

CSSE7013 - Advanced Computer Architecture Tutorial 2 Partial Solutions

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
this means 5 bits per register field
- allowance is made for encoding up to 64 operations
this means 6 bits for the opcode
- memory addresses in instructions require 20 bits
this means any memory-referencing instruction needs 20 bits (we make no assumption as to whether this is the entire address, a displacement plus a register, or some other mode).

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

- a. Work out the *minimum* bits each instruction requires.
stack push and pops have an opcode + address = 26 bits; add is just 6 bits for opcode
all given accumulator instructions are opcode+address=26 bits
load-store loads and stores are opcode+register+address=31 bits
- b. 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.
stack push, pop: 32 bits; add 8 bits; total: 3x32+8=104 bits
accumulator: 3x32 bits = 96 bits
load-store: 4x32 bits = 128 bits
- c. 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.
all are: 3x32 = 96 bits
- d. Taking all the above into account:
 - i. Which design has the fewest instruction bits?
accumulator
 - ii. 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, $C = A+B$; $D = B - C + A$)?
All the same in this case. Expect Load-store to have less data traffic in the more complex example because it can reuse values once they are in registers.
 - iii. 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.
Accumulator. Stack could be more competitive in some cases because the arithmetic instructions have no operands. Load-store would only win with a lot of repetitive use of variables.

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.

- a. How can a 64-bit address be constructed using these modes?
You could do 4 immediate "loads" (faked by something like ADD R1,R0,#value; here to be different, we use OR – should be the same result except we don't have to worry about managing the sign bit), shifting the data as it was loaded, then adding it onto the new total, something like this:

OR R1,R0,#value3 ; the high-order 16 bits (R0 always = 0)

0x0000	0x0000	0x0000	value3
--------	--------	--------	--------

SLL R1,R1,16 ; shift left 16 bits

0x0000	0x0000	value3	0x0000
--------	--------	--------	--------

OR R1,R1,#value2 ; the next 16 bits

0x0000	0x0000	value3	value2
--------	--------	--------	--------

SLL R1,16 ; shift left 16 bits

0x0000	value3	value2	0x0000
--------	--------	--------	--------

OR R1,R1,#value1 ; the next 16 bits

0x0000	value3	value2	value1
--------	--------	--------	--------

SLL R1,R1,16 ; shift left 16 bits

value3	value2	value1	0x0000
--------	--------	--------	--------

OR R1,R1,#value0 ; the lowest 16 bits

value3	value2	value1	value0
--------	--------	--------	--------

Exercise: find out if MIPS has more efficient options for creating 64-bit constants. It has a more efficient way of creating a 32-bit constant: a load upper immediate instruction which loads a 16-bit value into the high half of a register. This allows creating a 32-bit value in 2 steps: LUI gets the high half, followed by an OR immediate as we did here to create the low half.

- b. 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.
Use the standard MIPS mode with displacement 0, i.e.,
ADD R3, 0(R1)
 - Direct or absolute, e.g., ADD R3, (42) in which 42 is a memory address.
Use immediate mode to get the address into a register, e.g. R1, then it reduces to case (i).
 - Memory indirect, e.g., ADD R3, @(R3) in which R3 is a memory address of a pointer to the actual memory location.
Load the value pointed to by R3 into another register, e.g., R1, then you have the same case as (i).

QUESTION 3—Control Flow

- a. 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.
Left for further exercise.
- 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?
- b. A conditional transfer of control is called a branch, and unconditional one a jump. Why is it useful to distinguish the two cases?
A branch not only requires its destination effective address to be computed, but whether it is taken or not must be determined, with 2 different outcomes. Also, branches are often used for short jumps (loops, if statements) whereas an unconditional jump may be used for longer-range transfer of control (e.g., return from call) and so the two tend to attract different addressing modes. Can you name them? See (c) below ...
- c. 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.