AVR Microcontroller

Microprocessor Course
Chapter 0
NUMBERING AND CODING SYSTEMs

Esfand 1397 (version 1.0)

AVR Microcontroller

0.1 NUMBERING AND CODING SYSTEMS

Where as human beings use base 10 (decimal) arithmetic, computers use the base 2 (binary) system. In this section we explain how to convert from the decimal system to the binary system, and vice versa.

The convenient representation of binary numbers, called hexadecimal, also is covered. Finally, the binary format of the alphanumeric code, called ASCII, is explored.

AVR Microcontroller
0.1 NUMBERING AND CODING SYSTEMS

Decimal and binary number systems

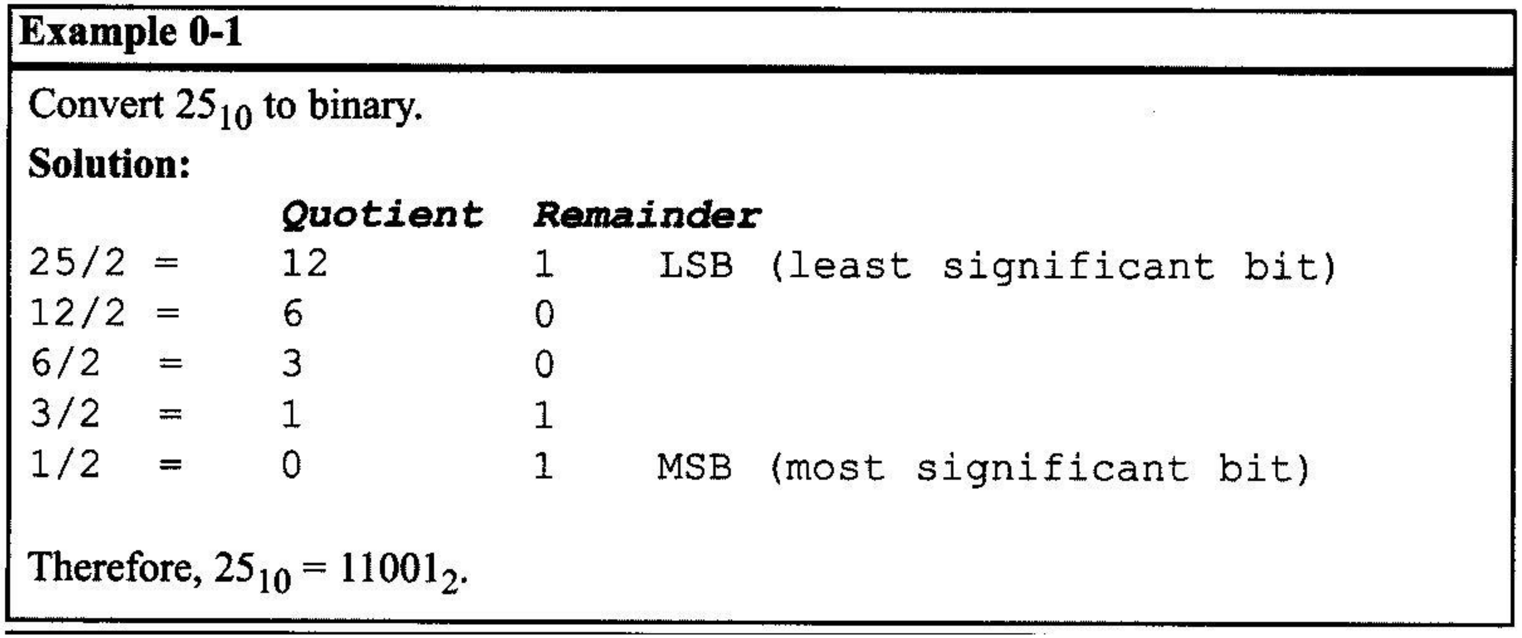
The binary system is used in computers because 1 and 0 represent the two voltage levels of on and off. Whereas in base 10 there are 10 distinct symbols, 0, 1, 2, …, 9, in base 2 there are only two, 0 and 1, with which to generate numbers. Base 10 contains digits 0 through 9; binary contains digits 0 and 1 only. These two binary digits, 0 and 1, are commonly referred to as bits.

AVR Microcontroller

0.1 NUMBERING AND CODING SYSTEMS

Converting from decimal to binary

One method of converting from decimal to binary is to divide the decimal number by 2 repeatedly, keeping track of the remainders. This process continues until the quotient becomes zero. The remainders are then written in reverse order to obtain the binary number. This is demonstrated in Example 0-1.



Example 0-1

Convert 25_{10} to binary.

Solution:

	Quotient	***Remainder***	
25/2 =	12	1	LSB (least significant bit)
12/2 =	6	0	
6/2 =	3	0	
3/2 =	1	1	
1/2 =	0	1	MSB (most significant bit)

Therefore, $25_{10} = 11001_2$.

AVR Microcontroller

0.1 NUMBERING AND CODING SYSTEMS

Converting from binary to decimal

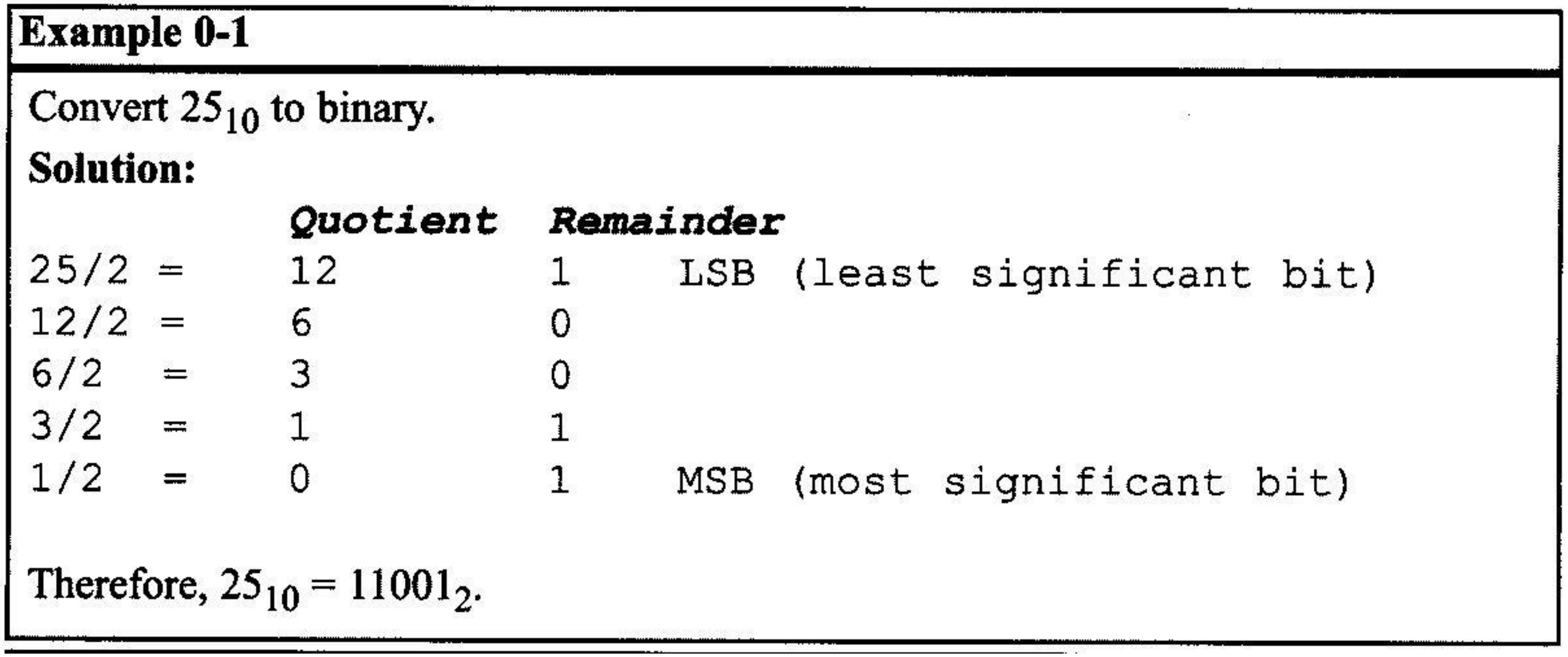
To convert from binary to decimal, it is important to understand the concept of weight associated with each digit position. First, as an analogy, recall the weight of numbers in the base 10 system, as shown in the diagram. By the same token, each digit position of a number in base 2 has a weight associated with it:

740683_{10}	=	
3×10^0	=	3
8×10^1	=	80
6×10^2	=	600
0×10^3	=	0000
4×10^4	=	40000
7×10^5	=	700000
		740683

110101_2 =				**Decimal**	**Binary**
1×2^0	=	1×1	=	1	1
0×2^1	=	0×2	=	0	00
1×2^2	=	1×4	=	4	100
0×2^3	=	0×8	=	0	0000
1×2^4	=	1×16	=	16	10000
1×2^5	=	1×32	=	32	100000
				53	110101

AVR Microcontroller

0.1 NUMBERING AND CODING SYSTEMS



Example 0-1

Convert 25_{10} to binary.

Solution:

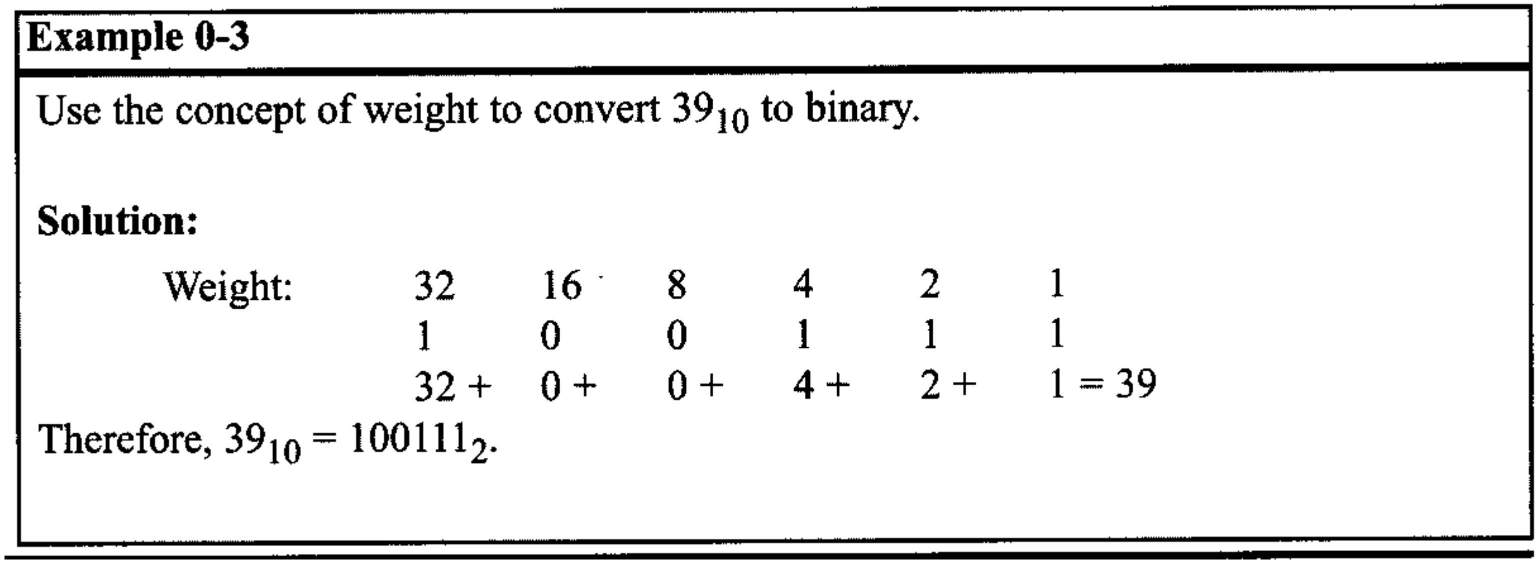
		Quotient	*Remainder*	
25/2	=	12	1	LSB (least significant bit)
12/2	=	6	0	
6/2	=	3	0	
3/2	=	1	1	
1/2	=	0	1	MSB (most significant bit)

Therefore, $25_{10} = 11001_2$.

AVR Microcontroller

0.1 NUMBERING AND CODING SYSTEMS

Knowing the weight associated with each binary bit position allows one to convert a decimal number to binary directly instead of going through the process of repeated division. This is shown in Example 0-3.



Example 0-3

Use the concept of weight to convert 39_{10} to binary.

Solution:

Weight:	32	16	8	4	2	1
	1	0	0	1	1	1
	32 +	0 +	0 +	4 +	2 +	1 = 39

Therefore, $39_{10} = 100111_2$.

AVR Microcontroller

0.1 NUMBERING AND CODING SYSTEMS

Hexadecimal system

Base 16, or the hexadecimal system is used as a convenient representation of binary numbers. For example, it is much easier for a human being to represent a string of Os and 1 s such as 100010010110 as its hexadecimal equivalent of 896H.

The binary system has 2 digits, 0 and 1.

The base 10 system has 10 digits, 0 through 9.

The hexadecimal (base 16) system has 16 digits. In base 16, the first 10 digits, 0 to 9, are the same as in decimal, and for the remaining six digits, the letters A, B, C, D, E, and F are used.

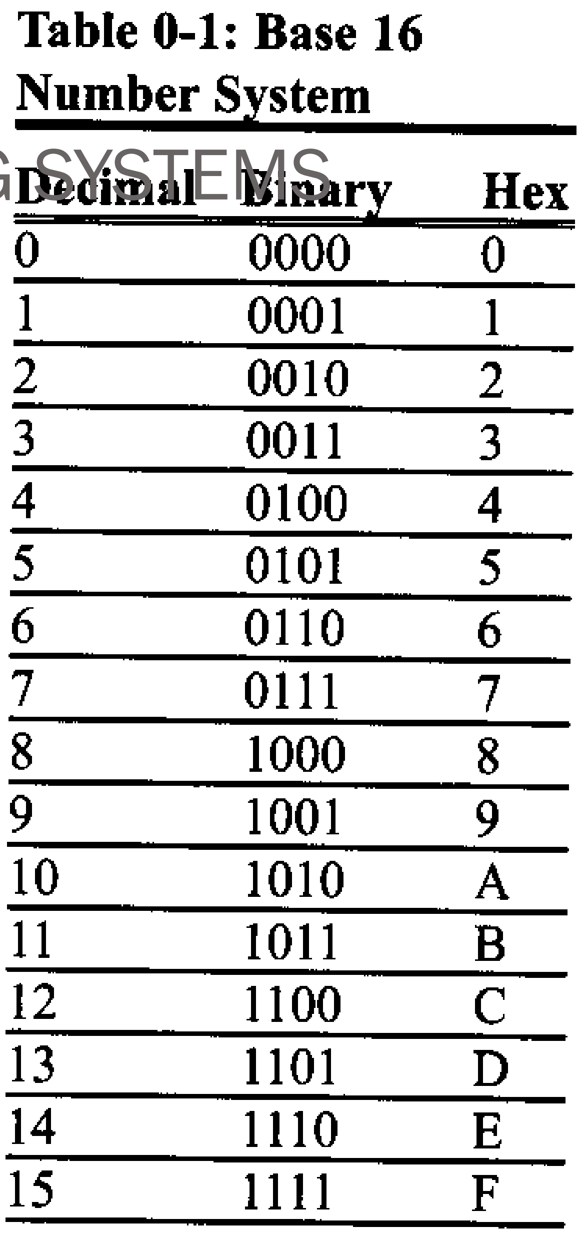


Table 0-1: Base 16 Number System

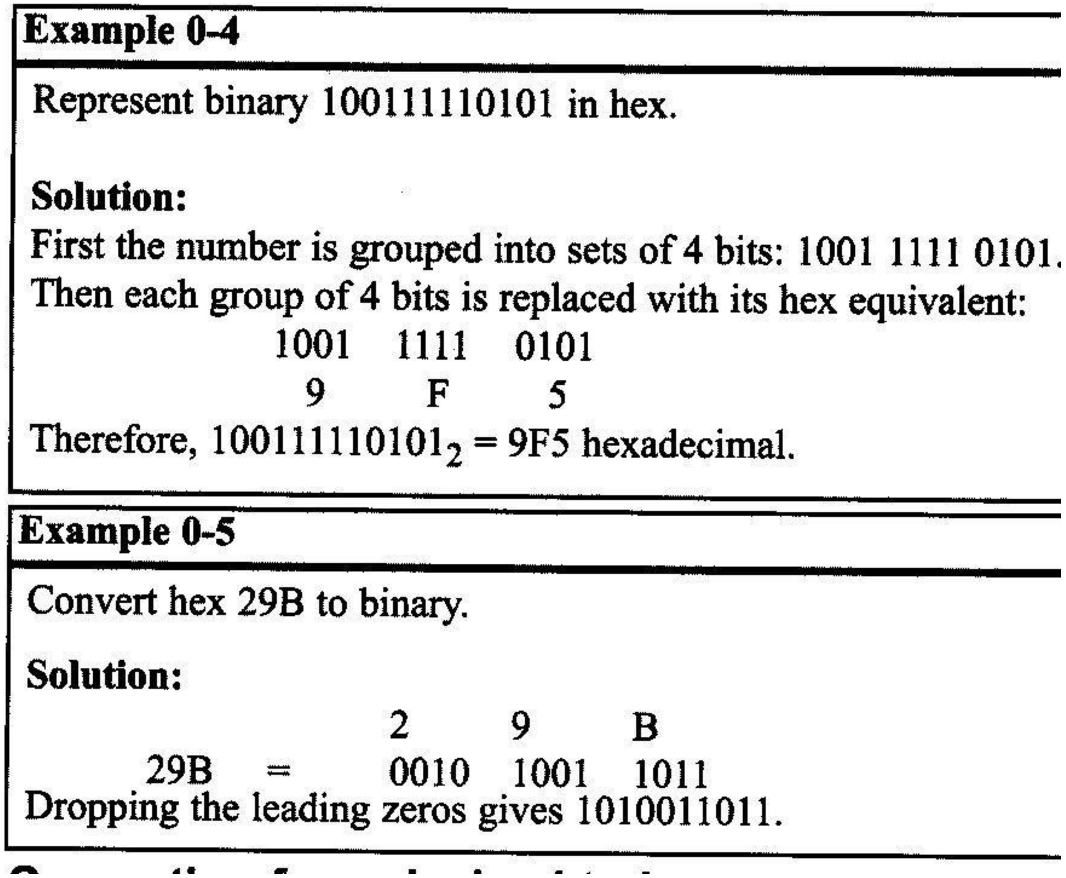
Decimal	Binary	Hex
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

AVR Microcontroller

0.1 NUMBERING AND CODING SYSTEMS

Converting between binary and hex

To represent a binary number as its equivalent hexadecimal number, start from the right and group 4 bits at a time, replacing each 4-bit binary number with its hex equivalent shown in Table 0-1. To convert from hex to binary, each hex digit is replaced with its 4-bit binary equivalent.



Example 0-4

Represent binary 100111110101 in hex.

Solution:

First the number is grouped into sets of 4 bits: 1001 1111 0101.
Then each group of 4 bits is replaced with its hex equivalent:

```
        1001   1111   0101
          9      F      5
```

Therefore, 100111110101_2 = 9F5 hexadecimal.

Example 0-5

Convert hex 29B to binary.

Solution:

```
                    2      9      B
     29B    =     0010   1001   1011
```

Dropping the leading zeros gives 1010011011.

