Computer Organization and Architecture

X86 Assembly

Content

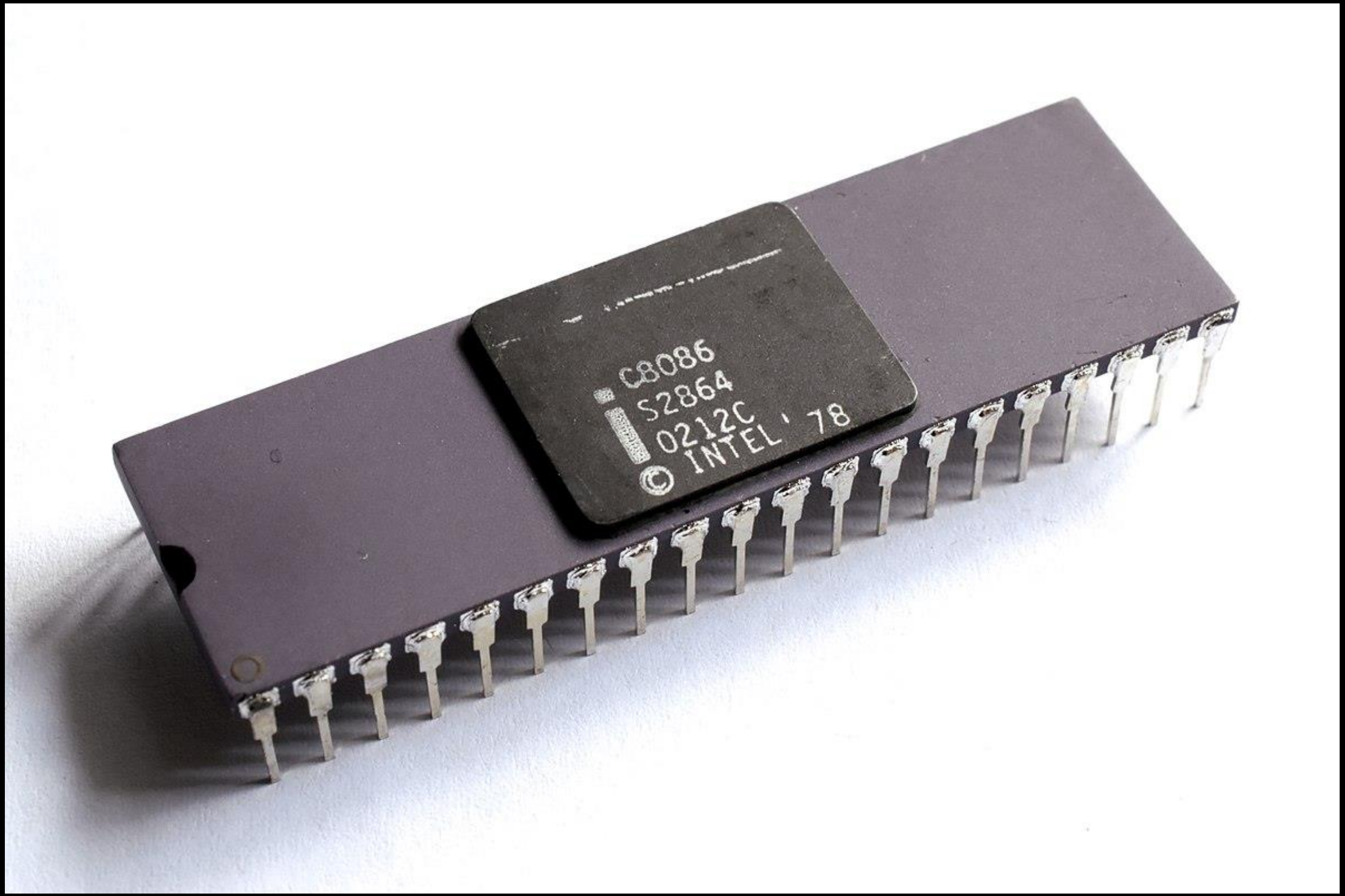
	Agenda
➡	The 8086 Processor
	Assembly Programming
	Program Segments
	Exercises

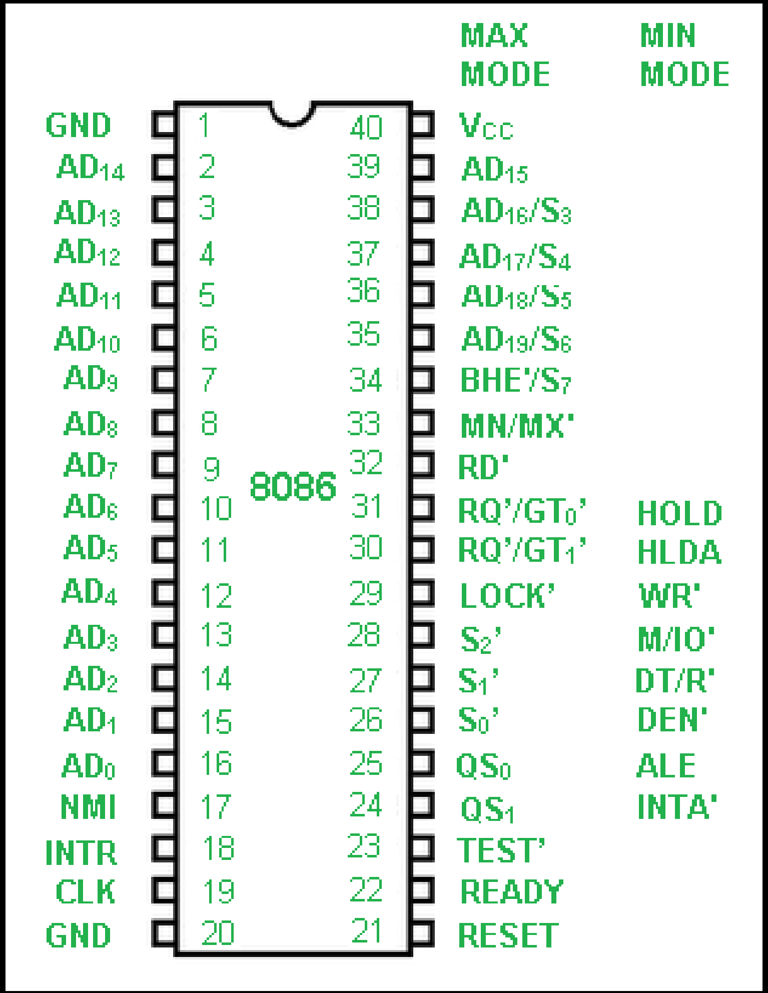
The 8086 Processor

There are several improvements of the 8086 processor from the previous generation:

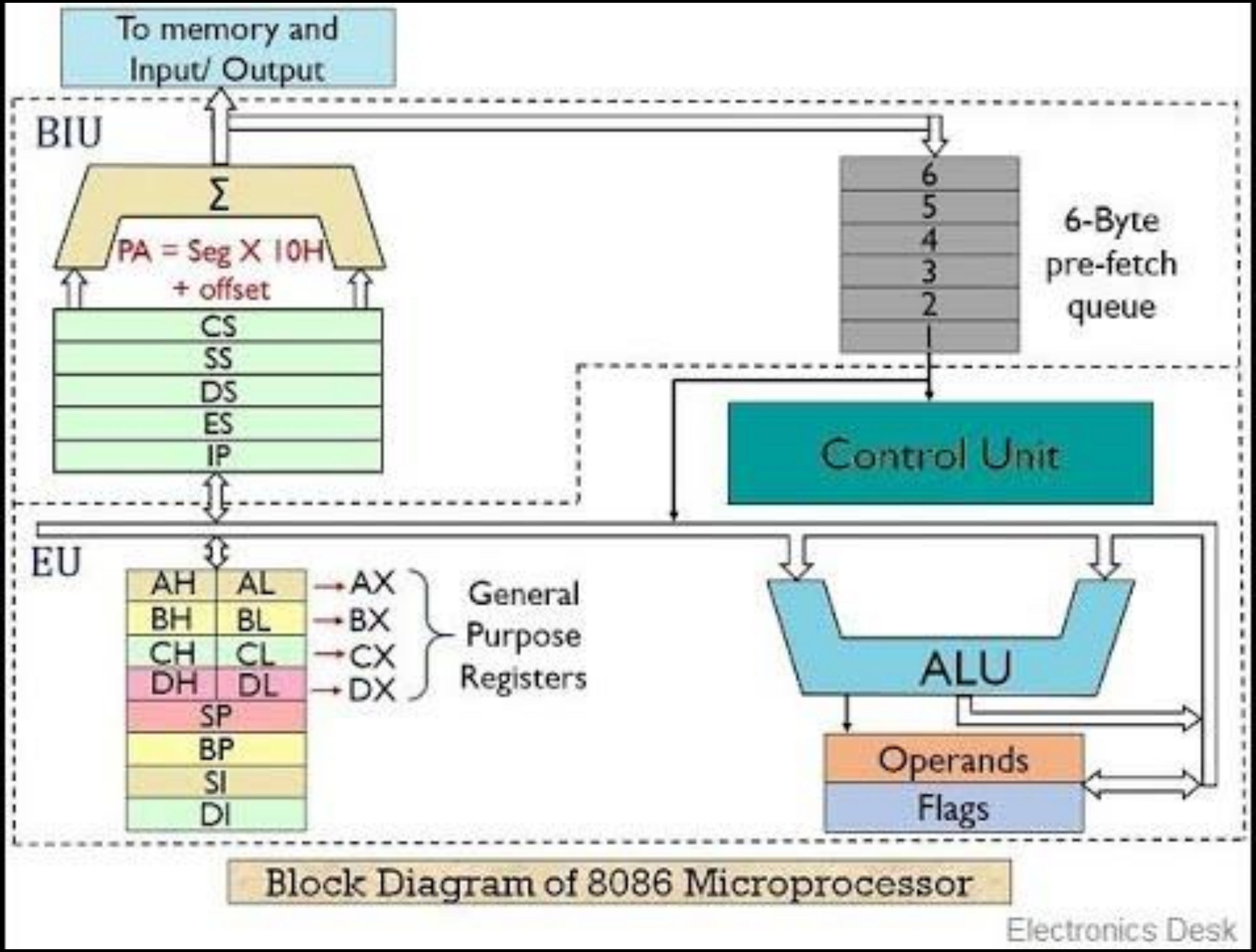
- The processor cand address 1 megabyte of memory.
- It is a 16-bit micro-processor. Meaning that the processor can work on 16-bits of data at a time.
  - All registers are 16 bits wide and there is a 16-bit data bus to transfer data in and out of the CPU.
- The 8086 was a pipelined processor. Pipelined means that the processor can process information at the same time the buses are busy transferring data.
  - Thereby increasing the effective processing power of the microprocessor.

The 8086 Processor




MAX MODE
MIN MODE
8086
GND 1 — 40 V$_{CC}$
AD$_{14}$ 2 — 39 AD$_{15}$
AD$_{13}$ 3 — 38 AD$_{16}$/S$_3$
AD$_{12}$ 4 — 37 AD$_{17}$/S$_4$
AD$_{11}$ 5 — 36 AD$_{18}$/S$_5$
AD$_{10}$ 6 — 35 AD$_{19}$/S$_6$
AD$_9$ 7 — 34 BHE'/S$_7$
AD$_8$ 8 — 33 MN/MX'
AD$_7$ 9 — 32 RD'
AD$_6$ 10 — 31 RQ'/GT$_0$' HOLD
AD$_5$ 11 — 30 RQ'/GT$_1$' HLDA
AD$_4$ 12 — 29 LOCK' WR'
AD$_3$ 13 — 28 S$_2$' M/IO'
AD$_2$ 14 — 27 S$_1$' DT/R'
AD$_1$ 15 — 26 S$_0$' DEN'
AD$_0$ 16 — 25 QS$_0$ ALE
NMI 17 — 24 QS$_1$ INTA'
INTR 18 — 23 TEST'
CLK 19 — 22 READY
GND 20 — 21 RESET

