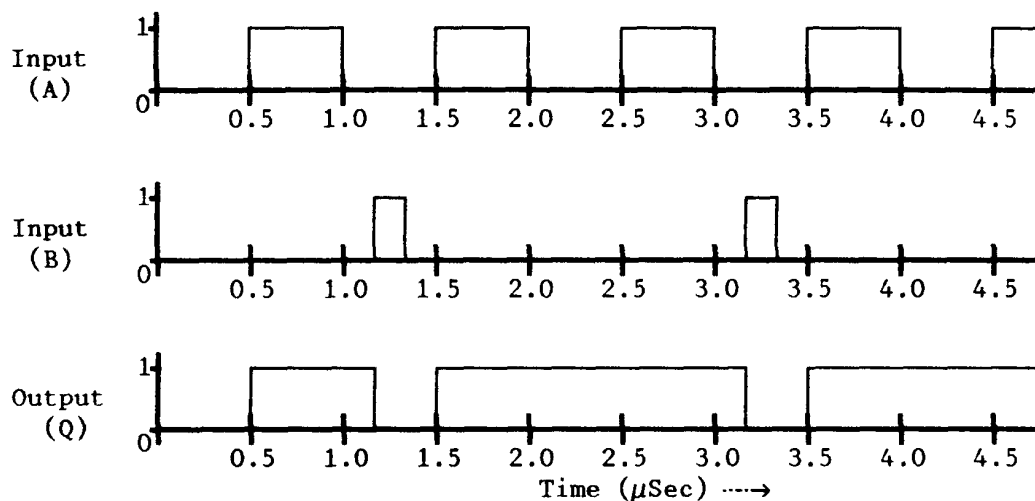


# Chapter 4. Sequential Logic

Sequential logic devices are used to construct sequential circuits such as computer memories, counters, frequency dividers, and data converters. Sequential circuits differ from combinational circuits because their outputs are determined not only by the current state of the inputs but also by past behavior.

Timing diagrams describe the state of input and output signals of a digital circuit as they vary with time. Timing diagrams are used to analyze sequential circuits because sequential logic devices are a function of not only the current state of the inputs, but also the past behavior. Figure 4.1 illustrates a timing diagram for a sequential circuit with two inputs (A and B) and one output (Q). Time is located on the horizontal axis while the state of the signal is shown on the vertical axis. The logic variables (A, B, and Q) are represented as waveforms in either the 1 or 0 state as they vary in time.

Figure 4.1 Timing Diagram



Most sequential logic circuits utilize a clock signal to synchronize components. Typically, clock signals are either a periodic square wave (Input A of Figure 4.1) or a periodic pulse waveform (Input B of Figure 4.1), for which the period is constant and the transition time between states is minimized. The period of the waveform is the time required to complete one full cycle. In Figure 4.1, Input A has a period of 1.0  $\mu$ Seconds and Input B has a period of 2.0  $\mu$ Seconds. Often the clock signal is specified not by its period, but by its frequency. *Frequency* is specified in units of cycles per second, which is usually called Hertz (Hz) after the 19th century physicist Heinrich Rudolf Hertz. Computer circuits have very high clock frequencies, so the terms Kilo-Hertz (KHz), Mega-Hertz (MHz), and Giga-Hertz (GHz) are used to specify units of thousand cycles per second, million cycles per second, and billion cycles per second respectively.

The frequency of the waveform is found by taking the reciprocal of its period. The frequencies of inputs A and B are determined to be 1.0 MHz and 500 KHz, as shown below:

$$\text{Frequency of A} = f_A = \frac{1}{\text{Period}} = \frac{1}{1.0 \times 10^{-6} \text{ Sec}} = 1.0 \times 10^6 \frac{\text{Cycles}}{\text{Second}} = 1.0 \text{ MHz}$$

$$\text{Frequency of B} = f_B = \frac{1}{\text{Period}} = \frac{1}{2.0 \times 10^{-6} \text{ Sec}} = 500 \times 10^3 \frac{\text{Cycles}}{\text{Second}} = 500 \text{ KHz}$$

## 4.1. Sequential Logic Devices

Sequential logic devices utilize an input signal called a *clock* input to trigger themselves at specific moments in time. The output of the device can only change when a trigger condition occurs. Trigger conditions can be either level sensitive or edge sensitive.

Level sensitive sequential components are called latches, and trigger in the one state. The D-latch is an example of a level triggered device.

Edge sensitive sequential components are called flip-flops. Flip-flops are either positive or negative edge triggered. A positive edge occurs when the clock signal changes from the '0' to '1' state, as shown in Figure 4.1 for Input A at times 0.5, 1.5, 2.5, 3.5, and 4.5 microseconds. A negative edge occurs when the clock changes from the '1' to '0' state, as shown in Figure 4.1 for Input A at times 1.0, 2.0, 3.0, and 4.0 microseconds. Flip-flop inputs are read at the instant a transition edge occurs. The output may change state the instant after the transition edge occurs.

### 4.1.1. D Flip-Flop and Latch

Transition tables describe what happens to the latch or flip-flop output after a clock trigger condition occurs. An example of a transition table is shown in Figure 4.2b. The output column of the transition table, denoted by  $Q^+$ , is the value of the output (Q) after some instant in time. The output can only change at the instant after a clock trigger condition has occurred; otherwise, the other inputs are ignored and the output cannot change.

The D flip-flop (Data flip-flop) can be used to store one bit of data. The D flip-flop has a data input (D), a clock input (C), and two outputs (Q and  $\bar{Q}$ ). The output  $\bar{Q}$  is always the opposite state of output Q. When a trigger edge occurs on the clock input, the state of output Q will become the state of input D. Therefore, the state of the D input is stored at output Q until the next clock trigger edge occurs. This relationship is depicted in the transition table of Figure 4.2b. Timing diagrams for both positive edge and negative edge triggered D flip-flops are illustrated in Figure 4.3.

D latches differ from a D flip-flops in that they are level triggered. The transition table for the D latch is the same as the D flip-flop. When the clock input is in the 1 state the Q output of a D latch will be the same state as the D input. When the clock input is in the 0 state the data at the output will be "latched" and will not change. A timing diagram for the D latch is illustrated in the last diagram of Figure 4.3.

Figure 4.2 D Flip-Flop (- Edge Triggered)

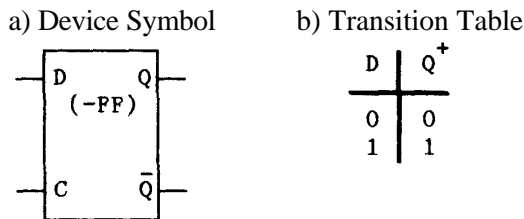
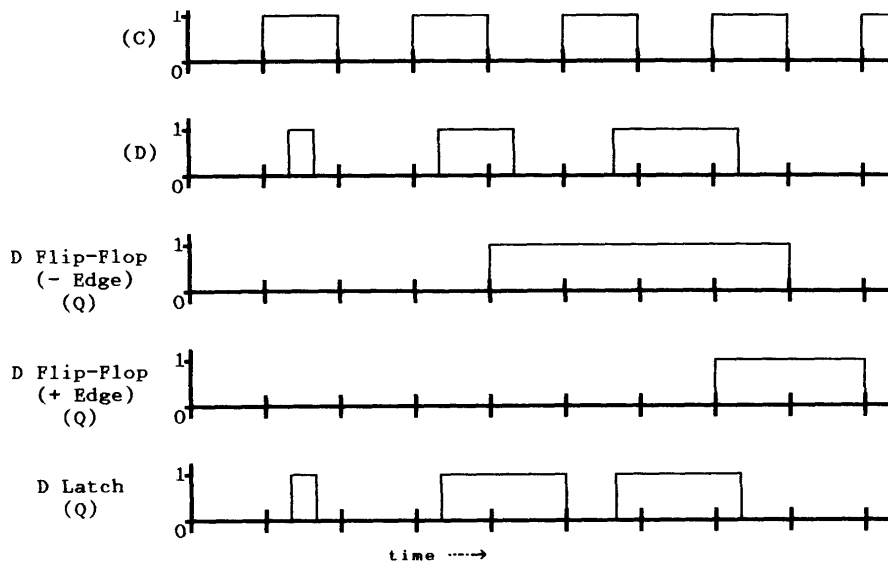


Figure 4.3 Timing Diagram for the D Flip-Flop and D Latch

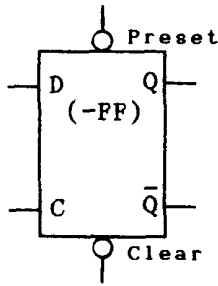


### 4.1.2. Preset And Clear Inputs

Flip-flops commonly have preset and clear inputs, as illustrated in Figure 4.4. These inputs are shown on the top and bottom of the flip-flop and are symbolized with an inversion circle by the labeled input. Sometimes these inputs are abbreviated such that  $\bar{R}$  = Clear and  $\bar{S}$  = Preset. Preset and clear inputs are *unclocked* negative logic inputs and take priority over all other inputs.

When the Preset or  $\bar{S}$  input is in the '0' state, the output Q is set to '1'. When the Clear or R input is in the '0' state, the output Q is cleared to '0'. At no time should both the clear and preset inputs be activated simultaneously, as the output state is then undefined.

Figure 4.4 D Flip-Flop with Preset and Clear Inputs



### 4.2. Timing Diagram Construction For Sequential Circuits

Timing diagrams have been constructed for single flip-flops and latches in Figures 4.1 and 4.3. Given timing diagrams for the inputs, the output timing diagram can be drawn for any sequential logic device. The procedure for making a timing diagram is outlined below:

Step 1: Determine the type of sequential logic device, including whether it is level triggered, or positive or negative edge triggered.

Step 2: Mark triggered clock edges (flip-flop) or trigger area (latches) where a transition of output may occur.

Step 3: Determine the state of all inputs to the sequential logic device just before a trigger condition occurs.

Step 4: Use the input values and the transition table for the given sequential logic device to determine the value of output Q after the trigger condition. Mark the state of the outputs on the timing diagram, until the next transition of the output can occur. Go back to Step 3.

This same procedure can be applied to construction of timing diagrams for sequential circuits with more than one sequential device. The procedure is applied by recognizing that the outputs of some sequential devices are connected to the inputs of other sequential devices. Therefore, after the timing diagram for an output of one device has been found, it may be used as an input to construct the timing diagram for the connecting device. This process is similar to the process used to evaluate combinational circuits using truth tables.

Verify the timing diagrams for single sequential logic devices of Figures 4.1 and 4.3. Then examine and verify the timing diagrams for the sequential circuits discussed in Section 4.3.

