Assembly Language Syntax



Course Title: Computer Organization & Architecture

Dr. Nazib Abdun Nasir Assistant Professor CS, AIUB nazib.nasir@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 is **not case sensitive**, however, we use <u>UPPER CASE</u> to differentiate code from rest of the text.

> Statements:

- Programs consist of statements (one per line).
- Each statement can be any of following types:
 - Instructions that are translated into machine code.
 - Assembler directives that instruct the assembler to perform some specific tasks:
 - ➤ Allocating memory space for variables
 - Creating procedures



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, ME &YOU

TOOL TOOL

Solve the Following

- ➤ Which of the following names are legal in IBM PC assembly language?
- TWO_WORDS
- TwoWOrDs
- 2-words
- ?1
- \$145
- LET'S_GO
- T = time

Operation Field



- Operation field contains a symbolic operation code (opcode).
- The assembler translates a symbolic opcode into a machine language.
- Opcode symbols describe the operations function (e.g. MOV, ADD, SUM).
- In assembler directive, the operation field contains pseudooperation code.
- 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.

TOOLA DESIGNATION OF THE PARTY OF THE PARTY

Operand Field

- **Operand** field specifies the data that are to be **acted on** by the operation.
- Some instructions do not store any result.
- An instruction may have zero, one or two operands.
- First operand is **Destination** (i.e. register or Memory location).
- Second operand is Source and its not usually modified by instruction.

SA PARTIES OF THE PAR

Comment Field

- **Comment:** Put instructions 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.

TOOL TOOL

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

TOOL TOOL

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: The following 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 4h

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

Name DW Initial_value WRD DW 4021h



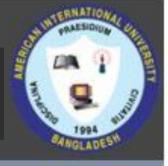
Arrays

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

B_ARRAY DB 10H,20H,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	\mathbf{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

MAESIDIDA MAESIDIDA MOLADESIN

High and Low bytes of Word

- Sometimes we may need to refer to the **high** and **low** bytes of a word variable.
- ➤ If we define like the below, the **low byte** of WORD1 contains 34h (symbolic address: WORD1) and **high byte** contains 12h (symbolic address: WORD1+1).

WORD1 DW 1234H

- > Character String: An array of ASCII codes.
 - LETTER DB 'ABC'
 - LETTER DB 41h,42h,43h
 - MSG DB 'HELLO', 0Ah, 0Dh, '\$'
 - MSG DB 48h,45h,4Ch,4Ch,4Fh,0Ah,0Dh,24h

TOOLA DESIGNATION OF THE PARTY OF THE PARTY

Named Constant

- Using a symbolic name for constant quantity make the assembly code much easier.
- ➤ EQU (Equates): Assign a name to a constant LF EQU 0Ah [LF= 0Ah]
- (LF=0Ah is applicable to whole code after assigning)



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]

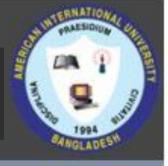


Instructions: XCHG

> XCHG is used to exchange the contents between two registers or register and memory-location.

> Syntax: XCHG destination, source

XCHG AH, **BL** [exchange value of AH with BL]



Instructions: ADD

➤ **ADD** is used to **add** content of two registers, register and memory-location or add a number to register or memory location.

> Syntax: **ADD** destination, source

ADD WORD1, AX [reads Add AX to WORD1]



Instructions: SUB

> SUB is used to subtract content of two registers, register and memory-location or subtract a number from register or memory location.

> Syntax: **SUB** destination, source

SUB AX, DX [reads Subtract DX from AX]



Instructions: INC

> INC is used to add 1 to the content of a register or memory-location.

> Syntax: INC destination

INC WORD1 [reads Add 1 to WORD1]



Instructions: DEC

➤ **DEC** is used to **subtract 1** from the content of a register or memory-location.

> Syntax: **DEC** destination

DEC WORD1 [reads subtract 1 from WORD1]



Instructions: NEG

- > **NEG** is used to **negate** the content of the destination.
- > NEG does this by replacing the content by its two's complement.

> Syntax: **NEG** destination

NEG BX [reads negate the content of BX]

TOPA OTHOGRAPH OF THE PARTY OF

Instruction: LEA

➤ LEA (Load Effective Address) puts copy of the source offset address into the destination.

> Syntax:

LEA destination, source

LEA DX, MSG; will load address of MSG to DX



Agreement of Operator

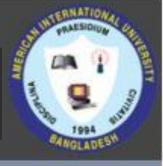
• The operands of a two-operand instruction MUST be same type. (i.e. both bytes or words). Thus,

• MOVAX, BYTE1 ; its illegal

• MOV AH, 'A'; legal

• MOVAX, 'WD' ; legal if source is a word

Program Structure



- > A program consists of
 - Stack
 - Data
 - Code
- Each part occupies memory segments.
- > Program segment is **translated** into memory segment by assembler.
- The size of code and data of a program can be specified by memory model using **.MODEL** directive.

.MODEL Memory_model

.MODEL SMALL

[Code in one segment and Data in one segment]

TOOL ANGLADED

Stack Segment

- Allocate a block of memory (stack area) to store the stack.
- The stack area should be big enough to contain the stack at its maximum size.
- Declaration:

.STACK size

.STACK 100H

- ** Allocates 100 bytes for stack area which is reasonable size for most applications.
- ** If size is omitted 1KB is allocated for stack area.



Data Segment

- Contains all the **variable** definitions and sometimes constant definitions (constant does not take any memory).
- ➤ To declare data segment, **.DATA** directive is used followed by variable and constant declarations.

> Declaration:

.DATA

WORD1 DW 2

BYTE1 DB 1

MSG DB 'THIS IS A MESSAGE'

MASK EQU 10010001B



Code Segment

- > Contains the program's instructions.
- > Declaration:

.CODE name [name is optional]

- There is no requirement of **name** in **SMALL** program
- ➤ Inside a code segment, instructions are organized as procedures.

name PROC

; body of the procedure

name ENDP

- > Here **name** is the name of the procedure.
- **PROC** and **ENDP** are pseudo-ops.



Program Structure

.MODEL SMALL

.STACK 100H

.DATA

; data definitions here

.CODE MAIN

MAIN PROC

; instructions go here

MAIN ENDP

; other procedures go here

END MAIN

• *** The last line of the program should be the END directive, followed by the name of main procedure.



Program Segment Prefix (PSP)

- PSP contains information about the program to facilitate the program access in this area.
- DOS places its segment number in both DS and ES before program execution.
- Usually, DS does not contain the segment number of the data segment.
- Thus, a program with data segment will start with these two instruction.

MOV AX, @DATA [name of data segment defined in DATA] MOV DS, AX

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.