EE 457 Midterm

Summer '14 ● Redekopp

Name: _____

Closed Book / 105 minutes **No CALCULATORS** **Score: _____ / 100**

1. **(17 pts.) Short Answer** [Fill in the blanks or select the correct answer]
   a. If a control signal must be valid during the majority of the clock cycle it is advisable to use a _____ (**Mealy** / **Moore**)-style signal.

   b. A single-cycle CPU sets as constant the _____ (**CPI** / **clock cycle time**) while letting the other vary based on the design and instruction set.

   c. Perform the indicated arithmetic operations, showing your work, for the specified representation system. Answers are limited to 8-bits (not 9-bits). (Do not use the borrow method for subtraction, use the 2's complement method of subtraction.) Finally, state whether overflow has occurred and **briefly explain why or why not**.

a.) 2's comp. system

$$\begin{array}{r} 10010001_2 \\ -\ 01111111_2 \\ \hline \end{array}$$

10010001
10000000
\+ 1
00010010

Overflow: **Y** / **N**

b.) Unsigned system

$$\begin{array}{r} 10101011_2 \\ +\ 01101010_2 \\ \hline \end{array}$$

00010101

Overflow: **Y** / **N**

   d. A processor with a 32-bit _____ (**address** / **data**) bus would necessarily limit memory to 4 GB.

   e. ____ **True** / **False**: State machines built using one-hot encoding use more flip-flops than other encoding mechanisms.

   f. Branch instructions perform the following operation: __**PC = PC+d**_____

   g. A processor with a 16-bit data bus would generally imply ____ (**0** / **1** / **2** / **4**) Byte Enable signals and allow the lower ___ (**0** / **1** / **2** / **4**) bits of the address bus to be unused.

   h. Amdahl's law would argue that HW optimization of a processor should focus on _____
      1.) **The most frequently executed instruction**
      2.) The slowest instruction
      3.) The fastest instruction

   i. Ideal CPI of a pipelined processor is ___1___ (fill in the blank) while in practice it will be _____ (**higher** / **lower**) due to hazards.

2. **(26 pts.) State Machine Design**. Consider a memory containing 8 data values. Tommy Trojan needs to **<u>find the smallest two values (i.e. min1=smallest and min2=next smallest)</u>** in the data values. You may assume that all values are unique and there are no duplicates in the data.

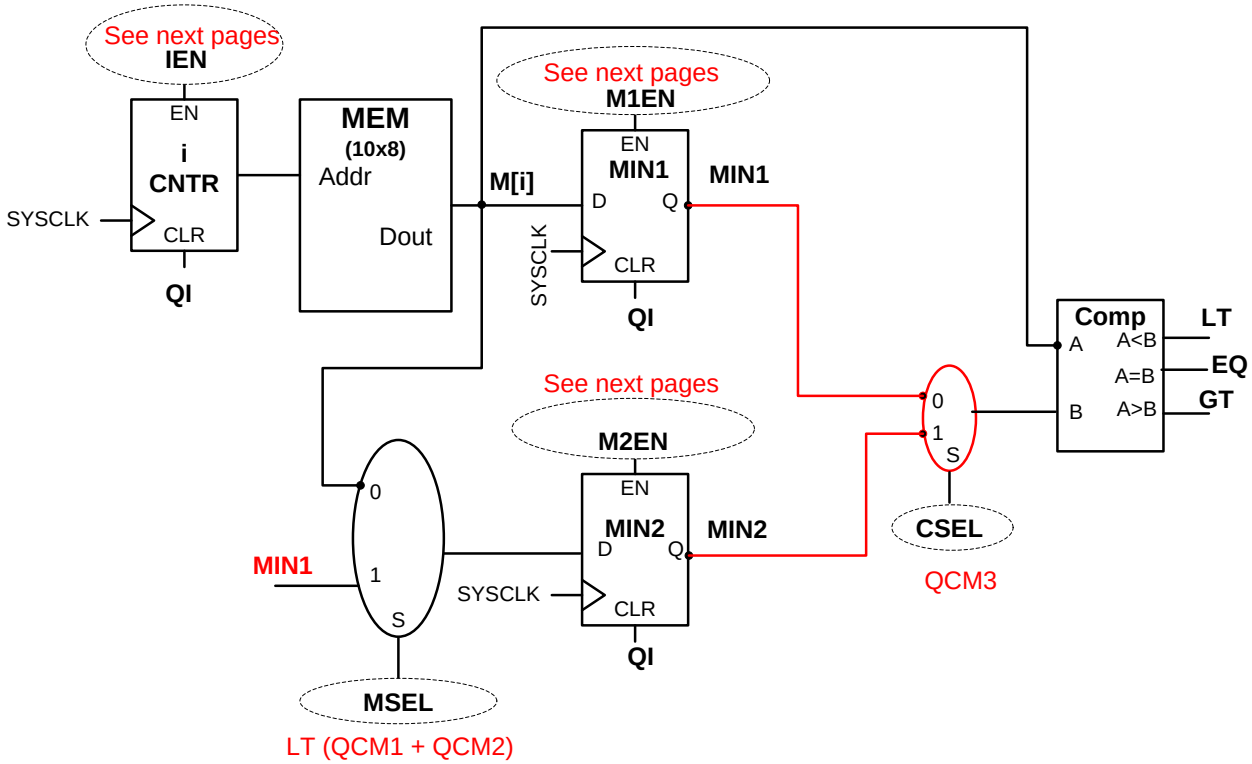
Memory Contents	*Address*	0	1	2	3	4	5	6	7
	MEM	10	8	7	6	3	5	1	11

