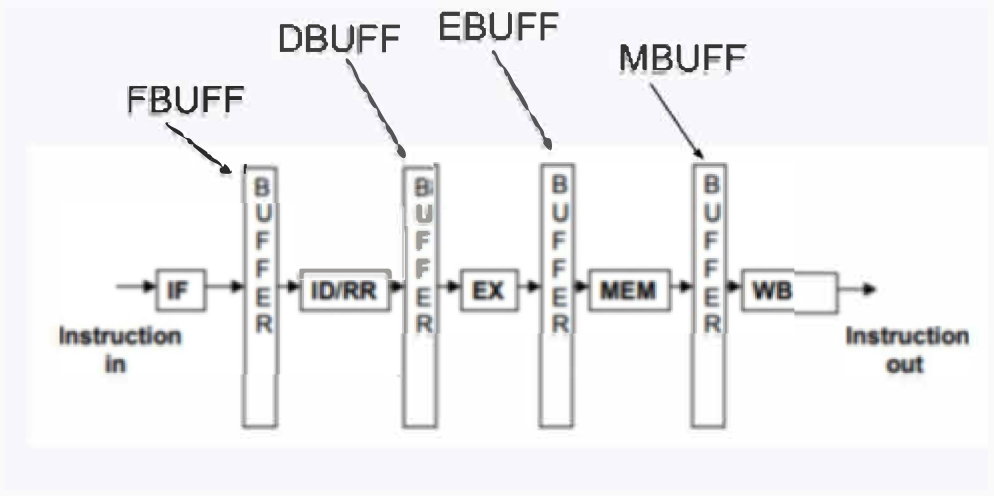
Exam 2 ● GRADED

Q1 Pipeline Buffer
11 Points



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.

