ECE 123h

Lecture 3

Basic Instructions

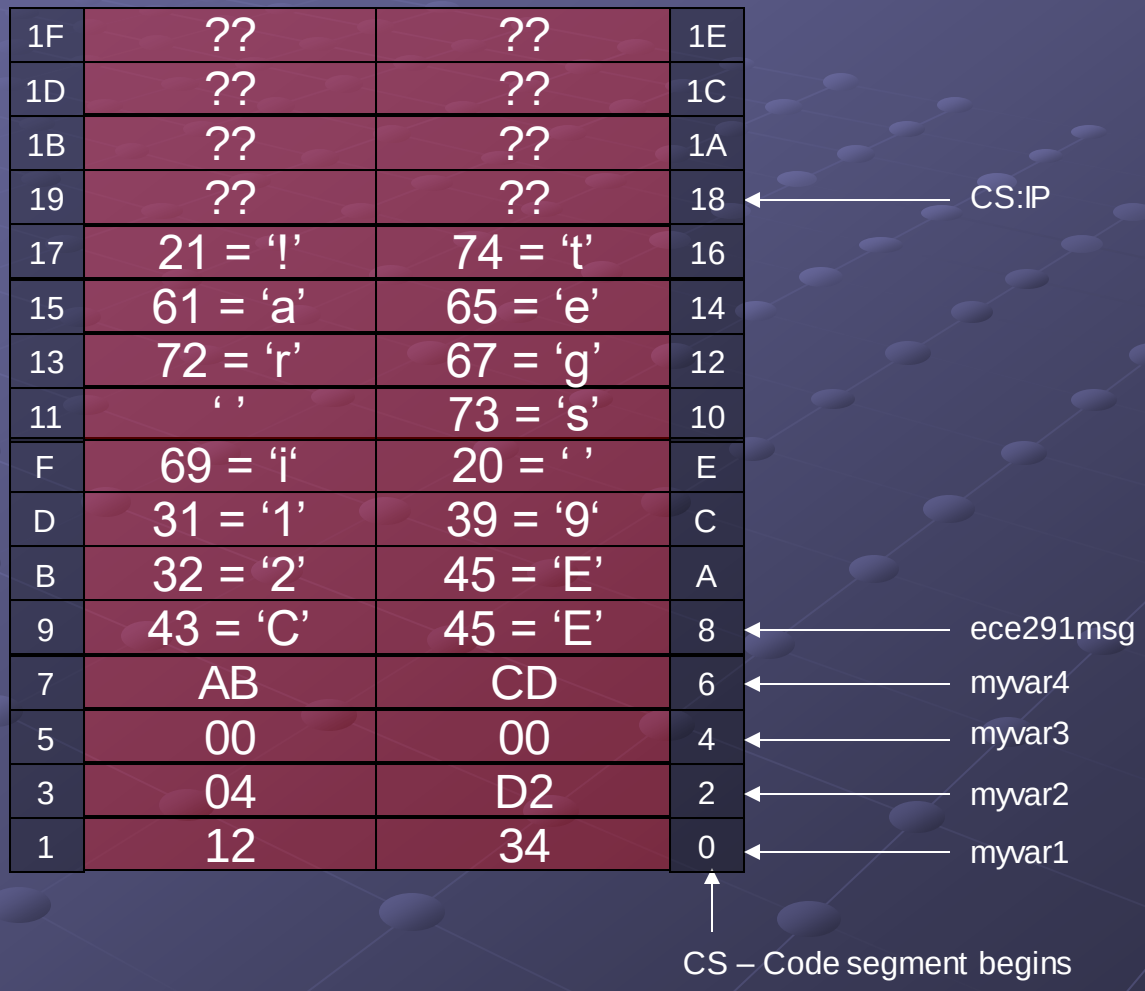
Lecture outline

- Memory access example
- Flags
- Logic instructions
- Shifting instructions
- Addition and subtraction
- Overflows and carries
- MP3 introduction

Memory example - declarations

```
SEGMENT code

myvar1        DW    1234h       ; define word variable
                                  ; value=1234h
myvar2        DW    1234        ; define word variable
                                  ; value=1234d=4D2h
myvar3        RESW              ; define word variable
                                  ; value not specified
myvar4        DW    0ABCDh      ; define word variable
                                  ; value = ABCDh
                                  ; note the leading zero
ece291msg     DB    "ECE291 is great!"
                                  ; define strings as series of bytes
                                  ; this is not one byte but 16!
..start                           ; denotes execution starting point
```

