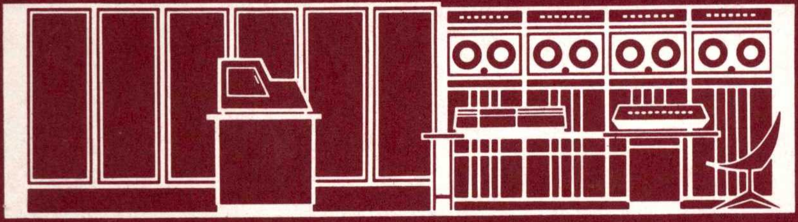
HONEYWELL 1800

PROGRAMMERS' REFERENCE MANUAL



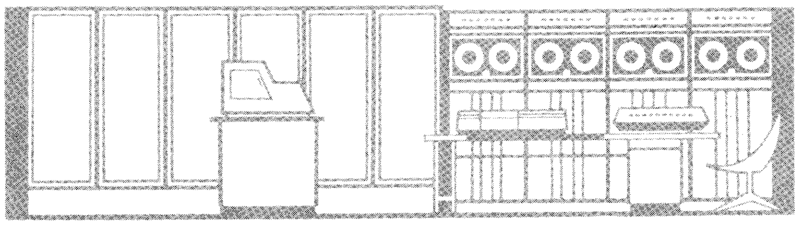
Honeywell

ELECTRONIC DATA PROCESSING

Litho in U.S.A.
NP—2822

DSI—170
2563
2664

HONEYWELL 1800
PROGRAMMERS' REFERENCE MANUAL



Honeywell
ELECTRONIC DATA PROCESSING

PRICE $4.00

Questions and comments regarding this manual should be addressed to

Honeywell Electronic Data Processing
Merchandising Division
60 Walnut Street
Wellesley Hills, Massachusetts 02181

HONEYWELL

FIRST EDITION
First Printing, May, 1963
Second Printing, June, 1964

FOREWORD

The purpose of the Programmers' Reference Manual is to define the internal machine language of the Honeywell 1800 Electronic Data Processing System and the manner in which that language is interpreted and manipulated by the system. This manual is not intended to be either an introduction to programming or an introduction to the Honeywell 1800. Instead, it is written as a handbook for the experienced programmer who has completed the Honeywell 1800 (or 800) programming course.

Throughout the manual the phrase "the performance of the system is unspecified" refers to instruction configurations whose results cannot be predicted from one execution to another.

NOTE

This is the second printing of the Programmers' Reference Manual. The principal differences between this and the first printing are as follows.

1. The discussion of input/output traffic control (pp. 18ff) has been clarified.
2. A new Appendix F discusses the extension of main memory beyond 32,768 words.
3. A new Appendix G describes the memory barricade feature.

TABLE OF CONTENTS

TABLE OF CONTENTS (cont)

TABLE OF CONTENTS (cont)

TABLE OF CONTENTS (cont)

LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS (cont)

SECTION I

INTRODUCTION

The Honeywell 1800 Electronic Data Processing System is designed to handle large-scale business and scientific applications. Its ability to deal efficiently with such applications derives from the provision of large main memory capacity (up to 65,536 twelve-digit words), high computational speed (e.g., 125,000 three-address additions per second), automatic parallel processing of up to eight independent programs, high-speed magnetic tape input and output (133,300 decimal digits/second per magnetic tape unit), and optional floating-point arithmetic. The system is fully compatible with the Honeywell 800 in programming logic, input/output capabilities, and general physical and environmental characteristics.

Programming Aids

Every Honeywell 1800 user is furnished with a complete automatic programming package which eliminates many of the routine human tasks involved in program preparation, checkout, and execution. This package includes the following compatible elements:

1. ARGUS, the H-800 assembly system which includes subroutine capabilities, sort and collate generators, and a program test facility for obtaining diagnostic information about a series of unchecked programs during automatic, full-speed execution.
2. COBOL, the COmmon Business Oriented Language which facilitates the interchangeability among various computers of programs prepared in a narrative language closely resembling everyday business english.
3. FACT, a powerful, English-language compiler developed especially to facilitate the programming of business applications for the H-800.
4. Automath, the scientific counterpart of COBOL and FACT. Automath interprets a series of problem statements in a FORTRAN-comparable arithmetic and logical notation and produces a complete machine-language program specifically directed to the solution of a program in the scientific area.
5. A library of thoroughly tested subroutines and macro routines for inclusion in compiled or assembled programs, and a program (LAMP) to maintain it.
6. ADMIRAL and Executive, two sophisticated monitoring systems which direct the operation of multi-program runs to facilitate maximum usage of the benefits of Honeywell automatic parallel processing.

The extent and power of this automatic programming package permit programs to be prepared for the Honeywell 1800 without reference to the language set forth in the Programmers' Reference Manual. In other words, the ARGUS Manual of Assembly Language, and the

COBOL, FACT, and Automath Compiler manuals are the standard programming documents for the Honeywell 1800. Nevertheless, this manual is offered for use by the sophisticated programmer who is eager to take advantage of the unusual and powerful features of the machine. When this level of experience is reached, an understanding of the internal configurations of instructions and data, as presented in the following sections, will prove invaluable.

Throughout the manual, the binary-digit structure of instructions and addresses is presented wherever it can enhance the explanation. Moreover, other numbering systems, such as octal and hexadecimal, are utilized wherever the subject matter requires them. For comparative purposes, the ARGUS format of many examples is also shown.

Fixed-Point Numbers

Most numbering systems in general use today are based on positional notation. This means that each system is based on a root number, called the radix, and that each position within a number represents a specific power of the radix of the particular system being used. Positive and negative powers of the radix are separated by an indicator point (the radix point), with the zeroth power of the radix appearing immediately to the left of the point. Positive powers of the radix appear in successive positions to the left and negative powers in successive positions to the right of the indicator point. The radix of a numbering system is equal to the number of digits comprising that system; these digits cover the range from zero to one less than the radix. Numbers written in positional notation are called fixed-point numbers.

Decimal System

The familiar decimal system is based on a radix of 10 and uses 10 digits from 0 through 9. Each position in a fixed-point decimal number represents a specific power of 10 and can have any of 10 values. The total value of a fixed-point decimal number is computed by multiplying the value of each digit by the positional value (power of 10) of its position within the number and then summing all of these products. For example, the decimal number 356. has the value 3×10^2 plus 5×10^1 plus 6×10^0, or 300 + 50 + 6. The number 3.56 has the value 3×10^0 plus 5×10^{-1} plus 6×10^{-2}, or $3 + \frac{5}{10} + \frac{6}{100}$. When positional notation is understood in the familiar decimal context, the interpretation of any other positional system becomes clear.

Binary System

This system is based on a radix of 2 and uses the two binary digits (or bits) 0 and 1. Binary numbers are the common internal system for digital computation due to the relative simplicity of recording, storing, and recognizing variables of only two values. The value of a fixed-point binary number is computed by multiplying the value of each digit by the corresponding power of 2 and summing all of the products. For example, the binary number 1001 has the value 1×2^3 plus 0×2^2 plus 0×2^1 plus 1×2^0, or 8 + 0 + 0 + 1, which equals 9. Where the system in use is not made clear by the context, its radix may be appended to the number as a subscript as, for example, to distinguish the binary number 1001_2 from the decimal number 1001_{10}.

Binary Codes

In addition to the use of "pure binary" numbers, as described in the preceding paragraph, binary digits may be grouped so that each group represents a decimal digit, alphabetic character, or other symbol. For example, bits may be manipulated in groups of four with each group representing a decimal digit (from 0000_2 to 1001_2). Similarly, groups of six bits may represent up to 64 digits, characters, or symbols. Such 4-bit and 6-bit codes are called "binary-coded decimal" and "alphanumeric," respectively. They facilitate the handling of the external decimal and alphabetic symbols by machine elements which recognize only variables of two values.

Octal and Hexadecimal Systems

The octal system and the hexadecimal system, based on radices of 8 and 16, respectively, are useful as shorthand methods of writing pure binary numbers. If a binary number is divided into groups of three bits, proceeding in either direction from the indicator point, each group may be replaced directly by its octal equivalent, since a 3-bit group has a total of eight possible values. If the same number is divided into 4-bit groups in the same manner, each group may be replaced directly by its hexadecimal equivalent, since a 4-bit group has a total of 16 possible values. The 16 hex digits are represented in this manual by the symbols 0 - 9 and B - G, respectively.

Floating-Point Numbers

It can be seen from the foregoing discussion that the value of a fixed-point number can be altered by moving the point. The decimal numbers 54321, 5.4321, and 5432.1 are all quantitatively different, although in every case the component digits and their ordering are identical. Moving the indicator point n positions to the right or left, respectively, increases or decreases the value of a number by a factor equal to the radix of the system raised to the power of n. When fixed-point numbers are used in scientific computations, the programmer must devote considerable attention to the scaling (i.e., the radix point positions) of his numbers and to the avoidance of overflow. The use of floating-point numbers in such computations allows the computer to handle scaling automatically and greatly reduces the problem of overflow.

In floating-point form, a number is expressed in terms of two other numbers, known as the mantissa and the exponent. The mantissa is composed of the sign of the fixed-point equivalent, plus the same component digits with an implied indicator point to the left of the high-order digit. The exponent is an integral power of the radix which specifies the true location of the indicator point relative to the point implied in the mantissa. If the true indicator point is actually at the point implied in the mantissa, the exponent is zero. If the true indicator point is to the right of the implied point, the exponent is positive; if it is to the left of the implied point, the exponent is negative. When the mantissa of a floating-point number is multiplied by the radix raised to the power of the exponent, the result is the equivalent fixed-point number. Note that numbers in any positional notation system, such as decimal, binary, octal, or hexadecimal, can be expressed in floating-point form.

A few examples should help to clarify the concept of floating-point numbers:

Fixed-Point Number	Sign, Mantissa, & Exponent (Floating-Point Number)			Application
Decimal:				
+95	+	.95	+2	$+.95 \times 10^{2} = +95$
-.030	-	.30	-1	$-.30 \times 10^{-1} = -.030$
Binary:				
+.001	+	.100	-2	$+.100 \times 2^{-2} = +.001$
-111.000	-	.111	+3	$-.111 \times 2^{3} = -111.000$

Note that high-order zeros in the fixed-point numbers are suppressed in the floating-point representations. This process, which results in the first significant digit being immediately to the right of the point, is called normalization. Normalization is discussed fully in Section XIII, in which floating-point arithmetic is explored in greater detail; the Honeywell 1800 instructions that manipulate data in floating-point form are also described in Section XIII.

SECTION II

THE HONEYWELL 1800 SYSTEM

The Equipment

A Honeywell 1800 Electronic Data Processing System consists of a central processor plus varying types and numbers of input and output units. The programmer must know the system configuration with which he is to work. An understanding of the function of each component and its relation to the entire system will make his task easier.

1. The Central Processor (1801)

The basic central processor consists of a control unit, a control or special-register memory, an arithmetic unit, and four banks of main (or high-speed) memory, each able to store 2048 Honeywell 1800 words. (A Honeywell 1800 word is composed of 48 information bits and six checking bits.) To this basic unit, additional memory banks can be added in modules of 8192 words (model 1802) up to 32,768 words; further expansion is available in modules of 16,384 words (model 1802-1) up to a maximum of 65,536 words. An optimal floating-point unit (1801-B, discussed below) is also available.

The control unit with its control memory is the nerve center of the central processor. As the site of traffic control and multiprogram control (explained below), it monitors the time sharing of the entire system to achieve maximum efficiency of operation. In addition to its multiplexing function, it is also the unit that selects, interprets, and directs the execution of instructions, and governs address selection in both control memory and main memory.

The memory cycle time is two microseconds. This is the time required to read one Honeywell 1800 word from memory (access) and to replace it in its original form (restoration).

The control memory is a magnetic-core array providing storage for 256 eighteen-bit words. The read-restore cycle of the control memory is out of phase with that of the main memory in such a way that if reference must be made to the control memory between references to main memory, it is usually possible to make such reference without loss of a main-memory cycle. As discussed more fully in Section V, the control memory contains eight identical groups of special registers such as sequencing counters, index registers,

registers used for indirect addressing, etc., the contents of which are used to select a full Honeywell 1800 word from the main memory. The offset cycle of control memory makes it possible to anticipate an address selection involving the contents of a special register and to prepare the address of a second operand while another unit is using the first operand. Because of this anticipatory technique, it is unnecessary in many cases to add memory cycles to an instruction for indexed or indirect addressing. Even when the contents of a special register are modified before they are restored, no extra memory cycle need be added, since the special register circuitry includes a separate adder, with complete and independent checking, used only for special register modification. This applies to both automatic modification, as when a sequencing counter is incremented after use, and program-controlled modification, as when an increment is specified in an address. However, while the two memory units are sufficiently out of phase to allow reading from the control memory prior to the start of a main memory cycle, a read-restore operation in which the result of an instruction is returned to a special register cannot overlap a main memory cycle; in this case, an extra cycle must be added to the instruction time.

The arithmetic unit is the portion of the central processor in which digits are combined to form new arrays in accordance with the logical rules of the command codes. The Honeywell 1800 central processor has provision for both binary and decimal arithmetic, complete logical abilities, and competent internal checking. For the interested reader, a complete description of the fixed-point addition logic can be found in Appendix A.

2. The Floating-Point Option (1801-B)

The floating-point option (1801-B) is a second control and arithmetic unit which provides the Honeywell 1800 user with 19 instructions that manipulate data in floating-point form, plus two fixed-point division instructions. The control portion of the unit selects, interprets, and directs the execution of these instructions by the arithmetic portion. In an 1800 system not equipped with an 1801-B, these instructions are not executed directly but are interpreted as pseudo instructions which call in library routines to perform the desired operations. Descriptions of this unit, its component registers, and the instructions it provides are presented in Section XIII.

3. The Console

The Honeywell 1800 console is basically a part of the central processor and is multiplexed into the system via multiprogram control (see below). A monitor typewriter is used by the operator to communicate directly with the central processor. Manual operations on the typewriter can start and stop individual programs and interrogate Honeywell 1800 storage. Under program control, the console typewriter can also print information useful to the operator. An additional typewriter, called the console slave typewriter, can be added to the system. No manual operations can be performed from the slave typewriter, but printing can be programmed to occur on either the slave or the console monitor typewriter. In addition to the typewriter(s), the console includes display lights that give the operator an at-a-glance summary of the number of active programs, their control centers, and their progress. The console also includes a modular display panel which has indicators and displays for monitoring the status of tape units and other peripheral devices.

4. Magnetic Tape Units (804) and Tape Controls (803)

The Honeywell 1800 magnetic tape units are designed for reliability and accuracy. The recording surface of the tape is untouched except by the read/record head, insuring that wear and damage to the surface are held to an absolute minimum. Vacuum is used to seat the reels on the hubs, to maintain loops in the loop chambers, and to provide contact with the capstans which cause the tape to move under the head, giving accurate control without the dangers inherent in mechanical techniques. The tape is edge guided along its entire path to protect the tape edges from damage and to avoid skew.

The system uses 3/4-inch-wide tape with Mylar[1] (polyester film) base and oxide coating. The recording portion of a full reel of magnetic tape is 2500 feet in length, ± 50 feet. A 30-foot clear leader precedes the recording portion and a 25-foot clear leader follows it. Information is written on tape in 10 longitudinal channels, eight for information bits from the word, one for a parity-checking bit, and one for a clocking indicator. One array of bits across the tape is called a frame. Information from six frames makes up the 54-bit word, which includes six parity bits. The clocking channel gives positive indication of the frame location on tape.

Variable-length recording is a basic feature of the Honeywell 1800, and records of any size may be read from or written on tape, although for control purposes a maximum size limit may be placed on records at the beginning or end of tape. Gaps between records are 0.67 inch long.

[1]Mylar is a registered trademark of E. I. du Pont de Nemours and Co., (Inc.).

Three different tape units are available for the Honeywell 1800. Their recording and rewind speeds, recording densities, and transfer rates are shown in the table below. All three tape systems are compatible: data recorded by one system can be read at recording density by the others.

Model	Speed (inches per second)		Recording Density (frames per inch)	Transfer Rate (decimal digits per second)
	Read/Write	Rewind		
804-1 Magnetic Tape Unit	120	360	400	96,000
804-2 High-Density Magnetic Tape Unit	120	360	555.5	133,300
804-3 Economy Magnetic Tape Unit	60	180	400	48,000

When rewinding, the tape moves at three times normal speed. A small photo-electric device in the tape unit senses the presence of edge "windows" (clear Mylar) in the tape to provide beginning-of-tape and end-of-tape indications for the programmer. A physical slot in each leader is used to negate the vacuum and stop the unit when the end of tape is reached.

When a metal ring is inserted in the front of a tape reel, writing is allowed to take place on the tape. When this ring is removed, the tape is protected and cannot be written upon. The ring may be installed or removed without rewinding the tape or removing the reel. There is, in addition, a manual switch on the tape unit panel which can be set to prohibit writing.

A tape control can control up to eight tape units. The model 803-1, 803-2, and 803-3 tape controls are used with 804-1, 804-2, and 804-3 tape units, respectively. Each control uses an input and an output channel so that one of the tape units attached to the control may be reading and another writing simultaneously. Designation of tape units as active or inactive is accomplished through the use of the tape address patchboard on the maintenance and indicator panel of the tape control. The patchboard also allows for the re-addressing of tape units; for example, the unit physically designated as number 1 may have, as a result of patchboard plugging, an effective address of 3.

5. <u>The Card Equipment</u>

a. Card Readers (823)

Two 80-column card readers are available with the Honeywell 1800

System. From the programmer's point of view, these two readers differ only in that the 823-1 reads 240 cards per minute and the 823-2 reads 650 cards per minute. In the manner in which they respond to instructions and transmit information, they are identical. Either may be used on-line (connected to the control processor via a card reader control) or off-line (connected to a tape unit via a card reader control and an off-line auxiliary control, see below).

In on-line operation, the card reader with its control reads the card, converts the punch configurations to Honeywell 1800 notation, and transmits the information to the main memory. By means of a 3-position switch, the operator selects the mode of conversion – alphanumeric (normal) or transcription. If the alphanumeric mode is used, card conversion results in the transmission of 10 Honeywell 1800 words of eight characters each, where the high-order character of the first word corresponds to column 1 of the card and each successive 6-bit character in the next lower-order position corresponds to the next higher-numbered column. In one position (FULL), the switch defines as legal the entire 64-code set listed in Table I, page 167, which shows the correspondence between punched-card codes and Honeywell 1800 codes; in a second position (STANDARD), the legally defined set is 50 characters, excluding the asterisked codes in Table I.

A third switch position (TRANSCRIBE) allows the processing of information punched in transcription mode, in which the information from each card read is transmitted to the main memory in 20 Honeywell 1800 words. The value of each of the 960 bits indicates the presence or absence of a punch in a specific punching position. Figure II-1 shows the correspondence between card format and memory format in both the alphanumeric and transcription modes.

Each of the card readers has two reading stations. The results of the two readings of each card are compared, and any discrepancy is noted. As the 80 columns of information are converted in the control, additional checking is done to insure correct conversion.

One extra word is appended to each card record as it is sent to memory, indicating the status of the error indicators at the completion of the reading and conversion operation for that card. Thus, a standard card record is either 11 words (alphanumeric mode) or 21 words (transcription mode). When cards are read on-line, the control word is composed entirely of zero bits with the exception of bits 15 and 16, which are used to indicate:

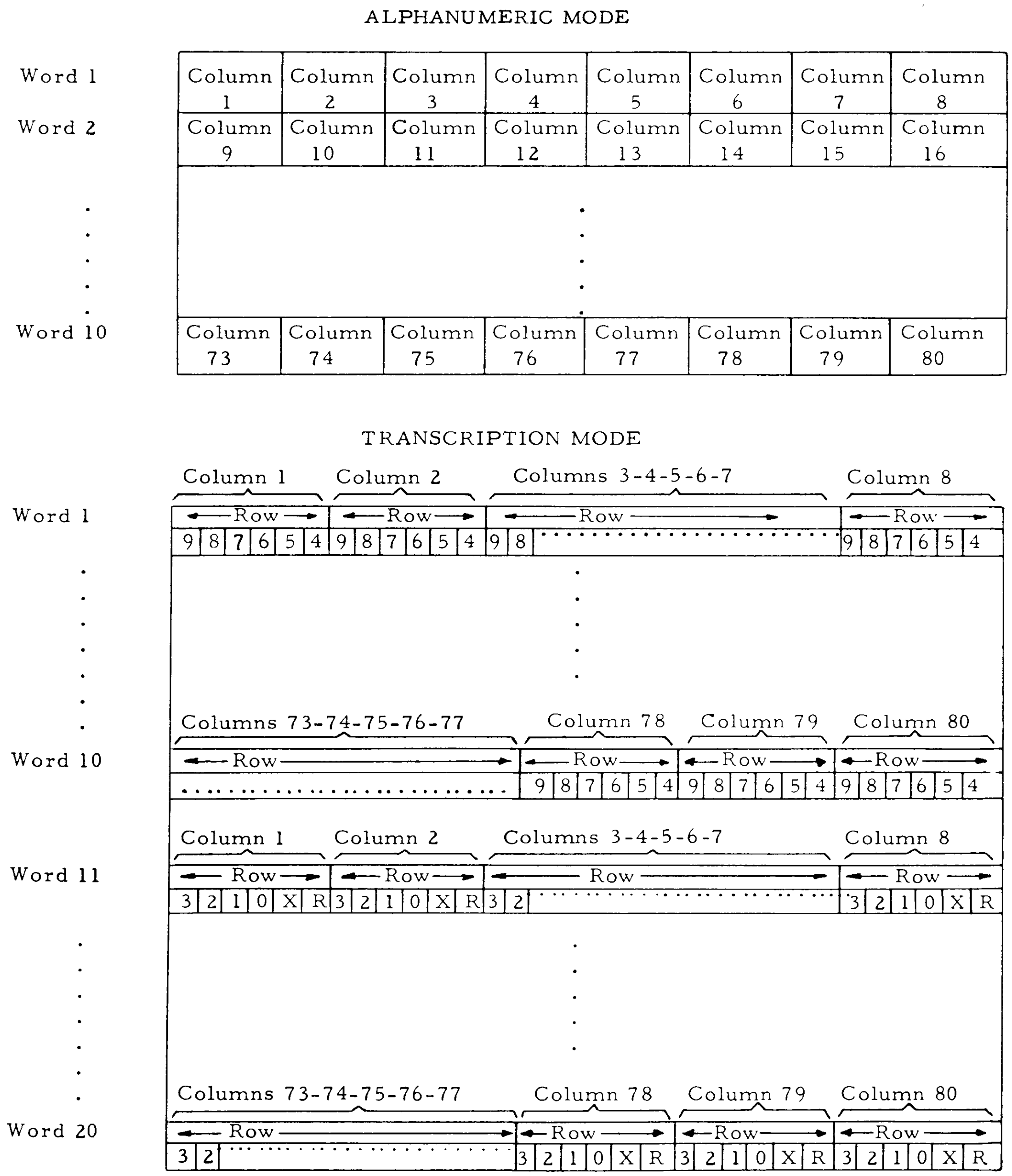

ALPHANUMERIC MODE
Word 1
Column 1, Column 2, Column 3, Column 4, Column 5, Column 6, Column 7, Column 8
Word 2
Column 9, Column 10, Column 11, Column 12, Column 13, Column 14, Column 15, Column 16
Word 10
Column 73, Column 74, Column 75, Column 76, Column 77, Column 78, Column 79, Column 80
TRANSCRIPTION MODE
Column 1, Column 2, Columns 3-4-5-6-7, Column 8
Word 1
Row
9 8 7 6 5 4
Columns 73-74-75-76-77, Column 78, Column 79, Column 80
Word 10
Word 11
3 2 1 0 X R
Word 20

Figure II-1. Conversion of Punch Positions to Honeywell 1800 Character Positions

1. That the card was punched, read, and converted correctly (11 in bits 15-16);
2. That the card contained a punch combination defined as illegal but was read and converted correctly (01 in bits 15-16);
3. That an error occurred during reading or conversion (10 in bits 15-16).

Note: In the event that an illegal punch coincides with an incorrect read or conversion, the latter signal takes precedence (i.e., bits 15-16 are 10).

When cards are read off-line, bits 23-26 of the control word are also coded to indicate the occurrence of any tape write errors. The contents of bits 23-26 normally alternate between four ones (1111) and four zeros (0000) in each successive record. However, when a tape write error is detected, the contents of bits 23-26 in the error record are repeated in the following record.

Since an anticipatory card reading technique is used (actually the card has been read and converted before the read instruction is received by the card reader), all errors except detection of a parity failure on transmission to the central processor must be indicated by the control word. Parity failure on transmission is indicated by an automatic transfer of control at the time of the next read instruction (see Section XI).

b. Card Punch (824-1)

The 824-1 punches 100 cards per minute, operating either on-line or off-line. The card punch is capable of punching in the same two modes in which the card readers can operate, and the relationship between the card format and the format of information within the Honeywell 800 is identical (see Figure II-1). The selection of the mode to be used in punching an individual card, however, is made by the program. A control word precedes the actual information to be punched, sets the punch to accept either 10 or 20 information words, and informs the punch control as to which conversion mode is desired. Only four bits of the control word are used by the punch control. The values of bits 16 and 17 specify the desired conversion mode and hence the number of words to be accepted. If the control word contains zeros in both bit 16 and bit 17, the alphanumeric mode is used; if these bit positions are both ones, the transcription mode is used. A zero-one or one-zero combination is illegal, and can cause the punch to enter either mode. Ones in bit positions 13 and 14 cause the punch to stop with an indicator lit, signifying that the punching of a file is completed. The control word is coded identically for both on-line and off-line operation.

c. Card Reader — Card Punch (827-1)

The card reader — card punch, model 827-1, reads 800 cards per minute and punches 250 cards per minute. Otherwise, it has all of the properties of the card readers and punch described above, including the capability for either on-line or off-line operation. The 827-1 requires separate peripheral controls (or a single model 811 multiple terminal unit control, see page 21) to handle the reader and punch sections. This unit is capable of simultaneous reading and punching, except when connected to a multiple terminal unit control.

6. The Printers (822)

a. Standard-Speed Printers

The standard-speed printers, models 822-1 and 822-2, are 407 and 408 tabulating machines, printing 150 lines per minute. Both printers can operate either on-line or off-line. Ten characters are printed to the inch over a spread of 12 inches, providing a total of 120 characters per line. Vertical spacing is six or eight lines to the inch. Forty-seven different characters are available: 26 alphabetic, 10 numeric, and 11 special symbols.

A 12-channel, punched paper carriage tape is used for vertical format control. In on-line operation, the central processor transmits a control word (containing vertical format information) and 15 information words to the printer control. The information words are treated as 120 alphanumeric characters, with the high-order character of word one corresponding to the leftmost printing position. The vertical format control word contains two 6-bit characters (bits 19-30) which correspond to the 12 channels of the carriage tape. A one in any of these bit positions designates a controlling carriage-tape channel, and the carriage is advanced until the next punch is sensed in the designated channel. By means of plugboard wiring, these 12 bits may be used to effect most of the normal plugboard controls, such as single or double spacing, suppressed spacing, extra spacing, overflow, non-print, skip, head of form, etc. In addition, ones in bits 13 and 14 of the control word cause the printer to stop with an indicator lit, signifying that the printing run is completed. The coding of the control word is identical for both on-line and off-line operation. Values of bit positions other than those specified are ignored.

The tabulators retain most of the abilities normally provided through plugboard wiring. In parituclar, the comparison abilities of the 408 are retained. The 822-2 (bill-feed printer) can print on cut forms as well as on continuous forms.

b. High-Speed Printer

The high-speed printer, model 822-3, has a rated speed of 900 lines per minute for continuous single-space printing and approximately 800 lines per minute for continuous double-space printing. Printing can be accomplished either on-line or off-line. There are 160 print positions of which any prescribed array of up to 120 can be activated during a run. At each print position, a total of 56 characters is available: 26 alphabetic, 10 numeric, and 20 special symbols in either standard or selfchek #12F font.

Horizontal spacing is 10 characters to the inch. Vertical spacing is six lines to the inch; a special option allows the selection of either six or eight lines to the inch by means of a switch. Skipping speed in the non-printing mode is approximately 20 inches per second. Vertical format is specified by a control word in conjunction with a pre-punched, 6-channel, paper carriage tape. Both horizontal and vertical vernier adjustment of form position are standard.

For each printed line, 16 words are sent to the printer control. The first word contains vertical-format information, and the remaining 15 are interpreted as 120 characters to be printed. The high-order character in the first information word corresponds to the first of the 120 print positions selected to be active. Line spacing is directed by the vertical-format word as follows:

Bit 1	Position at head of form, as indicated by a punch in channel 4 of the carriage tape, if bit 1 is a one.
Bit 6	Inspect channel 3 of the carriage tape for punch indicating end of form. If bit 6 is a one, the macine senses for end of form as determined by a punch in channel 3 of the paper tape. When the punch is sensed, the carriage is advanced to beginning of form. Normally, in a listing, bit 6 always has the value of one, so that the end-of-form channel is continually examined.
Bits 7-12	Advance paper the number of lines (0-63) indicated by the contents of bits 7-12 after the accompanying line has been printed.
Bits 13-14	Stop for end of run. If bits 13 and 14 are ones, the printer stops with the END RUN light glowing to signal that the run is complete.

Values of bit positions other than those described above are ignored, making it possible to edit a record for listing on either a standard-speed or a high-speed printer. The coding of the control word is identical for both on-line and off-line printing.

7. The Paper Tape Equipment

a. Paper Tape Reader and Control (809)

The model 809 paper tape reader and control operates on-line in the Honeywell 1800 System, reading punched paper tape into the central processor. The input tape must be chadded (clean hole), opaque, non-oiled, and non-metallic, although a special option permits the use of metallic paper tape. Three sizes of tape can be used with the reader: 11/16 inch, five-channel tape; 7/8 inch, six- or seven-channel tape; and one-inch, eight-channel tape. The tape may be read under reel-servo control or as strip segments. Two reel sizes are available: 5-1/2 inch reels which store 350 feet of tape, and seven-inch reels which store 700 feet, plus four feet of leader at each end.

Each read forward instruction causes one frame (character) to be read from paper tape into main memory. Under reel-servo control, a nominal tape reading speed of either 500 frames (50 inches) or 1000 frames (100 inches) per second may be selected by means of a switch; strip segments are always read at 500 frames per second. Rewind speed is either 50 or 100 inches per second, depending on the reading speed selected, and the tape can be wound onto either the supply reel or the takeup reel. During reading, frame punches are converted and stored as the low-order bits of a main memory location, with zeros supplied as high-order fill. If the tape has parity punches, the control checks frame parity; it can be set to handle either odd or even parity.

The logical end of tape is normally indicated by the presence of adhesive-backed pieces of metal foil. When the foil is not used, the end of tape may be detected by the presence of a particular punch pattern, by the absence of sprocket holes, or by the loss of tape tension.

b. Paper Tape Punches and Controls (810)

The models 810-1 and 810-2 paper tape punches and controls operate on-line with the 1801 central processor. Both models combine a control unit with a punch which punches ten frames to the inch at a speed of 110 frames per second.

The 810-1 handles 11/16 inch paper tape and punches 5-level code; the 810-2 can handle either 7/8 inch tape for punching up to seven levels, or 1-inch tape for punching up to eight levels. The punches will accommodate most kinds of commercially available paper tape, in reels of up to 1200 feet. Oiled tape is recommended for use wherever possible. Metallic paper tape can not be used. The punches produce one frame of chadded (clean hole) paper tape in response to each write.instruction from the central processor.

All bits of the frame to be punched, including parity bits (if any), must be stored in the main memory as the low-order bits of a word. Tape movement is anticipatory; after a frame has been punched, the tape is advanced one frame length in anticipation of the next write instruction. Bit 40 is used to signal end of run; unused bits from bit 33 to the first information bit must be zero-filled. The high-order 32 bits are not sensed and their values are irrelevant.

A standard parity check is used to verify the correct transfer of information from main memory to the punch control. The amount of tape on the supply and takeup reels is sensed by detectors which signal the central processor when the supply of tape drops below approximately 20 feet or the takeup reel becomes full.

8. Off-Line Controls (815, 816, 817, and 818)

When a terminal unit is used with a magnetic tape unit independently of the central processor, as in a card-to-tape or tape-to-printer operation, an off-line auxiliary control is placed between the peripheral control and the tape unit to provide control signals and power normally supplied by the central processor. If the terminal unit is a card reader, then an off-line input auxiliary control (816) must be used. In like manner, a printer or card punch requires an off-line output auxiliary control (815) and a multiple terminal unit control (see "System Configurations," below) calls for an off-line input/output auxiliary control (817).

For off-line printing or card punching, information is read from magnetic tape into the auxiliary control and stored frame-by-frame in the memory of the peripheral control. Reading occurs only in the forward direction of tape travel. When the peripheral control has received and stored the number of words required to process a record, it ceases to accept information. To accomplish the desired printing or punching the tape records should be organized in the following manner:

1. For printing, each record should contain one control word, 15 information words, and two orthotronic control words, in that order.
2. For alphanumeric mode card punching, each record should contain one control word, 10 information words, and two orthotronic control words, in that order.
3. For transcription mode card punching, each record should contain one control word, 20 information words, and two orthotronic control words, in that order.
4. The control word of the last record on a tape should contain an end-of-run code.

The end-of-record word generally appearing at the end of each tape record is not sensed in the off-line auxiliary control, which instead senses the interrecord gap and signals the magnetic tape unit to stop the tape. When the record has been correctly printed or punched, the next tape record is read into peripheral memory.

An alternative off-line printing configuration is available, using the model 818 off-line printer control to connect a high-speed printer and a magnetic tape unit. Thus, the 818 performs the functions of both the peripheral control and the off-line auxiliary control. In this configuration, unlike the use of an off-line auxiliary control, the printer is not available for on-line printing.

When reading cards off-line, each card is read, converted, and checked in the same manner as in on-line reading, then written on tape as a record. In alphanumeric mode, each record consists of 10 information words, one control word, two orthotronic control words, and one end-of-record word, in that order. In transcription mode, each record consists of 20 information words, one control word, two orthotronic control words, and one end-of-record word, in that order. The two orthotronic control words and the end-of-record word are automatically generated by the off-line auxiliary control.

9. Additional Equipment Available in the Honeywell 1800 System

a. Random Access Storage and Control (860)

The 860 series random access storage files and controls provide large-capacity auxiliary storage in the form of magnetic discs. Each disc stores 524,288 words or 6,291,456 decimal digits. The number of discs in a random access storage file ranges from twelve in model 860-1 to 192 in model 860-9.

b. Optical Scanning Unit and Control (840)

The model 840 optical scanning system enables documents printed in Selfchek #12F font by the high-speed printer to be used as on-line input to the 801 central processor. Honeywell has also developed an Orthoscanner, a device which optically reads pre-printed bar codes.

c. Communications Control (880)

The 880 communications control allows the 1801 central processor to receive and transmit data over toll or leased-wire circuits. With a communications control as part of the equipment complement, an 1800 system can communicate with a similarly equipped Honeywell system, and with a number of other systems. At any given instant, the communications control can either transmit or receive information.

d. Tape Control (836)

The model 836 tape control transmits data between an IBM type 729 II tape transport and the 1801 central processor.

e. Magnetic Tape Switching Unit (805)

The model 805 magnetic tape switching unit is used to switch control of one 804 magnetic tape unit from one to the other of two controls, which can be any combination of the following: 803 magnetic tape control, 815 off-line output auxiliary control, 816 off-line input auxiliary control, 817 off-line input/output auxiliary control, and 818 off-line printer control.

f. Real Time Control Units (812)

For real time operations Honeywell real time control units provide buffer storage and associated control equipment for buffering information into and out of the 1801 central processor, plus a specified interface to which external real time equipment may be designed. The model 812-1 control is an on-line device which transmits and accepts data non-simultaneously. The model 812-2 control, also an on-line device, transmits and accepts data simultaneously.

g. Programmed Elapsed Time Clock (1813-3)

The model 1813-3 elapsed time clock is connected to the 1801 central processor via either a peripheral input or output trunk. It stores a count of time, in seconds and sixtieths of a second, during which the central processor is in operation.

h. Programmed Real Time Clock (1813-4)

The 1813-4 real time clock is used to supply time-of-day information, in hours, minutes, and seconds, to the central processor and to a visual display.

i. Honeywell 1800 III

An H-1800 III system consists of an 1801 central processor and its normally associated peripheral devices, augmented by an H-200 computer which is directly connected via a model 212 on-line adapter. The adapter provides all of the buffer and indicator facilities necessary to allow any of a possible eight programs running in the larger system to control a program running in the H-200 and to initiate memory-to-memory data transfers between the two systems. Typically, the facilities provided by the 212 adapter are used to allow an H-1800 program to initiate and control: 1) simple input/output operations involving reading or writing by H-200 peripheral equipment and memory-to-memory data transfers; and 2) semi-independent "macro" operations such as card-to-tape, tape-to-printer, and tape-to-card media conversions which are running in the H-200. Programming considerations for the H-1800 III are presented in the bulletin "Model 212 On-Line Adapter" (DSI-274).

Traffic Control

Traffic control is the Honeywell 1800 element which directs the time-sharing of memory by the tape and peripheral units and the central processor. Peripheral buffer control is the element which reconciles the 6-microsecond buffer cycle of a peripheral control to the 2-microsecond H-1800 memory cycle. Multiprogram control is the element which directs the time-sharing of the central processor by the active program control centers. A clear concept of these elements is basic to the understanding of parallel processing and allowable system configurations and is the key to a thorough knowledge of the Honeywell 1800.

Traffic control has as its main object the efficient use of the entire system according to a set of priorities which derive directly from the nature of the equipment and are independent of the programs. For example, an 804-1 magnetic tape unit reading at full speed assembles one Honeywell 1800 word in a one-word buffer every 125 microseconds. If instant access is not provided to memory, a second word of buffer storage must be provided to retain this word. At the end of the next 125 microseconds, another word will have been read. If the first word has not yet been placed in main memory, another word of buffer storage must be provided. Since eventually one access to memory must be made for each word to be stored, it is obviously economic to store each word as it is assembled from tape and thus reduce the required buffer storage to a minimum. However, to keep the memory continuously available to the tape buffer during a read operation would be to introduce inefficiencies in the system, for only one memory cycle (two microseconds) is needed to store each word, and during the remaining 123 microseconds the entire system would be idle.

In the Honeywell 1800, one access to memory every 125 microseconds is guaranteed each tape unit. When a word is assembled from six frames for storage, a demand signal is generated by the buffer for one access to memory and is honored within 125 microseconds, clearing the buffer. In this case, only two words of storage are needed for each active tape unit, and the memory is utilized by the input/output operation only 2n microseconds out of each 125 microseconds, where n equals the number of active units. To achieve this time-sharing, traffic control monitors the demand signals from the buffers andarranges access to the memory within the prescribed time for each buffer demand.

The peripheral buffer control is an interface which makes the 6-microsecond buffer cycle of the peripheral controls compatible with the 2-microsecond memory cycle of the Honeywell 1800. Specifically, six microseconds are required to effect a 1-word transfer between the peripheral buffer control and a standard peripheral control unit. Only two microseconds, however, are required to transfer one word between the main memory and the peripheral buffer control. The two microseconds of main memory transfer are overlapped with the first two of the six microseconds needed for the transfer to the peripheral control unit.

As its name implies, traffic control monitors the transmission of information to and from the main memory. Its operation is represented schematically in Figure II-2. The 17 divisions of the band are called "stages" and one stage is assigned to each of the eight output channels, each of the eight input channels, and the central processor.

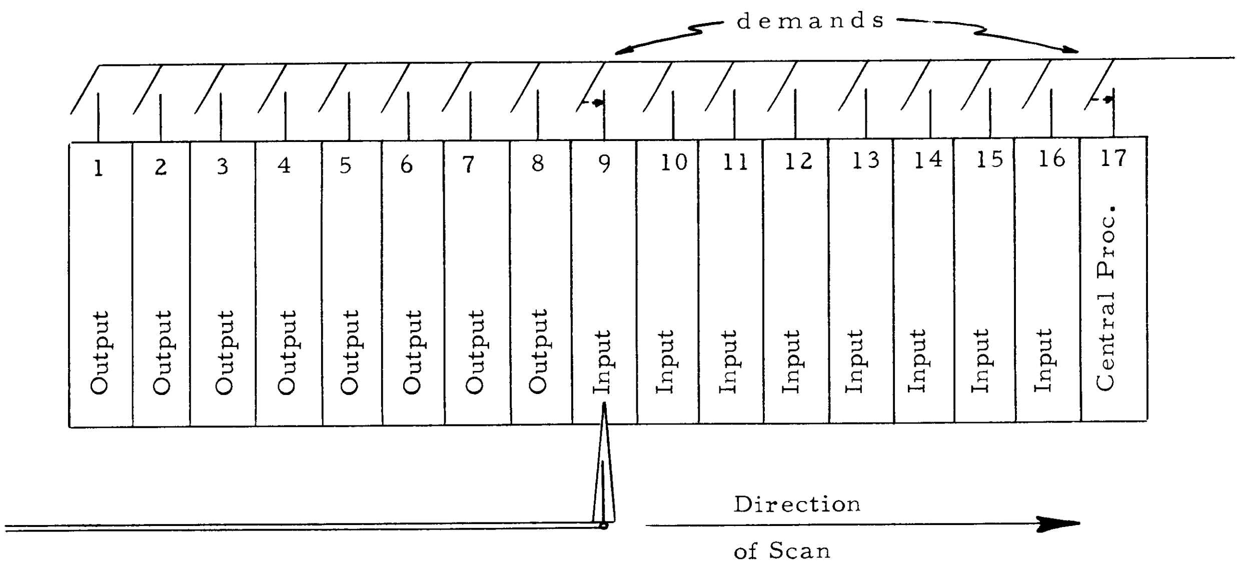


Figure II-2. Stages of Traffic Control

The creation of a demand signal by any device is represented in Figure II-2 by the closing of the switch shown in the corresponding control stage. When any program has been turned on in the central processor, the switch corresponding to the central processor stage is continuously closed. Traffic control begins each scan at the left end of the band. It proceeds to the right, ignoring all stages which show no demand signal, until a demand stage is reached. This stage is allowed access to the main memory for one memory cycle only. Traffic control then returns to the left end of the band to begin the next scan. Because the control search is anticipatory, no system time is consumed in bypassing stages in which no demand exists.

If no input/output devices are active but a program is running, the central processor stage will have complete use of the memory since each scan of the band will find no other demand signals. If a magnetic tape unit attached to channel 1 is reading, then once every 125 microseconds a demand signal will halt traffic control at the stage marked "input 9" for one memory cycle to allow transfer of the assembled word from buffer storage to memory. Should all 16 input/output stages issue demands simultaneously, a total of 96 microseconds (16 times 6) would elapse before all demands were satisfied, but the central processor would only be interrupted for 32 (16 times 2) microseconds.

Since, in normal read-write operations, only one buffer cycle may be allowed any stage before the next scan of the band, a maximum of 16 buffer cycles or 96 microseconds will elapse between successive interrogations of any stage. As this is within the 125 microsecond maximum, eight 804-1 tape units may be reading and eight may be writing simultaneously with central processor operations without conflict in demands for access to memory. Furthermore, since the memory is made available to the central processor for any cycle in which it cannot be utilized by an input/output stage, no idle time is introduced as long as any program is active. Thus, traffic control insures that the system responds to input/output device demands as required without introducing idle memory cycles, and that as long as any program can proceed, useful work is being done.

