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**	
benefit: lower CPI		
drawback: higher instruction count		
CISC	**RISC**	
benefit:	lower instruction count	
drawback:	Complex instructions can have larger CPI	
Fixed Length ISA	**Variable Length ISA**	
benefit:	Simplifies decoding and fetch	
drawback:	limited instruction bits	
Booth's algorithm	**3rd version of multiply algorithm**	
benefit:	handles signed multiplication	
drawback:	longer worst case delay	
Multicycle Datapath	**Single Cycle Datapath**	
benefit:	better clock rate	
drawback:	larger CPI	
Ripple Carry Adder	**Partial Carry Lookahead Adder**	
benefit:	simple implementation, less hardware	
drawback:	longer latency	
2-Address Code	**3-Address Code**	**(Assume your ISA supports one or the other, but not both)**
benefit:	more bits for immediates and opcodes	
drawback:	One source operand must be the destination → less reuse of source operands	

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

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

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

sw $t4, 8 ($s1)

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

13th

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:

$$.2 \times 5 + .65 \times 4 + .15 \times 3 =$$

$$1.0 + 2.6 + .45 = 4.05$$

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)
add x, a, c

in the application with

mad x, b, c

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:

$$\frac{.20}{90} \times 5 + \frac{.55}{90} \times 4 + \frac{15}{90} \times 3 =$$

$$\frac{100 + 220 + 45}{90} = 4.06$$

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).

$$ET_{NEW} = 4.06 \times (.9)(IC) \times (1.1)(CT)$$

$$ET_{OLD} = 4.05 \times IC \times CT$$

speed(new) = 1/ETnew
speed(old) = 1/ETold

speedup = speed(new)/speed(old) - 1
= ETold/ETnew -1

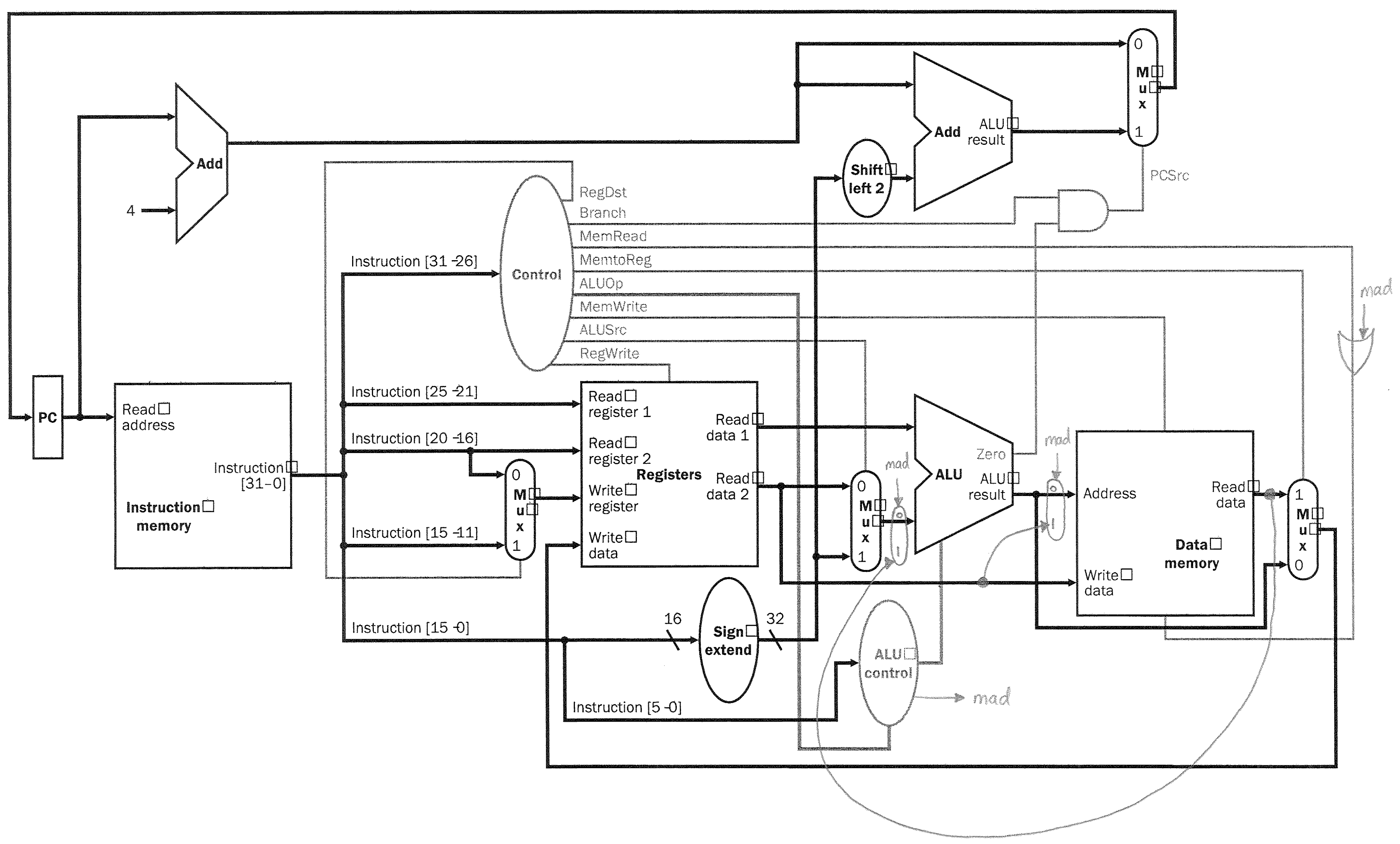
speedup = 1%

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

d. Single cycle datapath $CPI_{original}$: 1.0 Single cycle datapath CPI_{mad}: 1.0

#4)

