Embedded Systems with ARM Cortex-M Microcontrollers in Assembly Language and C

Chapter 8
Stack and Recursive Functions

Dr. Yifeng Zhu
Electrical and Computer Engineering
University of Maine

Spring 2020

Recursive Functions

- A recursive function is one that solves its task by calling itself on smaller pieces of data.
- An effective tactic is to
  - divide a problem into sub-problems of the same type as the original,
  - solve those sub-problems, and
  - combine the results

Defining Factorial(n)

Product of the first n numbers

$$1 \times 2 \times 3 \times \ldots \times n$$

```
factorial(0) = 1
factorial(1) = 1                = 1 × factorial(0)
factorial(2) = 2×1              = 2 × factorial(1)
factorial(3) = 3×2×1            = 3 × factorial(2)
factorial(4) = 4×3×2×1          = 4 × factorial(3)
factorial(n) = n×(n-1)×…×1      = n × factorial(n-1)
```

Classic Example: Factorial

- Factorial is the classic example:
  - `6! = 6 × 5!`
  - `6! = 6 × 5 x 4!`

  `...`
  - `6! = 6 × 5 × 4 × 3 × 2 × 1`
- The factorial function can be easily written as a recursive function:

```
int Factorial(int n) {

    if (n < 2)
        return 1; /* base case */

    return (n * Factorial(n - 1));

}
```

Classic Example: Fibonacci Numbers

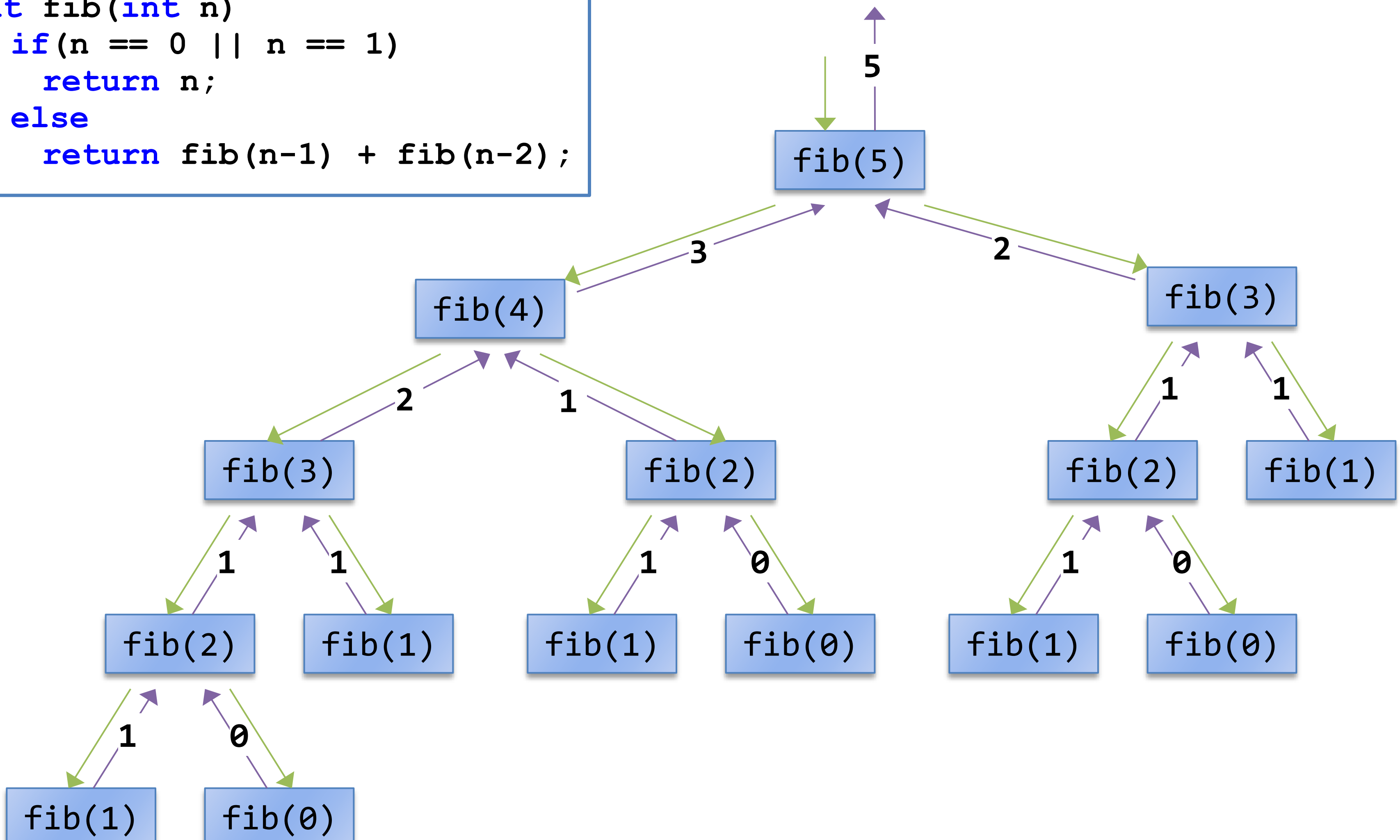
```
f(n) = f(n-1) + f(n-2)
f(0) = 1
f(1) = 1
```