Answer format: FBuff/DBuff/EBuff/MBuff (**NO SPACES**)
Example: 32/0/4/0

Q1.1 Instruction
1 Point

32/0/0/0

1.1	Instruction	1 / 1 pt
✓	+ 1 pt	Correct
	+ 0 pts	Incorrect

Q1.2 OPCODE
1 Point

0/0/0/0

1.2	OPCODE	0 / 1 pt
	+ 1 pt	Correct
✓	+ 0 pts	Incorrect

Q1.3 RX
1 Point

0/0/0/0

1.3	RX	1 / 1 pt
✓	+ 1 pt	Correct
	+ 0 pts	Incorrect

Q1.4 RY
1 Point

0/0/0/0

1.4	RY	1 / 1 pt
✓	+ 1 pt	Correct
	+ 0 pts	Incorrect

Q1.5 RZ
1 Point

0/0/0/0

1.5	RZ	1 / 1 pt
✓	+ 1 pt	Correct
	+ 0 pts	Incorrect

Q1.6 Decoded RX
1 Point

0/32/32/0

1.6	Decoded RX	1 / 1 pt
✓	+ 1 pt	Correct
	+ 0 pts	Incorrect

Q1.7 Decoded RY
1 Point

0/32/0/0

1.7 Decoded RY **1** / 1 pt

✓ **+ 1 pt** Correct

+ 0 pts Incorrect

Q1.8 Decoded RZ
1 Point

0/0/0/0

1.8 Decoded RZ **1** / 1 pt

✓ **+ 1 pt** Correct

+ 0 pts Incorrect

Q1.9 SEXT Offset
1 Point

0/20/0/0

1.9 SEXT Offset **0** / 1 pt

+ 1 pt Correct

✓ **+ 0 pts** Incorrect

Q1.10 ALU output (RY + Offset)
1 Point

0/0/32/0

1.10 ALU output (RY + Offset) **1** / 1 pt

✓ **+ 1 pt** Correct

+ 0 pts Incorrect

Q1.11 MEM[address]
1 Point

0/0/0/0

1.11 MEM[address] **1** / 1 pt

✓ **+ 1 pt** Correct

+ 0 pts Incorrect

Q2 Branch prediction
10 Points

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
```

Q2.1
6 Points

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

CPI = Num cycles / num instruction cycles.

The number of instructions can be calculated as: 2 instructions before the loop PLUS 4 instructions in the loop TIMES 4 iterations of the loop - three instructions. i.e. 2 + 4(4) - 3 = 2 + 16 = 15

Num cycles = num instruction cycles + 4 (for all to complete) + num bubbles = 15 + 4 + 2(3) + 1(1) = 15 + 4 + 6 + 1 = 26

26/15 = 1.73

2.1 (no title) 6 / 6 pts

✓	+ 6 pts	Correct (1.73)
	+ 4 pts	Only found the total number of clock cycles taken. (26)
	+ 2 pts	Only found the number of cycles without bubbles. (23)
	+ 2 pts	Only found 15 instructions were executed
	+ 0 pts	Incorrect

Q2.2
4 Points

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.

With the branch prediction the last branch doesn't have a bubble in it because it predicted correctly (no other branches for the first three iterations are cut down on in terms of bubbles) so we have one less clock cycle than the above question

25/15 = 5/3 = 1.67

2.2 (no title) 4 / 4 pts

✓	+ 4 pts	Correct (1.67)
	+ 3 pts	Only found that the cycles were reduced by 1 cycle (25)
	+ 1 pt	Only kept the same number of instructions (15)
	+ 0 pts	Incorrect

Q3 Pipeline Depth
9 Points

Q3.1
3 Points

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?

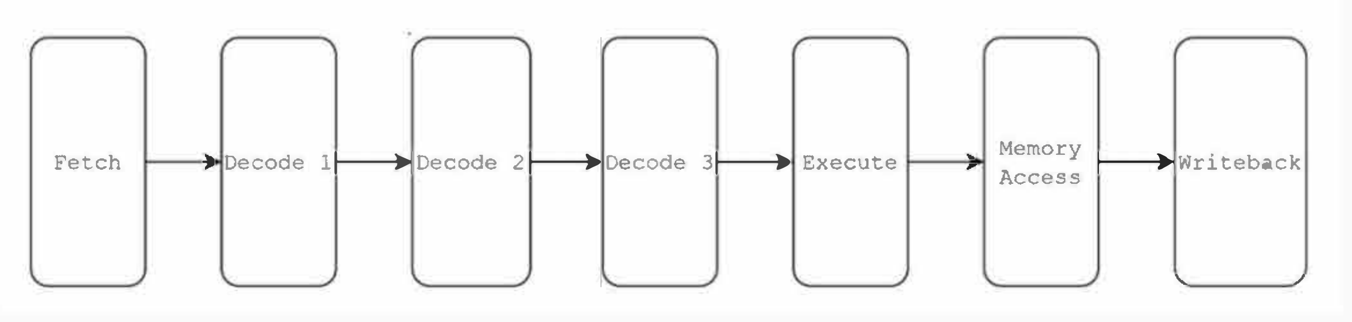
Deeper pipelines have multiple decode and functional pieces of hardware. With a deeper pipeline you can split up the work between stages more evenly. This means that there is less work to do per stage meaning that the clock cycle can be faster meaning that the throughput of the system is better for deeper pipelines.

3.1 (no title) 3 / 3 pts

✓	+ 3 pts	Correct: Allows for... Parallel execution of parts of some instructions, reduced clock cycle width, higher throughput. - Overall improved efficiency.
	+ 0 pts	Incorrect

Q3.2
3 Points

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.

There would be four bubbles since there are 4 stages before the EXECUTE stage.

3.2 (no title) 3 / 3 pts

✓	+ 3 pts	Correct (4 bubble generated)
	+ 0 pts	Incorrect

Q3.3
3 Points

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

It is harder to manage shared resources like registers.

It is more costly since there is more hardware (and complexity) involved.

If the processor didn't have any data forwarding, there would be a larger latency to handle data hazards and would thus bring down the CPI.

3.3 (no title) **3** / 3 pts

✓ **+ 3 pts** Correct with mention of any of the following:

- More bubbles
- More time to clear the pipeline
- Increased pipeline latency
- Greater cycles per instruction (CPI)
- Increased probability of hazards

+ 0 pts Incorrect

Q4 Data Hazards

9 Points

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
```

Q4.1

3 Points

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).

