Wawrzynek & Weaver CS 61C Sp 2018 Great Ideas in Computer Architecture Final Exam

PRINT your name: _____, _____
(last) (first)

I am aware of the Berkeley Campus Code of Student Conduct and acknowledge that any academic misconduct on this exam will be reported to the Center for Student Conduct and may lead to a "F"-grade for the course. I am aware that Nick believes *in retribution.*

SIGN your name: _____

PRINT your class account login: `cs61c-`_____ and SID: _____

Your TA's name: _____

Number of exam of person to your left: _____

Number of exam of person to your right: _____

You may consult four sheets of notes (each double-sided). You may not consult other notes, textbooks, etc. Calculators, computers, and other electronic devices are not permitted. Please write your answers in the spaces provided in the test.

You have 170 minutes. There are 13 questions, of varying credit (180 points total). The questions are of varying difficulty, so avoid spending too long on any one question. Parts of the exam will be graded automatically by scanning the **bubbles you fill in**, so please do your best to fill them in somewhat completely. Don't worry—if something goes wrong with the scanning, you'll have a chance to correct it during the regrade period.

If you have a question, raise your hand, and when an instructor motions to you, come to them to ask the question.

Do not turn this page until your instructor tells you to do so.

Question:	1	2	3	4	5	6	7	8	9	10	11	12	13	Total
Points:	15	10	14	14	9	15	18	10	13	18	16	20	8	180

Exam Clarifications

Q1: Part b)ii) Question should say: "How many valid representations can we represent"
d) Do not use the pow() function or include math.h for this problem
Q2(b): new-¿unit_price etc should be blue_monday-¿unit_price etc..
Q5: Non-pipelined circuit should be labeled "Figure 1", Pipelined circuit should be labeled "Figure 2"
Q7 (d)(iii) "Overall AMAT of Loop 1 2 only"
Q9: Representation should say 10 significand bits, not 11 bits.
b) Question should read: "What is the smallest positive non-zero number..."
Q10(c) lines 3 and 4 of the RISC-V code should be srli... slli...
line 13 should be "add t1, t1, _____"
Q12: In the problem statement, it says the TLB is 4-way Fully Associative. This should be JUST Fully Associative (no 4-way).
Q12: First paragraph, 16KB pages should be 16KiB pages
c) M is the least significant bit, not N twice Mystery Sum: 2nd for loop should have bound NITER not NITIR

Problem 1 ***[MT1-1] Number Rep*** **(15 points)**

Answer the following questions about number representation:

(a) Unsigned Base 4

(i) What is the range that a 4 digit unsigned base 4 number can represent? Write the bounds in decimal.

Solution: $0000_4 \sim 3333_4 = 0_{10} \sim 255_{10}$

(ii) Convert 107_{10} to unsigned base 4.

Solution: $107_{10} = 64 + 16 * 2 + 4 * 2 + 3 = 1223_4$

(b) Signed Base 4

(i) Suppose we wanted to use a bias in order to represent negative numbers in base4. If we are working with a 4 digit base 4 number, what should we choose as our bias? (Our bias should create equal amounts of negative and positive numbers for our range. If this is not possible, select a bias that will result in 1 more negative number than positive numbers). Express your answer in decimal.

Solution: 255/2 = 127. So the bias is -128 to favor negative numbers.

(ii) Suppose rather than using a bias notation, we decide to do the following.

For each base 4 number, we will reserve the most significant digit to strictly be used as a sign bit. A digit value of 1 will indicate a negative number, and a digit value of 0 will indicate a positive number. Any other values will result in an invalid number. For instance:

$$0003_4 = +3 \qquad 1003_4 = -3 \qquad 2003_4 = Invalid$$

How many valid representation can we represent with a 4 digit base 4 number using this scheme?

Solution: 2*4*4*4 = 128

(c) Given the following function in C:

```
int shifter(int x, int shift) {
    if (x > 0) {
        return x >> shift;
    }
    return -1 * (x >> shift);
}
```

Given y is a negative integer, and that `shifter(y, 2)` outputs 4, what is the range of values of y?

hint: `-8 >> 1 = -4`

Solution: -16 ~ -13

(d) Implement the function `unsigned int base_convert(unsigned int num, unsigned int base)`. This function takes in non-negative integers `num` and `base`. You are guaranteed the following:
- `base` is an integer in the range [2, 10], no need to error check this
- `num` is comprised of "digits" with a value between 0 and `base` - 1.
- All values fit inside an `unsigned int`.

Your job is to make it so the function returns the decimal value of `num` in base `base`. For example, `base_convert(30, 4)` would return 12, since 30_4 is 12_{10}. You may not use additional lines (do not put multiple lines on the same line via ;) but you may not need all the lines provided. In addition, you may not include `<math.h>` or use `pow()`.

