University of California at Berkeley
College of Engineering
Department of Electrical Engineering and Computer Sciences

EECS152/252A
Spring 2022

J. Wawrzynek
2/22/22

Midterm Exam 1

Name: ______________________________

Student ID number: ______________________________

You have until 11AM to take the exam.

This is a *closed-book, closed-notes* exam. Also no calculators, phones, pads, or laptops allowed.

Each question is marked with its number of points (one point per expected minute of time). Start by answering the easier questions then move on to the more difficult ones. You can use the backs of the pages to work out your answers. Neatly copy your answer to the allocated places. **Neatness counts.** We will deduct points if we need to work hard to understand your answer.

Write the student ID numbers of the person to your left and to your right on this page. If you are sitting on an aisle, just indicate "aisle" for left or right.

Left student ID number: ______________________________

Right student ID number: ______________________________

Before you turn in your exam, write your student ID number on all pages.

Monday 21st February, 2022 14:22

1. MICROPROGRAMMING [12 pts]

You are asked to write microcode for a new instruction called `SQUARE`, defined as follows:

`R[rd] = R[rs1] * R[rs1]`

This instruction will be implemented on the microcoded single-bus RISC-V machine that we introduced in Problem Set 1, see below.

Unfortunately, your ALU does not support a multiplication operation, but you can perform the square operation as a simple loop of consecutive additions, as follows:

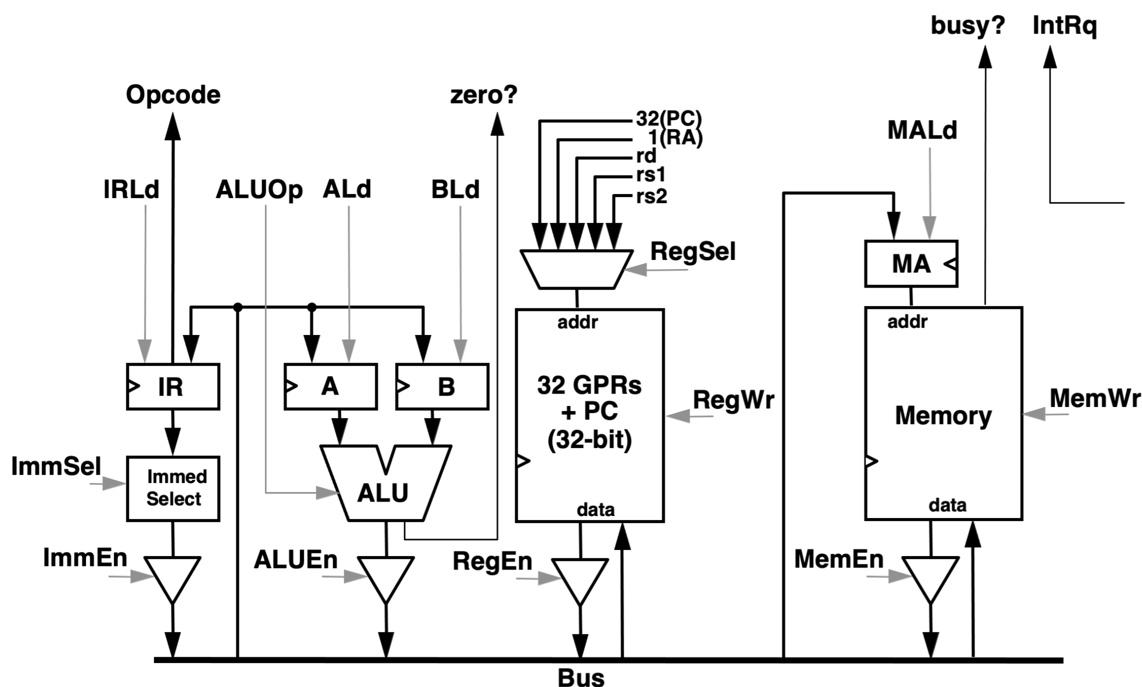
```
// Compute x*x
int result = x;
int count = x - 1;

while (count != 0) {
   result += x;
   count -= 1;
}
```

You are permitted to modify `R[rd]` and `R[rs2]`, but you should not modify any other architectural state. Also, assume that "x" ≥ 1.

Express your answer by filling in the microcode table below. Remember to mark "don't care" entries with a "*". Minimize the total number of micro-instructions you use.

Hint: One approach is to store "x" in `B`, temporary values for "count" in `R[rs2]`, and temporary values for "result" in `R[rd]`. (You are not required to follow this convention).

The microcoded machine is identical to the one we introduced in Problem Set 1. For your reference, we have reproduced the single-bus datapath, as well as information about the ALU and μbranch logic.



