

Problem 1 (10 points)

Given the following object code files, perform the linking process and fill in the executable file. You may assume that foo() is laid out in memory earlier than bar().

OBJECT FILE HEADER 1:

	Name	foo()	
	Text Size	0x150	
	Data Size	0x40	
Text Segment	Address	Instruction	
	0	lw \$a0 0(\$gp)	
	4	sw \$t0 0(\$gp)	
	8	sw \$t0 0(\$gp)	
	12	jal 0	
	...		
Data Segment	0	(A)	
	4	(B)	
	
Relocation Info	Address	Instruction Type	Dependency
	0	lw	A
	4	sw	B
	8	sw	A
	12	jal	bar
Symbol Table	Label	Address	
	A	?	
	B	?	
	bar	?	

OBJECT FILE HEADER 2:

	Name	bar()	
	Text Size	0x240	
	Data Size	0x9A	
Text Segment	Address	Instruction	
	0	sw \$t0 0(\$gp)	
	4	jal 0	
	...		
Data Segment	0	(X)	
	
Relocation Info	Address	Instruction Type	Dependency
	0	sw	X
	4	jal	foo
Symbol Table	Label	Address	
	X	?	
	foo	?	

FILL IN THIS PORTION: EXECUTABLE FILE HEADER

	Text Size	0x390
	Data Size	0xda
Text Segment	Address	Instruction
	0x00400000	lw \$a0 0x8000(\$gp)
	0x00400004	sw \$a0 0x8004(\$gp)
	0x00400008	sw \$a0 0x8000(\$gp)
	0x0040000c	jal 0x00400150
	...	

	0x00400150	sw \$t0 0x8040(\$gp)
	0x00400154	jal 0x00400000
	...	
Data Segment	Address	
	0x10000000	(A)
	0x10000004	(B)
	...	
	0x10000040	(X)
	...	

In this problem, the beginning of the text segment and beginning of the data segment are 0x00400000 and 0x10000000 respectively, as defined by the MIPS memory layout (see pg. 87 of book)

Problem 2 (5 points)

Although the IEEE 754 floating point format allows for many numbers with fractional parts to be represented using 32 bits, not all values can be represented exactly. Consider the number 0.3.

$$\frac{1}{3} = 0.\overline{3}$$

- What is the closest value to this number that we can get using the IEEE 754 floating point format? (2 points)
- Give your answer in decimal and floating point representation. (2 points)
- What is the percent error in using floating point for this case? (1 point)

Floating point = 0 01111101 010101010101010101010

Decimal = 0.333333313465118

% error = 0.0000059605%

Problem 3 – ALU Design (5 points)

In this problem, you will design a 2-bit ALU for the LC2K9 processor. The ALU has two data inputs named A[1:0] and B[1:0] and one output called Res[1:0]. (The notation W[X:0] indicates a group of wires name W numbered from X to 0) For instance, a 32-bit instruction is written as Inst[31:0]. For this problem, assume that A[1], B[1], Res[1] are the most significant bits). The ALU performs the following four operations - ADD, AND, NAND, XOR. (You can ignore the

carry-out bit CarryOut[1] and assume that the carry-in signal for bit0=CarryIn[0] is zero). The ALU also has a 2-bit controlling signal called ALU_Sel[1:0] that selects which of the four operations will be written to Res[1:0].

- So, ALU_Sel = 00 ==> Res = ADD(A,B),
- ALU_Sel = 01 ==> Res = AND(A,B),
- ALU_Sel = 10 ==> Res = NAND(A,B),
- ALU_Sel = 11 ==> Res = XOR(A,B).

You are to draw the complete ALU block diagram and label all the signals in the figure. You can use the following building blocks:

- NOT gates
- 2-input AND gates
- 2-input NAND gates
- 2-input OR gates
- 2-input XOR gates
- 4-to-2 Multiplexors (MUXes)

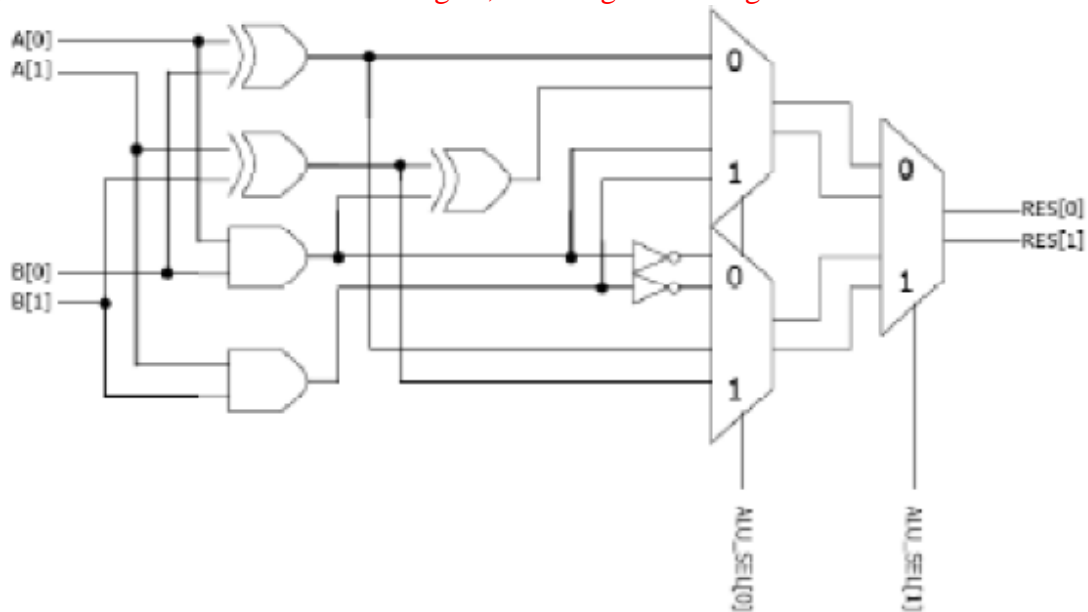
Note that you cannot use half-adders or full-adders directly in this design. Instead you need to construct the circuit for the ADD operation using only the given gates/blocks.

Draw your design, and indicate the number of gates required to implement your design. (Count your MUX as 1 gate).

SOLUTION:

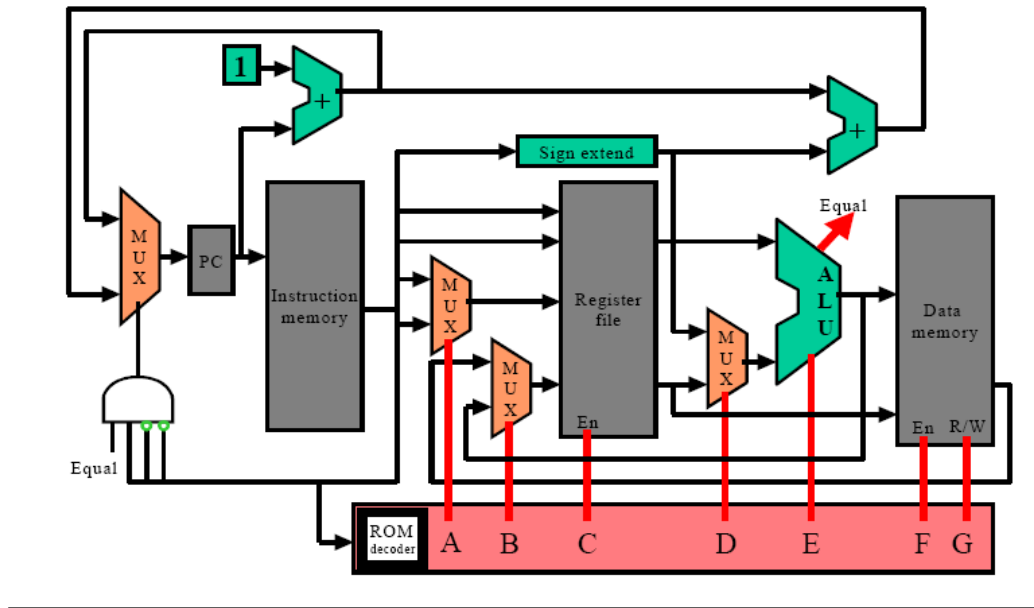
- SUM[0] = A[0] xor B[0]
- COUT[0] = A[0] & B[0]
- SUM[1] = A[1] xor B[1] xor COUT[0]
- Can ignore CIN[0] and COUT[1]

Assuming that the 4-to-2 MUXes count as a gate, the design uses 10 gates



Problem 4 (10 points)

This problem refers to the following single-cycle datapath implementation of LC-2K9 that was covered in lecture.



a) Assume the following delays for the datapath components

Read from memory: 10ns

Write to memory: 5ns

Read from register file: 2ns

Write to register file: 1ns

ALU: 5ns

All other components have negligible delay.

If you had the following LC2K Snippet:

lw 0 1 15

sw 0 1 16

noop

beq 0 1 10

How long would it take to run each instruction? Assuming that these (lw, sw, noop, beq) were the ONLY instructions in your instruction set, what would be your minimum clock cycle? Please show ALL work.

b. Which of the following are correct statements about the benefits of using a multicycle datapath rather than a single cycle? There may be more than one answer.

- Decreases number of cycles per instruction
- Decreases clock period
- Decreases execution time of every instruction
- Decreases overall execution time
- Increases efficiency of datapath components

ANSWERS:

lw

read from instr. Memory 10

read from register 2

alu operation 5

read from data memory 10

write to register 1

=28ns

Sw:

Read from instr memory: 10

Read from register: 2

Alu operation: 5

Write to memory: 5

=22ns

Noop

Read from instr memory: 10

=10ns

Beq

Read from instr memory: 10ns

Read from register: 2ns

Alu operation: 4ns

=16ns

How long would it take to run each instruction? Assuming that these (lw, sw, noop, beq) were the ONLY instructions in your instruction set, what would be your minimum clock cycle? (28ns due to lw) Please show ALL work.

b.

- I. Decreases number of cycles per instruction (NO)
- II. Decreases clock period (YES)
- III. Decreases execution time of every instruction (Not Necessarily)
- IV. Decreases overall execution time (Not Necessarily)
- V. Increases efficiency of datapath components (YES)

Problem 5 (8 points)

Consider the fictional gate MAJORITY, defined as a gate with 3 inputs and 1 output. If 0 or 1 of the inputs are 1's and the rest are 0's, the output is 0. Otherwise, (that is, if 2 or 3 of the inputs are 1's), the output is 1. In other words, if a majority of the inputs are 1's, the output is a 1.

Given the following truth table with the inputs ABCD and the outputs XY, draw a circuit that implements it using only MAJORITY gates. Try to do it with as few gates as possible. Hint: you can do it with only two gates.

A	B	C	D	X	Y
0	0	0	0	0	0
0	0	0	1	0	0
0	0	1	0	0	1
0	0	1	1	0	1
0	1	0	0	0	0
0	1	0	1	0	0
0	1	1	0	0	1
0	1	1	1	1	1
1	0	0	0	0	0
1	0	0	1	1	0
1	0	1	0	1	1
1	0	1	1	1	1
1	1	0	0	1	0
1	1	0	1	1	0
1	1	1	0	1	1
1	1	1	1	1	1

SOLUTION:

