Programming in Assembler

8086 addressing modes, instructions, directives and program structure

BEKKOUCHE Mohammed
m.bekkouche@esi-sba.dz

8086 addressing modes

- Instructions and their operands (parameters) are stored in main memory.
- The total size of an instruction depends on the instruction type and on the type of operand.
- Each instruction is coded using an integer number of bytes to facilitate decoding by the processor.

8086 addressing modes

The most general structure of an instruction is:

```
INST Operand1, Operand2
```

- The operation is performed between the two operands and the result is recovered in the left operand.
- Some instructions act on a single operand:

```
INST Operand
```

- Operands can be registers, constants, or contents of memory cells.
- The ways of designating operands constitute the **addressing modes**.

8086 addressing modes

The 8086 microprocessor has seven addressing modes:

1. Register addressing: `INST R, R`
2. Immediate addressing: `INST R, im`
3. Direct addressing: `INST R, [addr]`
4. Indirect addressing: `INST R, Rseg:[Rdisp]`
5. Based addressing: `INST R, Rseg:[Rb+disp]`
6. Indexed addressing: `INST R, Rseg:[Ri+disp]`
7. Indexed-based addressing: `INST R, Rseg:[Rb+Ri+disp]`

Register addressing

Definition

Use of one or two registers.

```
INST R    INST R, R
```

Examples

```
INC AX    MOV AX, BX
```

`MOV AX, BX` means that the operand stored in `BX` is transferred to `AX`.

With register addressing, operations are performed internally: there is no memory exchange, which increases processing speed.

Immediate addressing

Definition

One operand is a constant.

```
INST im     INST R, im
```

Examples

```
JMP 1345h     MOV AX, 134     MOV AX, 'A'
```

`MOV AX,500H` stores the value `500H` immediately in the `AX` register.

Direct addressing

Definition

One of the operands is in memory.

```
INST R,[addr]  INST [addr],R  INST size [addr],im
```

Examples

```
MOV AX,[134]     MOV [134],AX     MOV AX,SS:[134]
```

Normally, the displacement is combined with DS to form the 20-bit effective address. Other segment registers can also be used, for example:

```
MOV AX, ES:[addr]
```

Indirect addressing

Definition

The displacement is given in a register.

```
INST R,Rseg:[Rdisp]
```

Examples

```
MOV AX,[BX]     MOV ES:[BP],AX     MOV WORD PTR [BX],5
```

`MOV AX,[BX]` places in `AX` the contents of the memory cell whose address is held in `BX`.

Based addressing

Definition

The displacement is given in BX or BP, with an optional additional displacement.

```
INST R,Rseg:[Rb+disp]
```

Examples

```
MOV AX,[BX]      MOV AX,[BX+5]
MOV AX,[BP-200]      MOV AX,ES:[BP]
```

`MOV AX,[BX+2]` reads the memory contents addressed by BX+2.

Equivalent forms

```
MOV AX,[BX+2]   MOV AX,[BX]+2   MOV AX,2[BX]
```

Indexed addressing

Definition

The displacement is given in `SI` or `DI`, with an optional additional displacement.

```
INST R,Rseg:[Ri+disp]
```

Examples

```
MOV AX,[SI]      MOV AX,[SI+500]
MOV AX,[DI-8]      MOV AX,ES:[SI+4]
```

Indexed-based addressing

Definition

The displacement is the sum of a base register and an index register, with an optional displacement.

```
INST R,Rseg:[Rb+Ri+disp]
```

Examples

```
MOV AX,[BX+SI]      MOV AX,[BX+DI+5]
MOV AX,[BP+SI-8]
```

Important

Only `BX`, `SI`, `DI`, and `BP` can be used inside square brackets.

The source file (source code)

A program written in assembler includes data and instructions, each written on a line of text.

Data

Declared using directives: special keywords understood by the assembler.

Instructions

Intended for the microprocessor and executed during program execution.

Instruction syntax

The syntax of an instruction is:

```
{Label:} Mnemonic {operand} {; comment}
```

Label field

- Marks a line that can be the target of a jump or branch instruction.
- Maximum length: 31 alphanumeric characters.
- Mnemonics and directive names cannot be used as labels.
- Ends with `:` and cannot start with a number.
- Uppercase and lowercase letters are not distinguished.

Instruction syntax: mnemonic

Mnemonic field

A mnemonic is generally composed of letters and denotes the operation performed by the instruction.

Examples

```
MOV     CMP     LOOP
```

Mnemonics may be written using lowercase or uppercase letters.

Instruction syntax: operand field

Operand field

Operands indicate the data to be processed.

They may be expressed by:

- constant values;
- a memory address where the data is stored;
- a register containing the data;
- a register containing the memory address of the data.

Instruction syntax: comment field

Comment field

A comment starts with ; and ends at the end of the line.

Comments are optional but particularly useful in assembler programming.

- The number of instructions can be high.
- Programs can therefore be delicate and difficult to understand.
- Comments improve program readability for users.

Example of an instruction

Example

```
ET1: MOV AX, 500H ; put the value 500h in the AX register
```

The line contains a label, a mnemonic, an operand, and a comment.

