ECE411 Computer Organization and Design
Mid-Term Exam 1
10/11/2018

NetID:_____

Exam Guidelines:

1. This exam has **6 problems.** Make sure you have a complete exam before you begin **[20 pages + the cover page]**.
2. Write your name on every page in case pages become separated during grading.
3. You will have **3 hours** to complete this exam.
4. Write all of your answers on the exam itself. If you need more space to answer a given problem, continue on the back of the page, but clearly indicate that you have done so.
5. This exam is closed-book. You may use one sheet of notes. You may use a calculator.
6. **DO NOT** do anything that might be perceived as cheating. The minimum penalty will be a grade of zero.
7. Show all of your work on all problems. Correct answers that do not include work demonstrating how they were generated may not receive full credit, and answers that show no work cannot receive partial credit.
8. The exam is meant to test your understanding. Ample time has been provided. So be patient and read the questions/problems carefully before you answer.
9. **Good luck!**

Problem	Points
1. 24 pt	
2. 22 pt	
3. 30 pt	
4. 30 pt	
5. 40 pt	
6. 22 pt	
Total: 168 pt	

Name:_____

Question 1: Processor Pipelining

Given the following 5 stage pipeline with WB->EX & MEM->EX forwarding and register file bypass logic

Fetch stage combinational delay = 4ns
Decode stage combinational delay = 5ns
Execute stage combinational delay = 7ns
Memory stage combinational delay = 5ns
Writeback stage combinational delay = 4ns

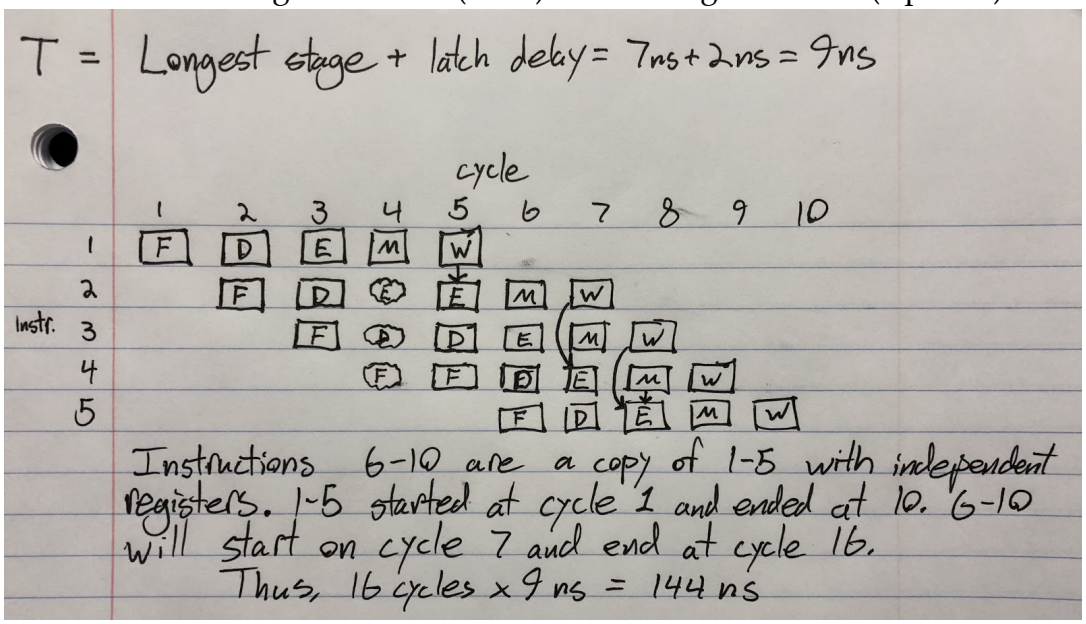
Code:

```
ld   r1, DATA0 ; r1 <= M[DATA0]
andi r3, r1, 0 ; r3 <= r1 & 0x0
ld   r2, DATA1 ; r2 <= M[DATA1]
add  r3, r3, r1; r3 <= r3 & r1
add  r3, r3, r2; r3 <= r3 & r2

ld   r4, DATA2
andi r6, r4, 0
ld   r5, DATA3
add  r6, r6, r4
add  r6, r6, r5
```

Assume latch delay is 2ns. Assume the processor operates at maximum possible safe frequency. Assume 100% cache hit rates and branch prediction rates.

