High-Level RISCV Simulator

Utkarsh Rajauria
Electronics & Communication Engineering

Indraprastha Institute of Information Technology - Delhi

Delhi, India
utkarsh19214@iiitd.ac.in

Vishal Bansal
Electronics & Communication Engineering

Indraprastha Institute of Information Technology - Delhi

Delhi, India
vishal19217@iiitd.ac.in

Apoorv Singh
Electronics & Communication Engineering

Indraprastha Institute of Information Technology - Delhi

Delhi, India
apoorv19151@iiitd.ac.in

***Abstract* — We have implemented all functionalities of a high level RISCV processor that are Assembler, Simulator and Cache (Set Associative, Associative and Direct Mapped).**

I. Introduction

In this project, we have implemented the high level RISCV Simulator. We have divided it into two parts i.e. the Assembler and the Simulator. Assembler takes the raw assembly code as input and outputs a 32 – bit binary instruction, which then, is taken as input to the Simulator. There are five pipelined stages in the Simulator which are – Fetch, Decode, Execute, Memory and Write Back. For memory related operations, we have implemented single level cache which can perform tasks in three types of Cache architectures that are – Set Associative, Associative, Direct Mapped. This simulator is able to perform all defined assembly operations of the project.

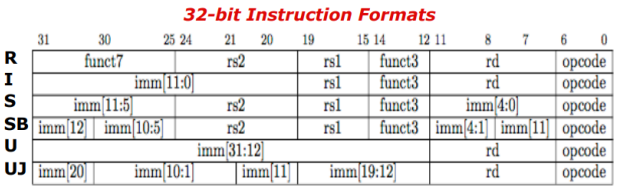
II. OPERATIONS IMPLEMENTED

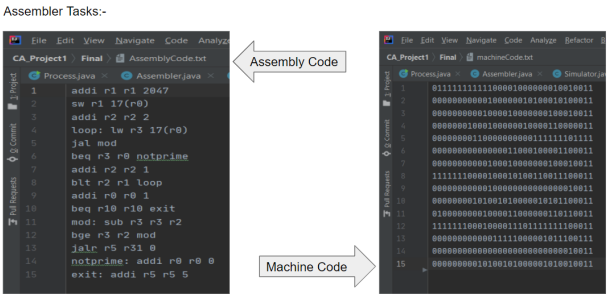
CODE	**OPERATION**
ADD	Addition
ADDI	Add Immediate
SUB	Subtraction
LW	Load Word
SW	Store Word
JALR	Jump and Link register
JAL	Jump and Link
BEQ	Branch Equal to
BNE	Branch Not Equal to
BLT	Branch Less than
BGE	Branch greater than equal
LUI	Load Upper Immediate
AND	AND
OR	OR
XOR	XOR
SLL	Shift Logic Left
SRA	Shift right Arithmetic

III. Architecture

A. Assembler

We have constructed the assembler on the official instruction format of RISCV processor where there are 6 instruction types that are - register type, Immediate type, Store type Branch type, Upper Immediate type, Jump & Link type. Corresponding bits for source registers, immediate values, address formation have been assigned in the 32 - bit instruction accordingly.





B. Simulator

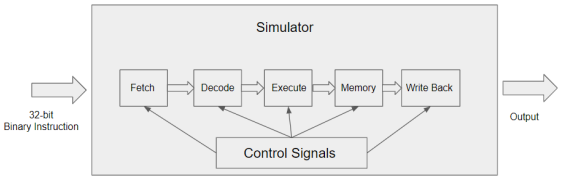
Simulator has been divided into 5 pipelined stages that are - Fetch, Decode, Execute, Memory, Write Back. The Simulator takes 32 - bit binary instruction as input and performs the task assigned through these stages. For memory operations, Single cache memory has been constructed with the three required cache architectures that are -- Associative cache, Set Associative Cache and Direct Mapped cache. Array data structure has been used to form tag array, data array and the main memory. Apart from this Cache is also able to implement 'write through' and 'write back' write policies and all the three replacement policies LRU(Least Recently Used), FIFO(First In First Out) and RANDOM. A function named 'updateControlSignals' has been created to generate and update control signals for given pipeline stages.

