CS 110 : Assembly Language and Programming
Semester - 4

INTRODUCTION TO ASSEMBLY LANGUAGES

Sanskriti
Paridhi Sharma
Geetika Sharma
Shreya Kashyap

Contents

- Overview
- Programming Languages & types
- Compiler & Machine code
- Limitation of machine languages
- Assembly languages
- Assembly v/s Machine Code
- General format
- Instructions

- Assembler
- Opcode
- Labels
- Assembler Directives
- Translation to machine code
- Examples of assembly programs
- Conclusion
- Reference

Overview

Assembly Language is a low-level programming language. In computers, there is an assembler that helps in converting the assembly code into machine code executable. Assembly language is designed to understand the instruction and provide it to machine language for further processing. It mainly depends on the architecture of the system, whether it is the operating system or computer architecture.

Assembly Language mainly consists of mnemonic processor instructions or data and other statements or instructions. Assembly Language helps in fine-tuning the program.

What is Language?

Language is a mode of communication that is used to share ideas, opinions with each other. For example, if we want to teach someone, we need a language that is understandable by both communicators.

What is a Programming Language?

A programming language is a computer language that is used by programmers (developers) to communicate with computers. It is a set of instructions written in any specific language (C, C++, Java, Python) to perform a specific task.

Types of Programming Languages

Machine Language

Considered a native language as it can be directly understood by a (CPU). It's not so easy to understand, as the language uses the binary system in which the commands are written in 1 and 0 form which is not easy to interpret.

Assembly Language

Low-level programming language in which there is a very strong correspondence between the instructions in the language and the architecture's machine code instructions.
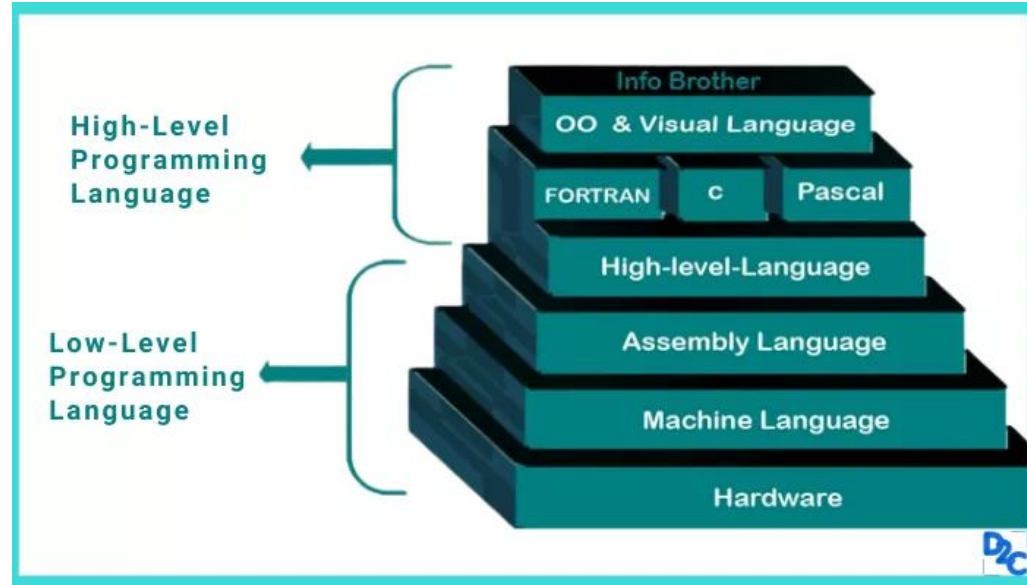
High Level Language

Easy to understand and the code can be written easily as the programs written are user-friendly. The other advantage of code written is independent of a computer system The high-level of language uses the concept of abstraction

High-Level Language

Most programming now-a-days is done using so-called "high-level" languages (such as

FORTRAN, BASIC, COBOL, PASCAL, C, C++, JAVA, SCHEME, Lisp, ADA, etc.)

