**HOW TO CREATE AN INSTRUCTON SET SIMULATOR**

* **Memory :**

+ ***Read hex file***: The simulator’s memory read hex file base on Intel hex format to defines how to represent hexadecimal data and memory addresses in hex files.

**Example:** :100094004111114506C6EFD0811DB7E706009387CD

":" : start code

10 : byte count 0x10 = 16 (dec) = 16 byte data

0094 : Address

00 : Record type = Data

4111114506C6EFD0811DB7E706009387 : data

CD : check sum = two’s complement (sum( address + record type +data ))

***+ Write memory*** : The simulator’s memory will store each part of data(usually 1 byte = 2 hexadecimal notation) for each address.

* **Fetch instruction**

***+ Program counter initialization***: The program counter (PC) is initialized to the memory address of the first instruction that needs to be executed.

***+Memory access***: The simulator’s memory uses the data from the address that program counter (PC) point to fetch instruction.

* **Initialization Registers :**

+ Initialize registers with default or initial values.

+ Create a data structure to represent registers: array, ..

**Ex:**  A architecture has 32 register, need create a array such as:

***Uint32\_t registers[32]***

+ Code function read and write to registers.

* **Initialization CPU, RV32 , Bus, Trace, Timer:**

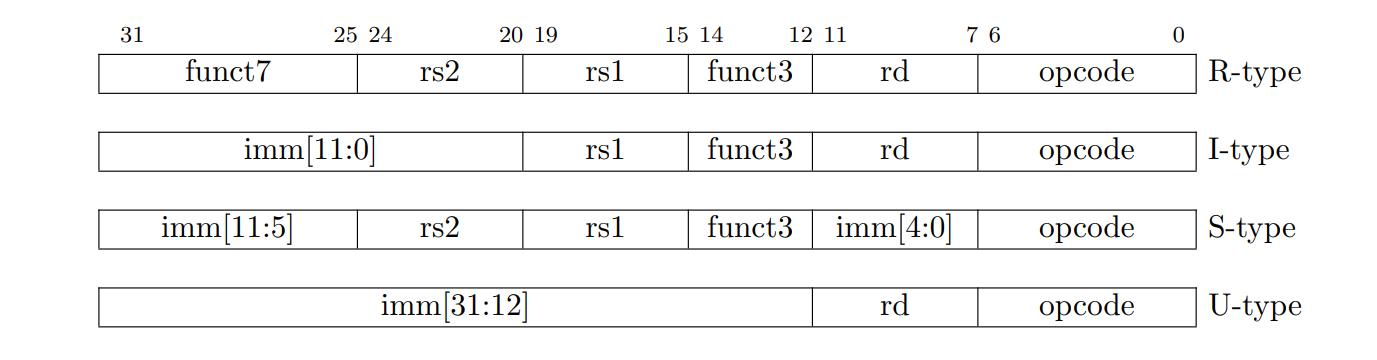
**+ CPU :**

* Initializes and sets initial values for various variables and components in the CPU class of an instruction set simulator.

**+ RV32 :** Defines the architecture of the system, including the number of registers, data width, memory, and other features related to the RV32 architecture.

* Base on PC take each address instruction to copy that data into a variable (Instruction\_variable).
* This variable(Instruction\_variable) is then decoded and executed.
* ***Prepare Base ISA***
* **Decode :**

***Ex:*** Instruction\_variable: 32 bit



**+** Check opcode ( Instruction\_variable[6-0]): Determine with bit[ 6-0] is belong what instruction.

**Exception:**

If bit[6:0] of Instruction\_variable is not unique to determine the specific instruction, we continue to examine funct3 and funct7 to conclude the exact instruction:

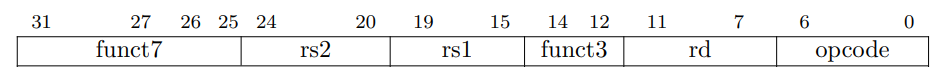
+ Check funct3 Instruction\_variable[14-12] Ex: ADDI, ANDI, ORI,...

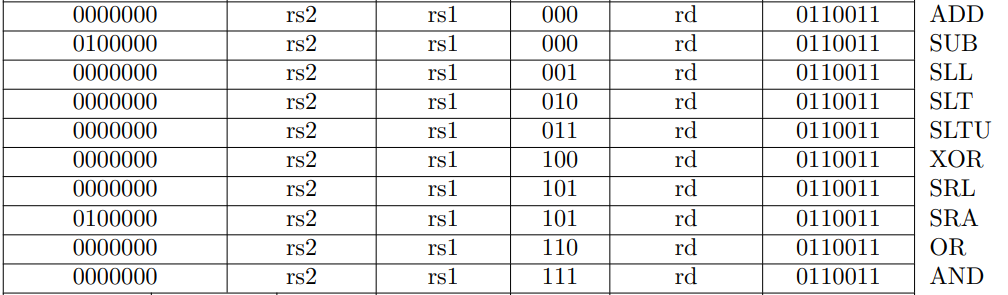
+ Check funct7 Instruction\_variable[31-25] Ex: ADD, SUB,...

**Ex:**

Instruction format:

0100000 00000 00000 000 00000 0110011





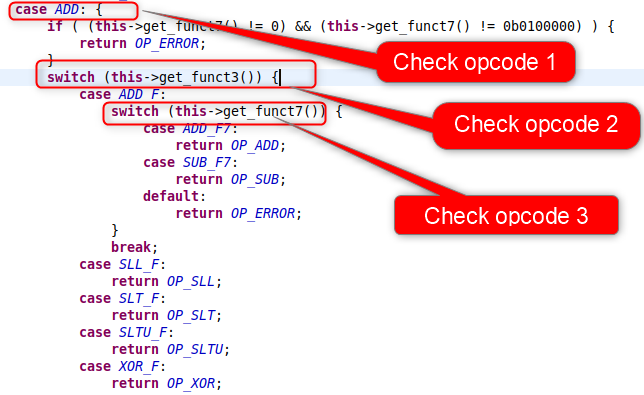
0100000 00000 00000 000 00000 0110011

Opcode 1 : 0110011

Opcode 2 : 000

Opcode 3 : 0100000

* SUB



* **Execute instruction**: Execute operation corresponse to determined instruction.
* **Update Program counter: PC** will be updated by + 4 bytes (only +2 for C extension) to take address of next instruction.

[1]. Mehrdad Reshadi, Prabhat Mishra, Mikil Dutt, *Hybrid-Compiled Simulation: An Efficient Technique for Instruction Set Architecture Simulation*, (pp.7-8/21),

<https://www.cise.ufl.edu/research/cad/Publications/tecs07.pdf>