서 울 대 학 교
SEOUL NATIONAL UNIVERSITY

Systems Software & Architecture Laboratory
Dept. of Computer Science and Engineering

4190.308: Computer Architecture
Final Exam. (Fall 2020)

11:00 – 12:15, December 14, 2020. Instructor: Jin-Soo Kim

Student ID: _____

Name: _____(sign)____

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

Q0 (5)		Q4 (50)	
Q1 (30)		Q5 (10)	
Q2 (30)		Q6 (40)	
Q3 (55)		Q7 (40)	

0. Which of the following statement is true (i.e., not a fallacy)? (5 points)

A. Computers at low utilization use little power.

B. Designing for performance and designing for energy efficiency are related goals.

C. More powerful instructions mean higher performance.

D. The importance of commercial binary compatibility means successful instruction sets don't change.

E. Pipelining ideas can be implemented independent of technology.

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

(1) Datacenter managers often care about increasing ( ) or bandwidth – the total amount of work done in a given time.

(2) The efficiency of the ( ) affects both the instruction count and average cycles per instruction, since it determines the translation of the source language instructions into computer instructions.

(3) Logic components that contain state are also called ( ), because their outputs depend on both their inputs and the contents of the internal state.

(4) One approach is to lookup the address of the instruction to see if the conditional branch was taken the last time this instruction was executed, and, if so, to begin fetching new instructions from the same place as the last time. This technique is called ( ) branch prediction.

(5) ( ) miss is a cache miss that occurs because the cache, even with full associativity, cannot contain all the blocks needed to satisfy the request.

(6) The segment of the stack containing a procedure's saved registers and local variables is called a procedure ( ) or activation record.

2. The following table shows the instruction type breakdown of an application (program A) executed on a non-pipelined 1GHz processor. Answer the following questions.

Program	The number of instructions executed			CPI		
	Arithmetic	Load/Store	Branch	Arithmetic	Load/Store	Branch
A	4000	4000	2000	1	3	2

(1) What is the average CPI for the instruction mix of program A? (5 points)

CPI: _____

(2) What is the total CPU time to execute the program A? (5 points)

Total CPU time: _____ (μs)

(3) We have obtained a new program B by compiling the same application with an optimizing compiler. The number of instructions executed has been changed as shown in the right table. How much is the program B faster than the original program A? (5 points)

Program	The number of instructions executed		
	Arithmetic	Load/Store	Branch
B	5000	3000	1000

Program B is _____ times faster than Program A.

(4) The next-generation CPU operates at 2GHz, but it has increased the CPI as shown in the right table. How much does the program B run faster than the program A on the new 2GHz non-pipelined processor? (5 points)

CPI (at 2GHz)		
Arithmetic	Load/Store	Branch
1	4	3

Program B runs _____ times faster than Program A on the new 2GHz processor.

(5) We are about to design a new processor which has the faster clock frequency than 2GHz. However, we find that the CPI of arithmetic operations should be increased from 1 to 2 to increase the clock frequency beyond 2GHz as shown in the right table. What should be the clock frequency of the new processor to double the performance of program B compared to the case with 2GHz processor? (10 points)

CPI (> 2GHz)		
Arithmetic	Load/Store	Branch
2	4	3

Program B runs two times faster on _____GHz processor than Program B on 2GHz processor.

