Project 2 Design Document

Kevin Greer (keg84) Yungton Yang (yty4)

March 26, 2015

1 Introduction

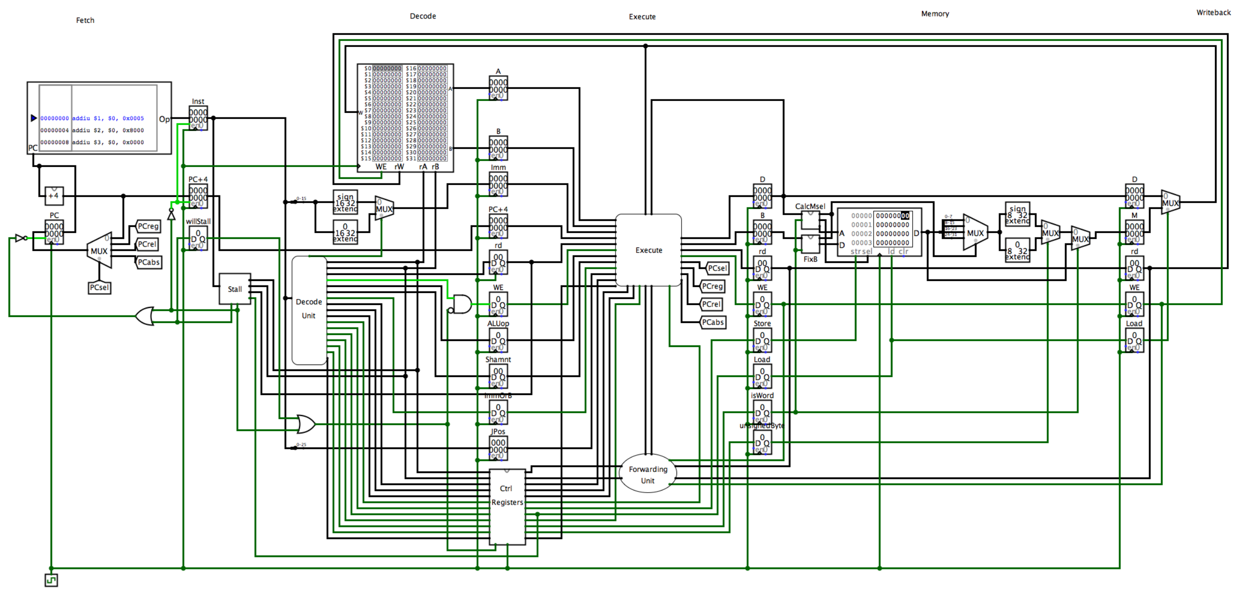
This document intends to explain and clarify the MIPS five stage pipelined processor developed by Kevin Greer and Yungton Yang.

2 Overview

This five stage pipelined processor is intended to perform a large subset of instructions in the MIPS assembly language. The five stages of the circuit will be:

1. Instruction Fetch
2. Instruction Decode
3. Execute
4. Memory
5. Write back

Between each stage is a set of register that stores certain values calculated by previous stages and needed in future stages. Since there are five stages in this processor, five different instructions can be processed in parallel. It employs data forwarding to prevent data hazards, strategic stalling to prevent control hazards, and there is a single clock connected to every register in the circuit.

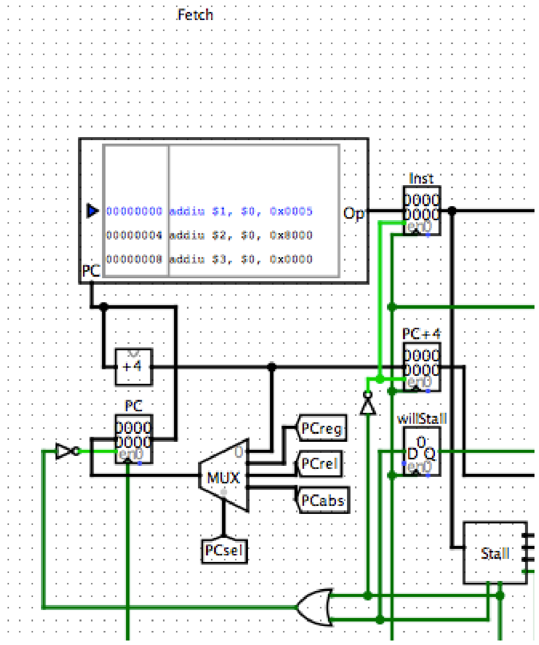


A high-level view of the pipelined processor

3 Stages

3.1 Fetch

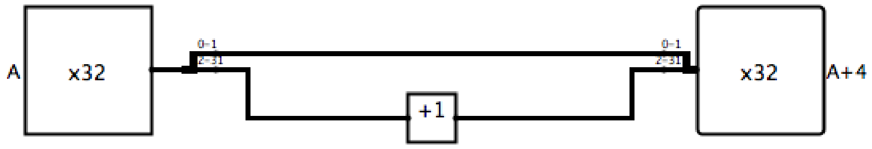
The first stage of the five stage pipelined processor is instruction fetch (IF). In this stage, current value of PC, the program counter, is input into Program ROM which contains all of the instructions for the processor. The 32-bit instruction at the address input by PC is output from the Program ROM into the IF/ID pipeline register.


Fetch
00000000 addiu $1, $0, 0x0005
00000004 addiu $2, $0, 0x8000
00000008 addiu $3, $0, 0x0000
PC
Op
Inst
PC+4
+4
PC
willStall
MUX
PCreg
PCrel
PCabs
PCsel
Stall

The instruction fetch stage

Notice that each register has an enable bit input that is the negation of the stalling unit in decode. The details of the stalling unit will be discussed in the decoding section, but note one or all of the registers in instruction fetch are disabled when a stall occurs.

While this happens, PC will go through an incrementer to calculate PC + 4. Since the given incrementer only adds 1, we will add 4 by splitting off the 0th and 1st bit and adding 1 to the rest of the bits then merging the 32-bits back together. This essentially is the same as adding 4 to the original 32-bit number. After being calculated, PC + 4 will be input to the pipeline register to be used in future stages.

