COMP15111 Notes

Chris Williamson

October 18, 2013

Contents

1 Lecture 1:Introduction

1.1 A Computational Model

The simplest, earliest, commonest, most important model is the **Von-Neumann Imperative Procedural Computer Model**

A computer can:

1. Store information
2. Manipulate the stored information
3. Make decisions depending on the stored information

1.2 Simple View Of A Computer

$$Memory \Leftrightarrow Bus \Leftrightarrow Processor$$

1.2.1 Memory

- A set of locations which can hold information, such as numbers(or programs)
Each memory location has a unique (numerical) address
Typically thousands of millions of different locations
There are various ways of depicting memory; a common one is a 'hex dump':
Each item has a unique address.

1.2.2 Bus

- Bidirectional communication path
Transmits addresses and numbers

1.2.3 Processor

- Obeys a sequence of instructions = program (code)
Historically the processor was often referred to as a CPU
Nowadays this is inappropriate - typically several processing 'cores'

1.3 Three-address instructions

Every kind of processor has a different set of instructions, real world examples include: pentium, ARM....and others
We will start with something simple
Each 3-address instruction:

1. Copies the values from any two memory locations and sends them to the processor (source operands)
2. Copies some operation e.g. adds the copied numbers together

3. Copies the result back from the processor into a third memory location (destination operand)

e.g. sum of two values 'a' and 'b'
a holds 2, b holds 3 "source operands"
2+ 3 is 5 "operation"
Sum becomes 5 "destination operand"

1.4 Registers

Reduce accesses to main memory by using registers:

1. Very small, very fast memory within the processor
2. Each register can hold a single value

Processors contain up to a few dozen registers e.g. ARM: 16 registers - R0 to R15

1.5 Instruction Styles

Make instructions as simple as possible while allowing for the best use of registers
One-address style (e.g. Intel Pentium):

1. Only use (at most) 1 memory location in each instruction
2. Use registers for the other operands

$$register \leftarrow register + memorylocation$$

Load-store style (e.g. ARM): Use registers for all three operands

$$register \leftarrow register + register$$

So we need extra instructions to get at memory locations:

1. **Load** (from memory): register ← memory location
2. **Store** (to memory): memory location → register

Sum = a + b + c;
Becomes:
Register1 ← a (i.e. load from a)
Register2 ← b (i.e. load from b)
Register3 ← register 1 + register2 (i.e. a+b)
Register4 ← c (i.e. load from c)
Register5 ← register3 + register4 (i.e. (a+b)+c)
sum → register5 (i.e. store to sum)
One load or store instruction per variable (a, b, c, sum)
One arithmetic instruction per operation (+, +)
Lots of very simple, very fast instructions

2 Lecture 2

Computers obey program which are sequences of instructions
Instructions are coded as values in memory
The sequences are held in memory adjacent memory locations
Values in memory can be interpreted as:

1. Numbers (in several different ways)
2. Instructions
3. Text
4. Colours
5. Music
6. Anything you want

Values are often represented as numbers for convenience.

2.1 Assembly Language

Assembly language is a means of representing machine instructions in a human readable form.
Each type of processor has its own assembly language but they typically have a lot in common:

1. A mnemonic specifies the type of operation
2. A destination a register on this case
3. And one or more sources also registers
4. Possibly with a comment too

2.2 ARM instructions

ARM has many instructions but we only need three categories:

1. Memory operations
2. Processing operations
3. Control flow

Memory operation move data between the memory and the registers
Processing operations perform calculations using value already in registers
Control flow instructions are used to make decisions, repeat operations etc.

2.3 ARM memory instructions

Memory operations load a register from the memory or store a register value to the memory
e.g. LDR R1, a means: R1 $\leftarrow$ a
e.g. STR R5, sum means: R5 $\rightarrow$ sum (i.e. sum $\leftarrow$ R5)
a and sum are aliases for the addresses of memory locations

2.4 ARM processing instructions

Processing operations such as addition, subtraction, multiplication.
e.g. ADD R3, R1, R2 means: R3 $\leftarrow$ R1 + R2

2.5 ARM control instructions

Fundamentally, these are branches to other code sequences.
Often, branches are made conditional to allow decisions to be made.
e.g. B somewhere means: branch to somewhere
e.g. BEQ elsewhere means: branch to elsewhere IF previous result was equals
e.g. BNE wherever means: branch to wherever IF previous result was not equal

2.6 Stored programs and the Program Counter

A computer can make decisions, and choose which instructions to obey next depending upon the results of those decisions.
How? First we need to see how the sequence of instructions is controlled.