4190.308: Computer Architecture
Final Exam. (Fall 2021)

14:00 – 15:15, December 14, 2021. Instructor: Jin-Soo Kim

Student ID: ______________________________

Name: _______________(sign)____

Nickname: ______________________________
(*Will be used to post your exam score*)

Q1 (50)		Q6 (40)	
Q2 (60)		Q7 (30)	
Q3 (30)		Q8 (30)	
Q4 (30)		Q9 (50)	
Q5 (60)		Total (380)	

1. Fill in the blank with a single English word. (5 points each)

(1) The datapath elements in the RISC-V implementation consist of two different types of logic elements: elements that operate on data values and elements that contain state. The elements that operate on data values are all (   ), which means that their outputs depend only on the current inputs.

(2) An edge-triggered methodology allows a state element to be read and written in the same clock cycle without creating a(an) (   ) that could lead to indeterminate data values.

(3) Pipelining would not decrease the time to complete one load of laundry, but when we have many loads of laundry to do, the improvement in (   ) decreases the total time to complete the work.

(4) Under ideal conditions and with a large number of instructions, the speed-up from pipelining is approximately equal to the number of pipe (   ).

(5) (   ) hardware predictors, in stark contrast, make their guesses depending on the behavior of each conditional branch and may change predictions for a conditional branch over the life of a program.

(6) The (   ) determines the number of source program instructions executed and hence the number of processor instructions executed. It may also affect the CPI, by favoring slower or faster instructions.

(7) Although personal mobile devices like the iPad use individual DRAMs. memory for servers is commonly sold on small boards called dual (   ) memory modules (DIMMs).

(8) (   ) misses are cache misses that occur in set-associative or direct-mapped caches when multiple blocks compete for the same set.

(9) A virtual memory block is called a page, and a virtual memory miss is called a page (   ).

(10) Modern processors include a special cache that keeps track of recently used translations. This special address translation cache is traditionally referred to as a translation-(   ) buffer (TLB), although it would be more accurate to call it a translation cache.

