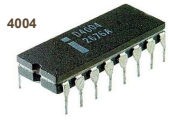
Computer Structure and Language

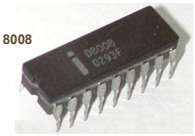
The 8086/8088 Assembly Language

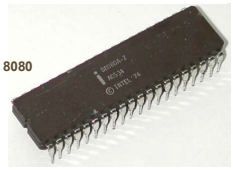
Hamid Sarbazi-Azad
Department of Computer Engineering
Sharif University of Technology (SUT)
Tehran, Iran

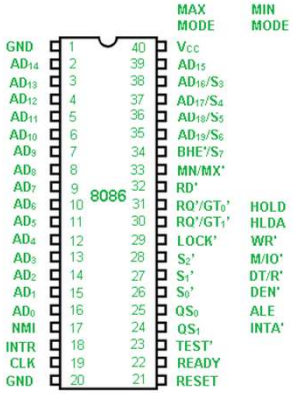
Intel's 4004 to 8086 Microprocessor

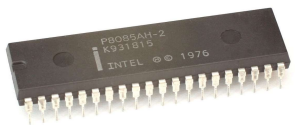
4004: 4-bit microprocessor designed for desktop printing calculator (2300 Transistors), 1971
8008: Early byte-oriented microprocessors (3500 Transistors), 1972
4040: Successor to 4004 (3000 Transistors), 1974
8080: Second 8-bit microprocessor (6000 Transistors), 1974
8085: 8-bit microprocessor (8080 software-binary compatible, 6500 Transistors), 1976
8086: 16-bit microprocessor (29000 Transistors), 1978





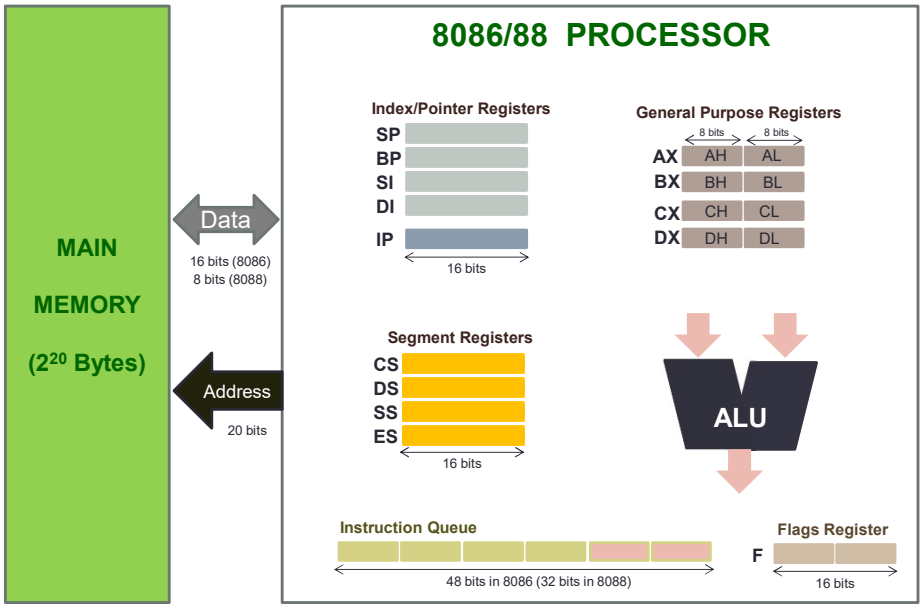






NOTE. Wafer Scale Engine (WSE) has 1.2 trillion (1200 000 000 000) transistors with a die area of 46225 mm² (with 400000 cores for AI/ML processing and 18 GB on-die SRAM (reported at HotChips 2019).

The 8086 System Structure

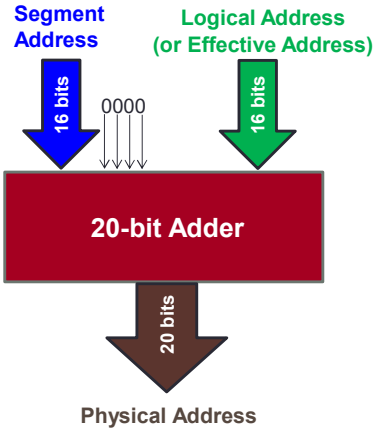


8086/88 features:

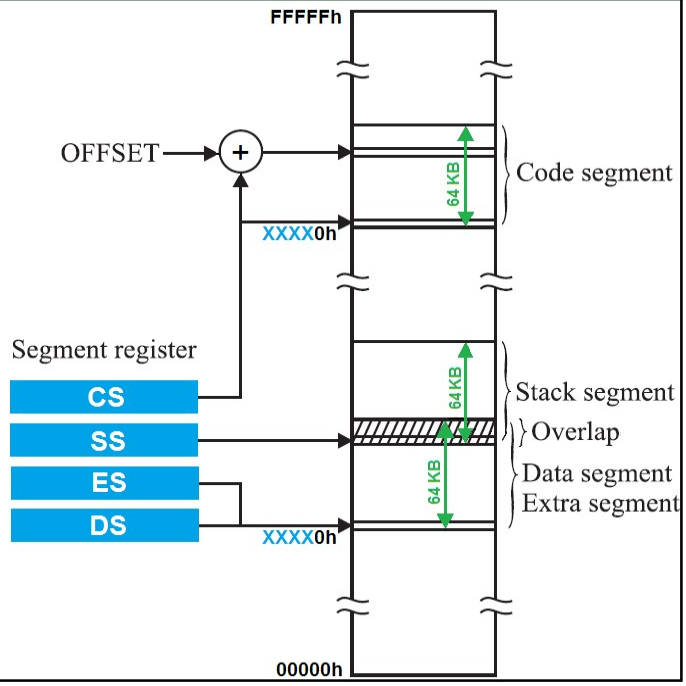
- First 16-bit microprocessor (8088 has an 8-bit data bus)
- 2^{20}-byte addressable segmented main memory (20-bit address), Little Endian
- Data types:
  - 8/16/32 bit (byte, word, double-word) binary (signed, unsigned),
  - 8/16 bit decimal BCD (partially supported),
  - Character, String
- Addressing modes:
  - Implied
  - Immediate (d8, d16)
  - Direct (register, memory): AX, BX, CX, …, and d16
  - Indirect (register, memory): (BX), (SI), (DI), (BP)
  - Indexed: (SI)±d8, (SI)±d16, (DI)±d8, (DI)±d16
  - Base-displacement: (BX)±d8, (BX)±d16, (BP)±d8, (BP)±d16
  - Base-indexed: (SI)+(BX), (DI)+(BX), (SI)+(BP), (DI)+(BP)
  - Base-displacement-indexed: (SI)+(BX)±d8, (DI)+(BX)±d8, (SI)+(BP)±d8, (DI)+(BP)±d8, (SI)+(BX)±d16, (DI)+(BX)±d16, (SI)+(BP)±d16, (DI)+(BP)±d16
  - Segmented addressing as 8086's *base addressing mode*

Address Generation Scheme (Segmented)

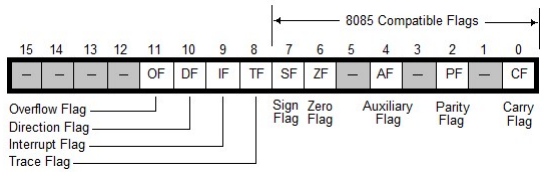
Physical Address = 16 x Segment Address + Logical (Effective) Address

