RECURSION IN ASSEMBLY LANGUAGE

Recursion

- A recursive subroutine is one that calls itself, either directly or indirectly. Recursion, the practice of calling recursive subroutines, can be a powerful tool when working with data structures that have repeating patterns. Examples are linked lists and various types of connected graphs where a program must retrace its path.

Endless Recursion

- The most obvious type of recursion occurs when a subroutine calls itself. The following program, for example, has a procedure named endless that calls itself repeatedly without ever stopping:

```
endless: mov ax, bx
     call endless
     ret

start: call endless

     mov ax, 0x4c00
     int 21h
```

Useful recursive subroutines

- **Useful recursive subroutines** always contain a terminating condition. When the terminating condition becomes true, the stack unwinds when the program executes all pending RET instructions.
- There are two cases in recursion: base case and recursive case
- To illustrate, let’s consider the recursive procedure named calcsum, which sums the integers 1 to n,

```
int calsum(int n)
{
   if (n==0)  // termination condition
      return 0
   int x=calsum(n-1) // recursive call
      return n+x
}
```

Useful recursive subroutines

Code in assembly:

Consider array is global and n is an input parameter passed in CX. calcSum returns the sum in AX

```
[0rg 0x0100]
        jmp start
calSum: cmp cx,0 ; check  termination condition
        je l2
        add ax, cx
        dec cx
        call calSum

l2:     ret

start:  mov cx, 4
        mov ax, 0
        call calSum

l1:     mov ax, 4C00h
        int 21h
```

Stack and Registers on Calls

- Calls if n =5
  - ax=0 cx=5 and L1 is pushed on stack (first call to calSum(5))
  - ax=5 cx=4 and L2 is pushed on stack (recursive call to calSum(4))
  - ax=9 cx=3 and L2 is pushed on stack (recursive call to calSum(3))
  - ax=12 cx=2 and L2 is pushed on stack (recursive call to calSum(2))
  - ax=14 cx=1 and L2 is pushed on stack (recursive call to calSum(1))
  - ax=15 cx=0 and L2 is pushed on stack (recursive call to calSum(0))
  - Ret by calSum(0), ax=15 cx=0
  - Ret by calSum(1), ax=15 cx=0
  - Ret by calSum(2), ax=15 cx=0
  - Ret by calSum(3), ax=15 cx=0
  - Ret by calSum(4), ax=15 cx=0
  - Ret by calSum(5), ax=15 cx=0
  - Final answer is in ax=15

STEPS : How to Call the Recursive Function

1. Create Space for Result on Stack (Result space)
2. Prepare and Push input parameters on the stack
3. Call your Recursive Function (Subroutine)
4. Use the results (Pop from stack)

STEPS : To Write the Recursive Function /Subroutine

a) Save all registers used in this function/Subroutine on to the stack
b) Write your base Condition (go to step e after this)
c) Write your non base condition in which you call the same subroutine again(next call using step 1 to 3)
d) Use the results from this call (Pop the result from stack and do all the necessary calculations)
e) Push the result in stack (on Result space created in step 1)
f) restore all registers back from the stack(Saved in Step a)
g) release/pop the stack (Except for the Result - step 1 and e) and Exit