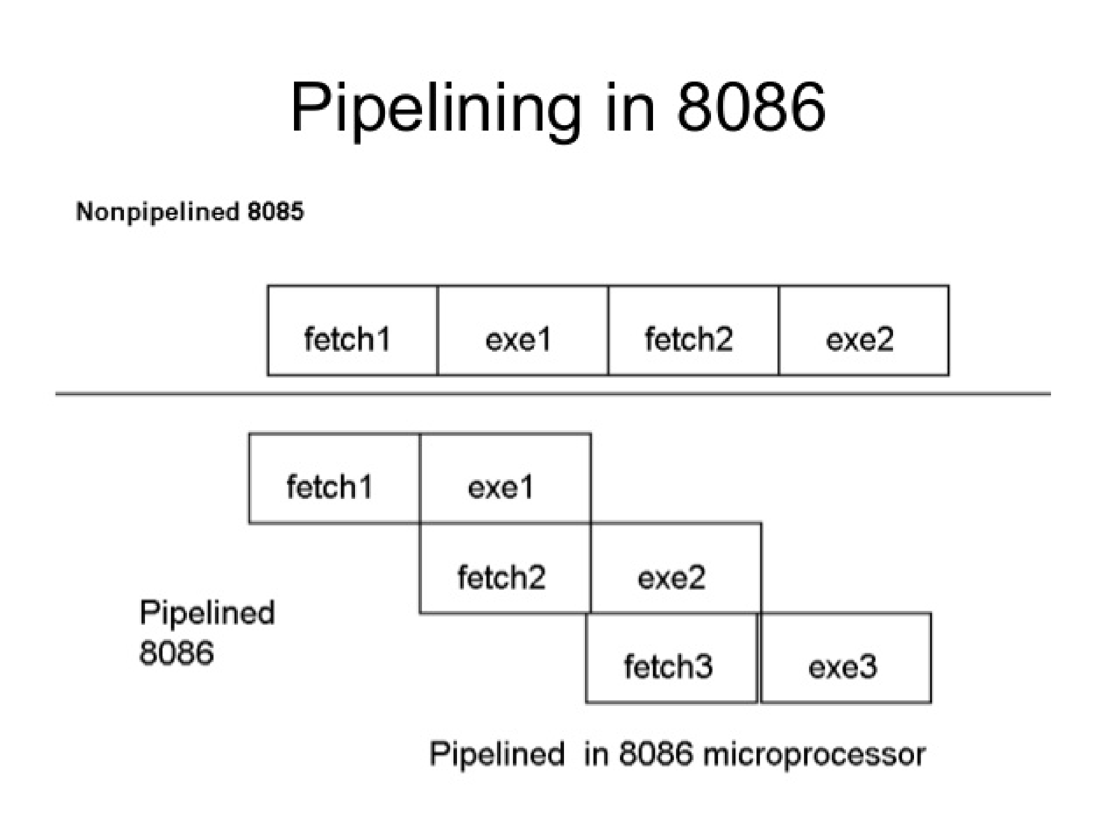
The 8086 Processor



Block Diagram of 8086 Microprocessor

The 8086 Processor

- The idea of pipelining in its simplest form is to allow the CPU to fetch and execute at the same time.



Pipelining in 8086

Nonpipelined 8085

fetch1 exe1 fetch2 exe2

fetch1 exe1

fetch2 exe2

fetch3 exe3

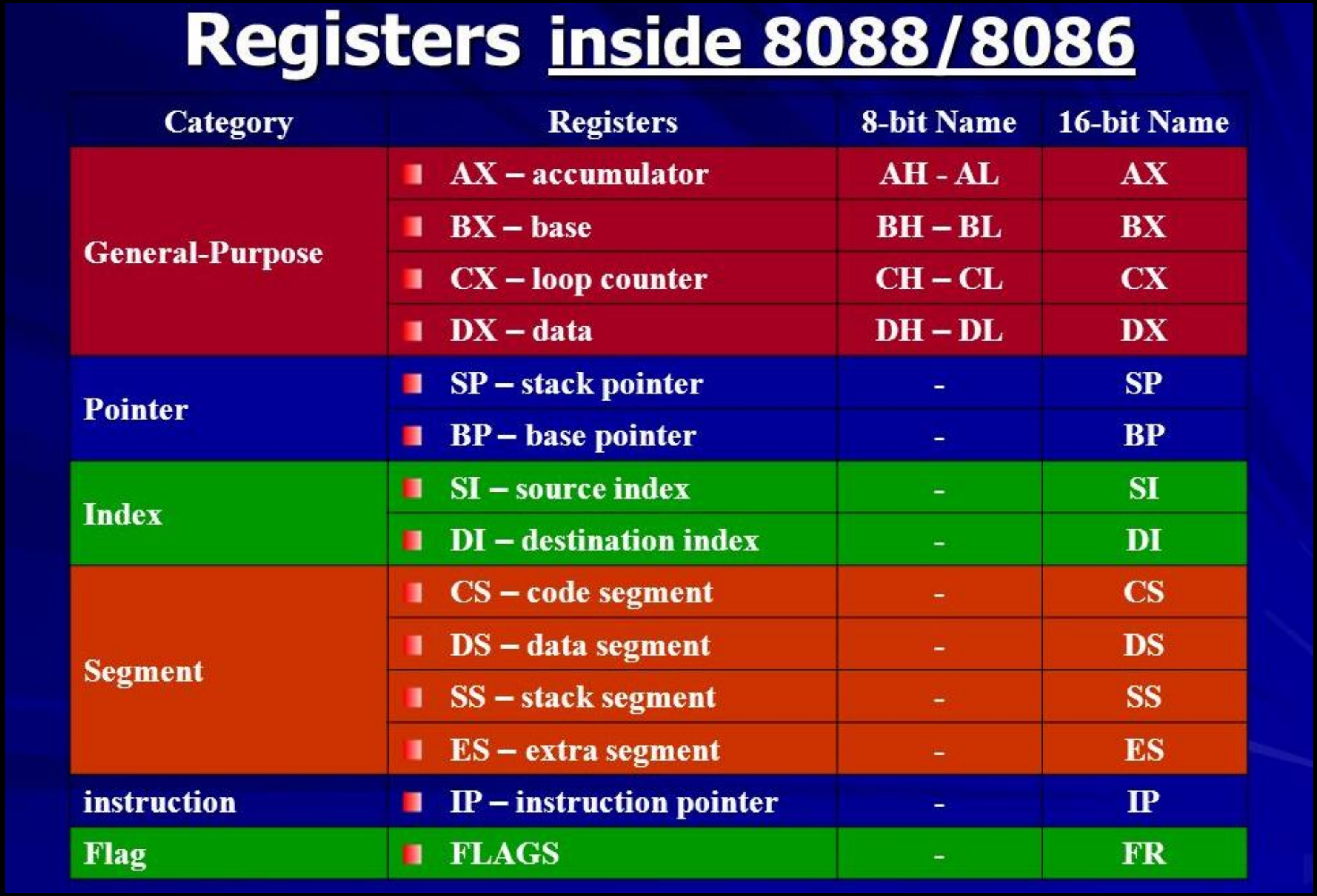
Pipelined 8086

Pipelined in 8086 microprocessor

The 8086 Processor

- In the CPU, registers are used to store information temporarily.
- The registers of the 8088/86 fall into the six categories.

the use of registers will be described in the context of instructions and their application in a given program.