mad : $R[rd] \leftarrow R[rs] + M[R[rt]]$



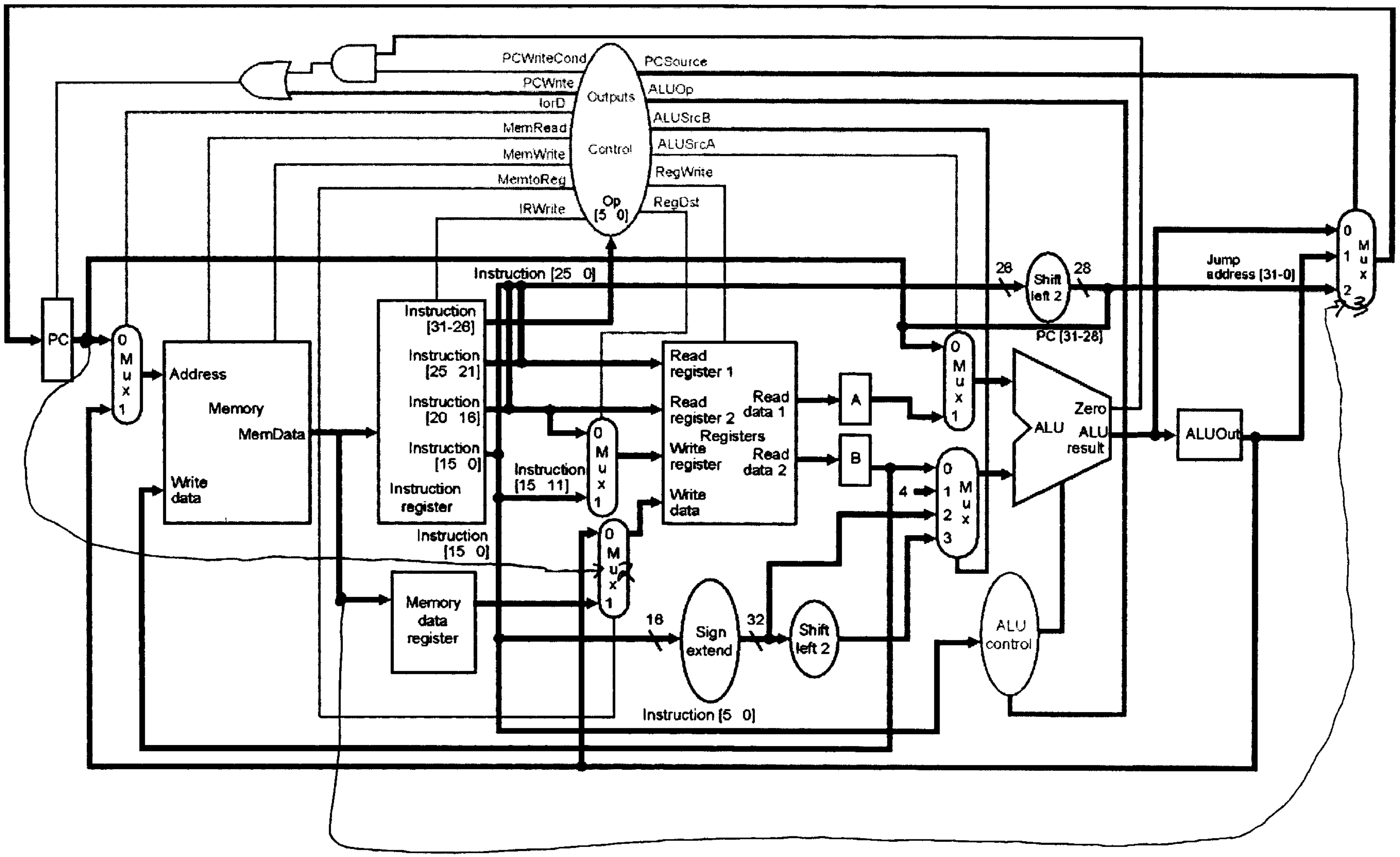
#4

Main Controller

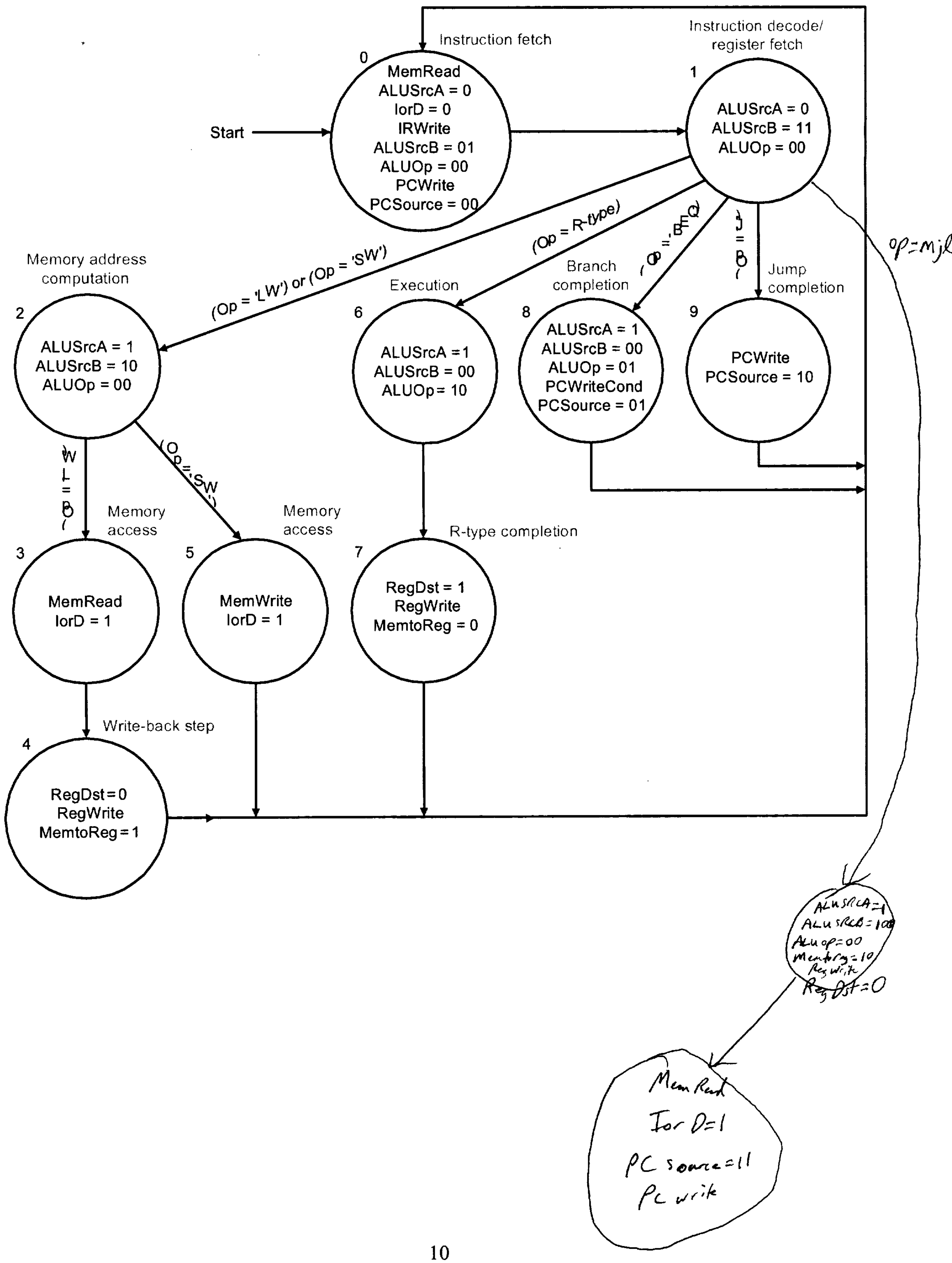
Input or Output	Signal Name	R-format	lw	sw	Beq
Inputs	Op5	0	1	1	0
	Op4	0	0	0	0
	Op3	0	0	1	0
	Op2	0	0	0	1
	Op1	0	1	1	0
	Op0	0	1	1	0
Outputs	RegDst	1	0	X	X
	ALUSrc	0	1	1	0
	MemtoReg	0	1	X	X
	RegWrite	1	1	0	0
	MemRead	0	1	0	0
	MemWrite	0	0	1	0
	Branch	0	0	0	1
	ALUOp1	1	0	0	0
	ALUOp0	0	0	0	1