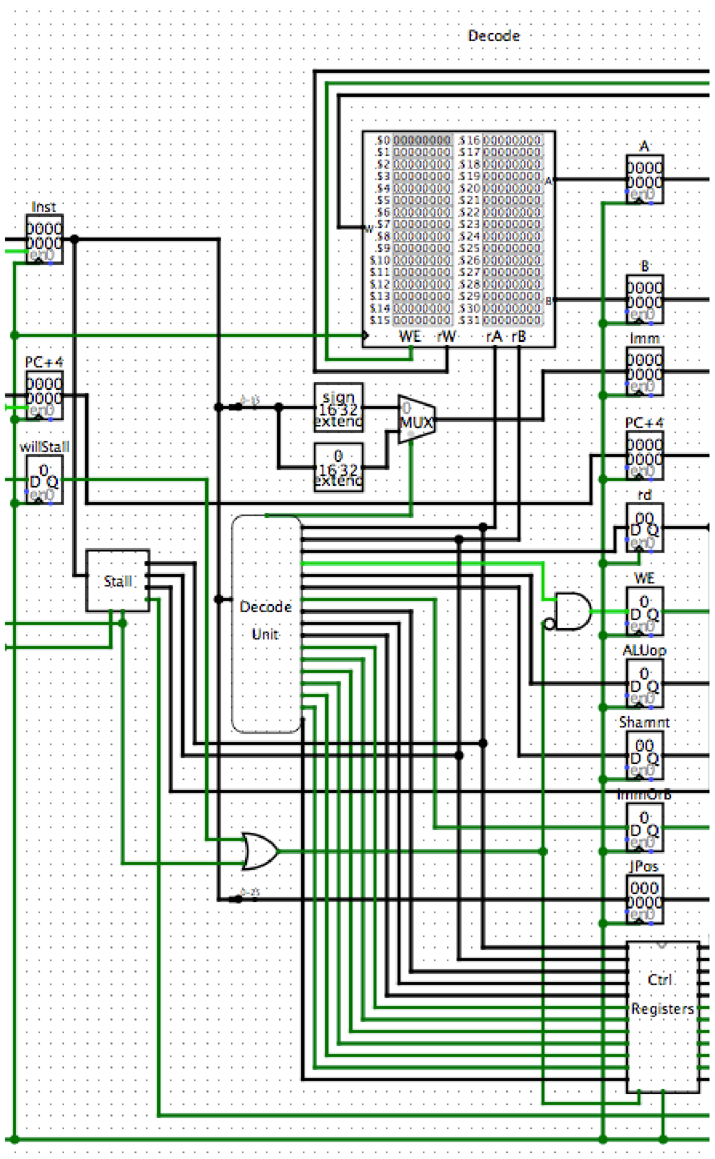


+4 Incrementer

Then, the new PC value is determined by a multiplexor with two select bits, PCsel. The four options for the new PC are PCreg, PCrel, PCabs, and PC + 4. The latter three values are calculated in the Execute stage. For each instruction, PCsel will be calculated and will choose the proper PC value for the instruction.

3.2 Decode

The second stage of the five stage pipelined processor is instruction decode (ID). In this stage, we feed the instruction from the instruction pipeline register into a decoding unit that will decode the instruction for use in the rest of the pipeline. From the register file, the values stored in registers A and B will be output to the ID/Ex pipeline register. The immediate value will be either sign extended or zero extended and stored into the pipeline register. PC + 4 will be passed through to the ID/Ex pipeline register for use in future stages. The control logic will also be stored in the ID/Ex pipeline register to be used in the execution stage.



The instruction decode stage

Also in this stage, the stalling unit determines whether or not to stall the pipeline (i.e. insert a NOP into the pipeline while not changing the IF/ID registers). Instead of inserting an actual NOP, our stalls instead set ”Write Enable” and ”Memory Write (a.k.a. Load)” to 0, which effectively makes the instruction do nothing. Since we chose to calculate branches and jumps in the execute stage, there are two different types of stalls in our circuit, and thus there are two different outputs of the stalling unit: Stall later, and stall now (the computation will be discussed later).

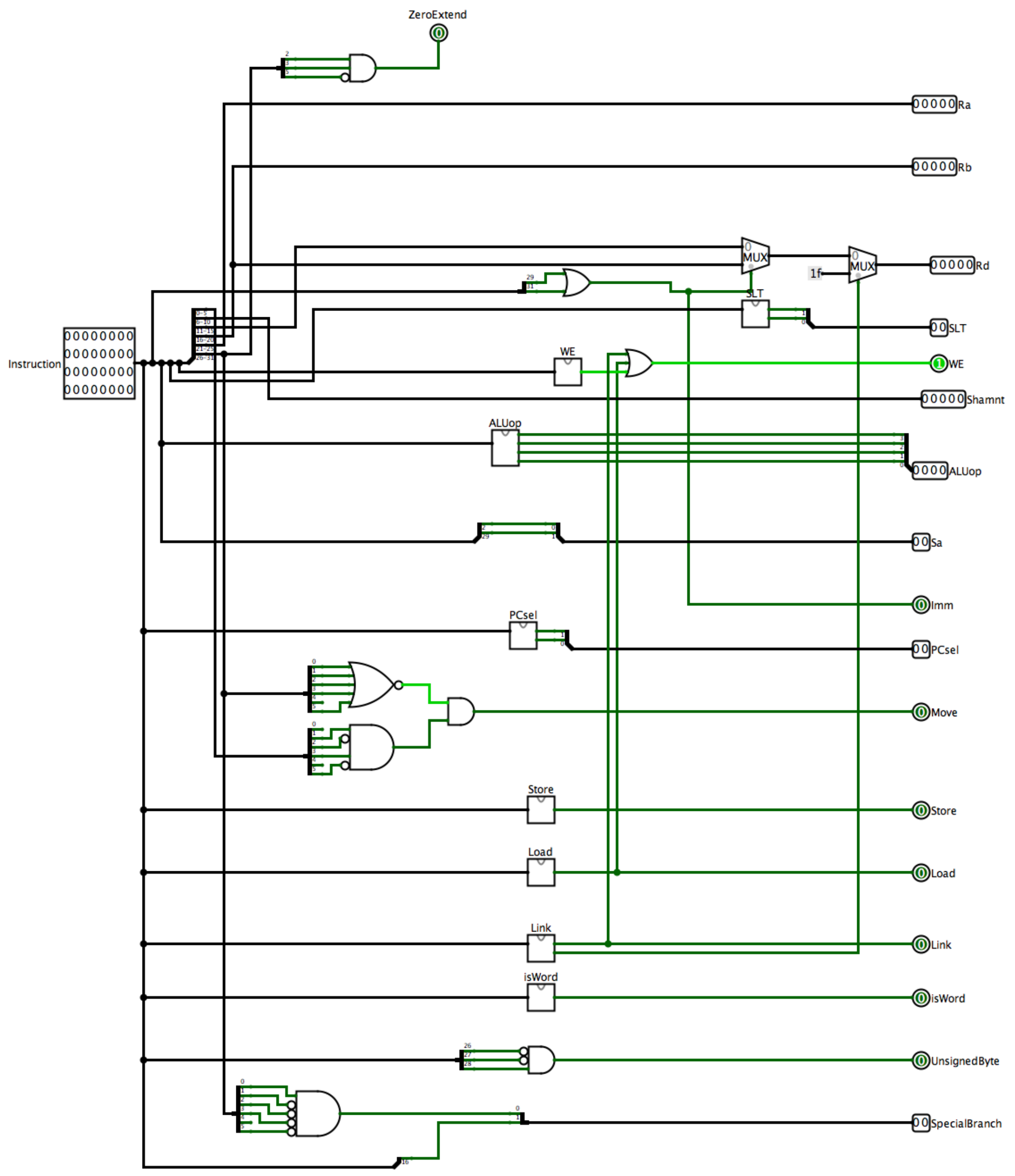
1. Control hazard stall (Stall later): A stall that must occur to give the circuit time to determine the correct next instruction. Since the circuit implements one delay slot, one stall must also be executed. If the instruction is a jump or a branch, the circuit must let the current instruction through the pipeline (the branch or jump itself) and stall on the next instruction. This is accomplished through the register one-bit ”willStall”

in the IF/ID pipeline. The output "StallLater" is put into the "willStall" register, and the register causes a single stall on the following cycle. Stalls of this type only disable the PC register.

