CMU 18-447 Introduction to Computer Architecture, Spring 2015
HW 1: Instruction Set Architecture (ISA)

Instructor: Prof. Onur Mutlu
TAs: Rachata Ausavarungnirun, Kevin Chang, Albert Cho, Jeremie Kim, Clement Loh

Assigned: Wed., 1/14, 2015
Due: **Wed., 1/28, 2015**

1 The SPIM Simulator [5 points]

As you work through this homework assignment and Lab 1, you may want to examine the execution of MIPS programs you write with a known good reference. For this purpose, it will be helpful to learn the basic operation of the SPIM simulator (and its graphical counterpart, `xspim`). The SPIM simulator is described in [P&H] and is available as `spim447` or `xspim447` on the ECE Linux workstations (`/afs/ece/class/ece447/bin`). Work though the "Getting Started with `xspim`" tutorial from [P&H] at `http://pages.cs.wisc.edu/~larus/xspim.pdf`. There is nothing to turn in for this question, but it is up to you to learn and use SPIM.

2 Big versus Little Endian Addressing [5 points]

Consider the 32-bit hexadecimal number 0xcafe2b3a.

1. What is the binary representation of this number in *little endian* format? Please clearly mark the bytes and number them from low (0) to high (3). (You may find the discussion on page 24 of the MIPS R4000 User's Manual at `http://www.ece.cmu.edu/~ece447/s15/lib/exe/fetch.php?media=mips_r4000_users_manual.pdf` helpful.)

2. What is the binary representation of this number in *big endian* format? Please clearly mark the bytes and number them from low (0) to high (3).

3 Instruction Set Architecture (ISA) [25 points]

Your task is to compare the memory efficiency of five different styles of instruction sets for the code sequence below. The architecture styles are:

1. A zero-address machine is a stack-based machine where all operations are done using values stored on the operand stack. For this problem, you may assume that its ISA allows the following operations:

   - PUSH M - pushes the value stored at memory location M onto the operand stack.
   - POP M - pops the operand stack and stores the value into memory location M.
   - OP - Pops two values off the operand stack, performs the binary operation OP on the two values, and pushes the result back onto the operand stack. The popped values are NOT stored back to the memory.

   Note: To compute A - B with a stack machine, the following sequence of operations are necessary: PUSH A, PUSH B, SUB. After execution of SUB, A and B would no longer be on the stack, but the value A-B would be at the top of the stack.

2. A one-address machine uses an accumulator in order to perform computations. For this problem, you may assume that its ISA allows the following operations:

   - LOAD M - Loads the value stored at memory location M into the accumulator.
   - STORE M - Stores the value in the accumulator into memory location M.
   - OP M - Performs the binary operation OP on the value stored at memory location M and the value present in the accumulator. The result is stored into the accumulator (ACCUM = ACCUM OP M).

3. A two-address machine takes two sources, performs an operation on these sources and stores the result back into one of the sources. For this problem, you may assume that its ISA allows the following operation:

   - OP M1, M2 - Performs a binary operation OP on the values stored at memory locations M1 and M2 and stores the result back into memory location M1 (M1 = M1 OP M2).

4. A three-address machine, in general takes two sources, performs an operation and stores the result back into a destination different from either of the sources.

   Consider

   (a) A three-address memory-memory machine whose sources and destination are memory locations. For this problem, you may assume that its ISA allows the following operation:

   - OP M3, M1, M2 - Performs a binary operation OP on the values stored at memory locations M1 and M2 and stores the result back into memory location M3 (M3 = M1 OP M2).

   (b) A three-address load-store machine whose sources and destination are registers. Values are loaded into registers using memory operations (The MIPS is an example of a three-address load-store machine). For this problem, you may assume that its ISA allows the following operations:

   - OP R3, R1, R2 - Performs a binary operation OP on the values stored at registers R1 and R2 and stores the result back into register R3 (R3 = R1 OP R2).
   - LOAD R1, M - Loads the value at memory location M into register R1.
   - STORE R2, M - Stores the value in register R2 into memory location M.