AVR Microcontroller

0.1 NUMBERING AND CODING SYSTEMS

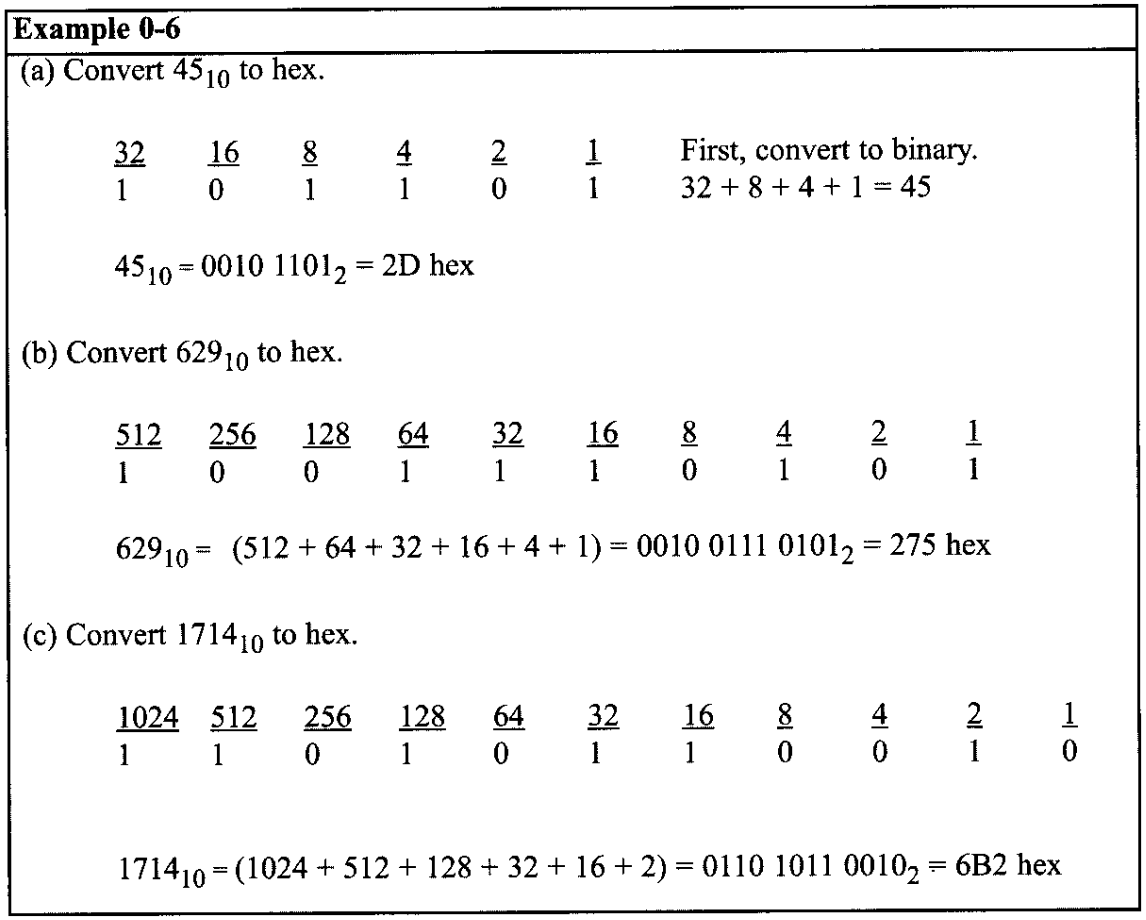
Converting from decimal to hex

Converting from decimal to hex could be approached in two ways:

1. Convert to binary first and then convert to hex. Example 0-6 shows this method of converting decimal to hex.
2. Convert directly from decimal to hex by repeated division, keeping track of the remainders. Experimenting with this method is left to the reader.

AVR Microcontroller

0.1 NUMBERING AND CODING SYSTEMS



Example 0-6

(a) Convert 45_{10} to hex.

32	16	8	4	2	1
1	0	1	1	0	1

First, convert to binary.
32 + 8 + 4 + 1 = 45

$45_{10} = 0010\ 1101_2 = $ 2D hex

(b) Convert 629_{10} to hex.

512	256	128	64	32	16	8	4	2	1
1	0	0	1	1	1	0	1	0	1

$629_{10} = (512 + 64 + 32 + 16 + 4 + 1) = 0010\ 0111\ 0101_2 = $ 275 hex

(c) Convert 1714_{10} to hex.

1024	512	256	128	64	32	16	8	4	2	1
1	1	0	1	0	1	1	0	0	1	0

$1714_{10} = (1024 + 512 + 128 + 32 + 16 + 2) = 0110\ 1011\ 0010_2 = $ 6B2 hex

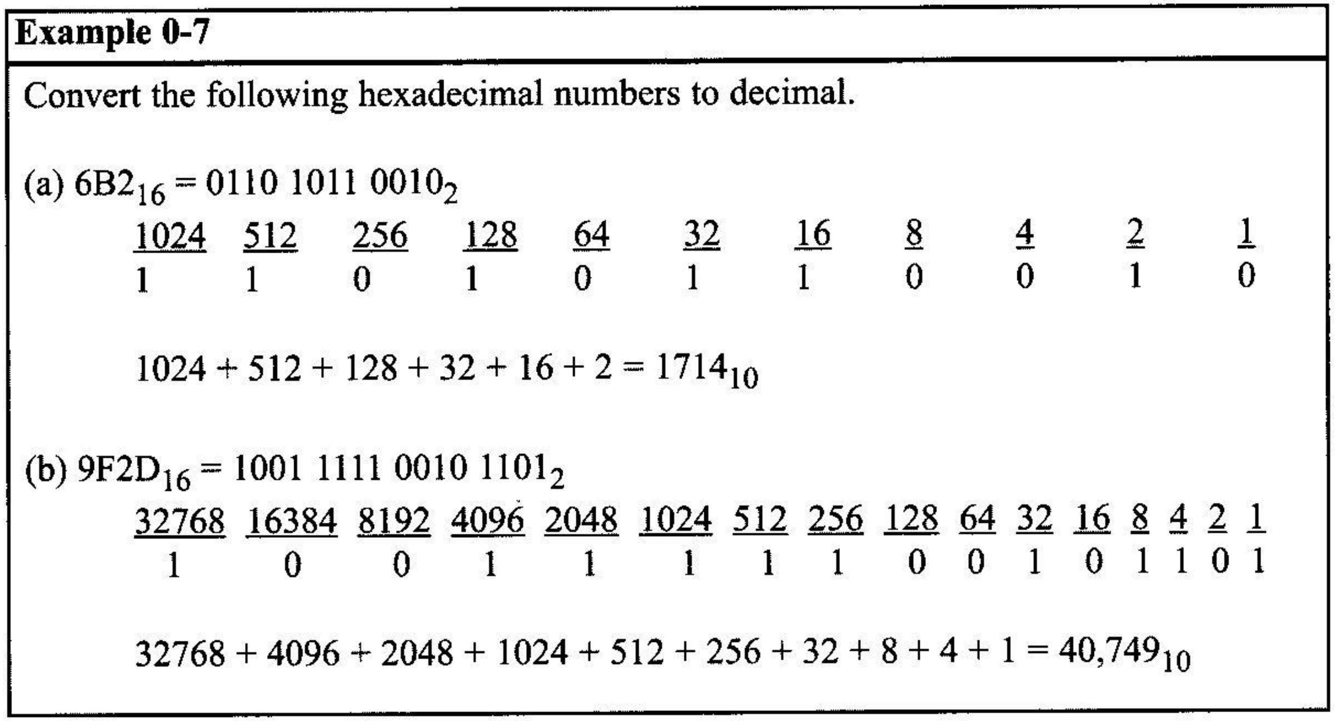
AVR Microcontroller

0.1 NUMBERING AND CODING SYSTEMS

Converting from hex to decimal

Conversion from hex to decimal can also be approached in two ways:

1. Convert from hex to binary and then to decimal. Example 0-7 demonstrates this method of converting from hex to decimal.
2. Convert directly from hex to decimal by summing the weight of all digits.



Example 0-7

Convert the following hexadecimal numbers to decimal.

(a) $6B2_{16} = 0110\ 1011\ 0010_2$

1024	512	256	128	64	32	16	8	4	2	1
1	1	0	1	0	1	1	0	0	1	0

$1024 + 512 + 128 + 32 + 16 + 2 = 1714_{10}$

(b) $9F2D_{16} = 1001\ 1111\ 0010\ 1101_2$

32768	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1
1	0	0	1	1	1	1	1	0	0	1	0	1	1	0	1

$32768 + 4096 + 2048 + 1024 + 512 + 256 + 32 + 8 + 4 + 1 = 40{,}749_{10}$

AVR Microcontroller

0.1 NUMBERING AND CODING SYSTEMS

Counting in bases 10, 2, and 16

To show the relationship between all three bases, in Table 0-2 we show the sequence of numbers from 0 to 31 in decimal, along with the equivalent binary and hex numbers. Notice in each base that when one more is added to the highest digit, that digit becomes zero and a 1 is carried to the next-highest digit position. For example, in decimal, 9 + 1= 0 with a carry to the next-highest position. In binary, 1 + 1 = 0 with a carry; similarly, in hex, F + 1 = 0 with a carry.

AVR Microcontroller

0.1 NUMBERING AND CODING SYSTEMS

Addition of binary and hex numbers

The addition of binary numbers is a very straightforward process. Table 0-3 shows the addition of two bits. The discussion of subtraction of binary numbers is bypassed since all computers use the addition process to implement subtraction. Although computers have adder circuitry, there is no separate circuitry for subtractors. Instead, adders are used in conjunction with 2's complement circuitry to perform subtraction. In other words, to implement "x - y", the computer takes the 2's complement of y and adds it to x.

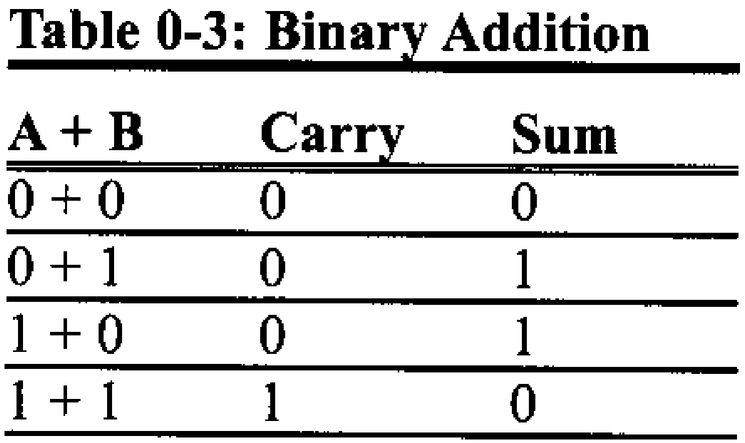
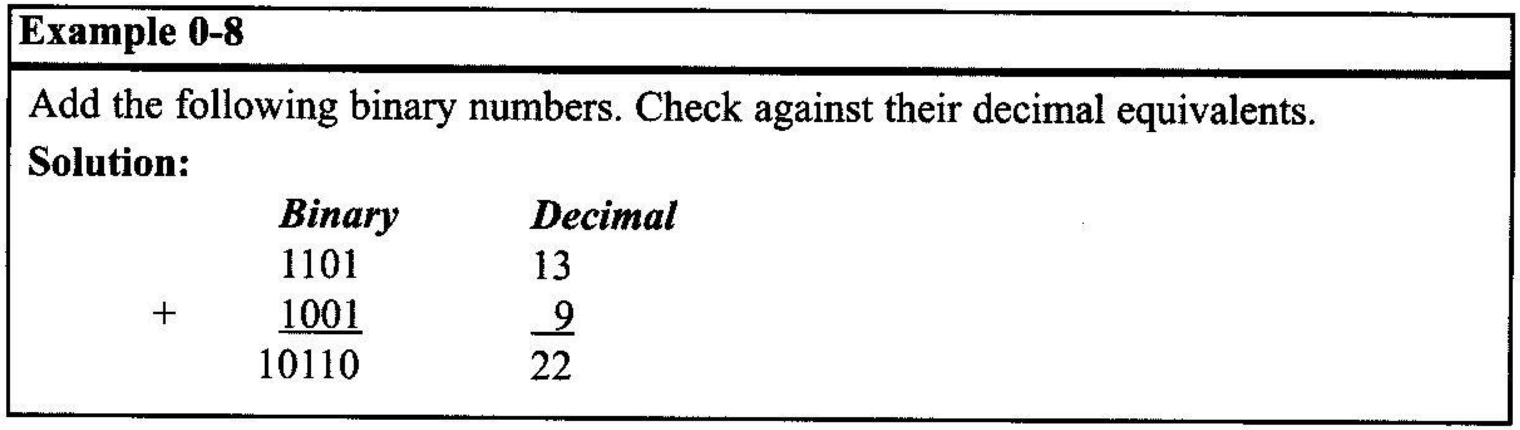


Table 0-3: Binary Addition

A + B	Carry	Sum
0 + 0	0	0
0 + 1	0	1
1 + 0	0	1
1 + 1	1	0

AVR Microcontroller

0.1 NUMBERING AND CODING SYSTEMS



Example 0-8

Add the following binary numbers. Check against their decimal equivalents.

Solution:

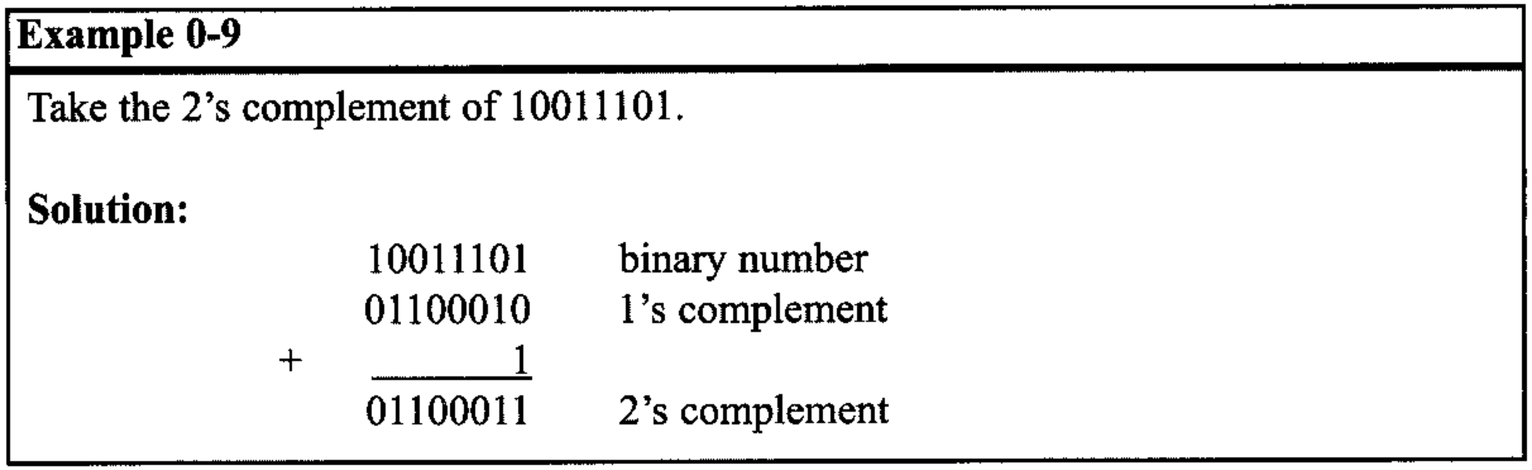
$$\begin{array}{crr} & \textit{Binary} & \textit{Decimal} \\ & 1101 & 13 \\ + & \underline{1001} & \underline{9} \\ & 10110 & 22 \end{array}$$

AVR Microcontroller

0.1 NUMBERING AND CODING SYSTEMS

2's **complement**

To get the 2's complement of a binary number, invert all the bits and then add 1 to the result. Inverting the bits is simply a matter of changing all 0s to 1s and 1s to 0s. This is called the 1's complement.



Example 0-9

Take the 2's complement of 10011101.

Solution:

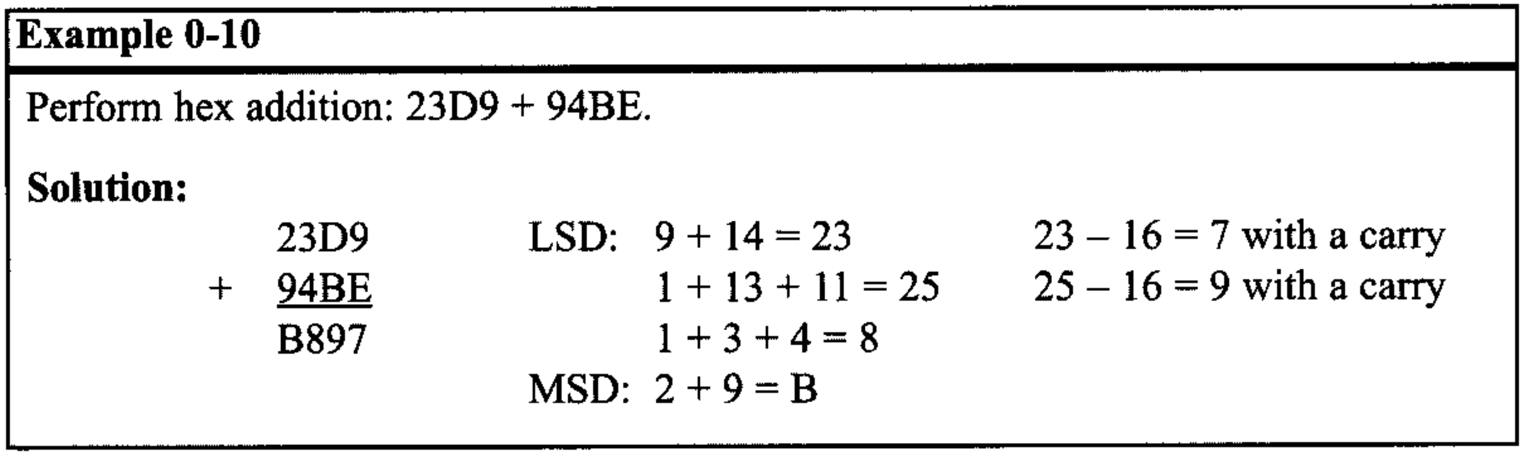
```
        10011101     binary number
        01100010     1's complement
  +            1
        --------
        01100011     2's complement
```

AVR Microcontroller

0.1 NUMBERING AND CODING SYSTEMS

Addition of hex numbers

This section describes the process of adding hex numbers. Starting with the least significant digits, the digits are added together. If the result is less than 16, write that digit as the sum for that position. If it is greater than 16, subtract 16 from it to get the digit and carry 1 to the next digit.



Example 0-10

Perform hex addition: 23D9 + 94BE.

Solution:

```
       23D9      LSD:  9 + 14 = 23          23 – 16 = 7 with a carry
    +  94BE            1 + 13 + 11 = 25     25 – 16 = 9 with a carry
       B897            1 + 3 + 4 = 8
                 MSD:  2 + 9 = B
```

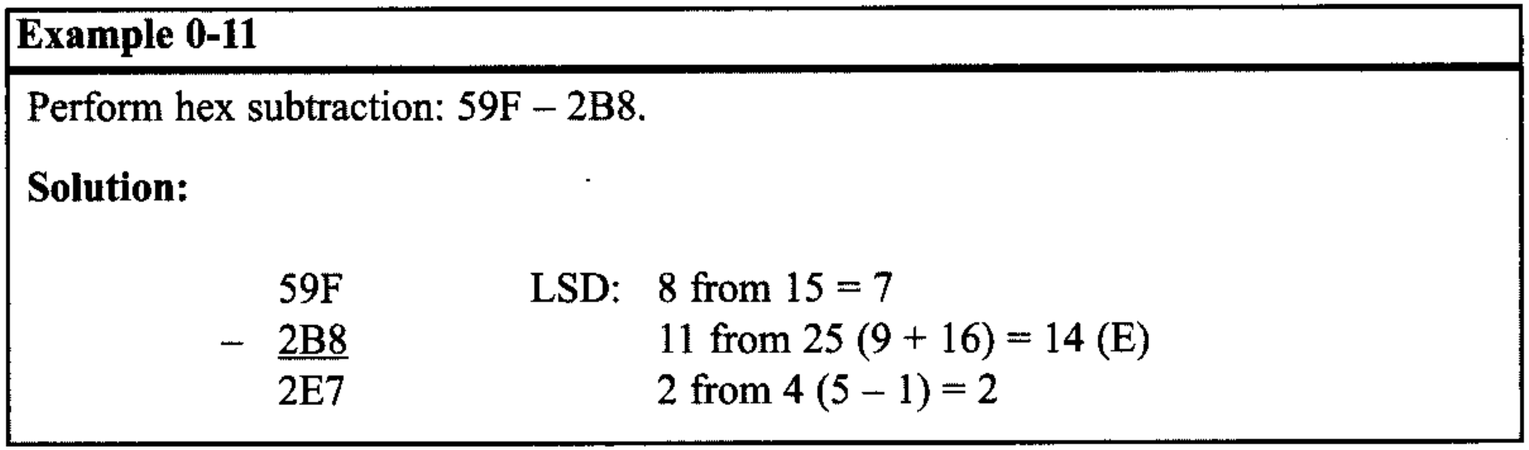
AVR Microcontroller
0.1 NUMBERING AND CODING SYSTEMS

Addition and subtraction of hex numbers

In studying issues related to software and hardware of computers, it is often necessary to add or subtract hex numbers. Mastery of these techniques is essential. Hex addition and subtraction are discussed separately below.

Subtraction of hex numbers

In subtracting two hex numbers, if the second digit is greater than the first, borrow 16 from the preceding digit. See Example 0-11.



Example 0-11

Perform hex subtraction: 59F – 2B8.

Solution:

```
      59F        LSD:  8 from 15 = 7
   –  2B8              11 from 25 (9 + 16) = 14 (E)
      2E7              2 from 4 (5 – 1) = 2
```

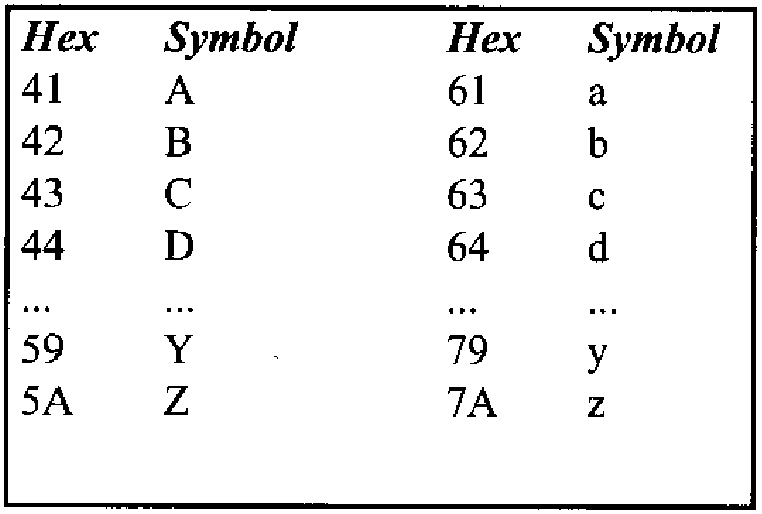
AVR Microcontroller

0.1 NUMBERING AND CODING SYSTEMS

ASCII code

The discussion so far has revolved around the representation of number systems. Because all information in the computer must be represented by 0s and 1s, binary patterns must be assigned to letters and other characters. The great advantage of this system is that it is used by most computers, so that information can be shared among computers. The ASCII system uses a total of 7 bits to represent each code. For example, 100 0001 is assigned to the uppercase letter "A" and 110 0001 is for the lowercase "a".

