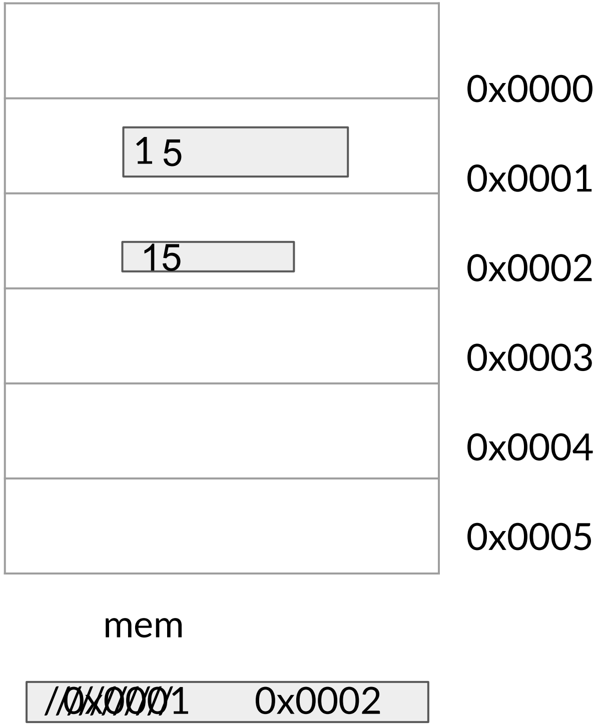
جامعة عين شمس
AIN SHAMS UNIVERSITY

Section 5

Sheet 4

pointer

```
Void main ()
{
   Char x=5;
   Char * pr;
   ptr= &x;
   *ptr=15;
   ptr++;
   *ptr=15;

}
```



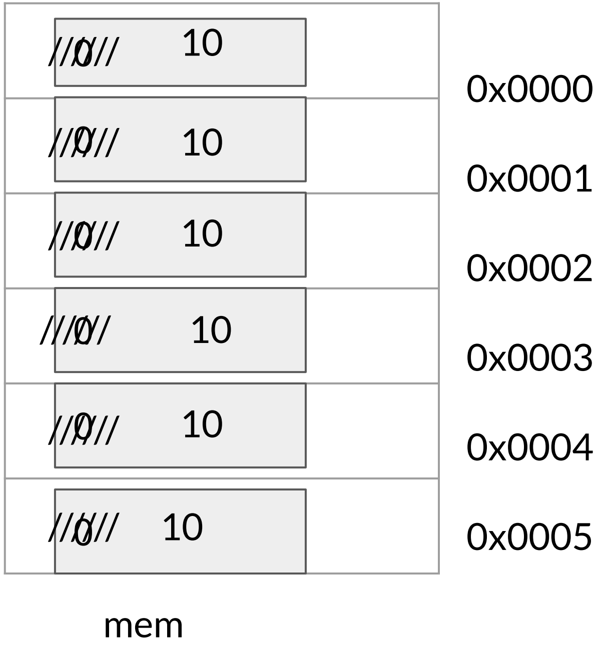
mem	
	0x0000
1 5	0x0001
15	0x0002
	0x0003
	0x0004
	0x0005

mem

~~0x0001~~ 0x0002

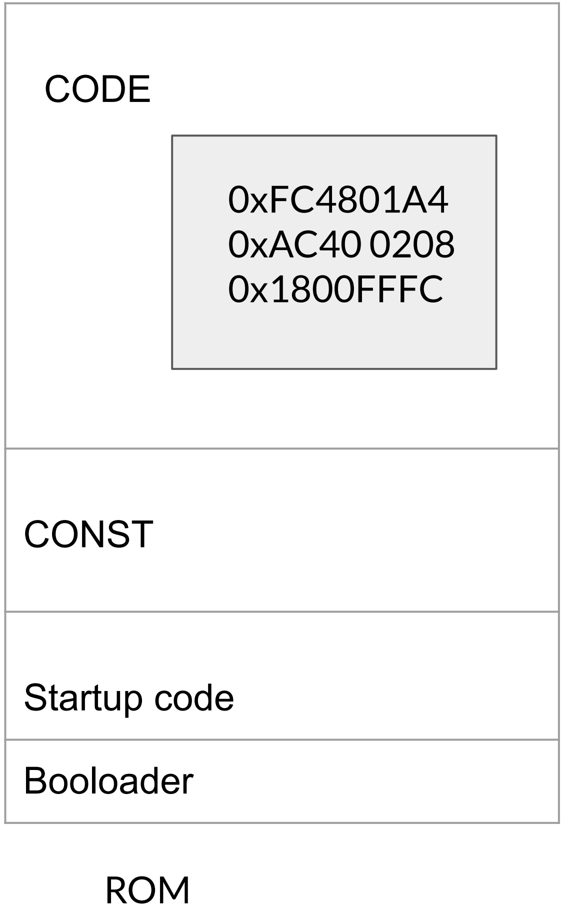
pointer

```
Void main ()
{

   Char arr[6]={0,0,0,0,0,0};

   Char *ptr=arr;

   for(i=0;i<6;i++)
     *(ptr+i)=10;


}
```

