

SUBROUTINES IN COMPUTER ORGANIZATION

In a given program, it is often necessary to perform a particular subtask many times on different data-values. Such a subtask is usually called a subroutine. For example, a subroutine may evaluate the sine function or sort a list of values into increasing or decreasing order.

It is possible to include the block of instructions that constitute a subroutine at every place where it is needed in the program. However, to save space, only one copy of the instructions that constitute the subroutine is placed in the memory, and any program that requires the use of the subroutine simply branches to its starting location. When a program branches to a subroutine we say that it is calling the subroutine. The instruction that performs this branch operation is named a Call instruction.

After a subroutine has been executed, the calling program must resume execution, continuing immediately after the instruction that called the subroutine. The subroutine is said to return to the program that called it by executing a Return instruction.

The way in which a computer makes it possible to call and return from subroutines is referred to as its subroutine linkage method. The simplest subroutine linkage method is to save the return address in a specific location, which may be a register dedicated to this function. Such a register is called the link register. When the subroutine completes its task, the Return instruction returns to the calling program by branching indirectly through the link register.

The Call instruction is just a special branch instruction that performs the following operations

- Store the contents of the PC in the link register
- Branch to the target address specified by the instruction

The Return instruction is a special branch instruction that performs the operation

- Branch to the address contained in the link register .

Fig a illustrates this procedure

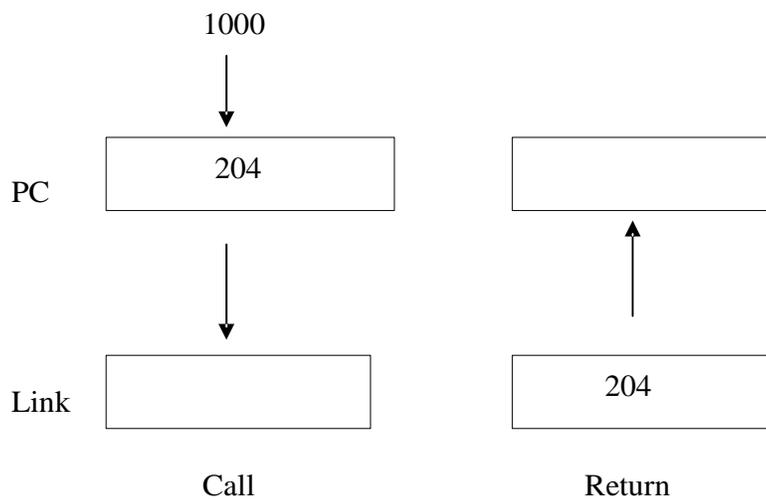
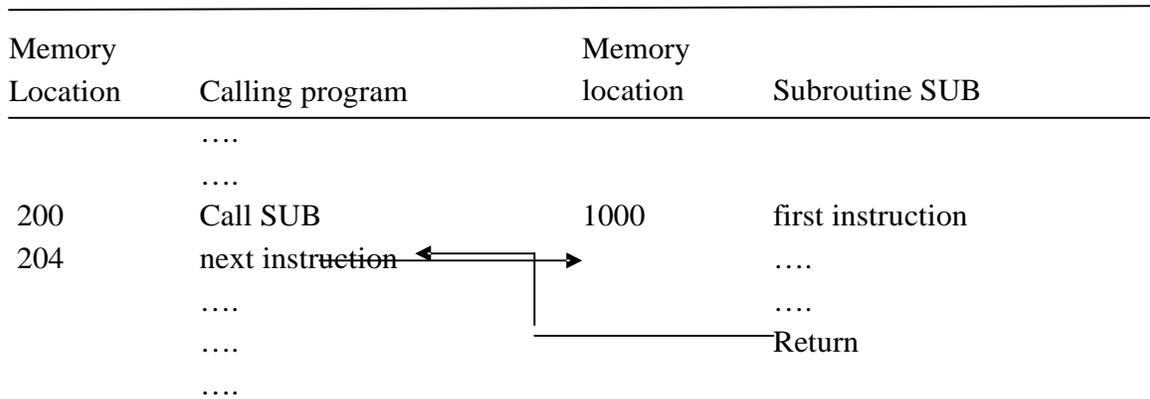


Fig b Subroutine linkage using a link register

SUBROUTINE NESTING AND THE PROCESSOR STACK:-

A common programming practice, called subroutine nesting, is to have one subroutine call another. In this case, the return address of the second call is also stored in the link register, destroying its previous contents. Hence, it is essential to save the contents of the link register in some other location before calling another subroutine. Otherwise, the return address of the first subroutine will be lost.

