Instruction Set Architecture

(Interface between hardware and software)

Dr. Nausheen Shoaib

Types of Operations

Logical: Most machines also provide a variety of operations for manipulating individual bits of a word or other addressable units, often referred to as “bit twiddling.”

Some of the basic logical operations that can be performed on Boolean or binary data are shown in Table 12.6.

NOT operation inverts a bit. AND, OR, and Exclusive- OR (XOR) are the most common logical functions with two operands.

EQUAL is a useful binary test. These logical operations can be applied bitwise to n-bit logical data units.

Table 12.6 Basic Logical Operations

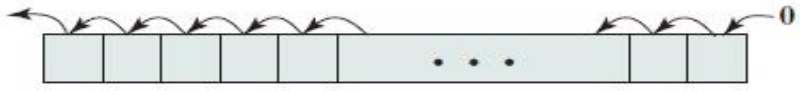
P	Q	NOT P	P AND Q	P OR Q	P XOR Q	P = Q
0	0	1	0	0	0	1
0	1	1	0	1	1	0
1	0	0	0	1	1	0
1	1	0	1	1	0	1

Types of Operations

Logical shift: the bits of a word are shifted left or right. On one end, the bit shifted out is lost. On the other end, a 0 is shifted in.

Logical shifts are useful primarily for isolating fields within a word. The 0s that are shifted into a word displace unwanted information that is shifted off the other end.

(a) Logical right shift



(b) Logical left shift

Types of Operations

Arithmetic Shift: Arithmetic shift operation treats the data as a signed integer and does not shift the sign bit.

On a right arithmetic shift, the sign bit is replicated into the bit position to its right.

On a left arithmetic shift, a logical left shift is performed on all bits but the sign bit, which is retained.

With numbers in two's complement notation, a right arithmetic shift corresponds to a division by 2, with truncation for odd numbers. Both an arithmetic left shift and a logical left shift correspond to a multiplication by 2 when there is no overflow.

If overflow occurs, arithmetic and logical left shift operations produce different results, but the arithmetic left shift retains the sign of the number.

(c) Arithmetic right shift

(d) Arithmetic left shift

Types of Operations

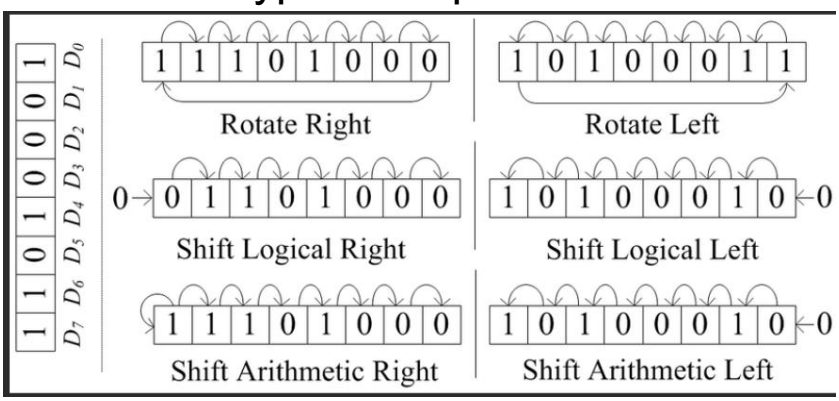
Rotate, or cyclic shift: operations preserve all of the bits being operated on. One use of a rotate is to bring each bit successively into the leftmost bit, where it can be identified by testing the sign of the data (treated as a number).

As with arithmetic operations, logical operations involve ALU activity and may involve data transfer operations. Table 12.7 gives examples of all of the shift and rotate operations.

