Core)	
M10		Memory is assigned to computer
M11	$\overline{M10}$	
M12	MA = MP	V = Memory Preset
M13	M4 • I9	Arithmetic instruction, memory operand and result not stored
M14	IOMI = 0 • Drum Action	

P NUMBERS

Number	Conditions	Definition
P0	PI = 00	Perform the next instruction.
P1	PI = 01	Read a new instruction word.
P2	PI = 10	Modify an index register.
P3	PI = 11	Count down repeat counter.
P4	$\overline{R = 1}$	R bit = 0
P5	R = 1	
P6	$X_c = 0 \cdot \overline{P4}$	Counter bit = 0 and R bit = 1
P7	$X_c = 1 \cdot \overline{P4}$	
P8	PI = X1	
P9	PI = 1X	
P11	PI* = 01	
P12	PI* = 10	
P13	PI* = 11	
P14	PI* = 10 · R6 · V59 · V55	After modifying X, do I_1, already in PR.
P15	P13 · V41	Continue repeat mode, N $\neq$ 0.
P16	PI* = 10 · V67	Jump after modifying X
P17	PI* = 10 · R0 · V54	Modifying X, then get next instruction word in normal sequence.
P18	P13 · V56	End repeat mode, N = 0.
P20	$\overline{MF} \cdot \overline{STOP} \cdot \overline{EF} \cdot \overline{CF} \cdot$ RUN MODE	Run
P21		Breakpoint switch OFF
P22		OVF switch OFF
P23		Breakpoint switch to IGNORE
P24	P1 · P20	
P25	PI · [R1 v (JFF = 1) v (INT FF = 0)]	
P26	PI · R0 · (JFF = 0) · (INT FF = 1)	

(bit positions for 8-index register system)

	(value of) R	IA_2	IA_1	IA_0	V_{12}
P27	0	0	1	0	0
P28	0	0	0	0	1
P29	0	0	0	1	0
P30	0	0	0	1	1

P31	Skip INT FF = 1
P32	P11 · R0 · (JFF = 0) · (INT FF = 1)

R NUMBERS

Number	Conditions	Definitions
R0	RPT bit = 0	
R1	RPT bit = 1	
R2	$\overline{R3}$	$\alpha \gamma = 0$
R3	$(\alpha = 1 \cdot SW2 = 0) \vee (\gamma = 1 \cdot SW2 = 1)$	Instruction being repeated is to be repeat modified.
R4	$(\beta = 0 \cdot SW2 = 0) \vee (\delta = 0 \cdot SW2 = 1)$	Repeat modification is add.
R5	$(\beta = 1 \cdot SW2 = 0) \vee (\delta = 1 \cdot SW2 = 1)$	Repeat modification is subtract.
R6	I bit = 0	Perform I_o after I_1 in repeat mode.
R7	I bit = 1	Perform I_1 after I_1 in repeat mode.
R8	$R1 \cdot R2$	In repeat mode, no repeat modification.
R9	$R3 \cdot R4$	$\alpha \beta \vee \gamma \delta = 10$

V NUMBERS

Number	Conditions	
V1	$a_o \neq s_o$	
V2	$a_o = s_o$	
V3	$ANIC_o \neq ANIC_{-1}$	
V4	$ANIC_o = ANIC$	
V5	$S_o = S_{-1}$	
V6	$d^*_o \neq S_o$	
V7	$d^*_o = S_o$	
V8	$SC = SAT$	
V9	$q^*_{-47} = 0$	
V10	$q^*_{-35} = 0$	
V11	$q^*_{-1} = q^*_{-2}$	
V12	a^*_{+1} ~~d^*_o~~ $= a^*_o$	
V13	$QC = 0$	
V14	$q^*_{-35} = 1$	
V15	$a^*_o = a^*_{-1}$	
V16	$q^*_{-47} = 1$	
V17	$(QC = 1) \cdot (q^*_o = 1)$	
V18	$(ANIC_{-36} = 1) \cdot (ANIC_{-37} = 0)$	
V19	$(ANIC_{-36} = 0) \cdot (ANIC_{-37} = 1)$	
V20	$ANIC_{-36} = ANIC_{-37}$	
V21		now tied to ground
V22	$S_{-36} = 0$	
V23	$S_{-36} = 1$	
V24	$\lvert S \rvert < 35$	
V25	$\lvert S \rvert > 35$	
V26	$EO = 0$	
V27	$EO = 1$	
V28	$SC = SAT$	
V29	$UF = 0$	
V30	$UF = 1$	
V31	$a^*_o = a^*_{-2}$	
V32	$II = 1$	
V33	$II = 0$	

Number	Conditions
V40	ICOF = 0
V41	$AN2C_{11} = 0$
V42	SW2 = 1
V43	SW2 = 0
V45	OVF = 0
V46	OVF = 1
V47	$a_o = 0$
V48	$a_o = 1$
V49	$q_o = 0$
V50	$q_o = 1$
V51	$q_{-47} = 0$
V52	$q_{-47} = 1$
V53	$d_o = 0$
V54	MOD 2 = 0
V55	MOD 2 = 1
V56	$AN2C_{11} = 1$
V57	$d_o = 1$
V58	SC = 0
V59	JFF = 0
V60	$(AN1C_o = 0) \cdot (AN1C_{-1} = 1)$
V61	$(AN1C_o = 1) \cdot (AN1C_{-1} = 0)$
V62	$S_o = 1$
V63	$S_o = 0$
V64	$AN1C_o = 0$
V65	$AN1C_o = 1$
V66	AN2CC
V67	JFF = 1
V69	$d_{-24} = 0$
V70	$AN2C_{10} = 0$
V71	$AN2C_{10} = 1$
V72	PT Interlock FF = 0
V73	PT Counter FF = 0
V74	Card Interlock FF = 0
V75	PT Fault FF = 1
V76	Card Fault FF = 1

I/O COMMANDS

D REGISTER

STARTING ADDRESS (0–11)	STARTING ADDRESS (12–15)	UNIT ADDRESS (16–19)	UNIT ADDRESS (20–23)	BUFFER CHANNEL (24–27)	AMOUNT OF INFORMATION TO BE TRANSMITTED (28–35)	AMOUNT OF INFORMATION TO BE TRANSMITTED (36–39)	COMMAND FROM (40–43)	COMMAND TO (44–47)		
—	—	—	WHICH DEVICE	—	—	—	0000	0001	REAL TIME DEVICE → CORE	
— BAND NO. DRUM STARTING ADDRESS	DRUM STARTING ADDRESS	—	WHICH DRUM	—	NO. OF WORDS	NO. OF WORDS	0001	0010	CORE → DRUM	
— BAND NO. DRUM STARTING ADDRESS	DRUM STARTING ADDRESS	—	WHICH DRUM	—	NO. OF WORDS	NO. OF WORDS	0010	0001	DRUM → CORE	
—	—	—	—	—	NO. OF WORDS	NO. OF WORDS	0001	0100	CORE → PAPERTAPE PUNCH [1]	
—	—	—	—	—	NO. OF WORDS	NO. OF WORDS	0100	0001	PAPER TAPE READER → CORE [1]	
—	—	—	—	—	NO. OF WORDS	NO. OF WORDS	0001	0110	CORE → HIGH SPEED PRINTER [1]	
—	—	—	WHICH BUFFER	WHICH I/O	NO. OF CARDS [3]	WORDS PER CARD [3]	0001	0111	CORE → I/O DEVICE [2]	
—	—	—	WHICH BUFFER	WHICH I/O	NO. OF CARDS [3]	WORDS PER CARD [3]	0111	0011	I/O DEVICE → BUFFER [2]	
—	—	—	WHICH BUFFER	—	—	—	0011	0001	BUFFER → CORE	
—	NO. OF BLOCKS TO BE SPACED	—	WHICH TAPE	—	—	NO. OF BLOCKS	0001	1001	CORE → MAG. TAPE	MODE 1 — FOWARD
—	NO. OF BLOCKS TO BE SPACED	—	WHICH TAPE	—	—	NO. OF BLOCKS	1001	0001	MAG. TAPE → CORE	MODE 1 — FOWARD
—	NO. OF BLOCKS TO BE SPACED	—	WHICH TAPE	—	—	NO. OF BLOCKS	0001	1010	CORE → MAG. TAPE	MODE 2 — FOWARD
—	NO. OF BLOCKS TO BE SPACED	—	WHICH TAPE	—	—	NO. OF BLOCKS	1010	0001	MAG. TAPE → CORE	MODE 2 — FOWARD
—	NO. OF BLOCKS TO BE SPACED	—	WHICH TAPE	—	—	NO. OF BLOCKS	0001	1011	CORE → MAG. TAPE	MODE 3 — FOWARD
—	NO. OF BLOCKS TO BE SPACED	—	WHICH TAPE	—	—	NO. OF BLOCKS	1011	0001	MAG. TAPE → CORE	MODE 3 — FOWARD
—	NO. OF BLOCKS TO BE SPACED	—	WHICH TAPE	—	—	NO. OF BLOCKS	1101	0001	MAG. TAPE → CORE – MODE 1	REVERSE
—	NO. OF BLOCKS TO BE SPACED	—	WHICH TAPE	—	—	NO. OF BLOCKS	1110	0001	MAG. TAPE → CORE – MODE 2	REVERSE
—	NO. OF BLOCKS TO BE SPACED	—	WHICH TAPE	—	—	NO. OF BLOCKS	1111	0001	MAG. TAPE → CORE – MODE 3	REVERSE
—	—	—	WHICH TAPE	—	—	—	1000	1000	CONTINUE	MAG. TAPE
—	—	—	WHICH TAPE	—	—	—	1111	1000	STOP	MAG. TAPE
—	—	—	WHICH TAPE	—	—	—	1000	1001	RESUME	MAG. TAPE
—	—	—	WHICH TAPE	—	—	—	1000	1010	REWIND	MAG. TAPE
—	—	—	WHICH TAPE	—	—	—	1000	1011	REWIND W/LOCKOUT	MAG. TAPE
—	—	—	WHICH TAPE	—	—	—	1100	1100	RELEASE	MAG. TAPE
—	—	—	WHICH TAPE	—	—	—	1100	1101	−1 READ	MAG. TAPE
—	—	—	WHICH TAPE	—	—	—	1100	1110	ERASE	MAG. TAPE
—	—	—	WHICH TAPE	—	—	—	1100	1111	EDIT	MAG. TAPE

FUNDAMENTAL CODES

0000 REAL TIME DEVICES	0100 PAPER TAPE [1]
0001 MAGNETIC CORE	0101
0010 MAGNETIC DRUM	0110 HIGH SPEED PRINTER [1]
0011 BUFFER-CONTROLLER	0111 I/O DEVICE [2]

1000 THRU 1111 INCLUSIVE ARE USED FOR MAG. TAPE

NOTES

1 - CODES 0100 AND 0110 ARE USED IN INSTALLATIONS WHERE THESE DEVICES DO NOT OPERATE THROUGH A BUFFER - CONTROLLER.

2 - CODE 0111 IS USED FOR ANY DEVICE OPERATING THROUGH A BUFFER - CONTROLLER.

3 - THESE BITS ARE USED ONLY WHEN THE I/O DEVICE SPECIFIED IS A PUNCHED CARD SYSTEM

4 - UNUSED FIELDS ARE INDICATED BY A LINE IN PLACE OF A LEGEND.

TABLE OF CONTENTS

PHILCO TRANSAC S-2000 SYMBOLISM

PREFACE

This report has been written to enable debuggers, field engineers, technicians, and others who use Transac S-2000 logical diagrams to understand the functions and operations performed by the circuits represented on such diagrams.

One aim in its preparation has been to present sufficient illustrative examples to provide the reader with an understanding of the logic and at the same time to avoid superfluity. Hence, no attempt has been made to present every conceivable combination and application. On the other hand, the rules have been stated in general terms and a limited number of examples of more complicated combinations have been provided. Neither logic design rules nor values of circuit parameters have been specified, since these data are given in the applicable engineering drawings and specifications.

PHILCO TRANSAC S-2000 SYMBOLISM

CHAPTER 1. INTRODUCTION

1.0 A logical diagram comprises symbols for logical and storage elements and interconnecting lines and their labels. A line may be thought of as carrying a signal from the output of one logical element to the input of another.

1.1 Logical Operations

The simplest logical operation is that of negation or contradiction. For example, consider the sentence "Today is Wednesday." Its negation or contradiction is "Today is not Wednesday" or "It is false that today is Wednesday." If we let the letter "W" stand for "Today is Wednesday," and use the prime symbol (') stand for "It is false that", then W' is the negation of W. W' means "Today is not Wednesday." W' is read "not W". For any statement P, P' is true if and only if P is false.

A second useful operation called "inclusive or" is symbolized "v". If W means "Today is Wednesday," and R means "It is raining," then WvR means "Today is Wednesday or it is raining (or both)". For any two statements P and Q, PvQ is true if (1) P is true or (2) Q is true or (3) P is true and Q is true. This operation is called "inclusive-or" because the case when P is true and Q is true is included in the cases when PvQ is true.

The third common logical operation is called "AND". Defining "R" and "W" as before, the sentence "It is raining and today is Wednesday" may be abbreviated RW or R·W (read "R and W"). For any 2 statements P and Q, PQ (or P·Q) is true if and only if P is true and Q is true.

We can now see that PvQ is true if PQ or PQ' or P'Q is true. If the term PQ were omitted from the above expression, the remainder (PQ' or P'Q) would express an exclusive OR. (PQ is excluded.) The symbol for an exclusive OR is "$\wedge$" and $P\wedge Q = PQ'vP'Q$. Defining "R" and "W" as before, $R\wedge W$ means "either it is raining or today is Wednesday (but not both)."

1.2 Lines

If the logical design requires an indication that $x = 1$, then a line is used with the label "$x = 1$". The line has two possible logical states (active and normal) associated with the two possible truth values (true and false) of the label.

The active state (of the line) prevails if and only if the label is true (or the condition denoted by the label exists.) The normal state prevails at all other times.

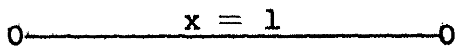
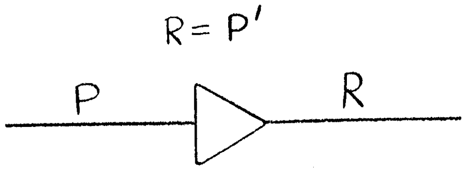


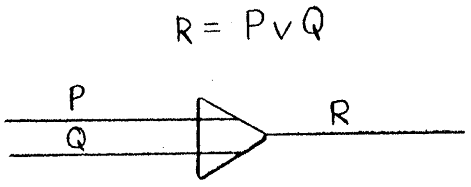
Fig. 1.1 Logical line and label

1.3 Logical Elements

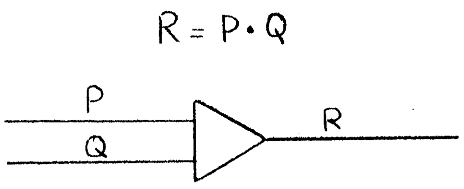
When we desire to express logical relations by the use of diagrams instead of equations, we employ graphic symbols for the logical operations and connect these symbols by logical lines. In functional diagrams (which indicate what operations are performed rather than the means by which they are performed) the basic logical element symbol is a triangle. The symbols used are those for NOT, AND, and INCLUSIVE-OR.



(a) NOT



(b) INCLUSIVE-OR



(c) AND

Fig. 1.2 Symbols and Equations for Logical Elements

1.4 De Morgan's Rules

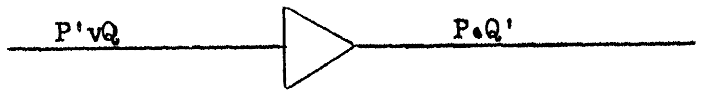
In logic as in English two negatives make a positive, i.e., $(X')' = X$. Furthermore, it is possible to express OR functions in terms of AND and NOT and to express AND functions in terms of OR and NOT.

For example:

$$XvY = (X' \cdot Y')'$$
or
$$(XvY)' = X' \cdot Y'$$

and $$P \cdot Q = (P'vQ')'$$
or
$$(P.Q)' = P'vQ'$$

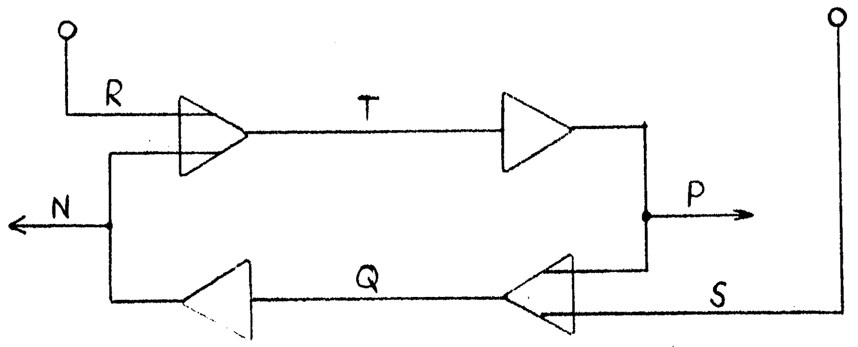
These equations are known as De Morgan's rules and are much used in logical design. For example, if a signal representing $P'vQ$ is available and $P \cdot Q'$ is desired, it may be obtained by negation as follows:



Explanation: The output of the NOT element is $(P'vQ)'$; $(P'vQ)' \equiv P'' \cdot Q' \equiv P \cdot Q'$.

1.5 Storage Element

The logical elements shown thus far have no power to store information; their outputs at any instant are determined solely by their inputs at that instant. It is possible however to combine OR and NOT elements in a way which will provide a storage operation.



$N = Q'$
$P = T'$
$Q = PvS$
$T = NvR$

Fig. 1.3 Storage Element comprising Logical Elements.

The Theory of Operation for the storage element:

I. Assume line R active & S normal:

If R is active, then T is active

If T is active, then P is normal

If P is normal and S is normal, then Q is normal.

If Q is normal, then N is active. Therefore, when R is active and S is normal, N is active and P is normal. By similar reasoning, if R is normal and S is active, then N is normal and P is active.

II. Assume both R and S are active.

Since R is active, T is active and P is normal. Since S is active, Q is active and N is normal. Therefore, if R and S are both active, then N and P are both normal.

III. We have found that R active with S normal makes N active. However, once N becomes active the active state of R is no longer necessary to maintain T active. Therefore, if R and S are normal and N is active, then N will remain active until S becomes active. Conversely, if R and S are normal and P is active, then P will remain active until R becomes active.

The storage element is so useful that it has been assigned a particular symbol.

