EEM332
Introduction to Microprocessor

Review of logic circuits

Combinational logic:

Sequential logic

•Finite state machine (FAM)

•Memory

Microprocessor based system

Instruction cycle

Machine cycle

Digital circuits:

•Combinational circuit

•Sequential Circuits

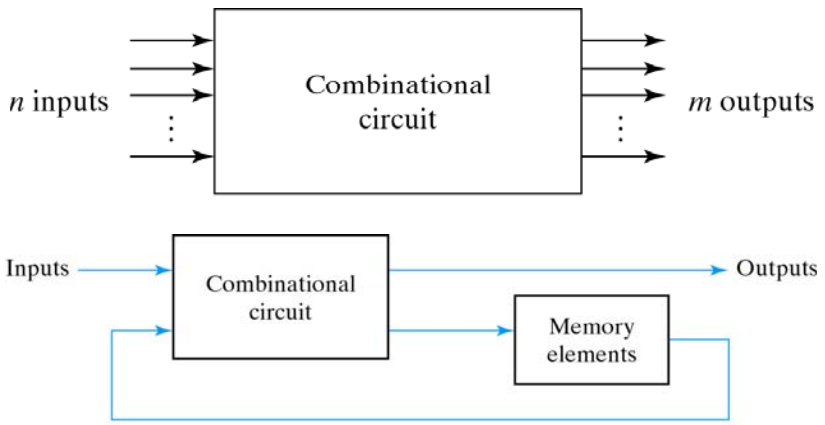


Fig. 5-1 Block Diagram of Sequential Circuit

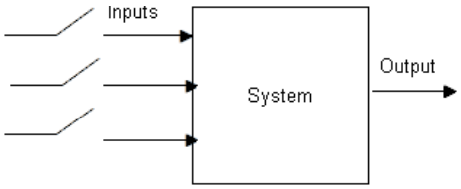
Combinational logic: Example

Bir salonun bir lambası, üç giriş kapısı ve her kapının yanında, lambayı yakıp, söndürebilen bir anahtar vardır. (3 anahtar, bir lamba). Anahtarın ON olması (**1/high**) lambayı yakar (**1/high**), Anahtarın OFF (**0/Low**) olması lambayı söndürür(**0 /Low**) .

Devrenin çalışma şartları:

1)Her 3 anahtar OFF iken lamba yanmaz. **2**) Her 3 anahtar OFF iken herhangi bir anahtar ON olursa Lamba yanar

3)Lamba yanarken ikinci anahtarın ON olması lambayı söndürür. **4**)Her üç anahtar ON iken lamba yanar



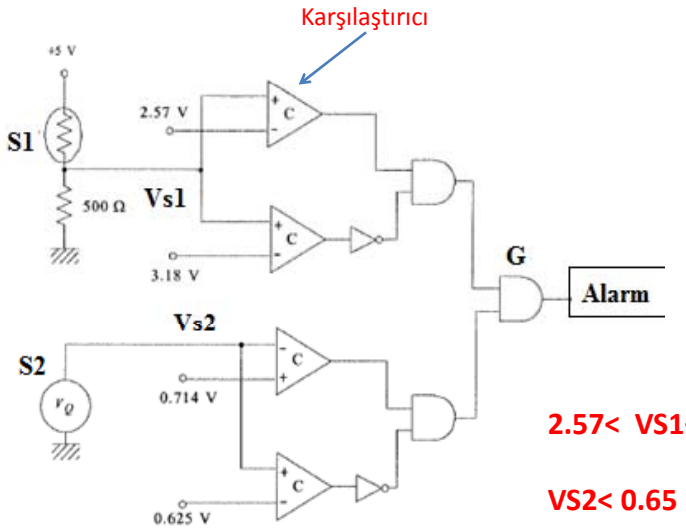
Constant switch

x_1	x_2	x_3	f
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Analog-Digital example:

Verilen devrede **G çıkışına** bağlı olan alarm, **s1 ve s2** algılayıcılardan gelen gerilim değerlerine bağlı olarak aktif olabilir. Alarmın aktif olması için G'nin çıkışı High olmalı.,

