COMPUTER PERIPHERALS AND INTERFACING

Recommended Texts

- Dr. M. Rafiquzzaman, “Microprocessors and Microcomputer-based System Design”
- Douglas V. Hall, “Microprocessors and Interfacing – Programming and Hardware”
- N. Mathivanan, “Microprocessors, PC Hardware and Interfacing”
- Jyoti Snehi, “Computer Peripherals and Interfacing”

Peripherals

- Peripherals are any device that can be attached to a CPU, e.g. hard disk, mouse, printer.

- Peripherals are slow. They hold up the CPU from carrying out its' tasks.

- For Example - playing a game and it needs to load the next level. The level is being read from the CD and written to memory.

Interfaces

- **What is an interface?**
  - The interface is the combination of hardware and software needed to link the CPU to the peripherals and to enable them to communicate with the CPU despite all their differing characteristics.

    - The hardware is the bit you connect the cable into e.g. USB.

    - The software is the driver disk that you usually need to install when you get the device, e.g. printer driver.

Interfaces

- Computer peripherals have different characteristics. For example, they may:
  - Have different data transfer rates;
  - Use a wide variety of codes and control signals;
  - Transmit data in serial or in parallel form;
  - Even work at higher voltages than the CPU; &
  - All operate at much slower speeds than the CPU.

Interfaces

- The main functions of an interface that you need to know about are:
    - Buffering;
    - Converting data to and from serial and parallel forms;
    - Converting data to and from analogue and digital forms;
    - Voltage conversion;
    - Protocol conversion; &
    - Handling of status signals.

Interfaces

- **Buffering**:
  - This is an area of RAM within the interface which stores the data while in transit between the processor and the peripheral.
  - The interface uses the buffer to temporarily store the data it is working with.
  - It also uses the buffer to compensate for the differences in speed between the peripherals and the CPU by temporarily storing incoming data so that the faster CPU can process it in manageable blocks rather than waiting for the slower peripheral.

Interfaces

- **Converting data to and from serial and parallel forms:**
  - Data transmission is the passing of data from one device to another.
  - A **serial interface** uses **serial data transmission;**
  - A **parallel interface** uses **parallel data transmission**.

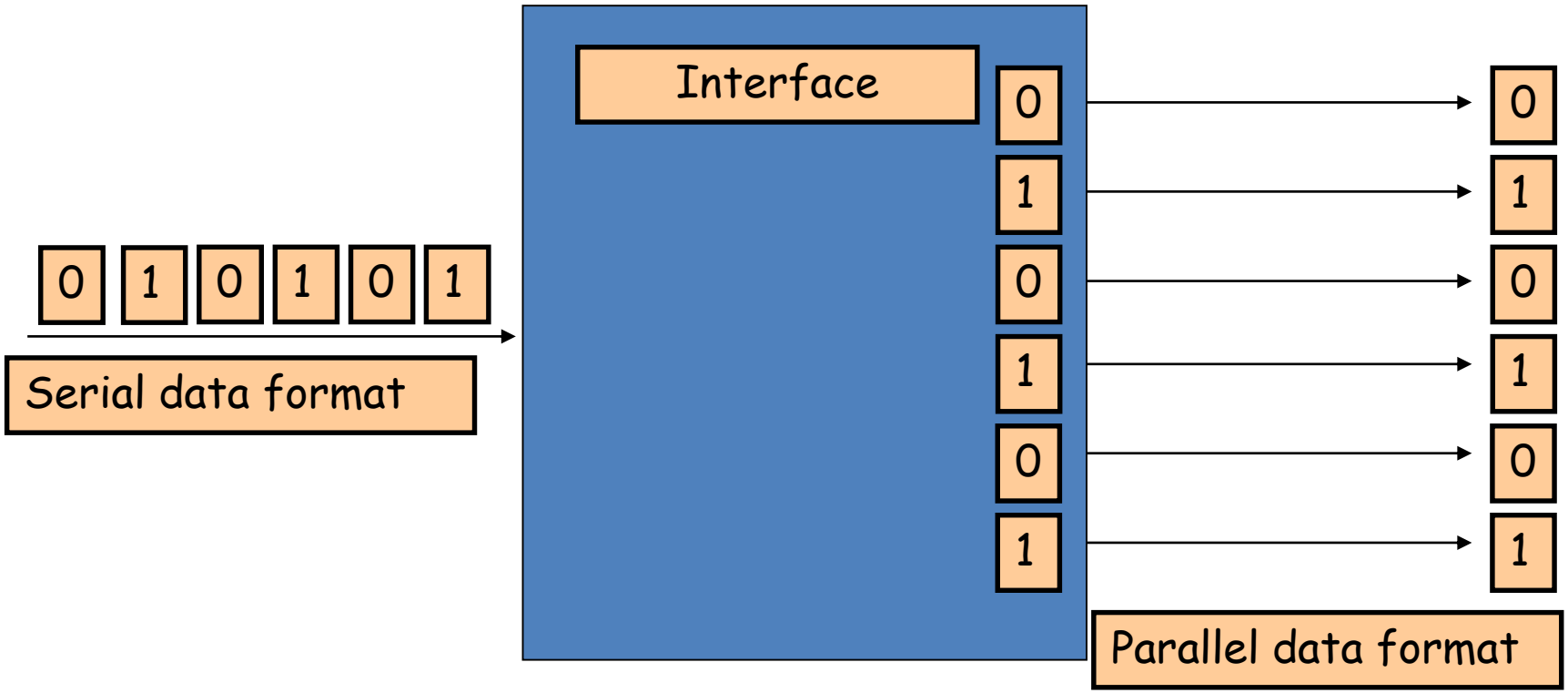
  - **Serial data transmission** - is when data is transmitted along a communication channel one bit after another in sequence. Very slow but efficient over long distances.

Interfaces

- **Parallel data transmission** - transmit several bits of data simultaneously across a series of parallel channels, often transmitting 16 or 32 bits at a time. Very fast but only suitable for short distances.

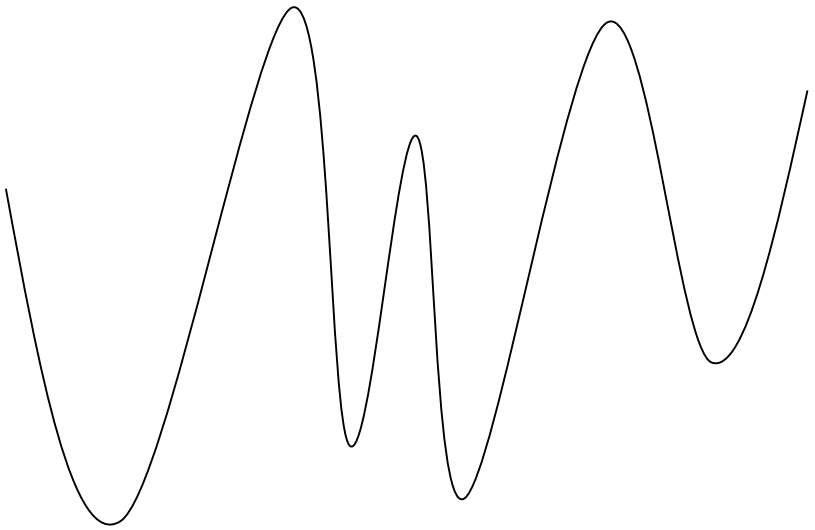
- The buses internal to the processor are parallel channels. Any data coming from a serial device has to be sent to an interface which buffers the data then converts it to parallel form before it is passed to the processor.

Interfaces


0
1
0
1
0
1
Serial data format
Interface
0
1
0
1
0
1
0
1
0
1
0
1
Parallel data format

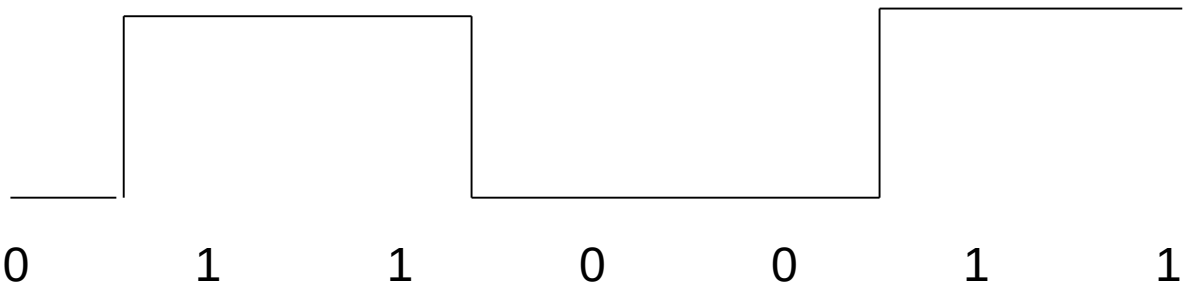
Interfaces

- **Converting data to and from analogue and digital forms:**
  - **Analogue signals** – many electrical signals are analogue signals. These signals vary between two limits. Analogue signals that are sent in from peripherals to the digital form that the CPU can handle. If you could see an analogue signal it would look roughly like this:



Interfaces

- **Digital signals** – computers can only work with digital signals, which have only two values – on or off. A digital signal therefore consists of a series of 'ons' and 'offs'. An on signal is represented by a 1 and an off by a 0



Interfaces

- **ADC and DAC**
  - A computer is connected to a peripheral by an interface. This interface has to be able to change the digital signals from the computer to an analogue signal that the other device can understand. This is done by a DAC – **Digital to Analogue Converter**.
  - Signals can be changed in the other direction by an ADC – **Analogue to Digital Converter**.

Interfaces

- **Voltage Conversion**
  - Peripherals send data using a different voltage from that used by the processor and its associated components on the motherboard of the computer.
  - An interface is used to compensate for these differences.

Interfaces

- **Protocol Conversion**
  - A protocol is a standard that enables the connection, communication and data transfer between computers or between a computer system and a peripheral. Protocol conversion means ensuring that the protocols used by the peripheral can be understood by the computer it is attached to and vice versa.

- **Handling of Status Signal**
  - The purpose of the status information is to show whether or not a peripheral device is ready to communicate. This information is used to inform the user of a problem requiring attention.

Buffers and Spoolers

- Buffers and spoolers are terms generally associated with printers.
- The terms are not dedicated to printers but that is the context that will be described.

Buffers and Spoolers

- **What is a Buffer?**
  - A buffer is an area of memory used for the transfer of data between a computer and a peripheral.
  - A buffer provides temporary storage of data.
  - Using a buffer provides a link between a device and the processor and helps compensate for any differences in their working speeds.

Buffers and Spoolers

- **Why use buffers?**
  - Peripherals operate at much slower speeds than the CPU. Using buffers helps the computer system compensate for the differences in operating speeds between CPU and its peripherals.
  - When transferring data out to a peripheral such as a printer, the processor can transfer it faster into the buffer and the buffer will send it to the printer at a speed that the printer can cope with.
  - The use of buffers reduces the frequency with which the CPU is interrupted to deal with input. Data from a keyboard is stored in a buffer until there is a larger amount of data for the CPU to process.

Buffers and Spoolers

- **Spoolers**
  - Spooling is another technique used in the transfer of data to a slow peripherals. In this case the data intended for the peripheral, the best example is a printer, is transferred to storage, often a hard disk. Then when the processor is idle it will transfer the data to the printer at an acceptable speed. This is also called **background printing**.
  - This frees up the much faster CPU to process other tasks. Spooling is another possible method of improving system performance.

Buffers and Spoolers

- **Buffer v's Spooler**
  - If the CPU is very busy and doesn't get much idle time then spooling can be a very slow process.
- **Spooler v's Buffer**
  - A buffer is limited by the amount of RAM whereas a spooler uses backing storage which has a very large capacity.
- Use BOTH for optimum efficiency.

Current Interface Trends and Wireless Comms.

- New interfaces are continually being developed.
  - One focus of development is to increase the speed at which interfaces allows the peripheral and computer to communicate.
  - Another aim is to allow wireless communication between peripheral devices and the CPU.

Increasing Interface Speeds

- Interface speeds are measured in Megabits per second (Mbps).
  - N.B. do not confuse Megabits and Megabytes
  - Manufacturers normally use Megabits per second in their advertising as this allows them to print larger numbers on their advertisements.

Increasing Interface Speeds

- One example of increasing interface speeds is the development of the *USB 2* and the *Firewire 800* interfaces.
  - The *USB 2* improves upon the maximum 12 Mbps speed of the *USB 1* interface by 40 times to 480 Mbps.

  - The *Firewire 800* interface provides 800 Mbps, double the speed of the *Firewire 400* interface.

Interface Standards

Interface	*Description*
RS232	Recommended Standard (Serial)
SCSI	Small computer Systems Interface (Parallel)
IDE	Integrated Drive Electronics
SATA	Serial Advanced Technology Attachment – up to 1.5Gbps
IEEE	Institute of Electrical and Electronic Engineers e.g. firewire
MIDI	Musical Instrument Digital Interface
PCI	Peripheral Component Interconnect
PCMCIA	Personal Computer Memory Card International Association
USB1 & 2	Universal Serial Bus

- The use of interface standards by a computer manufacturer means that their computers will be able to connect to peripherals using the same standards.
- The use of interface standards by peripheral manufacturers means that their peripherals will connect to a computer which uses the same standards.
- Makes them COMPATABLE!!!

The latest Interfaces

Interface	*Description*	*Transfer Speeds*
USB1	Universal Serial Bus A means of connecting external devices	Fast transfer rate: 12 MBPS for fast devices
USB2	An improvement of the USB.	Up to 480 Mbps
Firewire	A high speed serial interface used for connecting audio/ visual and multimedia applications like digital camcorders.	In its latest version, IEEE 1394b, up to a max of 800Mbps. 3.2Gbps are under Development.

Wireless Communication

- Current trends in wireless communication include the standards WiFi and Bluetooth.

- Bluetooth and WiFi both use radio waves at the same frequency.

  - Radio waves can pass through most materials and walls, and devices do not need to be pointing at one another, unlike, infrared data transmission (TV remote control).

Wireless Communication

Bluetooth

- Bluetooth can make short-range links between personal devices, such as mobile phones and headsets, palmtops (PDA) and laptop (notebook) computers.
- Bluetooth is also used for wireless keyboards and mouse.
- It is expected that later versions of Bluetooth will be able to transmit data at a speed of 2 Mbps and for longer distances

Wireless Communication

WiFi

- WiFi stands for the Wireless Fidelity Alliance.

- The main use for WiFi is in wireless local area networking (WLAN). WiFi devices have typical ranges from 15 to 50 meters and typical data transfer rates from 5 to 20 Mbps.

Solid State Storage Devices

- A solid-state storage device contains no moving parts.
  - Examples include flash cards and USB flash memory
    - Both of these types of device contain the same type of backing storage medium, namely *flash* ROM.
    - The effective difference between them is that they use different interfaces to connect to a computer system or another peripheral.

Flash Cards

- Flash cards are used mainly for data storage in cameras, although they can hold any type of program or data file.
- There a number of different standards of flash cards.
- Each digital camera normally uses only one type of flash card
- Specialised card readers have been developed which have 'slots' to fit all the different varieties of flash cards.
  - So called 'all-in-one' devices (printer, scanner, photocopies) have slots into which flash cards may be placed. This allows documents to be printed directly from the card, without a computer system having to be connected.

USB *Flash* Memory

- There are two types of USB *Flash* Memory, according to the type of interface being used.
  - The are USB 1 and USB 2.

- Most of these devices are now compatible with USB 2.

Features and Advantages of Solid-State Storage Devices

- Solid-state storage devices are *small*.
- Flash cards can fit inside the camera.
- USB flash memory can fit on key rings and in watches.
- Solid-state storage devices are robust/strong because they have no moving parts.
  - This means that they are ideal for wearing because they are unaffected by vigorous movement.
  - iPods take advantage of this feature of Solid-state storage devices.

