

EEM332 Lecture2.

History of Microprocessor

Introduction to Intel 8086/8088

Registers of 8086

Simple instructions

8086 Microprocessor

History

1959: first IC's

1971: μ P 4004	4Bit	Intel (~2000 Tr.)
1972: μ P 8008	8Bit	Intel (~5000 Tr.)
1974: μ P 8080	8Bit	Intel (~6000 Tr.)
1974: μ P 6800	8Bit	Motorola
1975: μ P Z80	8Bit	Zilog
1976: μ P 9900	16Bit	Texas Instruments
1978: μ P 8086	16Bit	Intel (50.000 Tr)
1982: μ P 8286	16Bit	Intel (100.000 Tr.)
1985: μ P 80386	16Bit	Intel (300.000 Tr.)
1989: μ P 80486	16Bit	Intel (10^6 Tr.)
1993: μ P 80586	16Bit	Intel ($5 \cdot 10^6$ Tr.)

...

Name	Date	Transistors	Microns	Clock speed	Data width	MIPS
8080	1974	6,000	6	2 MHz	8 bits	0.64
8088	1979	29,000	3	5 MHz	16 bits 8-bit bus	0.33
80286	1982	134,000	1.5	6 MHz	16 bits	1
80386	1985	275,000	1.5	16 MHz	32 bits	5
80486	1989	1,200,000	1	25 MHz	32 bits	20
Pentium	1993	3,100,000	0.8	60 MHz	32 bits 64-bit bus	100
Pentium II	1997	7,500,000	0.35	233 MHz	32 bits 64-bit bus	~300
Pentium III	1999	9,500,000	0.25	450 MHz	32 bits 64-bit bus	~510
Pentium 4	2000	42,000,000	0.18	1.5 GHz	32 bits 64-bit bus	~1,700

BRIEF HISTORY OF THE x86 FAMILY evolution from 8080/8085 to 8086

In 1978, Intel Corporation introduced the 16-bit 8086 microprocessor, a major improvement over the previous generation 8080/8085 series.

The 8086 capacity of 1 megabyte of memory exceeded the 8080/8085 maximum of 64K bytes of memory.

8080/8085 was an 8-bit system, which could work on only 8 bits of data at a time.

Data larger than 8 bits had to be broken into 8-bit pieces to be processed by the CPU.

8086 was a pipelined processor, as opposed to the nonpipelined 8080/8085

Table 1-1: Evolution of Intel's Microprocessors (from the 8008 to the 8088)

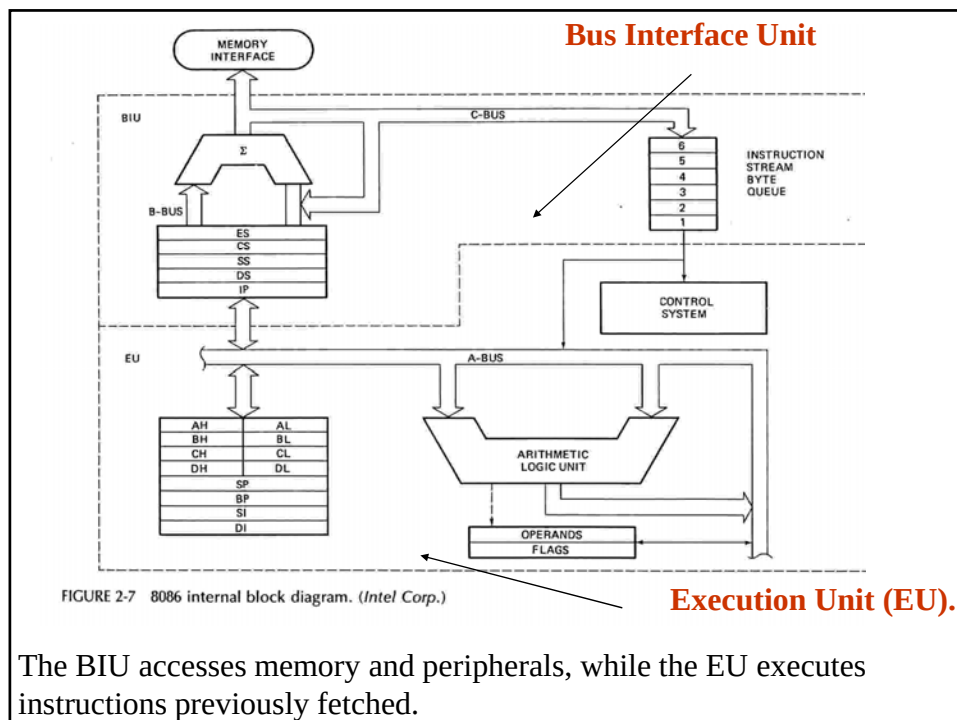
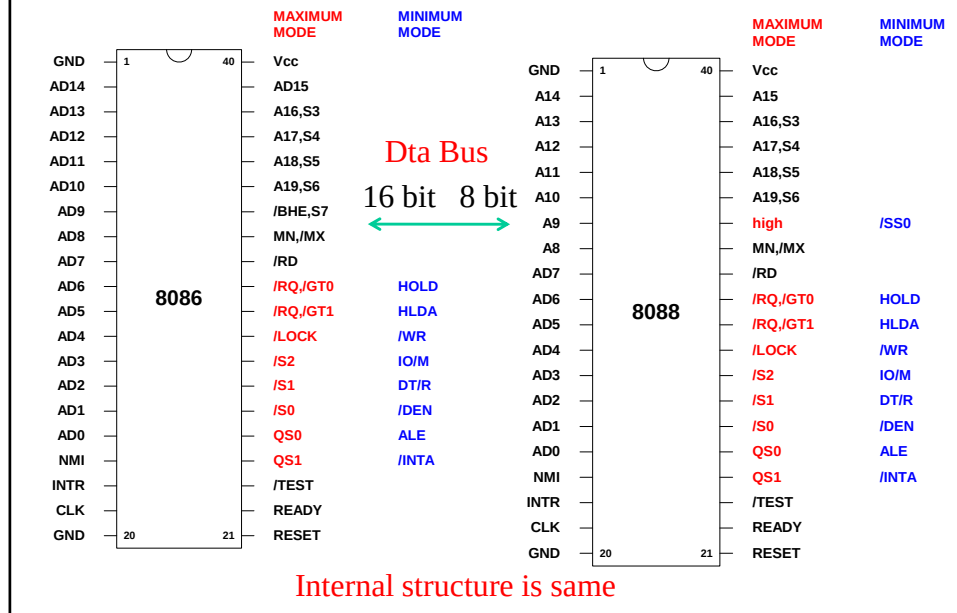
Product	8008	8080	8085	8086	8088
Year introduced	1972	1974	1976	1978	1979
Technology	PMOS	NMOS	NMOS	NMOS	NMOS
Number of pins	18	40	40	40	40
Number of transistors	3000	4500	6500	29,000	29,000
Number of instructions	66	111	113	133	133
Physical memory	16K	64K	64K	1M	1M
Virtual memory	None	None	None	None	None
Internal data bus	8	8	8	16	16
External data bus	8	8	8	16	8
Address bus	8	16	16	20	20
Data types	8	8	8	8/16	8/16

