EC-310 Microprocessor & Microcontroller based Design

Dr. Ahsan Shahzad

Department of Computer and Software Engineering (DC&SE)

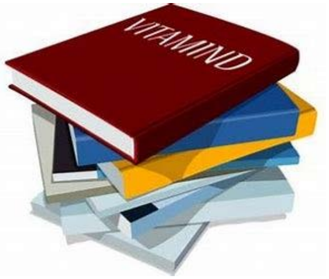
College of Electrical and Mechanical Engineering (CEME)

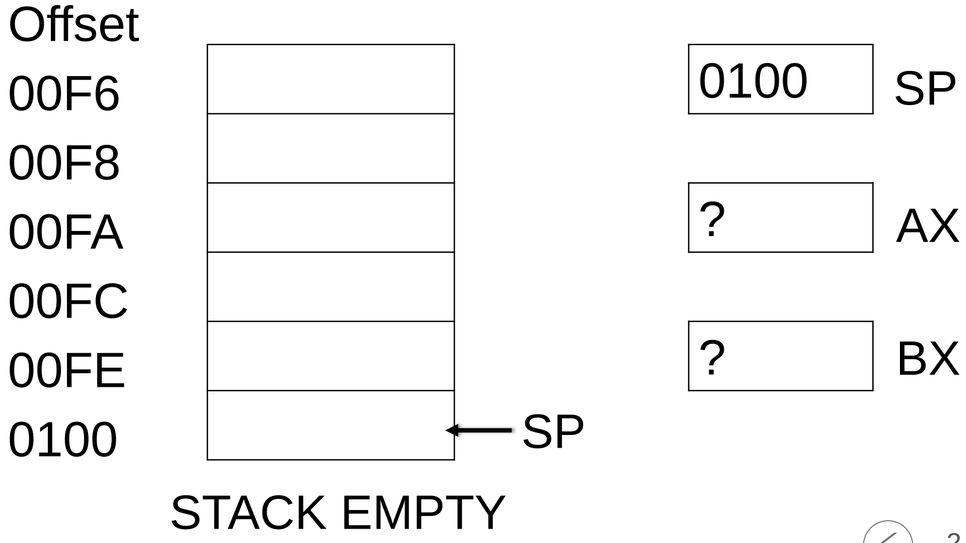
National University of Science and Technology (NUST)

Week # 4

The STACK

- Stack Segment of a Program is used for temporary storage of data and addresses.
- Stack works in LIFO (last in first out) manner
- To set aside a block of memory for stack
  - .STACK 100h
- SS will contain the segment number of the stack
- SP contains the offset address of the top of the stack

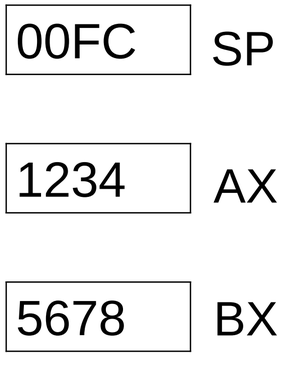




STACK (PUSH, PUSHF, POP, POPF)

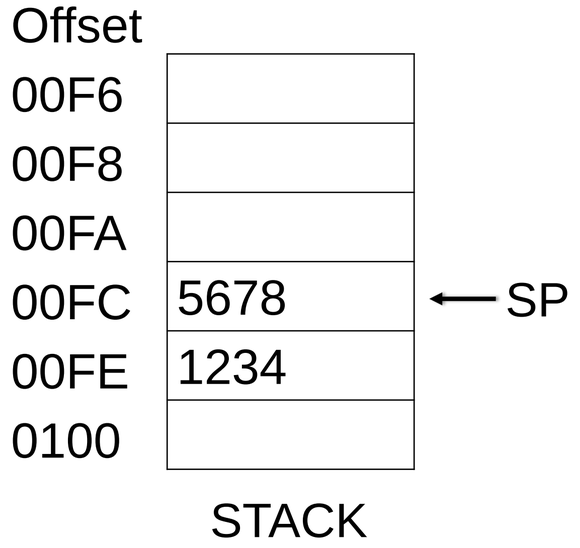
Example

```
MOV AX, 1234h
MOV BX, 5678h
PUSH AX
PUSH BX
```