The rule of operation which states that only one buffer cycle may be allowed any stage before the next scan has one exception. When a magnetic tape unit is executing a distributed read or distributed write operation, two consecutive buffer cycles are allowed a stage for modifying the appropriate address counter on recognition of the demand signal accompanying an end-of-item word. Distributed read and write instructions are more fully discussed in Section XI. Two points in connection with this exception bear mentioning here. First, the difference between the normal 96-microsecond cycle and the maximum allowable 125-microsecond cycle allows servicing of four such demands without timing conflicts. Secondly, in the improbable case that all system channels are active and five or more channels are being used in distributed tape operations, and at least five of these each assemble an end-of-item word and create a demand signal within 125 microseconds of each other, an error signal will be generated for any unit whose demand is not serviced within the required time limit. This signal will appear to the program as a normal reading error signal indicating the need for a reread from the specified unit. Complete information is available to the program so that intelligent action can be taken under program control.

The preceding discussion of traffic control is based on the use of model 804-1 magnetic tape units which have an instantaneous transfer rate of 96,000 decimal digits per second. The use of input/output units with higher transfer rates may not permit simultaneous use of all 16 input/output channels. For example, the model 804-2 magnetic tape unit, with an instantaneous transfer rate of 133,300 decimal digits per second, requires one access to memory every 90 microseconds when operating at full speed.

All existing 1800 peripheral devices have 6-microsecond buffer cycles and are designed to be attached to peripheral buffer control. However, it is possible to attach special high-speed devices having 2-microsecond buffer cycles directly to traffic control. It is also possible, by means of a simple field change, to modify the traffic control priority of any input/output trunk(s) within the following restriction: a 6-microsecond read priority cannot be rewired to interrupt a sequence of 6-microsecond write priorities.

System Configurations

The central processor is the heart of any Honeywell 1800 installation. Peripheral devices and magnetic tape units are attached to the eight input and eight output channels of the central processor. Input and output channels alike are numbered 1, 2, 3, 4, 5, 6, 7, and 0. The output channels are associated with stages 1-8 of traffic control; the input channels are associated with stages 9-16. Each channel has a priority according to the sequence in which the corresponding stage of traffic control is interrogated. In other words, output channel 1 has the highest priority and input channel 0 the lowest. Each special register group includes an input buffer counter and an output buffer counter associated, respectively, with an input and an output channel. Thus output and input channels 1 are associated with the buffer counters of special register group 1, while output and input channels 0 are associated with the buffer counters of special register group 0. There exists no relationship between the special register group whose buffer counters are associated with a particular device and the special register group controlling the program which uses that device.

Any device requiring both an input and an output channel for simultaneous use, such as a tape control, must be assigned to channels whose associated buffer counters reside in the same special register group. Otherwise, any input device may be assigned to any input channel and any output device to any output channel, subject only to the restriction that tape controls must be assigned to channels of higher priority than peripheral controls. Specifically, if an installation includes two tape controls, a card reader, and two printers, the first tape control is assigned to output and input channels 1 and the second tape control to output and input channels 2. The card reader may be assigned to any remaining input channel and the printers to any remaining output channels.

Each tape control may control up to eight magnetic tape units. The tape units connected to a tape control are normally assigned consecutive positions starting with position 1. However, since tape addresses are assignable by patchboard, the programmer will find this no restriction.

In most systems each terminal unit will have its own control. The exception is the use of a model 811 multiple terminal unit control to control a card reader, a card punch, and a printer (or any two of these devices). The multiple control unit has a single buffer which can handle traffic in either direction. Thus, only one device, either input or output, may be used at a time. Selection of the device to be used is made by a manual switch on the multiple terminal unit control. If the multiple control unit has both a card reader and an output device attached, it must be assigned both an input and an output channel. Since only one device may be used at a time, however, the channels assigned need not be a corresponding pair.

Multiprogram Control

Multiprogram control directs the time-sharing of the central processor by the active programs. Each of the eight groups of special registers may direct the execution of an independent program. After a program is loaded, one of the special register groups is activated to direct the selection of instructions. This special register group, and the program it directs, is said to be "on." When the program is completed, the directing special register group is inactivated (turned "off"). Special register groups may be turned on and off independently of each other, either from the console or by program control.

Each time traffic control allows the central processor access to main memory, one memory cycle of one instruction is performed. If only one special register group is active, all cycles allowed the central processor are used in executing instructions from the active program. Since traffic control allows the central processor all available cycles except those needed to honor the intermittent demands of tape and peripheral devices, this case represents that of the conventional single-program computing machine with the ability to implement input/output operations simultaneously with computing.

When more than one special register group becomes active, central processor cycles must be shared among the several programs. The rules under which multiprogram control operates are as follows:

1. When an active special register group is selected, the next cycle allowed the central processor must be used to select the next instruction to be executed in the program controlled by that group.
2. All succeeding central processor cycles must be devoted to the execution of this instruction until it is completed. (This may range from two cycles upward.)
3. If the instruction is one which does not leave unstored information in the arithmetic or control units, its completion causes multiprogram control to scan or "hunt" for the next active special register group in sequence.
4. If the instruction does leave unstored information in the arithmetic or control unit, scanning or hunting is inhibited and the next instruction is selected from the same program. Hunting is not allowed if:
   a. the instruction generates a two-word result, of which only one word is stored in memory as the result of the instruction execution, e.g., multiply;
   b. the execution of the instruction results in a sequence change; in the case of a conditional change, hunting is inhibited only if the condition is satisfied;
   c. the instruction has an inactive C address;
   d. an unprogrammed transfer takes place as the result of executing the instruction;
   e. the instruction is capable of specifying that hunting shall not take place, and does so specify.

For example, if three programs are active, then one instruction is performed in turn from program 1, then 2, then 3, then 1, and so on, as long as all instructions selected allow hunting. Such alternation is temporarily held up if an instruction is selected which does not allow hunting, but it is resumed as soon as an instruction is selected which does allow hunting. Thus, the central processor cycles are shared on a fairly equal basis among all active programs.

The true power of multiprogram control appears when an active program attempts to execute an instruction which cannot be implemented because of the unavailability of a system component. Suppose, for example, that a write instruction calling for tape 4 is selected from an active program. The central processor, in attempting to execute this instruction, finds that either tape 4 or its associated output channel is involved in executing another instruction from the same or some other program. Seeing that the instruction cannot be executed immediately and recognizing the reason therefore, multiprogram control places the special register group in a "stall" condition. This condition indicates to multiprogram control that a) this program, although still active, shall not be allowed any central processor cycles as long as the "stall" indication remains, and b) when the channel and/or the device involved completes its present task, the stall condition shall be automatically removed and the program restored to its full active status.

Thus, when an instruction cannot proceed because of input/output conflicts with either the same or another program, the central processor cycles which it would have used are made available to the other active programs, causing them to proceed faster. The result of the operation of multiprogram control is that there is never an idle memory cycle in a Honeywell 1800 system as long as there is any active program in which an instruction can be executed.

If more than one active program is stalled because of input/output conflict, multiprogram control remembers which program was stalled first and operates on a "first off - first on" basis. Other stalled programs are activated according to the normal scanning sequence.

Although it need not concern the programmer, the reader may be interested in a short discussion of the actual machine procedure in case of program "stalls." The computer will have selected the instruction and entered the second cycle of execution before the unavailability of the input or output channel and/or device is discovered. No memory alteration will have been made, but the sequencing counter will have been advanced by one. When the stall condition is ascertained, the computer will subtract one from the sequencing counter and deactivate the program from multiprogram control. Each time that any input/output channel or device terminates an operation, all programs in a stalled state are reactivated. Multiprogram control looks again at each reactivated program, replacing in a stalled condition those programs whose input/output demands still conflict.

Multiprogram control also receives demand signals generated by the console and by inquiry stations. The console is regarded by multiprogram control as a ninth group of special registers, except that it takes precedence over any active program in the assignment of available central processor time. A demand signal from the console is serviced at the time of the next hunt, although a non-hunting sequence of instructions will not be interrupted.

When a console demand is recognized, the computer generates an instruction to implement the activity indicated by the console command. Sufficient memory cycles are then assigned to execute this generated instruction in the same fashion as if it had been selected from memory. When the instruction has been executed, the normal hunting process is resumed. If the system includes one or more remote inquiry stations, demand signals from these stations are recognized and implemented in a similar fashion. This technique allows the operator to communicate manually with the central processor without stopping the computer, and thus allows the system full-efficiency operation even during manual manipulation on one program.

Orthotronic Control and Checking

Orthotronic control is a powerful technique, exclusive with Honeywell, which insures against loss of information from magnetic tape during writing, storage, or subsequent reading. Experienced data processing personnel know that long storage periods or inept operator handling can cause information to disappear from a tape even though the accuracy of the record was checked at the time the record was written. Even infrequent occurrences of this type can result in many man-hours and machine-hours spent in re-creation of the records. While no technique will ever completely eliminate information loss, the high reliability and accuracy of the Honeywell 804 tape units, plus the presence of orthotronic control as a standard feature of every Honeywell 1800 system, insure that such loss is eliminated as a practical problem.

Orthotronic control is based on studies of the types and extent of information losses which have occurred on magnetic tape systems. It is partly automatic and partly program-controlled. An instruction is provided which automatically creates two orthotronic words for a specified record. These words are a logical combination of all the words in the record such that only a highly unlikely periodicity of error can go undetected and uncorrected. The orthotronic words are automatically positioned to accompany the record as it is written. Read and write instructions assume the presence of the orthowords and automatically include them in the record, using them in an automatic first-level check of the correctness of the information handled. The instruction which generates the original orthowords may also be used to reconstruct missing information if loss is detected. A full discussion of orthotronic control can be found in Appendix B.

Orthotronic control is a checking device peculiar to the magnetic tape units. In on-line operation, the central processor must be programmed to reconstruct lost data from a garbled record. When tapes are read or written in an off-line configuration, however, an off-line auxiliary control provides, in part, the central processor function. The off-line input auxiliary control automatically generates the two orthowords which must accompany each record when it is read by the central processor, and performs the first-level check on the information which involves these two words. The off-line output auxiliary control performs a check of every record read exactly as the central processor does, and is capable not only of detecting an error in the record but, in the majority of cases, of reconstructing the garbled information. The corrected information can then be printed or punched without stopping or repositioning the output device as would be necessary without such automatic error correction.

In addition to orthotronic control, and in some ways complementing it, a parity bit is written on tape accompanying each frame. The parity bit is read from tape together with the eight information bits of the frame and remains with these bits as frames are collected to form words. As each word of six frames is transmitted to memory, the accompanying parity bits are monitored to insure an error-free transmission. Each time a word is sent to or from main memory, a transmission check is performed using these six parity bits, and when the word is again written on tape, each bit accompanies its corresponding frame.

When a word is brought to the arithmetic unit, the computer generates a modulo-3 check on each frame pair (16 information bits) for use in checking arithmetic operations. The value of the 2-bit mod-3 check digit is the remainder (a value from 0 to 3, where either 0 or 3 may represent no remainder) which results when the decimal equivalent of the 16 bits is divided by 3. After the mod-3 check is generated, the parity bits are checked and then replaced by the six mod-3 checking bits. When arithmetic operations have been completed and the mod-3 check has been performed to insure that they have been completed correctly, the parity bits are again generated, replacing the mod-3 bits.

The control unit of the central processor checks the interpretation and execution of the program instructions. Selection of instructions and operand locations is checked. The checking process of an add instruction illustrates the thoroughness of the Honeywell 1800 checking system.

1. The selection of the instruction location is verified.
2. The instruction itself is verified for proper parity.
3. During the processing of the A address:

a. a mod-3 check group is attached to the address, then independently recalculated and compared with the original when this information is transferred to the memory selection circuits;

b. the selection operation is verified by comparing with the mod-3 check for the original address the special check digits delivered with the operand;

c. the operand itself is checked for proper parity when read from memory;

d. three mod-3 check digits associated with the operand are generated and stored.

4. During the processing of the B address (when the contents of the B address are added to those of the A address), steps a, b, and c are repeated for the B address. Also, three mod-3 check digits are generated as in 3d, but are added (mod-3) to the check digits previously stored.

5. As the result of the addition is transferred to memory, the C address memory selection is verified as in 3a, b, and c, and a new set of mod-3 check digits is formed from the computed sum and compared for equality with the check digits sum formed in (4) above. If the two sets of digits are equal, then the add instruction has been processed properly.

6. An example of the mod-3 arithmetic follows:

Number in A address	8426	9721	4075
The associated mod-3 check digits	2	1	1
Number in B address	1276	0216	4925
The mod-3 check digits	1	0	2
The sum of A and B is	9702	9937	9000
The mod-3 check digits	0	1	0
The mod-3 sum of the check digits is	0	1	0

In the general case where carries might occur between the operand groups, corrections of +1 and +2 are added to the appropriate check digits. Since two groups are simultaneously affected by a carry correction, any error in the addition, including the carry generation or correction process, is automatically detected.

Each of the special registers retains a mod-3 check on the 16 information bits it contains, which is used to check transmissions and arithmetic operations within the control unit. When the contents of a special register are transferred to the arithmetic unit or to main memory, they are expanded to full-word form and the mod-3 check is replaced by parity bits.

Card reading is checked not only for correct reading by the equipment, but, in the alphanumeric mode, for correctness of conversion and proper keypunching also. Card punching is checked on the 824-1 punch for double-punched and blank columns. Thirty columns of double-punch, blank-column detection are provided in the standard-speed punch; 80 columns are provided in the high-speed punch. The punch section of the 827-1 reads each punched card and checks its contents against the punch image. Printing is also checked by comparing echo pulses generated by the printer against the print image.

SECTION III

THE HONEYWELL 1800 WORD

The basic unit of information in the Honeywell 1800 System is a fixed-length word consisting of 54 binary digits, of which six are parity bits used by the automatic checking circuitry and 48 are information bits. Each main memory location is capable of storing one such word, and each arithmetic register is one word in length. A main memory word may represent a machine instruction or one or more pieces of data. In addition to the main memory, the central processor includes the control memory of 256 special registers, used primarily for control purposes and address modification. A special register has the capacity to store a partial word consisting of 16 information bits and two checking bits. The extension of the special register word to 24 bits to handle memory capacities in excess of 32,768 words is described in Appendix F.

The check bits of the main memory and special register words are not directly available to the programmer, nor are their values subject to program control. Subsequent discussions of the Honeywell 1800 word, therefore, will refer only to the information bits, unless otherwise noted.

Data Words

A computer program generally manipulates data in one or more different forms: decimal, alphanumeric, binary, or a combination of these. The Honeywell 1800 is capable of handling all these types of information. It may interpret the 48 bits of a word in groups of four for the purpose of binary-coded-decimal operation, in groups of six for alphanumeric operation, or as individual units of information for pure binary operation. It may also interpret the 48 bits as a mantissa and an exponent for floating-point operation. Figure III-1 illustrates the structures of these different words.

A decimal word in the Honeywell 1800 contains either 11 decimal digits with a sign, or 12 decimal digits without sign. The decimal arithmetic instructions interpret all operands as a sign and 11 digits. The sign consists of four bits which may represent either the sign of the entire word or individual, 1-bit signs for as many as four different pieces of information within the word. Although a positive sign is normally represented by four binary ones and a negative sign by four binary zeros, a non-standard configuration is perfectly acceptable as input to the arithmetic unit, which interprets any combination of bits except four binary zeros as a positive sign. The sign supplied with the result of an arithmetic operation, however, is always one of the two standard conventions, either four binary ones or four binary zeros. A more detailed discussion of sign conventions can be found in Section VI.

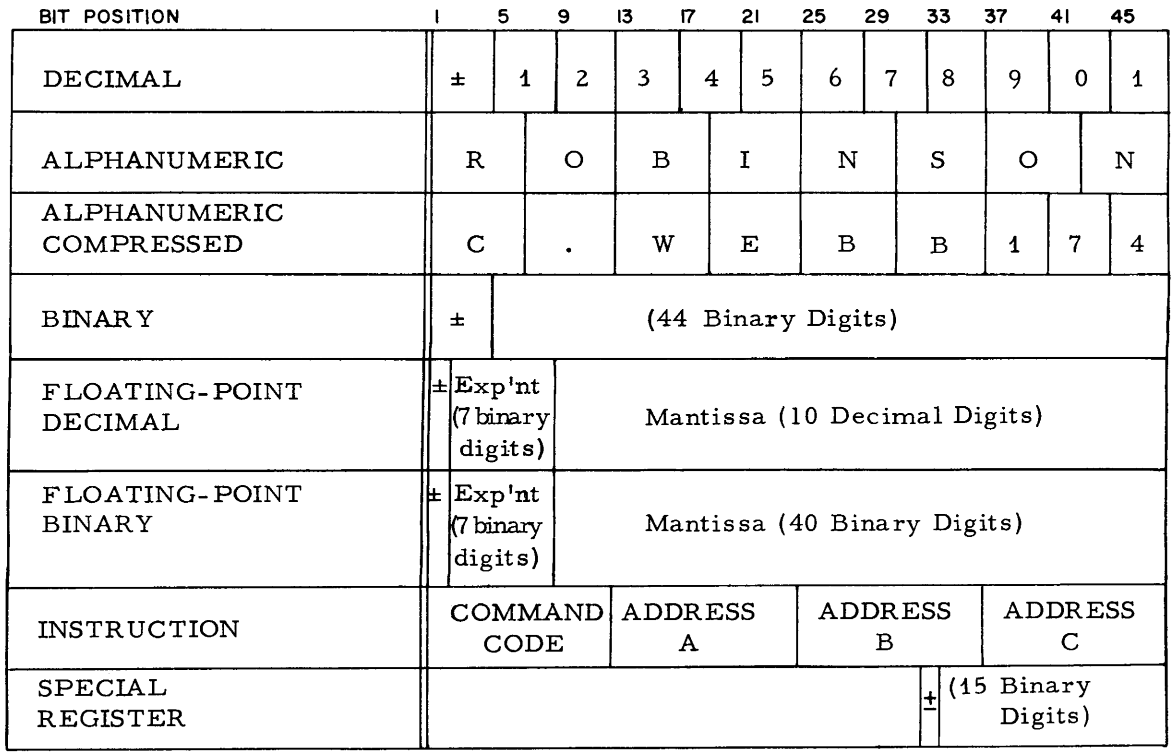


Figure III-1. Honeywell 1800 Word Structure

A Honeywell 1800 alphanumeric word comprises eight 6-bit groups. Each group can represent any of 26 alphabetic characters, 10 decimal digits, or 20 such special characters as punctuation marks, etc., (see Table I, page 167). Numbers may be stored in alphanumeric (6-bit) form, but the arithmetic unit cannot manipulate them as such; it handles numbers in pure binary or binary-coded decimal form. Between the central processor and the printers, information is transferred in the alphanumeric mode; between the central processor and the card equipment, information is transmitted in either the alphanumeric or the transcription mode.

The 48 binary digits of a word may also represent a pure binary number, which may be stored as a sign and 44 bits, or as 48 unsigned bits. With the exception of the instructions word add and word difference, which treat their operands as 48-bit unsigned numbers, the binary arithmetic instructions interpret operands as signed 44-bit numbers. The sign convention in binary arithmetic is identical to that described for decimal words.

The Honeywell 1800 word can also be handled as a floating-point number, composed of one-bit sign, seven-bit exponent, and 40-bit mantissa. The floating-point decimal word has an exponent that can represent a power of 10 from the -64th to the +63rd, and a mantissa that can represent a 10-digit number from .1000000000 through .9999999999 (when normalized). In floating-point binary form (represented in hexadecimal notation), the exponent can represent a power of 16 from the -64th to the +63rd, and the mantissa a 40-bit number from .00010000... 0000 through .1111...1111 (when normalized); the mantissa can represent the equivalent of approximately 12 decimal digits. A one in the sign bit position of the floating-point word indicates that the number is positive, and a zero that it is negative. Floating-point numbers are discussed in Section XIII.

The data words described above are identified in ARGUS language by the following constant codes: DEC, fixed-point decimal number (signed or unsigned); ALF, alphanumeric word; FXBIN, fixed-point binary number; M (mixed constant), compressed alphanumeric word; FLDEC, floating-point decimal number; and FLBIN, floating-point binary number. In addition, ARGUS recognizes an octal word identified by the constant code OCT. This word contains 16 unsigned or 15 signed octal digits. If 15 signed digits are specified, the most significant digit must be less than four, since a sign is represented by four bits, leaving only two bits for the high-order octal digit.

Several differences should be noted between ARGUS notation for data words and the format shown in Figure III-1. When ARGUS notation is used for decimal words, high-order zeros in signed decimal numbers and low-order zeros in unsigned decimal numbers need not be expressed. For example, ARGUS converts the number +125 to the signed 11-digit number +00000000125 and the unsigned number 32 to the 12-digit number 320000000000. A binary word in ARGUS notation is not expressed as a 44- or 48-bit binary number, but as the decimal equivalent of the desired information bits. Therefore, a binary word in ARGUS may contain up to 14 decimal digits and a sign. For complete details on the specification of data words in ARGUS language, reference should be made to the ARGUS Manual of Assembly Language.

Special Register Words

As previously noted, a special register can store 16 information bits, or one-third of a full Honeywell 1800 word. When these bits are manipulated within the special register circuitry, the high-order bit is interpreted as a sign (1 = plus, 0 = minus). Depending upon the type of addressing used, the remaining 15 bits of a special register word may be interpreted as a main memory address, consisting of a bank indicator and subaddress, or as a special register address, consisting of a group indicator and subaddress (see Figures IV-1 and IV-2). When a special register word is modified arithmetically within the special register circuitry, the value of the sign bit determines whether it is incremented or decremented. A special register word is identified in ARGUS language by the constant code SPEC.

Instruction Words

The 48 bits of a Honeywell 1800 instruction word are interpreted as four groups of 12 bits each. Bits 1-12 represent the command code; bits 13-24, 25-36, and 37-48 are designated as the A address group, B address group, and C address group, respectively. The address portions of instructions normally are used to designate the locations of operands and results, but in certain instructions they may contain special information such as the number of words to be moved, the number of bits to be shifted, a change of sequence counter, and so forth. A detailed discussion of addressing in the Honeywell 1800 will be found in Section IV.

Machine instructions fall into five major categories: general instructions, unmasked and masked; inherent mask instructions; peripheral and print instructions; simulator instructions; and scientific instructions. The masked general instructions and the peripheral and print instructions are uniquely designated by six-bits — bits 7 through 12 of the instruction word. The unmasked general instructions, the inherent mask instructions, and the scientific instructions are uniquely designated by eight bits — bits 7 through 12, plus bits 2 and 3. The simulator instructions are uniquely defined by only three bits — bits 10 through 12. These groups of bits which uniquely specify the operation to be performed are called the operation code. The bits of the command code which are not used for the operation code serve various other purposes which will be described as the instruction types are discussed. A graphic summary of the format of the major command code types appears in Figure III-2. The command codes for the individual instructions, together with their mnemonic operation codes in ARGUS language, are set forth by major instruction type in Table II, page 168.

General Instructions

General instructions include the arithmetic operations, logical operations, decisions, and information transfers. As noted in Table II, certain of these instructions may only be performed without masks; others may be performed either with or without masks. Regardless of masking, bit 1 of a general instruction command code, called the bisequence bit, always specifies the source of the next instruction (see Figure III-2). If bit 1 is a zero, the next instruction will be taken from the sequence counter; if bit 1 is a one, the next instruction will be selected from the cosequence counter. In ARGUS language, the source of the next instruction is specified in column 23 of the ARGUS input card by an "S" or blank for sequence counter or by a "C" for cosequence counter.

Unmasked General Instructions

Unmasked general operation codes are specified by command code bits 2, 3, and 7 through 12. Bits 4, 5, and 6 designate whether the A, B, and C addresses, respectively, refer to a main memory or a control memory location. If a memory designator bit is zero, then the corresponding address refers to main memory; a designator bit of one denotes a control memory address. In ARGUS language, the memory designator bit is not explicitly stated but is implied by the type of addressing used (see Section IV).

BITS	1	2	3	4	5	6	7	8	9	10	11	12
General Instructions Unmasked	S/C	Op Code		Memory Designator A	B	C	Operation Code					
General Instructions Masked	S/C	Partial Mask Address					Operation Code					
Inherent Mask Instructions	S/C	Op Code		Memory Designator A	B	C	Operation Code					
Peripheral Instructions	Peripheral Address: I/O Channel			Device			Operation Code					
Print Instructions	S/C	Irrelevant		Memory Designator A	B	C	Operation Code					
Scientific Instructions	S/C	Op Code		Memory Designator A	B	C	Operation Code					
Simulator Instructions	D/I	Remainder of Address								1	1	1

Address Used

Notes: S/C = Sequence or Cosequence Counter
I/O = Input or Output
D/I = Direct or Indexed

Figure III-2. Honeywell 1800 Major Instruction Types

Masked General Instructions

When general instructions are performed under the control of masks, they usually designate partial words as operands and results. For this reason, they are frequently referred to as "field" instructions. When a field instruction is performed, the same mask is applied to operands and result. Only those bit positions of the operands which correspond to binary ones in the mask word are used in the operation. The positions of the result location which do not correspond to binary ones in the mask are not altered by the operation. The location of the mask used in a field instruction is specified by bits 2 through 6 of the command code, in conjunction with bits 2 through 5 and 11 through 16 of a special register called the mask index register (MXR). A complete description of the way in which the five command code bits, called the partial mask address, are united with the ten bits of the MXR to designate the location of the mask will be found in Section V under the discussion of the mask index register.

In ARGUS language the location of the mask is specified by writing its symbolic tag in the command code field, following the operation code and separated from it by a comma. Thus, the instruction

```
        DS, MASK2    WAGES    DEDUCTNS    WEEKSPAY
```

is performed under control of the mask stored in the memory location assigned by ARGUS to the symbolic tag MASK2. Since field instructions use the memory designator bit positions in the partial mask address, it is impossible for these instructions to address control memory.

Inherent Mask Instructions

The use of masks is not restricted to field instructions, but extends to the inherent mask instructions. These instructions, which have the same command code format as the unmasked general category, include five shift instructions, a substitute, and an extract instruction. The chief distinction between the two types of masked instructions lies in the fact that the inherent mask instructions use bits from the B address group rather than from the command code to specify the location of the mask. For the shift instructions, the low-order six bits of the B address group are used in conjunction with bits 2 through 5 and 6 through 10 of the mask index register to locate the mask. In the substitute and extract instructions, the entire B address group is used to specify the location of the mask, without reference to the MXR.

A further difference between inherent mask and field instructions is that the latter always operate in the "protected" mode; in other words, the portions of the result location corresponding to binary zeros in the mask are preserved during the operation. The inherent mask group, on the other hand, includes three instructions which operate in the "unprotected" mode, in which the unmasked portions of the result location are cleared to zeros. In ARGUS language, the location of the mask for a shift instruction is specified in the same way as for field instructions: by writing its symbolic tag in the command code field following the operation code.

Peripheral and Print Instructions

Every instruction in the peripheral group performs some function involving a magnetic tape unit or a peripheral device. The high-order six bits of the peripheral instruction command codes are used to specify a magnetic tape or peripheral address. , Thus, these instructions cannot specify the source of the next instruction or address the control memory. The peripheral address bits are divided into two groups of three bits each. Bits 1 through 3 specify one of eight input or output channels (the operation code itself defines whether the channel is input or output) and bits 4 through 6 specify one of the devices attached to this channel. A more detailed explanation of the assignment of peripheral address bits will be found in Section XI.

The print instruction involves the use of the console typewriter. In this instruction, the high-order six bits of the command code are used as follows: bit 1 designates the sequence or cosequence counter as the source of the next instruction; bits 2 and 3 are irrelevant; and bits 4, 5, and 6 serve as A, B, and C address memory designators, respectively. Thus, this instruction can specify the source of the next instruction and address the control memory.

Several differences should be noted between the machine command code and ARGUS notation for these instructions. First, the peripheral command codes in ARGUS language are reversed in terms of machine language. In other words the mnemonic operation code is written first, followed by the device address expressed as an alphabetic code from AA to HH. Secondly, although there is but one machine instruction for the print function, ARGUS recognizes three distinct mnemonic codes to indicate alphanumeric (PRA), hexadecimal (PRD), or octal (PRO) print format. In machine language, the type of print format is specified by bits 5 and 6 of the B address group.

Simulator Instructions

Any instruction in which command code bits 10 through 12 are all ones is called a simulator instruction, since it permits the programmer to represent with a single instruction any function not built into the equipment logic, such as a machine instruction for some other data processing system. Each such instruction provides an entry to a simulator routine which is coded by the programmer and stored beginning with the next memory location after the address specified by command code bits 2 through 12. When the instruction is performed, it is transferred to the memory location specified by command code bits 2 through 12, the cosequence counter is set to the next higher address, and the next instruction is taken from the cosequence counter. If bit 1 of the command code is zero, bits 2 through 12 are interpreted as a main memory subaddress. If bit 1 is one, bits 2 through 12 are interpreted as a 3-bit index register designator and an 8-bit augmenter (see Section IV). The address portions of a simulator instruction have no assigned function and may be used to store parameters used by the simulator routine. The command code for an ARGUS simulator instruction is S, followed by a comma and an address designated by a symbolic tag or by an index register designator with an augmenter of seven.

Scientific Instructions

Eighteen of the twenty-one scientific instructions manipulate data in floating-point form, two provide for fixed-point decimal and binary division, and one converts data between fixed-point decimal form and floating-point binary form. In an 1800 system equipped with the 1801-B option, these instructions are executed directly; in a system that does not include an 1801-B, they are interpreted as pseudo instructions that call in library routines to perform the desired operations.

The operation codes for the scientific instructions are uniquely designated by eight bits in the command code field: bits 2, 3, and 7 through 12. Bit 1 is the bisequence bit, and bits 4, 5, and 6 are the memory designator bits referring to the A, B, and C address groups. Thus, these instructions can specify the source of the next instruction and address the control memory.

Special Words

Two special Honeywell 1800 words — the end-of-record word and the end-of-item word — deserve separate mention in this section. The end-of-record word is a word whose 48 information bits are:

1010 1010 0000 0000 1110 1110 1110 1110 1101 1101 1101 1101

This word is used to designate the end of a group of words constituting a single record. When records are being written on magnetic tape, the write operation stops only when an end-of-record word is sensed in memory. End-of-record words are automatically generated in the central processor during execution of compute orthocount and record transfer instructions. Their function will be detailed further as these instructions are discussed.

The end-of-item word is a word whose high-order 32 bits are identical to the high-order 32 bits of the end-of-record word. The low-order 16 bits are irrelevant for purposes of identification. As the name implies, an end-of-item word is used to designate the end of a group of words constituting a single item within a record. A record may contain an unspecified number of items, each of which is followed by an end-of-item word, or in the case of the last item, by an end-of-record word. End-of-item words are automatically generated in the central processor during execution of an item transfer instruction, while certain other instructions sense for these words during execution. Their function will be highlighted in the discussion of these instructions.

SECTION IV

ADDRESSING

The basic main memory of the Honeywell 1800 consists of four banks, each capable of storing 2048 words, giving a total basic capacity of 8192 words. Up to three model 1802 memory modules of 8192 words each can be added to the basic memory to expand the memory capacity to 16,384 words, 24,576 words, or 32,768 words.[1] Each main memory location is directly addressable and is uniquely designated by a 15-bit configuration. This array of bits may also be thought of as an 11-bit subaddress to specify, in binary, one of the 2048 locations in a bank and a 4-bit bank indicator to specify a memory bank. The bit structure of a main memory address is shown in Figure IV-1. (It is sometimes convenient to express the value of such a 15-bit array as five octal digits. In this notation, the memory addresses of a 32,768-word system range from 00000 to 77777.)

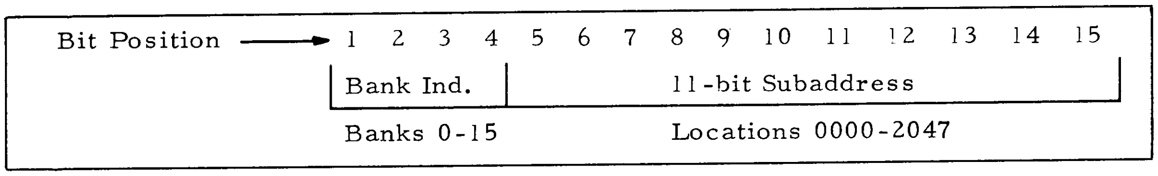


Figure IV-1. Main Memory Address

The control memory consists of 256 special registers divided into eight groups of 32 registers each. Every special register is directly addressable by a unique 8-bit configuration. This array of bits may also be thought of as a 3-bit group indicator designating one of the eight special register groups and a 5-bit subaddress specifying one of the 32 registers in a group (see Figure IV-2).

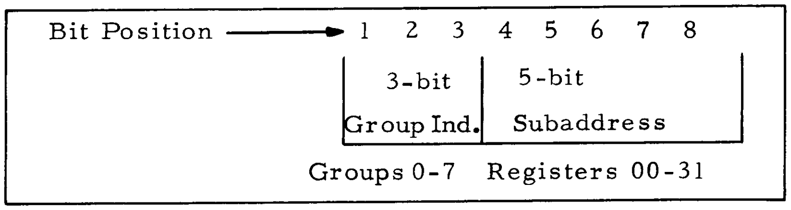


Figure IV-2. Special Register Address

Since both the main and control memories are directly addressable, some means must be provided to specify which memory is being addressed in each of the three address groups of an instruction. This is accomplished by defining one memory designator bit position in the command code of the instruction for each of the three address groups. A zero designator bit indicates that the respective address refers to main memory; a designator bit of one indicates that the address refers to a control memory location (special register). For those command

[1] Up to two model 1802-1 memory modules of 16,384 words each can be added to the three model 1802 modules to expand the memory to 49,152 or 65,536 words, as explained in Appendix F.

codes which do not provide the memory designator bit positions, designators of zero are always implied and the respective addresses are always interpreted as main memory addresses. During the execution of an instruction, the designator bit does not appear in the address selectors, but is stored in a separate unit of the control circuitry.

Since an instruction address includes 13 bits (12 bits in the address group plus a memory designator bit, explicit or implied), it does not precisely specify a complete main memory address (15 bits) or a complete control memory address (8 bits). The instruction address bits may be interpreted by the central processor in a number of ways to form a complete main or control memory address. For example, direct addressing is the explicit statement of the desired main or control memory subaddress in the instruction. Indexed addressing refers to the technique of augmenting a main or control memory address stored in an index register to form the desired address. Indirect addressing refers to the technique of stating the address of a special register in which the desired main memory address is stored. Main memory locations can be addressed in any of these ways; special registers are addressed either directly or by indexing.

As noted in Section III, the main memory may be addressed by all instructions in any of the five major categories. Special registers, on the other hand, may be addressed only by general unmasked instructions, inherent mask instructions, scientific instructions, and the print instruction, since these are the only types of instruction which provide for the memory designator bits in the command code.

Direct Memory Location Address

Each address group in a Honeywell 1800 instruction word consists of 12 binary digits. Bit 1 of this 12-bit configuration specifies whether the address is direct or indexed. If bit 1 is zero, the address is direct; if bit 1 is one, the address is indexed.

If bit 1 of an address group is zero (direct) and the corresponding memory designator bit is zero (main memory), then the remaining 11 bits of the address group are interpreted as a subaddress designating one of the 2048 locations in a bank of memory. The bank indicator stored in the sequencing counter[1] which selected the instruction is appended to this subaddress to form a complete 15-bit address (see Figure IV-3). Thus, every direct memory location address in an instruction always refers to the bank in which the instruction was stored.

[1] Although instructions are normally selected by a sequence or cosequence counter, they may be selected by an unprogrammed transfer register (see Section V). In this case, the bank indicator is taken from the unprogrammed transfer register to form a direct memory location address.

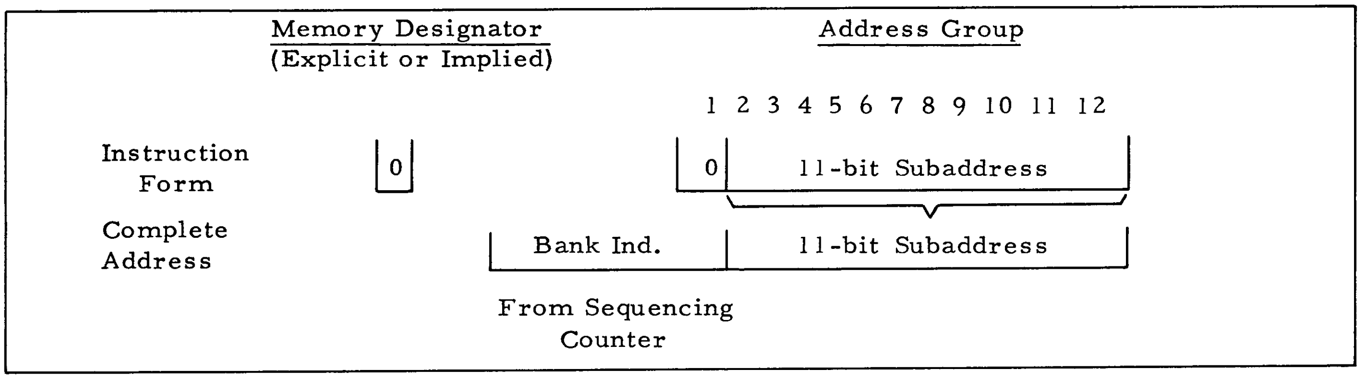


Figure IV-3. Direct Memory Location Address

In an ARGUS instruction, a direct memory location address is specified by a symbolic tag or by address arithmetic, in which the address is designated according to its relative position with reference to the instruction in which it appears or with reference to a symbolic tag. These three types of direct memory location addressing are illustrated in the ARGUS instruction:

```
DS    SALARY    C, +20    SALARY - 2.
```

SALARY is the symbolic tag of a main memory location; C, + 20 represents the location 20 after the location of the instruction itself; and SALARY - 2 represents the location two before that tagged SALARY.

Direct Special Register Address

If bit 1 of the address group is zero (direct) and the memory designator bit is one (control memory), bits 8-12 of the address group are interpreted as the subaddress of one of the 32 special registers in the group which includes the sequencing counter that selected the instruction. The central processor attaches to this subaddress the group indicator associated with the sequencing counter to form a complete 8-bit special register address (see Figure IV-2). If bit 7 of the address group (called the tabular bit) is zero, then the 8-bit array is interpreted as a direct special register address; that is, the specified register is used as an operand location or as a result location. Bits 2 through 6 of the address group specify an increment in the range 0 through 31 (in binary), which may be added, under control of the special register sign bit, to the low-order bits of the special register after use, thereby altering them permanently. If the special register sign bit is positive, the value of the increment is added to the contents of the special register, and the contents are said to be incremented. If the sign is negative, the value of the increment is subtracted from the contents of the special register and the contents are said to be decremented. Incrementing (or decrementing) always occurs when the special register is addressed as the source of an operand, never when the special register is addressed as a result location. The formation of a direct special register address is illustrated in Figure IV-4.

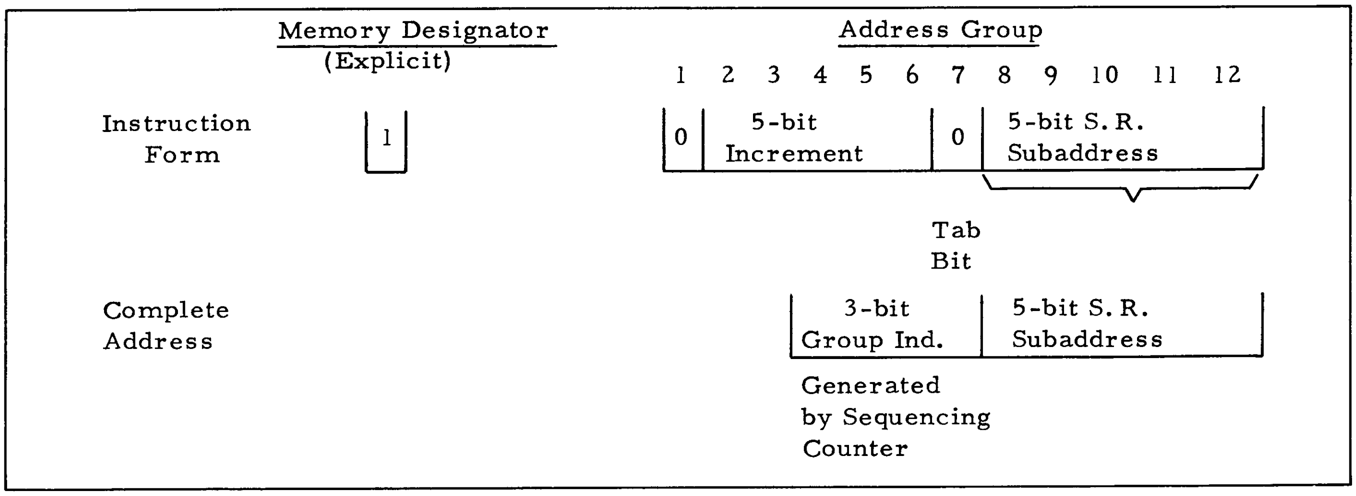


Figure IV-4. Direct Special Register Address

Since the group indicator attached to the special register subaddress is always that of the sequencing counter which selected the instruction, a direct special register address always refers to the special register group containing the sequencing counter which selected the instruction.

In ARGUS language, the direct address of a special register is indicated by the letter Z, followed by a special register designation and an unsigned increment from 0 to 31, all separated by commas. The letter Z, in effect, represents a one in the memory designator bit position of the command code, a zero in the first bit position of the address group, and a zero in the tabular bit position of the address group. The special register designation may be either the numeric subaddress or the mnemonic subaddress of the desired register, as shown in Figure V-1 (page 52). If the contents of the special register are not to be altered, the programmer may specify an increment of zero or may omit the increment entirely. The ARGUS address

```
Z,   R1,   10
```

indicates that general purpose register 1 is addressed directly as the source of an operand or as a result location; if addressed as an operand source, its contents are to be incremented (or decremented) by 10 after use. The address

```
Z,   X0
```

indicates that index register 0 is directly addressed and that no incrementing is to take place.

It should be noted that the relative positions of the special register subaddress and the increment are reversed in ARGUS language from their machine language arrangement. In other words, the increment appears in the low-order position of the ARGUS address, but in the high-order bits of the machine address group.

Indexed Memory Location Address

Each special register group includes eight index registers. An indexed address refers to one of these registers in the special register group of the sequencing counter which selected the instruction and is defined by a one in bit 1 of the address group. The remaining 11 bits of the address group are interpreted as an index register number and an augmenter to be added to the contents of that index register before use. Bits 2 through 4 designate one of the eight registers in the group, while bits 5 through 12 specify, in binary, a number from 0 to 255 which augments the low-order bits of the contents of the index register. (Note that an indexed address specifying index register 7 with an augmenter of 255 is interpreted as an inactive address, see page 47). It should be emphasized that in indexed addressing the index register is not identified by its 5-bit subaddress, but by only three bits which designate its position within the group of eight index registers. Whenever a special register is addressed in an instruction by its full 5-bit subaddress, it is said to be explicitly addressed. When it is referenced in any other way, it is said to be implicitly addressed or referenced. Since the index register in an indexed address is denoted by only three bits, an index register is always referenced implicitly in indexed addressing.

Like all special registers, an index register has the capacity to store 16 information bits of which bit 1 is a sign bit. If the memory designator bit in the command code of the instruction is zero (either explicit or implied), the low-order 15 bits stored in the referenced index register are interpreted as a bank indicator and an 11-bit main memory subaddress. When the instruction is performed, the 8-bit augmenter is added to the stored 15-bit address, under control of the index register sign, to form the desired main memory address. This process has no effect upon the contents of the index register, which retains the unaugmented address. The interpretation of an indexed memory location address is shown in Figure IV-5.