2. Load-use stall (Stall now): A stall that must occur to give the memory stage time to read from memory and forward the correct value to the execute stage. A stall of this type must occur immediately, that is: it must hold the current instruction while sending an effective NOP through the pipeline in its place. This is accomplished by disabling the PC register, the PC+4, and the instruction register while changing "Write Enable" and "Memory Write" to 0.

Stalls occur if "willStall" or "StallNow" is asserted. "Write Enable" and "Memory Write" are changed to 0 by pushing them through an AND gate with the negated stall. E.g. Write Enable will be asserted only if it is calculated to be asserted AND stall is not asserted.

3.2.1 Decoding Unit



The instruction decode stage

The decoding unit takes the 32-bit instruction as an input and has 18 outputs:

1. rA[5]: The first register to be read. This is always bits 21-25 no matter what the instruction so bits 21-25 are output as rA directly.

2. rB[5]: The second register to be read (if the operation does not call for a second register to be read, this value will be irrelevant). This is always bits 16-20 no matter what the instruction so bits 16-20 are output as rB directly.

3. rd[5]: The destination register of the result (computed in DecodeRD/ImmOrB). This puts the OpCode of the instruction through an OR gate which will decide whether

the ALU should use the B value from rB or the immediate value from the instruction. This is because all R-type instructions have a 0 opcode, while all I-type instructions have a nonzero opcode. The output from the DecodeRd/ImmOrB circuit is used as the select bit for a multiplexor which decides whether to use bits 11-15 (R-type) or 16-20 (I-type) as the rD.

4. SLT[2]: This determines if the instruction is any sort of SLT (SLTI, SLTIU, SLT, SLTU) and is used as control logic for muxes in the execute stage. The 0th bit of SLT determines if the SLT is signed or unsigned and the 1st bit decides if the operation is in fact some sort of SLT instruction. SLT[2] is computed in the circuit DecodeSLTMux. The inputs of the truth table are bits 0,1,2,3,5,26,27,28,29, and 31 from the instruction and the two outputs are the 1st bit of SLT and the 0th bit of SLT. The circuit was created with Logisim's "analyze circuit" tool which creates a circuit based on the truth table. These bits were chosen from the instruction because these are used to decide whether the instruction is a SLT and if its signed or not. For this project, the 30th bit is always 0 so it does not affect the outcome. The outputs of SLT are put into a splitter and output as SLT[2].

5. WE: Write enable of the register file. Write enable is used in the WriteBack stage to decide whether or not to allow the instruction to write to a register. WE is computed by using an OR gate of 3 things: the output of Decode WE, the output of DecodeLink, and the output of DecodeLoad. The DecodeWE circuit implements the truth table and minimizes it. The truth table takes bits 1, 3, 26, 27, 28, 29, and 31 as inputs and outputs WE. These bits determine whether the instruction will need WE to be 1 or 0. This was determined by the fact that WE is 1 for every instruction except for branches, sw, sb, j, and jr. The outputs of DecodeLink and DecodeLoad show if the instruction is a link (like jalr and jal) or a load (lw, lb, lbu). So if the output of DecodeWE is 1, the instruction is a link, or the instruction is a load, then WE should be 1. The circuit in DecodeWE was created with logisims analyze circuit tool which creates a circuit based on the truth table.

6. ALUop[4]: The control signal for the ALU in the execute stage. This is computed in the circuit DecodeALUop which implements a truth table containing bits 0, 1, 2, 3, 5, 16, 26, 27, 28, 29, and 31 from the instruction as input and outputs the 4 bit ALU op code accordingly. Bit 30 is not needed because all instructions requiring the ALU have 0s for those bit. The circuit was created with logisims analyze circuit tool which creates a circuit based on the truth table.

7. Shamnt[5]: Shift amount. Bits 6-10 are taken directly from the instruction and output as the shift amount. Shamnt is then used in the execute stage later on.

8. Sa[2]: Control logic which is used in the execute stage. The 0th bit of Sa decides whether the ALU should use the bits 6-10 for a regular shift or the bits 21-25 in rA for a variable shift. The 1st bit of Sa decides whether the instruction is LUI which would then use a constant shift amount of 16 bits. Sa is calculated in the DecodeShamtMux circuit which implements a truth table. The truth table has two input bits, bit 2 and

bit 29 from the instruction. The truth table uses these two bits because they are all that are needed to distinguish between LUI, variable shifts, and constant shifts.

9. Imm: The select bit for the multiplexer in the execute stage that chooses to push the immediate or B into the ALU. If Imm is 1, the mux will choose the immediate. If it is 0, then B will go through to the ALU. Imm is computed in DecodeRd/ImmOrB which implements a truth table of bits 26, 27, 28, 29, and 31 as inputs from the instruction. These bits distinguish between instructions that require an immediate or a value B from the register rB.

10. PCsel[2]: The control logic for choosing the correct next PC value in the fetch stage. This is computed from the circuit DecodePCSelect made from a truth table with bits 0, 1, 3, 26, 27, 28, 29, and 31 as input from the instruction. These bits distinguish whether the instructions should choose between PC+4, PCreg, PCrel, and PCabs.

11. Move: The move control bit decides whether the instruction is MOVN or MOVZ which is used in a mux as a select bit in the execution stage. Move is computed from the circuit which takes the opcode and function code from the instruction and decides whether the instruction is MOVN or MOVZ using combinational analysis. This bit is 1 if the instruction is a move and 0 otherwise.

12. Store: This control bit gets sent to the memory stage where it is plugged into the memory RAM. This bit determines whether the data D should be stored in the memory address A. If its 0 it should not store; if its 1 it should store. The data circuit in DecodeStore uses the opcode from the instruction to determine if the instruction is SW or SB which decides if Store should be 0 or 1.

13. Load: This control bit gets sent to the memory stage where it is plugged into the memory RAM. This bit determines whether the instruction should load a value from the memory at the address A to output to a register. It its 0, memory should not load; if its 1 memory should load. The data circuit in DecodeLoad uses the opcode from the instruction to determine if the instruction is LW, LB, or LBU which decides if Load should be 0 or 1.

14. Link: This control bit gets sent to execute where it is plugged into a mux to determine if the output D should be PC + 8 which is used for jump and link instructions. This output is determined from the DecodeLink circuit which takes the opcode from the instruction and determines if the instruction is jal or jalr which would assert the Link output. If it is JAL, the output ”rd=31” is asserted which serves as the select bit to determine if rd should be the decoded value or a constant 31.

15. isWord: This control bit gets sent to the memory stage where it is used in the circuit CalcMSel and also is used as the select bit for a mux which determines whether D from memory should be the full word or only the selected bytes loaded from memory. In CalcMSel, if isWord is 1, MemSel is 1111 which is put into the memory RAM telling it to read the entire word in the memory address. IsWord is calculated in DecodeIsWord which was implemented by taking the opcode and determining if the instruction is lw or sw.

