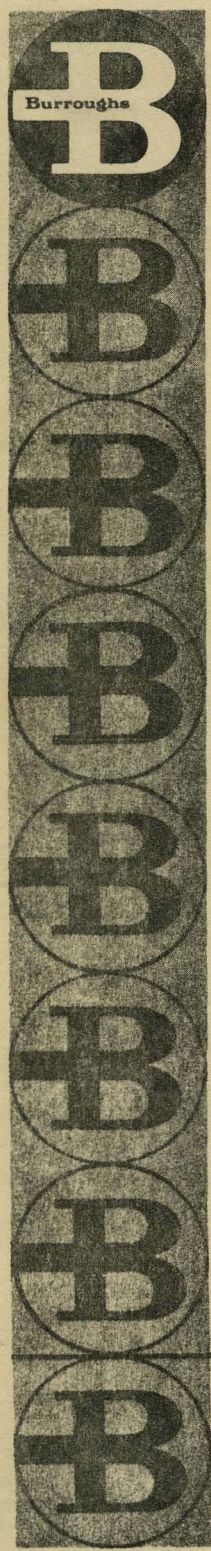


THE BURROUGHS D 825 MODULAR DATA PROCESSING SYSTEM

PROGRAMMING MANUAL

31 JANUARY 1962

Contract No. Nonr 3521 (00) (x)

SUBMITTED TO

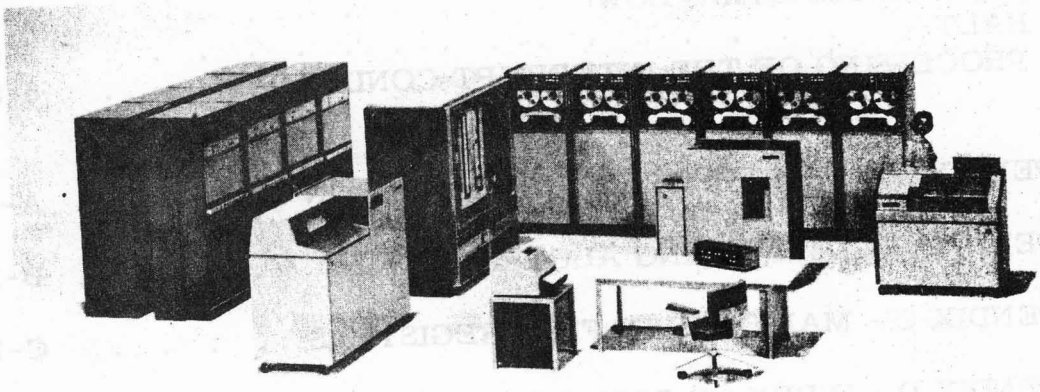
U.S. NAVAL RESEARCH LABORATORY
WASHINGTON D.C.

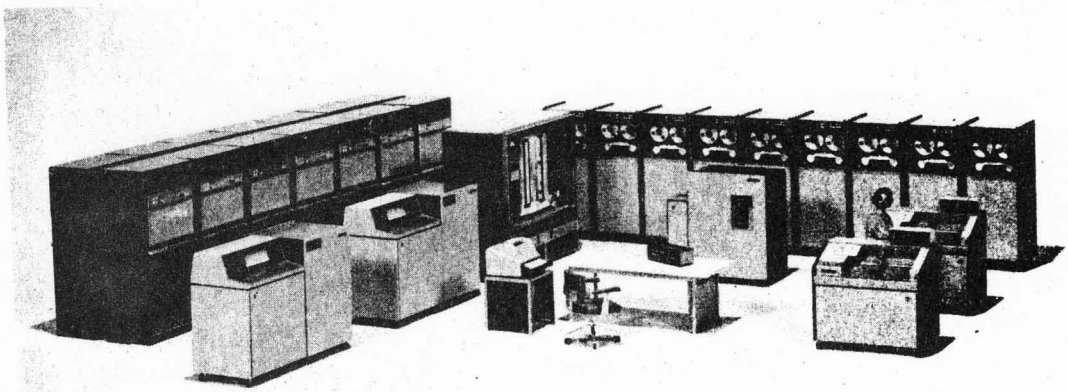
Burroughs Corporation

CONTENTS

CONTENTS (Cont'd)

CONTENTS (Cont'd)





SECTION 1

INTRODUCTION

This programming manual describes the machine language programming techniques used with the D825 Modular Data Processing System. This introduction includes system organization and operation to familiarize the programmer with the D825, and succeeding sections present the programming features of the D825, the structure of the instruction syllables, the operation codes, input/output programming, and the interrupt system. In addition, there are four appendices; Appendix A gives the D825 execution times, Appendix B gives the timing algorithms with an example, Appendix C gives the addressing structure for the thin film registers, and Appendix D is an index of operation codes.

SYSTEM ORGANIZATION

The Burroughs D825 Modular Processor is organized for a specific application in an appropriate complement of Computer Modules, Memory Modules, Input/Output (I/O) Control Modules, and a common exchange of I/O devices. Physically, a complete system, including power supplies, is housed in a number of standard cabinets, an operating console, various input/output equipments, and off-line test equipment. The general system organization is illustrated in figure 1-1.

The Computer Module arithmetic unit operates in parallel, but receives data from the switching interlock in serial-parallel form. A thin-film register storage and operand stack in each Computer Module operates at the 3 megacycle clock rate and greatly reduces the required accesses to Memory Modules. The command list of the computer includes binary fixed and floating point arithmetic instructions with the computer organization oriented toward efficient

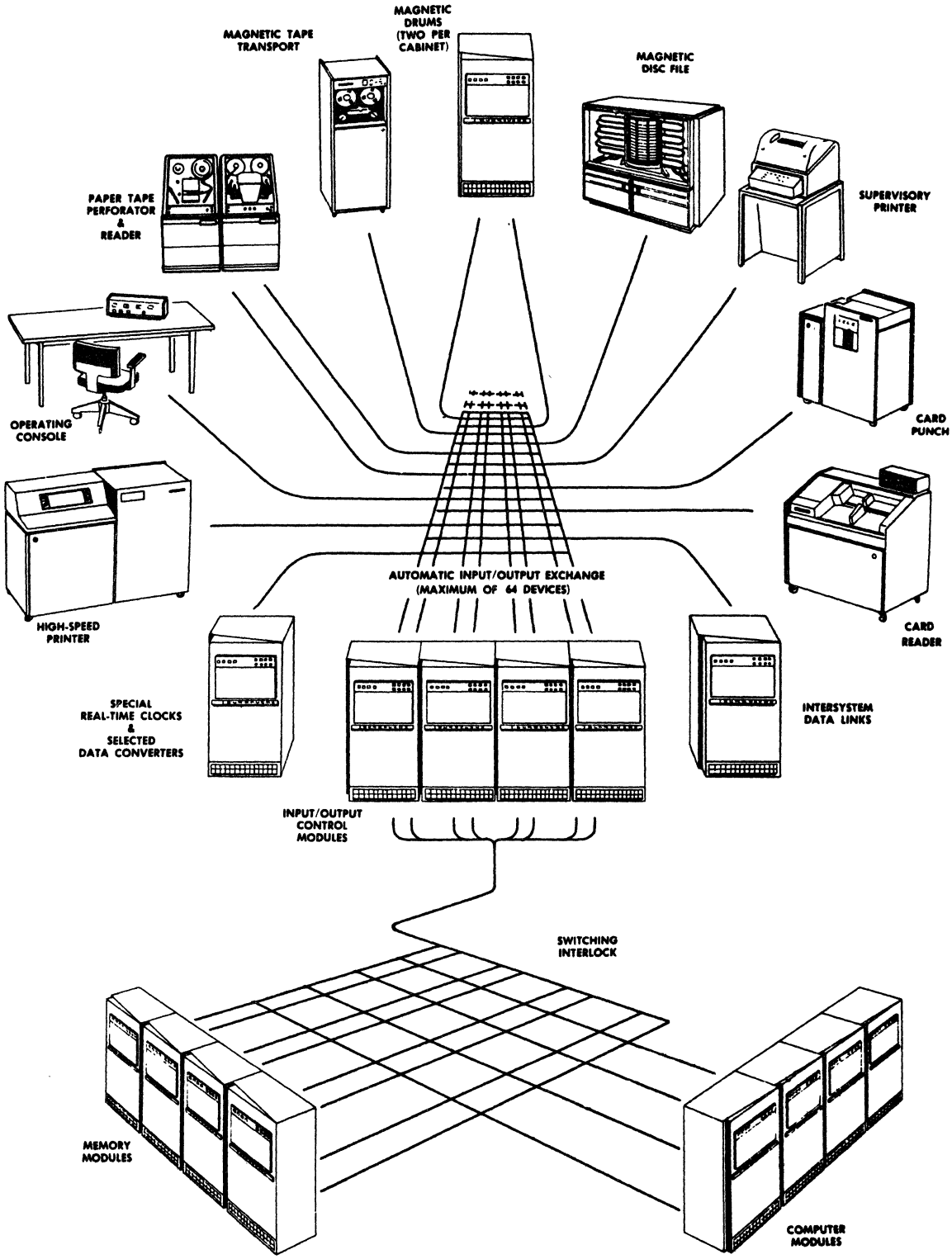


Figure 1-1. D825 System Organization

floating-point computation. The addressing structure of the computer has been designed to incorporate all of the power of a three-address machine. (Less than the maximum of three addresses can be processed with each instruction, with commensurate savings in instruction time and program storage.)

The Memory Module is a linear-select (word-organized), random-access, ferrite-core memory which was once available only in computers designed for extreme environments. Each module contains 4096 words of 48 bits plus parity, and a fully expanded system of 16 modules provides 65,536 words of memory.

I/O Control Modules, consist essentially of control and data manipulation registers and associated decoding and timing circuits. Each is capable of controlling any device of the I/O complement, and there can be as many simultaneous I/O operations as there are I/O Control Modules. The I/O exchange automatically connects control modules with specified I/O devices on command from computer modules. The console, effectively an I/O device, displays system status to the operator and permits him to effect inquiries and manual interrupts.

SYSTEM OPERATION

The D825 adapts instantly to real-time influence, to new programs, to changes in program priorities, and to manual or automatic interrupt signals. Its operational structure permits broad programming flexibility and efficient operation and program storage. Man/machine communication is comprehensive.

Parallel Operations

Each D825 Computer Module has exclusive use of a data transfer bus by which it can communicate, via the switching interlock, with any memory module in the system. The I/O Control Modules of an I/O exchange share a single bus, and two exchanges, each on a separate bus, are available as an option.

Memory may be used concurrently by all computer and I/O exchange buses. If two or more buses simultaneously address the same Memory Module, the switching interlock automatically resolves the

conflict according to priority and queues the lower priority items. One functional bus gains immediate access while the other is delayed only until completion of the first memory transfer.

Operational Structure

The D825 has been designed with a binary data word of sufficient length (49 bits including sign and parity) for almost all computing problems, and for really useful binary floating-point computation (36 bits of mantissa, including sign, and 12 bits of characteristic or exponent). This provides as much resolution for floating-point arithmetic as many large-scale computing systems offer in fixed-point arithmetic. The D825 alphanumeric data word contains eight characters.

Operands may be called from memory or from a four-position stack of operand registers within the thin-film storage of the computer. The results of operations can be stored in memory or in the operand stack for subsequent processing at the will of the programmer.

The operand stack is a device, extremely useful in arithmetic and processing operations, which reduces the number of references to main memory by holding partial or intermediate results of computation. The stack operates in two modes: normal and hold. The hold mode is useful for list manipulation and for repeated use of a number.

The first syllable of an instruction supplies the operation code and three address indicators. The address indicator provides choice between fetching the operand from the stack or the memory, and indicates whether the stack mode is normal or "hold" and whether the memory address is to be indexed or not. Address syllables or syllable strings follow the operation syllable for each memory accesses called out. Each memory address syllable contains an eleven-bit literal address and an indirect address bit. The literal address is added to the 16-bit base address register in order to refer to an area of memory known as the direct-address area.

The contents of the direct-address area location may be either an operand or another memory address. Indirect addressing of any desired number of levels is available by this technique.

Any or all of the three operand addresses which can be developed for each instruction may be modified by three of fifteen thin-film index registers. This capability, combined with the powerful indirect addressing capability of this system, provides immensely flexible address control. A full discussion of D825 programming features is presented in Section 2.

Modes of Operation

The D825 Modular Data Processing System has two modes of operation, normal and control. The control mode has a slightly larger order code than the normal mode, in that several special control instructions are necessary in addition to the normal instructions. The interrupt system provides the means of transposing operation from the normal mode to the control mode, either by computer instruction or by the occurrence of an event internal or external to the computer. When a computer is in the control mode, it can adjust its mask register to accept or ignore certain of these events.

Object programs are performed in the normal mode, and control programs, such as an Automatic Operating and Scheduling Program, are performed wholly, or in part, in the control mode. Control mode is always used for transmitting I/O descriptors.

Automatic Interrupt Capability

The interrupt system provides the facility for interrupting the "normal" data processing mode of operation of the computer system. It recognizes programmed and hardware-generated interrupt conditions caused by situations arising in the execution of a program; recognizes manually-initiated requests and automated external requests for communication with the computer system; and also recognizes equipment faults such as parity errors, illegal operations, and primary power failures.

In general, an interrupt condition causes transfer of control of the interrupted computer from the object program to an Automatic Operating and Scheduling Program (AOSP). Interpreting the interrupt condition at hand, the AOSP transfers control to the appropriate routine for handling the condition. When the interrupt condition has been satisfied, control is returned to the object program.

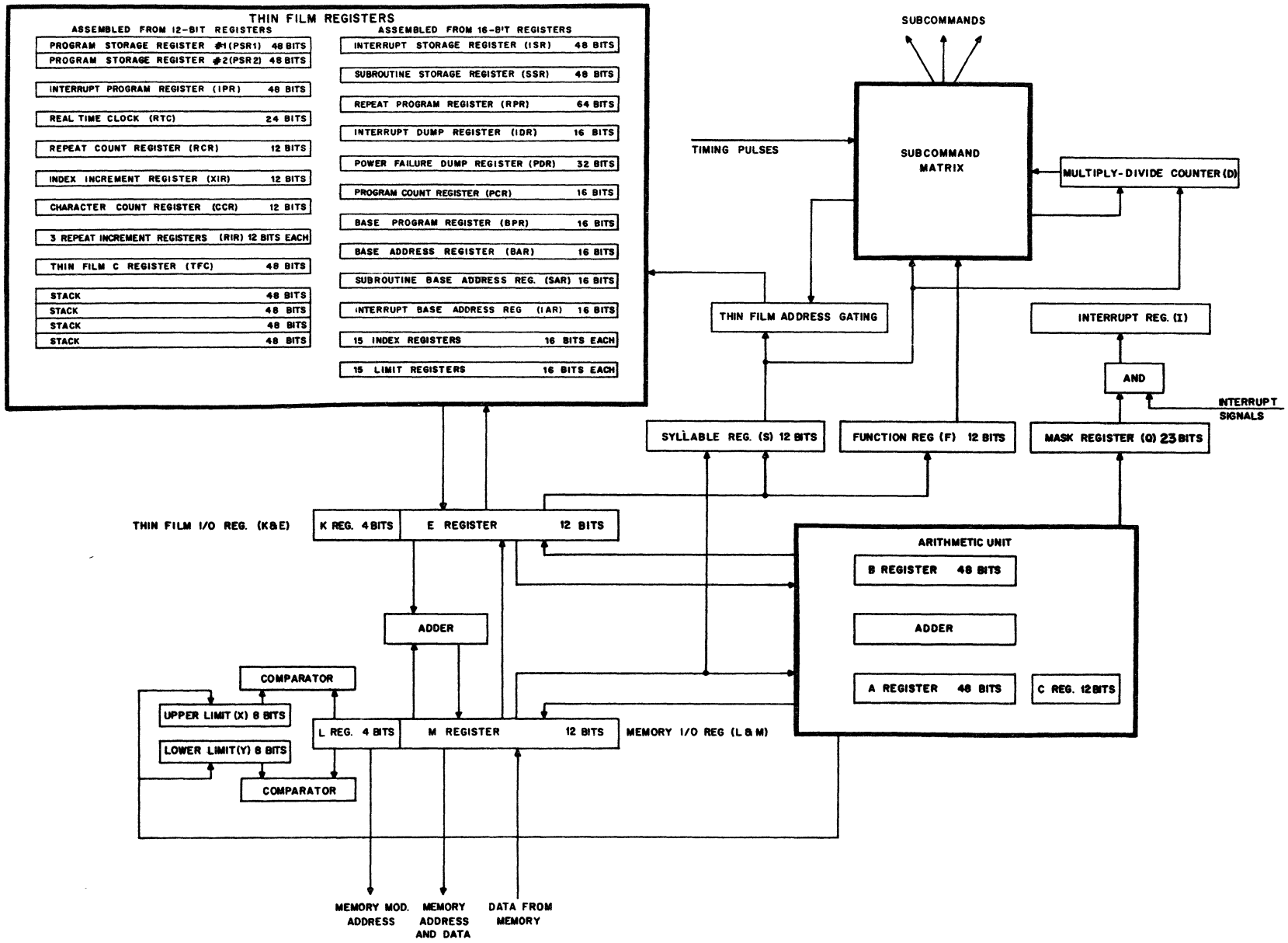


Figure 1-2. Computer Module Block Diagram

COMPUTER MODULES D825-1

The system can accommodate up to four Computer Modules. Each Computer Module consists of three functional areas. The first area, the arithmetic unit, is made up of three registers A, B, and C with associated controls. The second area is a set of 53 registers contained in a small thin-film magnetic storage. The third area is the control section which includes capability for indexing, address accumulation, indirect addressing, and the command and subcommand matrices. Figure 1-2 is a block diagram of the Computer Module showing all functional areas.

The memory I/O register (L and M) is a multipurpose register. To initiate a memory transfer, the memory address is transferred to the memory I/O register. The portion of this address that is the L register designates a specific Memory Module and is sent as dc levels to the switching interlock circuitry of the memory trunk. Address data for the Memory Module and information words entering the Computer Module from the memory module are transmitted through the M register 12 bits at a time.

The A register, B register and C register are the working arithmetic registers of the Computer Module. The A and B registers with associated adder circuitry perform the actual arithmetic operations. The A register is capable of shifting in optimum combinations of 12, 6, and 1 places to the right and iteratively 1 place to the left.

The function register (F) is a 12-bit register that holds the operator syllable being executed and provides the dc levels for driving the command and sub-command matrices.

The Multiply/Divide counter (D) controls the number of add or subtract cycles to be executed during multiply or divide operations. The counter also controls the number of shifts to be executed during an instruction.

The five operand registers include four operand stack registers which make up the thin-film operand stack, and the thin-film C (TFC) register which is used to store the least significant half of a double-length product and the remainder for division.

The two program-storage registers (PSR1 and PSR2) provide storage for eight instruction syllables and permit overlapped instruction fetch during long instructions.

The base address register (BAR) holds the base address of the data direct-address area. The base program register (BPR) holds the base address of the program address area. The program count register (PCR) holds the address of the last instruction (most recently fetched from memory) in the program-storage registers.

There are 15 index registers and 16 comparison limit registers. Any three of the index registers may be addressed by each index address syllable and used to modify each operand address. The index registers may be incremented, decremented, and compared in six different ways with the comparison limit registers. The index increment register (XIR) is used by the logic during the execution of the index increment instruction.

The contents of the real-time clock (RTC), a 24-bit register, is automatically read out and incremented once every 10 milliseconds. The real-time clock may be sampled and set by the program. An interrupt is initiated when the count overflows.

The character count register (CCR) is used by the character search instruction to indicate the character position last examined for the specified character.

When the repeat instruction is used, the repeat program register (RPR) provides storage for the four syllables of the program word being repeated; the repeat count register (RCR) contains the number of iterations yet to be performed; three repeat increment registers (RIR) contain the increments corresponding to the three addresses of the instruction being repeated.

The subroutine base address register (SAR) contains the base address of a list of subroutine addresses. When a subroutine is executed, the subroutine storage register (SSR) holds subroutine return information, i. e., the former contents of the BAR, BPR, and PCR.

The interrupt system registers provide storage for data in the operational registers in the event of an interrupt. The interrupt base address register (IAR) contains the base address of the interrupt routines; the contents of this thin film register are protected during the

normal operation mode. The interrupt storage register (ISR) holds interrupt return information, i. e. , the former contents of the BAR, BPR, and PCR. The interrupt program register (IPR) provides storage for the contents of the presently addressed PSR, during interrupt. The interrupt dump register (IDR) holds the PSR and repeat controls for interrupt return. The power failure dump register (PDR) holds the contents of the control flip-flops and the flip-flop interrupt register in the event of a power failure.