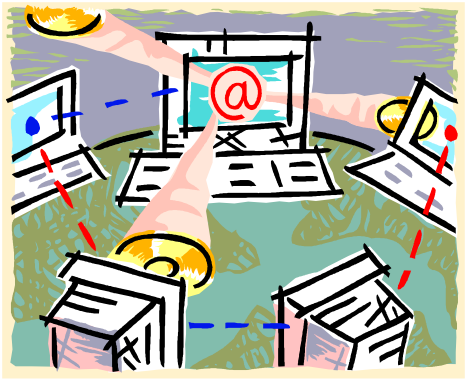
Features and Advantages of Solid-State Storage Devices

- Solid-State storage devices *use less power* than hard disk drives.
  - This lower power requirement means that a music player which uses solid-state storage will play music for a longer time relative to a hard disk-based music player with the same battery capacity.

Features and Advantages of Solid-State Storage Devices

- Solid-state storage devices are available in a range of capacities, from 256 Mb to 8 Gb (at the time of writing).
- USB flash memory has now replace floppy disk as a convenient, portable storage medium, which can fit into almost any modern computer system.
- USB flash memory is used for security applications.
  - One type has fingerprint recognition and another type works with security software to prevent a computer system from starting up unless the USB flash memory is plugged in.

Developments in Backing Storage Devices



Increased capacity

DVD

- The DVD-Recordable format capacity was increased with the introduction of the dual layer DVD-R disk with a capacity of 8.4 Gb (Single –layer DVD-R capacity is 4.7 Gb)

- Most computers now incorporate this dual layer drive.

Developments in Backing Storage Devices

Blu-ray

- Blu-ray disc (BD) is the name of the next generation optical disc format.
- The format was developed to enable recording, rewriting and playback of high-definition video (HD).
- The format may become a standard for PC data storage and high-definition movies.

Developments in Backing Storage Devices

Increased read and write speeds

- Faster interfaces are constantly being developed, e.g. USB 2, *Firewire 800*, with speeds of 480 Mbps and 800 Mbps, respectively

Reduced Physical Size

- In 2004, Guinness World Records certified that Toshiba's 0.85 inch diameter hard disk drive as the smallest hard disk drive in the world.

Lower Cost Per Unit of Storage

- In 1995, a 100 Mb capacity hard disk drive cost £300. In 2005, a 100 Mb capacity ZIP drive cost £5, and a 512 Mb USB flash memory cost £25.

Developments in Backing Storage Devices

Implications of Development Trends

- One implication may be that new peripherals and media may not work with old 'legacy' hardware.
  - An older computer may not have a USB port, or if it does, it may not be USB 2, only USB 1.
  - A DVD re-writer may not be able to read or write to new disk formats.
  - Care must also be taken with archived data, to make sure that it is regularly copied to a current storage format so that it can always be accessed.

- In 2004 we say some electrical stores discontinue the sales of VHS video recorders in favour of DVD.

- Digital video cameras, which record directly to DVD instead of tape are now common.

Types of Computers

- Mainframe
- Super Computer
- Mini Computer
- Micro Computer

Mainframe

- The largest and most powerful computer
- They are designed to work at very high speed
- Large data words, typically 64 bits or greater
- They have massive amount of memory
- Used in military defense control, business data processing, computer graphic display.
- Example: IBM 4381

Super Computer

- The fastest and more powerful mainframes are called Super Computer
- Example: Cray Y-MP/ 832
- Used by largest firms, government agencies and universities

Mini Computer

- Scaled-down versions of mainframe Computer
- Runs slowly, works with smaller data word
- Does not have as much memory as mainframe
- Used in scientific research and industrial control

Micro Computer

- Small computer
- CPU is usually a single microprocessor
- Example: Desktop, Laptop, Palmtop

Intel Core i7

Intel 8086

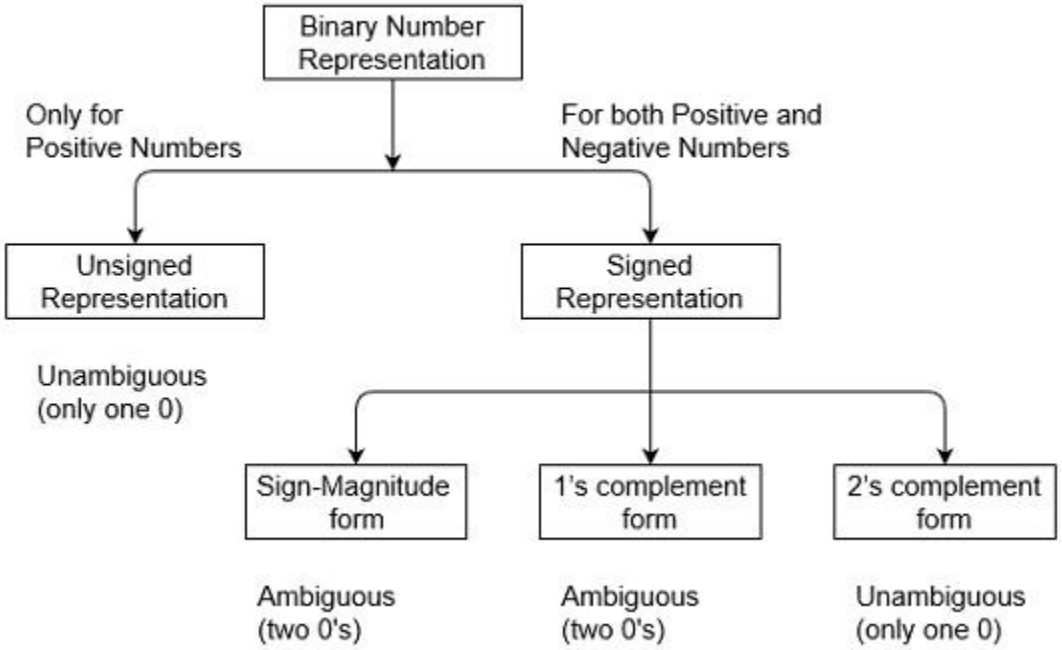
Microprocessor Data Type

- Bit
- Byte
- Word
- Unsigned and Signed Binary Integers
- BCD (Binary Coded Decimal) Numbers
- ASCII
- Floating Point Numbers

Microprocessor data type

- Bit:
  - smallest unit of information
  - It represents either 1 or 0
- Byte:
  - 8 bits of data
- Word:
  - Data that is handled by a microprocessor at a time
  - Ex: 8 bit, 16 bit, 32 bit word

Unsigned and Signed Binary Integers



Unsigned And Signed Binary Integers

- An unsigned binary integer has no arithmetic sign
- Example of unsigned integer is memory address
- Unsigned uses the leading bit as a part of the value, while the signed version uses the left-most-bit to identify if the number is positive or negative.
- Signed integer is represented in true form for a positive number and in two's complement form for a negative number

Representation of Unsigned Binary Numbers

Example-1: Represent decimal number 92 in unsigned binary number.

Solution: Simply convert it into Binary number, it contains only magnitude of the given number.

$= (92)_{10}$

$= (1x2^6+0x2^5+1x2^4+1x2^3+1x2^2+0x2^1+0x2^0)_{10}$

$= (1011100)_2$

It's 7 bit binary magnitude of the decimal number 92.

Example-2: Find range of 5 bit unsigned binary numbers. Also, find minimum and maximum value in this range.

Solution: Since, range of unsigned binary number is from 0 to (2^n-1). Therefore, range of 5 bit unsigned binary number is *from* 0 to (2^5-1) which is equal from minimum value 0 (i.e., 00000) to maximum value 31 (i.e., 11111).

Representation of Signed Binary Numbers

(a) Sign-Magnitude form:

The range of Sign-Magnitude form is from $-(2^{(n-1)}-1)$ to $(2^{(n-1)}-1)$.

For example, range of 6 bit Sign-Magnitude form binary number is from $-(2^5-1)$ to (2^5-1) which is equal from minimum value -31 (i.e., 1 11111) to maximum value +31 (i.e., 0 11111). And zero (0) has two representation, -0 (i.e., 1 00000) and +0 (i.e., 0 00000).

(b) 1's complement form:

The range of 1's complement form is from $-(2^{(n-1)}-1)$ to $(2^{(n-1)}-1)$.

For example, range of 6 bit 1's complement form binary number is from $-(2^5-1)$ to (2^5-1) which is equal from minimum value -31 (i.e., 1 00000) to maximum value +31 (i.e., 0 11111). And zero (0) has two representation, -0 (i.e., 1 11111) and +0 (i.e., 0 00000).

(c) 2's complement form:

The range of 2's complement form is from $-(2^{(n-1)})$ to $(2^{(n-1)}-1)$.

For example, range of 6 bit 2's complement form binary number is from (2^5) to (2^5-1) which is equal from minimum value -32 (i.e., 1 00000) to maximum value +31 (i.e., 0 11111). And zero (0) has one representation, +0 (i.e., 0 00000).

Unsigned And Signed Binary Integers

- The range of unsigned binary number is from 0 to (2^n-1).
- For 8-bit number system
  - Unsigned: It consists of only non-negative values i.e 0 to 255.
  - Signed: It consist of both negative and positive values but in different formats like
    - 0 to +127
    - -1 to -128

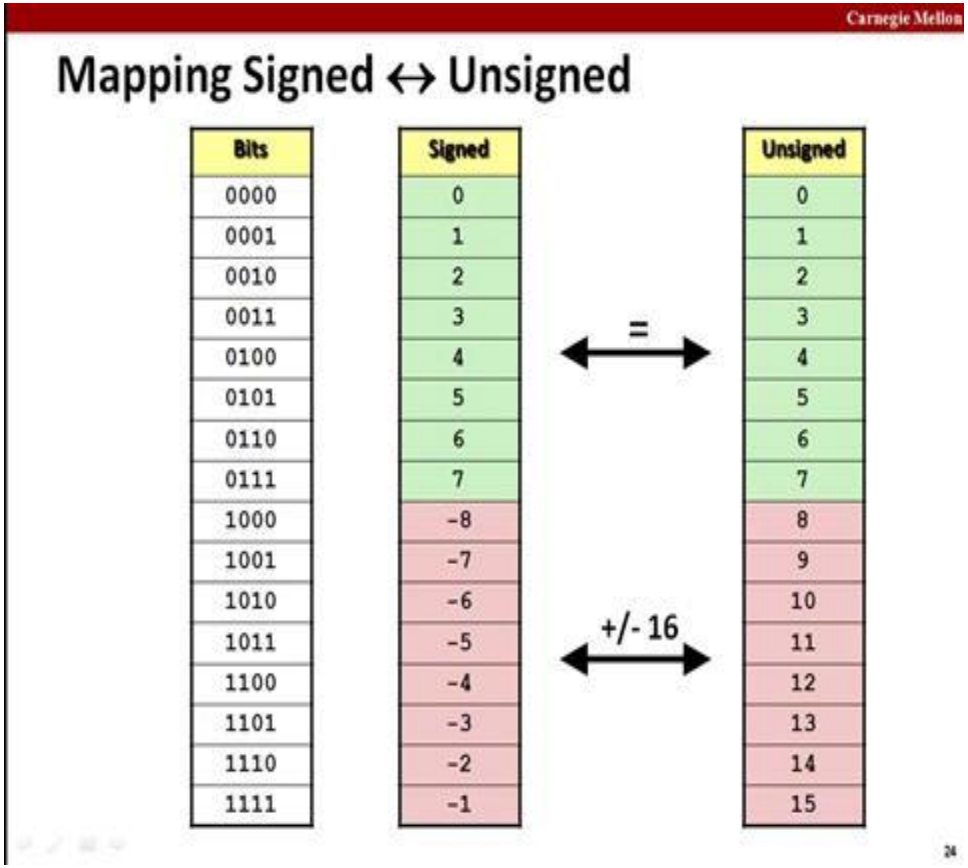
Ranges of Signed Integers

The unsigned range is divided into two signed ranges for positive and negative numbers

Storage Type	Range (low–high)	Powers of 2
Signed byte	−128 to +127	-2^7 to (2^7-1)
Signed word	−32,768 to +32,767	-2^{15} to $(2^{15}-1)$
Signed doubleword	−2,147,483,648 to 2,147,483,647	-2^{31} to $(2^{31}-1)$
Signed quadword	−9,223,372,036,854,775,808 to +9,223,372,036,854,775,807	-2^{63} to $(2^{63}-1)$

Practice: What is the range of signed values that may be stored in 20 bits?

Unsigned And Signed Binary Integers



BCD (Binary Coded Decimal) Numbers

- Each decimal digit is represented by four bit binary number
- Microprocessor stores BCD numbers in two forms, packed and unpacked
- The unpacked BCD number represents each BCD digit as a byte
- The packed BCD number represents two BCD digits in a byte

ASCII

- Each character is represented by an integer.
- This code represents alphanumeric in a microprocessor’s memory
- It also represents special symbols

Floating-Point Number

- Floating point numbers contains three components – sign, exponent and mantissa
- For the decimal value -2.5x10^-2, sign is negative, exponent is -2 and mantissa is 2.5
- A binary floating point number is represented as a normalized binary fraction raised to a power of 2

Floating-Point Number

- **The Conversion Procedure:**
  - Convert the absolute value of the number to binary, perhaps with a fractional part after the binary point. This can be done by converting the integral and fractional parts separately.
  - Append × 2^0 to the end of the binary number (which does not change its value).
  - Normalize the number. Move the binary point so that it is one bit from the left. Adjust the exponent of two so that the value does not change.
  - Place the mantissa into the mantissa field of the number. Omit the leading one, and fill with zeros on the right.
  - Add the bias to the exponent of two, and place it in the exponent field. The bias is 2^(*k*−1) − 1, where *k* is the number of bits in the exponent field. For the eight-bit format, *k* = 3, so the bias is 2^(3−1) − 1 = 3. For IEEE 32-bit, *k* = 8, so the bias is 2^(8−1) − 1 = 127.
  - Set the sign bit, 1 for negative, 0 for positive, according to the sign of the original number.

Using The Conversion Procedure

•Convert 2.625 to our 8-bit floating point format.

1.The integral part is easy, $2_{10} = 10_2$. For the fractional part:

0.625	× 2 =	1.25	1	*Generate 1 and continue with the rest.*
0.25	× 2 =	0.5	0	*Generate 0 and continue.*
0.5	× 2 =	1.0	1	*Generate 1 and nothing remains.*

So $0.625_{10} = 0.101_2$, and $2.625_{10} = 10.101_2$.

2.Add an exponent part: $10.101_2 = 10.101_2 \times 2^0$.

3.Normalize: $10.101_2 \times 2^0 = 1.0101_2 \times 2^1$.

4.Mantissa: 0101

5.Exponent: $1 + 3 = 4 = 100_2$.

6.Sign bit is 0.

The result is

0	100	0101

.

Represented as hex, that is 45_{16}.

Try Yourself: Convert -4.75 to our 8-bit floating point format.

Using The Conversion Procedure

- Convert -4.75 to our 8-bit floating point format.
  - The integral part is $4_{10} = 100_2$. The fractional:

0.75	× 2 =	1.5	1	*Generate 1 and continue with the rest.*
0.5	× 2 =	1.0	1	*Generate 1 and nothing remains.*

So $4.75_{10} = 100.11_2$.

