CS1101: Lecture 34

The ISA Level: Instruction Sets

Dr. Barry O'Sullivan
b.osullivan@cs.ucc.ie

Course Homepage
http://www.cs.ucc.ie/~osullb/cs1101

Lecture Outline

- Instruction Types
  - Data Movement Instructions
  - Dyadic Operations
  - Monadic Operations
  - Comparisons and Conditional Branches
  - Procedure Call Instructions
  - Loop Control
  - Input/Output
- Instruction Sets
  - The Pentium II
  - The UltraSPARC II
- Flow of Control
  - Procedures
  - Coroutines
  - Traps
  - Interrupts
- **Reading**: Tanenbaum, Chapter 5, Sections 5 & 6.

Introduction

- ISA level instructions can be approximately divided into a half dozen groups that are relatively similar from machine to machine, even though they may differ in the details.
- In addition, every computer has a few unusual instructions, added for compatibility with previous models, or because the architect had a brilliant idea, or for some other reason.
- We will look at the instruction sets of two example machines, the Pentium II and the UltraSPARC.
- Each of these has a core of instructions that compilers normally generate, plus a set of instructions that are rarely used, or are used only by the operating system.
- We will focus on the common instructions.

Data Movement Instructions

- Copying data from one place to another is the most fundamental of all operations.
- By copying we mean the creating of a new object, with the identical bit pattern as the original.
- When we say that the contents of memory location 2000 have been moved to some register, we always mean that an identical copy has been created there and that the original is still undisturbed in location 2000.
- There are two reasons that data may be copied from one location to another.
  1. To support the assignment of values to variables.
  2. The second reason for copying data is to stage it for efficient access and use.

Dyadic Operations

- Dyadic operations are those that combine two operands to produce a result.
- All ISAs have instructions to perform addition and subtraction on integers.
- Multiplication and division of integers are nearly standard as well.
- It is presumably unnecessary to explain why computers are equipped with arithmetic instructions.
- Another group of dyadic operations includes the Boolean instructions.
- Although 16 Boolean functions of two variables exist, few, if any, machines have instructions for all 16. – usually, AND, OR, and NOT are present; sometimes EXCLUSIVE OR, NOR, and NAND are there as well.

Monadic Operations

- Monadic operations have one operand and produce one result.
- Because one fewer address has to be specified than with dyadic operations, the instructions are sometimes shorter, though often other information must be specified.
- For example, a monadic form of the ADD instruction if the INC instruction, which adds 1.
- The NEG operation is another example NEG X is really computing 0 - X.

Comparisons and Conditional Branches

- Nearly all programs need the ability to test their data and alter the sequence of instructions to be executed based on the results.
- A simple example is the square root function, $\sqrt{x}$.
- If x is negative, the procedure gives an error message; otherwise, it computes the square root.
- A function $sqrt$ has to test x and then branch, depending on whether it is ne-ative or not.
- A common method for doing so is to provide conditional branch instructions that test some condition and branch to a particular memory address if the condition is met.

Procedure Call Instructions

- A procedure is a group of instructions that performs some task and that can be invoked (called) from several places in the program.
- The term subroutine is often used instead of procedure, especially when referring to assembly language programs.
- In Java, the term used is **method**.
- When the procedure has finished its task, it must return to the statement after the call. Therefore, the return address must be transmitted to the procedure or saved somewhere so that it can be located when it is time to return.
- The return address may be placed in any of three places: memory, a register, or the stack.
- The ability for a procedure to call itself, called **recursion**, is important.

Loop Control

- The need to execute a group of instructions a fixed number of times occurs frequently and thus some machines have instructions to facilitate doing this.
- All the schemes involve a counter that is increased or decreased by some constant once each time through the loop.
- The counter is also tested once each time through the loop.
- If a certain condition holds, the loop is terminated.

Input/Output

- Considerable variety from machine to machine.
- Three different I/O schemes are in current use in personal computers:
  - Programmed I/O with busy waiting.
  - Interrupt-driven I/O.
  - DMA I/O.
- Programmed I/O CPUs usually have a single input instruction and a single output instruction – each of these instructions selects one I/O device.
- Interrupt-driven I/0 instructions transite characters with interrupts to control bus access.
- DMA (Direct Memory Access) IO uses a controller with direct access to the bus – DMA I/O instructions manage the interface between the controller and the IO devices.

The Pentium II

- The Pentium II instruction set is a mixture of instructions that make sense in 32-bit mode and those that hark back to its former life as an 8088.
- Many of the Pentium II instructions reference one or two operands, either in registers or in memory.
- The SRC fields are sources of information and are not changed.
- In contrast, the DST fields are destinations and are normally modified by the instruction.

The UltraSPARC II

- The UltraSPARC II really is a reduced instruction set computer.
- The UltraSPARC II architecture is a a **load/store architecture**
- That is, the only operations that access memory directly are LOADs and STOREs, instructions to move data between the registers and the memory.
- Operands come from registers and are saved in registers.

Flow of Control

- Flow of control refers to the sequence in which instructions are executed dynamically, that is, during program execution.
- **Procedure calls** cause the flow of control to be altered, stopping the procedure currently executing and starting the called procedure.
- **Coroutines** are related to procedures and cause similar alterations in the flow of control – useful for simulating parallel processes.
- **Traps and interrupts** also cause the flow of control to be altered when special conditions occur.

Procedures

- The most important technique for structuring programs is the procedure.
- When finished performing its task, a procedure returns control to the statement or instruction following the call.
- However, from another point of view, a procedure body can be regarded as defining a new instruction on a higher level.
- To understand a piece of code containing a procedure call, it is only necessary to know what it does, not how it does it.
- One particularly interesting kind of procedure is the **recursive procedure** – a procedure that calls itself, either directly or indirectly via a chain of other procedures.

Coroutines

- Sometimes it is useful to have two procedures, A and B, each of which calls the other as a procedure.
- When B returns to A, it branches to the statement following the call to B.
- When A transfers control to B, it does not go to the beginning (except the first time) but to the statement following the most recent ”return,” that is, the most recent call of A.
- Two procedures that work this way are called **coroutines**.
- A common use for coroutines is to simulate **parallel processing** on a single CPU.
- Each coroutine runs in pseudo-parallel with the others, as though it had its own CPU.

Traps

- A trap is a kind of automatic Procedure call initiated by some condition caused by the program, usually an important but rarely Occurring condition].
- A good example is overflow – if the result of an arithmetic operation exceeds the largest number that can be represented, a trap occurs.
- Flow of control is is switched to a **trap handler**.
- A trap handler performs some appropriate action, such as printing an error message.
- If the result of an operation is within range, no trap occurs.

Interrupts

- Interrupts are changes in the flow of control caused not by the running program, but by something else, usually related to I/0.
- For example, a program may instruct the disk to start transferring information, and set the disk up to provide an interrupt as soon as the transfer is finished.
- Like the trap, the interrupt stops the running program and transfers control to an interrupt handler, which performs some appropriate action.
- When finished, the interrupt handler returns control to the interrupted program.