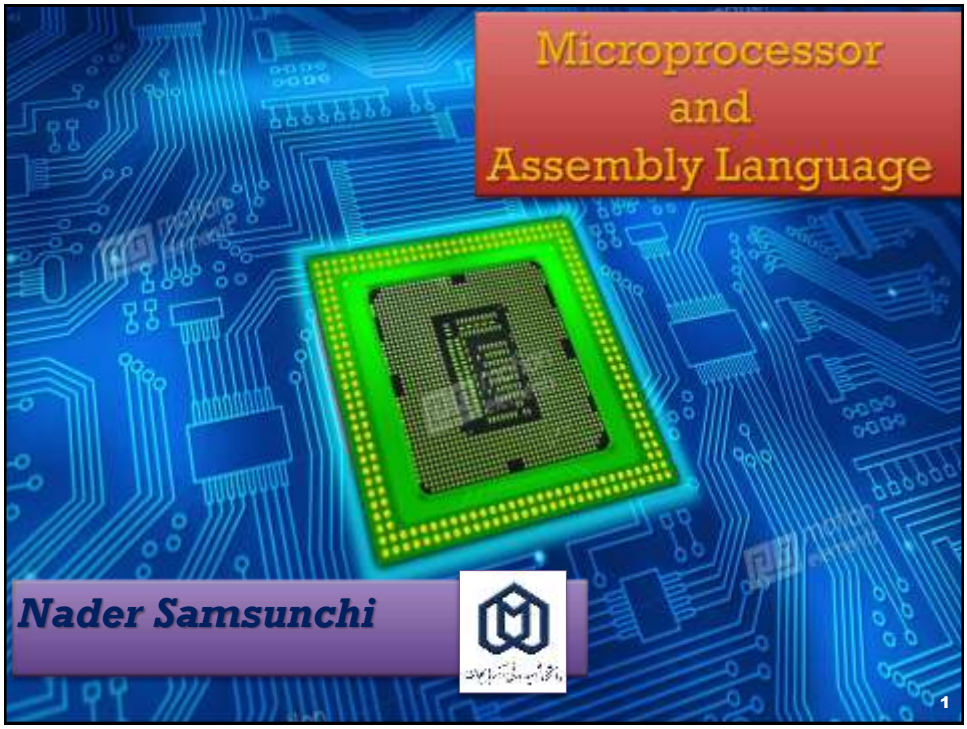


Computers are everywhere !
maybe Hidden (Embedded)

A "short list" of embedded systems

Modems
MPEG decoders
Network cards
Network switches/routers
On-board navigation
Pagers
Photocopiers
Point-of-sale systems
Portable video games
Printers
Satellite phones
Scanners
Smart ovens/dishwashers
Speech recognizers
Stereo systems
Teleconferencing systems
Televisions
Temperature controllers
Theft tracking systems
TV set-top boxes
VCR's, DVD players
Video game consoles
Video phones
Washers and dryers

Anti-lock brakes
Auto-focus cameras
Automatic teller machines
Automatic toll systems
Avionic systems
Battery chargers
Camcorders
Cell phones
Cell-phone base stations
Cordless phones
Cruise control
Digital cameras
Disk drives
Electronic card readers
Electronic instruments
Electronic toys/games
Factory control
Fax machines
Fingerprint identifiers
Home security systems
Life-support systems
Medical testing systems

And the list goes on and on

What is a Computer ?

Memory

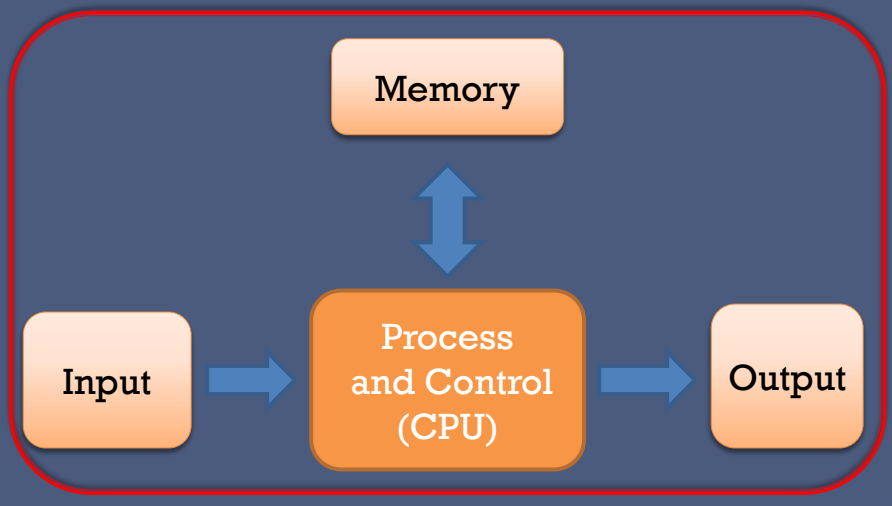
Input

Process and Control (CPU)

Output

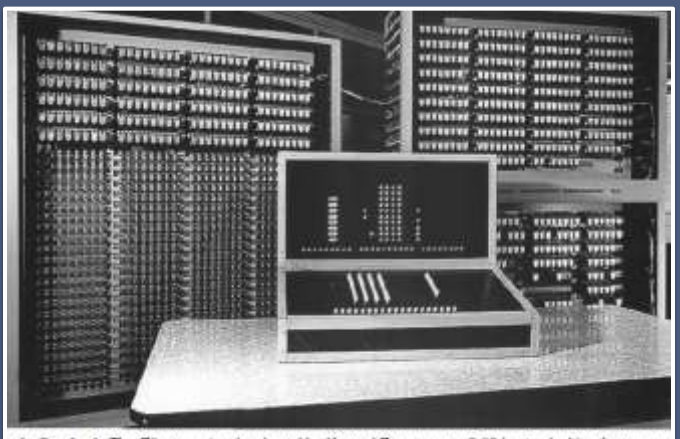
How to make it ?

Idea #1 :Make Computer as a whole system



A Brief History

- 1941 : Zuse Z3



1941: The Z3 computer developed by Konrad Zuse uses a 5.33 hertz clocking frequency. (Photo courtesy of Horst Zuse, the son of Konrad.)

N. Samsunchi 7

A Brief History

- 1946 : ENIAC , 30,000Kg , $7,000,000,f=100KHz

N. Samsunchi 8

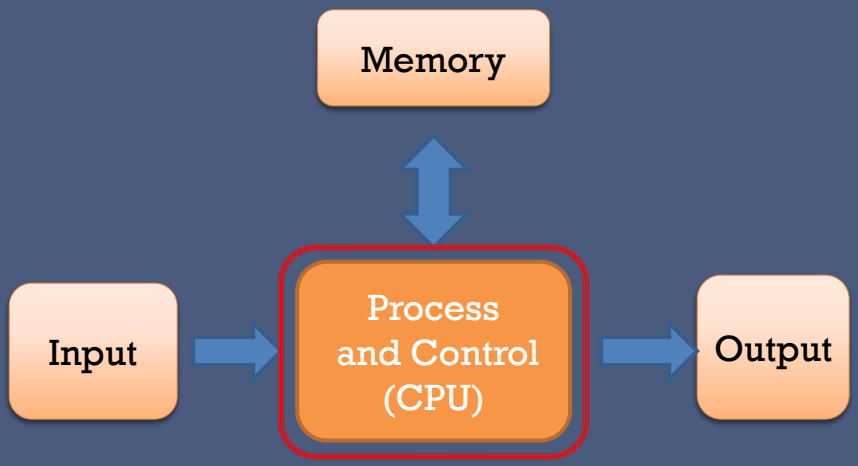
A Brief History

- 1949 : EDVAC
- Main Players : IBM , DEC, Honeywell

Mainframe Computers :

- Big ,
- Expensive ,
- Independent,
- Dedicated Hardware and Software,
- Single purpose ,
- Incompatible.

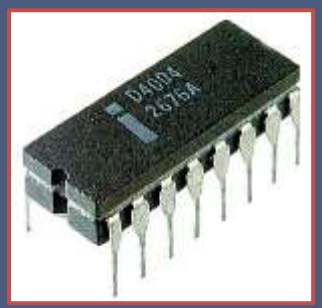
Idea #2 : Separate the CPU ➔Microprocessor(uP)



N. Samsunchi 11

A Brief History …

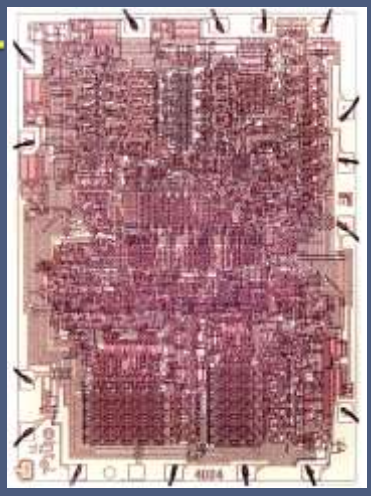
- 1971 : Intel 4004
  - For Busicom calculator
- 2300 transistors
- 400 – 800 kHz
- 4-bit word size
- 16-pin DIP package
- Masks Drawn with color pencils !



N. Samsunchi 12

A Brief History ..

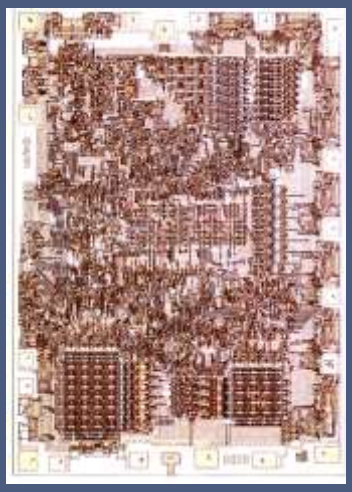
- 1971 : Intel 4004
  - For Busicom calculator
-



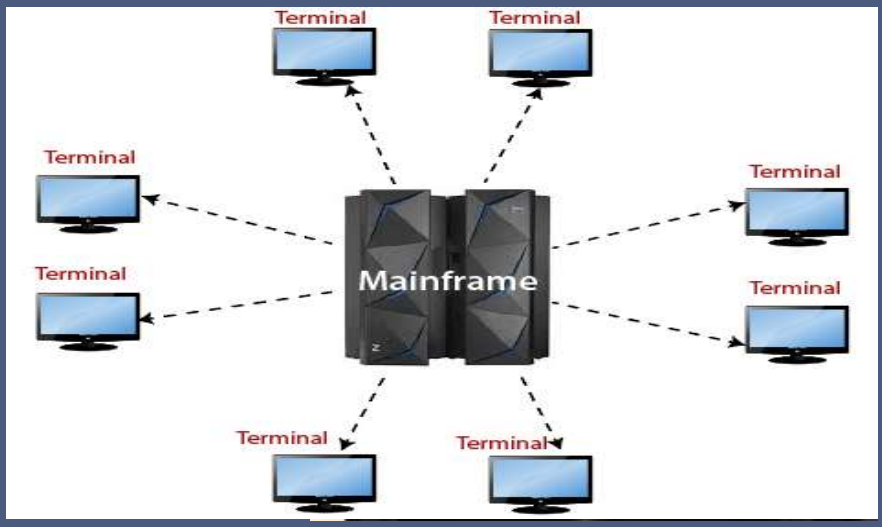
lor pencils !

A Brief History …

- 1972 : Intel 8008
- For Terminals
- 3500 transistors
- 500 – 800 kHz
- 8-bit word size
- 18-pin DIP package

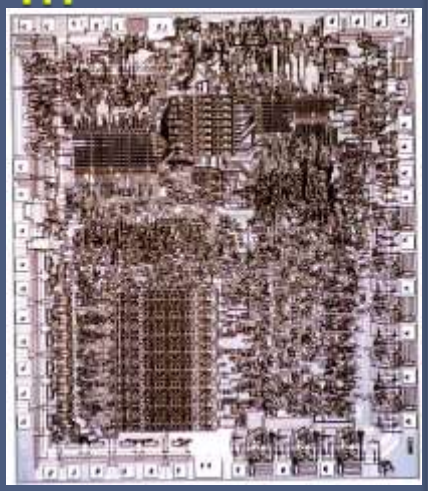


A Brief History …



A Brief History …

- 1974 : Intel 8080
- Used in Altair computer
  - (early hobbyist PC)
- 4500 transistors
- 2 MHz
- 8-bit word size
- 16-bit address bus
- 40-pin DIP package
- ➔ Intel 8085



A Brief History …

- 1974 : Intel 8080
- Used in Altair computer
  - (early hobbyist PC)

A Brief History …

- 1974 : Motorola 6800
- 1976 : Zilog Z80

A Brief History …

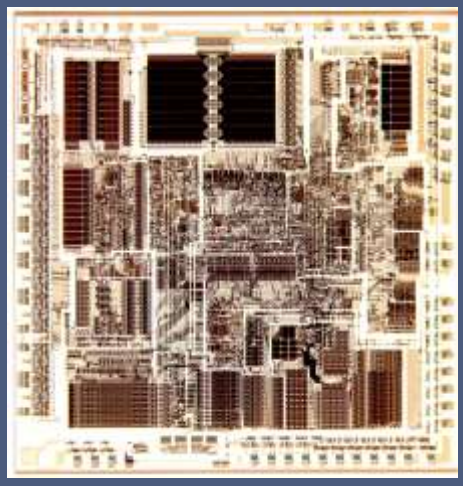
- 1980 : Intel 8086
- → Intel 8088
- Revolutionary products
- 29,000 transistors
- 5-10 MHz
- 16-bit word size
- 40-pin DIP package
- Introduced x86 Architecture

A Brief History .

- 1981 : IBM PC
- 1983→PC/XT
- Intel 8086/88→80186→80286→80386→80486

A Brief History …

- 1982 : 80286
- IBM PC AT
- 134k transistors
- 6-12 MHz
- 16-bit word size
- 68-pin

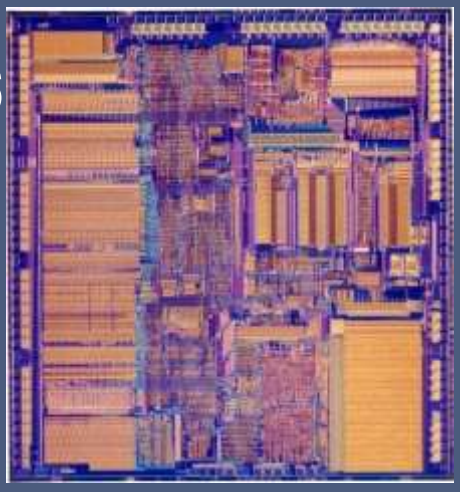


A Brief History …

- 1982 : 80286
- IBM PC AT
- 134k transisto
- 6-12 MHz
- 16-bit word si
- 68-pin

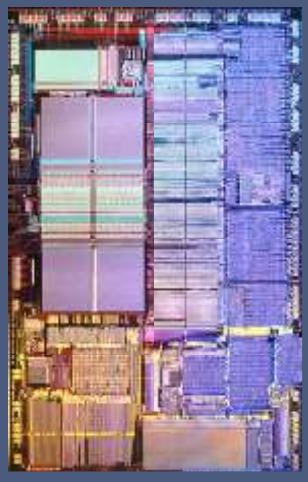
A Brief History …

- 1985 : 80386
  - Modern x86 Architecture
  - 275k transistors
  - 16-33 MHz
  - 32-bit word size
  - 100-pin



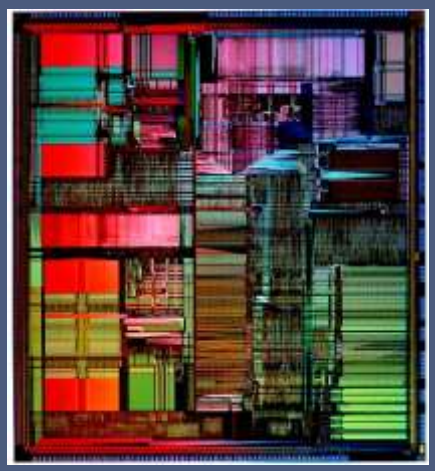
A Brief History …

- 1989 : 80486
  - Floating point unit
  - 1.2M transistors
  - 25-100 MHz
  - 32-bit word size
  - 168-pin



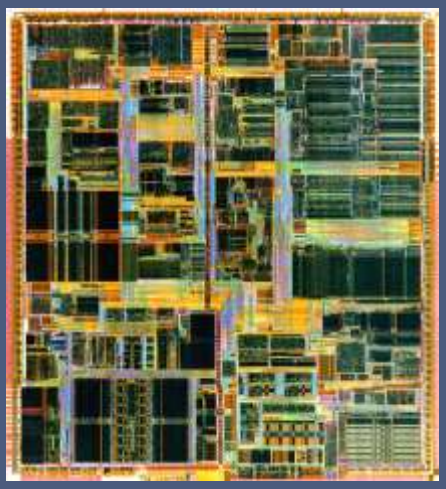
A Brief History …

- 1993 : Pentium
  - 3.2M transistors
  - 60-300 MHz
  - 32-bit word size
  - 296-pin



