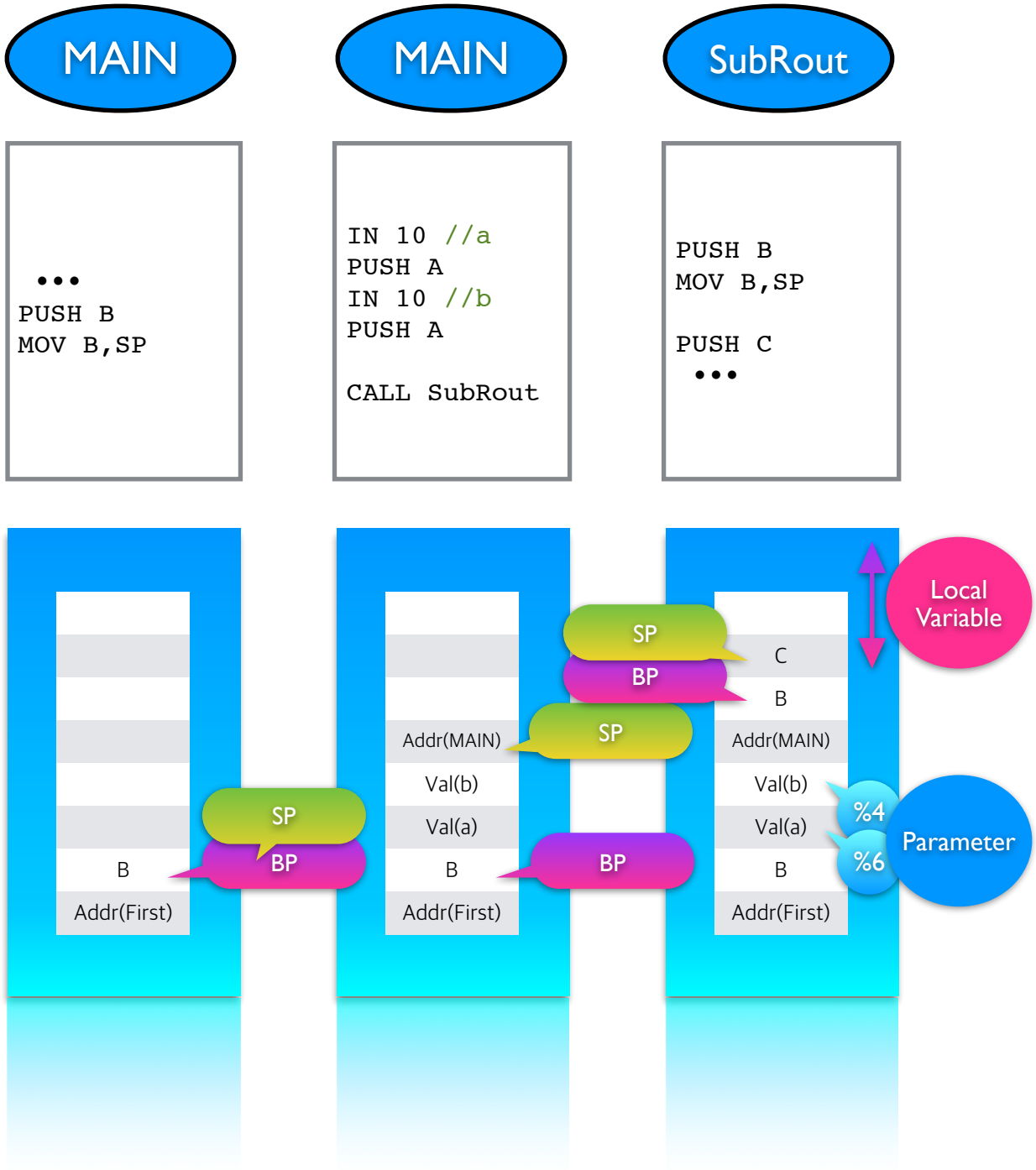


- **파라미터 전달과 반환값 전달, 그리고 지역 변수 배치에 대한 개념을 설명하시오.**

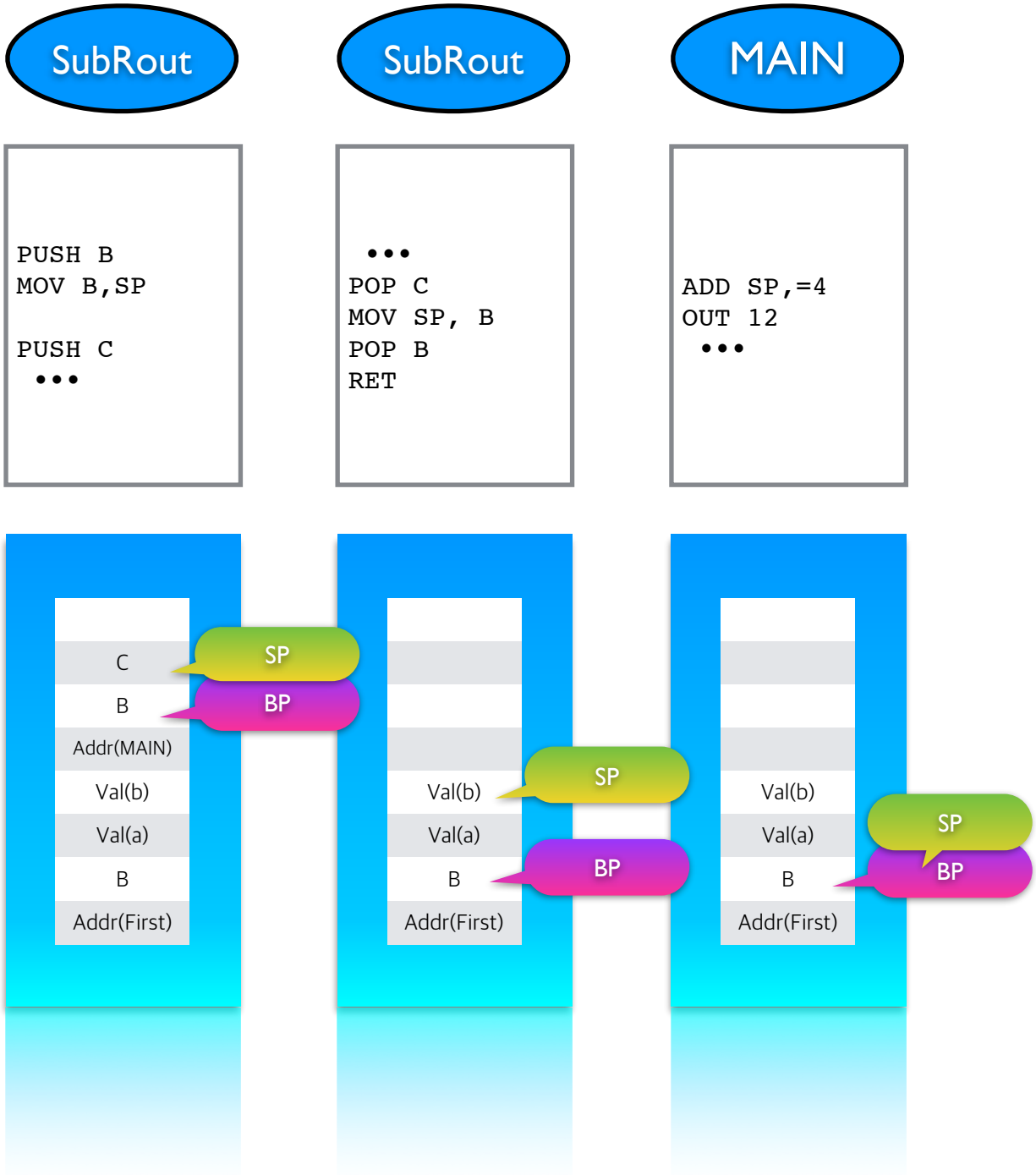
왼편의 그림을 참고하여, 우선 B레지스터의 값을 SP 레지스터가 가리키는 곳으로 둡니다. 그 후에 **서브루틴으로 넘겨줄 값 들을 POP** 하여 스택에 넣어 주며, 그 후 서브루틴을 CALL 합니다.

따라서 왼 쪽 그림에서 보이듯, **B레지스터와 메인으로의 복귀주소 사이의 값**들은 메인에서 **파라미터**로 넘겨준 변수라고 볼 수 있습니다.

이제 이 값을 서브루틴 내에서 사용하기 위해, 우선 현재 B레지스터값 POP 하여 스택에 저장해 두고, 그 값이 저장된 스택 위치를 다시 B레지스터에 넣어줍니다. 이로써 **Base Addressing Mode(%)**를 통해 파라미터 값을 사용할 수 있게 됩니다.

서브루틴 내에서의 코드가 시작 되면서, 그 안에서 새로운 함수나 변수가 사용 되어야 할 경우가 있습니다. 여기서 새로운 변수들을 POP 하게 되면, 이는 **서브루틴 내의 지역변수(Local Variable)**가 됩니다.

지역변수는 **B레지스터보다 위(낮은 주소)**에 쌓이게 되며, 서브루틴 내에서 사용하고 지우거나, 혹은 또 다른 서브루틴을 호출할 때 Parameter 로 넘겨줄 수 있습니다.



서브루틴 내에서의 코드를 진행하기 전, 레지스터의 사용을 자유롭게 하기 위해 사용하지 않을 레지스터 값을 POP 하여 스택에 저장해 둘 수 있습니다. 이 값은 따로 사용하지 않고 두었다가 다시 상위 루틴으로 돌아갈 때 POP 하여 복귀시켜주면 됩니다.

진행이 끝나고 이제 서브루틴이 리턴값을 다시 메인으로 넘겨주어야 합니다. **간단한 계산에서는 A레지스터에 저장**해 놓고 메인에서 출력해 주어도 되지만, 서브루틴 내에서 넘겨받은 Parameter 들에 대해 어떤 변동을 준 후, 메인함수로 돌아와서 그 값을 참조해야 할 경우가 있습니다. 이 Parameter 값들은 상위루트로 복귀하면서 따로 POP 해 주지 않았기에, **Base Addressing Mode(%)를 통해 넘겨받은 각 변수에 대해 접근** 할 수 있습니다.

