

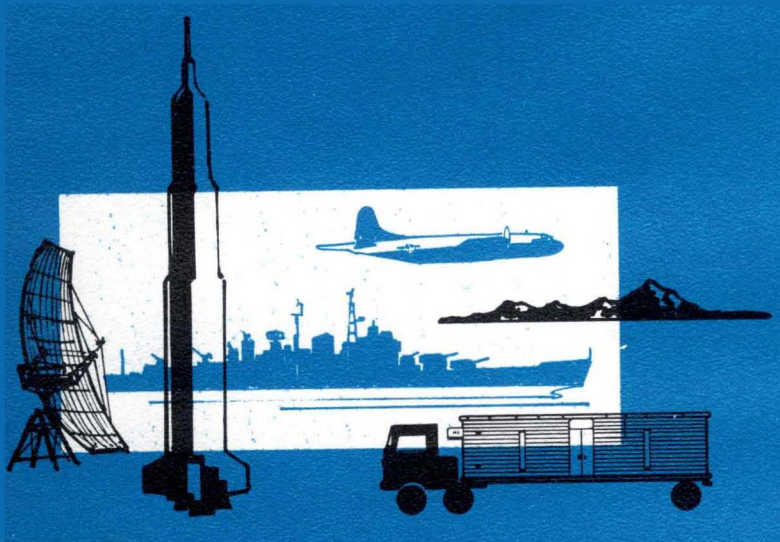
UNIVAC[®]

1230

**MILITARY
COMPUTER**



PROGRAMING MANUAL



PROGRAMING MANUAL

FOR

1230 COMPUTER

(15 BIT MODE)

PX 3892

FEBRUARY 1966

UNIVAC

DIVISION OF SPERRY RAND CORPORATION

Copyright © 1965 by the Sperry Rand Corporation. Printed in the United States of America.
All rights reserved. This book, or parts thereof, may not be reproduced in any form without
permission of the Sperry Rand Corporation.

PREFACE

In accord with UNIVAC's policy of maintaining compatibility in its computer designs, thus minimizing transition time and costs as users upgrade and expand their systems, the 1230 Computer has been designed to operate with CP-642B Software and operational programs. Mechanically, this is accomplished through the setting of a console switch to operate the computer in the 15-bit mode.

When operating the 1230 in 15-bit mode additional instructions are available which are not included with the standard CP-642B and, significantly, instruction execution times are at least twice as fast.

Improvements in the 1230 include:

- Twice as fast execution times of instructions
- Continuous data mode data transfers
- Externally specified index
- Externally specified address
- 128 or 256 (as opposed to 64 in the CP-642B) word thin film control memory
- Overlapping memory banks

Any reference to "the computer" in this manual refers to the UNIVAC 1230 Computer.

For additional information concerning the 1230 system, refer to the following 1230 manuals:

Compiling System Manual	PX 3890
Operating and Support Manual	PX 3891

CONTENTS

The modular nature of this manual makes a normal table of contents impractical. Instead, the manual has color-coded index tabs to give an overview of the contents and provide for easy access to any portion of the manual. Each index tab identifies the section of the manual that follows it. Where appropriate, individual sections list the contents of the section. Index tab colors have the following meaning:

- Blue index tabs - main sections
- White index tabs - first level of subsection
- Silver index tabs - second level of subsection

COMPUTER CHARACTERISTICS

INTRODUCTION

The UNIVAC 1230 Computer is a compact, medium-scale, military, stored-program machine. It is a powerful real-time device designed for rapid processing of continuous high-rate, through-put data. The computer is especially suited for such real-time applications as missile guidance, range safety, process monitoring, and tactical control and display. Relative to other general-purpose systems, the computer emphasizes rapid communication between external devices and a large internal random access storage. It is equipped with a 400-nanosecond control memory operating in synchronism with the 2-microsecond main core memory. The main memory further enhances the operational speed of the computer by its ability to operate as two parallel, overlapped banks, thereby producing an effective cycle time of 1 microsecond. This advanced design feature results in a command execution time as low as 2 microseconds and an input/output transfer capability of 500,000 30-bit words per second. The computer (complete with 32K 30-bit words of random access core storage, 128 or 256 words of control memory, 16 input channels and 16 output channels under full buffer and/or real-time control, arithmetic section, control circuitry, power supply, and operator's or maintenance panel) is packaged in a ruggedized cabinet occupying less than 60 cubic feet of space.

The computer has in its repertoire 77 instructions that specify basic input/output, arithmetic, or logical operation to be performed. Computer operation is fully automatic because the sequence of operation is determined by a program of internally stored single-address instructions capable of self-modification. To attain high computing speed, the computer operates in a parallel mode, i.e., all the digits of a word are operated upon simultaneously. It can process large quantities of constantly changing complex data at high speeds on a real-time basis. The basic word is 30 binary digits or bits. Optional operation with 15-bit half words may be used. A word may be an instruction, number, or an arbitrarily coded quantity. The core memory cycle time, or the time needed to read and restore 1 word, is 2 microseconds. The readout time, or the time between a given function assuming control of memory and the delivery of the data from memory, is approximately 0.9 microsecond.

The versatile input/output system of the computer permits it to operate on-line with associated communications peripheral equipment, other computers, and standard peripheral input/output equipment on a time-shared basis. Communications with associated peripheral equipment are handled by block transfer of data or by real-time events requesting one or more words. Once initiated, neither of these requires further main program attention unless immediate processing is demanded.

The 1230 Computer is designed to operate as a central processor of a system supplying up to 32K words of memory in its main frame or it can, by its addressing structure, command an additional external overlapped bank memory unit that can contain up to five 16,384-word modules of core memory and four additional input/output channels. A multi-processing capability is a further objective of the design since two 1230 Computers may be

connected to the external memory unit, each having access to the overlapped memory modules and the four input/output channels on a time shared basis. (See Figure 1.)

Thus, each of the 1230 Computers has access to its own dual bank core memory and any one of the five memory modules of the external memory at any time. Logical memory lockout and priority is provided to prevent interference.

COMPUTER FEATURES

TYPE

General-purpose, medium-scale, solid-state, parallel, binary

MEMORY

Main Memory—Main Frame

Magnetic core - random access

- 2 overlapped banks
- 0.9 microsecond read access time, coincident current
- 2.0 microsecond read-write cycle time (effectively 1 microsecond)
- 30-bit word length, parallel transfers
- 32,608 directly addressable, half or full word operands

Certain locations are assigned in the first 653 addresses for the Interrupt Entrance registers

Control Memory

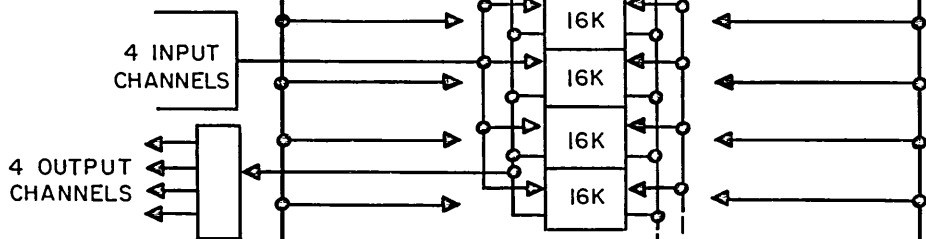
Word oriented magnetic thin film

- 400-nanosecond cycle time
- 30-bit word length, parallel transfers
- 128 words, directly addressable, half or full word operands
- 256 words, optional

Directly addressable as cells 00100 to 00277 or 00477; used for storage of buffer control words, 7 index registers, and the real-time clock; operates independently and concurrently with main memory

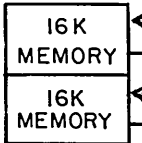
OPTIONAL EXTERNAL MEMORY

ESI NOT AVAILABLE
 ESA ON ALL 4 CHANNELS
 CDM ON CHANNEL 3 ONLY

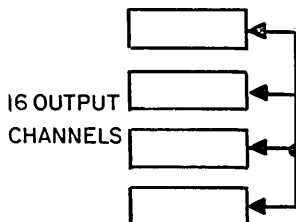


ESI }
 ESA } ALL CHANNELS
 CDM }

MEMORY SELECT



CONTROL



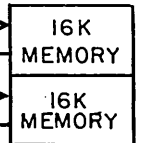
16 INPUT CHANNELS

ARITHMETIC

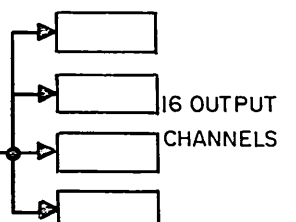
UNIVAC 1230 COMPUTER A

ESI }
 ESA } ALL CHANNELS
 CDM }

MEMORY SELECT



CONTROL



16 INPUT CHANNELS

ARITHMETIC

UNIVAC 1230 COMPUTER B

Figure 1. Dual Computer Configuration

NDRO Memory

Word oriented transformer cores

- 300-nanosecond read time
- 30-bit word length, parallel transfers
- 2 32-word memories

Two nondestructive memories provided for initial load/error recovery purposes; selected by switch at the console; addressable as cells 540-577

ARITHMETIC

Organization

30-bit parallel, one's complement integer binary

Registers

Two 30-bit registers, addressable as A and Q, may be linked as one 60-bit register

Functions

- Arithmetic operations include add, subtract, multiply, divide, and square root
- Fifteen logical operations

Instruction Execution Timing (including indexing)

- Add, subtract, logical - 2 to 4 microseconds
- Multiply - 8 microseconds
- Divide - 14 microseconds
- Square root - 8 microseconds
- Compare, mask compare, and branch - 2 to 4 microseconds
- Register shifts - 2 to 4 microseconds (maximum)

CONTROL

- Single address instruction organization
- Branching according to six algebraic conditions of A and Q available on most instructions
- Control memory operates in the shadow of main memory and normally does not add to execution time

- Half-word addressing
- Address modification via seven thin-film-memory-contained index registers
- Sequential execution of instructions
- Provides timing and addressing structure for internal and external core memory references

INPUT/OUTPUT

Channels

Sixteen input and 16 output channels provide input/output transfers under full buffer control, do not require program attention, and operate asynchronously at the rate required by external devices. Words located in the Control Memory guide data transfers which require only two microseconds of main memory time for each 30-bit word transferred in or out.

Intercomputer Transfers

Thirty-bit parallel data transfers allow direct communication with UNIVAC CP-642A, CP-642B, 1218, 1219, CP-667, and other compatible computers.

Transfer Rate

- (Fast Interface) One channel - 166,667 30-bit words per second (maximum)
 3 or more channels - 500,000 30-bit words per second (maximum)
- (Slow Interface) One channel - 41,666 30-bit words per second (maximum)
 Multichannel - 333,333 30-bit words per second (maximum)

Real-Time Clock

Internal, 1024 cps, with a provision for external substitution

Interrupts

Eighty-one unique interrupts are provided as follows:

- 1 Fault Interrupt
- 16 External Interrupts (one per channel)
- 16 External Function Monitor Interrupts (one per channel)
- 16 Output Monitor Interrupts (one per channel)
- 16 Input Monitor Interrupts (one per channel)
- 16 Intercomputer Timeout Interrupts (one per channel)

Priority

Priority of interrupts is determined according to channel, with a subpriority evaluation in case of ties according to function in the order listed above.

Input/Output Control

Ten basic program instructions are devoted to the control of input/output, providing positive control and a high degree of sophistication in programing.

Continuous Data Mode (CDM)

This feature allows automatic reinitiation of previously established buffers under program control. The termination of the buffer is also program controlled.

Externally Specified Addressing (ESA)

This feature enables a data word to be stored or read from an address directly specified by an external device.

Externally Specified Indexing (ESI)

ESI is used to transfer data words, indirectly specified by the external device; that is, the external device specifies the address of the buffer control words for this particular transfer.

ORGANIZATION

Internal storage of the computer main frame consists of two 16K-word, ferrite-core main memory banks operating simultaneously. A complete cycle for storage or retrieval of two 30-bit words requires two microseconds. An additional storage area, designated as control memory, provides 128 addressable locations with a read/restore cycle time of 400 nanoseconds.

Single address instructions are employed, most of which have an execution time of four to six microseconds when programed in a single bank; two to four microseconds when the dual banks feature is utilized.

Arithmetic and logical operations are performed in the parallel binary mode. For most operations, the result appears in a 30-bit accumulator register. Arithmetic is one's complement, subtractive with a modulus ($2^{30}-1$). Computer operation is controlled by a stored program capable of self-modification. Each program instruction contains a function code (6 bits), an instruction operand designator (13 or 15 bits), and three or four execution modifiers (2, 3, or 4 bits). Execution modifiers provide for branch point designation, operand interpretation, input/output channel, minor function, address modification, or memory bank designation. The operand designator may be a simple constant that may or may not be modified by the contents of an index register, or it may be a partial execution address extended as dictated by the memory bank designator with or without index modification. The operand specified by the execution address may be interpreted as a 30-bit quantity or as a 15-bit half word with or without sign extension. The next sequential program step may be skipped if the arithmetic, logical or key position condition specified by the branch condition designator is met.

The 1230 Computer can be operated within its 32K internal memory as a CP-642B Computer in which case the Special Registers are ignored or it can be operated with the full capabilities of the 1230 thereby extending its memory addressing capacity to a maximum of 114K words.

Communications between the computer and its associated external equipment are normally accomplished by a buffered transfer of data, with timing under control of the external device. Operating asynchronously with the main computer program, such transfers of data have independent access to storage. The number of data words transferred is under program control by specifying the first and last memory address in the buffer.

The input/output section of the computer is capable of communicating with other UNIVAC Military Computers and with other military and commercial peripheral equipment.

A communication path is established by a sequence of request and acknowledge signals between external equipment and computer. The communication may be initiated by either the computer or the external device. External request signals interrupt the main computer program and cause the computer to establish a communications channel between the external equipment and

the computer core memory. Once the communication line has been created, the computer returns to the main program sequence, and transfer of input or output data proceeds without program reference until completed.

Up to 16 input and 16 output channels are provided in the computer main frame; each channel consists of 30 parallel data lines plus control lines.

- Any input/output channel is capable of communicating with either another computer, or by changing printed circuit jumper cards, with peripheral equipment. (Either fast or slow interface may be used, depending upon the type of I/O amplifiers plugged into the chassis.)

A group of four channels may be converted from fast to slow interface and vice versa by simply changing plug-in, printed circuit cards within the channel circuitry.

In addition to data words, output channels carry external function words to the external equipment. These words specify the function that the external device is to perform. Control of the external function transfer is accomplished in the buffer mode. This feature allows the computer to continue engaging an external device after completion of each function. An external function word to a tape control unit, for example, may specify Rewind Tape Unit 2 . When Tape Unit 2 has informed the computer that the operation has been initiated, the computer can respond by transmitting another external function word, for example, Write on Tape Unit 1 without program interruption.

Transfers of input and output data are controlled by priority and access control logic. The computer is designed to give first priority to channels and secondary priority to function. By circuit card placement, it is possible to assign a priority to any chassis of four channels, and then to the channels in that chassis. In this new method it is possible to assign top selection to any channel and varying degrees of lesser priority to the other channels.

The descending order of function priority is as follows:

- Advance Real-Time Clock
- External Interrupt
- External Function Request
- Output Request
- Input Request

- Intercomputer Monitor Interrupt
- External Function Monitor Interrupt
- Output Monitor Interrupt
- Input Monitor Interrupt

CONTROL SECTION

The Control Section (see Figure 2) consists of those registers and circuits necessary to procure, modify, and execute the instructions of the program.

The U register (30 bits) is the program control register. It holds the instruction word during execution of an instruction. The function code and the various execution modifiers are translated from appropriate sections of the register. The lower order 13 or 15 bits of the U register have additional properties, modulus $2^{15}-1$ or $2^{17}-1$.

The 17-bit R and B registers in the control section are nonaddressable communications registers which work in conjunction with the Special Registers, the P register, the U_L register, and the Control and nondestructive readout (NDRO) memories. The B register supplies the indexing value taken from the control memory index registers (B_1 through B_7) to the control address while the R register supplies the natural or assembled extended y address. During input or output transfers, the R and B registers are used in address extension, comparison, and incrementation.

The 17-bit B registers, B_1 through B_7 , store the quantities used for U_L modification. These B registers, also called index registers, occupy the lower 17 bits of control memory addresses (161-167) in the expanded memory mode and the lower 15 bits in the 32K memory mode.

The P register (17 bits) holds the memory address of a computer instruction word--that of the current instruction at the beginning of the instruction sequence which is incremented to the address of the next sequential instruction.

The S registers (14 bits) hold the storage addresses during main memory references. At the beginning of a memory cycle period, the address is transferred to the S register involved for the particular 16K module. The contents of the appropriate S register is then translated to activate the storage selection system.

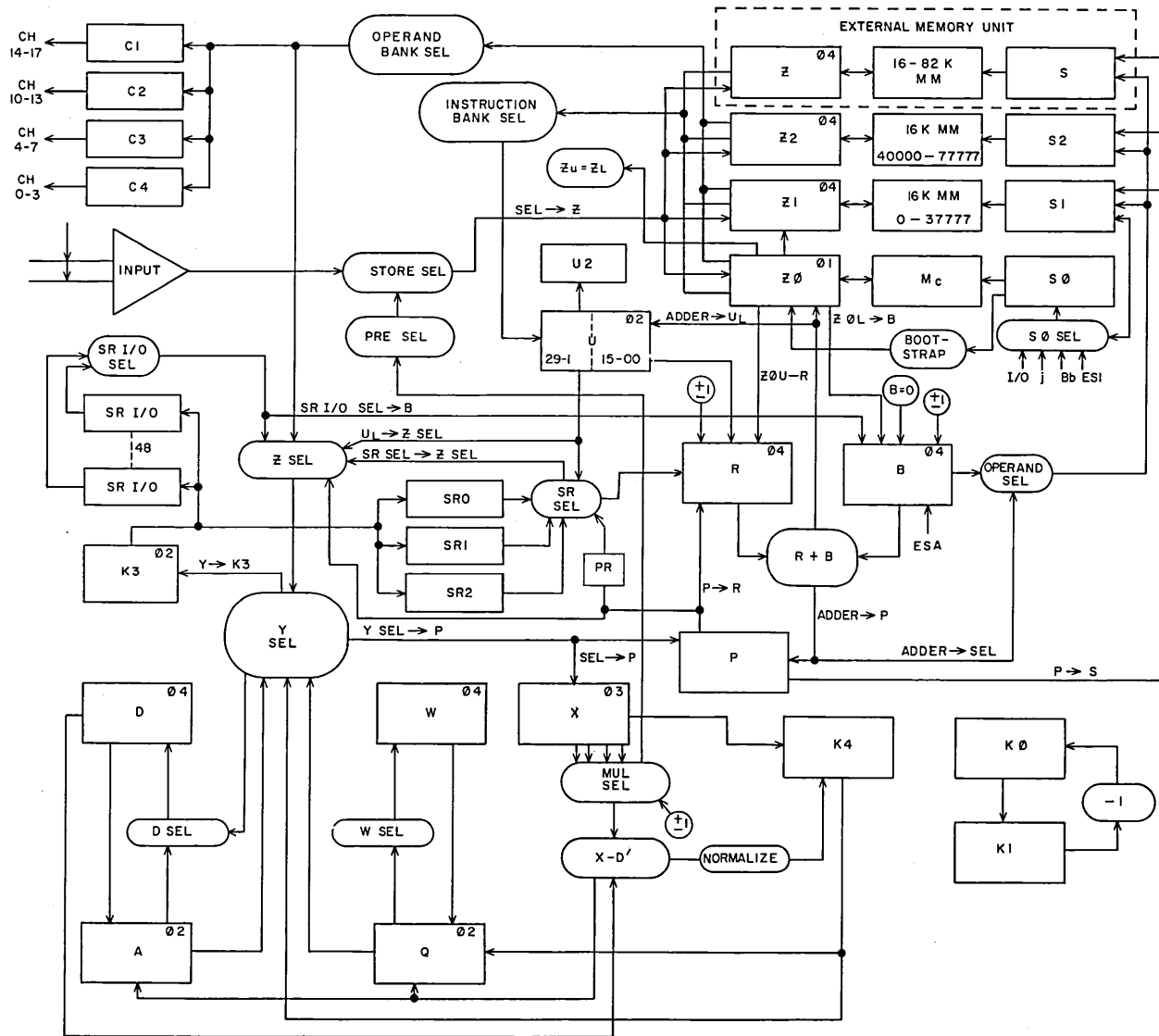


Figure 2. Simplified Block Diagram of 1230 Computer

The SO-register (eight bits) acts in the same manner as an S-register except that it holds the address for control memory and NDRO memory during the memory cycle time.

The K registers function as shift control and counter registers for all arithmetic operations that involve shifts. Other instructions employing the K registers are multiply, divide, and square root.

ARITHMETIC SECTION

The arithmetic section performs numeric and logical calculations. Although greatly simplified, Figure 6 is a block diagram of the 1230 Computer.

The A register (30 bits) may be considered for programming purposes, as a conventional accumulator. Because of the logic employed, however, the A register is actually only the main rank of the accumulator; the D register serves as a second rank.

The add operation is typical of the relationship between the A and D registers: the augend and addend are initially contained in the A and D registers. Before the addition is performed, the contents of the A register are transmitted to the X register. The values of X and D are combined by the add network to form the sum of the two numbers in a parallel manner and placed in the A register.

The Q register (30 bits) is used principally during multiply and divide operations. The contents of both A and Q may be shifted left or right, either individually or as one double-length, 60-bit word.

The X, D, and W registers are 30-bit, nonaddressable registers. These registers are used primarily for the exchange of data within the arithmetic section and for communicating with the remaining sections of the computer. The W register is not displayed on the control panel of the computer; the A, Q, X, and D registers have indicators which allow the operator to inspect the contents of these registers during debugging and maintenance operations. The A and Q registers are addressable arithmetic registers.

STORAGE SECTION

The storage section consists of three basic memories:

- Main storage section constructed of modular arrays of ferrite cores
- Control memory constructed from magnetic thin-film elements
- Bootstrap memory, a nondestructive readout type, which utilizes the transformer core type of storage

The main frame storage section has a capacity of 32,768 30-bit words. It is divided into two 16K modules that operate concurrently in time so that two memory references can be made during each two-microsecond read/restore cycle. The main storage is coincident-current driven destructive read-out core memory. The Z registers (30 bit) are buffer registers through which all information to and from core locations must pass. Because of the optional use of 15-bit half word operand, the Z registers are split into two 15-bit sections termed Z-upper and Z-lower.

The control memory of 128 30-bit words (256 optional) is a word oriented magnetic thin-film memory with a read/restore cycle time of 400 nanoseconds. The magnetic thin-film memory is a fast reliable form of memory. Storage media consist of spots of a Permalloy* ferromagnetic material, deposited upon a substrate such as a thin glass plate. The geometry of these spots permits the magnetic state of a spot to be switched in billionths of a second with only a small amount of power applied. Since these spots have only two stable states of magnetization, they can readily store binary information.

Of these 128 locations, 104 are special purpose and provide storage for input buffer control words, output buffer control words, output command buffer control words, the real-time clock, 7 index registers, and Continuous Data Mode Reload. The other 24 memory locations are used for data storage. Instructions can be run from the control memory, input/output transfers can take place to or from this memory, and any operand reference can be accomplished.

The permanent storage is a nondestructive readout (NDRO) type of memory used in the computer for an initial load routine and automatic program recovery (i.e., in bootstrap programs). This storage area is capable of reading 64 words with a read cycle time of 300 nanoseconds per word. Either one of the two 32-word bootstrap programs in the NDRO storage may be selected by a switch. This bootstrap memory may be entered from any point in a program, and the exit from this memory area requires no special instruction.

The Z0 (30 bits) register is the memory buffer register for the control and NDRO memories. All information read from these memories must pass through this register. All information stored in the control memory locations must pass through this register. No storage is possible to NDRO locations.

Table I is a list of the memory address assignments.

*Registered trademark of Western Electric Company

TABLE I. UNIVAC 1230 MEMORY ADDRESS ASSIGNMENT

Core	Address	Function
Magnetic Core	00000	Fault Entrance
	00001 - 00017	Unassigned
	00020 - 00037	External Interrupt Entrance
	00040 - 00057	Input Monitor Interrupt Entrance
	00060 - 00077	Output Monitor Interrupt Entrance
Magnetic Thin Film	00100 - 00117	Input Buffer Control Registers
	00120 - 00137	Output Buffer Control Registers
	00140 - 00157	External Function Buffer Control Registers
	00160	Real-Time Clock
	00161 - 00167	B ₁ through B ₇ Index Registers
	00170 - 00177	Unassigned Film Locations
	00200 - 00217	ESI Input Buffer Terminate Storage or CDM Reload
	00220 - 00237	ESI Output Buffer Terminate Storage or CDM Reload
	00240 - 00257	ESI EF Buffer Terminate Storage
	00260 - 00277	Unassigned
Magnetic Core	00300 - 00477	Unassigned
	00500 - 00517	External Function Monitor Interrupt Entrance
Transformer Core	00520 - 00537	External Interrupt Code Storage
	00540 - 00577	NDRO Bootstrap Program I
Magnetic Core	00540 - 00577	NDRO Bootstrap Program II
	00600 - 00617	Intercomputer Time-Out Interrupt Entrance
	00620 - 00653	Memory Unit 4 Channel Interrupt Addresses
	00654 →	Unassigned

CONSOLE CONTROL

The maintenance and control panel located on the upper front of the computer, includes indicator lamps which display a detailed report of the internal status of the computer and controls to permit manual initiation of various operations. It is not necessary during normal operations, however, to monitor the maintenance panel or console.

Each register is represented on the maintenance panel by a row of display lamps each of which can be used to enter a "one" manually into the corresponding bit position, and a clear button which can be used to enter "zeros" manually into all-bit positions of the register. Many of the registers are involved only in the mechanics of executing instructions and are not directly accessible to the program.

INPUT/OUTPUT SECTION

GENERAL

All references to input or output in this discussion are made from the standpoint of the computer; that is, input is always input to the computer, and output is always output from the computer.

Communication with the 1230 Computer is carried on in a 30-bit, parallel mode over the input/output channels. Each computer is equipped with 16 channels (numbered 0 through 15). These channels are assembled on four chassis, each of which contains four identical input/output channels.

The four C registers (C_1 , C_2 , C_3 , and C_4 ; one for each chassis) hold information for peripheral equipment during output or external function transfers. Each is 30 bits in length and acts as a buffer register for four output channels.

Groups of four channels can be provided with a fast or slow interface. The slow interface provides communications transfers rates of up to a nominal 40K words per channel. The fast interface will provide transfer rates of up to 166K words per channel and 500K words on three or more channels.

In the CP-642A Computer, the transfer of input and output data words is asynchronous with the computer program, but the program maintains synchronous control over the issuance of External Functions. In the 1230 Computer, transmission of External Functions may be handled the same as data transmission. To utilize this method, the peripheral equipment must set a line indicating it is capable of accepting a command word from the buffer; therefore, the transmission of the word need not be synchronous with the computer program. Provision has been made, however, to achieve synchronization of program and input/output control to be compatible with existing peripheral equipment. Transmission of External Functions to equipment not containing logic for requesting functions (commands) is provided by the Force External Function instruction.

Externally Specified Index

This outstanding feature provides peripheral devices with a means of specifying core storage areas in the computer memory for any input or output transfers they may request. The Externally Specified Index (ESI) mode of operation is useful as a multiplexing device for a number of slow transfer peripheral units occupying one channel. The buffer control words governing

the transfers are located at the INDEX address. If input is desired, an Input Request is presented with the Index on the lower order eight bits of the input lines and the data on the remaining bits. If output is desired, an Output Request is presented with the Index address. An active channel is provided by the program for response to this feature.

Externally Specified Addressing

The ESA feature provides peripheral devices with a means of specifying an absolute core memory location for storage or retrieval of data. An active channel mode of operation is required for computer response to this function. The address is presented on the lower 17 bits of the input lines and the data transmission path on the remaining bits. If input is desired, the external device presents an Input Request with the address and data. If output is desired, an Output Request is presented with the address.

Continuous Data Mode

The Continuous Data Mode, requested when initiating a buffer on a channel, by executing instruction code 7764 or 7765, is a feature which provides an automatic continuation of the buffer process. Buffer control words are transferred to the control memory buffer control addresses from the control memory CDM addresses for that channel. These may be reloaded during the buffer process. The Monitor Interrupt can be incorporated with the CDM. Certain instruction codes terminate the CDM. The CDM is especially useful when a continuous, high rate, stream of data must be transferred in or out of the computer.

CONTROL COMMUNICATION

The 1230 Computer is designed to use a d-c level input/output system. Signals are d-c levels which may be changed upon interchange of control information. Signals may exist for microseconds or for days, depending upon the nature of the particular task.

DEFINITION OF CONTROL SIGNALS

The control signals used for input and output operation are defined in Table II, and the lines which carry them are given the same names. These control lines are carried in the same cables as the data lines and have the same voltage levels.

TABLE II. CONTROL SIGNALS USED IN INPUT/OUTPUT

Channel	Signal Name	Origin	Meaning
Input Channel	Interrupt Enable (IE)	Computer	"I have enabled my input section to honor an Interrupt on your channel "
	Input Data Request (IDR)	Peripheral Equipment	"I have a data word on my output lines ready for you to accept "
	Input Acknowledge (IA)	Computer	"I have sampled your data lines "
	External Interrupt (INT)	Peripheral Equipment	"I have an Interrupt code word on my output lines ready for you to accept "
Output Channel	Output Data Request (ODR)	Peripheral Equipment	"I am in a condition to accept a word of data from you. "
	Output Acknowledge (OA)	Computer	"I have put a data word for you on the data lines; sample them now. "
	External Function Request (EFR)	Peripheral Equipment	"I am in a condition to accept an External Function message on my data lines. "
	External Function (EF)	Computer	"I have put an External Function message for you on the data lines; sample them now. "

COMPUTER—TO—PERIPHERAL EQUIPMENT INTERFACE

Data and Control Signals

Each of the 16 input channels and 16 output channels communicates over an associated cable (32 possible cables), containing 30 information lines plus four control lines. Table III compares control line designations in the CP-642A/USQ-20 and the 1230 Computers. The functions of three of the control lines on each cable are the same as in CP-642A/USQ-20 operation; however, an additional control signal has been added to implement an improved technique for the handling of control words between computer and equipment.

TABLE III. CONTROL SIGNALS IN NORMAL PERIPHERAL EQUIPMENT CHANNELS

Signal Origin	Functional Name (1230)	Equivalent Signal in CP-642A/USQ-20
P E	External Function Request	(None)
C	External Function	External Function
P E	Output Data Request	Output Data Request
C	Output Acknowledge	Output Acknowledge
C	Interrupt Enable	(None)
P E	Interrupt	Interrupt
P E	Input Data Request	Input Data Request
C	Input Acknowledge	Input Acknowledge

P E = Peripheral Equipment C = Computer

Sequence of Events

Examples will clarify the use of the control lines. Figure 3 shows the computer communicating with a peripheral equipment over both input and output cables. Request and Interrupt signals always originate at the peripheral equipment. Acknowledge and External Function signals always originate at the computer.

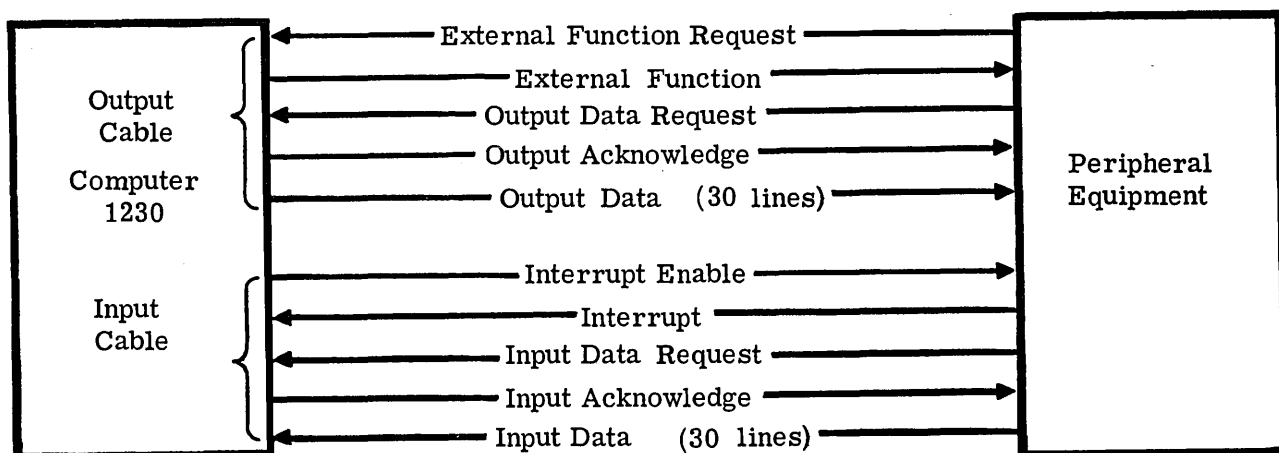


Figure 3. Computer-to-Peripheral Equipment Interface

The sequence of events for each of the four cases of communication between the computer and peripheral equipment is given below.

A normal output sequence for data transfer from computer to peripheral equipment (buffer mode) is as follows:

- a. Program control initiates output buffer for given channel
- b. Peripheral equipment sets the Output Request line when it is in a condition to accept data
- c. Computer (at its convenience) detects Output Request and clears the information lines if any data exist
- d. Computer places data on the output information lines
- e. Computer sets the Output Acknowledge line, indicating that data are ready for sampling
- f. Peripheral equipment detects the Output Acknowledge
- g. Peripheral equipment may drop Output Request any time after detecting Output Acknowledge
- h. Peripheral equipment samples the data on the output lines
- i. Computer drops Output Acknowledge

Steps b. through i. of this sequence are repeated for every data word until the number of words specified in the output buffer have been transferred.

The following sequence of events occurs when the computer is transmitting External Function messages to external equipment (buffer mode):

- a. Computer initiates External Function buffer for given channel
- b. Peripheral equipment sets the External Function Request line indicating that it is in a condition to accept External Function messages*

*Peripheral equipment not equipped with external function request logic should have the ability to accept Forced External Functions when in the Idle or Ready state by responding to sequence steps d., e., f., and h

- c. Computer (at its convenience) detects External Function Request and clears the data lines if any data exist
- d. Computer places External Function message on the data lines
- e. Computer sets the External Function line indicating that an External Function message is ready for sampling
- f. Peripheral equipment detects the External Function
- g. Peripheral equipment may drop the External Function Request any time after detecting the External Function
- h. Peripheral equipment samples the External Function message on the data lines
- i. Computer drops the External Function line

Steps b. through i. of this sequence are repeated for every External Function message until the number of words specified in the External Function buffer have been transferred.

Normal input sequence for data transfer to the computer from peripheral equipment is as follows:

- a. Program control initiates input buffer for given channel
- b. Peripheral equipment places data word on information lines
- c. Peripheral equipment sets the Input Request line to indicate that it has data ready for transmission
- d. Computer detects the Input Request at its convenience
- e. Computer samples the information lines
- f. Computer sets the Input Acknowledge line, indicating that it has sampled the data
- g. Peripheral equipment senses the Input Acknowledge line
- h. Peripheral equipment drops the Input Request line

Steps b. through h. of this sequence are repeated for every data word until the number of words specified in the input buffer have been transferred.

Sequence for transmitting an interrupt from peripheral equipment to the computer is as follows:

- a. Computer sets the Interrupt Enable* when it is ready to accept an External Interrupt**
- b. Peripheral equipment detects the Interrupt Enable
- c. Peripheral equipment status requires Computer to be interrupted
- d. Peripheral equipment places the Interrupt word on the data lines
- e. Peripheral equipment sets the Interrupt line to indicate that the Interrupt word is on the data lines
- f. Computer detects the interrupt signal and, at its convenience, accepts the Interrupt word and sets the Input Acknowledge line
- g. Computer drops the Interrupt Enable
- h. Peripheral equipment detects the Input Acknowledge and clears the Interrupt line and the data lines

The Input Acknowledge of an interrupt will be initiated at the same time that the Interrupt Enable is cleared. The simultaneous occurrence of these conditions should be used by peripheral equipment to differentiate between the acknowledge of an Interrupt and an Input Request.

COMPUTER—TO—COMPUTER INTERFACE

Data and Control Signals

Since all input/output channels of the 1230 Computer can be converted to intercomputer communication channels, it is possible for a computer to communicate with 16 other computers. The control signals and lines governing intercomputer communication are shown in Figure 4. Figure 4 illustrates the interface between two computers. Computer A is transmitting to Computer B. The selection of a given channel as an intercomputer channel affects only the

*Interrupt Enable is set either by a computer Master Clear or by execution of 60000·00000 or 60100·Y instructions (refer to Appendix)

**Peripheral equipment not equipped with interrupt enable logic may interrupt the program using sequence steps c., d., e., f., and h

logic concerned with the output and external function buffers. A peripheral channel, which is sending data or external function messages to a given peripheral equipment, holds the data in the output registers for a fixed minimum time period; after which any other Output or External Function Request on any channel can cause the data to be changed. However, intercomputer channel sending data or External Function messages to another computer must hold the information in the output registers until the receiving computer acknowledges receipt of those data. This acknowledge signal is received on what is known as the Output Request line when not on intercomputer mode. This line, in the intercomputer mode, is known as the Resume line.

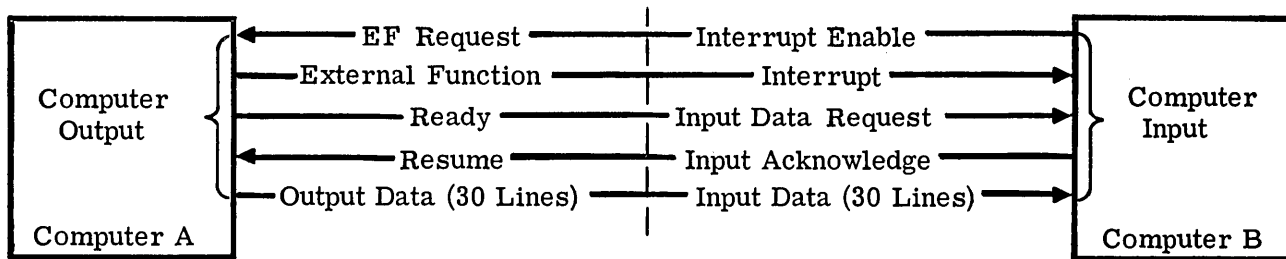


Figure 4. Computer-to-Computer Interface

The control signals in the input cable are the same for intercomputer communication as for communication with peripheral equipment. In the output cable, Ready and Resume signals are used to control the intercomputer transfer of data.

Sequence of Events

The sequence of events for each of the two cases of intercomputer communication appears below.

Intercomputer command word transfer from Computer A to Computer B is as follows:

- a. Computer B sets the Interrupt Enable when it is ready to accept a command word from Computer A
- b. Computer A recognizes the Interrupt Enable as an External Function Request and places the External Function code on the information lines
- c. Computer A sets the External Function to indicate that the External Function code is on the information lines
- d. Computer B recognizes the External Function as an Interrupt and accepts the command word at its convenience
- e. Computer B clears the Interrupt Enable line and sets the Input Acknowledge line
- f. Computer A recognizes the Input Acknowledge as a Resume and clears the External Function line

NOTE: In the event that Computer A sets the External Function line while the Interrupt Enable line is cleared (this is possible when an External Function with Force instruction is used), all communications on the associated group of output channels in A will be suspended until Computer B acknowledges receipt of the External Interrupt or until an intercomputer Time-Out Interrupt in A permits A to resolve the problem.

Intercomputer data transfer from Computer A to Computer B is as follows:

- a. Computer B initiates an input buffer and Computer A initiates an output buffer for the required channel. The output buffer of Computer A must be less than or equal to the input buffer of Computer B
- b. Computer A places data on the information lines
- c. Computer A sets the Ready line to indicate that the data are on the lines
- d. Computer B recognizes the Ready signal as an Input Data Request signal and, at its convenience, accepts the data word
- e. Computer B sets the Input Acknowledge
- f. Computer A recognizes the Input Acknowledge as a Resume signal and clears the Ready line and the data lines

Steps b. through f. of this sequence are repeated for every data word until the number of words specified in the output buffer have been transferred.

REPertoire OF INSTRUCTIONS

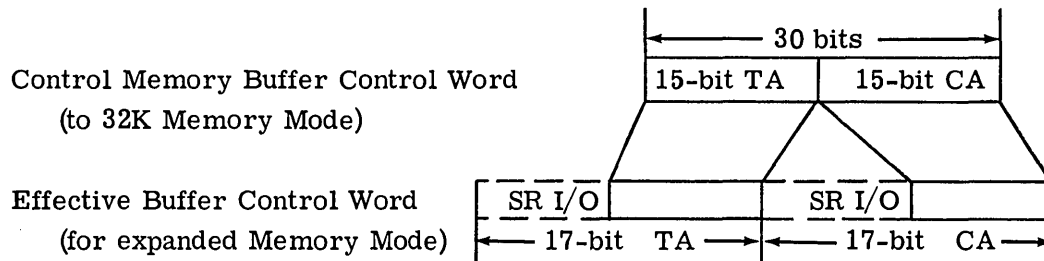
INTRODUCTION

The UNIVAC 1230 Military Computer is specified as a self-modifying, single-address computer. Although this means that one reference or address is provided for the execution of an instruction, this reference can be modified during a program sequence. References are modified by using the index registers, which contain any previously stored constants. In order to modify the address, the contents of a selected index register are added to the extended or natural operand designator, y. A programmed address is coded using octal notation with each octal digit denoting three binary digits. The instructions are read sequentially from memory storage except after satisfied Jump or Skip instructions. Every instruction executed by the computer is transmitted from memory to the Z register and then to the U register and the U2 register. The components of the instruction are translated to direct the control section in executing the operation specified. The U2 register is used for continued translation during execution of the operation sequence when the memory bank overlap feature is utilized. This allows the U-register to be cleared for reception of the next instruction with concurrent completion of the previous operation.

BUFFER CONTROL WORD FORMAT

The computer operating in the expanded memory mode can select any 16K module of memory for its Input and Output operation. There are three I/O Special Registers for each channel -- one each for Input, Output, and External Commands.

Each I/O control word in control memory contains the initial (or current) address (CA) and the terminal address (TA) of a buffer located anywhere in 32K of memory. Each I/O Special Register can contain the two bits necessary to expand the 32K control address to 131K. During Input or Output in the 1230 expanded mode, the two bits of the applicable SR I/O extend the 15-bit terminal and current addresses to 17 bits for memory circuit translation as follows:



The following symbols and definitions will be used in the discussion of Table IV.

- y_{SR} The 13-bit y prefaced by the core bank designator bits of the SR designated by $s=0, 1$ or 2
- y_P The 13-bit y prefaced by the core bank designator bits of P designated by $s=3$
- Y The operand used in an operation regardless of source
- \underline{Y} The quantity formed by $y + (B_b)$, $y_P + (B_b)$, $y_{SR} + (B_b)$ or a constant contained in an addressable register
- P The Program Address register
- SR Special Register, 4-bit core memory bank designator
- () Contents of the address or register
- ()_i Initial contents of the address or register
- ()_f Final contents of the address or register
- ()_n Designates any single n th bit of the contents of a register
- :
- The colon in a logical expression indicates COMPARISON
- $L() ()$ The bit-by-bit or logical product (logical AND) defined by the table:
- or
- () \odot ()
- | | |
|---|---|
| 0 | 1 |
| 0 | 0 |
| 1 | 0 |
- () \vee () Logical sum, or inclusive OR defined by the table:
- | | |
|---|---|
| 0 | 1 |
| 0 | 0 |
| 1 | 1 |
- () \oplus () Half add, half subtract, or exclusive OR defined by the table:
- | | |
|---|---|
| 0 | 1 |
| 0 | 1 |
| 1 | 0 |
- ()['] or $\bar{()}$ The one's complement of the contents of the address or register
- () () Algebraic product of the contents of two locations
- \rightarrow Transfer the quantity stated at the left of the symbol to the address or register stated at the right of the symbol

TABLE IV. REPERTOIRE OF INSTRUCTIONS

Code (Octal)	Instruction	Instruction Execution Time (Overlapped) in μ Sec	Code (Octal)	Instruction	Instruction Execution Time (Overlapped) in μ Sec
00	(Fault Interrupt)		50	Selective Set	2
01	Right Shift Q	2	51	Selective Complement	2
02	Right Shift A	2	52	Selective Clear	2
03	Right Shift AQ	2	53	Selective Substitute	2
04	Compare	2	54	Replace Selective Set	4
05	Left Shift Q	2	55	Replace Selective Complement	4
06	Left Shift A	2	56	Replace Selective Clear	4
07	Left Shift AQ	2	57	Replace Selective Substitute	4
10	Enter Q	2	60*	Jump (Arithmetic)	4
11	Enter A	2	61*	Jump (Manual)	4
12*	Enter B _j	4	62*	Jump on C _j Active Input Buffer	4
13	External Function on C _j [^]	4	63*	Jump on C _j Active Output Buffer	4
14	Store Q	2	64*	Return Jump (Arithmetic)	6
15	Store A	2	65*	Return Jump (Manual)	6
16	Store B _j	2	66	Terminate C _j Input Buffer	2
17	Store C _j or Test EFB	4	67	Enable, or Disable Interrupts	
				Terminate C _j Output Buffer	2
				or All Buffers	
20	Add A	2	70*	Repeat	4
21	Subtract A	2	71*	B Skip on B _j	4
22*	Multiply	8	72*	B Jump on B _j	4
23*	Divide or Square Root	14-8	73*	Input Buffer on C _j [^]	4
24	Replace A+Y	4		(without monitor mode)	
25	Replace A-Y	4	74*	Output Buffer on C _j [^]	4
26	Add Q	2		(without monitor mode)	
27	Subtract Q	2	75*	Input Buffer on C _j [^]	4
				(with monitor mode)	
30	Enter Y+Q	2	76*	Output Buffer on C _j [^]	4
31	Enter Y-Q	2		(with monitor mode)	
32	Store A+Q	4			
33	Store A-Q	4	7700	Fault Interrupt	
34	Replace Y+Q	4	7707	Normalize	4
35	Replace Y-Q	4	7760	Enter Special Register	2
36	Replace Y+1	4	7761	Enter Input SR I/O	2
37	Replace Y-1	4	7762	Enter Output SR I/O	2
			7763	Enter E F SR I/O	2
40	Enter Logical Product	2	7764*	Enable Input CDM	2
41	Add Logical Product	2	7765*	Enable Output CDM	2
42	Subtract Logical Product	2	7766*	Disable Input CDM	2
43	Compare Masked	2	7767*	Disable Output CDM	2
44	Replace Logical Product	4	7770	Store Special Register	2
45	Replace A+Logical Product	4	7771	Store Input SR I/O	2
46	Replace A-Logical Product	4	7772	Store Output SR I/O	2
47	Store Logical Product	2	7773	Store EF SR I/O	2
			7774*	Disable 17-Bit Address Mode	2
			7775*	Enable 17-Bit Address Mode	2
			7777	Fault Interrupt	

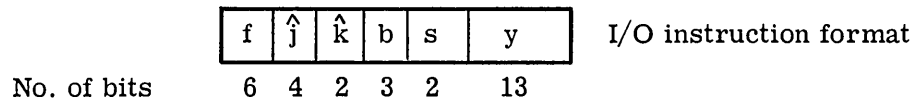
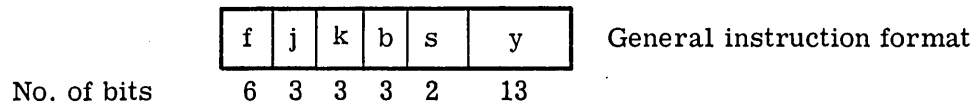
For nonoverlapped, add two microseconds to the execution time
 *Execution time is constant - overlapped or not

INSTRUCTION WORD FORMAT

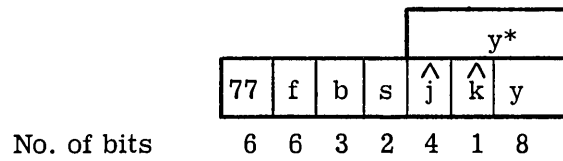
Two types of instruction words make up the repertoire for the 1230 Computer. Type I contains the operation code in the most significant six bits of the word. Type II instruction contain octal 77 in the most significant six bits and the operation code in the next six bits.

The formats for Type I and II instruction words in the repertoire of the 1230 Computer while utilizing the expanded memory feature are as follows:

Type I

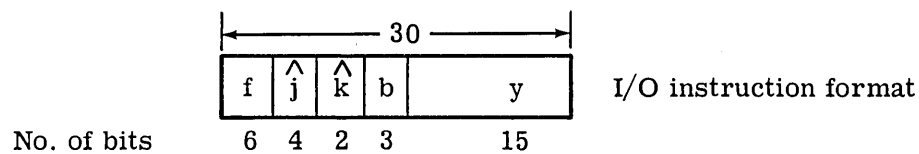
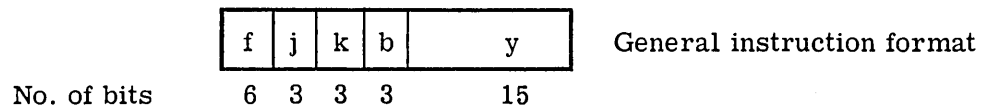


Type II

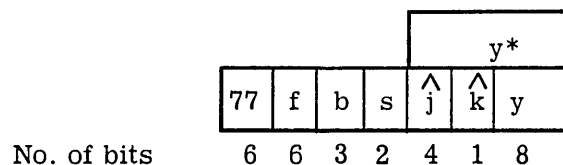


In the repertoire of the 1230 Computer, the formats for Type I and II instruction words (while utilizing the addressing structure limited to the internal 32K memory) are as follows:

Type I



Type II



* y is a 13-bit quantity. k does not exist but is forced as an unconditional 3

The elements of the function word format are interpreted as follows:

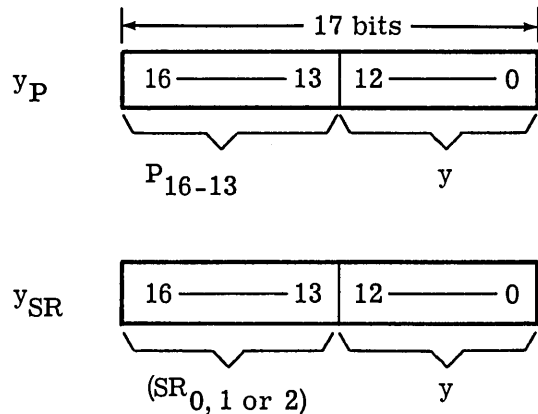
- f is a 6-bit Function Code Designator
- j is a 3-bit Branch Condition Designator
- \hat{j} is a 4-bit Input/Output Channel Designator or Special Register Designator
- k or \hat{k} is a 1, 2, or 3-bit Operand Interpretation Designator
- b is a 3-bit Index Register Designator
- s is a 2-bit Address Extension Designator
- y is a 15-, 13-, or 8-bit Operand or Address Designator

Function Code Designation

The f designator appears in bit positions 29 through 24 of the U-register or an instruction word in Type I instructions and in bit positions 23 through 18 in Type II instructions. All values of f other than 00 are defined in the instruction repertoire. Code 00 in either Type I or Type II instructions is a program fault condition: if executed, it causes a fault interrupt and a jump to address 00000, the fault entrance register, or address 00540 of memory, depending on the Automatic Recovery switch setting.

Operand Designator

The y designator appears in bit positions 12 through 0 of the U register or instruction of Type I instructions requiring a memory reference for the operation when the computer is operated in the expanded memory mode. The y designator appears in bit positions 7 through 0 or 12 through 0 in Type II instructions in either mode. y_{SR} or y_P is the 13-bit y prefaced by the memory bank designator bits of the SR or of (P) as designated by s. This extended operand designator is described below:



The operand or address of the operand, designated Y is obtained from these 17 bits plus the contents of an index register if specified.

When the computer is operated within the 32K internal memory as a CP-642B Computer, the y designator appears in bit positions 14 through 0. In this mode, the designator supplies the required bit configuration for the addressing structure of the main memory. No references are made to the Special Registers or the upper 2 bits of (P). \underline{Y} is then defined as the 15-bit y plus the contents of an index register if specified.

Address Extension Designator

The s designator (2 bits) appears in bit positions 13 and 14 and is interpreted in instructions requesting memory references. It does not exist in Type I instructions requiring no memory reference for the operation since the y designator occupies 15-bit positions. The s designator is used to specify which of the three Special Registers, or the upper 4 bits of P are used to extend the operand address. The designator is interpreted as follows:

- $s = 0$: Extend y by (SR0)
- $s = 1$: Extend y by (SR1)
- $s = 2$: Extend y by (SR2)
- $s = 3$: Extend y by (P_{16-13})

- a) For read instructions, 01-12, 20-23, 26-31, 40-43, 50-53, 70 and 71 when $k \neq 0, 4, 7$
- b) For store instructions, 14-16, 32, 33 and 47 when $k \neq 0, 4$
- c) For replace instructions 24, 25, 34-37, 44-46 and 54-57 when $k \neq 0, 4, 7$
- d) For read instructions 60-63 and 72 for all k values
- e) For I/O instructions 73-76, when $\hat{k} \neq 0$
- f) For I/O instructions 13 and 17
- g) For Type II instructions when $\hat{k} \neq 0$
- h) For return jump instructions 64 and 65 for all k values

NOTE: It is possible for the return address to be larger than 15 bits. Therefore, it is required that (P) + 1 be stored at the whole word at memory address Y as designated by $k = 3$ and then jump to address $y + 1$. Therefore, all return or exits from a subroutine will be indirect with a k value of 3 in the exit command.

For the restricted k values mentioned above, y is a 15-bit quantity and s does not exist. In I/O instructions 66 and 67, the low order 15 bits are not translated and may be used as extra storage of data.

Index Designator

The b designator appears in bit positions 17, 16, and 15 of the U register or an instruction word. It specifies which index register (B register), if any, will be used to modify the operand designator, y , y_P or y_{SR} to form the operand or operand address, \underline{Y} . The computer uses an

additive type adder to produce the quantity $\underline{Y} = y \oplus y_P \oplus y_{SR} + (Bb)_L$; hence, a quantity consisting of all zeros cannot result unless both the form of y and $(Bb)_L$ are zeros. Use of a B register for modification causes the higher order bits of that B register to become zeros. The contents of the lower order 17 or 15 bits of the index register serve as the modifier.

The effect of the various values of the b designator is as follows:

- b = 0: Do not modify y , y_P or y_{SR}
- b = 1: Add (B1) to y , y_P or y_{SR}
- b = 2: Add (B2) to y , y_P or y_{SR}
- b = 3: Add (B3) to y , y_P or y_{SR}
- b = 4: Add (B4) to y , y_P or y_{SR}
- b = 5: Add (B5) to y , y_P or y_{SR}
- b = 6: Add (B6) to y , y_P or y_{SR}
- b = 7: Add (B7) to y , y_P or y_{SR}

The modifying index register quantities occupy the lower order 17 bits of control memory addresses 00161 through 00167 during operation in the expanded memory mode. They occupy the lower order 15 bits of those control memory addresses during operation as a CP-642B computer (limited to 32K memory).

Branch Condition Designator

The j designator appears in bit positions 23, 22, and 21 of the U register or an instruction; it is used in a majority of the instructions. The three primary uses of j are for jump and skip determination, for B register specification, and for repeat status interpretation. Interpretations of the j designator are listed either below or under the descriptions of the individual instructions requiring special interpretations.

For those instructions in which the j designator has no special interpretation, it specifies the condition under which the next sequential instruction in the program will be skipped. This permits branching from a sequence without executing a Jump instruction, as would normally occur if a skip condition were not satisfied.

A skip of the next sequential instruction is determined by the following rules in all Type I instructions except 04, 12, 13, 16, 17, 23, 26, 27, 60-67, and 70-76:

- j = 0: Do not skip the next instruction
- j = 1: Skip the next instruction
- j = 2: Skip the next instruction if (Q) is positive
- j = 3: Skip the next instruction if (Q) is negative
- j = 4: Skip the next instruction if (A) is zero (positive zero)
- j = 5: Skip the next instruction if (A) is nonzero
- j = 6: Skip the next instruction if (A) is positive
- j = 7: Skip the next instruction if (A) is negative

When the branch condition involves the sign of the quantity in A or Q, the evaluation examines the sign bit of these quantities; hence, positive zero (all zeros) is considered a positive quantity, and negative zero (all ones) is considered a negative quantity.

For input/output instructions, the \hat{j} designator appears in bit positions 23, 22, 21 and 20 of the U-register or an input/output instruction and specifies the channel for the instruction. Bit 23 assumes a value of eight, bit 22 a value of four, bit 21 a value of two, and bit 20 a value of one; thus, the \hat{j} designator provides accessibility to the 16 (decimal) input/output channels (number 0 - 15). Instructions 13, 17, 62, 63, 66, 67, 73, 74, 75, and 76 use the \hat{j} designator configuration. In Type II instructions, \hat{j} appears in bit positions 12, 11, 10, and 9 and designates an I/O channel or a Special Register.