	T1	T2	T3	T4
I1	⬛	⬛	⬛	⬛
I2		⬛	⬛	⬛
I3			⬛	⬛
I4				⬛

Row: I2 Column: T1

[✓ RAW]

[✓ WAR]

WAW

none

Row: I3 Column: T1

RAW

[✓ WAR]

WAW

none

4.1 (no title) **3** / 3 pts

✓ **+ 3 pts** Correct

+ 0 pts Incorrect

Row: I3 Column: T2

RAW

WAR

WAW

[✓ none]

Row: I4 Column: T1

RAW

WAR

[✓ WAW]

none

Row: I4 Column: T2

[✓ RAW]

[✓ WAR]

WAW

none

Row: I4 Column: T3

[✓ RAW]

WAR

WAW

none

Q4.2

4 Points

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.

Bubbles are only a consequence of RAW data hazards. Due to the RAW of I2 trying to read t0 before it is written in I1, this RAW hazard will result in 3 nops to correct since I1 and I2 are back to back and I2 will be in the ID/RR stage and I1 will be in the EX stage so I2 will have to wait for I1 in EX to go through both MEM and WB (3 cycles).

The other RAW hazards come as a consequence of I4. Because I4 tries to read t2 before I3 can write to it, this would also cause 3 nops since I3 and I4 are right next to each other. There is another RAW hazard with I4 trying to read t1 which is written in I2 and would cause 2 nops already, but since the other raw hazard mentioned earlier in this paragraph is already longer than 2 nops we don't have to consider it.

all in all, 3 + 3 = 6.

We have 6 bubbles

4.2 (no title) **4** / 4 pts

✓	**+ 4 pts**	Correct (6 bubbles)
	+ 2 pts	Incorrect working out
	+ 0 pts	Incorrect

Q4.3

2 Points

Count the number of bubbles with full data forwarding support. Explain your answer for credit.

With data forwarding, the only bubble we have is when we have a RAW data hazard where the writing instruction is a load instruction. The only RAW data hazard like this is due to I4 trying to read t2 before I3 can load it into t2. Because of this, when I3 is in the EX stage and I4 is in the ID/RR stage, I4 needs to value of t2 but it has not been loaded yet. Thus, a nop is introduced to the EX stange when I4 goes to the MEM stage. Before this clock cycle is over, I4 will forward the data to I3 in the ID/RR stage so it can continue. Thus, we only need one nop.

We only have 1 bubble with data forwarding.

4.3 (no title) **2** / 2 pts

✓ **+ 2 pts** Correct (1 bubble from I3)