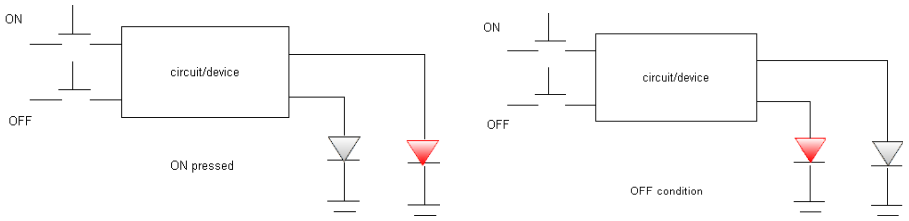
Op-amp gerilimi: 0 ve 5 volt

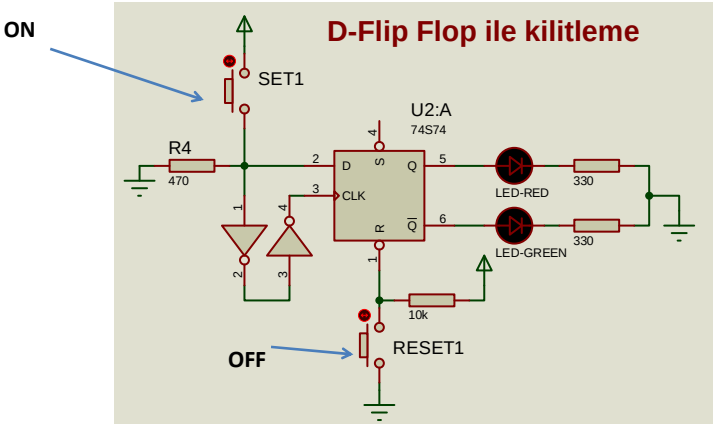


2.57< VS1<3.18

VS2< 0.65

Sequential logic (Flip Flop: Memory element) **Input Switch : Momentary Switch**

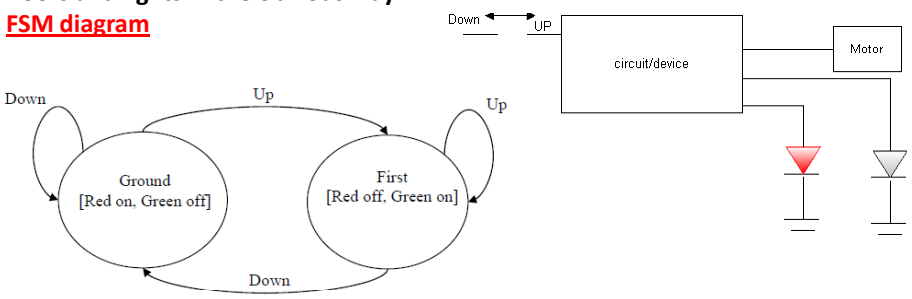


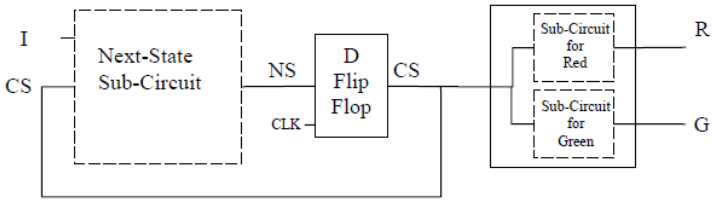


Elevator Example (FSM)

The elevator can be at one of two floors: Ground or First. There is one button that controls the elevator, and it has two values: Up or Down. Also, there are two lights in the elevator that indicate the current floor: Red for Ground, and Green for First. At each time step, the controller checks the current floor and current input, changes floors and lights in the obvious way.

FSM diagram

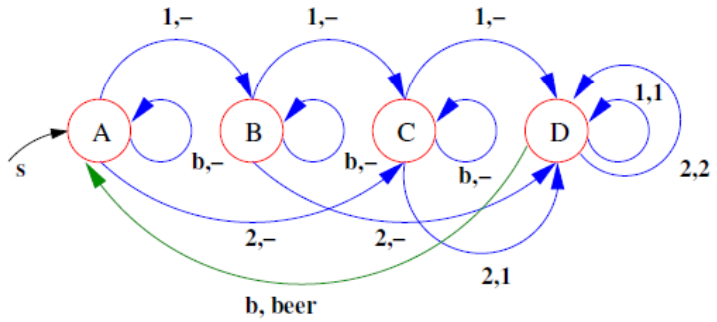




Design: Vending Machine

Consider a *magical* vending machine that when given $3, will output a can of beer after a button has been pressed. The machine can only take $1 and $2 coins, and if more than $3 is input, the remainder must be returned.

State diagram:



State Machines

A state machine is a digital circuit composed of "n" flip-flops that are interconnected to one another and to other inputs, with Boolean logic, in order to generate 2^n internal states. **The movement from one state to another is determined by a combination of the Boolean logic interconnecting the flip-flops and the inputs to the state machine.** The internal states are then decoded with additional Boolean logic to provide useful outputs at required time intervals. **A counter is therefore a very simple state machine.**

