The assembly program is divided into directives and instructions.

- Directive \( \rightarrow \) directs the assembler or compiler during the translation process where to put the machine code.

  \[
  \text{Org, End}
  \]

- Instruction: Statements have machine code and is executed by the microprocessor.

  \[
  \text{Machine language}
  \]

* Instruction are written in the program memory and one instruction is composed of 14 bits which are called a word.
* Each instruction is divided into opcode & operand.

14 bit instruction

| Opcode | Operand |

Op-code \(\rightarrow\) operational code. It is the code to distinguish an instruction is written.

* The instructions have three forms:

1. Byte oriented operations.
2. Bit oriented operations.
3. Literal & control operations.

(1) Byte oriented operations so

<table>
<thead>
<tr>
<th>13</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Opcode</td>
<td>d</td>
<td>f</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(2) \(f\) \(\rightarrow\) file register \(\rightarrow\) it is the address of either GPR or SFR.

* Note: GPR & SFR are 8 bit registers, but in the instruction, it has only 7 bits, the last bit is dropped.

Diagram: (Diagram of data memory and registers showing GPR and SFR locations.)
Example: 

port A → 05h "address"
trisA → 85h "address"

These bits are dropped

These bits only appear in the instruction set.

Status & R6

⑬ d → destination

0 → work register
1 → file register.

Examples 8

① ADDWF portA , W

00 0111

\[ \text{opcode} \]

**d** ff ff ff ff

\[ \text{operand} \]

Address of port A 05h → d = 0

Instruction will be

\[ \begin{align*}
00 & \quad 0111 \\
3 & \quad 0000 \\
5 & \quad 0101
\end{align*} \]
2) Bit oriented operations

\[ \begin{array}{c|c|c|c|c}
    & 7 & 6 & 5 & 0 \\
   \hline
   \text{opcode} & b & f \\
\end{array} \]

- \( f \rightarrow \) file register.
- \( b \rightarrow \) Bit address

\( \text{3 bits } f b \text{ of } SFR \text{ or GPR registers} \)

\[ \begin{array}{c|c}
    000 & 0 \\
    111 & 7 \\
\end{array} \]

Example

1. BST Port B, 2

- \( f \rightarrow \) Port B
- \( b \rightarrow \) address \( 06h \)

\[ \begin{array}{c}
    01 \ 01 \ 01 \ 0000 \ 0110 \\
\end{array} \]

2. BST TrisB, 2

- \( f \rightarrow \) Tris B
- \( b \rightarrow \) address \( 86h \)

\[ \begin{array}{c}
    01 \ 01 \ 01 \ 0000 \ 0110 \\
\end{array} \]
* Note *
BSF PortB, 2, BSF TrisB, 2
have same machine code

3) Literal & Control Operations
It has 2 forms:

1) \[ \begin{array}{c}
13 \\
8 \\
7 \\
0 \\
\end{array} \]
   \begin{array}{c}
\text{op code} \\
k \\
\end{array} \\
2) \[ \begin{array}{c}
13 \\
11 \\
10 \\
0 \\
\end{array} \]
   \begin{array}{c}
\text{op code} \\
k \\
\end{array}

* In form 1, k is a constant and since all
  registers are 8 bit registers, maximum value that
  can be stored in the register is 255 (FFh)

* In form 2, k is address of program memory
  It is used with Call & Goto instructions.
  It is 11 bit because max. range 0→ 2047 (7FFh)

Examples:

1) XORLW 15h
   \[ \begin{array}{c}
11 \\
1010 \\
kkkk \\
kkkk \\
\end{array} \]
   \[ \begin{array}{c}
11 \\
1010 \\
0001 \\
0101 \\
3 \\
5 \\
1 \\
5 \\
\end{array} \]
* write machine code for the following program:

LIST P = PIC16F84A

# Include < PIC16F84A.Inc >

CONFIG XT Osc & PORTE ON & CP OFF & WDT OFF

(Org) 0X000 \rightarrow \text{directive}

Goto Start

(Org) 0X004 \rightarrow \text{directive}

RETFIE

Start

BSF Status, RPo

BCF TrisA, 0

BSF TrisB, 0

BCF Status, RPo

Loop

MOVF PORTB, W

MOVWF PORTA

Goto Loop

End

Program memory

| 0000 | reset code |
| 0004 | interrupt code |
| 0005 | Start |
1. Goto Start
   10 1kk kkk kkk kkk
   2 8 0 5

2. RETFIE
   00 0000 0000 1001
   0 0 0 9

3. BSF Status, RPo
   01 0110 1000 0011
   1 6 8 3

4. BCF TrisA,0
   01 0000 0000 0101
   1 0 0 5

5. BSF TrisB,0
   01 0100 0000 0110
   1 4 0 6

6. BCF Status, RPo
   01 0010 1000 0011
   1 2 8 3

7. MOVF portB,0
   00 1000 0000 0110
   0 8 0 6

8. MOVWF portA
   00 0000 1000 0011
   0 6 8 5

9. Goto loop
   10 1000 0000 1001
   2 8 0 9
Configuration

2007h
2 n 7
3 n 7
6 4 n 7
Assembler

13 bit

0 bit

1

1 0 0 0 1

II IIII IIII 0001

3 F F F 1

hexa file

Contains the machine language that deals with PIC.
The form of the hexa file is:

<table>
<thead>
<tr>
<th>Record mark</th>
<th>Record length</th>
<th>Offset</th>
<th>Record type</th>
<th>Info. or data</th>
<th>Check Sum</th>
</tr>
</thead>
</table>

* Record mark => It is a colon ':' character. It has ASCII (3A).

* Record length => number of bytes of Information or data

example: 02 => 2 bytes => byte: 8 bits

2 bytes => 2 * 8 bits = 16 bits

4 hexadecimal digits.
8 ⇒ 8 byte ⇒ 8 * 8 bits = 64 bits
\[ \frac{64}{4} = 16 \] hexadecimal digits

* Offset: نستخدم الاعداد كيت اللمة
word لا يفوت byte ليس ل

example 0008

* Program memory

<table>
<thead>
<tr>
<th>Offset</th>
<th>6 bit</th>
<th>8 bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0001</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>0002</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>0003</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>0004</td>
<td>9</td>
<td>8</td>
</tr>
</tbody>
</table>

* Record type 8

explain the remaining information within the record.

example 00 ⇒ data 301 ⇒ end of file

* Information or Data:

  * hexadecimal digits
  * record type

* Check sum ⇒ error detection