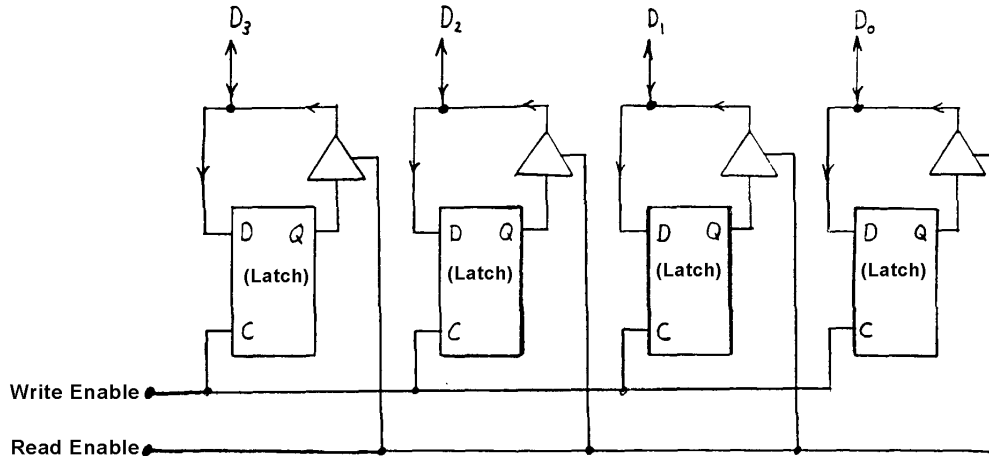
### 4.3. Sequential Circuits

Sequential circuits are constructed by interconnecting flip-flops, latches, and logic gates. Sequential circuits are used to perform many operations required by a computer. Discussed in this section are several simple sequential circuits that are found in computer hardware. These sequential circuits include frequency dividers, counters, data registers, shift registers, and data converters.

### 4.3.1. Data Registers

Data registers are used to store and retrieve binary data. Data registers are constructed using D-latches, as illustrated in Figure 4.5. A 4-bit data register can be used to store and retrieve four bits of data from a bi-directional data bus. In a store or write operation, data present on the data bus is stored in the D-latches by applying a trigger pulse at the clock inputs. In a retrieve or read operation, data is gated from the outputs of the latches through the tri-state buffers to the data bus. The read operation uses the bi-directional data bus as register outputs, and the write operation uses the data bus as register inputs.

Figure 4.5 Four Bit Data Register



### 4.3.2. Shift Registers

Shift Registers are a class of data registers that are used to shift a group of bits to the left or right. Shown in Figure 4.6 is a shift register, which will shift a group of four bits to the right.

The operation of the shift register can be best understood by examining and verifying the shift register timing diagram of Figure 4.6. The Reset input is used to initially clear all flip-flop outputs. The input value is then stored in the left-most flip-flop while its previous output value is stored in the next flip-flop to the right. The timing diagrams for the outputs illustrate the data shifting properties of the shift register. Notice the data at output  $Q_D$  is the same as the data on the input; however, it will be delayed close to four clock cycles. Shift registers can be used as a delay device with the maximum delay time equal to the number of flip-flops multiplied by the clock period.

### 4.3.3. Data Converters

Data can be transmitted one bit at a time or as a group of bits. When data is transmitted one bit at a time it is called serial data. Only one wire is required to transmit serial data. When several bits are transmitted simultaneously through parallel conductive paths, it is called parallel data transmission. A digital device that is used to perform conversions between parallel and serial data formats is called a data converter.

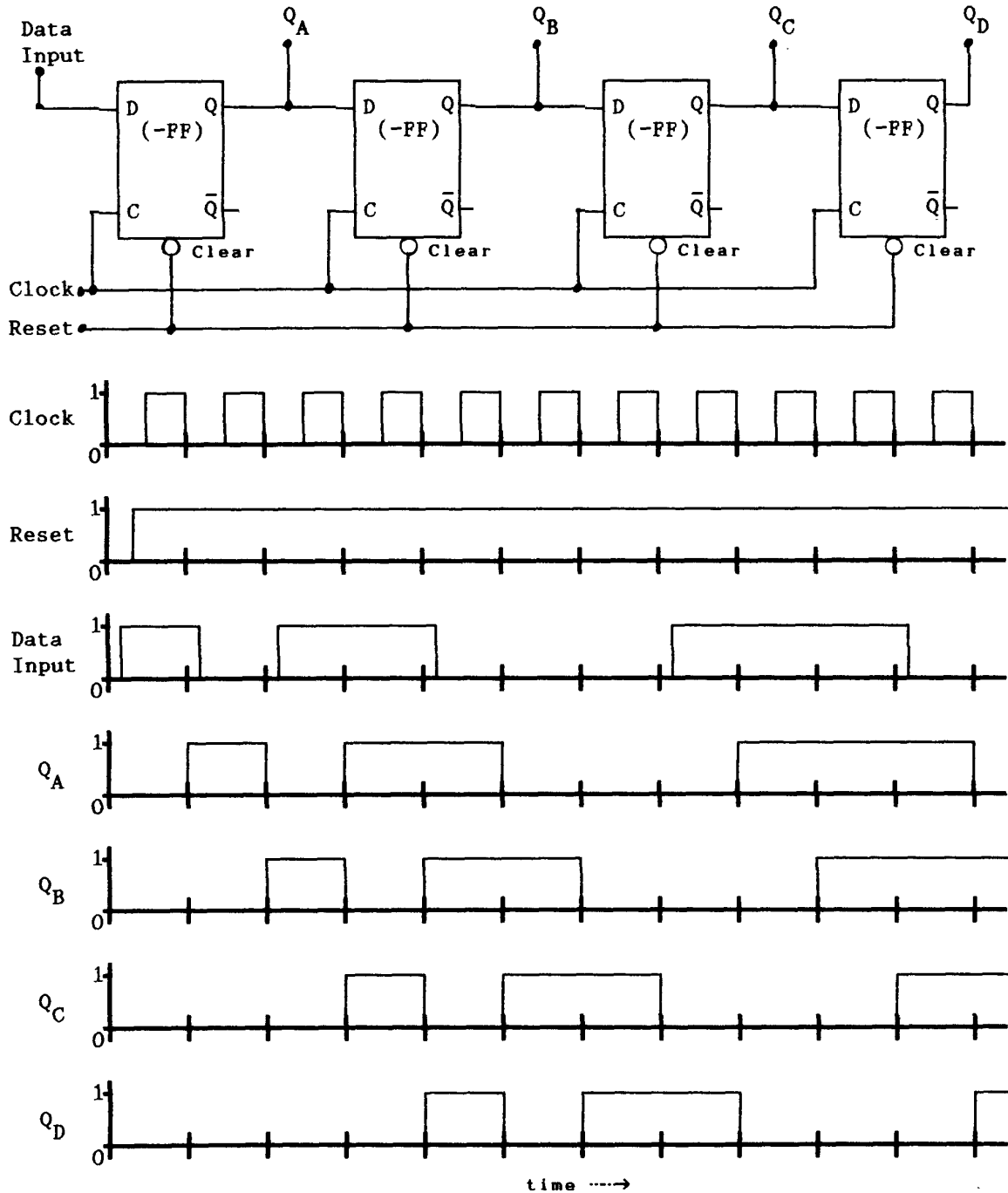
Generally, the number of bits per second of data that can be transmitted on a single conductive path is constrained by its conductive media (e.g. wire, printed circuit path, and fiber optic cable). When the required data transmission rate exceeds the maximum bit rate for the media, then parallel data transmission must be used instead of serial data transmission. Four bit parallel data can transmit four times as many bits per second as serial data for the same speed media. Of course, four parallel conductive paths would be required for parallel data, while only one is required for serial data transmission.

#### 4.3.3.1. Serial to Parallel Data Converter

The shift register of Figure 4.6 can be used as a serial to parallel converter. The single input shown is used as the serial input. The four flip-flop outputs are used as the four parallel outputs of the data converter. To read the parallel data properly, the data converter outputs must be read at one quarter the

frequency at which the data is loaded into the serial input. Data should only be read when the outputs  $Q_A$ ,  $Q_B$ ,  $Q_C$ , and  $Q_D$  are stable. This can be accomplished by reading the data on the fifth, ninth, and thirteenth positive edges of the clock.

Figure 4.6 Four Bit Shift Register



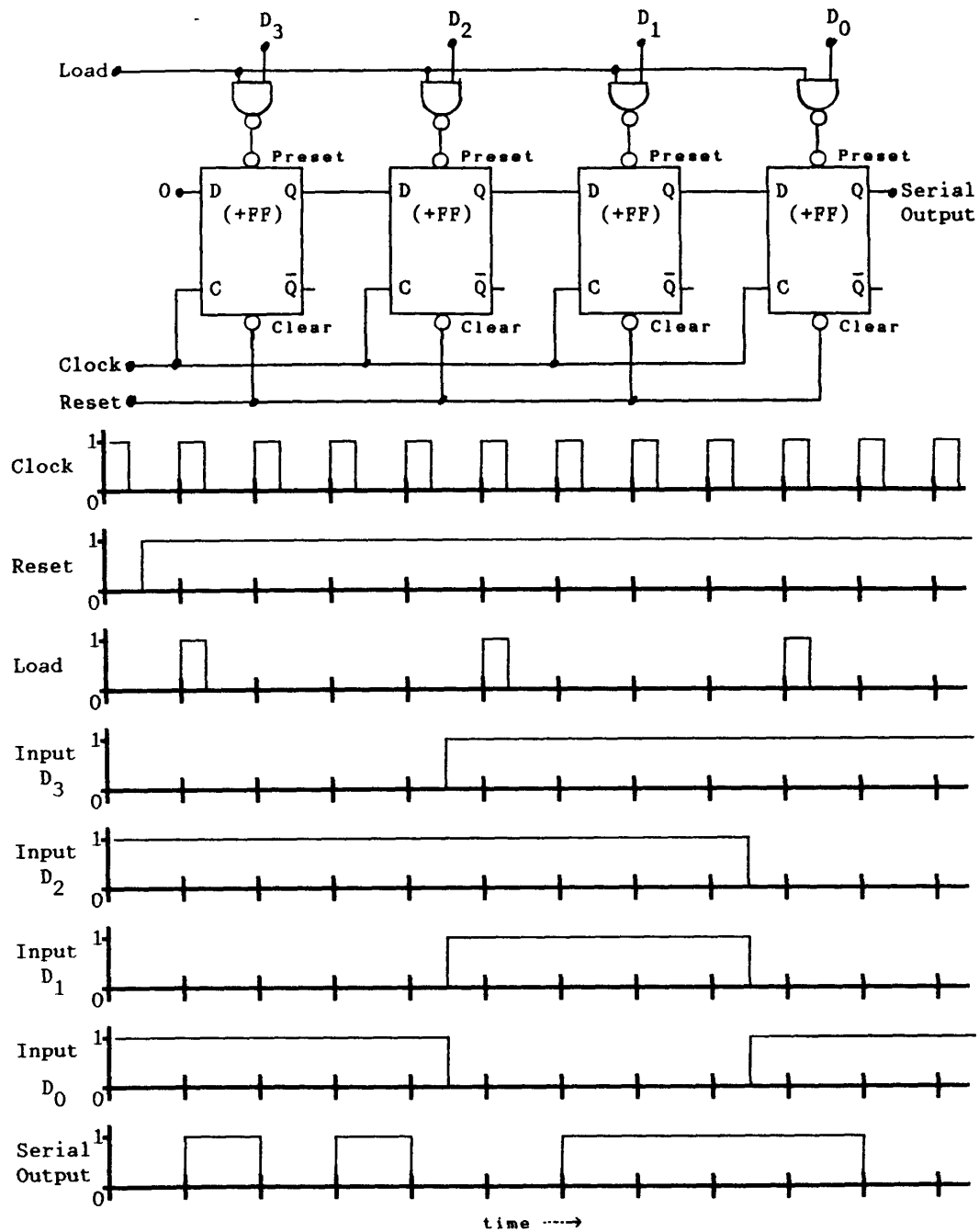
#### 4.3.3.2. Parallel to Serial Data Converter

A 4-bit parallel to serial data converter can be constructed using four D flip-flops and four NAND gates as shown in Figure 4.7. The serial data is transmitted at four times the frequency at which the parallel data is loaded. This can be verified by comparing the Clock input waveform with the Load input waveform.