이로써 서브루틴을 여러번 호출하거나, 혹은 재귀를 할 때에도 복귀할 주소와 변수가 스택에 따로 저장되기에 정보가 사라지거나 실수로 변경되는 것을 막을 수 있습니다.

```
Execute% Assemble 4:cassette/prog_1_sumn.lmc
                          1         ORG     0
                          2
                          3 //#begin
                          3 //% FIRST MOV  SP #STK_BTM
4355 0000 0000   0000:0000   3 FIRST   LD  SP #0
                          3 //#end
                          4 //#begin
                          4 //% MOV B =0
4310 0053 0000   0000:0003   4         LD  B =0
                          4 //#end
6700 0010 0000   0000:0006   5         CALL    MAIN
0700             0000:0009   6         COB
                          7
9881             0000:0010   8 MAIN    PUSH    B
                 0000:0011   9 INVAL_N RESDBOX 1
0510             0000:0013  10         IN      10
                          11 //#begin
                          11 //% MOV INVAL_N A
5300 0011 0000   0000:0014  11         ST  A INVAL_N
                          11 //#end
                 0000:0017  12 TOTAL   RESDBOX 1
9880             0000:0019  13         PUSH    A
6700 0035 0000   0000:0020  14         CALL    SUM_N
                          15 //#begin
                          15 //% MOV  TOTAL %-2
9880             0000:0023  15         PUSH A
4205 9998        0000:0024  15         LD  A %-2
5300 0017 0000   0000:0026  15         ST  A TOTAL
9890             0000:0029  15         POP A
                          15 //#end
                          16 //#begin
                          16 //% MOV  A,TOTAL
4300 0017 0000   0000:0030  16         LD  A TOTAL
                          16 //#end
0612             0000:0033  17         OUT     12
9999             0000:0034  18         RET
                          19
9881             0000:0035  20 SUM_N   PUSH    B
                          21 //#begin
                          21 //% MOV B SP
4115             0000:0036  21         LD  B SP
                          21 //#end
9882             0000:0037  22         PUSH    C
                          23 //#begin
                          23 //% MOV C %4
4225 0004        0000:0038  23         LD  C %4
                          23 //#end
```

- 다음 프로그램에 대한 LMC 소스코드를 작성하시오. (주석과 함께)

```
main() {
    n = data_in();
    total = sum(n);
    print(total);
 }
```

```
5990 0047 0000   0000:0040  24 LOOP    JLOOP   ENDLOOP
1102             0000:0043  25         ADD     A C
5900 0040 0000   0000:0044  26         JUMP    LOOP
                          27 //#begin
                          27 //% ENDLOOP MOV %4 A
5205 0004        0000:0047  27 ENDLOOP ST  A %4
                          27 //#end
9892             0000:0049  28         POP     C
                          29 //#begin
                          29 //% MOV SP B
4151             0000:0050  29         LD  SP B
                          29 //#end
9891             0000:0051  30         POP     B
9999             0000:0052  31         RET
                          32
                          33
                          34
0000 0000        0000:0053  35 $             =0
                          35 END
```

#	Label	mnemonic	Comment
1	**FIRST**	START 0	
2		MOV SP, #STK_BTM	SP.Reg, B.Reg 를 스택바닥 위치로 초기화 후 메인함수 실행
3		MOV B,=0	
4		CALL MAIN	
5		COB	
6			
7	**MAIN**	PUSH B	기존의 B.Reg 값 스택에 저장
8	**INVAL_N**	RESDBOX 1	빈칸하나를 두어 여기에 input n 을 저장
9		IN 10	키보드 입력
10		MOV INVAL_N,A	n값 저장
11	**TOTAL**	RESDBOX 1	빈칸하나를 두어 서브루틴 호출 후 결과값을 여기에 저장
12		PUSH INVAL_N	파라미터를 스택에 저장
13		CALL SUM_N	서브루틴(SUM 함수) 호출
14		MOV TOTAL,%-2	스택에 저장된 서브루틴 결과값을 TOTAL에 저장
15		MOV A, TOTAL	TOTAL값을 출력 후 FIRST로 복귀.
16		OUT 12	
17		RET	
18			
19	**SUM_N**	PUSH B	기존의 B.Reg 값 스택에 저장
20		MOV B SP	B.Reg = SP.Reg
21		PUSH C	기존의 C.Reg 값 스택에 저장
22		MOV C,%4	파라미터로 받은 n을 C.Reg 에 저장
23	**LOOP**	JLOOP ENDLOOP	n만큼 루프를 실행한 후 ENDLOOP로 점프
24		SUM A,C	A에 n-i(i = 루프를 돈 횟수)가 계속 더해진다.
25		JUMP LOOP	Loop 로 점프
26	**ENDLOOP**	MOV %4, A	더한 결과값을 파라미터값에 넣어줌
27		POP C	C.Reg를 서브루틴 전의 값으로 복귀
28		MOV SP, B	SP.Reg = B.Reg
29		POP B	B.Reg를 서브루틴 전의 값으로 복귀
30		RET	MAIN으로 복귀.