1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, …

```
int Fibonacci(int n) {

  if (n <= 1)
    return 1;   /* base case */

  return (Fibonacci(n-1) + Fibonacci(n-2));

}
```

Analysis of fib(5)

```
int fib(int n)
   if(n == 0 || n == 1)
     return n;
   else
     return fib(n-1) + fib(n-2);
```



Example of Recursive Function: Testing Palindrome

```
bool isPalindrome(char* s, int len) {
  if(len < 2)
    return TRUE;
  else
    return s[0] == s[len-1] && isPalindrome(&s[1], len-2);
}
```

Recursion vs Iteration

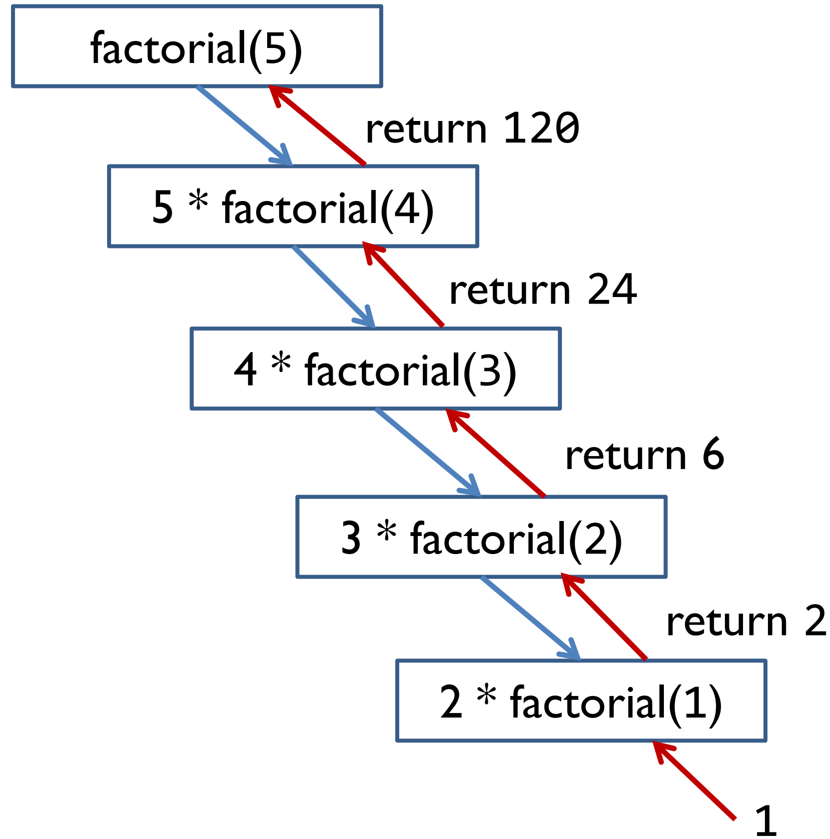
Any problem that can be solved recursively can also be solved iteratively (using loop).