Solution:

```
unsigned int base_convert(unsigned int num, unsigned int base) {
    unsigned int value = 0, power = 1;
    while (num > 0) {
        value += (num % 10) * power;
        power *= base;
        num /= 10;
    }
    return value;
}
```

Problem 2 ***[MT1-2] Allocating an Order*** **(10 points)**

You are working on an e-commerce platform. Internally, orders are tracked through a struct called `order_t`. Your task is to write a function to allocate and initialize a new order. There's a catch though! This platform must be robust to errors, so you are required to return an error value from this function in addition to the newly allocated order. The possible errors are defined for you as preprocessor directives.

(a) **Write** `new_order`: Fill in the following code. Keep in mind the following requirements:

- You must return `BAD_ARG` if any inputs are invalid. The criteria for valid arguments is:
  - Unit price should be positive (no negative prices)
  - An order cannot be for more than `MAX_ORDER` items
  - Inputs must not cause your function to crash (or execute undefined behavior)
- You must return `NO_MEM` if there are any errors while allocating memory
- The tax rate is always initialized to `TAX_RATE`
- If your function returns `OK`, then `new` points to a valid struct that has been initialized with the provided values.

```
typedef struct order {
    int quantity;
    double unit_price;
    double tax_rate;
} order_t;

#define OK 0          /* Function executed correctly */
#define NO_MEM 1     /* Could not allocate memory for order */
#define BAD_ARG 2     /* An invalid argument was given */

#define TAX_RATE 1.08
#define MAX_ORDER 100
```

Solution:

```
/* Allocate and initialize a new order */
int new_order(order_t **new, int quantity, double unit_price) {
     /* Validate Arguments */
     if (new == NULL || quantity > MAX_ORDER || unit_price < 0)
          return BAD_ARG;

     /* Allocate "new" */
     *new = malloc(sizeof(order_t));
     if(*new == NULL) {
          return NO_MEM;

     /* Initialize "new" */
     (*new)->unit_price = unit_price;
     (*new)->quantity = quantity;
     (*new)->tax_rate = TAX_RATE;

     return OK;
}
```

(b) **Calling** `new_order`: How would you use `new_order()` to allocate and initialize `blue_monday` with a quantity of 10 and a unit price of 3.50 in the example below?

Solution:

```
order_t *blue_monday;
double total;
ret_t ret;

/* Fill in the arguments to new_order here */

ret = new_order(&blue_monday, 10, 3.50);

if (ret == OK) {
     printf("Total:  %lf\n",
               (blue_monday->unit_price *
                    blue_monday->quantity * blue_monday->tax_rate));
} else {
     printf("Error\n");
}
```

Problem 3 ***[MT1-3] RISCY*** **(14 points)**

The function RISCY is known to take in two arguments, in `a0` and `a1`.

(a) Fill in the blanks such that the code below executes properly and evokes `ecall` to print the value in register `s1`. You may assume that `ecall` is a function that takes in two arguments `a0` and `a1`. When `a0` is 1, it prints the value in register `a1`.

Solution:

```
RISCY: # Prologue
          addi sp, sp, -20
          sw s0, 0(sp)
          sw s1, 4(sp)
          sw ra, 8(sp)
          addi s0, x0, 1
          add s1, x0, x0
Loop:     addi a0, a0, 4
          beq a1, s0, Ret
          lw t1, 4(a0)
          lw t2, 0(a0)
          sub t1, t1, t2
          bge t1, x0, Cont
          neg t1, t1
Cont:     blt t1, s1, next
          mv s1, t1
next:  # print value in s1 for debugging purpose.
          sw a0, 12(sp)
          sw a1, 16(sp)
          addi a0, x0, 1
          mv a1, s1
          ecall # ecall takes in a0(=1 for print) and a1(=register to
print)
          lw a0, 12(sp)
          lw a1, 16(sp)
          addi s0, s0, 1
          j Loop
Ret       mv a0, s1
          # Epilogue
          lw s0, 0(sp)
          lw s1, 4(sp)
          lw ra, 8(sp)
          addi sp, sp, 20
          jr ra
```

(b) Convert the RISCV instruction `bge t1, x0, Cont` into machine code in **binary**.

Assume `mv` and `neg` expands to one instruction. Express your answer in **binary** in the fields below.

Solution: 0 000000 00000 00110 101 0100 0 1100011
Binary: 0b00000000000000011010101000 1100011
Hex: 0x00035463

(c) Translate `RISCY` into C code. You may or may not need all of the lines provided below. You can assume you have access to a new print function `printint` which takes in one argument, an integer, and prints it out:

```
void printint(int x);
```

Solution:

```
int RISCY(int* a0, int a1) {
     int max_diff = 0;
     for (int i = 1; i < a1; i++) {
          int diff = a0[i] - a0[i-1];
          if (diff < 0)
               diff = - diff;
          if (diff >= max_diff)
               max_diff = diff;
          printint(max_diff);
     }

     return max_diff; }
```