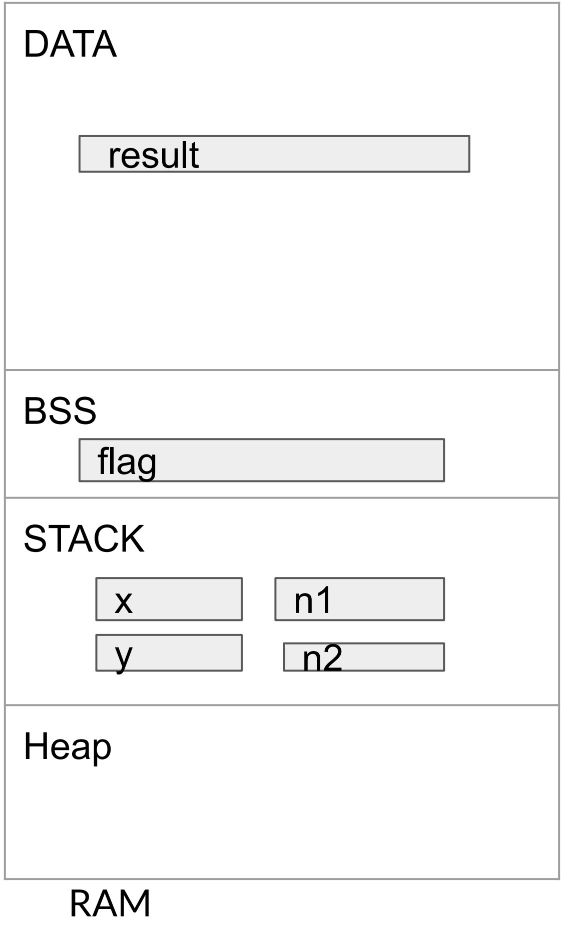


0x000 0x0002 and so on 0x0001

stack



```
int result =1;
char flag;
int add(int x,inty)
{
   result=x+y;
   return result;
}
int sub(int n1,n2)
{
 if(n1>n2){flag=1;}
   result=n1-n2;
   return result;
}

Void main()
 result=sub(5,4);
 if(flag==1)
{
 printf(result);
}
```