To measure memory efficiency, make the following assumptions about all five instruction sets:

- The opcode is always 1 byte (8 bits).
- All register operands are 1 byte (8 bits).
- All memory addresses are 2 bytes (16 bits).
- All data values are 4 bytes (32 bits).
- All instructions are an integral number of bytes in length.

There are no other optimizations to reduce memory traffic, and the variables A, B, C, and D are initially in memory. You are only allowed to use the following instructions: `LOAD`, `STORE`, `PUSH`, `POP`, `ADD`, and `SUB`. As described above, `PUSH` and `POP` are only applicable to stack-based machines. For the two-address machines, only `ADD` and `SUB` are given to you for answering the questions.

(a) Write the code sequences for the following high-level language fragment for each of the five architecture styles. Be sure to store the contents of A, B, and D back into memory, but do not modify any other values in memory.

```
A = B + C;
B = A + C;
D = A - B;
```

(b) Calculate the instruction bytes fetched and the memory-data bytes transferred (read or written) for each of the five architecture styles.

(c) Which architecture is most efficient as measured by code size?

(d) Which architecture is most efficient as measured by total memory bandwidth required (code+data)?

4 The MIPS ISA [40 points]

4.1 Warmup: Computing a Fibonacci Number [15 points]

The Fibonacci number F_n is recursively defined as

$$F(n) = F(n-1) + F(n-2),$$

where $F(1) = 1$ and $F(2) = 1$. So, $F(3) = F(2) + F(1) = 1 + 1 = 2$, and so on. Write the MIPS assembly for the `fib(n)` function, which computes the Fibonacci number $F($`n`$)$:

```
int fib(int n)
{
  int a = 0;
  int b = 1;
  int c = a + b;
  while (n > 1) {
    c = a + b;
    a = b;
    b = c;
    n--;
  }
  return c;
}
```

Remember to follow MIPS calling convention and its register usage (just for your reference, you may not need to use all of these registers):

- The argument `n` is passed in register $4.
- The result (i.e., `c`) should be returned in $2.
- $8 to $15 are caller-saved temporary registers.
- $16 to $23 are callee-saved temporary registers.
- $29 is the stack pointer register.
- $31 stores the return address.

A summary of the MIPS ISA is provided at the end of this handout, and a MIPS reference sheet is on the wiki at `http://www.ece.cmu.edu/~ece447/s15/lib/exe/fetch.php?media=mips_reference_data.pdf`. The MIPS architecture reference manual is also on the wiki at `http://www.ece.cmu.edu/~ece447/s15/lib/exe/fetch.php?media=mips_r4000_users_manual.pdf`.

4.2 MIPS Assembly for REP MOVSB [25 points]

Recall from lecture that MIPS is a Reduced Instruction Set Computing (RISC) ISA. Complex Instruction Set Computing (CISC) ISAs—such as Intel's x86—often use one instruction to perform the function of many instructions in a RISC ISA. Here you will implement the MIPS equivalent for a single Intel x86 instruction, REP MOVSB, which we will specify here.[1]

The REP MOVSB instruction uses three fixed x86 registers: ECX (count), ESI (source), and EDI (destination). The "repeat" (REP) prefix on the instruction indicates that it will repeat ECX times. Each iteration, it moves one byte from memory at address ESI to memory at address EDI, and then increments both pointers by one. Thus, the instruction copies ECX bytes from address ESI to address EDI.

(a) Write the corresponding assembly code in MIPS ISA that accomplishes the same function as this instruction. You can use any general purpose register. Indicate which MIPS registers you have chosen to correspond to the x86 registers used by REP MOVSB. Try to minimize code size as much as possible.

[1]The REP MOVSB instruction is actually more complex than what we describe. For those who are interested, the Intel architecture manual (found on the wiki at `http://www.ece.cmu.edu/~ece447/s15/doku.php?id=techdocs`) describes the MOVSB instruction on page 1327 and the REP prefix on page 1682. We are assuming a 32-bit protected mode environment with flat addressing (no segmentation), and a direction flag set to zero. You do not need to worry about these details to complete the homework problem.

