CSC/CPE 142

SPRING 2016

Term Project

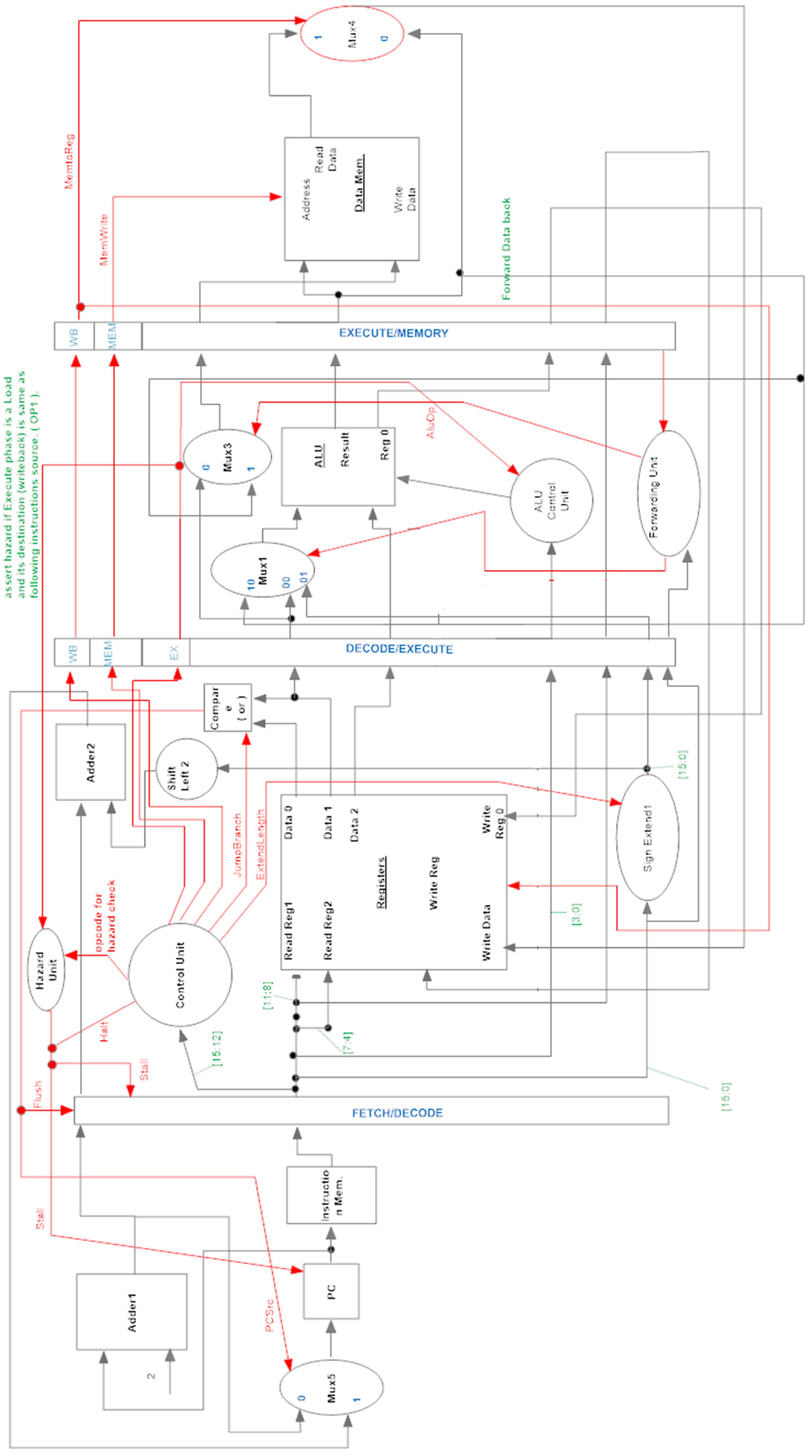
By

Andrew Sakoi – 50 %

Arnulfo Aguirre – 50%

Due 5-1-16

Table of Contents



MemtoReg
MemWrite
1
Mux4
0
Address
Read Data
Data Mem
Write Data
Forward Data back
WB
MEM
EXECUTE/MEMORY
assert hazard if Execute phase is a Load and its destination (writeback) is same as following instructions source. (OP1).
0
Mux3
1
ALU
Result
Reg 0
AluOp
ALU Control Unit
Forwarding Unit
10
Mux1
00
01
WB
MEM
EX
DECODE/EXECUTE
Compare (or)
Adder2
Shift Left 2
[15:0]
Data 0
Data 1
Data 2
Write Reg 0
Registers
Write Reg
Read Reg1
Read Reg2
Write Data
Sign Extend1
[3:0]
JumpBranch
ExtendLength
opcode for hazard check
Hazard Unit
Control Unit
Halt
Stall
[11:8]
[7:4]
[15:12]
Flush
FETCH/DECODE
[15:0]
Instruction Mem.
Stall
PC
PCSrc
Adder1
2
0
Mux5
1

