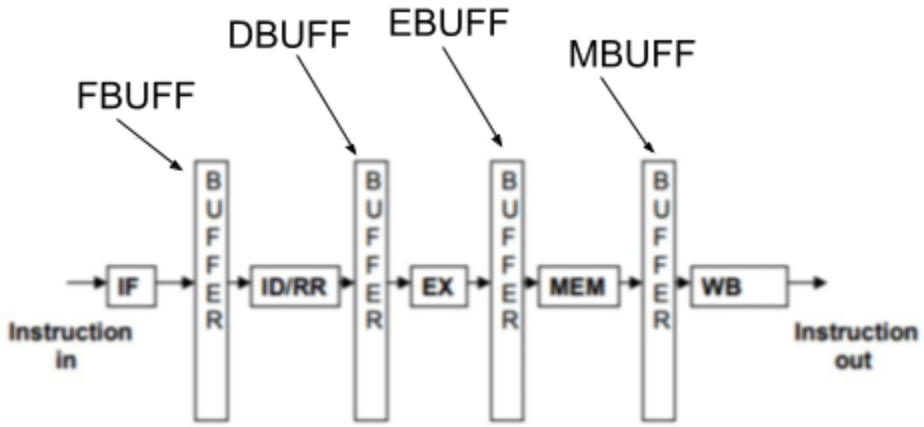
CS 2200 - EXAM 2

You have 70 minutes. The exam includes three bonus points, so the total points tally up to 103. Please be concise: avoid verbose answers. **Good luck!**

The Code educates all members of the Georgia Tech Community about the Institute's expectations and Students' rights and creates a standard by which Students are expected to conduct themselves for the purpose of establishing an environment conducive to academic excellence. Georgia Tech Students, Registered Student Organizations, and Groups are responsible for their own behavior, and the Institute has the authority to establish an internal structure for the enforcement of its policies and procedures, the terms of which students have agreed to accept by their enrollment.

Question 1. Pipeline Buffer (11 points) (6 minutes)



List the minimum size and contents of each pipeline buffer needed to execute the SW instruction of the LC-2200 ISA. Remember that this is a **32-bit** machine. If a certain field is not stored in a given buffer, leave that table entry empty.

Field	FBuff	DBuff	EBuff	MBuff
Instruction				
Opcode				
RX				
RY				
RZ				
Decoded RX				
Decoded RY				
Decoded RZ				
SEXT Offset				
ALU output (RY + Offset)				
MEM[address]				

Question 2. Branch prediction (10 points) (5 minutes)

We have modified the LC-2200 ISA to contain a new conditional branch instruction, **BNEQ**. When this instruction is executed, we branch to a target location if the two register operands are *not* equal. Assume we have a classical five-stage pipeline with all possible data forwarding paths. The pipeline is not equipped with any form of branch prediction and the ISA does not support branch delay slots. Given the following code:

```
        lea $t1, var
        addi $t0, $zero, 4
loop:   addi $t0, $t0, -1
        sw $t0,0($t1)
        bneq $t0, $zero, loop
        addi $v0, $zero, 0xf
var:    .fill 0
```

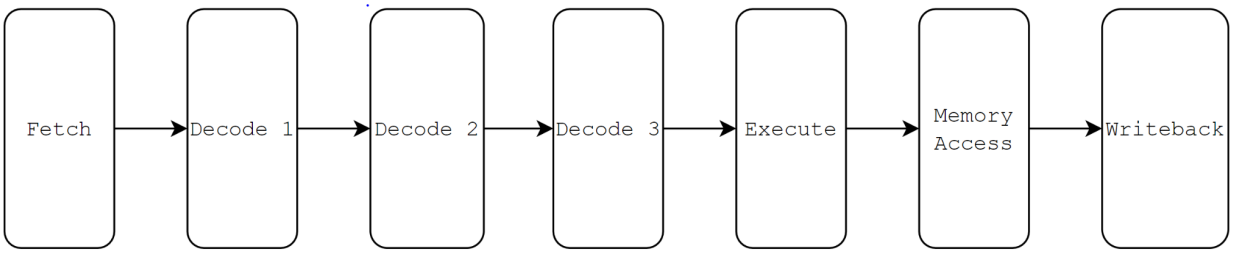
a. Calculate the Cycles Per Instruction (CPI) achieved by the pipeline without any branch prediction. Explain your answer for credit. (6 points)

b. Now assume we extend the pipeline's hardware to always predict that the branch is not taken and calculate the CPI again. Explain your answer for credit. (4 points)

Question 3. Pipeline Depth (9 Points) (6 minutes)

a. The classical pipeline we studied in cs2200 has a total of five stages. Modern processors, however, can have many more (upwards of 15 stages!). What is the benefit of a deeper pipeline compared to a shorter one? (3 points)

b. Say that we modified the LC-2200 pipeline to look like the following figure. Assume the pipeline uses branch prediction and a branch that is *not* taken is mispredicted as taken.



