12.1 Early Memory

Every year new memory technologies are developed promising faster response and higher throughput. This makes it difficult to maintain a printed document discussing the latest advances in memory technologies. Although this chapter does present some basic memory technologies and how they are used to improve performance, the focus is on memory organization and interfacing with the processors.

One of the earliest types of computer memory was called ***magnetic core memory***. It was made by weaving fine copper wires through tiny rings of magnetic material in an array. Figure 12-1 shows the basic arrangement of core memory.

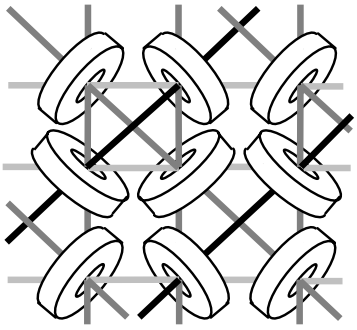


Figure 12-1 Diagram of a Section of Core Memory

Much like recording music to a magnetic tape, when electrical current was sent through the center of one of the magnetic rings, it polarized it with a magnetic charge. Each of these rings could have a charge that flowed clockwise or counter-clockwise. One direction was considered a binary 1 while the other was considered a binary 0.

The horizontal and vertical wires of the core memory were used to write data to a specific ring. By putting half the current necessary to polarize the magnetic material on one of the horizontal wires and the same level of current on one of the vertical wires, the ring where the two wires intersected had enough total current to modify the ring's polarity. The polarity of the remaining rings would be left unaltered.

The diagonal wires, called sense wires, were used to read data. They could detect when the polarity on one of the rings was changed. To read data, therefore, the bit in question would be written to with the horizontal and vertical wires. If the sense wire detected a change in polarity, the bit that had been stored there must have been opposite from the one just written. If no polarity change was detected, the bit written must have been equal to the one stored in that ring.

Magnetic core memory looks almost like fabric, the visible rings nestled among a lacework of glistening copper wires. It is for these reasons, however, that it is also impractical. Since the rings are enormous relative to the scale of electronics, a memory of 1024 bytes (referred to as a 1K x 8 or "1K by 8") had physical dimensions of approximately 8 inches by 8 inches. In addition, the fine copper wires were very fragile making manufacturing a difficult process. A typical 1K x 8 memory would cost thousands of dollars. Therefore, magnetic core memory disappeared from use with the advent of transistors and memory circuits such as the latch presented in Chapter 10.

12.2 Organization of Memory Device

Modern memory has the same basic configuration as magnetic core memory although the rings have been replaced with electronic memory cells such as the D-Latch. The cells are arranged so that each row represents a memory location where a binary value would be stored and the columns represent different bits of those memory locations. This is where the terminology "1K x 8" used in Section 12.1 comes from. Memory is like a matrix where the number of rows identifies the number of memory locations in the memory and the number of columns identifies the number of bits in each memory location.

To store to or retrieve data from a memory device, the processor must place a binary number called an address on special inputs to the memory device. This address identifies which row of the memory matrix or array the processor is interested in communicating with, and enables it.

Once a valid address is placed on the address lines, the memory cells from that row are connected to bi-directional connections on the memory device that allow ***data*** either to be stored to or read from the latches. These connections are called the data lines. Three additional lines, ***chip select***, ***read enable***, and ***write enable***, are used to control the transaction.

Figure 12-2 presents the basic organization of a memory device.

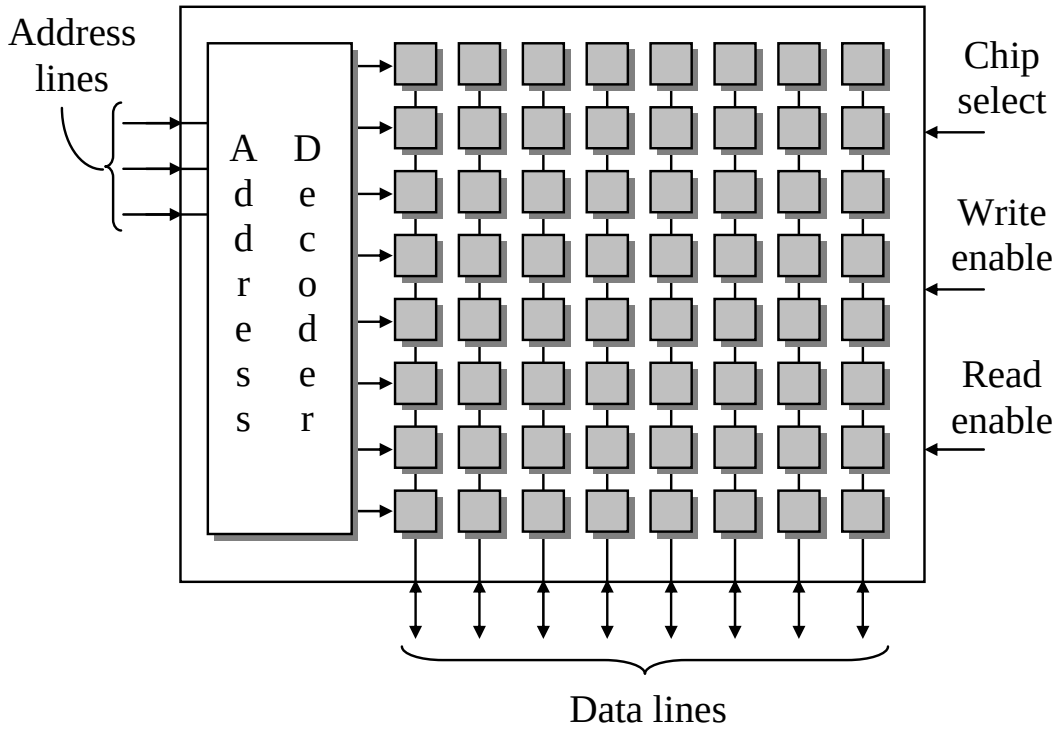


Figure 12-2 Basic Organization of a Memory Device

Remember from Chapter 8 that a decoder with *n* inputs has 2^n outputs, exactly one of which will be active for each unique pattern of ones and zeros at its input. For example, an active-low 2-input decoder will have four outputs. A different output will equal zero for each unique input pattern while all of the other inputs will be ones.

An ***address decoder*** selects exactly one row of the memory array to be active leaving the others inactive. When the microprocessor places a binary number onto the address lines, the address decoder selects a single row in the memory array to be written to or read from. For example, if $011_2 = 3_{10}$ is placed on the address lines, the fourth row of the memory will be connected to the data lines. The first row is row 0.

The processor uses the inputs ***read enable*** and ***write enable*** to specify whether it is reading data from or writing data to the selected row of the memory array. These signals are active low. When read enable is zero, we are reading data from memory, and when write enable is zero, we are writing data to memory. These two signals should never be zero at the same time.

Sometimes, the read enable and write enable signals are combined into a single line called $R/\overline{W}$ (pronounced "read write-bar"). In this case, a one on $R/\overline{W}$ initiates a data read while a zero initiates a write.