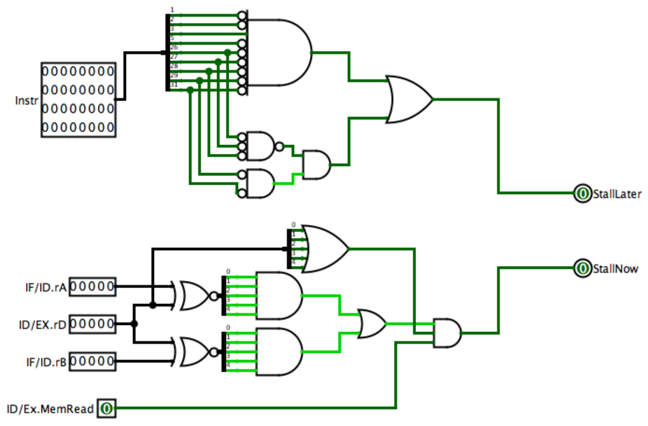
16. UnsignedByte: This control bit is sent to the memory stage where it is used as a select bit for a mux which determines whether to sign extend or zero extend the byte loaded from memory. UnsignedByte is determined from an AND gate with the 0th, 1st, and 2nd bit of the opcode as inputs to determine if the instruction is lbu.

17. SpecialBranch: This control bit is sent to the execute stage. The 0th bit of SpecialBranch determines if the instruction is a special branch which we defined as either bltz or bgez since they required more logic to determine. The 1st bit of SpecialBranch determines if the instruction is bgez or bltz by checking the 16th bit of the instruction. If the 16th bit of the instruction is 0, the instruction is bltz. If its 1, the instruction is bgez.

18. ZeroExtend: This control bit is sent to a mux in the decode which decides if the immediate should be zero-extended or sign-extended. ZeroExtend is determined by the 2nd, 3rd, and 5th bits of the opcode since andi, ori, and xori should be zero extended.

3.2.2 Stalling Unit

The stalling unit is comprised of two parts: Stall Later and Stall Now. These are equivalent to stall due to branch/jump and stall due to load-use hazard, respectively.



The stalling unit

Stall later is asserted if the instruction is a branch or a jump, and is 0 otherwise.

Stall now is asserted if the current instruction poses a load-use hazard, that is, if $IF/ID.rD \neq 0$ and $ID/Ex.MemRead$ and ($(IF/ID.Ra = ID/Ex.Rd)$ or $(IF/ID.Rb = ID/Ex.Rd)$). It is 0 otherwise.

3.2.3 DecodeSLTmux

The SLTmux control logic is computed by the following truth table:

OP5	OP3	OP2	OP1	OP0	F5	F3	F2	F1	F0	SLT	Signed
0	0	0	0	0	1	1	0	1	0	1	1
0	0	0	0	0	1	1	0	1	1	1	0
0	1	0	1	0	x	x	x	x	x	1	1
0	1	0	1	1	x	x	x	x	x	1	0

*Every combination of the input bits not found in this table results in SLT = 0 and Signed = x (an output of 0x)

3.2.4 DecodeWE

The WE control bit is computed by the following truth table:

OP5	OP3	OP2	OP1	OP0	F3	F1	F0	WE
0	0	0	1	0	x	x	x	0
0	0	0	0	0	1	0	0	0
0	0	1	0	0	x	x	x	0
0	0	0	0	1	x	x	x	0
0	0	1	1	0	x	x	x	0
0	0	1	1	1	x	x	x	0
0	0	0	0	1	x	x	x	0
1	1	0	1	1	x	x	x	0
1	1	0	0	0	x	x	x	0

*Every other combination of the input bits not found in this table results in WE = 1.

3.2.5 DecodeALUop

The ALUop control logic is computed from the following truth table:

OP5	OP3	OP2	OP1	OP0	F5	F3	F2	F1	F0	rt0	ALUop3	ALUop2	ALUop1	ALUop
0	0	0	0	0	0	0	x	0	0	x	0	0	0	x
0	0	0	0	0	1	0	0	0	1	x	0	0	1	x
0	1	0	0	1	x	x	x	x	x	x	0	0	1	x
0	0	0	0	0	0	0	x	1	0	x	0	1	0	0
0	0	0	0	0	0	0	x	1	1	x	0	1	0	1
0	1	1	0	0	x	x	x	x	x	x	1	0	0	0
0	0	0	0	0	1	0	1	0	0	x	1	0	0	0
0	1	1	0	1	x	x	x	x	x	x	1	0	1	0
0	0	0	0	0	1	0	1	0	1	x	1	0	1	0
0	1	1	1	0	x	x	x	x	x	x	1	1	0	0
0	0	0	0	0	1	0	1	1	0	x	1	1	0	0
0	0	0	0	0	1	0	1	1	1	x	1	1	1	0
0	0	1	0	0	x	x	x	x	x	x	1	0	0	1
0	0	0	0	0	0	1	0	1	0	x	1	0	0	1
0	0	1	0	1	x	x	x	x	x	x	1	0	1	1
0	0	0	0	0	0	1	0	1	1	x	1	0	1	1
0	0	1	1	1	x	x	x	x	x	x	1	0	1	1
0	0	1	1	0	x	x	x	x	x	x	1	1	1	1
0	0	0	0	0	1	1	0	1	x	x	0	1	1	x
0	1	0	1	x	x	x	x	x	x	x	0	1	1	x
0	0	0	0	1	x	x	x	x	x	1	1	1	0	1
0	0	0	0	1	x	x	x	x	x	0	1	1	1	1
1	x	x	x	x	x	x	x	x	x	x	0	0	1	x

*For every combination of the input bits not found in this table, the ALUop code does not matter.

3.2.6 DecodeShamntmux

The ShamntMux control logic controls which of the possible three shift amounts will be chosen (described in greater detail in the execute stage).

OP3	F2	SA1	SA0
1	x	1	0
0	0	0	0
0	1	0	1

*For every other combination of the input bits not found in this table, it doesnt matter what SA1 and SA0 are.

3.2.7 DecodePCSel

The PCselect control logic will be used to decide which of the four potential program counter values to use next.

OP5	OP3	OP2	OP1	OP0	F3	F1	PC1	PC0
0	0	0	0	0	1	0	0	1
0	1	x	x	x	x	x	0	0
0	0	1	x	x	x	x	1	0
0	0	0	0	1	x	x	1	0
1	x	x	x	x	x	x	0	0
0	0	0	1	x	x	x	1	1

For every other combination of the inputs not shown in this table, PCSel will be 00.