The 4-bit parallel to serial converter is initially reset to generate a '0' state at the outputs of all flip-flops. The Load input is then pulsed high to load the flip-flops with the data from the parallel inputs ( $D_3$  through  $D_0$ ).

The Clock input to this synchronous circuit is a periodic pulse waveform, which is used to synchronize the parallel to serial data transfer. The flip-flops are positive edge triggered. Serial data should be read at the negative edges of the Clock input because the Serial output is stable at the negative edges.

Figure 4.7 Parallel to Serial Data Converter



## 4.4. Sequential Integrated Circuits

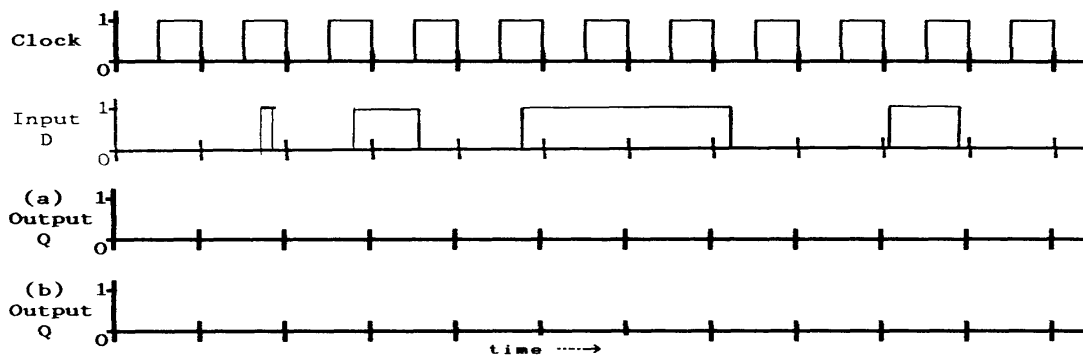
Flip-Flops and latches are manufactured as SSI circuit chips. The sequential circuits discussed in Section 4.4 are also available as MSI circuits. When designing a sequential circuit it is best to first determine which sequential functions can be performed using MSI chips and then use individual flip-flops (SSI chips) to complete the design. Descriptions of these integrated circuits can be found in a manufacturers data book, or downloaded from the Internet as a PDF document (e.g. <http://www.ti.com> , <http://www.national.com>)

### Problem Set

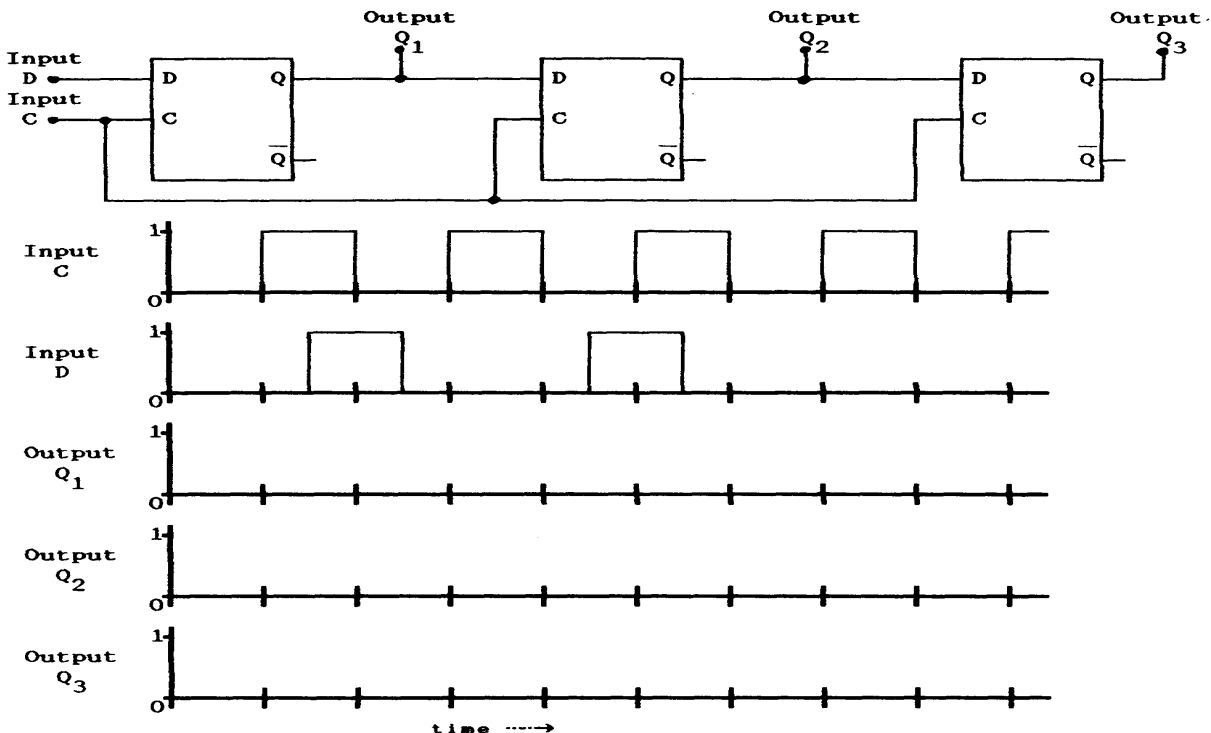
1) Complete the timing diagrams below for the following sequential logic devices. All devices are initially cleared at  $t=0$ .

a) D Latch (1 level triggered)

b) D Flip-Flop (negative edge triggered)



2) Complete the timing diagram below for outputs  $Q_1$ ,  $Q_2$ , and  $Q_3$ . The Flip-Flops are negative edge triggered D Flip-Flops and initially cleared at  $t=0$ .



## Chapter 5. Number Systems And Codes

---

The decimal number system (base 10) has become the standard number system used by people for counting and mathematical operations. The base ten system is used by most cultures primarily because people have 10 fingers. Each finger is used to represent one of ten possible values that a digit can assume.

Computers do not have 10 fingers. However, they are made up of electronic switches that represent Boolean variables in either a 1 or 0 state. For this reason, the binary (base 2) number system is used to represent the states of Boolean variables. A single binary digit is called a bit, which is in either a 1 or 0 state. A group of eight bits is called a byte. A half byte, which is a set of four bits, is often called a nibble.

Discussed in this chapter are the data formats used by computers to represent numbers and alphanumeric data. Also, binary addition is presented for both signed and unsigned numbers. The base of a number is denoted in this chapter by the subscript 2 for binary, 10 for decimal, and 16 for hexadecimal (base 16). In this text, commas are used with binary numbers to separate groups of four bits, which make the number more readable.

$$\begin{array}{l}
 \text{Hexadecimal (Base 16)} = 17_{16} \\
 \text{Decimal (Base 10)} = 23_{10} \\
 \text{Binary (Base 2)} = 0001,0111_2
 \end{array}
 \left. \vphantom{\begin{array}{l} \\ \\ \\ \end{array}} \right\} \text{These numbers are equivalent} \\
 \text{magnitudes in different bases}$$

### 5.1. Unsigned Binary Numbers

Binary numbers are base two numbers, which can be used to represent various integer quantities. The base two number system operates almost the same way as the decimal system; However, only two symbols (0 and 1) exist for each bit while ten symbols (0 through 9) exist for each digit of the decimal system.

Figure 5.1 compares the decimal and binary number systems. Both number systems are right justified. That is, the least significant bit (LSB) of a binary number is always the right-most bit, just as the least significant digit of a decimal number is the right-most digit. The next significant bit is always a power of two higher than the previous bit. The most significant bit (MSB) represents the highest power of two required for representing a number and is the left-most bit.

#### 5.1.1. Binary to Decimal Conversion

The four bit binary number of Figure 5.1 is easily converted to decimal notation. Simply sum the powers of two for all bits with a 1. Bits with a 0 are not added to this sum. Therefore, the conversion of  $1011_2$  to decimal is performed by the sum:

$$(1)2^3 + (0)2^2 + (1)2^1 + (1)2^0 = 8 + 2 + 1 = 11_{10}$$

Generally, the minimum data word length for microcomputers is 8 bits or one byte. To convert any binary number to decimal, determine the powers of two corresponding to each bit with a value of 1, and add up the appropriate magnitudes representing the power of two for each bit. For an eight-bit number represented by the eight bits  $b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0$  the least significant bit,  $b_0$ , represents the  $2^0$  or the 1's place and the most significant bit,  $b_7$ , represents the  $2^7$  or the 128's place. Conversion of an eight-bit number is represented by the following equation:

$$\begin{aligned}
 & b_7 2^7 + b_6 2^6 + b_5 2^5 + b_4 2^4 + b_3 2^3 + b_2 2^2 + b_1 2^1 + b_0 2^0 \\
 = & b_7 (128) + b_6 (64) + b_5 (32) + b_4 (16) + b_3 (8) + b_2 (4) + b_1 (2) + b_0 (1)
 \end{aligned}$$

The largest number that can be represented by eight bits would have a 1 for all eight bits  $b_7$  through  $b_0$ . This corresponds to  $1111,1111_2 = 255_{10}$ . The binary number  $0000,0000_2$  represent decimal zero. Therefore, eight bits may be used to represent any integer decimal number within the range of 0 to 255 inclusive. Additional bits are required to represent decimal integers greater than 255.

Figure 5.1 Decimal and Binary Number Systems

Humans use Base 10 (10 fingers)				Computers use Base 2 (On/Off)			
10 Symbols: 0-9				2 Symbols: 0-1			
Most Significant Digit (MSD)		Least Significant Digit (LSD)		Most Significant Bit (MSB)		Least Significant Bit (LSB)	
6 4		7 2 <sub>10</sub>		1 0		1 1 <sub>2</sub>	
Powers of 10				Powers of 2			
MSD			LSD	MSB			LSD
$10^3$	$10^2$	$10^1$	$10^0$	$2^3$	$2^2$	$2^1$	$2^0$
1000	100	10	1	8	4	2	1
$6 \times 1000 = 6000$ $4 \times 100 = 400$ $7 \times 10 = 70$ $2 \times 1 = 2$				$1 \times 8 = 8$ $0 \times 4 = 0$ $1 \times 2 = 2$ $1 \times 1 = 1$			
6472 <sub>10</sub>				11 <sub>10</sub>			

### 5.1.2. Decimal to Binary Conversion

To convert a decimal number to a binary number is more tedious than binary to decimal conversion. Tables such as Figure 5.2 are often used to facilitate these conversions. As an alternative to tables, a direct mathematical procedure is shown in Example 5.1. Consider the decimal number 19. Conversion is performed using successive divisions by 2. First 19 is divided by 2 which will generate a quotient of 9 and remainder of 1. Next divide the preceding quotient 9 by 2, which will generate the next quotient of 4 and remainder of 1.

**Example 5.1:** Convert 19 to Binary

	Q	R	
$19 \div 2 = 9$	1	(LSB)	Least Significant Bit
$9 \div 2 = 4$	1		
$4 \div 2 = 2$	0		
$2 \div 2 = 1$	0		
$1 \div 2 = 0$	1	(MSB)	Most Significant Bit

Therefore  $19_{10} = 10011_2 = 0001,0011_2$

Continue the division by 2 until a quotient of 0 exists. The remainder column represents the binary equivalent number. The final remainder is the most significant bit and the first remainder is the least significant bit. Therefore, 19 decimal is represented by the binary number 10011. Binary numbers are often zero filled to eight bits resulting in 0001,0011 to represent decimal 19.