Memory

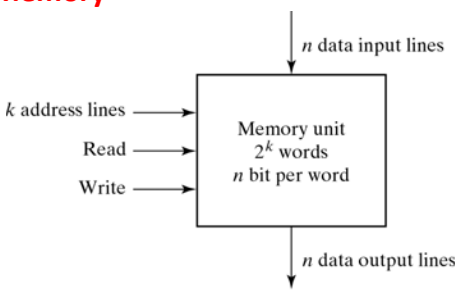


Fig. 7-2 Block Diagram of a Memory Unit

Memory address		Memory contest
Binary	decimal	
0000000000	0	1011010101011101
0000000001	1	1010101110001001
0000000010	2	0000110101000110
	⋮	⋮
1111111101	1021	1001110100010100
1111111110	1022	0000110100011110
1111111111	1023	1101111000100101

Fig. 7-3 Content of a 1024 × 16 Memory

Access to memory:
Adress, Data, control
Read cycle, Write cycle

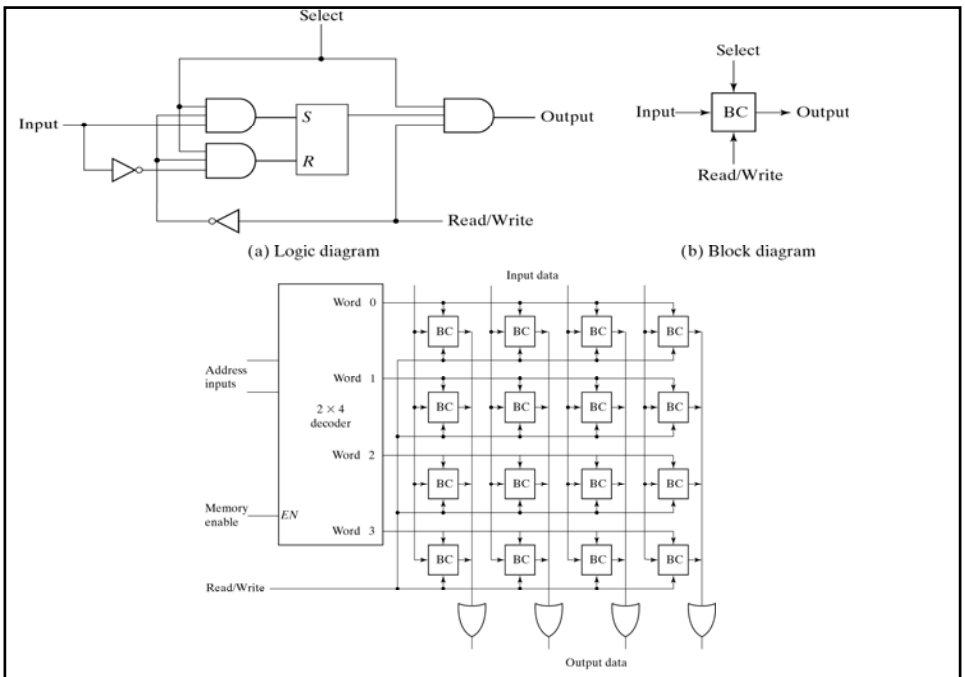
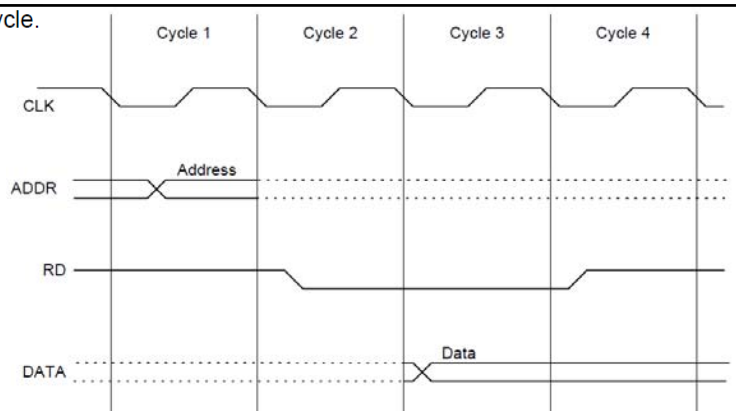


Fig. 7-6 Diagram of a 4 × 4 RAM



The following is a simplified description of what happens during each clock cycle as memory is read:

Cycle 1: The address bits of the memory operand are placed on the address bus (ADDR).

Cycle 2: The Read Line (RD) is set low (0) to notify memory that a value is to be read.

Cycle 3: The CPU waits one cycle to give memory time to respond. During this cycle, the memory controller places the operand on the data bus (DATA).

Cycle 4: The Read Line (RD) goes to 1, signaling that the CPU can now read the values on the

State machine and Microprocessor

State machines are now most commonly used where microprocessor solutions are too expensive (on very-high-volume, very-low-cost systems) or where it is necessary to perform very specific functions directly in hardware at a very high speed.

The state machine is also of importance to us because it is really the basis of the modern microprocessor. In other words, **a microprocessor is a generalised state machine that can be programmed to perform a specific sequence of functions.** This is analogous to the PAL which is a generalised Boolean logic device that can be programmed to produce a required logic.

