8.6 Subroutines

PTU - May 2

- Whenever we need to use a group of instructions several times throughout a program there are ways we can avoid having to write the group of instructions each time we want to use them.
- One way is to write the group of instructions as a separate subroutine. We can then CALL subroutine whenever we want to execute that group of instructions. For calling the subroutine, return address has to be stored back on to the stack.

The basic requirements which must be satisfied while calling a subroutine are :

(1) A subroutine call must save the address of the next instruction so that the return will be able to branch back to the proper place in the calling program.

(2) The registers used by the subroutine need to be stored before their contents are changed and then restored just before the subroutine is exited.

(3) A subroutine must have a means of communicating or sharing data with the routine that calls it and other subroutines.

The first requirement is met by the CALL and RET branch instructions. The second requirement is met by saving the register and flags by PUSH instruction and retrieving them back by POP instruction. The subroutines are implemented in 8085 using CALL and RET instructions. The CALL instruction is used to call a subroutine. Program control is transferred to subroutine. At the end of subroutine, RET instruction is given. When RET is executed, program control is transferred back to next instruction after CALL in the main program.

Advantages of subroutines

(1) If a part of a program appears repeatedly we may write it once and call it whenever required. This reduces the size of memory required for a program i.e. it saves memory space.

(2) If you are writing a large program then divide it in subsections, prepare and check each subsection separately and at the end combine all the subsections using main program. This makes programming MODULAR. At a time different peoples will work on individual subroutine. This improves efficiency, reduces errors and reduce time required for programming.

8.6.1 Replacing CALL and RET

At this point you may think; **is it possible to have subroutines without CALL and RET instructions?**

The answer is yes, we can have. But then to, switch over to subroutine and to return back from subroutine, we have to use Indirect method. The same is depicted as follows :

(i) The CALL instruction transfers program control to the specified memory location and before transferring the program control, the contents of PC i.e. return address is stored on to stack.

(ii) The RET instruction takes back the contents of PC i.e. return address from stack and program control is transferred back to the CALLing program.

The above two steps can be implemented by using other instruction. Let us first implement the RET instruction. For implementing the RET instruction it is essential to load the program counter with the return address. To load the 16 bit address into the program counter two instructions are possible. PCHL and JMP address. JMP address loads the program counter with fix address and return address is not fix. The return address depends on the location from where the subroutine is called in the main program. Hence, for implementing the RET instruction, the instruction JMP address cannot be used.

The instruction PCHL loads the 16 bit contents of the HL register pair to the program counter. If we can load the return address to the HL register pair, every time subroutine is called then RET instruction can be implemented using PCHL instruction. Program 4 shows how PCHL instruction replaces RET instruction in program 3.

Now, we will implement CALL instruction. It can be implemented using the JMP instruction because in case of CALL instruction we require to load the PC with subroutine address, as return address is already loaded in HL register pair. For this we can use JMP address instruction program 2 shows how JMP address instruction replaces the CALL instruction in program 1.

Program 1 :

C000 LXI SP, FFFF H
–
–
C050 CALL 4000 H
C053 Next instruction

Program 2 :

C000 LXI SP, FFFF H	Initialize SP
–	
–	
C050 PUSH H	Store contents of HL register to use HL for other use.
C051 LXI H, C058 H	Initialize HL with return address.
C054 PUSH H	Store return address on to stack.
C055 JMP 4000 H	Load PC with 4000 H address
C058 POP H	Take back contents of HL from stack.

Program 3 :

4000
–
–
4010 RET

Program 4 :

4000	
4010 POP H	Take back contents of return
	address from stack.
4011 PCHL	Return back to main program.

8.6.2 Parameter Passing Techniques

It is a common requirement of the calling program to pass some data as input to the subroutine. The subroutine in turn may generate some results for the calling program. The passing of data to and fro from a subroutine is called "**parameter passing**". The following techniques are adopted for parameter passing:

(1) Parameter passing through registers.
(2) Parameter passing through reserved memory locations.
(3) Parameter passing using pointer of memory.
(4) Parameter using the stack.

(1) Parameter passing through register

- This technique is used when the number of data bytes to be passed is fewer then the number of internal general purpose registers. In this technique, the main program loads internal registers with appropriate parameters before calling the subroutine.
- The subroutine obtains its parameters from pre-determined registers when called. The results generated by a subroutine are placed in predetermined registers before the return instruction is executed e.g. Multiplication subroutine is designed to multiply contents of register B with the contents of register A.
- This subroutine stores result in H and L registers. The main program loads register A with multiplier value and register B with multiplicand value. It then calls the subroutine.
- The subroutine stores product in HL pair before execution of RET instruction. The main program then gets the product from HL pair. Suppose we want to rotate data by 3 bits to the right. The 8 bit numbers are stored at location C300 H.

Main Program :

Label	Instruction	Comment
	LXI SP, DFFF H	Initialise stack
	LXI H, C300 H	Initialise pointer
	MVI C, 0A H	Initialise counter
LOOP :	MOV A,M	Load data in register A Data (parameter) passed in register A
	CALL ROTATE	Subroutine call
	MOV M, A	Store data, data received in register A only
	INX H	Pointer = Pointer + 1
	DCR C	Counter = Counter – 1
	JNZ LOOP	Continue loop, till C = 0
	HLT	

Subroutine

Input : In register A

Output : In register A

Function : Rotates contents of accumulator right, 3 times.