The use of ASCII is not only standard for keyboards used in the United States and many other countries but also provides a standard for printing and displaying characters by output devices such as printers and monitors.



Hex	*Symbol*	*Hex*	*Symbol*
41	A	61	a
42	B	62	b
43	C	63	c
44	D	64	d
...	...	...	...
59	Y	79	y
5A	Z	7A	z

Figure 0-1. Selected ASCII Codes

AVR Microcontroller

0.2 DIGITAL PRIMER

Computers use the binary number system because the two voltage levels can be represented as the two digits 0 and 1. Signals in digital electronics have two distinct voltage levels. For example, a system may define 0V as logic 0 and +5 V as logic 1.

Logic gates

Binary logic gates are simple circuits that take one or more input signals and send out one output signal.

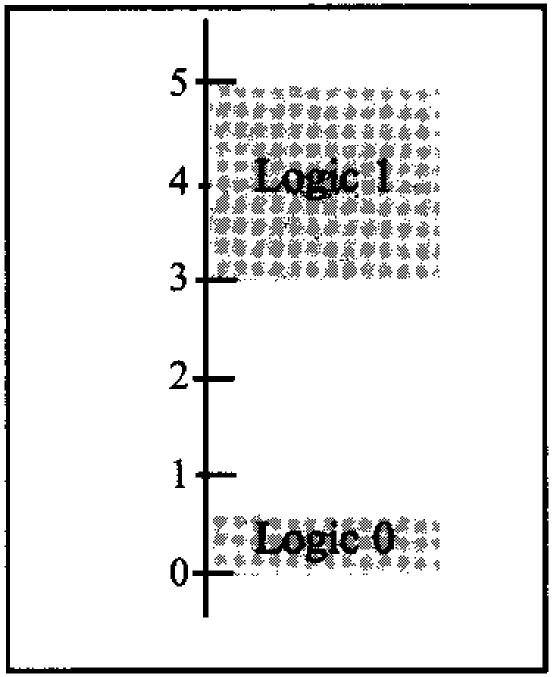


Figure 0-2. Binary Signals

Logical AND Function

Inputs	**Output**
X Y	**X AND Y**
0 0	0
0 1	0
1 0	0
1 1	1

AVR Microcontroller

0.2 DIGITAL PRIMER

AND gate

The AND gate takes two or more inputs and performs a logic AND on them. See the truth table and diagram of the AND gate. Notice that if both inputs to the AND gate are 1, the output will be 1. Any other combination of inputs will give a 0 output. The example shows two inputs, x and y. Multiple outputs are also possible for logic gates.

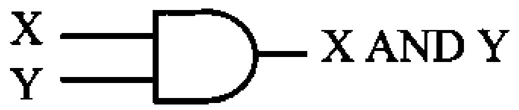
In the case of AND, if all inputs are 1, the output is 1. If any input is 0, the output is 0.

OR gate

The OR logic function will output a 1 if one or more inputs is 1. If all inputs are 0, then and only then will the output be 0.

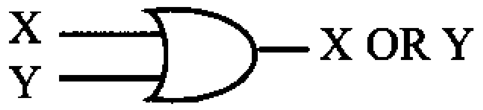
Logical AND Function

Inputs	Output
X Y	X AND Y
0 0	0
0 1	0
1 0	0
1 1	1



Logical OR Function

Inputs	Output
X Y	X OR Y
0 0	0
0 1	1
1 0	1
1 1	1



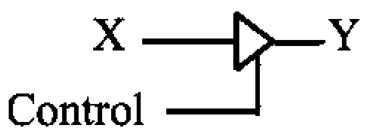
AVR Microcontroller

0.2 DIGITAL PRIMER

Tri-state buffer

A buffer gate does not change the logic level of the input. It is used to isolate or amplify the signal.

Buffer

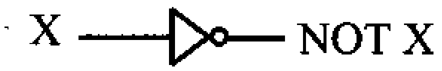


Inverter

The inverter, also called NOT, outputs the value opposite to that input to the gate. That is, a 1 input will give a 0 output, while a 0 input will give a 1 output.

Logical Inverter

Input	Output
X	**NOT X**
0	1
1	0



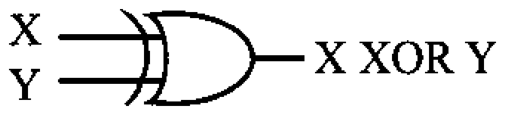
AVR Microcontroller
0.2 DIGITAL PRIMER

XOR gate

The XOR gate performs an exclusive-OR operation on the inputs. Exclusive-OR produces a 1 output if one (but only one) input is 1. If both operands are 0, the output is 0. Likewise, if both operands are 1, the output is also 0. Notice from the XOR truth table, that whenever the two inputs are the same, the output is 0.

Logical XOR Function

Inputs	**Output**
X Y	**X XOR Y**
0 0	0
0 1	1
1 0	1
1 1	0



AVR Microcontroller

0.2 DIGITAL PRIMER

NAND and NOR gates

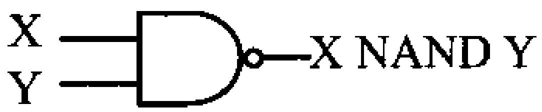
The NAND gate functions like an AND gate with an inverter on the output. It produces a 0 output when all inputs are 1; otherwise, it produces a 1 out-put.

The NOR gate functions like an OR gate with an inverter on the output. It produces a 1 if all inputs are 0; otherwise, it produces a 0.

NAND and NOR gates are used extensively in digital design because they are easy and inexpensive to fabricate. Any circuit that can be designed with AND, OR, XOR, and INVERTER gates can be implemented using only NAND and NOR gates.

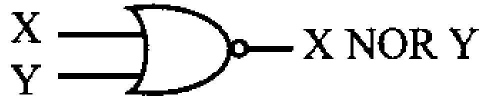
Logical NAND Function

Inputs	**Output**
X Y	**X NAND Y**
0 0	1
0 1	1
1 0	1
1 1	0



Logical NOR Function

Inputs	**Output**
X Y	**X NOR Y**
0 0	1
0 1	0
1 0	0
1 1	0



AVR Microcontroller

0.2 DIGITAL PRIMER

Logic design using gates

Next we will show a simple logic design to add two binary digits. If we add two binary digits there are four possible outcomes:

	Carry	*Sum*
0 + 0 =	0	0
0 + 1 =	0	1
1 + 0 =	0	1
1 + 1 =	1	0

Notice that when we add 1 + 1 we get 0 with a carry to the next higher place. We will need to determine the sum and the carry for this design.
Notice that the sum column above matches the output for the XOR function, and that the carry column matches the output for the AND function.

AVR Microcontroller
0.2 DIGITAL PRIMER

Figure 0-3(a) shows a simple adder implemented with XOR and AND gates. Figure 0-3(b) shows the same logic circuit implemented with AND and OR gates and inverters.

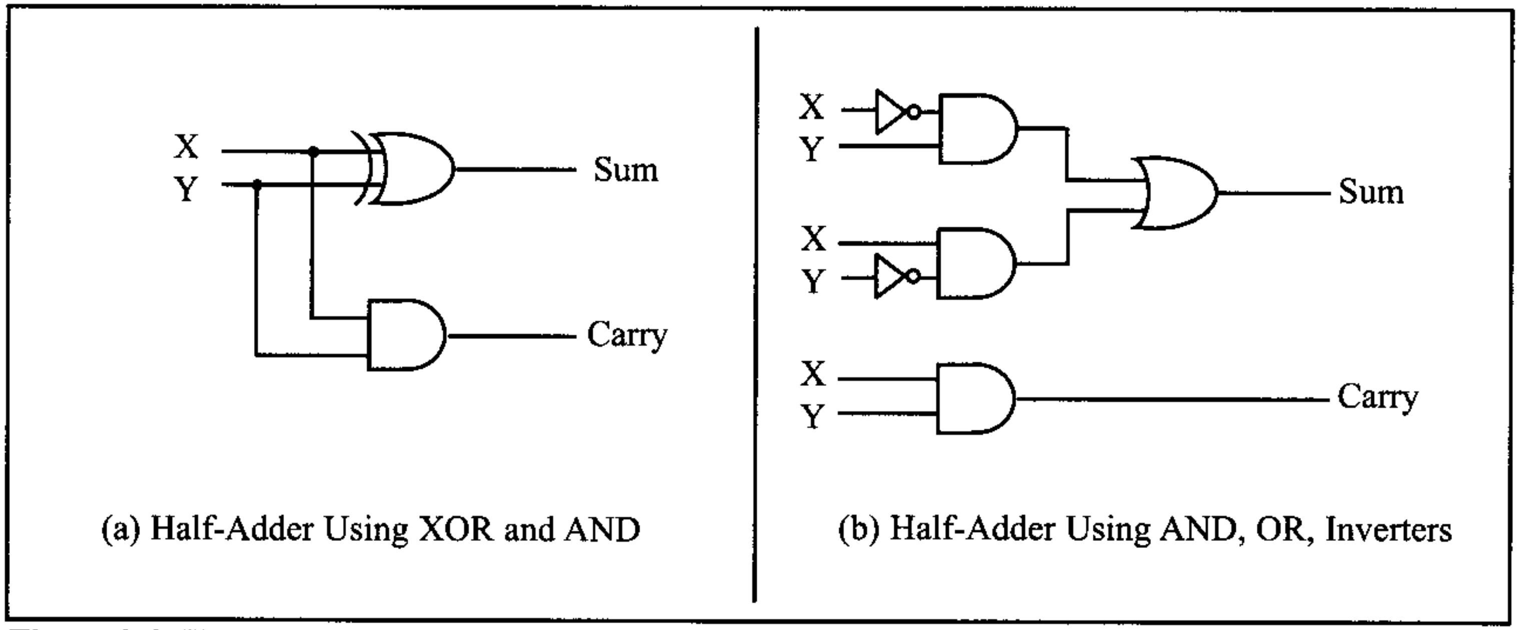

X
Y
Sum
Carry
(a) Half-Adder Using XOR and AND
X
Y
X
Y
X
Y
Sum
Carry
(b) Half-Adder Using AND, OR, Inverters

Figure 0-3. Two Implementations of a Half-Adder

AVR Microcontroller
0.2 DIGITAL PRIMER

Figure 0-4 shows a block diagram of a half-adder. Two half-adders can be combined to form an adder that can add three input digits. This is called a full-adder.

Figure 0-5 shows the logic diagram of a full-adder, along with a block diagram that masks the details of the circuit.

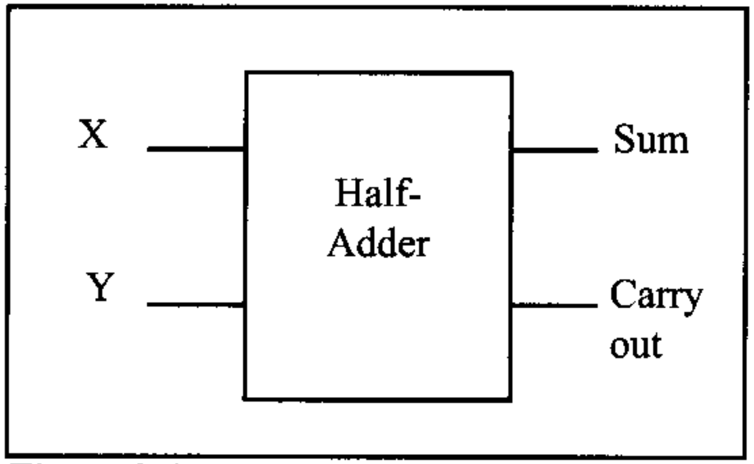


Figure 0-4. Block Diagram of a Half-Adder

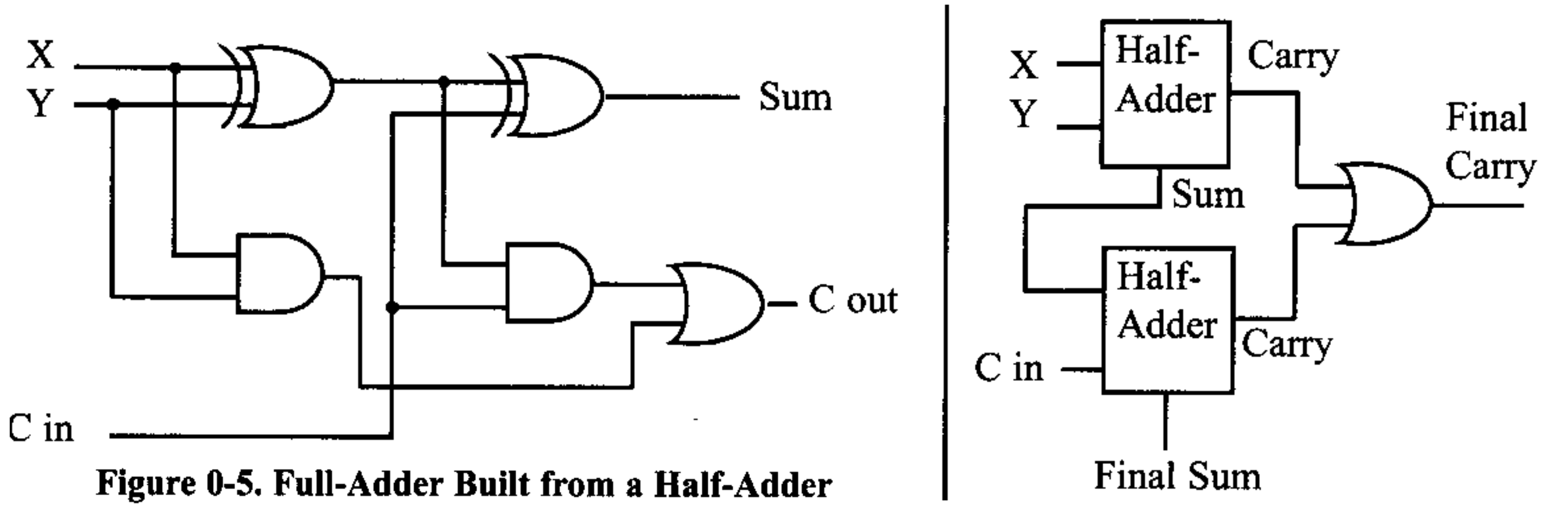


Figure 0-5. Full-Adder Built from a Half-Adder

AVR Microcontroller

0.2 DIGITAL PRIMER

Figure 0-6 shows a 3-bit adder using three full-adders

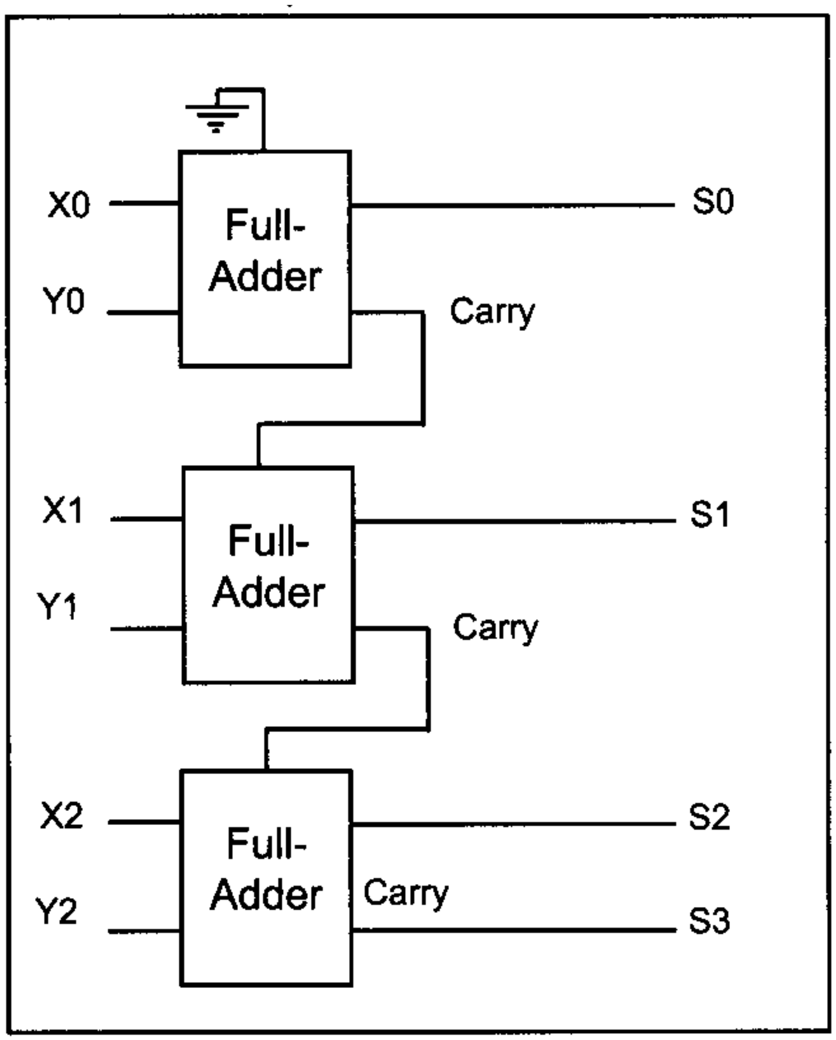


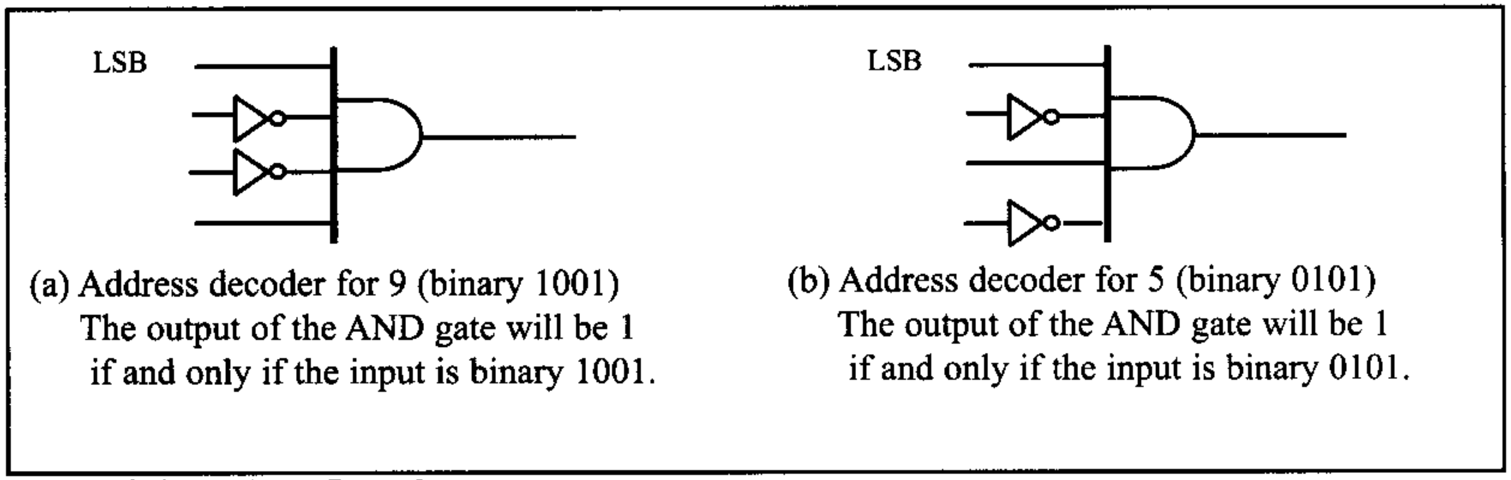
Figure 0-6. 3-Bit Adder Using Three Full-Adders

AVR Microcontroller

0.2 DIGITAL PRIMER

Decoders

Another example of the application of logic gates is the decoder. Decoders are widely used for address decoding in computer design. Figure 0-7 shows decoders for 9 (1001 binary) and 5 (0101) using inverters and AND gates.



(a) Address decoder for 9 (binary 1001)
The output of the AND gate will be 1
if and only if the input is binary 1001.

(b) Address decoder for 5 (binary 0101)
The output of the AND gate will be 1
if and only if the input is binary 0101.