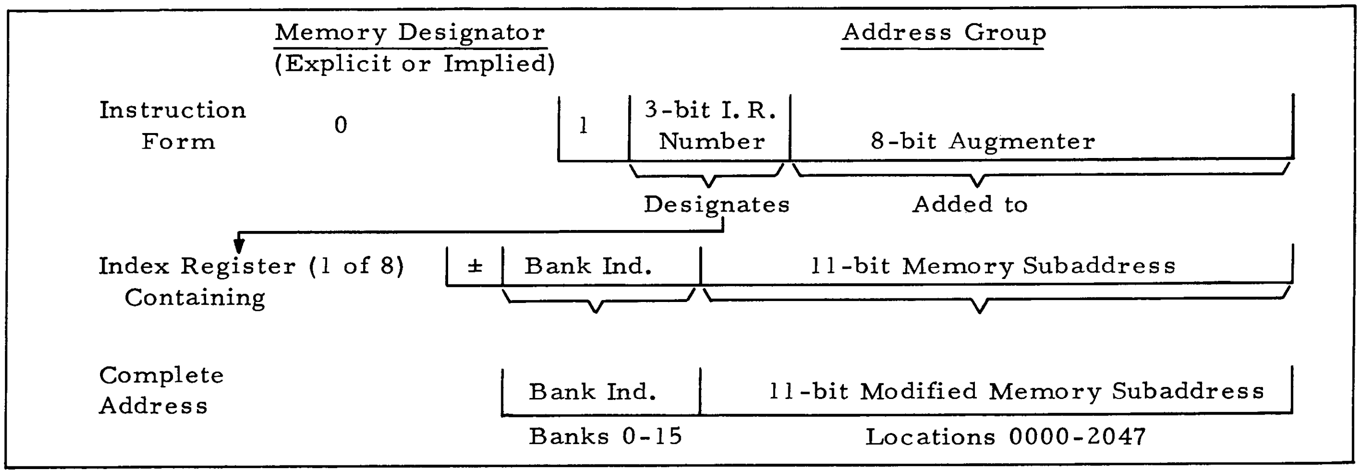


Figure IV-5. Indexed Memory Location Address

Since the index register contains a full 15-bit memory address, indexed addressing, unlike direct addressing, permits the programmer to address locations in any main memory bank, regardless of the bank indicator stored in the controlling sequencing counter. This type of addressing is also useful in processing multi-word items or in referring to a stored table, where the address of the first word of the item or table is stored in an index register and all references to the item or table are made using the index register with appropriate augmenter. It must be remembered, however, that positive augmentation occurs only if the index register sign is positive. If the sum of the augmenter plus the stored subaddress exceeds 2047, a carry occurs into the bank indicator, and the resulting address designates a location in a different bank from the address stored in the index register.

In ARGUS language, an indexed memory location address is indicated by writing an index register number (from 0 to 7) followed by a comma and a number from 0 to 255 or a symbolic tag to represent the augmenter. Thus the address

```
5,  10
```

specifies that the contents of index register 5 in the related special register group are to be augmented by 10 to form the complete memory address of an operand or result location. Reference should be made to the ARGUS Manual of Assembly Language for details on the use of symbolic tags to represent the augmenter.

Indexed Special Register Address

If bit 1 of the address group is one (indexed) and the memory designator bit is one (control memory), the address group is interpreted as an index register number and an augmenter, but the augmented contents of the referenced index register are interpreted as a special register address rather than a main memory address. When the augmenter has been added to the low-order eight bits of the index register, the resulting configuration is interpreted as shown in Figure IV-6.

Two facts illustrated by Figure IV-6 should be particularly noted. Since the augmented contents of the index register are interpreted as a special register address complete with group indicator, this type of addressing, unlike direct special register addressing, permits the programmer to address special registers in any group. As noted in Section V, this facility has particular significance in connection with access to certain special registers involved in reading and writing operations.

The second point involves the value of the tabular bit. Depending upon the original contents of the index register and the value of the augmenter, the tabular bit in the modified index register contents may be zero or one. If this bit is zero, then the special register is directly

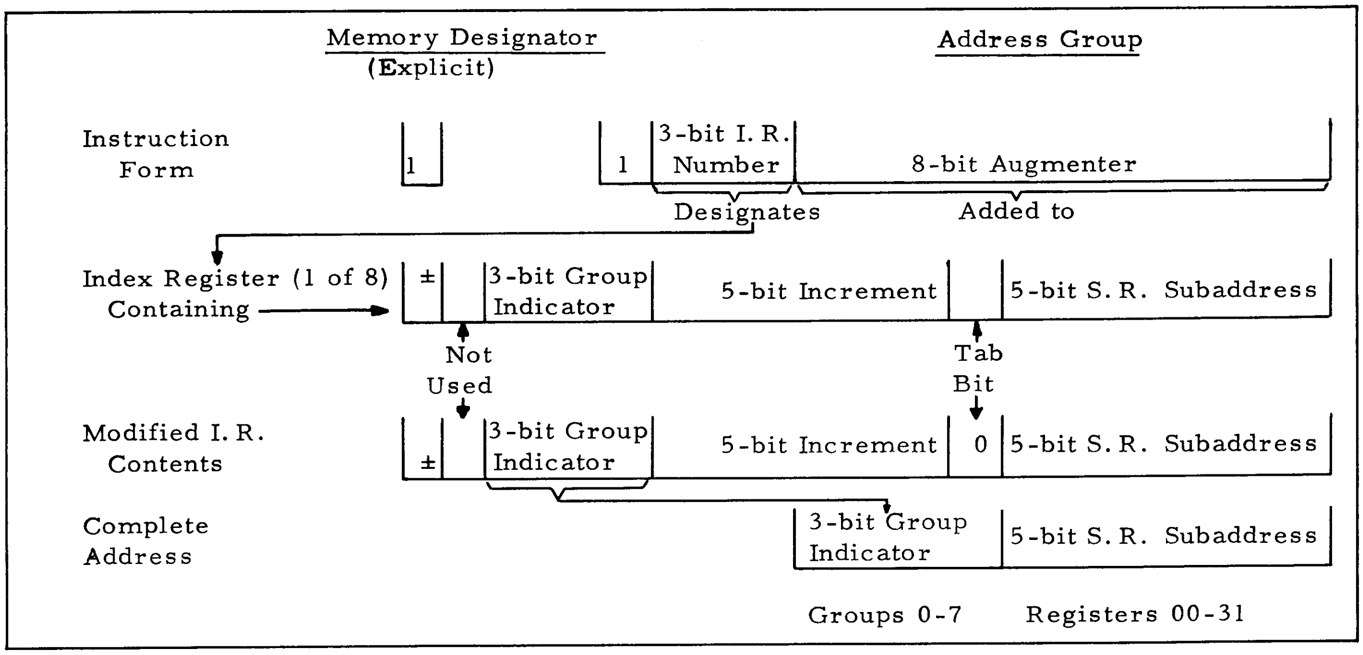


Figure IV-6. Indexed Special Register Address

addressed, as defined under the discussion of direct special register addressing. If the tab bit is one, on the other hand, the type of addressing is indirect, as described below under the discussion of indirect addressing. Regardless of the value of the tabular bit, the contents of the special register will be permanently modified, after use, by the value of the 5-bit increment, under control of the sign of the special register itself, provided that the special register is not addressed as a result location. As always, the contents of the index register are not altered by the indexing process.

In ARGUS language, an indexed special register address takes the form

Index Register Designator, Z, Special Register Designator, Increment.

The index register designator is a number from 0 to 7 which specifies one of the eight index registers related to the controlling sequencing counter. The special register designator may be a number from 0 to 31 or it may be mnemonic (see Figure V-1, page 52). The increment may be a number from 0 to 3 or it may be omitted. The manner in which these numbers are used to modify the index register contents and form a special register address is discussed in detail in the ARGUS Manual of Assembly Language.

Indirect Memory Location Address

In some instances, it is useful to be able to specify in the address group of the instruction the address of a special register where the main memory address of the desired operand is stored, rather than to specify the location of the operand directly. This method of locating an operand is called indirect memory location addressing.

In this type of addressing, the bit configuration of the address group is identical to that described under direct special register addressing with the exception that the tabular bit (bit 7) in the address group has the value of one rather than zero. Since the memory designator bit must also have the value of one (control memory), this type of addressing may be used only in unmasked general instructions, inherent mask instructions, scientific instructions, and print instructions. The special register whose address is generated, as shown in Figure IV-7, contains not the operand for the instruction, but the address of the operand in main memory.

The address generated is that of a special register in the same group as the sequencing counter which selected the instruction. The contents of this special register are interpreted as a sign, followed by a 4-bit bank indicator and an 11-bit subaddress designating the main memory location (in any bank) where the operand will be found. Thus, indirect memory location addressing, like indexed memory location addressing, provides access to operands in any bank of memory regardless of the bank indicator stored in the sequencing counter. After the contents of the special register have been used to locate the desired operand, the low-order bits of the stored contents are permanently modified by the increment specified in the address group of the instruction, under control of the special register sign. This incrementing (or decrementing) takes place regardless of whether the memory location addressed is an operand or a result location.

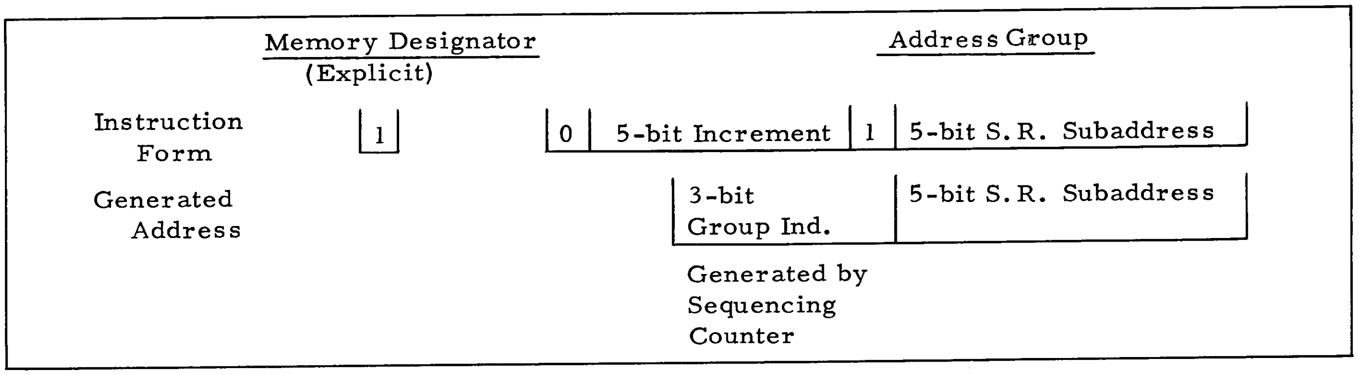


Figure IV-7. Indirect Memory Location Address

Indirect addressing is a useful tool for stepping sequentially through an array of items, processing the nth word of each item. The location of word n of the first item is stored in a special register. When this location is addressed indirectly, the use of the proper increment (equal to the item size) sets the contents of the special register to the location of word n of the second item, and so forth, until word n of each item has been processed.

ARGUS recognizes the use of indirect memory location addressing by the notation

N, Special Register Designator, Increment

where N represents a memory designator bit of one in the command code, a zero in the first bit position of the address group, and a one in the tabular bit position of the address group. The special register designator specifies one of the registers in the related group, either numerically or mnemonically (see Figure V-1, page 52), and the increment is a number from 0 to 31. The computer interprets the contents of the specified register as the bank indicator and subaddress of a memory location in any bank. The increment is added to the low-order bits of the contents of the special register <u>after</u> use, permanently altering them. Thus, the address

N, R1, 10

designates the contents of the related special register R1, which are interpreted as the location of an operand in main memory. After use, the contents of R1 are incremented by 10.

<u>Indexed Indirect Memory Location Address</u>

As noted in the discussion of indexed special register addressing, the contents of an index register may be interpreted as a special register group indicator and subaddress, a tabular bit, and an increment. If the tabular bit position of the augmented index register contents has the value of one, then the contents of the special register (designated by the augmented index register contents) are used to locate the operand in main memory. This type of addressing is called indexed indirect memory location addressing. The generation of an indexed indirect memory location address is identical to that shown in Figure IV-6. However, the tabular bit position in the augmented contents of the index register has the value of one for an indexed indirect memory location address whereas it has the value of zero for indexed special register addressing.

Indexed indirect memory location addressing makes it possible to use any of the 256 special registers in the system to address any available memory location indirectly. The retained contents of the special register are always modified <u>after</u> use by the amount of the increment, under control of the special register sign bit.

ARGUS notation for an indexed indirect memory location address resembles that for an indexed special register address, taking the form

Index Register Designator, N, Special Register Designator, Increment.

The comments made with respect to ARGUS notation for an indexed special register address are also applicable to an ARGUS indexed indirect memory location address.

Summary of Address Forms

The binary forms of six different address types are described in the preceding pages and illustrated in Figures IV-3 through IV-7. Figure IV-8 suggests a method with which the reader may determine by inspection the address type of any binary address configuration. This figure is not illustrative of the steps taken by the machine in interpreting addresses.

Significant Main Memory Addresses

Regardless of the amount of main memory available, the first memory bank of every Honeywell 1800 system includes certain locations whose use by the programmer is restricted. These locations are automatically involved in certain central processor functions described below. Although reserved for these functions, they are nevertheless directly addressable by the programmer and may be used with caution by a person familiar with the situations in which the central processor uses them automatically. (It is recommended that they not be used when processing is done in parallel.) The complete addresses, in octal notation, for these "reserved" locations are as follows:

00000-00017:	Used for automatic access in the multiply instructions
00021:	Main console typewriter buffer and Console slave typewriter buffer

In addition to its use with the multiply instructions, location 00000 is also used during the execution of any instruction which employs a mask and stores a result in a special register. (Note that location 00020 is not reserved.) Since the console typewriter is used at least to a limited degree in every system, its buffer location should not be used by the programmer for storage.

The multiply instructions (see Section VI) in the Honeywell 1800 generate a set of multiples of the multiplicand which are stored in locations 00000-00017, destroying any information previously stored in these locations by the programmer. Since 16 multiples of the multiplicand are generated during execution of a binary multiply, this instruction involves all 16 locations. The decimal multiply instruction, however, requires only 10 partial products, so that only locations 00000-00011 inclusive are affected by this instruction. It must be remembered that every program running in parallel uses these same locations whenever a multiply instruction is executed. Thus, a programmer who hopes to use these locations with impunity must consider the requirements of other programs which may be run at the same time as his own. It should also be noted that the multiples stored in these locations include modulo-3 check bits stored in the parity bit positions, with the result that these locations will generally contain words which the parity checking circuits will find invalid.

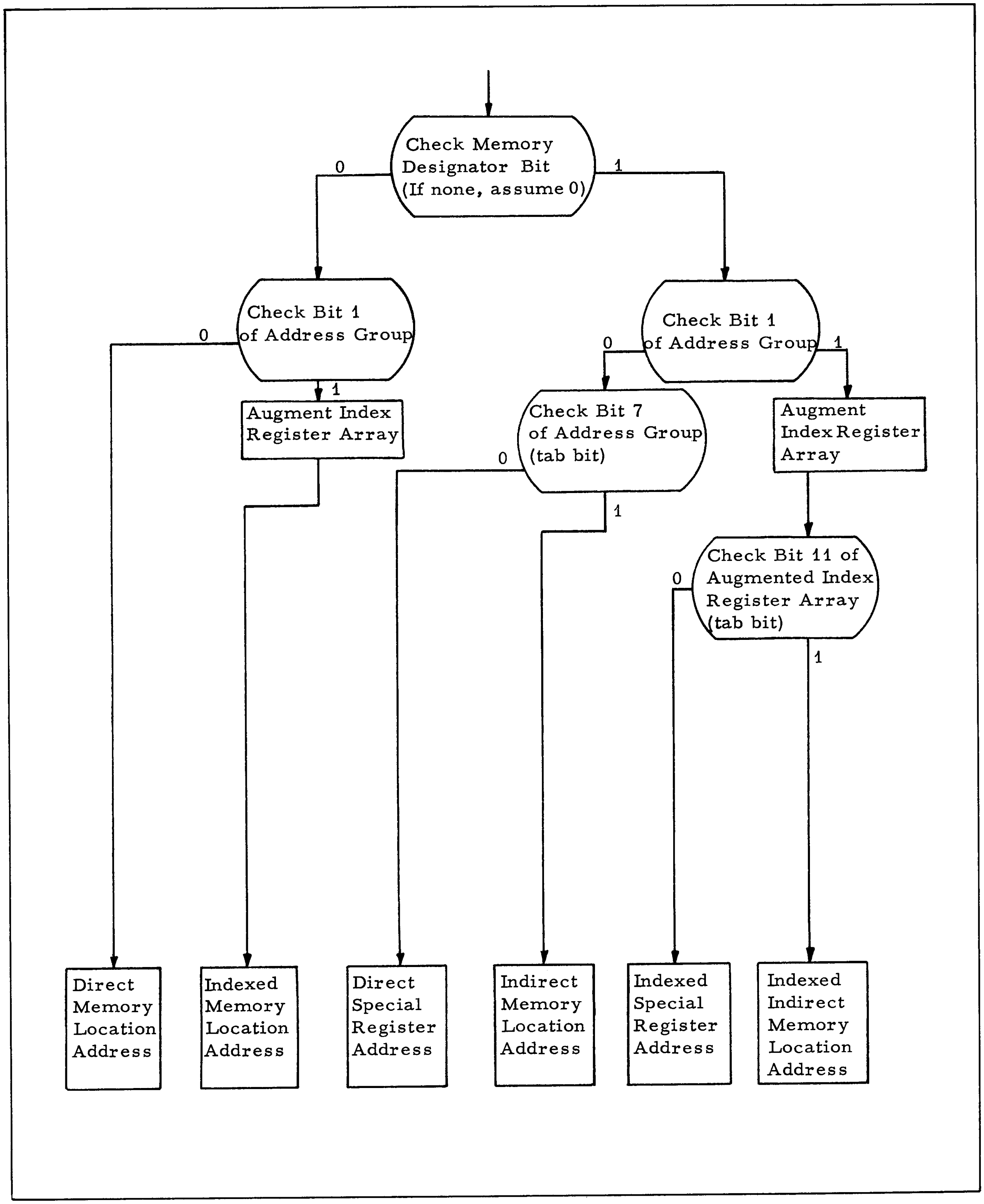

Check Memory Designator Bit (If none, assume 0)
0
1
Check Bit 1 of Address Group
0
1
Augment Index Register Array
Check Bit 1 of Address Group
0
1
Check Bit 7 of Address Group (tab bit)
0
1
Augment Index Register Array
Check Bit 11 of Augmented Index Register Array (tab bit)
0
1
Direct Memory Location Address
Indexed Memory Location Address
Direct Special Register Address
Indirect Memory Location Address
Indexed Special Register Address
Indexed Indirect Memory Location Address

Figure IV-8. Interpretation of Address Bit Structure

Stopper Address

When the contents of a special register, interpreted as a main memory address, are modified by incrementing or augmenting, a carry may occur from the 11-bit subaddress into the bank indicator bits. Thus, a sequencing counter can be stepped through successive memory banks, and a single peripheral or transfer instruction can handle a record which is not stored entirely within one memory bank. There is one address, however, which by definition is neither incremented nor decremented when it appears in a special register. This address, called a stopper address, represents the highest-numbered location in the memory of a given Honeywell 1800 system, regardless of the number of banks in the system. Its 11-bit subaddress, therefore, represents location 2047 in some memory bank. Its bank indicator is the highest such indicator in the particular system and varies from installation to installation. The largest possible value, in octal, for a complete stopper address is 77777. This occurs only in a system having 32,768 words of main memory.

The stopper location can only be addressed directly by a program under control of a sequencing counter which contains the highest bank indicator in the system. Other programs must address the stopper either by indexing or indirectly through a special register containing the highest bank indicator and a subaddress of 11 binary ones. By addressing the stopper in the A address group of a read instruction, it is possible to move tape without disturbing any memory locations except the stopper. Similarly, it is possible to read only part of a record into memory and discard the balance by causing the first unwanted word to be stored in the stopper location.

Although the stopper address cannot be incremented, it is possible for an address in a special register to receive an increment or augmenter greater than the difference between its initial value and the stopper or to be decremented or negatively augmented by an amount greater than its initial value. The effects of such operations, however, should be carefully noted. These are summarized below:

1. If a word in the control memory receives an increment greater than the difference between its initial value and 77777 or a decrement greater than its initial value, the result restored to the special register contains invalid parity bits. The next attempt to use the contents of this special register as an address (e.g., indirect addressing) will result in a control error (see Appendix D). However, it is possible to use these contents as an operand or to write into this special register without error.
2. Whenever the stopper address for a given installation is less than 77777 and a word in the control memory receives an increment greater than the difference between its initial value and the stopper (but not greater than the difference between its initial value and 77777), a legal special register word is created and direct addressing of this special register

may take place without error. The resulting special register word, however, represents the address of a non-existent main memory location for this installation. Thus, if this special register is referenced as the source of a main memory address (indirect memory location addressing), a control error will result.

3. If a legal special register word representing a memory location with an address greater than the stopper and having a negative sign bit appears in an index register, this index register may be used for indexed addressing without error, provided that the result of indexing is the address of an existing memory location.

Inactive Addresses

The Honeywell 1800 central processor contains three arithmetic registers which have no addresses: the accumulator, the mask register, and the low-order product register. Programmer access to these registers is provided by the technique of inactive addressing with certain specified instructions. When an address group in an instruction has the octal value 7777 (12 binary ones), that address is said to be inactive. The memory designator bit, if any, must be zero; otherwise, the behavior of the system is unspecified. Instructions in which the designator bits are used for other purposes and do not have to be zero are: masked general instructions, peripheral instructions, and simulator instructions. In addition, the designator bit for the A address group of the control program instruction (see Section XII) can be either "1" or "0," because the A address group is ignored in this instruction.

Access to the accumulator is provided by the proper use of inactive addressing in conjunction with the add instructions. Inactive addressing with the extract instruction provides access to the mask register. Access to the low-order product register is made possible by inactive addressing with the transfer and sequence change instruction (TS in ARGUS language).

The behavior of the accumulator when inactive addressing is used in the binary add, decimal add, or word add instruction is specified as follows:

1. If address A is inactive, the previous contents of the accumulator are used as if they were the contents of A. However, if the contents of the accumulator have already been delivered to a memory location by a previous instruction, a control error will result (see Appendix D).
2. If address B is inactive, the accumulator (which contains the contents of A if A is active or the previous contents of the accumulator if A is inactive) is left undisturbed.
3. If address C is inactive, the normal process of hunting for the next sequence counter in demand is inhibited, and the result remains in the accumulator at the conclusion of the instruction.

Thus, if the B and C addresses are inactive, the effect of the instruction is to transfer the contents of address A to the accumulator. If the A and B addresses are inactive, on the other hand, the effect is to transfer the contents of the accumulator to the location specified in the

C address. Certain restrictions should be noted with respect to the sequencing of these instructions when they contain inactive addresses. If a decimal or binary add instruction is used to place a word in the accumulator, the same type of instruction should be used to transfer the contents of the accumulator, with proper sign, to memory. Similarly, if a word has been placed in the accumulator by a word add instruction, the word add instruction must be used to deliver the contents of the accumulator to memory. The explanation for this restriction is reserved for the section discussing the arithmetic instructions in detail (Section VI).

The mask register, not to be confused with the mask index registers in the control memory, is a full-word register in the central processor which stores the mask during the execution of a masked instruction. In the extract instruction, the location of the mask is specified in the B address. At the conclusion of the execution of an extract instruction with all addresses active, the original contents of the B address are left in the mask register. Thus, a word may be loaded into the mask register, without disturbing memory, by using an extract instruction with an inactive C address. If address B is inactive, the previous contents of the mask register will be used as the mask. The contents of the mask register are transferred to the location specified in address C of an extract instruction if address B is inactive and the contents of address A consist of all (48) binary ones.

The low-order product register is of interest to the programmer primarily because it stores the low-order portion of the result of a multiply instruction. In order to obtain this result, the programmer may use the transfer and sequence change instruction (TS) with an inactive A address, which transfers the contents of the low-order product register to the location specified by the B address. If address A of this instruction is active and address B is inactive, the contents of A are transferred to the low-order product register and to the accumulator. Although these contents can then be retrieved from the low-order product register, as described above, any attempt to obtain them from the accumulator by inactive addressing with a word add instruction will probably result in a control error.

If any instructions intervene between the instruction which stores information in an arithmetic register and that which attempts to retrieve the information, the behavior of the system is unspecified. Moreover, if a masked instruction is used to attempt to retrieve information from the accumulator or low-order product register, the behavior of the system is unspecified unless the same mask was used when the information was stored there. Since masking is not permitted with the multiply instructions, a masked transfer and sequence change (TS) instruction should never by used to retrieve the low-order product.

In addition to its use in providing access to the accumulator, the mask register, and the low-order product register, the technique of inactive addressing has special significance in the read, write, and rewind instructions. These features are discussed in Section XI.

The 1801-B Floating-Point Option includes two additional arithmetic registers known as the floating-point accumulator (FLAC) and floating-point low-order product register (FLOP). In general, access to these two registers is obtained by the use of inactive addressing in the scientific instructions, as described in Section XIII.

Two other general situations in which the effect of inactive addressing has been specified should be mentioned:

1. If the C address group is inactive in an instruction that normally changes the sequencing counter to select the next instruction from the location given in the C address group, the counter remains unchanged, unless its contents have been altered under the direction of the A or B address group.
2. When an inactive C address occurs in any instruction for which such an address is allowed, the normal process of hunting for the next sequencing counter in demand is omitted.

In all cases of inactive addressing not specifically mentioned in this section, the behavior of the Honeywell 1800 is currently unspecified.

In ARGUS language, an inactive address is indicated by a hyphen (-) in the address field.

SECTION V

SPECIAL REGISTERS

Each of the 256 special registers in the Honeywell 1800 control memory is uniquely designated by eight bits, consisting of a 3-bit group indicator to specify one of eight special register groups and a 5-bit subaddress to designate one of 32 special registers in a group. Figure V-1 lists the 32 registers associated with each group, together with their numerical subaddresses from 0 to 31 and their mnemonic designations in ARGUS language. The function of each of these registers is detailed in this section.

Each special register has the capacity to store 16 bits of information, plus two checking bits. The information bits consist of a sign bit ("one" for plus, "zero" for minus) and 15 bits which usually represent the bank indicator and subaddress of a main memory location. When the contents of a special register are modified arithmetically within the special register addition circuitry, the sign bit determines whether they are incremented or decremented. When a 16-bit special register word is transferred to the accumulator or to a main memory location, it is stored in the low-order 16 bits (bit positions 33 to 48) of the specified location. The high-order 32 bit positions of the location are all cleared to zero. Thus, when a special register word is manipulated within the arithmetic unit by an instruction which treats its operands as full-word signed numbers (see Section VI), the word appears to be negative, since it contains four zeros in the sign bit positions. When information is transferred from the accumulator or from a main memory location to control memory, only the low-order 16 bits of the word are stored in the special register; the high-order 32 bits are discarded.

The discussion of indexed memory location addressing in Section IV states that whenever a special register is addressed by its full 5-bit subaddress in an instruction address group, it is said to be explicitly addressed; whenever a special register is referenced in any other way, it is said to be implicitly addressed or referenced. It is also stated that an index register is implicitly referenced when designated by the 3-bit index register number in an indexed address. Implicit addressing is further illustrated by the following examples:

1. A sequencing counter is referenced implicitly every time an instruction is selected and executed;
2. A read address or write address counter is referenced implicitly whenever a peripheral read or write instruction is executed;
3. Two arithmetic control counters are implicitly referenced whenever an N-word transfer instruction is executed.

It will be noted that in each of these examples the implicitly referenced special register is called a counter. The following rule may be stated: those special registers designated in Figure V-1 as counters are always automatically incremented (if the special register sign is positive) or decremented (if the special register sign is negative) by one each time they are referenced implicitly. When these counters are addressed explicitly in an instruction address group, however, incrementing is not automatic but occurs only under program control, if specified in the address group.

A one in the sign bit of a special register represents plus; a zero represents minus. In general, the value of this bit is changed by explicit addressing. With some special registers, however, the value of the sign bit can be changed by implicit addressing, as discussed under the descriptions of the individual registers.

Subaddress	Mnemonic Address	Name
00	AU1	Arithmetic Control Counter No. 1
01	AU2	Arithmetic Control Counter No. 2
02	SC	Sequence Counter
03	CSC	Cosequence Counter
04	SH	Sequence History Register
05	CSH	Cosequence History Register
06	UTR	Unprogrammed Transfer Register
07	MXR	Mask Index Register
08-15	X0-X7	Index Registers
16-23	R0-R7	General Purpose Registers
24-31	S0-S7*	General Purpose Registers
28	RAC	Read Address Counter
29	DRAC	Distributed Read Address Counter
30	WAC	Write Address Counter
31	DWAC	Distributed Write Address Counter

* In those special register groups associated with active input and/or output channels, S4-S7 are replaced by RAC, DRAC, and/or WAC, and DWAC.

Figure V-1. Special Register Names, Subaddresses, and Mnemonic Addresses

Under program control, the contents of a special register may be arithmetically modified within the special register circuitry in one of two ways: by augmenting an index register or by incrementing an explicitly addressed special register. Augmenting an index register does not alter the retained contents of the register, since the augmenter is actually added to the contents of the index register after they have been read out. One and only one address selection occurs each time the index register contents are augmented. Thus, even if the same index register is referenced in two successive instructions or twice within the same instruction, the same base address is used for each address selection. Incrementing, on the other hand, alters the retained contents of the special register. After the special register has been selected and its contents used, the increment specified in the address group is actually added to those contents, and the result is returned to the special register before the next address is selected. Thus, when the same special register is addressed twice within an instruction, the contents of the register at the second addressing are different from the contents at the first addressing, unless, of course, a zero increment is specified in the first instance. Whether done automatically or controlled by the programmer, incrementing (and augmenting) is a checked operation which takes place in addition circuitry peculiar to the special registers.

It is important to emphasize a few basic rules which govern programmed incrementing of explicitly addressed special registers. If a special register is addressed directly as the source of an operand, incrementing takes place. If a special register is addressed directly as a result location, on the other hand, no incrementing takes place even if programmed. When the contents of a special register are used to locate either a source or a result location in main memory (indirect memory location address), incrementing takes place as specified by the 5-bit increment in the address group. However, if a sequencing counter is used to address a memory location indirectly in the C address and the instruction is one which would normally change the contents of that counter to the memory location address specified by C, the contents of the sequencing counter will not be incremented even though an increment is specified in the address group.

The rules of incrementing which apply under indirect memory location addressing are illustrated by the transfer and sequence change instruction shown below (in ARGUS format):

```
          TS     ITEMA     N, R1, 1     N SC, 5
```

When this instruction is executed, the word at ITEMA is transferred to the memory location designated by the address stored in special register R1. The contents of R1 are incremented by one after use and replaced in R1. Since the increment of five is not added to the contents of the sequence counter, the result is the same as if the C address had been inactive.

Any instruction which can explicitly address a special register may operate on its contents. This means that a special register word may be shifted, may be operated upon arithmetically, may be compared, and may be moved around in either main or control memory. When a special register word is brought into the accumulator, bits 1 through 32 are filled with zeros. Since four zero bits in the sign position define a negative number, a control memory word is always negative when manipulated in the accumulator, regardless of the value of the special register sign bit.

The peripheral read and write command codes do not provide memory designator bits for explicit addressing of the control memory. It is therefore impossible to read directly into a special register from a peripheral device or to deliver the contents of a special register directly to a peripheral device. For the same reason, it is also impossible to address a special register in a masked general instruction.

Each group of special registers forms a control center for a single program. Thus, as many as eight independent programs may be active at the same time. Each program proceeds under control of the sequence or cosequence counter in its own special register group and references the other special registers (index registers, mask index register, and so forth) in this group. Direct memory location addressing allows the programmer to address only those 2048 memory locations within the bank specified by the bank indicator of the sequencing counter which referenced the instruction. When the main memory is addressed through the special registers, however, the program may have access to a location in any bank. Furthermore, when any of the counters in the control memory is incremented, any resulting carry may propagate throughout the full 15-bit address, with the result that the main memory is completely continuous when referenced through these counters. Thus, sequencing of control, reading, writing, and transfer of information may all proceed without regard to bank designation.

Sequencing Counters

Each special register group contains two sequencing counters called the sequence counter (SC) and cosequence counter (CSC). Except in the case of simulator instructions, the programmer may use either of these counters to sequence his program. Furthermore, in any instruction except the simulator, proceed, and peripheral instructions, he may specify which counter will select the next instruction, with the result that he may change control between the two with complete freedom. The use of two counters in this way is called the bisequence operation mode. Since the behavior of the two counters is identical, the following description of the sequence counter is also applicable to the cosequence counter.

The sequence counter contains a sign and 15 bits which are interpreted by the control circuitry as a bank indicator and a subaddress. These 15 bits represent the complete address of a main memory location from which an instruction is to be selected. Each time the sequence counter is implicitly referenced for the selection of an instruction, its contents are automatically incremented or decremented by one (according to the value of the sign bit) and immediately replaced in the counter. During the execution of an instruction selected from location N, for example, the sequence counter contains the quantity N + 1 if the sign bit is positive. In this case, therefore, instructions are taken from successively higher memory locations. If the sign bit is negative, on the other hand, the instructions will be selected from successively lower memory locations.

An instruction that references a sequencing counter implicitly by specifying a change in its contents to the location designated by the C address group can change the sign of the counter. If the C address specifies a direct or indexed memory location address, a positive sign is always inserted in the counter. If, however, the address specified is an indirect memory location or indexed indirect memory location, the new sign of the sequencing counter is determined by the sign of the designated special register, because the complete contents of the special register are transferred to the sequencing counter (see pages 80 and 81).

As previously noted, carries may propagate across the entire 15 bits of a sequencing counter during incrementation. Instruction sequences can therefore pass freely from one memory bank to another. If an attempt is made to sequence the counter beyond the highest memory address included in a particular system, however, a control error will result and the machine will stop (see Appendix D). The same result will occur if a sequencing counter containing a negative sign and 15 binary zeros is implicitly referenced.

The initial setting of the sequence counter is normally made by transferring from main memory a word whose low-order 16 bits represent the desired sign and the memory address from which the first instruction is to be selected. Alternatively, the starting address may be entered directly into the counter from the console typewriter. Once the sequence counter is set and referenced, it continues to select instructions from successive locations until an instruction is executed which specifies the alternate counter as the source of the next instruction or which changes the contents of the counter itself through explicit or implicit addressing. When an instruction specifies a change of counter but not a change in the contents of a counter, the only change which occurs in the two sequencing counters is the normal incrementation of the counter which selected the instruction. An instruction which explicitly addresses a sequencing counter as a result location simply causes the contents of that counter to be replaced. For example, the instruction

```
TX    Z, AU1   -    Z, SC
```

merely replaces the contents of the sequence counter with the contents of AU1, so that the next instruction is selected from the location whose address is stored in AU1. No record of such a sequence change is retained by the machine. An instruction which changes the contents of a counter by implicit reference, on the other hand, alters the contents of the counter specified as the source of the next instruction and stores the contents of the counter which selected this instruction in the history register (see below) associated with the counter whose contents are changed. Thus a record of the last implicit sequence change which affected the contents of either counter is always available internally. For example, an instruction selected under control of the sequence counter specifies the cosequence counter as the source of the next instruction and directs the program to transfer a word from A to B and select the next instruction from the location specified by C:

```
TS     C     RECORD     OUTPUT     SECTIONA
```

This instruction puts the address tagged SECTIONA in the cosequence counter, where it is selected as the address of the next instruction, and stores the incremented contents of the sequence counter in the cosequence history register.

History Registers

For each sequencing counter in the system, there is a corresponding history register called the sequence history register (SH) or the cosequence history register (CSH). These registers are used to store the entire contents of a sequencing counter (including sign) whenever the counter is implicitly addressed by an instruction specifying a change in its contents. If an instruction selected by the sequence counter from location M specifies a sequence change to location N, and the next instruction is also to be selected by this counter, then the address M + 1 is stored in the sequence history register and the sequence counter itself is set to the address N. If, however, the alternate counter is specified as the source of the next instruction, then the cosequence counter is set to N and M + 1 is stored in the cosequence history register. In other words, unless they are altered by explicit addressing, the contents of a history register always represent the incremented address of the instruction which last changed the contents of the associated sequencing counter by implicit reference. As previously noted, no change in the history register occurs if the contents of a sequencing counter are changed by explicit addressing.

Index Registers

The Honeywell 1800 contains a total of 64 index registers, of which eight (designated X0-X7) are located in each special register group. Like the other special registers, they contain 16 bits normally interpreted as a sign and a main memory or special register address. Although the index registers must always be loaded and unloaded by the use of an explicit address, they are always implicitly addressed by a 3-bit number when used for their intended purpose in indexed addressing. As explained in Section IV, an indexed address group includes a 3-bit index register number and an 8-bit augmenter to be added to the low-order contents of this register. The retained contents themselves are not modified; the special register addition circuitry merely uses the contents, together with the augmenter, to generate the main memory or special register address of an operand or result location. Since the sign of the register may be positive or negative, at the programmer's option, the generated address may be higher or lower than the base address stored in the register. The central processor accepts augmenters valued from 0 to 255.

Mask Index Register

Each special register group contains a mask index register (MXR) which is implicitly referenced whenever a field instruction or a shift instruction is executed. The 16 bits of this register are interpreted as a sign, a bank indicator, and the high-order portions of two different subaddresses. Bits 6 through 10 specify a partial address for masks used with the shift instructions; bits 11 through 16 serve the same purpose for masks used in field instructions (see Figure V-2).

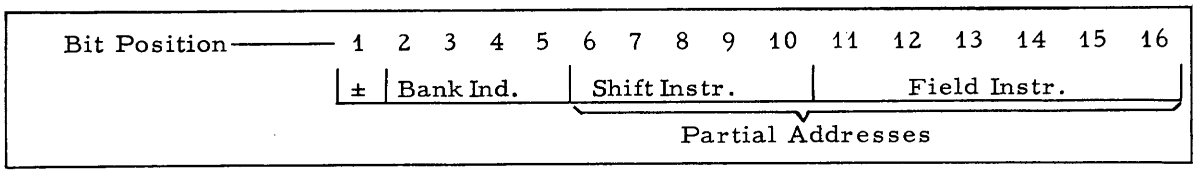


Figure V-2. Mask Index Register

The low-order portions of the mask addresses are found in the instructions themselves, as explained in Section III (see pages 31 and 32). When a shift instruction is executed, the central processor unites the low-order six bits of the B address group with the bank indicator and bits 6 through 10 from the mask index register to form the complete main memory address of the mask. This process is illustrated in Figure V-3.

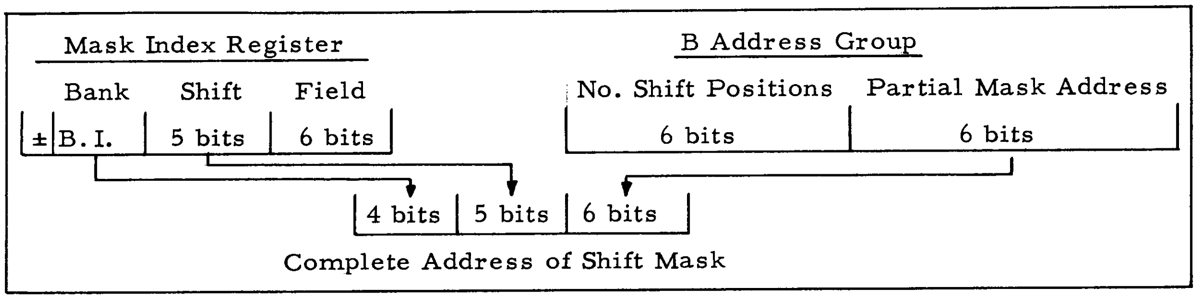


Figure V-3. Generated Mask Address in Shift Instructions

When a field instruction is executed, the central processor attaches the 5-bit partial mask address from the instruction command code to the bank indicator and bits 11 through 16 from the mask index register to form the complete main memory address of the mask. This process is illustrated in Figure V-4.

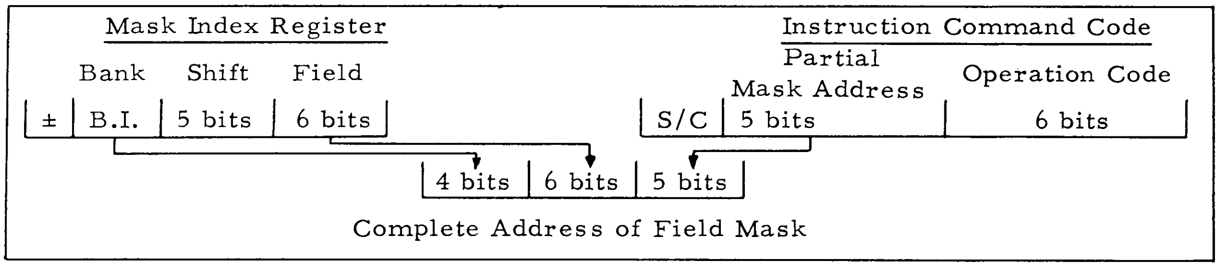


Figure V-4. Generated Mask Address in Field Instructions

Since the mask index register contains a single bank indicator, both shift masks and field masks are stored in the same memory bank. The value of the sign bit is not relevant in locating masks.

The mask index register is set by explicit addressing. Each time the programmer loads the register, he designates 96 memory locations as mask addresses, 64 for shift masks and 32 for field masks. Since the programmer may change the contents of the mask index register whenever he wishes, the number of masks available for his use is virtually unlimited.

General Purpose Registers

Each special register group contains a minimum of 12 general purpose registers (R0-R7, S0-S3), and some groups may contain 14 or 16, depending upon the assignment of input/ output channels (see Read-Write Counters, below). Like the index registers, general purpose registers are used primarily for address modification. Their use differs from that of index registers, however, in several important respects. First, they are always addressed explicitly. Secondly, the specified increment, which has an upper limit of 31, alters the retained contents of the register after use. As in the case of index registers, the address of a memory location generated by adding the increment to the original contents of the register may be higher or lower than the address originally contained in the register, according to the value of the sign bit. These registers are used mainly in the indirect addressing mode to address an operand or a result location in any bank of memory, but they may also be used as programmed counters, as temporary storage for the contents of other special registers, and for any other purpose the programmer may devise.

Read-Write Counters

Every Honeywell 1800 system includes eight channels for entry of information into the central processor from peripheral units and eight channels for output of information from the central processor to peripheral units. Two special registers known as the read address counter (RAC) and the distributed read address counter (DRAC) are associated with each input channel. For each output channel there are two similar counters known as the write address counter (WAC) and the distributed write address counter (DWAC).

Each of the eight special register groups contains a pair of read counters and a pair of write counters. The counters corresponding to the first input and output channels are located in special register group 1; those corresponding to the second input and output channels are located in special register group 2, and so forth up to the counters corresponding to the last channels, which are located in special register group 0. The read-write counters are, therefore, exceptions to the rule that an implicitly addressed special register always belongs to the group associated with the sequencing counter which selected the instruction. An implicitly addressed read or write counter is always one of the pair corresponding to the input or output channel connected to the device addressed. For example, if a program controlled by special register group 3 addresses a peripheral device attached to the first input channel, then the read or write counters in special register group 1 are activated, rather than the counters in special register group 3.

When peripheral equipment is attached to a given channel, the read or write address counter corresponding to that channel is implicitly referenced whenever the peripheral devices are addressed or are operative. In certain cases, the distributed read-write counters are also implicitly referenced. When no hardware is attached to a specific channel, or when the hardware is not being used by any active program, the associated read-write counters may be used as general purpose registers. Whenever these special registers are addressed explicitly, they lose their identity as automatically incremented counters.

When used to control input/output functions, those read-write counters used by a peripheral instruction are automatically loaded during execution of the instruction. The 16 bits initially loaded into the read or write address counter (RAC or WAC) always represent a sign bit, plus 15 bits (generated from the A address group) which specify the main memory location into which the first word will be read or from which the first word will be written. The distributed read or write counter (DRAC or DWAC) is automatically loaded during execution of a magnetic tape read or write instruction which senses for end-of-item words. Its initial setting represents the address (generated from the B address group) of a main memory location which contains the first entry in a table of addresses used to specify the starting location for each item to be read or written, beginning with the second item. (The starting address for the first item is obtained from the A address group and stored in RAC or WAC.)