(b) What is the size of the MIPS assembly code you wrote in (a), in bytes? How does it compare to REP MOVSB in x86 (note: REP MOVSB occupies 2 bytes)?

(c) Assume the contents of the x86 register file are as follows before the execution of the REP MOVSB:

```
EAX: 0xccccaaaa
EBP: 0x00002222
ECX: 0xFEE1DEAD
EDX: 0xfeed4444
ESI: 0xdecaffff
EDI: 0xdeaddeed
EBP: 0xe0000000
ESP: 0xe0000000
```

Now, consider the MIPS assembly code you wrote in (a). How many total instructions will be executed by your code to accomplish the same fuction as the single REP MOVSB in x86 accomplishes for the given register state?

(d) Assume the contents of the x86 register file are as follows before the execution of the REP MOVSB:

```
EAX: 0xccccaaaa
EBP: 0x00002222
ECX: 0x00000000
EDX: 0xfeed4444
ESI: 0xdecaffff
EDI: 0xdeaddeed
EBP: 0xe0000000
ESP: 0xe0000000
```

Now, answer the same question in (c) for the above register values.

5 Data Flow Programs [15 points]

Draw the data flow graph for the `fib(n)` function from Question 4.1. You may use the following data flow nodes in your graph:

- + (addition)
- > (left operand is greater than right operand)
- `Copy` (copy the value on the input to both outputs)
- `BR` (branch, with the semantics discussed in class, label the True and False outputs)

You can use constant inputs (e.g., 1) that feed into the nodes. Clearly label all the nodes, program inputs, and program outputs. Try to the use fewest number of data flow nodes possible.

6 DRAM Refresh [30 points]

Assume we are building a new supercomputer, Tartan, that has a DRAM-based memory system with the following configurations:

- The total capacity is 1 ExaByte (EB).
- The DRAM row size is 8 KiloByte (KB).
- The minimum retention time among all DRAM rows in the system is 64 ms. In order to ensure that no data is lost, every DRAM row is refreshed once per 64 ms.

For each calculation in this section, you may leave your answer in simplified form in terms of powers of 2 and powers of 10.

(a) How many DRAM rows does Tartan's memory system has?

(b) How many DRAM refreshes happen in 64ms?

(c) What is the total power consumption of DRAM refresh in 64ms? Hint: you will need to figure out how much power a refresh operation consumes. You can find useful information in the technical note by Micron `http://www.ece.cmu.edu/~ece447/s15/lib/exe/fetch.php?media=tn4704.pdf`. Use the current (IDD) numbers specified in the datasheet posted on the website `http://www.ece.cmu.edu/~ece447/s15/lib/exe/fetch.php?media=1gb_ddr3_sdram.pdf`. Clearly state all the assumptions and show how you derive the power numbers.

(d) What is the total energy consumption of DRAM refresh during a day?

7 Performance Metrics [10 points]

- If a given program runs on a processor with a higher frequency, does it imply that the processor always executes more instructions per second (compared to a processor with a lower frequency)? (Use less than 10 words.)
- If a processor executes more of a given program's instructions per second, does it imply that the processor always finishes the program faster (compared to a processor that executes fewer instructions per second)? (Use less than 10 words.)

8 Performance Evaluation [9 points]

Your job is to evaluate the potential performance of two processors, each implementing a different ISA. The evaluation is based on its performance on a particular benchmark. On the processor implementing ISA A, the best compiled code for this benchmark performs at the rate of 10 IPC. That processor has a 500 MHz clock. On the processor implementing ISA B, the best compiled code for this benchmark performs at the rate of 2 IPC. That processor has a 600 MHz clock.

- What is the performance in Millions of Instructions per Second (MIPS) of the processor implementing ISA A?
- What is the performance in MIPS of the processor implementing ISA B?
- Which is the higher performance processor: A B Don't know Briefly explain your answer.