Operand Interpretation Designation

The k designator (3 bits) appears in bit positions 20, 19, and 18 of the U-register or an instruction. The \hat{k} designator appears in bit positions 19 and 18 in Type I instructions or in bit position 8 of Type II instructions. Instructions 13, 17, 62, 63, 66, 67 and 73 through 76 and Type II use the \hat{k} designator configuration since they perform input/output or Special Register activities and require a \hat{j} designator for channel or Special Register specification. The \hat{k} designator controls operand and instruction interpretation for the input/output instructions as noted in the description of the applicable instruction.

In Type II instructions, the \hat{k} designator appears in bit position 8 and is used to interpret the operand as follows:

Store Class

$\hat{k} = 0$: Store in the least significant 8 bits of the Q-register
 $\hat{k} = 1$: Store in the lower 8 bits of the memory address leaving the remaining upper bits undisturbed

Read Class

$\hat{k} = 0$: (U_L) 8 bits \rightarrow Register
 $\hat{k} = 1$: Y_L 8 bits \rightarrow Register

The k designator controls operand interpretation. Those instructions that read an operand but do not replace it after the operation is performed are designated read instructions. Those instructions that do not read an operand but store it are designated store instructions. Instructions which both read and store operands are classified as replace instructions.

The various values of k affect the operand Y, except where noted otherwise under individual instruction description, as follows:

Read instruction (01-12, 20-23, 26, 27 30, 31, 40-43, 50-53, 60-65, 70-72):

k = 0: $Y_u = 0$'s; $Y_L = \underline{Y}$
k = 1: $Y_u = 0$'s; $Y_L = (\underline{Y})L$; Operand is lower half of memory location
k = 2: $Y_u = 0$'s; $Y_L = (\underline{Y})u$; Operand is upper half of memory location

- k = 3: Y = (Y)
- k = 4: Yu = same bits as Y14; YL = Y
- k = 5: Yu = same bits as Y14; YL = (Y)L; Operand is lower half of memory location with sign extension
- k = 6: Yu = same bits as Y14; YL = (Y)u; Operand is upper half of memory location with sign extension
- k = 7: Y = (A)

For instructions 22, 52, and 53, k = 7 is not used.

Store instructions (14-16, 32, 33, 47):

- k = 0: Store (A or Bj) in Q.*
- k = 1: Store (AL, QL, or Bj) in YL, leaving (Y)u undisturbed
- k = 2: Store (AL, QL, or Bj) in Yu, leaving (Y)L undisturbed
- k = 3: Store (A, Q, Cj, or Bj) in Y
- k = 4: Store (Q or Bj) in the A register **
- k = 5: Store complement of (AL, QL, or Bj) in YL, leaving (Y)u undisturbed
- k = 6: Store the complement of (AL, QL, or Bj) in Yu, leaving (Y)L undisturbed
- k = 7: Store the complement of (A, Q, or Bj) in Y (storing the complement of Bj is the same complement as for a 30-bit register)

Replace instructions (24, 25, 34-37, 44-46, 54-57):

- k = 0: Not used.
- k = 1: Read portion - Yu = 0's; YL = (Y)L
Store portion - Stores AL in YL leaving (Y)u undisturbed
- k = 2: Read portion - Yu = 0's; YL = (Y)u
Store portion - Stores AL in YU leaving (Y)u undisturbed
- k = 3: Read portion - Y = (Y)
Store portion - Stores A in Y
- k = 4: Not used.
- k = 5: Read portion - Yu = same bits as (Y)14; YL = (Y)L
Store portion - Stores AL in YL leaving (Y)u undisturbed
- k = 6: Read portion - Yu = same bits as Y29; YL = (Y)u
Store portion - Stores AL in YU leaving (Y)L undisturbed
- k = 7: Not used

Replace instructions require special interpretation when following a repeat instruction. (Refer to subsequent paragraph entitled 70 REPEAT .)

* A 1400000000 instruction complements (Q)

** A 1504000000 instruction complements (A)

List of Instructions

This section lists the repertoire of instructions.

01 RIGHT SHIFT Q

This instruction shifts (Q) to the right Y bit positions*; the higher order bits are replaced with the original sign bit, and the lower order bits are discarded as the word is shifted. Only the lower order six bits of Y are recognized for this instruction. The higher order 24 bits are ignored.

An 8-bit register example of right shift Q: Y = 2.

Contents of Q		Contents of Q	
(Q) _i (positive) =	01010011	(Q) _i (negative) =	10100011
First shift	00101001	First shift	11010001
Second shift	00010100	Second shift	11101000

02 RIGHT SHIFT A

This instruction shifts (A) to the right Y bit positions*. The higher order bits are replaced with the original sign bit, and the lower order bits are discarded as the word is shifted. Only the lower order six bits of Y are recognized for this instruction. The higher order 24 bits are ignored. The over-all operation is similar to the example given for Right Shift Q.

03 RIGHT SHIFT AQ

This instruction shifts (A) and (Q) as one 60-bit register. The shift is to the right Y bit positions*, with the lower bits of A shifting into the higher bit positions of Q. The higher order bits of A are replaced with the original sign bit, and the lower order bits are discarded as the word is shifted. Only the lower order six bits of Y are recognized for this instruction. The higher order 24 bits are ignored.

An 8-bit register example of right shift AQ: Y = 2:

Contents of AQ		Contents of AQ	
(AQ) _i (positive) =	01010011	(AQ) _i (negative) =	10001010
First shift	00101001	First shift	11000101
Second shift	00010100	Second shift	11100010

*The maximum shift count permitted is decimal 59 places

04 COMPARE

This instruction compares the signed value of Y with the signed value of (A) and (Q) or the value of either. It does not alter either (A) or (Q). The branch condition designator, j, is interpreted in a special way for this instruction as listed below:

- j = 0: Do not skip the next instruction
- j = 1: Skip the next instruction
- j = 2: Skip the next instruction if Y is less than or equal to (Q)
- j = 3: Skip the next instruction if Y is greater than (Q)
- j = 4: Skip the next instruction if Y is less than or equal to (Q), and Y is greater than (A)
- j = 5: Skip the next instruction if Y is greater than (Q), or if Y is less than or equal to (A)
- j = 6: Skip the next instruction if Y is less than or equal to (A)
- j = 7: Skip the next instruction if Y is greater than (A)

05 LEFT SHIFT Q

This instruction shifts (Q) circularly to the left, Y bit positions*. The lower order bits are replaced with the higher order bits as the word is shifted. Only the lower order six bits of Y are recognized for this instruction. The higher order 24 bits are ignored.

An 8-bit register example of left circular shift in Q: (Y) = 2

Contents of Q		Contents of Q	
(Q) _i (positive) =	00110001	(Q) _i (negative) =	11001100
First shift	01100010	First shift	10011001
Second shift	11000100	Second shift	00110011

06 LEFT SHIFT A

This instruction shifts (A) circularly to the left Y bit positions*. The lower order bits are replaced with the higher order bits as the word is shifted. Only the lower order six bits of Y are recognized for this instruction. The higher order 24 bits are ignored. The over-all operation is similar to the example given for left shift Q.

07 LEFT SHIFT AQ

This instruction shifts (A) and (Q) as one 60-bit register. The shift is circular to the left Y bit positions*. The lower order bits of A are replaced with the higher order bits of Q and the lower order bits of Q are replaced with the higher order bits of A. Only the lower order six bits of Y are recognized by this instruction. The higher order 24 bits are ignored.

*The maximum shift count permitted is decimal 59 places

An 8-bit register example of left shift AQ: $Y = 2$:

Contents of AQ		Contents of AQ	
$(AQ)_i$ (positive) =	01010011	$(AQ)_i$ (negative) =	10001011
First shift	10100110	First shift	00010111
Second shift	01001101	Second shift	00101110

10 ENTER Q

Clear the Q register; then transmit Y to Q.

11 ENTER A

Clear the A register; then transmit Y to A.

12 ENTER B_j

Clear the B register specified by j. Then transmit Y to this B register. The branch condition designator, j, is used to specify the selected B register for this instruction and is not available for its normal function.

13 EXTERNAL FUNCTION ON C_j^{\wedge}

Transfer the contents of storage address \underline{Y} lower to the external function buffer control register $140 + j^{\wedge}$ upper and lower and initiate a 1-word external function buffer on channel j^{\wedge} . The address of the function word is maintained in the lower order 15 bits of the control register. The k^{\wedge} values modify the instruction as follows:

- $k^{\wedge} = 0$: Establish the 1-word external function buffer with monitor and proceed to the next instruction. The function transfer is executed when the external equipment presents an External Function Request. A monitor interrupt follows the completion of the transfer
- $k^{\wedge} = 1$: Establish the 1-word external function buffer with monitor and with force. Hold the program until the function is transferred to the output channel. A monitor interrupt follows the completion of the transfer
- $k^{\wedge} = 2$: Establish the 1-word external function buffer and proceed to the next instruction. The function transfer is executed when the external equipment presents an External Function Request
- $k^{\wedge} = 3$: Establish the 1-word external function buffer with force. Hold the program until the function is transferred to the output channel

An external function buffer with force will transfer the function word from memory to the output register whether the external device requests it or not. If external equipment cannot accept external functions from consecutive program executions, restrictions must be made in the programming of external functions to this equipment.

14 STORE Q

Store (Q) at storage address Y as directed by the operand interpretation designator, k. If k = 0, complement (Q). If k = 4, store in A.

15 STORE A

Store (A) at storage address Y as directed by the operand interpretation designator, k. If k = 4, complement (A). If k = 0, store in Q.

16 STORE B_j

Store a 30-bit quantity, whose lower order 15 bits correspond to the contents of the B register specified by j and whose higher order 15 bits are zero, at storage address Y as directed by the operand interpretation designator, k. The branch condition designator, j, is used to specify the selected B register for this instruction and is not available for its normal function.

17 STORE C[^]_j OR TEST EFB

For $\hat{k} = 0$ and 1, when the jump condition is satisfied, the program register, P, is cleared, and a new next instruction address, Y or (Y)L respectively, is entered into P.

- $\hat{k} = 0$: And the external function buffer on channel \hat{j} is active, jump to address Y; if not active, execute the next sequential instruction
- $\hat{k} = 1$: And the external function buffer on channel \hat{j} is active, jump to address (Y)L; if not active, execute the next sequential instruction
- $\hat{k} = 2$: Store the contents of input channel \hat{j} at storage address Y and send an Input Acknowledge on that channel. Hold the program until the word is transferred to memory
- $\hat{k} = 3$: Store the contents of storage address $00520 + \hat{j}$ at storage address Y

20 ADD A

Add Y to the previous contents of the A register.

21 SUBTRACT A

Subtract Y from the previous contents of the A register.

22 MULTIPLY

Multiply (Q) times Y, forming the double-length product in AQ. If the factors are considered as integers, the product is an integer in AQ. k = 7 may not be used.

The branch condition designator, j , is interpreted prior to final sign correction, permitting sensing of a product with $(A)f = 0$. When $(A)f \neq + 0$, a double-length product has been formed with significant bit(s) in the A-register; however, if a skip does not occur for $j = 4$, an Add Q instruction can be performed (add zero to Q) with $j = 6$ or 7 to determine if Q29 contains a significant bit (one) of the product.

23 DIVIDE

If $k \neq 7$, divide (AQ) by Y , leaving the quotient in the Q-register and the remainder in the A register. The remainder bears the same sign as the dividend. No console indication is given if a divide overflow exists. By coding each divide instruction with $j = 2$ or 3 , a program test for the divide overflow is automatic. With a selection of $j = 3$, a skip of the next instruction occurs if divide overflow exists. The skip should be made to a routine which provides proper correction. With a selection of $j = 2$, a correct answer is indicated when the skip occurs, and the instruction is followed by a jump instruction to the remedial routine.

If $k = 7$, take the square root of (Q) , leaving the root in Q and the residue in A. $(A)f = (Q)i - (Q)f^2$. A selection of $j = 2$ in the square root instruction requests skip if a residue appears in A, and $j = 3$ requests skip if no residue appears in A. $j = 4, 5, 6$ or 7 is not used when $k = 7$.

24 REPLACE A + Y

Add Y to the previous contents of the A register. Store (A) at storage address Y.

25 REPLACE A - Y

Subtract Y from the previous contents of the A register. Then store (A) at storage address Y.

26 ADD Q

Interchange (A) and (Q) . Then add Y to (A) . Interchange (A) and (Q) . The contents of the A register are undisturbed by this instruction. The branch condition designator, j , has special meaning in this instruction as listed in instruction 27.

27 SUBTRACT Q

Interchange (A) and (Q) . Then subtract Y from (A) . Interchange (A) and (Q) . The contents of the A register are undisturbed by this instruction. The branch condition designator, j , has special meaning in this instruction as follows:

In instructions 26 and 27, the branch condition designator, j, has the following meaning:

j = 0: Do not skip the next instruction
j = 1: Skip the next instruction
j = 2: Skip the next instruction if (A) is positive
j = 3: Skip the next instruction if (A) is negative
j = 4: Skip the next instruction if (Q) is zero
j = 5: Skip the next instruction if (Q) is nonzero
j = 6: Skip the next instruction if (Q) is positive
j = 7: Skip the next instruction if (Q) is negative

30 ENTER Y + Q

Clear the A-register. Then transmit (Q) to A. Then add Y to (A).

31 ENTER Y - Q

Clear the A-register. Transmit (Q) to A and then subtract Y from (A). Finally, complement (A).

32 STORE A + Q

Add (Q) to the previous contents of the A register. Then store the sum in A and at storage address Y as directed by the operand interpretation designator, k.

33 STORE A - Q

Subtract (Q) from the previous contents of the A register. Then store the difference in A and at storage address Y as directed by the operand interpretation designator, k.

34 REPLACE Y + Q

Clear the A-register. Transmit (Q) to A; then add Y to (A). Store the sum in A and at storage address Y.

35 REPLACE Y - Q

Clear the A register. Transmit (Q) to A. Subtract Y from (A). Then complement the difference and store in A and at storage address Y.

36 REPLACE Y + 1

Clear the A register. Set (A) = 1. Add Y to (A) and store the sum in A and at storage address Y.

37 REPLACE Y - 1

Clear the A register. Set $(A) = 1$. Subtract Y from (A). Complement the difference and store in A and at storage address Y.

40 ENTER LOGICAL PRODUCT

Enter the bit-by-bit product of Y and (A) in the A register.

The j designator is interpreted in a special way for this instruction for the values $j = 2$ or 3 . If $j = 2$, skip if the parity of the (A)f is even. If $j = 3$, skip if the parity of (A)f is odd.

NOTE:

Even parity means an even number of ones in the A register.
Odd parity means an odd number of ones in the A register.

41 ADD LOGICAL PRODUCT

Add to (A) the bit-by-bit product of Y and (Q).

42 SUBTRACT LOGICAL PRODUCT

Subtract from (A) the bit-by-bit product of Y and (Q).

43 COMPARE MASKED

Subtract from (A) the bit-by-bit product of Y and (Q), and perform the evaluation for a skip of the next sequential instruction as directed by the branch condition designator, j. Then add to (A) the bit-by-bit product of Y and (Q).

This instruction results in no net change in the contents of any operational register. It provides a comparison of a portion of Y with (A).

44 REPLACE LOGICAL PRODUCT

Enter in the A register the bit-by-bit product of Y and (Q). Then store (A) at storage address Y.

The j designator is interpreted in a special way for this instruction for the values $j = 2$ or 3 . If $j = 2$, skip if the parity of (A)f is even. If $j = 3$, skip if the parity of (A)f is odd.

45 REPLACE A + LOGICAL PRODUCT

Add to (A) the bit-by-bit product of Y and (Q). Then store the sum in A and at storage address Y.

46 REPLACE A - LOGICAL PRODUCT

Subtract from (A) the bit-by-bit product of Y and (Q). Store the difference in A and at storage address Y.

47 STORE LOGICAL PRODUCT

Store in address Y the bit-by-bit product of (A) and (Q) as directed by the operand interpretation designator, k.

50 SELECTIVE SET

Set the individual bits of the A register corresponding to ones in Y, leaving the remaining bits of the A register unaltered.

51 SELECTIVE COMPLEMENT

Complement the individual bits of the A register corresponding to ones in Y, leaving the remaining bits of the A register unaltered.

52 SELECTIVE CLEAR

Clear the individual bits of the A register corresponding to ones in Y, leaving the remaining bits of the A register unaltered. In this instruction, k = 7 should not be used.

53 SELECTIVE SUBSTITUTE

Substitute individual bits of the A register with bits of Y corresponding to ones in Q, leaving the remaining bits of the A register unaltered.

54 REPLACE SELECTIVE SET

Set the individual bits of the A register corresponding to ones in Y, leaving the remaining bits of the A register unaltered. Store (A) at storage address Y.

55 REPLACE SELECTIVE COMPLEMENT

Complement the individual bits of A register corresponding to ones in Y, leaving the remaining bits of A register unaltered. Store (A) at storage address Y.

56 REPLACE SELECTIVE CLEAR

Clear individual bits of the A register corresponding to ones in Y, leaving the remaining bits of the A register unaltered. Store (A) at storage address Y.

57 REPLACE SELECTIVE SUBSTITUTE

Clear individual bits of the A-register corresponding to ones in Q, leaving the remaining bits of the A register unaltered. Form the bit-by-bit product of Y and (Q), and set ones of this product in corresponding bits of the A register, leaving the remaining bits unaltered. Store (A) at storage address Y.

60 JUMP (ARITHMETIC)

This instruction clears the program address register, P, and enters a new program address in P for certain conditions of either the A or Q register content. The branch condition designator, j, is interpreted in a special way for this instruction and thus determines the conditions under which a jump in program occurs. If the jump condition is not satisfied, the next sequential instruction in the current sequence is executed in a normal manner. If the jump condition is satisfied, as listed below, Y becomes the address of the next instruction and the beginning of a new program sequence.

- j = 0: No jump. Set interrupt enable and remove interrupt lockout, thus clearing bootstrap and interrupt modes. Continue with current program sequence
- j = 1: Execute jump. Set interrupt enable and remove interrupt lockout, thus clearing bootstrap and interrupt modes
- j = 2: Execute jump if (Q) is positive
- j = 3: Execute jump if (Q) is negative
- j = 4: Execute jump if (A) is zero (positive zero)
- j = 5: Execute jump if (A) is nonzero
- j = 6: Execute jump in (A) is positive
- j = 7: Execute jump if (A) is negative

61 JUMP (MANUAL)

This instruction clears the program address register, P, and enters a new program address in P for certain conditions of manual JUMP and STOP switch selections. The branch condition designator, j, is interpreted in a special way for this instruction and determines the conditions under which a jump in program occurs. If the jump condition is not satisfied, the next sequential instruction of the current sequence is executed in a normal manner. If the jump condition is satisfied, as listed below, Y becomes the address of the next instruction and the beginning of a new program sequence.

The program may be stopped by certain STOP selections upon execution of this instruction. The branch condition designator, j, specifies which switch selections are effective.

j = 0:	Execute jump regardless of switch selection.
j = 1:	Execute jump if JUMP 1 is selected.
j = 2:	Execute jump if JUMP 2 is selected.
j = 3:	Execute jump if JUMP 3 is selected.
j = 4:	Execute jump. Stop computation.
j = 5:	Execute jump. Stop computation if STOP 5 is selected.
j = 6:	Execute jump. Stop computation if STOP 6 is selected.
j = 7:	Execute jump. Stop computation if STOP 7 is selected.

62 JUMP ON C_j^{\wedge} ACTIVE INPUT BUFFER

If the input buffer on channel j^{\wedge} is active, clear the program address register, P, and enter a new program address, Y, in P. Y becomes the address of the next instruction. If the input buffer is not active, the next sequential instruction in the current sequence is executed in the normal manner.

63 JUMP ON C_j^{\wedge} ACTIVE OUTPUT BUFFER

If the output buffer on channel j^{\wedge} is active, clear the program address register, P, and enter a new program address, Y, in P. Y becomes the address of the next instruction. If the output buffer is not active, the next sequential instruction in the current sequence is executed in the normal manner.

64 RETURN JUMP (ARITHMETIC)

Execute a return jump sequence for certain conditions of either the A or Q register. The branch condition designator, j, is interpreted in a special way for this instruction and determines the conditions under which the return jump sequence is executed. If the return jump condition is not satisfied, the next sequential instruction in the current sequence is executed in a normal manner. If the return jump condition is satisfied, as listed below, the following sequence is performed.

Store (P) + 1 in the lower half of memory address Y, as designated by k. Then jump to Y + 1.

j = 0:	No action. Continue with the current program sequence
j = 1:	Execute return jump
j = 2:	Execute return jump if (Q) is positive
j = 3:	Execute return jump if (Q) is negative
j = 4:	Execute return jump if (A) is zero (positive zero)
j = 5:	Execute return jump if (A) is nonzero
j = 6:	Execute return jump if (A) is positive
j = 7:	Execute return jump if (A) is negative

65 RETURN JUMP (MANUAL)

Execute a return jump sequence for certain conditions of manual switch selections. The branch condition designator, j , is interpreted in a special way for this instruction and determines the conditions under which the return jump sequence is executed. If the return jump condition is not satisfied, the next sequential instruction in the current sequence is executed in a normal manner. If the return jump condition is satisfied, as listed below, then the following sequence is performed.

Store $(P) + 1$ in the lower half of memory address Y as designated by k . Then jump to $Y + 1$.

$j = 0$:	Execute return jump regardless of switch selections
$j = 1$:	Execute return jump if JUMP 1 is selected
$j = 2$:	Execute return jump if JUMP 2 is selected
$j = 3$:	Execute return jump if JUMP 3 is selected
$j = 4$:	Execute return jump. Then stop computation
$j = 5$:	Execute return jump. Stop computation if STOP 5 is selected
$j = 6$:	Execute return jump. Stop computation if STOP 6 is selected
$j = 7$:	Execute return jump. Stop computation if STOP 7 is selected

66 TERMINATE C_j^{\wedge}

Terminate Input Buffer/Enable Interrupts/Disable Interrupts as designated by \hat{k} and b as follows:

$\hat{k} = 0$:	Terminate the input/buffer on channel j
$\hat{k} = 1$ and $b = 0$:	Enable all interrupts
$\hat{k} = 1$ and $b \neq 0$:	Disable all interrupts
$\hat{k} = 2$ and $b = 0$:	Enable all external interrupts
$\hat{k} = 2$ and $b \neq 0$:	Disable all external interrupts
$\hat{k} = 3$ and $b = 0$:	Enable external interrupt on channel j
$\hat{k} = 3$ and $b \neq 0$:	Disable external interrupt on channel j

Ignore y for this instruction.

67 TERMINATE C_j^{\wedge}

Terminate Output buffer or all buffers as designated by k as follows:

$\hat{k} = 0$:	Terminate the output buffer on channel \hat{j}
$\hat{k} = 1$:	Terminate the external function buffer on channel \hat{j}
$k = 2$:	Terminate all output and external function buffers

For $\hat{k} = 1$ or 2 , if \hat{j} specifies an output register being used by an intercomputer group, a resume signal is simulated on that group.

In all cases, no output buffer monitor interrupt occurs. Ignore the b and y designators for this instruction.

70 REPEAT

Clear B7 and transmit Y to B7. If Y is nonzero, transmit (j) to r (repeat designator register), thereby initiating the repeat mode. If Y is zero, skip the next instruction. All interrupts are locked out during operation in the repeat mode.

The repeat mode executes the instruction immediately following the Repeat instruction Y times and decreases the count in B7 by one each time. B7 contains the number of executions remaining throughout the repeat mode.

If no skip condition is met for the repeated instruction, the repeat mode is terminated by exhausting the repeat count, and the instruction following the repeated instruction is executed. If the skip condition for the repeated instruction is met, the repeat mode terminates, and the instruction following the repeated instruction is skipped. Following the repeat mode termination, the count remains in B7.

In no way does the repeat mode alter a repeated instruction as stored in memory.

The three lower order bits of the r designator (from j of instruction 70) affect the operand indexing as follows:

- r = 0: Do not modify the operand address of the repeated instruction after each individual execution
- r = 1: Increase the operand address of the repeated instruction by one after each execution of the repeated instruction
- r = 2: Decrease the operand address of the repeated instruction by one after each execution of the repeated instruction
- r = 3: Repeat the initial B register modification of the repeated instruction before each execution
- r = 4: Do not modify the operand address of the repeated instruction after each individual execution. If the repeated instruction is a Replace instruction, the operand address is incremented by (B6) for the store portion of the Replace instruction
- r = 5: Increase the operand address of the repeated instruction by one after each execution of the repeated instruction. If the repeated instruction is a Replace instruction, the operand address is incremented by (B6) for the store portion of the Replace instruction
- r = 6: Decrease the operand address of the repeated instruction by one after each execution of the repeated instruction. If the repeated instruction is a Replace instruction, the operand address is incremented by (B6) for the store portion of the Replace instruction
- r = 7: Repeat the initial B register modification of the repeated instruction before each execution. If the repeated instruction is a Replace instruction, the operand address is incremented by (B6) for the store portion of the Replace instruction

71 B SKIP ON B_j

If the contents of the B register specified by j are equal to Y, skip the next instruction in the current sequence and proceed to the following instruction. Clear the B register specified by j.

If the contents of the B_j register are not equal to Y, proceed to the next instruction in the sequence in a normal manner and increase the contents of the B_j register by one.

The branch condition designator, j, is used to designate the selected B register in this instruction and is not available for its normal function.

72 B JUMP ON B_j

If the contents of the B_j register are nonzero, execute a jump in program to address Y. Reduce the contents of the B_j register by one.

If the contents of the B_j register are zero, proceed to the next instruction in a normal manner. Do not alter the contents of the B_j register.

The branch condition designator, j, is used to designate the selected B register in this instruction and is not available for its normal function. If the jump condition is satisfied, the lower order bits of Y become the address of the next instruction and the beginning of a new program sequence. The higher order bits of Y are not used in this instruction.

73 INPUT BUFFER ON C_j[^] (Without Monitor Mode)

Transfer Y as designated by \hat{k} to the input buffer control register $100 + \hat{j}$ and initiate as input buffer on channel \hat{j} . The next current input address is maintained throughout the buffer process in the lower order 15 bits of the control register.* This storage address is incremented by one during each individual word transfer. Subsequent to this instruction, each individual transfer will be executed at a rate determined by the external equipment. This mode continues until it is superseded by a subsequent input buffer or termination instruction on the same channel or until the higher order half and the lower order half of the control register are equal, whichever occurs first.

$\hat{k} = 0$: Store Y in the lower order half of control register $100 + \hat{j}$, leaving the higher order half undisturbed

$\hat{k} = 1$: Store (Y)_L in the lower order half of control register $100 + \hat{j}$, leaving the higher order half undisturbed

$\hat{k} = 2$: Is not permitted

$\hat{k} = 3$: Store (Y) in the control register $100 + \hat{j}$

*Refer to page 1 of this section

74 OUTPUT BUFFER ON $C_j^{\hat{j}}$ (Without Monitor Mode)

Transfer Y as designated by \hat{k} to output buffer control register $120 + \hat{j}$ ($k \neq 2$), or external command buffer control register $140 + \hat{j}$ ($k = 2$), and initiate the output buffer or the external function buffer respectively on channel \hat{j} . The next current address is maintained throughout the buffer process in the lower order 15 bits of the control register.* This storage address is incremented by one during each individual word transfer. Subsequent to this instruction, each individual transfer is executed at a rate determined by the external equipment. This mode continues until it is superseded by a subsequent output/external function buffer or termination instruction on the same channel or until the higher order half and the lower order half of the control register are equal, whichever occurs first.

- $\hat{k} = 0$: Store \underline{Y} in the lower order half of control register $120 + \hat{j}$, leaving the higher order half undisturbed
- $\hat{k} = 1$: Store $(\underline{Y})L$ in the lower order half of control register $120 + \hat{j}$, leaving the higher order half undisturbed
- $\hat{k} = 2$: Store (\underline{Y}) in external function buffer control register $140 + \hat{j}$
- $\hat{k} = 3$: Store (\underline{Y}) in output control register $120 + \hat{j}$

75 INPUT BUFFER ON $C_j^{\hat{j}}$ (With Monitor Mode)

Transfer Y as designated by \hat{k} to input buffer control register $100 + \hat{j}$ and initiate an input buffer on channel \hat{j} . The next current input address is maintained throughout the buffer process in the lower order 15 bits of the control register.* This storage address is incremented by one during each individual word transfer. Subsequent to this instruction, each individual transfer will be executed at a rate determined by the external equipment. This mode continues until it is superseded by a subsequent input buffer or termination instruction on the same channel or until the higher order half and the lower order half of the control register are equal, whichever occurs first. Upon completion of the buffer operation, a monitor interrupt occurs, and the next program instruction executed comes from interrupt entrance address $00040 + \hat{j}$.

- $\hat{k} = 0$: Store \underline{Y} in the lower order half of control register $100 + \hat{j}$, leaving the higher order half undisturbed
- $\hat{k} = 1$: Store $(\underline{Y})L$ in the lower order half of control register $100 + \hat{j}$, leaving the higher order half undisturbed
- $\hat{k} = 2$: Is not permitted
- $\hat{k} = 3$: Store (\underline{Y}) in control register $100 + \hat{j}$

76 OUTPUT BUFFER ON $C_j^{\hat{j}}$ (With Monitor Mode)

Transfer Y as designated by \hat{k} to output buffer control register $120 + \hat{j}$ ($k \neq 2$) or external buffer control register $140 + \hat{j}$ ($k = 2$) and initiate the output buffer or the external function buffer respectively on channel \hat{j} . The next current address is maintained throughout the buffer

* Refer to page 1 of this section

process in the lower order 15 bits of the control register.* This storage address is incremented by one during each individual word transfer. Subsequent to this instruction, each individual transfer is executed at a rate determined by the external equipment. This mode continues until it is superseded by a subsequent output external function buffer or Termination instruction on the same channel or until the higher order half and the lower order half of the control register are equal, whichever occurs first. Upon completion of the buffer operation, a monitor interrupt occurs, and the next program instruction executed comes from interrupt entrance register $00060 + \hat{j}$ if $\hat{k} \neq 2$ and from $00500 + \hat{j}$ if $\hat{k} = 2$.

- $\hat{k} = 0$: Store Y in the lower order half of control register $120 + \hat{j}$, leaving the higher order half undisturbed
- $\hat{k} = 1$: Store (Y)L in the lower order half of control register $120 + \hat{j}$, leaving the higher order half undisturbed
- $\hat{k} = 2$: Store (Y) in external function buffer control register $140 + \hat{j}$
- $\hat{k} = 3$: Store (Y) in output control register $120 + \hat{j}$

00, 7700, AND 7777 IMPROPER CODES

Function codes 00, 7700, and 7777 are improper codes, which, if translated, cause a program and an automatic jump to a location controlled by the Automatic Recovery switch.

77 07 NORMALIZE

Function codes 00, 7700, and 7777 are improper codes, which, if translated, cause a program fault and an automatic jump to a location controlled by the Automatic Recovery switch.

This instruction uses

6	6	3	2	13
77	f	b	s	y

as the format for the instruction. The \hat{k} designator is not used but an assumed value $\hat{k} = 3$ is forced into the instruction translator when executed. In all other 77 instructions y when used as an address is limited to 8 bits addressing 256 words of memory. However, with normal indexing being performed, an effective address Y can be generated to select any location in memory.

77 60 ENTER SPECIAL REGISTER

Clear the SR specified by \hat{j} . Then transmit Y_{3-0} to this SR.

77 61 ENTER INPUT SR I/O

Clear the Input Special register for the channel specified by \hat{j} . Then transmit Y_{1-0} to this Input SR and clear the Continuous Data Mode on this channel.

* Refer to page 1 of this section

77 62 ENTER OUTPUT SR I/O

Clear the Output Special register for the channel specified by \hat{j} . Then transmit Y_{1-0} to this Output SR and clear the Continuous Data Mode on this channel.

77 63 ENTER EXTERNAL FUNCTION SR I/O

Clear the External Function Special register for the channel specified by \hat{j} . Then transmit Y_{1-0} to this EF SR and clear the Continuous Data Mode on this channel.

77 64 ENABLE INPUT CDM

Enable the Continuous Data Mode for Input on the channel specified by \hat{j} . Ignore b, s, k, and y.

77 65 ENABLE OUTPUT CDM

Enable the Continuous Data Mode for Output on the channel specified by \hat{j} . Ignore b, s, k, and y.

77 66 DISABLE INPUT CDM

Clear the Continuous Data Mode for Input on the channel specified by \hat{j} . Ignore b, s, k, and y.

77 67 DISABLE OUTPUT CDM

Clear the Continuous Data Mode for Output on the channel specified by \hat{j} . Ignore b, s, k, and y.

77 70 STORE SPECIAL REGISTER

Store the contents of the Special Register specified by \hat{j} in \underline{Y} .

77 71 STORE INPUT SR I/O

Store the contents of the Input Special register for the channel specified by \hat{j} in \underline{Y} .

77 72 STORE OUTPUT SR I/O

Store the contents of the Output Special register for the channel specified by \hat{j} in \underline{Y} .

77 73 STORE EF SR I/O

Store the contents of the External Function Special register for the channel specified by \hat{j} in \underline{Y} .

77 74 DISABLE 17-BIT ADDRESS MODE

Disable the expanded memory mode and enable the computer to interpret a maximum of 15 bits for addresses thereby limiting it to operation within its internal 32K memory. All SR registers are ignored and y becomes a 15-bit value in Type I instructions.

77 75 ENABLE 17-BIT ADDRESS MODE

If the MODE switch is on Expanded Memory position enable the computer to assemble and interpret addresses up to 17 bits in length. The Special Registers become functional and s is interpreted when applicable. This is a do nothing instruction if the MODE switch is in 32K memory position.

INTRODUCTION

The Assembly System (AS-1) translates mnemonic code into Unit Computer machine instructions. This code, called the AS-1 Input (Source) Language, aids in defining information processing problems. The object program (that which is produced by AS-1 from the Input Language) runs on the Unit Computer and thus solves the problems (see Figure 1).

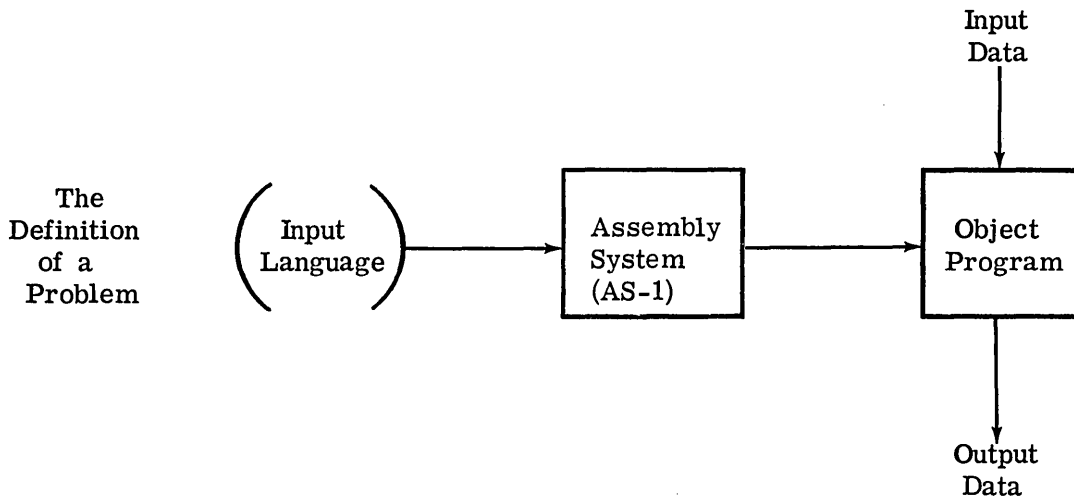
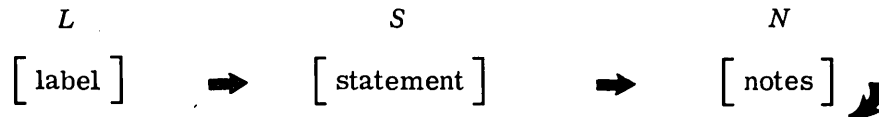


Figure 1. AS-1 Solution of a Problem

INPUT LANGUAGE ORGANIZATION

OPERATIONS

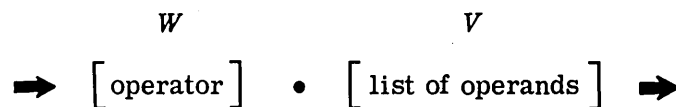
AS-1 Input Language consists of a large repertoire of *operations* which either state certain facts about entities in the input language, or perform a well-defined logical action. A list of these *operations*, which are the basic unit of the language, defines a problem. They have the configuration which follows.



- L* - The *label* is a *name* that uniquely identifies the operation; it consists of up to ten alphanumeric characters, but never starts with 0, X, or a number and never consists of only A, Q, B⁰ through B⁷, or C⁰ through C¹⁷. Only an operation which is referred to in the *statement* of another operation requires a *label*. A *tag* is a reference to the *label* of one operation in the *statement* of another; it has the same notation restrictions as a *label*
- S* - The *statement* defines the operation; it is always required
- N* - Descriptive *notes* may follow the statement; they are for the user's convenience and in no way alter the meaning of the operation
- - The straight arrow is a major *separator* which delimits the *statement*
- ↷ - The curved arrow designates the end of each *operation* and signals the start of the next one. It must precede the first *operation* of a program

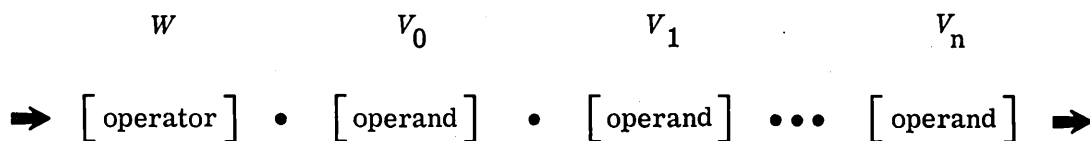
STATEMENTS

In general, the statement involves two sections:

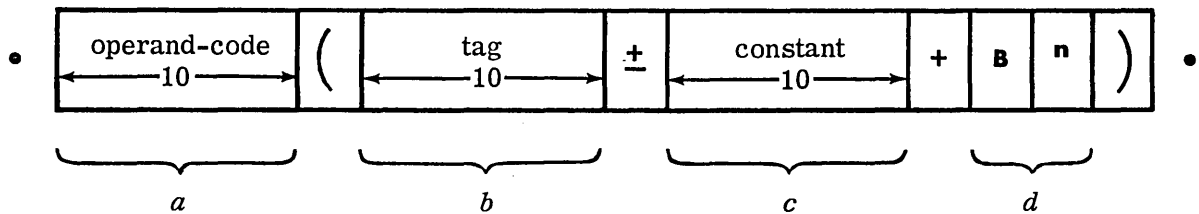


- W* - The *operator* specifies the general characteristics of the operation
- V* - The *operands* further define the operation. The number and the configuration of the *operands* depend upon the *operator* used
- - The *point separator* separates the *operator* from the *operand* section and separates the *operands* from one another

A statement, further divided, appears as:



The operands consist of basic terms a , b , c , d as follows:



- - is the point separator. It delimits the operand
- () - are parentheses which indicate an exact meaning
- a - is an alphanumeric operand-code symbol, consisting of up to 10 characters, which adds some basic information to the operator or to the terms following it. The operand-code may be alone, with other terms, or completely absent
- b - is an alphanumeric tag symbol, consisting of up to ten characters, which refers to a label or a number
- c - is an octal or decimal constant. If a positive constant is alone, the + is not required. A decimal number is a string of digits followed by **D**. The constant consists of up to 5 characters when it appears with the tag. It consists of up to 10 characters without the tag
- d - is a B register which modifies the rest of the expression. If **Bⁿ** is alone, the + is not required. The content of **Bⁿ** is implied in either case

Many operands consist of only term a . The majority, however, may consist of combinations of the specified terms. Of these, three classes are most prominent: 1) Read-class, 2) Store-class, and 3) Replace-class. If operands do not satisfy the forms of Read-, Store-, or Replace-classes, a description and use of those operands appear with the discussions of each operation.

READ-CLASS

Read-class operands can assume the following forms:

CONSTANTS: Read-class operands may consist of term c only. This term may be of up to ten digits, either negative or positive. If a decimal number, it is a string of digits followed by a **D**. If the constant is positive, the + is not required.

Examples:

7235
49678D
-1
-496782D
-0

Constants may be modified by preceding them with an **X**. The **X** means *extended*; that is, the bits of the upper half of a register are flushed with the value of the upper bit of the lower half. **X** has meaning only if it precedes 5 digits and the upper digit is 4, 5, 6, 7, or negative numbers.

Examples:

X40000
X77774

TAGS: Read-class operands may consist of term *b* only. The tag may consist of up to ten alphanumeric characters; it represents a 5-digit octal number. The number is usually the address of a memory location which is specified by an operation label.

Examples:

RANGE
CAT056

Tags may be modified by an increment consisting of a 5-digit octal constant or a 4-digit decimal and/or the content of a B register. If the tag refers to a labeled statement, the constant and B register specify an increment which locates a statement relative to the labeled statement.

Examples:

RANGE+2
CAT056-5+B2
FIDDLE+B5

The sign bit of the value represented by a tag with its increment is extended if term *a* is **X**.

Examples:

XRANGE
XCAT056-5+B2

CONTENTS OF: Read-class operands may specify the contents of memory locations. An operand-code, term *a*, and parentheses must always be present. The memory location can be specified by any combination of terms *b*, *c*, and *d*, that is, a tag, an absolute address, a

tag plus or minus a 5-digit increment, or any of the preceding modified by the content of a B register. The operand-code specifies the form of the numeric value in the following manner when the operand-code (term *a*) is:

- L** - the numeric value is the lower 15 bits of the memory location
- U** - the numeric value is the upper 15 bits of the memory location, normalized to the right of a 30-bit register
- W** - the numeric value is the whole 30 bits of the memory location
- LX** - the numeric value is the lower 15 bits of the memory location, with the sign extended to make a 30-bit word
- UX** - the numeric value is the upper 15 bits of the memory location, normalized to the right of a 30-bit register and the sign is extended

Examples:

L(COW+5+B6)

UX(CAT)

W(B5)

The last example means "the whole contents of the memory location whose address is in **B5**."

REGISTERS: Read-class operands may refer to the contents of registers. The registers are:

A , B1 , B2 , B3 , B4 , B5 , B6 or B7.

A constant may modify the content of the B registers, but not of the A register. For example:

B1+3

B5-12D

If a sign extension is desired on the content of a B register, an **X** precedes the register name. For example:

XB3

XB4-2

STORE-CLASS

Store-class operands can assume the following forms:

CONTENTS OF: Store-class operands may specify a memory location into which a numeric value is stored. An operand-code, term *a*, and parentheses must always be present. The

memory location can be specified by any combination of terms *b*, *c*, and *d*, that is, an absolute address, a tag plus or minus a 5-digit increment, or any of the preceding modified by the content of a B register. The operand-code designates the position and form of the numeric value when it is stored. If the operand-code (term *a*) is:

- L** - the numeric value treated as a 15-bit integer is stored in the lower 15 bits of the memory location. (The upper 15 bits remain unchanged)
- U** - the numeric value treated as a 15-bit integer is stored in the upper 15 bits of the memory location. (The lower 15 bits remain unchanged)
- W** - the numeric value treated as a 30-bit integer is stored in the memory location
- CPL** - the numeric value is complemented and stored as **L** above
- CPU** - the numeric value is complemented and stored as **U** above
- CPW** - the numeric value is complemented and stored as **W** above

Examples:

L(COW+5+B6)
CPU(CAT)
W(B5)

REGISTERS: Store-class operands may specify a register into which a numeric value is stored. These registers are:

A or Q.

REPLACE-CLASS

Replace-class operands can assume only the following form:

CONTENTS OF: A Replace-class operand specifies a numeric value which is in a memory location, and after some function is performed, specifies the same memory location into which the resulting numeric value is stored. The memory location can be specified by any combination of terms *b*, *c*, and *d*, that is, a tag, an absolute address, a tag plus or minus a 5-digit increment, or any of the preceding modified by the content of a B register. The operand-code specifies the form of the numeric value in the following manner when the operand-code (term *a*) is:

- L** - the numeric value is the lower 15 bits of the memory location; after the basic function is performed, the resulting numeric value treated as a 15-bit integer is

stored in the lower 15 bits of the same memory location.* (The upper 15 bits remain unchanged)

- U** - the numeric value is the upper 15 bits of the memory location, normalized to the right of a 30-bit register; after the basic function is performed, the resulting numeric value treated as a 15-bit integer is stored in the upper 15 bits of the same memory location.* (The lower 15 bits remain unchanged)
- W** - the numeric value is the whole 30 bits of the memory location; after the basic function is performed, the resulting numeric value is stored in the same memory location*
- LX** - the numeric value is the lower 15 bits of the memory location, with the sign extended to make a 30-bit word; after the basic function is performed, the resulting numeric value is treated as **L** above*
- UX** - the numeric value is the upper 15 bits of the memory location, normalized to the right of a 30-bit register, and the sign is extended; after the basic function is performed, the resulting value is treated as **U** above*

Examples:

L(COW+5+B6)
UX(CAT)
W(B5)

*Exception: The storage reference of the replace may be altered by B⁶ when following a **RePeaT** operation. (See Mono-operations, **RPT**, page 29.)

CLASSES OF OPERATIONS

Operations divide into two major classes: 1) AS-1 programming operations 2) assembler-control operations.

1. AS-1 programming operations express either 1) machine instructions in a mnemonic form, 2) groups of machine instructions which perform a simple function, or 3) input-output control on the standard external equipment. These operations generally refer to memory locations by their address (a label or a tag), to machine registers by name (A, Q, Bⁿ), and to data by the 30-bit word concept. A thorough knowledge of the Unit Computer is necessary to use these operations
2. *Assembler*-control operations specify information to the assembler executive routine. This information states the process it is to perform, such as assembling, editing, etc

Operations further divide into two categories: 1) Declarative, and 2) Action.

Declarative operations state certain facts about entities in the input language of which they are part. They 1) adapt the program to a specific memory configuration and input-output capabilities of the computer, and 2) identify different segments of input to the assembler. No machine instructions result directly from the *declarative* operations.

Action operations have operational or dynamic meaning and state rules of processing which give rise to actual machine instructions. Each section pertaining to these operations specifies to which category each operation belongs.

AS-1 ASSEMBLER GROUND RULES

The following Ground Rules are offered as aids to the programmer:





Case:

1. It is recommended that upper-case alphabetical and numeric characters be used in coding. This will eliminate a large portion of case shifting when preparing paper tapes
2. An upper-case symbol is required at the beginning of a tape

Coding Format:

3. The operator is always the first word of an L_0 operation statement, if no label is present.
4. The symbols + and - specify increments. Both + and - may be used with a number. B registers can be used only with a + sign
5. The j-designator operand, if used, appears as the last operand of a mono-operation
6. In a mono-code operation, the tag, basic term b , *must* precede the constant, term c , and the B register, term d . The constant and B register terms can be interchanged (see Programming Language, Page 3)

Control Symbols:

7. A  must precede and follow the header. A  also separates all operations
8. A  must always precede the statement
9. The operator and each of its operands are separated with a point separator, •
10. Parenthesis symbols indicate, in the assembler-language operations, grouping of interrelated operands by separating an operand code from the term(s) that it is to modify, i.e., **U(DOG6+267);L(DOG4+1);W(CAT4)**
11. A  must precede the notes
12. A double period following a carriage return indicates the end of paper tape

Decimal Numbers:

13. A decimal number is indicated by being followed by the letter **D**; i.e., **9286D**. Numerical characters 8 and 9 may then be used
14. Signed decimal numbers cannot exceed eight digits, since the sign and the **D** are included in the ten character maximum. Allowable decimal formats are:

+43274685D

-35942738D

536870911D*

Identifiers:

15. Since L_1 ID's are in octal notation, corrections to the L_1 ID's must be in octal
16. The HEADER operation of an L_0 program is given an L_1 ID of zero; the second operation has an L_1 ID of 1; numbering is sequential thereafter

Labels:

17. Only essential operation items need be labeled, i.e., header items, the **ENTRY** operation of a program, operation items referred to in the program, etc
18. Start each subroutine with a labeled **ENTRY** operation. The label should be the subroutine name
19. When used, the **ENTRY** operation of any routine *must* have a label
20. Use only alphanumeric characters in labels

Mono-operations:

21. Be sure to observe the distinctions between Read-class, Store-class, and Replace-class operands in coding mnemonic instructions
22. The V_0 operand in mono-operations is either r , a register designation, or e , a modifying expression; in a few mono-operations, V_0 is absent
23. j -operands are required in all **COM** operations: **COM•A**; **COM•Q**; **COM•AQ**; **COM•MASK**

*Maximum decimal number permitted

24. j -operands are not permitted if r in the V_0 operand is B1 through B7 or C0 through C17 in the operations: **ENT**; **STR**; **BJP**; **BSK**; **CL**; **IN**; **OUT**

Poly-operations:

25. Avoid, if possible, leaving information in registers A, Q, and B7 while stating poly-operations
26. Do not use a j -operand in the operation preceding a poly-operation except for **JP** and **RJP** operations

Register B:

27. The content of register Q cannot be transferred directly to a B register. It can be transferred only indirectly by first storing it in A or memory
28. Do not decrement a B register through zero (see **INCREMENT** operation)
29. An XB^n in a mono-code statement will give a sign extension of the content of register B^n on transfer to a 30-bit location or register

Miscellaneous

30. Headers are required on all input tapes
31. Code delete, stop code, color shift code, or leader may appear at any place on paper tape since they are discarded by the assembler program
32. The **ENT** • **Y-Q** • [read-class operand] • **AZERO** operation provides a convenient method of making a non-masked comparison
33. To clear an area of 1 word, the mono-code **CL** operation is more efficient time-wise than the poly-code **CLEAR** operation. A word number of 0 used with the **CLEAR** operation merely produces a "do-nothing" on delay instruction in the object program
34. 0 is translated as a zero in AS-1 code
35. A combined read-in of both normal-allocation and relative-allocation tapes results in the following:
 1. If **REL-ALLOC** tape precedes the **ALLOCATION** tape, the assembler suppresses all assembler-generated tags contained on the **ALLOCATION** tape. The assembler reassigns these tags

2. If the **ALLOCATION** tape precedes the **REL-ALLOC** tape, the compiler recognizes and accepts all assembler-generated tags on the **ALLOCATION** tape
36. Operations containing references to C-channels must use octal channel designations, i.e., C0-C7, C10-C17. Cⁿ is treated as a complete channel name, not a channel number

Example:

➡ **TERM • C11 (not C9D) • INPUT** ➡

MONO-OPERATIONS

<u>Operation</u>	<u>Page</u>
ENTer	3
SToRe	5
SToRe (channel)	6
NO-OP	7
CLear	8
Right SHift	9
Left SHift	10
ADD	11
SUBtract	12
MULTiPLY	13
DIVide	14
Square RoOT	15
COMpare	16
ComPlement	18
SElective	19
RePlace	20
Replace SElective	21
JumP	23
Return JumP	26
B JumP	27
B SKip	28
RePeaT	29
INput	31
OUTput	32
EXternal COMmand	33
EXternal COMmand-Multi Word	34
TERMinate	35
Set Interrupt Lockout	36

MONO-OPERATIONS (Cont.)

<u>Operation</u>	<u>Page</u>
Set Interrupt Lockout - EXternal	37
Remove Interrupt Lockout	38
Remove Interrupt Lockout - EXternal	39
Remove Interrupt Lockout and Jump	40

MONO-OPERATIONS

Operations which mnemonically express a machine instruction are *mono-operations*. Each mono-operation in the source language (L_0) is translated by AS-1 to one machine instruction in the object language (L_4), i.e., the translation is one-to-one.

Mono-operations have a definite format:

W V_0 V_1 V_2

➔ [operator] • [allied operand] • [y -operand] • [j -operand] ➔

W - gives a mono-code which defines a class of machine instructions, such as *enter*, *store*, etc

V_0 - gives added information which further defines a machine instruction, thus is called the *allied operand*. The allied operand may specify a register, r , or a simple logical or arithmetic expression, e . It is absent in some operations

Some of the mono-codes are multipurpose. They form a class of operations. In such cases, the allied operand combines with and modifies the operator to generate a distinct instruction in the object language. An example is the selective operator, **SEL**. When combined with the V_0 operand, **SET**, it generates a computer function code f of 50. Similarly, **SEL** • **CP** generates an f of 51, **SEL** • **CL** generates an f of 52, and **SEL** • **SU** generates 53. Another example of a multipurpose operator is **ADD**:

ADD • **A**
ADD • **Q**
ADD • **LP**

In each case the assembler generates a separate machine code instruction.

V_1 - specifies either 1) a numeric value, 2) the address of a memory location, or 3) a register (A, Q, or B^n). The y -operand is a Read-class operand, a Store-class operand, or a Replace-class operand (see the Basic Information, Section II-A, for a discussion of these). V_1 is absent in some operations

Note: Subsequent references to y include all of the above interpretations unless otherwise specified.

V_2 - specifies a *j*-operand which is primarily used for jump or skip determination or for repeat status interpretation. The action caused by these may be conditional or unconditional as directed by the operand used. Seven *j*-operands are applicable to the majority of mono-operations; these are called *normal j*-operands. Certain operations require the usage of unique *j*-operands, called *special j*-operands. These are explained in the discussions of those operations. The *j*-operand is absent on other operations

Normal j-operands are as follows:

Operand, <i>j</i>	Performance
(blank)	Will not skip the next operation.
SKIP	Skip the next operation unconditionally
Q POS	Skip the next operation if Q is positive
Q NEG	Skip the next operation if Q is negative
A ZERO	Skip the next operation if A is zero
A NOT	Skip the next operation if A is non-zero
A POS	Skip the next operation if A is positive
A NEG	Skip the next operation if A is negative

Special j-operands are required for use with the following operations: Jump, Return Jump, Divide, Repeat, Add Q, Subtract Q, and all non-mask Compares.

Mono-code operators, combining with allied operands in most cases, are capable of generating all the irredundant instructions of the computer's repertoire. Additional operations such as: "do nothing" operation, **NO-OP**; and "complement a register", **CP**; produce single instructions which achieve such actions which are not apparent in the names of computer function codes.

ENTER Operation:

$$\Rightarrow \overset{W}{\text{ENT}} \cdot \overset{V_0}{[r \text{ or } e]} \cdot \overset{V_1}{[y]} \cdot \overset{V_2}{[j]} \Rightarrow$$

The **ENT** operation either 1) first clears the register, r , and then transmits the numerical value expressed by y to register r , or 2) performs the function expressed by e and enters the result in A. The **Y** that appears in e refers to the numerical value which y defines.

V_0 - designates the register into which the numerical value is entered; r can be:

A, Q, or B0 through B7

or

V_0 - states one of several simple arithmetic or logical expressions, e , to be performed, which are then entered into A. These are:

Expression, e	Performance
(1) LP	LP (y) (Q)* \Rightarrow A
(2) Y+Q	$y + Q \Rightarrow$ A
(3) Y-Q	$y - Q \Rightarrow$ A

V_1 - gives a Read-class operand that defines y

V_2 - specifies a normal j -operand; it is optional when V_0 is **A, Q, Y+Q** or **Y-Q**

or

V_2 - specifies a j -operand when V_0 is **LP**. In this case the operation permits all normal j -operands except **QPOS** and **QNEG**. Substituted for **QPOS** and **QNEG** are two special j -operands:

EVEN - Even parity (even number of "ones" in A)

ODD - Odd parity (odd number of "ones" in A)

Note: If V_0 is B^0 through B^7 , V_2 must be absent.

*LP (y) (Q) means the bit-by-bit product of (y) and (Q)

Examples:

➡ ENT • Y+Q • UX(SACK+B4) ➡

➡ ENT • Q • X77776 • AZERO ➡

➡ ENT • LP • W(BAG9+3) • EVEN ➡

SToRe Operation:

$$\Rightarrow \overset{W}{\text{STR}} \cdot \overset{V_0}{[r, \text{ or } e]} \cdot \overset{V_1}{[y]} \cdot \overset{V_2}{[j]} \Rightarrow$$

The **STR** operation stores one of the following: 1) the content of register r , or 2) the result of an expression, e , in a storage location delegated by y .

V_0 - designates the register, r , whose content is stored in a memory location. V_0 can be: **A, Q, B0 through B7**

or

V_0 - states one of several simple arithmetic or logical expressions, e , to be performed, which are then stored in a memory location. These are:

Expression, e	Performance
(1) LP	$\text{LP}(A)(Q)^* \Rightarrow y$
(2) A + Q	$A + Q \Rightarrow y$ and A
(3) A - Q	$A - Q \Rightarrow y$ and A

V_1 - gives a Store-class operand that defines a memory location y

V_2 - specifies a normal j -operand. The j -operand is optional except where r is **B0** through **B7**

Examples:

\Rightarrow **STR** • **B7** • **L(PEN-5)** \Rightarrow

\Rightarrow **STR** • **C14** • **W(INK)** \Rightarrow

\Rightarrow **STR** • **A-Q** • **W(PAPER)** • **QNEG** \Rightarrow

* $\text{LP}(A)(Q)$ means the bit-by-bit product of A and Q

SToRe (channel) Operation:

W V₀ V₁ V₂

➔ **STR** • [channel] • [y] • [sub-function code] ➔

This operation provides the interrupt word at the specified location.

