

The Islamic University of Gaza

Computer Architecture Lab - 2023

LAB (1) || Introduction to Assembly Language

Eng: Amal I. Mahfouz , Eng: Hassan Ghazy

objective

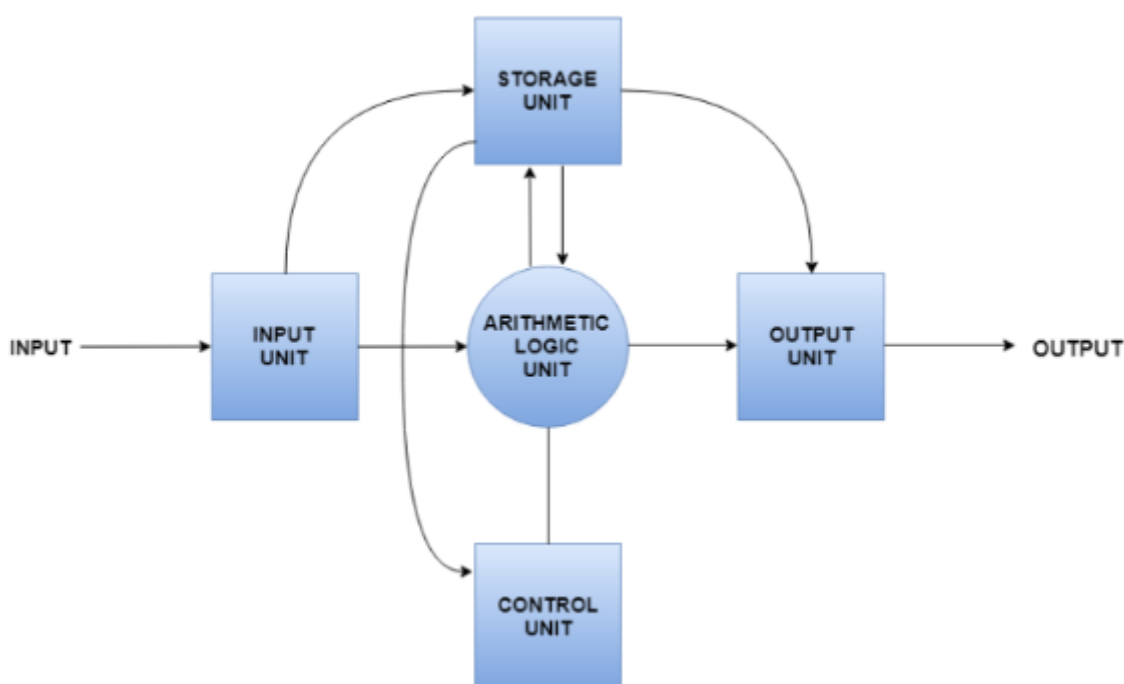
1. what is computer system architecture?
 2. what is classify of Computer Languages?
 3. write a simple program in assembly language.
 4. convert high level code to assembler code.
-

Computer System Architecture

A computer system is basically a machine that simplifies complicated tasks. It should maximize performance and reduce costs as well as power consumption. The different components in the Computer System Architecture are:

- Input Unit
- Output Unit
- Storage Unit
- Arithmetic Logic Unit
- Control Unit ,etc.

