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

Before you start, make sure you have all 13 pages attached to this cover sheet.

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: ______

ID: ______

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

Problem 1: 10 (12)

Problem 2: 8 (8)

Problem 3: 17.5 (20)

Problem 4: 20 (20)

Problem 5: 10 (20)

Problem 6: 17 (20)

Total: 83 (out of 100)

10/12

1. ***Flipping Through Your Notes? (12 points):*** For the following implementation choices, list ONE benefit and ONE drawback to the given choice.

use	*instead of*	*comments*
EXAMPLE		
adds and shifts	**multiplies**	just benefit & drawback of 1st column over 2nd column was fine here
benefit: lower CPI		
drawback: higher instruction count		
CISC ✓	**RISC**	
benefit: more instructions	more parallelism	
drawback: less pipelining and parallelism ✗	more inst for RISC fewer instructions	
Fixed Length ISA	**Variable Length ISA**	
benefit: easy fetch and decode ✓	more flexible and compact instruction set	
drawback: instruction bits are scarce ✓	require multi-step fetch and decode	
Booth's algorithm	**3rd version of multiply algorithm**	
benefit: ✗ greater speed - dependson case	less complex hardware	
drawback: more hardware required to find strings of 1's. ✓	slower if there are strings of 1's.	
Multicycle Datapath	**Single Cycle Datapath**	
benefit: faster instructions actually execute faster ✓	simpler design	
drawback: more complex control. ✓	all instructions take the same time to execute as the longest instruction.	
Ripple Carry Adder ✓	**Partial Carry Lookahead Adder**	
benefit: simple hardware	better speed	
drawback: Long latency ✓	more complex hardware	
2-Address Code ✓	**3-Address Code**	**(Assume your ISA supports one or the other, but not both)**
benefit: shorter Instrs use less memory.	✓ more flexibility in where you store instruction results	
drawback: less flexibility in instructions ✓	possibly longer instructions or less instruction available bits for other things	

8/8

2. ***What's Happening Now (8 points):*** Consider the following code sequence:

```
lw $t4, 8 ($s1) 5
add $t4, $s2, $t4 4
sw $t4, 8 ($s1) 4
```

a. If we executed this on the single cycle datapath, what would be happening in the 3rd cycle of execution?

(sw) The data at R[$t4] is stored in M[R[$s1]+8]

✓

b. If we executed this on the multicycle datapath, in what cycle would the sw actually write to memory?

✓ $\boxed{13}$

3. ***Got MAD? (20 points)***: This problem will make use of the multicycle datapath, using the design we covered in class. Suppose that you are targeting a specific application where the instructions are broken down as follows: 20% loads, 15% beq, 15% stores, 50% R-type.

   a. First, calculate the CPI for this application running on the multicycle datapath:

$$CPI = .2\overset{lw}{(5)} + .15\overset{Beq}{(3)} + .15\overset{sw}{(4)} + .5\overset{R\text{-type}}{(4)} = \boxed{4.05}$$

$\frac{\text{cycles}}{\text{Instr}}$

✓

Now let's add a new instruction to this architecture. This instruction, the memory add, will be an R-type instruction that works as follows:

mad $t1, $t2, $t3

$t1 = $t2 + M[$t3]

NOTE – we are using register indirect addressing here for the second operand

We will replace every instance of

lw a, 0(b) 5
add x, a, c 4

in the application with

mad x, b, c 5

where a, b, c, and x are registers. So we are putting two instructions in place of one instruction whenever possible. Assume that the MAD instruction takes 5 cycles.

   b. If 50% of loads can make use of this optimization, calculate the new CPI for this application workload and machine:

Say orig IC = 100

lw	20	
beq	15	
sw	15	
R-type	50	

.5(20) = 10 opts

~~Save 4 in~~
Save 1 instr per opt.

