Week 2
The 80x86 Microprocessor Architecture

Brief History of the 80x86 Family

- Evolution from 8080/8085 to 8086
  - In 1987, Intel introduced a 16-bit microprocessor called the 8086
  - It was a major improvement over the previous generation 8080/8085 microprocessors
    - 1 Mbyte memory (20 address lines) vs 8080/8085's capability of 64 Kbytes
    - 8080/8085 was an 8 bit system, meaning that the data larger than 8 bits should be broken into 8-bit pieces to be processed by the CPU; in contrast 8086 is a 16 bit microprocessor
    - 8086 is pipelined vs nonpipelined 8080/8085; in a system with pipelining the data and address busses are busy transferring data while the CPU is processing information
- Evolution from 8086 to 8088
  - 8086 is a microprocessor with a 16-bit data bus internally and externally
  - Internal because all registers are 16 bits wide
  - External because the data bus was 16 bits to transfer data in and out of the CPU
  - There was a resistance in using the 16 bit external data bus since at that time peripherals were designed around 8-bit microprocessors
  - Intel then came out with the 8088 version with 8-bit data bus

Brief History - Continued

- Success of 8088
  - IBM picked up the 8088 as their microprocessor of choice in designing the IBM PC
  - All specification of the hardware and software of the PC are made public by IBM and Microsoft (in contrast with Apple computers)
- Other microprocessors: 80386, 80386, 80486
  - Intel introduced 80286 in 1982
  - 16 bit internal and external data buses
  - 24 address lines (16 Mbyte main memory)
  - Virtual memory: a way of fooling the microprocessor into thinking that it has access to almost unlimited amount of memory by swapping data between disk storage and RAM
  - Real mode vs protected mode
  - Intel unveiled the 80386 (sometimes called the 80386DX) in 1985; internally and externally a 32 bit microprocessor with a 32 bit address bus (4 Gbyte physical memory)
  - Numeric data processing chips were made available: 8087, 80287, 80387 etc.

Evolution of Intel's microprocessors

Processor	Date of Introduction	Transistors on Chip	Maximum MIPS at Introduction[a]	Maximum Clock Frequency at Introduction[b]	On-chip Cache Memory	Maximum Addressable Memory
8086	1978	29K	0.8	8 MHz		1 MB
80286	1982	134K	2.7	12.5 MHz		16 MB
80386	1985	275K	6	20 MHz		4 GB
80486	1989	1.2M	20	25 MHz[c]	8K Level 1	4 GB
Pentium	1993	3.1M	100	60MHz	16K Level 1	4 GB
Pentium Pro	1995	5.5M	440	200 MHz	16K Level 1, 256K/ 512K Level 2	64 GB
Pentium II	1997	7M	466	266 MHz	32K Level 1, 256/ 512K Level 2	64 GB
PentiumIII	1999	8.2M	1,000	500 MHz	32K Level 1, 512K Level 2	64 GB

Virtual 8086 Mode

- Real Mode
  - Only one program can be run one time
  - All of the protection and memory management functions are turned off
  - Memory space is limited to 1MB
- Virtual 8086 Mode
  - The 386 hands each real mode program its own 1MB chunk of memory
  - Multiple 8086 programs to be run simultaneously but protected from each other (multiple MSDOS prompts)
  - Due to time sharing, the response becomes much slower as each new program is launched
  - The 386 can be operated in Protected Mode and Virtual 8086 mode at the same time.
  - Because each 8086 task is assigned the lowest privilege level, access to programs or data in other segments is not allowed thus protecting each task.
  - We'll be using the virtual 8086 mode in the lab experiments on PCs that do have either Pentiums or 486s.

The 80286 and above - Modes of Operation

- Real Mode
  - The address space is limited to 1MB using address lines A0-19; the high address lines are inactive
  - The segmented memory addressing mechanism of the 8086 is retained with each segment limited to 64KB
  - Two new features are available to the programmer
    - –Access to the 32 bit registers
    - –Addition of two new segments F and G

- Protected Mode
  - –Difference is in the new addressing mechanism and protection levels
  - –Each memory segment may range from a single byte to 4GB
  - –The addresses stored in the segment registers are now interpreted as pointers into a descriptor table
  - –Each segment's entry in this table is eight bytes long and identifies the base address of the segment, the segment size, and access rights
  - –In 8088/8086 any program can access the core of the OS hence crash the system. Access Rights are added in descriptor tables.

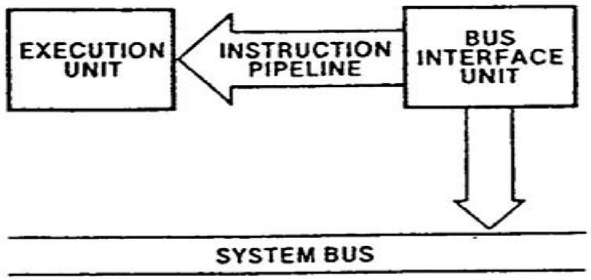
Virtual Memory

- 286 onward supported Virtual Memory Management and Protection™
- Unlimited amount of main memory assumed
- Two methods are used:
  - Segmentation
  - Paging
- Both techniques involve swapping blocks of user memory with hard disk space as necessary
  - If the program needs to access a block of memory that is indicated to be stored in the disk, the OS searches for an available memory block (typically using a least recently used algorithm) and swaps that block with the desired data on the hard drive
  - Memory swapping is invisible to the user
  - Segmentation: the block size is variable ranging up to 4GB
  - Paging: Block sizes are always 4 KB at a time.
- A final protected mode feature is the ability to assign a privilege level to individual tasks (programs). Tasks of lower privilege level cannot access programs or data with a higher privilege level. The OS can run multiple programs each protected from each other.

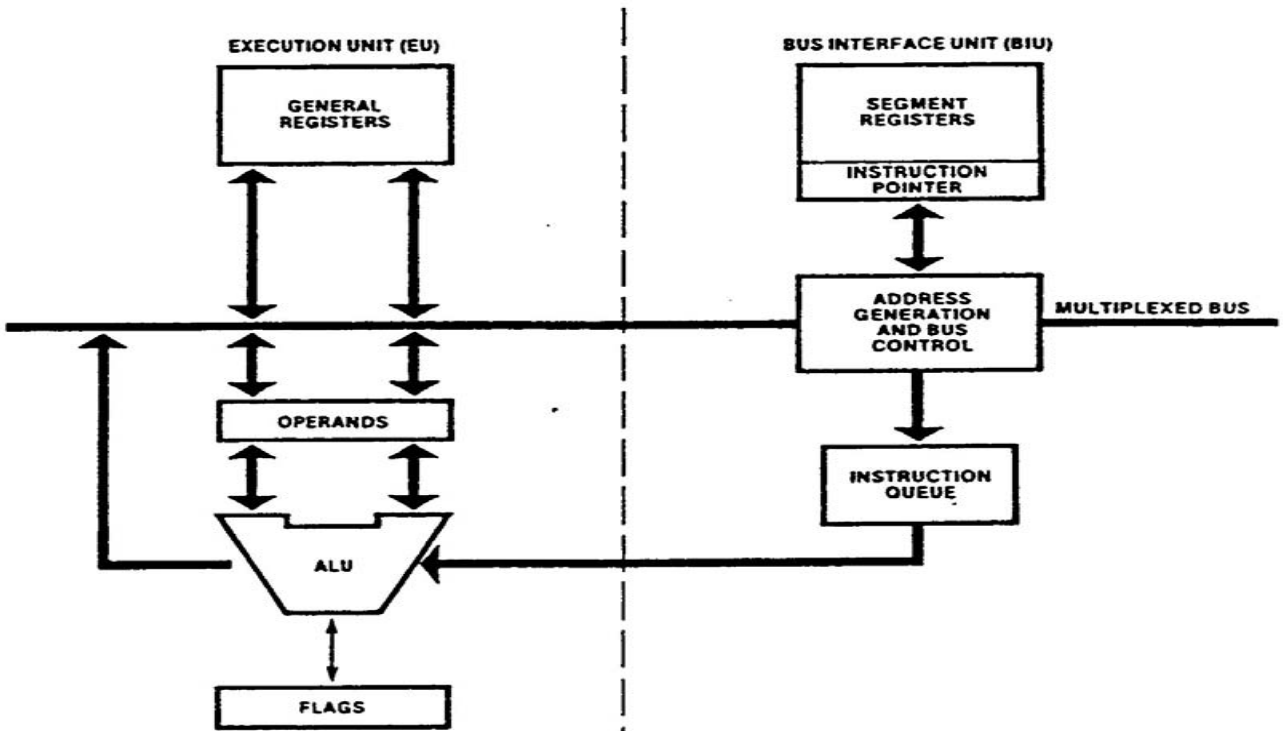
The 8086 and 8088

- The 8086 microprocessor represents the foundation upon which all the 80x86 family of processors have been built
- Intel has made the commitment that as new generations of microprocessors are developed, each will maintain software compatibility with this first generation part.
  - For example, a program designed to run on an **Intel** 386 microprocessor, which also runs on a Pentium, is **upward** compatible.
- **Processor model**
  - BIU (Bus Interface Unit) provides hardware functions including generation of the memory and I/O addresses for the transfer of data between itself and the outside world
  - EU (Execution Unit) receives program instruction codes and data from the BIU executes these instructions and stores the results in the general registers.
  - EU has no connection to the system busses; it receives and outputs all its data through the BIU.