A. Calculate how long it will take (in ns) to run the given code. (6 points)

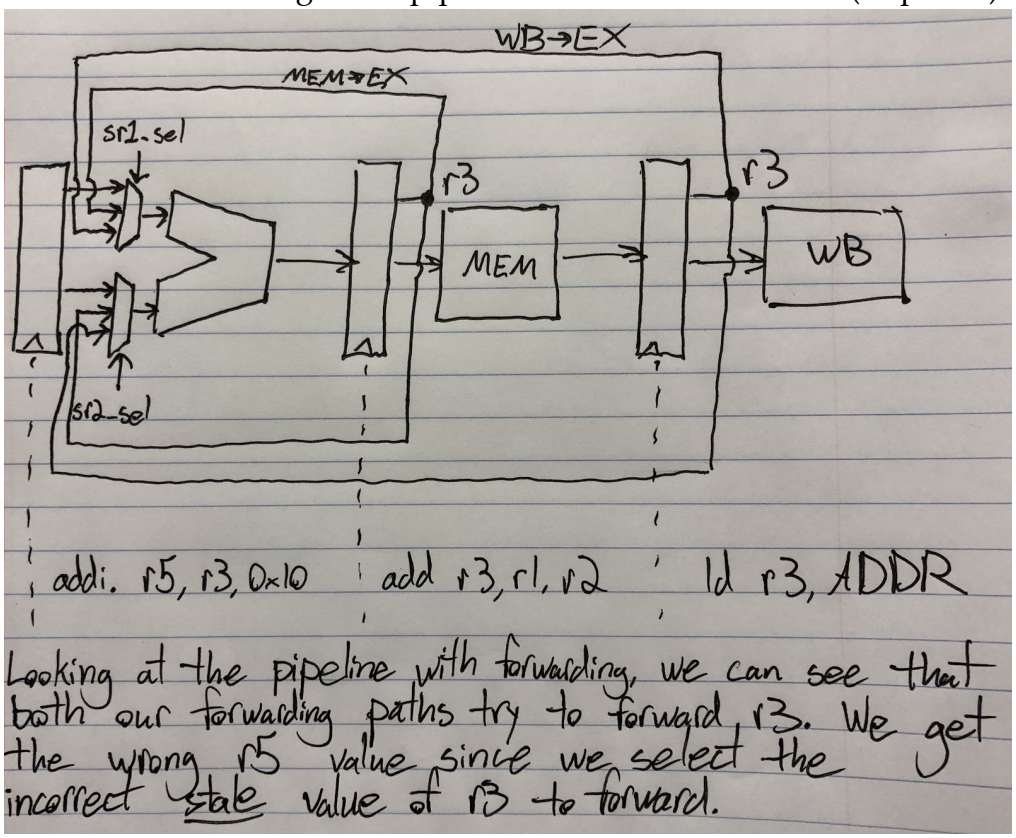


Name:_____

You now run the following code:

```
ld   r3, ADDR
add  r3, r1, r2
addi r5, r3, 0x10
```

B. You discover that this code produces the wrong value for r5 even after producing the correct result for the code at the beginning of the question. What might be a potential hardware bug in the pipeline that causes this error? (10 points)



C. You are asked to evaluate the benefits of adding an additional WB->MEM forwarding path to the pipeline. Write an example piece of assembly code that could benefit from this new forwarding path. (8 points)

```
add r1, r1, 0x1
st  r1, addr
```

Note: The second instruction *must* be a store in order to take advantage of the new forwarding path. Also, the forwarded value *must* be the source register for the store. The address for a memory operation is calculated in the execute stage!

Name:_____

Question 2: Caches

Consider an architecture with a 32-bit address range and byte addressable memory. In order to exploit locality, you as the logic designer choose to implement a cache with 64-byte cache lines. 'Cache Type' refers to Direct Mapped, Set-Associative or Fully Associative in this question.

A. Assume each set has 128 bytes of data. There are 47 bits of bookkeeping (tag, valid, dirty and LRU) overhead per set. How many bits are allocated per tag, index and offset? (6 points)

Byte addressable memory: log2(64 bytes) = 6 bits of offset

64 byte cache lines with 128 bytes of data per set -> 2 lines per set. (2-Way Set Assoc)
For 2-way set associative writeback cache: 2 bits of valid, 2 bits of dirty, 1 bit of LRU.
47 - 2 bits of Valid - 2 bits of Dirty - 1 bit LRU = 42 bits of Tag per set.
42 bits of Tag per set / 2 ways per set = 21 bits of Tag per cache line.

32 bit address => 21 bits of tag, 6 bits of offset leaves 5 bits for the index.

21-bit Tag, 5-bit Index, 6-bit Offset.

B. Replacement policies are a major contributor to how well each way is utilized and the hit rate of the cache. Least Recently Used is a straightforward replacement policy with two common flavors; Pseudo LRU and True LRU.