V₀ - Specifies the channel of the desired interrupt word. Channels C0 - C7, C10 - C17 are permitted. V₀ may specify a name which is identified by a **MEANS** operation or a **CHAN-SET** tape

V₁ - Specifies the location at which the interrupt word is to be stored. This operand may specify only the whole contents of a memory location

V₂ - Specifies the sub-function code:

(absent)** - means the contents of the appropriate address reserved for interrupt word storage will be transferred to Y as specified by V₁. This instruction is necessary with new line equipment to reset the Interrupt Request

FORCE - provides forcing the word on the line to be stored at Y as specified by V₁. Program will hold until the word is read causing an Input Acknowledge signal (this is an abnormal mode used for testing some equipments)

Examples:

➔ **STR** • C3 • W(CAT) ➔

➔ **STR** • SMPCHAN • W(DOG) • **FORCE** ➔

**The Input Acknowledge is set automatically when the interrupt word is read into the special address, which occurs in both old and new line equipment

NO-OP Operation:

W
➡ **NO-OP** ➡

The **NO-OP** operation is a "do nothing" operation. It generates a 12000 00000 in the object program, causing the computer to move on to the next operation.

CL ear Operation:

$\begin{matrix} W & & V_0 \\ \Rightarrow & \mathbf{CL} & \bullet & [r \text{ or } y] & \Rightarrow \end{matrix}$

The **CL** operation clears the memory location specified by y or the register specified by r .

V_0 - designates the register to be cleared; r can be:

A , Q , B1 through B7 ,

or

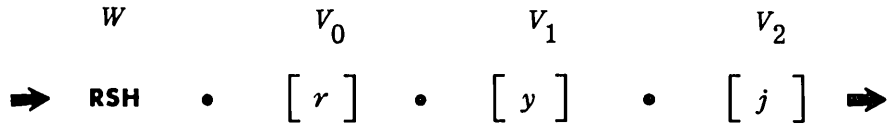
V_0 - gives a Store-class operand that defines y

Examples:

$\Rightarrow \mathbf{CL} \bullet \mathbf{Q} \Rightarrow$

$\Rightarrow \mathbf{CL} \bullet \mathbf{L(GIMME)} \Rightarrow$

Right SHift Operation:



The **RSH** operation shifts the content of the register, r , to the right y bit positions. As the information is shifted, the original sign bit replaces the higher order bits of register r ; the lower order bits are shifted off the end.

Only the lower-order 6-bits of y are recognized. The higher-order 24 bits are ignored.

V_0 - designates the register that the operation shifts; r can be:

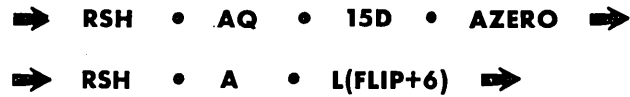
A , Q , or AQ

AQ represents the 60-bit register consisting of A and Q

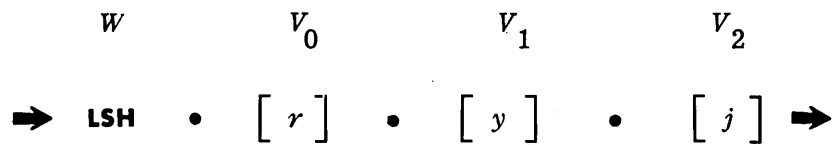
V_1 - gives a Read-class operand that defines y

V_2 - specifies a normal j -operand; it is optional

Examples:



Left SHift Operations:



The **LSH** operation shifts the content of the register, r , to the left y bit positions. The shift is circular; the low-order bits of r are replaced by the upper-order bits. Only the lower-order 6 bits of y are recognized. The higher-order 24 bits are ignored.

V_0 - designates the register that the operation shifts; r can be:

A , Q , or AQ

AQ represents the 60-bit register consisting of A and Q

V_1 - gives a Read-class operand that defines y

V_2 - specifies a normal j -operand; it is optional

Examples:

➔ **LSH** • **A** • **L(CAT)** • **QNEG** ➔

➔ **LSH** • **Q** • **B4** ➔

ADD Operation:

W V_0 V_1 V_2
⇒ ADD • [r or e] • [y] • [j] **⇒**

The **ADD** operation either 1) adds the numeric value expressed by y to the contents of r and replaces the result in r , or 2) performs the expression, e , and then adds its result to A.

V_0 - designates the register to which the numerical value is added

Register, r	Performance
A	$A + y \Rightarrow A$
Q	$Q + y \Rightarrow Q$

or

V_0 - states a logical function, e

Expression, e	Performance
LP	$A + LP(y)(Q)^* \Rightarrow A$

V_1 - gives a Read-class operand that defines y

V_2 - specifies a normal j -operand if V_0 is **A** or **LP**. If V_0 is **Q**, **AZERO** and **ANOT** are not permitted; **QZERO** and **QNOT** are substituted instead. V_2 is optional

Examples

⇒ ADD • LP • W(BOOK) ⇒
⇒ ADD • Q • 12D • QZERO ⇒

*LP(y)(Q) means the bit-by-bit product of y and Q

SUB tract Operation:

W V_0 V_1 V_2
 \Rightarrow **SUB** • [r or e] • [y] • [j] \Rightarrow

The **SUB** operation either 1) subtracts the numeric value expressed by y from the contents of r and replaces the result in r , or 2) performs the expression, e , and then subtracts its result from A.

V_0 - designates the register from which the numerical value is subtracted

Register, r	Performance
A	A - y \Rightarrow A
Q	Q - y \Rightarrow Q

or

V_0 - states a logical function, e

Expression, e	Performance
LP	A - LP(y)(Q)* \Rightarrow A

V_1 - gives a Read-class operand that defines y

V_2 - specifies a normal j -operand if V_0 is **A** or **LP**. If V_0 is **Q**, **AZERO** and **ANOT** are not permitted. **QZERO** and **QNOT** are substituted instead. V_2 is optional

Examples:

\Rightarrow **SUB** • A • 12D \Rightarrow
 \Rightarrow **SUB** • Q • B6 \Rightarrow

*LP(y)(Q) means the bit-by-bit product of y and Q

MUL multiply Operation:

W V_0 V_1 V_2

⇒ **MUL** • [absent] • [y] • [j] ⇒

The **MUL** operation multiplies Q by the numerical value expressed by y , leaving the double length product in AQ. All numbers involved are treated as integers.

V_0 - always absent

V_1 - gives a Read-class operand that defines y . A is not permitted

V_2 - specifies a normal j -operand

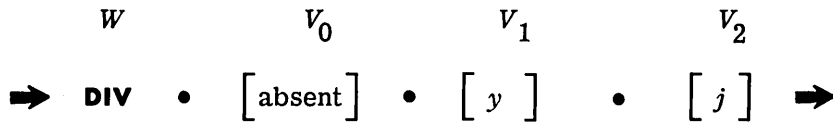
The actual multiplication is performed with positive numbers only; therefore, if the original sign bits of y and Q are not similar, an *end correction* is made by complementing the product. The branch condition j -operand is interpreted prior to the *end correction*, thus **ANEG** has no effect and **APOS** always gives an unconditional skip.

Examples:

⇒ **MUL** • L(PAPER-2) ⇒

⇒ **MUL** • 4 ⇒

DIV ide Operation:



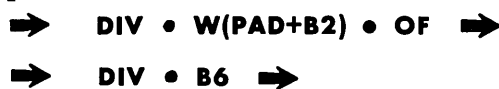
The **DIV** operation divides AQ by the numerical value expressed by y , leaving the quotient in the Q register and the remainder in the A register. The remainder bears the same sign as the quotient.

- V_0 - always absent
- V_1 - gives a Read-class operand that defines y . A is not permitted
- V_2 - specifies a skip-the-next-operation condition

Operand, j	Condition
(blank)	Does not skip on divide
SKIP	Unconditional skip
OF	Skip if there is an <i>overflow</i>
NOOF	Skip if there is no <i>overflow</i>
AZERO	Skip if $A = 0$
ANOT	Skip if $A \neq 0$
APOS	Skip if A is positive
ANEG	Skip if A is negative

Note: There is no indicator on the console to represent a *divide fault*. However, by coding each operation with a j of **OF**, a program test for a *divide fault* is automatic. With this selection for j , a skip of the next operation occurs if the *divide fault* exists. The skip would be made to a **JP** operation which provides remedial means of noting the error or of correcting it. Therefore, the operation which follows the **DIV** operation should have a j -operand of **SKIP** in order to preclude the **JP** operation whenever the divide sequence culminates in a correct answer. A divide fault can be detected also if the **DIV** operation is executed with a j of **NOOF**. In this case, a correct answer is indicated when a skip occurs. Since A is always positive at the time j is sensed, **ANEG** becomes meaningless.

Examples:



SQure Root Operation:

W V₀ V₁ V₂

➔ **SQRT** • [absent] • [absent] • [j] ➔

The **SQRT** operation finds $\sqrt{|Q|}$ and places it in Q. The remainder goes to A, always destroying the previous contents. The radix point of (Q) is assumed to be at the low order end of the register.

V₀ - always absent

V₁ - always absent

V₂ - specifies a skip-the-next-instruction condition

Operand, j	Condition
(blank)	Does not skip
SKIP	Always skip
REM	Skip if A ≠ 0
NOREM	Skip if A = 0

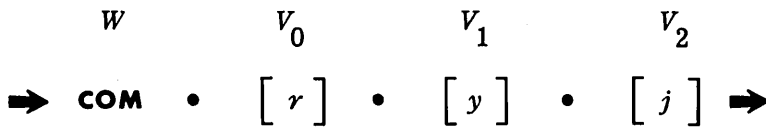
Examples:

➔ **SQRT** ➔

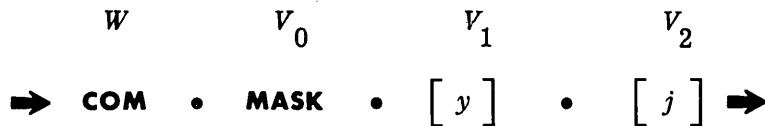
➔ **SQRT • NOREM** ➔

COM pare Operation:

Type A



Type B



Type A:

The **COM** operation compares the value expressed by y with r . A skip of the next operation takes place if the condition specified by j is satisfied. The content of r is not changed.

V_0 - designates the register with which the numeric value is compared

Register, r	Performance
A	A: y
Q	Q: y
AQ *	A: y and Q: y

V_1 - gives a Read-class operand that defines y

V_2 - specifies a skip condition; it *must* be present. The special meanings of j are:

Operand, j	Condition
YLESS	{ Skip if the value expressed by $y \leq Q$ Skip if the value expressed by $y \leq A$
YMORE	{ Skip if the value expressed by $y > Q$ Skip if the value expressed by $y > A$
YIN	{ Skip if $Q \geq$ value expressed by y and the value expressed by $y > A$. $Q \geq y > A$
YOUT	{ Skip if $Q <$ value expressed by y or the value expressed by $y \leq A$. $Q < y \leq A$

* Use only with j -operands **YIN** or **YOUT** y is compared with A and Q as individual 30 bit registers

Type B:

The **COM • MASK** operation compares A with the bit-by-bit product of the values expressed by y and Q. A skip of the next operation takes place if the condition specified by j is satisfied. The contents of A and Q are not changed.

V_0 - says **MASK**

V_1 - gives a Read-class operand that defines y

V_2 - specifies a normal j -operand; it must be present. The condition of A is tested after $LP(y)(Q)^*$ is subtracted from A. The $LP(y)(Q)^*$ is then added to A

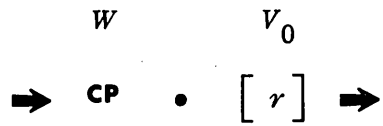
Examples:

⇒ **COM • AQ • W(TAB-2) • YIN** ⇒

⇒ **COM • MASK • L(TAB) • AZERO** ⇒

*LP(y) (Q) means the bit-by-bit product of y and Q

Complement Operation:



The **CP** operation complements all bits of the register specified by r .

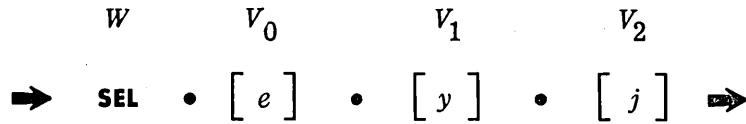
V_0 - designates the register which is complemented; r can be:

A or Q

Example:



SEL ective Operation:



The **SEL** operation performs logical manipulations specified by e on the content of A. A string of bits expressed by y controls these manipulations.

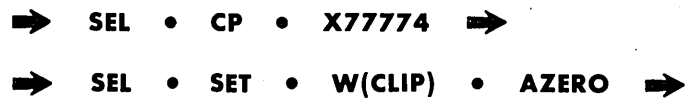
V_0 - states one of several logical functions. These are:

Expression, e	Performance
SET	Sets the individual bits of register A corresponding to <i>ones</i> in the numeric value expressed by y , leaving the remaining bits of A unaltered
CP	Complements the individual bits of register A corresponding to <i>ones</i> in the numeric value expressed by y , leaving the remaining bits of A unaltered
CL	Clears the individual bits of register A corresponding to <i>ones</i> in the numeric value expressed by y , leaving the remaining bits of A unaltered
SU	Replaces the bits of A with bits of the numeric value expressed by y corresponding to <i>ones</i> in Q

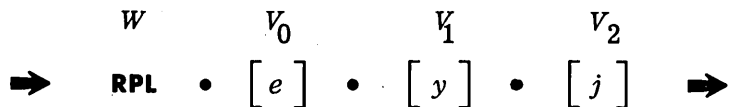
V_1 - gives a Read-class operand that defines y . A is not permitted

V_2 - specifies a normal j -operand; it is optional.

Examples:



RePLace Operation:



The **RPL** operation performs the function expressed by e , and stores the result in A and in a memory location established by y . The **Y** that appears in e refers to the numerical value which y defines.

V_0 - states a simple arithmetic or logical expression to be performed. These are:

Expression, e	Performance
(1) A+Y	$A + y \Rightarrow y$ and A
(2) A-Y	$A - y \Rightarrow y$ and A
(3) Y+Q	$y + Q \Rightarrow y$ and A
(4) Y-Q	$y - Q \Rightarrow y$ and A
(5) Y+1	$y + 1 \Rightarrow y$ and A
(6) Y-1	$y - 1 \Rightarrow y$ and A
(7) LP	LP (y) (Q)* $\Rightarrow y$ and A
(8) A+LP	$A + \text{LP } (y) (Q) \Rightarrow y$ and A
(9) A-LP	$A - \text{LP } (y) (Q) \Rightarrow y$ and A

V_1 - gives a Replace-class operand which defines address y

V_2 - specifies a normal j -operand; this is valid with all V_0 operands *except LP or*

V_2 - specifies the j -operand when V_0 is **LP**. In this case the operation permits all normal j -operands except **QPOS** and **QNEG**. Substituted for **QPOS** and **QNEG** are two special j -operands as follows:

EVEN - Even parity (even number of "ones" in A)

ODD - Odd parity (odd number of "ones" in A)

Examples:

$\Rightarrow \text{RPL} \cdot \text{A+LP} \cdot \text{W(CRUNCH)} \cdot \text{QNEG} \Rightarrow$

$\Rightarrow \text{RPL} \cdot \text{Y-Q} \cdot \text{UX (HOPTO+B6)} \Rightarrow$

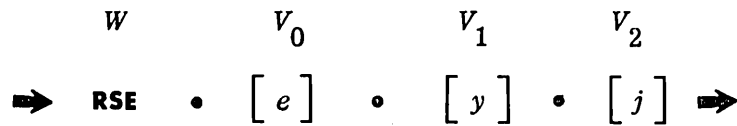
$\Rightarrow \text{RPL} \cdot \text{LP} \cdot \text{W (DOP+B4)} \cdot \text{ODD} \Rightarrow$

*LP (y) (Q) means the bit-by-bit product of (y) and (Q)

RPL

(f: 24, 25, 34, 35)
(36, 37, 44, 45, 46)

Replace SElective Operation:



The **RSE** operation performs logical manipulations specified by e on the content of A and then stores A in the memory location whose address is expressed by y . A string of bits in the same memory location controls these manipulations before the store takes place.

V_0 - states one of several logical functions. These are:

Expression, e	Performance
SET	Sets the individual bits of register A to <i>one</i> corresponding to <i>ones</i> in the numeric value expressed by y , leaving the remaining bits of A unaltered, then stores A at the storage address expressed by y
CP	Complements the individual bits of register A corresponding to <i>ones</i> in the numeric value expressed by y , leaving the remaining bits of A unaltered, then stores A at the storage address expressed by y
CL	Clears the individual bits of register A corresponding to <i>ones</i> in the numeric value expressed by y , leaving the remaining bits of A unaltered, then stores A at the storage address expressed by y
SU	Replaces the bits of A with bits of the numeric value expressed by y corresponding to <i>ones</i> in Q, then stores A at the storage address expressed by y

V_1 - gives a Replace-class operand that defines y

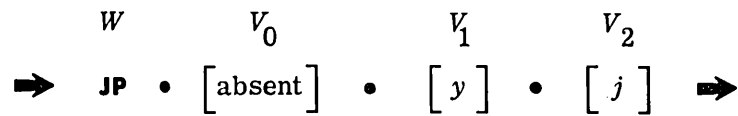
V_2 - specifies a normal j-operand; it is optional

Examples:

➡ RSE • SU • W(COVER+B4) ➡

➡ RSE • CL • LX(POW5) ➡

Jump Operation:



The **JP** operation clears the program address register **P**, and enters the address designated by *y* in **P** for certain conditions specified by *j*. Thus *y* becomes the address of the next operation and the beginning of a new program sequence. If a jump condition is not satisfied, the next sequential operation in the current sequence is executed in the normal manner.

- V_0 - always absent
- V_1^* - gives a Read-class operand which defines address *y*
- V_2 - specifies a jump condition

Operand <i>j</i>	Condition
QPOS	Jump if Q is positive
QNEG	Jump if Q is negative
AZERO	Jump if A is equal to zero
ANOT	Jump if A is not equal to zero
APOS	Jump if A is positive
ANEG	Jump if A is negative
(blank)	Unconditional jump
KEY1	Jump if Key 1 is set
KEY2	Jump if Key 2 is set
KEY3	Jump if Key 3 is set
STOP	Jump and then stop
STOP5	Jump and then stop if Key 5 is set
STOP6	Jump and then stop if Key 6 is set
STOP7	Jump and then stop if Key 7 is set
CⁿACTIVEIN	{ See next page for condition description
CⁿACTIVEOUT	

* If *j* is **CⁿACTIVEIN** or **CⁿACTIVEOUT**, an operand code of **X**, **LX**, **UX**, and **A** is not permitted

CⁿACTIVEIN *

Jump if the input buffer mode on channel n is active (n = 0, —, 17)

CⁿACTIVEOUT *

Jump if the output buffer on channel n is active (n = 0, —, 17)

Examples:

➔ JP • TRACE ➔

➔ JP • L(TRIG+B2) • KEY1 ➔

➔ JP • ROAR • C14ACTIVEIN ➔

* May be a *name* which is defined by a **MEANS** operation or a **CHAN-SET** tape

Jump Operation

W V₀ V₁ V₂ V₃

⇒ **JP** • [absent] • [location] • [channel] • [**COMACTIVE**] ⇒

This operation provides a means for determining whether an external function command buffer is active.

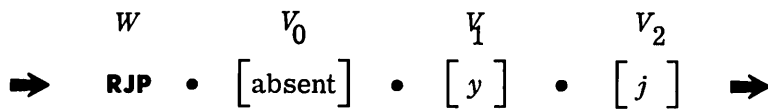
- V₀ - Always absent
- V₁ - Specifies the location to which control is to be transferred if the specified external function command buffer is active. This operand may contain only a tag or a tag with a K designator of L
- V₂ - Specifies the channel on which the external command buffer is to be tested. Channels C0-C7, C10-C17 are permitted. V₂ may specify a name which is identified by a **MEANS** operation or a **CHAN-SET** tape
- V₃ - Specifies that this test is for an active external function command buffer

Examples:

⇒ **JP** • **PTH** • **CIO** • **COMACTIVE** ⇒

⇒ **JP** • **PTH** • **TAPECHAN** • **COMACTIVE** ⇒

Return Jump Operation:



The **RJP** operation performs the following steps if conditions specified by *j* are satisfied: 1) it stores the content of the program address counter P, which is the address of the **RJP** operation plus one, into the lower 15 bits of the memory location which has the address specified by *y*, and 2) then it enters P with *y* + 1. Thus, *y* + 1 becomes the address of the next operation and the beginning of a new program sequence.

If the *j* condition is not satisfied, the next sequential operation in the current sequence is executed in the normal manner.

- V_0 - always absent
- V_1 - gives a Read-class operand which defines address *y*
- V_2 - specifies a jump condition

Operand, <i>j</i>	Condition
QPOS	Return jump if Q is positive
QNEG	Return jump if Q is negative
QZERO	Return jump if A is equal to zero
ANOT	Return jump if A is not equal to zero
APOS	Return jump if A is positive
ANEG	Return jump if A is negative
(blank)	Unconditional return jump
KEY1	Return jump if Key 1 is set
KEY2	Return jump if Key 2 is set
KEY3	Return jump if Key 3 is set
STOP	Return jump and then stop
STOP5	Return jump and then stop if Key 5 is set
STOP6	Return jump and then stop if Key 6 is set
STOP7	Return jump and then stop if Key 7 is set

Examples:



B **JumP** Operation:

W V_0 V_1

⇒ **BJP** • $[r]$ • $[y]$ **⇒**

The **BJP** operation tests the content of the B register specified by r . If (r) is zero, the normal sequence of operations continues. If (r) is non zero, (r) decreases by one, and a new sequence of operations begins at the address expressed by y .

V_0 - designates a B register: **B1** through **B7**

V_1 - gives a Read-class operand that defines y

Note: A j -operand is not permitted.

Examples:

⇒ **BJP** • **B5** • **DESK** **⇒**

⇒ **BJP** • **B1** • **U(EXIT+B2)** **⇒**

B SKip Operation:

W V_0 V_1

➔ **BSK** • [r] • [y] ➔

The **BSK** operation tests the content of the B register specified by r . If (r) is equal to the numeric value expressed by y , the control sequence skips the next operation and (r) is cleared. If (r) is not equal to the numeric value expressed by y , the normal sequence of operations continues, and (r) increases by one.

V_0 - designates a B register: **B1** through **B7**

V_1 - gives a Read-class operand that defines y

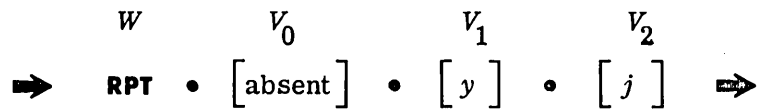
Note: A j -operand is not permitted

Examples:

➔ **BSK** • **B3** • **56** ➔

➔ **BSK** • **B4** • **B2** ➔

RePeaT Operation:



The **RPT** operation initiates a repeat mode of control which causes execution of the next sequential operation the number of times expressed by y , or until the j -operand condition of the next operation is satisfied, whichever occurs first. B^7 keeps count of the number of times execution is to take place. (B^7 decreases by one after each execution.)

- V_0 - always absent
- V_1 - gives a Read-class operand that defines y . If y is zero, the next instruction is skipped
- V_2 - specifies the mode of address modification of the repeated operation

Operand, j	Control
(blank)	Unmodified repeat of next operation
ADV	Advance the operand address of the repeated operation by one after each individual execution
BACK	Decrease the operand address of the repeated operation by one after each execution of the repeated operation
ADDB	Adds cumulatively the B register indicated in the repeated operation to its operand during each execution
R	Increase the operand address of the repeated Replace-class operation by the content of B^6 for the <i>store</i> portion of the replace only
ADVR	Increase the operand address of the repeated Replace-class operation by the content of B^6 for the <i>store</i> portion of the replace only; then increment the operand address of the repeated operation by one after each execution

Operand

Control

BACKR

Increase the operand address of the repeated Replace-class operation by the content of B⁶ for the *store* portion of the replace only; then decrement the operand address of the repeated operation by one after each execution

ADDBR

Adds cumulatively the B register indicated in the repeated Replace-class operation to its operand address during each execution; in addition to the above, increase the operand address of the repeated operation by the content of B⁶ only for *store* portion of the replace

Note: Use *j*-operands **R**, **ADVR**, **BACKR**, and **ADDBR** only when a **RPL** operation follows the **RPT** operation.

Examples:

➔ RPT • 39D ➔
➔ RPT • B7 • BACK ➔
➔ RPT • L(TRADE3) • ADDBR ➔

Note: Interrupts are locked out during the time the computer is in repeat mode.

INput Operation (With or Without Monitoring):

$$\Rightarrow \overset{W}{\text{IN}} \cdot \overset{V_0}{[\text{channel}]} \cdot \overset{V_1}{[y]} \cdot \overset{V_2}{[\text{absent or MONITOR}]} \Rightarrow$$

The **IN** operation establishes the control to transfer data from external equipment to the core memory via a specified channel. The address limits are defined by a numeric value expressed by y , which are transferred to memory address $00100+n$, where n is the number of the channel. Subsequent to this operation, but not as part of it, the individual buffer operations are executed at a rate determined by the external device. The starting address, initially established by this operation, is advanced by *one* following each individual buffer operation. The next current address is maintained throughout the buffer process in the lower order 15-bit positions of memory location with storage address $00100+n$. This mode continues until it is superseded by a subsequent initiation of an input buffer via the same channel, or until the higher order half and the lower order half of storage address $00100+n$ contain equal quantities, whichever occurs first. The first and last address of the memory area is specified in location $00100+n$.

V_0 - designates the Channel, C^n , through which buffering takes place:

C0, —, C17

V_1 - gives an operand that defines y . If V_1 is a number of five digits or less, or has an operand code of **L**, y replaces the lower half of address $00100+n$. If V_1 is a number of more than five digits, or has an operand code of **W**, y replaces the whole word of address $00100+n$. Operand codes of **X**, **U**, **LX**, **UX**, or **A**, are not permitted

V_2 - specifies whether the buffer operation is to be monitored or not. Monitoring is specified by V_2 being **MONITOR**. Otherwise V_2 is absent

A buffer operation is monitored if the main program is interrupted and control is transferred to $00040+n$ when the buffer operation is terminated by the control addresses in address $00100+n$ becoming equal

Examples:

$\Rightarrow \text{IN} \cdot \text{C5} \cdot 52367 \Rightarrow$

$\Rightarrow \text{IN} \cdot \text{C14} \cdot \text{W(LIMIT)} \cdot \text{MONITOR} \Rightarrow$

OUTput Operation (With or Without Monitoring):

$$\Rightarrow \text{OUT} \cdot \overset{W}{\text{channel}} \cdot \overset{V_0}{\text{channel}} \cdot \overset{V_1}{y} \cdot \overset{V_2}{\text{absent or MONITOR}} \Rightarrow$$

The **OUT** operation establishes the control to transfer data to external equipment from the core memory via a specified channel. The address limits are defined by a numeric value expressed by y ; these are transferred to memory address 00120+n, where n is the number of the channel. Subsequent to this operation, but not as part of it, the individual buffer operations are executed at a rate determined by the external device. The starting address, initially established by this operation, is advanced by *one* following each individual buffer operation. The next current address is maintained throughout the buffer process in the lower order 15-bit positions of memory location at storage address 00120+n. This mode continues until it is superseded by a subsequent initiation of an input buffer via the same channel, or until the higher order half and the lower order half of storage address 00120+n contain equal quantities, whichever occurs first. The first and last address of the memory area are specified in location 00120+n.

V_0 - designates the Channel, C^n , through which buffering takes place:

C0, —, C17

V_1 - gives an operand that defines y . If V_1 is a number of five digits or less, or has an operand code of **L**, y replaces the lower half of address 00120+n. If V_1 is a number of more than five digits, or has an operand code of **W**, y replaces the whole word of address 00120+n. Operand codes of **X**, **U**, **LX**, **UX**, or **A** are not permitted

V_2 - specifies whether the buffer operation is to be monitored or not. Monitoring is specified by V_2 being **MONITOR**. Otherwise V_2 is absent

A buffer operation is monitored if the main program is interrupted and control is transferred to 00060+n when the buffer operation is terminated by the control addresses in address 00120+n becoming equal.

Examples:

$$\Rightarrow \text{OUT} \cdot \text{C7} \cdot 41456 \Rightarrow$$

$$\Rightarrow \text{OUT} \cdot \text{C12} \cdot \text{L(LOC)} \cdot \text{MONITOR} \Rightarrow$$

EXternal-COMmand Operation:

W V₀ V₁ V₂

➔ **EX-COM** • [channel] • [external function code] • [sub-function code] ➔

The **EX-COM** operation initiates a one word external function buffer.

V₀ - Specifies the channel on which the external function code is transferred. Channels C0-C7, C10-C17 are permitted. V₀ may specify a name which is identified by a **MEANS** operation or a **CHAN-SET** tape

V₁ - Function code, this may be a ten digit number or less, or the whole contents of a memory location (i.e., operand code of w). Other operand codes are not permitted. B-Box modification is not allowed if V₁ is a constant

V₂ - Specifies the sub-function code

(absent) - The external function command is sent without force or monitor

FORCE - used when the communication is with external equipment which has not been designed to send an "external function request" to the computer. **MONITOR** may not be used in conjunction with an **EX-COM** with a V₂ operand of **FORCE**

MONITOR- Provides a transfer of control to location 00500+j when the buffer of the external function word is completed

MONFORCE - Provides the combined capabilities of **MONITOR** and **FORCE**

Examples:

➔ **EX-COM** • **CO** • **4300000016** • **FORCE** ➔

➔ **EX-COM** • **C17** • **W(EFUN)** ➔

➔ **EX-COM** • **TTY** • **CHAN** • **W(EFT)** • **MONITOR** ➔

➔ **EX-COM** • **SPILL** • **W(EFF)** • **MONFORCE** ➔

EXternal-COMmand-Multi Word Operation

W V₀ V₁ V₂

➔ **EX-COM-MW** • [channel] • [y] • [sub-function code] ➔

The **EX-COM-MW** operation sets up the appropriate external function buffer control word (at 00140+j) and initiates output buffering of the specified external function commands.

- V₀ - Specifies the channel on which the external function codes are sent. Channels C0-C7, C10-C17 are permitted. V₀ may specify a name which is identified by a **MEANS** operation or a **CHAN-SET** tape
- V₁ - Gives the buffer limits of the function codes to be transmitted. This may be the contents of a whole memory location only (i.e., operand code of w). Other operand codes are not permitted. B register modification is not allowed if V₁ is a constant
- V₂ - Specifies whether the buffering of the external function command words is to be monitored. When monitored, the completion of the buffer will cause transfer of control to external function buffer monitor interrupt entrance address 00500+j

Examples:

➔ **EX-COM-MW** • C3 • W(FCCW) • MONITOR ➔

➔ **EX-COM-MW** • TAPECHAN • W(SFCC) ➔

TERMinate Buffer Operation:

W V₀ V₁

➔ **TERM** • [channel number or **ALL**] • [buffer mode] ➔

The **TERM** function terminates input, output, external function command, or all buffers as specified by the V₀ and V₁ operands.

V₀ - Specifies the channel on which buffering is to be terminated. Channels C0-C7, C10-C17 are permitted. V₀ may specify a name which is identified by a **MEANS** operation or a **CHAN-SET** tape

ALL - causes all buffering including that of external function commands, input data, and output data to be halted. No V₁ operand is allowed when the V₀ operand is **ALL**.

V₁ - Specifies the mode of buffering to be terminated.

(absent) - The V₁ operand must be omitted if the V₀ operand was **ALL**.

COM - Terminates the buffering of external function commands on specified channel

INPUT - Terminates the buffering of input data on specified channel

OUTPUT - Terminates the buffering of output data on specified channel

Examples:

➔ **TERM • C6 • COM** ➔
➔ **TERM • ALL** ➔
➔ **TERM • C17 • OUTPUT** ➔

Examples: (illegal)

➔ **TERM • ALL • INPUT** ➔

Set Interrupt Lockout Operation:

W V₀
➔ **SIL • ALL** ➔

The operator **SIL** locks out both internal and external interrupts on all channels.

V₀ - The only V₀ operand allowed is **ALL**

Examples:

➔ **SIL • ALL** ➔

Example (illegal):

➔ **SIL • C6** ➔

Set Interrupt Lockout - **EX**ternal Operation

W V₀

➔ **SIL-EX** • [channel] ➔

The operator **SIL-EX*** sets external interrupt lockout for the specified channel.

V₀ - Specifies the channel on which external interrupts are to be locked out. Channels C0-C7, C10-C17 are permitted. V₀ may specify a name which is identified by a **MEANS** operator or a **CHAN-SET** tape

ALL - locks out external interrupts on all channels

Examples:

➔ **SIL-EX** • **CIO** ➔

➔ **SIL-EX** • **ALL** ➔

➔ **SIL-EX** • **FLEXCHAN** ➔

* The interrupts locked out by **SIL-EX**, can be released only by the **RIL-EX** operation

Remover Interrupt Lockout Operation

W V₀

➔ **RIL** • [absent or **ALL**] ➔

The operator **RIL** removes interrupt lockouts on all internal channels, and all external channels not previously locked out by **SIL-EX** operations.

V₀ - The effect on the computer is the same whether V₀ is **ALL** or absent

(absent) - If V₀ is absent an instruction of the type 600XX XXXXX will be generated

ALL - If V₀ is **ALL** an instruction of the type 66X10 XXXXX will be generated

Examples:

➔ **RIL** ➔

➔ **RIL** • **ALL** ➔

Remove Interrupt Lockout - EXternal Operation

W V₀

➔ **RIL-EX** * • [channel] ➔

The operator **RIL-EX** * releases the interrupt lockout for external interrupts.

V₀ - Specifies the channel on which the external interrupt lockout is to be released. Channels C0-C7, C10-C17 are permitted. V₀ may specify a name which is identified by a **MEANS** operator or a **CHAN-SET** tape

ALL - Removes the external interrupt lockout on all channels

Examples:

➔ **RIL-EX** • **CIO** ➔
➔ **RIL-EX** • **ALL** ➔
➔ **RIL-EX** • **TTYCHAN**

*This instruction must be used to remove interrupt lockouts on channels previously locked out by **SIL-EX** operations. **RIL** operations only release lockouts for external interrupts not locked out by **SIL-EX**, and of course internal interrupts

Remove Interrupt Lockout and Jump Operation:

\rightarrow $\overset{W}{\text{RILJP}}$ \cdot $\overset{V_0}{[y]}$ \rightarrow

The **RILJP** operation removes the interrupt lockout, thus allowing a subsequent interrupt, and jumps to address y unconditionally. The operation generates a 601nn nnnnn instruction.

V_0 - gives a Read-class operand which defines address y

POLY-OPERATIONS

<u>Operation</u>	<u>Page</u>
ENTRY	2
EXIT	3
CLEAR	5
PUT	6
MOVE	8
INCREMENT	10
TYPEC	12
TYPET	15
PUNCHC	17
PUNCHT	20
TYPE-DECimal.	22
PUNCH-DECimal	24
Upper TAG	26

POLY-OPERATIONS

Quite frequently, groups of contiguous instructions appear iteratively in a program. These instructions are alike because they perform a specific job or function. It is possible in cases such as this to generate the successive instructions with a single AS-1 operation. This is the familiar *one-to-many* relationship between instructions which shall be herein termed *poly-coding*, with the parent instruction being called a *poly-operation*. A poly-operation is capable of generating within the assembler system a unique sequence of computer instructions (in some cases a single instruction) designed to perform the specific task required.

It is permissible during the coding of a routine to intermix mono- and poly-operations in any order desired. However, the programmer must not attempt to skip a poly-operation with the *j*-operand of a mono-operation. The poly-operation usually results in the generation of more than one instruction in the assembled object program; the computer skips the first of these instead of the intended next mnemonic operation in the source program. The assembler-generated computer instructions appear in the object program in the order specified by the AS-1 coding.

Poly-operations are capable of producing assembler-generated, unique labels and tags for internal use during assembling.

The computer frequently employs registers A, Q, and B⁷ in object program instructions resulting from poly-operations. In so doing, it destroys any previous information contained in these registers. The programmer should therefore exercise caution in the use of these registers in statements preceding poly-operations. In cases where their use is necessary and the content of any of these registers is required later, the programmer must save their content by transfer to a temporary storage location for later reference.

ENTRY Operation:

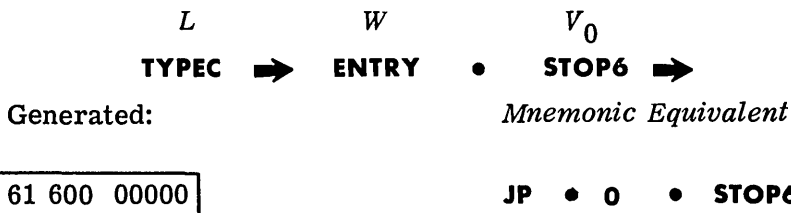


The **ENTRY** operation establishes a standard means of starting all subroutines. It produces either a normal entry with no jump conditions or a jump capability with Key Stop options. This operation is the first one in each subroutine; because of this it must have a label which gives the subroutine name. Although each **ENTRY** operation generates only one instruction, the variations which the instruction can assume make it a poly-operation.

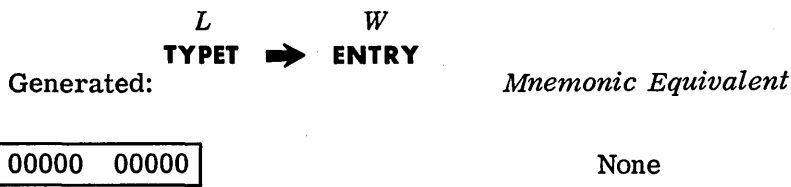
V_0 - names the key which must be set on the computer console if the programmer wishes the computer to stop on exiting from the subroutine. If operand V_0 is absent (no key stop specified), the assembler generates a word of 0's for the first subroutine word in the object program. If V_0 is present, the assembler generates a $61j00\ 00000$, where j is determined by the V_0 operand. The allowable entries for V_0 are:

- STOP5** ; $j = 5$
- STOP6** ; $j = 6$
- STOP7** ; $j = 7$

Example a:



Example b:



EXIT Operation:



The **EXIT** operation provides a means of exiting normally from a subroutine, i.e., it generates a jump back to the **ENTRY** operation of the subroutine and thence to the main routine. The **EXIT** operation is used at every place in the subroutine where an exit from it is desired; hence, any number of exits is permitted. The label is optional.

Although the assembler generates only one instruction per **EXIT** operation, the generated instruction can assume a variety of formats. The format depends on 1) whether the V_0 operand of the foregoing **ENTRY** of the subroutine is present or absent, and 2) the **EXIT** V_0 operand itself. Because of these variations this operation is classed as a poly-operation. If the V_0 operand of the preceding **ENTRY** operation is absent, the assembler generates a $61j\underline{10}$ nnnnn or a $60j\underline{10}$ nnnnn. If the V_0 operand of the preceding **ENTRY** operation is present, the assembler generates either $61j\underline{00}$ nnnnn or $60j\underline{00}$ nnnnn. The address assigned to the preceding **ENTRY** position is nnnnn. The compiler looks for this address, then inserts it in the tag position, nnnnn, of the **EXIT** operation.

V_0 - determines j in the instruction generated by the **EXIT** operations as follows:

If the **EXIT** V_0 is:

The generated instruction is:

		j	k	b	y
Absent	61	0	(1 or 0)*	0	nnnnn
QPOS	60	2	↓	↓	↓
QNEG	60	3	↓	↓	↓
AZERO	60	4	↓	↓	↓
ANOT	60	5	↓	↓	↓
APOS	60	6	↓	↓	↓
ANEG	60	7	↓	↓	↓

*If V_0 of previous **ENTRY** is absent, $k = 1$

If V_0 of previous **ENTRY** is present, $k = 0$

If the **EXIT** V_0 is:

KEY1
KEY2
KEY3
STOP
STOP5
STOP6
STOP7

The generated instruction is:

		j	k	b	y
	61	1	(1 or 0)*	0	nnnnn
	61	2	↓	↓	↓
	61	3			
	61	4			
	61	5			
	61	6			
	61	7			

*If V_0 of previous **ENTRY** is absent, $k = 1$

If V_0 of previous **ENTRY** is present, $k = 0$

Examples:

a. Previous **ENTRY** V_0 absent

	L	W	V_0
1)	MAP2	EXIT	KEY3

Generated:

Mnemonic Equivalent

61310 nnnnn

JP • L(nnnnn) • KEY3

2)	EXIT
----	-------------

Generated:

Mnemonic Equivalent

61010 nnnnn

JP • L(nnnnn)

b. Previous **ENTRY** V_0 present

	W	V_0
1)	EXIT	ANOT

Generated:

Mnemonic Equivalent

60500 nnnnn

JP • nnnnn • ANOT

	L	W	V_0
2)	MAP3	EXIT	QNEG

Generated:

Mnemonic Equivalent

60300 nnnnn

JP • nnnnn • QNEG

CLEAR Operation:

\xrightarrow{W} V_0 V_1
→ CLEAR • [number of words] • [starting address] →

The **CLEAR** operation clears (fills with 0's) a number of words of an area of core memory.

V_0 - specifies the number of words to be cleared. This is a Read-class operand; however, the operand code **A** is not permitted. If a value of 0 is used, the operation is a "do nothing" instruction which causes a delay of the computer. If a 1 is specified, the end result is the same as that of a mono-code CL operation

V_1 - gives the starting address of the area to be cleared. This may be a constant of maximum five digits, a tag, a tag with an increment, or a tag with an increment and a B-register designation

Example:

→ CLEAR • 6 • CAT+B6-2 →

Generated:

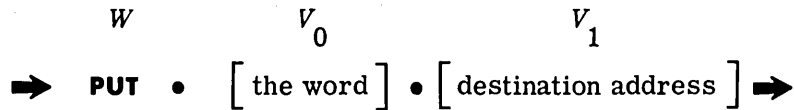
Mnemonic Equivalent

70100	00006
16036	nnnnn*

RPT • 6 • ADV
STR • B0 • W(CAT+B6-2)

 *nnnnn = constant specified by **CAT -2**

PUT Operation:



The **PUT** operation places a single word or half word in a designated storage address.

V_0 - expresses a Read-class operand; it maybe a tag, a constant, or the content of an address. This represents the source information

V_1 - specifies the address in memory at which the word or half word is to be stored. This is a Store-class operand; it gives a constant, a B register, a tag, a tag with increment, or a tag with increment and B-register designation preceded by an appropriate operand code. A and Q are not permitted.

Since register Q is used for the movement of the word, its original content is destroyed by this operation. The programmer must provide for preservation of the initial contents if desired

Example a:



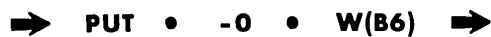
Generated:

Mnemonic Equivalent

10016	nnnnn*
14023	nnnnn

ENT • **Q** • **L(CAT+B6)**
STR • **Q** • **U(DOG+B3-2)**

Example b:



Generated:

Mnemonic Equivalent

10040	77777
14036	00000

ENT • **Q** • **-0**
STR • **Q** • **W(B6)**

*nnnnn is an allocated address corresponding to a tag

Example c:

➡ PUT • 77342106 • W(DOG) ➡

Generated:

10030	nnnnn*
14030	nnnnn

ENT • Q • W(A||||nnnn)*

STR • Q • W(DOG)

*A |||| nnnn is an assembler-generated tag

MOVE Operation:

W V_0 V_1 V_2
→ MOVE • [number of words] • [from address] • [to address] →

The **MOVE** operation moves masses of data from one area to another. The computer moves the words of information sequentially through the Q register and may use B^7 for indexing. It does not reinstate the original content to either the B^7 or the Q register; the programmer must save and restore such information if he wishes to retain it.

- V_0 - specifies the number of words to be moved; the programmer inserts the Read-class operand to indicate the number of words to be transferred; however, the operand codes **X**, **LX**, **UX**, or **A** are not permitted
- V_1 - indicates the initial address of the area from which data will be moved; it can be an absolute address, a B register, a tag, or a tag with an increment and/or a B-register designation
- V_2 - states the initial address of the area to which data will be transferred; it can be an absolute address, a B register, a tag, or a tag with an increment and/or a B-register designation.

The assembler generates instructions in numbers varying from 2 to 10, depending upon 1) the number of words to be moved, 2) whether the V_0 operand is mnemonic or not, and 3) whether B^n designations appear in V_1 and/or V_2 . If only one word is moved, the minimum number of instructions generated is two; if V_0 is mnemonic, the minimum is five instructions. Since the use of B-register designations in operands V_1 and V_2 changes the number of instructions generated by the assembler, two examples are given below. The first shows an operation with no B^n in either operand V_1 or V_2 ; the second contains a B^n in both operands

Example a:

→ MOVE • 4 • CAT • DOG →

Generated:

Mnemonic Equivalent

12700	00003
10037	nnnnn
14037	nnnnn
$[\alpha]$	$[\alpha - 2]$
72700	$[\alpha - 2]$

ENT • B7 • 3
ENT • Q • W(CAT+B7)
STR • Q • W(DOG+B7)
 α BJP • B7 • $\alpha - 2$

Example b:

➔ **MOVE • B5 • CAT+B4 • DOG+B7-3** ➔

Generated:

Mnemonic Equivalent

10004	nnnnn
14010	[$\alpha - 2$]
10007	nnnnn
14010	[$\alpha - 1$]
12705	00000
72700	[$\alpha - 2$]
61000	[$\alpha + 1$]
10037	[000000]
14037	[000000]
[α] 72700	[$\alpha - 2$]

ENT • Q • CAT+B4
STR • Q • L($\alpha - 2$)
ENT • Q • DOG+B7-3
STR • Q • L($\alpha - 1$)
ENT • B7 • B5
BJP • B7 • $\alpha - 2$

JP • $\alpha + 1$

ENT • Q • W(0+B7)
STR • Q • W(0+B7)

 α BJP • B7 • $\alpha - 2$

INCREMENT Operation:

$$\xrightarrow{W} \text{INCREMENT} \cdot \overset{V_0}{[\text{B register}]} \cdot \overset{V_1}{[\text{increment}]} \xrightarrow{\quad}$$

The **INCREMENT** operation provides a means to either increase the number contained in a B register (B^n) by a fixed increment or decrease the number in B^n by a fixed decrement.

V_0 - specifies the B register to be incremented

V_1 - states the value of the increment by which the content of the B register is to be altered. The increment is defined by a Read-class operand

Example a:

$\xrightarrow{\quad} \text{INCREMENT} \cdot \text{B2} \cdot -1 \xrightarrow{\quad}$

Generated:

Mnemonic Equivalent

$\begin{array}{l} [\alpha] \quad 72 \quad 200 \quad [\alpha + 1] \\ [\alpha + 1] \text{ Next Instruction} \end{array}$
--

$\alpha \quad \text{BJP} \cdot \text{B2} \cdot \alpha + 1$
 $\alpha + 1 \quad \text{Next Instruction}$

Example b:

$\xrightarrow{\quad} \text{INCREMENT} \cdot \text{B5} \cdot 32D \xrightarrow{\quad}$

Generated:

Mnemonic Equivalent

$12 \quad 505 \quad 00040$

$\text{ENT} \cdot \text{B5} \cdot \text{B5} + 32D$

Example c:

$\xrightarrow{\quad} \text{INCREMENT} \cdot \text{B3} \cdot -12 \xrightarrow{\quad}$

Generated:

Mnemonic Equivalent

$\begin{array}{l} 11 \quad 003 \quad 00000 \\ 20 \quad 040 \quad 77765 \end{array}$

$\text{ENT} \cdot \text{A} \cdot \text{B3}$
 $\text{ADD} \cdot \text{A} \cdot \text{X}(77765)$
 $\text{ENT} \cdot \text{B3} \cdot \text{A}$

Example d:

⇒ INCREMENT ◦ B4 ◦ L(CAT+6+B2) ⇒

Generated :

Mnemonic Equivalent

11 004 00000
20 052 mnnnn*
12 470 00000

ENT ◦ A ◦ B4

ADD ◦ A ◦ LX(CAT+6+B2)

ENT ◦ B4 ◦ A

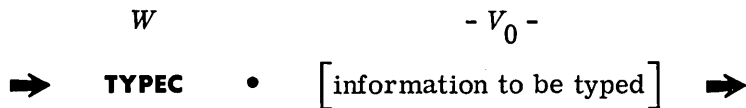
This poly-operation generates a variable number of object language instructions depending on the nature of the V_1 operand. A positive constant in V_1 causes a single instruction to be generated, a negative constant causes two instructions, and a symbolic name results in three instructions.

A special case occurs when the V_1 value is: -1. A B register can be decremented by one to reach zero, but not through zero; i.e., a B register containing zero, if decremented by one, remains zero.

The programmer should note that the A register is used in some cases and is not restored. If he wishes to preserve the previous content of register A for later use, he must provide for its storage in another location.

*nnnnn: The value allocated to the tag **CAT+6** by the assembler

TYPEC Operation:



The **TYPEC** operation causes the content (in octal) of A, Q, any B register, or any storage location to be typed by the on-line monitoring typewriter. In addition to specifying that the numerical information in any of the above registers be typed, the programmer may issue special commands to the typewriter. These commands, used as operands in the special format described below, may cause the typewriter to do the following three things:

Operand	Performance
• CR •	Causes the typewriter to do a carriage return
• SP •	Causes the typewriter to skip a space
• TAB •	Causes the typewriter to move to the next tabulation stop

By properly inserting these commands as operands between the operands denoting the information to be typed, the programmer can control the format (spacing and lines) of the information typed. The vertical bars are the special control symbols for indicating that the operand is an order directing the typewriter. Each of these three special operands *must* begin with and end with a vertical bar, and *must* each be separated by point separators from other operands.

$-V_0-$ - specifies the operands in the operand position in the order in which they are to be read and/or executed. These operands are of four types: *p1*, *p2*, *p3*, and *p4*. They may appear in any order, depending on the programmer's desires or needs. Point separators must separate each operand.

p1 - gives the locator of a value to be typed; it consists of an operand code of **L**, **U**, or **W** together with a normal Read operand in parentheses. The parentheses may contain a tag, B-register designation, or increment, or any combination of these

p2 - names a tag or label, without operand code, which the typewriter will type

p3 - specifies a constant, of five digits or less, to be typed

Exception: The value, zero, will not be typed if expressed as an operand. Zeros may be obtained by using the **TYPET** operation.

p4 - states a special typewriter command symbol. Valid symbols are |CR|, |SP|, and |TAB|; these command symbols cause the typewriter to perform a carriage return, to skip a space, and to move to a tabulator stop respectively

Example:

- V₀ -

L W

FIRST ⇒ **TYPEC** • **U(BETA+B3-6)** • **2576** • |CR| • **A** • |SP| • **Q** • **BETA** ⇒

p1 *p3* *p4* *p1* *p4* *p1* *p2*

LAST ⇒ **STR** • **Q** • **W (GAMMA)** ⇒

The **FIRST** operation above causes the following equivalent instructions and codes to be generated (except for the **LAST** operation):

FIRST ⇒ **RJP** • **TYPEC** *D*

 ⇒ **00023** • **BETA - 6** *D*

 ⇒ **00000** • **02576** *D*

 ⇒ **77450** • **00000** *D*

 ⇒ **00070** • **00000** *D*

 ⇒ **77040** • **00000** *D*

 ⇒ **00000** • **00000** *D*

 ⇒ **00000** • **BETA** *D*

LAST ⇒ **STR** • **Q** • **W(GAMMA)** ⇒

The **TYPEC** subroutine checks the first two characters of each of the operations following the Return Jump to **TYPEC** subroutine. If these are 00, it replaces them with 10 or 20; if they are 77, it interprets the characters following as commands to the typewriter (type *p4* operands).

The operation labeled **LAST** is not a part of the **TYPEC** performance. It illustrates that the programmer must follow the **TYPEC** operation with an

operation which will not cause the generation of a word of 0's in the object program. In other words, the next instruction in the object program must have a legitimate computer instruction code.

The assembler object program uses the **TYPEC** subroutine to produce the typeout. In general this poly-operation generates a Return Jump to the **TYPEC** subroutine, followed by an operation statement for each operand, directing the computer either to type the information as specified or to perform the command given. The **TYPEC** subroutine stores the contents of the registers it uses and restores them upon completion of the typeout.

TYPET Operation:

W $-V_0-$

➔ **TYPET** • [text and typewriter commands] ➔

The **TYPET** operation generates a section of object language program which, when run on the computer, causes the on-line typewriter to type the message given by the $-V_0-$ operand of the **TYPET** operation. No point separators appear between the parts of the V_0 operand. The commands to the typewriter, viz., carriage return, space, and tab, intersperse with the text according to the needs and desires of the programmer. These typewriter commands separate from the text by means of a vertical bar, |, before and after each command:

• text |**SP**| text |**CR**| text |**TAB**| text ➔

Exception: Where a space is desired between characters of the typed text, a space code symbol, Δ , may substitute for the command, |**SP**|.* The above example would then be:

• text Δ text |**CR**| text |**TAB**| text ➔

The programmer can use either symbol for a space. Where a tab follows a carriage return, the format should be |**CR**| |**TAB**| :

• text Δ text |**CR**| |**TAB**| text ➔

- V_0 - specifies the text to be typed, interspersed with the typewriter commands needed to produce the text format desired by the programmer. The typewriter commands and their symbols are:

Carriage Return:	CR
Tab:	TAB
Space:	SP or Δ

*Only four consecutive space codes permitted ($\Delta\Delta\Delta\Delta$)

Example:

FIRST ➡ TYPET • ABC | CR | DE | TAB | ➡

➡ TYPET • FGH | CR ||TAB | I Δ J ➡

produces the object language program:

```
FIRST ➡ RJP • TYPET
        65000 TYPET

        ↑ A  B  C )
        47302  31645

        D E → STOP
        22205  17700

        RJP • TYPET
        65000 TYPET

        ↑ F  G  H )
        47261  30545

        → I  Δ  J STOP
        51140  43277
```

During the running of the object program, the TYPET subroutine then uses the above object language program to produce the typewriter printout.

Any number of space commands can precede or follow the | CR | and | TAB | commands without affecting the text. Putting more than one space command between parts of the text has the effect of spreading these parts of the text farther apart on the typewritten page.

There is no provision for controlling the case of the characters in the output message. Alphabetical information is typed in upper case, numerical information in lower case. The TYPET subroutine, which unpacks the codes taken from the object language program, recognizes the end of the message by detecting the code, 77.

PUNCHC Operation:

W - V₀ -

➔ **PUNCHC** • [parameters for information and/or typewriter commands] ➔

The **PUNCHC** operation causes the content (in octal) of A, Q, any B register, or any storage location to be punched by the High-Speed Punch. In addition to directing that the numeric information in any of the above registers be punched, the programmer may write three special command symbols. These three symbols are typewriter commands which, when the punched paper tape is on a typewriter, will direct the typewriter to perform certain carriage operations. These operations control the format of the typewriter typeout; they include:

Operand	Performance
• CR •	Causes the typewriter to do a carriage return.
• SP •	Causes the typewriter to skip a space.
• TAB •	Causes the typewriter to move to the next tabulation stop.

By properly inserting these commands as operands between the operands denoting the information to be typed, the programmer can control the format (spacing and lines) of the information typed. The vertical bars are the special control symbols for indicating that the operand is an order directing the typewriter. Each of these three special operands *must* begin with and end with a vertical bar, and each *must* be separated by point separators from other operands.

- V₀ - specifies the operands in the operand position in the order in which they are to be read and/or executed. These operands are of four types: *p1*, *p2*, *p3*, and *p4*. They may appear in any order, depending on the programmer's desires or needs. Point separators must separate each operand.

p1 - gives the locator address of a value to be typed; it consists of a normal Read-class operand

p2 - gives a tag or label allocation value which the typewriter will type. This operand has no operand code

p3 - specifies a constant, of five digits or less, to be typed.

Exception: The value, zero, will not be typed if expressed as an operand.

Zeros may be obtained by using the **PUNCHT** operation

p4 - states a special typewriter command symbol. Valid symbols are |**CR**|, |**SP**|, and |**TAB**|; these command symbols cause the typewriter to perform a carriage return, to skip a space, and to move to a tabulator stop respectively

Example:

<i>L</i>	<i>W</i>	- V_0 -
FIRST	⇒ PUNCHC • <i>Q</i> • CR • L(ALPHA+B3) • TAB • 50 • SP • INST 4	⇒
	<i>p1</i> <i>p4</i> <i>p1</i> <i>p4</i> <i>p3</i> <i>p4</i> <i>p2</i>	
NEXT	⇒ ENT • A • U (GAMMA)	⇒

The **FIRST** operation above causes the following equivalent (in some cases, incomplete) operations to be generated:

FIRST	⇒ RJP	•	PUNCHC	↘
	⇒ 00000	•	00000	↘
	⇒ 77450	•	00000	↘
	⇒ 00013	•	ALPHA	↘
	⇒ 77510	•	00000	↘
	⇒ 00000	•	00050	↘
	⇒ 77040	•	00000	↘
	⇒ 00000	•	INST4	↘

The **PUNCHC** subroutine checks the first two characters of each of the operations following the Return Jump to **PUNCHC** subroutine. If these are 00, it replaces them with 10; if they are 77, it interprets the characters following as commands to the typewriter (type *p4* operands).