Table 12.7 Examples of Shift and Rotate Operations

Input	Operation	Result
10100110	Logical right shift (3 bits)	00010100
10100110	Logical left shift (3 bits)	00110000
10100110	Arithmetic right shift (3 bits)	11110100
10100110	Arithmetic left shift (3 bits)	10110000
10100110	Right rotate (3 bits)	11010100
10100110	Left rotate (3 bits)	00110101

Types of Operations



Types of Operations

12.4 Many processors provide logic for performing arithmetic on packed decimal numbers. Although the rules for decimal arithmetic are similar to those for binary operations, the decimal results may require some corrections to the individual digits if binary logic is used.

Consider the decimal addition of two unsigned numbers. If each number consists of N digits, then there are $4N$ bits in each number. The two numbers are to be added using a binary adder. Suggest a simple rule for correcting the result. Perform addition in this fashion on the numbers 1698 and 1786.

Half-carry flag - Wikipedia

Types of Operations

12.5 The tens complement of the decimal number X is defined to be $10^N - X$, where N is the number of decimal digits in the number. Describe the use of ten's complement representation to perform decimal subtraction. Illustrate the procedure by subtracting $(0326)_{10}$ from $(0736)_{10}$.

Types of Operations

Conversion: Conversion instructions are those that change the format or operate on the format of data. An example of a more complex editing instruction is the EAS/390 Translate (TR) instruction. This instruction can be used to convert from one 8-bit code to another, and it takes three operands:

TR R1 (L), R2

Operand R2 contains the address of the start of a table of 8-bit codes.

L bytes starting at the address specified in R1 are translated, each byte being replaced by the contents of a table entry indexed by that byte.

For example, to translate from EBCDIC to IRA, create a 256-byte table in storage locations, 1000-10FF hexadecimal.

The table contains the characters of IRA code in sequence of binary representation of EBCDIC code; that is, the IRA code is placed in the table at the relative location equal to the binary value of the EBCDIC code of the same character.

Types of Operations

- ✓ Locations 10F0 through 10F9 contain the values 30 through 39,
- ✓ because F0 is the EBCDIC code for the digit 0,
- ✓ 30 is the IRA code for the digit 0 to digit 9.

Now suppose we have the EBCDIC for the digits 1984 starting at location 2100 and we wish to translate to IRA. Assume the following:

- Locations 2100–2103 contain F1 F9 F8 F4.
- R1 contains 2100.
- R2 contains 1000.

Then, if we execute

TR R1 (4), R2

Types of Operations

Input and Output: There are a variety of approaches taken, including isolated programmed I/O, memory- mapped programmed I/O, DMA, and the use of an I/O processor. Many implementations provide only a few I/O instructions, with the specific actions specified by parameters, codes, or command words.

System control instructions: are those that can be executed only while the processor is in a certain privileged state or is executing a program in a special privileged area of memory. These instructions are reserved for the use of the operating system.A system control instruction may read or alter a control register.

Example is an instruction to read or modify a storage protection key is used in the EAS/390 memory system.

Another example is access to process control blocks in a multiprogramming system.

Types of Operations

Transfer of Control: There are a number of reasons why transfer-of-control operations are required. Among the most important are the following:

1. It is essential to be able to execute each instruction more than once and thousands of times. It may require thousands or millions of instructions to implement an application. If a table or a list of items is to be processed, a program loop is needed. One sequence of instructions is executed repeatedly to process all the data.

2. Virtually all programs involve some decision making. We would like computer to do one thing if one condition holds, and another thing if another condition holds.

For example, a sequence of instructions computes the square root of a number. At the start of the sequence, sign of the number is tested. If the number is negative, the computation is not performed, but an error condition is reported.

3. To compose correctly a large or even medium-size computer program is an exceedingly difficult task. It helps if there are mechanisms for breaking the task up into smaller pieces that can be worked on one at a time.

Types of Operations

Branch Instructions: A branch instruction, also called a jump instruction, has as one of its operands the address of the next instruction to be executed. The instruction is a **conditional branch instruction** .

Branch is made (update program counter to equal address specified in operand) only if a certain condition is met.

Next instruction in sequence is executed (increment program counter as usual). A branch instruction in which the branch is always taken is **an unconditional branch** .