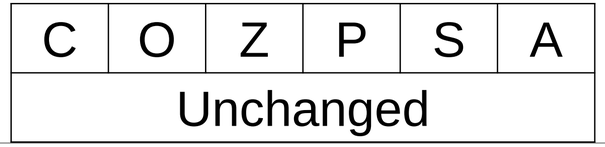
PUSH Instruction

- Store 16 bit value in the stack.
- Format: **PUSH Operand1** (Operand1 could be: 16bit REG, SREG, memory)
- Algorithm:
  - SP = SP -2
  - SS: [SP] (top of the stack) = Operand
- **PUSHF Instr.** (No Operands) Store 16 bit Flag Register in to the Stack



Operands Type

- **REG**: AX, BX, CX, DX, AH, AL, BL, BH, CH, CL, DH, DL, DI, SI, BP, SP.
- **SREG**: DS, ES, SS, and only as second operand: CS.
- **memory**: [BX], [BX+SI+7], variable, etc.
- **Immediate**: 5, -24, 3Fh, 10001101b, etc.



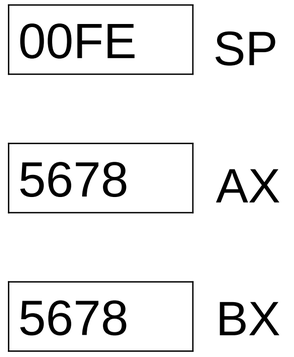
STACK (PUSH, PUSHF, POP, POPF)

Example
POP AX ; AX = 5678h

POP Instruction

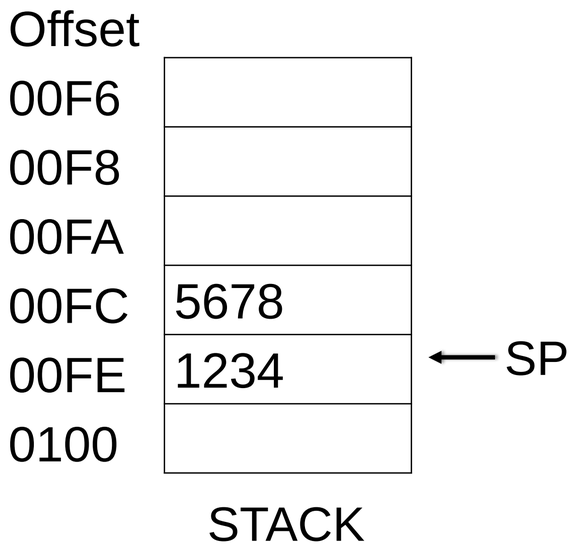
- Get 16 bit value from the top of the stack.
- Format: **POP Operand1** (Operand1 could be: 16bit REG, SREG, memory)
- Algorithm:
  - Operand = SS: [SP] (top of the stack)
  - SP = SP + 2

C	O	Z	P	S	A
Unchanged					



POPF Instruction

- Get 16 bit value from the top of the stack into the Flags register
- Format: **POPF**
- Algorithm:
  - Flag Reg. = SS: [SP] (top of the stack)
  - SP = SP + 2



Procedures (CALL, RET)

- Top-down program design
- One **MAIN** Procedure, which contains the entry point of the program.

Procedure Declaration

name PROC type

; body of the procedure

RET

name ENDP

- Optional operand
- **NEAR (default)**: calling statement and procedure in same segment
- **FAR:** Calling statement is in different segment

Return Instr. Transfers Control back to calling function

User-defined

Procedures (CALL, RET)

Procedure Call and Return

- **CALL instr.** Is used to invoke a procedure
- Formats:
  - **CALL name** ; **direct** method
  - **CALL address_expression** ; **indirect** method, address_expression specifies a REG or memory location containing the address of a procedure.
- CALL Instr. Execution:
  - Return address (offset of next instr. After CALL instr.) is saved on the stack (CS:IP)
  - IP gets the offset address of the first instr. of the procedure (control transfer)

Procedures (CALL, RET)

Procedure Call and Return

- **RET instr.** is used to return from a procedure
- Formats:
  - **RET** ; mostly used format
  - **RET pop_value** **;** The integer pop_value is optional argument
- RET Instr. Execution:
  - For a NEAR procedure, execution of RET instr. Causes the stack to be popped into IP
  - If a pop_value N is specified, then it is added to SP, and has the effect of removing N additional bytes from the stack.