High-Level Language

"High-level language" refers to the higher level of abstraction from machine language. Rather than dealing with registers, memory addresses, and call stacks, high-level languages deal with variables, arrays, objects, complex arithmetic or boolean expressions, subroutines and functions, loops, threads, locks, and other abstract computer science concepts.

with strong abstraction from the details of the computer. it may use natural language elements, be easier to use, the amount of abstraction provided defines how "high-level" a programming language is
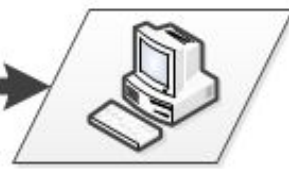
Programmer writes code in high level language → Code is turned into machine instructions by compiler or interpreter → Computer runs machine instructions

```csharp
namespace TriangleCS
{
    class Program
    {
        static double triangle_area(double _base, double height)
        {
            double area = (1.0 / 2.0) * _base * height;
            return area;
        }
        static void Main(string[] args)
        {
            double a1 =  triangle_area(10.0, 2.0);
            System.Console.WriteLine(a1.ToString());
        }
    }
}
```
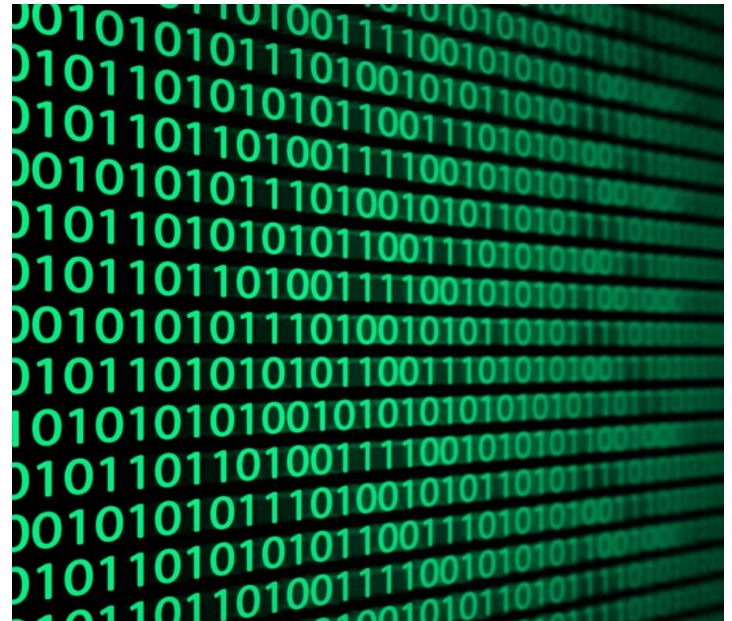
Compiler

A compiler is a special program that processes statements written in a particular programming language and turns them into machine language or "code" that a computer's processor uses. Typically, a programmer writes language statements in a language such as Pascal or C

hello_world.c

Compiler

hello_world.o

Low-level programming language

Low-level language is machine-dependent (0s and 1s) programming language. The processor runs low- level programs directly without the need of a compiler or interpreter, so the programs written in low-level language can be run very fast

Low-level language is further divided

 into two parts -

i. Machine Language

ii. Assembly Language

Machine code

Machine code, also known as machine language, is the elemental language of computers. It is read by the computer's central processing unit (CPU), is composed of digital binary numbers and looks like a very long sequence of zeros and ones.

Each CPU has its own specific machine language. The processor reads and handles instructions, which tell the CPU to perform a simple task. Instructions are comprised of a certain number of bits
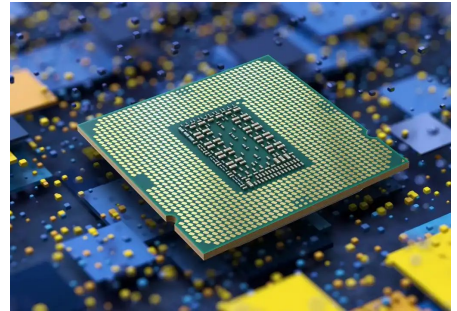
If instructions for a particular processor are 8 bits, for example, the first 4 bits part (the opcode) tells the computer what to do and the second 4 bits (the operand) tells the computer what data to use.

```
0100 1110      ->    ADD 14   ->   AC <- AC + 14
```

```
Example of 32 bit instructions

01001000 00110101 01101100 01101100
```

