example

Program Control Instructions.

Name	mnemonics
Branch	BR
Jump	JMP
Skip	SKP
Call	CALL
Return	RET
Compare (Subtract)	COMPARE
Test	Testbit

Call: It is used to call a subroutine or high level called Function, or assembly language called subroutine.

1000 : Call P1
1004 : Next Instruction.

1. stack(top) ← [PC] // Return address i.e 1004 is stored into the stack and then
2. PC ← Address of the subroutine // here it is represented by P1

Return : It is used to return from a subroutine.

PC ← stack[top] // Return address i.e 1004 is restored from the stack.

Compare : It is used to compare two numbers

Compare Src, dst

↗ It is non destructive compare sub-operation.

performs the operation : [dst] - [src]

- sets the condition code flags based on the result obtained
- Neither of the operands is changed.

Example
```
CMP # 10, R1
BP ≠ 0 L1
```

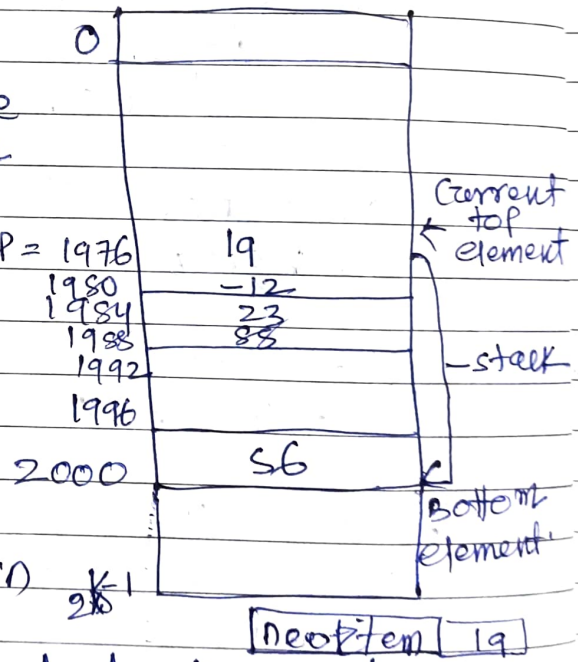
Test : Test instruction is used to check a particular bit Position value of the operand.

Test # bit position, operand.
(performs non-destructive AND operation. (Flag, register affected)
Test #2, R1
• Neither of the operands in changed.

(STACK)

Assumption: The machine is byte addresable and each element in the stack occupies 4 Bytes]

Push operation:
• SP always points to the top element, so to push a newitem, first it has to be updated to the location, where a new element can be pushed.



• As stack grows downward in the memory, so first SP is decremented, then the NEWITEM is pushed.

Note: This rule is applicable for pushing any(all) element into the stack.

Subtract #4, SP
MOVE NEWITEM, -(SP)

Pop operation:

- SP always points to the top element, so to pop an element, first, it has to be popped off, then the SP need to be updated to the next top element.
- As stack grows downward in the memory, so SP is incremented, after the top item is popped off.

```
MOV (SP), ITEM
ADD #, SP.
```
→ Move (SP)+, ITEM

Stack [Assumption: the stack is from 2000 to 1500]

Safe Push operation:

```
SAFEPUSH  Compare #1500, SP
          Branch <=0 FULLERROR
          MOVE NEWITEM, (-SP)
```

on a full stack, Push operation should not be done. so if the value of SP is 1500 or less than 1500, that means already the stack is full. Hence compare operation gives either 0 or a negative value after the operation. If SP is 1500, then next Push operation will be

done at 1496, which is not the part of the stack.

Safepop operation:

SAFEPOP compare #2000, SP
Branch >0 Emptyerror

From a empty stack, no element should be popped out. So, if the value of SP is greater than 2000, that mean already the stack is empty Hence Compared operation gives a +ve value after the operation.

(Subroutines.)

The way in which a computer makes it possible to call and return from subroutines is referred to as subroutine linkage method.

