UNIVERSITY OF CALIFORNIA, LOS ANGELES UCLA

BERKELEY · DAVIS · IRVINE · LOS ANGELES · RIVERSIDE · SAN DIEGO · SAN FRANCISCO SANTA BARBARA · SANTA CRUZ

CS M151B / EE M116C

Midterm Exam

All work and answers should be written directly on these pages, use the backs of pages if needed.

This is an open book, open notes quiz – but you cannot share books or notes.

We will follow the departmental guidelines on reporting incidents of academic dishonesty – do not make us enforce the rules. Keep your eyes on your own exam!

NAME: Solution

ID: _____

Do not write anything in the area below on this page:

Problem 1: _____ (10)

Problem 2: _____ (40)

Problem 3: _____ (50)

Total: _____ (out of 100)

1. ***Tradeoffs (10 points)***:
   a. Assume that the single cycle datapath we covered in class executes an application which has a mix of 25% lw instructions, 15% sw instructions, 20% beq instructions, and 40% R-type instructions. Given this particular mix of instructions, what is the CPI of the single cycle datapath?

CPI: 1

   b. The pipelined datapath we covered in class may ultimately reach a steady state where one instruction is in each stage of the pipeline. Assuming no hazards, how many cycles would it take for the 5-stage pipeline we covered in class to reach the steady state?

Cycles: 5 or 4
(full pipeline or empty pipeline)

   c. Data hazards may be resolved in software through code reordering or nop insertion. Another option is to use stalls in hardware to avoid data hazards. Compare these two alternatives:

      i CPI would likely be smaller with the software approach compared to the hardware approach. (fill in the blank with the word ***larger*** or ***smaller***)

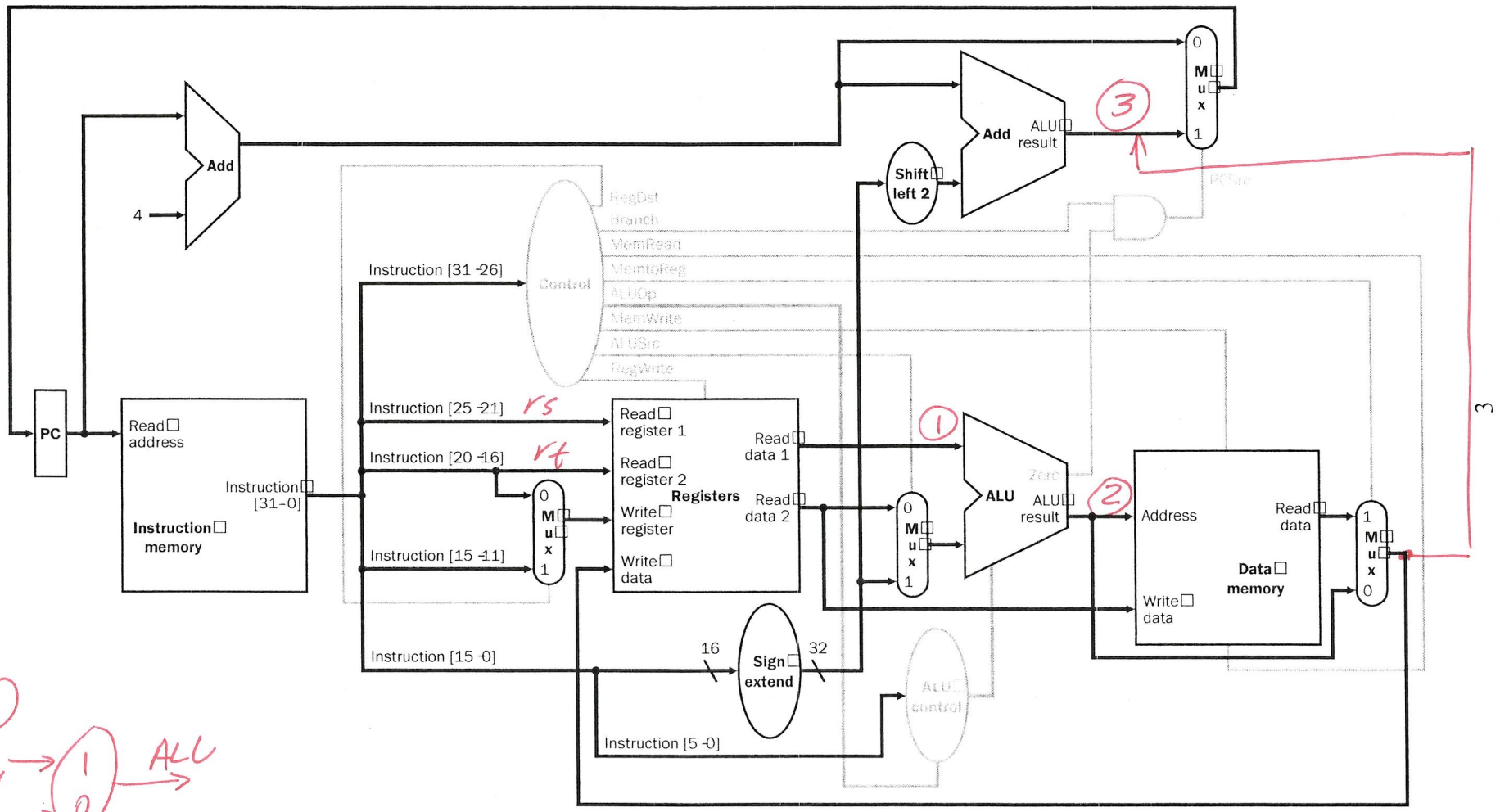
      ii Instruction count would likely be larger with the software approach compared to the hardware approach. (fill in the blank with the word ***larger*** or ***smaller***)