Figure 5.2 Unsigned Decimal to Binary Conversions

Decimal	Binary	Decimal	Binary
0	00000000	32	00100000
1	00000001	33	00100001
2	00000010	34	00100010
3	00000011	...	
4	00000100	62	00111110
5	00000101	63	00111111
6	00000110	64	01000000
7	00000111	65	01000001
8	00001000	66	01000010
9	00001001	...	
10	00001010	126	01111110
11	00001011	127	01111111
12	00001100	128	10000000
13	00001101	129	10000001
14	00001110	130	10000010
15	00001111	...	
16	00010000	252	11111100
17	00010001	253	11111101
...		254	11111110
31	00011111	255	11111111

## 5.2. Signed Binary Numbers

Signed binary number representations are used to represent both positive and negative numbers. They also allow signed binary addition and subtraction operations, which may yield negative results.

To utilize the same full adder circuit for both signed and unsigned binary numbers the *2's complement* data format is utilized to represent signed binary numbers. When referring to signed binary numbers, the 2's complement representation will always be used in this text.

An abbreviated table of 8-bit 2's complement numbers is shown in Figure 5.3. For positive numbers within the range of  $127_{10}$  through  $0_{10}$ , the 2's complement representation is identical to the unsigned binary format.

However, for negative numbers a conversion procedure is required. Conversion is performed for 2's complement negative numbers using the following three steps:

- Step 1) Determine the unsigned binary number magnitude
- Step 2) Complement (invert) the state of each bit
- Step 3) Add 1 to the result

Example 5.2 illustrates this 2's complement conversion procedure using four examples. This procedure is only used for converting negative numbers to 2's complement form. Again, positive numbers will have the same form for both unsigned binary and 2's complement representations.

This same three-step procedure can also be used to change the sign of the number. This is useful when converting a negative 2's complement number to decimal and for the subtraction operation. Subtraction can be performed on 2's complement numbers by first performing the three-step 2's complement procedure described to change the sign of the subtrahend; Then, add the two numbers together. By using this procedure to perform subtraction, no additional subtraction hardware is required to perform the subtraction operation on two 2's complement numbers.

The range of 8-bit 2's complement numbers is  $+127$  to  $-128$  inclusive, as shown in Figure 5.3. The most significant bit is the sign bit. If  $b_7 = 0$ , the number is positive, and if  $b_7 = 1$  the number is negative.



exceeds 1, a carry is generated into the next significant bit. The carry out of the most significant bit represents the carry flag. If this carry flag is in the 1 state and the numbers are unsigned binary, then the sum exceeds what can be shown with 8 bits. When a carry out exists from bit 6 with no carry out from bit 7, or vice versa, an improper sign change has occurred for the sum in 2's complement form and an overflow condition exists. When this overflow condition is true the resulting 2's complement sum is invalid.

Addition is performed using the same method for both unsigned and signed 2's complement numbers. The only differences are the conditions dictating the validity of the sums. Therefore, it is up to the computer programmer to keep track of whether the data is in signed or unsigned formats and to check the appropriate carry or overflow conditions to determine if the sums are valid.

Binary addition of both unsigned and 2's complement signed numbers can be performed using the full adder hardware of Figure 5.4. This figure is similar to Figure 2.14 with the exception that eight full adder circuits are cascaded together so that addition can be performed on 8-bit data and both a Carry flag (C) and Overflow flag (V) are available for examination. The C and V flags are often referred to as the carry and overflow condition codes in computer literature.

The carry flag is actually the carry out from the full adder of the most significant bit. The carry flag is considered only when performing unsigned binary addition. When the carry flag is a '0' it specifies that the resulting unsigned binary sum is within the range of valid binary numbers 0 through 255. A carry flag of one specifies that the resulting sum is greater than 255 and the result is not valid. The overflow flag is meaningless when performing unsigned binary addition.

Addition of signed 2's complement numbers is also performed using the circuitry of Figure 5.4. When performing 2's complement arithmetic the carry flag is meaningless and only the overflow flag specifies whether the resulting sum is within the valid range from -128 through +127. The overflow flag is generated by exclusive-OR'ing the carry out from bit 7 with the carry out of bit 6. If one, but not both, of these carry outs are a '1' it means that an improper sign change has occurred and the resulting sum is not valid. Therefore, if the overflow flag is 0, the 2's complement sum is valid; An overflow flag of 1 signifies that the sum is out of range (-128 through +127).

### Example 5.3: Binary Addition

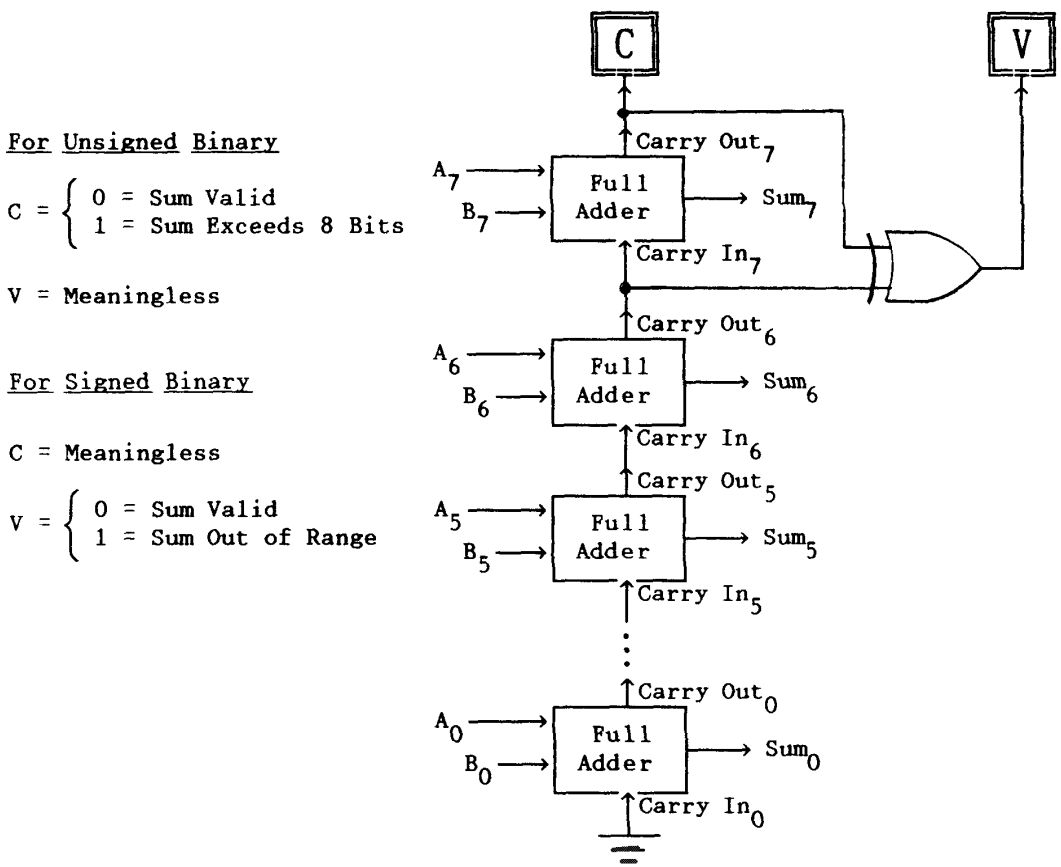
	Unsigned	Signed		Unsigned	Signed	
Carry				11		
A =	0001,0010	18	+18	0100,0011	67	+67
B =	0000,0101	5	+5	1000,0011	131	-125
Sum =	0001,0111	23	+23	1100,0110	198	-58
	Valid	Valid		Valid	Valid	
	C=0	V=0		C=0	V=0	

	Unsigned	Signed		Unsigned	Signed	
Carry				1111 111		
A =	0111,1101	125	+125	0000,0111	7	+7
B =	0010,0101	37	+37	1111,1101	253	-3
Sum =	1010,0010	162	+162	0000,0100	260	+4
	Valid	>+127		>255	Valid	
	C=0	V=1		C=1	V=0	

## 5.4. Binary Number Magnitude

The number of bits used for the data word restricts the magnitude of decimal numbers that can be represented by binary numbers. Binary numbers with eight bits can represent unsigned numbers in the range from 255 through 0, or signed numbers in the range from +127 through -128. Numbers outside of these ranges cannot be represented unless additional bits are used to increase the data word size.

Figure 5.4 8-Bit Adder Circuit With Overflow and Carry Flags



Binary numbers can represent a maximum of  $2^n$  decimal numbers, where  $n$  is the number of bits of the data word. For unsigned numbers the range begins at zero. For signed binary numbers the center of the range is zero.

Consider the case when the data word size is increased from one byte to two bytes (16 bits). A two-byte binary number can be used to represent 65,536 different decimal numbers. For unsigned binary numbers, 16 bits would represent decimal numbers in the range of 65,535 to 0. The signed 2's complement range is from +32,767 to -32,768. Once again, zero is considered a positive number. Therefore, the range of negative numbers appears to be one more than positive numbers.

Computers have a standard word size in which all data is represented as some multiple of eight bits (i.e. 8, 16, 32, 64).

## 5.5. Floating Point Representations

Signed integer numbers are represented using the 2's complement format of Section 5.2. Integer numbers are whole numbers. An alternate approach is required to represent real numbers.

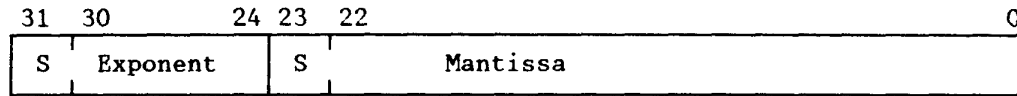
The floating-point representation is used to represent real numbers in much the same way as scientific notation. Consider the number -0.0004772, which can be represented in scientific notation as  $-0.4772 \times 10^{-3}$ . The mantissa is normalized to be a number between  $\pm 0.9999$  and  $\pm 0.1000$ . The exponent -3 is a signed integer quantity representing the power of ten that the mantissa is multiplied. The actual bit format used to represent floating point numbers varies dependent on the data word size of the computer.

Figure 5.6 shows a typical 32-bit format used to represent floating point numbers. For example both the mantissa and exponent can be represented using 2's complement numbers. In both cases a sign bit of 0 represents a positive number and a sign bit of 1 represents a negative number. The number of bits in the exponent determines the range of powers of ten. An 8 bit exponent can represent powers of ten from

$10^{+127}$  to  $10^{-128}$ . A 24-bit mantissa can represent signed decimal numbers of six significant digits for the range +999,999 to !999,999.

If more significant digits are required for an application, allocating more bits to the mantissa can increase the mantissa size. Most computer programming languages allow a higher precision (more significant digits) floating point number representation that is often called double precision floating point numbers.

Figure 5.5 Floating Point Number Representation



## 5.6. Hexadecimal Numbers

Representing eight bits of data as a string of 1's and 0's can be tedious.

