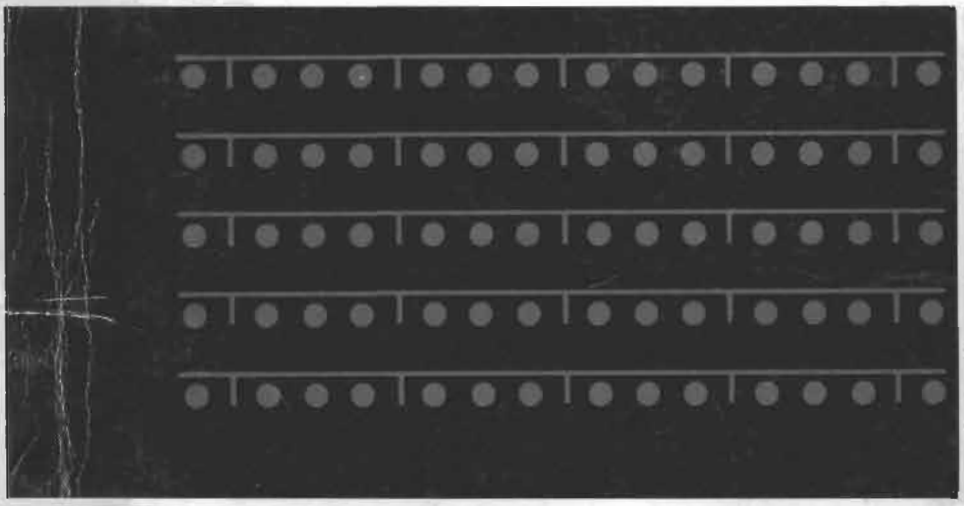
# 2114B COMPUTER

HEWLETT·PACKARD




## SPECIFICATIONS AND BASIC OPERATION

VOLUME 1

VOLUME ONE

SPECIFICATIONS AND BASIC OPERATION MANUAL

# MODEL 2114B COMPUTER


02114-90398

HEWLETT-PACKARD COMPANY
11000 WOLFE ROAD, CUPERTINO, CALIFORNIA, U.S.A.

Printed: MAR 1970

# HP Computer Museum
# www.hpmuseum.net

**For research and education purposes only.**

# TABLE OF CONTENTS

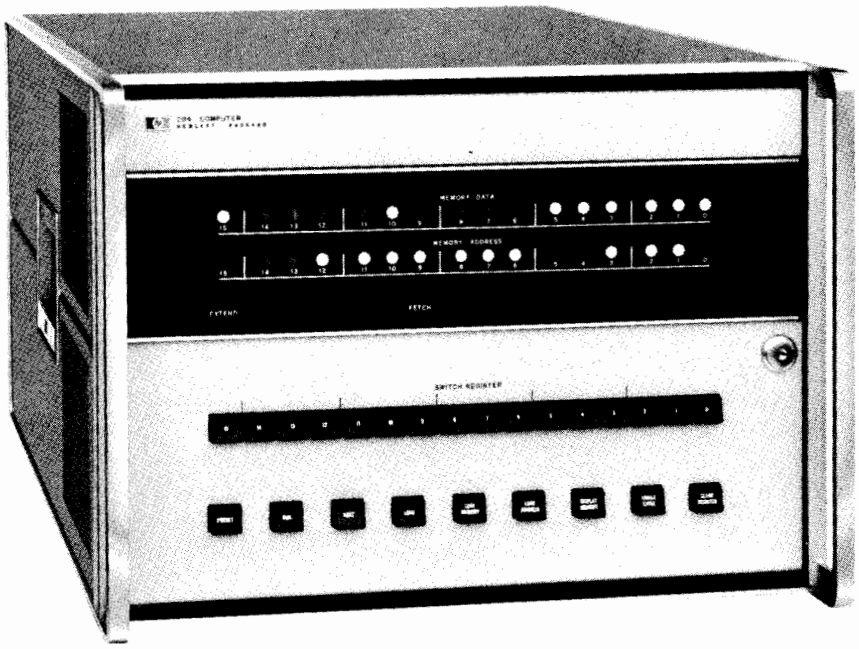


# LIST OF ILLUSTRATIONS

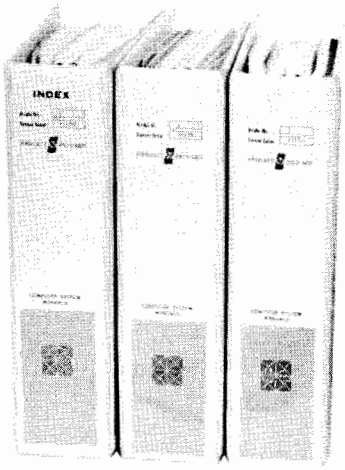
## LIST OF ILLUSTRATIONS (CONTINUED)



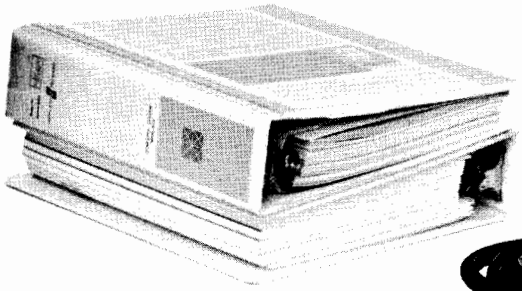
## LIST OF TABLES



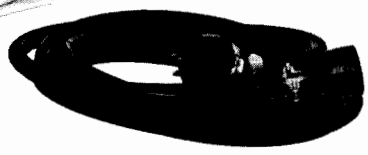
HP 2114B COMPUTER



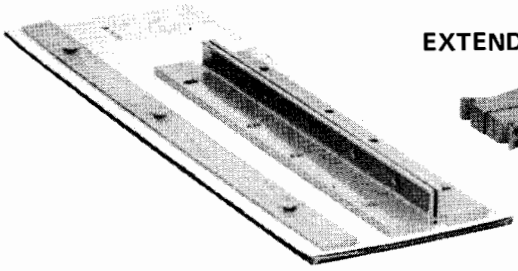

DOCUMENTATION



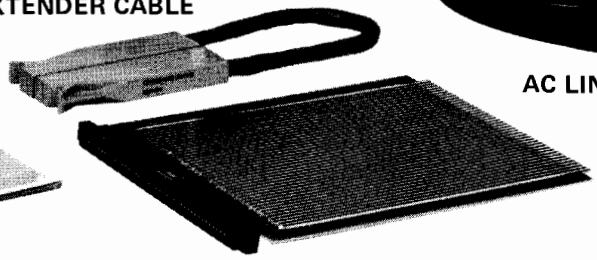
EXTENDER CABLE



AC LINE CORD



RACK MOUNT KIT



EXTENDER CARD

2038-1

Figure 1-1. Basic HP 2114B Computer

# SECTION I

# DOCUMENTATION DESCRIPTION

## 1-1. BASIC COMPUTER MANUALS.

1-2. Documentation supplied with the Hewlett-Packard 2114B Computer consists of four manuals, the contents of which are described briefly in paragraphs 1-3 through 1-12. When the basic HP 2114B Computer (figure 1-1) is purchased as part of a computer system, the system documentation will include a system supplement (paragraph 1-13) containing individual manuals for the peripheral equipment. Figure 1-2 illustrates the organization of the documentation supplied with a typical system.

1-3. SPECIFICATIONS AND BASIC OPERATION MANUAL.

1-4. Volume one is the specifications and basic operation manual, which describes the basic HP 2114B Computer, treated as an independent instrument operable from the front panel. Separate sections of this manual introduce the computer from the following standpoints:

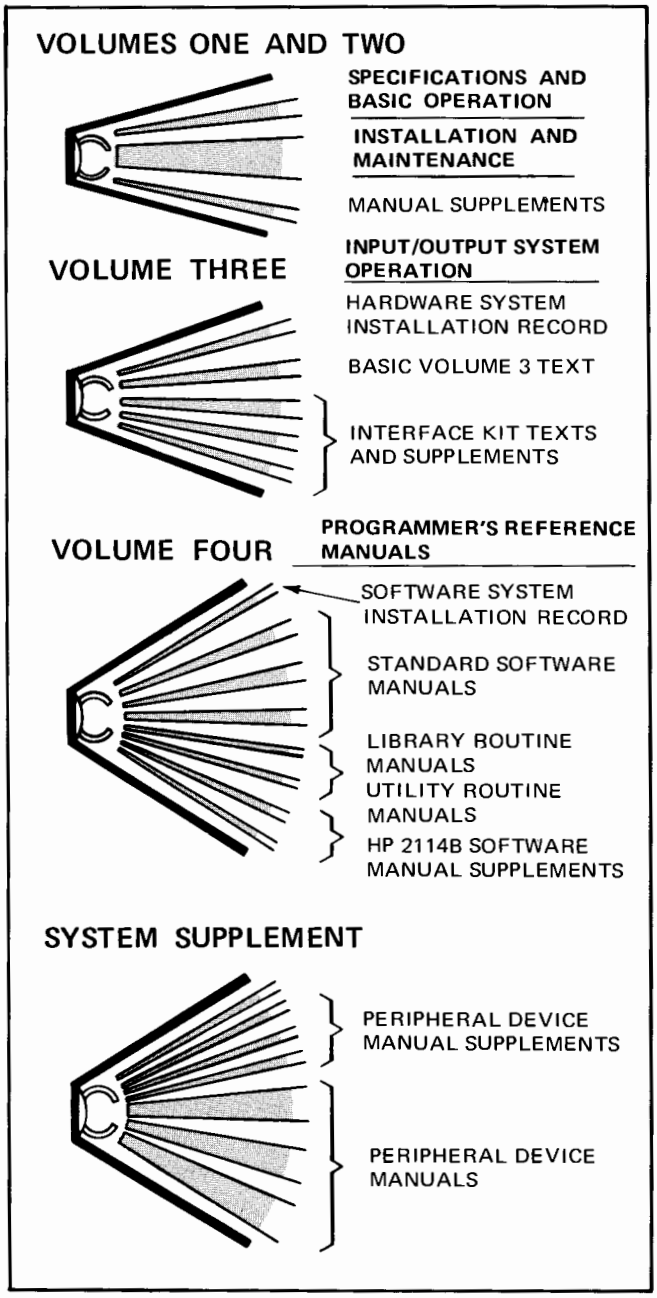
a. Specifications: The full capabilities of the HP 2114B Computer are defined, including standard hardware options and standard software. Information necessary for coding machine-language instructions is listed and described. This section is intended both as a reference for users who are familiar with computer terminology and as a source of detailed definitions, so that the material will be meaningful to readers at a wide range of levels.

b. Fundamentals of Computer Operation: For users with little or no previous experience with computers, this section gives a brief outline of how the computer works internally. This is not a detailed theory of operation, such as is presented in volume two (installation and maintenance) but the logic descriptions in volume two will assume at least this basic level of understanding. Thus a thorough reading of this section is advised before proceeding to the installation and maintenance manual.

c. Basic Operation of HP 2114B Computer: This is a continuation of the preceding section. Procedures for first-time usage are detailed, using the computer front-panel controls and indicators as an elementary input/output device. This section is essentially an introduction to machine-language programming. The assembler and other programming reference manuals included in volume four assume a basic knowledge of machine-language programming, such as presented in this section.

1-5. INSTALLATION AND MAINTENANCE MANUAL.

1-6. Volume two gives instructions for installation and maintenance of the main unit only (see volume three for interconnection and installation of peripheral equipment). Contents of this volume are as follows:




2038-2

Figure 1-2. HP 2114B System Documentation

a. General Information. This section contains a general description of the computer. Included are descriptions of the purpose and contents of the manual and a general description of the computer. Descriptions of the various computer assemblies, panel controls, and maintenance

features, and a list of required test equipment are also included.

b. Installation. This section contains procedures for installation and preparation of the unit for use. Topics covered include inspection, inventory, and performance checkout.

c. Theory of Operation. This section contains a description of the overall operation of the computer and detailed descriptions of the various operational sections of the computer. Reference is made to the logic diagrams in the maintenance section and block diagrams and waveforms in the troubleshooting section.

d. Troubleshooting. This section contains troubleshooting procedures for the computer. Included are pretest instructions, diagnostic interpretation information, logic equations, and timing diagrams. Procedures for running the diagnostic tests are contained in the Manual of Diagnostics. Detailed procedures for troubleshooting specific operational sections of the computer such as the central processory, memory, and power supply are also given in this section.

e. Maintenance. This section contains preventive and corrective maintenance information for the computer. Included in this section are adjustment and test procedures, a signal index, interconnection and wiring information, and schematic and parts location diagrams. Also included are tables of replaceable parts in order of reference designations for each computer assembly.

f. Replaceable Parts. This section contains information for ordering replacement parts for the computer. All replaceable parts are listed in order of the HP part number. The total quantity of each part used, a description of the part, the manufacturer, and the manufacturer's part number are also included in this section.

g. Appendixes. Appendixes containing explanations of the logic symbology used in the manual, operating characteristics for the logic circuitry and backdating information for the manual are included following the last section of the manual.

### 1-7. INPUT/OUTPUT SYSTEM OPERATION MANUAL.

1-8. Volume three describes the input/output structure and provides theory of operation for the I/O control card. Included are sections describing the operation of the interrupt and priority systems as well as the encoding and decoding of interrupt requests and select code addresses.

1-9. Sections for input/output options are inserted as required, according to the interface kits purchased as part of a particular system. The information in these sections condenses operating procedures from the manuals of the individual instruments, and adds material relating specifically to operation with the HP 2114B Computer. Maintenance information in these sections covers only the interface circuits, and not the peripheral itself. Complete operating and service manuals for the peripheral equipment are furnished in the system supplement when included in a particular system. Manual supplements describing production changes affecting volume three are included in the volume three binder.

1-10. A Hardware System Installation Record at the front of the system supplement defines the system configuration as originally shipped, and provides an index to the supporting documents in the system supplement. Space is provided for noting changes and additions.

### 1-11. PROGRAMMER REFERENCE MANUALS.

1-12. Volume four consists of one or more three-ring binders containing documentation for each item of software supplied with the computer. Both standard software programs and software specially originated for an individual user are fully described as to specifications and usage. A Software System Installation Record at the front of volume four lists all software furnished with the original shipment, and provides an index to the supporting documents in volume four. Space is provided for noting changes and additions, so that an up-to-date record can be maintained by the user. Programmer reference manuals normally included in volume four are:

a. HP Assembler
b. HP Symbolic Editor
c. HP Basic Control System
d. HP FORTRAN
e. HP Program Library
f. HP ALGOL
g. HP BASIC
h. HP Standard Software Systems Operating Manual

### 1-13. SYSTEM SUPPLEMENT.

1-14. Supplementary documentation for the hardware system is supplied in the system supplement, which consists of one or more three-ring binders. Individual manuals for the peripheral devices in the system are included here, as well as manual supplements describing any special modifications made to these devices by Hewlett-Packard.

Note

Each 2114B Computer is identified by a serial number on the rear panel (for example 1001A00600 or 949-00599). The first group of digits make up a serial prefix used to document equipment changes. This prefix does not change unless changes to the equipment have been made. The last five digits form a serial number to identify each piece of equipment. The serial prefix may be either three or four digits in length. If the serial prefix contains four digits, a code letter will be stamped between the serial prefix and the serial number indicating the country in which the equipment was manufactured. If the serial prefix on your equipment does not agree with that shown on the title page of the hardware manuals there are differences between your equipment and the equipment described in the manuals. These differences are described in change sheets and manual supplements available at the nearest HP Sales and Service Office.

# SECTION II

# HP 2114B SPECIFICATIONS

## 2-1. DEFINITION OF COMPUTER SYSTEM.

2-2. BASIC UNIT DESCRIPTION. The Hewlett-Packard 2114B Computer is a small general-purpose digital computer which combines performance and economy with small size. The computer has full compatibility with HP data measuring and recording instruments as well as a wide range of input/output devices. The computer is subject to rigid operational and environmental specifications. (Refer to paragraphs 2-6 and 2-7.) The logic design and software follow conventional standards of computer usage and notation so that the computer may also be used as a free-standing device or in other types of systems, such as process control, media conversion, data reduction or communication systems. The hardware and software are specially designed to permit interfacing of real-time devices (i.e., devices running asynchronously with respect to a program being run). The word length is 16 bits. The basic HP 2114B Computer includes the processor unit (mainframe) with a 4096-word memory. All specifications in this section apply to the basic unit only, unless specifically denoted as an option specification.

2-3. OPTIONS. Options for the HP 2114B Computer are of two general types:

a. Processor Options: These options extend the memory and computation capabilities of the basic unit, and are identified by five digit accessory numbers. (Refer to paragraph 2-122.)

b. Input/Output Options: These options add input and/or output facilities to the basic HP 2114B Computer. The option, identified by an interface kit number (paragraph 2-129), provides the circuitry, cabling, and software to enable the computer to operate with a specific input or output instrument (measuring, reading, or recording device) or with a series of instruments. Compatible instruments, not included in the interface kit, are separately available from Hewlett-Packard. When external devices are connected to the computer, the computer then becomes part of a computer system. (Refer to paragraph 2-4.)

2-4. SYSTEMS. Two general types of computer systems are available from Hewlett-Packard.

a. HP 2114B Computer Systems: Systems may be configured to individual requirements using combinations of standard input/output options. Nonstandard input/output options, not mentioned in this section or in the computer data sheet, can be obtained on special order; these options are also designated with interface kit accessory numbers. The software packages which are hardware dependent (basic control system and system input/output) will be made up in accordance with the hardware system configuration.

b. Data Acquisition Systems: Systems are available in standard configurations which combine Hewlett-Packard digital scanning, measuring, and recording equipment with the HP 2114B Computer. In these systems, the computer is programmed to exercise partial or complete control over the data taking process and to perform computations on data measured by the system. A data acquisition program is furnished with these systems. Capabilities of available instruments include measurements of ac or dc voltages, resistances, frequencies, time periods, temperatures, gas pressures, nuclear events, etc., from multiple inputs. (The functions of some instruments such as linearizers, comparators, scanner programmers, and output couplers are present in the basic HP 2114B Computer, or may be accomplished by options or programming.)

## 2-5. PHYSICAL SPECIFICATIONS.

2-6. POWER REQUIREMENTS.

a. Line voltage: 115 volts ac ± 10 percent (7 amperes) or 230 volts ac ± 10 percent (3.5 amperes) with a special transformer.

b. Line frequency: 47.5 to 66 hertz.

c. Power consumption: 800 watts maximum and 500 watts minimum (computer and teleprinter option only, for minimum value).

d. Power cable: uses a standard three-prong connector.

2-7. ENVIRONMENTAL LIMITS.

a. Temperature: 10° to 40°C (50° to 104°F).
b. Relative humidity: to 80 percent at 40°C.

2-8. VENTILATION.

a. Intake at rear and exhaust on sides.
b. Air flow: 400 cubic feet per minute.
c. Heat dissipation: 2200 BTU/hr, maximum.

2-9. PHYSICAL DIMENSIONS.

a. Width: 16-3/4 inches with adapters for standard 19 inch rack mounting (see figure 2-1).

b. Panel height: 12 inches.

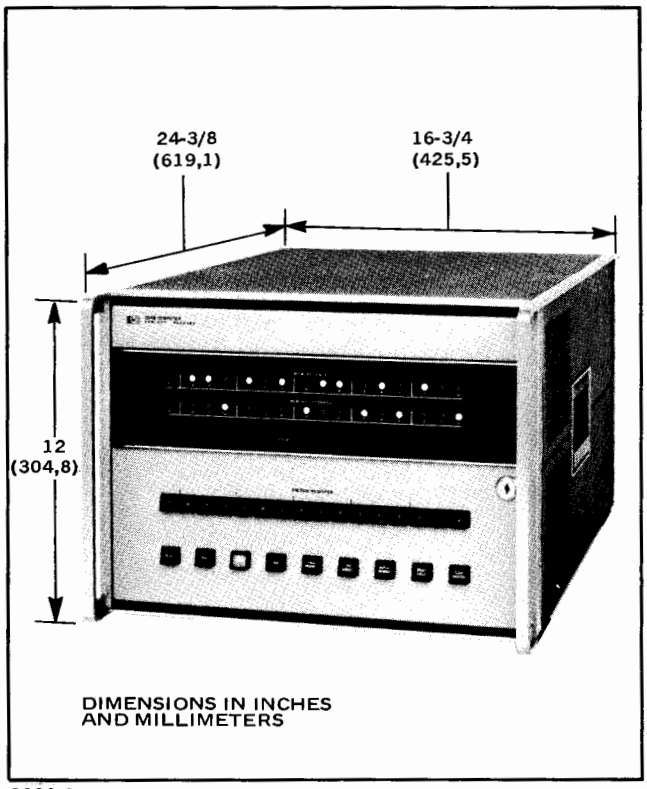
c. Depth: 24-3/8 inches.

d. Recommended cable clearance at rear: 5 inches minimum.

e. Recommended air exhaust clearance at sides: 2 inches minimum.

f. Net weight: 106 lb (48 kg).

g. Shipping weight: 132 lb (59,9 kg).




2038-3

Figure 2-1. HP 2114B Computer Dimensions

2-10. SERVICE ACCESS.

a. The front panel opens, providing access to test switches and protected controls.

b. The top panel slides back and up, permitting top access to input/output connectors, plug-in circuit boards, and wiring.

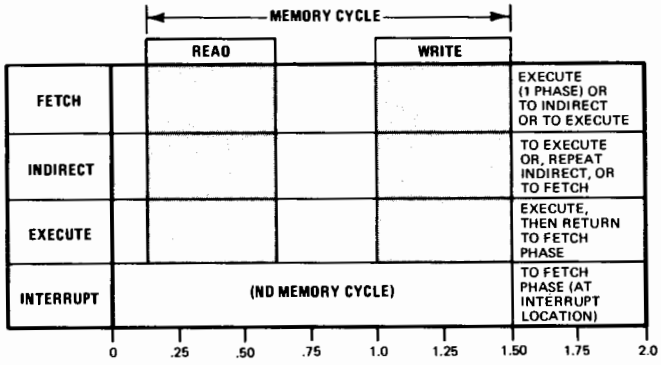
c. The bottom panel is removable for access to backplane wiring.

2-11. INPUT/OUTPUT EXTENDERS. The computer has two options for extending its input/output capability. The HP 2151A Input/Output Extender provides an added 17 I/O slots to the basic computer. It can be quickly and easily installed and has its own self-contained power supply. The multiplexed I/O option provides up to 56 I/O channels. Both units make use of the computer's priority interrupt system.

## 2-12. MACHINE TIMING.

2-13. An internal 8-MHz timing generator automatically generates read/write memory cycles every 2.0 microseconds when running (see figure 2-2). The basic HP 2114B Computer has four machine phases (fetch, indirect, execute, interrupt) of which the first three include a memory cycle. If the direct memory access option is installed, a fifth phase is possible, the suspend phase. (Refer to paragraph 2-19.)

2-14. Phases do not occur in a fixed sequence, but rather are determined by conditions which occur during operation. The computer can go directly from one of the first three phases to certain others in the manner indicated in figure 2-2, and an external device can cause the computer to go into the interrupt phase on completion of any current phase. The fetch phase may be thought of as the normal condition; the processing of each instruction begins with a fetch phase, and in many cases is fully executed within that phase. Each phase takes 2.0 microseconds with one exception: the execute phase of the ISZ (increment, and skip if zero) instruction takes 2.5 microseconds.




2000-44

Figure 2-2. Machine Timing

2-15. FETCH PHASE. The contents of the currently-addressed memory cell are read into the T-register during the read portion of the memory cycle, and written back into the memory cell during the write portion of the memory cycle. The information left in the T-register is taken as an instruction when read during the fetch phase. If the instruction is a memory reference instruction, and includes an indirect address bit (refer to paragraph 2-27), the computer sets the indirect phase condition. If the instruction does not have an indirect address bit but is a memory reference instruction, the computer sets the execute phase condition. Otherwise, the current instruction is fully executed at the end of the fetch phase, and the computer remains in the fetch state for the next memory cycle. An exception to these conditions is the JMP (jump) instruction, which is a memory reference group instruction but does not require an execute phase. The computer executes the instruction at the end of the fetch phase or the indirect phase and then sets the fetch phase again for the next memory cycle.

2-16. INDIRECT PHASE. The contents of the memory cell referenced during the fetch phase are read into the T-register and the entire 16-bit word (15 bits of address, plus a new direct/indirect bit) is taken as a new memory reference for the same instruction. The use of 15 bits for an address permits addressing of maximum memory capacity. If the direct/indirect bit again specifies indirect addressing,

the computer remains in the indirect state and reads another 16-bit address word out of memory as a continuation of multiple-step indirect addressing. If the direct/indirect bit specifies direct addressing, the computer sets the execute phase (or, in the case of a jump indirect, the fetch phase).

2-17. EXECUTE PHASE. The 16-bit data word in the memory cell referenced during a fetch phase or an indirect phase is read into the T-register and is operated on by the current instruction (retained from the fetch phase) at the end of the execute phase. At the end of this phase, the computer sets the fetch phase again to read the next instruction.

2-18. INTERRUPT PHASE. An input/output device requesting service at any time during one of the phases is acknowledged at the end of that phase, unless the interrupt is inhibited for any reason by the program being run. The computer then goes into the interrupt phase, which does not have a memory cycle. During this phase, the P-register is decremented so that no instruction in the main program will be skipped or executed twice. At the end of this phase, the interrupt address of the interrupting device is transferred into the M-register and the fetch phase is set to read the instruction contained in the interrupt address location. The interrupt phase cannot occur again until (at least) this instruction is completed.

2-19. SUSPEND PHASE. When the direct memory access accessory kit is installed, a fifth machine phase is used. When the DMA option is ready to make a data transfer between an I/O device and the computer memory, the normal phases are suspended at the completion of the current machine cycle. The DMA option then uses one machine cycle to perform the data transfer. At the end of the suspend phase the computer resumes operation at the point of the DMA interrupt.

## 2-20. MEMORY.


2-21. TYPE.

2-22. The HP 2114B Computer uses a ferrite core storage module capable of storing 4096 words or 8192 (option 04) words, 17 bits per word (16 bits of the computer word, plus a parity bit which is used by memory parity option 02, when included in the instrument).

2-23. LAYOUT.

2-24. The 4096-word module is logically divided into four pages of 1024 words each. A page is defined as the largest block of memory which can be addressed by the memory address bits of a memory reference instruction (excluding the zero/current page bit; see figure 2-3). In the HP 2114B Computer, memory reference instructions have 10 bits to specify a memory address, and thus the page size is 1024 locations (2000 in octal notation). Octal addresses of the four pages of the basic module, and also the double module (which can be added by option 04) are therefore:

| | |
|---|---|
| Basic Module: | 00000 to 01777 |
| | 02000 to 03777 |
| | 04000 to 05777 |
| | 06000 to 07777 |
| Double Module: | 10000 to 11777 |
| | 12000 to 13777 |
| | 14000 to 15777 |
| | 16000 to 17777 |

2-25. ADDRESSING.

2-26. ZERO/CURRENT PAGE. For direct addressing purposes, generally only two pages are of interest: page zero (the base page, consisting of locations 00000 through 01777), and the current page (the page in which the instruction itself is located). All memory reference instructions include a bit (bit 10) reserved to specify one or the other of these two pages. To address locations in any other page, indirect addressing is used (paragraph 2-27). Page references for direct addressing of memory reference instructions are specified by bit 10 as follows:

Logic 0 = Page Zero (Z)
Logic 1 = Current Page (C)

2-27. DIRECT/INDIRECT. All memory reference instructions use bit 15 to specify direct or indirect addressing. Direct addressing combines the instruction code and the effective address into one word, permitting a memory reference instruction to be executed in two machine phases (fetch and execute). Indirect addressing uses the address part of the instruction word to access another word in memory, which is taken as a new memory reference for the same instruction. This new address word is a full 16 bits long, 15 bits of address plus another direct/indirect bit. The 15-bit length of the address permits access to any location in any module. If bit 15 again specifies indirect addressing, still another address is obtained; this multiple-step indirect addressing may be done to any number of levels. The first address obtained in the indirect phase which does not specify another indirect level becomes the effective address for the instruction. Instructions with indirect addresses are therefore executed in a minimum of three machine phases (fetch, indirect, execute). Direct or indirect addressing is specified by bit 15 as follows:

Logic 0 = Direct
Logic 1 = Indirect

2-28. RESERVED LOCATIONS. The first 64 memory locations of the base page (octal addresses 00000 through 00077) are reserved as listed below. The first two addresses are the A and B flip-flop register addresses and are not core storage locations. Locations 5 through 77 are reserved in the sense that interrupt wiring is present for the priority order given. As long as the locations do not have actual interrupt assignments (as determined by the input/output devices included in the user's system), these locations may be used for normal program purposes.

| | |
|---|---|
| 00000 | Address of A-register. |
| 00001 | Address of B-register. |
| 00002<br>00003 | For exit sequence if A and B contents are used as executable words. |
| 00004 | Interrupt location, highest priority (reserved for power fail interrupt). |
| 00005 | Reserved for memory parity interrupt. |
| 00006 | Reserved for direct memory access. |
| 00007 | Not assigned. |
| 00010 thru 00077 | Interrupt locations in decreasing order of priority. |

2-29. LOADER PROTECTION. The last 64 locations of memory (octal addresses 07700 through 07777 in the standard HP 2114B Computer) are reserved for the basic binary loader. The basic binary loader (not to be confused with the relocating loader program described in paragraph 2-146) is a manually-entered program to permit reading and storage of binary information from punched paper tape, as read by an input device, such as a punched tape reader or a teleprinter. Absolute addresses are required in the loaded data. A protect switch (LOADER ENABLE), when set to NORMAL, protects the basic binary loader locations so that they cannot be altered in any way. For entering the basic binary loader manually into the computer this switch must be set to ON. For actual loading of tapes, both the LOAD and PRESET front panel switches must be pressed simultaneously. The LOADER ENABLE switch is effective for the last 64 locations of memory, regardless of memory size. Plug-in options which expand memory relocate the protected area automatically to the 64 highest numbered locations.

## 2-30. WORKING REGISTERS.

2-31. The HP 2114B Computer has seven working registers and gives continuous display of the contents of the T (MEMORY DATA) and M (MEMORY ADDRESS) registers by lights on the computer front panel. Five of these are 16 bit flip-flop registers, and two are 1-bit flip-flop registers indicated by panel lighting (on or off) of the register name.

2-32. T-REGISTER (MEMORY DATA). All data transferred into or out of memory is routed through the 16-bit T-register (transfer register). The T-register display therefore indicates exactly what information went into or out of a memory cell during the preceeding memory cycle.

2-33. P-REGISTER (PROGRAM COUNTER). On completion of each instruction, the P-register indicates the address of the next instruction to be fetched out of memory. The P-register automatically increments by one (or two, when executing a skip instruction) after the execution of each instruction. A jump instruction (JMP or JSB) can set the P-register to any core location number.

2-34. M-REGISTER (MEMORY ADDRESS). The M-register holds the address of the memory cell being read or written into. The M-register indication will differ from the P-register indication when multi-phase instructions are being processed, since the M-register will be changed by memory references in the instruction (which may be several in the case of indirect addressing) or by an interrupt, whereas the P-register remains constant until completion of the instruction. The M-register will equal the P-register during the fetch phase.

2-35. A-REGISTER (ACCUMULATOR). The A-register is an accumulator, holding the results of arithmetic and logical operations performed by programmed instructions. This register may be addressed by any memory reference instruction as location 00000, thus permitting inter-register operations such as "add B to A", "compare B with A", etc., using a single-word instruction.

2-36. B-REGISTER (ACCUMULATOR). The B-register is a second accumulator, which can hold the results of arithmetic and logical operations completely independent of the A-register. The B-register may be addressed by any memory reference instruction as location 00001 for inter-register operations with A.

2-37. EXTEND. The extend bit is a one-bit (E) register, and is used to link the A and B registers by rotate instructions or to indicate a carry from bit 15 of the A or B registers by an add instruction (ADA, ADB) or increment instruction (INA or INB, but not ISZ) which references these registers. This is of significance primarily for multiple-precision arithmetic. If already set, the extend bit is not complemented by a carry. It may be cleared, complemented, or tested by program instruction. The extend bit is set when the EXTEND panel light is on ("1") and clear when off ("0").

2-38. OVERFLOW. The overflow bit is a one-bit register which indicates, if on, that an add instruction (ADA, ADB) or an increment instruction (INA or INB, but not ISZ) referencing the A- or B-register has caused one of these accumulators to exceed the maximum positive or negative number which can be contained (+32767 or -32768, decimal). This condition is implied by a carry (or lack of carry) from bit 14 to bit 15 (paragraph 3-58). By program instructions, the overflow bit may be cleared, set, or tested. The OVERFLOW panel light remains on until the bit is cleared by an instruction and is not complemented if a second overflow occurs before being cleared. It will not be set by shift or rotate instructions.

## 2-39. PANEL CONTROLS.

2-40. SWITCH REGISTER. The switch register consists of sixteen proximity sense switches used to enter manually-set information into and output data from the computer. The switch register (on is a "1", off is a "0") may be used in the following ways:

a. A program may load the switch register setting into the A- or B-register using LIA or LIB instructions with switch register select code 01.

b. A program may merge the switch register setting (inclusive "or") into the A- or B-register using a MIA or MIB instruction, respectively, and a select code of 01.

c. A program may set the switch register by an output from the A- or B-register using OTA or OTB, respectively, and a select code of 01.

d. The switch register setting may be loaded into the P- and M-registers (simultaneously) by using the LOAD ADDRESS switch, thus directing the computer to a specific memory cell.

e. The switch register setting may be entered into the memory cell specified by the M-register by using the LOAD MEMORY switch, thus permitting the user to change the contents of any memory cell.

2-41. PRESET. Momentary proximity switch to preset the computer to the fetch phase, to turn off the interrupt system and all input/output control bits, to set all input/output flag bits, and to reset the parity halt light located on the computer front panel. It also clears the power fail interrupt circuits. An internal pulse accomplishing the same functions is generated each time power is switched on or off.

2-42. RUN. Momentary proximity switch to start operation at the current state of the computer. Switch is set when a program is running and cleared when the computer halts. When the RUN light is on, all front panel control switches except HALT, and CLEAR REGISTER are disabled.

2-43. HALT. Momentary proximity switch to stop computer operation at the end of the current phase. When the computer is halted, the HALT switch is lit and all front-panel control switches are enabled. (The P-register will not increment if the HALT and LOAD MEMORY or HALT and DISPLAY MEMORY switches are touched simultaneously.)

2-44. LOAD. Proximity switch associated with the last 64 locations in memory; for example, octal addresses 07700 through 07777 in 4K computers, or 17700 through 17777 in 8K computers. These locations are normally occupied by the basic binary loader. The LOAD switch is electrically coupled with the PRESET SWITCH. To load any absolute binary program using the last 64 locations of memory, clear the switch register, hold the PRESET button and simultaneously press LOAD. If switch register bit 0 is a "1", the loader program will read the tape and perform a checksum operation but will not alter memory. If the checksum is incorrect, a HLT 00 will occur; otherwise, the computer will go to a normal HLT 77. If bit 15 of the switch register is a "1", the loader program will perform a compare between the program tape and the stored program in memory but will not alter memory. If the taped program does not compare with the program stored in memory, a HLT 00 is generated; otherwise, the computer will go to a normal HLT 77. If both bit 0 and bit 15 are true, the compare operation will take precedence over the checksum operation.

2-45. LOAD MEMORY. Momentary proximity switch to store the contents of the switch register into the memory location specified by the address in the M-register. The P- and M-registers are automatically incremented after operation of the LOAD MEMORY switch to simplify storing data into consecutive memory locations. (Refer to paragraph 2-43.) The stored data remains displayed in the T-register, and the fetch phase is set at the end of the load operation.

2-46. LOAD ADDRESS. Momentary proximity switch to transfer the contents of the switch register into both the P- and M-registers, thus directing the computer to the desired address. The fetch phase is set at the end of the operation.

2-47. DISPLAY MEMORY. Momentary proximity switch to display, in the T-register, the contents of the location specified by the address in the M-register. The P- and M-registers are automatically incremented after operation of the DISPLAY MEMORY switch so that consecutive memory locations may be displayed simply by repeated operation of this switch. The P- and M-registers are therefore one step ahead of the T-register display. (Refer to paragraph 2-43.) The fetch phase is set after incrementing the P- and M-registers.

2-48. SINGLE CYCLE. Momentary proximity switch to execute one machine cycle each time the switch is pressed. The interrupt phase is not recognized in this mode.

2-49. CLEAR REGISTER. Momentary proximity switch to reset the switch register to "0".

2-50. PROTECTED CONTROLS.

2-51. POWER ON/OFF switch located behind the front panel on the computer chassis. Contents of memory are not affected by switching power off and on; contents of the working registers, however, are lost when power goes off (contents random following turn-on).

2-52. LOADER ENABLE ON/NORMAL switch located on the inside of front panel. The NORMAL position protects the basic binary loader (located in the last 64 positions in memory) making it available for loading tapes. The ON position allows the basic binary loader program to be loaded or changed.

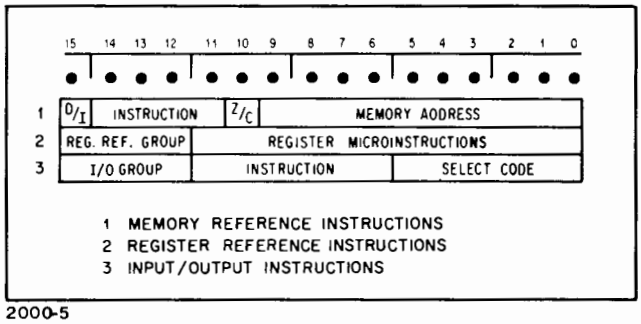
## 2-53. INSTRUCTIONS.

2-54. NUMBER. The HP 2114B Computer has 70 basic one-word instructions, all executable in 2.0 or 4.0 microseconds (except for ISZ, which is executable in 4.5 microseconds). These instructions are grouped in three types, described in paragraphs 2-60 through 2-104. Combinations of the register reference microinstructions, which are all

one-word instructions, executable in 2.0 microseconds, extend the total of different one-word instructions to over 1000.

2-55. FORMATS.

2-56. The three types of basic instructions are grouped according to the bit format of the instruction word. These types are: memory reference, register reference, and input/output instructions. A comparison of the three formats is given in figure 2-3, and detailed binary coding is included with the instruction descriptions following. A consolidated coding table appears in the appendix of this manual.



2000-5

Figure 2-3. Basic Instruction Formats

2-57. The first type comprises the memory reference instructions, using 10 bits (0 through 9) for a memory address, bit 10 to specify zero or current page, and bit 15 for direct or indirect addressing. This leaves four bits (14, 13, 12, 11) to encode the 14 instruction commands in this group.

2-58. The other two types use four bits (15, 14, 13, 12) to distinguish the register reference and the input/output instructions. The register reference type uses bits 11 through 0 to combine up to eight microinstructions (i.e., instructions formed by only 1, 2, or 3 bits), with the resulting multiple instruction operating on the A-, B-, or E-register as a single-word instruction. The input/output type uses bits 11 through 6 for a variety of input/output instructions and bits 5 through 0 to make the instruction apply directly to one of the input/output devices or functions.

2-59. The following paragraphs describe in detail each of the instructions in the three type groups.

Note

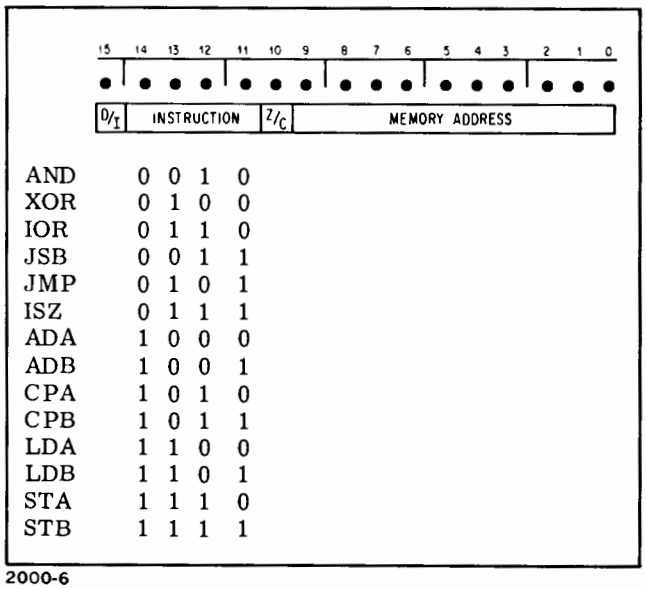
Functions of bits appearing in the form A/B, D/I, D/E, Z/C, or H/C throughout these specifications are obtained by coding "0" or "1" respectively (0/1). For example, A is specified by a zero-bit, and B by a one-bit.

2-60. MEMORY REFERENCE INSTRUCTIONS.