The design of state machines can become rather complex when a large number of internal states need to be produced and it is fair to say that most modern designers would prefer to apply microprocessors or Digital Signal Processors

Microprocessor System: software-programmable state machine.

In general, we refer to a microprocessor as being a single chip that is used as the CPU of a computer or computer-based system.

The microprocessor may be considered as a sophisticated state machine,
A program instruction is nothing more than a specific bit pattern of ones and zeros that appears at the data port of the microprocessor. The state machine within the microprocessor has been designed to move from one state to another based upon those specific bit patterns.

The hardware design of the state machine within the microprocessor defines the course of events that will take place whenever a specific bit pattern arrives at the data port of the microprocessor. The hardware "programs" that are developed by the semiconductor manufacturers are known as micro-code. Micro-code is the lowest possible form of programming in microprocessors and it is the micro-code that defines the characteristics of the processor that end-users would recognise. Micro-code is analogous to PAL programming in that it instils the logic that defines the operational sequence for a state machine and defines how it will move from one state to another, based upon bit patterns entering via the data bus.

Each cycle (tick) of the clock causes the microprocessor **to jump from one internal state to another.** As with other state machines, the "next state" of the microprocessor is determined by a logical combination of its current internal state, together with the condition of all the various input lines connected to it.

The microprocessor is a device composed of all the other elements of the **Combinational** and **Sequential logic** . These elements include:

- **State machines**
- **Counters**
- **Registers**
- **Numerical circuits (Addition, Subtraction, etc… ALU)**
- **Boolean logic circuits (OR, AND, etc.)**
- **Combinational Boolean logic.**

By utilizing these elements, **at each cycle of the system clock**, a microprocessor can execute a very simple operation - **read data in; write data out**; store data in an internal register; add data contained in internal registers; compare data in internal registers and so on. So, **The number of bits that a microprocessor can handle as data or instructions defines the "bit-size" of the microprocessor.** Typical sizes are 8, 16, 32 and 64 bits. Since both data and program instructions appear at the data bus of the microprocessor as a collection of ones and zeros, the microprocessor cannot differentiate between them in any way other than the sequence in which they arrive.

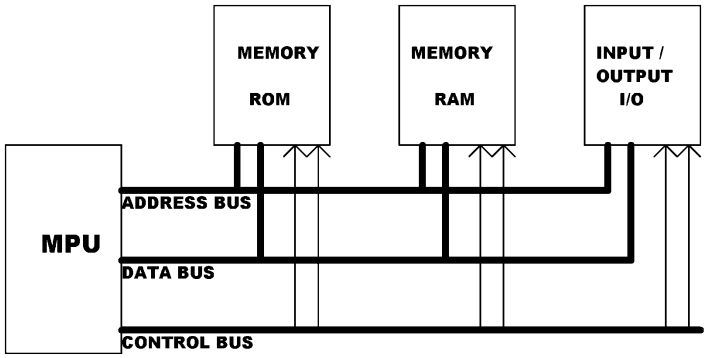
The microprocessor:

(i) **It is a device based upon a state machine**
(ii) It contains a number of **registers for temporary storage of data**
(iii) It contains **numerical and logical circuits** that can perform simple data manipulations between registers (eg: **ADD, OR, etc.)**
(iv) **It contains one input port for data (the data bus) and two output ports for data (the data bus and address bus)**
(v) The **state machine has been designed** (through micro-coding) to respond to particular bit patterns entering the data port. The specific bit patterns are the instruction set for the microprocessor
(vi) **The microprocessor moves from state to state at a speed determined by a "system clock" which is generally provided by another external chip**
(vii) **It takes one or more clock cycles to execute each instruction**. The number of clock cycles depends upon the architecture of the microprocessor (and its micro-coding) and the complexity of the instructions.

Points (vi) and (vii) are most important, since they are directly related to the performance of the microprocessor.