How many bits are needed per set for a tree-based pseudo LRU scheme for a 16-way set associative cache? What is the bit storage overhead of using a queue to track True LRU for n-ways where n > 2? (6 points)

Pseudo LRU: One bit per 2-ways to correctly distinguish LRU. Going up the tree, 8 + 4 + 2 + 1 = 15 bits. (N-1).

True LRU: ceil(log2(n)) to correctly encode each way, n entries in the queue. n*log2(n) overhead.

C. Prefetching is a scheme devised to predict what data will be useful in the near future and bring it into the caches before the CPU needs to access it. This allows potentially useful memory accesses to be overlapped with execution of non-memory operations.

Assume that we can build a highly accurate prefetcher that will only issue prefetch accesses to data that will be used at some point during program execution. Provide a scenario where performance degrades. (4 points)

The prefetcher will only issue prefetches to data that is useful at some point in the execution, but not necessarily immediately useful. If the issued prefetch brings data that will be useful much later, it can evict data in the cache that is more immediately useful, thus causing an additional miss.

If the miss causes a writeback or if the cache is blocked servicing a main memory read, those will also slow down the processor. The problem stems from the timing of the prefetch in this case, and not the accuracy

D. The program below shows nested for loops loading and operating on some data from the structure[] array. A stride is used to determine the address difference between each subsequent load. Given the following program, answer the listed questions.

```
int structure[...]; // a very large array
uint32_t stride = 128;
for (int j = 0; j < 32; j++) {
   stride = stride * 2;

   for (uint32_t i = 0; i < 16; i ++) {
        // Load an element from some structure
        int cur_element = structure[i*stride];

        // Operate on the cur_element
        ...
   }
}
```

You observe that the memory access times are increasing in the inner loop as the stride increases, which causes a decrease in performance. The memory access time saturates at very large strides. This behavior is caused by a near-zero hit rate in the data cache. What type of cache might be in the processor running the code above? Why? (6 points)

Direct Mapped or Set-Associative with low associativity (depending on the loop bound)

The stride is increasing between every iteration. This increases the address difference between subsequent accesses. The cache indexing scheme is organized as follows:

Address => {Tag, Index, Offset}

As the address jumps grow large, fewer and fewer offset and index bits change (if any at all). This causes a smaller subset of cache indices to be used. If the tag is the only part that changes, the accesses all map to the same index. If every access has a different tag and goes to the same index, then every access will evict the previously loaded data (or one of the previously loaded once in the case of set-associative with an LRU replacement policy). The higher the associativity, the higher the hit rate in this problem. Direct-Mapped will have 0 hit rate (or some, if there is an operating system running on top of the processor). Set-Associative will actually hit sometimes since the strided accesses will overlap with the previous iterations of the loop early on, but the hit rate is still very low due to the same problem as the direct-mapped cache.

Fully associative does not use index bits and therefore everything already maps to the same index but each entry is checked in parallel. It will have the highest hit rate out of the three cache types.

Name:_____

Question 3: Caches and Virtual Memory

Congratulations! you have accepted an offer from a new startup called "NoHope" as a HW architect in their processor design group. They are currently working on their cache design. The team leader explained their current preference - using virtually-indexed, virtually-tagged (VIVT) L1 instruction and data caches. After looking at their cache designs, you realized that they didn't handle any of the known issues associated with VIVT caches.

A. After hearing you explain the issues, your team leader got convinced and asked you to propose 2 software solutions each for a minimum of two VIVT issues. What would be your solutions? (8 points)

Homonym:
- OS flushes the caches on context switch
- Force non-overlapping address-space layout

Synonym:
- Restrict VM mapping so synonyms map to same cache set
- Copy-on-write or read-only shared pages
- OS flushes the caches on context switch (doesn't help for aliasing within address space)

B. Unfortunately, the team leader worries about performance degradation and complexity typical of software solutions and asks you to suggest two alternatives (hardware only or hardware supported by software solution). What are you going to propose (caches still need to be VIVT)? (8 points)

Homonym:
- HW flushes the caches on context switch
- Tag VA with address-space ID (ASID)
- Use physical tags

Synonym:
- HW flushes the caches on context switch (doesn't help for aliasing within address space)
- Max no sets = page size / cache line size
- MIPS R10000's solution

Name:_____

C. You also found out that they are using an 8 KB virtually indexed, physically tagged (VIPT) cache as the L2 cache. However, they would now like to double the size of the L2 cache (since new benchmarks have emerged) and decide on the associativity. With the information given in the table below what is the associativity that you recommend and why? (6 points)

Virtual address size	Page size	Size of each page table entry
32 bits	4 KB	8 bytes

4 way-set associative

D. Frustrated by the poor design choices made by NoHope, you decide to interview with other processor design companies. During an interview at ALittleHope, you have been given the following information:

- A processor with a 32-bit virtual address space.
- A 64KB of physical memory space.
- The size of a page is 1KB.
- TLB is a direct-mapped cache with 4 entries.