For the particular values shown above, then after the computation MIN1=1, MIN2=3.

To solve this problem we will iterate/scan over the data only once with two registers: MIN1 (stores the smallest) and MIN2 (stores the second smallest). We will also only have one comparator as shown.

a. Study the partial datapath below and complete the connection (adding any additional logic components) to produce the B input of the comparator as well as I1 of the MSEL mux.

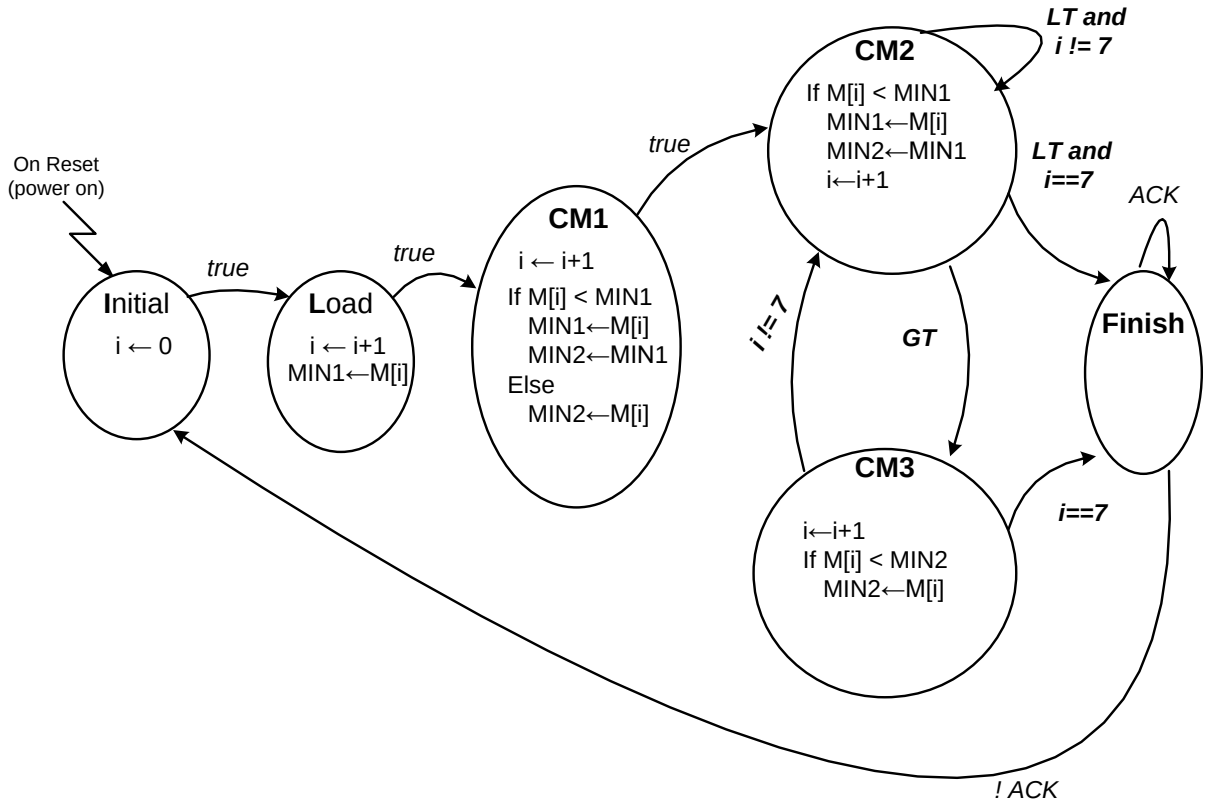
Do NOT try to complete the control signals now. Answer the questions on the bottom of the page and then go on to page 3 and complete the state diagram first.



