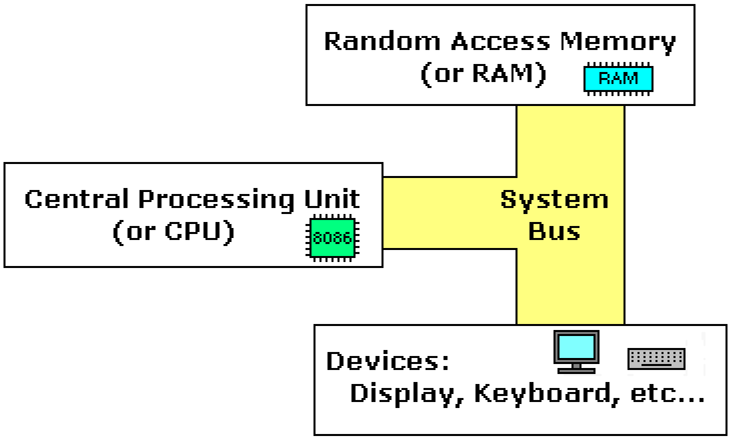
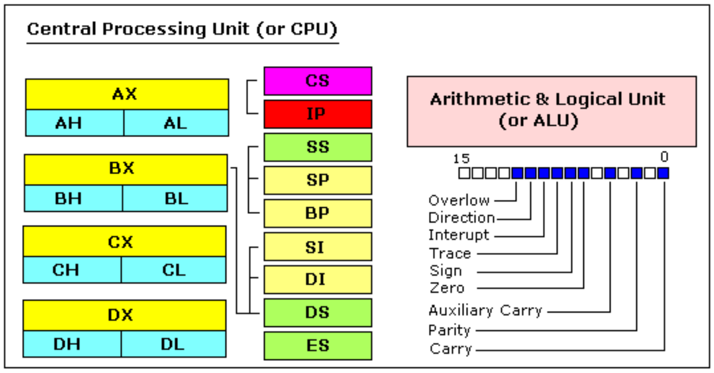
Contents

Assembly Language

- Assembly language is a low-level programming language.
- The simple computer mode



- Inside the CPU
  - **AX** - the accumulator register.
  - **BX** - the base address register.
  - **CX** - the count register.
  - **DX** - the data register.
  - **SI** - source index register.
  - **DI** - destination index register.
  - **BP** - base pointer.
  - **SP** - stack pointer



- 4 general purpose registers (AX, BX, CX, DX) are made of two separate 8 bit registers, for example if AX= 0011000000111001b, then AH=00110000b and AL=00111001b.
- The segment registers have a very special purpose - pointing at accessible blocks of memory.
  - **CS** - points at the segment containing the current program.
  - **DS** - generally points at segment where variables are defined.
  - **ES** - extra segment register, it's up to a coder to define its usage.
  - **SS** - points at the segment containing the stack.

Memory Access

- *MOV* instruction is used to copy the **second operand** (source) to the **first operand** (destination).
    - both operands must be the same size, which can be a byte or a word.

these types of operands are supported:

MOV REG, memory
MOV memory, REG
MOV REG, REG
MOV memory, immediate
MOV REG, immediate

REG: AX, BX, CX, DX, AH, AL, BL, BH, CH, CL, DH, DL, DI, SI, BP, SP.
memory: [BX], [BX+SI+7], variable
immediate: 5, -24, 3Fh, 10001101b

```
org 100h

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

ret
```

Segments

- A program consists of at least three segments:
  - stack segment: $.STACK$
  - data segment: $.DATA$
  - code segment: $.CODE$
- To define variables in the data segment, the compiler supports two data types: **BYTE** and **WORD**

name **DB** *value*

name **DW** *value*

- To define a code segment

```
              .CODE
LABEL         PROC         FAR/NEAR
            instructions ...
LABEL         ENDP
              END       LABEL
```

```
      .MODEL SMALL
      .STACK 64
      .DATA

DATA1    DB   52H
DATA2    DB   29H
SUM      DB   ?

      .CODE
MAIN     PROC     FAR                  ;this is the program entry point
    mov ax, @DATA                      ;load the data segment address
    mov ds, ax                         ;assign value to DS
    mov al, DATA1                      ;get the first operand
    mov bl, DATA2                      ;get the second operand
    add al, bl                         ;add the operands
    mov SUM, al                        ;store the result in location SUM
    MOV AH,4CH                         ;set up to return to DOS
    INT 21H
MAIN ENDP
     END     MAIN
```

