Name: Cicelia SIu

**CS 219 – Assignment #6**

Purpose: Become familiar with processor implementation

Points: 125

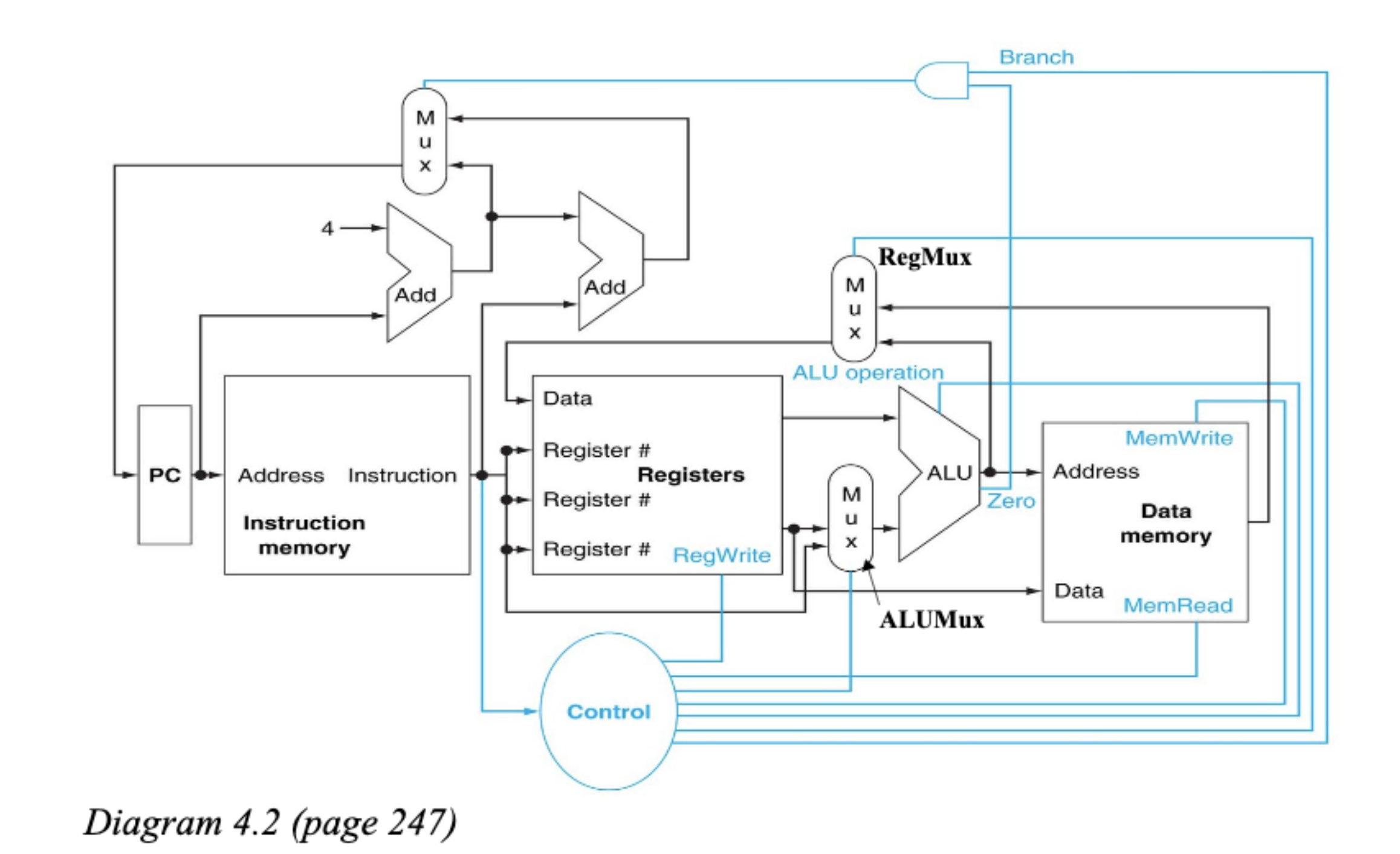
Reading/References:

Chapter 4

Assignment:

**Part A - Processor**

Given the following MIPS processor block diagram.



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]

**Part A, Table Question #1**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **RegWrite** | **MemRead** | **ALUMux** | **MemWrite** | **ALUOP** | **RegMux** | **Branch** |
| **1** | 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| **2** | 1 | 1 | 1 | 0 | 0 | 1 | 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]

1- Branch Adder produces but is not used. Data Memory does not produce output

2- Branch Adder produces but is not used. All blocks produce outputs.

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]

1- PC unit, Instruction Memory Unit, Register File, ALU.

2- PC unit, Instruction Memory Unit, Register File, ALU

b) Which new functional blocks (if any) are required for each instruction?

