EE457: Computer Systems Organization
Summer 2021 - Midterm Exam
06/22/21, 1:30PM – 3:30PM (Submit by 3:45 p.m)

Name: __Solutions________________

Student ID: ____

Email: ___________________@usc.edu

Lecture section:

Redekopp		
1:30 p.m.		

[0 points: Complete all the information in the box above.]

	Ques.	Your score	Max score	Recommended Time
	-			0 min.
	1		13	10 min.
	2		30	45 min.
	3		12	15 min.
	4		10	20 min.
	5		10	15 min.
	6		5	15 min.
	Total		80	

~~**Note: The last page is blank and can be used for scratch paper.**~~

~~**Please turn it in with your exam** (for in person exams only)~~

1. **(13 pts.) Short Answer** [Fill in the blank at the start of the question with the appropriate answer to the question or the selection that makes the statement true.] **ENTER YOUR ANSWER DIRECTLY ON GRADESCOPE**

1.1. When implementing forwarding logic in our pipelined CPU if both instructions in the MEM and WB state are writing to the same destination register needed by the instruction in the EX stage, the result should be forwarded from the ______ (**[✓ MEM] / WB**) stage?

1.2. We can say that the conditions associated with the ______ (**incoming / [✓ outgoing]**) transition arrows of a state should be mutually exclusive.

1.3. The halfword 0x1234 is written to address 0x89e4. If the byte read from address ________ (**[✓ 0x89e4] / 0x89e5**) is 0x12 then we can say that the system is big-endian.

1.4. Amdahl's law teaches use we should focus on improving the _____ (**[✓ common] / fastest / slowest**) case.

1.5. The ______ (**j / jal / [✓ jr]**) instruction of a 32-bit MIPS processor allows us to jump anywhere in main memory.

1.6. ___________ (**True / [✓ False]**) To branch if $8 > $9 will be accomplished with the sequence: SLT $1, $8, $9; BEQ $1, $0, LABEL;

1.7. ___________ (**[✓ True] / False)** The program counter in the single-cycle CPU does NOT require a write/load enable.

1.8. ___________ (**True / [✓ False])** The load enable signal of a register is produced by a state machine in the control unit. Thus, it needs to be a Moore style output.

1.9. ___________.(**True / [✓ False])** The sum bits in a Carry-Lookahead adder all have the same levels of logic (i.e. roughly the same delay).

Perform the indicated arithmetic operations, showing your work, for the specified representation system. Answers are limited to 8-bits (not 9-bits). (Do not use the borrow method for subtraction, use the 2's complement method of subtraction.) **For 1.10 and 1.12, just write your final 8 bit sum or difference in Gradescope.** Finally, state whether overflow has occurred and **briefly explain why or why not**.

1.10.) 2's comp. system

$$\begin{array}{r} 10011100_2 \\ -\quad \underline{01110010_2} \\ \text{Take A + \textasciitilde B + 1} \\ 00101010_2 \end{array}$$

(Simply type your final 8-bit result on Gradescope)

1.11) Overflow: **[✓ Y] / N**

1.12.) Unsigned system

$$\begin{array}{r} 01010011_2 \\ +\quad \underline{01111101_2} \\ 11010000_2 \end{array}$$

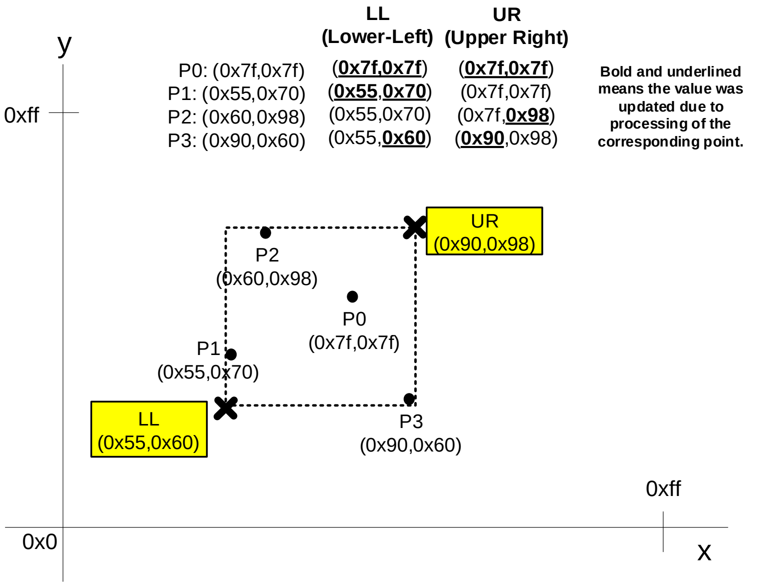
(Simply type your final 8-bit result on Gradescope)

1.13) Overflow: **Y / [✓ N]**

2.1 (22 pts.) Arithmetic, Datapath, and Control Unit Design. Submit PDF on Gradescope Q2

Given an array of x,y points on a Cartesian plane, find their bounding box (the rectangle that encloses all the points). The bounding box can be represented using the x,y coordinates of the **lower left (LL)** point and the **upper-right (UR)** point. Note: The LL and UR points are not necessarily actual points from the array data, but fictitious points that represent the LL and UR vertex of the bounding box. See the illustration below for four points.

The array will contain **20** x,y points. The x and y values are **unsigned** 8-bit numbers (each ranging from 0x00-0xff). The x- and y-value pair will be concatenated and stored as a 16-bit number in one location of a 20x16 memory. The upper 8-bits (bits 15-8) represent the x value and the lower 8-bits (bits 7-0) represent the y value.



	LL (Lower-Left)	**UR (Upper Right)**
P0: (0x7f,0x7f)	(**0x7f,0x7f**)	(**0x7f,0x7f**)
P1: (0x55,0x70)	(**0x55**,**0x70**)	(0x7f,0x7f)
P2: (0x60,0x98)	(0x55,0x70)	(0x7f,**0x98**)
P3: (0x90,0x60)	(0x55,**0x60**)	(**0x90**,0x98)

Bold and underlined means the value was updated due to processing of the corresponding point.

You will read one point at a time and update the value of the LL-x,LL-y and UR-x,UR-y values based on each point. Note that one point may update none, one, or both components of LL and UR, or one component of (potentially both) LL or UR points. For example, P3 in the figure to the right. It's x value updates LL-y (the lowest y value encountered) and UR-x (the right-most x value encountered).

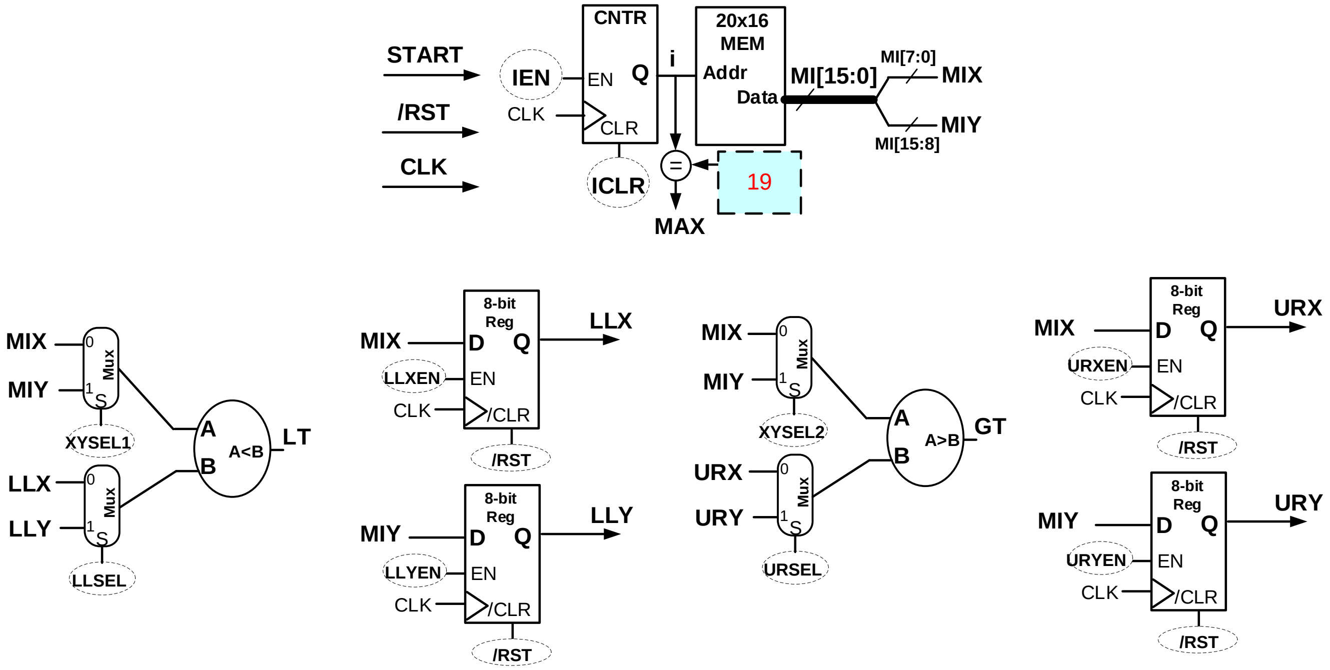
You will complete the state machine (control unit) and datapath on the next pages. The datapath contains **ONLY 2 Comparators**, along with registers to store the x,y components of LL and UR. Due to this limitation of 2 comparators, we may need 2 clock cycles **per point** to perform all the necessary comparisons and update the LL and UR components correctly. However, **you should implement your checks to take as few cycles as possible**. Thus, if the results of the 2 comparisons in the first cycle of comparison make it unnecessary to perform more comparisons in the second cycle, you should not waste time and simply move to the next point and begin comparing it.

We will use a Mealy-style state machine. From an initial state (**QI**), an asserted START signal will cause us to move to the LOAD **(QL)** state state where we initialize both LL and UR x,y components with the 1st point. From there we proceed to process subsequent points taking 1 (**QC1**) or 2 cycles (**QC1 and QC2**) each. Finally, we move to a DONE (**QD**) state.

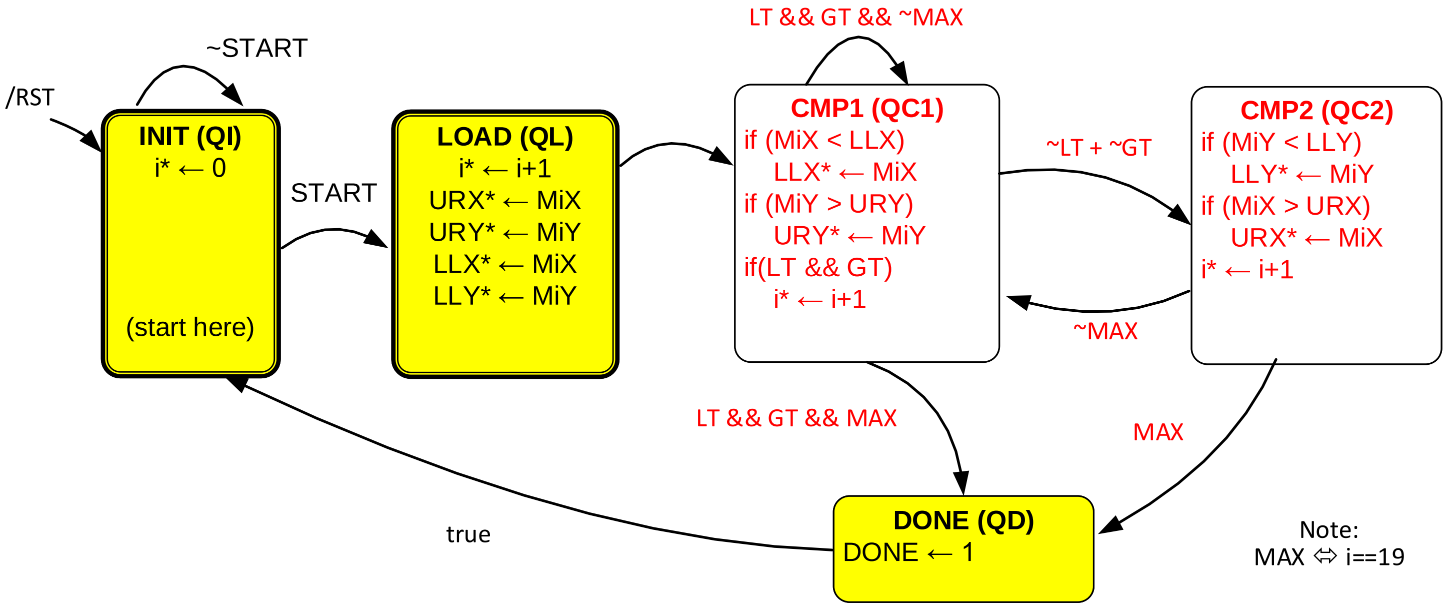
Examine the incomplete datapath on the next page and the skeleton of the state machine on the following page. Complete the state machine and then the data path by filling in the shaded regions, taking care to read the instructions for what is expected. Finally, complete the NSL and OFL.

a.) [1 pt] How many bits should the i-counter be? ____**5** (up 20 rows => 5 bits)_______

b.) [1 pts] The datapath is nearly complete and shown below. Fill in the value that should be compared to i to create MAX which will signal the control unit to move to the DONE state. The signals in the dotted circles will be generated by the OFL of the state machine or other signals from the datapath below. Those will be dealt with on the next page.



c.) [10 pts] Complete the missing transitions & operations for **CMP1 (QC1)** and **CMP2 (QC2)** states. Remember to perform comparisons in **CMP1** that may avoid the need to go to **CMP2** for a given point.



(5 pts.)

e.) Now suppose we implemented the state machine from the previous part and gave you the one-hot state outputs: QI, QL, QC1, QC2 and QD. Use these signals plus **others that are defined in the datapath** to write appropriate logic equations for the control signals indicated below.

LLXEN = __**QL + QC1 * LT**______________________________

IEN = ____ **QL + QC1 * LT * GT + QC2** _________________________

XYSEL1 = ___**QC2**____

XYSEL2 = ___ **QC1**____

LLSEL = _____ **QC2**____

(6 pts.)

f.) Below is a **subset** of the state flip-flops. Assuming a **1-hot state assignment**, complete the NEXT STATE LOGIC equations. You need not draw the gates but just write the equations for the specified D inputs.

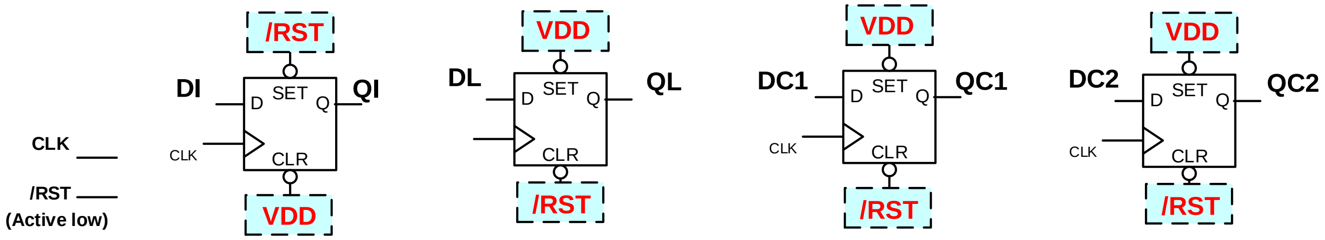
DI = ___ **QD + QI * !START**________________________

DC1 = ___ **QL + QC1 * LT * GT * !MAX + QC2 * !MAX** _______

DC2 = ____ **QC1 * (!LT + !GT)** **// note (!LT + !GT) == !(LT*GT)**___

(2 pts.)

g.) Using /RST, constants and other signals, show how to connect the /SET and /CLR inputs of the following state flip-flops (not all state flip-flops are shown, but those not shown can be ignored).



(5 pts.)

Answer the following questions regarding the datapath and control unit you implemented on the previous pages. **ENTER YOUR ANSWERS DIRECTLY ON GRADESCOPE (choosing T/F).**

2.2.) True/[✓ False]: ____ Suppose each comparator output both A<B and A>B (rather than just one or the other) without any other datapath changes. This would allow processing to take fewer clock cycles.

- I can still only compare two numbers (e.g. MIX and LLX at once and knowing MIX > LLX is not helpful)

2.3.) [✓ True]/False: ____ The 4 data registers do not actually need a /CLR input (i.e. do not need to be reset)

- We will set them to MIX and MIY in the load state (QL) so clearing them to 0 is not necessary.

2.4) True/[✓ False]: ____ The XYSEL1 and XYSEL2 muxes are duplicates and can be merged to a single mux since they mux the same inputs.

- While they mux the same values, they will be selected differently (i.e. they may each need to pass a different value at the same point in time).

2.5) [✓ True]/False: ____ Having 4 comparators of your choice (rather than only 2) would allow for removal of the 4 muxes and allow processing to take fewer clock cycles.

- Yes, we could compare MIX < LLX, MIY < LLY, MIX > URX, and MIY > URY would mean we do not need to share any comparators and thus don't need muxes. We could also do all comparisons in a single state (e.g. QC1) rather than 2 states, thus saving time.

2.6) True/[✓ False]: ____ Keeping the rest of the datapath the same, but having two separate 20x8 memories (separate memory for the X coordinates and another separate memory for the Y coordinates) would allow processing to take fewer clock cycles.

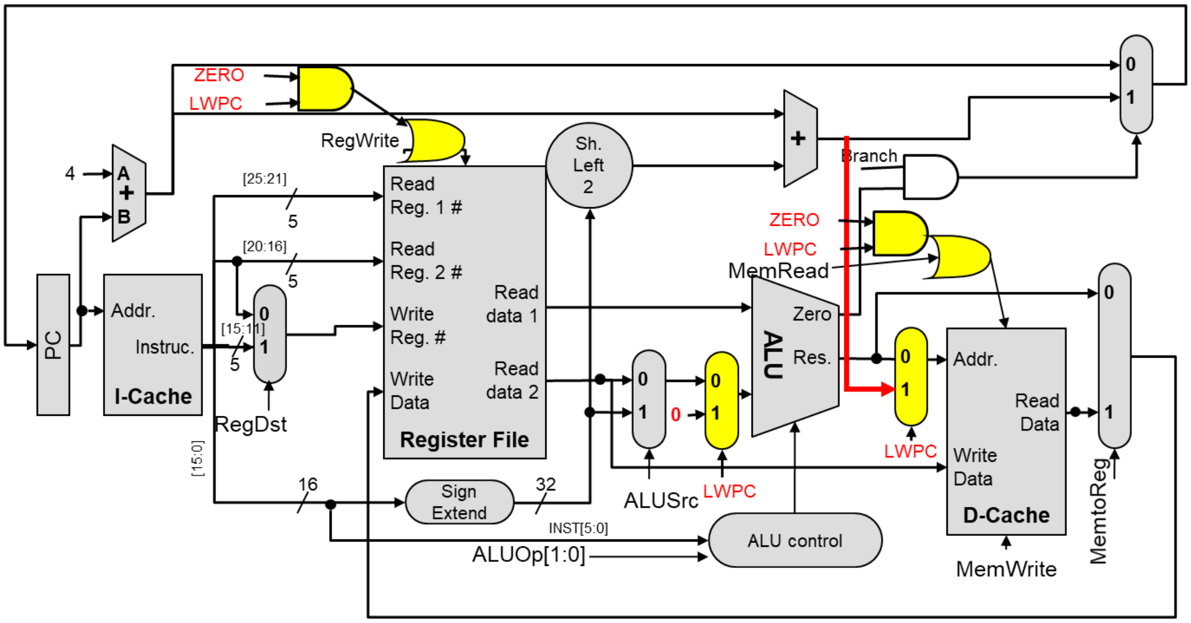
- We get both MIX and MIY components at the same time from the memory. Put another way, we don't need to access two different MI values (i.e. M[i] and M[k], i != k) and thus we don't need 2 memories.

3. **(12 pts.) ISA and Single-Cycle CPU Datapath**: **Submit PDF on Gradescope Q3**
We want to add support for a new instruction '**LWPC**` (**Load Word Using PC address**), while not affecting any other instructions. This new instruction has the format shown below and will load data from memory using the address given by PC + the shifted sign-extended immediate (i.e. $rt = M[`PC + {imm,00}] if Rs==0`), otherwise it continues executing sequentially (the next instruction). Implement any changes to the datapath and control signals to support this new **LWPC** instruction on the single-cycle CPU. Assume when this instruction executes a new **LWPC** control signal will be generated and set to '1'.

`LWPC $rs, $rt, imm`	`Operation:   if($rs == 0)`
	`$rt = M[PC+4+{imm,00}]`

This instruction will use the machine code I-format:

BMNE:	opcode 6	rs 5	rt 5	16-bit signed immediate



a. What operation should the ALU perform? ___SUBTRACTION (for comparing)___________

b. Sketch the additions/changes to the **datapath and control** above that would be needed to support this new instruction and its operations (provide a brief description in the space below if the sketch is unclear). Try not to use more logic/components than necessary (within reason). We've added 2 OR gates that you must complete their second input with additional logic

- Mux to allow 2nd input of ALU to get constant 0
- Mux for memory address to allow for PC+imm as the address
- Control logic over regwrite and memread

c. Show the values of the following control signals to implement this new instruction.

	`LWPC`	`MemToReg`	`Branch`	`ALUSrc`	`RegWrite`	`MemRead`	`RegDst`
Value (0,1,X)	**`1`**	**`1`**	**`0`**	**`X`**	**`0 (may vary)`**	**`0 (may vary)`**	**`0`**

4. (10 pts.) Performance - Submit PDF on Gradescope Q4
Assume a program consisting of 10E6 written instructions executes 50E6 instructions when the program is executed on a pipelined processor that uses a 200 MHz clock period.

a.) Assuming an ideal pipelined CPI of 1, what is the execution time of this program (in seconds)?

Time: __50E6 * 1 * (1/200E6) = 0.25______ **s *(show work)***

b.) Now assume, that branches are determined in the MEM stage (LATE determination) as discussed in class. What is the branch penalty (# of flushed instructions for taken branches)?

Branch Penalty: ___**3**_____

c.) Assume NO penalty for data dependencies and the normal "Not Taken" prediction for branches where we continue to fetch sequential instructions after a branch. Starting from the ideal pipelined CPI of 1, assume **BRANCH instructions account for 20% of instructions** and **60% of BRANCHES are TAKEN**. Given this information, calculate the average CPI.

Average CPI: __1 (all instructions take 1 cycle) + 0.2*0.6*3 (taken branches addition penalty) = 1.36______

d.) Now suppose the architects implement **2 BRANCH DELAY SLOTS**. Analysis shows the compiler can fill the delays slots according to the info below. *Assuming the instruction is a branch*, complete the table below for each case (bulleted list below) indicating the branch penalty (# of wasted cycles / inserted bubbles) then use the table for scratch work to compute the Overall Average CPI of the program running on this processor with 2 delay slots. We recommend using the blank space in each table cell to compute the contribution of that case to the overall average CPI.

- Both delay slots filled for 50% of branches,
- Only 1 delay slot filled for 30% of branches (and use a NOP for the other),
- No delay slots filled for 20% (use NOPs for both slots).

	Taken (60%)	**Not Taken (40%)**
Both delay slots filled (50%)	Branch Penalty (wasted cycles): _**1**__ 0.6*0.5*1 = 0.3	Branch Penalty (wasted cycles): _**0**__ 0.4*0.5*0 = 0
1 delay slot filled (30%)	Branch Penalty (wasted cycles): _**2**__ 0.6*0.3*2 = 0.36	Branch Penalty (wasted cycles): _**1**__ 0.4*0.3*1 = .12
No delay slots filled (20%)	Branch Penalty (wasted cycles): _**3**__ 0.6*0.2*3 = 0.36	Branch Penalty (wasted cycles): _**2**__ 0.4*0.2*2 = .16

Overall Average CPI: _1 + 0.2*(0.3+0.36+0.36+0.12+0.16) = 1 + 0.2*1.3 = 1.26_______

(continued on next page)

e.) Suppose we do NOT use branch delay slots but simply find a way to reduce the **branch penalty for taken branches to only 2 clock cycles** while making the **clock period 5% LONGER**. Would this approach yield faster execution time than the 2 BRANCH DELAY SLOT APPROACH analyzed previously in part d? Show your work

True/**False**: ____-_ This new approach will yield faster execution than the 2 BRANCH DELAY SLOT approach.

Show work:
CPI for this new part e approach: 1 + .2*.6*2 = 1.24

Since IC is the same for both parts, we leave that out and compare CPI and Period (P)
Part d: CPI * Period = **1.26 * 1P = 1.26P …compared to …**
Part e: CPI * Period = 1.24 * 1.05P = **1.302 P (worse)**

5. (10 pts) Instruction Sets - Submit PDF on Gradescope Q5

Given the following code snippets in C and its corresponding MIPS assembly, fill in the missing blanks. The following variables are allocated contiguously in memory starting at address **`0x5555a180`** (recall int's are 4-bytes, shorts are 2-bytes, chars are 1-byte). Then reorder the instructions of snippet 2 to avoid as many stalls due to data hazards as possible.

```
int v;  // variables allocated contiguously starting at 0x5555a180
unsigned short x,y;
char z;
```

Assuming the following code is executed before each code snipped:

```
lui 0x5555, $4
ori $4, $4, 0xa180
```

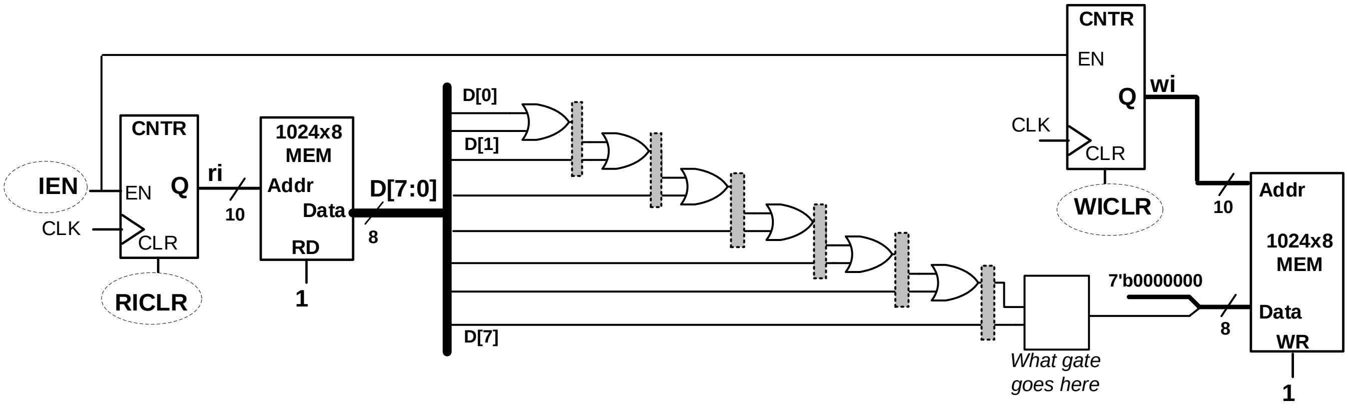
Snippet 1 C Code	**Snippet 2** C Code	Reorder the instructions (3-6) of just **Snippet 2** to the left to avoid as many stalls due to data hazards as possible.

`v = v >> __y__;`
` // v right shifted by some amount.`
` // Fill in the correct variable`
	`v -= __z__;`
	`x++;`	

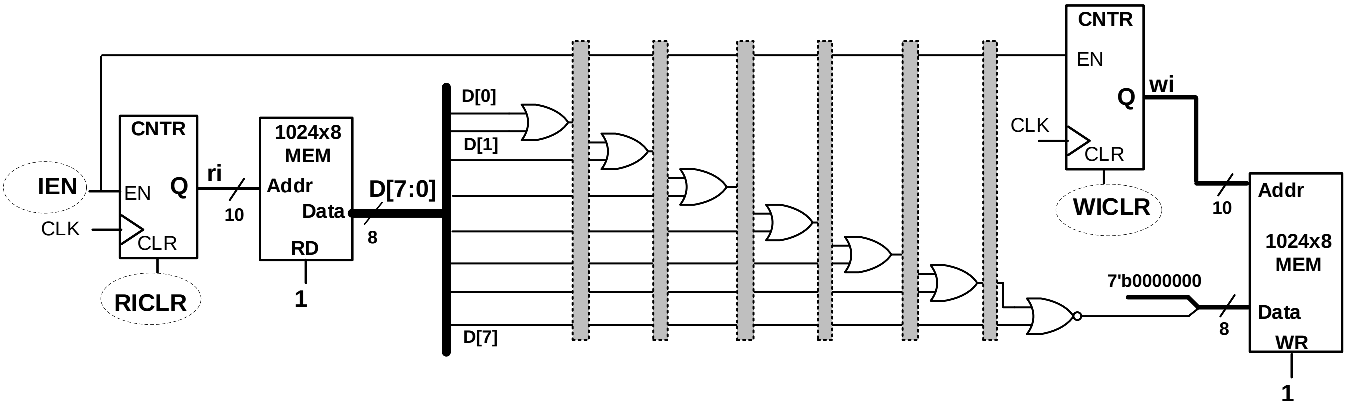
Corresponding MIPS assembly	Corresponding MIPS assembly	
`lw      $5, 0($4)`	`lw    $5, 0($4)`	1
`_lhu___ $6, 6($4)`	`l_b_  $6, 8($4)`	2
`_srav____ $5, $5, $6  <==Choose the`	`sub   $5, _$5_, _$6_`	`__4___`
(srav / srlv) `correct opcode`	`sw    $5, _ 0_($4)`	`__(5 or 6)___`
	`lhu   $7, 4($4)`	`__3___`
`sw      $5, __0__($4)`	`addi_ $7, $7, 1`	`__(5 or 6)___`
	`sh   _$7__, 4($4)`	7

6. **(5 pts.) Pipelining – Submit PDF on Gradescope Q6**
Consider an array/memory of 1024, 8-bit (byte) values, D[i]. Each byte can be 0x00-0xff. We want to convert each byte to a simple true / false value by taking the logical not (!D[i]) and save them to a new array/memory. So if the byte is 0x01-0xff we convert it to 0x00, but if the byte 0x00 we convert it to 0x01. Billy Bruin creates an initial working design but then tries to pipeline it to make it faster by adding pipeline registers (the shaded and dotted rectangles) in the diagram below.

   a. First, to produce the correct value, what type of gate is necessary in the blank box at the end of the pipeline (*NAND / NOR / AND / OR / XOR*) (choose 1): ___NOR____



   b. Tina Trojan looked at the design and saw some flaws in Billy's pipeline attempt. Without rearranging the OR gates, show where additional pipelining registers are needed by drawing additional shaded rectangles where pipeline registers would be needed.



Intentionally blank for scratch work. Please turn it in with your exam:

Name: ____ Section time: _____________