Instruction Sets Notes

Charlie Britton

November 2020

Contents

1 Introduction

Instruction sets help computers to understand what to do with the binary they're given. You can have many different types of data displayed within a stream of bytes, such as a char, an integer.

Take for example the bytes 0100 0010. This byte can be represented as ASCII B, a memory address, an instruction, or anything else (provided we know how to decode it).

2 What is an instruction set?

Instruction sets are a collection of instructions that are understood by the CPU, and understand the machine code or binary they are given. Instructions are usually represented as an assembly language (which we provide as a *mnemonic opcode*, such as `ADD`, `SUB`, or `MUL`. One line of an assembly program will typically produce one machine instruction.

2.1 The Elements of an Instruction

A typical CPU instruction is made up of the **opcode**, the **source operand reference**, a **result reference** and a **next instruction reference**. It is

important to know that not every instruction will do all of the above, although an opcode will always be required.

Once the CPU receives an instruction, it will generate the desired control inputs and flags for the ALU, which it will then feed the correct data. The ALU will execute a wide range of instructions and on the 74181 ALU, you provide it with a pattern of 4 S bits, an M bit and a C bit.

Every instruction has it's own bit pattern, and if programmers had to remember each different bit pattern it would all get very confusing very quickly and we'd be very prone to mistakes. Therefore, we use a symbolic mapping as it is much harder to confuse `ADD` and `MUL` than it is to remember which of 1001 0010 and 1010 0010 refers to which instruction. Operands can also be implicitly references, e.g. `R0` as opposed to the direct binary address.

2.2 What do operands reference?

They can interface with the main memory, which *may be* the virtual memory, a CPU register (special fast bit of memory), the set of all registers is known as the register file. Operands can also read and write to specific I/O devices, such as screens or speakers. It is important to note that nowadays, especially with very file-based operation systems like Linux, you will write to a virtual memory space and then the hardware which is optimized for moving around large chunks of data will write that to for example an audio buffer `/dev/audio`. In the simplest case we will just have the one register to reference and this will be implicit.

2.3 Different Types of Instructions

Instructions can be broadly grouped into four categories. These are *data processing*, which encompasses the use of the ALU to produce a result, *data storage*, which handles bringing things into and out of the main system memory, *program flow control* such as `JUMP` and *data movement*, which handles I/O such as speakers, mice, keyboards and monitors.

3 Instruction Set Formats

As with most things in the world, there is no one set way to go about how something is done. Therefore, different manufacturers make use of different type of instruction set architectures if they think it will give them a commercial advantage or they have good techniques for manufacturing that specific instruction. A few of the most common ones are listed below.

3.1 The Simple Instruction Set

In the simple instruction set, each instruction is 2 bytes, or 16 bits, with 4 bits dedicated to the opcode, 6 bits to operand 1 and 6 bits to operand 2.