b. State machine implementation. Given below is a ***partially completed*** state diagram. The Initial, Load, and Finish states are completed w/ transitions. Please complete the transition and operations for the remaining 3 states. Their names mean:

CM1=CompareMin1 = Update register(s) based on the value of M[1]
CM2=CompareMin2 = Compare M[i] with MIN1
CM3=CompareMin3 = For you to figure out…

Complete all the state transitions from CM2 and CM3 and indicate what operations should be performed in CM1, CM2, and CM3 using register-transfer level descriptions.

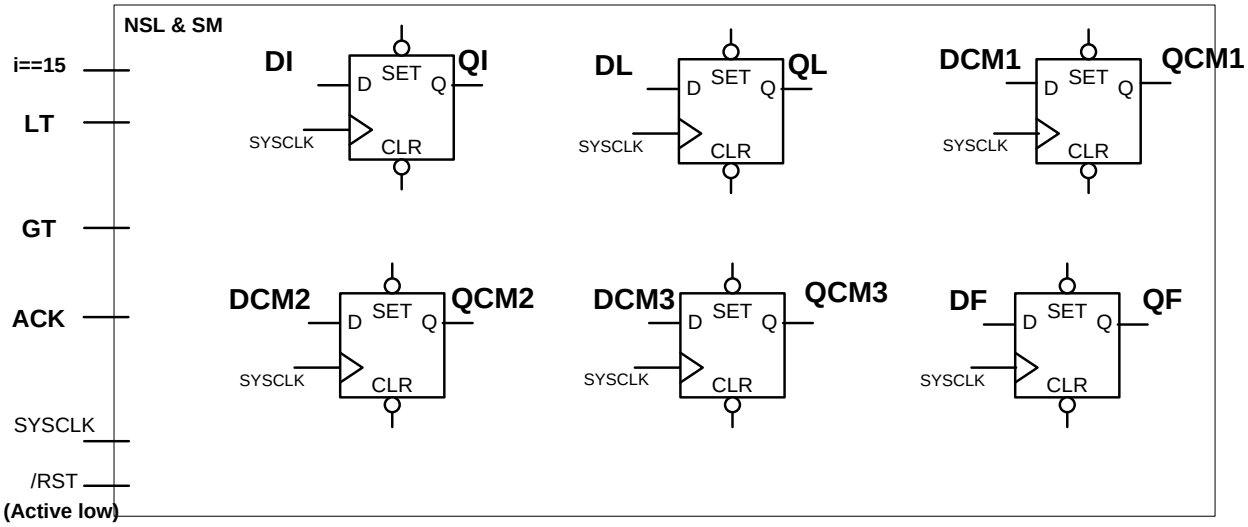


c. Given the content of the memory as shown below, show a sample execution of your state machine. Complete the following table for the value of ***i (the memory address counter)*** and ***STATE*** (which state you are in) during that clock cycle (i.e. each column in the table represents a clock cycle. Note: Not all columns/clock cycles may be needed…leave them blank.