The 32-bit instruction may have six fields: cond, op, funct, Rn, Rd, and Src2. The operation the instruction performs is encoded in the field: op (also called the opcode or operation code) and funct or function code; the cond field encodes conditional execution based on flags
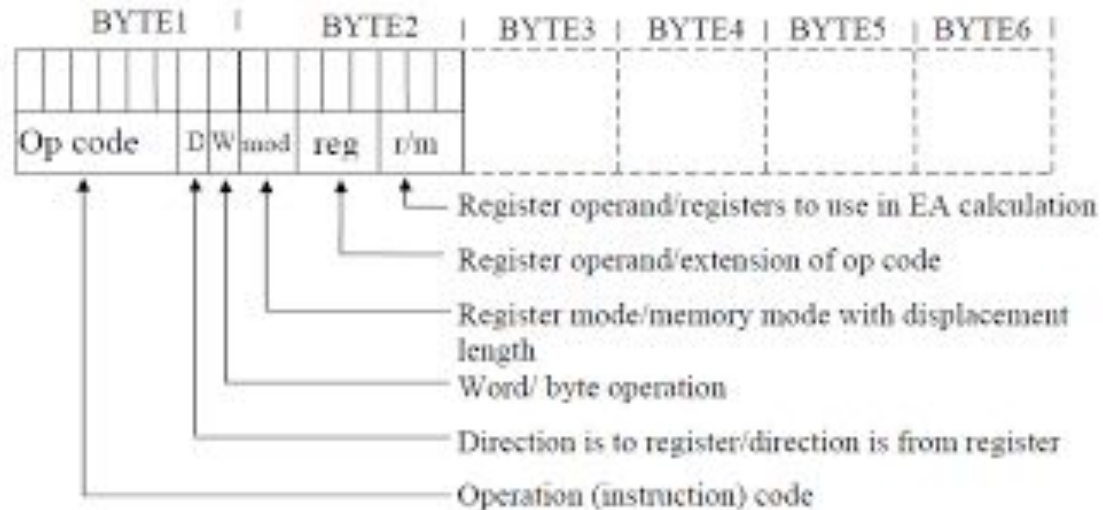
Operation Code		Operand
Machine instruction		
1 0 1 1 1 0	**0 1**	1 1 1 1 1 1 1 1

© teach-ict.com

Address Mode

Depending upon the processor, a computer's instruction sets may all be the same length, or they may vary, depending upon the specific instruction

The architecture of the particular processor determines how instructions are patterned

BYTE1 | BYTE2 | BYTE3 | BYTE4 | BYTE5 | BYTE6 |

| Op code | D | W | mod | reg | r/m |

— Register operand/registers to use in EA calculation

— Register operand/extension of op code

— Register mode/memory mode with displacement length

— Word/ byte operation

— Direction is to register/direction is from register
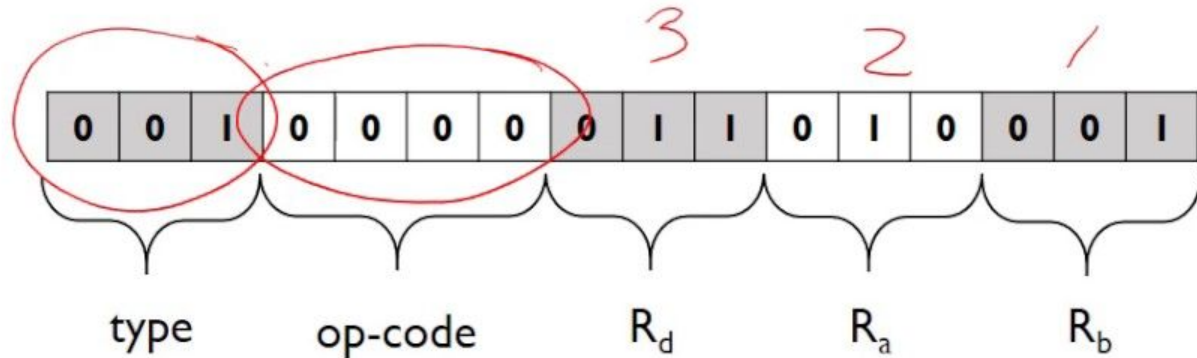
— Operation (instruction) code

Instruction format

- The fictitious example below shows a 16-bit instruction
- It has a **3-address format**