There is an over-under voltage detector which will detect and signal excursions of primary power beyond fixed voltage limits. The out-of-tolerance signal causes the computer module to store sufficient information to restart the program without loss of data. Provision is made for automatic program restart by automatically reloading stored data back into the flip-flop registers. The power supplies themselves have a sufficiently long time constant to protect the hardware, program, and data from all primary power transients and failures, and allows continuation of the program when stable primary power is restored.

The interrupt system handles interrupts arising from such conditions as arithmetic overflow, running down of the real-time clock, illegal orders, parity errors, external input-output requests, and I/O result situations, etc. Each Computer Module has an interrupt register which can be set through the interrupt mask register. When a particular condition has set a one at some bit position in the interrupt register, a program interrupt occurs. This interrupt stops the program being executed, stores sufficient registers to allow continuation of the interrupted program at a later time, and transfers control to a routine in an AOSP to service the interrupt.

SECTION 2

THE FEATURES OF D825 PROGRAMMING

The flexibility of D825 machine language programming is a result of the following programming features which are discussed in this section.

- Variable-Length Instructions
- A Four Level Operand Stack
- Single and Multiple Indexing
- Relative Addressing and Indirect Addressing
- Field-Defined Instructions
- Subroutine Control
- A Comprehensive Interrupt System

VARIABLE-LENGTH INSTRUCTIONS

The combination of two unique features - three address programming and variable length instructions - in the D825 yields new efficiency and versatility in programming. The three address capability takes advantage of the fact that many fundamental operations involve three factors, and therefore may need three storage locations: one each for the two operands and the result. The typical example is "add A and B; store the result in C". Single address programming, with each of the three factors using a separate storage location, requires three instructions to complete the operation: (1) clear and add A, (2) add B, and (3) store in C. There is a large group of such operations readily adaptable to three address programming; but there is

an equally large group of operations that require fewer than three addresses. If all instructions were to be of exactly three addresses, the program would be wasteful, and nothing would be gained by three address programming over conventional single address programming. For this reason, variable length instructions are allowed.

The program in memory consists of blocks of 48-bit program words; program words, in turn, consist of four units of information each. These four 12-bit units are called "syllables". A syllable can be of a variety of different formats, and these special formats are determined by the information that a specific instruction needs, such as an address or an instruction variant. In fact, the operation code of the instruction itself occupies one of the syllables, known as the operator syllable. The formats and uses of the different syllables will be discussed in Section 3 in more detail.

The variability of the system comes from the fact that the instructions are considered as "strings of syllables", and these strings can vary in length from one to seven syllables per instruction. Also, an instruction can begin at any syllable location in the memory word, and end with this or any succeeding syllable of the memory word, or can continue on into as many as two succeeding memory words.

Although a particular instruction requires certain addresses, the top of the operand stack may be implied as one (or more) of the addresses and therefore needs no syllable(s) to identify it. On the other hand, certain addresses may be indexed and therefore need two syllables to identify the address. Whatever the case may be it is indicated by three address tags of two bits each which appear in the operator syllable, along with the operation code of the instruction.

There are two important things to remember about syllable strings. (1) The first syllable of an instruction is always an operator syllable, and will identify any syllables that may follow. (2) Individual syllables are not addressable, although program words are. For this reason a branch refers to the first syllable of a memory word, and will execute that particular syllable as an operator syllable (i.e., the beginning of a syllable string). Therefore when a branch is planned, the operator syllable of the instruction to be executed must be the first syllable of the memory word.

OPERAND STACK

One of the features of the D825 is the fast access operand stack. Operands which are used again and again can be kept in the stack and addressed in a shorter time than would be required to get an operand out of memory. However, only the top of the stack is accessible at a given instant. The programmer must keep aware of what values are in each level of the stack and which level is currently accessible. The stack is effectively a four-word circular memory with an addressing counter. One of the four words is always being "pointed at" or is under the "read head". This circular concept and the pointer "read head" at the top of the stack are used in the examples shown in figure 2-1.

Whenever access is made to the stack, there is an option of either holding the stack or circulating the stack one step. Normal operation is to step the stack with each stack reference. The stepping operation follows all fetches from the stack and precedes a deposit made in the stack. As shown in figure 2-1 this stepping action is counterclockwise following a fetch and clockwise preceding a deposit.

In the D825 operation syllable, address tag values 00 and 01 designate the stack as the intended operand source. Codes 10 and 11 refer to the memory. Code 00 designates normal stepping of the stack wheel and 01 designates that the stack be held (not stepped).

Figure 2-1 gives examples showing the resulting condition of the stack after all the possible combinations of stack operations. Examples 1 through 9 include one or more memory operands with the results stored in the stack or memory. In examples 10 through 15, both operands are from the stack and the results are stored in the stack. In examples 16, 17, and 18, both operands are taken from the stack but the results are stored in the memory. The initial condition for all the examples is shown in the upper left-hand corner.

INDEXING

D825 indexing implements two processes: (1) the modification of an address by adding the contents of one, two or three index registers to the address in order to obtain the effective address, and (2) the modification of the contents of an index register, comparing the

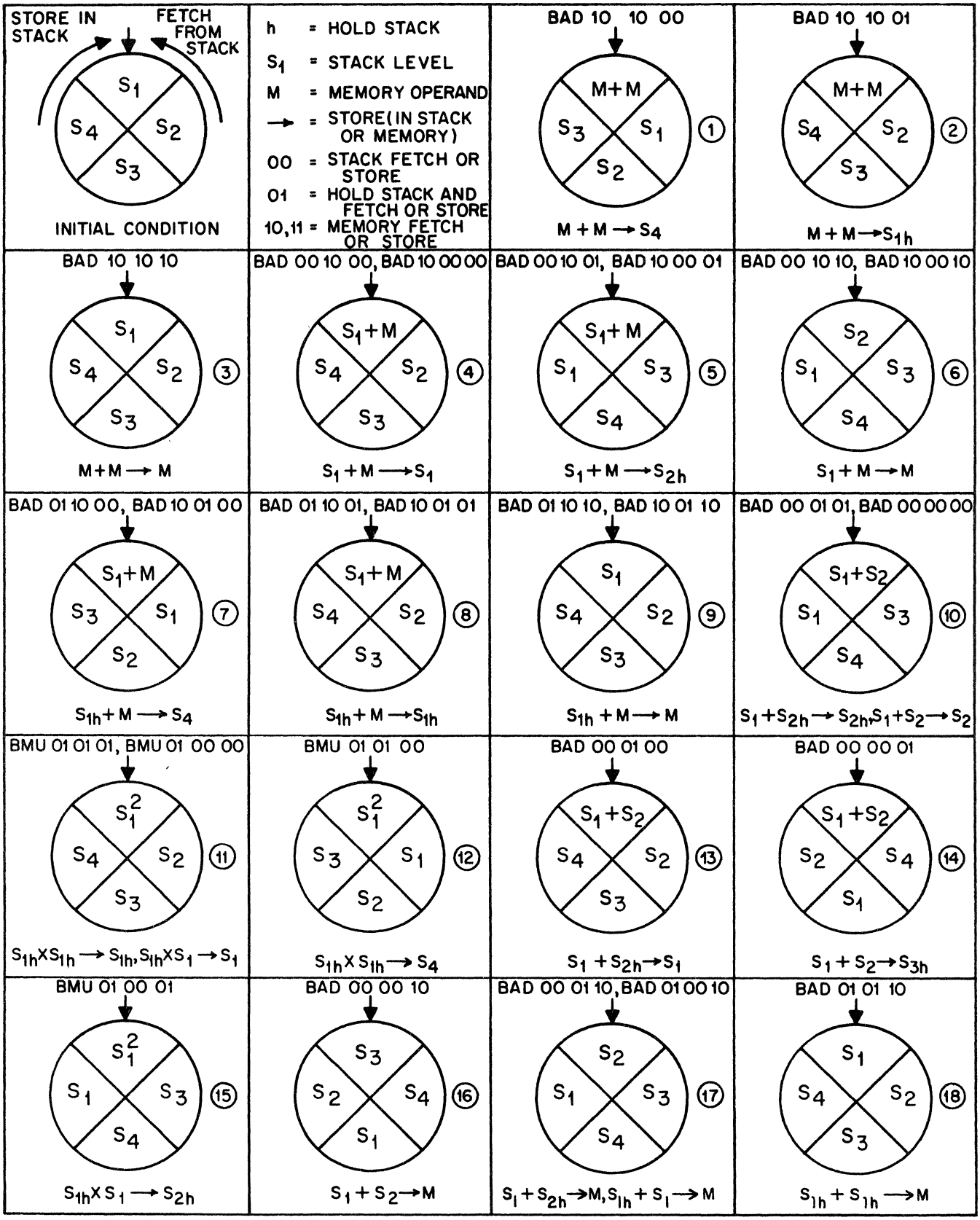


Figure 2-1. Operation of the D825 Operand Stack

modified contents of the index register with the contents of a specific limit register, and, depending on the results of the comparison, transferring program control to the appropriate location. The first process is performed automatically within an instruction, if desired, and can involve any or all of the three possible addresses of the instruction. The second process is performed by one specific instruction.

A special feature of the D825 is "multiple indexing", i.e., adding the contents of up to three index registers to the address. This is especially valuable in multicoordinate systems where it is necessary to maintain more than one separate index in computing the addresses of operands in an array.

Other features of the D825 indexing system are: (1) An ample quantity of index and limit registers (15 of each) is provided, and these are located in thin film, thus making them independent of program or data locations in memory. (2) The registers are 16 bits each, and therefore can contain an absolute memory address, i.e., not relative to any base address. (3) The contents of index and limit registers are unsigned quantities, but if a "negative address modifier" is desired, the two's-complement "positive address modifier" will serve the same purpose.

RELATIVE ADDRESSING AND INDIRECT ADDRESSING

The address syllable contains only 12 bits, in order that programs may be condensed. Sixteen bits, however, are required to specify an absolute machine address. Several techniques are available to the programmer to allow flexibility in addressing without the full sixteen bits.

Normally the program is contained in a block of contiguous memory locations, and programs for the D825 will in general be loaded this way. The beginning location of the block may be any location in memory. The address of this location is placed and retained in a thin film register called the BPR (Base Program Register). Any transfer of control in the program adds the literal specified in the branch syllable to the BPR and places this sum in the PCR (Program Count Register), another thin film register. In this way, transfers (branches) may be specified to locations relative to the beginning of the program block, and the program may be loaded and executed in any position in memory with no modifications of branch addresses and no wasted indexing.

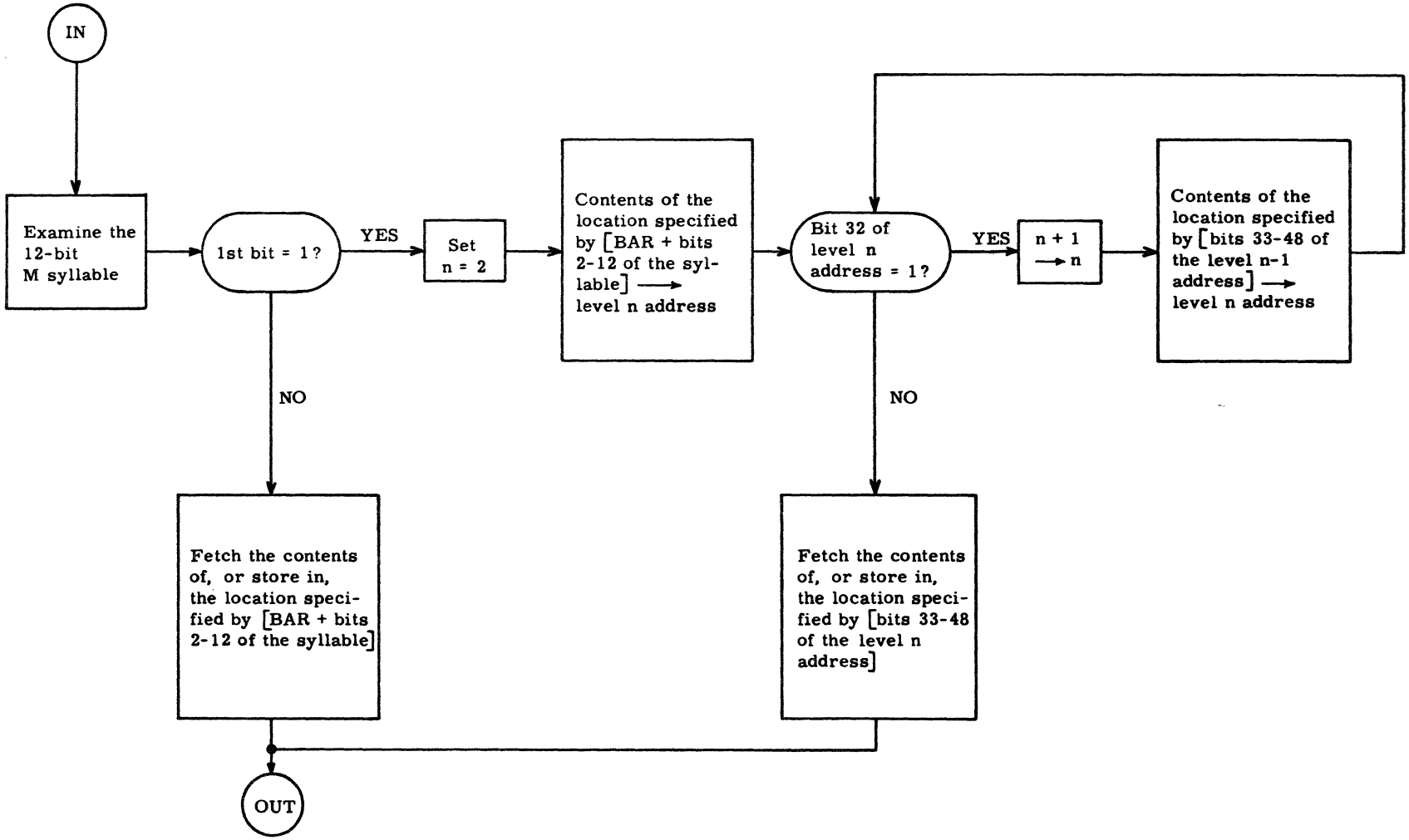


Figure 2-2. Flow Chart of Indirect Addressing

In the same way that instructions are contained in blocks, the constants and variable data may be placed together in a block of contiguous memory locations. This is a restriction of memory usage which is of considerable importance in being able to allocate any available space in memory to a program to be run. Eleven of the twelve memory syllable bits are used to specify the location of an operand relative to the beginning of the data block. The first location of the data block is placed in the thin film BAR (Base Address Register). When an operand memory reference is made, the eleven least significant bits of the memory syllable are added to the BAR to form the absolute address.

While only the first 2,048 words (direct address area) immediately following the location specified by the BAR are directly addressable, this restriction is trival due to the flexible indexing provided and the ability to indirect address beyond this area.

The most significant bit of the memory address syllable indicates, if it is a one, that the address is an indirect one. The 16 least significant bits of the location specified is then the absolute address of the operand or another indirect address. If the 17th-least significant bit of the location specified in the direct address area is a one, it indicates that the indirect address chain is to be continued; that is, the location specified in the direct address area is, in turn, an indirect address.

This indirect address chaining is provided for any number of levels. If indexing is specified for the operand in the instruction word, it is performed at the last level (i.e., at the completion of all indirect addressing). The flow chart (figure 2-2) describes the indirect addressing chain.

Indirect addressing can also be used to facilitate computer control of certain areas of the program by using a "snag bit." If the 18th-least significant bit of any level of addressing after the first is a "ONE"; an interrupt register bit will be set.

FIELD-DEFINED INSTRUCTIONS

A set of field-defined instructions is provided to facilitate code conversion and multiple use of a memory location. A field is defined to be a set of physically adjacent 6-bit characters in fixed locations in a

word. The ability to perform simple arithmetic, manipulation, and comparison on such fields is available.

SUBROUTINE CONTROL

The ability to transfer control to a remote subroutine, and to return, is provided. This transfer of control enters a subroutine indirectly through a list of all subroutine addresses, and the location of this list is specified by the SAR (Subroutine Base Address Register). This feature eases the bookkeeping required to relocate a subroutine. An index register is set by the computer logic to allow the addressing of constants in the subroutine area. The ability to locate the working-storage area of subroutines in the calling routine's direct address area is provided by the instruction logic. It is possible to have several computers independently and simultaneously execute a subroutine through the use of independent direct address areas. The facility of handling nested and recursive subroutines is considerable. The subroutine jump and return instructions are slightly more complex than those provided in other types of machines so that their use in a multiple computing system, with basic relative addressing, is enhanced.

INTERRUPT SYSTEM

A complex and comprehensive interrupt system is provided to facilitate the use of the computer in a control system. If an external request occurs, a computer will be interrupted at the completion of its current operation. The ability to automatically resume operation is provided. Computers may be interrupted by external requests, such as for I/O servicing, by internally generated signals such as the running-down of the clock, and by other computers. A computer which is interrupted is automatically placed in the control mode, in which it can execute a special class of instructions which are not available in the normal mode. The restrictions in the normal mode prevent changes to an AOSP by overwriting memory unexpectedly, modification of the interrupt controls, and initiating input/output for which the operating system would be unaware. These two modes of operation allow the operating system program, executed in the control mode, to maintain its monitoring of all other programs.

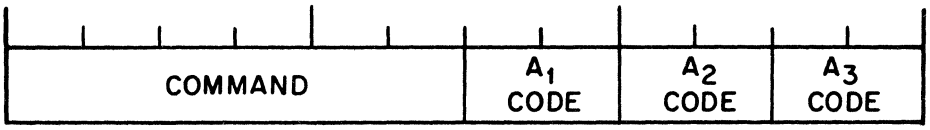
A special type of interrupt occurs if power fails. All registers and controls are stored in non-volatile storage, and the system operation may resume when power is restored.

SECTION 3

THE STRUCTURE OF PROGRAM SYLLABLES

The structure of the 18 syllable types used in machine language programming of the D825 is presented in this section. The operator syllable, the index syllable and basic address syllables (memory and branch addresses) are given first, followed by the special-use syllables. The letter symbols preceding the syllable headings are used in Section 4, D825 Instructions, to represent the syllable structures given in Section 3. All syllables are 12 bits in length.

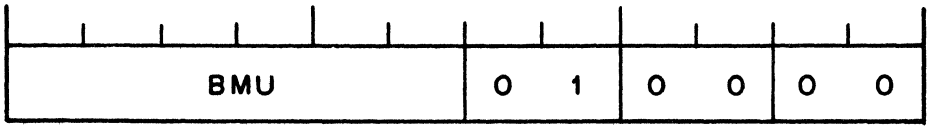
O-OPERATOR SYLLABLE



The operator syllable is the only one required for every instruction. It identifies any syllables that may follow and any stack usage for the instruction. This type of syllable is not indexable. The A_1, A_2, and A_3 codes identify the 3 possible "addresses" of the instruction as shown on the following page.

Code	Definition
00	Stack (or no address)
01	Hold stack
10	An unindexed syllable follows
11	An indexed syllable follows (index syllable followed by any other syllable, to which it is applied)

In the following example of the use of the operator syllable, the number contained in the top of the stack is squared, and the results are stored in the stack.

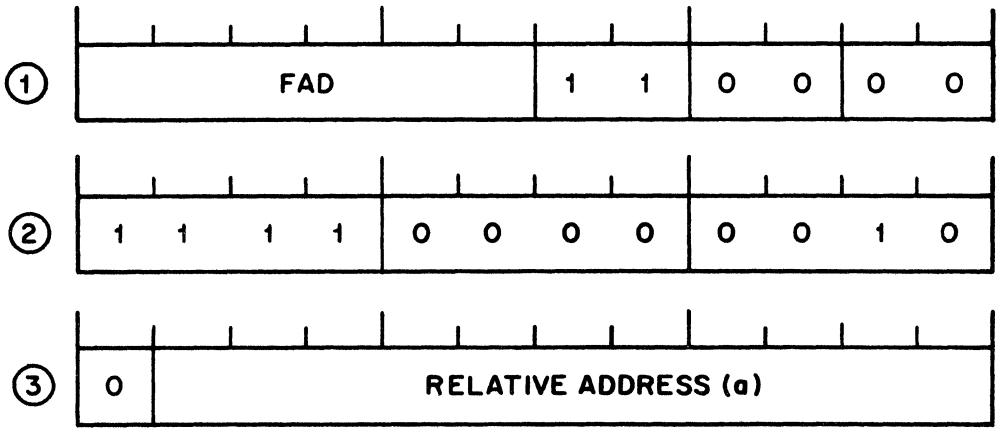


X-INDEX SYLLABLE

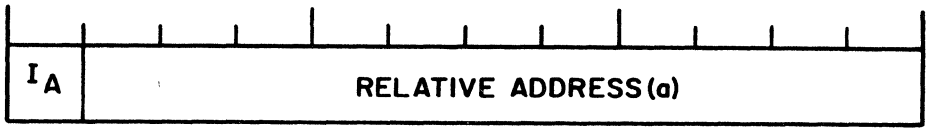
The index syllable can be used with any other syllable except the index syllable itself, the operator syllable, or any syllable used in a repeated instruction (see RPT instruction). In all cases it is optional. The contents of up to 3 index registers can be applied to (added to the contents of) one syllable. If an index register contains the 2's complement of a number, use of this index register effectively subtracts the number from the contents of the syllable to which it is applied. When a syllable is indexed it is immediately preceded by its index syllable. An index register address equal to zero means no index. In indexed

indirect-addressing, the contents of the index registers are applied to the last-level address only.