IC new = 100 − 10 = 90

new IC

			%
lw	10	/90	= .111
beq	15		= .166
sw	15		= .166
R-type	40		= .444
mad	10		= .111

✓

$$CPI_{new}\ \overset{MAD}{.111(5)} + \overset{LW}{.111(5)} + \overset{beq}{.166(3)} + .166(4) + .444(4)$$

$$= 4.0555 = \boxed{4.06}$$

10

Of course, we cannot really make a complete comparison between these two approaches without using execution time. Assume that the complexity of this optimization increases the cycle time by 10%.

c. Calculate the percent speedup in execution time from this optimization: (if the optimization results in a slowdown, express this as a negative speedup).

$G = $ CLK speed

$ET\ \cancel{orig} = IC \times CPI \times CCT$

$ET_{OLD} = IC \times 4.05 \times \frac{1}{1G}$ $\quad 4.05\frac{IC}{G}$

$ET_{new} = .9\,IC \times 4.06 \times \frac{.9}{\cancel{1.10}G} \approx$ [illegible] $\frac{IC}{G}$

$\text{Speedup} = \frac{ET_{OLD}}{ET_{New}} = 1.0$[illegible] $= \boxed{03\%\ \text{speedup}}$

+2.5

$\text{Speedup} = 1 - \frac{ET_{new}}{ET_{old}}$

By way of comparison, calculate the CPI for the single cycle datapath with and without the MAD instruction:

d. Single cycle datapath $CPI_{original}$: 1 Single cycle datapath CPI_{mad}: 1 ✓

7.5

4. ***Don't Get MAD, Get Even (20 points)***: Implement the memory add instruction on the **single cycle datapath.** Use the R-type instruction format. Implement your solution on the following two pages.

For the example instruction :

mad $t1, $t2, $t3

20

which has functionality :

$t1 = $t2 + M[$t3]

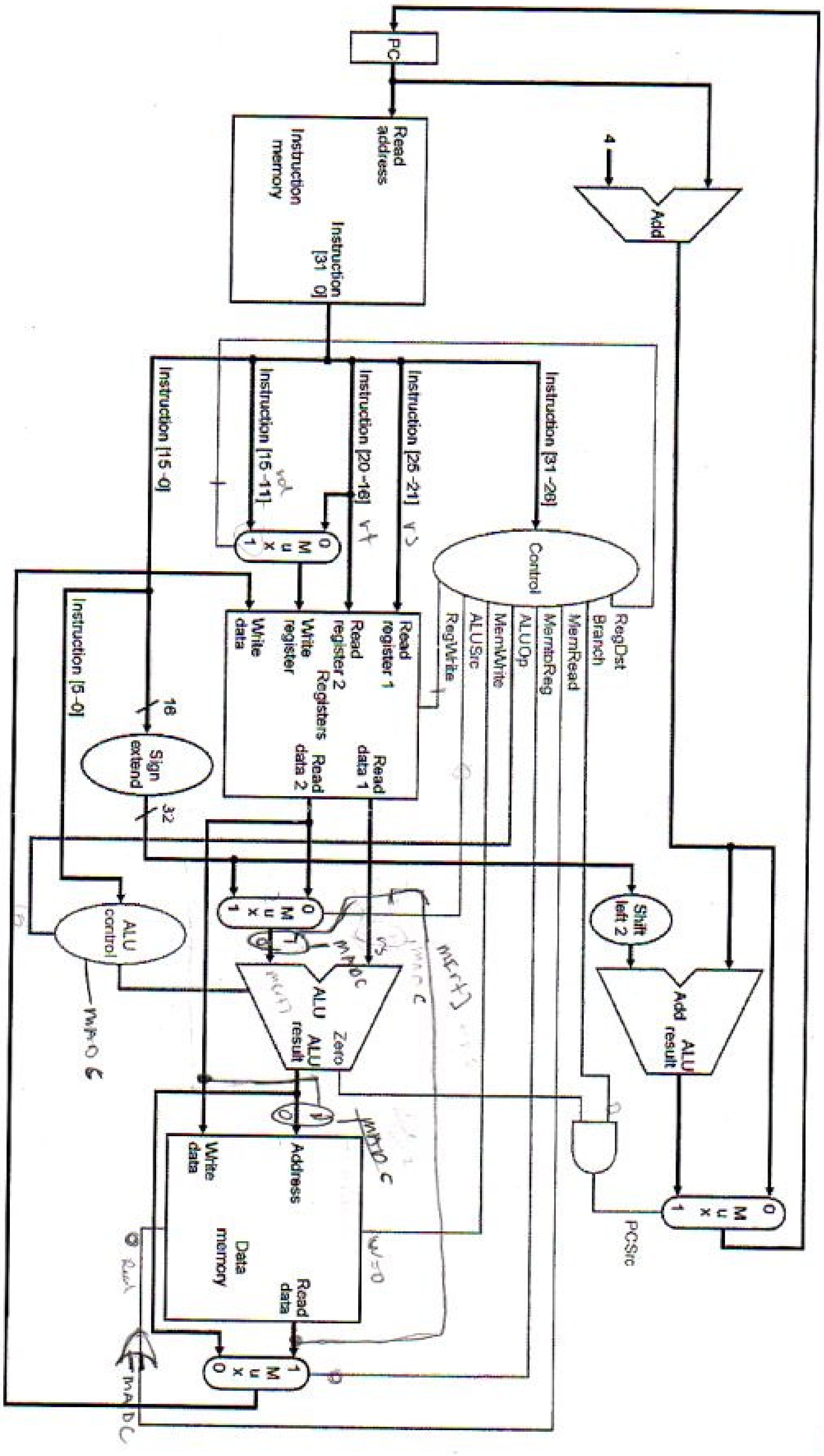
the R-type fields would map as follows: $t1 would be in rd, $t2 would be in rs, and $t3 would be in rt.