Problem 4 ***[MT1-4] CALL*** **(14 points)**

Consider the following C code and assembly code:

```
#include <stdio.h>

int main() {
     int i, sum = 0;
     for (i = 100; i !=0; i--)
          sum = sum + i * i;
     printf ("The sum of sq from 100 ..  1 is %d\n", sum);
}
```

Address	Assembly
	.data
	str:
0x80	.string "The sum of sq from 100 .. 1 is %d\n"
	.text
	main:
0x00	addi sp, sp, -4
0x04	sw ra, 0(sp)
0x08	mv a1, x0
0x0c	li t1, 100
0x10	j check
	loop:
0x14	mul t2, t1, t1
0x18	add a1, a1, t2
0x1c	addi t1, t1, -1
	check:
0x20	bnez t1 loop
0x24	la a0, str
0x28	jal printf
0x2c	mv a0, x0
0x30	lw ra, 0(sp)
0x34	addi sp, sp 4
0x38	ret

Figure 1: Assembly Code with Address

(a) Please fill in all lines in the above assembly code.

(b) How many pseudo-instructions are in the given assembly code? Count each occurrence as one pseudo-instruction.

- ○ 4
- ○ 5
- ○ 6
- ○ 7
- ● 8
- ○ 9

Solution: Pseudo-instructions are highlighted in the code above.

(c) Create the symbol table and relocation table.

Solution:

Label	Address
`main`	0x00
loop	0x14
check	0x20
str	0x80

Instruction	Address	Dependency
`la a0, str`	0x24	`str`
jal printf	0x28	printf

(d) Replace the labels of PC-relative targets with their immediate values. What is the offset value of `bnez` at address `0x20`? Write your answer in decimal.

Solution: -12

(e) The assembler takes two passes over the code to resolve PC-Relative target addresses.

- ● True
- ○ False

(f) The absolute target addresses can be resolved at the assembler stage.

- ○ True
- ● False

(g) Interpreted code should run faster than the compiled code.

○ True ● False

(h) The compiler still needs to go through the linker stage even if we only have 1 source file to compile.

● True ○ False

Problem 5 ***[MT2-1] Circuits and Timing*** **(9 points)**

In this question, you will be working with a circuit that takes in three 8-bit inputs. For all parts, assume the delays below:

$t_{clk-to-q} = 3\text{ps}, \quad t_{setup} = 4\text{ps}, \quad t_{shifter} = 1\text{ps}$
$t_{adder} = 5\text{ps}, \quad t_{multiplier} = 6\text{ps}, \quad t_{subtracter} = 4\text{ps}$

Furthermore, assume that the inputs A, B, and C take on their new values exactly at the rising edge of every clock cycle and that all registers are initialized to zero.

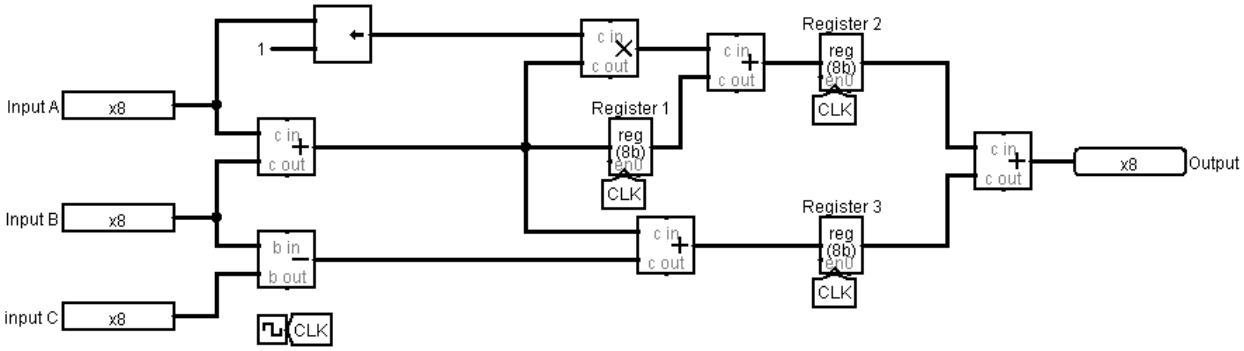

Figure 2: Non-pipelined circuit

(a) What is the maximum possible hold time that still ensures the correctness of the non-pipelined circuit in figure 2? (Select only one)

- ○ 1ps
- ● 5ps
- ○ 3ps
- ○ 7ps
- ○ 4ps

Solution: 5ps (Inputs A/B modify input to register 1 5ps after rising edge)

(b) What is the minimum possible clock period that still ensures the correctness of the non-pipelined circuit in figure 2? You may assume that for this question that all flip-flops have a 0ps hold time requirement. (Select only one)

- ○ 13ps
- ● 20ps
- ○ 16ps
- ○ 23ps

Solution: 20ps (Input B to Register 2)

Now consider the pipelined version of the circuit (shown below). You will be using this circuit for the remaining part of the question. All delays remain the same. You may assume that the hold time is 0ps for the following questions.