2-61. The 14 memory reference instructions execute some operation involving memory locations, such as transferring information in or out of a memory cell or checking the memory cell contents. The cell referenced (i.e., the absolute address) is determined by a combination of the ten memory address bits in the instruction word (0 through 9) and five bits (10 through 14) assumed from the current condition of the P-register. This means that memory reference instructions can directly address any word in the current page; additionally, if the instruction is given in some location other than the base page (page zero), bit 10 of the instruction word doubles the addressing range to 2048 words by allowing selection of either page zero or current page (i.e., bits 10 through 14 of the address in the M-register can be reset to zero, instead of assuming the current indication of the P-register). This feature provides a convenient linkage between all pages of memory, since page zero can be reached directly from any other page.

2-62. Note that since the A- and B-registers can be addressed (paragraphs 2-35 and 2-36), any memory reference instruction can apply to either of these registers as well as to memory cells. For example, ADA 0001 means add the contents of the B-register (its address being 0001) to the A-register; specify page zero for these operations, since the A and B register addresses are on page zero.

2-63. Figure 2-4 gives instruction codes and mnemonics for all 14 memory reference instructions. All memory reference instructions take a minimum of two machine phases (one to read the instruction word, and one to read the referenced memory cell), except for JMP, which is a one-phase instruction. Logic truth tables relating to the first three instructions described below are given in table 2-1. Note that logic operations are performed on a bit-for-bit basis (i.e., no carries).



2000-6

Figure 2-4. Memory Reference Instructions

Table 2-1. Logic Truth Table

| | AND | XOR | IOR |
|---|---|---|---|
| A-Register Contents | 0 0 1 1 | 0 0 1 1 | 0 0 1 1 |
| Memory | 0 1 0 1 | 0 1 0 1 | 0 1 0 1 |
| Result (in A-Register) | 0 0 0 1 | 0 1 1 0 | 0 1 1 1 |
| 1 = True, 0 = False | | | |

2-64. AND: "And" to A. The contents of the addressed location are logically "anded" to the contents of the A-register. The contents of the memory cell are left unaltered.

2-65. XOR: "Exclusive or" to A. The contents of the addressed location are combined with the contents of the A-register as an "exclusive or" logic operation. The contents of the memory cell are left unaltered.

2-66. IOR: "Inclusive or" to A. The contents of the addressed location are combined with the contents of the A-register as an "inclusive or" logic operation. The contents of the memory cell are left unaltered.

2-67. JSB: Jump to Subroutine. This instruction, executed in location P, causes computer control to jump unconditionally to the memory location (X) specified in the address portion of the JSB instruction word. The contents of the P-register plus one (return address) is stored in location X, and the next instruction to be executed will be that contained in the next location (X + 1). A return to the main program sequence at P + 1 may be effected by a jump indirect through location X.

2-68. JMP: Jump. This instruction transfers control to the addressed location. That is, JMP causes the P- and M-registers to be set according to the memory address portion of the instruction word, thus addressing memory cell X, so that the next instruction will be read from location X.

2-69. ISZ: Increment, and skip if zero. An ISZ instruction adds one to the contents of the addressed memory location. If the result of this operation is zero, the next instruction is skipped; i.e., the P- and M-registers are advanced by two instead of one. Otherwise, the program proceeds normally to the next instruction in sequence. The incremented value is written back into the memory cell in either case. An ISZ instruction referencing locations zero or one (A- or B-register) cannot cause setting of the extend or overflow bits (unlike INA and INB).

2-70. ADA: Add to A. The contents of the addressed memory location are added to the contents of the A-register, and the sum remains in the A-register. The result of the addition may set the extend or overflow bits (paragraphs 2-37 and 2-38). The contents of the memory cell are unaltered.

2-71. ADB: Add to B. The contents of the addressed memory location are added to the contents of the B-register, and the sum remains in the B-register. Extend or overflow bits may be set, as for ADA. The contents of the memory cell are unaltered.

2-72. CPA: Compare to A, skip if unequal. The contents of the addressed location are compared with the contents of the A-register. If the two 16-bit words are different, the next instruction is skipped; i.e., the P- and M-registers are advanced by two instead of one. If the words are identical, the program proceeds normally to the next instruction in sequence. The contents of neither the A-register nor the memory cells are altered.

2-73. CPB: Compare to B, and skip if unequal. Same as CPA, except comparison is made with B-register.

2-74. LDA: Load into A. The A-register is loaded with the contents of the addressed location. The contents of the memory cell are unaltered.

2-75. LDB: Load into B. The B-register is loaded with the contents of the addressed location. The contents of the memory cell are unaltered.

2-76. STA: Store A. The contents of the A-register are stored in the addressed location. The previous contents of the memory cell are lost; the A-register is unaltered.

2-77. STB: Store B. The contents of the B-register are stored in the addressed location. The previous contents of the memory cell are lost; the B-register is unaltered.

2-78. REGISTER REFERENCE INSTRUCTIONS.

2-79. The register reference instructions, in general, manipulate bits in the A-, B-, and E-registers. There is no reference to memory; thus these instructions are executed in only one machine phase. This type includes 39 basic instructions, which are combinable to form a one-word multiple instruction that can operate in various ways on the contents of the A-, B-, or E-registers. These microinstructions are divided into two subgroups, the shift-rotate group (SRG) and the alter-skip group (ASG). Three instructions (SLA, SLB, and CLE) appear in both groups and, being combinable in these different contents, are counted twice in the total of basic instructions. Microinstructions may be combined under the following general rules:

a. Instructions from the two groups cannot be mixed.

b. References to both A and B registers cannot be mixed.

c. Only one microinstruction can be chosen from each column of the selection tables in figure 2-5 and 2-6.

d. Use zeros to exclude unwanted microinstruction bits.

e. The sequence of execution is left to right in the selection tables (column 1, then column 2, etc.).

f. If two (or more) skip functions are combined, the skip will occur if either or both conditions are met. One exception exists for the RSS instruction (paragraph 2-82).

2-80. Register Reference. Instructions are recognized by the computer when the four most significant bits of the instruction word are zeros; the general format for this type of instruction (the dots representing variable microinstruction bits) is therefore:

| 15 | 14 13 12 | 11 10 9 | 8 7 6 | 5 4 3 | 2 1 0 |
|---|---|---|---|---|---|
| 0 | 0 0 0 | · · · | · · · | · · · | · · · |

2-81. SHIFT-ROTATE GROUP. The SRG instructions are specified by a zero for bit 10. (Compare figures 2-5 and 2-6.) Figure 2-5 gives both the bit format and the selection table for using these instructions. Definitions for the mnemonics used are listed below. Note that the extend bit is not affected by shifts or rotates unless specifically stated. All of the shift and rotate instructions can be executed either first or last in a combined instruction, or both times. This permits sequencing of CLE and SLA/B either before or after shifts and rotates.

NOP No operation. Memory read/write cycle only.

CLE Clear E-register.

SLA Skip next instruction if least significant bit of A-register is zero (i.e., skip if an even number is in A-register).

SLB Skip next instruction if least significant bit of B-register is zero (i.e., skip if an even number is in B-register).

ALS A-register left shift one place, arithmetically (15 bits only). A zero replaces vacated bit 0; bit shifted out of bit 14 is lost; bit 15 (sign bit) is not affected.

BLS B-register left shift one place, arithmetically (15 bits only). A zero replaces vacated bit 0; bit shifted out of bit 14 is lost; bit 15 (sign bit) is not affected.

ARS A-register right shift one place, arithmetically. Bit shifted out of bit 0 is lost; copy of sign bit (bit 15) shifted into bit 14; bit 15 is not affected.

BRS B-register right shift one place, arithmetically. Bit shifted out of bit 0 is lost; copy of sign bit (bit 15) shifted into bit 14; bit 15 is not affected.

RAL Rotate A-register left one place, all 16 bits. Bit 15 is rotated around to bit 0.

RBL Rotate B-register left one place, all 16 bits. Bit 15 is rotated around to bit 0.

RAR Rotate A-register right one place, all 16 bits. Bit 0 is rotated around to bit 15.

RBR Rotate B-register right one place, all 16 bits. Bit 0 is rotated around to bit 15.

ALR A-register left shift one place, same as ALS, but clear sign bit after shift.

BLR B-register left shift one place, same as BLS, but clear sign bit after shift.

ERA Rotate E-register right with A-register, one place (17 bits). Bit 0 is rotated into extend-register; extend content is rotated into bit 15.

ERB Rotate E-register right with B-register, one place (17 bits). Bit 0 is rotated into extend-register; extend content is rotated into bit 15.

ELA Rotate E-register left with A-register, one place (17 bits). Bit 15 is rotated into extend-register; extend content is rotated into bit 0.

ELB Rotate E-register left with B-register, one place (17 bits). Bit 15 is rotated into extend-register; extend content is rotated into bit 0.

ALF Rotate A-register left four places, all 16 bits. Bits 15, 14, 13, 12 are rotated around to bits 3, 2, 1, 0 respectively. Equivalent to four successive RAL instructions.

BLF Rotate B-register left four places, all 16 bits. Bits 15, 14, 13, 12 are rotated around to bits 3, 2, 1, 0 respectively. Equivalent to four successive RBL instructions.

2-82. ALTER-SKIP GROUP. The ASG instructions are specified by a "1" in bit 10. Figure 2-6 gives both the bit format and the selection table for using these instructions. Definitions for the mnemonics are used as follows:

CLA Clear A-register.

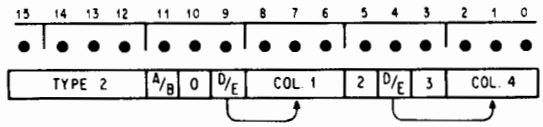
CLB Clear B-register.

CMA Complement A-register. One's complement, reversing the state of all 16 bits.

CMB Complement B-register. Reverses state of all 16 bits.

CCA Clear, then complement A-register. Puts 16 ones in the A-register; this is the two's complement form of -1.

CCB Clear, then complement B-register. Puts 16 ones in the B-register; this is the two's complement form of -1.




| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| NOP | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| CLE | 0 | 0 | 0 | 0 | | 0 | | | | | 1 | | | | | |
| SLA | 0 | 0 | 0 | 0 | 0 | 0 | | | | | | | 1 | | | |
| SLB | 0 | 0 | 0 | 0 | 1 | 0 | | | | | | | 1 | | | |
| ALS | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | | X | | X | X | X |
| BLS | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | | X | | X | X | X |
| ARS | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | | X | | X | X | X |
| BRS | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | | X | | X | X | X |
| RAL | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | | X | | X | X | X |
| RBL | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | | X | | X | X | X |
| RAR | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | | X | | X | X | X |
| RBR | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | | X | | X | X | X |
| ALR | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | | X | | X | X | X |
| BLR | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | | X | | X | X | X |
| ERA | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | | X | | X | X | X |
| ERB | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | | X | | X | X | X |
| ELA | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | | X | | X | X | X |
| ELB | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | | X | | X | X | X |
| ALF | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | | X | | X | X | X |
| BLF | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | | X | | X | X | X |

**SELECTION TABLE**

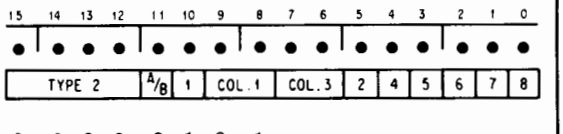
| 1 | 2 | 3 | 4 |
|---|---|---|---|
| ALS<br>ARS<br>RAL<br>RAR<br>ALR<br>ERA<br>ELA<br>ALF | CLE | SLA | ALS<br>ARS<br>RAL<br>RAR<br>ALR<br>ERA<br>ELA<br>ALF |
| BLS<br>BRS<br>RBL<br>RBR<br>BLR<br>ERB<br>ELB<br>BLF | CLE | SLB | BLS<br>BRS<br>RBL<br>RBR<br>BLR<br>ERB<br>ELB<br>BLF |

**COMBINING GUIDE**

1. Choose up to 4 instructions, one from each column of the Selection Table.
2. Use a one-bit for Bit 9 to Enable column 1 instructions, and a one-bit for Bit 4 to Enable column 4 instructions. Figure above shows column 1 enabled (executed first) with duplicate column 4 pattern (executed last) indicated by X's.
3. Use a one-bit for Bit 5 to select column 2 (CLE), or a zero-bit to exclude CLE.
4. Use a one-bit for Bit 3 to select column 3 (SLA/B), or a zero-bit to exclude SLA/B.

2000-7

Figure 2-5. Shift-Rotate Instructions




| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CLA | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | | | | | | | | |
| CLB | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | | | | | | | | |
| CMA | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | | | | | | | | |
| CMB | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | | | | | | | | |
| CCA | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | | | | | | | | |
| CCB | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | | | | | | | | |
| SEZ | 0 | 0 | 0 | 0 | | 1 | | | | | | | | 1 | | |
| CLE | 0 | 0 | 0 | 0 | | 1 | | | 0 | 1 | | | | | | |
| CME | 0 | 0 | 0 | 0 | | 1 | | | 1 | 0 | | | | | | |
| CCE | 0 | 0 | 0 | 0 | | 1 | | | 1 | 1 | | | | | | |
| SSA | 0 | 0 | 0 | 0 | 0 | 1 | | | | | | 1 | | | | |
| SSB | 0 | 0 | 0 | 0 | 1 | 1 | | | | | | 1 | | | | |
| SLA | 0 | 0 | 0 | 0 | 0 | 1 | | | | | | | 1 | | | |
| SLB | 0 | 0 | 0 | 0 | 1 | 1 | | | | | | | 1 | | | |
| INA | 0 | 0 | 0 | 0 | 0 | 1 | | | | | | | | 1 | | |
| INB | 0 | 0 | 0 | 0 | 1 | 1 | | | | | | | | 1 | | |
| SZA | 0 | 0 | 0 | 0 | 0 | 1 | | | | | | | | | 1 | |
| SZB | 0 | 0 | 0 | 0 | 1 | 1 | | | | | | | | | 1 | |
| RSS | 0 | 0 | 0 | 0 | | 1 | | | | | | | | | | 1 |

**SELECTION TABLE**

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|
| CLA<br>CMA<br>CCA | SEZ | CLE<br>CME<br>CCE | SSA | SLA | INA | SZA | RSS |
| CLB<br>CMB<br>CCB | SEZ | CLE<br>CME<br>CCE | SSB | SLB | INB | SZB | RSS |

**COMBINING GUIDE**

1. Choose up to 8 instructions, one from each column of the Selection Table.
2. Use the specified two-bit combinations of Bits 9 and 8, plus A/B Bit 11, to encode column 1 instructions.
3. Use the specified two-bit combinations of Bits 7 and 6 to encode column 3 instructions.
4. Use a one-bit in Bits 5, 4, 3, 2, 1, plus A/B Bit 11, to encode column 2, 4, 5, 6, 7 instructions respectively.
5. Use a one-bit for Bit 0 to encode column 8.

2000-8

Figure 2-6. Alter-Skip Instructions

CLE Clear E-register.

CME Complement E-register. Reverses state of the extend bit.

CCE Clear, then complement E-register. Sets the extend bit.

SEZ Skip the next instruction if E-register is zero.

SSA Skip next instruction if sign bit (bit 15) of A-register is zero; i.e., skip if the content of A is positive.

SSB Skip next instruction if sign bit (bit 15) of B-register is zero; i.e., skip if the content of B is positive.

SLA Skip next instruction if least significant bit of A-register is zero (i.e., skip if an even number is in A).

SLB Skip next instruction if least significant bit of B-register is zero (i.e., skip if an even number is in B).

INA Increment A-register by one. Can cause setting of extend or overflow bits (paragraphs 2-37 and 2-38).

INB Increment B-register by one. Can cause setting of extend of overflow bits (paragraphs 2-37 and 2-38).

SZA Skip next instruction if A-register is zero (16 zeros).

SZB Skip next instruction if B-register is zero (16 zeros).

RSS Reverse skip sense. Skip occurs for any of the preceding skip instructions, if present, when the non-zero condition is met. RSS without a skip instruction in the word causes an unconditional skip. If a word with RSS also includes both SSA/B and SLA/B, bits 15 and 0 must both be one for skip to occur; in all other cases the skip occurs if either or both conditions are met.

## 2-83. INPUT/OUTPUT INSTRUCTIONS.

2-84. The HP 2114B Computer has 17 basic input/output instructions, which provide the following general capabilities.

a. Fix the state of the flag, control, and overflow bits. (These bits are described in paragraphs 2-111 and 2-38.)

b. Test the state of the flag and overflow bits (i.e., skip if set or clear, as specified).

c. Enter data from a specific device into the A- or B-register.

d. Output data to a specific device from the A- or B-register.

e. Halt the program.

2-85. Input/output instructions are recognized by the computer when the four most significant bits of the instruction word are 1000 and bit 10 is a "1". The codes and mnemonics for all 17 instructions are given in figure 2-7 (the MAC instruction is not counted as a basic instruction; see paragraph 2-87). All input/output instructions are executed in one cycle (the fetch phase).

2-86. Note that bit 11, where relevant, specifies A- or B-register; otherwise it may be "1" or "0" without affecting the instruction, although the assembler will assign zeros (as shown). Bit 9, where not specified, offers the choice of holding (0) or clearing (1) the device flag after execution of the instruction. (Exception: the H/C bit associated with the last two instructions in this list holds or clears the overflow bit instead of the flag bit.) Bits 8, 7, and 6 identify the instruction; some of the instructions, however, require additional specific bits for the complete code. Bits 5 through 0 form select codes to make the instruction apply to one of up to 64 input/output devices or functions (see paragraph 2-107).

2-87. The MAC instruction listed in figure 2-7 is available to provide up to 2048 entries to macroinstruction subroutines. Since it is used only by special options and special software, MAC is not counted as one of the 70 basic machine instructions. The basic HP 2114B will treat MAC as a no-operation (NOP) instruction.

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | TYPE 3 | | | | A/B | * | H/C | INSTRUCTION | | | SELECT CODE | | | | | |
| MAC | 1 | 0 | 0 | 0 | | 0 | | | | | | | | | | |
| HLT | 1 | 0 | 0 | 0 | | 1 | | 0 | 0 | 0 | | | | | | |
| STF | 1 | 0 | 0 | 0 | | 1 | 0 | 0 | 0 | 1 | | | | | | |
| CLF | 1 | 0 | 0 | 0 | | 1 | 1 | 0 | 0 | 1 | | | | | | |
| SFC | 1 | 0 | 0 | 0 | | 1 | 0 | 0 | 1 | 0 | | | | | | |
| SFS | 1 | 0 | 0 | 0 | | 1 | 0 | 0 | 1 | 1 | | | | | | |
| MIA | 1 | 0 | 0 | 0 | 0 | 1 | | 1 | 0 | 0 | | | | | | |
| MIB | 1 | 0 | 0 | 0 | 1 | 1 | | 1 | 0 | 0 | | | | | | |
| LIA | 1 | 0 | 0 | 0 | 0 | 1 | | 1 | 0 | 1 | | | | | | |
| LIB | 1 | 0 | 0 | 0 | 1 | 1 | | 1 | 0 | 1 | | | | | | |
| OTA | 1 | 0 | 0 | 0 | 0 | 1 | | 1 | 1 | 0 | | | | | | |
| OTB | 1 | 0 | 0 | 0 | 1 | 1 | | 1 | 1 | 0 | | | | | | |
| STC | 1 | 0 | 0 | 0 | 0 | 1 | | 1 | 1 | 1 | | | | | | |
| CLC | 1 | 0 | 0 | 0 | 1 | 1 | | 1 | 1 | 1 | | | | | | |
| STO | 1 | 0 | 0 | 0 | | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| CLO | 1 | 0 | 0 | 0 | | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| SOC | 1 | 0 | 0 | 0 | | 1 | | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| SOS | 1 | 0 | 0 | 0 | | 1 | | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |

* Identifies Macroinstructions (0) or standard Input/Output instructions (1).

2000-9

Figure 2-7. Input/Output Instructions

2-88. HLT. Halt. Stops the computer and holds or clears the flag (according to bit 9) of any desired input/output device (bits 5 through 0). The HLT instruction has the same effect as the HALT pushbutton: the HALT switch lights, all front-panel control switches are enabled, and no interrupts may occur. The HLT instruction will be displayed in the T-register, and the M-register will normally indicate the HALT location plus one.

2-89. STF. Set flag. Sets the input/output flag of the selected device, thus causing an interrupt during the next machine phase if the interrupt system is enabled (paragraph 2-113), and the corresponding control bit is set. The interrupt system itself is enabled by an STF instruction with a select code of 6 zeros (octal 00).

2-90. CLF. Clear flag of selected device. Resets the flag, thus permitting the device to present another flag when ready again. A CLF with a select code of 6 zeros (octal 00) disables the entire interrupt system; this does not affect the status of individual input/output flags.

2-91. SFC. Skip if flag clear. Causes the computer to skip the next instruction if the flag bit of the selected device is zero (i.e., the device is not ready).

2-92. SFS. Skip if flag set. The next instruction is skipped if the flag bit of the selected device is one (i.e., the device is ready).

2-93. MIA. Merge input into A. The contents of the input/output buffer associated with the selected device are merged ("inclusive or") into the A-register.

2-94. MIB. Merge input into B. The contents of the input/output buffer associated with the selected device are merged ("inclusive or") into the B-register.

2-95. LIA. Load input into A. The contents of the input/output buffer associated with the selected device are loaded into the A-register. Previous contents of the A-register are lost.

2-96. LIB. Load input into B. The contents of the input/output buffer associated with the selected device are loaded into the B-register. Previous contents of the B-register are lost.

2-97. OTA. Output from A. The contents of the A-register are loaded into the input/output buffer associated with the selected device. If the buffer is less than 16 bits in length, the least significant bits of the A-register normally are loaded. (Some exceptions exist, depending on the type of output device.) A-register contents are not altered.

2-98. OTB. Output from B. The contents of the B-register are loaded into the input/output buffer associated with the selected device.

2-99. STC. Set control bit of the selected device. This commands or prepares the device to perform its input or output function, and enables its flag bit to interrupt the program being run (provided the program is not disabling the interrupt system).

2-100. CLC. Clear control bit of the selected device. This prevents the device from interrupting. A CLC instruction with a select code of 00 (octal) clears all control bits, effectively turning off all input/output devices. CLF 00 may be combined with this to additionally turn off the interrupt system.

2-101. STO. Set overflow. The overflow bit remains set until cleared by one of the following three instructions.

2-102. CLO. Clear overflow. Resets the overflow register.

2-103. SOS. Skip if overflow set. If the overflow register is set, the next instruction of the program is skipped. Use of the H/C bit will hold or clear the overflow bit following execution of this instruction (whether the skip is taken or not).

2-104. SOC. Skip if overflow clear. If the overflow register is clear, the next instruction of the program is skipped. Use of the H/C bit will hold or clear the overflow bit following execution of this instruction.

## 2-105. DATA FORMATS.

2-106. Data is represented in two's complement form internally in the computer. The basic format for arithmetic operations on numerical data is defined in figure 2-8. The data is assumed to be an integer (binary point to the right of bit 0), and is positive if the sign bit is "0", or negative if "1". The largest possible positive number (in octal) is +77777, or (in decimal) +32767; the largest possible negative number is -100000 (octal) or -32768 (decimal). Other possible formats, including packed data words, double-length fixed point, and floating point representations, are defined in standard software packages.

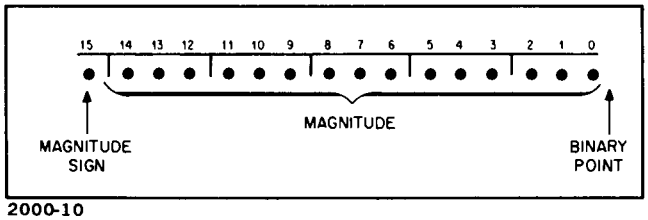



Figure 2-8. Basic Data Format

## 2-107. INPUT/OUTPUT SPECIFICATIONS.

2-108. INPUT/OUTPUT SYSTEM DESIGN.

2-109. GENERAL. Information is transferred into the computer from an external device, or out of the computer

to an external device, by way of its input/output capability, termed the input/output system. A transfer of information is initiated by a signal from a device indicating that it is ready for input or output. The transfer occurs by the process of interrupting a running program (which could be either a problem-solving program, or a program specifically designed to transfer data). The interrupt directs the computer to a location in memory uniquely associated with the interrupting device. This location in turn directs the computer to a program routine (service routine), which must previously have been stored in memory, and this routine will contain instructions which effect the actual transfer of information. Since interrupts can occur at almost any time, including during the service routine of an earlier interrupt, a priority network is present in the computer to establish the sequence in which interrupts are serviced. As shown in figure 2-9, the input/output system capability (including the priority network and the identical hardwiring for optional plug-in card slots) is an integral part of the computer unit. The remaining part is provided by input/output options (paragraph 2-129), which will include the plug-in interface cards and cables for specific devices and the appropriate software drivers and diagnostic programs. The interface cards may be plugged into any of the identical input/output slots, depending on the desired priority rating. Each combination of interface card and device, when plugged into the computer, constitutes an input/output channel.

2-110. NUMBER OF CHANNELS. The coding structure of input/output instructions (figure 2-7) allows 6 bits for a select code, making it possible to specify a total of 64 ($2^6$) channels and functions. Of this total, two select codes are assigned to non-interrupting functions (interrupt system enable/disable, and switch register overflow), two are used for control of the direct memory access option, two are not available for use with the computer, and the remaining 58 channels and functions have an interrupt capability. Two interrupt assignments are reserved for internal processor functions (power failure interrupt, and parity error interrupt). This leaves a possible 56 channels for input/output devices. The computer accommodates 7 of the 56 input/output channels. This may be extended to a total of 24 with the HP 2151A I/O Extender, or if the multiplexed I/O option is used, the entire 56 input/output channels are available.

2-111. INTERFACE COMPONENTS. Each plug-in interface card normally includes the following components, shown in figure 2-10:

a. An input/output buffer consisting of up to 16 flip-flops for temporary storage of data to be transferred in or out, so that it is not necessary to tie up a working register during the relatively long transfer periods. The actual number of buffer bits, from 1 to 16, will depend on the device for which the interface is intended. Data is transferred to the buffer from the A- or B-register by OTA or OTB instructions, and is brought in to the A- or B-register from the buffer by LIA, LIB, MIA, or MIB instructions. If the buffer is less than 16 bits in length, data is transferred to or from the least significant bits of the A- or B-register.

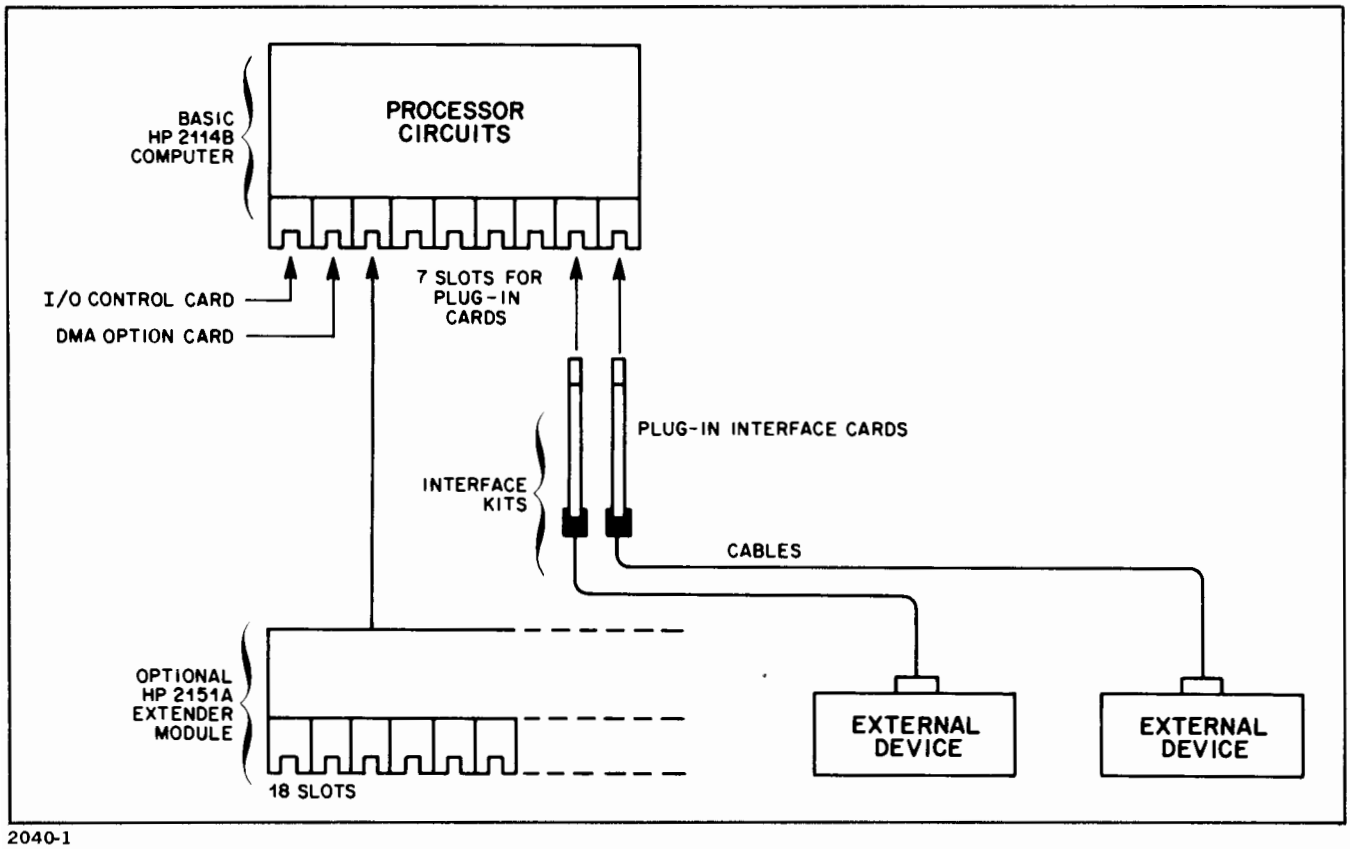
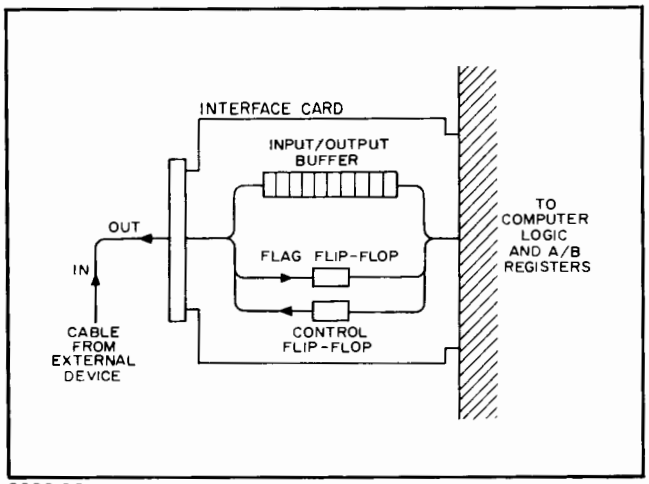



Figure 2-9. Input/Output Design Arrangement




2000-12

Figure 2-10. Components of Typical Input/Output Interface Cards

b. An input/output flag flip-flop, which will be set by a signal from the external device when the device has completed an operation. The flag may also be set, if desired, by program instruction (STF). Once set, the flag remains set until reset by a clear instruction (CLF or H/C bit). Provided it is itself not inhibited by the set flag of a higher priority device or otherwise disabled, the flag, when set, inhibits all interrupts for devices having lower priority. It will cause an interrupt after the current machine phase (paragraph 2-113). Successive interrupts for one device may occur on receipt of a number of flag signals without executing a clear flag instruction, thus making it possible to inhibit lower priority devices indefinitely until a desired number of high-priority transfers have been completed. The flag can be set and cleared even if its interrupt capability is inhibited or disabled, and may be tested by SFS or SFC instructions.

c. A control flip-flop to command or enable the external device to perform its input or output operation. In addition, the control bit controls the interrupt capability for that particular device; i.e., unless the control flip-flop is set, a received flag cannot cause an interrupt, nor can it inhibit the interrupt capability of any other device in the priority string. Thus, the control bit, when set, effectively turns on the individual input/output channel.

2-112. SELECT CODE ASSIGNMENTS. As mentioned previously in paragraph 2-85, bits 5 through 0 of the input/output instructions form a select code to specify one of 64 possible input/output devices or functions. Of the 64 select codes, some are reserved for specific uses while others are available for assignment to any optional input/output device. Table 2-2 lists these assignments and gives the corresponding interrupt location (i.e., the location containing the instruction to be executed when interrupt occurs). The first four (octal codes 00 through 03) are reserved for noninterrupting functions. Note that select code 00 is the access to the master interrupt enable flip-flop; a STF instruction with this select code enables the interrupt system. Select code 01 is assigned to the switch register when using input instructions (LIA, LIB, MIA, MIB, OTA, OTB), permitting the program to enter the switch register setting into the A- or B-register or output data to the switch register from the A- or B-register; when using instructions concerning the overflow register (STO, CLO, SOC, SOS), select code 01 is assigned to the overflow register. Select codes 02 and 06 are reserved for use by the direct memory access option. Select code 04 is the highest priority interrupt, reserved for power failure interrupt. Select code 05 is the next highest priority interrupt and is reserved for the memory protect and parity error options. Select codes 03 and 07 are not program accessible. The next 7 codes (10 through 16), are used for external devices capable of causing an interrupt, with decreasing priority. Select codes 20 through 77 are available only with the use of an input/output extender option.

Table 2-2. Select Code Assignments

| SELECT CODE (OCTAL) | INTERRUPT LOCATION | ASSIGNMENT |
|---|---|---|
| 00 | None | Interrupt System Disable/Enable |
| 01 | None | Switch Register or Overflow |
| 02 | None | DMA Initialize |
| 03 | None | Not Assigned |
| 04 | 00004 | Power Fail Interrupt/Central Interrupt Register |
| 05 | 00005 | Memory Parity Interrupt |
| 06 | 00006 | DMA Completion Interrupt |
| 07 | 00007 | Not Assigned |
| 10 thru 16 | 00010 thru 00016 | I/O Device, highest priority |
| 17 thru 37 | 00017 thru 00037 | Additional I/O Capability Available with HP 2151A I/O Extender |
| 17 thru 77 | 00017 thru 00077 | Additional I/O Capability Available with HP Multiplexer I/O Options |

2-113. INTERRUPT STRUCTURE.



2-114. OPERATION. On computer command (set control instruction STC), one or more external devices begin their read or record operation, putting data into (input) or taking data from (output) the input/output buffer on each individual interface card. During this time, the computer may continue running a program or may be programmed into a waiting loop to wait for a specific device. On completion of the read or record operation, each device returns an operation completed signal (flag) to the computer. The flags are passed through a priority network (paragraph 2-119), which allows only one device to be serviced regardless of the number of flags simultaneously present. The flag with the highest priority causes an interrupt at the end of the current machine phase, switching the computer into the interrupt phase (paragraph 2-18), except under any of the following circumstances.

a. Interrupt system disabled (paragraph 2-112), or device interrupt disabled.

b. Computer in HALT mode. SINGLE CYCLE pushbutton cannot step the computer into the interrupt phase.

c. JMP indirect or JSB indirect not fully executed. These instructions inhibit all interrupts until the instruction (plus one phase of the succeeding instruction) is completed.

d. Instruction in an interrupt location not fully executed, even if of lower priority. Any interrupt inhibits the entire interrupt system until one fetch phase has been completed. (JMP indirect and JSB indirect are exceptions and will be fully executed.)

e. Direct memory access option in process of transferring data. Exception: power failure control can interrupt a DMA transfer.

f. The current instruction is one which may affect the priorities of input/output devices (STC, CLC, STF, CLF). The interrupt in this case must wait until the end of the succeeding machine phase.

2-115. When interrupt occurs, the computer puts the select code number of the interrupting device into the M-register (with extra zeros to specify page zero), thus causing the next instruction to be read from the memory location having the same number as the select code. This location in memory is referred to as the interrupt location, and is reserved for that particular device. Example: a device specified by a select code of 10 will interrupt to (i.e., cause execution of the contents of) memory location 00010. The instruction in the interrupt location will usually be a jump to an input or output subroutine (JSB).

2-116. To prevent external devices from running when computer power is first turned on, turn-on of the POWER switch automatically clears all control bits, resets the interrupt enable flip-flop (disabling the interrupt system), and sets all device flags. Pressing the PRESET pushbutton accomplishes the same function when the computer is on (but not when running, since the control switches are disabled). Therefore, before any device can operate with the computer, it is necessary for the program to set interrupt system enable and (depending on the type of device) clear the individual flag bit and/or set the individual control bit.

2-117. INPUT INTERRUPT. The typical operation sequence for an input interrupt involves the following steps:

a. A STC instruction, usually accompanied by CLF, sends a command (equivalent to a read, or encode, or reset command pulse) to the external device.

b. The device reads its input, then puts the data into the input/output buffer on the interface card (paragraph 2-111).

c. Simultaneously the device supplies a flag signal (equivalent to a record or print command pulse) to the computer.

d. The flag is converted to an interrupt request by the device interface card.

e. The resulting interrupt causes a service subroutine for that device to begin, temporarily suspending operation of the main program.

f. The service subroutine enters data from the buffer into the A- or B-register, processes the data, then returns control to the main program.

2-118. OUTPUT INTERRUPT. The typical operation sequence for an output interrupt involves the following steps:

a. An OTA or OTB instruction puts data from the A- or B-register into the input/output buffer.

b. STC instruction sends a command (equivalent to a record or print command pulse) to the external device.

c. The device accepts (records) the data currently in the buffer.

d. After the data has been accepted, the device returns a flag signal (equivalent to the end of a hold-off or inhibit command pulse) to the computer.

e. The flag is converted to an interrupt request by the device interface card.

f. The interrupt causes a service subroutine for that device to begin.

g. The service subroutine loads new data into the buffer, repeating the sequence.

2-119. PRIORITY. The priority network gives highest interrupt priority to select code 04, reserved for power failure control interrupt, and decreasing priority to select codes in order from 05 through 77. The transfer of data by the optional direct memory access (DMA) channel (which transfers data directly to and from memory by inserting a special memory cycle, rather than by interrupt to a service subroutine) effectively has a priority between select codes 05 and 07 since it can inhibit all interrupts except power failure control, and parity error. When the multiplexed input/output option is used the priority given to any multiplexed device is limited by the hardwired priority of the I/O slot where the multiplexer data card is installed. The priority between the individual multiplexed device is assigned by the user in his interface circuitry.

2-120. A set flag inhibits all interrupt requests below it on the priority string (provided that its control flip-flop is also set), and once this flag is cleared, the next lower device can then interrupt. A service subroutine for any device can be interrupted by a higher priority device; then, after the

higher device is serviced, the subroutine may continue. In this way, it is possible for several service subroutines to be in a state of interruption at one time; each will be permitted to continue when the higher priority device is serviced. All service subroutines normally end with a JMP indirect instruction to return the computer to the point of interrupt.

2-121. TRANSFER RATE. It is possible to make up to 47,000 transfers per second, limited by the length of the service subroutine. If no subroutine is required, a maximum of 250,000 transfers per second may be made.

## 2-122. PROCESSOR OPTIONS.

2-123. The following options are all capable of being installed in the field. They consist of one or more plug-in cards, and in the case of option 04, a larger memory module as well. Other processor options are available, as either standard or custom modifications; consult the HP 2114B Technical Data Sheet or a Hewlett-Packard Sales and Service office.

2-124. 8K MEMORY. Option 04 comprises a set of memory addressing cards and a replacement core module, expanding memory of the HP 2114B from 4096 to 8192 words. Cards and module are installed in the computer mainframe.

2-125. MEMORY PARITY CHECK. Option 02, HP 12598A. Permits parity checking within memory. Odd parity is used. Accessory 12598A consists of one plug-in card for standard 4K memory and optional 8K memory. A parity error may either cause the machine to halt or interrupt to address 05.

2-126. POWER FAIL WITH RESTART. Option 08 permits the computer to store the contents of the working registers in memory in the event of a power failure. When proper power levels are restored the computer will continue processing the program at the point of interrupt.

2-127. DIRECT MEMORY ACCESS. The DMA option HP 12607A allows high-speed data transfers (up to 1/2 million per second) between I/O devices and memory. The option provides control signals to automatically control Hewlett-Packard input/output devices.

2-128. HIGH SPEED I/O CHANNEL. The High Speed Channel Option, HP 12616A, allows the user to access computer memory at arbitrary locations at a rate of 1/2 million transfers per second.

## 2-129. INPUT/OUTPUT OPTIONS.

2-130. Input/output options for the HP 2114B Computer, identified by interface kit accessory numbers, consist of a combination of plug-in cards, interconnecting cables, and appropriate software. In many cases, an optional interface kit is designed to operate with more than one kind of peripheral device, or with different versions of a device. Also, one peripheral may be associated with more than one interface. A device may require one interface to transfer its data into the computer, and another interface to accept function commands from the computer.

2-131. Most input/output options require only one card. This card by itself has no definite select code assignment or interrupt priority. Plugging the card into any of the seven general purpose input/output slots, each of which has a select code assignment, automatically gives the external device an interrupt priority, according to the select code of the slot.

