Intro to Computer Architecture and Organization

Project 4: Pipelined Control Unit

By Drew Lenz and Andrew Schmidlin

4/4/2017

Contents:

Objectives

The principle objective of this project is to demonstrate that one understands the basic operation and construction of a pipelined control unit for a simple instruction set.

Specifications

- A word size of 16-bits
- Byte addressable memory
- 64K byte main memory
- A 16-bit program status word (PSW) with status bits. The first two bits are the condition code bits Z and N; these bits are conditionally controlled and denote the results of comparisons of the instruction result to the values zero (Z=1-equality to zero; N=1- less than zero). In addition, a third bit denotes execution in either of privileged or user mode (some operations are prohibited in user mode). The remaining bits control operations/constraints in the memory space that are not addressed in this project.
- 16 instructions, 2 of which can be executed only in privileged mode. Attempting to execute these 2 instructions in user mode will cause a program check violation.
- 8 16-bit General Purpose Registers (Reg). The general purpose register allows for three read and two write operations to occur at the same time. Register 0 (Reg[0]) always holds the value 0 no matter what value is assigned to it.
- A 16-bit program counter (PC) which is also Reg[7]
- A 16-bit count-down timer that causes a timer interrupt when it reaches zero provided the machine is executing in user mode. Timer interrupts are ignored when executing in privileged mode.
- 2’s complement number representation

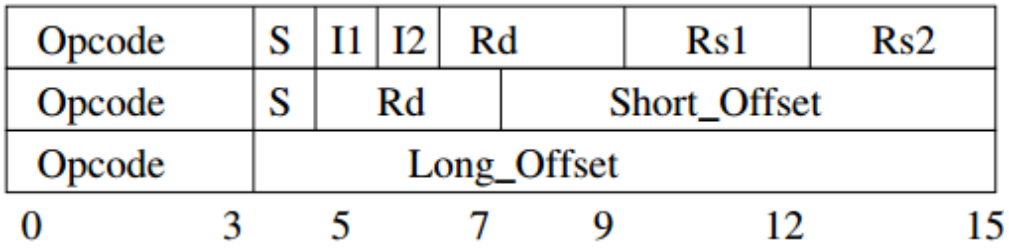

Figure 1: Instruction Format

Name	Opcode	Description
ADD	0	GPR[Rd] = OP1 + OP2
SUB	1	GPR[Rd] = OP1 - OP2
AND	2	GPR[Rd] = OP1 **and** OP2
SHL	3	GPR[Rd] = shift_left(OP1) by $OP2_{3-0}$
SHRA	4	GPR[Rd] = shift_right(OP1) by $OP2_{3-0}$
OR	5	GPR[Rd] = OP1 **or** OP2
NOT	6	GPR[Rd] = **not** MM[PC + Short_Offset]
LD	7	GPR[Rd] = MM[PC + Short_Offset]
ST	8	MM[PC + Short_Offset] = GPR[Rd]
BRN	9	if CC.N then PC = PC + Long_Offset
BRZ	10	if CC.Z then PC = PC + Long_Offset
BR	11	PC = PC + Long_Offset
JSR	12	GPR[Rd] = PC; PC = PC + Short_Offset
RTS	13	PC = GPR[Rd] + Short_Offset
CLK	14	Set timer to MM[PC + Long_Offset]
LPSW	15	PSW = MM[PC + Long_Offset]

Figure 2: Instruction Opcodes and Descriptions

Components

ALU

The ALU consists of multiple smaller circuits that all go into a multiplexer that allows the control unit to decide which function it would like the result of. The ALU is capable of addition and subtraction via a carry lookahead adder/subtractor, logical AND, OR, NOT, Arithmetic Shift Right, and Logical Shift Left. The carry lookahead adder differentiates between addition and subtraction via 2's complement of the second input.

GPR

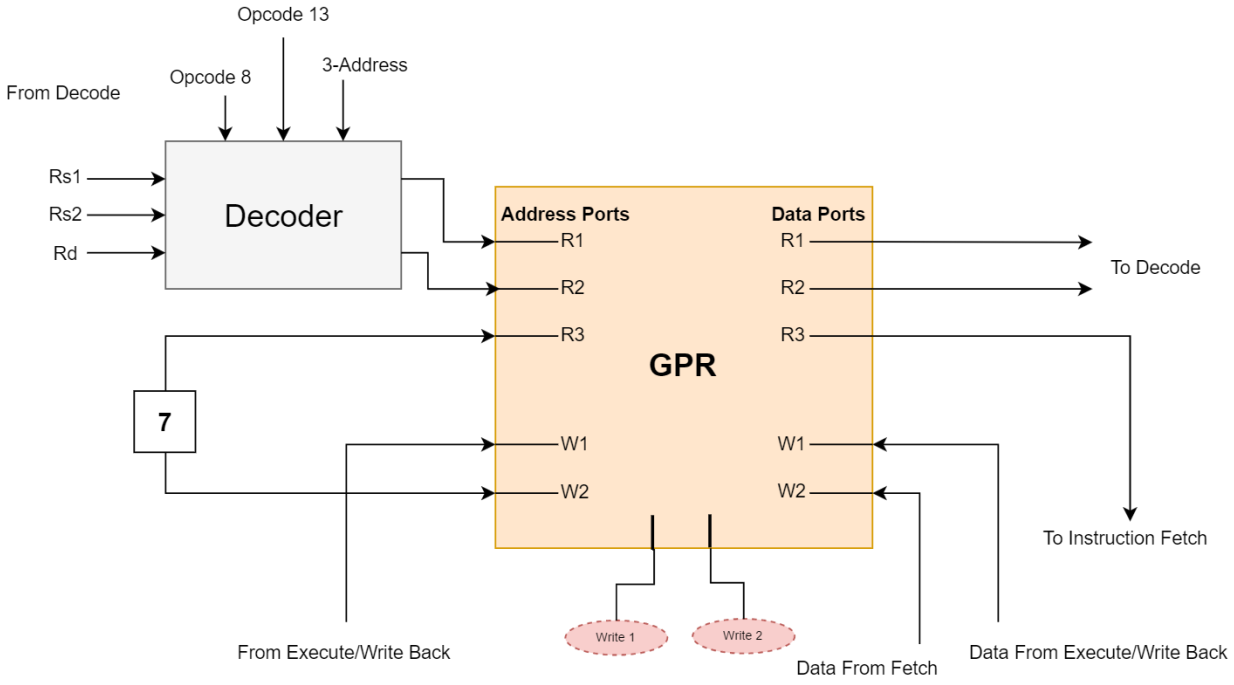

Figure 3: General Purpose Register

This processor has a general purpose register than consists of eight 16-bit registers. The specifications of this project stated that the GPR could have up to three read ports and 2 write ports, each of these can be used at the same time. The PC value is stored in register 7. To simplify accessing the PC, a value of 7 is hardwired to the R3 and W2 address ports. This allows one to access the data in the PC at the R3 and W2 data ports without having to pass an address of 7 for each access. The Write 1 input comes from the end of the Execute stage Figure 9. The Write 2 input comes from the Fetch stage and can be seen coming from the demultiplexer in Figure 6.