There are two common ways of generating the condition to be tested in a conditional branch instruction.

Most machines provide a 1-bit or multiple- bit condition code that is set as the result of some operations. This code can be thought of as a short user-visible register.

An arithmetic operation (ADD,SUBTRACT etc) could set a 2-bit condition code with one of the following four values: 0, positive, negative, overflow.

Types of Operations

Four different conditional branch instructions:

-BRP X Branch to location X if result is positive.

-BRN X Branch to location X if result is negative.

-BRZ X Branch to location X if result is zero.

-BRO X Branch to location X if overflow occurs.

Figure 12.7 shows a branch can be either forward (higher address) or backward (lower address). Unconditional and a conditional branch can be used to create a repeating loop of instructions. Instructions in locations 202 through 210 will be executed repeatedly until the result of subtracting Y from X is 0.

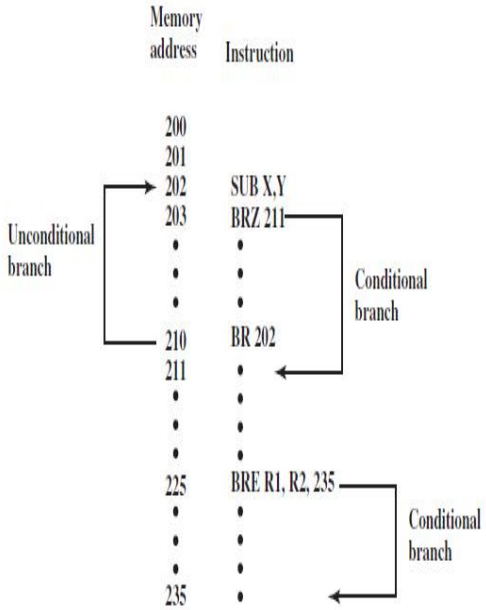


Figure 12.7 Branch Instructions

Types of Operations

SKIP Instructions: implies that one instruction be skipped;implied address equals the address of the next instruction plus one instruction length.

Skip instruction does not require a destination address field, it is free to do other things. A typical example is the increment- and- skip- if- zero (ISZ) instruction. Consider the following program fragment:

```
301
:
309   ISZ   R1
310   BR    301
311
```

Two transfer- of- control instructions are used to implement an iterative loop.

R1 is set with the negative of the number of iterations to be performed. At the end of the loop, R1 is incremented.

If it is not 0, the program branches back to the beginning of the loop.

Otherwise, the branch is skipped, and the program continues with the next instruction after the end of the loop.

Types of Operations

Procedure Call Instructions: A procedure is a self-contained computer program that is incorporated into a larger program. At any point in the program the procedure may be invoked, or called. The processor is instructed to go and execute the entire procedure and then return to the point from which the call took place.

The procedure mechanism involves two basic instructions:

- ✓ a call instruction that branches from the present location to the procedure.
- ✓ a return instruction that returns from the procedure to the place from which it was called.

Both of these are forms of branching instructions.

Types of Operations

Figure 12.8a illustrates the use of procedures to construct a program.

Main program starting at location 4000.

This program includes a call to procedure PROC1, starting at location 4500.

When this call instruction is encountered,processor suspends execution of main program and begins execution of PROC1 by fetching the next instruction from location 4500.

Within PROC1, there are two calls to PROC2 at location 4800.