In the following example, the index syllable and memory address ayllable follow the operator syllable. If index register 15 (1111) contains b and index register 2 (0010) contains c, then the contents of the memory location specified by [a + b + c + BAR] is added to the top of the stack (floating-point in this case) and the results are stored in the stack.



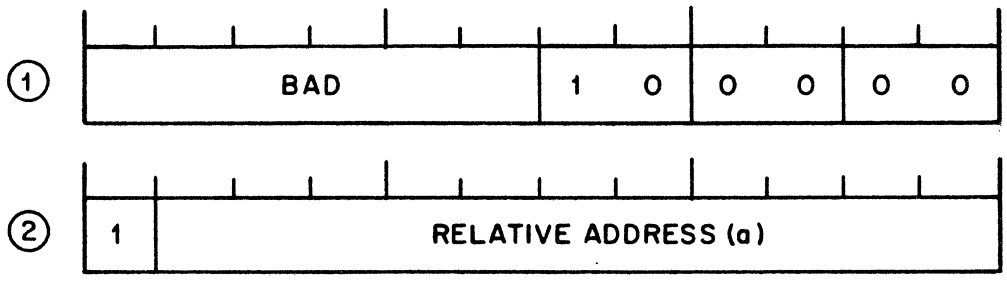
M-MEMORY ADDRESS SYLLABLE



The memory address syllable is used to address data words in memory. The relative address (a) is added to the contents of the base address register (BAR) to obtain the effective memory address. If I_A = 1 this address is indirect, and the location specified by [a + BAR] then contains the next level address (the 16 least significant bits, with the 17th bit as an I_A bit). Note that each level of addressing, after the first, contains an absolute address, and therefore is not added to the BAR.

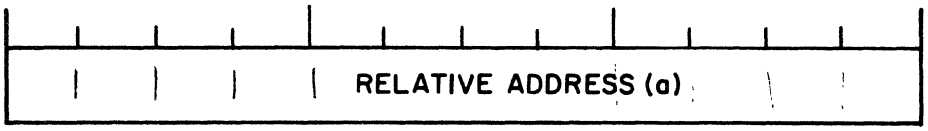
In this manual, an underlined memory symbol, M, represents the store address of the instruction, if any; memory symbols surrounded by parentheses, (M), represent optional syllables. If a memory syllable is omitted, the stack is the implied address; in any case, the A_1, A_2, and A_3 codes of the operator syllable define the address(es). For example, 0 (M) (M) (M) means operate on the contents of the locations specified by the first two addresses and store the result in the location specified by the third address. All three memory syllables are optional (i. e., can be replaced by stack references).

Since the first bit of the memory address syllable is a ONE in the example shown below, the memory address specified by [a + BAR] is indirect and contains the second level address. If a ZERO appears in the I_A bit of the second level address, the contents of the location specified by the second level address is added to the top of the stack and the results are stored in the stack. If the I_A bit is equal to ONE, this address also is indirect and contains the next level address; the indirect addressing chain will continue until an I_A bit is equal to ZERO.



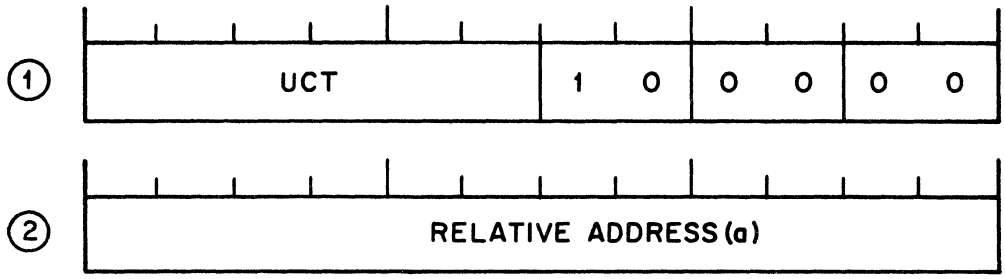
Second level (and subsequent level) absolute addresses are stored in bits 33 through 48 of the 48-bit memory word, and bit 32 is the I_A bit.

B-BRANCH ADDRESS SYLLABLE



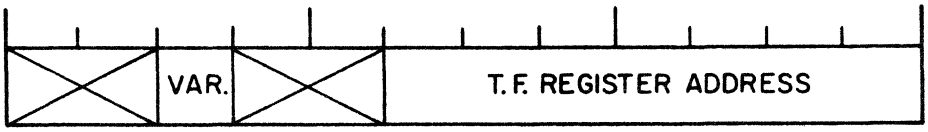
The branch address syllable is used to address program words in memory. The contents of the relative address (a) is added to the base program register (BPR) to obtain the effective branch address.

The following example is an unconditional transfer to the program word stored in the memory location specified by [a + BPR].



In all cases where a branch is used, the first syllable of the 48-bit memory location thus addressed must be the operator syllable of the next instruction to be performed. When a branch is made, the program count register (PCR) is automatically loaded with the address of the branch.

T-THIN FILM ADDRESS SYLLABLE

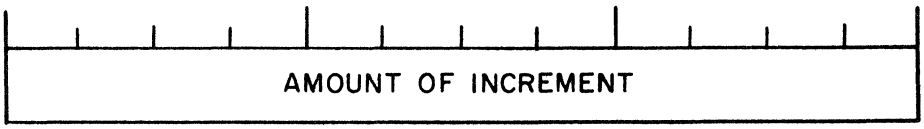


The thin film address syllable is used for addressing all thin film registers, and applies only to the STF and LTF instructions described in Section 4.

Iv-INDEX INCREMENT VARIANT SYLLABLE

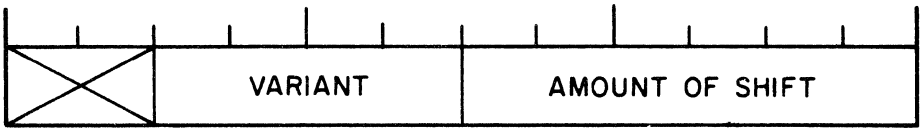
This syllable is used only with the XLC (index limit compare) instruction discussed in Section 4.

Ia-INDEX INCREMENT AMOUNT SYLLABLE



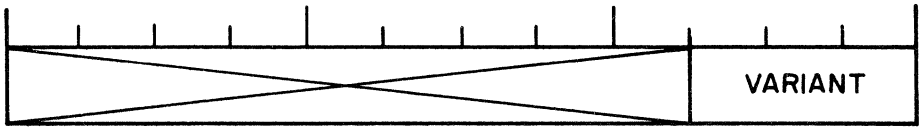
The index increment amount syllable is also used only with the XLC instruction and contains the amount to be added to, or subtracted from, the index register.

S-SHIFT SYLLABLE



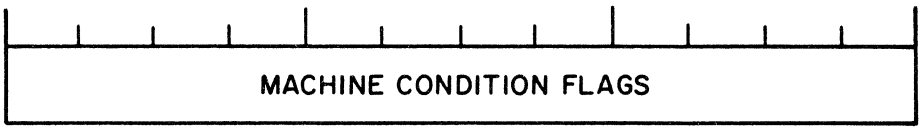
The shift syllable gives the type of shift (variant) and the amount of shift, and is used only with the SHF instruction discussed in Section 4.

Vt-TRANSMIT VARIANT SYLLABLE



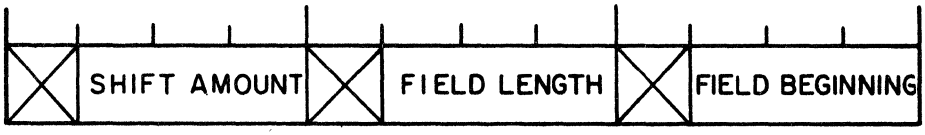
This syllable is used only with the TRM instruction for sign modification and rounding.

L-LOGICAL MACHINE CONDITION SYLLABLE



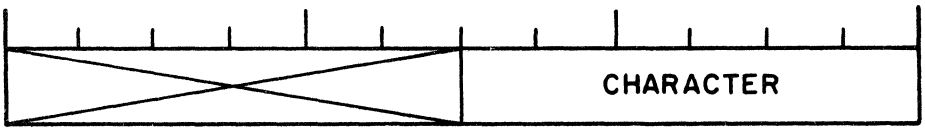
The logical machine condition syllable is used with the branch on condition (BRC) instruction to specify the conditions under which the branch is made.

F-FIELD DEFINITION SYLLABLE



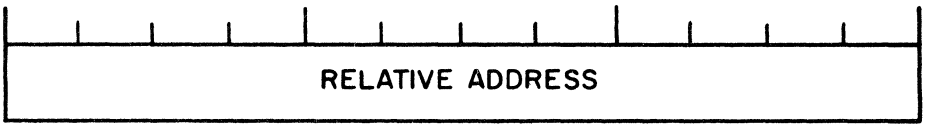
The field definition syllable is used with field-defined instructions as described in Section 4.

C-CHARACTER SYLLABLE



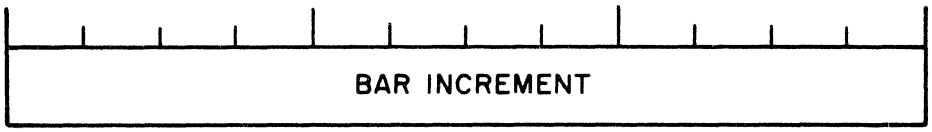
The six-bit character of the character syllable is the actual character being searched for in the character search instruction, CSE.

Ja-SUBROUTINE JUMP ADDRESS SYLLABLE



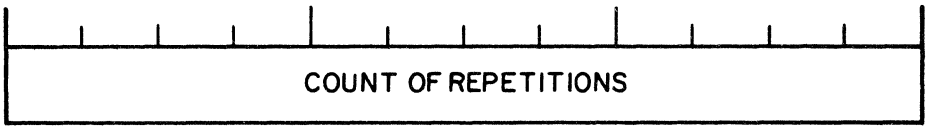
The relative address contained in this syllable is added to the SAR to obtain the effective address of the start of the subroutine in the SRJ instruction.

Ji-SUBROUTINE JUMP INCREMENT SYLLABLE



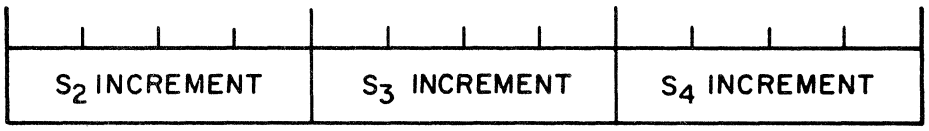
This syllable is used with the SRJ instruction to increment the base address register (BAR) when a subroutine jump is made. Both the Ja and Ji syllables are required for each subroutine jump instruction.

Rc-REPEAT COUNT SYLLABLE



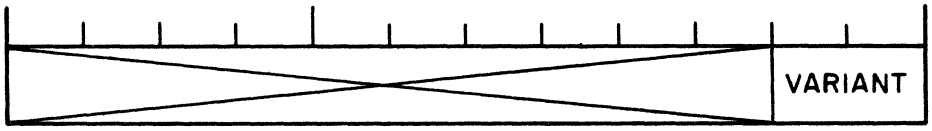
The repeat count syllable contains the number of repetitions to be performed by the repeat instruction, RPT.

Ri-REPEAT INCREMENT SYLLABLE



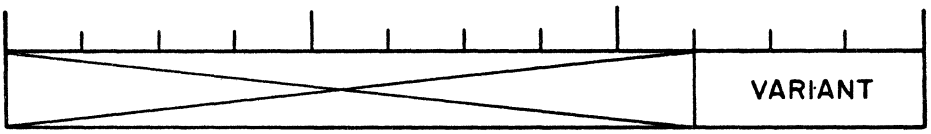
The repeat increment syllable is also used with the RPT instruction and gives the amounts by which the 2nd, 3rd, and 4th syllables are to be incremented with each repetition.

IO-I/O SYLLABLE



The IO syllable is used with the transmit input/output instruction, TIO, to select the I/O bus and to control I/O descriptors.

Vs-SPECIAL REGISTER AND COMPUTER INTERRUPT VARIANT SYLLABLE



This syllable is used only with the LSR instruction to designate the Computer Module to be interrupted, or for loading special registers.

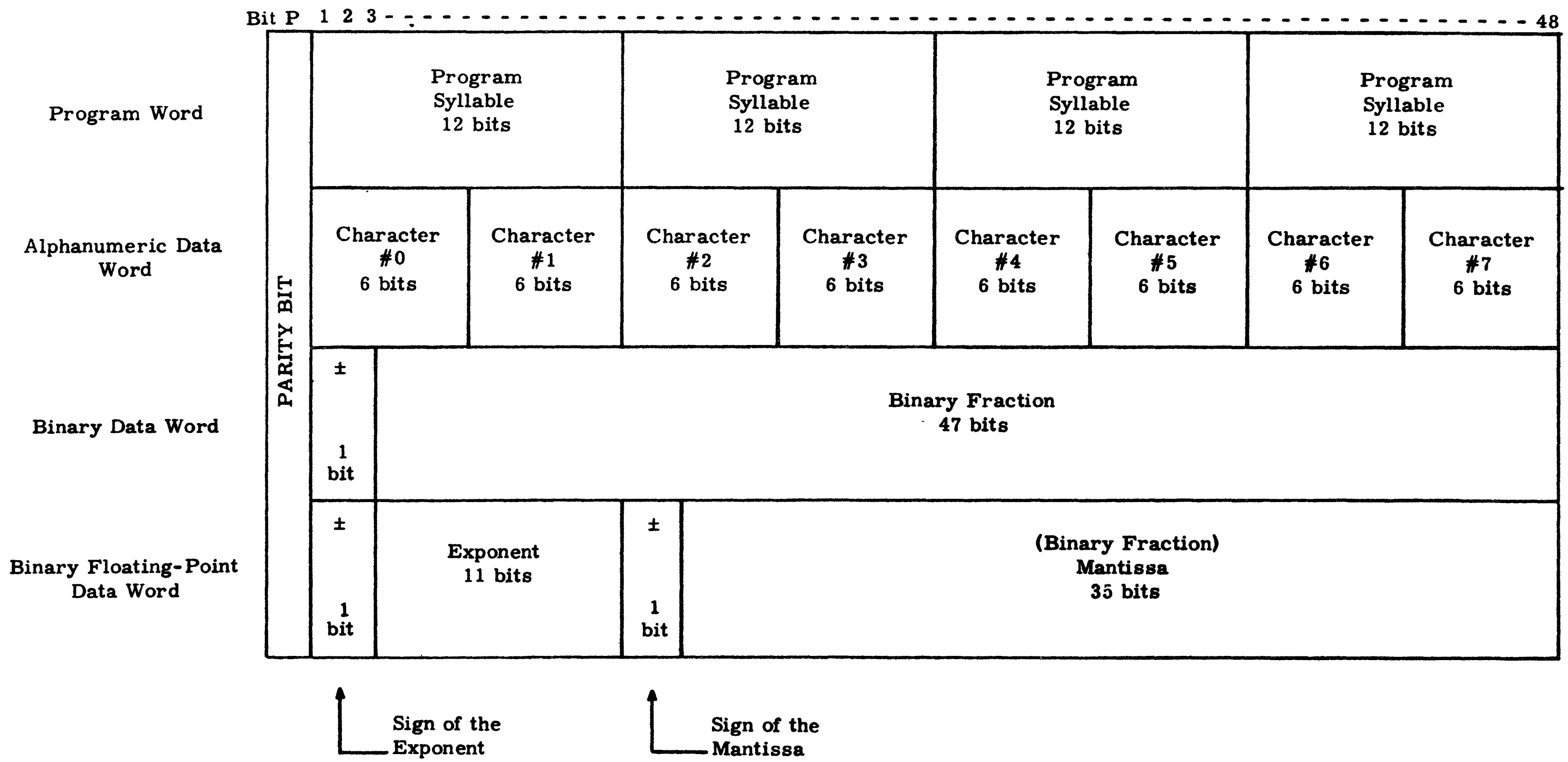
SECTION 4

D825 INSTRUCTIONS

This section contains the D825 instructions and the rules for their use. They are arranged approximately in the order of importance and usage. Each instruction heading includes mnemonic code, instruction name, and octal code on the left, and the syllable layout on the right. Appendix D is an alphabetical index to the instructions in this section. D825 word structure is given in figure 4-1 and a glossary of symbols is given below.

Symbol Glossary

A_1, A_2, A_3	1st, 2nd, and 3rd addresses
(A_N)	The contents of the address A_N
$\longrightarrow$	Store
, by A_N,	"as indicated by address A_N"
a:b	If a is true then branch to b
s	Stack Counter
.	Logical AND
V	Logical OR
$\textcircled{V}$	Logical EXCLUSIVE OR
$(\bar{A}_N)$	Logical (one's) COMPLEMENT the contents of address A_N



The conventions used in the above data words are:

"1" = minus
"0" = plus

Numbers are represented in signed magnitude form.

Figure 4-1. D825 Word Structure

FIXED-POINT ARITHMETIC INSTRUCTIONS

BAD - Binary Add - 65	0	(M) (M) (M) $(A_1) + (A_2) \rightarrow \underline{A}_3$

Algebraically add the contents of the location specified by A_1 to the contents of the location specified by A_2; store the result in the location specified by A_3.

NOTE 1: Overflow causes the program overflow flip-flop to be set, and in any event, the sum will be stored in the location specified by A_3. In the case of overflow the correct result (times 2^{-1}) can be obtained by program. Subtract out the end-around carry caused by over-flow, shift right 1, and add in the bit lost by overflow.

NOTE 2: If the result of an addition is zero, the sign is positive.

BSU - Binary Subtract - 64	0	(M) (M) (M) $(A_1) - (A_2) \rightarrow \underline{A}_3$

Algebraically subtract the contents of the location specified by A_2 from the contents of the location specified by A_1; store the result in the location specified by A_3.

NOTE 1: Overflow causes the program overflow flip-flop to be set; in any event, the difference will be stored in the location specified by A_3. In the case of overflow the correct result (times 2^{-1}) can be obtained by program. Subtract out the end-around carry caused by over-flow, shift right 1, and add in the bit 1 lost by overflow.

NOTE 2: If the result of the subtraction is zero, the sign is positive.

BMU - Binary Multiply - 61	0	(M) (M) (M) $(A_1) \times (A_2) \rightarrow \underline{A}_3$

Multiply the contents of the location specified by A_1 by the contents of the location specified by A_2; store the 47 high-order bits of the double-precision product, with sign, in the location specified by A_3, and leave the 47 low-order bits of the double-precision product, with sign, in the TFC register.

NOTE 1: If the result (the contents of the location specified by A_3) of a multiplication is zero, the sign is positive. The TFC register, however, may contain a negative zero.

BDV - Binary Divide - 60	0	(M) (M) (M) $(A_1) \div (A_2) \rightarrow A_3$

Divide the contents of the location specified by A_1 by the contents of the location specified by A_2; store the 47-bit quotient with sign in the location specified by A_3, and store the 47-bit remainder with sign in the TFC register.

NOTE 1: The remainder takes the sign of the dividend.

NOTE 2: At the end of the division process, if the absolute value of the divisor was not greater than the absolute value of the dividend, the program overflow flip-flop will be set, indicating quotient overflow.

NOTE 3: If the quotient is zero, the sign is positive. The TFC register, however, may contain a negative zero.

FLOATING-POINT ARITHMETIC INSTRUCTIONS

CBF - Convert Binary to Floating-Point - 25	0	(M) (M) $(A_1) \rightarrow A_2$, floating

Convert the contents of the location specified by A_1 from binary to floating-point format, and store the result in the location specified by A_2.