Label	Instruction	Comment
ROTATE	RRC	Rotate
	RRC	
	RRC	
	RET	Return (Results in accumulator only)

(2) Parameter passing through reserved memory locations

- In this technique, parameters are passed through data memory locations.
- The main program writes parameters (input) into the pre-determined memory locations.
- The subroutine stores output parameters into the determined memory locations.
- Suppose we decide reserved memory location is C400 H.

Main Program :

Label	Instruction	Comment
	LXI SP, DFFF H	Initialise stack pointer
	LXI H, C300 H	Initialise memory pointer
	MVI C, 0A H	Initialise counter
LOOP :	MOV A, M	Load data
	STA C400 H	Pass it to reserved memory location
	CALL ROTATE	Call subroutine
	LDA C400 H	Get result through reserved location
	MOV M, A	Store data
	INX H	Pointer = Pointer + 1
	DCR C	Counter = Counter – 1
	JNZ LOOP	Loop, till C = 0
	HLT	

Subroutine

Input : Data at location C400 H

Output : Result at location C400 H

Function : Rotate contents of memory location C400 H.

Label	Instruction	Comment
ROTATE :	LDA C400 H	Load data
	RRC	Rotate
	RRC	
	RRC	
	STA C400 H	Store data at location, pointed by HL
	RET	Return

(3) Parameter passing using memory pointer

When the subroutine requires a large number of parameters they can be placed in RAM and pointer to the data can be provided in internal register or in reserved memory locations. After writing all parameters into memory locations, the main program loads internal registers or reserved memory location with the starting address of parameter list. The subroutine reads parameters from the list with the help of parameter pointer.

We will use HL itself as a pointer

LXI SP, DFFF H	Initialise stack pointer
LXI H, C300 H	Initialise stack pointer
MVI C, 0A H	Initialise counter
LOOP : CALL ROTATE	Call subroutine
INX H	Pointer = Pointer + 1
DCR C	Counter = Counter – 1
JNZ LOOP	Loop
HLT	

Subroutine

Input : Data pointed by pointer HL Output

Output : Store at location pointed by HL

Function : Rotate data right three times

Label	Instruction	Comment
ROTATE :	MOV A, M	Load data pointed by HL
	RRC	Rotate
	RRC	
	RRC	
	MOV M, A	Store data at location, pointed by HL
	RET	Return

(4) Parameter passing using stack

- The stack can be used to pass parameters.
- The main program places parameters on the stack by using PUSH instructions. These parameters together with return addresses are stored on the stack, which is called stack frame. The stack frame is shown in Fig. 8.6.1.
- The subroutine pops the return address from the stack and saves it in internal register or reserved memory location.
- The subroutine pops the parameters from the stack as needed. When all the parameters are removed and processed, the subroutine pushes the return address back onto the stack and executes a return instruction.

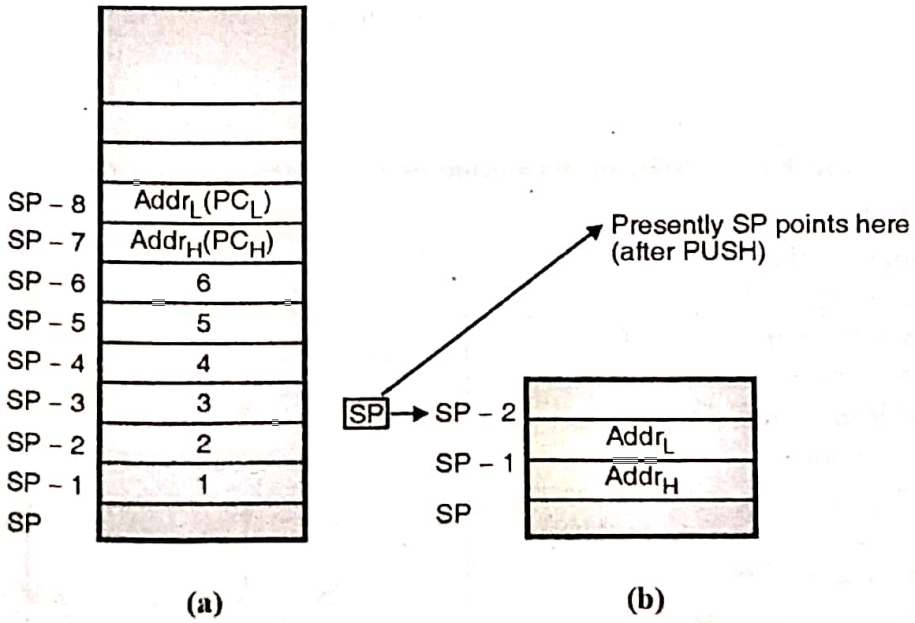
Main Program :

```
PUSH B
PUSH D
 :
PUSH H
CALL subroutine
ADD B
 :
 :
```

Subroutine

```
POP H      ;    Save return address (PC)
SHLD address
POP H
POP D
POP B
 :
 :
LHLD address    ;    Recall return address
PUSH H          ;    Push onto stack
RET
```

Main Subroutine



(a) (b)

Fig 8.6.1

Precaution : If all parameters pushed onto the stack are popped off, before executing the RET instruction the stack looks as shown in Fig. 8.6.1(b).