All other instructions must still work correctly after your modifications. You should **not** add any new ALUs, register file ports, or ports to memory.

$t1 = $t2 + M[$t3]

$\uparrow$ rd $\quad$ $\uparrow$ rs $\quad$ $\uparrow$ rt

R[rd] ← R[rs] + M[rt]



Main Controller

Input or Output	Signal Name	R-format	lw	sw	Beq
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
	ALUSrc	0	1	1	0
	MemtoReg	0	1	X	X
	RegWrite	1	1	0	0
	MemRead	0	1	0	0
	MemWrite	0	0	1	0
	Branch	0	0	0	1
	ALUOp1	1	0	0	0
	ALUOp0	0	0	0	1

ALU Controller

opcode	**ALUOp**	**instruction**	**function**	**ALU Action**	**ALUCtrl**	MADC
lw	00	load word	XXXXXX	add	010	0
sw	00	store word	XXXXXX	add	010	0
beq	01	branch equal	XXXXXX	subtract	110	0
R-type	10	add	100000	add	010	0
R-type	10	subtract	100010	subtract	110	0
R-type	10	AND	100100	AND	000	0
R-type	10	OR	100101	OR	001	0
R-type	10	SLT	101010	SLT	111	0
R-type	10	MAD	111111	add	010	1

✓

5. ***Branching Out (20 points)***: Use the multicycle datapath to implement the ***memory jump and link (mjl)*** instruction. This instruction uses the I-type instruction format, and has the following behavior:

   rt rs

   mjl $s3, $s4 (20) Imm

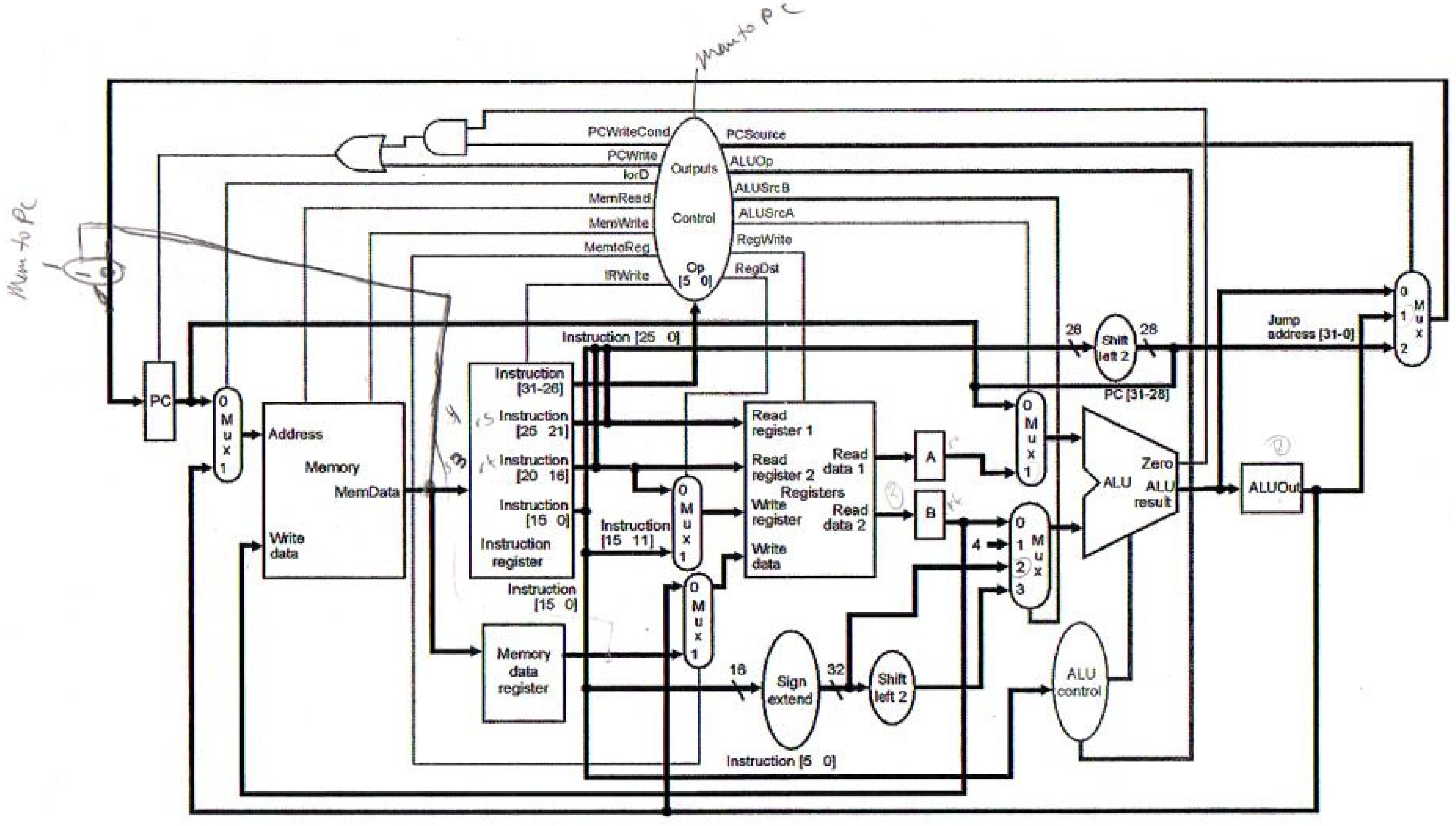
   will do the following:

   R[*$s3*] = *PC+4*
   PC = M[$s4+20]
   R[$s4]

Note that this instruction uses base+displacement addressing (i.e. we use the value in register $s4 plus the immediate value 20). Further note that we are storing PC+4 into $s3 **before** we modify the PC in the next line.

