Midterm 2

Name: ______________________________

Computing ID: __________

Pledge:
On my honor, I pledge that I have neither given nor received help on this assignment.

Signature: ______________________________

There are 12 questions with 17 parts worth 57 total points.

- Time limit: 75 minutes.
- No notes, books, or calculators. Scratch paper is allowed.
- Because the test is given in multiple locations, we cannot answer questions about the test during the test. If you find a question ambiguous or confusing, explain that on the page near the question and it will be considered during grading.
- Grading occurs in gradescope, which shows one question to the grader at a time. Keep your comments and answers near the question they apply to.
- You may use pencil or pen, and may mark answers in any way a human grader can easily understand.
- Please write clearly. Ambiguous answers, such as `Ŧ` when `T` or `F` was expected, are graded as wrong.
- Partial credit is awarded for answers that demonstrate partial understanding.
- Grading is not linear: earning 50% of the available points may result in more than a 50% grade on the test.
- Unless otherwise specified, questions that refer to pipelines assume the 5-stage pipeline from the textbook and your homeworks, namely:
    1. Fetch (with one instruction access)
    2. Decode (with two register reads)
    3. Execute (with one ALU operation)
    4. Memory (with one memory access)
    5. Writeback (with two register updates)
- For your reference, the Y86 instructions and their encoding as machine code are:

bit:	7 0	15 8	23 16	31 24	39 32	47 40	55 48	63 56	71 64	79 72
byte:	0	1	2	3	4	5	6	7	8	9
`halt`	0 0									
`nop`	1 0									
`rrmovq`/`cmovCC` rA, rB	2 cc	rA rB								
`irmovq` V, rB	3 0	F rB	V							
`rmmovq` rA, D(rB)	4 0	rA rB	D							
`mrmovq` D(rB), rA	5 0	rA rB	D							
`OPq` rA, rB	6 fn	rA rB								
`jCC` Dest	7 cc	Dest								
`call` Dest	8 0	Dest								
`ret`	9 0									
`pushq` rA	A 0	rA F								
`popq` rA	B 0	rA F								

We call the nyble that contains a number identifying the instruction above "`icode`" and the the nyble named `fn` and `CC` above "`ifun`".

Question 1 (5 pt)
Consider changing our 5-stage pipeline into a 6-stage pipeline by making one current stage into two stages. Which of the following changes would make two `pushq`s in a row into a data hazard? Assume full forwarding.
Select all that apply

- ☐ Fetch is two stages:
  F1 picks a PC to send to instruction memory,
  F2 gets the instruction and splits it into `icode`, `ifun`, and so on.
- ☐ Decode is two stages:
  D1 sends the register names to the register file,
  D2 gets the register values from the register file
- ☐ Execute is two stages:
  E1 sends the operation and operands to the ALU,
  E2 gets the result and sets the condition codes
- ☐ Memory is two stages:
  M1 sends the memory activity, address, and value to write to memory,
  M2 gets the value read from meory.
- ☐ Writeback is two stages:
  W1 prepares the register values for writing,
  W2 updates the register file and status

Question 2 (2 pt)
In the 5-stage pipeline conditional jumps need to be predicted to avoid stalling, but conditional moves do not. Which other conditional operations would require prediction to avoid stalling?
Select all that apply

- ☐ conditional `rmmovq`
- ☐ conditional `mrmovq`
- ☐ conditional `ret`
- ☐ conditional `pushq`

Question 4
Our textbook defined condition codes to be checked when the `jXX` or `cmovCC` is in the execute stage. Suppose we instead decided that comparing `ifun` to the condition code register was simple enough to include as part of our forwarding logic, so that we could check condition codes whenever the condition codes were set.

4.a. (2 pt) In our new forwarded-condition-check model, `subq` followed immediately by `jle`
Pick One

- ○ always requires stalling or prediction
- ○ only requires stalling or prediction for certain arguments to `subq`
- ○ never requires stalling or prediction

4.b. (6 pt) Which of the following instruction sequences will have a smaller misprediction penalty and/or fewer stalled stages with forwarded-condition-checks than with the check-during-execute?
Assume that the `subq` does change the condition codes in a way that changes whether the `le` condition is true.
Select all that apply

- ☐ `subq`, `jle`
- ☐ `subq`, `nop`, `jle`
- ☐ `subq`, `nop`, `nop`, `jle`
- ☐ `subq`, `cmovle`
- ☐ `subq`, `nop`, `cmovle`
- ☐ `subq`, `nop`, `nop`, `cmovle`

Question 3 (5 pt)
Assuming the five-stage pipeline with forwarding described in the textbook and implemented in your homework, fill in the rest of the pipeline diagram for the following program.
Rows are instructions and columns are cycles of operation. Leave a cell blank if the instruction is not in the pipeline that cycle.
Note that the program includes jumps, so the instructions are not listed in execution order; for example, we expect `halt` to be fetched in a later cycle than `xorq` is fetched.
There are more columns than you will need.

`irmovq $12,%r8`	F	D	E	M	W															
`mrmovq 0(%r8),%r9`																				
`subq %r9,%r9`																				
`call f`																				
`halt`																				
`f:`																				
`xorq %rax,%rax`																				
`ret`																				

Question 5
These questions ask about four possible new instructions we could add to Y86-64:

Instruction	Behavior
`immovq V, D(rB)`	moves an immediate into memory
`mmmovq C(rA), D(rB)`	moves a value from one memory address to another
`stackswap`	swaps the top two values in the stack so that "`pushq A`; `pushq B`; `stackswap`" is equivalent to "`pushq B`; `pushq A`"
`xchg rA, rB`	swaps the values in two registers

5.a. (4 pt) In the single-stage sequential processor, all of the instructions in Y86-64 could work by running through (a subset of) five stages in order:

1. Fetch (with one instruction access)
2. Decode (with two register reads)
3. Execute (with one ALU)
4. Memory (with one memory access)
5. Writeback (with two register updates)

Which of the new instructions **can** also be implemented by (a subset of) those same five stages in that same order?
Select all that apply

- ☐ `immovq V, D(rB)`
- ☐ `mmmovq C(rA), D(rB)`
- ☐ `stackswap`
- ☐ `xchg rA, rB`

5.b. (4 pt) How many bytes would the machine code encoding for each of these new instructions be?