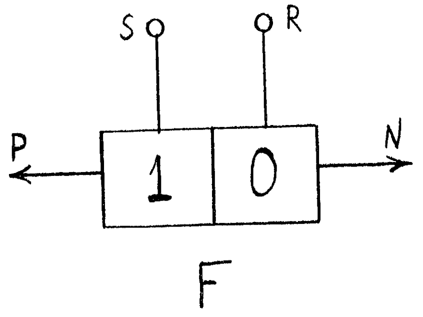


Fig. 1.4 Symbol for storage Element F.

The storage element symbol includes the digits for zero and one. They designate the state of the storage element. If N is active, we say the storage element contains a zero or (F) = 0. If P is active, we say the storage element contains a one, or (F) = 1. If R is active, we say that a zero is being stored or $0 \rightarrow F$. If S is active we say a one is being stored or $1 \rightarrow F$.

1.6 Wires

In S-2000 the logical elements are transistors and their interconnections are electrically conductive paths. For convenience we shall consider "wire" a synonym for "electrically conductive path."

The state of a wire is determined by its electrical potential (voltage). The S-2000 transistors discriminate between two potentials which are separated by a band of ambiguity. If the potential of a wire is more positive than the band of ambiguity, we say the state of the wire is <u>relatively positive</u>, or simply <u>positive</u>. If the potential of a wire is more negative than the band of ambiguity, we say that the wire is <u>relatively negative</u> or simply <u>negative</u>.

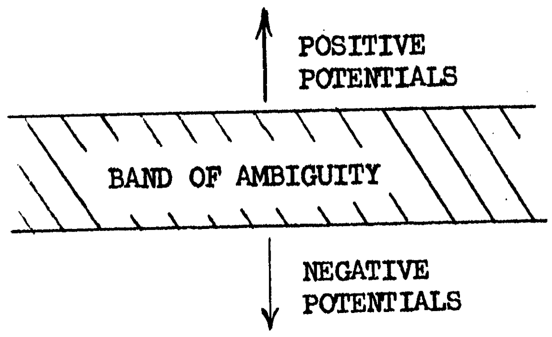


Fig. 1.5 Positive, Negative and Ambiguous Potentials

Since a wire may have two states (potentials), it may be used to convey information about the existence of a condition or the happening of some event. For example, a wire might be negative if and only if a specified register contained a negative number and positive otherwise.

However, given a specified condition (say $x = 1$), it is insufficient to say that the electrical state of the associated wire affirms or denies the existence of the condition $x = 1$.. It is also necessary to know if the positive state indicates $x = 1$; or vice versa.

This is accomplished by employing a separate form of line to represent the wire in each case.

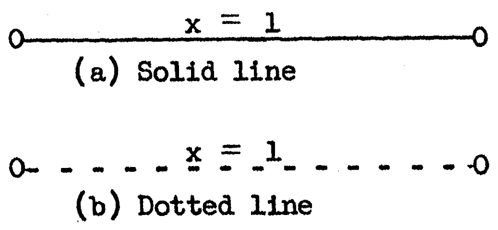


Fig. 1.6 Logical lines representing wires

To specify the relation between states of lines and states of the corresponding wires, we have the following convention:

I. The wire corresponding to a solid line is -

(A.) negative if and only if the line is active, and
(B.) positive if and only if the line is normal.

II. The wire corresponding to a dotted line is -

(A.) positive if and only if the line is active, and
(B.) negative if and only if the line is normal.

In summary, a wire may be positive or negative; a line may be active or normal; a label may be true or false or a label may denote a condition which is existent or non-existent; and a line may be solid or dotted. The form of line determines the relation between the state of the line and the state of the corresponding wire.

If we let "X" stand for "Condition x exists," and say X' means it is false that X, we can use X or X' as the label of a line.

For example:

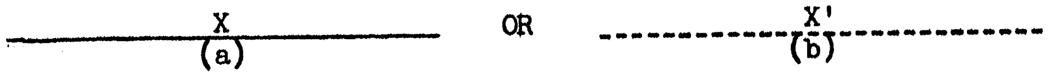


Fig. 1.7 Lines and Labels

Either (a) or (b) may be used to represent a wire which is negative when condition x exists.

The explanation is:

The solid line is active if X, and represents a wire which is then negative. It is normal when X', and its wire is then positive. The dotted line is active when X', and its wire is then positive. It is normal when X, and its wire is then negative. Hence either line (properly labeled) represents a wire which is negative when condition x exists.

Therefore the same wire may be represented by two line segments which differ in form and label; thus

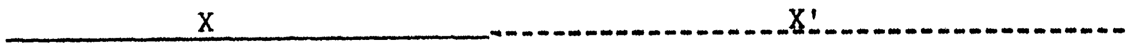


Fig. 1.8 Biform line

Where biform lines are used, the label applies only to the segment on which it appears. If the dotted segment bears no label, the proper label is the logical negative (contradiction) of the label born by the solid segment and vice versa. It must be understood that this rule applies only to biform lines. It does not apply to distinct lines separated by a logical element.

CHAPTER 2. GENERAL

A logical diagram serves as a functional and material specification of a circuit. A schematic diagram (employing conventional symbolism) serves well as a material specification for the circuit which it represents. However, the host of symbols used in such diagrams tends to obscure the functional relations between the circuit elements. Therefore, logical symbolism is used to reduce the number of symbols used and to emphasize the functional properties of the circuit.

Where no logical symbol has been adopted, a conventional symbol is used.

The majority of the logical symbols represent transistors and transistor circuits.

Transistor Logic may be classified as:

DCTL	-	Direct Coupled Transistor Logic
EFTL	-	Emitter Follower Transistor Logic
RCTL	-	Resistance Coupled Transistor Logic
		Composite Transistor Logic (A combination of 2 or more of the preceding)

Figures 1 thru 9 illustrate conventional and logical symbols for various types of transistors and transistor circuits. No attempt is made at this point to explain the operation or application of these circuits. They will be explained in the chapters on transistor logic.

Transistor circuits may be variously classified. The bases of classification to be used here are:

(1) Characteristic
- (a) PNP
- (b) NPN

(2) Configuration
- (a) Ungrounded
- (b) Common Emitter
- (c) Emitter Follower

(3) Coupling
- (a) Direct
- (b) Resistance
- (c) Capacitance
- (d) Composite (Resistance Coupled Configuration in cascade combination with a direct coupled emitter follower.)

(4) Combination
  (a) Cascade
  (b) Front-to-back
  (c) Parallel
  (d) Series (Cascode)
  (e) Tree

Conventional and logical symbols for PNP and NPN transistors are shown in Fig. 2.1. The semi-circle is the basic logical symbol for a transistor. It is used with modifications to represent various characteristics, configurations, combinations, and forms of coupling.

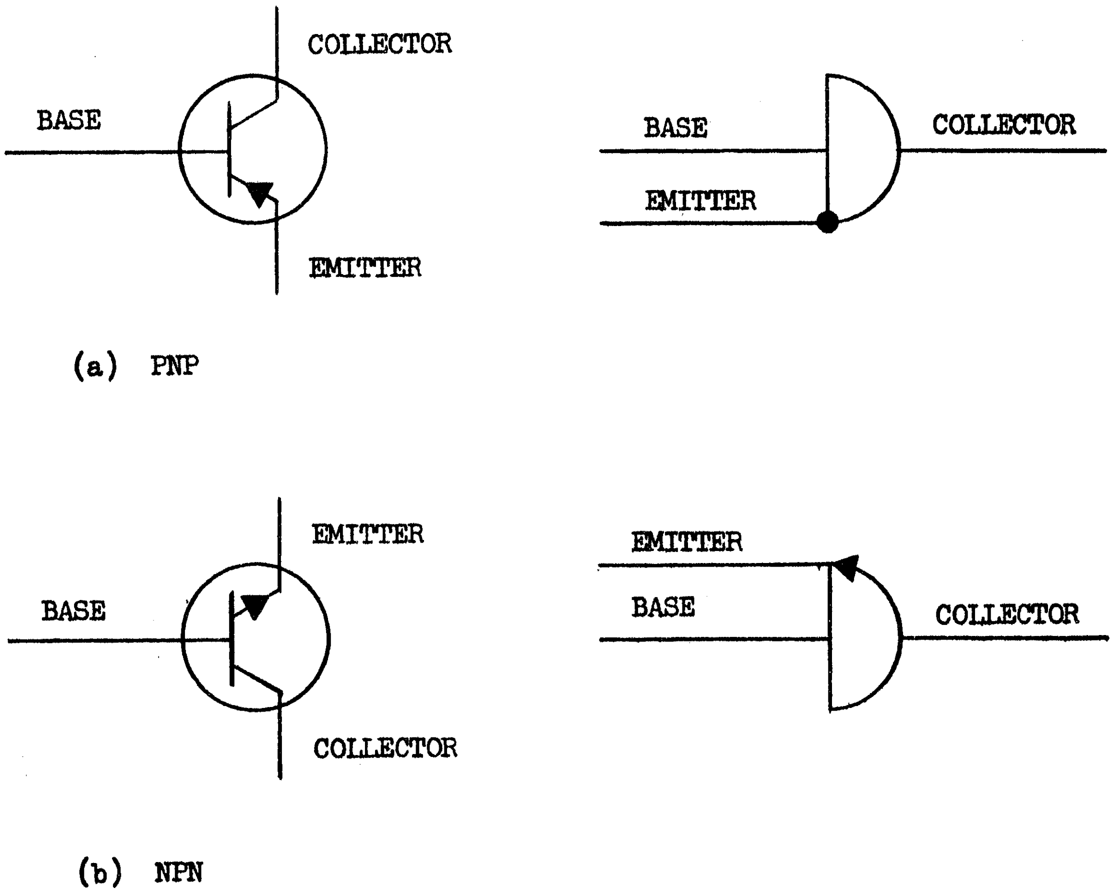
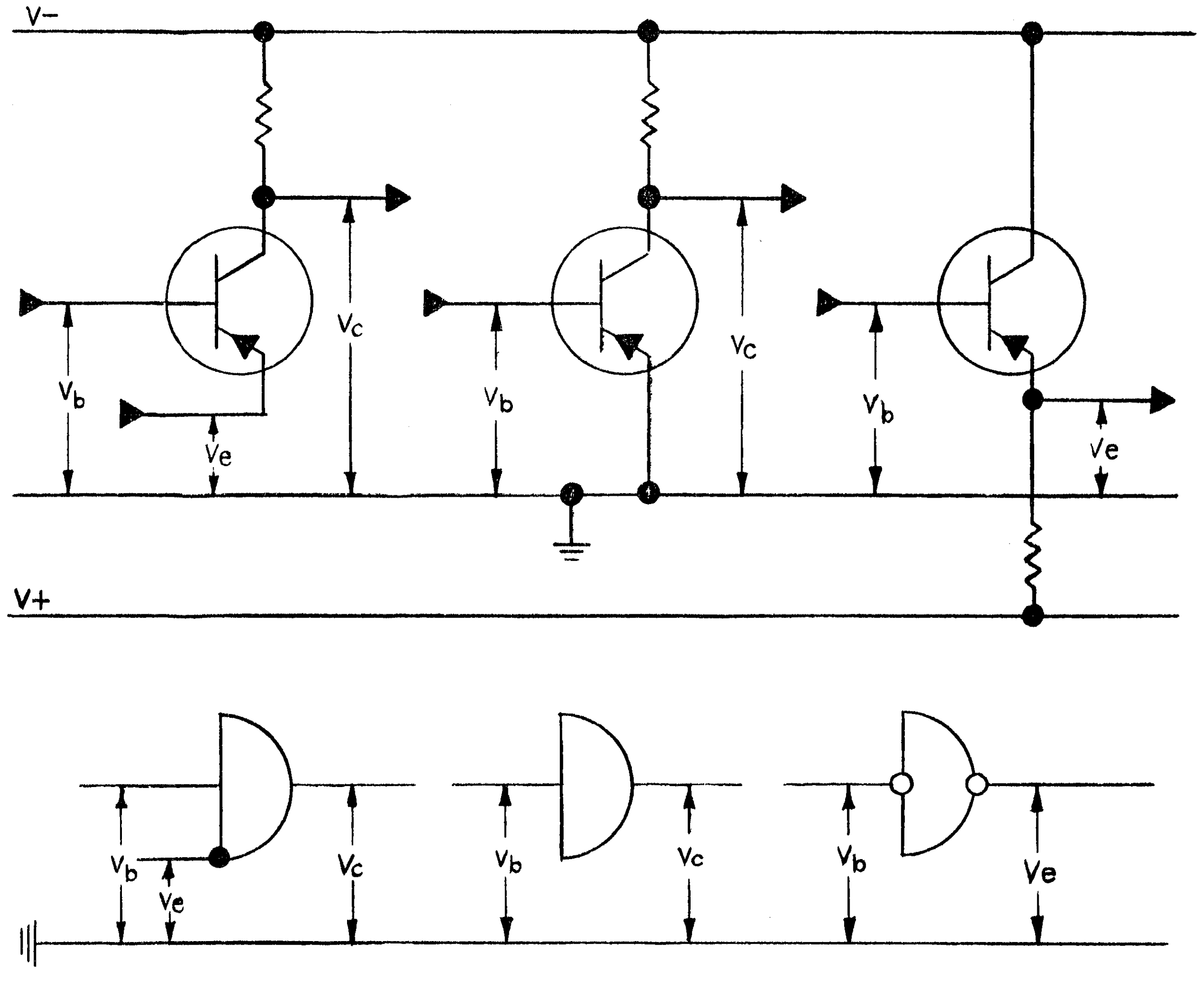


Fig. 2.1 Symbols for Transistors

The ungrounded configuration may be recognized by the presence of a dot and interconnecting line for the emitter. An emitter follower configuration is distinguished by adding 1 or more small circles to the basic symbol. Direct coupled transistors are represented by the symbol appropriate to the configuration without any distinguishing marks for the form of coupling. Conventional and logical representations for direct coupled transistors appear in Fig. 2.2



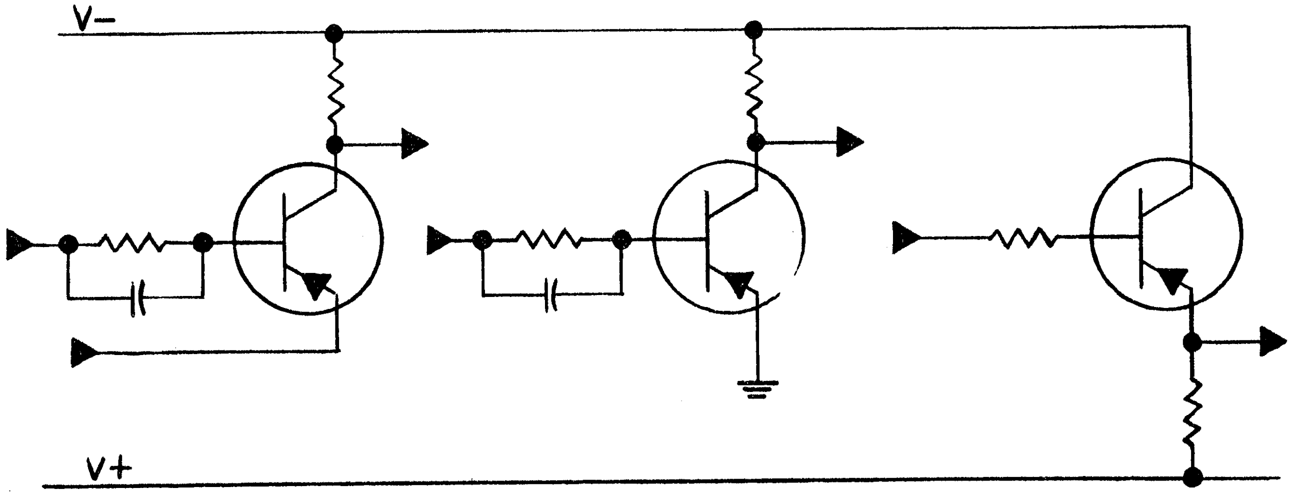
(a) UNGROUNDED CONFIGURATION

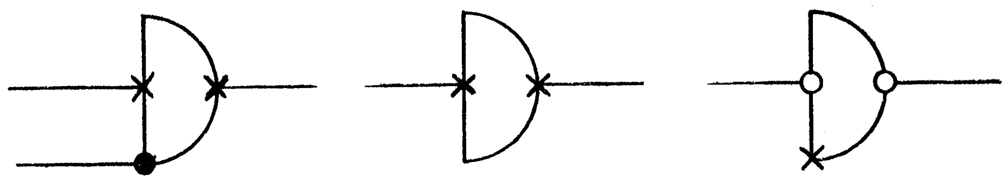
(b) COMMON EMITTER CONFIGURATION

(c) EMITTER FOLLOWER CONFIGURATION

Fig. 2.2 Direct Coupling Examples

Resistance coupling is indicated by addition of one or more oblique crosses to the basic symbol as illustrated in Fig. 2.3. The capacitors employed in the ungrounded and common emitter configurations serve a subsidiary function and their presence is not considered in determining the form of coupling.





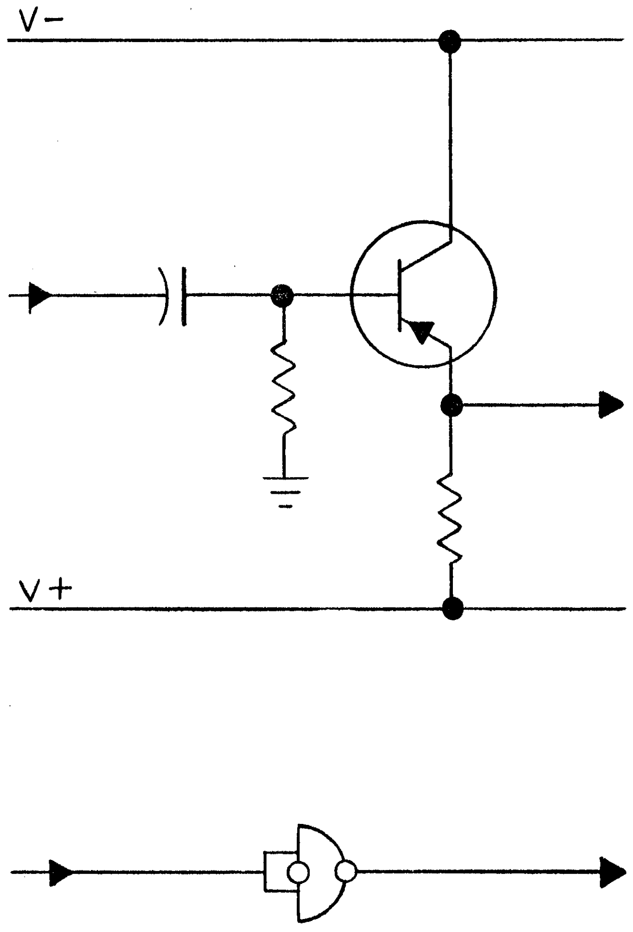
(a) UNGROUNDED CONFIGURATION

(b) COMMON EMITTER CONFIGURATION

(c) EMITTER FOLLOWER CONFIGURATION

Fig. 2.3 Resistance Coupling Examples

Capacitance Coupling to an emitter follower configuration is illustrated in Fig. 2.4.