You are then asked:

a. How many bits are stored for each TLB tag entry? (4 points)

20 bits (VPN - index bits stored as a tag, would have been 22 bits if tlb were fully associative)

b. How many bits are stored for each TLB data entry? (4 points)

6 bits - PPN

Name:_____

Question 4: ISA

Consider the following ISAs and instruction latencies for some simple multicycle implementation (like your MP1 design) of each ISA (do not worry about cache misses and hits, assume AMAT is baked into the latency of all instructions that have memory accesses):

RegReg: a register-register ISA (assume 16 general purpose registers named r0-r15) (140MHz)

Assembly Instruction formats	Instruction latency (# of cycles)
load DR, label	8
store SR, label	8
add DR, SR1, SR2	6
sub DR, SR1, SR2	6
mul DR, SR1, SR2	10

Stack: a stack ISA (100MHz)

Assembly Instruction formats	Instruction latency (# of cycles)
push label	8
pop label	8
add	6
sub	6
mul	10

Name:_____

MemMem: a memory-memory ISA (140MHz) (label1 is always the destination)

Assembly Instruction formats	Instruction latency (# of cycles)
add label1, label2, label3	12
sub label1, label2, label3	12
mul label1, label2, label3	16
mov label1, label2	9

A) Write assembly code to execute the following C function in each ISA with no optimization: (12 pts)

```
/* global variables, no need to produce code for this part, just
assume the labels are available to use */
int A;
int B;
int C;
int tmp;
int result;

void foo() {
    result = result + (B*(C-A));
}
```

As always, write any assumptions you make.

RegReg:

load x0, result
load x1, B
load x2, C
load x3, A
sub x4, x2, x3
mul x5, x1, x4
add x6, x0, x5
store x6, result

Stack:

push result
push B
push A
push C
sub
mul
add
pop label

MemMem:
sub tmp, C, A
mul tmp, B, tmp
add result, result, tmp

B) Imagine you want to add an atomic integer addition instruction to one of these ISAs. An atomic operation should write it's result to memory before any other thread has a chance to overwrite the memory operands of the instruction. Assume this is done by simply asserting a lock signal on the bus interface across multiple read/write operations (like the mem_read signal is asserted to do a read). This amendment would be easiest to implement with which ISA and why (6 pts)?
MemMem because you simply copy the control states of the add instruction and apply "lock = 1" to each.

C) What do you think would be the latency of this new instruction in number of cycles and why (6 pts)?
12 +/- some error if a good reason is given, also allowing for different answers if it follows logically from an incorrect answer in (C)

D) Which ISA uses the instruction cache most efficiently, considering that labels take many bits to encode (much more than register operands) and that compilers must emit more instructions for some ISAs than others? Why? (6 pts)
Any answer accepted if a good explanation is given

Question 5: MPs

This question contains multiple parts and aims to assess your ability to understand a specification and implement desired functionality using good design practices.

Programmers that optimize their code for high performance often desire information about the utilization of the underlying hardware during the execution of their programs. In order to provide this information, modern processors include blocks which monitor the state of the machine and instructions which allow access to these state registers.

a. You are tasked with writing a module that tracks the number of instructions executed by your processor as well as the number of cycles spent waiting on memory accesses. You should write this code in the provided skeleton below. You are told that the *enable* signal will be high for exactly one cycle during the execution of each unique instruction. The memory signals, *mem_read*, *mem_write*, and *mem_resp* behave identically to the class MPs. The *rst* signal is **active low** and should set your counters to zero. (4 pts)

