Full Name:_____

NetID:_____

Final Exam, Fall 2019 Date: 12/18

Instructions:

- Final exam takes 90 minutes. Read through all the problems and complete the easy ones first.
- This exam is **closed book**, except that you may bring a single doube-sided page of prepared note.

1 (xx/28)	2 (xx/25)	3 (xx/25)	4 (xx/22)	Bonus (xx/10)	Total (xxx/100+10)

1 Basic C and X86_64 machine instructions (28 points, 4 points each):

Circle *all* answers that apply. All questions in this section assume the x86-64 platform (Little Endian).

A. Given a binary sequence $(00000110)_2$, what is its decimal value?

1. `6`
2. `5`
3. `8`
4. `10`
5. None of the above.

B. Given a signed char -20, what is its binary representation?

1. $(1001\ 0100)_2$
2. $(1110\ 1011)_2$
3. $(1110\ 1100)_2$
4. $(0001\ 0100)_2$
5. None of the above

C. Suppose %eax contains signed `int` 255. After successfully executing `movl %eax, (%ecx)`, what is the *byte value* stored at the address given by %ecx? (See Appendix 5 to refresh on x86-64)

1. 0xff
2. 0x00
3. 0xf0
4. 0x0f
5. None of the above

D. Consider the following code snippet,

```
char *names[4] = {"C", "programming", "is", "hard"};
char **p;
p = names;
p = p + 2;
```

After executing the above, what is the value of `p[1][1]`?

1. '0'
2. 'r'
3. 'a'
4. 'p'
5. 's'
6. Undefined (or Segmentation fault)
7. None of the above

E. Consider the following code snipplet,

```
int a[3] = {1, 2, 3};
short *b;
b = (short *)a;
b++;
```

After executing the above, what is the value of `*b`?

1. 0
2. 1
3. 2
4. 3
5. Undefined (or Segmentation fault)
6. None of the above.

F. Which following expressions compute the remainder of x modulo 64 (x is of type `unsigned int`)?

1. `x % 64`
2. `x / 64`
3. `x >> 6`
4. `x & 0x0000003f`
5. `(x << 26) >> 26`
6. None of the above.

G. Which of the following statements are true w.r.t. `malloc`?

1. Every call to `malloc` results in the memory allocator making a syscall (e.g. `sbrk`) to request memory from OS.
2. `malloc` returns failure *if and only if* the memory allocator does not have any free chunks.
3. When using the implicit-list design, `malloc` tends to traverse more chunks than when using the explicit-list design.
4. None of the above.

2 Memory hierarchy

A. (4 points) What is the latency to access L1 cache and main memory, respectively? Do not forget to write down the time unit. (We consider the answer correct if it's within a factor of 10)

Answer:

L1 Cache	
Main memory	

The rest of the questions assume a cache whose total size is 1KB and each cache line/block is 64-byte.

B. (3 points) How many cacheline/blocks does the cache contain?

Answer. _____

C. (3 points) Suppose the cache is a direct mapped cache. Given a 32-bit address `0xab345f78`, which cacheline/block contains the cached data for this address? You should specify the index position of the cache line, which is an integer in the range [0, n-1] where n is your answer in B.

Answer. _____

D. (3 points) How many cacheline-aligned memory addresses can be mapped to the *same* cache location as `0xab345f78` (assuming 32-bit address space)?

Answer. _____

E. (3 points) If your answer in **D.** is bigger than 1, how can we determine which of the memory addresses is actually stored at a given cache location?

Answer. _____

F. (3 points) Suppose the cache is a 2-way associative cache, i.e. the cache is organized into sets each of which contains 2 cachelines. How many sets does the cache contain?

Answer. _____

G. (3 points) For the 2-way assocative cache, given a 32-bit address `0xab345f78`, which set may contain the cached data for this address? You should specify the index position of the set, which is an integer in the range [0, n-1] where n is your answer in **F.**

Answer. _____

H. (3 points) For the 2-way associactive cache, how many cacheline-aligned memory addresses can be mapped to the *same* set as `0xab345f78`?

Answer. _____

Question **I.** and **J.** are bonus questions.