Recursive functions vs Iterative functions

- Cons:
  - Recursive functions are slow
  - Recursive function take more memory
- Pros
  - Recursive functions resembles the problem more naturally
  - Recursive function are easier to program, and debug.

Recursive Factorial in C

```c
int factorial(int n);

int main(void){
  factorial(5);
  return 0;
}

int factorial(int n) {
  int f;
  if(n==1)
    return 1;
  else
    f = n * factorial(n-1);
  return f;
}
```

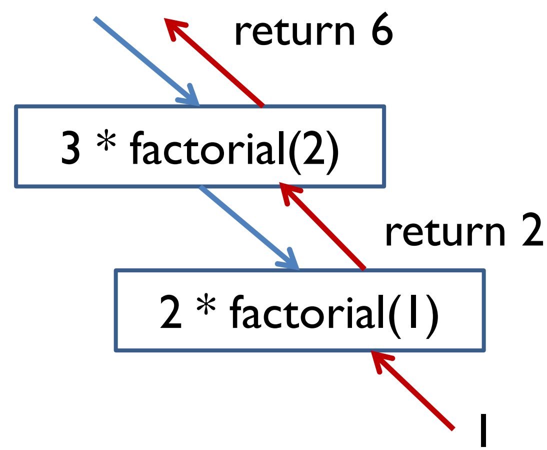


Recursive Functions

- `PUSH LR` (& working registers) onto stack before nested call
- `POP LR` (& working registers) off stack after nested return

Recursive Factorial in Assembly

```
                  AREA main, CODE, READONLY
                  EXPORT   __main

                  ENTRY

            __main PROC
                  MOV r0, #0x03
0x08000130        BL   factorial
0x08000134  stop  B    stop
                  ENDP

            factorial
0x08000136        PUSH {r4, lr}
0x08000138        MOV   r4, r0
0x0800013A        CMP   r4, #0x01
0x0800013C        BNE   NZ
0x0800013E        MOVS  r0, #0x01
0x08000140  loop  POP   {r4, pc}
0x08000142  NZ    SUBS  r0, r4, #1
0x08000144        BL    factorial
0x08000148        MUL   r0, r4, r0
0x0800014C        B     loop

                  END
```



