Srikar Mikkilineni, Pragathi Ashok, Yathin Vemula
Computer Organization CSCI 2500
Professor Kuzmin

I. Design

Design 1:

Opcode	Instruction	Description	Example Usage
0001	Load	Load value from memory to accumulator	100 110E (Load SumLoc)
0010	Add	Add value from memory to accumulator	102 210F (Add Neg1)
0011	Store	Store accumulator value into memory	103 310D (Store CurrNum)
0100	Skip	Skip next instruction if AC==0	109 2000 (Skip if AC == 0)
0101	Jump	Jump to specificed instruction address	10A 5101 (Jump to loop)
0110	Halt	Halt program execution	10B 6000 (Halt)

Benchmark Program:

```
100 110E //Load SumLoc
101 110C //loop: Load FibNo /ac = 11
102 210F //Add   Neg1  /ac = 10
103 310D //Store CurrNum /10
104 110E //Load SumLoc / 11
105 210D //Add   CurrNum //ac = 21
106 310D //Store SumLoc // 21
107 110D //Load CurrNum
108 310C //Store FibNo
109 2000 //skip  AC == 0
10A 5101 // Jump loop
10B 6000 // halt
10C `h00B /FibNo, 11
10D 0000 //CurrNum, 0
10E `h000B //SumLoc, 11
10F 'h00FF // todo: 2's complement -1, Dec -1
```

Load 0001 1
Add 0010 2
Store 0011 3
Skip 0100 4
Jump 0101 5
Halt 0110 6

Machine instructions
---
100 110E
101 110C
102 210F
103 310D
104 110E
105 210D
106 310D
107 110D
108 310C
109 2000
10A 5101
10B 6000
10C `h00B
10D 0000
10E `h000B
10F `h00FF

Summary:
Addressing mode is direct where the operand directly refers to the memory address. The ISA uses an 8-bit format, where the first 4 bits are the opcode and the remaining 4 are the addresses or operands. We made it so the certain memory addresses such as 10C, 10D, 10E, and 10F are tailored to specific numbers such as the Fibonacci number 11, the current number, sum location, and Neg1.

Design 1A:

Opcode	Instruction	Description
0001	Load	Load value from memory to accumulator
0010	Add	Add value from memory to

		accumulator
0011	Store	Store accumulator value into memory
0100	Skip	Skip next instruction if AC==0
0101	Jump	Jump to specificed instruction address
0110	Halt	Halt program execution
0111	Load Immediate	Load Immediate value into accumulator

Given the 16Ki x 8 bits size for the L1 cache, the cache design would have a total cache size of 128 Ki bits. The best mapping method would be a direct-mapped cache because it offers a good balance between simplicity and performance. The cache structure is that it has a size of 128 Ki bits, the block size word hace 8 bytes per block and have a total of 16Ki blocks. With a 16 bit address space we can break it down into three parts, the tag (upper bits of the address), index (the middle portion to index into the cache), and the block offset (the lower bits to find exact words within a block). To implement the logic we must have cache access, cache miss handling, writing data, cache replacement. When a memory address is accessed the cache controller takes the index part of the address to find the corresponding cache line, it then compares the tag that is stored in the cache line with the tag part of the address, and if they match the bit is a set and a cache hit occurs. To handle a miss, the block of data must be fetched from the main memory and stored in the cache and then update the tag. To write the data, we can implement a write through policy which can be used simultaneously to write the cache and the main memory. Since we are using a direct-mapped cache, each main memory block must map to exactly one cache line, and if a new block needs to be loaded into a cache line then the existing one must be overwritten.

Design 1B:

Opcode	Instruction	Format	Description
1010	Add register	1010 RRRR SSSS TTTT	Add two registers, store in third
1011	Subtract register	1011 RRRR SSSS TTTT	Subtract two registers, store in third

1100	Add immediate	1100 RRRR SSSS IIII	Add register and immediate, store in register
1101	Branch Equal	1101 RRRR SSSS AAAA	Branch if two registers are equal
1110	Load Indexed	1110 RRRR SSSS IIII	Load using indexed addressing
1111	Store Indexed	1111 RRRR SSSS IIII	Store using indexed addressing

Summary:
The R, S, and T are the register identifiers and I is the immediate value and A is the address for branching.