The following C snippet uses a double loop to increment each element of a 2D array `long a[ROWS][COLS]`. Note that C stores a 2D array in memory in row major format, i.e. each row is stored contiguously. There are a total of ROWS*COLS 8-byte memory access in the following code.

```
for (int i = 0; i < ROWS; i++) {
  for (int j = 0; j < COLS; j++) {
    a[i*COLS+j]++;
  }
}
```

I. (Bonus 5 points) Suppose ROWS=16, COLS=16. Out of ROWS*COLS total 8-byte memory accesses, how many of them result in cache miss (assuming the cache is direct-mapped and it's empty before the start of the loop)?

Answer. _____

Suppose we change the loop order into the following:

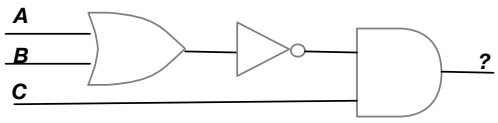
```
for (int j = 0; j < COLS; j++) {
  for (int i = 0; i < ROWS; i++) {
    a[i*COLS+j]++;
  }
}
```

J. (Bonus 5 points) Suppose ROWS=16, COLS=16. Out of ROWS*COLS total 8-byte memory accesses, how many of them result in cache miss (assuming the cache is direct mapped and it's empty before the start of the loop)?

Answer. _____

3 Logic design (25 points)

A. (5 points) What is the boolean expression that corresponds to what the following logic circuit outputs? (Please write the expression with the most direct correspondence to this circuit without simplification.)



Answer:

B. (5 points) Please simplify the boolean expression $(A + B) \cdot (\overline{A} + \overline{B})$ according to the boolean logic laws in Appendix 6. Write down the detailed steps where each step applies only one law and name that law.

C. (5 points) If we are to use a single logic gate to implement the simplified expression in **B.** Which gate to use?

1. AND
2. OR
3. NOR
4. NAND
5. XOR
6. None of the above.

Question **C.** and **D.** ask you to implement a combinatorial circuit to compute a 3-input parity function. A parity function takes multiple 1-bit input signals and computes a 1-bit parity bit as the output. The parity bit is the sum of all input signals modulo 2. For example, if the 3 input signals are 110, the parity output is `(1+1+0)%2 = 0`. If the 3 input signals are 111, the parity output is `(1+1+1)%2 = 1`.

C. (5 points) Please write the truth table for the 3-input parity function.

D. (5 points) Please draw the combinatorial circuit that implements the above parity function.

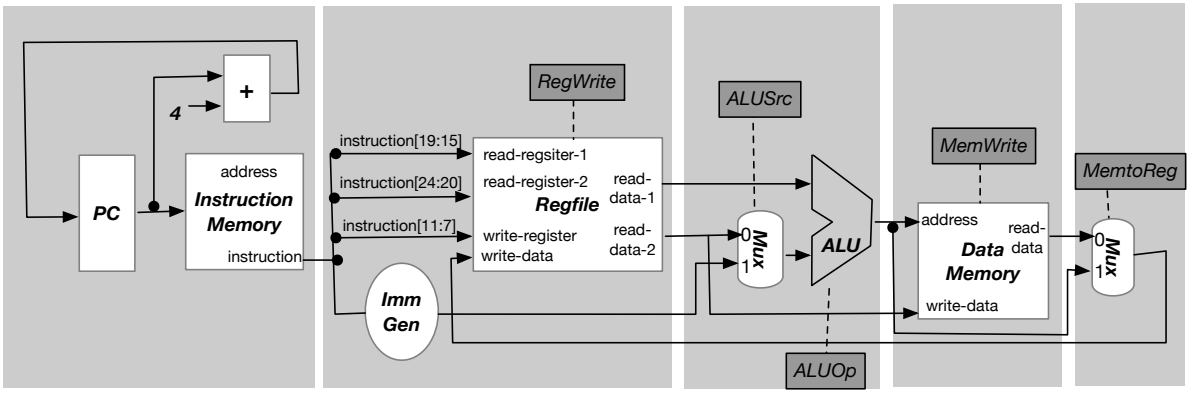


Figure 1: Basic RISC-V CPU design

4 CPU design (22 points)

Figure 1 shows a basic *single-cycle* CPU design for RISC-V. The design handles 3 types of instructions: arithmetic instructions involving two source registers and one destination register, memory load instructions and store instructions. Note that the design does not implement conditional branches.