The hexadecimal number system is used to simplify data representation by encoding 4 bits as one symbol. Hexadecimal numbers comprise the base 16 number system. The hexadecimal number system has sixteen different symbols, which represent the value of each hexadecimal digit. As shown in the conversion table of Figure 5.7a, 0 through 9 are used to represent the first ten hexadecimal symbols, and letters A through F are used to represent the last six symbols. Each hexadecimal symbol represents one of the sixteen possible combinations of four bits. Therefore, eight bits of data is more easily represented as two hexadecimal digits.

### 5.6.1. Binary to Hexadecimal Conversion

To convert a binary number to hexadecimal is quite easy. Starting from the least significant bit ( $b_0$ ), group the binary bits into groups of four. Use the table of Figure 5.7a to determine the hexadecimal symbol that represents each group of four bits.

For example:

$$1000,1100,0111,1010_2 = 8 \quad C \quad 7 \quad A_{16}$$

Figure 5.6 Hexadecimal Numbers

a)

Base 16	Base 2	Base 10
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
A	1010	10
B	1011	11
C	1100	12
D	1101	13
E	1110	14
F	1111	15

b)

Base 16 can represent 4 bits

16 Symbols: 0,1,2,3,4,5,6,7,  
8,9,A,B,C,D,E,F

A      5      2      F<sub>16</sub>

Most Significant      Least Significant  
Digit (MSD)                      Digit (LSD)

Powers of 16

MSD			LSD
$16^3$	$16^2$	$16^1$	$16^0$
4096	256	16	1

(A=)    10 x 4096 = 40,960  
 (5=)    5 x 256 = 1,280  
 (2=)    2 x 16 = 32  
 (F=)    15 x 1 = 15

42,287<sub>10</sub>

### 5.6.2. Hexadecimal to Binary Conversion

Converting from hexadecimal to binary is also quite simple. Use the table of Figure 5.7a to determine the group of four bits, which represents each digit of the hexadecimal number.

For example:

$$A \quad 5 \quad 2 \quad F_{16} = 1010,0101,0010,1111_2$$

### 5.6.3. Hexadecimal to Decimal Conversion

An example of hexadecimal to decimal conversion is shown in Figure 5.7b, for hexadecimal number A52F. To convert any hexadecimal number to decimal, determine the powers of sixteen representing each digit and multiply the number in each digit by the power of sixteen associated with that digit. Then add up the products of each digit to determine the equivalent decimal number.

Conversion of a four digit hexadecimal number,  $h_3 h_2 h_1 h_0$ , to decimal is represented by the following equation:

$$h_3 16^3 + h_2 16^2 + h_1 16^1 + h_0 16^0 = h_3 (4096) + h_2 (256) + h_1 (16) + h_0 (1)$$

### 5.6.4. Decimal to Hexadecimal Conversion

To convert from a decimal number to a hexadecimal number is more difficult.

One method is similar to the decimal to binary conversion procedure discussed in Example 2.1. The procedure for converting from a decimal number to a hexadecimal number is shown in Example 5.5. Consider the decimal number 42,287. Conversion is performed using successive divisions by 16. First divide 42,287 by 16 which will generate a quotient of 2642 and remainder of 15.

Next divide the preceding quotient 2642 by 16, which will generate the next quotient of 165 and remainder of 2. Continue the division by 16 until a quotient of 0 exists. The hexadecimal equivalent is determined by examining the remainder column and taking the final remainder as the most significant hexadecimal digit and the first remainder as the least significant digit.

Remainders in the range of 10 to 15 would have to be converted to their hexadecimal symbols described in the table of Figure 5.7a. Therefore, 42,287 decimal is equivalent to the hexadecimal number A52F.

**Example 5.5:** Convert 42,287 Decimal to Hexadecimal

	Quotient	Remainder	
42,287 ÷ 16 =	2642	15 = F	← LSD = Least Significant Digit
2,642 ÷ 16 =	165	2 = 2	
165 ÷ 16 =	10	5 = 5	
10 ÷ 16 =	0	10 = A	← MSD = Most Significant Digit

$$42,287_{10} = A52F_{16}$$

### 5.6.5. Hexadecimal Addition

Hexadecimal addition is similar to decimal addition except digits are incremented up to F (15<sub>10</sub>) before generating a carry into the next significant digit. This is demonstrated with the following examples:

$$\begin{array}{r} 5A_{16} \\ + 36_{16} \\ \hline 90_{16} \end{array}$$

$$\begin{array}{r} 52_{16} \\ + AC_{16} \\ \hline FE_{16} \end{array}$$

$$\begin{array}{r} 29_{16} \\ + 28_{16} \\ \hline 51_{16} \end{array}$$

## 5.7. Alphanumeric Data Representation

People use written language to communicate among themselves and to give instructions to a computer in the form user keyboard input. Alphanumeric data is represented by a binary code for each letter, number, and symbol commonly associated with a typewriter keyboard. The ASCII (American Standard Code for Information Interchange) Code is the most commonly used representation for alphanumeric data. Figure 5.8 presents a table used to convert from either hexadecimal or binary codes to the ASCII characters represented by these codes. All upper and lower case letters, numbers, and symbols used in the English language are in column 2 through 7. Special computer control characters are found in columns 0 and 1. A definition of each of these control characters is located below the conversion table.

An ASCII character is represented by a byte, with bit 7 (MSB) being the parity bit and bits 6 through 0 determined by the conversion table. The parity bit is used for error detection when transmitting and receiving data. Both the transmitter and receiver must be setup to transfer data with the same parity. The parity bit value is determined by the type of parity selected and the state of bits 6 through 0. Four types of parity exist as described in the table below. However, in most applications the parity bit is simply set to '0'.

0 Parity	Bit 7 set to 0 (Default)
1 Parity	Bit 7 set to 1
Even Parity	Bit 7 set to generate an even number of 1's for the byte
Odd Parity	Bit 7 set to generate an odd number of 1's for the byte

### 5.7.1. Binary String to ASCII Character Conversion

Given a string of binary bits or hexadecimal numbers, conversion to the ASCII characters is performed by separating the string into individual bytes. Again, it is assumed that the parity bit is equal to 0.

To convert bits 6 through 0 to ASCII, use the ASCII conversion table of Figure 5.8. The least significant nibble, bits 3 through 0, represents the rows of the conversion table. Entries are listed in both binary bits and hexadecimal digits. Bits 6 through 4 represent the columns of the table. Once the specific column and row are located, the ASCII character defined by the byte is found at the specified row and column. Continue this conversion process for each byte of the string. This procedure is illustrated in the example below.

As an example, convert the following hexadecimal representation of a binary string with '0' parity to ASCII characters.

```
Hexadecimal Representation = 576861743F
      57      68      61      74      3F
      0101,0111  0110,1000  0110,0001  0111,0100  0011,1111
Convert using ASCII Conversion Table gives you the following answer:
W      h      a      t      ?
= What?
```

### 5.7.2. ASCII Character to Binary String Conversion

The conversion of a string of characters to a string of bits or hexadecimal digits is performed using the reverse process of the above. Find the ASCII character in the ASCII Conversion table. Determine the byte representing this ASCII character by first determining the row of the character in the table. This specifies the least significant nibble, bits 3 through 0. Then determine the column of the character, which specifies bits 6 through 4. Bit 7 is the parity bit and assumed to be 0. Once all eight bits of the ASCII character byte are determined, they can be converted to hexadecimal, if desired. This procedure is illustrated below.

ASCII Character String = What?

W            h            a            t            ?

Conversion    101,0111 110,1000 110,0001 111,0100 011,1111

Parity Added  0101,0111 0110,1000 0110,0001 0111,0100 0011,1111

Hexadecimal    57            68            61            74            3F

Figure 5.7 ASCII Conversion Table

HEX	MSD	0	1	2	3	4	5	6	7
LSD	BINARY	000	001	010	011	100	101	110	111
0	0000	NUL	DLE	SPACE	0	@	P	'	p
1	0001	SOH	DC1	!	1	A	Q	a	q
2	0010	STX	DC2	"	2	B	R	b	r
3	0011	ETX	DC3	#	3	C	S	c	s
4	0100	EOT	DC4	\$	4	D	T	d	t
5	0101	ENQ	NAK	%	5	E	U	e	u
6	0110	ACK	SYN	&	6	F	V	f	v
7	0111	BEL	ETB	'	7	G	W	g	w
8	1000	BS	CAN	(	8	H	X	h	x
9	1001	HT	EM	)	9	I	Y	i	y
A	1010	LF	SUB	*	:	J	Z	j	z
B	1011	VT	ESC	+	;	K	[	k	{
C	1100	FF	FS	,	<	L	\	l	
D	1101	CR	GS	-	=	M	]	m	}
E	1110	SO	RS	.	>	N	^	n	~
F	1111	SI	US	/	?	O	_	o	DEL

**Control Characters:**

NUL	Null	VT	Vertical	SYN	Synchronous Idle
SOH	Start of Heading	FF	Form Feed	ETB	End Transmission Block
STX	Start of Text	CR	Carriage Return	CAN	Cancel
ETX	End of Text	SO	Shift Out	EM	End of Medium
EOT	End of Transmission	SI	Shift In	SUB	Substitute
ENQ	Enquiry	DLE	Data Link	ESC	Escape
ACK	Acknowledge	DC1	Device Control	FS	File Separator
BEL	Bell	DC2	Device Control	GS	Group Separator
BS	Backspace	DC3	Device Control	RS	Record Separator
HT	Horizontal	DC4	Device Control	US	Unit Separator
LF	Line Feed	NAK	Negative	DEL	Delete

## Problem Set

- Perform the following decimal to binary conversions. Verify your answers by performing binary to decimal conversions.
 

a) 25	b) 31	c) 173	d) 250
e) 320	f) -19	g) -102	h) -185
- Perform the following binary additions on one byte numbers. Determine the state of the carry flag [C] and the overflow flag [V] after the addition operation.
 

a) 0010,0110	b) 0110,0010	c) 1010,0111	d) 1010,1010
<u>+0001,1011</u>	<u>+0101,0001</u>	<u>+0110,0011</u>	<u>+1100,0011</u>
- Determine the decimal equivalents for the numbers of problem 2 for both binary signed and unsigned number representations. Verify that the resulting sum is correct by also determining the decimal equivalents.
- When performing binary addition what is the significance of the overflow flag and carry flags?
- Convert the following hexadecimal numbers to binary. Verify your answers by converting the binary result back to hexadecimal.
 

a) 87	b) 23	c) CF	d) 73B
-------	-------	-------	--------
- Convert the hexadecimal numbers of Problem 5 to decimal. Verify your answers by converting the decimal result back to binary.
- Write the hexadecimal string for the following ASCII character strings. Assume the parity bit is '0'.
 

a) What If?
b) PSC 561
c) Japan
d) 49931
- Convert the following hexadecimal string to ASCII characters.  
4A 46 53 4D 33 31 30 20 49 73 20 46 75 6E 21

## Chapter 6. Computer Architecture

---

Computer components are usually classified in one of three major areas: the central processor unit (CPU), memory, and input/output (I/O). All computers contain elements in each of these three areas. Computer architecture describes the structure and function of computer elements in a computer system.

The central processor unit contains all registers, bus control hardware, the arithmetic logic unit, and the control unit. The central processor unit is used to process data and to control most computer operations. The CPU executes a computer program consisting of machine code instructions. Central processor units are classified by their package and data word size. A central processor unit, contained in a single integrated circuit package, is called a microprocessor. There are many different microprocessors available from electronics manufacturers. The most common have data words of 8, 16, 32 and 64 bits.