When executing a write or a read forward instruction, whether addressed to a magnetic tape unit or to a terminal device, the sign of the RAC or WAC is automatically made positive. When a tape is read backward, the sign of the RAC is automatically made negative. In the execution of a distributed read or write, the sign bit in the distributed read-write counter (DRAC or DWAC) follows the same convention as that of the corresponding read-write address counter. It should be noted, however, that when the contents of the RAC (or WAC) are replaced by the contents of the memory location specified by DRAC (or DWAC) for distributed item handling, the sign of the RAC (or WAC) is determined by the value of bit 33 found in the main memory location containing the address of the first word in the current item.

Further details on the functions of the read-write counters are found in Section XI under the discussion of peripheral instructions.

Arithmetic Control Counters

Each special register group contains two arithmetic control counters (AU-CU counters) known mnemonically as AU1 and AU2. One or both of these counters are implicitly referenced, loaded, and automatically incremented during execution of an N-word transfer, item transfer, record transfer, binary or decimal multiply, compute orthocount, or simulator instruction. As an example, during execution of an N-word transfer of 10 words, the initial setting of AU1 represents the main memory or special register address from which the first word is to be transferred, while the initial contents of AU2 represent the location to which this word will be delivered. As successive words are transferred, the counters are automatically incremented to specify a source and result address for each word transferred. At the completion of the instruction, the counters contain addresses equal to their initial settings plus ten.

Implicit reference to an arithmetic control counter always causes the sign bit in the counter to be made positive with the following two exceptions:

1. If an arithmetic control counter is loaded with the contents of a special register, its sign bit takes the value of the sign stored in the special register.
2. When the contents of AU1 are changed for distributed item handling in the compute orthocount instruction, the sign bit in AU1 takes its value from bit 33 of the main memory location containing the address of the first word in the current item.

Since carries may propagate across the low-order 15 bits of the counter, a record which is divided between two memory banks may be transferred as easily as one contained entirely in one bank. It should be noted, however, that when the contents of an arithmetic control counter are interpreted as a special register address, a subaddress overflow will not change the group indicator but instead will change the value of the tabular bit (bit 11) from zero to one.

Like other special registers, the arithmetic control counters may be addressed explicitly in order to transfer their contents to main memory or to use them as general purpose registers. The programmer who uses them thus, of course, must remember that information stored in one or both of these registers will be destroyed by the execution of certain instructions. Whenever these special registers are addressed explicitly, they lose their identity as automatically incremented counters. The arithmetic control counters are described more fully in connection with the instructions which use them.

Unprogrammed Transfer Register

The Honeywell 1800 is so designed that the occurrence of certain unusual events during execution of a program does not stop the machine but rather effects a transfer of control out of the normal sequence of the program to initiate appropriate action as specified in a programmed subroutine. The unprogrammed transfer register (UTR) in the special register group controlling the program is the key to the location of these subroutines designed to handle the seven different types of conditions which may cause an unprogrammed transfer.

The unprogrammed transfer register is initially loaded, by explicit addressing, with 16 bits which represent a sign, a bank indicator, and an 11-bit main memory subaddress. The address thus loaded, called U, must be even or a control error will result when an unprogrammed transfer is attempted. When an unprogrammed transfer situation arises, the control circuitry reads out the contents of the UTR and inserts a one into bit 16 if the instruction causing the unprogrammed transfer was selected from the cosequence counter. The instruction itself is stored in the address thus generated. In other words, the instruction causing the transfer will be stored in U if it was selected by the sequence counter or in U + 1 if it was selected by the cosequence counter.

The conditions which result in unprogrammed transfers are listed in Figure V-5. Each of these events causes the execution of one instruction whose address is formed by adding a constant (n) to the contents of the UTR, under control of the UTR sign bit. If the UTR contains a positive sign, the unprogrammed transfer causes execution of an instruction whose address is U + n. If the sign bit in the UTR is negative, the unprogrammed transfer is made to U - n.[1] If the instruction causing the transfer was selected by the sequence counter, n is an even number from 2 to 14; if it was selected by the cosequence counter, n is an odd number from 3 to 15. Thus, the value of n depends upon the event that caused the transfer and the counter that selected the instruction, as illustrated in Figure V-5. (CAUTION: An unprogrammed transfer to a memory location which contains a peripheral or print instruction will cause

[1] In Figure V-5, the unprogrammed transfer addresses are listed as U ± n. For the sake of simplicity, they are referred to as U + n throughout the manual.

the error condition to be ignored in the event that the device addressed is momentarily unavailable to the unprogrammed transfer routine.)

Event Causing Unprogrammed Transfer	Instruction Stored In	Next Instruction From
Parity Failure		
Sequence Counter	U	U ± 2
Cosequence Counter	U + 1	U ± 3
Beginning or End of Tape		
Sequence Counter	U	U ± 4
Cosequence Counter	U + 1	U ± 5
Read or Write Error		
Sequence Counter	U	U ± 6
Cosequence Counter	U + 1	U ± 7
Addition or Subtraction Overflow		
Sequence Counter	U	U ± 8
Cosequence Counter	U + 1	U ± 9
Division Overcapacity		
Sequence Counter	U	U ± 10
Cosequence Counter	U + 1	U ± 11
Exponential Underflow		
Sequence Counter	U	U ± 12
Cosequence Counter	U + 1	U ± 13
Exponential Overflow		
Sequence Counter	U	U ± 14
Cosequence Counter	U + 1	U ± 15

Figure V-5. Unprogrammed Transfers of Control

It should be noted that every direct memory location address in an instruction selected by the unprogrammed transfer register has the bank indicator bits from the UTR appended to it to form a complete 15-bit address. In particular, if a change of sequence is made by direct main memory addressing in such an instruction, the bank indicator bits from the UTR are placed in the sequence or cosequence counter.

The execution of an unprogrammed transfer does not alter the contents of either sequencing counter. Thus, control returns immediately to the normal sequencing of the program unless the unprogrammed transfer instruction itself changes the contents of the sequencing counter from which the next instruction is selected. Note, however, that the bisequence bit in the instruction causing the transfer is ignored.

By inserting the appropriate address into the unprogrammed transfer register, the programmer may select any area of memory for use as a corrective routine selection table. He may also change the contents of the UTR at any point in the program or he may change the contents of any entry in the table to correspond with the particular portion of the program being executed at the time.

The unprogrammed transfer register can also be used in connection with fixed starts from the punched card equipment, the paper tape equipment, and the printers. As directed by the programmer, the operator can set a switch on the peripheral device to one of seven positions. When a peripheral instruction is next directed to this unit, a three-bit octal number (001 through 111) corresponding to the switch setting is stored in bits 1 - 3 of the program control register (see Section XII), the peripheral instruction is stored in U or U + 1, control is transferred to U + 6 or U + 7, and the peripheral device stops. In U + 6 or U + 7, the programmer should have placed the entry to a routine that analyzes the contents of the high-order three bits of the program control register and takes appropriate action. In effect, this gives the programmer seven courses of action within the framework of one routine.

With the exception of the parity failure condition, the events resulting in unprogrammed transfers are associated with a few specific instructions. Further details on these conditions appear under the descriptions of the appropriate instructions. Parity failures are discussed in detail in the paragraphs which follow.

If a parity error is detected in a word selected from memory, the current instruction is completed and a parity error unprogrammed transfer is executed to U + 2 or U + 3, except in the following cases:

1. If the word selected is the next instruction to be performed, a control error occurs (see Appendix D).
2. If the word selected is a control memory word used as an operand, the error is ignored. If, however, a control memory word is used, implicitly, or explicitly, as a main memory address, parity failure results in a control error.
3. If a parity failure occurs during data transfer in a record or item transfer, the instruction is not complete, and the unprogrammed transfer to U + 2 or U + 3 occurs immediately.
4. If a parity failure occurs in a word being transferred to a peripheral device or inserted in the distributed write address counter under control of a write instruction, a write error condition is stored, and the unprogrammed transfer action is as specified under the discussion of write errors in Section XI.
5. If a parity failure occurs in a word being transferred from a peripheral device under control of a read instruction with inactive B address, a read error indication is stored, and the unprogrammed

transfer action is as specified under the discussion of read errors in Section XI.

6. If a parity failure occurs in a word being transferred from a peripheral device under control of a read instruction with distributed item control, or in a word read from memory for insertion into the DRAC, the transfer of words to memory is immediately halted, and the unprogrammed transfer action is as specified in Section XI.

7. If a parity failure occurs in a word being printed under control of a print instruction, no unprogrammed transfer occurs. Instead, a space is inserted at the end of the word and two octal check characters are printed.

8. If a parity failure occurs in a check parity instruction, no unprogrammed transfer takes place, but a sequence change occurs as described in Section XII.

9. During a compute orthocount instruction, parity failure is ignored, and no unprogrammed transfers occur.

If a parity failure unprogrammed transfer occurs as the result of executing an instruction that normally changes the contents of the sequencing counter to the location specified by the C address group, the sequencing counter is not changed on completion of the instruction. This change does not occur until cycle zero of the next instruction.

When an instruction operand with bad parity is transferred in memory without alteration, it is stored in its original form, including parity bits; as a result, another attempt to manipulate the operand will also cause a parity failure unprogrammed transfer. If, on the other hand, an operand with bad parity is involved in any arithmetic or masking operation, the result of the operation is stored with correct parity bits. Note, however, that only the parity bits will be correct; the data bits will reflect any errors in the operand data bits.

SECTION VI

ARITHMETIC INSTRUCTIONS

Arithmetic instructions in the Honeywell 1800 involve the use of two arithmetic registers called the accumulator (AC) and the low-order product register (LOP). As discussed in Section IV, these registers are accessible to the programmer through the technique of inactive addressing with certain specified instructions. The accumulator is used in all the instructions described below. The low-order product register is used in the multiply instructions and in the masked arithmetic instructions.

The Accumulator

The accumulator consists of 48 flip-flops, each capable of storing a single binary digit. Used in conjunction with the accumulator is a single flip-flop called the sign flip-flop. Since the operands handled in most arithmetic operations are treated as 44-bit numbers with 4-bit signs, most arithmetic instructions use the sign flip-flop together with the low-order 44 bits of the accumulator. The exceptional instructions which handle unsigned 48-bit numbers are so noted as they are described.

For all arithmetic operations on signed numbers, the sign flip-flop, originally set to the value of zero, is changed to the value of one if any of the four sign bits in the A operand is a one. (This is the logical OR function.) The setting of the sign flip-flop, the sign of the B operand, and the operation code determine the sign of the result. When the result is read out of the accumulator, a sign consisting of four bits identical in value (zero or one) to the final setting of the sign flip-flop is attached to the low-order 44 bits of the accumulator. Thus only two sign configurations may be obtained as the result of an arithmetic operation: four binary zeros indicating a negative result or four binary ones indicating a positive result. If the result of an add or subtract instruction is zero, the result takes a sign based on the sign of the A operand.

When addition is performed on signed numbers, bits 1 through 4 of the accumulator are automatically filled with binary ones so that if overflow occurs it may be sensed in bit 1, the same position in which it is sensed during the addition of unsigned 48-bit operands. Similarly, bits 1 through 4 are filled with binary zeros when subtracting signed numbers in order that borrows may be sensed at the same position for both signed and unsigned numbers. If overflow occurs, the instruction is completed and the low-order 44 bits of the accumulator, plus a 4-bit sign based on the value of the sign flip-flop, are stored in the location specified by the

C address. An unprogrammed transfer is then made to U + 8 or U + 9, where the programmer should have stored the entry to a subroutine to handle this condition. The instruction which resulted in the overflow is stored in U or U + 1 (see Figure V-5, page 62).

The arithmetic operations are not actually performed in the accumulator but in an adder consisting of 12 gate buffer amplifiers. Both binary and decimal arithmetic are performed in the same adder, which handles three 4-bit groups at a time. For binary instructions, these 4-bit groups are considered as hexadecimal digits, with carry occurring after a group reaches the value of 15. This results in a pure binary operation. For decimal instructions, the adder is made to carry when a 4-bit group reaches the value of nine, so that decimal arithmetic is performed. If a decimal addition instruction involves an operand which contains hexadecimal digits, however, a variant on normal addition occurs in accordance with the following rules:

1. If the hexadecimal digit appears in the A operand, the corresponding digit in the B operand is added in hexadecimal fashion. In other words, 14 + 1 becomes 15, 14 + 3 becomes 1 with a carry of 1.
2. If the A operand contains decimal information and a hexadecimal digit occurs in the B operand, then the result is decimalized as follows: 3 + 14 becomes 7 with a carry of 1. If the result of decimalization is 19 or 20, a control error occurs (see Appendix D); if the result exceeds 20, the performance of the system is unspecified.

The contents of the operands are inspected digit by digit. Therefore, the result obtained by adding two words having both hexadecimal and decimal digits must be ascertained on a digit-by-digit basis. Since this condition is not considered an error by the central processor, except in the circumstance noted above, the programmer will receive no indication of the existence of a hexadecimal digit in an operand handled by a decimal instruction.

Addition and subtraction of signed numbers conforms to normal algebraic rules. Thus, an add instruction causes operands with like signs to be added and operands with unlike signs to be subtracted. A subtract instruction causes operands with unlike signs to be added and operands with like signs to be subtracted. A more detailed discussion of fixed-point addition in the Honeywell 1800 will be found in Appendix A.

Several precautions must be observed in working with the accumulator. In the discussion of inactive addressing (Section IV), it is pointed out that the result of an addition may be left in the accumulator by using an add instruction with an inactive C address and that the contents of the accumulator may be stored in memory by using an add instruction with inactive A and B addresses. It should also be noted that if the contents of the accumulator were formed by an instruction which treated the operands as signed 44-bit numbers, such an instruction must be used to store the contents of the accumulator in order to guarantee them the proper sign.

Otherwise, the entire 48-bit contents will be stored rather than the low-order 44 bits with a 4-bit sign determined from the value of the sign flip-flop. Similarly, if the contents of the accumulator were created by an instruction which treated the operands as unsigned 48-bit words, such an instruction must be used to transfer the entire contents of the accumulator to memory without reference to the sign flip-flop.

Another precaution involves the condition of the accumulator after its contents have been delivered to memory. After a result formed in the accumulator has been transferred to memory, the contents of the accumulator are invalid. A second attempt to transfer the result, therefore, will cause a control error and the machine will stop. Since a hunt for the next program demand will have occurred immediately after the first transfer, this behavior imposes no real restriction on the use of the accumulator.

The Low-Order Product Register

The low-order product register is a 48-bit register similar to the accumulator. As its name implies, the register is used to store the low-order portion of the result of a multiply instruction. The contents of the register are interpreted as a sign and a 44-bit number.

Binary Add, BA

The binary add instruction causes the contents of the location specified by A to be added algebraically to the contents of the location specified by B and the result of the operation to be stored in the location specified by C. The contents of both A and B are regarded as 44-bit numbers with 4-bit signs. If any sign bit for the A or B operand is a one, then the corresponding operand is considered positive. After the addition is complete, the low-order 44 bits of the accumulator, plus a 4-bit sign (1111 or 0000) corresponding to the value (1 or 0) of the sign flip-flop, are stored in the location specified by C. If overflow occurs, the instruction is stored in U if the sequence counter selected it or in U + 1 if the cosequence counter selected it, and the next instruction is taken from U + 8 or U + 9.

As an illustration, assume that the following information bits are stored in memory locations tagged MONTHDAY and CONSTNT1:

```
MONTHDAY      00110...........01110000
CONSTNT1      10000...........00011001
```

When the instruction

```
BA MONTHDAY CONSTNT1   TESTAREA
```

is executed, the following result will be stored in TESTAREA:

```
TESTAREA      11110...........010001001
```

In the unmasked version of the binary add, all three addresses may be direct, indexed, or indirect and may refer to main memory locations or special registers. When the instruction is masked to permit operations on partial words, however, the A, B, and C addresses may refer only to the main memory in the direct or the indexed mode.

If the A address of the instruction is active and B and C are inactive, the contents of A are placed in the accumulator. If the B address is active and A and C are inactive, then the contents of B are added to the contents of the accumulator. If the C address is active and A and B are inactive, the low-order 44 bits of the accumulator are stored in C with the sign bits 1111 or 0000, depending upon the value of the sign flip-flop.

The time required to execute an unmasked binary add instruction whose A, B, and C addresses directly specify main memory locations is generally four memory cycles. Occasionally, one or two extra memory cycles may be required, as explained in Appendix A. The effect on timing when the instruction is masked or when it uses indexed operands or special registers is summarized in Appendix C.

Decimal Add, DA

The decimal add instruction causes the contents of A to be added algebraically to the contents of B and the result of the operation to be stored in C. The instruction differs from binary add only in the fact that the contents of each operand are handled as eleven 4-bit groups with a 4-bit sign. The sign conventions are identical, and the result stored in memory consists of the low-order 44 bits of the accumulator and a 4-bit sign corresponding to the value of the sign flip-flop.

If memory locations tagged REGPAY and OVERTIME contain the following information

```
REGPAY      +00000012500
OVERTIME    +00000001750
```

and the instruction

```
DA REGPAY OVERTIME WEEKPAY
```

is executed, then the following result will be stored in WEEKPAY:

```
WEEKPAY     +00000014250
```

The comments on the binary add instruction with respect to overflow, addressing, masking, and timing are equally applicable to the decimal add instruction, with one exception: the behavior of the system is unspecified when the C address group of a decimal add instruction contains a direct special register or indexed special register address.

Binary Subtract, BS

This instruction causes the contents of B to be subtracted algebraically from the contents of A and the result to be stored in C. The contents of both A and B are regarded as 44-bit numbers with 4-bit signs. The sign and overflow conventions followed are the same as those described for the binary add instruction.

The remarks about addressing and masking with reference to the binary add instruction also apply to binary subtract. The time required to execute an unmasked binary subtract instruction whose A, B, and C addresses directly specify main memory locations is ordinarily four memory cycles. Under certain unusual conditions described in Appendix A, however, one or two additional memory cycles may be required. The timing effect of masking, indexing, and the use of special registers is summarized in Appendix C.

Decimal Subtract, DS

The decimal subtract instruction differs from binary subtract only in the fact that both operands are regarded as eleven 4-bit groups with 4-bit signs. The description of binary subtract is applicable to decimal subtract in every other respect but the following: the behavior of the system is unspecified when the C address group of a decimal subtract instruction contains a direct special register or indexed special register address.

Word Add, WA

Word add is one of the two arithmetic instructions which regard operands as unsigned 48-bit numbers. The instruction adds the absolute values of the entire 48-bit contents of A and B in binary and stores the entire 48-bits of the accumulator in C, making no reference to the sign flip-flop. In contrast to the example cited under the discussion of binary add, when the instruction

```
WA MONTHDAY CONSTNT1 TESTAREA
```

is executed with the same operands:

```
MONTHDAY      00110.........01110000
CONSTNT1      10000...........011001
```

then the following result will be stored in location TESTAREA:

```
TESTAREA      10110........010001001
```

Although the low-order 44 bits of the result stored in TESTAREA are identical for the two instructions, the high-order four bits are different since binary add inserts in these four positions a sign based on the value of the sign flip-flop whereas word add stores the entire contents of the accumulator, without regard for the sign flip-flop.

Overflow is sensed in bit 1. If overflow occurs, the instruction is completed, and the overflow conventions set forth under the description of binary add are followed. With respect to addressing, masking, and timing, the word add instruction is identical to binary add.

Word Difference, WD

The second arithmetic instruction which treats its operands as unsigned 48-bit numbers, word difference, causes the entire 48-bit contents of B to be subtracted in binary from the entire 48-bit contents of A. The entire 48 bits of the accumulator are stored in C. If the absolute value of the contents of B is greater than the absolute value of the contents of A, then overflow occurs, and the result stored in C is the difference of the absolute values of the words. In all other respects, the word difference instruction is identical to word add.

Binary Accumulate, BT

The binary accumulate instruction totals the absolute value of the contents of A the number of times specified by the high-order six bits of the B address group, a number which ranges from 0 through 63. Although the words added are treated as signed 44-bit numbers, only their absolute values are added. The accumulator is not cleared between successive additions except for the high-order four bits. At the conclusion of the series of additions, the 44 low-order bits of the accumulator are stored in C, together with a sign (four binary ones or four binary zeros) based on the value of the sign flip-flop. This value represents the sign of the first word added (the contents of the location originally specified by the A address group).

Step by step, the instruction functions as follows. The low-order 44 bits of the A operand are transferred to the accumulator and the high-order four bits of the accumulator are set to one. If A contains a special register subaddress, incrementing is performed as specified. The high-order four bits of the accumulator are again replaced with ones, and the low-order 44 bits of A are added in binary to the contents of the accumulator. (Note well that the location now specified by A will be different from the original A if incrementing took place.) The specified incrementing is again performed (if A contains a special register subaddress), the high-order four bits of the accumulator are replaced by ones and the low-order 44 bits of A are again added to the accumulator. This process is performed the number of times specified by the high-order six bits of the B address group. If the value of these bits is zero, no information is transferrred to the accumulator, the instruction is not executed, and the next instruction is selected from the sequencing counter specified by bit 1 of the command code. The low-order six bits of the B address group are ignored.

As an example, consider the instruction

```
BT N, R1, 1 4 TOTAL
```

in the case where R1 contains a main memory address tagged OPERAND. The high-order four bits of the accumulator are set to ones, and the low-order 44 bits of OPERAND are transferred to the accumulator. Rl is incremented by one, the high-order four bits of the accumulator are replaced by ones, and the low-order 44 bits of OPERAND + 1 are added in binary to the contents of the accumulator. This process is repeated until the contents of OPERAND + 3 have been added to the total in the accumulator. The accumulator now contains the sum of the absolute values of the low-order 44 bits of OPERAND, OPERAND + 1, OPERAND + 2, and OPERAND + 3. This sum, with a sign based on the sign of OPERAND, is then stored in the location tagged TOTAL. Register Rl contains the address of location OPERAND + 4 at the conclusion of the instruction.

Overflow in the accumulator is sensed in bit 1. If overflow is sensed, the instruction is completed, and the normal overflow procedure (described under the binary add instruction) is performed. Since the high-order four bits of the accumulator are replaced by ones between successive additions, the contents of these four positions are not available to indicate the number of overflows. Thus, unless the programmer knows from the logic of his problem precisely how many overflows may have occurred and at which points, he must repeat the addition process in pairs.

The time required to execute a binary accumulate instruction with direct and/or indirect memory location addresses is three memory cycles plus one memory cycle for each word accumulated. Masking is not permitted with the accumulate instructions.

Decimal Accumulate, DT

This instruction is implemented in precisely the same way as binary accumulate, with the exception that the words added are regarded as 11-digit decimal numbers and are added according to the rules of decimal arithmetic. If hexadecimal digits appear in the operands, they are added in the same fashion described under the decimal add instruction. The remarks on overflow, timing, and masking made with reference to binary accumulate also apply to decimal accumulate.

Binary Multiply, BM

The binary multiplication instruction in the Honeywell 1800 stores a set of 16 multiples of the multiplicand (or A operand) in the first 16 locations of memory bank 0 (see page 44). Any information previously stored in these 16 locations by the programmer will be destroyed during execution of a binary multiply instruction. Since the multiples stored in these locations use the parity bit positions for modulo-3 check digits, these locations, in general, will contain words which the parity checking circuits will find invalid.

At the beginning of the instruction, the address 00000 (in octal) is placed, with a positive sign bit, in AU-CU counter 1 (AU1). The number zero (zero times the A operand) is stored in location 00000, and AU1 is incremented by one to specify the storage location of the next partial product. The A operand is then placed in location 00001; AU1 is stepped and twice the A operand is placed in location 00002; the counter is stepped again and three times the A operand is placed in location 00003; and so forth until the 16 multiples possible in the hexadecimal system are stored. These multiples are then selected and added in accordance with the value of the digits of the multiplier (or B operand). At the conclusion of the instruction the high-order product found in the low-order 44 bits of the accumulator is stored, together with a sign corresponding to the value of the sign flip-flop, in the location specified by the C address group. The low-order product remains in the low-order product register. AU1 contains the address 00020 (in octal).

Since hunting for the next sequencing counter in demand is not allowed at the conclusion of a multiply instruction, the programmer has the opportunity to store the contents of the low-order product register before an instruction from another program destroys them. As explained under the discussion of inactive addressing (Section IV, page 47), this may be accomplished by performing a transfer and sequence change instruction with an inactive A address. If both halves of the product are stored, they will have the same sign. It should be noted that the high-order product stored in C is unrounded.

Several properties of the multiply instructions deserve particular attention. The contents of the mask register are destroyed, together with the contents of those locations used to store the partial products. Furthermore, if the C address of a multiply instruction is a direct or indexed special register address, a control error will occur and the machine will stop.

Masking is not permitted with the multiply instructions. Except for the restriction already stated with respect to the C address, the A, B, and C addresses may be direct, indexed or indirect. The time required to execute a binary multiply with direct memory location addresses is 33 memory cycles.

Decimal Multiply, DM

Decimal multiply is implemented exactly as the binary multiply instruction, with the exception that only 10 partial products are generated instead of 16. After the multiples of the A operand are generated, the proper multiples are selected and added in accordance with a digit-by-digit inspection of the multiplier, or B operand. If hexadecimal digits appear in the operands, an erroneous product will be generated, and a control error may be indicated.

Like binary multiply, the decimal multiply instruction produces a 2-word result. The high-order product, consisting of 11 decimal digits, is stored in the location specified by the C address group with a sign determined by the value of the sign flip-flop. The low-order result appears in the low-order product register and may be stored by the programmer in the manner described for binary multiply. At the conclusion of the instruction, AU1 contains the address 00012 (in octal).

Since only 10 multiples of the multiplicand are stored in a decimal multiply, compared with 16 multiples for a binary multiply, only memory locations 00000 through 00011 will be affected, and the time required to execute the decimal instruction with direct memory location addresses is six memory cycles less than for the binary instruction, or 27 memory cycles.

SECTION VII

LOGICAL INSTRUCTIONS

The four instructions which make up the logical group manipulate words on an individual-bit basis, combining bits from two words to form a third. The rules by which the bits are combined are similar to the rules under which the logical elements of a computer operate; hence the name logical instructions. All operands are regarded as 48-bit words in which each bit is an individual unit of information unrelated to any other bit.

The four logical instructions are extract, substitute, half add, and superimpose. Extract and substitute have command codes of the "inherent mask" format (see Section III) and utilize the entire B address group to specify the location of a mask. Half add and superimpose have command codes of the "general, masked or unmasked" format. The time required to execute an extract or substitute instruction with direct memory location addresses is five memory cycles; for an unmasked half add or superimpose instruction with direct memory location addresses, it is four memory cycles.

The logical instructions may address special registers in one or more address groups. The result of such an operation may be determined by applying the rules governing the transfer of a special register word to a 48-bit register and the transfer of a 48-bit word to a special register.

Extract, EX

The extract instruction places the A operand in the location specified by the C address group, using the B operand as a mask and not protecting the unmasked portions of C. (The mask index register is not used in locating the mask.) This is equivalent to combining the corresponding bits of the A and B operands in accordance with the following rule:

> If corresponding bit positions in the word at A and the word at B both contain ones, the result shall contain a one in this position. In all other cases, the result shall contain a zero. This is the "logical AND" function.

The execution of an extract (or logical AND) instruction utilizes the accumulator, the mask register, and the low-order product register, and takes place in the following steps:

1. The 48-bit A operand is placed in the accumulator and the 48-bit B operand is placed in the mask register.
2. The corresponding bits of these two registers are examined. Where the bits of both registers contain a one, a one is stored in the low-order product register. Where the bits are not both ones, a zero is stored in the low-order product register.

3. The 48-bit result generated in the low-order product register is then placed in the location specified by the C address group.

Whenever a masking operation is performed, whether in a logical, a general, or a shift instruction, a logical computer element called a "gate" is set by the value of each bit from the mask register to open one transmission path and close another. In unprotected masking, one of the paths transmits the word to be masked while the other transmits generated zero bits. In protected masking, the generated zeros are replaced by the contents of the location specified by the C address group.

For example, the following locations contain the words shown:

```
OPERAND     110100100010------------
EXMASK      000011110000------------
RESULT      101001110110------------
```

when the instruction

```
EX   OPERAND   EXMASK   RESULT
```

is executed. As a result of the instruction, the contents of location RESULT will be:

```
RESULT      000000100000------------
```

As discussed in Section IV (see page 47), the use of inactive addressing with the extract instruction provides access to the 48-bit arithmetic register called the mask register. When a mask is specified in an instruction, either in the command code or in the B address group, the contents of the specified location are placed in the mask register, where they remain until a subsequent instruction calls for a mask (or until they are destroyed by the execution of a multiply instruction). Therefore, the contents of the mask register are unrelated to the instruction being performed if this instruction does not specify a mask. Since the mask register has no address, the programmer cannot transfer its contents directly to memory. However, by executing an extract instruction which specifies the address of a word of 48 binary ones in the A address group and an inactive address in B, the programmer can store in the location specified by C a word guaranteed to be identical to the contents of the mask register. This guarantee can be verified by an inspection of the above rule of operation. If the programmer wishes to insert a full word into the mask register without disturbing any memory location, he may perform an extract instruction with an inactive C address.

Substitute, SS

The substitute instruction performs the same general function as extract except that the contents of the location specified by the C address group are protected. When this instruction is executed, therefore, the computer places the 48-bit contents of A in the accumulator and the 48-bit contents of B in the mask register and then forms a new word in the low-order product register according to the following rule:

> Wherever the B operand contains a one bit, the corresponding bit of the A operand is stored in the corresponding position of the low-order product register. Wherever the B operand contains a zero bit, the corresponding bit of the word at C is stored in the corresponding position of the low-order product register.

Finally, the word formed in the low-order product register is stored in the location specified by the C address group. The result of the operation may differ from the result of an extract having the same operands only in those bit positions for which the B operand has a value of zero. The behavior of this instruction with one or more inactive addresses is unspecified.

For example, if the instruction

```
SS    OPERAND    EXMASK    RESULT
```

is executed where the contents of the three locations specified are as shown under the discussion of extract, the contents of location RESULT will be:

```
RESULT          101000100110------------
```

Half Add, HA

The half add instruction performs a binary addition of the 48-bit A and B operands in the accumulator, discarding all carries, and stores the result in the location specified by the C address group. This is equivalent to combining the corresponding bits of the A and B operands in accordance with the following rule:

> If the corresponding bit positions in the A and B operands have the same value, the result shall contain a zero in this position. In all other cases, the result shall contain a one. This is the "logical exclusive OR" function.

The behavior of this instruction with one or more inactive addresses is unspecified.

The half add instruction is illustrated in terms of the same example used to explain extract. If the instruction

```
HA    OPERAND    EXMASK    RESULT
```

is executed and the operands are the same as in the preceding examples, the contents of location RESULT will be

```
RESULT          110111010010------------
```

Superimpose, SM

The superimpose instruction performs a superposition of the 48-bit A and B operands in such a way that the result contains a one in every position in which a one existed in either A or B or both, and stores the result in the location specified by the C address group. This is equivalent to combining the corresponding bits of the A and B operands in accordance with the following rule:

> If corresponding bit positions of the A and B operands are both zero, the result shall contain a zero in this position. In all other cases, the result shall contain a one. This is the "logical inclusive OR" function.

The behavior of this instruction with one or more inactive addresses is unspecified.

For example, if the above operands are manipulated by the instruction

```
SM    OPERAND    EXMASK    RESULT
```

the contents of the location RESULT will be:

```
RESULT          110111110010------------
```

SECTION VIII

TRANSFER INSTRUCTIONS

The logic of the Honeywell 1800 includes six instructions designed to transfer data within the main memory, within the control memory, or from one memory to another. Using these instructions, called transfer instructions, the programmer may move any desired quantity of information, from a single bit to an entire record. Two of the instructions move single words only, or, when masked, fields within single words. The other four instructions move groups of words and cannot be masked.

The command codes for all six instructions provide the ability to address special registers and to designate either sequencing counter as the source of the next instruction. In instructions which transfer groups of words, the A address group indicates the location of the first word to be transferred, while the C address designates the location to which this word is to be delivered. Except for the multiple transfer (MT) instruction, transfers involving groups of words occur under control of special registers AU1 and AU2, which initially contain one of the following bit configurations, depending upon the type of addressing used in the A and C address groups, respectively.

1. Direct Memory Location Address: A positive sign bit, the bank indicator taken from the sequencing counter which selected the instruction, and the low-order 11 bits of the address group.
2. Direct Special Register Address: A positive sign bit, the group indicator associated with the sequencing counter which selected the instruction, and the low-order 11 bits of the address group.
3. Indexed Memory Location Address: A positive sign bit and the augmented low-order 15 bits of the index register referenced in the address group.
4. Indexed Special Register Address: A positive sign bit and the augmented low-order 15 bits of the index register referenced in the address group.
5. Indirect Memory Location Address: The sign bit (positive or negative) and 15-bit main memory address stored in the special register designated in the address group.
6. Indexed Indirect Memory Location Address: The sign bit (positive or negative) and 15-bit main memory address stored in the special register whose address is obtained by augmenting the contents of the index register referenced in the address group.

As successive words are transferred, the arithmetic control counters are automatically incremented (or decremented if the sign bit is negative) by one to specify a source and a result address for each word transferred. At the completion of the instruction, the counters contain addresses equal to their initial settings plus (or minus) the number of words transferred, i. e., the address of the last word transferred plus (or minus) one.

Transfer A to C, TX

This instruction transfers one word from the location specified by the A address group to the location specified by the C address group. The B address group is ignored. Any of the six types of addressing previously discussed may be used in either the A or C address group, provided the instruction is used in its unmasked version. When the instruction is used with a mask, partial words (or fields) may be transferred from one memory location to another, protecting the unmasked portion of the result location. The masked version of the instruction, of course, cannot address special registers.

The time required to execute an unmasked instruction with direct memory location addresses is three memory cycles. The timing effect of masking, indexing, and the use of special registers is summarized in Appendix C.

Transfer and Sequence Change, TS

This instruction transfers one word from the location specified by the A address group to the location designated by the B address group and changes the setting of the specified sequencing counter so that the next instruction is selected from the main memory location designated by the C address group. If the C address is active, the appropriate history register is changed after transfer of information from A to B but before the setting of the sequencing counter is changed. The instruction is also used, with inactive addressing, to provide access to the low-order product register (see "Inactive Addressing," Section IV). The instruction is identical to the TX instruction with respect to masking.

The A and B addresses may use any of the six types of addressing previously discussed. If the C address is active, it may specify a direct memory location address, an indirect memory location address, an indexed memory location address, or an indexed indirect memory location address. The behavior of the system for each of these cases is described below.

1. Direct Memory Location Address. The low-order 11 bits of the C address group are placed in the specified sequencing counter, together with the bank indicator from the counter which selected the instruction. The sign bit is set to a plus value.

2. Indirect Memory Location Address. The complete contents of the addressed special register, including the sign and bank indicator, are transferred to the specified sequencing counter. The contents of the addressed special register are incremented in the usual fashion after use, unless the special register addressed is the counter to be changed (either sequence or cosequence counter), in which case incrementing does not take place. The tabular bit in the address group is ignored.

3. Indexed Memory Location Address. The contents of the referenced index register are augmented, and the complete 15-bit address thus formed is inserted, with a positive sign, into the specified sequencing counter.
4. Indexed Indirect Memory Location Address. A complete special register address is generated by augmenting the contents of the referenced index register in the usual fashion. The generated address specifies a special register whose entire 16-bit contents are transferred to the specified sequencing counter. The contents of the addressed special register are incremented in the usual fashion after use, unless this special register is the counter to be changed (either sequence or cosequence counter). The tabular bit in the address group is ignored.

If a parity error is detected in a word selected from main memory during execution of this instruction, an unprogrammed transfer occurs and the contents of the specified sequencing counter are not changed. The contents of the appropriate history register will be changed, however, and the bisequence bit in the program control register (see page 110) will be set in response to the bisequence bit in the command code of the instruction.

The time required to execute an unmasked TS instruction with three direct memory location addresses is four memory cycles.

N-Word Transfer, TN

This instruction transfers the number of words specified by the high-order six bits of the B address group from consecutive locations starting at A to consecutive locations starting at C. The number of words to be transferred can range from 0 to 63. If the B address group is zero, no information is transferred. The low-order six bits of B are ignored. The transfer of information occurs under control of special registers AU1 and AU2, according to the conventions outlined at the beginning of this section.

It should be noted that if a special register is directly addressed in the A address of an N-word transfer instruction and an increment other than zero appears in the address group, then this increment is applied after transfer to the contents of each special register thus addressed. For example, consider the instruction

```
        TN    Z,X0, 10    5    Z,R0
```

This instruction causes the contents of index registers X0 through X4 to be transferred to special registers R0 through R4. At the conclusion of the instruction, the contents of index registers X0 through X4 will have been incremented by 10, and the low-order five bits of AU1 and AU2 will contain the subaddresses of X5 and R5, respectively.

When the instruction

```
        TN    N,X0, 10    5    N,R0
```

is executed, on the other hand, only the contents of X0 are incremented by 10 after use, since no other special register is referenced by the A address group. If X0 initially contained the main memory address tagged TRANS1, and R0 contained the main memory address tagged LOC1, then at the conclusion of the instruction AU1 will contain the address TRANS1 + 5 and AU2 will contain the address LOC1 + 5.

The time required to execute an N-word transfer with either direct or indirect memory location addresses is 5 + 2n memory cycles, where n equals the number of words transferred.

Multiple Transfer, MT

The multiple transfer instruction transfers the contents of the location specified by the A address group to the location specified by the C address group, repeating the transfer the number of times specified by the high-order six bits of the B address group. This number may range from 0 to 63. If B is zero, no transfer of information takes place. The low-order six bits of B are ignored.

Although all types of addressing are permitted with this instruction, the instruction is most meaningful when used with indirect addressing. For example, an area of 20 words in memory may be cleared to zeros by storing a constant of zeros in the location tagged ALLZEROS, setting general purpose register R0 to the address of the first location to be cleared, and executing the instruction

```
MT      ALLZEROS      20      N,R0,1
```

Zeros will be transferred to the 20 main memory locations starting with the address initially contained in R0, and the contents of R0 at the completion of the instruction will be equal to the initial contents plus 20. The arithmetic control counters are not involved in the execution of this instruction.

As a second example, consider the instruction

```
MT      N,X0,10      5      N,R0,1
```

If X0 initially contains a memory address tagged ONESTORE and R0 contains a main memory address tagged WORKAREA, execution of this instruction causes the words from locations ONESTORE, ONESTORE + 10, ONESTORE + 20, etc., to be transferred to WORKAREA, WORKAREA + 1, etc. At the conclusion of the instruction, X0 contains the address ONESTORE + 50, while R0 contains the address WORKAREA + 5.

The time required to execute a multiple transfer instruction with either direct or indirect memory location addresses is 1 + 2n memory cycles, where n equals the number of times the transfer is performed.

Record Transfer, RT

The record transfer instruction is used to move a group of related words from one location to another. Such a group does not necessarily constitute a record. Although the record transfer instruction is most frequently used to manipulate "records," it may also be used to advantage whenever the number of words to be transferred is greater than 63, the maximum number which can be moved with the N-word transfer. In fact, the only restriction on the number of words which can be transferred is the practical limitation of available storage space in the memory.

When a record transfer instruction is executed, an end-of-record word[1] is stored in the location specified by the B address group. Consecutive words are then transferred from the location starting with A to consecutive locations starting with C, until an end-of-record word is transferred. The behavior of the system is unspecified when either the A or the C address is a direct or indexed special register address.

Note that the transfer of information is stopped whenever an end-of-record word is transferred, regardless of where this word was stored. In other words, the word stopping the transfer is not necessarily the same word which the instruction has stored in the location specified by the B address. The transfer also stops if a parity failure occurs. At the completion of the instruction, AU1 will contain the complete address generated from the A address group plus the number of words actually transferred; AU2 will contain the complete address generated from the C address group plus the number of words transferred.

The time required to execute a record transfer instruction with direct or indirect memory location addresses is 7 + 2n memory cycles, where n is the number of words transferred.

Item Transfer, IT

The item transfer instruction operates exactly as the record transfer, with the exception that instead of storing an end-of-record word, an end-of-item symbol is substituted for the high-order 32 bits of the contents of B, clearing the low-order 16 bits to zeros. The transfer stops with the transfer of an end-of-item or end-of-record word, or with a parity failure.

As described in Section III, an end-of-item word is any word whose high-order 32 bits are identical to those of an end-of-record word. The comments made with respect to the record transfer instruction are equally applicable to item transfer. The time required to execute the instruction is also the same.

[1] As previously stated, an end-of-record word is a word whose 48 information bits are:
1010 1010 0000 0000 1110 1110 1110 1110 1101 1101 1101 1101

SECTION IX

DECISION INSTRUCTIONS

The Honeywell 1800 logic includes four decision instructions which are used to branch to an alternate path of control in response to a precisely defined condition. Each of these instructions looks for a special condition based upon the relationship between the contents of the location specified by the A address group and the contents of the location specified by the B address group. If the condition is met, the sequencing counter designated as the source of the next instruction is changed so that the next instruction is selected from the main memory location specified by the C address group. If the condition fails, the current setting of the designated sequencing counter selects the next instruction.

Any type of addressing may be used in the A and B address groups of these instructions unless they are masked. The C address group may contain a direct memory location address, an indirect memory location address, an indexed memory location address, or an indexed indirect memory location address. If the condition is met and the sequencing counter is changed, the behavior of the system for each of the four possible types of C address is the same as that described for the transfer and sequence change instruction (see Section VIII, pages 80 and 81). If the C address is inactive, no change is made in either sequencing counter and hunting for the next program in demand is inhibited. If the C address is active and the condition is met, hunting is also inhibited. If the condition is not met, a normal hunt is made for the next program in demand.

The accumulator is used in executing the decision instructions. Following the completion of such an instruction, the word in the accumulator is invalid. Any attempt to deliver this word to the memory will therefore cause a control error (see Appendix D).

If a parity error is detected in a word selected from main memory during execution of a decision instruction, an unprogrammed transfer is executed to U + 2 or U + 3, and the contents of the sequencing counter are not altered. However, the contents of the appropriate history register may be changed: if the condition is met, the history register will be changed; if the condition fails, the history register will not be changed. In either case, the bisequence bit in the program control register (see page 110) is set in response to the bisequence bit in the command code of the instruction.

Two of the four decision instructions are inequality comparisons (one alphabetic, one numeric) in which the two operands are examined for inequality ($\neq$). If they are not equal, the next instruction is selected from the location specified by the C address group. The other two instructions (one alphabetic, one numeric) are called "less than" comparisons, since the two operands are inspected to determine whether the contents of A are less than or equal to the contents of B ($\leq$). If so, the next instruction is selected from the location specified by C. All four instructions may compare either the entire contents of both words (unmasked version) or respective fields of the words, as defined by a mask. The location of the mask, if any, is partially specified by bits of the command code.

The alphabetic comparisons make a bit-by-bit comparison of all 48 information bits of the words at locations A and B. If the word at A has a zero bit in the first position where they differ, reading from left to right, it is considered "alphabetically less" than the word at B. The order of the alphanumeric characters is shown in Table I, page 167. When the instructions are masked, the decision is based upon a bit-by-bit comparison of the selected fields of the operands.

The numeric comparisons treat both operands as signed 11-digit numbers and make a bit-by-bit comparison of information bits 5 through 48 of the contents of A and B. Thus, the 11 digits of the signed numeric words are compared as in the alphabetic comparisons. However, the sign bits (bits 1 through 4) of both words are checked before a final decision is made. If any of the four bits is a one, the sign is considered to be positive; if all four bits are zeros, the sign is negative. Any positive number is larger than any negative number, with the exception that two words with 11 low-order zero digits are considered equal regardless of sign. When two negative numbers are compared, the one with absolute value closer to zero is considered greater. Thus, the numeric comparisons are true algebraic comparisons of signed 11-digit numbers.