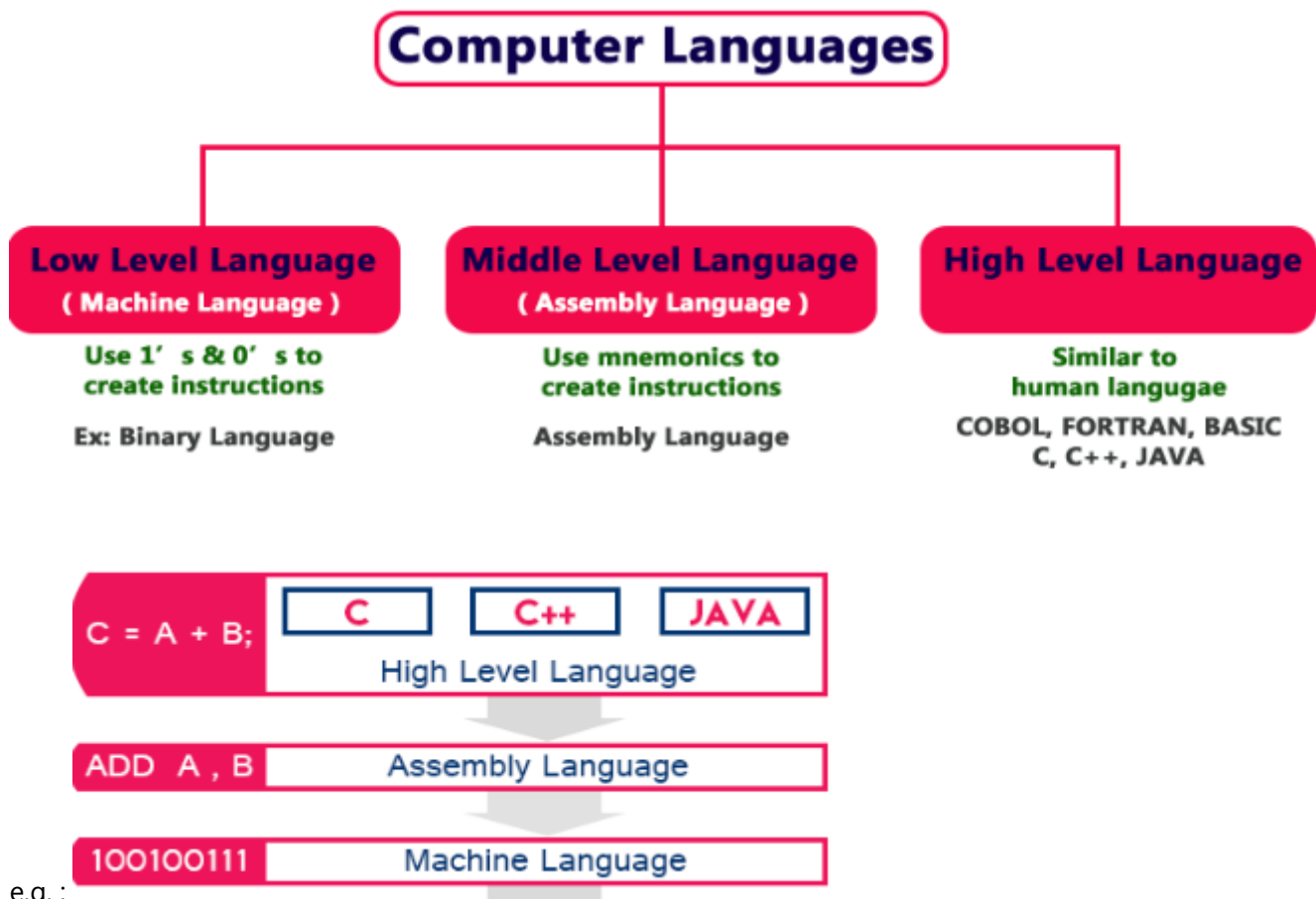
A diagram that shows the flow of data between these units is as follows :



The input data travels from input unit to ALU. Similarly, the computed data travels from ALU to output unit. The data constantly moves from storage unit to ALU and back again. This is because stored data is computed on before being stored again. The control unit controls all the other units as well as their data