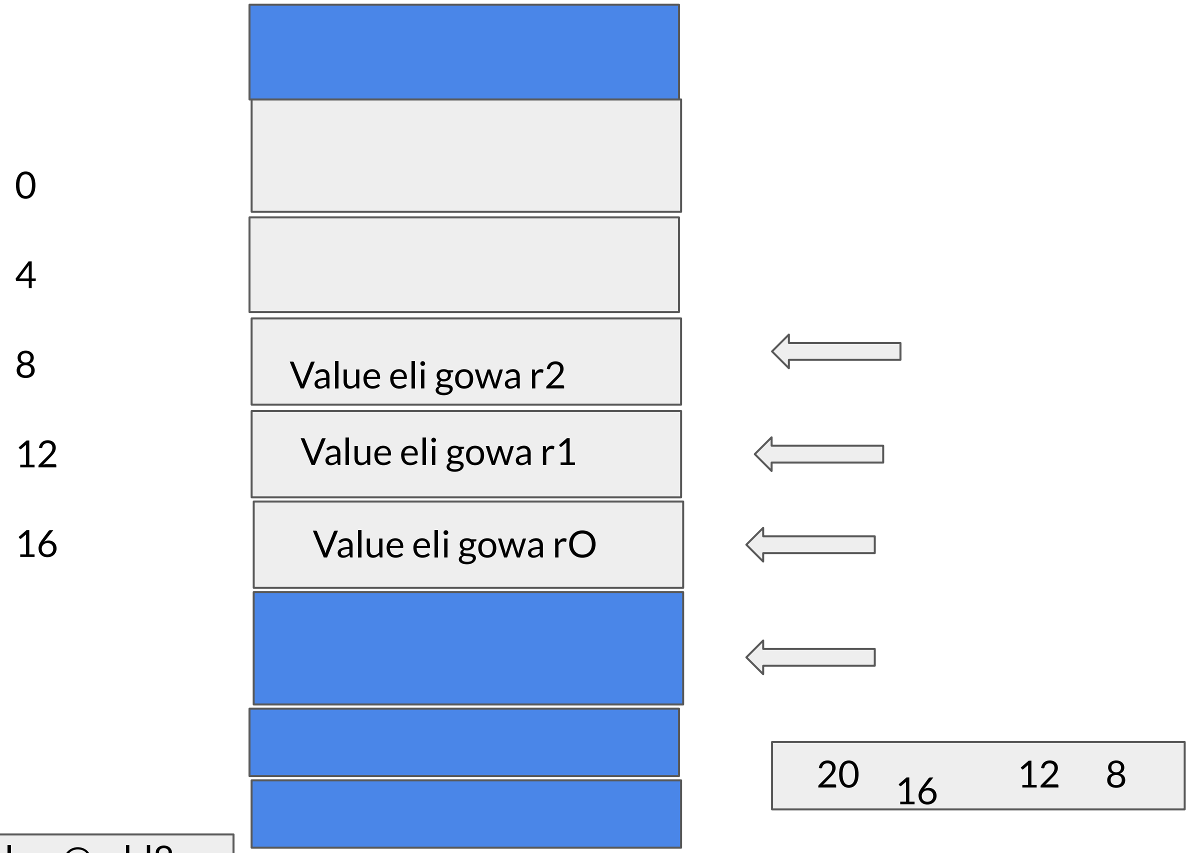


stack

PUSH {r0}

PUSH {r1}

PUSH {r2}



0

4

8

12

16

Value eli gowa r2

Value eli gowa r1

Value eli gowa rO

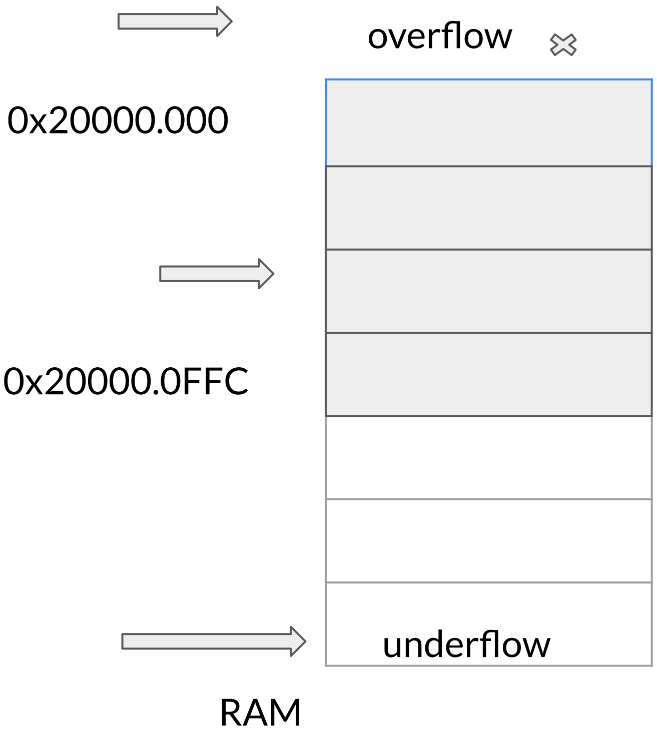
20 16 12 8

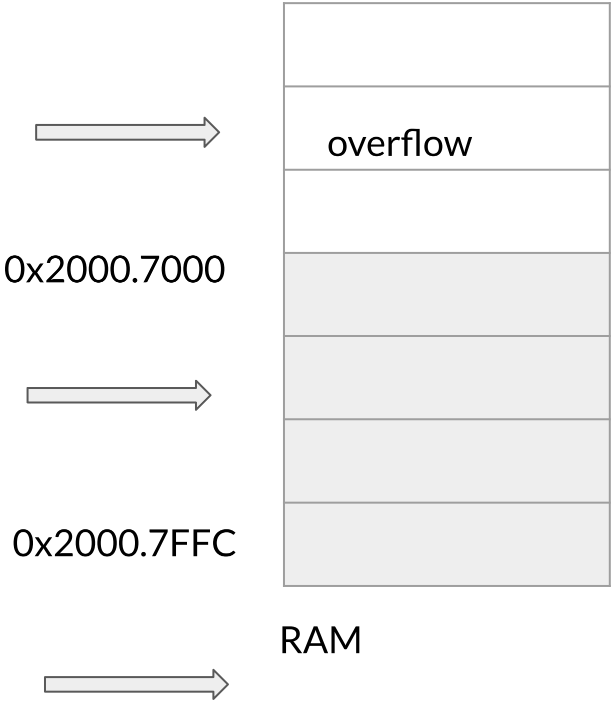
POP {r2}

POP{r1}

POP {r0}

Value @add16	Value @add12	Value @add8