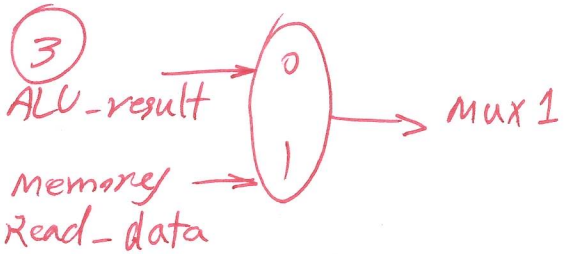
2. ***BEQM (40 points)***: Consider the single-cycle processor implementation. Your task will be to augment this datapath with a new instruction: the *beqm* instruction. This instruction will be an I-type instruction, and will have the following effect:

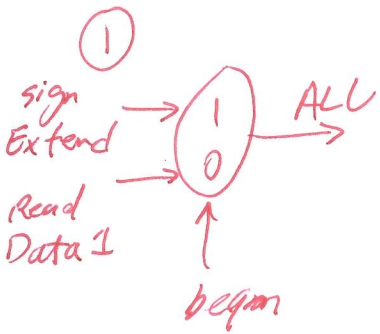
```
if (R[rt]==SE(I))
        PC=M[R[rs]]
else
        PC=PC+4
```

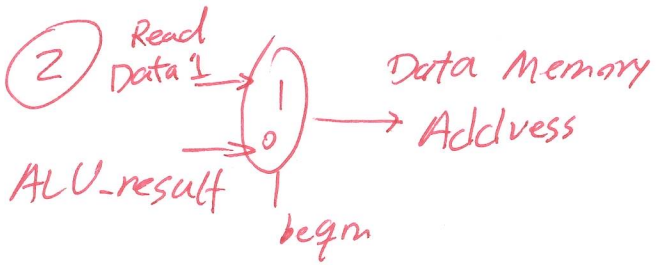
Implement your solution on the following two pages. All other instructions must still work correctly after your modifications. You should not add any new ALUs, register file ports, or ports to memory.

There are multiple solutions to this problem and all of them have been given full credit in case of correctness. Only one of them shown here:


Add
4
PC
Read address
Instruction [31-0]
Instruction memory
Instruction [31-26]
Control
RegDst
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Instruction [25-21]
rs
Instruction [20-16]
rt
Instruction [15-11]
Instruction [15-0]
Instruction [5-0]
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
0 M u x 1
Shift left 2
Add
ALU result
3
PCSrc
1
ALU
Zero
ALU result
2
Address
Read data
Data memory
Write data
16
Sign extend
32
ALU control
3