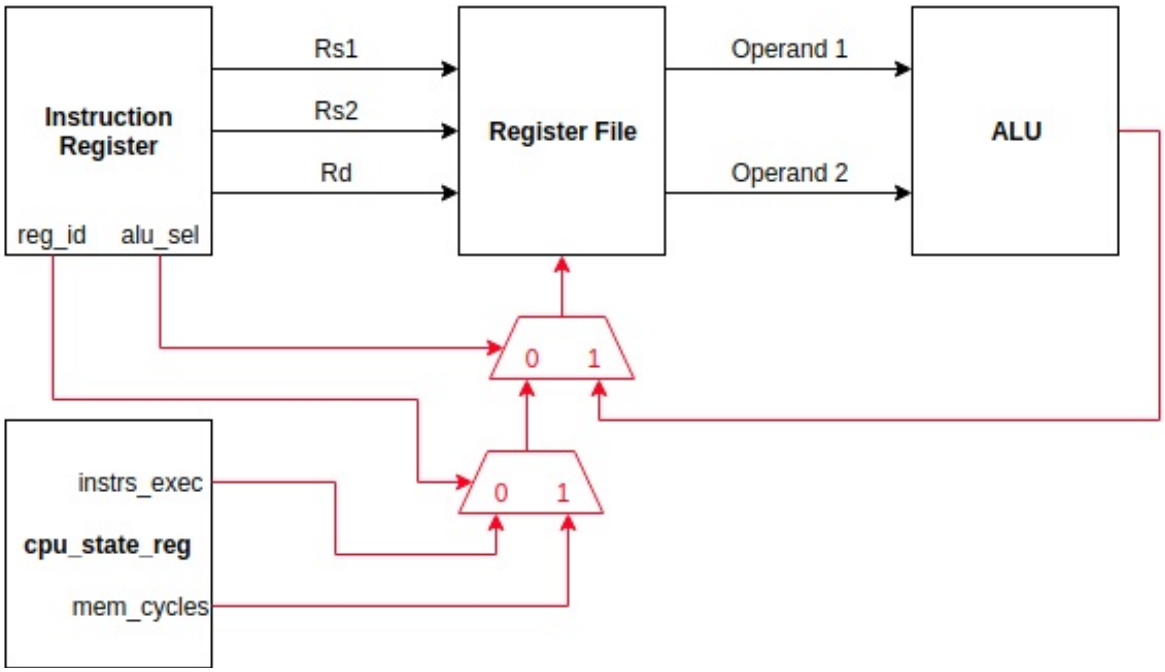
```
module cpu_state_reg (
        input logic clk, rst, enable, mem_read, mem_write, mem_resp,
        output logic[31:0] instrs_exec, mem_cycles
);
// Declare internal signals here
Logic waiting_on_mem; // Optional

// Combinational logic
always_comb begin
        Waiting_on_mem = (mem_read | mem_write) & ~mem_resp; // See below
end
// Sequential logic
always_ff @(posedge clk) begin
        if(~rst) begin // Active low, System-wide reset
                Instrs_exec <= '0; // Both get cleared
                Mem_cycles <= '0;
        End else begin // The fact that this is in the else case gives the rst priority
                if(enable) // Following given spec above
                        Instrs_exec <= instrs_exec + 1; // Could also be comb. logic above **
                if(waiting_on_mem) // Or replace this with the above eqn.
                        Mem_cycles <= mem_cycles + 1; // Same as above
        End
end
endmodule : cpu_state_reg
```

**Many students tried to set the registers to something (like 0 or rst) in the always_comb block, and then used the same signals as registers in the always_ff block. It should be clear why this doesn't work. If it's not post, on Piazza or come to my office hours to discuss.

Now that your design can keep track of its state, you must add support to access these registers and allow a user to see the statistics of their program. The new instruction, *rfsr*, stands for "read from state register" and has the following format: *rfsr Rd, <reg_id>* . This instruction reads from the module you created above and writes the counter value into the destination register, *Rd*. For this problem, *<reg_id>* is a 1-bit signal and can be thought of as the "address" of the counter which should be read. Let *reg_id = 0* correspond to the instruction count and *reg_id = 1* correspond to the memory cycle count. In real processors, there are many different registers to choose from.

b. Modify the partial datapath below by filling in the missing logic required to support the new *rfsr* instruction. Include labels where needed. Write a sentence or two describing each of the changes you've made to the datapath. If you utilize a block other than logic gates or muxes, you must explain the function of the block. *Alu_sel* is a 1-bit signal which is high when an ALU operation is being performed and low otherwise. (5 pts)