2-132. As shown in figure 2-11, each of the input/output slots actually has two select codes available, although usually only one is used by the interface cards. There can be no gaps in the priority string; continuity is required from select code position 10 up to the last used select code.

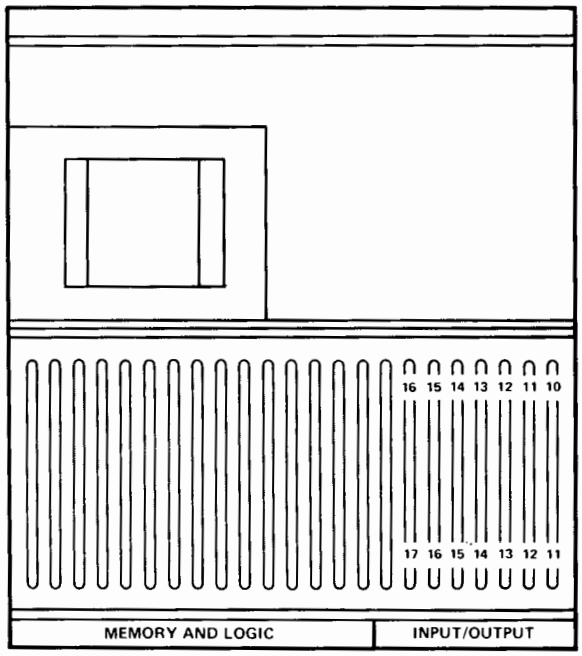



Figure 2-11. Input/Output Option Locations (Top View)

2-133. For more than seven input/output cards, an input/output extender, either the HP 2151A or the multiplexed I/O option is required. The HP 2151A is a separate unit with a self-contained power supply which can make available another 17 I/O channels for a total of 24. The multiplexed I/O option consists of a special I/O interface card called the multiplexer data card, which is inserted into one of the computer I/O slots, leaving six I/O slots in the mainframe of the computer available for interfacing to other devices. The standard I/O control card in the computer is modified for multiplexed I/O operation. Together these two cards make all computer I/O address and control signals available to the user. The computer decodes the I/O

instruction and transmits the select code (two digit octal) and necessary control signals to the user's interface. All channels have access to the computer's priority interrupt system.

2-134. TELEPRINTER INPUT/OUTPUT. The simplest configuration of an HP 2114B Computer system is provided by a combination of the HP 2752A Teleprinter (modified Teletype ASR-33) and accessory interface kit HP 12531B. The teleprinter combines a typewriter, punched tape reader, and tape punch. Data and instructions may be entered from punched tape or the keyboard. Output information is recorded on the typewriter, and may be recorded simultaneously on punched tape. The teleprinter operates at 10 characters/second for both data entry and data recording. Where heavy use of the teleprinter is anticipated, exceeding 5 hours per day or 30 hours per week, a heavy duty HP 2754B Teleprinter (modified ASR-35) is recommended. This device uses the same interface. The HP 2752A and HP 2754B Teleprinters perform the same functions and operate at the same speed.

2-135. HIGH-SPEED PUNCHED TAPE INPUT. For rapid entry of punched tape programs and data into the computer, a high speed tape reader is available. The HP 2748A Tape Reader, with its interface kit, HP 12597A-02, reads punched tape at the rate of 500 characters per second. The HP 2758A Tape Reader-Reroller uses the same interface kit and has an added reroller feature that automatically rerolls the punched tapes as they are processed.

2-136. HIGH-SPEED PUNCHED TAPE OUTPUT. Data output of the HP 2114B Computer can be recorded (asynchronously) on punched tape at 120 characters/second with an HP 2753A Tape Punch and HP 12597A-03 Interface Kit. This device includes a tape spooler, which accepts approximately 1000 feet of tape.

2-137. Input/Output options can be added, upgraded, or deleted, and service priorities changed, on a plug-in basis. No wiring changes to the computer are involved. For a complete list of input/output options available, contact your nearest Hewlett-Packard Sales and Service Office. Input/output software is modular, and a software configurator (paragraph 2-147) is furnished which allows the user to change his software operating system to handle different hardware configurations with minimal programming effort.

## 2-138. SOFTWARE.

2-139. GENERAL.

2-140. The HP 2114B Computer is supported by a full range of software, normally furnished in the form of punched paper tape. As standard accessories, the following software packages are supplied with all HP computers, unless additions or deletions are otherwise specified. All are operable with the minimum HP 2114B Computer system configuration (i.e., 4K memory and teleprinter input/output).

HP Basic Control System
HP Symbolic Editor
HP Assembler
HP FORTRAN Compiler
HP FORTRAN Library
HP System Input/Output
HP Hardware Diagnostics

2-141. Each of the software packages listed above consists in most cases of a number of individual tapes. The number of tapes furnished depends on the options purchased with a system; driver tapes and test tapes are furnished as accessories to interface options when purchased, either with the initial order or with field installation. Table 2-3 lists all the standard tapes furnished with a typical system, consisting of an HP 2752A Teleprinter, HP 2753A Tape Punch, and HP 2748A Tape Reader. In this case, 28 tapes would be furnished for computers having 4K memories. For 8K memory computers, 26 tapes would be furnished, since the FORTRAN compiler requires only two pass tapes instead of four. (Note: the list of standard software given in table 2-3 may change from time to time; check with the Hewlett-Packard Sales and Service Offices for latest information.) In addition to these standard tapes, two configured tapes, incorporating actual system device assignments, are furnished with the initial shipment, one for the system input/output drivers and one for the basic control system. The system input/output (SIO) drivers primarily provide input/output capability for the assembler, symbolic editor, desired in user programs. The basic control system, on the other hand, is primarily intended to provide a complete software input/output system for user programs (paragraph 2-144). These two tapes are unique to each system, and do not have HP accessory numbers and are not listed in the software catalog. Subsequent reconfiguring of system input/output and the basic control system, if desired, is easily accomplished by the user, with the aid of supplied software (system input/output dump, and prepare control system).

2-142. Each software tape is separately identifiable by description and HP accessory number, labeled on both the tape container and the tape itself. The letter at the end of the number identifies a particular version of the tape (e.g., B supersedes A). A detailed list of the software packed with the system is given in the software installation record, supplied with the system documentation at the front of volume four. When ordering new or duplicate tapes (or documentation), the latest applicable version will automatically be furnished. Software is ordered through Hewlett-Packard Sales and Service Offices. A fee is charged for all software, except for the one set of standard software tapes (paper only) defined in the preceding paragraphs, included with the computer (mylar tapes are extra cost). The following paragraphs, to the end of this section, give a brief description of the standard software packages supplied with the computer.

2-143. BASIC CONTROL SYSTEM.

2-144. The HP basic control system provides a complete software facility for input/output operations, so that pro-

Table 2-3. Standard HP Software

| |
|---|
| *Basic Control System |
| Input/Output Control |
| Relocating Loader |
| Debug Routines |
| Prepare Control System |
| **BCS Teleprinter Driver |
| **BCS Tape Reader Driver |
| **BCS Tape Punch Driver |
| Symbolic Editor |
| Assembler, Basic, Non-EAU |
| FORTRAN Complier (8K only): Pass 1 |
| Pass 2 |
| Pass 3 |
| Pass 4 |
| FORTRAN and ALGOL Library (8K only) |
| *System Input/Output |
| System Input/Output Dump |
| **SIO Teleprinter Driver |
| **SIO Tape Reader Driver |
| **SIO Tape Punch Driver |
| Hardware Diagnostics |
| Alter-Skip Instruction Test |
| Memory Reference Instruction Test |
| Shift-Rotate Instruction Test |
| Memory Address Test (Low Core) |
| Memory Address Test (High Core) |
| Memory Checkerboard Test (Low Core) |
| Memory Checkerboard Test (High Core) |
| ** Teleprinter Test |
| **Tape Reader Test |
| **Tape Punch Test |

*A configured tape is furnished with the initial shipment both for the system input/output and for the basic control system, in addition to the individual tapes listed above. These two additional tapes, unique to each system, do not have HP accessory numbers; they are identified only by system serial number.

**Driver tapes and test tapes are furnished for each type of device in a system. The nine tapes listed above are for a typical system.

grams written by the user need not include input/output subroutines within the program. This permits input/output statements in source programs to be general in nature (i.e., not tied to specific devices), and allows easy modification when input/output requirements change. When running relocatable programs, the basic control system will normally be present in the last page of memory, and its subroutines are available by call from any point in memory. To call input/output operations, the user programs a five-word request in assembly language. The request includes the function to be performed (read or write), the unit reference, a reject address (in case the unit is not available), a buffer address (the first location in core in which the data is stored or will be stored), and a buffer length (the number of words or characters that are to be transmitted). The basic control system interprets the request, initiates the data transfer, and returns control to the program. Interrupts which occur during or on termination of the data transfer are processed entirely by the basic control system; the program need not include interrupt handling subroutines.

2-145. The basic control system is modular in design, consisting of several programs which can be combined to suit the user's particular hardware configuration. In addition to the individual tapes (table 2-3), Hewlett-Packard furnishes with each system a complete configured tape, loadable by the basic binary loader and ready for use.

2-146. For loading and running relocatable programs, the routines required to be present in memory are:

a. Input/Output Control: This program supervises the transmission of data between the computer memory and input/output devices. It does this by transferring control to selected subroutines (input/output drivers) on request by the program being run.

b. Input/Output Drivers: A driver subroutine consists of specific instruction sequences to operate one external device, and to request interrupt of the main program when the device is ready for servicing. Driver subroutines are different for each type of device. The control program selects which driver is to be used with a particular device (initially set up by prepare control system).

c. Relocating Loader: This program is required for loading into memory relocatable user programs produced by the assembler and the FORTRAN compiler. (A relocatable program is one which can be shifted upward in memory a specified number of locations relative to location zero. This provides efficient loading of memory by minimizing or eliminating gaps.) Features of the relocating loader enable it to link a number of separately assembled relocatable programs into an integrated unit, assign indirect addressing and base page references, and select and load referenced library subroutines.

2-147. Routines not required for loading or running object programs but which are considered as part of the basic control system are:

a. Debugging Routines: This is a program consisting of several individual routines designed to help check out a user-generated program. Separate routines, which are individually selectable by typing in request statements on the teleprinter keyboard, enable: printing of selected areas of memory (memory dump); executing and printing of selected sections of the program (program trace); modification of selected areas of memory; execution of a program and termination of the program when a specified location or memory reference is used; and punching of a program in

an absolute binary format acceptable to the basic binary loader. The debugging routines program is loaded by the relocating loader.

b. Prepare Control System: This is an independent program used only to establish or change the composition of the basic control system. The desired basic control system components are read into the computer, and the prepare control system instructions load the new basic control system into the last page of memory. The new basic control system is then punched out for a permanent record, and space occupied by the prepare control system can be used for other purposes. This program establishes the equipment tables which input/output control uses to relate software input/output references to specific hardware peripherals.

2-148. SYMBOLIC EDITOR.

2-149. The HP symbolic editor is a program which enables use of the computer to simplify the correction or updating or a user's assembly language or FORTRAN language program (or any other symbolic program), thus avoiding the process of manually repunching the entire program off line. The symbolic editor produces an updated tape from the source tape and change instructions. Individual characters and entire source statements can be inserted, deleted or replaced. The symbolic editor will also provide a listing of a symbolic file, sequentially numbering the statements. Diagnostic messages are produced for errors detected in the format of the edit control statements. System input/output drivers (table 2-3) are required in order to use the symbolic editor.

2-150. ASSEMBLER.

2-151. The HP assembler is a program designed to convert a symbolic source program into either absolute or relocatable binary machine instructions, optionally selectable by the programmer. Basically, the assembler provides a means of using the computer itself to relieve the programmer from the tedious job of coding each instruction of his source program in binary machine language. By reading an input prepared in symbolic form by the programmer (using the three-letter mnemonics defined under paragraph 2-53, plus special assembler pseudo-instructions) the computer can produce (assemble) the full 16-bit binary representation of each instruction. If a relocatable output is to be prepared, the programmer need not be concerned about actual memory addresses, since the relocating loader (paragraph 2-146) will assign these.

2-152. The assembler is contained on a single spool of punched paper tape which, when loaded into the computer, resides in memory throughout the assembly process. To use the assembler, the teleprinter option is required (or an equivalent system) to read the user source program into the computer, punched the assembled result on tape, and print out error messages. System input/output drivers (table 2-3) are also required in order to use the assembler. Two or three passes of the source tape are required, depending on whether or not a printed listing of the assembled program is desired.

2-153. FORTRAN.

2-154. HP FORTRAN is an extended version of ASA (American Standards Association) basic FORTRAN; source programs written according to ASA basic FORTRAN specifications can be compiled and executed on the HP 2114B Computer. FORTRAN, being a compiler language, as opposed to assembler language, provides even greater user convenience since it is still further removed from binary machine language. Whereas the assembler requires a statement for each machine instruction, item for item, FORTRAN accepts statements in a form resembling algebraic formulas (hence the name FORmula TRANslation). Each FORTRAN statement may result in a large number of machine instructions.

2-155. HP FORTRAN is a four-pass system for computers having 4K memory; this reduces to two passes for 8K computers. The compiler is contained on several individual tapes, one for each of the passes. In addition, at least one system input/output driver is required (table 2-3). The output of the compile process is a relocatable machine language object program which can be loaded and executed under control of the basic control system.

2-156. ALGOL.

2-157. ALGOL (ALGOrithmic Language) is a compiler-type language that accepts as input a source program written in a language similar to that defined by the ALGOL 60 Revised Report, Communications of the ACM, January, 1968. In one pass, it produces, as an output, a relocatable binary object program which can be loaded and executed under control of the HP basic control system.

2-158. HP ALGOL allows the computer user to use familiar arithmetic conventions when formulating a program. It also includes the additional advantages of intermixing of real and integer identifiers in assignment statements, all variables treated as own variables, initialization of variables or arrays within type declaration, values assigned to variables with equate declaration, logical unit designation in input/output statements, and HP FORTRAN format specifications for input/output, or free field input data. HP ALGOL requires 8K of core memory.

2-159. BASIC.

2-160. BASIC is an interpretative compiler language that accepts a simple mathematical language which has certain similarities to FORTRAN and ALGOL. Syntax is checked as statements are entered into the computer and error messages are turned immediately. Compilation takes place in the computer memory. (BASIC does not produce an object tape as an output.) When successfully completed, the program is executed. Due to its interactive nature and its simplicity, BASIC is widely used by engineers.

2-161. HP BASIC is similar to that used in the HP time-sharing language and has the added features of a COM statement (to pass information blocks from one program to another), a CALL statement (to use assembly language

subroutines and special purpose input/output device drivers), and a WAIT statement (to temporarily delay program execution). HP BASIC requires 8K of memory.

## 2-162. HARDWARE DIAGNOSTICS.

2-163. To assist the user in hardware troubleshooting, an HP hardware diagnostic package is furnished with all HP 2114B Computers. The programs in this package are separate from the installation and maintenance manual. Operating procedures for the diagnostic programs are contained in the Manual of Diagnostics. The results of the diagnostic tests are used together with maintenance information given in the HP 2114B Installation and Maintenance Manual (Volume Two). Procedures are given in the Manual of Diagnostics to determine that the hardware system is capable of accepting and using the HP hardware diagnostics programs. The supplied software may then be loaded and run according to set procedures. Programs supplied (refer to table 2-3) are:

a. Instruction Tests: These tests check out all instruction codes in groups, halting the computer when an instruction fails to perform its function. The first test program checks out a few basic instructions (alter-skip), so that those instructions can be used the next test program (memory reference), which in turn enables checking out the final group (shift-rotate).

b. Memory Address Tests: A low-core test and a high-core test are supplied as separate test programs, so that the program may be loaded at the end of memory to check all core locations below the test block, or it may be loaded at the bottom of memory to check all higher locations. Each test checks the addressing logic of a selectable section of memory, and halts when an error is detected. The display on the computer front panel is used to identify the error.

c. Memory Checkerboard Tests: These tests, which also consist of a low-core test and a high-core test, verify that data is correctly stored in memory, and is correctly transferred to and from the T-register. Like the memory address tests, the computer halts when an error is detected, and identifies the error on the front-panel display.

d. Input/Output Tests: A separate test program is supplied for each type of input/output device in a user's hardware system. For example, the HP 2752A Teleprinter Test Program checks operation of the print, punch, and read functions with the computer. After it is determined that the print function is operating correctly, the program prints requests for data to be typed in so that the punch and read functions can be checked. Errors are indicated by a printout. (Test programs for other devices require that a message printing facility, such as provided by the HP 2752A Teleprinter, be present in the hardware system.)

# SECTION III

# FUNDAMENTALS OF COMPUTER OPERATION

## 3-1. INTRODUCTION.

3-2. This section describes how the HP 2114B Computer manipulates information internally to execute the basic instructions defined in the preceding specifications section. In the interest of users without previous computer experience, the material in this and the following section is organized to begin at an elementary level, and to progress on the basis of previously given information, in the form of a training course.

3-3. The fundamental operations described in this section (and the following section) are in practice nearly always accomplished with the aid of software and input/output devices. However, for simplicity it will be assumed that the computer is an independent instrument and will be operated only by front panel controls. Additionally, it will be assumed for descriptive purposes that the computer runs slowly enough to observe the operations step by step. When running, the HP 2114B Computer usually reads and executes each instruction in 2.0 or 4.0 microseconds. Thus, only the beginning and ending conditions are normally readable on the front panel display. (Note: It is possible to single-step the computer through each instruction, one phase at a time, by using the SINGLE CYCLE pushbutton. This technique will be used in section IV.)

3-4. The computer performs its operations solely by instructions inserted into its memory by the user. The front panel controls therefore do not operate the computer, but rather are used for entering instructions and data into memory, and for initiating operation at the starting instruction. Very basically, the overall operation is:

a. The user enters instructions and data (all manually set in binary coded numbers on the 16 switches of the SWITCH REGISTER) into computer memory, using the LOAD ADDRESS and LOAD MEMORY pushbuttons.

b. When the program of instructions is in memory and is ready to be run, the user enters the address of the starting instruction, which points the computer to the location in memory where this first instruction has been stored. The SWITCH REGISTER and LOAD ADDRESS switches are used for this purpose.

c. The user presses the RUN pushbutton.

d. The computer reads and executes the instruction contained in the memory cell designated by the starting address.

e. The computer automatically continues to the next and all succeeding instructions, operating on the internally stored data, until reaching a halt instruction.

f. The user, having prepared the instructions and knowing where the computed answer is stored, reads the result. (The LOAD ADDRESS and DISPLAY MEMORY pushbuttons may be used to display the answer on the front panel.)

3-5. FRONT PANEL PRESENTATION.

3-6. To present the material of this section in the most practical form from the user's point of view, the descriptions will relate to the front-panel view of the computer. Figure 3-1 is a simplified block diagram of the computer, showing the relationship of the display registers. The block diagram, which corresponds to the physical layout of the panel (shaded blocks), will be used for descriptions of register operations later in this section.

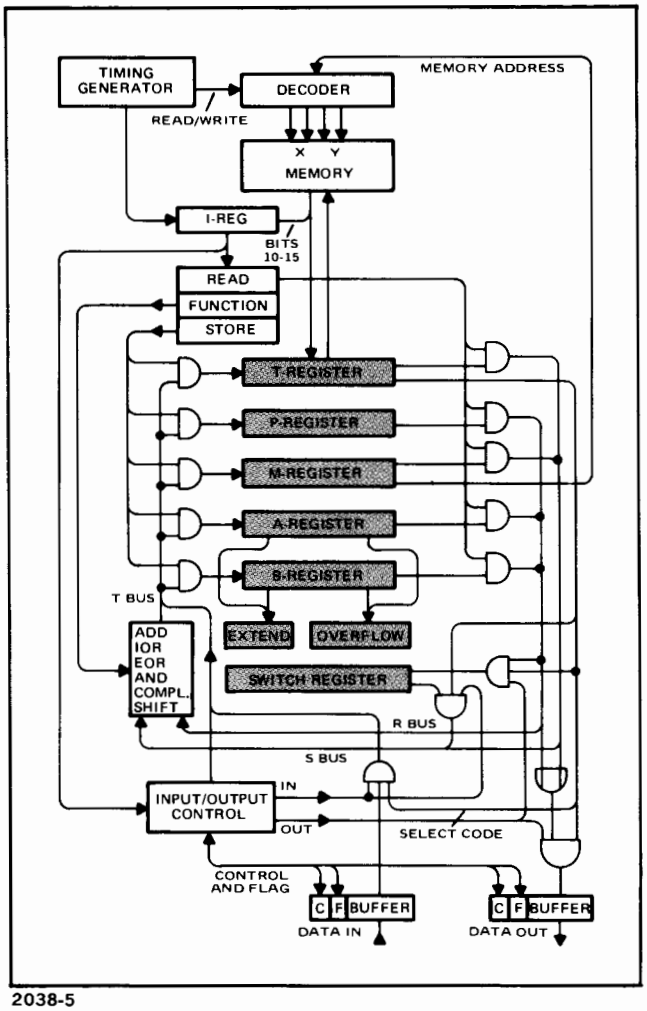


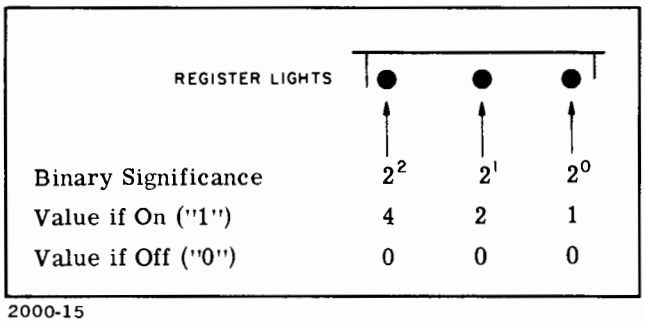

Figure 3-1. HP 2114B Computer Simplified Block Diagram

3-7. As observed from figure 1-1, information is displayed in rows of 16 lights, numbered 0 through 15, and the switch register consists of 16 switches similarly numbered. Each light or switch represents a bit (condensed from "binary digit") in the binary numbering system, where a light or switch off is a "0" and a light or switch on is a "1". In the binary system, there are only two digits, 0 and 1, which are easily stored and manipulated by a computer using bistable devices. Thus input information which is applied to the computer in binary form (such as by the switch register) is said to be in machine language since the computer can handle these numbers directly without conversions of any kind. For the user, however, binary numbers (such as 1011010011101000) are difficult to read and use, so the bits are grouped in threes for convenient notation in the octal numbering system.

3-8. Thus it is seen at this point that before a discussion of computer operation can by presented, some familiarity with both binary and octal numbering systems, as well as with conversions to and from the decimal system, is necessary. The remainder of this introduction (through paragraph 3-39) provides this basic information.

3-9. OCTAL NOTATION. There are five three-bit groups in each row of panel lights and the switch register, with one bit remaining at the left end. Since this last bit, bit 15, is normally used for special purposes (e.g., to indicate direct/indirect addressing or +/- numbers), the following introductory paragraphs, through paragraph 3-22, will disregard this bit and will deal only with the 15 bits numbered 0 through 14. The concept of using bit 15 for signed numbers is introduced later in paragraph 3-35.

3-10. In converting each group of three bits to an octal digit, the binary significance of each bit is converted to its absolute value, which is then considered to be absent or present, depending on whether the bit is a "0" (light off) or a "1" (light on), respectively. This is shown in figure 3-2.




2000-15

Figure 3-2. Composition of Octal Digits

3-11. By various combinations of on and off states, eight digits are possible, 0 through 7. The digits 8 and 9 never appear in the octal numbering system. Figure 3-3 lists all eight binary/octal equivalents, along with some examples of numbers as might be read from a computer display register.

| Binary | | Octal Interpretation | | Octal |
|---|---|---|---|---|
| 000 | = | 0 | = | 0 |
| 001 | = | 1 | = | 1 |
| 010 | = | 2 | = | 2 |
| 011 | = | 2 +1 | = | 3 |
| 100 | = | 4 | = | 4 |
| 101 | = | 4 +1 | = | 5 |
| 110 | = | 4 +2 | = | 6 |
| 111 | = | 4 +2 +1 | = | 7 |

EXAMPLES

| 5 | 2 | 6 | 0 | 1 |
|---|---|---|---|---|
| 1 0 1 | 0 1 0 | 1 1 0 | 0 0 0 | 0 0 1 |

| 7 | 4 | 3 | 5 | 0 |
|---|---|---|---|---|
| 1 1 1 | 1 0 0 | 0 1 1 | 1 0 1 | 0 0 0 |

| 7 | 7 | 7 | 7 | 7 |
|---|---|---|---|---|
| 1 1 1 | 1 1 1 | 1 1 1 | 1 1 1 | 1 1 1 |

2000-16

Figure 3-3. Binary/Octal Conversions

3-12. As can be seen from the last example in figure 3-3, the largest possible number which can be displayed by a register is 77777 (all lights on). Since there are no 8's or 9's in the octal system, this number must correspond to some lower value in the decimal system (specifically 32767; method of conversion given later under paragraph 3-18). To avoid confusion when numbers are written in more than one numbering system, a subscripted digit is attached to the number to identify the system used. Thus:

$$111111111111111_2 = 77777_8 = 32767_{10}.$$

3-13. The HP 2114B Computer manuals will use these subscripts or the word binary, octal, or decimal whenever such confusion may occur.

3-14. OCTAL COUNTING. When counting in the octal system, the "carry" to the next more significant column occurs as rollover from $7_8$ to $0_8$ occurs. That is, $10_8$ follows $7_8$. The counting sequence in octal is:

| | |
|---|---|
| 00000 | 00006 |
| 00001 | 00007 |
| 00002 | 00010 |
| 00003 | 00011 |
| 00004 | 00012 |
| 00005 | etc. |

3-15. NUMBER CONVERSIONS.

3-16. COMPARISON OF SYSTEMS. Integral and fractional parts of a number are separated by a decimal point in the decimal system, an octal point in the octal system, and a binary point in the binary system. The significance of digit positions in a number in any system increases by positive powers of the system base when going left from the point, and decreases by negative powers of the system base when going right from the point. This is shown in figure 3-4.

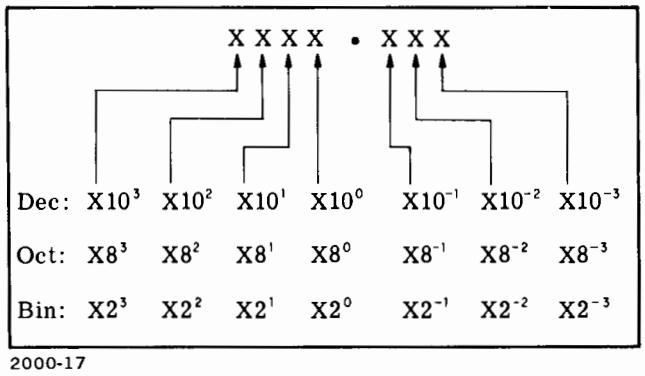



Figure 3-4. Significance of Digits in Three Systems

3-17. The information in figure 3-4 provides the basis for converting octal or binary to the decimal system. The procedure is given in paragraph 3-18. The reverse conversion from decimal to octal or binary is given in paragraph 3-20.

3-18. CONVERTING TO DECIMAL. Converting octal or binary numbers to the decimal system consists only of performing the individual multiplications indicated in figure 3-4 (digit times its significance) for each of the digits in the number, and then summing the individual results. Thus the octal number 7654.321 has the decimal equivalent of:

$$
\begin{aligned}
7 \times 8^3 &= 7 \times 512 = 3584. \\
6 \times 8^2 &= 6 \times 64 = 384. \\
5 \times 8^1 &= 5 \times 8 = 40. \\
4 \times 8^0 &= 4 \times 1 = 4. \\
3 \times 8^{-1} &= 3 \times \tfrac{1}{8} = .375 \\
2 \times 8^{-2} &= 2 \times \tfrac{1}{64} = .03125 \\
1 \times 8^{-3} &= 1 \times \tfrac{1}{512} = .001953125 \\
&\qquad\qquad 4012.408203125
\end{aligned}
$$

3-19. Using this method, the decimal equivalent of the highest whole positive number which can be contained in the computer registers is derived as shown below. (Note: special constructions to represent larger, fractional, and negative numbers will be discussed later.)

$$
\begin{aligned}
111111111111111_2 &= 1 \times 2^{14} = 16384 \\
&\quad 1 \times 2^{13} = 8192 \\
&\quad 1 \times 2^{12} = 4096 \\
&\quad 1 \times 2^{11} = 2048 \\
&\quad 1 \times 2^{10} = 1024 \\
&\quad 1 \times 2^{9} = 512 \\
&\quad 1 \times 2^{8} = 256 \\
&\quad 1 \times 2^{7} = 128 \\
&\quad 1 \times 2^{6} = 64 \\
&\quad 1 \times 2^{5} = 32 \\
&\quad 1 \times 2^{4} = 16 \\
&\quad 1 \times 2^{3} = 8 \\
&\quad 1 \times 2^{2} = 4 \\
&\quad 1 \times 2^{1} = 2 \\
&\quad 1 \times 2^{0} = 1 \\
&\qquad\qquad 32767_{10}
\end{aligned}
$$

$$
\begin{aligned}
77777_8 &= 7 \times 8^4 = 28672 \\
&\quad 7 \times 8^3 = 3584 \\
&\quad 7 \times 8^2 = 448 \\
&\quad 7 \times 8^1 = 56 \\
&\quad 7 \times 8^0 = 7 \\
&\qquad\qquad 32767_{10}
\end{aligned}
$$

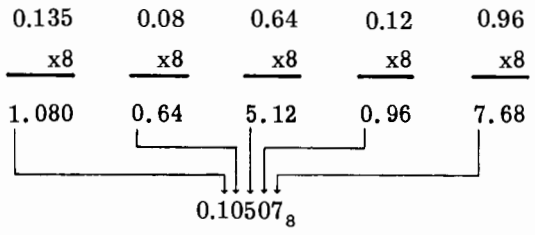


3-20. CONVERTING FROM DECIMAL. Integral and fractional parts of a decimal number require separate operations when converting to the binary or octal system. Because of this added complexity, the ease of octal/binary conversion, and the large number of operations required to construct a 15-bit binary number, it is recommended to limit decimal conversions to octal only, and then to construct the binary equivalent if necessary from the octal number. No example of decimal-to-binary conversion is given here, although the technique is identical to the decimal-to-octal conversion shown.

3-21. Basically, the procedure for the integral part of the number is first to divide the new base (8, if converting to octal) into this part of the number, stopping at the decimal point. The resulting number is a whole number and a fractional remainder (e.g., $32767 \div 8 = 4095$ plus a remainder of 7 eights). The remainder (7) becomes the least significant integer of the new number being constructed (i.e., immediately to the left of the octal point). The whole portion (4095) is again divided by the base (8), and the process is continued until the whole portion is reduced to zero.

$$
\begin{aligned}
32767 \div 8 &= 4095 + 7 \\
4095 \div 8 &= 511 + 7 \\
511 \div 8 &= 63 + 7 \\
63 \div 8 &= 7 + 7 \\
7 \div 8 &= 0 + 7 \\
&\qquad 7\ 7\ 7\ 7\ 7\ .\ _8
\end{aligned}
$$

3-22. To convert the fractional part of a decimal number to octal, multiply by the base and use the whole portion of the resulting number as the first digit to the right of the octal point. Continue by multiplying the fractional part of the same resulting number by the base again, to as many places of accuracy as desired. Thus 0.135 decimal is approximately:




3-23. ARITHMETIC OPERATIONS.

3-24. Since the computer performs arithmetic operations in binary and the user reads the numbers in octal, familiarity with basic binary and octal arithmetic is essential. The important rule to remember when performing arithmetic in any numbering system is that all digits, whether written or carried mentally, must be smaller than the system's base. Thus 2 or 3 cannot appear in the binary system and 8 or 9 cannot appear in the octal system.

3-25. ADDITION. In the decimal system, a carry is generated each time the addition in a column exceeds 9. Similarly, in the octal or binary systems, a carry is generated each time the addition in a column exceeds 7 or 1 respectively.

| | Decimal | Octal | Binary |
|---|---|---|---|
| Carries: | 111 | 111 | 111 |
| | 999 | 777 | 111 |
| | +001 | +001 | +001 |
| | 1000 | 1000 | 1000 |

| | Decimal | Octal | Binary |
|---|---|---|---|
| | | | 11 |
| Carries: | 222 | 222 | 1111 |
| | 789 | 567 | 111 |
| | 789 | 567 | 111 |
| | 789 | 567 | 111 |
| | 2367 | 2145 | 10101 |

3-26. To explain the latter octal addition, note that when adding the rightmost column (3 sevens), the total if adding decimal would be 21, which means that the base (8) has been exceeded twice (i.e., 16 or higher), with a remainder of 5. The remainder of 5 is written as the column sum (just as in the decimal system) and the number of times the base has been reached (2) is carried to the next column (again, this is exactly what is done for a decimal carry). In the case of the latter binary addition, the first column (rightmost) reaches the base once (one carry), while the second and third columns reach the base twice (two carries).

3-27. SUBTRACTION. Borrows from a preceding column have the value of the system's base. Thus a borrow in the decimal system is 10, in octal is 8, and in binary is 2.

| | Decimal | Octal | Binary |
|---|---|---|---|
| Borrows: | 101010 | 888 | 202 |
| | 9123 | 7123 | 1010 |
| | -798 | -567 | -101 |
| | 8325 | 6334 | 101 |

3-28. MULTIPLICATION. As in addition, a carry is generated each time a product reaches a multiple of the base. When the multiplier has more than one digit, remember to perform the final addition in the same system. Carries are not denoted for the second set of examples below: for practice, the reader should work out these problems independently to see how the answers are obtained.

| | Decimal | Octal | Binary |
|---|---|---|---|
| | | | 111 |
| | | | x11 |
| | 394 | 274 | 111 |
| | x5 | x5 | 111 |
| | 550 | 234 | 1001 |
| Carries: | 142 | 142 | 11 |
| | 1970 | 1654 | 10101 |

| | Decimal | Octal | Binary |
|---|---|---|---|
| | 563 | 563 | 1111 |
| | x75 | x75 | x111 |
| | 2815 | 3477 | 1111 |
| | 3941 | 5045 | 1111 |
| | 42225 | 54147 | 1111 |
| | | | 1101001 |

3-29. DIVISION. Division in the octal or binary system is the same as decimal division except that the intermediate multiplications and subtractions must be performed in the appropriate system. The borrows for subtraction are not shown in the examples below; again, the reader should work out every step of these problems to obtain the given answers.

| Decimal | Octal | Binary |
|---|---|---|
| 563 | 563 | 1111 |
| 75 ) 42225 | 75 ) 54147 | 111 ) 1101001 |
| 375 | 461 | 111 |
| 472 | 604 | 1100 |
| 450 | 556 | 111 |
| 225 | 267 | 1010 |
| 225 | 267 | 111 |
| | | 111 |
| | | 111 |

3-30. COMPUTER ARITHMETIC. In the basic instructions of the computer, there is an add instruction but no subtract, multiply, or divide. Therefore these three latter operations must be constructed from the add instruction or by some other method. Although it is possible to perform multiplication and division by successive addition or subtraction respectively, the more efficient method is by register manipulations available through special computer programming. The following paragraphs deal with subtraction and the representation of negative numbers.

3-31. To subtract, the operation is to convert the subtrahend (i.e., the negative number) to its true complement value, and then to add as if both numbers were positive. The result will be the true difference between the two numbers when the last carry digit is removed. Simple logic in the computer drops the excess carry, so that the user need not be aware of it.

3-32. The true complement of a number in any system is obtained by subtracting the number from any power of the base large enough to allow the arithmetic to be performed. That is, five digits are required if four-digit numbers are involved, as shown below. Using the same subtraction examples given in paragraph 3-27, the complements for the negative numbers are:

| Decimal | Octal | Binary |
|---|---|---|
| 10000 | 10000 | 10000 |
| -798 | -567 | -101 |
| 9202 | 7211 | 1011 |

3-33. Then, completing the operation by straight addition and dropping the excess carry, the answers are the same as obtained previously.

| Decimal | Octal | Binary |
|---|---|---|
| 9123 | 7123 | 1010 |
| +9202 | +7211 | +1011 |
| 18325 | 16334 | 10101 |
| or | or | or |
| 8325 | 6334 | 101 |

3-34. In computers such as the HP 2114B, it is simpler to use the one's complement (subtracting from 1's instead of 0's) since this is simply a matter of switching all 1's to 0's and 0's to 1's. This is precisely what the complement instructions do (CMA/B, CME, CCA/B, CCE). Adding one then converts the result to the true two's complement. One's complement in binary corresponds to nine's complement in decimal and seven's complement in octal. Using the same examples:

| | Decimal | Octal | Binary |
|---|---|---|---|
| | 9999 | 7777 | 1111 |
| | -798 | -567 | -101 |
| | 9201 | 7210 | 1010 |
| Add: | 1 | 1 | 1 |
| | 9202 | 7211 | 1011 |

3-35. Negative numbers are constructed and used in the computer in exactly this way. For example, if the negative number $07000_8$ is wanted for some later arithmetic, this number is taken in positive form, one's complemented and incremented, and is then ready for use as a two's complement negative number. Additionally, however, it is necessary to identify the number as negative, and this is done by a one-bit in the bit 15 position. In binary representation:

| | Sign ↓ | | | | | |
|---|---|---|---|---|---|---|
| Positive: | 0 | 000 | 111 | 000 | 000 | 000 |
| Complement: | 1 | 111 | 000 | 111 | 111 | 111 |
| Increment: | | | | | | +1 |
| Negative: | 1 | 111 | 001 | 000 | 000 | 000 |

(equals $171000_8$)

3-36. If it is now desired to perform a subtraction (for example $60000_8 - 07000_8 = 51000_8$), the computer will add the positive number and the two's complement representation of the negative number as shown below. (For comparison, a subtraction producing a negative answer is also shown.) Note that bit 15 is treated as part of the negative number in all arithmetic operations and, unless overflow occurs, it will always come up as a "0" for computed answers which are positive, or as a "1" for negative answers. Since there are only 16 bit places available to represent the total in any register, the final carry (17th bit, carried to the extend register) is disregarded, and the displayed result is the true difference.

POSITIVE ANSWER

| | Binary | Octal |
|---|---|---|
| | 0 110 000 000 000 000 | (+60000) |
| | 1 111 001 000 000 000 | (-07000) |
| (1) | 0 101 001 000 000 000 | (+51000) |

NEGATIVE ANSWER

| | | |
|---|---|---|
| | 1 010 000 000 000 000 | (-60000) |
| | 1 000 111 000 000 000 | (+07000) |
| | 1 010 111 000 000 000 | (-51000) |

3-37. Since the computer instruction list includes basic instructions to perform the positive-to-negative conversion (one's complement and increment), it is usually not necessary for the user to figure the complements before entering them into the computer. It should also be noted that the reverse conversion from negative to positive is done in exactly the same way (one's complement, then increment). Thus if the negative number $07000_8$ is present in computer memory (stored as $171000_8$), conversion back to positive would be:

| | | | | | | |
|---|---|---|---|---|---|---|
| Negative: | 1 | 111 | 001 | 000 | 000 | 000 |
| Complement: | 0 | 000 | 110 | 111 | 111 | 111 |
| Increment: | | | | | | +1 |
| Positive: | 0 | 000 | 111 | 000 | 000 | 000 |

(equals $007000_8$)

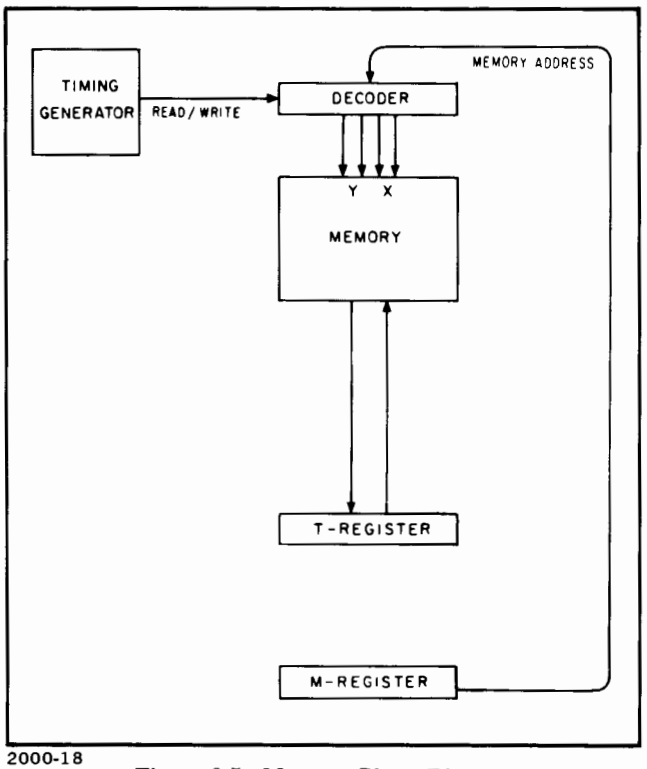
3-38. It should be apparent that as the negative number grows larger, its representation in two's complement form grows smaller. The largest negative number which can be represented in a display register is therefore a one with 15 zeros. This would be equivalent to a positive number of:

| | | | | | | |
|---|---|---|---|---|---|---|
| Negative: | 1 | 000 | 000 | 000 | 000 | 000 |
| Complement: | 0 | 111 | 111 | 111 | 111 | 111 |
| Increment: | | | | | | +1 |
| Positive: | 1 | 000 | 000 | 000 | 000 | 000 |
| | | | | or | $100000_8$ | |
| | | | | or | $32768_{10}$ | |

3-39. This number is one greater than the largest possible positive number ($32767_{10}$, or $077777_8$), as previously noted in paragraph 3-19), since, as shown by the preceding paragraph, $1000000000000000_2$ is legitimately interpreted as $-100000_8$.

