System Skill Final Quiz

Date: Thursday, April 1st, 2021
Instructor: Rachata Ausavarungnirun

Problem 1 (20 Points):	
Problem 2 (20 Points):	
Problem 3 (20 Points):	
Problem 4 (15 Points):	
Problem 5 (30 Points):	
Problem 6 (25 Points):	
Total (100+30 Points):	

Instructions:

1. This is a 18-hour exam. **If you get 100, you get a full score. Any points above 100 goes to your extra credit at the conversion rate of 50% per point.**
2. The total points is far greater than 100. This is intended so that 1) you can pick the questions you are more comfortable and 2) allows rooms for extra credit if you know all the class material. Please read all the questions first.
3. Submit your work as a zip file on Canvas.
4. Because this is a 24-hour exam, I expect everyone to **test your code**.
5. If not specified, input and output types are a part of the question. Please use appropriate input and output types that make sense for the purpose of the question.
6. Please clearly comment your code, especially if your code do not work perfectly.
7. Clearly indicate your final answer for each conceptual problem.
8. **DO NOT CHEAT.** If we catch you cheating in any shape or form, you will be penalized based on **my plagiarism policy** ($N*$ 10% of your total grade, where N is the number of times you plagiarized previously).

Tips:

- **Read everything.** Read all the questions on all pages first and formulate a plan.
- **Be cognizant of time.** It is a sad day if you click submit when the submission site close.
- **Canvas allows resubmission.** I will take a look at the last version you submit.
- **Show work when needed.** You will receive partial credit at the instructors' discretion.

1. Data Representation [20 points]

Scoring: For this question, each part is worth 3 points to the maximum of 20 points.

Please provide the **hexadecimal** number corresponding to the corresponding result values of the following operations. **Assume that a char is 1 byte, a short int is 2 bytes, an int is 4 bytes, a long int is 8 bytes and all shifts are arithmatic**.

```
(unsigned int) 15
```

What is the value (in hex) of the above statement?

```
(short int) 15
```

What is the value (in hex) of the above statement?

```
(unsigned long) -2
```

What is the value (in hex) of the above statement?

```
int i=-2;
unsigned long j = i;
```

What is the value (in hex) of j?

```
long i = -2;
char j = i >> 8;
```

What is the value (in hex) of i?

```
long i = -2;
char j = (unsigned char)i >> 32;
```

What is the value (in hex) of i?

2. Datalab ... Again! (20 points)

We are going to ask you to implement more bit arithmetic manipulation in a similar fashion as assignment 3.

Unlike assignment 3, you are to write the answer on this exam.

Similar to Assignment 3, you are *only* allowed to use the following eight operators:

```
 ! ˜ & ˆ | + << >>
```

Also, you are not allowed to use any constants longer than 8 bits.

Table 1 describes a set of functions that manipulate and test sets of bits. The "Rating" field gives the difficulty rating (the number of points) for the puzzle, and the "Max ops" field gives the maximum number of operators you are allowed to use to implement each function.

Name	Description	Rating	Max Ops
`bitXor(x)`	return `xˆy` without using only `˜` and `&`	1	14
`isEqual(x,y)`	return 1 of x == y, 0 otherwise	2	5
`anyOddBit(x)`	return 1 if any odd-numbered bit in our data word is set to 1	2	12
`conditional(x)`	same as x ? y : z	3	16

Table 1: Bit-Level Manipulation Functions.

For every question, **Please explain what you are trying to do**.

Write down the function body for `bitXor(x)` below.

Write down the function body for `isEqual(x)` below.

Write down the function body for `anyOddBit(x)` below. Note that this function check and return 1 if any of the odd bit position has the value 1.

Write down the function body for `conditional(x)` below.