The operation labeled **NEXT** is not a part of the **PUNCHC** performance. It illustrates that the programmer must follow the **PUNCHC** operation with an operation which will not cause

the generation of a word of 0's in the object program. In other words, the next instruction in the object program must have a legitimate computer instruction code.

The assembler object program uses the PUNCHC subroutine to produce the typeout. In general this poly-operation generates a Return Jump to the PUNCHC subroutine, followed by an operation statement for each operand, directing the computer either to type the information as specified or to perform the command given.

PUNCHT Operation:

W $-V_0-$

➔ PUNCHT • [text and/or typewriter commands] ➔

The **PUNCHT** operation causes the High-Speed Punch to punch the text(s) which the programmer has written in the $-V_0-$ operand position of the **PUNCHT** operation. It also punches the codes for **SP**, **CR**, and **TAB**, which control the typewriter carriage movements during a listing of the punched tape. The programmer controls the format of the typewriter listing by interspersing the carriage control symbols between his texts as he desires. No point separators appear between the parts of the $-V_0-$ operand. Each carriage control symbol must have a vertical bar, |, before and after it.

Exception: Where a space is desired between characters of the typed text, a space code symbol, Δ , may substitute for the command, |SP|.

Example:

a. ➔ PUNCHT • text |SP| text |CR| text |TAB| text ➔

b. ➔ PUNCHT • text Δ text |CR| text |TAB| text ➔

Where 2 carriage control symbols appear consecutively, each one must have vertical bars before and after it.

Example:

➔ PUNCHT • text |CR| |TAB| text ➔

$-V_0-$ specifies the text to be typed, interspersed with the typewriter commands needed to produce the format desired by the programmer.

If the text is too long to put into one L_0 **PUNCHT** operation, successive operations can be written. Labels on these operations are optional

Example:

FIRST ➔ PUNCHT • PAY Δ TAX |CR| ON |TAB| ➔

➔ PUNCHT • OCT |SP| 15 |CR| ➔

The operations and codes generated in the running program by the above poly-operations are:

```
FIRST → RJP • PUNCHT
        65000 PUNCHT

        ↑ P A Y Δ
        47153 02504

        T A X λ 0
        01302 74503

        N → STOP
        06517 70000

        RJP • PUNCHT
        65000 PUNCHT

        O C T Δ ↓
        03160 10457

        1 5 λ STOP
        52624 57700
```

When the running program is subsequently performed, the PUNCHT subroutine then causes the High-Speed Punch to punch out octal codes above.

Any number of space commands can appear consecutively anywhere in the text. The effect is to vary the spacing between parts of the texts on the hard copy.

There is no provision for controlling the case of the characters in the output message. Alphabetic information appears in upper case, numeric in lower case. The PUNCHT subroutine, which translates sequentially the codes taken from the object language program, recognizes the end of the message by detecting the code, 77.

TYPE-DECimal Operation:

W
- V₀ -

➔ **TYPE-DEC** • [information to be typed] ➔

The **TYPE-DEC** operation causes the content (in decimal) of A, Q, any B register, or any storage location to be typed by the on-line monitoring typewriter. In addition to specifying that the numerical information in any of the above registers be typed, the programmer may issue special commands to the typewriter. These commands, used as operands in the special format described below, may cause the typewriter to do the following three things:

Operand	Performance
• CR •	Causes the typewriter to do a carriage return
• SP •	Causes the typewriter to skip a space
• TAB •	Causes the typewriter to move to the next tabulation stop

By properly inserting these commands as operands between the operands denoting the information to be typed, the programmer can control the format (spacing and lines) of the information typed. The vertical bars are the special control symbols for indicating that the operand is an order directing the typewriter. Each of these three special operands must begin with and end with a vertical bar, and must each be separated by point separators from other operands.

- V₀ - specifies the operands in the operand position in the order in which they are to be read and/or executed. These operands are of four types: *p1*, *p2*, *p3*, and *p4*. They may appear in any order, depending on the programmer's desires or needs. Point separators must separate each operand.

p1 - gives the locator of a value to be typed; it consists of an operand code together with a normal Read operand in parentheses. The parentheses may contain a tag, B-register designation, or increment, or any combination of these

p2 - names a tag or label, without operand code, which the typewriter will type

p3 - specifies a constant, of five digits or less, to be typed

Exception: The value, zero, will not be typed if expressed as an operand.
 Zeros may be obtained by using the **TYPET** operation.

$p4$ -states a special typewriter command symbol. Valid symbols are **|CR|**, **|SP|**, and **|TAB|**; these command symbols cause the typewriter to perform a carriage return, to skip a space, and to move to a tabulator stop respectively

Example:

<i>L</i>	<i>W</i>	<i>-V₀-</i>
FIRST ➔	TYPE-DEC •	U(BETA+B3-6) •
	<i>p1</i>	<i>p3</i>
	2576 •	 CR •
	<i>p4</i>	<i>p1</i>
	A •	 SP •
	<i>p1</i>	<i>p4</i>
	Q •	BETA ➔
	<i>p1</i>	<i>p2</i>

PUNCH-DECimal Operation:

W - *V*₀ -

➔ **PUNCH-DEC** • [parameters for information and/or typewriter commands] ➔

The **PUNCH-DEC** operation causes the content (in decimal) of A, Q, any B register, or any storage location to be punched by the High-Speed Punch. In addition to directing that the numeric information in any of the above registers be punched, the programmer may write three special command symbols. These three symbols are typewriter commands which, when the punched paper tape is on a typewriter and typed out, will direct the typewriter to perform certain carriage operations. These operations control the format of the typewriter typeout; They include:

Operand

- | | |
|--------------------|--|
| • CR • | Causes the typewriter to do a carriage return. |
| • SP • | Causes the typewriter to skip a space. |
| • TAB • | Causes the typewriter to move to the next tabulation stop. |

By properly inserting these commands as operands between the operands denoting the information to be typed, the programmer can control the format (spacing and lines) of the information typed. The vertical bars are the special control symbols for indicating that the operand is an order directing the typewriter. Each of these three special operands must begin with and end with a vertical bar, and each must be separated by point separators from other operands.

- *V*₀ - specifies the operands in the operand position in the order in which they are to be read and/or executed. These operands are of four types: *p*1, *p*2, *p*3, and *p*4. They may appear in any order, depending on the programmer's desires or needs. Point separators must separate each operand

*p*1 - gives the locator address of a value to be typed; it consists of a normal Read-class operand

*p*2 - gives a tag or label allocation value which the typewriter will type. This operand has no operand code

p3 - specifies a constant, of five digits or less, to be typed

Exception: The value, zero, will not be typed if expressed as in operand.
Zeros may be obtained by using the **PUNCHT** operation.

p4 - states a special typewriter command symbol. Valid symbols are |**CR**|, |**SP**|, and |**TAB**|; these command symbols cause the typewriter to perform a carriage return, to skip a space, and to move to a tabulator stop respectively

Example

<i>L</i>	<i>W</i>	- V ₀ -																	
FIRST	➔	PUNCH-DEC	•	Q	•	CR	•	L(ALPHA+B3)	•	TAB	•	50	•	SP	•	INST	4	➔	
				<i>p</i> 1		<i>p</i> 4		<i>p</i> 1		<i>p</i> 4		<i>p</i> 3	<i>p</i> 4		<i>p</i> 2				

Upper-TAG Operation:

$$\begin{array}{ccc} W & & V_0 & & & & V_1 \\ \Rightarrow & \mathbf{U-TAG} & \bullet & [\text{upper tag name}] & \bullet & [\text{lower tag name, constant, or zero}] & \Rightarrow \end{array}$$

The U-TAG operation provides the programmer with a means of expressing the upper half of a storage address by means of a symbolic tag. This is the only method by which this may be done. The programmer has the option of specifying a tag in the lower half of the word also. This operation is useful for such purposes as the preparation of jump tables and the specification of upper and lower buffering limits.

- V_0 - gives the name of the upper tag. A constant is not permitted
- V_1 - gives the name of a lower tag if desired. If no tag is desired, this must be **0** (see example b, below)

Example a:

$$\mathbf{DOG16} \Rightarrow \mathbf{U-TAG} \bullet \mathbf{CAT4} \bullet \mathbf{MOUSE7} \Rightarrow$$

Tags **CAT4** and **MOUSE7** represent the upper and lower 15 bits respectively of the storage location represented by the label **DOG16**. Assume that the following allocation values are given on an allocation tape:

$$\begin{array}{ll} \mathbf{MOUSE7} \Rightarrow & \mathbf{563} \\ \mathbf{CAT4} \Rightarrow & \mathbf{53210} \\ \mathbf{DOG16} \Rightarrow & \mathbf{3000} \end{array}$$

The computer word produced as a result of the **U-TAG** poly-operation is: 03000 53210 00563.

Example b:

$$\mathbf{RAT13} \Rightarrow \mathbf{U-TAG} \bullet \mathbf{DCON} \bullet \mathbf{0} \Rightarrow$$

The tag **DCON** represents the upper 15 bits of the storage location represented by the label **RAT13**. The V_1 operand of 0 causes the lower half of the word produced to be filled with 00000.

DECLARATIVE OPERATIONS

<u>Operation</u>	<u>Page</u>
EQUALS	2
MEANS	4
RESERVE	6
COMMENT	7

DECLARATIVE OPERATIONS

The programmer frequently wishes to supply to the assembler certain information for use in the assembling process which does not generate an instruction. The information may be involved in subsequent operations in constructing a machine-code instruction; or it may be substituted for already existing data or information, thereby extending the scope and power of the operation. This is especially true where, by changing one operand in an operation, the operation may perform a variety of similar tasks.

Declarative operations, therefore, are operations which do not result in the generation of instructions in the object program; they rather 1) give information about relationships, such as equality between data and/or symbolic names, 2) make assertions, and 3) define a procedure. Declarative operations state facts and provide information which the assembler either utilizes, or stores and later incorporates into the object program instructions it generates.

In all cases, the programmer must state the declarative operation at some place ahead of the action operation which is to use it. These operations can intermingle with action operations anywhere in the program, provided they comply with the above priority restriction. It is often worthwhile for the programmer to place the declarative statements on a separate **PROGRAM** tape or punched cards, to be read into the assembler before the main program.

EQUALS Operation:

$$\begin{array}{ccc} L & & W \\ & & - V_0 - \\ [unknown\ tag] & \Rightarrow & \mathbf{EQUALS} \bullet [known\ value: lab/tag \pm i, or\ a\ constant] \Rightarrow \end{array}$$

The **EQUALS** operation establishes an equivalence between one expression, L , whose allocation value is *unknown* and another expression, V_0 , for which the allocation value is known. This provides the programmer with a versatile allocation aid whereby he can transfer an allocation value from one label or tag to another tag. Since this operator is concerned solely with allocation, an assembler function, it generates no instructions in the internal program.

This operation permits addition, +; subtraction, -; multiplication, () (); division, /, with known values. A term in the arithmetic process may be a constant or a tag (\pm increment is permitted); a factor is an expression made up of terms connected by + or - signs; it corresponds to an address. Computations progress sequentially upon factors, with the terms in each factor accumulated separately before multiplication and/or division. Thus the computations are essentially multiplications and/or divisions of addresses.

L - gives the name of the *unknown* tag to which a numeric value is to be assigned

$-V_0-$ - gives: a) the constant which the programmer wishes to assign, or b) the label or tag whose value is known, with or without an increment, or c) a combination of labels, tags, and/or constants in an arithmetic relationship. Each value may consist of one or more of the following: 1) a number; 2) a label or tag; 3) a numeric increment; 4) a numeric decrement; 5) a tag with increment; or 6) a tag with decrement. Two or more of these may be joined together by either successive multiplications or by successive divisions, but not a combination of the two processes. The expression may also be an accumulation of two or more additions and/or subtractions of known values. In expressions combining addition or subtraction with either multiplication or division, the addends or subtrahends are treated as increments or decrements to the factor with which they are immediately associated. For example, in the expression:

NUB-4/CHOP-5+COB

(NUB-4) serves as the dividend, and **CHOP-5+COB** are combined into a single divisor value. Regardless of how $-V_0$ - is expressed, a single absolute allocation value for the entire expression must be known by the assembler. The assembler stores this value for later use when the *unknown* tag, *L*, is referenced. B-register designations are not permitted in the $-V_0$ - operand position.

Example a:

CAT ➔ **EQUALS** • **DOG+2** - **HORSE** ➔

(e.g., if **DOG** = 300 and **HORSE** = 100; **CAT** = 202.)

Example b:

SR3 ➔ **EQUALS** • **SR4-33D+44+RATS** ➔

Example c:

TMAX ➔ **EQUALS** • **45600** ➔

Example d:

PECE ➔ **EQUALS** • **(DOVE+2) (MANY)** ➔

Example e:

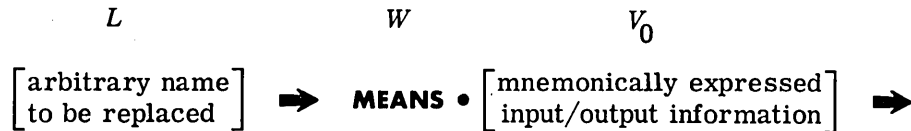
CRUMP ➔ **EQUALS** • **RACER-4/BOOL/5** ➔

In this example, the computer subtracts 4 from the value represented by **RACER**, divides this result by the value allocated to **BOOL**, then divides this result by 5; it then assigns this value to **CRUMP**. The **EQUALS** operation thus has the power to perform arithmetic computations within the compiler.

Note: In cases of multiplication, if the product exceeds five characters, an error printout occurs.

In cases of division, if the quotient is not an integer, an error printout occurs.

MEANS Operation:



The **MEANS** operation replaces an arbitrary name in the label, L , position with input/output information expressed in mnemonics. It permits programs to be written with complete flexibility concerning the assignment of channels to external equipment. By holding the assignment of external equipment open until needed, the programmer can at that time determine which of the specific channels are available for use. He then replaces the general assignment with the one he desires by entering a **MEANS** statement in the L_0 program prior to the operation performing the input/output function. The computer then takes what is in $-V_0-$ and replaces in the subsequent I/O operation the value assigned to L in the **MEANS** operation. Thus the programmer can assign an external equipment to any Input/Output or Function channel. L_0 operations which may contain replaceable general operands include: **STR, JP, TERM, IN, OUT, EX-COM, EX-COM-MW, SIL-EX,** and **RIL-EX.**

This operation does not generate any internal assembler instruction; it makes the indicated substitution, then drops out. The Input/Output operation(s) subsequently involved in the transfer then function as usual, using the substituted operand.

The **MEANS** operation is applicable only to the assignment of input/output parameters. It cannot be used interchangeably with **EQUALS**, nor can **EQUALS** be used to perform the task assigned to **MEANS**

- L - gives an arbitrary name to be replaced. This has normal label format; i.e., there are no special restrictions regarding the symbols entered in L
- V_0 - states the specific input/output assignment to be substituted for L . Entries in this operand position are presently restricted to information regarding Input/Output specifications. They may consist, therefore, of any unique external equipment assignment, e.g., **C4 ACTIVEOUT; C5;** or a function constant

Example **MEANS** operations:

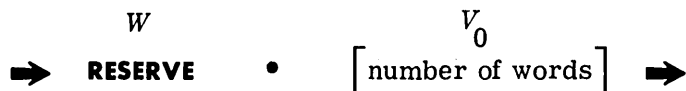
- a) **LOB** ⇒ **MEANS • C12**
- b) **PITCH** ⇒ **MEANS • C15ACTIVEOUT**
- c) **CUT** ⇒ **MEANS • C0**

Examples of Input/Output operations with which the foregoing examples may be used:

- a) ⇒ **EX-FCT • LOB • 426**
- b) ⇒ **JP • POST • PITCH**
- c) ⇒ **OUT • CUT • W(SNAP) • MONITOR**
- d) ⇒ **TERM • LOB • INPUT**

Note: **MEANS** operations appear either within an L_0 program or as separate input under a **PROGRAM** header. In both cases the **MEANS** operations become a part of the assembler's L_1 table storage.

RESERVE Operation:



The **RESERVE** operation sets aside a block of memory locations in the running (object) program. It does so by adding the number expressed by the V_0 operand to the current allocation address and storing the next generated instruction at the incremented address. Thus the reservation of space begins at the location following that of the previously generated instruction and includes the V_0 number of continuous locations. The assembler does not clear these locations; it merely by-passes them during allocation. Some of the special reasons for reserving such an area include:

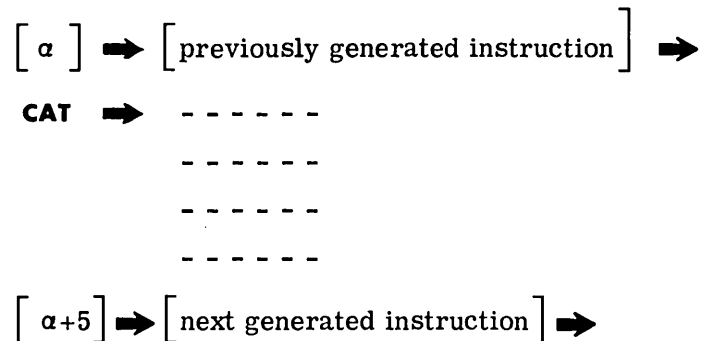
1. Setting aside a specific area for the storage of parameters
2. Leaving an area open for working storage
3. Reserving space, e.g., at the end of the program, for expansion purposes
4. Subsequent insertion of other program instructions

V_0 - specifies the number of words to be reserved. The programmer may enter only a constant in this operand location

Example:

CAT \Rightarrow **RESERVE** \cdot **4** \Rightarrow

Result:



The use of a label to identify the first word of the reserved area permits the referencing of the entire area or of any word location in it. The programmer may gain access to any word of information in the area by referencing the label, or the label plus the increment required to designate the desired word. The operation **ENT·A·W(CAT+3)**, for example, reads into register A the content of the fourth word in the reserved area in the example above.

COMMENT Operation:

➔ **COMMENT** • [message] ↘

The **COMMENT** operation permits the programmer to place a message(s) within the input program to provide added information for edited records of the problem definition. This operation is declarative; it has no dynamic meaning to the input language.

Example:

➔ **COMMENT** • **THIS** △ **SUB-PROCEDURE** △ **CONTAINS** △
TYPE-X △ **LISTING** △ **TECHNIQUES** ↘

DEBUGGING OPERATIONS

Debugging computer programs is a tedious and time-consuming process. AS-1 provides *debugging aids* to expedite program debugging. These aids utilize computer action, whenever possible, to assist the programmer in this task.

The debugging aids provide a *list of changes* in the contents of computer words within a *designated area* of core storage during execution of the program being debugged. In addition, they make a non-zero *dump* of the designated area, or areas, at any point of the program during its execution; they likewise may dump the contents of registers A, Q, and B¹ through B⁷.

Debugging operations establish an area of storage, called an *image*, which duplicates a *designated area* of core storage. Images are placed in core storage, providing it does not hinder the program. Individual operations near the beginning of the L₀ program must define the areas to the assembler (see Figure 1 below).

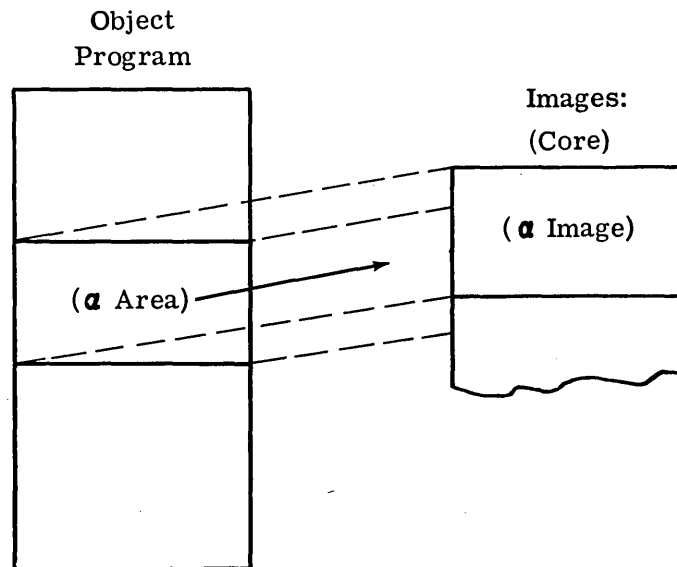


Figure 1. Area Image Formation

Testing an image indicates the changes made in an area since it was last tested. A test consists of a word-by-word comparison between a program area and its image. Those words found to have been changed are printed out, program *vs* image. New words from the program then replace the old in the image, thus updating the image after each printout. Image tests are permitted at any point of the program. (see Figure 2.)

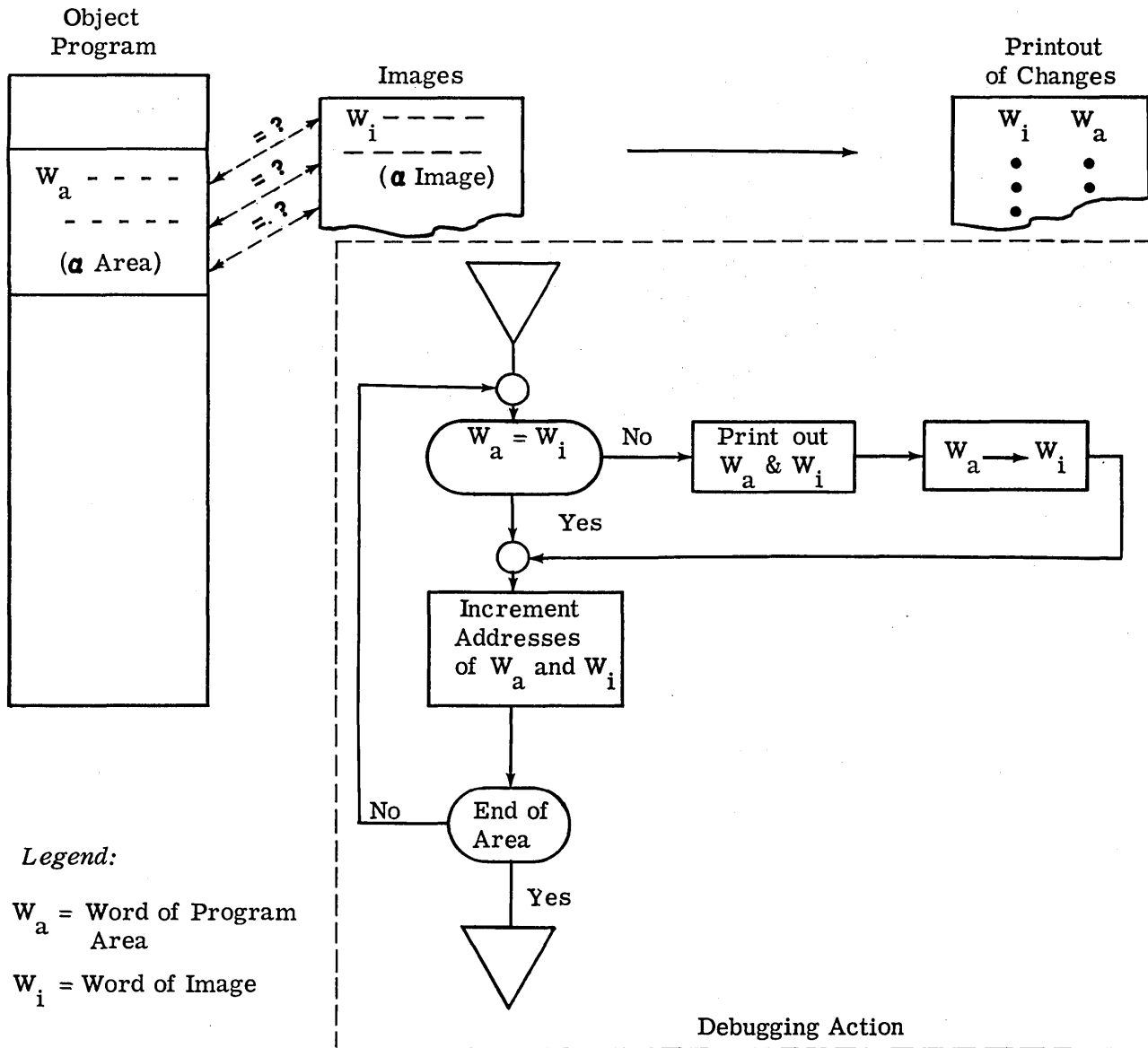


Figure 2. Test Image - Debugging Operation

Debugging action also dumps debugging areas or registers at designated points of the L_0 program. An area dump prints the entire area contents. A register dump prints the content of registers A, Q, and all B's.

An additional set of instructions, called the *Debugging Package*, performs debugging action while the object program is being executed. Debugging operations of the original L_0 program cause the formation of return jump instructions in the object program with the debugging package as their destination. This package is in relative format and is loaded at an initial

address 76000 unless specified differently by the programmer. The programmer places special allocation in his L_0 allocation tape whenever this package is not to be at address 76000. The allocation is as follows:

L W

DEBUG \Rightarrow [loading address of Debugging Package] \Rightarrow

Upon completion of debugging, the programmer requests editing and object program outputs without Debugging Aids. AS-1 automatically eliminates action by the Debugging operations of the L_0 program, and no object program instructions result from them.

Debugging operations divide into two types: 1) declarative and 2) action. The declarative operations delimit the areas and images for later reference and manipulation. These always precede the debugging action operations in the L_0 program. Debugging action operations either test images or dump areas and registers.

The following pages define individual debugging operations.

DECLARATIVE DEBUGGING OPERATIONS

DEFine- AREA Operation:

W V_0 V_1 V_2

➔ **DEF-AREA** • [area name] • [initial area address] • [number of words] ➔

The **DEF-AREA** operation defines an area of the L_0 program for debugging action. Debugging action operations later manipulate data in terms of these declarative area definitions.

- V_0 - names a debugging area. The name is symbolic and is used for identification purposes *
- V_1 - specifies the initial debugging area address. This operand is an allocated tag or an absolute address; the tag may be incremented
- V_2 - specifies the number of words in the debugging area. This operand is either a symbolic tag or an absolute number; if symbolic, it must be allocated

* It is permissible for the programmer to use an identical symbol to represent both V_0 and V_1

CORE-IMAGE Operations:

W V_0 V_1 V_2^*

➔ **CORE-IMAGE** • [area name] • [initial image address] • [key set condition] ➔

The **CORE-IMAGE** operation establishes an image of a debugging area on the core. This operation provides the initial image formation; thereafter it is updated with changed words of corresponding debugging area positions by action of the **TEST-IMAGE** operation.

V_0 - names the defined debugging area to be imaged

V_1 - specifies the initial core address for the image. This is a tag (allocated) or an absolute address; a tag may be incremented

V_2^* - specifies a key set condition. Key settings and their significance are as follows:

Perform operation if V_2 is **KEY1** and console key 1 is set.

Perform operation if V_2 is **KEY2** and console key 2 is set.

Perform operation if V_2 is **KEY3** and console key 3 is set.

*Operand use is optional

ACTION DEBUGGING OPERATIONS

DUMP-REG isters Operation:

W V_0^*

➔ **DUMP-REG** • [key set condition] ➔

The **DUMP-REG** operation causes a printout of registers A, Q, and B¹ through B⁷ at points specified in the running program.

V_0^* - specifies a key set condition. Key settings and their significance are as follows:

Perform operation if V_0 is **KEY1** and console key 1 is set.

Perform operation if V_0 is **KEY2** and console key 2 is set.

Perform operation if V_0 is **KEY3** and console key 3 is set.

Typical **DUMP-REG** printout:

A 02530 33330 Q 00000 22764

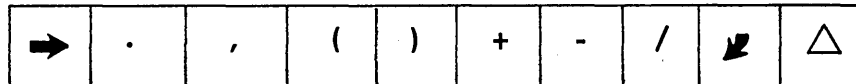
B¹ 00010 B² 07774 B³ 00022 B⁴ 00040 B⁵ 02040 B⁶ 00000 B⁷ 00012

*Operand use is optional

EXCHANGE Operation:

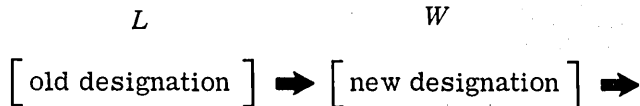


EXCHANGE, a minor header operation similar to **ALLOCATION** and **CHAN-SET** operations, incorporates special reassignment properties, enabling a programmer to substitute one element designation for another everywhere it appears in input. Operations immediately following an **EXCHANGE** header establish desired substitutions for any existing elements of an L_0 program or other input data. Each element to be substituted must exist between any two of the following normal separator symbols of an operation:



The **EXCHANGE** data must always be read into the computer prior to any data which it will affect. Any normal assembling process can be carried out in conjunction with the **EXCHANGE** operation, since the substitutions are made prior to placement of data in the L_1 table. All **EXCHANGE** substitutions are permanent and are reflected in assembler outputs. A label and identifying operands are optional with the **EXCHANGE** header. Subsequent assignment operations require labels.

EXCHANGE assignment operations are listed immediately after the **EXCHANGE** header and are in the following format:



- L - gives the designation of the *old* element for which substitution is to be made
- W - gives the *new* designation to be substituted for the old wherever it is found in the input data

The following is an example of a set of **EXCHANGE** operations incorporated into an **A-CONTROL** input tape together with the L_0 input program:

COUNTONES ➔ **A-CONTROL • SMITH • 20 JULY 63**
 ➔ **EXCHANGE**
60000 ➔ **61000**
SUM0 ➔ **SUMX**
B2 ➔ **B5**
B1 ➔ **B2**
35 ➔ **12**
L ➔ **W**
NWORDS ➔ **ZWORDS**
STOP5 ➔ **STOP7**
CTONES1 ➔ **CTONES4**
RJP ➔ **JP**
QNEG ➔ **QPOS**

 ➔ **ALLOCATION**
C TONES ➔ **60000**
SUM0 ➔ **63000**

..
COUNTONES ➔ **PROGRAM • SMITH • 20 JULY 63**
CTONES0 ➔ **ENTRY**
 ➔ **CL • B2**
CTONES1 ➔ **ENT • B1 • 35**
 ➔ **CL • A**
 ➔ **ENT • Q • L(WORD0+B2)**
CTONES2 ➔ **LSH • Q • 1 • QNEG**
 ➔ **ADD • A • 1**
 ➔ **BJP • B1 • CTONES2**
 ➔ **STR • A • W(SUM0+B2)**
 ➔ **BSK • B2 • NWORDS**
 ➔ **RJP • CTONES1 • STOP5**
CTONES3 ➔ **JP • CTONES3**
 ➔ **EXIT**

..

The effect of the **EXCHANGE** operations on the subsequent input data in the above example would be as follows:

```

COUNTONES  ➔ A-CONTROL • SMITH • 20 JULY 63
              ➔ ALLOCATION
BASE         ➔ 61000
ENTRANCE    ➔ CTONES0
SUMX        ➔ 63000

```

```

..
COUNTONES  ➔ PROGRAM • SMITH • 20 JULY 63
CTONES0     ➔ ENTRY
              ➔ CL • B5
CTONES4     ➔ ENT • B2 • 12
              ➔ CL • A
              ➔ ENT • Q • W(WORD0+B5)
CTONES2     ➔ LSH • Q • 1 • QPOS
              ➔ ADD • A • 1
              ➔ BJP • B2 • CTONES2
              ➔ STR • A • W(SUMX+5)
              ➔ BSK • B5 • ZWORDS
              ➔ JP • CTONES4 • STOP7
CTONES3     ➔ JP • CTONES3
              ➔ EXIT

```

..

NOTE 1: The most effective use of **EXCHANGE** operations is in making special corrections, or in conjunction with L_1 corrections, where an output No. 26 is desired.

NOTE 2: Because of the universal nature of the **EXCHANGE** operator, programmers should exercise a certain degree of caution when using it, remembering that a requested substitution is made everywhere in the input data where the old designation is encountered. Substitution cannot be selective.

DEBUGging-AIDS Operation:

L *W*
 ➔ **DEBUG-AIDS** ➔

DEBUG-AIDS, an independent operation, indicates to the assembler that the programmer wishes to make use of the debugging aids included in his program. When the **DEBUG-AIDS** operator is present, the assembler will generate a series of return jumps to the routines of the debugging package corresponding to those specified in the program (see Debugging Operations).

Used under **A-CONTROL**, no label or operands are required.

INTRODUCTION

ASSEMBLER SUPPORT PROCESSES

An assembling system of the scope, versatility, and complexity of AS-1 must necessarily carry with it various support, service, and control processes. These processes simplify the operation of the AS-1 Assembler by relieving the programmer and the computer operator of many of the repetitive functions concerned with the assembling system. The support process integrates the control of the assembler functions.

This section describes the support processes for the AS-1 Assembling System, and presents the rules for using them. It presents the rules for applying the AS-1 Programming Language to the solution of problems. In essence, this part tells the programmer what the assembling system does, what aids he has at his disposal to make it work, and how to use these aids.

The support processes described in this manual include:

- AS-1 Program Input
- Allocation
- Program Corrections
- AS-1 Program Output
- Assembler - Control Operations

In addition, the Appendix contains various charts, rules, and other pertinent information.

AS-1 LANGUAGE LEVELS

The AS-1 Input (Source) Language loads into AS-1 via paper tape. This working language is called L_0 . AS-1, by a series of translations, produces an object program on paper tape which can be loaded into the Unit Computer and executed. This working language is called L_4 . There are three intermediate forms in which information is stored in tables during the AS-1 operation. These "language levels" are called L_1 , L_2 , L_3 . A description of each of these languages follows:

- L_0 - is the original input language consisting of AS-1 operations and used by the programmer in writing routines and preparing them for assembler input

- L₁ - is a slightly modified L₀ in which the operation items are stored (in a special 6-bit code) within assembler tables. Certain input translations and error detection steps have been performed; operations remain in mnemonic form, however, and their basic characteristics are similar to L₀. The conversion of L₀ operations to L₁ operation items is a one-to-one relationship
- L₂ - consists of information tables and a list of operation items which are very similar to machine instructions. The *f j k b* designators are in absolute octal code, but the addresses are symbolic. The program is not yet committed to any given memory area
- L₃ - is the language of the object program, stored in table form within the assembler. All addresses have been assigned a permanent (absolute) address. It can also be buffered directly into core memory as object program
- L₄ - is the object program on paper tape which can be loaded into the Unit Computer and executed; it can also be buffered directly into core memory

AS-1 INPUT

The Paper Tape High-Speed reader handles L_0 paper tape input to AS-1 Assembler. These tapes are punched with an identifying header operation on the front of the tape. A header operation consists of a program name, a header-type, and identifying operator. Each of these is limited to 10 alphanumeric characters and conform to the following format.

$$\left[\begin{array}{c} \text{program identification} \\ \text{label} \end{array} \right] \cdot \left[\begin{array}{c} \text{header-type} \\ \text{operator} \end{array} \right] \cdot \left[\begin{array}{c} \text{programer's} \\ \text{name} \end{array} \right] \cdot \left[\text{date} \right]$$

The format control symbols, called separators, used when coding in AS-1 are as follows:

NAME	SYMBOL	SIGNIFICANCE
CARRIAGE RETURN	↵	Signifies end operation. Must precede header operation, delimits the statement
TAB	→	Must always precede the statement operator. Must precede notes: omit if notes not given
POINT SEPARATOR	•	Separates statement components
OPEN PARENTHESIS	(Separates a K-designator symbol from its interrelated operand
CLOSED PARENTHESIS)	Signifies the end of operand preceded by a K-designator symbol
SHIFT DOWN	↓	No significance to L_0 language except to obtain the proper symbols, i.e., + sign
SHIFT UP	↑	No significance to L_0 language except to obtain the proper symbols
SPACE	Δ	No significance in L_0 language controls. It merely leaves space in messages when they are typed out. (Exception: see poly-operations for typing, punching, and printing)
PLUS	+	Specifies addition
MINUS	-	Specifies subtraction

BAR		Delimits control keys when programing a message for printout on the typewriter
DOUBLE PARENTHESES	() ()	Specifies multiplication
SLANT SIGN	/	Specifies division
DOUBLE LOWER CASE PERIOD	..	Indicates the end of tape (stop reader)

Header operations are declarative. These operations direct the assembler as to the handling of succeeding operations. Header types are: 1) Program, 2) Allocation, 3) Correct-L₁, 4) Exchange, 5) Rel-Alloc, 6) Indr-Alloc, 7) Chan-Set, 8) Debug Aids and 9) A-control. These headers are grouped as major and minor headers where Program, Correct-L₁, and A-control belong to the major class. The usually extra size and usage frequency of the header-type tapes categorize the major classes. The examples below illustrate three typical headers.

<i>L</i>		<i>W</i>	<i>V</i> ₀	<i>V</i> ₁
↙ ↑	COUNTONES	➔	PROGRAM	• SMITH • 20 JULY 1963
↙ ↑	COUNTONES	➔	A-CONTROL	• SMITH • 20 JULY 1963
↙ ↑	COUNTONES	➔	CORRECT L1	• SMITH • 20 JULY 1963

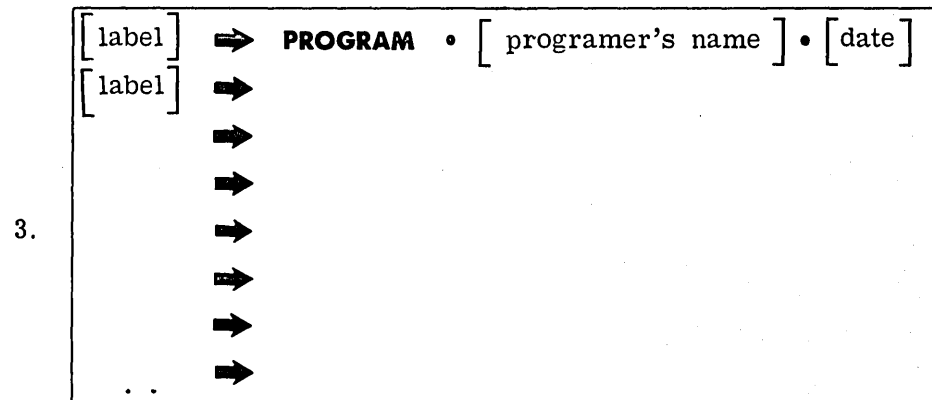
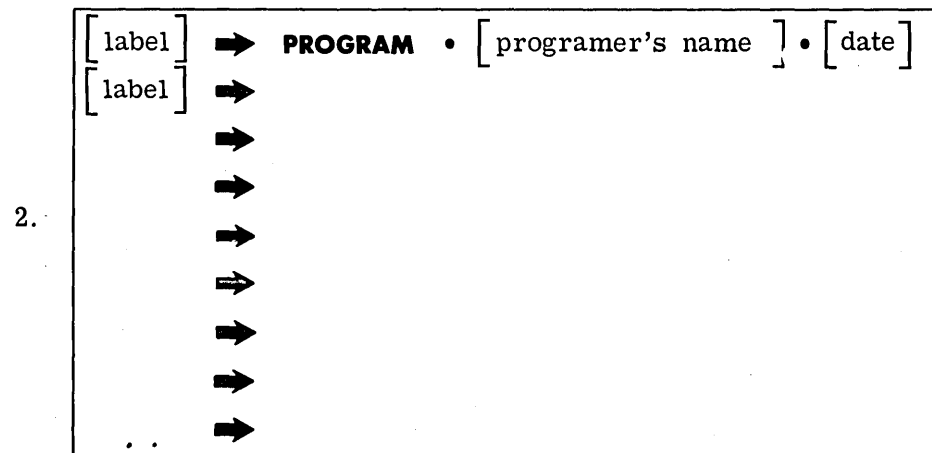
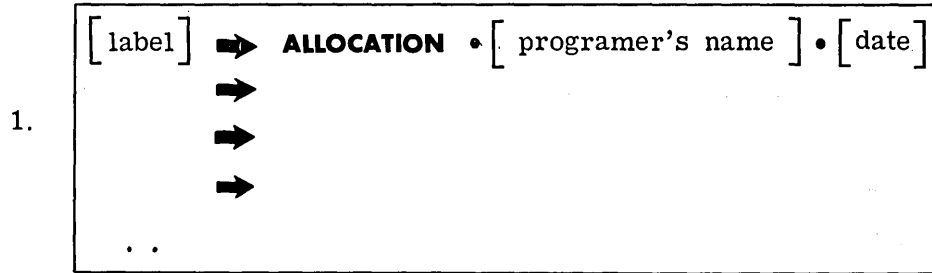
It is a good policy to maintain like labels and operands in headers when assembling a routine. The assembler does not require this complete format. In fact, the header operator works with only the header-type operand present. The extra operands supply the programmer with additional routine organizational data. The Correct-L₁ operation alters the program header to agree with its label and operands during the correction run. For this reason, a Correct-L₁ header should always contain the current date.

The A-control header merely categorizes control information to the assembler under a single header; its use is optional since the headers work independently. The A-control eliminates the need for loading many small declarative paper tapes. The extent of assembler control as AS-1 input is optional. The programmer may control all assembly activity by paper tape entries or he may instruct the computer operator to control much of the assembling at the console.

The program header declares to the assembler that the following operations go to the assembler's L₁ table storage. These operations may contain mono, poly, and declarative operations, organized by the programmer to accomplish some programming task. The declarative operations, such as means and equals must follow the first program header of the routine being assembled. A routine may be comprised of many paper tapes, each with a program header. The operations will appear in the L₁ table storage, sequentially as they load.

The following skeleton program samples illustrate typical AS-1 read-in arrangement.

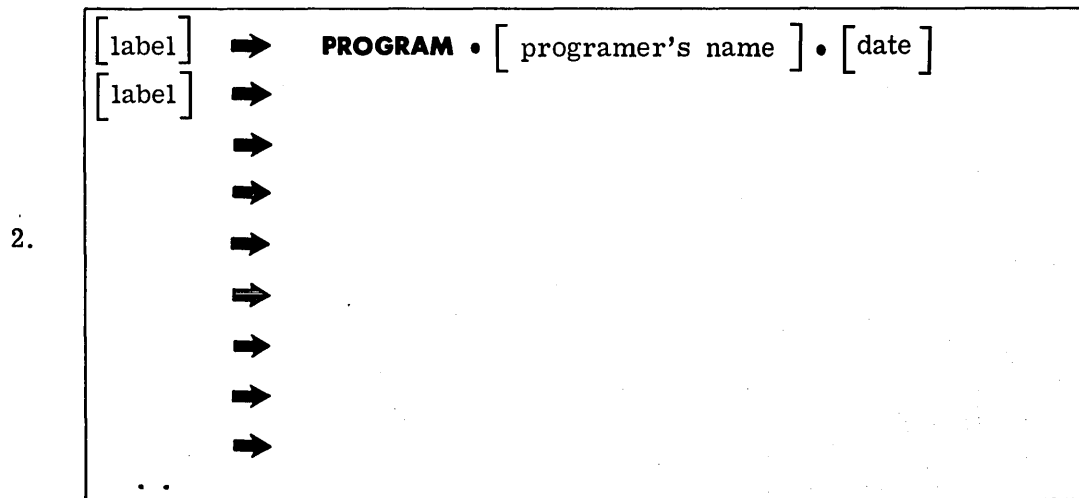
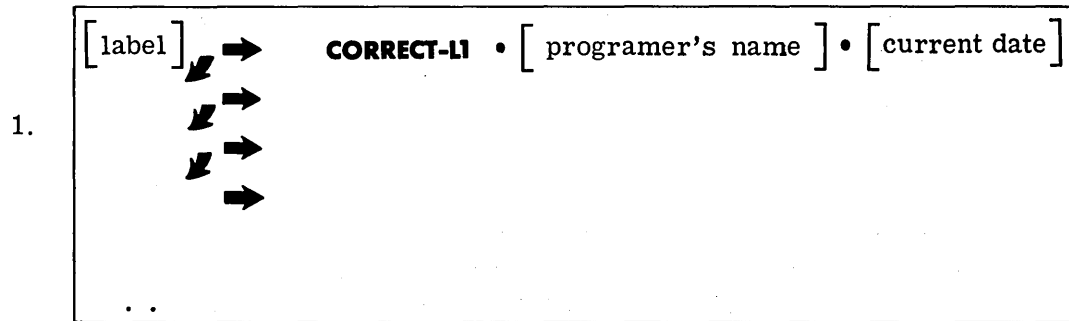
Sample 1. (Compile a Simple Routine)



Instructions to the assembler:

- 1) Read-in order of tapes
- 2) Debugging aids desired or not
- 3) Output types

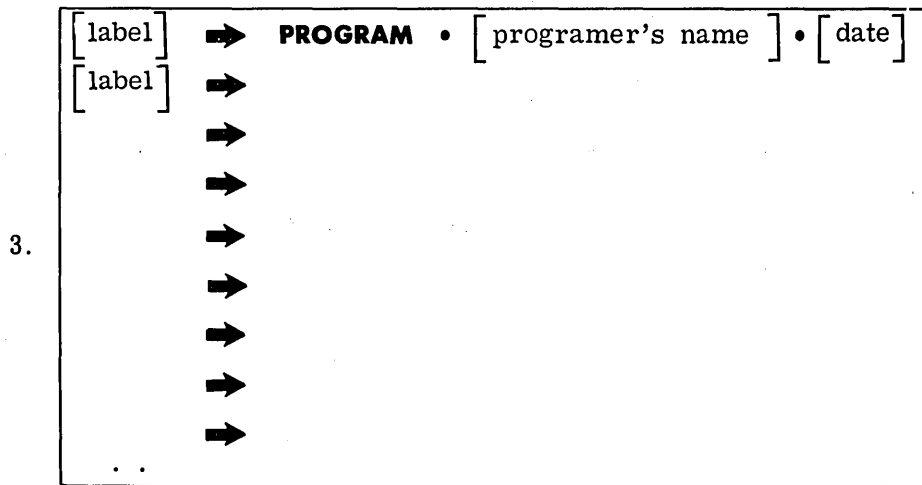
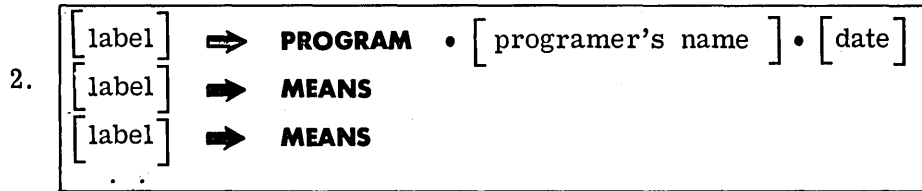
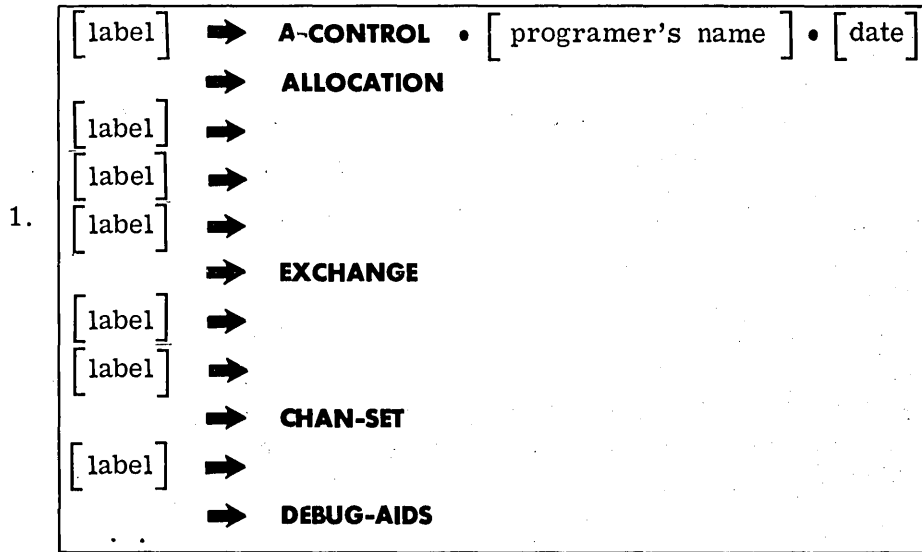
Sample 2. (Correct a Single Routine)



Instructions to assembler operator:

- 1) Read-in order of tapes
- 2) Request output numbers

Sample 3. (Assemble Single Routine under Programmer Control)










Instruction to assembler operator:

- 1) Read-in order of tapes
- 2) Request output numbers

PAPER TAPE INPUT

Input requirements permit either upper- or lower-case characters for all inputs with the exception of separators. However, upper case characters are recommended for all operation inputs except those which specifically require lower case characters. Refer to Table I for equivalent input format codes.

TABLE I. CODING SYMBOLS - PAPER TAPE INPUT

SOFTWARE NAME	SOFTWARE SYMBOL	FLEXOWRITER CODES	FIELD DATA SYMBOL SUBSTITUTION	FIELD DATA CODES
Carriage Return		45		04 03
Shift Up		47		01
Shift Down		57		02
Tab		51	Special 	76
Point Separator	•	44	Apostrophe ' 	72
Double Period	..	57 42 42		75 75
Space	Δ	04		05
Comma	,	57 46 47		56
Vertical Bar		57 50 47	Exclamation ! 	55
Plus	+	57 54 47		42
Minus	-	56		41

A double lower case period in the L coding position indicates the end of tape read-in. Therefore, each paper tape begins with a header and ends with a double-period end symbol.

The following examples illustrate the basic format for program operations and the common usage of separators therein.

<i>L</i>		<i>W</i>	<i>V₀</i>		<i>V₁</i>		<i>V₂</i>		<i>N</i>		
CAT4	➔	ENT	•	Q	•	W(RAT3-2+B6)	•	QNEG	➔	RATCHECK	↷
	➔	RPT	•	36	•	BACK					↷

Notice that it is essential to use a straight arrow before each operator even when a label is not given. The second straight arrow is used only when notes are given. The point symbol separates the components of the statement. Parenthesis symbols indicate contents of a storage location modified by an operand code. Also within the parenthesis symbols are data unit subnames and subscripts or multiplication factors. Spaces are permitted throughout the operation. The curved arrow indicates the end of the operation, or of the notes if present.

ALLOCATION

Allocation is the process of assigning numeric values to certain symbolic representations. (See Figure 1.) Such values usually pertain to addresses of storage locations within the computer; however, they may also give rise to constants. The AS-1 allocation process requires a specific L_0 allocation for the base address of the L_4 object program and for certain labels and tags; these are:

1. The initial address of the L_4 object program
2. Tags making reference to operations of other programs or to storage areas
3. Labels of first action operations following a program sequence break
4. Tags representing a numeric value other than an address. (Such values have a 5-octal digit limit)

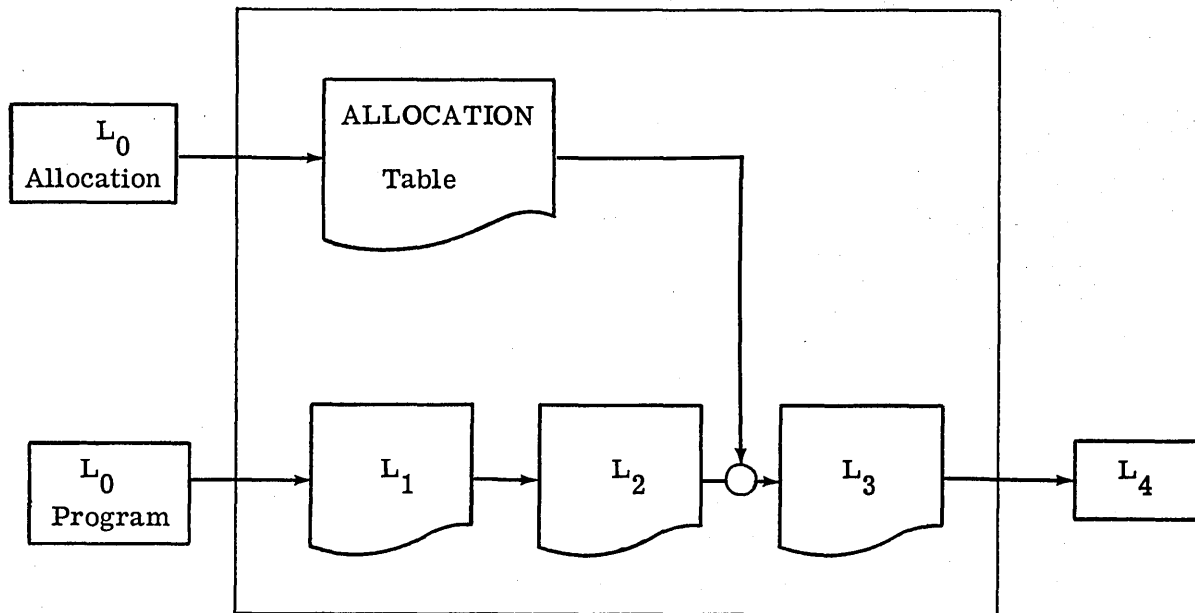


Figure 1. Allocation

LABEL AND TAG ALLOCATION

The AS-1 Assembly System provides a variety of methods of making allocations to tags and labels in programs written in relative format. These methods include the following header types or classifications:

1. **ALLOCATION** (normal)
 - a. Direct
 - b. Substitutive
2. **REL**ative - **ALLO**cation
3. **IND**iRect - **ALLO**cation

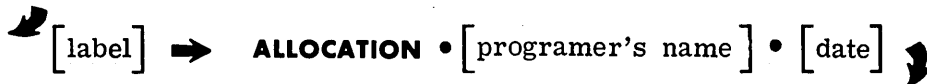
Types 1, a and b, and 2 above will be presented in this discussion. Because of the differences in performance and application, type 3, **IND**iRect - **ALLO**cation, will appear under another heading of this section (see Indirect Allocation.)

The programmer should note that the three allocation types listed above are the operators, *W*, of *header operations*. As such they form, together with a label and two identifying operands, the header operations of the allocation tapes written in L_0 format and resemble L_0 program headers. The programmer may use them separately for independent tape preparation.

Normal Allocation:

The **ALLOCATION** header introduces two methods of allocation: a) direct, and b) substitutive.

Example header:



The operation items following the header identify a list of L_0 allocations, each of these allocation items requires the use of two coding positions: 1) the lab/tag being allocated, placed in the *L* coding position; and 2) the allocation value or equivalent lab/tag, placed in the *W* coding position.

- a. *Direct* allocation sets a lab/tag to a given numeric value (a constant)

Direct allocation format:

$$\begin{array}{l} \overset{L}{[\text{label}]} \Rightarrow \overset{W}{\text{ALLOCATION}} \bullet [\text{programer's name}] \bullet [\text{date}] \\ [\text{lab/tag}] \Rightarrow [\text{numeric value}] \\ [\text{lab/tag}] \Rightarrow [\text{numeric value}] \\ \text{etc.} \end{array}$$

The programer may delete any tag or label previously allocated by specifying the lab/tag and stating the word **DELETE** as the operator:

$$\overset{L}{[\text{lab/tag}]} \Rightarrow \overset{W}{\text{DELETE}} \Rightarrow$$

- b. *Substitutive* allocation sets a new tag, L , equal to a known lab/tag, with or without increments, or to the result of two or more lab/tags, W , with or without increments, joined arithmetically. Addition, subtraction, multiplication, or division of terms is possible, a term being either a lab/tag with or without an increment, or a numeric constant. Any number of these arithmetic processes may be combined in an operation, the only restriction being that multiplication and division cannot be performed in the same operation item. The assembler performs all additions and subtractions first, then does the multiplications or divisions, just as in the EQUALS operation (see Declarative Operations)

Substitutive allocation formats:

$$\begin{array}{l} \overset{L}{[\text{label}]} \Rightarrow \overset{W}{\text{ALLOCATION}} \bullet [\text{programer's name}] \bullet [\text{date}] \\ [\text{lab/tag}_a] \Rightarrow [\text{lab/tag}_b \pm i^*] \Rightarrow \\ [\text{lab/tag}_a] \Rightarrow [\text{lab/tag}_b \pm i \left\{ \begin{array}{l} / \\ \pm \\ () \end{array} \right\} n] \Rightarrow \\ \\ [\text{lab/tag}_a] \Rightarrow [\text{lab/tag}_b \pm i] \pm [\text{lab/tag}_c \pm i] \Rightarrow \\ [\text{lab/tag}_a] \Rightarrow ([\text{lab/tag}_b \pm i]) ([\text{lab/tag}_c \pm i]) \Rightarrow \\ [\text{lab/tag}_a] \Rightarrow [\text{lab/tag}_b \pm i] / [\text{lab/tag}_c \pm i] \Rightarrow \\ [\text{lab/tag}_a] \Rightarrow [\text{lab/tag}_b \pm i] / [\text{lab/tag}_c \pm i] / [n] \Rightarrow \end{array}$$

In cases of multiplication, if the product exceeds five characters, an error printout occurs. In cases of division, if the quotient is not an integer, an error printout occurs.

