Report

Divvela Rakesh
CS23BTECH11017

November 12, 2024

RISC-V ASSEMBLER

Contents

1 Introduction

Developed a application for simulator running of all risc-v instructions.It also supports debugging tools like step break,del break points.It also shows stack.It also supports the simulation of cache_memory and statistics of Hits and misses

The whole implementation is divided into 4 seperate files named **main.cpp**,**implementation.cpp**,**cache.cpp**,**cache_simulator.hh**. functions used and implementation details are mentioned below.
All parts-1,2,3 are supported in my implementation

2 cache_simulator.hh

It basically contains all the function prototype used in the **assembler_implementation.cpp** and some global variables defined using **extern** key word.Also made a map of which stores label with line number a pair which consists string and int . It contains a new data type named sstack .Initialised a stack of type sstack.Vector of breakpoints is also declared here.
A type of struct is declared which is the building block of cache_memory.cache_mem is vector of nodes ,each element is of each set.
Some extra globlal variables like hits,misses are declared here with extern key word.
Also a bool for if the cache is enabled or not.

3 Implementation.cpp

3.1 Utility functions

In this utility functions are:

- **islabel**
  checks if the label is present in the map or not.
- **string_to_number**
  given an string with all integer characters and the base it is in .It converts the string to number in decimal format.
- **con_bin_to_hex**
  given an string in binary of n bits it converts it into a hexadecimal base and returns it.
- **register_number**
  given an register it returns the register number of it in its xa format.
- **number_binary**
  Given an number in decimal format and number of bits it gets the binary representation of it using rightshift operator until the number becomes

0.If a number is negative we represent it in 2's complement notation. we do this leveraging complement and 2's complement function.

- **complement**
  given an binary string it gives the complement of it.

- **twos_complement**
  In this given the complement string we add 1 to it to get 2's complement.Then return it.

- **datasection**
  It handles the storing of all data give in .data. It handles cases containing .dword, .word, .half, .byte.In this the .data section starts at 0x10000.

- **memoryinitialisation**
  This function initialises all the things in the memory to "0x00".

- **displaymem**
  It takes a string which is starting address and also an int which is no of memory locations to be printed.

- **showstack**
  It shows in which function we are in with line_no.

- **display_register_values**
  It displays the contents of registers.

3.2 Implementation functions

The implementation functions used in this implementation are:

- **execute**
  This takes an map consisting of all instructions with Program counters and total no of instructions.Then if ProgramCounter is less (total no of instructions)*4 Then it calls execute_one function which executes one instruction at a time.It also checks if the line number we are executing has break point or not if it has it stops executing.

- **execute_one**
  It takes a string containing instruction then it checks which format it belongs and then it calls the corresponding execute function for executions.The main thing we make reg[0]=0 before and after doing function calling making it hardwired to **zero**.

- **execute_R**
  It parses the instruction and gets the value of operands does the operation based on the instruction and The increments the value of ProgramCounter by 4;

- **execute_IA**
  Similarly here also it parses and the does the operation with the immediate it gets from instruction and then increments the value of ProgramCounter by 4.

- **execute_Il**
  It parses the instruction gets the value to be loaded from memory .we get memory address by adding immediate to the registervalue then store the value in hexadecimal format and it increments ProgramCounter by 4.In this jalr is also present The first operand register stores the value of Programcounter+4 and Then it updates Programcounter with offset+valueinregister.
  The above thing is if the cache is not enabled if the cache is enabled Then it first checks if the data is in cache or not by verifying tags of the specified index. If it is present(Hit) we fetch the data using **cachefetch function** else we first allocate it on cache and then fetch the data.

- **execute_S**
  It parses the instruction and then it gets the value which to be stored in memory and memory address at which it to be stored.Then increments the Program Counter by 4.
  Here also above thing is if the cache is not enabled.If it is enabled there are two cases
  1) Write Through
  In this if it is a hit we write the data in cache and memory.If it is a miss we update it only in memrory
  2) Write back
  In this if it is a hit we update it only in cache.If it is a miss we bring the block from memory and update it only in cache.If the cache is full for that index we remove the data by replacement policy,and update it if it is dirty block.

- **execute_B**
  it parses the instruction Then checks if the condition is satisfied or not.If it satisfies Then it updates ProgramCounter with Programcounter+offset.Else it increments ProgramCounter by 4.

- **execute_J**
  It parses the instruction Then updates the ProgramCounter with immediate we get,Then store the value of ProgramCounter+4 in first operand register.

- **execute_U**
  It parses the instruction then loads the immediate in 31-12 bits of register and sign bit extended till 63 bit.11-0 bits are made 0. Then Programcounter gets updated by 4.