A Brief History …

- 1995~1999 : Pentium Pro / II / III
  - Multimedia instructions
  - 5.5M-28M transistors
  - 166-1000 MHz
  - 32-bit word size
  - Xeon (for Servers)



A Brief History …

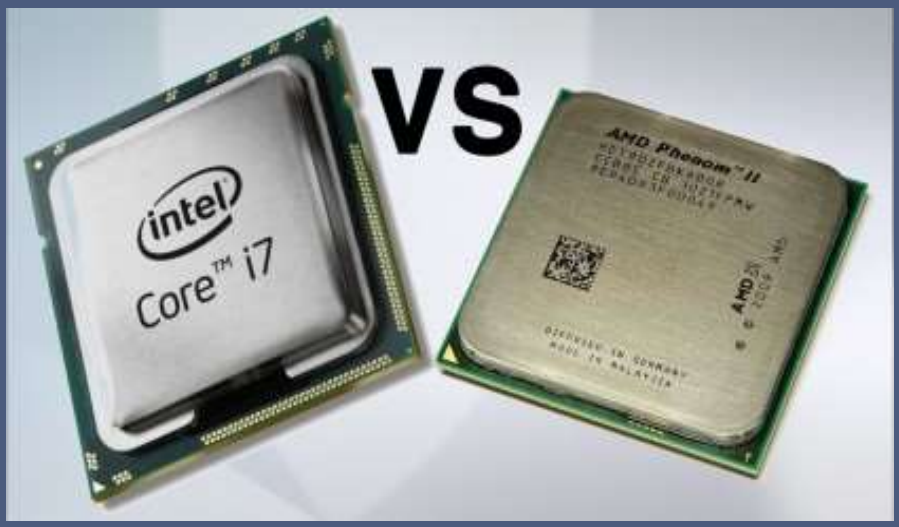
- 2001 : Pentium 4
- 42-125M transistors
- 1.4-3.4 GHz
- 32-bit word size
- 478-pin
- 2004:First 64-bit Instructions

A Brief History …

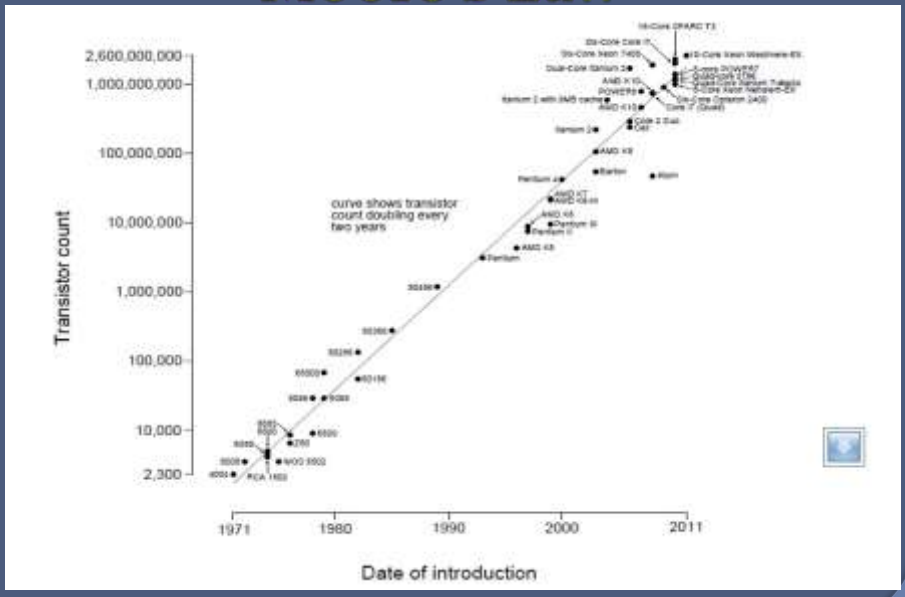
- Intel Pentium D
- Intel Pentium Dual-Core
- Intel Core
- Intel Core2
- Intel Core i3
- Intel Core i5
- Intel Core i7
- Intel Core i9
- Intel Itanium

- AMD

A Brief History …



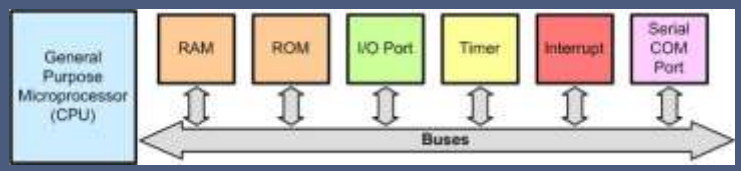
Moore's Law

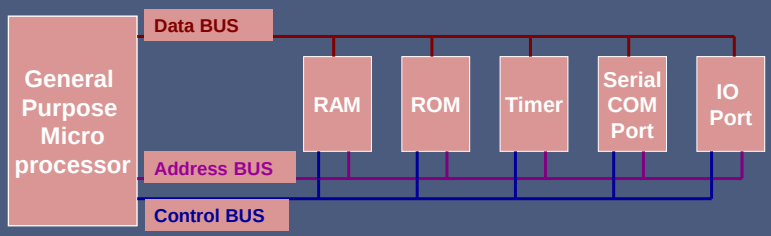


Microprocessor

- Cheaper
- Smaller
- General Purpose
- Hardware / Software Compatibility

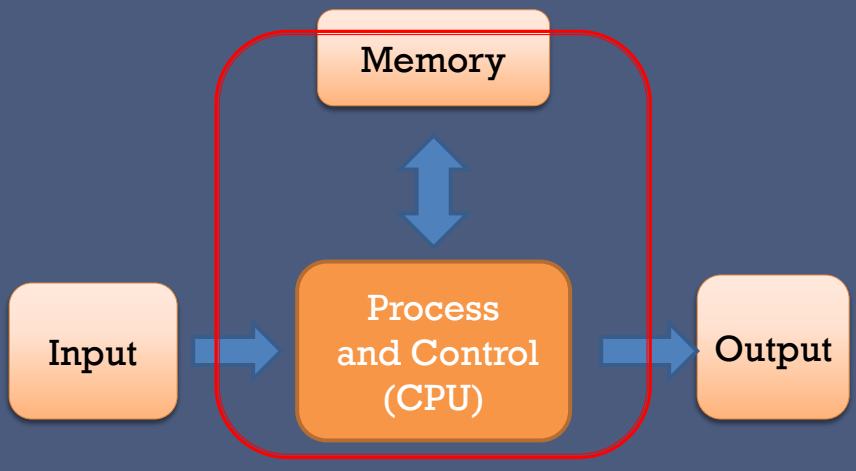
Microcomputer





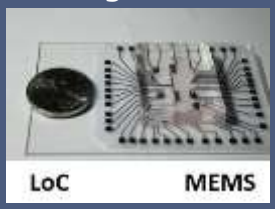
Microcomputer

Idea #3 : Include Some Memory and I/O ➔Microcontroller(uC)



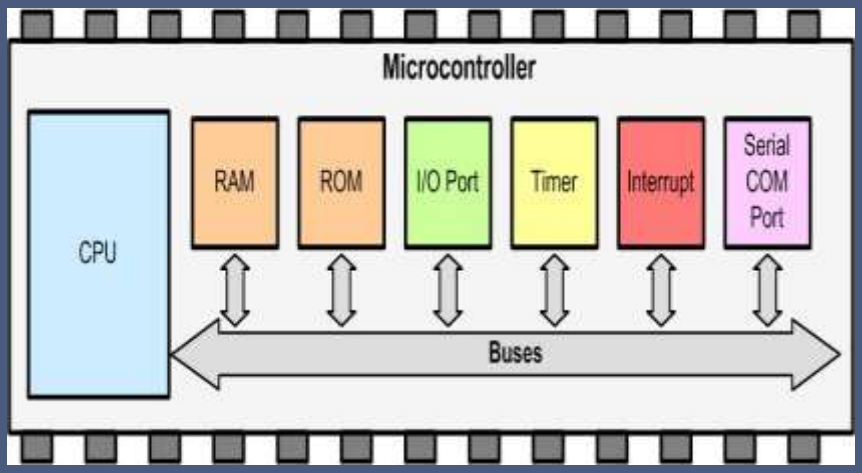
What is a Microcontroller?

- Integrated chip that typically contains integrated CPU, memory (RAM -ROM), I/O ports on a single Chip.
- Not a general-purpose computer .Designed to execute a specific task to control a single system (Embedded System)
- System on a Chip (SoC)
- Lab. On a chip (LoC)



- Smaller & Specified (cost reduction)

What is a Microcontroller?



Microcontroller vs Microprocessor

History

- 1974 : TMS1000
- 4 bit

Most common microcontrollers

- AVR (Atmel)
- PIC(Microchip Technology)
- HCS12 (Freescale)
- 8051(Intel)
- ARM (Acorn)
- Z8(Zilog)
- BASIC Stamp(Parallax)
- ESP

Most common microcontrollers

ESP8266

ESP-01 module by Ai-Thinker

Manufacturer	Espressif Systems
Type	32-bit microcontroller
CPU	@ 80 MHz (default) or 160 MHz
Memory	32 KiB instruction, 80 KiB user data
Input	16 GPIO pins
Successor	ESP32

ESP32

ESP-WROOM-32 module with ESP32-D0WDQ6 chip

Manufacturer	Espressif Systems
Type	Microcontroller
Release date	September 6, 2016[1]
CPU	Tensilica Xtensa LX6 microprocessor @ 160 or 240 MHz
Memory	520 KiB SRAM
Power	3.3 V DC
Predecessor	ESP8266

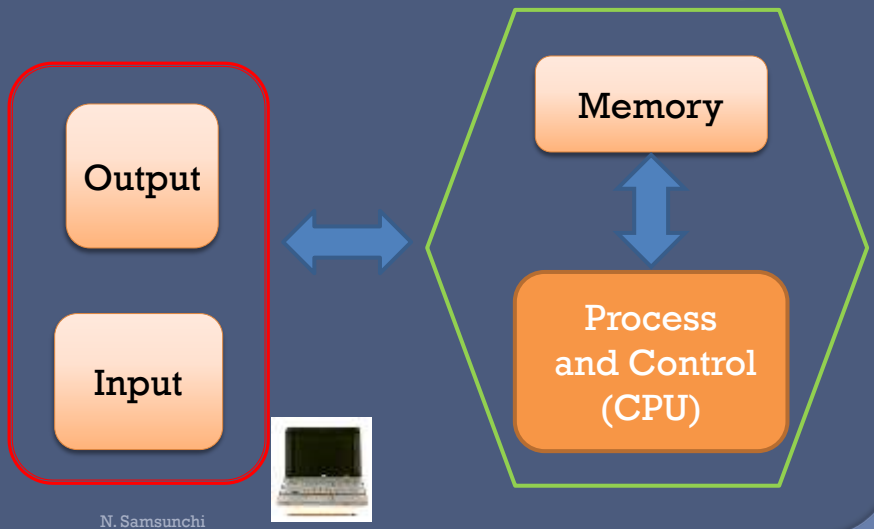
Most common microcontrollers

- ESP (Espressif Systems)
- ESP8266
- 16 I/O Pins
- 1MB Memory
- WiFi Network

Most common microcontrollers

- ESP (Espressif Systems)
- ESP32
  - 16 I/O Pins
  - 1MB Memory
  - WiFi Network
  - Bluetooth: v4.2,BLE5

Idea #4 : Separate I/O from Processing and Memory → Cloud Computing



N. Samsunchi

43

Cloud Computing Services

- Infrastructure as a service (IaaS)
- Platform as a service (PaaS)
- Software as a service (SaaS)
- Mobile "backend" as a service (MBaaS)

N. Samsunchi

44

Classes of Computers

- Server Computers
  - Network based.
  - High Capacity, Performance, Reliability
- Personal Computers
  - General Purpose.
  - Medium Capacity, Performance, Reliability
- Embedded Computers
  - Single Purpose.
  - Low Capacity, Performance
  - I/O oriented.

Levels of Program Coding

- High Level Language
  - Closer to problem domain.
  - Productivity, portability
- Assembly Programming Language
  - Textual representation of instructions.
  - Speed
  - Processor oriented
- Hardware Representation
  - Binary Digits(Bits)

Instruction Set Architecture (ISA)

- Instructions.
- Registers.
- Memory Architecture.
- Addressing Modes.

Benefits:

- Abstraction
- Multiple Implementations :
  - Physical (INTEL - AMD)
  - Virtual

Instruction Set Architecture (ISA)

Benefits:

- Abstraction
- Multiple Implementations
  - Physical (INTEL - AMD)
  - Virtual

How to select a proper CPU?

- Availability
  - Market
  - Support (Data Sheets, Development Tools, Experts,…)
  - Price
- Power
- Performance

What is Performance ?

Intel Corei3 , f=733MHz
Vs
ARM , f=1500MHz
Which has a higher performance?

CPU Clocking

- f : Clock Frequency (Hz : Cycles per second)
- T : Clock period (S: Duration of a clock cycle)
  - **T=1/f**
- Example : f = 2 GHz = 2 x 10^9 Hz
  - ➔ T= 1/f = 1/2GHz=0.5 nS=500 pS

What is Performance ?

Intel Corei3 , f=733MHz
Vs
ARM , f=1500MHz
Which has a higher performance?

Computing Performance factors

- Algorithm
  - Determines No. of Operations executed.
- Programming Language, Compiler , Architecture
  - Determines No. of Machine Instructions executed per operation.
- Processor and Memory System
  - Determines how fast instructions are executed.
- I/O System
  - Determines how fast I/O operations are Executed.

Performance
Relative Performance

- Definition: Performance=1/Execution Time
- Example : Time Taken to run a program :
  - Computer A : 10 S
  - Computer B : 15 S
  - Relative Performance = (1/10) / (1/15) = 15/10=1.5
    - ➔ Computer A is 1.5 times faster than Computer B.
- Benchmark Programs (Antutu , Landmark, ..)

CPU Time

- Definition :
  - CPU Time=No. of CPU Clock Cycles X Clock Cycle Time
  - **CPU Time=N.T=N/f**
- To improve Performance (reduce CPU Time) :
  - Decrease N
  - Increase f

→ Tradeoff ←

CPU Time Example :

- Computer A : f1=2GHz , CPU Time1 = 10 S
- Computer B : N2=1.2N1
  - CPU Time2=6 S
  - f2=?

Answer -CPU Time Example :

- CPU Time1=N1/f1=10S → N1=(10)(2GHz)
- CPU Time2=N2/f2=6S → f2=N2/6=1.2N1/6
  - f2=(1.2)(10)(2GHz)/6=4GHz

Instruction count and CPI

- No. of Clock Cycles=Instructions Count x Cycles Per Instruction
- **CPU Time=N.T=N/f**
- **CPU Time=(ICN)(CPI) / f**
- **ICN: Instruction Count**
- **CPI: Cycle Per Instruction**

Instruction count and CPI

- Example :
  - Computer A: f1=4GHz , CPI1= 2
  - Computer B: f2=2GHz. CPI2= 1.2 , same ISA
  - Which is faster ? By how much ?

Instruction count and CPI

- Answer :
  - Performance1/Performance2 =CPU Time2/CPU Time1
  - Performance1/Performance2 =(CPI2/CPI1)(f1/f2)
  - Performance1/Performance2 =(1.2/2)(4/2)=1.2
  - → **A is faster , by 1.2 times.**

CPI

- CPI=Total No. of Clock Cycles / No. of Instructions
- Example :

Type	A	B	C
No. of required Clock Cycles	1	2	3
No. of Instructions-Software 1	2	1	2
No. of Instructions-Software 2	4	1	1

CPI1=(2+2+6)/(2+1+2)=2
CPI2=(4+2+3)/(4+1+1)=1.5

Performance Summary

- **CPU Time=(ICN).(CPI).(1/f)**
- Performance depends on :
  - Algorithm :
    - affects ICN ,possibly CPI
  - Software (Programming Language , Compiler) :
    - affects ICN, CPI
  - ISA Hardware :
    - affects ICN, CPI, f

Power

- Need for Mobility
- Extra Heat Problem
- World Climate Problems.

- In Current technology (CMOS), Power depends on :
  - f
  - Voltage ^2

  - Low Voltage design (5V ➔ 1V)

Power is a limiting Factor (The Power Wall)

- We can NOT reduce Voltage further
  - Why?
- We can NOT reduce f
  - Why?
- We can NOT accept more heat
  - Why?