*The increment is optional in all the examples shown

RELative - ALLOCation:

Relative allocation provides a convenient technique for allocation. A **RELative- ALLOC**ation tape is assembler produced and has a format which resembles that of direct allocation except that the numeric value is relative to the *base* value (always 0 0 0 0) instead of a final value. The base increment value, independent of the **REL-ALLOC** tape, is furnished by the operator, who manually enters this value in the A register during a subsequent assembly run. The assembler then adds the relative values to the increments to form final allocations.

A **REL-ALLOC** tape results as an assembler output, output No. 34. The tape content consists of a header and a series of allocations thereafter (to the base) in the format as shown below:

```
[label] → REL-ALLOC  
[label] → [increment allocation value]  
[label] → [increment allocation value]  
etc.
```

Important applications of relative allocation follow:

1. Assembling Applications

- a. **RELative- ALLOC**ation tapes (output No. 34) may be used to allocate a program or a group of programs relative to some base as stated earlier in this section
- b. Relative load programs (output No. 24) produced by the assembler may be loaded anywhere in memory relative to a specified base address. Subsequent assembly runs requiring *allocation information* pertaining to the above program may obtain such allocation information from the content of a **REL-ALLOC** tape produced during the same assembly run as the relative load tape. In such cases the assembler operator assigns the same base to relative allocations as that used to position the relative load program

Example:

Program A, assembled in relative load format, produces a **REL-ALLOC** tape, and is assigned to address 60000. Program B, which will refer to labels in program A, is then assembled. To obtain the correct addresses of labels in program A for program B, the relative-allocation tape from program A must be supplied to program B as input. When this tape is loaded for program B, the A register is set to 60000; thus the programmer may move program A to any location without affecting its availability for program B as long as he keeps record of, and loads its proper base address in the A register.

The assembler permits using a number of **REL-ALLOC** tapes in conjunction with normal **ALLOCATION** tapes*.

2. Editing Applications

The ability to transfer routines to new locations and/or interconnect a number of routines using relative allocation assists in obtaining current edited data. After assignment of absolute allocation values by the relative-allocation process, allocation printouts provide the programmer with edited allocation information on final program positioning. Thus the programmer acquires complete edited allocation data in any order desired. (See outputs 30, 32, and 33)

*During a combined read-in of both normal allocation and relative-allocation tapes the following will occur:

- 1) If the **REL-ALLOC** tape precedes the **ALLOCATION** tape, the assembler suppresses all assembler-generated tags contained on **ALLOCATION** tape(s). The assembler re-assigns these tags
- 2) If the **ALLOCATION** tape precedes the **REL-ALLOC** tape, the assembler recognizes and accepts all assembler-generated tags of the **ALLOCATION** tape(s)

INDIRECT ALLOCATION

Indirect allocation is a means whereby a programmer may 1) use a particular subroutine at a single location for access by many different programs or for programs which are to be combined, or 2) change the storage location of subroutines without making the corrective changes in the running programs using these subroutines.

The programmer maintains a list of the subroutines, by name, for which indirect allocation can be made. With each subroutine name appears a reference address together with a designator value (1 or 2), telling whether the **ENTRY** address (beginning address) of the subroutine can be found in the upper or lower half of the reference address.

A series of **U-TAG** entries establish a jump table, to which entries and/or changes in assignment can be made. For example:



accompanied by an allocation tape containing the entries

<i>L</i>		<i>W</i>
DOG12	➔	04367
TFLX	➔	32050
MAGTAG	➔	36500

would establish the current address of **TFLX** = 32050 and **MAGTAG** = 36500 respectively in the upper and lower half of location **DOG12** = 04367.

The programmer keeps a list (or table) of subroutines, in which entries may appear as follows:

Subroutine Name	Location
_____	_____
_____	_____
TFLX	204367
_____	_____
_____	_____
MAGTAG	104367

The six-digit value giving the subroutine location consists of a **1** or **2** followed by the storage address in which the subroutine address is stored*. The 1 or 2 in the leftmost position provides the *k*-designator for the assembler-generated instruction which return jumps to the subroutine. To obtain the subroutine **TFLX**, in his program, the programmer simply writes:

➡ **RJP • TFLX** ➡

On his allocation tape, following an **INDR-ALLOC** header, the programmer would write as an operation the entry he finds in the subroutine table, as follows:

<i>L</i>	<i>W</i>
TFLX ➡	204367

The assembler would then combine the information from the **RJP** operation and the allocation operation to generate the following instruction: 65020 04367

This instruction in the running program directs the computer to the upper half of address 04367 which is 32050, the address of the **TFLX** subroutine. It can readily be seen that the value, 32050, could be changed at will without occasioning any change in the programs using the **TFLX** subroutine.

Note: The indirect allocation technique also applies to the mono-operations: **ENT • A • [tag]** and **ENT • Q • [tag]**. The machine instruction generated for each receives both *k* and *y* values via indirect allocation of the tag.

*Storage address specified in octal only

INDirect - ALLOCATION Operation:

L W V_0 V_1
[label] \Rightarrow **INDR-ALLOC** • [programer's name] • [date] \Rightarrow

The **INDirect - ALLOCATION** operation is a header operation appearing before items which provide indirect allocation information about subroutines to be used in a program. The items which follow this header have the following format:

L W
[S/R label] \Rightarrow [6 digit number]*

The subroutine labels and their accompanying six digit numbers appear in a list which the programer prepares for those subroutines which he wishes to obtain by indirect allocation. The use of subroutines prepared by other programers is permitted.

The six digit value listed with each subroutine provides a k -designator, either **1** or **2**, followed by a five-character address*. The address is the place where the subroutine locations (beginning addresses) are stored, either in the upper or in the lower half. The programer prepares the **INDR-ALLOC** tape, listing all subroutines he wishes to obtain by this method. In his program he writes a **RJP** to his desired subroutine. The assembler generates the appropriate Return Jump instruction to either the upper or lower half of the listed address.

Example indirect allocations:

TFLX \Rightarrow **INDR-ALLOC • VERSTEEG • 15SEPT63**
REG1 \Rightarrow **245000**
REG2 \Rightarrow **145000**
SIMI \Rightarrow **245001**
TAPI \Rightarrow **145001**

Note: Programers may also apply the indirect allocation technique to the mono-operations: **ENT • A • [tag]** and **ENT • Q • [tag]**.

*Storage address specified in octal only

PROGRAM CORRECTIONS

The AS-1 Assembler contains provisions for program corrections. Alterations are made to program operations while in L_1 table storage. Three types of corrections are permissible: 1) insertions, 2) deletions, and 3) replacements of program operations. (See Figure 1.)

The L_1 Corrector, a major assembler subroutine, makes the actual correction to programs in L_1 table storage. This correcting method makes use of paper tape L_0 input.

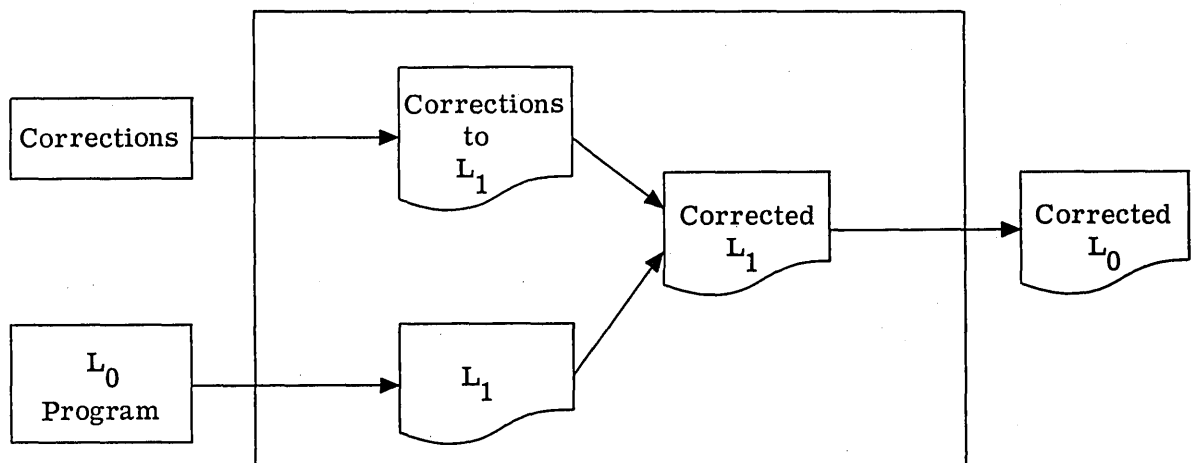


Figure 1. Corrections

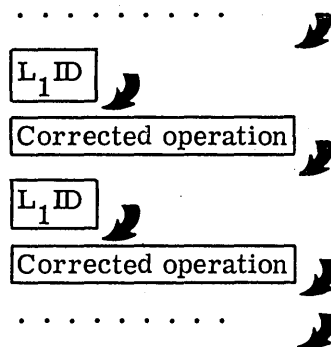
PROGRAM CORRECTIONS - PAPER TAPE INPUT

AS-1 provides a method for program corrections while in L_1 storage. This feature eliminates the necessity of retyping the entire L_0 program input tape for alterations. The L_1 Corrector, an integral part of the AS-1 Assembler, is the routine which makes these corrections. It accepts three types of correction operations: 1) *delete* operations, 2) *replace* operations, and 3) *insert* operations. The appropriate L_1 identifier precedes each correction operation to position the resulting operation item in L_1 storage. The L_1 Corrector permits any arrangement of corrections with regard to type and order.

An L_0 correction tape instructs the L_1 Corrector in making program alterations. The tape contains a header of the following format:

L	W	V_0	V_1
[program name]	→	CORRECT-LI	• [programmer's name] • [new date]

In making corrections, the programmer specifies a new date as the V_1 operand of the header operation. AS-1 automatically superimposes this on the program header, thus updating it. This provides the programmer with a means of distinguishing the old tapes from the new. Coding corrections consist of alternate L_1 identifiers and corrections as indicated in the diagram below:



An *insert* correction requires an L_1 identifier, the L_1 identifier of its preceding operation, and additional octal insert digit(s). The point symbol separates these. This locates the insert position in the L_1 item storage (e.g. an operation between L_1 identifiers 104 and 105 might be given an L_1 identifier of 104 • 1). Up to three octal insert digits are permitted in preparing *insert* corrections.

Example: *insert* correction between L_1 identified operations 27 • 4 and 27 • 5:

27 • 41 ↵
CAT3 → ADD • Q • 1 ↵

NOTE: An insertion will be replaced by a second insertion with the same ID, provided that both are read in before a real correction is made.

A *delete* correction lists the L_1 -identifier of the operation to be deleted followed by a point separator and a zero; an operation consisting of a straight arrow, →, the word **DELETE**, and a curved arrow, ↵, follows.

Example: *delete* correction, for removing the operation identified as 103:

103 • 0 ↵
→ DELETE ↵

A *replace* correction lists the L_1 -identifier of the operation to be replaced (with • 0) and the operation with which replacement is made. The new operation item is merely stored over the old, thus performing the replacement.

Example: *replace* correction, to replace the operation identified as 105:

105 • 0 ↵
→ ENT • B7 • 36 ↵

Upon reassembling, the correction tape is read in with the original L_0 program. Initial assembly action accomplishes the positioning of operation items in L_1 tables. Normal assembly runs proceed thereafter. As optional output, the programmer may request a corrected L_0 tape. This is Output No. 26 (see AS-1 OUTPUT.)

AS-1 OUTPUT

This section presents output from the AS-1 Assembler. Two external devices which produce assembler output are: 1) the on-line typewriter and 2) the High-Speed paper tape punch.

The on-line typewriter types error printout of each detected error during assembly runs. It also presents assembler status messages during assembly runs to inform the operator of the existing assembler conditions. AS-1 places the assembler object programs on punched paper tape. The off-line typewriter prepares hard-copy records from these tapes for editing purposes.

A header appears on the front of each paper tape output. The assembler punches this flex-coded data about one foot before the pertinent information. It punches declarative headers on outputs designed for input to the assembler and edits information on output designed for input to the computer as object programs (see example on page 6).

ON-LINE TYPEWRITER OUTPUT

The on-line typewriter is used primarily to inform the computer operator of assembly run conditions. The three categories of on-line typewriter output are:

1. Error Printouts
2. Assembler Status Messages
3. Program Identification (Output No. 1)

Error Printouts and Assembler Status Messages need not be requested; they are automatically typed as they are encountered during the assembly run. Output No. 1, which gives program identification, is typed automatically when assembled object program outputs are selected.

ERROR PRINTOUTS

The assembly process temporarily ceases whenever an error is detected. At this point the on-line typewriter immediately types an error printout. Action hesitates for additional input data or continues, depending on the type of error detected.

The error printout format provides for ample error identification and description. A print-out usually indicates 1) that there is an AS-1 error, 2) the kind of error, 3) the operation item label (plus or minus an increment, if any), 4) the L_1 identifier, and 5) the complete operation item containing the error. The errors UNALLOCATED TAGS and FIRST LABEL UNALLOCATED delay the assembly run until additional input has been given.

The following is a program with *deliberate* errors, and the resulting error printouts. This illustrates some of the detectable errors; these errors are encircled.

SORTNUMB	ALLOCATION•KOURAJIAN• ¹⁴ JUL ⁶³	
SORT ⁰	02500	
ZIP	06500	
..		
SORTNUMB	PROGRAM•KOURAJIAN• ¹⁴ JUL ⁶³	
SORT ⁰	ENT•B ¹ • ⁷⁷	SET TBL INDEX
SORT ⁵	ENT• <u>W</u> (ZIP+(C ¹))	
SORT ²	<u>RP</u> •B ¹ •BACK	
	COM•Q•W(ZIP ⁻¹ +B ¹)• <u>YMOST</u>	FIND LARGER NO
	JP• <u>SNORT</u> ¹	
<u>SORTNUMBER</u> ⁶	ENT•A•W(ZIP+B ⁷)	
	STR•Q• <u>H</u> (ZIP+B ⁷)	STR SMALLER NO
	RSH•AQ• ³⁶	LARGER NO TO Q

SORT' JP • SORT²
STR • Q • W(ZIP-B¹)
BJP • B¹ • (SORT³)
SORT JP • S¹ RT⁰ • STOP

REDUCE INDEX
END OF PROGRAM

Resulting Error Printout -

LABEL EXCEEDS 10C
SORTNUMBER +0

W(ZIP+C1) - INC FORMAT
- ILL R DES
00002 • ENT • W(ZIP+C1)

RP - ILL OPRTR
00003 • RP • B1 • BACK

YMOST - ILL J DES
00004 • COM • Q • W(Z-1+B1) • YMOST

H - ILL K DES
00007 • STR • Q • H(ZIP+B7)

- ILL B DES
00012 • STR • Q • W(ZIP-B1)

(SORT3) - NO K DES
00013 • BJP • B1 • (SORT3)

UNALLOCATED TAGS
00005 SNORT1 00000
00014 S1RT0 00000

ASSEMBLER STATUS MESSAGES

A number of console initiated steps starts assembler operation. The on-line typewriter relays pertinent information to the operator, in the form of *Assembler Status Messages* during the assembly run. With this information he is constantly aware of assembler action and assembly run status.

Examples:

```
SELECT OUTPUT
OUTPUT NOT POSSIBLE
SET KEY 1 IF DEBUG AIDS DESIRED    NO
SET KEY 1 IF DEBUG AIDS DESIRED    YES
CHECK SUM ERROR*
GENERATOR ERRORS
FORMAT ERROR
```

PROGRAM IDENTIFICATION

NO. 1 HEADER, NO. OF INSTRUCTIONS, AND AREA FORMED BY ALLOCATION: The on-line typewriter types output No. 1 at the end of each assembly run when requested; it is also produced automatically whenever any assembled program L2 output is requested. This output identifies 1) the assembled program, 2) the number of program operations, and 3) the storage address range formed in the assembled program.

Example:

```
COUNTONES           SMITH•1OCT1963
NO. OF INSTRUCTIONS  13
OUTPUT 1
50000 THRU 50012
```

* For explanation of Check Sum, refer to the Glossary in the Appendix.

PAPER TAPE OUTPUT

Output on paper tape is punched by the on-line High-Speed Punch in standard bioctal or keyboard printer code. The four categories of paper tape output and their output types are as follows:

1. Edited Data on Paper Tape:

NO. 5 L1 ID AND L1 PROGRAM
NO. 6 L1 ID AND LABELS

2. Assembled Object Program on Paper Tape:*

NO. 10 ASSEMBLED OBJECT PROGRAM IN BIOCTAL
NO. 12 ABSOLUTE ASSEMBLED OBJECT PROGRAM
NO. 22 ABSOLUTE ASSEMBLED OBJECT PROGRAM, L1 PROGRAM AND NOTES
NO. 24 RELATIVE ASSEMBLED OBJECT PROGRAM

3. Corrected L_0 on Paper Tape:

NO. 26 CORRECTED L_0
NO. 27 CORRECTED L_0 (SELECTIVE-BETWEEN L1 ID'S)

4. Allocation Data on Paper Tape:

NO. 30 LABELS AND ADDRESSES
NO. 31 SIGNIFICANT LABELS AND ADDRESSES
NO. 32 NUMERICALLY ORDERED ADDRESSES WITH LABELS
NO. 33 ALPHABETICALLY ORDERED LABELS AND ADDRESSES
NO. 34 RELATIVE ALLOCATION, LABELS AND ADDRESSES (base address should
 be allocated to 00000)

Paper tape output of categories 2, 3, and 4 serves two purposes: 1) as computer input (either to AS-1 or as a running program); 2) as additional edited data to that of category 1.

EDITED DATA ON PAPER TAPE

A hard copy printout of certain data is frequently desired for editing purposes. The assembler produces paper tape output which is thereafter listed to obtain the edited record.

*May be loaded via the AS-1 Utility Package

NO. 5 L_1 IDENTIFIERS AND L_1 PROGRAM: Paper tape output in this format presents a sequential list of L_1 identifiers, their corresponding L_1 operations, and labels. This output does not include notes.

Example:

0.	COUNTONES	PROGRAM•SMITH•10OCT63
1.	CTONES0	CL•B2
2.	CTONES1	ENT•B1•35
3.		CL•A
4.		ENT•Q•W(WORD0 + B2)
5.	CTONES2	LSH•Q•1•QPOS
6.		ADD•A•1
7.		BJP•B1•CTONES2
10.		STR•A•W(SUM0 + B2)
11.		BSK•B2•NWORDS
12.		JP•CTONES1•STOP5
13.	CTONES3	JP•CTONES3•STOP
..		

NO. 6 L₁ IDENTIFIERS AND LABELS: Paper tape output in this format consists of all program labels and their corresponding L₁ identifiers. No other L₁ identifiers are given.

Example:

COUNTONES L'ID	SMITH•10OCT63 LABEL
0.	COUNTONES
1.	CTONES0
2.	CTONES1
5.	CTONES2
13.	CTONES3
..	

ASSEMBLED OBJECT PROGRAMS ON PAPER TAPE

AS-1 presents assembled object programs, with or without additional data, on punched paper tape. The programmer uses these output tapes for two purposes: 1) as program input to the Unit Computer, and 2) as an intermediate medium to obtain a record for editing purposes. The AS-1 Utility Package reloads tapes of this category when used as program input to the computer. Listings of these tapes provide hard copies for editing purposes.

The large variety of output formats provides assembled object programs to best fit the programmer's need. It gives an option of absolute addressing or relative addressing in bioctal format. Absolute addressing assigns a specific storage address to each instruction. Relative addressing allows the programmer to specify a starting address at the time he loads the assembled program. Bioctal format is an abbreviated means of paper tape storage where each tape frame represents two octal characters, thus reducing the size of tapes considerably. Regardless of the format selected, the tapes include the necessary control codes and check sums for AS-1 Utility Package loading. In addition to the options heretofore mentioned, the programmer has the option of requesting additional data with the assembled program such as the items in L₁ language and notes. The inclusion of these in no way interferes with the loading of assembled programs into the computer.

NO. 10 ASSEMBLED PROGRAM IN BIOCTAL: Paper tape output in this format consists of the assembled program in bioctal code. The program is preceded by a 76 code and followed by check sums for AS-1 Utility Package loading.

Note: Punched paper tape in bioctal format cannot be listed or reproduced on a typewriter.

NO. 12 ABSOLUTE ASSEMBLED PROGRAM IN KEYBOARD PRINTER CODE. Paper tape output in this format consists of absolute-addressed machine code instructions in standard code. An 88 code precedes and check sums follow the program for AS-1 Utility Package loading.

COUNTONES SMITH•10OCT63
NO. OF INSTRUCTIONS 13
OUTPUT 12
50000 THRU 50012

88

50000 12200 00000
50001 12100 00035
50002 11000 00000
50003 10032 50030
50004 05200 00001
50005 20000 00001
50006 72100 50004
50007 15032 50100

50010 71200 00011
50011 61500 50001
50012 61400 50012

..
00004 36164
00003 10217

NO. 22 ABSOLUTE ASSEMBLED OBJECT PROGRAM, L₁ PROGRAM, AND NOTES: Paper tape output in this format consists of absolute-addressed machine code instructions, their corresponding L₁ operations (usually mnemonic), and notes. The tape begins with an 88 code and ends with check sums for AS-1 Utility Package loading.

Note: The check sums represent the assembled program only.

Example:

COUNTONES SMITH•10OCT63
 NO. OF INSTRUCTIONS 13
 OUTPUT 22
 50000 THRU 50012

50000	12200	00000	CTONES0	CL•B2	SET WORD INDEX
50001	12100	00035	CTONES1	ENT•B1•35	SET SHIFT INDEX
50002	11000	00000		CL•A	SET SUM0 TO ZERO
50003	10032	50030		ENT•Q•W(WORD0+B2)	
50004	05200	00001	CTONES2	LSH•Q•1•QPOS	TEST EACH BIT FOR 0 OR 1
50005	20000	00001		ADD•A•1	INCREASE SUM IF 1 FOUND
50006	72100	50004		BJP•B1•CTONES2	
50007	15032	50100		STR•A•W(SUM0+B2)	SUM STORAGE
50010	71200	00011		BSK•B2•NWORDS	
50011	61500	50001		JP•CTONES1•STOP5	CONTINUE COMPUTING SUMS
50012	61400	50012	CTONES3	JP•CTONES3•STOP	END
00004	36164				
00003	10217				

NO. 24 RELATIVE BIOCTAL ASSEMBLED OBJECT PROGRAM: Paper tape output in this format consists of the assembled program in bioctal code. The program tape begins with a 75 code and ends with the check sum. The other codes are assembled program data and relative addressing core storage codes.

CORRECTED L₀ ON PAPER TAPE

NO. 26 CORRECTED L₀: Output in this format is a complete dump of L₁ program operations, including notes, on paper tape. This output provides a corrected L₀ tape when corrections have been made to the original program with the L₁ corrector. The programmer may also request this output without corrections having been made.

The assembler treats the corrected L₀ tape as normal program input. No initial code appears on the tape since the AS-1 loading routine reloads the tape; however, a check sum appears at the end for verification or reloading.

NO. 27 CORRECTED L₀ (Selective - between L₁ ID's): Output in this format is a *selective* dump of L₁ program operations, including notes, on paper tape. The programmer selects, by means of L₁ identifiers, a portion or portions of L₁ program storage for paper tape output. The programmer specifies to the operator the initial and final L₁ identifiers of the area(s) to be dumped.

Output No. 27 is most frequently used to obtain a corrected L₀ input tape of a particular portion, or portions, of L₁ program storage when corrections are made to the original program with the L₁ corrector. The programmer can request this partial output without corrections.

AS-1 accepts the corrected L₀ tape as normal program input to the assembler. No initial code is placed on the tape since reloading is done by the AS-1 loading routine.

ALLOCATION DATA ON PAPER TAPE

NO. 30 LABELS AND ADDRESSES: Paper tape output in this format consists of all program labels and their corresponding addresses as they appear in the assembled program. This tape is acceptable as an allocation input tape.

Example:

```
COUNTONES          ALLOCATION•SMITH•10OCT 63

NWORDS      00011
WORD0       50030
SUM0        50100
CTONES0     50000
CTONES1     50001
CTONES2     50004
CTONES3     50012
```

..

No. 31 SIGNIFICANT LABELS AND ADDRESSES: Paper tape output in this format consists of a selection of all *significant labels* and their corresponding addresses within an assembled set of operations. [*Significant labels* accompany operations (mnemonic or numeric) that infer the beginning of a subroutine or a program segment.] In addition, it includes all labels and corresponding addresses of U-TAG or EQUALS operations.

Subroutines or program segments assembled for the Unit Computer will, in general, begin with *one* of three instructions:

```
61 j00 00000   (j may be any value)
60 100 00000
00 000 00000
```

The assembler searches for the above instructions and the operators U-TAG and EQUALS in presenting this output. Only labeled operations are included in this category.

NO. 32 NUMERICALLY ORDERED ADDRESSES WITH LABELS: Paper tape output in this format consists of all labels and their corresponding addresses in numerical order of the addresses. This is acceptable as an allocation tape.

NO. 33 ALPHABETICALLY ORDERED LABELS AND ADDRESSES: Paper tape output in this format consists of all labels and their corresponding addresses in alphabetical order of the labels. This is acceptable as an allocation tape.

NO. 34 RELATIVE ALLOCATION, LABELS AND ADDRESSES: Paper tape output in this format consists of labels and their corresponding addresses given relative to zero.

ASSEMBLER - CONTROL OPERATIONS

(A-CONTROL)

To assure himself a maximum of control over the assembling process via tape(s), a programmer may group several AS-1 input operations under one **A-CONTROL** header. A label and identifying operands may be used with the **A-CONTROL** header, but the assembler does not require them.

[label] → **A-CONTROL** • [programer's name] • [date] →

Subordinate to the **A-CONTROL** header are minor headers followed by their respective operations. So-called independent operations also follow. The **A-CONTROL** header merely categorizes selected input information to the assembler under a single header. Its use is optional, however, since each minor header with its respective operations and each independent operation can be used independently. Operations following an **A-CONTROL** header specify various types of input control data such as debugging aids, **CHAN-SET**, etc. Operations need not all appear on the same tape. (See AS-1 INPUT.)

Minor Header Operations. Several types of input operations must follow a descriptive minor header, such as: 1) **ALLOCATION**, 2) **INDR-ALLOC** (indirect allocation), 3) **REL-ALLOC** (relative allocation), 4) **CHAN-SET** (channel setting), and 5) **EXCHANGE**. Allocation operations must be grouped following an **ALLOCATION** header, indirect-allocation operations must be grouped following an **INDR-ALLOC** header, etc. When used with the **A-CONTROL** operation, minor header operations do not require a label or identifying operands. This does not preclude their use, however, if the programmer wishes to use them. Each of the minor headers with its respective group of operations, may, of course, be used independently without **A-CONTROL**, in which case normal header usage is followed. (See AS-1 INPUT.)

Independent Operation: This input operation does not require a descriptive header since it is an independent operation by itself, and like minor header operations, may be used with or without an **A-CONTROL** header. The **DEBUG-AIDS** operator is an independent operation.

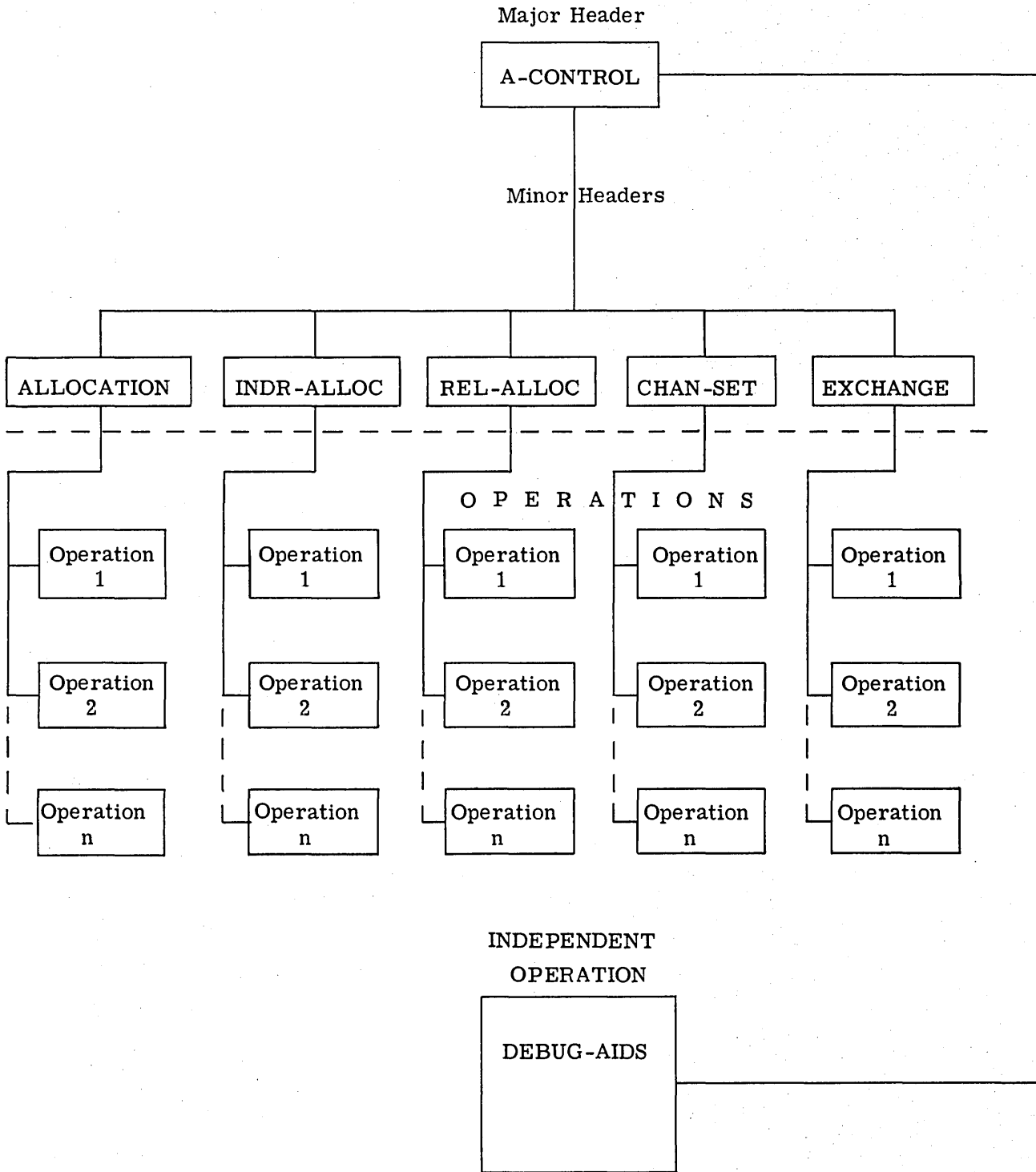
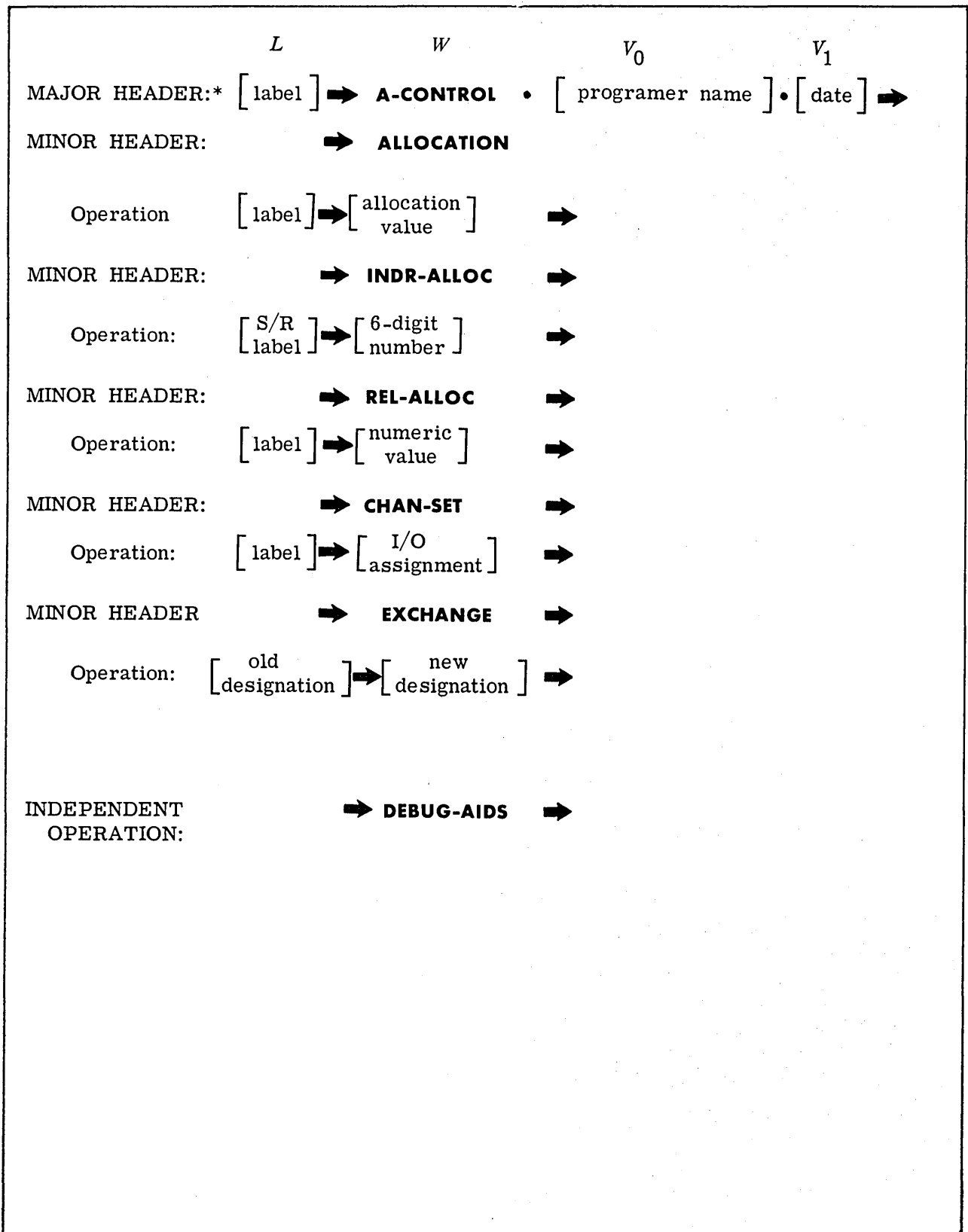


Figure 1. Typical **A-CONTROL** Headers and Operations



* Label and operands for the **A-CONTROL** header are optional.

Figure 2. **A-CONTROL** Header and Operation Formats

Assembler- CONTROL Operation:

L W V_0 V_1
[label] ➔ **A-CONTROL** • [programer's name] • [date] ➔

The **A-CONTROL** header operation indicates that assembler-control operations are to follow. These may be minor headers with their related operations and/or independent operations. The **A-CONTROL** header appears as the first operation on the first tape containing assembler-control operations. A label (the program name) and two identifying operands (the programer's name and the date in that order) may be used with the **A-CONTROL** header, but they are not required (see examples, AS-1 INPUT).

- L - when used, gives the name of the associated program
- V_0 - when used, gives the name of the programer
- V_1 - when used, gives the date (usually the date of preparation)

Examples:

- a) **SIMPLEX** ➔ **A-CONTROL** • **SMITH** • **JAN 1963** ➔
- b) ➔ **A-CONTROL** ➔

CHANNEL-SET Operation:

$$\begin{array}{ccccccc} & L & & W & & V_0 & & V_1 \\ [label] & \Rightarrow & \text{CHAN-SET} & \bullet & [\text{programer's name}] & \bullet & [\text{date}] & \Rightarrow \end{array}$$

The **CHAN-SET** operation, a minor header, precedes operations which provide input or output *assignments*. **CHAN-SET** and its succeeding operations permit programs to be written with symbolic input/output channel symbols. By holding the assignment of external equipment open until needed, the programmer can, at that time, determine which of the specific channels are available for use. He then replaces the symbolic representations with the actual assignments, using the Channel-Set feature. Thus the programmer can assign an external equipment to any permissible input, output, or function channel. The operations which may contain replaceable operands include: **JP**, **STR**, **IN**, **OUT**, **TERM**, **EX-COM**, **EX-COM-MW**, **SIL-EX**, and **RIL-EX**.

The **CHAN-SET** header and *assignment* operations are applicable only to input/output assignments. It cannot be used interchangeably with **EQUALS**. The header and its succeeding operations provide the basic capabilities of the **MEANS** operation in a more convenient format.

Used under **A-CONTROL**, the **CHAN-SET** header does not require a label or identifying operands. However, operations which follow this header must have labels. The **CHAN-SET** header may be used independently of the **A-CONTROL** header.

Channel-Set *assignment* operations appear as follows:

$$\begin{array}{ccc} & L & & W \\ [label] & \Rightarrow & [\text{input/output assignment}] \end{array}$$

- L - gives the symbolic input/output name to be replaced
- W - states the specific input/output assignment to be substituted for L . Entries in the W position are presently restricted to information pertaining to input/output specifications

Example **CHAN-SET** header and assignment operations:

- a) **SWAP** ➔ **CHAN-SET • BEVYOUNG • 18JAN63**
- b) **COB** ➔ **C10**
- c) **BUDDY** ➔ **C1**
- d) **ACE** ➔ **C11**

Examples of Input/Output operations with which the foregoing examples may be used:

- a) ➔ **EX-COM • COB • 513**
- b) ➔ **STR • BUDDY • W(POCKET)**
- c) ➔ **TERM • ACE • OUTPUT**

Note: The **CHAN-SET** header and its succeeding assignment operations are *never* permitted within the L_0 *input program*. They are *not* correctable by use of the L_1 corrector, because they never appear in the assembler's L_1 table.

DUMP-AREA Operation:

W $-V_0-$ V_1^*

➔ **DUMP-AREA** • [area name(s)] • [key set condition] ➔

The **DUMP-AREA** operation causes a printout of all non-zero words of the debugging area(s) requested. The printout also gives debugging area names and their absolute limits.

$-V_0-$ - names the debugging area(s) to be dumped. (See Appendix, Glossary, OPER- AND POSITION)

V_1^* - specifies a key set condition. Key settings and their significance are as follows:

Perform operation if V_1 is **KEY1** and console key 1 is set.

Perform operation if V_1 is **KEY2** and console key 2 is set.

Perform operation if V_1 is **KEY3** and console key 3 is set.

Typical **DUMP-AREA** printout:

```
ENT ADD 60056

WS36 61200 - 61210

61202 00006 00000
61203 00002 00000
61206 00007 00000
61207 00014 00000
```

*Operand use is optional

TEST-IMAGE Operation:

W $-V_0-$ V_1^*

➔ **TEST-IMAGE** • [area name(s)] • [key set condition] ➔

The **TEST-IMAGE** operation tests the specified areas for changes since they were last imaged. It makes a word-by-word comparison between the debugging areas and their corresponding images. It then causes a printout for each non-comparing position of the image word, the area word, the area word address, and the increment of this address in reference to the initial area address. (See Example Debugging Aids.) Thereafter, a new image is made.

$-V_0-$ - names of area(s) to be tested. (See Appendix, Glossary, OPERAND POSITION)

V_1^* - specifies a key set condition. Key settings and their significance are as follows:

Perform operation if V_1 is **KEY1** and console key 1 is set.

Perform operation if V_1 is **KEY2** and console key 2 is set.

Perform operation if V_1 is **KEY3** and console key 3 is set.

*Operand use is optional

Example Debugging Aids illustrating image testing:

Operations: **DEF-AREA**

CORE-IMAGE

TEST-IMAGE

```

*   ➔   DEF-AREA • TNP • WS3 +10 • 100
*   ➔   DEF-AREA • LTS • LTS • 55
**  ➔   CORE-IMAGE • TNP • 02000
**  ➔   CORE-IMAGE • LTS • 02000
***  ➔   TEST-IMAGE • TNP • LTS • KEY3
  
```

Each word of **TNP** and **LTS** is compared to its corresponding word in the image.

If the words are not equal, the image word and the present program word are punched (printed) out. (See sample printout on the following page.)

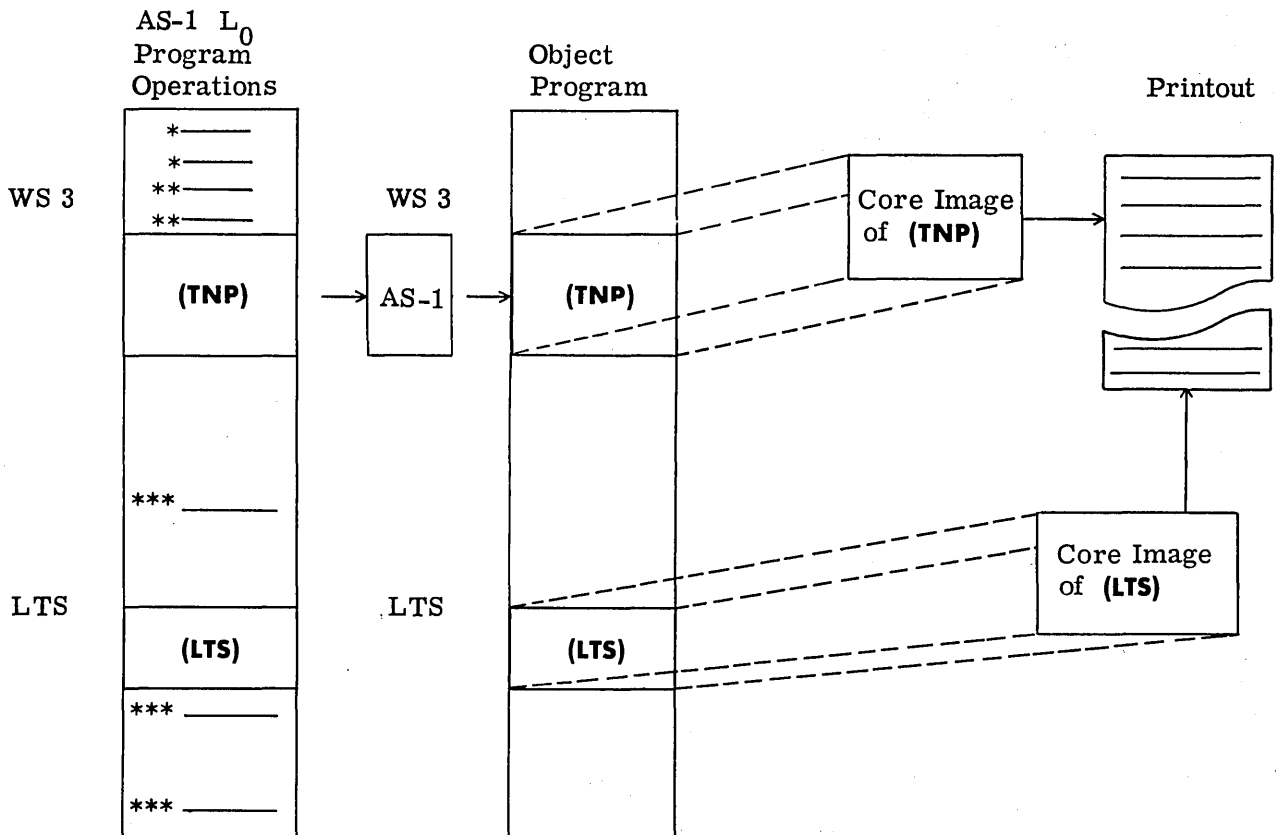


Figure 3. Example Debugging Aids

Typical **TEST-IMAGE** Printout:

A 02530 33330 Q 00000 22764

B¹ 00010 B² 07774 B³ 00022 B⁴ 00040 B⁵ 02040 B⁶ 00000 B⁷ 00012

ENT ADD 50000

TNP 60012 - 60112

address	image	program
+0 60012	00045 00063	00047 00064
+45 60057	00000 00113	00000 00000
+46 60060	00000 00114	00000 00000
+63 60075	00000 00000	00000 00031

LTS 70133 - 70210

address	image	program
+12 70145	00000 70137	12100 00145
+23 70156	77321 00010	77431 00010
+42 70175	11121 00011	71121 00011

INTRODUCTION

The 1230 Computer may be connected to a variety of military or commercial peripheral equipments.

These include:

- Teletype Printer Units
- Magnetic Tape Units
- High-Speed Printer Units
- Card Read/Punch Units
- Display and Display Interface Equipment
- Radar and Radar Adaption Interfaces
- Paper Tape Units
- Manual Entry Devices

UNIVAC 1232 INPUT/OUTPUT CONSOLE

BASIC INFORMATION

The UNIVAC 1232 Input/Output Console (Figure 1) has a paper tape punch and reader as standard equipment, with a keyboard and printer as an option. Input and output devices communicate with the computer through a single input/output channel. See Figure 2.

ON-LINE OPERATION

In the on-line operation the Input/Output Console provides means for entering data into the computer by punched tape or an alphanumeric keyboard. It provides means for recording output data from the computer by either punching tape or printing on paper media or both simultaneously.

OFF-LINE OPERATION

In the off-line operation the Input/Output Console provides means to:

- Print on paper media by Keyboard entry
- Perforate tape by Keyboard entry
- Perforate tape and print on paper media simultaneously by keyboard entry
- Print on paper media from a perforated tape
- Perforate tape from a perforated tape
- Perforate tape and print on paper media simultaneously from a perforated tape

INPUT-OUTPUT CONTROL

The input/output sequences are manually enabled from the control panel or automatically enabled by the computer program.

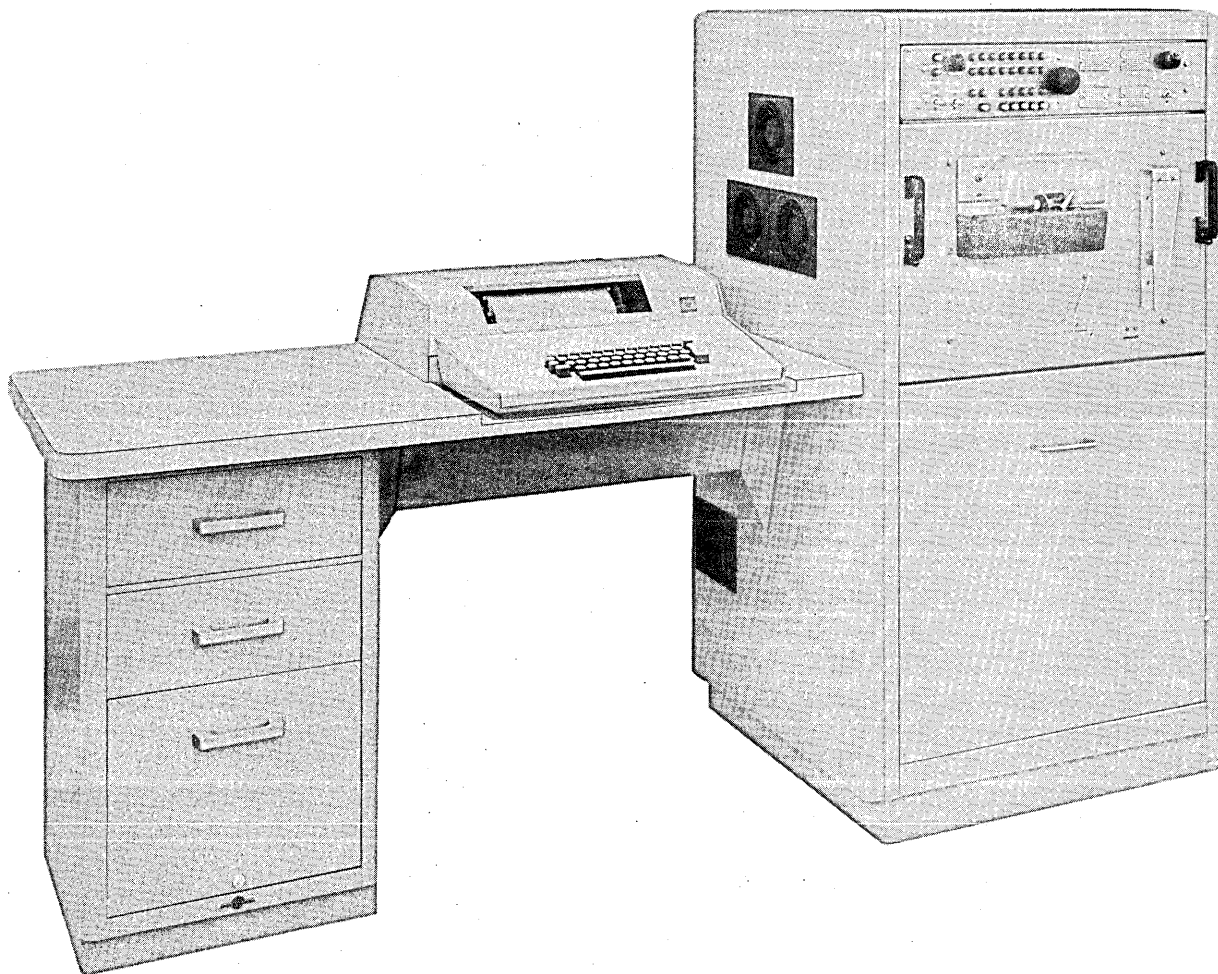


Figure 1. UNIVAC 1232A Input/Output Console

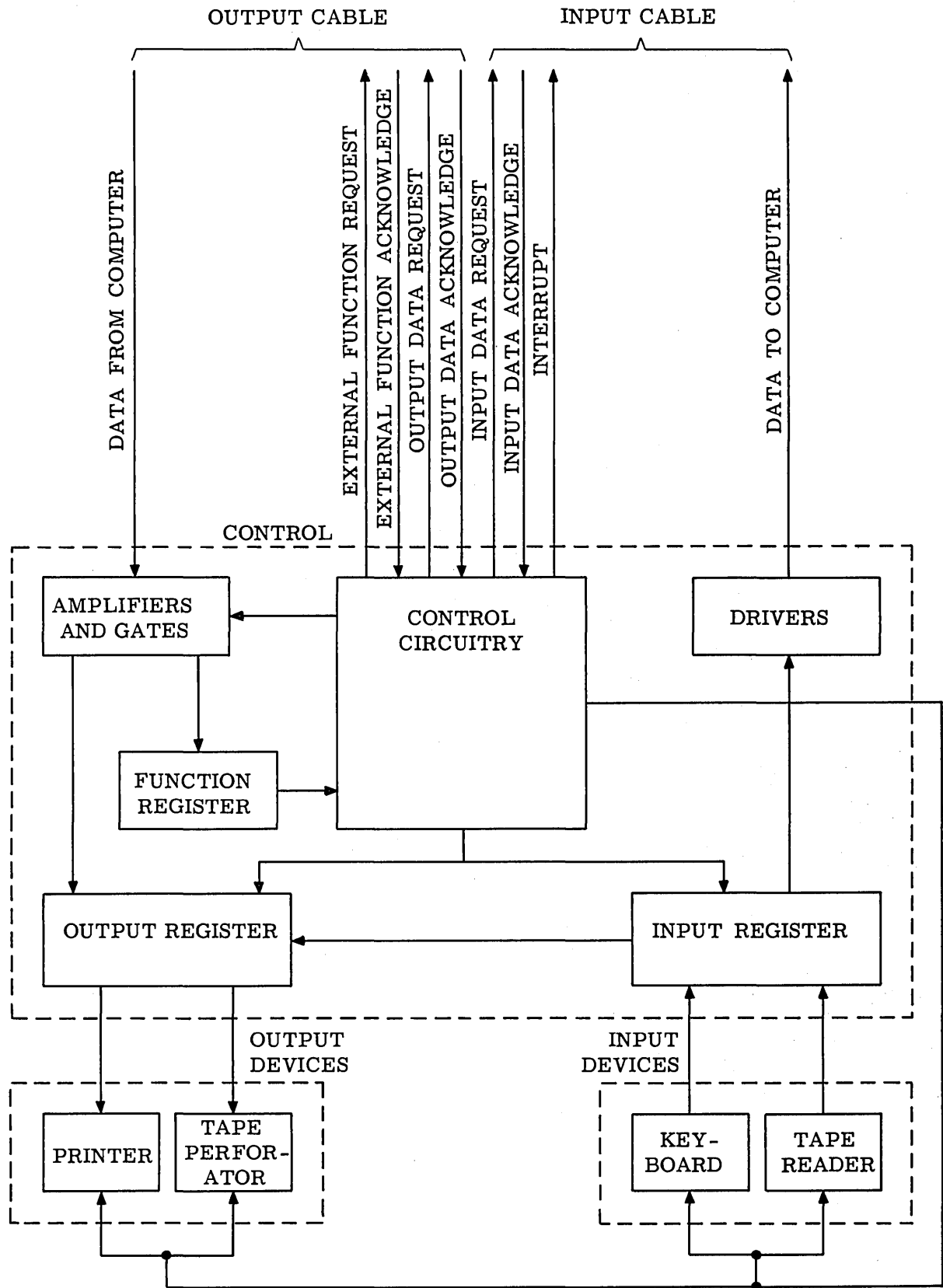


Figure 2. Block Diagram of Console

COMPUTER CONTROL

The computer controls the Input/Output Console through the external-function word, as specified in Figure 3, as follows:

Bits 0, 1, and 2 control the output devices. A "one" in bit 0 allows the status of the printer (bit 1) and perforator (bit 2) to be controlled by the information in bits 1 and 2. A "zero" in bit 0 causes bits 1 and 2 to be ignored and the status of the output devices to remain unchanged. With a "one" in bit 0; a "one" in bit 1 enables the printer and a "zero" in bit 1 disables the printer, and a "one" in bit 2 enables the perforator and a "zero" in bit 2 disables the perforator.

Bits 3, 4, 5, and 6 control the input devices. A "one" in bit 3 allows the status of the keyboard (bit 4) and reader (bits 5 and 6) to be controlled by the information in bits 4, 5, and 6.

A "zero" in bit 3 causes bits 4, 5, and 6 to be ignored and the status of the input devices remain unchanged. With a "one" in bit 3; a "one" in bit 4 enables the keyboard and a "zero" in bit 4 disables the keyboard, a "one" in bit 5 enables the reader and a "zero" in bit 5 disables the reader, and a "one" in bits 5 and 6 starts the reading operation and a "zero" in bits 5 or 6 stops the reading operation.

The status of the Input/Output Console is determined by the latest external-function word.

PANEL CONTROL

The computer external-function words are manually duplicated by the operation of the control panel switches specified in Table I.

TABLE I. MANUAL-AUTOMATIC CONTROLS

UNIT(S) CONTROLLED	EXTERNAL FUNCTION- WORD BIT	CONTROL PANEL SWITCHES	
		SET	CLEAR
Output Devices	0	None	None
Printer	1	Print	Print Clear
Perforator	2	Punch	Punch Clear
Input Devices	3	None	None
Keyboard	4	Keyboard	Keyboard Clear
Reader	5	Read	Read Clear
Reader	6	Start-Read	Start-Read Clear
	7 not used		

29	7	6	5	4	3	2	1	0
NOT USED								0 Disable output 1 Enable output
							0 Disable printer 1 Enable printer	
						0 Disable punch 1 Enable punch		
				0 Disable input 1 Enable input				
				0 Disable keyboard 1 Enable keyboard				
			0 Disable reader 1 Enable reader					
			0 Stop reading operation if $i_5 = 0$ 1 Start reading operation if $i_5 = 1$					

Figure 3. 1232 Input/Output Console, External Function Word

OPERATION OF UNITS

PERFORATED TAPE READER

The perforated tape reader is adjustable to read chad-type tape with 5, 6, 7, or 8 channels and widths of 11/16 inch, 7/8 inch, or 1 inch. The reader reads tape at a rate of 300 frames per second. The tape is transported through the reader by an electric motor drive with a pinch roller and a brake. The following sequence is typical:

1. The reader is enabled and the motor attains operating speed
2. Tape is placed in the reader and the START READ indicator-switch is operated
3. If a sprocket hole of the tape is positioned over the sensor, no advancement of the tape shall occur; if the tape is positioned so that the sensor is between sprocket holes, the clutch shall be engaged and the tape will be advanced
4. The next sprocket hole that reaches the sensor actuates the brake and the tape stops
5. The signal caused by the data holes in each frame sets the corresponding input lines through the action of the input register
6. The signal caused by the sprocket hole causes the control circuitry to set the input-data-request line
7. The computer responds with an input-data-acknowledge signal, which indicates that the input-data lines have been sampled. The control circuitry clears the input-data-request line, clears the input-data lines, and engages the clutch to advance the tape

Steps 4 through 7 are repeated until operation of the reader is stopped.

TAPE PERFORATOR

The tape perforator perforates chad-type tape. The tape perforator is adjustable to perforate 5, 6, 7, or 8 channels on 11/16 inch, 7/8 inch or 1 inch tape. It perforates 10 frames per inch at a tape speed of 11 inches per second. The tape is transported through the perforator by an electric motor drive. The following sequence is typical:

1. The perforator is enabled
2. The control circuitry sets the output-data-request line
3. The computer, in synchronism with internal priorities, detects the output-data-request signal

4. The computer places data on the output line
5. The computer sets the output-acknowledge line
6. The control circuitry detects the output-acknowledge signal, gates the data on the output-data lines to the output register, and clears the output-data-request line
7. A magnetic head associated with the perforator drive mechanism generates a pulse at the appropriate time to gate the output register content to the tape perforator. This energizes the perforating mechanism while the tape is stopped.
8. The control circuitry generates a pulse that de-energizes the perforating mechanism, clears the output register, and sets the output-data-request line

Steps 3 through 8 are repeated until perforator operation is stopped.