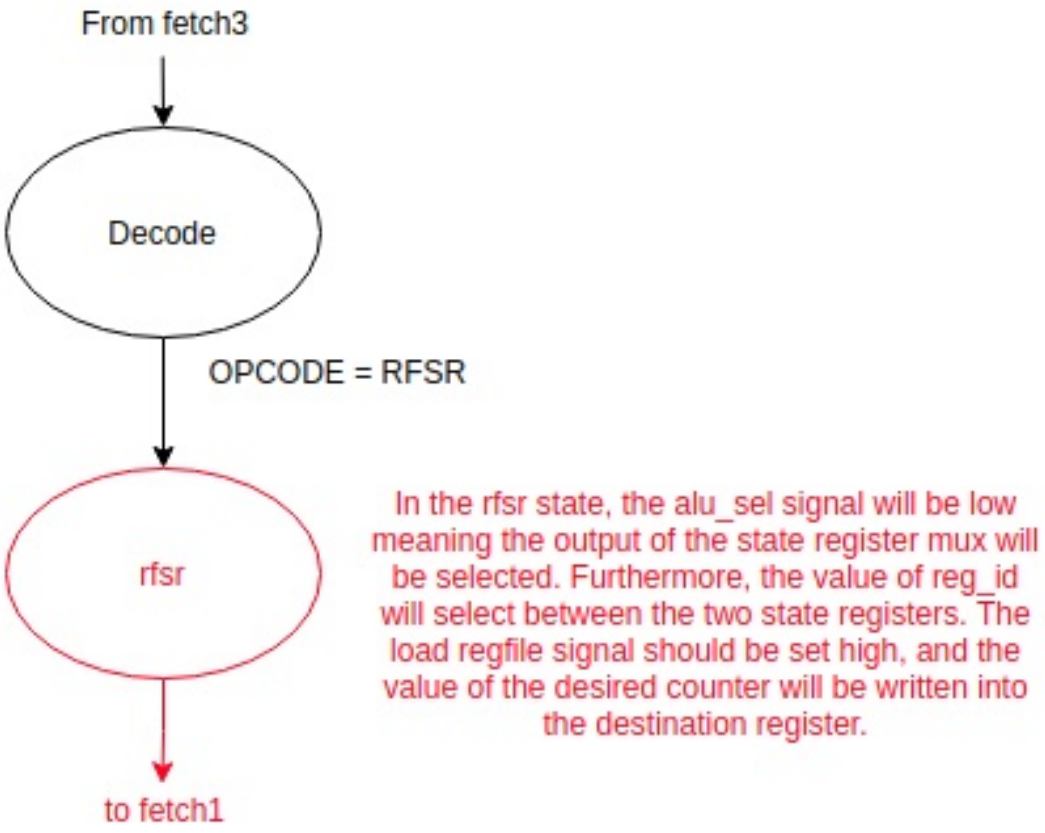
One or two muxes must be added to support the desired functionality.

If two muxes are used, then one will use alu_sel to choose between the ALU output and the "counter mux" output (1 and 0 respectively). The counter mux will use reg_id to select between instrs_exec and mem_cycles (0 and 1 respectively).

If one mux is used, then the two select bits will be reg_id concatenated with alu_sel. The inputs should be labeled and assigned correctly (in any *correct* order).

Finally, we must consider the impact of this instruction on the CPU controller. For this machine, there are three fetch states and a single decode state. These states operate identically to those implemented in MP0 and MP1. Following the decode state, you decide to add new state(s) to handle the *rfsr* instruction.

c. Draw the new state(s) needed to support the the *rfsr* instruction. The decode state has been provided for you. For each state you draw, you should write a sentence or two to describe what is being accomplished. Feel free to use signals to describe the functionality of the state(s), just be sure the signal names match your datapath drawn above. (4 pts)



Name:_____

Following the release of the processor designed in the previous three parts, your team discovers that most users read both of the counters back-to-back. To support the same functionality with fewer instructions, your task is to create a new instruction which reads from both counters. You decide to call it *rfbr* or "read from both registers". This instruction has the following format: *rfbr Rd*. Functionally, the instruction will place the value of *instrs_exec* into Rd and will then place the value of *mem_cycles* into the register **immediately following** Rd. For example, *rfbr x1* should write values into *x1* and *x2*.

d. Summarize the differences between the implementation of the *rfsr* instruction and the *rfbr* instruction. At a high level, your answer should address the changes needed within the CPU datapath and the changes needed within the CPU controller. Feel free to describe the changes in English, with diagrams, or a combination of the two. (4 pts)

In order to support the *rfbr* instruction, the datapath must support the calculation of Rd+1 and have a mechanism to select between Rd and Rd+1 for the destination of the register file. This can be accomplished using a 2-to-1 mux where the inputs are Rd and Rd+1 (from an adder block) with an added select signal.

The select signal will be generated by the new controller which requires a second state be added. The first state will store instrs_exec into Rd while the second state will store mem_cycles into Rd+1. The two states will both have load_regfile = 1, but the states will differ in their destination select signal. The first state should select Rd while the second selects Rd+1. Following the second state, return to fetch1.

Note: Students may find a solution which uses the ALU to calculate Rd+1. This will earn partial credit, but is not ideal since it would require additional hardware which has a greater cost than the above solution, and it would be a poor utilization of the ALU which is much wider than the encoded register -> high power.

e. Processor design is primarily about trade-offs. With your previous answers in mind, write one advantage of the *rfbr* instruction and one disadvantage of the *rfbr* instruction when compared to the *rfsr* instruction. Clearly label your responses.(3 points)

-rfsr has a larger code size and longer execution if both regs read
+ rfbr has a smaller code size and shorter execution if both regs read

+rfsr has a simpler datapath and will execute faster if only 1 reg is read
-rfbr has a more complex datapath and will execute more slowly if only 1 reg is read

Any reasonable response which considered latencies, complexity of the hardware, timing, extra states of execution, memory footprints, instruction cache utilization and so on were accepted. Come see Chance at office hours if you have questions.

