Practice Quiz #2

<table>
<thead>
<tr>
<th>Name</th>
<th>Athena login name</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Recitation section
- WF 11, 35-308 (Silvina)
- WF 12, 35-308 (Silvina)
- WF 1, 38-166 (Andy)
- None (pick up quiz in 32-G846)

Please enter your name, Athena login name, and recitation section above. Enter your answers in the spaces provided below. You can use the extra white space and the backs of the pages for scratch work.

The exam has an appendix which includes our 6.S084 reference guide and the RISC-V calling conventions.

Problem 1. Assembly Language (20 points)

(A) (6 points) You are given the following C code along with a partial translation of this code into RISC-V assembly. Please complete the assembly implementation of this code by fully specifying the 2 missing RISC-V instructions just before label L1. These must be actual RISC-V instructions, you may not use any pseudo-instructions.

```c
int A[1000];
...
for (int i = 0; i < 1000; i = i + 1) {
    A[i] = A[i] ^ 0x12345; // ^ is bitwise XOR in C
}

// Translation into RISC-V assembly:
addi x11, x0, 0       // x11: i
addi x12, x0, 1000    // x12: loop termination condition
addi x16, x0, 0x100   // x16: array base
missing instr
missing instr
```

```assembly
L1: bge x11, x12, L2   // byte offset into array
    slli x13, x11, 2
    add x17, x13, x16   // address of A[i]
    lw x14, 0(x17)      // x14 = A[i]
    addi x11, x11, 1
    xor x14, x14, x15
    sw x14, 0(x17)
    jal x0, L1

L2: ...
```

```assembly
. = 0x100
A: ...          // contents of array A
```

6.S084 Spring 2018 - 1 of 10 - Practice Quiz #2
For the RISC-V instruction sequences below, provide the hex values of the specified registers after each sequence has been executed. Assume that each sequence execution ends when it reaches the \texttt{unimp} instruction. Also assume that all registers are initialized to 0 before execution of each sequence begins.

(B) (6 points)

\begin{verbatim}
li x11, 0x242
slli x12, x11, 12
li x13, 0xff
or x14, x12, x13
unimp
\end{verbatim}

Value left in x11: 0x___________

Value left in x12: 0x___________

Value left in x13: 0x___________

Value left in x14: 0x___________

(C) (8 points)

\begin{verbatim}
\texttt{.} = 0x0
li x11, 0x200
lw x11, 0(x11)
blt x11, x0, L1
addi x12, x11, -1
jal x13, L2

L1: sub x12, x0, x11
L2: xori x12, x12, 0xffffffff
unimp

\texttt{.} = 0x200
X: .word 0x87654321
\end{verbatim}

Value left in x0: 0x___________

Value left in x11: 0x___________

Value left in x12: 0x___________

Value left in x13: 0x___________
Problem 2. RISC-V Assembly and Calling Conventions (18 points)

Our RISC-V processor does not have a multiply instruction, so we have to do multiplications in software. The C code below shows a recursive implementation of multiplication by repeated addition of unsigned integers (in C, `unsigned int` denotes an unsigned integer). Ben Bitdiddle has written and hand-compiled this function into the assembly code given below, but the code is not behaving as expected. Find the bugs in Ben’s assembly code and write a correct version.

**C code for unsigned multiplication**

```c
unsigned int mul(unsigned int x, unsigned int y) {
    if (x == 0) {
        return 0;
    } else {
        unsigned int lowbit = x & 1;
        unsigned int p = lowbit? y : 0;
        return p + (mul(x >> 1, y) << 1);
    }
}
```

**Buggy assembly code**

```assembly
mul:
    addi sp, sp, -8
    sw s0, 0(sp)
    sw ra, 4(sp)
    beqz a0, mul_done
   andi s0, a0, 1  // lowbit in s0
    mv t0, zero     // p in t0
    beqz s0, lowbit_zero
    mv t0, a0
lowbit_zero:
    slli a0, a0, 1
    jal mul
    srl a0, a0, 1
    add a0, t0, a0
    lw s0, 4(sp)
    lw ra, 0(sp)
    addi sp, sp, 8
mul_done:
    ret
```

Rewrite assembly code so that it behaves as expected and follows the RISC-V calling convention:
Problem 3. Procedures and Stacks (20 points)

You are given an incomplete listing of a C procedure and its translation to RISC-V assembly code (shown on the right):

```c
int f(int a, int b) {
    int tempa = a + a;
    int tempb = ***;
    if (a < b)
        return a + f(tempa, tempb);
    else return ???;
}
```

```assembly
f:  bge a0, a1, L1
xx: addi sp, sp, -8
    sw a0, 0(sp)
    sw ra, 4(sp)
    add a0, a0, a0
    addi a1, a1, 1
    jal ra, f
L1: sub a0, a0, a1
    jr ra
    L2: lw t0, 0(sp)
    lw ra, 4(sp)
    addi sp, sp, 8
    add a0, a0, t0
    jr ra
```

(A) (3 points) Give the HEX encoding of the ‘jr ra’ instruction at the end of the program.