3. Jump Table [20 points]

In this question, consider the following assembly codes below. Fill in the rest of the C code for each of the switch cases. Write "NOTHING HERE" if the space should be left blank or if that line of code should not exist (i.e., the program does not suppose to modify `result` at that line).

```
blah:
pushl %ebp
movl %esp, %ebp
movl 8(%ebp), %edx
movl 12(%ebp), %eax
cmpl $7, %edx
ja .L8
jmp *.L9(,%edx,4)
.section .rodata
.align 4
.align 4
.L9:
.long .L8
.long .L4
.long .L5
.long .L5
.long .L8
.long .L7
.long .L6
.long .L4
.text
.L4:
mov (%edx), %eax
jmp .L7
.L5:
mov (%eax), %eax
jmp .L2
.L6:
leaq (%eax, %%edx, 4)
jmp .L2
.L7:
addq %eax, %eax
.L8:
incl %eax
.L2:
popl %ebp
ret
```

In the space below, fill in the blank to reflect the assembly code above.

```
int blah(int a, int b)
{
int result;

switch(________)
{

    case ________:

    case ________:

        result = _________;
        break;

    case ________:

        result = _________;
        break;

    case ________:

    case ________:

        result = _________;

    case ________:

        result = _________;

    default:
        result = _________;
}

return result;
}
```

4. Creating a New ISA [15 points]

Consider the following 16-bit TGGS-ISA with the following features.

- The ISA is byte addressable and there are 8 16-bit registers.
- The machine stop its execution whenever the decoder observes the instruction JMP 0, in which case it finish all remaining instructions in the pipeline.
- There are 3 status bits: negative, zero, overflow.
- The negative bit is set to true if the destination register yield a negative result, in which case value of the destination register is the leftmost 16 bits.
- The zero bit is set to true if the destination register yield zero.
- The overflow bit is set to true if the destination register overflows (i.e., result in a number higher than 16 bits, in which case the destination register stores the leftmost 16 bits value).

Instruction Type	**Opcode**	**Op1**	**Op2**	**Op3**	**Unused**
Compute R Rs1, Rs2, Rd	4 bits	3 bits	3 bits	3 bits	3 bits
Compute I Rs1, Rd, IMM	4 bits	3 bits	3 bits	6 bits	None
Memory Type 1 Rd, Rs, IMM	4 bits	3 bits	3 bits	6 bits	None
Memory Type 2 Rd, IMM	4 bits	3 bits	9 bits		None
Cond. Type 1 IMM	4 bits	12 bits			None
Cond. Type 2 Rd	4 bits	3 bits			6 Bits
Cond. Type 3 Rs1, Rs2, Rd	4 bits	3 bits	3 bits	3 bits	3 Bits

Table 2: Bit pattern for each instruction types. The most significant bit is on the leftmost side and the least significant bit is on the rightmost side. For example, if there are 3 unused bits, then bits 0-2 are unused and can be any values (you can write xxx in the answerbox for these bits).

Instruction	**Opcode**	**Op1**	**Op2**	**Op3**	**Description**
ADD	0000	Rs1	Rs2	Rd	$R_d <= R_{s1} + Rs2$
ADDI	0001	Rs1	Rd	IMM	$R_d <= R_{s1} + IMM$
SUB	0010	Rs1	Rs2	Rd	$R_d <= R_{s1} - Rs2$
SUBI	0011	Rs1	Rd	IMM	$R_d <= R_{s1} - IMM$
AND	0100	Rs1	Rs2	Rd	$R_d <= R_{s1} and Rs2$
OR	0101	Rs1	Rs2	Rd	$R_d <= R_{s1} or Rs2$
XOR	0110	Rs1	Rs2	Rd	$R_d <= R_{s1} xor Rs2$
LD	0111	Rd	Rs	IMM	$R_d <= mem[Rs + IMM]$
LDI	1000	Rd	IMM		$R_d <= IMM$
ST	1001	Rd	Rs	IMM	$R_s => mem[Rd + IMM]$
JMP	1010	IMM			$PC <= IMM$
JMPR	1011	Rd			$PC <= Rd$
BLT	1100	Rs1	Rs2	Rd	If $Rs1 < Rs2$ then $PC <= Rd$ else $PC <= PC + 2$
BGT	1101	Rs1	Rs2	Rd	If $Rs1 > Rs2$ then $PC <= Rd$ else $PC <= PC + 2$
BNE	1110	Rs1	Rs2	Rd	If $Rs1 = Rs2$ then $PC <= Rd$ else $PC <= PC + 2$
LDPC	1111	Rd			$Rd <= PC$

Table 3: All instructions in the TGGS-ISA.