BRIEF HISTORY OF THE x86 FAMILY evolution from 8080/8085 to 8086

Table 1-2: Evolution of Intel's Microprocessors (from the 8086 to the Pentium Pro)

Product	8086	80286	80386	80486	Pentium	Pentium Pro
Year Introduced	1978	1982	1985	1989	1993	1995
Technology	NMOS	NMOS	CMOS	CMOS	BICMOS	BICMOS
Clock rate (MHz)	3–10	10–16	16–33	25–33	60, 66	150
Number of pins	40	68	132	168	273	387
Number of transistors	29,000	134,000	275,000	1.2 mill.	3.1 mill.	5.5 mill.
Physical memory	1M	16M	4G	4G	4G	64G
Virtual memory	None	1G	64T	64T	64T	64T
Internal data bus	16	16	32	32	32	32
External data bus	16	16	32	32	64	64
Address bus	20	24	32	32	32	36
Data types	8/16	8/16	8/16/32	8/16/32	8/16/32	8/16/32

The microprocessor is typically a single VLSI (Very Large-Scale Integration) chip



The BIU accesses memory and peripherals, while the EU executes instructions previously fetched.

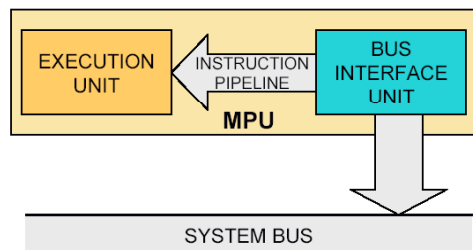
The 8086 has two parts, the **Bus Interface Unit (BIU)** and the **Execution Unit (EU)**.

The BIU fetches instructions, reads and writes data, and computes the 20-bit address.

The EU decodes and executes the instructions using the 16-bit ALU.

- **The BIU** contains the following registers:

IP - the Instruction Pointer
CS - the Code Segment Register
DS - the Data Segment Register
SS - the Stack Segment Register
ES - the Extra Segment Register



Software Model of the 8088/8086 Microprocessor

8088 microprocessor includes 13 16-bit internal registers.

- ❖ The instruction pointer, IP
- ❖ Four data registers, AX, BX, CX, DX
- ❖ Two pointer register, BP, SP
- ❖ Two index register, SI, DI
- ❖ Four segment registers, CS, DS, SS, ES

The status register, SR, with nine of its bits implement for **status and control flags**. The memory address space is 1 Mbytes and the I/O address space is 64 Kbytes in length.

