CIS 351 Sample Quizzes Solutions

Mon 6th Dec, 2021

Quiz 1

NR1: Binary

Note: To receive credit, you *must* show your work.

(a) Convert 50 from decimal to binary: 110010

(b) Convert 101011 from binary to decimal: 43

(c) What is the range of integers that can be represented using 7 bits? 0 – 127

(d) How many bits are needed to store integers up to (*and including*) 47? 6 bits

(e) Write the next six successive binary numbers by using the binary counting pattern. Do *not* convert to decimal or any other number base. Add notes to the right documenting your thought process. (You need only enough notes to convince me that you didn't just convert to base 10 on a scratch sheet of paper.)
110101
_____ 110110
_____ 110111
_____ 111000
_____ 111001
_____ 111010
_____ 111011

NR3: Hexadecimal

When converting, go *directly* between binary and hexadecimal. Do *not* convert to decimal

(a) Convert `43ca` from hexadecimal to binary: 100001111001010 (Leading 0s are optional.)

(b) Convert `111101011010010` from binary to hexadecimal: `7ad2` (Subscript is optional)

(c) Write the six successive hexadecimal numbers by following the hexadecimal counting pattern. Do *not* convert to decimal or any other number base. Add notes to the right documenting your thought process. (You need only enough notes to convince me that you didn't just convert to base 10 on a scratch sheet of paper.)

`0x5b98`

_____ 0x5b99
_____ 0x5b9a
_____ 0x5b9b
_____ 0x5b9c
_____ 0x5b9d
_____ 0x5b9e

NR4: Other Number Bases

Note: To receive credit, you *must* show your work.

(a) Convert 2011_3 from base 3 to decimal: 58

(b) Convert 172 from decimal to base 4: 2230_4

CL1: Truth Tables

Construct a truth table that shows the output of a circuit that takes a four-bit integer as input and returns `true` if the input is a multiple of 5.

A	B	C	D	
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

CL2: Boolean Expressions

(a) Complete the truth table below so that it describes the output of this Boolean expression: $xy + \bar{z}$.

X	Y	Z	$xy + \bar{z}$
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

(b) Write a Boolean expression in sum-of-product form that describes the following truth table:

X	Y	Z	
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

$\bar{X}\bar{Y}\bar{Z} + X\bar{Y}Z + XY\bar{Z} + XYZ$

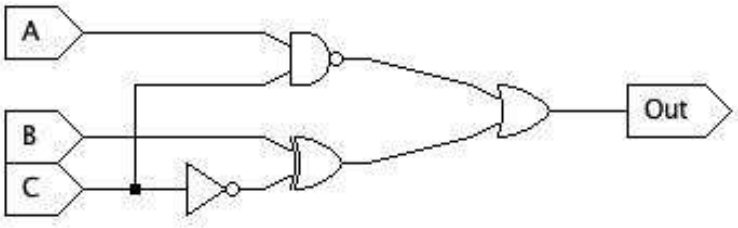
CL3: Circuit Representation

(a) Draw a circuit that implements the truth table below:

A	B	C	Out
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

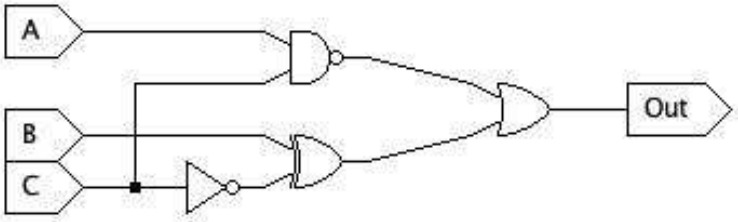
(b) Complete the truth table below to show the output of the circuit for each input:

A	B	C	Out
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1



(c) Draw a combinatorial circuit that implements this Boolean expression: $\overline{(AC)} + (B \oplus \bar{C})$

(d) Write the Boolean expression that implements the following circuit:

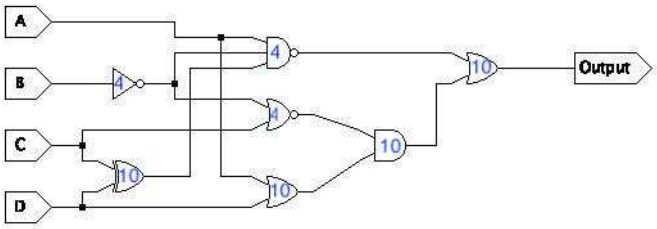


$\overline{(AC)} + (B \oplus \bar{C})$

Quiz 2

CL4: Propagation Delay

Compute the propagation delay of the circuit below given the individual gate delays:



30

ADD1: Binary Addition

Add 11001100 + 01010101 *directly* in binary (i.e., do not convert to decimal first). Show your work so it is clear that you performed the addition in binary.

```
  11001100
+ 01010101
----------
```

```
  11001100
+ 01010101
----------
 100100001
```

ADD2: Ripple Carry Adder

(a) Sketch a full adder.

(b) Sketch an 8-bit ripple carry adder. You may treat full adders as "black boxes".

(c) Consider an 8-bit ripple carry adder with a broken wire between C_{out_3} and C_{in_4}. This wire is "stuck at 0" (i.e., always presents as logical false to C_{in_4}). Assume input A is 10111000. Describe the set of values for B that will result in correct output in spite of the broken wire.

Any input for which B_3 is 0 will work.

FC1: Functional Completeness

Write a formal proof that the set {NOT, AND} is logically complete.

We will prove that {NOT, AND} is logically complete by building the logically complete set {NAND} using only NOT and AND gates. (We proved during lecture that {NAND} is logially complete.)

We can construct a {NAND} gate simply by attanding a NOT gate to the output of an AND gate. (*Be sure to include a picture when you answer this question on a quiz.*)

Beause we can construct a logically complete set using only NOT and AND gates, we have proven that {NOT, AND} is also logicaly complete.

Quiz3

NR2: Two's Complement

To receive credit, you *must* show your work.

(a) Convert -42 from decimal to and *eight-bit* two's complement number: 11010110

(b) Convert 11010100 from two's complement to decimal: -44

(c) What is the range of integers that can be represented using 8 two's complement bits? -128 – 127

(d) How many bits are needed to store integers between -76 and 37 (*inclusive*) using two's complement? 8 bits

(e) Write the six *successive* two's complement numbers by using the pattern. Do *not* convert to decimal or any other number base. Add notes to the right documenting your thought process. (You need only enough notes to convince me that you didn't just convert to base 10 on a scratch sheet of paper.)
11100001
_____ 11100010
_____ 11100011
_____ 11100100
_____ 11100101
_____ 11100110
_____ 11100111

ADD3: Carry Lookahead Adder

(a) Sketch the lookahead logic for the carry into column 4. (Columns are indexed beginning with 0.)
I'll check these individually.

(b) Suppose one were to make a mistake and accidentally use an `AND` gate in place of the rightmost `OR` gate for the carry lookahead logic. Describe the inputs A and B that would produce a value of 1 on the carry out.

The inputs must be all 1s (e.g., $A = 1111$ and $B = 1111$)

ADD4: Carry Select Adder

(a) Sketch the top-level of a 16-bit carry-select adder. I'll check these individually.

(b) Assume that the carry-select pattern is applied at only the top-level, and that the component 8-bit adders are standard ripple-carry adders. Fill in the blanks: The propagation delay of this carry select adder is _____ (more than, exactly, less than) _____ (give a fraction) that of a 16-bit ripple carry adder.

The delay will be *more than* one-half.

(c) Explain your choice of "more than", "exactly", or "less than" for the previous problem.

The 8-bit ripple carry adders will have a propagation delay of half that of a 16-bit ripple carry adder.[1] Those thee adders all run in parallel. After the ripple-carry adders complete, then the muxes must select the final output. These muxes make the total propagation delay slightly longer than half the running time of a 16-bit ripple-carry adder.

ADD5: Recognize Overflow

Given an 4-bit adder, complete the table below to show the carry out and overflow for each pair of inputs.

A		B		C_{out}	Overflow
0	(0000)	0	(0000)	0	0
-1	(1111)	-1	(1111)	1	0
5	(0101)	6	(0110)	0	1

Quiz4

BA1: Boolean Algebra

(a) Use Boolean algebra to show that $(B + \overline{C} + \overline{A}B)(BC + A\overline{B} + AC) \iff BC + A\bar{B}\bar{C}$.

$$(B + \overline{C} + \overline{A}B)(BC + A\overline{B} + AC)$$
$$(BBC + BA\overline{B} + BAC) + (\overline{C}BC + \overline{C}A\overline{B} + \overline{C}AC) + (\overline{A}BBC + \overline{A}BA\overline{B} + \overline{A}BAC)$$
$$(BBC + A\overline{B}B + ABC) + (B\overline{C}C + A\bar{B}\bar{C} + A\overline{C}C) + (\overline{A}BBC + \overline{A}AB\overline{B} + \overline{A}ABC)$$
$$(BC + ABC) + (A\bar{B}\bar{C}) + (\overline{A}BC)$$
$$(BC + \overline{A}BC) + A\bar{B}\bar{C}$$
$$BC + A\bar{B}\bar{C}$$

(b) Apply DeMorgan's law to $\overline{A + B + C(\bar{A} + D)}$ until only single terms are negated. (In other words, you answer may contain $\bar{A}$, but not $\overline{AB}$ or $\overline{A + B}$.)

$$\overline{A + B + C(\bar{A} + D)}$$
$$\bar{A}\bar{B}(\overline{C(\bar{A} + D)})$$
$$\bar{A}\bar{B}(\bar{C} + (\overline{\bar{A} + D}))$$
$$\bar{A}\bar{B}(\bar{C} + A\bar{D})$$
$$\bar{A}\bar{B}\bar{C}$$

CL5: Combinatorial Logic Design

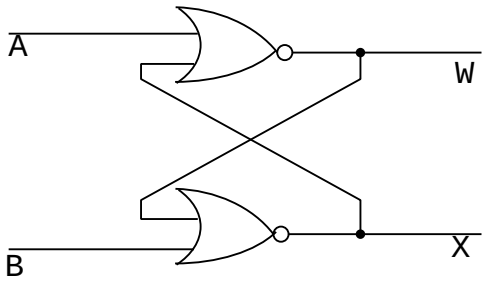
Design a circuit that takes two integers as input and returns the larger integer as output. Assume you have a "less than" circuit. You may draw the `less_than` circuit, muxes, decoders, equality circuits, adders, etc. as "black boxes".

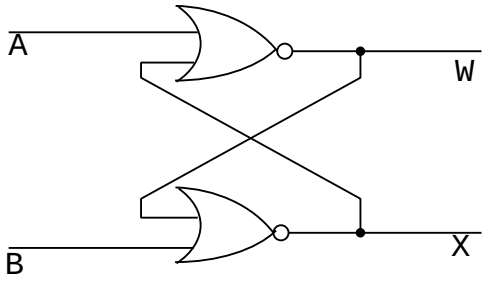
Quiz5

SL1: Latches

(a) Complete the characteristic table for the circuit shown below: Note, there is no clock pulse here. Your answers should show the states W and X after they have reached a steady state given A, B, and current value of W. (Remember to trace the circuit until it has reached a *steady state* — a state in which no further transitions will occur.)

A	B	W_{now}	X_{now}	W_{next}	X_{next}
0	0	0	0	rand	rand
0	0	0	1	0	1
0	0	1	0	1	0
0	0	1	1	rand	rand
0	1	0	0	1	0
0	1	0	1	1	0
0	1	1	0	1	0
0	1	1	1	1	0
1	0	0	0	0	1
1	0	0	1	0	1
1	0	1	0	0	1
1	0	1	1	0	1
1	1	0	0	0	0
1	1	0	1	0	0
1	1	1	0	0	0
1	1	1	1	0	0





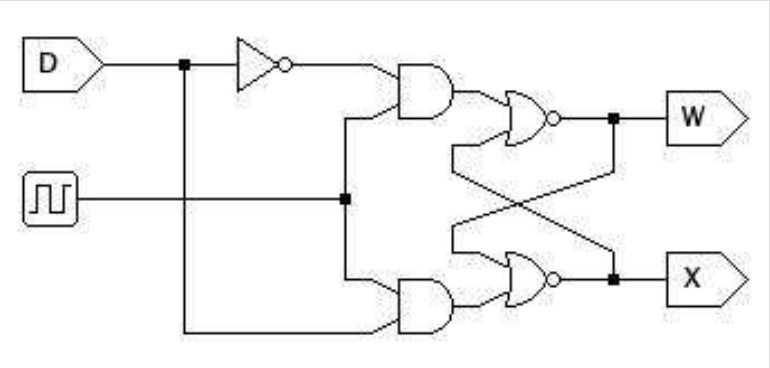
(Extra copy if you need more scratch space.) NOR latch

(b) The above circuit can be used as a latch (provided you avoid the inputs that lead to random state). What input combinations can be used for "set", "reset", and "hold"? (Hint #1: One or both of the inputs may be "active low". Hint #2: Don't assume that W and X should necessarily hold opposite values — that's why they aren't labeled W and $\bar{W}$.)

A = 0; B = 0 → "Hold"
A = 0; B = 1 → "Set"
A = 1; B = 0 → "Reset"
A = 1; B = 1 → "Don't Use"

(c) Explain how the circuit uses a feedback loop to "remember" the current state. Your explaination should, in part, trace the operation of the "hold" input.

(d) Construct a clocked D latch from the circuit above. Remember, the clocked D latch should set its state to the value of the D input whenever the clock is 1, and hold steady when the clock is 0.



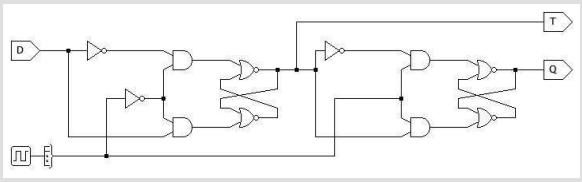
Quiz 6

SL2: Flip-Flops

Consider the master-slave flip-flop shown to the right:

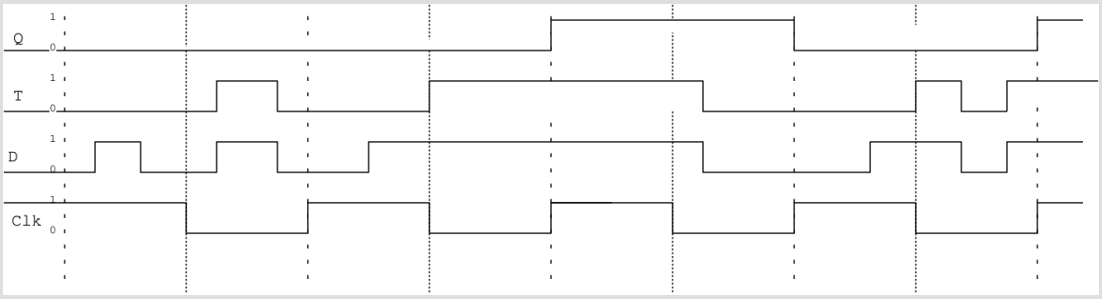
(a) Is this a rising-edge, or falling-edge flip-flop? How can you tell?

It is a *rising-edge* triggered flip-flop. You can tell because the latch attached to the output Q changes values when the clock rises from 0 to 1. (From the perspective of the user, Q contains the current state of the flip-flop.)

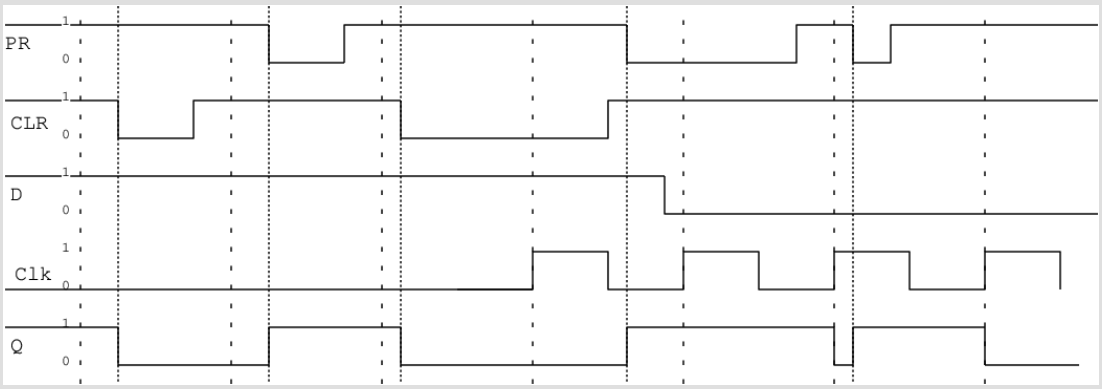


(b) Explain what gives this circuit its "instant trigger point."

(c) Complete the timing diagram below that shows the *internal* operation of the master-slave flip-flop.



(d) Complete the timing diagram below that shows the *external* behavior of the master-slave flip-flop. (This is similar to problems 6-8 on the Sequential Circuit lab.)



SL4: Tracing Sequential Circuits

(a) Complete the characteristic table for the circuit shown below.

Current state			Next State	
B	A_0	A_1	A_0	A_1
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	0	0
1	0	0	1	0
1	0	1	1	1
1	1	0	0	1
1	1	1	0	0

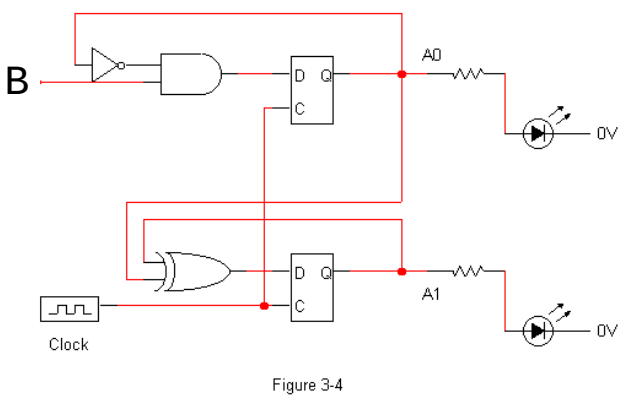
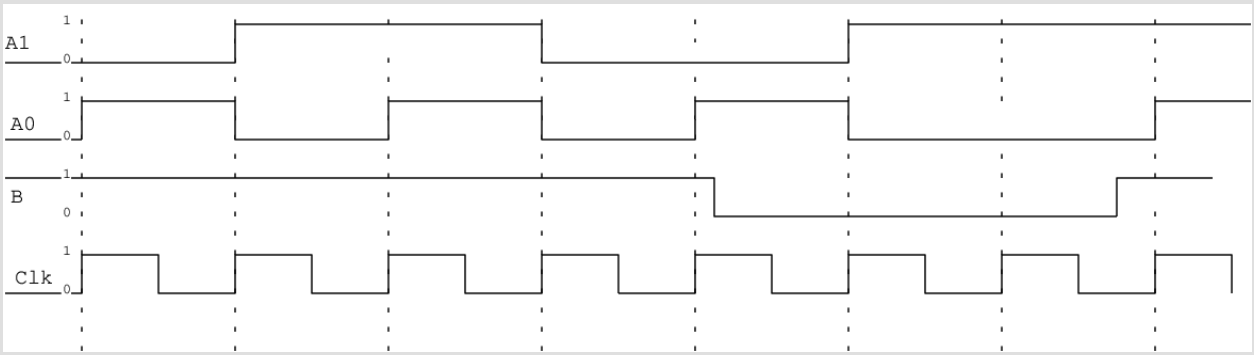


Figure 3-4

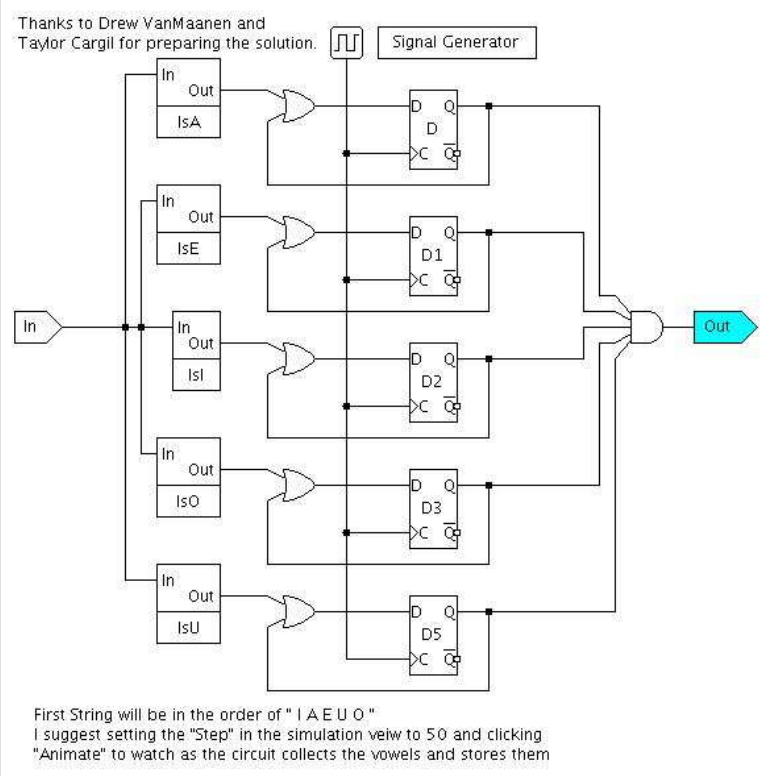
(b) Now complete the following timing diagram. Again, remember that the flip-flops are positive-edge triggered. This means that A_0 *and* A_1 *will change* **only** *when the clock rises from 0 to 1!*



Quiz 7

SL6: Design a Sequential Circuit

Design a sequential circuit that will examine a sequence of characters and determine whether all five vowels are present in the string. The circuit will have one 8-bit input that will contain the ASCII representation of a character. On each clock tick, the input will contain the next character in the sequence. The circuit also has a 1-bit output that is initially `false` and becomes `true` once all five vowels have been observed. You may treat sub-circuits "`is a`", "`is e`", "`is i`", "`is o`", and "`is u`" as "black boxes".



AL1: Basic Assembly

(a) Convert the following line of Java code to assembly: `t0 = t1 + t2 + t3 - t4 + t5`

There are many correct answers. Here is one:

```
add $t0, $t1, $t2
add $t0, $t0, $t3
sub $t0, $t0, $t4
add $t0, $t0, $t5
```

(b) Convert the following line of Java code to assembly: `t0 = (t1 ^ t2) & (t3 | !t4)`

There are many correct answers. Here is one:

```
xor $t0, $t1, $t2   # computes t1 ^ t2
xori $t5, $t4, -1    # computes !t4
or $t5, $t5, $t3    # computes t3 | !t4
and $t0, $t0, $t5
```

(c) Convert the following Java code to assembly. Your answer *must* use `slt` and either `beq` or `bne`. Do not use any pseudoinstructions. Note: This is not a function; it is simply a section of code. Set your code up as if there are more instructions following the block of code (i.e., don't use `jr $ra`).

```
if (t1 - 6 < t2) {
  t0 = t1;
} else {
   t0 = t2 + 4;
}
t1 = t1 + 7
```

There are many correct answers. Here is one:

```
addi $t3, $t1, -6
slt $t4, $t3, $t2
bne $t4, $zero, else
addi $t0, $t1, 0  # does a move
j done
else: addi $t0, $t2, 4
done: addi $t1, $t1, 7
```

(d) Convert the following Java code to assembly:

```
t1 = 0;
for (int t0 = a0; t0 >= 0; t0-= a1) {
  t1 += t0;
}
return t1;
```

There are many correct answers. Here is one:

```
li $t1, 0
move $t0, $a0
top: slt $t2, $t0, $zero
bne $t2, $zero, done
add $t1, $t1, $t0
sub $t0, $t0, $a1
j top
done: move $v0, $t1
```

Quiz 8

AL3: Read Assembly

Describe in common English what the following functions do.

(a)
```
mysteryFunction1:
slt $t0, $a0, $a1
slt $t1, $a1, $a2
and $v0, $t0, $t1
jr $ra
```

It returns whether the parameters are in strictly increasing order.

(b) Describe in common English what the following function does. Hint: It takes two integer parameters. `sra` stands for "shift right arithmetic". It moves all the bits in the register to the right the specified amount.

```
mysteryFunction2:
add $v0, $a0, $a1
sra $v0, $v0, 1
jr $ra
```

It returns the average of the two numbers.

Quiz 9

AL2: Intermediate Assembly

Make each element in an array even by subtracting 1 from each odd value. Assume `a0` contains the address of the array and that the array is terminated by the sentinel value -999.

Quiz 10

SS3: Operation of Single Cycle CPU

Answer the questions below with respect to Figure 1.

(a) For each instruction below, list the value on the `RegDest` control wire. *Explain your reasoning.* (The explanation is the important part. Convince me you did more than memorize the table.) Use "X" for "Don't Care" where appropriate.

add	addi	lw	sw	beq	j
1	0	0	X	X	X

`sw`, `beq`, and `j` don't write to a register, so the output of this mux doesn't matter.

(b) What is the purpose of the value on wire `G`? Describe the information it contains when executing a `lw` instruction. (To be clear, I'm looking for the *purpose* of the information, not the specific value.). "This value is not used for this instruction" a valid choice.

This wire contains the number of the register where the loaded data is to be stored

(c) What is the purpose of the value on wire `J`? Describe the information it contains when executing a `j` instruction. (To be clear, I'm looking for the *purpose* of the information, not the specific value.). "This value is not used for this instruction" a valid choice.

This value is not used by the jump instruction. (The jump instruction does not use the ALU.)

(d) What is the width (in bits) of the wire labeled `J`? Explain your reasoning / how you know this.

Wire J is 32 bits wide. It is an input to the 32-bit ALU.

(e) Suppose wire `I` breaks and provides unreliable values. Which of the follwing instructions will continue to function correctly? Explain your reasoning. `add`, `addi`, `lw`, `sw`, `beq`, `j`

beq and add will break because they use data from two registers. The remaining operations will continue to work correctly because they either (a) send an immediate value to the ALU, or (b) do not use the ALU.

(f) Suppose component `R` breaks and provides unreliable values. Which of the follwing instructions will continue to function correctly? Explain your reasoning. `add`, `addi`, `lw`, `sw`, `beq`, `j`

`sw`, beq, and `j` will continue to work correctly. These instructions do not write to a register, so the output of this mux does not matter.

SS5: Single Cycle Design

It would be difficult to add a `swap_register` instruction to the MIPS CPU (i.e, an instruction that takes two registers and swaps their contents). What would be the main difficulty in adding this instruction?

Such an instruction would need to be able to write to two registers at once. The CPU in the book can only write to one register at a time.

SS5: Single Cycle Design

Consider the instruction `sw R1, offset(R2)`. The CPU in the book implements this instruction by placing `R1` in the "`rt`" position (bits 16-20 in the instruction) and placing `R2` ini the "`rs`" position (bits 21-25) in the instruction.

Would it work equally well to swap the position of these two paramters (i.e., put `R1` on bits 21-25 and put `R2` on bits 16-20)? If so, explain why. If not, explain why not.

It would not work well. The base register `R2` needs to be an input into the ALU. It really only makes sense to make this value the top input into the ALU. If we instead made it the bottom input, we would have to add an extra mux to feed the immediate value into the ALU's top input.

Quiz 11

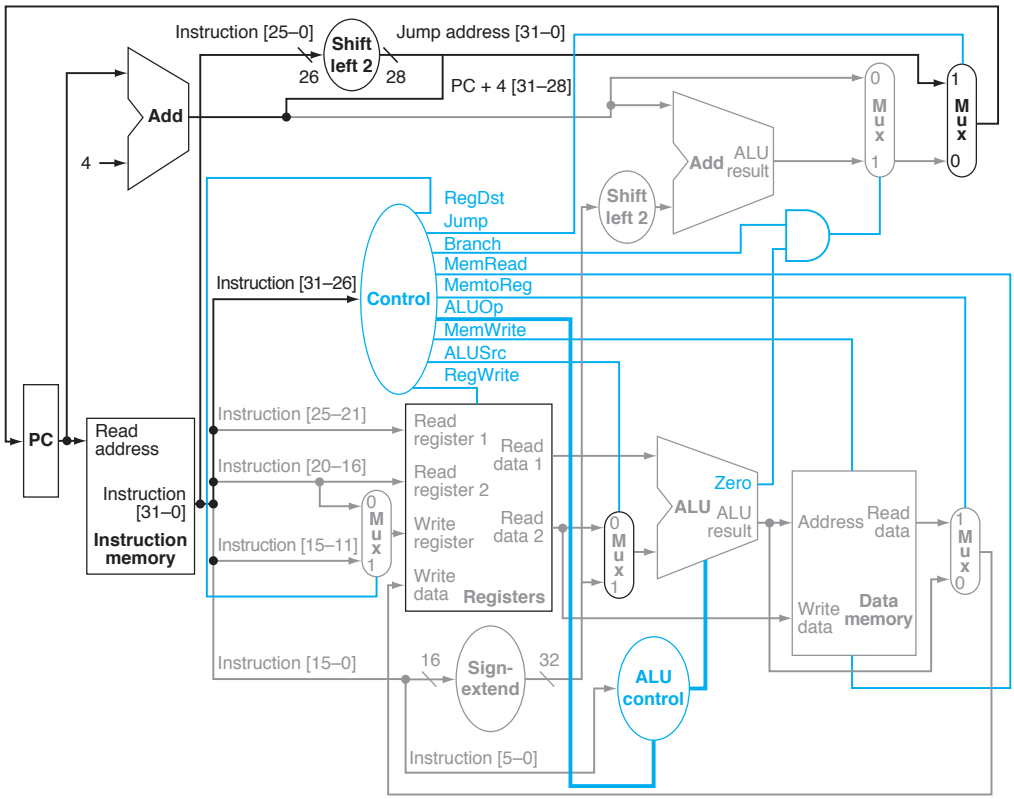
SS4: Modify the Single-Cycle CPU

Consider this following code:

```
for(int x = 0; x < 1000; x++) {
  array[x] = x*x;
}
```

We could save one cycle per loop by combining the storing of data into `array` with the increment of `x` into a new instruction *store word and increment* (`swi R1, R2, increment`). This new instruction will do the following in a single cycle: (1) store the contents of R1 into the memory location contained in R2, and (2) add `increment` to R2 and store the result back in R2. In other words, the instruction does this: `M[R2] = R1; R2 += increment`.

1. Design this instruction. Specify how each bit if the instruction is used. For example, specify which bits contain R1, which contain R2, and which contain the `increment`.
2. Make any necessary changes to the CPU to support this new instruction. You may add additional muxes and control wires if necessary.
3. Specify how each control wire is set for this instruction (including any new control wires you added).



SS4: Modify the Single-Cycle CPU

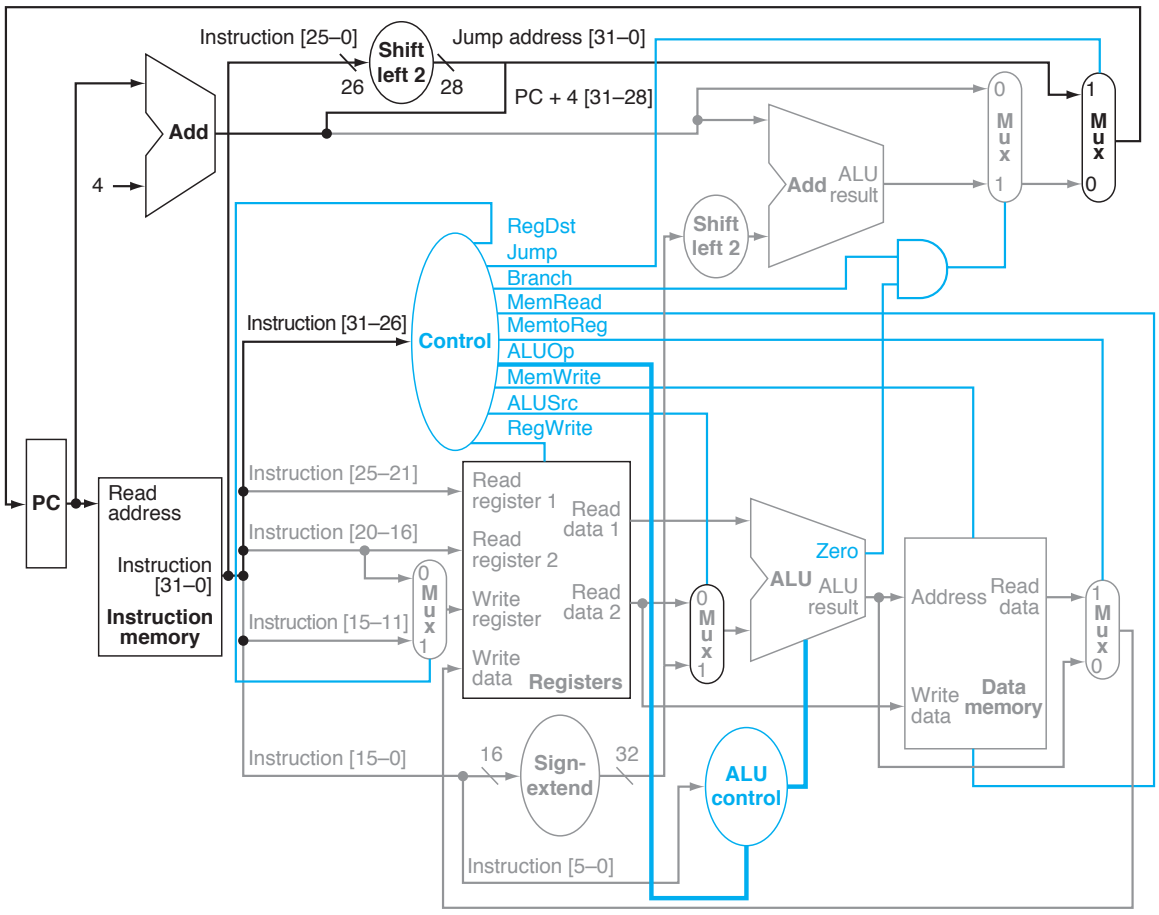
Consider the following `for` statement: `for (rs = 999; rs > 0; rs-= rt)`.

Compiling this code for the single-cycle computer presented in class normally requires at least two instructions: a subtract and a branch. However, it is possible to combine these two operations into a single "decrement and branch if zero" instruction. In particular `dabz $rs, $rt, target` will (1) compute `$rs - $rt`, (2) use that result to decide whether to branch, then (3) write the result back into `$rs`. Register `$rs` is always updated, regardless of whether the branch is taken.

1. Design this instruction. Specify how each bit if the instruction is used. For example, specify which bits contain `rs`, which contain `rt`, and which contain the branch target.

2. Make any necessary changes to the CPU to support this new instruction. You may add additional muxes and control wires if necessary.

3. Specify how each control wire is set for this instruction (including any new control wires you added).



M2: Cache Mechanics

Consider a 512KB, 8-way set associative cache with 16 byte blocks.

(a) Show how 32-bit addresses are divided into offset, index, and tag fields. You must show your work.

Tag, Index, Offset $\longrightarrow$ 16 – 12 – 4

(b) Compute the total size of this cache (including metadata). Assume pseudo-LRU replacement. Give your answer in bytes. Show your work.

The columns after "Correct" are so I can check if the only mistake is forgotten valid and/or LRU bits.

	Correct	No valid	No LRU	No LRU/No Valid
Bits / way	145	144	145	144
Bits / line	1167	1159	1160	1152
Bits / cache	4,780,032	4,747,264	4,751,360	4,718,592
Bytes / cache	597,504.0	593,408.0	593,920.0	589,824.0
KB / cache	583.5KB	579.5KB	580.0KB	576.0KB

(c) Sketch this cache.

M3: Trace Cache Behavior

You have an 128KB, byte addressable, 2-way set associative cache with 16 byte blocks. This cache uses an LRU replacement policy. For the following sequence of addresses, show whether each is a hit or a miss. For each cache miss, tell which bytes were replaced.

Offset: 4
Index: 12
Tag: 16

128KB = 2^{17} bytes.
Lines = $\frac{2^{17}}{2*16}$ = 4096. Thus, 12 index bits.
Tag = 32 - (index + offset) = 16.

Address	Tag	Index	H / M	Tags Replaced	Notes
0x12341110	1234	111	M		Cold
0x12341113	1234	111	H		Same block
0x12351113	1235	111	M		Cold
0x12341113	1234	111	H		
0x12361114	1236	111	M	0x1235	
0x12351113	1235	111	M	0x1234	
0x12361114	1236	111	H		
0x12341110	1234	111	M	0x1235	
0x12361110	1236	111	H		Same block
0x12351113	1235	111	M	0x1234	

M4: Effects of Cache Parameters

(a) Suppose you have a cache configured such that the memory address is broken down into t tag bits, i index bits, and o offset bits. If you were to double the cache size without changing the block size or associativity, how would t, i, and o change?

(b) Explain your reasoning for each parameter above (index, offset, and tag).

(c) How does increasing the cache size affect the number of Cold cache misses? Explain your reasoning.

(d) How does increasing the cache size affect the number of Conflict misses? Explain your reasoning.

P1: Pipeline Structure and Speedup

Consider the following process for baking cookies:

1. Mix sugar, flour, and water using a mixer for six minutes.
2. Pour the mixture into another bowl and add chocolate chips. Stir by hand for four minutes.
3. Spoon the cookie dough onto a baking pan (about four minutes).
4. Place the pan in the oven for fifteen minutes. One pan of cookies takes up the entire oven – you cannot have more than one pan in the oven at once.
5. Place the cookies on a cooling rack (about three minutes). (Assume you have unlimited space on the cooling rack.)

(a) Estimate the total time to bake a single batch of cookies (from when you begin mixing the sugar, flour, and water until all the cookies from the batch are on a cooling rack.) 32 minutes

(b) Suppose you want to pipeline the cookie-baking process, but are working alone. How many stages does your pipeline have? What steps are included in each stage?

Stage 1: Mix, Stir, Spoon
Stage 2: Bake, Remove

(c) What is the primary limitation (i.e., structural hazard) that prevents you from adding a third stage?

Without a friend, you can't remove the cookies from the oven and spoon them onto the pan at the same time. (Actually, you also need to account for the fact that you have only one pan.)

(d) Estimate (i) the total time to bake 10 batches of cookies and (ii) the average time per batch.

194 minutes. Stage 1 takes 14 minutes. At this point, every 18 minutes, a "baking/removing" phase finishes. This is 19.4 minutes per batch.

(e) Your per-batch time is less than the time to make a single batch of cookies. What is the *primary* source of this time savings. (Be specific. Don't just say "pipelining".)

The primary time savings is that you are mixing up batch $n+1$ while batch n is in the oven rather than waiting until batch n is on the cooling rack before beginning ot mix up batch $n+1$.

(f) Suppose your pipeline has k stages. The average time per batch is more than $\frac{1}{k}$ the time for a non-pipelined batch. Explain the primary factor that contributes to this less-than-optimal speedup.

The two stages don't take the same amount of time. Stage 1 takes 14 minutes while Stage 2 takes 18.

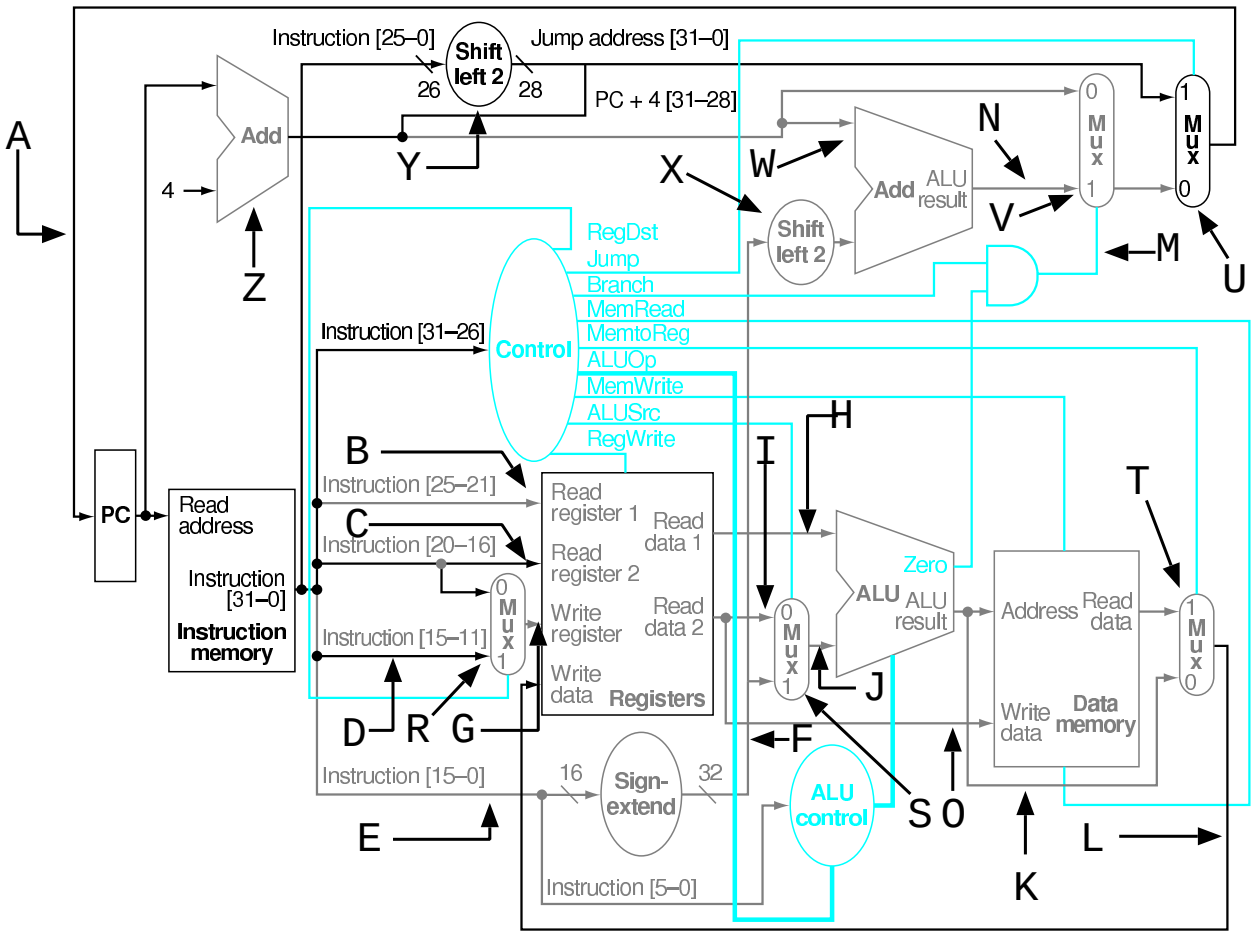


Figure 1: Single Cycle CPU with labeled points