June 22, 1959

INTRODUCTION

This is a set of notes that is intended to introduce the reader to the H-800 and simultaneously to computer programming. It is intended that these notes will be used side-by-side with the book by McCracken "Digital Computer Programming". Since McCracken (hence forth abbreviated Mc) emphasizes scientific and arithmetic computations, and we are more interested in data processing and data manipulation programming, our examples will have a different emphasis from those in Mc. Likewise, the nature of the Honeywell 800 requires that we depart from Mc's organization in several places, particularly following his Chapter 4. In general, however, we will make the effort to parallel Mc's references to the fictitious computer "TYDAC" with references to the Honeywell 800.

CHAPTER 1.

The attached figure is a general block diagram of the H-800 system.

An H-800 system generally includes punched card input and output, console typewriter and line printer output, and magnetic tape units for input, output, and intermediate storage. The memory of the H-800 consists of several banks, a bank consisting of 2048 words or memory locations. For the time being, we will not consider more than one bank of memory. A word may be made up of an 11 decimal digit number with a sign, 8 alphabetic (or numeric or punctuation symbol) characters, or 48 bits (binary digits) of binary coded information. A word may be either an instruction word or a data word. In addition to the above mentioned memory (frequently denoted by HSM, for High Speed Memory), the H-800 system includes 256 special registers which are used for control purposes. The special registers are divided into 8 groups, each containing 32 registers. We shall only consider one group of 32 special registers. These special registers perform such functions as sequencing counters, indexing and indirect addressing registers, and other specialized control functions.

In addition to these MSM and special register storage locations, the Arithmetic-Control Unit of the H-800 contains certain highly specialized registers that are capable of performing arithmetic and logical functions. These include an Accumulator, a Low-Order Product Register, and a Mask Register. Unlike the TYDAC or other one-address machines, the H-800 is a 3-address machine. For this reason, the arithmetic registers assume far less importance to a programmer than they do in a 1-address machine. (They operate in much the same fashion as they would in a 1-address machine, but their use is implicit rather than explicit).

Note to Section 1.4:

We will point out here that the present "state of the art" is such that "automatic programming" systems are an accepted fact. We will therefore in these notes use symbolic notation. In other words, our examples will be written not in terms of absolute machine language, but in terms of the more symbolic language which can be used by the programmer and which will automatically be translated to machine language by the machine itself. (This, incidentally, is the essence of "automatic", or machine-aided programming).

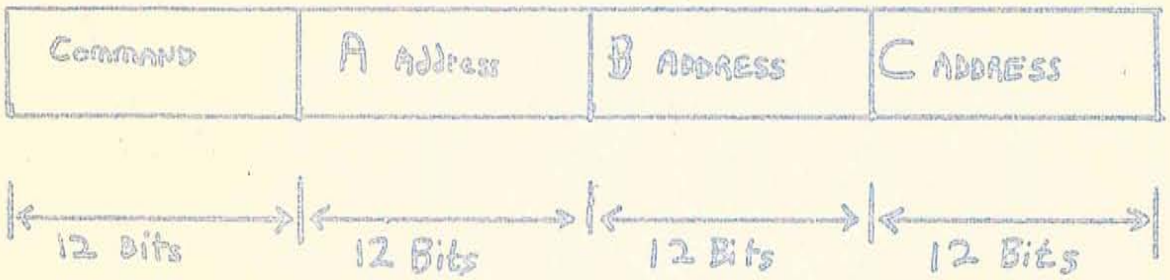
Similarly, the state of the art with respect to program checkout has advanced. There are presently available automatic aids to assist the programmer in "debugging", or detection of errors in programs which are in the process of being tested on a computer.

The ARGUS system for the H-800 incorporates all these features: automatic translation and memory assignment to allow the writing of programs in a convenient symbolic language, automatic program testing facilities, and automatic maintenance of files of programs, thus relieving the individual programmer of the responsibility of maintaining his own reel of tape or deck of cards; also increasing the efficiency of machine utilization by permitting automatic batch checkout of programs in a single program testing run.

CHAPTER 2. CODING FUNDAMENTALS AND INTRODUCTION TO H-800

It is suggested that the reader read Chapter 2 of Mc to become familiar with some of the basic ideas contained therein, remembering, however, that the structure of the H-800 differs considerably from the structure of the TYDAC, which is the specific machine to which Mc's examples refer.

The most significant difference between the TYDAC computer and the H-800 is that TYDAC has a single-address command structure, while the H-800 is a 3-address machine. Consider the illustration near the top of p. 15 of Mc. A similar illustration to describe the H-800 instruction word format is shown here:



Several points can be clarified in reference to this figure. First of all, the term "bit" is a contraction of the words "binary digit". Just as a decimal digit may assume one of the 10 values between 0 and 9, a binary digit may assume one of the 2 values 0 or 1. This subject will be covered in detail in Chapters 3. It is sufficient to note at this point that a word is divided into 4 parts or fields of equal size. The fields each contain 12 bits, and are known as the Command field, the A Address Field, the B Address field, and the C Address field. The Command portion of the instruction tells the machine what to do when it operates this instruction. The address fields are used to specify what memory locations are used as sources of operands or destinations of results in the course of executing the instruction. In any

one address field, it is possible to refer to any one of the 2048 words in one bank of RSM. In addition, other techniques may be used to code the address fields. This will be covered in more detail later.

Example 1 (p. 15) illustrates the TYDAC coding to perform the operation of adding two numbers together and delivering the result to a 3rd memory location. The following example illustrates the same operation in H-800 coding:

	COMMAND	AADDRESS	BADDRESS	CADDRESS
Operation:	DA	XYZ	ABC	RES

Contents of XYZ: before = + 1
ABC before = +92

} These registers are left unchanged by this operation.

Contents of RES before = - 123456789
after + 93

Note that, as in Mc's example 1, the previous contents of the result location (identified here by the symbol RES) are destroyed by the operation. This order has been written in symbolic form. The first field (command field) contains DA, which is the mnemonic operation code for the order, "Decimal Add the contents of A to the contents of B and send the result to C". The subsequent 3 fields are the A, B, and C address fields of the order. The symbols, or tags, "XYZ", "ABC", and "RES" are symbolic names assigned to particular memory locations. The numeric identification of these memory locations need not be of concern to the programmer. The addition operation is algebraic. Thus, if XYZ had contained a -1, the result would have been the algebraic sum of -1 and +92 or +91, even though the operation code was "ADD".

Example 2 illustrates a more elaborate computation (this does not correspond to Mc's Example 2).

Assume that we are in the midst of calculating an employee's payroll, and it is necessary to evaluate the following quantity:

U + V - W - D - E.

Where: U is Normal Gross pay
V is Overtime pay
W is Federal Income Tax With-holding
D is FICA (Social Security)
E is State With-Holding Tax

We wish to deliver the result of this computation to a memory location to which we have assigned the symbol NETPAY.

The following 4 H-800 orders will perform this computation:

COMMAND	A	B	C
DA	U	V	TEMP
DS	TEMP	W	TEMP
DS	TEMP	D	TEMP
DS	TEMP	E	NETPAY

Note that in addition to the symbols previously mentioned, a new symbol, TEMP, has been used. Following the first operation, the memory location TEMP contains the sum of U and V. Following the 2nd operation, a DECIMAL SUBTRACT order, the contents of TEMP have been diminished by the contents of W, and put back into TEMP. The 3rd takes away some more; the 4th operation does also, but delivers the result to NETPAY. In case anyone is interested, the location TEMP now contains the quantity U + V - W - D (which would be your net pay outside of Massachusetts). However, this location is now free and may be used as a temporary storage location for any other type of computation that it is desired to perform.

We shall, for the time being, omit discussion of multiplication and division and proceed to Example 6. (p. 25)

DETERMINING THE LARGER OF 2 NUMBERS.

The H-800 orders include 4 comparison orders. We shall consider here the 2 orders: NN (Inequality comparison, numeric) and LN (Less than or equal comparison, numeric). This example illustrates the use of the less-than or equal numeric comparison:

COMMAND	A	B	C
LN	NETPAY	X	Q

Where in memory location X we have the number 0.

This order operates as follows:

The contents of A (the word at the location specified in the A address field, also denoted by (A)) - in this case (NETPAY) - are compared with (X), or in this case, zero. If (A) is less than or equal to (B), a jump,

or sequence change will take place. If (A) is greater than (B), the next order will be taken in sequence. Thus, if we find ourselves executing the order at Q, we know that the poor fellow has managed to pile up more payroll deductions than he is able to handle. Likewise, the NM order will make a jump to the location specified in the C address field if and only if the 2 quantities (A) and (B) are not equal.

Section 2.5. It is appropriate to mention here the speed of the H-800 for comparison purposes.

To put our numbers on the same basis as those in Table 1, we must remember that the H-800 is a 3-address machine, and the machines tabulated are 1-address machines.

For instance, the "magnetic core, scientific, binary" machine of the right-hand column is capable of doing 40,000 single address orders per second. This corresponds to 16,667 3-address additions per second as illustrated in Mc's example 1. The H-800 is capable of doing 40,000 3-address additions per second. (Our example 1).

SUMMARY OF SELECTED H-800 OPERATION CODES:

DA	A	B	C	Decimal Add (A) to (B); send result to C.
DS	A	B	C	Decimal Subtract " " " "
NM	A	B	C	Inequality Numeric comparison of (A) and (B). Jump to C if Inequality condition is found.
LW	A	B	C	Less than or equal comparison of (A) and (B).
TX	A	-	C	Transfer (A) to C. (NOTE: B is ignored in this order)
TS	A	B	C	Transfer (A) to B and Jump (unconditionally) to C.
TM	A	n	C	Move n words from memory locations starting at A to memory locations starting at C.

EXAMPLES - CHAPTER 2.

1. It is desired to test the numbers at XA, XB, and XC. To the register denoted by ZCOUNT we want to deliver a count of the number of zero items. Assume that we have stored the following constants:

ZERO + 0

ONE + 1

The following coding will accomplish the desired result:

LOCATION	COMMAND	A	B	C	EXPLANATION
BEGIN	YX	ZERO	-	ZCOUNT	Initial setup.
	NN	XA	ZERO	Q2	Is the first one zero? If not,
	DA	ZCOUNT	ONE	ZCOUNT	yes - add 1 to ZCOUNT, jump to Q2
Q2	NN	XB	ZERO	Q3	Test 2nd number
	DA	ZCOUNT	ONE	ZCOUNT	
Q3	NN	XC	ZERO	Q4	Test 3rd number
	DA	ZCOUNT	ONE	ZCOUNT	
Q4	end.				

Note that we have written the orders in a format similar to that of the H-800 ARGUS coding form, and that we have in certain cases entered a symbol in the location field. It is not necessary to name or otherwise identify every order, so we have only named those orders which we refer to explicitly in our jump instructions, and the first order of the coding.

Example 2.

Evaluate the quantity 2x - y, where x is stored at 7FOD; y at 151NS. The result is to be delivered to 38LS.

One way to do this would be to multiply x by 2. However, a much faster way in this instance would be to add x to itself. Therefore, the following 2 orders will do the job:

DA	7FOD	7FOD	TEMP
DS	TEMP	151NS	38LS

Again, we have used a register for intermediate storage of the quantity 2x. We can avoid this by using this alternative scheme:

DA	7FOD	7FOD	38LS
DS	38LS	151NS	38LS

Example 3.

We have stored in memory certain information relating to the time it takes to get from here to Florida on two different airlines. We wish to program a selection of that airline which will get us there first. At location TWA, we have the arrival time for the first airline; in the following 3 registers there is other information such as flight number, etc. In 4 memory locations, the first of which is UNITED, we have similar information for the other airline. It is desired to move these words for the selected airline to 4 memory locations starting at FASTEST.

The following coding will do the job. Assume that upon completion of this routine, an unconditional jump to the order at GETSEAT is to be made.

```
        IN      UNITED      TWA     UFIRST
        TN      TWA         4       FASTEST
        TS      -           -       GETSEAT
UFIRST  TN      UNITED      4       FASTEST
        TS      -           -       GETSEAT
```

Note that the TN order allows us to move the 4 words with just one order. (The order can be used to move from 1 to 63 words at a time).

Note also the hyphens in the A and B fields of the TS order. Since all we wish to do is make a jump, we are not interested in transferring one word. Therefore, the hyphen indicates that the address is an inactive address, and that we wish no processing to take place on its account. Certain H-800 orders take on special significance when some of their addresses are inactive; these examples will be covered later. This use of the TS order is a particularly simple one.

CHAPTER 3 - Number Systems

The H-800 includes automatic coded decimal arithmetic operations. It can, however, also operate inthe binary number system, and can process individual bits of a machine word. It is therefore desirable that H-800 programmers become familiar with the fundamentals of the binary system.

Table 6, on Page 47, shows the Binary Coded Decimal system that is actually used in the H-800. Our figure, Table 6-a, illustrates the complete set of 4-bit codes known as hexadecimal notation. Notice that the first 10 entries here are identical with Table 6. The other 6 entries assign convenient (although somewhat arbitrary) names to the other 6 possible combinations of 4 bits. When it is desired to specify arbitrary binary quantities, but do not want to have to write out the individual 48 bits of a word. It is frequently convenient to use these hex codes for the 4 bit groupings. In the H-800, the programmer may use either hex or octal (3-bit groupings) notations.

At this time, it is appropriate that we discuss in somewhat more detail the composition of an H-800 word. As mentioned before, an H-800 word can consist of 11 decimal digits and a sign, 8 alphabetic characters, or 48 bits of binary data. We use a code which requires 4 bits per decimal position. 11 digits, therefore, require 44 bits of the word. The remaining 4 bits are used to specify the sign of the word. If the sign position is 0 (all 4 bits are zero), the word is treated as a negative quantity in arithmetic operations. Otherwise, the word is treated as positive. The result of an arithmetic operation will have a sign which is either 0 or C (all ones). It appears as if this sign rule is more complex than actually necessary, but the reason will become more evident later.

Figure 3-b shows the binary and hexadecimal representations of the operands in our old example 1, adding 1 to 92.

TABLE 6-a HEXADECIMAL NOTATION

8 4 2 1	
0 0 0 0	0
0 0 0 1	1
0 0 1 0	2
0 0 1 1	3
0 1 0 0	4
0 1 0 1	5
0 1 1 0	6
0 1 1 1	7
1 0 0 0	8
1 0 0 1	9
1 0 1 0	B (10)
1 0 1 1	C (11)
1 1 0 0	D (12)
1 1 0 1	E (13)
1 1 1 0	F (14)
1 1 1 1	G (15)

FIGURE 3-b

(XYZ):	1111 0000 0000	0000 0000 0000	0000 0000 0000	0000 0000 0001
	G 0 0	0 0 0	0 0 0	0 0 1
(ABC):	1111 0000 0000	0000 0000 0000	0000 0000 0000	0000 1001 0010
	G 0 0	0 0 0	0 0 0	0 9 2

The H-800, as mentioned above, is able to operate in either the binary or the decimal system. The following example illustrates a binary addition example, using the operation BA (Binary Add), and hex notation for binary numbers.

BA

Where: (ALPHA):	G 0 0 0 0 0 0 0 0 0 0 1 G	(Binary representation of the number 31)
(BETA) :	G 0 0 0 0 0 0 0 0 0 0 0 7	(Binary representation of 7)

The result will be:

G 0 0 0 0 0 0 0 0 0 0 2 6 (Binary representation of 38 - verify this)

So far, we have seen examples illustrating numeric and binary H-800 words. It was mentioned above that a word can also be used to store 8 alphanumeric characters. This is done by using 6-bit groupings within the the word, each group or code representing one of the characters in the word. The following code is used:

TABLE 6-C

FIRST 2 BITS:	00	01	10	11
Next 4 bits				
0001	1	A	J	/
0010	2	B	K	S
0011	3	C	L	T
0100	4	D	M	U
0101	5	E	N	V
0110	6	F	O	W
0111	7	G	P	X
1000	8	H	Q	Y
1001	9	I	R	Z

In addition, other codes are used for punctuation marks, etc. For instance, the code for "space" is 00 1101.

As an example, consider the 8 letter word SUNSHINE.