3.2.8 DecodeSpecialBranch

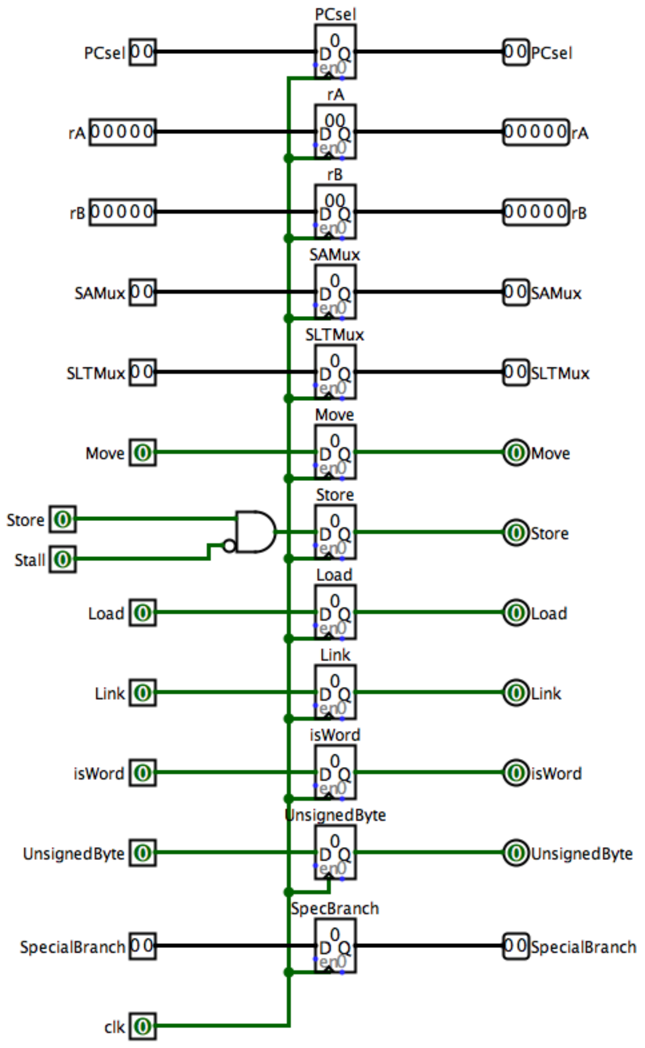
The Special Branch control logic is given by the following truth table:

OP5	OP3	OP2	OP1	OP0	Bit16	SB1	SB0
0	0	0	0	1	0	0	1
0	0	0	0	1	1	1	1

All other combinations result in 00.

3.2.9 Control Registers

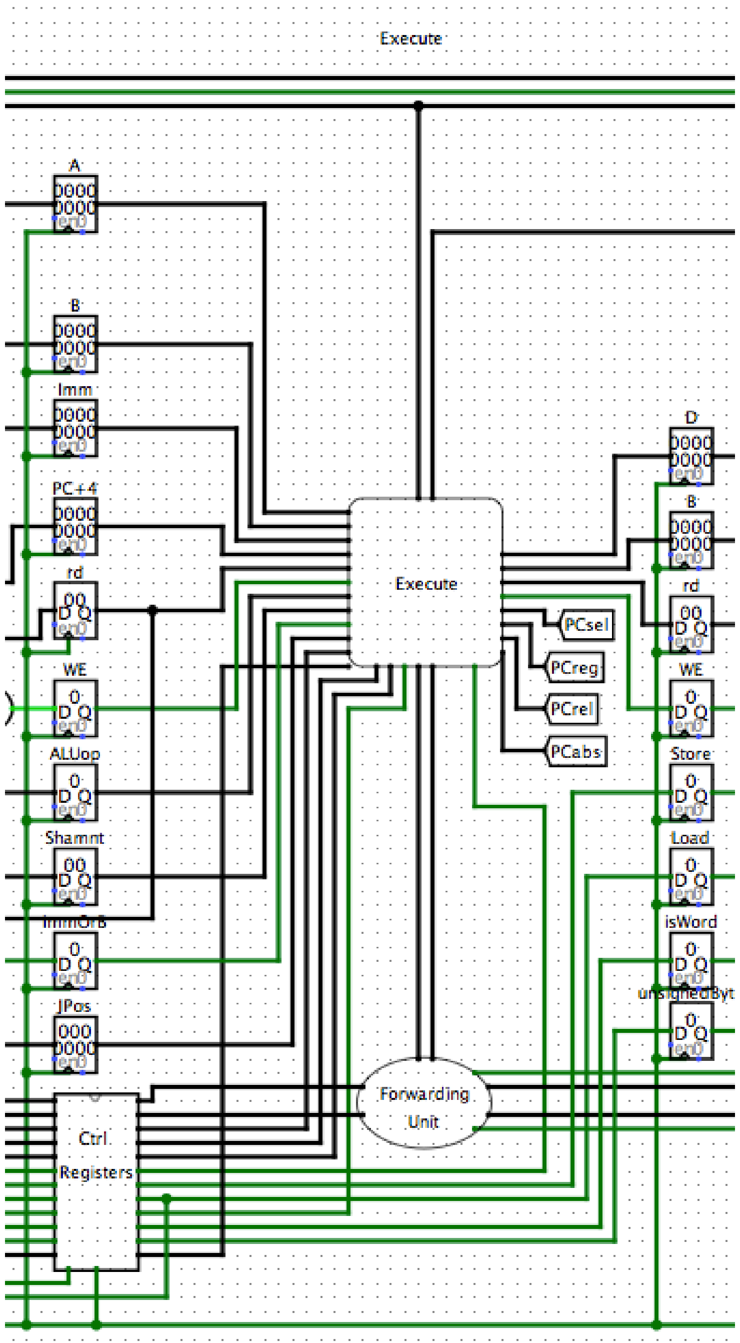
This is a subcircuit containing registers to simplify the top-level circuit.



Control registers in the ID/EX pipeline register

3.3 Execute

The third stage of the five stage pipelined processor is Execution (Ex). The execute stage consists of two subcircuits, a main execute unit, and a forwarding unit.