(a) In the table below, please fill in the bit pattern that represent each instruction. Write NOT POSSIBLE when there is no way to create a bit pattern for the instruction. You should use 'x' when specifying the don't care bit (i.e., the value of this bit does not matter)

Instruction	**Bits**
ADD R5 R2 R2	
BGT R0 R1 0x20	
LD R3 R4 0x7cc	
LD R5 0x20	
JMP 0x3cc	
SUBI R1 R2 0x100	

5. Caching [30 points]

In this question, let's assume that we have a 32-bit system with a single level 4-way set associative cache with 32 sets, and a cache block size of 64 bytes.

How many bits are needed for the setID and the tags? Draw the breakdown of the tag/index/byte-in-block bits.

For the following program, assume that an integer is 4 bytes.

```
int i; // Assume these variables are stored in the registers.
int a[2048]; // Assume that a = 0x10000000
int b[2048]; // Assume that b = 0x20000000

for(i=0;i<2048;i++)
    a[i] = i;

for(i=0;i<2048;i++)
    b[i] = a[i]++;
```

What is the total number of cache accesses? What is the number of cache hits and what is the number of cache misses? **Show your work.**

Now your friend "think" that you can modify the program to be more cache friendly. Consider the following changes:

```
int i,j,m = 16,n = 128;; // Assume these variables are stored in the registers.
int a[2048];            // Assume that a = 0x10000000
int b[2048];            // Assume that b = 0x20000000

for(j=0;j<m;j++)
{
    for(i=0;i<n;i++)
        a[i + j*128] = i + j*128;
        b[i + j*128] = a[i + j*128]++;
}
```

What is the total number of cache accesses? What is the number of cache hits and what is the number of cache misses? Assuming that you can change the program in anyway you want as long as it still produces the same result, is this the best hit rate you can have? **Show your work.**

From the same program above, if the solution is not optimal, then what is your proposed changes? If it is optimal, what is the maximum number of n that still allows the program to have the **minimum possible** cache hit rate? Why?

6. Virtual Memory [25 points]

What is a TLB? What is the main benefit of the TLB?

Why is VIPT faster than PIPT?

What problems you can run into if you use virtual address as the tag bits?

Assume a 32-bit byte-addressable machine that utilizes 4KB page size with a one-level page table.

What is the number of bits required for the page offset?

Let's create a simple **BIG endian** machine that still utilize **4KB** page size, and 32-bit address and each page table entry is 32-bit, where the physical page number is located at the most-significant bits of the page table entry.

Assuming that you are trying to **support 2GB physical memory (i.e., you have 2GB of DRAM on your machine)**, how many bits do you need for the physical page number? (Hint: Start by thinking what is the range of all possible addresses in a 2GB memory?)

Assuming the following data in the memory and the base address of the page table is at `0x80`.