Memory Contents	*Address*	0	1	2	3	4	5	6	7
	MEM	10	8	7	6	3	5	1	11

I	X	0	1	2	3	4	5	5	6	7	7	8
STATE	**I**	**L**	**CM1**	**CM2**	**CM2**	**CM2**	**CM2**	**CM3**	**CM2**	**CM2**	**CM3**	**F**

d. Complete the next state logic and state memory connections.
   - First connect the appropriate signals to all the /CLR and /SET inputs to the flip-flops (Assume GND and VDD are available for connection).
   - Next use the blank area below the box to draw the logic for the D-inputs (DI, DL, DC, …, DF) to each flip-flop. Please write equations next to the inputs of the buffers driving the corresponding D-input rather than drawing gates (which gets too messy). Note: (i==15) is a signal that is generated for you and will be '1' when i equals 15, '0' otherwise



e. Now write logic equations for the D-Input of the following flops using AND/OR/NOT sybmols.

DCM1 = **QL**

DCM2 = **QCM1 + QCM3·(I != 7) + QM2·LT·(I != 7)**

DCM3 = **QCM2·(!LT)**

DF = **QCM2·(LT)·(I==7) + CM3·(I==7) + QF!·ACK**

f. Now write logic equations for the control signals to the datapath back on page 2:

IEN = QL + QCM1 + LT·QCM2 + QCM3

M1EN = LT(QCM1 + QCM2) + QL

M2EN = LT(QCM2 + QCM3) + QCM1

MSEL = LT (QCM1 + QCM2)

Any other control signals you added that we did not show in the datapath need to have their logic defined here:

CSEL = QCM3

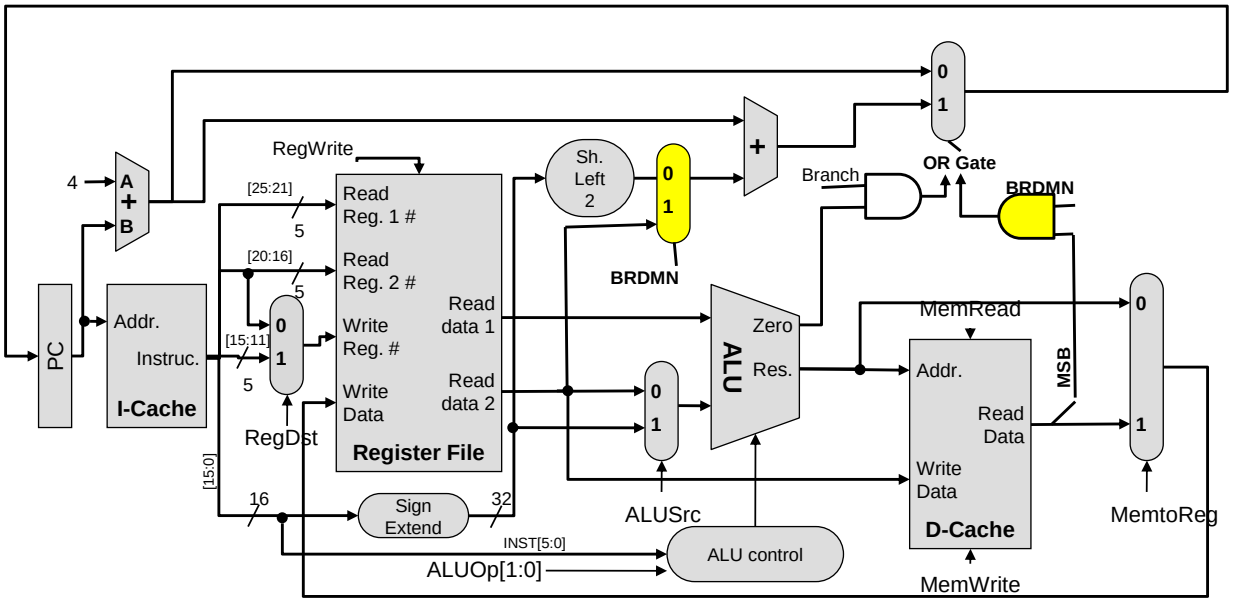
3. **(15 pts.) ISA and Single-Cycle CPU Datapath**: Recall the basic branch instruction. We want to add a new instruction 'BRDMN' (Branch using Register for Displacement if Memory value is Negative), while not affecting any other instructions. It loads a value given by the address in $rs (just like a LW but without an offset). If that value is negative then the PC should be updated with the sum of its current value plus four plus the value of $rt (PC = PC + 4 + $rt). Implement any changes to the datapath and control signals to support this new, 'brdmn' instruction on the single-cycle CPU. Assume when this instruction executes a BRDMN control signal will be '1'

```
BRDMN $rt,($rs)  instruction description:
if(M[rs] is neg.)   # load memory from addr $rs
                    #   and check if its negative
   PC = PC+4+$rt        # store PC+4+rt into PC
```