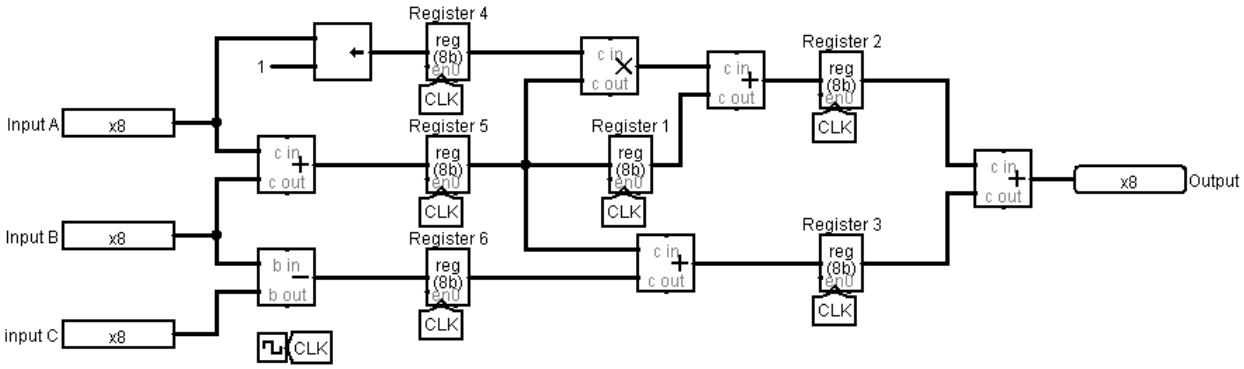

Figure 3: Pipelined circuit

(c) What is the minimum clock period of the pipelined circuit in figure 3 that maintains the circuit's correctness?

- ○ 10ps
- ○ 15ps
- ○ 13ps
- ● 18ps

Solution: 18ps (Register 4/5 to Register 2)

(d) How long does it take to compute the output for a given set of inputs? Assume the clock period is 11ps.

- ○ 22ps
- ○ 42ps
- ○ 27ps
- ○ 47ps
- ○ 28ps
- ○ 50ps
- ○ 31ps
- ● other: 30ps or 41 ps

Solution: Technically, 30ps is the time it will take for the values of one set of inputs to propogate to the output, We also accepted 41ps (3 clock cycles + clk-to-q + adder)

Problem 6 *[M2-2] Read and Write* **(15 points)**

Recall in class we learned that we can optimize our CPU pipeline by having register writes then reads within the same cycle. Let's call this implementation **write-read**.

Consider a new implementation where register reads happen before register writes within the same cycle. Let's call this implementation **read-write**.

Now consider the following RISC-V code and answer the following questions about a 5-stage RISC-V pipeline. **Assume no forwarding and no branch prediction**.

You are given that there needs to be at least one stall after line 4 for both implementations.

	loop:
1	slli t0 a1 2
2	or t2 a1 t1
3	add t0 t0 a0
4	lw t1 4(t0)
5	beq t1 x0 loop
6	addi t2 t2 5
7	sw t2 8(t0)
8	add a0 t2 x0

(a) Consider the code above and the **write-read** implementation. Which lines should be followed by a stall to guarantee correctness? (You are given that there needs to be at least one stall after line 4). For example, if an instruction on line A causes an instruction on line B to stall, bubble A.

- [x] 1
- [ ] 2
- [x] 3
- [x] 4
- [x] 5
- [ ] 6
- [ ] 7
- [ ] 8

Solution: line 1, 3, 4 (given), 5

(b) Still considering the **write-read** implementation, how many stalls are needed before the instruction on line 5 executes (do not include any stalls that occur after this

point)?

Solution: 3

(c) Now consider the **read-write** implementation, how many stalls are needed before the instruction on line 5 executes (do not include any stalls that occur after this point)?

Solution: 5

(d) Recall that you are given that there needs to be at least one stall after line 4 for both implementations. What type of hazard requires us to need at least one stall after line 4?

- ○ Structural
- ○ Control
- ● Data

(e) For **write-read**, how many stalls do we need between lines 4 and 5?

- ○ 1
- ○ 3
- ● 2
- ○ 4

(f) For **read-write**, how many nops do we need between lines 4 and 5?

- ○ 1
- ● 3
- ○ 2
- ○ 4

(g) If we decide to reorder instructions, which instruction is the best choice to replace a nop after line 4? Choose the line number of that instruction.