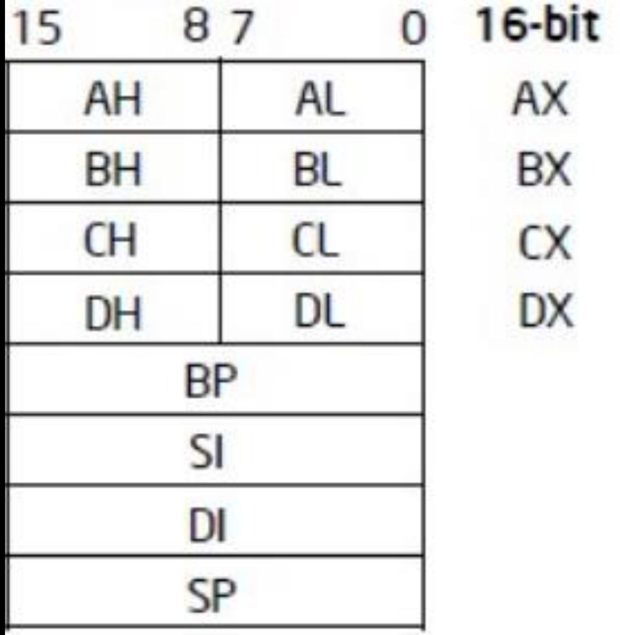


Registers inside 8088/8086

Category	Registers	8-bit Name	16-bit Name
General-Purpose	AX – accumulator	AH - AL	AX
	BX – base	BH – BL	BX
	CX – loop counter	CH – CL	CX
	DX – data	DH – DL	DX
Pointer	SP – stack pointer	-	SP
	BP – base pointer	-	BP
Index	SI – source index	-	SI
	DI – destination index	-	DI
Segment	CS – code segment	-	CS
	DS – data segment	-	DS
	SS – stack segment	-	SS
	ES – extra segment	-	ES
instruction	IP – instruction pointer	-	IP
Flag	FLAGS	-	FR

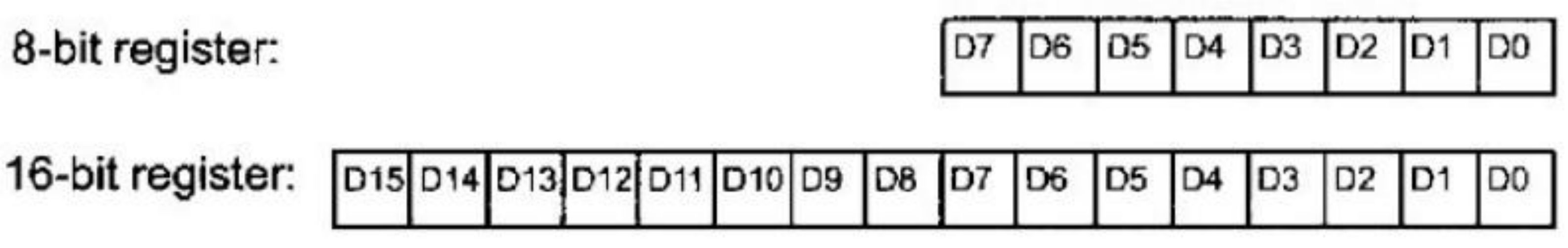
The 8086 Processor

- The general-purpose registers in 8086 microprocessors can be accessed as either 16-bit or 8-bit registers.
- All other registers are accessed only as the full 16 bits.



15 8	7 0	16-bit
AH	AL	AX
BH	BL	BX
CH	CL	CX
DH	DL	DX
BP		
SI		
DI		
SP		

- In the 8088/86, data types are either 8 or 16 bits.
    - o To access 12-bit data a 16-bit register must be used with the highest 4 bits set to 0.
- The bits of a register are numbered in descending order.



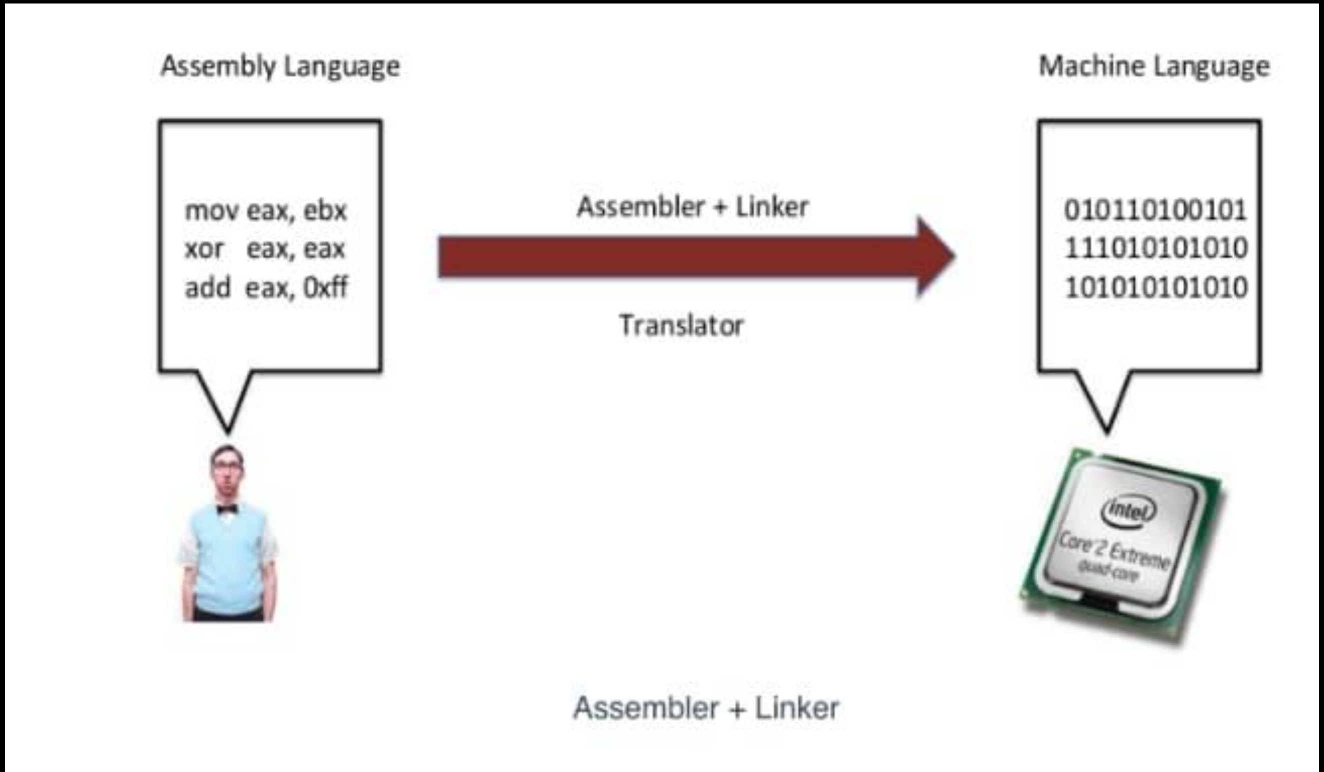
8-bit register:									D7	D6	D5	D4	D3	D2	D1	D0
16-bit register:	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0

