ECE 212 – Digital Circuits II

Lab11 – Single-cycle Processor

Otieno Maurice (Scribe)

Alex Villalba

https://github.com/otienomaurice1/ece212_alex_maurice.git

Introduction:

The objective of this lab was to modify the mips multicycle processor to compute the ori and the bne instructions.

a) The bne instruction

During the bne instruction, the alu compares the values held in the rs and rt registers and drives the zero signal to low. The given data path can compute the branch address and update the pc so we don't need to modify anything in the data path. However, we need to modify the control unit to generate the correct pc source signal. To do this, we add a new signal specific to the beq instruction named *branchne*. The main decoder should drive the *brachne* signal high when the instruction opcode is the same as the bne opcode. Next, we NEGATE the zero signal and then AND it with the *branchne* signal to give a signal we named *branchnenotzero*. The non -negated form of the zero signal is ANDED with the branch signal to give a signal named *branchzero* signal. Finally, we OR the *branchnenotzero* signal with the *branchzero* signal to give the signal *pcsrc*. Consequently, when zero is high, branchnenotzero is low, *branchzero* is high and pcsrc is high and we compute the *beq* instruction. When *zero* is low, *branchnenotzero* is high, *branchzero* is low and pcsrc is high and we compute the bne instruction.

b) The ori instruction

The given data path can handle the or instruction but can not handle the ori instruction. The ori instruction requires a value from the rs register to be ORED with a zero-extended immediate. The data path does not have a module to zero-extend the immediate so we created a module named *signextzero*. This module creates a 32-bit value who's upper sixteen bits are all set to zero and the lower sixteen bits are those of the instruction immediate. The alu design can handle an or instruction so we don't need a dedicated OR module. To make the alu compute ori, we extend the alu source b two-input multiplexer to a three-input multiplexer. The third input into this multiplexer is the zero-extended immediate. A three-input multiplexer would mean a two-bit selector signal. We go back to the control unit to generate a two-bit signal to control this multiplexer. Inside the main decoder we extend the alusrc signal to two bits. If the instruction opcode is an ORI, the alusrc would be set to two and the alu source b multiplexer would select the zero-extended immediate. The ori instruction is not an R-TYPE instruction, so we need to provide a different alu op signal. In this case we set the alu op signal to 2'b11. Inside the alu decoder we added a statement that specifies that the alu should compute an or instruction when the alu op is 2'b11.

The diagrams showing these changes are given below.