$R_2 + R_1$

| type | op-code | R_d | R_a | R_b |

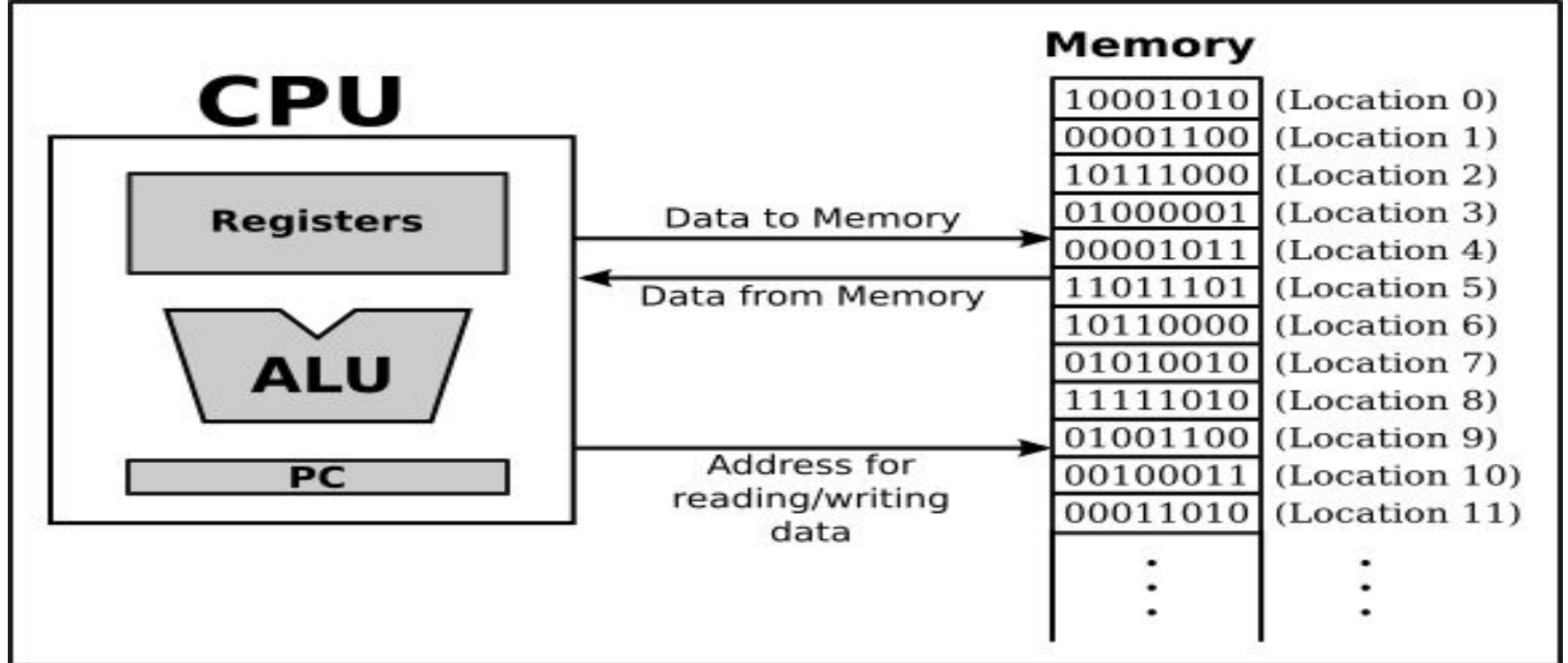Data operation $R_d = R_a$ *operation* R_b ADD   R3, R2, R1
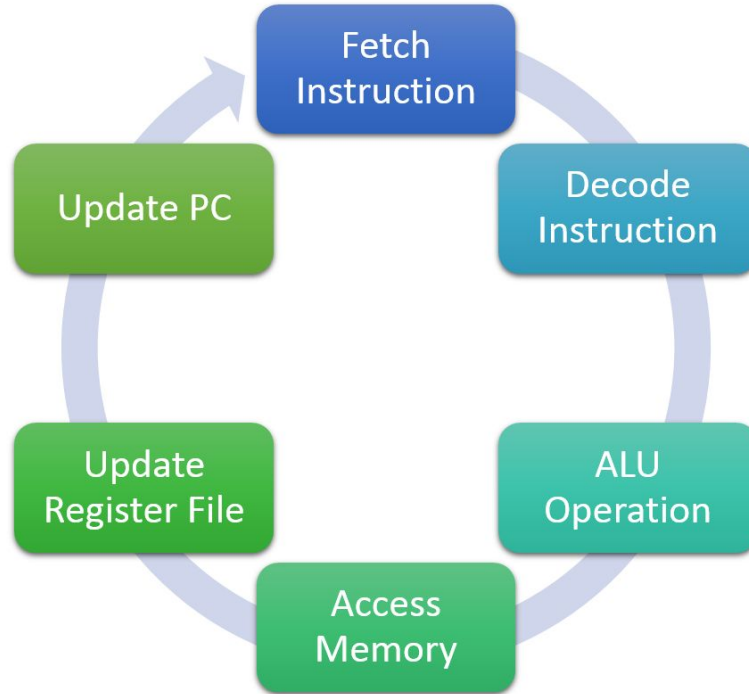
How Does CPU execute these?

