NetID:_____

Question 1: Tomasulo (14 points)

Consider a Tomasulo OoO execution computer with 3 integer units (int0, int1, int2) and 2 floating point units (fp0, fp1).

Instructions are issued one at a time, to the lowest numbered reservation station available. The instruction queue is preloaded with the entire program before the beginning of cycle 0., so the first instruction is issued in cycle 0. For all instructions, computation may begin as early as the cycle *after* it is issued, or the cycle after the last source has been committed. Reservation stations may be filled the cycle *after* an instruction is committed. Thus, if int1 commits in cycle 7, a new instruction may be placed in int1 in cycle 8 and can begin execution in cycle 9 (if it's operands are ready). Additionally, another instruction dependent on the result of what is in int1 may begin execution on cycle 8.

Instructions will commit their results in the last cycle of execution. Thus, if an instruction takes two cycles to complete and begins in cycle 3, it will commit in cycle 4. Integer addition and subtraction take 2 cycles; integer multiplication takes 5 cycles; floating point conversion takes 2 cycles; floating point multiplication takes 8 cycles; and floating point division takes 12 cycles. Branch instructions are calculated in an integer station and take 3 cycles to execute. If multiple instructions attempt to commit in the same cycle, the instruction that was issued first takes priority.

All arithmetic instructions are encoded as `operation dest, source1, source2` and floating point conversion is encoded as `fconv dest, source`. Floating point registers are `f#`, integer registers are `x#`, `source1` is the left hand operand, and `source2` is the right hand operand.

Registers `x0` and `f0` are hard wired to 0. The remaining registers have the following initial values:

x1	x2	x3	x4	x5	x6	x7	x8	f1	f2	f3	f4
5	7	12	0	0	0	236	0	0.0	24.0	8.0	0.0

NetID:_____

1. **(4 points)** Given the initial register values above, finish filling out the table below for the given instructions. **Source 1** and **Source 2** fields should contain the renamed register.

Instruction	Cycle Issued	Reservation Station Assigned	Source 1	Source 2	Start Cycle	Commit Cycle	Result
`add x4, x2, x1`	0						
`mul x5, x4, x3`							
`sub x5, x5, x2`							
`add x8, x4, x7`							
`add x6, x5, x8`							
`fdiv f1, f2, f3`							
`fconv f4, x5`							
`fmul f1, f4, f1`							

2. The example program above does not take advantage of the parallelism inherent in a Tomasulo computer. To improve the program, you reorder some of the operations in the original code.
    a. **(4 points)** Repeat part a with the reordered instructions below

Instruction	Cycle Issued	Reservation Station Assigned	Source 1	Source 2	Start Cycle	Commit Cycle	Result
`fdiv f1, f2, f3`	0						
`add x4, x2, x1`							
`mul x5, x4, x3`							
`sub x5, x5, x2`							
`add x8, x4, x7`							
`add x6, x5, x8`							
`fconv f4, x5`							
`fmul f1, f4, f1`							

b. **(2 points)** Calculate the performance improvement (speedup) of the two versions of the program. Express your answer as a multiplier (e.g. 1.7x, or 17/10).

3. Your partner, Ben Bitdiddle, thinks he has found a way to improve the efficiency of your design even further. He proposes that the common data bus be routed from the output of the functional units directly into the register file, and the inputs of the reservation stations come solely from the output of the register file, not from the CDB. Ben argues that this simplification will improve routing and increase fmax, as the logic feeding the reservation stations is simpler.

   a. **(2 points)** Assuming Ben's assumptions are correct, why may this solution not be ideal? Give a **quantitative** example of how this "improvement" affects the system.

   b. **(2 points)** Give an example of when Ben's solution would perform better, despite the issue identified above.

Question 2: Mild Adventures Into Complex Processors (15 points)

Part 1. Lucas wants to run a number of tests on **the same machine from the previous problem**, to ensure that the processor functions appropriately. In order to do so, he set up a testbench that repeatedly initializes the registers to random values and runs a short test code. Each modification of the register file is checked against the corresponding change in a functional multicycle processor. Having used the same testbench to thoroughly test many designs before, Lucas knows **the testbench is bug-free**. The current test code is given below:

```
mul x1, x2, x3
add x4, x5, x6
not x7, x4
```

To his surprise, the system continuously throws **warnings during execution**, even when the final output is correct, and he can't figure out why! He tries modifying the test code a little bit:

```
mul x1, x2, x3
add x4, x5, x1
not x7, x4
```

Now the program **runs perfectly fine**, which makes him even more confused. In a panic, he escapes to a new full-time job in California and is never seen again. Thus, as an ECE 411 trainee, it is now your job to fix the problem.

- **(3 points)** Why are warnings only thrown in the first test code and not in the second, and how can it be resolved by **modifying the processor**? During which cycles of the first test code does the issue occur?

Part 2. Your new boss, who calls himself Robert, is frustrated at the processor's exceedingly poor performance when dealing with branch and jump instructions, and wishes to add **support for speculative execution**. He changes the design of the Tomasulo processor to **include a ROB** (ReOrder Buffer, which is a backronym named after himself to inflate his ego), placed between the common output bus and the register file. He hands you the following design document listing some of its design features:

- **16 entries**, such that it will not be a bottleneck in the system.
- Entries are allocated in the ROB upon the issuing of an instruction, in circular order (starting from 0). **Only one entry can be committed** into the register file per cycle.
- In the event of a branch misprediction, the flushing process occurs on the final cycle of evaluation alongside the fetching of the new PC. The flushing process itself simply **replaces the flushed entries with NOPs (encoded as add x0, x0, x0),** and begins issuing new entries **after the last flushed entry.**
- Rob implements a **static not-taken branch predictor** in the processor

a. **(3 points)** How is **Rob's flushing logic inefficient (in terms of time and area),** and how do its effects manifest during the execution of various branch functions? How can you improve his system to resolve this issue?

After resolving the inefficient flushing logic, Rob continues to run various **randomly-generated test codes** on his machine. Even though **the functionality of the system is entirely correct,** he gets extremely frustrated on one particular test code, saying that it **performs far worse than it should.** Like any good boss, he sends you the test code and tells you to optimize the system for him:

```
      add x7, x0, x3
      beq x0, x0, LABEL0
LABEL0:
      mul x7, x7, x1
```

b. **(3 points)** Why does this piece of code still perform unexpectedly slowly on Rob's improved system? How can you **change Rob's ROB design to optimize** for this situation? Do you think this modification is necessary if the processor will run **programs compiled from higher-level languages**? Briefly justify your answer.
(Hint: Think about what makes this situation different from most other branches/jumps.)

Part 3. Rob wants to do better this time and decides to develop a **5-stage RISC-V processor pipeline**, wherein he develops a new feature that he calls the **RAT, or Register Alias Table.** He creates **64 registers** in a physical regfile, and uses a simple scheme to remap the logical destination register of an instruction to a **free, currently unused physical register upon issuing an instruction**.

a. **(3 points)** When he takes snapshots of the **logical register values** during the execution of various test codes, a number of the displayed registers are set unexpectedly to **uninitialized values or results from previously executed operations**! Why is this issue occurring in Rob's design, and how can he fix it to **allow for precise exceptions**? Give an example of a situation that would cause such an error.

b. **(3 points)** Rob now wants you to **fix his error using less than 50 bytes of additional metadata** (eg. you can create any data arrays, registers, buffers, etc. as long as their combined data capacity is less than 50B). Can your solution to the previous problem be implemented with this restriction imposed? If not, design a simple system that will work under this condition. **Justify your solution to Rob** by giving a brief analysis on the amount of data stored in your new solution.

**Keep your designs at a high level - your answer does not need to contain any detailed logic or signal definitions. For example, if your system needs to flush the RAT for some reason, just say so - don't waste time designing specific trigger logic for the flush.*

Question 3: SW Techniques for ILP (11 Points)

Consider a function to multiply a fixed-sized vector by a scalar:

```
void vec_muls(int32_t* vec, int32_t s) {
     for (size_t i = 0; i < 256; ++i) {
          vec[i] *= s;
     }
}
```

In RV32im, the loop is compiled into

```
; x12 is int32_t * vec
; x13 is int32_t s
addi x5, x12, 1024
loop:
     lw x6, 0(x12)
     addi x12, x12, 4
     mul x6, x6, x13
     sw x6, -4(x12)
bne x12, x5, loop
```

Assume a five stage pipeline architecture, and the 16 KB direct mapped L1 Data Cache has 32 B data lines. Assume that at the beginning of execution the L1 data cache is empty, the L1 instruction cache contains all of the relevant instructions, that cache misses result in a 25 cycle penalty, and that there is zero branch mispredict penalty. **Assume that there is a one cycle stall when there is a RAW dependency between a load instruction and any immediately subsequent instruction.**

1. **(3 points)** How many cycles does the assembly code stall for?

2. In order to increase program performance, the ISA is modified to have a prefetch cache line instruction: `prefd off12(rs1)` instruction takes a single cycle to execute and instructs the memory subsystem to prefetch the cache line at address `rs1 + sext(off12)` into the L1 data cache. The prefetching itself takes 25 cycles, but during this time period, the pipeline is still issuing and committing instructions.

    a. **(5 points + 2 EC)** Unroll the assembly loop and use the `prefd` instruction. The loop body should act on data at cache-line granularity instead of word granularity. The original assembly is repeated on this page for your convenience. (Extra Credit: Perform software pipelining concurrently with loop unrolling)

```
loop:
      lw x6, 0(x12)
      addi x12, x12, 4
      mul x6, x6, x13
      sw x6, -4(x12)
bne x12, x5, loop
```

b. **(3 points)** How many cycles does the unrolled assembly code stall for? Make sure to account for the **one cycle stall when there is a RAW dependency between a load instruction and any immediately subsequent instruction**.

Question 4: Memory Systems (15 points)

1. Consider the eight memory requests in the memory controller buffer shown in Fig 4.1. Each reference is represented by the triple (bank, row, column). The DRAM operations are bank precharge (3 cycles), row activation (3 cycles), and column access (1 cycle). Assume there is an active row in each bank different than the requested rows, and there is no refresh operation required within the given time window (60 Cycles).

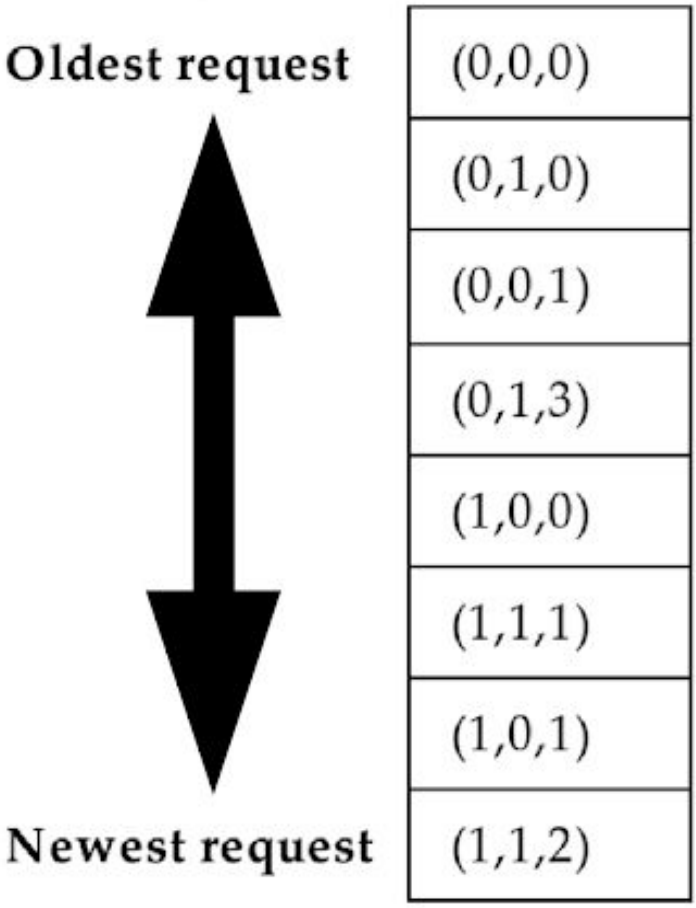
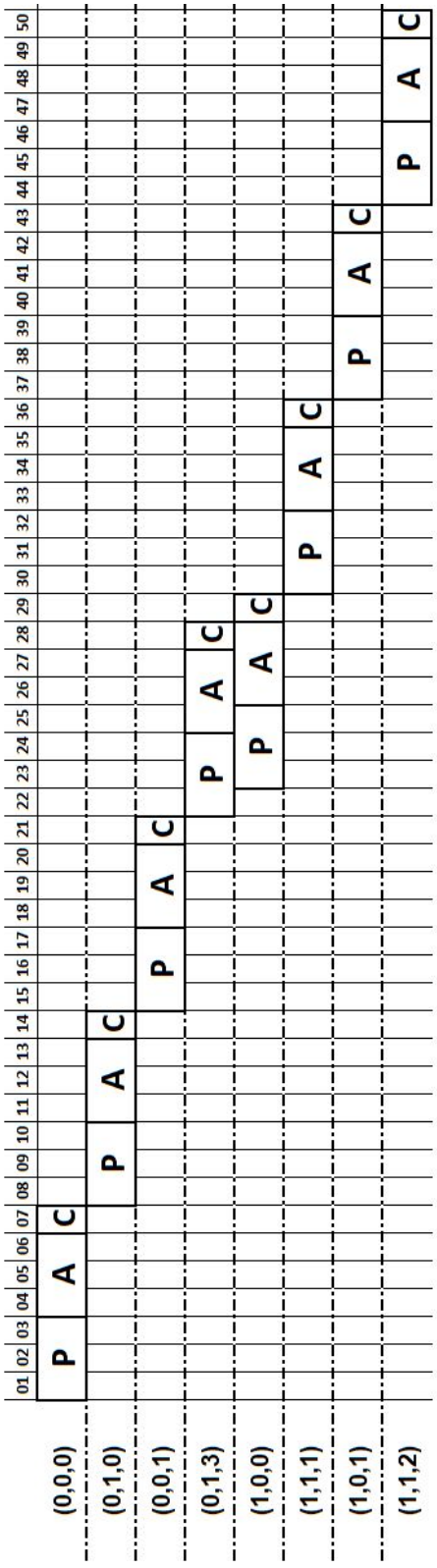
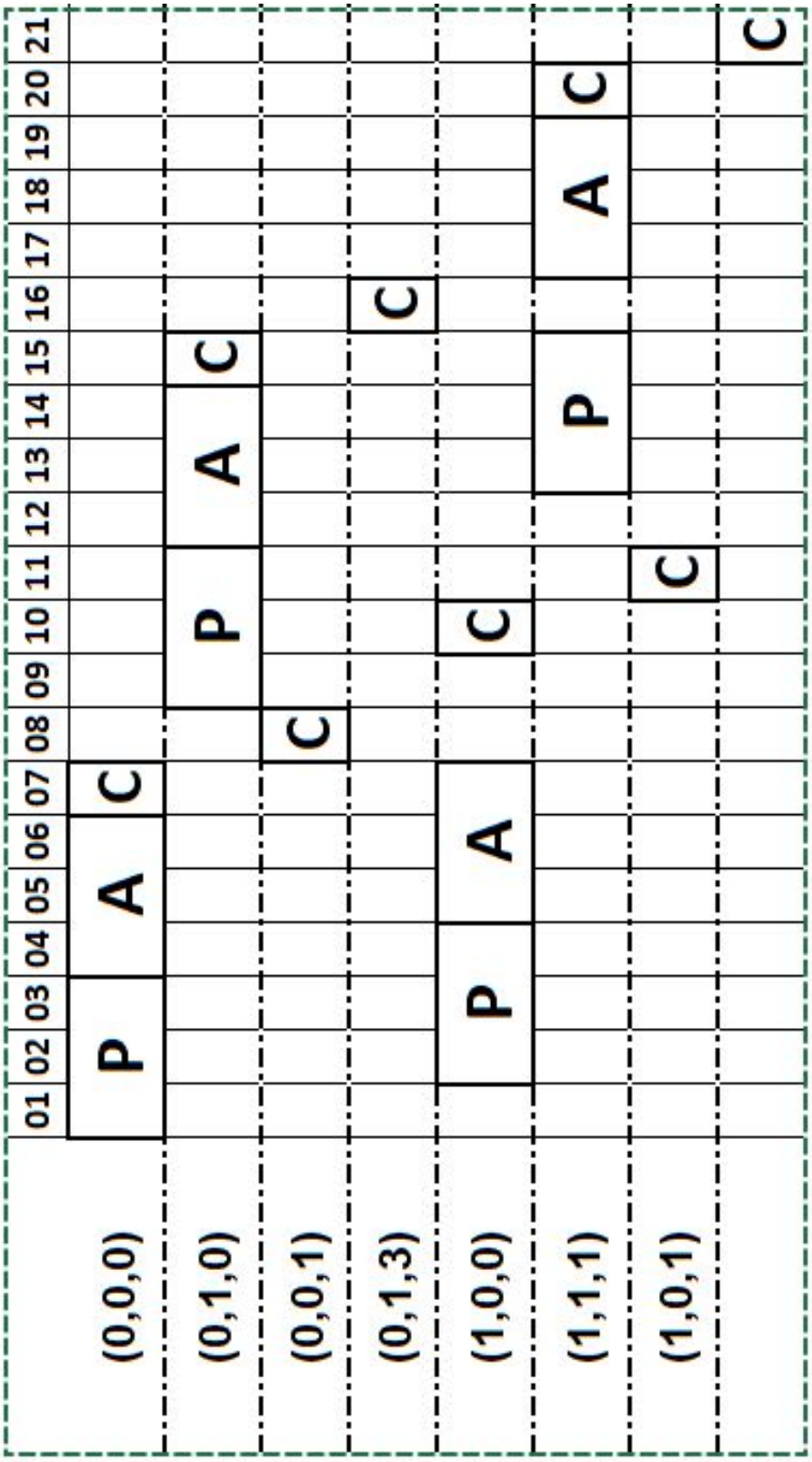