PRINTER

The printer prints data, one character at a time, on paper media. The printer prints a character corresponding to the field data code as specified in Table II. The printer can print 10 characters per second. The printout has 10 characters per inch horizontally, 72 characters per line, and 6 lines per inch vertically. The following sequence is typical:

1. The printer is enabled
2. The control circuitry sets the output-data-request line
3. The computer, in synchronism with internal priorities, detects the output-data-request signal
4. The computer places data on the output data lines
5. The computer sets the output-acknowledge line
6. The control circuitry detects the output-acknowledge signal, gates the data on the output-data lines to the output register, and clears the output-data-request line
7. The control circuitry causes the printer to perform the print or control function indicated by the data bits in the register
8. Upon completion of the print function the control circuitry clears the output register and sets the output-data-request line

Steps 3 through 8 are repeated until operation of the printer is stopped.

TABLE II. FIELD DATA CODE

SYMBOL OR FUNCTION	KEYBOARD SYMBOL	PRINTED SYMBOL	SIGNALS ON DATA LINES						OCTAL CODE
			2 ⁵	2 ⁴	2 ³	2 ²	2 ¹	2 ⁰	
Master Space*		⸍	0	0	0	0	0	0	00
Upper Case	UC	~	0	0	0	0	0	1	01
Lower Case	LC	⸍	0	0	0	0	1	0	02
Line Feed	LF		0	0	0	0	1	1	03
Carriage Return	RETURN		0	0	0	1	0	0	04
Space			0	0	0	1	0	1	05
A	A	A	0	0	0	1	1	0	06
B	B	B	0	0	0	1	1	1	07
C	C	C	0	0	1	0	0	0	10
D	D	D	0	0	1	0	0	1	11
E	E	E	0	0	1	0	1	0	12
F	F	F	0	0	1	0	1	1	13
G	G	G	0	0	1	1	0	0	14
H	H	H	0	0	1	1	0	1	15
I	I	I	0	0	1	1	1	0	16
J	J	J	0	0	1	1	1	1	17
K	K	K	0	1	0	0	0	0	20
L	L	L	0	1	0	0	0	1	21
M	M	M	0	1	0	0	1	0	22
N	N	N	0	1	0	0	1	1	23
O	O	O	0	1	0	1	0	0	24
P	P	P	0	1	0	1	0	1	25
Q	Q	Q	0	1	0	1	1	0	26
R	R	R	0	1	0	1	1	1	27
S	S	S	0	1	1	0	0	0	30
T	T	T	0	1	1	0	0	1	31
U	U	U	0	1	1	0	1	0	32
V	V	V	0	1	1	0	1	1	33
W	W	W	0	1	1	1	0	0	34
X	X	X	0	1	1	1	0	1	35
Y	Y	Y	0	1	1	1	1	0	36

TABLE II. FIELD DATA CODE (Cont.)

SYMBOL OR FUNCTION	KEYBOARD SYMBOL	PRINTED SYMBOL	SIGNALS ON DATA LINES						OCTAL CODE
			2 ⁵	2 ⁴	2 ³	2 ²	2 ¹	2 ⁰	
Z	Z	Z	0	1	1	1	1	1	37
)))	1	0	0	0	0	0	40
-	-	-	1	0	0	0	0	1	41
+	+	+	1	0	0	0	1	0	42
<	<	<	1	0	0	0	1	1	43
=	=	=	1	0	0	1	0	0	44
>	>	>	1	0	0	1	0	1	45
_	_	_	1	0	0	1	1	0	46
\$	\$	\$	1	0	0	1	1	1	47
*	*	*	1	0	1	0	0	0	50
(((1	0	1	0	0	1	51
"	"	"	1	0	1	0	1	0	52
:	:	:	1	0	1	0	1	1	53
?	?	?	1	0	1	1	0	0	54
!	!	!	1	0	1	1	0	1	55
,	,	,	1	0	1	1	1	0	56
Stop	Ⓢ	Ⓢ	1	0	1	1	1	1	57
0	0	0	1	1	0	0	0	0	60
1	1	1	1	1	0	0	0	1	61
2	2	2	1	1	0	0	1	0	62
3	3	3	1	1	0	0	1	1	63
4	4	4	1	1	0	1	0	0	64
5	5	5	1	1	0	1	0	1	65
6	6	6	1	1	0	1	1	0	66
7	7	7	1	1	0	1	1	1	67
8	8	8	1	1	1	0	0	0	70
9	9	9	1	1	1	0	0	1	71
'	'	'	1	1	1	0	1	0	72
;	;	;	1	1	1	0	1	1	73
/	/	/	1	1	1	1	0	0	74
.	.	.	1	1	1	1	0	1	75
Special	SPEC	□	1	1	1	1	1	0	76
Idle	↑	↑	1	1	1	1	1	1	77

*Master space indicates an absence of information

KEYBOARD

The keyboard, Figure 4, generates the data codes in Table II when corresponding labeled keys are operated. Data entered into the keyboard is simultaneously printed by the printer if the printer and the copy mode are enabled. The following sequence is typical:

1. The keyboard is enabled
2. When a key is operated, the corresponding input-data lines are set through the action of the input register
3. The control circuitry sets the input-data-request line
4. When the computer responds with an input-data-acknowledge signal, the control circuitry clears the input-data-request line and the data lines

Steps 2 through 4 are repeated each time a key is depressed until operation of the keyboard is stopped.

KEYBOARD INTERRUPT

The computer may be interrupted from the keyboard by the following sequence:

1. Keyboard is enabled
2. Printer and copy mode are enabled if printout of the interrupt code is desired
3. The interrupt indicator switch on the control panel has been operated
4. A keyboard key is operated which sets the corresponding field-data code on the input-data lines and generates an interrupt to the computer
5. When the computer responds with an input acknowledge the interrupt and the input-data lines will be cleared

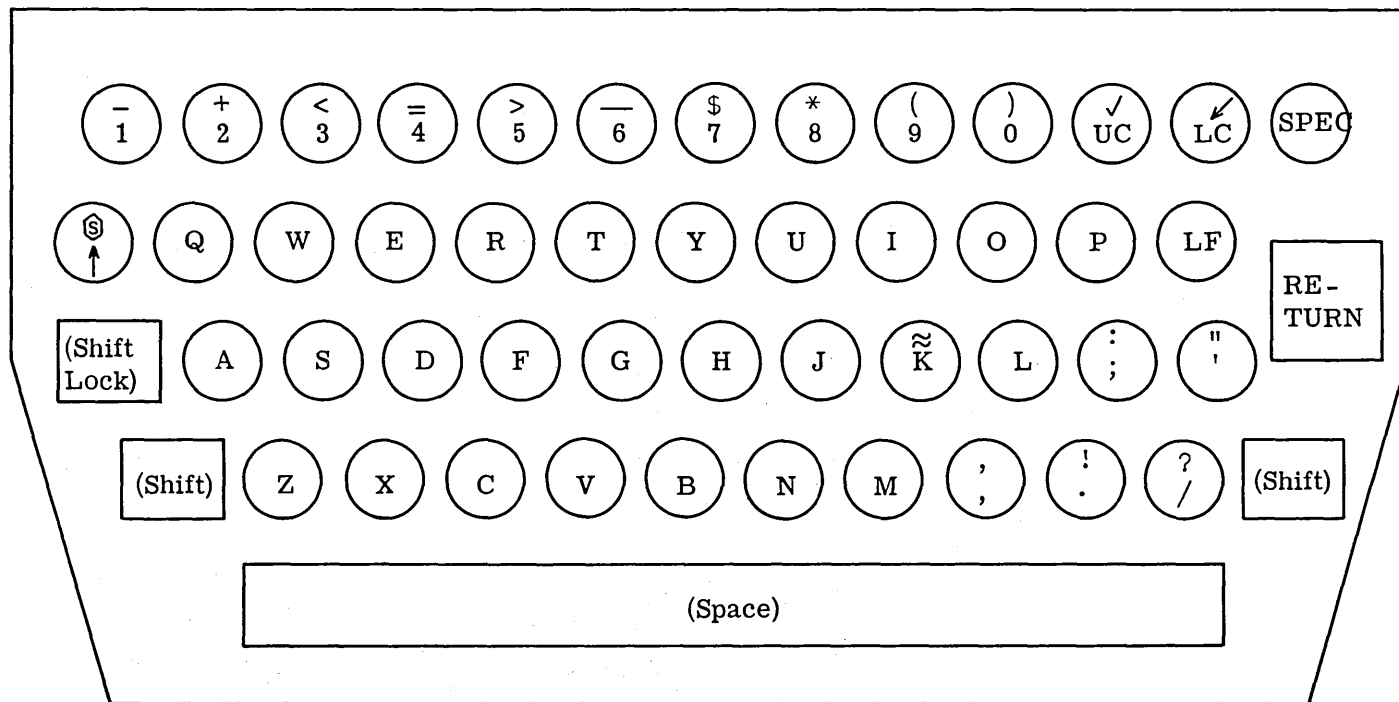
Steps 3 through 5 are repeated for each interrupt code to be sent.

SWITCHES AND INDICATORS

The switches and indicators of the Input/Output Console operate as follows:

Power Switch and Indicator

The power switch switches the input power on and off. The power indicator lights whenever the power switch is in the on position.



NOTE: Functions shown in parentheses are blank.

Figure 4. Keyboard Layout

On-Line Off-Line Switch

In the on-line position the Input/Output Console operates as an input/output device for the computer as specified herein. In the off-line position the Input/Output Console operates independently of the computer and performs the off-line functions specified herein.

Tape-Feed Indicator Switch

The tape perforator generates blank tape with only the sprocket holes perforated whenever the tape-feed indicator switch is operated.

Tape-Levels Switch

The tape-levels switch disables the perforated-tape-reader levels which are not selected.

Input-Data Indicator Switches

The eight input-data indicator switches displays the data stored in the input register and enables data to be manually entered into the input register.

Output-Data Indicator Switches

The eight output-data indicator switches displays the data stored in the output register and enables data to be manually entered into the output register.

Master Clear Switch

The master-clear switch stops operation of all units of the Input/Output Console and sets all the logic to an initial state.

Interrupt Indicator Switch

The interrupt indicator-switch enables the generation of an interrupt to the computer as specified in Keyboard Interrupt.

Read, Read-One Switch

In the Read position the perforated-tape reader will read continuously. In the Read-One position the perforated tape reader will read one frame, advance to the next frame, and stop. This switch is for *off-line* operation only.

Start-Read Indicator-Switch

The Start-Read Indicator-Switch starts the perforated-tape reading operation.

Copy Indicator Switch

The copy switch enables the Input/Output Console to reproduce the data being sent to the computer by any one of the following methods:

- Print the data on paper media
- Perforate the data on tape
- Print the data on paper media and perforate the data on tape simultaneously

Copy-Clear Switch

The copy-clear switch disables the copy mode of operation.

EXTERNAL FUNCTION MANUAL CONTROLS

The print, punch, keyboard, and read indicator-switches; and the print-clear, punch-clear, keyboard-clear, and read-clear switches enable and disable their respective units as specified in the Input/Output Panel Control.

MAGNETIC TAPE SYSTEM (TYPE 1240A)

BASIC INFORMATION

The type 1240A Magnetic Tape System provides a large capacity, medium-speed, auxiliary storage area.

The system employs various format selections. They include recording and reading in four moduli, two character types, odd and even parity, and low and high density. In order to provide compatibility with the high-speed printer, the Mod 5 format is used with a programmed fixed block length of 128 lines or tape frames to each block of information. One block contains 24 computer words. Mod 6 format is used for compatibility with some non UNIVAC systems. The density selection allows the 1240A tape unit to read or write at 200 frames per inch for low density and at 556 frames per inch for high density. The reading and writing operation is performed at a tape speed of 112.5 inches per second, and the rewind operation is done at a tape speed of 225 inches per second. The block length may vary between 24 computer words and total computer memory. The recording of 1240 tape system is the non-return to zero (change-on-one) technique.

The basic 1240A Magnetic Tape System cabinet (Figure 1) consists of three sections: 1) magnetic tape control, 2) one tape transport control, and 3) four tape transports. Auxiliary tape transport controls and tape transports may be added to the system (up to sixteen transports and four tape transport controls, Figure 2).

The Magnetic Tape Control enables the Magnetic Tape System to communicate with the Computer by performing the interface digital-to-digital conversion and performing the logical operations of selecting tape function and tape transport. The tape transport control receives signals from Magnetic Tape Control and performs the logic necessary to control either two or four tape transports. Only one Magnetic Tape Control is necessary in a system and one tape transport control is necessary for each cabinet of transports. Figure

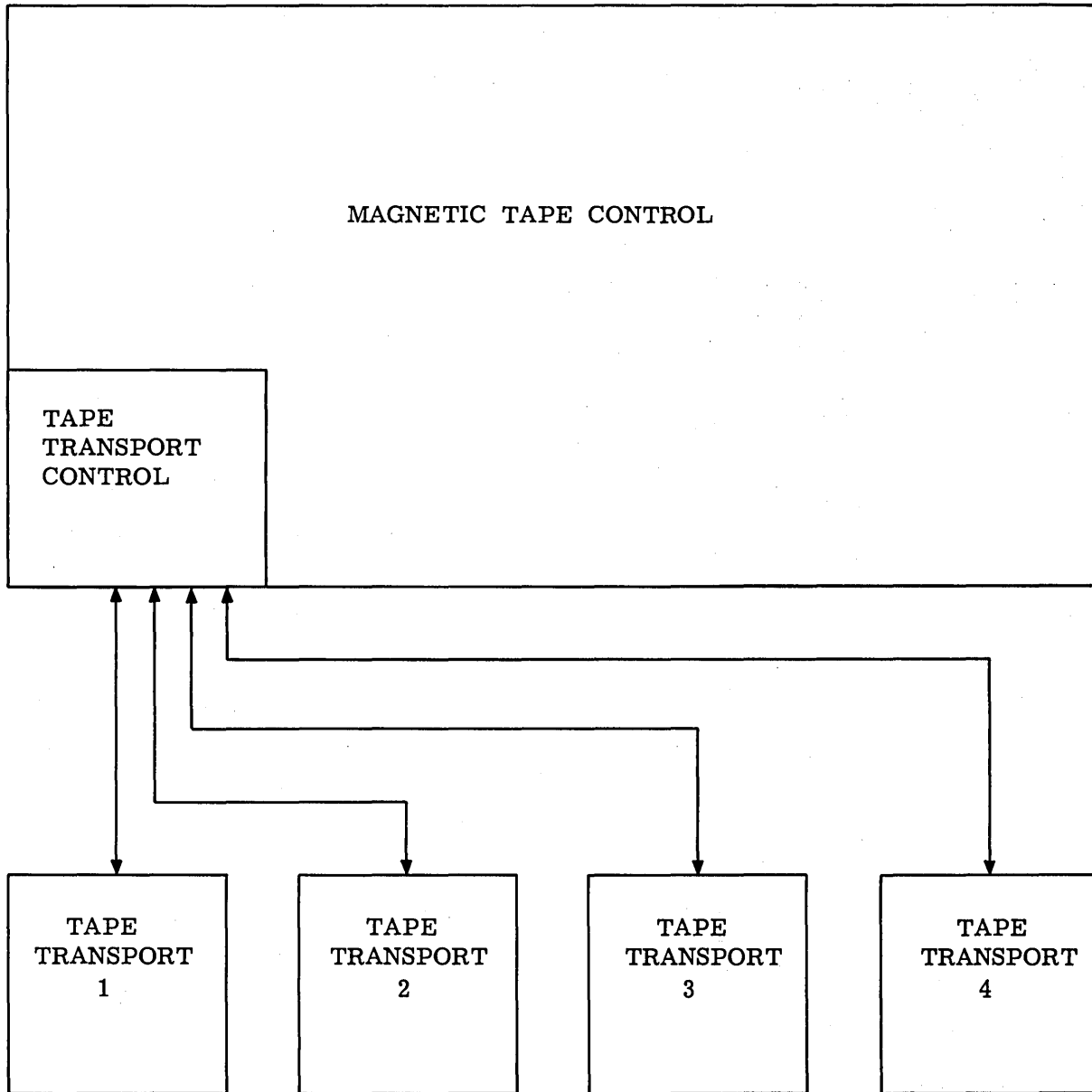


Figure 1. Block Diagram of Magnetic Tape System

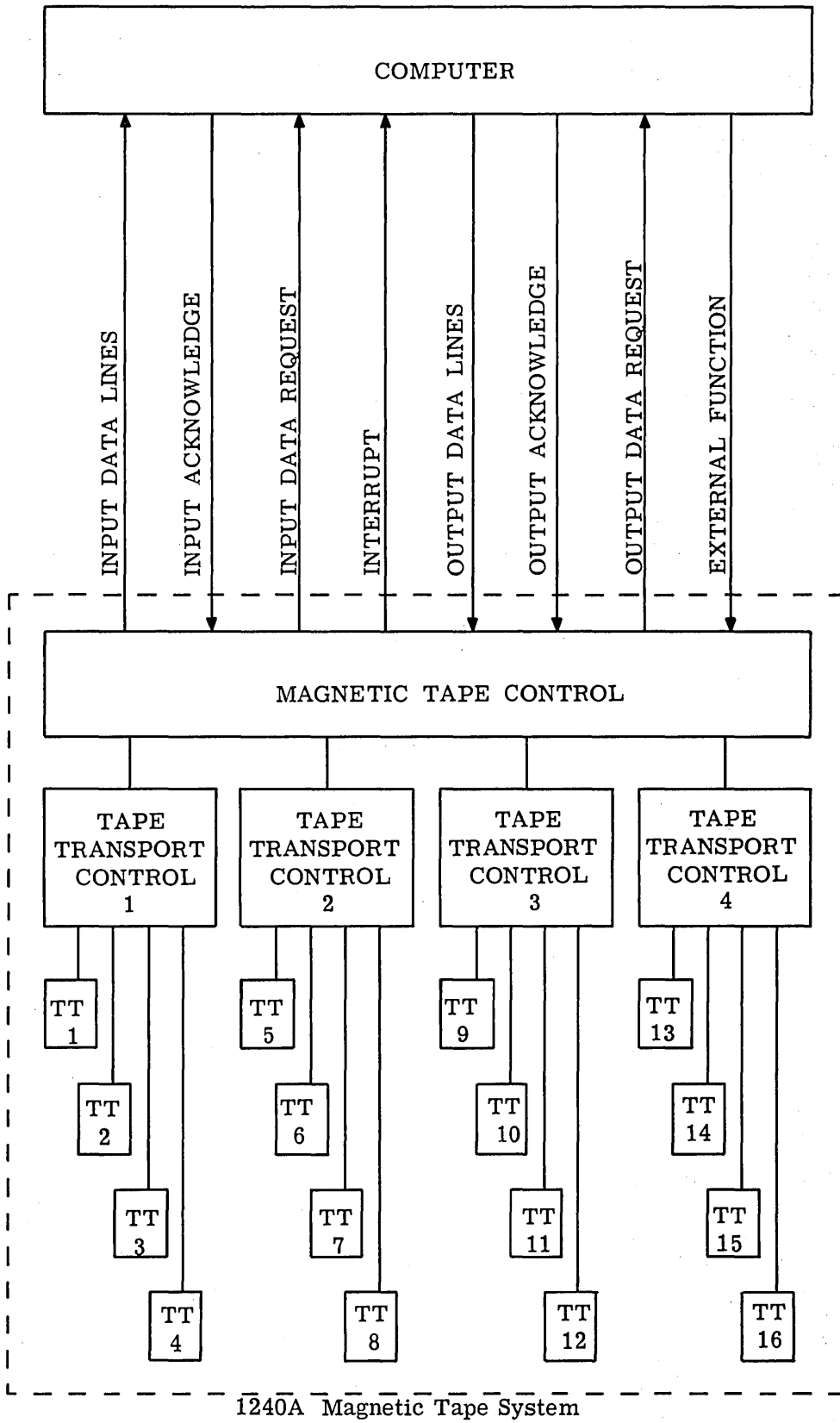


Figure 2. 1240A Interface

2 shows the 1240A interface of system with sixteen transports, four tape transport controls, and one Magnetic Tape Control.

INPUT/OUTPUT SEQUENCE FOR 1240A MAGNETIC TAPE SYSTEM

The input/output sequence, for the 1240A tape system and Computer, begins with the tape system in an idle state and the following events occur.

- The computer places an address word on the output data lines. (See Figure 3 for configuration of address word)
- The computer sets the external function line active
- The Magnetic Tape Control takes the address word and selects the correct cabinet and transport
- The computer places an instruction word on the output data lines. (See Figure 4 for configuration of instruction word)
- The computer sets the external function line active
- The Magnetic Tape Control samples the instruction word, becomes active and performs the operation specified by the instruction word
- The Magnetic Tape Control receives a status word from the tape transport control, and places it on the input data lines
- The Magnetic Tape Control sets the interrupt line active
- The computer accepts the interrupt according to priority
- The computer program handles the interrupt and determines the action to be taken using the status word
- The Magnetic Tape System becomes idle

ADDRESS WORD

The address word is received by the Magnetic Tape System via the external function command from the computer. Bit 17 is set, and will be as specified in Figure 3. The Magnetic Tape System operates with the selected cabinet and tape transport for all operations until another address word is received from the computer.

30	18	17	16	15	06	05	03	02	00	
No Meaning		Function Word Designator = 1	Not Used			Cabinet Address	Transport Address			
			Master Clear							
					0 = Cabinet 4 1 = Cabinet 1 2 = Cabinet 2 3 = Cabinet 3 4 = Cabinet 4 5 = Cabinet 1 6 = Cabinet 2 7 = Cabinet 3			0 = None 1 = TT No. 1 2 = TT No. 2 3 = TT No. 3 4 = TT No. 4 5 = TT No. 1 6 = TT No. 2 7 = TT No. 3		

Figure 3. Address Word

30	18	17	16	15	11	10	09	08	07	06	05	00
No Meaning		Function Word Designator = 0	Operation Code See Table I			Identification Code Selective Read = ID Code; Write, Tape Mark = 001111 Write, Tape Mark, -IXRG = 001111 Density; High = 1/Low = 0 Parity; Odd = 1/Even = 0 Character; Octal = 1/Bioctal = 0 MODULUS MOD 3 = 00 MOD 4 = 01 MOD 5 = 10 MOD 6 = 11						
			Master Clear									

Figure 4. Instruction Word

TABLE I. OPERATION CODES

OPERATION CODE	OPERATION
00000	READ
00001	READ; Selective
00010	READ; Ignore Error Halt
00011	Space File
00100	SEARCH Type I
00101	SEARCH Type II
00110	SEARCH-File Type I
00111	SEARCH-File Type II
01000	WRITE
01001	WRITE; XIRG
01010	WRITE; Ignore Error Halt
01011	WRITE XIRG, Ignore Error Halt
01100	WRITE Tape Mark
01101	WRITE Tape, XIRG
1000x *	Backspace
10010	Backspace-Read
10011	Backspace-File
10100	Backsearch Type I
10101	Backsearch Type II
10110	Backsearch File Type I
10111	Backsearch File Type II
110x0 *	Rewind
110x1 *	Rewind, Clear Write Enable
111x0 *	Rewind-Read
111x1 *	Rewind-Read, Clear Write Enable

* x may be either "0" or "1"

INSTRUCTION WORD

The instruction word is received by the Magnetic Tape Control via the external function command and bit 17 is set to a zero. The instruction word will be as specified in Figure 4.

INTERRUPT AND STATUS WORD

A status interrupt is sent to the computer by the Magnetic Tape Control, 222 microseconds following the completion of all functions except the MASTER CLEAR and TRANSPORT ADDRESS WORD operation. Along with the interrupt the Magnetic Tape Control puts a status word on the data lines. This status word is a signal from tape transport control as to the success of an operation performed on a tape transport. The computer acknowledges the interrupt signal and jumps to the interrupt entrance address for that channel. (Address $20 + \hat{C}$). The computer entrance address should contain a RJP (65000 xxxxx) instruction to computer interrupt program. This program determines if the tape operation was successful. (Figure 5 gives status bits that will be set for any errors that may occur).

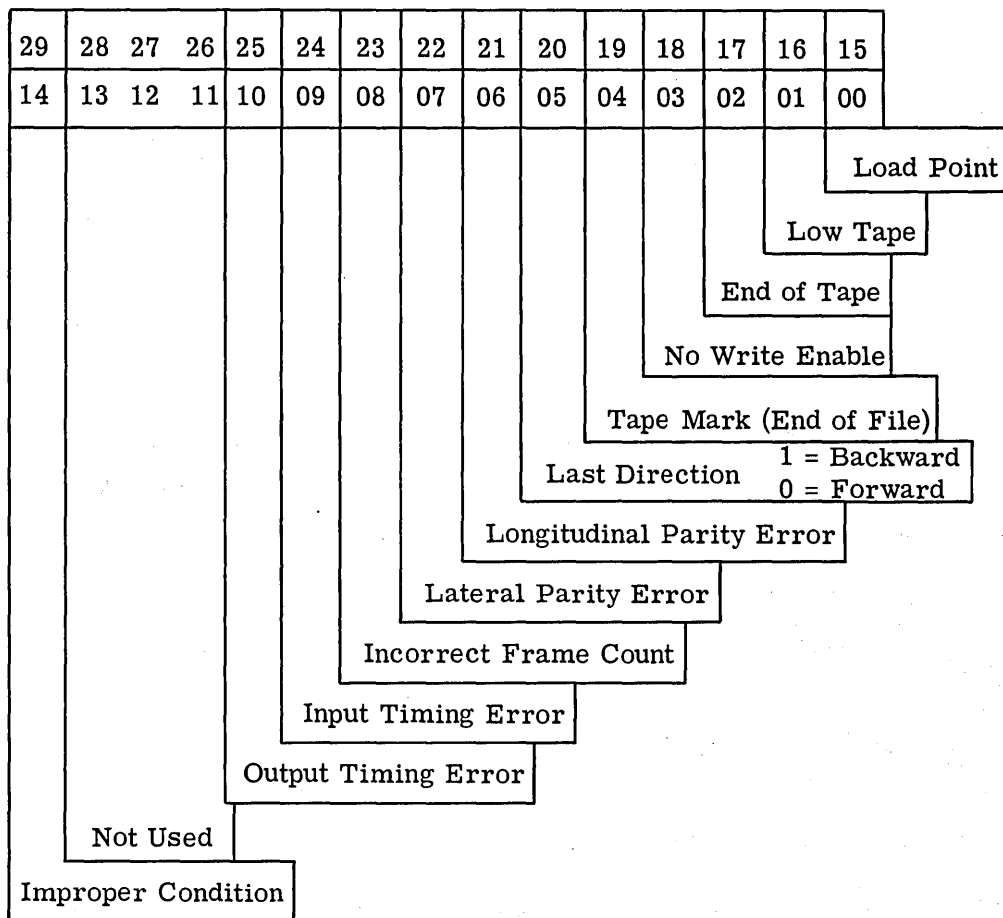


Figure 5. Status Word Format

MAGNETIC TAPE OPERATIONS

MASTER CLEAR (bit 16)

The Magnetic Tape Control performs a Master Clear whenever power is applied, when the MASTER CLEAR switch is operated, and whenever the Master Clear command is received via the external function command from the computer. The MASTER CLEAR performs the following:

- The MASTER CLEAR is accepted by the Magnetic Tape Control System at any time
- The MASTER CLEAR shall not be followed by a status-word interrupt
- The MASTER CLEAR will stop all tape motion (except a rewind of tape) and places the system in the idle state
- The Magnetic Tape Control accepts an external function command anytime after a MASTER CLEAR

READ (bits 11-15)

The read function is supplemented by format, density, and identification code selections. Two types of read operations are performed, normal and selective read. The read function performs the following:

- The Magnetic Tape Control, having received the read function, begins passing tape forward over the read head at a speed of 112.5 inches per second
- The tape transport control checks the parity of each frame, or seven bits, and passes the information onto Magnetic Tape Control
- The Magnetic Tape Control assembles the information into a 30 bit computer word for transfer to the computer. The number of frames required to make up a computer word will depend on the modulus in which it is written
- If the selective read function was selected, the lower six bits of the computer word are compared with the identification code. If a comparison is not correct, the assembled word will be disregarded and the next computer word is assembled. Therefore, only words in the record which have the lower six bits equal to the identification code will be transferred to the computer. In a normal read all words are transferred to the computer through the input data lines
- This process will continue until the record has been completely read, assembled, and transferred to the computer

- A status word is sent to the computer by Magnetic Tape Control at the completion of the read function, informing the computer of the success of operation
- If an error (parity or input timing) is detected during transmission of data, the Magnetic Tape Control will cease transferring data to the computer for the remainder of that record. At the end of the record a status word is sent to the computer informing it of the nature of failure

WRITE (bits 11-15)

The write function is supplemented by format and density selections. The tape speed for a write function is 112.5 inches per second. The following events will occur:

- The Magnetic Tape Control takes a word from the computer on the output data lines
- The Magnetic Tape Control disassembles the computer word according to modulus selection, generates a parity bit, and transfers the seven-bit frame to the tape transport control for recording on tape
- The read-head is activated, causing the tape transport to read back the information recorded, for parity check purposes
- If a parity error is detected, the write operation is halted and a status word telling the computer of failure is sent over interrupt lines. The computer program must then correct the procedure as necessary to perform the write function (write with extended inter-record gap function is the suggested correction measure)
- If no parity error is detected, the process of disassembling and recording data continues until the computer no longer acknowledges the output data request from the Magnetic Tape Control. This means that the complete buffer has been recorded on tape
- When Magnetic Tape Control detects the end of write, tape motion is stopped after 3/4 inches of tape has passed the write head. This 3/4 inch of tape is for Inter-Record Gap (IRG). Extended Inter-Record Gap is 3-1/2 inches
- A status word is sent to the computer and the tape system becomes idle
- In the *WRITE IGNORE HALT* function, the Magnetic Tape Control does not stop the write operation if lateral parity error is detected

REWIND (bits 11-15)

The rewind function causes the tape transport selected to rewind tape at a rate of 225 inches per second. If rewind clear write enable function is selected, the tape will stop at load point and a write function will not be accomplished on this unit. If the tape is located at load point when a rewind function is given, no improper conditions will occur.

REWIND-READ (bits 11-15)

The rewind-read function causes the same effect as normal rewind, except when the tape reaches load point. The first record will be read into the computer by normal read function. The status word will be sent after the first record is transferred to the computer. The rewind-read clear write enable disables the write enable making further writing on this unit impossible. This function is supplemented by format and density selection.

SPACE FILE FORWARD/BACKWARD (bits 11-15)

When the Magnetic Tape Control is instructed to space file forward/backward, it causes the addressed tape transport to move in the specified direction, beyond the next tape mark. The Magnetic Tape Control notifies the computer via the status word with a tape mark indication.

WRITE TAPE MARK (bits 11-15)

This function causes the Magnetic Tape Control to instruct the tape transport to write a Tape Mark, a special record having 17₈ in the first frame, 3 frames of zeros, and one frame of longitudinal parity.

BACK SPACE (bits 11-15)

The back space operation causes the selected tape transport to move one record in the backward direction. The tape is properly positioned in the Inter-Record Gap ready for a read or write function. The parity is checked while the backspace operation is performed and a status word is sent to the computer. The back space function is supplemented by format and density selections.

SEARCH (bits 11-15)

The search function combines the normal read with the ability to conduct a search on the first word of a record (in either the forward or backward direction) and transfer only the "Find" record to the computer. The search comparison is performed on the first word of a record with the identifier (search key) word. The search word is transmitted to the Magnetic Tape Control by the computer in a one-word buffer following the instruction word. The

search forward/backward file function performs the same function except the search is limited to a file mark. There are two types of searches which the Magnetic Tape Control can perform in comparing the key word with the first word of the record. Type I is defined as a per bit greater-than-or-equal compare. The following example demonstrates this definition:

search key:	001101
find:	011101
find:	001101
no find:	010101
no find:	001100

An identical compare (Type II) is defined as a search comparison with the search key and the first word of the record exactly identical.

FORMAT PORTION OF INSTRUCTION WORD

The format portion of the instruction word consists of modulus, character, and parity. A complete format selection must be included in all instruction words which require a record or read operation. The three sections of the format are discussed in the following paragraphs:

MODULUS

The Magnetic Tape System is capable of recording and reading in four different moduli. These moduli and the appropriate designator bits (bits 10-09) are:

- 00 = Modulus 3
- 01 = Modulus 4
- 10 = Modulus 5
- 11 = Modulus 6

These are discussed in the section titled "Tape System Moduli".

CHARACTER

There are two types of character recording; octal and bioctal. A "one" in bit 08 of the instruction word specifies octal. In this type, channels 3, 4, and 5 contain the same information as channels 0, 1, and 2 respectively for each frame, except that when channels 0, 1, and 2 are all "zeros", channels 3, 4, and 5 contain all "ones". Odd lateral parity is always generated when recording in octal character (see Table II). A "zero" in bit 08 specifies bioctal recording. The octal character allows for redundant recording for added reliability.

TABLE II. OCTAL RECORDING

CHARACTER		TAPE CHANNELS
Octal	Binary	
		6 543 210
0	000	0 111 000
1	001	1 001 001
2	010	1 010 010
3	011	1 011 011
4	100	1 100 100
5	101	1 101 101
6	110	1 110 110
7	111	1 111 111

PARITY

Two parity modes can be utilized, odd or even, bit 07 is used for parity mode selection. A "one" in bit 07 specifies odd parity, and in bit 07, a "zero" specifies even parity.

Data ordinarily are recorded in two formats: binary and binary-coded-decimal. The parity bit is chosen to make the total number of "ones" ("1's") bits in a frame odd in the binary format and even in the binary-coded-decimal format.

DENSITY

The Magnetic Tape System is capable of recording data on tape in two different programmable densities. The two densities are low density, at 200 frames per inch, and high density, at 556 frames per inch.

Bit 06 of the instruction word is used for density selection. When bit 06 is a "one", high density is selected; when it is a "zero", low density is selected.

TAPE SYSTEM MODULI

MODULUS 3: (Bits 10 and 09 = 00)

Mod 3 is obtained by reducing 18 bits of a computer word to three (bioctal character) frames of data. In reading Mod 3, a word is sent to the computer for every three (bioctal character) tape frames. These frames are assembled in the lower 18 bits (17-00) of the data word. The upper bits, if any, of the data word contain zeros.

Recording Mod 3, the 18 bits (17-00) of a computer word are recorded in three (bioctal characters) 7-bit frames, consisting of a 6-bit character, plus parity. For octal recording the number of tape frames is doubled. See Figure 6 for bioctal recording and Figure 7 for octal recording of bit arrangements on tape.

MODULUS 4 (Bits 10 and 09 = 01)

Mod 4 is obtained by reducing 24 bits of a computer word to four (bioctal characters) frames of data. In reading Mod 4, a word is sent to the computer for every four (bioctal character) tape frames. These frames are assembled in the lower 24 bits (23-00) of the data word. The upper bits, if any, of the data word contain zeros.

Recording Mod 4, the 24 bits (23-00) of a computer word are recorded in four (bioctal character) 7-bit frames, consisting of a 6-bit character, plus parity. For octal recording the number of tape frames is doubled.

MODULUS 5: (Bits 10 and 09 = 10)

Mod 5 is obtained by reducing 30 bits of a computer word to five (bioctal character) frames of data. In reading Mod 5, a word is sent to the computer for every five (bioctal character) tape frames. These frames are assembled in the lower 30 bits (29-00) of the data word. The upper bits, if any, of the data word contain zeros.

Recording Mod 5, the 30 bits (29-00) of a computer word are recorded in five (bioctal character) 7-bit frames, consisting of a 6-bit character, plus parity. For octal recording the number of tape frames is doubled.

MODULUS 6: (Bits 10 and 09 = 11)

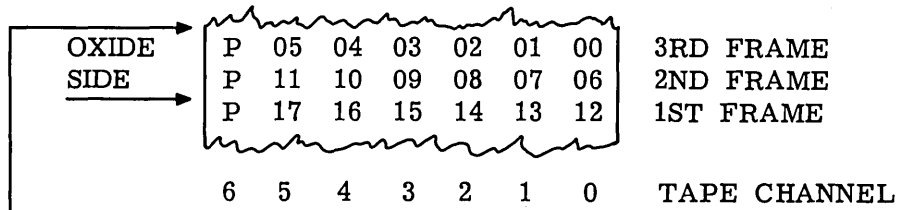
Mod 6 is obtained by reducing 36 bits of a computer word to six (bioctal character) frames of data. In reading Mod 6, a word is sent to the computer for every six (bioctal character) tape frames. These frames are assembled in the 36 bits (35-00) of the data word.

Recording Mod 6, the 36 bits (35-00) of a computer word are recorded in six (bioctal character) 7-bit frames, consisting of a 6-bit character, plus parity. For octal recording the number of tape frames is doubled.

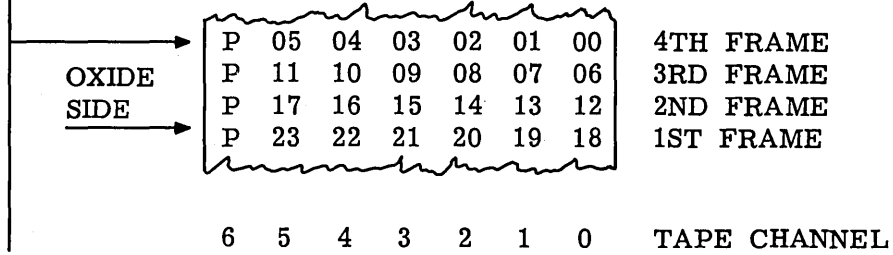
STATUS WORD

A Status interrupt is sent to the computer 222 microseconds following the completion of

FORWARD TAPE
DIRECTION

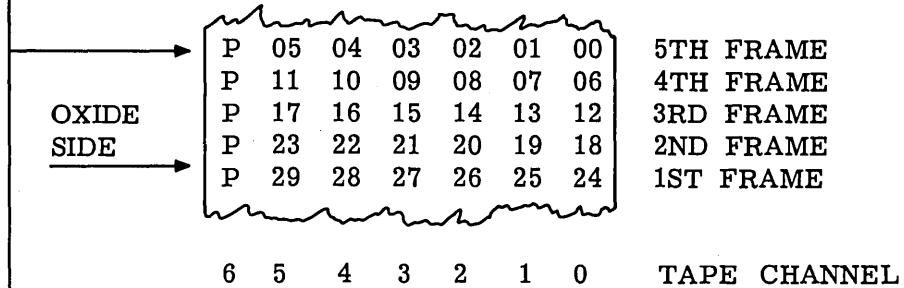


MODULUS 3

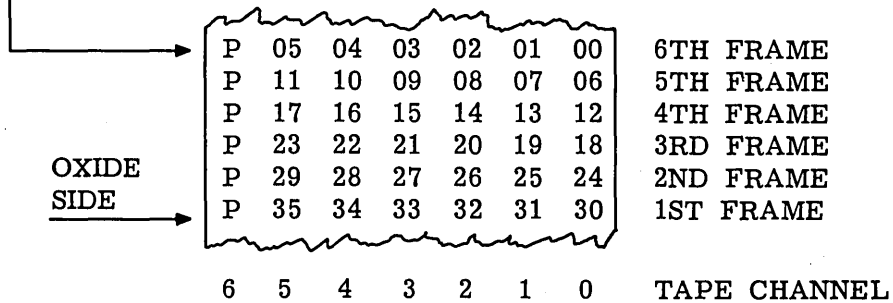


MODULUS 4

THIS EDGE
OF TAPE
NEXT TO
TRANSPORT



MODULUS 5



MODULUS 6

Figure 6. Biocital Tape Format

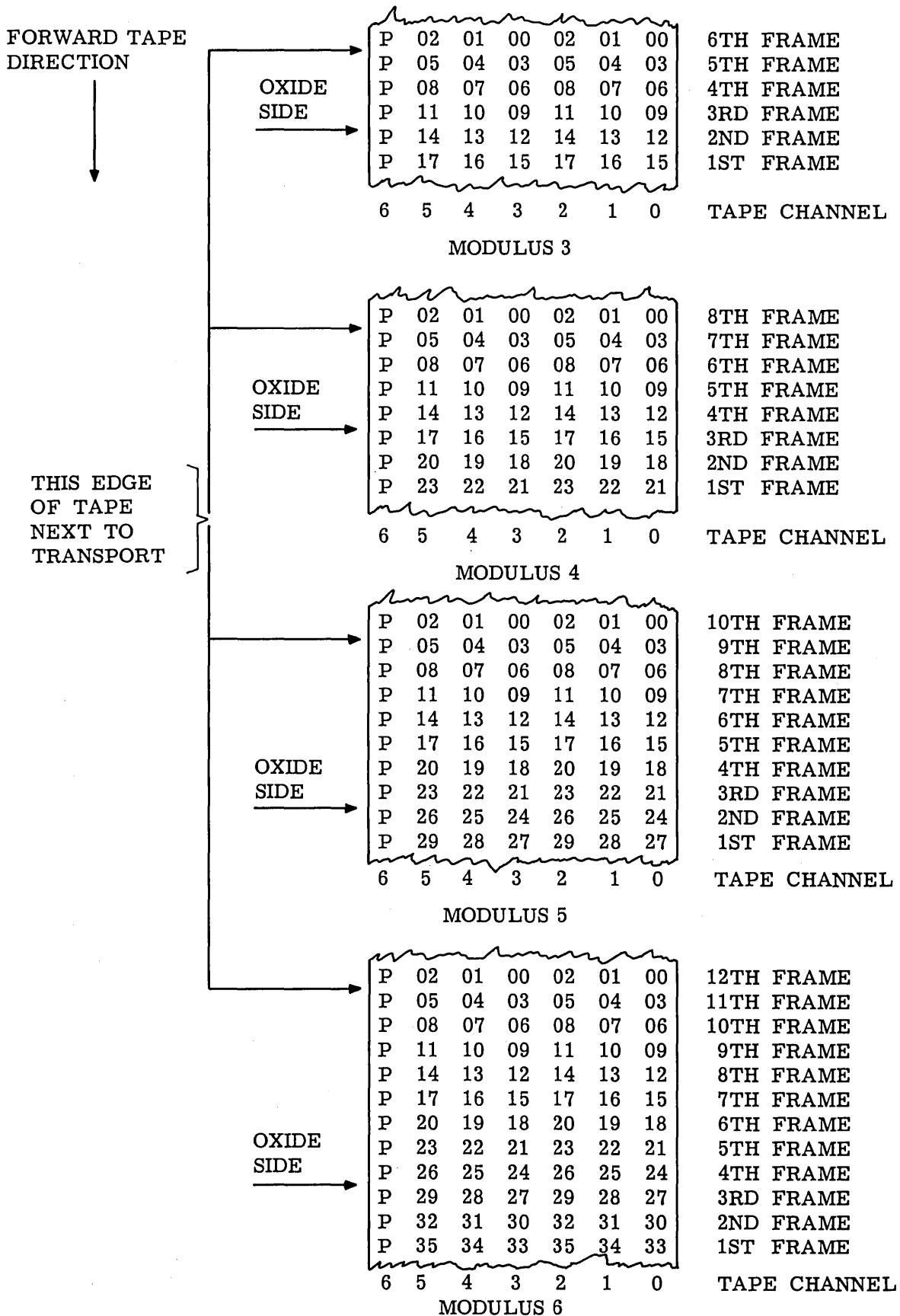


Figure 7. Octal Tape Format

every function except MASTER CLEAR and TRANSPORT ADDRESS SELECTION. A status word is placed on the data lines of the input cable. The bit structure of the status word enables the computer to determine whether or not the previous function was successfully completed.

The computer program must recognize that after issuing an external function instruction to the Magnetic Tape System, no subsequent external function command (except addressing and Master Clear) will be recognized until receipt of the acknowledge to the Status interrupt, signifying the end of the first instruction.

Figure 5 shows bit assignments in the status word. These conditions are described below.

IMPROPER CONDITION (Bits 29 and 14)

A one in bits 29 and 14 may imply that operator intervention is necessary to overcome the difficulty. An improper condition will occur whenever:

- Reference tape transport is not in automatic condition
- No tape transport is selected when one is required
- A forward command is sent to a tape transport whose tape is positioned at end of tape
- A reverse command is sent to a tape transport whose tape is positioned at Load Point (except a Rewind operation)
- A Write instruction is issued to a tape transport that has NO Write Enable

When the computer has been notified of an improper condition, the computer program may then refrain from issuing further external function commands to the tape system to allow visual inspection of the trouble, or it may issue another external function command. An incoming external function command to the tape system causes the Improper Condition indicator to extinguish.

A tape transport not in automatic condition implies one of the following situations:

- Tape transport was manually removed from automatic
- Tape transport not in ready condition for one of the following reasons:
 - Power off

- Tape broken
- Lamp burnout
- Tape load was not accomplished when tape was mounted

OUTPUT TIMING ERROR (Bits 25 and 10)

A "one" in bits 25 and 10 indicates that the computer did not acknowledge the first Output Data Request, or the computer acknowledged the Output Data Request too late (however, it did acknowledge the Output Data Request for the data word to be written in its proper place). The acknowledge time is related to format and density.

Also an output timing error can occur during Search operations if the Magnetic Tape System does not receive the search key before the start of reading the record. This time requirement may be as short as two milliseconds.

INPUT TIMING ERROR (Bits 24 and 09)

A "one" in bits 24 and 09 indicates that Magnetic Tape Control information on the input cable was not accepted by the computer before the subsequent word was to be placed on the input cable. This condition indicates that the computer "lost" one or more words of the last record. If an input timing error occurs, data transmission to the computer ceases for the remainder of the record.

INCORRECT FRAME COUNT (Bits 23 and 08)

A "one" in bits 23 and 08 indicates either some frames were lost, or improper modulus specified (i.e., there were not enough frames in the record to complete an integral number of computer words). This situation may result from one or more of the following:

- One or more characters were not properly read or recorded
- Bad spots on the tape caused characters to be lost
- Reading a record with the wrong format (for example, reading Mod 4 with a tape record in Mod 5)

A longitudinal parity error usually occurs in conjunction with an incorrect frame count if frames were lost.

LATERAL PARITY ERROR (Bits 22 and 07)

A "one" in bits 22 and 07 informs the computer that the lateral parity of one or more frames read did not agree with that specified in the format.

LAST TAPE MOTION (Bits 20 and 05)

A "one" in bits 20 and 05 indicates that the last tape motion was backward. A "zero" indicates that the last tape motion was forward.

LONGITUDINAL PARITY ERROR (Bits 21 and 06)

When recording, longitudinal parity is generated by Magnetic Tape Control for each channel and recorded after the last frame of the record. When reading (Read, Back Space, Post-Write Check) the longitudinal parity of a record is checked by Magnetic Tape Control, and if in error, noted in the status word (bits 21 and 06).

TAPE MARK (Bits 19 and 04)

A "one" in bits 19 and 04 indicates that the Magnetic Tape Control has sensed a Tape Mark during a Read, Write, (before a Search Comparison is made during a Search File instruction) or Back Space function.

NO WRITE ENABLE (Bits 18 and 03)

A "one" in bits 18 and 03 informs the computer that the referenced tape transport has no write enable when a Write operation is attempted or that the Write Enable Ring is not inserted in the tape reel.

END OF TAPE (Bits 17 and 02)

To prevent reading or writing off the end of the tape, an *end of tape* reflective marker is placed a minimum of 14 feet from the physical end of the tape.

When the *end of tape* mark is sensed, a 1/2 second "time-out" begins. When this time period is completed, no further forward movement of the tape will be possible. However, the tape may be moved in the reverse direction past the reflective marker and then moved forward. When the marker is again sensed, the "time-out" is initiated again, and the forward tape motion will halt after 1/2 second.

LOW TAPE (Bits 16 and 01)

A "one" in bits 16 and 01 informs the computer that the tape is positioned less than 100 feet from end of tape.

LOAD POINT (Bits 15 and 00)

Since the first several feet of tape undergo excessive wear and are required to load the transport, no recording is done on this portion of the tape. Recording begins at Load Point and this point is recognized by the Magnetic Tape System by means of a reflective marker placed at least ten feet from the physical beginning of tape. The write, load point, delay

allows information to be written on the tape approximately 3/4 inch beyond the load point marker with the tape moving in the forward direction.

TAPE MARKERS

The load point and end of tape markers are pressure-sensitive, adhesive-coated strips of aluminum 1 by 3/16 inch. The markers are detected by reflective photo-sensing means. Both markers are placed on the base (uncoated) side of the tape with the 1-inch dimension parallel to the tape. The load point marker is placed 1/32 inch from track "Ø", or the front edge of the tape. The end of tape marker is placed 1/32 inch from track "6" or inside edge of the tape.

LOGICAL SELECTION OF TAPE TRANSPORTS

Selector Switches to change the logical address of each tape transport are provided. Any physical tape transport may be switched to any logical address. If logical address selections are duplicated, the lowest order physical tape transport has priority; however, no two cabinets may have the same logical address. This will be the responsibility of the operator.

1240A HIGH-SPEED PRINTER OFF-LINE COMPATIBILITY

The magnetic tape subsystem is capable of communicating directly with the high-speed printer subsystem for OFF-LINE operation. The interface between the magnetic tape unit and printer is shown in Figure 8.

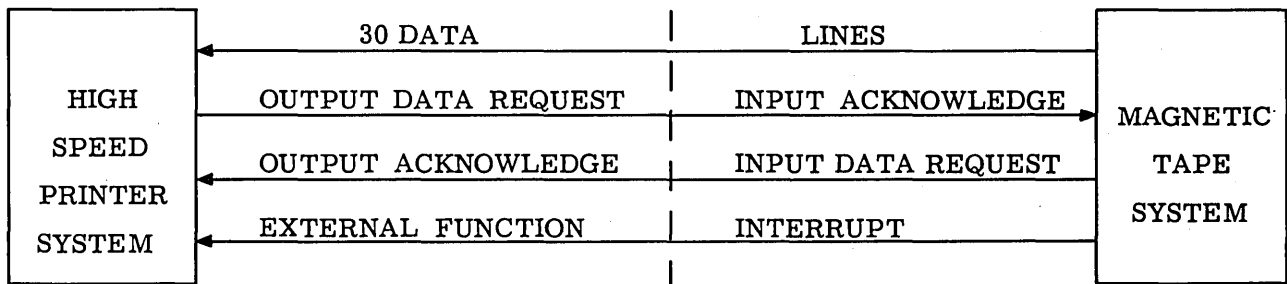


Figure 8. Magnetic Tape - High-Speed Printer Interface

The printer output is connected to the magnetic tape system's input (input and output as used here are in reference to the computer).

The data to be printed OFF-LINE must be recorded in records on tape in the following format:
Record Length of 120 Fielddata characters.

The Magnetic Tape System will read each record of data from tape in the following format:
Modulus 5.

At the magnetic tape unit function register, the operator will manually select the character, parity, and density.

Each record of 120 characters will form 24 30-bit data words which will be printed as one line by the high-speed printer.

A tape mark will be recognized by the high-speed printer as a top of form command. This will position the paper to the top of the next page.

A record of less than 24 words (preferably one computer word, five characters) will cause the high-speed printer to stop the printing operation. This record will not be printed, if the characters are space codes (05).

With the magnetic tape system switched to the Printer Mode, the desired tape transport selected and the tape positioned at load point, the high-speed printer will initiate operation when it is placed in the ON-LINE position.

The normal sequence of events for transfer of data from the magnetic tape system to the printer is as follows: (See Figure 9)

- The printer sets its Output Data Request (e)
- The Magnetic Tape System, in the idle state (a) recognizes this first Output Data Request from the printer as an external function and starts the read operation
- The Magnetic Tape System places the information on the data lines and sets its Input Data Request (c)
- The printer recognizes the Magnetic Tape System Input Data Request as an Output Acknowledge (f)
- The printer samples the data lines and clears its Output Data Request
- The Magnetic Tape System recognizes the clearing of the Printer Output Data Request as an Input Acknowledge (b)

Steps 3, 4, 5, and 6 are repeated for each word of the record.

The normal sequence for sending an external function command top of form from the magnetic tape system to the printer is the same as reading a record except that when the magnetic tape system detects the Tape Mark, it will set bit 4 in the status word, and when the interrupt (d) is set, the printer will recognize this as an external function command top of form (g) (Figure 9).

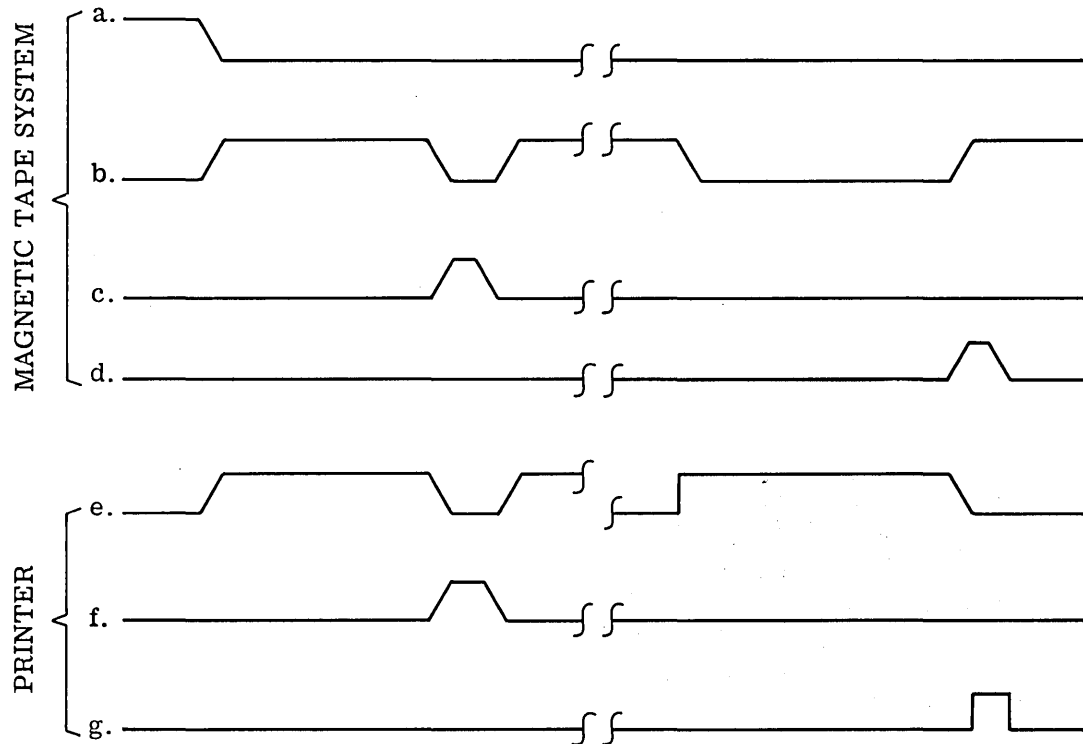


Figure 9. Sequence of Events in Tape-Printer Operation

PROGRAMING CONSIDERATIONS

GENERAL

The Magnetic Tape System is manually placed in an operational condition. The operator functions include mounting tapes on transports, turning power on, and initially positioning tapes. With the Magnetic Tape System operational, programed reference may begin. Generally, all programing of the tape system must be done with force and must conform to a standard sequence of reference. (Figure 10 illustrates this sequence.)