The Fetch and Execute Cycle

CPU Instruction Execution Steps

Limitation of Machine language

- It is machine dependent i.e. it differs from computer to computer.
- It is difficult to program and write.
- It is prone to errors
- It is difficult to modify.

Solution ?
We move on to the Assembly Language

Assembly language

It is a low level programming language that allows a user to write a program using alphanumeric mnemonic of instructions. It requires a translator as assembler to convert language into machine language so that it can be understood by the computer.

It is easier to remember and write than machine language.

Advantage(s)

Advantage(s) of using assembly language rather than machine language is/are:

A. It is mnemonic and easy to read

B. Addresses any symbolic, not absolute

C. Introduction of data to program is easier

D. easy to locate and correct errors.

Assembly vs Machine Code

```
Machine code bytes        Assembly language statements

                          foo:
B8 22 11 00 FF            movl $0xFF001122, %eax
01 CA                     addl %ecx, %edx
31 F6                     xorl %esi, %esi
53                        pushl %ebx
8B 5C 24 04               movl 4(%esp), %ebx
8D 34 48                  leal (%eax,%ecx,2), %esi
39 C3                     cmpl %eax, %ebx
72 EB                     jnae foo
C3                        retl


Instruction stream

B8 22 11 00 FF 01 CA 31 F6 53 8B 5C 24
04 8D 34 48 39 C3 72 EB C3
```

Key Ideas

There are two key ideas:

-- **mnemonic opcodes**: we use abbreviations of English language words to denote operations

-- **symbolic addresses**: we invent "meaningful" names for memory storage locations we need

These make machine-language understandable to humans – if they know their machine's design

Let's see our example-program, rewritten using actual "assembly language" for Intel's Pentium

32-bit Instructions

- Instructions are represented in memory by a series of "opcode bytes."

- A variance in instruction size means that disassembly is position specific.

- Most instructions take zero, one, or two arguments:

  instruction destination, source

For example: add eax, ebx

is equivalent to the expression eax = eax + ebx

Assembly Programming

- Assembly Language instruction consist of four fields

[label:]   mnemonic [operands] [;comment]

- Labels
  - See rules

- mnemonic, operands
  - MOV AX, 6764

- comment
  - ; this is a sample program

General Format

| mnemonic | operand(s) | ;comments |

| MOV | destination,source | ;copy source operand to destination |

Example:
MOV   DX,CX

Example 2:

MOV   CL,55H
MOV   DL,CL
MOV   AH,DL
MOV   AL,AH
MOV   BH,CL
MOV   CH,BH

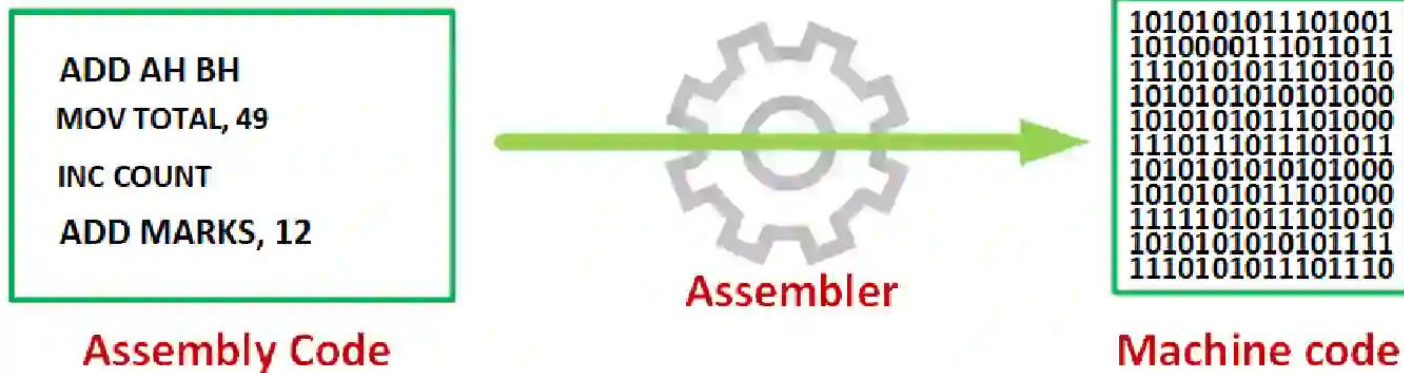| AH	AL
BH	BL
CH	CL
DH	DL

Instructions

- **Each instruction does something relatively simple.**
  - move some bits around
  - treat some bits as base 2 numbers and apply arithmetic operations.
  - send/read some bits to/from I/O devices.
 - select the group of bits that will make up the next instruction

