

CMU 18-447 INTRODUCTION TO COMPUTER ARCHITECTURE, SPRING 2013

HW 2: ISA TRADEOFFS

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

Assigned: Mon., 1/28, 2013

Due: **Mon., 2/11, 2013 (Midnight)**

Handin: `/afs/ece/class/ece447/handin/hw2`

1 LC-3b Microcode [80 points]

We wrote the microcode for at least one state of the LC-3b microarchitecture in class. In this homework, you will complete the microcode for all states of the LC-3b. Refer to Appendix C of Patt and Patel for the LC-3b state machine and datapath and Appendix A of Patt and Patel for the LC-3b ISA description.

Fill out the microcode in the `microcode.csv` file handed out with this homework. Enter a 1 or a 0 or an X as appropriate for the microinstructions corresponding to states. You do not need to fill out states 8, 10 and 11. We fill out state 18 as an example. Please turn in this CSV file electronically along with your homework.

2 Data Reuse in Different ISAs [30 points]

Consider two machines, X and Y , described below.

Machine X : A three-address memory/memory machine that takes two memory locations as source operands and one memory location as a destination operand. Machine X 's ISA has the following characteristics:

- **OP M3, M1, M2:** Performs the binary operation `OP` on the values stored at memory locations `M1` and `M2` and stores the result back into memory location `M3` ($M3 \leftarrow M1 \text{ OP } M2$).
- **OP M3, M1, Immediate:** Performs a binary operation `OP` on the value stored at memory location `M1` and value `Immediate` and stores the result back into memory location `M3` ($M3 \leftarrow M1 \text{ OP } \text{Immediate}$).
- Each instruction is encoded in 7 bytes.
- Data values are 4 bytes.

Machine Y : A three-address load/store machine whose sources and destination are registers (similar to MIPS). Values are loaded into registers using memory operations. Machine Y 's ISA has the following characteristics:

- **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` ($R3 \leftarrow R1 \text{ OP } R2$).
- **OP R3, R1, Immediate:** Performs a binary operation `OP` on the value stored at registers `R1` and immediate value `Immediate` and stores the result back into register `R3` ($R3 \leftarrow R1 \text{ OP } \text{Immediate}$).
- **LD R1, M:** Loads the value at memory location `M` into register `R1`.
- **ST R2, M:** Stores the value in register `R2` into memory location `M`.
- There are 32 general-purpose registers.
- Each instruction is encoded in 4 bytes.
- Memory data values are 4 bytes.

Consider the following two programs:

Program A

```
A = A + 1;
B = B + 2;
C = C + 3;
D = D + 4;
```

Program B

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

- (a) For each of the two programs, *A* and *B*, write the corresponding assembly code for each of the two machines, *X* and *Y*. For Machine *Y*, ensure to (i) **reuse** the register values whenever possible and (ii) store the register values back into memory after executing all the code.
- (b) Compute the total number of bytes (instruction *and* data) transferred to/from memory for each of the two programs, *A* and *B*, for each of the two machines, *X* and *Y*.

	Code A	Code B
Machine X		
Machine Y		

- (c) Between machines *X* and *Y*, (i) does the machine that transfers the smaller total number of bytes for Program *A* also transfer the smaller total number of bytes for Program *B*? (ii) If yes, explain why this is the case; otherwise, explain why there is a difference.

3 Fixed- vs. Variable-Length ISAs [20 points]

Consider the following two ISAs for a load/store machine.

1. The first is a fixed-length ISA that has the following instruction encoding.

Opcode	Operand 1 (Destination)	Operand 2 (Source 1) Reg/Imm	Operand 3 (Source 2) Reg/Imm
--------	-------------------------	---------------------------------	---------------------------------

- The opcode is 1 byte. Each operand is 1 byte.
- All register/register and register/immediate operations take 1 cycle, while all load and store instructions take 4 cycles.

2. The second is a variable-length ISA that the following instruction encoding.

Opcode	Operand 1 (Destination)	Operand 2 (Source 1) Reg/Imm	Operand 3 (Source 2) Reg/Imm (Optional)
--------	-------------------------	---------------------------------	---

- The opcode is 1 byte. Each operand (register/immediate) is 1 byte. **Note that operand 3 is optional, and this is implicitly specified in the opcode.** If an instruction does not need the third operand, it is not used.
- The variable length of this ISA increases its decode complexity. Therefore, all instructions take two cycles longer than the previously described fixed length ISA. In other words, all register/register and register/immediate operations take 3 cycles, while all load and store instructions take 6 cycles.