Memory is a device that is used to store and retrieve words of data. The data can represent machine code instructions of a program, numerical data, or ASCII characters. Memory is specified by both the number of memory locations and the word size. The notation [Number of words] x [Word size] is usually used when referring to memory. For example, a memory with 256 memory locations and a 4-bit data word size is specified as a 256 x 4-bit memory.

Each memory location has an address, which is represented by a unique binary code. Shown in Figure 6.1, is a representation of memory with 8-bit data words. Each memory location of 8-bits, has a 16 bit address which is shown in parenthesis. Often, hexadecimal numbers are used to represent binary data and addresses. Each hexadecimal digit represents four bits of a binary number.

Input/output refers to those elements used to receive and transmit data between the central processor unit and peripheral devices. Peripheral devices may be in the form of a keyboard, terminal display, floppy disk drive, or printer. Input/output consists of an interface device that is used to communicate with peripherals through an I/O port. The interface device connects to the microprocessor through computer buses.

### 6.1. COMPUTER BUSES

Computer buses are conductive circuit paths that interconnect the central processor unit, memory, and input/output devices. Three buses are commonly used for interconnection of computer components: the data bus, the address bus, and the control bus.

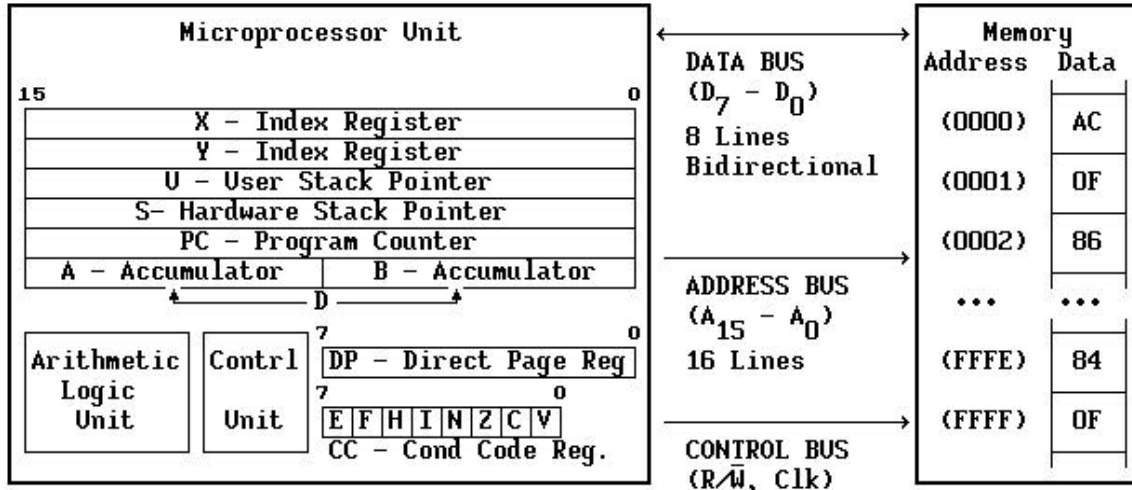
The data bus is a bi-directional bus as indicated by the double headed arrow in Figure 6.1. The data bus is used by the microprocessor to either read data from memory or write data to memory. The data bus usually contains the same number of conductive paths as the data word size of the central processor unit. For example the data bus for an 8-bit microprocessor is typically eight bits wide. For a 32-bit microprocessor, the data bus is 32 bits wide.

The address bus is an output from the microprocessor, and an input to memory and interface devices. The microprocessor selects a specific memory location or interface device by placing a binary number on the address bus. This binary number represents the address of a specific memory location or interface device. As shown in Figure 6.1 the address bus of the 8-bit microprocessor is 16 bits wide. Each possible combination of sixteen bits on the address bus specifies a different address. Therefore, a total of  $2^{16} = 65,536$  unique addresses can be specified by this microprocessor. A microprocessor with  $n$  address bus lines can specify  $2^n$  unique addresses.

The control bus is used to control and synchronize data transmission between the microprocessor, memory, and input/output. The control bus varies for different applications and microprocessors; However, it will always contain a Read/Write (R/W) line, and a Clock (Clk) line. The read/write line is an output from the microprocessor. It is used to specify the direction of data transmission on the data bus. If the R/W line is in the one state, a memory read operation is performed in which data is read from memory by the microprocessor. If the R/W line is in the zero state, data is written to memory from the microprocessor. This is denoted by the inversion bar over the W. The Clock line is used to synchronize data transfer operations. The Clock signal specifies the time interval when the address and data buses are

stable, and when data may be read from or written to memory. The Clock signal of Figure 6.1 is an output of the microprocessor, and an input to all other devices. It is derived from a quartz crystal of known frequency that is connected to the microprocessor crystal inputs.

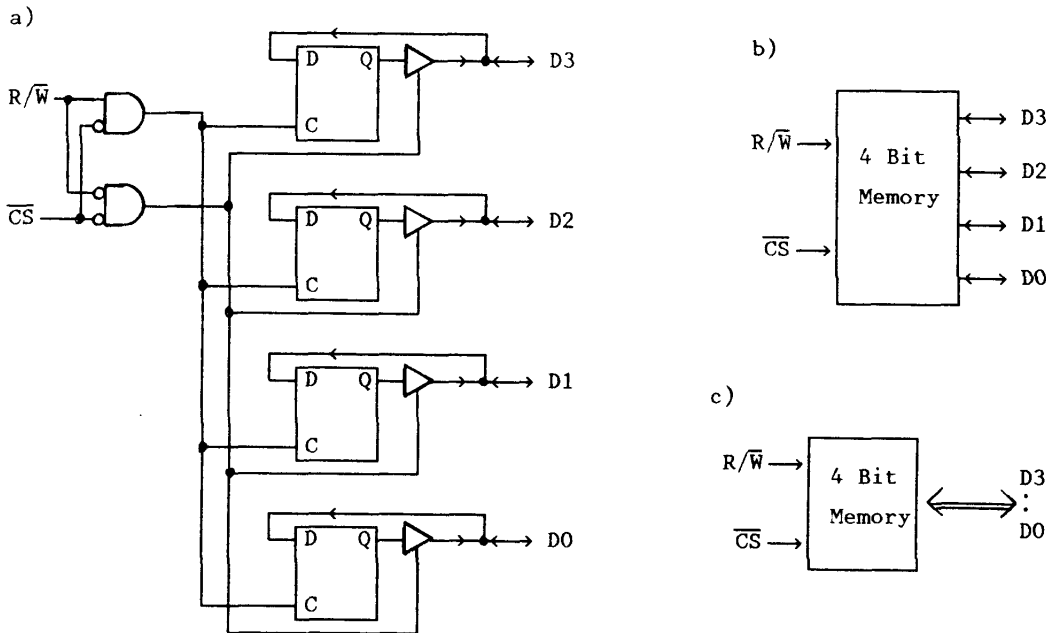
Figure 6.1 8-Bit Microprocessor Unit and Memory Bus Connections



## 6.2. MEMORY

Computer memory is accessed by the microprocessor through the buses and is used to store and retrieve data at specific addresses or memory locations. The number of bits of memory storage at each address is determined by the microprocessor used. An 8-bit microprocessor will require 8 bits of data storage per address. A 32-bit microprocessor would require thirty-two bits of data storage per address.

Figure 6.2: Four Bit Memory, a) Digital Circuit b) Block Diagram c) Block Diagram



Computer memory uses the D-latch as the fundamental sequential logic component. Each D-latch can be used to store one bit of information. A four-bit memory can be constructed using four D-latches as shown in Figure 5.2a. Bi-directional lines D0 through D3 are used to transmit and receive data from the data bus. A memory write operation occurs when both the R/W and CS (Chip Select) inputs are in the low

state. During a memory write operation, data is received from the data bus and stored at the D latch outputs. A memory read operation occurs if the R/W input is in the high state and the CS (Chip Select) input is in the low state. During a memory read operation, data is transmitted to the data bus from the D-latch outputs by gating it through tri-state buffers as illustrated in Figure 6.2a. Many outputs are connected to a single line of the data bus; therefore, tri-state buffers must be used.

Block diagram representations of a one word by 4-bit memory are shown in Figures 6.2b and 6.2c. Note that the same inputs and outputs exist on the block diagrams as in the digital circuit. The block diagram of Figure 6.2b shows the bi-directional data bus lines separately, while Figure 6.6c shows the data bus as a double line.

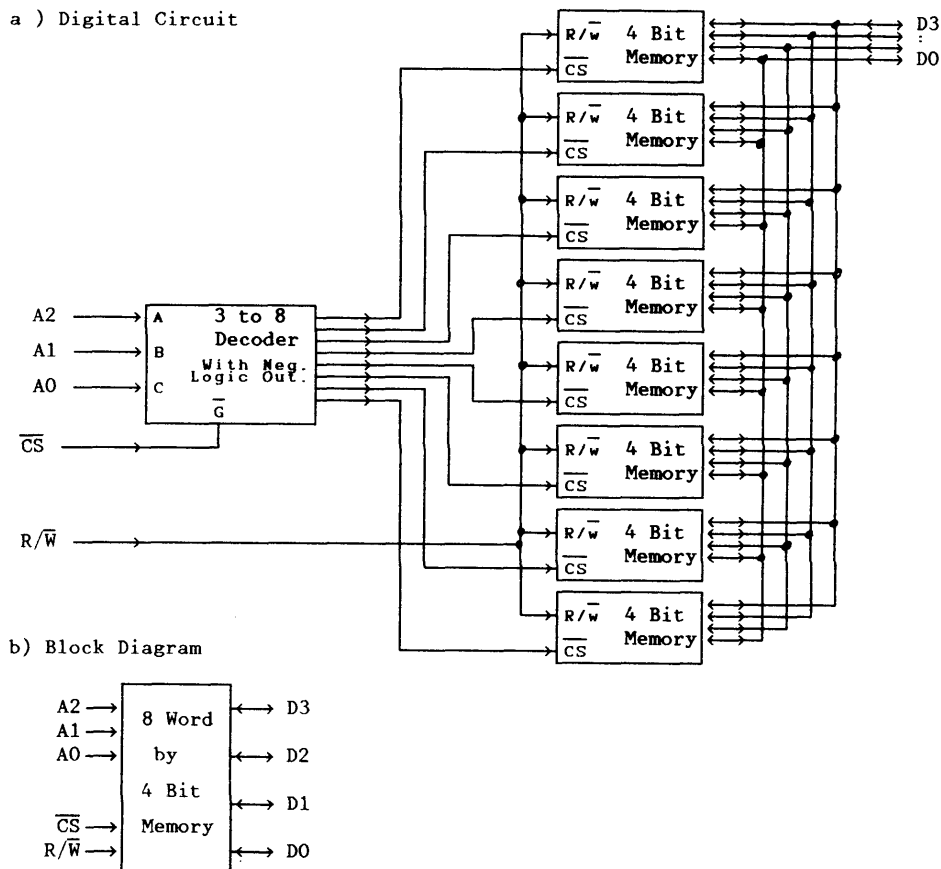
### 6.2.1. Address Decoding

Computer memory is configured so that one of many words of data can be accessed by the microprocessor through the three buses. A word of data in memory is selected by the microprocessor by transmitting its address on the address bus. Decoders are used with computer memory to decode this address and select the specified memory location.