Address	**Values (in hexadecimal) [Lowest bit – Highest bit]**
0x00	`00 10 20 30 40 50 60 70 80 90 a0 b0 c0 d0 e0 f0`
0x10	`10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f`
0x20	`00 10 20 30 40 50 60 70 80 90 a0 b0 c0 d0 e0 f0`
0x30	`30 31 32 33 34 35 36 37 38 39 3a 3b 3c 3d 3e 3f`
0x40	`ab 00 20 30 40 50 60 70 80 90 a0 b0 c0 d0 e0 f0`
0x50	`19 15 12 0a 6b 3a 4b 12 91 ac ff fe 3c 3d 3e 4f`
0x60	`12 50 62 8a 5e 5f df ea 99 ac 74 6b 91 44 33 ef`
0x70	`70 71 72 73 74 75 76 77 78 79 7a 7b 7c 7d 7e 7f`
0x80	`91 40 8a 00 8c 14 fe ff 74 13 02 ba 6b 12 4b 31`
0x90	`90 91 92 93 94 95 96 97 98 99 9a 9b 9c 9d 9e 9f`
0xa0	`80 00 8a 00 8c 14 fe ff 72 14 0a 6b 10 02 e1 ba`
0xb0	`30 31 32 33 34 35 36 37 38 39 3a 3b 3c 3d 3e 3f`
0xc0	`80 00 8a 00 8c 14 fe ff 72 14 0a 6b 10 01 e1 ba`
0xd0	`91 40 8a 00 8c 14 fe ff 74 13 02 ba 6b 12 4b 31`
0xe0	`70 00 8a 00 8c 14 fe ff 72 14 0a 6b 10 03 e1 ba`
0xf0	`91 40 8a 00 8c 14 fe ff 74 13 02 ba 6b 12 4b 31`
0x100	`00 10 20 30 40 50 60 70 80 90 a0 b0 c0 d0 e0 f0`
0x110	`10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f`
0x120	`00 10 20 30 40 50 60 70 80 90 a0 b0 c0 d0 e0 f0`
0x130	`30 31 32 33 34 35 36 37 38 39 3a 3b 3c 3d 3e 3f`
0x140	`ab 00 20 30 40 50 60 70 80 90 a0 b0 c0 d0 e0 f0`
0x150	`19 15 12 0a 6b 3a 4b 12 91 ac ff fe 3c 3d 3e 4f`
0x160	`12 50 62 8a 5e 5f df ea 99 ac 74 6b 91 44 33 ef`
0x170	`70 71 72 73 74 75 76 77 78 79 7a 7b 7c 7d 7e 7f`
0x180	`91 40 8a 00 8c 14 fe ff 74 13 02 ba 6b 12 4b 31`
0x190	`90 91 92 93 94 95 96 97 98 99 9a 9b 9c 9d 9e 9f`
0x1a0	`80 00 8a 00 8c 14 fe ff 72 14 0a 6b 10 02 e1 ba`
0x1b0	`30 31 32 33 34 35 36 37 38 39 3a 3b 3c 3d 3e 3f`
0x1c0	`80 00 8a 00 8c 14 fe ff 72 14 0a 6b 10 01 e1 ba`
0x1d0	`91 40 8a 00 8c 14 fe ff 74 13 02 ba 6b 12 4b 31`
0x1e0	`70 00 8a 00 8c 14 fe ff 72 14 0a 6b 10 03 e1 ba`
0x1f0	`91 40 8a 00 8c 14 fe ff 74 13 02 ba 6b 12 4b 31`

What are the virtual page number and the physical page number for a virtual address `0x00048765`? Put in **Not enough information** if the table does not provide enough information to get the physical address and explain why.

What is the physical address for a virtual address `0x00048765`? Put in **Not enough information** if the table does not provide enough information to get the physical address and explain why.

What are the virrtual page number and the physical page number for a virtual address `0x0001beef`? Put in **Not enough information** if the table does not provide enough information to get the physical address and explain why.

What is the physical address for a virtual address `0x0001beef`? Put in **Not enough information** if the table does not provide enough information to get the physical address and explain why.

Initials:

Log Table

N	$log_2 N$
1	0
2	1
4	2
8	3
16	4
32	5
64	6
128	7
256	8
512	9
1024 (1k)	10
2048 (2k)	11
4096 (4k)	12
8192 (8k)	13
16384 (16k)	14
32768 (32k)	15
62236 (64k)	16
131072 (128k)	17
262144 (256k)	18
524288 (512k)	19
1048576 (1M)	20
2097152 (2M)	21
4194304 (4M)	22
8388608 (8M)	23
16777216 (16M)	24

Stratchpad

Stratchpad