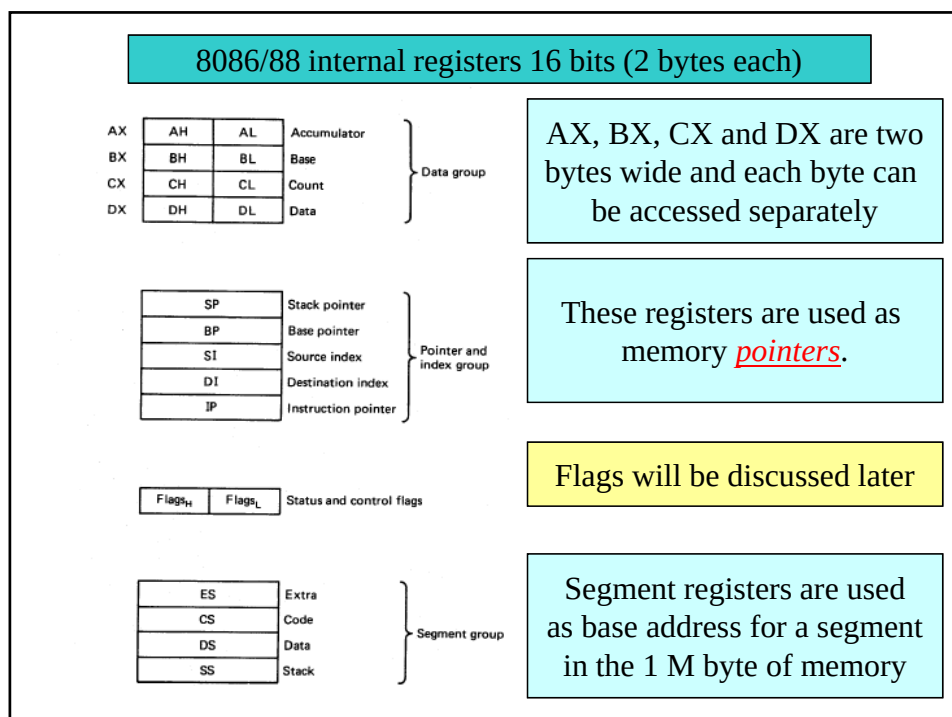
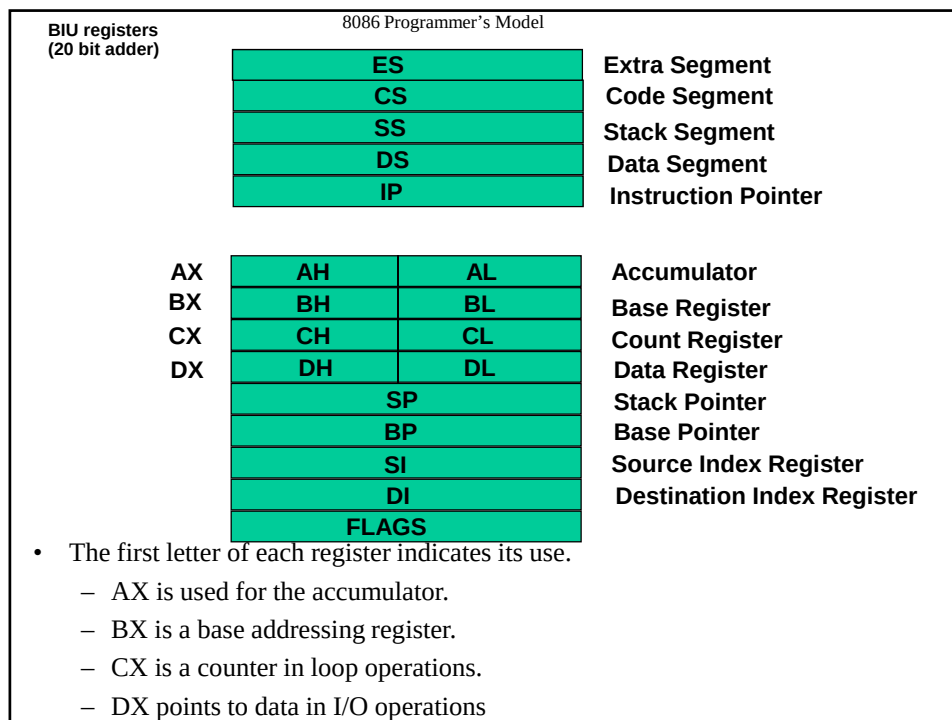


Table 1-4: Registers of the 8088/86/286 by Category

Category	Bits	Register Names
General	16	AX, BX, CX, DX
	8	AH, AL, BH, BL, CH, CL, DH, DL
Pointer	16	SP (stack pointer), BP (base pointer)
Index	16	SI (source index), DI (destination index)
Segment	16	CS (code segment), DS (data segment), SS (stack segment), ES (extra segment)
Instruction	16	IP (instruction pointer)
Flag	16	FR (flag register)

Note: The general registers can be accessed as the full 16 bits (such as AX), or as the high byte only (AH) or low byte only (AL).