The time required to execute an unmasked decision instruction with direct memory location addresses is four memory cycles, regardless of whether or not the condition is met. Reference should be made to the notes in Appendix C for the effect on timing of masking, indexing, and special register addressing.

Inequality Comparison, Alphabetic, NA

This instruction compares for inequality the 48-bit contents of the word at A and the 48-bit contents of the word at B. If the two words are not identical, the sequencing counter specified as the source of the next instruction is changed so that the next instruction is selected from the location designated by the C address group. If they are equal, the next instruction is selected from the location specified by the current contents of the designated sequencing counter. Plus zero is not equal to minus zero.

<u>Less Than or Equal Comparison, Alphabetic, LA</u>

This instruction also makes a bit-by-bit comparison of the 48-bit word at A and the 48-bit word at B. If the contents of the location specified by the A address are less than or equal to the contents of the location specified by B, the designated sequencing counter is changed so that the next instruction is selected from the location specified by the C address group. If the word at address A is greater than the word at address B, the next instruction is taken from the location specified by the current contents of the designated sequencing counter. Plus zero is greater than minus zero.

<u>Inequality Comparison, Numeric, NN</u>

The numeric inequality comparison examines the contents of the locations specified by A and B, regarded as signed words, to determine whether they are algebraically unequal. If they are unequal, the designated sequencing counter is changed to select the next instruction from the location specified by the C address group. If the words are equal, the address of the next instruction is selected from the location specified by the current contents of the designated sequencing counter. In a numeric inequality comparison, plus zero equals minus zero.

<u>Less Than or Equal Comparison, Numeric, LN</u>

When this instruction is executed, the contents of the locations specified by the A and B address groups are regarded as signed words and compared to determine whether the word at A is algebraically less than or equal to the word at B. If the A operand is less than or equal to the B operand, the designated sequencing counter is changed to select the next instruction from the location specified by the C address group. If the contents of A are algebraically greater than the contents of B, the next instruction is selected from the location specified by the current contents of the designated sequencing counter. Plus zero equals minus zero in a numeric less than or equal comparison.

SECTION X

SHIFT INSTRUCTIONS

The ability to pack as many as four separate pieces of signed information (or a greater number of pieces of unsigned information) into a single Honeywell 1800 word would have little practical value without the facility for manipulating each piece of information (or field) independently. In part, this facility is provided by the masking feature of the system, which permits certain instructions to operate on fields in preassigned positions within words. In order to achieve maximum flexibility in handling packed words, however, it is also necessary to provide the means for rearranging the positions of fields within words. This flexibility has been achieved in the Honeywell 1800 by the inclusion of five shift instructions. Two of these instructions shift the low-order 44 bits of a word, preserving sign bit positions 1 through 4. The remaining three shift the entire 48 bits of the word.

The shift instructions cause the contents of the location specified by the A address group to be placed in the accumulator, where they are shifted right the number of bit positions specified by the high-order six bits of the B address group. The direction of the shift is always right end around, with the result that bits shifted out of the low-order position recirculate to the high-order position of the word. A 48-bit full word shift, therefore, produces the same result as a 0-bit shift. The effective number of bit positions by which a word may be shifted ranges from 0 to 44 for shifts protecting the sign of the operand, and from 0 to 48 for shifts of the entire word. If the number of bit positions specified to be shifted exceeds the maximum effective shift (it is possible to specify a 63-bit shift in the high-order six bits of B), the machine will actually perform the specified shift. The net number of places shifted as a result of the operation, however, will be the number specified minus 44 in the case of shifts protecting sign or the number specified minus 48 in the case of shifts not protecting sign.

The shift instructions are inherent mask instructions; that is, they are always performed with a mask. The mask is applied to the shifted operand before delivery to the result location (specified by the C address group). If the programmer wishes to deliver the entire word, therefore, he must specify a mask consisting of 48 binary ones. The location of the mask is designated by the low-order six bits of the B address group, in conjunction with bits 2 through 5 and 6 through 10 of the mask index register (see Figure V-3, page 57). The two shift and substitute instructions and the shift and select instruction perform protected masking. This means that the values of the unmasked bit positions in the result location (those which do not correspond to binary ones in the mask) are preserved during the operation. The two shift and extract instructions perform unprotected masking; that is, the positions in the result location which do not correspond to binary ones in the mask are cleared to zeros. Protected masking is accomplished by gating the shifted contents of the accumulator and the original contents of the location specified by the C address group into the low-order product register under control of the mask in the mask register (see page 76). Unprotected masking is accomplished by gating the contents of the accumulator and a generated word of all zeros into the low-order product register under control of the mask in the mask register. In either case, the contents of the low-order product register are then delivered to the location specified by the C address group.

Any type of addressing may be used in the A and C address groups of the shift instructions, with the exception that certain restrictions apply to the C address of a shift and select instruction. These will be discussed under the description of that instruction. If the B address group of a shift instruction is all zeros, then no shifting is performed and the word at A is transferred to the location specified by C under control of a mask located by attaching six low-order zero bits to the partial address stored in the mask index register. If the B address group is inactive, the behavior of the system is unspecified.

ARGUS notation for the shift instructions is designed to simplify the specification of masks and shifts. If the mask is designated by the programmer, its symbolic tag is written in the command code field (rather than in the B address field), separated from the operation code by a comma. Reference should be made to the ARGUS Manual of Assembly Language for details on the generation of masks by ARGUS. The nature and extent of the shift is specified by three items of information in the B address field, all separated by commas:

1. A character to designate the type of characters shifted, i.e., A, alphanumeric; D, hexadecimal; B or blank, binary.
2. A number to indicate the number of positions to be shifted (0 to 8, alphanumeric; 0 to 12, hexadecimal; 0 to 48, binary).
3. A character to designate the direction of the shift as left, L, or right, R or blank.

Any valid ARGUS address format may be used in the A or C address of a shift instruction, subject only to the restrictions noted in connection with the C address of a shift and select instruction.

Shift and substitute and shift and extract instructions with direct and/or indirect memory location A and C addresses require a basic execution time of five memory cycles plus a variable number of memory cycles required for the actual shifting operation. The shift logic in the system is so designed that the machine shifts a word in multiples of 1, 4, or 16 bits. Each of these shifts requires the same amount of time; i.e., a 16-bit shift takes no longer than a 1-bit shift. Two such shifts may be performed in a memory cycle. Thus, a 32-bit shift requires a single memory cycle, as does a 20-bit shift or a 5-bit shift. The total time required to shift the operand the specified number of bits, therefore, is a function of the number of individual shifts which the machine must perform. On this basis, the number of memory cycles required for actual shifting of an operand ranges from 0 to 4, so that the time required for complete execution of the instruction varies from a minimum of five to a maximum of nine memory cycles.

The shift and select instruction requires a basic execution time of six memory cycles plus the number of cycles required for the shift itself. The total time to execute the instruction, therefore, varies from six to ten memory cycles.

The exact times required for each number of positions shifted, as well as variations resulting from indexing or direct special register addressing, are listed in Appendix C.

Shift Preserving Sign and Substitute, SPS

This instruction directs the machine to shift the word at A, excluding sign bits 1 through 4, right end around, the number of bit positions specified by the high-order six bits of the B address group. Bits shifted out of position 48 recirculate to bit position 5. The result is masked using a mask whose address is generated from the low-order six bits of B and the partial address stored in the mask index register, as illustrated in Figure V-3. The masked result is delivered to the location specified by C, protecting the bit positions of the contents of C which do not correspond to binary ones in the mask.

A shift of n bits to the left is specified in machine language by setting the high-order six bits of B equal to 44-n (in binary). Thus, a 12-bit shift to the left is effected by specifying a 32-bit shift (44 minus 12) to the right. In ARGUS language, as previously noted, the programmer may specify a left shift directly.

Shift Preserving Sign and Extract, SPE

This instruction is identical to shift preserving sign and substitute, with the exception that the bit positions in the location specified by C which do not correspond to binary ones in the mask are not protected but are cleared to zeros when the result is stored.

Shift Word and Substitute, SWS

This instruction directs the machine to shift the entire word at A, including sign, right end around, the number of bit positions specified by the high-order six bits of B. The result is masked using a mask whose address is generated as in shift preserving sign and substitute. The masked result is delivered to the location specified by C, protecting the bit positions of the contents of C which do not correspond to binary ones in the mask.

A shift of n bits to the left is specified in machine language by setting the high-order six bits of B equal to 48-n (in binary). Again, a left shift may be specified directly in ARGUS language.

Shift Word and Extract, SWE

This instruction is identical to shift word and substitute, with the exception that the bit positions in the location specified by C which do not correspond to binary ones in the mask are not protected but are cleared to zeros when the result is stored.

Shift Word and Select, SSL

The shift word and select instruction causes the machine to shift the entire word at A, right end around, the number of bits specified by the high-order six bits of B. The result of the shift is masked, using a mask whose address is generated as in shift preserving sign and substitute, and this masked result, with a positive sign implied, is added in binary to the address specified in C to form a new address. The sequencing counter designated as the source of the next instruction is then changed so that the next instruction is selected from the location specified by the modified C address group. Regardless of the mask used, the machine never adds more than 11 low-order bits to the C address.

The C address of a shift and select instruction may be a direct memory location address, an indirect memory location address, an indexed memory location address, or an indexed indirect memory location address. For these four types of address, the masked result, called C', is used with the C address as described below:

1. Direct Memory Location Address. C' is added to the low-order 11 bits of C, discarding the end carry, if any. The resulting 11 bits are placed in the specified sequencing counter, together with the sign and bank indicator from the counter that selected the current instruction.
2. Indirect Memory Location Address. C' is added to the low-order 11 bits of C, discarding the end carry, if any. The result is used to select a special register whose contents, including the sign and bank indicator, are transferred to the specified sequencing counter. The tab bit of the modified C address is ignored. The contents of the addressed special register are incremented in the usual fashion after use, unless the special register addressed is the counter to be changed (either sequence or cosequence counter), in which case incrementing does not take place.

3. Indexed Memory Location Address. The contents of the referenced index register are augmented in the usual fashion to form a complete 15-bit address. C' is then added to the low-order 11 bits of this complete address, discarding the end carry, if any. The resulting complete 15-bit address is transferred, with a positive sign, to the specified sequencing counter.

4. Indexed Indirect Memory Location Address. A complete special register address is generated by augmenting the contents of the referenced index register in the usual fashion. C' is added to the low-order 11 bits of this generated address, discarding the end carry, if any. The result of this addition is used to select a special register whose contents, including the sign and bank indicator, are transferred to the specified sequencing counter. The tab bit in the generated special register address is ignored. The contents of the special register selected are incremented in the usual fashion after use, unless this special register is the counter to be changed (either sequence or cosequence counter).

Since the shift and select instruction permits the addition of as many as 11 bits to the low-order bits of the C address, it provides the ability to transfer control to any one of 2048 memory locations.

If a parity error is detected in a word selected from main memory during execution of a shift and select instruction, an unprogrammed transfer is made to U + 2 or U + 3, and the contents of the sequencing counter are not altered. However, the contents of the appropriate history register will be changed, and the bisequence bit in the program control register (see page 110) will be set in response to the bisequence bit in the command code of the instruction.

SECTION XI

PERIPHERAL INSTRUCTIONS

One of the unusual features of the Honeywell 1800 is that all peripheral devices look the same to the central processor. The same 6-bit operation code is used to read from any input device, and another 6-bit code defines the instruction to write on an output device. These two instructions are read forward (RF) and write forward (WF), respectively. The other two peripheral instructions are read backward (RB), used only with magnetic tape units, and rewind (RW), which may be used with magnetic tape units and paper tape readers.

In machine language, a peripheral operation code is specified by the low-order six bits of the command code, while the high-order six bits designate one of 64 possible devices to be addressed. More exactly, the high-order three bits designate a channel through which the information will travel (one of eight input or eight output channels, the direction determined by the operation code), while the following three bits specify a particular device attached to this channel from which information will be read or upon which it will be written. The assignment of peripheral codes to magnetic tape units and other input-output devices is established individually at each Honeywell 1800 installation, according to the general rules laid down under "Systems Configurations" (Section II). From the table of peripheral assignments, the programmer must select a device which can perform the specified operation. For example, a write instruction cannot be executed by a card reader. Neither can a printer read backward nor a punch execute a rewind.

Addressing a tape control not connected to the central processor results in a control error (see Appendix D). If the tape control is properly connected but a particular tape address designator line is disconnected or plugged into an inactive hub on the patchboard, the corresponding tape unit presents a continuous busy signal at the tape control. If a tape unit is in a busy status when it is addressed, the program will stall until it becomes available. It is still possible, however, for a different program to execute a peripheral instruction addressed to another tape unit on the same tape control.

The peripheral commands in ARGUS language are reversed in terms of machine language; the operation code is written first, followed by the address of the peripheral device, expressed in an alphabetic code which ranges from AA to HH. Thus the instruction

```
WF,CD    OUTPUT    -    -
```

directs the machine to write one record, stored in memory beginning at location OUTPUT, on device CD, which may be a magnetic tape unit or any peripheral output device, depending upon the assignment of the code CD at the particular installation.

Since the entire 12 bits of the command code are used to specify the operation code and the device address, a peripheral instruction cannot address a special register explicitly or specify an alternate sequencing counter as the source of the next instruction. A peripheral instruction, therefore, is always followed by an instruction from the same sequencing counter which selected the peripheral instruction. The A address group of a read or write instruction specifies the memory location into which the first word of a record is to be read or from which the first word of a record is to be written. The B address group is used only when the instruction is to sense for end-of-item words in a magnetic tape operation and read the separate items into or write them from non-sequential areas in memory. In this case, called distributed reading or writing, the B address group specifies the memory location of the first entry in a table of addresses which denotes the starting locations for the second item and all subsequent items to be read or written. Finally, the C address group may be used, at the programmer's option, to specify a new setting for the sequencing counter which selected the current instruction. All three address groups may specify either direct or indexed main memory addresses. If the C address is active, the behavior of the system follows that described for direct memory location or indexed memory location addressing with the C address of the transfer and sequence change instruction (see Section VIII, page 80).

If address A of a read or write instruction is inactive, the addressed device is tested for interlocks and errors (see below), but the device is not activated nor does any transfer of information take place. The first read instruction following a rewind or write instruction addressed to the same magnetic tape unit must have an active A address. Similarly, the first write instruction following a rewind or read instruction to the same tape unit must have an active A address. Violation of this requirement can result in partial erasure of information on tape. If address B is inactive, end-of-item words are not sensed and the record is read into or written from successive memory locations. If the C address is inactive, the contents of the sequencing counter are not changed, and no hunt is made for another active special register group. Since a hunt for another program demand is never made after an instruction which implicitly changes the contents of the sequencing counter, it follows that under no condition does a hunt occur after execution of a read or write instruction.

The time required by the central processor to handle a read or write instruction to any device is determined in the following fashion. Five memory cycles are required to interpret a peripheral instruction whose A or B address (or both) is active and whose C address is inactive. If the C address is active but not indexed, two additional memory cycles are required. Indexing of the A and B addresses has no effect on timing; however one additional cycle is required if the C address is indexed. Thus, interpretation of a peripheral instruction with direct memory location A, B, and C addresses requires seven memory cycles; with all three addresses indexed it requires eight cycles. Complete details of the interpretation times for peripheral instructions are shown in Appendix C. In addition to the time required to interpret the instruction, one memory cycle is used for each word transferred from device to storage or from storage to device and an additional cycle is required for each item handled in distributed item reading or writing. The memory cycles used for information transfer are not consecutive but are allotted one at a time by traffic control, as described in Section II.

Read Forward, RF

When a read forward instruction is interpreted by the central processor, the main memory address generated from the A address group is sent, with a positive sign, to the read address counter associated with the input channel to which the addressed device is attached. The unit itself is signalled in order to initiate the mechanical operations, and then the buffer in the tape control or the peripheral control starts to receive information. From the time the unit is signalled until all the information has been transferred to memory, the corresponding input buffer interlock bit in the program control register is set to one (see Section XII, page 110). When a word in the buffer is ready to be transmitted to memory, the appropriate traffic control demand is turned on to indicate that a word may be brought in through this channel. As successive words are transmitted to memory from magnetic tape or from a punched card, the read address counter is stepped after each transfer so that it always specifies the location of the next word.

Once a read instruction is initiated, it stops only when it has completed the required reading. This condition is indicated by sensing the end-of-record gap on magnetic tape, by reading one punched card's worth of information, or by reading one frame from paper tape. However, there are several conditions that will inhibit the instruction and prevent its execution. If for any reason the device is not available to the machine, including the condition in which either the device or the buffer is busy, or if an error occurred on the preceding read from that device, no part of the read instruction is executed. Neither tape nor card is moved, no information is transferred, and if the instruction calls for a change of sequence, the change is not made. EXCEPTION: A read instruction with an inactive A address is not inhibited by a buffer busy condition, provided the device addressed is not busy.

Sensing an error during reading will not prevent completion of the instruction (except during a distributed read, as explained below). Instead, when an error is sensed in the incoming information, a parity error flip-flop is set and the read continues normally until all of the information has been stored in memory. When the next read instruction directed to this device tests the setting of the flip-flop and finds that an error was sensed during the previous read, the flip-flop is reset and execution of the instruction is inhibited. The instruction is stored in U or U + 1, an octal one (001) is transmitted to the central processor and inserted as the high-order three bits of the program control register (see Section XII), and an unprogrammed transfer is made to U + 6 or U + 7, depending upon whether the instruction was selected by the sequence or cosequence counter (see Figure V-5, page 62). The instructions stored in U + 6 and U + 7 serve as entries to a subroutine designed by the programmer to handle the error condition. Since the location to which the unprogrammed transfer is made reflects the sequencing counter which selected the instruction and since the instruction itself is stored (in U or U + 1), the programmer can determine where he is in his program and on which device the error was detected. From this information, he can determine how to proceed, in accordance with the various techniques discussed under the instructions compute orthocount and check parity (in Section XII and in Appendix B).

Another type of unprogrammed transfer occurs as a result of reading the first end-of-tape record on magnetic tape. The end of tape is physically marked by two optical windows, 32 inches apart. When the first window is sensed during tape recording, the record being written is completed. The beginning of the next record, called the first end-of-tape record, is written only after the second window has been sensed. When the tape is read forward, an unprogrammed transfer to U + 4 or U + 5 occurs on the instruction that reads the first end-of-tape record. The instruction itself is correctly executed and stored in U or U + 1; however, neither the contents of the sequencing counter nor those of its associated history register are changed if the C address is active, since the change in the sequencing counter does not take place until cycle zero of the next instruction. Every read forward instruction with an active A address occurring beyond the first end-of-tape record also creates an unprogrammed transfer to U + 4 or U + 5. Each such instruction is also executed correctly and stored in U or U + 1. An end-of-tape unprogrammed transfer never occurs on a read forward instruction with an inactive A address.

If any record in the neighborhood of the first end-of-tape record contains more than 2048 words or fewer than seven words, an unprogrammed transfer may occur one record earlier or one record later than the first end-of-tape record. The number of end-of-tape records which may be written, and hence read, after the first end-of-tape record is limited only by the length of the oxide trailer (see description of write forward instruction). If an error is sensed while reading any of these records, the next read instruction is inhibited and an error unprogrammed transfer occurs instead of a transfer to U + 4 or U + 5. When the end of tape is sensed on a paper tape reader, the reader presents a busy signal to the central processor so that any attempt to read beyond this point causes the program to stall.

The preceding discussion is based on a normal read instruction in which the B address is inactive and information is read into successive locations in memory. When magnetic tape is being read, an active B address specifies a distributed read, in which individual items of a record are read into non-sequential areas of memory. The central processor performs an additional operation in setting up such an instruction. As usual, the appropriate read address counter (RAC) is set to the main memory address generated from the A address group. The main memory address generated from the B address group is then sent, with a positive sign, to the corresponding distributed read address counter (DRAC). As the information is transmitted to memory, each successive word is placed in the next higher memory location. During this transmission, the central processor senses the input for an end-of-item word — a special symbol having the high-order 32 bits identical to the high-order 32 bits of an end-of-record word. When such a word is sensed, it is placed in proper sequence relative to the words which preceded it. The contents of the RAC are then replaced with the <u>contents</u> of the memory location specified by the DRAC, the contents of the DRAC are incremented by one, and transmission of information proceeds with the first word of the next item placed in the location specified by the new contents of the RAC. It is the programmer's responsibility to see that each of the memory locations whose addresses appear successively in the DRAC is loaded with a constant whose low-order 16 bits represent the desired sign and main memory address.

As an example, a record on tape CD consists of five items to be loaded into discrete sections of the memory, using the following instruction:

```
RF, CD    ITEMA    TABLE    -
```

The locations specified by TABLE, TABLE + 1, TABLE + 2, and TABLE + 3 contain a series of addresses (tagged ITEMB, ITEMC, ITEMD, and ITEME), each with a positive sign bit, which designate the starting location in memory for each of the last four items. When the instruction is interpreted, the RAC is set initially to the memory location tagged ITEMA and the DRAC is set to the address tagged TABLE. The first item, including the end-of-item word, is read into the successive locations ITEMA, ITEMA + 1, ITEMA + 2, etc. When the end-of-item word is sensed, the <u>contents</u> of TABLE are placed in the RAC and the DRAC is stepped to TABLE + 1. The next item is then loaded into ITEMB, ITEMB + 1, etc. When the end-of-item word is sensed, the <u>contents</u> of TABLE + 1 are transferred to the RAC and the DRAC is incremented to TABLE + 2. This process continues until the entire record has been read into memory. Thus, the third item is stored in successive locations starting with ITEMC while the last item is stored in successive locations starting with ITEME.

When an error is detected during a normal read, the read is completed before the parity error flip-flop is set. In a distributed read operation, however, no further information is transmitted to memory after an error is sensed, although the tape is allowed to move to the end of the record. As with the normal read, the next read instruction to the tape results in an unprogrammed transfer. If the programmer then wishes to reconstruct the record using the orthocorrection technique, he must reread it by means of a normal (non-distributed) read.

Whether the programmer uses a normal or a distributed read, he must remember that reading from any peripheral device proceeds concurrently with computation. This means that he must be certain a record has indeed been read into memory before he begins to operate upon the information. Several techniques are available for ascertaining that a read operation has been completed. The most obvious, perhaps, is to read records alternately into one of two buffer areas in memory. Thus, while information is being read into one area, information in the other area may be processed.

A shortcoming of this technique, especially if records are lengthy or if several tapes are being handled (as in a sort routine), may be the amount of memory required. If memory must be conserved, other options are available for checking the completion of a read operation. One of these is to give a read instruction with an inactive A address. This technique, which results in no tape movement and no transfer of information, not only insures completion of the previous read instruction but also guarantees (if no unprogrammed transfer occurs) that the information has been correctly read. Another technique by which the programmer may test the status of a peripheral operation is by sensing the interlock bit in the program control register to see whether the corresponding buffer is busy. This technique is safe, however, only if the pertinent buffer is handling a single device during the current run. A third approach involves the use of the read address counter. If the programmer is working with fixed-length records, he knows what address should be found in the RAC when the entire record has been brought into memory. By addressing the RAC explicitly, he can compare its contents repeatedly with a constant representing its final contents to determine when the instruction has been completed. These techniques are not equally appropriate for all situations. The nature of the problem and the ingenuity of the programmer, therefore, determine which approach will be most satisfactory.

Read Backward, RB

The read backward instruction, as its name implies, causes a magnetic tape to be read in the reverse direction. It can only be executed by a magnetic tape unit. The instruction is otherwise very similar to the read forward instruction, hence this discussion is limited to the dissimilar features.

The normal read backward instruction is implemented so that a record read in the reverse direction is stored in memory exactly as if it had been read forward, provided the A address group of the instruction specifies the memory location into which the last word of the record would have been placed by the read forward. This is accomplished as follows. When the instruction is interpreted, the main memory address generated from the A address group is inserted, with a negative sign, into the read address counter. As the counter is stepped with each successive word transfer, the effect is to decrement the counter so that each word is read into the next lower memory location until the record gap is reached.

When a distributed read backward instruction is executed, the main memory address generated from the B address group is placed in the distributed read address counter with a negative sign, so that this counter too is effectively decremented as successive items are handled. Thus, the machine not only stores successive words of an item in reverse order, but also sequences backward through the table set up for the distributed read. Since the contents of the locations listed in the table are transferred to the RAC, the addresses stored in these locations must contain negative sign bits (bit 33 of the main memory words) if the words of each successive item are to be placed in memory in reverse order. It should be noted that the end-of-record word, the first word to be brought into memory by a read backward instruction, does not effect a change in the contents of the RAC despite the fact that its high-order 32 bits have the same configuration as an end-of-item word. Instead, the end-of-record word, the two orthotronic words, and the last item of the record (through the end-of-item word for the penultimate item) are stored as one item.

When an error is sensed during a read backward (normal or distributed), the unprogrammed transfer conditions are identical to those described for the read forward instruction. The unprogrammed transfer convention associated with the beginning-of-tape condition, however, is slightly different from the end-of-tape convention which applies when tapes are read forward. When a tape is read backward, a beginning-of-tape unprogrammed transfer to U + 4 or U + 5 occurs on the instruction which reads the first record written on the tape, and the instruction is correctly executed. Correct execution of the instruction, however, does not include a change in the contents of the sequencing counter or its associated history register if the C address is active. Any read backward instruction with an active A address subsequently issued to this tape will also cause a beginning-of-tape unprogrammed transfer, as will a read backward instruction with active A address issued to a tape after rewinding. A read backward instruction at the beginning of tape moves the tape about 1/8 inch.

When a write instruction is given to a tape in rewound condition, the first record is written immediately following the clear leader. The beginning of the second record is not written until the head has reached an optical mark a fixed distance from the leader. If the first record written contains more than 2048 words, the beginning-of-tape unprogrammed transfer to U + 4 or U + 5 may fail to occur when this record is read backward. Furthermore, if the second record contains fewer than seven words, the unprogrammed transfer may occur upon reading the second record backward rather than the first.

Write Forward, WF

When a write forward instruction is interpreted by the central processor, the main memory address generated from the A address group is inserted, with a positive sign, in the write address counter (WAC) associated with the output channel to which the addressed device is attached. The unit is signalled, the output buffer interlock bit is set to one, and the first word is selected from the location specified by the WAC and sent to the buffer in the tape control or the peripheral control. The contents of the WAC are incremented, and when the buffer is empty, the traffic control demand for this channel is turned on so that the next word may be transmitted to the buffer.

If the instruction is directed to a magnetic tape unit, the write operation continues until an end-of-record word is sensed by the central processor. In other words, an instruction to write on tape starts with the word in the location specified by the A address group and continues through successively higher memory locations until an end-of-record word is encountered. Ifthe instruction addresses a printer or a card punch, the writing stops when 16 words have been delivered to the printer, or when 11 words (alphanumeric mode) or 21 words (transcription mode) have been sent to the card punch, regardless of the presence or absence of an end-of-record word. If the instruction addresses a paper tape punch, a single frame of paper tape is punched from the memory location specified by the A address group.

A continuous initial segment of magnetic tape is formed or extended by writing a sequence of records, uninterrupted by read or rewind instructions to the tape being written, and starting from a proper initial condition. A proper initial condition is defined as:

1. Following a rewind instruction; or
2. Following a read backward instruction over a record which is already part of a continuous initial segment.

Thus, a continuous initial segment always consists of an unbroken sequence of records, beginning with the first record on tape. Information on a continuous initial segment of tape may be freely read forward or backward, without restriction.

Information previously recorded beyond the continuous initial segment of a tape is not necessarily recoverable. In particular, any attempt to read the first record following a continuous initial segment is likely to result in an error. One or more records of information previously recorded in the region beyond the continuous initial segment may have been destroyed, and spurious information may have been recorded. However, once the region of erroneous information adjacent to the continuous initial segment has been passed, correct information is recoverable up to the end point of the previously existing continuous initial segment.

In the unique instance when the continuous initial segment consists precisely of the first record on tape and is less than 2048 words long but has at least seven more words than the information previously recorded there, the information in the second record is not destroyed, and not more than one read forward instruction after reading the first record forward will result in an error condition because of spurious information. Under these conditions, reading the first record in reverse will not generally be possible without error.

Part of the checking process which occurs during tape reading and writing involves a longitudinal or channel check based on the assumption that two correct orthowords are included in every record read or written. If these words are not included with the record, the machine will in most cases indicate an error when the record is read or written. Before writing a record on tape, therefore, it is customary to compute the two orthowords to be included with the record, using the compute orthocount instruction described in Section XII. This computation is omitted at the risk of an error indication, unless the programmer is certain that the record was read into memory with correct orthocount and has not since been altered in any way. When a record is printed or punched on-line, however, the question of orthowords is irrelevant, since no orthocheck is performed.

The same conditions which prevent execution of a read instruction also inhibit the write instruction. If either the device or the buffer is busy, or if an error occurred on the previous <u>write</u> instruction to that device, no part of the write instruction is performed. Again, a write instruction with an inactive A address is not inhibited if only the buffer is busy. When an error is detected in information going from memory to a magnetic tape unit printer, or card punch, a parity error flip-flop is set and the write is completed. In the case of a paper tape punch, the current frame is not punched and tape is not advanced. The next <u>write</u> instruction to the same device is stored in U or U + 1, an octal one (001) is inserted as the high-order three bits of the program control register and an unprogrammed transfer is made to U + 6 or U + 7. Since the instruction itself is stored, the programmer will have no difficulty in distinguishing between a write error and a read error, even though the unprogrammed transfer is made to the same location for both types of error. Note that because of the timing of the punch operation, an additional card will have been punched after the erroneously punched card before this unprogrammed transfer is made. A corrective error routine for punched output, therefore, should insure that two cards are repunched.

A special condition may result in an unprogrammed transfer to U + 6 or U + 7 during an instruction to write on magnetic tape. Each reel of tape has provision for manually inserting a ring in the hub; unless this ring is in place, the tape is protected from any attempt to write on it. The tape may also be protected, even when the ring is in place, by setting a switch on the tape drive to the position labelled PROTECT. If an attempt is made to write on a tape protected in either of these ways, the instruction is completely executed, the tape is moved normally, but no writing occurs. An unprogrammed transfer to U + 6 or U + 7 results on the next write instruction to this tape.

The end-of-tape unprogrammed transfer which occurs when writing is the same as when reading forward. The instruction that causes the first end-of-tape record to be written is stored in U or U + 1, and an unprogrammed transfer is made to U + 4 or U + 5. No change is made in the contents of the sequencing counter or its affiliated history register if the C address of the instruction is active. A write instruction with active A address occurring beyond this record is also executed correctly and creates the same unprogrammed transfer conditions. An end-of-tape unprogrammed transfer never occurs on a write instruction with inactive A address. The number of records that can be correctly written after the first end-of-tape record is limited only by the length of the oxide trailer. A hardware interlock stops tape motion when the end of tape is reached. When this interlock is activated, the tape unit presents a busy signal to the central processor so that any attempt to write beyond this point causes the program to stall.

When punching paper tape, the end-of-tape unprogrammed transfer occurs on each write instruction directed to the punch after the end-of-tape condition has been sensed. This occurs when the detector on the supply reel senses that approximately 20 feet of tape remains to be punched or when the detector on the takeup reel senses that this reel is full. In the end-of-tape condition, each write instruction directed to the punch is executed and stored in U or U + 1, the frame is punched, and an unprogrammed transfer is made to U + 4 or U + 5. The number of instructions that can be executed in this condition is limited only by the amount of tape on the supply reel.

If the B address group of a write forward instruction to magnetic tape is active, a distributed write operation is performed. Whereas the distributed read places in noncontiguous areas of the memory items which are part of a single tape record, the distributed write gathers up items scattered in the memory and writes them consecutively on tape as part of the same record. When the instruction is interpreted, the main memory address generated from the A address group is placed, with a positive sign, in the write address counter (WAC) and the main memory address generated from the B address group is inserted, with a positive sign. in the distributed write address counter (DWAC). As each word of an item is written, the WAC is incremented by one to represent the location from which the next word will be written. When an end-of-item word is sensed, it is written on tape, the contents of the location specified by the DWAC are transferred to the WAC, and the contents of the DWAC are incremented by one. This process continues until an end-of-record word is sensed and the entire record has been written on tape. Since the compute orthocount instruction (see Section XII) can sense for end-of-item words and change control for distributed item handling, it is possible to provide the orthotronic control words required for file protection without disturbing the arrangement of distributed items in memory. If an error is sensed while performing a distributed write, the record is completely written and the next write instruction to this tape results in an unprogrammed transfer to U + 6 or U + 7.

Since writing on any peripheral device may proceed concurrently with computation, the programmer must take care that he does not begin to obliterate a stored record after a write instruction until the record has been completely written. The techniques discussed under the read forward instruction for insuring that a read operation has been completed are equally useful, with appropriate modification, in guaranteeing the completion of a write instruction.

Rewind, RW

This instruction can be executed by a magnetic tape unit or a paper tape reader. When addressed to a magnetic tape unit, it is executed at three times normal tape speed. Upon receiving the command, the tape starts to accelerate to full rewinding speed after a delay of approximately one second. Retraction of the head takes place during the acceleration period. Rewinding continues at high speed until the leader is reached; the tape is then stopped and the head restored to operating position. The device interlock is removed when the head has reached operating position. Including the acceleration-deceleration factor, the time to rewind a full reel of magnetic tape is one and one-half minutes (or approximately three minutes for the model 804-3 economy tape unit).

Like the other peripheral instructions, the rewind instruction specifies in the high-order six bits of its command code the device to be rewound. If the specified device is not capable of executing a rewind, the action of the Honeywell 1800 is unspecified. If address A is active in a rewind addressed to magnetic tape, a manually removable interlock is set after the tape is rewound. Until the interlock has been removed at the tape unit, the central processor cannot activate the device. This feature is particularly useful in file maintenance operations when a new file tape has been written and it is desired to insure that the tape is removed and replaced before another instruction with this tape address is processed. If the A address is inactive, no interlock is set. The contents of the B and C address groups are irrelevant to the execution of the instruction and may be used for storing information. Whether the C address is active or inactive, a hunt for the next sequencing counter in demand always occurs after implementation of the instruction.

If the magnetic tape is already rewound when the instruction is given, it is moved a fraction of an inch, but otherwise no action occurs except the normal incrementation of the sequencing counter that selected the instruction. If the instruction is directed to a tape unit upon which an error was detected during the previous read or write, the error condition is reset, the instruction is executed, and no unprogrammed transfer occurs.

When a rewind instruction is addressed to a paper tape reader, tape is wound onto either the takeup reel or the supply reel, depending upon a switch setting at the reader. Rewinding speed is the same as the tape reading speed selected (50 or 100 inches per second). The paper tape reader cannot be rewound with interlock; therefore the contents of the A address group are irrelevant in such a rewind instruction. If an error was detected on the previous read from the paper tape reader, the error condition is reset and rewinding is inhibited. Instead the instruction is stored in U or U + 1, an octal one (001) is placed in the high-order three bits of the program control register, and an unprogrammed transfer is made to U + 6 or U + 7.

The time required to execute a rewind instruction is three memory cycles. An extra cycle is required if a busy condition is encountered.

SECTION XII

MISCELLANEOUS INSTRUCTIONS

Print, PRA, PRD, PRO

A Honeywell 1800 system may include either one or two automatic typewriters. The standard unit, located at the central console, is known as the console typewriter. An optional unit, called the slave typewriter, may also be located near the console. Provision of a slave typewriter allows program printouts to be physically separated from the console input information or permits two programs operating in parallel to produce printout information on separate typewriters.

Programmed communication between the central processor and the various typewriters is made possible by the print instruction, which is defined as follows: print the contents of the location specified by the A address group on the typewriter and in the format specified by the B address, and change the designated sequencing counter so that the next instruction is selected from the location specified by the C address. The command code for the print instruction permits special register addressing and change of sequencing counter. However, if the A address group contains a direct special register address or an indirect memory location address, the value of the increment must be zero or the behavior of the system is unspecified. Masking is not possible with this instruction.

The B address group of the print instruction contains special information designating the mode and format of the printout and the typewriter to be used. Memory designator bit B is ignored. The significance of each of the 12 bits in the address group is described below:

Bit 1:	This bit must always be zero.
Bit 2:	Unspecified.
Bit 3:	This bit, called the carriage-return bit, is zero to specify carriage return after the word has been printed, one to specify no carriage return. However, if bit 4 is zero, the carriage will be returned regardless of the value of bit 3.
Bit 4:	This bit, called the more-to-follow (MTF) bit, is zero to specify that no more information is to follow from this program, one to specify that more information is to follow. The programmer must set this bit to one whenever he wishes to insure that a message of more than one word will not be interrupted by printouts from other programs. Whenever this bit is one, the designated typewriter is interlocked against all other programs until another word is printed from the same program.
Bit 5-6:	These two bits specify the mode in which the information is to be printed, i.e.,

01 octal
11 hexadecimal
10 alphanumeric
00 this combination is not defined.

Bits 7-12: These bits designate the typewriter on which the information is to be printed. Address 00 designates the console typewriter, address 01 represents the slave typewriter (in octal).

Associated with the two automatic typewriters is a typewriter buffer location (location 00021) in main memory bank 0 (see Section IV, page 44).

If the A address of a print instruction is inactive, the contents of the accumulator are printed as specified by the B address. In this one case, a control error does not occur if the contents of the accumulator have already been delivered to a result location. If the B address is inactive, no typewriter is activated. If the C address is inactive, no sequence change is made. If the C address is active, the specified sequence counter is set in the same manner as described for a transfer sequence change instruction having an active C address (see Section VIII, pages 80 and 81). Hunting for the next program in demand is always inhibited.

When printing under program control, the typewriter prints each word specified in the indicated mode. In the alphanumeric mode, if the code for carriage return or tabulate is encountered, the corresponding symbol is printed but the function is not performed. When printing hexadecimal words, a space is inserted after every three characters, while a space appears after every four characters in an octal printout. Spaces appear in the alphanumeric print mode only where they occur as part of the data. Under program control, the carriage return function is activated only by reaching the physical end of the carriage or by specifying "carriage return" or "no more to follow" in the B address group of the print instruction.

Whenever the preceding print instruction did not establish the "more to follow" condition, the group indicator associated with the program causing the current printout appears to the left of the printed data, separated by spaces from the information. The values printed for the group indicator are 0 through 7.

If a parity error is detected in a word being printed on the typewriter, the word is followed by a space and two octal check characters. Each bit of the check characters represents one of the parity bits of the erroneous word: one if the corresponding parity bit is in error and zero otherwise.

Although there is but one machine instruction for the print function, ARGUS recognizes three separate print instructions with the following mnemonic operation codes:

PRA	alphanumeric print
PRD	hexadecimal print
PRO	octal print

The operation code may be followed in the command code field by a comma and the letter M (denoting more information to follow before carriage return) or the letters MR (denoting more information to follow after carriage return). If neither appears, the carriage is returned after printing, and the typewriter is released to print from another program.

The A and C address fields may contain any valid address format, subject to the restriction already noted with regard to the use of a special register in the A address. The B address field, which is used only to specify the active typewriter, contains a C, an S, or a 2-digit number specifying the typewriter which is to print. Either C or 00 indicates the console typewriter; S or 01 indicates the slave.

The time required to select and execute a print instruction with a direct memory location A address and an inactive C address is four memory cycles. One additional cycle is required if C is a direct memory location address. When the contents of A have been transferred to the appropriate buffer register and the sequence change, if any, has been effected, the central processor proceeds with the next instruction from the same program. When a character is printed (approximately once every 100 milliseconds), seven memory cycles are required to transfer the buffer contents to the accumulator, shift the next character into printing position, and return the result to the buffer.

Control Program, MPC

To facilitate parallel processing, the Honeywell 1800 central processor contains a non-addressable 48-bit register which is used to store information required for efficient control of a multi-program operation. Access to this register, called the program control register (PCR), is provided by the machine instruction control program. The information stored in the program control register is described below:

Bits 1-3: Peripheral Fixed-Start Bits. These bits (normally 000) are set to generated non-zero values whenever an unprogrammed transfer is made to U + 6 or U + 7. Specifically, when a read or write instruction is addressed to an on-line peripheral device at which the PROGRAM switch has been manually set to one of the UTR positions 1-7, the octal equivalent of the switch setting is stored in these bits. The use of this switch, which is discussed in Section V, page 63, as well as in the Equipment Operators' Manuals for the various peripheral devices, is a technique to be used only at the direction of the programmer. In addition, when a read or write instruction (or a rewind instruction addressed to a paper tape reader) discovers a stored error condition in the device addressed, an octal one (001) is stored in the peripheral fixed-start bits, as described in Section XI. These bits retain their generated values until the next peripheral instruction is selected, at which time they are cleared to zeros.

Bits 4-8: Control Error Bits. When a control error occurs in the central processor, the type of error is indicated by the configuration in bit positions 4 through 8 of the program control register, as well as by a set of five console lights having a one-to-one correspondence with these five bits. The significance of the various bit configurations is shown in Appendix D.

Bits 9-16: Input Buffer Interlock Bits (one for each input channel). Each such bit is a one if the corresponding input buffer is interlocked and a zero if it is not.

Bit 17: Unassigned.

Bits 18-20: Control Group Indicator. These bits store an octal value 0 through 7 denoting the special register group under whose control the current instruction was selected.

Bits 21-28: Bisequence Bits (one per special register group). Each such bit is a one if the next instruction to be executed under control of the respective special register group is to be selected by the cosequence counter or a zero if the next such instruction is to be selected by the sequence counter.

Bits 29-36: Program Demand Bits (one per special register group). Each such bit is a one if the corresponding sequence counter or cosequence counter is in demand and a zero otherwise. By "in demand" is meant program running, including the case in which the instruction is stalled to await availability of a peripheral device.

Bits 37-40: Console Fixed-Start Bits. The values of these four bits are generated when a hexadecimal key is struck as part of a console fixed-start instruction. The operator uses this instruction to restart a program without knowing the location from which the start is to be made; he specifies the group indicator of the controlling special register group, followed by a hexadecimal digit which the programmer specifies. This instruction is described in detail in the <u>Equipment Operators' Manual - Console</u>. The console fixed-start bits retain the generated values until another console fixed-start is executed.

Bits 41-48: Output Buffer Interlock Bits (one for each output channel). Each such bit is a one if the corresponding output buffer is interlocked and a zero otherwise.

For any group of eight bits which represent the eight special register groups (input buffer interlock bits, bisequence bits, program demand bits, output buffer interlock bits), the highest-order bit corresponds to special register group 0, and the succeeding bits correspond to groups 1, 2, 3, etc. For example, bit 29 is the program demand bit for group 0, bit 30 represents group 1 and so forth up to bit 36, which represents group 7. It should be noted particularly that this assignment differs from the order of priority for traffic control. Figure XII-1 summarizes the relationship of each group to priority in traffic control and to the program control register.

The control program instruction is defined as follows. Ignore the A address group. Place the contents of the program control register in the location specified by the C address group. Then alter the bits of the program control register specified by bits 5 through 12 of the B address group of this instruction. Bits 1 through 4 of the B address define the manner in which the bits of the program control register are altered. Hunt for the next program in demand if the memory designator for the B address (bit 5 of the command code) is one; otherwise do not hunt. Figure XII-2 shows the manner in which B_{1-4} (the high-order four bits of B) determine the interpretation of B_{5-12} (the low-order eight bits of B).

The command code for the control program instruction allows the programmer to specify whether the next instruction shall be selected by the sequence or cosequence counter. However, if a conflict exists between the value assigned to a bisequence bit by the B address of a control program instruction and the value assigned to the same bit by command code bit 1 of the same instruction, the value corresponding to bit 1 takes precedence. The affected bisequence bit in the PCR is set according to the value of bit 1, but not until the PCR contents have been stored.

Since the A address of the instruction is ignored, it may be used to store 12 bits of data at the discretion of the programmer. The A address designator bit may be zero or one. The C address may be a direct, indexed, or indirect memory location address.

