

Assembly Language Programming

Course Code: 0052

Course Title: Computer Organization and
Architecture



Dept. of Computer Science
Faculty of Science and Technology

Lecturer No:	4(a)	Week No:	4	Semester:	FALL 23
Lecturer:	<u>Saeeda Sharmeen Rahman, sharmeen@aiub.edu</u>				



1. Creating, Assembling and executing assembly language program.

2. By the end of this lesson we will be able to write simple but interesting assembly program.

Overview



Four Steps

1. Learn Syntax
 2. Variable declarations
 3. Introduction of basic data movement
 4. Program organization: Code, Data and stack
- Assembly language instructions are so basic. So, I/O is much harder unlike high-level languages.
 - We Use DOS functions for I/O as they are easy to invoke and faster
 - A program is must be converted to machine language before execution

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 assembler 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(cont'd...)



- 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.

NOP	No operands; does nothing
INC AX	Adds one to the contents of AX
ADD WORD1,2	Add 2 to the contents of WORD1

- 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 DW -2

Array



- 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.

Solution



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

Array (Cont.)



- **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 Contant



- 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]

Before	After
0006	0008
AX	AX
0008	0008
WORD1	WORD1

****Copy of WORD is sent to AX**

Legal Combinations of Operands for MOV



Source Operand	General Register	Segment Register	Memory location	Constant
General Register	Yes	Yes	Yes	No
Segment Register	Yes	No	Yes	No
Memory location	Yes	Yes	No illegal: MOV W1,W2	No
Constant	Yes	No	Yes	No

Solve the Following



- What is the value of BX and A after MOV BX,A ?[assume value of A is 24h]
- Using previous values, find the value of AX and BX from MOV AX, BX
- Tell us whether the following instructions are legal or illegal?

MOV DS,AX

MOV DS,1000h

MOV CS,ES

MOV W1,DS

MOV W1,B1

Instructions: XCHG



- MOV is used to **exchange** the contents between **two registers or register and memory-location.**
- Syntax: **XCHG** **destination, source**

XCHG AH, BL

[reads exchange value of AH with BL]

Before		After	
1A	00	05	00
AH	AL	AH	AL
00	05	00	1A
BH	BL	BH	BL

Legal combinations of operands for XCHG



Source Operand	General Register	Memory location
General Register	Yes	Yes
Memory location	Yes	No illegal: XCHG W1,W2

Solve the following



- What is the value of BX and A after XCHG BX,A?[assume value of A is 15h].
- Also find, AX and A after MOV AX,A ?
- Using previous values, find the value of AX and BX from XCHG AX, BX?
- Tell us whether the following instructions are legal or illegal?

XCHG W1,W2

XCHG AX,W1

Solution



- **XCHG or MOV operation is not allowed between memory locations. So, What could be the way out?**

➤ Using Register,

`MOV AX, W2`

`MOV W1, W2`

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]

Before	After
01BC	01BC
AX	AX
0523	06DF
WORD1	WORD1

****Copy of WORD1 is added with content of AX and stored in WORD1**

Legal Combinations of Operands for ADD



Source Operand	General Register	Memory location
General Register	Yes	Yes
Memory location	Yes	No illegal: ADD W1,W2
Constant	Yes	Yes

Solve the Following



- What is the value of BX and A after ADD BX,A ?[assume value of BX is 5h and A is 9h]
- using previous values[AX=9h], find the value of AX and BX from ADD AX, BX
- Tell us whether the following instructions are legal or illegal?

ADD B1,B2

ADD AL,256

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]

Before	After
0000	FFFF
AX	AX
0001	0001
DX	DX

****Subtracts the content of DX from AX and stored in AX.**

Legal Combinations of Operands for ADD



Source Operand	General Register	Memory location
General Register	Yes	Yes
Memory location	Yes	No illegal: SUB W1,W2
Constant	Yes	Yes

Solve the Following



- What is the value of BX and A after SUB BX,A ?[assume value of BX is F and A is 9h]
- Using previous values[AX=9h], find the value of AX and BX from SUB AX, BX
- Tell us whether the following instructions are legal or illegal?

SUB B1,B2

SUB AL,256

Instructions: INC



- **INC** is used to **add 1** to the contents of a register or memory-location.
- **Syntax: INC destination**

INC WORD1 [reads Add 1 to WORD1]

Before	After
0002	0003
WORD1	WORD1

**** 1** is added to WORD1 and result is stored in WORD1

Solve the Following



- What is the value of BX and A?[assume BX=3h and A=9h]
 - INC BX
 - INC A

Instructions: DEC



- **DEC** is used to **subtract 1** from the contents of a register or memory-location.
- **Syntax: DEC destination**

DEC WORD1 [reads subtract 1 from WORD1]

Before	After
FFFE	FFFD
BYTE1	BYTE1

**** 1** is subtracted from BYTE1 and result is stored in BYTE1

Solve the Following



- What is the value of BX and A?[assume BX=3h and A=9h]
 - DEC BX
 - DEC A

Instructions: NEG



- **NEG** is used to **negate** the contents of the destination

NEG does this by replacing the contents by its two's complement.

- **Syntax: NEG destination**

NEG BX [reads negate the contents of BX]

Before	After
0002	FFFE
BX	BX

** The content of BX is replaced with its two's complement

Solve the Following



- What is the value of BX and A? [assume BX=3h and A=9h]
 - NEG BX
 - NEG A

Agreement of Operator



- The operand of the preceding two-operand instruction MUST be same type. (i.e. both bytes or words). Thus,
- **MOV AX,BYTE1 ; its illegal**
- **MOV AH,'A' ; legal**
- **MOV AX,'A' ; legal if source is a word**

Translation of High-Level Language to Assembly Language



Statement	Translation
B = A	MOV AX,A MOV B,AX ** A direct memory move is illegal
A = 5-A	MOV AX,5 SUB AX,A MOV A,AX or NEG A ADD A,5
A=B-2*A	MOV AX,B SUB AX,A SUB AX,A MOV A,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.whoishostingthis.com/resources/assembly-language/>



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.