ALU Controller

opcode	**ALUOp**	**instruction**	**function**	**ALU Action**	**ALUCtrl**	mad
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	110000	add	010	1



PCWriteCond
PCWrite
IorD
MemRead
MemWrite
MemtoReg
IRWrite
Outputs
Control
Op [5 0]
PCSource
ALUOp
ALUSrcB
ALUSrcA
RegWrite
RegDst
PC
0 Mux 1
Address
Memory
MemData
Write data
Instruction [31-26]
Instruction [25 21]
Instruction [20 16]
Instruction [15 0]
Instruction register
Instruction [25 0]
Instruction [15 11]
Instruction [15 0]
Memory data register
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
A
B
4
0 1 2 3 Mux
ALU
Zero
ALU result
ALUOut
ALU control
16
Sign extend
32
Shift left 2
26
Shift left 2
28
PC [31-28]
Jump address [31-0]
0 1 2 Mux
3
Instruction [5 0]


Start
Instruction fetch
0
MemRead
ALUSrcA = 0
IorD = 0
IRWrite
ALUSrcB = 01
ALUOp = 00
PCWrite
PCSource = 00
Instruction decode/ register fetch
1
ALUSrcA = 0
ALUSrcB = 11
ALUOp = 00
(Op = 'LW') or (Op = 'SW')
(Op = R-type)
(Op = 'BEQ')
(Op = 'J')
op=mjl
Memory address computation
2
ALUSrcA = 1
ALUSrcB = 10
ALUOp = 00
Execution
6
ALUSrcA = 1
ALUSrcB = 00
ALUOp = 10
Branch completion
8
ALUSrcA = 1
ALUSrcB = 00
ALUOp = 01
PCWriteCond
PCSource = 01
Jump completion
9
PCWrite
PCSource = 10
(Op = 'LW')
(Op = 'SW')
Memory access
3
MemRead
IorD = 1
Memory access
5
MemWrite
IorD = 1
R-type completion
7
RegDst = 1
RegWrite
MemtoReg = 0
Write-back step
4
RegDst = 0
RegWrite
MemtoReg = 1
ALUSrcA=1
ALUSrcB=100
ALUop=00
Memtoreg=10
RegWrite
RegDst=0
MemRead
IorD=1
PC source=11
PC write

/

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.

(out of 5 points)

Option A

There are 8 iterations each iteration in the worst case consists of an add and a shift by two

∴ worst case delay is

$8(A+S)$

Option B

the last shift is unnecessary so

delay $= 8A + 7S$

Though option B is correct, option A was not marked wrong.