Execution and Bus Interface Units



(a)



(b)

Fetch and Execute Cycle

- Fetch and execute cycles overlap
  - BIU outputs the contents of the IP onto the address bus
  - Register IP is incremented by one or more than one for the next instruction fetch
  - Once inside the BIU, the instruction is passed to the queue; this queue is a first-in-first-out register sometimes likened to a pipeline
  - Assuming that the queue is initially empty the EU immediately draws this instruction from the queue and begins execution
  - While the EU is executing this instruction, the BIU proceeds to fetch a new instruction.
    - BIU will fill the queue with several new instructions before the EU is ready to draw its next instruction
  - The cycle continues with the BIU filling the queue with instructions and the EU fetching and executing these instructions

Pipelined Architecture

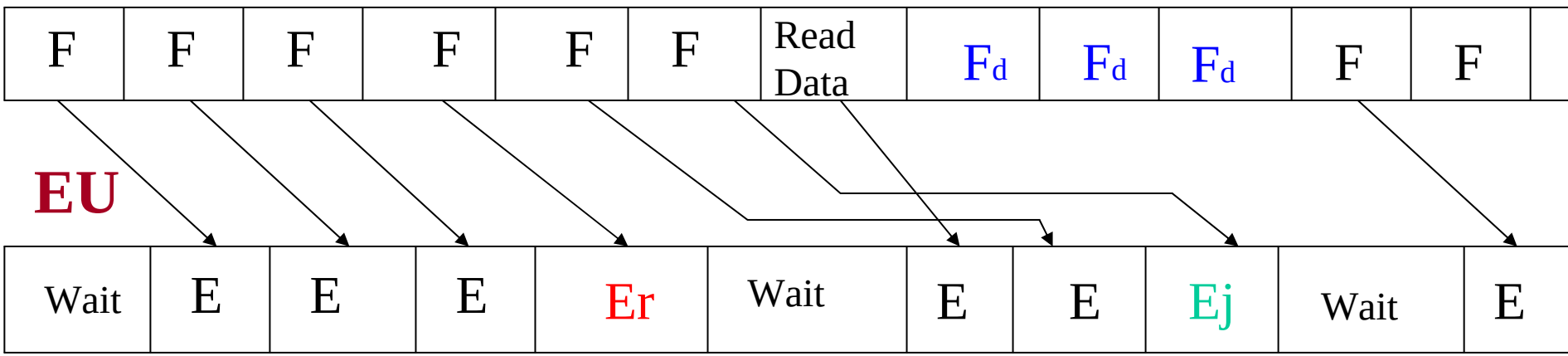
- Three conditions that will cause the EU to enter a wait mode
  - when the instruction requires access to a memory location not in the queue
  - when the instruction to be executed is a jump instruction; the instruction queue should be flushed out (known as branch penalty too much jumping around reduces the efficiency of the program)
  - during the execution of slow instructions
    - for example the instruction AAM (ASCII Adjust for Multiplication) requires 83 clock cycles to complete for an 8086
- 8086 vs 8088
  - BIU data bus width 8 bits for 8088, BIU data bus width 16 bits for 8086
  - 8088 instruction queue is four bytes instead of six
  - 8088 is found to be 30% slower than 8086
    - WHY
  - Long instructions provide more time for the BIU to fill the queue

Nonpipelined vs pipelined architecture

Non-pipelined architecture

BIU

EU



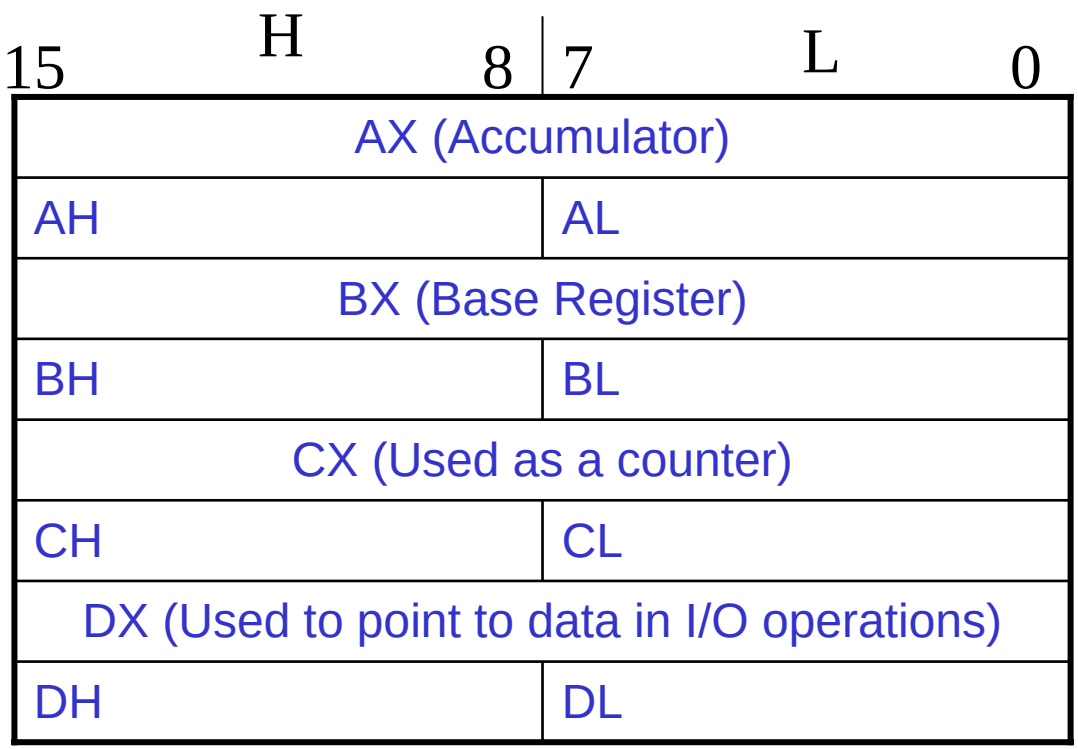
Pipelined architecture

Er: a request for data not in the queue
Ej: jump instruction occurs
Fd: Discarded

Registers of the 8086/80286 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), Base Pointer (BP)
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)

General Purpose Registers



- **Data Registers are normally used for storing temporary results that will be acted upon by subsequent instructions**
- **Each of the registers is 16 bits wide (AX, BX, CX, DX)**
- **General purpose registers can be accessed as either 16 or 8 bits e.g., AH: upper half of AX, AL: lower half of AX**

Data Registers

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

Pointer and Index Registers

SP	**Stack Pointer**
BP	**Base Pointer**
SI	**Source Index**
DI	**Destination Index**
IP	**Instruction Pointer**

The registers in this group are all 16 bits wide

Low and high bytes are not accessible

These registers are used as memory pointers

- Example: MOV AH, [SI]

Move the byte stored in memory location
whose address is contained in register SI to register AH

IP is not under direct control of the programmer

Computer Programming

- Machine Language vs Assembly Language
  - Machine language or object code is the only code a computer can execute but it is nearly impossible for a human to work with
  - E4 27 88 C3 E4 27 00 D8 E6 30 F4 the object code for adding two numbers input from the keyboard
- When programming a microprocessor, programmers often use assembly language
  - This involves 3-5 letter abbreviations for the instruction codes (mnemonics) rather than the binary or hex object codes

Address	Hex Object Code				Mnemonics		Comment
					Op-Code	Operand	
0100	E4	27			IN	AL,27H	Input first number from port 27H and store in AL
0102	88	C3			MOV	BL,AL	Save a copy of register AL in register BL
0104	E4	27			IN	AL,27H	Input second number to AL
0106	00	D8			ADD	AL,BL	Add contents of BL to AL and store the sum in AL
0107	E6	30			OUT	30H,AL	Output AL to port 30H
0109	F4				HLT		Halt the computer

Source code

Edit, Assemble, Test, and Debug Cycle

- Using an *editor*, the source code of the program is created. This means selecting the appropriate instruction mnemonics to accomplish the task
- A compiler program which examines the source code file generated by the editor and determines the object code for each instruction in the program, is then run. In assembly language programming, this is called an *assembler* (MASM (Chapter 2 of the textbook, DEBUG: Appendix A of the textbook, etc.,)
- The object code produced by the computer is loaded into the target computer's memory and is then *run.*
- *Debugging:* locating and fixing the source of error
- High-level programming Languages
  - Basic, Pascal, C, C++

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 non-segment 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 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
```

MOV Instruction

• MOV AX,58FCH	√
• MOV DX,6678H	√
• MOV SI,924BH	√
• MOV BP,2459H	√
• MOV DS,2341H	x
• MOV CX,8876H	√
• MOV CS,3F47H	x
• MOV BH,99H	√

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

DESTINATION = DESTINATION + SOURCE

```
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

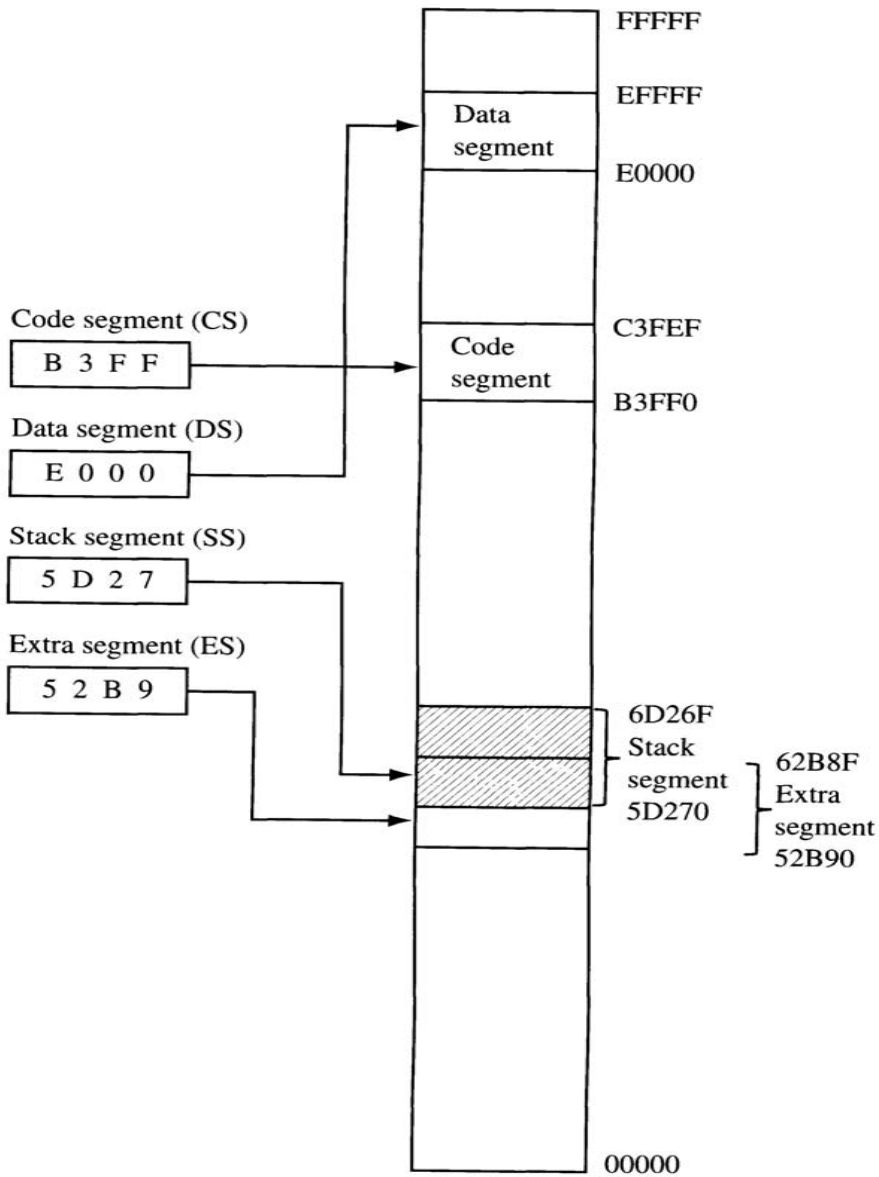
Origin and Definition of a Segment

- A segment is an area of memory that includes up to 64 Kbytes and begins on an address divisible by 16 (such an address ends with an hex digit 0h or 0000b)
  - 8085 could address 64Kbytes 16 address lines
- In the 8085, 64 K is for code, data, and stack
- In the 8086/88, 64 K is assigned to each category
  - Code segment
  - Data segment
  - Stack Segment
  - Extra Segment