Illustrated in Figure 6.3, is an eight word by 4-bit memory. Consider each of the eight, 4-bit memories to be the module of Figure 6.2c. These modules are interconnected with data bus lines D0 through D3 and the R/W input going to each module. The CS inputs from each module are connected to the outputs of the 3 to 8 decoder with negative logic outputs. (i.e. selected output in low state while all other outputs in high state). Eight possible combinations exist (i.e. 000 through 111) for the three select inputs of the 3 to 8 decoder. Each combination represents an address of one of the eight 4-bit words. This 3 bit code comes from the three least significant bits of the address bus; therefore, the address bus lines A0, A1, and A2 are connected to the decoder select inputs A, B, and C.

Figure 6.3: Eight Word by Four Bit (8 word x 4 bit) Memory

a) Digital Circuit b) Block Diagram



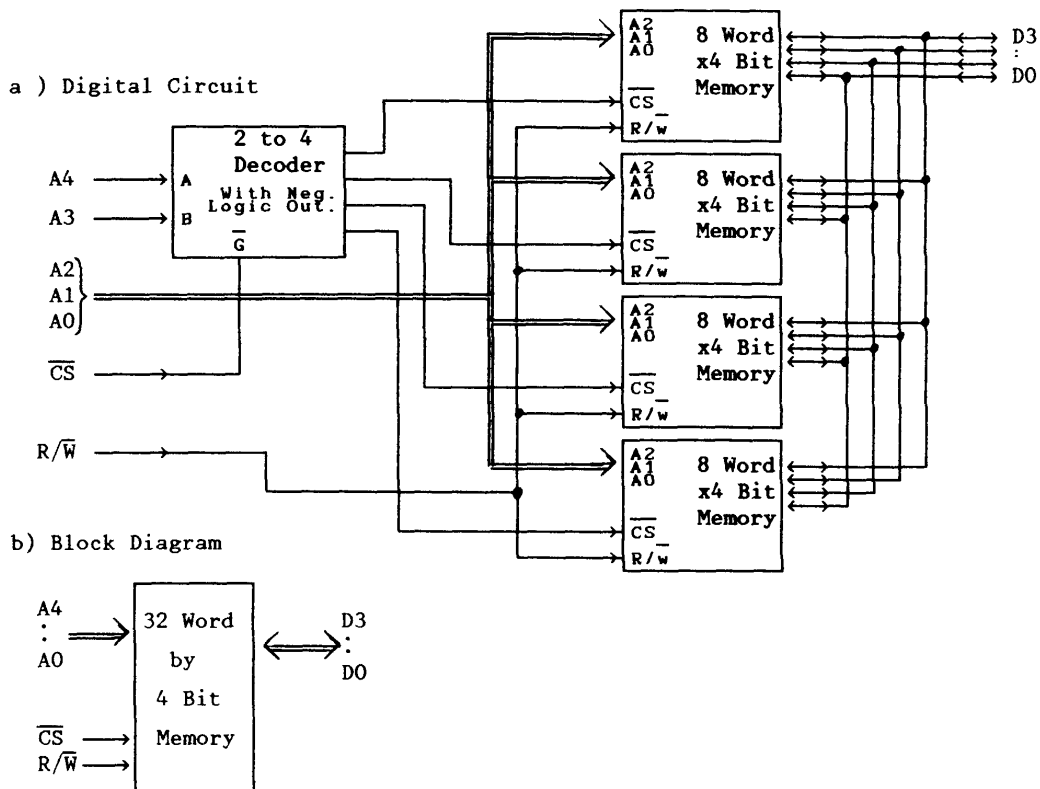
Decoders commonly have gate inputs, which will disable all outputs. In Figure 6.3a the decoder gate input, G, can be used as chip select input for the 8 x 4-bit memory module. A block diagram of the 8 word by 4 bit memory is illustrated in Figure 6.3b.

The number of words, which can be accessed in a memory, is dependent on the number of address bus inputs available on the memory. For n address lines,  $2^n$  addresses can be specified. Contained in all memory is an n to  $2^n$  decoder which will decode the n-bit address and access one of  $2^n$  memory locations.

Several memory modules can be connected so that more words of data can be accessed. For example, suppose that a 32 word by 4 bit memory is to be constructed using 8 x 4-bit memory modules of Figure 6.3a. The 32 by 4-bit memory can be constructed using four, 8 x 4-bit memory modules and a 2 to 4 decoder. This circuit is illustrated in Figure 6.4a. To select one of 32 words, 5 address bus lines, A0 through A4, must be decoded. The 3 to 8 decoder in each memory module is used to decode address bus lines A0, A1, and A2. Address lines A3 and A4 are connected to the two select inputs, A and B, of the 2 to 4 decoder. The negative logic outputs of the decoder are connected to the CS inputs of the 8 x 4 bit memory modules. Data bus lines and the R/W input are interconnected to form the data and control bus. The decoder gate input, G, can be used as a chip select input,  $\overline{CS}$ , for this 32 x 4-bit memory module. A block diagram for the 32 x 4 bit memory is shown in Figure 6.4b.

Figure 6.4: 32 Word by Four Bit (32 x 4 bit) Memory

a) Digital Circuit b) Block Diagram



### 6.2.2. Memory Configurations

Computer memory can be considered as an m x n array of D-latches configured as m words of data with n bits per word. A 1024 x 4 bit memory would contain 4096 D-latches. Any one of 1024 words could be selected with each word being four bits.

Most memory is manufactured as a group of words, often numbering several thousand. Often, memory is said to contain so many kilo words of storage. The number of kilo-words is usually rounded down to the nearest whole power of two. When the word size is 8-bits, the term kilobyte is usually used.

A 1024 x 8-bit memory is usually described as a 1K-byte memory. A 32,768 x 8-bit memory is described as a 32K-byte memory.

The number of address pins available on the chip or memory module determines the maximum words, which can be accessed for a memory. Figure 6.5 contains a table of standard memories available in terms of number of words and the number of address pins required to access this number of words.

Figure 6.5: Word Size versus Address Lines

Name	Word Storage	Address Lines Required
1/2K	512	9
1K	1024	10
2K	2048	11
4K	4096	12
8K	8192	13
16K	16,384	14
32K	32,768	15
64K	65,536	16
...	...	...
512K	524,288	19
1M	1,048,576	20

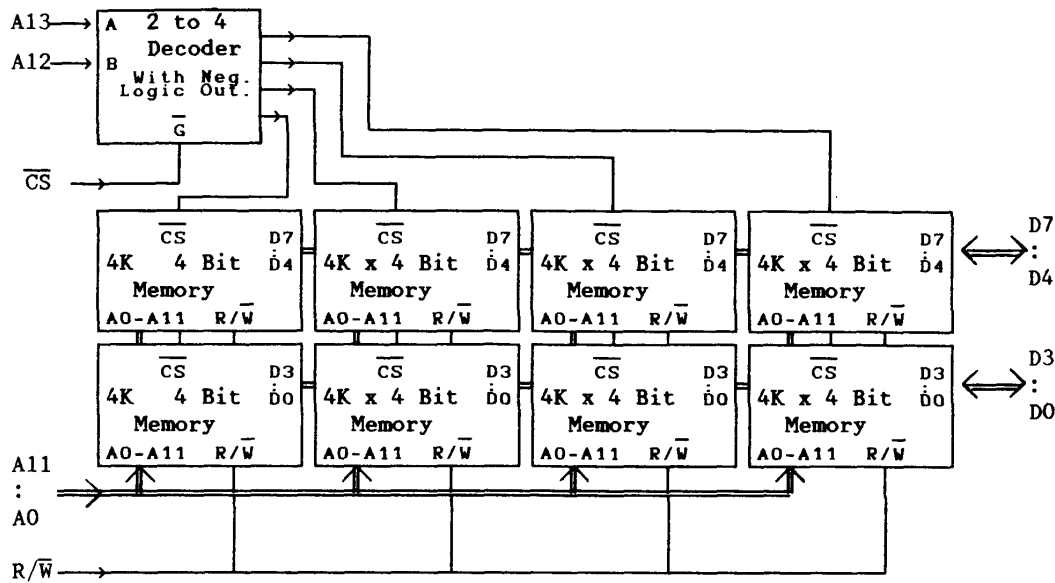
When constructing computer memory, the designer will often construct a large memory using smaller memory chips. For example, suppose that a designer wishes to construct a 16K x 8-bit memory using 4K x 4-bit memory chips. The design would require eight memory chips, as shown in Figure 6.6. The memory can be configured as a matrix with two rows and four columns. The R/W inputs and address lines, A0 through A11, are connected to all memory chips in the module. The two rows represent an increase in the data bus lines from four to eight bits. All memory chips in the same row would be connected to the same four data bus lines. The four columns of the memory matrix represent that the total number of words has been increased by a factor of four. A 2 to 4 decoder is required to select memory chips in one of the four columns.

Address lines, A12 and A13, are connected to the inputs of the 2 to 4 decoder. The four decoder outputs are connected to the CS inputs of all memory chips in the same column. The gate input,  $\overline{G}$ , on the decoder can be used as a chip select input for the 16K x 8-bit memory module.

The microprocessor clock output is connected to memory using the memory module  $\overline{CS}$  input. The clock output is used to provide proper timing between the microprocessor and memory. The address bus must be stable before a memory read or write operation can occur. Thus, the clock output specifies the time interval for which this operation can occur. The memory modules of Figures 6.3 and 6.4 could be used with a 4-bit microprocessor. The clock output of the microprocessor would be connected to the chip select input for either module.

The memory described by Figure 6.6 could be used with the 8-bit microprocessor, described in figure 6.1. When designing memories for microprocessor based systems, the designer must consider clock timing for both the microprocessor as well as memory devices.

Figure 6.6: 16K x 8 Bit Memory Configuration



### 6.2.3. RAM Memory

RAM memory is read/write memory. Data can be read from RAM and written to RAM by the microprocessor. RAM is actually an abbreviation for Random Access Memory. However, since all memory devices are constructed such that memory locations can be accessed at random using the address bus, this description has little meaning.

Two kinds of RAM are available. The first kind is called Static RAM. Static RAM operates as one self-contained unit and connects directly to the computer buses as illustrated in Figures 6.6. Dynamic RAM is the second type of RAM that is less expensive than Static RAM. However, it does require extra refresh circuitry. For large memory applications, Dynamic RAM is usually used because the lower cost of memory chips more than offsets the cost of the additional refresh circuitry.

RAM memory is said to be volatile. Volatile memory will lose all data if a power loss occurs on the memory device. Non-volatile memory will retain all data whether power is on or off.

### 6.2.4. ROM Memory

ROM memory is non-volatile. ROM is an acronym for Read Only Memory. Data can be read from ROM but cannot be written to ROM. ROM connects to all buses, just as RAM, except a R/W input is not supplied since only a memory read can occur. ROM is used primarily to store permanent programs used by the computer.

Several types of ROM are available from integrated circuit manufacturers.

Standard ROM comes from the manufacturer with all data permanently stored at specific addresses. Programmable ROM, or PROM, is memory that may be written to only once. Data cannot be changed after the initial programming. Erasable PROM, or EPROM, can be programmed for a specific application. If the data must be altered, the entire memory can be erased using Ultra-Violet light and then reprogrammed. Electrically Erasable PROM or EEPROM can be erased without removing it from the circuit. EEPROM will not substitute for RAM because its write operation is slow and cost is relatively high.

One variant of EEPROM currently used in digital cameras and portable MP3 music players is called Flash Memory. It is relatively inexpensive, hold large amounts of data (64Mbytes), and is non-volatile. Flash memory cards typically use the standard PCMCIA card slot configuration.

