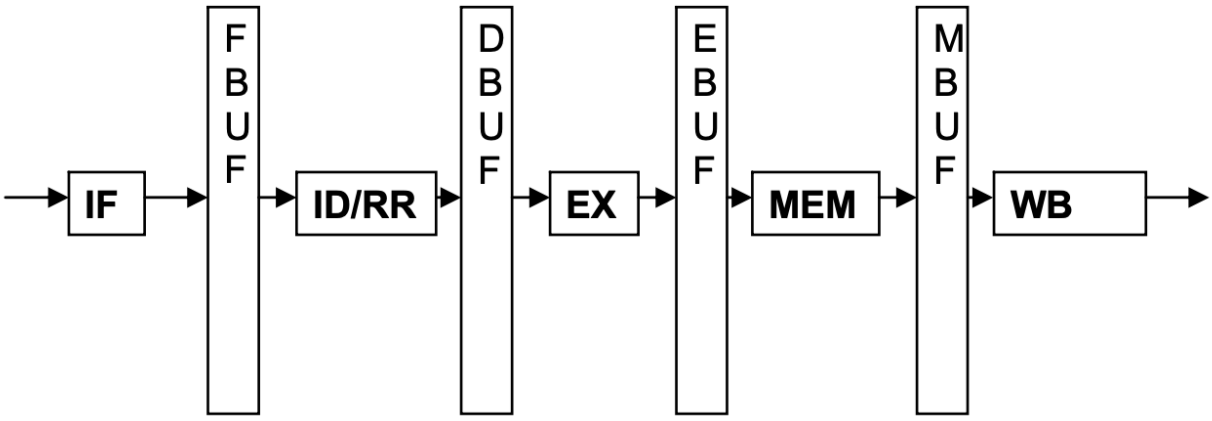
CS 2200 Spr24 Exam 2: Released

Question #	Topic	Total
1	Pipeline buffer contents	8
2	Data hazards	10
3	Pipeline performance + branch prediction	18
4	Scheduling timeline	11
5	Round Robin Waiting Time	13
6	Virtual memory	10
Total		70

Note: Released questions will not change qualitatively, but can change quantitatively. For example, numbers, instructions, order of events, etc. may be different in the exam. **In the actual exam, any such changes will be bolded.** The exam uses the LC-2200 instruction set, with the addition of the LEA and BGT instructions.

Question 1 (8 points)

Consider the following five stage pipeline for the LC-2200 ISA.



List the minimum buffer contents needed to execute the following two instructions in our LC-2200 pipeline ISA. Refer to the Appendix for instruction formats. Interrupts must be supported. (1 pt for each buffer). Please ensure that your answer matches the following notations.

- Notation to denote a 32 bit instruction: Instruction
- Notation to denote Opcode: Opcode
- Notation to denote register specifiers: Rx, Ry, Rz
- Notation to denote contents of registers: [Rx], [Ry], [Rz]
- Notation to denote sign-extended offset: Offset
- Notation to denote ALU output: ALU-result
- Notation to denote PC: PC

Q1.1 (4 points)

List the minimum buffer contents needed to execute the SW Instruction.

```
SW    Rx, offset20(Ry)
```

Encoding

31 30 29 28	27 26 25 24	23 22 21 20	19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0100	Rx	Ry	offset20

Operation

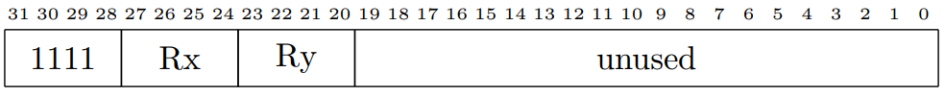
```
MEM[Ry + SEXT(offset20)] = Rx;
```

Q1.2 (4 points)

List the minimum buffer contents needed to execute the CAP Instruction. The CAP instruction performs the logical NOT of the number in the source register Ry and stores the result in destination register Rx. Hint: Logical NAND of a number by itself is equivalent to the logical NOT of the number.

```
CAP Rx, Ry
```

Encoding



Operation

```
Rx = ~Ry
```

Question 2 (10 points)

Q2.1 (8 points)