NOTE 1: If the contents of the location specified by A_1 is zero, the operation is immediately terminated by inserting floating-point zero in the location specified by A_2. Floating-point zero is the smallest possible positive number, 0×2^{-2047}, and appears as a negative exponent of all ones with a positive mantissa of all zeroes.

NOTE 2: Otherwise, the contents of the location specified by A_1 is treated as a 48-bit signed fractional number with a zero exponent; the exponent is decreased as the number is normalized, and after normalization is complete, the 35-bit mantissa is stored, together with the resulting exponent, in floating-point format. Floating-point format is defined, from left to right, as:

- sign of the exponent
- 11-bit exponent
- sign of the mantissa
- 35-bit mantissa

FAD - Floating Add - 67	0	(M)	(M)	(M)
	Floating,	(A_1)	+ (A_2)	$\rightarrow A_3$

Add the contents of the location specified by A_1 to the contents of the location specified by A_2; store the normalized floating-point sum in the location specified by A_3. Both operands are assumed to be in floating-point format.

NOTE 1: After the addition is performed, mantissa overflow, if it occurs, is corrected, as is the exponent. If exponent overflow then occurs, the operation is terminated and the program overflow flip-flop is set. In this case the correct mantissa, with the overflow exponent (+.0———01), is inserted in the location specified by A_3.

NOTE 2: If the resulting mantissa (without sign) is zero, floating-point zero is inserted in the location specified by A_3, and the program under-flow flip-flop is set.

NOTE 3: If the resulting mantissa is not equal to zero, normalization is performed until completed except that normalization is not performed in the case where the signs of the exponents of the two operands are different and the absolute value of the sum of the two exponents is greater than 2047. The exponent is then corrected; if exponent underflow should occur, the program underflow flip-flop is set and floating-point zero is inserted in the location specified by A_3.

NOTE 4: Floating-point zero is the smallest possible positive number, or 0×2^{-2047}, and appears as a negative exponent of all ones with a positive mantissa of all zeros.

FSU - Floating Subtract - 66	0	(M)	(M)	(M)
	Floating,	(A_1)	- (A_2)	$\rightarrow A_3$

Subtract the contents of the location specified by A_2 from the contents of the location specified by A_1; store the normalized floating-point difference in the location specified by A_3. Both operands are assumed to be in floating-point format.

NOTE 1: After the subtraction is performed, mantissa overflow, if it occurs, is corrected, as is the exponent. If exponent overflow then occurs, the operation is terminated and the program overflow flip-flop is set. In this case the correct mantissa, with the overflow exponent (+.0———01), is inserted in the location specified by A_3.

NOTE 2: If the resulting mantissa (without sign) is zero, floating-point zero is inserted in the location specified by A_3, and and the program underflow flip-flop is set.

NOTE 3: If the resulting mantissa is not equal to zero, normalization is performed until completed except that normalization is not performed in the case where the signs of the exponents of the two operands are different and the absolute value of the sum of the two exponents is greater than 2047. The exponent is then corrected; if exponent underflow should occur, the program underflow flip-flop is set and floating-point zero is inserted in the location specified by A_3.

NOTE 4: Floating-point zero is the smallest possible positive number, or 0×2^{-2047}, and appears as a negative exponent of all ones with a positive mantissa of all zeros.

FMU - Floating Multiply - 63	0	(M)	(M)	(M)
	Floating,	$(A_1) \times$	$(A_2) \rightarrow$	A_3

Multiply the contents of the location specified by A_1 by the contents of the location specified by A_2; store the most significant portion of the floating-point double-precision product in the location specified by A_3, and store the least significant portion of the floating-point product in the TFC register (see Note 5). Both operands are assumed to be in floating-point format.

NOTE 1: If the mantissa of either operand is not normalized, or if either one contains floating-point zero, the program non-normalized flip-flop is set and the operation is performed.

NOTE 2: When the exponents of the two operands are added, overflow or underflow can occur, in which case the FMU instruction is immediately terminated and the program overflow or underflow flip-flop is set. In the case of overflow, the absolute value of the mantissa portion of the location specified by A_1, together with the overflow exponent, is inserted in the location specified by A_3; for underflow, floating-point zero is inserted in the location specified by A_3.

NOTE 3: If the resulting mantissa (without sign) is zero, floating-point zero is inserted in the location specified by A_3, and the program underflow flip-flop is set.

NOTE 4: If the resulting mantissa is not equal to zero, it will be normalized one bit position, if necessary, and the exponent will be corrected. In correcting the exponent, underflow may occur, in which case floating-point zero is inserted in the location specified by A_3, and the program underflow flip-flop is set.

NOTE 5: The mantissa of the TFC register contains the thirty-five least significant bits of the product and the sign is the same as the sign of the mantissa of the result. The exponent of the TFC

register is the same as the exponent of the most significant portion of the product, except in the case where the result has been normalized one bit position. In this case, the exponent of the most significant portion will be one less than the exponent of the TFC register, and the least significant bit of the mantissa of the result will be the same as the most significant bit of the mantissa of the TFC register.

NOTE 6: Floating-point zero is the smallest possible positive number, or 0×2^{-2047}, and appears as a negative exponent of all ones with a positive mantissa of all zeroes.

FDV - Floating Divide - 62	0	(M)	(M)	(M)
	Floating,	$(A_1) \div (A_2) \rightarrow A_3$		

Divide the contents of the location specified by A_1 by the contents of the location specified by A_2; store the floating-point quotient in the location specified by A_3, and store the floating-point remainder in the TFC register. Both operands are assumed to be in floating-point format.

NOTE 1: If the mantissa of either operand is not normalized, or if either contains floating-point zero, the program non-normalized flip-flop is set and the operation is performed.

NOTE 2: When the exponents of the two operands are subtracted, overflow or underflow can occur, in which case the FDV instruction is immediately terminated and the program overflow or underflow flip-flop is set. In the case of overflow, the absolute value of the mantissa portion of the location specified by A_1, together with the overflow exponent, is inserted in the location specified by A_3; for underflow, floating-point zero is inserted in the location specified by A_3.

NOTE 3: If the mantissa of the quotient (without sign) is zero, floating-point zero is inserted in the location specified by A_3.

NOTE 4: Quotient overflow of 1 bit, if it occurs, is automatically corrected, as is the exponent. If exponent overflow then occurs, the operation is terminated and the program overflow flip-flop is set. In this case the correct mantissa,

with the overflow exponent (+. 0————————01) is inserted in the location specified by A_3. Quotient overflow resulting from non-normalized operands will cause the program overflow flip-flop to be set. Automatic mantissa and exponent overflow correction of 1 bit still occurs, but all other overflow bits are lost.

NOTE 5: The TFC contains the 35 bit remainder, right justified, with the sign of the dividend and a zero exponent.

This comparison test can be avoided by using the following equation: The correct remainder exponent = 2(dividend exponent) - (34 + divisor exponent + quotient exponent).

NOTE 6: Floating-point zero is the smallest possible positive number, or 0×2^{-2047}, and appears as a negative exponent of all ones with a positive mantissa of all zeroes.

REGISTER MANIPULATION AND DATA MOVING INSTRUCTIONS

TRS - Transmit - 35	0	(M) $(A_1) \rightarrow \overline{(M)}$ $\overline{A_2}$

Transmit the contents of the location specified by A_1 to the location specified by A_2.

TRM - Transmit Modified - 34	0	(M) (A_1), by V_t A_2, $\rightarrow \overline{(M)}$ $\overline{A_3}$

Transmit the contents of the location specified by A_1, modified as indicated by the contents of A_2, to the location specified by A_3.

NOTE 1: The transmit variant syllable (A_2), numbering the bits from left to right, is coded as shown on the following page.

Bit: 1 - 9	10	11	12	Operation Indicated
Blank	–	0	1	The operation is transmit, with the sign set to plus.
	–	1	0	The operation is transmit, with the sign set to minus.
	–	1	1	The operation is transmit, with the sign changed.
	1	–	–	The operation is transmit, with the number rounded, and can occur with sign modification.

NOTE 2: Rounding is accomplished by testing the most significant data bit (bit 2) of the TFC register. If it is a one, one is added to the least significant bit-position of the number contained in the location specified by A_1. If overflow occurs during round, the program overflow flip-flop is set and the overflow result is stored in the location specified by A_3. Rounding is for fixed-point numbers.

STF - Store Thin Film - 15	0 T (M) Thin film, $(A_1) \rightarrow A_2$

Store the contents of the thin film register (or group of registers) specified by A_1 in the least significant end (or full word) of the location specified by A_2.

NOTE 1: The thin film syllable, numbering the bits from left to right, is coded thus:

Bits 1, 2, 4, and 5 are not used.

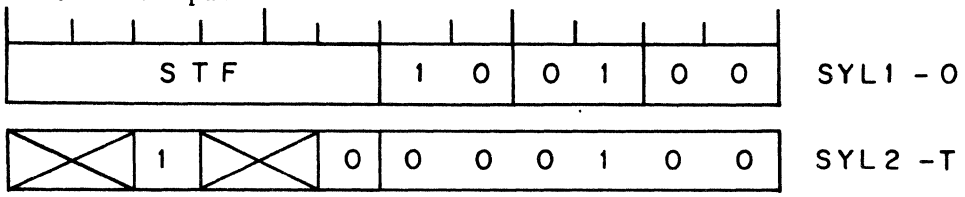
Bit 3 if "one" indicates that more than one thin film register is addressed, and if "zero," indicates that one thin film register is addressed.

Bit 6 if "one," indicates that 12-bit thin film registers are addressed, and if "zero," indicates that 16-bit thin film registers are addressed.

Bits 6 through 12 are the thin film register address.

NOTE 2: If bit 3 of the thin film syllable is equal to 1, 48 bits, composed of four 12-bit or three 16-bit consecutive registers, are referenced. In this case, bits 11-12 of the thin film address are treated as zero. Although the operand stack registers are located in thin film, they are not normally addressed via the thin film instructions since they have special addressing controls associated with them. (See thin film map, Appendix C.)

NOTE 3: Example



Store the contents of index register number 4 in the 16 least significant bits of the top of the stack. Store the contents of index register number 5 in the next 16 bits of the top of the stack. Store the contents of index register number 6 in the 16 most significant bits of the top of the stack.

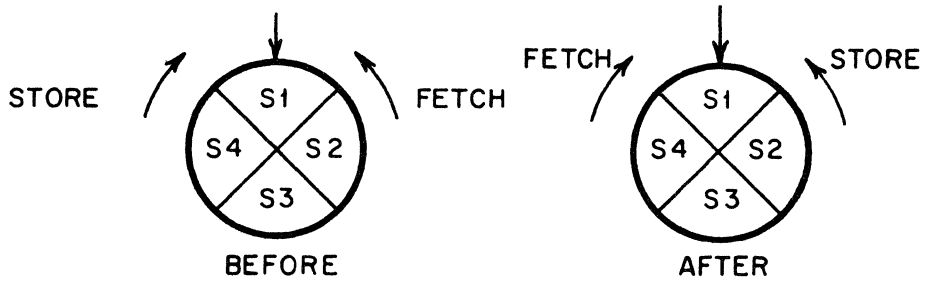
LTF - Load Thin Film - 30	0 (M) T $(A_1) \rightarrow A_2$, thin film

Load the least significant end (or whole word) of the contents of the location specified by A_1 into the thin film register (or group of registers) specified by A_2. (See notes 1 and 2 of STF for the thin film syllable description.)

NOTE 1: The IAR can only be loaded during control mode operation; all other thin film registers are accessable in both modes.

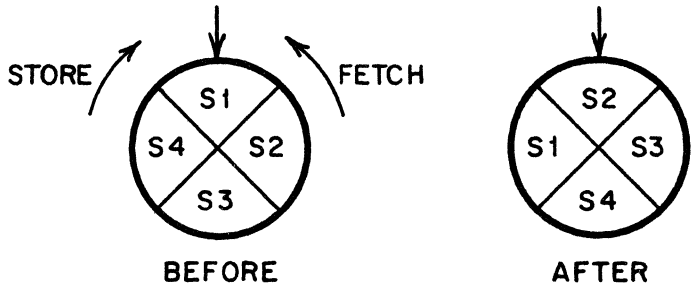
RVS - Reverse Stack - 06	0

Reverse the direction of the stack counter.



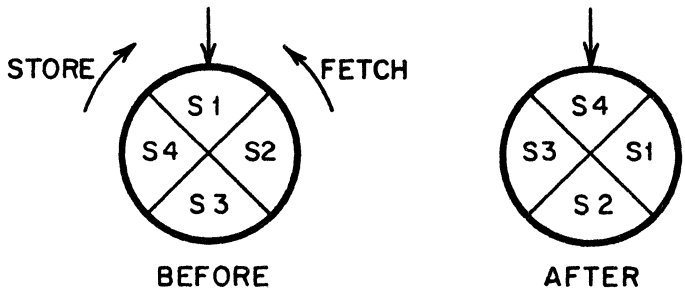
SSU - Step Stack Up - 02	0 s + 1 ⟶ s

Step the stack counter one in the fetch direction.



SSD - Step Stack Down - 03	0 s - 1 ⟶ s

Step the stack counter one in the store direction.



XLC - Index, Limit - Compare - 12	0 Ia Iv B

Increase or decrease the contents of the index register specified by bits 5 thru 8 of A_2, by the amount specified in A_1. Then compare the contents of this index register with the contents of the limit register specified by bits 9 thru 12 of A_2 for the condition(s) indicated by bits 2 thru 4 of A_2. If the condition(s) is (are) true, take the next instruction from the location specified by A_3; otherwise, continue in sequence.

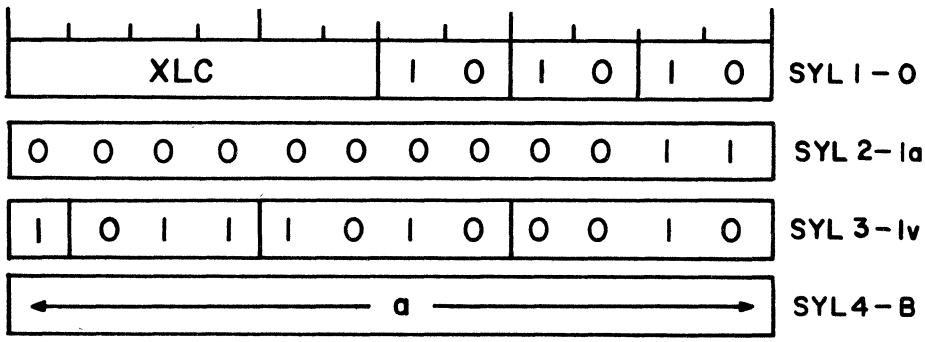
NOTE 1: The index variant syllable (A_2), numbering the bits from left to right, is coded thus:

Bit 1	Branch Condition 2 3 4	5 - 8	9 - 12
increase if 0 decrease if 1	0 0 0 none 0 0 1 equal 0 1 0 index > limit 0 1 1 index ≥ limit 1 0 0 index < limit 1 0 1 index ≤ limit 1 1 0 index ≠ limit 1 1 1 unconditional	Index register number (1 thru 15)	Limit register number (0 thru 15)

NOTE 2: If limit register address zero is used, the contents of the specified index register is compared with the number "zero".

NOTE 3: If an index register underflows when it is decreased or overflows when it is increased, the result is in the modulo 65, 536 form.

NOTE 4: Example:



Decrease the contents of index register #10 by 3; compare the new contents of index register #10 with the contents of limit register #2, and if the contents of the index register are ≥ to the contents of the limit register, take the next instruction from (unconditional transfer to) the location specified by $\lceil a + BPR \rceil$; otherwise, continue in sequence.

SER - Store External Requests - 21	0 (M)

Store the external request lines in the least significant bits of the location specified by A_1. The external request lines are inputs to the D825 system whose 0 or 1 state serve to communicate a signal to a computer regarding external requests for input-output processing service.

CLA - Clear - 20	0 (M) Zeroes → A_1

Make the contents of the location specified by A_1, including the sign, equal to zero.

UNPACKING AND PARTIAL WORD INSTRUCTIONS

BAF - Binary Add, Field - 43	0 (M) F (M)

Strip (extract) first the contents of the location specified by A_1 and then the contents of the top of the stack, as indicated by the contents of A_2. Perform a 48-bit unsigned addition on these two quantities and strip the result as indicated by the contents of A_2. Clear a field of the contents of the location specified by A_1, also defined by the contents of A_2. Logically OR these two results, and adjust (shift) as indicated by the contents of A_2; store this final result in the location specified by A_3.

NOTE 1: Note that the original contents of the top of the stack and of the location specified by A_1 remain unchanged unless the contents of A_3 specifies the same location as one of these.

NOTE 2: The implied stack address mentioned above cannot be identified in the command syllable, since the instruction is essentially a 4-address instruction. The stack will be held, in order to make repeating of the instruction more useful. Note that this stack-read is included in the execution time of the instruction.

NOTE 3: The special field syllable, numbering the bits from left to right, is defined thus:

a. A field is composed of from 1 to 8 physically adjacent 6-bit characters; the character positions are numbers 0-7 from left to right.

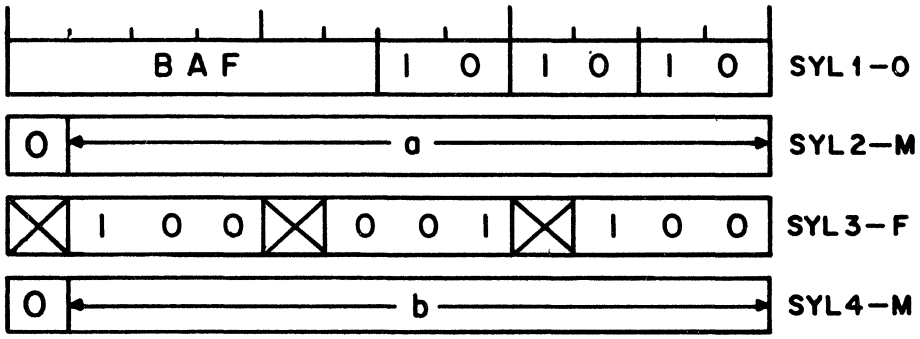
b. Bits 6-8 of the field syllable contain the length (number of characters) of the field which is to be stripped. A length of zero means that the field to be stripped is a full 8-character word.

c. Bits 10-12 of the field syllable contain the position number of the left-most character of the field to be stripped.

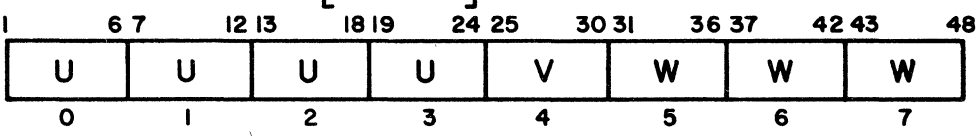
d. Bits 2-4 of the field syllable contain the number of character positions through which the stripped field is to be shifted (adjusted). This shift is a single, right logical, end-around shift of 0-7 character positions.

NOTE 4: The thin film C register is used to store the contents of A_1.

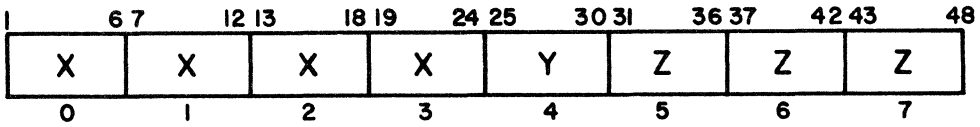
NOTE 5: Example



INITIALLY LOCATION [a + BAR] CONTAINS:



INITIALLY THE TOP OF THE STACK CONTAINS:



Extract a field (of length 1 and starting position 4) of the contents of both the location specified by a + BAR and the top of the stack (0000V000 and 0000Y000, respectively). Add these two extracted portions, without sign, and from the result, again extract a field of length 1 and starting position 4, giving 0000(V + Y)000. Store this field in the corresponding position of the contents of the location specified by a + BAR , giving UUUU(V + Y)WWW, shift right end-around 4 positions, and store the result in the location specified by b + BAR , giving (V + Y)WWWUUUU.

BSF - Binary Subtract, Field - 42	0 (M) F (M)

The operation is a 48-bit unsigned subtraction; the result is the absolute value of the difference of the two operands. See BAF for description and notes.

LAF - Logical AND, Field - 47	0 (M) F (M)

The operation is logical AND. See BAF for description and notes.

LOF - Logical OR, Field - 44	0 (M) F (M)

The operation is logical OR. See BAF for description and notes.

LXF - Logical EXCLUSIVE OR, Field - 45	0 (M) F (M)

The operation is logical EXCLUSIVE OR. See BAF for description and notes.