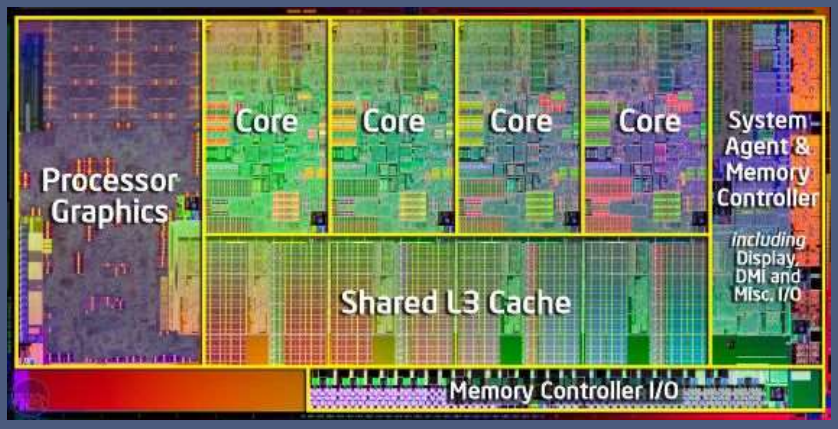
- What else to do?
  - **Change the architecture**

Unproportioned Power Consumption

- Example : AMD OPTERON X4 CPU SPEC Power Benchmark
  - At %100 Load : 295 W (Full Power)
  - At %50 Load : 246W (%83 Full Power)
  - At %10 Load : 180W (%61 Full Power)
  - At %0 Load :141W (%48 Full Power)
- Google Data Centers : (0.3 Wh per search)
  - Mostly operate at %10- %50 load
  - At %100 load less than %1 of the time.
  - ➔ Do NOT Overestimate Processing Power needs!

Multicore microprocessors

- More than one processor per chip.



Parallel Processing …

- New Problems :
  - Parallel Programming
  - Load Balancing
  - Communication and Synchronization

How to speed up the CPU ?

- Change the architecture
  - Parallel Processing
  - Pipelining
  - RISC vs CISC

Old Architectures (Fetch, Decode , Execute)

Instruction	1	1	1	2	2	2
Fetch						
Decode						
Execute						
Clock	1	2	3	4	5	6

Pipelining

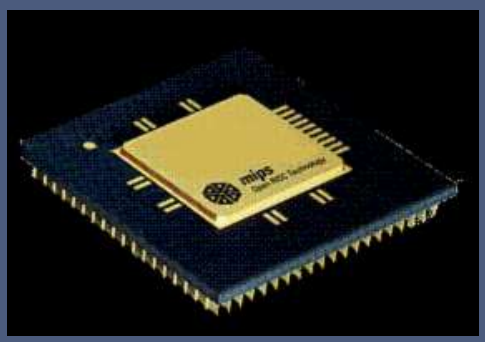
Instruction	1	1 , 2	1,2,3	2,3,4
Fetch				
Decode				
Execute				
Clock	1	2	3	4

RISC vs. CISC

- CISC (Complex Instruction Set Computer)
  - Put as many instruction as you can into the CPU
- RISC (Reduced Instruction Set Computer)
  - Reduce the number of instructions, and use your facilities in a more proper way.

RISC architecture

- 1980 : The "20/80" Rule ,By Patterson.
- Berkeley University → SPARC architecture.
- Stanford University → MIPS architecture.



RISC architecture …

- IBM Corp. → PowerPC architecture.
- Atmel Corp. →AVR architecture.

RISC architecture …

- Acorn Corp. → ARM architecture.

Cortex-A
Highest performance
Optimized for rich operating systems

Cortex-R
Fast response
Optimized for high-performance, hard real-time applications

Cortex-M
Smallest/lowest power
Optimized for discrete processing and microcontroller

SecurCore
Tamper resistant
Optimized for security applications

ARM SecureCore

N. Samsunchi

77

Smart Tags

N. Samsunchi

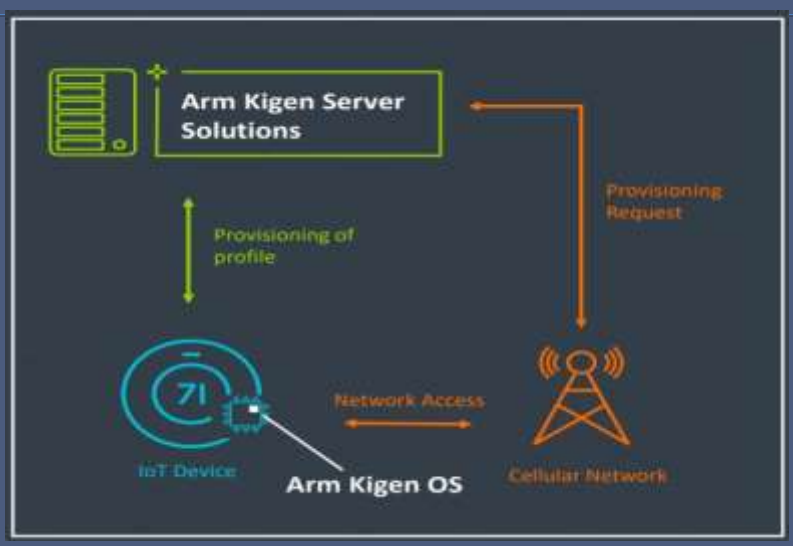
78

Smart Tags

Smart Tags

ARM iSIM

ARM iSIM



The Amdahl's Law

$$T_{\text{Improved}} = \frac{T_{affected}}{\text{Improvement Factor}} + T_{unaffected}$$

The Amdahl's law-example

- Total Time=100s
  - Multiplication Time=75S
  - We want 2 times better Performance. How much improve Multiplication?

The Amdahl's law-example

- P2=2P1 → T2=T1/2=100S/2=50S
- 50=(75/IF)+(100-75) → IF=3

- We want 4 times performance. Recalculate IF.
  - → It can NOT done!

MIPS as a Performance Metric

- MIPS : Million Instructions Per Second.

- Example :
  - CPU 1 : 100 MIPS.
  - CPU 2: 120 MIPS.
  - Which is Better ?

  - It is a method of measuring the raw speed of a computer's processor

CPU Benchmarking

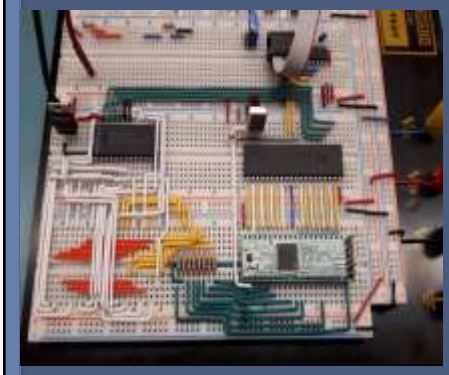
- SPEC (Standard Performance Evaluation Corp)
- How to do benchmark:
  1. Execute n Standard Programs and measure execution times.
  2. Normalize relative to Reference machine.
  3. Calculate geometric mean :

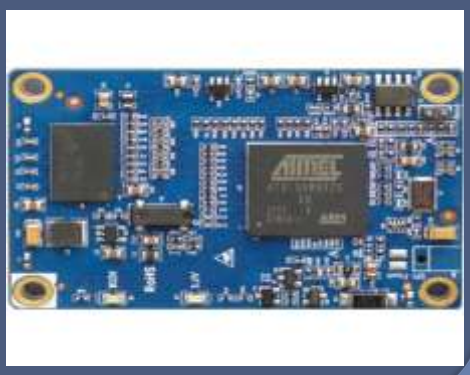
$$\sqrt[n]{\prod_{1}^{n} Normalized\ Execution\ Times}$$

- **Assignment 1:**
- Why geometric mean and NOT arithmetic mean?

Project Implementation

- Breadboards
  - Design, Test and Prototyping
- Dedicated Board.
  - Production





Project Implementation

- Single Board Computers (SBC)

Project Implementation

- Raspberry Pi Single Board Computer (SBC)

Project Implementation …

- RaspberryPi Single Board Computer (SBC)

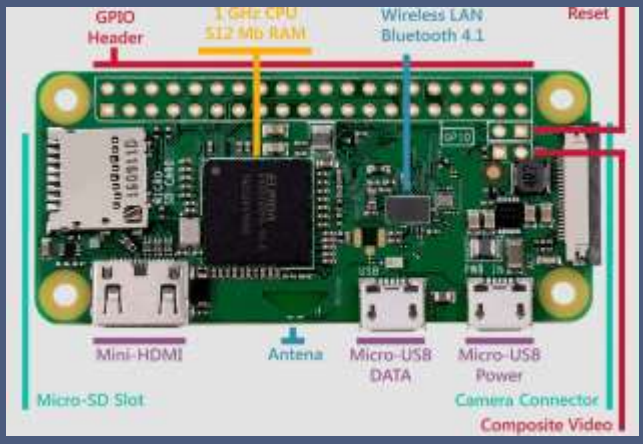
https://news.engineering.utoronto.ca/u-of-t-engineering-team-programs-single-board-computers-to-remotely-monitor-covid-19-patients-and-protect-health-care-workers/

Project Implementation …

- RaspberryPi Single Board Computer (SBC)

Project Implementation …

- RaspberryPi Zero W



Project Implementation …

- Full Computer :
- PC (Desktop, Laptop)
- Industrial PC (IPC)

Project Implementation …

- Military PC (MPC)

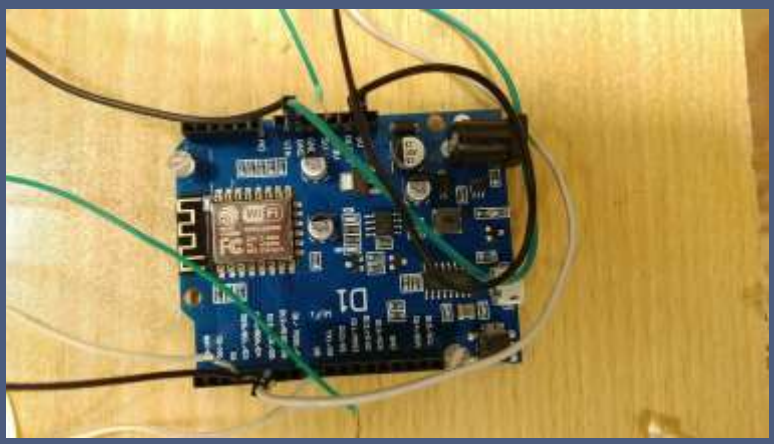
Assignment 2: What is new?

- A. Newest and the most Powerful Intel and AMD X86 *desktop grade* CPUs and their specifications(f , No. of Cores)?
- B. *Newest* and *the most Powerful* Arduino and Raspberry Pi SBCs and their specifications(uP or uC, f, Memory)?
- C. Introduce "Halocode" SBC.

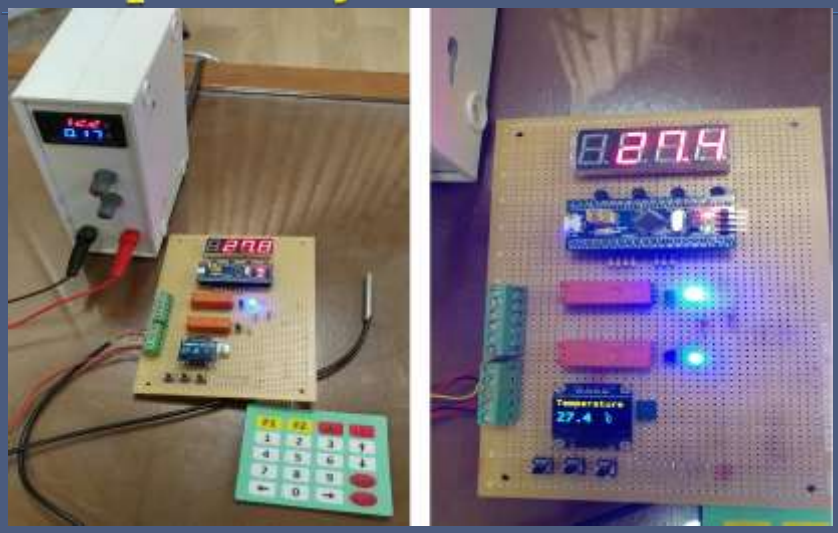
Sample Projects

Sample Projects …

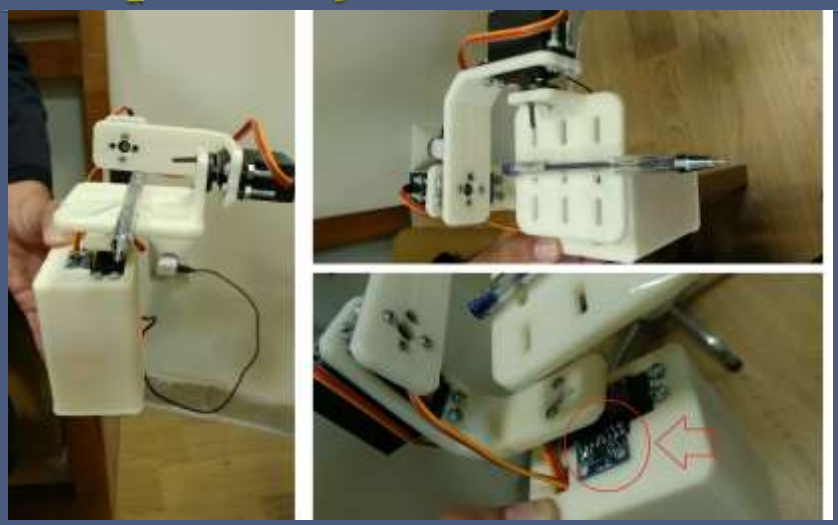
- WEMOS D1 Single Board Computer (SBC)



Sample Projects …

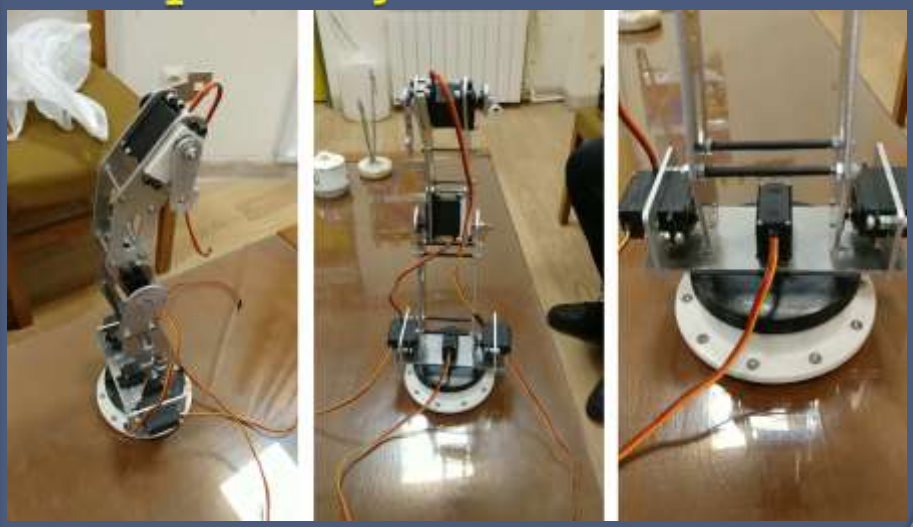


Sample Projects …



gimbal

Sample Projects …

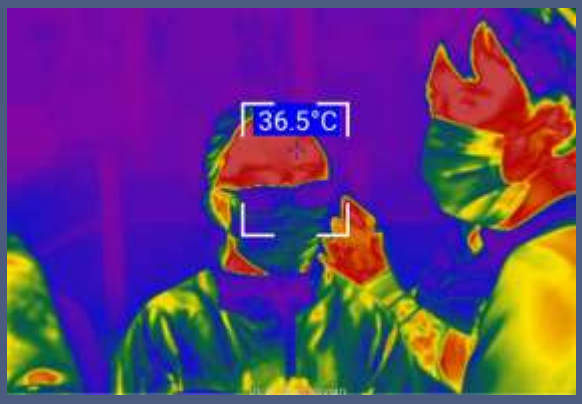


Assignment 3: Smart Helmet

- Main Idea

Assignment 3: Smart Helmet...