A decoder is used to access the R1 and R2 ports during the Decode stage of the pipeline. The GPR decoder looks at the instruction in Decode's RS1, RS2, and Rd bits. It will send up to two of these addresses to the R1 or R2 data ports based on which opcode indicator is on. In the

case that 3-Address is high, RS1 will go to the R1 port and RS2 will go to the R2 port. In the case that Opcode 8 is high, Rd will go to R2 and R1 is not used. In the case that Opcode 13 is high, Rd will go to port R1 and R2 is not used.

Main Memory

This system contains a main memory unit that is divided into an instruction part and a data part. This allows for instructions to be read from the main memory while data is read or written to the main memory. The instruction memory does not use MAR or MDR registers because it can be connected point-to-point style with an address and a data register. The data main memory uses both an MAR register and an MDR register.

ROM: Read Only Memory

The ROM is used to store constants that are used to point to specific points in memory used when reassigning and saving the PSW and PC when either a Program Check Violation or a Timeout exception occurs.

PSW: Program Status Word

The PSW is 16-bits including status bits. The first two bits are the conditionally controlled bits Z and N. They denote the results of comparisons of the instruction result to zero. Z=1 equality to zero. N=1 negative. The third bit is the P bit which controls whether execution is done in either privileged mode or user mode. Some operations cannot be executed while operating in user mode. Bits 4-16 control operations/constraints in memory space and will not be addressed in this report.

Clock

The clock outputs a pulse that causes states to run and control signals to be sent.

Control Unit

Overview

The control unit in this processor consists of a very simple state machine and a counter that sends control signals. Because this is a pipelined datapath, there are less states than one would see in a non-pipelined processor such as the one completed for Project 3. The 2 states of this state machine are Stages Enabled and Stage Move. There are also separate states for Timeout and Program Check Violation exceptions, but these are not part of the overall operation of the state machine. They will be described in detail in the following sections.

Finite State Automata

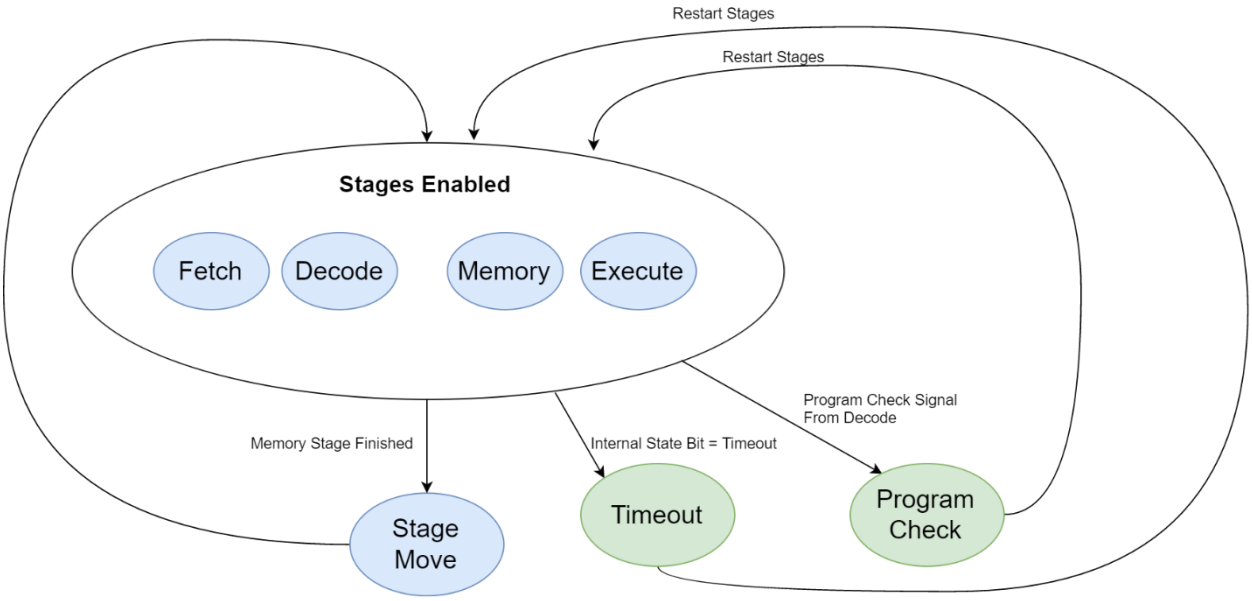

Figure 4: Finite State Machine

The state machine for this processor's control unit is shown above in Figure 4. The two main states are Stages Enabled and Stage Move. It is by cycling between these two states that instructions are executed and moved down the pipeline. The Stages Enabled is where all control signals are sent to the Fetch, Decode, Memory, and Execute stages of the pipeline. When the control unit receives the Memory Stage Finished signal, it moves to the Stage Move state. It is during this state that the instructions and any operands or data are moved to the next buffer in between stages.

The state machine also shows the Timeout and Program Check states. These states are where exceptions are handled. All operations within the pipeline are stopped to handle exceptions. When the exception is complete, the Stages Enabled state is re-entered and the instructions restart their operations. In the case of a program check violation, the valid bit for the problematic instruction has been set to 0 so the instruction can be flushed from the

pipeline. In the case of a timeout exception, the instruction in the pipeline simply resume operation. This is because the timeout exception is considered to be an imprecise exception.

Control Signals

Stages Enabled

The control signals in this control unit are pretty simple. The signals for Fetch, Decode, MEM, and Execute are all sent at the same time. To make the explanation of these control signals more understandable, they will be described separately first.

Fetch Control Signals

1. Read Instruction MM, Temp PC reg_{in}, Valid Bit_{in}
2. GPR Write 2 Port Enable, IR_{in}

Decode Control Signals

1.
2. $OP1_{in}$, $OP2_{in}$

MEM Control Signals

1. Address, Use MM
2. Bus Activate, OP_{in}, Address, Use MM
3. Bus Activate, OP_{in}

Execute Control Signals

1. $Result_{in}$
2. Result WB

However, the signals described above are all sent together so that the signals sent for each clock cycle in the Stages Enabled state are as follows.

Stages Enabled Control Signals

1. Read Instruction MM, Temp PC reg_{in}, Valid Bit_{in}, IND, Use MM, $Result_{in}$
2. GPR Write 2 Port Enable, IR_{in}, $OP1_{in}$, $OP2_{in}$, Bus Activate, OP_{in}, IND, Use MM, Result WB
3. Bus Activate, OP_{in}

Stage Move

For the Stage Move state, a simple stage move control signal is sent to enable the buffers separating each stage. This allows for instructions and other data to be safely passed from one stage to another once all the enabled stages have finished executing.