Advantages of Segmented Memory

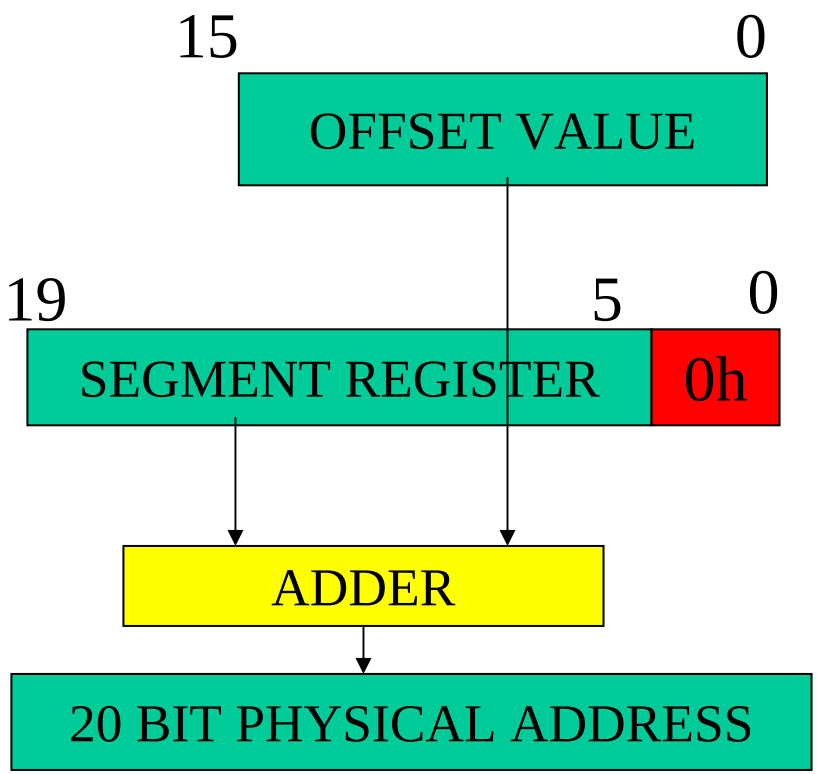
- One program can work on several different sets of data. This is done by reloading register DS to a new value.
- Programs that reference logical addresses can be loaded and run anywhere in the memory: relocatable
- Segmented memory introduces extra complexity in both hardware in that memory addresses require two registers.
- They also require complexity in software in that programs are limited to the segment size
- Programs greater than 64 KB can be run on 8086 but the software needed is more complex as it must switch to a new segment.
- Protection among segments is provided.

Segment Registers



Logical and Physical Addresses

- Addresses within a segment can range from address 0 to address FFFFh. This corresponds to the 64Kbyte length of the segment called an *offset*
- An address within a segment *logical address*
- *Ex.* Logical address 0005h in the code segment actually corresponds to B3FF0h + 5 = B3FF5h.



Example 1:
Segment base value: 1234h
Offset: 0022h

12340h
+ 0022h

12362h is the physical 20 bit address

Two different logical addresses may correspond to the same physical address.
D470h in ES 2D90h in SS
ES:D470h SS:2D90h

Example

•If DS=7FA2H and the offset is 438EH

a) Calculate the physical address

7FA20 + 438E = 83DAE

b) calculate the lower range

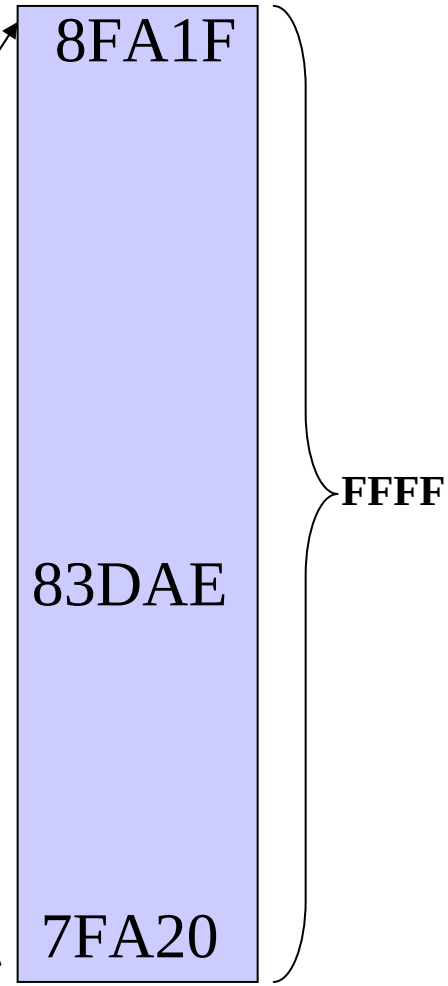
7FA20 + 0000 = 7FA20

c) Calculate the upper range of the data segment

7FA20 + FFFF = 8FA1F

d) Show the logical Address

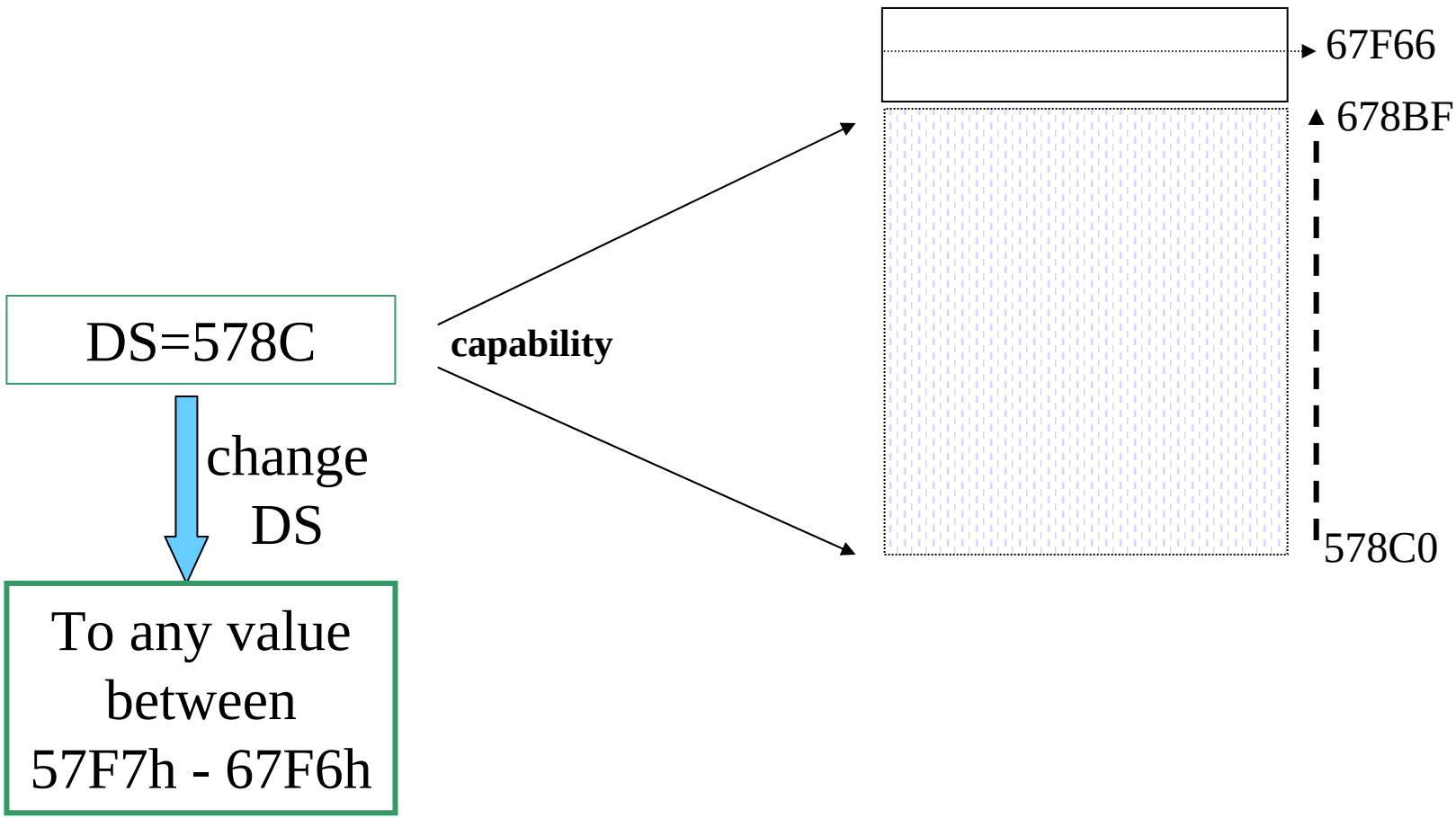
7FA2:438E



Example

Question:

Assume DS=578C. To access a Data in 67F66 what should we do?



Code Segment

- To execute a program, the 8086 fetches the instructions (opcodes and operands) from the code segment
- The logical address is in the form CS:IP

- Example: If CS = 24F6h and IP = 634Ah, show
  - The logical address
  - The offset address

  and calculate
  - The physical address
  - The lower range
  - The upper range

Logical Address vs Physical Address in the CS

CS:IP	Machine Language	Mnemonics
1132:0100	B057	MOV AL,57h
1132:0102	B686	MOV DH,86h
1132:0104	B272	MOV DL,72h
1132:0106	89D1	MOV CX,DX
1132:0108	88C7	MOV BH,AL
1132:010A	B39F	MOV BL,9F
1132:010C	B420	MOV AH,20h
1132:010E	01D0	ADD AX,DX
1132:0110	01D9	ADD CX,BX
1132:0112	05351F	ADD AX, 1F35h

- Show how the code resides physically in the memory

Data Segment

- 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 instructions will need to be modified

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

Little Endian Convention

"Little Endian" means that the low-order byte of the number is stored in memory at the lowest address, and the high-order byte at the highest address. (The little end comes first.)
Intel uses Little Endian Convention.
For example, a 4 byte LongInt

Byte3 | Byte2 | Byte1 |Byte0 will be arranged in memory as follows:

Base Address+0 Byte0
Base Address+1 Byte1
Base Address+2 Byte2
Base Address+3 Byte3

- **Adobe Photoshop** -- Big Endian
- **BMP (Windows and OS/2 Bitmaps)** – little Endian
- **GIF** -- Little Endian
- **IMG (GEM Raster)** -- Big Endian
- **JPEG** -- Big Endian

ENDIAN TYPE	$B_3B_2B_1B_0$ = 0XAABBCCDD	SAMPLE MICROPROCESSORS
Little-Endian	aa bb cc dd	Intel x86, Digital (VAX, Alpha)
Big-Endian	dd cc bb aa	Sun, HP, IBM RS6000, SGI, "Java"