**Data
Registers**

Register	Operations
AX	Word multiply, word divide, word I/O
AL	Byte multiply, byte divide, byte I/O, translate, decimal arithmetic
AH	Byte multiply, byte divide
BX	Translate
CX	String operations, loops
CL	Variable shift and rotate
DX	Word multiply, word divide, indirect I/O

The 8086/8088 Microprocessors: Registers

- **Registers**
 - Registers are in the CPU and are referred to by specific names
 - **Data registers**
 - Hold data for an operation to be performed
 - There are 4 data registers (AX, BX, CX, DX)
- Low and High bytes of the data registers (AX, BX, CX, DX) can be accessed separately
 - AH, BH, CH, DH are the high bytes
 - AL, BL, CL, and DL are the low bytes
 - **Address registers**
 - Hold the address of an instruction or data element
 - Segment registers (CS, DS, ES, SS)
 - Pointer registers (SP, BP, IP)
 - Index registers (SI, DI)
 - **Status register**
 - Keeps the current status of the processor
 - On an IBM PC the status register is called the FLAGS register
 - In total there are fourteen 16-bit registers in an 8086/8088

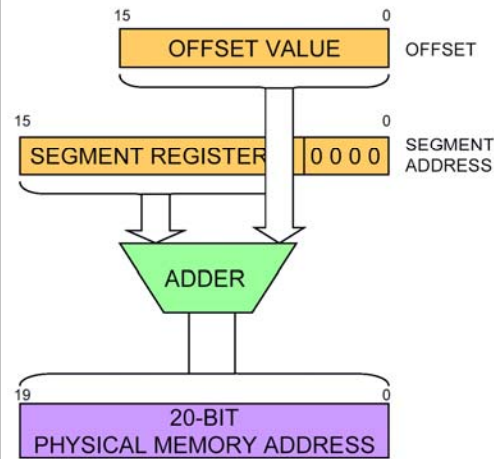
Pointer and Index Registers

- Contain the offset addresses of memory locations
- Can also be used in arithmetic and other operations
- **SP: Stack pointer**
 - Used with SS to access the stack segment
- **BP: Base Pointer**
 - Primarily used to access data on the stack
 - Can be used to access data in other segments
- **SI: Source Index register**
 - is required for some string operations
 - When string operations are performed, the SI register points to memory locations in the data segment which is addressed by the DS register. Thus, SI is associated with the DS in string operations.

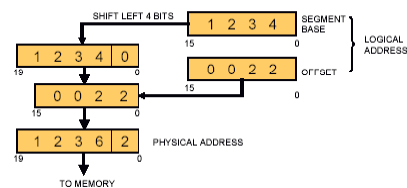
Segment Registers - CS, DS, SS and ES

- 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: Memory Segment
 - Block of 64K (65,536) consecutive memory bytes
 - A segment number is a 16 bit number
 - 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

Generating a Memory Address



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



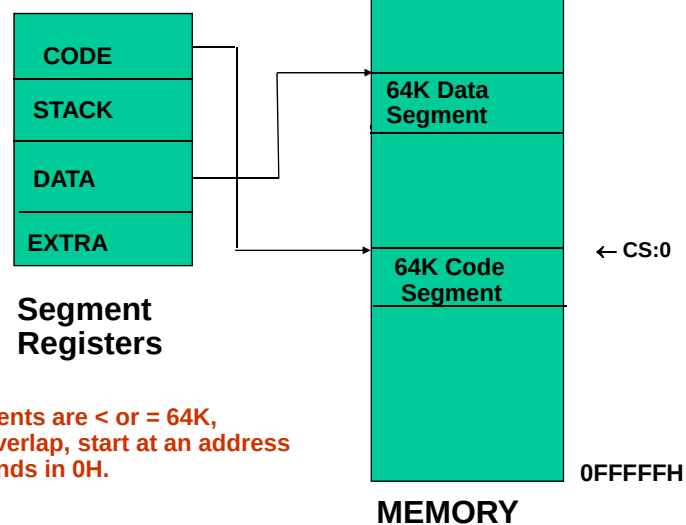
Generating a physical address

Offset Registers for various Segments

Segment register	CS	DS	ES	SS
Offset register(s)	IP	SI, DI, BX	SI, DI, BX	SP, BP

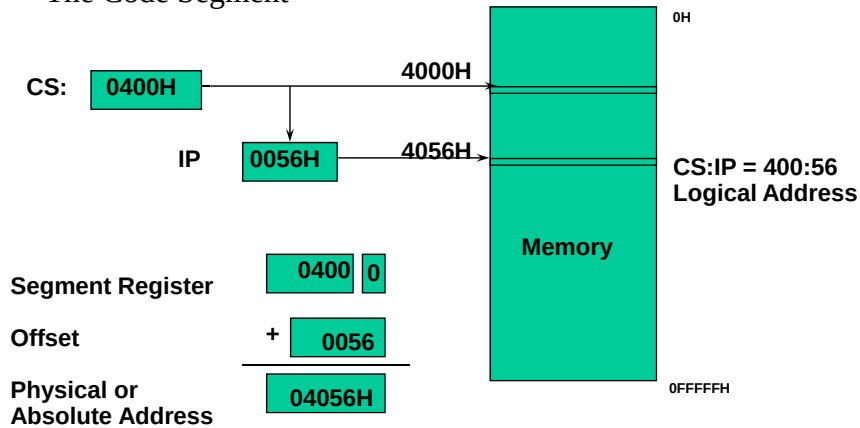
Segments

Segment Starting address is segment register value shifted 4 places to the left.