Linkage using Link Register —

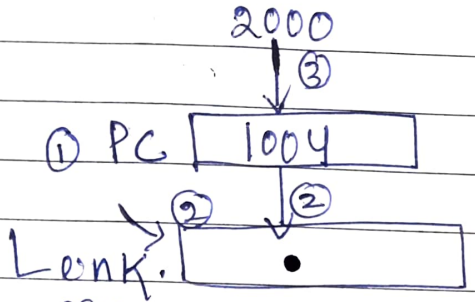
On call

1. Store the contents of the PC in the link register.
2. Branch to the target address specified by the instruction.

On returning from a subroutine

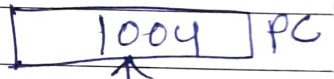
- Branch to the address contained in the link register.

1000 CALL sum.
1004 Next Instruction.



(when CALL instruction executed).

2000 First Instruction
:
:
Return.



Return execution.

Limitations ??

No support for Nesting of functions.

Limitations of Link Register

1020 call P
1024 Next Inst.

P
2000 Inst.
2040 call q
2044 Next Inst?
2070 Return.

q
3000 1st Instr
3050 Return

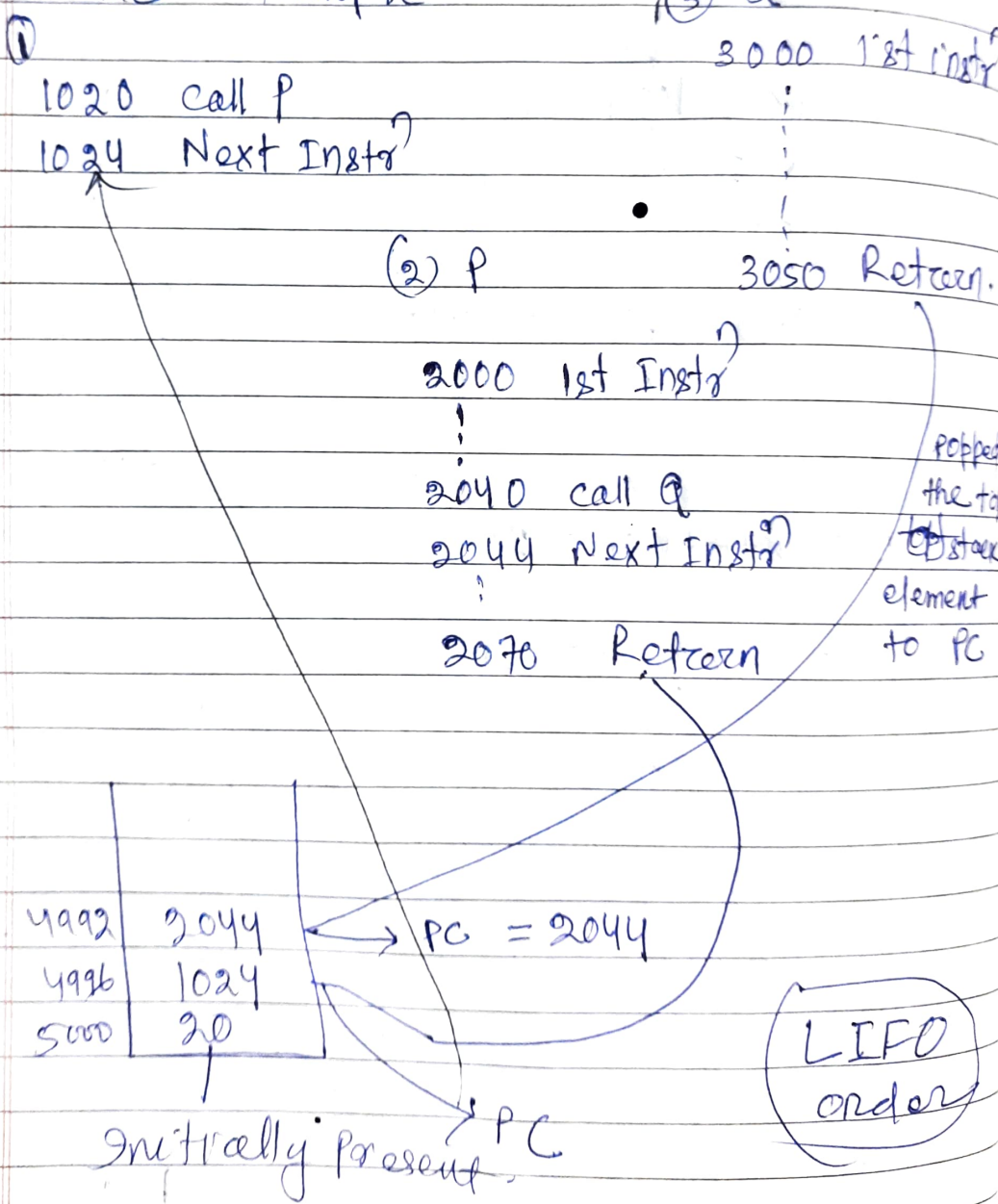
Note
There is a single Link register.

2044 over 1024 - last Link register

As a summary we can conclude that using link registers it is not possible to support subroutine nesting.
But calling a single subroutine link register is OK.

Solution: Using stack:

Same Example.



Stack as Subroutine Linkage Method.

Using stack: calling a function and returning from it.

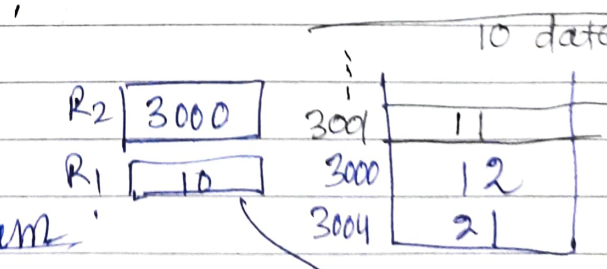
Assumptions.

- Parameters are passed through general purpose registers.
- Returning values through general purpose registers.

Example:

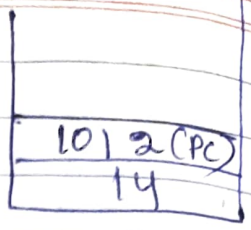
We are going to add N numbers stored in consecutive memory locations starting from the symbolic address NUM1 using a function. The function is going to return the summation result to the caller.

Solution



Calling Program:

```
1000    MOVE  N, R1      ; N = counter
1004    MOVE  #NUM1, R2.
1008    Call  LISTADD
1012    MOVE  R0, SUM
```




```
Subroutine:
LISTADD  Clear R0
LOOP     ADD (R2)+, R0    ; [R0] = 0+12
                          =12
         Decrement R1
         Branch >0 LOOP
         Return           ; PC ← 1012
```

The result is at R0.
We are moving here to location [Sum].

Numerical

(1) The content of the top memory stack is 2452. The content of SP is 1258. A two byte call subroutine instruction is located in memory address & 1456 followed by address field of 5490 at location 1457. What are the content of PC, SP and top of stack
(1) Before call instruction execution.
(2) After call instruction execution.
(3) After return from subroutine.

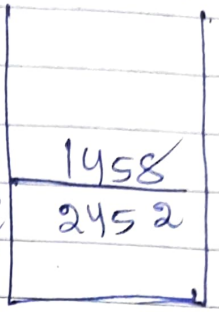
Ans

Given [1456] CALL
[1457] Addr = 5490

[1458] Next instr

Before call instruction execution.

(i) [PC] = 1456
[SP] = 1258
Top of stack (STACK[SP] = 2452.



(ii) After Call instruction execution.

Return address will be pushed onto the stack and PC will be loaded with the value of the address field of the call instr.

[PC] = 5490
[SP] = 1256
STACK [SP] = ~~2452~~ 1458

(iii) After Return from Subroutine.

Top stack content pointed by SP will be popped into PC to return back to the caller

[PC] = 1458
[SP] = 1258
STACK [SP] = 2452

2. How many times a subroutine should be called so that the stack becomes full. Assume that the stack address space ranges from 2000 to 1600 and each stack word consumes 4 bytes and machine is byte addressable. [Note No Parameters, return value, registers, local variables are stored in the stack free to subroutine call.]

Solution:

2000 to 1600: no of memory locations = 400

- Each word on the stack occupies 4 Bytes, and the machine is byte addressable, meaning is that each byte will occupy one location.
- Each word in the stack occupies 4 locations.
- So in 400 locations, 400/4 = 100 words can be stored.
- So if we call a subroutine 100 number of times, the stack will become full.