Consider the following snippet of LC-2200 assembly code. List the different types of data dependencies (RAW, WAR, WAW) that exist between every pair of instructions in the following listing. If there is no applicable data hazard, enter “N/A.”

```
i1: ADDI $s0, $s1, 20
i2: LW   $t0, 3($s0)
i3: SW   $t0, 1($t1)
i4: NAND $s1, $t0, $s2
```

Example answer format:
RAW: i5-i6, i10-i12
WAR: N/A
WAW: i7-i8

Q2.2 (2 points)

Given the following instruction i1, write an instruction i2 that directly follows i1 and has both a WAR and a RAW data dependency with i1.

```
i1: ADD $s0, $t1, $t2
```

Question 3 (18 points)

Q3.1 (8 points)

Consider the following snippet of assembly code. Assume that the initial values for registers $t0 and $t1 are $t0 = 3 and $t1 = 6. Note that the code uses the BNE instruction (branch if not equal to). You can assume that ADDI works the same as it does in the LC-2200 ISA.

```
1 loop:
2 addi $t0, $t0, -1        ! Decrement $t0 ($t0 = $t0 - 1)
3 addi $t1, $t1, -1        ! Increment $t1 ($t1 = $t1 + 1)
4 addi $v0, $v0, 1         ! Increment $v0 by 1
5 bne  $t0, $t1, loop      ! Branch to end if $t0 not equal to $t1 (Not taken)
6 bne  $t0, $zero, loop    ! Branch to loop if $t0 not equal to 0
7 addi $t1, $zero, 15      ! $t1 = $zero + 15
```

Assume these instructions are executed on the LC-2200's five-stage pipeline, but with register forwarding available from all stages following the ID/RR stage. The pipeline is not equipped with any form of branch prediction, and the ISA does not support branch delay slots.

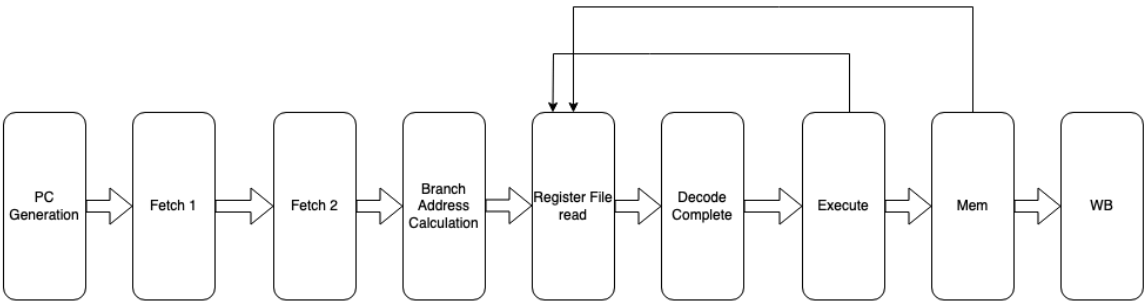
Calculate the Cycles per Instruction (CPI) achieved with conservative branch handling for the entire program's execution. You may assume that the pipeline is empty (i.e., all five stages are NOP) at the very start.

Q3.2 (10 points)

Consider the following snippet of assembly code.

```
1 add  $v0, $zero, $zero   ! Initialize $v0 to 0, $v0 =$zero + $zero
2 addi $t0, $zero, 2       ! Initialize $t0 to 2, $t0 = $zero +2
3 addi $t0, $t0, -1        ! Decrement $t0, $t0 = $t0 - 1
4 addi $v0, $t1, 1         ! Perform: $v0 = $t1 + 1
5 addi $t1, $zero, 15      ! Perform: $t1 = $zero + 15
```

Assume the following snippet is executed using the following pipeline:



The above 9-stage pipeline has data forwarding only from the “MEM” and “Execute” stages to the “Register File read” stage. Reference this pipeline structure for the following two questions. Assume that arithmetic for ADD/ADDI instruction happens in the “Execute” stage of the pipeline.

Q3.2.a (6 points)

Considering the above code snippet and the 9-stage pipeline, list EACH pair of instructions that will result in a data hazard; and for each such pair state the number of pipeline bubbles that will occur.

For each hazard type, example format: (I1,I2):2,(I1,I3):1 for 2 bubbles between I1 and I2 and 1 bubble between I1 and I3.. If there is no applicable data hazard, enter “N/A.” If applicable, clearly state any assumptions for this question in the assumptions box at the end of the exam.

a) List all the RAW hazard pairs and bubbles.

b) List all the WAR hazard pairs and bubbles.

c) List all the WAW hazard pairs and bubbles.