This instruction will use the machine code I-format:

brdmn:

opcode	rs	rt	Unused
6	5	5	16-bits **[Assume Will be Set to 0's]**



a. Sketch the additions/changes to the **datapath** above that would be needed to support this new instructions and its operations (provide a brief description below if the sketch is unclear):

We add one mux to choose the $rt value to add to PC+4 to produce the new target address and we update the Branch mux select logic to also trigger if BRDMN is true and the result from memory is negative (i.e. MSB = 1).

---

b. Show the control values of the following control signals to implement this new instruction.

	BRDMN	MemRead	MemWrite	MemToReg	RegWrite	Branch	ALUSrc
Value (0,1,X)	1	**1**	**0**	**X**	**0**	**0**	**1**

4. **(12 pts.) Performance, Pipelining, and Hazards**
Tommy Trojan decides that LW and SW rarely use offsets for address calculation and thus DO NOT need to use the ALU/adder to compute its effective address but instead just use the contents of the base register. He then proposes to swap the order of the EX and MEM pipeline stages.

Thus the pipeline order is now as shown in the table below along with the stage delay. Assume branches are NOT resolved using EARLY DETERMINATION in the DECODE stage but still require the ALU in the EX stage for comparison.

Fetch	Decode	MEM	EX/ALU	WB
10 ns	6 ns	10 ns	12 ns	5 ns

a. Tommy Trojan's partner, Miss Bruin, says that since SW and LW no longer use the EX/ALU stage that the clock cycle time of the pipelined processor can be reduced to only 10 ns. Do you agree? Explain your reasoning either way in 1-2 sentences.

No, the delay must be set for the worst case stage delay (i.e. EX/ALU which is still used by other instructions). The cycle time would remain at 12 ns.

b. Assume NO EARLY branch determination (i.e. branches are resolved in the stage AFTER the EX/ALU stage. This new organization will require flushing of how many instructions?

