Week 1 Problem Set

Course Introduction and Instruction Set Architecture

Contents

Reference: Instruction Set Architectures

This section describes three different types of instruction set architectures and exemplifies a subset of their instructions.

Accumulator ISA

An example of an accumulator architecture has a set of accumulator registers. The operands of an instruction are implicitly specified by the instruction used.

Table 1 – Examples of accumulator instructions

Instruction	Description	Operation
ADD.A	Add a value from memory to A.	`A += Mem[Address]`
MOV.A	Store the value in A to memory.	`Mem[address] = A`
LOAD	Load a value from memory into the accumulator.	`A = Mem[address]`
NEGATE.A	Negate the value in A.	`A = -A`

Aside: Learn more about x86 and its accumulator register (the register: eax).

Stack ISA

An example of a stack architecture contains a hardware stack made up of a set of registers. The operands to an instruction are implicitly specified by the top of the stack.

Table 2 – Examples of stack instructions

Instruction	Description	Operation
PUSH	Load a value from memory to the top of the stack.	`S[++top] = Mem[Address]`

ADD	Add the top two values of the stack, pop them off, and place the result at the top of the stack	`temp1 = S[top--]`
		`temp2 = S[top--]`
		`S[++top] = temp1 + temp2`

POP	Store the value at the top of the stack into memory	`Mem[address] = S[top--]`

Load-store ISA (register-register)

An example of a load-store architecture contains a set of addressable registers. Instructions explicitly specify source and destination operands by indicating register addresses.

Table 3 - Examples of load-store instructions

Instruction	Description	Operation
LW	Load a value from memory to a register rd.	`[rd] = Mem[Address]`
ADD	Add the values found in registers rs and rt to register rd.	`[rd] = [rs] + [rt]`
SW	Store the value found in register rt to memory.	`Mem[Address] = [rt]`

Problems: Speedup

You have been transported back in time to the 1960s and find yourself as the head of IT in a software company, ABC Industries. The company would like to purchase an IBM System/360, but it is unsure of which model to buy.

The Model 30 implements floating-point arithmetic in software. But the Model 70 adds dedicated hardware for floating-point arithmetic. IBM estimates that the Model 70 reduces the time taken by floating-point instructions by a factor of 10 compared to the Model 30.

Let F_{fp} be the fraction of execution time the software application at ABC Industries spends on floating-point instructions. What range of values should F_{fp} be for the Model 70 to yield a speedup of at least 2 over the Model 30?

Variations of this problem

Consider changing the "factor of 10" in the above. Or consider making the speedup factor a variable and set the fraction of execution time to a specific value.

Problems: Comparing Instruction Set Architectures

This problem set is adapted from a question in: *Dubois, Michel, Murali Annavaram, and Per Stenström. Parallel computer organization and design. cambridge university press, 2012.*

We can compare instruction set architectures in terms of their code size (in bytes). Consider the high-level code below:

```
A = A * B * D

C = C – A + B
```

Compile

Compile the high-level code for the accumulator, stack, and load-store ISAs. Compare each ISA in terms of the static instruction count. You can assume that instructions not listed in the reference exist, within reason (e.g., a multiply instruction).

Code size

Based on your answer in Compile, calculate the size of the code in bytes for each ISA. For this question, the code size is simply the sum of the opcode and any operands. You should assume that:

- All opcodes require 8 bits
- All memory address operands require 16 bits
- For the load-store architecture, register address operands require 4 bits.

Demand on data memory

For each ISA, based on your answer in Compile, calculate the total amount of data (in bytes) that needs to be transferred from memory to the CPU for data (ignore the memory needed for instructions). You should assume that all data operands are 4 bytes long.

Evaluate

Use your understanding of the architectures, and the metrics calculated in the previous sub-sections, to compare and contrast the different ISAs. Your answer should go beyond the “ISA 1 is better for code size than ISA 2” by explaining why that is the case.

Variations of this problem

Consider creating your own variation of this problem by coming up with your own high-level code. For example, assume that none of the ISAs contain a multiply instruction, then think about a simple software algorithm for multiplication and translate it for each architecture (this would require some branching instructions).