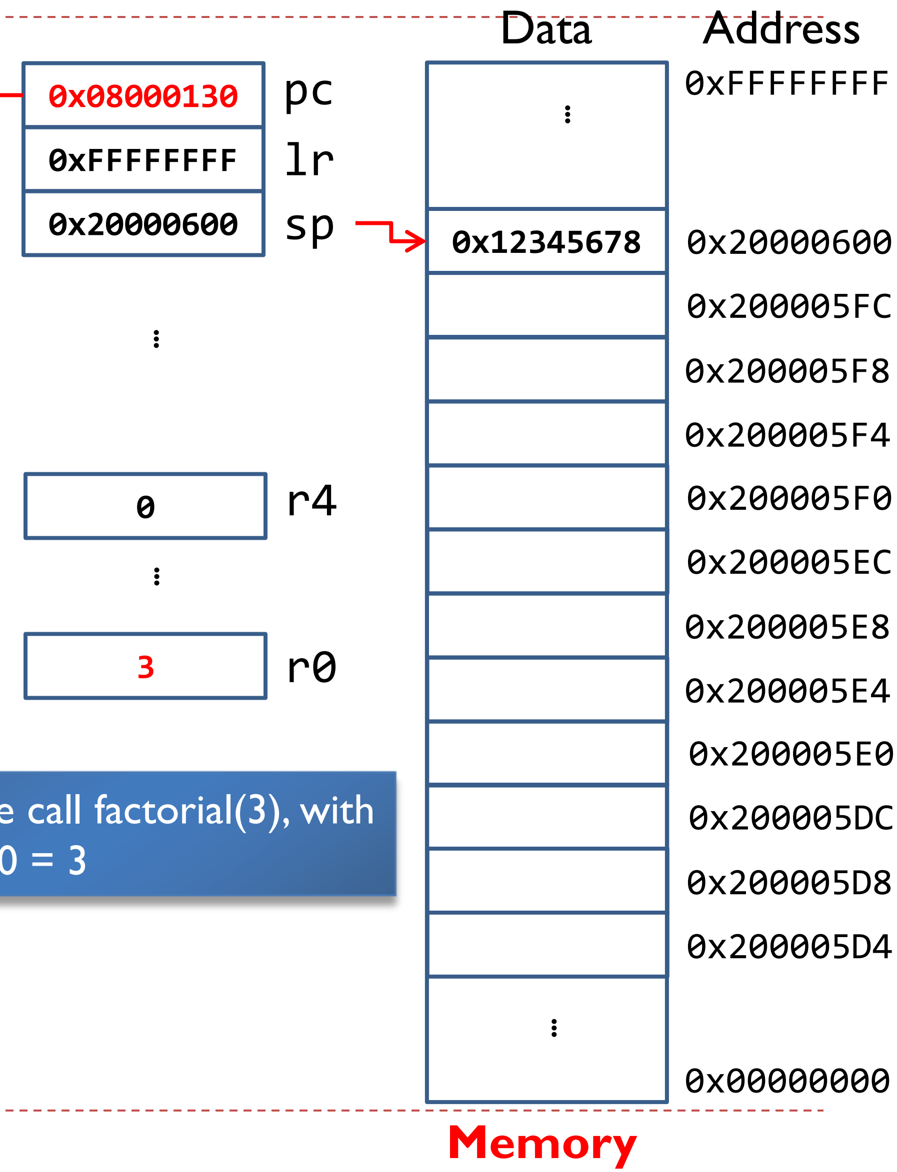
Recursive Factorial in Assembly

```
 AREA main, CODE, READONLY
     EXPORT  __main
     ENTRY
__main PROC
    MOV   r0, #0x03                   ; Set argument n = 3 in r0 (r0 holds first arg)
    BL    factorial                   ; Call factorial(n); LR gets return address;
                                       ; result returns in r0
stop
    B     stop                        ; Halt by branching to self (infinite loop)
    ENDP
; --- Recursive factorial(n) ---
factorial
    PUSH  {r4, lr}                    ; Save callee-used r4 and return address LR on
stack
    MOV   r4, r0                      ; Preserve n in r4 across the recursive call
    CMP   r4, #0x01                   ; Check base case: n == 1 ?
    BNE   NZ                          ; If n != 1, branch to NZ for recursive case
    MOVS  r0, #0x01                   ; Base case: return 1 in r0
loop
    POP   {r4, pc}                    ; restore r4, and return by popping PC
NZ
    SUBS  r0, r4, #1                  ; Prepare argument r0 = n - 1 for recursive call
    BL    factorial                   ; r0 <- factorial(n-1) after return
    MUL   r0, r4, r0                  ; r0 = n * factorial(n-1)
    B     loop
    END
```