Control Logic

PC:

Stall Control Signal	Operation
0	Perform normal operation
1	NOP

Control Unit:

[15:12] Opcode from Instruction	ExtendLength	JumpBranch	MemToReg	MemWrite	ALUOp	Halt	RegWrite
0000	X	000	0	0	10	0	1
1000	00	000	1	0	01	0	1
1011	00	000	0	1	01	0	0
0100 || 0101 || 0110	01	010 – 011 – 100 (respectively)	0	0	00	0	0
1100	10	001	0	0	00	0	0
1111	X	000	0	0	00	1	0

ShiftLeft Module:

ExtendLength control signal	output
00	Sign extend remaining 12 bits + [3:0]
01	Sign extend remaining 8 bits + [7:0]
10	Sign extend remaining 4 bits + [11:0]

PCSrc Control Signal (Mux 5):

PCSrc Control signal	Output
0	Current Address + 2 Bytes
1	Current Address + 2 bytes + offset value

Compare Module:

JumpBranch Control Signal	Operation	PCSrc Output Signal
000	NOP	0
001	Jump. Unconditional.	1
010	Branch if Reg0>OP1	1
011	Branch if Reg0<OP1	1
100	Branch if Reg0=OP1	1

Multiplexor 1 (ALU OP1 Input from Forwarding Unit):

ForwardOP1 Signal from Forwarding Unit	Output to ALU
00	Register1 Data direct
01	Data from Sign-Extend
10	Forwarded data from ALU output

Multiplexor 3 (Data sent to Memory from Forwarding Unit):

ForwardOP3 Signal from forwarding Unit	Output to Memory Module
00	Register 1 Data direct
01	Forwarded Data directly from ALU result

ALU Control Unit:

ALUOp Control Signal from CU	[3:0] from instruction	Operation ALU to perform	Output Signal
00	X – Don't Care	NOP	0000
01	X- Don't Care	ADD (for store/load)	1111
10	1111	ADD	1111
10	1110	SUB	1110
10	1101	Bitwise and	1101
10	1100	Bitwise or	1100
10	0001	Multiplication	0001
10	0010	Division	0010
10	1010	Shift left	1010
10	1011	Shift right	1011
10	1000	Rotate left	1000
10	1011	Rotate right	1011

Forwarding Unit:

D/EX buffer Funct	D/EX buffer Opcode	EX/Mem buffer Opcode	ForwardOP1	ForwardOP2	ForwardOP3
! (1011 || 1010 || 1000 || 1001)	0000 (read op1)	0000 (wb to op1)	10 (use forwarding)	0 (use reg2)	X
1011 || 1010 || 1000 || 1001	0000	0000	10	1 (use sign-ext immediate)	X
! (1011 || 1010 || 1000 || 1001)	0000	! 0000	00 (use reg1 data)	0	X
1011 || 1010 || 1000 || 1001	0000	!0000	00	1	X
X	1011(store)	0000	01 (use sign-ext)	0	1 (use ALU result to write)
X	1011	!0000	01	0	0 (use Reg1 Data to write)
X	1111 || 1100 || 0110 || 0101 || 0100	X	X	X	X
X	1000 (load)	X	01	0	X

Hazard Unit:

Decode Phase OPcode	Execute Phase Opcode	Hazard Stall Signal output
0000 || 0100 || 0101 || 0110 (Any reg read of OP1)	1000 (load)	1 (assert stall)
X	! 1000	0
1011 || 1000 (store or load)	1000	0

CSc/CPE 142

Term Project Status Report

Complete this form by typing the requested information and include the completed form in your report after TOC. Gray cells will be filled by the instructor.

Name	*% Contribution*	*Grade*
Andrew Sakoi	*50%*	
Arnulfo Aguirre	*50%*	

Please do not write in the first table

Project Report/Presentation 20%	/200
Functionality of the individual components 40%	/400
Functionality of the overall design 25%	/250
Design Approach 5%	/50
Total points	/900

A: List all the instructions that were implemented correctly and verified by the assembly program on your system:

Instructions	State any issue regarding the instruction.
Signed addition	Implemented ; but doesn't detect overflow properly
Signed subtraction	Implemented
bitwise and	Implemented
bitwise or	Implemented