Fig 4.1 Memory Requests in the request queue in the memory controller

a. **(4 points)** Assuming the memory controller is using the First-Come First-Serve (FCFS) scheduling policy, fill the following:



b. **(4 points)** Assuming the memory controller is using the First-Ready First-Come First-Serve (FR-FCFS) scheduling policy, fill the following.



2. A memory system has four channels, each channel has two DIMMs, and each DIMM has one rank. Each memory channel is controlled by a separate memory controller. Each rank of DRAM contains eight banks. A bank contains 32K rows. Each row in one bank is 8KB. The minimum retention time among all DRAM rows in the system is 64 ms. In order to ensure that no data is lost, every DRAM row is refreshed once per 64 ms. Every DRAM row refresh is initiated by a "refresh" command from the memory controller which occupies the command bus on the associated memory channel for 5 ns and the associated bank for 40 ns. Let us consider a 1.024 second span of time.
   (We define a resource utilization for a given operation as the fraction of total time for which that resource is occupied by the given operation.)

   a. **(2 points)** How many refreshes are performed by the memory controllers during the 1.024 second period in total across all four memory channels?

   b. **(2 points)** What command bus utilization, across all memory channels, is directly caused by DRAM refreshes?

   c. **(1 point)** What data bus utilization, across all memory channels, is directly caused by DRAM refreshes?

   d. **(2 points)** What bank utilization (on average across all banks) is directly caused by DRAM refreshes?

NetID:_____

Question 5: Potpourri (10 points)

1. **(2 points)** What changes would you make to the Tomasulo design if you want to support speculation, but do not mind out-of-order completion? Expain. Only correct explanation earns points.

2. **(2 points)** Does the optimal number of pipeline stages increase or decrease when the optimization metric is not performance, but energy? Explain. Only correct explanation earns points.

3. **(3 points)** In a deeply speculative processor (e.g., one with a large number of branch instructions in-flight), flushing of all speculative state on a branch misprediction can be expensive. Suggest an optimization to reduce the cost of speculation for such processors.

4. **(2 points)** Suggest two reasons why specialization often improves energy efficiency.

5. **(1 point)** Answer one or both of the following two:

- "Design a cool logo for your MP3 team."

- "Write your own (short) question for us to answer!" (any question)

Problem 3: DRAM (12 Points)

A DRAM system has the following configuration

- First Come First Serve Scheduling (Requests are processed in the order they arrive)
- 64 bit Data Bus
- Single Memory Channel, Single DIMM, and Single Rank
- RAS delay of 40ns
- CAS delay of 15ns
- Precharge delay of 20ns
- Addresses are 12 bits
- Bits [4:0] are used for column address
- Bits [7:5] are used for bank index
- Bits [11:8] are used for row index

The following accesses are made:

1. 1000 101 00000
2. 1100 101 10010
3. 1010 100 00000
4. 1010 010 10000
5. 1000 101 11000
6. 1001 111 11000

1. Assume the DRAM has an Open Page/Row policy. Calculate the total latency of this sequence of accesses. Show your work (4 Points)

2. Suggest an indexing scheme to maximize performance for the open row policy. Your answer should explicitly list which bits are used for column, bank and row index. Calculate the latency with your new proposed scheme. (4 Points)

3. One way of improving performance in the DRAM subsystem is increasing the number of banks to reduce the number of bank conflicts. Suggest an indexing scheme for a DRAM with double the number of banks. (4 Points)

Problem 6: Potpourri (11 Points)

1. Suggest two techniques to improve scalability of load store queue. (3 Points)

2. Would a good value predictor increase or decrease optimal pipeline depth? Why? (3 Points)

3. How would you modify gdb to work with Tomasulo without ROB? (3 Points)

4. A quatrain is a four-line stanza in a poem. An example quatrain is: (Upto 2 Points)

   There are things I don't understand
   And would really like to know
   Such as why they call it rush hour
   And you move so freakin' slow

   A couplet is similarly a two-line stanza:

   Tick tock...tick tock...
   Life is counting down on your internal clock.

   Write a couplet about ECE411, MP4 progress, or this exam. Write a quatrain instead for one more point.