If latches are used for the memory cells, then the data lines are connected to the *D* inputs of the memory location latches when data is written, and they are connected to the *Q* outputs when data is read.

The last input to the memory device shown in Figure 12-2 is the ***chip select***. The chip select is an active low signal that enables and disables the memory device. If the chip select equals zero, the memory activates all of its input and output lines and uses them to transfer data. If the chip select equals one, the memory becomes idle, effectively disconnecting itself from all of its input and output lines. The reason for this is that the typical memory device shares the address and data lines of a processor with other devices.

Rarely does a processor communicate with only one memory device on its data lines. Problems occur when more than one device tries to communicate with the processor over shared lines at the same time. It would be like ten people in a room trying to talk at once; no one would be able to understand what was being said.

The processor uses digital logic to control these devices so that only one is talking or listening at a time. Through individual control of each of the chip select lines to the memory devices, the processor can enable only the memory device it wishes to communicate with. The processor places a zero on the chip select of the memory device it wants to communicate with and places ones on all of the other chip select inputs.

The next section discusses how these chip selects are designed so that no conflicts occur.

12.3 Interfacing Memory to a Processor

The previous section presented the input and output lines for a memory device. These lines are shared across all of the devices that communicate with the processor. If you look at the electrical traces across the surface of a motherboard, you should see collections of traces running together in parallel from the processor to then from one memory device to the next. These groups of wires are referred to as the ***bus***, which is an extension of the internal structure of the processor. This section discusses how the memory devices share the bus.

12.3.1 Buses

In order to communicate with memory, a processor needs three types of connections: data, address, and control. The ***data lines*** are the electrical connections used to send data to or receive data from

memory. There is an individual connection or wire for each bit of data. For example, if the memory of a particular system has 8 latches per memory location, i.e., 8 columns in the memory array, then it can store 8-bit data and has 8 individual wires with which to transfer data.

The ***address lines*** are controlled by the processor and are used to specify which memory location the processor wishes to communicate with. The address is an unsigned binary integer that identifies a unique location where data elements are to be stored or retrieved. Since this unique location could be in any one of the memory devices, the address lines are also used to specify which memory device is enabled.

The ***control lines*** consist of the signals that manage the transfer of data. At a minimum, they specify the timing and direction of the data transfer. The processor also controls this group of lines. Figure 12-3 presents the simplest connection of a single memory device to a processor with *n* data lines and *m* address lines.

Unfortunately, the configuration of Figure 12-3 only works with systems that have a single memory device. This is not very common. For example, a processor may interface with a BIOS stored in a non-volatile memory while its programs and data are stored in the volatile memory of a RAM stick. In addition, it may use the bus to communicate with devices such as the hard drive or video card. All of these devices share the data, address, and control lines of the bus. (BIOS stands for Basic Input/Output System and it is the low-level code used to start the processor when it is first powered up.)

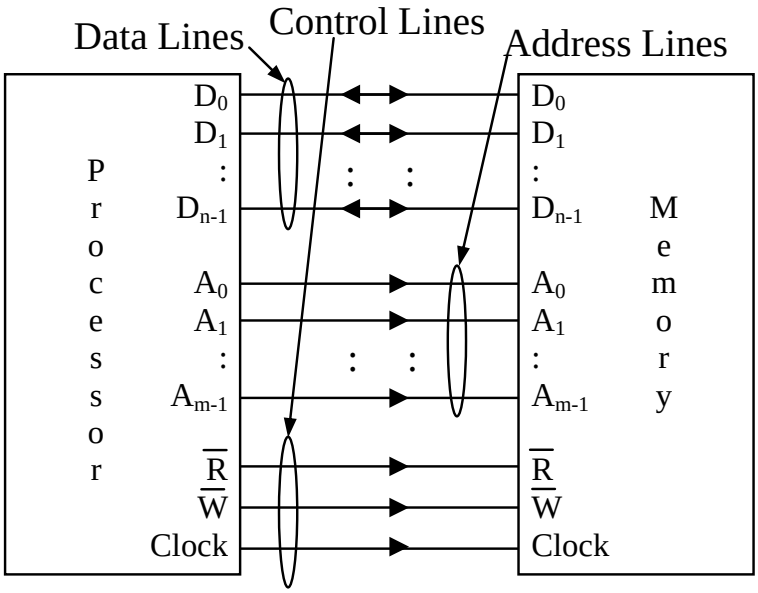


Figure 12-3 Basic Processor to Memory Device Interface

A method had to be developed to allow a processor to communicate to multiple memory devices across the same set of wires. If this wasn't done, the processor would need a separate set of data, address, and control lines for each device placing an enormous burden on circuit board designers for routing wires.

By using a bus, the processor can communicate with exactly one device at a time even though it is physically connected to many devices. If only one device on the bus is enabled at a time, the processor can perform a successful data transfer. If two devices tried to drive the data lines simultaneously, the result would be lost data in a condition called ***bus contention***.

Figure 12-4 presents a situation where data is being read from memory device 1 while memory device 2 remains "disconnected" from the bus. Disconnected is in quotes because the physical connection is still present; it just doesn't have an electrical connection across which data can pass.

Notice that Figure 12-4 shows that the only lines disconnected from the bus are the data lines. This is because bus contention only occurs when multiple devices are trying to output to the same lines at the same time. Since only the microprocessor outputs to the address and control lines, they can remain connected.

In order for this scheme to work, an additional control signal must be sent to each of the memory devices telling them when to be connected to the bus and when to be disconnected. This control signal is called a ***chip select***.

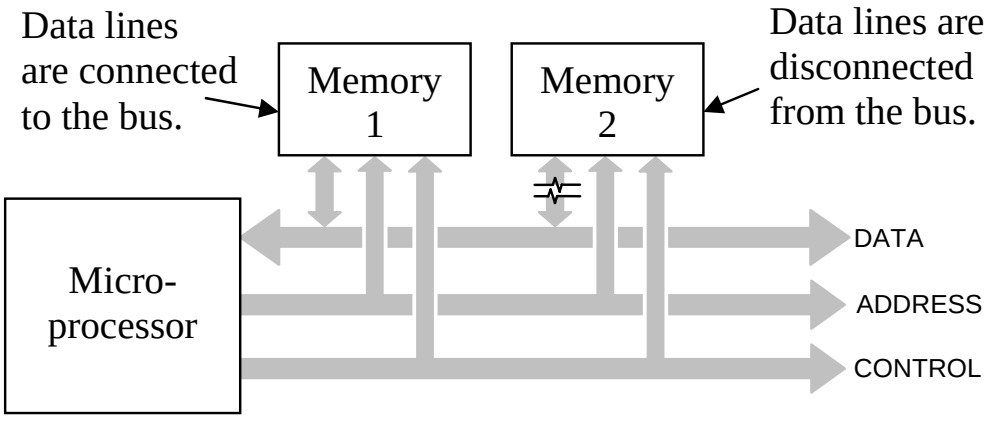


Figure 12-4 Two Memory Devices Sharing a Bus

A chip select is an active low signal connected to the enable input of the memory device. If the chip select is high, the memory device remains idle and its data lines are disconnected from the bus. When the processor wants to communicate with the memory device, it pulls that device's chip select low thereby enabling it and connecting it to the bus.