  - Normalize: $100.11_2 = 1.0011_2 \times 2^2$.
  - Mantissa is 0011, exponent is $2 + 3 = 5 = 101_2$, sign bit is 1.
  - So -4.75 is 1 101 0011 = $d3_{16}$

Using The Conversion Procedure

- Convert 0.40625 to our 8-bit floating point format.
  a. Converting:

$0.40625 \times 2 = 0.8125$	$\boxed{0}$	*Generate 0 and continue.*
$0.8125 \times 2 = 1.625$	$\boxed{1}$	*Generate 1 and continue with the rest.*
$0.625 \times 2 = 1.25$	$\boxed{1}$	*Generate 1 and continue with the rest.*
$0.25 \times 2 = 0.5$	$\boxed{0}$	*Generate 0 and continue.*
$0.5 \times 2 = 1.0$	$\boxed{1}$	*Generate 1 and nothing remains.*

So $0.40625_{10} = 0.01101_2$.

  b. Normalize: $0.01101_2 = 1.101_2 \times 2^{-2}$.

  c. Mantissa is 1010, exponent is $-2 + 3 = 1 = 001_2$, sign bit is 0.

So 0.40625 is $\boxed{0}\boxed{001}\boxed{1010} = 1\mathbf{a}_{16}$

- Convert -12.0 to our 8-bit floating point format.
  a. $12_{10} = 1100_2$.

  b. Normalize: $1100.0_2 = 1.1_2 \times 2^3$.

  c. Mantissa is 1000, exponent is $3 + 3 = 6 = \mathbf{110}_2$, sign bit is 1.

So -12.0 is $\boxed{\mathbf{1}}\boxed{\mathbf{110}}\boxed{1000} = \mathrm{e}8_{16}$

Using The Conversion Procedure

- Convert decimal 1.7 to our 8-bit floating point format.
  a. The integral part is easy, $1_{10} = 1_2$. For the fractional part:

$0.7 \times 2 = 1.4$ [1] *Generate 1 and continue with the rest.*
$0.4 \times 2 = 0.8$ [0] *Generate 0 and continue.*
$0.8 \times 2 = 1.6$ [1] *Generate 1 and continue with the rest.*
$0.6 \times 2 = 1.2$ [1] *Generate 1 and continue with the rest.*
$0.2 \times 2 = 0.4$ [0] *Generate 0 and continue.*
$0.4 \times 2 = 0.8$ [0] *Generate 0 and continue.*
$0.8 \times 2 = 1.6$ [1] *Generate 1 and continue with the rest.*
$0.6 \times 2 = 1.2$ [1] *Generate 1 and continue with the rest.*

...

The reason why the process seems to continue endlessly is that it does. The number 7/10, which makes a perfectly reasonable decimal fraction, is a repeating fraction in binary, just as the faction 1/3 is a repeating fraction in decimal. (It repeats in binary as well.) We cannot represent this exactly as a floating point number. The closest we can come in four bits is .1011. Since we already have a leading 1, the best eight-bit number we can make is 1.1011.

  b. Already normalized: $1.1011_2 = 1.1011_2 \times 2^0$.
  c. Mantissa is 1011, exponent is $0 + 3 = 3 = 011_2$, sign bit is 0.

The result is | 0 | 011 | 1011 | $= 3\mathbf{b}_{16}$. This is not exact, of course. If you convert it back to decimal, you get 1.6875.

Using The Conversion Procedure(IEEE 32 bit)

- Convert -1313.3125 to IEEE 32-bit floating point format.

  a. The integral part is $1313_{10} = 10100100001_2$. The fractional:

  $0.3125 \times 2 = 0.625$ | 0 | *Generate 0 and continue.*

  $0.625 \times 2 = 1.25$ | 1 | *Generate 1 and continue with the rest.*

  $0.25 \times 2 = 0.5$ | 0 | *Generate 0 and continue.*

  $0.5 \times 2 = 1.0$ | 1 | *Generate 1 and nothing remains.*

  So $1313.3125_{10} = 10100100001.0101_2$.

  b. Normalize: $10100100001.0101_2 = 1.0100100010101_2 \times 2^{10}$.

  c. Mantissa is 01001000010101000000000, exponent is $10 + 127 = 137 = 10001001_2$, sign bit is 1.

So -1313.3125 is | 1 | 10001001 | 01001000010101000000000 | $= c4a42a00_{16}$

Using The Conversion Procedure(IEEE 32 bit)

- Convert 0.1015625 to IEEE 32-bit floating point format.
  a. Converting:

$0.1015625 \times 2 = 0.203125$	0	*Generate 0 and continue.*
$0.203125 \times 2 = 0.40625$	0	*Generate 0 and continue.*
$0.40625 \times 2 = 0.8125$	0	*Generate 0 and continue.*
$0.8125 \times 2 = 1.625$	1	*Generate 1 and continue with the rest.*
$0.625 \times 2 = 1.25$	1	*Generate 1 and continue with the rest.*
$0.25 \times 2 = 0.5$	0	*Generate 0 and continue.*
$0.5 \times 2 = 1.0$	1	*Generate 1 and nothing remains.*

  So $0.1015625_{10} = 0.0001101_2$.

  b. Normalize: $0.0001101_2 = 1.101_2 \times 2^{-4}$.

  c. Mantissa is 10100000000000000000000, exponent is -4 + 127 = 123 = 01111011_2, sign bit is 0.

  So 0.1015625 is | 0 | 01111011 | 10100000000000000000000 | = $3dd00000_{16}$

- Convert 39887.5625 to IEEE 32-bit floating point format.
  a. The integral part is $39887_{10} = 1001101111001111_2$. The fractional:

$0.5625 \times 2 = 1.125$	1	*Generate 1 and continue with the rest.*
$0.125 \times 2 = 0.25$	0	*Generate 0 and continue.*
$0.25 \times 2 = 0.5$	0	*Generate 0 and continue.*
$0.5 \times 2 = 1.0$	1	*Generate 1 and nothing remains.*

  So $39887.5625_{10} = 1001101111001111.1001_2$.

  b. Normalize: $1001101111001111.1001_2 = 1.00110111100111111001_2 \times 2^{15}$.

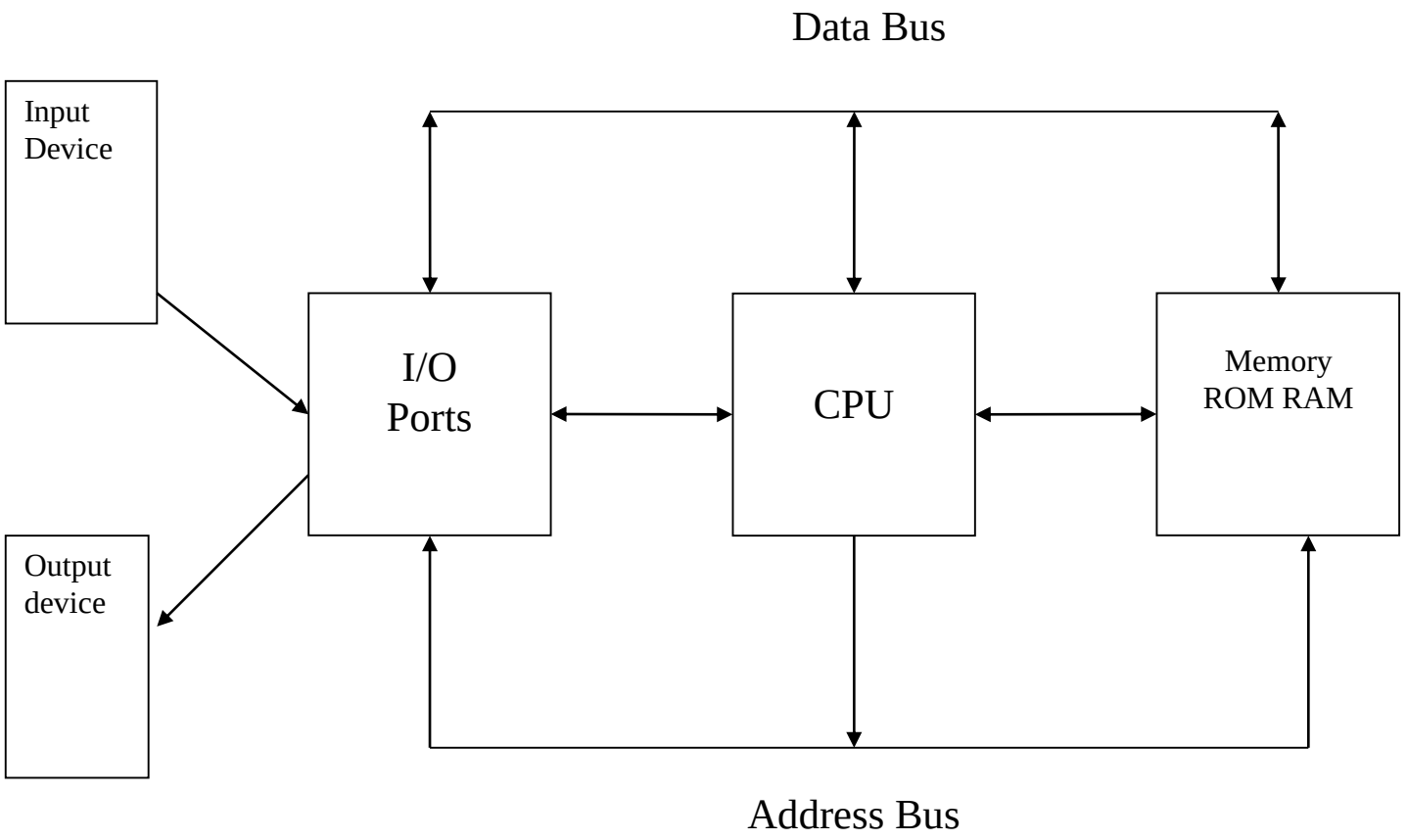
  c. Mantissa is 00110111100111110010000, exponent is 15 + 127 = 142 = 10001110_2, sign bit is 0.

  So 39887.5625 is | 0 | 10001110 | 00110111100111110010000 | = $471bcf90_{16}$

Using The Conversion Procedure(IEEE 64 bit)

- **A number in 64 bit double precision IEEE 754 binary floating point standard representation requires three building elements:**
  - **sign (it takes 1 bit and it's either 0 for positive or 1 for negative numbers),**
  - **exponent (11 bits),**
  - **mantissa (52 bits)**
  - **Bias = 1023**

Overview of Microcomputer Structure and Operation



Major Parts:

- CPU
- Memory
- Input / Output circuitry
- Buses:
  - Address bus
  - Data bus
  - Control bus

Memory:

- It stores the binary codes for the sequences of instructions
- It stores binary coded data
- Example: ROM, RAM, magnetic / optical disks

Input / Output:

- They are used to take in data from outside world or send data to the outside world
- I/O devices are connected with microprocessor through I/O ports
- Example: Keyboards, video display terminals, printers, modems

Central Processing Unit:

- It controls the operation of computer
- The CPU fetches binary-coded instructions from memory
- Decodes the instructions into a series of simple actions
- Carries out these actions in a sequence of steps
- Important components: General purpose register, dedicated registers and control bus signal generating circuits

Registers

(a) General Purpose Registers –

The 8085 has six general-purpose registers to store 8-bit data; these are identified as- B, C, D, E, H, and L. These can be combined as register pairs – BC, DE, and HL, to perform some 16-bit operation. These registers are used to store or copy temporary data, by using instructions, during the execution of the program.

(b) Specific Purpose Registers –

Accumulator: The accumulator is an 8-bit register (can store 8-bit data) that is the part of the arithmetic and logical unit (ALU). After performing arithmetical or logical operations, the result is stored in accumulator. Accumulator is also defined as register A.

Flag registers: Sign Flag, Zero Flag, Parity Flag, Carry Flag, Overflow Flag

(c) Memory Registers – Program Counter, Stack Pointer

Registers

- At the beginning we need to emphasize two points:
  - ALU doesn't care whether you are doing signed or unsigned mathematics.
  - ALU just does the binary math and sets the flags appropriately.
- In signed numbers, **the sign bit** is a bit that indicates the sign of the number. and It's located in the most significant bit.
- In **unsigned** arithmetic, we need to watch the **carry flag** to detect errors.
- In **signed** arithmetic, we need to watch the **overflow flag** to detect errors.
- The **Carry Flag** is set if:
  - the addition of two numbers causes a carry out of the most significant (leftmost) bits added.
    - 1111 + 0001 = 0000 => carry flag is turned on.
  - the subtraction of two numbers requires a borrow into the most significant (leftmost) bits subtracted.
    - 0000 - 0001 = 1111 => carry flag is turned on.

Registers

- The **Overflow Flag** is set if:
  - the sum of two numbers with the sign bit off yields a result number with the sign bit on.
    - 0100 + 0100 = 1000 => overflow flag is turned on.
  - the sum of two numbers with the sign bit on yields a result number with the sign bit off.
    - 1000 + 1000 = 0000 => overflow flag is turned on.
- *Note that*: Mixed-sign addition never turns on the overflow flag.
- To make it more clear let's see following examples:
  - assume the following binary addition `0111 + 0010 = 1001`
  - if we are doing **unsigned** mathematics
    - 0111 + 0010 = 1001 // overflow flag is turned on.
    - 7 + 2 = 9 // its ok, we only care about the carry flag
  - if we are doing **signed** mathematics
    - 0111 + 0010 = 1001 // overflow flag is turned on.
    - 7 + 2 = -7 // error detected

Mapping Signed ↔ Unsigned

Bits	Signed		Unsigned
0000	0		0
0001	1		1
0010	2		2
0011	3	=	3
0100	4		4
0101	5		5
0110	6		6
0111	7		7
1000	-8		8
1001	-7		9
1010	-6		10
1011	-5	+/- 16	11
1100	-4		12
1101	-3		13
1110	-2		14
1111	-1		15

Registers

- assume the following binary addition `1111 + 0001 = 0000`
- if we are doing unsigned mathematics
  - 1111 + 0001 = 0000 // carry flag is turned on.
  - 15 + 1 = 0 // error detected
- if we are doing signed mathematics
  - 1111 + 0001 = 0000 // carry flag is turned on.
  - -1 + 1 = 0 // its ok we only care about the overflow flag

- Remember: Mixed-sign addition never turns on the overflow flag.