- ○ 1
- ○ 5
- ○ 2
- ○ 6
- ○ 3
- ○ 7
- ○ 4
- ● None - no reordering needed

Problem 7 *[M2-3] $$$* **(18 points)**

Assume we have a single L1 data cache having the following characteristics:

- 4 KiB cache size
- 16 byte blocks
- Direct Mapped

Assume the following piece of code is run in a 32-bit address space with `sizeof(int) = 4` :

```
#define SIZE 8192 // 2^13

int ARRAY[SIZE]; // note:  extra aligned:  ((int) ARRAY) % 64 == 0

int main() {
     ARRAY[0] = ARRAY[4] + ARRAY[8]; // This happens before Loop 1

     for (int i = 0; i < SIZE - 16; i += 4) { // Loop 1
          ARRAY[i] += ARRAY[i + 4] + ARRAY[i + 8] + ARRAY[i + 12];
     }

     for (int i = SIZE - 1; i >= 0; i -= 32) { // Loop 2
          ARRAY[i] += 10;
     }
}
```

(a) Calculate the number of tag, index, and offset bits for this L1 cache.

Solution: Tag: 20 Index: 8 Offset: 4

(b) Now, what is the hit rate for Loop 1 in the data cache? Assume that we start with a cold cache from the start of `main()`.

Solution: 4/5
We loaded in the first three block before Loop 1. Now, our step size is 4 ints or 16 bytes, meaning we get 1 step per block. Each step, we read the array four times and write into it once, resulting in 5 accesses, however, each of the reads is in a different block, resulting in 4 distinct blocks being used per step. The first three are brought in already, and we miss on the fourth read. The write is in the same block as the first read, so we have a 4/5 hit rate.

(c) What is the hit rate for Loop 2 given that the cache is NOT reset after Loop 1?

Solution: 9/16
We know that our cache is $4KiB = 2^{12}$ B large and our array is $2^{13} * 2^2$ B large, meaning that at the end of Loop 1, because we touched every single block, the last 1/8th of the array is in our cache. Now, for the first 1/8th of Loop 2, we reuse the blocks already in our cache, meaning we have a 100% hit rate for 1/8th, but 50% hit rate for the rest of the 7/8th accesses, 50% since there are two accesses per step into the same block. Thus, we get a hit rate of $1/8 + 1/2 * 7/8 = 9/16$.

(d) Assume that accessing memory takes 100 cycles, accessing data that is in the cache takes 5 cycles, Also assume for this part that Loop 1's hit rate is 60% and Loop 2's hit rate is 75%, which may or may not be the correct hit rates. What is the average memory access time (AMAT) **in cycles** for (Please reduce fractions):

(i) Loop 1:

Solution: 45 cycles. AMAT = HT + MR * MP = 5 + 40% * 100 = 45

(ii) Loop 2:

Solution: 30 cycles. AMAT = HT + MR * MP = 5 + 25% * 100 = 30

(iii) Overall AMAT of Loop 1 2 (an expression of REDUCED fractions is alright. You may use "T1" as the Loop 1 AMAT and "T2" as the Loop 2 AMAT in your calculation of this value):

Solution: 20/21 * T1 + 1/21 * T2
The first loop has 5 accesses per step and $2^{13}/4$ total steps. The second loop has 2 accesses per step and $2^{13}/32$ total steps. This gets that the first loop does $5 * 2^{11}$ accesses while the second loop does $2 * 2^8$ or 2^9 accesses. The weighted average of AMAT then becomes: $T1 * (5 * 2^{11})/(5 * 2^{11} + 2^9) + T2 * (2^9)/(5 * 2^{11} + 2^9) = T1 * (5 * 2^2)/(5 * 2^2 + 1) + T2 * 1/(5 * 2^2 + 1) = T1 * 20/21 + T2 * 1/21$.

Now we add in a L2 cache with the following characteristics:
- 16 KiB cache size
- 16 byte blocks
- Fully Associative

We re-run `main()` with cold L1 and L2 caches.

(e) What would be the local MISS rate of the L2 cache for Loop 1?

Solution: 100%.
We get no hits because we only access L2 if we miss in L1, but after the first miss in L1 per step, we never access L2 since we hit in L1 for the rest of the accesses, resulting in one miss and no hits per step.

(f) What would be the local MISS rate of the L2 cache for Loop 2? Assume the caches are NOT reset after Loop 1. Also, don't take into account the miss rate for Loop 1 when calculating Loop 2.

Solution: 4/7.
Our L2 cache is $16KiB = 2^{14}$ B while our array is 2^{15} B, which means that at the end of Loop 1, starting from the end of the array, we have half of it. The first 1/8th is already in L1, so we don't access L2, but for the next 7/8th we use L2. L2 has 3/8th the cache, so (3/8)/(7/8) = 3/7 HR = 4/7 Miss Rate.

Problem 8 ***[M2-4] Datapath*** **(10 points)**

Recall the standard 5-stage, single cycle datapath contains stages for Instruction Fetch, Decode, Execute (ALU), Memory, and Write-back. Datapath designers are interested in reducing the phases necessary for execution such that instead of accessing both the Execute (ALU) phase and the Memory phase, instructions access either one or the other, but not both. This would create a 4-stage, single cycle datapath with the following stages: Instruction Fetch, Decode, Execute OR Memory, and Write-back.

Instr Fetch	Instr Decode	Execute (ALU)	Memory	Write-back
100ps	150ps	200ps	350ps	150ps

(a) Given the table above and the described datapath above, what is the time it takes for a single instruction that utilizes all stages to execute on the typical 5-stage, single cycle datapath?