3. The followings show both the C code and the generated 64-bit RISC-V assembly code for the recursive algorithm to find the Fibonacci number, $fib(n) = fib(n-1) + fib(n-2)$ for $n \geq 2$ where $fib(0) = fib(1) = 1$. Answer the following questions.

```
long fib(long n)
{
  if (n <= 1)
    return 1;
  else
    return fib(n-1) + fib(n-2);
}
```

```
0000000080000000 <_start>:
    80000000:   80020137    lui     sp, 0x80020
    80000004:   00300513    addi    a0, x0, 4
    80000008:   008000ef    call    fib
    8000000c:   00100073    ebreak      # terminate program

0000000080000010 <fib>:
    80000010:   00100793    addi    a5, x0, 1
    80000014:   00a7c663    blt     a5, a0, L1 <80000020>
    80000018:   00100513    addi    a0, x0, 1
    8000001c:   00008067    ret
L1: 80000020:   fe810113    addi    sp, sp, -24
    80000024:   00113823    sd      ra, 16(sp)
    80000028:   00813423    sd      s0, 8(sp)
    8000002c:   00913023    sd      s1, 0(sp)
    80000030:   00050413    addi    s0, a0, 0
    80000034:   fff50513    addi    a0, a0, -1
    80000038:   fd9ff0ef    call    fib
    8000003c:   00050493    addi    s1, a0, 0
    80000040:   ffe40513    addi    a0, s0, -2
    80000044:   fcdff0ef    call    fib
    80000048:   00a48533    add     a0, s1, a0
    8000004c:   01013083    ld      ra, 16(sp)
    80000050:   00813403    ld      s0, 8(sp)
    80000054:   00013483    ld      s1, 0(sp)
    80000058:   01810113    addi    sp, sp, 24
    8000005c:   00008067    ret
```

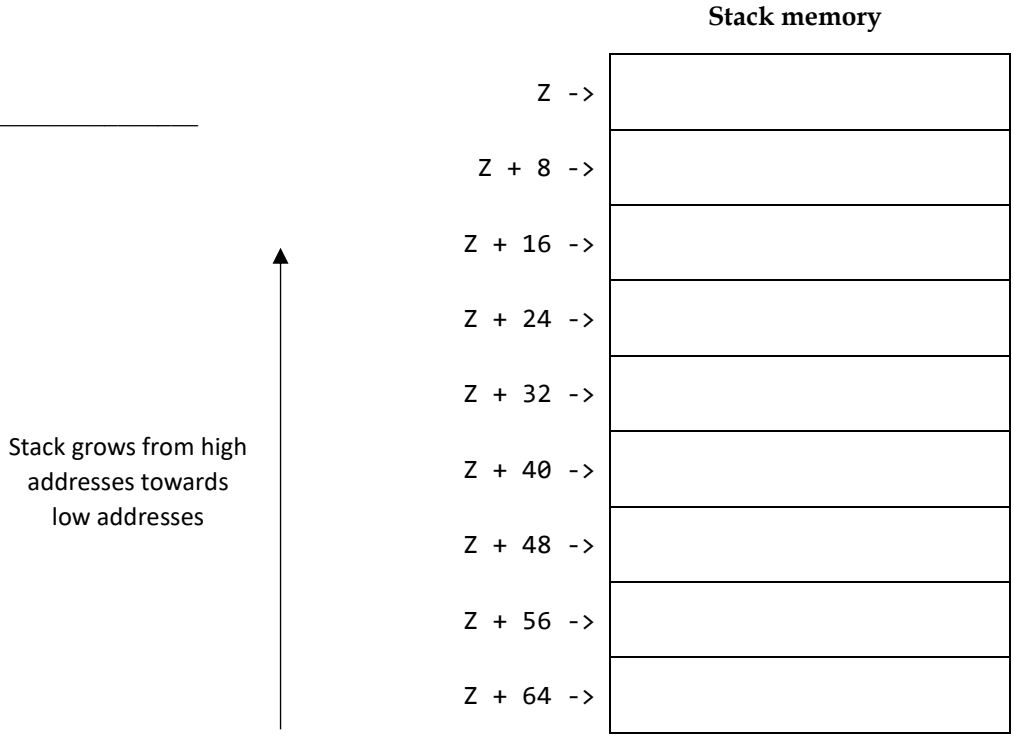
(1) How many times the instruction at the memory address `0x80000010` is executed? (5 points)

(2) How many times the instruction at the memory address `0x80000018` is executed? (5 points)

(3) How many times the instruction at the memory address `0x80000058` is executed? (5 points)

(4) Consider a situation when the control reaches the address `0x80000018` for the first time as a result of the call to the function `fib()` with the argument `4`. Let us assume that the value of the **`sp`** register when the CPU is about to execute the **`addi`** instruction at `0x80000018` is Z. What is the value of Z? Also, show the contents of the following memory locations. Assume that all the memory contents and registers are initialized to zero before running the program. Also note that the **`sp`** register was initialized to `0x80020000` at the beginning of the program. Write "UNKNOWN" if you don't have enough information for the value. (40 points)

Value of Z: __0x_____



RISC-V RV64I 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

4. Suppose we have a 32-bit RISC-V processor with the following L1 data cache configuration. Memory address is 32-bit long. Answer the following questions.