II. Verilog:

CPU Module:

Program Counter (PC): The pc (Program Counter) is essential for fetching instructions from memory. In the waveforms, we see the pc incrementing as expected, indicating the CPU is fetching the next instruction on every clock cycle. For instance, the pc increments from 0 to 1 as the clk goes from low to high, which is a positive edge trigger that advances the program counter in your cpu module.

Instruction Fetch and Decode: Each instruction fetched by the pc is sent to the instruction_decoder module. This module decodes the fetched instruction into opcode, source and destination registers, immediate values, and jump addresses. The waveform shows various instruction values such as 1234, 2456, etc., being decoded with corresponding opcode and register values.

ALU Operations: The alu module performs arithmetic and logic operations. The control signal alu_control dictates the operation. For example, if alu_control is 0001, the ALU performs an addition operation as shown in the result waveform, where inputs a and b (a15:0 and b15:0) result in an output f.

Testbench and Debugging

Write Operations: The write_enable signal controls whether the ram module performs a write operation. When write_enable is high (1), the data present on the data[15:0] input is written to the memory location addressed by address[14:0]. The waveform shows data being written to memory[address] when write_enable transitions from 0 to 1.

Read Operations: When write_enable is low (0), the ram module outputs the data stored at the memory location addressed by address[14:0] to data_out[15:0]. This is indicated in the waveform where data_out shows XXXX when write_enable is 0, reflecting a read operation.

Waveform Analysis for Debugging:

Initial State: At time = 0, the system is reset, which sets up the initial state.

Data Write: At time = 10, write_enable is 1, indicating a write operation to RAM. The address 1a3b and data aaaa are reflected in the waveforms.

Data Read: At time = 15, write_enable goes back to 0, indicating that now data can be read from RAM. The data_out signal shows the data that has been read.

The model uses the clock signal to progress through instruction fetch, decode, execute, and memory read/write cycles. The waveforms corroborate the sequence of operations: fetch an instruction, decode it, execute the operation (like ALU computations), and interact with memory. The $monitor statements complement waveforms by providing a textual output of the internal state for each cycle, which is especially useful for pinpointing the exact moment an unexpected behavior occurs.

Contributions:

This project required a lot of work from all of us. As this project had multiple sections and multiple parts that had to be done we all sat down together and completed it step by step. We did this by having regularly scheduled meetings ever since the project.pdf came out and we were able to work on it. We all then made decisions as a team and built off the design that we thought was the best. Throughout this project all of us were equally responsible for every aspect of this project so no single part fell on one's shoulders. Overall this was a great project and all of us are happy with each other as a team and feel like we put in a good amount of work for this project.

Yathin: I worked on the SystemVerilog files and tested them, then I described how our Verilog model simulates execution of your benchmark programs and got waveforms to refer to.

Srikar: I worked on typing up the design of the ISA. This task was a little confusing at first as I wasn't sure where and how I should create the design but after attending the Webex meeting I came to an understanding how to design the ISA the way I did.

Pragathi: I engaged in the development of our programs using assembly language, and the translation of these programs into machine code. Additionally, I managed the storage of these machine codes in the instruction memory, a crucial component where the central processing unit (CPU) retrieves instructions for execution.

References:

Your project report must include the “References” section that lists all resources that you used when working on this team project. You don’t have to list every single resource that you accessed (e.g., if you opened a Web page just to see that it is not what you want) but every resource that you actually used (read, found useful, etc.) must be listed. Resources include not just our zyBook but also any other books, articles, blogs, forum posts, YouTube videos, tutorials, etc. For each reference you need to provide as much information as possible (title, author, publisher, publication date, URL, date accessed, page numbers or chapters, if it is a longer paper or a book, etc.)

Programiz. "C Program to Display Fibonacci Sequence." Programiz, Programiz, https://www.programiz.com/c-programming/examples/fibonacci-series., 12/03/2023

Menezes, Meesh. "Instruction Set Architecture." University of Maryland, Department of Computer Science, https://www.cs.umd.edu/~meesh/411/CA-online/chapter/instruction-set-architecture/index.html., 12/02/2023

McKinley, Kathryn S. "Lecture 06: Computer Organization." The University of Texas at Austin, Department of Computer Science, https://www.cs.utexas.edu/users/mckinley/352/lectures/06.pdf., 12/02/2023

Smith, John. "Arithmetic Logic Unit Design." Witscad, Witscad Inc., https://witscad.com/course/computer-architecture/chapter/arithmetic-logic-unit-design., 12/04/2023