Each memory device has its own chip select, and at no time do two chip selects go low at the same time. For example, Table 12-1 shows the only possible values of the chip selects for a system with four memory devices.

Table 12-1 The Allowable Settings of Four Chip Selects

	CS_0	CS_1	CS_2	CS_3
Only memory device 0 connected	0	1	1	1
Only memory device 1 connected	1	0	1	1
Only memory device 2 connected	1	1	0	1
Only memory device 3 connected	1	1	1	0
All devices disconnected	1	1	1	1

The disconnection of the data lines is performed using ***tristate outputs*** for the data lines of the memory chips. A tristate output is digital output with a third state added to it. This output can be a logic 1, a logic 0, or a third state that acts as a high impedance or open circuit. It is like someone opened a switch and nothing is connected.

This third state is controlled by the chip select. When the active low chip select equals 1, data lines are set to high impedance, sometimes called the ***Z state***. A chip select equal to 0 causes the data lines to be active and allow input or output.

In Figure 12-5a, three different outputs are trying to drive the same wire. This results in bus contention, and the resulting data is unreadable. Figure 12-5b shows two of the outputs breaking their connection with the wire allowing the first output to have control of the line. This is the goal when multiple devices are driving a single line. Figure 12-5c is the same as 12-5b except that the switches have been replaced with tristate outputs. With all but one of the outputs in a Z state, the top gate is free to drive the output without bus contention.

The following sections describe how memory systems are designed using chip selects to take advantage of tristate outputs.

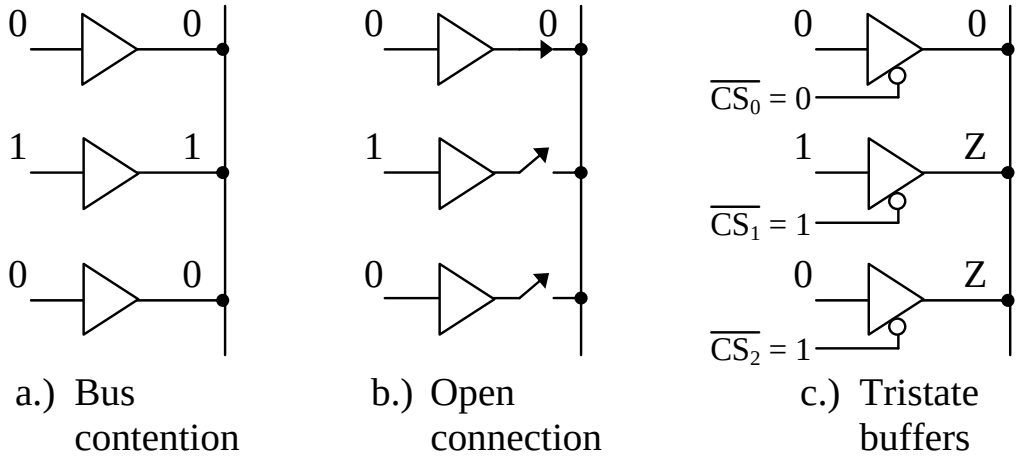


Figure 12-5 Three Buffers Trying to Drive the Same Output

12.3.2 Memory Maps

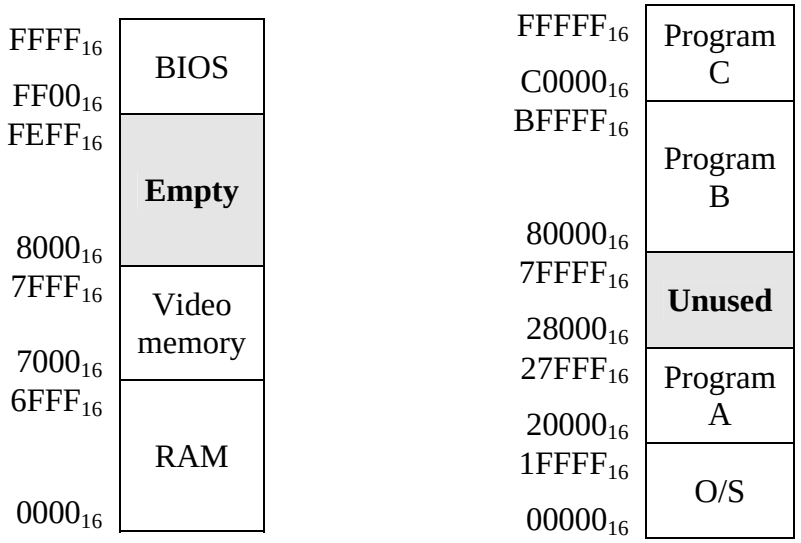
Think of memory as several filing cabinets where each folder can contain a single piece of data. The size of the stored data, i.e., the number of bits that can be stored in a single memory location, is fixed and is equal to the number of columns in the memory array. Each piece of data can be either code (part of a program) or data (variables or constants used in the program). Code and data are typically stored in the same memory, each piece of which is stored in a unique address or row of memory.

Some sections of memory are assigned to a predefined purpose which may place constraints on how they are arranged. For example, the BIOS from which the computer performs its initial startup sequence is located at a specific address range in non-volatile memory. Video memory may also be located at a specific address range.

System designers must have a method to describe the arrangement of memory in a system. Since multiple memory devices and different types of memory may be present in a single system, hardware designers need to be able to show what addresses correspond to which memory devices. Software designers also need to have a way to show how the memory is being used. For example, which parts of memory will be used for the operating system, which parts will be used to store a program, or which parts will be used to store the data for a program.

System designers describe the use of memory with a ***memory map***. A memory map represents a system's memory with a long, vertical column. It is meant to model the memory array where the rows correspond to the memory locations. Within the full range of addresses

are smaller partitions where the individual resources are present. Figure 12-6 presents two examples of memory maps.



a.) Hardware-specific b.) Software-specific

Figure 12-6 Sample Memory Maps

The numbers along the left side of the memory map represent the addresses corresponding to each memory resource. The memory map should represent the full address range of the processor. This full address range is referred to as the processor's ***memory space***, and its size is represented by the number of memory locations in the full range, i.e., 2^m where *m* equals the number of address lines coming out of the processor. It is up to the designer whether the addresses go in ascending or descending order on the memory map.

As an example, let's calculate the memory space of the processor represented by the memory map in Figure 12-6b. The top address for this memory map is $FFFFF_{16}$ = 1111 1111 1111 1111 1111_2. Since the processor accesses its highest address by setting all of its address lines to 1, we know that this particular processor has 20 address lines. Therefore, its memory space is $2^{20} = 1{,}048{,}576_{10}$ = 1 Meg. This means that all of the memory resources for this processor must be able to fit into 1 Meg without overlapping.

In the next section, we will see how to compute the size of each partition of memory using the address lines. For now, however, we can determine the size of a partition in memory by subtracting the low

address from the high address, then adding one to account for the fact that the low address itself is a memory location. For example, the range of the BIOS in Figure 12-6a starts at $FF00_{16} = 65{,}280_{10}$ and goes up to $FFFF_{16} = 65{,}535_{10}$. This means that the BIOS fits into 65,535 – 65,280 +1 = 256 memory locations.