What it looks like to the computer



Memory example - MOVing

```
mov ax, cs                  ; make data seg same as code seg
mov ds, ax

mov ax, [myvar2]            ; move value of myvar2 to ax
                            ; same as mov ax, [2]
                            ; ax now contains 04D2h
mov si, myvar2              ; si now points to myvar2
                            ; it contains 0002h

mov ax, [si]                ; move value pointed to by si to ax
                            ; ax again contains 04D2h
                            ; this was an indirect reference
```

Memory example - MOVing

```
mov bx, ece291msg           ; bx now points to the beginning of
                            ; out "ECE291 is great" string.
                            ; bx points to the first 'E'
                            ; bx contains 0008h

dec byte [bx + 1]           ; new instruction!  arg = arg - 1
                            ; Just changed the C in "ECE" to a B

mov si, 1                   ; now lets use SI as an index

inc byte [ece291msg + si]   ; new instruction! arg = arg + 1
                            ; evaluates to inc [8 + 1] = inc [9]
                            ; B becomes a C again!
```

Memory example - MOVing

```
; Memory can be addressed using four registers only!!!
;  SI -> offsets from DS
;  DI -> offsets from DS
;  BX -> offsets from DS
;  BP -> offsets from SS

mov ax, [bx]                ; ax = word in memory pointed to by bx
mov al, [bx]                ; al = byte in memory pointed to by bx
mov ah, [si]                ; ah = byte in memory pointed to by si
mov cx, [di]                ; cx = word in memory pointed to by di
mov ax, [bp]                ; ax = [SS:BP] Stack Operation!

; In addition, BX + SI, BX + DI, BP + SI, and BP + DI are allowed
mov ax, [bx + si]           ; ax = [ds:bx+si]
mov ch, [bp + di]           ; ch = [ss:bp+di]
```

