Kristy Nguyen
Dr. Ed Jorgensen
CS 219-1002: Computer Organization
19 March 2021

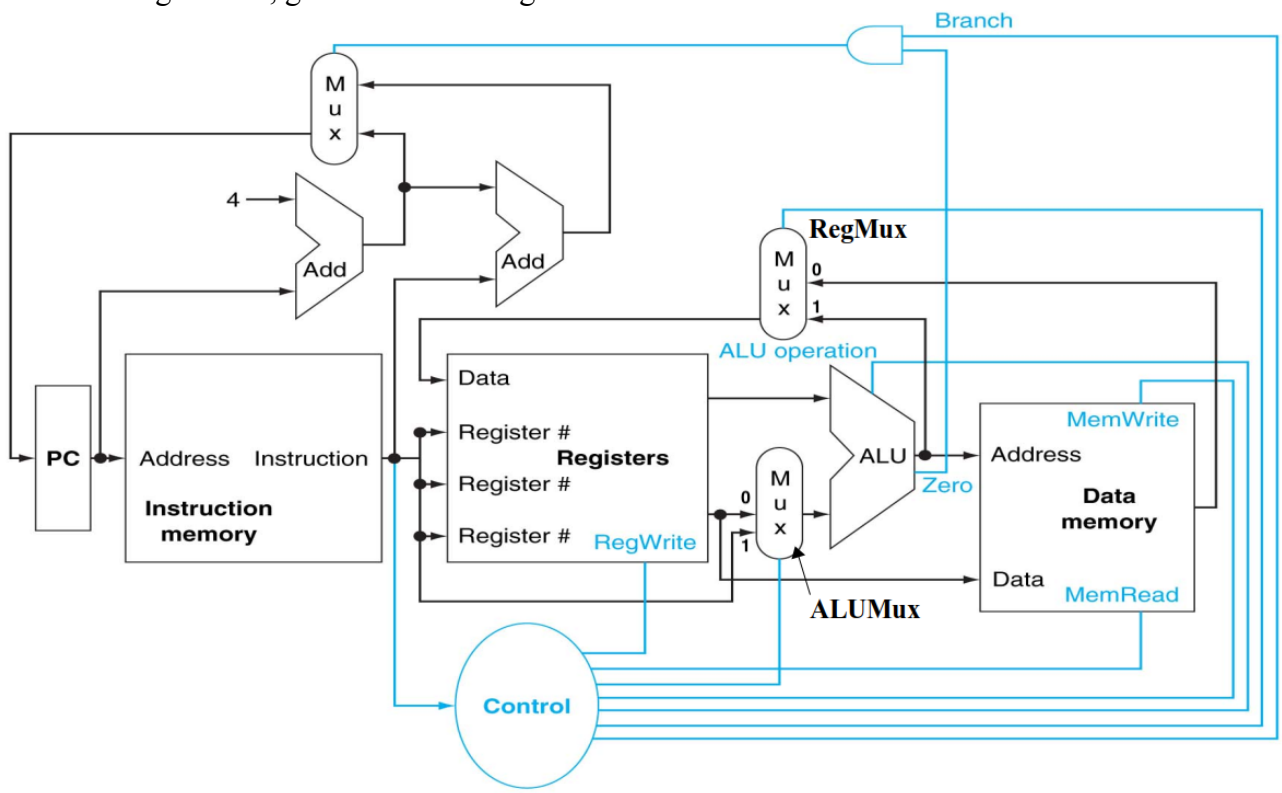
CS 219 – Assignment #6

Purpose: Become familiar with processor implementation Points: 125

Reading/References:
Chapter 4

Part A - Processor
Given the following MIPS processor block diagram. *Diagram 4.2 (page 247)*

Assignment:
1) Different instructions utilize different hardware blocks in the basic single-cycle implementation. Based on Diagram 4.2, given the following instructions:



1 `sub Rd, Rs, Rt` where, Rd = Rs - Rt
2 `lw Rt, offset(Rs)` where, Rt = Memory[Rs+offset]

a) What are the values of control signals generated by the control for each instruction? *Note*, '1' is asserted (on) and '0' is not asserted (off). For ALUop, add is 0 and sub is 1. [5 pts]

	RegWrite	MemRead	ALUMux	MemWrite	ALUOp	RegMux	Branch
1	1	0	0	0	1	1	0
2	1	1	1	0	0	0	0

The resource blocks shown in the block diagram include *PC Unit*, *Instruction Memory Unit*, *Register File*, *ALU*, *Data Memory Unit*, and *Branch Adder Unit*.

b) Which resources (blocks) perform a useful function for each instruction? [5 pts]
1: PC Unit, Instruction Memory Unit, Register File, ALU
2: PC Unit, Instruction Memory Unit, Register File, ALU, Data Memory Unit

c) Which resources (blocks) produce outputs, but their outputs are not used for each instruction? Which blocks do not produce outputs for each instruction? [5 pts]
The Branch Adder Unit produces outputs but their outputs are not used for each instruction. For instruction 1, the Data Memory Unit does not produce an output. For instruction 2, all resources (blocks) produce an output.

2) The basic single-cycle MIPS implementation in the diagram can only implement some instructions. New instructions can be added to an existing ISA, but the decision whether or not to add new instruction(s) depends, among other things, on the cost and the complexity such new instructions introduce into the processor datapath and control. Based on Diagram 4.2, given the following instructions:

1 `sub3 Rd, Rs, Rt, Rx` where, Rd = Rs + Rt + Rx
2 `srl Rd, Rt, shftamt` where, Rd = Rt << shftamt (shift right)

a) Which existing blocks (if any) can be used for each instruction? [5 pts]
For instruction 1, the Program counter (PC), Instruction Memory Unit, Register File, and Arithmetic Logic Unit (ALU) can be used. For instruction 2, the PC, Instruction Memory Unit, Register File, and ALU can be used.

b) Which new functional blocks (if any) are required for each instruction? *Note*, if the change is difficult or relatively easy. [5 pts]
No new blocks are required. The existing blocks register file and ALU need to be updated in order to implement instruction 1. The change for instruction 1 would be difficult because of the additional function units required. The change for instruction 2 would be simple because no hardware changes would be required.

c) What new signals are required (if any) from the control unit to support this instruction? [5 pts]

To support instruction 1, a new control signal that informs the new ALU to perform the new instruction would be needed.