Solution: $100 + 150 + 200 + 350 + 150 = 950$ps

(b) What is the time it takes for a single instruction that utilizes all stages to execute on the new 4-stage, single cycle datapath?

Solution: $100 + 150 + MAX(200, 350) + 150 = 750$ps

(c) If the designers go ahead with this modification, which instructions will NOT function correctly? Why? Please limit your answer to two sentences or less.

Solution: Load and store instructions which use a non-zero offset will not function correctly because they require BOTH the ALU and Memory phase. The address must first be calculated before memory can be accessed.

(d) Propose a program-level modification that will fix the issue. Do NOT propose a modification to the datapath. Please describe your modification in two sentences or less.

Solution: Instead of allowing load/store instructions with non-zero offsets to appear in code, the compiler (or the programmer) can expand these instructions into a load/store + addi pair. This way, the address is calculated by a separate instruction before the memory access takes place.

Problem 9 ***[F-1] Floating Point*** **(13 points)**

IEEE 754-2008 introduces half precision, which is a binary floating-point representation that uses 16 bits: 1 sign bit, 5 exponent bits (with a bias of 15) and 10 significand bits. This format uses the same rules for special numbers that IEEE754 uses. Considering this half-precision floating point format, answer the following questions:

(a) For 16-bit half-precision floating point, how many different valid representations are there for NaN?

Solution: $2^{11} - 2$

(b) What is the smallest positive non-zero number it can represent? You can leave your answer as an expression.

Solution: bias $= 2^{5-1} - 1 = 2^4 - 1 = 15$
Binary representation is: 0 00000 0000000001
$= 2^{-14} * 2^{-10} = 2^{-24}$

(c) What is the largest non-infinite number it can represent? You can leave your answer as an expression.

Solution: Binary representation is: 0 11110 1111111111
$= 2^{16} - 2^5 = 65504$

(d) How many floating point numbers are in the interval [1, 2) (including 1 but excluding 2)?

Solution: 2^{10}
0b0 01111 0000000000 = 1
0b0 10000 0000000000 = 2
There are 2^{10} numbers within the interval.

Problem 10 ***[F-2] All Kinds of Parallelism*** **(18 points)**

SIMD within a Register (SWAR) in RISCV: You are planning to obfuscate some messages before they get released to the world. Instead of doing it properly (via encryption), you want a simpler implementation. Your first idea is to add 1 to each character, e.g. turning "aabb" into "bbcc"? If we apply this method to "a quick brown fox jumps over the lazy dog", it becomes "b!rvjdl!cspxo!gpy!kvnqt!pwfs!uif!mb{z!eph"! It looks promising. And the implementation is plain and simple (both in C and RISCV):

```
void obfuscate(char* d, size_t n) {
     for (int i = 0; i < n; i++) {
          d[i] += 1;
     }
}
```

```
obfuscate:
     beqz a1, END
     add t0, x0, x0
LOOP:
     lb t1, 0(a0)
     addi t1, t1, 1
     sb t1, 0(a0)
     addi t0, t0, 1
     addi a0, a0, 1
     blt t0, a1, LOOP
END:
     ret
```

Things look great so far. Then you realize that you learned all kinds of crazy techniques to speedup a function in CS61C and this looks very similar to the many SIMD examples you have seen. Although RISCV does have SIMD instructions via a vector extension set, we want to implement our own version of RISCV SIMD. Our idea is to pack multiple characters into a single 32-bit integer. In fact, we do not even need to load and pack the data: four characters have the same width of an integer. Assume `d` is word aligned and that all input characters in the message are less than 254. Also assume `n` is the number of characters in the message and that register `a0` holds the value of `d` and register `a1` holds the value of `n`.

(a) Complete the following implementation for a vectorized version of `obfuscate`:

Solution:

```
void obfuscate_vec(char* d, size_t n) {
     for (int i = 0; i < n / 4 * 4; i += 4) {
          *( (int*) (d + i) ) += INC;
     }
     /* handle tail cases */
     for (int i = n / 4 * 4; i < n; i++) {
          d[i] += 1;
     }
}
```

(b) Refer to the constant `INC` in the code above. What should the value of `INC` be such that `obfuscate_vec` works correctly? Write your answer in hexadecimal.