Hex encoding of jr ra: 0x ________

(B) (2 points) Give the HEX encoding of the ‘bge a0, a1, L1’ instruction at the beginning of the program.

Hex encoding of bge a0, a1, L1: 0x ________

(C) (2 points) What is the missing C expression corresponding to the ‘***’ in the above program?

(give C code expression)

(D) (2 points) What is the missing C expression corresponding to the ‘???’ in the above program?

(give C code expression)

(E) (2 points) Suppose the instruction ‘jal ra, f’ were replaced by the instruction ‘j f’ in the above program. What would be the result? Mark exactly one answer:

The procedure would still work just fine: ________

The procedure would compute the proper value, but not restore registers properly: ________

The procedure would no longer compute f(a,b) properly: ________
The program calls `f(2, 5)` via the instruction `jal ra, f`. The program is interrupted just prior to an execution of the `bge` instruction at label `f`. The diagram in the right shows the contents of a region of memory. All addresses and data values are shown in hex. **The current contents of the SP are 0x128.**

(F) (3 points) What are the arguments to the current (most recent) call to `f`?

Current arguments, `a = __________ ; b = __________`

(G) (2 points) What value was in SP just prior to the initial call to `f`?

Initial contents of SP: 0x __________

(H) (2 points) What is the address of the `jal ra, f` instruction that made the original call to `f(2, 5)`?

Address of original call to `f(2,5)`: 0x __________

(I) (2 points) What is the hex address of the instruction at L2?

Address of instruction at L2: 0x __________
Problem 4. Hardware Implementation (18 points)

We want to add the following instruction to the RISC-V ISA:

lwr rd, rs1, rs2

The behavior of this new lwr instruction is that it adds the contents of source registers rs1 and rs2 and uses the result as the address for your load. In other words:

\[
\text{reg}[rd] \leftarrow \text{mem}[\text{reg}[rs1] + \text{reg}[rs2]] \\
\text{pc} \leftarrow \text{pc} + 4
\]

Suppose you were given a decoder that was fully functioning prior to the addition of this new instruction, and you were told that a new type \texttt{IType LWR} has been added for this instruction. Fill in the required changes to the Decode and Execute functions in order to support adding this instruction. We have included relevant portions of these functions below to simplify this task. Assume that the rest of the hardware implementation is the same as what you have seen in lecture and lab.

(A) (12 points) Fill in the required additions to the code below.

```plaintext
typedef enum { OP, OPIMM, BRANCH, LUI, JAL, JALR, LOAD, STORE, GCD, AUIPC, LWR, Unsupported} IType deriving (Bits, Eq, FShow);

function DecodedInst decode(Bit#(32) inst);
    DecodedInst dInst = ?;
    dInst.iType = Unsupported;
    case(opcode)
    ...
    opLoad: if (funct3 == fnLW) begin
    end
    endcase
    return dInst;
endfunction
```

Write your code below. Please add arrows to indicate where your code fits in the \texttt{decode} function.
function ExecInst execute(DecodedInst dInst, Word rVal1, Word rVal2, Word pc);
  let imm = dInst.imm;
  let brFunc = dInst.brFunc;
  let aluFunc = dInst.aluFunc;
  Word data = ?;
  Word nextPc = ?;
  Word addr = ?;
  case (dInst.iType)
    OP: begin data = alu(rVal1, rVal2, aluFunc); nextPc = pc+4; end
    OPIMM: begin data = alu(rVal1, imm, aluFunc); nextPc = pc+4; end
    BRANCH: begin nextPc=aluBr(rVal1,rVal2,brFunc)? pc+imm : pc+4; end
    LUI: begin data = imm; nextPc = pc+4; end
    JAL: begin data = pc+4; nextPc = pc+imm; end
    JALR: begin data = pc+4; nextPc = (rVal1+imm) & ~1; end
    LOAD: begin addr = rVal1+imm; nextPc = pc+4; end
    STORE: begin data = rVal2; addr = rVal1+imm; nextPc = pc+4; end
    AUIPC: begin data = pc+imm; nextPc = pc+4; end
  endcase
  ExecInst eInst = ?;
  eInst.iType = dInst.iType;
  eInst.rd = dInst.rd;
  eInst.data = data;
  eInst.addr = addr;
  eInst.nextPc = nextPc;
  return eInst;
endfunction

Write your code below. Please add arrows to indicate where your code fits in the execute function.