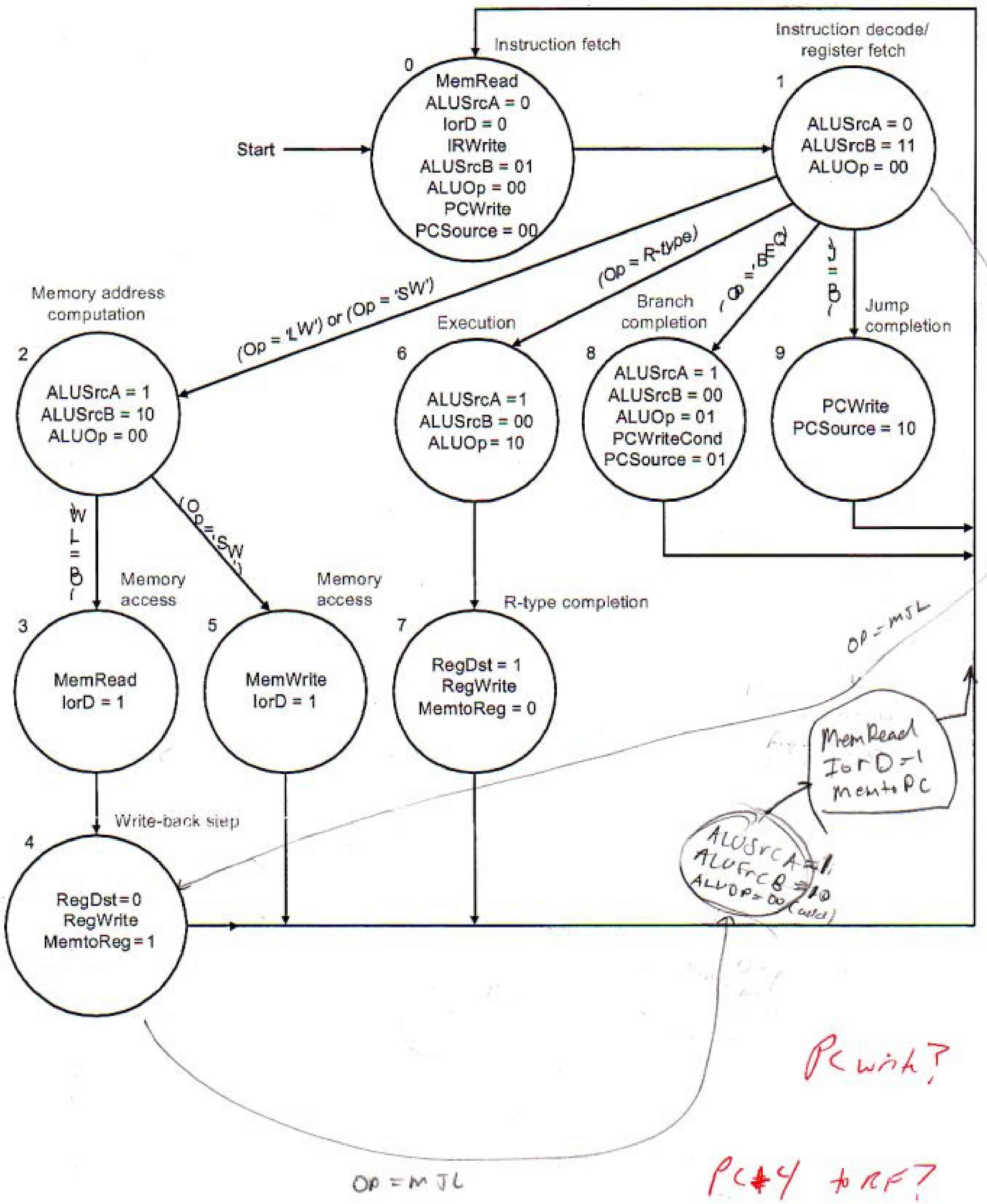
For the I-type format, the rs field will give the base address (i.e. $s4 above), the immediate field will give the displacement (i.e. 20 above), and the rt field will give the register to store PC+4 (i.e. $s3 above).

Implement this instruction 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.



R[rt] = PC+4
PC = M[RS+I]

R[$s3] = PC+4
PC = M[$s4+20]
rs I
immediate


Start
0
Instruction fetch
MemRead
ALUSrcA = 0
IorD = 0
IRWrite
ALUSrcB = 01
ALUOp = 00
PCWrite
PCSource = 00
1
Instruction decode/ register fetch
ALUSrcA = 0
ALUSrcB = 11
ALUOp = 00
(Op = 'LW') or (Op = 'SW')
(Op = R-type)
(Op = 'BEQ')
(Op = 'J')
2
Memory address computation
ALUSrcA = 1
ALUSrcB = 10
ALUOp = 00
6
Execution
ALUSrcA = 1
ALUSrcB = 00
ALUOp = 10
8
Branch completion
ALUSrcA = 1
ALUSrcB = 00
ALUOp = 01
PCWriteCond
PCSource = 01
9
Jump completion
PCWrite
PCSource = 10
(Op = 'LW')
(Op = 'SW')
3
Memory access
MemRead
IorD = 1
5
Memory access
MemWrite
IorD = 1
7
R-type completion
RegDst = 1
RegWrite
MemtoReg = 0
4
Write-back step
RegDst = 0
RegWrite
MemtoReg = 1
Op = MJL
ALUSrcA = 1
ALUSrcB = 10
ALUOp = 00 (add)
MemRead
IorD = 1
MemtoPC
Op = MJL
PC write?
PC+4 to RF?

