Chapter 10 – Instruction Sets: Characteristics and Functions

Reading: 10.1 Machine Instruction Characteristics pp. 336-342

We have already learned that an instruction is composed of a series of bytes where a portion of the instruction is for the opcode and the other portion is for one or more operands.

Simple opcodes include: ADD, SUB, MUL, DIV, MOV, ...

A simple instruction format might be:

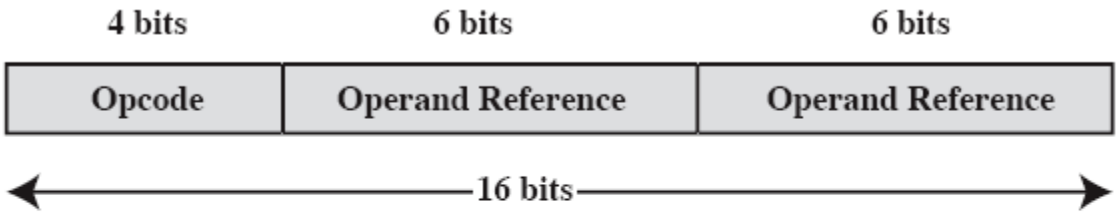

Figure 10.2 A Simple Instruction Format

We have spent most of our time in the high-level programming world. A high-level language must eventually be translated into some kind of machine language usually through some assembly language.

Machine language instructions typically fall into one of four categories:

1. Data Processing: Arithmetic and logical instructions
2. Data Storage: Memory instructions
3. Data Movement: I/O instructions
4. Control: Test and branch instructions

Historically, processor architectures have been defined in terms of the number of addresses contained within the instruction.

For the expression below, what machine level expressions would be used to compute it? We'll do this for 3-address, 2-address, and 1-address instructions:

$$Y = (A - B)/[C + (D * X)]$$

What do the number of addresses imply?

Fewer addresses:

- More primitive instructions requiring a less complex processor
- Instructions are shorter in length
- Programs contain more instructions
- Execution times are generally longer

Q1: What is the major disadvantage of a one-address instruction set?

When designing an instruction set, the following fundamental design issues must be answered:

- Operation repertoire – which operations to provide, how many & how complex
- Data types – which types of data are represented and which operations operate on which types of data
- Instruction format – what is the instruction length, number of addresses, and sizes of various fields
- Registers – how many registers exist and what are their uses
- Addressing – what are the possible addressing modes for a particular operand