Lecture Title

Course Code: Course Title:

Dept. of Computer Science
Faculty of Science and Technology

Lecturer No:	**2**	**Week No:**	**2**	**Semester:**	**Sp 21-22**
Lecturer:	*Dr. Md. Sohidul Islam* *sohidul@aiub.edu*				

Lecture Outline

1. Learn Syntax
2. Variable declarations
3. Introduction of basic data movement
4. Program organization: Code, Data and stack

Assembly Language Syntax

- Assembly language is **not case sensitive**, however, we use upper case to differentiate code from rest of the text.

➢ **Statements:**

  - Programs consist of statements (one per line)
  - Each statement can be any of following types:
    - Instruction that are translated into machine code
    - Assembler directives that instruct the assemble to perform some specific task:
      - ➢Allocating memory space for variables
      - ➢Creating procedure

Fields

- Instructions and directives can have up to **four fields**:

Name Operation Operand(s) comment

START MOV CX,5 ; initialize counter

****[Fields must appear in this order]**

MAIN PROC [creates a Procedure]

- At least one **blank** or **tab** character must separate the fields

Name Field

- **Name**: it is used for instruction levels, procedure names and variable names.
  - The assembler translates names into variable names.
  - Can be 1 to 31 characters long and consists of letter , digit and special characters.
  - Embedded blanks are not allowed.
  - Names may not begin with number.
  - **UPPERCASE** and **lowercase** in name are same.
  - **Examples: COUNTER1, $1000, Done?, .TEST**
    - **Illegal names TWO WORD, 2AB, A45.28, ME &YOU**

Solve the Following

- Which of the following names are legal in IBM PC assembly language?

TWO_WORDS

TwoWOrDs

?1

.@?

$145

LET'S_GO

T = .

Operation Field

- Operation field contains a symbolic operation code (opcode).
- The assembler translates a symbolic opcode into a machine language.
- Opcode symbols often describe the **operations function** (e.g. **MOV, ADD, SUM** etc..).
- In assembler directive, the operation field contains pseudo operation code (pseudo-ops).
- Pseudo-ops are NOT translated into machine code. they simply **tell** the assembler to do something.

  e.g. **PROC** pseudo-op is used to create procedure.

- Operand field species the data that are to be **acted on** by the operation.
- An instruction may have zero, one or two operands. e.g.
- First operand is **Destination** (i.e. register or Memory location)

  some instruction do not store any result
- Second operand is **Source** and its not usually modified by instruction

Comment Field

- **Comment:** put instruction into the context of program.
- Comment field of a statement is used to say something about what the statement does?
- **Semicolon (;) marks in the beginning of this field**
- Assembler ignores anything typed after " ; "

** Comment is very important in assembly language and it is almost impossible to understand assembly code without comment.

** Commenting is considered as good programming practice

Program Data

- Processor operates only on **binary** data.
- So, the assembler MUST **translate** all data representation into binary numbers.
- In assembly program, we may express data as **binary**, **decimal** or **hex** numbers and even characters.

➢ **Numbers:**

- **Binary:** a binary number is written as bit string followed by the letter **B** or **b** (e.g. 1010**B**)
- **Decimal:** A decimal number is a string of decimal digits. It ends with optional "D" or "d" (e.g. 1234).
- **Hex:** A hex number begins with a decimal digit and ends with the letter **H** or **h (e.g.** 12AB**h**).

➢ **Characters:**

Character strings must be enclosed with single or double quotes.

- **e.g. 'A' or "hello" is translated into ASCII by assembler. So, there is no difference between 'A' or 41h or 65d.**

Solve the Following

- Which of the following are legal numbers? if they are legal tell whether they are Binary, decimal or hex numbers?

246

246h

1001

1,001

2A3h

FFFEh

0Ah

Bh

1110b

Variables

- We use a variable to store values temporarily.
- Each variable has a data type and is assigned a memory address by the program.
- We will mostly use DB (define byte) and DW(define word) variables.

➢ **Byte Variables:** In the following , the directive **associates a memory byte to ALPHA and initialize it to 4.** A **"?"** mark can be used for uninitialized byte. The range of values in a byte is **2^8 or 256**

Name DB Initial_value

ALPHA DB 4

➢ **Word Variables:** Similar to byte variable and the range of initial values is **2^16 or 65536.**

Name DW Initial_value

WRD

Arrays

- Array is just a **sequence** of bytes or words.
- i.e. to define a three-byte array, we write

B_ARRAY DB 10h,20,30h

Name B_ARRAY is associated with first byte, B_ARRAY+1 with second and B_ARRAY+2 with third.

B_ARRAY	**200**	**10H**
B_ARRAY+1	**201**	**20H**
B_ARRAY+2	**202**	**30H**

Array Exercise

- Create a word array (named MY_W_ARRAY) table of which the starting address is 500 and values are 2000,323,4000 and 1000.

MY_W_ARRAY DW 2000,323,4000,1000

MY_W_ARRAY	**500**	**2000**
MY_W_ARRAY+2	**502**	**323**
MY_W_ARRAY+4	**504**	**4000**
MY_W_ARRAY+6	**506**	**1000**

High and Low bytes of Word

- Sometimes we may need to refer to the high and **low bytes** of a word variable. i.e. if we define,

WORD1 DW 1234H

the **low byte** of WORD1 contains 34h (symbolic address: WORD1) and **High byte** contains 12h (symbolic address: WORD1+1).

- **Character string:** An array of ASCII codes.
  - LETTER DB 'ABC'
  - LETTER DB 41h,42h,43h [UPPERCASE]
  - MSG DB `HELLO', 0Ah, 0Dh, '$' [combination is also possible]
  - MSG DB 48h,45h,4Ch,4Ch,4Fh,0Ah,0Dh,24h

Named Constant

- Using a symbolic name for constant quantity make the assembly code much easier.

➢ **EQU (Equates):** Assign a name to a constant

e.g. LF EQU 0Ah [LF= 0Ah]

(LF=0Ah is applicable to whole code after assigning)

- **PROMPT EQU 'Type Your Name'**

No memory is allocated for EQU names

Instructions: MOV

- **MOV** is used to **transfer** data between registers, register and memory-location or move number directly into register or memory location.
- **Syntax:** **MOV destination, source**

  MOV AX, WORD1 [reads Move WORD1 to AX]

example

```
.MODEL SMALL
.STACK 100H
.DATA

A DW 2h
B DW 5h
SUM DW ?
.CODE

MAIN PROC
  MOV AX, @DATA
   MOV DS, AX

   MOV AX,A
   ADD AX,B
   ;MOV SUM,AX

EXIT:
   MOV AX,4CH
   INT 21H

  MAIN ENDP
END MAIN
```

References

- Assembly Language Programming and Organization of the IBM PC, Ytha Yu and Charles Marut, McGraw Hill, 1992. (ISBN: 0-07-072692-2).
- https://www.tutorialspoint.com/assembly_programming/index.htm

Books

- Assembly Language Programming and Organization of the IBM PC, Ytha Yu and Charles Marut, McGraw Hill, 1992. (ISBN: 0-07-072692-2).

- Essentials of Computer Organization and Architecture, (Third Edition), Linda Null and Julia Lobur

- W. Stallings, “Computer Organization and Architecture: Designing for performance”, 67h Edition, Prentice Hall of India, 2003, ISBN 81 – 203 – 2962 – 7

- Computer Organization and Architecture by John P. Haynes.