+ 0 pts Incorrect

Q5 Branch Table Buffer
9 Points

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
```

Q5.1
2 Points

What will be the action in **IF** stage in the next clock cycle?

So currently the BTB is predicting that the branch is taken. Because it predicts the branch to be taken, it will retrieve the instruction at the address of the branch target (Address = 1200) in I-MEM during the IF stage.

5.1 (no title) **2** / 2 pts

✓ **+ 2 pts** Correct (target instruction in 1200 loaded)

+ 0 pts Incorrect

Q5.2
7 Points

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?

Because the branch was not taken, there are currently two instructions in our pipeline that are bad. BEQ is in the EX stage meaning that the processor predicted two bad instructions (1 is currently in the IF stage and another is in the ID/RR stage). Upon noticing that the branch is not taken, the EX stage will assert its feedback line to the ID/RR stage to tell it that it must "flush" its instruction. The ID/RR stage will relay this assertion on its feedback line to the IF stage to "flush" the IF stage's instruction. The bad instructions will then be replaced with nops (bubbles).

The BTB will be updated with the misprediction.

5.2 (no title) **5** / 7 pts

+ 7 pts Correct (does all 3)

✓ **+ 2 pts** Only flush the instructions that was predicted in the IF and ID/RR stages.

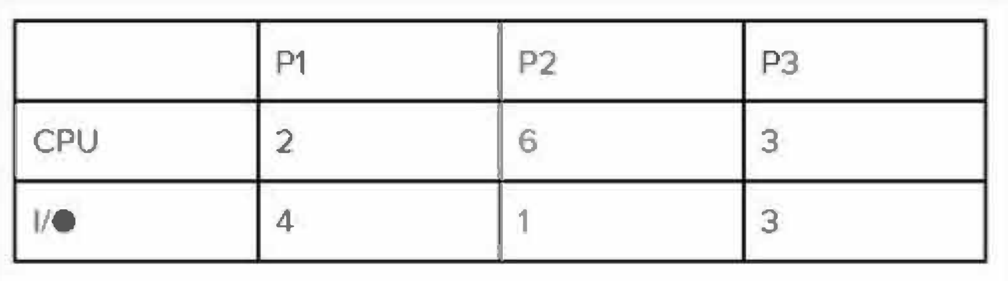
+ 2 pts Only load instruction in address 1001 in IF stage.

✓ **+ 3 pts** Only update the BTB with Not Taken

+ 0 pts Incorrect

Q6 Scheduling
12 Points

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/●	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**

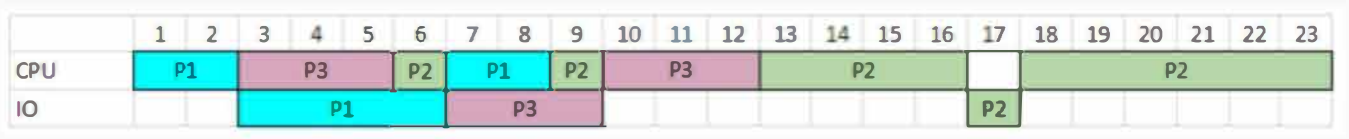
Priorities:

- 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.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
CPU	P1		P3			P2	P1		P2	P3			P2					P2					
IO			P1				P3										P2						

Q6.1

3 Points

What scheduling algorithm is implemented here? Explain your choice.

Because at time t = 7, P1 is given priority over P2 (which was being executed in t = 6) and, at t = 10 P3 is given priority over P2, we have the SRTF since at t = 7 P1 needs only two clock cycles to complete its CPU burst (whilst P2 needs five more) and at t = 10 P3 only needs three more clock cycles to complete its CPU burst (whilst P2 needs four more).

