Jackson Hsu & Krystal Kallarackal

EC413

12/3/15

**Milestone 4 Writeup**

Our multi cycle CPU includes components such as MUX, Registers, Data 0 extension, Sign extension, Instruction Memory, ALU, Controller, and the Datapath.

For our Controller, we have multiple control signals for the various other components of the design.

**CONTROLLER**

CONTROL SIGNALS

-PCSource (Controls the Next PC Source)

-00 Add 1 to Current PC and send to PC for writing

-01 Jump Address is sent to the PC

-10 Branch Address is sent to the PC

-ALUSrcA (Controls MUX ALU Operand1)

-00 Sends Current PC

-01 Sends Content stored in R2

-10 Sends 0

-ALUScrB (Controls MUX ALU Operand2)

-00 Sends Content of R1 or R3 in regard to ReadSelect

-01 Sends 1

-10 Sign Extended Imm

-11 Zero Extended IMM

-ReadSelect (Determines which Register to read for ReadData2)

-0 Read Register R1

-1 Read Register R3

-MemtoReg (Determines WriteBack Data)

-00 ALUOut

-01 ZE Imm

-10 Last 16 bits of R1, First 16 bits of ZE

-11 First 16 bits ZE, First 16 bits of R1

-ALUOP (Determines ALU operation)

-0000 MOV operation

-0001 NOT operation

-0010 ADD operation

-0011 SUB operation

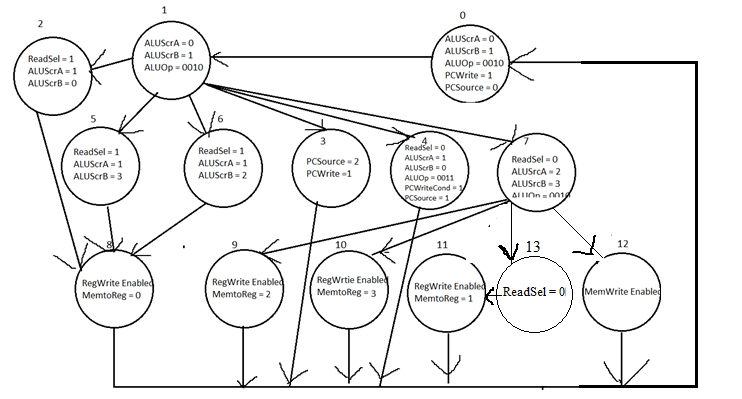
-0100 OR operation

-0101 AND operation

-0110 XOR operation

-0111 SLT operation

Below is the state diagram corresponding to our controller.



State 0

In this State the Instruction is read from the Instruction Memory. The Program Counter is

incremented while setting ALUSrcA as 0 and ALUSrcB as 1. The ALUOp is set to ADD (0010) operation. PCWrite is enabled and PCSource is sending out the control signal 00 to its respective MUX to take the Current PC incremented as the next PC.

State 1

In this state the Jump and Branch target addresses are calculated when the designated

instruction is to jump or branch. The address is calculated by adding the current program counter and the sign extended imm. ALUSrcA is set to 0 while ALUSrcB is set to 2. The ALUOp is set to ADD (0010) instruction.

State 2

This state deals with R-type operations. These instructions are all reading Register 2

and 3. By setting ReadSelect as 1, the ALUSrcB is set to 0 and will always be taking the contents of R3. ALUSrcA is set to 1 and will be taking the contents of R2. ALUOp is determined by the instructions opcode. This state proceeds into state8.

State 3

This state deals with the Jump Instructions operation. ALUSrcA and B are not relevant

for this operation since there is no condition needed for the Jump instruction to be taken. Therefore PCSource is set to 2 which is the jump target address. PCWrite is enabled as well as PCEn. This state proceeds into state0.

State 4

This state deals with the Branch Instructions operation. This instruction is a step more

complex than the Jump instruction as the Branch instruction requires a condition to be taken. This instruction compares the contents of R2 and R1. Therefore we set the ReadSel to 0. The ALU checks equality of the two values by using the subtraction operation (0011). If the result from the subtraction is equal to 0, then we set the Zero wire to 1. The wire goes through an AND gate with the PCWriteCond that is previously set to 1. PCSource is set to 1 which is the target address calculated for branch instructions by state2. This state proceeds into state0.

State 5

This state deals with Arithmetic and Logical Operations with a Zero Extended imm.

ALUSrcA is set to 1 and takes the value of R2. ALUSrcB is set to 3 and takes the value of the zero extended immediate. This state proceeds into state8.

State 6

This state deals with the Arithmetic and Logical operations that take a sign extended immediate. ALUSrcA is set to 1 and takes the value of R2. ALUSrcB is set to 2 and takes the sign extended immediate. This state proceeds into state8.

State 7

This state deals with loading and storing operations. The ReadSel is set to 0 so that we could concatenate half of the data in R1 to LI and LUI for the next state. ALUSrcA is set to 3 and takes 0. ALUSrcB is set to 2 and takes the sign-extended immediate. ALUOp is set to ADD(0010).This state proceeds into states 9 through 12 depending on the instruction.

State 8

This state deals with writing back the values calculated by the ALU to the register files. MemtoReg is set to 0 and takes ALUOut as a write-back data. RegWrite is also enabled so that it can take that value. This state proceeds into state0.

State 9

This state deals with the LI operation that proceeds state7. MemtoReg is set to 2 and takes the concatenation of the last 16 bits of R1 and the first 16 bits of ZE as write-back data. RegWrite is enabled to take that value. This state proceeds into state0.