Assembler

It is a computer programme which converts or translate assembly language into machine language. It assembles the machine language program in the main memory of the computer and makes it ready for execution.


```
ADD AH BH
MOV TOTAL, 49
INC COUNT
ADD MARKS, 12
```

Assembly Code → **Assembler** →

```
1010101011101001
1010000111011011
1110101011101010
1010101010101000
1010101011101000
1110111011101011
1010101010101000
1010101011101000
1111110101110100
1010101010101111
1110101011101110
```

Machine code

Basic Assembly Program

An assembly program can be divided into

three sections –

- The data section,
- The stack section, and
- The text section.
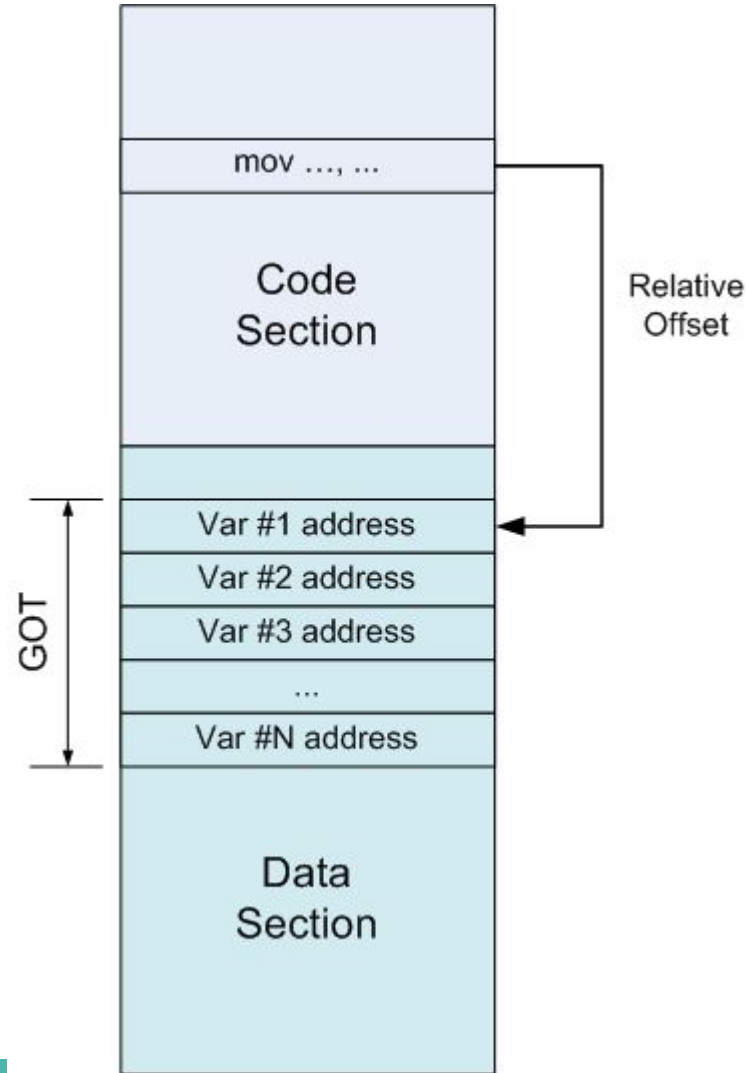
Opcode

In computing, an opcode (abbreviated from operation code, also known as instruction machine code, instruction code, instruction syllable, instruction parcel or opstring) is the portion of a machine language instruction that specifies the operation to be performed.

DATA TRANSFER	ARITHMETIC	LOGICAL	BOOLEAN	PROGRAM BRANCHING
MOV	ADD	ANL	CLR	LJMP
MOVC	ADDC	ORL	SETB	AJMP
MOVX	SUBB	XRL	MOV	SJMP
PUSH	INC	CLR	JC	JZ
POP	DEC	CPL	JNC	JNZ
XCH	MUL	RL	JB	CJNE
XCHD	DIV	RLC	JNB	DJNZ
	DA A	RR	JBC	NOP
		RRC	ANL	LCALL
		SWAP	ORL	ACALL
			CPL	RET
				RETI
				JMP

8051 MICROCONTROLLER INSTRUCTION SET

Functional Group	Example Mnemonics
Move Instructions	MOV
Math Instructions	MUL DIV ADD SUB
Logic Instructions	AND IOR XOR NEG
Rotate/Shift Instructions	ASR LSR SL
Bit Instructions	BSET BCLR BTG BTST
Compare/Skip/Branch	BTSC BTSS CPBEQ CPBGT
Flow Control Instructions	BRA CALL RCALL REPEAT
Shadow/Stack Instructions	LNK POP PUSH ULNK
Control Instructions	NOP CLRWDT PWRSAV RESET
DSP Instructions	MAC LAC SAC SFTAC

Labels

A label can be placed at the beginning of a statement. During assembly, the label is assigned the current value of the active location counter and serves as an instruction operand.

