*Index*

|  |  |
| --- | --- |
| Topic | Page No. |
| Index | 1 |
| How to use ? | 2 |
| Instruction Format summary | 3 |
| General Architecture | 4 |
| Assembly instruction to binary encoding | 5 |
| Operation codes | 6 |
| Error Handling | 7 |
| Instruction Set Examples | 8 |
| Workflow of the Assembler | 9 |

*How to use?*

Easy !

Run the ‘assemblerMainCode.py’ after placing the ‘sourceCode.txt’ containing the source Assembly program in the same directory. Then check the new files created for the output.

Important: ‘assemblerMainCode.py’ and ‘sourceCode.txt’ should be in the same directory.

New files created as outputs in the same directory are:

1. ‘symbolTable.txt’
2. ‘literalTable.txt’
3. ‘labelTable.txt’
4. ‘opCodeTable.txt’
5. ‘machineCode.txt’

Please ensure to follow the Instruction formats for no errors. However, any mistakes in the source code, will be printed on the terminal without hindering the code conversion.

*Instruction Formats*

**Comments**

Comments are not read by the assembler. They should be placed only in the starting of each assembly instruction line starting with ‘@’. Also the value of Location Counter is not incremented.

Example:

@ This is a comment !

@ Manas and Prachi made this awesome project !!!

**Operation Codes**

Type 1: no operands required

[opcode]

Example:

CLA

STP

Type 2: requires 1 operand

[opcode] [operand]

Example:

MUL R1

INP R7

Note: The operand can be a register, an immediate value, symbol or a literal.

In case of branch statements, it should be a label.

Example:

BRP loop

BRZ here

**Pseudo Opcodes**

Not converted into machine Code.

START: It indicates beginning of source code program. It is followed by the value of the Location Counter(LC). If no value is given, then Location Counter is set to default as 0.

Example:

START // LC = 0

START 21 // LC = 21

START 0 // LC = 0

LTORG: It indicates Literal origin. It is followed by initialization of a literal in order of its appearance in the source code.

Note that the declaration of a literal begins with an asterisk ‘\*’.

There are two ways to declare a literal:

Type 1: Decimal form

In this case the value is preceded by a ‘d’.\*\*

Also if the value is not preceded by a character, then it is by default taken as a decimal.

Example:

LTORG

\* =’10’ //decimal value of =’10’ is 10

\* =d’10’ //decimal value of =d’10’ is 10

Type 2: Hexadecimal form

In this case the value is preceded by ‘h’.\*\*

Example:  
 LTORG

\* =h’a9’ //decimal value of =h’a9’ is 169

**\*\* characters used for initialization of decimal or hexadecimal as well as characters used in hexadecimal numbers are case sensitive and should be lowercase only.**

DC: Indicates declaration of a symbol/variable and the value is taken in decimal form.

Example:

Z DC ‘1’ //decimal value of Z is 1

DS: Indicates declaration of a symbol/variable and the value is taken in hexadecimal form.

Example:

A DS ‘f9’ //decimal value of A is 24

Immediate values:

Send the value of the operand itself.

Example:

MUL #45 //multiply by 45

SUB #3 //subtract 3

Immediate values can be between 0 and 63 only. (Limits Inclusive)

Note: In case of any empty line in the source program, the location counter is not incremented.

*General Architecture*

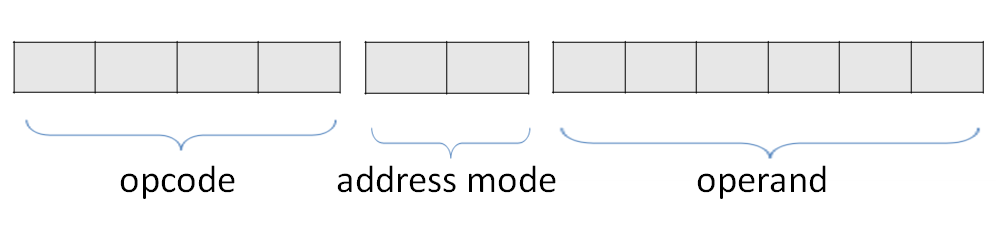
The assembler is built for a 12 bit accumulator architecture consisting of 16 registers and 64 memory locations.