Segments are $\leq 64K$, can overlap, start at an address that ends in 0H.

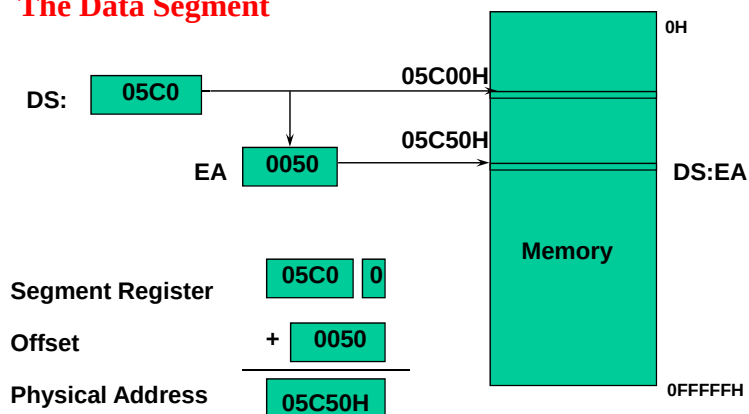
The Code Segment



The offset is the distance in bytes from the start of the segment.
The offset is given by the IP for the Code Segment.
Instructions are always fetched with using the CS register.

The physical address is also called the absolute address.

The Data Segment

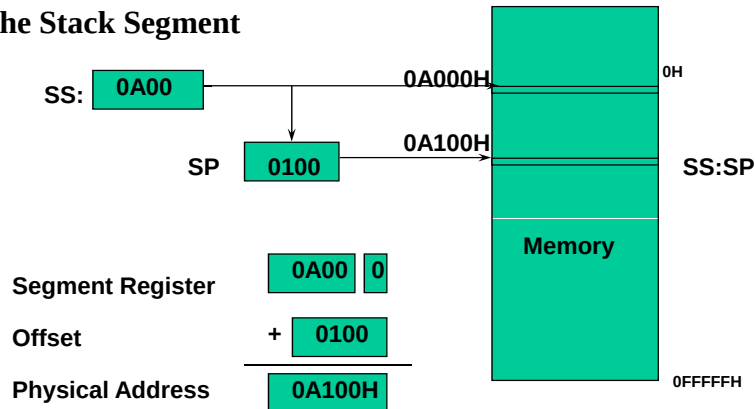


Data is usually fetched with respect to the DS register.
The effective address (EA) is the offset.
The EA depends on the addressing mode.

Offset Registers for various Segments

Segment register	CS	DS	ES	SS
Offset register(s)	IP	SI, DI, BX	SI, DI, BX	SP, BP

The Stack Segment



The offset is given by the SP register.

The stack is always referenced with respect to the stack segment register.

The stack grows toward decreasing memory locations.

The SP points to the last or top item on the stack.

- Logical Address is specified as **segment:offset**
- Physical address is obtained by shifting the segment address 4 bits to the left and adding the offset address
- Thus the physical address of the logical address A4FB:4872 is $A4FB0 + 4872 = A9822$

Example 1-1

If CS = 24F6H and IP = 634AH, show (a) the logical address, and (b) the offset address. Calculate (c) the physical address, (d) the lower range, and (e) the upper range of the code segment.

Solution:

- (a) 24F6:634A (b) 634A (c) 2B2AA (24F60 + 634A)
 (d) 24F60 (24F60 + 0000) (e) 34F5F (24F60 + FFFF)

Example 1-3

If DS = 7FA2H and the offset is 438EH, calculate (a) the physical address, (b) the lower range, and (c) the upper range of the data segment. Show (d) the logical address.

Solution:

- (a) 83DAE (7FA20 + 438E) (b) 7FA20 (7FA20 + 0000)
 (c) 8FA1F (7FA20 + FFFF) (d) 7FA2:438E

Example 1-4

Assume that the DS register is 578C. To access a given byte of data at physical memory location 67F66, does the data segment cover the range where the data resides? If not, what changes need to be made?

Solution:

No, since the range is 578C0 to 678BF, location 67F66 is not included in this range. To access that byte, DS must be changed so that its range will include that byte.

INTRODUCTION TO PROGRAM SEGMENTS

A typical Assembly language program consists of at least three segments:

A code segment - which contains the Assembly language instructions that perform the tasks that the program was designed to accomplish.

A data segment - used to store information (data) to be processed by the instructions in the code segment.

A stack segment - **used by the CPU to store information temporarily.**

The stack is a section of RAM,

A *segment* is an area of memory that includes up to 64K bytes, and begins on an address evenly divisible by 16 (such an address ends in 0H)

8085 addressed a maximum of 64K of physical memory, since it had only 16 pins for address lines. ($2^{16} = 64K$)

Limitation was carried into 8088/86 design for compatibility.