Stage Move Control Signals

1. Stage Move (IR_{in})

Exceptions

There are two possible exceptions that can occur in this processor: program check violations (traps) and timeout exceptions. These exceptions occur infrequently, but they need to be handled to ensure proper behavior in the processor.

Program Check Violation

A program check violation is found in the decode stage of the pipeline. The violation is triggered when the P bit of the program status word is 0 and the user is trying to execute privileged instructions with opcodes 14 or 15. To resolve a program check violation, the PC and PSW are swapped. If this violation occurs, an indicator bit is turned on to notify the control unit that the program check control signals need to be sent out. This bit will also reset the valid bit of the instruction that caused the violation. This will allow for the instruction to be flushed from the pipeline. The control signals sent during this stage are as follows.

Program Check Control Signals

Note: The main memory referred to by these control signals is the data main memory.

1. $ROM[0]_{out}$ to MAR, MAR_{in}, PSW_{out} to MDR, MDR_{in}, Write_MM
2. $ROM[2]_{out}$ to MAR, MAR_{in}, PC_{out} to MDR, MDR_{in}, Write_MM
3. $ROM[4]_{out}$ to MAR, MAR_{in}, Read_MM
4. MDR_{out} to PSW, PSW_{in}, $ROM[6]_{out}$ to MAR, MAR_{in}, Read_MM
5. MDR_{out} to PC, PC_{in}
6. Restart Stages

After the PC and PSW have been swapped, a signal is sent within the control unit to return to the Stages Enabled state. At this point, a new instruction is fetched at the address indicated by the new PC value and the instructions in the pipeline continue their execution. The instruction that caused the program check violation does not have a valid data bit and is flushed from the pipeline.

Timeout Exception

A timeout exception is found in the execute stage of the pipeline. A timeout occurs when the system's countdown timer reaches 0 while executing in user mode (the P bit of the program status word is 0). This timeout state exists to interrupt the control unit and modify the main memory, program counter, and program status word. Like for the program check violation, if these conditions are met, an indicator bit is turned on to notify the control unit that the timeout control signals need to be sent out.

Timeout Exception Control Signals

Note: The main memory referred to by these control signals is the data main memory.

1. $ROM[8]_{out}$ to MAR, MAR_{in}, PSW_{out} to MDR, MDR_{in}, Write_MM
2. $ROM[10]_{out}$ to MAR, MAR_{in}, PC_{out} to MDR, MDR_{in}, Write_MM
3. $ROM[12]_{out}$ to MAR, MAR_{in}, Read_MM
4. MDR_{out} to PSW, PSW_{in}, $ROM[14]_{out}$ to MAR, MAR_{in}, Read_MM
5. MDR_{out} to PC, PC_{in}
6. Restart Stages

After the timeout exception has been handled, a signal is sent within the control unit to return to the Stages Enabled state. At this point an instruction is fetched at the address indicated by the new PC value. The instructions that were in the pipeline continue their execution. The instruction that caused the timeout exception is flushed from the pipeline because it does not have a valid data bit.

Data-Path

Overview

An overview of the pipelined datapath implemented in this processor is shown in Figure 5 below. It consists of Fetch, Decode, Memory, and Execute/Writeback stages. These stages are separated by IR buffers that contain each instruction and its valid bit as well as any operands or results that need to progress to the next stage of the pipeline. The Stage Move control signal is used move instructions and results from one stage to the next.

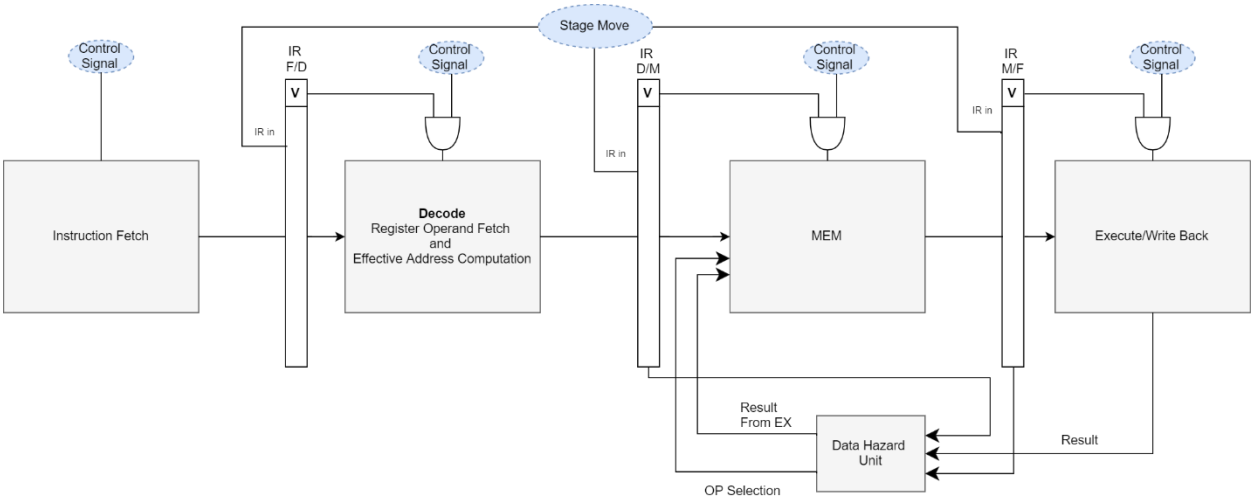

Figure 5: Pipelined Data Path Overview

Fetch (always 2 clock cycles)