Note: DO ALL THIS WHILE KEEPING IN MIND THE POSITION OF VARIABLES ON STACK

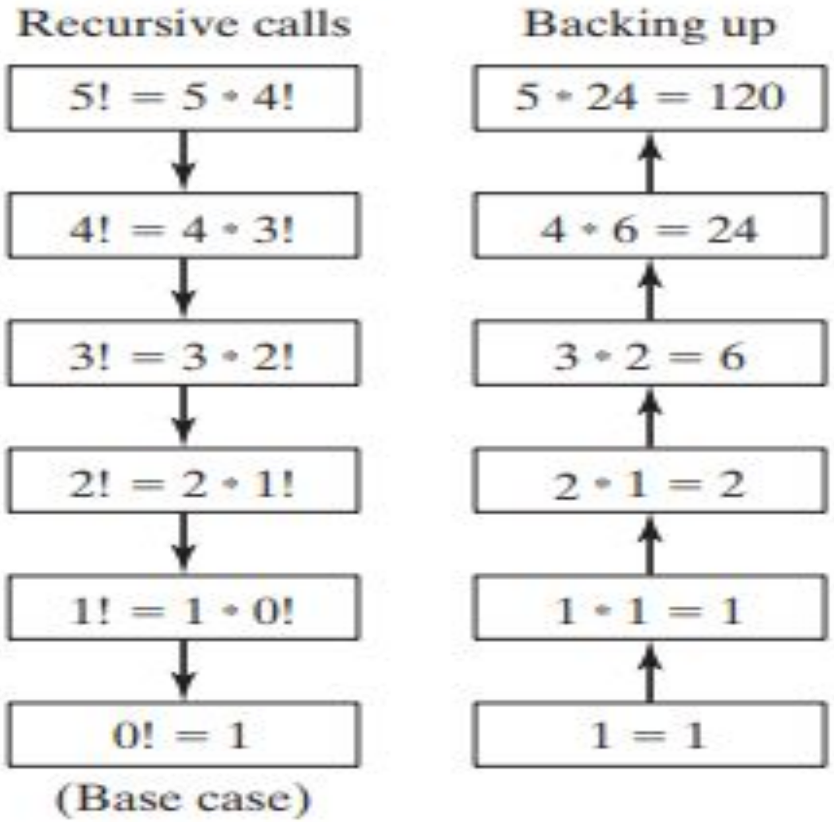
SO that you can access your variables

Example:

This example will show all these steps for Factorial Calculating Function

```
int factorial(int n){

    if(n==0 || n==1) ; Base Condition
        return 1;
    else
        return n*factorial(n-1); Non Base Condition
}
```

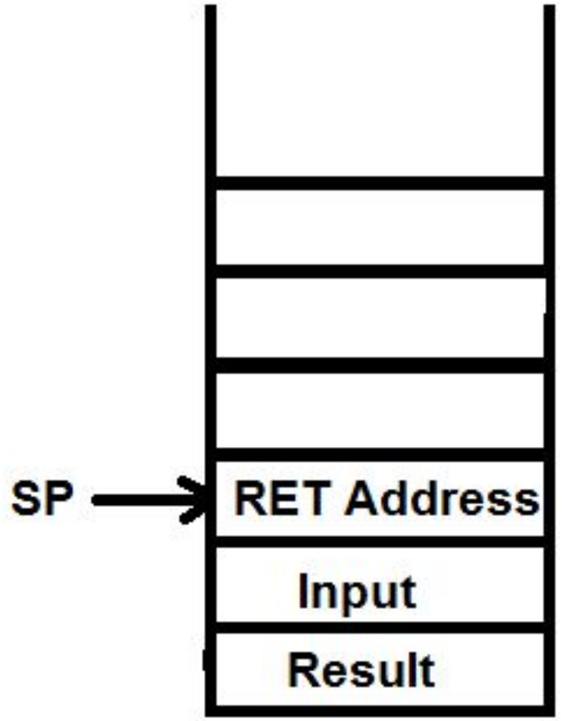

Recursive calls
Backing up
5! = 5 * 4!
4! = 4 * 3!
3! = 3 * 2!
2! = 2 * 1!
1! = 1 * 0!
0! = 1
(Base case)
5 * 24 = 120
4 * 6 = 24
3 * 2 = 6
2 * 1 = 2
1 * 1 = 1
1 = 1

A). First Write your Main

main:

push ax ; *Step 1 :Create Space for Result on Stack (Result space)*
push 3 ; *Step 2 : Push input parameters on the stack*
call fact ; *Step 3: Call your Recursive Function (Subroutine)*
pop ax ; *Step 4: Use the results (Pop from stack)*
;ax will have the result of fact

mov ax,0x4c00 ; Terminate
int 21h



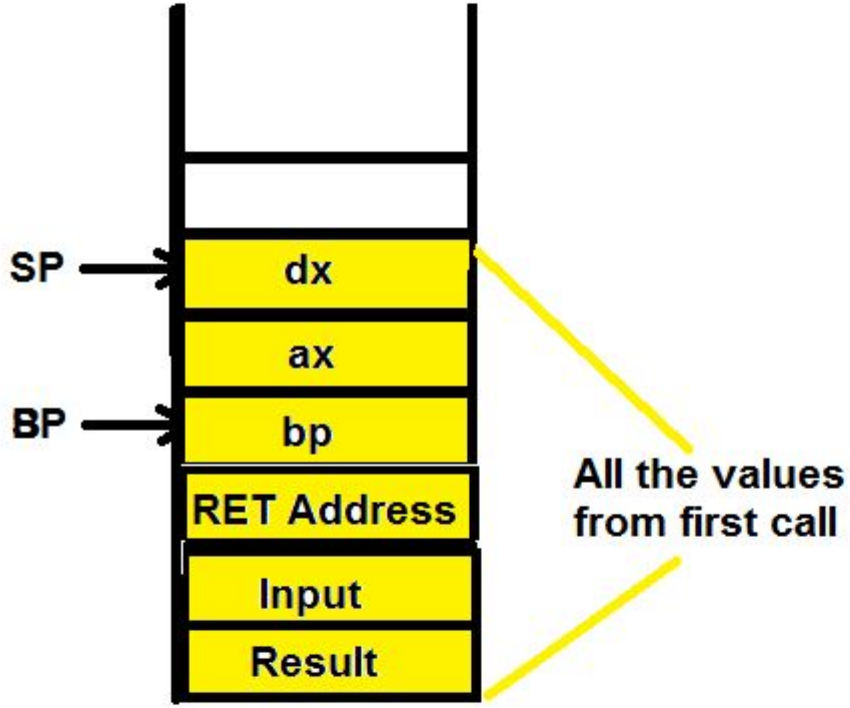
Stack after Step 3

B). Write your Subroutine

Step a. Save all registers used in this function/Subroutine on to the stack

```
push bp
mov bp,sp
push ax
push dx
```

Note: You can modify this step on end once you finalize which registers you will be using in your subroutine



Step b. Write your base Condition

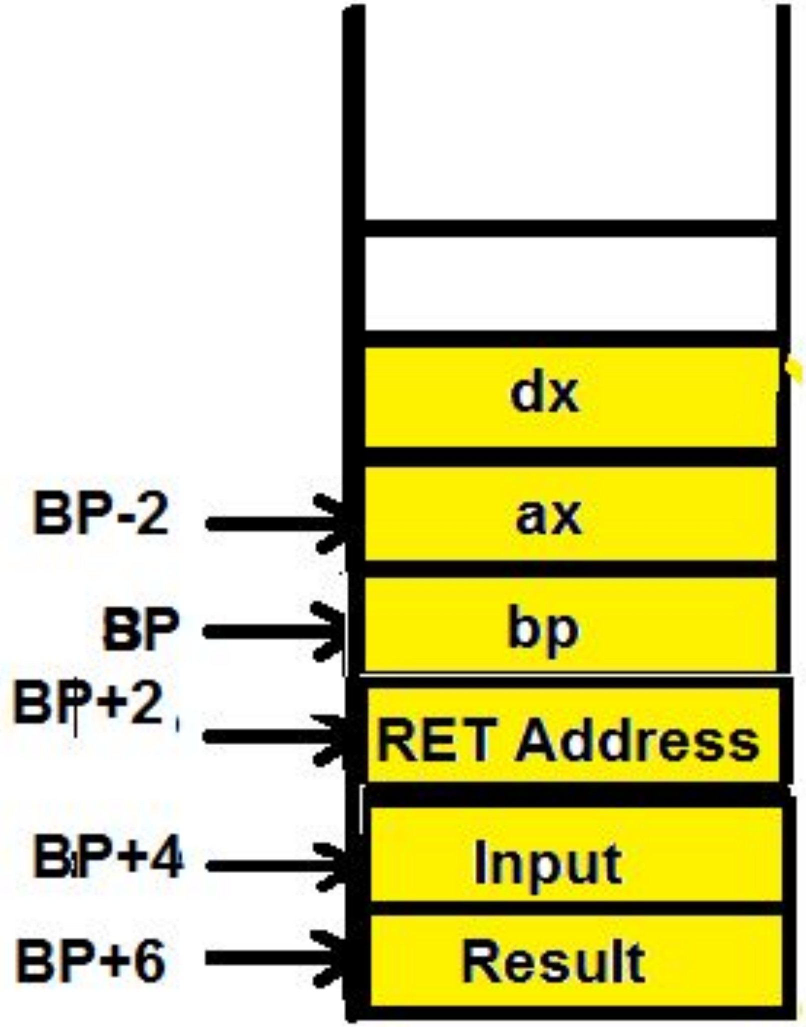
In C++

```
if(n==0 || n==1) ; Base Condition
        return 1;
```

