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 #1

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: _____ (18)

Problem 2: _____ (20)

Problem 3: _____ (30)

Problem 4: _____ (32)

Total: _____ (out of 100)

♠

1. ***Hardware Design ISA-bout Tradeoffs (18 points):*** Consider the three properties of execution time. For the following design decisions, indicate how these three properties could impact a MIPS processor running an application ***A*** with the following instruction breakdown and latencies:

Instruction	% of Instructions in ***A***	Instruction Latency (cycles)
Load	20%	5
Store	10%	4
Simple R-type (i.e. adds, ands, shifts)	45%	4
Multiply	10%	5
BEQ/BNE	10%	3
Jump	5%	3

Note that this architecture is not the single cycle datapath and is not pipelined. Your answer for each should be that it either ***could increase, could decrease, or will stay the same.*** You may only circle one answer per property. The first one is done for you as an example.

Example: ***Design Decision***: Use a carry lookahead adder instead of a ripple carry adder.

- *Instruction Count* could increase could decrease (will stay the same)
- *Cycle Time* could increase (could decrease) will stay the same
- *CPI* could increase (could decrease) will stay the same

a. ***Design Decision***: Increase the size of the immediate field for I-type instructions by reducing the number of registers in the ISA.

- *Instruction Count* could increase could decrease will stay the same
- *Cycle Time* could increase could decrease will stay the same
- *CPI* could increase could decrease will stay the same

b. ***Design Decision***: A new physical design technology is used which can reduce wire latency.

- *Instruction Count* could increase could decrease will stay the same
- *Cycle Time* could increase could decrease will stay the same
- *CPI* could increase could decrease will stay the same

c. ***Design Decision***: A new compiler is used to compile application ***A***. The compiler uses shifts in place of multiply instruction – it takes multiple shift instructions to replace a single multiply.

- *Instruction Count* could increase could decrease will stay the same
- *Cycle Time* could increase could decrease will stay the same
- *CPI* could increase could decrease will stay the same

a. ***Design Decision*:** Increase the size of the immediate field for I-type instructions by reducing the number of registers in the ISA.

Instruction Count: Could increase, could decrease

Justification for "could increase":
The decrease in the amount of registers could mean more register spilling. This could increase the amount of lw and sw instructions, increasing the overall instruction count.

Justification for "could decrease":
The increase in the size of the immediate field of I-Type instructions means more flexibility in some instructions:
- addi can now add larger immediates
- beq/bne can now branch farther
- lw/sw can now calculate a farther offset.

It is possible that the original code often needed to explicitly create large immediates (case 1):

```
lui $t0, UPPER
ori $t0, $t0, LOWER
add $t1, $t1, $t0
```

...or needed to branch to distant locations using j (case 2):

```
bne $t0, $t1, LABEL
j ADDR
LABEL:
```

...or both (case 3):

```
lui $t0, UPPER
ori $t0, $t0, LOWER
bne $s0, $s1, LABEL
jr $t0
LABEL:
```

...or in a pathological case, used multiple branches to get to a distant destination (case 4):

```
beq $t0, $t1, LABEL1
...
LABEL1:
beq $t0, $t1, LABEL2
```

Case 1 can be reduced to a single addi and the remaining cases could be reduced to a single beq if the new size of the immediate field is large enough to accommodate the desired immediate. As a result, the IC could be reduced. Note, according to the

instruction type distribution, there are no I-Type instructions other than lw, sw, and beq. Thus, for this particular set of instructions, this justification is most valid when concerning case 2 (reduce the number of jumps) or case 4 (reduce the number of branches).

Cycle Time: Could decrease

Justification for "could decrease":
Due to the reduced number of registers, the hardware complexity is reduced which may decrease the latency of accessing the register file. If the stage in which register access occurs was previously the stage with the greatest latency, the faster register access could result in a reduction of cycle time.

CPI: Could increase

Justification for "could increase":
Due to the decrease in registers, there could be more register spilling, resulting in a higher proportion of loads (CPI = 5) and stores (CPI = 4). The current CPI of the instructions is 4.15. If the result of the increased spilling is the addition of new lw/sw pairs, then the CPI will increase (adding a set of instructions that have an average CPI of 4.5).

Justification for "could increase":
Due to the increased immediate field, there may be a reduction in the number of the following instructions:
- Simple R-Type, CPI = 4 (case 1)
- Non-branch/lw/sw I-Type (addi, ori, lui)
- beq/bne, CPI = 3 (case 4)
- j, CPI = 3 (case 2)

Because the current CPI is 4.15, reducing the number of any one of these would increase the CPI.

Justification for "could decrease":
Consider a case where the original datapath had multiple instruction decode stages. If the reduced number of registers means that multiple decode stages could be reasonably coalesced into one stage due to the decreased complexity, the CPI could decrease due to a reduction of stages in the datapath.

---

b. ***Design Decision:*** A new physical design technology is used which can reduce wire latency.

Instruction Count: Will stay the same

Justification for "will stay the same":
This is purely a change to the hardware. Since the instruction count exists in the software domain, a change to the hardware cannot affect the instruction count unless the

instructions/software explicitly specify that particular hardware component (ex. instructions specify registers).

Cycle Time: Could decrease

Justification for "could decrease":
Because the wire latency has decreased, the latency through each stage of the datapath would likely decrease. As a result, the cycle time could decrease to account for the reduced latency of each stage.

CPI: Could decrease

Justification for "could decrease":
If the reduced wire latency means that multiple stages in the datapath could be coalesced into a single stage as part of the hardware modification, the CPI could be reduced.

---

c. ***Design Decision:*** A new compiler is used to compile application A. The compiler uses shifts in place of multiply instruction – it takes multiple shift instructions to replace a single multiply.

Instruction Count: Could increase

Justification for "could increase":
Since a single multiply is replaced by several shifts, the number of instructions will increase if this multiply replacement is done.

Cycle Time: Could decrease, must stay the same

Justification for "could decrease":
If we make the assumption that not only are we using a different compiler that eliminates multiply instructions, but we are also using different hardware that doesn't include the multiply hardware, the complexity of the hardware is reduced. If the multiply hardware was part of the stage with the maximum latency, then eliminating the multiply hardware could result in a reduced cycle time.

Justification for "must stay the same":
If we make the assumption that ONLY the compiler has changed, then the hardware is unchanged and therefore the cycle time is also unchanged.

CPI: Could decrease

Justification for "could decrease":
Replacing each multiply with multiple shifts has the following effects:
- Reduces the number of multiply instructions which have a CPI of 5
- Increases the number of shift instructions which have a CPI of 4

Since the current CPI is 4.15, both of these changes will bring the overall CPI closer to 4, which will reduce the CPI.

Justification for "could decrease":
As with the possible justification for why CT may decrease, the reduced complexity of not requiring multiply hardware may mean if the EX stages can be reduced or coalesced, the CPI decreases.

2. ***Performance Anxiety (20 points)***: Consider the same application ***A*** and baseline MIPS processor from the previous problem. Application ***A*** executes three billion instructions. Answer the questions below – show your work.

   a. What is the CPI for application ***A***?

$.2\times 5 + .1\times 4 + .45\times 4 + .1\times 5 + .1\times 3 + .05\times 3 = 4.15$

CPI: 4.15

   b. The hardware running the application from part (a) has a cycle time of 300 ps (that is picoseconds). What is the execution time for that hardware to run the application?

$ET = 3\times 10^{9} \times 4.15 \times 300\times 10^{-12}$

ET: 3.735 s

   c. Now suppose we add an instruction that does a multiply/accumulate operation. The multiply/accumulate operation replaces one multiply instruction and one add instruction. This optimization allows the compiler to replace 50% of multiplies. What is the new CPI?

Multiply 10 → 5
R-type 45 → 40
MAC 5

CPI: 4.27

$\frac{90}{95}\times 5 + \frac{10}{95}\times 4 + \frac{40}{95}\times 4 + \frac{5}{95}\times 5 + \frac{10}{95}\times 3 + \frac{5}{95}\times 3 + \frac{5}{95}\times$

$1.32 + 2.11 + 0.47 + 0.37 = 4.27$

   d. What is the execution time for the same hardware from part (b) to run the new application from part (c)?

$ET = (.95)(3)(10^{9}) \times 4.27 \times 300\times 10^{-12}$

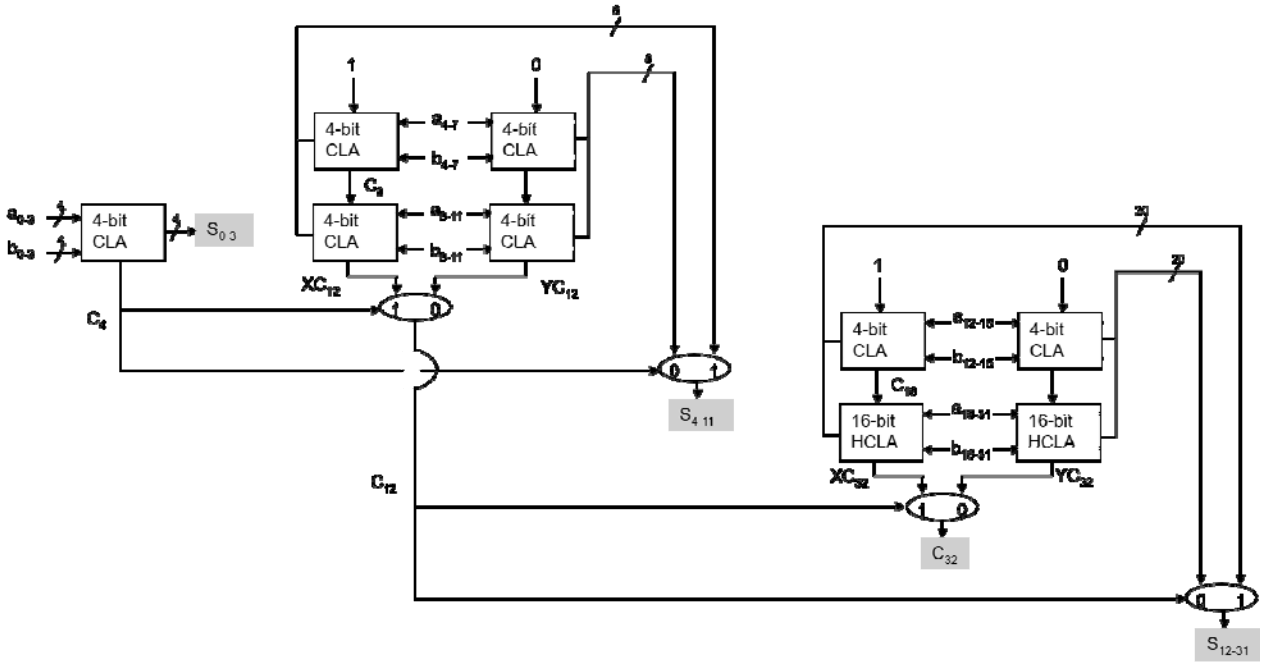
ET: 3.65 s

3. ***Putting the CLA into UCLA (30 points)***: Assume for the rest of this problem that all logic gates have the following delays:

Fan In	Delay
1	5T
2	6T
3	7T
4	8T
5	9T
6 or more	2T x fan-in

So a 2-input AND gate would have delay 6T and a 4-input OR gate would have delay 8T. For simplicity, assume that mux's have delay 6T regardless of fan-in.

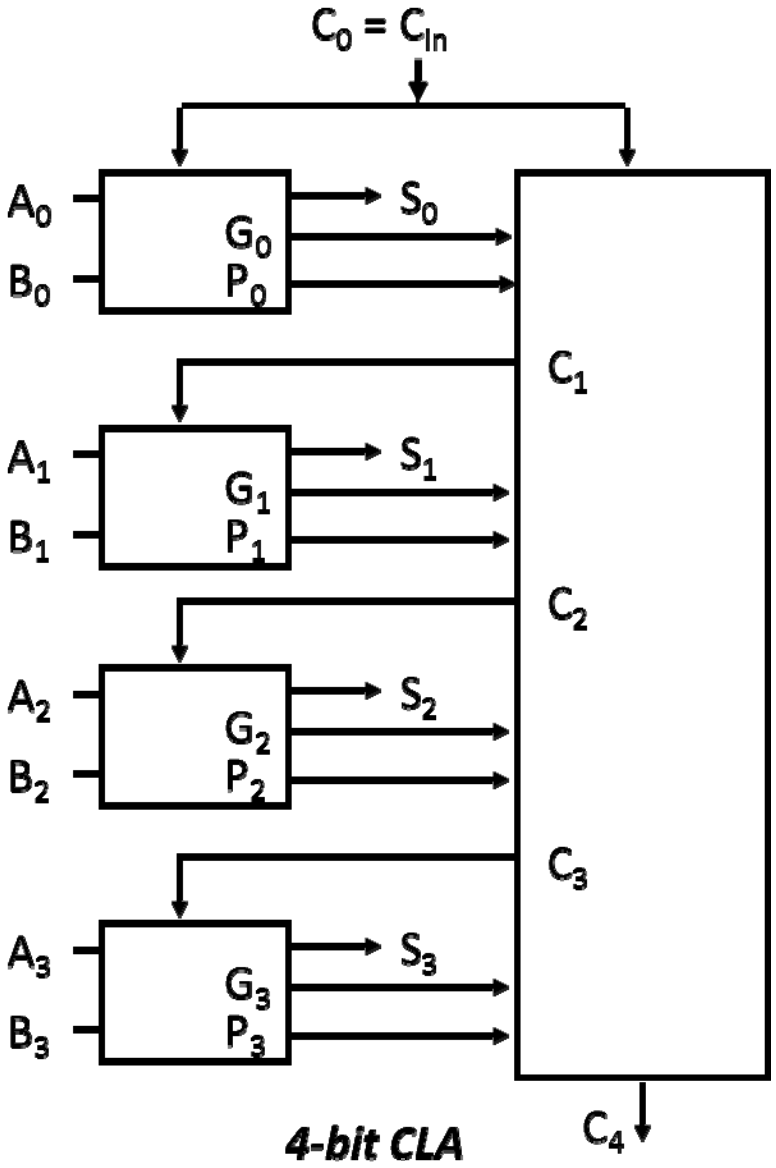
We will create a 32-bit adder out of some building blocks we've covered in class. We will use the 4-bit carry lookahead (4-bit CLA) that we covered in class as one basic building block of this design. And we will use it (as we did in class) to make a 16-bit hierarchical CLA (16-bit HCLA). But instead of connecting these in series to make a 32-bit adder, we will use carry select to speed up the 32-bit adder. The design will look as follows (be sure to note where we are using the 4-bit CLA and where we are using the 16-bit HCLA):



The grey boxes show the outputs of the 32-bit adder, including the Carry Out (C_{32}) and Sum bits S_0 through S_{32}. The individual CLA and HCLA blocks take input from a carry in and the A and B operands. For example, the left most 4-bit CLA is computing the sum of inputs A_0-A_3 and B_0-B_3 (i.e. producing sums S_0-S_3). The carry out of this 4-bit CLA (marked C_4) selects the sums and carry out of the two 4-bit CLAs in Ripple Carry. The carry out of the these Ripple Carry 4-bit CLAs (marked C_{12}) selects the sums and carry out of the 4-bit CLA and 16-bit HCLA in Ripple Carry. The 0 or 1 at the top the adders in carry select are the hardwired C_{in} for these blocks. Multiplexers are shown as ovals.

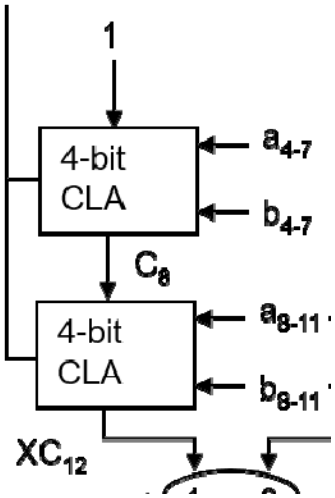
Your task is to find the maximal delay of this design – i.e. determine the delays of S_{0-31} and C_{32} – the maximal delay of these outputs will be the maximal delay of the entire design. To do this (and to help with possible partial credit) please use the diagrams on the following pages and fill in the tables in every page.

Single 4-bit CLA:



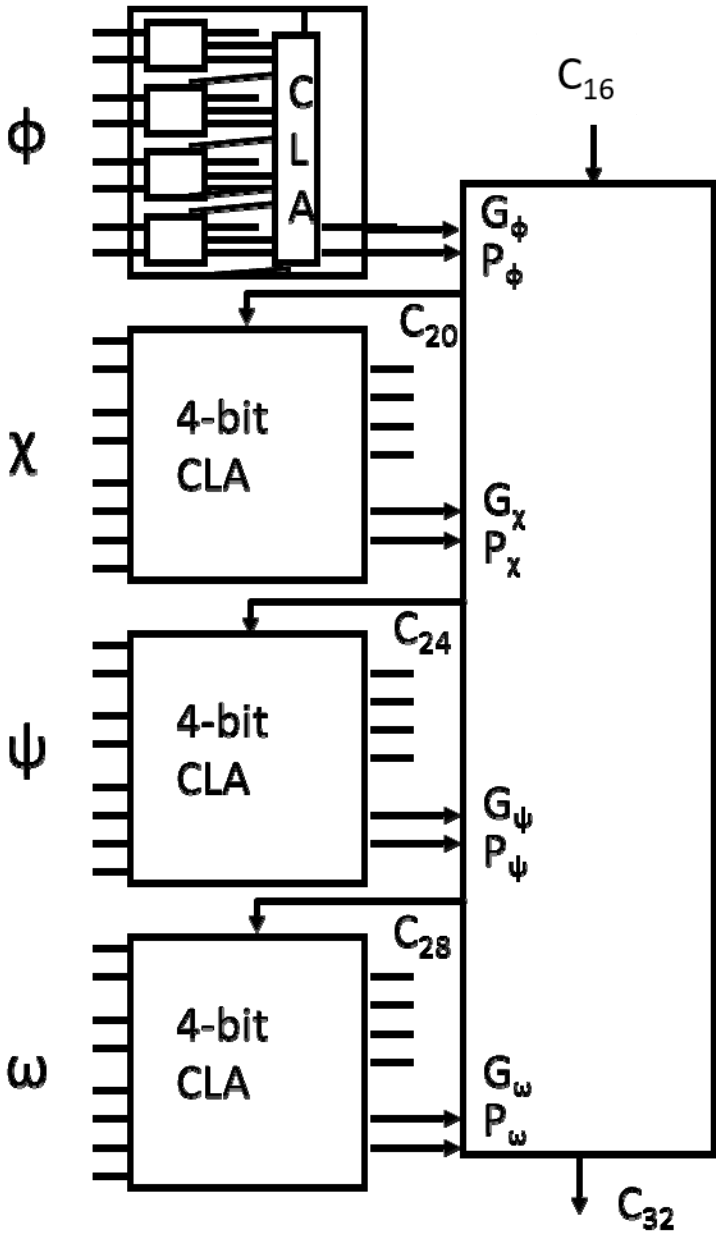
Output	Delay (in terms of T)	
G0		*(1 points)*
P0		*(1 points)*
G3		*(1 points)*
P3		*(1 points)*
C3		*(2 points)*
C4		*(2 points)*
S3		*(1 points)*

Ripple Carry 4-bit CLAs:



Output	Delay (in terms of T)	
C8		*(1 points)*
XC12 (before any muxing)		*(2 points)*
S11 (before any muxing)		*(2 points)*

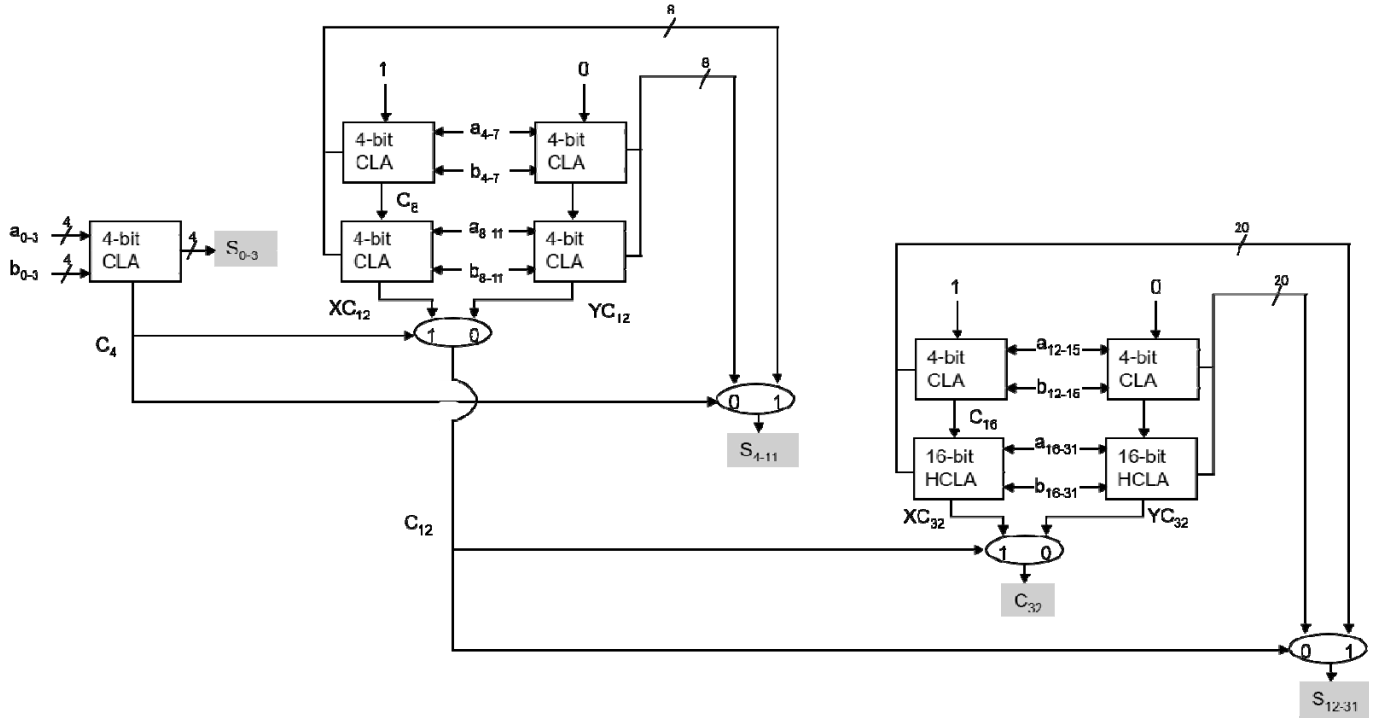
16-bit HCLA:



16-bit HCLA

Output	Delay (in terms of T)	
Gω		*(2 points)*
Pω		*(2 points)*
C16		*(2 points)*
C28		*(2 points)*
C32 (before any muxing)		*(2 points)*
S31 (before any muxing)		*(2 points)*

Entire Design:



Output	Delay (in terms of T)	
C12 (after the mux)		*(1 points)*
C32 (after the mux)		*(1 points)*
S31 (after the mux)		*(1 points)*

Find the maximum delay **in terms of T** of the 32-bit adder – take the maximum of all output bits – including the sum bits (S_0-S_{31}) and the final carry out (C_{32}).

Maximal Delay: _____ *(1 points)*

*: AND
+: OR
^: XOR
Using Sn = (An^Bn^Cn) (3-input XOR)

1. G0 = A0*B0
- Delay(G0) = 6T

2. P0 = A0^B0
- Delay(P0) = 6T

3. G3 = A3*B3
- Delay(G3) = 6T

4. P3 = A3^B3
- Delay(P3) = 6T

5. C3 = G2 + G1*P2 + G0*P1*P2 + C0*P0*P1*P2
- Delay(C3) = 6T (P0/P1/P2) + 8T (4-input AND) + 8T (4-input OR) = 22T

6. C4 = G3 + G2*P3 + G1*P2*P3 + G0*P1*P2*P3 + C0*P0*P1*P2*P3
- Delay(C4) = 6T (P0/P1/P2/P3) + 9T (5-input AND) + 9T (5-input OR) = 24T

7. S3 = (A3^B3^C3)
- Delay(S3) = 22T (C3) + 7T (3-input XOR) = 29T

8. C8= G7+G6*P7+G5*P6*P7+G4*P5*P6*P7+C4*P4*P5*P6*P7
-Delay(C8) = 6T (P4/P5/P6/P7) + 9T (5-input AND) + 9T (5-input OR) = 24T

9. C12= G11+G10*P11+G9*P10*P11+G8*P9*P10*P11+C8*P8*P9*P10*P11
-Delay(C12) = 24T (C8) + 9T (5-input AND) + 9T (5-input OR) = 42T

10. S11= A11^B11^C11
C11= G10+ G9*P10+ G8*P9*P10+ C8*P8*P9*P10
-Delay(C11) = 24T (C8) + 8T (4-input AND) + 8T (4-input OR) = 40T
-Delay (S11) = 40T (C11) + 7T (3-input XOR)= 47T

11. Gω= G31 + G30*P31 + G29*P30*P31 + G28*P29*P30*P31
-Delay (Gω) = 6T (G28) + 8T(4-input AND) + 8T(4-input OR) = 22T

12. Pω= P31*P30*P29*P28
-Delay (Pω) = 6T (P28/P29/P30/P31) + 8T (4-input AND) = 14T

13. C16= G15 + G14*P15 + G13*P14*P15 + G12*P13*P14*P15 + C12*P12*P13*P14*P15
-Delay (C16) = 6T (P15/P14/P13/P12) + 9T (5-input AND) + 9T (5-input OR) = 24T

14. C28= Gψ + Gχ*Pψ + GΦ*Pχ*Pψ + C16*PΦ*Pχ*Pψ
-Delay (C28) = 24 T(C16) + 8T (4-input AND) + 8T(4-input OR) = 40T

15. C32 = Gω + Gψ*Pω + Gχ*Pψ*Pω + GΦ*Pχ*Pψ*Pω + C16*PΦ*Pχ*Pψ*Pω
-Delay (C32) = 24T (C16) + 9T (5-input AND) + 9T(5-input OR)= 42T

16. S31 = A31^B31^C31
C31= G30 + G29*P30 + G28*P29*P30 + C28*P28*P29*P30
-Delay(C31) = 40T (C28) + 8T (4-input AND) + 8T (4-input OR) = 56T
-Delay(S31) = 56T(C31) + 7T(3-input XOR)= 63T

17. C12 (after MUX) = 42T(C12 before MUX) + 6T (MUX)= 48T

18. C32 (after MUX) = 48T (C12 after MUX) + 6T (MUX)= 54T

19. S31 (after MUX) = 63T (S31 before MUX) + 6T (MUX)= 69T

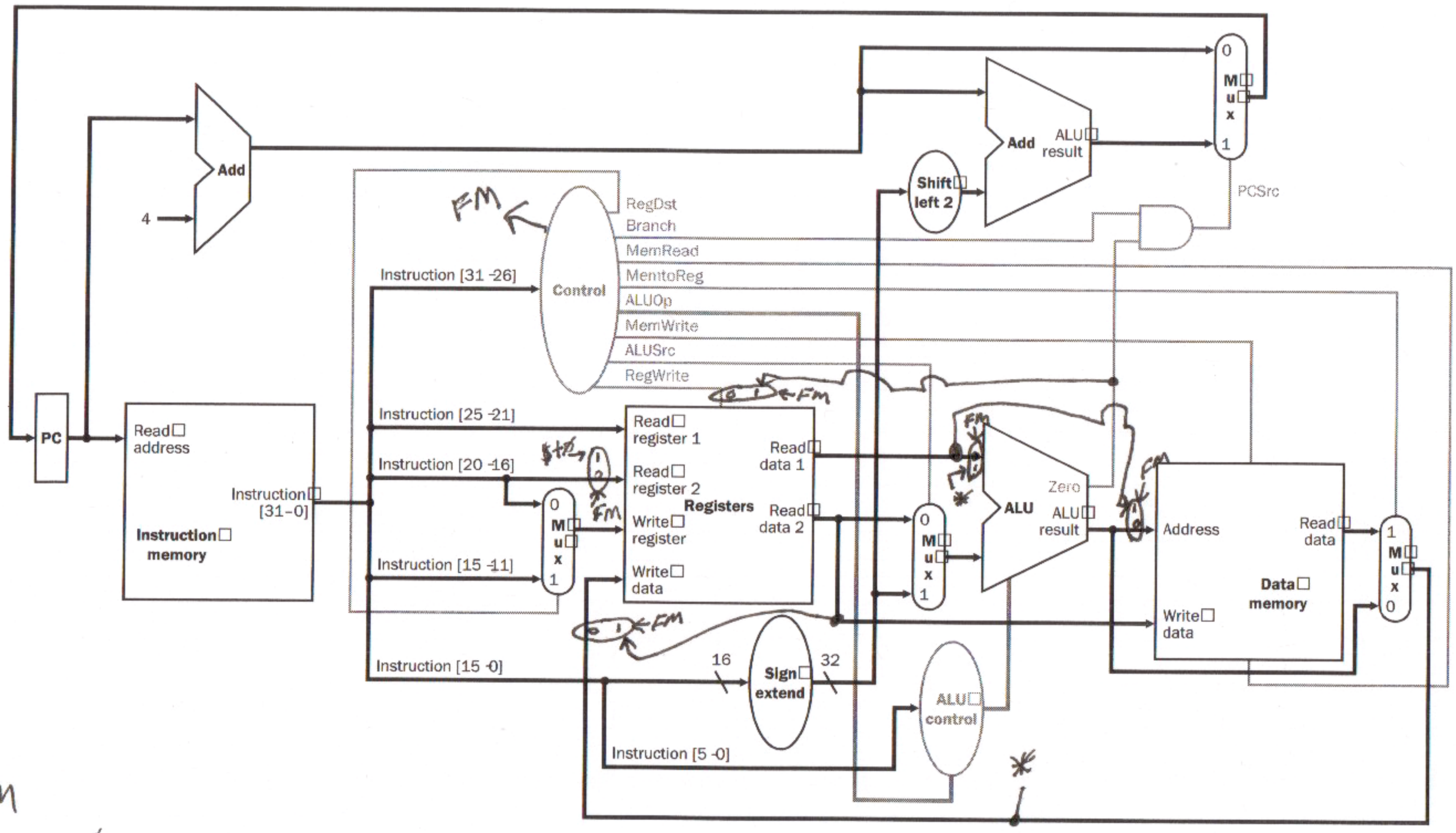
Maximal delay = 69T

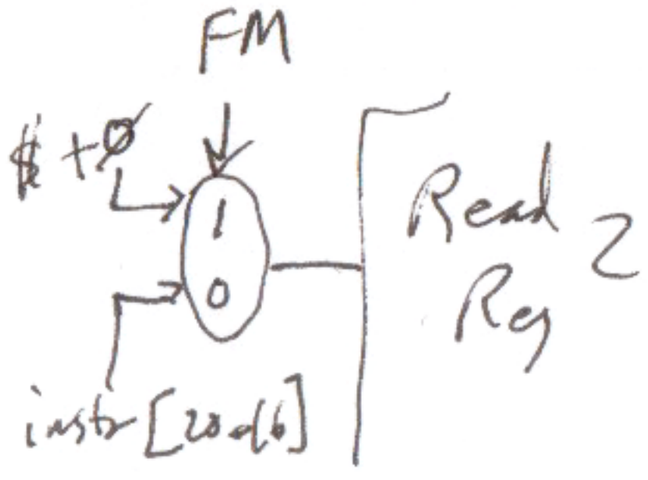
4. ***FMOV (32 points)***: Consider the single-cycle processor implementation. Your task will be to augment this datapath with a new instruction: the *fmov* instruction. This instruction will be an I-type instruction, and will have the following effect:

```
if (M[R[rs]]==SE(I))
        R[rt]=R[$t0]
```

Note that the *fmov* always uses register $t0 as the source value that is put into R[rt] – it is an implicit operand.

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.


Add
4
PC
Read address
Instruction [31-0]
Instruction memory
Instruction [25-21]
Instruction [20-16]
Instruction [15-11]
Instruction [15-0]
Instruction [5-0]
RegDst
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Control
FM
Shift left 2
Add
ALU result
0
M u x
1
PCSrc
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
$t0
16
Sign extend
32
ALU control
Zero
ALU
ALU result
Address
Read data
Data memory
Write data


FM
$t0
1
0
Read Reg 2
instr[20-16]

Main Controller

Input or Output	Signal Name	R-format	lw	sw	Beq	Fmov
Inputs	Op5	0	1	1	0	U
	Op4	0	0	0	0	~
	Op3	0	0	1	0	I
	Op2	0	0	0	1	Q
	Op1	0	1	1	0	Y
	Op0	0	1	1	0	Z
Outputs	RegDst	1	0	X	X	0
	ALUSrc	0	1	1	0	1
	MemtoReg	0	1	X	X	X
	RegWrite	1	1	0	0	X
	MemRead	0	1	0	0	1
	MemWrite	0	0	1	0	0
	Branch	0	0	0	1	0
	ALUOp1	1	0	0	0	0
	ALUOp0	0	0	0	1	1
	Fm1	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