r0	r1	r2


overflow
0x20000.000
0x20000.0FFC
underflow
RAM


overflow
0x2000.7000
0x2000.7FFC
RAM

stack

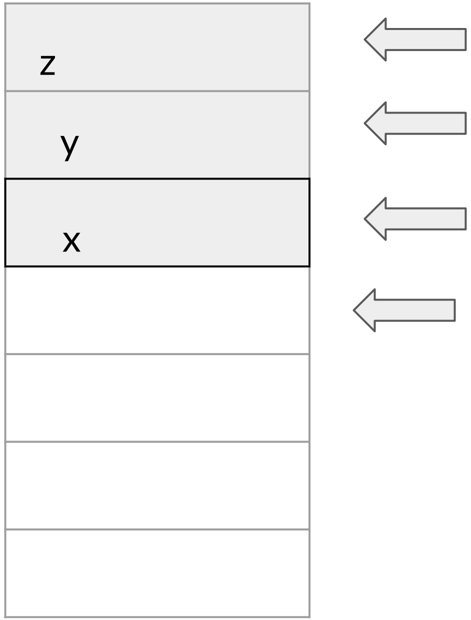
```
void main
{
  r4=x,r5=y,6=z
  f1()
  r4?

}
```

```
f1()
{

      Push {r6,r5,r4}


    Move r4 #100
    add r5 r5 r6
    Pop{r6}
    Pop{r5}
    pop{4}
}
```