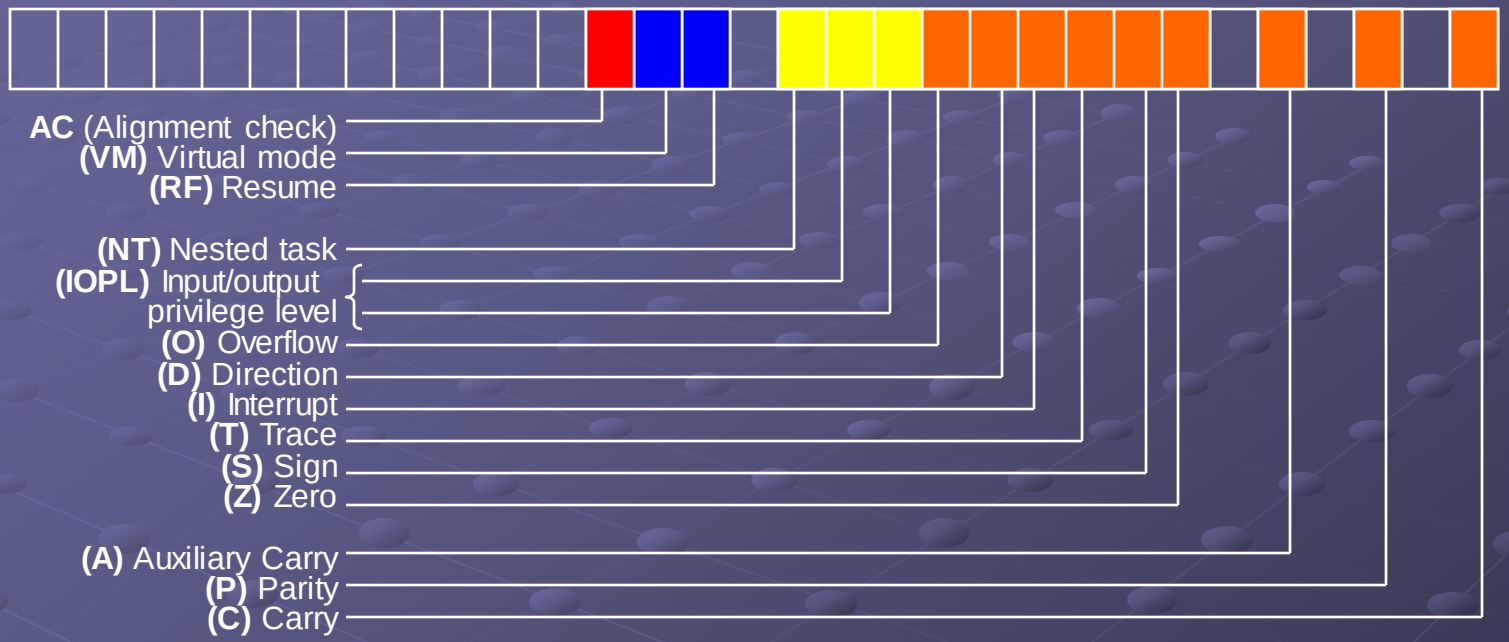
Memory example - MOVing

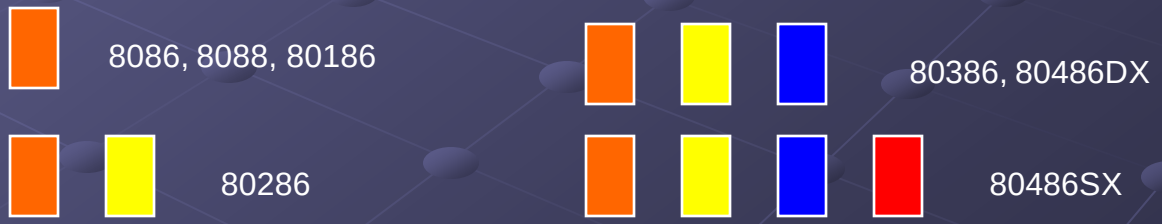
```
; Furthermore, a fixed 8- or 16-bit displacement can be added

mov ax, [23h]              ; ax = word at [DS:23h]
mov ah, [bx + 5]           ; ah = byte at [DS:bx+5]
mov ax, [bx + si + 107]    ; ax = word at [DS:bx+si+107]
mov ax, [bp + di + 42]     ; ax = word at [SS:bp+di+42]

; Remember that memory to memory moves are illegal!
mov [bx], [si]             ; BAD BAD BAD
mov [di], [si]             ; BAD BAD BAD

; Special case with stack operations
pop word [myvar]           ; moves data from top of stack to
                           ; memory variable myvar
```

The flag register

- Flags indicate the condition of the CPU as well as its operation
- Flag bits change after many arithmetic and logic instructions complete execution
- Example flags:
  - C(carry) indicates carry after addition or borrow after subtraction in most significant bit
  - O(overflow) is a condition that can occur when signed numbers are added or subtracted
  - Z(zero) indicates that the result of an operation is zero

The flag register





Logic instructions

- Logic instructions operate on a bit-by-bit basis:

Assembly	**C**
NOT A	A = ~A
AND A, B	A &= B
OR A, B	A |= B
XOR A, B	A ^= B

- Except for NOT these instructions affect the flags as follows:
  - clear the carry (C)
  - clear the overflow (0)
  - set the zero flag (Z) if the result is zero, or clear it otherwise
  - copy the high order bit of the result into the sign flag (S)
  - set the parity bit (P) according to the parity (number of 1's) in the result
  - scrambles the auxiliary carry flag (A)
- The NOT instruction does not affect any flags

Logic examples

AL	1100 1010
NOT AL	
AL	0011 0101

AL	0011 0101
BL	0110 1101
AND AL, BL	
AL	0010 0101

AL	0011 0101
BL	0000 1111
AND AL, BL	
AL	0000 0101

AL	0011 0101
BL	0110 1101
OR AL, BL	
AL	0111 1101

AL	0011 0101
BL	0000 1111
OR AL, BL	
AL	0011 1111

AL	0011 0101
BL	0110 1101
XOR AL, BL	
AL	0101 1000

AND/OR applications

- AND and OR instructions are often used to mask out data
  - a mask value is used to force certain bits to zero or one within some other value
  - a mask typically affects certain bits and leaves other bits unaffected
    - AND forces selected bits to zero — AND CL, 0Fh
    - OR forces selected bits to one — OR CL, 0Fh

Logic instructions

- TEST is a non-destructive AND
  - logically ands two operands just as AND would
  - sets flags the same as AND
  - doesn't modify contents of destination
  - TEST AL, 1 sets flags same as AND AL, 1 but doesn't change AL
  - useful when setting up conditional jumps

Shift left

- Syntax: SHL reg, count (immediate count)
  SHL reg (count stored in CL)
  - moves each bit of the operand one bit position to the left the number of times specified by the count operand
  - zeros fill vacated positions at the L.O. bit
  - H.O. bit shifts into the carry flag
- a quick way to multiply by two
- useful in *packing data,* e.g., consider two nibbles in AL and AH that we want to combine

```
shl ah, 4
or al,ah
```