## 3-40. COMPUTER STRUCTURE.

3-41. Figure 3-1, the simplified block diagram of the HP 2114B Computer, is the basis for the partial versions used to illustrate descriptions in this section. This figure will be reconstructed step by step as the explanations progress. The first step is figure 3-5, which outlines the blocks and signal routes mentioned in the following discussion of memory, paragraphs 3-42 through 3-49. The block diagrams make use of several "and" gate symbols in addition to circuit blocks. These gates can produce an output only when all inputs are present (true). For example (referring to figure 3-1), data on the T-bus can enter the T-register only if a store signal is also present at the gate leading to the T-register input. Since the store signal is selective (although this is not indicated on the diagram), only this one gate is enabled, while the remaining four are disabled. Thus the data enters only the selected register.



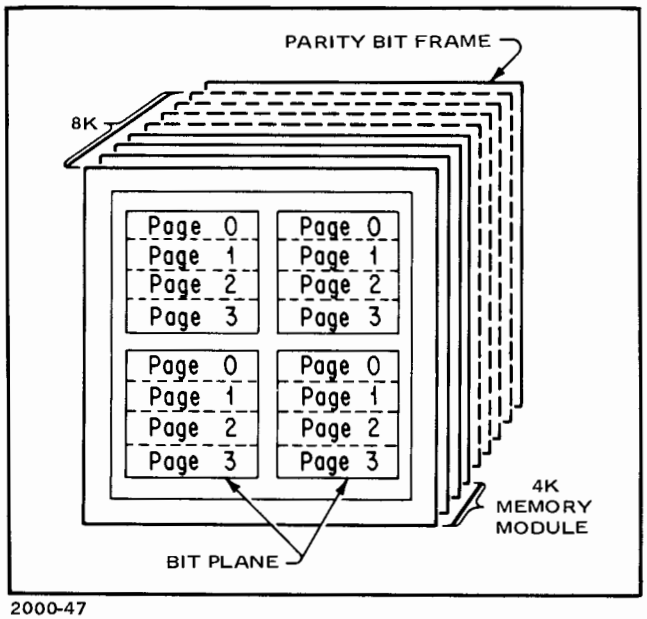

2000-18

Figure 3-5. Memory Block Diagram

3-42. THE MEMORY MODULE.

3-43. A computer's memory is its information storage area. Information is a broad term intended to cover anything which can be represented as a binary number; this includes instruction codes, memory addresses, and alphabetic codes, as well as pure numeric data. The primary storage of the HP 2114B Computer is a core memory, and is internal in the computer. Auxiliary storage for the computer is available in the form of disc storage and magnetic tape; however, these units are accessed through the computer input/output system (paragraph 3-69) and are not treated as an extension of memory in this discussion. Figure 3-6 shows the physical structure of the memory module, and the following paragraphs (through 3-49) describe each of the four components identified in the figure, beginning with the smallest individual component, the ferrite core.

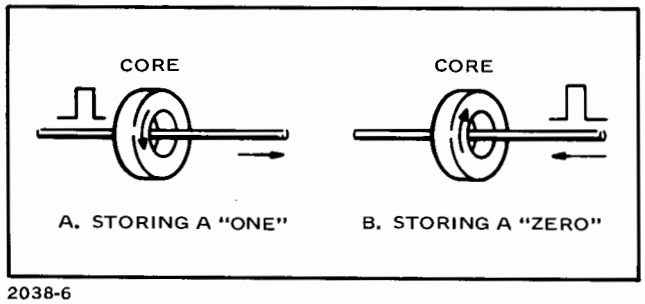
3-44. CORE. As explained in the introduction to this section, the computer handles all information in binary form; i.e., as a number representable by only two digits, zero and one. The ferrite core, which is a small ring of magnetic material, has the ability to store this binary information in that clockwise and counterclockwise magnetization can be assigned digital values of one and zero. By threading a current-carrying wire through the core, the direction of core magnetization can be reversed simply by changing direction of the current. Since the mass of the core is very small (diameter of .03 inch), little magnetizing force is required to switch the binary state, thus permitting fast switching speeds (about 400 nanoseconds in the HP 2114B Computer). The magnetic state remains indefinitely




2000-47

Figure 3-6. Core Memory Module

after the current is removed, so that switching can be accomplished by bidirectional current pulses. This is shown in figure 3-7.



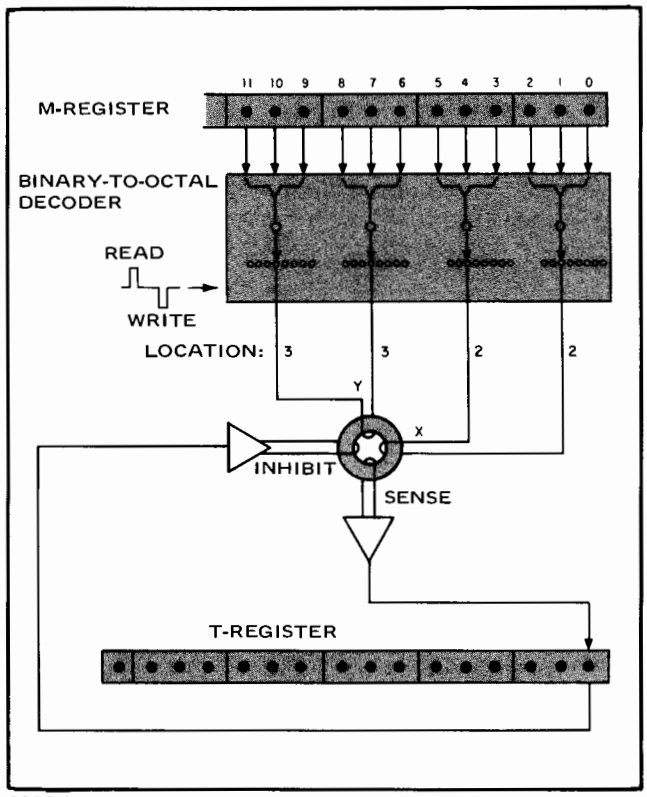

2038-6

Figure 3-7. Binary Storage in a Magnetic Core

3-45. Since it is necessary to be able to select desired units of information in the module, four wires are required to be threaded through each core, as in figure 3-8. In practice, the wires do not loop through the core, as shown for clarity in the figure, but simply pass through the center of a series of cores. Figure 3-8 shows how one bit of information is addressed and transferred to and from the T-register. Action is as follows:

a. Assume that the computer is running, and that the program has set the M-register to a memory location number (address), desiring access to that location.

b. The address from the M-register, consisting of 12 binary bits, is applied to a binary-to-octal decoder, which reduces the 12 binary address lines to four octal lines which thread, in pairs, through the selected core. For purposes of illustration, the diode decoding matrix is shown as four switches. Note that each of these switches can select one of eight ends of X and Y wires, thus making possible 8x8x8x8 = 4096 combinations to address 4096 core




2000-21

Figure 3-8. Core Addressing, Reading, and Writing

locations. The second 4096-word module is selected by the 13th bit (i.e., bit 12).

c. At a specific time in the computer timing sequence (start of each memory cycle), all 16 bits of the T-register are reset to zero.

d. A read pulse is then applied to the decoder. Many cores will receive either Y-current or X-current pulses, neither of which alone is sufficient to switch the state of the core, but only one core out of 4096 on a plane (paragraph 3-48) receives both Y-current and X-current pulses. The read current is always in the direction which would magnetize the core in the "0" direction. (If more than one module is present, module selection is accomplished simply by routing the read pulse to the appropriate module, as determined by bit 12 of the M-register.)

e. If the core was previously magnetized in the "1" direction, the read current, in switching the core, causes a flux change which induces a current into the sense output line. This output is amplified and used to set the corresponding bit flip-flop of the T-register (assumed as bit 0 in figure 3-8). If the core was in the "0" state, there is no flux change and the T-register bit remains "0" (as reset in step "c").

f. Since steps "d" and "e" destroyed the stored information, it is necessary to write the information back. This information, which is now in the T-register, is connected back to the core via the inhibit line. Then the X and Y lines

are pulsed with a write current pulse, which is of opposite polarity to the read pulse (i.e., tending to magnetize in the "1" direction).

g. If the inhibit current is not turned on, the core switches back to the "1" state. If the inhibit current is turned on, it cancels part of the write magnetizing force, so that the core cannot switch, and the core remains in the "0" state.

3-46. The sequence of events in the preceding paragraph briefly describes the computer memory cycle. There are two exceptions which modify the memory cycle slightly: (1) during the execute phase of the store instructions (STA, STB, JSB), the output of the sense amplifier is inhibited, and instead the data to be stored is transferred into the T-register from the A- or B-register during the read time period; (2) during the execute phase of the ISZ instruction (increment, skip if zero), the T-register is incremented between the read and write time periods.

3-47. MEMORY LOCATION. The word length of the HP 2114B Computer is 16 bits, only one of which is shown in figure 3-8. To store one 16-bit word, 16 cores are required, as indicated in figure 3-6. These 16 cores comprise a memory location, sometimes also referred to as a memory cell. When information is transferred into or out of a memory location, the information in all 16 bits must be transferred simultaneously. Therefore the X and Y selection lines will be strung through the 16 cores, causing reading and writing of all 16 cores simultaneously. Figure 3-9 illustrates this, showing only three cores for simplicity. Note that each of these cores is on a different bit plane.

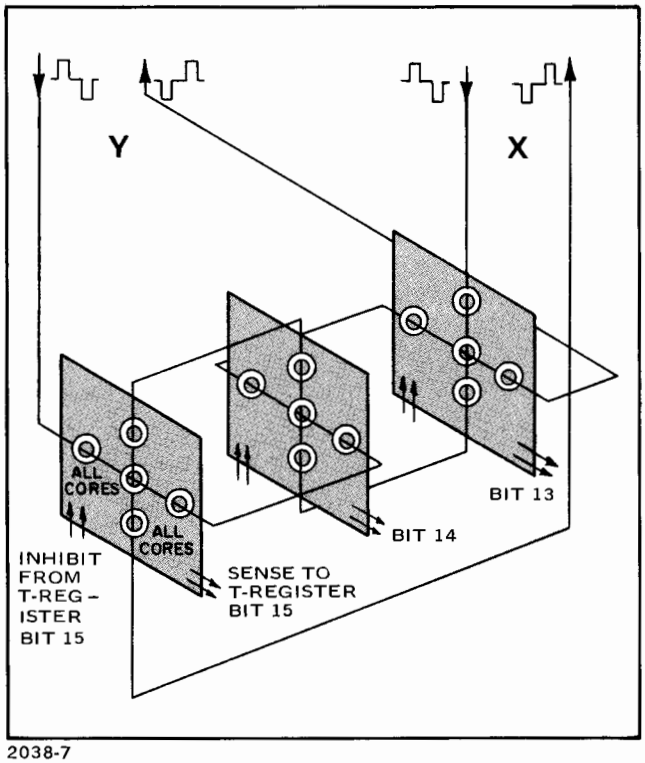



Figure 3-9. Memory Cell Selection

3-48. BIT PLANE. Cores are strung on a grid of wires as shown in figure 3-10. There are 4096 cores on this grid, called a bit plane, and a 4K module consists of five frames (nine frames if 8K) with each of the first four frames (first eight if 8K) having four bit planes each (figure 3-6) and the fifth frame (ninth if 8K) having only one (two if 8K) bit plane. The last frame in both the 4K and 8K module contain the bit plane (or planes) for the parity bit. Each bit position of the T-register is wired by the sense and inhibit lines through all 4096 cores on the corresponding bit plane. Since only one core on an individual bit plane is sensed (addressed) at a given instant of time, the sense line needs only to detect a flux change anywhere on the bit plane. Similarly, the inhibit signal is applied to the entire bit plane when writing, but actually affects only the selected core.

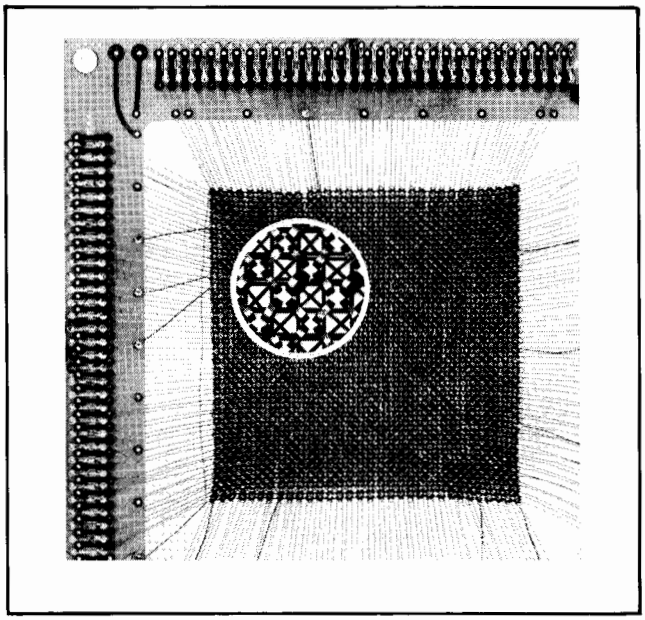
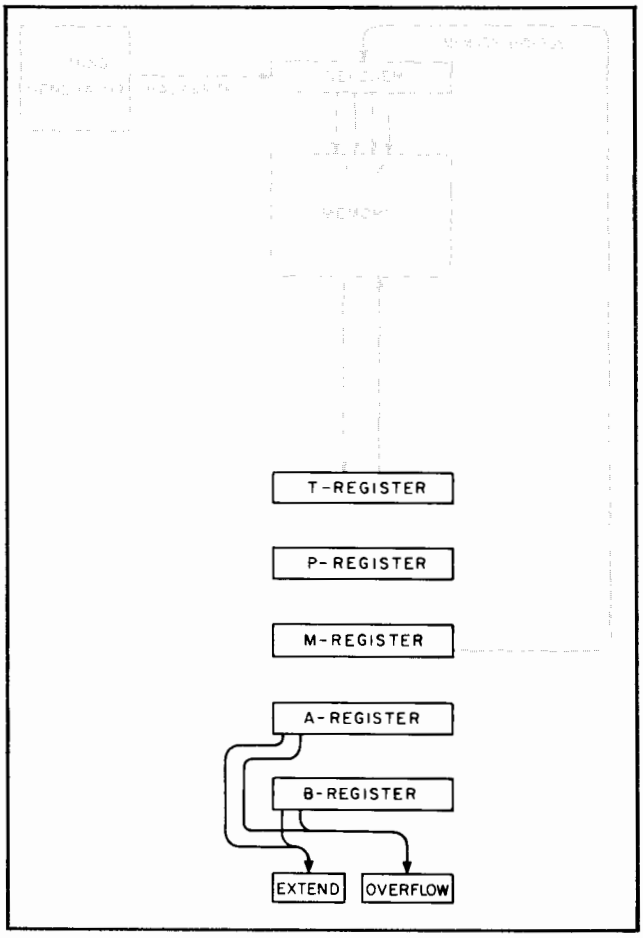


Figure 3-10. Memory Bit Plane and Frame (Upper Left Corner)

3-49. PAGE. Pages of memory are not physical divisions of the module. Wiring of the bit planes is symmetrical and does not account for page boundaries. The page boundaries are determined only by the bit format of memory reference instructions, and are shown as broken lines in figure 3-6 for visualizing the physical placement of memory pages.

3-50. THE REGISTERS.

3-51. Figure 3-11 shows the seven working registers of the computer. The five principal registers (T, P, M, A, B) are purposely shown as being independent of each other since, in fact, information is not transferred directly from register to register. Rather, information is transmitted via the bus system (described later under paragraph 3-59) under command of the instruction logic (paragraph 3-63). The following paragraphs, through 3-58, explain why the registers are needed, not how they are operated. In essence, these registers are short-term information storage devices consisting of flip-flop circuits, with front-panel indicator lamps to indicate the status of each bit (M- and T-registers only).




2000-24

Figure 3-11. Register Block Diagram

3-52. T-REGISTER. The T-register was briefly mentioned in the description of how memory operates (paragraph 3-45). As can be assumed from that description, and from the front-panel engraving (MEMORY DATA), the T-register holds data that is read out of and written into memory. For the majority of operations when a computer is running, the principal concern is with the data read out of a memory cell; once a word of information is in the T-register, it is accessible for arithmetic operations and for transfers to other registers via the bus system. For the reverse (write) operation, the T-register is loaded by transfers from other registers, and the information is stored in memory during the latter half of the memory cycle.

3-53. P-REGISTER. The P-register is the computer program counter. This means that this register goes through a step-by-step counting sequence and causes the computer to read successive memory locations, corresponding to the existing count. In the simplest case, the P-register would start at zero when the RUN pushbutton is pressed, causing memory location 00000 to be read into the T-register; the computer would act on the instruction code in the read-out data, then advance the P-register to one (memory location 00001). This process of stepping through memory locations (at a rate of 2.0 or 4.0 microseconds per step for most instructions) continues until one of the instructions read out is a halt, which terminates the program. Of necessity, this simple case is not typical. First, programs do not normally begin at locations lower than 00077, since these locations are reserved for special purposes (paragraph 2-28). Therefore the starting address of a program must be manually set into the P-register before pressing RUN. Second, the strict sequential stepping can be altered in the course of a program, either by a skip instruction (which causes the P-register to increment by two instead of one, thus skipping one memory location) or by a jump instruction (which transfers numbers from another register into the P-register, thus causing the program to continue at a different point in memory).

3-54. M-REGISTER. As implied by figure 3-1, the M-register (MEMORY ADDRESS) is the means of addressing specific memory locations. The addressing of memory was previously discussed in paragraph 3-45. The setting of the M-register can occur from any of the other registers, depending on the effects of instructions. In the preceding paragraph, it could be assumed that the P-register directly addresses memory; in actual fact, however, the computer must transfer the desired address from the P-register to the M-register, which in turn addresses the desired memory location. Thus it is seen that these two registers will frequently contain the same number. The reason why both registers are needed is that it is necessary for one register (the P-register) to keep track of the location of the current instruction in case the instruction is a multiple phase type. In this case, the M-register may have to be changed several times in the course of executing an instruction. A common example would be when the instruction is to add the contents of location $100_8$ to the A-register (ADA 100). The P- and M-registers would be identical while reading this instruction out of memory (say the instruction is in location $500_8$; both registers indicate this value). Then the M-register would have to change to 100 to get the contents of this location for the addition. After the addition has been executed, the contents of the P-register are incremented by one ($501_8$). The P and M registers are then both set to this new value, and the computer is then ready to read the next instruction.

3-55. A-REGISTER. The A-register is one of the computer's two accumulators. An accumulator in a computer accumulates the results of arithmetic operations. A simple example was given in the preceding paragraph, where one number from memory was added to the existing contents of the A-register. Assuming that the A-register previously held the number $1000_8$, and the number in location 100 was $22_8$, the number left in the A-register after execution of the instruction would be $1022_8$. Other types of operations which may be done with the A-register are: boolean logic operations ("and", "exclusive or", "inclusive or"), comparison for equality with a memory word, shifting or rotating of bits left or right, testing the status of individual bits, complementing of bits, and accepting or holding data for transfer to and from external devices. All of these operations are accomplished by the instruction logic (paragraph 3-63).

3-56. B-REGISTER. The B-register is the second of the two accumulators. It has the same capabilities as the A-register, except that the three boolean logic instructions (AND, XOR, IOR) can apply only to the A-register. The main reason for having two accumulators is to provide faster, more flexible arithmetic than can be accomplished with one accumulator. This advantage will be seen later in programming of the computer.

3-57. EXTEND. The extend register is shown connected to bit 15 (left end bit) of both A and B registers. This is to indicate that this one-bit register becomes set whenever there is a carry out of bit 15 of either accumulator; i.e., whenever the quantity accumulated exceeds 16 ones. This fact is frequently of significance. For example, if the quantity in an accumulator is 16 ones and an ADD instruction adds one, the result in the accumulator will be 16 zeros. This answer is obviously incorrect; it is correct if the extend bit, which is now in the set state ("1") is temporarily assumed to be "bit 16". The program can be written to make this assumption, and it can proceed without error on the basis of the resulting information. To be certain that the extend information is valid, the extend register is normally cleared by an instruction (CLE) before the addition is done. Another valuable feature of the extend register, is its ability to link the two accumulators (effectively providing a single 32-bit accumulator).

3-58. OVERFLOW. The overflow register is similar in purpose to the extend register. The difference is that, whereas the extend register indicates that the largest 16-bit quantity has been exceeded, the overflow register indicates that the largest signed quantity has been exceeded. (A program may work with both signed and unsigned numbers.) Since bit 15 is the sign bit, bit 14 (as shown in figure 3-11) is the source of the significant carry. Having two possible signs (+ and -) means that detection of overflow requires two different sets of conditions. For addition of two positive numbers, overflow occurs if there is a carry from bit 14 to bit 15 in one of the accumulators. For addition of two negative numbers (which are represented in two's complement form), overflow occurs if there is not a carry from bit 14 to bit 15. Obviously overflow cannot occur when adding numbers of opposing signs, since the resulting quantity cannot be greater than the larger of the two numbers. As with the extend register, the overflow register should be cleared before an addition.

3-59. THE BUS SYSTEM.

3-60. Figure 3-12 outlines the routes by which data travels internally from one register to another. Although the buses are represented by a single line in this figure, assume each line to be composed of 16 individual lines, one for each register bit. Included in the figure is an arithmetic logic block, which has not previously been discussed. It is shown here mainly to illustrate the linkage between buses.

3-61. The HP 2114B Computer uses an R-S-T bus configuration. This is a conventional notation designating a three-bus system which applies two input buses (R and S) to an arithmetic unit with output on the third bus (T). The use of two input buses permits arithmetic operations combining the contents of two registers. A common example would be the execution of the ADA 100 instruction previously used in paragraphs 3-54 and 3-55. In this example, the contents of location 100 is the number $22_8$. During execution of the instruction, this number (22) would be read into the T-register. The other number ($1000_8$) is in the A-register. Simultaneously (by a method described under paragraph 3-63) both the T-register and the A-register are read onto their respective buses (S and R). The two numbers are added in the arithmetic logic circuits, and the result ($1022_8$) is stored via the T-bus back into the A-register as the accumulated sum.

3-62. Note that several register combinations are possible as inputs to the arithmetic logic. One point worth noting is that since the A and B registers are addressable as memory locations, the contents of these registers can be transferred via the T and R buses into the T-register. From this point, the contents can be combined in the manner described above with either accumulator (including combining the number with itself; e.g., add A to A). This is all accomplished in one instruction.

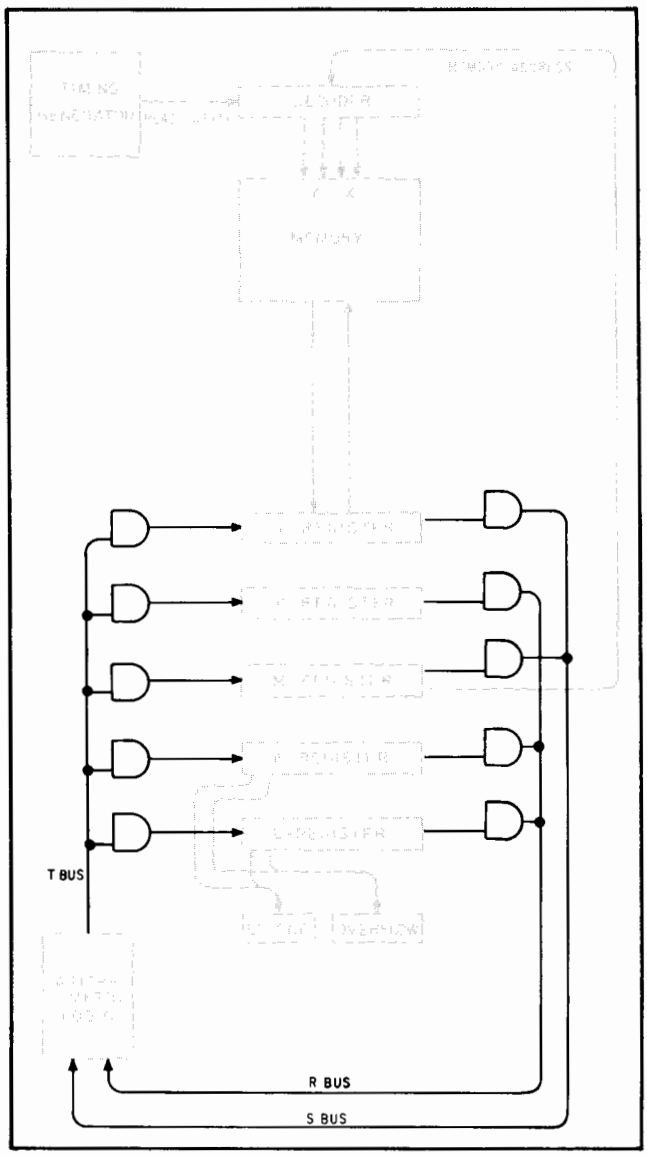
3-63. THE INSTRUCTION LOGIC.

3-64. Figure 3-13 shows the elements of the instruction logic in the HP 2114B Computer. As indicated in the figure, timing is essential to the operation of the instruction logic. The following descriptions do not detail all timing relationships, since these vary with instructions, but it should be understood that timing pulses are gated with each operation to make it occur in proper sequence. A general introduction to machine timing is given in paragraph 2-13 of the specifications section.

3-65. As shown in figure 3-13, the six most significant bits read out of memory during each memory cycle are applied to the 6-bit instruction register, which decodes the instruction. (Actually, the instruction register receives its information via the T-register; for simplicity figure 3-13 shows a direct connection to memory.) Only during the fetch phase, however, are these bits recognized as an instruction code (as determined by a fetch phase signal from the timing generator). At this time, the decoded instruction enables three functional operations, which in turn will become active at specific times, depending on the instruction. These operations are described individually in the next three paragraphs.

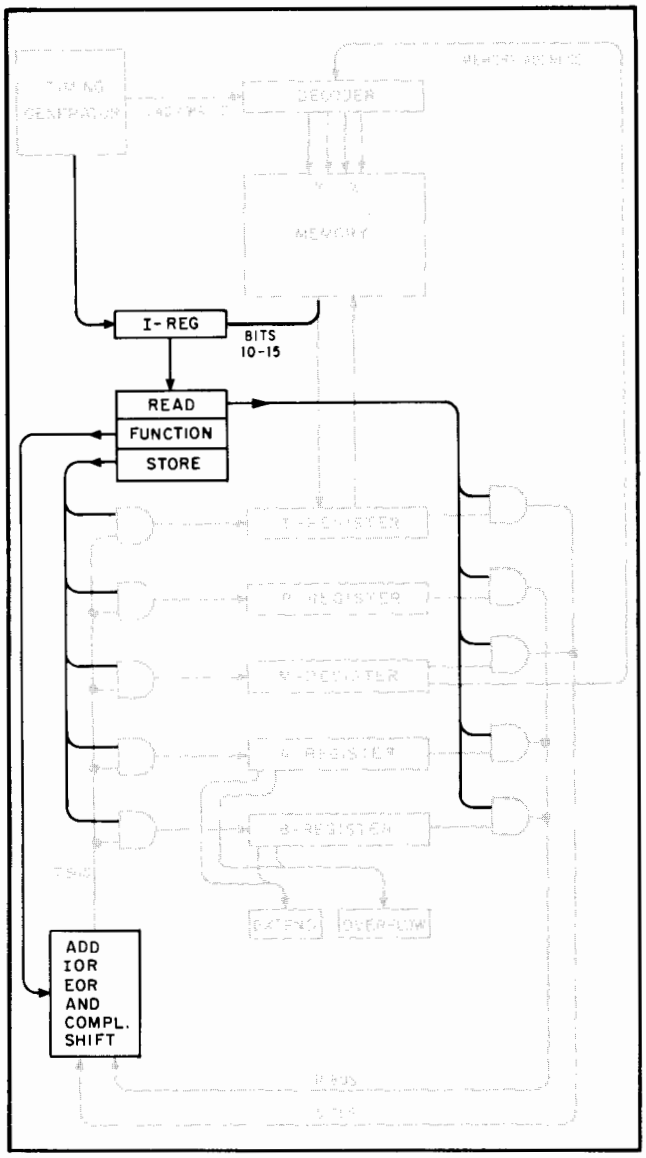
3-66. READ. The read signal, shown connected to the output gate of all five working registers, strobes the data of one or two registers onto their corresponding buses (R and S). This places the data at the inputs of the arithmetic logic circuits.

3-67. FUNCTION. The function signal activates one of the six listed arithmetic functions. The selected function alters or combines the data on the R- and/or S-buses, and routes the resulting data out on the T-bus.




2000-25

Figure 3-12. Bus System Block Diagram




2000-26

Figure 3-13. Instruction Logic Block Diagram

3-68. STORE. The store signal, shown connected to the input gate of all five working registers, effectively opens the input of one or more of these registers to accept the data which appears on the T-bus (preceding paragraph). In many cases, depending on the instruction, only part of the information on the T-bus is stored into a register.

3-69. THE INPUT/OUTPUT SYSTEM.

3-70. Figure 3-14 shows the means by which data is transferred in and out of the computer. This is the input/output system; all elements shown are contained within the mainframe. Interface arrangements are shown for only two external devices, one input and one output. Actually the arrangement has the capability to handle seven interfaces in the mainframe. The switch register is shown as part of the input/output system, and is considered to be an input/output device.

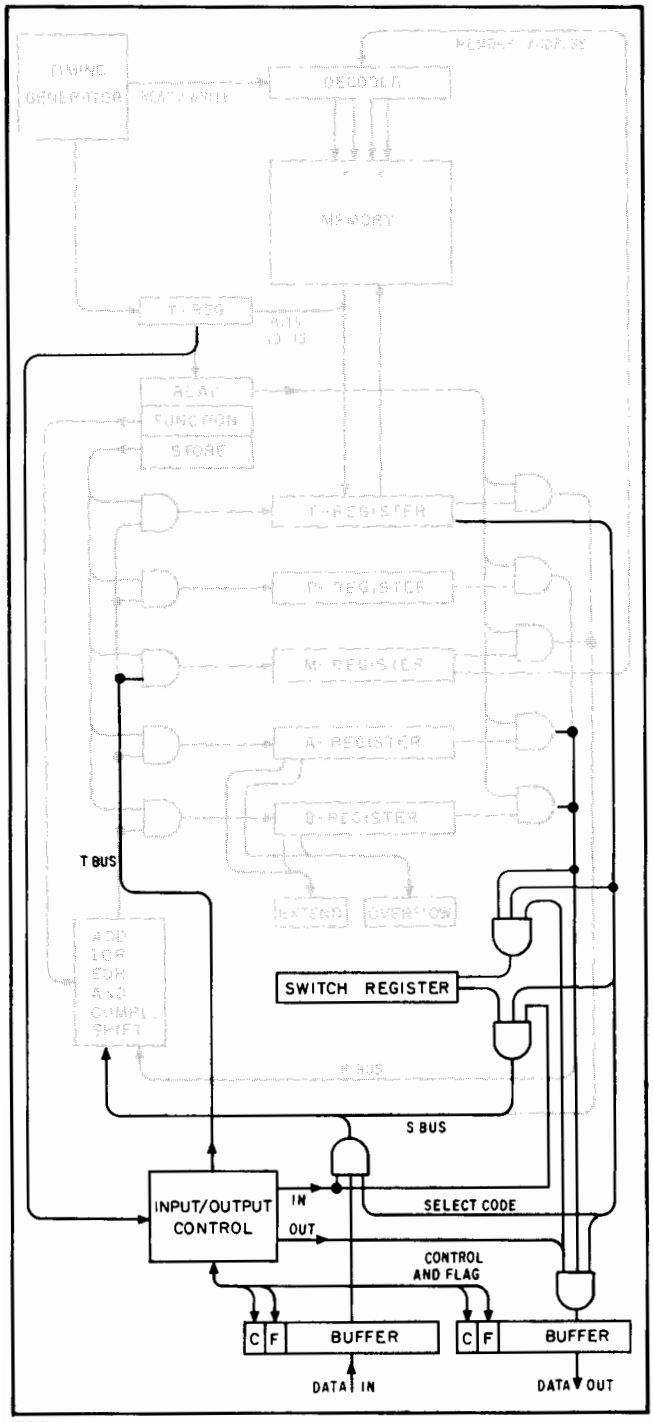
3-71. As indicated by figure 3-14, the input/output control logic is used to process all input/output operations. Input/output control operates in two ways:

a. Processes input/output instructions.

b. Processes service requests by peripheral devices.

3-72. These two types of operations are separately discussed in the following paragraphs.

3-73. PROCESSING INSTRUCTIONS. Input/output instructions decoded by the instruction register are routed to input/output control, which translates the instruction into appropriate driving signals. One such signal is an in signal, which strobes all interface positions for input (represented by two "and" gates in figure 3-14, one accepting data from a buffer register and one accepting data from the




2038-8

Figure 3-14. Input/Output System Block Diagram

switch register). Only one of these interface positions can be enabled, according to the select code (bits 0 through 5 from the T-register), and the corresponding data is strobed by the in pulse onto the T-bus. From there it is transferred via the T-bus into the A- or B-register (as enabled by a store signal at the A or B input gate).

3-74. Another driving signal is the out signal. This signal strobes all interface positions for output (one shown in figure 3-14). The select code from the T-register enables one interface position, and permits the out signal to strobe the data on the R-bus into the corresponding output buffer. (The data on the R-bus was read out of the A- or B-register by a read signal.)

3-75. In addition to transferring data, as in the preceding two paragraphs, input/output control can (according to instruction) send out signals to test the state of control and flag bits (C and F) or to set or reset these bits. The select code determines which interface will receive the signal from input/output control. The control and flag bits are command signals for transferring data between the buffer and the peripheral device (peripheral not shown).

3-76. PROCESSING SERVICE REQUESTS. If a specific instruction has at some previous time enabled the interrupt system (considered to be in the input/output control block in figure 3-14), a peripheral device may request new data from the computer (if output) or request to feed new data to the computer (if input). This request for service is done by setting the interface flag bit. The flag signal, via input/output control, interrupts the computer's operation by forcing the M-register to be set (via the T-bus) to a memory address uniquely specified by the flag. At the same time, the fetch phase is set so that the computer must execute the instruction contained in the specified memory cell. Generally this instruction will be a jump to a service subroutine. This subroutine consists of instructions that will prepare or accept the new data. On completion of service, it is the subroutine's responsibility to return the P- and M-registers to the values they contained before being interrupted.

## 3-77. IMPLEMENTATION OF INSTRUCTIONS.

3-78. The following paragraphs, through 3-154, describe how the 70 basic instructions are implemented internally in the computer. The three illustrations on the following pages expand on the machine timing diagram (figure 2-2) given in section II, specifications. Figure 3-1, the simplified block diagram, is also used as a reference throughout the following descriptions. Most signals named can be identified in this figure; e.g., "read A onto R bus" is the line from the read block to the A-register output gate (which outputs onto the R bus). The block diagram should be referred to frequently as the discussion progresses, in order to visualize the bit manipulations. The right-pointing arrows in the figures should be read as into or onto (e.g., into T-register, or onto R-bus). New mnemonics are introduced in these descriptions which will be defined within the text; however the alphabetical listing of mnemonics in the appendix of this volume may also be referred to if necessary.

3-79. The cycle of time periods shown at the top of figures 3-15, 3-16, and 3-17 (T0 through T7) repeats continuously every 2.0 microseconds while computer power is on. The read/write memory cycle, although shown only once at the top of each of these figures, actually occurs once in every phase (except interrupt). It is important to remember this throughout the following descriptions.

3-80. MEMORY REFERENCE.

3-81. By comparing figures 3-15, 3-16, and 3-17, it is seen that memory reference instructions are the only type of instructions requiring more than one machine phase to execute; indirect and execute phases are associated only with memory reference instructions. In the case of all these instructions except JMP, the action during the fetch and indirect phases (phases 1 and 2) is similar, so these phases are shown only once, implying that they are common to all memory reference instructions. The exception, JMP, is unique in that it does not use an execute phase; execution can occur in either the fetch or the indirect phase. The action for JMP is shown separately in figure 3-15 and is discussed first below.

Note

The descriptions for JMP and AND instructions are more detailed than for succeeding instructions, which are similar in many respects. These two should therefore be studied in detail before advancing to the others. It should also be noted that the descriptions assume knowledge of instruction definitions, as outlined in the specifications (paragraph 2-60.)

3-82. JMP. The fetch phase for all instructions, regardless of type, begins in exactly the same way, since at this time the computer logic cannot know anything about the instruction which is about to be read out of memory. The only fact known is that the word from memory will be read as an instruction (not data); getting an instruction from memory is the first function of the fetch phase. During the first three time periods of the fetch phase, the following actions occur:

a. During T0 the T-register is cleared.

b. The read portion of the memory cycle begins to read the contents of the currently addressed memory cell into the T-register. This continues until the middle of T2.

c. During T1 the instruction register is cleared.

d. Bits 10 through 15 (the instruction group and code identification) of the T-register are transferred into the six-bit instruction register at the middle of T2.

3-83. During the latter portion of T2, the functions to be used in implementing the JMP instruction are set up. This includes read and store as well as any arithmetic functions (none in the case of JMP). Functions are gated with time periods to occur in the correct sequence.

3-84. At this point in time (end of T2), the instruction information is in bits 10 through 15 of the T-register, and in the instruction register. The memory address information is in bits 0 through 9 of the T-register. The next event to occur is to clear the P-register at time T5 if the page zero condition exists (i.e., if bit 10 of the instruction register is a zero). This is done by a store T-bus into P function. Since nothing has been read onto any of the buses, the T-bus is in the all-zero state, and 16 zeros are therefore stored into the P-register. (Actually, for resetting the program to page zero, it is only necessary to clear bits 10 through 14 of the P-register; however it is convenient to clear the entire P-register at this time.) Note that the 6 most significant bits of the page zero address are zeros (refer to paragraph 2-26); e.g., the last address on page zero is:

0 000 001 111 111 111

3-85. During time periods T6 and T7, the page zero indicator (if present) clears bits 10 through 15 of the M-register (not the entire register). The method is the same as described above: store T-bus in M-register, bits 10 through 15; the T-bus is still all zeros. Thus at this time both the P- and M-registers point to page zero, if so coded by bit 10 being a zero (otherwise these registers are not changed, leaving bits 10 through 15 at the current page indication).

3-86. Also during T6 and T7, the direct/indirect bit (bit 15) of the T-register is looked at, to see if the memory address currently in the T-register is the effective address (the final address being jumped to), or if another jump should be made from that address to whatever address is contained in that location (indirect addressing). Since the concept of indirect addressing is important and not always simple to grasp initially, it is treated separately in following paragraphs. For direct addressing, the execution is completed by the following steps:

a. The T-register contents are read onto the S-bus, and appear on the T-bus.

b. Bits 0 through 9 of the T-bus are stored in the P- and M-registers. This directs the computer to the jump location. (Remember from the preceding paragraphs that bits 10 through 15 of the P- and M-registers either have been reset to zero for page zero or have been left alone for current page.)

c. The phase 1 (fetch) condition remains set so that the contents of the jump location will be read out and interpreted as an instruction during the next machine phase.