It is vital to note that there is an exact method to selecting the upper and lower addresses for each of the ranges in the memory map. Take for example the memory range for Program A in Figure 12-6b. The lower address is 20000_{16} while the upper address is $27FFF_{16}$. If we convert these addresses to binary, we should see a relationship.

$$20000_{16} = 0010\ 0000\ 0000\ 0000\ 0000_2$$
$$27FFF_{16} = 0010\ 0111\ 1111\ 1111\ 1111_2$$

It is not a coincidence that the upper five bits of these two addresses are identical while the remaining bits go from all zeros in the low address to all ones in the high address. Converting the high and the low address of any one of the address ranges in Figure 12-6 should reveal the same characteristic.

The next section shows how these most significant address bits are used to define which memory device is being selected.

12.3.3 Address Decoding

Address decoding is a method for using an address to enable a unique memory device while leaving all other devices idle. The method described here works for many more applications than memory though. It is the same method that is used to identify which subnet a host computer is connected to based on its IP address.

All address decoding schemes have one thing in common: the bits of the full address are divided into two groups, one group that is used to identify the memory device and one group that identifies the memory location within the selected memory device. In order to determine how to divide the full address into these two groups of bits, we need to know how large the memory device is and how large the memory space is. Once we know the size of the memory device, then we know the number of bits that will be required from the full address to point to a memory location within the memory device.

Just as we calculated the size of the memory space of a processor, the size of the memory space of a device is calculated by raising 2 to a power equal to the number of address lines going to that device. For

example, a memory device with 28 address lines going into it has 2^{28} = 256 Meg locations. This means that 28 address bits from the full address must be used to identify a memory location within that device. All of the remaining bits of the full address will be used to enable or disable the device. It is through these remaining address bits that we determine where the memory will be located within the memory map.

Table 12-2 presents a short list of memory sizes and the number of address lines required to access all of the locations within them. Remember that the memory size is simply equal to 2^m where *m* is the number of address lines going into the device.

Table 12-2 Sample Memory Sizes versus Required Address Lines

Memory size	Number of address lines
1 K	10
256 K	18
1 Meg	20
16 Meg	24

Memory size	Number of address lines
256 Meg	28
1 Gig	30
4 Gig	32
64 Gig	36

The division of the full address into two groups is done by dividing the full address into a group of most significant bits and least significant bits. The block diagram of an *m*-bit full address in Figure 12-7 shows how this is done. Each bit of the full address is represented with a_n where *n* is the bit position.

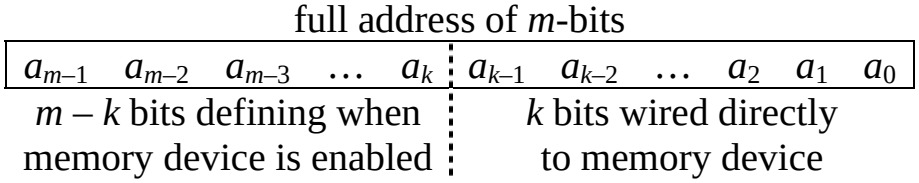


Figure 12-7 Full Address with Enable Bits and Device Address Bits

The bits used to enable the memory device are always the most significant bits while the bits used to access a memory location within the device are always the least significant bits.

Example

A processor with a 256 Meg address space is using the address $35E3C03_{16}$ to access a 16 Meg memory device.

- How many address lines are used to define when the 16 Meg memory space is enabled?
- What is the bit pattern of these enable bits that enables this particular 16 Meg memory device?
- What is the address within the 16 Meg memory device that this address is going to transfer data to or from?
- What is the lowest address in the memory map of the 16 Meg memory device?
- What is the highest address in the memory map of the 16 Meg memory device?

Solution

First, we need to determine where the division in the full address is so that we know which bits go to the enable circuitry and which are connected directly to the memory device's address lines. From Table 12-2, we see that to access 256 Meg, we need 28 address lines. Therefore, the processor must have 28 address lines coming out of it.

The memory device is only 16 Meg which means that it requires 24 address lines to uniquely identify all of its addresses.

a_{27} a_{26} a_{25} a_{24}	a_{23} a_{22} … a_2 a_1 a_0
4 bits that enable memory device	24 bits going to address lines of memory device

Therefore, the ***four most significant address lines*** are used to enable the memory device.

By converting $35E3C03_{16}$ to binary, we should see the values of each of these bit positions for this memory location in this memory device.

$$35E3C03_{16} = 0011\ 0101\ 1110\ 0011\ 1100\ 0000\ 0011_2$$

The four most significant bits of this 28-bit address are $\mathbf{0011_2}$. This, therefore, is the bit pattern that will enable this particular 16 Meg memory device: $a_{27} = 0$, $a_{26} = 0$, $a_{25} = 1$, and $a_{24} = 1$. Any other pattern

of bits for these four lines will disable this memory device and disallow any data transactions between it and the processor.

The 16 Meg memory device never sees the most significant four bits of this full address. The only address lines it ever sees are the 24 that are connected directly to its address lines: a_0 through a_{23}. Therefore, the address the memory device sees is:

$$0101\ 1110\ 0011\ 1100\ 0000\ 0011_2 = 5E3C03_{16}$$

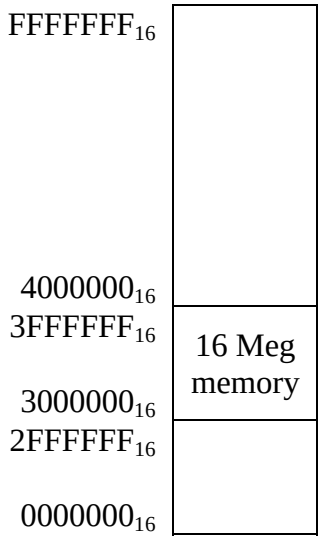
As for the highest and lowest values of the full address for this memory device, we need to examine what the memory device interprets as its highest and lowest addresses. The lowest address occurs when all of the address lines ***to the memory device*** are set to 0. The highest address occurs when all of the address lines ***to the memory device*** are set to 1. Note that this does not include the four most significant bits of the full address which should stay the same in order for the memory device to be active. Therefore, from the standpoint of the memory map which uses the full address, the lowest address is the four enable bits set to 0011_2 followed by 24 zeros. The highest address is the four enable bits set to 0011_2 followed by 24 ones.

	4 bits that enable memory device				24 bits going to address lines of memory device					
	a_{27}	a_{26}	a_{25}	a_{24}	a_{23}	a_{22}	…	a_2	a_1	a_0
Highest address	0	0	1	1	1	1	…	1	1	1
Lowest address	0	0	1	1	0	0	…	0	0	0

Therefore, from the perspective of the memory map, the lowest and highest addresses of this memory device are:

$$\text{Highest} = 0011\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_2 = 3FFFFFF_{16}$$
$$\text{Lowest} = 0011\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_2 = 3000000_{16}$$

The following memory map shows how this 16 Meg memory is placed within the full range of the processor's memory space. The full address range of the processor's memory space is determined by the fact that there are 28 address lines from the processor. Twenty-eight ones is $FFFFFFF_{16}$ in hexadecimal and 28 zeros is 0000000_{16}.



