# CMU 18-447 Introduction to Computer Architecture, Spring 2013
# HW 1 Solutions: Instruction Set Architecture (ISA)

Instructor: Prof. Onur Mutlu
TAs: Justin Meza, Yoongu Kim, Jason Lin

## 1 The SPIM Simulator [5 points]

There isn't a solution per se to this. The expectation is that you are familiar with SPIM/XSPIM, understand its strengths and limitations, and are using it to debug your labs and homeworks.

## 2 Big versus Little Endian Addressing [5 points]

1.

| 7d | ea | d3 | 21 |
|---|---|---|---|
| 0 | 1 | 2 | 3 |

2.

| 21 | d3 | ea | 7d |
|---|---|---|---|
| 0 | 1 | 2 | 3 |

## 3 Instruction Set Architecture (ISA) [25 points]

**Code size**:
Each instruction has an opcode and a set of operands

- 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 operands are 4 bytes (32 bits).
- All instructions are an integral number of bytes in length.

**Memory Bandwidth**:
Memory bandwidth consumed = amount of code transferred (code size) + amount of data transferred
Amount of data transferred = number of data references * 4 bytes

We will call the amount of code transferred as I-bytes and the amount of data transferred as D-bytes.

### (a), (b)

| Instruction Set Architecture | Opcode | Operands | I-bytes | D-bytes | Total Bytes |
|---|---|---|---|---|---|
| Zero-address | | | | | |
| | PUSH | B | 3 | 4 | |
| | PUSH | C | 3 | 4 | |
| | ADD | | 1 | | |
| | POP | A | 3 | 4 | |
| | PUSH | A | 3 | 4 | |
| | PUSH | B | 3 | 4 | |
| | ADD | | 1 | | |
| | POP | B | 3 | 4 | |
| | PUSH | A | 3 | 4 | |
| | PUSH | B | 3 | 4 | |
| | ADD | | 1 | | |
| | POP | D | 3 | 4 | |
| | | | 30 | 36 | 66 |

| Instruction Set Architecture | Opcode | Operands | I-bytes | D-bytes | Total Bytes |
|---|---|---|---|---|---|
| One-address | | | | | |
| | LOAD | B | 3 | 4 | |
| | ADD | C | 3 | 4 | |
| | STORE | A | 3 | 4 | |
| | ADD | B | 3 | 4 | |
| | STORE | B | 3 | 4 | |
| | LOAD | A | 3 | 4 | |
| | ADD | B | 3 | 4 | |
| | STORE | D | 3 | 4 | |
| | | | 24 | 32 | 56 |
| Two-address | | | | | |
| | SUB | A, A | 5 | 12 | |
| | ADD | A, B | 5 | 12 | |
| | ADD | A, C | 5 | 12 | |
| | ADD | B, A | 5 | 12 | |
| | ADD | D, D | 5 | 12 | |
| | ADD | D, A | 5 | 12 | |
| | ADD | D, B | 5 | 12 | |
| | | | 35 | 86 | 121 |
| Three-address Memory-Memory | | | | | |
| | ADD | A, B, C | 7 | 12 | |
| | ADD | B, A, B | 7 | 12 | |
| | ADD | D, A, B | 7 | 12 | |
| | | | 21 | 36 | 57 |
| Three-address Load-Store | | | | | |
| | LD | R1, B | 4 | 4 | |
| | LD | R2, C | 4 | 4 | |
| | ADD | R2, R1, R2 | 4 | | |
| | ST | R2, A | 4 | 4 | |
| | ADD | R3, R1, R2 | 4 | | |
| | ST | R3, B | 4 | 4 | |
| | ADD | R3, R1, R3 | 4 | | |
| | ST | R3, D | 4 | 4 | |
| | | | 32 | 20 | 52 |

**(c)** The three-address memory-memory machine is the most efficient as measured by code size - 21 bytes.

**(d)** The three-address load-store machine is the most efficient as measured by total memory bandwidth consumption (amount of code transferred + amount of data transferred) - 52 bytes.

## 4 The ARM ISA [40 points]

### 4.1 Warmup: Computing a Fibonacci Number [15 points]

*NOTE: More than one correct solution exists, this is just one potential solution.*

```
fib :
mov r1 , #0       // initialize the function
mov r2 , #1
add r3 , r1 , r2

branch :
cmp r0 , #0       // check if we are done
ble done
```

```
add r3 , r1 , r2  // fo fib(n) = fib(n-1)+fib(n-2)
mov r1 , r2       // increment the indices
mov r2 , r3
subs r0 , r0 , #1 // decrement the counter
b branch          // loopback

done :
mov r0 , r3
mov r15 , r14     // return
```

### 4.2 ARM Assembly for REP MOVSB [25 points]

(a)