Suppose the major logic blocks in Figure 1 have the following latencies (If a logic block's latency is not specified, it is small enough to be ignored.)

Read from instruction memory	400ps
Read from regfile	150ps
Write to regfile	150ps
ALU	250ps
Read/write to data memory	400ps

A. (6 points) What is the minimum latency to execute each of the following instructions in the single-cycle design:

Instruction	Instruction meaning	Latency
add x1, x2, x3	x1 = x2+x3	
sd x1, 16(x2)	Memory[x2+16] = x1	
ld x1, 16(x2)	x1 = Memory[x2+16]	

B. (4 points) What is fastest clock cycle time that can be used for the single-cycle CPU design?

Answer: _____

C. (4 points) Suppose we change the CPU in Figure 1 into 5-stage pipelined design: IF(instruction fetch), ID(instruction decode and register read), EX(execute operation or calculate address), MEM(memory access), WB(write back to regfile). The five stages of the pipeline are marked as the large grey rectangles in Figure 1. What is the fastest clock cycle time that can be used for the 5-stage design?

Answer: _____

Question **E.** and **F.** assume the following two instruction sequence:

```
ld x1, 16(x2) // i.e. x1 = Memory[x2+16]
add x5, x1, x4 // i.e. x5 = x1+x4
```

E. (4 points) Suppose there is no forwarding between pipeline stages, how many bubbles (aka *nop*) must be inserted between the two instructions to ensure correctness? (Note that we assume the regfile can perform both read write in a single cycle where the read returns the value of the write)

Answer: _____

F. (4 points) *Suppose the result of MEM stage is forwarded to the EX stage*, how many bubbles (aka *nop*) must be inserted between the two instructions to ensure correctness?

Answer: _____

— END OF EXAM —

5 Appedix: X86 Cheatsheet

5.1 Registers

x86 registers are 8-bytes. Additionally, the lower order bytes of these registers can be independently accessed as 4-byte, 2-byte, or 1-byte register. The register names are:

8-byte register	Bytes 0-3	Bytes 0-1	Byte 0 (lowest order byte)
%rax	%eax	%ax	%al
%rbx	%ebx	%bx	%bl
%rcx	%ecx	%cx	%cl
%rsi	%esx	%si	%sil
%rdi	%edi	%di	%dil

...the rest is omitted...

5.2 Instructions

Instruction convention: We use the assembly convention that puts source operand before destination operand. e.g. mov S, D copies data from S the source operand to D the destination operand.

Instruction suffixes:

”byte” (`b`)	1-byte
”word” (`w`)	2-bytes
”doubleword” (`l`)	4-bytes
”quardword” (`q`)	8-bytes

Complete memory addressing mode: A memory operand of the form `D(Rb, Ri, S)` accesses memory at address `D + val(Rb) + val(Ri)*S`, where val(Rb) and val(Ri) refer to the value of registers Rb and Ri respectively, D is a constant, and S is a constant of value 1, 2, 4, or 8.

6 Appedix: Boolean Law

Identity law:	$A + 0 = A$	$A \cdot 1 = A$
Zero and One laws:	$A + 1 = 1$	$A \cdot 0 = 0$
Inverse Laws:	$A + \overline{A} = 1$	$A \cdot \overline{A} = 0$
Commutative laws:	$A + B = B + A$	$A \cdot B = B \cdot A$
Associative laws:	$A + (B + C) = (A + B) + C$	$A \cdot (B \cdot C) = (A \cdot B) \cdot C$
Distributaive laws:	$A \cdot (B + C) = (A \cdot B) + (A \cdot C)$	$A + (B \cdot C) = (A + B) \cdot (A + C)$
DeMorgan’s theorems:	$\overline{(A + B)} = \overline{A} \cdot \overline{B}$	$\overline{A \cdot B} = \overline{A} + \overline{B}$