The method for resolving the subnet of an IP address is the same as enabling a specific memory device within a processor's memory space. When configuring a computer to run on a network that uses the Internet Protocol version 4 addressing scheme, it must be assigned a 32-bit address that uniquely identifies it among all of the other computers on that network. This 32-bit address serves a second purpose though: it identifies the sub-network or subnet that this computer is a member of within the entire network. A subnet within the entire IP network is equivalent to a memory device within the memory space of a processor.

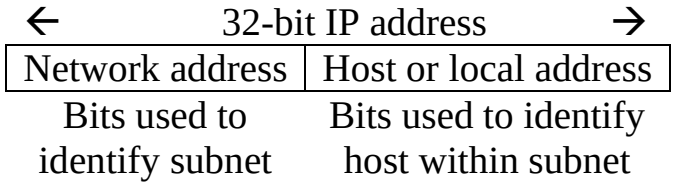


Figure 12-8 IPv4 Address Divided into Subnet and Host IDs

According to IPv4 standard, there are four classes of addressing, Class A, Class B, Class C, and Class D. Each of these classes is defined by the number of bits that are assigned to identify the subnet and how many bits are left for the host ID. For example, a Class A subnet uses 8 bits to identify the subnet. This leaves 24 bits to identify the host within the subnet. Therefore, a Class A network can ideally contain a maximum of 2^{24} = 16,777,216 hosts. The actual number of hosts is two less. Two addresses for every subnet are reserved: one for a broadcast address and one for the subnet itself.

A Class C network uses 24 bits to identify the subnet and 8 bits to identify the host within the subnet. Therefore, a Class C network can have at most $2^8 - 2 = 254$ machines on it, far fewer than a Class A. The drawback of a Class A network, however, is that if the entire network were assigned to Class A subnets, then there would ideally only be room for $2^8 = 256$ subnets. Whenever the number of bits used to identify the subnet is increased, the number of possible subnets is increased while the number of hosts within a subnet is decreased.

Example

The IPv4 address 202.54.151.45 belongs to a Class C network. What are the subnet and the host ids of this address?

Solution

First, IPv4 addresses are represented as four bytes represented in decimal notation. Therefore, let's convert the IP address above into its 32-bit binary equivalent.

$$202_{10} = 11001010_2$$
$$54_{10} = 00110110_2$$
$$151_{10} = 10010111_2$$
$$45_{10} = 00101101_2$$

This means that the binary address of 202.54.151.45 is:

11001010.00110110.10010111.00101101

Remember that the Class C network uses the first twenty-four bits for the subnet id. This gives us the following value for the subnet id.

$$\text{Subnet id}_{202.54.151.45} = 110010100011011010010111_2$$

Any IPv4 address with the first 24 bits equal to this identifies a host in this subnet.

The host id is taken from the remaining bits.

$$\text{Host id}_{202.54.151.45} = 00101101_2$$

12.3.4 Chip Select Hardware

What we need is a circuit that will enable a memory device whenever the full address is within the address range of the device and

disable the memory device when the full address falls outside the address range of the device. This is where those most significant bits of the full address come into play.

Remember from our example where we examined the addressing of a 16 Meg memory device in the 256 Meg memory space of a processor that the four most significant bits needed to remain 0011_2. In other words, if the four bits a_{27}, a_{26}, a_{25}, and a_{24} equaled 0000_2, 0001_2, 0010_2, 0100_2, 0101_2, 0110_2, 0111_2, 1000_2, 1001_2, 1010_2, 1011_2, 1100_2, 1101_2, 1110_2, or 1111_2, the 16 Meg memory device would be disabled. Therefore, we want a circuit that is active when $a_{27} = 0$, $a_{26} = 0$, $a_{25} = 1$, and $a_{24} = 1$. This sounds like the product from an AND gate with a_{27} and a_{26} inverted. Chip select circuits are typically active low, however, so we need to invert the output. This gives us a NAND gate.

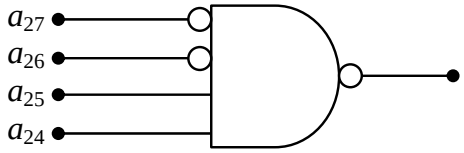


Figure 12-9 Sample Chip Select Circuit for a Memory Device

So the process of designing a chip select is as follows:

- Using the memory space of the processor and the size of the memory device, determine the number of bits of the full address that will be used for the chip select.
- Using the base address where the memory device is to be located, determine the values that the address lines used for the chip select are to have.
- Create a circuit with the address lines for the chip select going into the inputs of a NAND gate with the bits that are to be zero inverted.

Example

Using logic gates, design an active low chip select for a 1 Meg BIOS to be placed in the 1 Gig memory space of a processor. The BIOS needs to have a starting address of $1E00000_{16}$.

Solution

First of all, let's determine how many bits are required by the 1 Meg BIOS. We see from Table 12-2 that a 1 Meg memory device requires

20 bits for addressing. This means that the lower 20 address lines coming from the processor must be connected to the BIOS address lines. Since a 1 Gig memory space has 30 address lines (2^{30} = 1 Gig), then 30 – 20 = 10 address lines are left to determine the chip select.

Next, we figure out what the values of those ten lines are supposed to be. If we convert the starting address to binary, we get:

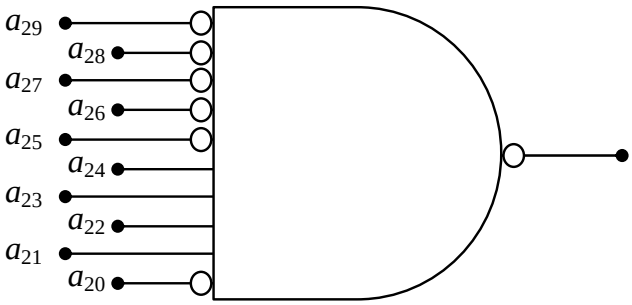
$$1E00000_{16} = 00\ 0001\ 1110\ 0000\ 0000\ 0000\ 0000\ 0000$$

Notice that enough leading zeros were added to make the address 30 bits long, the appropriate length in a 1 Gig memory space.

We need to assign each bit a label. We do this by labeling the least significant bit a_0, then incrementing the subscript for each subsequent position to the left. This gives us the following values for each address bit. (a_{18} through a_2 have been deleted in the interest of space.)

a_{29}	a_{28}	a_{27}	a_{26}	a_{25}	a_{24}	a_{23}	a_{22}	a_{21}	a_{20}	a_{19}	a_{18}	…	a_1	a_0
0	0	0	0	0	1	1	1	1	0	0	0	…	0	0

Bits a_{20} through a_{29} are used for the chip select.



Example

What is the largest memory device that can be placed in a memory map with a starting address of $A40000_{16}$?