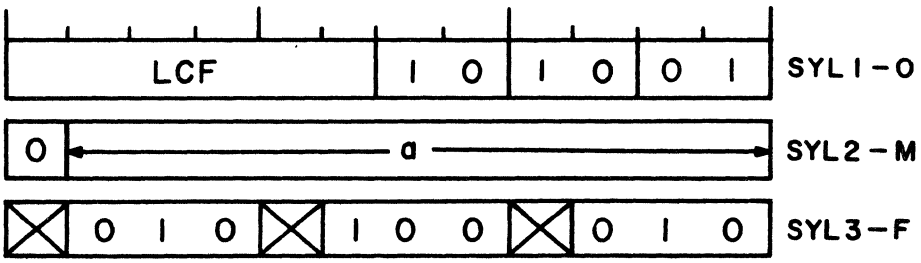
LCF - Logical COMPLEMENT Field - 46	0	(M)	F	(M)

Strip (extract) the contents of the location specified by A_1 as indicated by the contents of A_2. One's COMPLEMENT, and strip the result as indicated by the contents of A_2. Clear a field of the location specified by A_1; this field is defined by the contents of A_2. Logically OR these two quantities and adjust (shift) the result as indicated by the contents of A_2; store the final result in the location specified by A_3.

NOTE 1: Note that the original contents of the location specified by A_1 remain unchanged unless the contents of A_3 specify the the same location.

NOTE 2: See BAF Notes 3 and 4.

NOTE 3: Example



INITIALLY LOCATION [a + BAR] CONTAINS:

1 – 6	7 – 12	13 – 18	19 – 24	25 – 30	31 – 36	37 – 42	43 – 48
A	B	C	D	E	F	G	H
0	1	2	3	4	5	6	7

INTERMEDIATE RESULTS:

	1 – 6	7 – 12	13 – 18	19 – 24	25 – 30	31 – 36	37 – 42	43 – 48
(1)	0	0	C	D	E	F	0	0
(2)	0	0	$\overline{C}$	$\overline{D}$	$\overline{E}$	$\overline{F}$	0	0
(3)	A	B	$\overline{C}$	$\overline{D}$	$\overline{E}$	$\overline{F}$	G	H
(4)	G	H	A	B	$\overline{C}$	$\overline{D}$	$\overline{E}$	$\overline{F}$

(1) Extract a field (of length 4 and starting position 2) of the contents of the location specified by [a + BAR].

(2) Complement, extract the same field.

(3) Store this field in the corresponding position of the contents of the location specified by [a + BAR].

(4) Shift right end-around 2 positions, and store the result in the top of the stack.

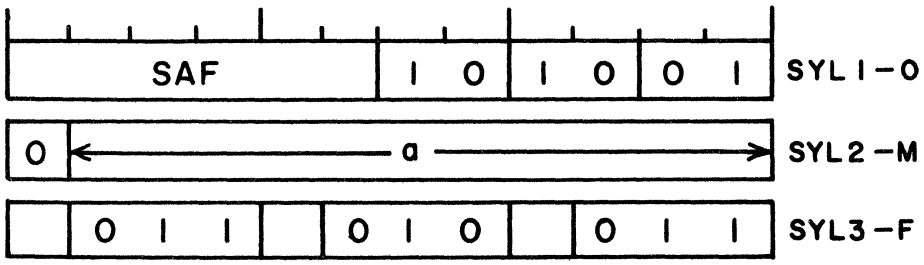
SAF - Strip and Adjust Field - 41	0 (M) F (M)

Strip (extract) and adjust (shift) the contents of the location specified by A_1, as indicated by the contents of A_2, and store the result in the location specified by A_3.

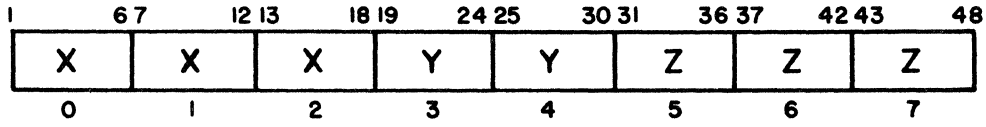
NOTE 1: Note that the original contents of the location specified by A_1 remain unchanged unless A_1 and A_3 specify the same location.

NOTE 2: See BAF for the field syllable definition.

NOTE 3: Example



INITIALLY LOCATION [a + BAR] CONTAINS:



Extract a field (of length 2 and starting position 3) of the contents of the location specified by [BAR + a]. Shift this extracted portion (000YY000) right, end-around, 3 character positions. Store the result (000000YY) in the top of the stack.

AIF - Adjust and Insert Field - 40	0 (M)	F (M)

Clear a field of the contents of the location specified by A_1; this field is defined by the contents of A_2.

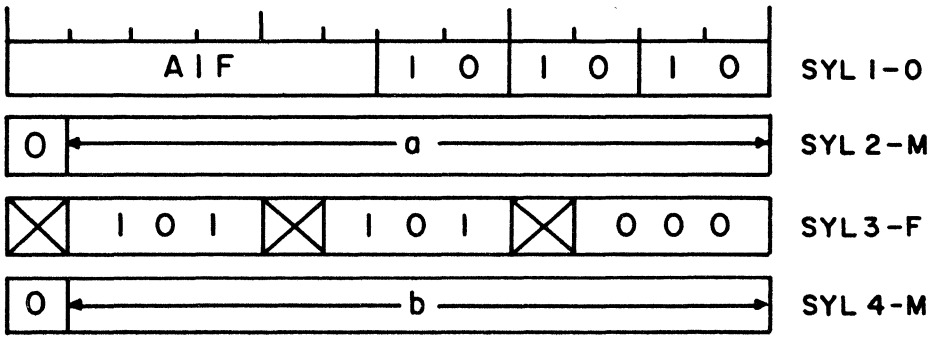
Adjust (shift) the contents of the top of the stack, as indicated by the contents of A_2. Logically OR these two quantities and store the result in the location specified by A_3.

NOTE 1: Note that the original contents of the top of the stack and of the location specified by A_1 remain unchanged unless the contents of A_3 specify the same location as one of these.

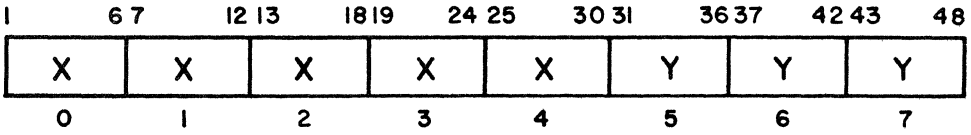
NOTE 2: The implied stack address of the second operand can not be identified in the command syllable, since the instruction is essentially a 4-address instruction. The stack will be held, in order to make repeating of the instruction more useful. The contents of the top of the stack are assumed to contain an isolated field (a field surrounded by zeroes) since the inserting in another word is done by ORing.

NOTE 3: See BAF for the field syllable definition.

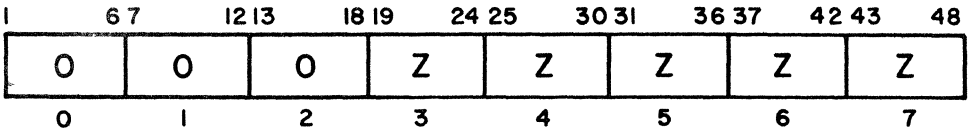
NOTE 4: Example



INITIALLY LOCATION [a + BAR] CONTAINS:



INITIALLY THE TOP OF THE STACK CONTAINS:



Clear a field (of length 5 and starting position 0) of the contents of the location specified by [a + BAR], giving 00000YYY. Shift the contents of the top of the stack right, end-around, 5 character positions, giving ZZZZZ000. OR these two quantities and store the result (ZZZZZYYY) in the location specified by [b + BAR]

CEF - Compare Equal Field - 52 CGF - Compare Greater Field - 51 CLF - Compare Less Field - 50	0	(M)	F	B

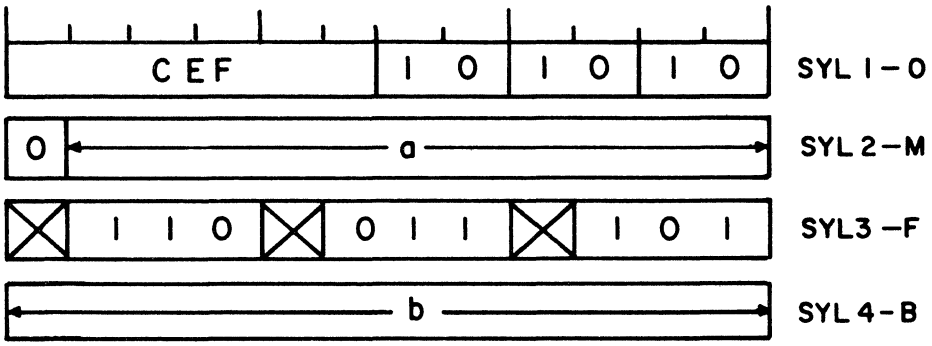
Strip (extract) and adjust (shift) the contents of the location specified by A_1, as indicated by the contents of A_2. Alphanumerically compare this quantity with the contents of the top of the stack; if the condition is true (e.g. the 1st quantity greater, equal, or less than the contents of the top of the stack), take the next instruction from the location specified A_3; otherwise continue in sequence.

NOTE 1: The contents of the top of the stack and of the location specified by A_1 remain unchanged.

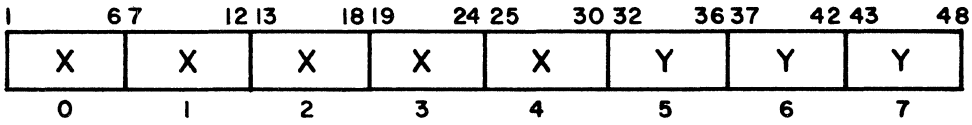
NOTE 2: The implied stack address of the second operand cannot be identified in the command syllable, since the instruction is essentially a 4-address instruction. In this case the stack will be held, in order to make repeating of the instruction more useful. The contents of the top of the stack are assumed to contain an isolated field (a field surrounded by zeroes), since the comparison is made on all 48 bits.

NOTE 3: See BAF for the field syllable definition.

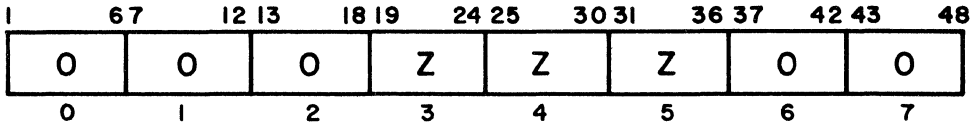
NOTE 4: Example



INITIALLY LOCATION [a + BAR] CONTAINS:



INITIALLY THE TOP OF THE STACK CONTAINS:



Extract a field (of length 3 and starting position 5) of the contents of the location specified by [a + BAR]. Shift this extracted portion (00000YYY) right, end-around, 6 character positions. Compare this (000YYY00) with the contents of the top of the stack. If equal, take the next instruction from the location specified by [b + BPR]; otherwise, continue in sequence.

CSE - Character Search - 32	0 (M) C B
Character,	$(A_1) = A_2 : A_3$

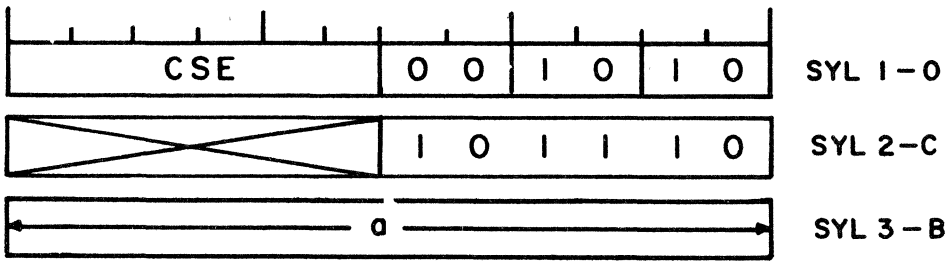
Compare the 6-bit character specified by bits 7 through 12 of A_2 with successive characters of the contents of the location specified by A_1. The search is begun to the left of the character position indicated by the character count register (CCR) and continued on to the leftmost character of the word, or until the character is found. If the CCR, a

thin film register, contains zero, the search is begun with the rightmost character of the word. The search is controlled by a counter which is loaded with the contents of the CCR. This counter is first stepped down, then the character position it indicates is examined, and then the counter is tested for zero. At the end of the instruction the contents of the counter is preserved in the CCR. If the character is found, the computer takes the next instruction from the location specified by A_3, and the CCR indicates the position in which the character was found. If the character is not found the next instruction in sequence is executed, and the CCR contains zero.

NOTE 1: The 6-bit character-positions of the data word are numbered 0-7, from left to right. In order to search at the beginning (rightmost character) of a word, the character count register must previously be reset (equal to zero).

NOTE 2: A second character search on the same word will take up the search on the character after (to the left of) the previous one found.

NOTE 3: Example



Decrease the character counter and, beginning with the character position indicated by this count, right-to-left search the contents of the top of the stack for the character, "101110". Unconditional transfer to the location specified by [a + BPR] if found; otherwise continue in sequence. In either case store the last character count in the character count register.

LAN - Logical AND - 56	0	$(M)_{(A_1)} \cdot (M)_{(A_2)} \rightarrow \overline{(M)}_{A_3}$

Logically AND, bit-by-bit, the contents of the location specified by A_1 with the contents of the location specified by A_2; store the result in the location specified by A_3.

NOTE 1: The sign bits are also ANDed.

LOR - Logical OR - 55	0	$(M)_{(A_1)} \vee (M)_{(A_2)} \rightarrow \overline{(M)}_{A_3}$

Logically OR, bit-by-bit, the contents of the location specified by A_1 with the contents of the location specified by A_2; store the result in the location specified by A_3.

NOTE 1: The sign bits are also ORed.

LXR - Logical EXCLUSIVE OR - 54	0	$(M)_{(A_1)} \oplus (M)_{(A_2)} \rightarrow \overline{(M)}_{A_3}$

Logically EXCLUSIVE OR, bit-by-bit, the contents of the location specified by A_1 with the contents of the location specified by A_2; store the result in the location specified by A_3.

NOTE 1: The sign bits are also EXCLUSIVE ORed.

LCM - Logical COMPLEMENT - 24	0	$\overline{(M)}_{(A_1)} \rightarrow \overline{(M)}_{A_2}$

Logically (one's) COMPLEMENT, bit-by-bit, the content of the location specified by A_1; store the result in the location specified by A_2.

NOTE 1: The sign bit is also COMPLEMENTed.

SHF - Shift - 36	0 (M) S (M) (A_1) shifted, A_2, $\overline{A_3}$

Shift the contents of the location specified in A_1, as indicated by the contents of A_2, and store the result in the location specified by A_3.

NOTE 1: The shift syllable (A_2), numbering the bits from left to right, is coded thus:

BIT	If 1	If 0
1	-	-
2	-	-
3	double	single
4	right	left
5	logical	arithmetic
6	end-off	end-around
7 through 12	amount	amount

NOTE 2: Single Shifts

a. Arithmetic shifts are performed on bits 2 through 48; logical shifts are performed on bits 1 through 48.

b. Left shifts are performed 1 bit at a time; right shifts are automatically performed in optimum combinations of 1, 6, and 12 bits.

NOTE 3: Double Shifts

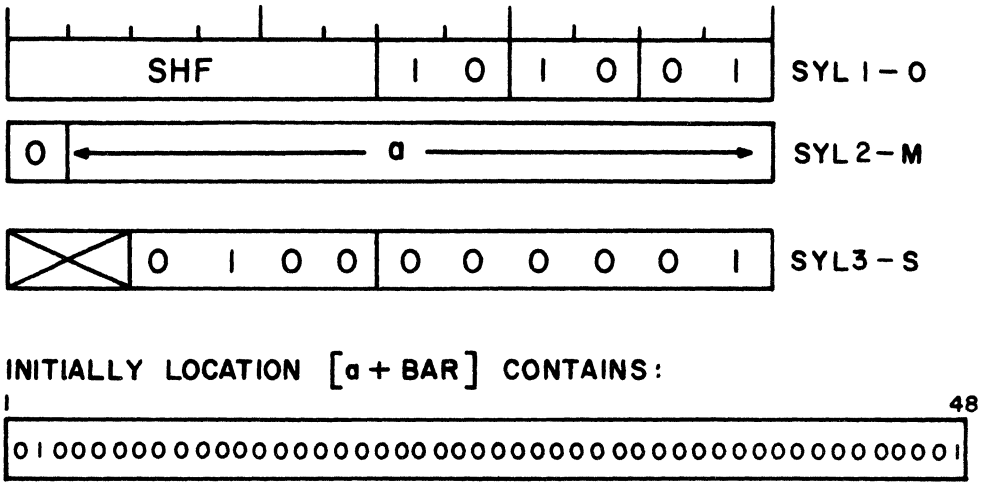
a. The "amount" must be $\leq$ 11 for the left double arithmetic shifts; for all other double shifts the "amount" must be $\leq$ 12.

b. Arithmetic shifts are performed on bits 2 through 48 of the location specified by A_1, as the most significant half, and on bits 2 through 48 of the TFC register, as the least significant half; logical shifts are performed on bits 1 through 48 of the location specified by A_1, as

the most significant half, and on bits 1 through 48 of the TFC register, as the least significant half.

c. All double shifts are performed 1 bit at a time.

NOTE 4: Example



Single, right, arithmetic, end-around shift the contents of the location specified by [a + BAR] 1 position, and store the result (0110 ⟷ 0) in the top of the stack.

PROGRAM CONTROL INSTRUCTIONS

CEQ - Compare Equal - 76	0	(M) (A_1)	(M) $= (A_2)$:	B A_3

Algebraically compare the contents of the location specified by A_1 with the contents of the location specified by A_2; if equal, take the next instruction from the location specified by A_3, otherwise continue in sequence.

NOTE 1: Positive zero and negative zero are not considered equal. Negative zero cannot occur as a result of an arithmetic operation.

CGR - Compare Greater - 75	0 (M) (M) B
Algebraic,	$(A_1)>(A_2)$: A_3

Algebraically compare the contents of the location specified by A_1 with the contents of the location specified by A_2; if the contents of the location specified by A_1 are greater, take the next instruction from the location specified by A_3, otherwise continue in sequence.

NOTE 1: Positive zero is considered greater than negative zero. Negative zero cannot occur as a result of an arithmetic operation.

CLS - Compare Less - 74	0 (M) (M) B
Algebraic,	$(A_1)<(A_2)$: A_3

Algebraically compare the contents of the location specified by A_1 with the contents of the location specified by A_2; if the contents of the location specified by A_1 are less, take the next instruction from the location specified by A_3, otherwise continue in sequence.

NOTE 1: Negative zero is considered less than positive zero. Negative zero cannot occur as a result of an arithmetic operation.

ACE - Alphanumeric Compare Equal - 72	0 (M) (M) B
Alphanumeric,	$(A_1)=(A_2)$: A_3

Compare the contents of the location specified by A_1 with the contents of the location specified by A_2; if equal, take the next instruction from the location specified by A_3, otherwise continue in sequence.

NOTE 1: Comparison is made on 48-bit unsigned words.

ACG - Alphanumeric Compare Greater - 71	0 (M) (M) B
Alphanumeric,	$(A_1)>(A_2)$: A_3

Compare the contents of the location specified by A_1 with the contents of the location specified by A_2; if the contents of the location specified by A_1 are greater, take the next instruction from the location specified by A_3, otherwise continue in sequence.

NOTE 1: Comparison is made on 48-bit unsigned words.

ACL - Alphanumeric Compare Less - 70	0 (M) (M) B $(A_1) < (A_2)$: A_3

Compare the contents of the location specified by A_1 with the contents of the location specified by A_2; if the contents of the location specified by A_1 are less, take the next instruction from the location specified by A_3, otherwise continue in sequence.

NOTE 1: Comparison is made on 48-bit unsigned words.

UCT - Unconditional Transfer - 22	0 B : A_1

Take the next instruction from the location specified by A_1.

BRB - Branch on Bit - 26	0 (M) ($\underline{M}$) B

If the least significant bit of the contents of the location specified by A_1 is a one, take the next instruction from the location specified by A_3; otherwise continue in sequence. In either case, perform a single, right, logical, end-around shift of one position, on the contents of the location specified by A_1, and store the result in the location specified by A_2.

BRC - Branch On Condition - 11	0 L B A_1 : A_2

If the condition (s) corresponding to any one (s) in the contents of A_1 is true, take the next instruction from the location specified by A_2; otherwise continue in sequence.

NOTE 1: The conditions occurring are recorded in flip-flops, and only by testing for the conditions can the corresponding flip-flops be reset. However, testing is effective only when the computer mask bit corresponding to arithmetic overflow is not set; if the bit is set, automatic interrupt occurs immediately after execution of the instruction causing the condition.

