Design and Implement a simple 16-bit CPU

Author:

- Xiomara Figueroa Fontánez

Introduction

In the Fall semester 2013, as part of the Computer Architecture course, I, and each of my classmates, implemented an 16-bit CPU in Logisim. The specifications are described below:

Design

A register file with 16 16-bit registers feeds into a simple ALU. We use multiplexers, decoders and demultiplexers to select the registers for the ALU to process. A control table drives the components based on the instruction to be executed.

The design is a simplified version of the MIPS design presented in the textbook David A Patterson, John L Hennessy, *Computer Organization and Design, Fourth Edition: The Hardware/Software Interface*, 2009. ISBN 978-0-12-374493-7

Specifications

We had to build a simple CPU up from logic gates. We can use only the memory (RAM and/or ROM), but we built everything else up. The CPU will execute 16-bit instructions. With 16 registers we can let 4 bits represent the rs, rt and rd registers, and have 4 bits for the op:

0000	add rd rs rt	R format
0001	sub rd rs rt	R format
0010	lw rd rs rt	R format
0011	sw rd rs rt	R format
0100	and rd rs rt	R format
0101	or rd rs rt	R format
0110	not rd rs rt	R format
1000	addi rd imm8	I format
1001	ldhi rd imm8	I format
1010	bz rd imm8	I format
1011	bnz rd imm8	I format
1101	jal rd imm8	I format
1110	jr rd	R format
1111	j imm12	J format

Components

ALU

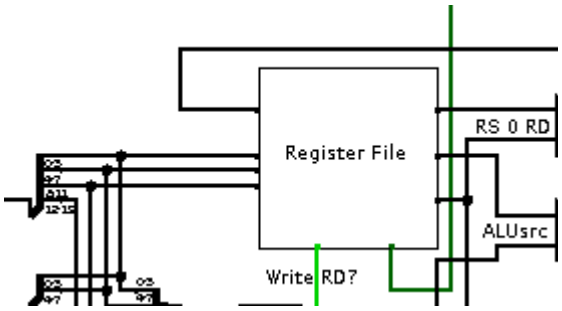
The ALU of this CPU is the simplest possible, thanks to Logisim. It is built with tiny circuits that Logisim already have for both, the adder and the subtractor. The ALU takes two 16-bit numbers and execute five operations. It's the ALU control who decides what would be the output of the ALU.

The following table shows the ALU control lines for each function of the ALU:

ALU control lines	Function
0000	OR
0001	AND
0010	ADD
0011	SUB
0100	NOT

The ALU also return, constantly, if the number is zero or not. It return true (1) if the number is zero or false (0) if not. This ALU can be improved with the implementation of the ALU carry look ahead.

Register File



As we can see, The register file receives many inputs and also outputs. It receives the data to be store at the register (first input in the left side). The others three in the left side are the register numbers (rt, rs, rd) from top to bottom. The two entries in the bottom (Write rd? And clock) are the bits that, when they're true (1) allows to write the register in the correct time.

The write port for a register file is implemented with a decoder that is used with the write signal to generate the input to the registers. All three inputs (the register number, the data, and the write signal) will have setup and hold-time constraints that ensure that the correct data is written into the register file.

Control

The input to the control unit is the 4-bit opcode field from the instruction. The output of the control unit consist of three 1-bit signals that are use to control multiplexers (ALUSrc, MemtoReg, RS o RD), three signals for controlling reads and writes in the register file and data memory (Write RD?, MRead, Mwrite), a 1-bit signal used in determing whether to possibly branch (PCSrc), and 4-bit control signal for the ALU (ALUOp). An AND gate is used to combine the branch control signal and the Zero output from the ALU. The AND gate output controls the selection of the next PC.

The setting of the control lines in completely determined by the opcode fields of the instruction. An R-type instruction writes a register (WriteRD? = 1), but neither reads nor writes data memory. The ALUSrc and ALUOp fields are set to perform the address calculation. The Mread and Mwrite are set to perform the memory access. The branch instruction is similar to an R-format operation, since it sends the rs and rt register to the ALU. Also, this control have a signal to determing a Jump. When the Jump control signal is 0, the PC is unconditionally replaced with PC + 1; otherwise, the PC is replaced by the PCSrc target.

Inst.	OP	PCsrc	WriteRd?	ALUsrc	ALUcontrol	MRead?	MWrite?	M/Reg?	Rs o Rd?
add	0000	00	1	0	0010	0	0	1	0
sub	0001	00	1	0	0011	0	0	1	0
lw	0010	00	1	0	0010	1	0	0	0
sw	0011	00	0	0	0010	0	1	1	0
and	0100	00	1	0	0001	0	0	1	0
or	0101	00	1	0	0000	0	0	1	0
not	0110	00	1	0	0100	0	0	1	0
addi	1000	00	1	1	0010	0	0	1	1
ldhi	1001	00	1	1	0000	1	0	0	1
bz	1010	10	0	1	0000	0	0	0	1
bnz	1011	01	0	1	0000	0	0	0	1
jal	1101	00	1	1	0000	0	0	0	1
j	1111	11	0	0	0000	0	0	0	0

PCSrc control lines	Function
00	PC + 1
01	BNZ
10	BZ
11	JUMP

PC + 1:

When the PCSrc control signal is 0, the PC is unconditionally replaced with PC + 1; otherwise, the PC is replaced by the branch target or the jump. The PC + 1 control line (00) add 1 to the Program Counter.

BZ:

When the PCSrc control signal is 0, the PC is unconditionally replaced with PC + 1; otherwise, the PC is replaced by the branch target or the jump. The branch zero control line (10) if it's 1, add the 8 bits immediate to the Program Counter; else, add 1 to the Program Counter.

BNZ:

When the PCSrc control signal is 0, the PC is unconditionally replaced with PC + 1; otherwise, the PC is replaced by the branch target or the jump. The branch zero control line (10) if it's 0, add the 8 bits immediate to the Program Counter; else, add 1 to the Program Counter.

JUMP:

When the PCSrc control signal is 0, the PC is unconditionally replaced with PC + 1; otherwise, the PC is replaced by the branch target or the jump. The jump PCSrc control line (11) add the 12-bit immediate to the Program Counter.

JAL:
The control have a row that is always zero and it's one only when the instruction is a jump to the link. This line goes to the multiplexer who decides what would be the PC src. Whenever is zero, whatever the PC src will be (BZ BZN or JUMP) is what is going to add to the program counter. Otherwise, if it is one (1) it would execute the jal, add the 8-bit immediate to the program counter and saves PC + 1 in rd.

ldhi:

The ldhi instruction have a separate control signal in the control unit. When the control input is equal to the opcode of the instruction, the only 1-bit signal that the second division of the control have would be equal to one (1) to determine that it is a LDHI and performs a shift of the 8-bit immediate to save it in the first 8 bits of the rd register.