Address Bus:

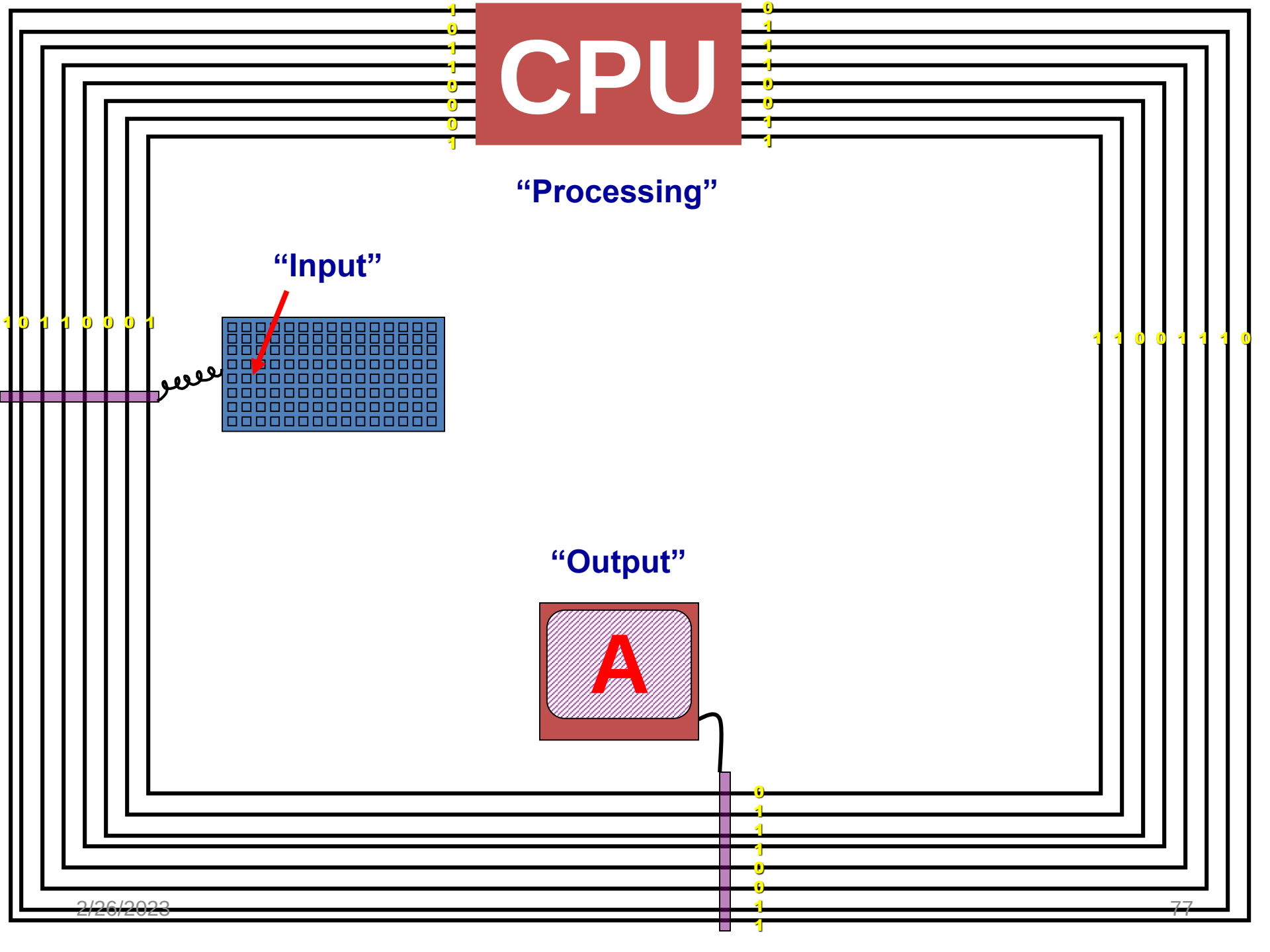
- It consists of 16, 20, 24, 32 or 36 parallel unidirectional signal lines
- On these lines the CPU sends out the address of the memory location or I/O port that is to be written to or read from
- The number of locations that the CPU can address is determined by the number of address lines

Data Bus:

- Data bus consists of 8, 16, 32 parallel bidirectional signal lines
- Many devices in the system will have their output connected to data bus, but only one device at a time will have its output enabled

Control Bus:

- The control bus consists of 4 to 10 parallel signal lines
- The CPU sends out signals on the control bus to enable the outputs of addressed memory devices or port devices
- Example of control signals: Memory read, Memory write


CPU
"Processing"
"Input"
1 0 1 1 0 0 0 1
1 1 0 0 1 1 1 0
"Output"
A

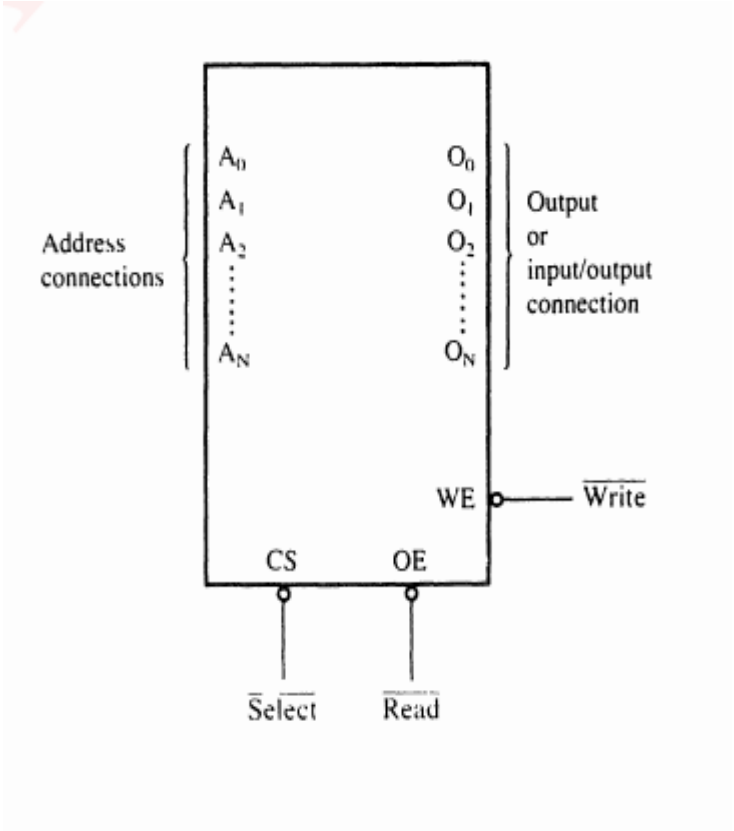
Memory System

- Whether simple or complex, every microprocessor-based system has a memory system.
- Two main types of memory: read-only memory (ROM) and random access memory (RAM).

Memory Pin Connections

- Pin connections common to all memory devices are the
  - address inputs,
  - data outputs or input/outputs,
  - some type of selection input, and
  - at least one control input used to select a read or write operation.

Memory Pin Connections



Address Connections

- Address inputs that select a memory location within the memory device. Address inputs are almost always labeled from A_0, the least significant address input, to A_n

- A 1K memory device has 10 address pins (A0-A9)

Data Connections

- The data connections are the points at which data are entered for storage or extracted for reading.
- A 16K x 1 is a memory device containing 16K 1-bit memory locations.
- 64K x 4 memory device is listed as a 256K device.

Selection Connections

- Each memory device has an input, sometimes more than one, that selects or enables the memory device.
- This kind of input is most often called a chip select (CS), chip enable (CE), or simply select (S) input.

Control Connections

- The control input most often found on a ROM is the output enable (OE) or gate (G) connection, which allows data to flow out of the output data pins of the ROM. If OE and the selection input are both active, then the output is enabled; if OE is inactive, the output is disabled at its high-impedance state.

Control Connections

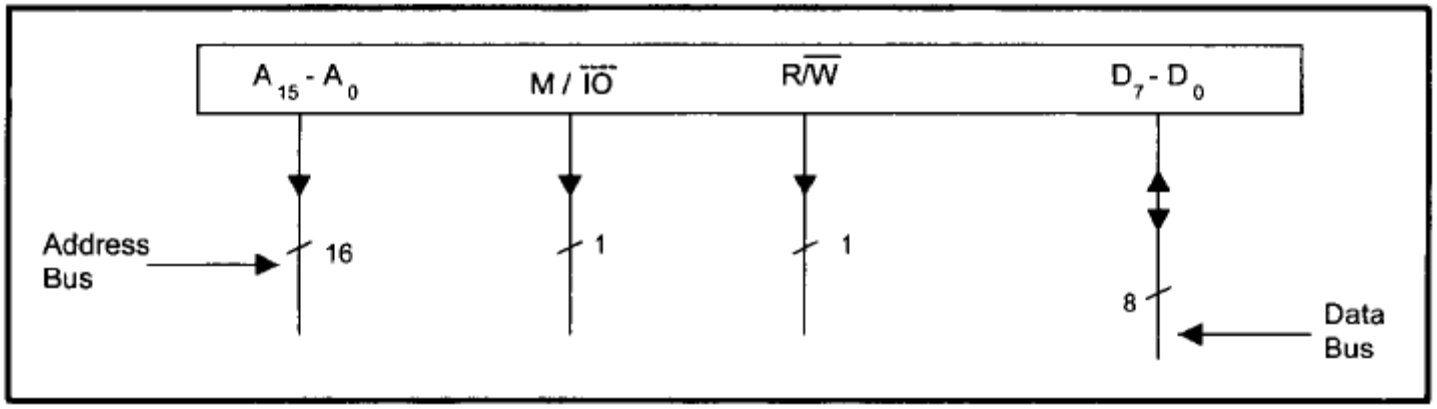
- A RAM memory device has either one or two control inputs.
- If there is one control input, it is often called R/W. This pin selects a read operation or a write operation only if the device is selected by the selection input (CS).
- If the RAM has two control inputs, they are usually labeled WE (or W) and OE (or G). Here, WE (write enable) must be active to perform a memory write operation, and OE must be active to perform a memory read operation. When these two controls (WE) and OE) are present, they must never both be active at the same time.

ADDRESS DECODING

- In order to attach a memory device to the microprocessor, it is necessary to decode the address from the microprocessor to make the memory function at a unique section or partition of the memory map
- Without an address decoder, only one memory device can be connected to a microprocessor, which would make it virtually useless.

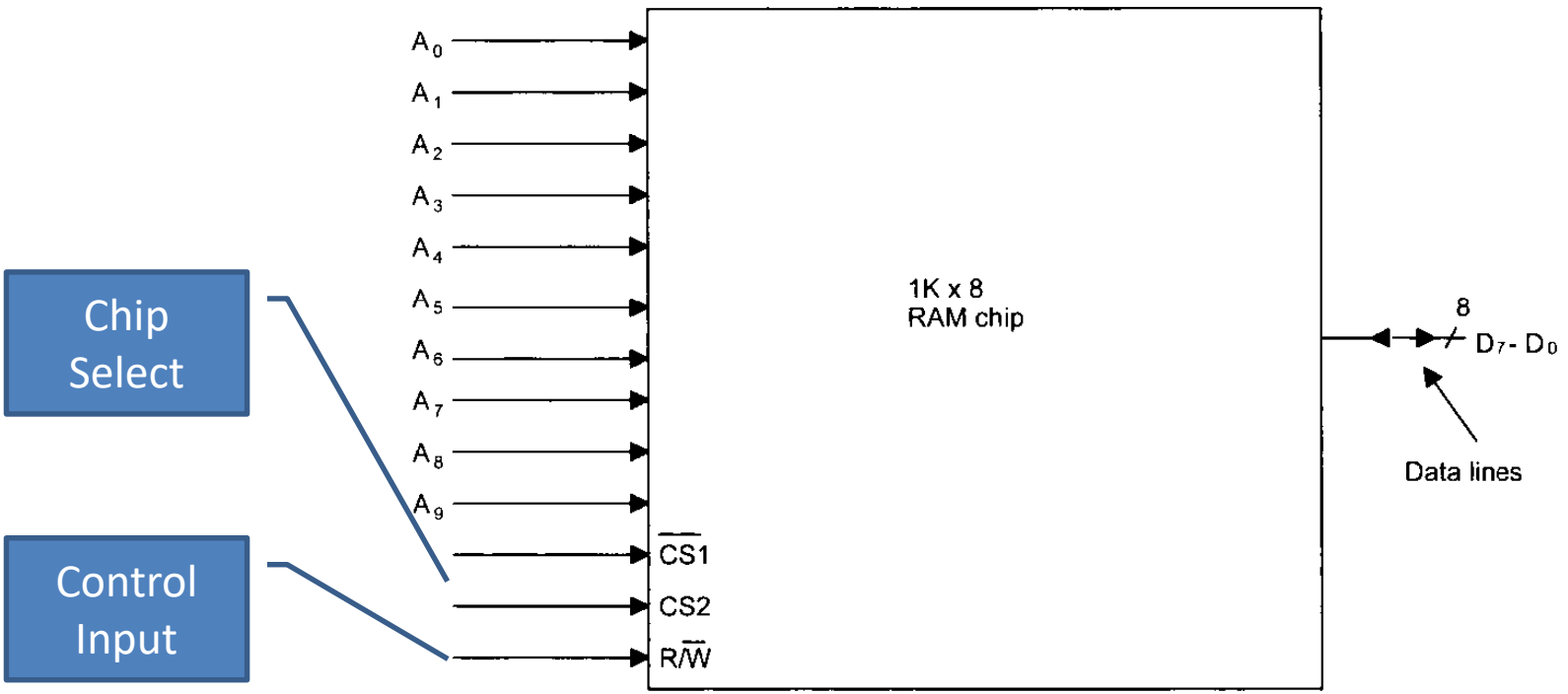
Main Memory Array Design

- **Memory Array Design** means: Interconnecting several memory chips.
- A 16 bit microprocessor can address directly a maximum of 2^{16} = 65,536 or 64K bytes of memory locations.
- M /$\overline{IO}$ › LOW : if the microprocessor executes an I/O instruction
- M /$\overline{IO}$ › HIGH: if the microprocessor executes a memory instruction.



Pertinent signals of a typical microprocessor required for main memory interfacing.

Main Memory Array Design



Typical 1K × 8 SRAM chip.

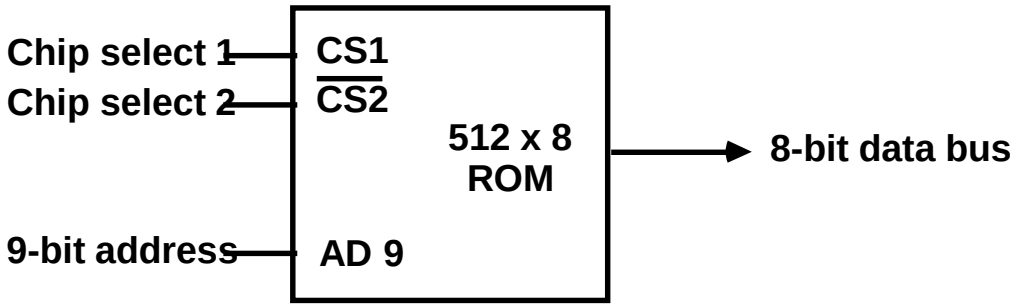
TABLE Truth Table for Controlling SRAM.

$\overline{CS1}$	CS2	$R/\overline{W}$	Function
0	1	0	Write Operation
0	1	1	Read Operation
1	X	X	The chip is not selected
X	0	X	The chip is not selected

Main Memory Array Design

- To connect a microprocessor to ROM/RAM chips, three address-decoding techniques are commonly used:
  - **Linear decoding**
  - **Full/Partial decoding**
  - **Memory decoding using PALs**

Typical ROM chip



Main Memory Array Design

- **Linear decoding**

Suppose we have

4K SRAM chip array comprised of the four 1K SRAM chips

- *This system* can be expanded up to a total capacity of 6K using A_{14}, and A_{15}, as chip selects for two more 1K SRAM chips.