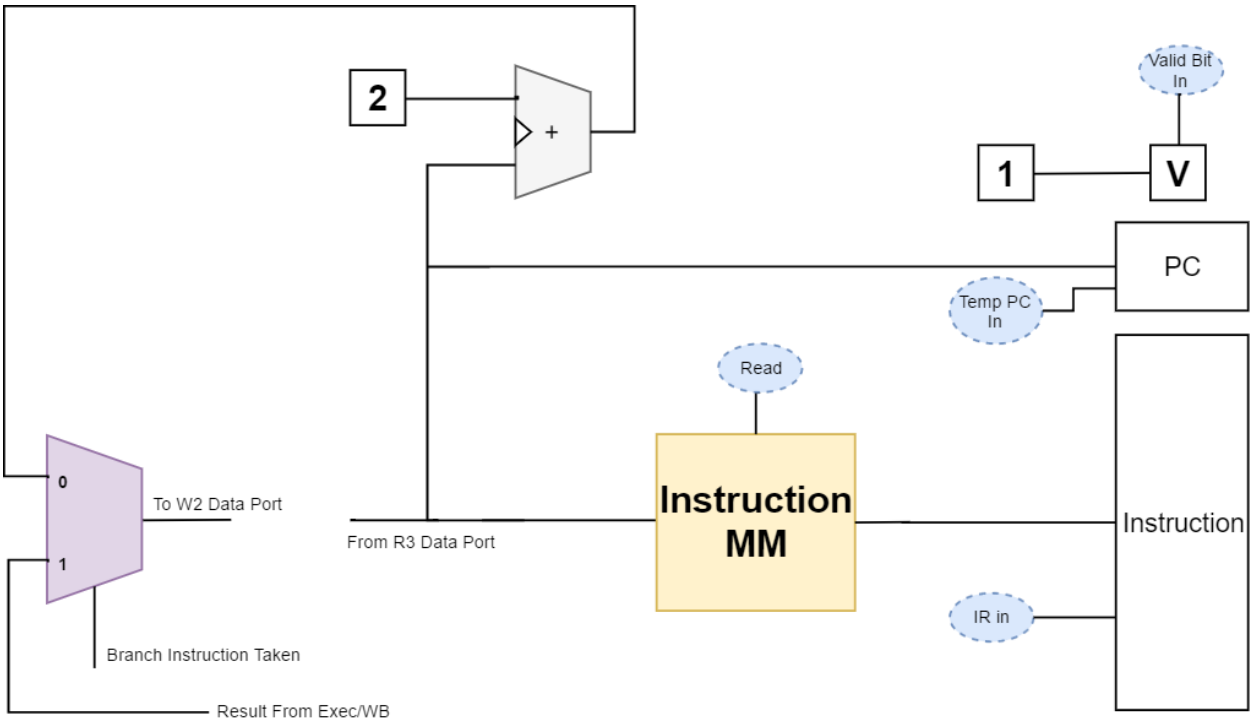

Figure 6: Fetch Stage

The Fetch stage is where an instruction indicated by the program counter is read from the instruction main memory and stored in an instruction register. The wire labeled 'From R3 Data Port' in Figure 6 carries the PC value from the GPR shown in Figure 3 as wire 'R3' → 'To Instruction Fetch' to the ALU and the PC buffer register. The 'Temp PC In' control signal will store the PC value in a temporary PC register. The ALU will automatically increment the PC value by adding it to the hard-wired 2 value. This incremented value is sent to the 0 port of a multiplexer. The multiplexer also takes in a conditional branch result from the ALU in the Execute stage. The 'Branch Instruction Taken' wire also comes from the Execute Stage and it allows for a branched PC value to be safely written to the PC instead of using the incremented PC value. The result is then written to the W2 Data Port in the GPR. Lastly, the valid bit is also initialized as 1 at this point.

Decode

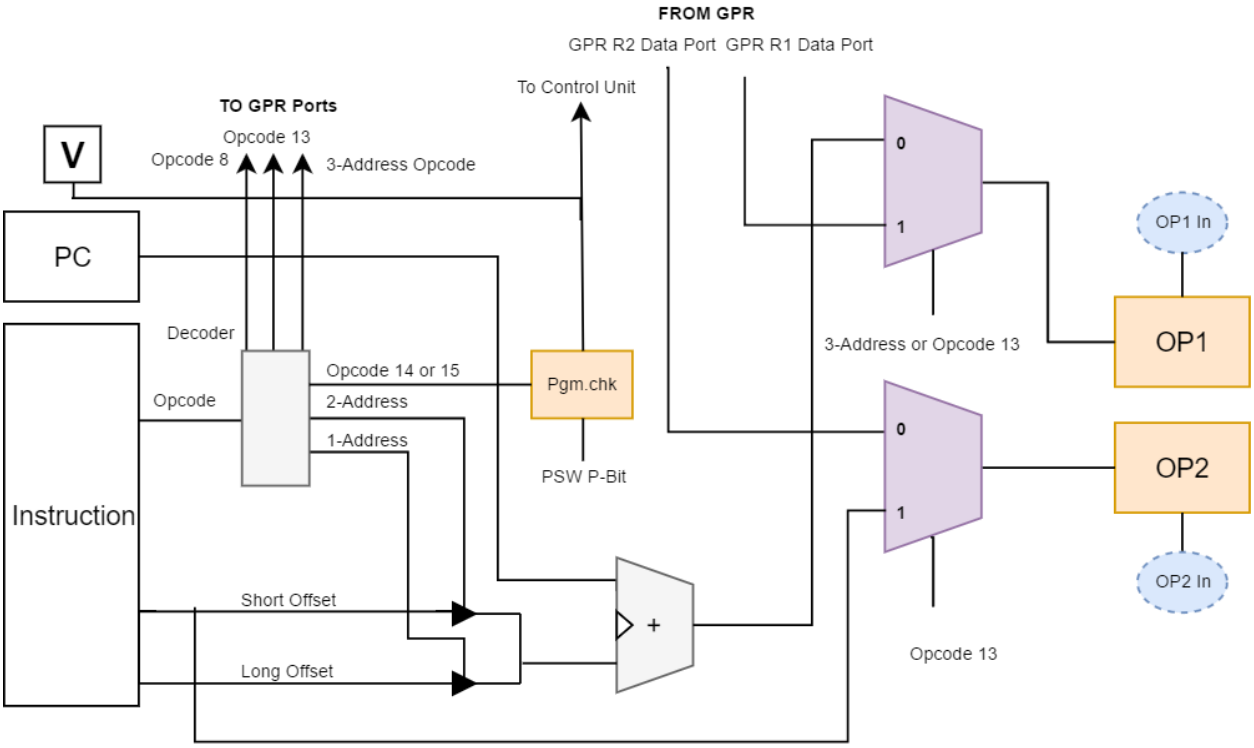

Figure 7: Decode Stage

The Decode stage handles the fetching of operands from the GPR as well as the calculating of effective addresses. The decoder connected to the Instruction register decodes opcodes 8, 13 and all 3-Address Opcodes. In the presence of any of these opcodes, the decoder sends a signal to the GPR decoder shown in Figure 3. The decoder also decodes for opcodes 14 or 15 and sends them to the program check register which takes in the P-bit of the PSW to decides if the instruction has permission to execute. If the user does not have the necessary privileges, a program check violation signal is sent to the control unit. This signal is also sent to reset the valid bit of this instruction to 0. This makes sure that the instruction is viewed as a no-op instruction to be flushed from the pipeline. The decoder also detects if the instruction is a 2-Address or 1-Address instruction. These signals enable buffers for sign-extended Short Offset and Long Offset which will be added to the PC value via an ALU. The result of that addition goes into the low input of the top multiplexer and GPR R1 Data Port goes into the high input. The 3-Address or Opcode 13 signal decide which input to take. The lower multiplexer takes in short offset to the high input and GPR R2 Data Port into the low input. Opcode 13 decides which input to take. The results of the multiplexers go into OP1 and OP2 respectively with the control signals $OP1_{in}$ and $OP2_{in}$.