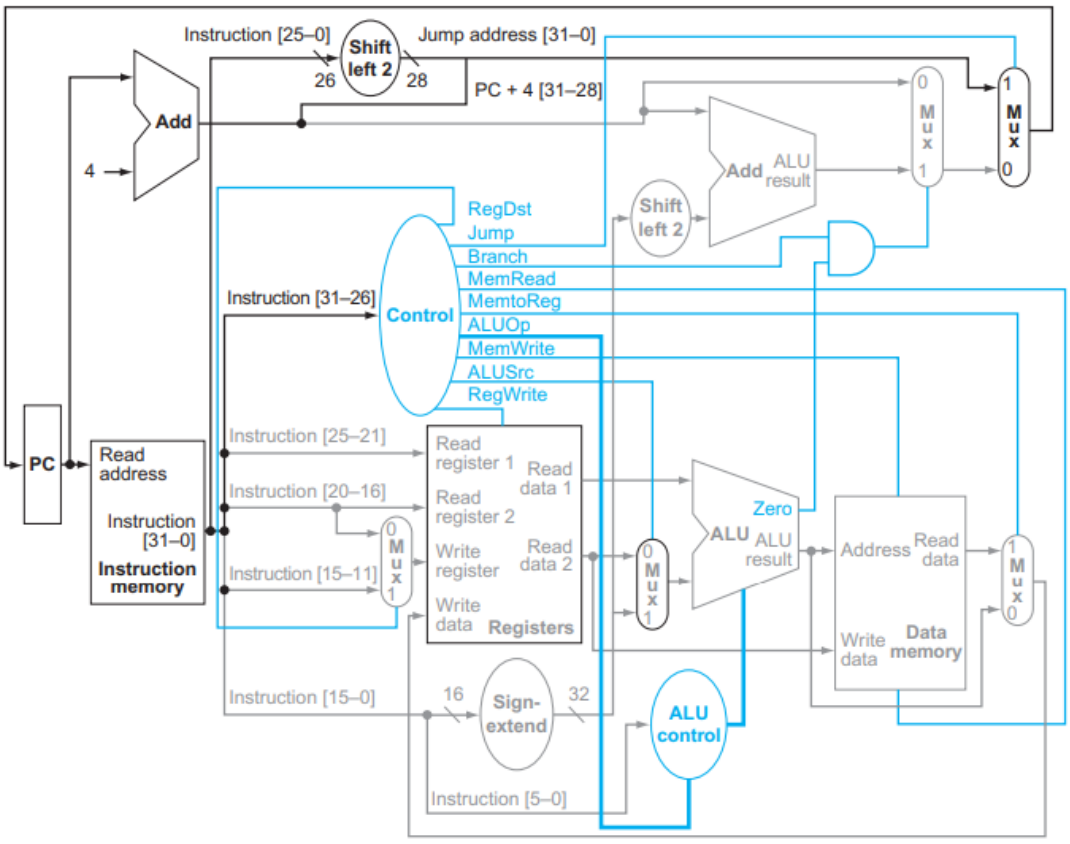


Diagram 4.24 (page 271)

3) A manufacturing defect can cause a single line to have a constant logical value. This is referred to as a "stuck-at-0" or "stuck-at-1" fault. Using the above diagram, and the below signal faults, answer the following questions.

Fault 1: Instruction Memory, output instruction, bit 6
Fault 2: Control Unit → output *MemRead*

a) Assume that processor testing is performed by filling the $pc, registers, data, and instruction memories with some values and letting a single instruction execute. What type of instruction would be required to test each possible fault (#1 and #2) for a "stuck at-0" type fault? [5 pts]

To test fault 1, `addi $1, $0, 64` is required to test for instruction memory, output instruction, bit 6. To test fault 2, `l<type> Rdest, mem` is required to test for the control unit to output MemRead.

4) We examine the operation of the single-cycle datapath for specific instructions. Based on Diagram 4.24, given the following instructions:

1 `lw $1, 40($6)`
2 `lbl: bne $1, $2, lbl`

a) What is the value of the instruction word? Show the answer in hex or binary. If unknown, use '?' to indicate. [5 pts]
1: `100011 00110 00001 0000000000101000`
2: `000101 00001 00010 ????????????????`

b) What is the register number supplied to the register file's "Read Register 1" input? Is this register actually read? What is the register number supplied to the register file's "Read Register 2" input? Is this register actually read? [5 pts]
1: The register number supplied to the register file's "Read Register 1" input is $6, which is actually read. The register number supplied to the register file's "Read Register 2" input is $1, which is not actually read.

2: The register number supplied to register file's "Read Register 1" input is $1, which is actually read. The register number supplied to the register file's "Read Register 2" input is $2, which is actually read.

c) What is the register number supplied to the register file's "Write Register" input? Is this register actually written? [5 pts]
1: The register number supplied to the register file's "Write Register" input is $1, which is actually written.

2: The register number supplied to the register file's "Write Register" input is $2, which is not actually written.

5) Different instructions require different control signals to be asserted in the datapath.

	Control Signal 1	**Control Signal 2**
1	RegDst	MemRead
2	RegWrite	MemRead

a) What is the value of these two signals for each instruction (#1 and #2)? [5 pts]
1: The value for the RegDst for instruction #1 is 0. The value for the MemRead for instruction #1 is 1.

2: The value for the RegWrite for instruction #2 is 0. The instruction for the MemRead for instruction #2 is 0.

6) Based on Diagram 4.24, given the following instruction words:

1 `0b10001100010000000000000000010000`
2 `0b00010000001000110000000000001100`

a) What are the outputs of the sign-extend and the jump "Shift Left 2" unit (upper left corner of Diagram, 4.24) for each instruction word? [5 pts] *Note, the question refers to the shift left in the upper left side.*

1: sign extend - `00000000000000000000000000010000`
shift left 2 - `00000000000000000000000001000000`

2: sign extend - `00000000000000000000000000001100`
shift left 2 - `00000000000000000000000000110000`

b) What are the values of the ALU control unit's inputs for this instruction? [5 pts]

1: The value of the ALU control unit's input for this instruction is 0010.

2: The value of the ALU control unit's input for this instruction is 0110.

c) What is the new **`$pc`** address after each instruction is executed? [5 pts]

The new $pc address after instruction #1 is `PC + 4`. The new $pc address after instruction #2 is `PC + 4 + BranchAddr * 4`.