In Assembly

```
cmp word [bp+4] , 1 ; if n==1
je exit
cmp word [bp+4] , 0 ; if n==0
je exit

exit:
mov word [bp+6], 1  ; Return 1 (Placing 1 at result position) Step e.
```



Step c . Write your non base condition in which you call the same subroutine again(next call using step 1 to 3)

```
sub sp, 2   ; Step 1 Create Space for Result on Stack (Result space)
mov ax, [bp+4] ; Creating the input
dec ax
push ax     ; Step 2 : Push input parameters on the stack (n-1)
call fact   ; Step 3: Call your Recursive Function (Subroutine)
```

Step d . Use the results from this call (Pop the result do all the necessary calculations)

```
pop ax      ; returned result
mov dx,0 ; Initialization for MUL
mul word [bp+4]  ; ax= n* f(n-1) i.e. [bp+4] * ax
```

Note : BP+4 was the input for this call

Step e. Push the result in stack (on Result space created in step 1)

mov [bp+6], ax ; *result*

Step f. restore all registers back from the stack(Saved in Step a)

jmp exit2

exit2:
pop dx
pop ax
pop bp
; Note this is in reverse order as of Step a where we push bp first then ax and rhen dx

Step g. release/pop the stack and Return (Except for the Result - step 1 and e)

Ret2

Summing Up all the code

```
[org 0x100]

jmp main

fact:
        push bp
        mov bp,sp

        push ax
        push dx


        ;Base Condition
        cmp word [bp+4] , 1 ;comparing n recieved in parameters is equal to 1 or not
        je exit
        cmp word [bp+4] , 0 ; if n==0
        je exit
```

```
; prepare for next call
sub sp, 2    ; space for return value
mov ax, [bp+4]
dec ax
push ax      ; pushing n-1
call fact
pop ax       ; return result

mov dx,0
mul word [bp+4]  ; n* f(n-1) i.e. [bp+4] * ax
mov [bp+6], ax  ; result
jmp exit2
```

```
exit:   mov word [bp+6], 1    ; base case return
exit2:

        pop dx
        pop ax


        pop bp
        ret 2





main:

        push ax  ; dummy push for func return value
        push 3   ; to calculate 4!
        call fact

        pop ax   ; ax will have the result of fact

        mov ax,0x4c00
        int 21h
```

Exercise 1 : Fibonacci

Go through all these steps to create a recursive function FIB

```
int fibo(int x){

if (x==1 || x==0)
    return x;

else
    return fib(x-1)+fib(x-2);
}
```

Exercise 2 : Palindrome

Write this recursive function in Assembly language following all the steps given in slides

```
int palindrome(char* str, int length) {
    if (length<=1)
        return 1;
    else
        return (palindrome(str+1, length-2) && (*str== *(str+length-1)));
}
```

Exercise 3 : Tower of Hanoi

- Follow the link to understand the problem of tower of Hanoi and using the given pseudo code on link write an assembly language code to solve this problem

https://www.cs.cmu.edu/~cburch/survey/recurse/hanoiimpl.html

References

- https://sites.google.com/view/coal-fall-2019