Acts as a place marker

There are two types of lables: **symbolic and numeric**

Assembler Directives

Assembler directives are the instructions used by the assembler at the time of assembling a source program.

More specifically, we can say, assembler directives are the commands or instructions that control the operation of the assemble
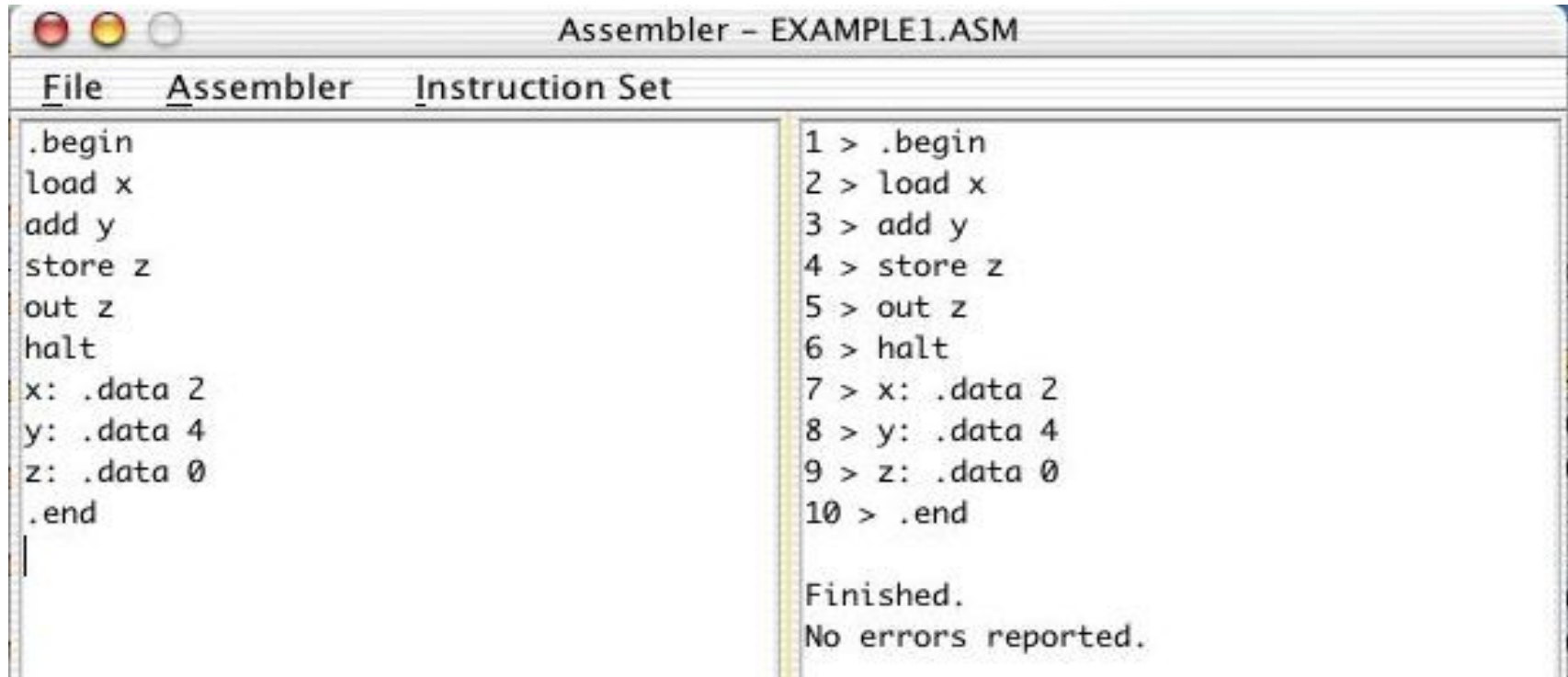
Comments