It is represented in the binary code as follows:

11 0010 11 0100 10 0101 11 0010 01 1000 01 1001 10 0101 01 0101

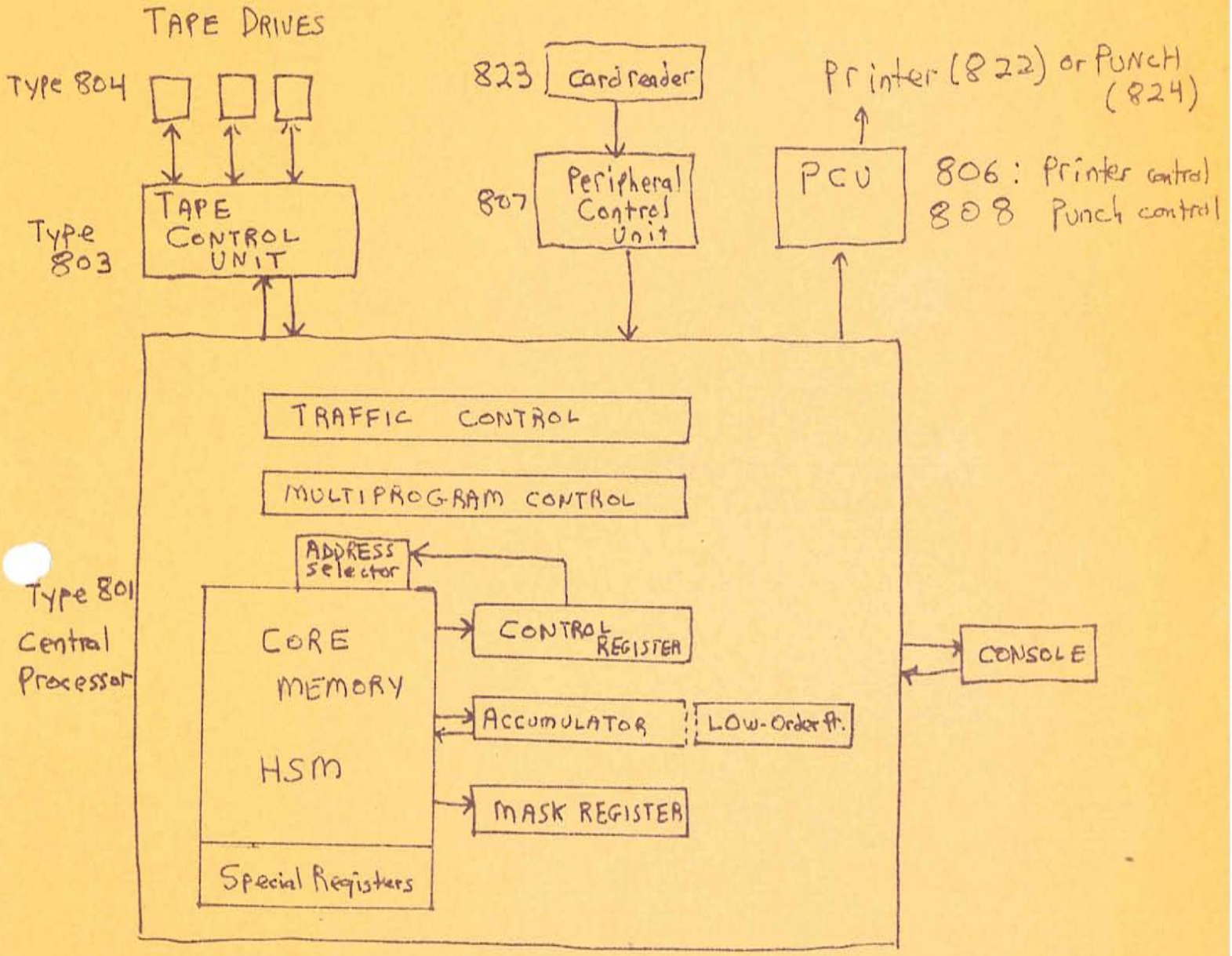
Note that the above configuration of bits is also the representation of the hexadecimal word:

C
D2A 972 619 955

There is no distinction made inside the machine between any of the various types of words. Of course, when a decimal add order refers to a particular word, it assumes that it is working with a decimal number. When a word is to be printed in 6 bit alphabetic form, it is treated as 8 6-bit characters. If a mistake were

made, and a decimal number were used instead, 8 characters would still be printed, and would probably look like gibberish. It is important to realize, therefore, that the distinction between the various types of words is a programmed distinction, and that the programmer or systems analyst would have control over this. Thus, an item may consist of mixed alphabetic, numeric, and binary coded information. An example would be a payroll record, where alphabetic words would be used for name, numeric words for pay rate and hours worked, and certain coded information stored in the individual bits of a control word. (For example, one bit might be 0 for male, and 1 for female).

TYPICAL H 800 SYSTEM DIAGRAM



NOTES:

1. CORE MEMORY IS EXPANDABLE (Type 802)
2. UP TO 8 TAPES MAY BE CONNECTED TO ONE TCU
   UP TO 8 TCUs or PCUs MAY BE TIED IN.
3. OTHER TYPES OF PERIPHERAL DEVISES ARE ALSO AVAILABLE.

~~4~~

6/23/59

CHAPTER 4.

Mc, Chapter 4, is on decimal point positioning problems. It is suggested that this chapter be read; we will not, however, specifically cover the problem in these notes.

The H-800 includes several operations of the type commonly known as logical operations. These operations are not arithmetic or simple transfer orders, but rather operate on the individual characters (or bits, if desired) of an H-800 word.

The first of these orders is the extract order,

EX A B C

The effect of this order is to generate a word based on the contents of A and B, and deliver this word to C. The rule for forming the result is to make every bit zero except where both (A) - (a notation for "contents of A") and (B) have bits which are one.

EXAMPLE 1: EX BB BC BD

Where:

(BB) = 1111 0000 0000 0000 0000 0000 0000 0000 0000 0001 0010 0101
+ 00 000 000 125

(BC) = 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 0000
or: GGG GGG GGG GGO

Then BD will be:

1111 0000 0000 0000 0000 0000 0000 0000 0000 0001 0010 0000
or: + 00 000 000 120

The word at BC, in this case, is an extractor control word that allows everything to get through except the low order character. Note the application of this to rounding a number.

(AE) = + 00 000 000 005

Then the following orders will round the word at QRT, discarding the low order digit - assume that (QRT) is always positive.

DA	QRT	AE	TEMP
EX	TEMP	BC	RESULT

(Chapter 4 continued)

The two comparison orders, NN and IN have already been covered. The H-800 has 2 additional comparisons. NA (compare for inequality alphabetic) and LA (less than or equal compare, alphabetic). The term alphabetic is somewhat of a misnomer here, for, while these are the orders to be used when testing alphabetic data, it is more appropriate to describe them as word comparisons. The NA order will result in a jump if any of the 48 bits of the 2 operands are different. Note that, if this order is used, it is possible to distinguish between -0 and +0. The LA order considers the 2 operands as 48 bit binary numbers. If (A) is less than or equal to (B), in this sense, the jump will take place.

Note the following example:

EXAMPLE 2:

(MIN4) : -00 000 000 004
(MIN8) : -00 000 000 008

The comparison IN MIN4 MIN8 LESS

will fail (not jump) since -4 is numerically greater than -8. However, if we remember that the 4 bit sign for a minus sign is 0000, and examine the two words, we see that the comparison

LA MIN4 MIN8 LESS

will succeed, since 000 000 000 004 is smaller than 000 000 000 008 (hexadecimal representation of the 2 binary numbers). Beware: it is generally a safe practice to use alphabetic comparisons unless it is known that signed numeric quantities are involved.

EXAMPLE 3:

It is desired to jump to JKL if the low order character of the word at GROAN is Z. The word at GROAN is an alphabetic word.

(TESTLO6) : 000 000 000 03G (hex)

Note that this word has ones in the low order 6 bits, and 0's elsewhere.

(LOZ) : 00 00 00 0Z (alphabetic); zeros except for the last character

```
      EK      GROAN     TESTLO6     TEMP
HERE  NA      TEMP      LOZ         HERE+2
      TS      -         -           JKL
```

The substitute order (SS A B C) is more complicated, in that it will deliver to C a word that may depend upon the previous contents of C. The algorithm for this order can be stated as follows:

> Wherever there is a 1-bit in (B) replace that bit position of C by the contents of the corresponding bit position of A.
>
> Wherever there is a 0 bit in (B), do not disturb the bit of C.

EXAMPLE 4: SS ASDFG HJKL CEE

```
(ASDFG) : 0000 0000 0000  0000 0000 0000  0000 1111 1111  1111 1111 1111
 HJKL)  : 0000 0000 0000  0000 0000 1111  1111 0000 0000  0000 1111 1111

before:
(CEE)   : 0000 0000 1111  0000 0000 0000  1111 0000 0000  1111 0000 1111

after:
(CEE)   : 0000 0000 1111  0000 0000 0000  0000 0000 0000  1111 1111 1111
```

Note that the contents of C are protected (i.e. left alone) where (B) has zero bits, while in the extract order, the write into C is destructive; i.e. those bit positions are not protected.

Two more orders will be briefly mentioned:

Superimpose (SM A B C).

Generate a word that consists of one-bits wherever either (A) or (B) or both has a 1 bit; where both (A) and (B) have zero bits, the generated word has a zero bit. Deliver this word to C.

Half-add (HA A B C).

Generate a word that has one-bits in those positions where the bit of (A) differs from the bit of (B); zeros where (A) is the same as (B). Deliver this word to C.

One application of this order, which is also known as Add without Carry, or symmetric difference, is to change every bit of a word (this is known as complementing the word). This can be done by half-adding the word to a word of all ones. The verification of this is left as an exercise to the reader. DO THIS?

Note that all the above orders except SS are commutative with regard to A and B.

CHAPTER 5 SHIFT ORDERS:

The H-800 has one type of shift order; shift one word right and around the end. There are 4 variants of this order. By end-around shift is meant the following:

Consider the following word -

012 345 678 9BC

If we shift this word right, end around by 4 bit positions, we get:

C01 234 567 89B

If we shift the original word right, end around, 44 bit positions, we get:

123 456 789 BC0

Note that this is identical to what we would get if we were able to shift the word left 4 bits.

In general, the information that comes around the end is of no great value to the programmer. In order to allow him to discard it easily, the shift orders incorporate "mask control". A mask address is specified as part of the shift order. The operation of the order is first to perform the indicated shifting, and then to deliver the shifted word to C under mask control. This may be of 2 varieties; protective masking, or extracting masking.

The other variants on the shift order are accounted for by the fact that it is possible to exclude the sign position of the word from shifting, if desired.

The 4 operation codes are:

SWS	Shift Word and Substitute into C
SPS	Shift, Preserving sign, and Substitute
SWE	Shift Word, Extract to C
SPE	Shift Preserving sign, Extracting to C

The format of the order as written by the programmer is:

Sxx, M A n C

(Chapter 5 continued)

M is a symbol that refers to a mask. It is set off from the operation code by a comma, but written in the command field. This is one instance of the symbolic language varying slightly from the machine structure of the word, for the mask information is actually partially specified in the B field of the machine word. n indicates the number of bits of shifting that is desired. It may take on any value between zero and 48, inclusive. If preferred, it may be written as A, r (for alphabetic characters) or D, s (for decimal numbers). In this case, r is the number of 6-bit character positions, and s the number of 4-bit characters to be shifted. The following 3 orders are identical in their operation and in their machine form:

```
SWE,GGH     X        12      Y
SWE,GGH     X        A,2     Y
SWE,GGH     X        D,3     Y
```

EXAMPLE 1:

Evaluate 0.1 SYA - SYB; deliver the result to SYN

```
SPE, XA  SYA   D,1   TEMP
DS       TEMP  SYB   SYN
```

Where the SHIFT MASK, XA is:

```
S,XA              GOG   GGG   GGG   GGG
```

Note that this mask has a "hole" in the position where the digit we threw away in the shift comes up in the word as shifted end around.

EXAMPLE 2:

Evaluate 10 SYB - SYC; deliver result to SYP

```
SPE,XB   SYB   D,10  TEMP
DS       TEMP  SYC   SYP
```

```
S,XB              GGG  GGG  GGG  GGO
```

- what happens to the high order 2 pos. which is zero. Temp. -

Note that the numeric shift shifts 44 bits of the word, leaving the 4 bits in the sign position undisturbed. Thus, a shift of 10 numeric places right brings everything around one digit position to the left of where it was originally.

The other 2 instructions, shift word, operate in the same fashion except that they treat all 48 bit positions of the word alike.

EXAMPLE 3.

At NUM is the word 000 $0d_1d_2$ $d_3d_4d_5$ $d_6d_7d_8$. It is desired to convert this word to its alphanumeric equivalent for printing and deliver the result to ALPHA.

```
EX          NUM     LOW4        ALPHA       d1
SWS,M1      NUM     46          ALPHA       d2
SWS,M2      NUM     44          ALPHA
SWS,M3      NUM     42          ALPHA
SWS,M4      NUM     40          ALPHA
SWS,M5      NUM     38          ALPHA
SWS,M6      NUM     36          ALPHA
SWS,M7      NUM     34          ALPHA
```

Where

```
LOW4  :000 000 000 000
S,M1  :000 000 000 3D0
S,M2  :000 000 00G 000
S,M3  :000 000 3D0 000
S,M4  :000 00G 000 000
S,M5  :000 3D0 000 000
S,M6  :00G 000 000 000
S,M7  :3D0 000 000 000
```

CHAPTER 6. INDIRECT ADDRESSING TECHNIQUES

Thus far we have seen the use of simple symbolic addresses to refer to specific fixed HSM locations. We have been able to handle simple examples, but we would not be able to add up 100 numbers without writing down 100 add orders, which rapidly leads to writer's cramp, and becomes impractical if the quota is 1500 numbers instead of 100. Actually, this process is one that is quite easily adapted to computation, by means of a loop.

Consider the following abstract statement of the job:

1. Set i to zero.
2. Set SIGMA to zero.
3. SIGMA + N_i into SIGMA
4. i+1 into i.
5. If i is less than or equal to 99, go back to 3.

Clearly, this procedure will suffice to add up the 100 numbers $N_{00} - N_{99}$. All that remains is to indicate how we can actually do this on the H-800.

The block diagram of the H-800 with Chapter 1 indicated that the memory included both HSM and some "Special Registers". It is these special registers which can be used for indirect addressing, in addition to being used for certain control functions. (For instance, the sequence counter is actually nothing more than one of the special registers). These special registers may be addressed directly, just as HSM memory locations. In addition, they may be used for indirect addressing; meaning that the actual HSM location involved is specified by the contents of the designated special register. It is useful to define a special type of constant for this use; this is the SPEC constant. Only the A field of this is used; in it is written the symbolic address desired to be used.

To refer to a special register explicitly, the code "Z", is used, while the code "N", is used to perform indirect addressing. The comma is followed by the designation of the special register, which may either be absolute numeric or literal.

There are 32 special registers that are easily accessible at any one time; they constitute a group. At present, we shall not consider more than one group.

We shall use the "general purpose" special registers R0 - R7 (16 - 24).

EXAMPLE 1. The 2 orders:

```
TX      ADCON     -      Z,R0
DA      N,R0      TAX    DEDNS
```

Where

```
ADCON:      SPEC      FICA
```

Have the same effect as the single order:

```
DA      FICA      TAX      DEDNS
```

EXAMPLE 2. We will now take a crack at adding up the 100 numbers that are stored in consecutive memory locations starting with the location whose symbolic address is N.

```
STEP1     TX      K1        -        COUNTR
STEP2     TX      K1        -        SIGMA
STEP3     WA      BASEAD    COUNTR   Z,R0
          DA      SIGMA     N,R0     SIGMA
STEP4     WA      CNTR      K2       CNTR
STEP5     LA      CNTR      K3       STEP3
```

Where:

```
K1          DEC     0
K2          DEC     000 000 000 001
K3          OCT     0000 0000 0000 0143          (binary 99)
BASEAD      SPEC    N
```

Note the use of the Word Add order on the unsigned quantities involved in the control computations.

This example can be improved upon; the location CNTR can be abolished by directly incrementing the contents of R0.

```
STEP1    TX    BASEAD    -      Z,RO
STEP2    TX    K1        -      SIGMA
STEP3    DA    SIGMA     N,RO   SIGMA
STEP4    WA    Z,RO      K2     Z,RO
STEP5    LA    Z,RO      ENDAD  STEP3
```

Where

```
ENDAD    SPEC    N+99
```

and we have gotten rid of K3.

We can still further improve on this example by taking advantage of another feature of special registers. This is the automatic incrementing feature. As a special register is used, it is possible to increment it for subsequent use. The size of the increment may be between zero and 31. Thus far, we have used zero increments, and have not needed to specify them. If an increment is desired, it is written, with a 2nd comma, following the special register address. Thus, N,RO,1 in an address field specifies indirect addressing with RO, incrementing it by 1. It is important to remember that the effective address (that is, the contents of the special register) is not incremented for the current reference. Instead, the incrementation takes place immediately after the use of the original contents of the special register.

Thus, the 2 orders

```
TX    BASEAD      -    Z,RO
EX    N,RO,29     B    C
```

will use the word at N as an operand in the extract order, and not the word at N+29. However, RO now contains:

```
SPEC    N+29
```

We can apply this principle to our example, and eliminate STEP4.

```
STEP1    TX    BASEAD    -      Z,RO
STEP2    TX    K1        -      SIGMA
STEP3    DA    SIGMA     N,RO,1 SIGMA
STEP5    LA    Z,RO      ENDAD  STEP3
```

It would be equally satisfactory to increment RO in the reference STEP5.