In 8085 there was 64K bytes of memory for all code, data, and stack information.

In 8088/86 there can be up to 64K bytes in each category.

The code segment, data segment, and stack segment.

The Stack

- The **stack** is implemented for **temporary storage** of information such as data or addresses.
- The stack is 64KBytes long and is organized from a software point of view as 32K words.
- The contents of the **SP and BP registers** are used as offsets into the stack segment memory while the
- segment base value is in the SS register. Push instructions (**PUSH**) and pop instructions (**POP**)
- Top of the stack** (TOS) and **bottom of the stack** (BOS)
- The 8088 can push **word-wide data** and address information onto the stack from registers or memory.
- Many stacks can exist but only one is active at a time.
 - As data is *pushed onto* the stack it is **decremented**.
 - As data is *popped off* the stack into the CPU, it is **incremented**.

Example

- If DS=7FA2H and the offset is 438EH

a) Calculate the physical address

$$7FA20 + 438E = 83DAE$$

b) calculate the lower range

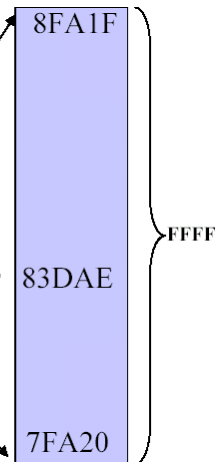
$$7FA20 + 0000 = 7FA20$$

c) Calculate the upper range of the data segment

$$7FA20 + FFFF = 8FA1F$$

d) Show the logical Address

7FA2:438E



16 bit Segment Register Assignments

Type of Memory Reference	Default Segment	Alternate Segment	Offset
Instruction Fetch	CS	none	IP
Stack Operations	SS	none	SP, BP
General Data	DS	~~CS~~ , ES, SS	BX, address
String Source	DS	~~CS~~ , ES, SS	SI, DI, address
String Destination	ES	None	DI

Examples of ass. Language for 8086

Examples

- MOV BL,AL
- Opcode for MOV = 100010
- We'll encode AL so
 - D = 0 (AL source operand)
- W bit = 0 (8-bits)
- MOD = 11 (register mode)
- REG = 000 (code for AL)
- R/M = 011

opcode	D	W	MOD	REG	R/M
--------	---	---	-----	-----	-----

MOV AL,BL = 10001000 11000011 = 88 C3h

ADD AX,[SI] = 00000011 00000100 = 03 04 h

ADD [BX],[DI] + 1234h, AX = 00000001 10000001 ____ h
= 01 81 34 12 h

MOV Instruction

- MOV destination,source
 - 8 bit moves
 - MOV CL,55h
 - MOV DL,CL
 - MOV BH,CL
 - Etc.
 - 16 bit moves
 - MOV CX,468Fh
 - MOV AX,CX
 - MOV BP,DI
 - Etc.

MOV Instruction

- Data can be moved among all registers but data cannot be moved directly into the segment registers (CS,DS,ES,SS).
 - To load as such, first load a value into a nonsegment register and then move it to the segment register

MOV AX,2345h

MOV DS,AX

- Moving a value that is too large into a register will cause an error

MOV BL,7F2h ; illegal

MOV AX,2FE456h ; illegal

- If a value less than FFh is moved into a 16 bit register. The rest of the bits are assumed to be all zeros.

MOV BX,5 ; BX = 0005 with BH = 00 and BL = 05

INTRODUCTION TO ASSEMBLY PROGRAMMING

Example for MOV instruction:

This program first loads CL with value 55H, then moves this value around to various registers inside the CPU.

```
MOV CL,55H ;move 55H into register CL
MOV DL,CL ;copy the contents of CL into DL (now DL=CL=55H)
MOV AH,DL ;copy the contents of DL into AH (now AH=DL=55H)
MOV AL,AH ;copy the contents of AH into AL (now AL=AH=55H)
MOV BH,CL ;copy the contents of CL into BH (now BH=CL=55H)
MOV CH,BH ;copy the contents of BH into CH (now CH=BH=55H)
```

The use of 16-bit registers is shown here:

```
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
```

```
MOV AX,58FCH ;move 58FCH into AX (LEGAL)
MOV DX,6678H ;move 6678H into DX (LEGAL)
MOV SI,924BH ;move 924B into SI (LEGAL)
MOV BP,2459H ;move 2459H into BP (LEGAL)
MOV DS,2341H ;move 2341H into DS (ILLEGAL)
MOV CX,8876H ;move 8876H into CX (LEGAL)
MOV CS,3F47H ;move 3F47H into CS (ILLEGAL)
MOV BH,99H ;move 99H into BH (LEGAL)
```

Values cannot be loaded directly into any segment register (CS, DS, ES, or SS).