6.1 (no title) 3 / 3 pts

✓ + 3 pts Correct (Preemptive SRTF)

+ 1 pt Correct but wrong reasoning

+ 0 pts Incorrect

Q6.2

3 Points

Why does the scheduler pick P3 to run on the processor at time 15?

Correction: This question should be at time t = 10

Because we are running SRTF scheduling and at t = 10 P3 only needs three more clock cycles to complete its CPU burst (whilst P2 needs four more). Thus there are fewer clock cycles to complete P3's CPU burst than P2's.

6.2 (no title) 3 / 3 pts

+ 0 pts Incorrect

✓ + 3 pts Correct (When P3 enters the queue, it should have a shorter time remaining)

Q6.3

3 Points

What is the wait time for P2? Explain your answer.

wait time = time not executing

wait time for P2 = 5 + 2 + 3 = 10

6.3 (no title) 3 / 3 pts

✓ + 3 pts Correct (10)

+ 2 pts Correct time it stays in the queue (23)

+ 1 pt Correct time of running (13)

+ 0 pts Incorrect

Q6.4

3 Points

What is the turnaround time for P3? Explain your answer.

Turnaround time = time it takes to complete a process

Turnaround time for P3 = 12

12 clock cycles

6.4 (no title) 3 / 3 pts

✓ + 3 pts Correct (12)

+ 2 pts Correct completion time (12)

+ 1 pt Correct arrival time (0)

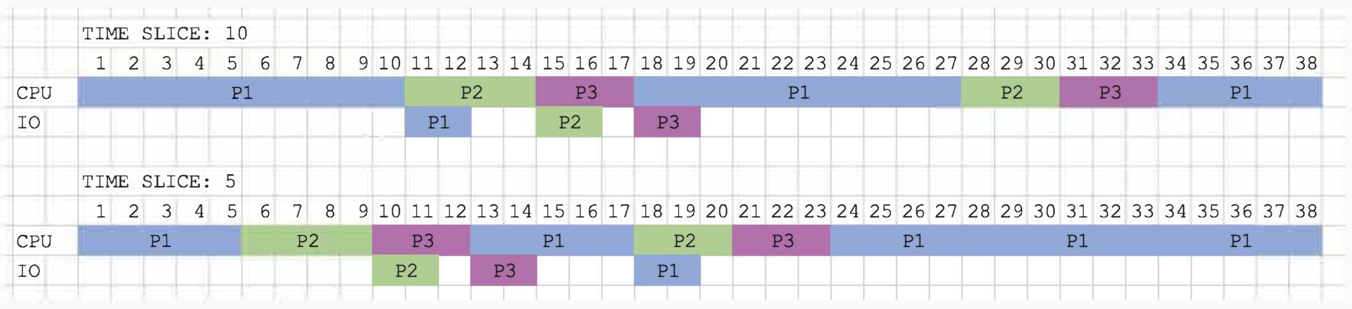
+ 0 pts Incorrect

Q7 Round Robin Scheduler

17 Points

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



Q7.1

3 Points

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.

Average wait time = Σ(wait times of all processes)/num processes

Wait time of process 1 = (2+3) + (3 + 3) = 11
Wait time of process 2 = 10 + (1 + 10) = 21
Wait time of process 3 = 14 + (8 + 3) = 25

Average wait time = (11 + 21 + 25)/3 = 19

7.1 (no title) **3** / 3 pts

✓	**+ 3 pts**	Correct (19)
	+ 1 pt	Correct P1 wait time (11)
	+ 1 pt	Correct P2 wait time (21)
	+ 1 pt	Correct P3 wait time (25)
	+ 0 pts	Incorrect

Q7.2

3 Points

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.

Average wait time = Σ(wait times of all processes)/num processes

Wait time of process 1 = (4 + 3) + 4 = 11
Wait time of process 2 = 5 + (1 + 5) = 11
Wait time of process 3 = 9 + (3 + 3) = 15

Average wait time = (11 + 11 + 15)/3 = 12.33

7.2 (no title) **3** / 3 pts

✓	**+ 3 pts**	Correct
	+ 1 pt	Correct P1 wait time (11)
	+ 1 pt	Correct P2 wait time (11)
	+ 1 pt	Correct P3 wait time (15)
	+ 0 pts	Incorrect

Q7.3

5 Points

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.