EMITTER FOLLOWER CONFIGURATION

Fig. 2.4 Capacitance Coupling Example

The simplest combination of 2 transistors is the cascade combination shown in Fig. 2.5. The diagram shows an RC (Resistance Coupled, Common Emitter) transistor feeding its output to an emitter follower. Fig. 5 also illustrates the use of condensed logical symbolism.

The cross-and-circle symbol is an abbreviation for the cascade combination of the double-cross and double-circle symbols used in expanded logical symbolism.

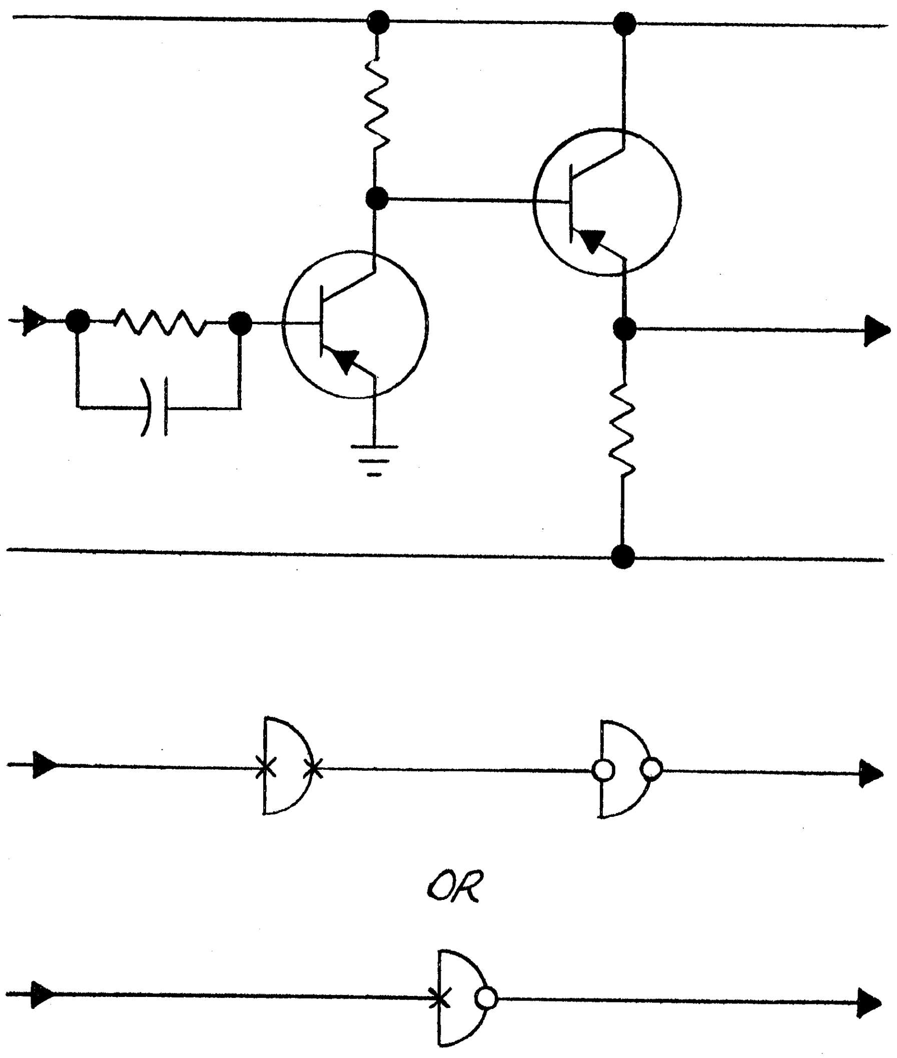


Fig. 2.5 Cascade Combination

Fig. 2.6 shows an example of the front-to-back combination of two DC (direct-coupled, common emitter) transistors. This particular combination forms a flip-flop, which in condensed logical symbolism is represented by the double square symbol.

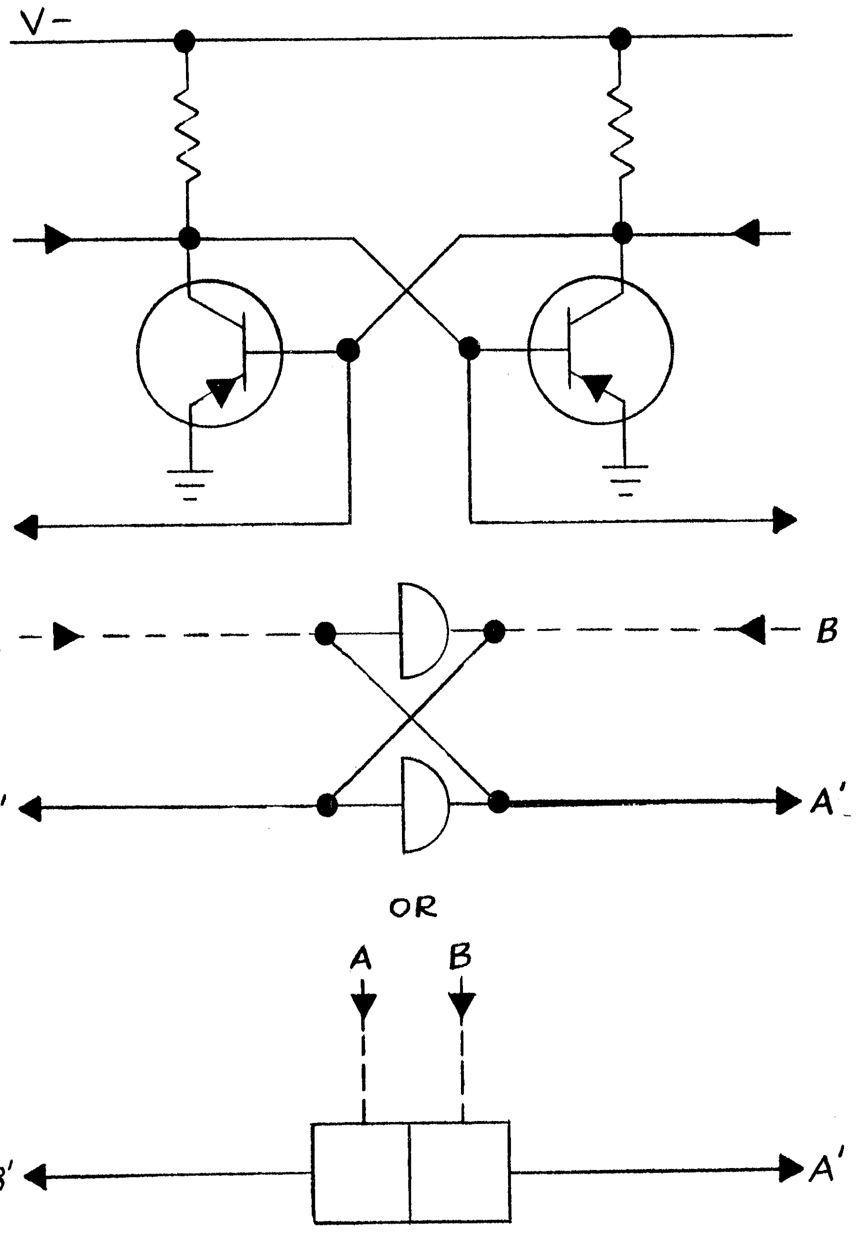


Fig. 2.6 Front-to-back Combination

Fig. 2.7 illustrates the parallel combination of DC and emitter follower transistors. Note that the condensed symbols for parallel combinations are distinguished by carrying the input lines thru the diameter of the semi-circle. The combinations shown are called 3-wide gates.

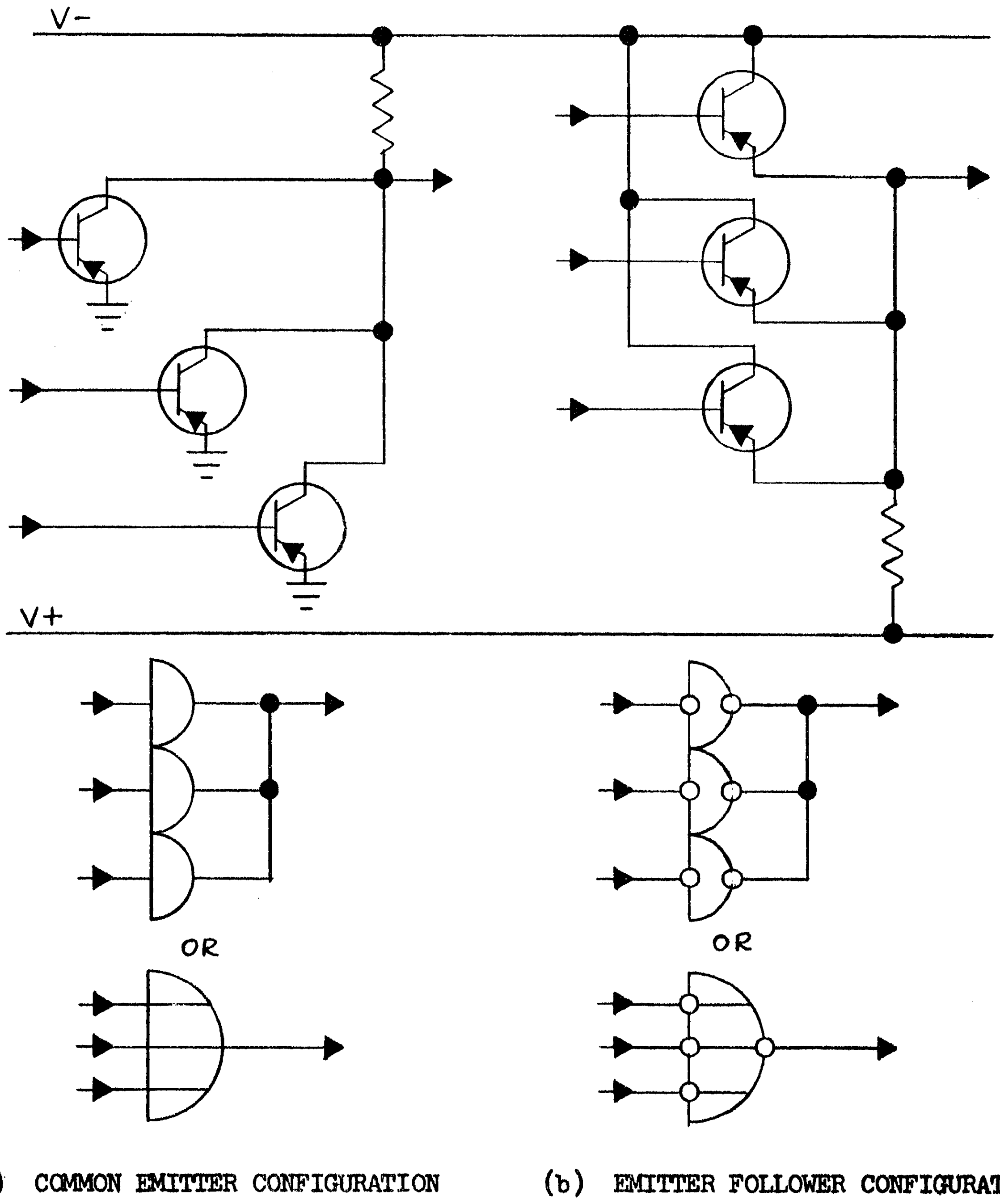

V−
V+
OR
OR
(a) COMMON EMITTER CONFIGURATION
(b) EMITTER FOLLOWER CONFIGURATION

Fig. 2.7 Parallel Combinations

Fig. 2.8 depicts a series combination of 2 direct coupled transistors. The common emitter transistor is called the bottom transistor, while the ungrounded transistor is called the top transistor. The condensed symbol for a series combination is distinguished by termination of the input lines at the diameter of the semi-circle. This combination is a two high gate. Where it is necessary to distinguish between top and bottom transistors, expanded logical symbols should be used.

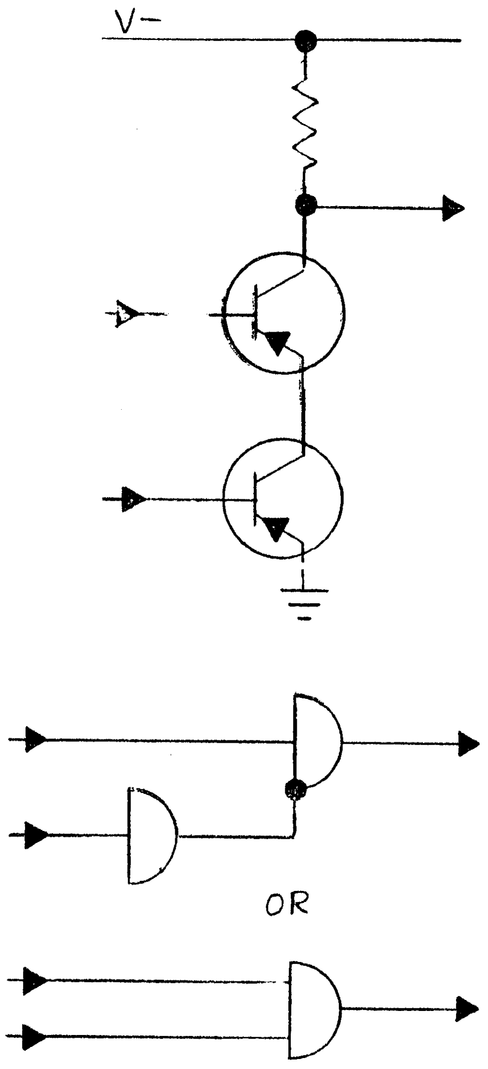


Fig. 2.8 Series Combination

A tree combination of four resistance coupled emitter followers is given in Fig. 2.9. This particular combination performs no logical function in a circuit, but is used when many transistors must be coupled to the same signal.

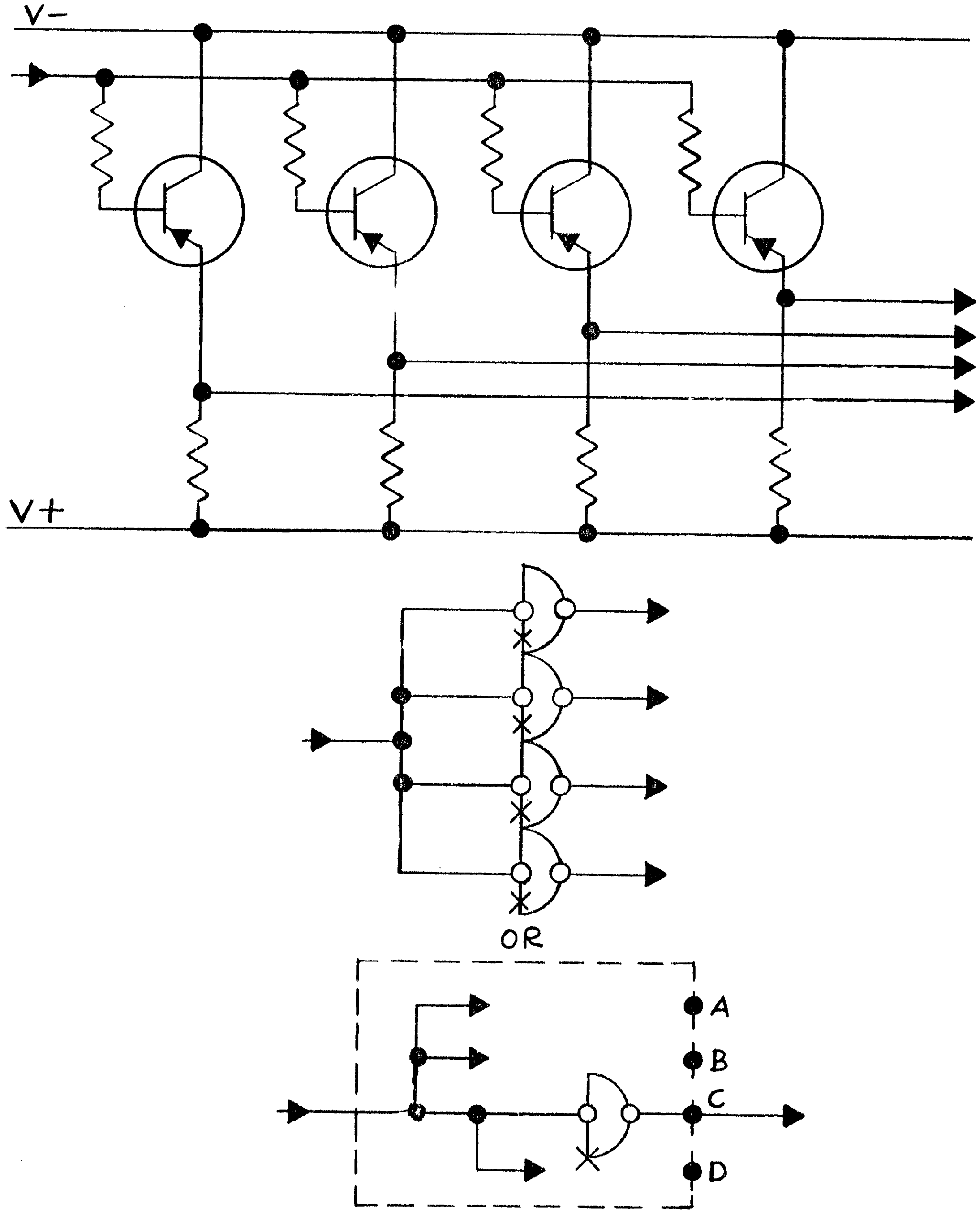


Fig. 2.9 Tree Combination

Having now become familiar with several classes of transistor circuits, we can proceed to consideration of the application of these (and other) circuits.

CHAPTER 3. PNP TRANSISTOR LOGIC

3.0 This chapter presents the most commonly encountered circuits and includes their conventional and logical symbols, their rules of operation and examples of their application.

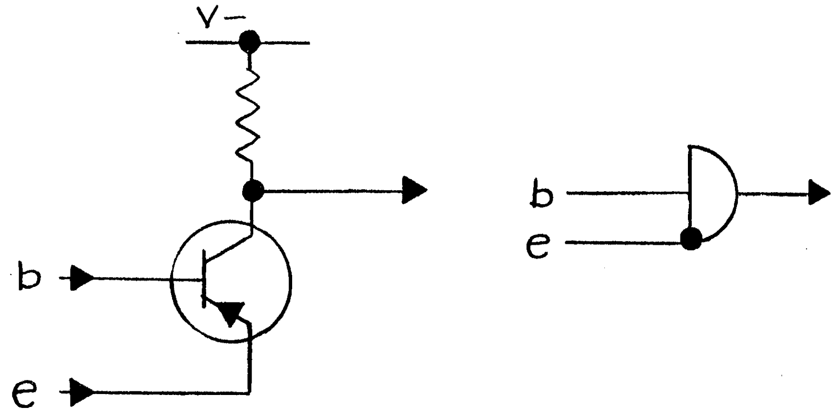
Under "Symbols" conventional symbolism is employed on the left and logical symbolism on the right. Where two logical symbols are shown with the word "OR" between them, the two symbols are equivalent. If "OR" does not appear, the different symbols stand for varieties of the circuit shown which are not physically interchangeable. Under "Logic" the most common symbol is used to illustrate the logic of the circuit type.