Solution

This may seem like a rather odd question, but it actually deals with an important aspect of creating chip selects. Notice that for every one of our starting addresses, the bits that go to the chip select circuitry can be ones or zeros. The bits that go to the address lines of the memory device, however, ***must all be zero.*** This is because the first address in any memory device is 0_{10}. The ending or highest address will have all ones going to the address lines of the memory device.

Let's begin by converting the address $A40000_{16}$ to binary.

$$A40000_{16} = 1010\ 0100\ 0000\ 0000\ 0000\ 0000_2$$

If we count the zeros starting with the least significant bit and moving left, we see that there are 18 zeros before we get to our first one. This means that the largest memory device we can place at this starting address has 18 address lines. Therefore, the largest memory device we can start at this address has 2^{18} = 256 K memory locations.

Example

True or False: $B000_{16}$ to $CFFF_{16}$ is a valid range for a single memory device.

Solution

This is much like the previous example in that it requires an understanding of how the address lines going to the chip select circuitry and the memory device are required to behave. The previous example showed that the address lines going to the memory device must be all zero for the starting or low address and all ones for the ending or high address. The address lines going to the chip select, however, must all remain constant.

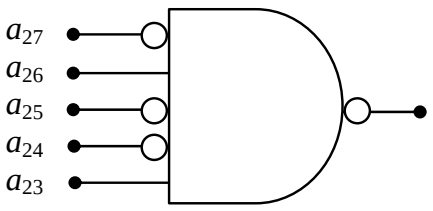
Let's begin by converting the low and the high addresses to binary.

	a_{15}	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0
Low	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0
High	1	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1

Note that it is impossible to make a vertical division through both the high and the low addresses where all of the bits to the left are the same for both the high and the low addresses while every bit to the right goes from all zeros for the low address to all ones for the high address. Since we cannot do this, we cannot make a chip select for this memory device and the answer is ***false***.

Example

What is the address range of the memory device that is enabled with the chip select shown?



Solution

To begin with, the addressing can be determined from the subscripts of the address lines identified in the figure. The address lines coming out of the processor go from a_0 (always assumed to be the least significant bit of the address) to a_{27}. This means that the processor has 28 address lines and can access a memory space of 2^{28} = 256 Meg.

The chip select only goes low when all of the inputs to the NAND gate (after the inverters) equal 1. This means that $a_{27} = 0$, $a_{26} = 1$, $a_{25} = 0$, $a_{24} = 0$, and $a_{23} = 1$. We find the lowest address by setting all of the remaining bits, a_{22} through a_0, to zero and we find the highest address by setting all of the remaining bits to 1. This gives us the following binary addresses.

	a_{27}	a_{26}	a_{25}	a_{24}	a_{23}	a_{22}	a_{21}	…	a_1	a_0
High address	0	1	0	0	1	1	1	…	1	1
Low address	0	1	0	0	1	0	0	…	0	0

When we convert these values to hexadecimal, we get:

High address = $0100\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_2 = 4FFFFFF_{16}$
Low address = $0100\ 1000\ 0000\ 0000\ 0000\ 0000\ 0000_2 = 4800000_{16}$

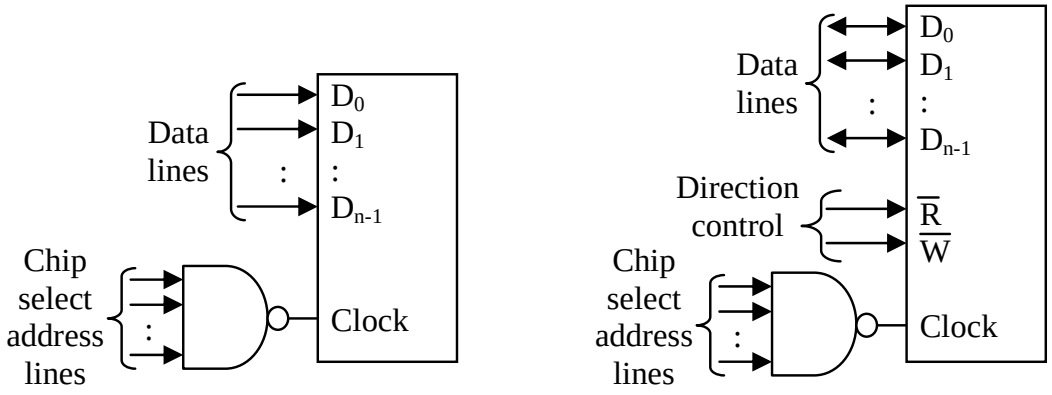
12.4 Memory Mapped Input/Output

Some devices do not contain a memory array, yet their interface to the processor uses data lines and control lines just like a memory device. For example, an analog-to-digital converter (ADC) reads an analog value and converts it to a digital number that the processor can use. The processor reads this digital value from ADC exactly the same way that it would read a value it had stored in a memory device.

The ADC may also require parameters to be sent to it from the processor. These parameters might include the method it uses for conversion, the time it waits between conversions, and which analog input channels are active. The processor sets these values by writing to the ADC in the same way it would store data to a memory device.

The practice of interfacing an input/output (I/O) device as if it was a memory device is called ***memory mapping***. Just like the bus interface for a memory device, the memory mapped interface to a bus uses a chip select to tell the device when it's being accessed and data lines to pass

data between the device and the processor. Some memory mapped I/O even use a limited number of address lines to identify internal registers. In addition, I/O devices use the write enable and read enable signals from the processor to determine whether data is being sent to the device or read from it. Some devices may only support writing (purely output) while others may only support reading (purely input).



a.) Memory-mapped output device

b.) Memory-mapped I/O device

Figure 12-10 Some Types of Memory Mapped I/O Configurations

12.5 Memory Terminology

There are many different purposes for memory in the operation of a computer. Some memory is meant to store data and programs only while the computer is turned on while other memory is meant to be permanent. Some memory contains application code while other memory is meant to store the low-level driver code to control devices such as an IDE interface or a video card. Some memory may have a larger capacity while other memory may be faster.

In order to understand what memory technologies to apply to which processor operation, we need to understand a little bit more about the technologies themselves. This section discusses some of the terminology used to describe memory.

12.5.1 Random Access Memory

The term ***Random Access Memory*** (RAM) is typically applied to memory that is easily read from and written to by the microprocessor. In actuality, this is a misuse of this term. For a memory to be random access means that any address can be accessed at any time. This is to differentiate it from storage devices such as tapes or hard drives where

the data is accessed sequentially. We will discuss hard drive technologies in Chapter 13.

In general, RAM is the main memory of a computer. Its purpose is to store data and applications that are currently in use. The operating system controls the use of this memory dictating when items are to be loaded into RAM, where they are to be located in RAM, and when they need to be removed from RAM. RAM is meant to be very fast both for reading and writing data. RAM also tends to be volatile in that as soon as power is removed, all of the data is lost.

12.5.2 Read Only Memory

In every computer system, there must be a portion of memory that is stable and impervious to power loss. This kind of memory is called ***Read Only Memory*** or ROM. Once again, this term is a misnomer. If it was not possible to write to this type of memory, we could not store the code or data that is to be contained in it. It simply means that without special mechanisms in place, a processor cannot write to this type of memory. If through an error of some sort, the processor tries to write to this memory, an error will be generated.