R4 ~~x~~ 100

R5 ~~y~~ 10

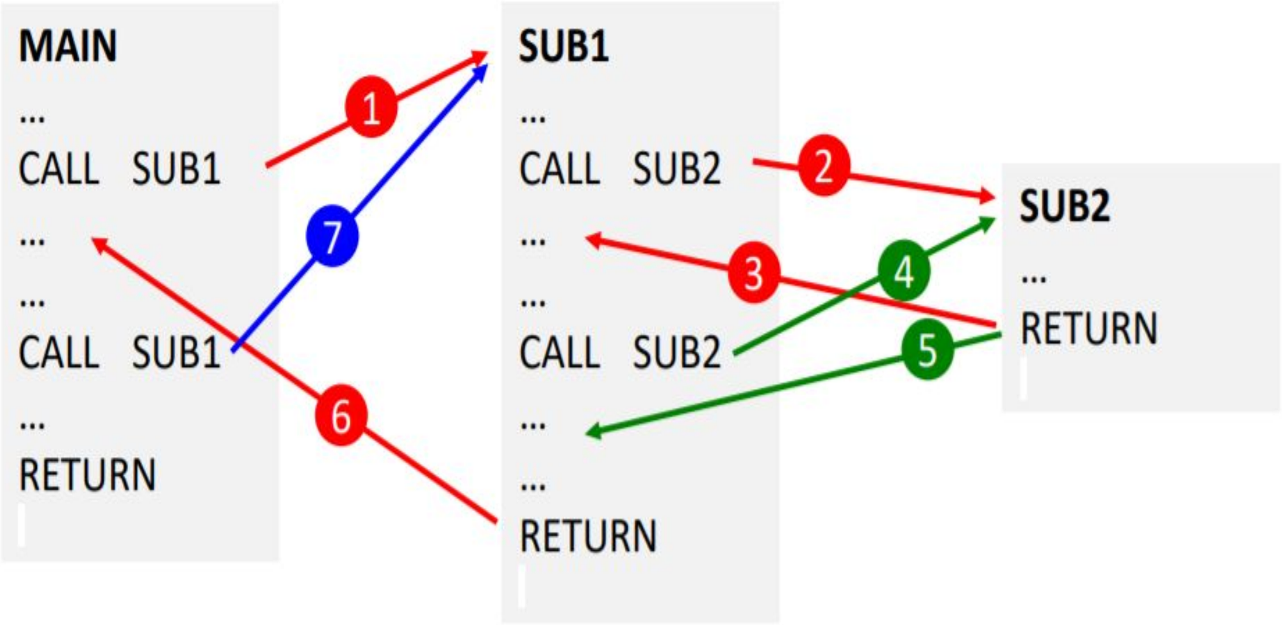
r6 z



mem

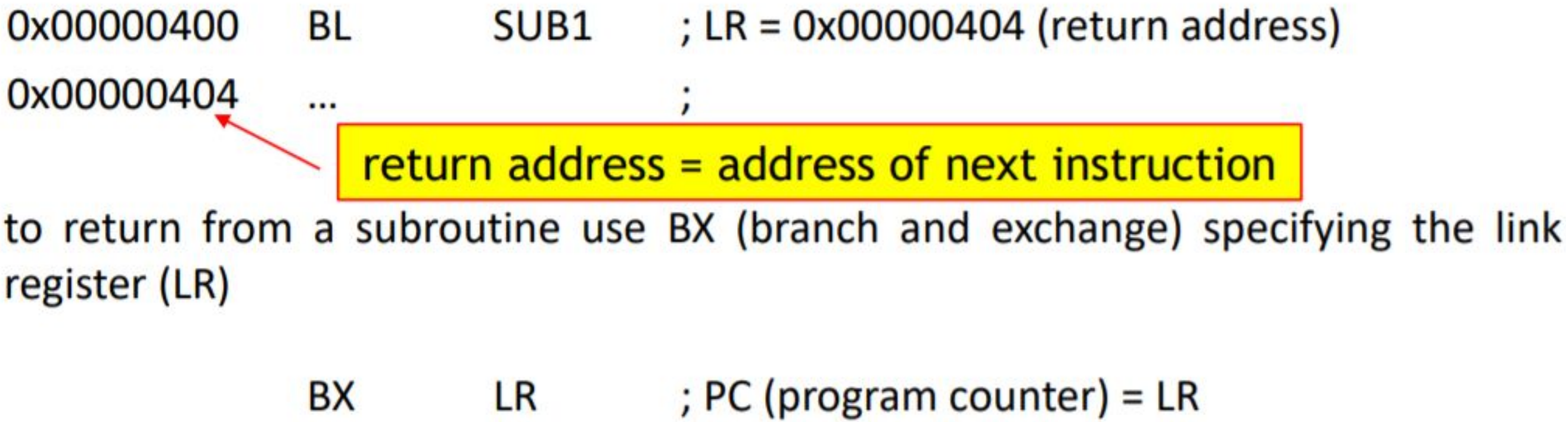
Subroutine

a subroutine is a sequence of instructions that performs a particular task
subroutine called wherever task needs to be performed