Microprocessor based system

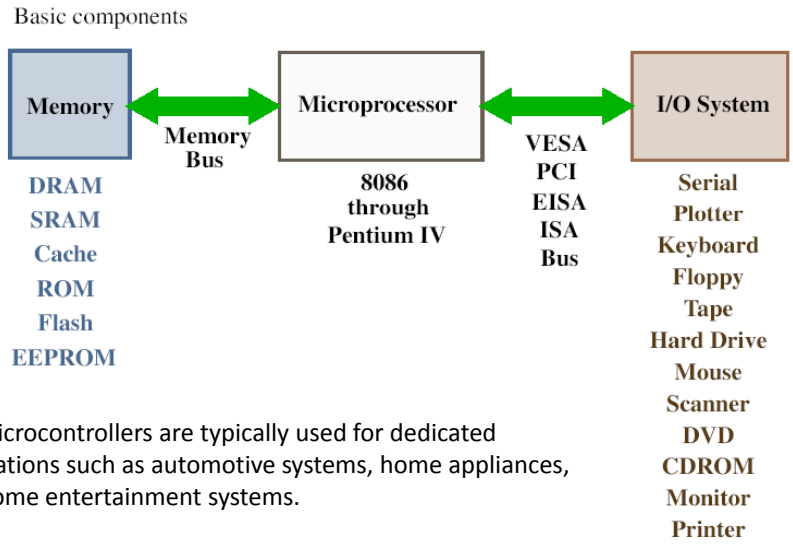
Address Bus ; Data Bus, Control Bus



CPU function is to execute (process) information stored in memory.
I/O devices, such as keyboard & monitor provide a means of communicating with the CPU.
The CPU is connected to memory and I/O through a group of wires **called a bus**. Allows signals to carry information from place to place.
In every computer there are three types of buses**: Address bus; Data bus; Control bus.**

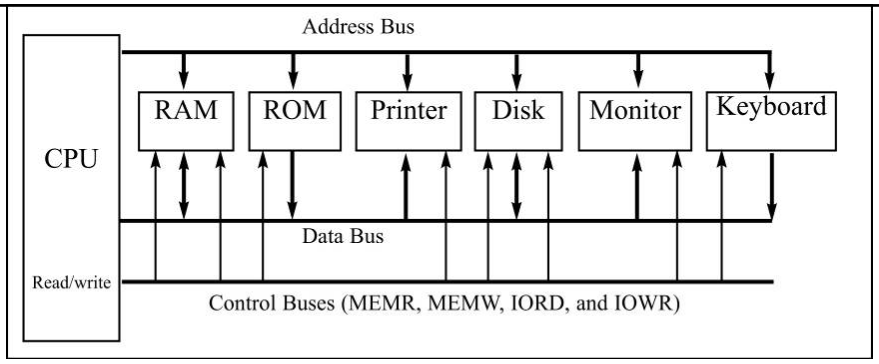
Examples of Embedded Systems

- Consumer electronics
  Cell phones, pagers, digital cameras, camcorders, PDAs, DVD players, calculators
- Home Appliances
  Microwave ovens, answering machines, thermostats, home security systems
- Office Automation
  Fax machines, copiers, printers, scanners
- Business Equipment
  Cash registers, curbside check-in, alarm systems, card readers, ATMs, product scanners
- Automobiles
  Transmission control, cruise control, fuel injection, antilock brakes, active suspension
- Computing system peripherals and networking systems
  Networking equipment, routers, printers, network cards, monitors and displays
- Any many more
- E.g. In 1999, a typical American household had one PC and about 40-50 embedded computers, this has risen to about 300-400 by 2004. An average car can have more than 50 embedded computers.



The microcontrollers are typically used for dedicated applications such as automotive systems, home appliances, and home entertainment systems.

A microprocessor does not know anything about "C" or Assembly Language.



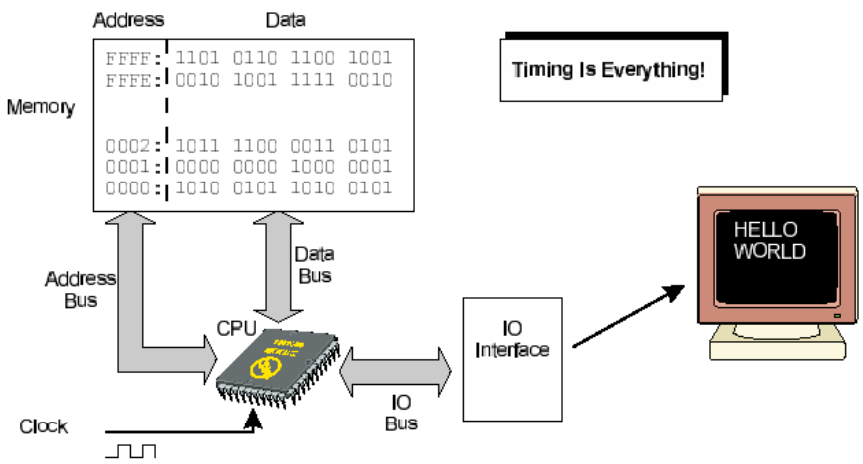
- **For a device (memory or I/O) to be recognized by the CPU, it must be assigned an address.**
  - The address assigned to a given device must be unique.
- The CPU puts the address (in binary) on the address bus & decoding circuitry finds the device.
  - ***The CPU then uses the data bus either to get data from that device or to send data to it.***
- **Control buses provide device read/write signals.**
  - To indicate if the CPU is asking for, or sending information.