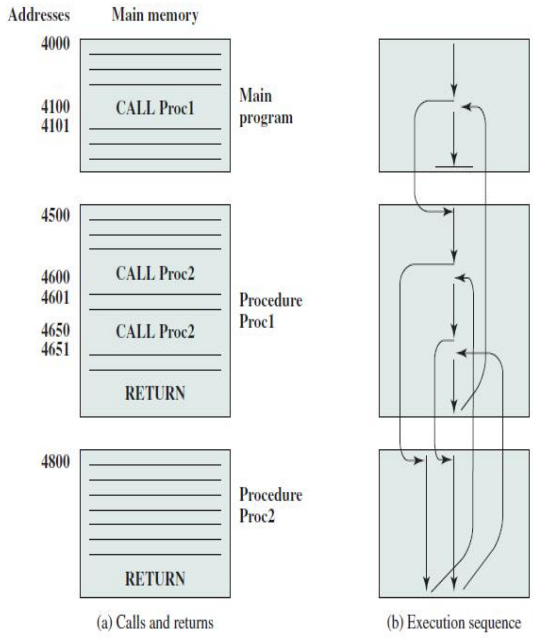


Figure 12.8 Nested Procedures

Types of Operations

In each case, the execution of PROC1 is suspended and PROC2 is executed.

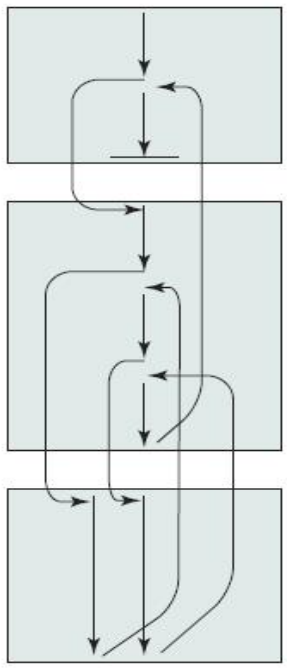
RETURN statement causes the processor to go back to calling program and continue execution at the instruction after the corresponding CALL instruction in Fig 12.8b.

-Procedure can be called from more than one location.
-Procedure call can appear in a procedure. This allows the nesting of procedures to an arbitrary depth.

-Each procedure call is matched by a return in the called program.

Processor must save the return address so that the return can take place appropriately. There are three common places for storing the return address:

- ✓ Register
- ✓ Start of called procedure
- ✓ Top of stack



(b) Execution sequence

Types of Operations

1.Consider a machine-language instruction CALL X, which stands for call procedure at location X. If the register approach is used, CALL X causes the following actions:

$$RN \leftarrow PC + \Delta$$
$$PC \leftarrow X$$

- ✓ RN is a register that is always used for this purpose,
- ✓ PC is the program counter,
- ✓ Δ is the instruction length.

Called procedure can now save the contents of RN to be used for the later return.

2. Second possibility is to store the return address at the start of the procedure. In this case, CALL X causes

$$X \leftarrow PC + \Delta$$
$$PC \leftarrow X + 1$$

This is quite handy. The return address has been stored safely away.

Types of Operations

The only limitation of these approaches is that they complicate the use of reentrant procedures.

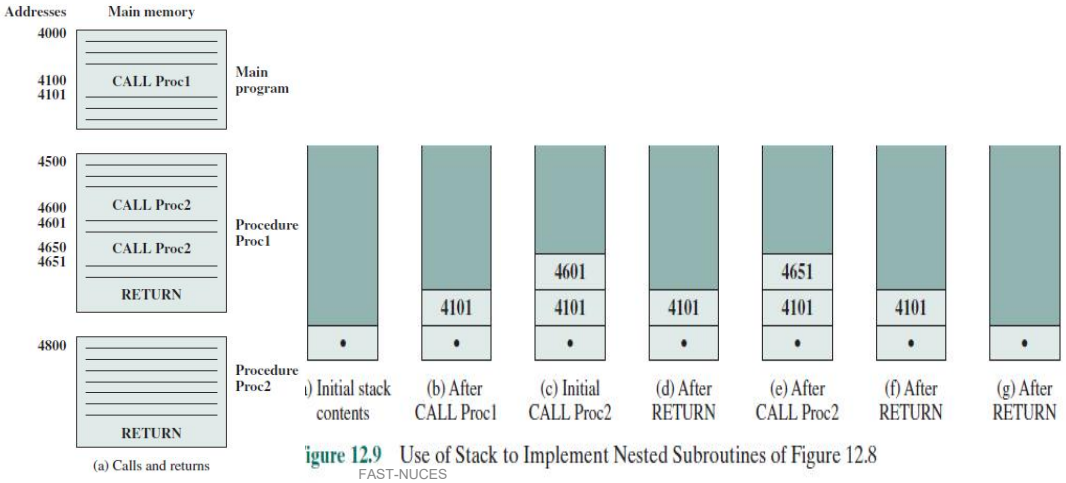
Reentrant procedure is one in which it is possible to have several calls open to it at the same time.