- You can access the value of any element in array using square brackets [ ].

```
    .DATA
arr DB  55h, 12h, 11h, 10h

    .CODE
MAIN    PROC    FAR
mov ax, @DATA
mov ds, ax

    mov al, arr[0]
    mov bl, arr[3]

MOV AH,4CH
INT 21H
MAIN ENDP
     END    MAIN
```

Interrupts

- Interrupts can be seen as functions to do a specific task.
- To make an interrupt: $INT\ value$
  - Where value can be a number between 0 to 255 (or 0 to 0FFh)
- Each interrupt may have sub-functions.
  - To specify a sub-function AH register should be set before calling interrupt.
- To print a character on the screen, use the interrupt $INT\ 10H/0EH$.

```
MAIN    PROC    FAR
    mov ax, @DATA
    mov ds, ax

    mov ah, 0eh

    mov al, 'H'
    int 10h

    mov al, 'E'
    int 10h

    mov al, 'L'
    int 10h

    mov al, 'L'
    int 10h

    mov al, 'O'
    int 10h


    MOV AH,4CH
    INT 21H
MAIN ENDP
     END    MAIN
```

Arithmetic and Logic Instructions

- ADD: adds the second operand to the first.
    - $op1 = op1 + op2$

```
MOV AL, 5    ; AL = 5
ADD AL, 3    ; AL = 8
```

- SUB: subtracts the second operand from the first.
    - $op1 = op1 - op2$

```
MOV AL, 5
SUB AL, 1           ; AL = 4
```

- MUL: multiplies an operand by the value in the AL or AX.
    - when operand is a **byte**: $AX = AL * operand.$
    - when operand is a **word**: $(DX\ AX) = AX * operand.$

```
MOV AL, 200    ; AL = 0C8h
MOV BL, 4
MUL BL         ; AX = 0320h (800)
```

- DIV: divides the value in the AX or $(DX\ AX)$ by an operand
    - when operand is a **byte**: $AL = AX\ /\ operand$
      AH = remainder (modulus)
    - when operand is a **word**: $AX = (DX\ AX)\ /\ operand$
      DX = remainder (modulus)

```
MOV AX, 203    ; AX = 00CBh
MOV BL, 4
DIV BL         ; AL = 50 (32h), AH = 3
```

- AND: Logical AND between all bits of two operands.

```
MOV al, 011101b
AND al, 100010b
```

- OR - Logical OR between all bits of two operands.

```
MOV al, 011101b
OR al, 100010b
```

- XOR - Logical XOR (exclusive OR) between all bits of two operands.

```
MOV al, 011101b
XOR al, 100010b
```

- NOT - Reverse each bit of operand.

```
MOV AL, 00011011b
NOT AL   ; AL = 11100100b
```

- NEG - Make operand negative (two's complement). Reverses each bit of operand and then adds 1 to it.

```
MOV AL, 5   ; AL = 05h
NEG AL      ; AL = 0FBh (-5)
NEG AL      ; AL = 05h (5)
```

- CMP: subtract second operand from first operand and compare the operands by changing the flags only.
    - result is not stored anywhere, flags are set (OF, SF, ZF, AF, PF, CF) according to result.

```
MOV AL, 5
MOV BL, 5
CMP AL, BL  ; AL = 5, ZF = 1 (so equal!)
```

- $TEST$: Logical AND between all bits of two operands for flags only.
    - These flags are affected: ZF, SF, PF. Result is not stored anywhere.

```
MOV AL, 00000101b
TEST AL, 1          ; ZF = 0.
TEST AL, 10b        ; ZF = 1.
```

- INC: increment operand by 1.
  - $op = op + 1$

```
MOV AL, 4
INC AL       ; AL = 5
```

- DEC: decrement operand by 1.
  - $op = op - 1$

```
MOV AL, 4
DEC AL       ; AL = 3
```

- SHL: Shift operand1 Left.
  - The number of shifts is set by operand2.

```
MOV al, 11100000b
SHL al, 3
```

- SHR: Shift operand1 Right.
  - The number of shifts is set by operand2.

```
MOV al, 11100000b
SHR al, 3
```

Program Flow Control

- JMP: Unconditional Jump. Transfers control to another part of the program.

```
    mov al, 55h
    jmp label1
    add al, 5
label1:
    sub al, 5
```

- JE: Short Jump if first operand is Equal to second operand
  - set by CMP instruction.
  - if ZF = 1 then jump.

```
    mov al, 5
    cmp al, 5
    je label1
    add al, 10
label1:
    sub al, 2
```

- JNE: Short Jump if first operand is Not Equal to second operand
  - set by CMP instruction.
  - if ZF = 0 then jump.

```
    mov al, 5
    cmp al, 2
    jne label1
    add al, 10
label1:
    sub al, 2
```