`immovq`: ____ bytes

`mmmovq`: ____ bytes

`stackswap`: ____ bytes

`xchng`: ____ bytes

Question 6 (2 pt)
If I can achieve only one of the following options for my processor, which one will reduce the runtime of common software the most?
Pick One

- ○ higher latency
- ○ higher throughput
- ○ lower latency
- ○ lower throughput

Question 7 (3 pt)
Suppose the memory stage can run in anywhere between 1 and 200 cycles, depending on cache performance. It does this by providing a 1-bit "miss" output that is `0` if data is available and `1` if it is not.

On a miss, we want to let later instructions bypass the memory operation, completing out-of-order. Ignoring possible complications to the forwarding logic and precise exceptions, which of the following could safely pass a cache-missing `mrmovq 16(%rax), %rbx`?
Select all that apply

- ☐ `addq %rax, %rax`
- ☐ `addq %rbx, %rbx`
- ☐ `irmovq $30, %rbx`
- ☐ `ret`

Question 8
In the following diagrams, indicate the control signals to give each pipeline register by putting a single letter in each box; use N for normal, B for bubble, and S for stall.

8.a. (3 pt) Assume that i_4 resulted from an incorrect speculation and should be squashed, but all other instructions are OK and may continue to execute normally.

F		D		E		M		W	
☐	i_5	☐	i_4	☐	i_3	☐	i_2	☐	i_1

8.b. (3 pt) Assume that i_4 requires some information that is not yet available and needs another cycle in the D stage, but all other instructions are OK and may continue to execute normally.

F		D		E		M		W	
☐	i_5	☐	i_4	☐	i_3	☐	i_2	☐	i_1

Question 9 (2 pt)
Suppose that instead of a single-issue 5-stage pipeline we had 5-issue single-stage processor. The clock cycle would be roughly 5× slower, but potentially 5 instructions could finish each cycle.

Suppose we run the same real-world program on both processors, getting runtimes of i for the 5-issue single-stage processor and p for the single-issue 5-stage pipelined processor. We'd expect
Pick One

- ○ $2 < \frac{i}{p}$, meaning 5-stage is much faster
- ○ $1 < \frac{i}{p} < 1.5$, meaning 5-stage is a little faster
- ○ $1 < \frac{p}{i} < 1.5$, meaning 5-issue is a little faster
- ○ $2 < \frac{p}{i}$, meaning 5-issue is much faster

Question 10
Consider the five-stage pipeline described in the textbook and implemented in your homework running the instruction sequence

```
addq %rax,%rcx      // 1
subq %rdx,%rax      // 2
cmovle %rcx,%rax    // 3
xorq %rcx,%rsi      // 4
andq %rcx,%rax      // 5
```

10.a. (2 pt) When the `cmovle` instruction is in the writeback stage, the operation that set the condition codes it is using to decide whether to write to `%rax` is
Pick One

- ○ in fetch
- ○ in decode
- ○ in execute
- ○ in memory
- ○ no longer in the pipeline

10.b. (5 pt) Which register values must be forwarded (instead of being read from the register file) for each instruction? Assume these are the only instructions in the program. If an instruction reads no values or reads all the values it uses from the register file, answer "`none`". If the answer depends on whether `cmovle` moves or not, add an astrisk next to the register in question like "`%r8*`".

Instruction 1: ______________________

Instruction 2: ______________________

Instruction 3: ______________________

Instruction 4: ______________________

Instruction 5: ______________________

10.c. (3 pt) If we made the program counter a program register `%rpc`, we could remove two instructions:

- `ret` would be just `popq %rpc`
- `jmp f` would be just `irmovq f, %rpc`

How would this redesign change the pipeline performance?
Select all that apply

- ☐ `popq %rpc` would require fewer stalls than `ret`
- ☐ `popq %rpc` would require more stalls than `ret`
- ☐ `irmovq f, %rpc` would require fewer stalls than `jmp f`
- ☐ `irmovq f, %rpc` would require more stalls than `jmp f`
- ☐ previously-OK instructions like `OPq` could now potentially cause a misprediction penalty
- ☐ previously-OK instruction sequences like pairs of `OPq`s could now potentially cause a data dependency-based stall

Question 11 (1 pt)
Compare hazards and dependencies
Pick One

- ○ All hazards are dependencies but not all dependencies are hazards
- ○ All dependencies are hazards but not all hazards are dependencies
- ○ All dependencies are hazards and all hazards are dependencies
- ○ None of the above

Question 12 (5 pt)
Perform register renaming for the following instruction sequence, crossing off used registers from the free queue and filling in the new entries of the register map.

Y86-64	Renamed
`addq %rax,%rcx`	________ + ________ → ________
`irmovq $3330,%rcx`	________ → ________
`rrmovq %rcx,%rax`	________ → ________

Free queue: `%x5`, `%x61`, `%x0`, `%x78`, `%x81`, `%x82`, `%x83`, `%x84`

Architectural	Hardware
`%rax`	%x20
`%rcx`	%x44