Address Range (Hex)	SRAM Chip Number
3800-3BFF	I
3400-37FF	II
2C00-2FFF	III
1C00-1FFF	IV

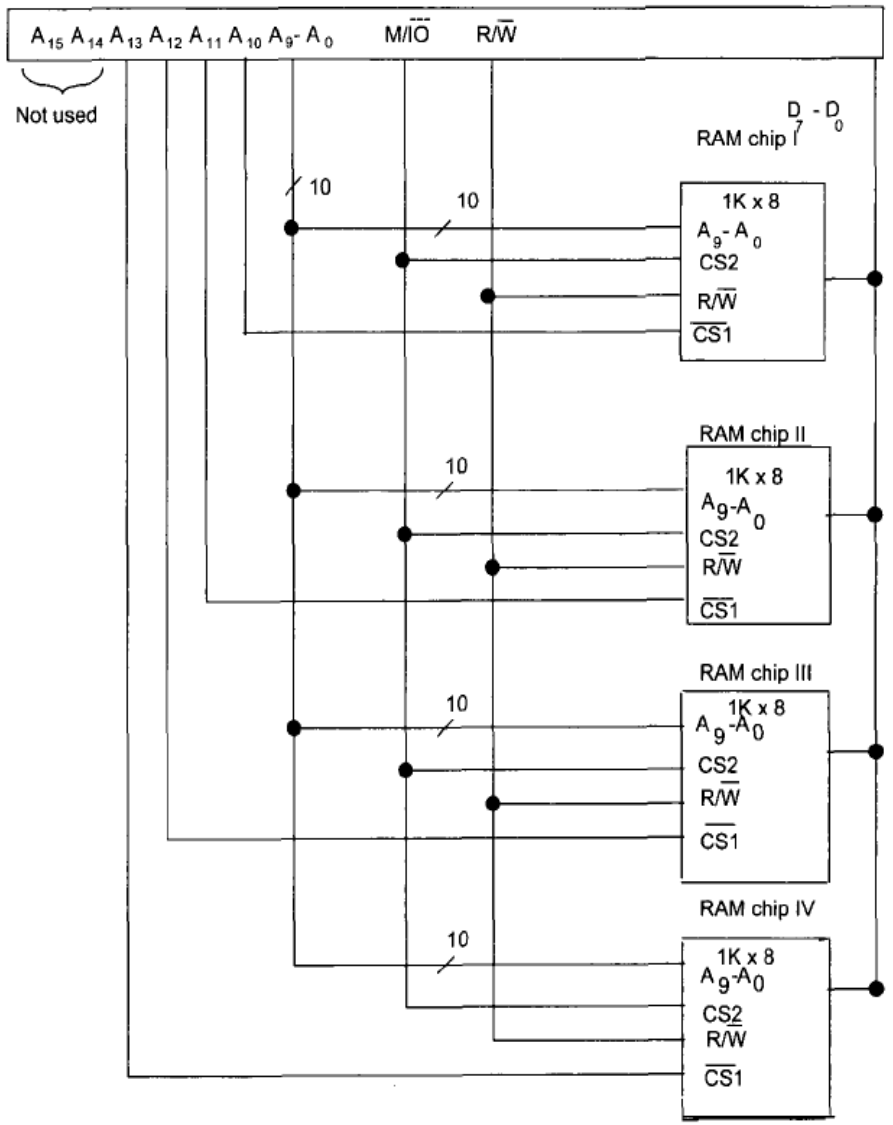


FIGURE 3.8 **Microprocessor connected to 4K SRAM using the linear select decoding technique.**

Binary Address Pattern A15 A14 A13 A12 A11 A10 A9 A7 A6 A5 A4 A3 A2 A1 A0	Device Selected	Address Assignment in Hex
0 0 0 0 0 1 0 0 0 0 0 0 0 0 0	RAM CHIP 0	0400 to 07FF
. . . . .		
0 0 0 0 0 1 1 1 1 1 1 1 1 1 1		
0 0 0 0 1 0 0 0 0 0 0 0 0 0 0	RAM CHIP 1	0800 to 0BFF
. . . . .		
0 0 0 0 1 0 1 1 1 1 1 1 1 1 1		
0 0 0 1 0 0 0 0 0 0 0 0 0 0 0	RAM CHIP 2	1000 to 13FF
. . . . .		
0 0 0 1 0 0 1 1 1 1 1 1 1 1 1		
0 0 1 0 0 0 0 0 0 0 0 0 0 0 0	RAM CHIP 3	2000 to 23FF
. . . . .		
0 0 1 0 0 0 1 1 1 1 1 1 1 1 1		

Main Memory Array Design

- **Linear decoding Advantage**

does not require decoding hardware.

- **Linear decoding Disadvantage**

1. Bus Conflict: two or more of lines A_{10}-$A_{13}$are low at the same time, more than one SRAM chip are selected.

Solution: software must be written such that it never reads into or writes from any address in which more than one of bits A_{10}-$A_{13}$are low.

2. Memory Foldback : Wastes a large amount of address space. For example, whenever the address value is B800 or 3800, SRAM chip I is selected.

Solution: use full decoded memory addressing.

Main Memory Array Design

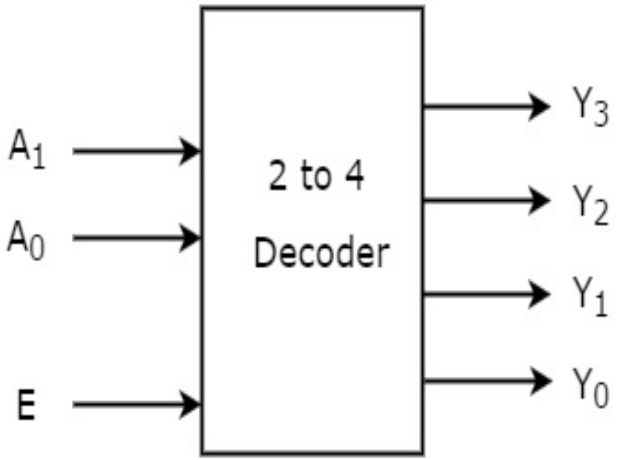
- **If only a portion of the addressable space is going to be implemented there are two basic address decoding strategies**
  - Full address decoding
    - All the address lines are used to specify a memory location
    - Each physical memory location is identified by a unique address
  - Partial address decoding
    - Since not all the address space is implemented, only a subset of the address lines are needed to point to the physical memory locations
    - Each physical memory location is identified by several possible addresses (using all combinations of the address lines that were not used)

Decoder

- **Decoder** is a combinational circuit that has 'n' input lines and maximum of 2^n output lines. One of these outputs will be active High based on the combination of inputs present, when the decoder is enabled. That means decoder detects a particular code. The outputs of the decoder are nothing but the **min terms** of 'n' input variables lines, when it is enabled.

Decoder

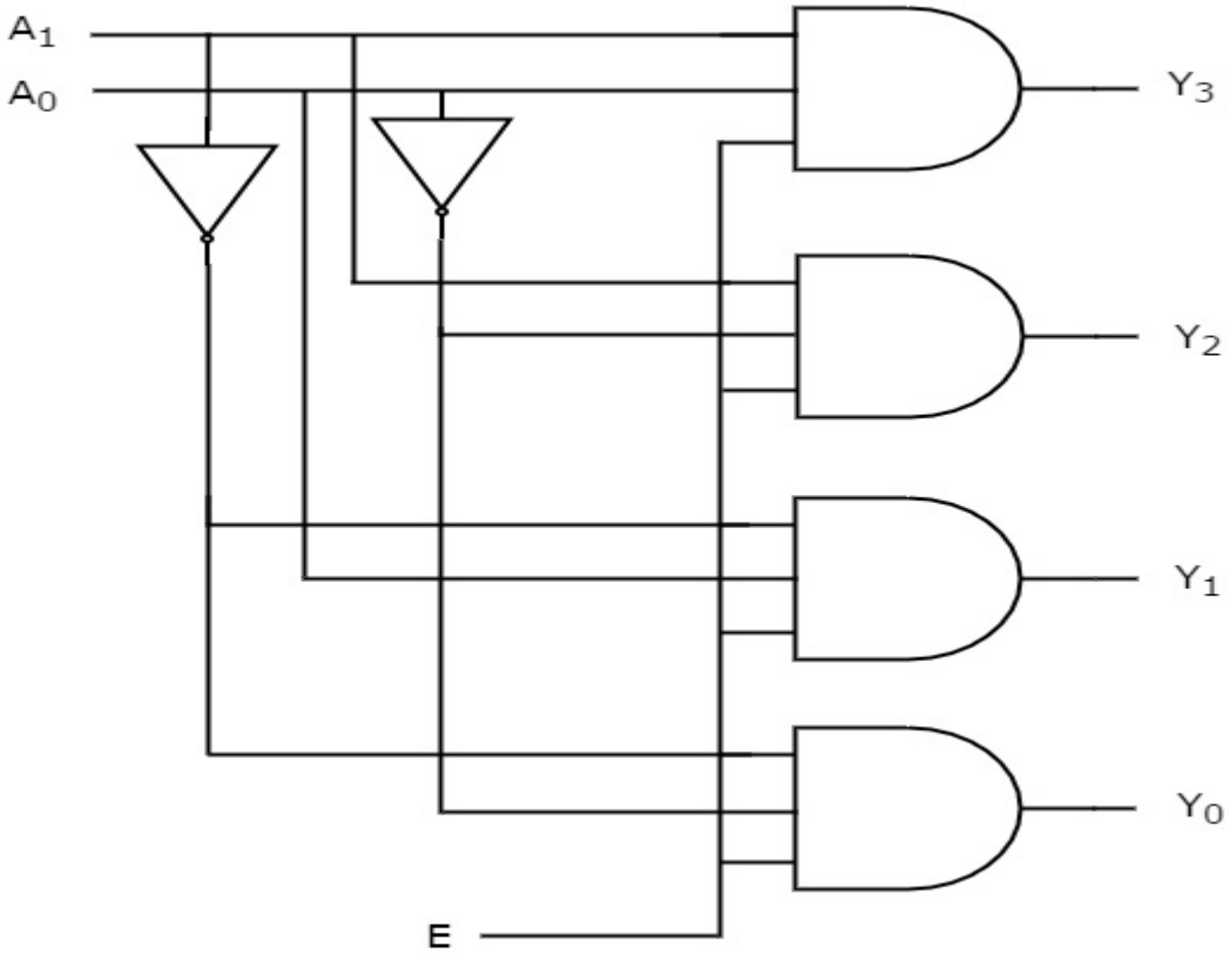
- **2 to 4 Decoder**
- Let 2 to 4 Decoder has two inputs A_1 & A_0 and four outputs Y_3, Y_2, Y_1 & Y_0.



Enable	Inputs		Outputs			
E	A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	x	x	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

- One of these four outputs will be “1”, for each combination of inputs, when Enable, E is “1”.

Decoder

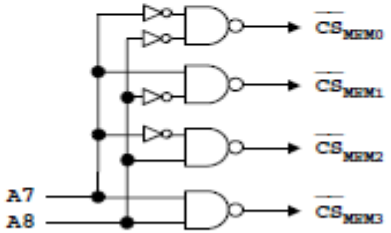


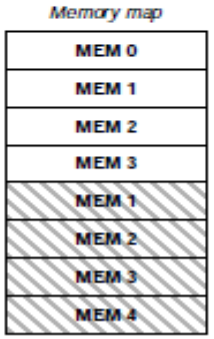
Main Memory Array Design

- **Partial address decoding**
- **Let's assume a microprocessor with 10 address lines (1KB memory)**
  - However, this time we wish to implement only 512 bytes of memory
  - We still must use 128-byte memory chips
  - Physical memory must be placed on the upper half of the memory map
- **SOLUTION**

Not used | *Used for Address Decoding* | *Used to reference memory cells on each memory IC*

Device	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
MEM 0	X	0	0	X	X	X	X	X	X	X
MEM 1	X	0	1	X	X	X	X	X	X	X
MEM 2	X	1	0	X	X	X	X	X	X	X
MEM 3	X	1	1	X	X	X	X	X	X	X



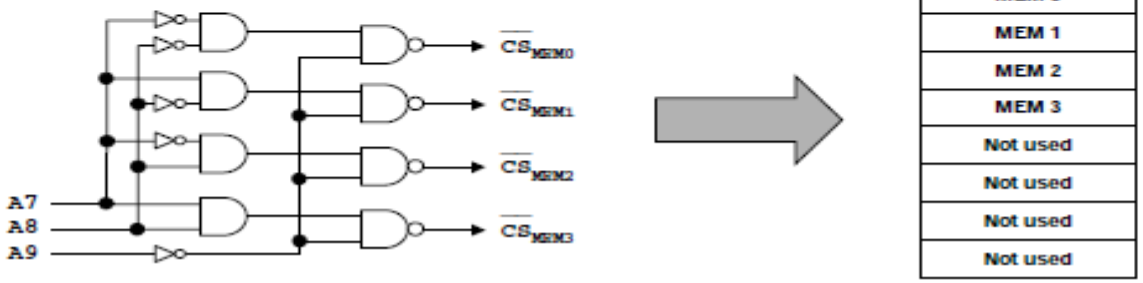


Main Memory Array Design

- **Full address decoding**
- **Let's assume the same microprocessor with 10 address lines (1KB memory)**
  - However, this time we wish to implement only 512 bytes of memory
  - We still must use 128-byte memory chips
  - Physical memory must be placed on the upper half of the memory map
- **SOLUTION**

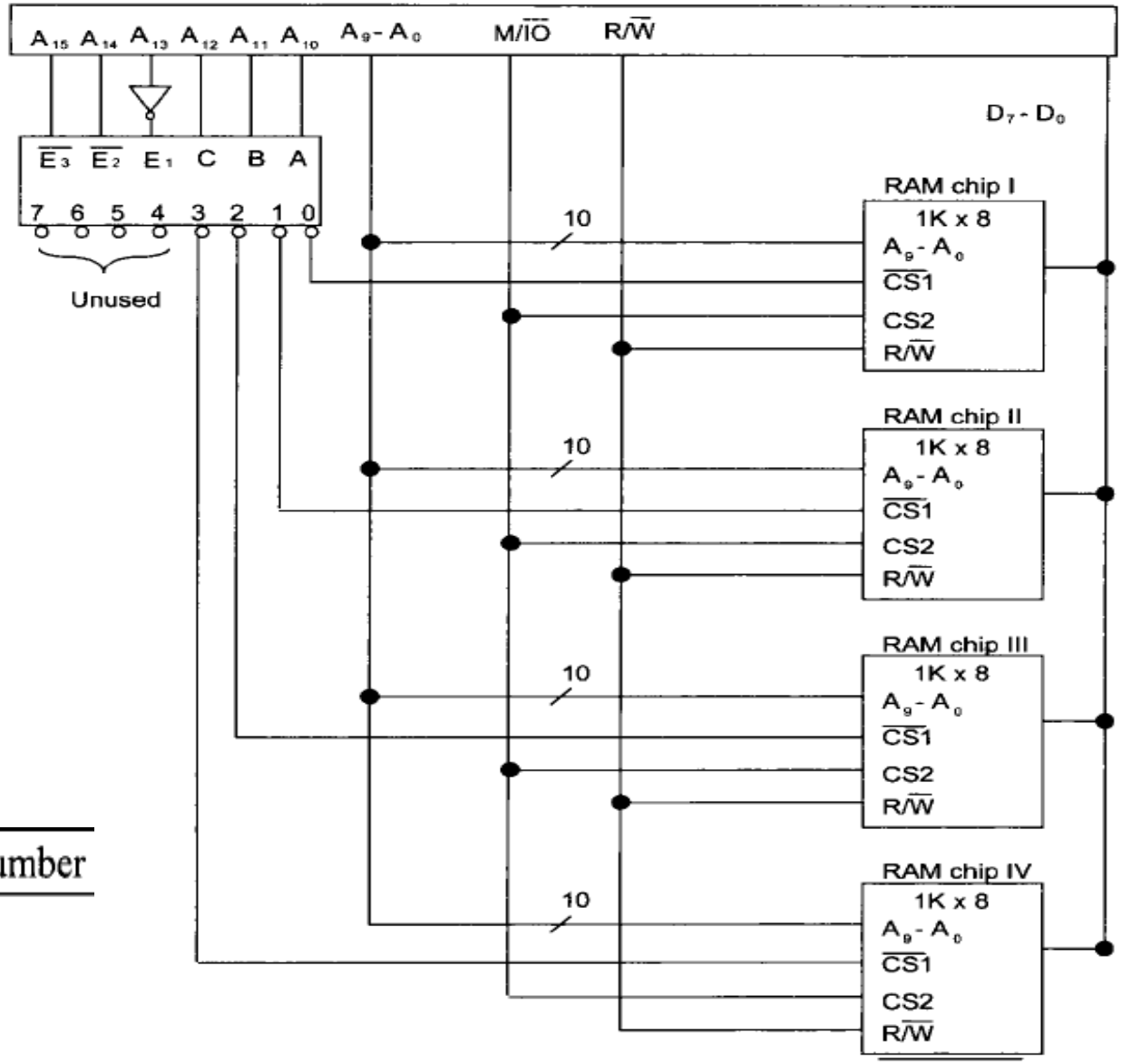
Used for Address Decoding | *Used to reference memory cells on each memory IC*

Device	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
MEM 0	0	0	0	X	X	X	X	X	X	X
MEM 1	0	0	1	X	X	X	X	X	X	X
MEM 2	0	1	0	X	X	X	X	X	X	X
MEM 3	0	1	1	X	X	X	X	X	X	X



Main Memory Array Design

- **Full Decoding.**
- Using 3x8 decoder output selects one of the four IK SRAM chips, depending on the values of A_{12}, A_{11}, and A_{10}
- Decoder output enabled only when $\overline{E3} = \overline{E2} = 0$ and E1 = 1



Interconnecting a microprocessor with a 4K RAM using full decoded memory addressing.