The one case where incrementing never takes place is when a word is delivered to a special register. For instance,

```
TX      WIGGLE    -     Z,RO,22
```

puts the contents of

```
WIGGLE    SPEC    GROSSPAY
```

and not GROSSPAY + 22 in RO. This applies only to direct addressing of special registers, of course.

There is also available in the H-800 another variety of indirect addressing. The 32 special registers in one group include 8 registers known as index registers (denoted by X0 - X7 or decimal addresses 8 - 15). These registers may, if desired, be used as indirect addressing registers (ex. N,X2,1).

They may also be used in a somewhat different fashion. Consider that the index register is capable of storing an address. It is possible, in indexed addressing, to refer to the word at the address specified, or to the word at the address resulting from the addition of the address in the index register to an augmenter stored in the order.

Indexed addressing is denoted, in ARGUS language, by writing the number of the index register, followed by a comma. The comma may be followed by the augmenter.

Examples: 3,
5,2

The operation of the augmenter is different from the operation of incrementers in special register addressing. In the first place, the augmenter is used in calculating the effective address. Also, the augmenter does not affect the contents of the index register.

We can illustrate the use of index registers by considering the example on page 8 of these notes. Assume that it is desired to indicate to the routine at GETSEAT which airline we have selected, and that the routine at GETSEAT is going to add the 4th word of the item for the selected item to a word called COST.

Then we could do this:

```
                LN      UNITED      TWA       UFIRST
                TS      ATWA        Z,X2      GETSEAT
UFIRST          TS      AUNIT       Z,X2      GETSEAT
ATWA            SPEC    TWA
AUNIT           SPEC    UNITED
GETSEAT   DA    COST    2,3         COST
```

Note that, in order to put a word into an index register, we must address it by special register address, but to use it for indexed addressing, we use the 2, type notation. Note also that if we want to refer to the 4th word

of the item, we must use an augmenter of 3, since it is the 3rd word following the first word of the item.

Unlike indirect addressing by special registers (N, type), indexed addressing may be used in any H-800 order. It is frequently a useful programming technique to use an index register for indexed addressing in part of a routine, and for special register indirect addressing elsewhere in the routine. Note the following scheme for processing 10 word items, where it is desired to go to NEWBATCH after every 12 items have been processed. The processing refers to the 2nd, 4th and 5th word of the items. The 12 items are stored in consecutive memory locations starting with BASEAD.

```
BEGIN     TX      K1      -       Z,X3
PROCESS    x      3,1
           x              3,4
           x      3,3
          LA      K2      Z,X3,10 NEWBATCH
          TS      -       -       PROCESS
```

Where:

```
K1        SPEC    BASEAD
K2        SPEC    BASEAD + 110
```

Note that, when we get to NEWBATCH, the number in X3 has overshot, since the incrementation by 10 takes place regardless of the output of the comparison. Note also that careful analysis is required in choosing a combination of comparison orders and comparison constants for termination of the loop. In this case, the following analysis applies:

> The last item starts at BASEAD + 110. Therefore, when we have finished processing it, X3 will still contain its address. The use of the LA comparison will fail as long as BASEAD + 110 (K2) is greater than the base address of the item just processed. When it succeeds, we will break out of the loop.

Note an alternative solution for the last 2 orders:

```
          NA      K2      Z,X3,10 PROCESS
          TS      -       -       NEWBATCH
```

Company Private

DATAmatic

DIVISION OF MINNEAPOLIS-HONEYWELL REGULATOR COMPANY

1123-658

Inter-office Memorandum

DATE June 8, 1959

TO Systems Analysis Department

C. C. TO:

FROM J. Thomas

SUBJECT:

Attached are rough notes intended to up-date the H-800 Introductory Lecture Notes.

ADDENDA, REVISIONS and RESTATEMENTS to an incomplete copy of the H-800 Introductory Lecture Notes. June 2, 1959

3-1-1

Multiply

DM α β ϕ γ

High order product to γ (and in acc.) UNROUNDED

LO product in LOP

to get it: ⟵A⟵

TS ⟵ A ⟵

CAN'T ADDRESS LOP
MUST USE TS - B -
TO STORE LOP IN B

⟵ : INACTIVE

3-1-6

Shift Orders: Example 6

SWS, mask A Fb C

Mask Index Register:	MASKBASE	64	0	1
SWS, 20	1200	12	1438	

---

p. 3-1-14

IX:

INDEXED INSTRUCTIONS

Example 17 (Bank 1)

Contents of:	X1	SPEC, B1	0183
	X2	SPEC, B1	0951
	X3	SPEC, B1	1033

BM	1, 0	2, 5	3, 11
	or		
	1,		

Multiplies (183) (951 ÷ 5) = (956) (1033 ÷ 11) = (1044)

Contents of X1, X2, X3 do NOT change.

COMPARISONS

Numeric comparisons should not be used on SPEC constants. The sign bit in these constants has special significance only in the circuitry of the special registers. It does not constitute a numeric sign. All words from special registers will be treated as negative, since zeros are always present in the high order 4 bits.

3-5-2

S/C bit is the HIGH ORDER bit of the word. Blank or S for sequence
C for cosequence

The example on page 3 - 5 - 4 (Figure 2) is WRONG. The instruction at location 190 will pick up the next order from CSC regardless of how the comparison comes out. Thus, the order at 583 (?) might be executed instead of 191 if A = B.

TRANSFER

3-4b-2

TX

Transfer contents of A address to location specified by C address; ignore B.

TS

Transfer contents of A address to location specified by B address. The C address portion will replace the contents of the sequencing counter unless the C address is left blank (what we) in which case the sequencing counter will not be changed.

Select Instruction

3-7-2

SSL, Mask A B C

The mask must be of the form

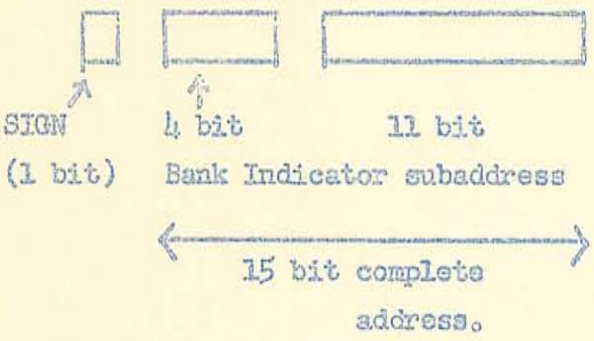
0000 ---------0XXX XXXX XXXX

High order 37 bits must = 0. Low order 11 bits = desired mask

---

Special Registers:

3-10-1



Example of coding indicating ARGUS Maskgroup process.

MASKGRP, 1

Loc.	Command	A	B	C
F, HALF	DEC		GGGGGG000000	
F, GEE	DEC		000000000000	
S, 3 NUMR	DEC		G000GGGGGGGG	
S, 3 NUML	DEC		GGGGGGGGG000	
S, SIX	DEC		000000000030	
Z, MIR	MASKBASE, 1			
	WA, GEE	X	Y	Z
	SSL, SIX	Q	0	R

The MASKBASE constant takes care of both base addresses for the MIR. The MASKS are automatically located in suitable memory spaces by virtue of the "F," or "S,", and the use of the mask tag in the command field guarantees that the proper subaddress will be provided.

History Registers:

3-10-3

We have a subroutine starting at Y which is entered from several places in the program. The bisequence mode cannot be used in this case.

To get to the subroutine:

June 8, 1959

X TS (⇒) (⇒) Y

Contains Sequence History SPEC X + 1

Y TX Z, SH - Temp

⇒ Process

⇒

⇒

⇒

⇒ end

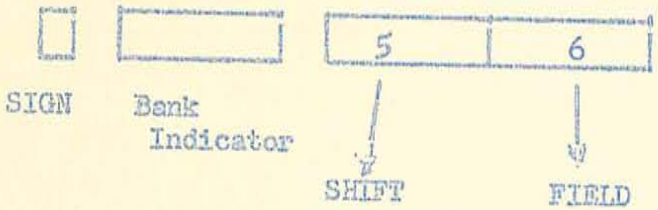
TX Temp - Z, SC

or

TS TEMP Z, AUA N, AUA

Mask Index Register:

3-10-7



This constant can be specified as follows:

MASKBASE X Y

X is the base address of type S masks.

Y is the base address of type F masks.

Use of Special Registers:

DIRECT ADDRESSING 3-10-13

Z, rr, ΔΔ

rr may be decimal $0 \leq rr \leq 31$
or literal (SC, X3, etc.)

ΔΔ may be decimal $0 \leq \Delta\Delta \leq 31$
or omitted.

Z, rr is sufficient.

Special register addresses are to be considered as follows:

ONE of 256 Special Registers.

---

256 = 8 x 32

8 groups; which one is determined by the Identification number of your Sequence Counter.

32 special registers in a group.

INDIRECT ADDRESSING: 3-10-17

Like direct, but N, rr, ΔΔ

The effective address is the full address (including bank indicator) which is stored in the designated special register.

Examples:

(60) = SPEC, B0 1000
(61) = SPEC, B0 1500
(1000)= +00 111 222 333

(1500)= +00 111 111 111

DA N, 60 N, 61 1200

puts in 1200 +00 222 333 444

INDEXED ADDRESSING:

3-10-20

is r, ———

——— may be:

(a) numeric (0 ≤ — ≤255)

blank ≡ 0

(b) symbolic = defined elsewhere by EQUALS.

Example:

GROSSPAY EQUALS 100

DA 3,GROSSPAY

is ≡ DA 3,100

SPECIAL REGISTER INDEXED

(c) $\left.\begin{matrix} N, \\ Z, \end{matrix}\right\}$ XX, Δ'

XX is a special register address.

(numeric $0 \leq XX \leq 31$)

or literal.

Δ' is an increment: $0 \leq \Delta' \leq 3$.

Index register must contain a suitable constant.

Example A

Our program is in Group 1.

(X4) = SPEC, G2 0

order: TX 4, N, SC, 0 --- Q

Transfers the next order about to be executed under sequence counter control in program 2 to location Q.

Example B

(X4) = SPEC, G2 N, SC

order: TX 4, Z, 0

same effect.

The method of the first example is preferable, since it allows easy reference to any register in group 2, and the addressing scheme is more straightforward. However, since the effective address is computed, as for HSM indexed operations, by adding the complete address from the index register and the 8 bit augmenter from the order word, either method is acceptable.

Note that 4, N, SC, 0 is translated as follows:

Designator bit	= 1	(because special register)
Indexed bit	= 1	(because indexed)
1 designation	= 100	(IR 4)
Augmenter	= 00100010	

This augmenter is identical to the low order 8 bits of an address we would get if we wrote N, SC .

Note also the reason that $\Delta' \leq 3$.

This is because the 8 bits available in the order word correspond to the
special register address (5 bits)
direct-indirect bit (1 bit)
incrementer (2 bits)

Thus the augmenter can only contribute to the 2 lowest bit positions of the incrementer portion of the effective address.

Defined, or SIMULATOR instructions.

Form: S, XXXX A B C

XXXX should previously have been defined by the Assembly Control code SIMULATE. It refers to an address which must end in octal 7.

or S, n, ----

n is an index register.

---- is an augmenter (symbolic or numeric)

It must end in octal 7.

The effective address need not end in octal 7.

ACTION:

1. The order itself is stored in the effective address of operation field.
2. The CSC is set to this address + 1.
3. The Address A is stored in AUA (Special register 0).

This address may be an effective address as a result of indexing.

4. The effective C address is stored in AUC. (special register 1)
5. The CSC is used to select the next order.

AUA and AUC can be used immediately for indirect addressing of 2 locations of importance to the simulator routine.

TYPOGRAPHICAL ERRORS:

P 3-1-15 Example 19

Shift order should be a 3 bit shift.

P 3-1-3 Example 2

The product is wrong, and should be shifted right 1 digit.

P 3-1-2 Figure 1.

The order should be DM, not BM

P 2-1-12

The add order should be BA, not DA.

II. GENERAL TRANSFER INSTRUCTIONS

A. One Word Transfers

The H-800 has two one-word transfer instructions:

(1) Transfer A to C, Ignore B (TX A B C)
This one-word transfer instruction causes the contents of the A address to be transferred to the location specified by the C address. The B address of the TX instruction is ignored and may be used to store data.

(2) Transfer A to B, go to C (TS A B C)
This one-word transfer causes the contents of the A address to be transferred to the location specified by the B address. The C address portion of the TS instruction will replace the contents of the sequenci counter*, unless the C address is left blank in which case the sequencing counter will not be changed.

(The A and B addresses can be left blank and the C address used to change the sequencing counter.)

*This means the sequencing counter specified for the next order. Thus, TS C A B Q will cause the cosequence counter to be set to Q, and the next order executed will be that at Q.

B. Multiple Word, Transfer Instructions

(1) Transfer N words (TN A B C)
This instruction causes the number of words specified by the B address to be transferred from successive memory locations starting with the location specified by the A address to successive locations starting with the one specified by the C address. The number of words transferred may be any number from 0 to 63.

(2) Item Transfer (IT A B C)
This instruction first substitutes the end of item symbol for the High Order 32 bits of B, protecting the Low-Order 16 bits of B. Secondly this instruction transfers from consecutive memory locations starting with A to consecutive locations starting with C until an end of item symbol has been transferred. The end of item symbol which terminates the operation need not be the one placed in B by the instruction. No address used in this instructinn should be special register direct.

(3) Record Transfer (RT A B C)
This instruction is used to transfer an entire record within the memory The instruction is identical to the Field Transfer instruction, except that an end of record word is written at B instead of an end of item symbol and an end of record word terminates the transfer. No address used in this instruction should be special register direct.

(4) Transfer A to C N times (MT A B C)
This causes the contents of the A address to be transferred to the location specified by the C address. This operation will be repeated as many times as specified by the contents in the B address, which must be less than 64.
The A and C addresses of the instruction are not modified during execution of the order, however, if special registers are specified, their contents may be incremented after each transfer.

TABLE OF CONTENTS

LECTURE NOTES

Introduction to Electronic Data Processing

COMPANY HISTORY

Datamatic was originally formed as a corporation in April 1955 through the joint efforts of Minneapolis-Honeywell Regulator Company and Raytheon Manufacturing Company. Its nucleus was a group of scientists and engineers who had been active in electronic computer research and development since the pioneering days of World War II. Prominent members of this staff were key personnel in the work done at the Harvard Computation Laboratory, at the Aberdeen Proving Grounds, and at other such early developmental areas.

In addition to this personnel, Datamatic enjoys a well-trained and experienced staff which was developed at Raytheon over many years. The staff designed and built the RAYDAC, a giant scientific computer currently in use by the Department of the Navy, and performed basic research and design projects for various governmental agencies.

Next came the DATAmatic 1000 which begins its career as RAYCOM, the business data processing counterpart of RAYDAC. Initial work, including drawing up specifications and logical design, was begun in 1952. The next few years were spent in the development and construction of the machine with special emphasis on the requirements of todays business applications.

In June 1957, Honeywell acquired Raytheon's interest in the separate corporation and Datamatic subsequently became a division of Minneapolis-Honeywell Regulator Company.

LECTURE NOTES

Introduction to Electronic Data Processing

COMPANY HISTORY

Datamatic was originally formed as a corporation in April 1955 through the joint efforts of Minneapolis-Honeywell Regulator Company and Raytheon Manufacturing Company. Its nucleus was a group of scientists and engineers who had been active in electronic computer research and development since the pioneering days of World War II. Prominent members of this staff were key personnel in the work done at the Harvard Computation Laboratory, at the Aberdeen Proving Grounds, and at other such early developmental areas.

In addition to this personnel, Datamatic enjoys a well-trained and experienced staff which was developed at Raytheon over many years. The staff designed and built the RAYDAC, a giant scientific computer currently in use by the Department of the Navy, and performed basic research and design projects for various governmental agencies.

Next came the DATAmatic 1000 which begins its career as RAYCOM, the business data processing counterpart of RAYDAC. Initial work, including drawing up specifications and logical design, was begun in 1952. The next few years were spent in the development and construction of the machine with special emphasis on the requirements of todays business applications.

In June 1957, Honeywell acquired Raytheon's interest in the separate corporation and Datamatic subsequently became a division of Minneapolis-Honeywell Regulator Company.

were to be performed, and the disposal of the result. Control of the sequence of computing processes was to be carried out through a set of punched cards. These plans, over 100 years old, are amazingly modern in concept.

Babbage died in 1871. Although the analytical engine had been designed in great detail on paper and Babbage had spent a considerable sum of money on it, only a small portion of the machine had been constructed. After his death, part of the mill was built by his son, H. P. Babbage and may now be seen in the Science Museum at South Kensington.