3
ALU_result
Memory
Read_data
0
1
Mux 1


1
sign
Extend
Read
Data 1
1
0
ALU
begin


2
Read
Data 1
1
0
Data Memory
Address
ALU_result
begin

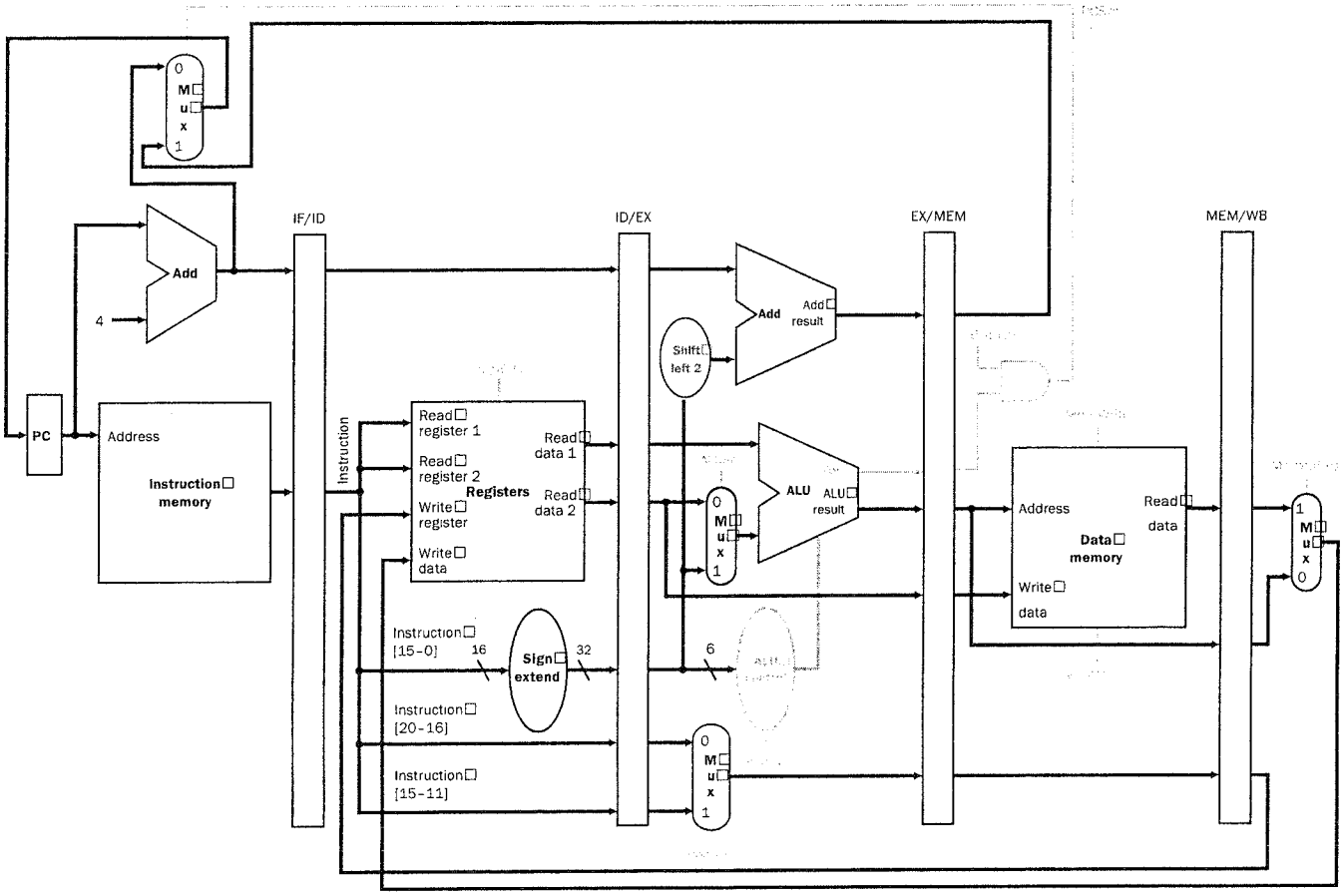
Main Controller

Input or Output	Signal Name	R-format	lw	sw	Beq	Beqm
Inputs	Op5	0	1	1	0	
	Op4	0	0	0	0	
	Op3	0	0	1	0	
	Op2	0	0	0	1	
	Op1	0	1	1	0	
	Op0	0	1	1	0	
Outputs	RegDst	1	0	X	X	X
	ALUSrc	0	1	1	0	0
	MemtoReg	0	1	X	X	X
	RegWrite	1	1	0	0	0
	MemRead	0	1	0	0	1
	MemWrite	0	0	1	0	0
	Branch	0	0	0	1	1
	ALUOp1	1	0	0	0	0
	ALUOp0	0	0	0	1	1
	Beqm	0	0	0	0	1

ALU Controller

Opcode	**ALUOp**	**instruction**	**function**	**ALU Action**	**ALUCtrl**
Lw	00	load word	XXXXXX	add	010
Sw	00	store word	XXXXXX	add	010
Beq	01	branch equal	XXXXXX	subtract	110
R-type	10	add	100000	add	010
R-type	10	subtract	100010	subtract	110
R-type	10	AND	100100	AND	000
R-type	10	OR	100101	OR	001
R-type	10	SLT	101010	SLT	111
Beqm	01	BEQM	XXXXXX	subtract	110

3. ***Pipelining (50 points)***: Consider the 5-stage scalar pipeline we have explored in class, shown below.



Branches are resolved in the MEM stage as shown in the figure above. Assume that instruction and data memory are perfect – i.e. accesses only take a single cycle. Further assume that the pipeline is extended to be able to handle *addi* and *j* instructions. These extensions still result in a 5-stage pipeline.