The most common application of ROM is to store the computer's BIOS. Since the BIOS is the code that tells the processor how to access its resources upon powering up, it must be present even when the computer is powered down. Another application is the code for embedded systems. For example, it is important for the code in your car's computer to remain even if the battery is disconnected.

There are some types of ROM that the microprocessor can write to, but usually the time needed to write to them or the programming requirements needed to do so make it unwise to write to them regularly. Therefore, these memories are still considered read only.

In some cases, the processor cannot write to a ROM under any circumstances. For example, the code in your car's computer should never need to be modified. This ROM is programmed before it is installed. To put a new program in the car's computer, the old ROM is removed and discarded and a new ROM is installed in its place.

12.5.3 Static RAM versus Dynamic RAM

For as long as memory has existed, scientists and engineers have experimented with new technologies to make RAM faster and to cram more of it into a smaller space, two goals that are typically at odds.

Nowhere is this more obvious than in the two main classifications of RAM: Static RAM (SRAM) and Dynamic RAM (DRAM).

SRAM is made from an array of latches such as the D-latch we studied in Chapter 10. Each latch can maintain a single bit of data within a single memory address or location. For example, if a memory stores eight bits per memory address, then there are eight latches for a single address. If this same memory has an address space of 256 K, then there are $2^{18} \cdot 8 = 2^{21} = 2{,}097{,}152$ latches in the device.

Latches are not small devices as logic circuits go, but they are very fast. Therefore, in the pursuit of the performance goals of speed and size, SRAMs are better adapted to speed. In general, SRAMs:

- store data in transistor circuits similar to D-latches;
- are used for very fast applications such as RAM caches (discussed in Chapter 13);
- tend to be used in smaller memories which allows for very fast access due to the simpler decoding logic; and
- are volatile meaning that the data remains stored only as long as power is available.

There are circuits that connect SRAMs to a back up battery that allows the data to be stable even with a loss of power. These batteries, about the size of a watch battery, can maintain the data for long periods of time much as a battery in a watch can run for years. On the negative side, the extra battery and circuitry adds to the overall system cost and takes up physical space on the motherboard

A bit is stored in a DRAM using a device called a capacitor. A capacitor is made from a pair of conductive plates that are held parallel to each other and very close together, but not touching. If an electron is placed on one of the plates, its negative charge will force an electron on the other plate to leave. This works much like the north pole of a magnet pushing away the north pole of a second magnet.

If enough electrons are deposited on the one plate creating a strong negative charge, enough electrons will be moved away from the opposite plate creating a positive charge. Like a north pole attracting the south pole of a second magnet, the charges on these two plates will be attracted to each other and maintain their charge. This is considered a logic '1'. The absence of a charge is considered a logic '0'.

Since a capacitor can be made very small, DRAM technology is better adapted to high density memories, i.e., cramming a great deal of bits into a small space.

Capacitors do have a problem though. Every once in a while, one of the electrons will escape from the negatively charged plate and land on the positively charged plate. This exchange of an electron decreases the overall charge difference on the two plates. If this happens enough, the stored '1' will disappear. This movement of electrons from one plate to the other is referred to as ***leakage current***.

The electrons stored on the plates of the capacitors are also lost when the processor reads the data. It requires energy to read data from the capacitors, energy that is stored by the position of the electrons. Therefore, each read removes some of the electrons.

In order to avoid having leakage current or processor reads corrupt the data stored on the DRAMs, i.e., turning the whole mess to zeros, additional logic called refresh circuitry is used that periodically reads the data in the DRAM then restores the ones with a full charge of electrons. This logic also recharges the capacitors each time the processor reads from a memory location. The refresh circuitry is included on the DRAM chip making the process of keeping the DRAM data valid transparent to the processor. Although it adds to the cost of the DRAM, the DRAM remains cheaper than SRAM.

The refresh process involves disabling memory circuitry, then reading each word of data and writing it back. This is performed by counting through the rows. The process does take time thus slowing down the apparent performance of the memory.

In general, DRAMs:

- have a much higher capacity due to the smaller size of the capacitor (The RAM sticks of your computer's main memory are DRAMs.);
- will "leak" charge due to the nature of capacitors eventually causing the data to disappear unless it is refreshed periodically;
- are much cheaper than SRAM; and
- are volatile meaning that the data is fixed and remains stored only as long as power is available.

12.5.4 Types of DRAM and Their Timing

The basic DRAM storage technology is unchanged since first RAM chips, but designers have used alternate techniques to improve the

effective performance of the DRAM devices. For example, the number of pins that electrically connect the DRAM to the system can be quite large, one of the largest culprits being addressing. A 1 Gbyte memory, for example, requires 30 pins for addressing. If the number of pins could be reduced, the memory would have a smaller footprint on the circuit board and a more reliable connection.

The number of address lines can be cut in half by presenting the memory address to the DRAM in two stages. During the first stage, the first half of the address is presented on the address lines and stored or latched in the memory device. The second stage presents the last half of the address on the same pins. Once it receives both halves of the address, the DRAM can process the request.

Time →

	Cycle 1	Cycle 2	Cycle 3
Address lines:	1st half of addr.	2nd half of addr.	
Data lines:	**No data**	**No data**	Valid data

Figure 12-11 Basic Addressing Process for a DRAM

This allows the addressable space of the DRAM to be twice that which could be supported by the address pins. Unfortunately, it comes at the cost of the delay of the second addressing cycle. It will be shown later, however, that this can be turned into an advantage.

The presentation of the address in two stages is viewed as a logical reorganization of the memory into three-dimensions. The upper half of the address identifies a row. The lower half of the address identifies a column. The intersection of the row and column is where the data is stored, the multiple bits of the storage location is the third dimension. This concept of rows and columns is represented in Figure 12-12 for a memory with four data bits per addressable location.

To support the two-stage delivery of the address, two additional active-low control signals are needed. The first is called *row address select* or $\overline{RAS}$. This signal is used to strobe the row address into the DRAM's row address latch. The second, column address select or $\overline{CAS}$, strobes the column address into the DRAM's column address latch. This requires two additional input pins on the DRAM.

By using the two-stage addressing of a DRAM, the addition of a single address line will quadruple the memory size. This is because an

additional address line doubles both the number of rows and the number of columns.