To load a value into a segment register, load it to a non-segment register, then move it to the segment register

```
MOV AX,2345H ;load 2345H into AX
MOV DS,AX ;then load the value of AX into DS

MOV DI,1400H ;load 1400H into DI
MOV ES,DI ;then move it into ES, now ES=DI=1400
```

ADD Instruction

- ADD destination,source
- The ADD instruction tells the CPU to add the source and destination operands and put out the results in the destination

MOV AL,25H

MOV BL,34h

ADD AL,BL ; (AL should read 59h once the instruction is executed)

MOV DH,25H

ADD DH,34h ; (AL should read 59h once the instruction is executed)

Immediate operand



- Assume that a program is written to add 5 bytes of data 25h,12h,15h,1Fh, and 2Bh.
- One way to do it

```
MOV AL,00h
ADD AL, 25h
ADD AL, 12h
ADD AL,15h
ADD AL,1Fh
ADD AL,2Bh
```
- Data and code are mixed in the instructions here
- The problem with it is if the data changes, the code must be searched for every place the data is included and data retyped.
- It is a good idea then to set aside an area of memory strictly for data

Data Segment

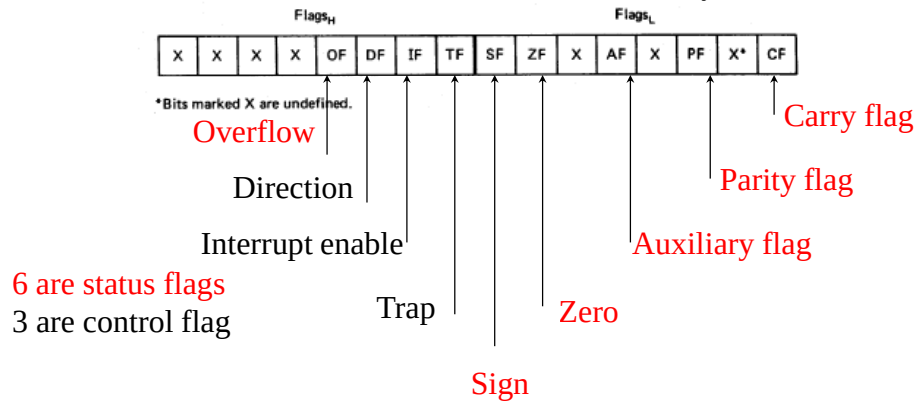
- The data is first placed in the memory locations
DS:0200 = 25h
DS:0201 = 12h
DS:0202 = 15h
DS:0203 = 1Fh
DS:0204 = 2Bh
- Then the program is written as
MOV AL,0
ADD AL,[0200] ; bracket means add the contents of DS:0200 to AL
ADD AL,[0201]
ADD AL,[0202]
ADD AL,[0203]
ADD AL,[0204]
- If the data is stored at a different offset address, say 450 h, the program need to be rewritten

Data Segment

- The term pointer is used for a register holding an offset address
- Use BX as a pointer
MOV AL,0
MOV BX,0200h
ADD AL,[BX]
INC BX
ADD AL,[BX]
INC BX
ADD AL,[BX]
INC BX
ADD AL,[BX]
INC BX
ADD AL,[BX]
- If the offset address of data is to be changed, only one instruction will need to be modified

Flag register

Many Assembly language instructions alter flag register bits & some instructions function differently based on the information in the flag register. The flag register is a 16-bit register sometimes referred to as the *status register*. Although 16 bits wide, only some of the bits are used. The rest are either undefined or reserved by Intel.



Status flags (called *conditional flags*): They are set according to some results of **arithmetic operation**. You do not need to alter the value yourself.

Control flags: Used to control some operations of the MPU. These flags are to be set by you in order to achieve some specific purposes.

CF, the Carry Flag: This flag is set whenever there is a carry out, either from d7 after an 8-bit operation, or from d15 after a 16-bit data operation.

PF, the Parity Flag: After certain operations, the parity of the result's **low-order byte** is checked. If the byte has an **even number of 1s**, the **parity flag is set to 1**; otherwise, it is cleared.

AF, the Auxiliary Carry Flag: If there is a carry **from d3 to d4 of an operation** this bit is set to 1, otherwise cleared (set to 0).

ZF, the Zero Flag: The ZF is set to 1 if the result of the arithmetic or logical operation is zero, otherwise, it is cleared (set to 0).

SF, the Sign Flag: MSB is used as the sign bit of the binary representation of the signed numbers. After arithmetic or logical operations the MSB is copied into SF to indicate the sign of the result.

TF, the Trap Flag: When this flag is set it allows the program to **single step**, meaning to execute one instruction at a time. Used for debugging purposes.

IF, Interrupt Enable Flag: This bit is set or **cleared to enable or disable only the external interrupt requests**.

DF, the Direction Flag: This bit is used to control the direction of **the string operations**.

OF, the Overflow Flag: This flag is set whenever the result of a signed number operation is too large, causing the high-order bit to overflow into the sign bit.

Not: the zero is used flag to implement program loops.