Subroutine

- to call a subroutine use the BL (branch and link) instruction
- saves return address in link register (LR = R14)



```
0x00000400     BL        SUB1      ; LR = 0x00000404 (return address)
0x00000404     ...                 ;
```

return address = address of next instruction

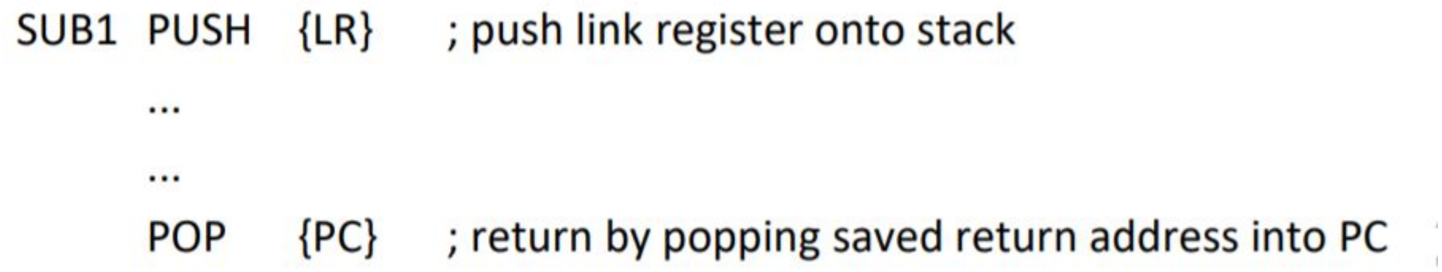
to return from a subroutine use BX (branch and exchange) specifying the link register (LR)

```
               BX        LR        ; PC (program counter) = LR
```

Subroutine

at the start of every non leaf subroutine, push the contents of LR (link register), which contains the return address, onto the system stack

return from a non leaf subroutine by popping return address from stack and assigning to the PC (program counter)



```
SUB1  PUSH   {LR}      ; push link register onto stack
      ...
      ...
      POP    {PC}      ; return by popping saved return address into PC
```

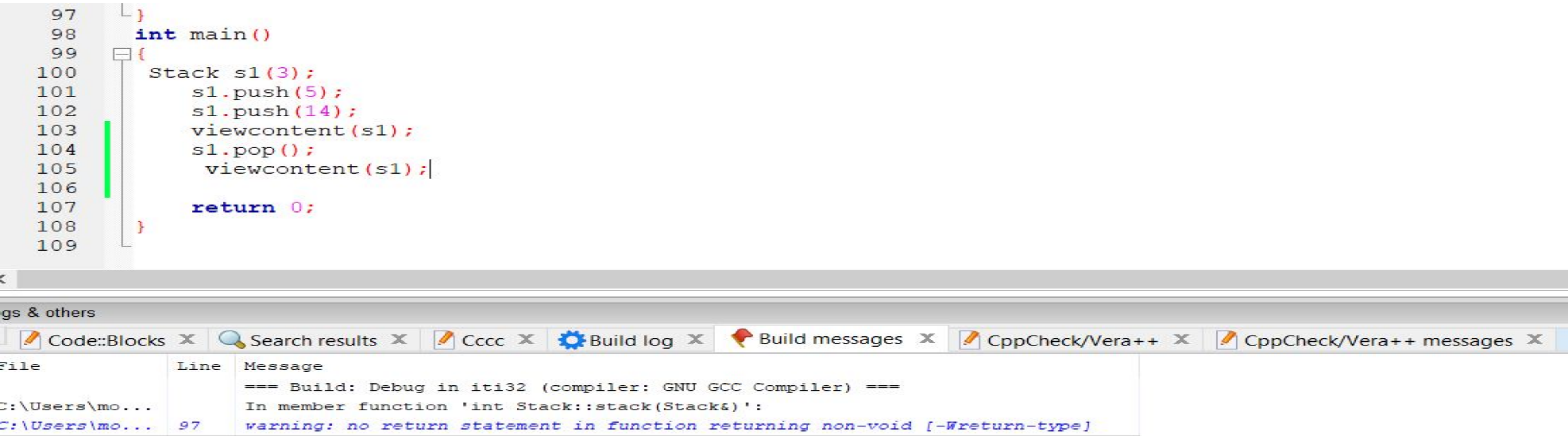
Stack task

Write by c++ stack:

1- user can create stack with any size ,push,pop and view content.

2- guarantee no stack over flow ,or stack underflow.