Note, of the change is difficult or relatively easy. [5 pts]

1- Another read register for Rx with another ALU for Rx ; difficult

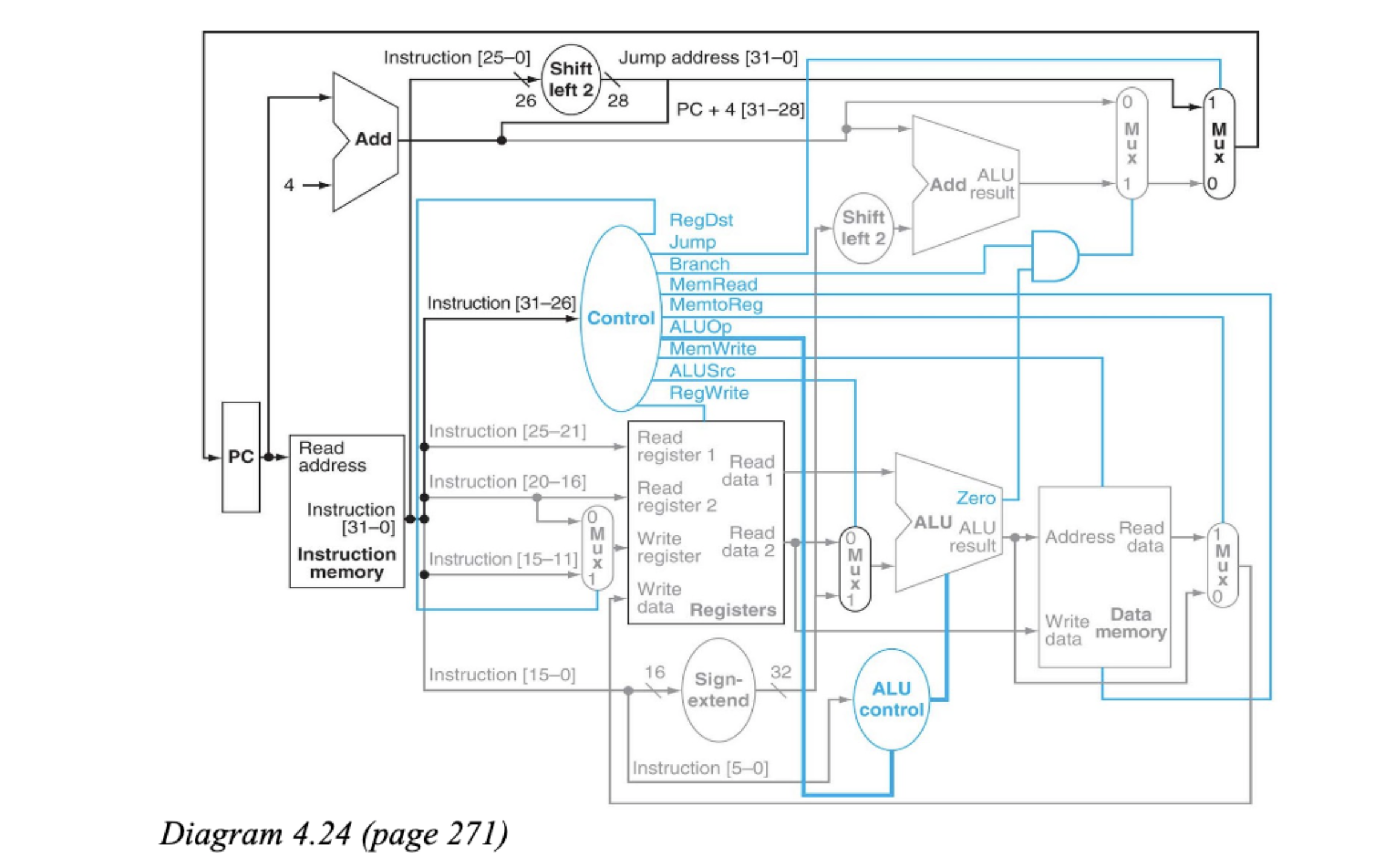
2- Extend existing ALU ; relatively easy

c) What new signals are required (if any) from the control unit to support this

instruction? [5 pts]

1 - Tell new ALU for Rx what to do.

2 - change control signals for shift right operation



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]

Fault 1 - bit 6 instruction is a offset so add $1, $zero, 64 ( all bits up to bit 6 = 64)

Fault 2 - LW $1, 1024(zero). The value in $1 is supposed to be zero. If Memread is stuck-at-0, the value in the register will be 1024

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 answer in hex or binary. If unknown,

use '?' to indicate. [5 pts]

1 - 100011 | 00110 | 00001 | 0000000000101000 = 0x8CC10028

2 - 000101 | 00001 | 00010| ???????????????? = 0x1422????