NOTE 2: The logical machine conditions are program overflow (POV), program underflow (PUN), and program non-normalized (PNN). The logical machine condition syllable is coded thus:

Bit 1 - 9	10	11	12
blank	if 1 - test PNN flip-flop	if 1 - test PUN flip-flop	if 1 - test POV flip-flop

NOTE 3: The program overflow (POV) flip-flop is set by:

a. Fixed-point arithmetic overflow resulting from addition, subtraction, or division.

b. Overflow resulting from the round instruction (TRM).

c. Exponent overflow resulting from a floating-point arithmetic operation. Exponent overflow is defined as overflow, resulting from the addition of two positive-signed exponents.

d. Quotient overflow resulting from use of the FDV instruction with non-normalized operands.

NOTE 4: The program underflow (PUN) flip-flop is set by:

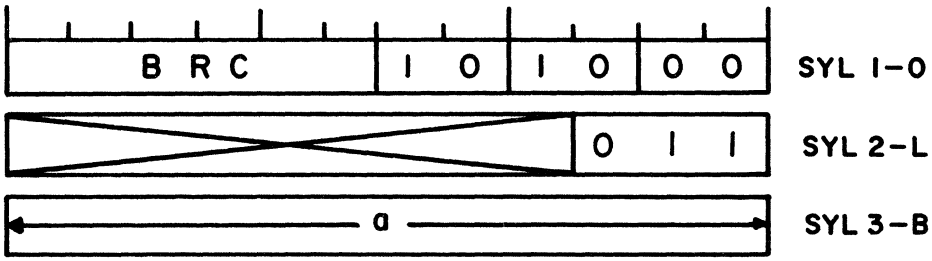
a. Floating-point exponent underflow. Exponent underflow is defined as overflow, resulting from the addition of two negative-signed exponents.

b. A zero answer resulting from floating-point addition or subtraction.

c. A zero answer resulting from floating-point multiplication using non-normalized operands.

NOTE 5: The program non-normalized (PNN) flip-flop is set by the use of an operand with a leading zero in the mantissa in floating-point multiplication or division.

NOTE 6: Example



If the POV or PUN flip-flops have been set by a previous operation, reset them and take the next instruction from the location specified by a [a + BPR]; otherwise continue in sequence.

NOP - No Operation - 00	0

Execute the next instruction in sequence.

SRJ - Subroutine Jump - 14	0 Ja Ji

Store the contents of the BAR, BPR, and the PCR in the thin film subroutine storage register. The low order 16-bits of the memory location specified by the contents of the subroutine base address register (SAR) plus A_1 (the jump address syllable) comprise the starting address of the subroutine. Load BPR and PCR with the starting address. Increment the BAR by A_2 (the jump increment syllable). Load index register #15 with the new contents of BPR, minus the new contents of BAR. Transfer control to the starting address of the subroutine.

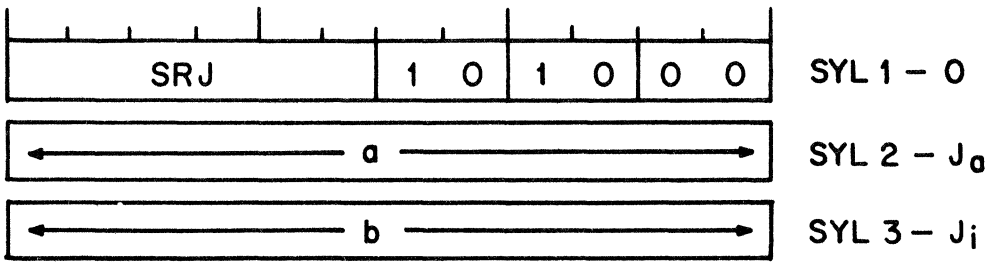
NOTE 1: While in the subroutine, if index register #15 is applied to the data address, the data thus referred to will be relative to the new contents of the BPR. If the difference between the two base registers is negative, it will appear in the two's complement form, and thus will be applicable as described.

NOTE 2: In nesting subroutines, it is necessary to save the contents of the subroutine storage register in memory, relative to the BAR of this subroutine, before going to the next subroutine and thus destroying the previous contents of the SSR.

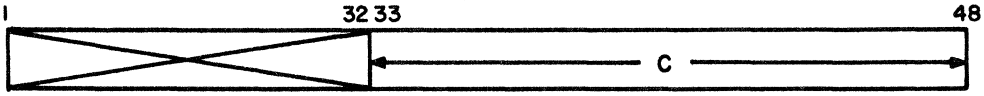
NOTE 3: Note that the location specified by [SAR + Ja] is an indirect address of one level only. If Ja is indexed, the location specified by [SAR + Ja + X] gives the indirect address, as opposed to adding the contents of the index register to the indirect address itself.

NOTE 4: The snag bit of the starting address (bit 31) may be used to set the snag bit interrupt in either control or normal mode. The interrupt will be serviced only in the normal mode.

NOTE 5: Example



INITIALLY LOCATION [a + SAR] CONTAINS:



First save the contents of the BAR, BPR, and PCR in the thin film SSR register, then load the BPR and PCR with "C". The next instruction executed will be the contents of the location specified by C, since the PCR now contains "C". Increment the BAR by "b", and load index register #15 with "C" - (BAR + "b").

SRR - Subroutine Return - 04	0

Load the BAR, BPR, and PCR with the contents of the thin film subroutine storage register. Load index register #15 with the new contents of the BPR, minus the new contents of the BAR (see note 1 of the SRJ instruction for use of index register #15). Take the next instruction in sequence, specified by the restored PCR. The program will continue, with its addressing now relative to the restored BAR, BPR and PCR.

NOTE 1: Since the location specified by the PCR is a 48-bit program word, the first syllable of the program word must contain the operator syllable of the instruction to be executed. This syllable layout requirement, necessary because of the variable-length instruction capability, is the responsibility of the coder.

RPT - Repeat - 10	0 R_c R_i B Repeat (A_3), by A_2, A_1 times

Perform the instruction contained in the location specified by A_3 the number of times specified by the contents of A_1 (the repeat count syllable). Each time the instruction is repeated (after the first time) its three (or less) "address" syllables are incremented, respectively, by the 3 4-bit syllable increments contained in A_2 (the repeat increment syllable).

NOTE 1: The first time through, the repeated instruction employs normal addressing and can therefore contain indirect addressing, although indexing is not allowed. In this case, subsequent iterations use the last-level address as the

base address and increment that address each time. Note that the three syllable increments correspond directly to the three (or less) syllables following the operator syllable of the repeated instruction.

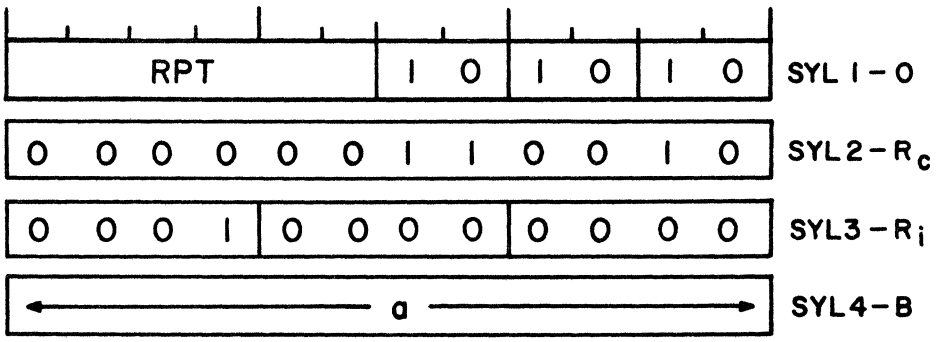
NOTE 2: The repeating of an instruction will cease if the instruction contains a branch which is executed. The repeat count register (RCR) will retain the repeat count, minus the number of times the instruction was executed prior to branching. Since RCR is a 12-bit register, the maximum repeat count is 4095.

NOTE 3: Since A_3 is a branch syllable, the repeated instruction must be stored with its operator syllable as the first syllable of the location specified by A_3.

NOTE 4: The RPT instruction itself may not be repeated. Other instructions which may not be repeated are XLC, SRJ, SRR, and IRR.

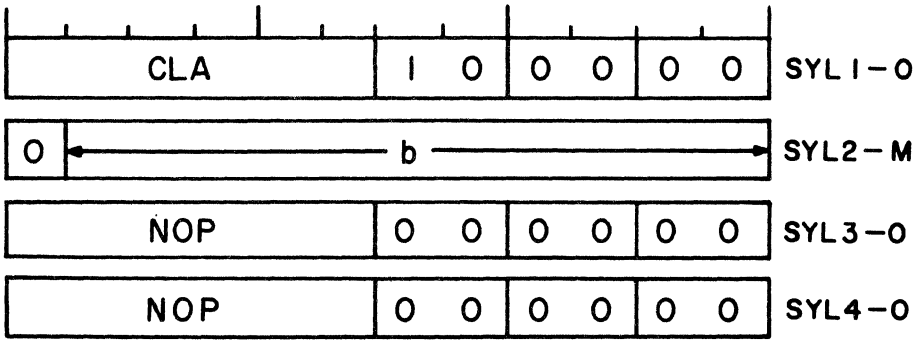
NOTE 5: If the repeat count is ZERO, no execution of the repeated instruction will occur.

NOTE 6: Example



Clear consecutive memory locations [b + BAR] through [b + 49 + BAR].

Location [a + BPR] contains the instructions:



INTERRUPT CONTROL INSTRUCTIONS

HLT - Halt - 01	0

This instruction has two distinct meanings, depending on which mode of operation prevails. In the normal mode it causes transfer to the control mode and involves the interrupt bookkeeping processes described in Section 6. When the halt instruction is used in the control mode, the computer stops.

LSR - Load Special Register - 31	0	(M)	Vs

This instruction, operable only in the control mode, has three variations which permit loading the mask register or memory bounds registers from the least significant portion of the contents of the location specified by A_1, or interrupting the computer designated by the least significant bits of the contents of the location specified by A_1.

The special register variant syllable, numbering the bits from left to right, is coded thus:

Bits 1 through 9 are blank.

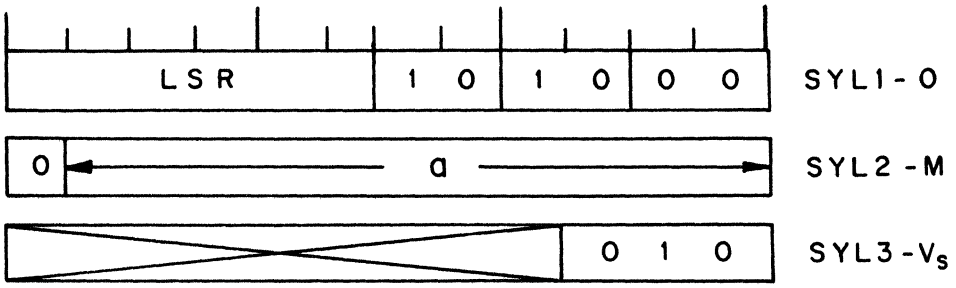
Bit 12, if equal to one, indicates that the mask register is to be loaded as described in Section 6.

Bit 11, if equal to one, indicates that the 8 least significant bits of the contents of the location specified by A_1 are to be loaded into the lower memory bounds register, and the next 8 least significant bits are to be loaded into the upper bounds register.

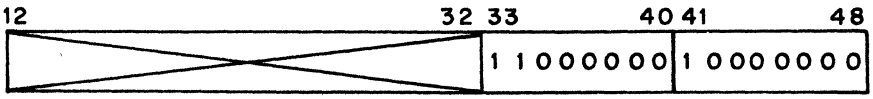
Bit 10, if equal to one, indicates that the computer designated by the three least significant bits of the contents of the location specified by A_1 is to be interrupted, and this interrupt has no corresponding mask bit.

NOTE 1: In the normal mode, a computer cannot write in memory outside the bounds whose eight most significant bits are specified by the upper and lower limit registers. These restrictions are removed in the control mode.

NOTE 2: Example



LOCATION [a + BAR] CONTAINS:



Load the lower memory limit register with 1 0 0 0 0 0 0 0. Load the upper memory limit register with 1 1 0 0 0 0 0 0. Thereafter, during normal mode operation, any attempt to write in a memory location whose address is less than 32, 768 or greater than 49, 407, will be stopped and the write-out-of-bounds interrupt register bit will be set. (Provided that the appropriate mask bit is a one.)

TIO - Transmit to Input/Output - 16	0 IO M B

This instruction can only be given in the control mode. It is employed to send the Setup Descriptor, Command Descriptor, or Release Descriptor, obtained from the memory location specified by A_2, to an I/O Control Module. (See Section 5, Input-Output Programming, for the use and format of these descriptors.)

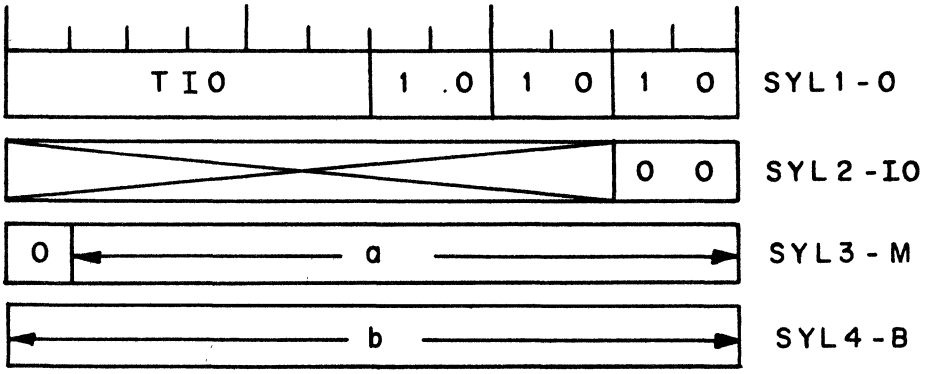
The instruction has two variations: the command TIO and the unconditional TIO. The command variation is used to transmit a Command Descriptor to an I/O Control Module, if there is one available. If there is no I/O Control Module available the computer will take its next instruction from the location specified by A_3. The unconditional variation is used to transmit a Setup Descriptor or Release Descriptor for immediate access to the I/O Control Modules. The branch syllable is ignored for the variation, since the descriptor is sent unconditionally. The descriptor involved will be sent to the I/O Control Modules associated with the I/O bus designated by A_1 and called bus "A" or bus "B".

The special input-output syllable (IO), numbering the bits from left to right, is coded thus:

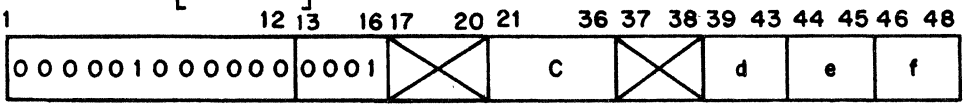
Bits 1 through 10 are blank.

Bit 11	Bit 12	Variation	Bus	Use
0	0	Command	A	Command Descriptor
0	1	Unconditional	A	Setup or Release Descriptor
1	0	Command	B	Command Descriptor
1	1	Unconditional	B	Setup or Release Descriptor

NOTE 1: Example



LOCATION [a + BAR] CONTAINS THE COMMAND DESCRIPTOR:



Transmit the Command Descriptor found in memory location [a + BAR] to the lowest-numbered non-busy I/O Control Module. If no I/O Control Modules are available, take the next instruction from the location specified by [b + BPR]. Otherwise, continue in sequence, while the I/O Control Module takes over execution of the command as follows:

- perform operation "f", type "e",
- on device number "d",
- involving 1 record (bits 13 to 16),
- using memory locations "c" through "c + 63", or 64 words (bits 1 to 12).

IRR - Interrupt Return - 05	0

Restore the contents of the BAR, BPR, and PCR, as well as the contents of the PSR and associated controls. Reset the control-mode flip-flop. Execute the next instruction contained in the restored PSR; this automatically continues the object program after interrupt processing. This instruction is operable only in the control mode.

SECTION 5

INPUT/OUTPUT PROGRAMMING

In the D825 system, input/output control is provided by the Transmit to Input/Output instruction (TIO) and a group of descriptors which are stored in memory. The TIO instruction, described in Section 4, may be executed only in the control mode, and the memory address referenced is a descriptor which will control the communications between the Memory Modules and the I/O exchange (I/O Control Modules and terminal devices). Externally requested I/O operations are made known to the system by one of the 16 external request lines (see Section 6). In the descriptor formats, a record count of zero will be interpreted as 16.

DESCRIPTOR TYPES

Five descriptor types are used in D825 I/O operations as follows:

- Setup Descriptor
- Command Descriptor
- In-Process Descriptor
- Result Descriptor
- Release Descriptor

The Command, Release, and Setup Descriptors, are constructed by program and used in the control mode of operation. The In-Process and Result Descriptors are generated by the logic of the system. The program-generated descriptors are used for communication to the Control Modules, and the logic generated descriptors are used in communication from the Control Modules.

DESCRIPTOR FUNCTIONS

In normal operation the Command Descriptor initiates and controls all programmed I/O operations, and all other descriptors perform subordinate functions. The Release Descriptor is used to return an I/O Control Module to a non-busy status after an I/O operation. The Setup Descriptor is used to establish the base address of the descriptor list which consists of an In-Process Descriptor and a Result Descriptor for each I/O Control Module. The descriptor list functions as a permanent indication of the current status of each I/O Control Module.

The sequence of events in a full I/O operation is given below.

1. A TIO instruction, with a Setup Descriptor, is used to establish the descriptor base address of where the I/O Control Modules return descriptors. The Setup Descriptor is sent at initial loading of the operating system or to move the descriptor list. I/O Control Modules remain busy after receipt of a Setup Descriptor.

2. A TIO instruction is used with a Release Descriptor to release the I/O modules that are used for subsequent Command Descriptors.

3. A TIO instruction, with a Command Descriptor, is used to initiate each I/O operation. The Command Descriptor contains the terminal device number, operation code, record count, word count, and memory starting address.

4. At the I/O Control Module which is processing the I/O operation, certain status bits of the Command Descriptor may be set, and a copy of this modified Command Descriptor is transmitted to the descriptor list as an In-Process Descriptor.

5. When the I/O operation is completed or terminated, a second modified copy of the Command Descriptor is transmitted to the descriptor list as the Result Descriptor.

5. An I/O termination interrupt is initiated, and the Result Descriptor is examined by the computer interrupted. An I/O termination interrupt can only be caused by a Result Descriptor which was generated by an I/O Control Module.

6. If the I/O operation was successful, a TIO instruction is used with a Release Descriptor to release the I/O Control Module for subsequent I/O operations. Only a Release Descriptor can place a busy I/O Control Module in the non-busy state.

In a system using two busses for I/O, the I/O exchange being addressed is selected by the A_1 syllable of the TIO instruction.

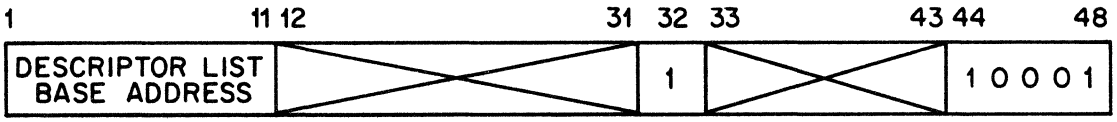
All I/O Control Modules examine every descriptor transmitted from memory. If the descriptor is a Setup Descriptor, the base address of the descriptor list is accepted by all. If the descriptor is a Command Descriptor, the I/O operation is initiated by the first non-busy I/O Control Module. If the descriptor is a Release Descriptor, the I/O Control Module number is contained in the device address field, and only that I/O Control Module is released.

If a non-busy I/O Control Module accepts a descriptor, the In-Process Descriptor returned is sent to the first location in the descriptor list plus twice the Control Module number. The Result Descriptor is sent to the following memory location.

In the event that an I/O Control Module cannot return a descriptor to the specified place, a "one" is inserted in bit 4 of the descriptor base address and an attempt is made to send the descriptor to the next higher Memory Module. Because of this feature, the descriptor list base address in the original Setup Descriptor should refer to an even numbered Memory Module. Once bit four is set in the "one" state, it remains in this state until another Setup Descriptor is sent.

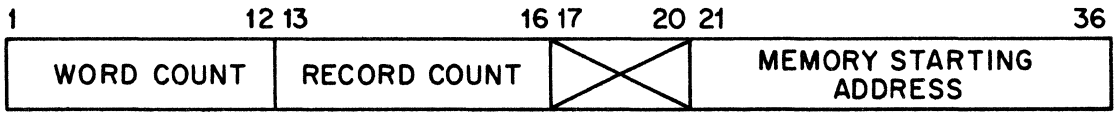
In the event of simultaneous memory requests, the I/O bus has a higher priority than any computer requesting at the same time. If more than one I/O Control Module have simultaneous requests, those with the priority bit set are serviced in numerical order and then non-priority requests are processed in numerical order.