(B) (3 points) Could a store operation that calculated the address in the same manner as lwr be added without major changes to the RISC-V ISA?

Yes                  No

(C) (3 points) If you answered Yes to B then provide a brief summary of the required changes. If you answered No to B then explain why not.
Problem 5. Caches (24 points)

Consider the diagram to the right for a 2-way set associative cache to be used with our RISC-V processor. Each cache line holds a single 32-bit word of data along with its associated tag and valid bit (0 when the cache line is invalid, 1 when the cache line is valid).

(A) (2 points) RISC-V uses 32-bit byte addresses, A[31:0]. To ensure the best cache performance, which address bits should be used for the cache index? Which address bits should be used for the tag field?

address bits used for cache index: A[____:____]

address bits used for tag field: A[____:____]

(B) (2 points) Suppose the processor does a read of location 0x5678. Identify which cache location(s) would be checked to see if that location is in the cache. For each location, specify the cache way (A or B) and set (row) number (0 through 7). E.g., 3A for set 3, way A. If there is a cache hit on this access, what would be the contents of the tag data for the cache line that holds the data for this location?

cache location(s) checked on access to 0x5678: _______________________

cache tag data on hit for location 0x5678 (hex): 0x_____________

(C) (2 points) Assume that checking the cache on each read takes 1 cycle and that refilling the cache line on a miss takes an additional 8 cycles. If we wanted the average access time over many reads to be 1.1 cycles, what is the minimum hit ratio the cache must achieve during that period of time? You needn’t simplify your answer.

minimum hit ratio for 1.1 cycle average access time:_____________
(D) (5 points) Find the cache hit ratio for the following program. Because the loop is executed many times, you can ignore code before/after the loop and focus on the average behavior of a loop iteration. Assume the cache is empty before execution begins (all valid bits are 0) and that an LRU replacement policy is used. Remember the cache is used for both instruction and data accesses.

```assembly
.l = 0
.addi x4, x0, 0x100
.mv x1, x0
.lui x2, 1
    // x2 = 0x1000
loop: lw x3, 0(x4)
.addi x4, x4, 4
.add x1, x1, x3
.addi x2, x2, -1
.bnez x2, loop
.sw x1, 0x100(x0)
.unimp
    // halt
.l = 0x100
source:
    . = . + 0x4000
    // Set source to 0x100, reserve 0x1000 words
```

hit ratio: _____________

(E) (3 points) After the program of part (D) has finished execution what information is stored in set (row) 4 of the cache? Give the addresses for the two locations that are cached (one in each of the sections) or briefly explain why that information can’t be determined.

Addresses whose data is cached in set 4: _____________ and _____________

For the rest of this problem, the cache is modified so that it has a block size of 2 words per line and the number of words in the cache remains the same (i.e., the cache now has 4 lines per way).

(F) (2 points) Specify which bits of the address should be used for the block offset, the cache index, and the tag.

address bits used for block and byte offset: A[_____:_____]  
address bits used for cache index: A[_____:_____]  
address bits used for tag field: A[_____:_____]  

(G) (4 points) Find the hit ratio for the program in part (D) using this new cache configuration.

hit ratio: _____________

END OF PRACTICE QUIZ 2!
Appendix:

RISC-V Calling conventions:

<table>
<thead>
<tr>
<th>Symbolic name</th>
<th>Registers</th>
<th>Description</th>
<th>Saver</th>
</tr>
</thead>
<tbody>
<tr>
<td>a0 to a7</td>
<td>x10 to x17</td>
<td>Function arguments</td>
<td>Caller</td>
</tr>
<tr>
<td>a0 and a1</td>
<td>x10 and x11</td>
<td>Function return values</td>
<td>Caller</td>
</tr>
<tr>
<td>ra</td>
<td>x1</td>
<td>Return address</td>
<td>Caller</td>
</tr>
<tr>
<td>t0 to t6</td>
<td>x5-7, x28-31</td>
<td>Temporaries</td>
<td>Caller</td>
</tr>
<tr>
<td>s0 to s11</td>
<td>x8-9, x18-27</td>
<td>Saved registers</td>
<td>Callee</td>
</tr>
<tr>
<td>sp</td>
<td>x2</td>
<td>Stack pointer</td>
<td>Callee</td>
</tr>
<tr>
<td>gp</td>
<td>x3</td>
<td>Global pointer</td>
<td>---</td>
</tr>
<tr>
<td>tp</td>
<td>x4</td>
<td>Thread pointer</td>
<td>---</td>
</tr>
</tbody>
</table>