This chapter also includes some non-logical symbols of particulary frequent occurrence.

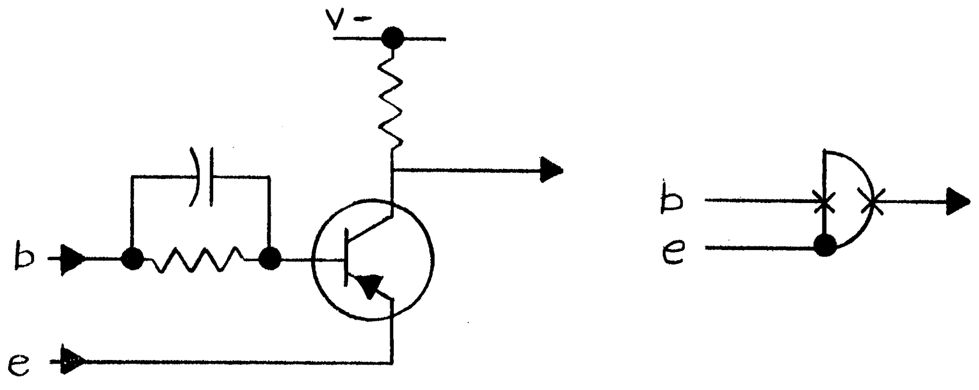
3.1 PNP UNGROUNDED CONFIGURATION

1. Symbols

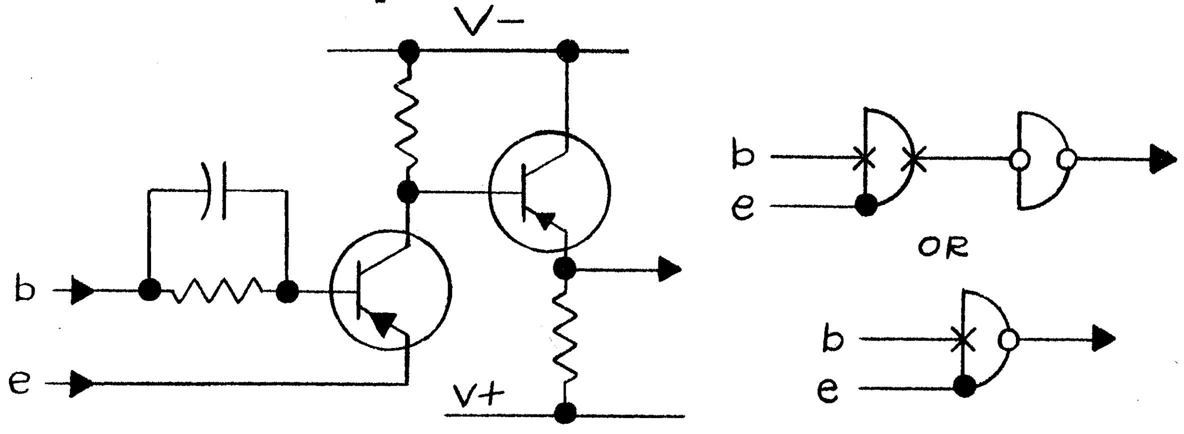
A. Direct Coupled



B. Resistance Coupled



C. Composite

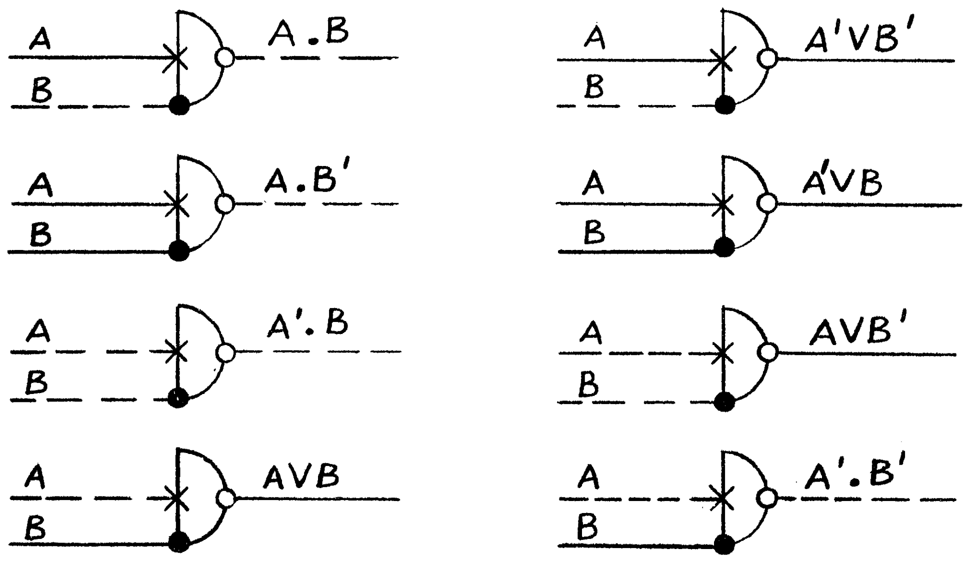


2. **Operation**

The output is negative if b is positive or e is negative.

The output is positive if b is negative and e is positive.

3. **Logic**

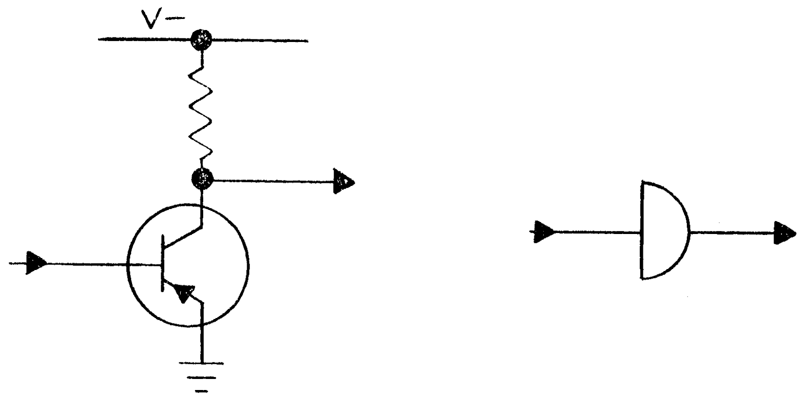


The logic of the direct and resistance coupled versions of this configuration is the same as that of the composite version shown above.

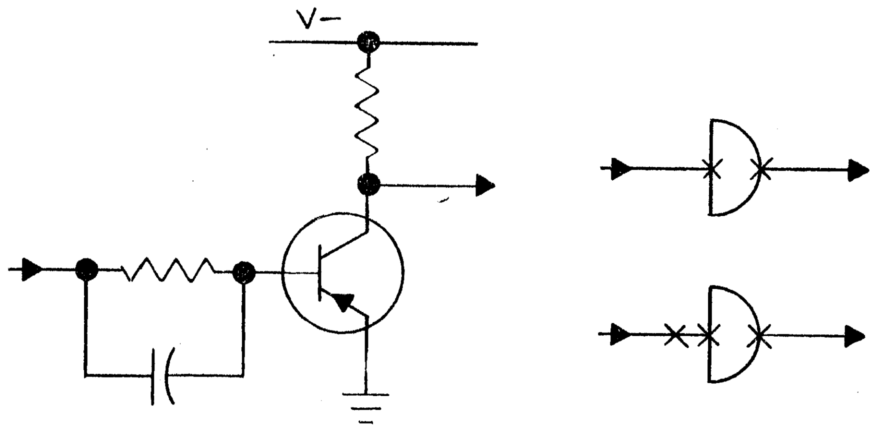
3.2 PNP COMMON EMITTER CONFIGURATION

1. Symbols

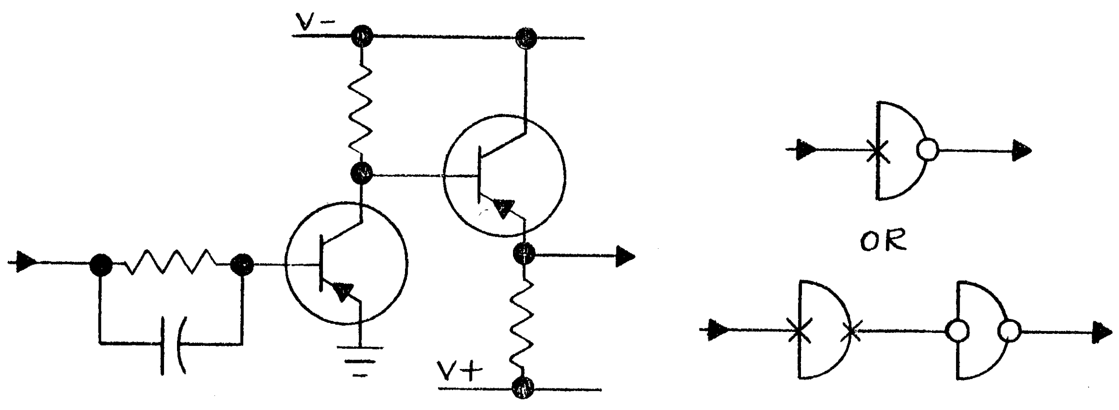
A. Direct Coupled



B. Resistance Coupled



C. Composite

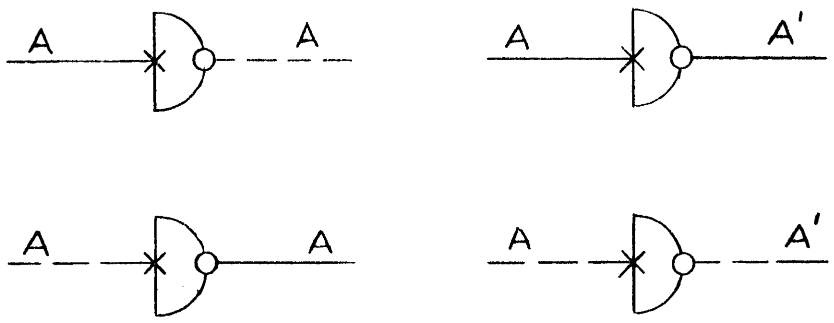


2. Operation

The output is negative if the input is positive.

The output is positive if the input is negative.

3. Logic

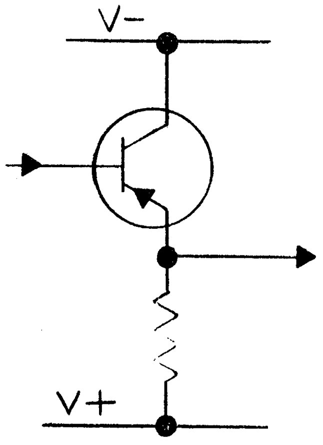


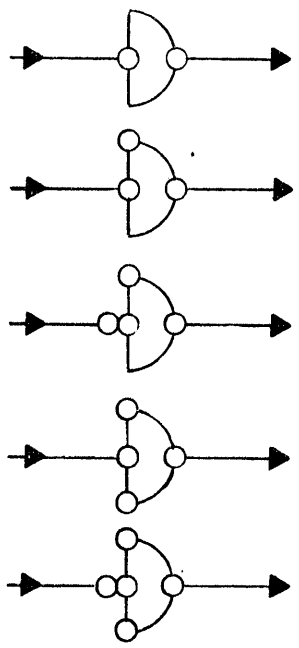
The logic of the direct and resistance coupled versions of this configuration is the same as that of the composite version shown above.

3.3 PNP EMITTER FOLLOWER CONFIGURATION

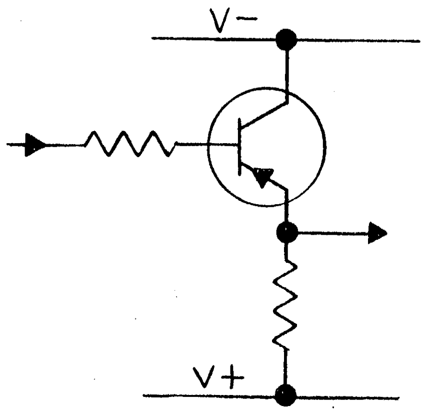
1. Symbols

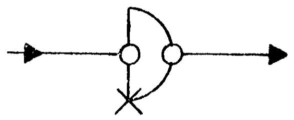
A. Direct Coupled





B. Resistance Coupled



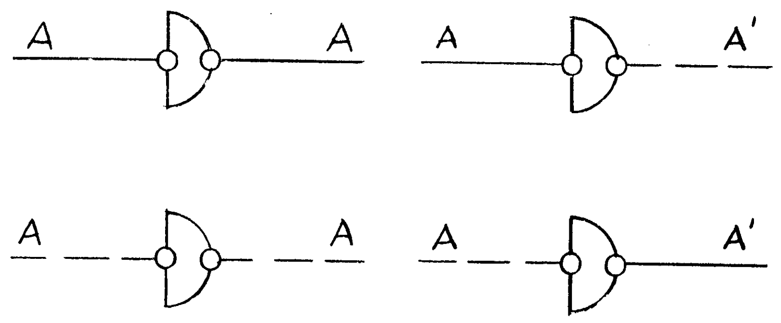


2. Operation

The output is positive if the input is positive.

The output is negative if the input is negative.

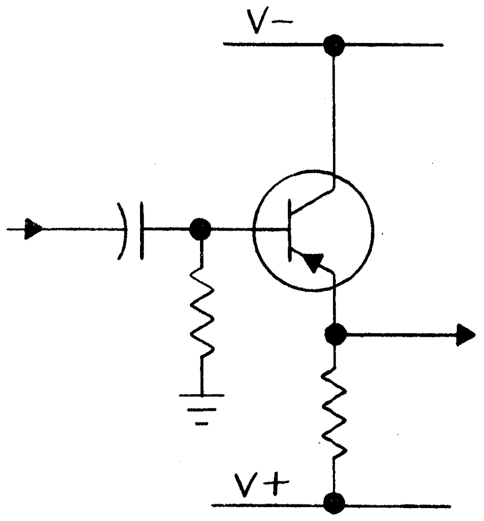
3. Logic

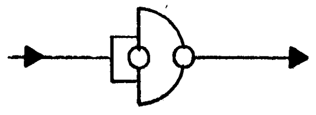


The logic of the resistance coupled version of this configuration is the same as that of the direct coupled version shown above.

3.4 PNP EMITTER FOLLOWER, CAPACITANCE COUPLED

1. Symbols





2. Operation

The output is positive except when the input is changing from positive to negative.

The output is negative when the input is changing from positive to negative.

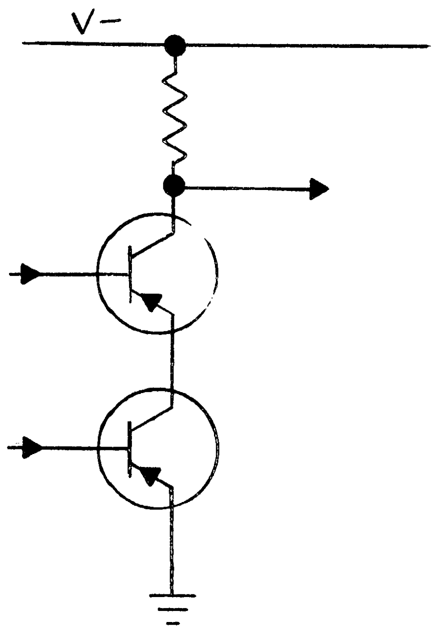
3. Logic

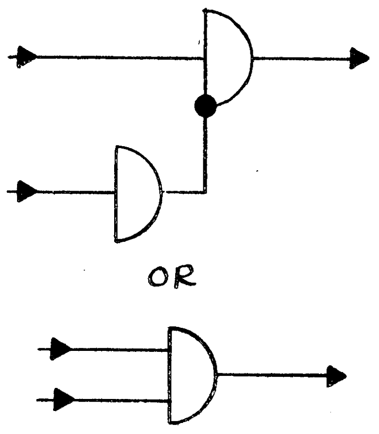
This circuit is used in combination with a counter Flip-Flop.