Solution: 0x01010101

(c) **Loop Unrolling:** You can optimize this procedure further! Loop unrolling is supposed to reduce the number of branch instructions. Complete the following:

Solution:
```
void obfuscate_vec_unroll(char* d, size_t n) {
     for (int i = 0; i < n / 8 * 8; i += 8) {
          *((int*) (d + i)) += INC;
          *((int*) (d + i + 4)) += INC;
     }

     /* handle tail cases */
     for (int i = n / 8 * 8; i < n; i++) {
          d[i] += 1;
     }
}
```

Solution:
```
obfuscate_vec_unrolled:
     beqz a1, END
     add t2, a1, x0
     srli t2, t2, 3
     slli t2, t2, 3
     beqz t2, TAIL
     add t0, x0, x0
     li t3, INC
LOOP_VEC:
     lw t1, 0(a0)
     add t1, t1, t3
     sw t1, 0(a0)
     addi a0, a0, 4
     lw t1, 0(a0)
     add t1, t1, t3
     sw t1, 0(a0)
     addi a0, a0, 4
     addi t0, t0, 8
     blt t0, t2, LOOP_VEC
TAIL:
     add t0, t2, x0
LOOP_TAIL:
     lbu t1, 0(a0)
     addi t1, t1, 1
     sb t1, 0(a0)
     addi t0, t0, 1
     addi a0, a0, 1
     blt t0, a1, LOOP_TAIL
END:
     ret
```

(d) Given a message of length `n` characters, how many instructions are needed after loop unrolling? Express your answer in terms of `n`, such as `3n + 4`. In addition, what is the speed up when `n` is approaching infinity in comparison to the **original non-optimized function** `obfuscate`? Count pseudo-instructions as 1 instruction. You do not need to simplify your expressions.

# of Instructions: $(7 + (n/8) * 10 + 1 + (n\%8) * 6 + 1)$ Speedup: 4.8X

(e) You decide to further improve the code with thread parallelism using 4 threads! Fill in proper OpenMP directive to the blank below:

Solution:

```
void obfuscate_vec_unroll(char* d, size_t n) {
    #pragma omp parallel for
    for (int i = 0; i < n/8*8; i+=8) {
        *((int*) (d + i)) += INC;
        *((int*) (d + i + 4)) += INC;
    }
    # handle tail cases
    for (int i = n/8*8; i < n; i++) {
        d[i] += 1;
    }
}
```

Someone tells you to add `#pragma omp parallel` at Location A in the code above. If you do this, which statement is true about the second `for` loop (the tail case)?

- ○ Always Incorrect
- ○ Always Correct, slower than serial
- ● Sometimes Correct
- ○ Always Correct, faster than serial

(f) Denote the speedup when `n` is approaching infinity of `obfuscate_vec_unroll` from part (d) as "S". Suppose the overhead of running OpenMP is negligible in comparison to the rest of the code, and we can run **four** threads, what is the maximum speed up compared to the **original non optimized function** `obfuscate`?

Solution: $4 * S$

(g) **WSC and Amdahl's Law**: The above program now runs in the cloud with many machines. `obfuscation_vec_unroll` is 90% of all execution (AFTER applying SWAR, unrolling, and OpenMP), and `obfuscation_vec_unroll` can be parallelized across machines.

(i) If we run `obfuscate_vec_unroll` on a cluster of 16 machines, what is the speedup? You may leave your answer as an expression.

Solution: $1/(0.1 + 0.9/16) = 6.4$

(ii) What is the maximum possible speedup we can achieve if we have an unlimited number of machines?

Solution: $1/0.1 = 10.0$

Problem 11 ***[F-3] Pikachu Learns Spark*** **(16 points)**

We are given the entire dataset of every Pokémon and we want to **find the mean of all Pokémon id numbers by type**. Some Pokémon have dual types so that Pokémon's id number will contribute to the average total of both types. For example, Kyurem is both a dragon and an ice type so his id number will contribute to both type's sum when considering the average. **Fill in the blanks for the Python code below**. Use the following Spark Python functions when necessary: `map, flatMap, reduce, reduceByKey`.

Sample input (pokemon_id, pokemon_name, pokemon_types):

```
646 Kyurem Dragon Ice
25 Pikachu Electric
257 Blaziken Fire Fighting
```

Sample output (Type, Number):

```
(Dragon, 587)
(Electric, 412)
```

Solution:

```
def parseLine(line):
    tokens = line.split(" ")
    types = tokens[2:]
    results = []
    for type in types:
        results.append((type, (tokens[0], 1))
    return results


def reduceFunc(v1, v2):

    return (v1[0] + v2[0], v1[1] + v2[1])


def average(k, v):

    return (k, v[0] / v[1])


pokemonData = sc.parallelize(pokemon)

out = pokemonData.flatMap(parseLine)

                 .reduceByKey(reduceFunc)

                 .map(average)
```

Problem 12 ***[F-4] Virtual Memory*** **(20 points)**

Demand paging (storing part of a process' memory on disk) is yet another example of caching in computer systems. If we think of main memory as a cache for disk, what are the properties of this cache? Assume a machine with 64 bit addresses, 16KB pages, a 4-way fully associative TLB, and 8B words.

(a) Associativity?

- ○ Direct Mapped
- ● Fully Associative
- ○ N-Way Set Associative

(b) Block size:

Solution: 16KB

(c) Address layout. Your answer should be of the form [N:M] where N is the bit number of the most significant bit of the field and N is the bit number of the least significant bit of the field. For example, if the tag consists of the first 4 least-significant bits, you should write [3:0]. If the field is not applicable to paging, you may write "N/A".