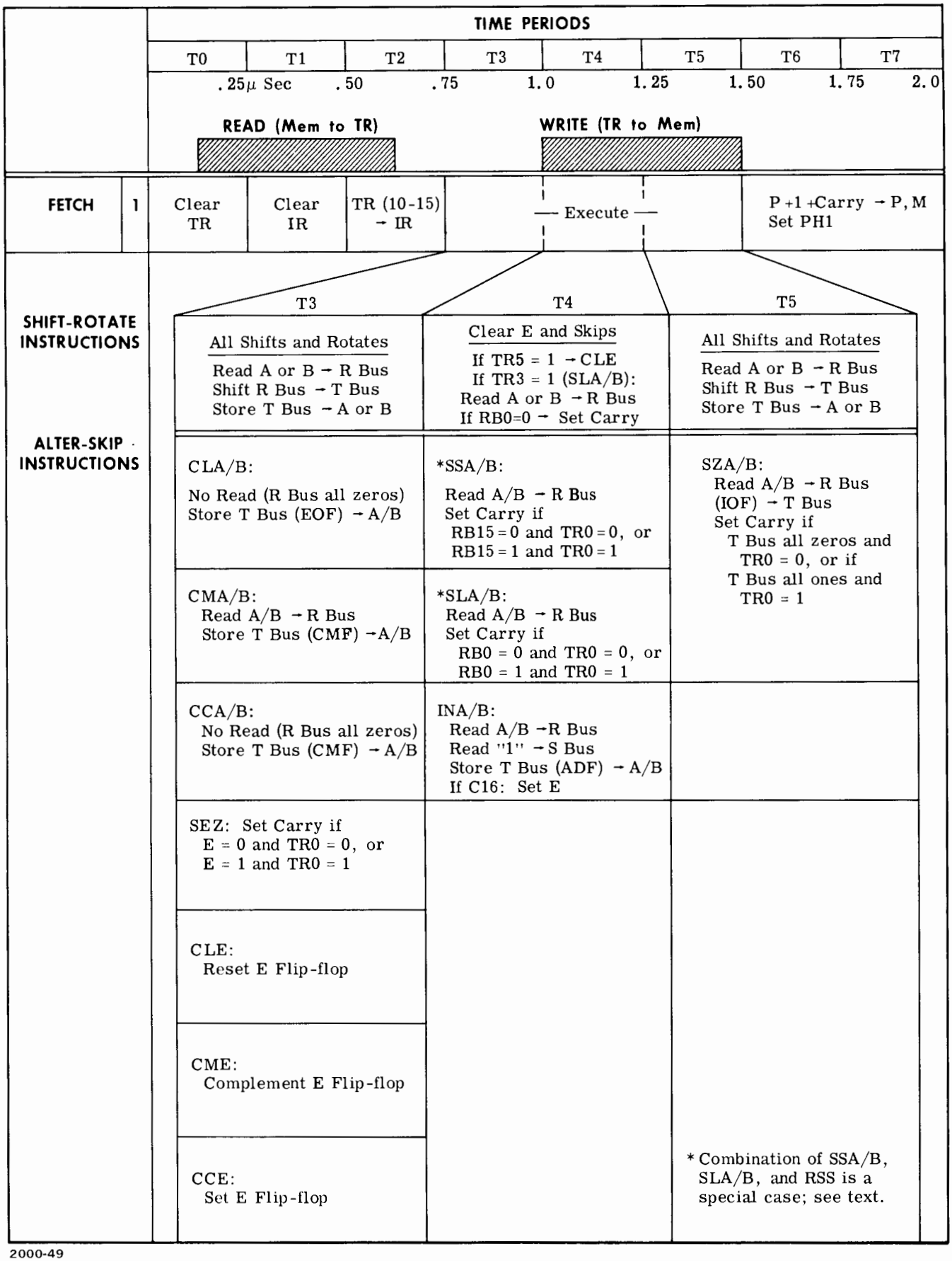
3-87. Basically, the indirect addressing indicator (bit 15 of T-register being a one) tells the computer logic that the contents of the location being jumped to is not the next instruction, but rather the address for another jump. This additional jump is a continuation of the same instruction, but requires an additional phase. During T6 and T7 of phase 1, the T-register contents are transferred to the M-register (not both P and M as for the direct condition). During T7 the phase 2 condition (PH2) is set, and the indirect phase begins.

3-88. During T0, the T-register is cleared. Since the jump is still in progress, the instruction register is not cleared during T1. The contents of the location now addressed by

| PHASE | | TIME PERIODS T0 | T1 | T2 | T3 | T4 | T5 | T6 | T7 |
|---|---|---|---|---|---|---|---|---|---|
| | | .25μ Sec | .50 | .75 | 1.0 | | 1.25 | 1.50 | 1.75 2.0 |
| | | READ (Mem to TR) | | | | WRITE (TR to Mem) | | | |
| FETCH (JMP) | 1 | Clear TR | Clear IR | TR(10-15) → IR (Set Functions) | | | If Z: 0 → P | | If Z: 0 → M (10-15)<br>If D: TR → P, M (0-9) and set PH1<br>If I: TR → M (0-9) and set PH2 |
| INDIRECT (JMP) | 2 | Clear TR | | | | | | | If D: TR → P, M and set PH1<br>If I: TR → M and set PH2 |
| | | | | | | | | | |
| FETCH | 1 | Clear TR | Clear IR | TR (10-15) → IR (Set Functions) | | | | | TR → M (0-9)<br>If Z: 0 → M (10-15)<br>If I: Set PH2<br>If D: Set PH3 |
| INDIRECT | 2 | Clear TR | | | | | | | TR → M<br>If I: Set PH2<br>If D: Set PH3 |
| EXECUTE AND | 3 | Clear TR | | | Read A → R Bus<br>Read TR → S Bus<br>Store T Bus (ANF) → A | | | | Read P → R Bus<br>Read "1" → S Bus<br>Store T Bus (ADF) → P, M<br>Set PH1 |
| XOR | | Clear TR | | | A (EOF) TR → A | | | | P+ 1 → P, M<br>Set PH1 |
| IOR | | Clear TR | | | A (IOF) TR → A | | | | P + 1 → P, M<br>Set PH1 |
| JSB | | Clear TR Inhibit Mem. Data | P+1 → TR | | M → P | | | | P +1 → P, M<br>Set PH1 |
| ISZ | | Clear TR | | | TR+1 → TR<br>If C16: Set Carry<br>Inhibit Write | | Write (Add 0.5 μ Sec) | | P+1+ Carry → P, M<br>Set PH1 |
| ADA/B | | Clear TR | | | If A: A (ADF) TR → A<br>If B: B (ADF) TR → B<br>If C16: Set E | | | | P + 1 → P, M<br>Set PH1 |
| CPA/B | | Clear TR | | | If A: A (EOF) TR → T Bus<br>If B: B (EOF) TR → T Bus<br>If T Bus not zero, set Carry | | | | P + 1 + Carry → P, M<br>Set PH1 |
| LDA/B | | Clear TR | | | If A: TR → A<br>If B: TR → B | | | | P + 1 → P, M<br>Set PH1 |
| STA/B | | Clear TR Inhibit Mem. Data | If A: A → TR<br>If B: B → TR | | | | | | P+ 1 → P, M<br>Set PH1 |

2000-48

Figure 3-15. Implementing Memory Reference Instructions




2000-49

Figure 3-16. Implementing Register Reference Instructions

| PHASE | | TIME PERIODS T0 (.25μ Sec) | T1 (.50) | T2 (.75) | T3 (1.0) | T4 (1.25) | T5 (1.50) | T6 (1.75) | T7 (2.0) |
|---|---|---|---|---|---|---|---|---|---|
| | | | READ (Mem to TR) | | | WRITE (TR to Mem) | | | |
| FETCH<br>HLT | 1 | Clear TR | Clear IR | TR(10-15) → IR | | | | P+1 → P, M<br>Reset Run FF | |
| STF | | Clear TR | Clear IR | TR → IR | Set Flag: Select Code | | | P+1 → P, M<br>Set PH1 | |
| CLF | | Clear TR | Clear IR | TR → IR | Set Flag: Select Code | Clear Flag: Select Code | | P+1 → P, M<br>Set PH1 | |
| SFC | | Clear TR | Clear IR | TR → IR | SFC → Interface | SKF → Carry | | P+1+Carry → P, M<br>Set PH1 | |
| SFS | | Clear TR | Clear IR | IR → IR | SFS → Interface | SKF → Carry | | P+1+Carry → P, M<br>Set PH1 | |
| MIA/B | | Clear TR | Clear IR | TR → IR | | Read A/B → R Bus<br>Buffer → S Bus<br>Store T Bus (IOF) → A/B<br>TR9: CLF | | P+1 → P, M<br>Set PH1 | |
| LIA/B | | Clear TR | Clear IR | TR → IR | | Buffer → S Bus<br>Store T Bus (IOF) → A/B<br>TR9: CLF | | P+1 → P, M<br>Set PH1 | |
| OTA/B | | Clear TR | Clear IR | TR → IR | | Read A/B → R Bus<br>R Bus → Buffer<br>TR9: CLF | | P+1 → P, M<br>Set PH1 | |
| STC | | Clear TR | Clear IR | TR → IR | | Set Control (Sel. Code) | | P+1 → P, M<br>Set PH1 | |
| CLC | | Clear TR | Clear IR | TR → IR | | Clr. Control (Sel. Code) | | P+1 → P, M<br>Set PH1 | |
| STO | | Clear TR | Clear IR | TR → IR | STF → Overflow | | | P+1 → P, M<br>Set PH1 | |
| CLO | | Clear TR | Clear IR | TR → IR | | CLF → Overflow | | P+1 → P, M<br>Set PH1 | |
| SOC | | Clear TR | Clear IR | TR → IR | SFC → OVF | SKF Carry | | P+1+Carry → P, M<br>Set PH1 | |
| SOS | | Clear TR | Clear IR | TR → IR | SFS → OVF | SKF → Carry | | P+1+Carry → P, M<br>Set PH1 | |
| INTERRUPT | 4 | | Read P → R Bus<br>Store T Bus (CMF) → P | | Read P → R Bus<br>Read "1" → S Bus<br>Store T Bus (ADF) → P | | Read P → R Bus<br>Store T Bus (CMF) → P | | Reset M (6-15)<br>Store T Bus (0-5) → M<br>Set PH1 |

2000-50

Figure 3-17. Implementing Input/Output Instructions

the M-register are read into the T-register during the read memory cycle. Then, during T6 and T7 (assuming bit 15 of the T-register is now 0 for direct), all 16 bits of the T-register are transferred into the P- and M-registers in the usual way: read T-register onto S-bus, and store T-bus (with no arithmetic) in P- and M-registers. These registers now contain the effective address, so phase 1 is set, and the next machine phase will be a fetch phase, to read out the next instruction from that address. Note that if bit 15 of the T-register were again a one (for indirect) a jump would be made to still another location by repeating the process of these two paragraphs (3-87 and 3-88).

3-89. In summary, as illustrated in figure 3-18, an indirect jump occurs by the following register actions:

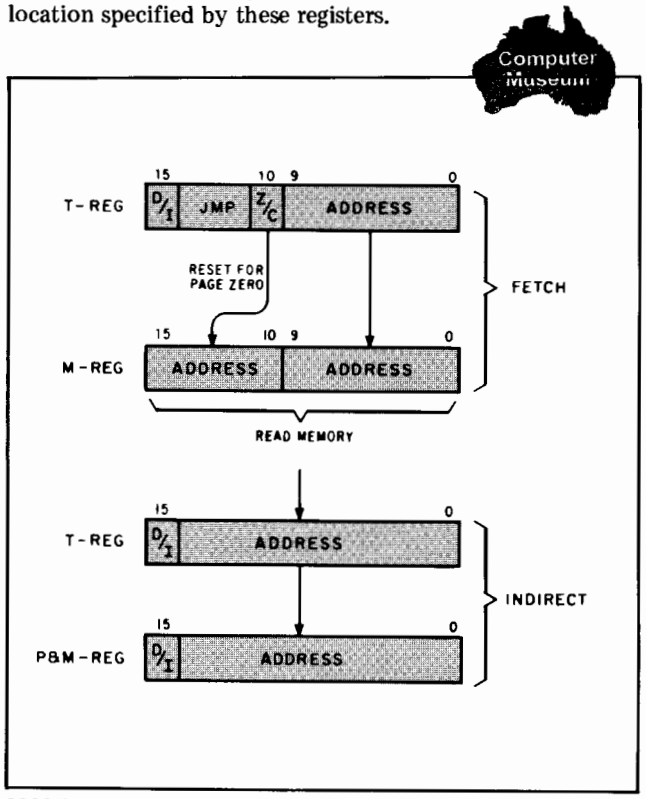
a. The word containing the jump instruction is read out of memory by a fetch phase into the T-register.

b. The address portion of the read-out word is transferred into the corresponding portion of the M-register.

c. The zero/current page bit of the read-out word tells the computer logic to clear (zero) or leave (current) the remaining bits (10 through 15) of the M-register.

d. Steps "b" and "c" now comprise the address of a location which is read out of memory into the T-register at the start of the indirect phase.

e. All bits of this new read-out word are transferred into the P- and M-registers. The computer is now at the location specified by these registers.




2000-31

Figure 3-18. Register Manipulations for Indirect Jump

3-90. AND. The fetch phase for the AND instruction is the same as for all other memory reference instructions listed below it in figure 3-15, with the exception that different functions will be set up at T2. This phase begins in the same way as for JMP. The T-register is cleared at time T0, the read memory cycle reads the instruction word into the T-register, the instruction register is cleared during T1, and T-register bits 10 through 15 (instruction code) are transferred into the instruction register at T2. At this time all necessary functions for this instruction are set up, to be used at the approriate times. During T6 and T7, T-register bits 0 through 9 (memory address portion of the instruction word) are transferred into the corresponding bits of the M-register (via the S- and T-buses). If the zero page indicator is present (bit 10 of the instruction register is a zero), a reset M(10-15) command clears bits 10 through 15 of the M-register.

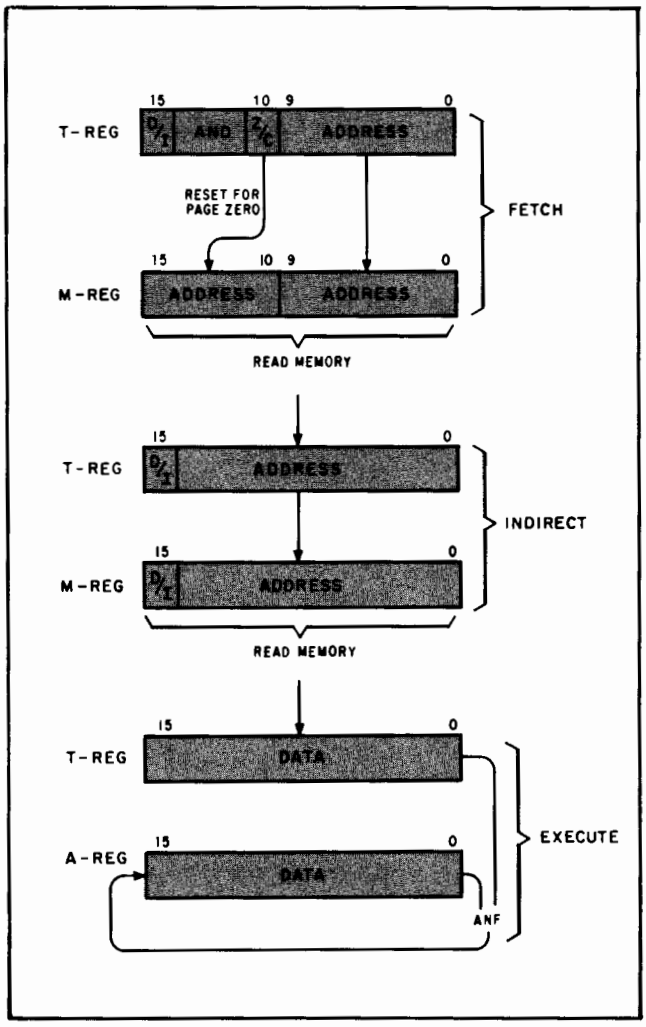
3-91. Unlike the JMP instruction, an execute or an indirect phase must follow the fetch phase of an AND instruction. (Execute never occurs for JMP; indirect is optional.) If bit 15 of the T-register is zero (for direct), phase 3 (execute) is set. Assume an indirect phase is required (bit 15 = 1). (If the direct condition exists, the action of the next paragraph would be skipped.)

3-92. The indirect phase begins by clearing the T-register during T1. Then a new word is read into the T-register from the memory location specified by the M-register (as set up in paragraph 3-90). This word is an address, not data, since indirect addressing really means: go to another location for the data. During T6 and T7 of the indirect phase, this address is transferred from the T-register to the M-register (all 16 bits). Note that it is possible for bit 15 to again specify indirect addressing; if so, phase 2 remains set and the procedure of this paragraph is repeated, and could be repeated several times, When bit 15 is a zero (direct), phase 3 is set.

3-93. The execute phase begins by clearing the T-register. The instruction register remains unchanged, since the various functions are still needed. This time, the read portion of the memory cycle reads data from memory into the T-register. During T3 and T4, this data is read onto the S-bus and the A-register contents are read onto the R-bus. The "and" function (ANF) previously set up by the instruction register, now combines the data on the two buses by "anding". (See table 2-1 for the arithmetic resulting from an "and" operation.) The result on the T-bus is then stored into the A-register.

3-94. To advance the computer to the next instruction, the P- and M-registers must be incremented by one. This is done during T6 and T7 of the execute phase. It is accomplished by reading the P-register onto the R-bus and a one onto the S-bus, then adding the two buses (add function: ADF) and storing the result into the P- and M-registers.

3-95. In summary, as illustrated in figure 3-19, an AND indirect instruction is executed by the following register actions:




2000-32

Figure 3-19. Register Manipulations for Indirect "And"

a. The word containing the AND instruction is read out of memory by a fetch phase into the T-register.

b. The address portion of the read-out word is transferred into the corresponding portion of the M-register.

c. The zero/current page bit of the read-out word tells the computer logic to clear (zero) or leave (current) the remaining bits of the M-register.

d. Steps "b" and "c" now comprise the address of a location which is read out of memory into the T-register at the start of the indirect phase.

e. All bits of this new read-out word are transferred into the M-register, thus addressing the location of the desired data.

f. At the start of the execute phase, the data thus addressed is read into the T-register from memory.

g. The contents of the T-register and A-register are "anded" together and deposited back into the A-register.

Note

For the remainder of memory reference instructions, the fetch and indirect phases are the same as described above for the AND instruction (paragraphs 3-90 through 3-92). The following paragraphs therefore describe only the execute phase for each instruction.

3-96. XOR. The execute phase of the XOR (exclusive or) instruction begins as usual by clearing the T-register just before the read portion of the memory cycle. The action occurring during T3 and T4 is shown in abbreviated form in figure 3-15, to be read as follows: the contents of the A-register are combined by an "exclusive or" function with the contents of the T-register, and stored back into the A-register. Actually this action consists of three steps as shown for the AND instruction. For XOR, these three steps are: 1) read T-register onto S-bus; 2) read A onto R-bus; 3) store T-bus (which carries the "exclusive or" combination of the S- and R-buses) into the A-register. The action during T6 and T7 is also abbreviated: add one to P, and store into P and M. The three steps which accomplish this are detailed for the AND instruction in figure 3-15. The last action is to reset the computer to the phase 1 (fetch) condition.

3-97. IOR. The execute phase of the IOR (inclusive or) instruction is the same as XOR described in the preceding paragraph, except that the "inclusive or" function is used in place of "exclusive or". The difference in arithmetic is shown in table 2-1 of the specifications section.

3-98. JSB. The principal operation of the execute phase for JSB (jump to subroutine) is to store the return address (program counter contents plus one) in the memory location being jumped to. This is done during T0 through T2. Since the only way into memory is through the T-register, the T-register must be loaded with the return address prior to the write portion of the memory cycle. Therefore the memory contents read out during the read portion of the memory cycle must be inhibited, and instead (during T1 and T2) the current contents of the P-register, plus one, is stored into the T-register. (Action: read P onto R-bus, read "1" onto S-bus, store with add function into T-register.) This information is then stored into memory during write. To complete the jump process, the contents of the M-register (which received the jump memory address during the fetch or indirect phase) must be transferred into the P-register. This is done during T3: Read M onto S-bus, store T-bus in P. As usual, to advance the computer to the location of the next instruction both P and M registers are incremented by one during T6 and T7, and the fetch phase condition is set.

3-99. ISZ. During the execute phase of the ISZ instruction (increment, skip if zero), the contents of the addressed memory cell must be altered and checked between the read and write portions of the memory cycle. These actions require more time than is normally available in this interval, so the write portion is delayed. Once the word read from memory is in the T-register (T3 and T4), it is incremented

by reading onto the S-bus, adding one in the arithmetic logic, and storing back into the T-register. If previously the word read out was all ones, the addition of another one causes a rollover to all zeros, and produces a signal (C16) which sets a carry flip-flop in the arithmetic logic. Then, at T5, the write portion of the memory cycle is permitted to begin, and two time periods (0.5 microsecond) are inserted at this time for writing the incremented value back into memory. During T6 and T7, the P-register is read onto the R-bus, and a one is read onto the S-bus. These are added together, and if the carry flip-flop is set, another one is added and the result is stored in the P- and M-registers. Thus, if the carry flip-flop was set, the P- and M-registers are incremented by two instead of one, skipping one memory location for the next fetch phase. (The carry flip-flop is automatically reset at the start of the next phase.)

3-100. ADA/B. If bit 11 of the instruction register indicates A (zero), the contents of the A-register are combined with the T-register contents by the add function (ADF), and stored into the A-register. Similar action involving the B-register occurs during this time (T3 through T4) if bit 11 of the instruction register is a one.

3-101. CPA/B. Depending on the status of bit 11 of the instruction register, either the A-register or the B-register is combined with the T-register contents by the "exclusive or" function. The result appears on the T-bus, but is not stored anywhere. Logic not shown in figure 3-1 tests the T-bus for a non-zero condition which, if it exists, sets the carry flip-flop. Then during T6 and T7 (as for ISZ), the P- and M-registers are incremented by either one (carry FF not set) or two (carry FF set).

3-102. LDA/B. During T3 and T4, the information read into the T-register by the read portion of the memory cycle is simply transferred to either the A- or B-register via the S- and T-buses.

3-103. STA/B. Like JSB, the STA/B instruction (store A or B) deposits new information into a memory cell, with no concern for the existing memory contents. The memory data read out during the read portion of the memory cycle is therefore inhibited while the A- or B-register contents are read and stored into the T-register (during T1 and T2). The write portion of the memory cycle deposits this information into memory.

3-104. REGISTER REFERENCE.

3-105. All register reference instructions, as shown by figure 3-16, are fully executed in only one phase (fetch). Actual execution is accomplished during time periods T3 through T5. Actions during the other time periods are similar to those previously described for memory reference instructions:

a. During time periods T0 through T2, the T-register and instruction register are cleared, and bits 10 through 15 of the instruction word read out of memory are transferred to the instruction register. Unlike memory reference, the instruction register does not set up functions, but rather it provides gating signals to identify the type (register reference) and group (shift-rotate, or alter-skip) of instructions. The remaining bits of the T-register are used to execute the individual instructions by setting up the appropriate functions. Figures 2-5 and 2-6 define which bits encode each instruction.

b. During time periods T6 and T7, the P-register is read onto the R-bus and a one is read onto the S-bus. If the carry flip-flop has been set by a skip condition during T3 through T5, another one is added, and the total (P-register incremented by one or two) is stored into the P- and M-registers. This advances the computer to the next instruction.

3-106. Paragraphs 3-107 through 3-132 detail the actions which execute all register reference instructions.

3-107. SHIFT-ROTATE INSTRUCTIONS.

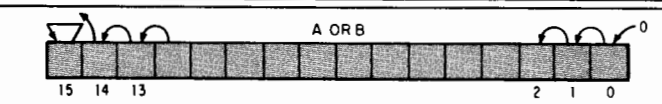
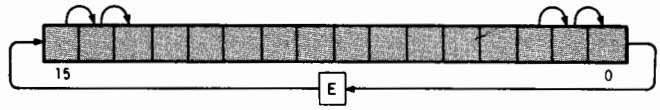
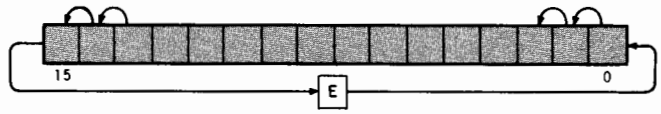
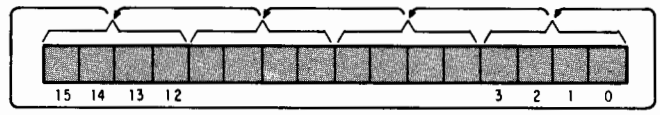
3-108. Figure 3-16 shows that shifts and rotates can be executed either during T3 or T5, or both. CLE (clear extend) or SLA/B (skip if least significant bit of A- or B-register is zero) can be executed only during T4. The shifts and rotates are executed simply by reading A- or B-register onto the R-bus, applying a shift function to shift some or all of the bits to a different position on the T-bus, then storing the T-bus back into the A- or B-register. Since the shift function is the key to understanding how shifts and rotates occur, the following instruction descriptions, through paragraph 3-116, concentrate on this aspect (CLE and SLA/B are described later in paragraphs 3-117 and 3-118). Table 3-1 is the main reference for these descriptions.

3-109. A/BLS. As shown by the table 3-1 diagram for A/BLS (A or B left shift), the desired end result is to have bits 0 through 13 shifted left one place, with bit 15 unchanged and a zero moved into bit 0. Assuming that bits 6 through 9 of the T-register dictate an A/BLS during T3, an SLM (shift left magnitude) signal at this time is "anded" with each of the 14 R-bus bits (0 through 13), with the output of each "and" gate appearing on the next higher T-bus line. The function listed in table 3-1 for this instruction (SLM RB(0-13)) is therefore to be read: shift left magnitude "anded" with R-bus bits 0 through 13. Bit 15 of the R-bus is routed directly out to bit 15 of the T-bus. Since nothing has been placed onto bit 0 of the T-bus, its state is "0", and therefore no deliberate action is necessary to ensure storing a "0" in bit 0 of the A- or B-register.

3-110. A/BRS. A shift right magnitude "anded" with R-bus bits 1 through 15 shifts these bits to bits 0 through 14 of the T-bus. Bit 0 of the R-bus is not recognized, and bit 15 (as well as moving onto bit 14 of the T-bus) also is routed directly to bit 15 of the T-bus.

3-111. RA/BL. To rotate A or B left, an SLM "anded" with T-bus bits 0 through 13, together with a shift left bit

Table 3-1. Shift Rotate Functions

| INSTRUCTION | FUNCTIONS | DIAGRAM |
|---|---|---|
| A/BLS | SLM · RB(0-13)<br>RB15 → TB15 |   |
| A/BRS | SRM · RB(1-15)<br>RB15 → TB15 |   |
| RA/BL | SLM · RB(0-13)<br>SL14 · RB14<br>RLL · TB15 |   |
| RA/BR | SRM · RB(1-15)<br>RRS · RB0 |   |
| A/BLR | SLM · RB(0-13) |   |
| ERA/B | SRM · RB(1-15)<br>E → TB15<br>RB0 → E |   |
| ELA/B | SLM · RB(0-13)<br>SL14 · RB14<br>E → TB0<br>RB15 → E |   |
| A/BLF | RL4 · RB(0-15) |   |

| | | | |
|---|---|---|---|
| SLM | Shift Left Magnitude | RB | R Bus |
| SRM | Shift Right Magnitude | TB | T Bus |
| RLL | Rotate Left to Least significant bit | SL | Shift Left |
| RRS | Rotate Right to Sign bit | RL | Rotate Left |

2000-33

14 to R-bus bit 14, move bits 0 through 14 to bit 1 through 15 of the T-bus. Rotating bit 15 of the R-bus around to bit 0 of the T-bus is accomplished by "anding" RLL (rotate left to least significant bit) with R-bus bit 15; the "and" gate outputs to T-bus bit 0.

3-112. RA/BR. A shift right magnitude "anded" with R-bus bits 1 through 15 shifts these bits to bits 0 through 14 of the T-bus. An RRS (rotate right to sign bit) "anded" with R-bus bit 0 rotates this bit to bit 15 of the T-bus.

3-113. A/BLR. A shift left magnitude with R-bus bits 0 through 13 shifts these bits to bits 1 through 14 of the T-bus. Bits 0 and 14 of the T-bus remain in the "0" state, since nothing is placed on these lines.

3-114. ERA/B. A shift right magnitude with R-bus bits 1 through 15 causes shift to T-bus bits 0 through 14. The content of the extend register is transferred into bit 15 of the T-bus. Then, during the latter half of T3 (or T5), bit 0 of the R-bus is transferred into the extend register.

3-115. ELA/B. A shift left magnitude "anded" with R-bus bits 0 through 13, and a shift left 14 with R-bus bit 14 shifts these bits to bits 1 through 15 of the T-bus. The extend content is transferred onto T-bus bit 0, and then bit 15 of the R-bus is transferred into the extend register.

3-116. A/BLF. A rotate left 4 "anded" with all bits of the R-bus shifts each bit four places to the left on the

T-bus. The four most significant bits are placed into the least significant bit positions.

3-117. CLE. During T4, if bit 5 of the T-register is a "1", a reset signal is generated which clears the extend register.

3-118. SLA/B. During T4, if bit 3 of the T-register is a "1" the A- or B-register is read onto the R-bus. (Bit 11 determines which register is read out.) If bit 0, the least significant bit, is a "0", the carry flip-flop is set. This will cause the P- and M-registers to be incremented by two (for a skip) during T6 and T7.

3-119. ALTER-SKIP INSTRUCTIONS.

3-120. Figure 3-16 individually lists all alter-skip instructions. The grouping into three time periods explains the grouping of columns in the selection table of figure 2-6. That is, during T3 one instruction involving the accumulators can be executed (clear, complement, or clear-complement), and two possible instructions involving the extend register can be executed (skip if zero, and clear or complement, or clear-complement). Incrementing of accumulators (INA/B) effectively occurs after tests for sign and least significant bits (SSA/B and SLA/B, at T4), but before the test for zero accumulator (SZA/B, at T5).

3-121. The alter instructions (clear, complement, and increment) use a store or direct transfer function. The skip instructions, however, simply read information onto the T-bus for testing; a store function is not required. If skip conditions are met, the carry flip-flop is set, causing the P- and M-registers to be incremented by two during T6 and T7.

3-122. CLA/B. To clear the A- or B-register, the read function is omitted. This means that both R- and S-buses are in the all-zero state. The "exclusive or" function, in combining zeros with zeros, can only produce zeros on the T-bus. Thus when the T-bus is stored into A or B, the result is all zeros.

3-123. CMA/B. To complement A or B, the register is read onto the R-bus, the complement function (CMF) reverses each bit before being released to the T-bus, and the T-bus is stored back into the A- or B-register.

3-124. CCA/B. The procedures of the two preceding paragraphs are combined to clear and complement an accumulator; i.e., with no read, R- and S-buses remain all-zero, and the complement function reverses this state to all ones on the T-bus. The T-bus is then stored into the A- or B-register.

3-125. SEZ. If bit 5 of the T-register is a one, the extend flip-flop and bit 0 of the T-register (reverse skip sense) are looked at by the computer logic, causing the carry flip-flop to be set if: a) both bits are zero, b) both bits are one. Although the next three instructions described below can alter the state of the extend flip-flop, the test is completed before the alteration.

3-126. CLE. If bits 6 and 7 of the T-register encode the clear E instruction, a reset signal is generated during the latter half of T3 to reset the extend flip-flop.

3-127. CME. If bits 6 and 7 of the T-register encode complement E, the state of the extend flip-flop is reversed during the latter half of T3.

3-128. CCE. If bits 6 and 7 of the T-register encode clear and complement E, the extend flip-flop is set during the latter half of T3.

3-129. SSA/B. If bit 4 of the T-register is a one, the A- or B-register is read onto the R-bus. Bit 15 of the R-bus (sign bit) and bit 0 of the T-register (reverse skip sense) are tested. The carry flip-flop will be set if both bits are zero (meaning: skip if sign bit is zero), or if both bits are one (meaning: skip if sign bit is not zero). This is accomplished during T4.

3-130. SLA/B. If bit 3 of the T-register is a one, the A- or B-register is read onto the R-bus. Bit 0 of the R-bus (least significant bit) and bit 0 of the T-register (reverse skip sense) are tested. The carry flip-flop will be set if both bits are zero (meaning: skip if least significant bit is zero), or if both bits are one (meaning: skip if least significant bit is not zero). This is accomplished during T4. The combination of SLA/B, SZA/B, and RSS is a special case; refer to the RSS description in paragraph 2-82.

2-131. INA/B. If bit 2 of the T-register is a one, the A- or B-register is read onto the R-bus, and a "one" is read onto the S-bus. These are combined by an add function (ADF) and stored back into the A- or B-register during the latter half of T5.

3-132. SZA/B. If bit 1 of the T-register is a one, the A- or B-register is read onto the R-bus and transmitted to the T-bus. All bits of the T-bus are applied to an "inclusive or" gate. The output of this gate and bit 0 of the T-register are tested. The carry flip-flop will be set if both TR0 and the gate output are zero (meaning: skip if accumulator is zero), or if both TR0 and the gate output are one (meaning: skip if accumulator is not zero).

3-133. INPUT/OUTPUT INSTRUCTIONS.

3-134. Like the register reference instructions, input/output instructions, as shown by figure 3-17, are fully executed in only one phase (fetch). The interrupt phase, shown at the bottom of figure 3-17, is not involved in the discussion at the end of this section (paragraph 3-150), since it is related to input/output operations as described under paragraph 2-113 of the specifications.

3-135. The following descriptions will concentrate on actions occurring during time periods T3, T4, and T5, since as can be seen from figure 3-17, the actions during other time periods are nearly identical from instruction to instruction. That is, the T-register is cleared during T0, the instruction register is cleared during T1, and the P- and

M-registers are incremented by one (or two, if a carry bit is present) during T6 and T7. The method of incrementing by one was described in paragraph 3-94, and the method for incrementing by two was described in paragraph 3-99. In all cases, bits 10 through 15 of the T-register are transferred to the instruction register during T2.

3-136. HLT. If bits 8, 7, 6 of the T-register encode the halt instruction, these bits cause the run flip-flop to be reset during the latter half of T7.

3-137. STF. During T3 a set flag signal is routed to all input/output interface cards, and will set the flag flip-flop of the card which is currently enabled by the select code (bits 0 through 5 of the T-register).

3-138. CLF. During T4 a clear flag signal is routed to all input/output interface cards, and will reset the flag flip-flop of the card which is currently enabled by the select code.

3-139. SFC. A skip if flag clear signal (SFC) is routed to the selected interface card beginning at T3. The interface card will return a skip flag signal (SKF) during T4 if its flag flip-flop is not set. This signal sets the carry flip-flop to cause a skip during T6 and T7.

3-140. SFS. A skip if flag set signal (SFS) is routed to the selected interface card beginning at T3. The interface card will return a skip flag signal (SKF) during T4 if its flag flip-flop is set. This signal sets the carry flip-flop to cause a skip during T6 and T7.

3-141. MIA/B. During T4 and T5 an IOI signal (I/O input control) transfers the input data from the interface buffer register to the S-bus. During the same time the A- or B-register is read onto the R-bus, and the R- and S-bus data is combined by the "inclusive or" function (IOF) and applied to the T-bus. The result (a merge, or "inclusive or") is stored back into the A- or B-register. If bit 9 of the T-register is a one, a clear flag signal (CLF) is routed to the flag flip-flop of the selected interface card, as described in paragraph 3-138.

3-142. LIA/B. The action for LIA/B (load input into A or B) is the same as described for MIA/B in the preceding paragraph, except that nothing is read onto the R-bus. The "inclusive or" function therefore transmits the R-bus unchanged to the T-bus for storing into the A- or B-register. As for MIA/B, bit 9 can clear the flag flip-flop.

3-143. OTA/B. During T4 and T5 the A- or B-register is read onto the R-bus, which in turn is transferred by an IOO signal (I/O output control) to the interface buffer register. As for MIA/B, bit 9 can clear the flag flip-flop.

3-144. STC. A set control signal is routed to all input/output interface cards, and during T4 will set the control flip-flop of the interface card which is currently enabled by the select code (bits 0 through 5 of the T-register).

3-145. CLC. A clear control signal is routed to all interface cards during T4, and will reset the control flip-flop of the interface card currently enabled by the select code.

3-146. STO. A set flag signal during T3, combined with the select code for the overflow flip-flop (01, octal), sets the overflow flip-flop.

3-147. CLO. A clear flag signal during T4, combined with the select code for the overflow flip-flop (01, octal), resets the overflow flip-flop.

3-148. SOC. During T3, a skip if flag clear signal (SFC), combined with the select code for overflow, tests the state of the overflow flip-flop. If this flip-flop is in the reset state, a skip flag signal (SKF) sets the carry flip-flop at T4, to cause a skip at T6 and T7.

3-149. SOS. During T3, a skip if flag set signal (SFS), combined with the select code for overflow, tests the state of the overflow flip-flop. If this flip-flop is in the set state, a skip flag signal (SKF) sets the carry flip-flop at T4, to cause a skip at T6 and T7.

3-150. INTERRUPT PHASE.

3-151. The actions occurring during the interrupt phase (phase 4) are shown at the bottom of figure 3-17. Two operations are accomplished during the interrupt phase:

a. The P-register is decremented. This is done so that any instruction which has not been fully executed at the time of interrupt will be repeated. On the other hand, if the instruction is fully executed (which means that the P-register has been advanced for the next instruction), it is still necessary to decrement. This is because the P-register is incremented for a second time following execution of the instruction contained in the interrupt location.

b. The "interrupt address" must be transferred into the M-register, and phase 1 is set. This causes the instruction contained in the interrupt location to be read out of memory for execution during the next machine phase. Note that the interrupt address is not placed into the P-register. While the instruction in the interrupt location is being executed, the P-register remains at the value one lower than the point at which interrupt occurred.

3-152. Decrementing the P-register is accomplished by complementing, incrementing, then complementing again. In simplified form, using only four binary digits for an example, this process is:

| | | |
|---|---|---|
| Original Value: | $0110_2$ | $(6_8)$ |
| Complement: | 1001 | |
| Increment: | 1010 | |
| Complement: | 0101 | $(5_8)$ |

3-153. During T1 and T2 of the interrupt phase (remember that there is no read/write memory cycle), the P-register is read onto the R-bus. The complement function

(CMF) reverses all bits before application to the T-bus, and then the T-bus is stored back into the P-register. During T3 and T4 the P-register is again read onto the R-bus. A one read onto the S-bus is combined with this by the add function (ADF) and the incremented result is stored back into the P-register. During T5, the P-register is read onto the R-bus for the third time, is complemented, applied to the T-bus, and stored back into the P-register.

3-154. The interrupt address is placed into the M-register during T7. Since no interrupt address is greater than $77_8$ (see table 2-2), M-register bits 6 through 15 are first reset. The interrupt address is read directly onto the T-bus from input/output control logic (see figure 3-1), and bits 0 through 5 are stored into the M-register. Setting the phase 1 condition completes the interrupt phase.

# SECTION IV

# BASIC OPERATION OF HP 2114B COMPUTER

## 4-1. INTRODUCTION.

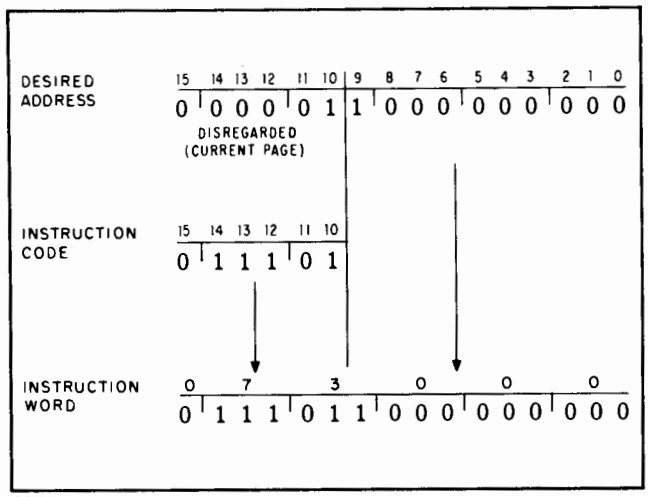
4-2. The purpose of this section is to relate the theoretical operations described in the preceding section to actual visible actions. Specific information is given for the user to gain familiarity with the panel controls, and to be able to perform basic operations on the computer, when necessary, without input/output devices or software aids. These purely manual operations are most commonly encountered in computer maintenance, and for loading, examining, and changing small sections of memory (e.g., loading the basic binary loader).

4-3. Obviously manual usage of the computer is not the intended mode of operation for practical applications. Therefore this section does not attempt to teach programming to the extent of practical problem solving. This aspect is the subject of training materials supplied with the user training course, which is provided by Hewlett-Packard. User training concentrates on the efficient use of software to solve problems. Instructions for usage of the computer via input/output devices are given in volumes three and four.

## 4-4. CODING.

4-5. This section assumes familiarity with binary and octal numbering systems, as outlined in the introduction to section III. Table A-4 (Consolidated Coding Table) in the appendix of this volume is used as a reference for instruction codes; if more detail is required, refer also to the information given under paragraph 2-53 (Instructions) in section II. As a reminder: a "one" is coded by a switch of the switch register being in the on state, and is indicated by the register light being on. A "zero" is coded by a switch in the off state, and is indicated by the register light being off.

