Introduction to IBM PC Assembly Language

Anika Sayara
anikasayara@outlook.com

Assembly Language Syntax

Case Sensitivity

Assembly language code is generally *not case sensitive*.

Statements

- **Programs consist of *statements, one per line*.**
- **Each statement is *either an instruction or an assembler directive* (which instructs the assembler to perform some specific task, such as allocating memory space for a variable or creating a procedure).**
- **Both instructions and directives *have up to four fields:***
  - ***name***
  - ***operation***
  - ***operand(s)***
  - ***comment***

Statements

The fields must appear in the following order –

Name operation operand(s) comment

Example: START: MOV CX,5 (Instruction)

MAIN PROC (Assembler Directive)

Name field

- **The name field is** ***used for instruction labels, procedure names, and variable names.***
- **Names** ***can be from 1 to 31 characters long*****.**
- **May consist of** ***letters, digits, and special characters ? . @ _ $ %.***
- **Embedded** ***blanks are not allowed.***
- **If a period is used, it** ***must be the first character.***
- **Names** ***may not begin with a digit.***
- **The assembler** ***does not differentiate between upper and lower case in a name.***

Name field:
legal or illegal names?

- **COUNTER1**
- ***TWO WORDS***
- ***@character***
- ***2abc***
- ***SUM_OF_DIGITS***
- ***$1000***
- ***A45.28***
- ***DONE?***
- ***.TEST***
- ***YOU&ME***

Operation Field

- **The operation field *contains a symbolic operation code (opcode).***
- **Assembler translates opcode *translated into machine language opcode***
- **Opcode symbols *describe the operation's function;* for example, *MOV, ADD, SUB.***
- **In an assembler directive, 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*.**
  For example, the PROC pseudo-op is used to create a procedure.

Operand Field

- *Specifies the data* **that are to be acted on by the operation**
- **An instruction** ***may have zero, one or more operands.***

Examples:

```
NOP             ;no operand, does nothing
INC AX          ;one operand, adds 1 to the contents of AX
ADD WORD1, 2    ; two operands, adds value 2 to the contents
                  of memory location WORD1
```

- **In two-operand instruction,** ***first operand is destination, second operand is source.***
- **For an assembler directive,** ***operand field represents more information about the directive***

Comment Field

- **Say something about what the statement does**
- ***Marked by semicolon in the beginning***
- **Assembler ignores anything after semicolon**
- **Optional**
- **Good practice**

Program Data

Program Data

In assembly language, you can express data in:

- ***Binary***
- ***Decimal***
- ***Hexadecimal***
- ***Characters***

Numbers

- **A binary number is written as a *bit string followed by the letter "B" or "b"*; for example, 1010B**
- **A decimal number is a string of decimal digits, *ending with an optional "D" or "d".***
- **A hex number *must begin with a decimal digit and end with the letter "H" or "h";* for example, 0ABCH**

****Any of the preceding numbers may have an optional sign.**

Characters

- **Characters and character strings *must be enclosed in single or double quotes*. For example, "A" or 'hello’.**
- **Characters are translated into their ASCII codes by the assembler, so there is *no difference between using "A" and 41h in a program.***

Variables

Variables

Each variable *has a data type and is assigned a memory address* by the program.

Variables : Data defining pseudo ops

Pseudo-ops	Description	Bytes
DB	Define Byte	1
DW	Define Word	2
DD	Define Double Word	4
DQ	Define Quad Word	8
DT	Define Ten Bytes	10

Byte Variables

Name DB initial_value

- **where the pseudo-op *DB stands for "Define Byte".***
  For example, ALPHA DB 4

- **A question mark *(" ? ") used in place of an initial value sets aside an uninitialized byte*. For example, BYT DB ?**

- **The decimal range of initial values,that can be specified is –**
  - **-128 to 127 for signed interpretation**
  - **0 to 255 for an unsigned interpretation.**

Word Variables

Name DW initial_value

- **where the pseudo-op *DW stands for "Define Word".***
- **A question mark *(" ? ") used in place of an initial value sets aside an uninitialized word*.**
- **The decimal range of initial values,that can be specified is –**
  - **-32768 to 32767 for signed interpretation**
  - **0 to 65535 for an unsigned interpretation.**

Arrays: an array of bytes and words

- **In assembly language, an array is just a sequence of memory bytes or words.**
- **For example, to define a three-byte array called B_ARRAY, whose initial values are 1Oh, 20h, and 30h, we can write,**

```
B_ARRAY    DB    10H,20H,30H
```

The name B_ARRAY is associated with the first of these bytes, B_ARRAY+1 with the second, and B_ARRAY +2 with the third.