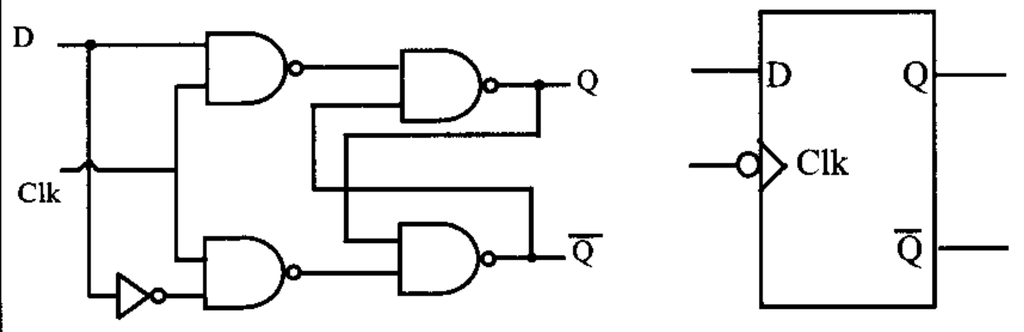
Figure 0-7. Address Decoders

AVR Microcontroller

0.2 DIGITAL PRIMER

Flip-flops

A widely used component in digital systems is the flip-flop. Frequently, flip-flops are used to store data. Figure 0-8 shows the logic diagram, block diagram, and truth table for a flip-flop. The D flip-flop is widely used to latch data. Notice from the truth table that a D-FF grabs the data at the input as the clock is activated. A D-FF holds the data as long as the power is on.



(a) Circuit diagram (b) Block diagram

Clk	D	Q
No	x	no change
↓	0	0
↓	1	1

x = don't care

(c) Truth table

Figure 0-8. D Flip-Flops

AVR Microcontroller

0.3 SEMICONDUCTOR MEMORY

In this section we discuss various types of semiconductor memories and their characteristics such as capacity, organization, and access time.
We will also show how the memory is connected to CPU. Before we embark on the subject of memory, it will be helpful to give an overview of computer organization and review some widely used terminology in computer literature.

AVR Microcontroller

0.3 SEMICONDUCTOR MEMORY

Some important terminology

Recall from the discussion above that a bit is **a binary digit** that can have the value 0 or 1. **A byte** is defined as **8 bits**. **A nibble** is half a byte, or 4 bits. **A word** is two bytes, or **16 bits**. The display is intended to show the relative size of these units. Of course, they could all be composed of any combination of zeros and ones.

A kilobyte is 2^{10} bytes, which is 1024 bytes. The abbreviation K is often used to represent kilobytes.

A megabyte, or meg as some call it, is 2^{20} bytes. That is a little over 1 million bytes; it is exactly 1,048,576 bytes. Moving rapidly up the scale in size.

A gigabyte is 2^{30} bytes (over 1 billion), and a terabyte is 2^{40} bytes (over 1 trillion).

As an example of how some of these terms are used, suppose that a given computer has 16 megabytes of memory. That would be 16 x 2^{20}, or 2^4 x 2^{20}. which is 2^{24}. Therefore 16 megabytes is 2^{24} bytes.

AVR Microcontroller

0.3 SEMICONDUCTOR MEMORY

Two Types of Memory in Computers

Two types of memory commonly used in microcomputers are RAM, which stands for “Random Access Memory", and ROM, which stands for “|Read-Only Memory."

RAM is used by the computer for temporary storage of programs that it is running. That data is lost when the computer is turned off. For this reason, RAM is sometimes called volatile memory.

ROM contains programs and information essential to operation of the computer. The information in ROM is permanent, cannot be changed by the user, and is not lost when the power is turned off. Therefore, it is called nonvolatile memory.

AVR Microcontroller

0.3 SEMICONDUCTOR MEMORY

Internal organization of computers

The internal working of every computer can be broken down into three parts: CPU (central processing unit), memory, and I/O (input/output) devices. Figure 0-9 shows a block diagram of the internal organization of a computer.

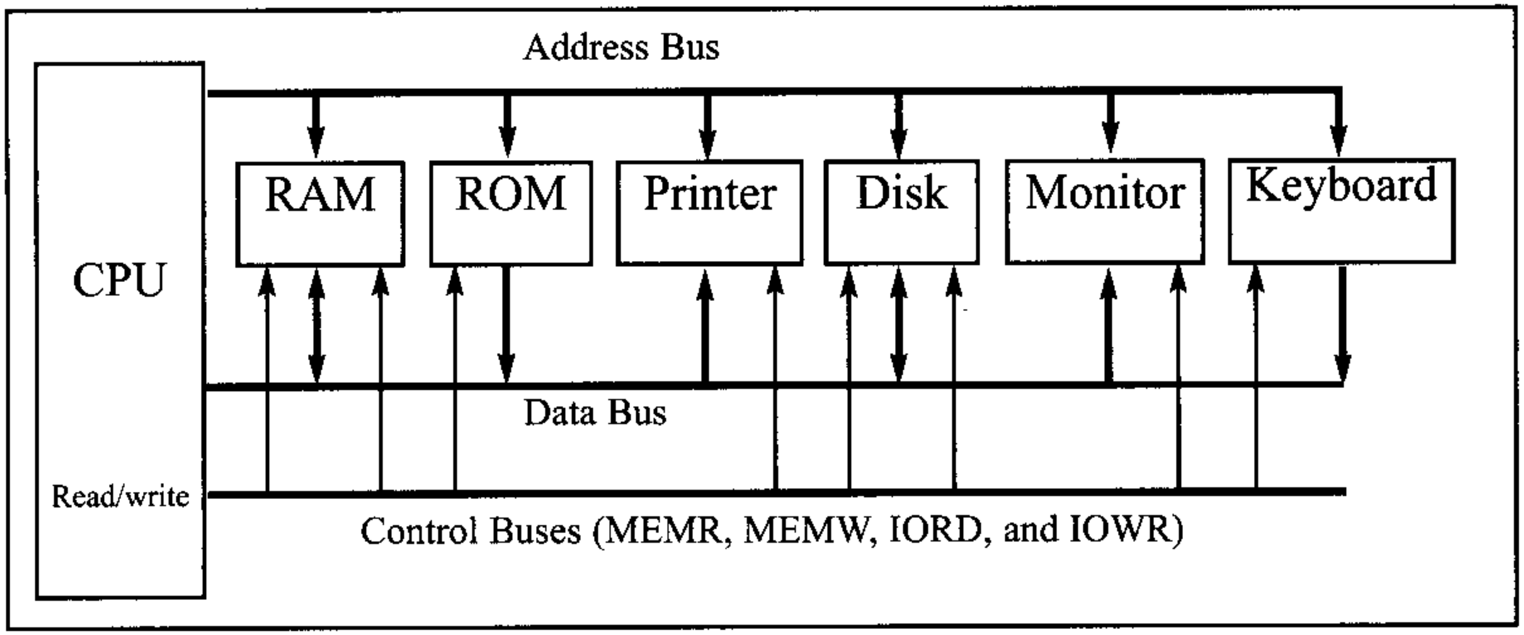


Figure 0-9. Internal Organization of a Computer

AVR Microcontroller
0.3 SEMICONDUCTOR MEMORY

Function of CPU & I/O

The function of the CPU is to execute (process) information stored in memory.

The function of I/O devices such as the keyboard and video monitor is to provide a means of communicating with the CPU.

The CPU is connected to memory and I/O through strips of wire called a bus.

The bus inside a computer allows carrying information from place to place just as a street allows cars to carry people from place to place.

In every computer there are three types of buses: address bus, data bus, and control bus.

AVR Microcontroller

0.3 SEMICONDUCTOR MEMORY

For a device (memory or I/O) to be recognized by the CPU, it must be assigned an address. The address assigned to a given device must be unique; no two devices are allowed to have the same address. The CPU puts the address (in binary) on the address bus, and the decoding circuitry finds the device.

Then the CPU uses the data bus either to get data from that device or to send data to it. The control buses are used to provide read or write signals to the device to indicate if the CPU is asking for information or sending information. Of the three buses, the address bus and data bus determine the capability of a given CPU.

AVR Microcontroller
0.3 SEMICONDUCTOR MEMORY

More about the data bus

Because data buses are used to carry information in and out of a CPU, the more data buses available, the better the CPU.

By the same token, that increase in the number of lanes increases the cost of construction. More data buses mean a more expensive CPU and computer. The average size of data buses in CPUs varies between 8 and 64 bits. Early personal computers such as Apple 2 used an 8-bit data bus, while supercomputers such as Cray used a 64-bit data bus. Data buses are bidirectional, because the CPU must use them either to receive or to send data.

The processing power of a computer is related to the size of its buses, because an 8-bit bus can send out 1 byte a time, but a 16-bit bus can send out 2 bytes at a time, which is twice as fast.

AVR Microcontroller

0.3 SEMICONDUCTOR MEMORY

More about the address bus

Because the address bus is used to identify the devices and memory connected to the CPU, the more address buses available, the larger the number of devices that can be addressed. In other words, the number of address buses for a CPU determines the number of locations with which it can communicate. The number of locations is always equal to 2^x, where x is the number of address lines, regardless of the size of the data bus.

For example, a CPU with 16 address lines can provide a total of 65,536 (2^{16}) or 64K of addressable memory. Each location can have a maximum of 1 byte of data.

As another example, the IBM PC AT uses a CPU with 24 address lines and 16 data lines. Thus, the total accessible memory is 16 megabytes (2^{24} = 16 megabytes). In this example there would be 2^{24} locations, and because each location is one byte, there would be 16 megabytes of memory. The address bus is a unidirectional bus, which means that the CPU uses the address bus only to send out addresses.

AVR Microcontroller

0.3 SEMICONDUCTOR MEMORY

CPU and its relation to RAM and ROM

For the CPU to process information, the data must be stored in RAM or ROM. The function of ROM in computers is to provide information that is fixed and permanent. This is information such as tables for character patterns to be displayed on the video monitor, or programs that are essential to the working of the computer, such as programs for testing and finding the total amount of RAM installed on the system, or for displaying information on the video monitor.

In contrast, RAM stores temporary information that can change with time, such as various versions of the operating system and application packages such as word processing or tax calculation packages. These programs are loaded from the hard drive into RAM to be processed by the CPU. The CPU cannot get the information from the disk directly because the disk is too slow.

AVR Microcontroller
0.3 SEMICONDUCTOR MEMORY

In other words, the CPU first seeks the information to be processed from RAM (or ROM). Only if the data is not there does the CPU seek it from a mass storage device such as a disk, and then it transfers the information to RAM.

For this reason, RAM and ROM are sometimes referred to as primary memory and disks are called secondary memory.

AVR Microcontroller

0.3 SEMICONDUCTOR MEMORY

Memory organization

Memory chips are organized into a number of locations within the IC. Each location can hold 1 bit, 4 bits, 8 bits, or even 16 bits, depending on how it is designed internally. The number of bits that each location within the memory chip can hold is always equal to the number of data pins on the chip.

How many locations exist inside a memory chip? That depends on the number of address pins. The number of locations within a memory IC always equals 2 to the power of the number of address pins. Therefore, the total number of bits that a memory chip can store is equal to the number of locations times the number of data bits per location.

To summarize:

1. A memory chip contains 2^x locations, where x is the number of address pins.
2. Each location contains y bits, where y is the number of data pins on the chip.
3. The entire chip will contain $2^x \times y$ bits, where x is the number of address pins and y is the number of data pins on the pins.

AVR Microcontroller

0.3 SEMICONDUCTOR MEMORY

Speed

One of the most important characteristics of a memory chip is the speed at which its data can be accessed. To access the data, the address is presented to the address pins, the READ pin is activated, and after a certain amount of time has elapsed, the data shows up at the data pins. The shorter this elapsed time, the better, and consequently, the more expensive the memory chip.

The speed of the memory chip is commonly referred to as its access time. The access time of memory chips varies from a few nanoseconds to hundreds of nanoseconds, depending on the IC technology used in the design and fabrication process. The three important memory characteristics of capacity, organization, and access.

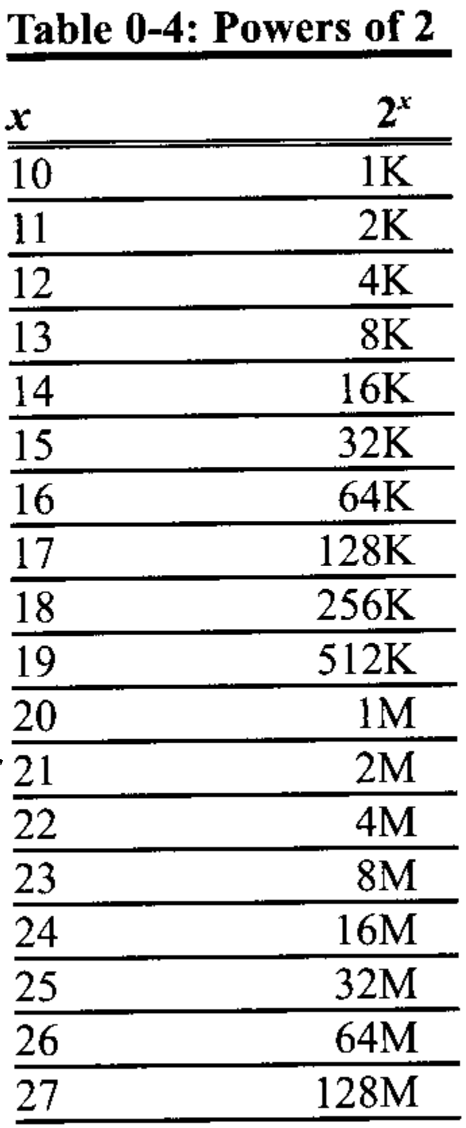


Table 0-4: Powers of 2

x	2^x
10	1K
11	2K
12	4K
13	8K
14	16K
15	32K
16	64K
17	128K
18	256K
19	512K
20	1M
21	2M
22	4M
23	8M
24	16M
25	32M
26	64M
27	128M

AVR Microcontroller

0.3 SEMICONDUCTOR MEMORY

Example 0-12

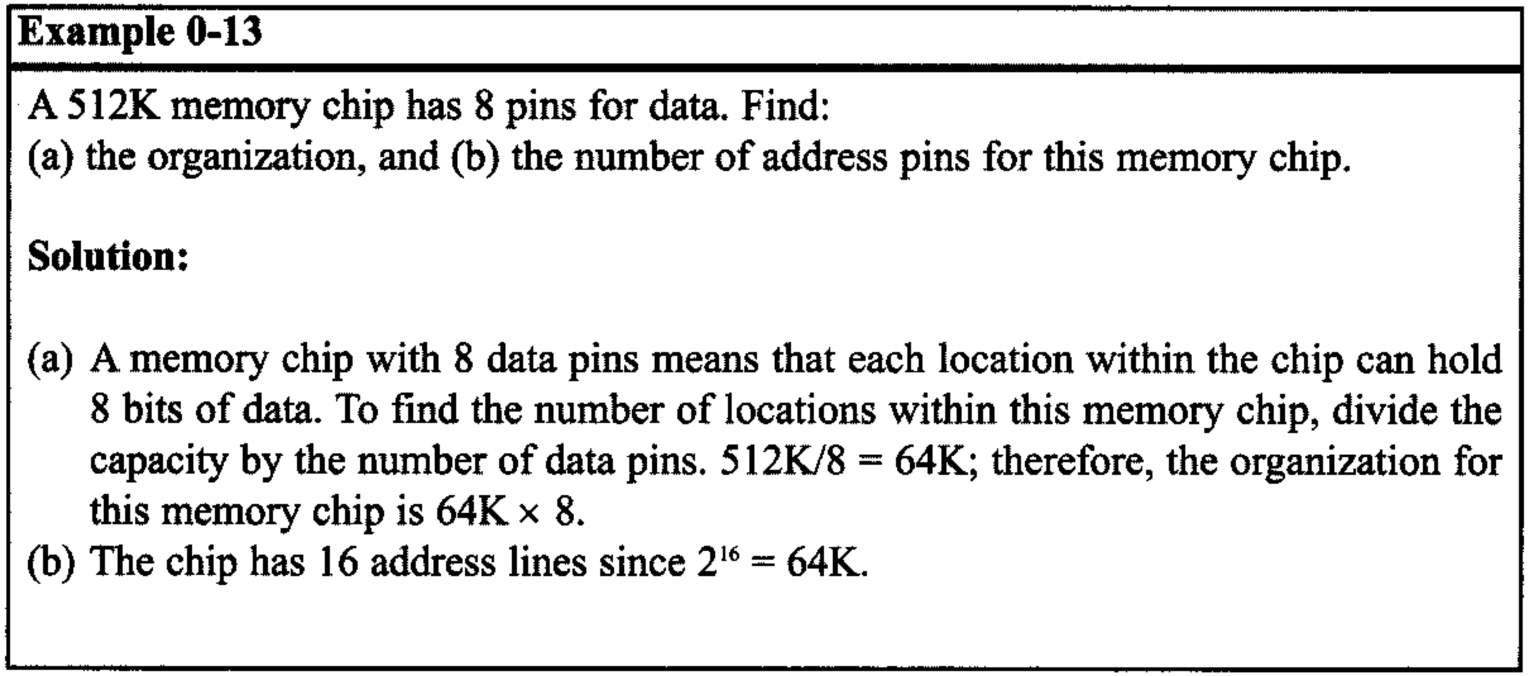
A given memory chip has 12 address pins and 4 data pins. Find:
(a) the organization, and (b) the capacity.

Solution:

(a) This memory chip has 4,096 locations ($2^{12} = 4{,}096$), and each location can hold 4 bits of data. This gives an organization of 4,096 × 4, often represented as 4K × 4.
(b) The capacity is equal to 16K bits since there is a total of 4K locations and each location can hold 4 bits of data.

AVR Microcontroller

0.3 SEMICONDUCTOR MEMORY



Example 0-13

A 512K memory chip has 8 pins for data. Find:
(a) the organization, and (b) the number of address pins for this memory chip.

Solution:

(a) A memory chip with 8 data pins means that each location within the chip can hold 8 bits of data. To find the number of locations within this memory chip, divide the capacity by the number of data pins. 512K/8 = 64K; therefore, the organization for this memory chip is 64K × 8.
(b) The chip has 16 address lines since 2^{16} = 64K.

AVR Microcontroller

0.3 SEMICONDUCTOR MEMORY

ROM (read-only memory)

ROM is a type of memory that does not lose its contents when the power is turned off. For this reason, ROM is also called non-volatile memory. There are different types of read-only memory, such as PROM, EPROM, EEPROM, Flash EPROM, and mask ROM.

PROM (programmable ROM) and OTP

PROM refers to the kind of ROM that the user can burn information into. In other words, PROM is a user-programmable memory. For every bit of the PROM, there exists a fuse. PROM is programmed by blowing the fuses. If the information burned into PROM is wrong, that PROM must be discarded since its internal fuses are blown permanently. For this reason, PROM is also referred to as OTP (one-time programmable). Programming ROM, also called burning ROM, requires special equipment called a ROM burner or ROM programmer.

AVR Microcontroller

0.3 SEMICONDUCTOR MEMORY

EPROM (erasable programmable ROM) and UV-EPROM

EPROM was invented to allow making changes in the contents of PROM after it is burned. In EPROM, one can program the memory chip and erase it thousands of times. A widely used EPROM is called UV-EPROM, where UV stands for ultraviolet. The only problem with UV-EPROM is that erasing its contents can take up to 20 minutes. All UV-EPROM chips have a window through which the programmer can shine ultraviolet (UV) radiation to erase the chip's contents. For this reason, EPROM is also referred to as UV-erasable EPROM or simply UV-EPROM.