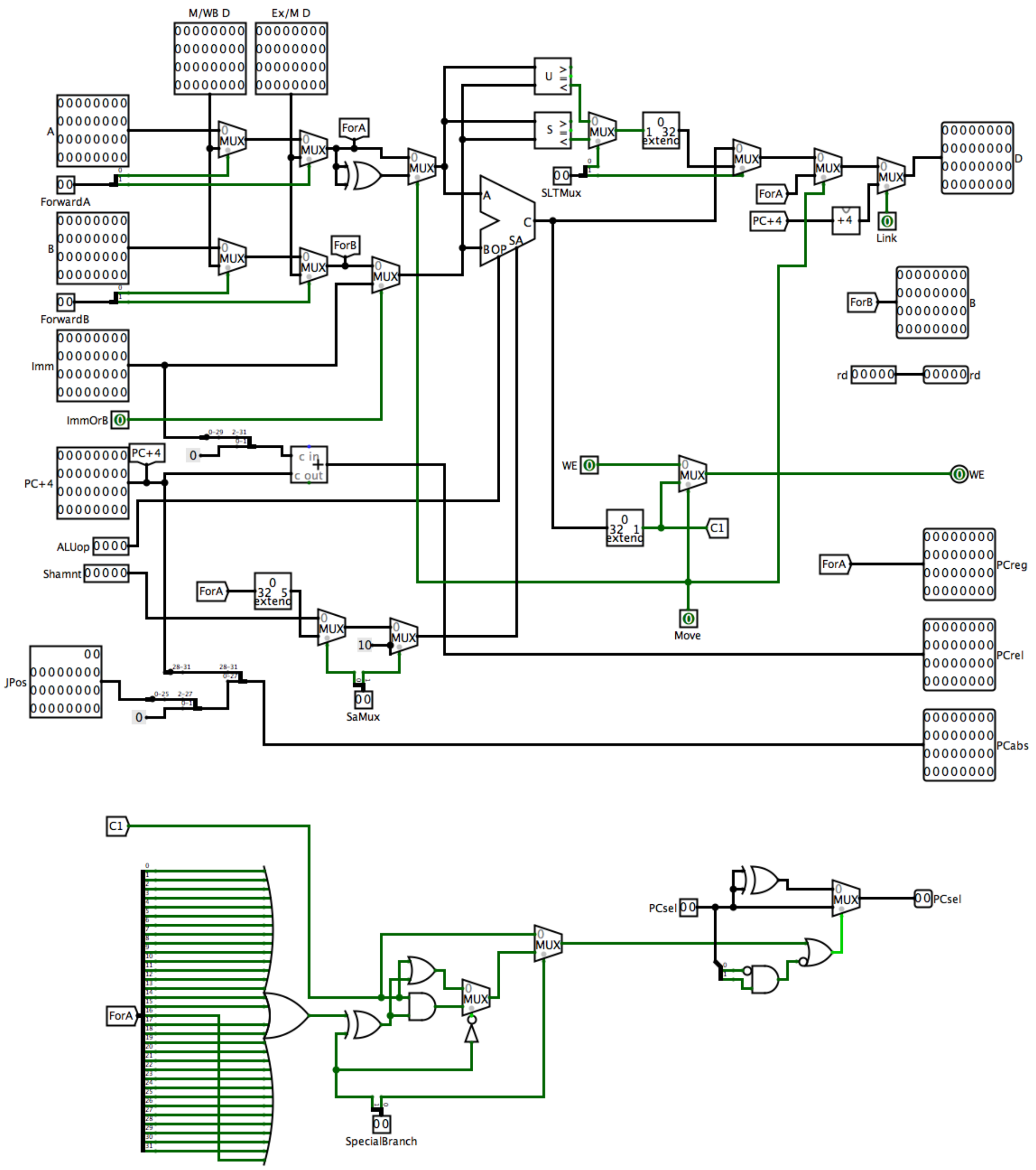
The execute stage

3.3.1 Main execute

This unit calculates the desired result of the operation; it has 14 inputs, they are:

1. A[32]: The value read from the register at rA in the decoding stage.
2. B[32]: The value read from the register at rB in the decoding stage.
3. Imm[32]: The least significant 16 bits of the instruction that have been sign extended to 32 bits.

4. PC+4[32]: The instruction to be executed after the one executing currently in this stage. This data is not used in this project.
5. ALUop[4]: The control logic for the ALU, calculated in the decode stage
6. Shamnt[5]: The shift amount obtained from the instruction.
7. ImmOrB: Whether the immediate or B should go into the ALU
8. JPos[26]: The least significant 26 bits of the instruction that contains the position in code to jump to.
9. rd[5]: The address of the destination register
10. WE: Whether a value should be written to a rd.
11. SaMux[2]: Control logic for the correct shift amount
12. SLTMux[2]: Control logic for selecting the correct comparator and the correct output data
13. ForwardA[2]: Control logic from the forwarding unit
14. ForwardB[2]: Control logic from the forwarding unit
15. M/WB D[32]: This input is the forwarded value from the M/WB register to be used in the current instruction.
16. Ex/M D[32]: This input is the forwarded value from the Ex/M register to be used in the current instruction.
17. Move[1]: Control bit for the mux which decides whether WE should be its original value or the value calculated from the ALU for movn or movz instructions.
18. Link[1]:Control bit for the mux that decides if the value output from Execute should be PC + 8 if Link is asserted.
19. PCSel[2]: Control logic which determines which PC value to choose for the next instruction. In execute, if PCSel is 10 (it is a branch), then the result of the condition is used to determine if PCsel should be changed to 00 (do not branch) or remain 01 (take the branch).
20. SpecialBranch[2]: Additional control logic for evaluating the condition of BLTZ and BGEZ.



The main component of the execute stage

In this stage, both A and B are put through multiplexers that determine if data should be forwarded based on the control logic received. The logic of the forwarding unit will be described in detail below. After forwarding has been handled, each of the inputs go through an additional multiplexor. A goes through a multiplexor with $A \bigoplus A$, which is 0. The output of this depends on whether or not the current instruction is a move. If the instruction is a move, then B must be compared to 0, thus 0 will be pushed into the ALU. B goes through a multiplexor that decides if the forwarded B or the extended immediate value should be pushed into the ALU. This is determined by the ImmOrB control bit.

The correct shift amount must be chosen from the original instruction (an immediate shift), a truncated value of the forwarded A (a variable shift), or a constant 16 (load upper immediate). We realize that a constant is typically bad design, but since the alternative would be overly complicated, a constant was still used. The first multiplexor decides between an immediate or variable shift, and the second between the output of the first or a constant 16 shift. The output of the second multiplexor is input into the ALU along with the ALUop control logic determined in the decode stage.

While the ALU is calculating its result, the two inputs of the ALU are also sent into two separate comparators, one that is unsigned and one that is signed. These comparators execute the SLT family of instructions. A multiplexor is used to determine if the comparison should be signed or unsigned, and another multiplexor decides if the result should be the output of the ALU or of the comparators. The result of that multiplexor and the forwarded A value are chosen based on whether or not the instruction is a move. This is because a move instruction must write the contents of A into the destination, not the result of the condition. Lastly, the result of that multiplexor is pushed through another multiplexor with PC+8. The final result will be decided based on whether or not the instruction is a Link instruction.

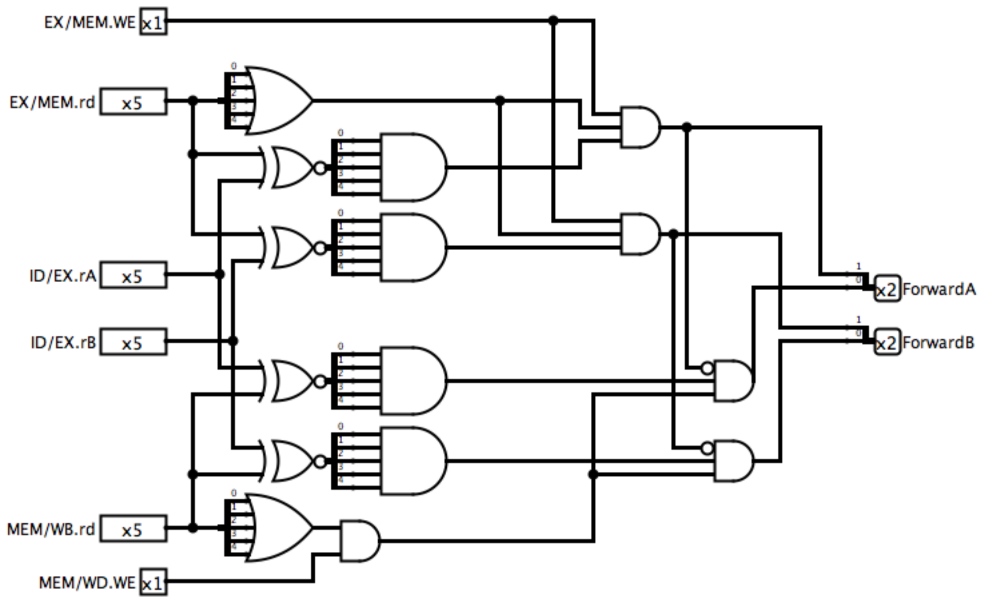
The ALU condition is still used though. In fact, if the instruction is a move, the condition result becomes the write enable bit, since the instruction will only write a value if the condition is true. If the instruction is not a move, the write enable output will be the same as the write enable input. This logic is represented by the multiplexor in the lower right-hand side of the diagram. The condition also goes to the bottom of the circuit for branch calculation.