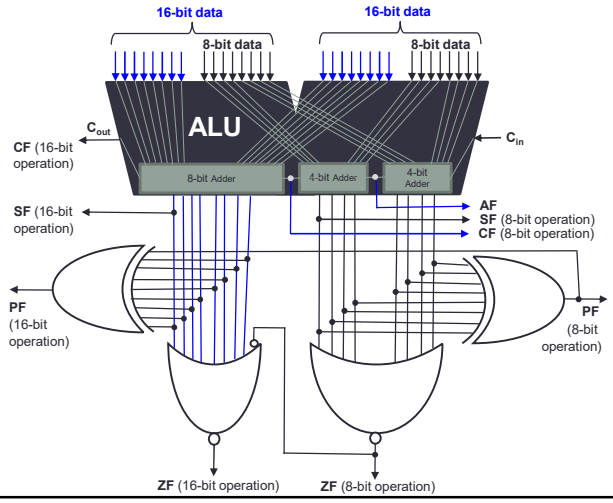


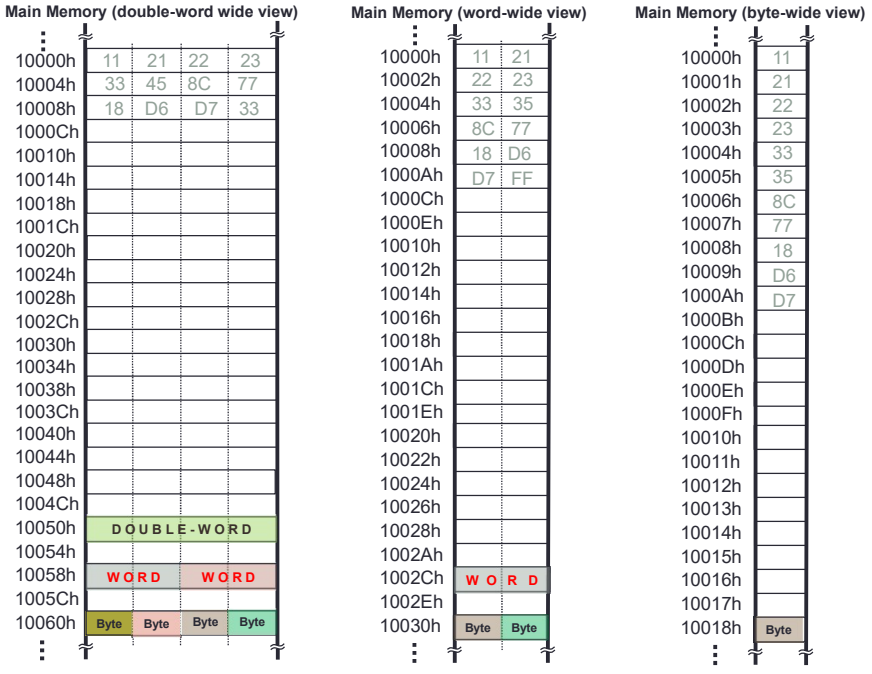
8086/88 Segmented Memory



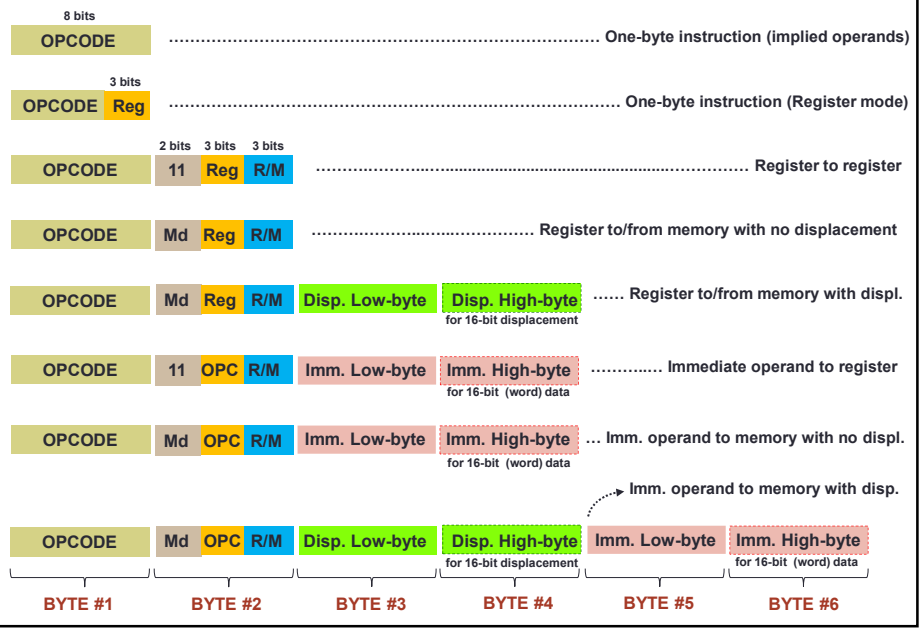
Flags Register







The 8086/88 Instruction Format



Memory address calculation using Md and R/M fields

	Memory mode			Register mode		
	Md = 00	Md = 01	Md = 10	Md = 11		
	SR : EA	SR : EA	SR : EA	W = 0	W = 1	
000	DS : (BX)+(SI)	DS : (BX)+(SI)+d8	DS : (BX)+(SI)+d16	AL	AX	000
001	DS : (BX)+(DI)	DS : (BX)+(DI)+d8	DS : (BX)+(DI)+d16	CL	CX	001
010	SS : (BP)+(SI)	SS : (BP)+(SI)+d8	SS : (BP)+(SI)+d16	DL	DX	010
011	SS : (BP)+(DI)	SS : (BP)+(DI)+d8	SS : (BP)+(DI)+d16	BL	BX	011
100	DS : (SI)	DS : (SI)+d8	DS : (SI)+d16	AH	SP	100
101	DS : (DI)	DS : (DI)+d8	DS : (DI)+d16	CH	BP	101
110	DS : d16	SS : (BP)+d8	SS : (BP)+d16	DH	SI	110
111	DS : (BX)	DS : (BX)+d8	DS : (BX)+d16	BH	DI	111