The parameters for the Cache taken as input are

1. Size (no .of cache lines)
2. Miss penalty
3. Hit time
4. Block Size
5. Associativity - Parameters - (Direct Mapped, Set Associative (K-way), Fully Associative)
6. Write policy - Parameters - (Write back (WB), write-through (WT)).
7. Replacement Policy - Parameters - (LRU, RANDOM, FIFO)

It is unified (data and instruction) cache

Flow of Instruction in Simulator:-

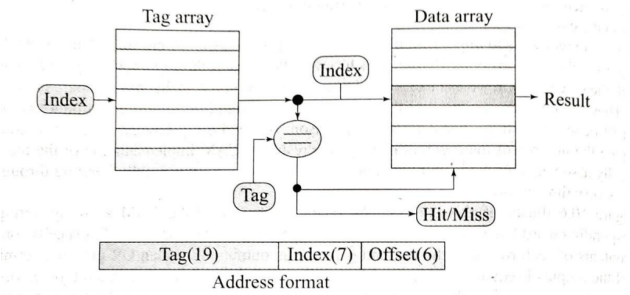


Implementation of caches

Direct Mapped Cache

Here, the 32- bit instruction has been divided into tag, index and offset. Tag is to determine the particular address corresponding to tag array and Index is uniquely specified to a particular position in the tag. This unique index is calculated by applying modulo operation on address in main memory with the size of the cache.

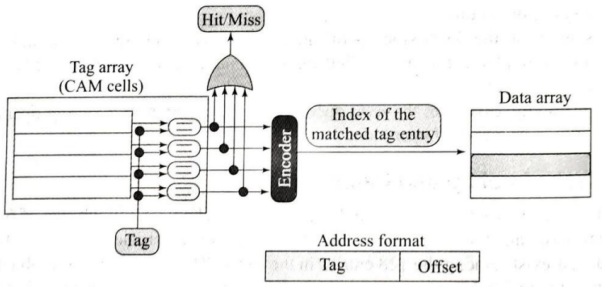
It is fast but loses temporal and spatial locality.



Fully Associative Cache

The fully associative instruction set consists of only two parts in 32 bit instruction, that are - tag for tag array to determine block and offset to determine specific work in that block. We look for every possible entry in the cache here to perform the operations.

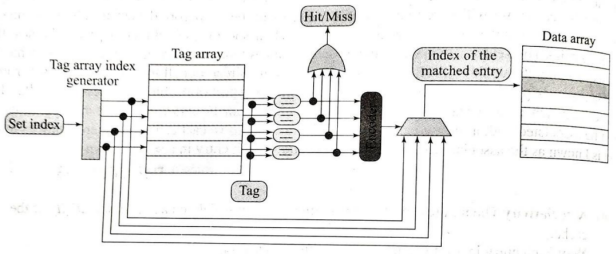
It preserve temporal and spatial locality if implemented with LRU replacement policy which we have done using a counter array to check which entry has been used least in the cache. Though, it is very time consuming process to check for every entry in this type of cache.



Set Associative Cache

It is a mid way between 'Direct Mapped' and 'Fully Associative' as here we divide cache into sets. Which set has to be accessed is determined by the 'Direct Mapped' procedure while the set contains a number of blocks in it where we check for every possible block in that set.

So, there will be a trade off here that how many blocks we want to put in a set according to our desired performance.



IV. OBSERVATIONS

Parameters of Main Memory are-

1) Access Time - 10 cycles
2) Size - 256 Memory Locations

Parameters of Cache used -

1) Hit Time - 2 cycles
2) Miss Penalty - 5 cycles
3) Block Size - 8 blocks
4) Cache Size - 32 cache lines
5) Associativity - 4 way Set Associative
6) Replacement Policy - LRU
7) WritePolicy - 2 writeThrough

Memory Structure	Performance Boost
Without Cache	316 Cycles
With Cache	155 Cycles

ACKNOWLEDGMENT

We wish to show our appreciation to our Instructor Dr. Sujay Deb as well as our Teaching Assistants and Teaching Fellow, who helped us during the implementation of this project. Their assistance played a crucial role in clearing our doubts and solving the various difficulties that we encountered during the course of this project.

REFERENCES

- https://msyksphinz-self.github.io/riscv-isadoc/html/rvi.html#ebreak
- https://riscv.org/technical/specifications/
- https://www.geeksforgeeks.org/cache-memory-in-computer-organization/