Instructions	State any issue regarding the instruction.
signed multiplication	Implemented
signed division	Implemented
Logical shift left	Implemented
Logical shift right	Implemented
rotate left	Implemented
rotate right	Implemented
load	implemented
store	During testing, could not figure out how to get the memWrite signal to stop occilating (0~1) but does load, when asserted.
branch on less than	Implemented
branch on grater than	Implemented
branch on equal	Implemented
jump	Implemented
Halt	Implemented

B: Fill out the next table:

Individual Components	Does your system have this component	Does it work ?	List the name of partner who designed and validated this block	List problems with the component, if any.
ALU	Yes	Yes	Andrew Sakoi	Doesn't have exception handling Doesn't detect overflow
ALU control unit	Yes	Yes	Andrew Sakoi	None.
Memory Unit	Yes	Yes	Andrew Sakoi	Works when regWrite is asserted, however I can't seem to get the signal to stop occilating. Unsure why that is.
Register File	Yes	Yes	Andrew Sakoi	none
PC	Yes	syntax	Arnulfo	syntax errors
IM	Yes	syntax	Arnulfo	syntax errors
Converting the HEX values to Machine code	Yes	syntax	Arnulfo	syntax errors
Phase1 and IF/DC Buffer	Yes	Yes	Arnulfo	Syntax errors on phase 1 only
DC/EX Buffer	yes	yes	Andrew Sakoi	none

Individual Components	Does your system have this component	Does it work ?	List the name of partner who designed and validated this block	List problems with the component, if any.
EX/M Buffer	yes	yes	Andrew Sakoi	none
Phase 2(for testing)	Yes	illegal assgnment	Andrew Sakoi	Illegal assignment with wires.
SignExtend	Yes	yes	Andrew Sakoi	none
Shift left	Yes	yes	Andrew Sakoi	none
Mux 5	Yes	yes	Arnulfo	none
Mux 1	Yes	Yes	Andrew Sakoi	none
Mux 3	Yes	Yes	Andrew Sakoi	none
Mux 4	Yes	Yes	Andrew Sakoi	none
exception handler 1. Unknown opcode 2. Arith. Overflow …..	No	no	X	X
Control Unit	Yes	Yes	Andrew Sakoi	none

Individual Components	Does your system have this component	Does it work ?	List the name of partner who designed and validated this block	List problems with the component, if any.
Hazard Unit	Yes	Yes but…	Andrew Sakoi	Unable to test entire System so unsure as to whether it will hold under a combination of instructions. On paper yes.
Forwarding Unit	Yes	Yes but…	Andrew Sakoi	Unable to test entire System so unsure as to whether it will hold under a combination of instructions. On paper yes.

How many stages do you have in your pipeline? ……………. 4 Phases.(Combined Memory and Write Back in to 1 phase.

C: **State any issue regarding the overall operation of the datapath?**

When designing this datapath on paper, every module seemed to work out according to specs both individually and as a whole. However, we were not able to get any phase, let alone the entire system to run without any compilation/syntax errors. For this reason, we are unaware of how the system would perform, as a whole.

Our Hazard Unit seems to be on point when checking the instruction in the Execution Phase, and comparing this value with that of the following instruction, in the Decode Phase. If a LW occurs, which only uses OP1 as a Destination, than proper logic checking in the Decode phase on part of the Hazard unit will decide whether a hazard is present. In this case, the Hazard Unit is prepared to submit the proper signals to the Buffer and the PC. Whether or not the Hazard Unit would have submitted the signals in time, using our combinational logic, has not been proven by our system.

Another Issue, that I was unable to test due to our system not working without errors, was that of the Compare module. While we have tested to the validity of the compare module, I would speculate that the timing of the signal from the compare module, during a jump/branch would cause issues with our current design. After doing some additional research, I have a hard time grasping the idea as to when and how exactly the compare module should assert a flush/stall during a jump/branch. Is it to occur by sending a signal through the Decode/Execute buffer? Or immediately send it back to the PC.

A few other issues I can speculate about would be of the same nature as the Compare situation. Knowing precisely how and when to submit the signals from the forwarding unit can be quite a challenge. At first thought, it seems intuitive, but when attempting to verify, using Verilog, we have realized that it can be challenging to make all modules in a given phase work seamlessly in 1 clock cycle.

With the Program Counter not being able to run correctly because of syntax errors it is hard to test the addresses that are sent to instruction memory which also has syntax errors causing us problems on not knowing if they run correctly. If this was able to run correctly we be able to convert out HEX values into binary. I suspect that the issues for these syntax errors has to with needing more time understanding on how each module would work using Verilog and getting all the functions correctly to perform each modules task.

Phase 1 isn’t able to run correctly due to other modules having syntax errors. I was also not sure if I was properly connecting all the modules together for them to run simultaneously and go into the buffer to get a result. Not sure if I couldn’t grasp doing this or haven’t had enough experience in working with Verilog to complete the task.