R/M (left) — Reg (right)

SR: Segment Register **EA:** Effective Address

The 8086/88 supports 7 types of instructions:

1. **Data Transfer Instructions:** mov,push,pop,pusha,popa,xchg,xlat,in, out,lea,lds,les,lahf,sahf,pushf,popf
2. **Arithmetic Instructions:** add,adc,inc,aaa,daa,sub,sbb,dec,neg,cmp, aas,das,mul,imul,aam,div, idiv,aad,cbw,cwd
3. **Bit Manipulation Instructions:** not,and,or,xor,test,shl/sal,shr,sar,rol,ror, rcr,rcl
4. **String Instructions:** rep,repe/repz,repne/repnz,movs/movsb/movsw, cmps/cmpsb/cmpsw,ins/insb/insw,outs/outsb/outsw,scas/sacsb/scasw, lods/lodsb/lodsw
5. **Program Execution Transfer Instructions:** call,ret,jmp,ja/jnbe, jae/jnb,jbe/jna,jc,je/jz,jg/jnle,jge/jnl,jl/jnge,jle/jng,jnc,jne/jnz,jno,jnp/jpo, jns,jo,jp/jpe,js,loop,loope/loopz,loopne/loopnz,jcxz
6. **Processor Control Instructions:** stc,clc,cmc,std,cld,sti,cli
7. **Interrupt Instructions:** int,into,iret

The 8086/88 Program Structure/Layout:

```
Seg_name1   SEGMENT
            .
            .   Directives/instructions
            .
Seg_name1   ENDS

Seg_name2   SEGMENT
            .
            .   Directives/instructions
            .
Seg_name2   ENDS

            .
            .

Seg_nameX   SEGMENT

Start_label:
            .
            .   Directives/instructions
            .
Seg_nameX   ENDS
            END  Start_label
```

Useful notes:

- General format of an assembly instruction is:

  Label: Mnemonic Operand, Operand ;Comment to document and help readers

  Example: Loop: mov ax, array+100[bx] ; Load the next element of the array into accumulator

- You can use numbers in the assembly program in Decimal, Octal, Binary and Hexadecimal bases using D, O, B, and H suffixes. For example, we can write:

  Example: 000001010111B == 0127O == 87D or 87 == 057H

- You can use:
  - DB (define byte), DW (define word), DD (define double-word) directives to define a variable and initialize it; (sophisticated ways to define more complex types)
  - prefix SEGMENT (OFFSET) before a label to get its Segment address (Offset within its segment)
  - prefix PTR is used before an address or expression to indicate its type (byte, word, dword) or its reference distance (near, far)
  - EQU (Equate) directive to define a constant/variable/expression and name it.
    Example: Ten EQU 10 ; then use as: add AX,Ten == Add AX,10
    Convert_byte_to_word EQU CBW ; then use as: Convert_byte_to_word == CBW
  - ORG (Originate) directive to move Location Counter at a given address.
    Example: ORG 100h
  - EVEN directive to move Location Counter at next even address (pass one byte if the current address is odd).

Examples of Variable Definition:

```
        ....
; Let us assume Location Counter = 0018h here.
var1:  db 10, 2 dup(-1), 101b, 76O, 2 dup (0, 11h)  ;@var1= 0018h
var2:  dw -2, 7689h, 256, 2 dup (-1,?)   ; my words (@var2 = 0021h)
ham:  dd  100, ?, 12345678h    ; @ham = 002Fh
my_vector:  db 10 dup (?)        ; @my_vector = 003Bh
my_str:  db  'abcdefg', 0, -1, 3 dup ('A')      ; @my_str = 0045h
Titan:     db  2 dup('allah', 0)                ; @Titan = 0052h
Array5:  db       20 dup (101b)                 ; @Array5 = 005Eh
; Here Location Counter = 0072h
        ....
        ....
```

Symbol Table

Symbol	Address
var1	0018h
var2	0021h
ham	002Fh
my_vector	003Bh
My_str	0045h
Titan	0052h
Array5	005Eh

Memory Segment (logical view)