3.5 PNP TWO HIGH GATE

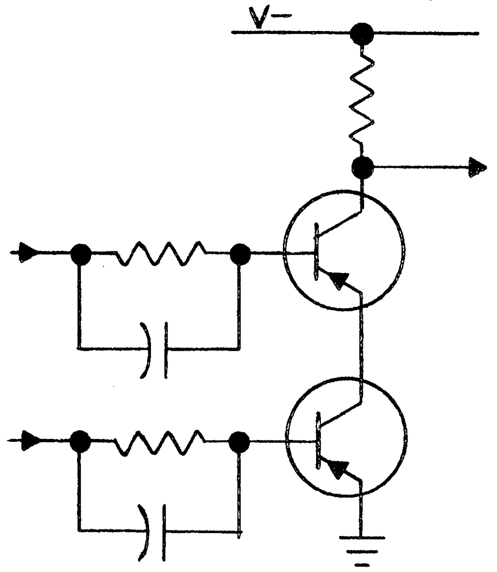
1. Symbols

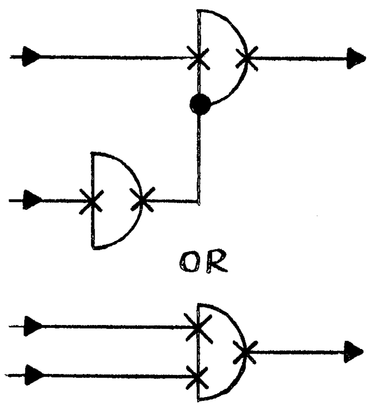
A. Direct Coupled



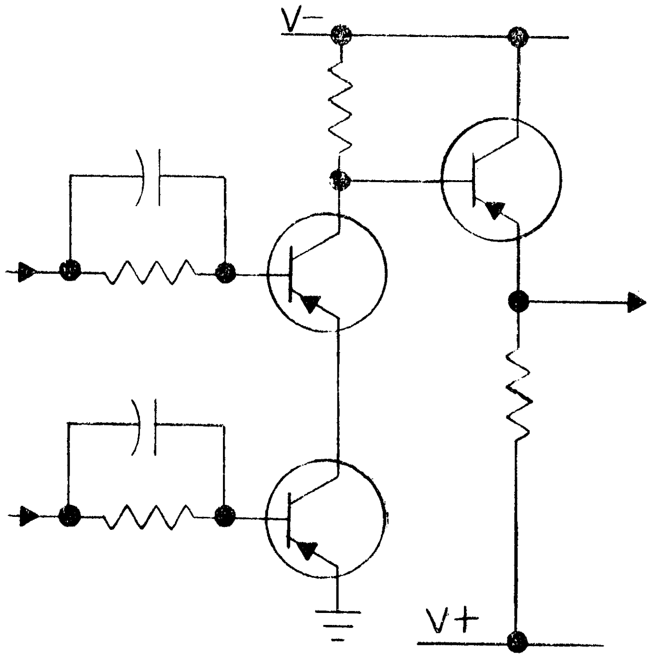


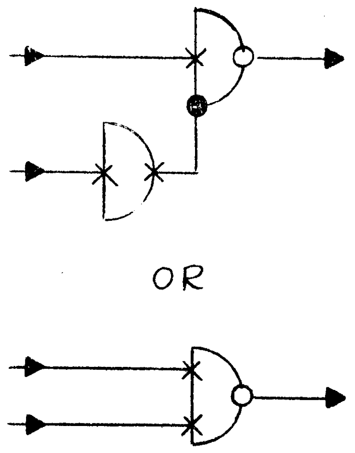
B. Resistance Coupled





C. Composite



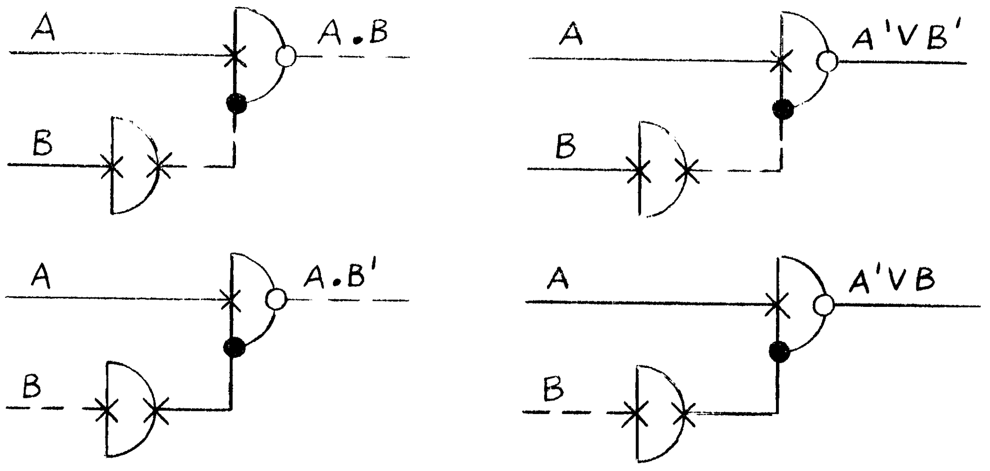


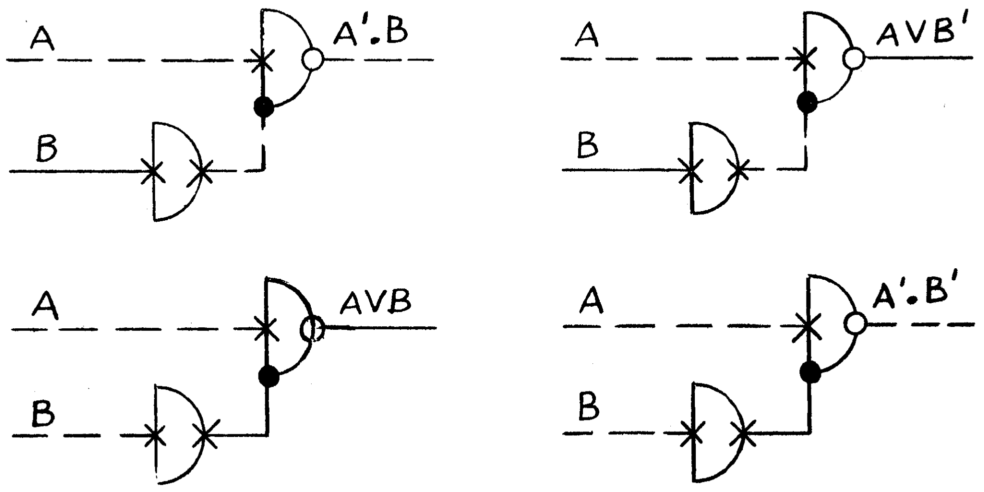
2. Operation

The output is negative if either input is positive.

The output is positive if both inputs are negative.

3. Logic





The logic of the direct and resistance coupled versions of this configuration is the same as that of the composite version shown above.

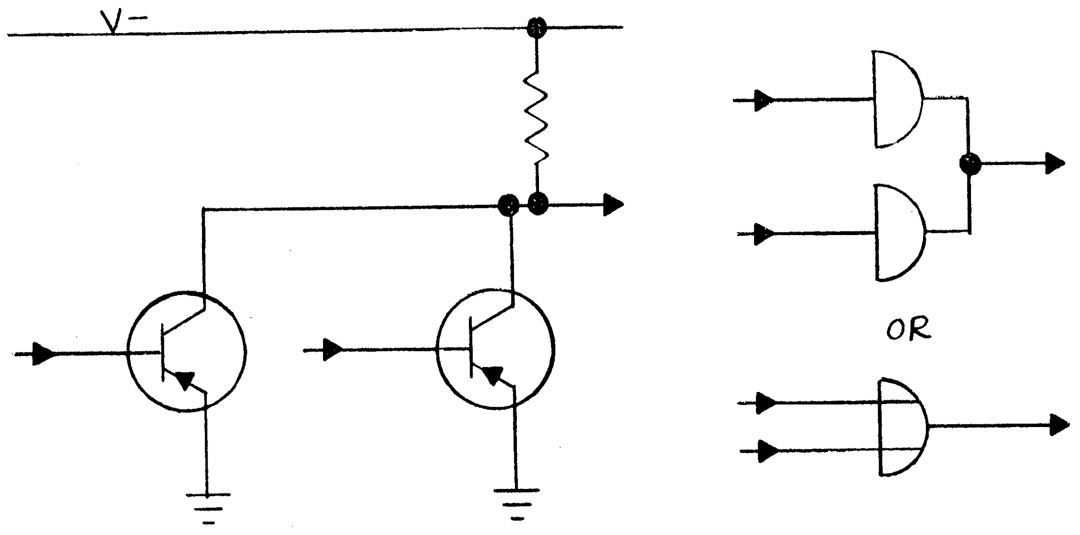
4. Remarks

Note that for this combination a solid output line bears an OR function and a dotted output line bears an AND function.

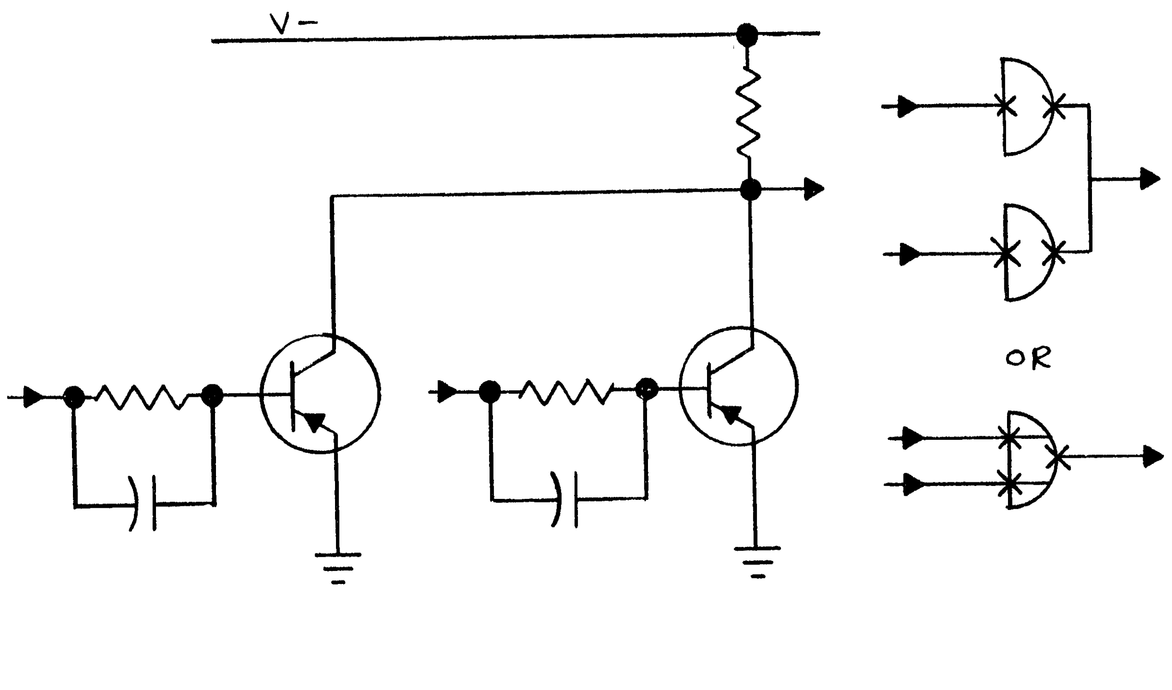
3.6 PNP PARALLEL GATE-COMMON EMITTER CONFIGURATION

1. Symbols

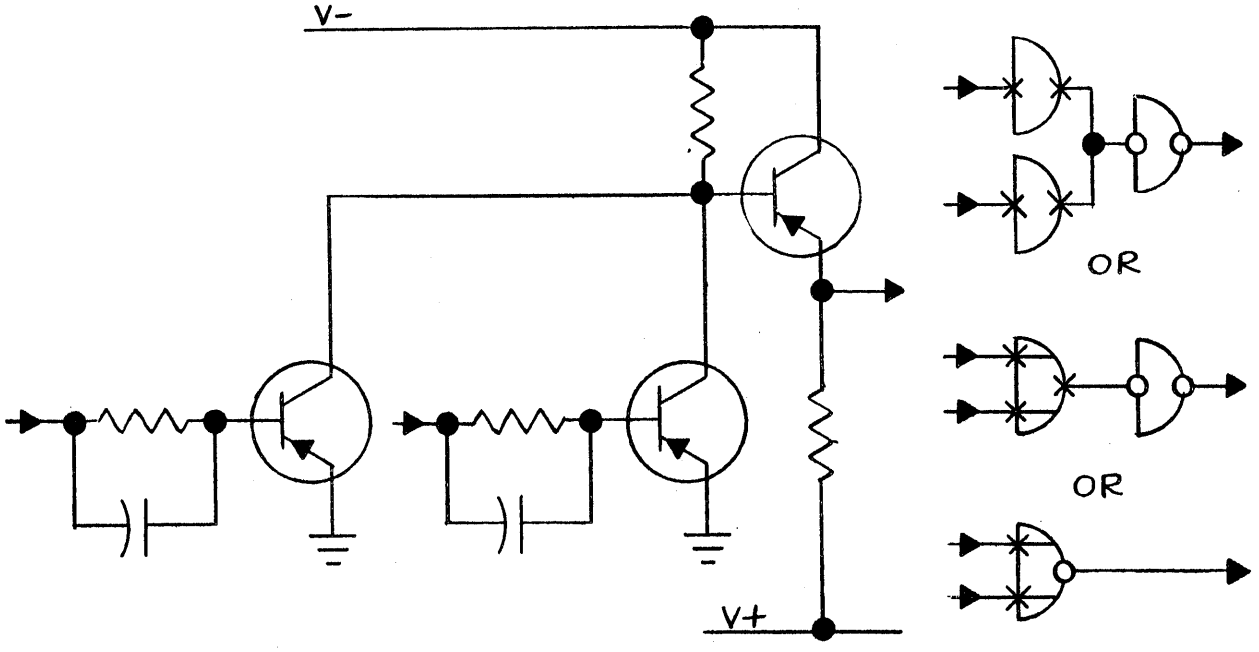
A. Direct Coupled



B. Resistance Coupled



C. Composite

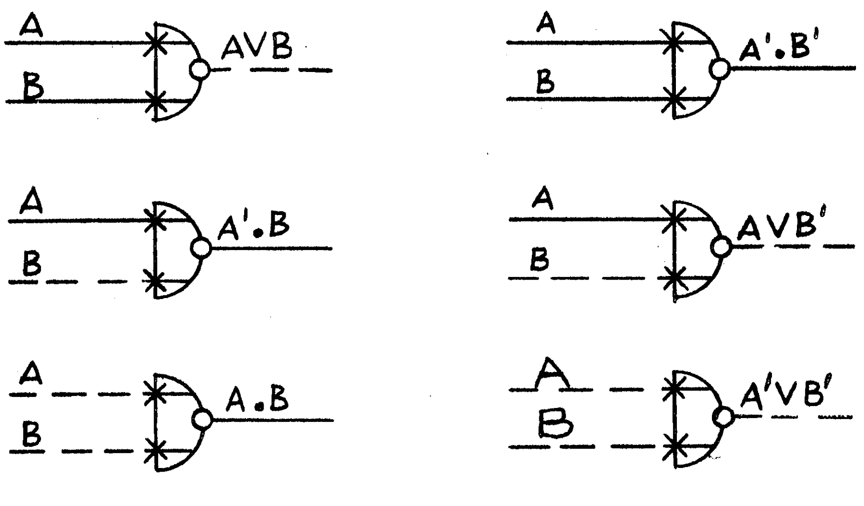


2. Operation

The output is negative if all inputs are positive.

The output is positive if any input is negative.

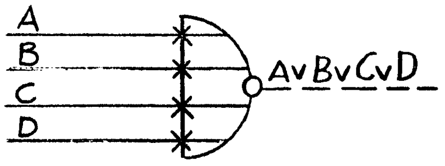
3. Logic

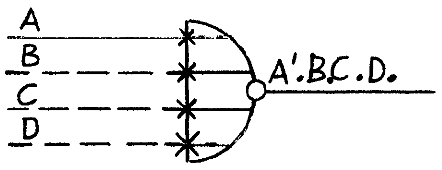


The logic of the direct and resistance coupled versions of this combination is the same as that of the composite version shown above.

4. Remarks

(a) The logic of this combination may be extended to cover cases involving more than two inputs. For example:



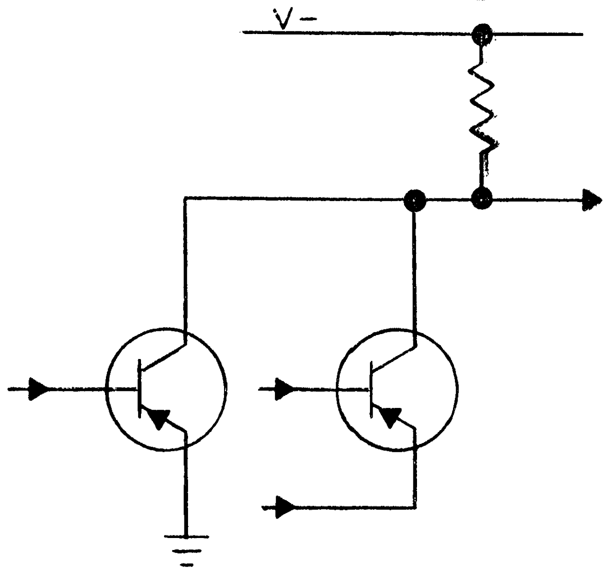


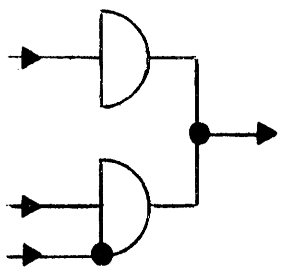
(b) Note that for this combination a solid output line bears an AND function and a dotted line bears an OR function.