- Specifications:
- 1.Contactless Temperature Measurement
-

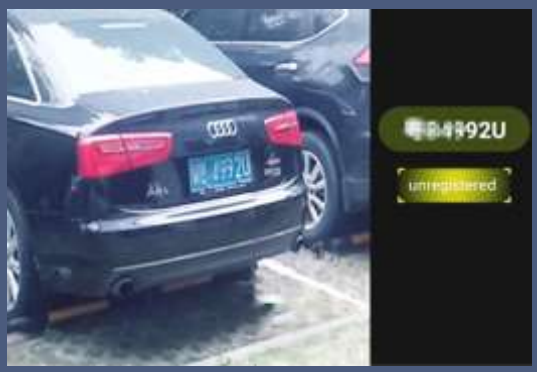


Assignment 3: Smart Helmet…

- Specifications:
- 1. Contactless Temperature Measurement
- Distance:5 m
- Accuracy: 0.3 °C

Assignment 3: Smart Helmet…

- Specifications:
- 2. License Plate Recognition
- Distance:5 m
- Accuracy: 99%



Assignment 3: Smart Helmet…

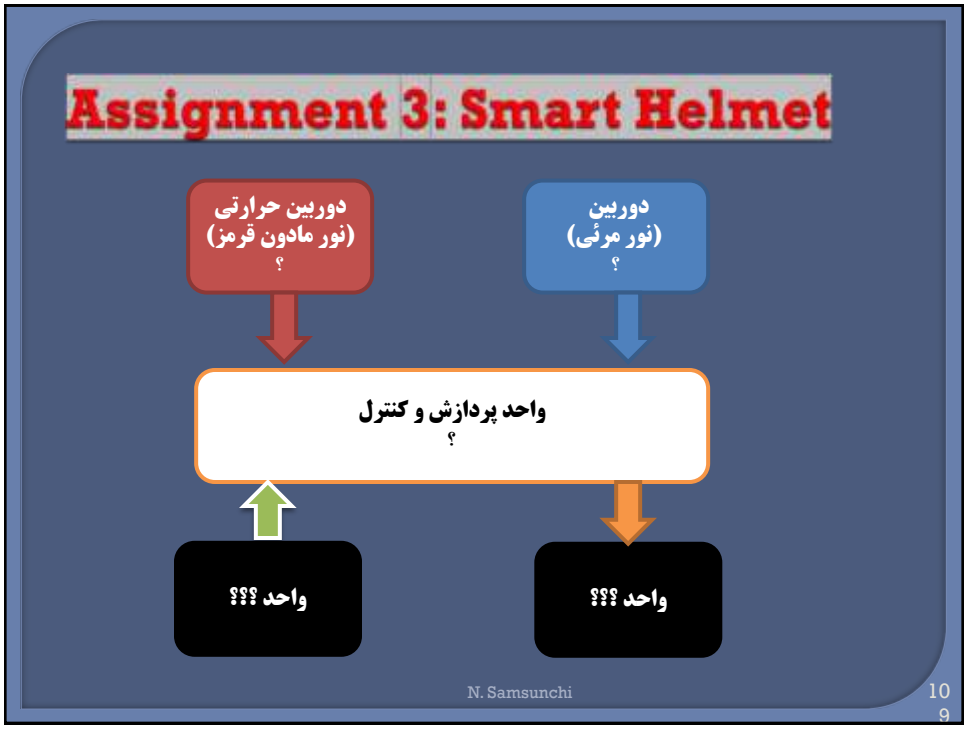
- Specifications:
- Max weight: 1 Kg
- Power: Battery.
- Working Time:5h
- Standby Time: 24h

- Price <1000$
- ….

Assignment 3: Smart Helmet

- A. Block diagram Design : Blocks and their interconnections.

- B. Design implementation : Which ? Why?


Assignment 3: Smart Helmet
دوربین حرارتی
(نور مادون قرمز)
؟
دوربین
(نور مرئی)
؟
واحد پردازش و کنترل
؟
واحد ؟؟؟
واحد ؟؟؟
N. Samsunchi
10
9

Intel 8086 Assembly Language

Nader Samsunchi

1

Intel 8086 Microprocessor

2

8086 Microprocessor pin definitions

Common Signals

Name	Function	Type
AD15–AD0	Address/Data Bus	Bidirectional, 3-State
A19/S6–A16/S3	Address/Status	Output, 3-State
BHE/S7	Bus High Enable/ Status	Output, 3-State
MN/MX	Minimum/Maximum Mode Control	Input
RD	Read Control	Output, 3-State
TEST	Wait On Test Control	Input
READY	Wait State Control	Input
RESET	System Reset	Input
NMI	Non-Maskable Interrupt Request	Input
INTR	Interrupt Request	Input
CLK	System Clock	Input
Vcc	+5V	Input
GND	Ground	

Minimum Mode Signals (MN/MX = Vcc)

Name	Function	Type
HOLD	Hold Request	Input
HLDA	Hold Acknowledge	Output
WR	Write Control	Output, 3-State
M/IO	Memory/IO Control	Output, 3-State
DT/R	Data Transmit/ Receive	Output, 3-State
DEN	Data Enable	Output, 3-State
ALE	Address Latch Enable	Output
INTA	Interrupt Acknowledge	Output

Maximum Mode Signals (MN/MX = GND)

Name	Function	Type
RQ/GT1, 0	Request/Grant Bus Access Control	Bidirectional
LOCK	Bus Priority Lock Control	Output, 3-State
S2–S0	Bus Cycle Status	Output, 3-State
QS1, QS0	Instruction Queue Status	Output

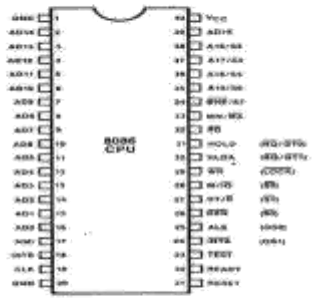
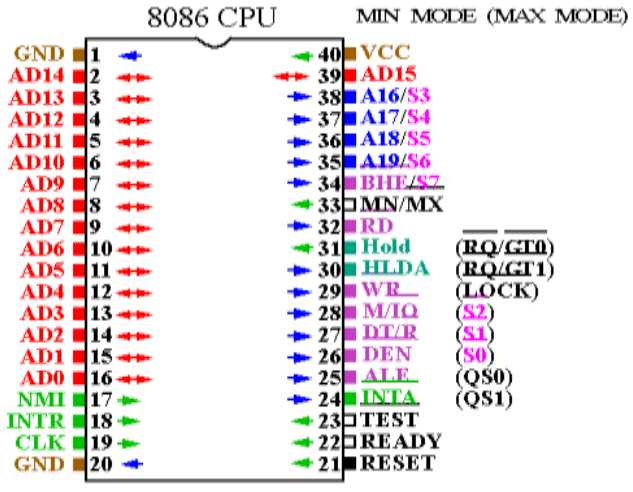


Figure 4-1. 8086 Pin Definitions

3

8086 Microprocessor pin definitions



4

8086 Microprocessor Internal block diagram

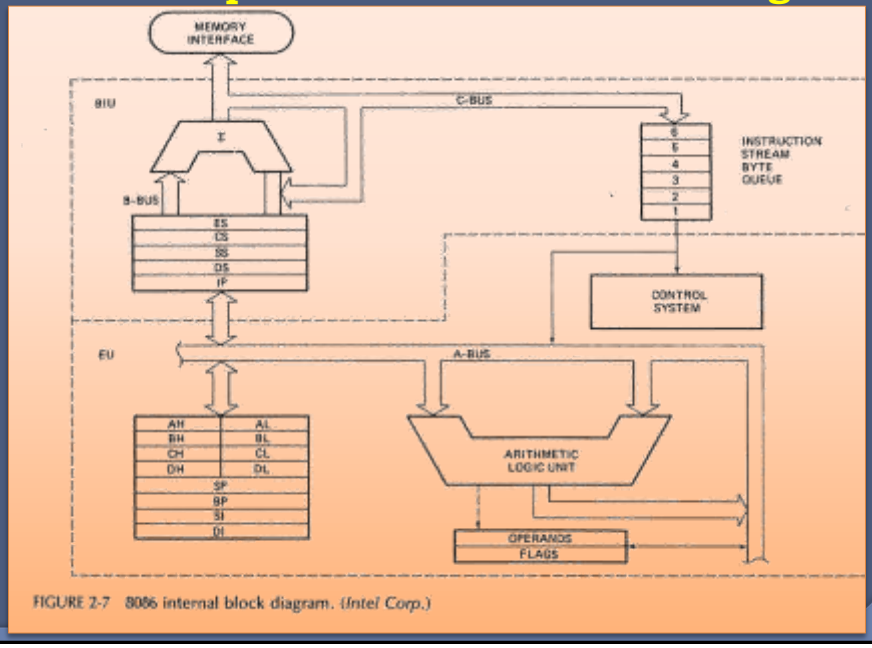


FIGURE 2-7 8086 internal block diagram. (*Intel Corp.*)

5

8086 ISA

6

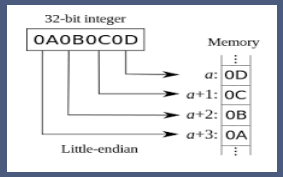
Basic Terminology

- Bit (Binary Digit – 0 , 1)
- Nibble (4 Bits)
- Byte (8 Bits , 2 Nibbles)
- Word (16 Bits, 2 Bytes)
- Double Word (2 Words,32 Bits,4 Bytes)
- Quad Word(4 Words,64 Bits,8 Bytes)
- Kilo Byte (2^10 Bytes : 1024 Bytes)
- Mega Byte (2^20 Bytes :1,048,576 Bytes)
- Giga Byte (2^30 Bytes :1,073,741,824 Bytes)

7

How to store Word-sized Data?

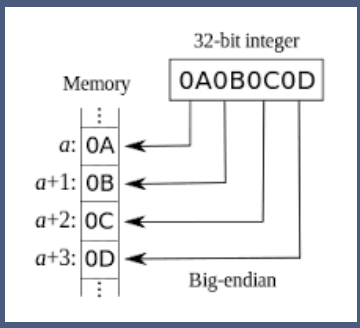
- A word (16-bits) is formed with two bytes of data.
- Method 1: The least significant byte always stored in the lowest-numbered memory location.Most significant byte is stored in the highest.
- This method of storing a number is called the **little endian** format.



8

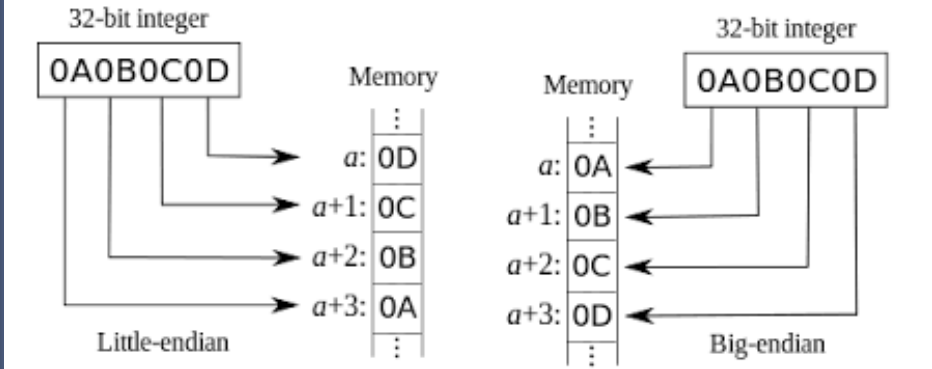
How to store Word-sized Data?...

- Method 2: Alternate method is called the **big endian** format.
- Numbers are stored with the lowest location containing the most significant data.



9

How to store Word-sized Data?...



10

How to store Word-sized Data?...

- Intel x86 processors use little-endian. also Zilog Z80 (including Z180 and eZ80)
- Motorola 68000 series , Xilinx, IBM z/Architecture, Atmel AVR32 are Big-endian. Also the Internet protocol suite, such as IPv4, IPv6, TCP, and UDP

11

How to store Word-sized Data?...

- Method 3: ARM versions 3 and above, PowerPC, Alpha, SPARC V9, MIPS, and Intel IA-64 : Bi-endian
- Hardware/Software switchable endianness in data fetches and stores, instruction fetches, or both.

12

Real Mode /Protected Mode

- 8086/8088 mode of operation is known as Real Mode Operation.
  - 8086: 20 address lines => 1MB addressable Memory
- 80286 and above operate in either the real or protected mode.

13

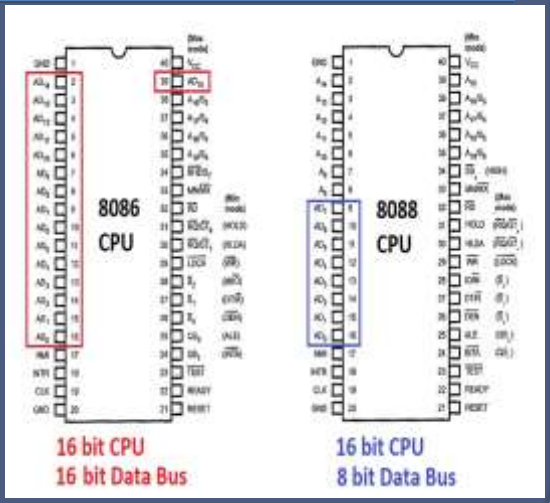
Real Mode /Protected Mode

- **Real mode operation** allows addressing of only the first 1M Byte of memory space—even in Pentium 4 or Core2 microprocessor.
  - the first 1M byte of memory is called the **real memory**, **conventional memory**, or **DOS memory**
- Real Mode operation is for binary compatibility.
- Compatibility:
  - Source code
  - Binary

14

8086 Registers

- Registers are in the CPU and are referred to by specific names
- 14 Registers (all 16 Bits)
- General Purpose (Data) Registers
- Address Registers
- Status and Control Registers



15

General Purpose (Data) registers

- Hold data for an operation to be performed
- Instructions execute faster if the data is in a register
- AX, BX, CX, DX are the data registers
- Low and High bytes of the data registers can be accessed separately
  - AH, BH, CH, DH are the high bytes (8 bits)
  - AL, BL, CL, and DL are the low bytes (8 bits)
- Data Registers are general purpose registers but they also perform special functions

16

General Purpose (Data) registers

- AX
  - Accumulator Register
  - Preferred register to use in arithmetic, logic and data transfer instructions
  - Must also be used in I/O operations
- BX
  - Base Register
  - Also serves as an address register

17

General Purpose (Data) registers

- CX
  - Count register
  - Used as a loop counter
  - Used in shift and rotate operations
- DX
  - Data register
  - Used in multiplication and division
  - Also used in I/O operations

18

Address Registers

- Hold the address of an instruction or data element
- Segment registers (CS, DS, ES, SS)
- Pointer registers (SP, BP)
- Index registers (SI, DI)

19

Status and Control Registers

- IP
  - Instruction Pointer
- Flags Register
  - Keeps the current status of the processor
  - Control CPU operation.

20

Flags Register

Overflow

Direction

Interrupt enable

Trap

Sign

Zero

Auxiliary flag

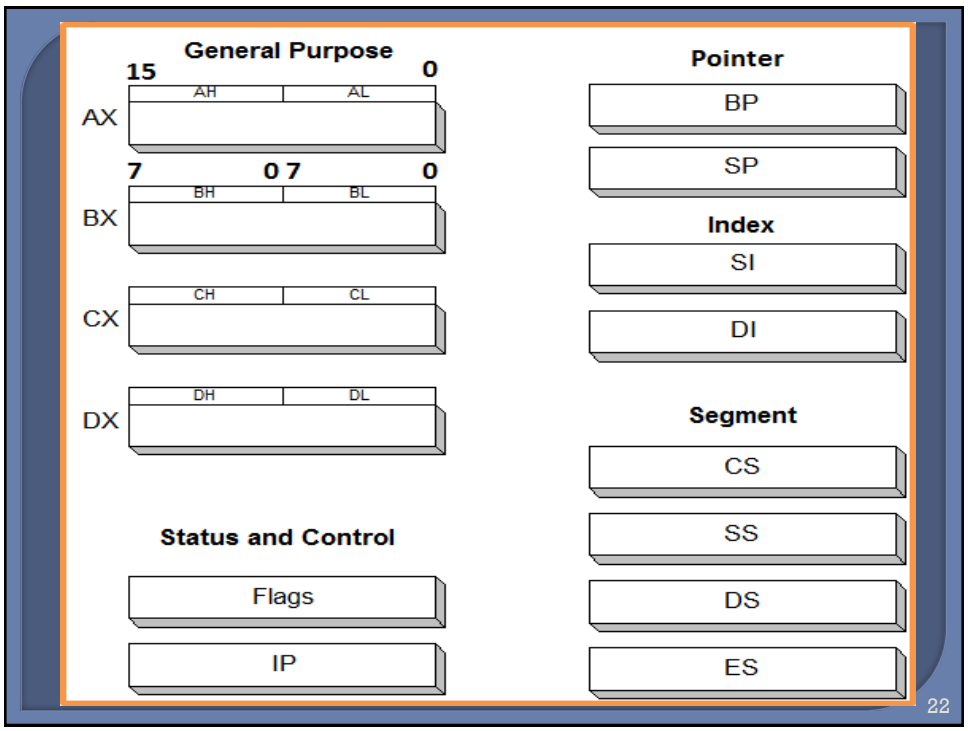
Parity flag

Carry flag

6 are status flags

3 are control flag

21



22

Segment Registers :CS, DS, SS,ES

- Are Address registers, Store the memory addresses of instructions and data
- Memory Organization
  - Each byte in memory has a 20 bit address starting with 0 to (2^20)-1 or 1 meg of addressable memory
  - Addresses are expressed as 5 hex digits from 00000 - FFFFF
  - Problem: But 20 bit addresses are TOO BIG to fit in 16 bit registers!
  - Solution: Segmented Memory

23

Segmented Memory

Segmented memory addressing: Physical (absolute , linear) address is a combination of a 16-bit segment value and a 16-bit offset value.