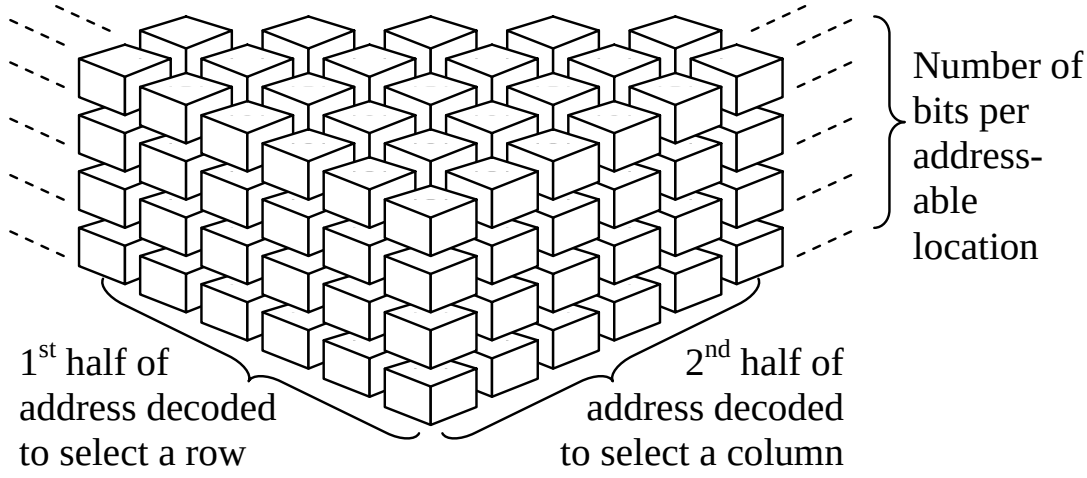


Figure 12-12 Organization of DRAM

Now let's get back to the issue of the delay added by a second address cycle. Most data transfers to and from main memory take the form of block moves where a series of instructions or data words are pulled from memory as a group. (For more information on memory blocks, see the section on caches in Chapter 13.)

If the processor needs a block from memory, the first half of the address should be the same for all the items of the block. Because of this, the memory access process begins with the row address then uses only the column address for subsequent retrievals. This is called Fast Page Mode (FPM), the data of a single row being referred to as a page. The $\overline{RAS}$ line is held low as long as the row address is valid. Figure 12-13 presents an example of FPM for a memory block of size four.

Time →

Address:	Row addr	Col. addr. 0		Col. addr. 1		Col. addr. 2		Col. addr. 3	
Data:	No data	No data	Data word 0	No data	Data word 1	No data	Data word 2	No data	Data word 3

Figure 12-13 Example of an FPM Transfer

FPM requires both the row and column addresses to be presented for the first cycle only. For subsequent addresses, only the column address is needed thereby saving one address cycle. Furthermore, since the decoding circuitry for the column address only uses half of the bits of the full address, the time it takes to decode each individual column address is shorter than it would have been for the full address.

The memory access time can be further shorted by having the processor present the column address for the next data element while the current data element is being returned on the data lines. This overlap of the DRAM returning a word of data while the processor is writing the column address for the next word is called Extended Data-Out (EDO). For data reads within the same page, EDO results in a savings of approximately 10 ns for each read. Figure 12-14 presents an example of EDO for a memory block of size four.

Time →

Address:	Row addr.	Column addr. 0	Column addr. 1	Column addr. 2	Column addr. 3	
Data:	No data	No data	Data word 0	Data word 1	Data word 2	Data word 3

Figure 12-14 Example of an EDO Transfer

If the processor needs to fetch a sequence of four data words from memory, Burst EDO (BEDO) can further speed up the process by using the initial column address, and then using a counter to step through the next three addresses. This means that the processor would only be required to send the row address and column address once, then simply clock in the four sequential data words.

12.5.5 Asynchronous versus Synchronous Memory Interfacing

In all logic circuits, there is a delay between the time that inputs are set and the outputs appear. The inputs of a memory include address lines and control lines while the data lines can be either inputs or outputs. When a processor sets the inputs of a memory device, it has to wait for the memory to respond. This is called ***asynchronous*** operation.

Asynchronous operation makes it difficult to design the timing of motherboards. The processor has to run slow enough to account for the slowest type and size of memory that is expected to be used. One

memory may be ready with the data before the processor expects it while a different memory may take longer.

Some processors, however, are designed so that the memory follows a precise timing requirement governed by a clock that is added to the bus to keep everything attached in lock-step. This type of interface is referred to as ***synchronous*** and the memory that uses this type of interface is referred to as synchronous DRAM or SDRAM.

The main benefit derived from controlling the memory interface with a clock is that the memory's internal circuitry can be set up in what is referred to as a pipelined structure. Pipelining is discussed in Chapter 15, but at this point it is sufficient to say that pipelining allows a device to perform multiple operations at the same time. For example, an SDRAM might be able to simultaneously output data to the processor and receive the next requested address. Overlapping functions such as these allows the memory to appear faster.

One improvement to SDRAM allows two words of data to be written or read during a single clock cycle. This means that every time the processor accesses the memory it transfers two words, one on the rising edge of the clock and one on the falling edge. This type of memory is referred to as Double Data Rate SDRAM or DDR SDRAM. DDR2 and DDR3 double and quadruple the DDR rate by allowing four and eight data words respectively to be transferred during a single clock pulse. This is made possible by taking advantage of the fact that the when a DRAM row is accessed, simply counting through the columns retrieves consecutive bytes of data which can be buffered and send out the data lines as quickly as the processor can receive them.

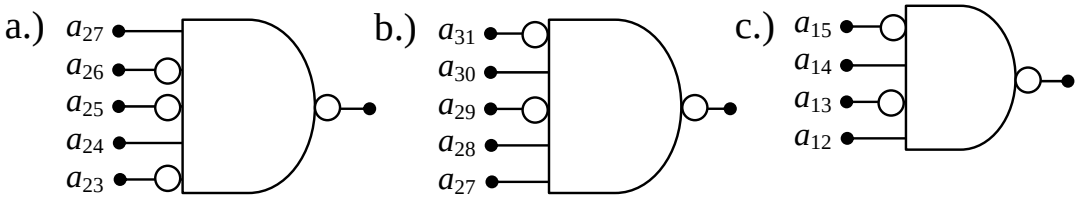
12.6 What's Next?

This chapter has only examined a small part of the information storage solutions used in a computer system. In the next section, we will discuss the operation and purpose of all levels of data storage by examining the different characteristics of each. The major characteristics that will be examined are speed, size, and whether data is lost with a power loss. Each level has specific needs and therefore is implemented with different technologies.

Problems

1. What is the largest memory that can have a starting or lowest address of 160000_{16}?

2. What are the high and low addresses of the memory ranges defined by each of the chip selects shown below?



3. What is the processor memory space for each chip select in problem 2?
4. What is the memory device size for each chip select in problem 2?
5. How many 16 K memories can be placed (without overlapping) in the memory space of a processor that has 24 address lines?
6. Using logic gates, design an active low chip select for the memory device described in each of the following situations.

   a.) A 256 K memory device starting at address 280000_{16} in a 4 Meg memory space

   b.) A memory device in the range 30000_{16} to $37FFF_{16}$ in a 1 Meg memory space.

7. How many latches are contained in a SRAM that has 20 address lines and 8 data lines?
8. True or false: DRAM is faster than SRAM.
9. True or false: DRAM is cheaper per bit than SRAM.
10. True or false: More DRAM can be packed into the same area (higher density) than SRAM.
11. Which is usually used for smaller memories, DRAM or SRAM?
12. When data is passed from a memory chip to the processor, what values do the bus signals $\overline{R}$ and $\overline{W}$ have?
13. What is the subnet and host id of the Class C IPv4 address 195.164.39.2?
14. Taking into account the addresses for the subnet and broadcast, how many hosts can be present on a Class C IPv4 subnet?