Memory (1-3 clock cycles)

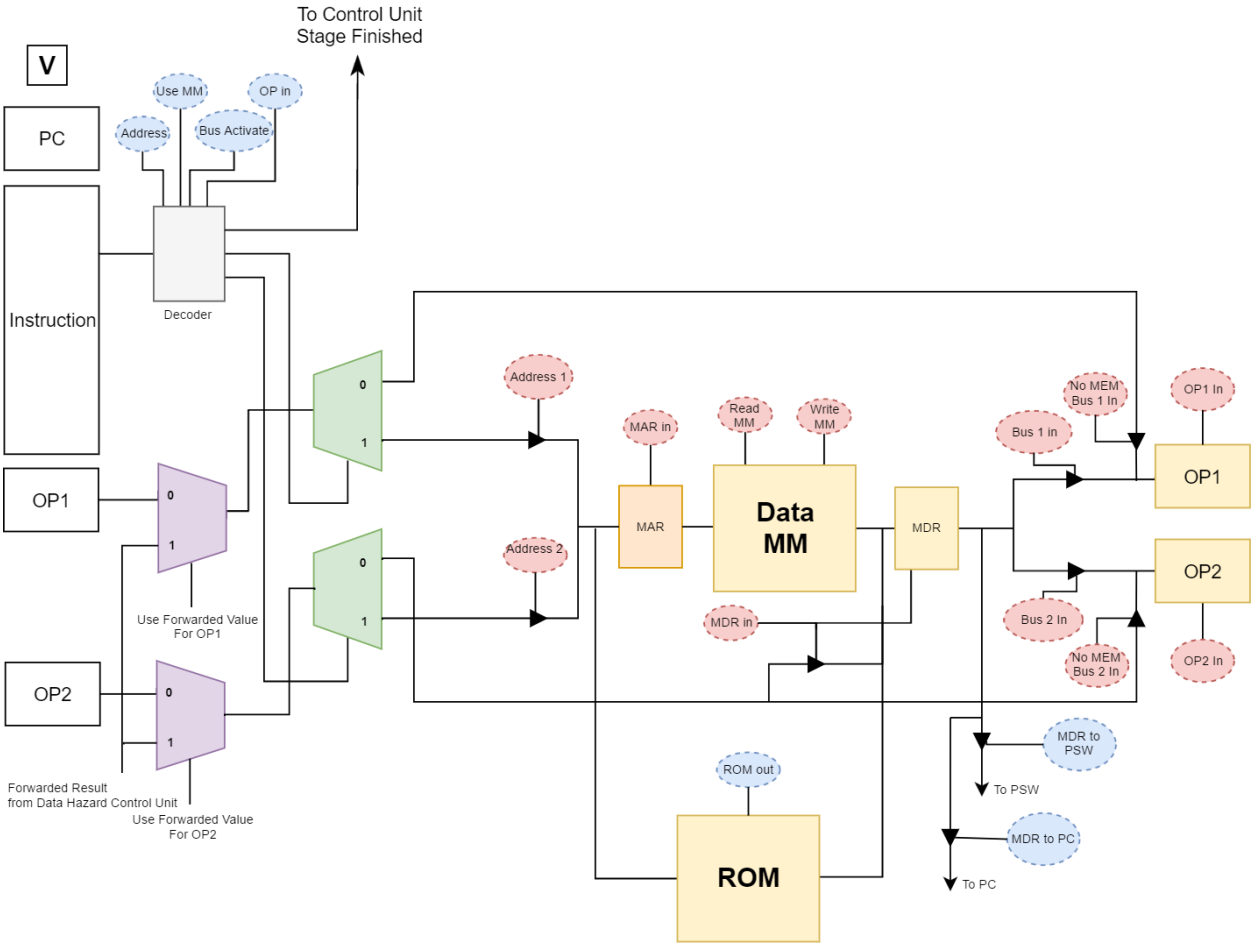

Figure 8: MEM Stage

The MEM stage is where all operations that access the data portion of the main memory occur. Examples of a pipelined control unit shown in the course textbook place the Execution stage before the MEM stage. However, following this example would introduce a large structural hazard to this pipeline solution because of indirect operands. These operand values need to be accessed prior to the Execution stage. Since the data main memory can only perform one operation at a time, placing the MEM stage after Execute would require stalling, which will drastically increase the time it takes this processor to complete an instruction. Placing the MEM stage before Execution allows for successful handling of all read/write instructions while supporting indirect operands.

A representation of the MEM stage is shown above in Figure 8. The control signals represented by blue ovals come directly from the processor's control unit. These signals are sent to a decoder which is also passed the instruction's opcode and I bits as inputs. This decoder box uses combinational logic to determine which red sub-signals should be activated in the stage on each clock cycle. The decoder box can also select if the values in the operand

registers should be passed to the memory unit or if they should be loaded into the final operand registers as-is. During the last batch of internal sub-signals, a Stage Finished signal is sent back to the control unit. This signal is important because it allows for variation in the number of clock cycles needed to complete the MEM stage. The MEM stage will use two clock cycles to complete most instructions, but instructions with two indirect operands will take three. This is because the memory unit needs to be accessed twice during the stage instead of only once. Sending the Stage Finished signal at the right time allows movement to the next stage without having to wait for an unnecessary third clock cycle.

The first type of instruction the MEM decoder handles is a 3-address instruction with no indirect operands. The decoder first determines if the instruction is a 3-address instruction with an opcode of 0-5. If this is true, the decoder will use the I bits of the instruction to determine if either value in the operand is an address. In cases when both I bits are 0, the decoder will output 0 to both selection demultiplexers which are shown in green in Figure 8 above. This allows the values to be passed directly into the operand registers at the end of the stage. No internal sub-signals are sent during the first clock cycle. On the second clock cycle, the Bus Activate signal is decoded such that both No MEM Bus In sub-signals are sent. The OP_{in} signal is decoded so that values are written to OP1 and OP2 registers. Lastly, the stage finished signal is sent to the control unit. The second Address and Use MM control signals are ignored by the decoder for this kind of instruction. The internal MEM sub-signals for a 3-address instruction are as follows.