From the standpoint of hunting for the next program in demand, it is irrelevant for this one instruction whether the C address is active or inactive. Hunting is controlled by the B address designator bit in the command code; a one in this bit allows hunting. In addition, hunting always occurs if the program executing the control program instruction is stalled (see B_{1-4} = 1000, Figure XII-2). If the C address is inactive, the contents of the program control register are stored in both the accumulator and the low-order product register. Following a control program instruction with inactive C address which does not allow hunting, the former contents of the program control register may be retrieved from the accumulator by immediately executing a word add instruction with inactive A and B addresses, or from the low-order product register by immediately executing a transfer and sequence change (TS) instruction with an inactive A address.

Multiprogram control operates in such a way that the next active control group is selected during the execution of an instruction that allows hunting. For this reason, execution in the hunt mode of a control program instruction which turns on the demand for a new group without a group already in demand intervening will result in one complete search cycle before the execution of an instruction in the newly activated group. On the other hand, execution in the hunt mode of a control program instruction which turns off the demand of the next selected group will cause

Special Register Group Indicator (PCR Bits 18-20)	Program Demand Bits PCR Position	Bisequence Bits PCR Position	Output Buffer Interlock Bits PCR Position	Input Buffer Interlock Bits PCR Position	Peripheral Channel Address (Command Code Bits 1-3)	Traffic Control Priority	
						Output	Input
0	29	21	41	9	0	8	16
1	30	22	42	10	1	1	9
2	31	23	43	11	2	2	10
3	32	24	44	12	3	3	11
4	33	25	45	13	4	4	12
5	34	26	46	14	5	5	13
6	35	27	47	15	6	6	14
7	36	28	48	16	7	7	15

Figure XII-1. Honeywell 1800 Special Register Group Indicator Relationships

	B_{1-4}	Mnemonic Code Where Applicable	Interpretation of B_{5-12}*
These four combinations do not affect the bisequence bits.	0000	STOP	Turn off the program which initiated this instruction
	0001	DOFF	Turn off all program demand bits corresponding to zeros in B_{5-12}; do not alter program demand bits corresponding to ones.
	0010	DON	Turn on all program demand bits corresponding to ones in B_{5-12}; do not alter program demand bits corresponding to zeros.
	0011	MPC	Turn on all program demand bits corresponding to ones in B_{5-12}; turn off all program demand bits corresponding to zeros.
These five combinations may affect the bisequence bits.	0100	SPCR	Do not alter the contents of the program control register.
	0101	SCON	Turn on the program demand bits and set to SC the bisequence bits corresponding to zeros in B_{5-12}; do not alter the program demand bits or the bisequence bits corresponding to ones in B_{5-12}.
	0110	CSCON	Turn on the program demand bits and set to CSC the bisequence bits corresponding to ones in B_{5-12}; do not alter the program demand bits or the bisequence bits corresponding to zeros in B_{5-12}.
	0111	MPC	Do not alter any program demand bits; set to SC the bisequence bits corresponding to zeros in B_{5-12}; set to CSC the bisequence bits corresponding to ones in B_{5-12}.
	1000	MPC	Turn on the program demand bit and set to SC the bisequence bit corresponding to a one in B_5 or B_7. If B_5 and B_7 are both ones, the behavior of the system is unspecified. Select the next instruction from the group thus turned on. Turn off the group which selected this instruction (but do not alter the bisequence bit) if the corresponding bit (B_6 or B_{8-12}) is a zero. If the corresponding bit is a one, do not alter the program demand bit or the bisequence bit for this group. Do not alter the program demand bit or bisequence bit of any other program. If the program demand bit corresponding to a one in B_5 or B_7 is already on, do not execute this instruction but stall the group which selected it until the specified program demand bit is turned off.

*Bits 5 through 12 of the B address are in one-to-one correspondenece with the bisequence bits (21 through 28) and the program demand bits (29 through 36) of the program control register. In other words, B_5 controls special register group 0, B_6 controls special register group 1, etc.

Figure XII-2. B Address Group Function in Control Program Instruction

multiprogram control to resume searching for the next active group. Finally, if a control program instruction which does not permit hunting turns off the demand of the program which selected it, this program remains in control until it encounters an instruction which allows hunting.

Six of the nine operations that can be performed by the control program instruction are represented in ARGUS notation by a group of program control instructions, each designated by a unique mnemonic operation code. These six instructions, which perform most of the control operations required in normal usage, are explained in detail in the ARGUS Manual of Assembly Language. To perform the other three functions, ARGUS also recognizes a control program instruction written in machine language with the mnemonic operation code MPC. In this case, the 12 bits of the B address group are written as three hexadecimal digits.

Without exception, the time required to execute the control program instruction is always four memory cycles.

Proceed, PR

This instruction, whose 8-bit operation code consists entirely of zeros, results in no operation other than the normal incrementing of the sequencing counter that selected it. Bits 1, 5, and 6 of the command code are irrelevant, while bit 4 must be a zero; therefore, proceed cannot specify the source of the next instruction and is always followed by an instruction selected by the same sequencing counter.

Because the A, B, and C address groups are irrelevant, they can be used to store information. However, if the information stored in the C address group consists of 12 binary ones (inactive address), hunting for the next program in demand is inhibited.

The time required to execute the proceed instruction is two memory cycles.

Simulator, S

Simulator instructions permit the programmer to represent a subroutine with a single instruction in his program. For each simulator instruction used, he codes a subroutine which is stored elsewhere in memory, beginning with the next memory location higher than the address specified by the command code. The simulator instruction sets the cosequence counter to the starting address of the subroutine and then gives control to this counter.

A simulator instruction is specified by a machine command code in which the low-order three bits are ones. If the high-order bit of the command code is a zero, the low-order 11 bits are interpreted as a main memory subaddress ending in octal seven. If the high-order bit is a one, the low-order 11 bits are interpreted as a 3-bit index register number and an 8-bit augmenter ending in octal seven. Since the high-order bit of the command code is used to indicate the type of addressing, the programmer does not have the option of specifying the source of the next instruction. In fact, the next instruction is automatically taken from the cosequence counter. This is the only instance in which the machine makes a functional distinction between the sequence counter and the cosequence counter.

When a simulator instruction is executed, the instruction itself is stored in the location specified by the command code. If this address refers directly to a memory location, then the instruction is stored in the same bank of memory from which it was executed; if the address is indexed, the instruction may be stored in any bank, according to the value of the bank indicator bits in the referenced index register. The cosequence counter is set to the next higher address and the next instruction is taken from the cosequence counter. The contents of the source counter, after normal incrementing, are stored in the cosequence history register to provide a return to the main program.

Since the address portions of a simulator instruction have no assigned functions, they may be used to store subroutine parameters such as the sources of operands, the location in which a result is to be stored, the number of words to be manipulated, and so forth. Since the machine automatically stores in special register AU1 an address generated from the A address group and in AU2 an address generated from the C address group, it is advantageous to use these address groups for operand or result locations. The A and C address groups may contain direct memory location addresses (bit 1 = zero) or indexed memory location addresses (bit 1 = one). If bit 1 is zero, the low-order 11 bits of the address group and the 4-bit bank indicator from the sequencing counter which selected the instruction are placed in the arithmetic control counter with a positive sign to form a complete address. If bit 1 is one, the augmented contents of the referenced index register are placed in the counter with a positive sign. The contents of the locations whose addresses are stored in the two counters may then be referenced easily through the technique of indirect memory location addressing.

The command code for an ARGUS simulator instruction is S, followed by a comma and an address designated by a symbolic tag or by an index register number with augmenter equal to seven. The A and C address fields may contain symbolic tags, with or without address modifiers, or index register numbers with augmenters.

The time required to execute the simulator instruction is seven to nine memory cycles, as indicated in Appendix C.

Compute Orthocount, CC

The technique of orthotronic control incorporated in the Honeywell 1800 assumes that every record written on tape will include two orthowords, one associated with the odd-numbered data words in the record, the other associated with the even-numbered words. If the record contains an even number of data words, then orthoword 1 represents the orthocount of the odd words (words 1, 3, 5 etc.,) and orthoword 2 represents the orthocount of the even words. If an odd number of words is orthocounted, then orthoword 1 contains the orthocount of the even words (words 2, 4, 6 etc.,) and orthoword 2 contains the count of the odd words. Each orthoword is the complement of the binary half add of the words associated with it.

The machine instruction which performs the orthocount of a record and generates the two orthowords is called compute orthocount. When this instruction is executed, the machine first generates an end-of-record word which is placed in the location specified by the C address group. It then orthocounts the record, starting with the location specified by the A address group and ending when an end-of-record word is reached. The first end-of-record word sensed will terminate the operation, regardless of whether the location specified in C has been reached. Orthoword 1 is stored in the location specified by C; orthoword 2 is stored in C + 1; and an end-of-record word is placed in C + 2. If address B is inactive, control is not changed for distributed item handling. If the B address is active, distributed item control is exercised, using a memory location table stored in consecutive words beginning with the location specified by B.

The behavior of the system is unspecified if either the A or C address group of the compute orthocount instruction contains a direct or indexed special register address. If C is an indirect memory location address with a non-zero increment, the behavior of the system is also unspecified.

At the beginning of the instruction, special register AU1 is set to the location designated by the A address group. As each word is orthocounted, the counter is automatically incremented by one to specify the location of the next word to be orthocounted. At the conclusion of the instruction, AU1 contains the address of the end-of-record word which terminated the instruction, plus one. If the B address of the instruction is active, special register AU2 is initially set to the location specified by the B address group. This counter performs the same function for the distributed orthocount as the distributed read address and distributed write address counters perform for the distributed read and write instructions (see Section XI). As each item is handled, the counter is automatically incremented by one to reference the entry in the address table where the starting address of the next item is found. At the conclusion of the instruction, special register AU2 always contains the address of the location specified in the C address of the instruction plus two, regardless of whether the instruction was normal or distributed. (NOTE: The sign bits of AU1 and AU2 conform to the conventions stated in Section V, page 60.)

The provision for generating an end-of-record word and storing it in the location designated by the C address guarantees the programmer that the record being orthocounted has a valid end-of-record word. This is a necessary precaution, since the compute orthocount instruction is terminated only when an end-of-record word is sensed. During execution of the instruction, every other word is half-added, in binary, to form orthoword 1, and the alternate words are similarly added to form orthoword 2. The accumulator is set to all binary ones before performing the first half-add for each orthoword. The inclusion of a word of binary ones in the half add assures that each orthobit is an "odd" parity check on the identical bit positions of all the associated words. The accumulator and mask register are used to store partial results as the successive alternate words are half-added. When the orthocount is terminated by an end-of-record word, which is not included in the orthocount, the contents of the accumulator are transferred as orthoword 1 to the location specified by the C address, and the contents of the mask register are transferred as orthoword 2 to C + 1. If distributed item control is exercised, the end-of-item words are included in the orthocount.

The compute orthocount instruction is used in conjunction with the check parity instruction (see below) to reconstruct data in which an error has been detected. The channel in which the parity failure occurred may be identified by computing new orthowords for the record in error and comparing them with the orthowords accompanying the record. The check parity instruction may then serve to identify the erroneous word or frame. The formation and use of orthowords is described in greater detail in Appendix B.

The time required to execute a compute orthocount instruction with inactive B address and direct memory location A and C addresses is 11 + n memory cycles, where n equals the number of words orthocounted. Variations in timing for distributed item handling are shown in Appendix C.

Check Parity, CP

When an error is detected during a read from tape, the check parity instruction is used to pinpoint the incorrect word(s) or frame(s). The instruction may be performed without a mask to check the parity of the entire word, or it may be used with a frame mask to test parity on one or more frames. The checking is accomplished by transferring the word to be checked (stored in the location specified by the A address group) to the location specified by the B address group. During this transfer, parity is automatically checked by the parity-check circuits, and an indicator is set if parity is incorrect. The correct parity bits from the parity-check circuits are delivered to B, together with the information bits contained in A, thus guaranteeing that the word stored in B has proper parity so that it can be manipulated in the machine without causing a control error. The parity-error indicator is then sampled, and if the parity of the word at address A was incorrect, the sequencing counter specified as the source of the next instruction is changed to select the next instruction from the memory address designated by C. The C address group may contain a direct memory location address, an indexed memory location address, an indirect memory location address, or an indexed indirect memory location address, as described under the transfer and sequence change (TS) instruction (see Section VIII, pages 80 and 81).

When the check parity instruction is used with a mask, only those information bits corresponding to binary ones in the mask are checked for parity. If the mask defines one or more complete frames, the sequencing counter will be changed if any frame in the masked operand has incorrect parity. The instruction is undefined for any other type of mask.

In order to test parity on an individual frame basis, the programmer must be aware of the relationship between the information bits of a word on tape and their positions in memory. This relationship is shown in Figure XII-3.

From this figure it will be seen that the mask used to check the parity of frame 1 must have the following bit configuration:

1100 1100 1100 1100 0000 0000 0000 0000 0000 0000 0000 0000,

while the mask required to check the parity of frame 2 would have the configuration:

0011 0011 0011 0011 0000 0000 0000 0000 0000 0000 0000 0000.

The time required to execute an unmasked check parity instruction with direct memory location addresses is four memory cycles.

Word in Memory

$\underbrace{P1\ P2\ P3\ P4\ P5\ P6}_{\text{Parity Bits}}$ $\underbrace{1\ 2\ 3\ 4\ 5\ \ldots\ldots\ 46\ 47\ 48}_{\text{Information Bits}}$

Word on Tape

Frame										
	1	P1	1	2	5	6	9	10	13	14
	2	P2	3	4	7	8	11	12	15	16
	3	P3	17	18	21	22	25	26	29	30
	4	P4	19	20	23	24	27	28	31	32
	5	P5	33	34	37	38	41	42	45	46
	6	P6	35	36	39	40	43	44	47	48
Channel		9	8	7	6	5	4	3	2	1

$\uparrow$ tape movement

Figure XII-3. Bit Layout -- Memory and Tape

SECTION XIII

SCIENTIFIC INSTRUCTIONS

The 21 scientific instructions include 18 instructions that manipulate data in floating-point form, two that perform fixed-point decimal and binary division, and one that converts data between fixed-point decimal and floating-point binary form. In a system that includes the 1801-B floating-point option, these 21 instructions are executed as machine instructions. In a system not equipped with an 1801-B, they are interpreted as pseudo instructions that call in library routines to perform the desired operations.

Floating-Point Numbers

As explained in Section I, a number can be represented in floating-point form by a mantissa (signed proper fraction) and an exponent (integral power of the radix), allowing the computer to keep track of the radix points and greatly reducing the problem of overflow. A floating-point number can be converted to fixed-point form by obtaining the product of the mantissa and the radix raised to the power of the exponent.

In Section III, the structure of a Honeywell 1800 floating-point word is shown as a 1-bit sign, a 7-bit exponent, and a 40-bit mantissa. In a floating-point decimal word, the mantissa is interpreted as 10 decimal digits; in a floating-point binary word, it is interpreted as 40 binary (or 10 hexadecimal) digits. A binary one in the sign position indicates that the number is positive, a binary zero that it is negative. The seven bits of the exponent can represent 128 different exponent values. In order to store floating-point words having negative as well as positive exponents, the actual exponent is defined as a number 64 less than the effective (or working) exponent stored in these seven bits. Thus, the range of the actual exponent in a floating-point word is from -64 to +63 as shown in the following table. In a floating-point decimal word, the actual exponent is interpreted as a power of 10; in a floating-point binary word, it is interpreted as a power of 16.

Effective Exponent		Actual Exponent
(binary)	(decimal)	(decimal)
0000000	0	-64
0111111	63	-1
1000000	64	0
1111111	127	+63

In floating-point form, there is no single number that uniquely represents a given fixed-point number. For example, the fixed-point decimal number .000563 can be represented in floating-point decimal form as

$$.000563 \times 10^{0};$$
or $$.0563 \times 10^{-2};$$
or $$.563 \times 10^{-3};$$ etc.

The normalized (or normal) form for expressing floating-point numbers is with the implied indicator point immediately to the left of the first significant digit. Therefore, the third example above shows the number .000563 in normalized floating-point form. The Honeywell 1800 would store this normalized number as follows:

1 0111101 0101 0110 0011 0000 0000 0000 0000 0000 0000 0000

which represents a positive sign, an effective exponent of 61 (actual exponent -3), and a mantissa of 5630000000. (Note that in floating-point binary, the same configuration represents the value $+.01010110001100\text{-------}00 \times 16^{-3}$.)

Since a normalized decimal mantissa may range from .1000000000 up to .9999999999, the range of a normalized floating-point decimal word is from $.1 \times 10^{-64}$ (or 10^{-65}) up to $.9999999999 \times 10^{63}$ (or virtually 10^{63}). In the same manner, the range of a normalized floating-point binary word is from $.0001 \times 16^{-64}$ (or 16^{-65}) up to $.1111\text{----}11 \times 16^{63}$ (or virtually 16^{63}).

The number zero can be represented by any floating-point number in which the mantissa is zero. A normalized zero, however, is defined in the Honeywell 1800 by the configuration 1000 0000 .... 0000, which represents a positive sign, a working exponent of zero (or actual exponent of -64), and a mantissa of zero. Any instruction which produces normalized results will deliver a high-order result of zero in this form.

The operands of floating-point instructions do not have to be normalized (except for floating-point divisors and comparison operands). The results are normalized, with the exception of:

1. Quotients of division operations involving non-normalized dividends;
2. Sums and differences of unnormalized addition and subtraction; and
3. The low-order portions of double-precision results.

If non-normalized operands are used in an operation which produces a normalized result, a significant digit may be lost in the result for every high-order zero in the operands.

A number that can be stored in one machine word is called a single-precision number; one that requires two machine words is known as a double-precision number. A double-precision floating-point number is defined as two single-precision numbers in which the signs are identical and the exponent of the low-order portion is ten less than the exponent of the high-order portion. A double-precision number is considered normalized if the high-order portion is in normalized form.

Certain floating-point instructions in the Honeywell 1800 always produce double-precision results. If the high-order portion of such a result is zero, it is produced in normalized form and the low-order portion is also a normalized zero. If, however, the high-order portion is not zero and the low-order portion is zero, the latter will not be normalized; i.e., the low-order portion will have an exponent ten less than that of the high-order portion. This non-normalized zero will, however, be treated as normalized if it is used in any instruction which produces a normalized result. Care must be exercised if this non-normalized zero is used in a comparison instruction in which the other operand is a normalized zero.

Floating-point data words are identified in ARGUS language by the constant codes FLDEC, floating-point decimal number; FLBIN, floating-point binary number; and EBC, extended binary number. A floating-point number may be specified with an indicator point, an explicit exponent, or both. An explicit exponent is expressed as an "E" and a signed or unsigned exponent immediately following the number. A floating-point decimal number may consist of up to ten signed or unsigned decimal digits and may have any value within the acceptable range. A floating-point binary number may consist of up to 13 signed or unsigned decimal digits and may have any value within the acceptable range. ARGUS converts these numbers to normalized floating-point decimal and binary words, respectively. An extended binary number may consist of up to 25 signed or unsigned decimal digits and may have any value within the acceptable range for floating-point binary words. ARGUS converts this number to normalized double-precision floating-point binary form, retaining 80 bits of the mantissa. The high-order 40 bits are stored with proper exponent and sign as one machine word; the low-order 40 bits are stored, with the same sign and an exponent 10 less than that of the high-order word, as the following word.

Floating-Point Arithmetic Registers

The 1801-B contains two arithmetic registers whose contents are of interest to the programmer: the floating accumulator, known as FLAC and composed of 48 flip-flops; and the 48-bit floating low-order product register, known as FLOP. In general, FLAC contains the principal result at the end of a floating-point operation and FLOP contains either the low-order or secondary portion of a double-precision (2-word) result or else a second result.

Three floating-point checks are performed on the operations in FLAC, and in FLOP if pertinent. If an error is detected during the transfer of information between the 1801-B floating-point unit and the 1801 central processor or if the A operand of a conversion instruction does not possess the proper mod-3 configuration, a transfer check is signalled and bits 4-8 of the program control register are set to 10111. If an error is detected during mantissa operations in FLAC (and in FLOP, if pertinent), a mantissa check is signalled and bits 4-8 of the program control register are set to 10100. If an error is detected during exponent operations in FLAC (and in FLOP, if pertinent), an exponent check is signalled and bits 4-8 of the program control register are set to 10110.

Instruction Configurations

The operation codes for the 21 scientific instructions (see Appendix E) are uniquely designated by command code bits 2, 3, and 7 through 12. Bit 1 is the bisequence bit, and bits 4 through 6 are the A, B, and C memory designator bits, respectively. All six types of addressing described in Section IV can be used with the scientific instructions. In most cases, however, only the four types that reference main memory are meaningful, due to the nature of the floating-point word. The scientific instructions comprise the following groups: floating addition, floating subtraction, floating multiplication, floating division, fixed division, floating comparison, normalization, conversion, and multiple unload.

Inactive Addresses

Except where otherwise specifically stated below, any address or combination of addresses may be inactive in a scientific instruction. In general, if the C address is active, the contents of FLAC are delivered to the location specified by C and the system hunts for the next sequencing counter in demand. Exceptions are the multiple unload and the comparison instructions. If the C address is inactive, hunting is inhibited to permit retrieval of the contents of FLAC and, if desired, of FLOP. In either case, the result is retained in FLAC and in FLOP if pertinent, although its retrieval cannot be guaranteed when hunting occurs.

The general rule for an inactive A or B address or both is as follows: the operand represented by the inactive address is replaced by the contents of FLAC (including the sign and exponent). For example, if A is inactive, then FLAC is treated as the A operand. Note that an addition instruction with inactive A and B addresses may be used to multiply the contents of FLAC by two, while a multiply with inactive A and B addresses may be used to square the contents of FLAC. Here the exceptions are the multiple unload and fixed-to-floating normalize instructions.

The presence of an inactive A or C address has no effect on the timing of a scientific instruction. The presence of an inactive B address, on the other hand, reduces the execution time by one memory cycle.

Exponential Overflow and Underflow

Although the use of floating-point numbers greatly reduces the problem of overflow, it does not entirely resolve this problem. Right shifts occur automatically to counter any mantissa overflow in a result, but exponential overflow occurs if the exponent of a result exceeds +63, i.e., if the working exponent is greater than +127. Exponential underflow, on the other hand, occurs if the result exponent drops below -64, i.e., if the working exponent is less than zero. Either condition results in an unprogrammed transfer of control. If exponential overflow occurs, the instruction is stored in U or U + 1, depending on which sequencing counter selected it, and the next instruction is taken from U + 14 or U + 15 (see Figure V-5, page 62). The exponent of the result is reduced by 128, bringing it within the acceptable range. In other words, if one is added to a working exponent of 127 (actually +63), the result will have a working exponent of zero (actually -64). If exponential underflow occurs, the instruction is stored in U or U + 1, the next instruction is taken from U + 12 or U + 13, and the result exponent is increased by 128.

Exponential overflow or underflow in a result which is deliverable to the location specified by C normally causes an immediate unprogrammed transfer, whether the C address is active or inactive. An exception occurs in the case of an instruction which uses the previous contents of FLAC as an operand and which does not deliver a result to main memory (inactive C address). If exponential overflow or underflow occurs in such a case, an overflow or underflow indicator is set. The unprogrammed transfer is delayed until the first occurrence of one of the following:

1. A scientific instruction does not use the previous contents of FLAC as an operand;
2. An instruction delivers a result to main memory; or
3. The actual exponent of the result becomes greater than +127 or less than -128.

Upon the occurrence of an overflow unprogrammed transfer, the programmer can determine whether the unprogrammed transfer was caused by a "normal" overflow (i.e., exponent greater than +63) or by a "double" overflow (i.e., exponent greater than +127) by examining the high-order bit of the result exponent. Figure XIII-1 shows the ranges of exponents for both valid floating-point numbers and stored overflow and underflow conditions. Note that following an unprogrammed transfer for normal overflow the high-order bit of the exponent is a zero, whereas following an unprogrammed transfer for double overflow this bit is a one. The high-order bit of the exponent can be used in the same manner following an underflow unprogrammed transfer.

The floating-point division instructions do not conform to the above discussion of stored overflow and underflow. In the event of overflow or underflow in the quotient of such an instruction, the unprogrammed transfer immediately follows the completion of the instruction. Neither condition is ever stored.

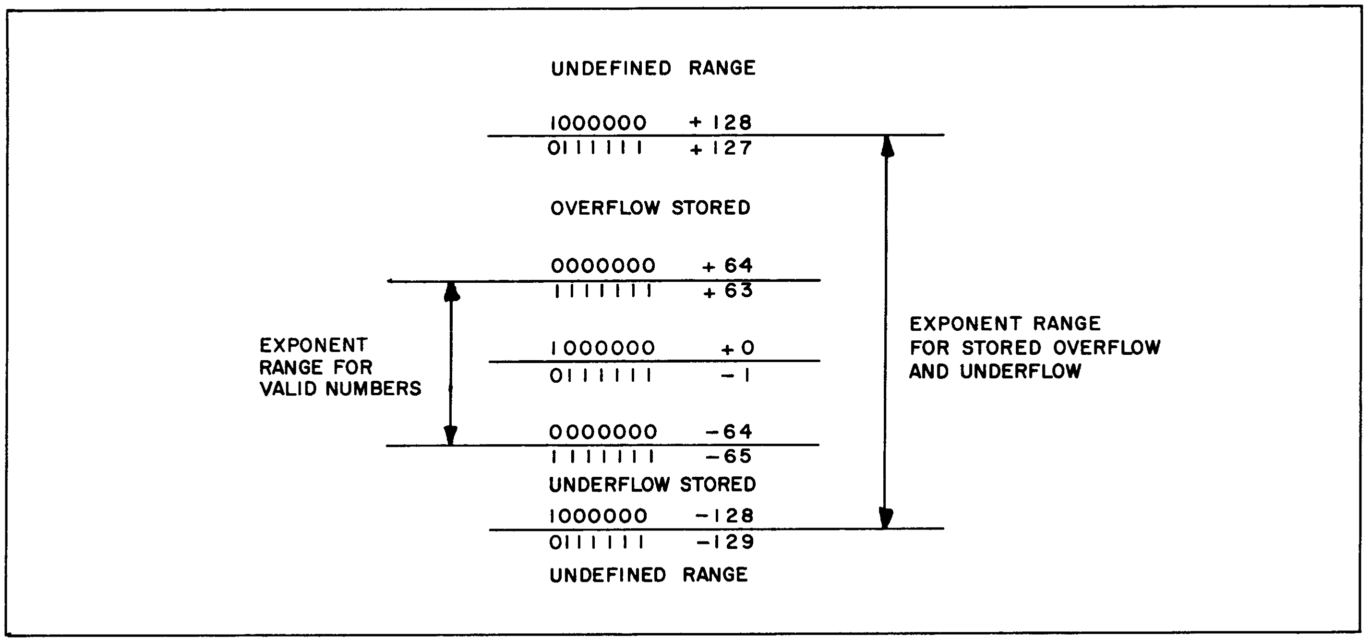


Figure XIII-1. Exponent Ranges in the Floating-Point Option

Exponential overflow[1] or underflow in the contents of FLOP does not cause an unprogrammed transfer but merely sets an indicator which is reset by the start of any instruction except multiple unload. Note that whenever a multiple unload follows an instruction which used the previous contents of FLAC as an operand and which did not deliver a result to main memory, an exponential underflow unprogrammed transfer may result from exponential underflow in either FLAC or FLOP.

Checking

Each operand delivered to the floating-point unit and each result delivered to the central processor is accompanied by modulo-3 check bits which are used to verify the accuracy of the information delivered. Modulo-3 checks are also performed to verify the contents of FLAC and, where pertinent, of FLOP. The floating-point unit uses a modulo-3 check on the A operand of the conversion instruction (FCON) to verify the operation code of the instruction, as described on page 134.

Failure of any modulo-3 check in the floating-point unit results in a control error (see Appendix D). In the event that the current instruction does not deliver a result to main memory, the control error indication may be delayed. In every case, however, this indication will occur

[1] Since the result in FLOP usually has a smaller exponent than the result in FLAC, exponential overflow in the contents of FLOP is normally impossible. An exception is the case of a floating-point division instruction which uses the previous contents of FLAC as an operand, in which case it is possible to have exponential overflow in the remainder.

no later than the conclusion of either the first succeeding instruction which delivers a floating-point result to main memory or the first succeeding instruction which does not use the contents of FLAC as an operand.

Floating Binary Add, FBA

This instruction performs an algebraic binary addition of the contents of the location specified by the A address group and the contents of the location specified by the B address group, producing as a result a normalized, single-precision, floating-point binary number in FLAC. If the C address group is active, the result is also delivered to the specified location and the system hunts for the next sequencing counter in demand. If C is inactive, the result is not delivered to main memory and hunting is inhibited. Provision is made for automatic handling of exponential overflow and underflow.

Floating Binary Subtract, FBS

Prior to the delivery of the operands to the addition circuitry, this instruction changes the sign of the B operand; a floating binary add is then executed.

Floating Decimal Add, FDA

This instruction operates in the same manner as floating binary add, except that the addition is decimal rather than binary.

Floating Decimal Subtract, FDS

This instruction operates in the same manner as floating binary subtract, except that after the sign change, the addition is decimal rather than binary.

Floating Binary Add, Extended Precision, FBAE

This instruction performs an algebraic binary addition of the contents of the location specified by the A address group and the contents of the location specified by the B address group, producing as a result a normalized, double-precision, floating-point binary number in FLAC and FLOP. If the C address group is active, the high-order result is delivered to the location specified and the system hunts for the next sequencing counter in demand. If C is inactive, no result is delivered to main memory and hunting in inhibited. Provision is made for automatic handling of exponential overflow and underflow on the result in FLAC. If exponential underflow occurs on the result in FLOP, the low-order underflow indicator is set; the nature of the double-precision result precludes the occurrence of low-order overflow.

Floating Binary Subtract, Extended Precision, FBSE

This instruction first changes the sign of the B operand and then executes a floating binary add, extended precision.

Normalized Floating-Point Addition and Subtraction

In the execution of the preceding six floating-point addition and subtraction instructions, the following stages occur:

1. Prenormalization. During this stage, all zero operands are normalized and the non-zero operand with the larger exponent is normalized, if necessary. If the exponents of the operands become equal during this process, the prenormalization halts.
2. Right Justification. The mantissa of the operand with the smaller exponent is shifted right until its exponent is equal to that of the other operand; this exponent is retained as the tentative exponent of the result. In single-precision instructions the bits that are shifted off are discarded immediately, but in extended-precision instructions they are discarded only after they have been shifted more than 40 bit positions.
3. Mantissa Arithmetic. Depending upon the operation code and upon whether the operand sign bits are like or unlike, the arithmetic circuits are set to perform an effective addition or subtraction of the mantissas. In the extended-precision instructions, the operation involves an 80-bit mantissa; a 40-bit mantissa is involved in the other floating add and subtract instructions. If mantissa overflow occurs, the mantissa of the result is shifted right four positions and the tentative result exponent is increased by one.
4. Result Normalization. The result is renormalized (if necessary) upon completion of the mantissa arithmetic. Exponential overflow and underflow are checked after the result exponent has been decreased by one for each 4-bit left shift. If a zero result is obtained from the execution of an extended-precision instruction, normalized zeros are provided for both the high-order and low-order results.
5. Result Transmission. The high-order result is retained in FLAC and the low-order result, if any, in FLOP. If C is active, the high-order result is also transmitted to the location specified by C. Arithmetic and control checks are completed at this time whether C is active or inactive.

Unnormalized Floating-Point Addition and Subtraction

The operation of the following four floating-point addition and subtraction instructions is identical to that of the normalized instructions described above, except that stages 1 (prenormalization) and 4 (result normalization) are omitted and exponential overflow and underflow are checked at the completion of stage 3 (mantissa arithmetic).

Unnormalized zero results may be either positive or negative. If the operand of larger exponent is zero, but the operational result is not zero, the result exponent is that of the zero operand while the sign is normally that of the non-zero operand. An exception occurs, however, if the non-zero operand in a subtraction operation appears in the location specified by the B address. In this case, the sign of the result is the opposite of the sign of the non-zero operand.

Floating Binary Add, Unnormalized, FBAU

This instruction is the same as floating binary add, except that the sum is not normalized.

A 4-bit shift to the right occurs if necessary to compensate for mantissa overflow, but no compensating left shifts occur to renormalize a result with zero in the most significant mantissa digit.

Floating Binary Subtract, Unnormalized, FBSU

This instruction first changes the sign of the B operand and then executes a floating binary add, unnormalized.

Floating Decimal Add, Unnormalized, FDAU

This instruction operates in the same manner as floating binary add, unnormalized, except that the arithmetic is decimal rather than binary.

Floating Decimal Subtract, Unnormalized, FDSU

This instruction operates in the same way as floating binary subtract, unnormalized, except that the arithmetic is decimal rather than binary.

Timing Notes on Floating-Point Addition and Subtraction

The timing of the above 10 floating-point addition and subtraction instructions is summarized in Figure XIII-2. The times given are based on the use of both normalized and unnormalized operands. All addresses are assumed to be active and direct. In general, the minimum times are achieved by instructions in which no prenormalization, justification, or result normalization is required. As stated earlier, instruction times are not affected by the presence of an inactive A or C address but are reduced by one cycle when the B address is inactive. The effects of indexed and indirect addressing are presented in Appendix C.

Instruction	Time in Memory Cycles		
	Minimum	Average	Maximum
Floating Binary Add/ Subtract (FBA, FBS)	4	4.95	6
Floating Decimal Add/ Subtract (FDA, FDS)	5	6.5	7
Floating Binary Add/Subtract, Extended (FBAE, FBSE)	4	6	7
Floating Binary Add/Subtract, Unnormalized (FBAU, FBSU)	4	5	6
Floating Decimal Add/Subtract, Unnormalized (FDAU, FDSU)	5	6	7

Figure XIII-2. Timing of Floating Add and Subtract Instructions

Floating Binary Multiply, FBM

This instruction multiplies the contents of the location specified in A by the contents of the location specified in B. The result is a normalized, double-precision, floating-point binary number in FLAC and FLOP. If the C address is active, the high-order result is delivered to the location specified and the system hunts for the next sequencing counter in demand. If C is inactive, no result is delivered to main memory and hunting is inhibited. Provision is made for automatic handling of exponential overflow and underflow in the high-order result; if underflow occurs in the low-order result, the low-order underflow indicator is set.

A zero result in floating-point nultiplication is a normalized zero (i.e., positive sign, zero mantissa, and -64 actual exponent) in both the high-order and low-order portions. Exponential overflow or underflow cannot occur with a zero result.

Floating Decimal Multiply, FDM

This instruction operates in the same manner as floating binary multiply, except that the arithmetic is decimal rather than binary.

Timing Notes on Floating-Point Multiplication

The timing of the floating-point multiplication instructions is summarized in Figure XIII-3. Again these times are based on the use of both normalized and unnormalized operands with all addresses active and direct. The minimum time for a floating decimal multiply is achieved when none of the multiplier digits is 4, 5, or 6, the most significant multiplier digit is not 7, 8, or 9, and the result does not require normalization.

Instruction	Time in Memory Cycles			
	Minimum	Average	Maximum	One Zero Operand
Floating Binary Multiply, FBM	5	5	6	4
Floating Decimal Multiply, FDM	8	10	12	4

Figure XIII-3. Timing of Floating Multiply Instructions

Floating Binary Divide, FBD

This instruction performs a floating binary division of the contents of B (dividend) by the contents of A (divisor), retaining the quotient in FLAC and the remainder in FLOP. If the C address is active, the quotient is also delivered to the location specified and the system hunts for the next sequencing counter in demand. If C is inactive, no result is delivered to main memory and hunting is inhibited. Provision is made for exponential overflow and underflow in the quotient. If exponential overflow or underflow occurs in the remainder (see footnote on page 126), the appropriate indicator is set.

The quotient will be normalized only if the dividend is in normalized form. The remainder will not be normalized unless the dividend is zero, in which case both the quotient and the remainder will be normalized zeros and exponential overflow and underflow cannot occur. The exponent of the remainder is nine less than that of the dividend if the absolute value of the dividend mantissa equals or exceeds the absolute value of the divisor mantissa; if the absolute values of the mantissas are equal, the remainder is an unnormalized zero with an exponent nine less than that of the dividend. The exponent of the remainder is ten less than that of the dividend if the absolute value of the dividend mantissa is less than the absolute value of the divisor mantissa. The sign of the remainder is the same as that of the dividend.

If the divisor is unnormalized or zero, the instruction is not performed. Instead, it is stored in U or U + 1, depending upon which sequencing counter selected it, and a division overcapacity unprogrammed transfer occurs to U + 10 or U + 11. (Note: The behavior of the system is unspecified if the program attempts to retrieve a result following division overcapacity by means of a multiple unload instruction, see below.)

Floating Decimal Divide, FDD

This instruction operates in the same manner as floating binary divide, except that the arithmetic is decimal rather than binary.

Fixed Binary Divide, BD

The fixed-point binary and decimal divide instructions are included in the complement of scientific instructions. The binary instruction performs a fixed-point binary division of the contents of B (dividend) by the contents of A (divisor), retaining the quotient in FLAC and the remainder in FLOP. If the C address is active, the quotient is also delivered to the location specified and the system hunts for the next sequencing counter in demand. If C is inactive, no result is delivered to main memory and hunting is inhibited.

If the absolute value of the contents of B is equal to or greater than the absolute value of the contents of A, the instruction is not performed but is stored in U or U + 1. An immediate division overcapacity unprogrammed transfer is executed to U + 10 or U + 11. As noted above, the behavior of the system is unspecified if the program attempts to retrieve a result following division overcapacity by means of a multiple unload instruction (see below).

Fixed Decimal Divide, DD

This instruction operates in the same manner as fixed binary divide, except that the arithmetic is decimal rather than binary.

Timing Notes on Division

The timing of the fixed- and floating-point division instructions is summarized in Figure XIII-4. These times are based on the use of both normalized and unnormalized dividends with all addresses active and direct.

Instruction	Time in Memory Cycles		
	Minimum	Average	Maximum
Floating Binary Divide, FBD	11	14	17
Floating Decimal Divide, FDD	17	20	22
Fixed Binary Divide, BD	12	15	18
Fixed Decimal Divide, DD	20	22	23

Figure XIII-4. Timing of Fixed and Floating Divide Instructions

Normalized Less Than Comparison, FLN

This instruction performs a direct algebraic comparison of the contents of the locations specified by the A and B address groups, according to the following rules:

1. If the contents of A and B have different signs, the operand with the positive sign exceeds the operand with the negative sign.
2. If the contents of A and B are both positive, the operand with the larger exponent exceeds the other operand. If the exponents are identical, the operand with the larger mantissa exceeds the other operand.
3. If the contents of A and B are both negative, the operand with the smaller exponent exceeds the other operand. If the exponents are identical, the operand with the smaller mantissa exceeds the other operand.

Note that this comparison is based upon the assumption of normalized operands. If either operand (or both) is unnormalized, the operand of larger magnitude may be interpreted as the smaller.

If the contents of A are less than or equal to the contents of B, the sequencing counter specified as the source of the next instruction is changed to the memory location address specified by C and hunting for the next sequencing counter in demand is inhibited. If the contents of A are greater than the contents of B, no change is made to the contents of the specified sequencing counter and hunting occurs. If C is inactive, no change is made to the contents of the specified sequencing counter and hunting is inhibited. The C address group may contain any of the four types of addresses permitted in the C address group of the transfer and sequence change (TS) instruction. The behavior of the system for each of these four types of addresses is the same as that described in Section VIII, pages 80 and 81. The basic execution time for the normalized less than comparison instruction is four memory cycles, regardless of the outcome of the comparison.

This instruction is actually built into the 1801 central processor and uses the accumulator in its execution. Following the completion of a normalized less than comparison, the word in the accumulator is invalid; any attempt to deliver this word to memory will therefore cause a control error.

Normalized Inequality Comparison, FNN

This instruction, which is actually built into the 1801 central processor, is identical to the inequality comparison, alphabetic (NA) instruction (see Section IX). In other words, it compares the contents of the locations specified by the A and B address groups for inequality. If the two operands are not identical, the specified sequencing counter is changed to the memory location address specified by C. If the two operands are equal, the specified sequencing counter is not changed. All properties of the inequality comparison, alphabetic instruction are retained. Note that a positive zero is not equal to a negative zero. This comparison is also based on the assumption of normalized operands. If either operand (or both) is unnormalized, identical operands may be interpreted as unequal or vice versa. The basic execution time for a normalized inequality comparison is four memory cycles, regardless of the outcome of the comparison.

Fixed-to-Floating Normalize, FFN

This instruction produces a normalized floating-point result based on the contents of the location specified by B and a portion of the contents of the location specified by A. The low-order 44 bits of the B operand are normalized to form the result mantissa. The sign of the result is the same as the sign of the B operand; i.e., the result is positive if the B operand contains one or more ones in the sign bit positions. The value of bits 2 through 8 (the exponent bits) of the A operand is taken as a tentative exponent of the result. The final exponent is determined during the normalization of the mantissa according to the following rules:

1. If the low-order 44 bits of B are shifted four bits to the right to form the mantissa, the tentative exponent becomes the final exponent.
2. If the low-order 44 bits of B are not shifted at all, the tentative exponent is decreased by one to form the final exponent.
3. If the low-order 44 bits of B are shifted to the left, the tentative exponent is decreased by n to form the final exponent, where n is one greater than the number of 4-bit shifts performed.

The result is retained in FLAC and FLOP, the low-order 36 bits of FLOP being set to zero. If the B operand is plus or minus zero, both FLAC and FLOP will contain normalized zeros. If the C address is active, the high-order result is also delivered to the location specified and the system hunts for the next sequencing counter in demand. If C is inactive, no result is delivered to main memory and hunting is inhibited. Exponential underflow in the high-order result produces a normal unprogrammed transfer. If exponential underflow occurs in the low-order result, the low-order underflow indicator is set.

If the A address is inactive, the exponent of the previous contents of FLAC is used as the A operand. Note that if the previous contents of FLAC are in fixed-point form (e.g., the result of a fixed-point divide or a binary-to-decimal conversion), the operation will not produce a meaningful result. If the B address is inactive and the previous contents of FLAC are in fixed-point form, these contents are used as the B operand. If the previous contents of FLAC are in floating-point form, the sign bit in FLAC is retained as the sign of the result, the exponent in FLAC is ignored, and the mantissa in FLAC is prefixed by four binary zeros to form the 44-bit B operand mantissa. When this mantissa is normalized, the exponent from the A operand is reduced by one to form the result exponent. The behavior of the system is unspecified if both the A and B addresses are inactive. The minimum and average execution time for this instruction is four memory cycles; the maximum is five memory cycles.

Multiple Unload, ULD

This instruction does not reset the exponential overflow and underflow indicators. It transfers the contents of FLAC to the location specified by A and the contents of FLOP to the location specified by C. The B address must be inactive; otherwise the behavior of the system is unspecified. If the A address is inactive, the contents of FLOP are transferred to the location specified by C and the contents of FLAC are undefined. If the C address is inactive, the contents of FLAC are transferred to the location specified by A and the contents of FLOP are placed in FLAC. Hunting for the next sequencing counter in demand occurs only if the C address is active. NOTE: In order to print the contents of FLAC and FLOP at the console, it is necessary to perform a multiple unload instruction and then print the contents of the corresponding memory locations.

If the exponential underflow indicator is set when the multiple unload instruction is initiated, this instruction is followed by an unprogrammed transfer to U + 12 or U + 13. If the exponential overflow indicator is set, the unprogrammed transfer is to U + 14 or U + 15. The execution time for a multiple unload instruction is four memory cycles.

Conversion, FCON

This instruction converts the contents of the location specified by B according to the contents of the location specified by A. If the B address is inactive, the previous contents of FLAC are used as the B operand. If the A address is inactive, the behavior of the system is unspecified. Two different kinds of conversion can be specified by the value of the A operand: fixed decimal to floating binary conversion, and floating binary to fixed decimal conversion.