The following instruction sequence will be executed on this architecture:

```
THERE:   addi $t2, $t2, 16
         lw $t0, 4 ($t2)
         bne $t0, $s0, HERE
         lw $t1, 8 ($t2)
         j ADDIT
HERE:    lw $t1, 12 ($t2)
ADDIT:   addu $s2, $t1, $s2
         bne $s1, $t2, THERE
```

This sequence has two conditional branches (i.e. *bne's*). The *bne* to HERE is taken every other time it executes – i.e. the value loaded into $t0 by the first *lw* instruction is not equal to the value in $s0 50% of

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
addi $t2, $t2, 16	IF	ID	EX	MEM	WB																									
lw $t0, 4 ($t2)		IF	ID	E	M	WB																								
bne $t0, $s0, HERE			IF	ID	ID	E	M	WB																						
lw $t1, 12 ($t2)								IF	ID	E	M	WB																		
addu $s2, $t1, $s2									IF	ID	ID	E	M	WB																
bne $s1, $t2, THERE											IF	ID	E	M	WB															
addi $t2, $t2, 16															IF	ID	E	M	WB											
lw $t0, 4 ($t2)																IF	ID	E	M	WB										
bne $t0, $s0, HERE																	IF	ID	ID	E	M	WB								
lw $t1, 8 ($t2)																			IF	ID	E	M	WB							
j ADDIT																				IF	ID	E	M	WB						
addu $s2, $t1, $s2																								IF	ID	E	M	WB		
bne $s1, $t2, THERE																									IF	ID	E	M	WB	

the time. You may assume that the first time the *bne* to HERE is encountered, the branch is taken. The next time it will be not taken, and so on.

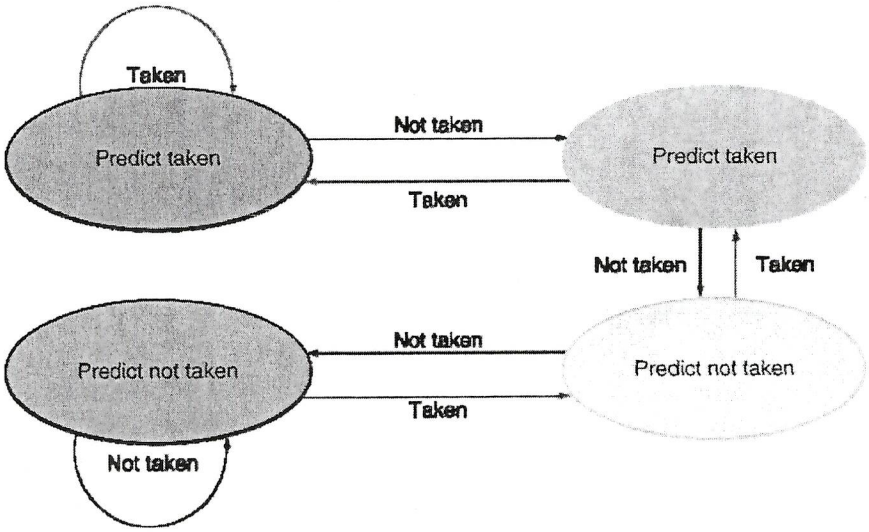
The *bne* to THERE is taken once and then not taken – so it will be executed two times all together. You only need to track the second instance of the *bne* to THERE, and do not need to worry about the instruction after that second execution of the *bne* to THERE.

I suggest that you take some time to understand what exactly the instruction sequence is doing before continuing with the rest of the problem.

a. First assume that data hazards are handled with full forwarding and branch hazards are handled by a static prediction where all branches (including jumps) are predicted not taken. This simplifies the hardware design since we do not need to worry about early determination of the branch target. We are going to determine how many cycles it will take to finish the second instance of the *bne* to THERE. Fill in the table on the following page to show when each instruction is in each stage of the pipeline. The first instruction has been done for you. Make sure you show your work in the table.

Total # of cycles: 29 *(40 points)*

b. Now let's try a dynamic branch predictor. Assume the 2-bit dynamic branch predictor from class:



The branch predictor is accessed in the instruction fetch stage to determine whether a particular branch is taken or not taken. Assume that we have some other structure to predict whether an instruction is a branch in the instruction fetch stage, and to what target address the branch may go.

The branch predictor will have eight entries – each entry will hold two bits. The two bits will represent one of the four states of the above FSM – 11 represents the strongly biased predict taken state in the upper left, 10 represents the predict taken state in the upper right, 01 represents the predict not taken state in the lower right, and 00 represents the predict not taken state in the lower left. The branch predictor initially has the following entries:

PC		index	
500	←	0	01
504	←	1	10
508	←	2	00
512	←	3	01
516	←	4	11
520	←	5	00
524	←	6	01
528	←	7	10

The predictor is indexed as: (PC >> 2) % 8. Where % is the modulo operation. The *addi* instruction is stored at address 500. For the two instances of *bne* instruction to THERE, will the predictor correctly predict the branch direction?

1st instance of the bne to THERE: correct (correct or incorrect?) (*5 points*)

2nd instance of the bne to THERE: incorrect (correct or incorrect?) (*5 points*)