3-Address MEM Sub-signals

1.
2. No MEM Bus 1_{in}, No MEM Bus 2_{in}, $OP1_{in}$, $OP2_{in}$, Stage Finished

The next type of instruction the MEM decoder handles is a 3-address instruction with only one indirect operand. The decoder determines if the instruction has a 3-address opcode similar to the process described above. Using the I bits, the decoder determines which operand register contains an address for an indirect operand. The green selection demultiplexer corresponding to the active I bit is given a selection bit of 1, allowing the address to be passed to the MAR register. On the first clock cycle, the Address control signal is decoded to an Address 1 or Address 2 sub-signal and the Use MM signal is decoded to MAR_{in} and Read MM sub-signals. On the second clock cycle, the Bus Activate control signal is decoded so that the value from the MDR is placed on the correct operand bus while the other value is taken directly from the operand. Also, the Stage Finished signal is sent out at this time. The second Address and Use MM control signals are ignored by the decoder for this kind of instruction. The internal MEM sub-signals for these instructions are as follows.

3-Address MEM Sub-Signals, I1 = 1 and I2 = 0

1. Address 1, MAR_{in}, Read MM
2. Bus 1_{in}, $OP1_{in}$, No MEM Bus 2_{in}, $OP2_{in}$, Stage Finished

3-Address MEM Sub-Signals, I1 = 0 and I2 = 1

1. Address 2, MAR_{in}, Read MM,
2. No MEM Bus 1_{in}, $OP1_{in}$, Bus 2_{in}, $OP2_{in}$, Stage Finished

The next type of instruction the MEM decoder handles is a 3-address instruction with both operands accessed indirectly. As before the decoder determines if the opcode represents a 3-address instruction and sends selection bits of 1 to both green selection demultiplexers. At this point, reading the addresses needs to be staggered so that data conflicts don't occur when putting both values on a bus or in the MAR at the same time. The sub-signals used to read address 1 are sent and are then followed by the sub-signals used to read address 2. The internal MEM sub-signals for this type of instruction are as follows.

3-Address MEM Sub-Signals, I1 = 1 and I2 = 1

1. Address 1, MAR_{in}, Read MM
2. Bus 1_{in}, $OP1_{in}$, Address 2, MAR_{in}, Read MM
3. Bus 2_{in}, $OP2_{in}$, Stage Finished

The next kind of instruction that the MEM decoder handles is a 2-address or 1-address instruction that requires a value to be read from main memory and stored in a GPR or set to the PSW or timer. These instructions are indicated by opcodes 6, 7, 14, and 15. The decoder determines the presence of one of these opcodes and sends a 1 to the OP1 selector demultiplexer to indicate that there is an address in OP1 that should be read. It is safe to assume that the needed address is stored in the OP1 register because that is taken care of during the decode stage. For the first clock cycle, the decoder will turn the Address control signal into an Address 1 sub-signal and will turn the Use MM signal into MAR_{in} and Read MM sub-signals. On the second clock cycle, the data read from the memory is placed in the OP1 register. To do this, the decoder decodes the Bus Activate control signal to a Bus 1_{in} sub-signal and it decodes the OP_{in} signal to an $OP1_{in}$ sub-signal. The second Address and Use MM control signals are ignored by the decoder for this kind of instruction. The internal MEM sub-signals for this type of instruction are as follows.

1-Address or 2-Address Read MEM Sub-Signals

1. Address 1, MAR_{in}, Read MM
2. Bus 1_{in}, $OP1_{in}$, Stage Finished

The last kind of instruction that the MEM decoder handles is a write instruction, which is indicated by opcode 8. The address being written to has been stored in OP1 during the decode and the data being written has been stored in OP2. OP2 is wired in such a way that a tri-state buffer can allow the value in OP2 to be stored in the MDR for this instruction. If opcode 8 is detected, the decoder decodes the Address control signal send during the second clock cycle to an Address 1 sub-signal. The Use MM control signal send during the second clock cycle is used to send MAR_{in}, MDR_{in}, and Write MM sub-signals. After this value has been written, no

additional signals are necessary aside from a Stage Finished signal. These signals are sent on the second clock cycle instead of the first to help avoid a control hazard that could occur if a branch instruction came before this instruction.

Write MEM Sub-Signals

1.
2. Address 1, MAR_{in}, MDR_{in}, Write MM, Stage Finished

Any of the remaining instructions (opcodes 9, 10, 11, 12, and 13) do not require use of the memory unit. Therefore, the values in OP1 and OP2 are simply passed on as if the instruction was a 3-address instruction with no indirect operands. The control signals for the remaining instructions are as follows.

No Memory MEM Sub-Signals

1.
2. No MEM Bus 1_{in}, No MEM Bus 2_{in}, $OP1_{in}$, $OP2_{in}$, Stage Finished

In addition to handling all the memory related control signals for each instruction in this processor, the MEM stage is wired in a way that allows the ROM to access the data main memory. This is necessary for handling program check violations and timeout exceptions. During these exceptions, the instruction in the register is not used because the control signals come directly from the control unit.

Execute/Writeback (always 2 clock cycles)

