//Definition:

Why Assembly language?

Time efficient

Less Memory Consumption

Easy hardware manipulation

Assemblers:

Programs that convert programs written in low-level language to machine code (0 and 1)

Eg.: nasm, tasm, masm for x86 ISA

On Linux:

Gcc -S <filename.c>

Filename.s is its assembly representation

Then type: gcc filename.s : will generate a binary ‘a.out’

Von-Neuman Architecture: Single memory for data and instructions

View of Registers:

Registers: named storage locations

In ARM: r0, r1, r2, … r15

In x86: eax, ebx, ecx, edx, esi, edi

Registers with special functions:

Stack pointer, program counter, return address

Memory:

One large array of bytes

Each location has an address

The address of the first location is 0 and increases by 1 for each subsequent location.

The program is stored in a part of a memory

The program counter contains the address of the current instruction.

Storage of Data in Memory:

Data types: char (1 byte), short (2 bytes) int (4 bytes) long int (8 bytes)

Little endian vs Big Endian:

0x87654321: MSB=8, LSB=1

Big endian: Starts with MSB

|  |  |  |  |
| --- | --- | --- | --- |
| 87 | 65 | 43 | 21 |
| 0 | 1 | 2 | 3 |

Little endian: Starts with LSB

|  |  |  |  |
| --- | --- | --- | --- |
| 12 | 34 | 56 | 78 |
| 0 | 1 | 2 | 3 |

At one location, we are storing one byte. So for eg, for an int, we need 4 locations.

2-dimensional arrays:

Row major:

1st row as 1d array, and then so on…

Column major:

1st column as 1d array, and then so on…

Structure of a statement:

Instruction + operand 1 + (operand 2 + … operand n)

Instruction:

Operand:

Eg: sub r3, r1, r2

Subtract the contents of r2 from r1 and save the result in r3

Three operands: r3 r1 and r2

Types of instructions:

Data Processing Instructions:

Data Transfer Instructions

…

…

If an instruction takes n operands, then it is said to be in the n-address format

Addressing Mode: The method of specifying and accessing an operand in an assembly statement is known as the addressing mode.