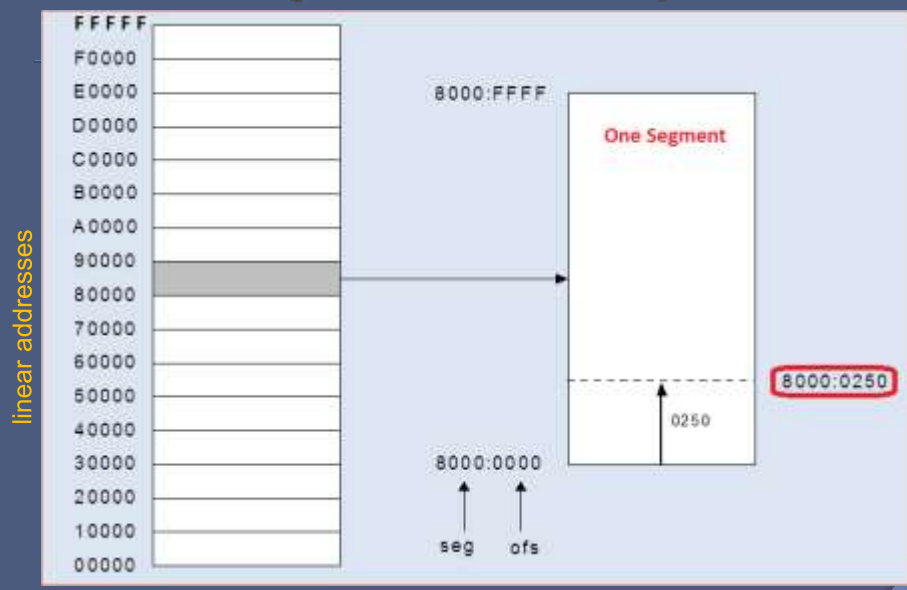
24

Segmented Memory

- Logical Address → Segment:Offset
  - Segment numbers range from 0000 to FFFF
  - Within a segment, a particular memory location is specified with an offset
  - An offset also ranges from 0000 to FFFF

25

Segmented Memory



26

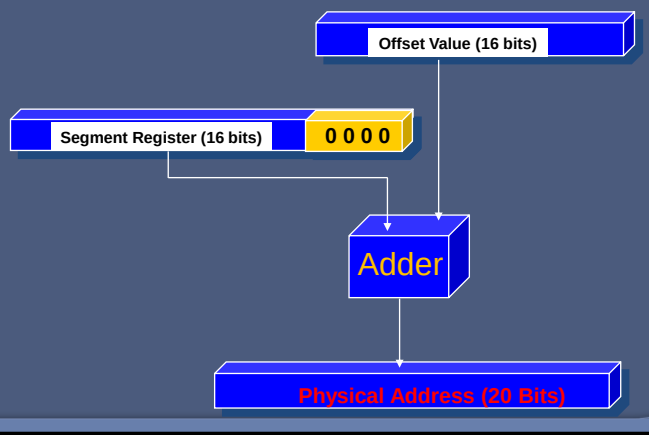
Segment Registers

- A Segment Register contains the Starting location of a segment.
- CS : Code Segment
- DS : Data Segment
- ES : Extra Segment
- SS : Stack Segment

27

Memory Address Generation

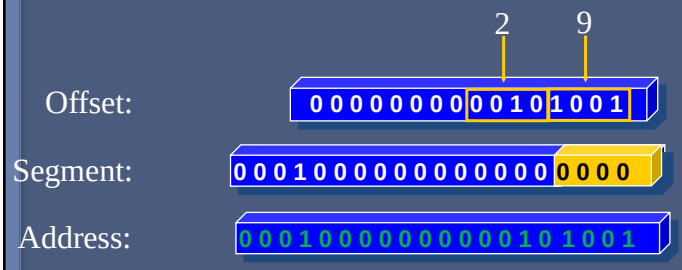
- The CPU has a dedicated adder for determining physical memory addresses



28

Address Calculation : Example 1

- If DS = 1000h and Offset =29h,
- Where is the actual data?



29

Address Calculation : Example 2

- The physical address of the logical address A4FB:4872 is

A4FB0 h

+ 4872 h =

A9822 h

30

Address Calculation : Example 3

- If DS=7FA2 , Offset=438E
  - a) Calculate Physical Address.
  - b) Calculate the lower range of the data segment.
  - c) Calculate the upper range of the data segment.
  - d) Show the logical address.

31

Example

•If DS=7FA2H and the offset is 438EH

a) Calculate the physical address

7FA20 + 438E = 83DAE

b) calculate the lower range

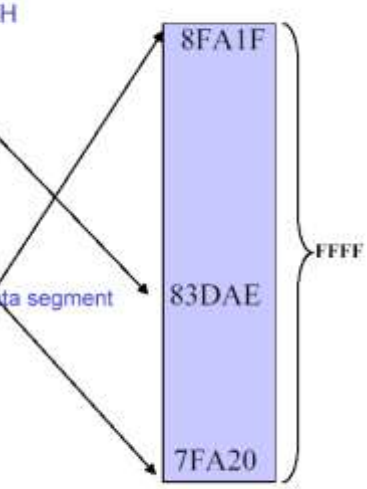
7FA20 + 0000 = 7FA20

c) Calculate the upper range of the data segment

7FA20 + FFFF = 8FA1F

d) Show the logical Address

7FA2:438E



32

Address Calculation : Example 4

What segment addresses correspond to the linear address 28F30h?

Many different segment-offset addresses .For example:

28F0:0030, 28F3:0000, 28B0:0430, . . .

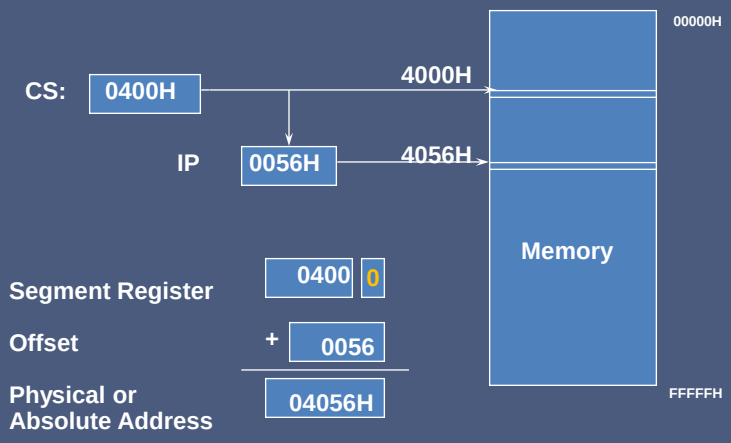
33

The Code Segment (CS)

- **Instructions are always fetched with using the CS register.**
- **The offset is given by the IP for the Code Segment.**
- **CS:IP**

34

Example 1: CS:IP=0400:0056



35

The Data Segment (DS)

- **Data is usually fetched with respect to the DS register.**

  DS:data address

36

The Extra Segment (ES)

- **ES is like DS. It is usually used for string operations.**

  ES:data address

37

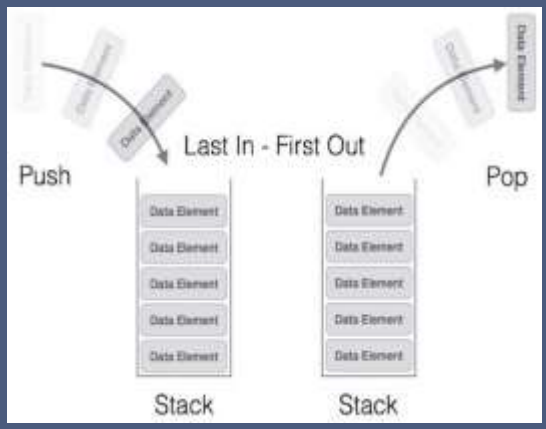
The Stack Segment (SS)

- **The stack is always referenced with respect to the SS register.**
- **The offset is given by the SP register.**
- **SS:SP**
- **The stack usually used for storage of temporary data.**

38

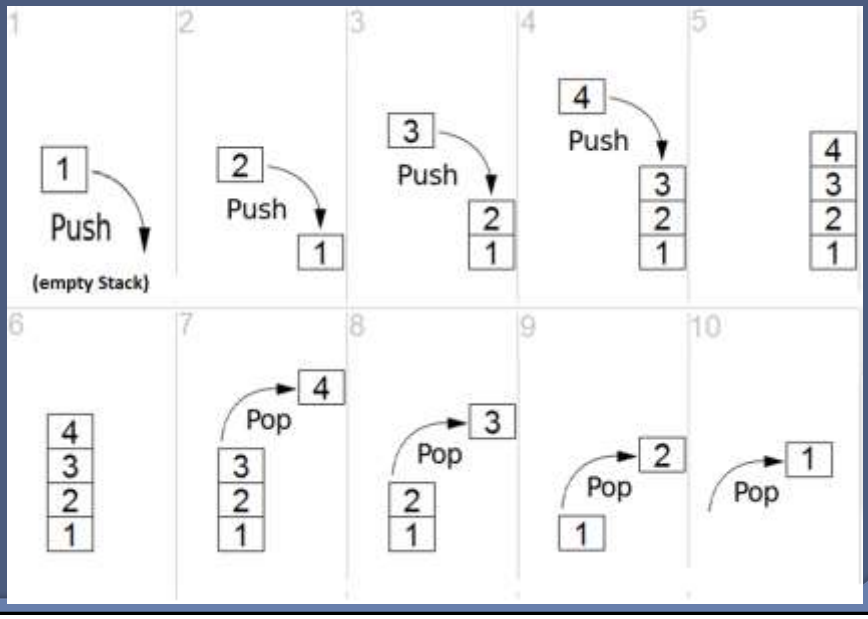
Stack

- LIFO : Last In – First Out



39

Stack



40

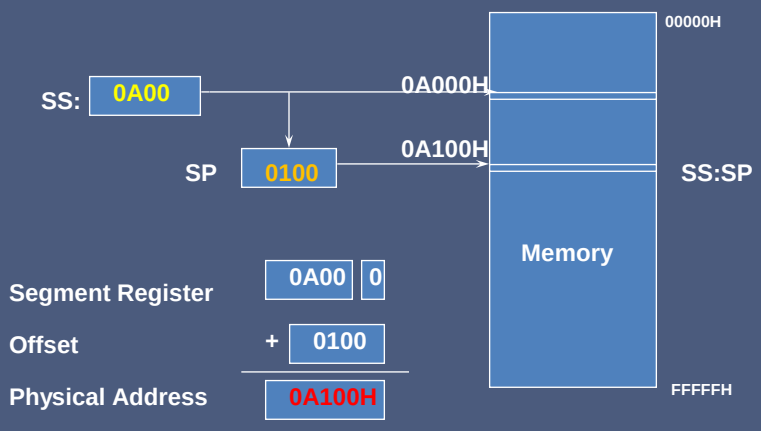
Stack in X86 Architecture

- The stack grows toward decreasing memory locations(Lower Addresses).
- The SP points to the last or top item on the stack.

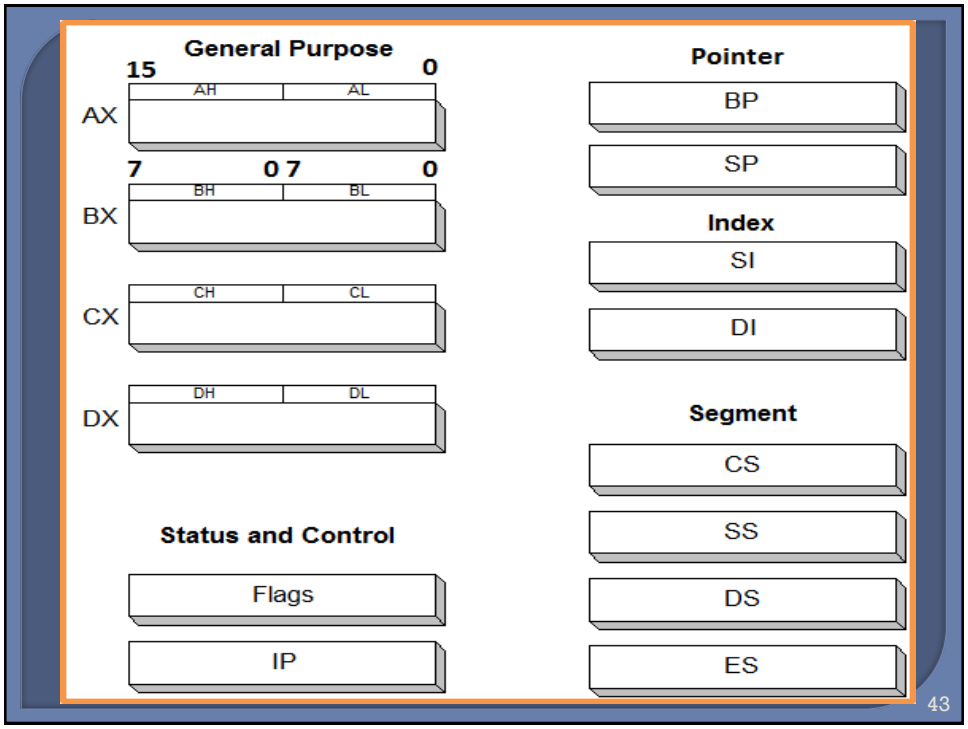
- PUSH : Decrement the SP
- POP : Increment the SP

41

Example : SS:SP



42



Pointer Registers

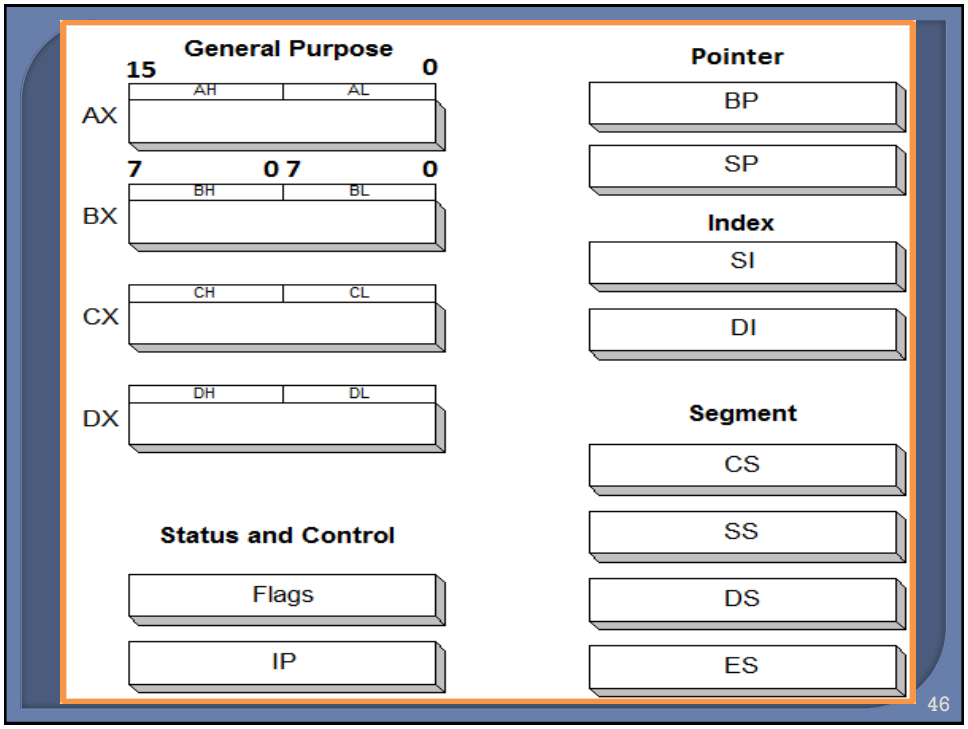
- Contain the offset addresses of memory locations
- SP (Stack pointer)
  - Used with SS to access the stack.
  - SS:SP
- BP (Base Pointer)
  - Primarily used to access data on the stack.

44

Index Registers

- SI (Source Index register)
  - is required for some string operations
  - SI is associated with the DS. DS:SI
- DI (Destination Index register)
  - is also required for some string operations.
  - DI is associated with the ES. ES:DI
- The SI and the DI registers may also be used to access data stored in arrays

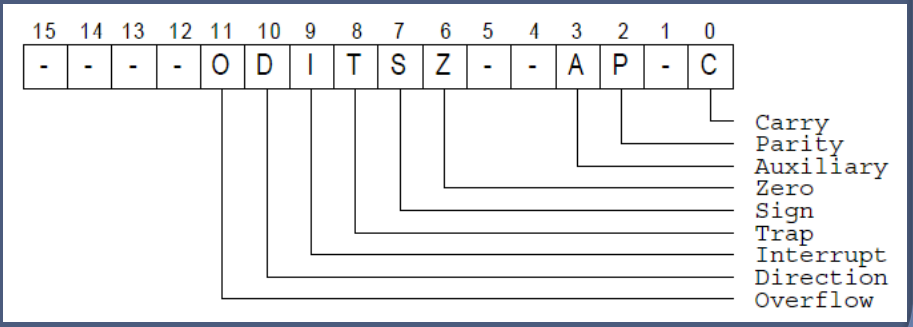
45



46

Flags Register

- 6 are status flags. (important : C ,Z, S , O)
- 3 are control flag. (important : I)



47

C: Carry flag

- Carry flag will be set whenever there is a carry.
- Example : AL=77h.
- AL + 50h = C7h. → C=0
- AL + 50h = 17h. → C=1

48

Z: Zero Flag

- The zero flag will be set (Z=1) whenever the result is zero.
- Example :
- AL=10h.
- AL + F0h = 00h , → Z=1

49

S: Sign flag

- Sign flag will be set whenever the result is negative. S shows MSB.
- Example :
- AL= -20=ECh=1110 1100b
- +5=05h=0000 0101b
- AL + 05h = -15=F1h=1111 0001b → S=1

50

S: Sign flag …

- Example 2:
- AL=+119=77h=0111 0111b
- +80=50h=0101 0000b
- AL + 50h = C7h=1100 0111b → S=1
- C7h=199 >127
- How to detect this error?