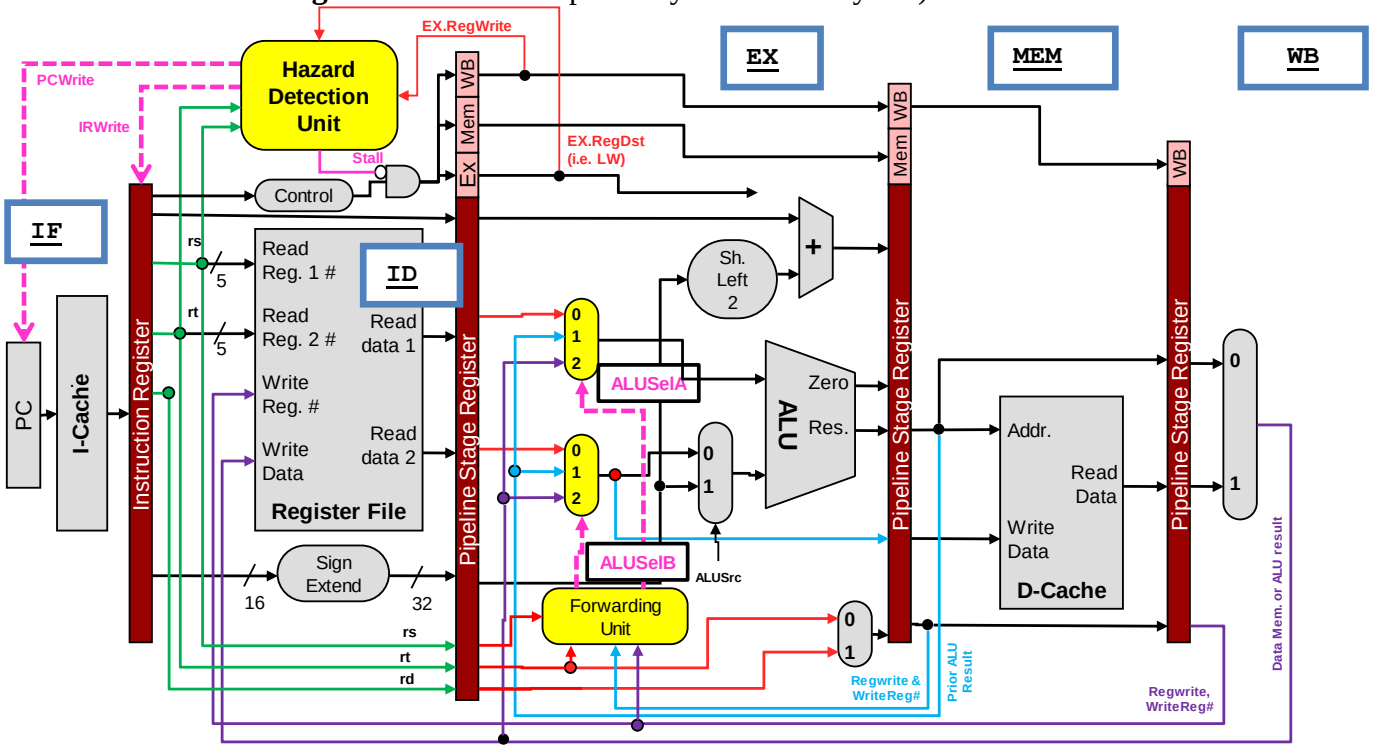
4 instructions (the branch updates the PC at the end of the WB stage requiring flush of everyone behind him in F, D, E, M)

c. In the original pipeline organization even with forwarding we required a stall when a LW was followed by a dependent instruction. Given the basic 4 instructions: **LW, SW, BEQ**, and **ALUType** list all the sequences of 2 instructions that would necessitate stall cycle(s) to be inserted (even with forwarding available), giving an instruction sequence example, and indicating how many stall cycles must be inserted by the hardware between the two instructions.

```
ADD $4, $0,$0
SW   $4, 0($3)
(insert 1 stall cycles)

ADD $4, $0,$0
LW   $3, 0($4)
(insert 1 stall cycles)
```

5. **(15 pts.) Pipelining**: Examine the 5-stage pipeline with forwarding unit in its original location (forwarding takes place in the EX stage) and LATE branch determination (MEM stage). Finally **assume internal forwarding register file.** Given the instruction sequence shown below, complete the time-space diagram below showing which stage each instruction is in for each clock cycle until 7 instructions have been completed (i.e. **STOP when the 7th instruction reaches the WB stage** and **DON'T** complete any more rows/cycles).




```
Instruc. Sequence
     lw  $2, 0($3)
     bne $2, $0, L3 (NT)
L2:  lw  $6, 0($7)
     add $6, $6, $5
     sw  $6, 0($7)
L1:  beq $2, $0, L2 (T,NT)
L3:  and $2, $2, $2
     lw  $5, 0($2)
     or  $2, $2, $2
```

T = Taken and should cause the CPU to go to the labeled instruction (eg. L1) rather than continuing sequentially

Cycle	IF	ID	EX	MEM	WB
1	LW				
2	BNE	LW			
3	LW	BNE	LW		
4	LW	BNE	X	LW	
5	ADD	LW	BNE	X	LW
6	SW	ADD	LW	BNE	X
7	SW	ADD	X	LW	BNE
8	BEQ	SW	ADD	X	LW
9	AND	BEQ	SW	ADD	X
10	LW	AND	BEQ	SW	ADD
11	OR	LW	AND	BEQ	SW
12	LW	X	X	X	BEQ
13	ADD	LW	X	X	X
14	SW	ADD	LW	X	X
15	SW	ADD	X	LW	X
16	BEQ	SW	ADD	X	LW
17					