Name:_____

Question 6: Potpourri:

a. Consider an ISA with three instructions (Inst1, Inst2, Inst3). Write an instruction sequence with 3 instructions that will have worse performance on a single cycle design compared to multi-cycle design? Why? Assume that inst1, inst2, and inst3 take 1, 2, and 3 cycle respectively on the multi-cycle design. Will this instruction sequence have higher performance on a 3-stage pipeline (vs single-cycle design)? Why? (8 points)

Any sequence besides Inst3; Inst3; Inst3; Worse performance than multi-cycle because any sequence of 3 instructions will take 3 cycles but we know the multi-cycle frequence will be 3 times faster, so as long as the sequence takes less than 9 cycles to be executed by the multi-cycle design, it will be better than the single cycle design.
Will have better performance with pipeline, even though it will take 5 cycles (no matter what the sequence) to go through the pipeline vs 3 cycles to go through the single cycle design, we know the pipeline frequency will be 3 times faster.

b. If D is dynamic power of a CMOS processor design, S is static power of the design, draw the D/S curve showing how it has changed over last 20 years. Make sure to mark 1 on y-axis. Make sure to mark 2018 on x-axis. (6 points)
Should start with D/S >> 1 in 1998 and decrease monotonically, dipping below 1 roughly somewhere between 2008 and 2018.

c. Here's a comparison between STT-RAM, a memory technology based on magnetic spins and SRAM:

	Area (mm^2)	Read Energy (nJ)	Write Energy (nJ)	Leakage Power at (mW)	Read Latency (ns)	Write latency (ns)	Read @ 2 GHz (cycles)	Write @2 GHz (cycles)
1 MB SRAM	2.61	0.578	0.578	4542	1.012	1.012	2	2
4MB STT-RAM	3.00	1.035	1.066	2524	0.998	10.61	2	22

Where would you put STT-RAM in the memory hierarchy? Why? (6 points)
Instruction cache - although the access energy is greater, overall power is lower if you assume an access every 2 cycles at 2 GHz (total power works out to roughly 5W for SRAM vs 3.5W for STT-RAM under this assumption). Even though you never "store" to instruction cache, you still have the 22 cycle write latency any time you load data (think of an instruction cache miss). You can get around this by buffering the data that is being brought in but in the worst case, write throughput could become a bottleneck (frequent instruction cache misses that are

hits in the next level cache, assuming next level cache responds in less than 22 cycles on a hit).

d. If McDonalds is "I am loving it", Skittles is "Taste the Rainbow", Google is "Don't be Evil", ECE411 is "Computer Organization & Design" was the modal answer (2 points)

Name:_____

Appendix A:

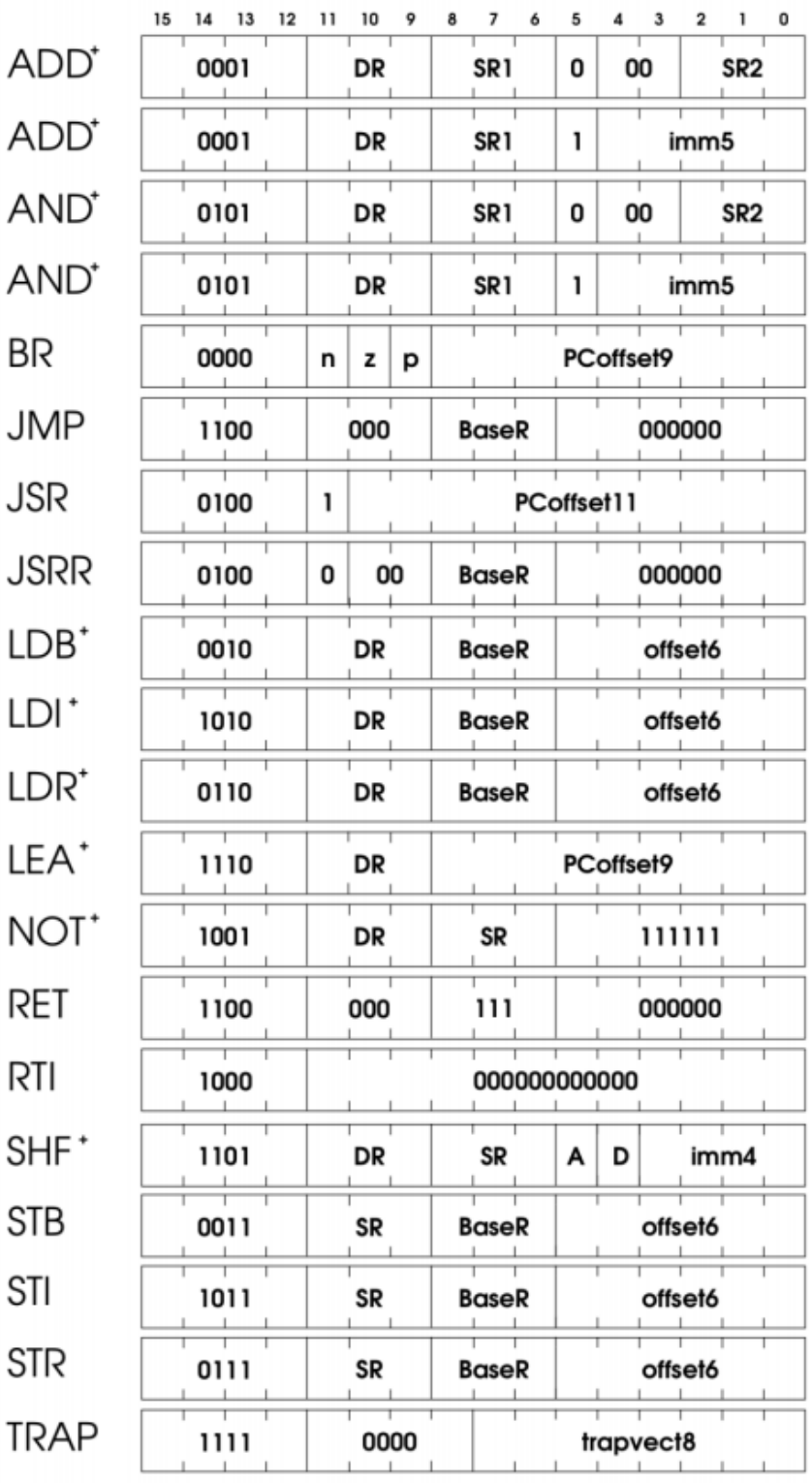


Figure 1.2: LC-3b Instruction Formats. NOTE: + indicates instructions that modify condition codes.

Name:_____

Appendix B:

31 27	26 25	24 20	19 15	14 12	11 7	6 0	
funct7		rs2	rs1	funct3	rd	opcode	R-type
imm[11:0]			rs1	funct3	rd	opcode	I-type
imm[11:5]		rs2	rs1	funct3	imm[4:0]	opcode	S-type
imm[12|10:5]		rs2	rs1	funct3	imm[4:1|11]	opcode	B-type
imm[31:12]					rd	opcode	U-type
imm[20|10:1|11|19:12]					rd	opcode	J-type

RV32I Base Instruction Set

imm[31:12]			rd	0110111	LUI
imm[31:12]			rd	0010111	AUIPC
imm[20|10:1|11|19:12]			rd	1101111	JAL
imm[11:0]	rs1	000	rd	1100111	JALR
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	BEQ
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	BNE
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011	BLT
imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011	BGE
imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011	BLTU
imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011	BGEU
imm[11:0]		rs1	000	rd	0000011	LB
imm[11:0]		rs1	001	rd	0000011	LH
imm[11:0]		rs1	010	rd	0000011	LW
imm[11:0]		rs1	100	rd	0000011	LBU
imm[11:0]		rs1	101	rd	0000011	LHU
imm[11:5]	rs2	rs1	000	imm[4:0]	0100011	SB
imm[11:5]	rs2	rs1	001	imm[4:0]	0100011	SH
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	SW
imm[11:0]		rs1	000	rd	0010011	ADDI
imm[11:0]		rs1	010	rd	0010011	SLTI
imm[11:0]		rs1	011	rd	0010011	SLTIU
imm[11:0]		rs1	100	rd	0010011	XORI
imm[11:0]		rs1	110	rd	0010011	ORI
imm[11:0]		rs1	111	rd	0010011	ANDI
0000000	shamt	rs1	001	rd	0010011	SLLI
0000000	shamt	rs1	101	rd	0010011	SRLI
0100000	shamt	rs1	101	rd	0010011	SRAI
0000000	rs2	rs1	000	rd	0110011	ADD
0100000	rs2	rs1	000	rd	0110011	SUB
0000000	rs2	rs1	001	rd	0110011	SLL
0000000	rs2	rs1	010	rd	0110011	SLT
0000000	rs2	rs1	011	rd	0110011	SLTU
0000000	rs2	rs1	100	rd	0110011	XOR
0000000	rs2	rs1	101	rd	0110011	SRL
0100000	rs2	rs1	101	rd	0110011	SRA
0000000	rs2	rs1	110	rd	0110011	OR
0000000	rs2	rs1	111	rd	0110011	AND