THE PUNCHED CARD

In 1886, the census of the people of the United States, which had been taken in the year 1880, was still being sorted and counted. Among the men then studying census problems was a statistician and inventor, Dr. Herman Hollerith, the chief of the Census' tabulation section. He saw that existing methods were so slow that the next census (1890) would not be finished before the following census (1900) would have to be begun. Adapting the principle of the punched-paper control system, similar to that used by Babbage, he applied the method to the accumulation of statistical data. This was a most important stride in mechanical computation, for it introduced into a working system the concept of mechanically stored information. He employed this idea for many calculations or tabulations without the necessity for re-entering the data from a keyboard. He experimented with sorting the counting, using punched holes in cards, and with electrical devices to detect the holes and count them. The Hollerith equipment was one of the ancestors of familiar punch-card machines. He devised a coding system which bears his name, the Hollerith code.

A short time thereafter, another member of the Census Bureau staff, James Powers developed another type of mechanical tabulator which also used punched cards. Initially the holes in the card used by both Powers and Hollerith

were round but in subsequent years the holes in the Hollerith card were made rectangular in shape. Both types of equipment were used by the Census Bureau for many years and, in fact, are still in use.

Punched cards were used later in machines built for accounting and statistical purposes. The idea of an automatic calculating device in connection with punched-card machines led to the development of the first large automatic digital computer, the Mark I. It was built for the U. S. Navy in 1944 by the Computation Laboratory at Harvard University. Although not a punched-card machine its input and output mechanisms are controlled by punched-cards. This machine uses mechanical counters driven through electro-magnetic clutches controlled by relay circuits. In principle and in its function, it is somewhat similar to Babbage's conception of the Analytical Engine. Although in construction, in its free use of electrical components, it is very different from the engine; in principle, in what it does, it is rather similar. Other machines, developed later, use relays not only for control but for the storage and arithmetical units of the machine themselves. The first computer to make use of electronic circuits, with the speed of operation of which they are capable, was the ENIAC, installed at the Aberdeen Proving Grounds in 1948. It was one thousand times as fast as the Mark I.

ORGANIZATION OF DATA PROCESSING SYSTEMS

Data Processing is the inevitable paper work involved in all phases of general business procedures. The basic elements which comprise record-keeping systems may be classified into several functions.

1. Reading and recording of source data
2. Input; introduction and extraction of data from the system
3. Processing data which includes sorting, classifying, computing and summarizing
4. Filing
5. Output

The four significant stages of advancement from the manual routines, through key-driven machine operation, to the punched card method and finally to electronic data-processing systems are illustrated in Figure 1. A purely manual system of data-processing would presumably use pencil and paper for transcribing and computing and a conventional file cabinet for filing. Manual methods are not only slow and costly, but are subject to errors especially if the work is routine or if the work load is heavy.

The second stage is one of mechanization where typewriters, desk calculators, accounting and bookkeeping machines are now standard office equipment. The unit cost of these devices is relatively low considering the accuracy and speed available over manual methods. The advent of the punched paper tape introduced compatibility to a wide range of equipment. Many of the standard office machines can now be used in combination with each other through the common-language medium of the punched paper tape.

In the next stage, punched card machines were introduced wherein the punched card often serves as the source document and as a data storage and communication medium. These methods are rapid, accurate and relatively low-cost for the volume of work which they are capable of producing. Limitations, which have been overcome by electronic data processing, are evident in the punched card system. For example, the scope of operation performed is limited so that exceptions must be handled manually. Furthermore, the record or item size is restricted to the number of columns punched on a single card.

In the electronic data-processing system, many of the basic functions of record-keeping are accomplished at electronic speeds. Source data may be transcribed onto paper tape, magnetic tape or punched cards for conversion into binary code, a language which can be understood by the central processor. The use of magnetic tape as a storage medium for files permits the storage of tremendous volumes of data in a small space, with rapid access to the information in the files. Manipulation of data within the central processing unit, such as sorting, classification, computation, etc., is performed automatically as directed by a series of instructions called a program. The output from the

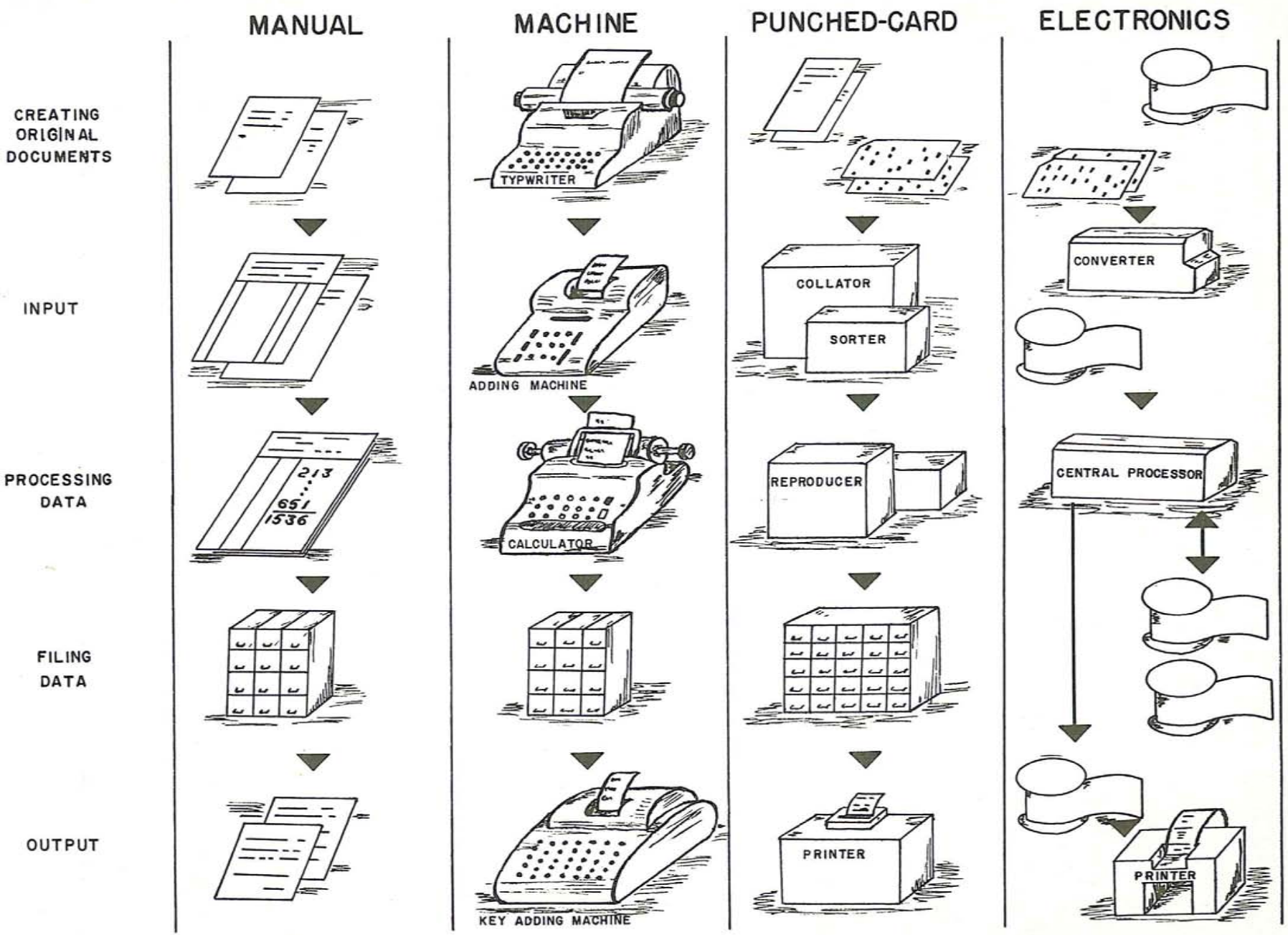


Figure 1. Organization of Data Processing Systems

central processor is written upon magnetic tape and may be transcribed to paper tape, punched cards, or fed into high-speed printers. In addition to the high degree of effectiveness in inventory and production control and sales analysis, more timely information for effective management control is attained through the increased speeds of which electronic computers are capable. Furthermore, the electronic processor introduces a major increase in reliability and freedom from error.

PRESENT DAY DATA PROCESSING METHODS

The standard punch card in use today consists of two main areas, the upper zone section and the lower numerical section (see Figure 2). This latter portion is subdivided into ten horizontal rows, one for each digit, zero through nine. The zone section is subdivided into three horizontal rows, the upper row being called the R or 12-zone, the second row the X or 11-zone and the 0-zone which is the zero row in the numerical section.

The card is composed of 80 vertical columns in which holes may be punched to represent all the digits (0-9), the alphabetic characters (A-Z) and several special symbols. The letters and symbols are characterized by a punch in each of the numerical and zone sections. These punch configurations are known as the Hollerith code.

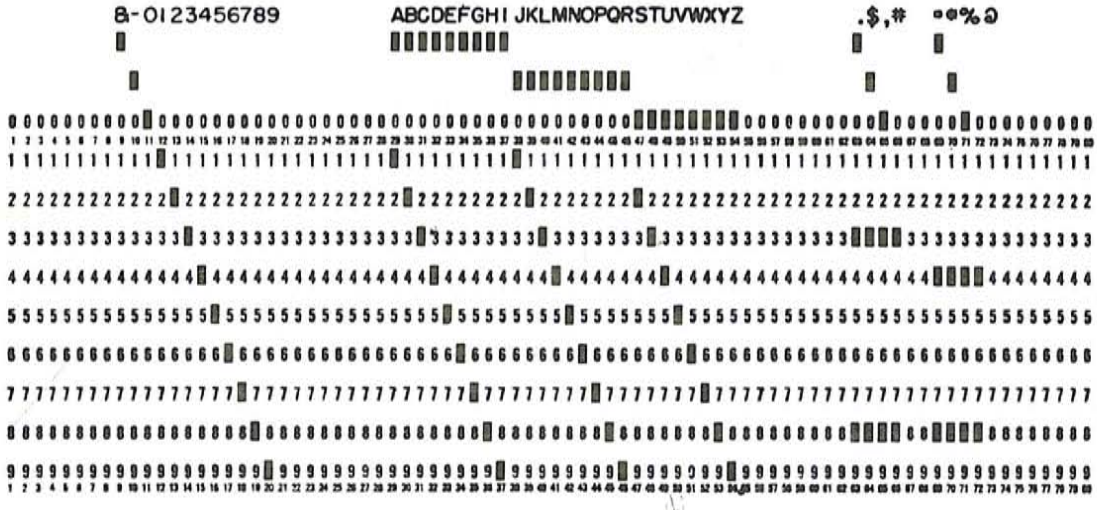


Figure 2. Punched Card Showing Card Codes

In order for the data processing machine to operate upon the data contained in the punched card, the code on the card must be translated into a code which is recognized by the machine. The punch configurations which occupy 12 levels or channels on a Hollerith card may be reduced to six levels for use with the machine. Since 64 different configurations of punches or no punches are possible in a six level code, there is an ample representation for all digits, letters and special symbols.

Instead of a punch or no punch on a card, an electronic principle may be employed wherein a transistor may be either conducting or not, or a magnetic core may be magnetized in one state or another. Through the use of six transistors or six magnetic cores, any one digit, character or special symbol may easily be represented. After a brief explanation of the evolution of numbering systems, the manner in which the binary numbering system is applicable to electronic computers through the use of transistors or magnetic cores will be shown.

NUMBERING SYSTEMS

The evolution of numbering systems from their primordial origin to their present phase of development has been marked by several important discoveries. Chief among these is the concept of positional notation. Although the principle is a familiar one, the title may not be so well-known. For example, the number four hundred forty nine might be expressed in several ways. It could be written as 449 or 400 + 40 + 9 or 4 (100) + 4 (10) + 9 (1). It should be noted that in reading the decimal number from right to left, that the decimal multipliers are 1, 10, 100, etc. (successive powers of 10). Since 100 is equal to 10 x 10, it can be represented as 10^2. Similarly, 1000 equals 10 x 10 x 10 or 10^3. In general, the product of n tens can be expressed as 10^n. In particular, 10 can be written 10^1 and 1 as 10^0. This is called the power notation. Therefore using the power notation, 449 can be written as $4 \times 10^2 + 4 \times 10^1 \times 9 \times 10^0$.

Positional notation is a name given to indicate that a digit in one location of a multi-digit number carries more or less weight than all other digits of the number. For example, the two 4's of the number 449 are unequal because of their position in the number. The leftmost 4 is said to occupy the most significant location. It represents the quantity 400 or, more generally expressed, 4×10^2. This 4 occupies the hundreds digit position. The next 4 occupies the tens digit position and, because of its position, depicts the quantity 4×10^1. The 9, in the units digit position, occupies the least significant location and represents the quantity 9×10^0. This method of numbering is, of course, the decimal system.

In this system, only ten states are used in any one column (from 0 to 9). Any attempt to add one to the maximum quantity (9) in any one column results in a return to zero in that column and a carry of one to be added to the next higher column. The decimal system is said to have a base of 10 or a radix of 10. It should be noted that the highest quantity that can be represented by a single digit is the radix minus one (10 - 1 = 9). However, the base used in a numbering system need not be limited to a radix of 10. The numbering system which is widely employed in the digital computation field is called the binary system. Here the base is 2, implying that the highest value which can be used in any position is one. The binary system is particularly suited for use with digital computers since a controllable electrical signal can be varied readily between two states (on or off, plus or minus, etc.) Either of these two conditions may be defined as the "one" state, and the other as the "zero" state.

Arithmetic in the binary system is much simpler than in the decimal system, a factor of considerable importance in connection with the logical design of a computer. The significance of a binary digit or "bit" (contraction of binary digit) in a number depends upon positional notation. For example, 11001 in the binary number system converts to $(1 \times 2^4) + (1 \times 2^3) + (0 \times 2^2) + (0 \times 2^1) + (1 \times 2^0)$ or to 25 in the decimal system. It should be noted that in reading the binary number 11001 from right to left that the decimal multipliers (or weights) are 1, 2, 4, 8, 16, etc. (successive powers of 2). The first seventeen numbers from 0 to 16 in the decimal notation correspond with the numbers in binary notation as shown in Table 1.

Table 1. Decimal-to-Binary Conversion

DECIMAL	BINARY	DECIMAL	BINARY
0	0	8	1000
1	1	9	1001
2	10	10	1010
3	11	11	1011
4	100	12	1100
5	101	13	1101
6	110	14	1110
7	111	15	1111
		16	10000

A modification of the pure binary system, known as the binary-coded decimal system, makes it possible to easily convert numerical decimal information into a kind of binary equivalent (4-bit code) which the machine recognizes. Conversely, numerical information expressed in the binary-coded decimal notation can be easily and quickly converted into the convenient equivalent decimal form for easy reading.

Table 2 shows the result of conversion from the decimal numbers 0 through 9 to the 4-bit code. Thus, the decimal number 95 is 1001 0101 in the binary-coded system. Similarly, the binary-coded decimal number 0011 0000 0110 0010 is 3062 in the decimal system.

Table 2. The Four-Bit Binary Code

DECIMAL	BINARY	DECIMAL	BINARY
0	0000	5	0101
1	0001	6	0110
2	0010	7	0111
3	0011	8	1000
4	0100	9	1001

To adapt the code on punched cards to a form acceptable to the electronic computer, an interpreter or translator unit is incorporated. Basically, this operation translates from one language to another. For example, the translator unit accepts punched holes in a card and converts this information to electrical impulses for machine use. The illustration, Figure 3, depicts a method of transforming the numeral five from the language of the punched card to that of the binary system through the use of four transistor flip-flop circuits. Two axioms must be considered in the explanation: (1) a single pulse alters the state of the transistor, i.e., conducting or not conducting; (2) a conducting transistor will generate a pulse when cut off. Five pulses corresponding to the numeral five on the punched card are fed sequentially to the least significant or rightmost flip-flop. The first pulse causes transistor No. 1 to conduct as shown in b of the diagram. This transistor is then cut off by the application of the second pulse but in the process generates its own signal which energizes flip-flop No. 2. By the same line of reasoning, it can be seen that after the fifth pulse, transistors No. 3 and No. 1 are conducting and represent the digit 5 in the binary-coded system.

In addition to the transistor storage method, metallic magnetic cores are also used to provide a memory device which is reliable, simple and compact. A core can be magnetized in either of two states. Thus, current passing through a core in one direction magnetizes the core in the "1" state while in the opposite direction, the current magnetizes the core in the "0" state.

TRANSISTOR FLIP-FLOPS

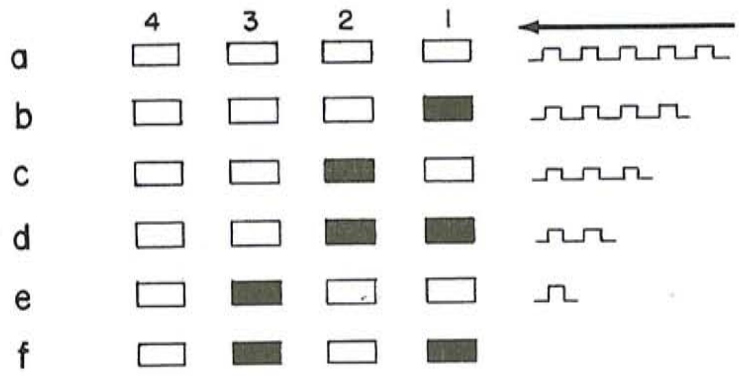


Figure 3. Electronic Counter Configurations

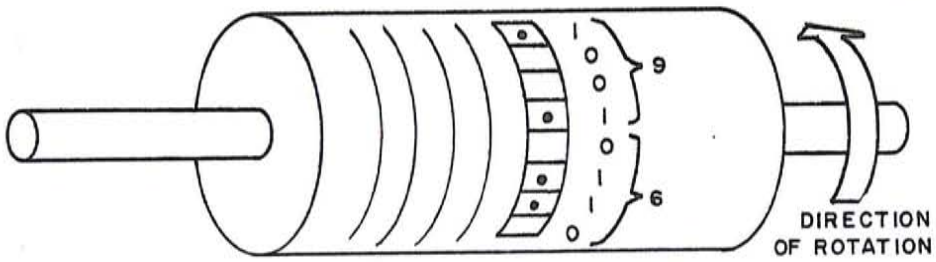


Figure 4. Magnetic Drum

HIGH-SPEED MEMORY DEVICES

The next consideration in the design of an ideal large-scale computer is the necessity for a simple reliable form of high-speed storage, with a large capacity. Historically, several forms of storage have been used. For example, they are acoustic delay lines, in which data are stored in the form of trains of sound waves, insulating screens in which they are stored in the form of a static charge distribution and magnetic material in the form of wire, tape or the surface of a drum, on which data are stored in the form of variations in the magnetic state of the material. Acoustic delay lines and electrostatic storage have fallen into disuse for engineering reasons.

Magnetic drums have been the most popular of the magnetic devices (see Figure 4). Information is stored on these drums by magnetizing certain areas on their surfaces. As the drum is rotated, reading heads pick up the patterns of dots and blank spaces and electronic circuits decode these patterns. Since the drum must rotate to the spot where the desired information is located, a relatively long period of time is consumed.

Ferrite magnetic cores constitute one of the best ways to store information. A memory composed of these cores has no moving parts and permits reaching any item of information directly. These cores are tiny doughnut-shaped rings arranged in crisscross patterns in matrix form as shown in Figure 5. A network of ferrite magnetic cores is strung on screens of wire, two wires passing through the hollow

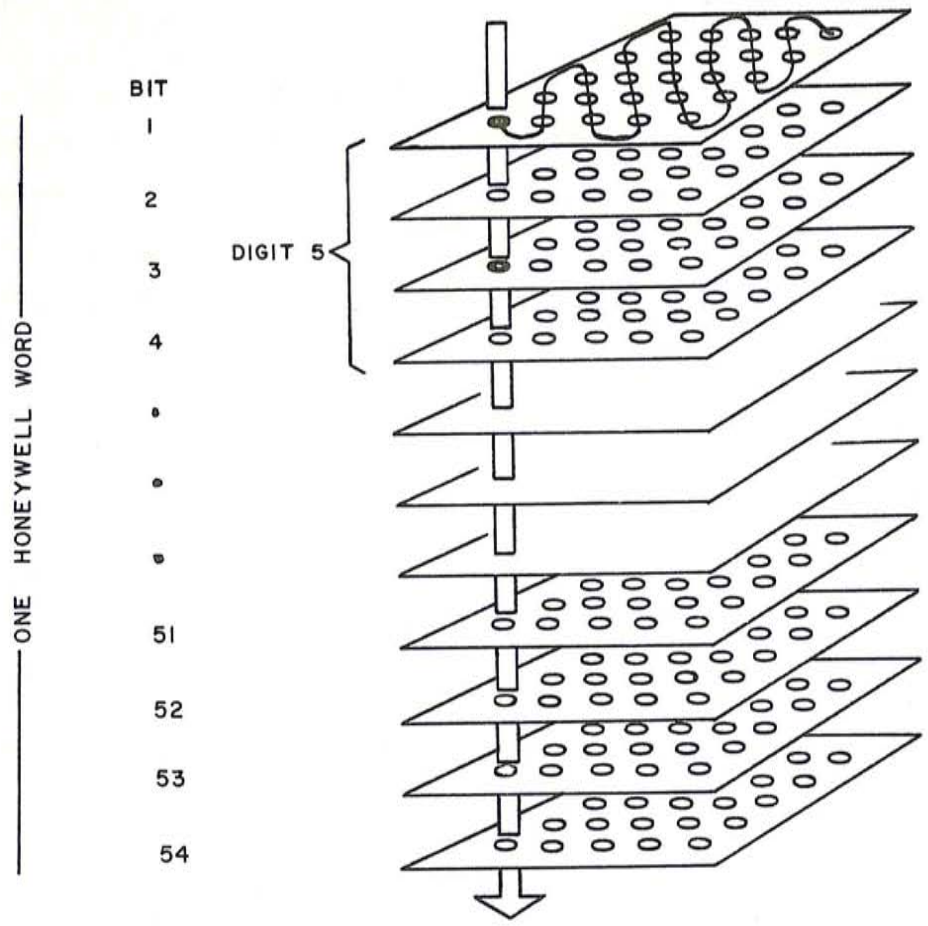


Figure 5. Magnetic Core Array

center of the core perpendicular to each other. When current is passed through a pair of wires, it magnetizes the core at the point where they cross, but at no other point. Information is called out of memory by reversing the process and reading the stored data by means of a third wire running diagonally to the others. The cores are extremely stable, and can last indefinitely. Their small size permits a large storage capacity to be housed in relatively little space. Most important of all is the speed at which data can be extracted, as short as a millionth of a second which makes this one of the fastest storage devices. The specific group of cores used to store information is referred to as a location and is designated by a number or symbol. Such a designation is called an address.

TAPE STORAGE

For storage of more than a few million characters only magnetic tapes are feasible at the present time. Tapes have been successfully developed which, on a standard-length reel, can store over 20,000,000 decimal digits. The speed of a tape is really measured in digits per second, rather than in inches per second, and consequently the tape speed can be increased by either moving it faster or by recording more digits per linear inch.

BASIC COMPUTER

Figure 6 is a block diagram showing the basic components of an electronic computer. These are features of all computers, whether general-purpose or special-purpose. The principle component is the memory, which can be of the magnetic drum, or magnetic core type. The arithmetic or logical unit can perform addition and subtraction, multiplication and division, comparison, transfer, and combinations of these functions. As indicated by the arrows, data can be transferred from the arithmetic unit to memory and back as many times as is necessary. Another important device is the Control Unit which is used to set the computer in motion and to check on its performance. Input and output sections constitute the remainder of a basic system.

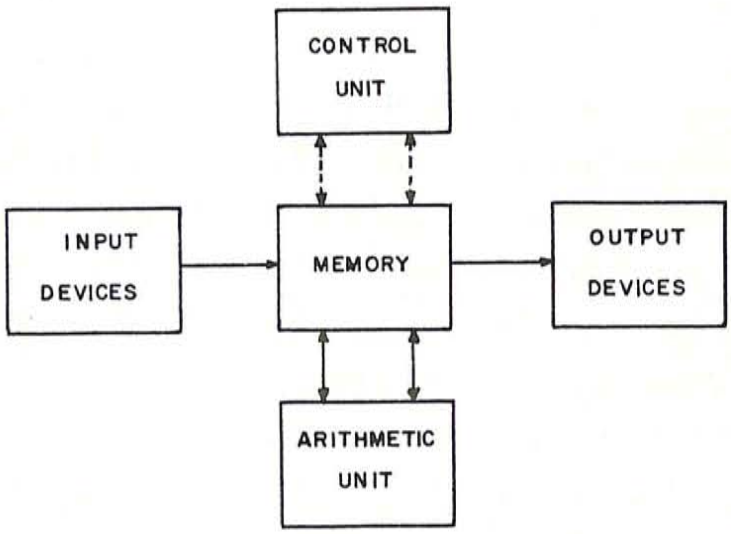


Figure 6. Basic Computer Components

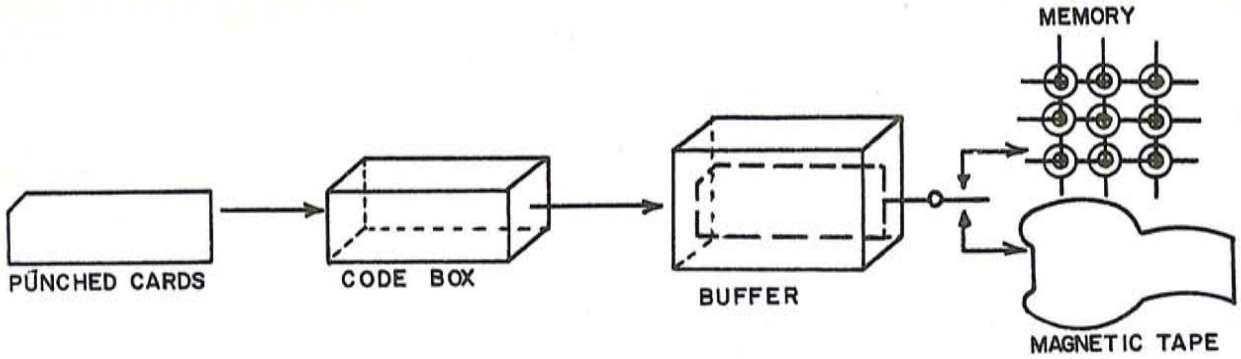


Figure 7. Input Section. Block Diagram

INPUT SECTION

The input devices in the basic computer are capable of reading data from punched cards or punched paper tape. A code box or interpreter translates this data from the language of the punched card to that of the binary code and the information is contained in buffer storage prior to being placed on magnetic tape in off-line operation or it is transferred to the Central Processor in on-line operation, (see Figure 7).

CENTRAL PROCESSING

To illustrate the basic features of a stored program in an elementary fashion consider a simple problem. Refer to Figure 8. Briefly the statement of the pro-

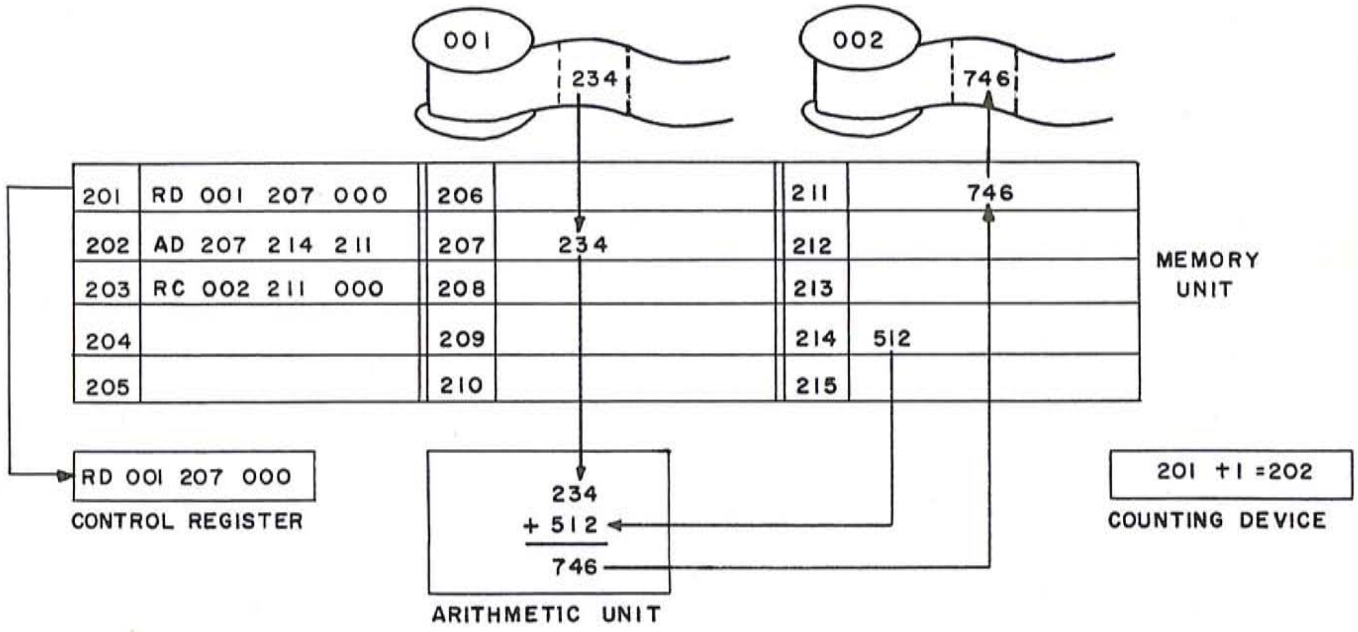


Figure 8. Processing with a Basic Computer

blem is to add the next number on magnetic tape reel number 001 to the number stored in location or address 214 in the high-speed memory and to record the result on the next available space on tape reel number 002. Assume that the three instructions necessary to complete this sequence have been recorded on magnetic tape and transferred to memory locations 201, 202, and 203. A counting device which designates the location of the next instruction to be processed has been set so that the first instruction will be taken from memory location 201 and placed in the control register. This instruction (Rd 001 207 000) states: read the next number from tape 001 and store it in memory location 207. Meanwhile the counting device has been automatically incremented by one so that the next operation will be taken from location 202. The instruction in 202 (AD 207 214 211) is interpreted to read: add the contents of location 207 to the contents of 214 and store the result in memory register 211. The counter, having been again incremented by one, now directs the next instruction to be placed in the control register. RC 002 211 000 requires that the contents of 211 be written on the next available space on tape reel 002.

OUTPUT SECTION

Once the result of central processing has been determined, it may be either recorded on magnetic tape and filed for future reference or it may be directly transcribed into printed form or punched cards may be generated. The Output Section block diagram (see Figure 9) illustrates that the data from magnetic

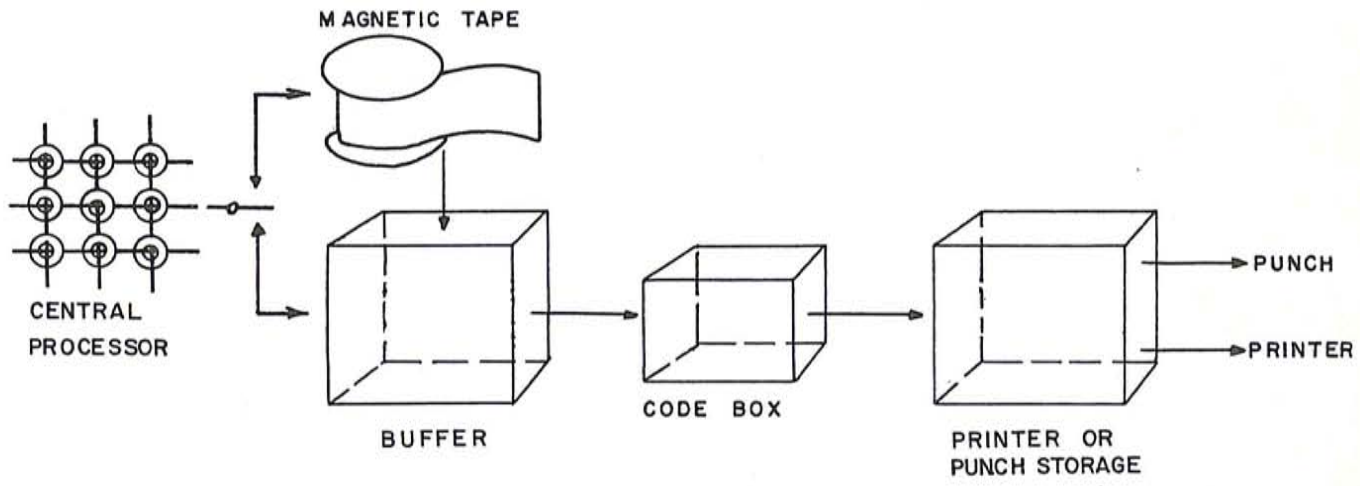


Figure 9. Output Section, Block Diagram

tape or from memory is temporarily stored in the output buffer device in the form of the binary code. The code box translates this information into electrical impulses which will drive specific equipment to print or punch the data.

In addition to the arithmetic operations involving payroll calculations, utility billing, mortgage payments, calculation of dividends, etc., the computer is capable of performing many other functions required by today's business procedures. It can make comparisons between numbers or other characters and to take the action called for by the result. Furthermore, it directs the processing operations within itself and controls the flow of input and output information. The electronic computer is ideally suited to perform one of the most cumbersome tasks in everyday business proceedings, that of file maintenance. Sorting, merging and production of updated files can be accomplished at phenomenal speeds with incredible accuracy. Summarization of production and control statistics, total expenses for accounting records and daily or weekly sales information present no problem to automatic data-processing methods. Competitive conditions necessitate the rapid consolidation of operating information to provide management with the tools of planning and control upon which to base its decision for future operational policies.