3.7 PNP PARALLEL GATE-COMMON EMITTER AND UNGROUNDED CONFIGURATIONS

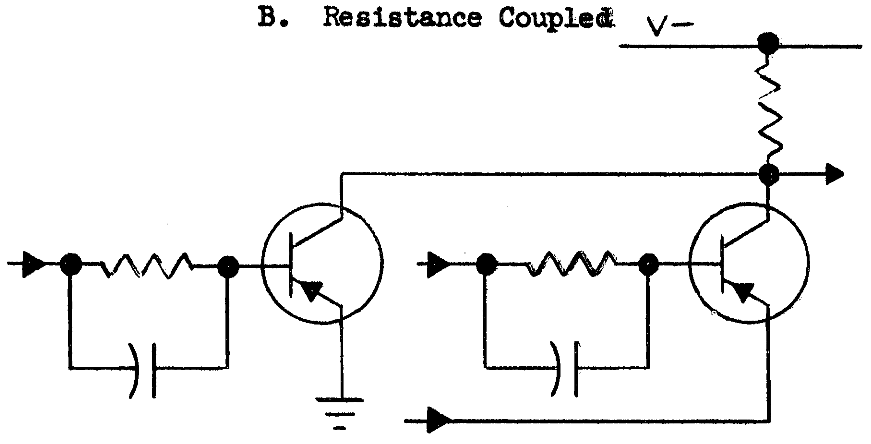
1. Symbols

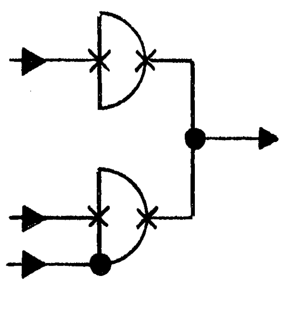
A. Direct Coupled





B. Resistance Coupled



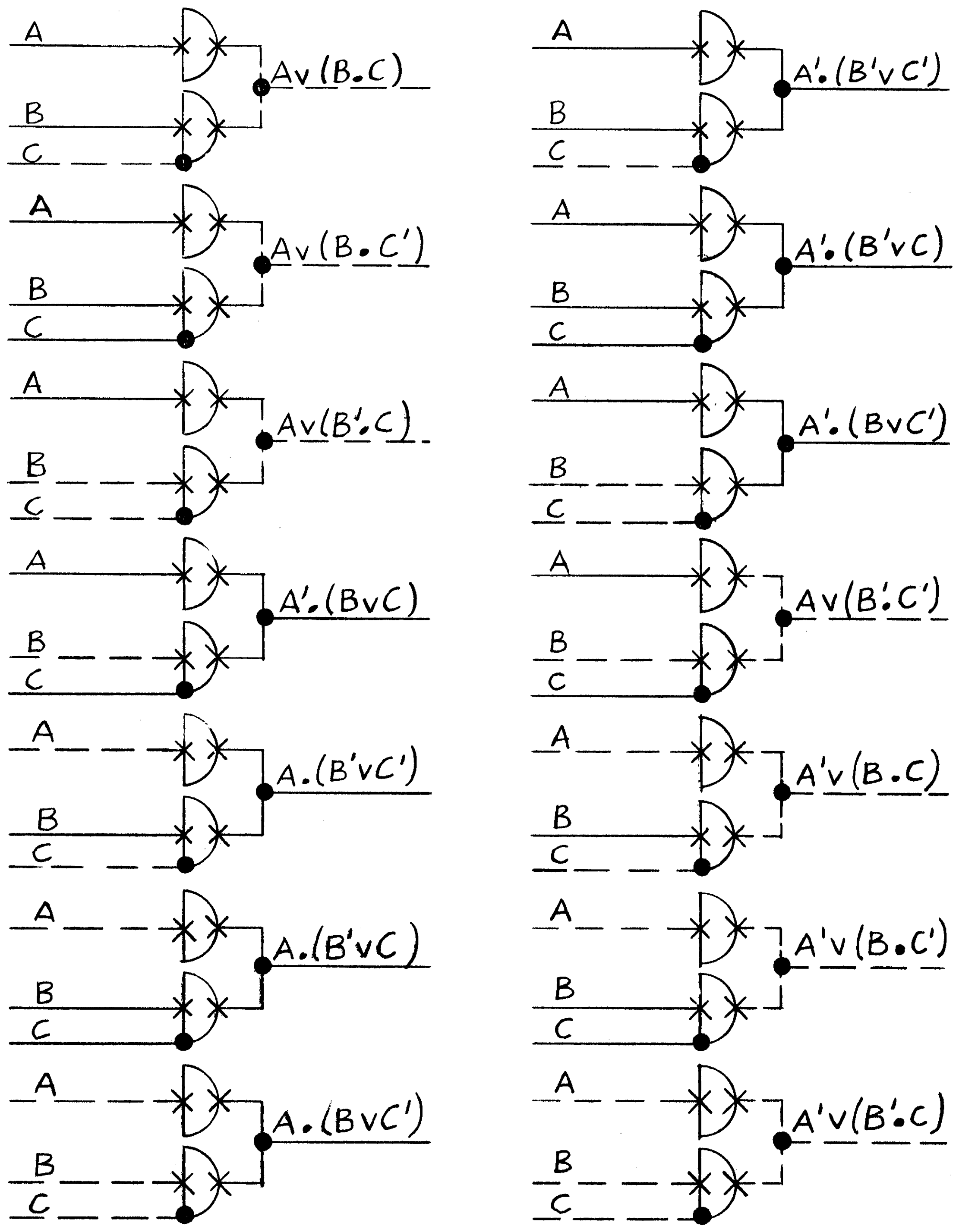


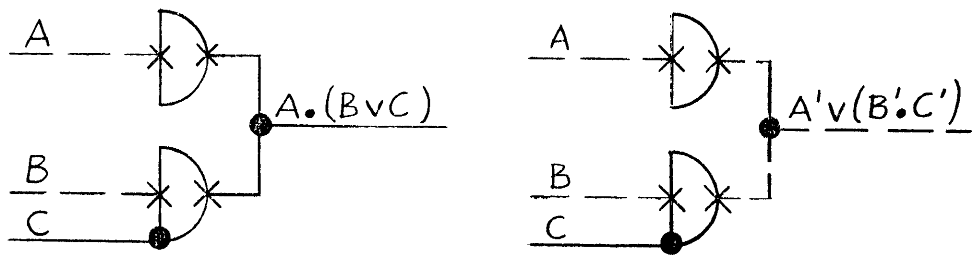
2. Operation

The output is negative if all common-emitter inputs are positive and each ungrounded transistor has either a positive base or a negative emitter or both.

The output is positive if any common emitter input is negative or any ungrounded transistor has a negative base and a positive emitter.

3. Logic

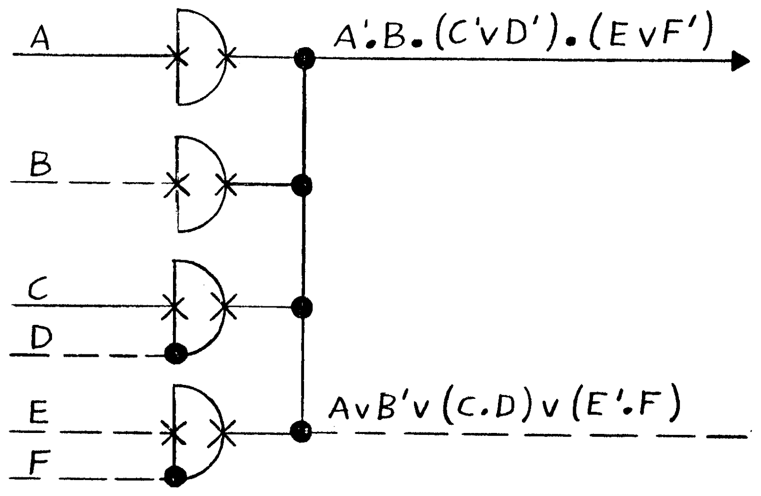

A
B
C
Av(B.C)
A'.(B'vC')
Av(B.C')
A'.(B'vC)
Av(B'.C)
A'.(BvC')
A'.(BvC)
Av(B'.C')
A.(B'vC')
A'v(B.C)
A.(B'vC)
A'v(B.C')
A.(BvC')
A'v(B'.C)



The logic of the direct coupled version of this combination is the same as that of the resistance coupled version shown above.

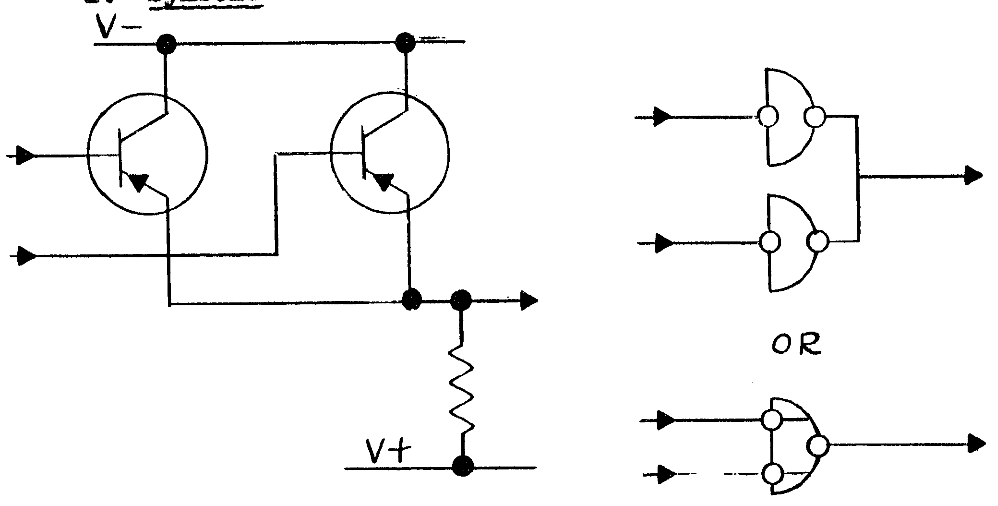
4. Remarks

The logic of this combination may be extended to cover cases involving more than two inputs. For example:



3.8 PNP PARALLEL GATE - EMITTER FOLLOWER CONFIGURATION

1. Symbols

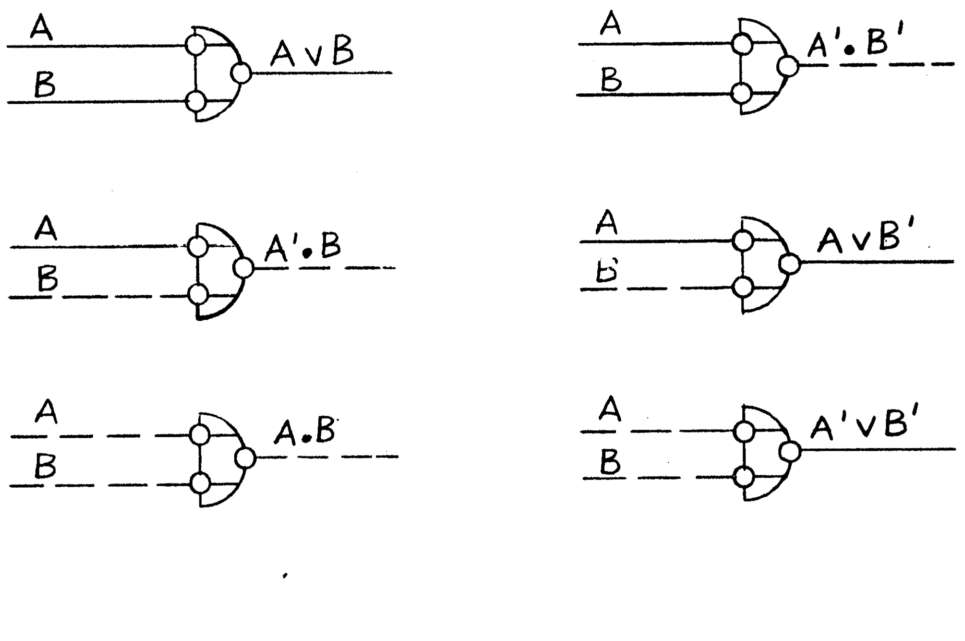


2. Operation

The output is positive if all inputs are positive.

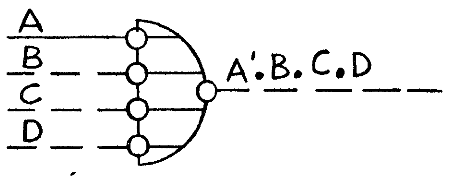
The output is negative if any input is negative.

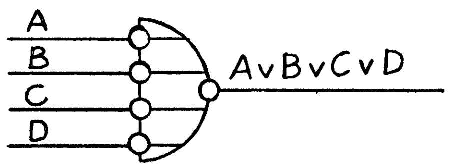
3. Logic



4. Remarks

(a) The logic of this combination may be extended to cover cases of more than two inputs. For example:



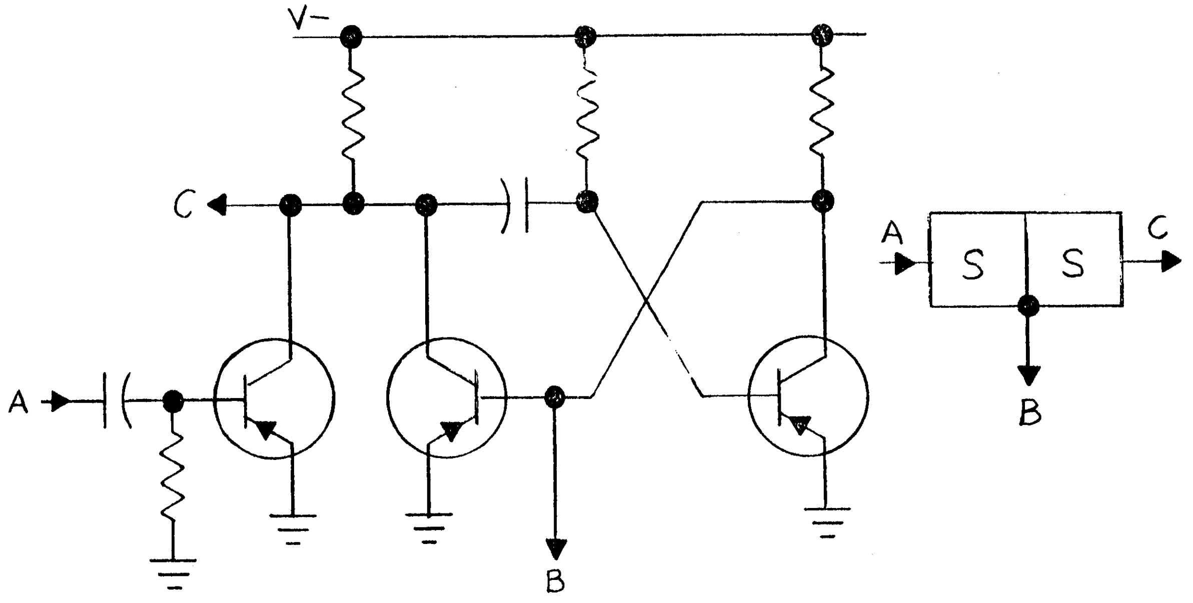


(b) Note that for this combination a solid output line bears an OR function and a dotted output line, an AND function.

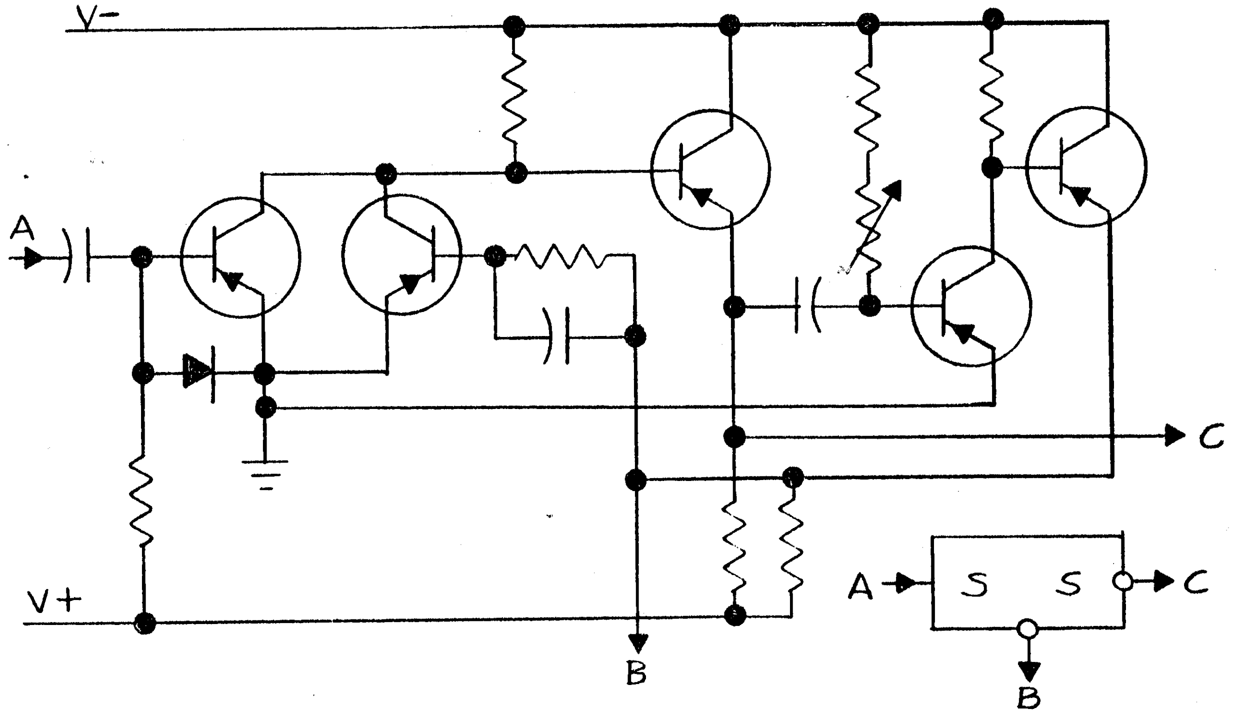
3.9 SINGLE SHOT

1. Symbols

A. Direct Coupled



B. Resistance Coupled

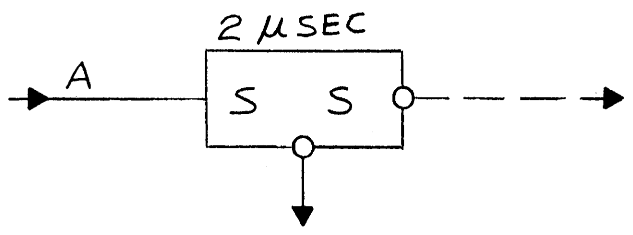


A single shot has a stable state and an unstable state.

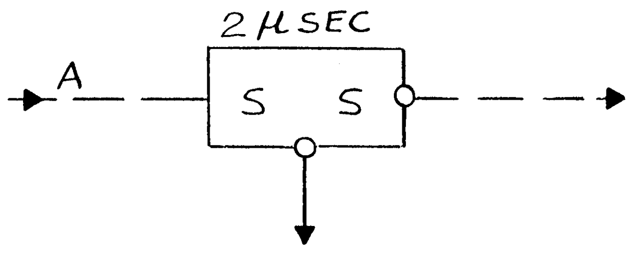
If the single shot is in its stable state, output B is positive and output C is negative. If it is in its unstable state, output B is negative and output C is positive.

The single shot goes from its stable state to its unstable state if and only if input A changes from positive to negative. The single shot remains in its unstable state for a predetermined interval then returns to its stable state.

3. Logic



The single shot output lines will become active when line A becomes active, remain active for 2 u sec and then become normal.

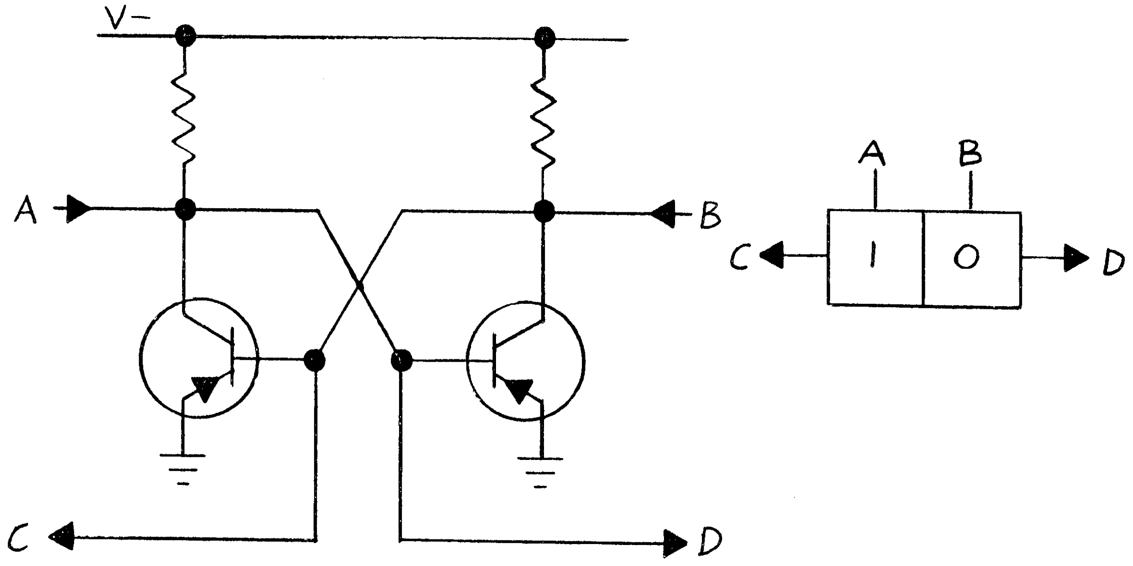


The single shot output lines will remain normal even though line A become active. If line A become active and then become normal, the output lines will become active when line A becomes normal, remain active for 2 u sec, and then become normal.