RISC-V RV32I/64I Integer Instructions

MNEMONIC	FMT	NAME	DESCRIPTION (in Verilog)
`add,addw`	R	ADD (Word)	R[rd] = R[rs1] + R[rs2]
`addi,addiw`	I	ADD Immediate (Word)	R[rd] = R[rs1] + imm
`and`	R	AND	R[rd] = R[rs1] & R[rs2]
`andi`	I	AND Immediate	R[rd] = R[rs1] & imm
`auipc`	U	Add Upper Immediate to PC	R[rd] = PC + {imm, 12'b0}
`beq`	SB	Branch EQual	if(R[rs1]==R[rs2) PC=PC+{imm,1b'0}
`bge`	SB	Branch Greater than or Equal	if(R[rs1]>=R[rs2) PC=PC+{imm,1b'0}
`bgeu`	SB	Branch ≥ Unsigned	if(R[rs1]>=R[rs2) PC=PC+{imm,1b'0}
`blt`	SB	Branch Less Than	if(R[rs1]<R[rs2) PC=PC+{imm,1b'0}
`bltu`	SB	Branch Less Than Unsigned	if(R[rs1]<R[rs2) PC=PC+{imm,1b'0}
`bne`	SB	Branch Not Equal	if(R[rs1]!=R[rs2) PC=PC+{imm,1b'0}
`csrrc`	I	Cont./Stat.RegRead&Clear	R[rd] = CSR;CSR = CSR & ~R[rs1]
`csrrci`	I	Cont./Stat.RegRead&Clear Imm	R[rd] = CSR;CSR = CSR & ~imm
`csrrs`	I	Cont./Stat.RegRead&Set	R[rd] = CSR; CSR = CSR | R[rs1]
`csrrsi`	I	Cont./Stat.RegRead&Set Imm	R[rd] = CSR; CSR = CSR | imm
`csrrw`	I	Cont./Stat.RegRead&Write	R[rd] = CSR; CSR = R[rs1]
`csrrwi`	I	Cont./Stat.Reg Read&Write Imm	R[rd] = CSR; CSR = imm
`ebreak`	I	Environment BREAK	Transfer control to debugger
`ecall`	I	Environment CALL	Transfer control to operating system
`fence`	I	Synch thread	Synchronizes threads
`fence.i`	I	Synch Instr & Data	Synchronizes writes to instruction stream
`jal`	UJ	Jump & Link	R[rd] = PC+4; PC = PC + {imm,1b'0}
`jalr`	I	Jump & Link Register	R[rd] = PC+4; PC = R[rs1]+imm
`lb`	I	Load Byte	R[rd] = {56'bM[](7),M[R[rs1]+imm](7:0)}
`lbu`	I	Load Byte Unsigned	R[rd] = {56'b0,M[R[rs1]+imm](7:0)}
`ld`	I	Load Doubleword	R[rd] = M[R[rs1]+imm](63:0)
`lh`	I	Load Halfword	R[rd] = {48'bM[](15),M[R[rs1]+imm](15:0)}
`lhu`	I	Load Halfword Unsigned	R[rd] = {48'b0,M[R[rs1]+imm](15:0)}
`lui`	U	Load Upper Immediate	R[rd] = {32b'imm<31>, imm, 12'b0}
`lw`	I	Load Word	R[rd] = {32'bM[](31),M[R[rs1]+imm](31:0)}
`lwu`	I	Load Word Unsigned	R[rd] = {32'b0,M[R[rs1]+imm](31:0)}
`or`	R	OR	R[rd] = R[rs1] | R[rs2]
`ori`	I	OR Immediate	R[rd] = R[rs1] | imm
`sb`	S	Store Byte	M[R[rs1]+imm](7:0) = R[rs2](7:0)
`sd`	S	Store Doubleword	M[R[rs1]+imm](63:0) = R[rs2](63:0)
`sh`	S	Store Halfword	M[R[rs1]+imm](15:0) = R[rs2](15:0)
`sll,sllw`	R	Shift Left (Word)	R[rd] = R[rs1] << R[rs2]
`slli,slliw`	I	Shift Left Immediate (Word)	R[rd] = R[rs1] << imm
`slt`	R	Set Less Than	R[rd] = (R[rs1] < R[rs2]) ? 1 : 0
`slti`	I	Set Less Than Immediate	R[rd] = (R[rs1] < imm) ? 1 : 0
`sltiu`	I	Set < Immediate Unsigned	R[rd] = (R[rs1] < imm) ? 1 : 0
`sltu`	R	Set Less Than Unsigned	R[rd] = (R[rs1] < R[rs2]) ? 1 : 0
`sra,sraw`	R	Shift Right Arithmetic (Word)	R[rd] = R[rs1] >> R[rs2]
`srai,sraiw`	I	Shift Right Arith Imm (Word)	R[rd] = R[rs1] >> imm
`srl,srlw`	R	Shift Right (Word)	R[rd] = R[rs1] >> R[rs2]
`srli,srliw`	I	Shift Right Immediate (Word)	R[rd] = R[rs1] >> imm
`sub,subw`	R	SUBtract (Word)	R[rd] = R[rs1] – R[rs2]
`sw`	S	Store Word	M[R[rs1]+imm](31:0) = R[rs2](31:0)
`xor`	R	XOR	R[rd] = R[rs1] ^ R[rs2]
`xori`	I	XOR Immediate	R[rd] = R[rs1] ^ imm

(For problems Q2 – Q4)

We consider the following two pipelined implementations of the RISC-V processor. (Note that the stage names are simplified.)

Processor name	Mechanisms for data hazards	Mechanisms for control hazards	Remarks
SNUCOM-I 5-stage: F(IF)-D(ID)-E(EX)-M(MM)-W(WB)	• Data forwarding is fully implemented. • A register can NOT be read and written in the same cycle.	• Branch outcome and the target address are calculated in E stage. • Always-not-taken branch prediction is used. • **`jal`** and **`jalr`** instructions are treated as mispredicted branch.	This processor behaves the same as the **snurisc5** processor in Project #4.
SNUCOM-II 6-stage: F(IF)-D(ID)-E(EX)-1(M1)-2(M2)-W(WB)	• Data forwarding is fully implemented. • A register can NOT be read and written in the same cycle.	• Branch outcome and the target address are calculated in E stage. • BTFNT (Backward branch as Taken, Forward branch as Not Taken) prediction scheme is used for conditional branch instructions.	This processor behaves the same as the **snurisc6** processor in Project #4 <u>except that return address stack is not used.</u>

In the following questions Q3 – Q4, you need to simulate the execution until the function is returned by the `ret` instruction. Assume that there is no cache miss during the execution. You should minimize the number of stalled cycles. For any cancelled or bubbled stage, mark it with " — ".