INTRODUCTION TO THE HONEYWELL 800

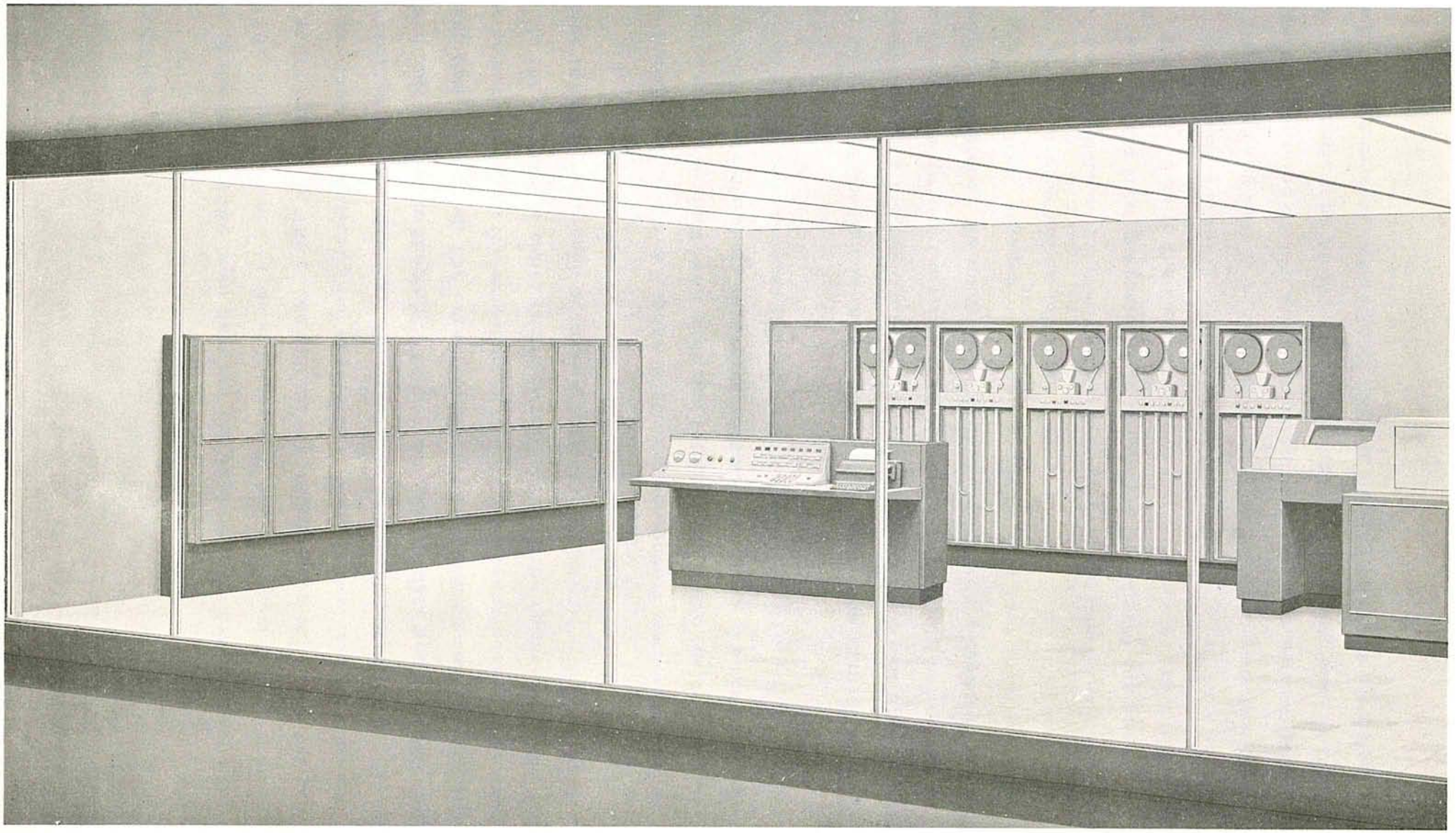
The Honeywell 800 is a high-capacity electronic data-processing system designed specifically for application to the increasingly complex problems and procedures demanded by today's modern business. It is composed of fully integrated business machines, employing high-speed computer principles designed to expedite all phases of record-keeping and accounting. The system incorporates significant new systems techniques as well as several basically new component developments dictated by a thorough understanding of the needs of its users. One of the primary features of the Honeywell 800 is its ability to process up to eight completely independent programs simultaneously. In addition, any reasonable

number of peripheral devices such as tape units, card readers, punches, printers, etc., may be in simultaneous operation. Data can be fed from each tape unit to the processing section and back to tape at a transfer rate of 96,000 decimal digits per second. In addition, the operational speed of the processing section maintains full compatibility with this high speed of information transfer.

Two of the most cumbersome aspects of most business problems are sorting and file maintenance. The Honeywell 800 is equipped with an extensive and flexible set of instructions, designed specifically to excel in the performance of these functions and many others. Automatic programming aids, including a library of utility programs, is supplied with each system. Library routines for fundamental scientific problem solving, and for automatic generation of sorting and merging routines according to parameters supplied by the programmer are also available.

In the Honeywell 800, reliability is a foremost consideration throughout every aspect of design and engineering. The design of the electronic circuitry is highly conservative. The circuitry is entirely constructed of solid state devices such as transistors, magnetic cores, semi-conductor diodes, etc., with the high reliability inherent in such devices. In addition, an extensive checking system is incorporated which will immediately sense any failure in data storage, data transmission, or control. All units of the system are constructed of easily replaced standard packages to facilitate maintenance. A system of marginal testing includes circuitry and a special program which may be run periodically to locate any package which should be replaced because of marginal performance. With proper use of this facility, most potential machine malfunctions will be corrected before they occur.

The Honeywell 800 system may be conceived functionally as comprising three main sections, the Input, Central Processor, and the Output sections. The requirements of the user will determine the componenents that are necessary for his particular installation. A typical Honeywell 800 System Layout appears in Figure 10.



INPUT SECTION

One of the units of the Input Section, the High-Speed Card Reader, reads standard 80-column punched cards at the rate of 900 fully punched cards per minute. In its operation, the cards are loaded in quantities of over 3,500 at a time and are then automatically fed sequentially through two reading stations. If these readings are not identical, three options are available to the programmer: (1) the card may be sent to the reject hopper without the data being transferred; (2) the card may be ejected, the error noted and the data transferred; or (3) the card may be accepted with the error noted and the data transferred. The operator controls the settings of panel switches which direct the course of action that the machine is to follow in each of these situations.

The function of the Card Reader Control is to convert source data from punched-card code into Honeywell code and temporarily store the contents of the card prior to processing or recording on magnetic tape. The card of information is stored as ten 54-bit words (48 information and 6 parity).

The Magnetic Tape Unit is the device through which the Central Processor communicates with its magnetic tape. The volume of transactions and the complexity of operations govern the number of Magnetic Tape Units to be used in a Honeywell 800 system. As many as 64 of these units with a capacity of millions of digits can be actively engaged in a single system. The four principal functions performed by this device are as follows:

1. Records data on magnetic tape from off-line peripheral equipment.
2. Reads data from magnetic tape to off-line peripheral equipment.
3. Transfers data to the Central Processor.
4. Records data from the Central Processor.

A single Magnetic Tape Unit is capable of reading or recording as many as 96,000 decimal digits or 64,000 alphanumeric characters per second (peak read-write rate). Magnetic tape moves under the reading-recording head at 120 inches per second and can be scanned while the tape is moving in either direction.

The operation of a tape unit may be an off-line as well as an on-line process. That is, it operates independently of or simultaneously with the data-processing functions.

CENTRAL PROCESSOR

The Central Processor is the heart of the Honeywell 800 system. It contains the electronic elements and circuitry for carrying out, at high speeds, all arithmetic and logical operations. In the execution of a typical data-processing task, the Central Processor performs several functions.

1. Reads the step-by-step instructions from a Program tape and stores them in the High-Speed Memory component.
2. Obeying these instructions, the Central Processor automatically reads data from other magnetic tapes or from peripheral devices and stores this data in memory.
3. In accordance with the instructions, the data is manipulated and the results are retained in memory.
4. These results may then be read onto magnetic tape for storage purposes, transcribed to printed copy, or converted to punched cards or punched paper tape.
5. With Parallel-Processing, the functions listed above relative to eight independent programs may be in simultaneous operation.

The Central Processor includes the Arithmetic and Control Units, the High-Speed Memory, Traffic Control, Multi-program control, and the Console.

Information flows from buffers located in the control units of the input devices or tape unit. Under control of the stored program, the data enters the High-Speed Memory, is processed and proceeds serially through buffers located in the control units of the output device or tape unit. These buffers permit a

steady flow of information to and from memory and enable the memory to read from eight tapes and write on another eight simultaneously. The fast and reliable High-Speed Memory consists of tiny, doughnut-shaped magnetic cores having a capacity of from 4,096 to 16,384 Honeywell words.

The design of the Central Processor and the provision of special instructions are specifically aimed at the attainment of high speeds for sorting, merging, and file-maintenance procedures. Some examples of the speed achieved are:

1. Sorting--up to 500,000 decimal digits per second
2. Merging--up to 500,000 decimal digits per second
3. File Maintenance--up to 660,000 decimal digits per second

The sequence of performance of the stored instructions for up to eight programs in parallel is directed by the Multi-Program Control. Arithmetic calculations are carried out by the Arithmetic Unit. The high speeds attained by this unit are exemplified as follows:

1. Addition or subtraction--30,000 average (separate operations) per second
2. Multiplication--4,500 (average) per second

The supervision and master control of the system is exercised through the Central Console. Although the Central Processor is automatic in its operation, there may be occasions when manual intervention is desirable. The operator of the Console, therefore, has available a special automatic typewriter which is used for the manual insertion of data or instructions to the Central Processor and for the interrogation of storage areas within the machine.

OUTPUT SECTION

Components of the Output Section, printers, card punches, etc., accept

information recorded on magnetic tape or transmitted from the High-Speed Memory and generate suitable electrical signals to permit the printing or punching of this information by specific equipment. Because of the diversified nature of business applications, both standard-speed and high-speed units are available for each type of output. An installation may contain either or both of these units depending upon its particular need. The operation of these units may be on-line or may be entirely independent of the main machine. Each output device accomplishes the conversion of information from machine language into the form suitable for generating punched cards, printed copy, or punched paper tape.

The Printer Control reduces data stored on magnetic tape or transmitted from memory to a form acceptable to a standard 150-line-per-minute tabulator or a 600/900-line-per-minute printer. Information is transferred a word at a time to the peripheral buffer. The data is then decoded and stored in Print Storage in quantities of 15 data words. Each group of 15 words becomes the basis of one line of printed output. A code box translates the machine language into printer code for actuating the print hammers. The line length for either printer is 120 characters (10 to the inch) and line spacing is six to the inch. A total of 56 characters: 26 letters, 10 digits and 20 special symbols are available. A speed of 600 or 900 lines per minute, selected by a switch, is attained by the high-speed printer.

Either of two Card Punches may be used with the Card Punch Control. Data from magnetic tape or from memory is transferred a word at a time to the peripheral buffer in the Control Unit. Ten words, representing a fully punched card, are temporarily stored in the buffer and become the basis of one 80-column punched card. The data is then translated into the card code. An output rate of 6,000 cards per hour per punch is achieved by the standard-speed punch and 15,000 cards per hour by the high-speed punch.

HONEYWELL TAPE

Magnetic tape is used in the Honeywell 800 as the primary storage medium since it makes possible large volume capacity with considerable space economy. Information is recorded on 3/4 inch Mylar-base magnetic tape in nine longitudinal channels (see Figure 11). Eight information bits and one checking or parity bit are recorded across the width of the tape and constitute one frame. Six frames, containing 48 information bits (and 6 parity bits), constitute a Honeywell 800 word.

Maximum capacity of each reel of tape is approximately 20,000,000 decimal digits or 1,666,666 words. The information transfer rate is 96,000 decimal digits per second, per magnetic tape unit.

Each group of words written or read as a result of a single instruction is called a record. These records may vary in length up to a maximum of 400 words. Successive records are separated by an inter-record gap.

Writing occurs with the tape moving in a forward direction. Reading may be done with the tape moving either forward or backward. This read-in-reverse feature eliminates wasteful rewinding time that would otherwise be necessary in operations such as sorting, searching and file maintenance.

In addition, the user has the option of using a fast-rewind feature included

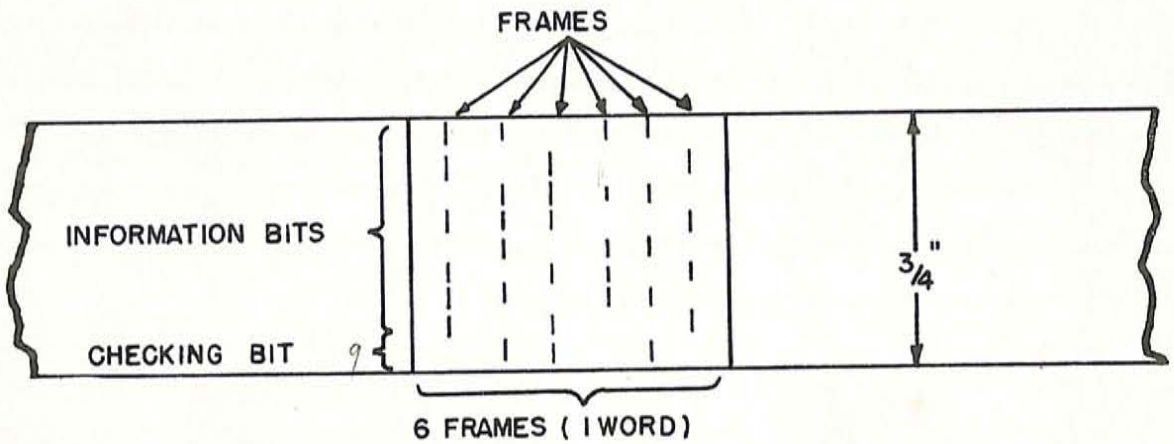


Figure 11. Honeywell 800 Tape

in the magnetic tape units, or removing wound tapes and rewinding them off-line.

All Honeywell 800 magnetic tape units are compatible with all other Honeywell 800 units and the high degree of switching flexibility available minimizes tape handling and the number of magnetic tape units required by a given application.

THE HEXADECIMAL NUMBERING SYSTEM

Table 2 illustrated that only ten of the possible sixteen 4-bit configurations were required to represent the ten decimal digits (0 through 9). For the purpose of compacting information and in order to realize the full inherent possibilities of the Honeywell 800, it is desirable at times to make use of a number system which utilizes all sixteen 4-bit configurations. The hexadecimal numbering system employs 16 distinct symbols for representing numerical information. The particular symbols employed in this method and their decimal and binary equivalents are shown in Table 3. The significance of any one of these hexadecimal symbols in a number depends upon its particular position in the number. For example, the hexadecimal number 2 (10) (15) is represented as 0010 1010 1111 in the binary-coded hexadecimal system or, in the decimal system, as $(2 \times 16^2) + (10 \times 16^1) + (15 \times 16^0)$ or 687. Once again, reading from right to left, the decimal multi-

Table 3. Hexadecimal Numbers

Decimal	Hexadecimal	Binary	Decimal	Hexadecimal	Binary
0	0	0 0 0 0	8	8	1 0 0 0
1	1	0 0 0 1	9	9	1 0 0 1
2	2	0 0 1 0	10	(10) or (B)	1 0 1 0
3	3	0 0 1 1	11	(11) or (C)	1 0 1 1
4	4	0 1 0 0	12	(12) or (D)	1 1 0 0
5	5	0 1 0 1	13	(13) or (E)	1 1 0 1
6	6	0 1 1 0	14	(14) or (F)	1 1 1 0
7	7	0 1 1 1	15	(15) or (G)	1 1 1 1

pliers are successive powers of a number: 1, 16, 256, etc. As can be seen from the preceding table, the hexadecimal number 2 (B) (G) is equivalent to 2 (10) (15) . Whenever their use clearly indicates that hexadecimal symbols are meant, they will, in the future, be written as 2 B G.

THE OCTAL NUMBERING SYSTEM

Another system of numbering which is of interest to users of electronic equipment is based upon a radix of 8. This "octal" notation is directly analogous to the position notation concept previously described, having eight digits (from 0 to 7) and requiring that the digit in each position be multiplied by the appropriate power of eight. For example, the number 3742_8 can be converted to a decimal quantity as follows:

$$3 \times 8^3 + 7 \times 8^2 + 4 \times 8^1 + 2 \times 8^0 = 1536 + 448 + 32 + 2 = 2018_{10}$$

The usefulness of octal notation stems from the fact that it is a convenient shorthand method for writing pure binary quantities. Any pure binary number may be written in octal by combining bits in groups of three, working from the indicator point in either direction or both if necessary, and writing the octal value of each three-bit group. The binary number 1001100101, for example, is converted to octal as follows:

$$\begin{array}{cccc} 1 & 001 & 100 & 101 \\ \downarrow & \downarrow & \downarrow & \downarrow \\ 1 & 1 & 4 & 5 \end{array} \quad \text{or } 1145_8$$

Note that an octal number can be converted easily into binary form by merely writing the binary equivalent of each octal digit.