- the 8086 and 8088 allow an immediate shift of 1 only

Example

```
; Pack the lower nibbles from AH and AL into a single byte in AL
; for example AH = 0110 1101 and AL = 1010 0111

and al, 0Fh                 ; clears upper nibble in AL
                            ; AL = 0000 0111
shl ah, 4                   ; Moves lower nibble in AH up
                            ; AH = 1101 0000
or  al, ah                  ; combines nibbles
                            ; AL = 1101 0111

; Mission accomplished
```

Shift right

- Syntax: SHR reg, count (immediate count)
  SHR reg (count stored in CL)
  - moves each bit of the operand one bit position to the right the number of times specified by the count operand
  - zeros fill vacated positions at the H.O. bit
  - L.O. bit shifts into the carry flag
- a quick way to divide by two
- useful in *unpacking data,* e.g., suppose you want to extract the two nibbles in the AL register, leaving the H.O. nibble in AH and the L.O. nibble in AL

```
mov ah, al
shr ah, 4
and al, 0Fh
```

- the 8086 and 8088 allow an immediate shift of 1 only

Shift arithmetic right

- Syntax: SAR reg, count (immediate count)
  SAR reg (count stored in CL)
  - moves each bit of the operand one bit position to the left the number of times specified by the count operand
  - H.O. bit replicates
  - L.O. bit shifts into the carry flag
- main purpose is to perform a *signed division* by some power of two

```
mov ax, -15
sar ax, 1          ; result is -8
```

- In 80286 and later you can use SAR to sign extend one register into another

```
mov ah, al
sar ah, 7
```

if AL contains 1111 0001 then AX will be 1111 1111 1111 0001 after execution

Rotate through carry L/R

- Syntax: RCL reg, count (immediate count)
  RCL reg (count stored in CL)
  - rotates bits to the left, through the carry flag
  - bit in the carry flag is written back into bit zero on the right
- Syntax: RCR reg, count (immediate count)
  RCR reg (count stored in CL)
  - rotates bits to the right, through the carry flag
  - bit in the carry flag is written back into the H.O. bit on the left

Rotate left/right

- Syntax: ROL reg, count (immediate count)
  ROL reg (count stored in CL)
  - rotates bits to the left
  - H.O. bit goes to carry flag and L.O. bit
- Syntax: ROR reg, count (immediate count)
  ROR reg (count stored in CL)
  - rotates bits to the right
  - L.O. bit goes to carry flag and H.O. bit

Shifting example

```
mov ax,3           ; Initial register values        AX = 0000 0000 0000 0011
mov bx,5           ;                                BX = 0000 0000 0000 0101
or ax,9            ; ax <- ax | 0000 1001           AX = 0000 0000 0000 1011
and ax,10101010b   ; ax <- ax & 1010 1010           AX = 0000 0000 0000 1010
xor ax,0FFh        ; ax <- ax ^ 1111 1111           AX = 0000 0000 1111 0101
neg ax             ; ax <- (-ax)                    AX = 1111 1111 0000 1011
not ax             ; ax <- (~ax)                    AX = 0000 0000 1111 0100
Or ax,1            ; ax <- ax | 0000 0001           AX = 0000 0000 1111 0101
shl ax,1           ; logical shift left by 1 bit    AX = 0000 0001 1110 1010
shr ax,1           ; logical shift right  by 1 bit  AX = 0000 0000 1111 0101
ror ax,1           ; rotate right (LSB=MSB)         AX = 1000 0000 0111 1010
rol ax,1           ; rotate left (MSB=LSB)          AX = 0000 0000 1111 0101
mov cl,3           ; Use CL to shift 3 bits         CL = 0000 0011
shr ax,cl          ; Divide AX by 8                 AX = 0000 0000 0001 1110
mov cl,3           ; Use CL to shift 3 bits         CL = 0000 0011
shl bx,cl          ; Multiply BX by 8               BX = 0000 0000 0010 1000
```

Addition

- Syntax: ADD A, B
  - A and B can be register or memory (not both memory)
  - B can also be immediate data
- Function: A = A + B
- Examples:
  - ADD AX, BX ; register addition
  - ADD AX, 5h ; immediate addition
  - ADD [BX], AX ; addition to memory location
  - ADD AX, [BX] ; memory location added to register
  - ADD DI, MYVAR ; memory offset added to register
- Flags modified: S, C, A, Z, P, O