Adress bus:
In the *address bus, information transfer takes place only in one direction, from the* microprocessor to the memory or I/O elements. Therefore, this is called a **"unidirectional bus."** This bus is typically 20 to 32 bits long. The size of the address bus determines the total number of memory addresses available in which programs can be executed by the microprocessor. The address bus is specified by the total number of address pins on the microprocessor chip. **The number of locations of memory is equal to to 2^x, where *x* is the number of address lines**

Data bus:
In the *data bus, data can flow in both directions, that is, to or from the* microprocessor. Therefore, this is a bidirectional bus. In some microprocessors, the data pins are used to send other information such as address bits in addition to data. This means that the data pins are time-shared or multiplexed. The Intel 8086 microprocessor is an example where the 20 bits of the address are multiplexed with the 16-bit data bus and four status lines.

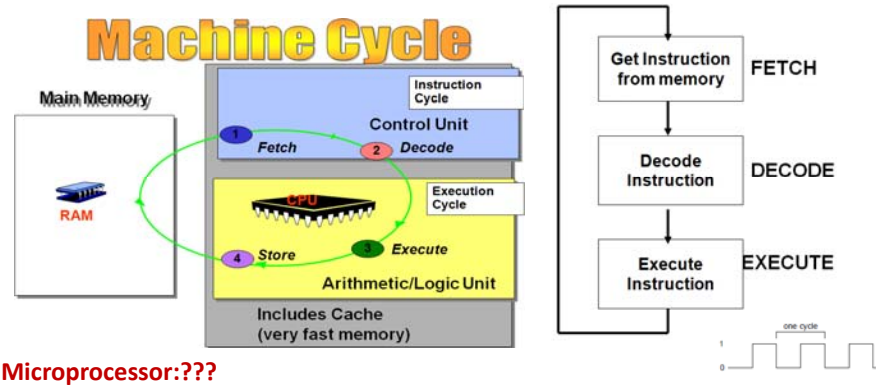
Control bus:
The *control bus consists of a number of signals that are used to synchronize the* operation of the individual microcomputer elements. The microprocessor sends some of these control signals to the other elements to indicate the type of operation being performed. Each microcomputer has a unique set of control signals. However, there are some control signals that are common to most microprocessors

Microprocessor based system



For the CPU to process information, the data must be stored in RAM or ROM.
The CPU cannot get the information from the disk directly because the disk is too slow. RAM & ROM are often referred to as *primary memory*.
Disks are called *secondary memory*.

All common microprocessor follows this sequence



Microprocessor:???

The microprocessor is typically a single VLSI (Very Large-Scale Integration) chip that contains all the registers, control unit, and arithmetic/ logic circuits of the microcomputer.

Clock Each operation involving the CPU and the system bus is synchronized by an internal clock that repeatedly pulses at a constant rate. The most basic unit of time for machine instructions is called the *machine cycle* (or *clock cycle time*), which is the time required for one complete clock pulse.

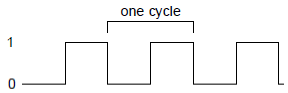
Machine instruction

A machine instruction requires at least one clock cycle to execute, and a few require in excess of 50 clocks (the multiply instruction on the 8088 processor, for example). Instructions requiring memory access often have empty clock cycles called *wait states* because of the differences between the speed of the CPU, the system bus, and memory circuits. (Recent research suggests that in the near future we may abandon the synchronized computing model in favor of a type of asynchronous operation that would not require a system clock.)

- ***Fetch:*** The control unit fetches the instruction, copying it from memory into the CPU and increments the program counter (PC).
- ***Decode:*** The control unit determines the type of instruction to be executed. It passes zero or more operands to the arithmetic logic unit (ALU) and sends signals to the ALU that indicate the type of operation to be performed.
- ***Fetch operands:*** If a memory operand is used, the control unit initiates a read operation to retrieve the input operand from memory.
- ***Execute:*** The arithmetic logic unit executes the instruction, sends its data to the output operand, and updates status flags providing information about the output.
- ***Store output operand:*** If the output operand is in memory, the control unit initiates a write operation to store the data.

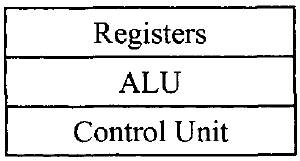
Clock Signals

The system clock signals are contained in the control bus. These signals generate the appropriate clock periods during which instruction executions are carried out by the microprocessor. The clock signals vary from one microprocessor to another. Some microprocessors have an internal clock generator circuit to generate a clock signal.



Clock Each operation involving the CPU and the system bus is synchronized by an internal clock that repeatedly pulses at a constant rate. The most basic unit of time for machine instructions is called the *machine cycle* (or *clock cycle time*), which is the time required for one complete clock pulse.

