Chapter 6

Definition 0.0.1: Architecture

"A programmer's view of a computer" Defined by the instruction set and operand locations (registers and memory)

Here are some examples of popular Computer architecures, most computers use ARM and x86

- x86
- ARM
- MIPS
- PowerPC
- SPARC

0.1 Assembly

assembly is the human-readable representation of a Computer Architecture's native language.

0.1.1 Instructions

Assembly is made up of single line instructions that tell the computer what and how to perform different operations

High-Level Code	MIPS Assembly Code

`a = b + c - d;  // single-line comment`
`/* multiple-line`
`comment */`
	`sub t, c, d  # t = c - d`
	`add a, b, t  # a = b + t`

0.1.2 Design Principles

> In this section we will include a few principles to remember. Here are two

Principle 1: Simplicity favors regularity - instructions with consistent number of operands is better. In the above image, we use multiple instructions instead of one instruction with multiple operands

Principle 2: Make the common case fast. - One thing that MIPS does is keeping the number of different operations low. More complex operations are made into multiple instructions.

0.1.3 Operands, Registers, Memory and Constants

> Operands are the things upon which the instuctions act on. They can be stored in registers, memory, or constants in the instruction itself. The instruction sets themselves only hold 32-64 bits

0.1.4 Registers

> Memory takes a long time to access so there are Registers to quickly access common operands.

MIPS uses 32 registers

Principle 3: Smaller is faster - For example, its is better to search through a small set of 32 registers than 1000.

High-Level Code	MIPS Assembly Code

`a = b + c - d;`	`# $s0 = a, $s1 = b, $s2 = c, $s3 = d`
	
	`sub $t0, $s2, $s3  # t = c - d`
	`add $s0, $s1, $t0  # a = b + t`