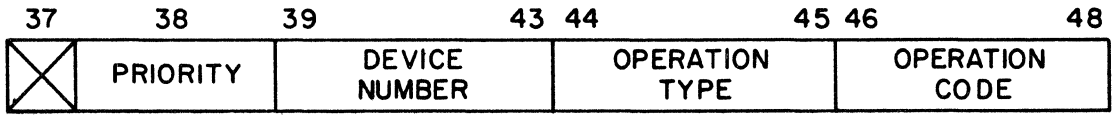
SETUP DESCRIPTOR FORMAT



The descriptor-list base address contains the 11 most significant bits of the memory address. The remaining five bits are supplied by the logic in each I/O Control Module. Two descriptor lists, and therefore two base addresses, are required with a system having two I/O busses. In the Setup Descriptor format, bits 12 through 31 and 33 through 43 must be zero.

COMMAND DESCRIPTOR FORMAT





Programming of word count and record count will vary between I/O devices. In the case of magnetic tapes, standard tape formats may be used. Memory starting address is the 16-bit starting address of the block being written into memory or read out of memory. Priority for an I/O operation is indicated by programming bit 38 as a one. The device number is programmed in bits 39 to 43. Device numbers are assigned according to the I/O complement of specific D825 systems.

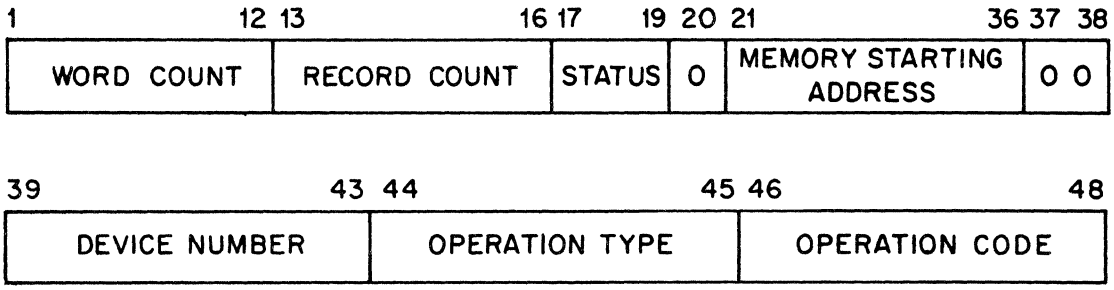
Operation type, bits 44 and 45, is programmed as follows:

Bit 44	Bit 45	
0	0	Write operation with a one way device.
1	0	Read operation with a one way device.
0	1	Write operation with a two way device.
1	1	Read operation with a two way device.

The operation codes for bits 46, 47 and 48, are given in the publications dealing with the specific I/O devices.

Bits 17 to 20 and bit 37 are status bits and are not used in the Command Descriptor.

IN-PROCESS DESCRIPTOR FORMAT

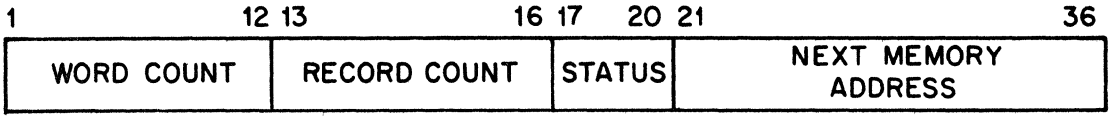


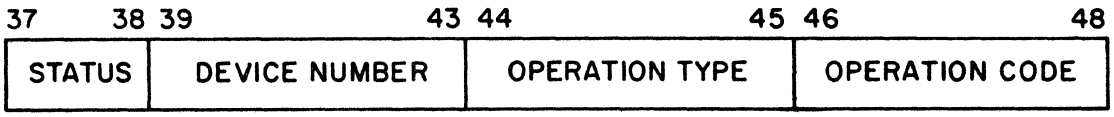
Except for the bits marked "status" and bits 20, 37, and 38 in the above diagram, all bits of the In-Process Descriptor are identical to those of the Command Descriptor which initiated the I/O operation. In this sense the In-Process Descriptor may be considered a "reflection" of the Command Descriptor from the I/O Control Module, with certain significant changes.

The significance of the status-bit coding of the In-Process Descriptor is shown below:

Bits 17	18	19	
1	1	1	Parity of the descriptor was incorrect on arrival at the I/O Control Module.
0	0	1	The I/O device requested by the Command Descriptor is not available. The device is either inoperable or connected,by the exchange,to another I/O Control Module.
0	0	0	None of the conditions for which status codes are available have been encountered. A successful I/O operation.

RESULT DESCRIPTOR FORMAT





In the Result Descriptor, the device number, operation type, and operation code are identical to the Command Descriptor which initiated the I/O operation. The first 16 bits will contain the word count and record count of the operation existing at the time the operation ended or was interrupted by a Release Descriptor. Bits 21 to 36 contain the memory address following the last address used in the I/O operation.

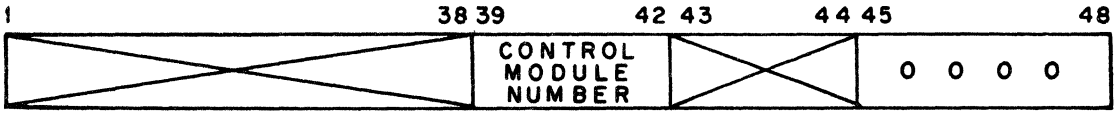
The significance of the status coding for bits 17, 18 and 19 is shown below.

Bits 17	18	19	
1	1	1	Parity error in data read from memory.
1	1	0	Parity error in data from the I/O device.
1	0	1	Power failure interrupt.
0	1	1	Word count has been reduced to zero.
1	0	0	Access to memory is not available.
0	1	0	Operation not completed when released.
0	0	0	None of the above.

The significance of the status coding for bits 20, 37, and 38 depends largely on the I/O device used in the I/O operation, and the actual coding is given in publications dealing with the specific I/O devices. The following codes apply to all devices.

Bits 20	37	38	
0	0	1	Record count has been reduced to zero.
0	0	0	None of the other conditions.

RELEASE DESCRIPTOR FORMAT



The Release Descriptor, used to release an I/O Control Module for another operation, consists of an operation code of zero, and the I/O Control Module number in bits 39-42. In the Release Descriptor format, bit 44 must be a one.

SECTION 6

THE D825 INTERRUPT SYSTEM

To implement the interrupt system, each computer module contains a 12-bit interrupt register (including 2 spare bits) and a 23-bit mask register (including 2 spare bits) (see figure 1-2). Four of the bits in the interrupt register are set directly by the occurrence of an interrupt condition; five others are set only if the corresponding mask bit is in the "1" state when an interrupt condition occurs. One bit is set during normal mode operation by any or all of sixteen "external request" interrupt conditions in coincidence with their individual mask bits. There are two additional pseudo interrupt conditions which require neither mask nor interrupt register bits, as they are of highest priority. A bit in the interrupt register is reset only when the corresponding interrupt condition is processed. Mask bits are assigned to the computers by an AOSP (LSR instruction) in order that; (a) local interrupt conditions can either be ignored or processed by the computer in which they occur, and (b) system interrupt conditions can be assigned to any computer in the system, depending on the work load and urgency of the request.

At the end of each instruction or completed iteration of a repeated instruction, the computer is available to process an interrupt condition. If either of the two highest priority conditions exists, the higher one will be processed immediately. If neither of these conditions exists but some bit in the interrupt register is set, the computer will process the existing interrupt condition of highest priority, provided it is not already processing one (operating in the control mode). The following table lists the interrupt and pseudo interrupt conditions and their characteristics:

Interrupt Condition	Interrupt Register bit Number	Number of Mask Register bits Required	Mode in which Recognized
Primary power failure	—	0	control or normal
Count real time clock	—	0	control or normal
Restart after primary power failure	1	0	control or normal
16 External requests	2	16	normal
I/O Termination	3	1	control or normal
Interrupt computer N	4	0	control or normal
RTC overflow	5	1	control or normal
Write out of bounds	6	0	normal
Illegal instructions	7	0	normal
Internal parity error or no access to memory	8	0	normal
Arithmetic overflow	9	1	normal
Normal mode halt instr. or snag bit	10	0	normal
Spare	11	spare	
Spare	12	spare	

The mask register is loaded by means of the LSR instruction, which is available during control mode operation only, as follows.

- Bits 21 thru 36 of the location specified by A_1 are the mask for external request lines 1 thru 16 respectively.
- Bit 39 is the mask for I/O termination.
- Bit 41 is the mask for RTC overflow.

- Bit 45 is the mask for arithmetic overflow.

- Bits 47 and 48 are spares; all others are not used.

Note that an external request line that is "on" will maintain its level until an I/O control unit is connected to the external device requesting service.

PRIMARY POWER FAILURE

The primary power failure interrupt condition occurs when the input ac voltage is detected out-of-tolerance. Storage circuits maintain dc supply voltages at normal levels for a period following failure detection; during this period, automatic storage of the contents of interrupt registers and control flip-flops necessary for restart is accomplished. This interrupt signal bypasses the interrupt register.

COUNT REAL TIME

The real-time clock count signal occurs once every 10 milliseconds. The count signal also bypasses the interrupt register.

RESTART AFTER PRIMARY POWER FAILURE

When a computer is turned on, bit 14 of the PDR will indicate whether this is a restart after primary power failure or not. If it is, the corresponding interrupt bit is set; the interrupt register and control flip-flops necessary for restarting the program after primary power failure are automatically loaded with the contents of the PDR; bit 14 is then reset. The computer will return to the instruction after the one during which primary power failure had occurred. If this return point is control mode operation, the interrupt processing will be completed. Once normal mode operation is in effect, however, the restart-after-primary-power-failure interrupt condition will prevail, and interrupt processing by an AOSP will begin.

EXTERNAL REQUESTS

External requests are signals to the computer(s) from the terminal devices. These signals can be examined during control mode operation by use of the SER instruction. Note that all I/O processing is handled by an AOSP in order to centralize scheduling problems and to protect the system from the possibility of data destruction by conflicting normal mode programs.

I/O TERMINATION

Input/output termination, for any reason whatever, is also a signal to the computer(s) from an I/O Control Unit. In this case a result descriptor from the I/O Control Unit is transmitted to the memory location specified by the contents of a register in the I/O Control Unit.

INTERRUPT COMPUTER N

The interrupt computer N signal occurs as the direct result of a variation of the LSR instruction which is available in the control mode only.

REAL-TIME CLOCK OVERFLOW

Real-time clock overflow can occur after the count real-time clock interrupt condition is processed. The RTC is loaded by the LTF instruction.

WRITE OUT OF BOUNDS

The write-out-of-bounds interrupt condition is a method of memory protection provided for normal mode operation. Its restrictions are: attempts to write into memory areas outside of the upper and lower memory limits, and attempts to use the LTF instruction to load the thin film Interrupt Address Register (IAR). The memory bounds registers are loaded during control mode operation by the LSR instruction. When this interrupt condition occurs, the memory write syllable (M) or the thin film syllable (T) will be skipped.

ILLEGAL INSTRUCTION

An illegal instruction during normal mode operation is defined as use of a control mode instruction or of a non-existent operation code. In the control mode this interrupt condition applies to the use of non-existent operation codes only and causes the computer to halt. The instructions which are forbidden in normal mode operation are LSR, TIO, and IRR. When this interrupt condition occurs during normal mode operation only the operator syllable (O) involved will be skipped before interrupting.

PARITY ERROR

Internal parity is checked every time a data or program word is read from memory; the parity bit is appended to the word on each instance of memory write. If the error condition occurs during control mode operation, the computer will halt. If the error condition occurs during normal mode operation, no attempt will be made to continue the instruction involved. Note that the I/O Control Module has its own parity-checking circuitry, which is not connected to the interrupt system. Thus, parity error in any descriptors or data transmitted between memory and the I/O Control Module will not cause a halt.

The interrupt register bit corresponding to this interrupt condition is also utilized to indicate failure to gain access to memory. If two consecutive count RTC signals are received without servicing the first one, and if the memory request flip-flop is set indicating an attempt to gain access to memory, the no-access-to-memory interrupt condition has occurred. If the error condition occurs during control mode operation, the computer will halt. If the error condition occurs during normal mode operation, no attempt will be made to continue the instruction involved.

ARITHMETIC OVERFLOW

Arithmetic overflow results from the following conditions:

1. Fixed-point arithmetic overflow resulting from addition, subtraction, or division.

2. Overflow resulting from the round instruction (TRM).

3. Exponent overflow resulting from a floating-point arithmetic operation.

4. Quotient overflow resulting from use of the FDV instruction with non-normalized operands.

The occurrence of any one of these conditions will cause the POV flip-flop to be set and remain set until the BRC instruction is used or until the interrupt bit is set. The arithmetic overflow interrupt bit will be set by the POV flip-flop during normal mode operation if the corresponding mask bit is set.

HALT

The HLT instruction, employed in the normal mode, causes an interrupt condition and consequent transfer to the control mode of operation. In the control mode, the HLT instruction will halt the computer. The interrupt register bit corresponding to this interrupt condition is also utilized for another purpose: in indirect addressing, if the 18th least significant bit (the snag bit) of any level of addressing after the first is a "ONE", the interrupt register bit will be set and the instruction will be completed. This capability, available in both modes, is employed to facilitate computer control of certain areas of the program.

PROCESSING OF THE INTERRUPT CONDITIONS

The two highest priority interrupts, primary power failure and count the real-time clock, are processed automatically, i.e. do not involve an AOSP or ordinary control mode operation, and are therefore classified as pseudo interrupts.

A primary power failure causes all I/O operations to cease, sending result descriptors to the memory locations specified by the contents of registers in the I/O Control Units. There is sufficient time, between detection of the condition and shut-down of the core memory, to accomplish this and to complete the execution of the present instruction or iteration of a repeated instruction. Once the execution

of the instruction is completed, the contents of all control flip-flops necessary for restart are stored in the thin film (non-volatile) Power-Failure Dump Register (PDR). The first sixteen bits of this register are the same as those loaded into the IDR (see below) with bit 14 set and bit 13 set if the primary power failure interrupt condition occurred during control mode operation. The next twelve bits are used to store the contents of the interrupt register (including 2 spare bits), and the last four bits are blank.

The count real-time clock signal occurs approximately every 10 milliseconds. This interrupt condition is processed by the hardware; it cannot upset the running of any program and can delay it only briefly (3.67 μsec). The processing is a matter of reading the contents of the RTC from thin film, incrementing it, writing it back into thin film, and setting the RTC overflow interrupt bit if the overflow condition occurred.

Handling of the ten interrupt conditions recorded by the interrupt register involves transfer to the control mode of operation, wherein the appropriate AOSP routine will service the interrupt. Transfer from the control mode back to the normal mode is the responsbility of an AOSP, and is accomplished by using the IRR instruction. The process of transfer from normal mode to control mode is automated (in order) as follows:

a. The contents of the BAR, BPR, and PCR are stored in the thin film interrupt storage register (ISR) in that order. Note that the PCR, which is stored, always contains the address of the program word to be used on returning to normal mode operation. In other words, overlap has been lost, and the PCR corrected accordingly.

b. The contents of the presently - addressed PSR are stored in the thin film interrupt program register (IPR). This is the PSR now containing the next syllable to be read if either PSR is filled.

c. The contents of certain control flip-flops are stored in the thin film interrupt dump register (IDR). Numbering the bits from left to right, the IDR will contain the information shown on the following page.

bit	
1-3	The address of the next PSR syllable. This syllable should be an operator syllable, since the transfer to control mode can occur only at the end of an instruction. The PSR1 syllables are numbered, from the most significant end, 3-2-1-0, and the PSR2 syllables, 7-6-5-4.
4	A "one" if a repeated instruction was interrupted.
5	A "one" if a repeated instruction was interrupted before execution of the first iteration.
6-7	A "one" for each PSR (if any) that is presently filled (bit 6 for PSR1 and bit 7 for PSR2). If the last syllable of a PSR was used as the last syllable of the instruction before interrupt, then this PSR is no longer filled; if the last syllable was not used, this PSR is filled. If overlap had occurred, then the other PSR is filled, otherwise it is not. When these bits are restored, if both of these are in the "one" state due to overlap, one of them will be reset since the overlap has been lost.
8-10	The contents of the POV, PUN, and PNN flip-flops, respectively.
11-12	The address of the top of the stack, numbered 0 thru 3 (the contents of the stack counter).
13	A "one" if the computer was operating in the control mode when primary power failure was recognized.
14	A "one" if the interrupt condition is primary power failure.
15	A "one" for reversed counting of the stack counter.
16	Not used.

Bits 1 thru 10 are the only ones of interest to an AOSP routine and consequently are the only ones whose corresponding flip-flops are restored as a result of the IRR instruction.

d. The control mode flip-flop is set, thus marking transfer to the control mode, changing the interpretation of certain instructions and interrupt conditions, and temporarily preventing the processing of other interrupt conditions that may occur, with the exception of the two highest priority interrupt conditions.

e. The BAR and BPR are each loaded with the contents of the IAR. Note that the contents of the IAR can be altered only during control mode operation.

f. The effective address is computed, by adding the relative address, associated with the specific interrupt condition to be processed,* to the contents of the IAR, and is stored in the PCR. The contents of the location specified by this effective address will be one of a list of instructions, all unconditional transfers, to facilitate entry to the appropriate AOSP routine.

g. The bit in the interrupt register, corresponding to the interrupt condition about to be processed, is reset.

h. The POV, PUN, PNN and all other necessary control flip-flops are reset to allow the control mode program to use them without first resetting them. In the event of arithmetic overflow interrupt, the servicing of the overflow is performed by the control mode program. The POV is therefore reset as soon as the interrupt bit is set, since the control mode program would of necessity have to reset the corresponding bit from which it is loaded on return to normal mode, in order to prevent an interrupt loop.

---

*This relative address is the interrupt register bit number plus 1. For example, the RTC overflow effective interrupt address is the contents of the IAR plus 6.

APPENDIX A

D825 INSTRUCTION EXECUTION TIMES (in μsec)

ACE:	0.33	first bit different
	2.33	first bit alike
ACG:	0.33	first bit different
	2.33	first bit alike
ACL:	0.33	first bit different
	2.33	first bit alike
AIF:	3.00 + 0.33 (n-1)	n = number of adjustments (1 through 7)
	3.00	if n = 0

AUTOMATIC INTERRUPT JUMP: 7.00

BAD:	1.33	signs alike
	2.00	signs different
BAF:	8.33 + 0.33 (n-1)	n = number of adjustments (1 through 7)
	8.33	if n = 0

BDV:	53.67	
BMU:	24.00 + 0.67n	n = number of 1's in the multiplier (A_1), excluding 1's in bit positions 1, 12, 24, 36, 48.
BRB:	0.33	
BRC:	0.33	
BSF:	8.33 + 0.33 (n-1)	n = number of adjustments (1 through 7)
	8.33	if n = 0
BSU:	1.33	signs different
	2.00	signs alike
CBF:	1.33 + 0.33n	n = number of shifts-left needed to normalize
	0.33	if the number is zero
CEF:	4.67 + 0.33 (n-1)	n = number of adjustments (1 through 7)
	4.67	if n = 0
CEQ:	0.33	signs different
	2.33	signs alike
CGF:	4.67 + 0.33 (n-1)	n = number of adjustments (1 through 7)
	4.67	if n = 0

CGR:	0.33	signs different
	2.33	signs alike
CLA:	0.00	
CLF:	4.67 + 0.33 (n-1)	n = number of adjustments (1 through 7)
	4.67	if n = 0
CLS:	0.33	signs different
	2.33	signs alike
CSE:	1.33 + 0.33n	n = number of character positions examined (1 through 8)
FAD:	6.67 + 0.33 (n+m)	n = number of shifts-right needed (in units of 1, 6, & 12 bits) for lining up the operands, and m = number of shifts-left needed to normalize the result. n = 0 if the absolute value of [the (A_1) exponent minus the (A_2) exponent] is ≥ 35.
	2.33	The signs of the exponents of the 2 operands are different, and the absolute value of the sum of the two exponents is greater than 2,047.
FDV:	43.67	No quotient overflow
	44.33	Quotient overflow of 1 position has occurred and been corrected.
	1.33	Exponent overflow or underflow has occurred.