A_{12}	A_{11}	A_{10}	SRAM Chip Number
0	0	0	I
0	0	1	II
0	1	0	III
0	1	1	IV

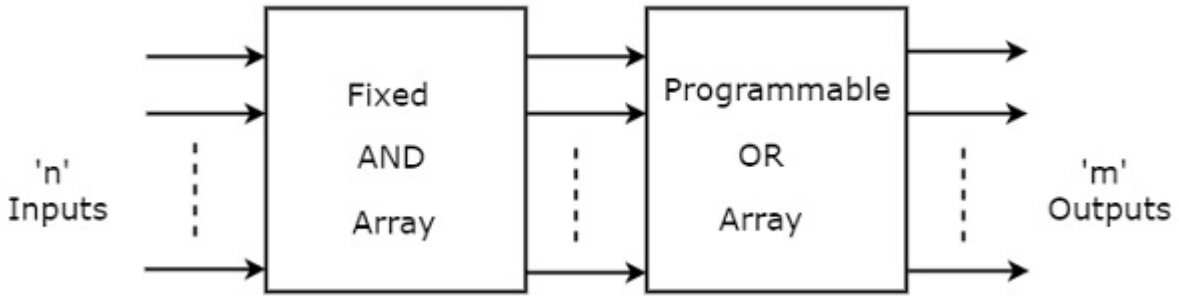
Binary Address Pattern A15 A14 A13 A12 A11 A10 A9 A7 A6 A5 A4 A3 A2 A1 A0	Device Selected	Address Assignment in Hex
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 . . . . . . 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1	RAM CHIP 0	0000 to 03FF
0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 . . . . . . 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1	RAM CHIP 1	0400 to 07FF
0 0 1 0 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	RAM CHIP 2	0800 to 0BFF
0 0 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	RAM CHIP 3	0C00 to 0FFF

Main Memory Array Design

- Programmable Logic Devices (PLDs) are the integrated circuits. They contain an array of AND gates & another array of OR gates. There are three kinds of PLDs based on the type of arrays, which has programmable feature.
  - Programmable Read Only Memory
  - Programmable Array Logic
  - Programmable Logic Array
- The process of entering the information into these devices is known as **programming**. Basically, users can program these devices or ICs electrically in order to implement the Boolean functions based on the requirement. Here, the term programming refers to **hardware programming but not software programming**.

Main Memory Array Design

- Programmable Read Only Memory (PROM)
  - Read Only Memory (ROM) is a memory device, which stores the binary information permanently. That means, we can't change that stored information by any means later. If the ROM has programmable feature, then it is called as **Programmable ROM (**PROM). The user has the flexibility to program the binary information electrically once by using PROM programmer.
  - PROM is a programmable logic device that has fixed AND array & Programmable OR array. The **block diagram** of PROM is shown in the following figure.



Main Memory Array Design

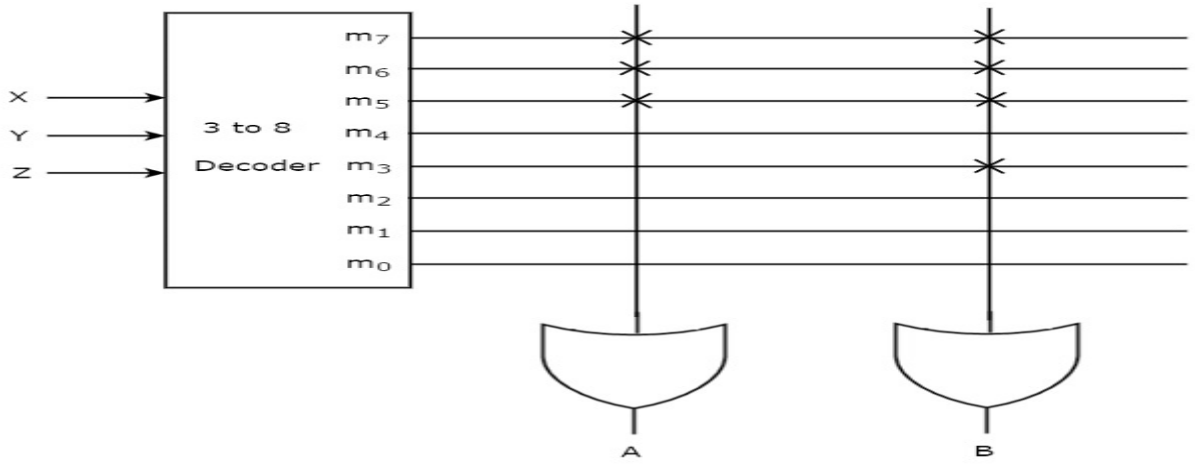
- Here, the inputs of AND gates are not of programmable type. So, we have to generate 2^n product terms by using 2^n AND gates having n inputs each. We can implement these product terms by using $nx2^n$ decoder. So, this decoder generates 'n' **min terms**.
- Here, the inputs of OR gates are programmable. That means, we can program any number of required product terms, since all the outputs of AND gates are applied as inputs to each OR gate. Therefore, the outputs of PROM will be in the form of **sum of min terms**.
- Example
- Let us implement the following **Boolean functions** using PROM.

$$A(X,Y,Z) = \sum m\,(5,6,7)$$

$$B(X,Y,Z) = \sum m\,(3,5,6,7)$$

Main Memory Array Design

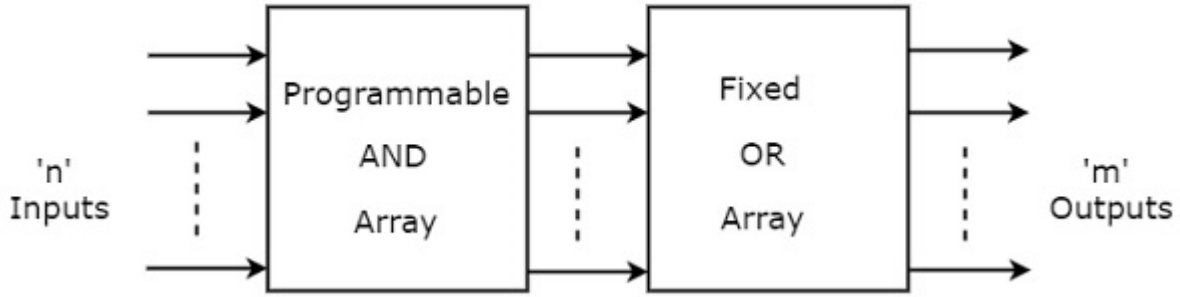
- The given two functions are in sum of min terms form and each function is having three variables X, Y & Z. So, we require a 3 to 8 decoder and two programmable OR gates for producing these two functions. The corresponding **PROM** is shown in the following figure.



- Here, 3 to 8 decoder generates eight min terms. The two programmable OR gates have the access of all these min terms. But, only the required min terms are programmed in order to produce the respective Boolean functions by each OR gate. The symbol ‘X’ is used for programmable connections.

Main Memory Array Design

- Programmable Array Logic (PAL)
  - PAL is a programmable logic device that has Programmable AND array & fixed OR array. The advantage of PAL is that we can generate only the required product terms of Boolean function instead of generating all the min terms by using programmable AND gates. The **block diagram** of PAL is shown in the following figure.



Main Memory Array Design

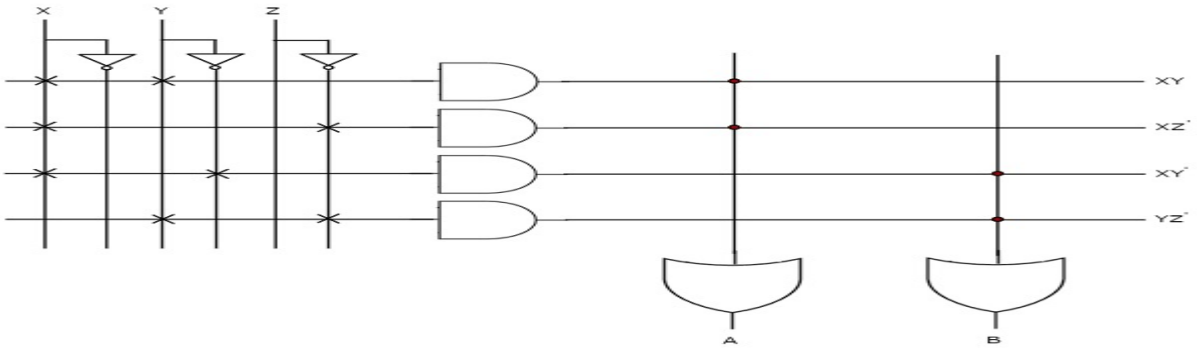
- Here, the inputs of AND gates are programmable. That means each AND gate has both normal and complemented inputs of variables. So, based on the requirement, we can program any of those inputs. So, we can generate only the required **product terms** by using these AND gates.
- Here, the inputs of OR gates are not of programmable type. So, the number of inputs to each OR gate will be of fixed type. Hence, apply those required product terms to each OR gate as inputs. Therefore, the outputs of PAL will be in the form of **sum of products form**.
- Example
- Let us implement the following **Boolean functions** using PAL.

$$A = XY + XZ'$$

$$A = XY' + YZ'$$

Main Memory Array Design

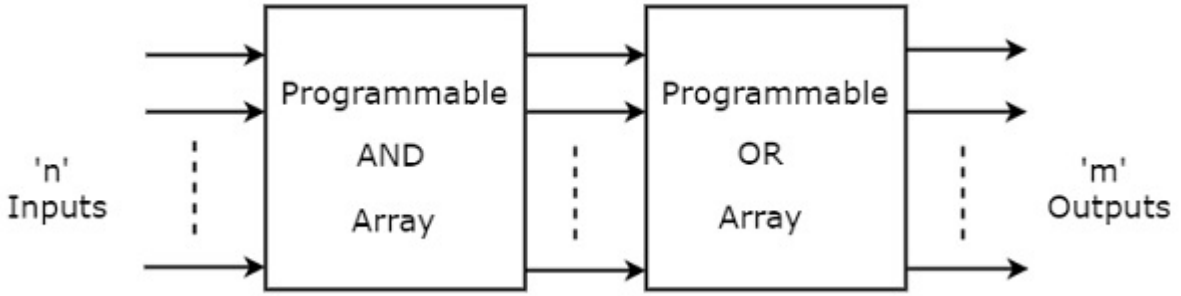
- The given two functions are in sum of products form. There are two product terms present in each Boolean function. So, we require four programmable AND gates & two fixed OR gates for producing those two functions. The corresponding **PAL** is shown in the following figure.



- The **programmable AND gates** have the access of both normal and complemented inputs of variables. In the above figure, the inputs X, *X*'X', Y, *Y*'Y', Z & *Z*'Z', are available at the inputs of each AND gate. So, program only the required literals in order to generate one product term by each AND gate. The symbol 'X' is used for programmable connections.
- Here, the inputs of OR gates are of fixed type. So, the necessary product terms are connected to inputs of each **OR gate**. So that the OR gates produce the respective Boolean functions. The symbol '.' is used for fixed connections.

Main Memory Array Design

- Programmable Logic Array (PLA)
  - PLA is a programmable logic device that has both Programmable AND array & Programmable OR array. Hence, it is the most flexible PLD. The **block diagram** of PLA is shown in the following figure.



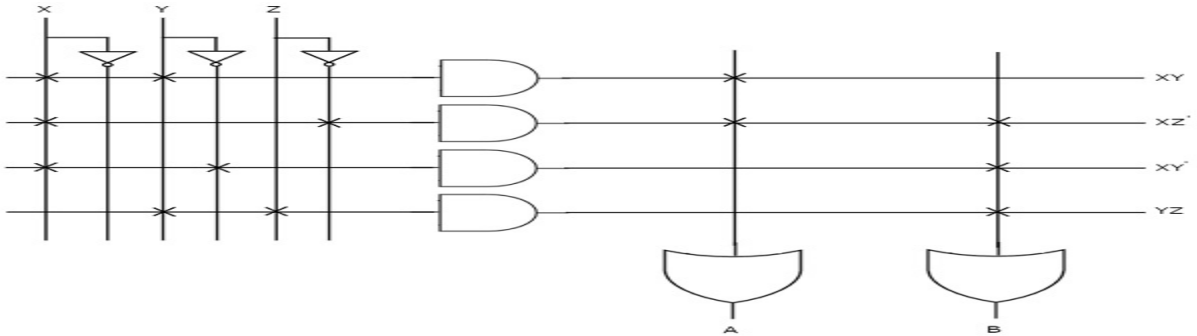
  - Here, the inputs of AND gates are programmable. That means each AND gate has both normal and complemented inputs of variables. So, based on the requirement, we can program any of those inputs. So, we can generate only the required **product terms** by using these AND gates.
  - Here, the inputs of OR gates are also programmable. So, we can program any number of required product terms, since all the outputs of AND gates are applied as inputs to each OR gate. Therefore, the outputs of PAL will be in the form of **sum of products form**.
  - Example
  - Let us implement the following **Boolean functions** using PLA.

$$A = XY + XZ'$$

$$B = XY' + YZ + XZ'$$

Main Memory Array Design

- The given two functions are in sum of products form. The number of product terms present in the given Boolean functions A & B are two and three respectively. One product term, *Z*'*X*Z'X is common in each function.
- So, we require four programmable AND gates & two programmable OR gates for producing those two functions. The corresponding **PLA** is shown in the following figure.



- The **programmable AND gates** have the access of both normal and complemented inputs of variables. In the above figure, the inputs X, *X*'X', Y, *Y*'Y', Z & *Z*'Z', are available at the inputs of each AND gate. So, program only the required literals in order to generate one product term by each AND gate.
- All these product terms are available at the inputs of each **programmable OR gate**. But, only program the required product terms in order to produce the respective Boolean functions by each OR gate. The symbol 'X' is used for programmable connections.