Consider the following assembly code:

```
ADD r3, r1, r2 // r3 = r1+r2
SLL r3, 0x2    // r3 = r3 << 2
```

```
MOV r5, 0xa    // r5 = 0x0a
STW r3, (r5)   // MEMORY[r5] = r3
```

- (a) What would be the code size for the given assembly code, for each of the two ISAs?
- (b) What is the number of cycles taken to execute the code sequence for each of the two ISAs?
- (c) Which of these two ISAs has lower code size for the given assembly code? Why?
- (d) Which of these two ISAs has lower execution time for the given assembly code? Why?

4 Addressing Modes [12 points]

We covered the following addressing modes in class:

- Absolute
- Register indirect
- Based (base + displacement)
- Scale indexed (base + index \times constant)
- Memory indirect
- Auto increment/decrement (by 1 byte)

Consider the following high-level programs:

- `uint8_t a[100]; // a is allocated in memory`
`for (i = 0; i < 100; i++) {`
 `a[i] = 10;`
`}`
- `int a[100]; // a is allocated in memory`
`for (i = 0; i < 100; i++) {`
 `a[i] = 10;`
`}`
- `int *p; // *p is allocated in memory`
`*p = 100;`
- `int **p; // *p and **p are allocated in memory`
`**p = 100;`

Assume that in the first two programs, a register contains the address of the start of the array, and in the last two programs, a register contains the value of `p`.

For each of the above three programs, which of the addressing modes, do you think, would lead to the minimum number of instructions? (Note that no addressing mode fits perfectly. You might require other instructions for address computation.)

5 Addressability [15 points]

Say we have 64 megabytes of storage. Calculate the number of bits required to address a location if:

- (a) (i) the ISA is bit-addressable
- (ii) the ISA is byte-addressable
- (iii) the ISA is 8-byte-addressable
- (iv) the ISA is 32-byte-addressable

- (b) A nibble is half of a byte (i.e., it contains 4 bits). Write a MIPS program that computes the sum of two nibbles: (1) the nibble at the offset stored in \$4 of the word located at the address stored in \$5 and (2) the nibble at the offset stored in \$6 of the word located at the address stored in \$7. You are only allowed to perform load/store accesses to addresses that are 4-byte (32-bit) aligned. Assume a little-endian format. The computed sum should be stored (zero-extended) in \$2.

6 Microarchitecture vs. ISA [20 points]

- (a) Briefly explain the difference between the microarchitecture level and the ISA level in the transformation hierarchy. What information does the compiler need to know about the microarchitecture of the machine in order to compile the program correctly?
- (b) Classify the following attributes of a machine as either a property of its microarchitecture or ISA:
- (i) The machine does not have a subtract instruction.
 - (ii) The ALU of the machine does not have a subtract unit.
 - (iii) The machine does not have condition codes.
 - (iv) A 5-bit immediate can be specified in an ADD instruction.
 - (v) It takes n cycles to execute an ADD instruction.
 - (vi) There are 8 general purpose registers.
 - (vii) A 2-to-1 mux feeds one of the inputs to ALU.
 - (viii) The register file has one input port and two output ports.

7 Single-Cycle Processor Datapath [30 points]

In this problem, you will modify the single-cycle datapath we built up in lecture to support the JAL instruction. The datapath that we will start with (from Lecture 5, Slide 53) has been reproduced on the next page. Your job is to implement the necessary data and control signals to support the JAL instruction, which we define to have the following semantics (note that we will not consider branch delay slots and so this definition will differ slightly from the one found in the MIPS R4000 user's manual):

$$\begin{aligned} \text{JAL : } \quad R31 &\leftarrow PC + 4 \\ PC &\leftarrow PC_{31..28} \parallel \text{Immediate} \parallel 0^2 \end{aligned}$$

Add to the datapath on the next page the necessary data and control signals to implement the JAL instruction. Draw and label all components and wires very clearly (give control signals meaningful names; if selecting a subset of bits from many, specify exactly which bits are selected; and so on).