Here are our standard possible values for the μBR column:

N	next
J	jump
EZ	branch-if-equal-zero
NZ	branch-if-not-zero
D	dispatch
S	spin on `busy?` signal

Finally, here are the operations the ALU supports:

ALUOp	**ALU Result Output**
COPY_A	A
COPY_B	B
INC_A_1	A+1
DEC_A_1	A-1
INC_A_4	A+4
DEC_A_4	A-4
ADD	A+B
SUB	A-B
SLT	Signed(A) <Signed(B)
SLTU	A <B

State	PseudoCode	ldIR	Reg Sel	Reg Wr	en Reg	ldA	ldB	ALUOp	en ALU	ld MA	Mem Wr	en Mem	Imm Sel	en Imm	μBr	Next State
FETCH0:	MA <- PC; A <- PC	*	PC	0	1	1	*	*	0	1	0	0	*	0	N	*
	IR <- Mem	1	*	0	0	0	*	*	0	0	0	1	*	0	S	*
	PC <- A+4	0	PC	1	0	0	*	INC_A_4	1	*	0	0	*	0	D	*
. . .																
NOP0:	microbranch back to FETCH0	*	*	0	0	*	*	*	0	*	0	0	*	0	J	FETCH0
SQUARE0:																

2. Virtual Memory [19 pts]

(a) **Page Table Calculations** [3 pts]

Consider a computer with 256-byte pages, 16-bit virtual addresses, and 16-bit PTEs (4 bits of which are used for valid and protection). The computer uses *two-level hierarchical* page tables. Assume the machine is byte-addressable.

i. How many virtual pages can be addressed by this computer?

ii. What is the maximum size of the physical memory that can be supported by this computer?

iii. Suppose that a running program is currently using 300 bytes of memory. What is the smallest possible number of PTEs and PTPs that can be valid in the page table(s) of this program?

(b) **Memory Trace With Linear Page Table** [10 pts]

Suppose that a computer has 8-bit virtual addresses, 32-byte pages, 4-byte PTEs, a *linear (i.e. single-level)* page table, and a 4-entry TLB with LRU replacement policy.

At the beginning, the TLB is empty and the free page list is (in this order): 0x1, 0x6, 0x3, 0x5, 0xA.

The following virtual addresses are accessed by the programmer (in this order): 0x4A, 0x22, 0x15, 0x20, 0x81, 0x62.

Fill out the following tables:

- Table 1, describing the TLB hits, page fault, and physical addresses of different memory accesses.
- Table 2, describing the linear page table at the end of the memory access trace. (Assume the page table begins at address 0x0). Leave page table entries blank if they are invalid.
- Table 3, describing the TLB at the end of the memory access trace.

We have already filled in the beginnings of the tables for you.

Table 1: Memory Access Trace

Virtual Address	**Page table index**	**TLB hit/miss**	**Page hit/fault**	**Physical address**
0x4A	0x2	miss	hit	0x8A
0x22				
0x15				
0x20				
0x81				
0x62				

Table 2: Page Table Entries

Address	**PTE (excluding valid bit)**
0x00	
0x04	
0x08	0x4
0x0c	0x20
0x10	
0x14	
0x18	0x2
0x1C	

Table 3: TLB

VPN	0x2			
PPN	0x4			

(c) **Two-Level Page Table** [2 pts]

A computer has two-level page tables. All page tables (at either level) hold four 8-byte entries. Pages are 32 bytes long. Virtual addresses are 9 bits wide. Currently, a program is running with only one page allocated. This page has a VPN of 0xA and a PPN of 0x3. The base page table starts at address 0x0, and the second-level page table starts at address 0x20. Given this information:

i. Can we know, from the information above, which entries in the base page table are valid? If so, which entries are valid and what are the PTPs/PPNs stored in those entries?

ii. Can we know, from the information above, which entries in the second-level page table are valid? If so, which entries are valid and what are the PPNs stored in those entries?

(d) **Qualitative Short Question** [4 pts]

i. Suppose you increased the length of your virtual addresses. How would this affect the speed of your page table walker? Explain your reasoning. Assume that the page size and the page table size (in bytes) do not change.

ii. Under what conditions would you prefer a linear page table over a hierarchical page table?

iii. What is one advantage and one disadvantage of replacing a virtually-indexed, physically-tagged cache (VIPT) with a virtually-indexed, virtually-tagged cache (VIVT)?

3. Caches [16 pts]

(a) **Cache Sub-Blocks** [8 pts]

We will consider a cache optimization known as "sub-blocking" (also called "sectored caches"):

- The number of sets and ways is unchanged.
- Each cache block is broken up into smaller "sub-blocks".
  - Each sub-block has its own valid bit.
  - On a cache miss, only the cache sub-blocks accessed by the user's program are loaded in.
    - Other sub-blocks remain in the "invalid" state until they are also loaded in.
- Make sure you understand that "sets" are not "sub-blocks"!