Content

Agenda
The 8086 Processor
➡ Assembly Programming
Program Segments
Exercises

Assembly Programming

- Assembly language is
  - a symbolic language to represent machine code.
  - converted to a machine code via *Assembler.*
  - a low-level language, as it deals directly with the internal structures of the CPU.
- To program in Assembly language, the programmer must know the number of registers and their size and details of the CPU.



```
mov eax, ebx
xor  eax, eax
add  eax, 0xff
```

```
010110100101
111010101010
101010101010
```

Assembly Programming

MOV instruction

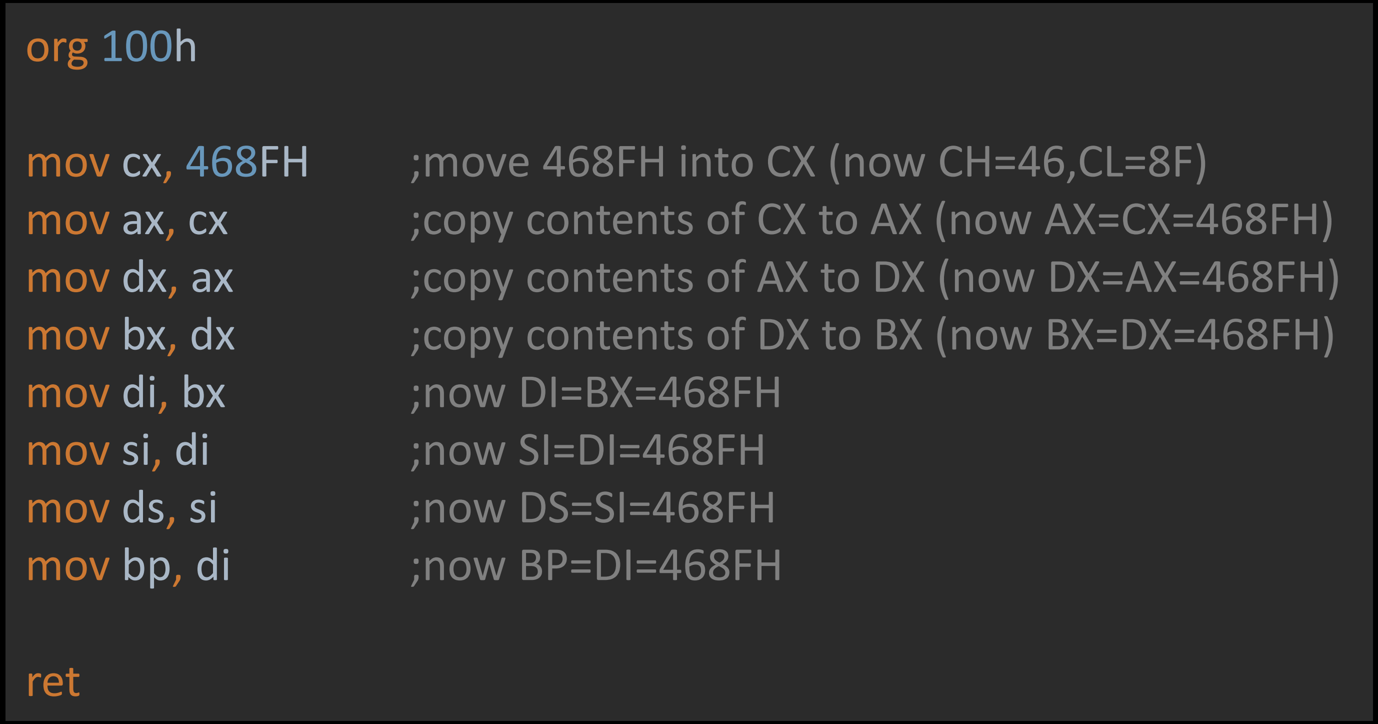
- Copies data from one location to another.
- Syntax: $MOV\ [destination], [source]$
- Example: using 8-bit registers

```
org 100h              ;start location of the machine code in memory

mov cl, 55h           ;copy the 55 hex value to register cl
mov dl, cl            ;copy from cl to dl
mov ah, dl            ;copy from dl to ah
mov al, ah            ;copy from ah to al
mov bh, cl            ;copy from cl to bh
mov ch, bh            ;copy from bh to ch


ret
```