Recursive procedure (one that calls itself) is an example of the use of this feature.

If parameters are passed via registers or memory for a reentrant procedure, some code must be responsible for saving the parameters so that the registers or memory space are available for other procedure calls.

Types of Operations

A more general and powerful approach is to use a stack. When processor executes a call, it places the return address on the stack. When it executes a return, it uses the address on stack. Figure 12.9 illustrates the use of the stack.



igure 12.9 Use of Stack to Implement Nested Subroutines of Figure 12.8

Types of Operations

- ✓ In addition to providing a return address, it is necessary to pass parameters with a procedure call, passed in registers.
- ✓ Another possibility is to store the parameters in memory just after the CALL instruction. Return must be to the location following the parameters.

Again, both of these approaches have drawbacks.

- ✓ If registers are used, called program and the calling program must be written to assure that the registers are used properly.
- ✓ Storing of parameters in memory makes it difficult to exchange a variable number of parameters.

Both approaches prevent the use of reentrant procedures.

Types of Operations

A more flexible approach to parameter passing is the stack. When the processor executes a call, it stacks the return address and parameters to be passed to the called procedure.

The called procedure can access the parameters from stack. Upon return, return parameters can also be placed on the stack.

Stack frame: entire set of parameters, including return address, that is stored for a procedure invocation is referred to as a stack frame.

Types of Operations

Figure 12.10 shows **Procedure P in which the local variables x1 and x2** are declared, and **procedure Q has local variables y1 and y2** . P can call Q.

Return point for each procedure is the first item stored in the corresponding stack frame.

Next is stored a pointer to the beginning of the previous frame . This is needed if the number or length of parameters to be stacked is variable.

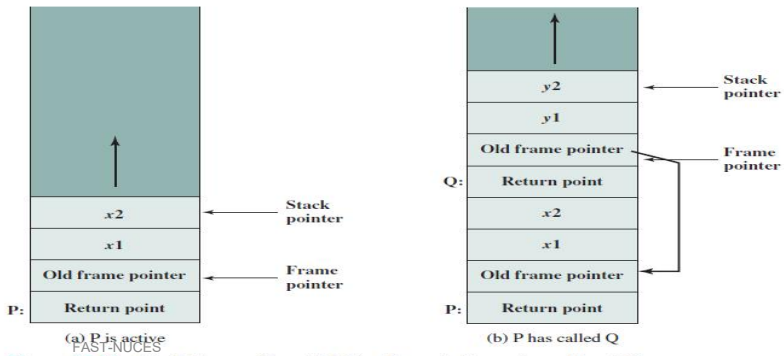


Figure 12.10 Stack Frame Growth Using Sample Procedures P and Q

Intel x86 and ARM Operation Types

Call/return instructions: x86 provides four instructions to support procedure call/return: CALL, ENTER, LEAVE, RETURN. It will be instructive to look at the support provided by these instructions.

Figure 12.10 shows stack frames. When new procedure is called, the following must be performed upon entry to the new procedure:

- ✓ Push the return point on the stack.
- ✓ Push the current frame pointer on the stack.
- ✓ Copy the stack pointer as the new value of the frame pointer.
- ✓ Adjust the stack pointer to allocate a frame.

CALL instruction pushes the current instruction pointer value onto the stack and causes a jump to the entry point of the procedure by placing the address of the entry point in the instruction pointer.