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 - Read register 1 is $6 or 000110. This is read and used.

the second one is $1 or 00001. It is read but not used.

2 - Read register 1 is $1 or 00001. It is read and used.

Read Register 2 is $2 or 00010. It is read and used.

c) What is the register number supplied to the register file's “Write Register” input? Is this

register actually written? [5 pts]

1- Write register is $1 or 00001. It is written and used.

2 - write register is $2 or 00010. But could also be the number of the lbl.

Register is not 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 - lw : RegDst = 0; MemRead = 1

2 - bne: RegWrite = 0 ; MemRead = 0

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

1 0b 1000 1100 0100 0000 0000 0000 0001 0000

2 0b 0001 0000 0010 0011 0000 0000 0000 1100

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 - bits 0-15 are 0000 0000 0001 0000

sign extend = 0000 0000 0000 0000 0000 0000 0001 0000

bits 0-25 are 00 0100 0000 0000 0000 0001 0000

jump shift left 2 = 0001 0000 0000 0000 0000 0100 0000

2 - bits 0-15 are 0000 0000 0000 1100

sign extend = 0000 0000 0000 0000 0000 0000 0000 1100

bits 0-25 are 00 0010 0011 0000 0000 0000 1100

jum shift left 2 = 00000 1000 1100 0000 0000 0011 0000

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

1 - OP code is 100011 so lw and 6 bit function field is 010000. ALUOp is 00

2 - OP code is 000100 so bne and 6 bit function field is 001100. ALUOp is 01

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

1 - PC + 4

2 - PC + 4 if taken.

**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]

Will probably have to take longer when broken in half. The IF process will be waiting for the first

operation and will not execute in time.

2) A pipelined processor has a clock rate if 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?

time = # of instructions / clockrate

= 1.5 x 106 / 2.5 x 109

= **0.6 x 10-3 seconds**

b) What is the throughput (in Millions of Instructions Per Second) of the pipelined processor?

MIPS = 1 x 106

2.5 GHZ = 2500 x 106

= **2500 MIPS**

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?

None.

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

performance of the CPU?

50 % worse.

4) Briefly explain each of the following methods for dealing with a conditional branch instruction?

[4 pts, 1 pts each]

a) Stall - delay the execution of the instruction

b) Static Branch Prediction - guess based on branch and forward/backward reference

c) Dynamic Branch Prediction - maintain internal history and predict based on history

d) Delayed Branch - reordering code with safe instructions

5) Present one very simple possible method for which could be used to perform branch prediction

using a two bits for the history? [6 pts]

2 bit predictor. Remembers the history of taken (1) and not taken (0). Possibly 00, 01, 10, or 11.

Uses possible history patterns to predict. (2n possible patterns)

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.

1- lw $1, 40($6)

**nop op**

add $2, $3, $1

add $1, $6, $4

sw $2, 20($4)

**nop op**

and $1, $1, $4

2-add $1, $5, $3

**nop op**

sw $1, 0($2)

lw $1, 4($2)

**nop op**

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.

1-lw $**15**, 40($6)

**add $1, $6, $4**

**add $2, $3, $15**

**nop op**

sw $2, 20($4)

and $1, $1, $4

2-add $1, $5, $3

**lw $15, 4($2)**

**sw $1, 0($2)**

add $5, $5, $15

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

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| instruction set #1 | pipeline cycle | | | | | | | | | | | |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| lw | IF | ID | EX | ME | WB |  |  |  |  |  |  |  |
| beq |  | IF | ID | EX | ME | WB |  |  |  |  |  |  |
| stall |  |  |  |  |  |  |  |  |  |  |  |  |
| stall |  |  |  |  |  |  |  |  |  |  |  |  |
| beq |  |  |  |  | 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 |  | 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, 5pts 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 |  |  | IF | ID | EX | ME | WB |  |  |  |  |  |  |
| and |  |  |  | IF | ID | EX | ME | WB |  |  |  |  |  |
| beq |  |  |  |  | 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 | 13 |
| add | IF | ID | EX | ME | WB |  |  |  |  |  |  |  |  |
| beq |  | IF | ID | EX | ME | WB |  |  |  |  |  |  |  |
| and |  |  | IF | ID | EX | ME | WB |  |  |  |  |  |  |
| sw |  |  |  | IF | ID | EX | ME | WB |  |  |  |  |  |
| beq |  |  |  |  | IF | ID | EX | ME | WB |  |  |  |  |
| and |  |  |  |  |  | IF | ID | EX | ME | WB |  |  |  |
| shl |  |  |  |  |  |  | IF | ID | EX | ME | WB |  |  |
| and |  |  |  |  |  |  |  | IF | ID | EX | ME | WB |  |
| sw |  |  |  |  |  |  |  |  | IF | ID | EX | ME | WB |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |