Midterm Results for ZHANG, CHARLES XIAN

Score for this quiz: **91.5** out of 100
Submitted Feb 2 at 11:50am
This attempt took 107 minutes.

Question 1

6 / 6 pts

For the following design decision, indicate the impact on the three components of execution time:

We manufacture our existing single cycle MIPS architecture datapath using new transistors that have lower latency. We run the same applications as before, without the need to recompile anything.

a. Instruction Count: must stay the same

b. CPI: must stay the same

c. Cycle Time: could decrease

Answer 1:

Correct!

must stay the same

Answer 2:

Correct!

must stay the same

Answer 3:

Correct!

could decrease

Question 2

4 / 6 pts

For the following design decision, indicate the impact on the three components of execution time:

We increased the size of the register file in the MIPS architecture for our single cycle datapath, which reduced register spilling. The increase in the number of bits in the register specifier field meant that we had to reduce other instruction fields, but assume there was no performance impact from that reduction in other instruction fields.

a. Instruction Count: could decrease

b. CPI: must stay the same

c. Cycle Time: must stay the same

Answer 1:

Correct!

could decrease

Answer 2:

Correct!

must stay the same

Answer 3:

orrect Answer

could increase

ou Answered

must stay the same

Question 3

4 / 6 pts

For the following design decision, indicate the impact on the three components of execution time:

We have a MIPS processor design (not a single cycle design) with a compiler. We make one change – we install a new compiler that employs a new strength reduction optimization: the compiler avoids using a single complex, multi-cycle instruction and instead uses multiple simpler instructions that each execute in fewer cycles.

a. Instruction Count: could increase

b. CPI: could decrease

c. Cycle Time: could decrease

Answer 1:

Correct! could increase

Answer 2:

Correct! could decrease

Answer 3:

ou Answered could decrease

orrect Answer must stay the same

Question 4

10 / 10 pts

We are designing a 4 GHz processor that is not the single-cycle datapath we covered in class. Instructions on this datapath take multiple cycles – but there is no pipelining – only one instruction may execute at a time. R-type instructions take 11 cycles. LW instructions take 13 cycles. SW instructions take 10 cycles. BEQ/BNE instructions take 9 cycles. For a particular application, 30% of instructions are LW, 20% are BEQ/BNE, 10% are SW, and the rest are R-type. What is the CPI of the application on this processor? Show your work.

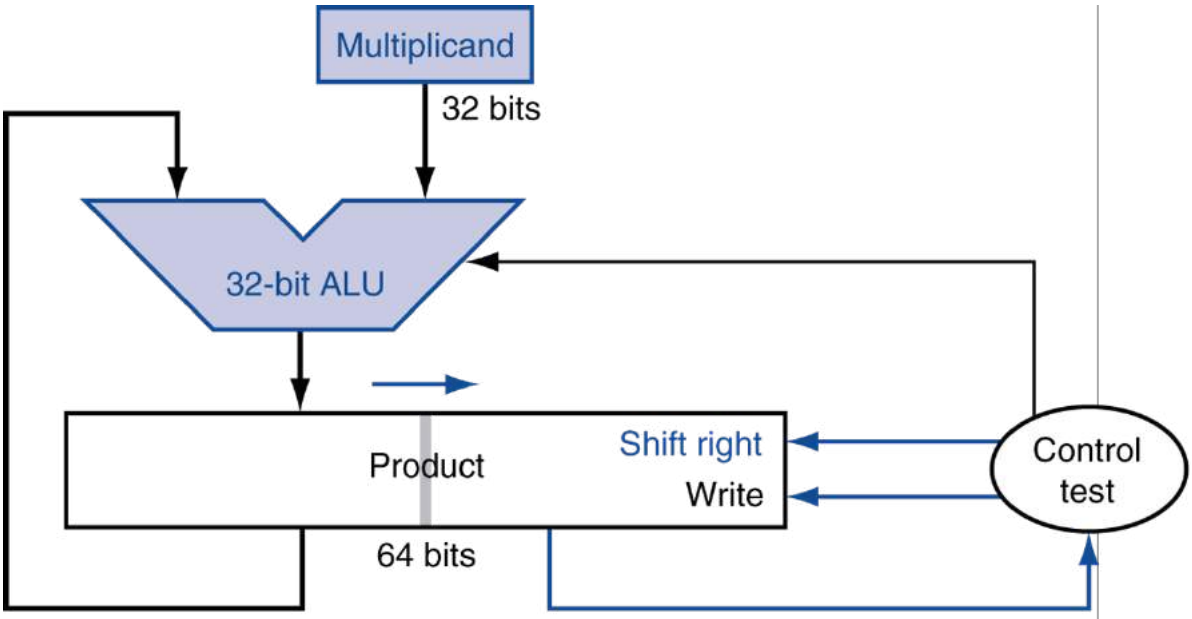
Correct! 11.1

orrect Answers 11.1 (with margin: 0)

Question 5

10 / 10 pts

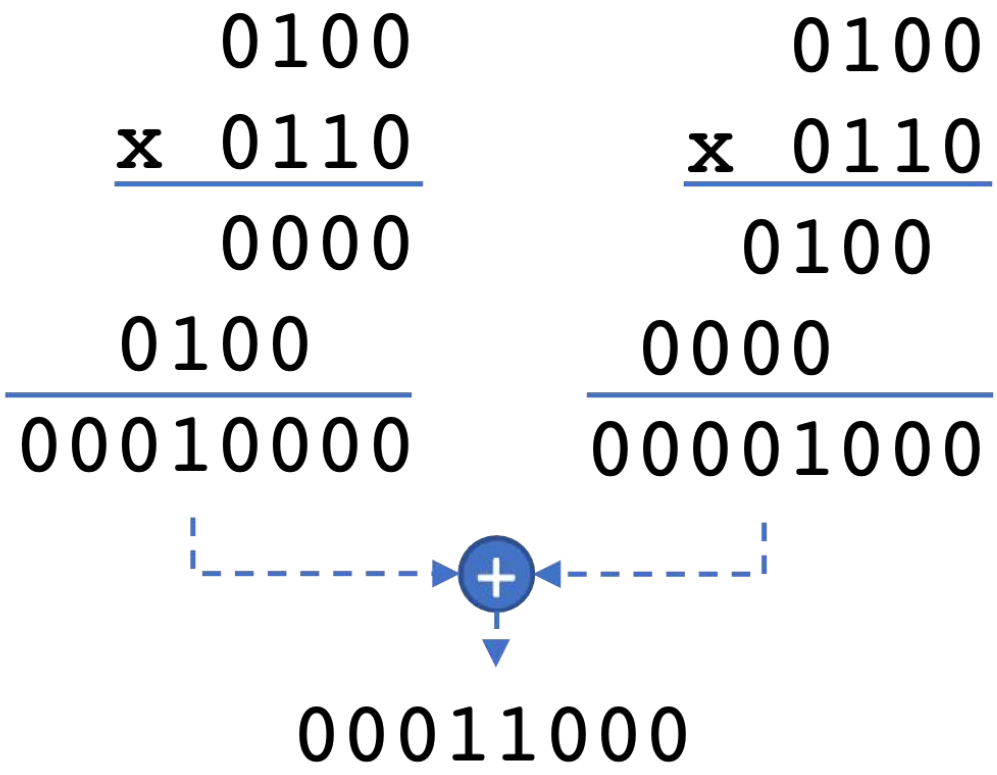
We covered a multiplication unit in class that looked like this:



Let's try and make a version that has more parallelism. Instead of one unit like the above, let's use two of them to make things faster. We'll leverage the associative property of addition to make this work. Consider the example 4-bit multiplication below of 4 x 6 to form an 8-bit product. This is the usual way we would work it out:

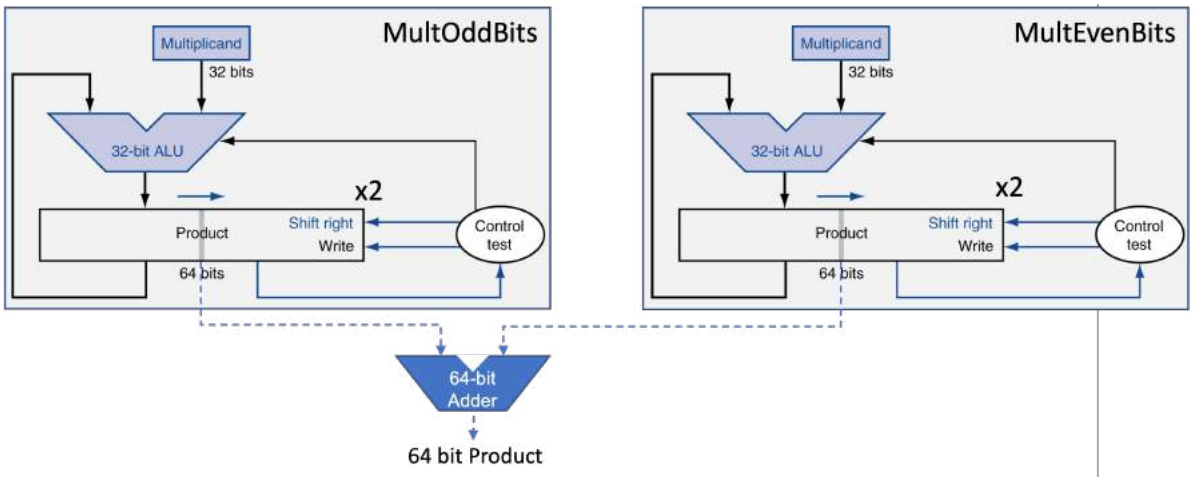
```
   0100
x  0110
------
   0000
  0100
 0100
0000
--------
00011000
```

It is the sum of four 4-bit integers – and these integers are formed by looking at each of the four bits of the multiplier (0110 in this case). But what if we did two parallel multiplications where we only looked at half of the four bits of the multiplier. Like this:



In this case, we handle the odd bits of the multiplier at the left (a sum of two 4-bit integers) and the even bits of the multiplier at the right (a sum of two 4-bit integers) – and then we add those two 8-bit partial sums together to form the final 8-bit integer.

Now let's do the same thing with the multiplication unit – we'll have two multiplication units, one to handle the odd bits of the multiplier (MultOddBits) and one to handle the even bits of the multiplier (MultEvenBits). And we will change our control unit to shift by two places instead of one (designated by the x2). We'll put the multiplier in the right half of the product register in the MultOddBits multiplication unit, and we'll put the multiplier shifted right by one into the right half of the product register in the MultEvenBits multiplication unit. Then, instead of running 32 iterations in the multiplication unit, since we are only doing half the bits (shifting by 2 and only doing either odd or even bits), we will only run for 16 iterations. Here's the design:



Given this design, answer the following questions.

a) if it takes 7 nanoseconds for each iteration of the multiplication unit, how long would it take the original multiplication unit to multiply two 32 bit integers? Enter the time in nanoseconds (just the number, no label):

224

b) how long would it take our modified design (with MultOddBits and MultEvenBits) to multiply two 32 bit integers? Assume that shifting by 2 takes the same amount of time as shifting by 1. Further assume that a 64-bit adder takes 7 nanoseconds to compute a sum. Enter the time in nanoseconds (just the number, no label): 119

Answer 1:

Correct!

224

Answer 2:

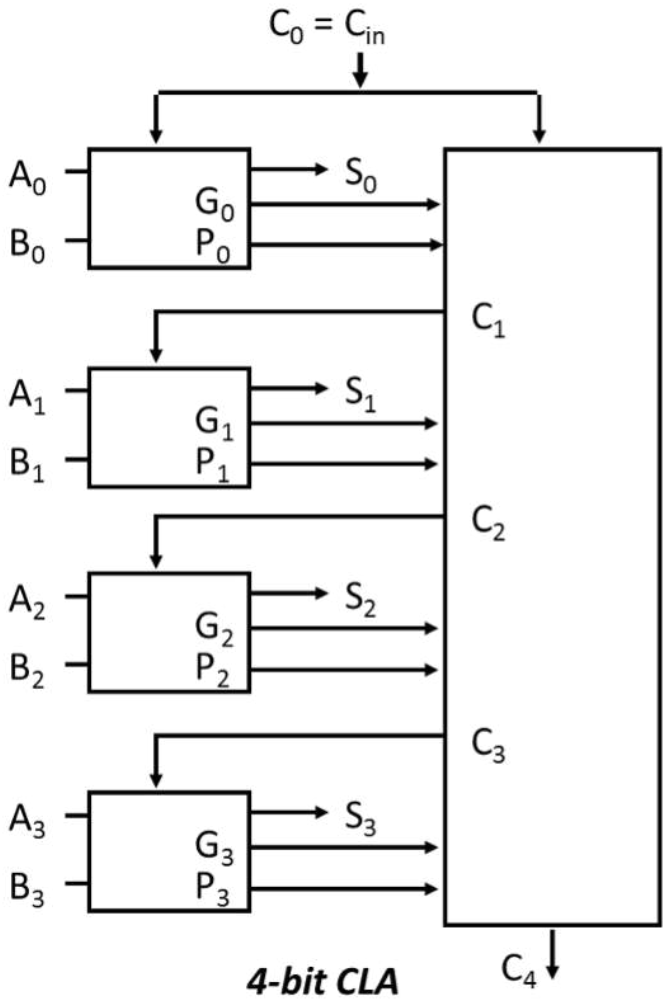
Correct!

119

Question 6

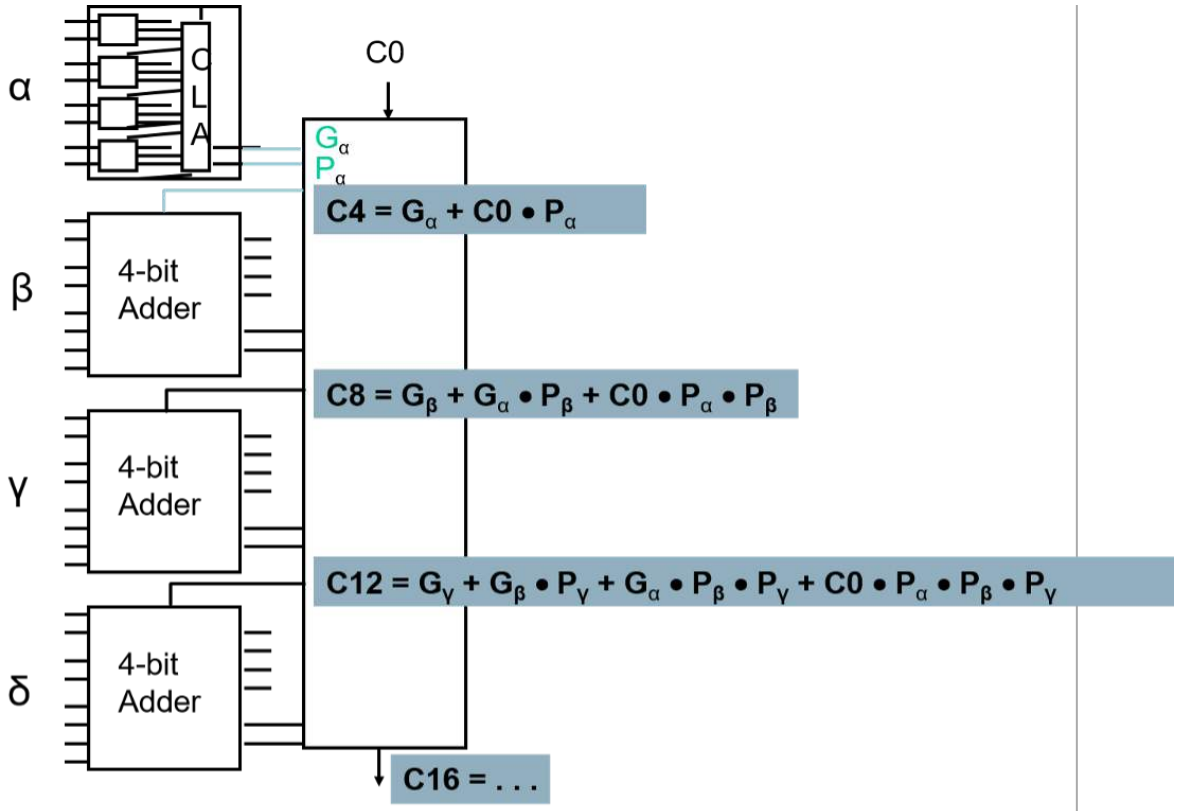
16 / 16 pts

In class, we discussed the design of a 4-bit CLA, like this:



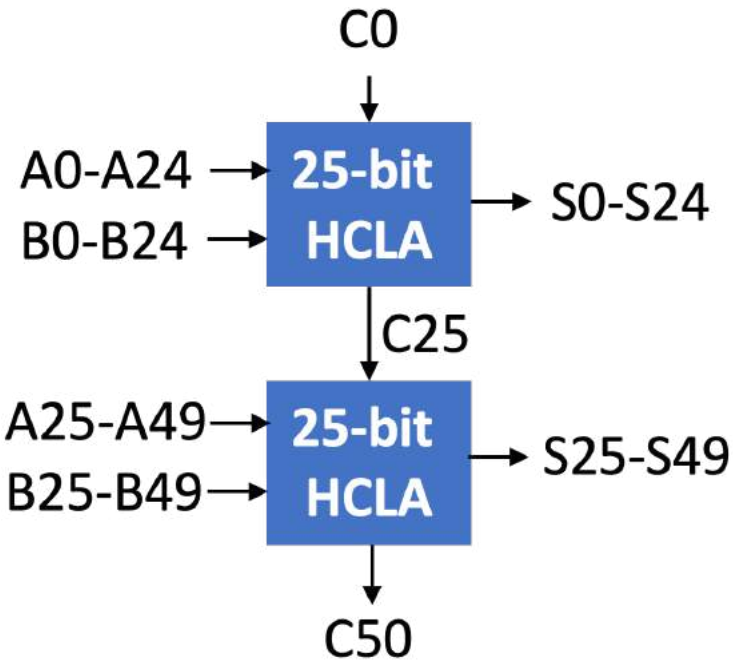
As part of this question, we are going to extend this design and make a ***5-bit CLA***.

In class, we then talked about making a 16-bit HCLA that was comprised of four 4-bit CLAs (alpha, beta, gamma, delta):



As part of this question, we are going to extend this design and make a ***25-bit HCLA*** comprised of 5-bit CLAs.

In particular, we want to make a specialized 50-bit adder for an application that requires 50 bits of precision. We are going to change the designs above to use 5-bit CLAs (comprised of five single bit adders) and 25-bit HCLAs (comprised of five 5-bit CLAs with labels alpha, beta, gamma, delta, and epsilon). The two 25-bit HCLAs are going to be connected together in ripple carry fashion (e.g. C25 from the first HCLA will directly connect to the Cin of the second HCLA):



Assume that the delay of any gate (e.g. OR, XOR, AND) is (k+7)T, where k is the fan-in of the gate, and T is some technology dependent variable. For example, a 2-input AND gate would have delay 9T. All of your answers should be in terms of T.

Determine the maximum delay of the adder: 115T

Show your work by filling in the intermediate delays below:

G0 = 9T

P0 = 9T

G25 = 9T

P25 = 9T

Galpha = 33T

Palpha = 21T

C25 = 58T

C50 = 84T

C45 = 82T

C49 = 106T

S49 = 115T

Answer 1:

ou Answered 115T

orrect Answer 113T

Answer 2:

Correct! 9T

Answer 3:

Correct! 9T

Answer 4:

Correct! 9T

Answer 5:

Correct! 9T

Answer 6:

ou Answered 33T

orrect Answer 31T

Answer 7:

ou Answered 21T

orrect Answer 20T

Answer 8:

ou Answered 58T

orrect Answer 56T

Answer 9:

ou Answered 84T

orrect Answer 82T

Answer 10:

ou Answered 82T

orrect Answer 80T

Answer 11:

ou Answered 106T

orrect Answer 104T

Answer 12:

ou Answered 115T

orrect Answer 113T

Question 7

17 / 18 pts

Extend the single cycle datapath and control we covered in class with the ***WTF*** instruction:

WTF

- R-Type Instruction (opcode is the same as all other R-Type Instructions)

- Function field is unique.

- Pseudocode:

If (R[RT]==SE(SA))

R[RD]=PC+4+M[R[RS]]

This is an instruction that checks if the value in register RT is equal to the sign extended value in the shift amount field of the R-type format. Note that this instruction uses the shift amount field in a way not originally intended - and you may assume that sign extension hardware will extend a 5-bit value into a 32-bit value. If the value in RT is equal to the sign extended shift amount field of the instruction, then we set register RD to the sum of the PC+4 and the contents of memory at the address contained in register RS.

Implement your solution on the datapath and control you printed out for the exam (alternatively you may use an electronic copy). All other instructions supported in the datapath/control must still work correctly after your modifications. You should not add any additional ALUs, register file ports, or ports to memory

CamScanner 02-02-2022 11.38.pdf
(https://bruinlearn.ucla.edu/files/8091275/download)

Question 8

14 / 14 pts

Consider the single cycle datapath we covered in class. The processor has a 3.5 GHz clock. We are running an application with 300 billion

instructions. The application has the following instruction mix.

Instruction	% of Instructions
LW	40%
SW	10%
Other arithmetic R-Type or I-Type (e.g. ADD, ADDI, AND, OR, SUB)	30%
BEQ/BNE	15%
J	5%

What is the Execution Time for this application running on this processor? Indicate the requested intermediate results below. Round to the nearest hundredths place.

CPI = 1

CT = 0.29 (in nanoseconds - 10^{-9} seconds)

IC = 300 (in billions of instructions)

ET = 87 (in seconds)

Now we are going to modify this architecture and the application. 25% of all LW instructions are immediately followed by an ADD instruction that uses the result of the LW. For example:

```
LW $t0, 4($s0)
ADD $s1, $s1, $t0
```

In these cases, the result written by the LW (e.g. register $t0) is only used once, by the following ADD instruction, and does not need to be stored for future instructions (e.g. the value in register $t0 is live up to the point of the ADD instruction, and the value is dead after that). So in all of these cases, we are going to replace these two instructions with a single **new** instruction we are adding to the ISA – the LWA instruction. The LWA instruction is an I type instruction with the following pseudocode:

```
R[ RT ] += M[R[ RS ] + SE(I)]
```

So the example code:

```
LW $t0, 4($s0)
ADD $s1, $s1, $t0
```

can be replaced with a single LWA instruction:

```
LWA $s1, 4($s0)
```

Now let's assume that the implementation you made adds 300 picoseconds of latency to the critical path of the single cycle datapath. Find the new Execution Time for this modified architecture and application. Indicate the requested intermediate results below. Round to the nearest hundredths place.

CPI = 1

CT = 0.59 (in nanoseconds - 10^{-9} seconds)

IC = 270 (in billions of instructions)

ET = 159.3 (in seconds)

Answer 1:

Correct! 1

orrect Answer 1.0

orrect Answer 1.00

Answer 2:

Correct! 0.29

orrect Answer .29

Answer 3:

Correct! 300

orrect Answer 300.00

Answer 4:

ou Answered 87

orrect Answer 85.71

Answer 5:

Correct! 1

orrect Answer 1.0

orrect Answer 1.00

Answer 6:

Correct! 0.59

Answer 7:

Correct! 270

orrect Answer 270.00

Answer 8:

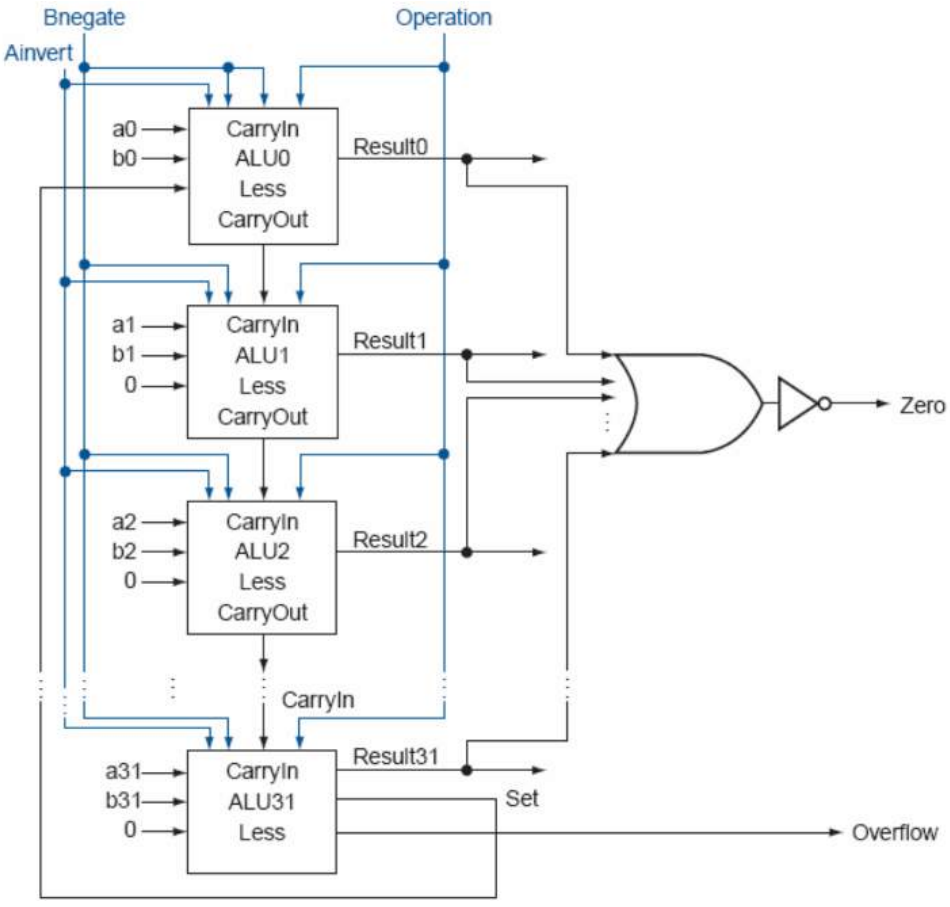
ou Answered 159.3

orrect Answer 158.14

Question 9

10.5 / 14 pts

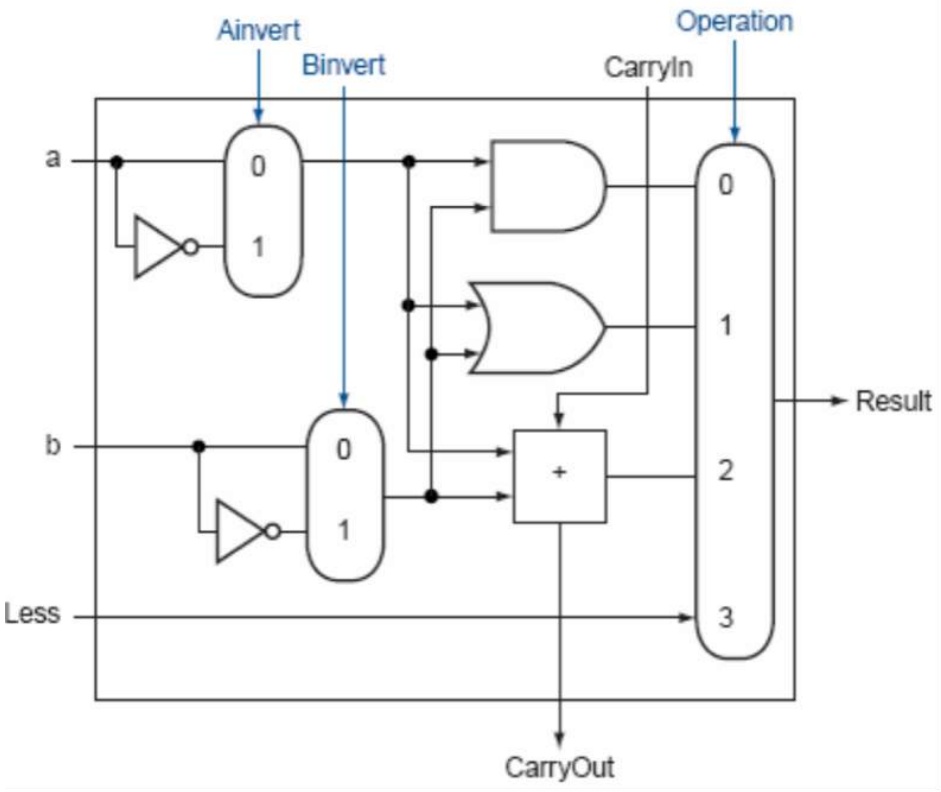
Here is the 6-function 32-bit ALU we covered in class:



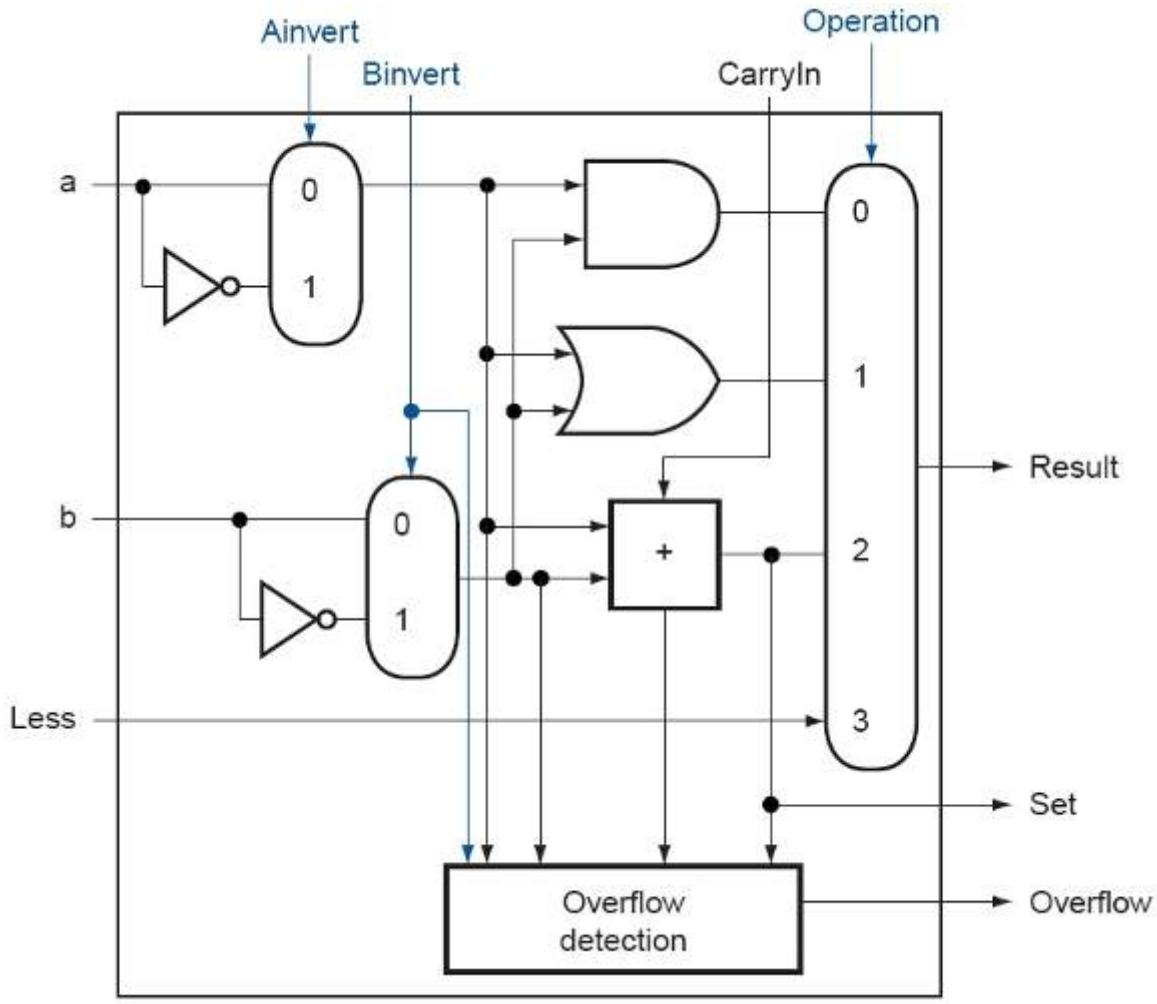
This 32-bit ALU has inputs A and B (broken down into individual bits a0, a1, a2, etc), control signals Ainvert, Bnegate, and Operation, and outputs Result (broken down into individual bits Result0, Result1, etc), Zero, and Overflow. Recall that this ALU includes an implementation of a set-less-than function (slt). The 6 functions and their control signals are shown below:

Function	Ainvert	Binvert	Operation
and	0	0	00
or	0	0	01
add	0	0	10
subtract	0	1	10
slt	0	1	11
nor	1	1	00

As we covered in class, each 1-bit ALU (except the ALU for the most significant bit) has the following design:

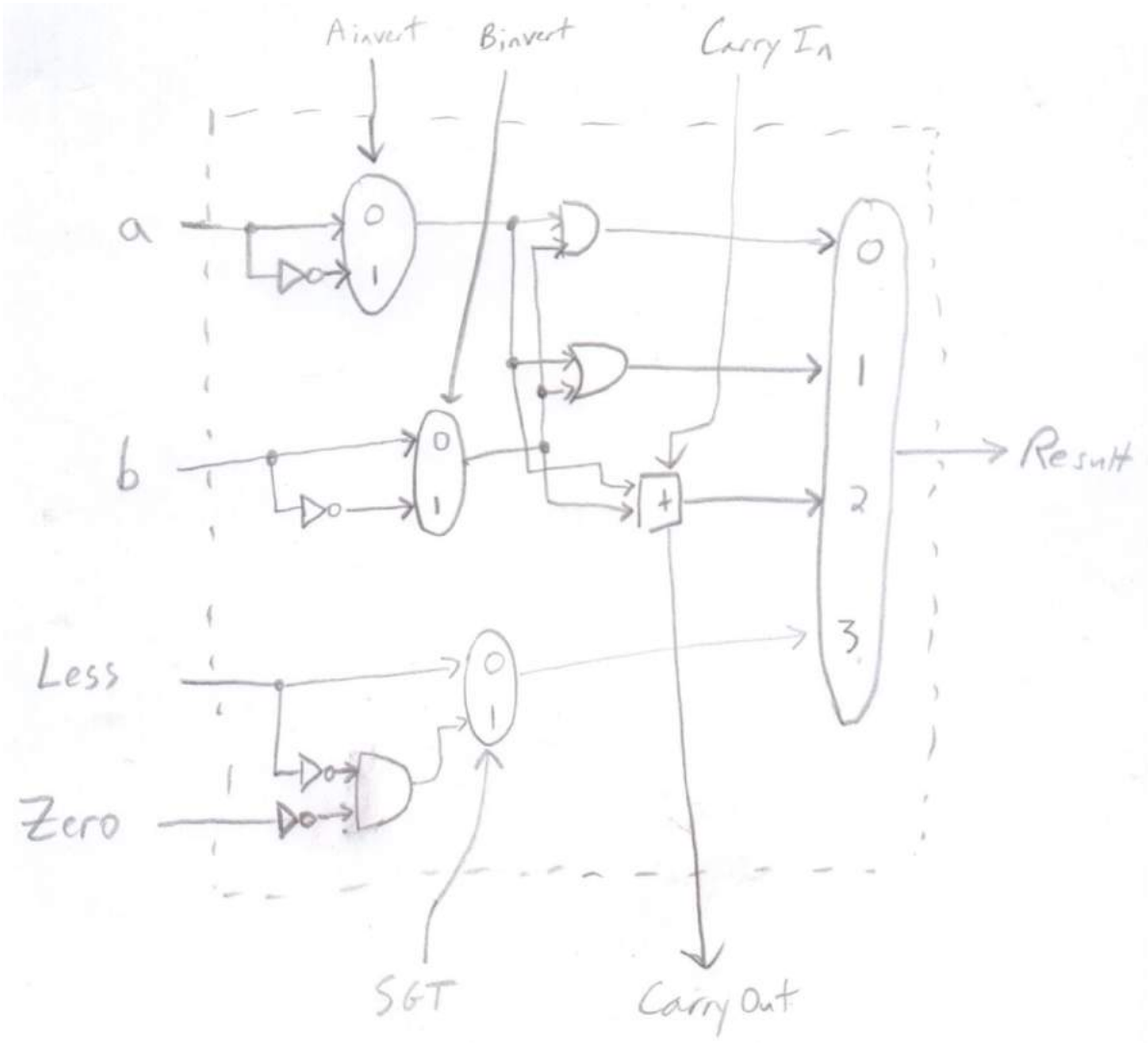


And the ALU for the most significant bit has the following design:



Given this design (again, the one we covered in class), you want to add support for a new function – so you are designing a 7-function 32-bit ALU. The new function will be set-greater-than (sgt) which will set the output to 1 if a > b and 0 otherwise.

Your friend suggests using the slt function as a starting place to building the sgt function. They suggest a change to just the ALU for the least significant bit in this professionally drawn diagram below:



The change here is related to the Less input. There is a control signal (SGT) that selects between the regular less input and the AND of the negation of the Less input and the negation of the Zero output of the entire 7-function 32-bit ALU. Your friend argues that negation of the Less input will be 1 when a ≥ b (NOT slt) and that the negation of the Zero output will be 1 when a ≠ b (just like a BNE instruction). So if both of these are 1, it will be the case that a > b (hence the AND). The other ALUs (other than the least significant bit) are unmodified.

There is an error in your friend's overall design. In at most three sentences describe (1) what is wrong and (2) how to fix it in the space below.

Your Answer:

The error with the design is that the Zero output is output from the entire 32-bit ALU as a whole. Here, we're using the Zero as an input to the ALU logic and an output, when no port for it exists in the LSB ALU. We would fix this by introducing Zero as an input into the LSB ALU.

Quiz Score: **91.5** out of 100