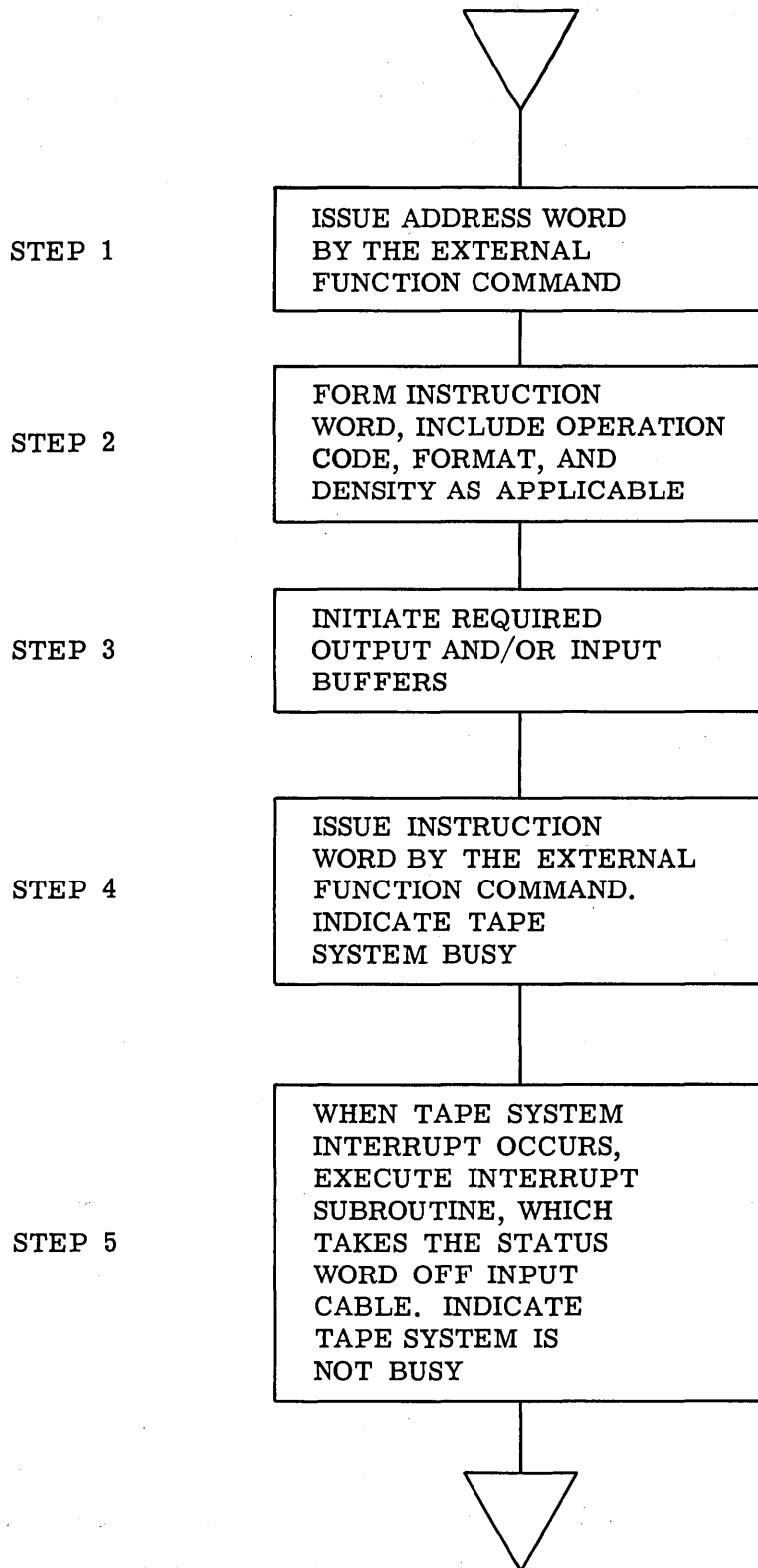


Figure 10. Sequence of Programming References - Magnetic Tape System

Once the tape system receives and starts to execute the operations in an instruction word, further external function commands, other than *Master Clear*, are ignored. The programmer must remember when an external function command may be logically issued. After issuing an external function command, other than *Address Word* and *Master Clear*, the computer may not logically issue another function command until the computer acknowledges the receipt of an interrupt from the tape system.

Step 1 of Figure 10 is required only on the initial reference of a tape transport or when a reference to another transport is desired.

WRITE PROCEDURES

To write, the instruction word must include complete format selection (modulus, character, and parity), and density. Use of the procedure outlined in Figure 10 will result in a record of words being written on tape. Length of record is determined when the output buffer is initiated.

After the Status interrupt is received, signifying end of a write operation, the program must check the following four conditions to determine successful completion of the write operation:

- No improper condition in the status word
- No output timing error in the status word
- No lateral parity error in the status word
- Output buffer is terminated

If the status word indicates an output timing error, the computer did not acknowledge the first output data request, or the computer acknowledged the output data request too late (however, it did acknowledge the request) for the data word to be written in its proper place. The acknowledge time is related to format and density.

It is possible for an output timing error to occur that will not be shown in the status word. Such a condition results if the computer did not acknowledge the output data request (other than the first output data request from the tape system). The tape system recognizes this situation as *end of record* and, consequently, indicates no error. However, if words are left unwritten in the output buffer, this constitutes an output timing error condition.

If a lateral parity error is indicated in the status word, the write operation was terminated when the post write check detected an incorrect frame. It is the responsibility of the program to decide and provide the recovery procedures.

READ PROCEDURES

To read, the instruction word must include complete format selection, Identification Code (if read selective) and density. Use of the procedure outlined in Figure 10 will result in a record being read from tape. The length of the input buffer must be long enough to cover the record to be read.

An input timing or parity error will terminate the data input to the computer unless the read operation is with the ignore error halt option. The Status interrupt will still be sent by the Magnetic Tape System at the end of that record.

SEARCH PROCEDURES

To search, the instruction word must include complete format selection and density. The identifier word will be received by the Magnetic Tape System via a one-word output buffer. Tape motion is started upon the receipt of the instruction word and an output timing error will occur if the Magnetic Tape System does not receive the identifier within the time from start of tape motion and when the compare is made. This time requirement may be as short as two milliseconds. One record will then have been passed. The search is terminated by an output timing error and a parity error.

When searching backwards, the 30-bit identifier word sent by the computer must be reversed, characterwise; its configuration is dependent upon format. Examples are given below:

Original Computer	Biocral	Octal
Word	3456745321	3456745321
MOD 5	2153745634	1235476543
MOD 4	0053745634	0035476543
MOD 3	0000745634	0000476543
MOD 6	5374563400	3547654300

RECORD LENGTH

There are no limits on record length within physical tape capacity. When reading tapes of unknown record length, the input buffers must be made sufficiently large to insure reading the entire record. Another method is to initiate an input buffer with monitor and make provision for the initiation of additional buffers to read the complete record.

END OF FILE

The normal end-of-file Inter-Record Gap is approximately 3/4-inch long followed by a Tape

Mark (001 111) and its associate check character. The end of file is always recorded with even parity.

EDITING OF TAPE

By suitable programing, an Inter-Record Gap of any length may be written to precede any record. Records may be rewritten for type updating, and they may be lengthened, provided suitable Inter-Record Gap was used on a previous recording. A record may be inserted for a previously used extended Inter-Record Gap.

BAD TAPE

If when writing a record, a tape "bad spot" is encountered, where recording is marginal or impossible, the tape may be back spaced to the beginning of the record and rewritten with an extended Inter-Record Gap. The long Inter-Record Gap will probably be sufficient to move the bad spot past the recording head. Successive extended Inter-Record Gap's may be written if the bad spot still appears.

UNIVAC HIGH-SPEED PRINTER (MODEL 1469)

BASIC INFORMATION

The high-speed printer subsystem (Figure 1) consists of an Anelex 4-1000 printer and a UNIVAC printer control unit. The printer control unit provides the required timing and control signals for the printer and provides a compatible interface for communication between the computer and the subsystem. The subsystem is designed for on-line use with a computer or for off-line use with the magnetic tape subsystem. Figure 2 is a block diagram of the subsystem, when used on-line with a computer, showing the information flow between the printer, printer control unit, and computer. Figure 3 is a block diagram of the sub-system, when used off-line with the magnetic tape subsystem, showing the exchange of data and information between the two subsystems.

PRINTER SPECIFICATIONS

The high-speed printer subsystem has the following characteristics:

- Two printing speeds
 - Slow rate of 560 to 667 lines per minute
 - Fast rate of 780 to 1000 lines per minute
- Paper slew rate of 25 inches per second
- Vertical line spacing of six per inch
- One hundred twenty characters per line (ten characters per inch)
- Up to 63 printable characters, each represented by a 6-bit binary code. There are 64 possible codes, 05 being used for the space code.
- Vertical control of paper position by means of an 8-track punch tape control loop in conjunction with the Control Panel buttons and the External Function. There are

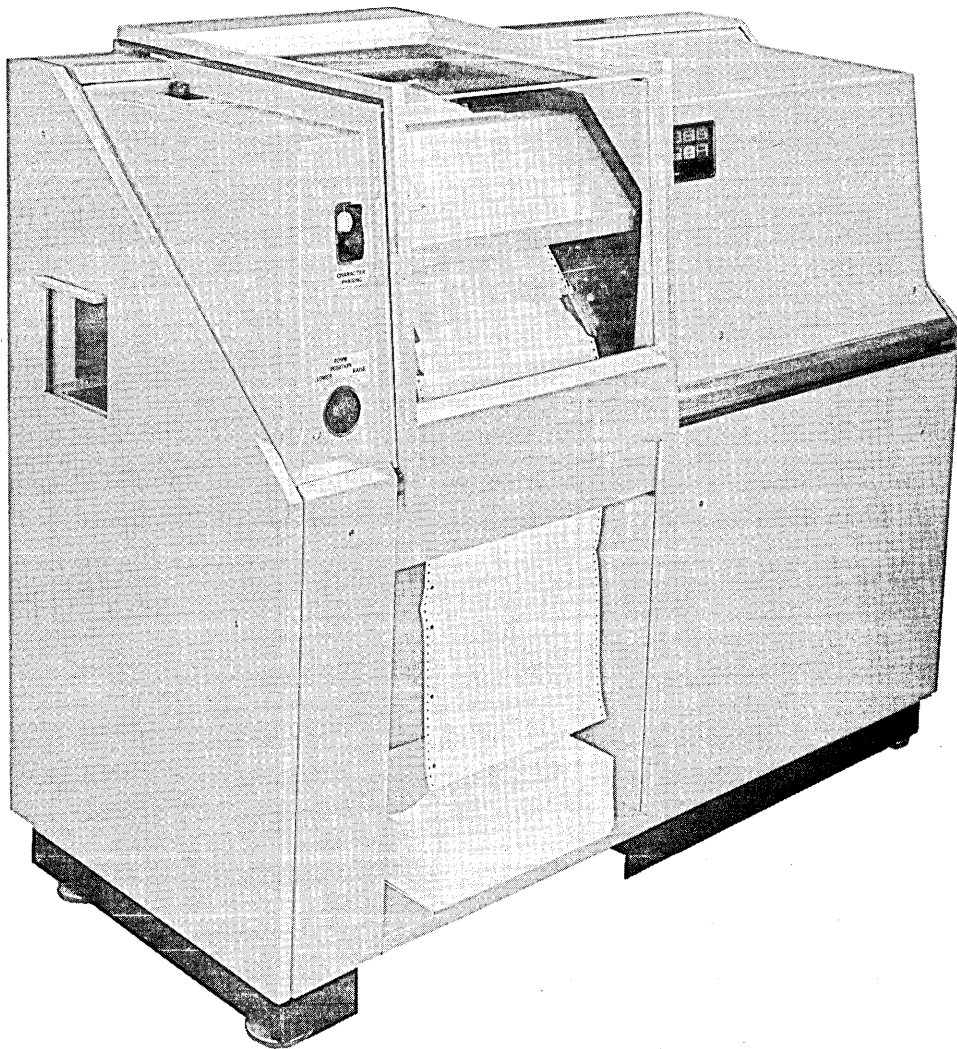


Figure 1. UNIVAC High-Speed Printer (Model 1469)

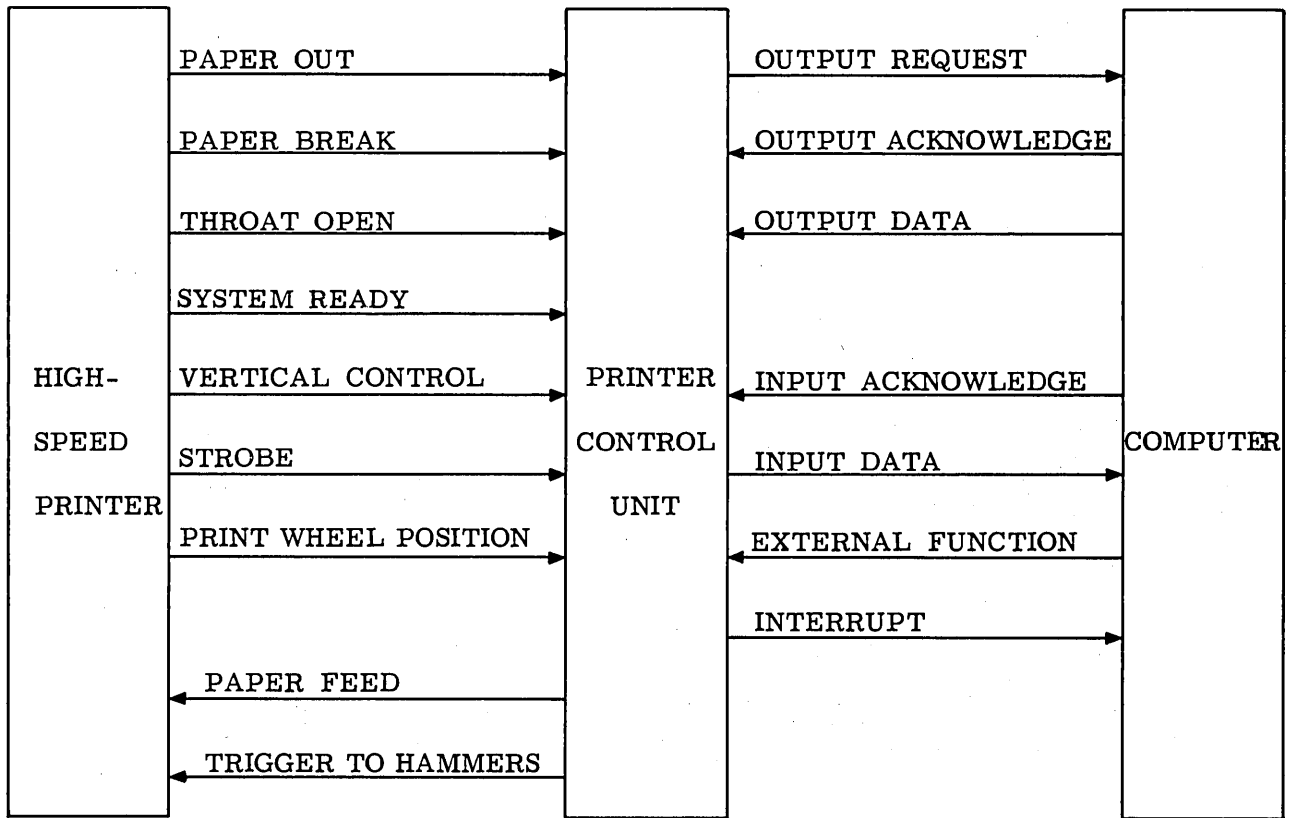


Figure 2. Block Diagram of High-Speed Printer Subsystem, Used On-Line With A Computer

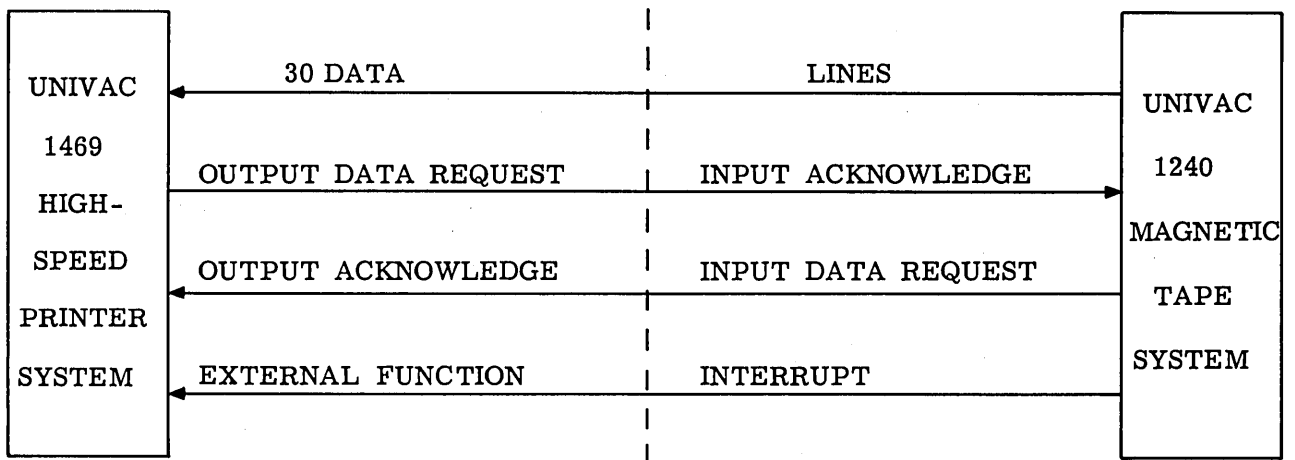


Figure 3. Magnetic Tape - High-Speed Printer Interface

one computer controlled vertical positioning and two manual vertical positionings. Another vertical positioning is used to stop the paper when an out-of-paper condition is detected

The high-speed printer prints 120-character lines; therefore, 24 computer words must be buffered to the Printer Control Unit. These words contain five characters of six bits each. When the Output Request goes up at the high speed printer, the computer begins the output buffer operation.

The computer words are transferred to the printer one at a time and stored in the core memory of the Printer Control Unit one character at a time. When the memory is full, the print cycle begins. The order in which the characters are printed is shown in Figure 4.

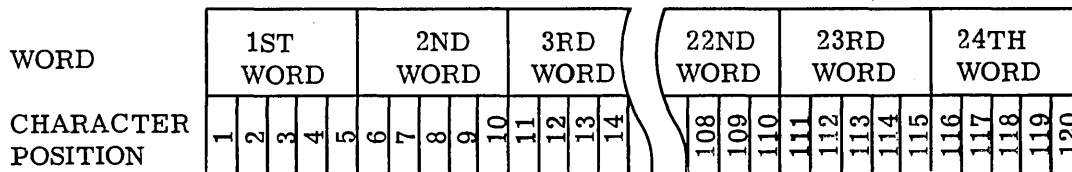


Figure 4. Printing Order

Table I contains a list of the characters used on the printer and their associated codes.

PRINTER CONTROL UNIT

The Printer Control Unit controls the transfer of data from the computer to the printer. The Printer Control Unit contains a 120-character core memory and has the following registers:

- B Register
 - A 30-bit register in which the data words from the computer are temporarily stored until the five characters can be transferred through the Z register to memory
- Z Register
 - A 6-bit transient register used as a memory buffer
- S Register
 - A 7-bit counter used both to count the characters when loading and reading memory and as the memory address register

TABLE I. HIGH-SPEED PRINTER CHARACTER CODES

PRINTER CODE	OCTAL EQUIV.	PRINTING CHARACTER	PRINTER CODE	OCTAL EQUIV.	PRINTING CHARACTER
000000	00	absolute value	100000	40) parenthesis, close
000001	01	↑ arrow, vertical	100001	41	- minus
000010	02	8 subscript eight	100010	42	+ plus
000011	03	[bracket, open	100011	43	< less than
000100	04] bracket, close	100100	44	= equal
000101	05	Space - not printed	100101	45	> greater than
000110	06	A	100110	46	≤ less than or equal
000111	07	B	100111	47	{ brace, open
001000	10	C	101000	50	* asterisk
001001	11	D	101001	51	(parenthesis, open
001010	12	E	101010	52	≥ greater than or equal
001011	13	F	101011	53	: colon
001100	14	G	101100	54	} brace, close
001101	15	H	101101	55	V "OR"
001110	16	I	101110	56	, comma
001111	17	J	101111	57	≠ not equal
010000	20	K	110000	60	0 zero
010001	21	L	110001	61	1
010010	22	M	110010	62	2
010011	23	N	110011	63	3
010100	24	O	110100	64	4
010101	25	P	110101	65	5
010110	26	Q	110110	66	6
010111	27	R	110111	67	7
011000	30	S	111000	70	8
011001	31	T	111001	71	9
011010	32	U	111010	72	∧ "AND"
011011	33	V	111011	73	; semicolon
011100	34	W	111100	74	/ virgule
011101	35	X	111101	75	. period
011110	36	Y	111110	76	→ arrow, horizontal
011111	37	Z	111111	77	x multiply

- C Register

A 6-bit counter used to count character pulses coming from the print wheel. This register always contains the code of the next character to come up for printing

- K Register

A 5-bit shift register used to gate the correct character from B to Z when loading memory

The Printer Control Unit operation consists of the three following sequences:

TRANSFER SEQUENCE

During the transfer sequence, the Output Request is set, data are received from the computer, and the data are stored in the Printer Control Unit memory. When the acknowledge is received from the computer, the 30-bit data word is stored in the B register, and the Output Request is dropped. The contents of the B register are transferred, one character at a time, to the Z register and then to memory. The first 6-bit character taken out of the B register is that in bits 24 through 29. The first character of the first word goes into address zero of memory, the second character to address one, etc. After each character is transferred to memory, S is advanced by one and K is decremented by one to prepare for transferring the next character to memory. After the five characters of a word have been transferred to memory, the Output Request is again set. This sequence of transferring characters to memory is continued until either S has counted to 119 or the top of form command is received. When one of these conditions occurs, the transfer sequence is terminated, and the print sequence begins.

PRINT SEQUENCE

When the transfer sequence has ended and a character pulse is received from the printer, a search of memory begins to determine if a code is present that is identical with the one of the next character appearing under the print hammers. The contents of address zero are read out of memory and compared with C (which contains the code of the next character to be printed). If the information compares, a "one" is set in bit zero of the Hammer register, and 05 (space code) is written in memory at that address. Then address one is compared with C, etc. The first bit position of the Hammer register corresponds to the leftmost printing position on the paper, and when a "one" is set in the Hammer register, the character is printed in that column. When the contents of a memory address do not compare with the character code in C, a "zero" is set in the corresponding bit position of the Hammer register, and the character is written back in memory. When 120 bits have been set in the Hammer register and the

next character pulse occurs, the hammers are triggered and the loading sequence starts again for the next character. This sequence is repeated until memory contains nothing but 05's (space codes). At this time the transfer sequence begins again, coincident with the paper advance sequence.

PAPER ADVANCE SEQUENCE

As soon as the print sequence is complete, the paper advance sequence starts. If an external function has disabled the line feed, the paper is not advanced. If the external function has enabled the line feed, the paper is advanced. If an external function top of form command is received and the line feed is enabled, the paper is advanced to the top of the next form. The print sequence cannot start again until 10 milliseconds after the paper advance sequence has ended.

The following signals are sent from the Printer Control Unit to the Printer:

- Data Lines

A positive transition of 6 volts occurs on the data lines if the bit in the Hammer register is set. These lines go up at each character pulse if "ones" are set in the respective bits of the Hammer register

- Paper Feed

A signal to the printer which starts or stops paper movement. This signal is set to -6 volts to advance the paper. The paper continues to move until this line returns to zero volts. The duration of this signal is determined by the desired number of lines of paper advance. This line is set to -6 volts at the end of the line if the line feed is enabled, or the manual LOAD PAPER or TOP OF FORM buttons are depressed

The following signals will be made available from the Printer to the Printer Control Unit:

- System Ready

A contact closure is available when the printer is ready for use (i.e., power is on, etc.). When the contact opens, an interrupt is sent to the computer

- Paper Out

A contact closure is available when the printer is out of paper. When the contact

closes, the Printer Control Unit waits until the bottom of the form is detected and then sends an interrupt to the computer

- Paper Break

A contact closure is available when the paper is not correctly positioned between the feed tractors. When the contact closes, an interrupt is sent to the computer

- Throat Cover Open

A contact closure is available when the throat cover is not correctly positioned on the printer. When the contact closes, an interrupt is sent to the computer

- Print Wheel Position

The print wheel position is defined by two output signals, one providing a marker pulse once per print wheel revolution (Index pulse) and the other providing a marker pulse once per character position of the print wheel (Character pulse). The Index pulse is set to -6 volts for 12 microseconds, sometime during the interval between the character pulses associated with the codes 77 and 00. The Character pulse is set to -6 volts for 12 microseconds when each character on the print wheel is in the proper position for printing. There is one Character pulse per character position on the print wheel, or 64 Character pulses per revolution of the print wheel

- Paper Position

The paper position is defined by an output signal providing a marker pulse for each line of paper movement (called a Strobe pulse). This signal is set to -6 volts for 12 microseconds each time the paper is advanced vertically by one line, and is always at zero volts when the paper is stopped

COMPUTER CONTROL

The external function word serves three functions:

- To enable single line feed
- To disable single line feed
- To position paper to the top of the next form

Three bits of the external function word signify which of these operations is to be performed

(see Figure 5). When the printer is set to the Enable Line Feed condition, the paper automatically is advanced by one line after each line of print.

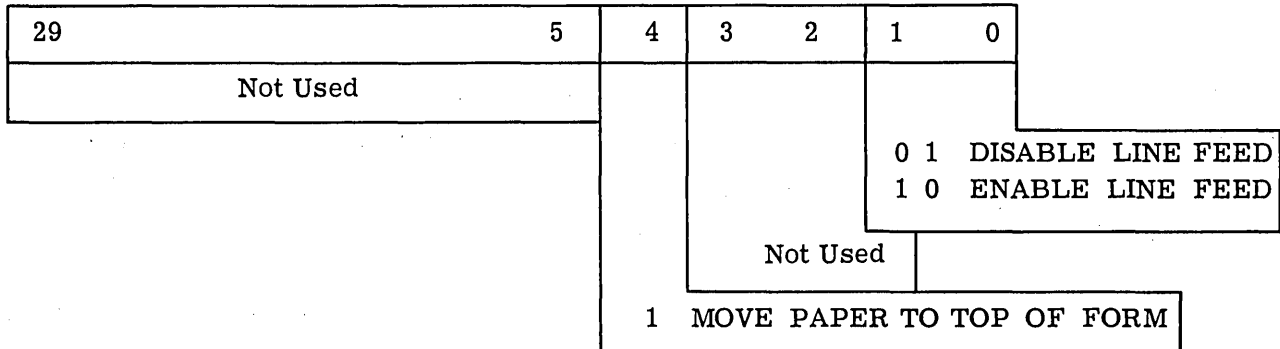


Figure 5. External Function Word

When the printer is set to the Disable Line Feed condition, the paper is not advanced. Master Clear sets the printer to an Enable Line Feed condition. The Top of Form command positions the paper so that the type line is at the top of the next form. If this command is sent when the printer is in a Disable Line Feed condition, or is sent together with a Disable Line Feed command (external function word equal to 21), the paper will not be moved.

The exact position of the top of the next form is determined by the punch tape control loop. Three tape tracks are used on the control loop to control vertical positioning of the paper. The first track will have a hole punched at a position relative to the top of the form. This is used to stop paper when it is advanced by the top of form command. The second track has a hole punched so that it will set the tape feed mechanism in a position for loading paper. This is used in conjunction with the Load Paper button on the Control Panel (Figure 6). The third track has a hole punched at a position relative to the bottom of the form. This hole is used to advance paper to the top of the form and to stop the paper when the paper supply is out.

When the printer goes from on-line to off-line or off-line to on-line one of two interrupts is sent to the computer from the printer: a malfunction interrupt or a start interrupt. The malfunction interrupt is sent from the printer to the computer when any of the following malfunctions exist:

- Printer is out of paper
- Paper is torn or incorrectly positioned in the feed tracks
- Throat Cover is not in position

- There is a loss of AC power input
- A main circuit breaker is open
- A fuse is blown

A status word of zero is sent to the computer. The "out of paper" malfunction interrupt does not terminate the print operation until the bottom of the current form is detected. The start interrupt is sent to the computer when the printer is Master Cleared and set to operate ON-LINE. A status word of one (bit zero set) is sent to the computer.

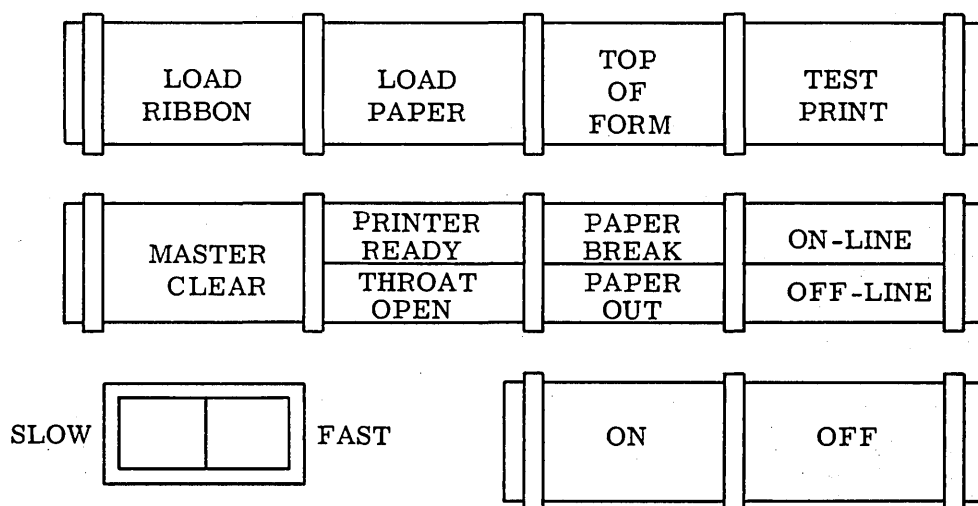


Figure 6. Control Panel

PROGRAMING IMPLICATIONS

GENERAL

Certain programing functions are of importance to the programer. These functions consist of the following:

- Processing interrupts
- Character positioning
- Assembly of buffers
- Magnetic tape operation
- Sending External Function to the printer

The programer must know how these functions are performed and what they accomplish before programing for the printer.

PROCESSING INTERRUPTS

The program should distinguish between the malfunction and the start interrupts. The reception of the start interrupt indicates to the program that the printer is available for an output. The start interrupt is used to initiate the printer program operation; it should be utilized whenever possible. If it is not possible to utilize the start interrupt, it can be ignored; however, the programmer must be certain that the printer is operable before he sends an output. This precaution is necessary because if the output is an External Function command it will not be performed and will be lost if the printer is not operable. The first output to the printer is normally an External Function Top of Form command, and if this is lost, the forthcoming data may be printed in the wrong position on the print form.

The malfunction interrupt indicates to the program that the printer is not operable. Normally, the malfunction condition will destroy some data, and the program will have to repeat some or all of the previous output operations. The exception to this is the malfunction interrupt caused by the Paper Out condition. This condition will terminate printer operation when the bottom of the last form is reached. Therefore, the program can continue its output when new paper has been loaded, and the paper has been positioned to the top of form position, without a resultant loss of data. When a malfunction interrupt is received, the programmer may want to provide a means of making an operator decision regarding whether the program can continue, or some or all of the data will have to be reprinted. Here again, the programmer should utilize the start interrupt, if possible, to restart the program. Before the printer is initially put ON-LINE, the program should be ready to process either of the two types of interrupts mentioned above, or a fault condition will result at the computer.

CHARACTER POSITIONING

Horizontal and vertical character positioning is accomplished in two ways: with an external function command, and/or by programming. The external function command is used for vertical positioning. The Top of Form command positions the top of form position on the paper under the print hammers. The top of form position is determined by the tape loop. The enable and disable line feed commands enable the program to print on a line more than once. When the printer is Master Cleared, the line feed is enabled. The paper is fed after the line is printed. The amount of time between initiation of the output buffer and execution of the line feed depends on the type of data being printed. Therefore, if a line is to be printed more than once, the disable line feed command should be sent after the line of data that is to be printed over. The enable line feed command must be sent after the last line of data that is printed

over the previous data. The program controls character position with space codes. Vertical line positioning is controlled by sending lines of space codes. Horizontal character positioning is controlled by filling in the character positions that are not to be printed with space codes.

ASSEMBLY OF BUFFERS

The printer will not print a line of data until 120 printer codes (24 computer words) have been received, or less than 120 printer codes are followed by a top of form command. Therefore, if only the first 60 character positions of a line of print are to contain data, the other 60 positions must be filled in with space codes unless the line is followed by a top of form command.

MAGNETIC TAPE OPERATION

Data sent to the printer from the computer can also be assembled on magnetic tape and sent to the printer OFF-LINE in the same way, with one exception: the use of the enable and disable line feed external function commands. It is not possible to use these commands on magnetic tape since the top of form command is the only external function available in the magnetic tape operation. Therefore, if a program is to be compatible with both computer output and magnetic tape output to the printer, provision must be made in the program to handle an enable or disable line feed condition.

SENDING EXTERNAL FUNCTION TO THE PRINTER

An external function to the printer must be programmed with force as follows:

EX-COM • CN • W(PRT) • FORCE

where CN is the printer channel number, and PRT = 00000 00002, the enable line feed function word.

OPERATOR CONTROL

PRINTER OPERATIONS

The following controls are available on the printer:

- Vertical Position

Controls the vertical positioning of the paper with relation to the printed line. The maximum adjustment is ± 0.25 inch

- Character Phasing

Adjusts the index and character pulse timing to compensate for print wheel speed. The maximum adjustment is 15 degrees

- Paper Tension

Controls the paper tension by varying the distance between corresponding feed pins on the upper and lower feed tractors. The maximum adjustment is 0.125 inch

- Penetration Control

Provides fine adjustment of the spacing between the print hammers and the print wheel in order to vary the density of print and accommodate different paper thicknesses

CONTROL PANEL OPERATIONS

Various operations are initiated from the Control Panel by the following indicators and switches:

- POWER ON Indicator Switch

Initiates the power sequencing cycle and indicates completion of the cycle

- POWER OFF Indicator Switch

Turns AC power off. The switch lamp is illuminated when AC power is available at the output of the main circuit breaker and a power sequencing cycle has not been completed

- OFF-LINE/ON-LINE Indicator Switch

Controls the logical connection of printer to computer. When the printer is Master Cleared and ON-LINE is selected, an interrupt is sent to the computer

- SLOW/FAST Switch

A rocker type switch that changes the print wheel speed from 1000 to 667 rpm

- LOAD PAPER Switch

Positions the paper feed assembly so that paper can be more conveniently loaded. This operation is only available when the printer is set to operate OFF-LINE

- **LOAD RIBBON Switch**

Respools the inked ribbon prior to its replacement. This operation is not available while the printer is printing

- **TEST PRINT Switch**

Actuates a test cycle under local control that utilizes a pattern which prints all characters in all column positions. When the Pattern Select switch on the control chassis is in the normal position, the test cycle causes all 63 characters to appear in all column positions. In any given line, one character will appear in all of the 120 character positions, and after 63 lines of print a line of each character has been printed. When the Pattern Select switch is in the M position, all M's are printed, and when it is in the E position, all E's are printed. The paper automatically advances one line after each line of print. The operator depresses the TEST PRINT switch to terminate the test operation. This operation is only available when the printer is set to operate OFF-LINE

- **TOP OF FORM Switch**

Advances the paper so that the top of the next form is positioned at the type line. This operation is only available when the printer is set to operate OFF-LINE

- **MASTER CLEAR Switch**

Clears the circuit logic, sets Enable Line Feed, sets the interrupt, and places the printer OFF-LINE

- **PAPER OUT Indicator**

Indicates that the paper supply is exhausted. This situation is detected 22 inches ahead of the type line when the throat cover is closed

- **PAPER BREAK Indicator**

Indicates absence of paper above the print station

- **THROAT OPEN Indicator**

Indicates that the printer throat cover is not in position. A switch is available for overriding the "throat open" indication

- **PRINTER READY Indicator**

Indicates that the printer is ready for operation. The indicator goes out when any of the following conditions exist:

- The printer is out of paper
- The paper is torn or incorrectly positioned in the feed tractors
- The throat cover is not in position
- There is no AC power input
- A main circuit breaker is open
- A fuse is blown

CONTROL CHASSIS OPERATIONS:

The Pattern Select switch selects one of three patterns that can be used in the test print operation:

- Normal Position - print all characters in the order they appear on the print wheel
- M Position - print all M's
- E Position - print all E's

OFF-LINE COMPATIBILITY

The magnetic tape subsystem is capable of communicating directly with the high-speed printer subsystem for OFF-LINE operation. The interface between the magnetic tape and printer subsystem is shown in Figure 3.

The printer output is connected to the Magnetic Tape System's input (input and output as used here are in reference to the computer).

The data to be printed OFF-LINE must be recorded in records on tape in the following format: record length of 120 Fielddata characters (24 words).

The Magnetic Tape System will read each record of data from tape in the following format: Modulus 5.

Each record of 120 characters will form 24 30-bit data words which will be printed as one line by the high-speed printer.

A tape mark will be recognized by the high-speed printer as a top of form command. This will position the paper to the top of the next page.

A record of less than 24 words (preferably one computer word, five characters) will cause the high-speed printer to stop the printing operation. This record will not be printed if the record contains space codes.

With the Magnetic Tape System switched to the Printer Mode, the desired tape transport selected, and the tape positioned at load point, the high-speed printer will initiate operation when it is placed in the ON-LINE position.

The normal sequence of events for transfer of data from the Magnetic Tape System to the printer is as follows:

1. The printer sets its Output Data Request
2. The Magnetic Tape System in the idle state, recognizes this first Output Data Request from the printer as an external function and starts the read operation
3. The Magnetic Tape System places the information on the data lines and sets its Input Data Request
4. The printer recognizes the Magnetic Tape System's Input Data Request as an Output Acknowledge
5. The printer samples the data lines and clears its Output Data Request
6. The Magnetic Tape System recognizes the clearing of the printer Output Data Request as an Input Acknowledge

Steps 3, 4, 5, and 6 are repeated for each word of the record.

The normal sequence for sending an external function command top of form from the Magnetic Tape System to the printer is the same as reading a record except that when the Magnetic Tape System detects the Tape Mark, it will set bit 4 in the Status word, and when the interrupt is set, the printer will recognize this as an external function command top of form.

GLOSSARY

ABSOLUTE FORMAT: A form of assembled program where all addressing is expressed in actual computer addresses

ACCESS: Used as a verb in computer terminology, meaning to obtain or procure; to read or sense the information in a register or storage device

ACTION OPERATIONS: The operations which act upon or with defined data to produce object language (L_4) instructions

ALLIED OPERAND: In mono-coding, the specially designated operand V_0 , which modifies and completes the definition of multi-purpose mono-operators; it names the register, r , or the expression, e , which together with the operator identifies the specific machine function code, (f)

ALLOCATION: The process of assigning numeric values, usually representing absolute computer addresses, to symbolic (relative) labels and tags; the assignment may be done 1) manually by the programmer, 2) by the computer via allocation tapes, and 3) independently and internally by the assembler (assembler-generated tags)

BIOCTAL CODE: An abbreviated means of paper tape storage where each tape frame represents two octal characters, thus reducing the size of tape considerably. Addressing in bioctal is absolute

BOOT STRAP:

1. The process of manually entering a routine into the computer to perform some task, such as loading the AS-1 Utility Package
2. A routine which is manually entered into the computer to perform some task
3. A routine wired semipermanently into computer to load tapes

CHANNEL: A means of transmission of electronic data or information, physically consisting of one or more lines; used with external equipment

CHECK SUM: A safety check after a program has been debugged and is operational. It ensures that a program has been loaded into memory. It generally consists of a cumulative sum of the units (holes in paper tape) which the computer senses and compares with the corresponding value obtained during the tape preparation. Separate counts on the upper and lower halves of the words in memory are usually made. To indicate a memory or Photoelectric Reader fault, "CHECK SUM ERROR" printed on the typewriter is all that is necessary. This procedure is currently employed in most program-load operations

COMBINED OPERAND: (See *OPERAND POSITION*)

ASSEMBLER-GENERATED TAGS: A system of unique, 10-character tags, generated and allocated internally by the assembler and used by it to build programs with symbolic addressing without the aid of a programmer. The general typewriter format of these tags is A |||| nnnn, where n is any AS-1 code from 1 through Z and the four leftmost bars, |, occupy positions which may become AS-1 characters from A through Z

Certain poly-operations produce intermittent instructions, or groups of instructions, within the object program; the assembler uses these assembler-generated tags to make possible the referencing of, and interaction between these instructions

ASSEMBLER INPUT LANGUAGE: A set of symbolic notations, called operations, which present meaningful program input to the assembling system. The input language is divided into two categories of operations. These are:

1. Programming language
2. Assembler-control language

AS-1 UTILITY PACKAGE: A service program used primarily to load data into and dump data from the computer. It may also be used to punch out an area of core storage, or inspect a storage address and alter this if desired

DEBUGGING AIDS: In the AS-1 system, a series of special routines designed to aid the programmer in locating errors of various types in his program while it is being executed. These routines are of two types: 1) initializing routines which delimit areas or blocks of information for later reference and manipulation; and 2) data testing routines which perform such functions as giving contents of selected registers at strategic locations in the program or following the course of a program within defined limits

DECLARATIVE OPERATION: An operation which does not result in an object program instruction, but provides information for the assembler's use in constructing the object program by stating certain facts about entities within the assembler in the same language as that of the input. These operations 1) adapt the program to a specific memory configuration and input-output capabilities of the computer, and 2) identify different segments of input to the computer

EXTERNAL FUNCTION: In general, an operation or procedure dealing with the giving of orders to external equipment by means of the function channels; also the specific operator, **EX-COM** or **EX-COM-MW** designed to perform these commands within the assembler

HEADER: The initial operation item on paper or magnetic input tape placed there for the purpose of identification of the contents of the tape, e.g., Program, Allocation, Correct-L₁

HEADER-TYPE OPERATOR: The declaration-type word in the operator position of the header on a tape which identifies the type of routine or program on the tape

IMAGE: An exact copy or duplication of the contents of a program or portion thereof in memory, reproduced in another portion of core memory

INPUT PARAMETERS: The requirements, in the form of items of information or units of data, which must be entered into the computer for use in conjunction with a subroutine, and which describe, define, or limit the subroutine

LABEL: The name preceding the operator which uniquely identifies an operation; it consists of up to ten alphanumeric characters, but must never start with 0, X, or a number, nor may it consist of only A, Q, B⁰ through B⁷, or C⁰ through C¹⁷. The following operations require labels:

1. Any operation referred to by a tag in another operation
2. Each **ENTRY** operation (a subroutine entry)
3. Allocation operations
4. The first action operation following a break in the program sequence

LAB/TAG: A symbolic name, either a label or a tag

LANGUAGE LEVELS (L₀, L₁, L₂, L₃, L₄): The successive forms of symbolic expression through which an assembler proceeds in its translation of a program from the reference

language (the language in which the programmer composes the program) to the object language (the symbolic form in which the machine can accept the program). The AS-1 Assembler system involves five language levels, termed L_0 , L_1 , L_2 , L_3 , and L_4

- L_0 - The language of the programmer consisting of English words, expressions using the English alphabet, and numeric and algebraic terms, in which a problem is defined
- L_1 - The first of the intermediate language levels. L_0 is converted into a standardized AS-1 code and the words of each statement have been identified and assigned to specific locations in an item according to their role in the operation. A printout of this level would appear virtually the same as L_0 because the AS-1 code must necessarily be reconverted to the L_0 format
- L_2 - The second intermediate AS-1 language level. The L_1 (or L_0) format has been translated into machine code with the exception of the address structure which appears in symbolics (tags and labels)
- L_3 - The final machine code stored on tables within the assembler. Essentially it is the same as L_2 , with the symbolic addressing converted to absolute machine addresses
- L_4 - The object program after it has been produced as output in a form readable to the computer as input, e.g., punched paper tape, or buffered into core memory

L_1 CORRECTOR: A routine with which corrections can be made to an input program while at language level L_1 of the AS-1 Assembler. The correction items are entered into the computer by a separate correction tape. Positioning of the corrections is accomplished by means of L_1 identifiers. Three kinds of operation corrections can be made: insertions, deletions and replacements

L_1 IDENTIFIER: The successive numbering of the operations by the assembler, beginning with 0 as the header operation. This numerical value precedes each operation, and is used by the L_1 Corrector for the selection of a statement

MNEMONIC OPERATIONS: The English-type statements used by the programmer in the preparation of a program or routine; the L_0 language of the AS-1 Assembler

MONO-OPERATIONS: The class of programmer-written mnemonic statements, each of which produces one machine instruction when translated by the AS-1 Assembler. The operators of these operations, together with their allied operands, correspond, with some exceptions, to the 62 machine function codes. (See *POLY-OPERATIONS*)

MONO-CODE: The operator portion of a mono-operation; it identifies the type of function the operation performs, e.g., **JP** , **ENT** , **STR**

NORMAL *j*-OPERANDS: The group of *j*-designator operands, including **SKIP**, **QPOS**, **QNEG**, **AZERO**, **ANOT**, **APOS**, and **ANEG**, used with the majority of mnemonic operations. (See *SPECIAL *j*-OPERANDS*)

NOTES: The portion of an operation which is used for optional notations for the programmer's information only. Notes are nonfunctioning, that is, they do not affect the functions of the operation

OBJECT PROGRAM: The ready-to-run program, on paper tape, consisting of assembled machine code instructions produced as output by the assembler

OFF-LINE: Of or pertaining to external equipment not connected to the computer by direct circuitry; operations or functions performed by such equipment

ON-LINE: Of or pertaining to equipment external to the computer but connected to it by direct circuitry; operations or functions performed by such equipment

OPERAND: A single or multi-word expression which modifies or explicitly defines the operator of a statement by stating items of information or defining parameters. (See *OPERAND POSITION*)

OPERAND-CODE: The first symbolic term, *a*, within an operand position, consisting of up to ten characters and adding some basic information to the operator or to the terms following it. It specifies a *k*-designator, a name, a location, or an area; it may appear alone, with other items, or be completely absent

OPERAND POSITION: The term used to describe a class of operands (one or more) combined to serve a like function and included under the symbol, $-V_n-$

OPERATIONS: The basic units of assembler input consisting of labels, statements and notes, of which labels and notes are optional. The three categories are:

- 1) Assembler language operations
- 2) Assembler-control operations
- 3) Program corrections

OPERATION ITEM: A unit of program storage in the AS-1 Assembler. An operation item must include an operation statement (operator and operands) and may or may not include a label and/or notes. It may also be defined as one complete AS-1 program operation in L_1 , L_2 , or L_3 storage

OPERATION STATEMENT: (See *STATEMENT*)

OPERATOR: The initial word of a statement; it specifies the basic operational characteristics of the statement

OUTPUT CONVERSION: The process of converting (translating) internal assembler-stored data into some language which is compatible with external output equipment and is also meaningful to the programmer

POINT SEPARATOR: A separator symbol corresponding to a multiply sign, used in coding to separate the words of a statement

POLY-CODE: The *operator* of a poly-operation. Operators of this type denote the production of one or more instructions in the object program, or make declarations to the assembler

POLY-OPERATIONS: A category of mnemonic operations with which one or more machine code instructions can be produced in an object program. Poly-operations are considered as "one-to-many" operations as opposed to "one-to-one" for mono-codes. In other words, a poly-operation generally has the ability of producing a series of machine code instructions in the output, or object program, while mono-instructions can produce only one

RELATIVE FORMAT:

1. An object program format, compatible with AS-1 Utility Package loading, in which the initial address is relatively allocated to zero with all other addresses allocated successively unless specifically allocated to an absolute address. Upon loading a program of this format, the Utility Package is to be instructed to increment all relatively allocated addresses by an amount which equals the initial address desired.

Thus, a relative format program may be loaded at any portion of core storage (1000 or above)

2. A program written in modified machine code with symbolic addressing

ROUTINE: A generalized term which defines a stretch of coding that performs a particular task. This may encompass either the routine or subroutines of a program; in some cases it may constitute the whole program

SEPARATORS: A set of symbols used in coding to represent input control for operations; e.g., a **➡** symbol always precedes an operation statement and follows the statement when notes are included

SIGNIFICANT LABELS: Those labels which are accompanied by a mnemonic operation or numeric instruction that infers the beginning of a subroutine or program, i.e., 61000 00000, 60100 00000, or 00000 00000. In addition it includes all labels of operations with the operators **U-TAG** or **EQUALS**

SIMPLE STATEMENT: A complete statement which does not require the use of connectors

SOURCE PROGRAM: The program of mnemonic instructions written by the programmer and used as input to the assembler; also known as the L_0 program or Input Language

SPECIAL j -OPERANDS: Those mnemonic j -operands which pertain to operations which require the use of unique j -designators. Most operations which employ the j -designator use j -operands of a similar nature called *normal j -operands*. Operations which require *special j -operands* are so identified in the discussions on the individual mono-operations, (see also MONO-OPERATIONS, *SPECIAL j -OPERANDS*)

STATEMENT: The portion of an operation which defines its function and is composed of an operator and all operands

SUBROUTINE: A specific kind of routine in which the entrance and exit are at the same point. A subroutine usually begins with the operation **JP • 0** which also serves as its exit, since it is called upon by a ReturnJump operation (see **RJP** operation, MONO-OPERATIONS). Subroutines may be considered as an extension of or annex to the main program routine

SYMBOLIC ADDRESS: A symbol, usually alphanumeric, representing a computer address

SYMBOLIC ADDRESSING: The process of using relative symbolic addresses in the form of tags and labels, to represent computer addresses while coding or in assembler-generated tags. These are later assigned to an absolute value by the allocation routine

TAG:

1. The symbolic expression used in relative programming in place of an absolute computer address. With the exception of the poly-code **U-TAG**, these tags appear in the y-operand position and usually refer to a label or other relative symbolic address
2. An alphanumeric or alphabetic symbol which refers to, or can be allocated to, some absolute address

ABSOLUTE CODING AND DIRECT CONSTANT ENTRIES

ABSOLUTE CODING

The AS-1 Assembling System permits coding with absolute machine instructions. The programmer enters $f j k b$ and y designators following the \Rightarrow symbol on the coding form.

Addressing of such instructions *must be* symbolic and conform to the requirements of AS-1 labels and tags. Label placement is in the conventional LABEL coding position. (The first instruction must be labeled.) The y portion of the instruction is either a symbolic tag, a tag \pm increment, or a constant; when y is a *constant* of less than five digits, only the digits used are given following a point separator (e.g., the constant 00007 is coded as $\bullet 7$).

Absolute coding permits the inclusion of *notes* with instructions; these follow a second straight arrow (\Rightarrow) symbol placed on the coding form.

Allocation of the lab/tags in absolute coding conforms to the rules of AS-1 allocation (see ALLOCATION).

DIRECT CONSTANT ENTRIES

The programmer is permitted to make direct constant entries while coding. AS-1 permits either octal or decimal constants, decimal digits being following by a D. The maximum decimal number permitted is 536870911D.

Normalizing of direct constant entries is to the right with non-used upper digit positions being filled with zeros. By use of a point separator, normalizing is permitted in both the upper and lower halves of a storage location. For example, a coded entry of $\Rightarrow 43 \bullet 11$ produces 00043 00011 in storage.

INPUT/OUT CODING

Input/output instructions may appear randomly in the program. In order to clarify the usage of input/output instructions, a description of communicating with the computer is given in four examples which follow.

Examples 1 and 3 describe the use of a pseudo programming technique to accomplish input and output. Examples 2 and 4 are written with absolute channel assignments. An important advantage of the use of the symbolic method of programming input or output is that it is not channel sensitive; that is, if the channel assignment is changed for any peripheral device, only the MEANS cards for the device affected need be changed in order to reflect the new channel assignment throughout the program. In contrast, a program written with absolute channel assignments is channel sensitive; that is, whenever the channel assignment is changed for any peripheral device, each program location which references the affected device must be changed.

In Examples 1 and 3, the MEANS operator is used to define the channel sensitive labels. No coding is generated for MEANS statements.

The normal method of programming input/output is to give the transfer instruction before enabling the particular device; this is to ensure transfer readiness regardless of timing. Disregarding interrupts from other peripheral devices, the transfer instruction could be written before the enabling instruction. Some devices, however, do dictate a specific method to be used to sequence the input/output instructions.

The transfer buffer limits are defined at the address given in the V_1 operand of the transfer instruction. At this address, the buffer limits may be defined 1) absolutely, or 2) by use of a U-TAG operator if symbolic coding is used. The upper 15 bits define the terminal address and the lower 15 bits the initial address.

The EX-COM operator is used to enable or disable the peripheral device. The function code which tells the particular peripheral device what to do, is located at the address defined by the V_1 operand.

Unless the transfer is accomplished with internal interrupt, MONITOR, a wait instruction sometimes should be programmed to prevent the execution of other instructions until the transfer is complete. This allows the programmer to control the transfer.

It should be noted that the label defined by the operand in the enabling or disabling, or the transfer instruction may appear either before or after the instruction in which it is used. Similarly, the transfer buffer may be located anywhere in memory and either a decimal or octal number may be used with the U-TAG or RESERVE operators.

Example 1:

	EXAMIN	PROGRAM • CBROWN • MARCH66
	TAPE	MEANS • C0
	TAPEIN	MEANS • C0ACTIVEIN
01000 73030 01235		IN • TAPE • W(READ) ➡ INPUT TRANSFER
01001 13030 01234		EX-COM • TAPE • W(SWITCH) • FORCE ➡ ENABLE READER
01002 62000 01002	WAIT	JP • WAIT • TAPEIN ➡ WAIT FOR TRANSFER COMPLETION
.	.	.
.	.	.
.	.	.
01171 13030 01233		EX-COM • TAPE • W(SWITCHOF) • FORCE ➡ DISABLE READER
.	.	.
.	.	.
.	.	.
01233 00000 00010	SWITCHOF	0 • 10 ➡ DISABLE READER CODE
01234 00000 00150	SWITCH	0 • 150 ➡ ENABLE READER CODE
01235 02024 02000	READ	U-TAG • BUFLO+20D • BUFLO ➡ INPUT BUFFER LIMITS
.	.	.
.	.	.
.	.	.
02000	BUFLO	RESERVE • 21D ➡ INPUT BUFFER
.	.	.
.	.	.
.	.	.
02024		

Example 2:

01000 13030 01234		EX-COM • C0 • W(SWITCH) • FORCE ➡ ENABLE READER
01001 73030 01235		IN • C0 • W(READ) ➡ INPUT TRANSFER
01002 62000 01002	WAIT	JP • WAIT • COACTIVEIN ➡ WAIT FOR TRANSFER COMPLETION
.		.
.		.
.		.
01171 13030 01233		EX-COM • C0 • W(SWITCHOF) • FORCE ➡ DISABLE READER
.		.
.		.
.		.
01233 00000 00010	SWITCHOF	0 • 10 ➡ DISABLE READER CODE
01234 00000 00150	SWITCH	0 • 150 ➡ ENABLE READER CODE
01235 02024 02000	READ	02024 • 02000 ➡ INPUT BUFFER LIMITS

Example 3:

	EXAMOUT	PROGRAM • JOELGOODGUYMARCH66
	PERFORM	MEANS • C7
	PUNCHOUT	MEANS • C7ACTIVEOUT
		.
		.
		.
10050 13370 13210		EX-COM • PERFORM • W(SWTHP) • FORCE ➡ ENABLE PUNCH
10051 74370 12311		OUT • PERFORM • W (BUFOUT) ➡ OUTPUT TRANSFER
10052 62000 10052	INSTEP	JP • INSTEP • PUNCHOUT ➡ WAIT FOR TRANSFER COMPLETION
.		.
.		.
.		.
13101 13370 13207		EX-COM • PERFORM • W(SWITHO) • FORCE ➡ DISABLE PUNCH
.		.
.		.
.		.

Example 3 Continued

13207 00000 00001	SWTHO	0 • 1 ➡ DISABLE PUNCH CODE
13210 00000 00005	SWTHP	0 • 5 ➡ ENABLE PUNCH CODE
13211 15023 15000	BUFOUT	U-TAG • PUNBUF+19D • PUNBUF ➡ TRANSFER BUFFER LIMITS
.	.	.
.	.	.
15000	PUNBUF	RESERVE • 20 D ➡ OUTPUT BUFFER
.	.	.
15023		

Example 4:

10050 74370 13211		OUT • C7 • W(BUFOUT) ➡ OUTPUT TRANSFER
10051 13370 13210		EX-COM • C7 • W(SWTHP) • FORCE ➡ ENABLE PUNCH
10052 62000 10052	INSTEP	JP • INSTEP • C7ACTIVEOUT ➡ WAIT FOR TRANSFER COMPLETION
.	.	.
.	.	.
13101 13370 13207		EX-COM • C7 • W(SWTHO) • FORCE ➡ DISABLE READER
.	.	.
.	.	.
13207 00000 00001	SWTHO	0 • 1 ➡ DISABLE PUNCH CODE
13210 00000 00005	SWTHP	0 • 5 ➡ ENABLE PUNCH CODE
13211 15023 15000	BUFOUT	15023 • 15000 ➡ TRANSFER BUFFER LIMITS

POLY-OPERATIONS

SYMBOL LEGEND

SYMBOL	MEANING
α	Read-Class Operand
β	Store-Class Operand
Ξ	Replace-Class Operand
ξ	Tag, Number, or Tag \pm Number
Σ	Same as ξ , except a B register can also be specified
ζ	Operand-Code Only
σ	Operand-Code or Number
θ	Number Only

Poly-Operations

OPERATOR		OPERAND SECTION								DESCRIPTION	
Sep	W	Sep	V_0	Sep	V_1	Sep	V_2	Sep	V_3	Sep	
➔	CLEAR	•	number of words ^{α}	•	base addr of area ^{Σ}	➔					Clears V_0 words starting at address V_1
➔	ENTRY	• or ➔	key set condition ^{ζ}	➔							Establishes the entry of a subroutine. Stop condition is effective on exit
➔	EXIT	• or ➔	jump condition ^{ζ}	➔							Establishes an exit via the preceding ENTRY if V_0 is satisfied
➔	INCREMENT	•	B register	•	increment ^{α}	➔					The B register, specified by V_0 is modified by V_1
➔	MOVE	•	number of words ^{α}	•	base address of area ^{Σ}	•	base address of area ^{Σ}	➔			Moves V_0 words from address V_1 to address V_2
➔	PUT	•	numeric value ^{α}		address ^{β}	➔					Stores the word V_0 at address V_1
➔	TYPEPC*	•	location or Flex function ^{α}	➔							Types content of location or function expressed by V_0
➔	TYPET	•	message	➔							Types message that follows the operator

* V_0 may be a list of similar operands

Debugging Operations

OPERATOR		OPERAND SECTION								DESCRIPTION		
Sep	W	Sep	V_0	Sep	V_1	Sep	V_2	Sep	V_3	Sep		
➔	CORE-IMAGE	•	name of area	ξ	•	base addr. of image	ξ	•	key set condition	ξ	➔	Images the area V_0 in core at address V_1 if V_2 is satisfied
➔	DUMP-AREA	•	area name(s)	ξ	•	key set condition	ξ	➔				Dumps all non-zero numbers in the areas listed, if V_1 is satisfied
➔	DUMP-REG	•	key set condition	ξ	➔							Dumps A, Q, B ₁ ---B ₇ if V_0 condition is satisfied
➔	TEST-IMAGE*	•	area name(s)	ξ	•	key set condition	ξ	➔				Compares the areas listed with their image and dumps any differences if V_1 is satisfied
➔	DEF-AREA	•	name of area	ξ	•	base addr. of area	ξ	•	number of words	ξ	➔	States that the area named V_0 starts at address V_1 and contains V_2 words

* V_0 may be a list of similar operands

General Declarative Operations

OPERATOR		OPERAND SECTION								DESCRIPTION	
Sep	W	Sep	V_0	Sep	V_1	Sep	V_2	Sep	V_3	Sep	
➔	ALLOCATION	•	program- mers name	•	date	➔					Header operation, specifies an allocation tape
➔	CORRECT- L_1	•	program- mers name	•	date (new)	➔					Header operation, specifies a correction tape
➔	EQUALS	•	address ξ	➔							Must have LABEL, assigns the LABEL a known value V_0
➔	MEANS	•	mnemonic ζ code	➔							Must have a LABEL, assigns a machine oriented value to the LABEL
➔	PROGRAM	•	program- mers name	•	date	➔					Header operation, specifies program input

CODE EQUIVALENCE CHART

NAME	CODING SYMBOL	FLEXOWRITER	PUNCHED CARD	CARD TO TAPE CONVERSION	EDIT CARDS H S PR	CS-1 CODE	EDIT CS-1 OUTPUT H S - PR
Letters	A - Z	A - Z	A - Z	A - Z	A - Z	A - Z (except 0)	A - Z
Numbers	0 - 9	0 - 9	0 - 9	0 - 9	0 - 9	0 - 9	0 - 9
Plus	+	+	+	&	&	+	+
Minus	-	-	-	-	-	-	-
Point Separator	•	.	*	*	*	.	*
Period
Comma	,	,	,	,	,	,	,
Left Paren	(((%	%	((
Right Paren)))	:	:))
Straight Arrow (first)	➔	(Tab)	(Column Position)	(Word Position)	(Spaces)	(Word Position)	(Spaces)
Straight Arrow (second)	➔	(Tab)	- - - -	Ⓞ Ⓞ Ⓞ	(Spaces)	a a a	(Spaces)
Curved Arrow	↵	(Carriage Return)	(Release Card)	(Block Space)	(Next Line)	(Not Used)	(Next Line)
Zeta			\$	\$	\$	ζ	\$

CONSTANT POOL

AS-1 establishes a *constant pool* for each assembled program requiring constants of a special nature. This *pool* is a list of constants which are specified in the Read-class operand position and are more than five octal digits. Such constants are stored only once, even though many operations refer to the same constants. This *pool* is positioned at the end of the program.

Example:

Section of Input Problem

```
CRWD  ➡ ENT • A • 5760435
      ➡ _____
      ➡ _____
BLK   ➡ ADD • Q • 24356105
      ➡ _____
      ➡ _____
      ➡ _____
      ➡ PUT • 5760435 • W(CAT)
      ➡ _____
```

Corresponding Section in L2 (Mnemonic Form)

```
CRWD  ➡ ENT • A • A|||||11Z8*
      ➡ _____
```

→ _____
BLK → ADD • Q • A| | | | 11Z9
→ _____
→ _____
→ _____
→ ENT • Q • A| | | | 11Z8
→ STR • Q • W(CAT)

At the End of the Program

A| | | | 11Z8 0005760435
A| | | | 11Z9 0024356105

FLEXOWRITER CODE CHART

The upper case, UC, or lower case, LC, character is typed according to the position of the type bars.

TYPE LETTER		OCTAL	TYPE LETTER		OCTAL	PERFORM TYPE-WRITER OPERATION	OCTAL	
UC	LC		UC	LC				
A	a	30	1	1	52	Space	04	
B	b	23	2	2	74	Shift up	47	
C	c	16	3	3	70	Shift down	57	
D	d	22	4	4	64	Back space	61	
E	e	20	5	5	62	Car. return	45	
F	f	26	6	6	66	Tabulator	51	
G	g	13	7	7	72	Color shift	02	
H	h	05	8	8	60	Code delete	77*	
I	i	14	9	9	33	Stop	43*	
J	j	32	0	0	37			
K	k	36						
L	l	11	TYPE SYMBOL					
M	m	07	UC		LC		OCTAL	
N	n	06	-	(Superscript Minus)	-	(Hyphen or Minus)	56	
O	o	03	.	(Multiply)	=	(Equals)	44	
P	p	15	/	(Virgule)	+	(Plus)	54	
Q	q	35	((Open Parens)	,	(Comma)	46	
R	r	12)	(Close Parens)	.	(Period)	42	
S	s	24	_	(Underline)		(Absolute)	50	
T	t	01						
U	u	34						
V	v	17						
W	w	31						
X	x	27						
Y	y	25						
Z	z	21						

NOTE: Codes not used are: 00, 10, 40, 41, 53, 55, 63, 65, 67, 71, 73, 75, 76.

*Codes 43 and 77 are considered illegal codes in operations with the computer

AS-1 CHARACTER CODE (6 Bits)

	0	1	2	3	4	5	6	7
		DATA CONTROL CHARACTERS						
0	i	Δ	α	β	γ	δ	ε	ζ
1	0	1	2	3	4	5	6	7
2	8	9	A	B	C	D	E	F
3	G	H	I	J	K	L	M	N
4	NOT USED	P	Q	R	S	T	U	V
5	W	X	Y	Z	()	NOT USED	NOT USED
6	-	+	*	/	NOT USED	NOT USED	NOT USED	NOT USED
7	NOT USED	NOT USED	NOT USED	NOT USED	.	,	NOT USED	NOT USED

i = ignore code

Δ = space

The data control characters act as sentinels.

TELETYPE CODE CHART

TELETYPE CODE (TEXT TO CODE CONVERSION)

LTRS	FIGS	OCTAL CODE
A	-	30
B	?	23
C	:	16
D	\$	22
E	3	20
F	:	26
G	&	13
H	#*	05
I	8	14
J	,	32
K	(36
L)	11
M	.	07
N	,	06
O	9	03
P	0	15
Q	1	35
R	4	12
S	Bell	24
T	5	01
U	7	34
V	;	17
W	2	31
X	/	27
Y	6	25
Z	"	21
Carriage Ret.		02
Line Feed		10
Space		04
LTRS		37
FIGS		33
Blank		00

* Appears printed on the tape, but not on the printed page

TELETYPE CODE (CODE TO TEXT CONVERSION)

OCTAL CODE	LTRS	FIGS	FUNCTION
00			Blank
01	T	5	
02			Carriage Return
03	0	9	
04			Space
05	H	# *	
06	N	,	
07	M	.	
10			Line Feed
11	L)	
12	R	4	
13	G	&	
14	I	8	
15	P	0	
16	C	:	
17	V	;	
20	E	3	
21	Z	"	
22	D	\$	
23	B	?	
24	S	Bell	
25	Y	6	
26	F	!	
27	X	/	
30	A	-	
31	W	2	
32	J	'	
33			FIGS
34	U	7	
35	Q	1	
36	K	(
37			LTRS

*Appears printed on the tape, but not on the printed page

1004 EXCESS-THREE CODE (6 BITS)

	0	1	2	3	4	5	6	7
0	Δ (Space)]	— (Minus)	∅	1	2	3	4
1	5	6	7	8	9	∖	;	[
2	&	:	.	?	A	B	C	D
3	E	F	G	H	I	#	<	=
4	t	*	\$!	J	K	L	M
5	N	O	P	Q	R	(∅	Δ
6	≠	%	' (Comma)	+	/	S	T	U
7	V	W	X	Y	Z	□	>)

Note:

CS-1 Characters Represented Differently by the High-Speed Printer	CS-1 Character	High-Speed Printer Character
Point Separator	.	*
Vertical Bar		\$

FIELD DATA CODE CHART

	0	1	2	3	4	5	6	7
0	Master Space	Upper Case	Lower Case	Line Feed	Car- riage Ret.	Space	A	B
1	C	D	E	F	G	H	I	J
2	K	L	M	N	O	P	Q	R
3	S	T	U	V	W	X	Y	Z
4)	-	+	<	=	>	_	\$
5	*	("	:	?	!	,	Ⓢ Stop
6	0	1	2	3	4	5	6	7
7	8	9	"	;	/	.	□ spec	↑ idle

POWERS OF TWO TABLE

2^n	n	2^{-n}																				
1	0	1.0																				
2	1	0.5																				
4	2	0.25																				
8	3	0.125																				
16	4	0.062	5																			
32	5	0.031	25																			
64	6	0.015	625																			
128	7	0.007	812	5																		
256	8	0.003	906	25																		
512	9	0.001	953	125																		
1 024	10	0.000	976	562	5																	
2 048	11	0.000	488	281	25																	
4 096	12	0.000	244	140	625																	
8 192	13	0.000	122	070	312	5																
16 384	14	0.000	061	035	156	25																
32 768	15	0.000	030	517	578	125																
65 536	16	0.000	015	258	789	062	5															
131 072	17	0.000	007	629	394	531	25															
262 144	18	0.000	003	814	697	265	625															
524 288	19	0.000	001	907	348	632	812	5														
1 048 576	20	0.000	000	953	674	316	406	25														
2 097 152	21	0.000	000	476	837	158	203	125														
4 194 304	22	0.000	000	238	418	579	101	562	5													
8 388 608	23	0.000	000	119	209	289	550	781	25													
16 777 216	24	0.000	000	059	604	644	775	390	625													
33 554 432	25	0.000	000	029	802	322	387	695	312	5												
67 108 864	26	0.000	000	014	901	161	193	847	656	25												
134 217 728	27	0.000	000	007	450	580	596	923	828	125												
286 435 456	28	0.000	000	003	725	290	298	461	914	062	5											
536 870 912	29	0.000	000	001	862	645	149	230	957	031	25											
1 073 741 824	30	0.000	000	000	931	322	574	615	478	515	625											
2 147 483 648	31	0.000	000	000	465	661	287	307	739	257	812	5										
4 294 967 296	32	0.000	000	000	232	830	643	653	869	628	906	25										
8 589 934 592	33	0.000	000	000	116	415	321	826	934	814	453	125										
17 179 869 184	34	0.000	000	000	058	207	660	913	467	407	226	562	5									
34 359 738 368	35	0.000	000	000	029	103	830	456	733	703	613	281	25									
68 719 476 736	36	0.000	000	000	014	551	915	228	366	851	806	640	625									
137 438 953 472	37	0.000	000	000	007	275	957	614	183	425	903	320	312	5								
274 877 906 944	38	0.000	000	000	003	637	978	807	091	712	951	660	156	25								
549 755 813 888	39	0.000	000	000	001	818	989	403	545	856	475	830	078	125								

DECIMAL TO OCTAL CONVERSION TABLE

Decimal: 0 through 399

Octal: 0 through 617

UNITS

		0	1	2	3	4	5	6	7	8	9	
00	0	0	1	2	3	4	5	6	7	10	11	
	1	12	13	14	15	16	17	20	21	22	23	
	2	24	25	26	27	30	31	32	33	34	35	
	3	36	37	40	41	42	43	44	45	46	47	
	<i>TENS</i>	4	50	51	52	53	54	55	56	57	60	61
		5	62	63	64	65	66	67	70	71	72	73
		6	74	75	76	77	100	101	102	103	104	105
		7	106	107	110	111	112	113	114	115	116	117
		8	120	121	122	123	124	125	126	127	130	131
		9	132	133	134	135	136	137	140	141	142	143
01	0	144	145	146	147	150	151	152	153	154	155	
	1	156	157	160	161	162	163	164	165	166	167	
	2	170	171	172	173	174	175	176	177	200	201	
	3	202	203	204	205	206	207	210	211	212	213	
	<i>TENS</i>	4	214	215	216	217	220	221	222	223	224	225
		5	226	227	230	231	232	233	234	235	236	237
		6	240	241	242	243	244	245	246	247	250	251
		7	252	253	254	255	256	257	260	261	262	263
		8	264	265	266	267	270	271	272	273	274	275
		9	276	277	300	301	302	303	304	305	306	307
02	0	310	311	312	313	314	315	316	317	320	321	
	1	322	323	324	325	326	327	330	331	332	333	
	2	334	335	336	337	340	341	342	343	344	345	
	3	346	347	350	351	352	353	354	355	356	357	
	<i>TENS</i>	4	360	361	362	363	364	365	366	367	370	371
		5	372	373	374	375	376	377	400	401	402	403
		6	404	405	406	407	410	411	412	413	414	415
		7	416	417	420	421	422	423	424	425	426	427
		8	430	431	432	433	434	435	436	437	440	441
		9	442	443	444	445	446	447	450	451	452	453
03	0	454	455	456	457	460	461	462	463	464	465	
	1	466	467	470	471	472	473	474	475	476	477	
	2	500	501	502	503	504	505	506	507	510	511	
	3	512	513	514	515	516	517	520	521	522	523	
	<i>TENS</i>	4	524	525	526	527	530	531	532	533	534	535
		5	536	537	540	541	542	543	544	545	546	547
		6	550	551	552	553	554	555	556	557	560	561
		7	562	563	564	565	566	567	570	571	572	573
		8	574	575	576	577	600	601	602	603	604	605
		9	606	607	610	611	612	613	614	615	616	617

DECIMAL-TO-OCTAL CONVERSION TABLE

Decimal: 400 through 799

Octal: 620 through 1437

UNITS

		0	1	2	3	4	5	6	7	8	9	
04	0	620	621	622	623	624	625	626	627	630	631	
	1	632	633	634	635	636	637	640	641	642	643	
	2	644	645	646	647	650	651	652	653	654	655	
	3	656	657	660	661	662	663	664	665	666	667	
	TENS	4	670	671	672	673	674	675	676	677	700	701
		5	702	703	704	705	706	707	710	711	712	713
		6	714	715	716	717	720	721	722	723	724	725
		7	726	727	730	731	732	733	734	735	736	737
		8	740	741	742	743	744	745	746	747	750	751
		9	752	753	754	755	756	757	760	761	762	763
05	0	764	765	766	767	770	771	772	773	774	775	
	1	776	777	1000	1001	1002	1003	1004	1005	1006	1007	
	2	1010	1011	1012	1013	1014	1015	1016	1017	1020	1021	
	3	1022	1023	1024	1025	1026	1027	1030	1031	1032	1033	
	TENS	4	1034	1035	1036	1037	1040	1041	1042	1043	1044	1045
		5	1046	1047	1050	1051	1052	1053	1054	1055	1056	1057
		6	1060	1061	1062	1063	1064	1065	1066	1067	1070	1071
		7	1072	1073	1074	1075	1076	1077	1100	1101	1102	1103
		8	1104	1105	1106	1107	1110	1111	1112	1113	1114	1115
		9	1116	1117	1120	1121	1122	1123	1124	1125	1126	1127
06	0	1130	1131	1132	1133	1134	1135	1136	1137	1140	1141	
	1	1142	1143	1144	1145	1146	1147	1150	1151	1152	1153	
	2	1154	1155	1156	1157	1160	1161	1162	1163	1164	1165	
	3	1166	1167	1170	1171	1172	1173	1174	1175	1176	1177	
	TENS	4	1200	1201	1202	1203	1204	1205	1206	1207	1210	1211
		5	1212	1213	1214	1215	1216	1217	1220	1221	1222	1223
		6	1224	1225	1226	1227	1230	1231	1232	1233	1234	1235
		7	1236	1237	1240	1241	1242	1243	1244	1245	1246	1247
		8	1250	1251	1252	1253	1254	1255	1256	1257	1260	1261
		9	1262	1263	1264	1265	1266	1267	1270	1271	1272	1273
07	0	1274	1275	1276	1277	1300	1301	1302	1303	1304	1305	
	1	1306	1307	1310	1311	1312	1313	1314	1315	1316	1317	
	2	1320	1321	1322	1323	1324	1325	1326	1327	1330	1331	
	3	1332	1333	1334	1335	1336	1337	1340	1341	1342	1343	
	TENS	4	1344	1345	1346	1347	1350	1351	1352	1353	1354	1355
		5	1356	1357	1360	1361	1362	1363	1364	1365	1366	1367
		6	1370	1371	1372	1373	1374	1375	1376	1377	1400	1401
		7	1402	1403	1404	1405	1406	1407	1410	1411	1412	1413
		8	1414	1415	1416	1417	1420	1421	1422	1423	1424	1425
		9	1426	1427	1430	1431	1432	1433	1434	1435	1436	1437

DECIMAL-TO-OCTAL CONVERSION TABLE

Decimal: 800 through 1199

Octal: 1440 through 2257

		<i>UNITS</i>										
		0	1	2	3	4	5	6	7	8	9	
08	0	1440	1441	1442	1443	1444	1445	1446	1447	1450	1451	
	1	1452	1453	1454	1455	1456	1457	1460	1461	1462	1463	
	2	1464	1465	1466	1467	1470	1471	1472	1473	1474	1475	
	3	1476	1477	1500	1501	1502	1503	1504	1505	1506	1507	
	<i>TENS</i>	4	1510	1511	1512	1513	1514	1515	1516	1517	1520	1521
		5	1522	1523	1524	1525	1526	1527	1530	1531	1532	1533
		6	1534	1535	1536	1537	1540	1541	1542	1543	1544	1545
		7	1546	1547	1550	1551	1552	1553	1554	1555	1556	1557
		8	1560	1561	1562	1563	1564	1565	1566	1567	1570	1571
		9	1572	1573	1574	1575	1576	1577	1600	1601	1602	1603
09		0	1604	1605	1606	1607	1610	1611	1612	1613	1614	1615
	1	1616	1617	1620	1621	1622	1623	1624	1625	1626	1627	
	2	1630	1631	1632	1633	1634	1635	1636	1637	1640	1641	
	3	1642	1643	1644	1645	1646	1647	1650	1651	1652	1653	
	<i>TENS</i>	4	1654	1655	1656	1657	1660	1661	1662	1663	1664	1665
		5	1666	1667	1670	1671	1672	1673	1674	1675	1676	1677
		6	1700	1701	1702	1703	1704	1705	1706	1707	1710	1711
		7	1712	1713	1714	1715	1716	1717	1720	1721	1722	1723
		8	1724	1725	1726	1727	1730	1731	1732	1733	1734	1735
		9	1736	1737	1740	1741	1742	1743	1744	1745	1746	1747
10		0	1750	1751	1752	1753	1754	1755	1756	1757	1760	1761
	1	1762	1763	1764	1765	1766	1767	1770	1771	1772	1773	
	2	1774	1775	1776	1777	2000	2001	2002	2003	2004	2005	
	3	2006	2007	2010	2011	2012	2013	2014	2015	2016	2017	
	<i>TENS</i>	4	2020	2021	2022	2023	2024	2025	2026	2027	2030	2031
		5	2032	2033	2034	2035	2036	2037	2040	2041	2042	2043
		6	2044	2045	2046	2047	2050	2051	2052	2053	2054	2055
		7	2056	2057	2060	2061	2062	2063	2064	2065	2066	2067
		8	2070	2071	2072	2073	2074	2075	2076	2077	2100	2101
		9	2102	2103	2104	2105	2106	2107	2110	2111	2112	2113
11		0	2114	2115	2116	2117	2120	2121	2122	2123	2124	2125
	1	2126	2127	2130	2131	2132	2133	2134	2135	2136	2137	
	2	2140	2141	2142	2143	2144	2145	2146	2147	2150	2151	
	3	2152	2153	2154	2155	2156	2157	2160	2161	2162	2163	
	<i>TENS</i>	4	2164	2165	2166	2167	2170	2171	2172	2173	2174	2175
		5	2176	2177	2200	2201	2202	2203	2204	2205	2206	2207
		6	2210	2211	2212	2213	2214	2215	2216	2217	2220	2221
		7	2222	2223	2224	2225	2226	2227	2230	2231	2232	2233
		8	2234	2235	2236	2237	2240	2241	2242	2243	2244	2245
		9	2246	2247	2250	2251	2252	2253	2254	2255	2256	2257

DECIMAL-TO-OCTAL CONVERSION TABLE

Decimal: 1200 through 1599

Octal: 2260 through 3077

UNITS

		0	1	2	3	4	5	6	7	8	9	
12	0	2260	2261	2262	2263	2264	2265	2266	2267	2270	2271	
	1	2272	2273	2274	2275	2276	2277	2300	2301	2302	2303	
	2	2304	2305	2306	2307	2310	2311	2312	2313	2314	2315	
	3	2316	2317	2320	2321	2322	2323	2324	2325	2326	2327	
	TENS	4	2330	2331	2332	2333	2334	2335	2336	2337	2340	2341
		5	2342	2343	2344	2345	2346	2347	2350	2351	2352	2353
		6	2354	2355	2356	2357	2360	2361	2362	2363	2364	2365
		7	2366	2367	2370	2371	2372	2373	2374	2375	2376	2377
		8	2400	2401	2402	2403	2404	2405	2406	2407	2410	2411
		9	2412	2413	2414	2415	2416	2417	2420	2421	2422	2423
13		0	2424	2425	2426	2427	2430	2431	2432	2433	2434	2435
	1	2436	2437	2440	2441	2442	2443	2444	2445	2446	2447	
	2	2450	2451	2452	2453	2454	2455	2456	2457	2460	2461	
	3	2462	2463	2464	2465	2466	2467	2470	2471	2472	2473	
	TENS	4	2474	2475	2476	2477	2500	2501	2502	2503	2504	2505
		5	2506	2507	2510	2511	2512	2513	2514	2515	2516	2517
		6	2520	2521	2522	2523	2524	2525	2526	2527	2530	2531
		7	2532	2533	2534	2535	2536	2537	2540	2541	2542	2543
		8	2544	2545	2546	2547	2550	2551	2552	2553	2554	2555
		9	2556	2557	2560	2561	2562	2563	2564	2565	2566	2567
14		0	2570	2571	2572	2573	2574	2575	2576	2577	2600	2601
	1	2602	2603	2604	2605	2606	2607	2610	2611	2612	2613	
	2	2614	2615	2616	2617	2620	2621	2622	2623	2624	2625	
	3	2626	2627	2630	2631	2632	2633	2634	2635	2636	2637	
	TENS	4	2640	2641	2642	2643	2644	2645	2646	2647	2650	2651
		5	2652	2653	2654	2655	2656	2657	2660	2661	2662	2663
		6	2664	2665	2666	2667	2670	2671	2672	2673	2674	2675
		7	2676	2677	2700	2701	2702	2703	2704	2705	2706	2707
		8	2710	2711	2712	2713	2714	2715	2716	2717	2720	2721
		9	2722	2723	2724	2725	2726	2727	2730	2731	2732	2733
15		0	2734	2735	2736	2737	2740	2741	2742	2743	2744	2745
	1	2746	2747	2750	2751	2752	2753	2754	2755	2756	2757	
	2	2760	2761	2762	2763	2764	2765	2766	2767	2770	2771	
	3	2772	2773	2774	2775	2776	2777	3000	3001	3002	3003	
	TENS	4	3004	3005	3006	3007	3010	3011	3012	3013	3014	3015
		5	3016	3017	3020	3021	3022	3023	3024	3025	3026	3027
		6	3030	3031	3032	3033	3034	3035	3036	3037	3040	3041
		7	3042	3043	3044	3045	3046	3047	3050	3051	3052	3053
		8	3054	3055	3056	3057	3060	3061	3062	3063	3064	3065
		9	3066	3067	3070	3071	3072	3073	3074	3075	3076	3077

DECIMAL-TO-OCTAL CONVERSION TABLE

Decimal: 1600 through 1999

Octal: 3100 through 3717

UNITS

		0	1	2	3	4	5	6	7	8	9	
16	0	3100	3101	3102	3103	3104	3105	3106	3107	3110	3111	
	1	3112	3113	3114	3115	3116	3117	3120	3121	3122	3123	
	2	3124	3125	3126	3127	3130	3131	3132	3133	3134	3135	
	3	3136	3137	3140	3141	3142	3143	3144	3145	3146	3147	
	<i>TENS</i>	4	3150	3151	3152	3153	3154	3155	3156	3157	3160	3161
		5	3162	3163	3164	3165	3166	3167	3170	3171	3172	3173
		6	3174	3175	3176	3177	3200	3201	3202	3203	3204	3205
		7	3206	3207	3210	3211	3212	3213	3214	3215	3216	3217
		8	3220	3221	3222	3223	3224	3225	3226	3227	3230	3231
		9	3232	3233	3234	3235	3236	3237	3240	3241	3242	3243
17		0	3244	3245	3246	3247	3250	3251	3252	3253	3254	3255
	1	3256	3257	3260	3261	3262	3263	3264	3265	3266	3267	
	2	3270	3271	3272	3273	3274	3275	3276	3277	3300	3301	
	3	3302	3303	3304	3305	3306	3307	3310	3311	3312	3313	
	<i>TENS</i>	4	3314	3315	3316	3317	3320	3321	3322	3323	3324	3325
		5	3326	3327	3330	3331	3332	3333	3334	3335	3336	3337
		6	3340	3341	3342	3343	3344	3345	3346	3347	3350	3351
		7	3352	3353	3354	3355	3356	3357	3360	3361	3362	3363
		8	3364	3365	3366	3367	3370	3371	3372	3373	3374	3375
		9	3376	3377	3400	3401	3402	3403	3404	3405	3406	3407
18		0	3410	3411	3412	3413	3414	3415	3416	3417	3420	3421
	1	3422	3423	3424	3425	3426	3427	3430	3431	3432	3433	
	2	3434	3435	3436	3437	3440	3441	3442	3443	3444	3445	
	3	3446	3447	3450	3451	3452	3453	3454	3455	3456	3457	
	<i>TENS</i>	4	3460	3461	3462	3463	3464	3465	3466	3467	3470	3471
		5	3472	3473	3474	3475	3476	3477	3500	3501	3502	3503
		6	3504	3505	3506	3507	3510	3511	3512	3513	3514	3515
		7	3516	3517	3520	3521	3522	3523	3524	3525	3526	3527
		8	3530	3531	3532	3533	3534	3535	3536	3537	3540	3541
		9	3542	3543	3544	3545	3546	3547	3550	3551	3552	3553
19		0	3554	3555	3556	3557	3560	3561	3562	3563	3564	3565
	1	3566	3567	3570	3571	3572	3573	3574	3575	3576	3577	
	2	3600	3601	3602	3603	3604	3605	3606	3607	3610	3611	
	3	3612	3613	3614	3615	3616	3617	3620	3621	3622	3623	
	<i>TENS</i>	4	3624	3625	3626	3627	3630	3631	3632	3633	3634	3635
		5	3636	3637	3640	3641	3642	3643	3644	3645	3646	3647
		6	3650	3651	3652	3653	3654	3655	3656	3657	3660	3661
		7	3662	3663	3664	3665	3666	3667	3670	3671	3672	3673
		8	3674	3675	3676	3677	3700	3701	3702	3703	3704	3705
		9	3706	3707	3710	3711	3712	3713	3714	3715	3716	3717

DECIMAL-TO-OCTAL CONVERSION TABLE

Decimal: 2000 through 2399

Octal: 3720 through 4537

UNITS

		0	1	2	3	4	5	6	7	8	9	
20	0	3720	3721	3722	3723	3724	3725	3726	3727	3730	3731	
	1	3732	3733	3734	3735	3736	3737	3740	3741	3742	3743	
	2	3744	3745	3746	3747	3750	3751	3752	3753	3754	3755	
	3	3756	3757	3760	3761	3762	3763	3764	3765	3766	3767	
	TENS	4	3770	3771	3772	3773	3774	3775	3776	3777	4000	4001
		5	4002	4003	4004	4005	4006	4007	4010	4011	4012	4013
		6	4014	4015	4016	4017	4020	4021	4022	4023	4024	4025
		7	4026	4027	4030	4031	4032	4033	4034	4035	4036	4037
		8	4040	4041	4042	4043	4044	4045	4046	4047	4050	4051
		9	4052	4053	4054	4055	4056	4057	4060	4061	4062	4063
21		0	4064	4065	4066	4067	4070	4071	4072	4073	4074	4075
		1	4076	4077	4100	4101	4102	4103	4104	4105	4106	4107
		2	4110	4111	4112	4113	4114	4115	4116	4117	4120	4121
		3	4122	4123	4124	4125	4126	4127	4130	4131	4132	4133
	TENS	4	4134	4135	4136	4137	4140	4141	4142	4143	4144	4145
		5	4146	4147	4150	4151	4152	4153	4154	4155	4156	4157
		6	4160	4161	4162	4163	4164	4165	4166	4167	4170	4171
		7	4172	4173	4174	4175	4176	4177	4200	4201	4202	4203
		8	4204	4205	4206	4207	4210	4211	4212	4213	4214	4215
		9	4216	4217	4220	4221	4222	4223	4224	4225	4226	4227
22		0	4230	4231	4232	4233	4234	4235	4236	4237	4240	4241
		1	4242	4243	4244	4245	4246	4247	4250	4251	4252	4253
		2	4254	4255	4256	4257	4260	4261	4262	4263	4264	4265
		3	4266	4267	4270	4271	4272	4273	4274	4275	4276	4277
	TENS	4	4300	4301	4302	4303	4304	4305	4306	4307	4310	4311
		5	4312	4313	4314	4315	4316	4317	4320	4321	4322	4323
		6	4324	4325	4326	4327	4330	4331	4332	4333	4334	4335
		7	4336	4337	4340	4341	4342	4343	4344	4345	4346	4347
		8	4350	4351	4352	4353	4354	4355	4356	4357	4360	4361
		9	4362	4363	4364	4365	4366	4367	4370	4371	4372	4373
23		0	4374	4375	4376	4377	4400	4401	4402	4403	4404	4405
		1	4406	4407	4410	4411	4412	4413	4414	4415	4416	4417
		2	4420	4421	4422	4423	4424	4425	4426	4427	4430	4431
		3	4432	4433	4434	4435	4436	4437	4440	4441	4442	4443
	TENS	4	4444	4445	4446	4447	4450	4451	4452	4453	4454	4455
		5	4456	4457	4460	4461	4462	4463	4464	4465	4466	4467
		6	4470	4471	4472	4473	4474	4475	4476	4477	4500	4501
		7	4502	4503	4504	4505	4506	4507	4510	4511	4512	4513
		8	4514	4515	4516	4517	4520	4521	4522	4523	4524	4525
		9	4526	4527	4530	4531	4532	4533	4534	4535	4536	4537

DECIMAL-TO-OCTAL CONVERSION TABLE

Decimal: 2400 through 2799

Octal: 4540 through 5357

UNITS

		0	1	2	3	4	5	6	7	8	9	
24	0	4540	4541	4542	4543	4544	4545	4546	4547	4550	4551	
	1	4552	4553	4554	4555	4556	4557	4560	4561	4562	4563	
	2	4564	4565	4566	4567	4570	4571	4572	4573	4574	4575	
	3	4576	4577	4600	4601	4602	4603	4604	4605	4606	4607	
	TENS	4	4610	4611	4612	4613	4614	4615	4616	4617	4620	4621
		5	4622	4623	4624	4625	4626	4627	4630	4631	4632	4633
		6	4634	4635	4636	4637	4640	4641	4642	4643	4644	4645
		7	4646	4647	4650	4651	4652	4653	4654	4655	4656	4657
		8	4660	4661	4662	4663	4664	4665	4666	4667	4670	4671
		9	4672	4673	4674	4675	4676	4677	4700	4701	4702	4703
25		0	4704	4705	4706	4707	4710	4711	4712	4713	4714	4715
	1	4716	4717	4720	4721	4722	4723	4724	4725	4726	4727	
	2	4730	4731	4732	4733	4734	4735	4736	4737	4740	4741	
	3	4742	4743	4744	4745	4746	4747	4750	4751	4752	4753	
	TENS	4	4754	4755	4756	4757	4760	4761	4762	4763	4764	4765
		5	4766	4767	4770	4771	4772	4773	4774	4775	4776	4777
		6	5000	5001	5002	5003	5004	5005	5006	5007	5010	5011
		7	5012	5013	5014	5015	5016	5017	5020	5021	5022	5023
		8	5024	5025	5026	5027	5030	5031	5132	5033	5034	5035
		9	5036	5037	5040	5041	5042	5043	5044	5045	5046	5047
26		0	5050	5051	5052	5053	5054	5055	5056	5057	5060	5061
	1	5062	5063	5064	5065	5066	5067	5070	5071	5072	5073	
	2	5074	5075	5076	5077	5100	5101	5102	5103	5104	5105	
	3	5106	5107	5110	5111	5112	5113	5114	5115	5116	5117	
	TENS	4	5120	5121	5122	5123	5124	5125	5126	5127	5130	5131
		5	5132	5133	5134	5135	5136	5137	5140	5141	5142	5143
		6	5144	5145	5146	5147	5150	5151	5152	5153	5154	5155
		7	5156	5157	5160	5161	5162	5163	5164	5165	5166	5167
		8	5170	5171	5172	5173	5174	5175	5176	5177	5200	5201
		9	5202	5203	5204	5205	5206	5207	5210	5211	5212	5213
27		0	5214	5215	5216	5217	5220	5221	5222	5223	5224	5225
	1	5226	5227	5230	5231	5232	5233	5234	5235	5236	5237	
	2	5240	5241	5242	5243	5244	5245	5246	5247	5250	5251	
	3	5252	5253	5254	5255	5256	5257	5260	5261	5262	5263	
	TENS	4	5264	5265	5266	5267	5270	5271	5272	5273	5274	5275
		5	5276	5277	5300	5301	5302	5303	5304	5305	5306	5307
		6	5310	5311	5312	5313	5314	5315	5316	5317	5320	5321
		7	5322	5323	5324	5325	5326	5327	5330	5331	5332	5333
		8	5334	5335	5336	5337	5340	5341	5342	5343	5344	5345
		9	5346	5347	5350	5351	5352	5353	5354	5355	5356	5357

DECIMAL-TO-OCTAL CONVERSION TABLE

Decimal: 2800 through 3199

Octal: 5360 through 6177

UNITS

		0	1	2	3	4	5	6	7	8	9	
28	0	5360	5361	5362	5363	5364	5365	5366	5367	5370	5371	
	1	5372	5373	5374	5375	5376	5377	5400	5401	5402	5403	
	2	5404	5405	5406	5407	5410	5411	5412	5413	5414	5415	
	3	5416	5417	5420	5421	5422	5423	5424	5425	5426	5427	
	TENS	4	5430	5431	5432	5433	5434	5435	5436	5437	5440	5441
		5	5442	5443	5444	5445	5446	5447	5450	5451	5452	5453
		6	5454	5455	5456	5457	5460	5461	5462	5463	5464	5465
		7	5466	5467	5470	5471	5472	5473	5474	5475	5476	5477
		8	5500	5501	5502	5503	5504	5505	5506	5507	5510	5511
		9	5512	5513	5514	5515	5516	5517	5520	5521	5522	5523
29		0	5524	5525	5526	5527	5530	5531	5532	5533	5534	5535
	1	5536	5537	5540	5541	5542	5543	5544	5545	5546	5547	
	2	5550	5551	5552	5553	5554	5555	5556	5557	5560	5561	
	3	5562	5563	5564	5565	5566	5567	5570	5571	5572	5573	
	TENS	4	5574	5575	5576	5577	5600	5601	5602	5603	5604	5605
		5	5606	5607	5610	5611	5612	5613	5614	5615	5616	5617
		6	5620	5621	5622	5623	5624	5625	5626	5627	5630	5631
		7	5632	5633	5634	5635	5636	5637	5640	5641	5642	5643
		8	5644	5645	5646	5647	5650	5651	5652	5653	5654	5655
		9	5656	5657	5660	5661	5662	5663	5664	5665	5666	5667
30		0	5670	5671	5672	5673	5674	5675	5676	5677	5700	5701
	1	5702	5703	5704	5705	5706	5707	5710	5711	5712	5713	
	2	5714	5715	5716	5717	5720	5721	5722	5723	5724	5725	
	3	5726	5727	5730	5731	5732	5733	5734	5735	5736	5737	
	TENS	4	5740	5741	5742	5743	5744	5745	5746	5747	5750	5751
		5	5752	5753	5754	5755	5756	5757	5760	5761	5762	5763
		6	5764	5765	5766	5767	5770	5771	5772	5773	5774	5775
		7	5776	5777	6000	6001	6002	6003	6004	6005	6006	6007
		8	6010	6011	6012	6013	6014	6015	6016	6017	6020	6021
		9	6022	6023	6024	6025	6026	6027	6030	6031	6032	6033
31		0	6034	6035	6036	6037	6040	6041	6042	6043	6044	6045
	1	6046	6047	6050	6051	6052	6053	6054	6055	6056	6057	
	2	6060	6061	6062	6063	6064	6065	6066	6067	6070	6071	
	3	6072	6073	6074	6075	6076	6077	6100	6101	6102	6103	
	TENS	4	6104	6105	6106	6107	6110	6111	6112	6113	6114	6115
		5	6116	6117	6120	6121	6122	6123	6124	6125	6126	6127
		6	6130	6131	6132	6133	6134	6135	6136	6137	6140	6141
		7	6142	6143	6144	6145	6146	6147	6150	6151	6152	6153
		8	6154	6155	6156	6157	6160	6161	6162	6163	6164	6165
		9	6166	6167	6170	6171	6172	6173	6174	6175	6176	6177

DECIMAL-TO-OCTAL CONVERSION TABLE

Decimal: 3200 through 3599

Octal: 6200 through 7017

UNITS

		0	1	2	3	4	5	6	7	8	9	
32	0	6200	6201	6202	6203	6204	6205	6206	6207	6210	6211	
	1	6212	6213	6214	6215	6216	6217	6220	6221	6222	6223	
	2	6224	6225	6226	6227	6230	6231	6232	6233	6234	6235	
	3	6236	6237	6240	6241	6242	6243	6244	6245	6246	6247	
	TENS	4	6250	6251	6252	6253	6254	6255	6256	6257	6260	6261
		5	6262	6263	6264	6265	6266	6267	6270	6271	6272	6273
		6	6274	6275	6276	6277	6300	6301	6302	6303	6304	6305
		7	6306	6307	6310	6311	6312	6313	6314	6315	6316	6317
		8	6320	6321	6322	6323	6324	6325	6326	6327	6330	6331
		9	6332	6333	6334	6335	6336	6337	6340	6341	6342	6343
33		0	6344	6345	6346	6347	6350	6351	6352	6353	6354	6355
	1	6356	6357	6360	6361	6362	6363	6364	6365	6366	6367	
	2	6370	6371	6372	6373	6374	6375	6376	6377	6400	6401	
	3	6402	6403	6404	6405	6406	6407	6410	6411	6412	6413	
	TENS	4	6414	6415	6416	6417	6420	6421	6422	6423	6424	6425
		5	6426	6427	6430	6431	6432	6433	6434	6435	6436	6437
		6	6440	6441	6442	6443	6444	6445	6446	6447	6450	6451
		7	6452	6453	6454	6455	6456	6457	6460	6461	6462	6463
		8	6464	6465	6466	6467	6470	6471	6472	6473	6474	6475
		9	6476	6477	6500	6501	6502	6503	6504	6505	6506	6507
34		0	6510	6511	6512	6513	6514	6515	6516	6517	6520	6521
	1	6522	6523	6524	6525	6526	6527	6530	6531	6532	6533	
	2	6534	6535	6536	6537	6540	6541	6542	6543	6544	6545	
	3	6546	6547	6550	6551	6552	6553	6554	6555	6556	6557	
	TENS	4	6560	6561	6562	6563	6564	6565	6566	6567	6570	6571
		5	6572	6573	6574	6575	6576	6577	6600	6601	6602	6603
		6	6604	6605	6606	6607	6610	6611	6612	6613	6614	6615
		7	6616	6617	6620	6621	6622	6623	6624	6625	6626	6627
		8	6630	6631	6632	6633	6634	6635	6636	6637	6640	6641
		9	6642	6643	6644	6645	6646	6647	6650	6651	6652	6653
35		0	6654	6655	6656	6657	6660	6661	6662	6663	6664	6665
	1	6666	6667	6670	6671	6672	6673	6674	6675	6676	6677	
	2	6700	6701	6702	6703	6704	6705	6706	6707	6710	6711	
	3	6712	6713	6714	6715	6716	6717	6720	6721	6722	6723	
	TENS	4	6724	6725	6726	6727	6730	6731	6732	6733	6734	6735
		5	6736	6737	6740	6741	6742	6743	6744	6745	6746	6747
		6	6750	6751	6752	6753	6754	6755	6756	6757	6760	6761
		7	6762	6763	6764	6765	6766	6767	6770	6771	6772	6773
		8	6774	6775	6776	6777	7000	7001	7002	7003	7004	7005
		9	7006	7007	7010	7011	7012	7013	7014	7015	7016	7017

DECIMAL-TO-OCTAL CONVERSION TABLE

Decimal: 3600 through 3999
 Octal: 7020 through 7637

UNITS

		0	1	2	3	4	5	6	7	8	9	
36	0	7020	7021	7022	7023	7024	7025	7026	7027	7030	7031	
	1	7032	7033	7034	7035	7036	7037	7040	7041	7042	7043	
	2	7044	7045	7046	7047	7050	7051	7052	7053	7054	7055	
	3	7056	7057	7060	7061	7062	7063	7064	7065	7066	7067	
	TENS	4	7070	7071	7072	7073	7074	7075	7076	7077	7100	7101
		5	7102	7103	7104	7105	7106	7107	7110	7111	7112	7113
		6	7114	7115	7116	7117	7120	7121	7122	7123	7124	7125
		7	7126	7127	7130	7131	7132	7133	7134	7135	7136	7137
		8	7140	7141	7142	7143	7144	7145	7146	7147	7150	7151
		9	7152	7153	7154	7155	7156	7157	7160	7161	7162	7163
37	0	7164	7165	7166	7167	7170	7171	7172	7173	7174	7175	
	1	7176	7177	7200	7201	7202	7203	7204	7205	7206	7207	
	2	7210	7211	7212	7213	7214	7215	7216	7217	7220	7221	
	3	7222	7223	7224	7225	7226	7227	7230	7231	7232	7233	
	TENS	4	7234	7235	7236	7237	7240	7241	7242	7243	7244	7245
		5	7246	7247	7250	7251	7252	7253	7254	7255	7256	7257
		6	7260	7261	7262	7263	7264	7265	7266	7267	7270	7271
		7	7272	7273	7274	7275	7276	7277	7300	7301	7302	7303
		8	7304	7305	7306	7307	7310	7311	7312	7313	7314	7315
		9	7316	7317	7320	7321	7322	7323	7324	7325	7326	7327
38	0	7330	7331	7332	7333	7334	7335	7336	7337	7340	7341	
	1	7342	7343	7344	7345	7346	7347	7350	7351	7352	7353	
	2	7354	7355	7356	7357	7360	7361	7362	7363	7364	7365	
	3	7366	7367	7370	7371	7372	7373	7374	7375	7376	7377	
	TENS	4	7400	7401	7402	7403	7404	7405	7406	7407	7410	7411
		5	7412	7413	7414	7415	7416	7417	7420	7421	7422	7423
		6	7424	7425	7426	7427	7430	7431	7432	7433	7434	7435
		7	7436	7437	7440	7441	7442	7443	7444	7445	7446	7447
		8	7450	7451	7452	7453	7454	7455	7456	7457	7460	7461
		9	7462	7463	7464	7465	7466	7467	7470	7471	7472	7473
39	0	7474	7475	7476	7477	7500	7501	7502	7503	7504	7505	
	1	7506	7507	7510	7511	7512	7513	7514	7515	7516	7517	
	2	7520	7521	7522	7523	7524	7525	7526	7527	7530	7531	
	3	7532	7533	7534	7535	7536	7537	7540	7541	7542	7543	
	TENS	4	7544	7545	7546	7547	7550	7551	7552	7553	7554	7555
		5	7556	7557	7560	7561	7562	7563	7564	7565	7566	7567
		6	7570	7571	7572	7573	7574	7575	7576	7577	7600	7601
		7	7602	7603	7604	7605	7606	7607	7610	7611	7612	7613
		8	7614	7615	7616	7617	7620	7621	7622	7623	7624	7625
		9	7626	7627	7630	7631	7632	7633	7634	7635	7636	7637

DECIMAL-TO-OCTAL CONVERSION TABLE

Decimal: 4000 through 4096
 Octal: 7640 through 10000

UNITS

		0	1	2	3	4	5	6	7	8	9	
40	0	7640	7641	7642	7643	7644	7645	7646	7647	7650	7651	
	1	7652	7653	7654	7655	7656	7657	7660	7661	7662	7663	
	2	7664	7665	7666	7667	7670	7671	7672	7673	7674	7675	
	3	7676	7677	7700	7701	7702	7703	7704	7705	7706	7707	
	TENS	4	7710	7711	7712	7713	7714	7715	7716	7717	7720	7721
		5	7722	7723	7724	7725	7726	7727	7730	7731	7732	7733
		6	7734	7735	7736	7737	7740	7741	7742	7743	7744	7745
		7	7746	7747	7750	7751	7752	7753	7754	7755	7756	7757
		8	7760	7761	7762	7763	7764	7765	7766	7767	7770	7771
		9	7772	7773	7774	7775	7776	7777	10000			

OCTAL TO TO DECIMAL FRACTION CONVERSION TABLE

Octal: 0.000400 through 0.000777

OCTAL	0	1	2	3	4	5	6	7
.00040	.000976	.000980	.000984	.000988	.000991	.000995	.000999	.001003
.00041	.001007	.001010	.001014	.001018	.001022	.001026	.001029	.001033
.00042	.001037	.001041	.001045	.001049	.001052	.001056	.001060	.001064
.00043	.001068	.001071	.001075	.001079	.001083	.001087	.001091	.001094
.00044	.001098	.001102	.001106	.001110	.001113	.001117	.001121	.001125
.00045	.001129	.001132	.001136	.001140	.001144	.001148	.001152	.001155
.00046	.001159	.001163	.001167	.001171	.001174	.001178	.001182	.001186
.00047	.001190	.001194	.001197	.001201	.001205	.001209	.001213	.001216
.00050	.001220	.001224	.001228	.001232	.001235	.001239	.001243	.001247
.00051	.001251	.001255	.001258	.001262	.001266	.001270	.001274	.001277
.00052	.001281	.001285	.001289	.001293	.001296	.001300	.001304	.001308
.00053	.001312	.001316	.001319	.001323	.001327	.001331	.001335	.001338
.00054	.001342	.001346	.001350	.001354	.001358	.001361	.001365	.001369
.00055	.001373	.001377	.001380	.001384	.001388	.001392	.001396	.001399
.00056	.001403	.001407	.001411	.001415	.001419	.001422	.001426	.001430
.00057	.001434	.001438	.001441	.001445	.001449	.001453	.001457	.001461
.00060	.001464	.001468	.001472	.001476	.001480	.001483	.001487	.001491
.00061	.001495	.001499	.001502	.001506	.001510	.001514	.001518	.001522
.00062	.001525	.001529	.001533	.001537	.001541	.001544	.001548	.001552
.00063	.001556	.001560	.001564	.001567	.001571	.001575	.001579	.001583
.00064	.001586	.001590	.001594	.001598	.001602	.001605	.001609	.001613
.00065	.001617	.001621	.001625	.001628	.001632	.001636	.001640	.001644
.00066	.001647	.001651	.001655	.001659	.001663	.001667	.001670	.001674
.00067	.001678	.001682	.001686	.001689	.001693	.001697	.001701	.001705
.00070	.001708	.001712	.001716	.001720	.001724	.001728	.001731	.001735
.00071	.001739	.001743	.001747	.001750	.001754	.001758	.001762	.001766
.00072	.001770	.001773	.001777	.001781	.001785	.001789	.001792	.001796
.00073	.001800	.001804	.001808	.001811	.001815	.001819	.001823	.001827
.00074	.001831	.001834	.001838	.001842	.001846	.001850	.001853	.001857
.00075	.001861	.001865	.001869	.001873	.001876	.001880	.001884	.001888
.00076	.001892	.001895	.001899	.001903	.001907	.001911	.001914	.001918
.00077	.001922	.001926	.001930	.001934	.001937	.001941	.001945	.001949

OCTAL-TO-DECIMAL FRACTION CONVERSION TABLE

Octal: 0.000000 through 0.000377

OCTAL	0	1	2	3	4	5	6	7
.00000	.000000	.000003	.000007	.000011	.000015	.000019	.000022	.000026
.00001	.000030	.000034	.000038	.000041	.000045	.000049	.000053	.000057
.00002	.000061	.000064	.000068	.000072	.000076	.000080	.000083	.000087
.00003	.000091	.000095	.000099	.000102	.000106	.000110	.000114	.000118
.00004	.000122	.000125	.000129	.000133	.000137	.000141	.000144	.000148
.00005	.000152	.000156	.000160	.000164	.000167	.000171	.000175	.000179
.00006	.000183	.000186	.000190	.000194	.000198	.000202	.000205	.000209
.00007	.000213	.000217	.000221	.000225	.000228	.000232	.000236	.000240
.00010	.000244	.000247	.000251	.000255	.000259	.000263	.000267	.000270
.00011	.000274	.000278	.000282	.000286	.000289	.000293	.000297	.000301
.00012	.000305	.000308	.000312	.000316	.000320	.000324	.000328	.000331
.00013	.000335	.000339	.000343	.000347	.000350	.000354	.000358	.000362
.00014	.000366	.000370	.000373	.000377	.000381	.000385	.000389	.000392
.00015	.000396	.000400	.000404	.000408	.000411	.000415	.000419	.000423
.00016	.000427	.000431	.000434	.000438	.000442	.000446	.000450	.000453
.00017	.000457	.000461	.000465	.000469	.000473	.000476	.000480	.000484
.00020	.000488	.000492	.000495	.000499	.000503	.000507	.000511	.000514
.00021	.000518	.000522	.000526	.000530	.000534	.000537	.000541	.000545
.00022	.000549	.000553	.000556	.000560	.000564	.000568	.000572	.000576
.00023	.000579	.000583	.000587	.000591	.000595	.000598	.000602	.000606
.00024	.000610	.000614	.000617	.000621	.000625	.000629	.000633	.000637
.00025	.000640	.000644	.000648	.000652	.000656	.000659	.000663	.000667
.00026	.000671	.000675	.000679	.000682	.000686	.000690	.000694	.000698
.00027	.000701	.000705	.000709	.000713	.000717	.000720	.000724	.000728
.00030	.000732	.000736	.000740	.000743	.000747	.000751	.000755	.000759
.00031	.000762	.000766	.000770	.000774	.000778	.000782	.000785	.000789
.00032	.000793	.000797	.000801	.000805	.000808	.000812	.000816	.000820
.00033	.000823	.000827	.000831	.000835	.000839	.000843	.000846	.000850
.00034	.000854	.000858	.000862	.000865	.000869	.000873	.000877	.000881
.00035	.000885	.000888	.000892	.000896	.000900	.000904	.000907	.000911
.00036	.000915	.000919	.000923	.000926	.000930	.000934	.000938	.000942
.00037	.000946	.000949	.000953	.000957	.000961	.000965	.000968	.000972

OCTAL-TO-DECIMAL FRACTION CONVERSION TABLE

Octal: 0.400 through 0.777

OCTAL	0	1	2	3	4	5	6	7
.40	.50000	.50195	.50391	.50586	.50781	.50977	.51172	.51367
.41	.51563	.51758	.51953	.52148	.52344	.52539	.52734	.52830
.42	.53125	.53320	.53516	.53711	.53906	.54102	.54297	.54492
.43	.54688	.54883	.55078	.55273	.55469	.55664	.55859	.56055
.44	.56250	.56445	.56641	.56836	.57031	.57227	.57422	.57617
.45	.57813	.58008	.58203	.58398	.58594	.58789	.58984	.59180
.46	.59375	.59570	.59766	.59961	.60156	.60352	.60547	.60742
.47	.60938	.61133	.61328	.61523	.61719	.61914	.62109	.62305
.50	.62500	.62695	.62891	.63086	.63281	.63477	.63672	.63867
.51	.64063	.64258	.64453	.64648	.64844	.65039	.65234	.65430
.52	.65625	.65820	.66016	.66211	.66406	.66602	.66797	.66992
.53	.67188	.67383	.67578	.67773	.67969	.68164	.68359	.68555
.54	.68750	.68945	.69141	.69336	.69531	.69727	.69922	.70117
.55	.70313	.70518	.70703	.70898	.71094	.71289	.71484	.71680
.56	.71875	.72070	.72266	.72461	.72656	.72852	.73047	.73242
.57	.73438	.73633	.73828	.74023	.74219	.74414	.74609	.74805
.60	.75000	.75195	.75391	.75586	.75781	.75977	.76172	.76367
.61	.76563	.76758	.76953	.77148	.77344	.77539	.77734	.77930
.62	.78125	.78320	.78516	.78711	.78906	.79102	.79297	.79492
.63	.79688	.79883	.80078	.80273	.80469	.80664	.80859	.81055
.64	.81250	.81445	.81641	.81836	.82031	.82227	.82422	.82617
.65	.82813	.83008	.83203	.83398	.83594	.83789	.83984	.84180
.66	.84375	.84570	.84766	.84961	.85156	.85352	.85547	.85742
.67	.85938	.86133	.86328	.86523	.86719	.86914	.87109	.87305
.70	.87500	.87695	.87891	.88086	.88281	.88477	.88672	.88867
.71	.89063	.89258	.89453	.89648	.89844	.90039	.90234	.90430
.72	.90625	.90820	.91016	.91211	.91406	.91602	.91797	.91992
.73	.92188	.92383	.92578	.92773	.92969	.93164	.93359	.93555
.74	.93750	.93945	.94141	.94336	.94531	.94727	.94922	.95117
.75	.95313	.95508	.95703	.95898	.96094	.96289	.96484	.96680
.76	.96875	.97070	.97266	.97461	.97656	.97852	.98047	.98242
.77	.98438	.98633	.98828	.99023	.99219	.99414	.99609	.99805

OCTAL-TO-DECIMAL FRACTION CONVERSION TABLE

Octal: 0.000 through 0.377

OCTAL	0	1	2	3	4	5	6	7
.00	.000000	.001953	.003906	.005859	.007812	.009765	.011718	.013671
.01	.015625	.017578	.019531	.021484	.023437	.025390	.027343	.029296
.02	.031250	.033203	.035156	.037109	.039062	.041015	.042968	.044921
.03	.046875	.048828	.050781	.052734	.054687	.056640	.058593	.060546
.04	.062500	.064453	.066406	.068359	.070312	.072265	.074218	.076171
.05	.078125	.080078	.082031	.083984	.085937	.087890	.089843	.091796
.06	.093750	.095703	.097656	.099609	.101562	.103515	.105468	.107421
.07	.109375	.111328	.113281	.115234	.117187	.119140	.121093	.123046
.10	.125000	.126953	.128906	.130859	.132812	.134765	.136718	.138671
.11	.140625	.142578	.144531	.146484	.148437	.150390	.152343	.154296
.12	.156250	.158203	.160156	.162109	.164062	.166015	.167968	.169921
.13	.171875	.173828	.175781	.177734	.179687	.181640	.183593	.185546
.14	.187500	.189453	.191406	.193359	.195312	.197265	.199218	.201171
.15	.203125	.205078	.207031	.208984	.210937	.212890	.214843	.216796
.16	.218750	.220703	.222656	.224609	.226562	.228515	.230468	.232421
.17	.234375	.236328	.238281	.240234	.242187	.244140	.246093	.248046
.20	.250000	.251953	.253906	.255859	.257812	.259765	.261718	.263671
.21	.265625	.267578	.269531	.271484	.273437	.275390	.277343	.279296
.22	.281250	.283203	.285156	.287109	.289062	.291015	.292968	.294921
.23	.296875	.298828	.300781	.302734	.304687	.306640	.308593	.310546
.24	.312500	.314453	.316406	.318359	.320312	.322265	.324218	.326171
.25	.328125	.330078	.332031	.333984	.335937	.337890	.339843	.341796
.26	.343750	.345703	.347656	.349609	.351562	.343515	.355468	.357421
.27	.359375	.361328	.363281	.365234	.367187	.369140	.371093	.373046
.30	.375000	.376953	.378906	.380859	.382812	.384765	.386718	.388671
.31	.390625	.392578	.394531	.396484	.398437	.400390	.402343	.404296
.32	.406250	.408203	.410156	.412109	.414062	.416015	.417968	.419921
.33	.421875	.423828	.425781	.427734	.429687	.431640	.433593	.435546
.34	.437500	.439453	.441406	.443359	.445312	.447265	.449218	.451171
.35	.453125	.455078	.457031	.458984	.460937	.462890	.464843	.466796
.36	.468750	.470703	.472656	.474609	.476562	.478515	.480468	.482421
.37	.484375	.486328	.488281	.490234	.492187	.494140	.496093	.498046