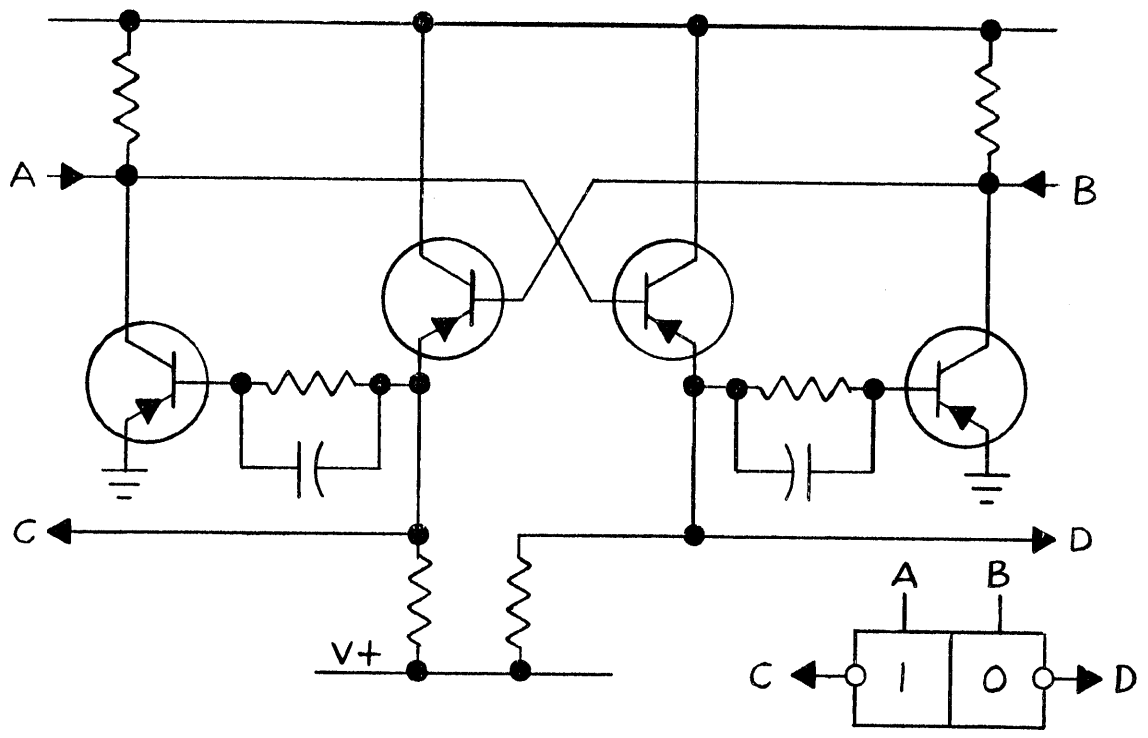
3.10 **FLIP-FLOP**

1. Symbols

A. Direct Coupled



B. Resistance Coupled



2. Operation

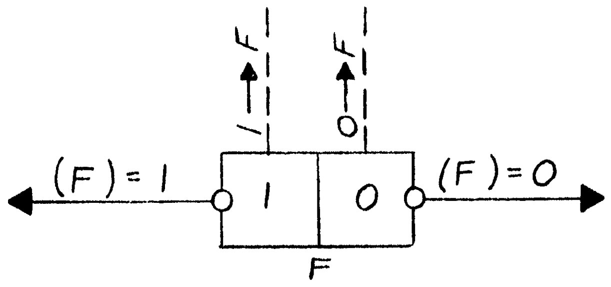
The flip-flop has 2 stable states called "0" and "1".

When it is in state 0, A and D are negative and B and C are positive. When it is in state 1, A and D are positive and B and C are negative.

If the flip-flop is in state 0 and A is made positive by an external condition, it will go to state 1 and remain in state 1 even though the condition which made A positive cease to exist.

If the flip-flop is in state 1 and B is made positive by an external condition, it will go to state 0 and remain in state 0 even though the condition which made B positive cease to exist.

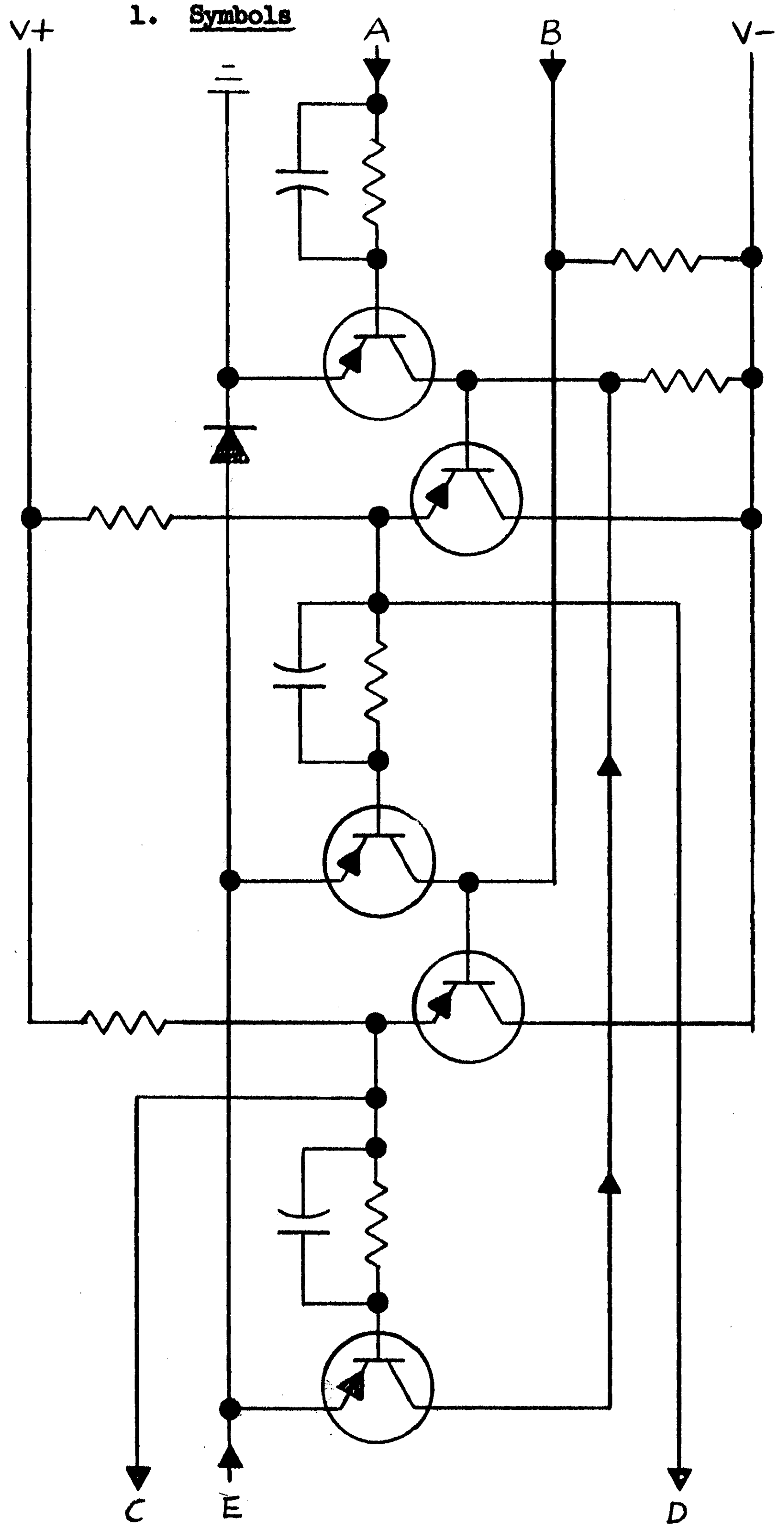
3. Logic

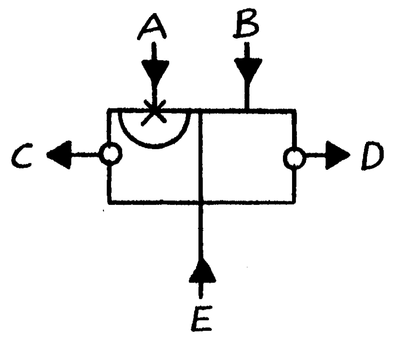


The logic of the direct coupled flip-flop is the same as that of the resistance coupled version shown above.

3.10a COUNTER FLIP-FLOP

1. **Symbols**





2. Operation

The operation of the counter flip-flop is like that of the resistance coupled flip-flop with the following exceptions:

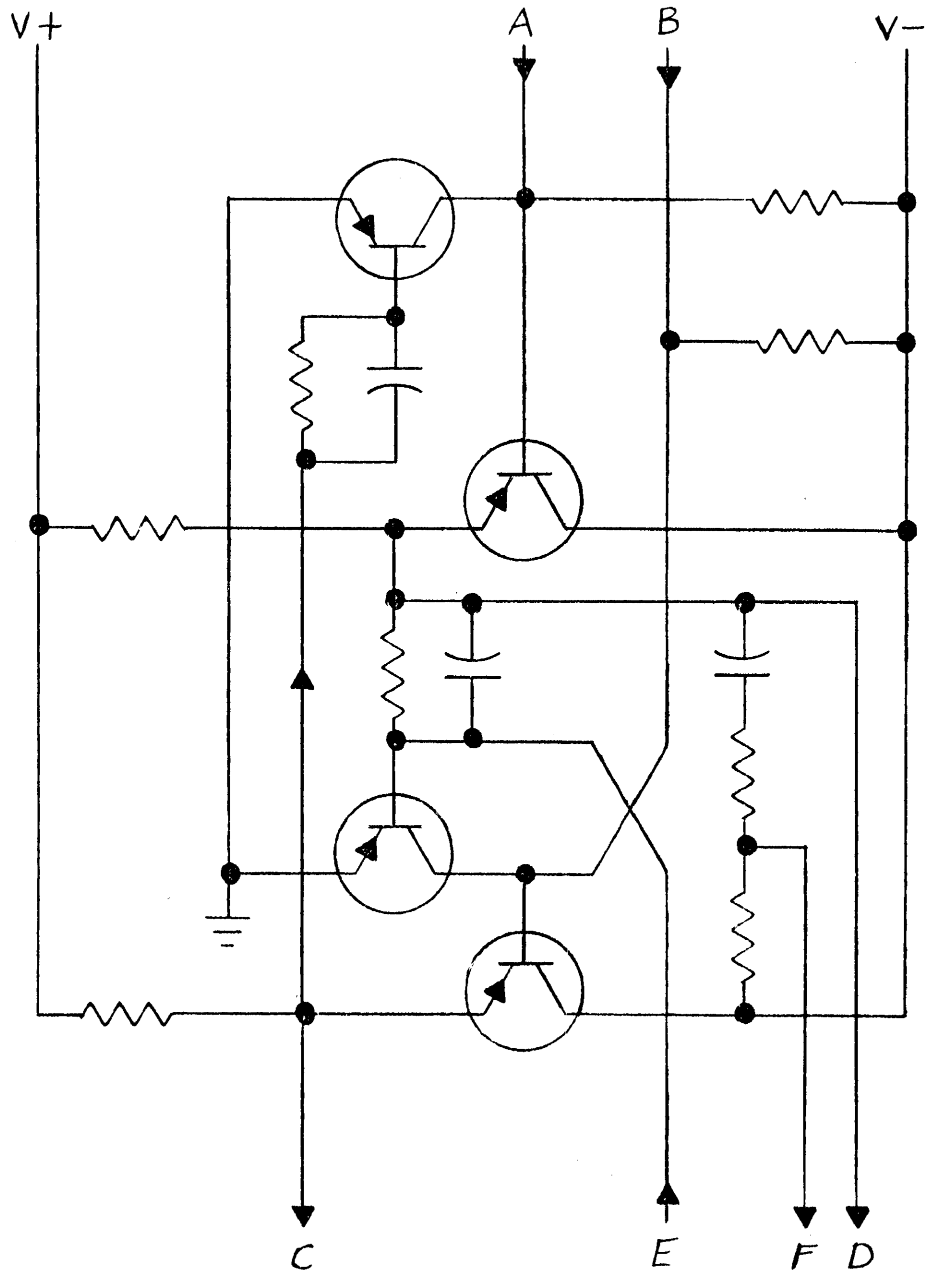
(A) C becomes negative (and D positive) when A is made negative.
(B) The counter flip-flop changes state (0 to 1 or 1 to 0) when a negative pulse appears at E.

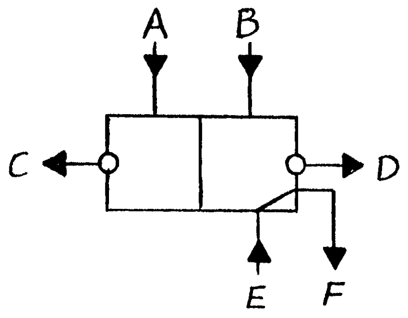
3. Application

Counter flip-flops are used in conjunction with capacitance-coupled emitter followers to form multi-stage binary counters.

3.10b SHIFT FLIP-FLOP

1. Symbols





2. Operation

The operation of a shift flip-flop is like that of a resistance-coupled flip-flop with the following exceptions:

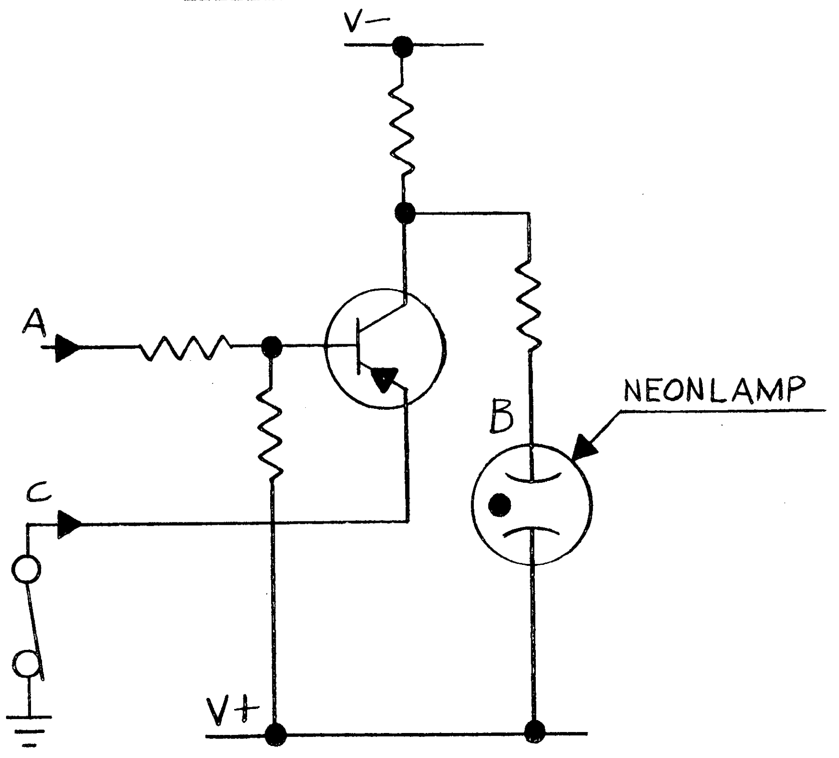
(A) At such time as D changes from negative to positive, a positive pulse appears at F.
(B) At such time as D changes from positive to negative, a negative pulse appears at F.
(C) At other times, F is negative.
(D) If a negative pulse be imposed on E, then D becomes (and remains) negative and C becomes (and remains) positive.

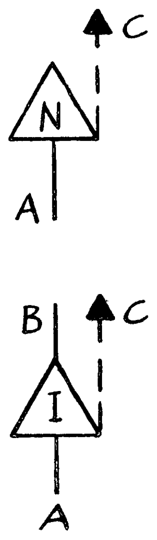
3. Application

Shift flip-flops are used with NPN ungrounded transistors to form single-rank shift registers

3.11 NEON-INDICATOR CIRCUIT

1. Symbols





2. Operation

The lamp is lit if the input is positive or point C is ungrounded.

The lamp is extinguished if the input is negative and point C is grounded.

3. Logic

This circuit has no logical function.

4. Remarks

An 'N' in the triangle means that the lamp is in the same assembly of the transistor and resistors.

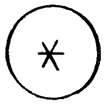
An 'I' in the triangle means that the lamp is remote from the transistor and resistors.

The switch is remote from the transistor and resistors.

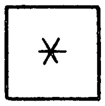
3.12 TERMINAL

1. Symbols

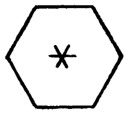
A. Input



B. Output



C. Intraconnecting



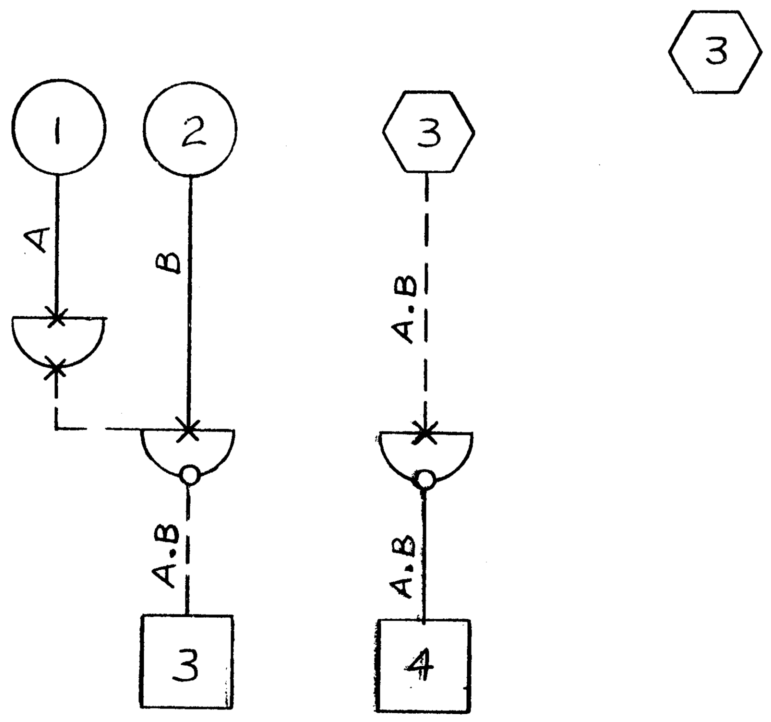
* The asterisk is not a part of the symbol. It is replaced by the terminal designation.

2. Rule

A. Terminal symbols are placed at the ends of lines. On schematic logic diagrams they bear numerical designations. On chassis logic diagrams they bear the designations of the electrical terminals to which they correspond.

B. Intraconnecting Terminal symbols are used to indicate a connection between two points on the same diagram. An intraconnecting terminal may correspond to an input terminal, an output terminal, or another intraconnecting terminal. In any case, an intraconnecting terminal bears the same designation as the other terminals to which it refers.