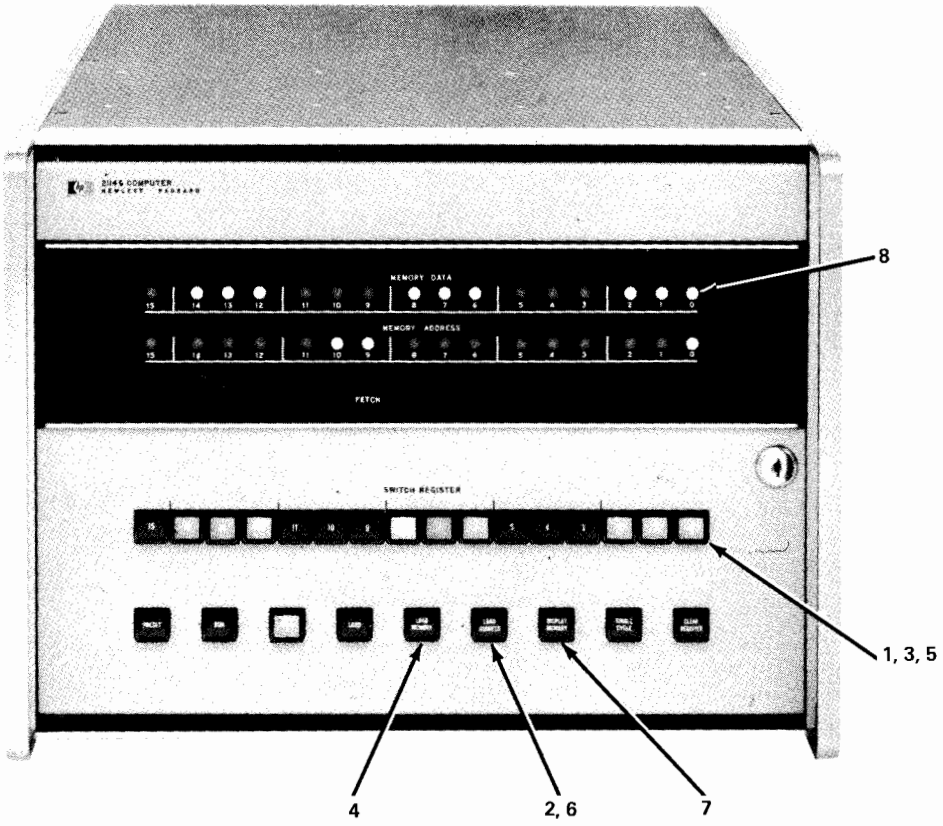
4-6. All numbers used for addresses or contents in this section are octal numbers unless otherwise specified. Notation of instruction codes in octal numbers is an operator convenience for loading and reading binary information. The meaning of the octal code can be understood only when it is broken down into its binary elements. For example, note the first instruction code to appear in this text, which occurs in paragraph 4-19 (also step 3 of figure 4-4). The instruction is STA 3000 (store A-register into memory location 003000; initial zeros of address assumed). The coded instruction word is 073000. Refer now to the consolidated coding table (table A-4 in the appendix) or to figure 2-4 in section II. Note that the code for STA consists of ones in bit positions 14, 13, and 12, and a zero in bit position 11. Since indirect addressing is not being used at this time, bit 15 is a zero. Bit 10 must be a one, since the program and all references will be on the same (current) memory page. (An elaboration of the page concept is given later under paragraph 4-47.) This accounts for bits 10 through 15. See figure 4-1. The remaining bits (0 through 9), which comprise the memory address, are simply the corresponding bits of the desired address. The desired address in this case is 003000. This breaks down in binary form as shown in the top row of figure 4-1. Note that all bits higher than bit 0 of the desired address are disregarded by the programmer when composing the instruction word. This is because these bits fall outside of the page-size limits. The M-register, which contains the page-designating bits, will hold the bits constant at execute time, as commanded by bit 10 of the instruction code.




2000-34

Figure 4-1. Coding a Memory Reference Instruction Word

4-7. It is evident that the octal digit 3 in the resultant instruction word 073000 is the result of three individual factors: bit 11 (a zero) specified the A-register, bit 10 (a one) specifies current page, and bit 9 (a one) is an address bit. This requirement of using bits having separate, individual meanings to compose an octal digit is frequently encountered. For example, suppose that it is desired to rotate the B-register left three places and clear the extend bit, all in one instruction. From the shift-rotate group definitions (paragraph 2-81), it is determined that a suitable method for a three-place rotation is to rotate the B-register left four places (BLF), then right one place (RBR). The resultant octal code for the instruction which will accomplish these actions (including the clearing of the extend bit) is 005763. The way this number was composed can be shown by breaking it down into its binary components, as follows:




STORE

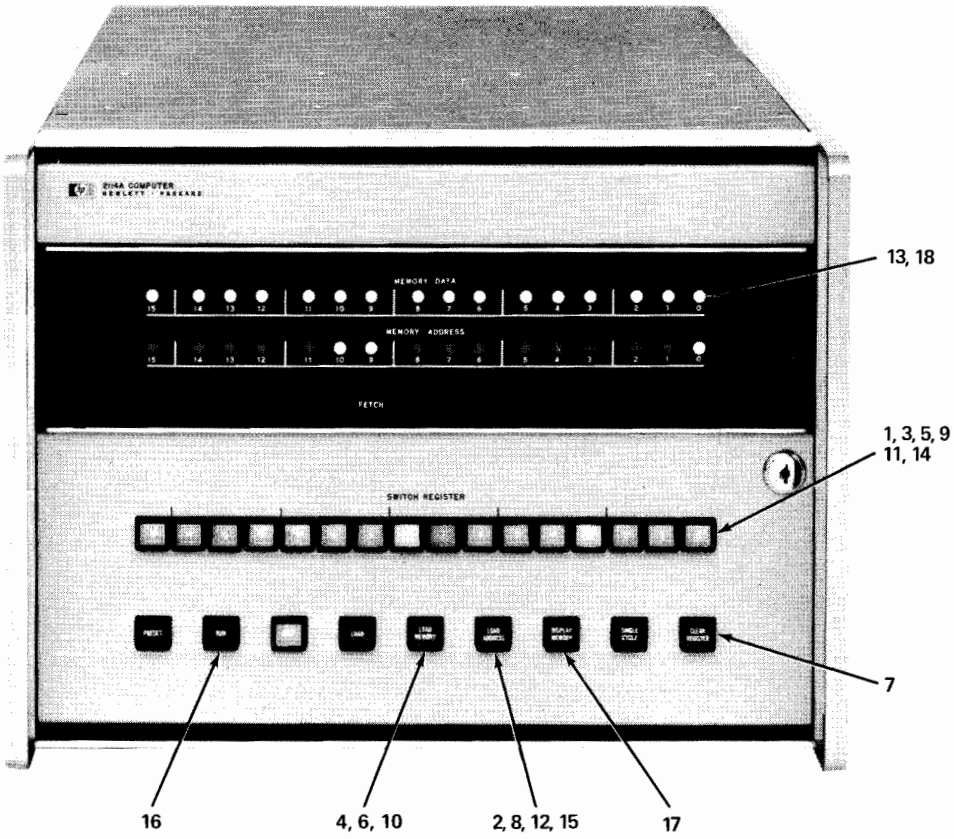
1. Set to 003000 (0 000 011 000 000 000).
2. Press LOAD ADDRESS.
3. Set to 070707 (0 111 000 111 000 111).
4. Press LOAD MEMORY. Photograph shows conditions existing at this time.

CHECK

5. Set to 003000.
6. Press LOAD ADDRESS.
7. Press DISPLAY MEMORY.
8. T-register indicates contents of memory location 003000: 070707 (no change).

2038-10

Figure 4-3. Storing Information Manually




**LOAD PROGRAM**

1. Set Switch Register to 002775.
2. Press LOAD ADDRESS.
3. Set to 073000 (STA 3000).
4. Press LOAD MEMORY.
5. Set to 102000 (HLT).
6. Press LOAD MEMORY.

**LOAD NEW INFORMATION**

7. Press CLEAR REGISTER.
8. Press LOAD ADDRESS.
9. Set S Register to 177777.
10. Press LOAD MEMORY.

**CHECK OLD INFORMATION**

11. Set to 003000.
12. Press LOAD ADDRESS.
13. Press DISPLAY MEMORY. T-Register indicates contents of memory location 003000: 070707.

**RUN PROGRAM**

14. Set S Register to 002775.
15. Press LOAD ADDRESS.
16. Press RUN.

**CHECK NEW INFORMATION**

17. Press DISPLAY MEMORY.
18. T-Register indicates new contents of memory location 003000: 177777. Photograph shows conditions existing at this time.

2038-11

Figure 4-4. Storing Information by Program

Table 4-1. Program Table

| ADDRESS | CONTENTS | | | | | | REMARKS |
|---|---|---|---|---|---|---|---|
| | INSTRUCTION (OR DATA) | MEMORY REFERENCE | D/I | A/B | Z/C | OCTAL CODE | |
| 002776 | STA | 3000 | | | C | 073000 | Get pattern from A, put in 3000. |
| 002777 | HLT | | | | | 102000 | Halt. |
| 003000 | | | | | | | Reserved for answer. |

4-28. ADDRESS. The address column of the program table states where in memory the program words (contents) are to be stored. The first listed address states where the program is to begin; this is termed the starting address. The starting address of the program shown in table 4-1 is 002776; the program stops at the location immediately following (002777). Although the program never advances to location 003000 (the location immediately following 002777), this address must be listed in the program table as a reminder that this memory location will be used by the program.

4-29. CONTENTS. As explained above (paragraph 4-24), the stored program can consist of three types of words: instructions, data, or even the address of another location. Therefore the contents of a location specified by an address may take various forms in the contents column. Most memory locations of a program will be instructions; the instruction mnemonic is listed under instruction (or data) in the table. If the content is not an instruction (usually a pure number representing data or an address), it will also appear under this heading, as shown in table 4-2. In the case of memory reference instructions, the address of the location affected by the instruction is listed under the memory reference heading. For example, the first instruction listed in table 4-1 is a command to store the A-register contents into location 003000. Location 003000 is the affected location (i.e., the memory reference). The D/I, A/B, and Z/C headings are also used only in the case of memory reference instructions. As a reminder to code a one-bit for I (indirect addressing), B (B-register), and C (current page), only these three indicators will be given in the tables; D (direct addresing), A (A-register), and Z (page zero), all coded by zero-bits, are otherwise assumed. The octal code column is used for the coded version of the desired contents. This column comprises the machine-language program, since this is the information which is loaded into the computer. As far as the computer is concerned, these numbers are the program. Note that no specific contents need be loaded for address 003000, since the STA 3000 instruction will destroy any information previously contained here.

4-30. REMARKS. A short explanation accompanying each assigned address of the program is helpful in communicating the intent of program details to other persons, and also can serve as a reminder to the original programmer when re-examining the program at a later time. Words used for the remarks column should be carefully chosen to be as concise and meaningful as possible. Understanding a given program can be difficult enough without adding confusion through vague documentation. For example, it would not be incorrect to say for the first instruction of table 4-1: store contents of A in location 3000. However this does not say any more than the instruction word itself says (STA 3000). The remark suggested in table 4-1 states what is expected to be in the A-register (a pattern), and raises the questions of what the pattern is, and how it happened to get into the A-register. This leads the operator to look for further documentation (in this case the text of this manual), which tells him how to preset the A-register. Additional words to indicate the need for presetting the A-register could be added, improving the message still further. Conversely, the halt in the next line requires no additional comment.

4-31. PROGRAM EXECUTION.

4-32. Table 4-2 lists the program used as an example in this discussion. The main purpose of the program is to show where and when the three types of program words (instruction, data, and address) occur. In the process of so doing, detailed actions for simple addition and indirect addressing will also be illustrated. The program adds 5 to 5, and puts the result (10 decimal, or 12 octal) into location 003000. Note that the middle three lines of the program are the same as the example given in table 4-1. The first two lines expand the program to accomplish the addition, and the last two lines are data and address words used by the program.

4-33. LOADING THE PROGRAM. The program is loaded into the computer manually, using the sample procedure given in steps 1 through 4 of figure 4-3. Steps 1 and 2 need be done only once for most of the program, since each LOAD MEMORY operation automatically increments the address in the P- and M-registers. Specifically, the procedure is:

a. Set the switch register to the starting address (002774), and press LOAD ADDRESS.

b. Set the switch register to the first word of the program (063001), and press LOAD MEMORY.

Table 4-2. Program to Show Instruction, Data, and Address Words

| ADDRESS | CONTENTS | | | | | | REMARKS |
|---|---|---|---|---|---|---|---|
| | INSTRUCTION (OR DATA) | MEMORY REFERENCE | D/I | A/B | Z/C | OCTAL CODE | |
| 002774 | LDA | 3001 | | | C | 063001 | Put augend in A. |
| 002775 | ADA | 3777 | I | | C | 143777 | Add the addend specified by 3777. |
| 002776 | STA | 3000 | | | C | 073000 | Put answer in 3000. |
| 002777 | HLT | | | | | 102000 | Halt. |
| 003000 | | | | | | - | Reserved for answer. |
| 003001 | 5 | | | | | 000005 | Data. |
| 003777 | 3001 | | | | | 003001 | Address of addend is 3001. |



c. Set the switch register to the next word of the program, press LOAD MEMORY, and repeat this step until the first six words have been loaded. For the fifth word (which requires no contents), it is convenient to simply press LOAD MEMORY with the HLT code still in the switch register. A halt instruction in this location does no harm.

d. For the seventh word, which is not in sequence with the other six, it is necessary to set the address (003777) into the switch register and press LOAD ADDRESS. Then set the switch register to the contents (003001), and press LOAD MEMORY.

4-34. RUNNING THE PROGRAM. Again set the switch register to the starting address (002774) and press LOAD ADDRESS. Now press RUN. Immediately the computer switches to the halt condition, having executed the problem and stored the answer in location 003000 in 16.0 microseconds. To verify that the computer has arrived at the right answer (000012), press the DISPLAY MEMORY pushbutton. The answer is in the T-register. This demonstrates how fast the computer operates, but does not show what operations it went through to arrive at its answer. Therefore the following paragraphs will rerun the program step by step in order to show these operations.

4-35. SINGLE CYCLE OPERATION. Table 4-3 shows the contents of the register following each operation of the SINGLE CYCLE pushbutton. The program will be executed in eight steps (i.e., eight machine phases). The following eight paragraphs describe each of these steps. The program is initially set up by setting the switch register to the starting address (002774) and pressing the LOAD ADDRESS pushbutton. The conditions now existing are shown in the top line of table 4-3: the P- and M-registers hold the starting address, and the remaining registers can be in any state. The FETCH phase indicator light on the panel is on, indicating that the first machine phase will be a fetch phase; this is an effect of the LOAD ADDRESS switch.

4-36. Press the SINGLE CYCLE pushbutton (first step). The conditions of the registers after the computer has completed this first phase are shown in the step 1 line of table 4-3. As an additional reference, refer back to figure 3-15 in the preceding section; the fetch phase actions for all memory reference instructions except JMP apply to this discussion. Note also the read/write memory cycle, which is what reads the contents of the addressed location (contents of 002774 is 063001) into the T-register. This is accomplished early in the fetch phase. The computer interprets any word read out of memory during a fetch phase as an instruction word. It is the programmer's responsibility to ensure that the computer does find an instruction in every location to which the P-register goes. This is ensured by properly filling out the program table; e.g., in table 4-2, the program (P-register) starts at 002774, and stops at 002777. Every one of these locations must have an instruction word as its contents. Later in the fetch phase (T6 and T7), the memory reference bits (0 through 9) of the T-register are transferred into bits 0 through 9 of the M-register. The remaining bits of the M-register are left unchanged (since there is no reference to page zero), thus completing the memory reference address in the M-register. In comparing the contents of the T- and M-registers in step 1 of table 4-3, be careful not to assume that the complete octal digits 3001 are transferred; the digit 3 (like the situation shown in figure 4-1 and explained in paragraphs 4-6 and 4-7) is a composite of three binary bits with different code meanings. Also occurring at the end of the fetch phase is the setting of the execute (phase 3) condition. The P- and A-registers are not yet affected.

4-37. Press the SINGLE CYCLE pushbutton again (step 2) to complete execution of the LDA 3001 instruction. Step 2 of table 4-3 shows register conditions existing after completion of the execute phase. This is the phase in which the computer gets the data requested by the memory reference, and does with it whatever is commanded by the instruction code. The read portion of the memory cycle reads the contents of the location addressed by the M-register (now at 003001) into the T-register. This information, read out of memory by the execute phase, is a data word. It is the programmer's responsibility to ensure

Table 4-3. Single Cycle Execution of a Program

| STEP | INSTRUCTION | T-REGISTER | P-REGISTER | M-REGISTER | A-REGISTER | B-REGISTER | PHASE |
|---|---|---|---|---|---|---|---|
| | | Any | 002774 | 002774 | Any | (Not used) | FETCH |
| 1 | LDA | 063001 | 002774 | 003001 | Any | | EXECUTE |
| 2 | | 000005 | 002775 | 002775 | 000005 | | FETCH |
| 3 | ADA, | 143777 | 002775 | 003777 | 000005 | | INDIRECT |
| 4 | I | 003001 | 002775 | 003001 | 000005 | | EXECUTE |
| 5 | | 000005 | 002776 | 002776 | 000012 | | FETCH |
| 6 | STA | 073000 | 002776 | 003000 | 000012 | | EXECUTE |
| 7 | | 000012 | 002777 | 002777 | 000012 | | FETCH |
| 8 | HLT | 102000 | 003000 | 003000 | 000012 | | FETCH |

that a data word (or an indirect address) is contained in all locations to which there is a memory reference (unless the location is to be used by the program for storage). As seen in table 4-2, there are three memory references; therefore, the table accounts for three addresses in addition to the four addresses assigned to the program instructions. One of these three is a storage location, one is data, and one is an indirect address. In this step, the information read out is the data 5. As shown in figure 3-15 (LDA/B), the data is transferred from the T-register to the A-register during the execute phase. Therefore the number 5 exists in both registers. At the end of this phase, the P- and M-registers are set to the address of the next instruction (002775), and the fetch condition is set (FETCH light on) for reading of the next instruction.

4-38. Press SINGLE CYCLE (step 3). This fetches the next instruction (143777) out of location 002775. The code 143777 means: add to whatever is in the A-register the contents of a memory location which can be found by going first to location 3777 for more information. This is what is implied by the symbolic form: ADA 3777, indirect. The indirect bit (bit 15 of the word now in the T-register) caused the setting of the indirect phase (INDIRECT light on), and the memory references bits (0 through 9) have been transferred into the M-register. The P- and A-registers remain as they were. The indirect phase is ready to begin.

4-39. Press SINGLE CYCLE (step 4). The computer always interprets information read out of memory during an indirect phase as an address word. This word (003001) is transferred to the M-register as the new memory reference for the current ADA instruction. Both T- and M-registers therefore now contain 003001. Since bit 15 of this word is a zero (direct address), the execute condition is set (EXECUTE light on). If this bit had been a one (indirect), the indirect condition would remain set, and a further memory reference would be obtained in the next step. However, with this example, the computer now knows that the addend data is located in 003001. It happens, in this example, that this is the same location from which the augend was taken; however, the address word could just as well refer to any location in memory.

4-40. Press SINGLE CYCLE (step 5). In the execute phase of the ADA instruction, the data in location 003001 is read out (the number 5), and is added to the existing contents of the A-register (which up until now also contained the number 5). The T-register therefore contains 5, and the A-register contains 12. As usual, the last operation for any instruction is to advance the P- and M-registers to the location of the next instruction (002776) and to set the fetch phase condition.

4-41. Press SINGLE CYCLE (step 6). The fetch phase of the STA 3000 instruction reads the instruction word (073000) out of location 002776, transfers the memory reference bits to the M-register and sets the execute phase condition.

4-42. Press SINGLE CYCLE (step 7). The execute phase puts the A-register contents (000012) into the memory via the T-register. Therefore both registers indicate this value. As usual, the P- and M-registers are advanced to the address of the next instruction (002777), and the fetch phase condition is set.

4-43. Press SINGLE CYCLE (step 8). The halt instruction is read out of memory, and the computer is in the same state as after the running of the program in paragraph 4-34. As before, the DISPLAY MEMORY

pushbutton can now be pressed to verify that location 003000 again has received the correct answer, 000012.

## 4-44. REFERENCING OTHER PAGES.

4-45. The procedures given in the preceding paragraphs used three memory reference instructions: LDA 3001; ADA 3777, I; and STA 3000. All of these instructions were stored in the second page of memory (refer to paragraph 2-23); i.e., they were stored in locations 2774, 2775, and 2776. In addition, the addresses to which these instructions referred (3001, 3777, 3000) were also located in the second page of memory. Thus each memory reference is a current page reference; i.e., no reference is made to an address which is outside the page in which the program itself is operating.

4-46. One program reference (ADA 3777,I) went to the page limit. This instruction could not have been ADA 4000,I, which refers to a location just one address higher. Location 4000 is not on the current page. On the other hand, ADA 1777 (with or without I) is possible, even though location 1777 also is not on the current page. The following paragraphs, through 4-62, deal with the special considerations for referencing memory pages other than the current page. The first step is to know what constitutes a page of memory.

4-47. CONCEPT OF THE MEMORY PAGE.

4-48. The necessity for dividing memory into pages arises in small computers, such as the HP 2114B, from the fundamental design concept of combining the instruction code and the memory reference into one computer word. This contributes to speed and efficiency in the computer, but also limits the number of bits available for the memory reference. As shown in figures 2-3 and 2-4 of the specifications section, bits 0 through 9 of the memory reference instructions are available for the memory reference address. Refer now to table 4-4 and note under the MEMORY REFERENCE BITS column that the possible range of numbers using these bits is (in octal) 0000 through 1777. To form addresses any higher than 1777 requires the addition of bits listed under the PAGE BITS column.

4-49. In the computer, a reference to memory is implemented by transferring bits 0 through 9 of the instruction word from the T-register to the M-register during the fetch phase (see figure 3-15). The remaining bits, during the fetch phase, can only stay at the value they used when addressing the current instruction location, before the fetch phase began. (Optionally, these bits can be reset to zero for a reference to page zero; this is relatively simple to accomplish internally.) Thus the programmer must know if these bits currently agree with the

Table 4-4. Memory Pages

| PAGE NO. | OCTAL ADDRESSES | COMPLETE BINARY ADDRESSES (M-REGISTER) | | | | | |
|---|---|---|---|---|---|---|---|
| | | PAGE BITS | | MEMORY REFERENCE BITS | | | |
| 0 | 00000 | (*) 0 0 0 | 0 0 | 0 | 0 0 0 | 0 0 0 | 0 0 0 |
| | 01777 | 0 0 0 | 0 0 | 1 | 1 1 1 | 1 1 1 | 1 1 1 |
| 1 | 02000 | 0 0 0 | 0 1 | 0 | 0 0 0 | 0 0 0 | 0 0 0 |
| | 03777 | 0 0 0 | 0 1 | 1 | 1 1 1 | 1 1 1 | 1 1 1 |
| 2 | 04000 | 0 0 0 | 1 0 | 0 | 0 0 0 | 0 0 0 | 0 0 0 |
| | 05777 | 0 0 0 | 1 0 | 1 | 1 1 1 | 1 1 1 | 1 1 1 |
| 3 | 06000 | 0 0 0 | 1 1 | 0 | 0 0 0 | 0 0 0 | 0 0 0 |
| | 07777 | 0 0 0 | 1 1 | 1 | 1 1 1 | 1 1 1 | 1 1 1 |
| 4 | 10000 | 0 0 1 | 0 0 | 0 | 0 0 0 | 0 0 0 | 0 0 0 |
| | 11777 | 0 0 1 | 0 0 | 1 | 1 1 1 | 1 1 1 | 1 1 1 |
| 5 | 12000 | 0 0 1 | 0 1 | 0 | 0 0 0 | 0 0 0 | 0 0 0 |
| | 13777 | 0 0 1 | 0 1 | 1 | 1 1 1 | 1 1 1 | 1 1 1 |
| 6 | 14000 | 0 0 1 | 1 0 | 0 | 0 0 0 | 0 0 0 | 0 0 0 |
| | 15777 | 0 0 1 | 1 0 | 1 | 1 1 1 | 1 1 1 | 1 1 1 |
| 7 | 16000 | 0 0 1 | 1 1 | 0 | 0 0 0 | 0 0 0 | 0 0 0 |
| | 17777 | 0 0 1 | 1 1 | 1 | 1 1 1 | 1 1 1 | 1 1 1 |

*Direct/Indirect bit does not form part of an address.

corresponding bits of the address he wishes to reference. To assist the programmer in this task, the convention is established of dividing memory into blocks called pages. Each block contains 2000 (octal) memory locations (or 1024 decimal). This block size is determined by the range of direct addressing capability (0000 through 1777), and each such block is assigned a page number.

4-50. Identification of page numbers is simplified by considering the 5 page bits (see table 4-4) as a separate binary word. Thus 00000 is page 0; 00001 is page 1; etc. Going back to the problem example in paragraph 4-46 (where it was stated that the ADA instruction in location 2775 could not directly reference location 4000), the situation can be analyzed as follows:

a. Current address is 000 010 111 111 101 (02775).
b. Page number (first five bits) is $00001_2$ (page 1).
c. Desired reference is 000 100 000 000 000 (04000).
d. Page number (first five bits) is $00010_2$ (page 2).

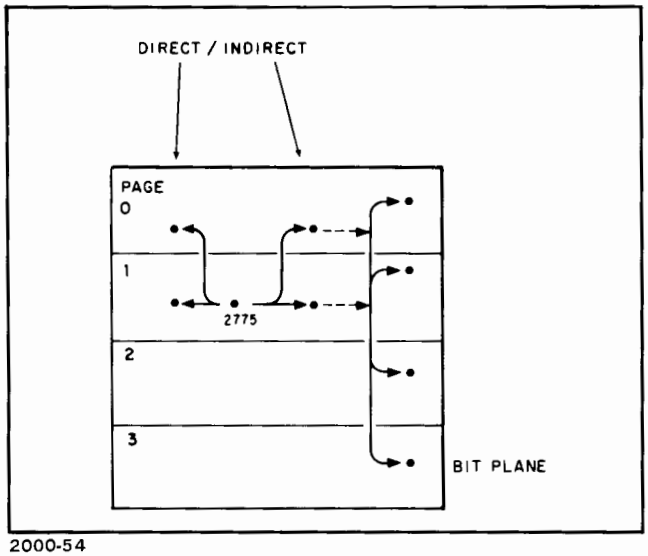
4-51. The desired reference requires a page change, or, in other words, bits 10 through 15 of the M-register must be altered in addition to the usual alteration of bits 0 through 9. To do so requires the use of a programming technique described under paragraph 4-55 (indirect references). A simpler technique of addressing another page (limited to page zero only) is discussed first in the following paragraph. Figure 4-5 shows individual memory cells which are addressable from a location on page 1. This source location may be thought of as location 2775, the same example as in the preceding discussions. Page 1 is the current page.

4-52. DIRECT REFERENCES.

4-53. The arrows going left from location 2775 in figure 4-5 show that, without using an indirect address, an instruction at this point can reference a location on either the current page or page 0. This doubles the range of possible references for instructions which are located on any page other than page 0. Bit 10 of the instruction word is reserved for distinguishing which page is referenced (zero for page 0, or one for current page). This distinction must always be considered when coding any memory reference instruction, or an erroneous reference may be made. The memory reference bits alone are not sufficient to identify a location. For example, ADA 5777 and ADA 1777 (assuming that the program is operating in page 2) have identical memory reference bits:

ADA 5777: 0 100 01(1 111 111 111)
ADA 1777: 0 100 00(1 111 111 111)

4-54. Only bit 10, the zero/current indicator, can make the distinction. The C in the coding table is a reminder that bit 10 must be coded a one when referencing current page. Otherwise it must be a zero for all memory reference instructions. Remember that bit 10 of the



2000-54

Figure 4-5. Direct and Indirect References to Other Pages

instruction word is not an address bit. Its function is to control bits 10 through 15 of the M-register: to either reset these bits to zero, or leave them alone. This provides an easy, direct access to information on page 0 from any other page, thus making page 0 useful for storage of data. Programs are generally stored in other pages (as the examples in this section do) in order to reserve page 0 for information which may be referred to frequently.

4-55. INDIRECT REFERENCES.

4-56. The arrows going right from location 2775 in figure 4-5 show that, by using an indirect address in the first referenced location, any location in memory can then be accessed. As in the preceding paragraph, the initial reference (contained in the instruction word), can refer to a location on either the current page or page 0. Broken lines in figure 4-5 indicate this optional choice. Either way, the initial reference is simply an intermediate step to the final desired reference. Obviously an added machine operation (indirect phase) is required, as well as the added memory location. The means of telling the computer that this additional step is desired is to code a one in bit 15 of the instruction word. An I in the coding table is a reminder to do this.

4-57. PROGRAM EXAMPLE.

4-58. Table 4-5 lists a program illustrating both a direct reference to page 0 and an indirect reference to page 2. As before, the program itself operates approximately in the middle of page 1. This program differs from that of table 4-2 in that the data, instead of being stored on the current page (location 3001), now appears in two different locations: location 1001 on page 0, and location 4000 on page 2. Figure 4-6 shows in simplified form the referencing accomplished by this program.

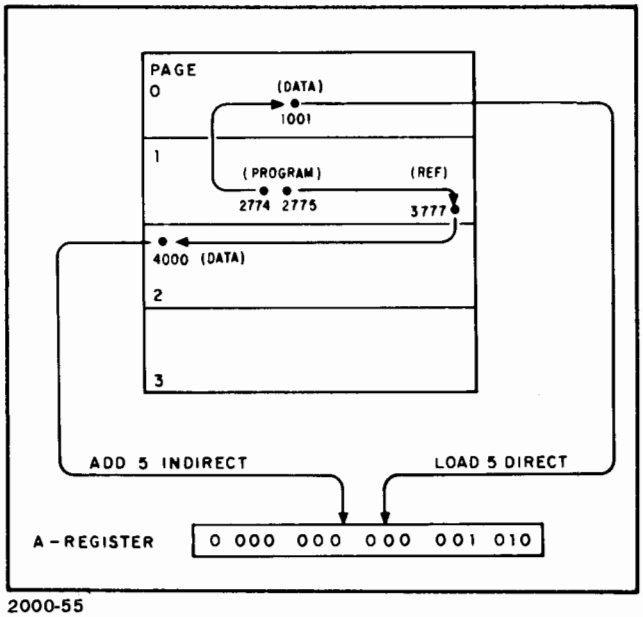
Table 4-5. Program for Interpage Referencing

| ADDRESS | CONTENTS | | | | | | REMARKS |
|---|---|---|---|---|---|---|---|
| | INSTRUCTION (OR DATA) | MEMORY REFERENCE | D/I | A/B | Z/C | OCTAL CODE | |
| 002774 | LDA | 1001 | | | | 061001 | Get augend from page Zero, put in A. |
| 002775 | ADA | 3777 | I | | C | 143777 | Add the addend specified by 3777. |
| 002776 | STA | 3000 | | | C | 073000 | Put answer in 3000. |
| 002777 | HLT | | | | | 102000 | Halt. |
| 003000 | | | | | | - | Reserved for answer. |
| 003777 | 4000 | | | | | 004000 | Address of addend is 4000. |
| 004000 | 5 | | | | | 000005 | Data (on Page 2). |
| 001001 | 5 | | | | | 000005 | Data (on Page 0). |

4-59. LOADING THE PROGRAM. Unless memory has been disturbed, the program of table 4-5 can be loaded by making a few changes to the existing conditions of the computer on completion of the preceding procedures. (The reader, at this point in the text, should be able to load a complete program, given octal addresses and octal-coded contents; refer back to paragraph 4-33 if necessary.) Changes required are:

a. Load location 002774 with contents 061001.
b. Load location 003777 with contents 004000.
c. Load location 004000 with contents 000005.
d. Load location 001001 with contents 000005.

4-60. DIRECT REFERENCE. Set the switch register to the starting address (002774), and press LOAD ADDRESS. Remembering that only bits 0 through 9 of the word about to be read out of memory are transferred to the M-register, watch bit 10 of the M-register and press SINGLE CYCLE once. Bit 10, a page bit, has changed from a one to a zero, thus changing pages. This situation is shown in figure 4-6, where the instruction word in location 2774 is causing location 1001 to be addressed. The contents of location 1001 is known to be 5; this will be loaded into the A-register in the next (execute phase). Again watch bit 10 of the M-register and press SINGLE CYCLE. The page indication returns to page 1 to address the next instruction (in 2775), and the data (octal 5) is in the A-register. Referring to figure 4-6, an instruction on page 1 has commanded data from page 0 (by direct reference) to be put into the A-register.




2000-55

Figure 4-6. Examples of Interpage Referencing

4-61. INDIRECT REFERENCE. Previously, in paragraphs 4-46 and 4-50, it was pointed out that a direct reference from location 2775 to 4000 is not possible. These two paragraphs describe the indirect method for making this reference. Briefly, the method is to make an initial reference to a location on the current page, pick up a 15-bit address there, and use that address to reference location 4000 (refer to figure 4-6). Although the initial reference could be anywhere on the current page or page 0, location 3777 (which is immediately adjacent to location 4000) has been chosen to emphasize the concept of page boundaries.

4-62. Watching bits 11 and 10 of the M-register, press SINGLE CYCLE. These bits remain $01_2$ (page 1) for the initial reference to location 3777 on the current page. Note that the computer has acknowledged the fact that indirect addressing is desired, since the INDIRECT light is on; this condition was specified by a one in bit 15 of the instruction word (now visible in the T-register). Again watching bits 11 and 10 of the M-register, press SINGLE CYCLE. These bits change to $10_2$ (page 2) for the indirect reference to location 4000. Since bit 15 of the T-register is now a zero (not indirect), the EXECUTE phase condition is indicated. This means that the next phase will execute the instruction, and the M-register will return to page 1 for the next instruction. Watching bits 11 and 10 of the M-register, press SINGLE CYCLE. These bits return to $01_2$ to address location 2776. The remaining actions are the same as in table 4-3,

steps 6, 7, and 8. Press SINGLE CYCLE three more times to complete the program.

## 4-63. JUMPS.

4-64. In all previous examples, although random references to various points in memory were made, the program itself (i.e., the list of instruction words) was located in a few consecutive locations in page 1. This strict sequential operation would be severely limiting for practical applications. Therefore provision must be made for the program to move freely throughout available memory. The jump instructions (JMP and JSB) provide this capability.

4-65. The essential difference between these two instructions is that the JMP (jump) instruction unconditionally suspends operation at the currently used area of memory and continues operation in a new area, whereas JSB (jump to subroutine) provides a means of remembering the location where the jump command was given, thus enabling a return to that point at some later time. Table 4-6 illustrates both kinds of jumps by treating the program previously developed as a subroutine (to add 5 + 5), and adding a few preliminary instructions. These preliminary instructions represent the main program; for simplicity of illustration, several NOP (no operation) instructions are inserted to represent a more lengthy sequence of working instructions.

4-66. The special considerations for referencing other pages, as covered in the preceding discussion (paragraphs 4-44 through 4-62), apply to the jump instructions. This means that the program can jump directly to any location on either current page or page zero, or indirectly to any location in memory. The program example in table 4-6 illustrates both a direct JMP and an indirect JMP, but only a direct JSB. An indirect JSB occurs in the same way as does the indirect JMP.

4-67. LOADING THE PROGRAM. If memory remains undisturbed from preceding procedures, the new program can be loaded simply by loading the octal code contents into the corresponding address for those items not shaded in table 4-6. Otherwise it is necessary to load all 15 addresses listed in the table. Note that LOAD ADDRESS must be used three times, since three separate areas of memory are being loaded.

4-68. THE JMP INSTRUCTION. Set the switch register to the starting address (002100) and press LOAD ADDRESS. Assume that a working program has been running sequentially up to this point (i.e., the P-register increments by one on completion of each instruction). For example, we may monitor the contents of the P-register by observing the contents of the M-register at the end of each execute phase. At this time the P-register is incremented and transferred to the M-register. By pressing the SINGLE CYCLE button the first NOP instruction is executed, and the P-register (also the M-register) advances from 002100 to 002101. In location 2101 is the instruction to jump to location 2200. Since a direct jump is a one-phase instruction, the jump will be completed in the next operation. When the SINGLE CYCLE button is pressed the P-register (also the M-register) does not increment by one, but rather jumps from 002101 to 002200. If the intervening instructions had contained instructions, those instructions would be omitted from the sequence of this program. Press SINGLE CYCLE two more times and note that at the end of the execute phase, the P-register (also the M-register) increments normally from the new operating point of 002200.

Table 4-6. Examples of Program Jumps

| ADDRESS | CONTENTS | | | | | | REMARKS |
|---|---|---|---|---|---|---|---|
| | INSTRUCTION (OR DATA) | MEMORY REFERENCE | D/I | A/B | Z/C | OCTAL CODE | |
| 002100 | NOP | | | | | 000000 | Program starts here (no operation). |
| 002101 | JMP | 2200 | | | C | 026200 | Jump to 2200. |
| 002200 | NOP | | | | | 000000 | No operation. |
| 002201 | NOP | | | | | 000000 | No operation. |
| 002202 | JSB | 2773 | | | C | 016773 | Jump to 5 + 5 subroutine at 2773. |
| 002203 | HLT | | | | | 102000 | Halt. |
| 002773 | | | | | | - | Reserved for return address. |
| 002774 | LDA | 1001 | | | | 061001 | Get augend from page Zero, put in A. |
| 002775 | ADA | 3777 | I | | C | 143777 | Add the addend specified by 3777. |
| 002776 | STA | 3000 | | | C | 073000 | Put answer in 3000. |
| 002777 | JMP | 2773 | I | | C | 126773 | Return to main program via 2773. |
| 003000 | | | | | | - | Reserved for answer. |
| 003777 | 4000 | | | | | 004000 | Address of addend is 4000. |
| 004000 | 5 | | | | | 000005 | Data (on Page 2). |
| 001001 | 5 | | | | | 000005 | Data (on Page 0). |

4-69. THE JSB INSTRUCTION. The P-register is now at the location (2202) which contains the instruction to jump to the subroutine which begins at location 2773. This subroutine, as the remarks column states, is a procedure to add 5 plus 5. It is desired, on completion of the subroutine, to return to the main program at the succeeding location (2203). It happens that the HLT instruction is located in 2203, but a program-continuing instruction could as well be stored there, and the program (P-register) would advance as usual to 2204, 2205, etc.

4-70. The JSB instruction, unlike JMP, requires two phases. The first phase (fetch) only references the location being jumped to; i.e., the P-register does not change in this phase. Watch the M-register and press SINGLE CYCLE, noting that location 2773 is referenced. The P-register will still contain the location (2202) where the jump command was given. The next phase will store the return address into the referenced location, and will complete the jump. When SINGLE CYCLE is pressed both the P- and M-registers will contain the address of the first instruction of the subroutine location 2774. Note also that the T-register holds the number 2203, the return address, which was stored into location 2773 during the phase just completed. This value is one higher than the location jumped from since obviously a return to location 2202 would send the program right back into the subroutine, and it would loop continuously without ever reaching 2203.

4-71. Now press the SINGLE CYCLE pushbutton seven more times. This executes the three instructions of the subroutine, which are identical to the instructions of the previous program (table 4-5). The content of location 2777, however, is now an indirect jump via location 2773. Location 2773, remember, contains the return address. Watch the M-register and press SINGLE CYCLE; this references location 2773. Since the next phase will be an indirect phase (INDIRECT light is on), the content of the referenced location will be interpreted as an address. The indirect phase will complete the jump to that address. Again press SINGLE CYCLE. The M-register (also the P-register) now addresses location 2203 of the main program, completing the jump out of the subroutine. Pressing the SINGLE CYCLE button will execute the HLT instruction contained in location 2203.

4-72. The preceding three paragraphs show how subroutines are accessed. By definition, a subroutine is a sequence of instructions designed to perform a single task, with provisions included to allow entry from any point in a program and return to the same point. The contents of locations 2773 through 2777 comprise a typical subroutine. The single task is an addition, and the entry and return requirements is guaranteed by storing the return address in location 2773 (a function of the JSB instruction) and by including an indirect jump via this location at the end of the subroutine (programmer's responsibility).

## 4-73. INTRODUCTION TO PROGRAM DEVELOPMENT.

4-74. The program examples given in the preceding discussions have been simple enough that no explanations were offered to explain how the programs were derived. The main object has been to demonstrate the register manipulations which occur during the running of the program. Refer ahead to the next program example in table 4-8, and note that 12 lines have been added to the previous 15, nearly doubling the length of the program. Readers without previous programming experience may, at this point, wish to know just how this sequence of instructions was developed. For example, how was it known in advance that the new starting address of the program would be 2166?

4-75. The answer is that preliminary development in rough form preceded the assigning of actual addresses. Temporary labels were used in place of final addresses. This introduces the concept of symbolic programming, which later becomes the exclusive means of program writing when software is involved. For such purposes, however, specific rules governing the use of labels apply, which are beyond the scope of this volume. This volume therefore uses a symbolic notation (lower case letters) unique to these discussions, with the implication that such labels are temporary assignments for rough work only. The appearance of lower case letters in a written program provides an immediate and obvious indication that the program is not completely developed.