- Six of the flags are status indicators reflecting properties of the last arithmetic or logical instruction.
- For example, if register AL = 7Fh and the instruction ADD AL, 1 is executed then the following happen
 - AL = 80h
 - CF = 0; there is no carry out of bit 7
 - PF = 0; 80h has an odd number of ones
 - AF = 1; there is a carry out of bit 3 into bit 4
 - ZF = 0; the result is not zero
 - SF = 1; bit seven is one
 - OF = 1; the sign bit has changed
- Can be used to transfer program control to a new memory location

The status register, also called the flags register, indicate conditions that are produced as the result of executing an instruction. Only nine bits of the register are implemented. Six of these bits represent **status flags** and the other three bits represent **control flags**. The 8088 provides instructions within its instruction set that are able to use these flags to alter the sequence in which the program is executed.

TF, IF, and DF

- Three of the flags can be set or reset directly by the programmer and are used to control the operation of the microprocessor, these are TF, IF, and DF.
- When TF (Trap Flag) is set, control is passed to special address after each instruction is executed. Normally a program to display all the registers and flags is stored there. Single-stepping mode.
- When IF (Interrupt Flag) is set, external interrupt requests on the 8086's interrupt line INTR will be enabled.
 - For example a printer may spend several seconds printing a page of text from its internal buffer
 - When it is ready for new data, the printer control circuit drives the 8086's INTR input line
 - The processor then suspends whatever it is doing and begins running the printer *interrupt service routine* (ISR)
 - When the routine has finished via a IRET (interrupt return) instruction control is transferred back to the original instruction in the main program that was executing when the interrupt occurred
 - Hardware and software interrupts
- Direction Flag (DF) is used with block move instructions.
 - DF = 1 then the block memory pointer will automatically decrement
 - DF = 0, then the block memory pointer will automatically increment

Example 1-11

Show how the flag register is affected by

```
MOV  AL, 9CH      ;AL=9CH
MOV  DH, 64H      ;DH=64H
ADD  AL, DH        ;now AL=0
```

Solution:

```

    9C          1001  1100
+   64          0110  0100
-----
    00          0000  0000
```

CF = 1 since there is a carry beyond d7

ZF = 1 since the result is zero

AF = 1 since there is a carry from d3 to d4

SF = 0 since d7 of the result is zero

PF = 1 since there is an even number of 1s in the result

Example 1-12

Show how the flag register is affected by

```
MOV  AX, 34F5H    ;AX= 34F5H
ADD  AX, 95EBH    ;now AX= CAE0H
```

Solution:

```

    34F5          0011  0100  1111  0101
+   95EB          1001  0101  1110  1011
-----
    CAE0          1100  1010  1110  0000
```

CF = 0 since there is no carry beyond d15

ZF = 0 since the result is not zero

AF = 1 since there is a carry from d3 to d4

SF = 1 since d15 of the result is one

PF = 0 since there is an odd number of 1s in the lower byte

Example 1-12

Show how the flag register is affected by

```
MOV  AX, 34F5H    ;AX= 34F5H
ADD  AX, 95EBH    ;now AX= CAE0H
```

Solution:

```

    34F5          0011  0100  1111  0101
+   95EB          1001  0101  1110  1011
-----
    CAE0          1100  1010  1110  0000
```

CF = 0 since there is no carry beyond d15

ZF = 0 since the result is not zero

AF = 1 since there is a carry from d3 to d4

SF = 1 since d15 of the result is one

PF = 0 since there is an odd number of 1s in the lower byte

Example 1-13

Show how the flag register is affected by

```
MOV  BX, AAAAH    ;BX= AAAAH
ADD  BX, 5556H    ;now BX= 0000H
```

Solution:

```

    AAAA          1010  1010  1010  1010
+   5556          0101  0101  0101  0110
-----
    0000          0000  0000  0000  0000
```

CF = 1 since there is a carry beyond d15

ZF = 1 since the result is zero

AF = 1 since there is a carry from d3 to d4

SF = 0 since d15 of the result is zero

PF = 1 since there is an even number of 1s in the lower byte

Show how the flag register is affected by

```
MOV AX, 94C2H ;AX=94C2H
MOV BX, 323EH ;BX=323EH
ADD AX, BX ;now AX=C700H
MOV DX, AX ;now DX=C700H
MOV CX, DX ;now CX=C700H
```

Solution:

94C2	1001	0100	1100	0010
+ 323E	0011	0010	0011	1110
C700	1100	0111	0000	0000

After the ADD operation, the following are the flag bits:

CF = 0 since there is no carry beyond d15 ZF = 0 since the result is not zero

AF = 1 since there is a carry from d3 to d4 SF = 1 since d15 of the result is 1

PF = 1 since there is an even number of 1s in the lower byte

- As an example, to add 5 bytes of data, a *counter* can be used to keep track of how many times the loop needs to be repeated.

```
MOV CX, 05 ;CX holds the loop count
MOV BX, 0200H ;BX holds the offset data address
MOV AL, 00 ;initialize AL
ADD_LP: ADD AL, [BX] ;add the next byte to AL
INC BX ;increment the data pointer
DEC CX ;decrement the loop counter
JNZ ADD_LP ;jump to next iteration if counter
not zero
```

Each time the addition is performed the counter is decremented and the zero flag is checked. When the counter becomes zero, the zero flag is set (**ZF = 1**) and the loop is stopped.