## 6.3. MICROPROCESSOR

The internal structure for a typical 8-bit microprocessor is shown in Figure 6.7. The microprocessor contains registers, an arithmetic logic unit, a control unit, and bus drivers. This block diagram illustrates the Motorola MC6809 microprocessor .

### 6.3.1. Registers

Registers are data storage cells inside the microprocessor unit. Registers are made of D-latches just like address memory. However they are not addressed using the address bus and therefore are the fastest access memory devices in a computer. Assembly language instructions specify how data is passed between registers and what operations are to be performed on data contained in registers.

The MC6809 microprocessor has nine registers, which may be used for data storage. Five of the nine registers are 16 bits wide while the other four registers are 8 bits wide. Thus 14 bytes of register storage are available on the MC6809, for programming by the user.

Registers X, Y, S, U, DP, and PC are called address registers and are used primarily for specifying the address of a particular memory location. The X and Y registers are index registers, which are used to specify a base address when using index addressing with an instruction. The S and U registers are stack pointers and are used to perform stack operations as well as specifying a base address with index addressing. The DP (Direct Page) register is only eight bits wide, and is used to specify the most significant eight bits of an address when using direct addressing mode. The least significant byte of the address is supplied as part of the instruction when using direct addressing mode with the DP register.

The PC (Program Counter) register is contained in all microprocessors and contains the address of the next instruction to be executed. The PC register in the MC6809 is 16 bits wide and is incremented automatically by the microprocessor as each instruction is executed. All programs, which are run by the microprocessor, are stored in memory. To begin program execution, the PC register must point to the first byte of the program in address memory.

Registers A and B are the 8 bit accumulators usually used for arithmetic or logic operations.

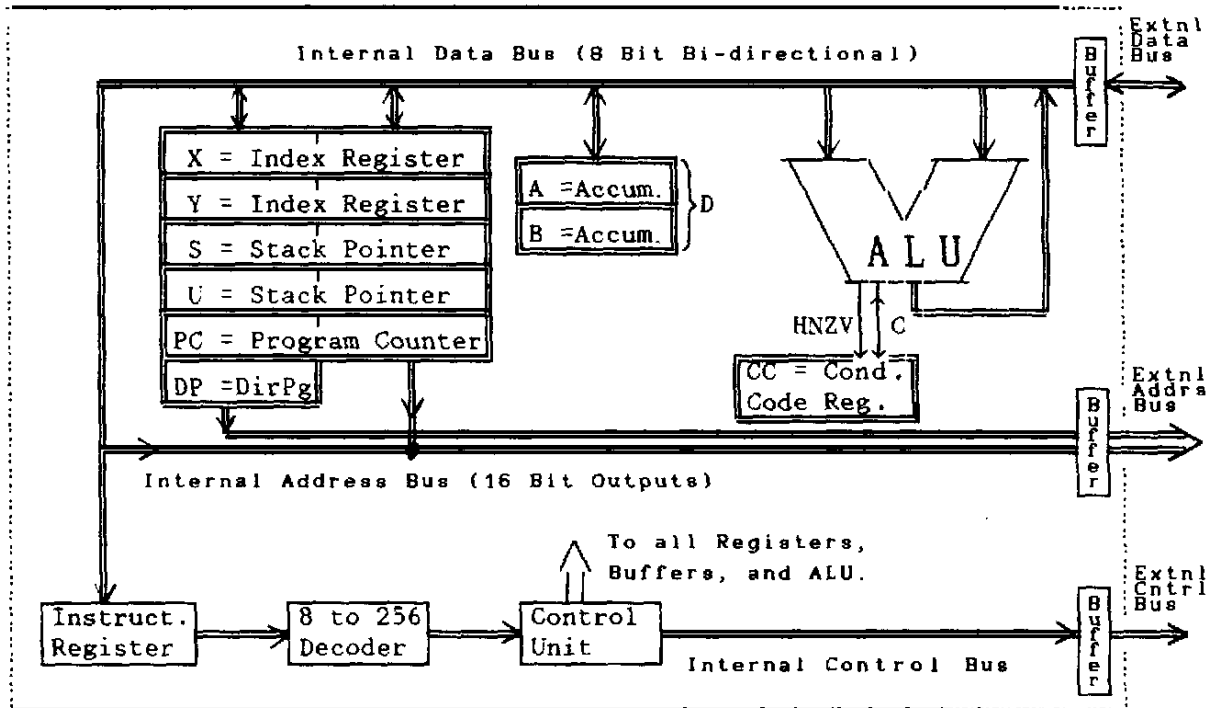
The Condition Code (CC) register contains the flags discussed in Chapter 5. The carry flag is represented by the C bit, the overflow flag is represented by the V bit, and the half carry flag is represented by the H bit of the condition code register and used for BCD addition. The N bit represents the negative flag and is set to the same state as the most significant bit of the result of the last operation. The Z bit represents the zero flag and is set if the last operation resulted in a zero. The E, F, and I flags are used for microprocessor interrupt applications.

The Instruction Register (IR) is contained in the microprocessor, however it is not accessible to the user. The instruction register contains the 8-bit code, which represents the instruction currently being executed. Each instruction has an 8 bit code called its operation code (op-code) which the microprocessor recognizes as the instruction.

### 6.3.2. CPU Arithmetic Logic Unit

The Arithmetic Logic Unit (ALU) contains the hardware required to perform arithmetic and logic operations on two 8-bit data words. The ALU of the MC6809 operates similar to the ALU described in Section 2.5.4. It can be considered to have eight 1-bit ALUs in parallel to perform the same operation on all eight bits. The ALU contained in the MC6809 will perform many more than the four functions described in Chapter 3. The result of an ALU operation may affect the flags of the condition code register.

Figure 6.7: Internal Structure of the MC6809 Microprocessor



### 6.3.3. Control Unit

The control unit is the nerve center of the microprocessor. After the instruction op-code contained in the instruction register is decoded, the control unit executes the instruction. The control unit controls all bus drivers, gates data between registers, performs operations on data using the ALU, and stores results in the proper registers.

### 6.3.4. Buses and Buffers

The internal data bus is bi-directional and is used to gate data between registers, the ALU, and the external data bus through buffers. Buffers contain the bus driver logic required to control the three external buses.

## 6.4. INSTRUCTION EXECUTION CYCLE

All programs that are executed by the microprocessor are contained in address memory. Each program instruction is represented by an eight-bit word, called the op-code, which is stored at a memory address. To execute instructions of a program the microprocessor performs an instruction execution cycle for each instruction. As depicted by Figure 6.8, the instruction execution cycle has three cycles: fetch, decode, and execute. Each of these cycles will be discussed in detail.

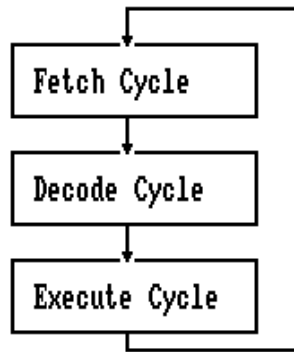
The fetch cycle begins by gating the contents of the PC register out on the address bus, and setting the control bus to a memory read condition. The byte, which represents the instruction op-code, is then put on the data bus by memory, and loaded into the instruction register (IR) of the microprocessor. Now that the instruction is received, the program counter is incremented by one to point to the next byte in program memory.

The decode cycle occurs after the instruction register load. The 8-bit op-code contained in the instruction register is decoded using the 8 to 256 decoder. The proper signal is sent to the control unit.

The execute cycle is performed by the control unit, based on the signal it receives from the decoder. Instruction execution may include reading additional data from memory, transferring data between registers, storing data in memory, and ALU operations.

After an instruction is completely executed, the instruction execution cycle begins again with another fetch cycle. The PC register will always contain the address of the next instruction to be executed at the end of the instruction execution cycle. The instruction register will always contain the 8 bit op-code of the instruction currently being executed.

Figure 6.8: Instruction Execution Cycle



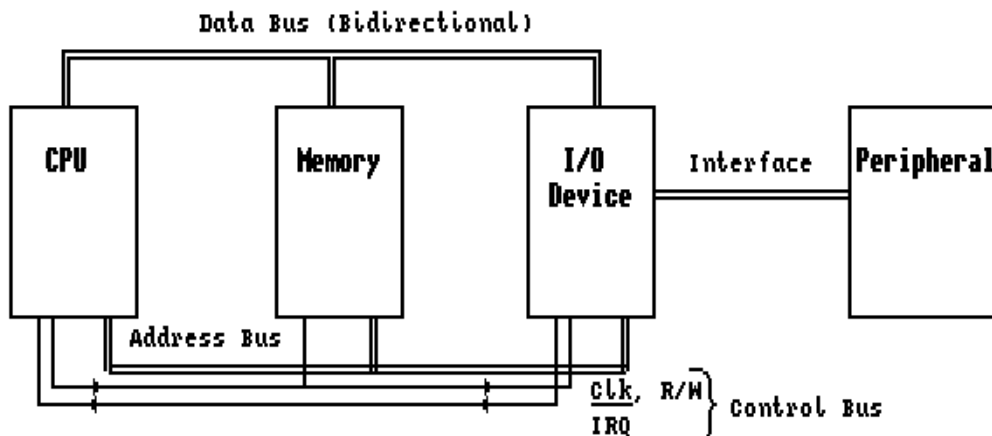
The fetch cycle begins by gating the contents of the PC register out on the address bus, and setting the control bus to a memory read condition. The byte, which represents the instruction op-code, is then put on the data bus by memory, and loaded into the instruction register (IR) of the microprocessor. Now that the instruction is received, the program counter is incremented by one to point to the next byte in program memory.

The decode cycle occurs after the instruction register load. The 8-bit op-code contained in the instruction register is decoded using the 8 to 256 decoder. The proper signal is sent to the control unit.

The execute cycle is performed by the control unit, based on the signal it receives from the decoder. Instruction execution may include reading additional data from memory, transferring data between registers, storing data in memory, and ALU operations.

After an instruction is completely executed, the instruction execution cycle begins again with another fetch cycle. The PC register will always contain the address of the next instruction to be executed at the end of the instruction execution cycle. The instruction register will always contain the 8 bit op-code of the instruction currently being executed.

Figure 6.9: Computer Architecture Block Diagram



Address	Opcode	Operand	Assembly Code
D000	86	12	LDA #\$12
D002	8B	0C	ADDA #\$0C
D004	B7	D100	STA \$D100
D007	BB	D110	ADDA \$D110
D00A	B7	D101	STA \$D101
D00D	8B	1E	ADDA #\$1E
D00F	B7	D102	STA \$D102
D012	24	07	BCC \$D01B
D014	86	00	LDA #\$00
D016	B7	D110	STA \$D110
D019	20	FC	BRA \$D007
D01B	3F		SWI
D01C	08		FCB 08
Data Section			
D110	C0		

Boot ROM	0000	
	0001	
	0002	
	...	
	3FFF	
Addresses of	4000	
Peripheral	4001	
Devices	4002	
	...	
System RAM	7FFF	
	8000	
	8001	
	8002	
	...	
Program Start	D000	86
	D001	12
	D002	8B
	D003	0C
	D004	B7
	D005	D1
	D006	00
	D007	BB
	D008	D1
	D009	10
	D00A	B7
	D00B	D1
	D00C	01
	...	
	FFFF	