4-76. The following description of looping and counting includes detailed information on the development of the program example. Before going into details of the program, however, it is first necessary to decide on general techniques, based on the problem to be solved. Suppose that the problem to be solved is:

$$[5 + 3(2)] + 5 = 16_{10}$$

4-77. The previously developed program showed how to use a subroutine to add two numbers, both of which happened to be 5. For convenience, the same subroutine can be used by letting one of the numbers be 5, and the other can be the result of the 5 + 3(2) calculation. Now it is only necessary, at some time before going into the subroutine, to perform the 5 + 3(2) calculation and store the result in an easily referenced location. It is the object of the following paragraphs to show how to do this calculation with a simple loop. Therefore the general techniques decided upon are: use a loop to calculate (5 + 3(2)), and use the previously established subroutine to add the result to the number 5.

## 4-78. LOOPING AND COUNTING.

4-79. THE PROGRAM LOOP.

4-80. To save core space (and, incidentally, to ease the burden on the programmer), it is frequently convenient to use a program loop when a sequence of instructions within

a program is to be repeated several times, with little or no change on each repeat. As in the present example, suppose it is desired to add 2+2+2 etc., any number of times to the number 5. To accomplish this, it would be possible to put the number 5 into location z, 2 into location y, and then add repeatedly:

```
a.  LDA  z
b.  ADA  y
c.  ADA  y
d.  ADA  y, etc.
z.  5
y.  2
```

4-81. By simply jumping back to the first add instruction immediately after it has been once completed, an endless program loop is created, accomplishing the same effect:

```
    a.  LDA  z
┌─→ b.  ADA  y
└── c.  JMP  b
    z.  5
    y.  2
```

4-82. The program starts at location a, which loads the contents of z (the number 5) into the A-register, then advances to location b, which adds the number 2 to the existing contents of the A-register (i.e., 5+2). Location c contains the instruction to jump back to location b, and thus add 2 again to the existing contents of the A-register (i.e., 5+2+2). This is the essential concept of the program loop. Obviously this simple sequence is not practical as it stands, since the loop will repeat endlessly. Some means must be provided for getting out of the loop after it has been repeated a desired number of times. This necessitates an instruction sequence to count each loop as it occurs, and then to exit from the loop when the desired count has been reached. The required sequence is next discussed.

4-83. COUNTING TO A LIMIT.

4-84. The ISZ instruction (increment, and skip if zero) is commonly used for counting to a preset limit, since its special features include both the counting (incrementing) and exit (skip) capabilities in one instruction word. A location in memory can be reserved for use as a counter; each time this location is referenced by the ISZ instruction, it is incremented by one (in the positive direction). If the counter location is initially set to a negative value, it will increment toward zero each time it is referenced. In the present example, if the counter is set to -3 before the loop is entered, the counter will go to zero on the third pass through the loop. This is the condition which causes the program to skip the next instruction. If the skipped instruction is the JMP instruction which causes the loop to repeat, the skip provides the means of getting out of the loop (after the third pass). This gives the following sequence:

```
    a.  LDA  z
┌─→ b.  ADA  y
│ ┌-- d.  ISZ  x
└─┼── c.  JMP  b
  └─→ e.  STA  w
    z.  5
    y.  2
    x.  -3
    w.  reserved for subtotal
```

4-85. Note that is has been necessary to insert a new location (labeled d) between locations b and c. Remember that the lower case letters are labels only; they need not be in alphabetic sequence. The instruction sequence here is a,b,d,c,e. The STA instruction in location e has been added to define where the program continues on exit from the loop. Also it has been necessary to add location x for the counter (preset to -3), and to add location w to store the result of the calculation. Storage of the result (which is obtained in the A-register) is necessary since the A-register will be used for other purposes in the program.

4-86. The program begins by loading 5 into the A-register, then advances to location b to add 2. Next, the ISZ instruction increments counter location x to -2. Then the JMP instruction causes a return to location b, where again 2 is added to the A-register. ISZ increments counter location x to -1. The JMP instruction causes a second return to location b, where 2 is added for the third (final) time to the A-register. ISZ increments counter location x to 0, and the program skips the JMP instruction and goes instead to location e. Here, the contents of the A-register are stored into location w, and the program continues to whatever instruction is next.

4-87. TALLYING.

4-88. Occasions arise in which it is desired simply to count, or produce a tally of the number of times a particular event occurs. This does not involve a loop or a skip, but again the incrementing feature of the ISZ instruction can be used. For example, suppose it is desired to know (or verify) how many passes through a loop are actually executed. In the present simple example of a program loop, the purpose of the tally would be to count how many times the number 2 is added to the number 5 (loaded into the A-register before the loop begins). Therefore an ISZ instruction, located ahead of the ADA instruction, can be used to increment a reserved tally location each time ADA is about to occur. (In this simple example, the tally could be placed either before or after the addition. In more complex programs, a definite placement may be dictated by the structure of the program). The tally, in this case should be 3 after the program has been run, indicating that three passes through the loop were made. Since the skip if zero feature of the ISZ instruction is not used, a NOP (no operation) instruction could follow ISZ, so that if the total should happen to exceed +32767 (and thus rolls over to zero), the resulting skip will not affect the operation of the program. The program loop now consists of the following sequence:

```
      a.  LDA  z
  ┌──►f.  ISZ  v
  │   g.  NOP
  │   b.  ADA  y
  │ ┌─d.  ISZ  x
  └─┼─c.  JMP  f  (note new reference)
    └►e.  STA  w
      z.  5
      y.  2
      x.  -3 (Counter)
      w.  reserved for subtotal
      v.  0  (Tally)
```

4-89. INITIALIZATION.

4-90. The need for initialization frequently occurs in programming, and is not exclusively associated with counting and tallying. It is introduced here as a typical example of the principle. Initialization enables a program to be repeated any number of times, by presetting to starting values all locations which must be in a specific state at the start of a program but are in a different state at the end of the program. This applies particularly to counters and tally locations. In the above examples, the counter starts at -3 and ends at 0, while the tally starts at 0 and ends at 3. To permit the program to be run a second time, the counter must be set back to -3 and the tally must be set back to 0. This is generally done at the start of a program; hence the term initialization.

4-91. Creating the two's complement form of a negative number can also be accomplished easily in the initialization, by using the combined register reference instructions CMA and INA (complement and increment the A-register). It is then necessary only to provide positive numbers for constants. Thus the complete initialization for both the counter and the tally would consist of five instructions:

```
aa. LDA  u      }
ab. CMA, INA    }  Set counter to -3
ac. STA  x      }
ad. CLA         }
ae. STA  v      }  Set tally to 0
u   3
```

4-92. Location u has been added to contain the positive number 3. The first instruction of the program puts this number into the A-register. The next instruction, in location ab, converts this number to -3. Then the result is stored in the location (x) previously established for the counter (paragraph 4-84). Location ad clears the A-register (all zeros), and this value of 0 is put into the location (v) previously established for the tally (paragraph 4-88).

4-93. COMPLETE PROGRAM.

4-94. Putting together all parts of the symbolic program developed in paragraphs 4-78 through 4-92, and then combining them with the previously developed subroutine, the partially developed listing given in table 4-7 is obtained. Note that two of the locations assigned symbolic addresses (z and w) already have actual addresses assigned: 3777, which references the addend requested by the subroutine, and 1001, which contains the augend (formerly the fixed number 5, now the subtotal produced by the loop). Looking under the memory reference column, it is seen that four other references (u,x,v,y) require an assignment in the address column. These are symbolically listed at the end of the program as a reminder to assign specific addresses for these references. There can be no unassigned references.

4-95. Now it is simply a matter of assigning actual addresses for the instructions by working backward from the first fixed address (2202), thus arriving at 2166 for the starting address. For ease of reference, the locations reserved for counters and constants are assigned locations on page 0, starting at the first fixed address, 1001. The resulting assignments for the fully developed program are shown in table 4-8.

4-96. A significant change in the remarks column has been introduced in the transposition from table 4-7 to table 4-8. In the former table it is necessary to read the remark for every instruction in order to understand the intended operation. Table 4-8 simplifies the reading by letting one remark apply to a group of instructions, assuming that the reader already understands such fundamentals as initialization, counting, looping, and subroutine entry and exit.

4-97. LOADING THE PROGRAM. If memory remains undisturbed from preceding procedures, the program of table 4-8 can be loaded simply by loading the octal code contents into the corresponding address for those items not shaded in the table. Otherwise it is necessary to load all 27 locations in order to run the program.

4-98. RUNNING THE PROGRAM. As before, it is possible to step through the program one phase at a time, by loading the new starting address and pressing SINGLE CYCLE for each phase. For a program of this length, 48 operations of the SINGLE CYCLE pushbutton are necessary to step through the entire program. If it is desired to examine in detail only the new portions of the program (initialization, looping, and counting), the instructions preceding the JSB instruction should be stepped through (36 operations of SINGLE CYCLE) and then press RUN to let the computer execute the remainder automatically. However, the program includes several locations which can be checked, after the program has been run, to verify that the program actually was executed in the manner prescribed by the written program. Simply load the starting address (002166), press RUN, and check the results as in the following paragraph.

4-99. First, load the answer address (003000) and press DISPLAY MEMORY. Since the probelm was stated to be 5 + 3(2) + 5, the answer, obviously, must be 16 (decimal) or 20, octal. That is, bit 4 of the T-register must be on, and all others off. To verify that the required three passes through the loop were completed, three locations can be checked: the subtotal, the loop counter, and the tally. Load

Table 4-7. Preliminary Program Development

| ADDRESS | CONTENTS | | | | | | REMARKS |
|---|---|---|---|---|---|---|---|
| | INSTRUCTION (OR DATA) | MEMORY REFERENCE | D/I | A/B | Z/C | OCTAL CODE | |
| aa | LDA | u | | | C | | Start. Put "3" in A. |
| ab | CMA, INA | | | | | | Convert to -3. |
| ac | STA | x | | | C | | Put -3 in Loop Counter. |
| ad | CLA | | | | | | Zero the A-Register. |
| ae | STA | v | | | C | | Put 0 in Tally. |
| a | LDA | 3777 (z) | I | | C | | Put "5" in A. |
| f | ISZ | v | | | C | | Add 1 to Tally. |
| g | NOP | | | | | | No Operation (void skip). |
| b | ADA | y | | | C | | Add 2 to value in A. |
| d | ISZ | x | | | C | | Add 1 to Loop Counter. Exit if count 0. |
| c | JMP | f | | | C | | Repeat Loop. |
| e | STA | 1001 (w) | | | | | Store subtotal in w on exit from loop. |
| 002202 | JSB | 2773 | | | C | | Jump to Add subroutine at 2773. |
| 002203 | HLT | | | | | | Halt. |
| 002773 | | | | | | | Reserved for return address. |
| 002774 | LDA | 1001 (w) | | | | | Get augend (subtotal), put in A. |
| 002775 | ADA | 3777 | I | | C | | Add the addend specified by 3777. |
| 002776 | STA | 3000 | | | C | | Put answer in 3000. |
| 002777 | JMP | 2773 | I | | C | | Return to main program via 2773. |
| 003000 | | | | | | | Reserved for answer. |
| 003777 | 4000 | | | | | | Address of Data is 4000. |
| 004000 z | 5 | | | | | | Data (on Page 2). |
| 001001 w | | | | | | | Data (subtotal, on Page 0). |
| u | 3 | | | | | | Constant. |
| x | | | | | | | Reserved for Loop Counter. |
| v | | | | | | | Reserved for Tally. |
| y | 2 | | | | | | Data. |

the address of the subtotal (001001) and press DISPLAY MEMORY. The T-register should indicate 000013 (11, decimal), the result of calculating 5 + 3(2). Press DISPLAY MEMORY to pass over the next location (a constant) and then once more to display the content of the loop counter. All T-register lights should be off (zero). Press DISPLAY MEMORY once more to display the tally, which should be 000003, indicating three passes.

## 4-100. SPECIAL ADDRESSING METHODS.

4-101. Table 4-9 is the final expansion of the program developed in the preceding portion of this section. Two special addressing methods are illustrated by the added instructions: address modification and inter-register referencing, in which an accumulator is referenced as though it were a memory location. For the purpose of illustration, the program is expanded to solve the following problem:

$$[5 + 3(2) + (\text{sum of 4 numbers})] - 10_{10}$$

4-102. The four numbers undefined in the term sum of 4 numbers could be subtotals from other parts of a complex program. Such a program could be arranged to store these subtotals into four consecutive locations, thus making the numbers easily accessible by the programming technique known as address modification. This technique is described under paragraph 4-104. For simplicity, four fixed numbers will be manually loaded into four consecutive locations, starting at location 4000. This location was previously assigned to contain the number 5; the remaining three will be loaded as follows:

4001: $21_8$
4002: $140_8$
4003: $3570_8$

4-103. In the previous program, the answer was stored in location 3000 during the subroutine. Since the new problem demands an additional operation (subtract $10_{10}$), the new program will delay storage of the answer until this additional operation has been completed (after the subroutine). The partial answer from the subroutine will be retained in the A-register while the B-register retains the number -10. Then the contents of the two accumulators can be combined by the inter-register operation described under paragraph 4-110, addressing the accumulators.

4-104. ADDRESS MODIFICATION.

4-105. In explaining the operation of counters under paragraph 4-83, it was shown that the ISZ instruction could

be used to advance (or modify) a number contained in a specific location. Since there is no restriction on the type of word that can be in the addressed location, the number could as well be an address. For example, in the subroutine of the program in table 4-8, location 3777 contains the address 4000. The corresponding remark states that the address of data is 4000. If an ISZ instruction, referencing location 3777, incremented the number to 4001, the applicable remark would be address of data is 4001. Furthermore, if a loop were used to increment location 3777 any number of times, an entire block of data can be referenced with relatively few instructions. The basic sequence is:

```
a. ADA 3777, I
b. ISZ  3777
c. JMP a
```

4-106. Assuming that location 3777 initially contains the address 4000, the instruction in location a adds to the A-register the data whose address is contained in location 3777 (i.e., the data in 4000 is added toA). The ISZ instruction in location b increments the contents of location 3777 to 4001. Then the program jumps back to location a, and the data in location 4001 is added to the A-register. As explained under looping and counting (paragraph 4-78), some means must be provided for getting out of the loop. A common method is to compare the current reference with the last address of the block (in this case 4003), and provide an indirect jump via the return address out of the subroutine. Since the B-register is not in use, it can be used to hold the final address, for comparison purposes, and is therefore first loaded with 4003. Thus the complete sequence for the loop (to be contained within the subroutine) is:

```
d. LDB  z
a. ADA  3777, I
e. CPB  3777
f. JMP  return address, I
b. ISZ  3777
c. JMP  a
z. 4003
```

4-107. With the comparison limit in the B-register, the program advances to location a, where the quantity 5 (see table 4-9) is added to the A-register, indirectly via location 3777 (which references 4000). Then location e compares the contents of 3777 (currently 4000) with the contents of the B-register (fixed at 4003). Since the two numbers are

Table 4-8. Program to Illustrate Looping and Counting

| ADDRESS | CONTENTS | | | | | | REMARKS |
|---|---|---|---|---|---|---|---|
| | INSTRUCTION (OR DATA) | MEMORY REFERENCE | D/I | A/B | Z/C | OCTAL CODE | |
| 002166 | LDA | 1002 | | | | 061002 | INITIALIZE. |
| 002167 | CMA, INA | | | | | 003004 | Set Loop Counter to -3. |
| 002170 | STA | 1003 | | | | 071003 | |
| 002171 | CLA | | | | | 002400 | INITIALIZE. |
| 002172 | STA | 1004 | | | | 071004 | Set Tally to 0. |
| 002173 | LDA | 3777 | I | | C | 163777 | PUT 5 into A. |
| 002174 | ISZ | 1004 | | | | 035004 | LOOP. |
| 002175 | NOP | | | | | 000000 | Add 2 three times to A. |
| 002176 | ADA | 1005 | | | | 041005 | Tally number of passes. |
| 002177 | ISZ | 1003 | | | | 035003 | |
| 002200 | JMP | 2174 | | | C | 026174 | |
| 002201 | STA | 1001 | | | | 071001 | STORE subtotal from Loop in 1001. |
| 002202 | JSB | 2773 | | | C | 016773 | JUMP to Add subroutine at 2773. |
| 002203 | HLT | | | | | 102000 | HALT. |
| 002773 | | | | | | - | SUBROUTINE. |
| 002774 | LDA | 1001 | | | | 061001 | Add 5 to subtotal from Loop. |
| 002775 | ADA | 3777 | I | | C | 143777 | |
| 002776 | STA | 3000 | | | C | 073000 | |
| 002777 | JMP | 2773 | I | | C | 126773 | |
| 003000 | | | | | | - | ANSWER. |
| 003777 | 4000 | | | | | 004000 | Address of Data is 4000. |
| 004000 | 5 | | | | | 000005 | Data (on Page 2). |
| 001001 | | | | | | - | Data (subtotal, on Page 0). |
| 001002 | 3 | | | | | 000003 | Constant. |
| 001003 | | | | | | - | LOOP COUNTER. |
| 001004 | | | | | | - | TALLY. |
| 001005 | 2 | | | | | 000002 | Data (on Page 0). |

unequal, the terminating jump in location f is skipped (see CPB definition in paragraph 2-73 of specifications), and location b increments the contents of 3777 to 4001. Location c causes a jump back to the start of the loop. The next pass through the loop adds the quantity 21 (contents of 4001) to the total accumulating in the A-register. The comparison (4001 vs 4003) causes another repeat of the loop, adding the quantity 140 (contents of 4002) to the A-register. The next comparison (4002 vs 4003) is still unequal and another repeat of the loop adds 3570 (contents of 4003). This time the CPB instruction finds the contents of 3777 and of the B-register to be equal (4003 vs 4003), and the JMP instruction in location f is taken. This ends the loop.

4-108. Note that location 3777 ends with the number 4003 in it, whereas initially it must contain 4000. As explained under paragraph 4-89, this condition requires initialization. This is accomplished prior to the start of the loop by getting the number 4000 into the A-register from a location reserved to store this number as a constant, and then storing it into location 3777. Thus the following words are added to the program:

Table 4-9. Program to Illustrate Special Addressing Methods

| ADDRESS | CONTENTS | | | | | | REMARKS |
|---|---|---|---|---|---|---|---|
| | INSTRUCTION (OR DATA) | MEMORY REFERENCE | D/I | A/B | Z/C | OCTAL CODE | |
| 002166 | LDA | 1002 | | | | 061002 | INITIALIZE. |
| 002167 | CMA, INA | | | | | 003004 | Set Loop Counter to -3. |
| 002170 | STA | 1003 | | | | 071003 | |
| 002171 | CLA | | | | | 002400 | INITIALIZE. |
| 002172 | STA | 1004 | | | | 071004 | Set Tally to 0. |
| 002173 | LDA | 1006 | I | | | 161006 | PUT 5 into A. |
| 002174 | ISZ | 1004 | | | | 035004 | LOOP. |
| 002175 | NOP | | | | | 000000 | Add 2 three times to A. |
| 002176 | ADA | 1005 | | | | 041005 | Tally number of passes. |
| 002177 | ISZ | 1003 | | | | 035003 | |
| 002200 | JMP | 2174 | | | C | 026174 | |
| 002201 | STA | 1001 | | | | 071001 | STORE subtotal from Loop in 1001. |
| 002202 | JSB | 2766 | | | C | 016766 | JUMP to subroutine at 2766. |
| 002203 | LDB | 1010 | | B | | 065010 | PUT -12 |
| 002204 | CMB, INB | | | B | | 007004 | into B. |
| 002205 | ADA | 0001 | | | | 040001 | ADD -12 to subroutine total. |
| 002206 | STA | 3000 | | | C | 073000 | PUT answer in 3000. |
| 002207 | HLT | | | | | 102000 | HALT. |
| 002766 | | | | | | - | SUBROUTINE. |
| 002767 | LDA | 1006 | | | | 061006 | Add block of numbers in |
| 002770 | STA | 3777 | | | C | 073777 | locations 4000 thru 4003 |
| 002771 | LDA | 1001 | | | | 061001 | to subtotal in 1001. |
| 002772 | LDB | 1007 | | B | | 065007 | |
| 002773 | ADA | 3777 | I | | C | 143777 | Add Loop. |
| 002774 | CPB | 3777 | | B | C | 057777 | Check for Last Address. |
| 002775 | JMP | 2766 | I | | C | 126766 | Exit. |
| 002776 | ISZ | 3777 | | | C | 037777 | |
| 002777 | JMP | 2773 | | | C | 026773 | |
| 003000 | | | | | | - | ANSWER. |
| 003777 | | | | | | - | Reserved for Block addresses. |
| 004000 | 5 | | | | | 000005 | DATA BLOCK |
| 004001 | 21 | | | | | 000021 | (on Page 2). |
| 004002 | 140 | | | | | 000140 | |
| 004003 | 3570 | | | | | 003570 | |
| 001001 | | | | | | - | Data (subtotal, on Page 0). |
| 001002 | 3 | | | | | 000003 | Constant. |
| 001003 | | | | | | - | LOOP COUNTER. |
| 001004 | | | | | | - | TALLY. |
| 001005 | 2 | | | | | 000002 | Data (on Page 0). |
| 001006 | 4000 | | | | | 004000 | First address of Block. |
| 001007 | 4003 | | | | | 004003 | Last address of Block. |
| 001010 | 12 | | | | | 000012 | Data (on Page 0). |

```
g.  LDA  y
h.  STA  3777
y.  4000
```

4-109. The instruction sequences listed in the two preceding paragraphs account for all but one of the instructions for the new version of the subroutine. The one remaining instruction (as in the previous program) must put the results of the earlier subtotal (in 1001) into the A-register before the loop begins, but after initialization. The resulting 10 locations for the subroutine can now be assigned absolute addresses and transferred into the program table (locations 2766 through 2777). Location 2777 is retained as the final location of the subroutine, and the other locations are assigned working backward from this point.

4-110. ADDRESSING THE ACCUMULATORS.

4-111. As stated in paragraphs 2-35 and 2-36 of the specifications section, the A-register and the B-register can be addressed as locations 0000 and 0001 respectively. The memory cells which would ordinarily be identified by these addresses are not available to the programmer. Thus, for example, an ADA 0001 instruction would add to the A-register the contents of the B-register. Since both of these registers are accumulators, it is possible to perform separate arithmetic operations on the two accumulators, and then combine the two accumulated results with a single instruction.

4-112. In solving the problem given in paragraph 4-101, the program, up to the point of coming out of the subroutine, has performed all the arithmetic except for the subtraction of the decimal number 10 (12 octal). The result exists in the A-register. If it were necessary to derive the subtrahend by some arithmetic, such as conversion from a positive number, this can be done in the B-register, while the minuend is held in the A-register. Then the instruction ADA 0001 (add B to A) can perform the subtraction, and the result (existing in the A-register) can be stored in the location reserved for the answer (3000). Since this completes the solution of the problem, the HLT instruction can follow, and the sequence of instructions which follow exit from the subroutine will therefore be:

```
a.  LDB  z
b.  CMB, INB
c.  ADA  0001
d.  STA  3000
e.  HLT
z.  12
```

4-113. The instruction in location a puts the octal number 12 into the B-register. The combined instruction in location b converts this number to -12, and the instruction in c adds the number to the existing contents of the A-register. Location d stores the final answer into location 3000, and the program halts at location e. This sequence can now be transferred to the program table as shown in table 4-9, locations 2203 through 2207, with the constant 12 in location 1010.

4-114. LOADING THE PROGRAM. If memory remains undisturbed from preceding procedures, the program of table 4-9 can be loaded simply by loading the octal code contents into the corresponding address for those items not shaded in the table. Otherwise it is necessary to load all 42 locations in order to run the program. Five separate areas of memory are loaded, so be sure to set LOAD ADDRESS for each block.

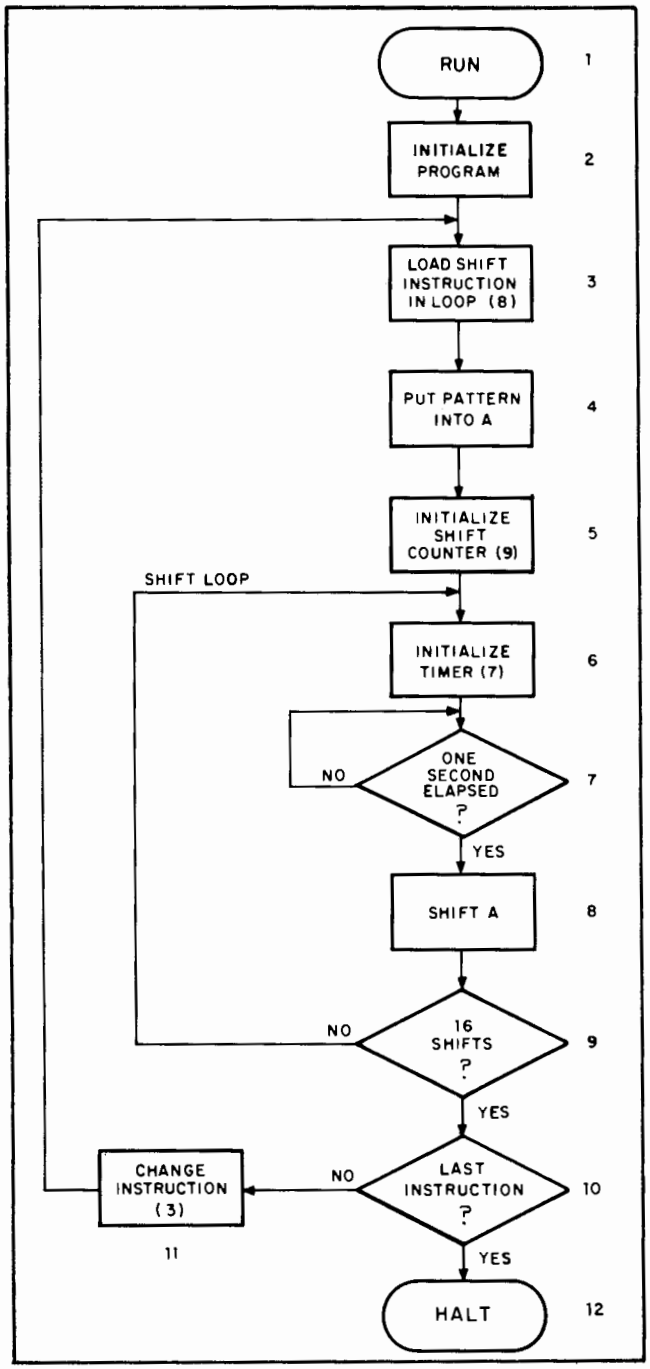
4-115. RUNNING THE PROGRAM. Set the starting address (002166) into the switch register and press LOAD ADDRESS and then RUN. To verify that the program actually was executed in the manner prescribed by the written program check the final answer and the subtotal. Load the answer address (003000) and press DISPLAY MEMORY. The answer to the problem stated in paragraphs 4-101 and 4-102 is 3757 (in octal, or in decimal 2031). Therefore the octal number 3757 must be displayed in the T-register. Now load the subtotal address (001001) and press DISPLAY MEMORY. The subtotal should be the same as in the previous program, octal 13. If the displayed subtotal is correct but the final answer is not correct, assume a loading error in the new portion of the program. If this is the case, use DISPLAY MEMORY to find the error. Reload the incorrect location and run the program again.

## 4-116. INTRODUCTION TO FLOWCHARTING.

4-117. In paragraph 4-76 it was stated that the first step in programming is to decide on general techniques, based on the problem to be solved. At this stage the programmer avoids thinking about the actions of specific instructions, but rather attempts to visualize overall operations. To assist the programmer in visualizing programs during development, flowcharts are commonly used. Figure 4-7 is an example of a flowchart. Documentation for computer software uses ASA standard block symbols in flowcharts, only three of which are used in figure 4-7. However the general principles of flowcharting can be illustrated with these few symbols. The following paragraphs trace the entire process of developing a program from a stated problem through to actual running of the program. The process consists of four distinct steps:

a. Flowcharting the program.
b. Writing the program.
c. Loading the program.
d. Running the program.

4-118. FLOWCHARTING THE PROGRAM. Suppose the problem is to set up a visual demonstration which will show, by observing the panel register lights, the action of shift and rotate instructions. (Such a demonstration may be of benefit to persons not yet well acquainted with computer operation.) The demonstration should be activated by pressing the RUN pushbutton (first symbol in figure 4-7), and should automatically terminate by a halt instruction at the end of the program (last symbol in figure 4-7). The shape of these symbols identifies a terminal operation (start or stop).




2000-40

Figure 4-7. Flowchart for Shift-Rotate Demonstration

4-119. An effective demonstration would be to put an easy-to-watch pattern into one of the accumulators (fourth block in figure 4-7), and then somehow take each of the shift-rotate instructions individually and move the bits slowly left or right from one end of the accumulator to the other. One shift every two seconds might be an acceptable rate, and 16 such shifts (the length of the accumulator) should be sufficient time to observe the action. The instruction being demonstrated should therefore change after every 16 shifts. (For brevity, a shift is meant to apply to the action of either a shift or a rotate instruction.)

4-120. The conditions of the preceding paragraph indicate the need for a means to time 1-second intervals (block 7), a means to determine if 16 shifts have occurred (block 9), and lastly a means to determine if all instructions have been demonstrated (block 10). Note that all of these blocks are diamond-shaped, which identifies a decision making capability. Generally the input fact is applied to the top of the diamond, with three possible output branches, representing yes or no decisions.

4-121. A practical means for timing and counting shifts would be to use counters; the check for last instruction could be a comparison of current instruction (block 8) with the code for the final instruction. Since both counters will go through their full count sequences several times, initialization must be provided for both (blocks 5 and 6). A means must also be provided (blocks 3 and 11) for inserting the instruction into block 8, and for changing the instruction for each demonstration loop. This accounts for all twelve blocks. It is now only necessary to arrange the program sequence and the internal loops.

4-122. As usual, the first event in the program (block 2) is to ensure, by initializing, that the program is repeatable. At this time it may not be known exactly what parts of the program will require initialization, so no specific action is stated. Next (block 3), the appropriate shift instruction must be put into the loop. Initialization in block 2 will ensure that the first listed shift instruction gets put into the loop; address modification can be used to ensure that subsequent shift instructions are put into the loop for succeeding demonstrations. Then, (block 4), since the demonstration pattern will be altered or destroyed during execution of the program, it is necessary to put the pattern into the A-register, at a point in the program where it will be reloaded at the start of each separate instruction demonstration. Next the shift counter and the timer should be initialized. The timer should start to run before executing the first shift, so that the starting condition of the pattern can be observed for at least two seconds; this is why block 7 is placed ahead of block 8. The timer (which is a very simple loop to check if two seconds have elapsed), loops back on itself for the no condition and proceeds to the execution block when the yes condition occurs. The counting of shift executions immediately follows block 8. Since the timer has run down to zero, it must be re-initialized; therefore the no branch for block 9 must loop back to a point ahead of block 6 (initialize timer). Block 5 cannot be included in the loop, or the shift counter would never advance to 16. After 16 loops have occurred (16 shifts), the yes branch of block 9 advances the program to block 10. The check for last instruction must be placed after the shift loop, since it is desired to have the yes condition halt the program; if the check were placed before the shift loop, the last shift instruction would never be demonstrated. If the comparison is a no (more instructions to demonstrate), the next event is to change the instruction in block 3, and loop back to block 3. The entire process will then be repeated for the new shift instruction.

4-123. WRITING THE PROGRAM. By creating the flow chart in figure 4-7, the following elements of the program have already been established before writing of the program begins:

a. The sequence of events.

b. The use of counters, loops, and comparisons at specific points in the sequence.

c. The number of shifts per demonstrated instruction (16).

4-124. Factors which have not been established are: how many loops comprise two seconds of elapsed time, what specific instructions are to be demonstrated, and what the pattern will be. A waiting loop to create a time delay would consist of two instructions: ISZ timer, and JMP back to ISZ. The ISZ instruction takes 4.5 microseconds to execute (refer to paragraph 2-54), and JMP takes 2.0 microseconds. This is a total of 6.5 microseconds (.0000065 second) per loop. Dividing this figure into 2 seconds gives the information that approximately 308,000 loops will provide a delay of the required time. Since the largest number the computer can handle is 65,534 or -32,768 to +32,767 (paragraph 2-106), the timer loop should count to 65,534 five times, by use of a loop within a loop:

```
a.  ISZ   z
b.  JMP   a
c.  ISZ   y
d.  JMP   a
e.  -
z.  0
y.  -5
```

4-125. Locations a and b increment location z from 0 to 65,534. The next increment returns the count to 0, location b is skipped, and location c increments location y from -5 to -4. Then the program loops back to location a, and the entire process repeats. After location z has rolled over to zero five times, location y will go from -1 to 0, causing a skip out of the loop to location e. Initialization of the loop consists of putting 0 into location z (aa and ab), and -5 into location y (ac through ae).

```
aa.  CLB
ab.  STB   z
ac.  LDB   x
ad.  CMB, INB
ae.  STB   y
x.   5
```

4-126. The instructions to be demonstrated can be stored in consecutive locations in the order listed in specifications (paragraph 2-81). This provides easy access by address modification, and also provides for convenient cross-reference to the text while the demonstration is in progress. Only the instructions which shift or rotate the A-register will be demonstrated, since the actions for the B-register are identical to those for the A-register, and since it is convenient to make use of the B-register during the program. Thus the instructions will be demonstrated in the following order:

ALS Left Shift (arithmetic)
ARS Right Shift (arithmetic)
RAL Left Rotate (16 bits)
RAR Right Rotate (16 bits)
ALR Left Shift, clear sign
ERA Rotate Right with Extend
ELA Rotate Left with Extend
ALF Rotate Left Four places



4-127. For most of these instructions, a pattern of $100401_8$ is suitable to show the movement of bits. In binary, this is:

1 000 000 100 000 001

4-128. For the ALF instruction, however, bits jump four places on each shift. Therefore a single one in the A-register would be better than 3 ones. A simple five-instruction sequence can be used to switch the pattern for the ALF instruction:

```
(not ALF)  ba.  CPB   w      (ALF)
           bb.  JMP   bf
           bc.  LDA   v
           bd.  JMP   bg
           bf.  CLA, INA
           bg.  next
           w.   ALF
           v.   100401
```

4-129. If the CPB instruction in location ba finds that the shift instruction which is about to be demonstrated is not ALF, location bb is skipped. Location bc puts the 100401 pattern into the A-register, and then a JMP instruction skips location bf. However, if the demonstration instruction is ALF, the program steps to location bb, where a jump to location bf clears the A-register, and increments the register to 000001 (CLA, INA).

4-130. Finally, all the elements of the program can be worked into the program table, as in table 4-10. Note that, in this example, the remarks column corresponds directly to the blocks in the flowchart, figure 4-7. This is not an absolute rule for programming, but a close relationship between flowchart and written program can frequently be a great help to anyone studying the program. For addresses, two blocks of memory locations (one for program instruction, one for reference data) have been assigned which are adjacent to, but do not interfere with, the locations assigned for the previous program (table 4-9). A CLE instruction has been inserted to ensure that all demonstrations begin with the extend light off.

4-131. LOADING THE PROGRAM. Set the switch register to the starting address (003001) and press LOAD ADDRESS. The first 29 addresses are in strict sequence from this starting address. Therefore memory can be loaded simply by setting the octal code into the switch register and pressing LOAD MEMORY once for each line of table 4-10. LOAD MEMORY automatically increments the address in

the P and M registers. Remember to press LOAD MEMORY once also for the reserved locations (which can be given any contents). After location 3037 has been loaded, set the switch register to 001020, press LOAD ADDRESS and load the remaining 15 locations.

4-132. RUNNING THE PROGRAM. Before running the program, refer to the definitions for shift and rotate instructions in paragraph 2-81. Set the starting address into the switch register, then press LOAD ADDRESS and RUN. Each of the eight A-register shifts and rotates will be demonstrated for 32 seconds, the results of the A-register being displayed on the switch register indicators. The total program run time is about four minutes.

## 4-133. SUMMARY.

4-134. This volume has presented a basic introduction to how the HP 2114B Computer operates, with equal emphasis on both hardware and programming. The succeeding three volumes present specialized descriptions on each of these two aspects. Volume two describes the processor hardware in detail, and volume three deals with the input/output hardware system. Volume four provides detailed information for programming of the computer with the aid of Hewlett-Packard software.

Table 4-10. Program to Demonstrate Shifts and Rotates

| ADDRESS | CONTENTS | | | | | | REMARKS |
|---|---|---|---|---|---|---|---|
| | INSTRUCTION (OR DATA) | MEMORY REFERENCE | D/I | A/B | Z/C | OCTAL CODE | |
| 003001 | LDA | 1036 | | | | 061036 | INITIALIZE |
| 003002 | STA | 3004 | | | C | 073004 | Get first Load instruction. |
| 003003 | CLE | | | | | 002100 | |
| 003004 | | | | | | - | LOAD shift instruction |
| 003005 | STB | 3027 | | B | C | 077027 | into loop. |
| 003006 | CPB | 1027 | | B | | 055027 | PUT pattern into A. |
| 003007 | JMP | 3012 | | | C | 027012 | If ALF, use 000001. |
| 003010 | LDA | 1035 | | | | 061035 | All others use 100401. |
| 003011 | JMP | 3013 | | | C | 027013 | |
| 003012 | CLA, INA | | | | | 002404 | |
| 003013 | LDB | 1034 | | B | | 065034 | INITIALIZE Shift Counter. |
| 003014 | CMB, INB | | | B | | 007004 | Set to -16. |
| 003015 | STB | 1033 | | B | | 075033 | |
| 003016 | CLB | | | B | | 006400 | INITIALIZE Timer. |
| 003017 | STB | 1030 | | B | | 075030 | Set to loop for 2 seconds. |
| 003020 | LDB | 1032 | | B | | 065032 | |
| 003021 | CMB, INB | | | B | | 007004 | |
| 003022 | STB | 1031 | | B | | 075031 | |
| 003023 | ISZ | 1030 | | | | 035030 | LOOP. |
| 003024 | JMP | 3023 | | | C | 027023 | Two seconds. |
| 003025 | ISZ | 1031 | | | | 035031 | |
| 003026 | JMP | 3023 | | | C | 027023 | |
| 003027 | | | | | | - | SHIFT. (Instruction loaded by 3003.) |
| 003030 | OTA | 0002 | | | | 102601 | DISPLAY A. |
| 003031 | ISZ | 1033 | | | | 035033 | LOOP. |
| 003032 | JMP | 3016 | | | C | 027016 | 16 Shifts, two per second. |
| 003033 | LDB | 1027 | | B | | 065027 | CHECK |
| 003034 | CPB | 3027 | | B | C | 057027 | for last instruction. |
| 003035 | HLT | | | | | 102000 | HALT. |
| 003036 | ISZ | 3004 | | | C | 037004 | CHANGE instruction |
| 003037 | JMP | 3003 | | | C | 027003 | and repeat demonstration. |
| 001020 | ALS | | | | | 001000 | DEMONSTRATION instructions. |
| 001021 | ARS | | | | | 001100 | |
| 001022 | RAL | | | | | 001200 | |
| 001023 | RAR | | | | | 001300 | |
| 001024 | ALR | | | | | 001400 | |
| 001025 | ERA | | | | | 001500 | |
| 001026 | ELA | | | | | 001600 | |
| 001027 | ALF | | | | | 001700 | |
| 001030 | | | | | | - | TIMER. |
| 001031 | | | | | | - | Rollover counter (-5). |
| 001032 | 5 | | | | | 000005 | Constant. |
| 001033 | | | | | | - | SHIFT COUNTER. |
| 001034 | 20 | | | | | 000020 | Decimal 16. |
| 001035 | 100401 | | | | | 100401 | Pattern. |
| 001036 | LDB | 1020 | | | | 065020 | First Load instruction. |

# APPENDIX A
# REFERENCE TABLES

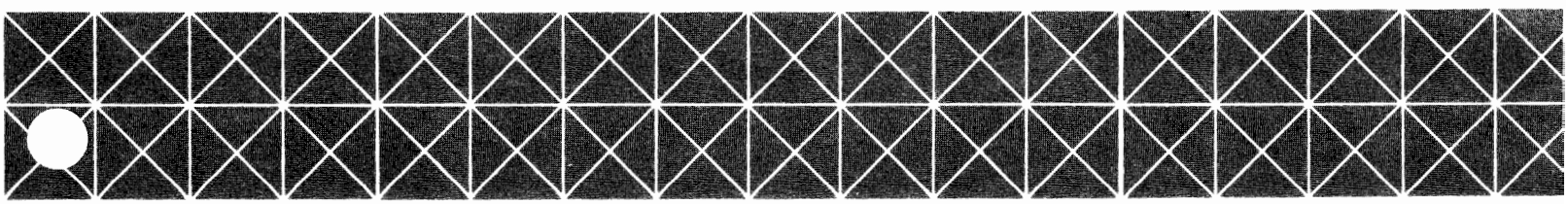


Table A-1. Glossary of Terms Used in this Volume

**absolute** — Pertaining to an address fully defined by a memory address number, or to a program which contains such addresses (as oppossed to one containing symbolic addresses).