Part B - Pipelining

1) Assuming it was possible to break instructions into two equal parts, explain why a two-stage pipeline will still not halve the instruction cycle time (as compared to no pipelining) in a realistic setting. [5 pts]

A two-stage pipeline will still not halve the instruction cycle time because splitting the stage only reduces the cycle time based on the new longest stage.

2) A pipelined processor has a clock rate of 2.5 GHz and executes a program with 1.5 million instructions. The pipeline has five stages and instructions are issued at a rate of one per clock cycle. Ignore penalties due to all hazards and filling the pipeline. Assume perfect overlap of instructions. [5 pts, 2.5 pts each]

a) What is the speed-up of this processor for this program compared to a non-pipelined processor?
The speed-up of this program compared to a non-pipelined processor is a factor of five.

b) What is the throughput (in Millions of Instructions Per Second) of the pipelined processor?
The throughput of the pipelined processor is 2500 Millions of Instructions Per Second since one instruction is completed per cycle.

3) Given basic MIPS five-stage pipeline: [5 pts, 2.5 pts each]

a) If the performance of the ALU is improved by 50%, what will be the impact on the overall performance of the CPU?

The impact on the overall performance of the CPU is none if the performance of the ALU is improved by 50% because the overall performance is still as long as the longest stage.

b) If the performance of the ALU is degraded by 50%, what will be the impact on the overall performance of the CPU?

If the performance of the ALU is degraded by 50%, the impact on the overall performance of the CPU is 50 percent worse because the ALU takes longer.

4) Briefly explain each of the following methods for dealing with a conditional branch instruction? [4 pts, 1 pts each]

a) Stall

Conditional branches need to make a branch decision, but other instructions are already being executed. A stall means to wait one stage, which solves the control hazard problem but slows the performance.

b) Static Branch Prediction

Static branch prediction is to assume the branches are not taken. When the static branch prediction is correct, the pipeline proceeds at full speed. Only when the branches are taken does the pipeline stall.

c) Dynamic Branch Prediction

Dynamic branch prediction is keeping a history for each branch as taken or untaken, and then using the recent past behavior to predict the future. When the guess is wrong, the pipeline must ensure that the instructions following the wrongly guessed branch have no effect and must restart the pipeline from the proper branch address.

d) Delayed Branch

Delayed branch always executes the next sequential instruction, with the branch taking place after that one instruction delay. MIPS software will place an instruction immediately after the delayed branch instruction that is not affected by the branch, and a taken branch changes the address of the instruction that follows this safe instruction.

5) Present one very simple possible method for which could be used to perform branch prediction using two bits for the history? [6 pts]

Using two bits to track the history of predictions for a branch is useful because there are four possible scenarios: predict taken and actually taken, predict taken and actually not taken, predict not taken and actually taken, and predict not taken and not actually taken.

6) Given basic MIPS five-stage pipeline and the following instruction sets: [10 pts, 5 pts each]

Instruction set #1

```
lw $1, 40($6)
add $2, $3, $1
```

```
add $1, $6, $4
sw $2, 20($4)
and $1, $1, $4
```

Instruction set #2

```
add $1, $5, $3
sw $1, 0($2)
lw $1, 4($2)
add $5, $5, $1
sw $1, 8($2)
```

a) If there is no forwarding or hazard detection, write the instructions (same order) and insert **nop**'s to force the appropriate number of stalls which would be required.

Instruction set #1:

```
lw $1, 40($6)
nop
nop
add $2, $3, $1
add $1, $6, $4
sw $2, 20($4)
and $1, $1, $4
```

Instruction set #2:

```
add $1, $5, $3
sw $1, 0($2)
nop
nop
lw $1, 4($2)
nop
nop
add $5, $5, $1
sw $1, 8($2)
```

b) Rearrange the instructions to avoid hazards where possible. Insert **nop**'s to force the appropriate number of stalls when a hazard can not be avoided. If needed, update/change the instructions. You can use register $15 to hold temporary values in the modified code.