6. ***Your Problems Are Multiplying (20 points):*** Consider the 2-bit Booth's Algorithm that you did on your homework. Assume that shifts take S seconds and adds/subtracts take A seconds.
   a. Assume that we are using 16-bit numbers. What is the worst case latency from this version of Booth's algorithm? Express in terms of A and S, and assume that shifting by two is the same cost as shifting by one – still S seconds. Further assume that you already have the multiplicand and the multiplicand<<1 stored in temporary registers before the start of the algorithm.

~~0101010101010101~~

Since shift by 2 each time:
Assume there is an add/sub for each shift

$8(S+A)$

$\boxed{8S+8A}$

✓

Now, we will try and quantify A and S a little further. We will look at a 16-bit ALU and a 16-bit right shifter. Assume that a multiplexor has delay **2T**. However, your design template has trouble with AND/OR/XOR gates that use more than two inputs.

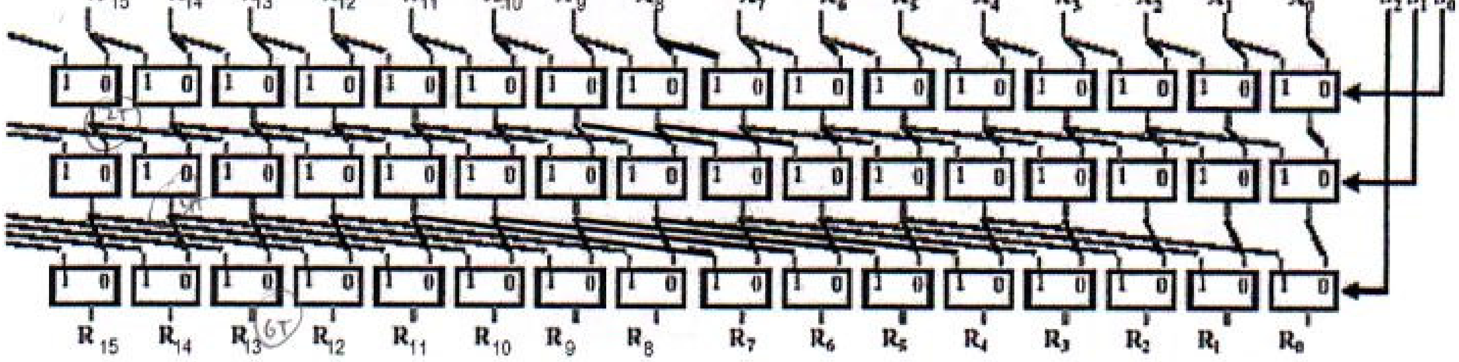
Use the following table to get the delay for AND/OR/XOR gates wih different # of inputs:

# of inputs	Delay
2	T
3	2T
4	3T
5	4T
6	5T

So the delay of a 2 input OR gate would be **T**, a 3 input OR gate would be **3T**, a 4 input OR gate would be **3T**, and a 5 input OR gate would be **4T**.

You must use these delay values for all parts of this question. Express all answers in terms of T. SHOW YOUR WORK ON THE DIAGRAMS!

b. What is the delay of this 16-bit shifter?