1. Fixed Decimal to Floating Binary Conversion. This operation is specified by a conversion instruction in which the A operand has the octal value

   XXXXXXXX00000001

   The high-order 24 bits may have any values provided that the modulo-3 sum of the entire operand is one. This sum is checked in the floating-point unit to verify the operation code.

   The B operand is interpreted as a signed decimal number with decimal point to the left of the high-order digit and is converted to a floating-point binary mantissa in FLAC (unnormalized). The exponent in FLAC is zero, while the sign is positive if the B operand contains one or more ones in the sign bit positions. The decimal remainder is retained in FLOP where it has the significance of a decimal fraction multiplied by 16^{-10}. If the C address is active, the result in FLAC is delivered to the location specified and the system hunts for the next sequencing counter in demand. If C is inactive, no result is delivered to main memory and hunting is inhibited. NOTE: If the following instruction reverses the value of the high-order exponent bit, an effective fixed decimal to fixed binary conversion is obtained.

   The execution time for fixed decimal to floating binary conversion is 20 memory cycles.

2. Floating Binary to Fixed Decimal Conversion. This operation is specified by a conversion instruction in which the A operand has the octal value

   XXXXXXXX00000004

   The high-order 24 bits may have any values provided that the modulo-3 sum of the entire operand is one. This sum is checked in the floating-point unit to verify the operation code.

   The B operand is interpreted as a floating-point binary number and its mantissa is converted to a 44-bit fixed-point decimal fraction in FLAC. The exponent of the B operand is ignored, while the sign of the B operand is placed in the high-order four bits of FLAC. If the B operand mantissa is zero, however, the sign in FLAC is a one followed by three zeros. The hexadecimal remainder is retained in FLOP where it has the significance of a hexadecimal fraction multiplied by 10^{-11}. If the C address is active, the result in FLAC is delivered to the location specified and the system hunts for the next sequencing counter in demand. If C is inactive, no result is delivered to main memory and hunting is inhibited. NOTE: This instruction can also be used to convert the low-order 40 bits of a fixed-point binary word to a 44-bit decimal equivalent. The execution time for floating binary to fixed decimal conversion is nine memory cycles.

SECTION XIV

SUMMARY OF INSTRUCTIONS

General Instructions - Unmasked or Masked

MNEMONIC OPERATION CODE	DESCRIPTION[5]	TIME IN MEMORY CYCLES[1]
BA	Binary Add algebraically (A) to (B). Store sum in C. If overflow occurs, take next instruction from U + 8 if the sequence counter selected this instruction, or from U + 9 if the cosequence counter selected this instruction. The sign of either operand is positive if any of its four sign bits is one. The sign of the sum is 0000 if negative, 1111 if positive.	4 (See note 4)
DA	Decimal Add algebraically (A) to (B). Store sum in C. Otherwise same as Binary Add.	4 (See note 4)
WA	Word Add. Binary add (A) to (B), considered as unsigned 48-bit numbers. Store 48-bit result in C. If overflow occurs, observe same conventions as in Binary Add.	4
BS	Binary Subtract algebraically (B) from (A). Store result in C. Observe same overflow and sign conventions as in Binary Add.	4 (See note 4)
DS	Decimal Subtract algebraically (B) from (A). Store result in C. Observe same overflow and sign conventions as in Binary Add.	4 (See note 4)
WD	Word Difference. Binary subtract (B) from (A). Otherwise identical to Word Add.	4
NA	Inequality Comparison, Alphabetic. Compare (A) and (B) including sign positions. If $(A) \neq (B)$, change sequencing counter to select next instruction from location specified by C. Plus zero is not equal to minus zero.	4
NN	Inequality Comparison, Numeric. Compare algebraically (A) and (B). If $(A) \neq (B)$, follow procedure for NA. Plus zero equals minus zero.	4
LA	Less Than Or Equal Comparison, Alphabetic. Compare (A) and (B) including sign positions. If $(A) \leq (B)$, change sequencing counter to select next instruction from location specified by C. Plus zero is greater than minus zero.	4
LN	Less Than Or Equal Comparison, Numeric. Compare algebraically (A) and (B). If $(A) \leq (B)$, follow procedure for LA. Plus zero equals minus zero.	4

General Instructions - Unmasked or Masked (cont)

MNEMONIC OPERATION CODE	DESCRIPTION[5]	TIME IN MEMORY CYCLES[1]
TX	Transfer (A) to C. Ignore B.	3
TS	Transfer (A) to B. If C is active, change sequencing counter to select next instruction from location specified by C.	4
HA	Half Add. Binary add without carry (A) to (B), considered as unsigned 48-bit numbers. Store 48-bit result in C. Result is zero wherever corresponding bits of (A) and (B) are identical, one wherever corresponding bits of (A) and (B) are different. This is "logical exclusive OR."	4
SM	Superimpose (A) and (B). Store result in C. Result is zero wherever bits of (A) and (B) are both zero, one everywhere else. This is "logical inclusive OR."	4
CP	Check Parity. Test (A) for correct parity. Place (A) with correct check bits in B. If (B) differs from (A), change sequencing counter to select next instruction from location specified by C.	4

General Instructions - Unmasked

BM	Binary Multiply (A) by (B). Store high-order product with proper sign in C and accumulator, low-order product with proper sign in low-order product register. Product signs are 0000 if negative or 1111 if positive.	33
DM	Decimal Multiply (A) by (B). Store high-order and low-order products as in Binary Multiply with same sign conventions.	27
BT	Binary Accumulate. Place low-order 44 bits of (A) in accumulator. Perform this transfer B' times, adding in binary (with positive sign implied) the successive 44-bit words transferred. B' (high-order 6 bits of B) = 0 to 63. Store 44-bit result, with sign of first word transferred, in C. Observe same overflow conventions as in Binary Add. Note that if A is an indirect address with non-zero increment, B different numbers are accumulated.	3 + n (See note 2)
DT	Decimal Accumulate. Same as Binary Accumulate except that transferred words are added as 11-digit decimal numbers.	3 + n (See note 2)
MT	Multiple Transfer. Transfer (A) to C. Perform this instruction B' times. B' (high-order 6 bits of B) = 0 to 63. Note that if A and C are indirect addresses with non-zero increments, B' different transfers are performed.	1 + 2n (See note 2)
TN	N-Word Transfer. Transfer B' words from consecutive locations starting at A to consecutive locations starting at C. B' = 0 to 63.	5 + 2n (See note 2)

General Instructions - Unmasked (cont)

MNEMONIC OPERATION CODE	DESCRIPTION[5]	TIME IN MEMORY CYCLES[1]
CC	Compute Orthocount. Write a generated end-of-record word in the location specified by C. Orthocount the record starting at A to the end-of-record word. Store orthoword 1 in C and orthoword 2 in C + 1. Place end-of-record word in C + 2. If B is inactive, control is not changed for distributed item handling. If B is active, end-of-item words are sensed and control is changed for distributed item handling.	11 + n (See note 2)
IT	Item Transfer. Substitute an end-of-item symbol for the high-order 32 bits of (B), clearing the low-order 16 bits of (B) to zeros. Transfer words from consecutive memory locations starting with A to consecutive memory locations starting with C until an end-of-item or end-of-record word is transferred.	7 + 2n (See note 2)
RT	Record Transfer. Store an end-of-record word in B. Transfer words from consecutive memory locations starting with A to consecutive memory locations starting with C until an end-of-record word is transferred.	7 + 2n (See note 2)
MPC	Control Program. Ignore A. Place (PCR) in the location specified by C. Then alter the bits of PCR specified by bits 5-12 of B, using bits 1-4 of B to define how the bits are altered. If B address memory designator bit is 1 or if the program executing this instruction stalls, hunt for the next program in demand. Otherwise, do not hunt.	4
PR	Proceed. If C is inactive, do not hunt for next program in demand.	2

Inherent Mask Instructions

SWS	Shift Word and Substitute. Shift right end around including sign (A) as directed by B'. Mask result and store in C (protected). B' (high-order 6 bits of B) specifies the number of 1-bit shifts.	5 + k (See note 3)
SPS	Shift Preserving Sign and Substitute. Shift right end around excluding sign (A) as directed by B'. Otherwise same as SWS.	5 + k (See note 3)
SWE	Shift Word and Extract. Same as SWS except that the unmasked portions of (C) are unprotected.	5 + k (See note 3)
SPE	Shift Preserving Sign and Extract. Same as SPS except that the unmasked portions of (C) are unprotected.	5 + k (See note 3)

Inherent Mask Instructions (cont)

MNEMONIC OPERATION CODE	DESCRIPTION[5]	TIME IN MEMORY CYCLES[1]
SSL	Shift and Select. Shift right end around including sign (A) as directed by B'. Binary add to C under mask control no more than 11 low-order bits of the shifted word, with positive sign implied. Change the sequencing counter to select the next instruction from the location specified by the modified C address. B' is interpreted as in the SWS instruction.	6 + k (See note 3)
SS	Substitute. Using (B) as a mask, transfer (A) to C and protect unmasked portions of (C).	5
EX	Extract or Logical AND. Using (B) as a mask, transfer (A) to C without protecting unmasked portions of (C). Result is one wherever bits of (A) and (B) are both one, zero everywhere else.	5

Peripheral and Print Instructions

RF	Read Forward from peripheral device XX into consecutive memory locations starting with A. XX represents command code bits 1-6. Set the RAC to +A. If B is inactive, do not change control for distributed item handling. If B is active, set the DRAC to +B and sense for end-of-item words. Change sequencing counter to select next instruction from location specified by C. If end of tape is sensed, take next instruction from U + 4 if the sequence counter selected this instruction or from U + 5 if the cosequence counter selected this instruction. If a parity error was detected during the last previous read from this device, reset the parity error flip-flop, do not perform the read. Instead, take next instruction from U + 6 or U + 7. This instruction is interlocked against device XX and the associated buffer.	7
RB	Read Backward from magnetic tape unit XX into consecutive memory locations starting with A. This instruction is otherwise identical with RF except that the RAC is set to -A and if B is active the DRAC is set to -B.	7
WF	Write Forward on peripheral device XX the contents of consecutive memory locations from A through the end-of-record word. Set the WAC to +A. If B is inactive, do not change control for distributed item handling. If B is active, set the DWAC to +B and sense for end-of-item words. Change sequencing counter to select next instruction from location specified by C. If an error was detected during the last previous write to peripheral device XX, reset the parity error flip-flop, do not perform the write. Instead, take next instruction from U + 6 or U + 7. This instruction is	7

Peripheral and Print Instructions (cont)

MNEMONIC OPERATION CODE	DESCRIPTION[5]	TIME IN MEMORY CYCLES[1]
WF (cont)	interlocked against device XX and the associated buffer. End-of-tape convention is identical with Read Forward.	
RW	Rewind tape on magnetic tape unit or paper tape reader XX. If A is active in magnetic tape rewind, set manually removable interlock after tape is rewound. B and C are ignored. If an error was detected during the last previous read or write for this magnetic tape unit, reset the parity error flip-flop and perform the rewind. In the case of a paper tape reader, do not perform the rewind but take next instruction from U + 6 or U + 7.	3
PRA, PRD, PRO	Print (A) on the typewriter and in the format specified by B. If C is active, change the sequencing counter to select the next instruction from the location specified by C. In an alphabetic print, eight 6-bit characters are printed. In a decimal print, 12 hexadecimal digits are printed. In an octal print, 16 octal numbers are printed.	5

Simulator Instructions

S	Simulator. Form a memory location address (direct or indexed) from the low-order 11 bits of the command code and store this instruction in the location thus specified. Change the cosequence counter to select the next instruction from the next higher address.	7

Scientific Instructions

FBA	Floating Binary Add. Binary add algebraically (A) to (B). Deliver result as a normalized floating-point number to C if C is active; retain result in FLAC if C is inactive. If exponential underflow occurs, take next instruction from U + 12 or U + 13. If exponential overflow occurs, take next instruction from U + 14 or U + 15.	4.95
FDA	Floating Decimal Add. Same as floating binary add, except that arithmetic is decimal rather than binary.	6.5
FBS	Floating Binary Subtract. Change the sign of the B operand and perform a floating binary add.	4.95
FDS	Floating Decimal Subtract. Same as floating binary subtract, except that arithmetic is decimal rather than binary.	6.5
FBAU	Floating Binary Add, Unnormalized. Same as floating binary add, except that result is not normalized. A 4-bit right shift is provided if necessary to compensate for mantissa overflow, but no compensating left shift occurs to renormalize a result with zero in the most significant mantissa digit.	5

Scientific Instructions (cont)

MNEMONIC OPERATION CODE	DESCRIPTION[5]	TIME IN MEMORY CYCLES[1]
FDAU	Floating Decimal Add, Unnormalized. Same as floating binary add, unnormalized, except that arithmetic is decimal rather than binary.	6
FBSU	Floating Binary Subtract, Unnormalized. Change the sign of the B operand and perform a floating binary add, unnormalized.	5
FDSU	Floating Decimal Subtract, Unnormalized. Same as floating binary subtract, unnormalized, except that arithmetic is decimal rather than binary.	6
FBAE	Floating Binary Add, Extended Precision. Form the normalized double-precision sum of (A) and (B). If C is inactive, retain the high-order and low-order parts in FLAC and FLOP. If C is active, store the high-order part in C; the contents of FLOP are unspecified. Sense for exponential overflow or underflow on the high-order result. If exponential underflow occurs on the low-order result, set the low-order underflow indicator.	6
FBSE	Floating Binary Subtract, Extended Precision. Change the sign of the B operand and perform a floating binary add, extended precision.	6
FBM	Floating Binary Multiply. Multiply (A) by (B) to form a normalized, double-precision, floating-point product. If C is inactive, retain the high-order and low-order products in FLAC and FLOP. If C is active, store the high-order product in C; the contents of FLOP are unspecified. Sense for exponential overflow or underflow on the high-order product. If exponential underflow occurs on the low-order product, set the low-order underflow indicator.	5
FDM	Floating Decimal Multiply. Same as floating binary multiply, except that arithmetic is decimal rather than binary.	10
FBD	Floating Binary Divide. Divide (B) by (A). If C is inactive, retain the quotient and remainder in FLAC and FLOP. If C is active, store the quotient in C in floating-point form; the contents of FLOP are unspecified. The quotient will be normalized if the dividend is normalized. The remainder is not normalized. Sense for exponential overflow or underflow in the quotient. Set the low-order underflow or overflow indicator if there is underflow or overflow in the remainder. If the divisor is unnormalized or zero, do not perform the instruction but take the next instruction from U + 10 or U + 11.	14

Scientific Instructions (cont)

MNEMONIC OPERATION CODE	DESCRIPTION[5]	TIME IN MEMORY CYCLES[1]
FDD	Floating Decimal Divide. Same as floating binary divide, except that arithmetic is decimal rather than binary.	20
BD	Fixed Binary Divide. Divide (B) by (A), handling both operands as fixed-point binary numbers. If C is inactive, retain the quotient and remainder in FLAC and FLOP. If C is active, store the quotient in C in fixed-point form; the contents of FLOP are unspecified. If the absolute value of (B) equals or exceeds the absolute value of (A), do not perform the instruction but take the next instruction from U + 10 or U + 11.	15
DD	Fixed Decimal Divide. Same as fixed binary divide, except that arithmetic is decimal rather than binary.	22
FLN	Normalized Less Than Comparison. Compare (A) with (B) algebraically. If (A) $\leq$ (B), change the specified sequencing counter to C.	4
FNN	Normalized Inequality Comparison. Compare (A) with (B) including sign positions. If (A) $\neq$ (B), change the specified counter to C.	4
FFN	Fixed-to-Floating Normalize. Form the normalized result mantissa from low-order 44 bits of (B). Result exponent is the exponent of (A) minus one for each 4-bit left shift plus one for each 4-bit right shift minus one. Result is positive if high-order 4 bits of (B) include any ones. If C is active, store result in C. If C is inactive, retain normalized double-precision result in FLAC and FLOP, where low-order 36 bits in FLOP are zeros. Sense for exponential underflow in high-order result. Set low-order underflow indicator if exponential underflow occurs in low-order result.	4
ULD	Multiple Unload. Do not reset the low-order underflow or overflow indicator. Store (FLAC) in A and (FLOP) in C. B must be inactive. If the low-order underflow indicator is set when the instruction is initiated, take the next instruction from U + 12 or U + 13. If the overflow indicator is set, take the next instruction from U + 14 or U + 15.	4
FCON	Conversion. Convert (B) as specified by (A). If C is active, store result in C. If C is inactive, retain result in FLAC and remainder in FLOP. The modulo-3 sum of (A) must be one.	
	1. Fixed Decimal to Floating Binary Conversion. If (A) is XXXXXXXX00000001 (octal), convert	20

Scientific Instructions (cont)

MNEMONIC OPERATION CODE	DESCRIPTION[5]	TIME IN MEMORY CYCLES[1]
FCON (cont)	the low-order 44 bits of (B) to the unnormalized result mantissa. The result exponent is zero. The result is positive if the high-order 4 bits of (B) include any ones.	
	2. Floating Binary to Fixed Decimal Conversion. If (A) is XXXXXXXX00000004 (octal), convert the mantissa of (B) to a 44-bit fixed decimal result. Ignore the exponent of (B). The result takes the sign of (B).	9

NOTES:

1. One memory cycle equals two microseconds. All addresses considered active, except B addresses of CC and ULD instructions. For variations in time, see Appendix C.
2. n = number of words accumulated, transferred, or ortho-counted.
3. Values of k are based on number of 16, 4, and 1-bit shifts required. See Appendix C for table of values.
4. Under certain conditions, as explained in Appendix A, an additional one or two memory cycles may be required.
5. Where "(A)" is read "the contents of the location designated by the A address group."

APPENDIX A

FIXED-POINT ADDITION IN THE HONEYWELL 1800

Two registers are involved in the fixed-point addition function of the Honeywell 1800; the accumulator and the bus. A symbolic representation of these registers is shown in Figure A-1. The accumulator is used in the performance of addition, subtraction, multiplication, shifting, comparison, transfer, and print instructions.

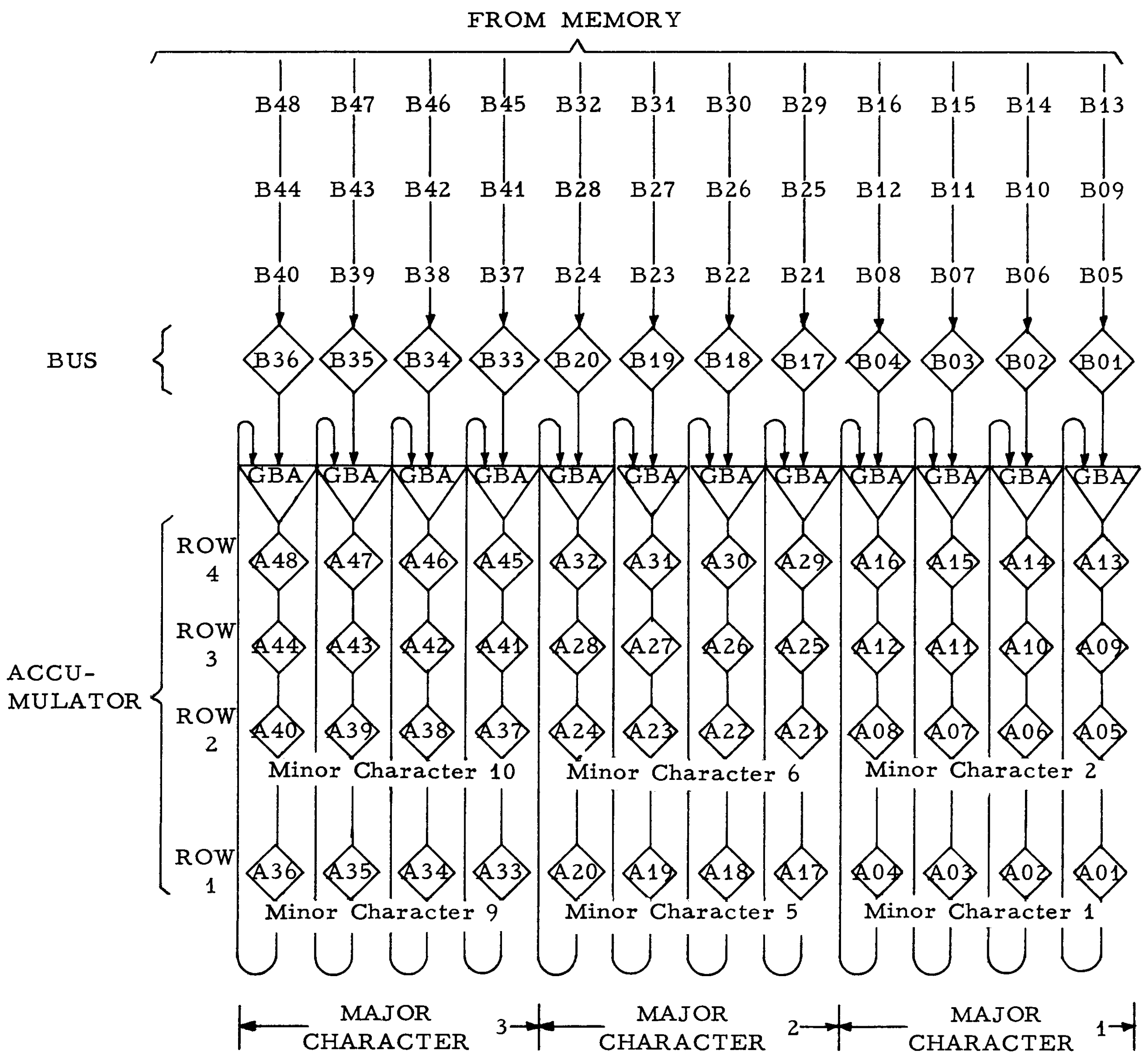
Since the Honeywell 1800 word contains 48 information bits, the accumulator requires 48 flip-flops, each capable of storing a single binary digit. These 48 flip-flops are arranged in three 4 x 4 arrays, called "major characters." Each major character contains four "minor characters." For example, minor character No. 1 contains bits 1, 2, 3, 4;[1] minor character No. 2 contains bits 5, 6, 7, 8; minor character No. 5 contains bits 17, 18, 19, 20; etc., (see Figure A-1).

The accumulator has 24 additional flip-flops and amplifiers called "carry functions" which permit carries or borrows generated during arithmetic operations to propagate through the arrays without interfering with the information bits. These carry functions are used to implement the anticipating techniques discussed below.

All the operations and manipulations accomplished by the accumulator are based on its ability to recirculate bits between rows. In other words, the information stored in the 48 flip-flops is not stationary. For example, the information stored in the 12 flip-flops of the first row remains static for less than one microsecond and is then transferred to the flip-flops in the fourth row. The information in the fourth row is fed in turn to the third row, the third to the second row, and the second to the first. By this recirculation process the accumulator stores information.

Each operand is delivered from the memory to the accumulator via the bus. In the accumulator, the least significant binary digit is stored in flip-flop 1, the next higher-order digit is stored in 2, and so on. The most significant binary digit of a signed number is stored in flip-flop 44; the most significant binary digit of an unsigned number (such as an operand of the word add instruction) is stored in 48. Since four binary digits are required to represent

[1] In this Appendix only, bits are numbered from right to left, conforming to standard engineering usage. This is in contrast to the programmers' left-to-right numbering convention which is followed elsewhere in the manual.


FROM MEMORY
B48 B47 B46 B45 B32 B31 B30 B29 B16 B15 B14 B13
B44 B43 B42 B41 B28 B27 B26 B25 B12 B11 B10 B09
B40 B39 B38 B37 B24 B23 B22 B21 B08 B07 B06 B05
BUS
B36 B35 B34 B33 B20 B19 B18 B17 B04 B03 B02 B01
GBA
ACCU-MULATOR
ROW 4
A48 A47 A46 A45 A32 A31 A30 A29 A16 A15 A14 A13
ROW 3
A44 A43 A42 A41 A28 A27 A26 A25 A12 A11 A10 A09
ROW 2
A40 A39 A38 A37 A24 A23 A22 A21 A08 A07 A06 A05
Minor Character 10
Minor Character 6
Minor Character 2
ROW 1
A36 A35 A34 A33 A20 A19 A18 A17 A04 A03 A02 A01
Minor Character 9
Minor Character 5
Minor Character 1
MAJOR CHARACTER 3
MAJOR CHARACTER 2
MAJOR CHARACTER 1

Note: Carry Logic not shown.

Figure A-1. Schematic Representation of Bus and Accumulator a decimal digit in the Honeywell 1800, the least significant decimal digit is stored in the first row of major character No. 1, the next higher-order decimal digit in the second row of major character No. 1, etc. The most significant decimal digit of a signed number is represented by the four bits in the third row of major character No. 3. In the accumulator, bits 45 through 48 are used to store either the high-order four bits of an unsigned number, or an arbitrary binary configuration when signed numbers are being operated upon.

The sign of the A operand is stored in a single flip-flop designated sign control (SGA). The content of SGA, the sign of the B operand, and the operation code determine the sign of the result and also the nature of the operation to be performed. For example, if SGA contains a plus sign, the B operand is negative, and the operation code specifies subtraction, the operation to be performed logically is addition. When addition is performed on signed numbers, bits 45 through 48 of the accumulator are filled with binary ones so that a carry out of bit 44 is reflected by a carry out of bit 48. Thus overflow may be determined by the presence of a carry out of bit 48 in addition of either signed 44-bit operands or unsigned 48-bit operands. In like manner, bits 45 through 48 are filled with binary zeros when subtracting signed numbers in order that borrows may be sensed at the same point for both signed and unsigned numbers. For addition of numbers with unlike signs, subtraction of numbers with like signs, or multiplication, the contents of SGA may change to correspond with the sign of the result generated in the accumulator.

As information is transferred from the bus to the accumulator, the four bits in each minor character receive the information simultaneously (in parallel) and the minor characters within a major character receive information serially. However, a group of three minor characters in the same row (1, 5, and 9, or 2, 6, and 10 - see Figure A-1) receive their information in parallel. This method of receiving and processing data characterizes the Honeywell *1800* as a parallel-serial-parallel machine.

The following detail steps are performed in executing a fixed-point addition instruction. First, the A operand is transferred from the memory to the bus and then is received by the accumulator in parallel-serial-parallel fashion. The B operand is then transferred from memory to the bus. As the first row of information is transmitted via the bus to the accumulator, each corresponding pair of bits, one from the bus and one from row 1 of the accumulator, is input to a gate buffer amplifier (GBA), a logical element in which the addition of the two binary digits takes place. The sum of the two digits is then transmitted to the fourth row. If a carry will occur within a minor character from this addition, it is anticipated and introduced as an input into the next higher-order GBA to be added to the bit from the accumulator (A operand) and the bit from the bus (B operand). If a carry occurs from the high-order binary digit of a minor character during the arrival time of rows 1, 2 or 3 at the GBA's, the carry is brought to the low-order GBA associated with the same major character at the time of arrival of the following row. Therefore, carries within a minor character or within a major character require no extra processing time.

If a carry occurs from the high-order bit position of a minor character in row 4 (in other words, from one major character to another), a second cycling of the accumulator is required. The carry is brought as input to the low-order GBA associated with the next major character as row 1 is again transmitted to row 4. (On this cycle, of course, there is no input to the GBA from the bus.)

A second cycle of the accumulator does not necessarily mean an extra memory cycle each time a carry occurs from a major character. The content of the accumulator is read from the output of the flip-flops of row 2 to storage and thus a "half cycle," to place the original contents of row 1 in row 2, is required before the contents of the accumulator are transferred to the bus and thence to memory. Therefore, if the carry from a major character propagates through no more than two minor characters, no additional memory cycle is needed. If the carry propagates further (including the worst case of a carry from major character No. 1 which propagates through major character Nos. 2 and 3 in turn), a maximum of one extra cycle is needed. Because the "borrow" logic includes the possibility of a carry from major character 3 end-around to major character 1, the addition of two words with unlike signs could require two extra memory cycles.

The above discussion is applicable to both binary and decimal fixed-point arithmetic. In the case of decimal operations, however, an additional operation takes place. As each minor character in the sum proceeds from row 4 to row 3, the 4-bit configuration is tested to see if it represents a decimal digit (0 through 9). If so, it proceeds as described above. If not, the 4-bit configuration is reduced by 10 (binary 1010) and a carry of one is introduced into the low-order bit of the next higher-order minor character. This discussion assumes that operands in decimal operations are made up of decimal numbers.

APPENDIX B

ORTHOTRONIC CONTROL

Orthotronic control is the technique incorporated in the Honeywell 1800 for the automatic detection and correction of errors that may occur with the use of magnetic tape. This file protection feature accomplishes error correction without the manual intervention and lost computer time so frequently associated with ordinary restart or rerun procedures. In other systems, the problem of restart and rerun becomes most serious when the record in error was properly recorded during a previous run but is now incapable of being read. It is to this type of error, the most costly in time and effort, that orthotronic control is applied most advantageously in the Honeywell 1800 system.

The basic principle of orthotronics may be compared to the accountant's practice of crossfooting. In crossfooting, a zero balance of rows and columns is obtained to insure accuracy. If a zero balance is not obtained, the crossfooting technique has accomplished its singular function of error detection. Orthotronic control not only performs this function of error detection but also provides a unique means of error correction.

The orthotronic technique involves three interdependent elements: frame parity, orthotronic control words, and channel parity. Frame parity is recorded in the ninth channel on tape. An "odd" parity system is used in the generation of the parity bit. The parity bit is the complement of the binary half adds of the bits in each frame. Thus, the parity bit is a one if there is an even number of ones in a frame (0, 2, 4, 6, 8 ones), and the parity bit is zero if there is an odd number of ones in a frame (1, 3, 5, 7 ones).

Before a new or altered record is written on tape, the machine instruction compute orthocount is used to compute two orthowords: one associated with the odd-numbered data words in the record, the other associated with the even-numbered words. These orthowords become part of the record written on tape. Orthoword 1 may be the orthocount of either the odd words or even words of a record. This is determined by the number of words to be orthocounted in the record (see Figure B-1). If the record contains an even number of data words, then orthoword 1 represents the orthocount of the odd words. If an odd number of data words is orthocounted, then orthoword 1 contains the orthocount of the even words. However, it should be noted that by working back from the orthoword, the relationship is invariant, regardless of the number of words. Each orthobit is an "odd" parity bit checking the corresponding bit positions of all the associated words.

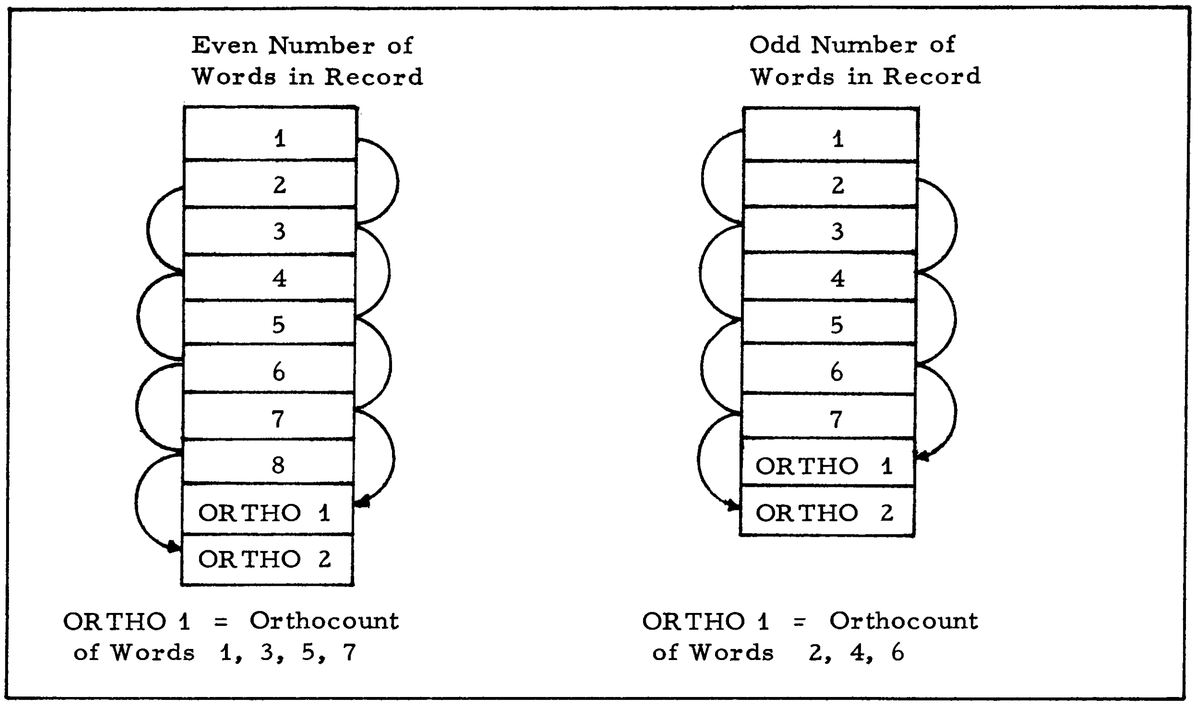

Even Number of Words in Record
1
2
3
4
5
6
7
8
ORTHO 1
ORTHO 2
ORTHO 1 = Orthocount of Words 1, 3, 5, 7
Odd Number of Words in Record
1
2
3
4
5
6
7
ORTHO 1
ORTHO 2
ORTHO 1 = Orthocount of Words 2, 4, 6

Figure B-1. Relation of Data Words to Orthowords

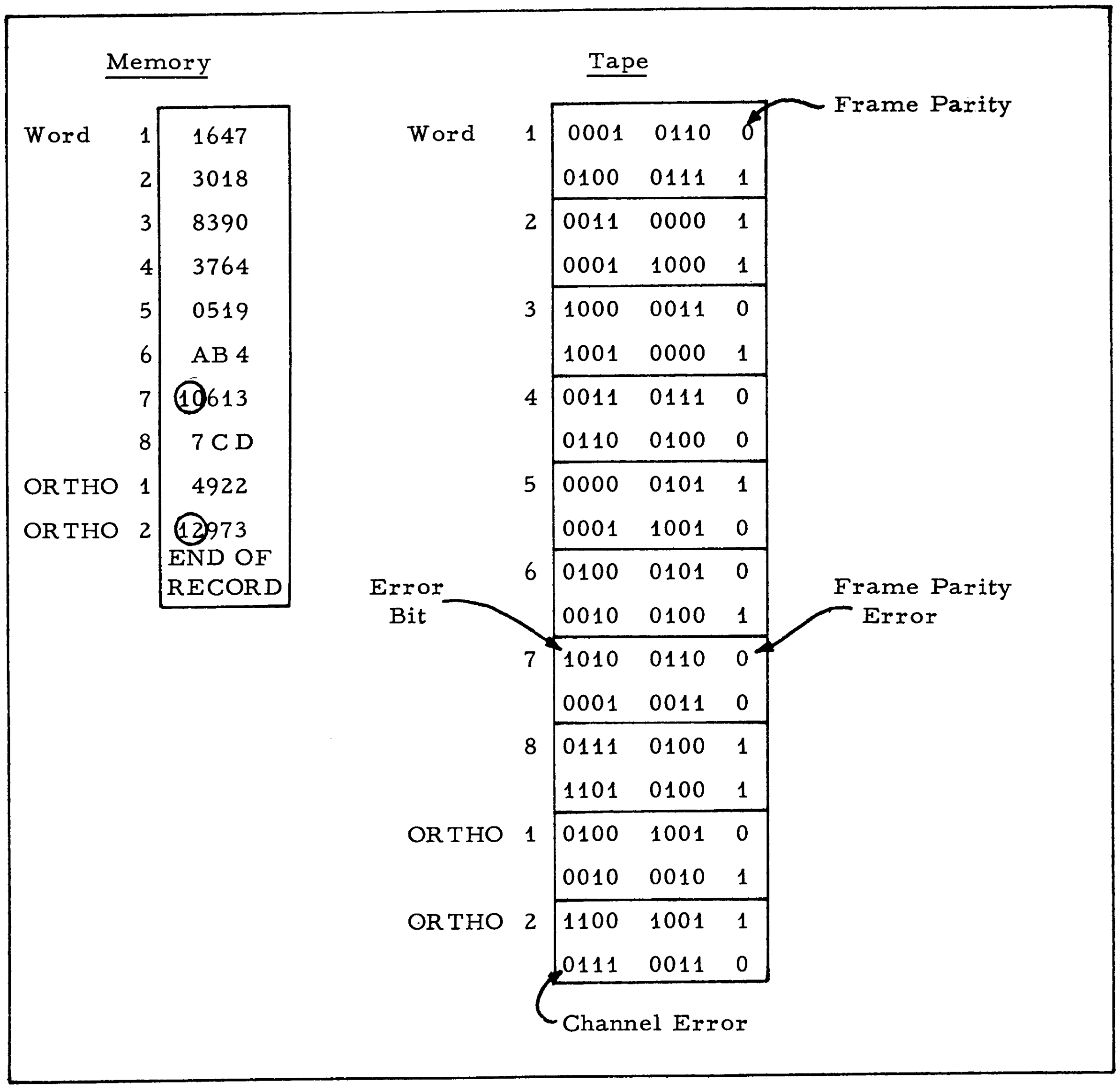
Consider a record in memory (for the sake of simplicity, 16-bit words are shown). Figure B-2 shows the method used to compute the orthowords. Each orthoword is the complement of the binary half add of the words associated with it.

			Odd Words				Even Words				
			(1111	1111	1111	1111	1111	1111	1111	1111)	Complement Bits
Word	1	1647	0001	0110	0100	0111					
	2	3018					0011	0000	0001	1000	
	3	8390	1000	0011	1001	0000					
	4	3764					0011	0111	0110	0100	
	5	0519	0000	0101	0001	1001					
	6	A B 4					0100	0101	0010	0100	
	7	2613	0010	0110	0001	0011					
	8	7 C D					0111	0100	1101	0100	
ORTHO	1	4922	0100	1001	0010	0010					
ORTHO	2	12973					1100	1001	0111	0011	
		END OF RECORD									

Figure B-2. Computation of Orthowords

Channel parity checking is automatically performed in the buffer to provide a longitudinal check of information being read or written. The generated orthotronic words guarantee an even channel parity up to the end-of-record word. The end-of-record word is such that its inclusion guarantees an odd channel parity. Thus, double errors in a frame, which will escape detection by frame parity, will be picked up by the channel parity check.

Using the example shown in Figure B-2, assume that a bit was altered in the high-order decimal digit of word 7 to make it 1010 instead of 0010. Figure B-3 shows the words in memory and a simplified, hypothetical representation of the same words on tape, with frame parity bits.



Memory

Word	1	1647
	2	3018
	3	8390
	4	3764
	5	0519
	6	AB 4
	7	10613
	8	7 C D
ORTHO	1	4922
ORTHO	2	12973
		END OF RECORD

Tape

Word	1	0001	0110	0
		0100	0111	1
	2	0011	0000	1
		0001	1000	1
	3	1000	0011	0
		1001	0000	1
	4	0011	0111	0
		0110	0100	0
	5	0000	0101	1
		0001	1001	0
	6	0100	0101	0
		0010	0100	1
	7	1010	0110	0
		0001	0011	0
	8	0111	0100	1
		1101	0100	1
ORTHO	1	0100	1001	0
		0010	0010	1
ORTHO	2	1100	1001	1
		0111	0011	0

Figure B-3. Orthotronic Check of Tape Error

When this record is read, the error is automatically detected by the frame parity check. The automatic channel parity check also shows up the error, since a binary half add of the bits in the left-most channel produces a one bit instead of a zero bit. If, in addition, a bit in the second high-order digit of word 7 were altered so that it became 0111 instead of 0110, this frame would now appear as follows on tape:

$$\begin{tikzcd}
\text{Error Bit} \arrow[r] & 1010 & 0111 & 0 & \text{Parity Bit} \arrow[l] \\
 & & \text{Error Bit} \arrow[u] & &
\end{tikzcd}$$

This combination of errors would escape the frame parity check only to be caught by the channel parity computation.

Regardless of the means of error detection, the result is the same: an unprogrammed transfer of control is initiated when the next read or write instruction to the same device is executed.[1] The procedure to be followed when the error occurs on writing is to read backward over the erroneous record and then rewrite it. The procedure to be followed when the error occurs on reading is to orthocount the entire record again (including the bad word and the orthowords), using the compute orthocount instruction. The result of this procedure is illustrated in Figure B-4, using the example in which a bit was altered in the high-order digit of word 7.

			Odd Words				Even Words				
			(1111	1111	1111	1111	1111	1111	1111	1111)	Complement Bits
Word	1	1647	0001	0110	0100	0111					
	2	3018					0011	0000	0001	1000	
	3	8390	1000	0011	1001	0000					
	4	3764					0011	0111	0110	0100	
	5	0519	0000	0101	0001	1001					
	6	AB 4					0100	0101	0010	0100	
	7	$\boxed{10}$613	1010	0110	0001	0011					
	8	7CD					0111	0100	1101	0100	
ORTHO	1	4922	0100	1001	0010	0010					
ORTHO	2	$\boxed{12}$973					1100	1001	0111	0011	
NEW ORTHO	1	8000	1000	0000	0000	0000					
NEW ORTHO	2	0000					0000	0000	0000	0000	

Figure B-4. Orthocount of Error Record

[1]As noted in Section XI, a read instruction checks the previous read while a write instruction checks the previous write.

This computation shows that the error lies in one of the odd words (1, 3, 5, or 7). The check parity instruction is then used to isolate the bad word (and/or the bad frame within the word, by masking the instruction). The check parity instruction also assigns correct parity to the word. If correction is to be made on a word basis, a binary half add is used to add the new orthoword and the bad word (with adjusted parity) to restore the correct word. If correction is to be made on a frame basis, a properly masked half add is used to restore the frame. If correction is made on a word basis, the binary half add produces the result shown in Figure B-5.

Bad Word	7	1010	0110	0001	0011	
New Ortho	1	1000	0000	0000	0000	
Restored Word	7	0010	0110	0001	0011	(cf. Figure B-2)

Figure B-5. Correction of Tape Error -- First Method

A different procedure for reconstructing the bad word would be first to locate it by the check parity instruction and then to cancel it by substituting a word of all binary zeros. Next, orthocount the entire record including the cancelled word and the orthowords, as shown in Figure B-6.

		Odd Words				
		(1111	1111	1111	1111)	Complement Bits
Word	1	0001	0110	0100	0111	
	3	1000	0011	1001	0000	
	5	0000	0101	0001	1001	
(Bad Word)	7	0000	0000	0000	0000	(Binary Zeros Substituted)
Old ORTHO		0100	1001	0010	0010	
New ORTHO		0010	0110	0001	0011	

Figure B-6. Correction of Tape Error -- Second Method

The new orthoword equals the reconstructed word (cf. Figure B-2) and may now be substituted for the word of binary zeros. This second technique can be used only when errors are detected in a single word associated with each orthoword. For example, this method can be used to correct an error in one even word and one odd word in the record, but not to reconstruct two odd or two even words. The technique illustrated by Figure B-5, on the other hand, may be used to reconstruct errors in several words or frames, as long as there is not more than one frame parity or ortho error associated with each of the twelve frames comprising the two orthowords.

