RV32 Single Cycle CPU

1st Maya Hussein
Computer Science & Engineering
The American University in Cairo
Cairo, Egypt
mayamakram@aucegypt.edu

2nd Nada Badawi
Computer Science & Engineering
The American University in Cairo
Cairo, Egypt
nada_badawi@aucegypt.edu

I. Introduction

A single cycle CPU was initially the first processor utilized. Single cycle processors execute an instruction in one clock period, making the period to be set by the longest instruction, in that case, the load word instruction. Hence, introducing an architecture that executes multiple instructions per clock period would make the processor more efficient and faster. The project consists of two major milestones: 1. implementing a single-cycle CPU, and 2. adapting the code into a pipelined architecture. Currently, our program is able to support all 40 unprivileged user-level instructions from riscv32I on a single-cycle CPU with major release features. The instructions were adopted from the The RISC-V Instruction Set Manual Volume I: Unprivileged ISA [1]

*Note: Most modules were adopted from this course's lab; however, major changes were made to the following modules.

II. Discussion

The major adjustments were made to the main components: The immediate generator, data memory, control unit, and the ALU control unit. New components and features were introduced to be able to support all forty instructions, such as the branching component and the four by 1 multiplexers.

A. Immediate Generator

The Immediate generator is now different than the previously learnt module throughout the course. The component now handles the shifting of the immediate values whenever needed. To illustrate, the shift left 1 component is now removed since the immediate generator now shifts any value 1 bit to the left as long as it is a branch instruction. The implementation of the component was taken from the 'Verilog Support Code'.

B. Data Memory

The structure of the data memory was changed to be byte addressable in order to match the true RV32 architecture and to be able to correctly implement the following instructions: sb, sh, sw, lb, lbu, lh, lhu, lw. The changes made were in terms of resizing each register in the array of registers called 'mem'. Consequently, the way of writing and reading into the memory has changed; it also depends on the instruction itself. Since the RISC-V architecture is Little Endian, the data memory component was designed to be of the same byte ordering system, where the least significant bit is stored in the least memory address.

C. Branching Component

The branching unit was introduced to support the different branching instructions. Each instruction is based on a different condition, such as equal, not equal, greater than or equal, less than, less than unsigned, greater than or equal unsigned. There are four flags: zero, carry, overflow, and sign. These flags are used to decide on the condition or the instruction that we want to execute and whether the condition is satisfied to take the branch or not.

D. 4X1 Multiplexers

It was required as part of the design to install two four by one multiplexers in two different areas. The first one was used to choose the data to be written in the register file. We have one of three options: the output of the multiplexer that chooses between the output of the ALU and the data memory output, the PC value plus 4, and the target address coming out of the PC and immediate generator adder. The second use of the four by 1 multiplexer is the one needed to determine the next PC or the PC input value. We have four different values that the PC can take: previous PC value plus 4, output of the ALU, target address coming out of the PC and immediate generator adder, and a hardwired zero value. The latter option is used during the ECALL and FENCE instructions to jump to PC value of zero.

E. Control Unit

Four new signals were introduced to support the rest of the forty instructions. These signals are: ALUjump, PCplus4toReg, AddertoReg, and UpdatePC. The first one (ALUjump) is used as a select line to the four by 1 multiplexer that results in updating the PC value to the output of the ALU. PCplus4toReg is a signal used to control the four by one multiplexer that results in writing the current PC value plus 4 into the register file in the destination register determined by the instruction. The AddertoReg signal is used to result in making the destination register to be updated with the value of the target address, which is the output of the immediate and PC value adder. Finally, updatePC is a signal that is used to indicate whether the PC register should be updated or not. This signal is vital since we sometimes need to halt the PC value in case of EBREAK.

Control Unit Signals

Instruction	Inst[6-2]	Branch	MemRead	MemtoReg	ALUOp	MemWrite	ALUSrc	RegWrite	PCplus4toReg	AddrtoReg	ALUJump	UpdatePC
R-Format	01100	0	0	0	10	0	0	1	0	0	0	1
Arith I-Format	00100	0	0	0	10	0	1	1	0	0	0	1
Load	00000	0	1	1	00	0	1	1	0	0	0	1
Store	01000	0	0	0	00	1	1	0	0	0	0	1
Branch	11000	1	0	0	01	0	0	0	0	0	0	1
JALR	11001	0	0	0	00	0	1	1	1	0	1	1
JAL	11011	1	0	0	00	0	0	1	1	0	0	1
AUIPC	00101	0	0	0	00	0	0	1	0	1	0	1
LUI	01101	0	0	0	11	0	1	1	0	0	0	1
ECALL	11100	1	0	0	11	0	0	0	1	0	1	1
EBREAK	11100	1	0	0	00	0	0	0	1	0	1	0
FENCE	00011	1	0	0	11	0	0	0	1	0	1	1

ALUOp	Inst[5]	Inst[14-12]	Inst[30]	ALU Sel	ALU Op
00	X	X	X	0010	ADD
01	X	X	X	0110	SUB
10	1	000	0	0010	ADD
10	1	000	1	0110	SUB
10	1	111	0	0000	AND
10	1	110	0	0001	OR
10	1	100	0	0111	XOR
10	1	101	0	1000	SRL
10	1	101	1	1010	SRA
10	1	001	0	1001	SLL
10	1	010	0	1101	SLT
10	1	011	0	1111	SLTU
10	0	000	X	0010	ADD
10	0	111	X	0000	AND
10	0	110	X	0001	OR
10	0	100	X	0111	XOR
10	0	101	0	1000	SRL
10	0	101	1	1010	SRA
10	0	001	X	1001	SLL
10	0	010	X	1101	SLT
10	0	011	X	1111	SLTU
11	X	X	X	0011	PASS

F. ALU Control Unit

The following table represents the updated ALU Control Select Lines based on the ALU Op. The input to the module is 2 bits from the control unit (ALUOp), Inst5 that is the 5th least significant bit of the instruction, which differentiates between the Immediate and Arithmetic Instructions. Inst[14-12] distinguuishes between instructuions of every format, and inst 30, 30th bit of the instruction word, is used to differentiate between ADD & SUB, SRA & SRL.

G. Test Programs

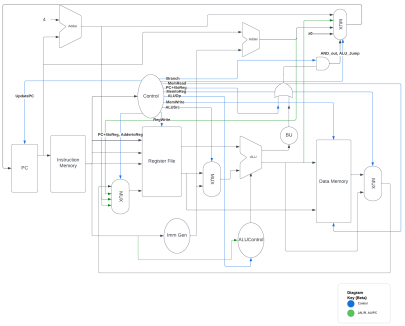
The test programs are installed in the instruction memory for now by reading the file using the readmemh() method. Whenever another test program is needed to be installed the file path should be changed in the instruction memory module. This feature required infinitesimal changes to be made to the instruction memory module only.

III. Schematic

IV. Future Prospects

The next release of the architecture will support a pipeline fashion to make better use of resources and reduce their idle time. Our Single-Cycle implementation still has room to be modified to support byte-addressable instruction memory with Little Endian byte ordering. This is crucial to get implemented since the next release will involve having a single memory (shared instruction and data memory) in our pipelined RISC-V architecture. Moreover, to make the testing easier, more feasible, and transparent, the file path is considered to be changed and accessible from the testbench. In other words, the file path of the test program will be read in the testbench and

Single Cycle CPU



Schematic

the hexadecimal content will be provided to the main module as an input parameter. In case we are implementing the design on an FPGA, it is considered that a sample of four programs will be loaded and installed as part of the implementation and two switches will be used to control which program to be tested.

V. GitHub Repo

The following is a link to the github repo that integrates everything: https://github.com/mayammakram/RISCV32_CPU

References

[1] Andrew Waterman, Yunsup Lee, David Patterson, Krste Asanović, "The RISC-V Instruction Set Manual, Volume I: User-Level ISA Version 2.1", Technical Report UCB/EECS-2016-118, EECS Department, University of California, Berkeley, May 31, 2016.