Computer Operating Systems

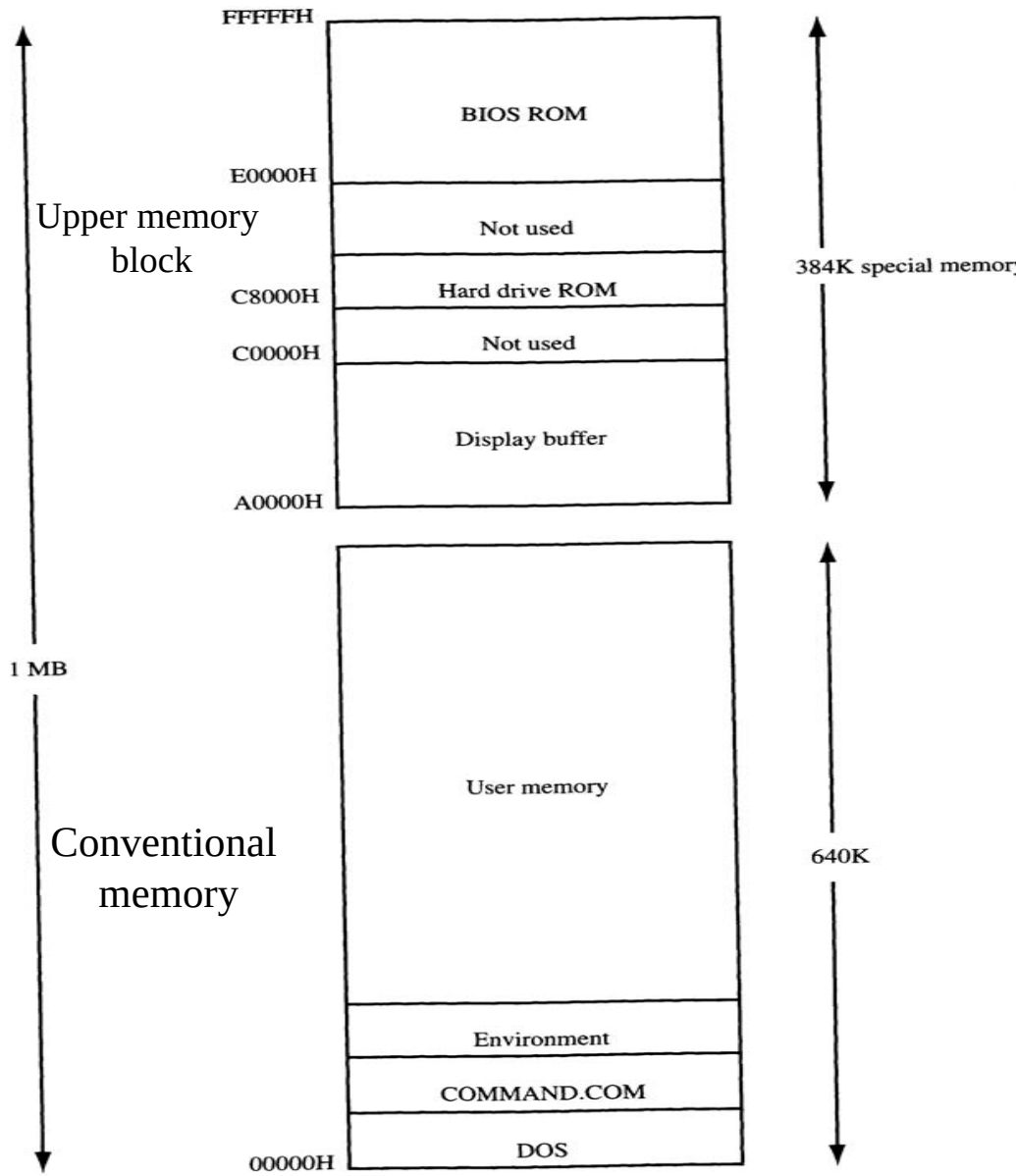
- What happens when the computer is first turned on?
- MS-DOS
  - A startup program in the **BIOS** (Basic Input Output System) is executed
  - This program in turn accesses the master boot record on the floppy or hard disk drive
  - A loader then transfers the system files **IO.SYS**
  - IO.SYS calls MSDOS.SYS. MS-DOS.SYS is basically the kernel of the operating system.
  - After initializing, MS-DOS.SYS then calls the command interpreter **COMMAND.COM** which is loaded into memory. This puts the DOS prompt on the screen that gives the user access to DOS's built-in commands like DIR, COPY, VER.

Memory Map of a PC

The 640 K Barrier

- DOS was designed to run on the original IBM PC 8088 microprocessor, 1Mbytes of main memory

- IBM divided this 1Mb address space into specific blocks
  - 640 K of RAM (user RAM)
  - 384 K reserved for ROM functions (control programs for the video system, hard drive



MS-DOS Functions and BIOS Services

BIOS: usually stored in ROM

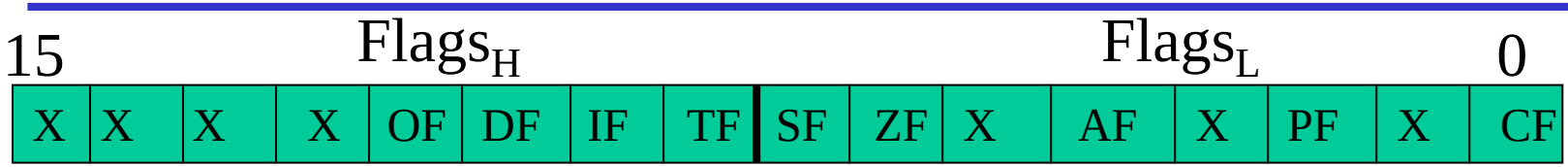
- tells the CPU what to do at startup
- these routines provide access to the peripheral devices of the PC, such as the keyboard, video, printer, and disk
- To test all the devices connected to the PC and alert if error

- Access to the BIOS is done through the software interrupt instruction Int *n*
- For example, the BIOS keyboard services are accessed using the instruction INT 16h
- In addition to BIOS services, DOS also provides higher level functions
  - INT 21h
  - More details later

More About RAM

- Memory management is one of the most important functions of the DOS operating systems and should be left to DOS
- Therefore, we do not assign any values for the DS,CS,SS registers; this is the job of DOS
- It is very important to remember that
  - The DS,CS, and DS values we will experiment will be different than those used by the textbook; do not worry

Flag (Status) Register



- 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

```
ADD AL,1
JNZ 0100h
```

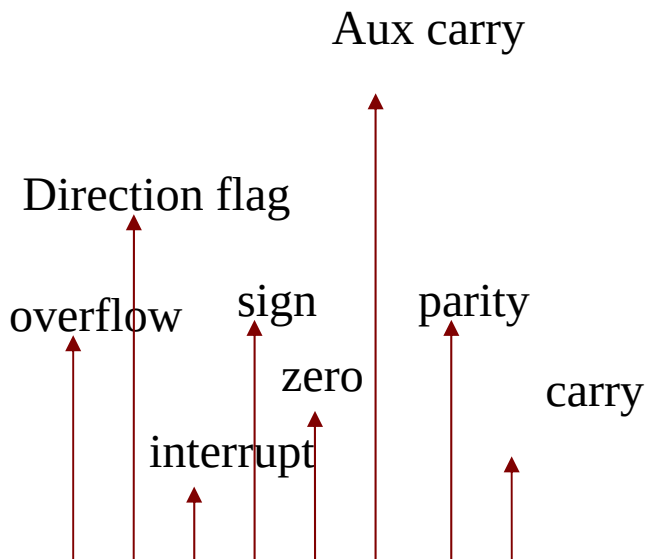
Example

- Show how the flag register is affected by
  - MOV AX, 34F5h
  - ADD AX,95EBh

0011 0100 1111 0101

1001 0101 1110 1011

1100 1010 1110 0000

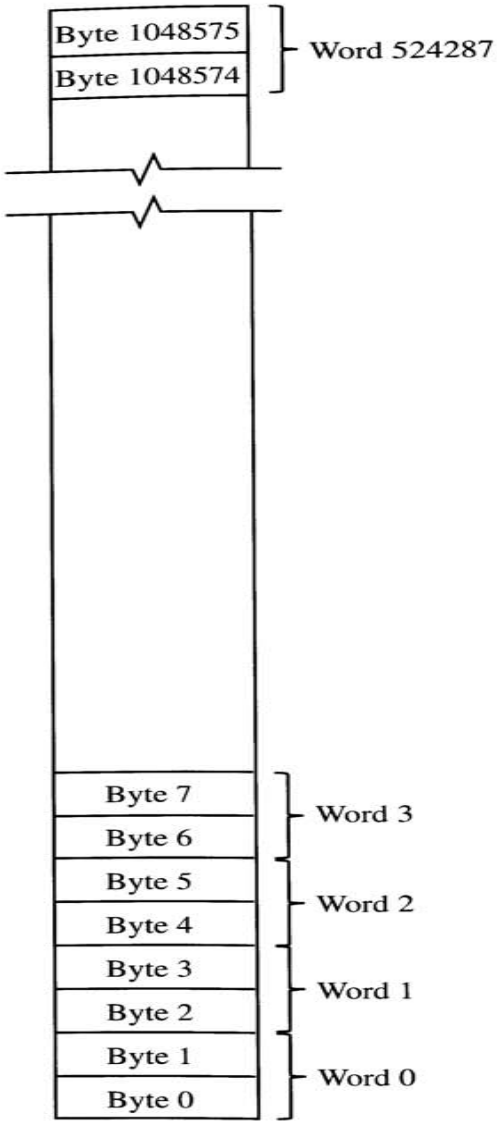



```
-t=1000

AX=34F5  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0B24  ES=0B24  SS=0B24  CS=0B24  IP=1003   NV UP EI PL NZ NA PO NC
0B24:1003 05EB95        ADD     AX,95EB
-t

AX=CAE0  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0B24  ES=0B24  SS=0B24  CS=0B24  IP=1006   NV UP EI NG NZ AC PO NC
0B24:1006 93            XCHG    BX,AX
```