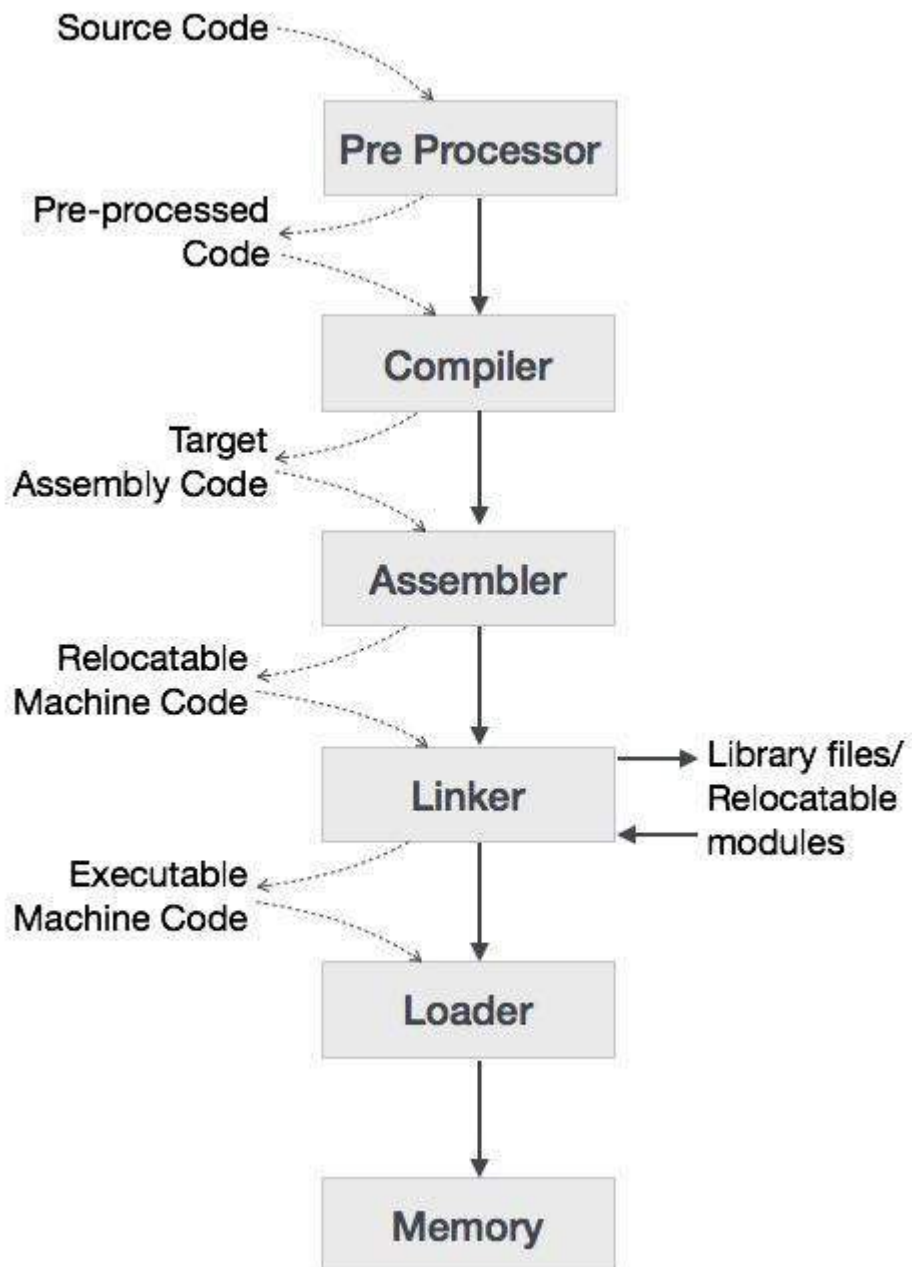
Computer Languages

Over the years, computer languages have been evolved from Low-Level to High-Level Languages. In the earliest days of computers, only Binary Language was used to write programs. The computer languages are classified as follows:



Assembly Language

An assembly language is a type of middle-level programming language that is intended to communicate with a computer's hardware. Unlike machine language, which consists of binary and hexadecimal characters, assembly languages are designed to be readable by humans.



Registers:

Registers are a type of computer memory used to quickly accept, store, and transfer data and instructions that are being used immediately by the CPU.

- The 32 MIPS registers are partitioned as follows:

- Register 0 : \$zero always stores the constant 0
- Regs 2-3 : \$v0, \$v1 return values of a procedure
- Regs 4-7 : \$a0-\$a3 input arguments to a procedure
- Regs 8-15 : \$t0-\$t7 temporaries
- Regs 16-23: \$s0-\$s7 variables
- Regs 24-25: \$t8-\$t9 more temporaries
- Reg 28 : \$gp global pointer
- Reg 29 : \$sp stack pointer
- Reg 30 : \$fp frame pointer
- Reg 31 : \$ra return address

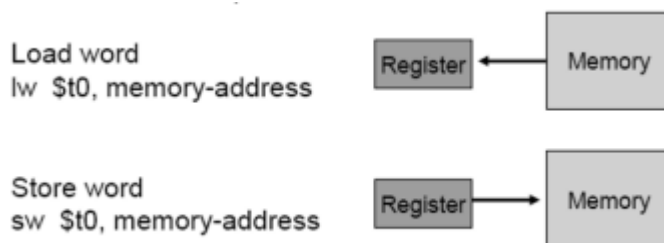
For example what instructions it has and what the instructions do. MIPS is a great language to learn the basics of assembly programming.

MIPS Instruction Set:(operation)

Arithmetic, Logical, Data Transfer, Conditional Branch, and Comparison Instructions. A MIPS instruction is 32 bits (always)

- **Data Transfer**

The MIPS instruction that loads a word from memory into a register is the **lw** instruction. The store word instruction is **sw**.