How many bubbles would result in the pipeline? Explain your answer. (3 points)

c. Comment on a potential disadvantage of a longer pipeline on the performance of the currently executing program, other than an increased branch penalty. (3 points)

Question 4. Data Hazards (9 Points) (8 minutes)

Consider the following snippet of LC-2200 assembly code:

```
I1: ADD $t0, $t1, $t2
I2: ADD $t1, $t0, $t1
I3: LW  $t2, 0($t2)
I4: ADD $t0, $t2, $t1
```

a. For each instruction, identify any of the three types of data hazards (RAW, WAR, WAW) that exist between all instruction pairs (e.g., for I4, you need to consider potential hazards with instructions I1, I2, and I3). (3 points)

	I1	I2	I3	I4
I1				
I2				
I3				
I4				

b. Assuming the classical five-stage pipeline without any data forwarding support, count the number of bubbles that will be present upon the execution of the program. Explain your answer for credit. (4 points)

c. Count the number of bubbles with full data forwarding support. Explain your answer for credit. (2 points)

Question 5. Branch Table Buffer (9 Points) (5 minutes)

Consider a 5-stage pipelined processor (same as the one in the textbook) equipped with a branch target buffer (BTB which has only 1 bit of branch history). Assume a 32-bit word-addressable architecture. Upon misprediction, there will be a 2-cycle penalty (i.e., two NOPs). A BEQ instruction is fetched from memory location 1000 by the "IF" stage. The BTB currently has an entry for this BEQ instruction:

```
PC=1000 | branch predicted TAKEN;

PC of BEQ target = 1200;

PC of next sequential instruction = 1001
```

a. What will be the action in IF stage in the next clock cycle? (2 points)
b. When BEQ is in the EX stage, the outcome of the branch turns out to be NOT TAKEN. What are the actions that will ensue as a result of this outcome? (7 points)

Question 6. Scheduling (12 Points) (8 minutes)

Shown below are the duration of the CPU burst and I/O burst times for the three processes.

	P1	P2	P3
CPU	2	6	3
I/O	4	1	3

Assume processes P1, P2, and P3 are ready to run at the beginning of time.
Assume each process does:

- CPU burst; first CPU burst
- I/O burst; one I/O burst
- CPU burst; second CPU burst
- Done; process execution complete

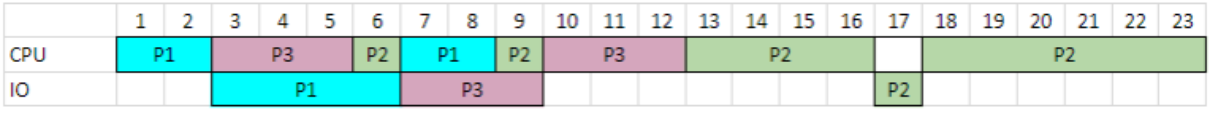
Priorities:

- P3 highest priority
- P2 next higher priority
- P1 lowest priority

Process arrival:

- P1, P2, P3 all arrive within milliseconds of each other in this order
- all are ready to be scheduled at time 0

Shown below are the timelines of activities on the CPU and I/O for the three processes with some scheduling discipline.



a. What scheduling algorithm is implemented here? Explain your choice. (3 points)

b. Why does the scheduler pick P3 to run on the processor at time 10? (3 points)

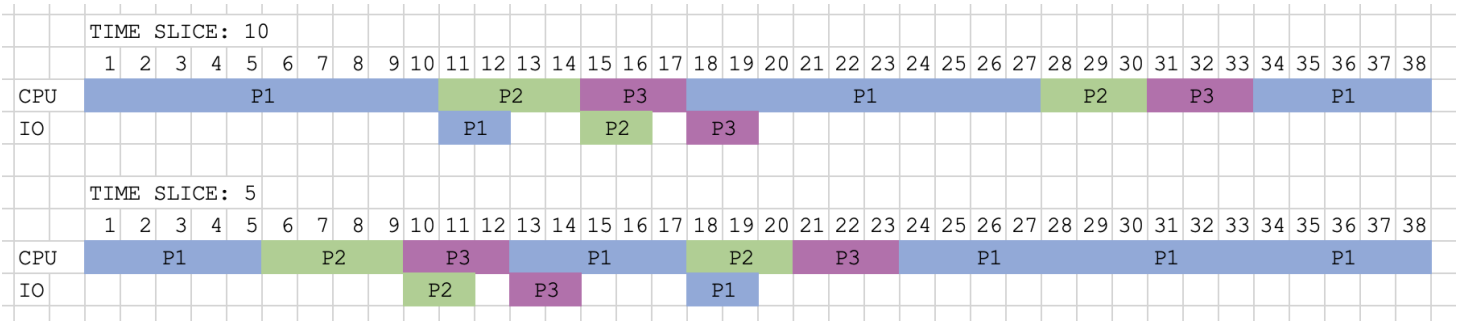
c. What is the wait time for P2? Explain your answer. (3 points)

d. What is the turnaround time for P3? Explain your answer. (3 points)

Question 7. Round Robin Scheduler (17 Points) (10 minutes)

We are writing a new scheduler that will implement a round robin algorithm in order to schedule processes on the processor. The ready queue contains three processes P1, P2, P3 in that order. The execution characteristics of the three processes are as shown in the table below.

