System Skill Final Quiz

Date:Thursday, July 21st, 2022
Due: Friday, July 22nd, 2022 at 11.59PM
Instructor: Rachata Ausavarungnirun

Problem 1 (25 Points):	
Problem 2 (10 Points):	
Problem 3 (20 Points):	
Problem 4 (20 Points):	
Problem 5 (30 Points):	
Extra Credit (10 Points):	
Total (100+15 Points):	

Instructions:

1. This is a 38-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. Submit your work as a pdf file on Canvas.
3. Clearly indicate your final answer for each conceptual problem.
4. **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). This also includes asking the internet for the answer to our questions.

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. Assembly and ISA! [25 points]

During the semester, we learn how x86 can be assembled and deassembled into assembly code and binaries in assignment 4 task 2. In this question, one of your TAs would like to be cool and design his own ISAs. Consider the following 16-bit MUIC-IS-COOL-ISA with the following features.

- The ISA is byte addressable and there are 8 16-bit registers from R0 to R7.
- 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**
Bits location	Higher bits <-----------> Lower bits				
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 1: 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.

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 2: All instructions in the MUIC-IS-COOL-ISA. Rs1 is the input source 1, Rs2 is the input source 2, Rd is the destination (output), and IMM is the immediate (constant value). Register bits are denoted based on their register ID. For example, if Rs1 is R3, it will have the value equal to 3 in the appropriate register field in the binary instruction.

(a) Now that we have establish the ISA specification. (15 points)

Assume PC starts at `0x30`. What is the code (in MUIC-IS-COOL assembly) from the memory snapshot below. **Note that for this memory snapshot, the bits within the data word in the table below is sorted using [highest bit – lowest bit] format (i.e., if the data word is `0x1234`, then the word is 0b'0001 0010 0011 0100).**

Address	Values (in hex) [Lowest address – Highest address]
0x00	`00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f`
0x10	`00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f`
0x20	`00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f`
0x30	`80 00 8a 00 8c 14 fe f2 72 14 0a 6b 10 02 e1 ba`
0x40	`91 40 a0 00 e2 ff 01 0f ff 2e be ef 24 31 a0 00`
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	`80 81 82 83 84 85 86 87 88 89 8a 8b 8c 8d 8e 8f`
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	`91 40 8a 00 8c 14 fe ff 74 13 02 ba 6b 12 4b 31`
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`

Instruction	**Bits Pattern**

(b) What are the values of all the registers inside the register files after the program finishes? You can put in XX for an unknown value. (10 points)

Address	**Values (in Decimal)**
R0	
R1	
R2	
R3	
R4	
R5	
R6	
R7	

2. Code Size [10 points]

For the following code, please fill in the number (in **hexadecimal base**) for the address of each instruction.

Address	Instruction (in binary)	Instruction (in Assembly)
5fa:	55	push %rbp
	48 89 e5	mov %rsp,%rbp
	89 7d fc	mov %edi,-0x4(%rbp)
	89 75 f8	mov %esi,-0x8(%rbp)
	8b 55 fc	mov -0x4(%rbp),%edx
	8b 45 f8	mov -0x8(%rbp),%eax
	01 d0	add %edx,%eax
	5d	pop %rbp
	c3	retq
60e:	55	push %rbp
	48 89 e5	mov %rsp,%rbp
	48 83 ec 08	sub $0x8,%rsp
	89 7d fc	mov %edi,-0x4(%rbp)
	89 75 f8	mov %esi,-0x8(%rbp)
	8b 55 f8	mov -0x8(%rbp),%edx
	8b 45 fc	mov -0x4(%rbp),%eax
	89 d6	mov %edx,%esi
	89 c7	mov %eax,%edi
	e8 cf ff ff ff	callq 5fa

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 support to modify `result` at that line).

Assume that both a is in `%rdi` and b is in `%rsi`

```
quiz3:
pushl %ebp
movl %esp, %ebp
movl %rdi, %edx
movl %rsi, %eax
cmpl $8, %edx
ja .L8
jmp *.L9(,%edx,4)
.section .rodata
.align 4
.align 4
.L9:
.long .L8
.long .L4
.long .L8
.long .L5
.long .L8
.long .L4
.long .L6
.long .L8
.long .L2
.text
.L4:
movl %edx, %eax
jmp .L2
.L5:
decl %eax
jmp .L2
.L6:
incl %eax
jmp .L2
.L8:
movl $-1, %eax
.L2:
popl %ebp
ret
```

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

```
int quiz3(int a, int b)
{
int result = _________;

int temp = _________; // Feel free to use this if you need to store
                      // any temp variable. Leave blank if not needed.

switch(_________)
{

    case _________:
    case _________:
        result = __________;
        break;

    case _________:

        result = __________;
        break;

    case _________:

        result = __________;
        break;

    case _________:

        result = __________;
        break;

    default:

        result = __________;
}
return result;
}
```

4. Caching [20 points]

In this question, let's assume that we have a 16-bit system with a single level 5-way set associative cache with 4 sets, and a cache block size of 32 bytes.

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

What is the total size of this cache?