A microprocessor chip with the main functional elements



Register Section

The number, size, and types of registers vary from one microprocessor to another. However, the various registers in all microprocessors *carry out similar operations. The* register structures of microprocessors play a major role in designing the microprocessor architectures. Also, the register structures for a specific microprocessor determine how convenient and easy it is to program this microprocessor.

Basic Microprocessor Registers:

Instruction Register (IR). The instruction register stores instructions. The contents of an instruction register are always decoded by the microprocessor as an instruction.

Program Counter (PC). The program counter contains the address of the instruction or operation code (op-code). The program counter normally contains the address of the next instruction to be executed.

•Upon activating the microprocessor's **RESET input**, the address of the first instruction to be executed is loaded into the program counter.

•**The program counter contents** are automatically incremented by the microprocessor's internal logic.

•**The size of the program counter is determined by the size of the address bus.**

•Many instructions, such as JUMP and conditional JUMP, change the contents of the program counter from its normal sequential address value.

Memory Address Register (MAR). The memory address register contains the address of data. The microprocessor uses the address, which is stored in the memory address register, as a direct pointer to memory.

Accumulator (A). For an 8 bit microprocessor, the accumulator is typically an 8bit register. It is used to store the result after most **ALU operations**.

Control Unit
The main purpose of the control unit is to read and decode instructions from the program memory. To execute an instruction, the control unit steps through the appropriate blocks of the **ALU based on the op-codes contained in the instruction register. The op-codes** define the operations to be performed by the control unit in order to execute an instruction. The control unit interprets the contents of the instruction register and then responds to the instruction by generating a sequence of enable signals

RESET. This input is common to all microprocessors. When this input pin is driven to HIGH or LOW (depending on the microprocessor), the program counter is loaded with a predefined address specified by the manufacturer. For example, in the 80486, upon hardware reset, the program counter is loaded with FFFFFFF0

READ/WRITE: This output line is common to all microprocessors. The status of this line tells the other microcomputer elements whether the microprocessor is performing a READ or a WRITE operation. A HIGH signal on this line indicates a READ operation and a LOW indicates a WRITE operation. Some microprocessors have separate READ and WRITE pins.

READY. This is an input to the microprocessor. Slow devices (memory and I/O) use this signal to gain extra time to transfer data to or receive data from a microprocessor. The READY signal is usually an active low signal, that is, LOW means that the microprocessor is ready

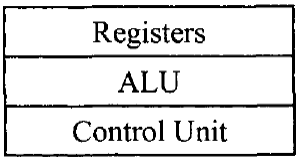
Interrupt Request (INT or IRQ). The external I/O devices can interrupt the microprocessor via this input pin on the microprocessor chip. When this signal is activated by the external devices, the microprocessor jumps to a special program, called the “interrupt service routine.”

Inside of Microprocessor : Arithmetic and Logic Unit (ALU):

The ALU performs all the data manipulations, such as arithmetic and logic operations, inside the microprocessor. The size of the ALU conforms to the word length of the microcomputer. This means that a 32-bit microprocessor will have a 32-bit ALU. Typically, the ALU performs the following functions:

Arithmetic and Logic Unit (ALU)

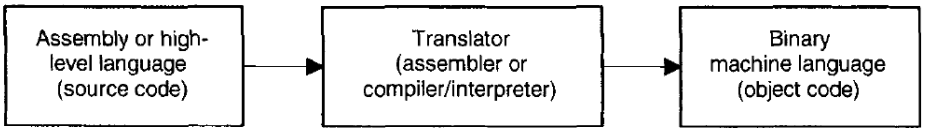
1. Binary addition and logic operations
2. Finding the ones complement of data
3. Shifting or rotating the contents of a general-purpose register 1 bit to the left or right through carry



Microcomputer Programming Languages

Microcomputers are typically programmed using semi-English-language statements (assembly language). In addition to assembly languages, microcomputers use a more understandable human-oriented language called the "high-level language." No matter what type of language is used to write the programs, the microcomputers only understand binary numbers. Therefore, the programs must eventually be translated into their appropriate binary forms. The main ways of accomplishing this are discussed later.

1. Machine language
2. Assembly language
3. High-level language



An assembler translates a program written in assembly language into a machine language program. A compiler or interpreter, on the other hand, converts a high-level language program such as C or C++ into a machine language program. Assembly or high-level language programs are called "source codes." Machine language programs are known as "object codes." A translator converts source codes to object codes. Next, we discuss the three main types of programming language in more detail.