The following method is suggested for the conversion of integral decimal numbers into either octal or binary. Divide the decimal integer by the radix of the desired notation (8 or 2). The remainder of this division is the low-order digit of the desired result. The quotient is again divided by the same radix to

obtain the next digit. This process is repeated until the quotient is zero. As an example of this method, the decimal number 979 is represented in binary and octal as follows:

quotient	remainder
979 ÷ 2 = 489	1
489 ÷ 2 = 244	1
244 ÷ 2 = 122	0
122 ÷ 2 = 61	0
61 ÷ 2 = 30	1
30 ÷ 2 = 15	0
15 ÷ 2 = 7	1
7 ÷ 2 = 3	1
3 ÷ 2 = 1	1
1 ÷ 2 = 0	1

quotient	remainder
979 ÷ 8 = 122	3
122 ÷ 8 = 15	2
15 ÷ 8 = 1	7
1 ÷ 8 = 0	1

Thus, the number 979_{10} is equivalent to 1111010011_2 and 1723_8. This conversion can be quickly checked by writing out the binary equivalent of each octal digit.

THE SIX-BIT CODE

A computer designed for business data processing must be able to manipulate conveniently the type of information contained in records, reports, tables, etc. This information often involves an intermixing of letters, numbers, and other alphabetic symbols arranged in some meaningful fashion. For the processing of alphanumeric data, it is desirable to have a binary code in which the same number of bits is used for each of the alphanumeric characters. A 5-bit code permits 32 distinct groups of binary zeros and ones. Since the Honeywell 800 recognizes 58 individual characters (10 digits, 26 letters and 22 symbols), a

6-bit code which permits up to 64 distinct character representations has been chosen as shown in Table 4. To obtain the 6-bit code for any character, write the two digits in the heading of the column containing the desired character followed by the four digits in the row containing the desired character.

Table 4. The Six-Bit Binary Code

0 0	0 1	1 0	1 1	
0	space	-	not used	0 0 0 0
1	A	J	/	0 0 0 1
2	B	K	S	0 0 1 0
3	C	L	T	0 0 1 1
4	D	M	U	0 1 0 0
5	E	N	V	0 1 0 1
6	F	O	W	0 1 1 0
7	G	P	X	0 1 1 1
8	H	Q	Y	1 0 0 0
9	I	R	Z	1 0 0 1
'	cr.	=	:	1 0 1 0
,	#	$	;	1 0 1 1
@	□	*	%	1 1 0 0
&	(	¢	½	1 1 0 1
.	)	not used	not used	1 1 1 0
+	not used	not used	not used	1 1 1 1

THE HONEYWELL WORD

The fundamental parcel of information which the Honeywell 800 can manipulate is called the Honeywell word. The five basic types of words are numeric, alphanumeric, binary, instruction, and floating point, each of which has a prescribed bit structure. Each word is composed of a collection of 54 binary zeros and ones (48 information and 6 parity). Since the parity bits are treated automatically and do not directly concern the programmer, only the 48 information bits will be considered. These bits are numbered 1 through 48 when read from right to left.

In a numeric word, (see Figure 12) the 48 bits are grouped into twelve 4-bit sets of numeric digits. The leftmost digit (bits 45-48) may be used to represent a legitimate numeric value or may contain a code which represents the sign of the word: 1 1 1 1 equals plus (+) and 0 0 0 0 equals minus (-).

Bit Position	48---------- 37			36--------25			24-------- 13			12----------1		
Digit Position	sign or 12th	11th	10th	9th	8th	7th	6th	5th	4th	3rd	2nd	1st

Figure 12. Honeywell Numeric Word

The alphanumeric word is composed of eight 6-bit characters (see Figure 13). The 64 configurations possible with the 6-bit code were listed in Table 4.

Bit Position	48--43	42--37	36--31	30--25	24--19	18--13	12---7	6---1
Character Position	8th	7th	6th	5th	4th	3rd	2nd	1st

Figure 13. Honeywell Alphanumeric Word

For the purpose of compacting information, it may be advantageous to use a mixed word form in which both 4-bit and 6-bit characters appear. For example, in an inventory control application, stock numbers may be comprised of letters and decimal digits with a typical stock number given as DADA123456. The four letters occupy 24 bit positions and the six decimal digits, 24 bit positions. Thus, the total of 48 bits constitutes a Honeywell word.

Each frame of eight binary information digits contains a ninth bit called the parity bit. If the sum of the "ones" in a frame is even, a one is placed in the parity channel. If the sum of the one bits is odd, a zero is used. Thus, each frame (including the parity channel bit) consists of an odd number of one bits. The machine performs a check on the reading, recording and transfer of all data to insure that each frame has an odd total number of ones including information and check positions.

ORTHOTRONIC CONTROL

Orthotronic Control is a method of replacing a defective word which has become garbled or lost on tape. Two Orthotronic words are appended to each record on tape and are generated in the following manner. For a specific channel, groups of six bits in successive odd words are summed without carry (Half Add) for the entire record. The resultant six-bit sum is appended to the specific channel. The remaining seven information channels are treated in a similar fashion. The result comprises one of the two Orthotronic words. The successive even words of the entire record are similarly treated and a second Orthotronic word is generated.

When a VTR parity error is detected, the machine locates the erroneous channel by Half Adding all words in the record and comparing the results with the Ortho-words. The frames containing errors are determined by the parity bits. Thus, through Orthotronic Control the location of the incorrect bits is completely specified in a cross-sectional manner by frame and channel. The erroneous bits are then corrected.

MAGNETIC-CORE MEMORY

A magnetic-core memory is the basic storage unit for both data and instructions. It is available in modules of 4,096 words, up to a maximum of four modules. Each word of core storage is individually addressable. Parallel transmission is employed in sending words to and from storage. Memory-access time is six microseconds. High-speed, random-access drum storage is available as optional equipment.

INSTRUCTIONS

Instruction words used in the Honeywell 800 have the same structure as data words (see Figure 14). The 48 information bits in each instruction are divided

Bit Position	48---------- 37	36----------25	24----------13	12----------1
Instruction Word	Operation Code	Address A	Address B	Address C

Figure 14. Honeywell Instruction Word

logically into four sections of 12 bits each. For most operations, these sections are interpreted as an operation code followed by three addresses. The operation code designates the type of work to be performed by the machine. For example, an instruction such as DA/600/601/602 signifies that the contents of memory location 600 are to be added decimally (binary addition is also possible) to the contents of memory location 601 and the result is to be placed in memory location 602. The 12 binary digits of the Honeywell 800 Operation Code are used to designate whether the instruction is peripheral, masked, standard, floating-point option, etc.

INDEXING

To facilitate programming, memory locations may be addressed in either absolute or indexable form. The first of the 12 bits in each address specifies whether the address is to be used as stated (absolute form), or whether it is to be augmented (indexable form). If the address is in absolute form, it means that the remaining 11 bits are used to denote the address of a specific memory location. If the address is in indexable form, the next three bits will be used to select one of eight index registers assigned to the particular program. (A total of eight index registers is included in each special register group.) The contents of the specified index register are then added to the remaining eight bits of the address group and the sum is used to denote the desired memory location.

PROGRAMMING

Up to eight programs may be active at the same time and each is a completely independent entity. Therefore, the logic and programming facilities may be explained as if only one program were being processed. All programs proceed in an automatic fashion and the programmer need not be concerned with the number or size of the other programs, nor with the necessity of designating priorities among the several programs.

Each program is controlled by an independent sequence counter and may have any reasonable number of input and output trunks available.

SEQUENCING

As data is processed within the Honeywell 800 System, instructions are carried out in the order specified by a sequence counter. The sequence counter designates the address of the next instruction to be performed. Normally, it is incremented by unity each time an instruction is performed. Certain instructions, however, can be used to set the sequence counter to a predetermined value permitting a departure from the basic sequence. To provide a means of returning to the proper point in the program, a history register is used to record the setting of the sequence counter at the point of departure.

To provide still greater flexibility, there is a co-sequence counter associated with each sequence counter. The selection of successive instructions to be performed may be controlled by either the sequence counter or the co-sequence counter at the option of the programmer. This is called operation in the bi-sequence mode. A history register is also provided for the co-sequence counter.

INTER-PROGRAM CONTROL

In some applications, notably those which require searching on tape, it may be advantageous to allow one program to control another. An example of this might be a file maintenance program for a low-activity application where a multiple-tape search technique is employed. In such a case, a program operating under control of one sequence counter may insert a starting value into another sequence counter and cause it to become active. Thus, a second program may be started with the same technique a human operator would employ at the console. A program may not only start another, but it may also monitor its progress and stop it, if necessary. Any simultaneous peripheral operation may also be monitored by a program to determine when it has been completed.

SIMULATOR INSTRUCTIONS

The complement of instructions in the Honeywell 800 has been designed to facilitate all normal data processing operations of both a business and scientific nature. For those exceptional situations where standard orders are not sufficient, the programmer has the unique facility of incorporating special-purpose instructions called simulator instructions.

A simulator instruction, though not a built-in instruction, is used by the programmer as if it were. The system responds to it by calling in a subroutine. There is no practical limit to the number of simulator instructions (and their associated subroutines) that may be included in the Honeywell 800, provided sufficient memory space is available. Thus, any set of orders may be incorporated in the system. This feature provides the ability to simulate any special-purpose order structure which the programmer may envision.

MASKING

In many instances it may be desirable for the programmer to compare portions of words, to perform arithmetic operations on fields of less than word length, or to select data from within a word for transfer to other processing operations. This is accomplished in the Honeywell 800 by highly versatile masking abilities. The use of mask words permits the internal processing instructions to isolate any portion of an information word and ignore the remainder. By inserting a number into a special register, called a mask index register, the programmer may designate any group of 96 memory locations as storage positions for masks used with his program. In each masked order, the programmer specifies which one of these is to apply for that operation. The mask index register contents may be changed by the programmer at any point in his program. Thus, an essentially unlimited number of locations for storing masks is at his disposal.

FLOATING-POINT ARITHMETIC

The floating-point arithmetic logic which may be added to the system greatly enhances the computing capabilities of the Honeywell 800. Although this type of arithmetic is not normally required in business data processing applications, it is extremely useful in scientific areas. Furthermore, the 40-bit mantissa provides about 50% more precision than that available in most computers designed specifically for scientific purposes. Inefficient and costly multi-precision programming may, therefore, be avoided in many cases.

SORTING AND FILE MAINTENANCE

Because of its magnetic tape speeds, Honeywell's DATAmatic 1000 was one of the first electronic data processing systems to make electronic sorting economical.

The Honeywell 800 is capable of sorting even faster. In minimal form and using the most common method of sorting, Honeywell 800 sorts more economically than any other system.

The same specifications that give the Honeywell 800 its sorting abilities provide advantages in file maintenance operations.

A typical file maintenance process involves matching a number of transactions against corresponding main file records, processing them, and writing both the updated records and the inactive records on a new magnetic tape. The updated record, or some part of it, is normally written on an output tape for subsequent printing. The number of transactions in a given run is nearly always small in relation to the total number of records. Consequently most time is spent bypassing inactive records, and high tape speeds provide an obvious advantage.

With large files the tape-speed advantage can be multiplied by a factor from two to eight. This is done by adding tape controls and magnetic tape units so that searching is speeded up. A second tape control, for example, will generally cut file maintenance time in half.

In summary, then, the Honeywell 800 provides management with a complete and unified system which is capable of performing economically (in terms of both time and cost) all the business processing activities which previously required both an electronic computer and associated tabulating equipment.

PROCESSING WITH THE HONEYWELL 800

The flow of data in the Honeywell 800 system is given in Figure 15. Under control of a single instruction contained in the control unit, a complete record of up to 400 words may be read from Tape 1 and temporarily stored in any convenient group of memory locations. The Tape Control contains a two-word input

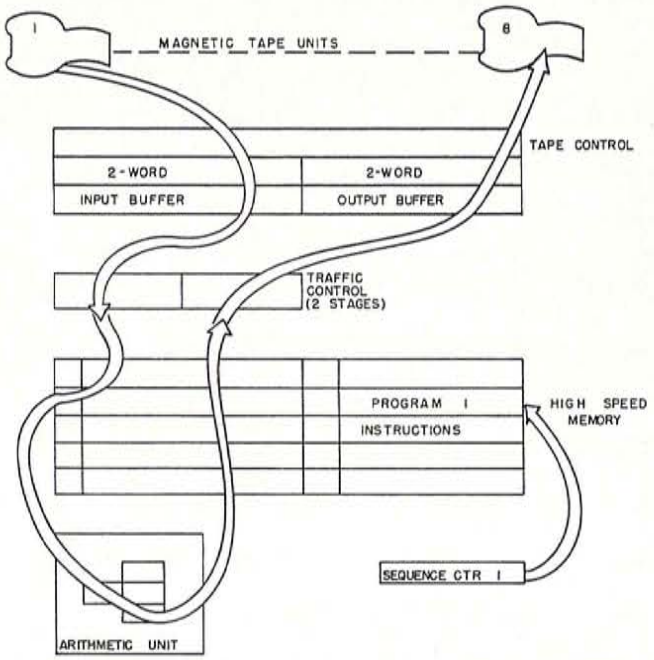


Figure 15. Central Processor Run

buffer and a two-word output buffer. It is important to note that this unit has the ability to accept one word of information from tape while transferring another word to the core memory for processing. An optimum rate of information flow between the Central Processor and tape units or any of the terminal equipments is insured by Traffic Control. When the proper traffic control signal has been generated, the Central Processor will accept a word from the input buffer or deliver a word to the output buffer. Each location or address in memory can store 48 information bits or one Honeywell word. The illustration indicates that a computation has been performed in the arithmetic unit, the result returned to memory and subsequently written on Tape 8.

Figures 16 and 17 illustrate the flow of data during an input conversion run and a printing run. All information is transferred to or from a terminal device and the Central Processor via a peripheral buffer. This buffer is a one-word storage unit and is an integral part of each control unit. Within these control units the information is decoded and checked. Figure 18 is a composite illustration of the three previously indicated programs all operating simultaneously. In order to control the transfer of data, each peripheral device is assigned a stage of traffic control and the Tape Control is assigned two stages of traffic control, one for input and one for output.

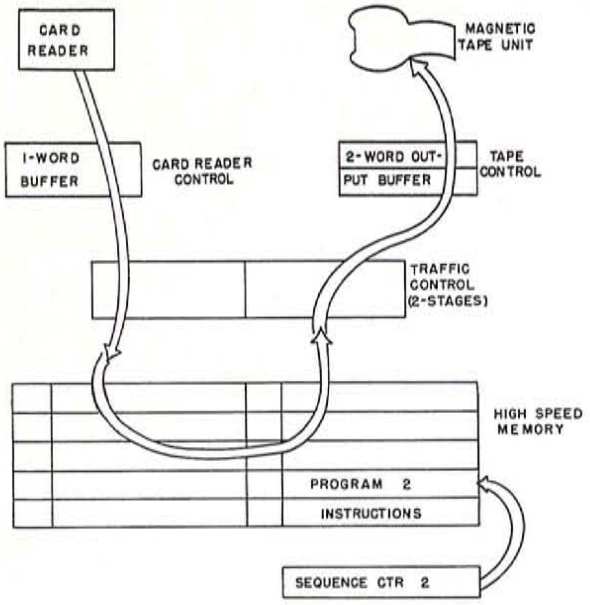


Figure 16. Input Conversion Run

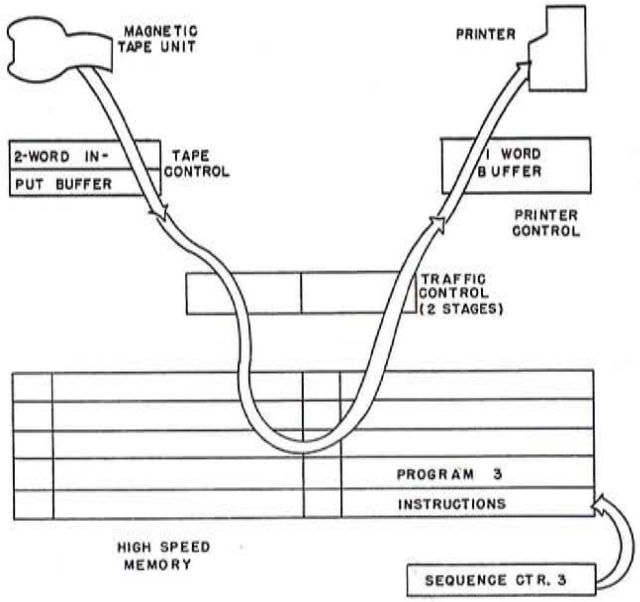


Figure 17. Printing Run

Each instruction, in turn, for each program is selected from the high-speed memory location whose address is contained in the sequencing counter related to that program. Unity is automatically added to the contents of the sequencing counter immediately following the selection so that the next operation to be performed in that program is obtained from the new setting of the sequencing counter.