Combinational PLDs

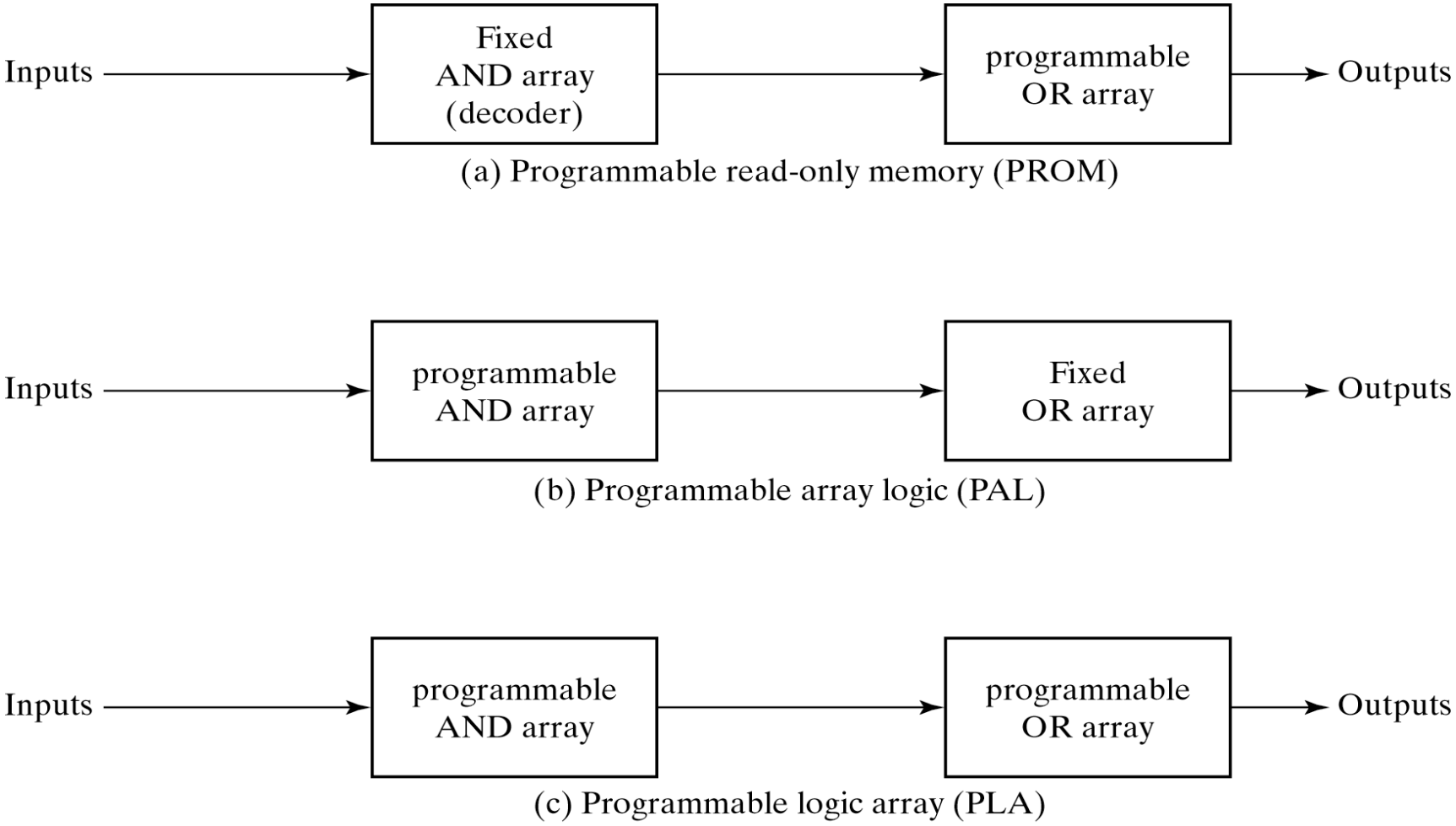


Fig. 7-13 Basic Configuration of Three PLDs

PAL

- PAL chips are usually identified by a two-digit number followed by a letter and then a digit.
  - Two-digit number specifies the number of input lines
  - Last digit defines the number of output lines
  - The fixed number of AND gates are connected to either an OR or a NOR gate
  - Letter H indicates that the output gates are OR gates
  - Letter L indicates that the output gates are NOR gates
- Example: 10H8, 10L8

Example: 01

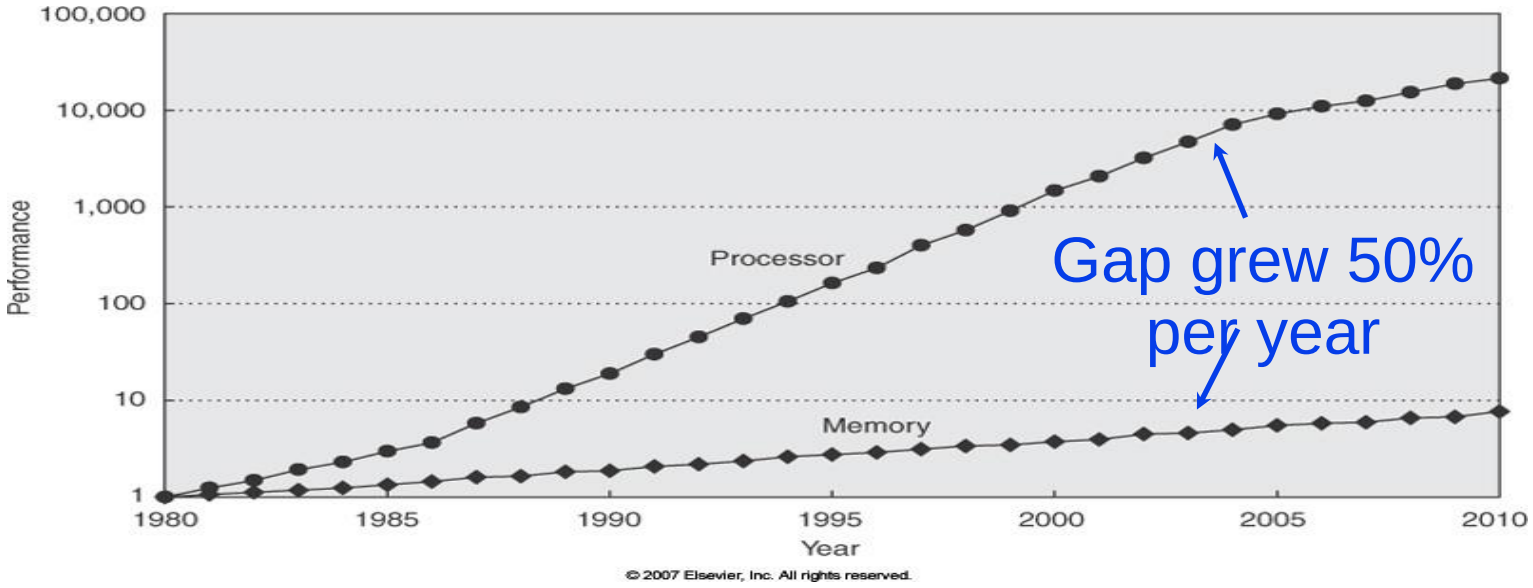
- A processor having 16 bits address line and it is byte addressable. Interface 4096x8 RAM with the processor.
  1. What is the memory capacity of the processor?
  2. How many addresses are required for a RAM?/
  3. How many RAM chips are required?
  4. What will be the size of address decoder?

Example: 02

- Interface two chips of 4k RAM and 1 chip of 2k ROM. Starting address of RAM is 8000H.

Memory Management Concepts

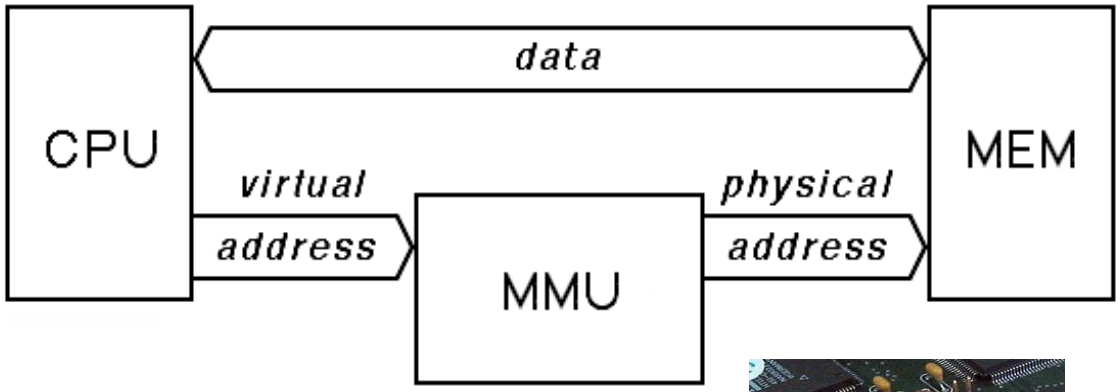
- Access to a hard disk is Slow.
- **Solution**: use a large and fast locally accessed semiconductor memory SRAM.
- Unfortunately, the storage cost per bit for this solution is very high.
- **A combination of both off-board disk (secondary memory) and on-board** semiconductor main memory must be designed into a system.

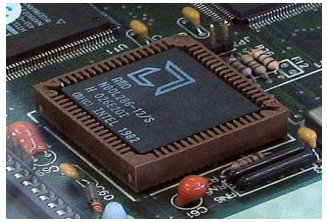


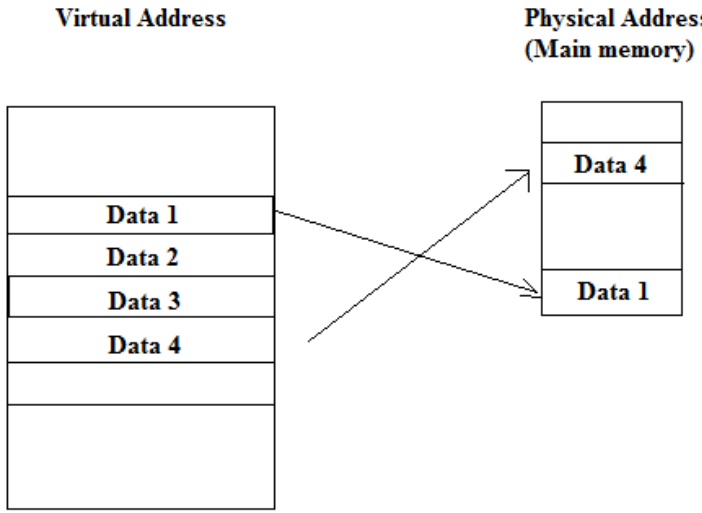
Memory Management Concepts

Memory Management Unit (MMU):

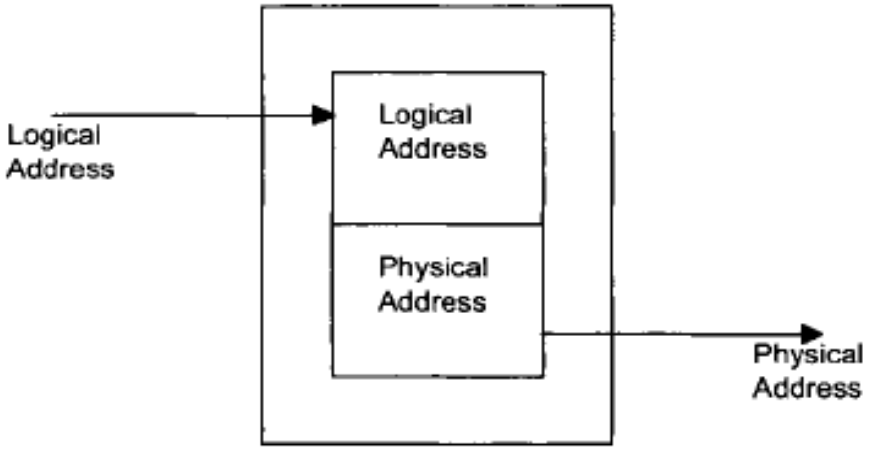
- A device, located between the microprocessor and memory
- The address used by a programmer will be called a **logical address/ Virtual address**
- An address in main memory is called a **physical address**
- Control accesses, perform address mappings, and act as an interface between the logical (programmer's memory) and physical memory



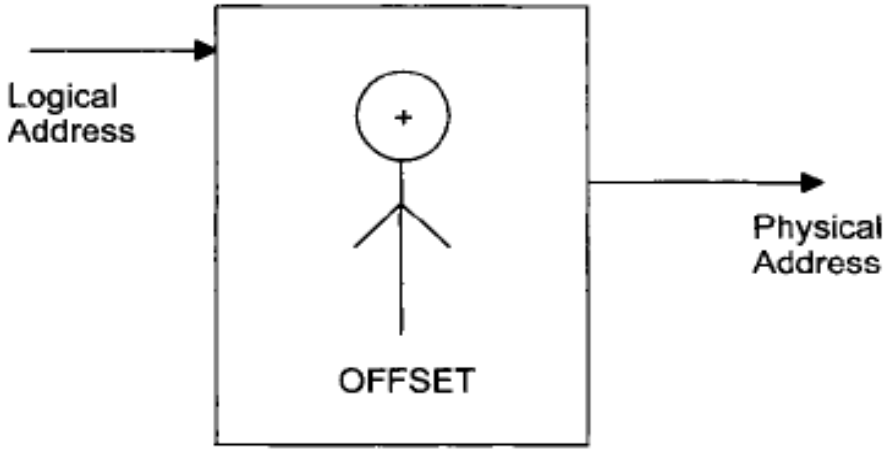




Memory Management Concepts



(a) using the substitution technique



(b) Using the offset technique

FIGURE 3.10 Address translation

MMU address translation:

- The MMU can perform address translation in one of two ways:
  1. **Substitution technique**
  2. **Addition of an offset to each logical address to obtain the corresponding physical address**

Memory Management Concepts

MMU address translation:

- Substitution technique is faster
- Offset method has the advantage of mapping a logical address to any physical address as determined by the offset value.

Memory Management Concepts

MMU address translation: Memory is usually divided into small manageable units:

- ***Page***
- ***Segment***.
- ***Combined Paging-Segmentation***

- ***Paging*** *divides the memory into* ***equal sized pages***
- **Segmentation** divides the memory into **variable-sized segments**.
- It is relatively easier to implement the address translation table if the logical and main memory spaces are divided into pages.

Memory Management Concepts

The Paging method

- The virtual memory system is managed by both hardware and software.
  - The **hardware** included in **MMU** handles address translation
  - **The memory management software** in **operating system** performs functions such as
    - removal of the desired page from main memory to accommodate a new page
    - Transferring a new page from secondary to main memory at the right instant of time
    - Placing the page at the right location in memory
  - If the main memory is full during transfer from secondary to main memory, it is necessary to remove a page from main memory to accommodate the new page. Two methods:
    - FIFO
    - LRU

Memory Management Concepts

Physical memory partitioned into equal sized *page frames or just pages*

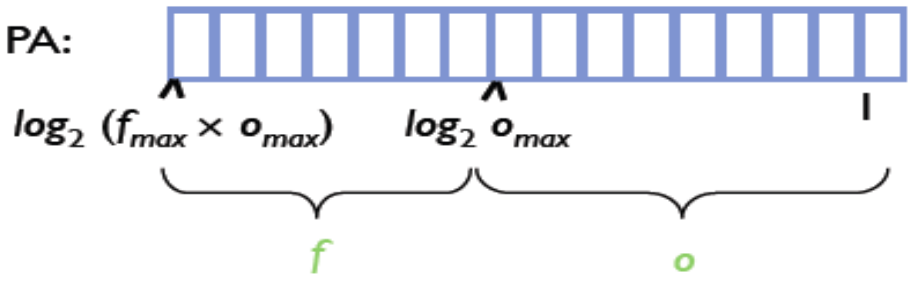
- Page frames avoid external fragmentation.