AVR Microcontroller

0.3 SEMICONDUCTOR MEMORY

Figure 0-10 shows the pins for UV-EPROM chips. To program a UV-EPROM chip, the following steps must be taken:

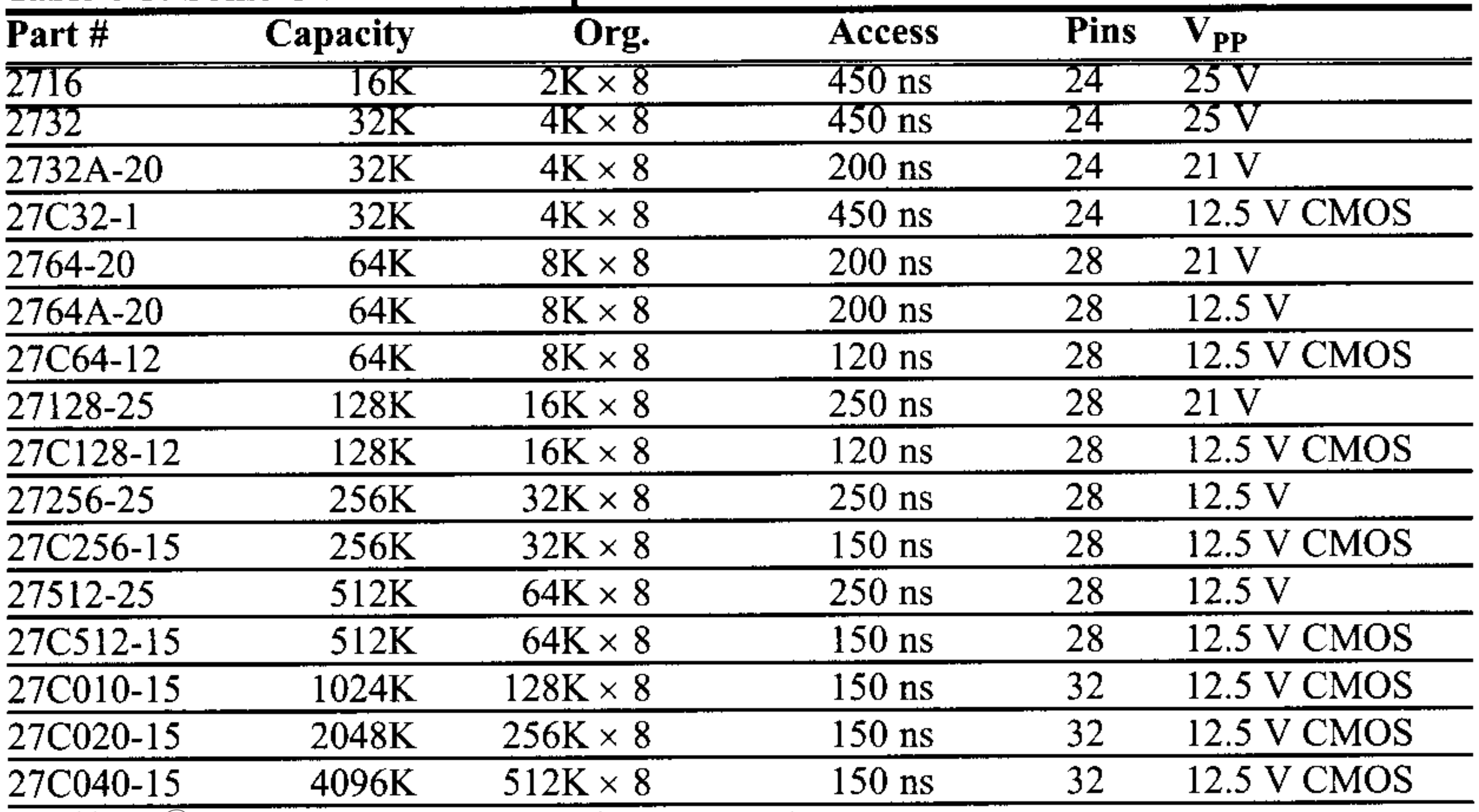
1. Its contents must be erased. To erase a chip, remove it from its socket on the system board and place it in EPROM erasure equipment to expose it to UV radiation for 15-20 minutes.
2. Program the chip. To program a UV-EPROM chip, place it in the ROM burner (programmer). To burn code or data into EPROM, the ROM burner uses 12.5 volts or higher, depending on the EPROM type. This voltage is referred to as Vpp in the UV-EPROM data sheet.
3. Place the chip back into its socket on the system board. As can be seen from the above steps, not only is there an EPROM programmer (burner), but there is also separate EPROM erasure equipment. The main problem, and indeed the major disadvantage of UV-EPROM, is that it cannot be erased and programmed while it is in the system board. To provide a solution to this problem, EEPROM was invented.

AVR Microcontroller

0.3 SEMICONDUCTOR MEMORY

Notice the patterns of the IC numbers in Table 0-5. For example, part number 27128-25 refers to UV-EPROM that has a capacity of 128K bits and access time of 250 nanoseconds.

Table 0-5: Some UV-EPROM Chips

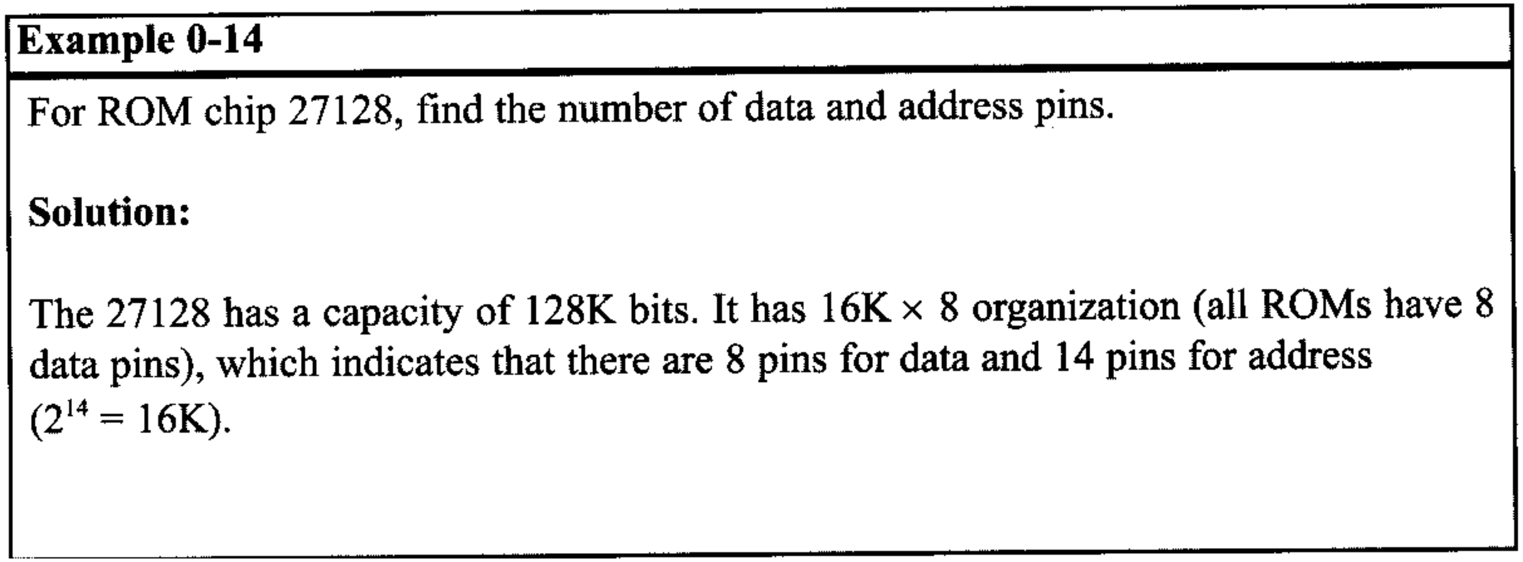


Part #	Capacity	Org.	Access	Pins	V_{PP}
2716	16K	2K × 8	450 ns	24	25 V
2732	32K	4K × 8	450 ns	24	25 V
2732A-20	32K	4K × 8	200 ns	24	21 V
27C32-1	32K	4K × 8	450 ns	24	12.5 V CMOS
2764-20	64K	8K × 8	200 ns	28	21 V
2764A-20	64K	8K × 8	200 ns	28	12.5 V
27C64-12	64K	8K × 8	120 ns	28	12.5 V CMOS
27128-25	128K	16K × 8	250 ns	28	21 V
27C128-12	128K	16K × 8	120 ns	28	12.5 V CMOS
27256-25	256K	32K × 8	250 ns	28	12.5 V
27C256-15	256K	32K × 8	150 ns	28	12.5 V CMOS
27512-25	512K	64K × 8	250 ns	28	12.5 V
27C512-15	512K	64K × 8	150 ns	28	12.5 V CMOS
27C010-15	1024K	128K × 8	150 ns	32	12.5 V CMOS
27C020-15	2048K	256K × 8	150 ns	32	12.5 V CMOS
27C040-15	4096K	512K × 8	150 ns	32	12.5 V CMOS

AVR Microcontroller

0.3 SEMICONDUCTOR MEMORY

See Example 0-14. In part numbers, C refers to CMOS technology. Notice that 27XX always refers to UV-EPROM chips.



Example 0-14

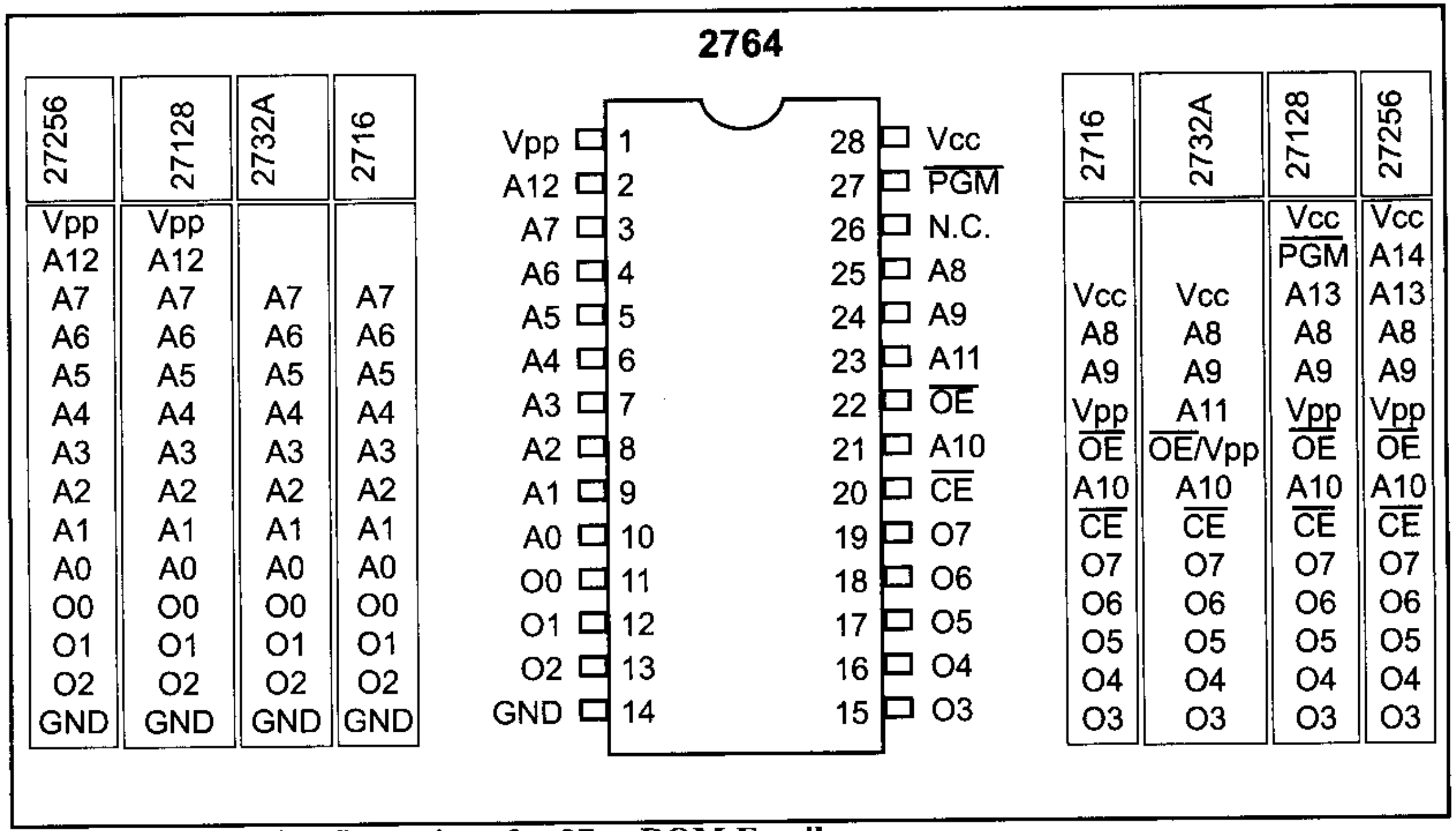
For ROM chip 27128, find the number of data and address pins.

Solution:

The 27128 has a capacity of 128K bits. It has 16K × 8 organization (all ROMs have 8 data pins), which indicates that there are 8 pins for data and 14 pins for address (2^{14} = 16K).

AVR Microcontroller

0.3 SEMICONDUCTOR MEMORY



27256	27128	2732A	2716	2764	Pin	Pin	2764	2716	2732A	27128	27256
Vpp	Vpp			Vpp	1	28	Vcc			Vcc	Vcc
A12	A12			A12	2	27	$\overline{PGM}$			$\overline{PGM}$	A14
A7	A7	A7	A7	A7	3	26	N.C.	Vcc	Vcc	A13	A13
A6	A6	A6	A6	A6	4	25	A8	A8	A8	A8	A8
A5	A5	A5	A5	A5	5	24	A9	A9	A9	A9	A9
A4	A4	A4	A4	A4	6	23	A11	Vpp	A11	Vpp	Vpp
A3	A3	A3	A3	A3	7	22	$\overline{OE}$	$\overline{OE}$	$\overline{OE}$/Vpp	$\overline{OE}$	$\overline{OE}$
A2	A2	A2	A2	A2	8	21	A10	A10	A10	A10	A10
A1	A1	A1	A1	A1	9	20	$\overline{CE}$	$\overline{CE}$	$\overline{CE}$	$\overline{CE}$	$\overline{CE}$
A0	A0	A0	A0	A0	10	19	O7	O7	O7	O7	O7
O0	O0	O0	O0	O0	11	18	O6	O6	O6	O6	O6
O1	O1	O1	O1	O1	12	17	O5	O5	O5	O5	O5
O2	O2	O2	O2	O2	13	16	O4	O4	O4	O4	O4
GND	GND	GND	GND	GND	14	15	O3	O3	O3	O3	O3

Figure 0-10. Pin Configurations for 27xx ROM Family

AVR Microcontroller

0.3 SEMICONDUCTOR MEMORY

EEPROM (electrically erasable programmable ROM)

EEPROM has several advantages over EPROM, such as the fact that its method of erasure is electrical and therefore instant, as opposed to the 20-minute erasure time required for UV-EPROM.

In addition, in EEPROM one can select which byte to be erased, in contrast to UV-EPROM, in which the entire contents of ROM are erased.

However, the main advantage of EEPROM is that one can program and erase its contents while it is still in the system board.

In other words, unlike UV-EPROM, EEPROM does not require an external erasure and programming device. To utilize EEPROM fully, the designer must incorporate the circuitry to program the EEPROM into the system board.

In general, the cost per bit for EEPROM is much higher than for UV-EPROM.

AVR Microcontroller

0.3 SEMICONDUCTOR MEMORY

Flash memory EPROM

Since the early 1990s, Flash EPROM has become a popular user-programmable memory chip, and for good reasons. First, the erasure of the entire contents takes less than a second, or one might say in a flash, hence its name, Flash memory. In addition, the erasure method is electrical, and for this reason it is sometimes referred to as Flash EEPROM.

The major difference between EEPROM and Flash memory is that when Flash memory's contents are erased, the entire device is erased, in contrast to EEPROM, where one can erase a desired byte. Although in many Flash memories recently made available the contents are divided into blocks and the erasure can be done block by block, unlike EEPROM, Flash memory has no byte erasure option. Because Flash memory can be programmed while it is in its socket on the system board, it is widely used to upgrade the BIOS ROM of the PC. Some designers believe that Flash memory will replace the hard disk as a mass storage medium.

AVR Microcontroller

0.3 SEMICONDUCTOR MEMORY

This would increase the performance of the computer tremendously, since Flash memory is semiconductor memory with access time in the range of 100 ns compared with disk access time in the range of tens of milliseconds.

For this to happen, Flash memory's program/erase cycles must become infinite, just like hard disks. Program/erase cycle refers to the number of times that a chip can be erased and reprogrammed before it becomes unusable. At this time, the program/erase cycle is 100,000 for Flash and EEPROM, 1000 for UV-EPROM, and infinite for RAM and disks.

AVR Microcontroller

0.3 SEMICONDUCTOR MEMORY

See Table 0-6 for some sample chips.

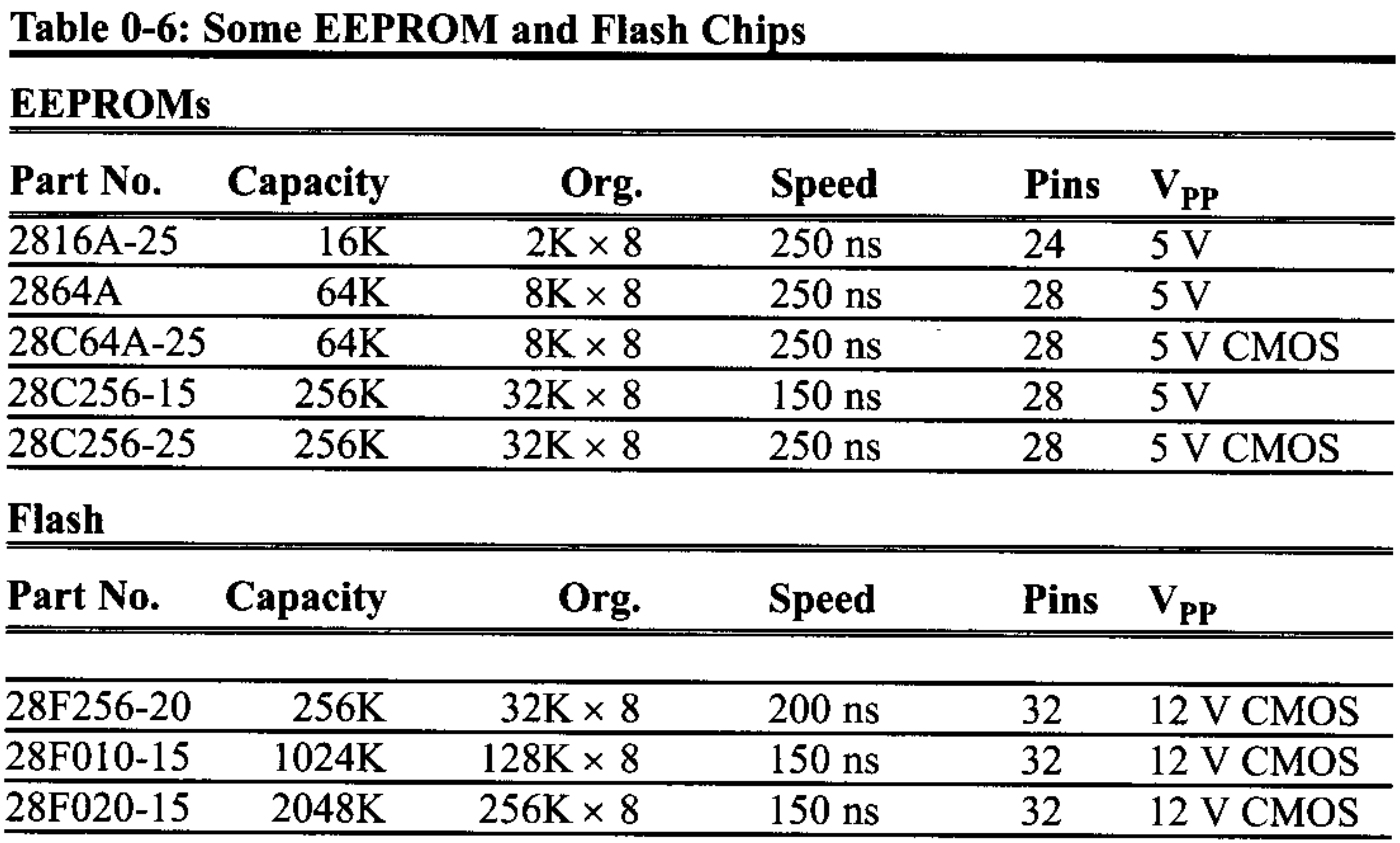


Table 0-6: Some EEPROM and Flash Chips

EEPROMs

Part No.	Capacity	Org.	Speed	Pins	V_{PP}
2816A-25	16K	2K × 8	250 ns	24	5 V
2864A	64K	8K × 8	250 ns	28	5 V
28C64A-25	64K	8K × 8	250 ns	28	5 V CMOS
28C256-15	256K	32K × 8	150 ns	28	5 V
28C256-25	256K	32K × 8	250 ns	28	5 V CMOS

Flash

Part No.	Capacity	Org.	Speed	Pins	V_{PP}
28F256-20	256K	32K × 8	200 ns	32	12 V CMOS
28F010-15	1024K	128K × 8	150 ns	32	12 V CMOS
28F020-15	2048K	256K × 8	150 ns	32	12 V CMOS

AVR Microcontroller

0.3 SEMICONDUCTOR MEMORY

Mask ROM

Mask ROM refers to a kind of ROM in which the contents are programmed by the IC manufacturer. In other words, it is not a user-programmable ROM. The term mask is used in IC fabrication. Since the process is costly, mask ROM is used when the needed volume is high (hundreds of thousands) and it is absolutely certain that the contents will not change.

It is common practice to use UV-EPROM or Flash for the development phase of a project, and only after the code/data have been finalized is the mask version of the product ordered. The main advantage of mask ROM is its cost, since it is significantly cheaper than other kinds of ROM, but if an error is found in the data/code, the entire batch must be thrown away. It must be noted that all ROM memories have 8 bits for data pins; therefore, the organization is x 8.

AVR Microcontroller
0.3 SEMICONDUCTOR MEMORY

RAM (random access memory)

RAM memory is called volatile memory since cutting off the power to the IC results in the loss of data. Sometimes RAM is also referred to as RAWM (read and write memory), in contrast to ROM, which cannot be written to. There are three types of RAM:

1- static RAM (SRAM),

2- NV-RAM (nonvolatile RAM), and

3- dynamic RAM (DRAM).