51

O: Overflow flag

- Overflow flag will be set whenever the result of signed operations is overflow.
- Example :
- AL range=(-128,+127)=(80h,7F)
- AL= 119=77h.
- AL + 50h = C7h=199 > (+127) → O = 1

52

I : Interrupt flag

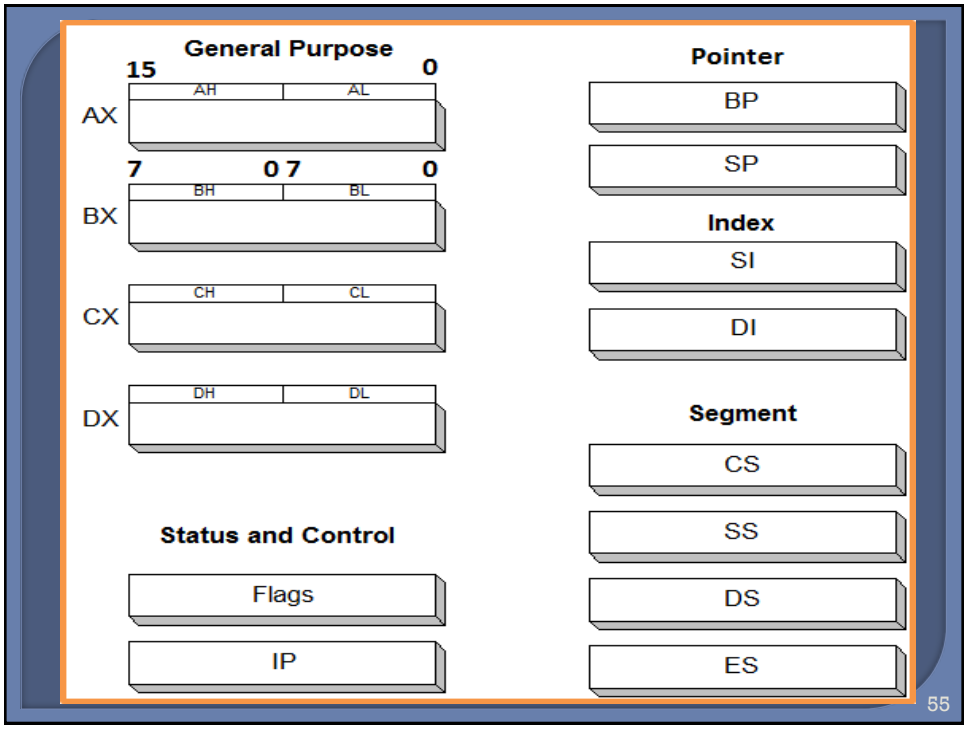
- I= 0 → Disable hardware Interrupt.
- I=1 → Enable hardware Interrupt.

53

More flags

- A: Auxiliary flag contains carry out of bit 3 into bit 4 (Lower nibble to higher nibble) for specialized arithmetic.
- P:Parity flag will be set whenever the number of bit "1" are even.
- D: Direction flag is used to specify direction (increment/decrement index register) in string operation.
- T: Trap flag is used to interrupt CPU after each operation.

54



8086 Addressing Modes

1. Implied
2. Immediate
3. Register
4. Direct
5. Register Indirect
6. Direct Indexed
7. Base Indexed
8. Base Relative

56

1. Implied Addressing

- No need to address or address is implied in instruction.
- Examples : (No need to Address)
- HLT HALT the CPU
- NOP No Operation!

- Examples : (Address Implied)

57

Flag set/reset instructions

- Carry flag
  - STC C=1
  - CLC C=0
  - CMC C → C' (Complement Carry)
- Direction flag
  - STD D=1
  - CLD D=0
- Interrupt flag
  - STI I=1
  - CLI I=0

58

Flag Transfer instructions

- Load AH with Flags. AH← Flags
  - LAHF
- Store AH to Flags. Flags←AH
  - SAHF
- Push Flags to Stack.
  - PUSHF
- Pop Flags from Stack.
  - POPF

59

2. Immediate Addressing

- An 8 Bit or 16 Bit constant is in instruction.
- Examples :
- MOV AL,-30 AL← - 30
- MOV CX,500 CX← 500

60

3. Register Addressing

- Both the operands are registers
- Example :
- MOV BL,AL BL← AL
- MOV DS,AX DS←AX

61

MOV Instruction

- MOV destination , source
- Copies source to destination.
- destination=source , source unchanged

62

4. Direct Addressing

- The memory address is directly given in the instruction
- Example :
- MOV AX,[0200h]
- AX ← value stored in memory location DS:0200

63

5. Register Indirect Addressing

- The memory address is in a register.
- Example :
- MOV AX,[BX]
- AX ← value stored at memory address contained in DS:BX

- Only BX,BP,SI,DI can be used !
- If register is SI, DI and BX then DS is by default segment register.
- If BP is used, then SS is by default segment register.

64

6. Direct Indexed Addressing

- Uses an index register(DI or SI)
- Useful for accessing elements in a table(array).

- Example : we have a TABLE1 array.
- MOV DI,2
- MOV AL,TABLE1[DI]
- AL ← The 3rd element of TABLE1

- Table elements start with 0.

65

How to define TABLE1 ?

- Use DB directive (Define Byte).
- Directives are commands to the assembler ,rather than to the CPU.

- Example1 : define TABLE1=(1,2,3,4).
- TABLE1 DB 1,2,3,4

- Example2 : define TABLE1=(1,1,1,1).
- TABLE1 DB 4 DUP(1)

66

7. Base Indexed Addressing

- The operand address is calculated as base register(BX) plus an index register (DI or SI).
- Useful for accessing elements in a 2 dimensional table(array).

- Example :
- MOV BX,2
- MOV DI,1
- MOV AL,[BX][DI] (or MOV AL,[BX+DI])
- AL ← value stored in memory location DS:2+1

67

8. Base Relative Addressing

- The operand address is calculated using one of the base registers (BX or BP)and an 8 bit or a 16 bit displacement.

- Example :
- MOV CL,[BX+04H]
- CL ← DS: [BX + 04H]

68

Question

- MOV [7000H],CL

69

Instruction Set Classification

- Data Transfer
- Arithmetic
- Logical
- Control and Branch
- String

70

Data transfer instructions

IN	**Input byte or word from port**
LAHF	**Load AH from flags**
MOV	**Move to/from register/memory**
OUT	**Output byte or word to port**
POP	**Pop word off stack**
POPF	**Pop flags off stack**
PUSH	**Push word onto stack**
PUSHF	**Push flags onto stack**
SAHF	**Store AH into flags**
XCHG	**Exchange byte or word**

71

Data transfer : MOV

- MOV Dest, Src
  - `MOV reg, reg` reg ← reg
  - `MOV reg, mem` reg ← mem
  - `MOV mem, reg` mem ← reg
  - `MOV reg, imm` reg ← imm
  - `MOV mem, imm` mem ← imm

72

MOV limitations

- Both operand must be in the same size.
- There is no instruction to put immediate value directly to a segment register. Have to use accumulator (AX) to accomplish this.
- There is no move mem← mem instruction. Have to use general registers to do this.

73

MOV Examples

```
MOV AX,1234h
MOV DX,5678h
MOV AL,DL
MOV BH,DH
MOV DX,BX
MOV [100h],AX
MOV BX,[100h]

MOV AX,2300h
MOV DS,AX
```

74

MOV Examples

```
MOV AX,1000h
```

AX	AH	AL
1000h	10h	00h

```
MOV AL,3Ah
```

AX	AH	AL
103Ah	10h	3Ah

```
MOV AH,AL
```

AX	AH	AL
3A3Ah	3Ah	3Ah

```
MOV AX,234h
```

AX	AH	AL
234h	02h	34h

75

MOV Examples

```
MOV   AX,6789h
MOV   DX,1234h
MOV   [100h],AX
MOV   [102h],DX
MOV   [104h],AH
MOV   [105h],DL
MOV   BX,[104h]
MOV   CX,[103h]
MOV   [106h],CL
```

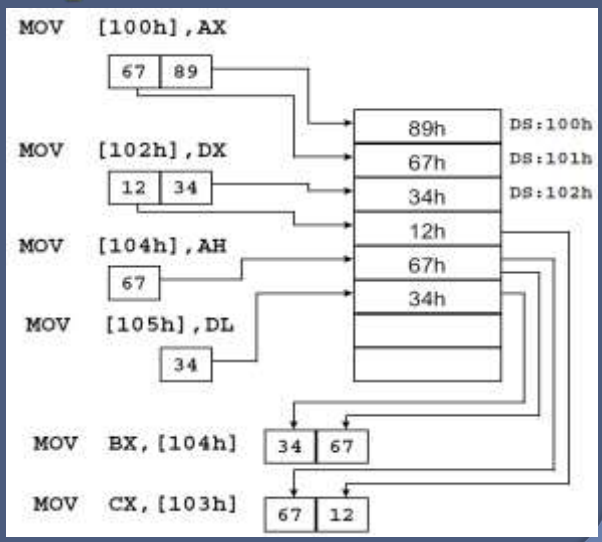
DS:100h	
DS:101h	
DS:102h	

- Given only offset where to put value, it will be automatically select DS as the segment register.

76

MOV Examples

```
MOV   AX,6789h
MOV   DX,1234h
MOV   [100h],AX
MOV   [102h],DX
MOV   [104h],AH
MOV   [105h],DL
MOV   BX,[104h]
MOV   CX,[103h]
MOV   [106h],CL
```



77

MOV Examples

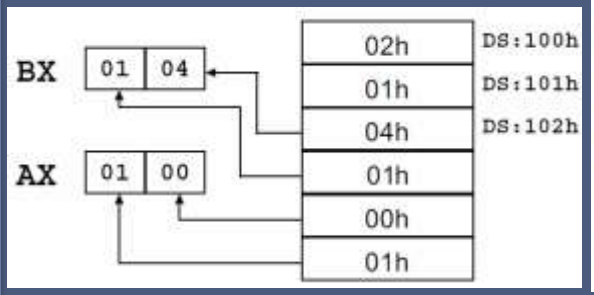
- AX = ?

```
MOV   AX,102h
MOV   BX,100h
MOV   CX,4004h
MOV   DX,1201h
MOV   [BX],AX
MOV   [BX+2],CX
MOV   [BX+3],DX
MOV   [BX+4],BX
MOV   BX,[102h]
MOV   AX,[BX]
```

78

MOV Examples

- AX = 0100h




```
MOV   AX,102h
MOV   BX,100h
MOV   CX,4004h
MOV   DX,1201h
MOV    [BX],AX
MOV    [BX+2],CX
MOV    [BX+3],DX
MOV    [BX+4],BX
MOV   BX,[102h]
MOV   AX,[BX]
```

79

MOV Examples

- MOV [100h] , 10h
- Address 100h = 10h
- What about address 101h?
- Word or Byte? To put immediate value directly to memory, we have to specify its size. (Byte/Word PTR)
  - `MOV BYTE PTR [100h], 10h`
  - `MOV WORD PTR [100h], 10h`

80

eXCHanGe

- XCHG *target, source*
  - reg, reg
  - reg, mem
  - mem, reg
  - Examples:
  - XCHG AH,BL
  - XCHG CX,DX
- This provides an efficient means to swap the operands
  - No temporary storage is needed
  - Sorting often requires this type of operation
  - This works only with the general registers
  - XCHG cannot perform memory to memory moves

81

PUSH , POP

- **PUSH source(16 bit)**
- **Example: PUSH AX**
- **POP destination(16 bit)**
- **Example: POP DX**

82

INPUT/OUTPUT(I/O)

- I/O devices(such as Keyboard, speaker,modem,…) allow the CPU to communicate with the outside world.
- The 8086 communicates with I/O devices through special circuits callled ports.

83

INPUT/OUTPUT(I/O)

- Each port identifies by a number.I/O port number is similar to a memory address
- There are 65,536(64K) I/O ports available.
- Circuit design identifies Port numbers.

84

IBM PC I/O Space

- The area below I/O location 0400H is considered reserved for system devices.
  - Examples:
  - 60H Keyboard.
  - 61H Buzzer.
  - 1F0H Master HDD Controller.
  - 201H Joystick.
  - 3F0H Floppy Disk Controller.
- Area available for expansion by plug-in cards (such as Modems,Network Cards) extends from I/O port 0400H through FFFFH.

85

INput

```
IN AX,PortNumber   (16 Bit)
IN AL,PortNumber   (8  Bit)

PortNumber is between 0 and 255,
refers to device address.
For addressing 64K Ports,Use:

MOV DX, PortNumber
IN AX, DX
IN AL, DX
```

86

OUTput

- OUT PortNumber,AX (16 Bit)
- OUT PortNumber,AL (8 Bit)

- PortNumber is between 0 and 255, refers to device address.
- For addressing 64K Ports,Use:

- MOV DX, PortNumber
- OUT DX,AX
- OUT DX,AL

87

Data transfer instructions

IN	Input byte or word from port
LAHF	Load AH from flags
MOV	Move to/from register/memory
OUT	Output byte or word to port
POP	Pop word off stack
POPF	Pop flags off stack
PUSH	Push word onto stack
PUSHF	Push flags onto stack
SAHF	Store AH into flags
XCHG	Exchange byte or word

88

Arithmetic instructions

INC	**Increment byte or word by one**
DEC	**Decrement byte or word by one**
NEG	**Compute 2's complement of byte or word**
ADD	**Add byte or word**
ADC	**Add byte or word plus carry**
SUB	**Subtract byte or word**
SBB	**Subtract byte or word and carry (borrow)**
MUL	**Multiply byte or word (unsigned)**
IMUL	**Multiply byte or word (Signed)**
DIV	**Divide byte or word(unsigned)**
IDIV	**Divide byte or word(Signed)**
CBW	**Convert byte to word**
CWD	**Convert word to double-word**
CMP	**Compare byte or word**

89

Increment (+1), Decrement(-1)

- INC / DEC
  - INC register DEC register
  - INC memory DEC memory
- Examples:
  - INC AX AX ← AX+1
  - DEC BL BL ← BL - 1
  - INC [100h] [100h]← [100h]+1

90

Negation

- Compute 2's complement.
- NEG reg
- NEG mem

91

Example

```
MOV CX, 10h    ; CX = 0010h
NEG CX         ; CX = FFF0h
MOV AX,0FFFFh  ; AX = FFFFh
NEG AX         ; AX = 1
MOV BL,1       ; BL = 1
NEG BL         ; BL = FFh
```

92

ADD

- ADD destination,source
- destination←destination+source

- ADD reg, imm
- ADD reg, mem
- ADD reg, reg
- ADD mem, imm
- ADD mem, reg

93

Add plus Carry(ADC)

- ADC destination,source
- destination←destination+source+carry

- ADC reg, imm
- ADC reg, mem
- ADC reg, reg
- ADC mem, imm
- ADC mem, reg

94

Example :

```
MOV AL, 10h        ;AL =   10h
ADD AL, 20h        ;AL =   30h
MOV BX, 200h       ;BX = 0200h
MOV AH, 89h        ;AX = 8930h
ADD AX, 9876h      ;AX = 21A6h,
                   ;Carry=1
ADC BX, 01h        ;BX = ?
```

BX = 0202h

95

Subtract

- SUB destination,source
  - destination← destination-source

- SUB reg, imm
- SUB reg, mem
- SUB reg, reg
- SUB mem, imm
- SUB mem, reg

96

Subtract with borrow

- SBB destination,source
  - destination← destination-source-carry

- SBB reg, imm
- SBB reg, mem
- SBB reg, reg
- SBB mem, imm
- SBB mem, reg

97

Example

```
MOV AL, 30h        ;AL =   30h
SUB AL, 20h        ;AL =   10h
MOV BX, 9876h      ;BX = 9876h
MOV AX, 8930h      ;AX = 8930h
ADD AX,BX          ;AX = 21A6h
                    ; Carry=1
SBB BX, 0001h      ;BX = ?
                   ;BX=9874h
SBB BX, 0001h      ;BX = ?
                   ;BX=9873h
```

98

Multiplication

- MUL (Multiply unsigned)
  - MUL reg
  - MUL mem
- IMUL (Multiply signed)
  - IMUL reg
  - IMUL mem
- Always perform with accumulator(AX).

99

8 bit multiplication

- AL is multiplicand
- AX keep the result
- Example :

```
MOV AL,15        ; AL = 0Fh
MOV CL,-10       ; CL = F6h
IMUL CL          ; AX = FF6Ah= -150D

MOV AL,15        ; AL = 0Fh
MOV CL,-10       ; CL = F6h=246D
MUL CL           ; AX = 0E6Ah=3690D
```

100

16 bit multiplication

- AX is multiplicand
- DX:AX keep the result
- Example :
- MOV AX,0100h ; AX = 0100h
- MOV BX,1234h ; BX = 1234h
- MUL BX ; DX = 0012h
  ; AX = 3400h

101

Division

- DIV (Unsigned division)
  - DIV reg
  - DIV mem
- IDIV (Signed division)
  - IDIV reg
  - IDIV mem
- Always perform with accumulator.