In computer programming, a comment is a programmer-readable explanation or annotation in the source code of a computer program. They are added with the purpose of making the source code easier for humans to understand, and are generally ignored by compilers and interpreters

```
 1  public class CommentsInJava {
 2      /** Javadoc comment...appropriate for documenting a class or method for javadoc utility.
 3          Example: The main method is run automatically by the Java Virtual Machine */
 4      public static void main (String[] args) {
 5          int i=100;
 6          // Single line comment...appropriate for documenting a statement.
 7          // Example: The following statement prints a string to the console:
 8          System.out.println("This is the first statement after the comment!");
 9
10          /* Multi-line comment...appropriate for commenting code that is
11             not needed currently but may be
12             necessary in the future.
13          Example: System.out.println("This is the integer variable I created: " + i);
14          System.out.println("This is the last statement in the block!");
15          */
16      }
17  }
```

Assembly translates to Machine Code

Machine code	Assembly code	Description
001 1 000010	LOAD #2	Load the value 2 into the Accumulator
010 0 001101	STORE 13	Store the value of the Accumulator in memory location 13
001 1 000101	LOAD #5	Load the value 5 into the Accumulator
010 0 001110	STORE 14	Store the value of the Accumulator in memory location 14
001 0 001101	LOAD 13	Load the value of memory location 13 into the Accumulator
011 0 001110	ADD 14	Add the value of memory location 14 to the Accumulator
010 0 001111	STORE 15	Store the value of the Accumulator in memory location 15
111 0 000000	HALT	Stop execution

Example assembly program

Interpreting an Assembly Instruction

To interpret machine language, one must decipher the fields of each instruction word. Suppose there is a 32 bit instruction, different instructions use different formats, but all formats start with a 6-bit opcode field. Thus, the best place to begin is to look at the opcode. If it is 0, the instruction is R-type; otherwise it is I-type or J-type.
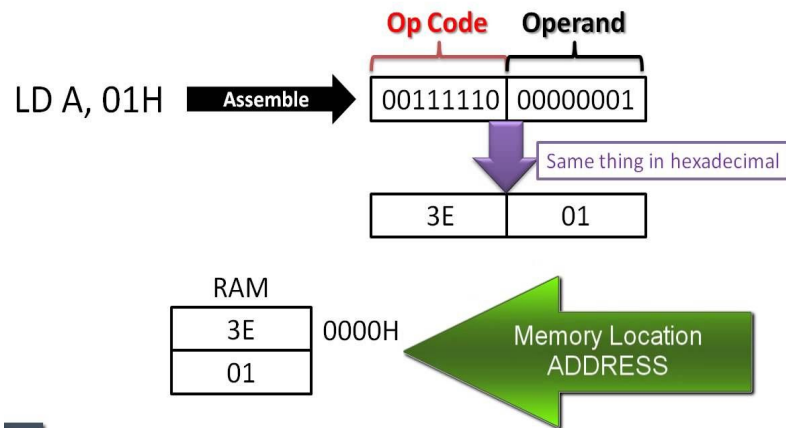
Translating Memory Instructions into Machine Language

Example, Translate the following assembly language statement into machine language.

STR R11, [R5], #-26

STR is a memory instruction, so it has an op of 012. According to Table 6.11, L = 0 and B = 0 for STR. The instruction uses post-indexing, so according to Table 6.10, P = 0 and W = 0. The immediate offset is subtracted from the base, so I = 1 and U = 0. shows each field and the machine code. Hence, the machine language instruction is 0xE405B01A.

When do we use Assembly

Assembly-level programming is still written, and fairly often, as well.

The most traditional groups of people who write assembly are compiler and OS programmers, but it's also used in a wide swath of other applications

It's used in:

- *Certain parts of video games*
- *Certain parts of virtual machines*
- *Most automobile software*
- *A lot of embedded systems*

Disadvantages of Assembly Language

Disadvantages Assembly Language:

1. One of the major disadvantages is that assembly language is machine dependent. A program written for one computer might not run in other computers with different hardware configuration.

2. Long programs written in such languages cannot be executed on small sized computers.

3. It takes lot of time to code or write the program, as it is more complex in nature.

Conclusion : Assembly Language

An assembly language is a type of programming language that translates somewhat english like mnemonics and addresses directly into machine language.

It is a necessary bridge between software programs and their underlying hardware platforms.

Today, assemble languages are rarely written directly, although they are still used in some niche applications such as when performance requirements are particularly high.

Reference

1. https://www.codecademy.com/learn/computer-architecture/modules/assembly-language

2. https://www.tutorialspoint.com/assembly_programming/index.htm

3. https://github.com/Sarvesh-CSE/Computer_Organization

4. https://www.massey.ac.nz/~mjjohnso/notes/59102/notes/l9.html