How to mark the table: *For each cycle, specify the instruction (S1 ~ S13 in Q2) fetched in that cycle in the leftmost column. If the entry (i, c) is marked with 'X', it represents that the instruction i has executed the 'X' stage of the pipeline in the cycle c. If an instruction is stalled for 1 cycle, then write the name of the stalled stage twice for current and next cycle. In a given cycle, the same stage should NOT appear more than once.*

2. Consider the following C program and its corresponding RISC-V assembly code.

```
// Assume that a sufficiently
// large amount of memory is
// already allocated at address A


int f(int *A, int N, int S)
{
    int i;
    int z = 0;

    for (i = 0; i < N; i += S)
        z = z + A[i];

    return z;
}
```

```
      f:
S1:       bge    zero, a1, L1
S2:       addi   a5, a0, 0
S3:       slli   a6, a2, 2
S4:       addi   a0, zero, 0
S5:       addi   a4, zero, 0
      L2:
S6:       lw     a3, 0(a5)
S7:       add    a0, a0, a3
S8:       add    a4, a4, a2
S9:       add    a5, a5, a6
S10:      blt    a4, a1, L2
S11:      ret
      L1:
S12:      addi   a0, zero, 0
S13:      ret
```

(1) When we call the function `f()` with `N` = 100 and `S` = 1, how many instructions are executed (from the `bge` instruction at **`S1`** to the `ret` instruction)? Do not include cancelled instructions, if any. (10 points)

(2) If we run the function with `N` = 100 and `S` = 1 on the SNUCOM-I processor (i.e., **snurisc5**), how many cycles are needed? The number of cycles is counted from when the `bge` instruction at **`S1`** is fetched in the F stage until the last `ret` instruction is completed in the W stage. Assume that instruction/data memory can be accessed in a single cycle. (20 points)

(3) If we run the function with `N` = 100 and `S` = 1 on the SNUCOM-II processor (i.e., **snurisc6**), how many cycles are needed? Assume that instruction/data memory can be accessed in a single cycle. (20 points)

(4) The SNUCOM-I processor runs at 2GHz and the SNUCOM-II processor runs at 2.5GHz. For the given input (`N` = 100 and `S` = 1) , which one is faster to run the function `f()` and how much? (10 points)

3. Complete the pipeline diagram below (instructions on the left, cycles on the top) when the function `f()` in Q2 is executed on the SNUCOM-I processor with `N` = 100 and `S` = 1. You can draw the diagram for the first 20 instructions fetched by the processor. Please refer to the bottom of Page 2 on how to represent stalls and bubbles. Assume that instruction/data memory can be accessed in a single cycle. (30 points)

Inst.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
S1	F	D	E	M	W																																										
		F																																													

4. Complete the pipeline diagram below (instructions on the left, cycles on the top) when the function `f()` in Q2 is executed on the SNUCOM-II processor with `N` = 100 and `S` = 1. You can draw the diagram for the first 20 instructions fetched by the processor. Please refer to the bottom of Page 2 on how to represent stalls and bubbles. Assume that instruction/data memory can be accessed in a single cycle. (30 points)

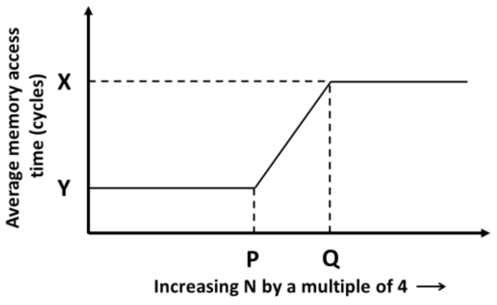
Inst.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
S1	F	D	E	1	2	W																																									
		F																																													

5. Suppose the SNUCOM-I processor now has a data cache, where data can be accessed within a single cycle on hits (i.e., the hit time is 1 cycle). On misses, however, the SNUCOM-I processor simply stalls the pipeline waiting for the data. Unless otherwise stated, assume the following data cache configuration. (60 points)

Cache size (the amount of data that can be cached excluding valid bits and tag bits)	4096 bytes
Cache block size	16 bytes
Cache associativity	Direct-mapped
Cache hit time (time to fetch data from the cache to the CPU)	1 cycle
Cache miss penalty (time to fetch a cache block from the memory to the cache)	100 cycles

We run the function `f()` in Q2 on the SNUCOM-I processor equipped with the data cache, with increasing the value of `N` by a multiple of 4 (i.e., `N` = 4, 8, 12, ...). The value of `S` is fixed to 1. The graph below shows the change in the average memory access time. The x-axis indicates the value of `N` and the y-axis represents the average time to access an element in the array `A`. Determine the following values with the configuration parameters shown in the above table. Please note the following:

- The only memory accesses are to the elements of the array `A`.
- The array `A` starts at memory address `0x80010000`.
- To warm up the cache, <u>we run the function `f()` with the same arguments twice, and then measure the cache misses and the average memory access time only for the second run of the function `f()`</u>.
- The physical address is 32 bits long (no virtual memory supported)



(1) The total number of cache blocks that can be cached: ________________________ (blocks)

(2) The total number of cache sets: ________________________

(3) The number bits for each tag: ________________________ (bits)

(4) **P** = ________________________ (should be a multiple of 4)

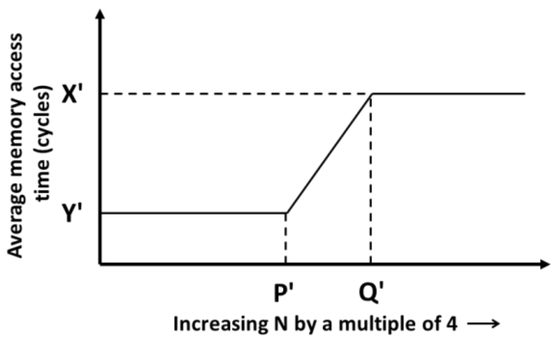
(5) **Y** = ________________________ (cycles)

(6) **Q** = ________________________ (should be a multiple of 4)

(7) **X** = ________________________ (cycles)

(8) The miss rate when **N** > **Q** (in the range of second plateau): ________________________

6. After successfully completing the Computer Architecture course, a group of students at SNU upgraded the data cache of the SNUCOM-I processor to a 4-way set-associative cache with LRU replacement policy. Now determine the new values of the following due to this upgrade when we run the function `f()`. Again, we vary the value of `N` by a multiple of 4 (i.e., `N` = 4, 8, 12, ...) with fixing the value of `S` to 1. Assume that the rest of the cache parameters and the evaluation methodology are same as in Q5. (*Note: The cache has been warmed up by running the same function before the measurement.*) (40 points)



(1) The total number of cache sets:

____

(2) The number bits for each tag:

________________________ (bits)

(3) **P'** = ________________________ (should be a multiple of 4)

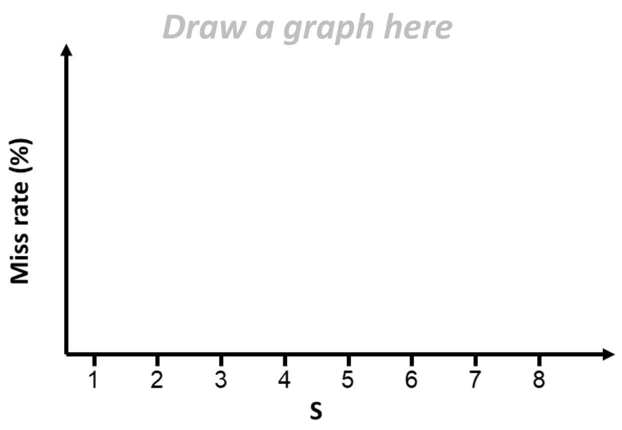
(4) **Y'** = ________________________ (cycles)

(5) **Q'** = ________________________ (should be a multiple of 4)

(6) **X'** = ________________________ (cycles)

7. To examine the relationship between `S` and <u>the miss rate</u>, we fix the value of **`N`** to 2048 and vary the value of `S` from 1 to 8 under the following cache configuration. Complete the graph shown below. Also, give the miss rates for `S` = 1, 2, 4, and 8. (*Note: The cache has been warmed up by running the same function before the measurement.*) (30 points)

Cache size	2048 bytes
Cache block size	16 bytes
Cache associativity	Fully-associative with LRU replacement policy
The value of `N`	2048



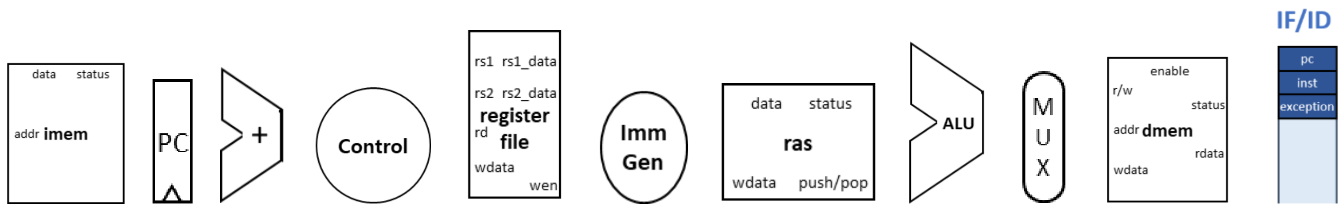
(1) Miss rate for `S` = 1:

(2) Miss rate for `S` = 2:

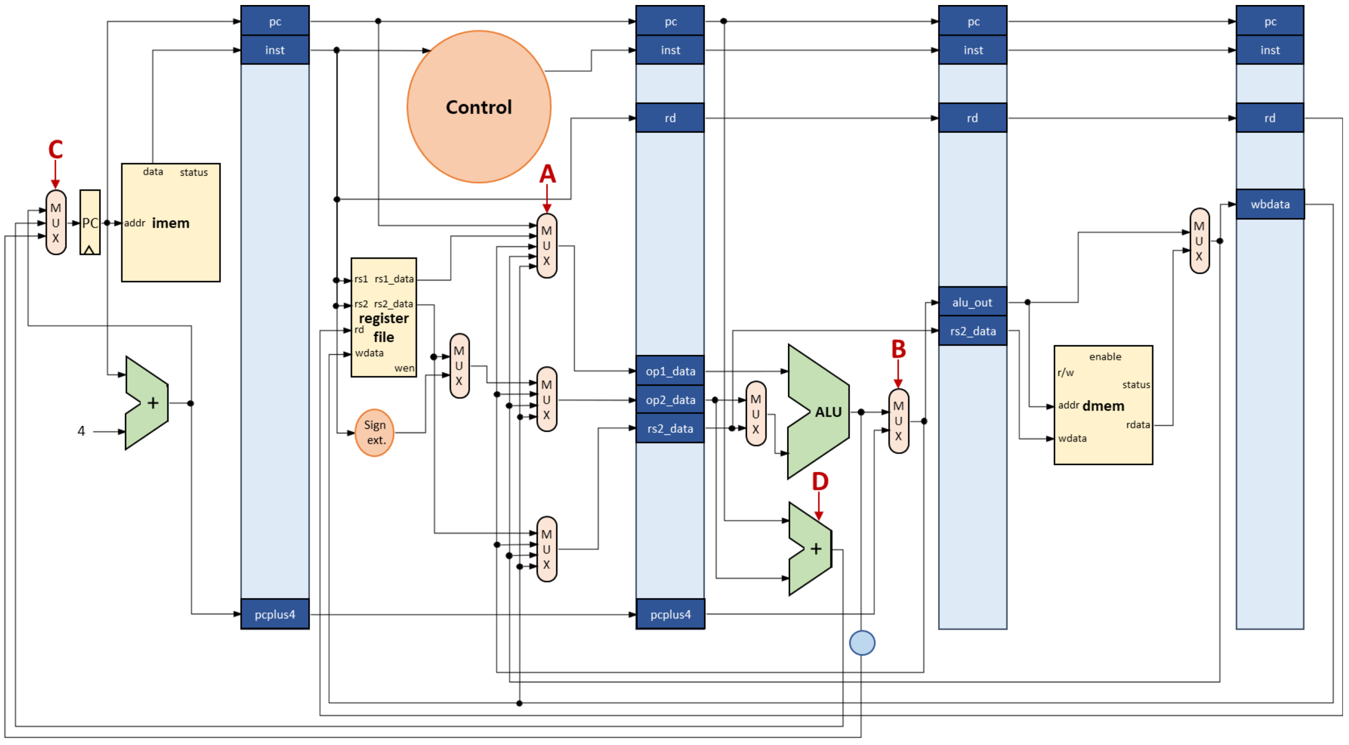
(3) Miss rate for `S` = 4:

(4) Miss rate for `S` = 8:

8. The following shows various components that are needed for implementing the 6-stage pipelined processor in Project #4. Among these components, which are sequential elements? Circle all the sequential elements. (*Right answer = n points, Wrong answer = -n points*) (30 points)



9. The following figure shows the datapath design of the SNUCOM-I (i.e., **snurisc5**) processor.



(1) When does the MUX **A** pass the first input (i.e., `pc`) to the output? For which instructions? (10 points)

(2) When does the MUX **A** pass the third input (i.e., the output of the MUX **B**) to the output? For which instructions? (10 points)

(3) When does the MUX **B** pass the second input (i.e., `pcplus4`) to the output? For which instructions? (10 points)

(4) When does the MUX **C** pass the third input (i.e., the output of ALU) to the output? For which instructions? (10 points)

(5) When is the adder **D** used? For which instructions? (10 points)