# CT215-3
# Coding in
# IBM PC
# Assembly Language

# Requirements for Coding in Assembly Language

- **Assemblers**
- **Program Comments**
- **Reserved Words**
- **Identifiers**
- **Statements**
- **Directives**

# Assemblers

- translate a source code in low-level language, assembly, to machine code (object code)
- each instruction in assembly code generates one machine instruction
- linker program converts the object code to executable machine language

# *Assembly Language Syntax*:

## *Statements*

Program consist of statements , one per line . Each statement is either an **instruction**, which assembler translates into machine code ,or an **assembler directive**, which instructs the assembler to perform some specific task, such as allocating memory space for variable or creating a procedure. Both instructions and Directives have up to four fields:

# Statements

- ***Instruction*** **-- a statement that is executed by the processor at runtime**
- ***Directive*** **-- a statement that affects either the program listing or the way machine code is generated**

| [identifier] | [operation] | [operands] | [;comment] |
|---|---|---|---|

**Maximum of 132 characters per line**

# Examples

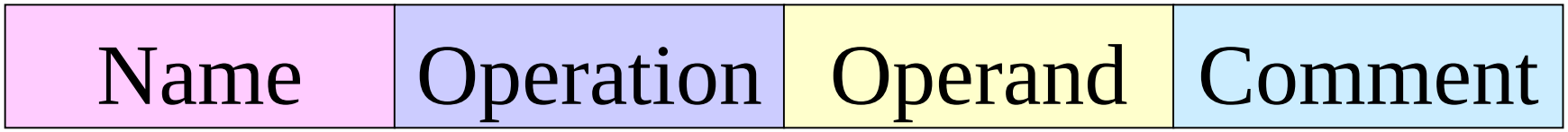
## Instructions:

```
CALL MySub ;transfer of control
MOV  AX,5  ;data transfer
ADD  AX,20 ;arithmetic
JZ   Next1 ;logical (jump if zero)
IN   A1,20 ;input/output (read from
  hardware port)
RET        ;return
```

# Examples

## Directives:

```
COUNTER DB 50     ;defined byte with
                    50
TOTAL   DW 4126H ; defined word
                    with 4126
```




Example : Instruction

```
Start:      MOV      CX , 5      ;Initialize counter
```

Example : Directive

```
MAIN        PROC
```

MAIN is the name , and the operation field contains PROC . This particular directive creates a procedure called MAIN.

# Identifiers

**A name assigned to an item in the program for referencing**

- ***Variable*** -- refers to address of a data item

```
COUNTER DB 0
```

- ***Label*** -- refers to the address of an instruction, procedure, or segment

```
MAIN PROC FAR
```

# Rules of Naming Identifiers

- alphabetic letters and digits -- begin with an alphabetic letter
- special characters: ? _ $ @ . -- not the starting one
- not case insensitive
- 31 characters (247 since MASM 6.0)
- register names are reserved

```
TOTAL, QTY250, NEXT, $P50
```

## Example of legal name :

COUNTER1

@CHARACTER

SUM_OF_DIGITS

$100

DONE?

.TEST

## Example of illegal name:

TWO WORDS ; contains a blank

2abc ; begin digits

A45.28 ; . Not first char

YOU&ME ; illegal char

# *Operation field :*

For an instruction ,the operation field contains symbolic operation code (Op-code) . The assembler translates a symbolic op-code into machine code . Example MOV , ADD , SUB

In an assembler directives, the operation field contains a pseudo-operation code ( pseudo-op) . pseudo- ops are not translated into machine code; rather, they simply tell the assembler to do something . Example PROC ( create procedure)_

# *Operand Field :*

For an instruction, the operand field specifies the data that are to be acted on by operation . An instruction may have zero ,one or two operands .Example

```
NOP                 ;no operand

INC   AX            ; one operand

ADD  WORD1,2        ; two operand
```

# Program Comments

- Documentation purposes
- ; -- comment symbol
- comments don’t generate machine code

```
MOV AX,0123   ;  move value 0123H to AX
SUB AX,BX     ;subtract contents of BX to AX
```

# Reserved Words

**Reserved words are name with predefined meaning**

**Instructions -- operations**

- **MOV, ADD**

**Directives -- provides information to assembler**

- **DB, END, SEGMENT**

**Operators -- use in expression**

- **FAR, SIZE**

**Predefined symbols -- return information to program**

- **@Data, @Model**

# Directives

## PAGE directive

control the format of source code

```
PAGE [length][,width]
```

```
PAGE 60,132  ; 60 lines/page
             132 characters/line
```

default -- `PAGE 50,80`

# TITLE Directives

print a title on line 2 of each page

```
TITLE [text][comment]
```

```
TITLE HELLO Hello World Program
TITLE ASM Assembly Program
```