Memory Instructions Format

Lw \$ destination register's address, offset(\$ source register's address).
 SW \$source register's address, offset(\$destination register's address)

- **Logical Instructions**

Logical ops	C operators	Java operators	MIPS instr
Shift Left	<<	<<	sll
Shift Right	>>	>>>	srl
Bit-by-bit AND	&	&	and, andi
Bit-by-bit OR			or, ori
Bit-by-bit NOT	~	~	nor

e.g.:

- **and \$s1, \$s2, \$s3** (this mean: $\$s1 = \$s2 \& \$s3$ Bitwise AND)
 - **or \$s1, \$s2, \$s3** (this mean: $\$s1 = \$s2 | \$s3$ Bitwise OR)
 - **sll \$s1, \$s2, 10** (this mean: $\$s1 = \$s2 \ll 10$ Shift left by constant number of bits)
 - **srl \$s1, \$s2, 10** (this mean: $\$s1 = \$s2 \gg 10$ Shift right by constant number of bits)
-

- **Arithmetic Instructions**

C: `a = b + c`

assembly code: `add $s0, $s1, $s2` `# add rd, rs, rt`

rs: 1st register operand (register source)

rt: 2nd register operand

rd: register destination

SUB (Subtract) : sub rd, rs ,rt

Example:

Converting High-level Programming Concepts to Assembly

f = (g+h) - (i+j)

- assume f in \$s2, g in \$s4, h in \$s5, i in \$s6, and j in \$s7

Mips assembly:

`add $s1, $s4, $s5`

`add $s3, $s6, $s7`

`sub $s2, $s1, $s3`

or :

`add $s2, $s4, $s5`

`sub $s2, $s2, $s6`

`sub $s2, $s2, $s7`

ps:

1. instructions are simple: fixed number of operands(unlike C language).
 2. a single line of C code is converted into multiple lines of assembly code.
 3. some sequences are better than other.
-

Immediate Operands

When an instruction requires two operands, the first operand is generally the destination, which contains data in a register or memory location and the second operand is the source. Source contains either the data to be delivered (immediate addressing) or the address (in register or memory) of the data.

An immediate instruction uses a constant number as one of the inputs (instead of a register operand) .

```
a = a + 4;
```

```
b = a - 12;
```

MIPS Assembly Code

```
# $s0 = a, $s1 = b
```

```
addi $s0, $s0, 4 # a = a + 4
```

```
addi $s1, $s0, -12 # b = a - 12
```

Example:

convert to assembly: c code : **d[3] = d[2] + a** assume **d[0] in \$s4, and a in \$s1** ps: A MIPS instruction is 32 bits (always) , 4 byte

```
lw  $t0, 8($s4)      # d[2]
lw  $t1, 0($s1)      # a
add $t0, $t0, $t1    # $t0 = d[2]+a
sw  $t0, 12($s4)     # store sum in d[3]
```

Control Instructions

1. **Conditional Branch** : branch on equal (beq), branch on not equal (bne) **beq register1, register2 , label instruction** this mean : jump to label instruction if register1 equals to register2
2. **Comparison** : set on less than (slt),set less than immediate. (slti) **slt \$s1, \$s2, \$s3**
If true, set \$s1 to 1. Otherwise, set \$s1 to 0.
3. **Unconditional Branch**: jump (j), jump register (jr)

Example:

assume: f in \$s0, i,j in \$s3, \$s4 , and g,h in \$s1, \$s2

Convert to assembly:

```
if (i == j)                bne  $s3, $s4, Else
    f = g+h;               add  $s0, $s1, $s2
else                        j    Exit
    f = g-h;               Else: sub  $s0, $s1, $s2
                           Exit:
```

Example:

if (save[i] >= k)

Code;

Assume save[i] is stored in \$t1 and k is stored in \$t2

slt \$t0, \$t1, \$t2

bne \$t0, \$zero, Exit

Code

Exit:

Example:

if (save[i] > k)

Code;

Assume save[i] is stored in \$t1 and k is stored in \$t2

slt \$t0, \$t1, \$t2

bne \$t0, \$zero, Exit

beq \$t1, \$t2, Exit

Code

Exit:

Task(1):

if (save[i] < k)

Code;

Assume save[i] is stored in \$t1 and k is stored in \$t2

Task(2):

```

if (i < j)
    k = k+i;
else
    k = k+j;

```

convert to assembly code:

Example: loop

High-level code

```
// add the numbers from 0 to 9
int sum = 0;
int i;

for (i = 0; i != 10; i = i+1) {
    sum = sum + i;
}
```

MIPS assembly code: assume : sum in \$s1 , i in \$s0

```
        addi $t0, $0, 10
for:    beq  $s0, $t0, done
        add  $s1, $s1, $s0
        addi $s0, $s0, 1
        j    for
done:
```

Assignment Statements:

Ex1:

Show the single MIPS instruction for this C statement:

```
b = 25 | a;
```

Ex2:

Convert the following C fragment to equivalent MIPS assembly language. Assume that the variables a and b are assigned to registers \$s0 and \$s1 respectively. Assume that the base address of the array D is in register \$s2.

```
while(a < 10) {
    D[a] = b + a;
    a += 1;
}
```