AVR Microcontroller

0.3 SEMICONDUCTOR MEMORY

SRAM (static RAM)

Storage cells in static RAM memory are made of flip-flops and therefore do not require refreshing in order to keep their data. This is in contrast to DRAM. The problem with the use of flip-flops for storage cells is that each cell requires at least 6 transistors to build, and the cell holds only 1 bit of data. In recent years, the cells have been made of 4 transistors, which still is too many. The use of 4-transistor cells plus the use of CMOS technology has given birth to a high-capacity SRAM, but its capacity is far below DRAM. Figure 0-11 shows the pin diagram for an SRAM chip.

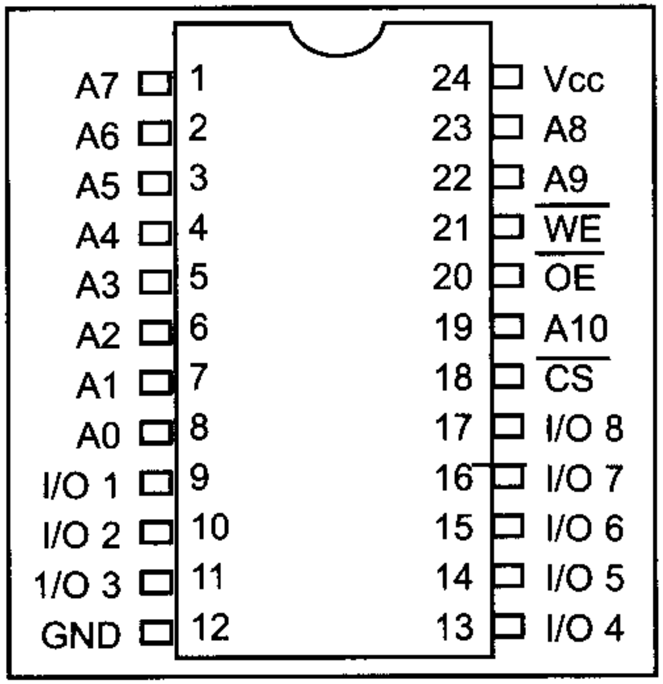


Figure 0-11. 2K × 8 SRAM Pins

AVR Microcontroller
0.3 SEMICONDUCTOR MEMORY

The following is a description of the 6116 SRAM pins. A0-A10 are for address inputs, where 11 address lines gives $2^{11} = 2K$.
WE (write enable) is for writing data into SRAM (active low).
OE (output enable) is for reading data out of SRAM (active low)
CS (chip select) is used to select the memory chip.
I/O0-I/O7 are for data I/O, where 8-bit data lines give an organization of 2K x 8.

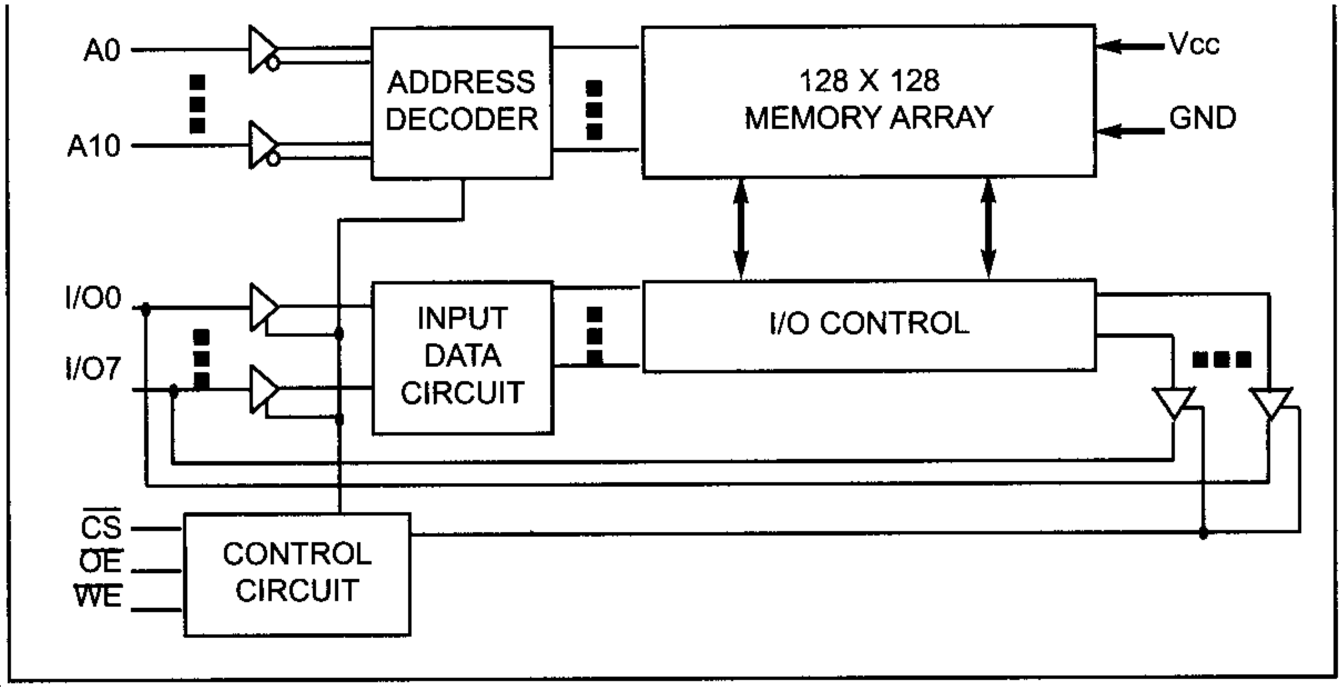


Figure 0-12. Functional Block Diagram for 6116 SRAM

AVR Microcontroller

0.3 SEMICONDUCTOR MEMORY

Figure 0-13 shows the following steps to write data into SRAM.

1. Provide the addresses to pins A0-A10.
2. Activate the CS pin.
3. Make WE = 0 while RD = 1.
4. Provide the data to pins I/O0-I/O7.
5. Make WE = 1 and data will be written into SRAM on the positive edge of the WE signal.

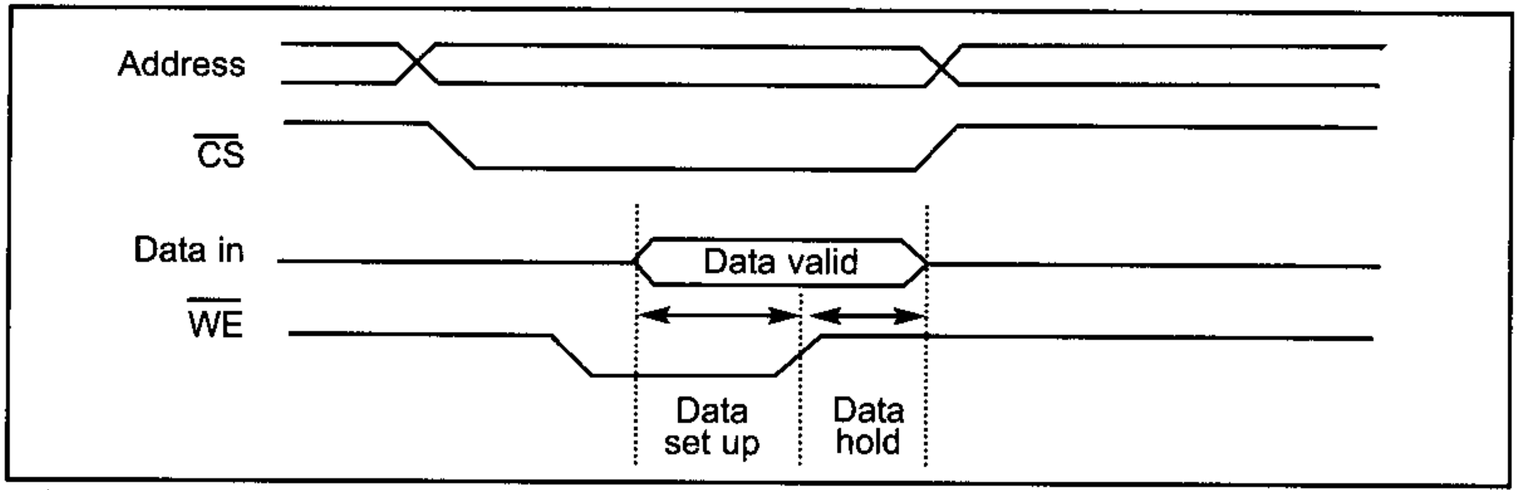


Figure 0-13. Memory Write Timing for SRAM

AVR Microcontroller

0.3 SEMICONDUCTOR MEMORY

The following are steps to read data from SRAM. See Figure 0-14.

1. Provide the addresses to pins A0-A10. This is the start of the access time (t_{AA}).
2. Activate the CS pin.
3. 3. While WE = 1, a high-to-low pulse on the OE pin will read the data out of the chip.

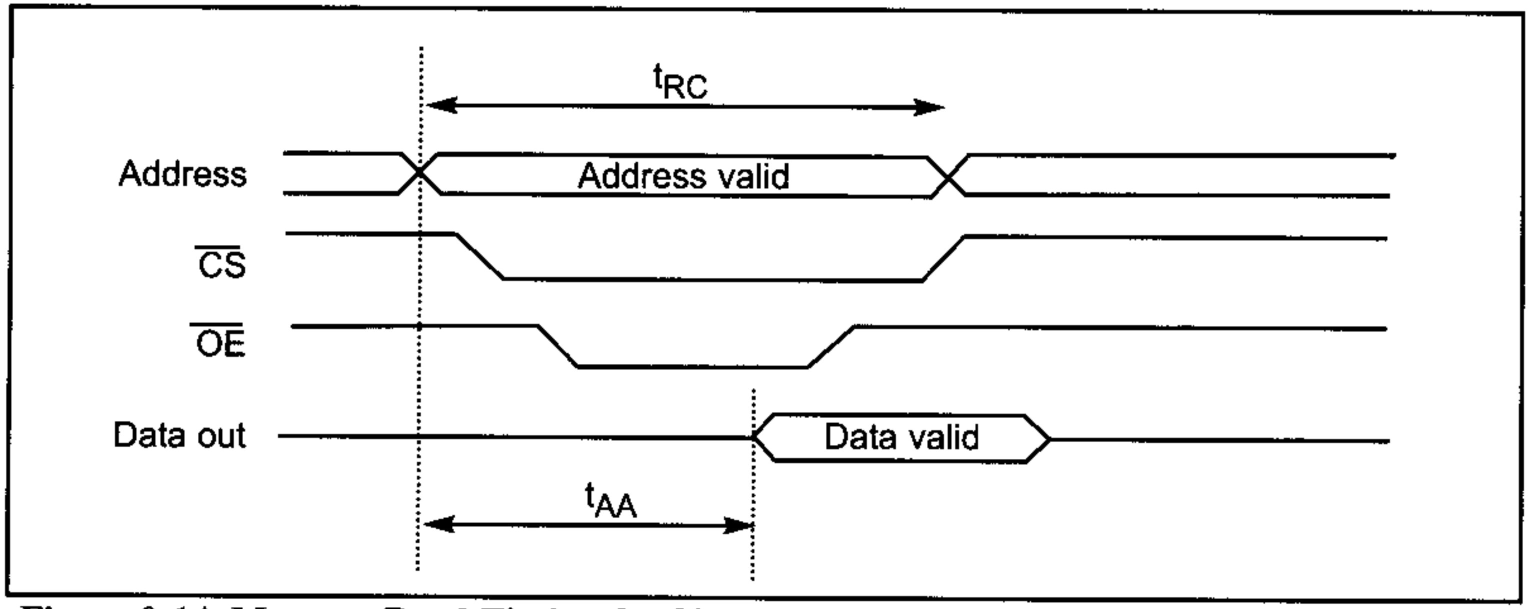


Figure 0-14. Memory Read Timing for SRAM

AVR Microcontroller

0.3 SEMICONDUCTOR MEMORY

NV-RAM (nonvolatile RAM)

Whereas SRAM is volatile, there is a new type of nonvolatile RAM called NV-RAM. Like other RAMs, it allows the CPU to read and write to it, but when the power is turned off the contents are not lost. NV-RAM combines the best of RAM and ROM: the read and write ability of RAM, plus the nonvolatility of ROM. To retain its contents, every NV-RAM chip internally is made of the following components:

1. It uses extremely power-efficient (very low-power consumption) SRAM cells built out of CMOS.
2. It uses an internal lithium battery as a backup energy source.
3. It uses an intelligent control circuitry. The main job of this control circuitry is to monitor the Vcc pin constantly to detect loss of the external power supply. If the power to the Vcc pin falls below out-of-tolerance conditions, the control circuitry switches automatically to its internal power source, the lithium battery. The internal lithium power source is used to retain the NV-RAM contents only when the external power source is off.

AVR Microcontroller

0.3 SEMICONDUCTOR MEMORY

It must be emphasized that all three of the components above are incorporated into a single IC chip, and for this reason nonvolatile RAM is a very expensive type of RAM as far as cost per bit is concerned. Offsetting the cost, however, is the fact that it can retain its contents up to ten years after the power has been turned off and allows one to read and write in exactly the same way as SRAM

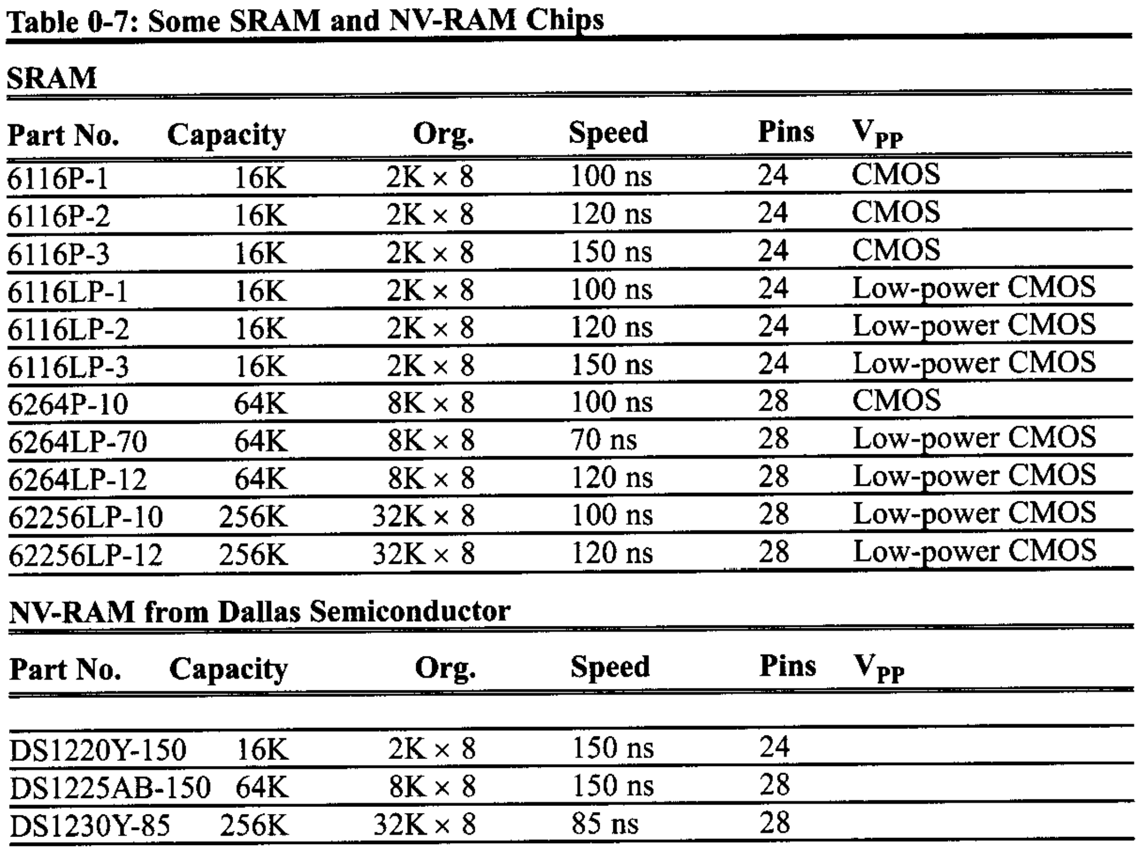


Table 0-7: Some SRAM and NV-RAM Chips

SRAM

Part No.	Capacity	Org.	Speed	Pins	V_{PP}
6116P-1	16K	2K × 8	100 ns	24	CMOS
6116P-2	16K	2K × 8	120 ns	24	CMOS
6116P-3	16K	2K × 8	150 ns	24	CMOS
6116LP-1	16K	2K × 8	100 ns	24	Low-power CMOS
6116LP-2	16K	2K × 8	120 ns	24	Low-power CMOS
6116LP-3	16K	2K × 8	150 ns	24	Low-power CMOS
6264P-10	64K	8K × 8	100 ns	28	CMOS
6264LP-70	64K	8K × 8	70 ns	28	Low-power CMOS
6264LP-12	64K	8K × 8	120 ns	28	Low-power CMOS
62256LP-10	256K	32K × 8	100 ns	28	Low-power CMOS
62256LP-12	256K	32K × 8	120 ns	28	Low-power CMOS

NV-RAM from Dallas Semiconductor

Part No.	Capacity	Org.	Speed	Pins	V_{PP}
DS1220Y-150	16K	2K × 8	150 ns	24	
DS1225AB-150	64K	8K × 8	150 ns	28	
DS1230Y-85	256K	32K × 8	85 ns	28	

AVR Microcontroller
0.3 SEMICONDUCTOR MEMORY

DRAM (dynamic RAM)

Since the early days of the computer, the need for huge, inexpensive read/write memory has been a major preoccupation of computer designers. In 1970, Intel Corporation introduced the first dynamic RAM (random access memory). Its density (capacity) was 1024 bits and it used a capacitor to store each bit. Using a capacitor to store data cuts down the number of transistors needed to build the cell; however, it requires constant refreshing due to leakage. This is in contrast to SRAM (static RAM), whose individual cells are made of flip-flops. Since each bit in SRAM uses a single flip-flop, and each flip-flop requires six transistors, SRAM has much larger memory cells and consequently lower density. The use of capacitors as storage cells in DRAM results in much smaller net memory cell size.

AVR Microcontroller
0.3 SEMICONDUCTOR MEMORY

The advantages and disadvantages of DRAM memory can be summarized as follows.

The major advantages are high density (capacity), cheaper cost per bit, and lower power consumption per bit.

The disadvantage is that it must be refreshed periodically because the capacitor cell loses its charge; furthermore, while DRAM is being refreshed, the data cannot be accessed. This is in contrast to SRAM's flip-flops, which retain data as long as the power is on, do not need to be refreshed, and whose contents can be accessed at any time.

Since 1970, the capacity of DRAM has exploded. After the 1K-bit (1024) chip came the 4K-bit in 1973, and then the 16K chip in 1976. The 1980s saw the introduction of 64K, 256K, and finally 1M and 4M memory chips. The 1990s saw 16M, 64M, 256M, and the beginning of 10-bit DRAM chips.

In the 2000s, 2G-bit chips are standard, and as the fabrication process gets smaller, larger memory chips will be rolling off the manufacturing line. Keep in mind that when talking about IC memory chips, the capacity is always assumed to be in bits. Therefore, a 1M chip means a 1-megabit chip and a 256K chip means a 256K-bit memory chip. However, when talking about the memory of a computer system, it is always assumed to be in bytes.

AVR Microcontroller

0.3 SEMICONDUCTOR MEMORY

Packaging issue in DRAM

In DRAM there is a problem of packing a large number of cells into a single chip with the normal number of pins assigned to addresses. For example, a 64K-bit chip (64K x 1) must have 16 address lines and 1 data line, requiring 16 pins to send in the address if the conventional method is used. This is in addition to Vcc power, ground, and read/write control pins. Using the conventional method of data access, the large number of pins defeats the purpose of high density and small packaging, so dearly cherished by IC designers. Therefore, to reduce the number of pins needed for addresses, multiplexing/demultiplexing is used.

The method used is to split the address in half and send in each half of the address through the same pins, thereby requiring fewer address pins. Internally, the DRAM structure is divided into a square of rows and columns. The first half of the address is called the row and the second half is called the column.

AVR Microcontroller
0.3 SEMICONDUCTOR MEMORY

For example, in the case of DRAM of 64K x 1 organization, the first half of the address is sent in through the 8 pins A0-A7, and by activating RAS (row address strobe), the internal latches inside DRAM grab the first half of the address. After that, the second half of the address is sent in through the same pins, and by activating CAS (column address strobe), the internal latches inside DRAM latch the second half of the address. This results in using 8 pins for addresses plus RAS and CAS, for a total of 10 pins, results in using 8 pins for addresses plus RAS and CAS, for a total of IO pins, instead of the 16 pins that would be required without multiplexing. To access a bit of data from DRAM, both row and column addresses must be provided. For this concept to work, there must be a 2-by-1 multiplexer outside the DRAM circuitry and a demultiplexer inside every DRAM chip. Due to the complexities associated with DRAM interfacing (RAS, CAS, the need for multiplexer and refreshing circuitry), some DRAM controllers are designed to make DRAM interfacing much easier. However, many small microcontroller-based projects that do not require much RAM (usually less than 64K bytes) use SRAM of types EEPROM and NV-RAM, instead of DRAM.