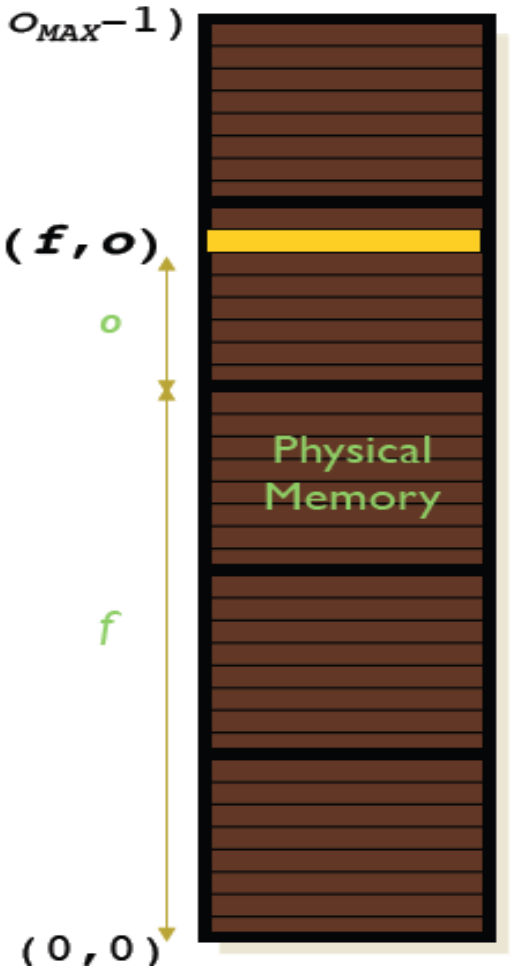
A memory address is a pair (*f*, *o*)

f — frame number (f_{max} frames)

o — frame offset (o_{max} bytes/frames)

Physical address = $o_{max} \times f + o$

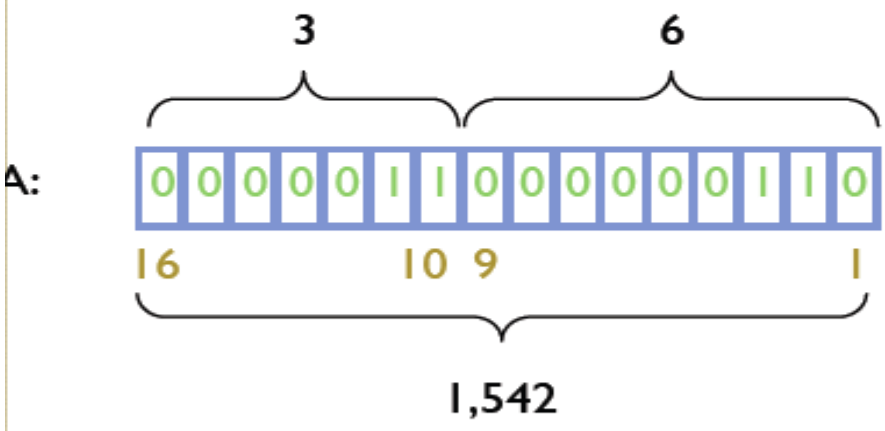


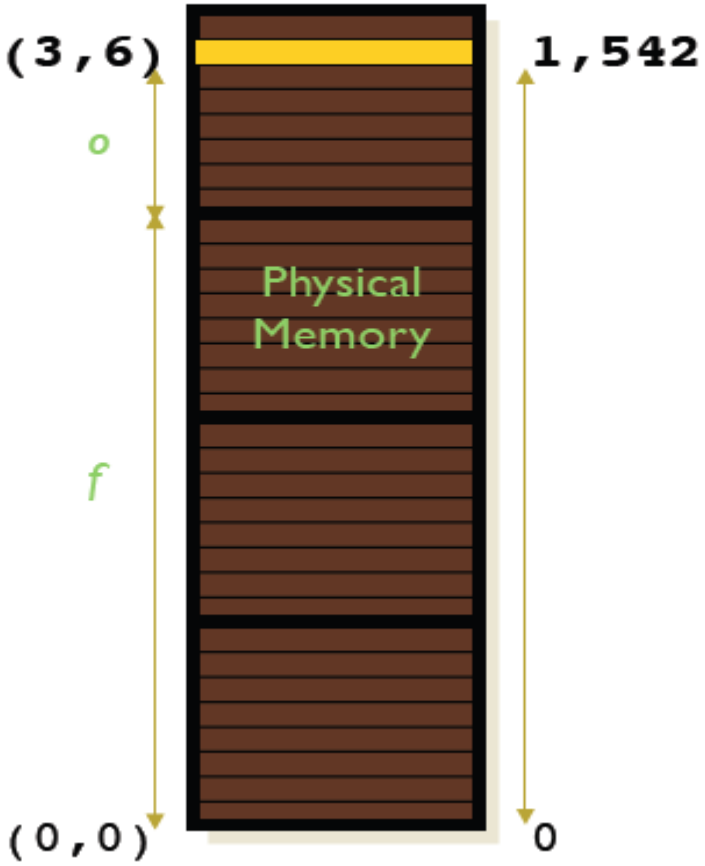


The Paging method

Example: A 16-bit address space with (o_{max} =) 512 byte page frames

- Addressing location (3, 6) = 1,542

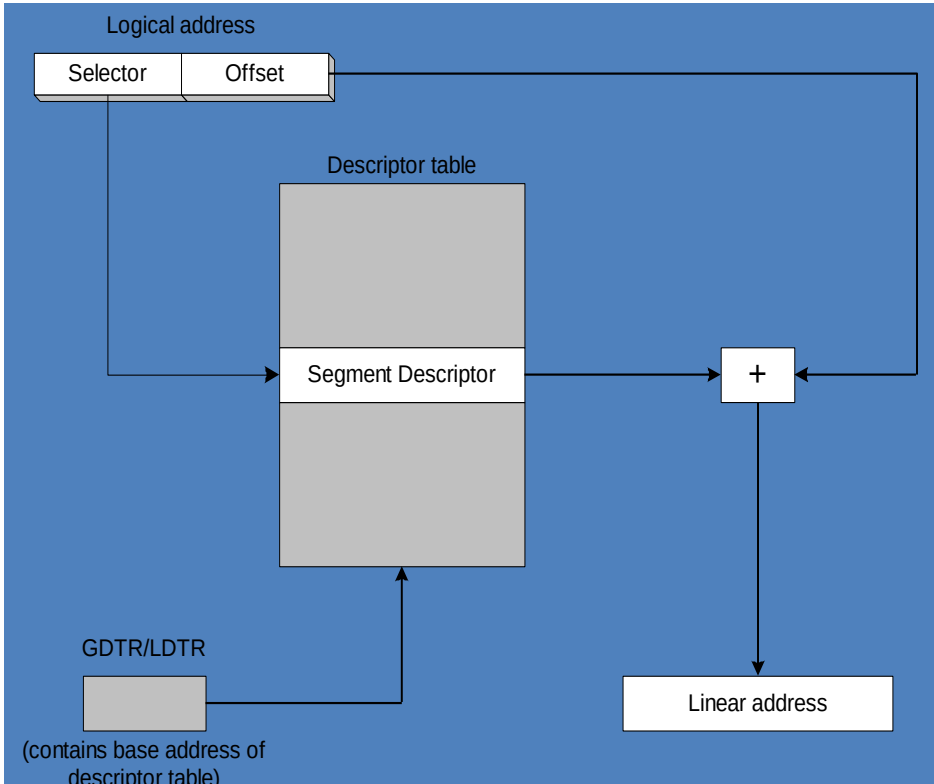


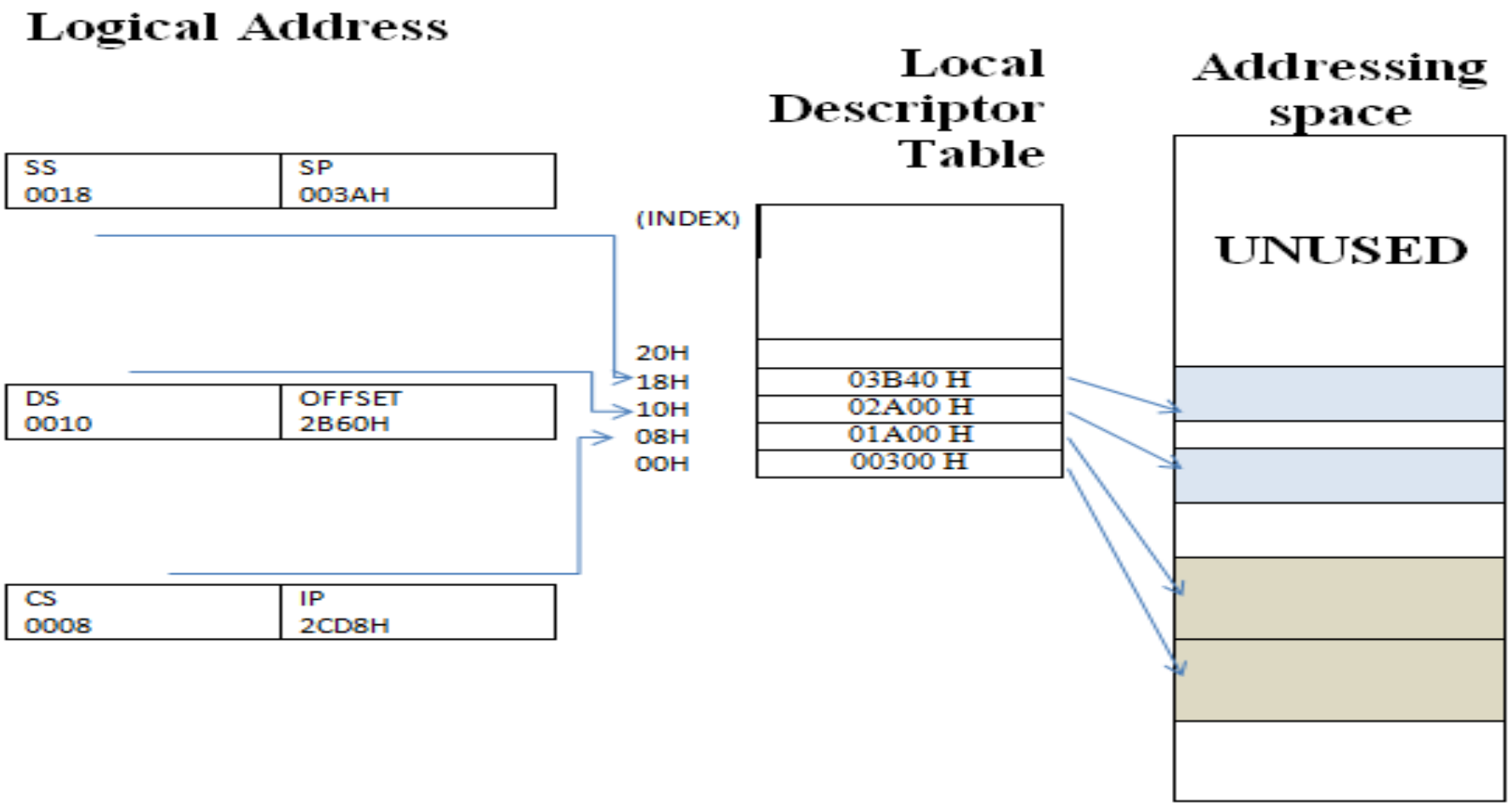


Memory Management Concepts

The Segmentation method

- An **MMU** utilizes the **Segment Selector** to obtain a descriptor from a table in memory containing several descriptors.
  - **Global Descriptor Table (GDT):** Contains
    1. The physical base address for a segment,
    2. The segment's privilege level,
    3. Some control bits.
- Each program has:
  - **Local Descriptor Table (LDT)**
    - That holds descriptor for each segment used by the program





Each segment descriptor indexes into the program's local descriptor table (LDT). Each table entry is mapped to a linear address:

Memory Management Concepts

The Segmentation method

1. When the **MMU** obtains a logical address from the microprocessor
2. **MMU** determines whether the segment is already in physical memory.
3. If it is, the **MMU** adds an offset component to the segment base component of the address obtained from the segment descriptor table to provide the physical address.
4. **MMU** then generates the physical address on the address bus for selecting the memory.

Memory Management Concepts

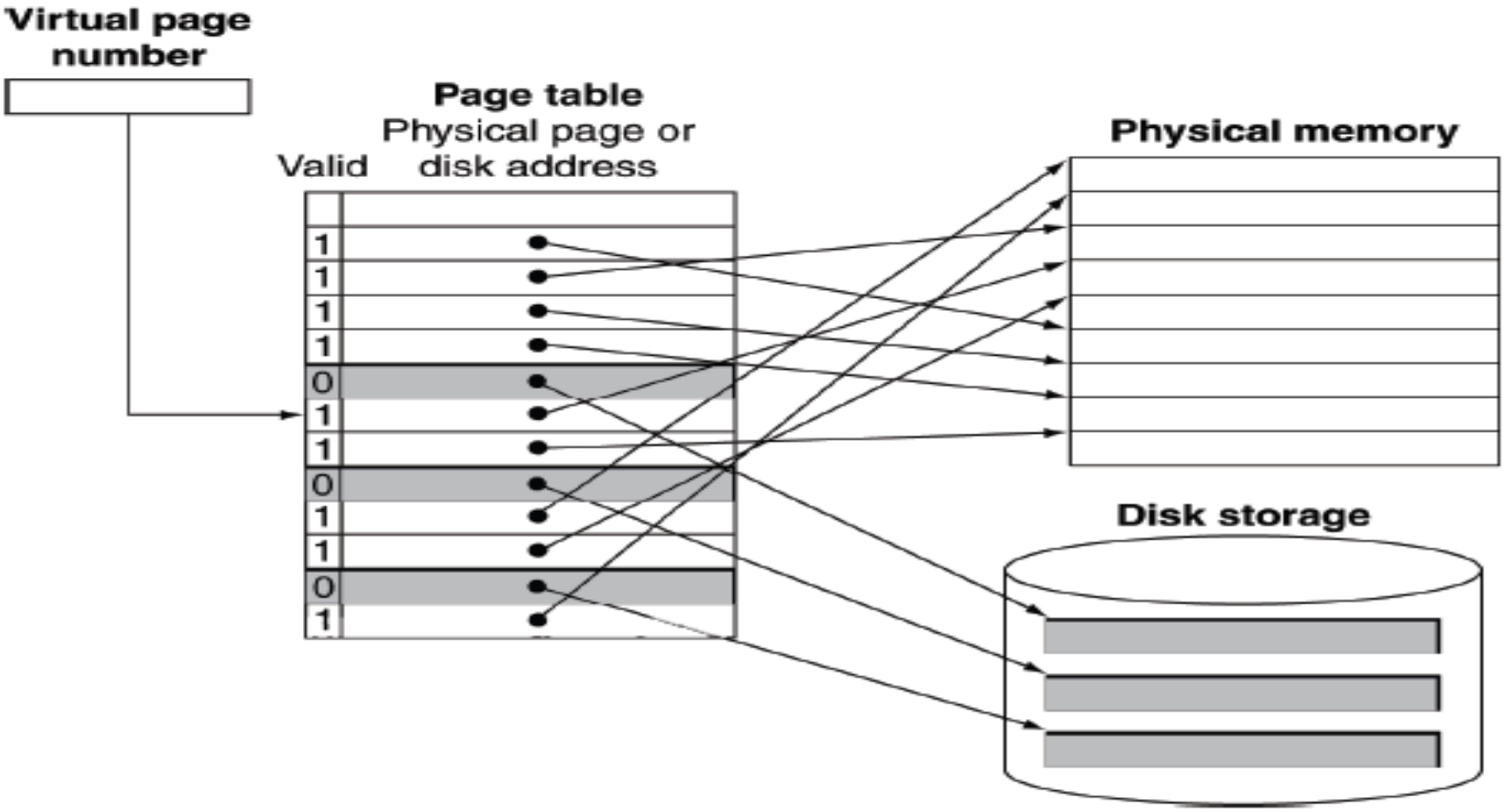
The paged-segmentation method

- Each segment contains a number of pages.
- The logical address is divided into three components: **segment**, page, and **word**.
- A page component of ***n*** *bits can provide up to* ***2^n pages***.
- *A segment can be assigned with one or* more pages up to maximum of ***2^n pages***; therefore, a segment size depends on the number of pages assigned to it.

Memory Management Concepts

- **The Virtual memory:** A technique that uses main memory as a "cache" for secondary storage.
- ***virtual address:*** An address generated by a user program is ***virtual address***
- The key idea behind the **virtual memory** is to allow a user program **to address more locations than those available in a physical memory**.
- Two major motivations for virtual memory:
  - to allow efficient and safe sharing of memory among multiple programs
  - to remove the programming burdens of a small, limited amount of main memory.
- **CPU and OS translate virtual addresses to physical addresses**

Mapping Pages to Storage



➢ Paging with virtual addressing

Programs share main memory, each program gets a private virtual address space holding its frequently used code and data and this is protected from other programs.