9 Research Paper Summaries [20 points]

Please read the following handout on how to write critical reviews. We will give out extra credit that is worth 0.5% of your total grade for each good summary/review.

- Lecture slides on guidelines for reviewing papers.
  `http://www.ece.cmu.edu/~ece447/s15/lib/exe/fetch.php?media=onur-447-s15-how-to-do-the-paper-reviews.pdf`

(a) Write a half-page summary for the following paper:
Patt, "Requirements, Bottlenecks, and Good Fortune: Agents for Microprocessor Evolution," in Proceedings of the IEEE, 2001. `http://www.ece.cmu.edu/~ece447/s15/lib/exe/fetch.php?media=00964437.pdf`

(b) Write a half-page summary for **one** of the following papers:

- Moscibroda and Mutlu, "Memory Performance Attacks: Denial of Memory Service in Multi-Core Systems," in Proceedings of the USENIX Security, 2007. `http://users.ece.cmu.edu/~omutlu/pub/mph_usenix_security07.pdf`
- Liu et al., "RAIDR: Retention-Aware Intelligent DRAM Refresh," in Proceedings of the International Symposium on Computer Architecture, 2012. `http://users.ece.cmu.edu/~omutlu/pub/raidr-dram-refresh_isca12.pdf`
- Kim et al., "Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors", in Proceedings of the International Symposium on Computer Architecture, 2014. `http://users.ece.cmu.edu/~yoonguk/papers/kim-isca14.pdf`

MIPS Instruction Summary

Opcode	Example Assembly	Semantics
add	`add $1, $2, $3`	`$1 = $2 + $3`
sub	`sub $1, $2, $3`	`$1 = $2 - $3`
add immediate	`addi $1, $2, 100`	`$1 = $2 + 100`
add unsigned	`addu $1, $2, $3`	`$1 = $2 + $3`
subtract unsigned	`subu $1, $2, $3`	`$1 = $2 - $3`
add immediate unsigned	`addiu $1, $2, 100`	`$1 = $2 + 100`
multiply	`mult $2, $3`	`hi, lo = $2 * $3`
multiply unsigned	`multu $2, $3`	`hi, lo = $2 * $3`
divide	`div $2, $3`	`lo = $2/$3, hi = $2 mod $3`
divide unsigned	`divu $2, $3`	`lo = $2/$3, hi = $2 mod $3`
move from hi	`mfhi $1`	`$1 = hi`
move from low	`mflo $1`	`$1 = lo`
and	`and $1, $2, $3`	`$1 = $2 & $3`
or	`or $1, $2, $3`	`$1 = $2 | $3`
and immediate	`andi $1, $2, 100`	`$1 = $2 & 100`
or immediate	`ori $1, $2, 100`	`$1 = $2 | 100`
shift left logical	`sll $1, $2, 10`	`$1 = $2 << 10`
shift right logical	`srl $1, $2, 10`	`$1 = $2 >> 10`
load word	`lw $1, 100($2)`	`$1 = memory[$2 + 100]`
store word	`sw $1, 100($2)`	`memory[$2 + 100] = $1`
load upper immediate	`lui $1, 100`	`$1 = 100 << 16`
branch on equal	`beq $1, $2, label`	`if ($1 == $2) goto label`
branch on not equal	`bne $1, $2, label`	`if ($1 != $2) goto label`
set on less than	`slt $1, $2, $3`	`if ($2 < $3) $1 = 1 else $1 = 0`
set on less than immediate	`slti $1, $2, 100`	`if ($2 < 100) $1 = 1 else $1 = 0`
set on less than unsigned	`sltu $1, $2, $3`	`if ($2 < $3) $1 = 1 else $1 = 0`
set on less than immediate	`sltui $1, $2, 100`	`if ($2 < 100) $1 = 1 else $1 = 0`
jump	`j label`	`goto label`
jump register	`jr $31`	`goto $31`
jump and link	`jal label`	`$31 = PC + 4; goto label`