

## CSSE7013 - Advanced Computer Architecture Tutorial 2

### Learning Objectives

This tutorial aims to help you to understand approaches to instruction set design.

### QUESTION 1—ISA Design Alternatives

For each of the following instruction set types (example from H&P Fig. 2.2), illustrated with the example of machine code for  $C = A+B$  under the assumptions

- there are 32 registers, and registers are encoded with the minimum number of bits
- allowance is made for encoding up to 64 operations
- memory addresses in instructions require 20 bits

stack	accumulator	load-store
push A	load A	load R1, A
push B	add B	load R2, B
add	store C	add R3,R1,R2
pop C		store R3,c

- Work out the *minimum* bits each instruction requires.
- Now, assuming a load-store architecture always has the same number of bits per instruction but the other types can use the minimum bits rounded up to a byte, work out the length of the code in each example.
- Now work out (assuming each memory reference moves 32 bits of data) the total amount of data movement for each example. You may assume that the stack is maintained in the CPU, i.e., only push and pop operations access memory.
- Taking all the above into account:
  - Which design has the fewest instruction bits?
  - Which design has the lowest amount of data traffic? Would your answer differ in a longer computation, where variables in the program were used more than once (for example,  $C = A+B$ ;  $D = B - C + A$ )?
  - Which design has the lowest overall memory traffic? Again, consider how this may change if you had a more complex situation where variables were reused.

### QUESTION 2—Addressing Modes

In the MIPS architecture, there is only one addressing mode for RAM-based data access: displacement. There is also an immediate mode of 16 bits (and also some PC-relative modes for branches and jumps). Note also that MIPS is a load-store architecture.

- How can a 64-bit address be constructed using these modes?
- How can each of the following modes be implemented using the MIPS modes?
  - Register indirect (use a register as an address), e.g., `ADD R3, (R1)` in which R1 contains a memory address.
  - Direct or absolute, e.g., `ADD R3, 42` in which 42 is a memory address.
  - Memory indirect, e.g., `ADD R3, @(R3)` in which R3 is a memory address of a pointer to the actual memory location.

### QUESTION 3—Control Flow

- In MIPS, a condition is set in much the same way as any arithmetic, by storing a result in a register. *Condition codes* are status bits set in the CPU by a large fraction of instructions. For example, if you do a subtraction, a condition code for “zero” would be set if the result was 0. If the result was negative, there could be a condition code for “negative”. The condition code would be tested by a branch instruction, for example, `bzero` would branch if the condition code “zero” was set.
  - Write out a sequence of instructions for the MIPS approach for the example:

```
if (a<b)
    b=c;
else
    b=a;
```
  - Write out a condition code version of the same example. Make reasonable assumptions about what the assembly language would look like.
  - Which is shorter?
  - If you had to change the order of instructions for some reason, which limits possibilities more?
- A conditional transfer of control is called a branch, and unconditional one a jump. Why is it useful to distinguish the two cases?
- What addressing modes are commonly used for control-flow instructions? Why might these differ from data movement instructions?

### QUESTION 4—More Exercise

- Work through the examples in Chapter 2.
- Work through questions at the end of Chapter 2. Warning: the exercises in the book can sometimes be hard e.g. because details are left out, requiring you to make assumptions.