APPENDIX C

TIMING SUMMARY

Instruction	Basic Time in Memory Cycles (mc)[1]	Modifications of Basic Time
<u>BA[5], DA[5], WA, BS[5], DS[5], WD, HA, SM</u>		
A. Unmasked		
1. If B is inactive, and		
a. A is active, C inactive	3	Add: 1 mc if A is indexed
b. A is inactive, C active	4	
c. A and C are both active	5	Add: 1 mc if A is indexed
2. If B is active	4	Add: 1 mc if A is inactive 1 mc if A or B is indexed *1 mc if A is a direct special register or indirect memory location address <u>and</u> B is indexed *1 mc if B is a direct special register or indirect memory location address <u>and</u> C is indexed *2 mc if A and B are direct special register or indirect memory location addresses <u>and</u> C is indexed
B. Masked		
1. If B is inactive, and		
a. A is active, C inactive	4	Add: 1 mc if A is indexed
b. A is inactive, C active	6	
c. A and C are both active	7	Add: 1 mc if A is indexed
2. If B is active	6	Add: 1 mc if A is inactive 1 mc if A or B is indexed Sub: 1 mc if C is inactive
<u>BT[2], DT[2]</u>		
A. If B address (n) = 0	2	
B. If B address (n) > 0	3 + n	Add: 1 mc if A is indexed
<u>BM</u>	33	Add: 1 mc for each indexed address
<u>DM</u>	27	Add: 1 mc for each indexed address

Instruction	Basic Time in Memory Cycles[1]	Modifications of Basic Time
NA, NN, LA, LN		
A. Unmasked	4	Add: 1 mc if A or B (or both) is indexed 1 mc if A is direct special register or indirect memory location address and B is indexed 1 mc if B is direct special register or indirect memory location address
B. Masked	5	Add: 1 mc if A or B (or both) is indexed
TX		
A. Unmasked	3	Add:*1 mc if A is indexed *1 mc if A is direct special register or indirect memory location address and C is indexed *2 mc if A is indexed special register or indexed indirect memory location address and C is indexed
B. Masked	5	Add: 1 mc if A or C (or both) is indexed
TS		
A. Unmasked		
1. If B is inactive, and		
a. A is inactive	4	Add: 1 mc if A is indexed Sub: 2 mc if C is inactive
b. A is active	5	
2. If B is active	4	Add:*1 mc if A is inactive *1 mc if A is inactive and B is indexed *1 mc if A or B is indexed *1 mc if A is direct special register or indirect memory location address and B is indexed *2 mc if A is indexed special register or indexed indirect memory location address and B is indexed 1 mc if B is direct special register or indirect memory location address

Instruction	Basic Time in Memory Cycles[1]	Modifications of Basic Time
TS (cont)		1 mc if C is indirect memory location address 1 mc if C is indexed
B. Masked		
1. If B is inactive, and		
a. A is inactive	5	Add: 1 mc if A is indexed 1 mc if C is indexed Sub: 2 mc if C is inactive
b. A is active	6	
2. If B is active	6	Add:*1 mc if A is inactive *1 mc if A is inactive and B is indexed *1 mc if A or B is indexed 1 mc if C is indexed
MT[2]		
A. If B address (n) = 0	2	
B. If B address (n) > 0	1 + 2n	Add:*n if A is indexed *n if A and C are both indexed *1 mc if C is indexed *n + 1 if A is a direct special register or indirect memory location address and C is indexed *2n if A is indexed special register or indexed indirect memory location address and C is indexed
TN[2]		
A. If B address (n) = 0	2	
B. If B address (n) > 0	5 + 2n	Add: 1 mc if A is indexed indirect memory location address n if A is direct or indexed special register address Add: 1 mc if C is indexed indirect memory location address n if C is direct or indexed special register address
IT[2], RT[2]	7 + 2n	Add: 1 mc if A is indexed indirect memory location address 1 mc if C is indexed indirect memory location address

Instruction	Basic Time in Memory Cycles[1]	Modifications of Basic Time
SWS, SPS, SWE, SPE	5 + k	Total no. 16-, 4-, 1-bit shifts / k: 0 / 0; 1 / 0; 2 / 0; 3 / 1; 4 / 1; 5 / 2; 6 / 2; 7 / 3; 8 / 3; 9 / 4 Add: 1 mc if A is indexed 1 mc if C is indexed 2 mc if C is direct special register address
SSL	6 + k	For values of k, see table for SWS, etc. Add: 1 mc if A is indexed 1 mc if C is indexed 1 mc if C is indirect memory location address
SS	5	Add: 1 mc if A or B is indexed 1 mc if A is direct special register or indirect memory location address and B is indexed 1 mc if B is direct special register or indirect memory location address and C is indexed 2 mc if C is direct special register address
EX		
A. If C is inactive, and B is active	4	Add: 1 mc if A or B is indexed 1 mc if A is direct special register or indirect memory location address and B is indexed
B. If C is active	5	Add: 1 mc if A or B is indexed 1 mc if A is direct special register or indirect memory location address and B is indexed

Instruction	Basic Time in Memory Cycles[1]	Modifications of Basic Time
EX (cont)		1 mc if B is direct special register or indirect memory location address and C is indexed 2 mc if C is direct special register address
RF[3], RB[3], WF[3]		
A. If A and/or B are active and		
1. C is inactive	5	
2. C is active	7	Add: 1 mc if C is indexed
B. If A and B are inactive and		
1. C is inactive	4	
2. C is active	6	Add: 1 mc if C is indexed
C. If end or beginning of tape	6	
D. If device is busy or error condition is stored.	4	
RW[3]	3	Add: 1 mc if device is busy
PRA, PRD, PRO		
A. If C is inactive	4	Add: 1 mc if A or B (or both) is indexed
B. If C is active	5	Add: 1 mc if A or B (or both) is indexed 1 mc if C is indexed 1 mc if C is indirect memory location address In addition, 7 mc are required to print each character, at the rate of 1 character every 100ms (approximately)
CC		
A. If B is inactive[2]	11 + n	Add: 1 mc if C is indexed
B. If B is active[4]	$9+2j+n_1$ $+n_2+..n_j$	Add: 1 mc if A is direct special register or indirect memory location address and B is indexed 1 mc if B is indexed special register or indirect memory location address 1 mc if C is indexed
CP		
A. Unmasked	4	Add: *1 mc if A or B (or both) is indexed *1 mc if A is direct special register or indirect memory

Instruction	Basic Time in Memory Cycles[1]	Modifications of Basic Time
CP (cont)		location address <u>and</u> B is indexed *2 mc if A is indexed special register or indexed indirect memory location address <u>and</u> B is indexed 1 mc if B is indirect memory location address 1 mc if C is direct special register or indirect memory location address 1 mc if C is indexed
B. Masked	6	Add: 1 mc if A or B (or both) is indexed 1 mc if C is indexed
<u>MPC</u>	4	
<u>PR</u>	2	
<u>S</u>	7	Add: 1 mc if D/I bit = 1 1 mc if C is indexed
<u>FBA, FBS[6]</u>	4 minimum 4.95 average 6 maximum	Add: 1 mc if A or B (or both) is indexed *1 mc if A is indirect and B is indexed *1 mc if B is indirect and C is indexed Sub: *1 mc if B is inactive
<u>FDA, FDS[6]</u>	5 minimum 6.5 average 7 maximum	Add: 1 mc if A or B (or both) is indexed *1 mc if A is indirect and B is indexed *1 mc if B is indirect and C is indexed Sub: *1 mc if B is inactive
<u>FBAU, FBSU[6]</u>	4 minimum 5 average 6 maximum	Add: 1 mc if A or B (or both) is indexed *1 mc if A is indirect and B is indexed *1 mc if B is indirect and C is indexed Sub: *1 mc if B is inactive
<u>FDAU, FDSU[6]</u>	5 minimum 6 average 7 maximum	Add: 1 mc if A or B (or both) is indexed *1 mc if A is indirect and B is indexed *1 mc if B is indirect and C is indexed

Instruction	Basic Time in Memory Cycles[1]	Modifications of Basic Time
FDAU, FDSU[6] (cont)		Sub: *1 mc if B is inactive
FBAE, FBSE[6]	4 minimum 6 average 7 maximum	Add: 1 mc if A or B (or both) is indexed *1 mc if A is indirect and B is indexed *1 mc if B is indirect and C is indexed Sub: *1 mc if B is inactive
FBM[6] A. If A and B are non-zero	5 minimum 5 average 6 maximum	Add: 1 mc if A or B (or both) is indexed *1 mc if A is indirect and B is indexed *1 mc if B is indirect and C is indexed Sub: *1 mc if B is inactive
B. If A or B is zero	4	
FDM[6] A. If A and B are non-zero	8 minimum 10 average 12 maximum	Add: 1 mc if A or B (or both) is indexed *1 mc if A is indirect and B is indexed *1 mc if B is indirect and C is indexed Sub: *1 mc if B is inactive
B. If A or B is zero	4	
FBD	11 minimum 14 average 17 maximum	Add: 1 mc if A or B (or both) is indexed *1 mc if A is indirect and B is indexed *1 mc if B is indirect and C is indexed Sub: *1 mc if B is inactive
FDD	17 minimum 20 average 22 maximum	Add: 1 mc if A or B (or both) is indexed *1 mc if A is indirect and B is indexed *1 mc if B is indirect and C is indexed Sub: *1 mc if B is inactive
BD	12 minimum 15 average	Add: 1 mc if A or B (or both) is indexed *1 mc if A is indirect and B is indexed

Instruction	Basic Time in Memory Cycles[1]	Modifications of Basic Time
BD (cont)	18 maximum	*1 mc if B is indirect and C is indexed Sub: *1 mc if B is inactive
DD	20 minimum 22 average 23 maximum	Add: 1 mc if A or B (or both) is indexed *1 mc if A is indirect and B is indexed *1 mc if B is indirect and C is indexed Sub: *1 mc if B is inactive
FLN, FNN	4	Add: 1 mc if A or B (or both) is indexed *1 mc if A is indirect and B is indexed *1 mc if B is indirect
FFN	4 minimum 4 average 5 maximum	Add: 1 mc if A or B (or both) is indexed *1 mc if A is indirect and B is indexed *1 mc if B is indirect and C is indexed Sub: *1 mc if B is inactive
ULD	4	Add: 1 mc if A or C (or both) is indexed or indirect
FCON A. Fixed Decimal to Floating Binary Conversion B. Floating Binary to Fixed Decimal Conversion	 20 9	Add: 1 mc if A or B (or both) is indexed *1 mc if A is indirect and B is indexed *1 mc if B is indirect and C is indexed Sub: *1 mc if B is inactive

NOTES:

1. One memory cycle (mc) equals 2 microseconds. All addresses assumed active except as otherwise specified. Address configurations for which the behavior of the system is unspecified have not been considered.
2. n = number of words accumulated, transferred. or orthocounted.
3. If the instruction is delayed by an interlock, 4 memory cycles are required for the first attempt to perform it, and an additional 4 cycles are required each time that any buffer completes a read or write instruction before this instruction proceeds.

4. j = number of items in record.

   $n_1, n_2, \ldots n_j$ = number words in item 1, number words in item 2, etc., up to n_j, number words in last item.

5. Under certain conditions, as explained in Appendix A, one or two additional memory cycles will be needed in excess of those computed from this summary.

6. See comments in Section XIII.

* Conditions so marked are mutually exclusive.

APPENDIX D

CONTROL ERRORS

When the Honeywell 1800 stops because of a control error, the exact error is indicated by a significant pattern of binary digits in positions 4 through 8 of the program control register (PCR). A configuration of lights at the console reflects this bit pattern in a one-to-one correspondence. Figure D-1 lists the control errors, displays the particular bit configuration that identifies each, and explains what caused the error to occur.

When a control error occurs, all main memory and control memory cycles stop, including any buffer cycles. Peripheral devices proceed to the end of the record if they are reading; output devices other than tapes are unable to complete the current instruction because of lack of buffer cycles. An end-of-record word is automatically generated for each tape that was being written on. Address selectors and local registers of main memory and control memory are frozen, including the cycle counters.

Bit Position					Type of Error
4	5	6	7	8	
0	0	0	0	1	Error in any type of main memory addressing.
0	0	0	1	0	Error in decrementing the shift control register during shift, multiply, or accumulate instructions.
0	0	0	1	1	Error in addressing an implicitly referenced special register, such as the sequence counter.
0	0	1	0	0	Error in the result of the accumulator's operation in an add, subtract, or transfer instruction.
0	0	1	0	1	The control check pulse, which initiates peripheral operation, has been active too long. More than one peripheral device may have been activated.
0	0	1	1	0	This indicates either a failure in the memory cycle counter (more than one cycle active at the same time), an error in operation code decoding, or a timing error in the control unit.
0	0	1	1	1	Error in the transfer of the group indicator from traffic control, the console, or the control register to the control memory.
0	1	0	0	0	Error in masking.
0	1	0	0	1	Error in selecting the partial products in the multiply instruction.
0	1	0	1	0	Bad parity in the instruction word.

Figure D-1. Control Error Bit Configurations

Bit Position					Type of Error
4	5	6	7	8	
0	1	0	1	1	Unspecified.
0	1	1	0	0	Combination error - No. 4 and No. 8.
0	1	1	0	1	Unspecified.
0	1	1	1	0	Error in the result of the accumulator's operation in a shift or multiply instruction.
0	1	1	1	1	Unspecified.
1	0	0	0	0	Unspecified.
1	0	0	0	1	Error during the compute orthocount operation.
1	0	0	1	0	Error in the instruction word during an unprogrammed transfer cycle or during a simulator instruction.
1	0	0	1	1	Unspecified.
1	0	1	0	0	Mantissa error in the floating-point unit.
1	0	1	0	1	The address read out of control memory does not correspond to the address in the control memory address selector.
1	0	1	1	0	Exponent error in the floating-point unit.
1	0	1	1	1	Error in transferring information to or from the floating-point unit.
1	1	0	0	0	Unspecified.
1	1	0	0	1	An error in the operation code portion of the instruction word.
1	1	0	1	0	The address read out of main memory does not correspond to the address in the main memory address selector.
1	1	0	1	1	Unspecified.
1	1	1	0	0	Unspecified.
1	1	1	0	1	Unspecified.
1	1	1	1	0	Unspecified.
1	1	1	1	1	Unspecified.

Figure D-1 (cont). Control Error Bit Configurations

Key Punch	Card Code	Honeywell 1800 Code	Octal	Standard Printer	High Speed Printer	Console	Key Punch	Card Code	Honeywell 1800 Code	Octal	Standard Printer	High Speed Printer	Console
0	0	000000	00	0 (zero)	0	0	-	X	100000	40	- (minus)	-	-
1	1	000001	01	1	1	1	J	X, 1	100001	41	J	J	J
2	2	000010	02	2	2	2	K	X, 2	100010	42	K	K	K
3	3	000011	03	3	3	3	L	X, 3	100011	43	L	L	L
4	4	000100	04	4	4	4	M	X, 4	100100	44	M	M	M
5	5	000101	05	5	5	5	N	X, 5	100101	45	N	N	N
6	6	000110	06	6	6	6	O	X, 6	100110	46	O	O	O
7	7	000111	07	7	7	7	P	X, 7	100111	47	P	P	P
8	8	001000	10	8	8	8	Q	X, 8	101000	50	Q	Q	Q
9	9	001001	11	9	9	9	R	X, 9	101001	51	R	R	R
	8, 2*	001010	12	9*	'	'		X,8,2*	101010	52	R*	#	#
#	8, 3	001011	13	=	=	=	$	X,8,3	101011	53	$	$	$
@	8, 4	001100	14	- (minus)	:	:	*	X,8,4	101100	54	*	*	*
Space	Blank	001101	15	Blank	Blank	Blank		X,8,5*	101101	55	**	"	"
	8, 6*	001110	16	=*	Blank*	¢		X,8,6*	101110	56	$*	Blank*	↓
	8, 7*	001111	17	- (minus)*	&	&		X, 0	101111	57	0*	Blank*	?
&	R	010000	20	+	+	+		8, 5*	110000	60	-(minus)*	Blank*	◇
A	R, 1	010001	21	A	A	A	/	0, 1	110001	61	/	/	/
B	R, 2	010010	22	B	B	B	S	0, 2	110010	62	S	S	S
C	R, 3	010011	23	C	C	C	T	0, 3	110011	63	T	T	T
D	R, 4	010100	24	D	D	D	U	0, 4	110100	64	U	U	U
E	R, 5	010101	25	E	E	E	V	0, 5	110101	65	V	V	V
F	R, 6	010110	26	F	F	F	W	0, 6	110110	66	W	W	W
G	R, 7	010111	27	G	G	G	X	0, 7	110111	67	X	X	X
H	R, 8	011000	30	H	H	H	Y	0, 8	111000	70	Y	Y	Y
I	R, 9	011001	31	I	I	I	Z	0, 9	111001	71	Z	Z	Z
	R, 8, 2*	011010	32	I*	;	;		0,8,2*	111010	72	Z*	@	@
●	R, 8, 3	011011	33	.	.	.	,	0,8,3	111011	73	,	,	,
□	R, 8, 4	011100	34	)	)	)	%	0,8,4	111100	74	(	(	(
	R, 8, 5*	011101	35	)*	%	%		0,8,5*	111101	75	(*	C_R	C_R
	R, 8, 6*	011110	36	.*	■	■		0,8,6*	111110	76	,*	Blank*	□
	R, 0	011111	37	0*	Blank*	△		0,8,7*	111111	77	(*	Blank*	⊗

Notes:	Key Punch:	Use MLT PCH key to overpunch omitted characters.
	Card Code:	* legal in illegal punch check Mode 2 only for card readers.
	Printer:	* indicates symbol which will be printed by otherwise non-standard printer bit configuration.

Table I. Honeywell 1800 Coding and Punched or Printed Equivalents

			1 2 3 4 5 6 7	8 9 10 11 12
GENERAL UNMASKED OR MASKED	BA	BINARY ADD		0 1 0 0 1
	DA	DECIMAL ADD		0 0 0 0 1
	WA	WORD ADD		0 1 1 0 1
	BS	BINARY SUBTRACT		1 1 0 0 1
	DS	DECIMAL SUBTRACT	MASKED	1 0 0 0 1
	WD	WORD DIFFERENCE		1 1 1 0 1
	NA	INEQUALITY COMPARISON, ALPHABETIC	S M M M M M 1	0 1 1 0 0
	NN	INEQUALITY COMPARISON, NUMERIC	S 1 0 A B C 0	0 1 0 0 0
	LA	LESS THAN OR EQUAL COMPARISON, ALPHABETIC		1 1 1 0 0
	LN	LESS THAN OR EQUAL COMPARISON, NUMERIC	UNMASKED	1 1 0 0 0
	TX	TRANSFER (A) TO C		1 0 0 0 0
	TS	TRANSFER (A) TO B AND GO TO C		0 0 1 0 0
	HA	HALF ADD (MOD. 2)		1 0 1 0 1
	SM	SUPERIMPOSE		0 0 1 0 1
	CP	CHECK PARITY		1 0 1 0 0
GENERAL UNMASKED	BM	BINARY MULTIPLY	S 0 0 A B C 0	0 1 0 1 1
	DM	DECIMAL MULTIPLY		0 0 0 1 1
	BT	BINARY ACCUMULATE	S 1 0 A B C 0	0 1 0 1 1
	DT	DECIMAL ACCUMULATE		0 0 0 1 1
	MT	MULTIPLE TRANSFER	S 0 0 A B C 0	1 0 0 0 0
	TN	N-WORD TRANSFER		1 0 0 0 0
	CC	COMPUTE ORTHOCOUNT	S 0 1 A B C 0	0 1 0 0 0
	IT	ITEM TRANSFER		1 1 0 0 0
	EBA	EXTENDED BINARY ADD		0 1 0 1 1
	EBS	EXTENDED BINARY SUBTRACT	S 1 1 A B C 0	1 1 0 1 1
	RT	RECORD TRANSFER		1 1 0 0 0
	MPC	CONTROL PROGRAM	S 1 0 A B C 0	0 0 0 0 0
	PR	PROCEED	X 0 0 0 X X 0	0 0 0 0 0
INHERENT MASK	SWS	SHIFT WORD AND SUBSTITUTE (PROTECTED)		0 0 1 1 0
	SPS	SHIFT PRESERVING SIGN AND SUBSTITUTE (PROTECTED)		0 0 0 1 0
	SWE	SHIFT WORD AND EXTRACT (UNPROTECTED)	S 1 0 A B C 0	0 1 1 1 0
	SPE	SHIFT PRESERVING SIGN AND EXTRACT (UNPROTECTED)		0 1 0 1 0
	SSL	SHIFT AND SELECT (PROTECTED)		1 0 1 1 0
	SS	SUBSTITUTE (PROTECTED)	S 0 0 A B C 0	0 0 1 1 0
	EX	EXTRACT (UNPROTECTED)		0 1 1 1 0
PERIPHERAL AND PRINT	RF	READ FORWARD		1 1 0 1 0
	RB	READ BACKWARD	P P P P P P 1	0 1 0 1 0
	WF	WRITE FORWARD		0 1 1 1 0
	RW	REWIND		0 0 0 1 0
	PRA	PRINT ALPHA		
	PRD	PRINT DECIMAL	S X X A B C 1	0 0 1 1 0
	PRO	PRINT OCTAL		
SIMU-LATOR	S	SIMULATOR		
		Direct	0 X X X X X X	X X 1 1 1
		Indexed	1 X X X X X X	X X 1 1 1
SCIENTIFIC	FBA	FLOATING BINARY ADD		0 0 0 0 1
	FDA	FLOATING DECIMAL ADD		1 0 0 0 1
	FBS	FLOATING BINARY SUBTRACT	S 0 1 A B C 0	0 1 0 0 1
	FDS	FLOATING DECIMAL SUBTRACT		1 1 0 0 1
	FBD	FLOATING BINARY DIVIDE		0 0 1 0 1
	FDD	FLOATING DECIMAL DIVIDE		1 0 1 0 1
	FBAU	FLOATING BINARY ADD, UNNORMALIZED		0 0 0 0 1
	FDAU	FLOATING DECIMAL ADD, UNNORMALIZED		1 0 0 0 1
	FBSU	FLOATING BINARY SUBTRACT, UNNORMALIZED		0 1 0 0 1
	FDSU	FLOATING DECIMAL SUBTRACT, UNNORMALIZED	S 1 1 A B C 0	1 1 0 0 1
	FBM	FLOATING BINARY MULTIPLY		0 0 1 0 1
	FDM	FLOATING DECIMAL MULTIPLY		1 0 1 0 1
	ULD	MULTIPLE UNLOAD		0 1 1 0 1
	FBAE	FLOATING BINARY ADD, EXTENDED PRECISION		0 0 0 0 1
	FBSE	FLOATING BINARY SUBTRACT, EXTENDED PRECISION		0 1 0 0 1
	BD	FIXED BINARY DIVIDE	S 0 0 A B C 0	0 0 1 0 1
	DD	FIXED DECIMAL DIVIDE		1 0 1 0 1
	FFN	FIXED-TO-FLOATING NORMALIZE		0 1 1 0 1
	FCON	CONVERSION		1 1 1 0 1
	FLN	NORMALIZED LESS THAN COMPARISON		1 1 0 0 0
	FNN	NORMALIZED INEQUALITY COMPARISON	S 1 0 A B C 0	0 1 1 0 0

Notes: PPPPPP = PERIPHERAL ADDRESS
S = SEQUENCE, COSEQUENCE CODE
A, B, C, = MEMORY DESIGNATOR
MMMMM = MASK ADDRESS
X = IRRELEVANT

Table II. Honeywell 1800 Command Codes

APPENDIX F

MEMORY EXTENSIONS BEYOND 32,768 WORDS

The memory capacity of the H-1800 can be extended to 49, 152 or 65, 536 words by the addition of either one or two model 1802-1 additional memory modules. In such a system, the first 16 memory banks (locations 0 through 32, 767) are called the first array of memory; all banks above location 32, 767 are called the second array.

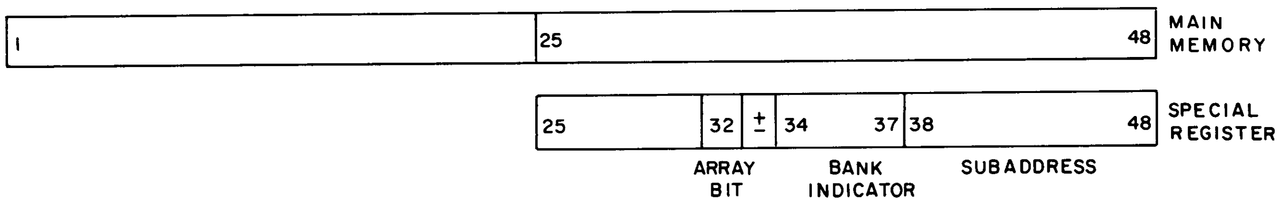
Address Compatibility

The addressing associated with extended memory provides for upward compatibility with smaller H-800/1800 systems. Programs can run in either of the two memory arrays but cannot cross the boundary between the two arrays. The highest address in each array (32, 767 and either 49, 151 or 65, 535) is a stopper address. However, a program can reference locations in either array by means of an extended control memory word. Two new instructions (described below) are provided to manipulate the added special register bits.

Note that the "reserved" memory locations (the submultiple locations and the console typewriter buffer, see page 44) exist only in the first memory array. Therefore, when an existing program is run in the second array, it can reference these locations only implicitly.

Control Memory

In a system with extended memory, each special register has the capacity to store 24 data bits plus two checking bits. The functions of the 24 data bits and their relationships to the bits of a main memory word are as follows:



Bits 25-31	These bits are always zero.
Bit 32	Array bit - 0 indicates an address in the first array, 1 indicates an address in the second array.
Bit 33	Sign bit.
Bits 34-48	These bits store the bank indicator and subaddress of an address in the range 0 to 32, 767. Together with the array bit, they can represent any address up to 65, 535. Incrementing and augmenting do not propagate to the left of bit 34. Note that the combination of ones in bits 32 and 34 represent a non-existent address in a 49, 152-word system.

Explicit Addressing

The rules of explicit addressing in extended H-1800 memories are as follows:

1. Direct Memory Location Address. The 11-bit subaddress from the address group is combined with the array bit, sign, and bank indicator from the controlling counter to form a complete main memory location address.
2. Indexed Memory Location Address. The 8-bit augmenter from the address group is combined with the entire address stored in the referenced index register to form the effective address. Augmentation is not permitted to propagate to the left of bit 34. Note that in a 49,152-word system, a non-existent address is created if a carry is propagated into bit position 34 and bit 32 is a one.
3. Indirect Memory Location Address. The referenced special register is selected and incremented in the normal manner. The entire address stored in this register is incremented to form a complete main memory address, except that incrementation is not permitted to propagate to the left of bit 34. Note that in a 49,152-word system, a non-existent address is created if a carry is propagated into bit position 34 and bit 32 is a one.
4. Indexed Indirect Memory Location Address. The 8-bit augmenter from the address group is combined with the low-order 14 bits of the referenced index register in the normal manner to form a special register address. The contents of this special register are used as described under indirect memory location addressing.
5. Direct Special Register Address. When a special register is addressed directly as the source of an operand, the low-order 16 bits of the special register are extracted to form the operand and are incremented in the normal manner. Bit 32 is not altered in the special register and is not part of the operand.

   When a special register is addressed directly as a result location, the low-order 16 bits of the result are substituted into the special register, the array bit (32) is substituted from the controlling counter, and the high-order seven bits are unaffected.
6. Indexed Special Register Address. The 8-bit augmenter from the address group is combined with the low-order 14 bits of the referenced index register in the normal manner to form a special register address. The contents of this special register are used as described under direct special register addressing.

Exceptions to the rules for direct and indexed special register addressing are stated below under "Extended Binary Add." Note that the address stored in the command code field of a simulator instruction must be either a direct or an indexed memory location address. The same is true of the addresses generated from the A and C address groups and stored in AU1 and AU2. Therefore, in an extended H-1800 memory, these addresses are formed as described above under direct and indexed memory location addressing.

Implicit Addressing

The rules of implicit addressing in extended H-1800 memories are as follows:

1. Sequencing Counters. All of the bits in the controlling counter are used to select an instruction. Incrementing does not propagate to the left of bit 34 (in a 49,152-word system, it does not propagate into bit 34).

2. Sequence Changes. The effective address for a sequence change is formed as described above under direct, indexed, indirect, or indexed indirect memory location addressing. All bits of the effective address are stored in the designated sequencing counter. The pertinent history register also receives all bits from the corresponding counter.

   When the C address group of a shift and select (SSL) instruction is a direct memory location address, the new subaddress is placed in the specified sequencing counter, together with the array bit, sign, and bank indicator from the counter that selected the instruction. Any other address type in this position is interpreted according tothe existing rules; when a special register is used to form the effective address, all bits of that register are used.

3. Masking. All pertinent bits of the mask index register are used to form the mask address.

4. Unprogrammed Transfers. All bits of the unprogrammed transfer register are used to form the subsequence address.

5. Use of AU-CU Counters. When an instruction loads a memory location address into an AU-CU counter, the effective address is formed as described above under direct, indexed, indirect, or indexed indirect memory location addressing, and all bits of the effective address are stored in AU1 or AU2. Incrementation of the counter during instruction execution does not propagate to the left of bit 34. Note that in a 49,152-word system, a non-existent address is created if a carry is propagated into bit position 34 and bit 32 is a one.

   When an instruction loads a special register address into an AU-CU counter, the counter is loaded and incremented as described on page 60.

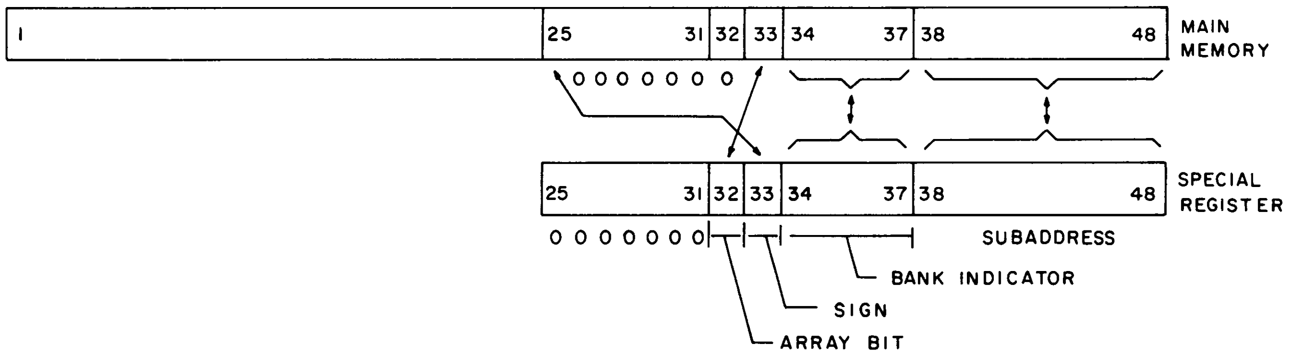
6. Read-Write Counters. Since the A address group of a peripheral read or write instruction can only refer to a direct or an indexed memory location, the effective address loaded into the read or write address counter is formed as described above under direct or indexed memory location addressing. Incrementing of the counter during instruction execution does not propagate to the left of bit 34. Note that in a 49,152-word system, a non-existent address is created if a carry is propagated into bit position 34 and bit 32 is a one.

   When a read-write counter is loaded from the word referenced by a distributed read-write counter, the low-order 16 bits of the referenced memory word are substituted into the read-write counter, and the value of the array bit is preserved.

Extended Binary Add (EBA) Instruction

The extended binary add instruction is an unmasked general instruction which is used in an H-1800 system with extended memory, along with the extended binary subtract instruction (see below), to manipulate the array bit (32) in a special register word. The rules for direct and indexed special register addressing with this instruction represent exceptions to the rules stated above for these address types.

This instruction is identical to the unmasked binary add instruction, except when a direct or indexed special register address appears in the A, B, or C address group. If the special register is addressed as the source of an operand, the entire contents of the register are used, preceded by 24 high-order zeros. In addition, the special register sign bit (33) is placed in bit position 25, bits 26 through 32 are all set to zero, and the array bit (32) is placed in bit position 33. The bank indicator and subaddress remain in bit positions 34 through 48. If the special register is addressed as a result location, the low-order 24 bits of the accumulator contents are transferred to the special register. In addition, bits 25 through 31 are all set to zeros, bit 33 of the accumulator word is placed in bit position 32 (the array bit), and bit 25 of the accumulator word is placed in bit position 33 (the sign bit). Bits 34 through 48 of the accumulator word are stored as the bank indicator and subaddress in the special register. These relationships are shown in the following diagram.



Extended Binary Subtract (EBS) Instruction

This instruction is identical to the unmasked binary subtract instruction, except that when a direct or indexed special register address appears in the A, B, or C address group, the bits of the special register operand are altered exactly as described above under "Extended Binary Add."

Console Typeout of Control Memory

The octal format of a word typed out of or into a special register is as follows:

Special Register Word (Octal)	Sign	Memory Range
0000 0000 001X XXXX	+	0 to 32K
0000 0000 000X XXXX	-	0 to 32K
0000 0000 003X XXXX	+	32K to 65K
0000 0000 002X XXXX	-	32K to 65K

APPENDIX G

MEMORY BARRICADE

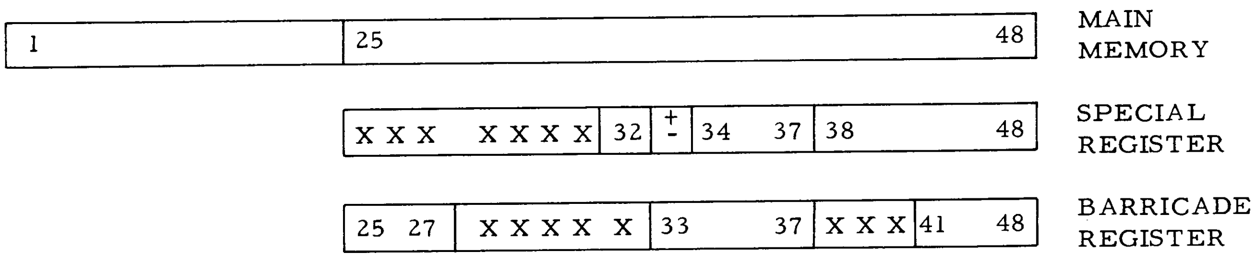
The memory barricade (feature 019) is an optional feature with the model 1801 central processor. It provides specific monitoring of all central processor operations to detect and protect against unwanted memory references and/or interference between programs, as defined by the contents of a barricade register. The execution times of instructions are not affected by the barricade.

Program Interference Protection

A variable area of memory can be protected against being altered by one or more programs designated as restricted, either directly or through peripheral operations which they initiate. Control memory registers are also protected against alteration in the same manner. Manual console operations are permitted in any area of main or control memory. The boundary location defined by the barricade register also functions as a stopper location, in addition to the system stopper location(s).

Barricade Register

The barricade register is a non-addressable storage register. Its relationship to main memory words and special register words is shown below.



Bit positions which are labeled with an X in the above diagram are disregarded and forced to zero. As shown in the diagram, 16 bits are used in the barricade register. The functions of these bits are as follows:

Bits 25-27	Violation Group Indicator. These bits indicate, in octal, the group number of the program which was most recently turned off due to a barricade violation or a limited control error (see below). These bits are printed as part of the barricade register contents during a manual or programmed typeout of this register. However, they are not altered when the register is loaded.
Bits 33-37	Boundary Indicator. These bits specify the high-order (array and bank indicator) bits of the memory location address at which the barricade is located. The low-order 11 bits of the address are

Bits 33-37 (cont)	assumed to be ones. The boundary may thus be assigned to location 2047 of any memory bank and is, in fact, the highest location included in the bounded area of memory. All locations above the boundary constitute the protected area of memory.
Bits 41-48	Group Designators. These bits correspond to control groups 0-7, respectively. A one in any of these bit positions indicates that the program running in the corresponding group is restricted to the bounded area of memory. A zero indicates that the corresponding program is permitted access to the protected area of memory.

Programmed access to the barricade register is by means of print instructions with specific configurations of the B address group. Within the B address group, the high-order bit must be zero; the values of the carriage-return, more-to-follow, and print-mode bits are irrelevant. If the low-order six bits of B have the octal value 04, the low-order 16 bits of the contents of the location specified by the A address group are transferred to the barricade register. Bits 38-40 of this word must be zero; otherwise a barricade failure control error will occur (see below). Bits 25-32 of the barricade register are not altered. If the low-order six bits of B have the octal value 10, the entire 24 bits of the barricade register are transferred to the low-order portion of the location specified by the A address group. The high-order 24 bits of the location specified by A are set to zero, as are bits 28-32 and 38-40.

With the above exceptions, print instructions used to reference the barricade register retain all of the properties of normal print instructions, including timing. In ARGUS notation, any type of print instruction (PRA, PRD, or PRO) may be written.

Program Compatibility

In a system which contains the memory barricade, reference to the barricade register is automatic. However, the barricade can be effectively "disabled" by setting the boundary indicator bits to all ones if it is desired to run existing programs without the barricade in effect. Alternatively, the boundary location can be disabled as a barricade but retained as a stopper location by setting all of the group designator bits to zero, so that any program can operate in any part of memory.

Memory Barricade Operation

The partitioning of memory by the barricade is illustrated schematically in Figure G-1. The protected area of memory consists of all locations above the boundary address. The bounded area consists of all locations below this address.

Any program running in a group whose designator bit in the barricade register is a one is not permitted to alter the contents of any protected memory location. It is, however, permitted to reference operands and instructions in any part of memory, including the protected area. Such a program is also permitted to reference any control memory register but is not permitted to alter the contents of any special register in a protected group. If a restricted program references a special register in a protected group, this register is not incremented; i.e., any increment stated in the address group is ignored.

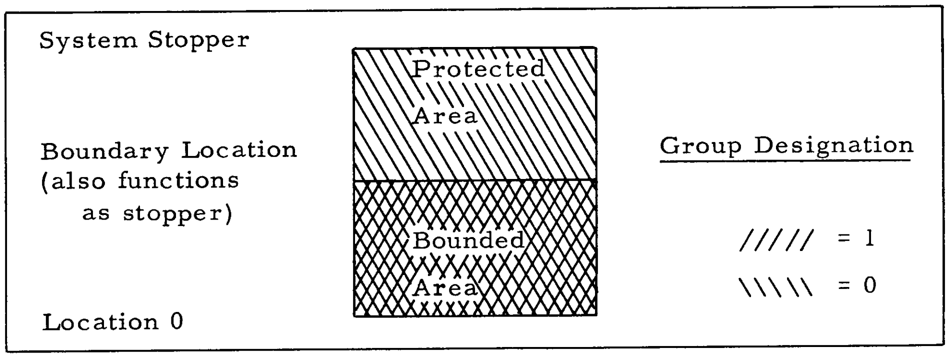


Figure G-1. Partitioning of Memory by the Barricade

The peripheral read-write counters RAC, DRAC, WAC, and DWAC are not subject to the above rules. Any of these registers can be referenced without restriction by any program, regardless of the contents of the barricade register. Moreover, there is no restriction on the use of the MPC instruction.

Any program running in a group whose designator bit in the barricade register is a zero is a protected program. Such a program is permitted to perform any normally legal operation, although it is subject to the stopper action of the boundary location.

Since, during peripheral buffer cycles, the responsible control group is not known, the protected area of memory is guarded against indirect alteration by a peripheral operation in the following manner:

1. During the execution of a peripheral read or write instruction, the starting address for the peripheral transfer is checked to see that it is in an area of memory which is legal for the program issuing the instruction.
2. The stopper action of the boundary location is in effect during all peripheral buffer cycles.

The contents of the barricade register (except bits 25-27) are continuously checked against a parity bit which is generated as the register is loaded. If this check fails, a critical control error (11000, see below) immediately stops the system. Violation of the barricade by a restricted program results in a limited control error (10000, see below). Such an error turns off the responsible program but does not stop the entire system.

Control Errors

In a system equipped with the memory barricade, the handling of control errors is modified. Such errors are divided into two classes, limited and critical, according to the probable cause and the effect of each error.

Limited control errors are those in which the probable cause and effect are or can be limited to the responsible program. In the event of a limited error, the responsible program is turned off as described below, and the appropriate indication is given. If the error occurs during a buffer cycle (e.g., from a non-existent memory address), a bus parity check is forced at the referenced peripheral control; the responsible program is not turned off, but memory protection remains in effect.

The following types of programming errors are detected, and their effects are limited to the responsible program:

1. Control check pulse (addressing a non-existent peripheral control).
2. Instruction word parity (data used as instructions).
3. Illegal operation code (data used as instructions).
4. Main memory addressing error (addressing a non-existent memory location).

Critical control errors are so basic to system operation that separation among programs cannot be maintained nor can the system be controlled effectively. In the event of a critical error, the system halts immediately with the appropriate check indication.

The following additions to Appendix D, "Control Errors," pertain to memory barricade operation:

Bit Position 4 5 6 7 8	Type of Error
1 0 0 0 0	A violation of the barricade
1 1 0 0 0	A failure in the barricade register

Figure G-2 is a list of all control error indications (in octal notation), indicating which are limited and which are critical errors.

System Response to Limited Control Errors

In general, when a limited control error occurs, the current instruction is completed normally, except that illegal alteration of main or control memory and escalation of errors are not allowed to happen. The responsible program is turned off, its group indicator is stored in bits 25-27 of the barricade register, and the appropriate control error indication is set up. In the following paragraphs, whenever an instruction is said to be "converted" to a proceed instruction, the proceed instruction exists only in the control register; the original instruction is not altered in memory.

Error Class	Error Indication (octal)	Error Class	Error Indication (octal)
Limited	01	Limited	16
Critical	02	Limited	20
Critical	03	Limited	21
Limited	04	Limited	22
Limited	05	Limited	24
Critical	06	Critical	25
Critical	07	Limited	26
Limited	10	Limited	27
Limited	11	Critical	30
Limited	12	Critical	31
Limited	14	(See Below)	32

Note: Error type 32_8 is critical if a failure in the address selectors is indicated, limited if a non-existent memory address is selected.

Figure G-2. Limited and Critical Control Errors

When a control error occurs during instruction extraction (types 01_8 and 12_8), the instruction is converted to and executed as a proceed instruction (with hunt). The group in control is then turned off with the appropriate check indication. Thus, if the error instruction is stored at location E, the sequencing counter stops on E + 1, and the instruction at E is not performed.

The A, B, and C address groups are checked independently for barricade violations (error type 20_8). (Note that a reference in a restricted program to a protected location or register does not constitute a violation, provided that the contents of the location or register are not altered. If a protected special register is referenced in a restricted program, any increment specified in the instruction address group is ignored.) Hunting is inhibited on an instruction causing a barricade violation: the same group selects the next instruction, which is converted to and executed as a proceed instruction (with hunt). This group is then turned off with the appropriate check indication. Assuming that the error instruction is stored at location E, one of the following three situations will result:

1. If the error instruction does not cause a sequence change or unprogrammed transfer, the sequencing counter stops on E + 2. E + 1 is not performed, but E is performed (without permitting illegal alteration of protected locations or registers).
2. If the error instruction causes a sequence change to location S, the sequencing counter stops on S + 1. S is not performed, but E is performed (without permitting illegal alteration of protected locations or registers).
3. If the error instruction causes an unprogrammed transfer to U ± n, the sequencing counter stops on E + 1. U ± n is not performed, but E is performed (without permitting illegal alteration of protected locations or registers).

Limited control error types 10, 11, 14, 21, 22, and 32 (octal) are handled in the same manner. If a barricade violation occurs during a peripheral instruction, the control check pulse signal normally sent to the peripheral device to initiate motion is inhibited. When the program is turned off as described above, the peripheral read-write counters will have been set up, but the peripheral buffer cycles will not have occurred. A control check pulse error produces the same result, except that the check indication is 05_8.

Control errors occurring in the accumulator result or in the floating-point unit (octal types 04, 16, 24, 26, and 27) are not detected until another instruction has been selected, possibly by a different control group. The system determines whether or not the error instruction was followed by a hunt for another active control group and proceeds accordingly.

If the error instruction was not followed by a hunt, then the group presently in control is responsible. The system proceeds as though a barricade violation (see above) had occurred on the current instruction (i.e., the one following the error instruction). For example, if the error instruction is stored at location E and results in no sequence change or unprogrammed transfer, E is executed but its result is in error, E + 1 is executed, E + 2 is converted to and executed as a proceed instruction (with hunt), and the sequencing counter stops on E + 3.

If the error instruction was followed by a hunt, then the group previously in control is responsible and is turned off immediately. For example, if the error instruction is stored at location E and more than one group is active, E is executed but its result is in error, and the sequencing counter is stopped on E + 1.

INDEX

HONEYWELL
ELECTRONIC
DATA
PROCESSING
WELLESLEY HILLS,
MASSACHUSETTS 02181