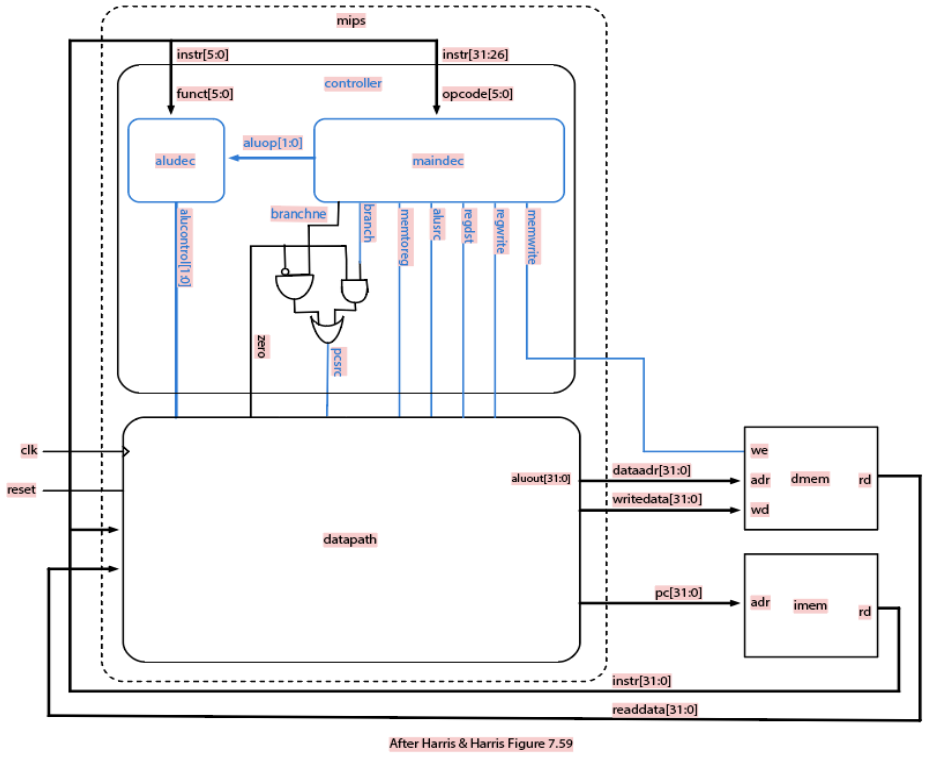

Figure 1top level diagram for the modified mips single processor

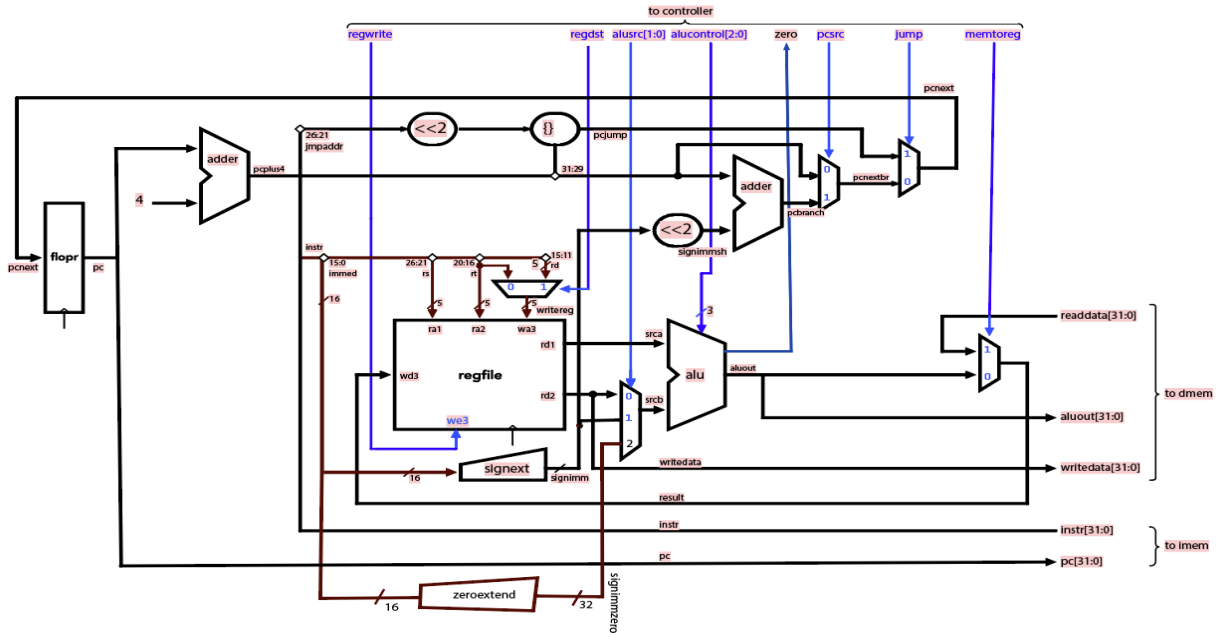

Figure 2 data path diagram for the modified mips single cycle processor

Extended functionality. Main Decoder:

Instruction	$Op_{5:0}$	RegWrite	RegDst	AluSrc	Branch	MemWrite	MemtoReg	$ALUOp_{1:0}$	Jump
R-type	000000	1	1	00	0	0	0	10	0
lw	100011	1	0	01	0	0	1	00	0
sw	101011	0	X	01	0	1	X	00	0
beq	000100	0	X	00	1	0	X	01	0
addi	001000	1	0	01	0	0	0	00	0
j	000010	0	X	XX	X	0	X	XX	1
ori	001101	1	0	1 0	0	0	0	11	0
bne	000101	0	x	00	1	0	x	01	0