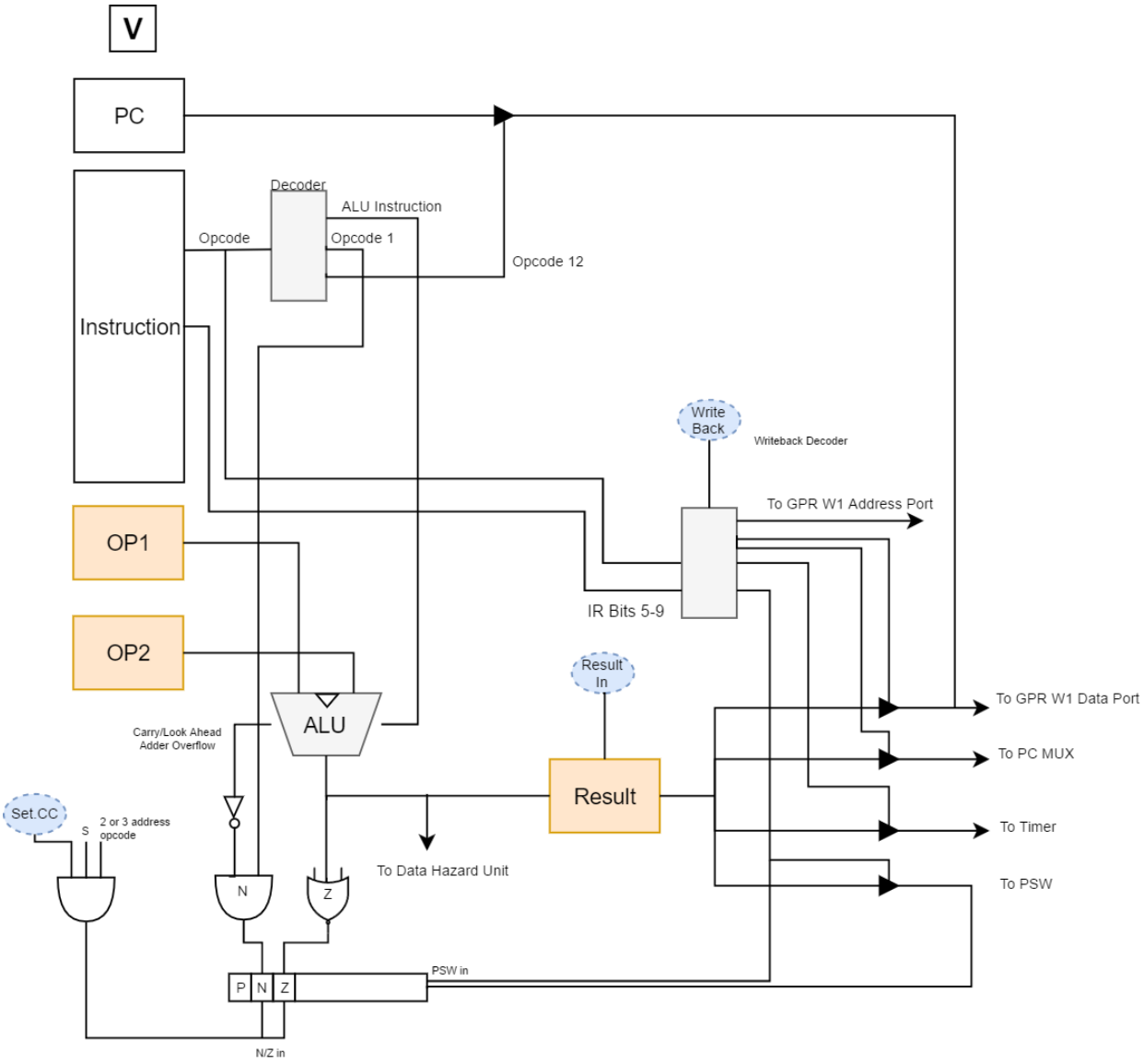

Figure 9: Execute Stage

The Execute stage is where all ALU operations occur to execute an instruction. Also, results are written back to the appropriate places in this stage. A decoder takes the opcode from the instruction register and outputs the appropriate ALU Instruction signal, the Opcode 1 signal, and the Opcode 12 signal. The ALU signal goes to the ALU and tells it what operation (add, subtract, shift, etc.) to output. In cases where an ALU operation is not needed, the ALU is told to do nothing, allowing the value in OP1 to pass into the Result register. Opcode 1 is subtraction. This is the only operation which can yield a negative result. Therefore, Opcode 1 is used to set the 'N' condition code. The output of the ALU is used to compute the 'Z' condition code. A control signal 'Set.CC' the 'S' bit of the instruction and an OR of 2 and 3 address instructions determines if the PSW's N and Z bits will be set. Opcode 12 is used to enable the

value held in PC for jump subroutine. The control signal 'Result In' pushes the result from the ALU into the Result register. The result is then passed to the rest of the circuit as selected by the Writeback decoder shown in Figure 10 below.

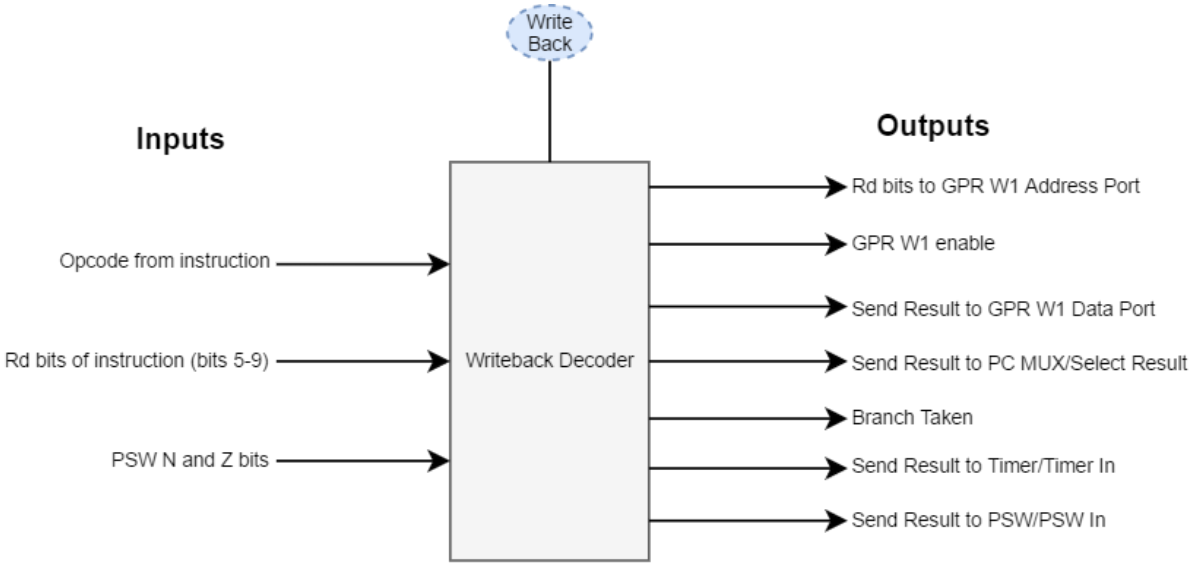

Figure 10: Writeback Decoder

The Writeback decoder is enabled by the 'Write Back' Control signal. The inputs for the writeback decoder are the instruction's opcode, PSW's N and Z bits, and bits five through nine of the instruction from the instruction register. These are the bits that could possibly contain the Rd bits if the instruction is a 3-address or 2-address instruction. The Writeback decoder has 7 outputs. The first two outputs are used to write a value to the GPR. The first is connected to the W1 address port of the GPR. This will send the correct R_d bits as an address to the GPR if the instruction's opcode indicates an instruction that writes to the GPR (opcodes 0-7 and 12). The next output is connected to the Write 1 signal port of the GPR. This writes the data at the address specified by R_d and is also activated by opcodes 0-7 and 12.

The next Writeback output signal enables a tri-state buffer that passes the value in the Result register to the W1 data port of the GPR. This signal is enabled if the instruction's opcode is 0-7. In the case of opcode 12, this signal is not needed because the value from the PC register is already at the W1 data port.

The next Writeback output signal enables a tri-state buffer that passes the value in the Result register to the PC multiplexer shown in the Fetch stage. This signal is enabled if the opcode indicates a branch instruction (opcode 9, 10, or 11) or a jump/return subroutine instruction (opcode 12 or 13). In the case of BRN, this signal will be enabled only if the opcode is 9 and the PSW's N bit is 1. In the case of BRZ, this signal will be enabled only if the opcode is 10 and the PSW's Z bit is 1. In addition to enabling the buffer, this signal also sends a 1 value to the Fetch multiplexer so that result value is selected over the incremented PC value. These instructions also cause a Branch Taken signal to be sent to the valid bits of each instruction in

the pipeline. The signal resets the valid bits for each instruction buffer. Resetting the valid bits is necessary for flushing instructions from the pipeline.