3. Examples

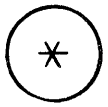


means that the two dotted lines A.B are equivalent.

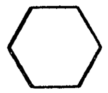
3.13 TERMINAL REFERENCE

1. Symbols

A. Interconnecting



B. Intraconnecting



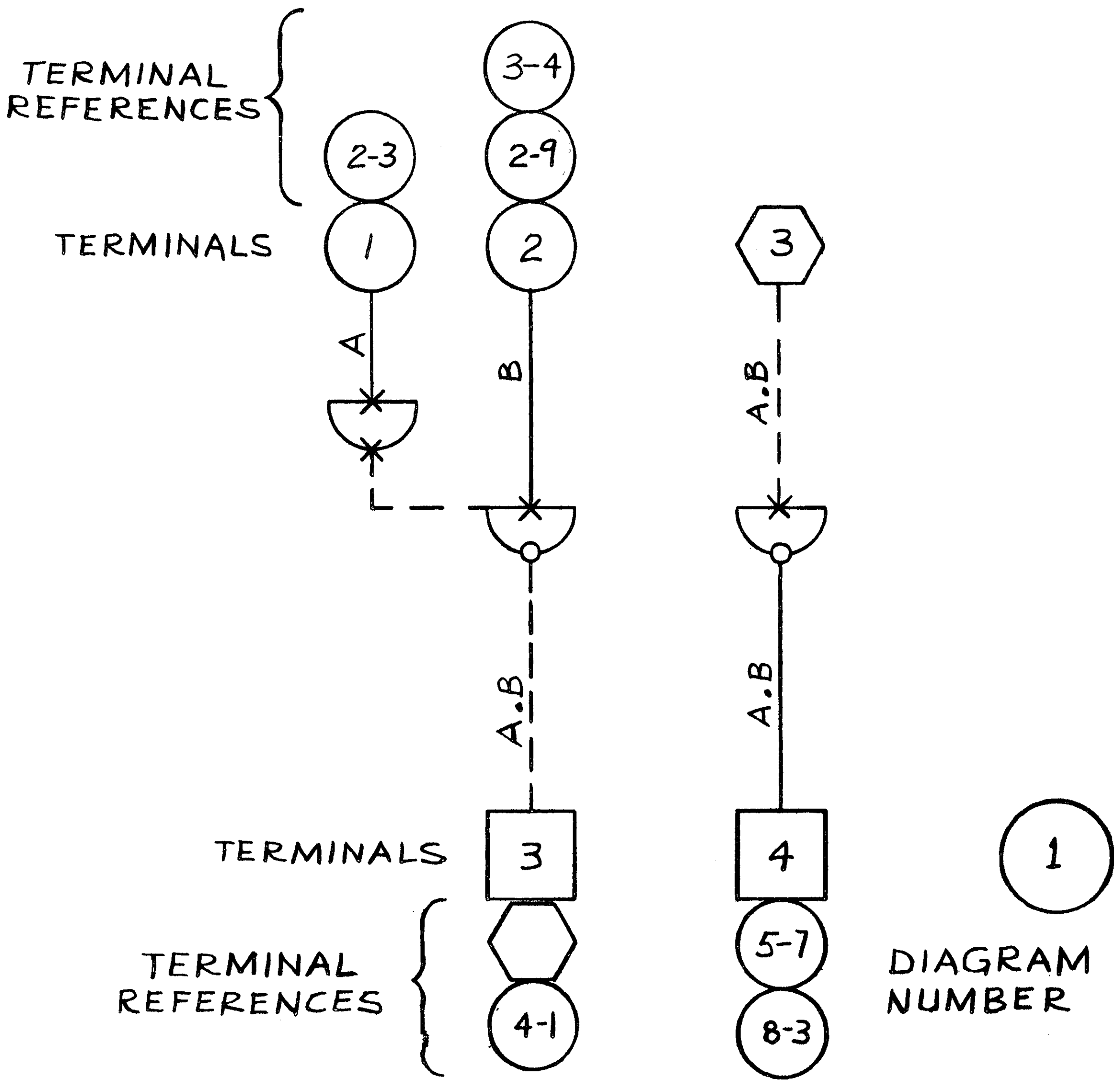
* The asterisk is not a part of the symbol. It is replaced by the diagram and terminal designations.

2. Rule

A. Interconnecting terminal reference symbols are used with input and output terminal symbols to facilitate the tracing of a line from one logical diagram to another. At an output, references are to inputs to which the signal goes. At an input, references are to the outputs from which the signal comes.

B. Intraconnecting terminal reference symbols indicate that there is an intraconnecting terminal to be considered in tracing the signal.

3. <u>Examples</u>


TERMINAL REFERENCES
3-4
2-3
2-9
TERMINALS
1
2
3
A
B
A.B
A.B
A.B
TERMINALS
3
4
1
TERMINAL REFERENCES
5-7
4-1
8-3
DIAGRAM NUMBER

(2-3) means that input (1) is equivalent to output no. 3 of diagram no. 2.

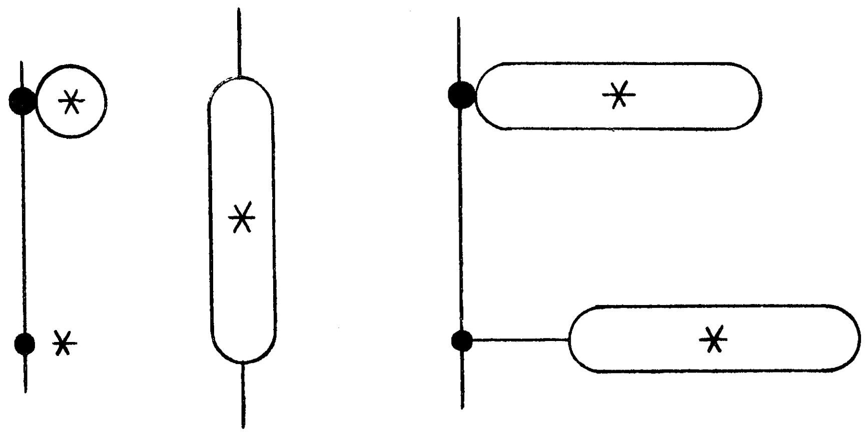
(3-4) (2-9) means that input (2) is equivalent to output no. 9 of diagram 2 and output no. 4 of diagram no. 3.

⬡ (4-1) means that output $\boxed{3}$ is equivalent to input ⟨3⟩ on this diagram and input no. 1 on diagram no. 4.

(5-7) (8-3) means that output $\boxed{4}$ is equivalent to input no. 3 on diagram no. 8 and input no. 7 on diagram no. 5.

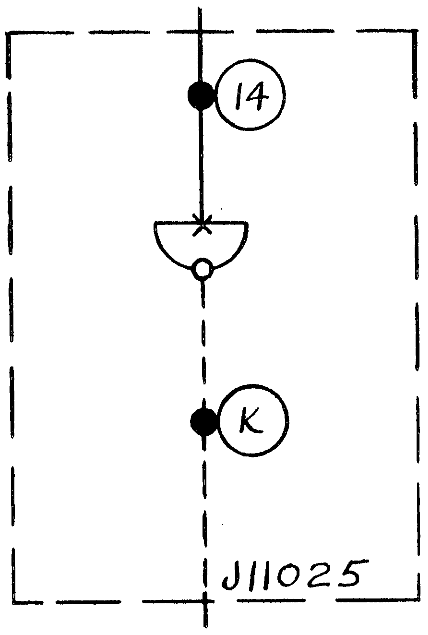
3.14 WIRING POINT

1. Symbols

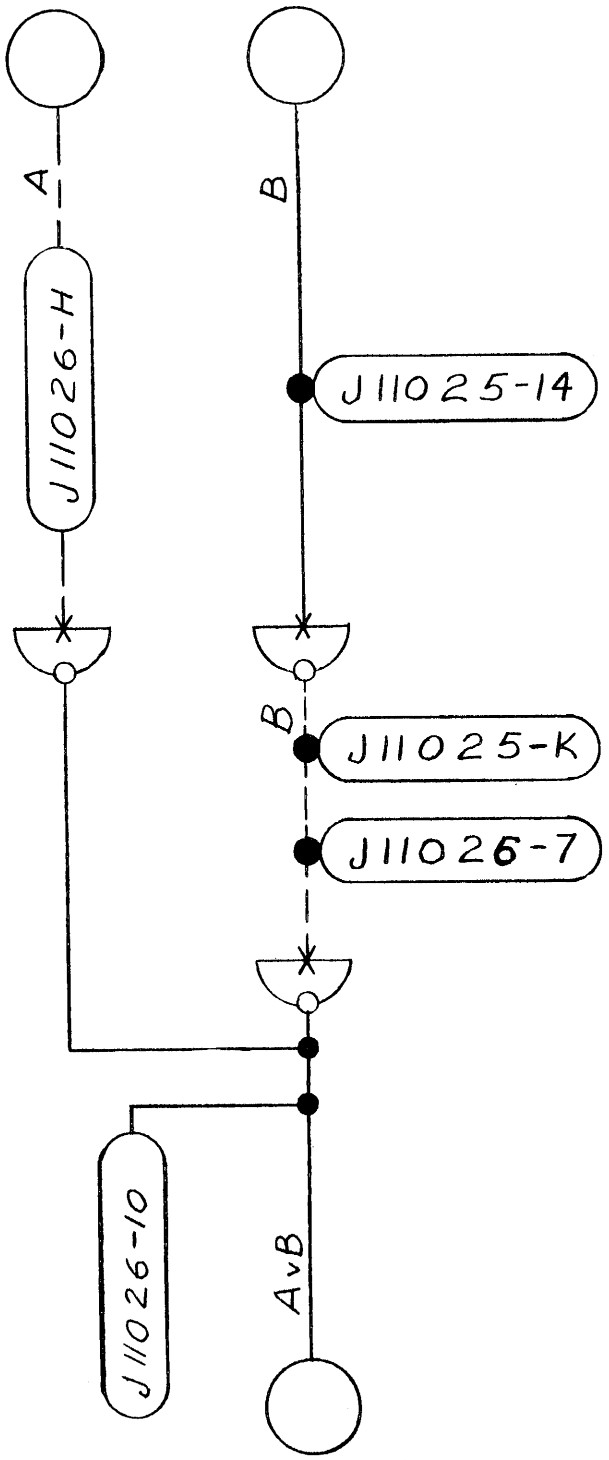


* The asterisk is not a part of the symbol. It is replaced by the wiring point designation or key to the wiring point table.

2. Examples

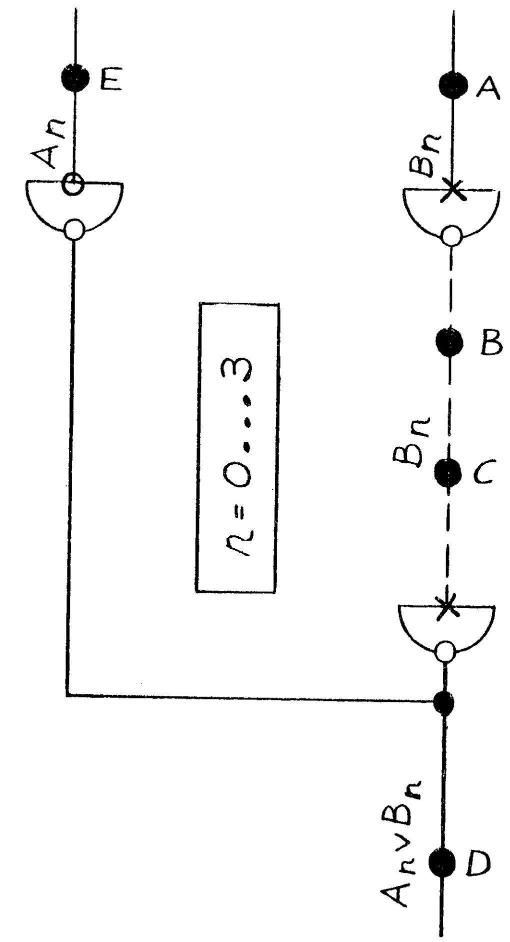


For this example, the normally positive wire may be found at terminal 14 and the normally negative wire at terminal K of receptacle J11025.



For this example signal A may be found at terminal H of receptacle J11026; signal B (normally positive), at terminal 14 of receptacle J11025; signal B (normally negative), at terminal K of receptacle J11025 and terminal 7 of receptacle J11026; and signal AvB, at terminal 10 of receptacle J11026.

Where the typical bit representation is used, one diagram may stand for several distinct but similar units of equipment. In such cases wiring point references are tabulated and a key letter is placed adjacent to the line.



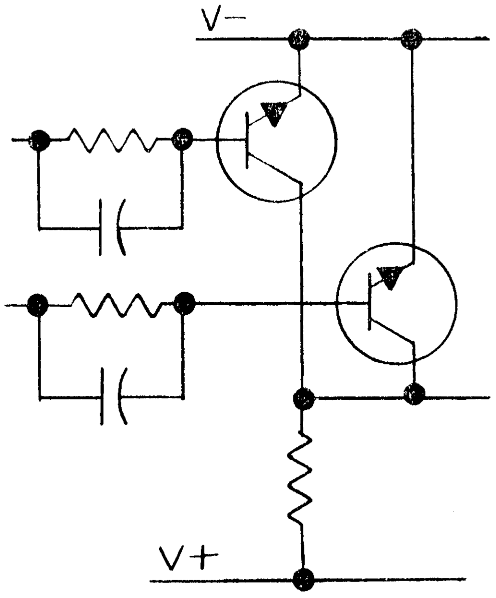
	0	1	2	3
A	J11025-14	J11025-15	J11027-14	J11027-15
B	J11025-K	J11025-L	J11027-K	J11029-L
C	J11026-7	J11026-8	J11028-7	J11028-8
D	J11026-10	J11026-11	J11028-10	J11028-11
E	J11026-H	J11026-J	J11028-H	J11028-J

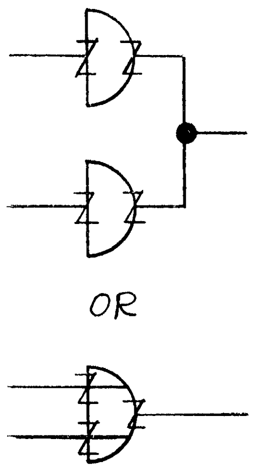
The wiring points for n = 0 are the same as those for the preceding example. The remainder of the table illustrates the method.

CHAPTER 4. NPN TRANSISTOR LOGIC

4.1 NPN PARALLEL GATE

1. Symbols



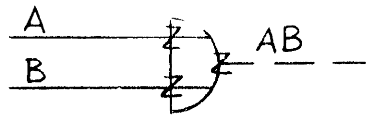


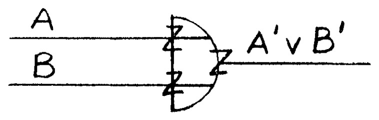
2. Operation

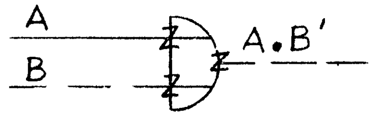
The output is positive if all inputs are negative.

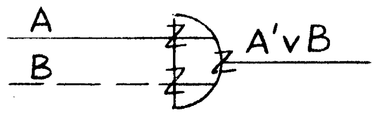
The output is negative if any input is positive.

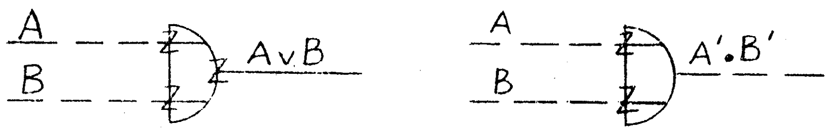
3. Logic









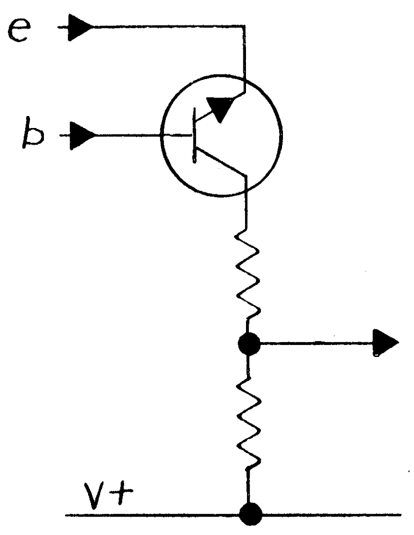


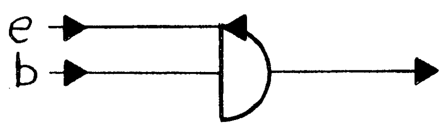
4. Remarks

The logic of this circuit is the same as that of the PNP two-high gate.

4.2 NPN UNGROUNDED CONFIGURATION

1. Symbols



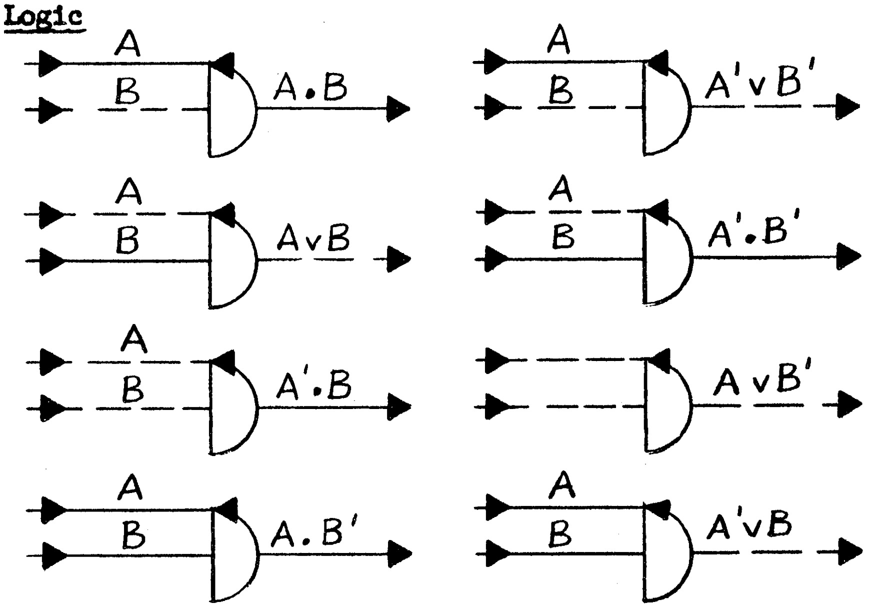


2. Operation

The output is negative if b is positive and e is negative.

The output is positive if b is negative or e is positive.

3. Logic


A
B
A.B
A
B
A'vB'
A
B
AvB
A
B
A'.B'
A
B
A'.B
AvB'
A
B
A.B'
A
B
A'vB