Cache size (the amount of data that can be cached excluding valid bits and tag bits)	512 bytes
Cache block size	16 bytes
Cache associativity	4-way set associative
Cache replacement policy	LRU
Write policy	Write-back/Write-allocate

(1) What is the total number of sets in the cache? (5 points)

_____ sets

(2) What is the size of the tag field? (5 points)

_____ bits

(3) We run the C code shown in the right on this processor. Fill in the table with the total number of load/store instructions executed and the total number of cache misses when we call the function `f()` with the matrices **A**, **B**, and **C** varying the value of *n* to 1, 2, 4, and 8. (40 points)

```
int f(int a[8][8], int b[8][8], int c[8][8], int n)
{
    int i, j, k, r;

    for (i = 0; i < n; i++) {
        for (k = 0; k < n; k++) {
            r = a[i][k];
            for (j = 0; j < n; j++)
                c[i][j] += r + b[k][j];
        }
    }
}
```

Assume the followings:

(a) The only memory accesses are to the array **A**, **B**, and **C**.

(b) **A**, **B**, and **C** are all 8 x 8 matrices of integers.

(c) Array **A** starts at memory address `0x80010000`.

(d) Array **B** starts at memory address `0x80014000`.

(e) Array **C** starts at memory address `0x80018000`.

(f) In the inner loop, **b[][]** is read before **c[][]**.

(g) Cache is initially empty.

(h) Virtual memory is not supported.

(i) `sizeof(int) == 4`

	Total number of load/store instructions executed	Total number of cache misses
`f(A,B,C,1)`		
`f(A,B,C,2)`		
`f(A,B,C,4)`		
`f(A,B,C,8)`		

5. Consider the following RISC-V assembly program. Assume that the CPU terminates the program when it meets the `ebreak` instruction.

(1) What is the value of the `a0` register when this program is terminated? (5 points)

(2) How many instructions are executed before the program is terminated (including the `ebreak` instruction)? (5 points)

```
   _start:
S1:     addi    a0, x0, 24
S2:     addi    a1, x0, 6
S3:     jal     ra, g
S4:     ebreak
   g:
S5:     beq     a1, a0, L10
   L3:
S6:     bge     a1, a0, L4
S7:     sub     a0, a0, a1
S8:     bne     a1, a0, L3
S9:     jalr    x0, 0(ra)
   L10:
S10:    jalr    x0, 0(ra)
   L4:
S11:    sub     a1, a1, a0
S12:    bne     a1, a0, L3
S13:    jal     x0, L10
```

(For problems 6 – 7)

We consider the following 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. • The same 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 (including **`jal`** and **`jalr`**)	This processor behaves the same as the **snurisc5** processor in Project #4.
SNUCOM-II 6-stage: F(IF)-D(ID)-R(RR)-E(EX)-M(MM)-W(WB)	• Data forwarding is fully implemented. • The same register can NOT be read and written in the same cycle.	• Branch outcome and the target address are calculated in E stage. • **Always-taken** branch prediction is used (including **`jal`** instruction) • **`jalr`** instruction is treated as in SNUCOM-I.	This processor behaves the same as the **snurisc6** processor in Project #4.

In the following questions 6 and 7, you need to simulate the execution until the program is terminated by the `ebreak` 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 the above box) 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.*

Example:
```
S6:   F D D E M W      # instruction S6 is stalled for one cycles in the D stage
S6:   F - - - -        # instruction S6 turns into bubble after the F stage
```

6. Complete the pipeline diagram below (instructions on the left, cycles on the top) when the program in Q5 is executed on the SNUCOM-I processor. (40 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																																										
S2		F																																													

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

7. Complete the pipeline diagram below (instructions on the left, cycles on the top) when the program in Q5 is executed on the SNUCOM-II processor. (40 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	R	E	M	W																																									
S2		F																																													

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