State10

This state deals with the LUI operation that proceeds state7. MemtoReg is set to 3 and takes the concatenation of the first 16 bits of ZE and the first 16 bits of R1 as write-back data. RegWrite is enabled to take that value. This state proceeds into state0.

State 11

This state deals with the LWI operation that proceeds state13. MemtoReg is set to 1, taking M[ZE(Imm)] as write-back data. RegWrite is enabled to take that value. This state proceeds into state0.

State 12

This state deals with the SWI operation that proceeds state7. ALUOut is set to the data memory address to write the data to. MemWrite is enabled to store that value. This state proceeds into state0.

State 13

This state deals with the LWI operation that proceeds state 7. This state provides another cycle for the LWI operation to work correctly. This state proceeds into state 11.

**Brief Summary of each Instruction**

-*Jump*

> State Flow => 0-1-3-0

> State 1 calculates Jump target address by adding PC and SE Imm

> PCWrite Enabled to write over the current PC

-*Branch (BEQ)*

> State Flow => 0-1-4-0

> State 1 calculates the Branch target address by adding PC and SE Imm

> PCWriteCond enabled to write over the current PC only when conditions are met.

-*R-Type Logical/Arithmetic Instruction*

> State Flow => 0-1-2-8-0

> ALUOp instructs ALU to do computations.

> Computed result stored in R1

-*I-Type Logical/Arithmetic Instruction*

> State Flow for ZE => 0-1-5-8-0

> State Flow for SE => 0-1-6-8-0

> ALUOp instructs ALU to do computations.

> Computed result stored in R1

-*Load Immediate*

> State Flow => 0-1-7-9-0

> Immediate replacing first 16 bits of R1

> Concatenation stored in R1

-*Load Upper Immediate*

> State Flow => 0-1-7-10-0

> Immediate replacing last 16 bits of R1

> Concatenation stored in R1

-*Load Word Immediate*

> State Flow => 0-1-7-11-0

> Load data from data memory of address ZE Imm

> Data stored in R1

-*Store Word Immediate*

> State Flow => 0-1-7-12-0

> Store data from R1 to data memory of address ZE Imm

> No write back to the register file.

**DATAPATH**

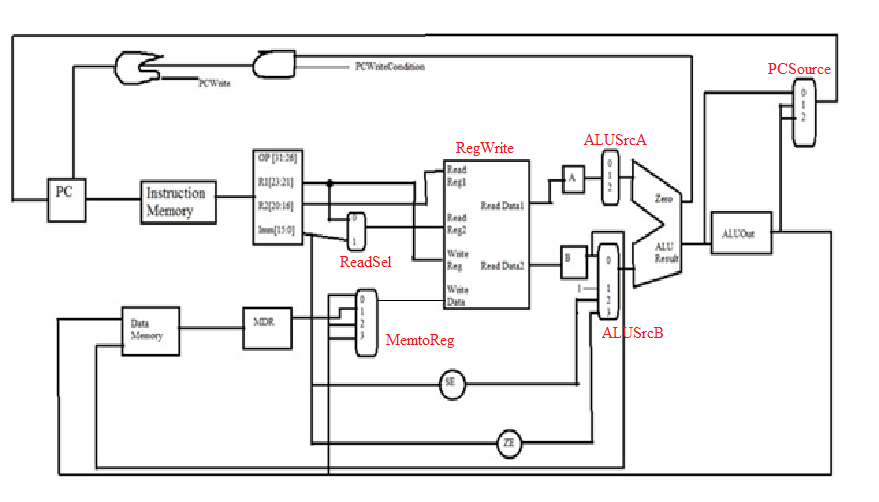
COMPONENTS:

-multiple MUX which take control signals and send the right output for the current state

-Instruction Memory (IMem) which stores 32-bits of Instructions.

-Data Memory (DMem) which stores 32-bits of Data.

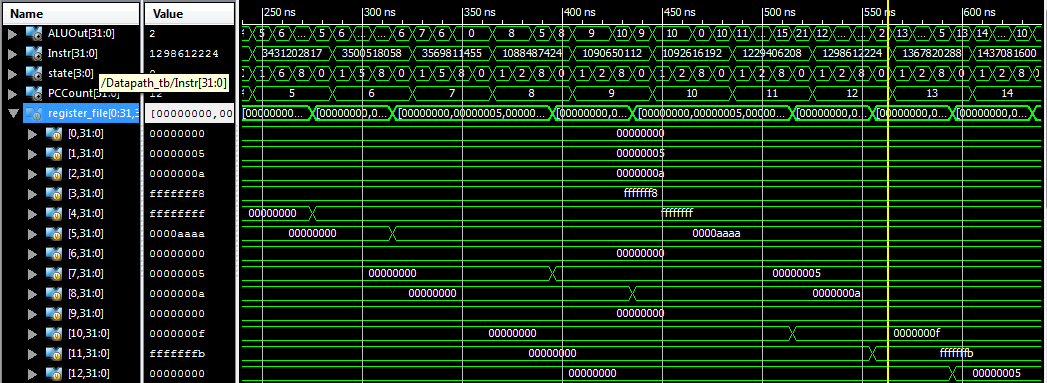
-General Purpose Register (GPR) which holds temporary data for a single cycle.

Above is a schematic of our CPU data path

**Waveforms**



Above is the waveform for the Jump instruction



Above is the waveform of the first instructions with their corresponding states and ALUOut