AVR Microcontroller

0.3 SEMICONDUCTOR MEMORY

DRAM organization

In the discussion of ROM, we noted that all of these chips have 8 pins for data. This is not the case for DRAM memory chips, which can have x1, x4, x8, or x16 organizations.

In memory chips, the data pins are also called I/O. In some DRAMs there are separate Din and Dout pins. Figure 0-15 shows a 256K x 1 DRAM chip with pins A0-A8 for address, RAS and CAS, WE (write enable),and data in and data out as well as power and ground.

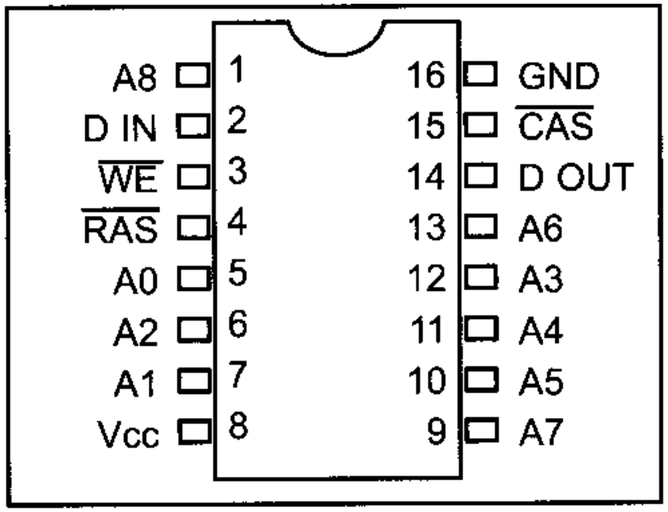


Figure 0-15. 256K × 1 DRAM

AVR Microcontroller

0.3 SEMICONDUCTOR MEMORY

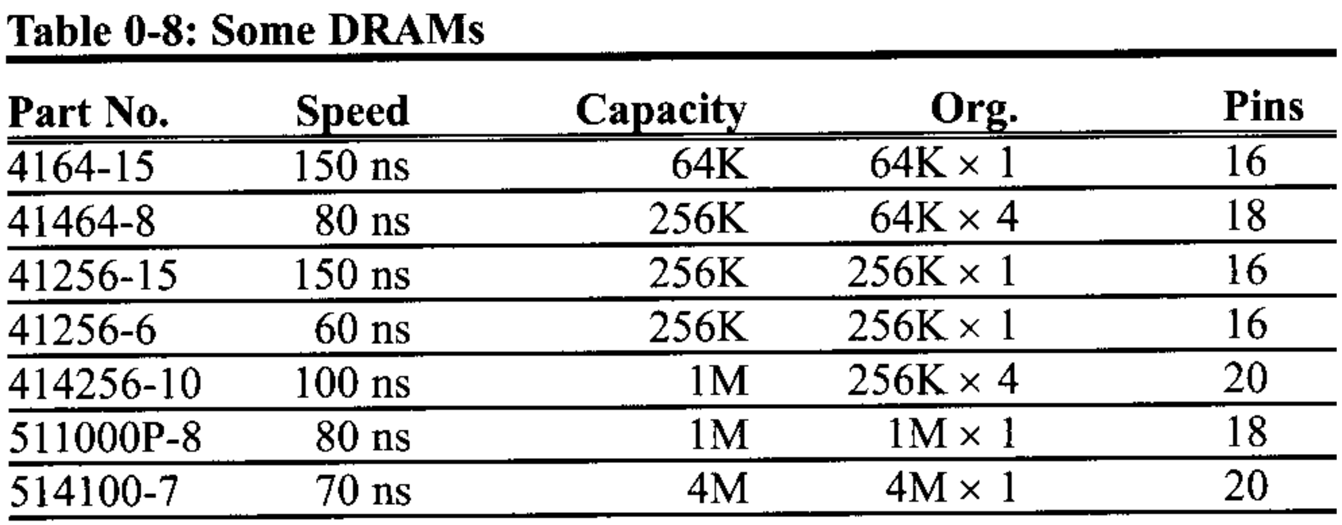
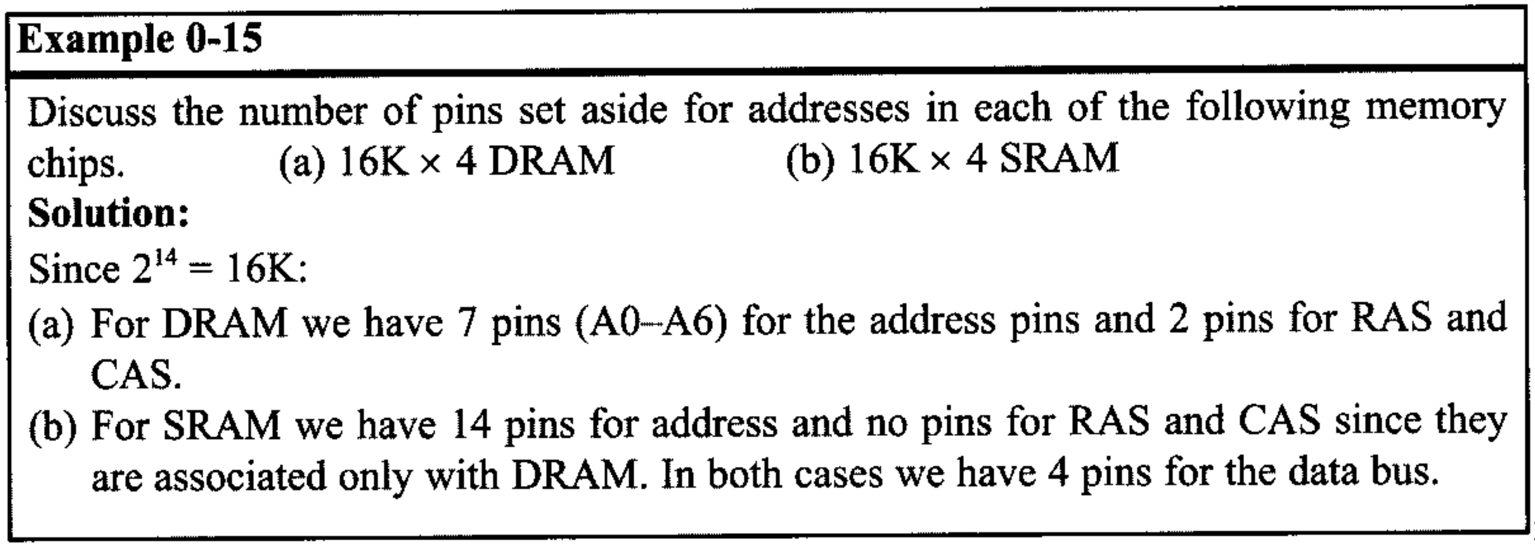


Table 0-8: Some DRAMs

Part No.	Speed	Capacity	Org.	Pins
4164-15	150 ns	64K	64K × 1	16
41464-8	80 ns	256K	64K × 4	18
41256-15	150 ns	256K	256K × 1	16
41256-6	60 ns	256K	256K × 1	16
414256-10	100 ns	1M	256K × 4	20
511000P-8	80 ns	1M	1M × 1	18
514100-7	70 ns	4M	4M × 1	20



Example 0-15

Discuss the number of pins set aside for addresses in each of the following memory chips. (a) 16K × 4 DRAM (b) 16K × 4 SRAM

Solution:

Since $2^{14} = 16K$:

(a) For DRAM we have 7 pins (A0–A6) for the address pins and 2 pins for RAS and CAS.

(b) For SRAM we have 14 pins for address and no pins for RAS and CAS since they are associated only with DRAM. In both cases we have 4 pins for the data bus.

AVR Microcontroller

0.3 SEMICONDUCTOR MEMORY

Memory address decoding

The CPU provides the address of the data desired, but it is the job of the decoding circuitry to locate the selected memory block. To explore the concept of decoding circuitry, we look at various methods used in decoding the addresses. In this discussion we use SRAM or ROM for the sake of simplicity. Memory chips have one or more pins called CS (chip select), which must be activated for the memory's contents to be accessed. Sometimes the chip select is also referred to as chip enable (CE). In connecting a memory chip to the CPU, note the following points:

1. The data bus of the CPU is connected directly to the data pins of the memory chip.
2. Control signals RD (read) and WR (memory write) from the CPU are connected to the OE (output enable) and WE (write enable) pins of the memory chip, respectively.
3. In the case of the address buses, while the lower bits of the addresses from the CPU go directly to the memory chip address pins, the upper ones are used to activate the CS pin of the memory chip. It is the CS pin that along with RD/WR allows the flow of data in or out of the memory chip. No data can be written into or read from the memory chip unless CS is activated.

AVR Microcontroller

0.3 SEMICONDUCTOR MEMORY

As can be seen from the data sheets of SRAM and ROM, the CS input of a memory chip is normally active low and is activated by the output of the memory decoder. Normally memories are divided into blocks, and the output of the decoder selects a given memory block. There are three ways to generate a memory block selector:

(a) using simple logic gates,
(b) using the 74LS138, or
(c) using programmable logics such as CPLD and FPGA.

AVR Microcontroller

0.3 SEMICONDUCTOR MEMORY

Simple logic gate address decoder

The simplest method of constructing decoding circuitry is the use of a NAND gate. The output of a NAND gate is active low, and the CS pin is also active low, which makes them a perfect match. In cases where the CS input is active high, an AND gate must be used. Using a combination of NAND gates and inverters, one can decode any address range. An example of this is shown in Figure 0-16, which shows that A15-A12 must be 0011 in order to select the chip. This results in the assignment of addresses 3000H to 3FFFH to this memory chip.

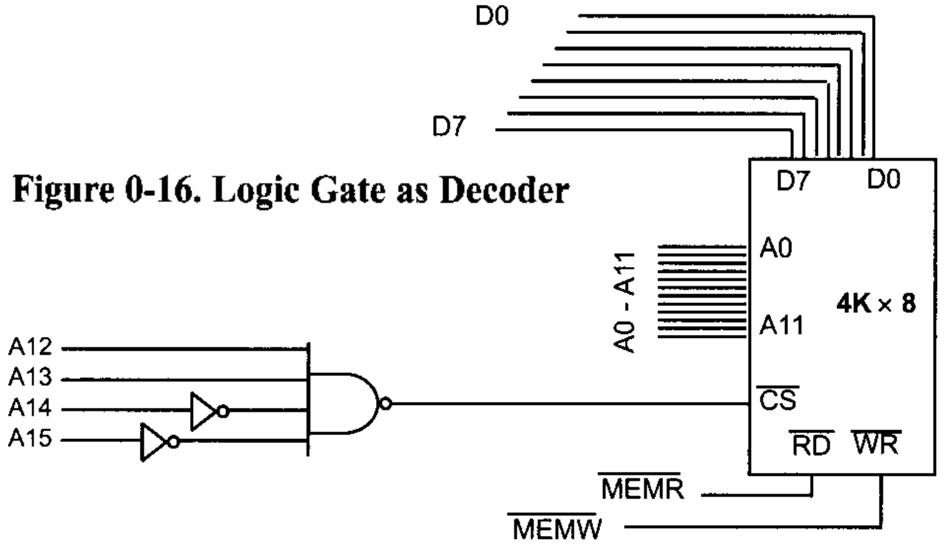


Figure 0-16. Logic Gate as Decoder

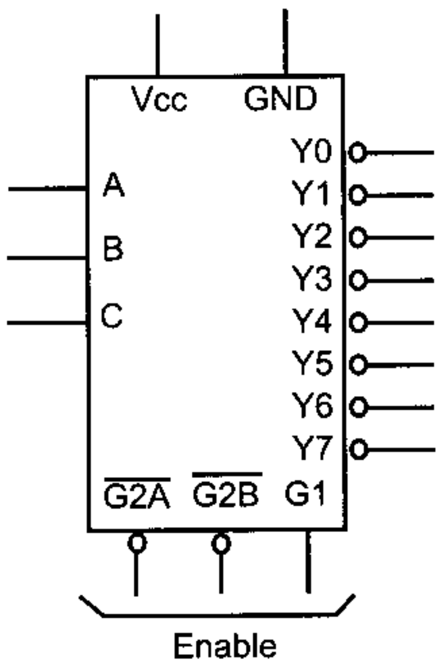
AVR Microcontroller

0.3 SEMICONDUCTOR MEMORY

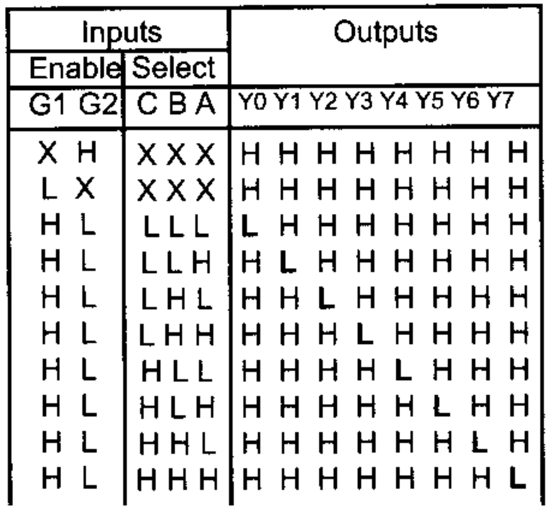
Using the 74LS138 3-8 decoder

This used to be one of the most widely used address decoders. The 3 inputs A, B, and C generate 8 active-low outputs Y0-Y7. Each Y output is connected to CS of a memory chip, allowing control of 8 memory blocks by a single 74LS138. In the 74LS138, where A, B, and C select which output is activated, there are three additional inputs, G2A, G2B, and Gl. G2A and G2B are both active low, and G1 is active high. If any one of the inputs G1, G2A, or G2B is not connected to an address signal (sometimes they are connected to a control signal), they must be activated permanently by either Vcc or ground.

Block Diagram



Function Table



Inputs					Outputs							
Enable		Select										
G1	G2	C	B	A	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7
X	H	X	X	X	H	H	H	H	H	H	H	H
L	X	X	X	X	H	H	H	H	H	H	H	H
H	L	L	L	L	L	H	H	H	H	H	H	H
H	L	L	L	H	H	L	H	H	H	H	H	H
H	L	L	H	L	H	H	L	H	H	H	H	H
H	L	L	H	H	H	H	H	L	H	H	H	H
H	L	H	L	L	H	H	H	H	L	H	H	H
H	L	H	L	H	H	H	H	H	H	L	H	H
H	L	H	H	L	H	H	H	H	H	H	L	H
H	L	H	H	H	H	H	H	H	H	H	H	L

Figure 0-17. 74LS138 Decoder

AVR Microcontroller

0.3 SEMICONDUCTOR MEMORY

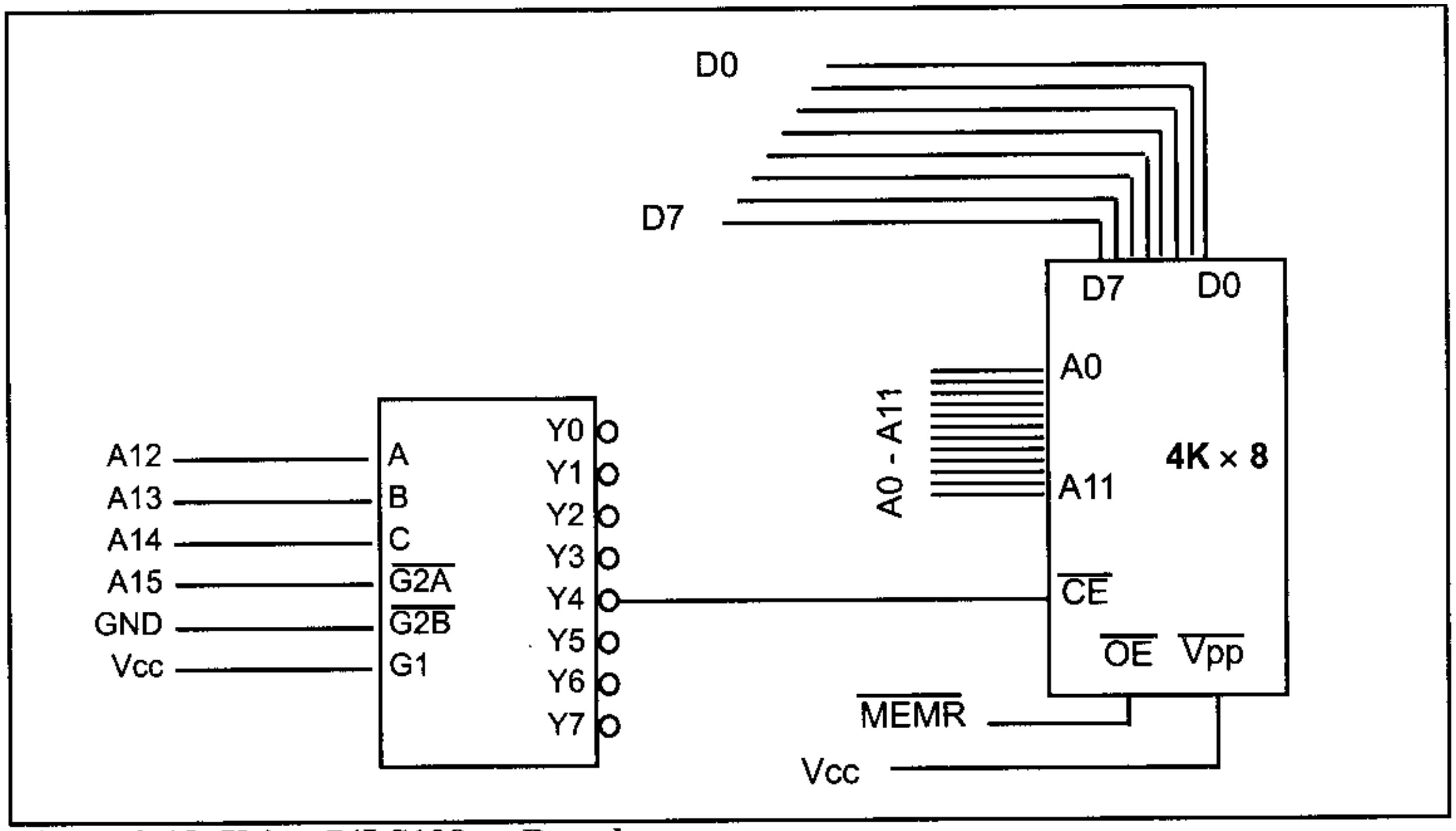
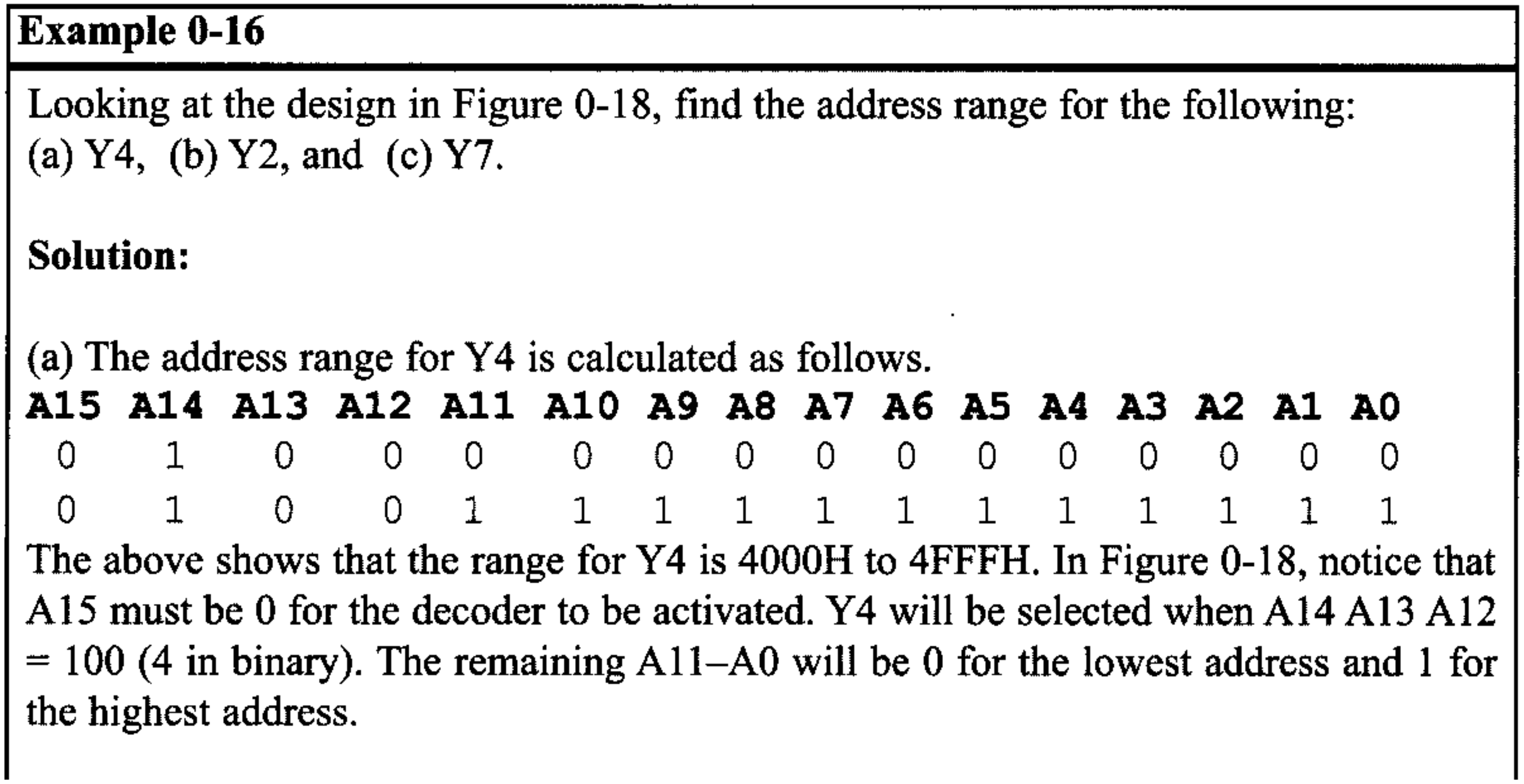


Figure 0-18. Using 74LS138 as Decoder

AVR Microcontroller

0.3 SEMICONDUCTOR MEMORY



Example 0-16

Looking at the design in Figure 0-18, find the address range for the following:
(a) Y4, (b) Y2, and (c) Y7.