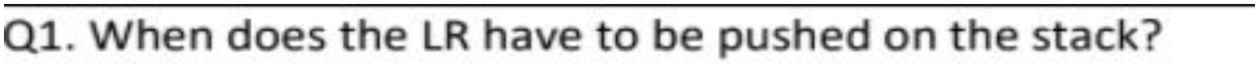
3- use copy constructor.



```
}
int main()
{
 Stack s1(3);
    s1.push(5);
    s1.push(14);
    viewcontent(s1);
    s1.pop();
     viewcontent(s1);

    return 0;
}
```

```
File            Line  Message
                      === Build: Debug in iti32 (compiler: GNU GCC Compiler) ===
C:\Users\mo...        In member function 'int Stack::stack(Stack&)':
C:\Users\mo...  97    warning: no return statement in function returning non-void [-Wreturn-type]
```

Sheet 4



Q1. When does the LR have to be pushed on the stack?

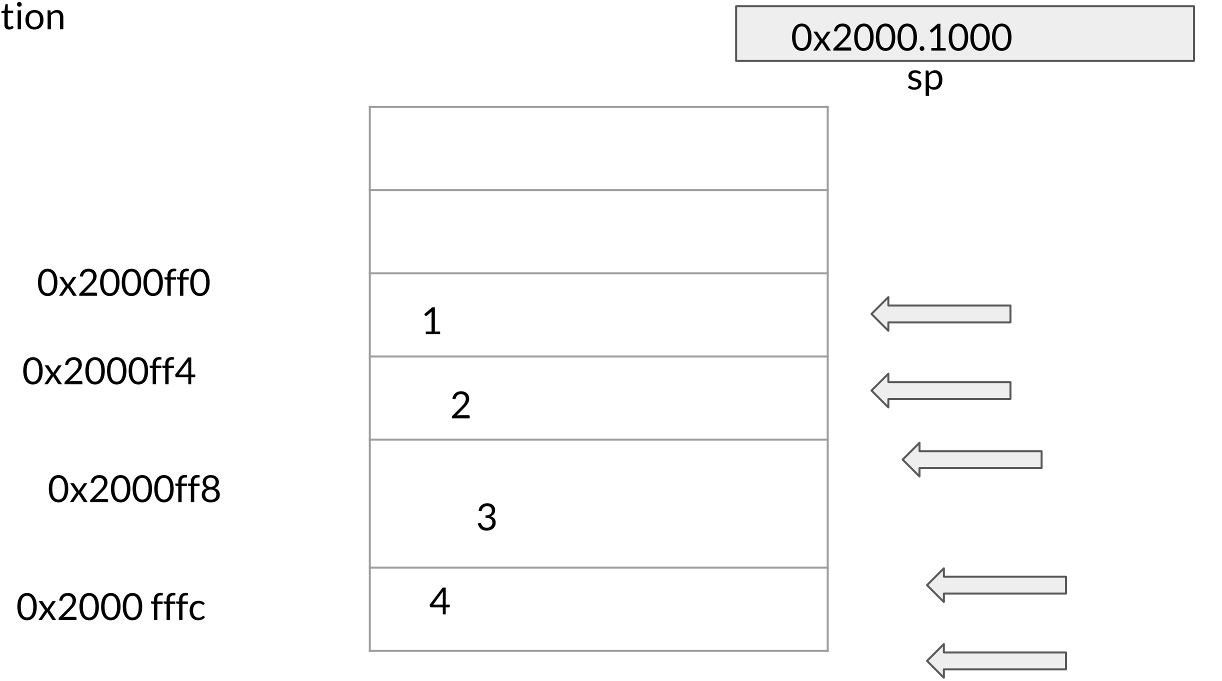
When a function calls another function, the LR is saved in the stack to avoid losing the return address of the caller function.

Q2. Show the SP value and the content of stack after executing this instruction PUSH{R4, R6-R8} assuming

the SP initially equals 0x2000.1000 and R4=1, R6=2, R7=3, R8=4. What will be the values of the registers R0-
R4 after executing this instruction