The next Writeback output signal enables a tri-state buffer that passes the value in the Result register to the Timer. This signal also sends a Timer In signal to allow the result to be set as the new timer value. This signal is sent only if the opcode is 14.

The final Writeback output signal enables a tri-state buffer that passes the value in the Result register to the PSW. This signal also sends a PSW In signal to allow the result to be stored in the PSW register. This signal is sent only if the opcode is 15. For opcode 8, the Writeback decoder will output no signals. This is because that Store instruction has already done its job during the MEM stage.

Optimizations

Data Hazards

A data hazard occurs when an instruction needs to access data that has not been written yet. In the case of the processor presented in this report, the data accessed during the decode stage may be incorrect by the time it gets to the execute stage. This is because the instruction before it may write data to an address the instruction uses. To make sure that the instruction performs its operation on the correct data, a data forwarding unit is used in the pipeline. The data forwarding unit looks at the instructions in the IR MEM/EX buffer and the IR D/MEM buffer. The reason the data forwarding unit is positioned here is to handle data hazards that may arise when an Execute result denotes a memory address or indirect operand address. The ALU result from the execute stage is also wired into the data forwarding unit. Wiring the ALU result directly into this unit allows the result to be used the moment the instruction is loaded instead of having to wait until the result is stored in a register. This is possible because an ALU and this data forwarding unit consist entirely of logic gates, meaning clock cycles are not necessary to obtain this data. An example of the inputs and outputs of the processor's data hazard unit is shown below.

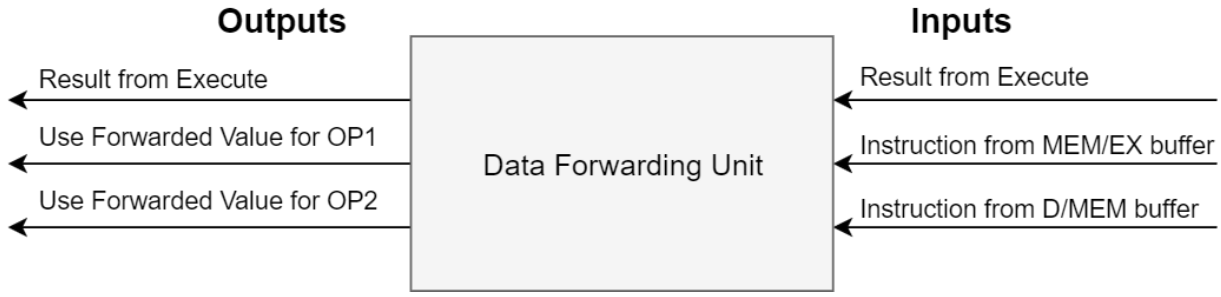

Figure 11: Data Forwarding Unit

Data Hazards That Can Occur in This Processor

Whenever an instruction writes to an address in the GPR, there is a possibility that a data hazard can occur if the instruction fetched after it needs to operate on the data stored in that address. The following table shows which kind of instructions can be affected by data hazards and which instructions can cause data hazards

Table 1: Instruction Data Hazard Information:

Instruction	Opcode	Affected	Can Cause
ADD	0	Y	Y
SUB	1	Y	Y
AND	2	Y	Y
SHL	3	Y	Y
OR	5	Y	Y
NOT	6		Y
LD	7		Y
ST	8	Y	
BRN	9		
BRZ	10		
BR	11		
JSR	12		Y
RTS	13	Y	
CLK	14		
LPSW	15		

The Data Hazard unit checks the instruction in the MEM/EX buffer to determine if the instruction is one of the instruction that could cause data hazards. It also checks the instruction in the D/MEM buffer to see if it is an instruction that could be affected by a data hazard. If both instructions have the possibility of a data hazard, the addresses in each instruction are compared. For opcodes 0-5, the RS1 and RS2 bits in the D/MEM buffer are compared with the Rd bits of the instruction in the MEM execute buffer. If RS1 matches Rd, the data hazard unit will output the result tapped off the ALU and the Use Forwarded Value for OP1 signal. If RS2 matches Rd, the data hazard will output the result tapped off the ALU and the Use Forwarded Value for OP2 signal. These signals are wired to the purple selection multiplexers shown in Figure 8. The positioning of these multiplexers allows for the forwarded value to be addressed in MEM or stored in an OP register for Execute without adding any clock cycles to the execution of an instruction. If an ST or RTS instruction is in the D/MEM buffer, the Rd bits of both instructions are checked. If a match is found, the Use Forwarded Value for OP1 signal is sent because a value from the GPR at Rd was stored in this register during the decode stage.

There is also a possibility that an instruction two buffers ahead may be writing to an address that is accessed when decoding an instruction. Instead of implementing a data forwarding unit to avoid this hazard, the decode control signals are sent on the second clock cycle instead of the first. This means that the Result value from the Execute stage is being written at the same time as the value is read in the Decode stage. Having been given permission to assume a master/slave relationship between GPR registers, it is safe to assume that the correct values will end up in the Decode stage's OP1 and OP2 registers when the control signals are sent on the second clock cycle. Overall, this change is an effective way to avoid this data

hazard because it does not require any extra hardware and it does not add any extra time because the Fetch instruction will always take 2 clock cycles.

Control Hazards

Control hazards can occur whenever a branch occurs. To resolve this data hazard, the pipeline statically predicts that a branch is not taken. When this prediction strategy is used, it is assumed that conditional branches are not taken and instructions are loaded into the pipeline until the branch instruction reaches the Execute stage. This assumption greatly increases the speed of the pipeline when branches are not taken because the next instructions has already moved through most of the pipeline. However, in cases where a branch is taken, the existing instructions need to be flushed from the pipeline. This is accomplished by the Writeback decoder in the execution stage. When a branch is taken, the Writeback decoder issues a control signal to reset each instruction's valid bits from 1 to 0. This will cause each subsequent stage to treat these instructions as no-ops and will flush them from the pipeline as valid instructions are fetched.

There is one risk in using the static branch not taken branch prediction strategy. It assumes that all instructions that are flushed in the pipeline have not stored any data or made any other permanent changes to the rest of the system. For most of the instructions in this instruction set, this assumption holds true. However, the store instruction (opcode 8) can cause a problem if it is preceded by one of the branching instructions indicated by opcodes 9, 10, 11, 12, or 13. This is because writing a value to main memory can be considered a permanent change to the system. This hazard has been avoided by sending the internal MEM sub-signals for write on the second clock cycle of the stage instead of the first. This gives the Execution stage time to determine if a branch is taken and reset the MEM instruction's valid bit before the write can occur. Implementing this fix does not require any new hardware and it does not add any clock cycles because the Fetch stage requires two clock cycles to complete anyways.