Subroutine nesting can be carried out to any depth. Eventually, the last subroutine called completes its computations and returns to the subroutine that called it. The return address needed for this first return is the last one generated in the nested call sequence. That is, return addresses are generated and used in a last-in-first-out order. This suggests that the return addresses associated with subroutine calls should be pushed onto a stack. A particular register is designated as the stack pointer, SP, to be used in this operation. The stack pointer points to a stack called the processor stack. The Call

instruction pushes the contents of the PC onto the processor stack and loads the subroutine address into the PC. The Return instruction pops the return address from the processor stack into the PC.

PARAMETER PASSING:-

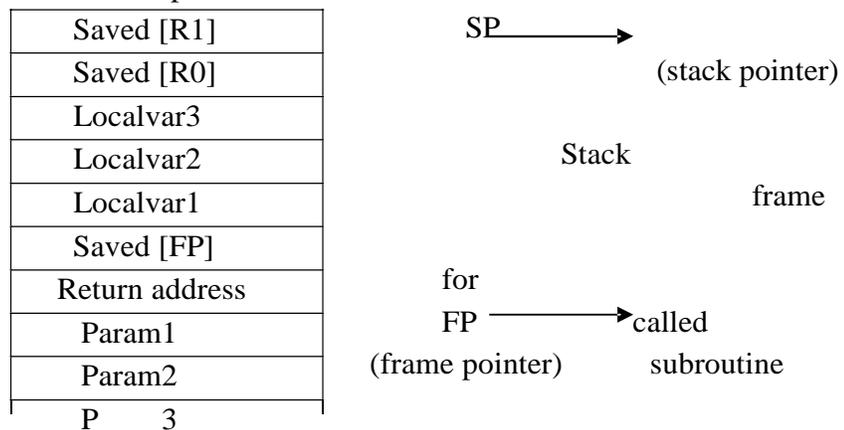
When calling a subroutine, a program must provide to the subroutine the parameters, that is, the operands or their addresses, to be used in the computation. Later, the subroutine returns other parameters, in this case, the results of the computation. This exchange of information between a calling program and a subroutine is referred to as parameter passing. Parameter passing may be accomplished in several ways. The parameters may be placed in registers or in memory locations, where they can be accessed by the subroutine. Alternatively, the parameters may be placed on the processor stack used for saving the return address.

The purpose of the subroutines is to add a list of numbers. Instead of passing the actual list entries, the calling program passes the address of the first number in the list. This technique is called passing by reference. The second parameter is passed by value, that is, the actual number of entries, n, is passed to the subroutine.

THE STACK FRAME:-

Now, observe how space is used in the stack in the example. During execution of the subroutine, six locations at the top of the stack contain entries that are needed by the subroutine. These locations constitute a private workspace for the subroutine, created at the time the subroutine is entered and freed up when the subroutine returns control to the calling program. Such space is called a stack frame.

Fig a A subroutine stack frame example.



Old TOS

fig b shows an example of a commonly used layout for information in a stack frame. In addition to the stack pointer SP, it is useful to have another pointer register, called the frame pointer (FP), for convenient access to the parameters passed to the subroutine and to the local memory variables used by the subroutine. These local variables are only used within the subroutine, so it is appropriate to allocate space for them in the stack frame associated with the subroutine. We assume that four parameters are passed to the subroutine, three local variables are used within the subroutine, and registers R0 and R1 need to be saved because they will also be used within the subroutine.

The pointers SP and FP are manipulated as the stack frame is built, used, and dismantled for a particular of the subroutine. We begin by assuming that SP point to the old top-of-stack (TOS) element in fig b. Before the subroutine is called, the calling program pushes the four parameters onto the stack. The call instruction is then executed, resulting in the return address being pushed onto the stack. Now, SP points to this return address, and the first instruction of the subroutine is about to be executed. This is the point at which the frame pointer FP is set to contain the proper memory address. Since FP is usually a general-purpose register, it may contain information of use to the Calling program. Therefore, its contents are saved by pushing them onto the stack. Since the SP now points to this position, its contents are copied into FP.

Thus, the first two instructions executed in the subroutine are

```
Move  FP, -(SP)
Move  SP, FP
```

After these instructions are executed, both SP and FP point to the saved FP contents.

```
Subtract #12, SP
```

Finally, the contents of processor registers R0 and R1 are saved by pushing them onto the stack. At this point, the stack frame has been set up as shown in the fig.

The subroutine now executes its task. When the task is completed, the subroutine pops the saved values of R1 and R0 back into those registers, removes the local variables from the stack frame by executing the instruction.

```
Add #12, SP
```

And pops the saved old value of FP back into FP. At this point, SP points to the return address, so the Return instruction can be executed, transferring control back to the calling program.