Assembly Programming

MOV instruction

- Example: using 16-bit registers



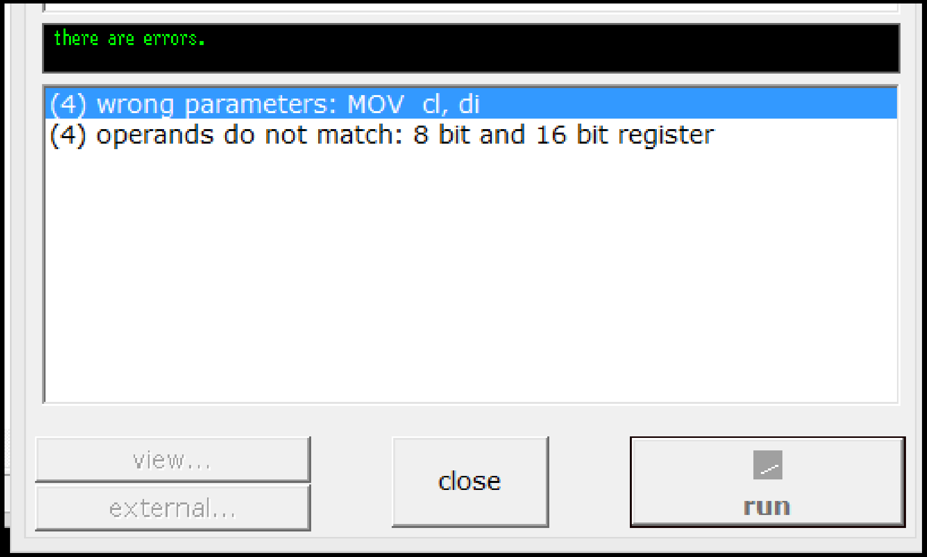
```
org 100h

mov cx, 468FH       ;move 468FH into CX (now CH=46,CL=8F)
mov ax, cx          ;copy contents of CX to AX (now AX=CX=468FH)
mov dx, ax          ;copy contents of AX to DX (now DX=AX=468FH)
mov bx, dx          ;copy contents of DX to BX (now BX=DX=468FH)
mov di, bx          ;now DI=BX=468FH
mov si, di          ;now SI=DI=468FH
mov ds, si          ;now DS=SI=468FH
mov bp, di          ;now BP=DI=468FH

ret
```

Assembly Programming

MOV instruction

- In the 8086, data can be moved among all the registers except the flag register.
  - The Flag register is a Special Purpose Register which shows the status of the task.
- If source and destination registers have different sizes, the emulator gives error.
  - In the example, di is 16-bits, while cl is 8-bits.

```
mov di, 468FH
mov cl, di
```

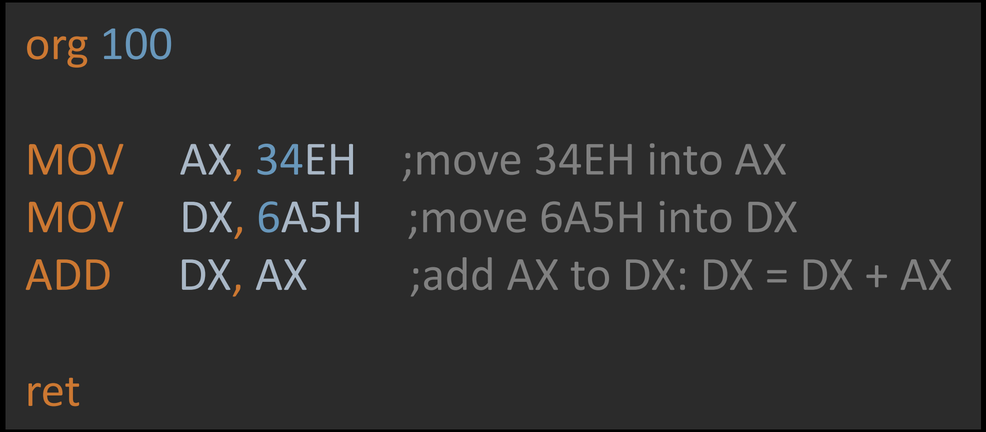


```
there are errors.
(4) wrong parameters: MOV  cl, di
(4) operands do not match: 8 bit and 16 bit register
```

Assembly Programming

ADD instruction

- Add the hexadecimal numbers $34EH + 6A5H = 9F3H$.
  - H for hex.



```
org 100

MOV    AX, 34EH    ;move 34EH into AX
MOV    DX, 6A5H    ;move 6A5H into DX
ADD    DX, AX      ;add AX to DX: DX = DX + AX

ret
```

Content

Agenda
The 8086 Processor
Assembly Programming
➡ Program Segments
Exercises

Program Segments

- A segment is an area of memory that includes up to 64K bytes.

- An Assembly language program consists of at four segments:
  - a code segment, contains the Assembly language instructions that perform the tasks
  - a data segment, stores information (data) that needs to be processed by the instructions in the code segment.
  - a stack segment, stores information temporarily.
  - an extra segment, an auxiliary data segment.

Program Segments

In the 8086 processor, there are three types of addresses:

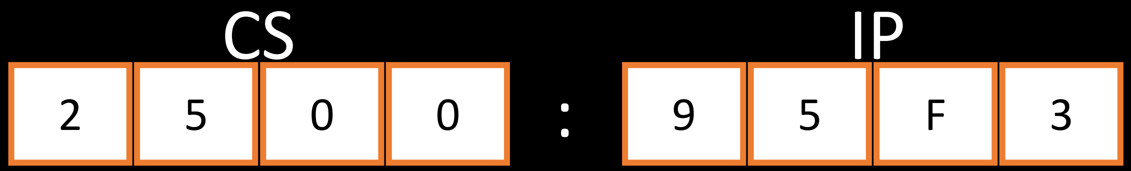
- The *physical address,* is the 20-bit address that is actually put on the address pins of the microprocessor and decoded by the memory interfacing circuitry.
  - This address can have a range of 00000H to FFFFFH.
  - This is an actual physical location in RAM or ROM.
- The *offset address,* is a location within a 64K-byte segment range.
  - Used to access locations within memory.
  - ranges from 0000H to FFFFH.
- The *logical address*, consists of a segment value and an offset address.

Program Segments

- The 8086 fetches the instructions (opcodes and operands) from the code segment.

Program Segments

- The 8086 fetches the instructions (opcodes and operands) from the code segment.
- The logical address of an instruction consists of a code segment and an instruction pointer.



CS

2 5 0 0 : 9 5 F 3

IP

  - The IP contains the offset address.

Program Segments

- The 8086 fetches the instructions (opcodes and operands) from the code segment.
- The logical address of an instruction consists of a code segment and an instruction pointer.