Solution: Tag bits:[63:14] Index bits:N/A Offset bits:[13:0]

(d) Write policy?

- ○ Write Through
- ● Write Back

(e) Allocation policy?

- ● Write Allocate
- ○ Write No Allocate

TLB Reach. We have written a strange and mysterious summation function. It uses a mystery constant called `T`. You may assume that `T` is defined (but you don't know what to) and that `arr` will always have enough elements (the function will never access outside of `arr`). The function is run on a machine with the following properties:

- 64 bit addresses
- 4 byte words
- 4KiB pages
- 4GiB of main memory
- 1MiB fully-associative cache with 64 byte blocks
- 2 entry fully associative TLB

- 4 level page table with 8 byte entries
- The OS uses LRU when paging to disk

```
#define NITER 10*1024*1024
#define T ???       // see below

int MysterySum(int *arr) {
     int i = 0;
     int sum = 0;
     for(; i < NITER / 2; i++)
          int p = (i % T)*4096;
          int b = i % 4096;
          sum += arr[p + b];
     }

     /* Timer starts here*/
     for(; i < NITIR; i++) {
          int p = (i % T)*4096;
          int b = i % 4096;
          sum += arr[p + b];
     }
     /* Timer ends here */

     return sum;
}
```

(f) **Performance of T**

Rank the the following values of T based on how fast the second loop only executes (assuming the first loop has already ran). You should state whether pairs of values are $<$ or $=$. For example, you should write $1 < 2$ if T=1 causes the second loop to run strictly slower than T=2. Likewise, you could write 8=2 if 8 is about as fast as 2.

T = 1, 2, 3, 4

Solution: $3 = 4 < 1 = 2$

(g) **System Design**

What system parameter would you change in order to maximize system performance for T=27. You must mark only one of the following (pick the one with the largest performance gain):

- ○ Address Size
- ○ Page Size
- ○ Word Size
- ○ Main Memory Size
- ○ Cache Capacity
- ○ Cache Block Size
- ● TLB Capacity
- ○ TLB Associativity
- ○ Page Table Depth
- ○ Page Table Entry Size

(h) **Page Table Walk**

Given the list of virtual addresses, find the corresponding physical addresses. For each address, you must also note whether the access was a TLB hit, Page Table hit, or Page Fault (by writing yes/no for each). If the access is a page fault, you should leave the PPN and PA fields blank. Do not add this entry to the TLB.

Our virtual memory space has 16-byte pages and maintains a fully-associative, two-entry TLB with LRU replacement. The page table system is hierarchical and has two levels. The two most-significant bits of the VPN index the L1 table, and the two least-significant bits of the VPN index the L2 table.

Solution:

Virtual Address	Virtual Page Number	Physical Page Number	Physical Address	TLB Hit, Page Table Hit, Page Fault?
0x10	0x1 = 0b00 01	0x12	0x120	Page Table Hit
0x5C	0x5 = 0b01 01			Page Fault
0x39	0x3 = 0b00 11	0x5C	0x5C9	Page Table Hit
0x1F	0x1 = 0b00 01	0x12	0x12F	TLB Hit

TLB:

VPN	PPN
0x1 → 0x2	0x12 → 0x9
0x3 → 0x1	0x5C → 0x12

Problem 13 ***[F-5] Potpourri*** **(8 points)**

Answer the following questions

(a) You have a computer that, well, stinks. It goes down on average 6 times a day and it takes 1 hour to get working again. What is the current system's availability?

- ○ 0.5
- ○ 0.7
- ○ 0.6
- ● 0.8

Solution: Availability = MTTF/(MTTF+MTTR) = 4/(4+1) = 4/5 = 0.8

(b) Assume you have the computer from part (a) when the manufacturer offers you a deal. **a:** A new computer that only crashes 4 times per day or **b:** support that can reduce the time to fix to 6 minutes. Which one should you choose?

- ○ a
- ● b

Solution: Availability(a) = 6/(6+1) = 6/7
Availability(b) = 4/(4+.1) = 4/4.1
Availability(b) > Availability(a)

(c) You have a processor that has a clock rate of 2GHz, a time to poll of 200 cycles for I/O, and you need to poll I/O at 100 Hz. If you use polling, what is the **percentage** of time you will need to spend polling?

- ○ 1%
- ○ 0.01%
- ○ 0.1%
- ● 0.001%

Solution: Polling at 100Hz means 100 times per second. Each time you poll it costs 200 cycles. Thus, you spend $100 * 200 = 20,000$ cycles polling per second. This is $20,000/(2 * 10^9) = 0.001\%$

(d) If the data comes in very infrequently do you want to use interrupts or polling? Why?

- ● interrupts
- ○ polling

Solution: Interrupts because we would be wasting a lot of cycles polling for infrequent data.

Figure 4: Good Luck And Don't F*** It Up