Example

```
; multiply AX by the decimal 10 (1010b)
; (multiply by 2 and 8, then add results)

shl ax, 1                    ; AX x 2
mov bx, ax                   ; save 2AX
shl ax, 2                    ; 2 x (original AX) x 4 = 8 x (orig AX)
add ax, bx                   ; 2AX + 8AX = 10AX

; multiply AX by decimal 18 (10010b)
; (multiply by 2 and 16, then add results)

shl ax, 1                    ; AX x 2
mov bx, ax                   ; save 2AX
shl ax, 3                    ; 2AX x 2 x 2 x 2 = 16AX
add ax, bx                   ; 2AX + 16AX = 18AX
```

Increment

- Syntax: INC A
  - A can be a register or memory location
- Function: A = A + 1
- Examples
  - INC AX ; AX = AX + 1
  - INC byte [BX] ; memory location increased by 1
  - INC word [MYVAR] ; memory location increased by 1
- Flags modified: S, A, Z, P, O
  - notice INC does **not** affect the carry flag

Add with carry

- Syntax: ADC A, B
  - A and B can be register or memory (not both memory)
  - B can also be immediate data
- Function: A = A + B + carry flag
  - used mainly to add numbers that are wider than 16 bits (when in 16-bit mode) or wider than 32 bits (when in 32-bit instruction mode)
- Examples:
  - add ax, cx ; this produces the 32 bit sum of
    adc bx, dx ; bx:ax + dx:cx
- Flags modified: S, C, A, Z, P, O

Subtraction

- Syntax: SUB A, B
  - A and B can be register or memory (not both memory)
  - B can also be immediate data
- Function: A = A – B
  - carry flag interpreted as borrow request in H.O. bit
- Examples:
  - mov ch, 22h
    sub ch, 34h
- Flags modified: S, C, A, Z, P, O

Result is -12 (1110 1110)
Flags change:
Z = 0 (result not zero)
C = 1 (borrow in H.O. bit)
S = 1 (result negative)
P = 0 (even parity)
O = 0 (no overflow)

Decrement

- Syntax: DEC A
  - A can be a register or memory location
- Function: A = A - 1
- Examples
  - DEC AX ; AX = AX - 1
  - DEC byte [BX] ; memory location decreased by 1
  - DEC word [MYVAR] ; memory location decreased by 1
- Flags modified: S, A, Z, P, O
  - notice DEC does **not** affect the carry flag

Subtract with borrow

- Syntax: SBB A, B
  - A and B can be register or memory (not both memory)
  - B can also be immediate data
- Function: A = A - B - carry flag
  - used mainly to subtract numbers that are wider than 16 bits (when in 16-bit mode) or wider than 32 bits (when in 32-bit instruction mode)
- Examples:
  - sub ax, di ; this produces the 32 bit difference of
    sbb bx, si ; bx:ax - si:di → bx:ax
- Flags modified: S, C, A, Z, P, O

The carry flag

- Carry
  - indicates a carry in the H.O. bit after addition or a borrow in the H.O. bit after subtraction
- 8-bit example
  - FFh + 1h = 00h, with carry
  - 02h – 03h = FFh, with carry
- Carry is set when an unsigned number goes out of range (this is an unsigned arithmetic overflow)

The overflow flag

- Overflow
  - condition that occurs when signed numbers go out of range
  - For 8-bit, that range is from -128 to + 127 decimal
- 8-bit overflow example
  - 7Fh + 1h = 80h, wanted 127 + 1 = 128 but got -128
  - 80h – 1h = 7F, we wanted -1 – 1 = -2 but got +127

Overflows and carries examples

```
mov ax,0fffeh          ; 65534 interpreted as unsigned
add ax,3               ; C = 1, O = 0   ---   Unsigned Overflow Condition

mov ax,0FFFEh          ; -2 interpreted as signed
add ax,3               ; C = 1, O = 0   ---   Okay as signed

mov bx,07FFFh          ; 32767 interpreted as signed
add bx,1               ; C = 0, O = 1   ---   Signed Overflow Condition

mov bx,07FFFh          ; 32767 interpreted as unsigned
add bx,1               ; C = 0, O = 1   ---   Okay as unsigned

mov ax,07h             ; 7 interpreted as either signed or unsigned
add ax,03h             ; C = 0, O = 0   ---   Okay as either
```