A_{15} A_{14} A_{13} A_{12} A_{11} A_{10} A_9 A_8 A_7 A_6 A_5 A_4 A_3 A_2 A_1 A_0
S_2 S_1 S_0
1 0
2T
4T
6T
R_{15} R_{14} R_{13} R_{12} R_{11} R_{10} R_9 R_8 R_7 R_6 R_5 R_4 R_3 R_2 R_1 R_0

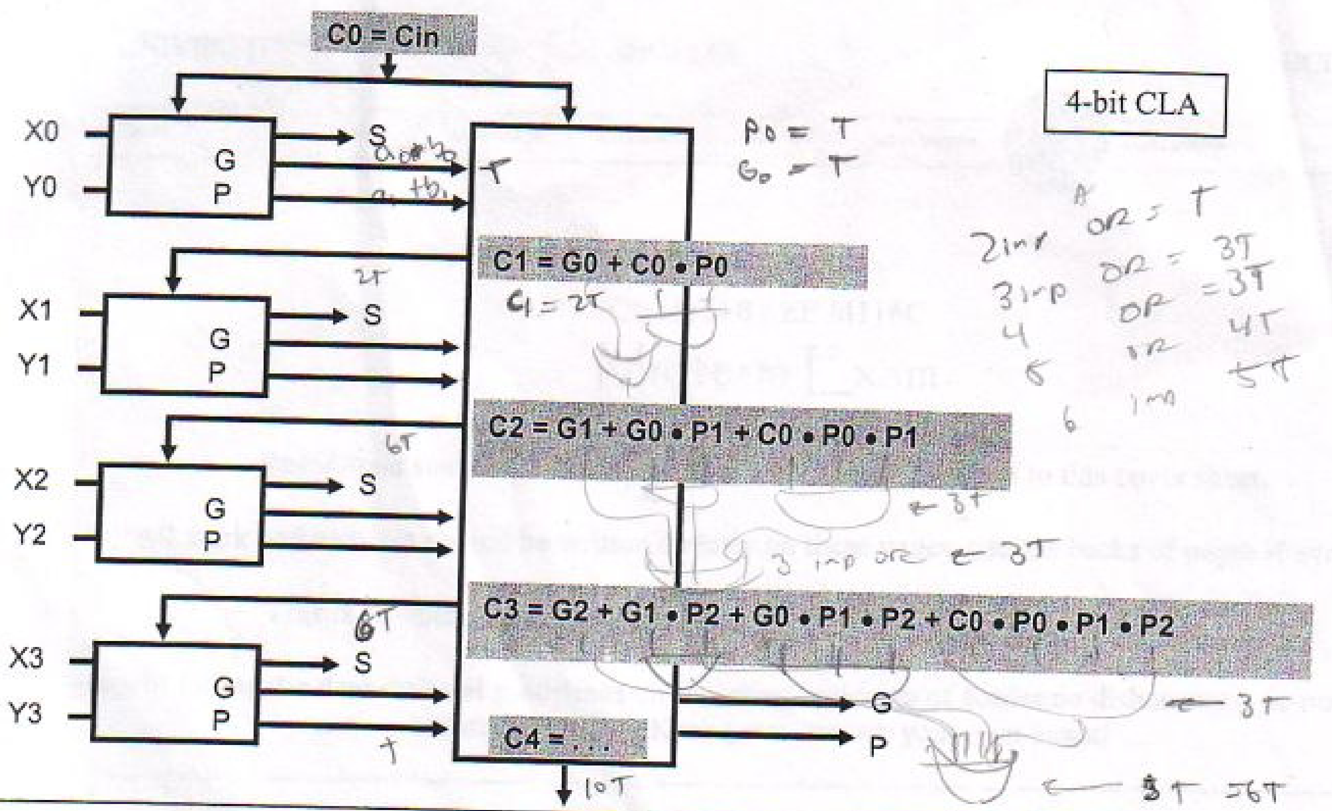
$2T + 2T + 2T = \boxed{6T}$

✓

c. On the next page we have the same 16-bit hierarchical carry lookahead adder (CLA) from class. Using the same delay values as above, what is the delay to get the sum bits for this adder?