**accumulator** — A register in which numbers are totaled or manipulated, or temporarily stored for transfers to and from memory or external devices.

**add** — Restrictive (HP 2114B): two's complement addition of binary numbers. General: any arithmetic addition.

**address** — A number (noun) which identifies a location in memory. Also (verb), the process of directing the computer to read a specified memory location (synonymous with "reference").

**address modification** — A programming technique of changing the address referred to by a memory reference instruction, so that each time that particular instruction is executed, it will affect a different memory location.

**address word** — A computer word which contains only the address of a memory location.

**ALGOL** — A programming language (or the compiler which translates this language) which permits programs to be written using common arithmetic conventions and terms.

**alter** — A modification of the contents of an accumulator or extend bit; e.g., clear, complement, or increment.

**analog** — Pertaining to information which can have continuously variable values, as opposed to digital information, which can be varied in degrees no smaller than the value of the least significant digit.

**"and"** — A logical operation in which the resultant quantity (or signal) is true if all of the input values are true, and is false if at least one of the input values is false.

**A-register** — One of the two HP 2114B Computer accumulator registers. These registers are used for arithmetic operations and for information transfers to and from device interfaces.

**arithmetic logic** — The circuitry involved in manipulating the information contained in a computer's accumulators.

**arithmetic operation** — Restrictive: a mathematical operation involving fundamental arithmetic (addition, substraction, multiplication, division), specifically excluding logical and shifting operations. General: any manipulation of numbers.

**Assembler** — A program for the HP 2114B Computer (or any computer) which converts a program prepared in symbolic form (i.e., using defined symbols and mnemonics to represent instructions, addresses, etc.) to binary machine language.

**base** — The quantity of different digits used in a particular numbering system. The base in the binary numbering system is two; thus there are two digits (0 and 1). In the decimal system (base 10), there are ten digits (0 through 9).

**base page** — The lowest numbered page of computer memory. It can be directly addressed from any other page.

**BASIC** — A programming language (or the compiler which translates this language) which permits programs to be written in a form that is simple and easy to learn.

**basic binary loader** — A series of instructions for the HP 2114B Computer which will load, into memory, programs prepared with absolute addresses, using defined input devices.

**basic control system** — A collection of programs for the HP 2114B Computer which direct the loading, combining, library searching, debugging, and input/output procedures for programs generated by the user.

**binary** — Denoting the numbering system based on the radix two. Binary digits are restricted to the values 0 and 1.

**binary-coded decimal** — A coding method for representing each decimal digit (0-9) by specific combinations of four binary bits. For example, the 8-4-2-1 "bdc" code commonly used with the HP 2114B Computer represents 1 as 0001, and 9 as 1001.

**binary point** — The fractional dividing point of a binary numeral; equivalent to decimal point in the decimal numbering system.

**binary program** — A program (or its recorded form) in which all information is in binary machine language.

**bit** — A single digit in a binary number, or in the recorded representation of such a number (by hole punches, magnetic states, etc.). The digit can have one of only two values, 0 or 1.

**bit density** — A physical specification referring to the number of bits which can be recorded per unit of length or area.

**bit-serial** — One bit at a time, as opposed to bit-parallel in which all bits of a character can be handled simultaneously.

Table A-1. Glossary of Terms Used in this Volume (Continued)

**bistable** — Pertaining to an electronic circuit having two stable states, controllable by external switching signals; analogous to an on-off switch.

**B-register** — One of the two HP 2114B Computer accumulator registers. These registers are used for arithmetic operations and for information transfers to and from device interfaces.

**buffer** — A register used for intermediate storage of information in the transfer sequence between the computer accumulators and a peripheral device. In the HP 2114B Computer, the buffer is located inside the computer on the device interface card.

**bus** — A major electrical path connecting two or more electrical circuits.

**carry** — A digit, or equivalent signal, resulting from an arithmetic operation which causes a positional digit to equal or exceed the base of the effective numbering system.

**central interrupt register** — A six-bit register which holds the address of the last I/O device to send an interrupt signal to the computer. The contents of the CIR are accessible by program using a LIA instruction with a select code of 04.

**character** — The general term to include all symbols such as alphabetic letters, numerals, punctuation marks, mathematical operators, etc. Also, the coded representation of such symbols.

**checkerboard** — An alternating pattern of zeros and ones stored in a computer for testing purposes.

**clear** — Reset; the binary "0" condition.

**code** — A system of symbols which can be used by machines such as a computer, and which in specific arrangements have a special external meaning.

**communication system** — A computer system having facilities for long-distance transfers of information between remote and central stations.

**comparator** — An instrument for comparing digitized measurements against presettable upper and lower limits, and giving an indication of the comparison reult.

**compiler** — A language translation program, used to transform symbols meaningful to a human operator to codes meaningful to a computer. More restrictively, a program which translates a machine-independent source language into the machine language of a specific computer, thus excluding assemblers.

**computation** — The processing of information within the computer.

**computer (digital)** — An electronic instrument capable of accepting, storing, and arithmetically manipulating information, which includes both data and the controlling program. The information is handled in the form of coded binary digits (0 and 1), represented by dual voltage levels, magnetic states, punched holes, etc.

**computer word** — See "word".

**configuration** — The arrangement of either hardware instruments or software routines when combined to operate as a system.

**configurator** — A computer program whose purpose is to combine a number of program segments into an integrated whole, in a specific desired manner (configuration).

**contents** — The information stored in a register or a memory location.

**control bit** — A signal, or the stored indication of this signal, which controls the transfer of information to and from peripheral devices associated with the HP 2114B Computer.

**core** — The smallest element of a core storage memory module. It is a ring of ferrite material, 0.03-inch in diameter in the HP 2114B Computer and can be magnetized in clockwise or counterclockwise directions to represent the two binary digits, 0 and 1.

**crossbar scanner** — A device for sequentially connecting multi-wire analog signals to a digital measuring device, using a crossbar switch (a switch specially designed for accurate transfer of low-level, high-frequency, and high-impedance signals).

**current page** — The memory page comprising all those locations which are on the same page as a given instruction.

**data acquisition** — The transformation of raw information gathered by measuring or recording equipment into a more condensed, organized, or useful form.

**data word** — A computer word consisting of a number, a fact, or other information which is to be processed by the computer.

**debug** — Check for and correct errors in a program.

**decimal** — Denoting the numbering system based on the radix ten.

**decrement** — To change the value of a number in the negative direction. If not otherwise stated, a decrement by one is usually assumed.

**device** — An electronic or electromechanical instrument. Most commonly implies measuring, reading, or recording equipment.

Table A-1. Glossary of Terms Used in this Volume (Continued)

**diagnostic** — (adj) Relating to test programs for detection of errors in the functioning of hardware or software, or the messages resulting from such tests. Also (noun), the test program or message itself.

**digital voltmeter** — An electronic voltage measuring device which provides a readout in digital form on the instrument panel, and commonly (essential for computer purposes) also codes the measurement result in binary-coded decimal form as an electrical output.

**direct memory access** — A means of transferring a block of information words directly between an external device and computer memory bypassing the need for repeating a service routine for each word. This method greatly speeds the transfer process.

**disable** — A signal condition which prohibits some specific event from proceeding.

**disc storage** — A means of storing binary digits in the form of magnetized spots on a rotating circular metal plate coated with a magnetic material. The information is stored and retrieved by read-write heads positioned over the surface of the disc.

**documentation** — Manuals and other printed materials (tables, listings, diagrams, etc.) which provide instructive information for usage and maintenance of a manufactured product, including both hardware and software.

**double-length word** — A word, due to its length, which requires two computer words to represent it. Double-length words are normally stored in two adjacent memory locations.

**driver** — An input/output routine to provide automatic operation of a specific device with the computer.

**dump** — To record memory contents on an external medium (e.g., tape).

**effective address** — The address of a memory location ultimately affected by a memory reference instruction. It is possible for one instruction to go through several indirect addresses to reach the effective address.

**electronic counter** — An electronic instrument used to measure physical quantities by specially controlled counting of electrical pulses.

**enable** — A signal condition which permits some specific event to proceed, whenever it is ready to do so.

**"exclusive or"** — A logical operation in which the resultant quantity (or signal) is true if at least one (but not all) of the input values is true and is false if the input values are all true or all false.

**execute** — To fully perform a specific operation, such as would be accomplished by an instruction or a program.

**execute phase** — A predetermined state of the internal computer logic which causes the computer to interpret as data the information read out of memory during a memory cycle.

**exit sequence** — A series of instructions to conclude operation in one area of a program and to move to another area.

**extend** — A one-bit register in the HP 2114B Computer, which extends the effective length of the A- or B-register to 17 bits for certain additions and rotations.

**fetch phase** — A predetermined state of the internal computer logic which causes the computer to interpret as an instruction the information read out of memory during a memory cycle.

**fixed point** — A numerical notation in which the factional point (whether decimal, octal, or binary) appears at a constant, predetermined position. Compare with floating point.

**flag bit** — A signal, or the stored indication of this signal, which indicates the readiness of a peripheral device of the HP 2114B Computer to transfer information.

**flip-flop** — An electronic circuit having two stable states, and thus capable of storing a binary digit. Its states are controlled by signal levels at the circuit input, and are sensed by signal levels at the circuit output.

**floating point** — A numerical notation in which the integer and the exponent of a number are separately represented (frequently by two computer words), so that the implied position of the fractional point (decimal, octal, or binary) can be freely varied with respect to the integer digits. Compare with fixed point.

**flowchart** — A diagram representing the operation of a computer program.

**format** — A predetermined arrangement of bits or characters.

**Formatter** — A program which provides the linkage between FORTRAN read/write statements and the basic control system input/output control program, with any appropriate conversions.

**FORTRAN** — A programming language (or the compiler which translates this language) which permits programs to be written in a form resembling algebra, rather than in detailed instruction-by-instruction form (as for assemblers).

Table A-1. Glossary of Terms Used in this Volume (Continued)

**FORTRAN Library** — A collection of programs for the HP 2114B Computer to provide the user with commonly used mathematical and formatting routines.

**gate** — An electronic circuit capable of performing logical functions such as "and", "or", "nor", etc.

**hardware** — Electronic or electromechanical components, instruments, or systems.

**hardware diagnostics** — A collection of programs for the HP 2114B Computer deisgned to assist in the identification of hardware malfunctions.

**high core** — Core memory locations having high-numbered addresses.

**"inclusive or"** — A logical operation in which the resultant quantity (or signal) is true if at least one of the input values is true, and is false if the input values are all false.

**increment** — To change the value of a number in the positive direction. If not otherwise stated, an increment by one is usually assumed.

**incremental magnetic tape** — A form of magnetic tape recording in which the recording transport advances by small increments (e.g. 0.005 inch), stopping the tape advancement long enough to record one character at the spot location under the recording head.

**indirect address** — The address initially specified by an instruction when it is desired to use that location to re-direct the computer to some other location to find the effective address for the instruction.

**indirect phase** — A predetermined state of the internal computer logic which causes the computer to interpret as an address the information read out of memory during a memory cycle.

**information** — A unit or set of knowledge represented in the form of discrete words, consisting of an arrangement of symbols or (so far as the digital computer is concerned) binary digits.

**inhibit** — To prevent a specific event from occurring.

**initialize** — The procedure of setting various parts of a stored program to starting values, so that the program will behave the same way each time it is repeated. The procedures are included as part of the program itself.

**input** — Information transferred from a peripheral device into the computer. Also can apply to the transfer process itself.

**input/output** — Relating to the equipment or method used for transmitting information into or out of the computer.

**input/output channel** — The complete input or output facility for one individual device or function, including its assigned position in the computer, the interface circuitry, and the external device.

**input/output control** — A program of the computer basic control system which provides linkage between the input/output requests of a user program and the appropriate drivers.

**input/output system** — The circuitry involved in transferring information between the computer accumulators and its peripheral devices.

**instruction** — A written statement, or the equivalent computer-acceptable code, which tells the computer to execute a specified single operation.

**instruction code** — The arrangement of binary digits which tell the computer to execute a particular instruction.

**instruction logic** — The circuitry involved in moving binary information between registers, memory, and buffers in prescribed manners, according to instruction codes.

**instruction register** — An internal 6-bit register of the HP 2114B Computer, which forms part of its instruction logic. The instruction register receives the 6 most significant bits of the T-register when each new instruction is read out of memory, and retains these bits for instruction identification. It is not usually considered to be a working register.

**instruction word** — A computer word containing an instruction code. The code bits may occupy all or (as in the case of memory reference instruction words) only part of the word.

**interface** — The connecting circuitry which links the central processor of a computer system to its peripheral devices.

**interrupt** — The process, initiated by an external device, which causes the computer to interrupt a program in progress, generally for the purpose of transferring information between that device and the computer.

**interrupt location** — A memory location whose contents (always an instruction) are executed upon interrupt by a specific device.

**interrupt phase** — A predetermined state of the internal computer logic which causes the computer to suspend operation of a program in progress, and branch to a specific service routine.

**jump** — An instruction which breaks the strict sequential location-by-location operation of a program, and directs the computer to continue at another specified location anywhere in memory.

Table A-1. Glossary of Terms Used in this Volume (Continued)

**label** — Any arrangement of symbols, usually alphanumeric, used in place of an absolute memory address in computer programming.

**language** — The set of symbols, rules, and conventions used to convey information, either at the human level or at the computer level.

**library routine** — A routine designed to accomplish some commonly used mathematical function, and kept permanently available on a library program tape (e.g., HP FORTRAN Library).

**linearizer** — An instrument for converting the measurements made by a digital voltmeter to the normal engineering units of the physical quantity being measured.

**load** — Put information into (memory, a register, etc.). Also (e.g., loading tape), to put the information medium into the appropriate device.

**loader** — A program designed to assist in transferring information from an external device into computer's memory.

**location** — A group of storage elements in the computer's memory (e.g., 17 cores in the HP 2114B memory module), which can store one computer word. Each such location is identified by a number ("address") to facilitate storage and retrieval of information in selectable locations.

**logical operation** — A mathematical process based on the principles of truth tables; e.g., "and", "inclusive or" and "exclusive or" operations.

**logic diagram** — A diagram that represents the detailed internal functioning of electronic hardware, using binary logic symbols rather than electronic component symbols (see "schematic diagram").

**logic equation** — A written mathematical statement, using symbols and rules derived from Boolean algebra. Specifically (hardware design), a means of stating the conditions required to obtain a given signal.

**loop** — A sequence of instructions in which the last instruction is a jump back to the first instruction.

**low core** — Core memory locations having low-numbered addresses.

**machine** — Pertaining to the computer hardware (e.g., machine timing, machine language).

**machine language** — The form of coded information (consisting of binary digits) which can be directly accepted and used by the computer. Other languages require translation to this form, generally with the aid of translation programs (assemblers and compilers).

**machine timing** — The regular cycle of events in the operation of internal computer circuitry. The actual events will differ for various processes, but the timing is constant through each recurring cycle.

**macroinstruction** — An instruction, similar in binary coding to the computer's basic machine language instructions, which is capable of producing a variable number of machine language instructions.

**magnetic tape recording** — A means of recording information on a strip of magnetic coated material, such that binary bits can be represented by reversals of the direction of magnetization.

**magnitude** — That portion of a computer word which indicates the absolute value of a number, thus excluding the sign bit.

**math routine** — A program designed to accomplish a single mathematical function.

**media conversion** — The transferral of recorded information from one recording medium (e.g., punched paper tape, magnetic tape, etc.) to another recording medium.

**memory** — An organized collection of storage elements (e.g., ferrite cores), into which a unit of information consisting of a binary digit can be stored, and from which it can later be retrieved. Also, a device not necessarily having individual storage elements, but which has the same storage and retrieval capabilities (e.g., magnetic discs).

**memory cycle** — That portion of the computer's internal timing during which the contents of one location of memory are read out (into the transfer register) and written back into that location.

**memory module** — A complete segment of core storage, capable of storing a definable number of computer words (e.g., 4096 words in the HP 2114B Computer memory module). Computer storage capacity is frequently rounded off and abbreviated as 4K (i.e., 4096 or approximately 4000 words) or 8K (8192 or 8000).

**memory protect** — A means of preventing inadvertent alteration of a selectable segment of memory.

**memory reference** — The address of the memory location specified by a memory reference instruction; i.e., the location affected by the instruction.

**merge** — "Inclusive or".

**microinstruction** — An instruction which forms part of a larger, composite instruction.

**mnemonic** — An abbreviation or arrangement of symbols used to assist human memory. For example, STB calls to mind the term "store B-register" much more readily than would, say, "instruction 74".

Table A-1. Glossary of Terms Used in this Volume (Continued)

**M-register** — The memory address register of the HP 2114B Computer; i.e., the register which controls the access to each memory location.

**multi-level indirect** — Indirect addressing using two or more indirect addresses in sequence to find the effective address for the current instruction.

**multiple-precision** — Referring to arithmetic in which the computer, for greater accuracy, uses two or more words to represent one number.

**Mylar** — A DuPont trademark for a polyester film used as a more durable medium (in place of paper tape) for punched tape records, and as a base for magnetic tape.

**nine's complement** — A number so modified that the addition of the modified number and its original value, plus one, will equal an even power of ten. A nine's complement number is obtained mathematically by subtracting the original value from a string of 9's.

**non-return to zero** — A technique of magnetic tape recording in which the recording device does not turn off the magnetizing flux between recording of individual characters. The flux is always at saturation level during recording, and bits are indicated by reversals of flux polarity.

**nuclear scaler** — A system of electronic instruments used to detect and analyze nuclear events, such as gamma ray measurements.

**octal** — Denoting a numbering system based on the radix eight. Octal digits are restricted to the values 0 through 7.

**octal code** — A notation for writing machine language programs with the use of octal numbers instead of binary numbers.

**octal point** — The fractional dividing point of an octal numeral; equivalent to decimal point in the decimal numbering system.

**off line** — Pertaining to the operation of peripheral equipment not under control of the computer.

**one's complement** — A number so modified that the addition of the modified number and its original value, plus one, will equal an even power of two. A one's complement number is obtained mathematically by subtracting the original value from a string of 1's, and electronically by inverting the states of all binary bits in the number.

**on line** — Pertaining to the operation of peripheral equipment under computer control.

**output** — Information transferred from the computer to a peripheral device. Also can apply to the transfer process itself.

**output coupler** — An instrument which provides the interconnecting circuitry between a measuring instrument and a recording instrument.

**overflow** — A one-bit register in the HP 2114B Computer, which indicates that the result of an addition in the A- or B-register has exceeded the maximum possible signed value (+32767 or -32768, decimal). The addition result will therefore be missing one or more significant bits.

**packed word** — A computer word containing two or more independent units of information. This is done to conserve storage when information requires relatively few bits of the computer word.

**page** — An artificial division of memory consisting of a fixed number of locations, dictated by the direct addressing range of memory reference instructions.

**page zero** — The memory page which includes the lowest numbered memory addresses.

**parity bit** — A supplementary bit added to an information word to make the total of one-bits be always either odd or even. This permits checking the accuracy of information transfers.

**pass** — The complete process of reading a set of recorded information (one tape, one set of cards, etc.) through an input device, from beginning to end.

**peripheral device** — An instrument or machine electrically connected to the computer, but which is not part of the computer itself.

**phase** — One of several specific states of the internal computer logic, usually set up by instructions being executed, to determine how the computer should interpret information read out of memory.

**photoelectric reader** — An input device which senses characters (on punched tape, cards, pages, etc.) by optical light strobe and detection circuits. An example is the HP 2748A Tape Reader.

**plane** — An arrangement of ferrite cores on a matrix of control and sensing wires. Several planes stacked together form a memory module.

**power failure control** — A means of sensing primary power failure so that a special routine may be executed in the finite period of time available before the regulated dc supplies discharge to unusable levels. The special routine may be used to preserve the state of a program in progress, or to shut down external processes.

**P-register** — The program counter register of the HP 2114B Computer; i.e., the register which keeps track of (or "counts") the stored locations of the instructions in a program being executed.

Table A-1. Glossary of Terms Used in this Volume (Continued)

**prepare control system** — A program designed to assist in the preparation of a basic control system program, to a specified arrangement of input/output devices.

**priority** — The automatic regulation of events so that chosen actions will take precedence over others in cases of timing conflict.

**process control** — Automatic control of manufacturing processes by use of a computer.

**processor** — The central unit of a computer system (i.e., the device which accomplishes the arithmetic manipulations), exclusive of peripheral devices. Frequently (when used as adjective) also excludes interface components, even though normally contained within the processor unit; thus processor options exclude interface (input/output) options.

**program** — A plan for the solution of a problem by a computer, and consisting of a sequence of computer instructions.

**program listing** — A printed record (or equivalent binary-output program) of the instructions in a program.

**programmer** — A person who writes computer programs. Also (hardware), an interface card or instrument which sets up (or programs) the various functions of one measuring instrument.

**programming** — The process of creating a program.

**proximity switch** — A capacitance activated contact switch.

**pseudo-instruction** — A symbolic statement, similar to assembly language instructions in general form, but meaningful only to the program containing it, rather than to the computer as a machine instruction.

**punched tape** — A strip of tape, usually paper, on which information is represented by coded patterns of holes punched in columns across the width of the tape. Commonly (as used with the HP 2114B Computer), there are 8 hole positions (channels) across the tape.

**quartz thermometer** — An electronic temperature measuring instrument using the linear temperature sensing properties of specially cut quartz crystals. An example is the HP 2801A Quartz Thermometer, which provides a digital output usable as an input to a digital computer, such as the HP 2114B Computer.

**read** — The process of transferring information from an input device into the computer. Also, the process of taking information out of computer memory (see "memory cycle").

**real time** — Time elapsed between events occurring externally to the computer. A computer which accepts and processes information from one such event and is ready for new information before the next event occurs is said to operate in a real-time environment.

**reference** — Shortened form of "memory reference".

**register** — An array of hardware binary circuits (flip-flops, switches, etc.) for temporary storage of information. Unlike mass storage devices such as memory cores, registers can be wired to permit flexible control of the contained information, for arithmetic operations, shifts, transfers, etc.

**relocatable** — Pertaining to programs whose instructions can be loaded into any stated area of memory.

**relocating loader** — An HP computer program capable of loading and combining relocatable programs (i.e., programs having symbolic rather than absolute addresses).

**reset** — A signal condition representing a binary "0".

**rotate** — A positional shift of all bits in an accumulator (and possibly an extend bit as well), with those bits lost off one end of the accumulator rotated around to enter vacated positions at the other end.

**routine** — A program or program segment designed to accomplish a single function.

**sampling** — The process of taking a measurement of a signal existing at a measuring instrument's input during a short (sample) period. The length of the sample period is a predetermined function of the measuring instrument.

**scanner** — A device for sequentially switching multiple signal sources to one measuring or recording instrument.

**schematic diagram** — A diagram that represents the detailed internal electrical circuit arrangement of electronic hardware, using conventional electronic component symbols.

**select code** — A number assigned to input/output channels for purposes of identification in information transfers between the computer and external devices.

**service routine** — A sequence of instructions designed to accomplish the transfer of information between a particular device and the computer.

**set** — A signal condition representing a binary "1".

**seven's complement** — A number so modified that the addition of the modified number and its original value, plus one, will equal an even power of eight. A seven's complement number is obtained mathematically by subtracting the original value from a string of 7's.

Table A-1. Glossary of Terms Used in this Volume (Continued)

**shift** — Restrictive (arithmetic shift): to multiply or divide the magnitude portion of a word (bits 0 through 14 in the HP 2114B Computer) by a power of two using a positional shift of these bits. General: any positional shift of bits.

**sign** — The algebraic plus or minus indicator for a mathematical quantity. Also, the binary digit or electrical polarity representing same.

**significant digit** — A digit so positioned in a numeral as to contribute a definable degree of precision to the numeral. In conventional written form, the most significant digit in a numeral is the leftmost digit, and the least significant digit is the rightmost digit.

**skip** — An instruction which causes the computer to omit the instruction in the immediately following location. A skip is usually arranged to occur only if certain specified conditions are sensed and found to be true, thus allowing various decisions to be made.

**software** — Computer programs. Also, the tapes or cards on which the programs are recorded.

**software package** — A complete collection of related programs, not necessarily combined as a single entity.

**source program** — A program (or its recorded form) written in some programming language other than machine language and thus requiring translation. The translated form is the object program.

**starting address** — The address of a memory location in which is stored the first instruction of a given program.

**statement** — An instruction in any computer-related language other than machine language.

**store** — To put information into a memory location, register, or device capable of retaining the information for later access.

**subroutine** — A sequence of instructions designed to perform a single task, with provisions included to allow some other program to cause execution of the task sequence as if it were part of its own program.

**symbolic address** — A label assigned in place of absolute numeric addresses, usually for purposes of relocation (see relocatable).

**symbolic editor** — A program for HP computers which is used to add, delete, or correct selectable portions of any symbolic program.

**symbolic file** — A recorded collection of computer words, with a symbolic address assigned to each word.

**system** — An assembly of units (e.g., hardware instruments or software routines), combined to work as a larger integrated unit having the capabilities of all the separate units.

**system input/output (software)** — A collection of input/output programs to add input/output capability to HP FORTRAN, assembler, and symbolic editor, and to some user programs.

**S-register** — The switch register of the HP 2114B Computer; i.e., the register used to input data either manually or by program or to output data by program control.

**time period** — The smallest division of time in the HP 2114B Computer internal timing cycle (see "machine timing").

**T-register** — The transfer register of the HP 2114B Computer; i.e., the register which directly receives words from memory, and directly applies words to memory.

**truth table** — A table listing all possible configurations and resultant values for any given Boolean algebra function.

**two's complement** — A number so modified that the addition of the modified number and its original value will equal an even power of two. Also, a kind of arithmetic which represents negative numbers in two's complement form so that all addition can be accomplished in only one direction (positive incrementation). A two's complement number is obtained mathematically by subtracting the original value from an appropriate power of the base two (i.e., from $1_1$, $10_2$, $100_2$, etc.), and electronically by inverting the states of all binary bits in the number and adding one (complement and increment).

**updated program** — A program to which additions, deletions, or corrections have been made.

**user** — The person or persons who program and operate a particular computer.

**utility routine** — A standard routine to assist in the operation of the computer (e.g., device drivers, sorting routines, etc.) as opposed to mathematical (library) routines.

**waiting loop** — A sequence of instructions (frequently only two) which are repeated indefinitely until a desired external event occurs, such as the receipt of a flag signal.

**word** — A set of binary digits handled by the computer as a unit of information. Its length is determined by hardware design; e.g., the number of cores per location, and the number of flip-flops per register.

Table A-1. Glossary of Terms Used in this Volume (Continued)

**working register** — A register whose contents can be modified under control of a program. Thus a register consisting of manually-operated switches is not considered a working register.

**write** — The process of transferring information from the computer to an output device. Also, the process of storing (or restoring) information into computer memory (see "memory cycle").

Table A-2. Mnemonics and Abbreviations

| | |
|---|---|
| A | A-register (A accumulator) |
| B | B-register (B accumulator) |
| C | Current page (page addressing) |
| C | Clear (flag or overflow) |
| C | Control (bit or signal) |
| C | Centrigrade |
| D | Direct (addressing) |
| D | Disable (microinstruction group) |
| E | Extend |
| E | Enable (microinstruction group) |
| F | Flag (bit or signal) |
| F | Fahrenheit |
| H | Hold (flag or overflow) |
| I | Indirect (addressing) |
| I | I-register (instruction register) |
| K | Kilo (thousand) |
| M | M-register (memory address) |
| P | P-register (program counter) |
| T | T-register (transfer register) |
| T | Time periods |
| Z | Page zero |
| HP | Hewlett-Packard |
| I/O | Input/output |
| IR | Instruction register |
| PH | Phase |
| RB | R-bus |
| RL | Rotate left |
| SL | Shift left |
| TB | T-bus |
| TR | T-register |
| ADA | Add to A |
| ADB | Add to B |
| ADF | Add function |
| ALF | Rotate A left four places |
| ALR | A left shift, clear sign |
| ALS | A left shift |
| AND | "And" instruction |
| ANF | "And" function |
| ARS | A right shift |
| ASA | American Standards Association |
| ASG | Alter-skip group |
| ASR | Automatic send-receive |
| BCS | Basic control system |
| BLF | Rotate B left four places |
| BLR | B left shift, clear sign |
| BLS | B left shift |
| BRS | B right shift |
| CCA | Clear and complement A |
| CCB | Clear and complement B |
| CCE | Clear and complement extend |
| CLA | Clear A |
| CLB | Clear B |
| CLC | Clear control |
| CLE | Clear extend |
| CLF | Clear flag |
| CLO | Clear overflow |
| CMA | Complement A |
| CMB | Complement B |
| CME | Complement extend |
| CMF | Complement function |
| CPA | Compare to A |
| CPB | Compare to B |
| DMA | Direct memory access |
| ELA | Rotate extend left with A |
| ELB | Rotate extend right with B |
| EOF | "Exclusive or" function |
| ERA | Rotate extend right with A |
| ERB | Rotate extend right with B |
| HLT | Halt |
| INA | Increment A |
| INB | Increment B |
| IOF | "Inclusive or" function |
| IOG | Input/output group |
| IOR | "Inclusive or" instruction |
| ISZ | Increment, skip if zero |
| JMP | Jump |
| JSB | Jump to subroutine |
| LDA | Load (memory) into A |
| LDB | Load (memory) into B |
| LIA | Load input into A |
| LIB | Load input into B |
| MAC | Macroinstruction |
| MIA | Merge into A |
| MIB | Merge into B |
| NOP | No operation |

Table A-2. Mnemonics and Abbreviations (Continued)

| | |
|---|---|
| OTA | Output from A |
| OTB | Output from B |
| OVF | Overflow flip-flop |
| RAL | Rotate A left |
| RAR | Rotate A right |
| RBL | Rotate B left |
| RBR | Rotate B right |
| RLL | Rotate left to least significant bit |
| RRS | Rotate right to sign bit |
| RSS | Reverse skip sense |
| SEZ | Skip if extend is zero |
| SFC | Skip if flag is clear |
| SFS | Skip if flag is set |
| SIO | System input/output |
| SKF | Skip on flag (signal) |
| SLA | Skip if least significant bit of A is zero |
| SLB | Skip if least significant bit of B is zero |
| SLM | Shift left magnitude |
| SOC | Skip if overflow clear |
| SOS | Skip if overflow set |
| SRG | Shift-rotate group |
| SRM | Shift right magnitude |
| SSA | Skip if sign of A is zero |
| SSB | Skip if sign of B is zero |
| STA | Store A |
| STB | Store B |
| STC | Set control |
| STF | Set flag |
| STO | Set overflow |
| SZA | Skip if A is zero |
| SZB | Skip if B is zero |
| XOR | "Exclusive or" instruction |
| IOIC | I/O input control |
| IOOC | I/O output control |
| NRZI | Non-return to zero, invert |
| ac | Alternating current |
| A | Amperes |
| bcd (BCD) | Binary-coded decimal |
| bin. | Binary |
| bpi | Bits per inch |
| BTU/hr | British thermal units, per hour |
| C16 | Bit 16 carry |
| Compl | Complement |
| dc | Direct current |
| Dec. | Decimal |
| e.g. | For example (exempli gratia) |
| Hz | Hertz (cycles per second) |
| i.e. | That is (id est) |
| in. | Inch |
| incl | Included |
| ips | Inches per second |
| kg | Kilograms |
| lb | Pound |
| mA | Milliamperes |
| MHz | Megahertz (megacycles per second) |
| ms | Milliseconds |
| mV | millivolts |
| oct | Octal |
| sec | Seconds |
| sel | Select (code) |
| V | Volts |
| Vac | Volts (alternating current) |

Table A-3. Powers of Two

| | | |
|---|---|---|
| 1 | 0 | 1 0 |
| 2 | 1 | 0 5 |
| 4 | 2 | 0 25 |
| 8 | 3 | 0 125 |
| 16 | 4 | 0 062 5 |
| 32 | 5 | 0 031 25 |
| 64 | 6 | 0 015 625 |
| 128 | 7 | 0 007 812 5 |
| 256 | 8 | 0 003 906 25 |
| 512 | 9 | 0 001 953 125 |
| 1 024 | 10 | 0 000 976 562 5 |
| 2 048 | 11 | 0 000 488 281 25 |
| 4 096 | 12 | 0 000 244 140 625 |
| 8 192 | 13 | 0 000 122 070 312 5 |
| 16 384 | 14 | 0 000 061 035 156 25 |
| 32 768 | 15 | 0 000 030 517 578 125 |
| 65 536 | 16 | 0 000 015 258 789 062 5 |
| 131 072 | 17 | 0 000 007 629 394 531 25 |
| 262 144 | 18 | 0 000 003 814 697 265 625 |
| 524 288 | 19 | 0 000 001 907 348 632 812 5 |
| 1 048 576 | 20 | 0 000 000 953 674 316 406 25 |
| 2 097 152 | 21 | 0 000 000 476 837 158 203 125 |
| 4 194 304 | 22 | 0 000 000 238 418 579 101 562 5 |
| 8 388 608 | 23 | 0 000 000 119 209 289 550 781 25 |
| 16 777 216 | 24 | 0 000 000 059 604 644 775 390 625 |
| 33 554 432 | 25 | 0 000 000 029 802 322 387 695 312 5 |
| 67 108 864 | 26 | 0 000 000 014 901 161 193 847 656 25 |
| 134 217 728 | 27 | 0 000 000 007 450 580 596 923 828 125 |
| 268 435 456 | 28 | 0 000 000 003 725 290 298 461 914 062 5 |
| 536 870 912 | 29 | 0 000 000 001 862 645 149 230 957 031 25 |
| 1 073 741 824 | 30 | 0 000 000 000 931 322 574 615 478 515 625 |
| 2 147 483 648 | 31 | 0 000 000 000 465 661 287 307 739 257 812 5 |
| 4 294 967 296 | 32 | 0 000 000 000 232 830 643 653 869 628 906 25 |
| 8 589 934 592 | 33 | 0 000 000 000 116 415 321 826 934 814 453 125 |
| 17 179 869 184 | 34 | 0 000 000 000 058 207 660 913 467 407 226 562 5 |
| 34 359 738 368 | 35 | 0 000 000 000 029 103 830 456 733 703 613 281 25 |
| 68 719 476 736 | 36 | 0 000 000 000 014 551 915 228 366 851 806 640 625 |
| 137 438 953 472 | 37 | 0 000 000 000 007 275 957 614 183 425 903 320 312 5 |
| 274 877 906 944 | 38 | 0 000 000 000 003 637 978 807 091 712 951 660 156 25 |
| 549 755 813 888 | 39 | 0 000 000 000 001 818 989 403 545 856 475 830 078 125 |
| 1 099 511 627 776 | 40 | 0 000 000 000 000 909 494 701 772 928 237 915 039 062 5 |
| 2 199 023 255 552 | 41 | 0 000 000 000 000 454 747 350 886 464 118 957 519 531 25 |
| 4 398 046 511 104 | 42 | 0 000 000 000 000 227 373 675 443 232 059 478 759 765 625 |
| 8 796 093 022 208 | 43 | 0 000 000 000 000 113 686 837 721 616 029 739 379 882 812 5 |
| 17 592 186 044 416 | 44 | 0 000 000 000 000 056 843 418 860 808 014 869 689 941 406 25 |
| 35 184 372 088 832 | 45 | 0 000 000 000 000 028 421 709 430 404 007 434 844 970 703 125 |
| 70 368 744 177 664 | 46 | 0 000 000 000 000 014 210 854 715 202 003 717 422 485 351 562 5 |
| 140 737 488 355 328 | 47 | 0 000 000 000 000 007 105 427 357 601 001 858 711 242 675 781 25 |
| 281 474 976 710 656 | 48 | 0 000 000 000 000 003 552 713 678 800 500 929 355 621 337 890 625 |
| 562 949 953 421 312 | 49 | 0 000 000 000 000 001 776 356 839 400 250 464 677 810 668 945 312 5 |
| 1 125 899 906 842 624 | 50 | 0 000 000 000 000 000 888 178 419 700 125 232 338 905 334 472 656 25 |
| 2 251 799 813 685 248 | 51 | 0 000 000 000 000 000 444 089 209 850 062 616 169 452 667 236 328 125 |
| 4 503 599 627 370 496 | 52 | 0 000 000 000 000 000 222 044 604 925 031 308 084 726 333 618 164 062 5 |
| 9 007 199 254 740 992 | 53 | 0 000 000 000 000 000 111 022 302 462 515 654 042 363 166 809 082 031 25 |
| 18 014 398 509 481 984 | 54 | 0 000 000 000 000 000 055 511 151 231 257 827 021 181 583 404 541 015 625 |
| 36 028 797 018 963 968 | 55 | 0 000 000 000 000 000 027 755 575 615 628 913 510 590 791 702 270 507 812 5 |

Table A-4. Consolidated Coding Table

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| MEMORY REFERENCE INSTRUCTIONS | | | | | | | | | | | | | | | |
| D/I | AND | | 001 | 0 | Z/C | ← Memory Address → | | | | | | | | | |
| D/I | XOR | | 010 | 0 | Z/C | | | | | | | | | | |
| D/I | IOR | | 011 | 0 | Z/C | | | | | | | | | | |
| D/I | JSB | | 001 | 1 | Z/C | | | | | | | | | | |
| D/I | JMP | | 010 | 1 | Z/C | | | | | | | | | | |
| D/I | ISZ | | 011 | 1 | Z/C | | | | | | | | | | |
| D/I | AD* | | 100 | A/B | Z/C | | | | | | | | | | |
| D/I | CP* | | 101 | A/B | Z/C | | | | | | | | | | |
| D/I | LD* | | 110 | A/B | Z/C | | | | | | | | | | |
| D/I | ST* | | 111 | A/B | Z/C | | | | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| SHIFT-ROTATE GROUP INSTRUCTIONS | | | | | | | | | | | | | | | |
| 0 | SRG | | 000 | A/B | 0 | D/E | *LS | | 000 | †CLE | D/E | ‡SL* | *LS | | 000 |
| | | | | A/B | 0 | D/E | *RS | | 001 | | D/E | | *RS | | 001 |
| | | | | A/B | 0 | D/E | R*L | | 010 | | D/E | | R*L | | 010 |
| | | | | A/B | 0 | D/E | R*R | | 011 | | D/E | | R*R | | 011 |
| | | | | A/B | 0 | D/E | *LR | | 100 | | D/E | | *LR | | 100 |
| | | | | A/B | 0 | D/E | ER* | | 101 | | D/E | | ER* | | 101 |
| | | | | A/B | 0 | D/E | EL* | | 110 | | D/E | | EL* | | 110 |
| | | | | A/B | 0 | D/E | *LF | | 111 | | D/E | | *LF | | 111 |
| | | | | NOP | 000 | | | | 000 | | 000 | | | | 000 |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| ALTER-SKIP GROUP INSTRUCTIONS | | | | | | | | | | | | | | | |
| 0 | ASG | | 000 | A/B | 1 | CL* | 01 | CLE | 01 | SEZ | SS* | SL* | IN* | SZ* | RSS |
| | | | | A/B | 1 | CM* | 10 | CME | 10 | | | | | | |
| | | | | A/B | 1 | CC* | 11 | CCE | 11 | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| MAC AND INPUT/OUTPUT INSTRUCTIONS | | | | | | | | | | | | | | | |
| 1 | MAC | | 000 | A/B | 0 | | | | | | | | | | |
| 1 | IOG | | 000 | A/B | 1 | H/C | HLT | | 000 | ← Select Code → | | | | | |
| | | | | | 1 | 0 | STF | | 001 | | | | | | |
| | | | | | 1 | 1 | CLF | | 001 | | | | | | |
| | | | | | 1 | 0 | SFC | | 010 | | | | | | |
| | | | | | 1 | 0 | SFS | | 011 | | | | | | |
| | | | | A/B | 1 | H/C | MI* | | 100 | | | | | | |
| | | | | A/B | 1 | H/C | LI* | | 101 | | | | | | |
| | | | | A/B | 1 | H/C | OT* | | 110 | | | | | | |
| | | | | 0 | 1 | H/C | STC | | 111 | | | | | | |
| | | | | 1 | 1 | H/C | CLC | | 111 | | | | | | |
| | | | | | 1 | 0 | STO | | 001 | | 000 | | | 001 | |
| | | | | | 1 | 1 | CLO | | 001 | | 000 | | | 001 | |
| | | | | | 1 | H/C | SOC | | 010 | | 000 | | | 001 | |
| | | | | | 1 | H/C | SOS | | 011 | | 000 | | | 001 | |

Notes:
1) * = A or B. Use with bit 11 as 0 (A-Register) or 1 (B-Register).
2) D/I, A/B, Z/C, D/E, H/C coded: 0/1.
3) †CLE: Only this bit is required.
4) ‡SL*: Only this bit and bit 11 (A/B as applicable) are required.