0018h	0A	FF	FF	05
001Ch	3E	00	11	00
0020h	11	FE	FF	89
0024h	76	00	01	FF
0028h	FF	??	??	FF
002Ch	FF	??	??	64
0030h	00	00	00	??
0034h	??	??	??	78
0038h	56	34	12	??
003Ch	??	??	??	??
0040h	??	??	??	??
0044h	??	61	62	63
0048h	64	65	66	67
004Ch	00	FF	02	41
0050h	41	41	61	6C
0054h	6C	61	68	00
0058h	61	6C	6C	61
005Ch	68	00	05	05
0060h	05	05	05	05
0064h	05	05	05	05
0068h	05	05	05	05
006Ch	05	05	05	05
0070h	05	05		

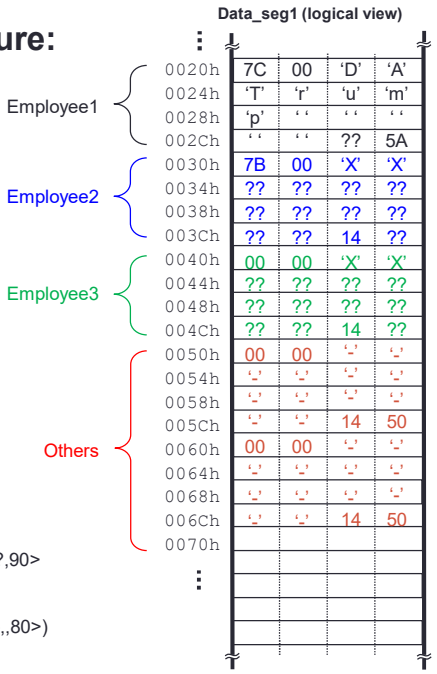
User Defined Type and Structure:

```
Structure name    STRUC
        .
        .     Sequence of DB, DW,
        .     and DD directives
        .
Structure name    ENDS
```

Example:

```
Personnel_data    STRUC
                ID          DW          0
                Initials    DB          'XX'
                Last_name   DB          10 Dup (?)
                Age         DB          20
                Weight      DB          ?
Personnel_data    ENDS

Data_seg1    SEGMENT
                ....
; Location counter =20h here
    Employee1   Personnel_data  <124, 'DA', 'Trump     ',?,90>
    Employee2   Personnel_data  <123,,,,>
    Employee3   Personnel_data  < >
    Others      Personnel_data  2 Dup (<,'- -',10 dup('-'),,80>)
Data_seg1    ENDS
```



User Defined Records:

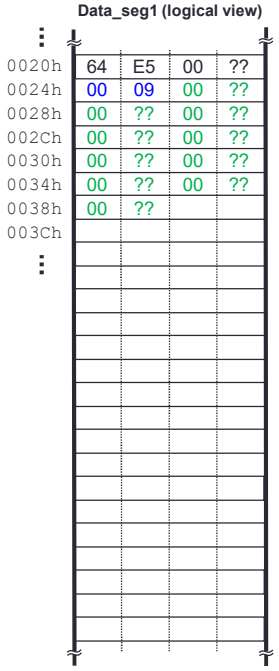
Record name RECORD field specification, …, field specification

Example:

```
Pattern   RECORD  OPCODE:8=0,Md:2=3,Reg:3,R_M:3
```

….

```
Data_seg1    SEGMENT
                ....
; Location Counter =20h here
    Instruction1   Pattern  <100, ,4,5>, <>
    Instruction2   Pattern  <,0,1,1>
    Program        Pattern  10 Dup(<>)
Data_seg1    ENDS
```



User Defined Type and Structure:

Example:

```
Pattern    RECORD   OPCODE:8=0,Md:2=3,Reg:3,R_M:3

Personnel_data    STRUC
            ID          DW          0
            Last_name DB          10 Dup (?)
            Age         DB          20
Personnel_data    ENDS

Data_seg1    SEGMENT
    Employees           Personnel_data 100  Dup (<1,10 dup(' '),>)
    New_Employee      Personnel_data <>
    Instr1                Pattern <>
    Instr2                Pattern <>
Data_seg1    ENDS
```

Now, we can write in our program:

```
            mov         New_employee.Age, AL
            add         Employees.ID[SI],1      ; SI can be incremented in a loop by 13 (structure size)
            mov         ax,Instr1
            or          ax, 1111011100111111B
            mov         Instr2,ax                   ; (Instr2) = 11110111 11 111 111B
```


Questions?