TF, IF, and DF

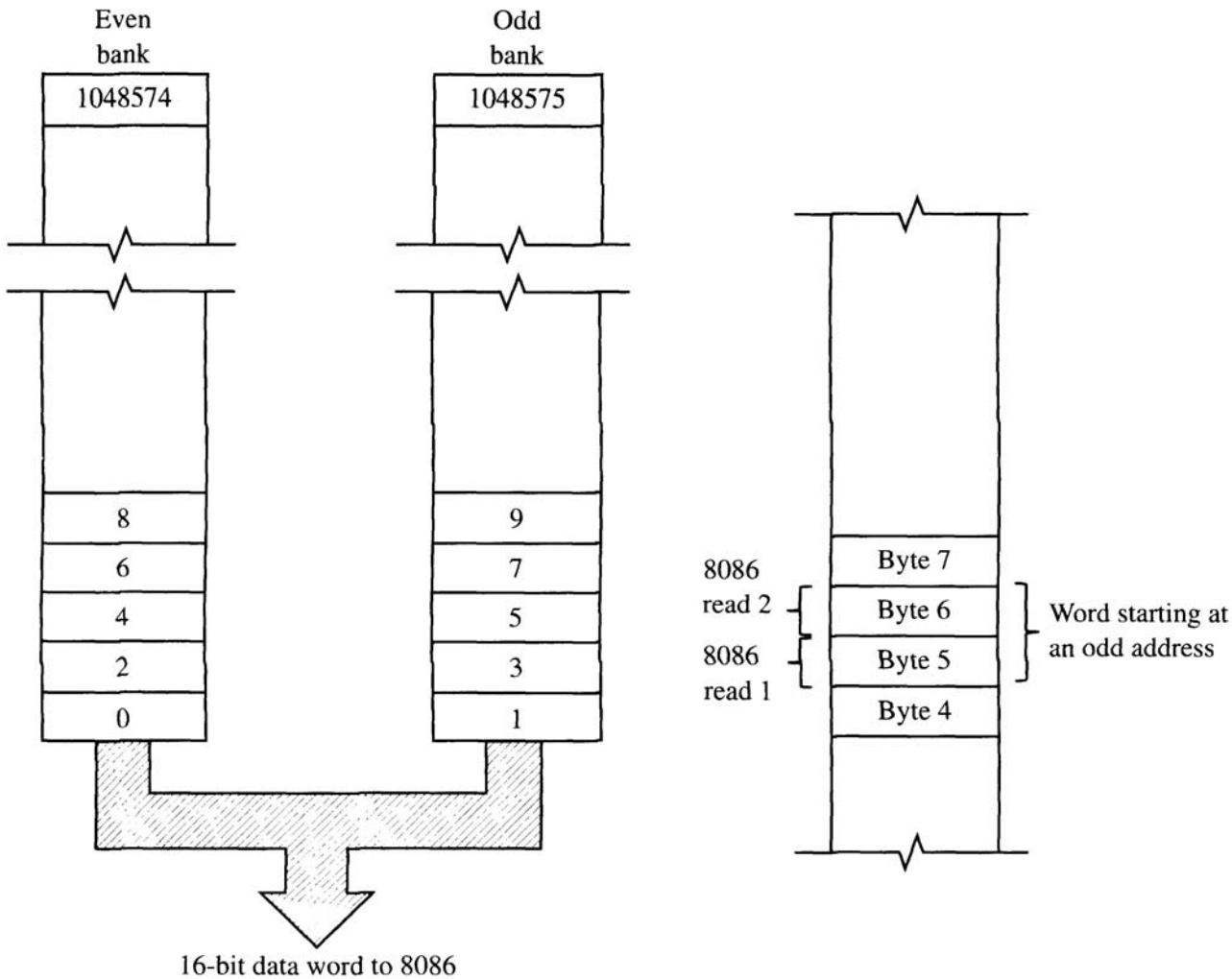
- Three of the flags can be set or reset directly by the programmer ands are used to control the operation of the microprocessor, these are TF, IF, and DF.
- When TF (Trap Flag) is set, control is passed to special address after each instruction is executed. Normally a program to display all the registers and flags is stored there. Single-stepping mode.
- When IF (Interrupt Flag) is set, external interrupt requests on the 8086's interrupt line INTR is 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
- DF (Direction Flag) is used with block move instructions (more later!!).
  - DF = 1 then the block memory pointer will automatically decrement
  - DF = 0, then the block memory pointer will automatically increment

Memory Address Space and Organization



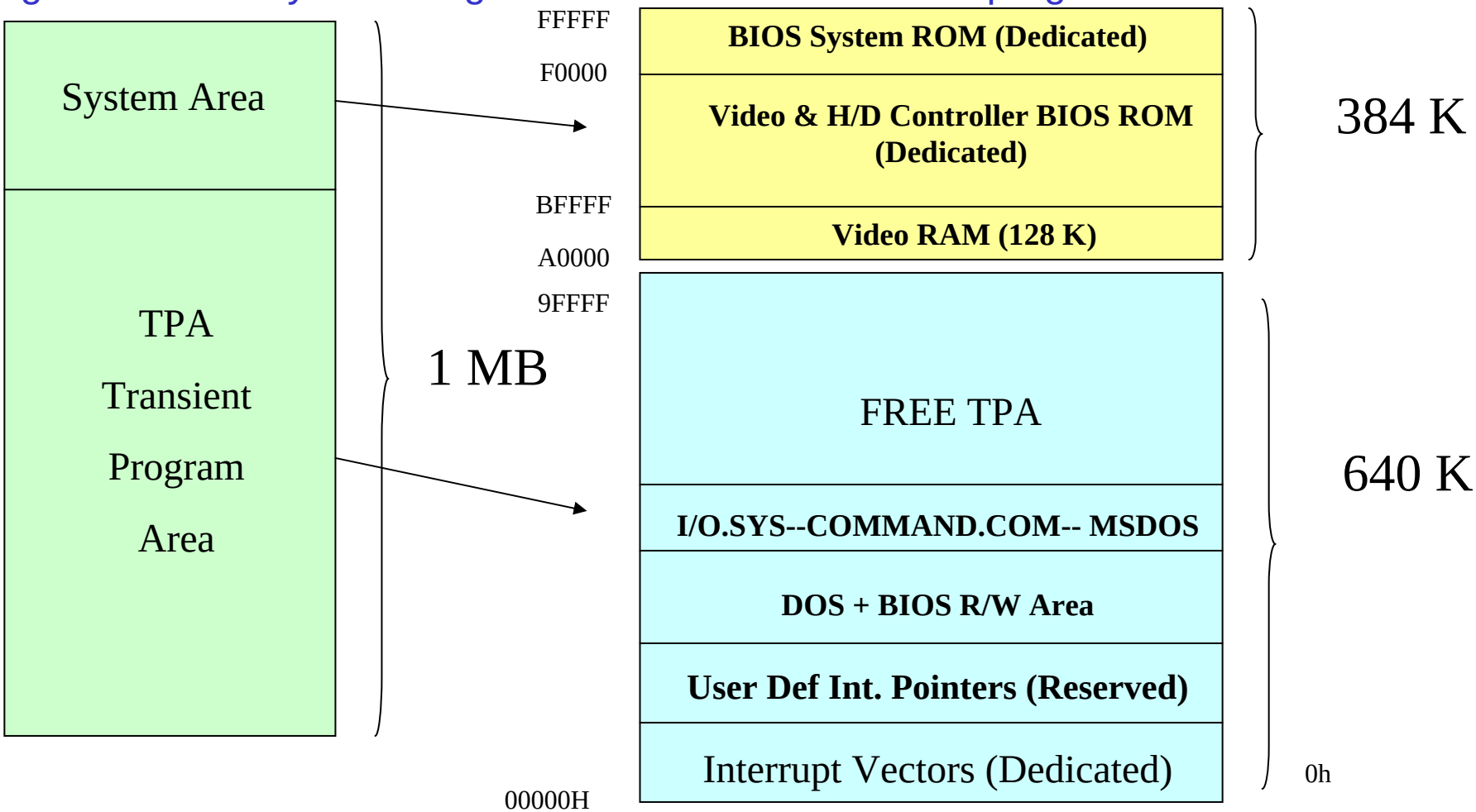
- Word
- Double Word
- Aligned Word
- Misaligned Word

Even addressed and odd-addressed banks



Dedicated, Reserved and General Purpose Memory

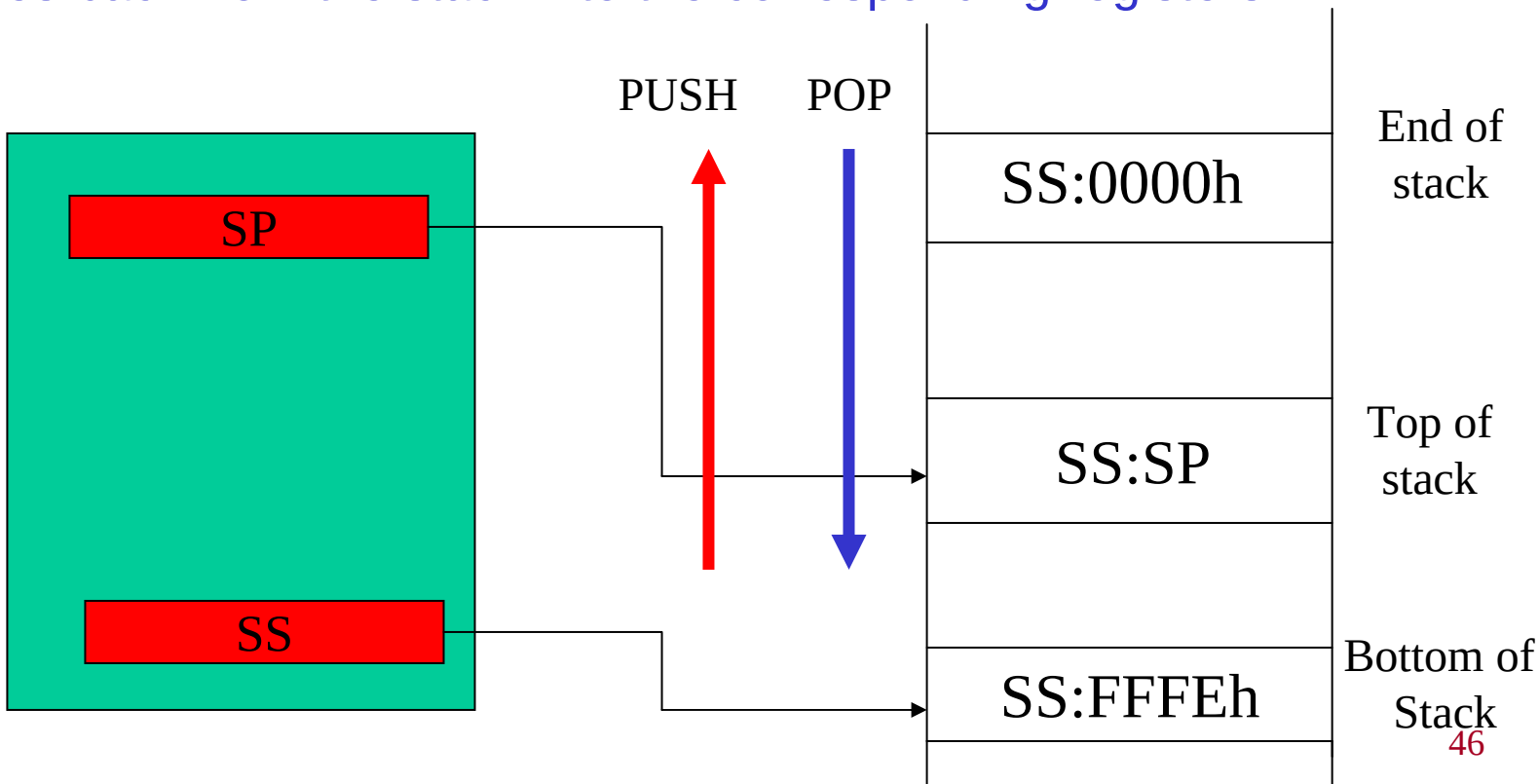
•Some address locations have dedicated functions and should not be used as general memory for storage of data or instructions of a program