- **In the same way, an array of words can be defined.**

```
W_ARRAY    DW    1000,40,29887, 329
```

Arrays: character strings

- **An array of ASCII codes can be initialized with a string of characters.**
  For example, LETTERS DB ‘ABC’
- **It is possible to combine characters and numbers in one definition;**
  For example, MSG DB 'HELLO’, 0AH, 0DH,'$'

Named Constants

Named Contants

Name EQU constant

For example, LF EQU 0Ah

Here, *0Ah is the ASCII for line feed.*

Basic Instructions

MOV

MOV destination, source

- **Transfer data**
  - ***Between registers***
  - ***Between register and a memory location***
  - ***Move a number directly to a register or a memory location***
- **Example: MOV AX, WORD1**

	Before	After
AX	0006	0008
WORD1	0008	0008

Legal combinations of operands for MOV

Destination Operand	Source Operand	Legal
General Register	General Register	YES
General Register	Memory Location	YES
General Register	Segment Register	YES
General Register	Constant	YES
Memory Location	General Register	YES
Memory Location	Memory Location	NO
Memory Location	Segment Register	YES
Memory Location	Constant	YES

XCHG

XCHG destination, source

- **Exchange the contents of**
  - **Two registers**
  - **Register and a memory location**
- **Example: XCHG AH, BL**

Legal combinations of operands for XCHG

Destination Operand	Source Operand	Legal
General Register	General Register	YES
General Register	Memory Location	YES
Memory Location	General Register	YES
Memory Location	Memory Location	NO

ADD

ADD destination, source

- **To add contents of**
  - **Two registers**
  - **A register and a memory location**
  - **A number to a register**
  - **A number to a memory location**
- **Example: ADD WORD1, AX**

	Before	After
AX	01BC	01BC
WORD1	0523	06DF

SUB

SUB destination, source

- **subtract the contents of:**
  - **Two registers**
  - **A register and a memory location**
  - **A number from a register**
  - **A number from a memory location**
- **Example: SUB AX, DX**

	Before	After
AX	0000	FFFF
DX	0001	0001

Legal combinations of operands for ADD and SUB

Destination Operand	Source Operand	Legal
General Register	General Register	YES
General Register	Memory Location	YES
General Register	Constant	YES
Memory Location	General Register	YES
Memory Location	Memory Location	NO
Memory Location	Constant	YES

Direct addition/subtraction between memory locations

Direct addition/subtraction between memory locations is illegal. A way around can be –

```
MOV AL, BYTE2
ADD BYTE1, AL
```

INC

INC destination

- **INC (increment) instruction is used to add 1 to the contents of**
  - **a register**
  - **memory location.**
- **Example: INC WORD1**

	Before	After
WORD1	0002	0003

DEC

DEC destination

- DEC (decrement) instruction is used to subtract 1 from the contents of
  - a register
  - memory location.
- Example: DEC BYTE1

	Before	After
BYTE1	FFFE	FFFD

NEG

NEG destination

- **Used to negate the contents of destination.**
- **Example : NEG BX**

	Before	After
BX	0002	FFFE

Translation of High-level Language to Assembly Language

Translation of High-level Language to Assembly Language

Statement	Translation
B = A	MOV AX, A MOV B, AX

A = 5 - A	MOV AX, 5
	SUB AX, A
	MOV A, AX
	OR
	NEG A
	ADD A, 5

A = B – 2 x A	MOV AX, B
	SUB AX, A
	SUB AX, A
	MOV A, AX

Program Structure

Memory Models

.MODEL memory_model

Determines *the size of data and code* a program can have.

Model	Description
SMALL	code in one segment, data in one segment
MEDIUM	code in more than one segment, data in one segment
COMPACT	code in one segment, data in more than one segment
LARGE	Both code and data in more than one segments. No array larger than 64KB
HUGE	Both code and data in more than one segments. Array may be larger than 64KB

Data Segment

.DATA

- **Contains all *variable definitions and Constant definitions*.**
- ***To declare a data segment, we use the directive .DATA, followed by variable and constant declarations. For example,***

```
.DATA
WORD1     DW 2
W0RD2     DW 5
MSG       DB 'THIS IS A MESSAGE'
MASK      EQU 100100105
```

Stack Segment

.STACK size

- **A block of memory to store stack**
- **Here *size is optional* and specifies the stack area size in bytes**
- ***If size is omitted, 1 KB set aside for stack area***
- **For example: .STACK 100h**

Code Segment

.CODE

- **Contains a program's instructions**
- **Inside a code segment instructions are *organized as procedures***

Basic Program Structure

```
.MODEL SMALL
.STACK 100h
.DATA
        ;data definition go here
.CODE
        MAIN PROC
                ;instructions go here
        MAIN ENDP
                ;other procedures go here
END MAIN
```