Average wait time = Σ(wait times of all processes)/num processes

Timeslice for 10

Wait time of process 1 = 11
Wait time of process 2 = 21
Wait time of process 3 = 25
overhead1 = 2 * 6 = 12
overhead2 = 2 * 4 = 8
overhead3 = 2 * 5 = 10

Average wait time = (11 + 21 + 25 + 12 + 8 + 10)/3 = 29

7.3 (no title) **3** / 5 pts

+ 5 pts Correct (29, 23,67)
✓ + 3 pts Only one correct
+ 0 pts Incorrect

Timeslice for 5

Wait time of process 1 = (4 + 3) + 4 = 11
Wait time of process 2 = 5 + (1 + 5) = 11
Wait time of process 3 = 9 + (3 + 3) = 15
overhead1 = 2 * 6 = 12
overhead2 = 2 * 4 = 8
overhead3 = 2 + 2 + 2 + 2 + 2 = 10

Average wait time = (11 + 11 + 15 + 12 + 8 + 10)/3 = 22.3

Q7.4
3 Points

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

For longer processes, it doesn't seem to change the waiting time (P1 waiting time for all questions above was 11) but it seems to really drive up the time for shorter jobs when it is longer. Having a shorter timeslice give a fairer share of CPU time thus decreasing the average wait time.

7.4 (no title) **3** / 3 pts

✓ + 3 pts Correct (less wait time)
+ 0 pts Incorrect

Q7.5
3 Points

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

It drives up wait time since they have to deal with switching between all the different processes. If the number of switches can be minimized we can minimize overhead.

7.5 (no title) **3** / 3 pts

✓ + 3 pts Correct (short CPU bursts have more overhead)
+ 0 pts Incorrect

Q8 Priority Scheduling
6 Points

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.

Q8.1
3 Points

Describe a scenario in which certain processes in our system could experience starvation.

If our scheduling scheme has priority, then processes with low priority may suffer from starvation.

With the information given, we know we are running a preemptive priority scheduler (so there is priority) and that process A has a higher priority than B when it starts. Thus, if type A processes keep getting requested before they are completed then they can continuously get scheduled over type B thus starving B.

8.1 (no title) **3** / 3 pts

✓ **+ 3 pts** Correct

+ 0 pts Incorrect

Q8.2
3 Points

Describe a policy to prevent processes of type B from starving.

Over time, we could up the priority of type B so that eventually it could gain more priority than type A and be scheduled on the CPU.

8.2 (no title) **3** / 3 pts

✓ **+ 3 pts** Correct (Dynamic Priority)

+ 0 pts Incorrect

Q9 Hidden Question
10 Points

Q9.1
5 Points

A process has not been running on the CPU for a long time. Assume the scheduler ensures that all runnable processes get a fair share of the CPU time. What could be the reason for the process not getting any CPU time? What data structure would you check to debug this situation? What specific field of the data structure would you check to ascertain the reason?

Priority Queue. It might not be getting CPU time since the queue keeps pushing more important processes before it. We should look at the

9.1 (no title) **0** / 5 pts

+ 5 pts Correct (check IO queue or PCB state)

+ 3 pts Only gave a reason the process didn't give CPU time

✓ **+ 0 pts** Incorrect

You would want to check the I/O queue to see if the process is waiting to complete its I/O burst, You could also check the PCB state to see if it is "Waiting."

Q9.2
5 Points

The dispatcher is an important component of the scheduler. What are the responsibilities of the dispatcher?

The dispatcher is what sends a process onto the CPU. Its responsibilities include looking at the priorities of the processes and which one should get CPU time.

9.2 (no title) **0** / 5 pts

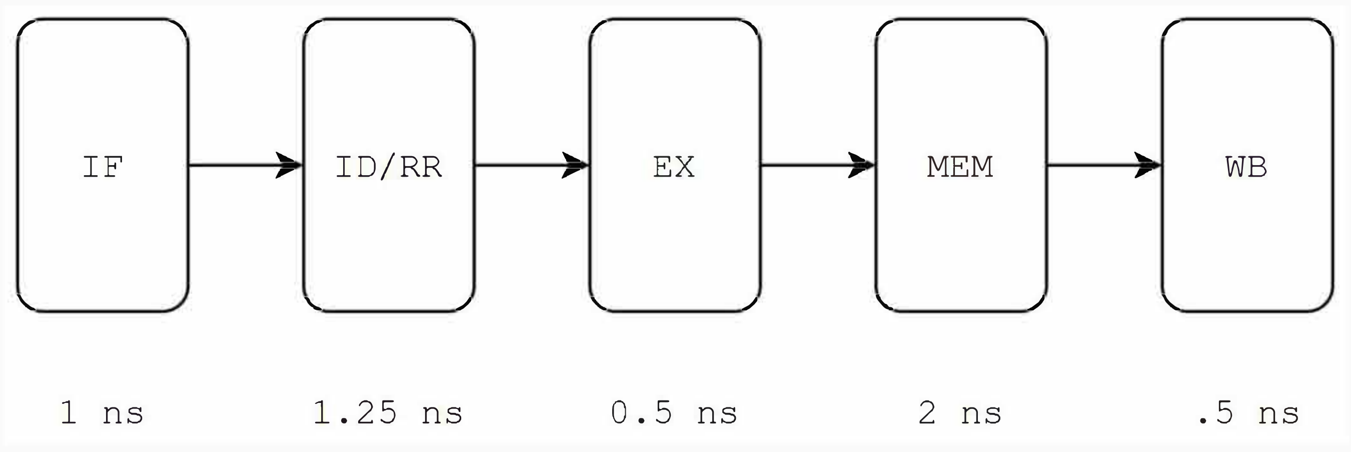
+ 5 pts Correct (load PC and registers from PCB)

✓ **+ 0 pts** Incorrect

The dispatcher, loads the PC and register values from the PCB. You have described a part of the scheduler

Q10 Hidden Question
10 Points

Assume we have the following pipeline and the critical path of each stage's datapath.



Q10.1
5 Points

What is the maximum clock cycle frequency this pipeline can operate at?

The clock must run at 2 ns since that is the slowest time in any stage. Each stage must be 1 clock cycle long, so despite every other stage but the MEM stage less than 2 ns, we must make the clock cycle 2 ns so that the MEM stage can finish what its doing before the next clock cycle.

10.1 — (no title) 3 / 5 pts

	+ 5 pts	Correct (500 MHz)
	+ 5 pts	Correct work, wrong conclusion
✓	+ 3 pts	Using the stage with the max clock cycle width (2ns)
	+ 2 pts	Correct formula for frequency (1/clock cycle width)
	+ 0 pts	Incorrect

Q10.2
5 Points

We now decide to modify the pipeline to only have four stages. We must combine two stages to accomplish this. Which two stages should we combine to maintain maximum performance? Explain your answer.