Brey 19
Mazidi 32

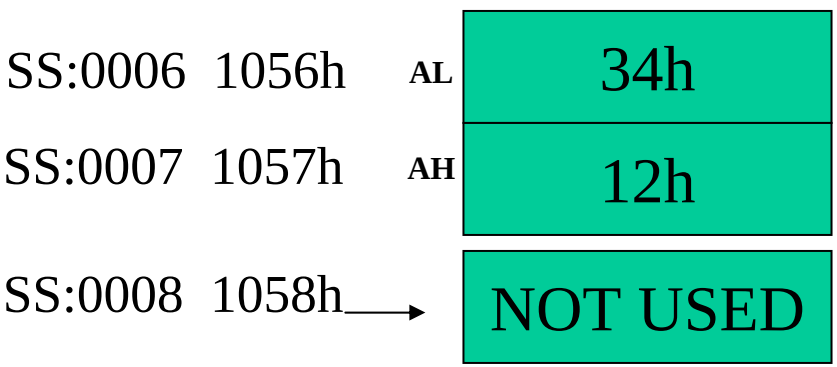
The Stack

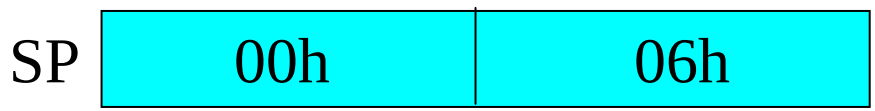
- The stack is used for temporary storage of information such as data or addresses; for instance when a call is executed the 8088 automatically pushes the current value of CS and IP onto the stack.
- Other registers can also be pushed
- Near the end of the subroutine, pop instructions can be used to pop values back from the stack into the corresponding registers



Example for PUSH

- Given
  - SS = 0105h
  - SP = 0008h
  - AX = 1234h
  - What is the outcome of the PUSH AX instruction?
- A_{BOS} = 01050 + FFFEh = 1104h
- A_{TOS} = 01050 + 0008h = 1058h
- Decrement the SP by 2 and write AX into the word location 1056h.





Example for POP

- What is the outcome of the following

  POP AX

  POP BX

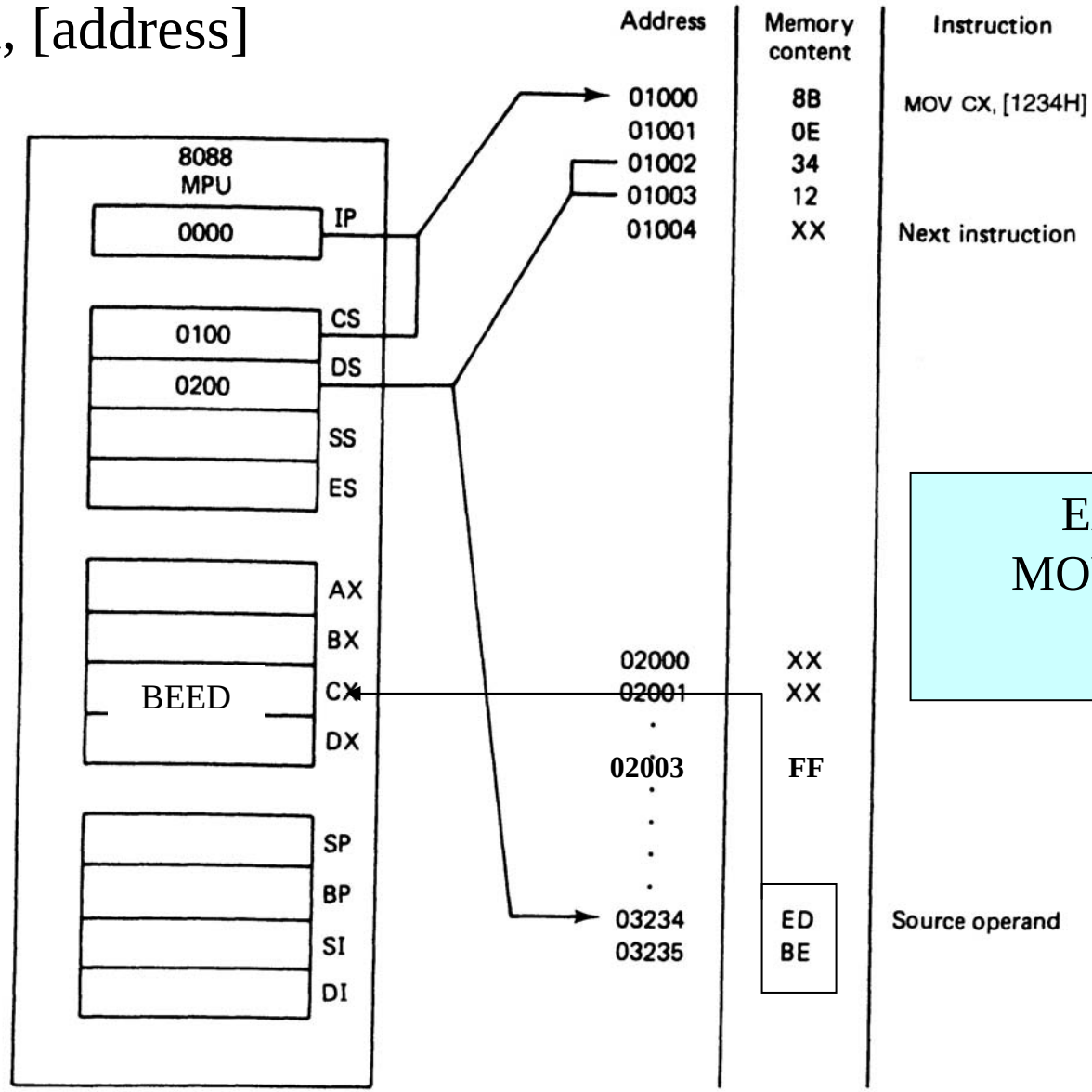
  – if originally 1058h contained AABBh?
- Read into the specified register from the stack and increment the stack pointer for each POP operation
- At the first POP
  – AX = 1234h SP = 0008h
- At the second POP
  – BX = AABBh SP = 000Ah

Addressing Modes

- When the 8088 executes an instruction, it performs the specified function on data
- These data, called operands,
  - May be a part of the instruction
  - May reside in one of the internal registers of the microprocessor
  - May be stored at an address in memory
- **Register Addressing Mode**
  - MOV AX, BX
  - MOV ES,AX
  - MOV AL,BH
- **Immediate Addressing Mode**
  - MOV AL,15h
  - MOV AX,2550h
  - MOV CX,625

Direct Addressing Mode

MOV CX, [address]

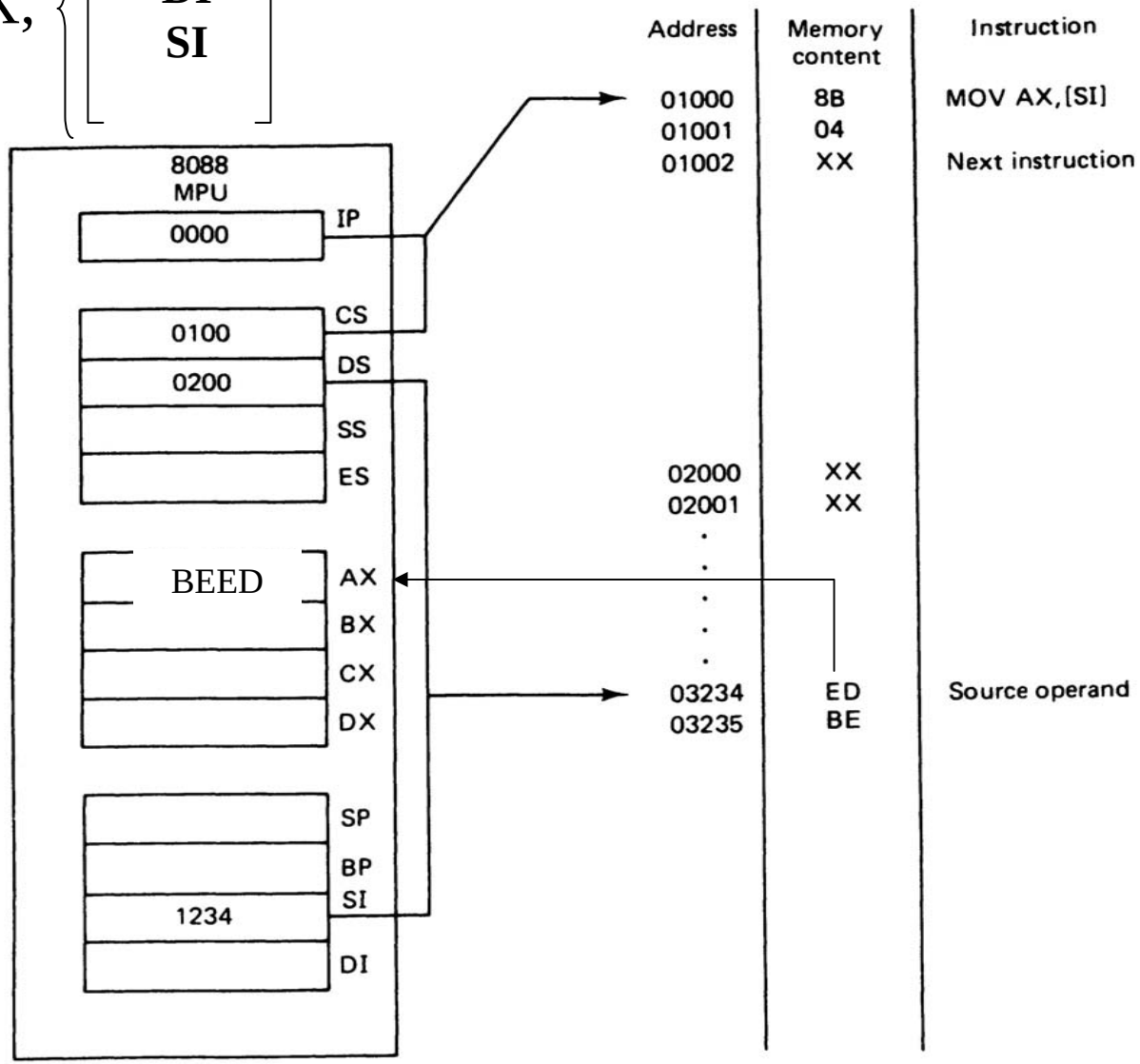


Example:
MOV AL,[03]

AL=?