Maximum length 60 characters

# SEGMENT Directive

- **Define the start of a segment (code, data, and stack)**
- **segment name must**
  - be present
  - be unique
  - follow assembler naming rules
- **Maximum size of a segment in real mode is 64K**

# SEGMENT Directive

| NAME | OPERATION | OPERAND | COMMENT |
|---|---|---|---|
| name | SEGMENT | [options] | ;begin segment |
| | . | | |
| | . | | |
| | . | | |
| Name | ENDS | | ;end segment |

```
name    SEGMENT align combine ‘class’
```

# SEGMENT Directive Options

- **Alignment type -- *align* entry indicates the starting boundary of a segment**
  - **PARA -- the segment aligns on a paragraph boundary**

```
name      SEGMENT  PARA
```

- **Combine type -- *combine* entry indicates whether the segment is to be combined with another segment at link time.**
  - **STACK, COMMON, PUBLIC, AT; default -- NONE**

```
name      SEGMENT  PARA  STACK
```

# Class Type

- ***Class* entry is used to group related segment when linking**
  - **Code, Data, and Stack**

```
name    SEGMENT PARA STACK 'STACK'
```

# PROC Directive

- **Code segment contains executable code for a program**

```
NAME      OPERATION   OPERAND      COMMENT
segname   SEGMENT     PARA
procname  PROC        FAR          ;one procedure
          .                        ;in a code
          .                        ;segement
          .
procname  ENDP
Name      ENDS                     ;end segment
```

# PROC Directive

- **Operands -- FAR, NEAR**
- **The first procedure in  a segment must have the FAR operator.  It indicates where execution is to start.**
- **Others procedures usually have the NEAR operator.**

# ASSUME Directive

- **Associates a segment register to address the segment using the segment name at assembly time**
- **Programmer may still have to code instructions at execute time to load physical address in segment registers**

| OPERATION | OPERAND |
|---|---|
| `ASSUME` | `SS:stackseg,CS:codeseg,DS:dataseg` |

# END Directive

- **End a section of a code segment**
  - ENDS -- end a segment
  - ENDP -- end a PROC
  - END -- end the entire program

| OPERATION | OPERAND |
|---|---|
| `END` | `[procname]` |

# Program data :

The processor operates only on binary data. Thus , the assembler must translate all data representation into binary numbers. However , in an assembly language program we may express data as binary, decimal or hex numbers, and even as characters.

| *Number* | *Type* |
| --- | --- |
| 11011 | decimal |
| 11011B | binary |
| 64223 | decimal |
| -218430D | decimal |
| 1B4DH | hex |
| 1B4D | illegal hex number |
| 1,234 | illegal |
| FFFFH | illegal |

# Characters :

Characters and character strings must be enclosed in single or double quotes for example “ A “ , ‘Hello’ Character translated into their ASCII codes by the assembler. So there is no difference between using “A” and 41H in a program.

# Program Structure

```
                PAGE   60, 132
TITLE           TASMPROG1     Skeleton of an .EXE Program
STACKSG         SEGMENT PARA STACK 'Stack'
                ...
STACKSG         ENDS
DATASG          SEGMENT PARA 'Data'
                …
DATASG          ENDS
CODESG          SEGMENT PARA 'Code'
MAIN            PROC     FAR
                …
MAIN            ENDP                    ; End of procedure
CODESG          ENDS                    ; End of segment
                END    MAIN             ; End of program
```

```
 1          PAGE 60, 123
 2 TITLE    A04 ASM1 Skeleton of an .EXE Program
 3 ;----------------------------------------
 4 stacksg SEGMENT PARA STACK 'STACK'
 5          ...
 6 stacksg ENDS
 7 ;----------------------------------------
 8 datasg  SEGMENT PARA 'DATA'
 9          ...
10 datasg  ENDS
11 ;----------------------------------------
12 codesg  SEGMENT PARA 'code'
13 main    PROC    FAR
14          ASSUME  SS:stacksg,DS:datasg,CS:codesg
15          MOV     AX,datasg ;set address of data
16          MOV     DS,AX     ;  segment in DS
17          ...
18          MOV     AX,4C00H  ;end processing
19          INT     21H
20 main    ENDP              ;end of procedure
21 codesg  ENDS              ;end of segment
22          END     main      ;end of program
```

```
          PAGE 60, 123
TITLE   A04ASM1 Move and add instructions
;-----------------------------------------
stacksg SEGMENT PARA STACK 'STACK'
        DW      32 DUP(0)
stacksg ENDS
;-----------------------------------------
datasg  SEGMENT PARA 'DATA'
FLDD    DW      175
FLDE    DW      150
FLDF    DW      ?
datasg  ENDS
;-----------------------------------------
codesg  SEGMENT PARA 'code'
main    PROC    FAR
        ASSUME  SS:stacksg,DS:datasg,CS:codesg
        MOV     AX,datasg ;set address of data
        MOV     DS,AX     ;  segment in DS

        MOV     AX,FLDD   ;move 0175 to AX
        ADD     AX,FLDE   ;move 0150 to AX
        MOV     FLDF,AX   ;store sum in FLDF

        MOV     AX,4C00H  ;end processing
        INT     21H
main    ENDP              ;end of procedure
codesg  ENDS              ;end of code segment
        END     main      ;end of program
```