Q3.2.b (4 points)

Could you improve the pipeline’s performance for the given set of instructions, by only adding extra forwarding paths in the pipeline? Please explain.

Question 4 (11 points)

Consider the following three processes P1, P2, and P3:

	P1	P2	P3
CPU Burst Time	4	2	1
I/O Burst Time	2	1	2

Assume each process first completes a CPU burst, followed by I/O burst, and then another CPU burst (of the same length as the first CPU burst) before terminating. Assume no preemption for both parts.

Q4.1 (3 points)

Assume that the scheduler uses the First-Come, First-Served (FCFS) scheduling algorithm. You can assume that all processes arrive at time 0, but the scheduler decides the arrival order to be P1, followed by P2, followed by P3. (0.5 points for each question answered correctly).

a) At what time does P1 start its first CPU burst?

b) At what time does P1 finish?

c) At what time does P2 start its first CPU burst?

d) At what time does P2 finish?

e) At what time does P3 start its first CPU burst?

f) At what time does P3 finish?

Q4.2 (3 points)

Assume that the scheduler now uses the Shortest Job First (SJF) scheduling algorithm. You can assume that all processes arrive at time 0. (0.5 points for each question answered correctly).

a) At what time does P1 start its first CPU burst?

b) At what time does P1 finish?

c) At what time does P2 start its first CPU burst?

d) At what time does P2 finish?

e) At what time does P3 start its first CPU burst?

f) At what time does P3 finish?

Q4.3 (2 points)

When the scheduler uses the FCFS scheduling algorithm, what is the average turnaround time? Please show work.

Q4.4 (3 points)

When the scheduler uses the FCFS scheduling algorithm, what is the waiting time for each process? Please show work.

Question 5 (13 points)

Harper is implementing a round robin scheduler. Each process gets a full time quantum on the processor when it is scheduled. If a process gives up the CPU to do I/O before its time quantum is finished, then the time quantum is reset for the next scheduled process.

The ready queue contains four processes: P1, P2, P3, P4 (which arrive in that order). The execution characteristics of the four processes are as shown below. Each process begins with a CPU burst, followed by an I/O burst, and then one final CPU burst before finishing. The length of each burst is also given in time units (note that the diagrams are not to scale).

P1 (arrives first)

CPU Burst (9 time units)	I/O Burst (2)	CPU Burst (2)

P2 (arrives following P1)

CPU Burst (2)	I/O Burst (2)	CPU Burst (2)

P3 (arrives following P2)

CPU Burst (2)	I/O Burst (1)	CPU Burst (2)

P4 (arrives following P3)

CPU Burst (2)	I/O Burst (1)	CPU Burst (1)

Harper is deciding between different timeslices for his round robin scheduler. Assume 0 time for context switching, scheduling, and dispatching.

Q5.1 (4 points)

Help Harper schedule these processes when the timeslice is 3 time units. (0.5 points each)

a) At what time does P1 start its first CPU burst?

b) At what time does P1 completely finish?

c) At what time does P2 start its first CPU burst?

d) At what time does P2 completely finish?

e) At what time does P3 start its first CPU burst?

f) At what time does P3 completely finish?

g) At what time does P4 start its first CPU burst?

h) At what time does P4 completely finish?

Q5.2 (1 point)

Help Harper by calculating the average waiting time for the above processes when the scheduler's timeslice is 5 time units instead. Show your work for credit, and leave your final answer as a fraction.

Q5.3 (4 points)

What is an advantage of longer timeslices? Why?

Q5.4 (4 points)

What is an advantage of shorter timeslices? Why?

Question 6 (10 points)

Consider the following tables. Table A demonstrates a variable size partition scheme. Table B demonstrates a fixed size partition scheme. Assume that P1 has a memory footprint of 4K, P2 has a footprint of 6K, and P3 has a footprint of 12K.