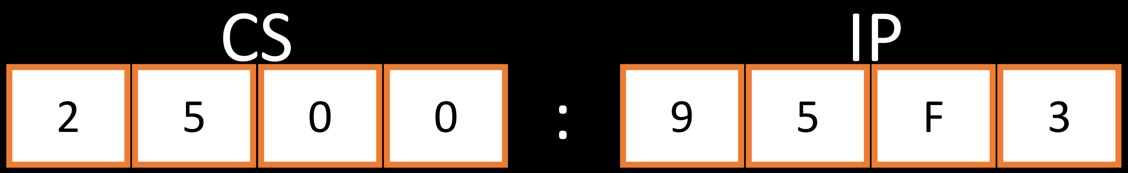
CS					IP			
2	5	0	0	:	9	5	F	3

  - The IP contains the offset address.
- The physical address for the location of the instruction is generated by
  - shifting the CS left one hex digit

2	5	0	0	0

Program Segments

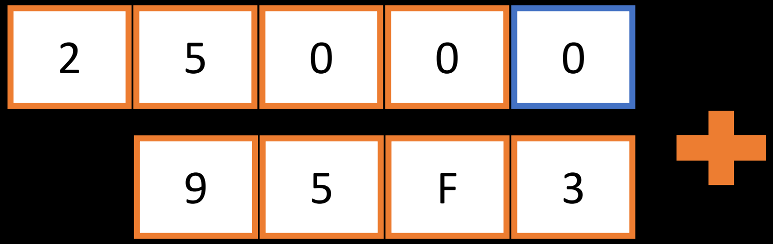
- The 8086 fetches the instructions (opcodes and operands) from the code segment.
- The logical address of an instruction consists of a code segment and an instruction pointer.



CS : IP

2 5 0 0 : 9 5 F 3

  - The IP contains the offset address.
- The physical address for the location of the instruction is generated by
  - shifting the CS left one hex digit
  - Add it to the IP



2 5 0 0 0

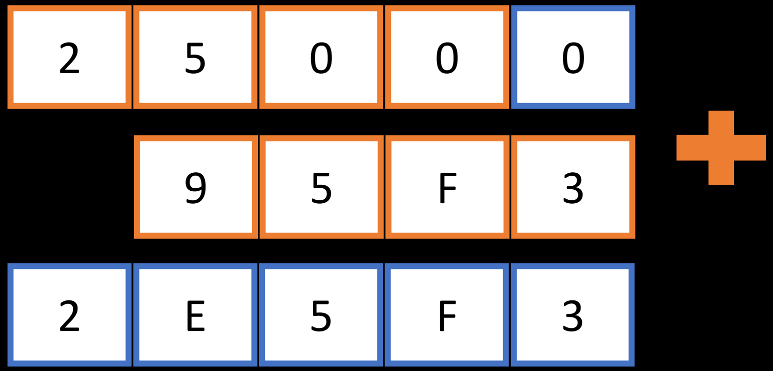
+ 9 5 F 3

Program Segments

- The 8086 fetches the instructions (opcodes and operands) from the code segment.
- The logical address of an instruction consists of a code segment and an instruction pointer.

```
      CS               IP
2 | 5 | 0 | 0  :  9 | 5 | F | 3
```

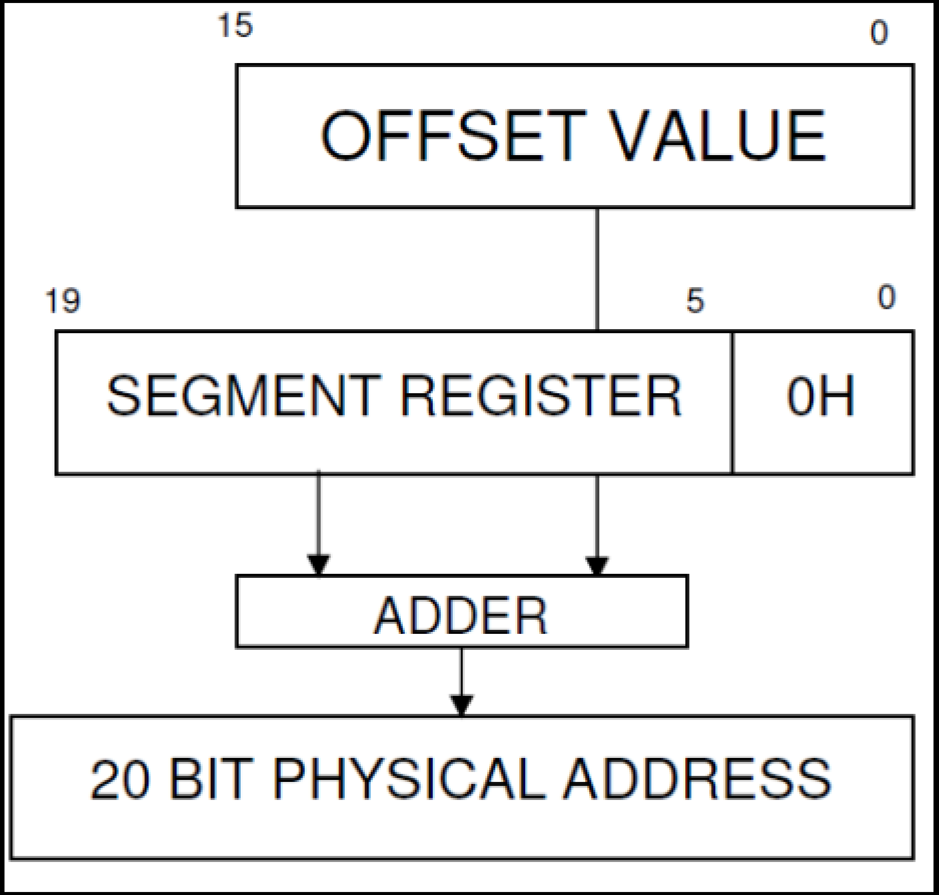
  - The IP contains the offset address.
- The physical address for the location of the instruction is generated by
  - shifting the CS left one hex digit
  - Add it to the IP
  - The resulting 20-bit address is the physical address