# Simplified Segment Directive

- **Memory model -- a shortcut in defining segments**

**.MODEL memory-model**

| Model | # code seg | # data seg |
|---|---|---|
| TINY | **.COM** | **.COM** |
| SMALL | **1** | **1** |
| MEDIUM | **More than 1** | **1** |
| COMPACT | **1** | **More than 1** |
| LARGE | **More than 1** | **More than 1** |

# Memory Models

- **TINY is used for .COM programs that have code, data, stack segments in one 64K segment**
- **SMALL requires code to fit in a 64K segment and data to fit in another 64K segment**
- **.MODEL automatically generates ASSUME statement**

# Format for Defining Segments

```
.STACK  [size]  ;default name STACK
                ;default 1024 bytes
.DATA           ;default name _DATA
.CODE   [name]  ;default name _TEXT
```

```
          PAGE 60, 123
TITLE     A04ASM1 Move and add instructions
;---------------------------------------
          .MODEL SMALL
          .STACK 64              ;define stack
          .DATA                  ;define data
FLDD      DW       175
FLDE      DW       150
FLDF      DW       ?
;---------------------------------------
          .CODE                  ;define code segment
main      PROC     FAR
          MOV      AX,@data      ;set address of data
          MOV      DS,AX         ;  segment in DS

          MOV      AX,FLDD       ;move 0175 to AX
          ADD      AX,FLDE       ;move 0150 to AX
          MOV      FLDF,AX       ;store sum in FLDF

          MOV      AX,4C00H      ;end processing
          INT      21H
main      ENDP                   ;end of procedure
          END      main          ;end of program
```

# Data Definition

| [name] | D[N] | expression |
|---|---|---|

- **Use to define the size of a data item**
  - **constant -- numeric value, character, string**
  - **undefined -- uninitialized**
- **Name -- reference to a data item**
- **Directive (D[N]) -- DB, DW, DD, DF, DQ, DT**
- **Expression -- uninitialized, initialize constant**

| Pseudo-op | stands for |
|---|---|
| DB | define byte |
| DW | define word |
| DD | define double word |
| DQ | define quadword |
| DT | define tenbytes |

# Examples

```
Counter DB  ?
Length  DB  25
vector  DB  21,22,23,24,25
        ...
        MOV  AL,vector+2 ;value 17H
```

Duplication expression that defines vectors

```
Counter DW 10 DUP(?)
Length  DB 5 DUP(12)
vector  DB 3 DUP(5 DUP(4))
```

# Character Strings

```
stuname DB “Billy John”
stuname DB ‘Billy John’
        DB “Billy John’s”
```

# Numeric Constant

- **Binary -- `11B`**
- **Decimal -- `12 or 12D`**
- **Hexadecimal -- `0DE8H`**
- **Real -- `23R`**

```
char DB ‘15’
num  DB  15
```

# EQU -- Equate Directive

- **Associate the value to a name**

```
factor EQU 12    ;in data segment
...
tablex DB  factor DUP(?)
```

- **Equated operand**

```
width EQU 11
      ...
      MOV CX,width
```

# EQU Directive

- **Equate symbolic name to a name**

```
width   DB 11
...
twidth  EQU width
```

- **Does not occupy a memory location**

# NAME CONSTANT
# EQU Directive

| Name | EQU | Constant |
|---|---|---|
| LF | EQU | 0Ah |

**MOV DL,0Ah or**

**MOV Dl,LF**

---

| | | |
|---|---|---|
| PROMPT | EQU | ‘TYPE YOUR NAME’ |
| MSG | DB | PROMPT |