The directives

To program in assembler, directives (or pseudo-instructions) are used in addition to assembly instructions.

Definition

A directive is information supplied by the programmer to the assembler. It is not translated into a machine-language instruction and therefore does not add instruction bytes to the compiled program.

Directives are statements that instruct the assembler.

The directives: syntax

A directive can be written using:

```
{Name} Directive {operand} {; comment}
```

- The operand field depends on the directive.
- The comment field follows the same rules as instruction comments.
- The name field identifies variables or symbols and is optional for some directives.

Numerical constants

Examples of numerical constants:

Constant	Base	Decimal value
`100110b`	Binary	38
`37o`	Octal	31
`4Ch`	Hexadecimal	76

A hexadecimal constant beginning with a letter is preceded by `0`; for example, `C3` is written `0C3h`.

Character string constants

A character string is a sequence of characters enclosed between quote characters.

Examples

```
'hello world'
```

```
"the' tree"
```

A quote character can be included by using the other quote style around the string.

Declaration of a constant

The `EQU` directive associates a value with a symbol that can be used in the program instead of the constant.

```
name EQU constant
```

Example

```
FOUR EQU 4
```

Every occurrence of `FOUR` can then be replaced by the value `4` by the assembler.

Variable declarations

Memory space is used to store constants and variables, such as character strings and numeric values.

Principle

Before using a variable, it must first be declared. Declaring a variable reserves a location in memory to store its value.

This mechanism is conceptually similar to variable declarations in high-level languages, although the allocation is exposed more directly.

Data sizes and declarations

Type	Size	Value range (signed)
Byte	1 byte	-128 to 127
Word	2 bytes	-32768 to 32767
Long	4 bytes	-2^{31} to $2^{31}-1$

The source also notes that an Integer in Pascal reserves 2 bytes.

DB / DW / DD directives

These directives declare variables and cause the assembler to assign an address to each variable.

- Variables are identified by names.
- A name may contain uppercase letters, lowercase letters, and digits, and starts with a letter.
- Maximum length: 31 characters.
- A variable may be assigned an initial value when it is declared.

DB (Define Byte)

Definition

`DB` defines an 8-bit variable and reserves one byte.

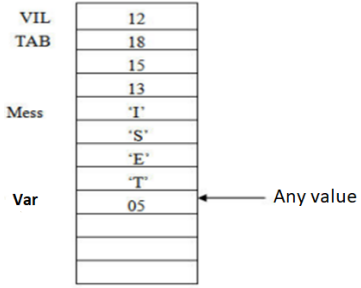
Unsigned values	0 to 255
Signed values	−128 to 127

```
[name] DB constant [, constant ]
```

DB examples

```
Vil DB 12H ; one byte with initial value 12h
Tab DB 18H, 15H, 13H ; a 3-byte table
Mess DB 'ISET' ; array containing ASCII codes
Var DB ? ; 8-bit variable with any initial value
```

The declaration of `Mess` creates an array whose elements are the ASCII codes of the characters.



DW (Define Word)

Definition

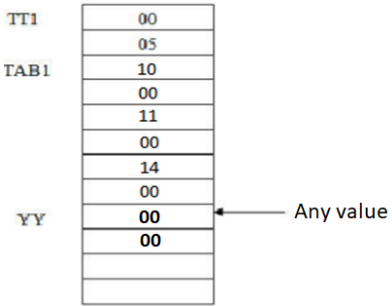
`DW` defines a 16-bit variable and reserves one word (two bytes).

Unsigned values	0 to 65535
Signed values	−32768 to 32767

```
[name] DW constant [, constant ]
```

DW examples

```
TT1 DW 500H ; reserves one word
TAB1 DW 10H,11H,14H ; table of three words = 6 bytes
YY DW ? ; word with any initial value
```



DD and DUP directives

DD (Define Double)

Reserves 4 bytes of memory.

```
ff DD 15500000h
```

DUP

Used when a table of *n* cells must be initialized with the same value.

```
tab DB 100 dup (15) ; 100 bytes initialized to 15
y DW 10 dup (?) ; 10 uninitialized 16-bit words
```

Word PTR and Byte PTR

In some cases memory addressing is ambiguous. For example:

```
MOV [BX], 0
```

The assembler does not know whether the operation concerns one or two consecutive bytes.

Solution

Specify the size explicitly:

```
MOV byte ptr [BX], 0 ; one byte
MOV word ptr [BX], 0 ; one word (2 bytes)
```

ORG, .data and .code

ORG

For example, `ORG 100h` indicates that a program is loaded at `CS:100h` by default for an 8086 `.COM` program.

.data / .code

These directives delimit data declarations and program instructions.

.COM and .EXE programs

.COM

- Cannot use more than one memory segment.
- Program size is limited to 64 KB.
- `ORG 100h` is characteristic of COM files.
- Displacements `000h`–`0FFh` are reserved for the PSP.

.EXE

- Can use multiple segments.
- Size is limited by available memory.
- The PSP has its own segment.
- The program starts at displacement `0h` in the code segment.

For segmented programs, `SEGMENT` and `ENDS` define segments, while `ASSUME` indicates the data and code segments used by the assembler.