4 Main.cpp

We created a command line interface for simulating a risc-v code.It contains

- load
- regs
- exit
- step
- break
- del break
- mem
- show-stack
- cache_sim enable config.txt
- cache_sim disable
- cache_sim invalidate
- cache_sim status
- cache_sim dump myfile.out
- cache_sim stats

4.1 load

We load the input file with command load input.s which contains series of instructions.We use "**ifstream**" for this.Then we extract each line using **getline()** function.

4.2 Handling Labels

We have to read the inputfile twice in this implementation.for the first time WE check if the label is present in the line or not .If it's present it adds it to the map with line_number of instruction.

4.3 regs

on giving command regs all the register values get printed in order from 0-31.

4.4 exit

It exits the code and gives a meaning full message.

4.5 step

It executes only the following instruction in the order and then waits for the next command.

4.6 break and del break

It sets the breakpoint at the given line and del break deletes the breakpoint at that particular line.

4.7 mem

It shows memory from the base address and the no of memory address it was given in the command.

4.8 cache_sim enable config.txt

It takes the configuration file and sets the Block_size,cache size ,etc. and initiliase the cache.

4.9 cache_sim disable

It disables the functionality of the cache.

4.10 cache_sim invalidate

It invalidates the cache blocks in cache by making valid bit zero.

4.11 cache_sim stats

It prints the statistics of the cache memory.

4.12 cache_sim status

It prints the configuration of the cache.

4.13 cache_sim dump myfile.out

Prints all the valid blocks of the cache.

5 cache.cpp

This file contains all function implementations which support cache memory.

- **Initialise cache**
  It creates a cache block and puts valid bit as 0 and dirty bit as 1 and pushes to cache mem.For fully associativity cache num_sets=1 and ways=cache_size/Block_size.

- **cache_invalidate**
  It makes valid bit of all cache blocks to zero.If the dity bit of the block is 0 it is writtem mem using write_to_mem function.

- **cache_details**
  It prints the cache configuration.

- **dumping**
  outputs all the valid blocks in cache.

- **cache_allocate**
  If the cache block is misses we allocate it on cache with,tag and data.If we want to allocate it but it is already of its max size then we remove one block and add .We remove that block using replacement policy.

- **Replacement policy**
  It calls the specificied replacement policy function based on replacement policy.

- **FIFO**
  It first erases the first element the cache blocks since as the name suggests "First in First out".Then we push back the newly fetched block from memory.

- **LRU**
  In this I remove the last element of the block and add the newly fetched block at the beginning.

- **Random**
  In this we remove the element based on the random number generated by random generator and then we push back newly fetched element.

- **Write to mem**
  It writes back the data to memory.

- **cache_fetch**
  It fetches the data based on the requirement datasize and if the memory is unaligned it throws out an error.

- **cache_write**
  It writes the elements in to the cache cache_block and assigns the dirty bit to 0.

- **cache_stats**
  It prints the statistics of the cache simulation.

- **disable**
  It popbacks all the sets of blocks in cachememory.

6 Testcases

The testcases used by me apart from given ones :

- Test case:1

```
.data
.dword 1, 2, 3, 4, 5
.word 10, 20, 30, 40
.half 100, 200, 300
.byte 1, 2, 3, 4
.text
lui x1, 0x10000
lui x2, 0x10000
addi x2, x2, 40
lui x3, 0x10000
addi x3, x3, 56
lui x4, 0x10000
addi x4, x4, 62
ld x5, 0(x1)
ld x6, 8(x1)
sd x5, 16(x1)
ld x7, 16(x1)
lw x8, 0(x2)
lw x9, 4(x2)
sw x8, 8(x2)
lw x10, 8(x2)
lh x11, 0(x3)
lh x12, 2(x3)
sh x11, 4(x3)
lh x13, 4(x3)
lb x14, 0(x4)
lb x15, 1(x4)
sb x14, 2(x4)
lb x16, 2(x4)
```

Listing 1: RISC-V Assembly Code

```
1024
16
1
LRU
WT
```

Listing 2: Config file

7 Limitations

- It can't handle if there are spaces before the commas in the input.

- if the memory acessed is outer than given bounds the program gets terminated.
- we can write into .text section Didn't handle the case where store instruction asks to store the value in .text dataspan.
- if given command not correct it just waits for next command without poping error message.
- can't handle emptylabels.
- if .data is present .text should be present else both shouldn't be present.
- outputfile is printed only if the program is runned entirely.
- can handle only single file instance.
- if we want run cache_simulation for a new file in multiple file instance there should be config file given again in my implementation.