We should combine the ID/RR and EX stages because they won't increase our maximum cycle time (1.75 ns vs 2 ns) and we can get the output of the REG file and then do combinational logic in EX to finish EX.

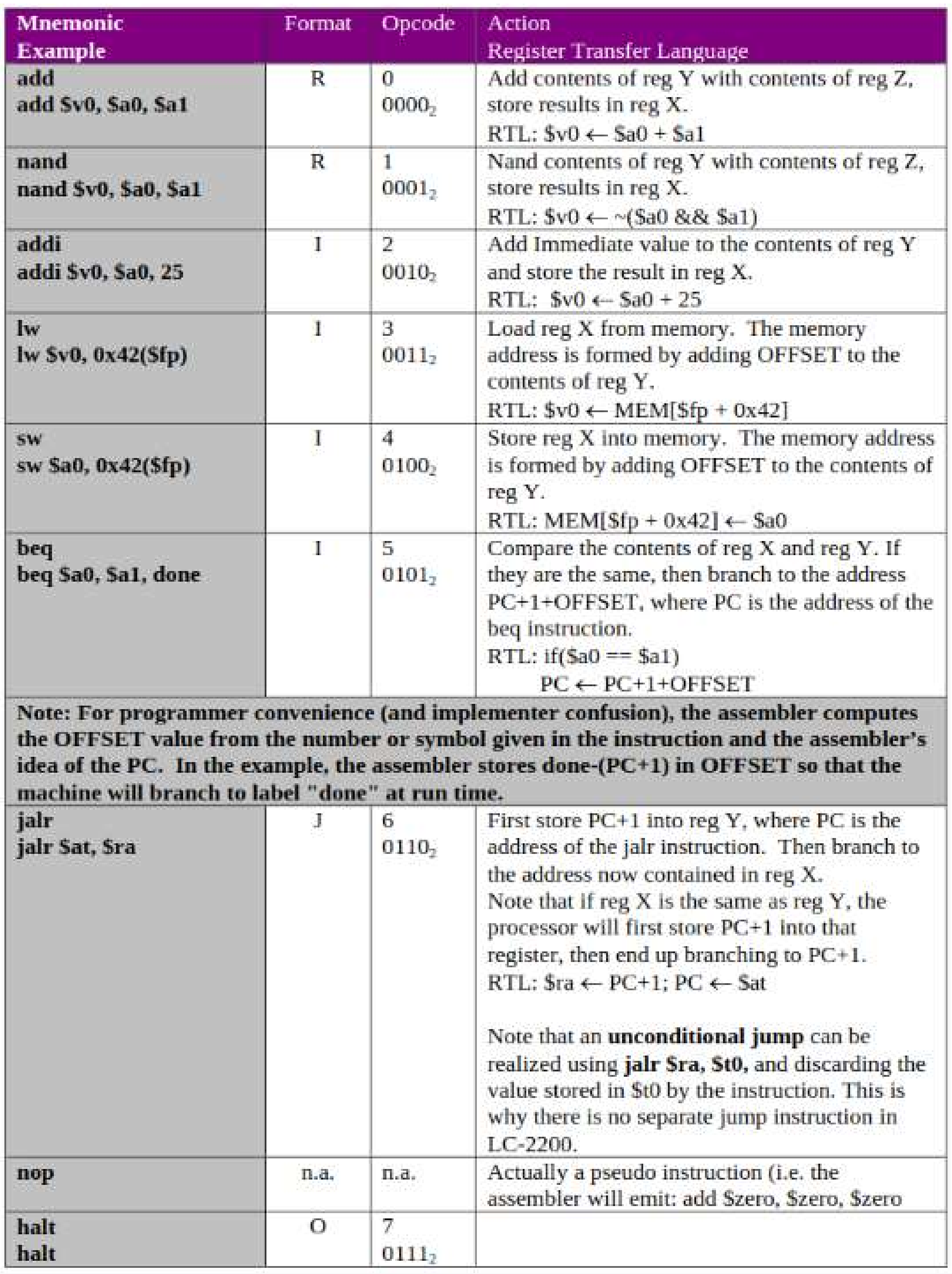
10.2 — (no title) 5 / 5 pts

✓	+ 5 pts	Correct (ID/RR and EX)
	+ 0 pts	Incorrect

Q11 Appendix A: Useful Formulas
0 Points

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})

Q12 Appendix B: LC-2200 ISA
0 Points



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	

Exam 2

● GRADED

STUDENT
Eric Anders Gustafson

TOTAL POINTS
85 / 103 pts

QUESTION 1

Pipeline Buffer **9** / 11 pts

1.1	Instruction	**1** / 1 pt
1.2	OPCODE	**0** / 1 pt
1.3	RX	**1** / 1 pt
1.4	RY	**1** / 1 pt
1.5	RZ	**1** / 1 pt
1.6	Decoded RX	**1** / 1 pt
1.7	Decoded RY	**1** / 1 pt
1.8	Decoded RZ	**1** / 1 pt
1.9	SEXT Offset	**0** / 1 pt
1.10	ALU output (RY + Offset)	**1** / 1 pt
1.11	MEM[address]	**1** / 1 pt

QUESTION 2

Branch prediction 10 / 10 pts

2.1 (no title) 6 / 6 pts

2.2 (no title) 4 / 4 pts

QUESTION 3

Pipeline Depth 9 / 9 pts

3.1 (no title) 3 / 3 pts

3.2 (no title) 3 / 3 pts

3.3 (no title) 3 / 3 pts

QUESTION 4

Data Hazards 9 / 9 pts

4.1 (no title) 3 / 3 pts

4.2 (no title) 4 / 4 pts

4.3 (no title) 2 / 2 pts

QUESTION 5

Branch Table Buffer 7 / 9 pts

5.1 (no title) 2 / 2 pts

5.2 (no title) 5 / 7 pts

QUESTION 6

Scheduling 12 / 12 pts

6.1 (no title) 3 / 3 pts

6.2 (no title) 3 / 3 pts

6.3 (no title) 3 / 3 pts

6.4 (no title) 3 / 3 pts

QUESTION 7

Round Robin Scheduler 15 / 17 pts

7.1 (no title) 3 / 3 pts

7.2 (no title) 3 / 3 pts

7.3 (no title) 3 / 5 pts

7.4 (no title) 3 / 3 pts

7.5 (no title) 3 / 3 pts

QUESTION 8

Priority Scheduling 6 / 6 pts

8.1 (no title) 3 / 3 pts

8.2 (no title) 3 / 3 pts

QUESTION 9

Hidden Question 0 / 10 pts

9.1 (no title) 0 / 5 pts

9.2 (no title) 0 / 5 pts

QUESTION 10

Hidden Question 8 / 10 pts

10.1 **(no title)** 3 / 5 pts

- **+ 5 pts** Correct (500 MHz)
- **+ 5 pts** Correct work, wrong conclusion
- ✓ **+ 3 pts** Using the stage with the max clock cycle width (2ns)
- **+ 2 pts** Correct formula for frequency (1/clock cycle width)
- **+ 0 pts** Incorrect

10.2 (no title) 5 / 5 pts

QUESTION 11

Appendix A: Useful Formulas 0 / 0 pts

QUESTION 12

Appendix B: LC-2200 ISA **0** / 0 pts