Forwarded B is output into B (for memory instructions), the rd input is output directly to the rd output. Forwarded A is output into PCreg. PC+4 is added to the immediate value, shifted by 2, to obtain PCrel. The Jump position is also shifted by 2 and concatenated with the upper 4 bits of PC+4 to obtain PCabs.

For branches, PCsel needs additional computation. The condition is tested to determine if PCsel should be set to 00. If the condition is asserted, OR PCsel is not a branch (i.e. not 01), it will select the original PCsel. If not, then PCsel will become 00. For non-special branches, the condition is not evaluated by the ALU alone. These instructions get evaluated in the ALU as $A \leq 0$ and $A > 0$, respectively, and thus another condition involving zero must also be taken into account. To accomplish this, each bit of the forwarded value of A is pushed through an OR gate, the output of which is $A \neq 0$. To evaluate the correct condition, we need to compute either $A \leq 0$ AND $A \neq 0$ for BLTZ or $A \leq 0$ OR $A = 0$ for BGEZ. The most significant bit of SpecialBranch determines whether or not to use the additional computation or the original result, while the least significant bit determines whether or not to negate $A \neq 0$ (based on if it is BLTZ or BGEZ), and it determines if the ALU result should be pushed through an OR or AND gate.

3.3.2 Forwarding Unit

The forwarding unit is a subcircuit of the execute stage that determines whether or not data should be forwarded from the Execute/Memory pipeline register, the Memory/Writeback pipeline register, or neither.



The forwarding unit

The output of the forwarding unit has the following meaning for both A and B (represented arbitrarily as X):

ForwardX1	ForwardX0	Meaning
0	0	Use original value
0	1	Use value from Mem/WB
1	0	Use value from Ex/Mem

The output 11 is not possible due to the fact ForwardX0 requires ForwardX1 to be 0.

Where ForwardX1 = EX/MEM.WE and (EX/MEM.rd != 0) and (EX/MEM.rd = ID/EX.rX).

And ForwardX0 = MEM/WB.WE and (MEM/WB.rd != 0) and (not ForwardX1) and (MEM/WB.rd = ID/EX.rX).

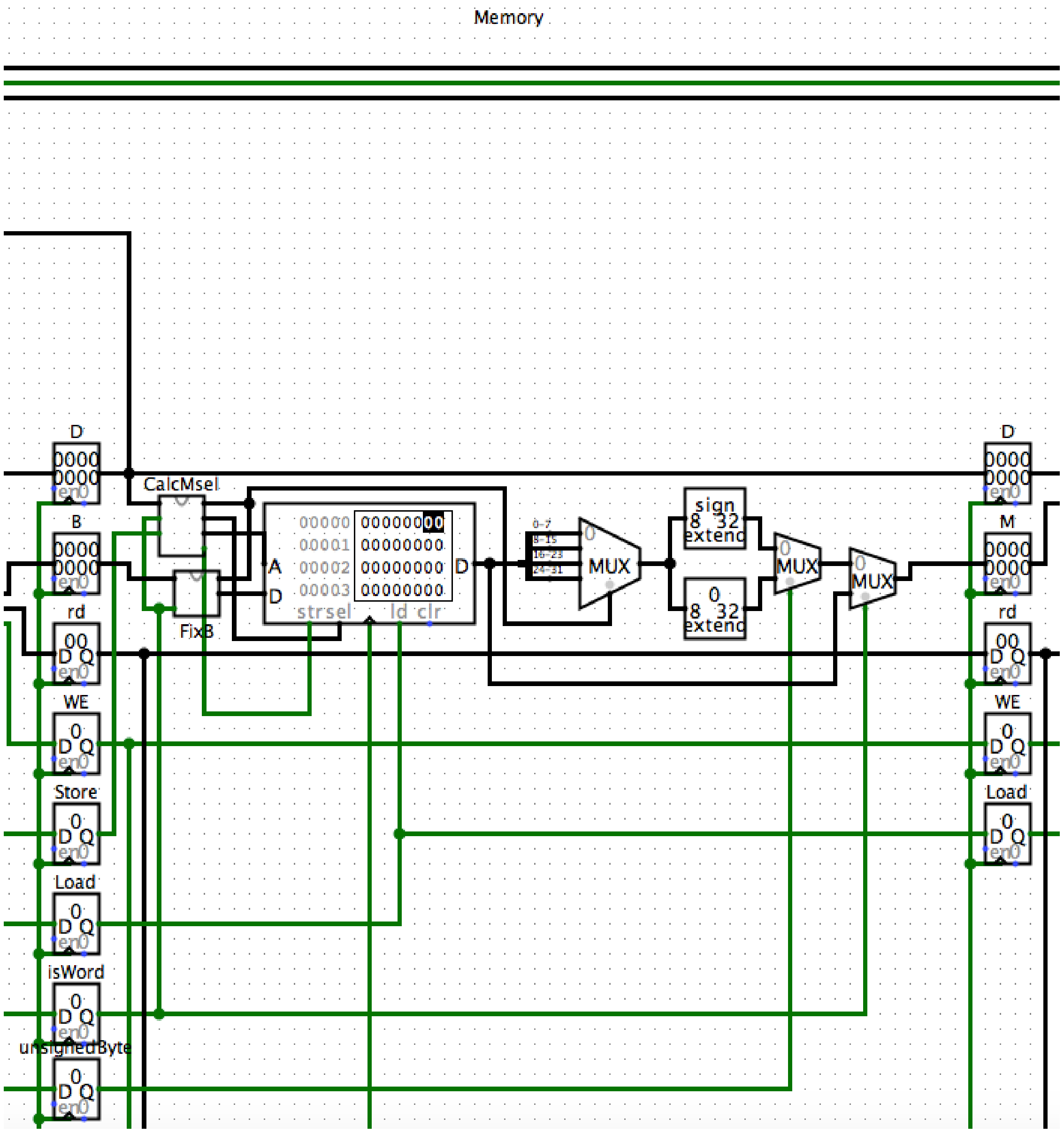
To determine if two 5-bit values are equal, they were pushed through an XNOR gate, which indicates the bits are the same. Each of the 5 bits output from the XNOR gate were pushed through an AND gate to determine if every bit in the values is the same.