Solution:

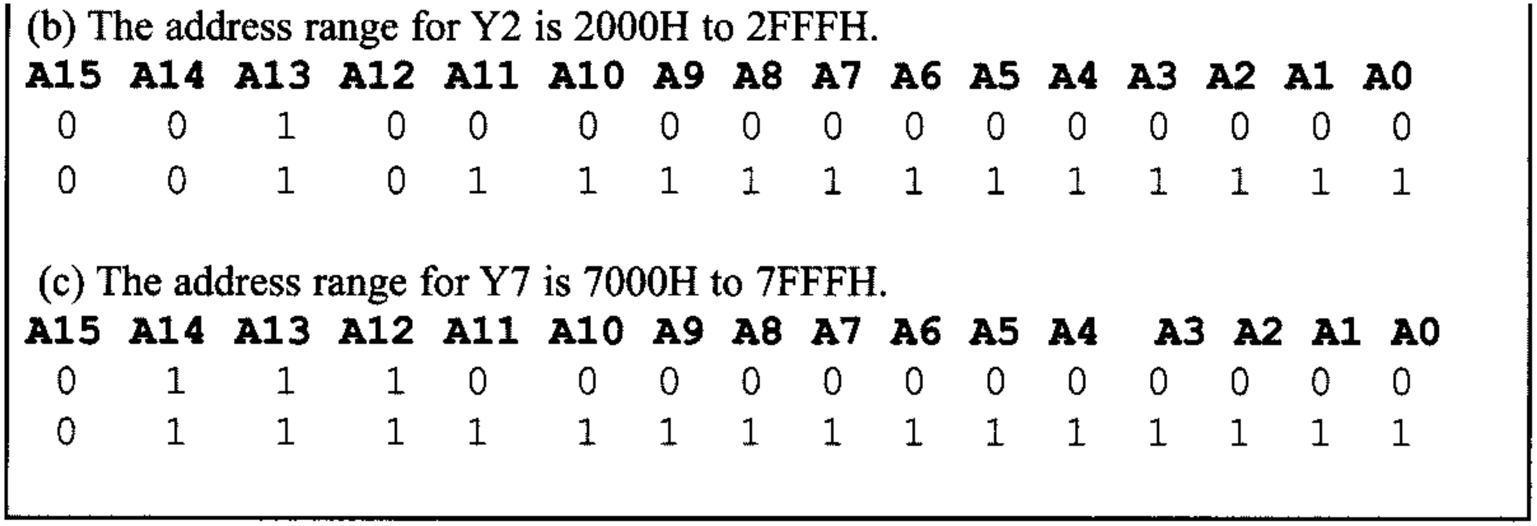
(a) The address range for Y4 is calculated as follows.

A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1

The above shows that the range for Y4 is 4000H to 4FFFH. In Figure 0-18, notice that A15 must be 0 for the decoder to be activated. Y4 will be selected when A14 A13 A12 = 100 (4 in binary). The remaining A11–A0 will be 0 for the lowest address and 1 for the highest address.

AVR Microcontroller

0.3 SEMICONDUCTOR MEMORY



(b) The address range for Y2 is 2000H to 2FFFH.

A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1

(c) The address range for Y7 is 7000H to 7FFFH.

A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

AVR Microcontroller
0.3 SEMICONDUCTOR MEMORY

Using programmable logic as an address decoder

Other widely used decoders are programmable logic chips such as PAL, GAL, and FPGA chips. One disadvantage of these chips is that they require PAL/GAL/FPGA software and a burner (programmer), whereas the 74LS138 needs neither of these. The advantage of these chips is that they can be programmed for any combination of address ranges, and so are much more versatile. This plus the fact that PAL/GAL/FPGA chips have 10 or more inputs (in contrast to 6 in the 74138) means that they can accommodate more address inputs.

AVR Microcontroller

0.4 CPU ARCHITECTURE

Inside CPU

A program stored in memory provides instructions to the CPU to perform an action. The action can simply be adding data such as payroll data or controlling a machine such as a robot. The function of the CPU is to fetch these instructions from memory and execute them. To perform the actions of fetch and execute, all CPUs are equipped with resources such as the following:

1. Foremost among the resources at the disposal of the CPU are a number of registers. The CPU uses registers to store information temporarily. The information could be two values to be processed, or the address of the value needed to be fetched from memory. Registers inside the CPU can be 8-bit, 16-bit, 32-bit, or even 64-bit registers, depending on the CPU. In general, the more and bigger the registers, the better the CPU. The disadvantage of more and bigger registers is the increased cost of such a CPU.
2. 2. The CPU also has what is called the ALU (arithmetic/logic unit). The ALU section of the CPU is responsible for performing arithmetic functions such as add, subtract, multiply, and divide, and logic functions such as AND, OR, and NOT.

AVR Microcontroller

0.4 CPU ARCHITECTURE

3. Every CPU has what is called a Program Counter. The function of the program counter is to point to the address of the next instruction to be executed. As each instruction is executed, the program counter is incremented to point to the address of the next instruction to be executed. The contents of the program counter are placed on the address bus to find and fetch the desired instruction. In the IBM PC, the program counter is a register called IP, or the Instruction Pointer.
4. The function of the instruction decoder is to interpret the instruction fetched into the CPU. One can think of the instruction decoder as a kind of dictionary, storing the meaning of each instruction and what steps the CPU should take upon receiving a given instruction. Just as a dictionary requires more pages the more words it defines, a CPU capable of understanding more instructions requires more transistors to design.

AVR Microcontroller

0.4 CPU ARCHITECTURE

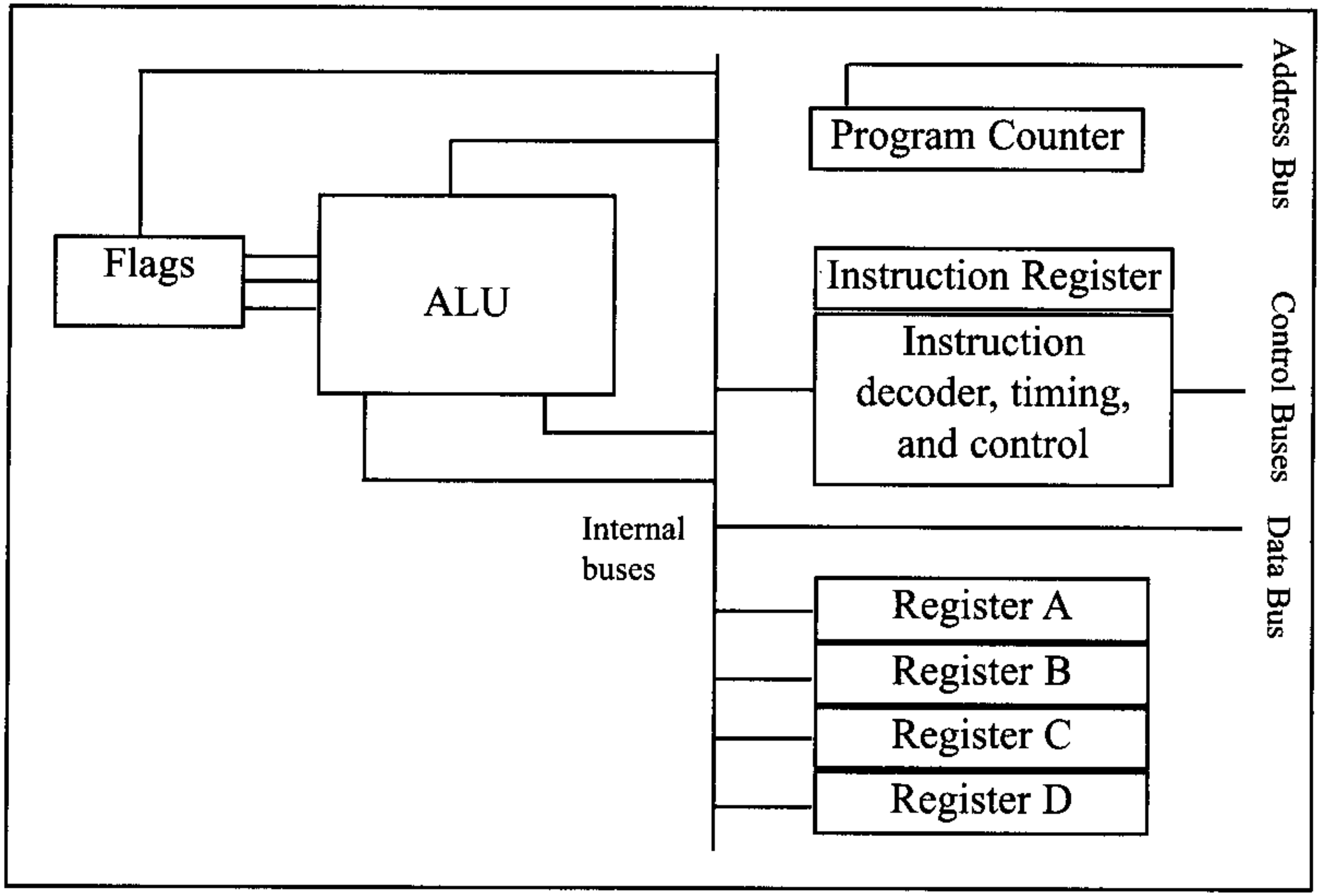


Figure 0-19. Internal Block Diagram of a CPU

AVR Microcontroller

0.4 CPU ARCHITECTURE

Internal working of CPUs

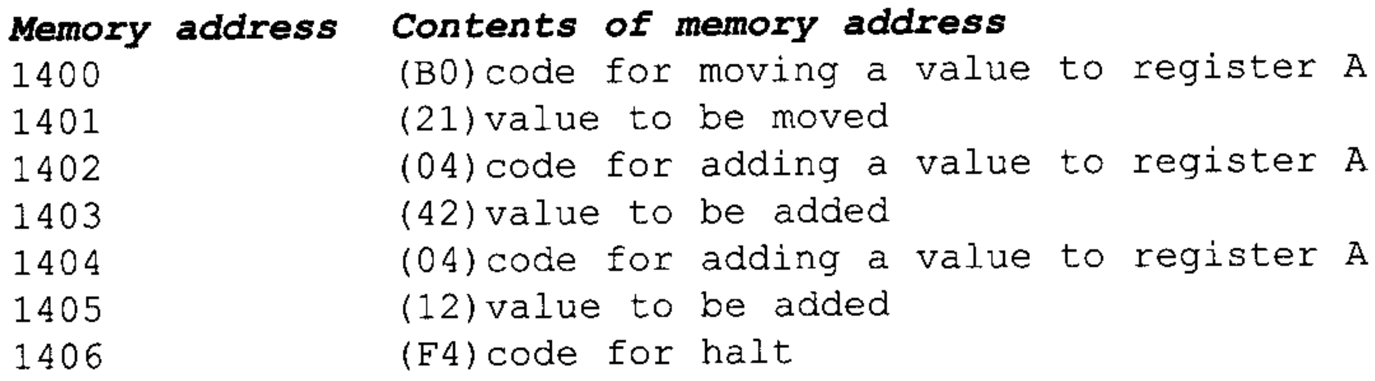
To demonstrate some of the concepts discussed above, a step-by-step analysis of the process a CPU would go through to add three numbers is given next. Assume that an imaginary CPU has registers called A, B, C, and D. It has an 8-bit data bus and a 16-bit address bus. Therefore, the CPU can access memory from addresses 0000 to FFFFH (for a total of 10000H locations). The action to be performed by the CPU is to put hexadecimal value 21 into register A, and then add to register A the values 42H and 12H. Assume that the code for the CPU to move a value to register A is 1011 0000 (BOH) and the code for adding a value to register A is 0000 0100 (04H). The necessary steps and code to perform these operation are as follows:

Action	**Code**	**Data**
Move value 21H into register A	B0H	21H
Add value 42H to register A	04H	42H
Add value 12H to register A	04H	12H

AVR Microcontroller

0.4 CPU ARCHITECTURE

If the program to perform the actions listed above is stored in memory locations starting at 1400H, the following would represent the contents for each memory address location:



Memory address	***Contents of memory address***
1400	(B0)code for moving a value to register A
1401	(21)value to be moved
1402	(04)code for adding a value to register A
1403	(42)value to be added
1404	(04)code for adding a value to register A
1405	(12)value to be added
1406	(F4)code for halt

The actions performed by the CPU to run the program above would be as follows:

1. The CPU's program counter can have a value between 0000 and FFFFH. The program counter must be set to the value 1400H, indicating the address of the first instruction code to be executed. After the program counter has been loaded with the address of the first instruction, the CPU is ready to execute.

AVR Microcontroller

0.4 CPU ARCHITECTURE

2. The CPU puts 1400H on the address bus and sends it out. The memory circuitry finds the location while the CPU activates the READ signal, indicating to memory that it wants the byte at location 1400H. This causes the contents of memory location 1400H, which is B0, to be put on the data bus and brought into the CPU.

3. The CPU decodes the instruction 130 with the help or its instruction decoder dictionary. When it finds the definition for that instruction it knows it must bring the byte in the next memory location into register A of the CPU. Therefore, it commands its controller circuitry to do exactly that. When it brings in value 21H from memory location 1401, it makes sure that the doors of all registers are closed except register A. Therefore, when value 21H comes into the CPU it will go directly into register A. After completing one instruction, the program counter points to the address of the next instruction to be executed, which in this case is 1402H. Address 1402 is sent out on the address bus to fetch the next instruction.

AVR Microcontroller

0.4 CPU ARCHITECTURE

4. From memory location 1402H the CPU fetches code 04H. After decoding, the CPU knows that it must add the byte sitting at the next address (1403) to the contents of register A. After the CPU brings the value (in this case, 42H) into register A, it provides the contents of register A along with this value to the ALU to perform the addition. It then takes the result of the addition from the ALU's output and puts it into register A. Meanwhile the program counter becomes 1404, the address of the next instruction.

Now suppose that address 1403H contained value 04 instead of 42H. How would the CPU distinguish between data 04 to be added and code 04? Remember that code 04 for this CPU means "move the next value into register A." Therefore, the CPU will not try to decode the next value. It simply moves the contents of the following memory location into register A, regardless of its value.

AVR Microcontroller

0.4 CPU ARCHITECTURE

Harvard and von Neumann architectures

Every microprocessor must have memory space to store program (code) and data. While code provides instructions to the CPU, the data provides the information to be processed. The CPU uses buses (wire traces) to access the code ROM and data RAM memory spaces. The early computers used the same bus for accessing both the code and data. Such an architecture is commonly referred to as von Neumann (Princeton) architecture.

That means for von Neumann computers, the process of accessing the code or data could cause them to get in each other's way and slow down the processing speed of the CPU, because each had to wait for the other to finish fetching.

To speed up the process of program execution, some CPUs use what is called Harvard architecture. In Harvard architecture, we have separate buses for the code and data memory.

AVR Microcontroller

0.4 CPU ARCHITECTURE

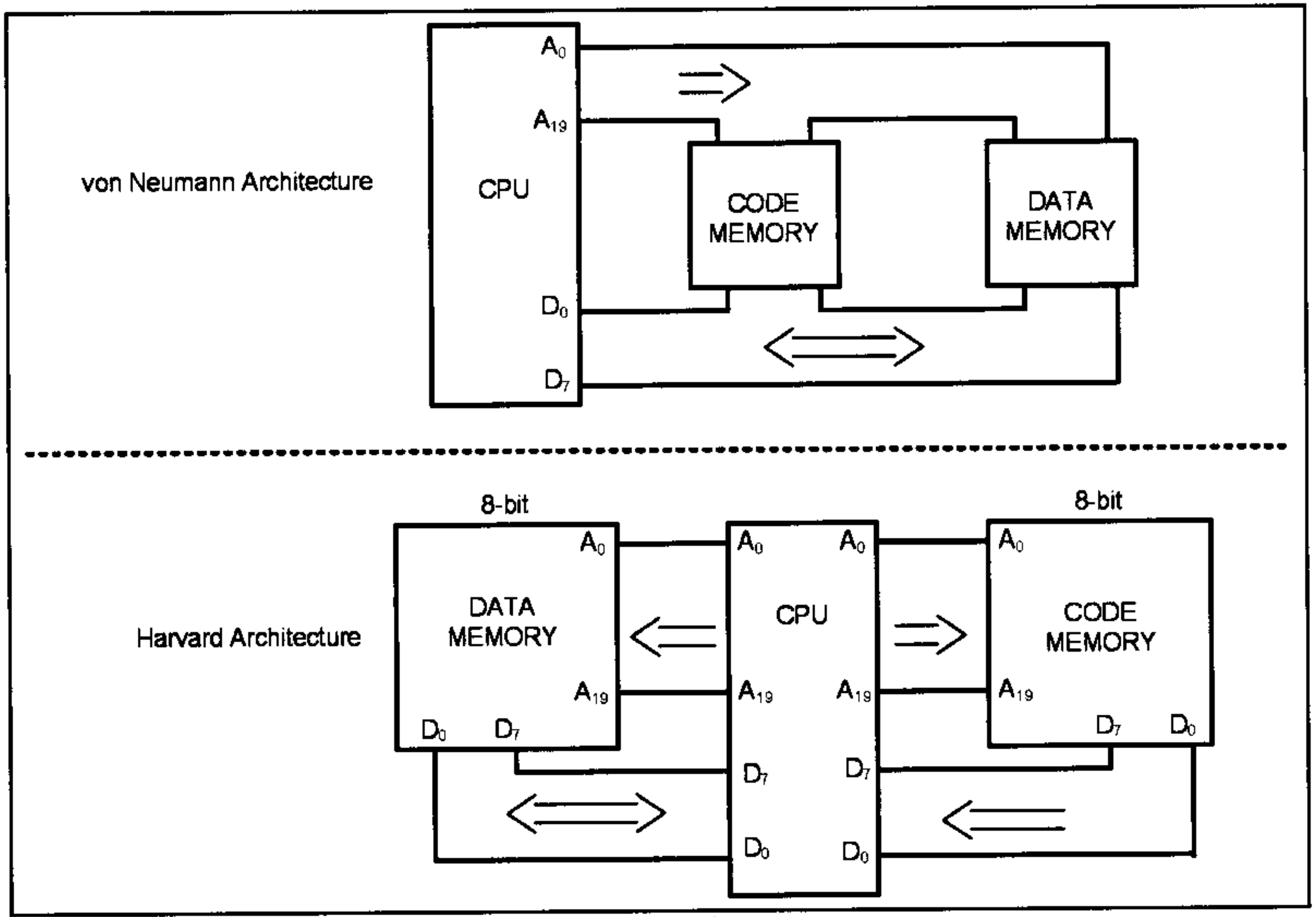


Figure 0-20. von Neumann vs. Harvard Architecture

AVR Microcontroller

0.4 CPU ARCHITECTURE

That means that we need four sets of buses:

(1) a set of data buses for carrying data into and out of the CPU,

(2) a set of address buses for accessing the data,

(3) a set of data buses for carrying code into the CPU, and

(4) an address bus for accessing the code.

This is easy to implement inside an IC chip such as a microcontroller where both ROM code and data RAM are internal (on-chip) and distances are on the micron and millimeter scale. But implementing Harvard architecture for systems such as x86 IBM PC-type computers is very expensive because the RAM and ROM that hold code and data are external to the CPU. Separate wire traces for data and code on the motherboard will make the board large and expensive.

AVR Microcontroller

0.4 CPU ARCHITECTURE

For example, for a Pentium microprocessor with a 64-bit data bus and a 32-bit address bus we will need about 100 wire traces on the motherboard if it is von Neumann architecture (96 for address and data, plus a few others for control signals of read and write and so on).

But the number of wire traces will double to 200 if we use Harvard architecture. Harvard architecture will also necessitate a large number of pins coming out of the microprocessor itself. For this reason you do not see Harvard architecture implemented in the world of PCs and workstations. This is also the reason that microcontrollers such as AVR use Harvard architecture internally, but they still use von Neumann architecture if they need external memory for code and data space. The von Neumann architecture was developed at Princeton University, while the Harvard architecture was the work of Harvard University.