X 14T

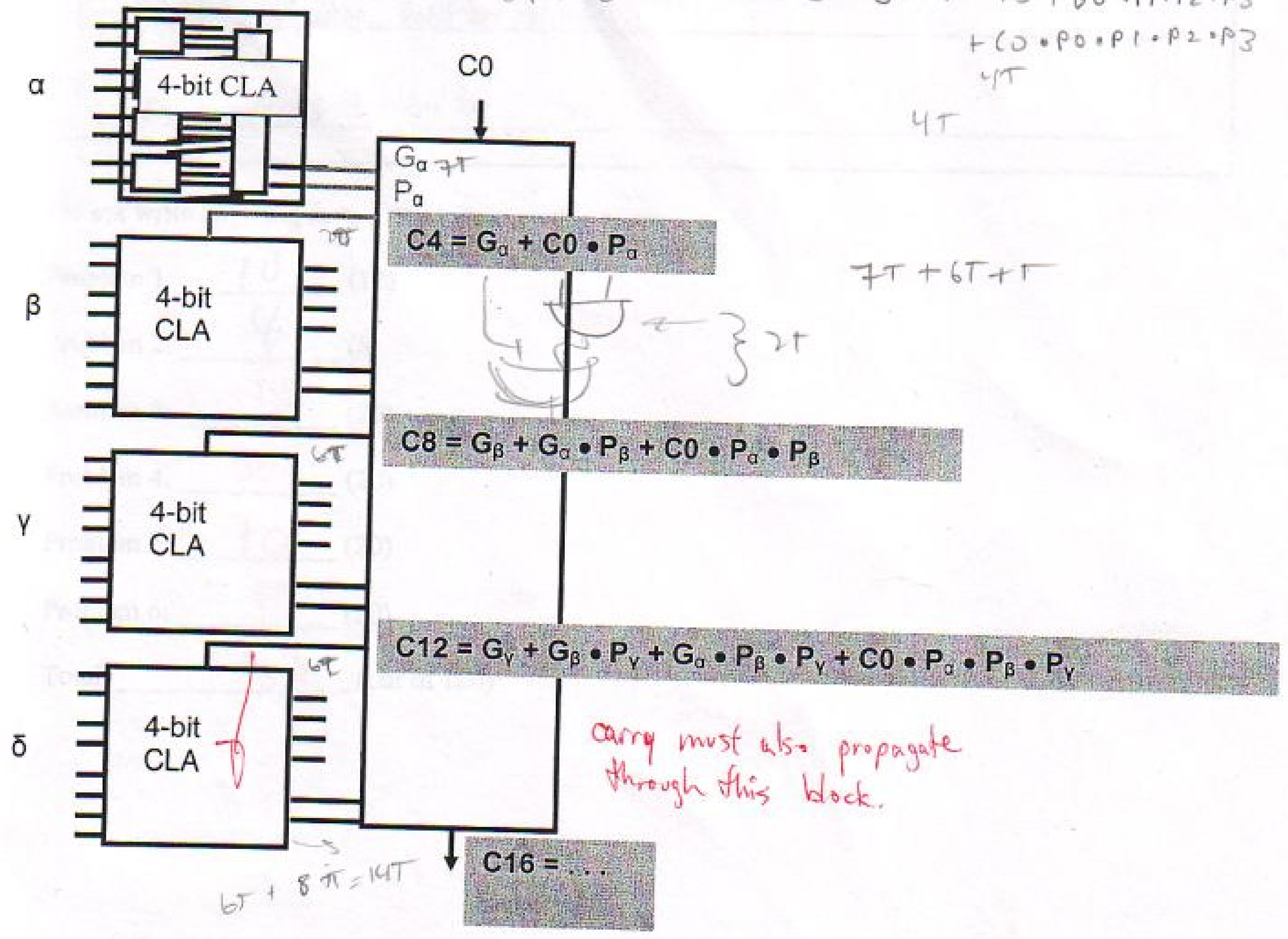
-3



C4 = G3 + G2 • P3 + G1 • P2 • P3 + G0 • P1 • P2 • P3 + C0 • P0 • P1 • P2 • P3

4T

4T


α
4-bit CLA
β
4-bit CLA
γ
4-bit CLA
δ
4-bit CLA
C0
Gα 7T
Pα
C4 = Gα + C0 • Pα
C8 = Gβ + Gα • Pβ + C0 • Pα • Pβ
C12 = Gγ + Gβ • Pγ + Gα • Pβ • Pγ + C0 • Pα • Pβ • Pγ
C16 = . . .
7T
6T
6T
2T
7T + 6T + T
6T + 8T = 14T
carry must also propagate through this block.