102

8 bit division

- AL is dividend
- AL keep the result
- AH keep the remainder

- Example :
- MOV AL, 23 ; AL = 17h = 23D
- MOV BL, -16 ; BL = F0h = -16D
- IDIV BL ; AL = FFh = -1D
- ; AH = 07h

103

16 bit division

- DX:AX dividend.
- AX keep the result, DX keep the remainder.

- Example:
- MOV AX,86A0h ;AX=86A0h
- MOV DX,0001h ; DX=0001h
- ; DX:AX=0001 86A0h=100 000D
- MOV CX,0EA60h ; CX=EA60h=60 000D
- DIV CX ;AX = 0001h
- ;DX = 9C40h=40 000D

104

Data Conversion

- Convert Byte to Word : CBW
  - Signed convert AL -> AX
- Convert Word to Double word : CWD
  - Signed convert AX -> DX:AX

105

Example

```
MOV AL,22h      ; AL=22h
CBW             ; AX=0022h

MOV AL,0F0h     ; AL=F0h=-16D
CBW             ; AX=FFF0h

MOV AX, 3422h   ; AX=3422h
CWD             ; DX=0000h
                ; AX=3422h
```

106

Compare

- CMP destination,source
- CMP reg, imm
- CMP reg, mem
- CMP reg, reg
- CMP mem,imm
- CMP mem, reg

- CMP dose not changes values of source or destination.
- CMP affects flags.

107

Examples:

```
MOV CX, 10h    ;CX=10h
CMP CX, 20h    ;S flag=1

MOV BX, 40h    ;BX=40h
CMP BX, 40h    ;Z flag=1

MOV AX, 30h    ;AX=30h
CMP AX, 20h    ;S flag=0
```

108

Flag affection

Instruction	Flag affected			
	Z-flag	C-flag	S-flag	O-flag
ADD	*yes*	*yes*	*yes*	*yes*
ADC	*yes*	*yes*	*yes*	*yes*
SUB	*yes*	*yes*	*yes*	*yes*
SBB	*yes*	*yes*	*yes*	*yes*
INC	*yes*	***no***	*yes*	*yes*
DEC	*yes*	***no***	*yes*	*yes*
NEG	*yes*	*yes*	*yes*	*yes*
CMP	*yes*	*yes*	*yes*	*yes*
MUL	*no*	*yes*	*no*	*yes*
IMUL	*no*	*yes*	*no*	*yes*
DIV	*no*	*no*	*no*	*no*
IDIV	*no*	*no*	*no*	*no*
CBW	*no*	*no*	*no*	*no*
CWD	*no*	*no*	*no*	*no*

109

Logical instructions

NOT Logical NOT of byte or word
AND Logical AND of byte or word
OR Logical OR of byte or word
XOR Logical exclusive-OR of byte or word

SHL Logical shift left byte or word
SHR Logical shift right byte or word

SAL Arithmetic shift left byte or word
SAR Arithmetic shift right byte or word

ROL Rotate left byte or word
ROR Rotate right byte or word

RCL Rotate left trough carry byte or word
RCR Rotate right trough carry byte or word

TEST Test byte or word

110

NOT

- NOT destination
- NOT reg
- NOT mem
- Flags are unaffected.
- Example :
- MOV AX,0FFFFh ;AX=FFFFh
- NOT AL ;AX=FF00h
- NOT AX ;AX=00FFh
- NOT WORD PTR [0120h]

111

AND,OR,XOR

- AND destination,source
- AND reg, imm
- AND reg, mem
- AND reg, reg
- AND mem, imm
- AND mem, reg

- Flags Affection: CF=0, OF=0,
- ZF,SF,PF changes.

112

Application : Masking

- Selective Set
- Selective Reset
- Selective Complement

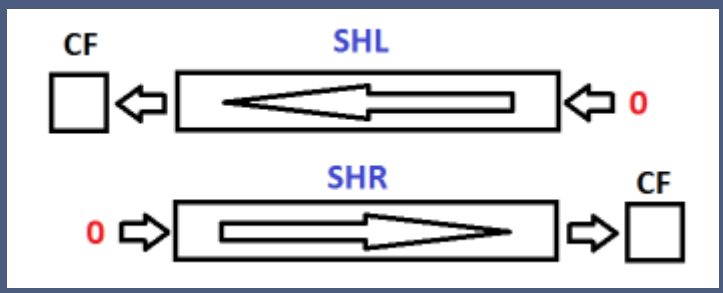
- Example:

```
MOV AL,11110000B      ;AL=11110000B
MOV BL,00001010B      ;BL=00001010B
OR  AL,BL             ;AL=11111010B
MOV BL,00111111B      ;BL=00111111B
AND AL,BL             ;AL=00111010B
MOV BL,00000011B      ;BL=00000011B
XOR AL,BL             ;AL=00111001B
```

113

SHL,SHR

- SHL destination,count
- SHR destination,count
- Destination : reg, mem
- Count : 1, CL



114

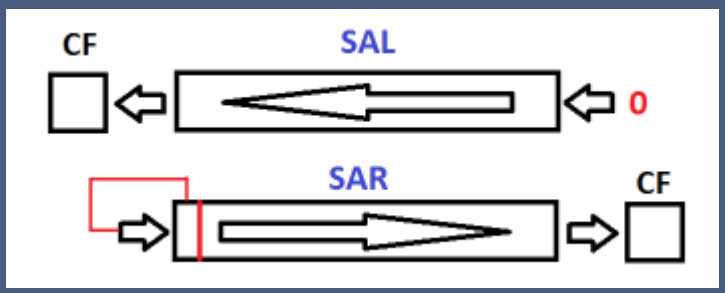
Examples:

```
MOV AL,10001000B  ;AL=10001000B
SHL AL,1          ;AL=00010000B
                   ;CF=1
MOV CL,4
SHR AL,CL         ;AL=00000001B
                  ;CF=0
```

115

SAL,SAR

- SAL destination,count
- SAR destination,count
- Destination : reg, mem
- Count : 1, CL



116

Examples:

```
MOV AL,2        ;AL=2
SAL AL,1        ;AL=4 , CF=0

MOV AL,-2       ;AL=-2=FEh
SAL AL,1        ;AL=-4=FCh , CF=1
```

- SAL : x 2 (Doubling value)

117

Examples:

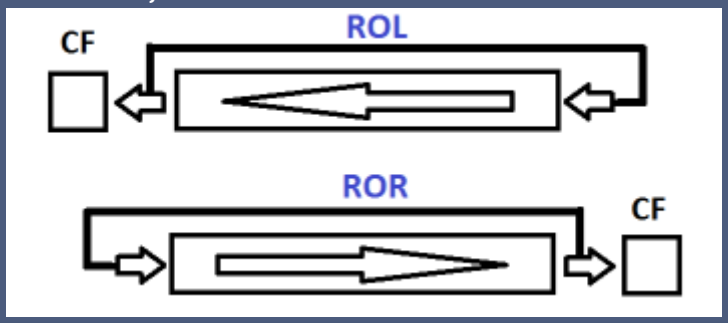
```
MOV AL,8        ;AL=8
MOV CL,2
SAR AL,CL       ;AL=2 , CF=0

MOV AL,-4       ;AL=-4=11111100B
SAR AL,1        ;AL=-2=FEh , CF=0

MOV AL,5        ;AL=5=00000101B
SAR AL,1        ;AL=2=00000010B
                ;CF=1
```

- SAR : / 2 (Halving value)

118

ROL,ROR

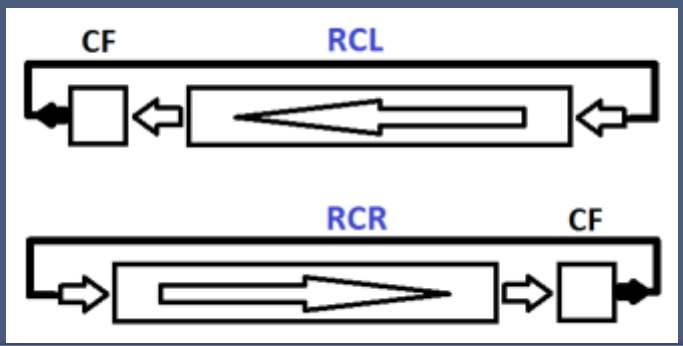
- ROL destination,count
- ROR destination,count
- Destination : reg, mem
- Count : 1, CL



119

RCL,RCR

- RCL destination,count
- RCR destination,count
- Destination : reg, mem
- Count : 1, CL



120

Examples:

```
MOV AL,10001000B  ;AL=10001000B
ROL AL ,1         ;AL=00010001B
                  ;CF=1
RCL AL,1          ;AL=00100011B
                  ;CF=0
```

121

Exercise 4.A:

- Using SHIFT instructions, Write a program to double a 32 bit value stored in DX:AX.

122

TEST

- TEST acts like AND ,
- but does not change values,
- only sets the flags

- Flags Affection: CF=0, OF=0,
- ZF,SF,PF changes.
-

123

Examples:

```
MOV AL, 00000101B     ;AL=00000101B
TEST AL, 00000001B    ; AL=00000101B
                      ;ZF = 0
TEST AL, 00000010B    ;AL=00000101B
                      ;ZF = 1
```

124

Exercise 4.B:

- Using TEST, write a program to check that bits 0,3,6,7 of AL are all '1' (ZF=1) or not (ZF=0).

125

Control and Branch instructions

JMP	**Unconditional Jump**
Jxx	**Conditional Jumps**
LOOP	**Loop CX times**
LOOPZ	**Loop if Zero**
LOOPE	**Loop if Equal**
LOOPNZ	**Loop if Not Zero**
LOOPNE	**Loop if Not Equal**

126

Control and Branch instructions…

CALL	**Call a Procedure**
RET	**Return from a Procedure**
CLI	**Clear Interrupt Flag**
STI	**Set Interrupt Flag**
INT	**Interrupt**
IRET	**Interrupt Return**
NOP	**No Operation**
HLT	**Halt**

127

JMP

- JMP Label
- Unconditional Jump to Label

- Example :
- JMP Quit
- ……
- ……
- Quit: MOV AL,10h

128

Jxx

- Jxx Label
- Conditional Jump to Label

- Jumps based on Flags
- Jumps based on Unsigned Data
- Jumps based on Signed Data
- Jump based on CX register

129

JZ	Jump if Zero	ZF=1
JNZ	Jump if Not Zero	ZF=0
JO	Jump if Overflow	OF=1
JNO	J. if Not Overflow	OF=0
JC	Jump if Carry	CF=1
JNC	Jump if No Carry	CF=0
JS	Jump if Sign	SF=1
JNS	Jump if No Sign	SF=0
JP	Jump if Parity Even	PF=1
JPE	Jump if Parity Even	PF=1
JNP	Jump if Parity Odd	PF=0
JPO	Jump if Parity Odd	PF=0

130

Jumps based on Unsigned Data

- JE Jump if Equal ZF=1
- JNE Jump in Not Equal ZF=0
- JA Jump if Above (CF=0 and ZF=0)
- JAE Jump if Above or Equal CF=0
- JB Jump if Below CF=1
- JBE Jump if Below or Equal (CF=1 or ZF=1)

- JNBE =JA (Not Below or Equal)
- JNB =JAE (Not Below)
- JNAE =JB (Not Above or Equal)
- JNA =JBE (Not Above)

131

Examples:

```
MOV AL,'A'          ;AL=41h
CMP AL,'B'          ;CF=1,ZF=0
JB Label1           ;True
JA Label2           ;False
```

132

Jumps based on Signed Data

- JE Jump if Equal ZF=1
- JNE Jump in Not Equal ZF=0
- JG Jump if Greater (ZF=0 and SF=OF)
- JGE Jump if Greater or Equal SF=OF
- JL Jump if Less SF<>OF
- JLE Jump if Less or Equal (ZF=1 or SF<>OF)

- JNLE =JG (Not Less or Equal)
- JNL =JGE (Not Less)
- JNGE =JL (Not Greater or Equal)
- JNG =JLE (Not Greater)

133

Examples:

```
MOV AL,-5          ;AL=-5=FBh
CMP AL,-7          ;CF=0,ZF=0
                   ; OF=0,SF=0
JG  Label1         ;True
JLE Label2         ;False

CMP AL,1
JG  Label1         ;False
JLE Label2         ;True
```

134

Jumps based on CX Register

- JCXZ Label
- Jump to Label if CX=0

135

Loop

- LOOP Label
- Loops CX times (Until CX<>0).

- Example :

```
MOV     CX,3
MOV     AL,0        ;AL=0
MOV     BL,0        ;BL=0
Lbegin:     ADD AL,10
            ADD BL,20
LOOP        Lbegin
                    ;AL=30   ,   BL=60
```

136

Example :

```
        initialization
        jcxz   endloop              ; CX =0 ?
label1:
        actions
        loop   label1               ; loop
endloop:
```

137

Conditional Loops

- LOOPZ Label
- LOOPE Label
- Loops if CX<>0 and ZF=1.

- LOOPNZ Label
- LOOPNE Label
- Loops if CX<>0 and ZF=0.

138

Example:

```
MOV     CX,3
MOV     AL,0        ;AL=0
MOV     BL,0        ;BL=0
Lbegin:     ADD AL,10
            ADD BL,20
LOOPZ       Lbegin
                    ;AL=10   ,   BL=20
```

139

CALL , RET

- CALL ProcedureName
- Calls a Procedure.

- RET
- Returns from a Procedure.

- Return Address will be saved in Stack.

140

How to Define a Procedure

```
ProcName    PROC     NEAR
            ------------------
            ------------------
            ------------------
             RET
ProcName    ENDP
            ------------------
            ------------------
             CALL          ProcName
```

141

Example:

```
Add7    PROC NEAR
        ADD AL,7
        RET
Add7    ENDP
.....
.....
MOV AL,0        ;AL=0
CALL Add7       ;AL=7
```

142

Interrupt

- Stop a running program, save the state, and allows a special program(an Interrupt Service Routine [ISR]) to run instead, then return to main program and restore the state.
- Mainly used for I/O Operations (generally better than polling).

143

Notice !

- In applications where loss of data cannot be tolerated (e.g. where safety would be affected) the designer must ensure that all of the devices can be properly serviced under worst case conditions. In some of these systems it may be easier to use polling to help ensure correct worst-case behaviour.

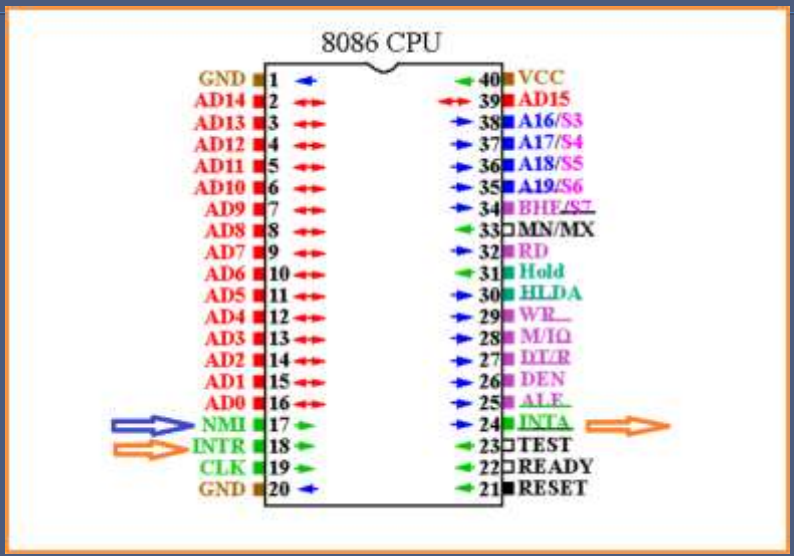
144

Interrupt Types

- Hardware
  - Maskable
  - Nonmaskable
- Software
  - BIOS
  - DOS
  - CPU exceptions (traps)

145

Hardware Interrupt



146

NMI

- Non Maskable Interrupt
- Can Not be Disabled.
- Highist priority.
- For Critical situations,such as:
  - System Overheating
  - Reactor Overpressure
- NMI always generates Interrupt No. 2.

147

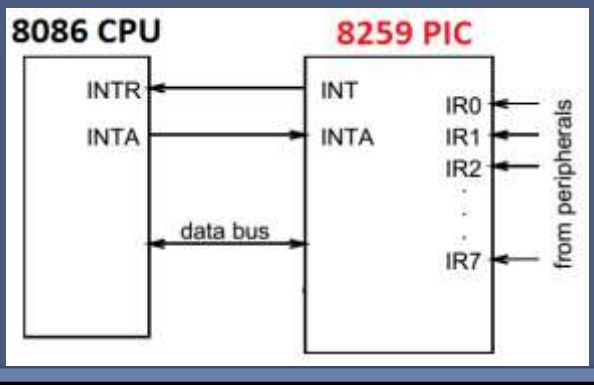
INTR

- Maskable Interrupt
- Can be Disabled by: CLI
  - Clear Interrupt Flag (IF=0)
- And Enabled by : STI
  - Set Interrup Flag (IF=1)
- Priority lower than NMI.
- INTA: Interrupt Acknowledge.
- After Acknowledgement, External device sends required interrupt Number over data bus.