FMU:	21.00 + 0.67n + 1.33	(if normalization of 1 position is necessary)
		n = number of 1's in the mantissa of the multiplier (A_1), excluding 1's in bit positions 1-13, 24, 36, 48.
	1.67	Exponent overflow or underflow has occurred.
FSU:	6.67 + 0.33 (n+m)	n = number of shifts-right needed (in units of 1, 6, & 12 bits) for lining up the operands, and m = number of shifts-left needed to normalize the result. n = 0 if the absolute value of [the (A_1) exponent minus the (A_2) exponent] is $\geq$ 35.
	2.33	The signs of the exponents of the 2 operands are different, and the absolute value of the sum of the two exponents is greater than 2,047.
HLT:	0.00	
IRR:	5.67	
LAF:	7.00 + 0.33 (n-1)	n = number of adjustments (1 through 7)
	7.00	if n = 0
LAN:	0.33	
LCF:	5.00 + 0.33 (n-1)	n = number of adjustments (1 through 7)
	5.00	if n = 0

LCM:	0.33	
LOF:	7.00 + 0.33 (n-1)	n = number of adjustments (1 through 7)
	7.00	if n = 0
LOR:	0.33	
LSR:	0.33	
LTF:	0.67	1 register addressed
	2.00	4 12-bit registers addressed
	4.00	3 16-bit registers addressed
LXF:	7.00 + 0.33 (n-1)	n = number of adjustments (1 through 7)
	7.00	if n = 0
LXR:	0.33	
NOP:	0.00	
RPT:	1.00	
RVS:	0.00	

SAF:	1.33 + 0.33 (n-1)	n = number of adjustments (1 through 7)
	1.33	if n = 0
SER:	0.00	
SHF:	0.67 + 0.33n (Single)	n = number of shifts-left, or the number of shifts-right in units of 1, 6, and 12 bits.
	5.33 + 0.67 (n-1) (Double)	n = number of shifts; n is limited to 11 for the 2 left arithmetic shifts and to 12 for the other 6 double shifts.
	0.67	shift of zero
SRJ:	5.33	no overlap of program word fetch had occurred
	6.00	overlap of program word fetch had occurred
SRR:	4.00	
STF:	2.00	12-bit register(s) addressed
	4.33	1 16-bit register addressed
	7.33	3 16-bit registers addressed
SSD:	0.00	
SSU	0.00	

TIO:	0.00	
TRM:	0.33	sign modification
	2.67	round, with or without sign modification
TRS:	0.00	
UCT:	0.00	
XLC:	3.33	

APPENDIX B

D825 TIMING ALGORITHMS

Two timing algorithms and a program which illustrates the use of the algorithms (and D825 programming) are presented in this appendix. The first algorithm is used to calculate accurate program running time, and the second algorithm, which is more simple than the first, may be used to calculate approximate running time.

TIMING ALGORITHM #1

This algorithm can be used to calculate the accurate running time for D825 programs and is illustrated in tables B-1 and B-2.

Sum of instruction execution times
+ 2.00 μsec per stack read*
+ 1.67 μsec per stack write
+ 4.33 μsec per level of indirect addressing**
+ 1.67 μsec per repeated instruction***
+ the individual processing time for each type of syllable used:
  1.33 μsec per program operator syllable
  5.33 μsec per memory syllable (read)
  4.33 μsec per memory syllable (write)
  0.33 μsec per branch (if executed)
  0.67 μsec per branch (if not executed)
  1.67 μsec per index syllable
  1.67 μsec per any other type of syllable
+ 5.00 μsec per program word - fetch**
- 5.00 μsec per program word - fetch overlap

*0.33 μsec. for a stack read if it immediately follows a stack write of the preceding instruction.

**Counted only the first time if repeated.

***Not counted the first time.

(Thin film storage provides enough space for 2 full program words; if 1 word is empty and the present instruction is lengthy in execution, the 2nd word can be filled during the execution. This overlap saves 5.00 μsec. All floating point instructions, as well as BDV and BMU, allow overlap.)

TIMING ALGORITHM #2

This abridged algorithm provides a good estimate of program running times:

Sum of instruction execution times

+10 μsec per program word (each 4 syllables) processed
+ 4 μsec per operand or level of indirect addressing
+ 2 μsec per operand stack reference
+ 2 μsec per indexed address
- 5 μsec per overlap

The abridged timing algorithm may be used with confidence for all tasks where the average instruction execution times are used.

AN EXAMPLE OF THE APPLICATION OF ALGORITHM #1

Given four columns of numbers, each of length "n", which are stored in floating-point format in 4n consecutive memory words.

Location	Contains	Symbolic Notation
BAR + Col 1	Column 1-1st number	A_1
BAR + Col 1 + 1	Column 1-2nd number	A_2
-	-	-
-	-	-
-	-	-
BAR + Col 1 + n-1	Column 1-nth number	A_n
BAR + Col 1 + n	Column 2-1st number	X_1
-	-	-
-	-	-
-	-	-
BAR + Col 1 + 2n-1	Column 2-nth number	X_n
BAR + Col 1 + 2n	Column 3-1st number	Y_1
-	-	-
-	-	-
-	-	-
BAR + Col 1 + 3n-1	Column 3-nth number	Y_n
BAR + Col 1 + 3n	Column 4-1st number	Z_1
-	-	-
-	-	-
-	-	-
BAR + Col 1 + 4n-1	Column 4-nth number	Z_n

Form three product - sums, by summing the products of each number in column 1 times the corresponding number in another column; the product - sums are to be stored in absolute memory locations 0, 1, and 2. Symbolically,

$$A_1 X_1 + A_2 X_2 + A_3 X_3 + \text{- - - - - - - - - - } A_n X_n \rightarrow \text{Location 0}$$

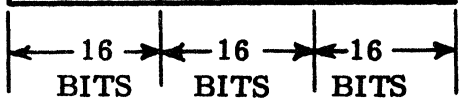
$$A_1 Y_1 + A_2 Y_2 + A_3 Y_3 + \text{- - - - - - - - - - } A_n Y_n \rightarrow \text{Location 1}$$

$$A_1 Z_1 + A_2 Z_2 + A_3 Z_3 + \text{- - - - - - - - - - } A_n Z_n \rightarrow \text{Location 2}$$

It is assumed that program overflow, program underflow, and program non-normalized will not occur.

Octal notation and coding symbols are used in place of binary format. Constants and storage used are described below.

Location	Contains	Use
BAR + a	n | 0←→0 | 0←→0	Index register loader; indirect address
BAR + b	0←→0 | 3n | n	Limit register loader



Index and limit register use

n → XR4	(XR5 increment)
0 → XR5	(column - address increment)
0 → XR6	(number - address increment)
0 → LR4	(not used)
3n → LR5	(XR5 limit)
n → LR6	(XR6 limit)

Program coding is given in table B-1, and the timing for the program words is shown in table B-2.

Table B-1. Program Coding

Word No.	Symbolic Address	Program Word				Comments
		Syllable 1	Syllable 2	Syllable 3	Syllable 4	
1	Start	LTF, 220	a	XR4, XR5, XR6	LTF, 220	Load XR's and LR's. Assume top of stack contains zero.
2		b	LR4, LR5, LR6	UCT, 200	(2)	
3	(1)	LTF, 220	a	XR6	CLA, 000	Re-load XR6; clear stack
4	(2)	FMU, 330	XR6	COL 1	XR4, XR5, XR6	(Col 1 number) (Col 2-4 number) ⟶ stack
5		COL 1	FAD, 000	XLC, 222	0 ⟷ 01	Add to former sum; address a new number
6		+, ≠, XR6, LR6	(2)	XLC, 322	XR4	Continue in sequence after n numbers; address a new column
7		0 ⟷ 0	+, ≠, XR5, LR5	(1)	RVS, 000	Continue in sequence after 3 columns; manipulate stack
8		SSU, 000	SSU, 000	RPT, 222	0 ⟷ 03	Transmit 3 product-sums to locations 0-2
9		1, 0, 0	(3)	UCT, 200	EXIT	Exit
10	(3)	TRS, 020	(I) a	———	———	(Repeated 3 times; (I) = indirect address)

Table B-2. Program Timing

Word No.	Number of Times Performed	Instruction Execution Time (Average)	Stack Reference	Indirect Address	Repeated Instruction	Syllables				Program Word-Fetch	Total (μsec)
						Operator	Memory	Branch	Other		
1	1	8.00				2.67	5.33		1.67	5.00	22.67
2	1					1.33	5.33	.33	1.67	5.00	13.67
3	2	0.67	1.67			2.67	5.33		1.67	5.00	34
4	3n	38.33	1.67			1.33	5.33		3.33	5.00	165 n
5	3n	15.67	4.00			2.67	5.33		1.67	5.00	103 n
6	Branch: 3n-3 No Branch: 3	3.33				1.33		.33 .67	1.67 3.33		6n - 6 26
7	Branch: 2 No Branch: 1					1.33		.33 .67	3.33 3.33	5.00 5.00	17.33 10.33
8	1	1.00				4.00			1.67	5.00	11.67
9	1					1.33		.67	1.67	5.00	8.67
10	1st time: 1 Other times: 2		2.00 2.00	4.00	1.67	1.33 1.33	4.33 4.33			5.00	16.67 18.67

Total = 274n + 173.67 μsec.

APPENDIX C

MAP OF THIN FILM REGISTERS

MAP OF 16-BIT T. F. REGISTERS

THIN FILM ADDRESS	OCTAL CODE	REGISTER NAME
0 0 0 0 0 0 0 ●	0 0 0	not used
0 0 0 0 0 0 1	1	1
0 0 0 0 0 1 0	2	2
0 0 0 0 0 1 1	3	3
0 0 0 0 1 0 0 ●	4	4
0 0 0 0 1 0 1	5	5
0 0 0 0 1 1 0	6	6
0 0 0 0 1 1 1	0 7	7
0 0 0 1 0 0 0 ●	1 0	8 } Index Registers 1-15
0 0 0 1 0 0 1	1	9
0 0 0 1 0 1 0	2	10
0 0 0 1 0 1 1	3	11
0 0 0 1 1 0 0 ●	4	12
0 0 0 1 1 0 1	5	13
0 0 0 1 1 1 0	6	14
0 0 0 1 1 1 1	1 7	15
0 0 1 0 0 0 0 ●	2 0	0 Spare
0 0 1 0 0 0 1	1	1 }
0 0 1 0 0 1 0	2	2
0 0 1 0 0 1 1	3	3
0 0 1 0 1 0 0 ●	4	4
0 0 1 0 1 0 1	5	5
0 0 1 0 1 1 0	6	6
0 0 1 0 1 1 1	2 7	7
0 0 1 1 0 0 0 ●	3 0	8 } Limit Registers 1-15
0 0 1 1 0 0 1	1	9
0 0 1 1 0 1 0	2	10
0 0 1 1 0 1 1	3	11
0 0 1 1 1 0 0 ●	4	12
0 0 1 1 1 0 1	5	13
0 0 1 1 1 1 0	6	14
0 0 1 1 1 1 1	0 3 7	15 }
0 1 0 0 0 0 0 ●	0 4 0	}
0 1 0 0 0 0 1	1	ISR - Interrupt Storage Register
0 1 0 0 0 1 0	2	}
0 1 0 0 0 1 1	3	Spare
0 1 0 0 0 1 0 0 ●	4	}
0 1 0 0 0 1 0 1	5	RPR - Repeat Program Register
0 1 0 0 0 1 1 0	6	}
0 1 0 0 0 1 1 1	4 7	}
0 1 0 0 1 0 0 0 ●	5 0	}
0 1 0 0 1 0 0 1	1	SSR - Subroutine Storage Register
0 1 0 0 1 0 1 0	2	}
0 1 0 0 1 0 1 1	3	Spare
0 1 0 0 1 1 0 0 ●	4	BPR - Base Program Register
0 1 0 0 1 1 0 1	5	BAR - Base Address Register
0 1 0 0 1 1 1 0	6	Spare
0 1 0 0 1 1 1 1	5 7	PCR - Program Count Register
0 1 0 1 0 0 0 0 ●	6 0	SAR - Subroutine Base Address Register
0 1 0 1 0 0 0 1	1	Spare
0 1 0 1 0 0 1 0	2	XIR - Index Increment Register
0 1 0 1 0 0 1 1	3	IAR - Interrupt Base Address Register
0 1 0 1 0 1 0 0 ●	4	} PDR - Power Failure Dump Register
0 1 0 1 0 1 0 1	5	}
0 1 0 1 0 1 1 0	6	} Spare
0 1 0 1 0 1 1 1	6 7	}
0 1 0 1 1 0 0 0 ●	7 0	IDR - Interrupt Dump Register
0 1 0 1 1 0 0 1	1	}
0 1 0 1 1 0 1 0	2	}
0 1 0 1 1 0 1 1	3	}
0 1 0 1 1 1 0 0 ●	4	} Spare
0 1 0 1 1 1 0 1	5	}
0 1 0 1 1 1 1 0	6	}
0 1 0 1 1 1 1 1	0 7 7	}

● Starting address for 3-register blocks

MAP OF 12-BIT T. F. REGISTERS

THIN FILM ADDRESS	OCTAL CODE	REGISTER NAME
1 0 0 0 0 0 0 ●	1 0 0	PSR1 - Program Storage Register #1
0 0 1	1	
0 1 0	2	
0 1 1	3	
1 0 0 ●	4	PSR2 - Program Storage Register #2
1 0 1	5	
1 1 0	6	
0 1 1 1	0 7	
1 0 0 0 ●	1 0	IPR - Interrupt Program Register
0 0 1	1	
0 1 0	2	
0 1 1	3	
1 0 0 ●	4	RTC - Real Time Clock
1 0 1	5	
1 1 0	6	Spare
0 1 1 1 1	1 7	
1 0 0 0 0 ●	2 0	RCR - Repeat Count Register
0 0 1	1	Spare
0 1 0	2	Spare
0 1 1	3	CCR - Character Count Register
1 0 0 ●	4	
1 0 1	5	TFC - Thin Film C Register
1 1 0	6	
0 1 1 1	2 7	
1 0 0 0 ●	3 0	RIR - Repeat Increment Registers
0 0 1	1	
0 1 0	2	
0 1 1	3	
1 0 0 ●	4	
1 0 1	5	Spare
1 1 0	6	
1 0 1 1 1 1 1	1 3 7	
1 1 0 0 0 0 0 ●	1 4 0	
0 0 1	1	STK1
0 1 0	2	
0 1 1	3	
1 0 0 ●	4	
1 0 1	5	STK2
1 1 0	6	
0 1 1 1	4 7	Operand Stack Registers
1 0 0 0 ●	5 0	
0 0 1	1	STK3
0 1 0	2	
0 1 1	3	
1 0 0 ●	4	
1 0 1	5	STK4
1 1 0	6	
0 1 1 1 1	5 7	
1 0 0 0 0 ●	6 0	
0 0 1	1	
0 1 0	2	
0 1 1	3	
1 0 0 ●	4	
1 0 1	5	
1 1 0	6	
0 1 1 1	6 7	Spare
1 0 0 0 0 ●	7 0	
0 0 1	1	
0 1 0	2	
0 1 1	3	
1 0 0 ●	4	
1 0 1	5	
1 1 0	6	
1 1 1 1 1 1 1	1 7 7	

● Starting address for 4-register blocks

APPENDIX D

INSTRUCTION INDEX

MNEMONIC AND OCTAL	INSTRUCTION NAME	SYLLABLE LAYOUT A_1 A_2 A_3	SYMBOLIC DESCRIPTION	PAGE
ACE-72	Alphanumeric Compare Equal	0 (M) (M) B	$(A_1) = (A_2) : A_3$ (alphanumeric)	4-26
ACG-71	Alphanumeric Compare Greater	0 (M) (M) B	$(A_1) > (A_2) : A_3$ (alphanumeric)	4-26
ACL-70	Alphanumeric Compare Less	0 (M) (M) B	$(A_1) < (A_2) : A_3$ (alphanumeric)	4-27
AIF -40	Adjust and Insert, Field	0 (M) F (M)	—	4-19
BAD-65	Binary Add	0 (M) (M) (M)	$(A_1) + (A_2) \longrightarrow A_3$	4- 3
BAF-43	Binary Add, Field	0 (M) F (M)	—	4-14
BDV-60	Binary Divide	0 (M) (M) (M)	$(A_1) \div (A_2) \longrightarrow A_3$	4- 4
BMU-61	Binary Multiply	0 (M) (M) (M)	$(A_1) \times (A_2) \longrightarrow A_3$	4- 3
BRB-26	Branch on Bit	0 (M) (M) B	—	4-27
BRC-11	Branch on Condition	0 L B	$A_1 : A_2$	4-27
BSF-42	Binary Subtract, Field	0 (M) F (M)	—	4-16
BSU-64	Binary Subtract	0 (M) (M) (M)	$(A_1) - (A_2) \longrightarrow A_3$	4- 3
CBF-25	Convert Binary to Floating Point	0 (M) (M)	$(A_1) \longrightarrow A_2$, floating	4- 4
CEF-52	Compare Equal, Field	0 (M) F B	—	4-20
CEQ-76	Compare Equal	0 (M) (M) B	$(A_1) = (A_2) : A_3$ (algebraic)	4-25
CGF-51	Compare Greater, Field	0 (M) F B	—	4-20
CGR-75	Compare Greater	0 (M) (M) B	$(A_1) > (A_2) : A_3$ (algebraic)	4-26
CLA-20	Clear	0 (M)	zeroes $\longrightarrow A_1$	4-14
CLF-50	Compare Less, Field	0 (M) F B	—	4-20
CLS-74	Compare Less	0 (M) (M) B	$(A_1) < (A_2) : A_3$ (algebraic)	4-26
CSE-32	Character Search	0 (M) C B	$(A) = A_2 : A_3$	4-21
FAD-67	Floating Add	0 (M) (M) (M)	floating, $(A_1) + (A_2) \longrightarrow A_3$	4- 5
FDV-62	Floating Divide	0 (M) (M) (M)	floating, $(A_1) \div (A_2) \longrightarrow A_3$	4- 8
FMU-63	Floating Multiply	0 (M) (M) (M)	floating, $(A_1) \times (A_2) \longrightarrow A_3$	4- 7
FSU-66	Floating Subtract	0 (M) (M) (M)	floating, $(A_1) - (A_2) \longrightarrow A_3$	4- 6
HLT-01	Halt	0	—	4-33
IRR -05	Interrupt Return	0	—	4-36
LAF-47	Logical AND, Field	0 (M) F (M)	—	4-16
LAN-56	Logical AND	0 (M) (M) (M)	$(A_1) \cdot (A_2) \longrightarrow A_3$	4-23
LCF-46	Logical COMPLEMENT, Field	0 (M) F (M)	—	4-17
LCM-24	Logical COMPLEMENT	0 (M) (M)	$(A_1) \longrightarrow A_2$	4-23
LOF-44	Logical OR, Field	0 (M) F (M)	—	4-16
LOR-55	Logical OR	0 (M) (M) (M)	$(A_1) \vee (A_2) \longrightarrow A_3$	4-23
LSR-31	Load Special Register	0 (M) Vs	—	4-33
LTF-30	Load Thin Film	0 (M) T	$(A_1) \longrightarrow A_2$, thin film	4-11
LXF-45	Logical EXCLUSIVE OR, Field	0 (M) F (M)	—	4-16
LXR-54	Logical EXCLUSIVE OR	0 (M) (M) (M)	$(A_1) \oplus (A_2) \longrightarrow A_3$	4-23
NOP-00	No Operation	0	—	4-29
RPT-10	Repeat	0 Rc Ri B	Repeat (A_3), by A_2, A_1 times	4-31
RVS-06	Reverse Stack	0	—	4-11
SAF-41	Strip and Adjust, Field	0 (M) F (M)	—	4-18
SER-21	Store External Requests	0 (M)	—	4-14
SHF-36	Shift	0 (M) S (M)	(A_2) shifted, by A_1, $\longrightarrow A_3$	4-24
SRJ -14	Subroutine Jump	0 Ja Ji	—	4-29
SRR-04	Subroutine Return	0	—	4-31
STF-15	Store Thin Film	0 T (M)	Thin film, $(A_1) \longrightarrow A_2$	4-10
SSD -03	Step Stack Down	0	$s - 1 \longrightarrow s$	4-12
SSU -02	Step Stack Up	0	$s + 1 \longrightarrow s$	4-12
TIO -16	Transmit to Input/Output	0 IO M B	—	4-35
TRM-34	Transmit Modified	0 (M) Vt (M)	(A_1), by A_2, $\longrightarrow A_3$	4- 9
TRS-35	Transmit	0 (M) (M)	$(A_1) \longrightarrow A_2$	4- 9
UCT-22	Unconditional Transfer	0 B	: A_1	4-27
XLC-12	Index, Limit - Compare	0 Ia Iv B	—	4-12