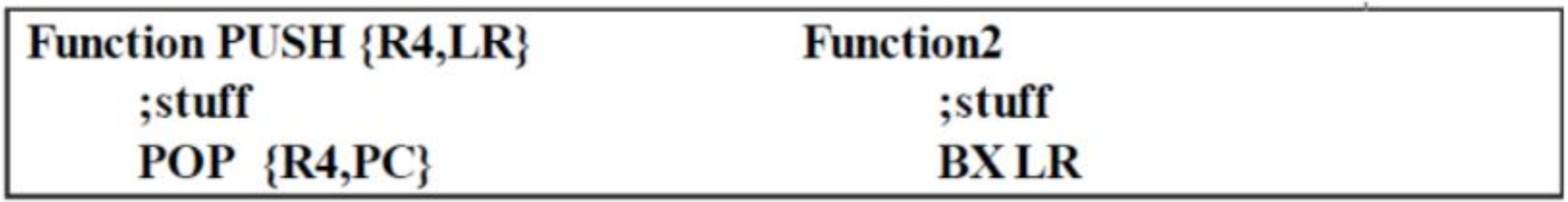
POP{R0-R3}?

R0	1
R1	2
R2	3
r3	4



R0=1, R1=2, R2=3, R3=4

Q3. Explain how does the return from subroutine work in these two functions?

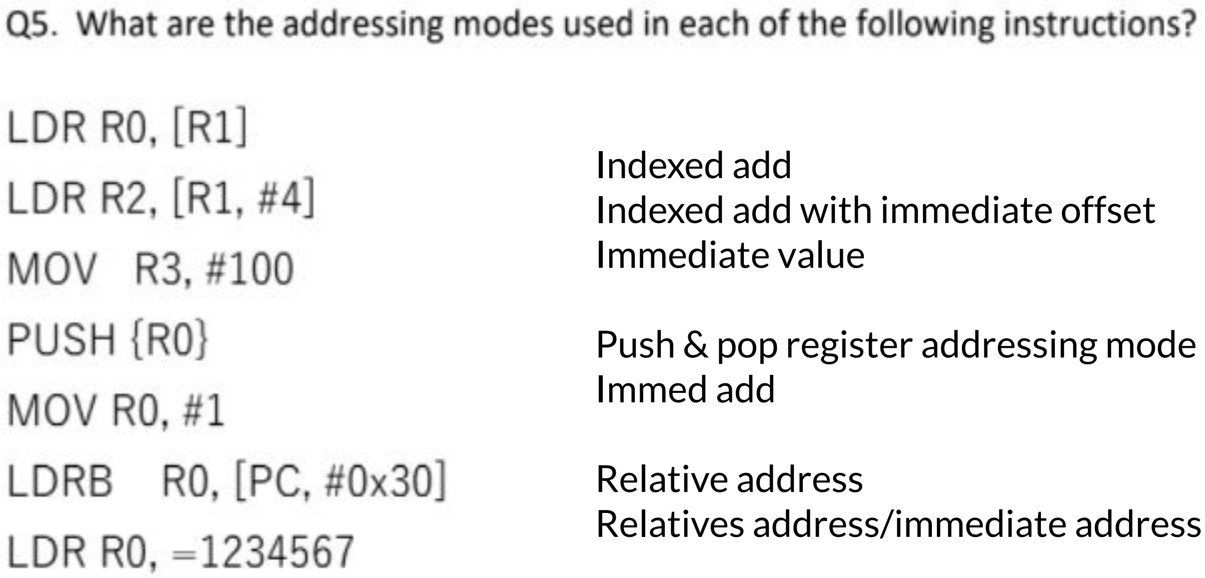


```
Function PUSH {R4,LR}                    Function2
        ;stuff                                   ;stuff
        POP  {R4,PC}                             BX LR
```

Q4. Write assembly code that pushes registers R1, R3, and R5 onto the stack.

Answer

```
PUSH {R1}
PUSH {R3}
PUSH {R5}
```



Q5. What are the addressing modes used in each of the following instructions?

```
LDR R0, [R1]
LDR R2, [R1, #4]
MOV   R3, #100
PUSH {R0}
MOV R0, #1
LDRB    R0, [PC, #0x30]
LDR R0, =1234567
```

Indexed add
Indexed add with immediate offset
Immediate value

Push & pop register addressing mode
Immed add

Relative address
Relatives address/immediate address

BL func pc relative mode

Q6. Write a complete ARM assembly program that calls the procedure func1 which in turn calls a procedure func2. The procedure func2 is defined in Q1 of Sheet 3.

```
     Reset_Handler

      BL func1

     DeadLoop B DeadLoop

func1        PUSH {LR}

             BL func2

             POP {LR}

              BX LR
```