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 = 0x1000
int b[2048]; // Assume that b = 0x8000000

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

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

Is it possible for me to have a combination of X, Y and Z such that the cache hit rate is 0%. Why or why not? **Show your work.**

5. Virtual Memory [30 points]

Let's create a simple **BIG endian** machine that utilize two-level page table with a 4KB page size (similar to what we learn in class), 4KB page table, and this processor also uses 32-bit address. Assuming the following:

- Data in the memory and the page table root is at `0x10`.
- The status bits in the PTE for both levels are 12 bits, and the page table entries is 32-bit long, where the n most significant bits after the page offset are either used as the ID of the next page (for the first level) or the physical page number (for the second level).
- To get the address of the first entry in the second level of the page table, our machine will take the ID of the next page. Then, it appends this ID with m additional zero bits, where m is the number of bits required for the page table size. For example, if your page table size is 64 bytes and the ID is 5, m is 6. So, the next level page for this ID is at address 0x140.

Address	**Values (in hexadecimal) [Lowest byte – Highest byte]**
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 08 00 00 00`
0x20	`19 15 12 0a 6b 3a 60 70 19 15 12 dd 6b d0 e0 f0`
0x30	`30 31 ee 33 34 35 36 37 00 10 0e aa 3c 3d 3e 3f`
0x40	`00 10 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`
0x100000	`00 10 20 30 40 50 60 70 80 90 a0 b0 c0 d0 e0 f0`
0x100010	`10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f`
0x100020	`00 10 20 30 40 50 60 70 80 90 a0 b0 c0 d0 e0 f0`
0x100030	`30 31 32 33 34 35 36 37 38 39 3a 3b 3c 3d 3e 3f`
0x100040	`00 10 20 30 40 50 60 70 80 90 a0 b0 c0 d0 e0 f0`
0x100050	`19 15 12 0a 6b 3a 4b 12 91 ac ff fe 3c 3d 3e 4f`
0x100060	`12 50 62 8a 5e 5f df ea 99 ac 74 6b 91 44 33 ef`
0x100070	`70 71 72 73 74 75 76 77 78 79 7a 7b 7c 7d 7e 7f`
0x8000000	`91 40 8a 00 8c 14 fe ff 74 13 02 ba 6b 12 4b 31`
0x8000010	`90 91 92 93 94 95 96 97 98 99 9a 9b 9c 9d 9e 9f`
0x8000020	`80 00 8a 00 8c 14 fe ff 72 14 0a 6b 10 02 e1 ba`
0x8000030	`30 31 32 33 34 35 36 37 38 39 3a 3b 3c 3d 3e 3f`
0x8000040	`80 00 8a 00 8c 14 fe ff 72 14 0a 6b 10 01 e1 ba`
0x8000050	`91 40 8a 00 8c 14 fe ff 74 13 02 ba 6b 12 4b 31`
0x8000060	`70 00 8a 00 8c 14 fe ff 72 14 0a 6b 10 03 e1 ba`
0x8000070	`91 40 8a 00 8c 14 fe ff 74 13 02 ba 6b 12 4b 31`

(a) What is the physical address for a virtual address `0x0000beef`? Put in **Not enough information** if the table does not provide enough information to get the physical address.

(b) What is the physical address for a virtual address `0x00803fff`? Put in **Not enough information** if the table does not provide enough information to get the physical address.

(c) What is the physical address for a virtual address `0x0000beef` if this system were to use a single level 16KB page instead? Put in **Not enough information** if the table does not provide enough information to get the physical address.

(d) Assuming that the memory access takes 100 cycles to access DRAM, the system has 4-level page table (i.e., a page walk have to access the memory 4 times before it can access its data), an TLB access takes 1 cycle, and a L1 cache access to the set takes 1 cycle and the tag comparison in the L1 cache takes another 1 cycle. How long does it takes to load a data that has a TLB miss and a L1 cache hit? Feels free to explain your answer.

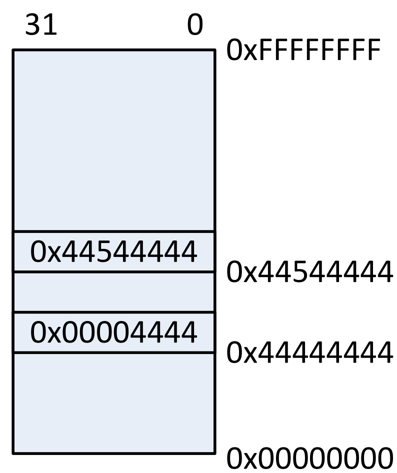
6. Extra Credit: 0x44444444 [10 points]

Do not attempt this until you are done with other questions.

A 32-bit processor implements paging-based virtual memory using a single-level page table. The following are the assumptions about the processor's virtual memory.

- A page table entry (PTE) is 4-bytes in size.
- A PTE stores the physical page number in the least-significant bits.
- The base address of the page tables is page-aligned.

The following figure shows the physical memory of the processor at a particular point in time.



4GB Physical Memory

At this point, when the processor executes the following piece of code, it turns out that the processor accesses the page table entry residing at the physical address of `0x44444444`.

```
char *ptr = 0x44444444;
char val = *ptr; // val == 0x44
```

What is the page size of the processor? Show work in detail.

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