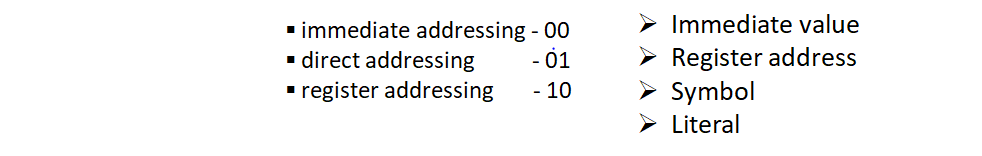
The 16 registers are named as follows-

'R0', 'R1', 'R2', 'R3', 'R4', 'R5', 'R6', 'R7', 'R8', 'R9', 'R10', 'R11', 'R12', 'R13', 'R14', 'R15'.

Each memory location has length 1 word where 1 word = 12 bits.

*Assembly instruction to Binary Encoding*





Instruction size is constant, i.e. 12 bits for every Instruction.

For type 1 operation codes - ‘CLA’ and ‘STP’, instruction is 12 bits long as well (no exceptions).

For ‘CLA’:

The first 4 bits denote the opcode of CLA - 0000.

The next 2 bits denote the immediate addressing mode (to denote that the value of Accumulator Register has been set to 0).

The next 6 bits denote the value of Accumulator Register which is set to 0.

Thus, the machine code becomes 0000 00 000000.

For ‘STP’:

The first 4 bits denote the opcode of STP - 1100.

To keep the Instruction length constant, the next 2 + 6 bits are stored as a string of 0s.

Thus, the machine code becomes 1100 00 000000.

*List of all Operation Codes (OpCodes)*

|  |  |  |
| --- | --- | --- |
| Operation Code | Instruction | Mnemonic |
| 0000 | Clear accumulator | CLA |
| 0001 | Load into accumulator from address | LAC |
| 0010 | Store accumulator contents into address | SAC |
| 0011 | Add address contents to accumulator contents | ADD |
| 0100 | Subtract address contents from accumulator contents | SUB |
| 0101 | Branch to address if accumulator contains zero | BRZ |
| 0110 | Branch to address if accumulator contains negative value | BRN |
| 0111 | Branch to address if accumulator contains positive value | BRP |
| 1000 | Read from terminal and put in address | INP |
| 1001 | Display value in address on terminal | DSP |
| 1010 | Multiply accumulator and address contents | MUL |
| 1011 | Divide accumulator contents by address content. Quotient in R1 and remainder in R2 | DIV |
| 1100 | Stop execution | STP |

*Error Handling*

Note: As soon as the assembler encounters an error, it will report the error on the python IDE along with the line number and will leave a blank line in the machine code output.

Important: When an error occurs, location counter is decremented. So in total, errors will not increment Location counter.

Zero Division Error:

When programmer sends 0 to the accumulator as immediate value with DIV opcode.

Declaration Error:

It is of two types-

Type 1:

Symbol/literal/label has been declared multiple times.

Type 2:

Symbol/literal/label has not been declared but has been used in the code.

Declarative Statement Error:

If a pseudo opcode like ‘LTORG’ or ‘START’ or an opcode such as ‘STP’ has been used multiple times in the code.

Important Note:

In case of LTORG, error is reported only when LTORG is again used just after initial use of LTORG, rendering the second LTORG redundant.

Exceeded Memory Limit Error:

6 bits have been allocated to store memory address in the Location Counter(LC).

For every instruction, the LC increments by 1 since the size of every instruction is constant ie 12 bits.

Therefore, the maximum value that the LC can store is 63.

Thus if the value of LC exceeds 63, an exceeded memory limit error pops up.

Word Limit Exceeded Error:

Since 6 bits have been allocated to store the operand, so if the value of symbols

*Instruction set Examples*

*Workflow of the Assembler*

Project by –

Prachi Goyal

Manas Gupta