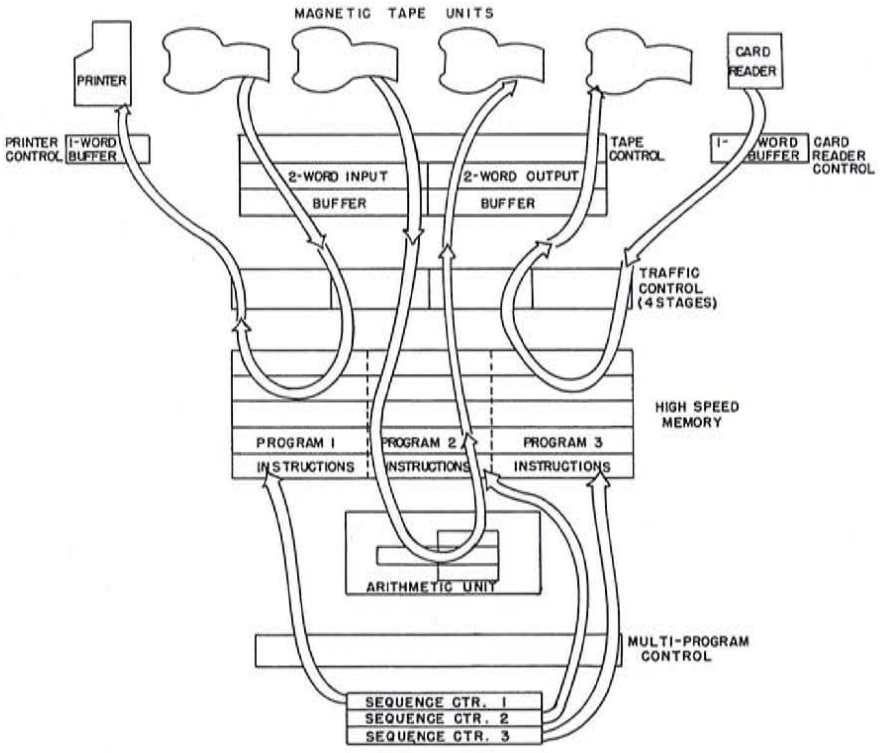


Figure 18. Three Programs Operating Simultaneously

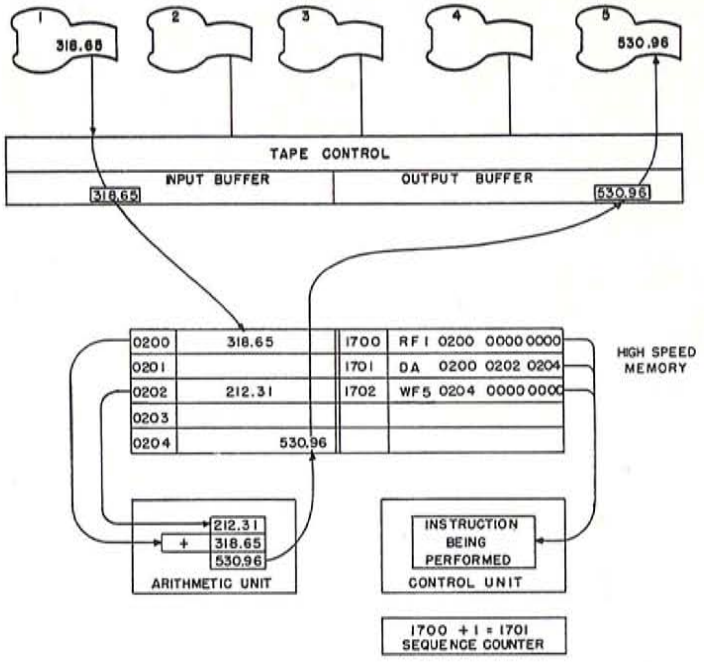


Figure 19. An Example in Addition

An example of a simplified problem in addition, with the associated instructions for its completion, is given in Figure 19. Assume that the sequencing counter at this time reads 1700. Therefore, the instruction in memory location 1700 (RF 1 0200 0000 0000) will be transferred to the control unit. RF 1 is the code which instructs the machine to read Tape 1 in a forward direction and to transfer the entire record into memory, beginning with location 0200. Meanwhile, the sequencing counter has been incremented by one and when the read instruction is completed, the instruction DA 0200 0202 0204 in location 1701 will be performed. This requires that the arithmetic unit add decimally the contents of location 0200 (318.65) to the contents of 0202 (212.31) and store the result (530.96) in location 0204. The instruction in 1702 states that the result contained in register 0204 be written in a forward direction onto the next available tape space on Magnetic Tape Unit 05.

1. Read data from tape or memory

2. Change machine code to card or printer code

3. Punch cards or print data

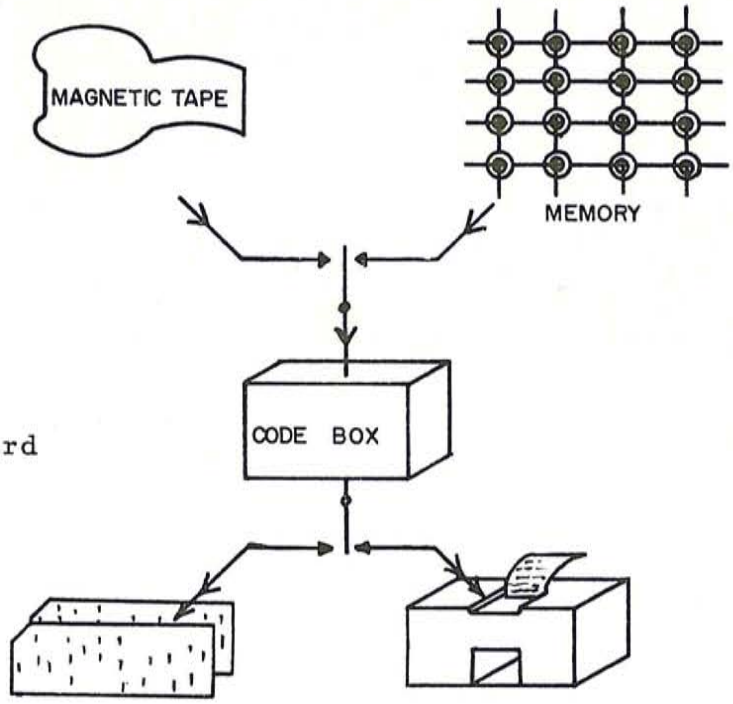


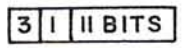
Figure 20. Output Conversion

Data from magnetic tape or memory is converted into printed or punched card form as shown in the block diagram, Figure 20. The function of the code box is similar to the one used in the input circuit except that the Honeywell code representation is converted to electrical impulses which will drive specific equipment to print or punch this information. Editing may be accomplished in the Central Processor to obtain any desired format.

HONEYWELL 800 ADDRESS NOTATION SUMMARY

SYMBOL	A	X	S	T	Y*
DESCRIPTION	ABSOLUTE	INDEXED, HSM	SPECIAL REGISTER	TABULAR	INDEXED, SR
FORM	0 | 11 BITS	1 | 3 BITS | 8BITS	0 | 5 BITS | 1 | 5BITS	0 | 5BITS | 0 | 5BITS	1 | 3BITS | 8BITS
LEGEND	aaaa	r iii	ii ss	ii ss	r iii
LEGEND DEFINITION	ABSOLUTE ADDRESS (0-2047)	IR (0-7) INCREMENT (0-255)	INCREMENT (0-31) SR LOC. (0-31)	INCREMENT (0-31) SR LOC. (0-31)	IR (0-7) INCREMENT (0-255)
MODIFIED BEFORE USE	NO	YES	NO	NO	YES
MODIFIED AFTER USE	NO	NO	YES	YES	NO
TAB BIT	—	—	1	0	—
ADDRESS DESIG.	0 (HSM)	0 (HSM)	1 (SR)	1 (SR)	1
BANK INDICATOR, SOURCE (HSM)	SEQ. CTR.	IR	—	SR	—
SECONDARY BANK INDICATOR, SOURCE (SR)	—	—	SEQ. CTR.#	SEQ.CTR. #	IR BANK INDICATOR IS USED INSTEAD OF THAT OF SEQ. CTR. IN SUBSEQUENT S OR T.
EFFECTIVE ADDRESS	aaaa	HSM ADDRESS IN (IR) + iii	ss	(ss)[1]	(IR) + iii = S OR T (TAB=1, TAB=0)

SPECIAL REGISTER WORD



BASE INDEX BINARY ADDRESS

*Resultant comprises bank indicator from Index Register plus sum of 11 bits from IR and 8 bits from Address. Resultant is interpreted as an effective S or T address depending on Tab position bit of resultant.

1 (ss) - HSM location specified by contents of Special Register location ss.

HONEYWELL 800 ORDER SUMMARY SHEET

OPERATION CODES

Class	Op	M/P	S/C	Description
S or -	00	--	S/C	General
P or -	00	PP	-	Peripheral
M	00	MM	S/C	Masked
S	00	--	S/C	Shift
F	00	--	S/C	Floating
D	aa	aa		Defined

OO: Op Code
S: (or blank) if next order comes from Sequence Register
C: if next order comes from Cosequence Register
PP: Peripheral address
MM: Mask address (low-order part)
aaaa: Address of defined routine
- represents blank

INSTRUCTION ADDRESSES (A-B-C)

Type	Location	Description
A or -	aaaa	High-Speed Memory
X	riii	Indexed High-Speed Memory Address
S	iiss	Special Register, direct use
T	iiss	Tabular use of Special Register
Y	riii	Indexed addressing of Special Registers

aaaa: High-Speed Memory Address
r: Index Register Code
ii: Increment (range 0-31)
iii: Increment (range 0-255)
ss: Special Register Code (Absolute or Symbolic)

SPECIAL REGISTER ASSIGNMENTS

Absolute	Symbolic	Function
0	AA	AUCU Counter 1
1	AC	AUCU Counter 2
2	SC	Sequence Counter
3	CC	Cosequence Counter
4	SH	Sequence History
5	CH	Cosequence History
6	UX	Unprgmd. Trf. Ind. Reg.
7	MX	Mask Index Register
8-15	X0-X7	Index Registers 0-7
16-23	G0-G7	General Purpose Reg. 0-7
24-31	H0-H7	General Purpose Reg. 8-15

CONSTANT SYMBOLS

BC Binary Constant (octal)
DC Decimal Constant
AC Alphanumeric Constant
AM Address constant in proper format for Special Register
C Control (assembly)
K Mixed Constant

*These orders are also available in the masked

OPERATION CODES

Op. Code	General	**WC
*DA	Decimal Add	4
*DS	Decimal Subtract	4
DM	Decimal Multiply	25
DD	Decimal Divide	52
*BA	Binary Add	4
*BS	Binary Subtract	4
BM	Binary Multiply	31
BD	Binary Divide	52
*WA	Word Add (binary 48 bit)	4
*WD	Word Difference (binary 48 bit)	4
*HA	Half-Add (binary 48 bit)	4
*SM	Superimpose	4
SS	Substitute	4
EX	Extract	5-6
*TX	Transfer A to B	3
*TS	Transfer A to B and go to C	4
TN	N-word Transfer	4+2n
MT	Multiple Transfer	4+2n
FT	Field Transfer	5+2n
RT	Record Transfer	5+2n
*NA	Not equal comparison, alphabetic	4
*LA	Less than or equal comp. alpha.	4
*NN	Not equal comparison numeric	4
*LN	Less than or equal comp. num.	4
AP	Alphabetic print	4
HP	Hexadecimal print	4
BP	Octal print	4
PRR	Proceed (no operation)	2
CC	Compute Orthocount	5+n
*CP	Check Parity	4

PERIPHERAL

RF	Read Forward	4
RB	Read Backward	4
WF	Write Forward	4
RW	Rewind	4
WT	Wind	4

SHIFT

WE	Shift Word, Extracting	5-7
WS	Shift Word, Substituting	5-7
PE	Shift preserving sign, Extracting	5-7
PS	Shift preserving sign, Substituting	5-7
SL	Shift and Select	5-8

FLOATING POINT

BA	Binary Add (Floating)	11 av
BS	Binary Subtract "	11 av
BM	Binary Multiply "	31 av
BD	Binary Divide "	54 av
DA	Decimal Add "	11 av
DS	Decimal Subtract "	11 av
DM	Decimal Multiply "	31 av
DD	Decimal Divide "	54 av
NZ	Normalize	8 av
NF	Not equal comparison floating	4
LF	Less than or equal comp. floating	4

**Word cycle equals six microseconds

Honeywell

DATAmatic
ELECTRONIC DATA PROCESSING

HONEYWELL 800 EQUIPMENT PRICE SCHEDULE

EFFECTIVE APRIL 1, 1959

TYPE NUMBER	DESCRIPTION	MONTHLY RENTAL	PURCHASE PRICE
	CENTRAL PROCESSOR		
801	Central Processor including 4,096-word memory, arithmetic and control section and console	$8,550	$410,400
801-A	Floating-Point Option	1,250	60,000
802	Additional Memory Blocks (4,096 words per block)	3,200	153,600
	TAPE CONTROLS		
803	Tape Control	2,000	96,000
804	Magnetic Tape Unit	900	43,200
805	Magnetic Tape Switching Unit	75	3,600
	PERIPHERAL CONTROLS		
	Printer Controls		
806-1	Printer Control (for 822-1)	1,050	50,400
806-2	Printer Control (for 822-2)	1,250	60,000
806-3	Printer Control (for 822-3)	1,450	69,600
	Card Reader Controls		
807-1	Card Reader Control (for 823-1)	950	45,600
807-2	Card Reader Control (for 823-2)	1,100	52,800
807-3	Card Reader Control (for 823-3)	1,350	64,800
	Card Punch Controls		
808-1	Card Punch Control (for 824-1)	1,050	50,400
808-2	Card Punch Control (for 824-2)	1,150	55,200
	Paper Tape Controls		
809-1	Paper Tape Reader Control (for 825-1)	650	31,200
809-2	Paper Tape Reader Control (for 825-2)	650	31,200
810-1	Paper Tape Punch Control (for 826-1)	650	31,200
	Multiple Terminal Unit Controls		
811-1	Printer—Card Reader—Card Punch Control (for use with 822-1, 823-1 or 823-2, 824-1 or 824-2)	1,700	81,600
811-2	Printer—Card Reader—Card Punch Control (for use with 822-2, 823-1 or 823-2, 824-1 or 824-2)	1,850	88,800

Continued on reverse side

TYPE NUMBER	DESCRIPTION	MONTHLY RENTAL	PURCHASE PRICE
811-3	Printer—Card Reader—Card Punch Control (for use with 822-3, 823-1 or 823-2, 824-1 or 824-2)	1,950	93,600
	OFF-LINE TERMINAL UNIT CONTROLS		
815	Off-Line Output Auxiliary Control	700	33,600
816	Off-Line Input Auxiliary Control	700	33,600
817	Off-Line Input-Output Auxiliary Control	950	45,600
	TERMINAL UNITS		
	PRINTERS		
822-1	Standard-Speed Printer (150 LPM)	800	42,000
822-2	Bill Feed Printer	1,055	63,300
822-3	High-Speed Printer (600/900 LPM)	1,950	79,800
	CARD READERS		
823-1	Standard-Speed Card Reader (240 CPM)	125	7,700
823-2	High-Speed Card Reader (650 CPM)	325	14,700
823-3	High-Speed Card Reader (900 CPM)	1,200	57,600
	CARD PUNCHES		
824-1	Standard-Speed Card Punch (100 CPM)	120	6,100
824-2	High-Speed Card Punch (250 CPM)	475	20,250
	PAPER TAPE UNITS		
825-1	Standard-Speed Paper Tape Reader	175	8,000
825-2	High-Speed Paper Tape Reader	325	15,000
826-1	Standard-Speed Paper Tape Punch	75	3,600
	MAGNETIC TAPE		
	Full Reel (2,500 feet — ¾ inch)		89.50
	Half Reel (1,300 feet — ¾ inch)		59.50
	One-Quarter Reel (700 feet — ¾ inch)		39.50

The above listed prices are those currently in effect.

All prices are f.o.b. the Honeywell plant in Newton or Boston, Massachusetts, and are subject to change without notice.

Any applicable city, state, or federal taxes involving the sale, lease, or maintenance of the above equipment are to be added to the listed prices.

A purchase option credit of fifty percent (50%) of the primary-shift rentals paid is applicable to all of the above equipment purchased within two years after installation.

Usage outside of primary-shift time is charged for at the rate of fifty per cent (50%) of the primary-shift charge, prorated for the actual time used.