An assembler translates a program written in assembly language into a machine language program. A compiler or interpreter, on the other hand, converts a high-level language program such as C or C++ into a machine language program. Assembly or high level language programs are called "source codes." Machine language programs are known as "object codes."

```
1011 1000 0000 0001 0000 0000
1011 1011 0000 0010 0000 0000
0000 0001 1101 1000
1111 0100
```

→

```
B80100
BB0200
01D8
F4
```

To increase the programmer's efficiency in writing a machine language program, hexadecimal numbers rather than binary numbers are used

Assembly Language

The next programming level is to use the assembly language. Each line in an assembly language program includes four fields:

1. Label field
2. Instruction, mnemonic, or op-code field
3 . Operand field
4. Comment field

Label	Mnemonic	Operand	Comment
START	MOV	AX, 1	move 1 into AX
	MOV	BX, 2	move 2 into BX
	ADD	AX, BX	add the contents of AX with BX
	JMP	START	jump to the beginning of the program

Assembly language

Software

- The sequence of commands used to tell a microcomputer what to do is called a program,
- Each command in a program is called an instruction
- 8088 understands and performs operations for 117 basic instructions
- The native language of the IBM PC is the machine language of the 8088
- A program written in machine language is referred to as machine code
- In 8088 assembly language, each of the operations is described by alphanumeric symbols instead of just 0s or 1s.

ADD AX, BX

Opcode

Source operand

Destination operand

Instructions

LABEL: INSTRUCTION ; COMMENT

Address identifier

Does not generate any machine code

Ex. START: MOV AX,BX ; copy BX into AX

- There is a one-to-one relationship between assembly and machine language instructions
- What is found is that a compiled machine code implementation of a program written in a high-level language results in inefficient code
  - More machine language instructions than an assembled version of an equivalent handwritten assembly language program
- Two key benefits of assembly language programming
  - It takes up less memory
  - It executes much faster

3

Converting Assembly Language Instructions to Machine Code

- An instruction can be coded with 1 to 6 bytes
- Byte 1 contains three kinds of information
  - Opcode field (6 bits) specifies the operation such as add, subtract, or move
  - Register Direction Bit (D bit)
    - Tells the register operand in REG field in byte 2 is source or destination operand
      - 1: destination
      - 0: source
  - Data Size Bit (W bit)
    - Specifies whether the operation will be performed on 8-bit or 16-bit data
      - 0: 8 bits
      - 1: 16 bits
- Byte 2 has two fields
  - Mode field (MOD)
  - Register field (REG)
  - Register/memory field (R/M field)

Continued

- REG field is used to identify the register fro the first operand

REG	W = 0	W = 1
000	AL	AX
001	CL	CX
010	DL	DX
011	BL	BX
100	AH	SP
101	CH	BP
110	DH	SI
111	BH	DI

- 2-bit MOD field and 3-bit R/M field together specify the second operand

CODE	EXPLANATION
00	Memory Mode, no displacement follows*
01	Memory Mode, 8-bit displacement follows
10	Memory Mode, 16-bit displacement follows
11	Register Mode (no displacement)

*Except when R/M = 110, then 16-bit displacement follows

(a)

MOD = 11			EFFECTIVE ADDRESS CALCULATION			
R/M	W = 0	W = 1	R/M	MOD = 00	MOD = 01	MOD = 10
000	AL	AX	000	(BX) + (SI)	(BX) + (SI) + D8	(BX) + (SI) + D16
001	CL	CX	001	(BX) + (DI)	(BX) + (DI) + D8	(BX) + (DI) + D16
010	DL	DX	010	(BP) + (SI)	(BP) + (SI) + D8	(BP) + (SI) + D16
011	BL	BX	011	(BP) + (DI)	(BP) + (DI) + D8	(BP) + (DI) + D16
100	AH	SP	100	(SI)	(SI) + D8	(SI) + D16
101	CH	BP	101	(DI)	(DI) + D8	(DI) + D16
110	DH	SI	110	DIRECT ADDRESS	(BP) + D8	(BP) + D16
111	BH	DI	111	(BX)	(BX) + D8	(BX) + D16

(b)

An Assembly language instruction consists of a mnemonic, optionally followed by one or two *operands*.

Operands are the data items being manipulated.
Mnemonics are commands to the CPU, telling it what to do with those items.

Advantages of Assembly Language

1. Shows how program interfaces with the processor, operating system, and BIOS.
2. Shows how data is represented and stored in memory and on external devices.
3. Clarifies how processor accesses and executes instructions and how instructions access and process data.
4. Clarifies how a program accesses external devices.

Reasons for using Assembly Language:

1. A program written in Assembly Language requires considerably less memory and execution time than one written in a high –level language.
2. Assembly Language gives a programmer the ability to perform higly technical tasks that would be difficult, if not impossible in a high-level language.
3. Although most software specialists develop new applications in high-level languages, which are easier to write and maintain, a common practice is to recode in assembly language those sections that are time-critical.
4. Resident programs (that reside in memory while other program execute) and interrupt service routines (that handle input and output) are almost always develop in Assembly Language.