Register Indirect Addressing Mode

MOV AX, { BX, DI, SI }



Example for Register Indirect Addressing

- Assume that DS=1120, SI=2498 and AX=17FE show the memory locations after the execution of:

MOV [SI],AX

DS (Shifted Left) + SI = 13698.

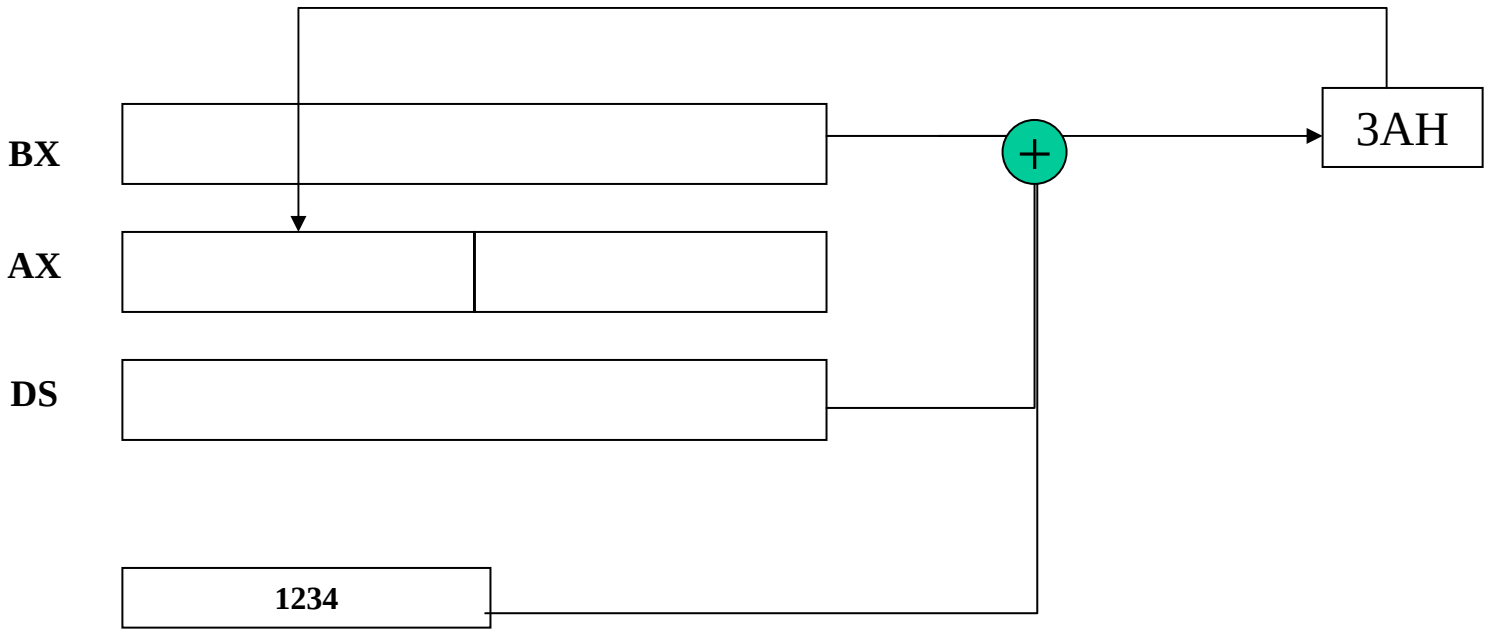
With little endian convention:

Low address 13698 → FE

High Address 13699 → 17

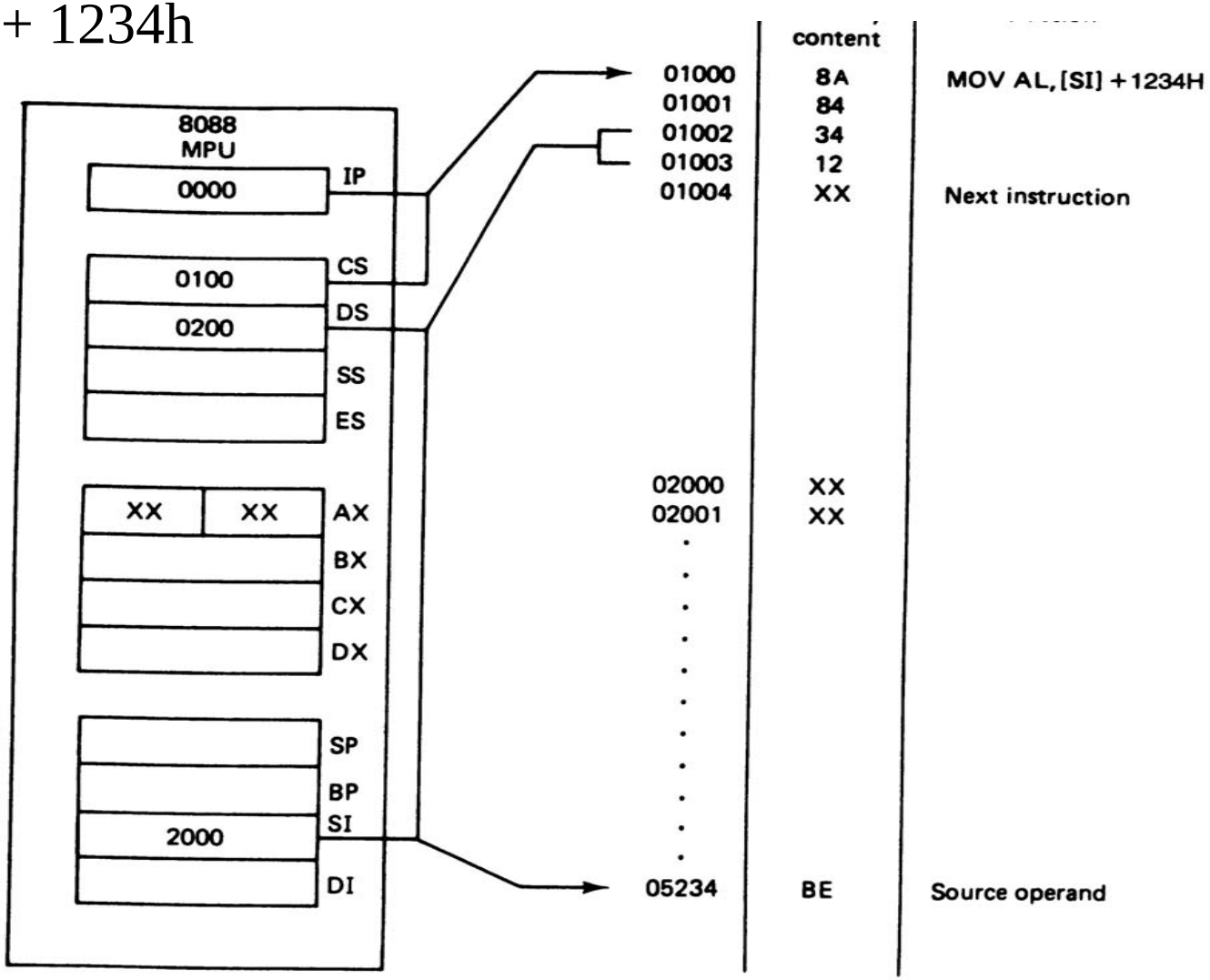
Based-Relative Addressing Mode

MOV AH, [**DS:BX** / **SS:BP**] + 1234h



Indexed Relative Addressing Mode

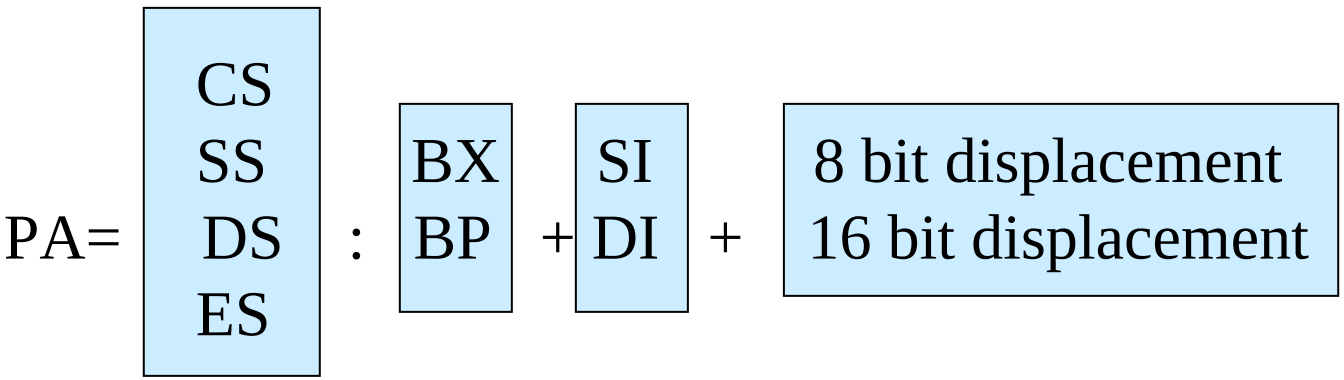
MOV AH, [$\frac{\text{SI}}{\text{DI}}$] + 1234h



Example: What is the physical address MOV [DI-8],BL if DS=200 & DI=30h ?
DS:200 shift left once 2000 + DI + -8 = 2028

Based-Indexed Relative Addressing Mode

- Based Relative + Indexed Relative
- We must calculate the PA (physical address)