6. **[15 pts.]** Consider the traditional pipeline organization shown in the previous question. Recall that originally we did not have the forwarding unit but only the hazard detection unit in the Decode stage which would simply stall the pipeline until the hazard resolved. We then added the forwarding unit to reduce stalls. **Again, remember the register file supports internal forwarding, with or without forwarding logic**.

```
ADD $t0,$t1,$t2
SUB $t3,$t0,$t4
```

a. [For this page, assume only the HDU; No forwarding logic]
Fill in the blanks below.
WITHOUT the forwarding unit an instruction immediately followed by a dependent instruction (see to the right) would require __2____ stall cycles.

```
ADD $t0,$t1,$t2
Indep. Instr.
SUB $t3,$t0,$t4
```

Similarly, if 1 or more independent instructions sit in between the producer (ADD) and dependent instruction (SUB) it would require _____ (**fewer / more / the same**) stall cycles. However if ___2___ or more independent instructions sit in between the producer and dependent instruction, then no stall cycles would be necessary.

Perform the calculations detailed below.
We can calculate the average CPI of a pipelined processor using the ideal CPI of 1 and then adding the average stall cycles an instruction incurs
(i.e. Average CPI = Ideal CPI + Average Stall Cycles = 1 + Average Stall Cycles).

What could a competitor company claim is the worst case CPI of this processor (again assuming only stalls due to data hazards)?

Worst case is every instruction is followed by a dependent instruction requiring 2 stall cycles for every 1 instruction finished:
ADD $2, $2, $2
ADD $2, $2, $2

Final answer for worst-case CPI = _1 + 2 stall cycles per instruction = 3____

Examining a few representative programs, we find the following relationships for dependent instructions:

Probabilities of an instruction being followed by a dependent instruction…	Probability
…Immediately (i.e. next instruction)	40%
…2 instructions later	20%
…3 instructions later	10%
…4 instructions later	10%
…More than 4 instructions later	5%
…Never (no dependent instruction)	15%

Assuming we only stall due to data dependencies calculate the Average CPI:
Average CPI = 1 + Average Stall cycles
Average Stall Cycles = 2*.4 + 1*.2 + 0*.4 = 1

Final answer for average CPI = ___1 + 1.0 = 2.0_____

b. Assume the following stage delays for the pipeline processor w/o forwarding:

IF	ID	EX	ME	WB
10 ns	8 ns	7 ns	10 ns	6 ns

Now assume the forwarding unit and muxes are added as shown on page 7. Suppose that the forwarding unit itself requires 3 ns of delay to produce ALUSELA and ALUSELB and that all muxes require 2 ns of delay once their inputs have arrived. What clock cycle time is appropriate for the processor to use when the forwarding unit and muxes are added?

EX stage time is now: 7 + 3 for Forwarding Unit + 2 for mux = 12 ns

Final answer for appropriate clock cycle time? _____**12 ns**_____

Using the probabilities of dependent instructions on the previous page and given an instruction mix as shown below, compute the average CPI for pipeline process with this newly added forwarding logic (again assume stalls only occur due to data hazards).

LW	30%	**ALU-Type**	40%
SW	20%	**Branches**	30%

Average CPI = 1 + P(LW)*P(LW followed by depend. Instruct)*1 stall cycle =
1 + (.3*.4*1) = 1.12

Final answer for Average CPI = __1.12

Determine the speedup (or speed-down) of the pipelined processor with forwarding logic that you analyzed on this page (with its CPI & cycle time) vs. the pipeline processor WITHOUT forwarding logic that you analyzed on this and the previous page). Assume they execute the same program and set of instructions.

Exec Time w/o Forwarding = IC * 2.0 * 10ns

Exec Time w/ Forwarding = IC * 1.12 * 12 ns =

Speedup = old / new = 2.0*10 / 1.12*12 = 20 / 13.44 = around 1.5

Proc. w/ forwarding logic [circle one] **(Speedup / Speeddown)** = _20/13.44=1.489 times