Memory Footprints of each process

Process	**Memory Footprint**
P1	4K
P2	6K
P3	12K

Variable size partitions

Index	**Owning Process**	**Memory**
1	P1	4K
2	Unallocated	11K
3	P3	12K
4	Unallocated	5K
5	P2	6K

Fixed size partitions

Index	**Owning Process**	**Memory**
1	P1	8K
2	Unallocated	5K
3	P3	5K
4	P3	10K
5	Unallocated	2K
6	P2	8K

Q6.1 (2 points)

A new process (P4) is about to be run with a memory footprint of 1K. For each of the two partitioning schemes, indicate in which partition process 4 will be placed. Where it is relevant, use a best fit policy, which allocates the smallest free partition that meets the requirement of the requesting process.

a) What index would P4 be allocated to in the variable size partitioning scheme?

b) What index would P4 be allocated to in the fixed size partitioning scheme?

Q6.2 (4 points)

Assume that shortly after P4 starts, P3 ends its execution and exits. Also assume that you had placed P4 in index 2 for both schemes. Please show work for all parts.

a) What is the total internal fragmentation for the variable size partitioning scheme?

b) What is the total internal fragmentation for the fixed size partitioning scheme?

c) What is the total external fragmentation for the variable size partitioning scheme?

d) What is the total external fragmentation for the fixed size partitioning scheme?

Q6.3 (4 points)

List an advantage for variable partitioning schemes. List an advantage for fixed size partitioning schemes.

Appendix:

R-type instructions (add, nand):

bits 31-28: opcode
bits 27-24: reg X
bits 23-20: reg Y
bits 19-4: unused (should be all 0s)
bits 3-0: reg Z

I-type instructions (addi, lw, sw, beq):

bits 31-28: opcode
bits 27-24: reg X
bits 23-20: reg Y
bits 19-0: Immediate value or address offset (a 20-bit, 2s complement number with a range of -524288 to +524287)

J-type instructions (jalr):[6]

bits 31-28: opcode
bits 27-24: reg X (target of the jump)
bits 23-20: reg Y (link register)
bits 19-0: unused (should be all 0s)

O-type instructions (halt):

bits 31-28: opcode
bits 27-0: unused (should be all 0s)

LC 2200 ISA with an additional LEA and BGT instruction		
Mnemonic Example	**Opcode (Binary)**	**Action Register Transfer Language**
add add $v0, $a0, $a1	0000	Add contents of reg Y with contents of reg Z, store results in reg X. RTL: $v0 ← $a0 + $a1
nand nand $v0, $a0, $a1	0001	Nand contents of reg Y with contents of reg Z, store results in reg X. RTL: $v0 ← ~($a0 && $a1)
addi addi $v0, $a0, 25	0010	Add Immediate value to the contents of reg Y and store the result in reg X. RTL: $v0←$a0+25
lw	0011	Load reg X from memory. The memory address is

lw $v0, 0x42($fp)		formed by adding OFFSET to the contents of reg Y. RTL: $v0 ← MEM[$fp + 0x42]
sw sw $a0, 0x42($fp)	0100	Store reg X into memory. The memory address is formed by adding OFFSET to the contents of reg Y. RTL: MEM[$fp + 0x42] ← $a0
beq beq $a0, $a1, done	0101	Compare the contents of reg X and reg Y. If they are the same, then branch to the address PC+1+OFFSET, where PC is the address of the beq instruction. RTL: if($a0 == $a1) PC ← PC+1+OFFSET
jalr jalr $at, $ra	0110	First store PC+1 into reg Y, where PC is the address of the jalr instruction. Then branch to the address now contained in reg X. Note that if reg X is the same as reg Y, the processor will first store PC+1 into that register, then end up branching to PC+1. RTL: $ra ← PC+1; PC ← $at Note that an **unconditional jump** can be realized using **jalr $ra, $t0,** and discarding the value stored in $t0 by the instruction. This is why there is no separate jump instruction in LC-2200.
halt	0111	Halt the machine
bgt bgt $a0, $a1, done	1000	Compare the contents of reg X and reg Y. If the value in reg X is greater than the value in reg Y, then branch to the address PC+1+OFFSET, where PC is the address of the bgt instruction. RTL: if($a0 > $a1) PC ← PC+1+OFFSET
lea lea $a0, stack	1001	An address is computed by sign-extending bits [19:0] to 32 bits and adding this result to the incremented PC (address of instruction + 1). It then stores the computed address into register DR. RTL: $a0 = MEM[stack]