3. Given the following Program fragment.

Main Program:	SUB1	SUB2.
2000 ADD R_1, R_2	3000 MOV R_1, R_2	4000 SUB R_6, R_1
2004 XOR R_3, R_4	3004 ADD R_5, R_1	4008 XOR R_1, R_5
2008 CALL SUB1	3008 CALL SUB2	4012 RETURN.
2012 SUB R_4, R_5	3012 Return.	

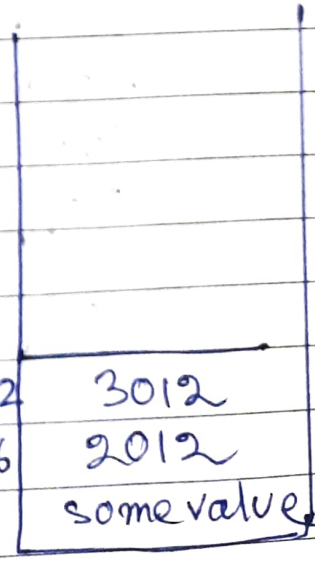
Initially the stack Pointer Ps contains 5000 what are the content of PC, SP, and the top of the stack?

(i) After the subroutine call instruction is executed in the main Program?
(ii) After the subroutine call instruction is executed in the subroutine SUB1?
(iii) After the return from SUB2 subroutine?

Solution.

(i) After the Subroutine call instr? is executed-

- Given, [2008] CALL SUB1
  Given [2012] Next instr?.
- After call instruction will be pushed into the stack and PC will be loaded with the value of the address field of the call instr?.
  [PC] = 3000
  [SP] = 4996



STACK[SP] = 2012.

4.

(ii) After the subroutine call instruction is executed in the subroutine SUB1?

Given [3008] CALL SUB2
[3012] Next Instr
After call instruction execution in SUB1 Return address will be pushed onto the stack and PC will be loaded with the value of the address field of the call instr

[PC] = 4000
[SP] = 4992
STACK[SP] = 3012.

(iii) After the return from SUB2 subroutine?

Given, [4012] RETURN

[PC] - 3012
After the return from SUB2 subroutine Top stack content pointed by SP will be popped into PC to return back to the Caller.
[SP] = 4996
STACK[SP] = 2012

The content of the top of the memory stack is 5000. The content of the stack pointer SP is 3000. Assume you want to organize a nested subroutine calls on a computer as follows:
The routine main calls a subroutine SUB1 by executing a two-word call subroutine instruction located in memory at address 1000 followed by the address of 6000 at location 1001. Again subroutine SUB1 calls another subroutine SUB2 by executing a two-word call subroutine instruction located in memory at address 6050 followed by the address field of 8000 at location 6051. What are the content of PC, SP and the top of the stack?

Solution

(i) After the subroutine call instruction is executed in the main routine?

Given the following Program fragment.

Main Program	SUB1 (6000)	SUB2
1000 call SUB1 (6000)	6050 CALL SUB2(8000)	8000 1st instr?
1002 Next instr?	6052 Next instr?	⋮
	6080 Return	8080 Return

Initially the stack Pointer containg 3000

Given [1000] CALL SUB1
[1002] Next instr?

(i) [PC] = 6000
[SP] = 2998
STACK [SP] = 1002.

SP=2996	6052
SP=2998	1002
SP=3000	5000

(ii) After the subroutine call instr? is executed in the subroutine SUB1
Given [6050] call SUB2
[6052] Next Instr?

[PC] = 8000
[SP] = 2996
STACK [SP] = 6052.

(iii) After the return from SUB2 Subroutine.?

Given, [8080] RETURN

[PC] = 6052
[SP] = 2998
STACK [SP] = 1002.

Stack as Subroutine Linkage Method [Parameter Passing and returning Values using stack]

When a large number of parameters are required to pass to a function, we may not have that many general purpose registers.

Solution —

- Parameter are passed onto the task before calling the function.
- Returning values through the stack.