Instruction set #1:

```
lw $1, 40($6)
add $1, $6, $4
add $2, $3, $1
sw $2, 20($4)
and $1, $1, $4
```

Instruction set #2:

```
add $1, $5, $3
lw $15, 0($2)
lw $1, $4($2)
sw $15, 4($2)
add $5, $5, $1
sw $1, 8($2)
```

7) Given basic MIPS five stage pipeline, with data forwarding, and the following instruction sets: [10 pts, 5 pts each]

Instruction set #1

```
L1: lw $1, 40($6)
 beq $2, $3, L2 # taken
 add $1, $6, $4
L2: beq $1, $2, L1 # not taken
 sw $2, 20($4)
 and $1, $1, $4
```

Instruction set #2:

```
 add $1, $5, $3
 beq $2, $4, L2 # not taken
L1: sw $1, 0($2)
 add $2, $2,$3
L2: add $5, $5, $1
 sw $1, 8($2)
```

a) Assuming no delayed branches and that branches execute in the EX stage, draw the pipeline execution diagram for this code.

instruction set #1	pipeline cycle											
	1	2	3	4	5	6	7	8	9	10	11	12
lw	IF	ID	EX	ME	WB							
beq (t)		IF	ID	EX	ME	WB						
beq (nt)					IF	ID	EX	ME	WB			
sw						IF	ID	EX	ME	WB		
and								IF	ID	EX	ME	WB

Note, it is not required to show the stall explicitly

instruction set #2	pipeline cycle											
	1	2	3	4	5	6	7	8	9	10	11	12
add	IF	ID	EX	ME	WB							

beq (nt)		IF	ID	EX	ME	WB						
sw			IF	ID	EX	ME	WB					
add				IF	ID	EX	ME	WB				
add					IF	ID	EX	ME	WB			
sw						IF	ID	EX	ME	WB		

8) Given basic MIPS five-stage pipeline, with data forwarding, and the following instruction sets:
a) Assuming there are delayed branches (so branches execute in the ID stage), and the instruction following the conditional branch is a safe instruction (i.e., allowed in the delay slot), draw the pipeline execution diagram for this code. [10 pts, 5 pts each]

Instruction set #1 (original):

```
L1: lw $2, 4($5)
 sub $1, $7, $4
 and $9, $10, $12
 beq $1, $2, L2 # taken
 xor $13, $14, $15
 mul $5, $6, $8
L2: beq $1, $3, L3 # not taken
 sw $8, 4($9)
L3: or $1, $1, $4
```

Instruction set #1 (re-written by assembler to support delayed branches):

```
L1: lw $2, 4($5)
 sub $1, $7, $4
 beq $1, $2, L2 # taken
 and $9, $10, $12 # safe inst
 mul $5, $6, $8
L2: beq $1, $3, L3 # not taken
 xor $13, $14, $15 # safe inst
 sw $8, 4($9)
L3: or $1, $1, $4
```

Instruction set #2 (original):

```
 add $1, $5, $3
 and $9, $10, $12
 beq $2, $4, L2 # taken
 mul $4, $5, $6
 div $4, $5, $6
L1: sw $1, 0($7)
 add $2, $2, $3
 beq $2, $4, L2 # not taken
 shl $2, $2, $3
```

```
 add $5, $5, $1
L2: sw $1, 8($2)
```

Instruction set #2 (re-written by assembler to support delayed branches):

```
add $1, $5, $3
 beq $2, $4, L2 # taken
 and $9, $10, $12 # safe inst
 mul $4, $5, $6
 div $4, $5, $6
L1: sw $1, 0($7)
 beq $2, $4, L2 # not taken
 add $2, $2, $3 # safe inst
 shl $2, $2, $3
 add $5, $5, $1
L2: sw $1, 8($2)
```

instruction set #1	pipeline cycle											
	1	2	3	4	5	6	7	8	9	10	11	12
lw	IF	ID	EX	ME	WB							
sub		IF	ID	EX	ME	WB						
beq (t)			IF	ID	EX	ME	WB					
and				IF	ID	EX	ME	WB				
beq (nt)					IF	ID	EX	ME	WB			
xor						IF	ID	EX	ME	WB		
sw							IF	ID	EX	ME	WB	
or								IF	ID	EX	ME	WB

instruction set #2	pipeline cycle											
	1	2	3	4	5	6	7	8	9	10	11	12
add	IF	ID	EX	ME	WB							
beq (t)		IF	ID	EX	ME	WB						

and			IF	ID	EX	ME	WB					
sw				IF	ID	EX	ME	WB				