To determine if a 5-bit value is not 0, the 5 bits are pushed through an OR gate to test if any bit is nonzero.

3.4 Memory

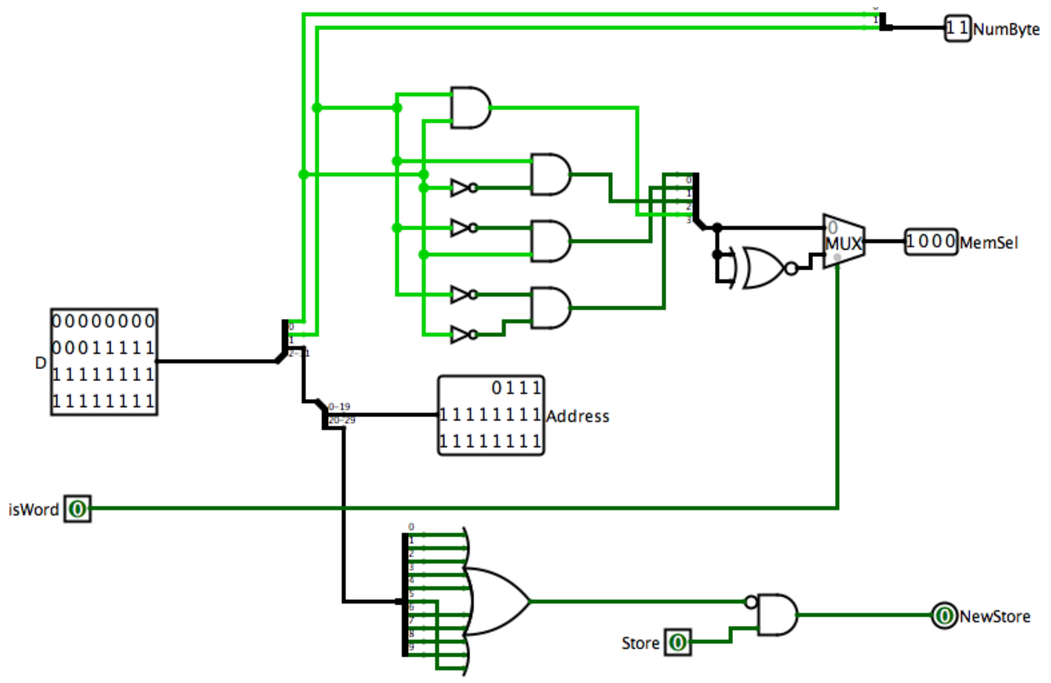
The fourth stage of the five stage pipelined processor is Memory (Mem). In this stage of the pipeline, the processor handles instructions dealing with memory, whether its loading a value from the memory RAM or storing a value into it. A critical piece in this stage is the

MIPS RAM which stores 4 byte words into specific addresses. The RAM has inputs for A, D, str, sel, ld, and clr and 1 output D. A is the address to be read from or written to in memory. D is the data to be possibly stored into memory. Str tells the RAM whether to store or not. Ld tells the RAM whether or not to load. Sel is a 4-bit input that chooses which byte of the word to access.



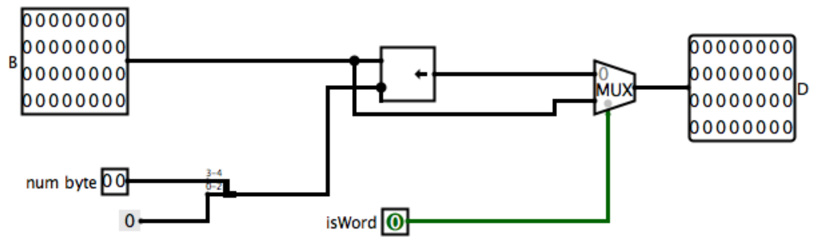
The memory stage

A, sel, and another control signal NumByte are calculated in CalcMSel. The address is determined by dividing D by 4 which is the same as shifting right by 2 which is effectively the same as simply splitting off the 2nd to 31st bit of D and truncating that to 20 bits to conform to the RAMs syntax. NumByte is essentially the remainder of dividing D by 4 since it is the 0th and 1st bits of D. These two bits are transformed into MemSel by the following logic: if NumByte = 00, then the remainder is 0 so MemSel = 0001. If NumByte = 01 then remainder = 1 so MemSel = 0010, and so on up until remainder = 3. The idea behind the implementation of this is based off the explanation given about MemSel(sel for the RAM) in the PA2 instructions. NumByte is used as select bits for the mux which decides which bits of the output D from the RAM to pass through. Also, CalcMSel receives store as an input to decide whether the RAM should really store or not. If any of the top 10 bits of D are 1, then the location of memory is out of range so store should be set to 0.



CalcMSel

NumByte is also used in the subcircuit FixB where it is shifted left logical by 3 and then that value is used as the shift amount for a shift in the circuit. The subcircuit FixB fixes B, where B is the data in rB to be stored in memory, to line it up with the correct byte depending on the value of NumByte. It does this by shifting B to the left by the value of NumByte shifted two to the left. This is then muxed with the original B with the control bit isWord to decide whether to use this fixed B or the original B depending on if the instruction involves a word or a byte. This resulting B is the input D for the MIPS RAM.



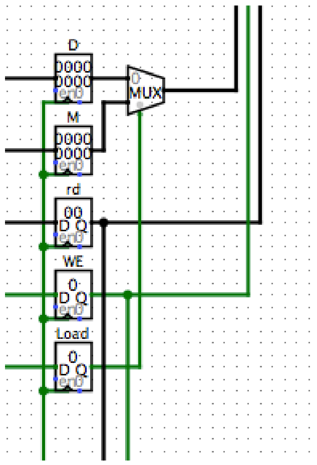
FixB

The output D from the MIPS RAM is split into its four bytes and put through a mux with NumBytes as the control bits to decide which of the four bytes passes through. The byte that passes through is either sign extended or zero extended depending on whether the instruction is unsigned or not. This is then muxed with the original output D with isWord as the select bit. If the instruction is a word instruction it passes through the original D, whereas if the instruction is a byte instruction it uses the split then extended D. This is then stored in the M pipeline register.

Lastly, Rd, WE, and Load are passed through to the Write Back stage.

3.5 Writeback

The fifth and final stage of the five stage pipelined processor is Writeback. In this stage, the output D is muxed with the value M which is chosen from the control bit "Load," a.k.a. "MemWrite." This value is sent back to the register file along with control logic which contains rd, the address of the register to write back into, and WE (Write Enable) which will decide whether a value can be written into a register or not.



The write back stage

4 Testing

To test the final pipelined processor, a program was written to test every instruction that can be handled by the processor. For each instruction in tables A and B, an average case was tested, as well as all possible hazard cases. If the instruction read from multiple inputs, both inputs were tested for hazards. Once the program was written, it was executed one instruction at a time, and the result was compared against a manual computation of the result.

Additionally, the processor was tested with three different versions of the hailstone algorithm: a recursive one, an iterative one, and a memoization one.