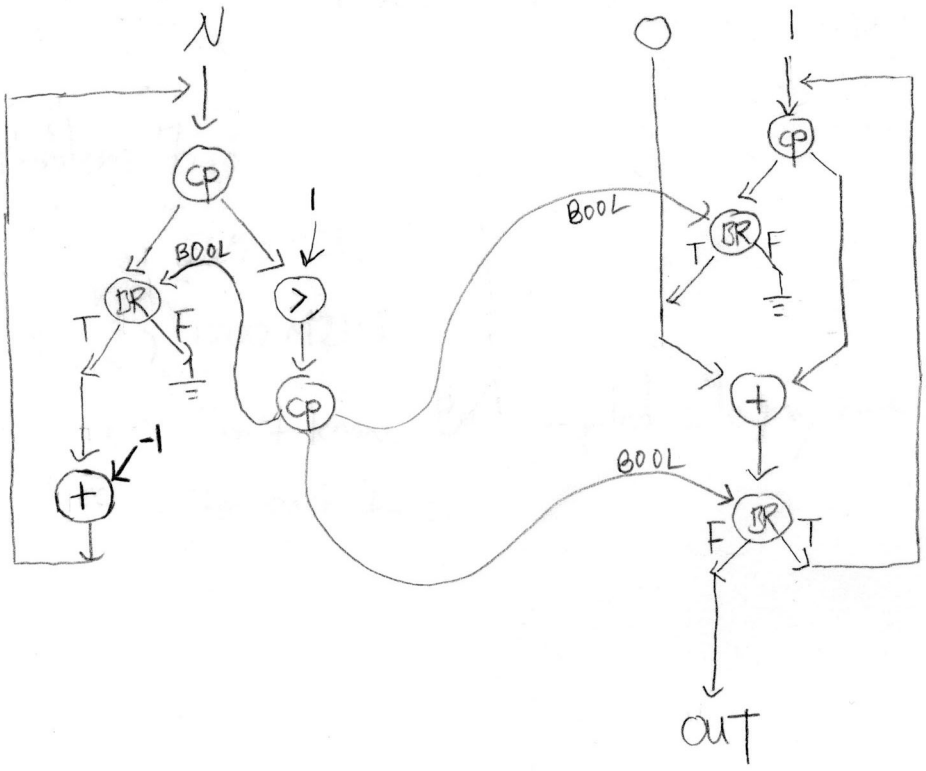
```
subs r0, r3, #1,    // check the condition
blt finish

copy :
ldrb r4, [r1] #1    // load 1 byte, move the source pointer to the next addr
strb r4, [r2] #1    // store (copy) 1 byte, move the destination pointer to the next addr
subs r3, r3, #1     // decrement the counter
bge copy

finish:
Following instructions
```

(b) The size of the ARM assembly code is 4 bytes $\times$ 6 = 24 bytes, as compared to 2 bytes for x86 REP MOVSB.

(c) The count (value in ECX) is 0xcafebeef = 3405692655. Therefore, loop body is executed (3405692655 * 4) = 13622770620 times. Total instructions executed = 13622770620 + 2 (instructions outside of the loop) = 13622770622.

(d) The count (value in ECX) is 0x00000000 = 0. In this case, the total instruction executed is 2 instructions (instructions outside the loop. Note that this can varies based on the assemble codes).

## 5 Data Flow Programs [15 points]




## 6 Performance Metrics [10 points]

- No, the lower frequency processor might have much higher IPC (instructions per cycle).

  More detail: A processor with a lower frequency might be able to execute multiple instructions per cycle while a processor with a higher frequency might only execute one instruction per cycle.
- No, because the former processor may execute many more instructions.

  More detail: The total number of instructions required to execute the full program could be different on different processors.

## 7 Performance Evaluation [15 points]

- ISA $A$: $6\frac{instructions}{cycle} * 400,000,000\frac{cycle}{second} = 2400$ MIPS
- ISA $B$: $2\frac{instructions}{cycle} * 800,000,000\frac{cycle}{second} = 1600$ MIPS
- Don't know.
  The best compiled code for each processor may have a different number of instructions.

## 8 Fixed Length vs. Variable Length [15 points]

**Code size**:
We can use Huffman encoding to encode the opcode

| Instruction | Opcode | numBits | Total Bits |
|---|---|---|---|
| SUB | 0 | 10 | 100 |
| ADD | 10 | 11 | 55 |
| BEQ | 110 | 17 to 41 (Assuming immediate is 8 to 32 bits) | 51 to 123 |
| MULT | 1110 | 10 | 20 |
| TOTAL | | | 226 to 298 |

In a traditional MIPS, this will take 640 bits (20 instructions, 32 bits each) So, the saving is 64.7% to 46.6%.