Procedures (CALL, RET)

Communication between Procedures

- Method to receive inputs from the calling Procedure and to send back outputs
- Unlike High-level language, no parameters list exist in assembly language procedures
- Programmer dependent: mostly general purpose registers and stack are used

Procedure Documentation

- Required to inform reader about procedure purpose / job, where it will read input and where it will deliver output

  ; (describe what the procedure does)

  ; input: (where it receives info. From calling program)

  ; output: (where it will deliver results to the calling program)

  ; uses: (a list of other procedures that it calls)

Multiplication (MUL, IMUL)

Example

```
MOV AL, 200
MOV BL, 4
MUL BL
 RET
```

0C8h

AL

0320h (800)

AX

MUL Instruction

- For unsigned multiplication
- Format: **MUL Source** (Operand1 could be: 8 bit or 16bit REG, memory)
- Algorithm:
  - When operand is a **byte**
    AX = AL * Source(operand1)
  - when operand is a **word**
    (DX AX) = AX * operand.

- CF=OF=0 when high section (upper half of destination) of the result is zero.

C	O	Z	P	S	A
r	r	?	?	?	?

Multiplication (MUL, IMUL)

Example

```
MOV AL, -2
MOV BL, -4
IMUL BL
RET
```

8

AX

IMUL Instruction

- Signed multiply
- Format: **IMUL Operand1** (Operand1 could be: 8 bit or 16bit REG, memory)
- Algorithm:
  - When operand is a **byte**
    AX = AL * operand.
  - when operand is a **word**
    (DX AX) = AX * operand.

- CF=OF=0 when upper half of the result is the sign extension of the lower half.
- =1 otherwise

C	O	Z	P	S	A
r	r	?	?	?	?

Division (DIV, IDIV)

Example

```
MOV AX,203
MOV BL, 4
DIV BL
RET
```

00CBh	AX
50 (32h)	AL
3	AH

DIV Instruction

- Unsigned divide
- Format: **DIV** **Divisor** (Divisor could be: 8 bit or 16bit REG, memory)
- Algorithm: (Dividend is of double size than divisor)
  - When operand is a **byte**
    AL = AX / operand
    AH=remainder (modulus)
  - when operand is a **word**
    AX=(DX AX) / operand
    DX=remainder (modulus)
- Quotient and remainder have the same size as divisor
- For DIV/IDIV, all status flags are undefined

C	O	Z	P	S	A
?	?	?	?	?	?

Division (DIV, IDIV)

Example

```
MOV AX,-203
MOV BL, 4
IDIV BL
RET
```

0FF35 h	AX
-50 (0CEh)	AL
-3 (0FDh)	AH

IDIV Instruction

- Signed divide
- Format: **IDIV** **Divisor** (Divisor could be: 8 bit or 16bit REG, memory)
- Algorithm:
  - When operand is a **byte**
    AL = AX / operand.
    AH=remainder (modulus)
  - when operand is a **word**
    AX=(DX AX) / operand.
    DX=remainder (modulus)

Divide Overflow:

- Quotient too big to fit in AL/AX
- Happens when Divisor is too small
- "Divide Overflow" error msg will print and program terminates

- Remainder has the same sign as dividend
- For DIV/IDIV, all status flags are undefined

C	O	Z	P	S	A
?	?	?	?	?	?

Sign Extension of Dividend

CBW Instruction

- Convert byte into word
- Format: **CBW** (No Operands)
- Algorithm:
  - If high bit of AL = 1 then,
    AH = 255 (0FFh)
  - else
    AH = 0

Example

```
MOV AX, 0 ; AH = 0, AL = 0
MOV AL, -5 ; AX = 000FBh (251)
CBW          ; AX = 0FFFBh (-5)
RET
```

CWD Instruction

- Convert word into double word
- Format: **CWD** (No Operands)
- Algorithm:
  - If high bit of AX = 1 then,
    DX = 65535 (0FFFFh)
  - else
    DX = 0

C	O	Z	P	S	A
Unchanged					