Table 1showing the expected control and select signals for each instruction

Extended functionality. ALU Decoder:

$ALUOp_{1:0}$	Meaning
00	Add
01	Subtract
10	Look at funct field
11	or

Table 2showing the expected alu op and their meanings

Results:

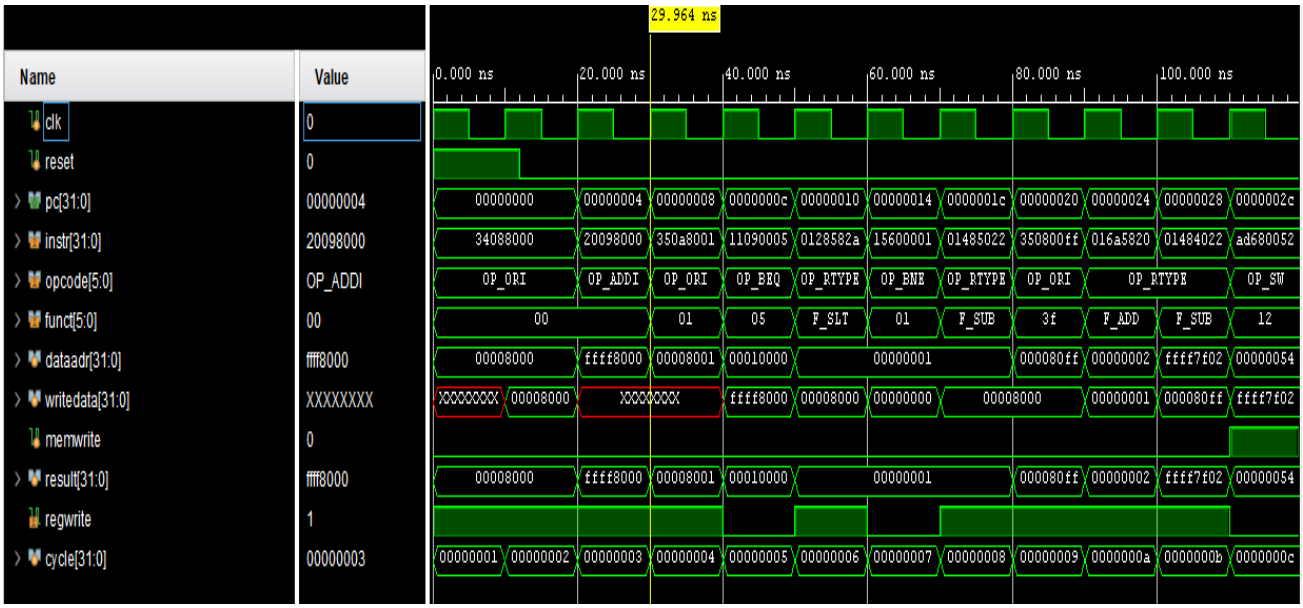

Figure 3simulation scope showing the execution of test.dat program

```
Built simulation snapshot testbench_behav
INFO: [USF-XSim-69] 'elaborate' step finished in '2' seconds
Vivado Simulator 2020.2
Time resolution is 1 ps
Simulation succeeded
$stop called at time : 120 ns : File "C:/Users/otienom/Documents/ece212_alex_maurice/lab8/sv/testbench.sv" Line 44
relaunch_sim: Time (s): cpu = 00:00:00 ; elapsed = 00:00:05 . Memory (MB): peak = 1742.020 ; gain = 0.000
```

Figure 4tcl console showing success of the test.data program

Conclusion:

In this lab we learnt how to extend the data path and generate signals for instructions that were previously no supported. We also learn that the data computed is written at the next clock edge for a given instruction. The knowledge gained would help us modify the processor to compute even more instructions should we be interested in doing so in the future.

Generally, this was a straight forward lab with no difficulties.

Time spent on lab:

2 hrs