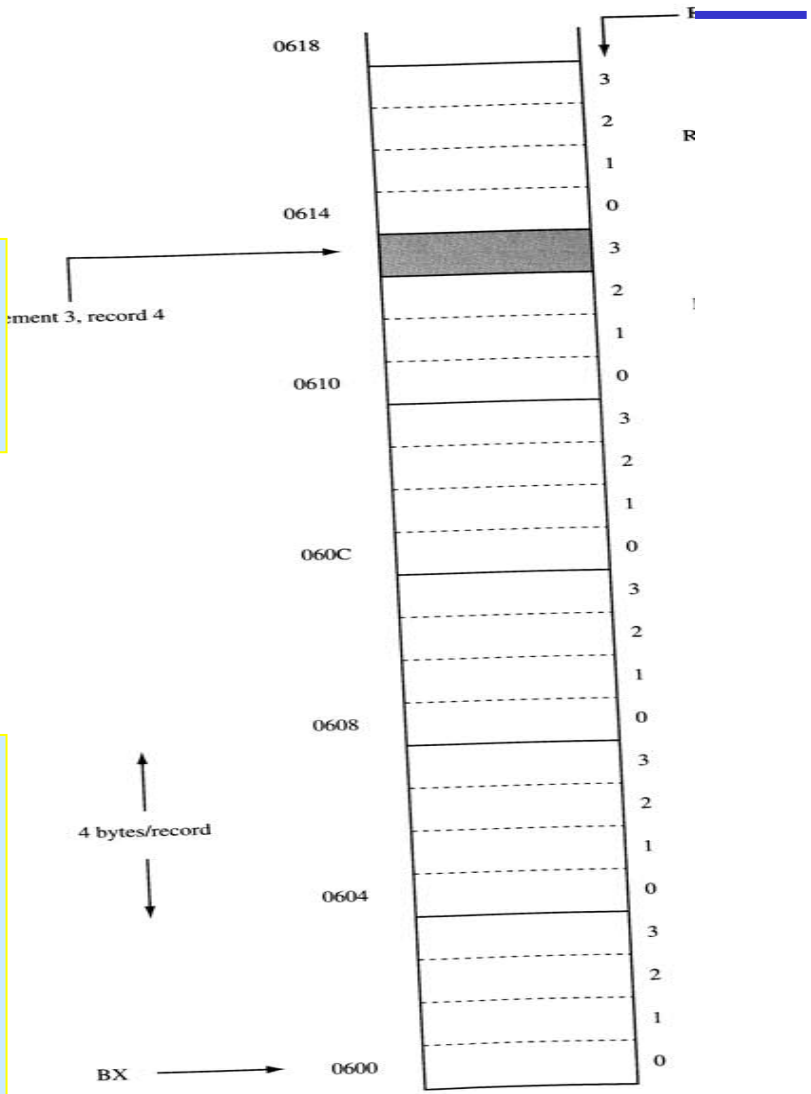


MOV AH,[BP+SI+29]
or
MOV AH,[SI+29+BP]
or
MOV AH,[SI][BP]+29

The register order does not matter

Based-Indexed Addressing Mode

ure 4–4.
e base + index +
placement
dressing mode can
used to access a
rticular element in a
rticular record of an
ray.




```
MOV BX, 0600h
MOV SI, 0010h ; 4 records, 4 elements each.
MOV AL, [BX + SI + 3]
```

OR

```
MOV BX, 0600h
MOV AX, 004h ;
MOV CX,04;
MUL CX
MOV SI, AX
MOV AL, [BX + SI + 3]
```

Summary of the addressing modes

Addressing Mode	Operand	Default Segment
Register	Reg	None
Immediate	Data	None
Direct	[offset]	DS
Register Indirect	[BX] [SI] [DI]	DS DS DS
Based Relative	[BX]+disp [BP]+disp	DS SS
Indexed Relative	[DI]+disp [SI]+disp	DS DS
Based Indexed Relative	[BX][SI or DI]+disp [BP][SI or DI]+disp	DS SS

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

Segment override

Segment Registers	CS	DS	ES	SS
Offset Register	IP	SI,DI,BX	SI,DI,BX	SP,BP

Instruction Examples	Override Segment Used	Default Segment
MOV AX,CS:[BP]	CS:BP	SS:BP
MOV DX,SS:[SI]	SS:SI	DS:SI
MOV AX,DS:[BP]	DS:BP	SS:BP
MOV CX,ES:[BX]+12	ES:BX+12	DS:BX+12
MOV SS:[BX][DI]+32,AX	SS:BX+DI+32	DS:BX+DI+32

Example for default segments

- The following registers are used as offsets. Assuming that the default segment used to get the logical address, give the segment register associated?

a) BP b)DI c)IP d)SI, e)SP, f) BX

- Show the contents of the related memory locations after the execution of this instruction

  MOV [BP][SI]+10,DX

  if DS=2000, SS=3000,CS=1000,SI=4000,BP=7000,DX=1299 (all hex)

SS(0)=30000

30000+4000+7000+10=3B010

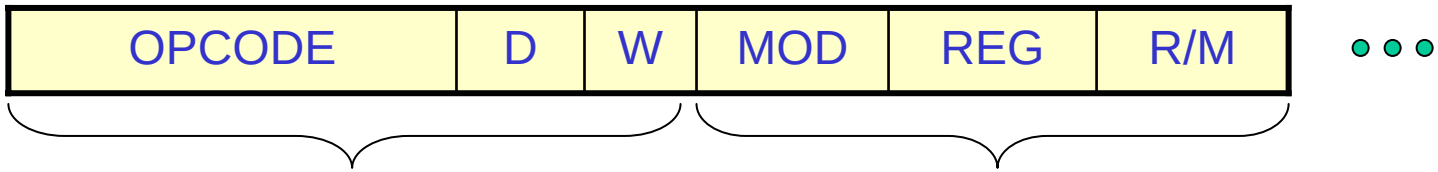
Assembly Language

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

Languages in terms of applications

- One of the most beneficial uses of assembly language programming is **real-time applications**.
- Real time means the task required by the application must be completed before any other input to the program that will alter its operation can occur
- For example the device service routine which controls the operation of the floppy disk drive is a good example that is usually written in assembly language
- Assembly language not only good for controlling hardware devices but also **performing pure software operations**
  - searching through a large table of data for a special string of characters
  - Code translation from ASCII to EBCDIC
  - Table sort routines
  - Mathematical routines
- Assembly language: perform real-time operations
- High-level languages: Those operations mostly not critical in time.

Converting Assembly Language Instructions to Machine Code



- An instruction can be coded with 1 to 6 bytes
- **Byte 1 contains three kinds of information:**
  - Opcode field (6 bits) specifies the operation such as add, subtract, or move
  - Register Direction Bit (D bit)
    - Tells the register operand in REG field in byte 2 is source or destination operand
      - 1:Data flow to the REG field from R/M
      - 0: Data flow from the REG field to the R/M
  - Data Size Bit (W bit)
    - Specifies whether the operation will be performed on 8-bit or 16-bit data
      - 0: 8 bits
      - 1: 16 bits
- **Byte 2 has two fields:**
  - Mode field (MOD) – 2 bits
  - Register field (REG) - 3 bits
  - Register/memory field (R/M field) – 2 bits

Continued

- **REG field is used to identify the register for the first operand**

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

Continued

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

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

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

(a)

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

(b)

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
100010	0	0	11	000	011

MOV BL,AL => 10001000 11000011 = 88 C3h

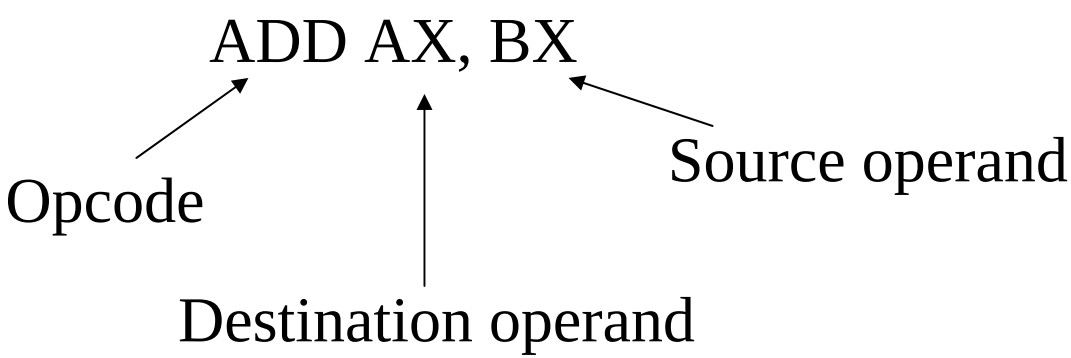
ADD AX,[SI] => 00000011 00000100 = 03 04 h

ADD [BX][DI] + 1234h, AX => 00000001 10000001 __ __ h

=> 01 81 34 12 h

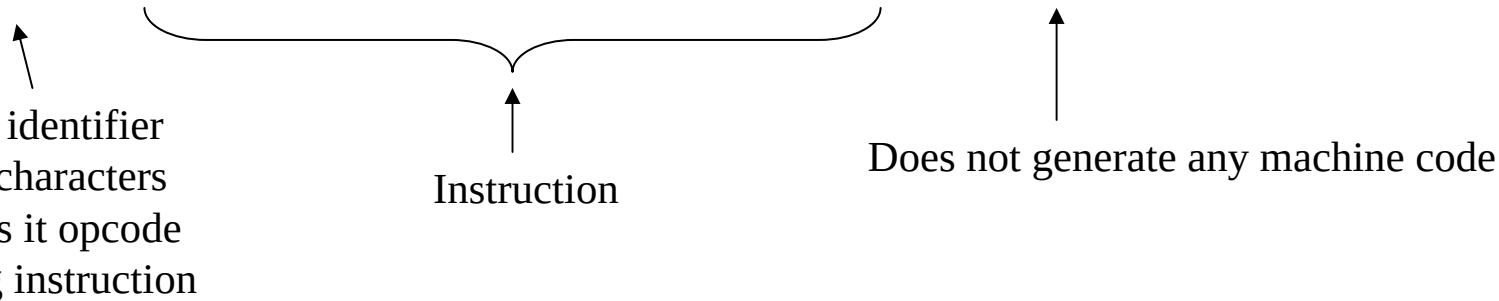
Software

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



Instructions

[LABEL:] MNEMONIC [OPERANDS] [; COMMENT]



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

DEBUG program instruction set (page 825 mzd)

- Debug instructions
- List of commands
  - a Assemble [address] you can type in code this way
  - c range address ; compare c 100 105 200
  - d [range] ; Dump d 150 15A
  - e address [list] ; Enter e 100
  - f Fill range list F 100 500 ‘ ‘
  - g Go [=address] addresses runs the program
  - h Value1 Value2 ; addition and subtraction H 1A 10
  - i Input port I 3F8
  - r Show & change registers Appears to show the same thing as t, but doesn't cause any code to be executed.
  - t Trace either from the starting address or current location.
  - u UnAssemble

Some examples with debug

0100 mov ax,24b6

0103 mov di, 85c2

0106 mov dx,5f93

0109 mov sp,1236

010c push ax

010d push di

010e int 3

Display the stack contents after execution.

-D 1230 123F

Some examples with DEBUG

- 0100 mov al,9c
- 0102 mov dh,64
- 0104 add al,dh
- 0109 int 3

trace these three commands and observe the flags

- After the code has been entered with the A command
- Use CX to store data indicating number of bytes to save. BX is the high word.
- Use N filename.com
- Then W command to write to file.
- L loads this file.

Example

Copy the contents of a block of memory (16 bytes) starting at location 20100h to another block of memory starting at 20120h

```
        MOV AX,2000
        MOV DS,AX
        MOV SI, 100
        MOV DI, 120
        MOV CX, 10
NXTPT:  MOV AH, [SI]
        MOV [DI], AH
        INC SI
        INC DI
        DEC CX
        JNZ  NXTPT
```