Calling Program

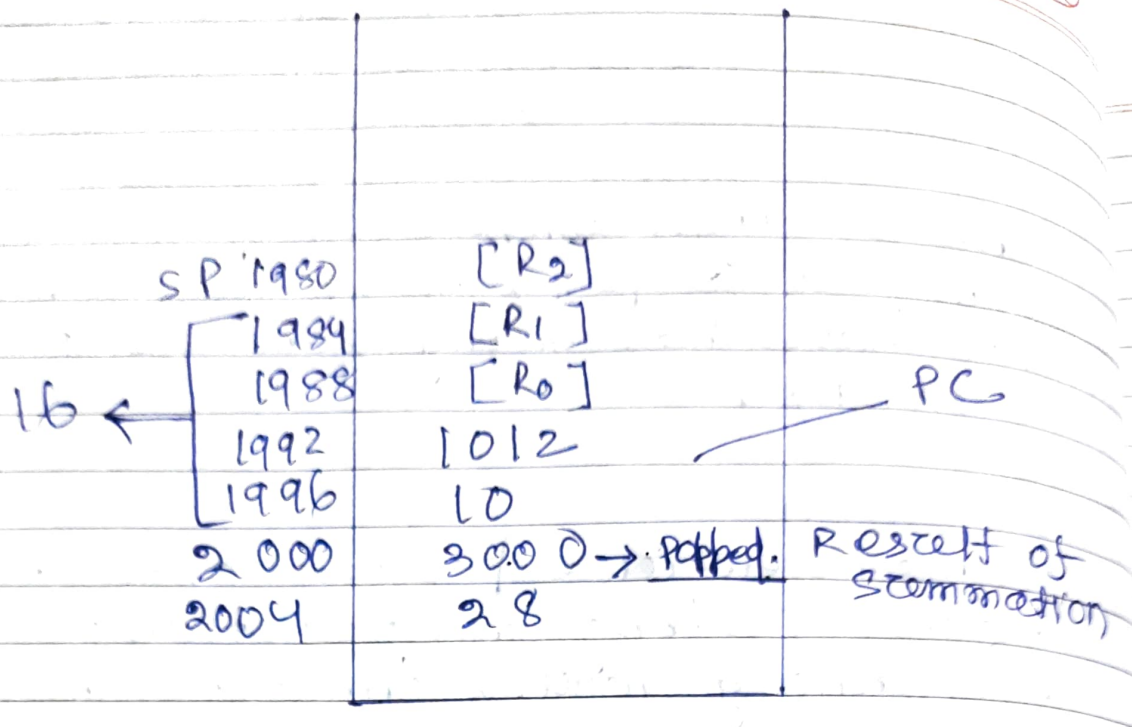
```
1000  MOVE #NUM1, (-SP)
1004  MOVE N, -(SP)
1008  Call Listadd
1012  MOVE 4(SP), SUM
2016  ADD #8, SP
```

Name of the function.

1992	1012 (Ret. addr)
1996	10 counter (N)
2000	3000 (Base addr) (NUM1)
2004	28

Subroutine:

```
Listadd   MOVE Multiple R0-R2, -(SP)
          MOVE  16(SP), R1    ← R1 = 10
          MOVE  20(SP), R2    ← R2 = 3000
          CLEAR R0
Loop      ADD   (R2)+, R0
          Decrement R1
          Branch >0 Loop
          MOVE  R0, 20(SP)
```



MoveMultiple (SP)+, R0-R2
RETURN.

Example-2

When a large numbers of Parameters are required to pass to a function, we may not have that many general purpose registers.

Register Ri is used in a program to point to the top of a stack. Assume that each stack word consumes 4 bytes and machine is byte addressable. Write a sequence of instructions using the Index, Autoincrement and Autodecrement addressing modes to perform

each of the following tasks:

(a) Pop the top two items off the stack, add them, and the push the result into the task.

(b) copy the Fourth item from the top into the register R_2.

(c) Remove the top five items from the stack.

Solution:-

a) Move $(R_1)+, R_5$
ADD $(R_1)+, R_5$
Move $R_5, -(R_1)$

b) Move $16(R_1), R_2$

c) ADD #20, R_1