Chapter 2: Assembly Language Fundamentals

Prof. Ben Lee

Oregon State University

School of Electrical Engineering and Computer Science

1

Chapter Goals

- Understand the importance of assembly language.
  - ◦ The interface between hardware and software.
- Understand how assembly instructions are represented.
  - ◦ What and how much information is encoded in the instructions.
  - ◦ How these design choices affect the instruction format, and thus code density and power the power of the instructions

2

Contents

3

2.1 Introduction

4

Assembly Language

I speak Spanish to God,
Italian to women,
French to men,
And German to my horse

Charles V, King of France (1337-1380)

(An excerpt from Computer Organization & Design: The Hardware/Software Interface by Patterson and Hennessy)
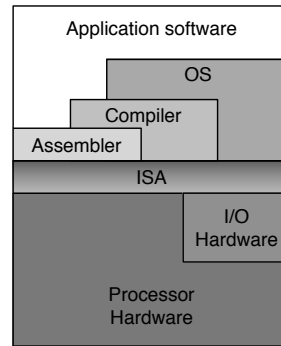
5

2.2 How Do We Speak the Language of the Machine

6

How Do We Speak the Language of the Machine?

- Instruction Set Architecture (ISA) is the portion of the machine visible to the programmer or compiler writer.
  - The basic operations of a computer are defined by its ISA.
  - Because instructions are the only information that the machine understands, an instruction set is also a machine language.
  - Sequences of instructions are called machine language programs, and the act of constructing such programs is machine language programming.

Application software	
	OS
	Compiler
Assembler	
ISA	
Processor Hardware	I/O Hardware

7

2.3 Instruction Set Architecture

8

4

Classification of ISA

- Four dimensions of instruction sets:
  - Operations provided in the instruction set.
  - Number of explicit operands named per instruction.
  - Location of the operands in the CPU and how operand locations are specified.
  - Type and size of operands.

9

Operations in the ISA

- Important criteria for designing an instruction set are
  - Functional completeness
  - Efficiency (power of the instruction)
  - Simplicity of hardware design and/or programming

10

Categories of Instructions

1. **Data transfer** - which cause information to be copied from one location to another either in the processor's internal memory (registers) or in the external main memory.
2. **Arithmetic** - which perform operations on numeric data (e.g., add, subtract....).
3. **Logical** - which include Boolean and other non-numerical operations (e.g., and, not....)
4. **Control transfer** - which change the sequence in which program are executed (jump, conditional branch...)
5. **I/O** - which cause information to be transferred between the processor or its main memory and external I/O devices.
6. System - used for operating system call, virtual memory management instructions....
7. Floating point - used for floating point operations.
8. Decimal - decimal arithmetic, decimal-to-character conversions...
9. String - string move, compare, search....

11

Number of Operands per Instruction

- The number of operands associated with each instruction can be considered in terms of the following issues:
  - Control circuit complexity (decoding).
  - Storage required for instructions (code density).
  - Power of instructions.
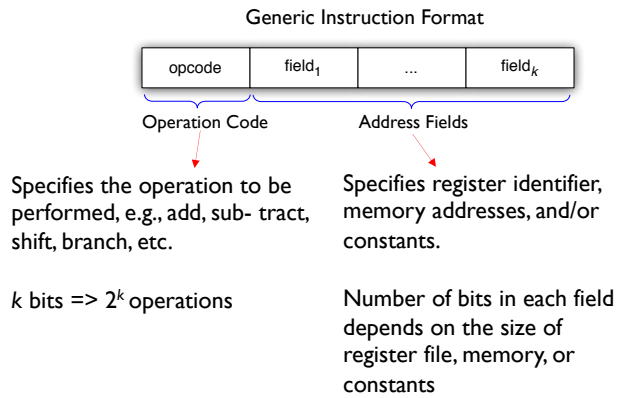 - Number of instructions required to perform a given task.

12

Example

Let *x* and *y* represent memory locations or registers.

- 4-address instruction: No longer in use

  ADD $x + y \rightarrow z$, goto q

- 3-address instruction: General Purpose Register (GPR) architecture

  ADD $x + y \rightarrow z$ (q is implied)

  ADD $x + y \rightarrow x$, goto q

- 2-address instruction: General Purpose Register (GPR) architecture

  ADD $x + y \rightarrow x$

- 1-address instruction: Accumulator-based architecture

  ADD x (accumulator based)

- 0-address instruction: Stack architecture

  ADD   (use of a stack)

13

2.4 Instruction Format

14

Instruction Format

Generic Instruction Format

opcode	field$_1$	...	field$_k$

Operation Code   Address Fields

Specifies the operation to be performed, e.g., add, sub- tract, shift, branch, etc.

k bits => 2^k operations

Specifies register identifier, memory addresses, and/or constants.

Number of bits in each field depends on the size of register file, memory, or constants

Tradeoff among sizes of the various fields!

15

2.4 A Pseudo-ISA

16

A Pseudo-ISA

- Data transfer Instructions
    - ◦ LDA (Load Accumulator) – Loads a memory word to accumulator
        - · LDA x; x is a location in memory
    - ◦ STA (Store Accumulator) – Stores the contents of accumulator to memory
- Arithmetic and Logical Instructions
    - ◦ ADD (Add to Accumulator) – Adds the content of memory word specified by the effective address to the value of the accumulator.
        - · ADD x; x is a location in memory
    - ◦ NAND (NAND to Accumulator)
    - ◦ SHFT (Shift Accumulator) - Shifts the content of AC by one bit to the left. The bit shifted in is 0.
- Control Transfer
    - ◦ J (Jump to x) – Transfers the program control to the instruction specified by the address.
        - · J x; jump to instruction in memory location x
    - ◦ BNE (Branch Conditionally to x) – Transfers the program control to the instruction specified by the address based on condition NE (not equal).
        - · BNE x; branch to instruction in memory location x if content of AC is not zero
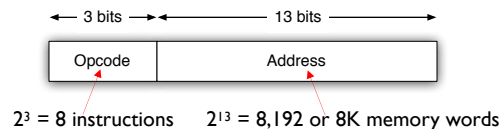
17

Instruction Format of Pseudo-ISA

1-address Instruction Format

←— 3 bits —→ ←———————— 13 bits ————————→

Opcode	Address

$2^3 = 8$ instructions $2^{13} = 8,192$ or 8K memory words

18

9

Example Assembly Program

```
; A simple C program
main()
{   int    A=83, B=-23, C=0;
    C = 4*A + B;
}
```

Label	Instructions		Comments
	ORG	0	/Origin of the program is location 0
	LDA	A	/Load operand from location A
	SHIFT		/Multiply A by 2
	SHIFT		/Multiply 2*A by 2
	ADD	B	/Add operand from location B
	STA	C	/Store sum in location C
Loop:	J	Loop	/Loop forever
A:	DEC	83	/Decimal operand
B:	DEC	-23	/Decimal operand
C:	DEC	0	/Sum stored in location C
	END		/End of symbolic program

Why Program in Assembly?

- For many applications the speed and/or size a program is critically important, e.g., *embedded application*, such as ABS.
    ◦ Need to respond rapidly and predictably to events, e.g., 1 ms after the sensor detects that a tire is skidding.
    ◦ Compiler introduces uncertainty about the time cost of operations. For complex problems, HLL preferable.

Questions?

21