Suppose that we have an 8 KB, two-way set associative cache with 4-word (16-byte) cache lines. The replacement policy is LRU. *Each cache block is broken up into four smaller sub blocks.*

We will evaluate the following two loops:

```
// Loop A
sum = 0;
for (int i = 0; i < 128; i++)
  for (int j = 0; j < 32; j++)
    sum += buf[i*32 + j];

// Loop B
sum = 0;
for (int j = 0; j < 32; j++)
  for (int i = 0; i < 128; i++)
    sum += buf[i * 32 + j];
```

i. What is the number of misses for Loop A and for Loop B with the sectored cache?

ii. What is the number of misses for Loop A and for Loop B if the cache is *not sectored* (i.e. no sub-blocks)?

iii. Qualitatively explain whether our sectoring scheme has improved the average memory access time of Loop A and Loop B. If so, why? If not, why not?

(b) **Memory Access Time Tradeoffs** [2 pts]

Suppose that we removed an outer-level cache to free up area on our chip. With this new area, we doubled the size of our L1 cache.

Suppose that this optimization worsened the L1 hit time from 1 cycle to two cycles, and increased the miss penalty from 50 cycles to 100 cycles. *Before* this optimization, the L1 miss rate was 10%.

What does the new miss rate have to be for our new optimization to improve the average L1 cache access time?

(c) **Short Qualitative Questions** [6 pts]

i. How do write buffers affect write miss penalties in a write-through cache? (A write miss occurs when the program attempts to STORE to an address that is not in the cache). Explain your reasoning.

ii. How do write buffers affect read miss penalties? (A read miss occurs when the program attempts to LOAD from an address that is not in the cache). Explain your reasoning.

iii. What advantages does adding a victim cache have over simply extending the size of an existing cache?

4. IRON LAW [6 pts]

(a) Suppose we could add complex instructions (like polynomial instructions or memory-to-memory instructions) to the RISC-V Base ISA. How would this affect processor performance and hardware complexity? Refer to elements of the Iron Law in your answer.

(b) Suppose we extended the RISC-V Base ISA with new 8-bit instructions. (RISC-V Base ISA instructions are 32 bits). Programmers would be permitted to freely mix 32-bit and 8-bit instructions in their code. How would this affect processor performance and hardware complexity? Refer to elements of the Iron Law in your answer.

5. PIPELINING [15 pts]

For this question, consider the six-stage pipeline below, designed for a *single-accumulator ISA*:

This is similar to the accumulator ISA described in Problem Set 1. Note the accumulator is read during the DECODE stage, and written to in the WRITEBACK stage. In this ISA, the only way to reference memory locations is with immediate offsets from the PC. There is no register-indirect addressing.

There are three types of instructions in this ISA: Arithmetic, Load/Store, and Branch instructions.

Arithmetic instructions load one operand from memory, apply it to the value in the accumulator, and store the result in the accumulator. E.g: ADD 112 (add the value stored in PC+112 to the accumulator).

Load/store instructions load or store the value in the accumulator to/from memory. E.g. Load 112 (load the value stored in PC+112 to the accumulator).

Branch instructions check if the accumulator is zero or not zero, and branch accordingly. Branches are resolved in the EXECUTE stage of the pipeline. In case of a mispredict, we flush the pipeline of earlier instructions. E.g: Branch-if-Zero 112 (branch to PC+112 if the accumulator value is 0).

(a) **Hazards** [8 pts]

i. Which bypass paths are necessary to minimize stalls caused by RAW hazards in this design?

ii. Are WAR, WAW, or control hazards possible in this design? If so, what hardware interlocks must be added to resolve those hazards?

(b) **Pipeline Diagram** [3 pts]

Consider the code below which runs on the pipelined CPU. Remember that all instructions use *PC-relative* offsets.

PC	**Instruction**	**Comment**
0	`Add 100`	// Result of Add is 0
4	`Store 100`	
8	`Load 96`	
12	`Branch-if-zero 8`	// This branch is taken
16	`Nop`	// This Nop is skipped over by the previous Branch-if-zero
20	`Add 80`	
...		
100	`0x0`	// This address stores data, rather than instructions
104	`0x0`	// This address also stores data, rather than instructions

Assume that the pipeline always predicts that branches are *not* taken. In other words, even though the Branch-if-zero above was taken, the CPU initially predicted that it was not taken.

Also, assume that the pipeline above is *fully bypassed.* This means that you can bypass results from the end of any stage to the end of any earlier stage.

Fill in this pipeline diagram of these instructions. The first row has been filled in for you:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
`Add 100`	F	D	A	M	E	W														
`Store 100`																				
`Load 96`																				
`Branch-if-zero 8`																				
`Add 80`																				

(c) **Exceptions** [4 pts]

Assume that branch, load, and store instructions can all cause exceptions. Exceptions may be detected in the Address-Gen, Memory, and Execute stages. Under these conditions, the pipeline above does *not* support precise exceptions, unless new hardware modifications are made.

i. Propose a hardware modification which would make exceptions precise for this pipeline.

ii. Suppose that the hardware designer refuses to add support for precise exceptions. What could the programmer do when running on this six-stage pipeline to eliminate the possibility of imprecise exceptions?