148

Question: What if we have many I/O devices?

- Solution : Use Programmable Interrupt Controller (PIC).



149

INTR, Examples

- In IBM PC:
- Keyboard : Interrupt No. 9
- Serial Port (used for mouse,Modem ,...) : Interrupt No. 12
- Parallel Port (used for printer) : Interrupt No. 15

150

Software Interrupts

- Interrupts that triggered by software(code).
- Applications:
  - Test ISRs
  - Use ISRs (System Calls)
  - CPU Error handling.

151

Software Interrupt Types

- BIOS (Basic Input-Output System)
  - ISR is in System ROM (EPROM)
- DOS (Disk Operating System)
  - ISR is in System RAM,read from Disk.
- CPU exceptions (traps)
  - A exceptional condition in the CPU generates the interrupt.

152

INT

- INT intNumber
- intNumber is between 0 and 255.
- There are 256 ISRs. Their begining addresses calld 'Interrupt Vector's.

153

Interrupt Vector Table(IVT)

- The IVT consists of 256 four-byte pointers, always resides at the first 1KB of memory, ranging from 00000h to 003FFh.
- Example : Interrupt Number 2(NMI), ISR Vector ?
- 2x4=8=8h
- ISR Vector : Offset = [00009h:00008h]
- Segment= [0000Bh:0000Ah]

154

Notice !

- The 8086 operates beginning with the instruction in absolute location FFFF0h of memory , NOT from location 00000h !

155

How to transfer to ISR

- Push to stack:
  - flags register
  - CS
  - IP (Return address)
- Use Interrupt Vector :
  - IP is loaded from the contents of the word location 'intNumber' × 4
  - CS is loaded from the contents of the next word location.
- IF = 0

156

How to Return from ISR

- IRET
- Interrupt Return
- Must be used in the end of ISR.
- Pops from stack:
  - IP
  - CS
  - flags register
- Enable interrupt (IF=1).

157

Example: INT 10h (BIOS)

- INT 10h / AH = 0Eh - Character output.
- AL = character to write.
- this functions displays a character on the screen, advancing the cursor and scrolling the screen as necessary.
- example:

```
MOV     AL, 'N'
MOV     AH, 0Eh
INT     10h
```

158

CPU exceptions (traps)

- A exceptional condition in the CPU generates the interrupt.

- Example 1 : Divide by Zero
- Generates Interrupt No. 0
- ISR Vector : Offset : [00001h:00000h]
- Segment : [00003h:00002h]

- Example 2 : Invalid Opcode
- Generates Interrupt No. 6
- ISR Vector : Offset : [00019h:00018h]
- Segment : [0001Bh:0001Ah]

159

NOP

- NOP
- No Operation
- Applications:
- As a place holder in machine code.
- To create delay for purpose of timing.

160

HLT

- HLT
- Enter Halt State
- CPU halts, waiting for an interrupt.

161

String instructions

MOVSB	**Move byte string**
MOVSW	**Move word string**
CMPSB	**Compare byte string**
CMPSW	**Compare word string**
SCASB	**Scan byte string**
SCASW	**Scan word string**
LODSB	**Load byte string**
LODSW	**Load word string**
STOSB	**Store byte string**
STOSW	**Store word string**
REP	**Repeat**
REPE (REPZ)	**Repeat while equal (zero)**
REPNE (REPNZ)	**Repeat while not equal (not zero)**
LEA	**Load Effective Address**

162

String

- A set of Characters (Bytes or Words)
- Examples: 'Nader' , 'aB91%'
- Registers Implied :
  - DS,SI → DS:SI used for Source String
  - ES,DI → ES:DI used for Destination String.
  - AL,AX
  - DF (Direction Flag)
- Useful for Text Editing and Database Applications.

163

LEA

- Load Effective Address
- LEA reg,StringName
- Reg=Offset Address of StringName
- Examples:
- 0700:0100h LEA SI,str1
- 0700:0103h str1 DB 'Nader'
- SI=0103h

164

REP

- String instructions reference only ONE Byte or ONE Word.
- For repeated execution of string instructions (MOVSB,MOVSW,LODSB,LODSW,STOSB, STOSW), use REP prefix.
- CX : Number of repetations.

165

DF

- The Direction Flag determines the direction of a repeated operation:
- DF=0 : Process string from Left to Right
- DF=1 : Process string from Right to Left.
- To set Direction :
- CLD : Clear D (DF=0)
- STD : Set D (DF=1)

166

MOVSB : Move Byte String

- Copy byte at [DS:SI] to [ES:DI]. Update SI and DI.
- [ES:DI] = [DS:SI]

- if DF = 0 then
  - SI = SI + 1
  - DI = DI + 1
- else
  - SI = SI - 1
  - DI = DI - 1

167

MOVSB : Example

- CLD
- LEA SI, a1
- LEA DI, a2
- MOV CX, 5
- REP MOVSB
- HLT
- a1 DB 'Nader' ;a1='Nader'
- a2 DB 5 DUP(0) ;a2='00000'

- a2='Nader'

168

MOVSW : Move String Word

- Copy word at [DS:SI] to [ES:DI]. Update SI and DI.
- [ES:DI] = [DS:SI]

- if DF = 0 then
  - SI = SI + 2
  - DI = DI + 2
- else
  - SI = SI - 2
  - DI = DI - 2

169

MOVSW : Example

- CLD
- LEA SI, a1
- LEA DI, a2
- MOV CX, 2
- REP MOVSW
- HLT
- a1 DW 1,2 ;a1=01000200h
- a2 DW 2 DUP(0) ;a2=00000000h

- a2=01000200h

170

CMPSB : Compare String Byte

- Compare bytes: [ES:DI] with [DS:SI]. Update SI and DI.
- Calculates [DS:SI] – [ES:DI]
- set flags according to result:
  OF, SF, ZF, AF, PF, CF
- if DF = 0 then
  - SI = SI + 1
  - DI = DI + 1
- else
  - SI = SI - 1
  - DI = DI - 1

171

CMPSW : Compare String Word

- Compare words: [ES:DI] with [DS:SI]. Update SI and DI.
- Calculates [DS:SI] – [ES:DI]
- set flags according to result:
  OF, SF, ZF, AF, PF, CF
- if DF = 0 then
  - SI = SI + 2
  - DI = DI + 2
- else
  - SI = SI - 2
  - DI = DI - 2

172

REPZ / REPE

- Repeat following string instructions while ZF = 1 (result is Zero, Equal), maximum CX times.
- If ZF=0 or CX=0 the EXIT from repetition cycle.
- Can be used for CMPSB, CMPSW, SCASB, SCASW instructions.

173

REPNZ / REPNE

- Repeat following string instructions while ZF = 0 (result is NOT Zero, NOT Equal), maximum CX times.
- If ZF=1 or CX=0 the EXIT from repetition cycle.
- Can be used for CMPSB, CMPSW, SCASB, SCASW instructions.

174

CMPSB : Example

- name1='Nader'
- name2='Naser'
- name1=name2 ?

```
CLD
LEA SI, name1
LEA DI, name2
MOV CX, 5
REPE CMPSB
HLT
name1 DB 'Nader'        ;name1='Nader'
name2 DB 'Naser'        ;name2='Naser'
```

175

SCASB : Scan Byte String.

- Compare bytes: AL with [ES:DI]. Update DI.
- Calculates AL – [ES:DI]

- set flags according to result:
  OF, SF, ZF, AF, PF, CF

- if DF = 0 then
  - DI = DI + 1
- else
  - DI = DI - 1

176

SCASW : Scan Word String.

- Compare words: AX with [ES:DI]. Update DI.
- Calculates AX – [ES:DI]

- set flags according to result:
  OF, SF, ZF, AF, PF, CF

- if DF = 0 then
  - DI = DI + 2
- else
  - DI = DI - 2

177

SCASB : Example

- name1='Nader'
- find first 'e' in name1.

- CLD
- LEA DI, name1
- MOV AL,'e'
- MOV CX, 5
- REPNE SCASB
- HLT
- name1 DB 'Nader' ;name1='Nader'

178

LODSB : Load Byte String.

- Load byte at [DS:SI] into AL. Update SI.
- AL = [DS:SI]

- if DF = 0 then
  - SI = SI + 1
- else
  - SI = SI - 1

179

LODSW : Load Word String.

- Load word at [DS:SI] into AX. Update SI.
- AX = [DS:SI]

- if DF = 0 then
  - SI = SI + 2
- else
  - SI = SI - 2

180

LODSB : Example

- name1='Hello'
- Display name1 in screen using INT 10h.

```
        CLD
        LEA SI, name1
        MOV CX, 5
        MOV AH, 0Eh
m:   LODSB
        INT 10h
        LOOP m
HLT
name1 DB 'Hello'
```

181

تمرین 5- نمایش آرایه حرفی با طول دلخواه.

- برنامه ای بنویسید که یک آرایه حرفی با طول دلخواه و نامعین (یعنی طول آرایه از قبل توسط برنامه نویس مشخص نشده است) را با استفاده از INT 10h بر روی نمایشگر نشان دهد.
- توضیح دهید که هر خط برنامه چه کاری انجام می دهد.
- پاسخنامه : به صورت دست نویس و با ذکر نام و نام خانوادگی و شماره دانشجویی.

182

STOSB : Store Byte String.

- Store byte in AL into [ES:DI]. Update DI.
- [ES:DI] = AL

- if DF = 0 then
  - DI = DI + 1
- else
  - DI = DI - 1

183

STOSW : Store word String.

- Store word in AX into [ES:DI]. Update DI.
- [ES:DI] = AX

- if DF = 0 then
  - DI = DI + 2
- else
  - DI = DI - 2

184

STOSB : Example

- name1='AAAAA'
- Define name1 using STOSB.

- CLD
- LEA DI, name1
- MOV AL, 'A'
- MOV CX, 5
- REP STOSB
- HLT
- name1 DB 5 dup(0)

185

Assembly Program Statements

```
Label: operation operand(s)  ;comment

Example:
START:   MOV AX,BX     ;Copy BX to AX
```

- Space or tab separates fields.
- Comments does not generate any machine code.
- Max. length of each line=128 Characters.

186

Labels

- Can use :
- a to z
- A to Z
- 0 to 9
- Maximum length = 31 Characters
- Must be start with an alphabet.
- Can NOT be a reserved word.
- Meaningful , short labels are better!

187

ORG

- ORG number
- Defines the beginning point of program in memory.
- COM programs always begin in 100h.
- Example :
- ORG 100h
- MOV AL,30

188

Define memory contents

- DB - byte(s)
- DW - word(s)
- DD – double word(s)
- DQ – quad word(s)

- To define a Character or string, use ‘ or “
- Example : S1 DB ‘A’
- S1 : 41h (ASCII code of Character ‘A’)
- A ? represents an uninitialized storage location. Example :

```
D2 DB ?
D2  :  00h
```

189

Arrays

- Any consecutive storage locations of the same size can be called an array

```
X DB 'This is an array'
Y DW 40CH,10B,-13,0
Z DD -109236, FFFFFFFFH, -1, 100B
```

- Components of X are at X, X+1, X+2, X+3,…,X+15
- Components of Y are at Y, Y+2, Y+4, Y+8
- Components of Z are at Z, Z+4, Z+8, Z+12

190

Addressing Arrays

Example :

```
MOV AL,b        ; AL=10h
MOV AL,b+1      ;AL=20h

b  DB 10h,20h,30h
```

- The assembler computes an address based on the expression.
- *NOTE: These are address computations done at assembly time*

  MOV AL, b+1

  will <u>not</u> Add 1 to the value stored at b

191

Word Storage

- Word, double word, and quad word data are stored in little endian format in memory.
- Examples:

```
Directive          Bytes in Storage
s0 DW 35DAh        DA 35
S1 DW 256          00 01
S2 DD 1234567H     67 45 23 01
S3 DQ 10           0A 00 00 00 00 00 00 00
```

192

DUP

- Allows a sequence of storage locations to be defined or reserved
- Examples :
- `S1  DB 8 DUP (5)`
- `S2  DB 5 DUP (?)`
- `S3  DB 3 dup ("AB")`

193

Named Constants

- Named constants are symbols created to represent specific values determined by an expression
- Named constants can be numeric or string
- No storage is allocated for these values

194

EQU Directive

- name EQU expression
  - expression can be string or numeric
  - Use ‘ or “ to specify a string
  - these symbols cannot be redefined later in the program
  - Examples:

```
sample EQU 7Fh
aString EQU "1.234"
message EQU 'This is a message'
```

195

Example

- FACTOR EQU 03H
- ADD AL, FACTOR
- ; will code as ADD AL, 03H

- The advantage of using EQU in this manner is, if FACTOR is used many times in a program and you want to change the value, all you had to do is change the EQU statement at beginning, it will changes the rest of all.

196

Macros

- A group of instructions that perform one task, just as a procedure.
  - a procedure is accessed via a CALL instruction
  - a macro & all instructions defined in the macro, is inserted in the program at the point of usage
- Macros execute faster than procedures because there is no CALL or RET instruction to execute.

197

Why use Macros?

- To simplify assembly coding.
- To make the program more readable.
- To reduce errors caused by repetitive coding.

198

Macro Definition

- Macroname MACRO parameters
- ……
- Assembly code
- Assembly code
- ……
- ENDM

- It's better to define macros in the beginning of code.

199

Macro Example :

- A simple macro to add 2 numbers:

```
Add2    MACRO nbr1 , nbr2
        MOV AL,nbr1          ; first number
        ADD  AL, nbr2        ; second number
        ENDM
```

- Using this Macro :

```
        Add2      12h,20h
        Add2      40h,50h
```

200

Macro expansion:

- MOV AL,12h
- ADD AL,20h

- MOV AL,40h
- ADD AL,50h

201

LOCAL variables

- A local variable is one that appears in the macro, but is NOT available outside the macro.
- to define use the LOCAL directive
- The LOCAL directive must always be used on the line immediately following the MACRO statement.
- The LOCAL statement may have up to 35 labels, all separated with commas.

- LOCAL label1,label2, …

202

Example :

- A simple macro to find minimum of two values :

```
Min2      MACRO    first, second
          LOCAL      endcmp
 ; put smaller of two words in the AX register
          MOV        AX, first          ; first value
          CMP        AX, second         ; Compare
          JLE        endcmp             ; exit if so
          MOV       AX, second          ; otherwise
endcmp:  NOP
          ENDM
```

203

Macro Libraries

- Macro definitions can be placed in the program file or ,a file can be created that contains only macros.
- To be included with other program files, Use the INCLUDE directive to indicate a program file will include a module that contains external macro definitions.
- It acts as a library of macro sequences.
- Programs may contain both macro include files and library files.

204

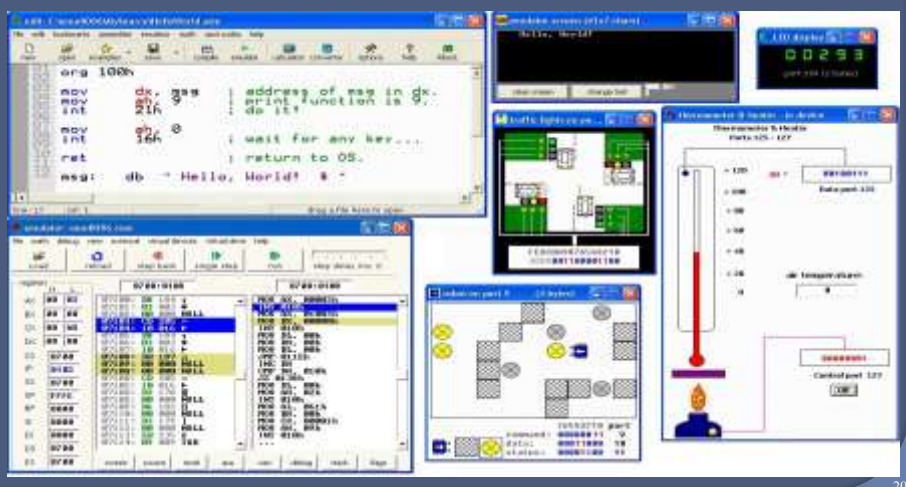
How to practice and test programs ?

- CPU Development Kits

205

How to practice and test programs ?

- CPU Simulators/Emulators
- EMU8086



206

توضیح در مورد امتحان پایان ترم

امتحان : **تستی (چهار گزینه ای – بدون نمره منفی)** در تاریخ و ساعت اعلام شده در سامانه سما.

توضیح در مورد امتحان پایان ترم ...

سامانه برگزاری امتحان: سامانه جام.

- **حتما راهنمای شرکت در آزمونهای این سامانه را قبل از امتحان مطالعه کنید.**
- **هر خبر و اطلاعیه در مورد درس و امتحان در سامانه جام ذکر خواهد شد.**


Q&A