- JZ: Short Jump if first operand is Equal to the second operand.
    - Set by CMP, SUB, ADD, TEST, AND, OR, XOR instructions.
    - if ZF = 1 then jump.

```
    mov al, 5
    xor al, 5
    jz label1
    add al, 10
label1:
    sub al, 2
```

- JNZ: Short jump if first operand is Not Equal to the second operand.
    - Set by CMP, SUB, ADD, TEST, AND, OR, XOR instructions.
    - if ZF = 0 then jump.

```
    mov al, 4
    test al, 5
    jnz label1
    add al, 10
label1:
    sub al, 2
```

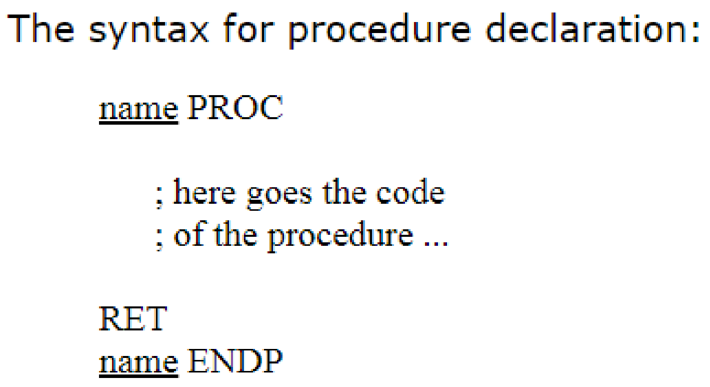
- $LOOP$: Decrease CX, jump to label if CX not zero.
    - Algorithm:

```
CX = CX - 1
if CX $\neq$ 0 then
        jump
else
        no jump, continue
```

```
    mov cx, 5
    mov al, 1
sum:
    add al, 1
    loop sum
```

Procedures

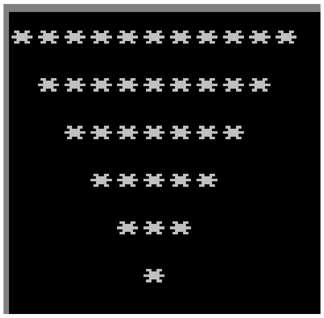
- Procedure is a part of code that can be called from your program in order to make some specific tasks.



The syntax for procedure declaration:

```
name PROC

      ; here goes the code
      ; of the procedure ...

name ENDP
RET
name ENDP
```

```
MAIN     PROC     FAR
mov ax, @DATA
mov ds, ax

    mov cl, 5
    mov al, 8

    call sum
    call subtract

MOV AH,4CH
INT 21H
MAIN ENDP

sum      PROC      NEAR
    add al, cl
    RET
sum      ENDP

subtract     PROC     NEAR
    sub al, cl
    RET
subtract     ENDP

     END     MAIN
```



```
***********
 *********
  *******
   *****
    ***
     *
```

```
    .DATA
crnt_bl  db ?
crnt_dl  db ?
new_line db 13, 10, "$"
    .CODE
MAIN     PROC     FAR     ;this is the program entry point
mov ax, @DATA             ;load the data segment address
mov ds, ax                ;assign value to DS
;-------------------------------------------------------;
    mov dl, 0                  ; left space
    mov bl, 12                 ; stars to print
    mov cx, 6                  ; line counter
    mov crnt_dl, dl
    mov crnt_bl, bl
START:
        cmp dl, 0
        jnz PRNT_SPACE
        jmp PRNT_STAR
PRNT_SPACE:
        mov al, ' '                ; print a space
        mov ah, 0Eh
        int 10h
        dec dl                     ; decrement dl
        jmp start                  ; jump to start
PRNT_STAR:
        mov al, '*'
        mov ah, 0Eh
        int 10h
        dec bl
        cmp bl, 1
        jnz PRNT_STAR
        jmp PRNT_LINE
PRNT_LINE:
        mov dx, offset new_line
        mov ah, 09
        int 21h
call update_counters
loop START
;-------------------------------------------------------;
MOV AH,4CH                ;return to DOS
INT 21H
MAIN ENDP

update_counters     PROC
    inc crnt_dl
    mov dl, crnt_dl
    dec crnt_bl     ;decremnt twice
    dec crnt_bl
    mov bl, crnt_bl
    ret
    update_counters ENDP

END    MAIN
```

Instruction Summary

- ADD
- AND
- CALL
- CMP
- DEC
- DIV
- INC
- INT
- JE
- JMP
- JNE
- JNZ
- JZ
- LOOP
- MOV
- MUL
- NEG
- NOT
- OR
- SHL
- SHR
- SUB
- TEST
- XOR