Process ID	CPU Burst	I/O Burst	CPU Burst
P1	10	2	15
P2	4	2	3
P3	3	2	3



a. Given the above timeline with a timeslice of ten (10) time units, and assuming no scheduling or context-switching overhead, compute the average waiting time for the three processes. Show your work for credit. (3 points)

b. Given the above timeline with a timeslice of five (5) time units, and assuming no scheduling or context-switching overhead, compute the average waiting time for the three processes. Show your work for credit. (3 points)

c. Now assume that the scheduling and context-switching overhead (i.e., the time to pick the next process to run and dispatch it on the processor) is two (2) time units and I/O completions do not introduce any additional overheads. Compute the average waiting time for the above processes, *for each of the two timeslice values* (5 and 10). Show your work for credit. (5 points)

d. Qualitatively, state your intuition on the effect of the choice of timeslice on the waiting time for processes with short CPU bursts. (3 points)

e. Qualitatively, state your intuition on the effect of the scheduling and context-switching overhead on the waiting time for processes with short CPU bursts. (3 points)

Question 8. Priority Scheduling (6 Points) (4 minutes)

Assume an operating system with a preemptive priority scheduler that runs two application classes A and B. When processes of type A start, they are assigned a high priority; when processes of type B start, they are assigned a low priority.

a. Describe a scenario in which certain processes in our system could experience starvation. (3 points)

b. Describe a policy to prevent processes of type B from starving. (3 points)

Question 9. Hidden? (10 Points) (7 minutes)

Question 10. Hidden? (10 Points) (7 minutes)

Appendix A: Useful Formulas

Name	Notation	Units	Description
CPU Utilization	-	%	Percentage of time the CPU is busy
Throughput	n/T	Jobs/sec	System-centric metric quantifying the number of jobs *n* executed in time interval *T*
Avg. Turnaround time (t_{avg})	$(t_1+t_2+\ldots+t_n)/n$	Seconds	System-centric metric quantifying the average time it takes for a job to complete
Avg. Waiting time (w_{avg})	$((t_1-e_1) + (t_2-e_2)+ \ldots + (t_n-e_n))/n$ or $(w_1+w_2+ \ldots +w_n)/n$	Seconds	System-centric metric quantifying the average waiting time that a job experiences
Response time/ turnaround time	t_i	Seconds	User-centric metric quantifying the turnaround time for a specific job *i*
Variance in Response time	$E[(t_i - e_i)^2]$	$Seconds^2$	User-centric metric that quantifies the statistical variance of the actual response time (t_i) experienced by a process (P_i) from the expected value (t_{avg})

Appendix B: LC-2200 ISA

Mnemonic **Example**	Format	Opcode	Action Register Transfer Language
add **add $v0, $a0, $a1**	R	0 0000_2	Add contents of reg Y with contents of reg Z, store results in reg X. RTL: $v0 ← $a0 + $a1
nand **nand $v0, $a0, $a1**	R	1 0001_2	Nand contents of reg Y with contents of reg Z, store results in reg X. RTL: $v0 ← ~($a0 && $a1)
addi **addi $v0, $a0, 25**	I	2 0010_2	Add Immediate value to the contents of reg Y and store the result in reg X. RTL: $v0 ← $a0 + 25
lw **lw $v0, 0x42($fp)**	I	3 0011_2	Load reg X from memory. The memory address is formed by adding OFFSET to the contents of reg Y. RTL: $v0 ← MEM[$fp + 0x42]
sw **sw $a0, 0x42($fp)**	I	4 0100_2	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, $a1, done**	I	5 0101_2	Compare the contents of reg X and reg Y. If they are the same, then branch to the address PC+1+OFFSET, where PC is the address of the beq instruction. RTL: if($a0 == $a1) PC ← PC+1+OFFSET
Note: For programmer convenience (and implementer confusion), the assembler computes the OFFSET value from the number or symbol given in the instruction and the assembler's idea of the PC. In the example, the assembler stores done-(PC+1) in OFFSET so that the machine will branch to label "done" at run time.			
jalr **jalr $at, $ra**	J	6 0110_2	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.
nop	n.a.	n.a.	Actually a pseudo instruction (i.e. the assembler will emit: add $zero, $zero, $zero
halt **halt**	O	7 0111_2	