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:

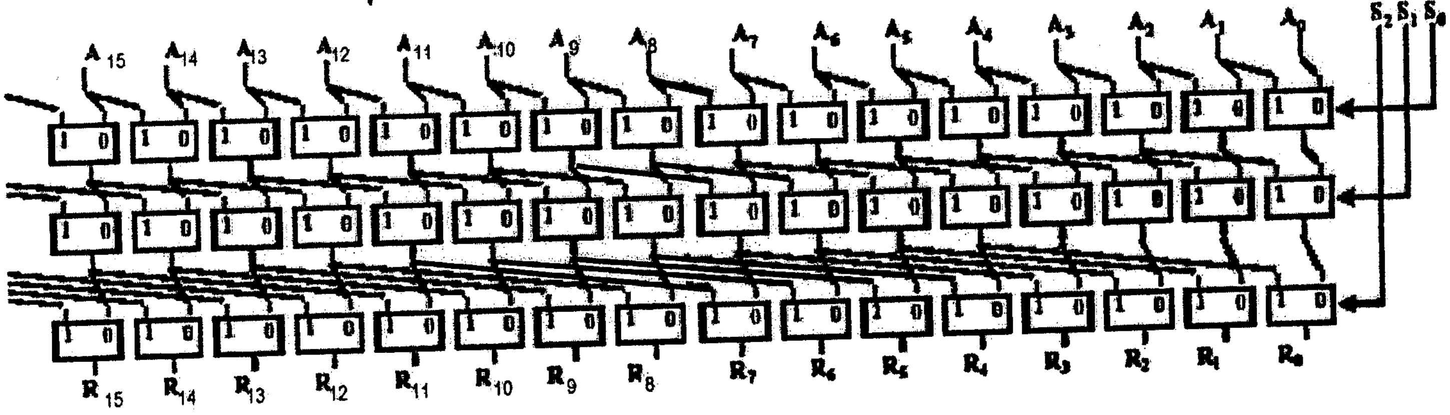
# of inputs	Delay
2	T
3	2T
4	3T
5	4T
6	5T

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?

(out of 5 points)



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
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

Each multiplexor has delay 2T

A_x signals must pass through 3 multiplexors $\therefore$ delay $= 3 \times 2T = 6T$

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?

6c) (out of 10 points)

Delay in CLA α

$G_\alpha = G_3 + G_2P_3 + G_1P_2P_3 + G_0P_1P_2P_3$ delay G_α = 1T + 3T + 3T = 7T

$P_\alpha = P_0P_1P_2P_3$ delay P_α = 1T + 3T = 4T

At heirarchical level: 1 there are two possible answers

Option B is correct though option A was not penalized

Option A

$C_4 = G_\alpha + C_0P_\alpha$ (adds 2T additional delay)

delay C_4 = 2T + 7T = 9T

$C_8 = G_\beta + G_\alpha P_\beta + C_0P_\alpha P_\beta$ (adds 4T delay)

delay C_8 = 4T + 7T = 11T

$C_{12} = G_\gamma + G_\beta P_\gamma + G_\alpha P_\beta P_\gamma + C_0P_\alpha P_\beta P_\gamma$ (adds 6T delay)

delay C_{12} = 6T + 7T = 13T

$C_{16} = G_\delta + G_\gamma P_\delta + G_\beta P_\gamma P_\delta + G_\alpha P_\beta P_\gamma P_\delta + C_0P_\alpha P_\beta P_\gamma P_\delta$ (adds 8T delay)

delay C_{16} = 8T + 7T = 15T

Option B

Since delay G_α > delay P_α we ignore the contribution of the propagation chain

delay C_4 = T + 7T = 8T

delay C_8 = 3T + 7T = 10T

delay C_{12} = 5T + 7T = 12T

delay C_{16} = 7T + 7T = 14T

Delay in CLA δ block

$C_{15} = G_{14} + G_{13}P_{14} + G_{12}P_{13}P_{14} + C_{12}P_{12}P_{13}P_{14}$

delay C_{15} = 3T + delay C_{12} + 3T
= 19T

delay C_{15} = 6T + 12T = 18T

calculation of S_{15} bit

if assume 3 input XOR gate delay S_{15} = 2T + 19T = $\boxed{21T}$

if assume two 2 input XOR gate delay S_{15} = T + 19T = $\boxed{20T}$

delay S_{15} = 2T + 18T = $\boxed{20T}$

delay S_{15} = T + 18T = $\boxed{19T}$

These are the three possible unpenalized answers

It is interesting to note that if you limit the implementation to <u>just</u> two input logic gates then delay of 14T can be obtained (see section 2 midterm answer)