UNIT I Computer Organization and Instructions

Instruction Sequencing

Introduction

The tasks carried out by a computer program consist of a sequence of small steps, such as adding two numbers, testing for a particular condition, reading a character from the keyboard, or sending a character to be displayed on a display screen. A computer must have **instructions** capable of performing four types of operations:

- Data transfers between the memory and the processor registers
- Arithmetic and logic operations on data
- Program sequencing and control
- I/O transfers

Right transfer Notation

We need to describe the transfer of information from one location in a computer to another. Possible locations that may be involved in such transfers are memory locations, processor registers, or registers in the I/O subsystem. Most of the time, we identify such locations symbolically with convenient names.

Example:

Names that represent the addresses of memory locations may be `LOC`, `PLACE`, `A`, or `VAR2`

Predefined names for the processor registers may be `R0` or `R5`.

Registers in the I/O subsystem may be identified by names such as `DATAIN` or `OUTSTATUS`.

Right transfer Notation cont..

To describe the transfer of information, the contents of any location are denoted by placing square brackets around its name. Thus, the expression

$$R2 \leftarrow [LOC]$$

means that the contents of memory location LOC are transferred into processor register R2.

Right transfer Notation cont..

As another example, consider the operation that adds the contents of registers R2 and R3, and places their sum into register R4. This action is indicated as

$$R4 \leftarrow [R2] + [R3]$$

This type of notation is known as **Register Transfer Notation** (RTN). Note that the right hand side of an RTN expression always denotes a value, and the left-hand side is the name of a location where the value is to be placed, overwriting the old contents of that location.

Assembly Language Notation

We need another type of notation to represent machine instructions and programs. For this, we use assembly language. For example, a generic instruction that causes the transfer described above, from memory location LOC to processor register R2, is specified by the statement

```
Load R2, LOC
```

The contents of LOC are unchanged by the execution of this instruction, but the old contents of register R2 are overwritten. The name Load is appropriate for this instruction, because the contents read from a memory location are loaded into a processor register.

RISC and CISC Instruction Sets (CISC)

An alternative to the RISC approach is to make use of more complex instructions which may span more than one word of memory, and which may specify more complicated operations. This approach was prevalent prior to the introduction of the RISC approach in the 1970s.

Although the use of complex instructions was not originally identified by any particular label, computers based on this idea have been subsequently called **Complex Instruction Set Computers** (CISC).

Instruction Execution and Straight-Line Sequencing

At the start of execution of a program, all instructions and data used in the program are stored in the memory of a computer. Processor registers do not contain valid operands at that time. If operands are expected to be in processor registers before they can be used by an instruction, then it is necessary to first bring these operands into the registers. This task is done by Load instructions which copy the contents of a memory location into a processor register. Load instructions are of the form

```
Load destination, source
```

Instruction Execution and Straight-Line Sequencing

Let us now consider a typical arithmetic operation. The operation of adding two numbers is a fundamental capability in any computer. The statement

```
C = A + B
```

in a high-level language program instructs the computer to add the current values of the two variables called `A` and `B`, and to assign the sum to a third variable, `C`. When the program containing this statement is compiled, the three variables, `A`, `B`, and `C`, are assigned to distinct locations in the memory. Using **Register Transfer Notation** this action is

$$C \leftarrow [A] + [B]$$

Instruction Execution and Straight-Line Sequencing

The required action can be accomplished by a sequence of simple machine instructions.
We choose to use registers R2, R3, and R4 to perform the task with four instructions:

```
Load R2, A
Load R3, B
Add R4, R2, R3
Store R4, C
```

Instruction Execution and Straight-Line Sequencing

We say that `Add` is a three-operand, or a three-address, instruction of the form

```
Add destination, source1, source2
```

The `Store` instruction is of the form

```
Store source, destination
```

where the source is a processor register and the destination is a memory location. Observe that in the **Store** instruction the source and destination are specified in the reverse order from the Load instruction; this is a commonly used convention.

Instruction Execution

Executing a given instruction is a two-phase procedure. In the first phase, called ***instruction fetch***, the instruction is fetched from the memory location whose address is in the **PC**. This instruction is placed in the ***instruction register*** (**IR**) in the processor. At the start of the second phase, called ***instruction execute***, the instruction in **IR** is examined to determine which operation is to be performed. The specified operation is then performed by the processor. This involves a small number of steps such as fetching operands from the memory or from processor registers, performing an arithmetic or logic operation, and storing the result in the destination location. At some point during this two-phase procedure, the contents of the **PC** are advanced to point to the next instruction.


Thank You
For Your
Attention
shutterstock.com · 1562318596