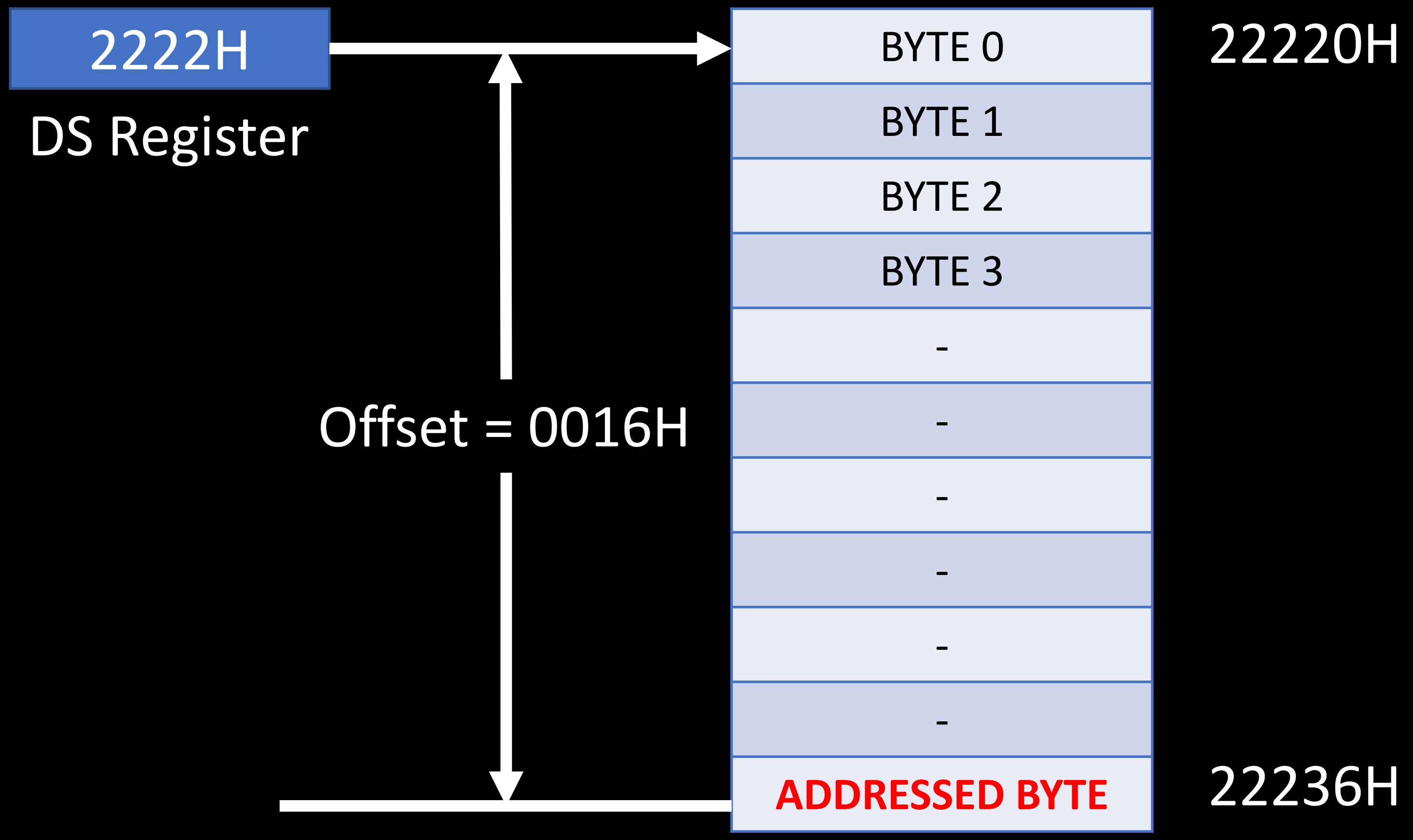
```
2 | 5 | 0 | 0 | 0
    9 | 5 | F | 3    +
2 | E | 5 | F | 3
```

Program Segments

- In memory, data is stored as bytes.
- Each byte has a specific address.
- Intel 8086 has 20 lines address bus.
- With 20 address lines, the memory that can be addressed is 2^{20} bytes = 1 MB.
- 8086 can access memory with address ranges from 00000H to FFFFFH (4 bits * 5 hex).
- We shift one hex because the segment registers (16 bits) cannot store 20 bits.



Program Segments



2222H

DS Register

Offset = 0016H

BYTE 0

BYTE 1

BYTE 2

BYTE 3

-

-

-

-

-

-

ADDRESSED BYTE

22220H

22236H

Content

Agenda
The 8086 Processor
Assembly Programming
Program Segments
Exercises

Exercises

- Add the numbers 5+6

```
org 100h

mov al, 5   ;AL=5
add al, 6   ;AL=11=B

ret
```

- Add the numbers 8 + -3

```
org 100h

mov al, 8    ;AL=8
add al, -3   ;AL=5

ret
```

Exercises

- Subtract the numbers: 13 – 10

```
org 100h

mov al, 13   ;AL=D
sub al, 10   ;AL=3

ret
```

- Or use HEX.
  - Note that, if we delete the leading 0, it will be considered the DH register.
  - The H is for hex.

```
org 100h

mov al, 0DH   ;AL=D
sub al, 0AH   ;AL=3

ret
```

Exercises

- Logical XOR (Exclusive OR) between all bits of two operands.
- These rules apply:
    - 1 XOR 1 = 0
    - 1 XOR 0 = 1
    - 0 XOR 1 = 1
    - 0 XOR 0 = 0

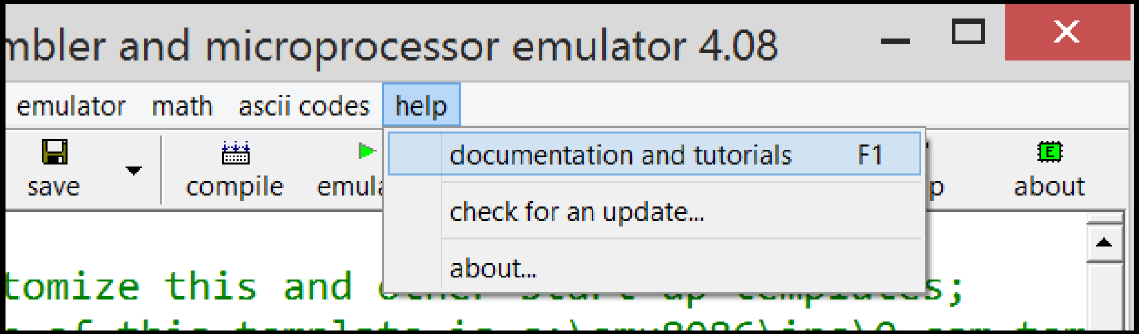
- 7 XOR 2 = 5
    - “b” for binary.

```
org 100h

mov dl, 00000111b    ;DL=7
xor dl,   00000010b     ;DL=00000101b = 5

ret
```

TASK

- Apply the following instructions for the numbers 99 and 18. What is the final output in which register ?
  - CMP
  - AND
  - NOT
  - OR
- Write the program: 9 xor 15, then move the result to lower 8-bits of the accumulator, then add 5, then move the result to lower 8-bits of the CX.

To see the instruction set:


mbler and microprocessor emulator 4.08
emulator
math
ascii codes
help
save
compile
emula
documentation and tutorials F1
check for an update...
about...
about
tomize this and