Recursive Factorial in Assembly

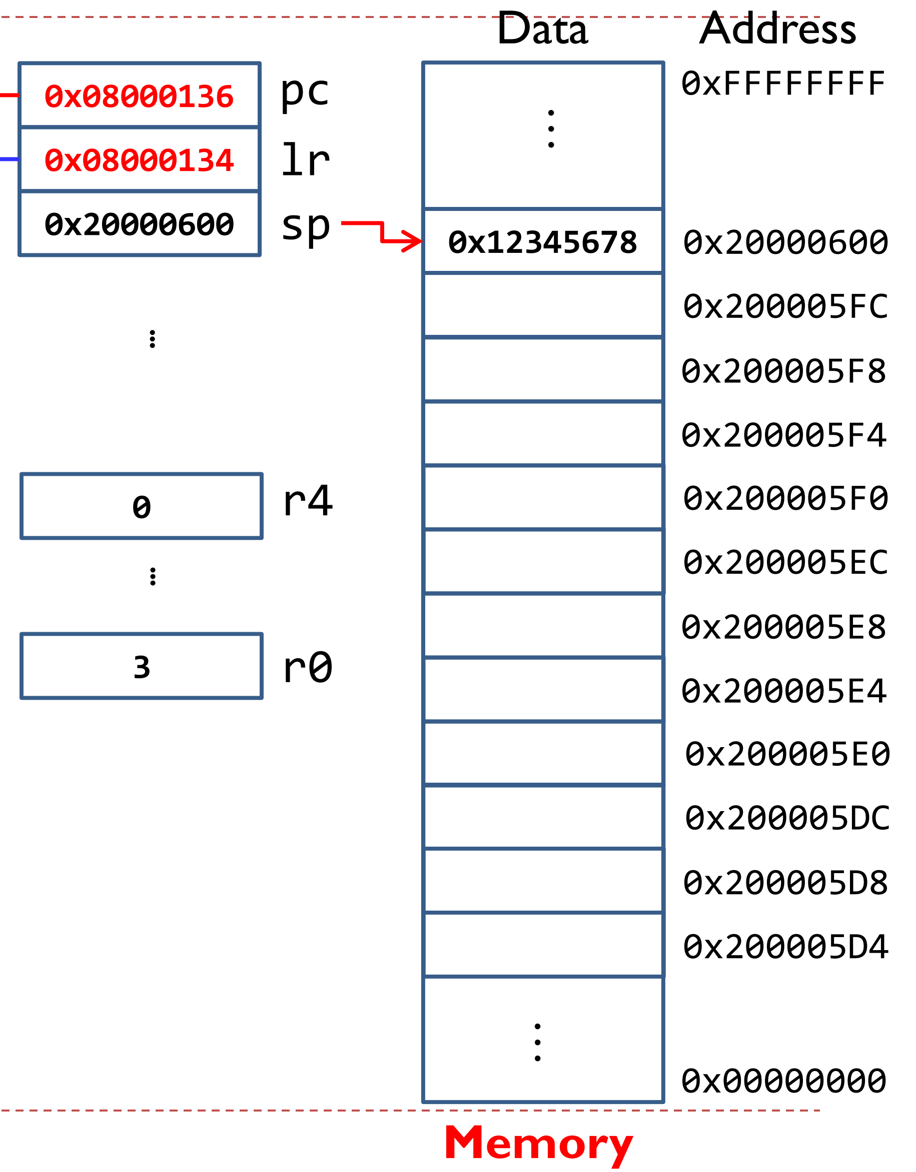
```
                AREA main, CODE, READONLY
                EXPORT   __main

                ENTRY

           __main PROC
                MOV r0, #0x03
0x08000130      BL   factorial
0x08000134 stop B    stop
                ENDP

           factorial
0x08000136      PUSH {r4, lr}
0x08000138      MOV  r4, r0
0x0800013A      CMP  r4, #0x01
0x0800013C      BNE  NZ
0x0800013E      MOVS r0, #0x01
0x08000140 loop POP  {r4, pc}
0x08000142 NZ   SUBS r0, r4, #1
0x08000144      BL   factorial
0x08000148      MUL  r0, r4, r0
0x0800014C      B    loop

                END
```

1st recursive call factorial(3), with argument r0 = 3



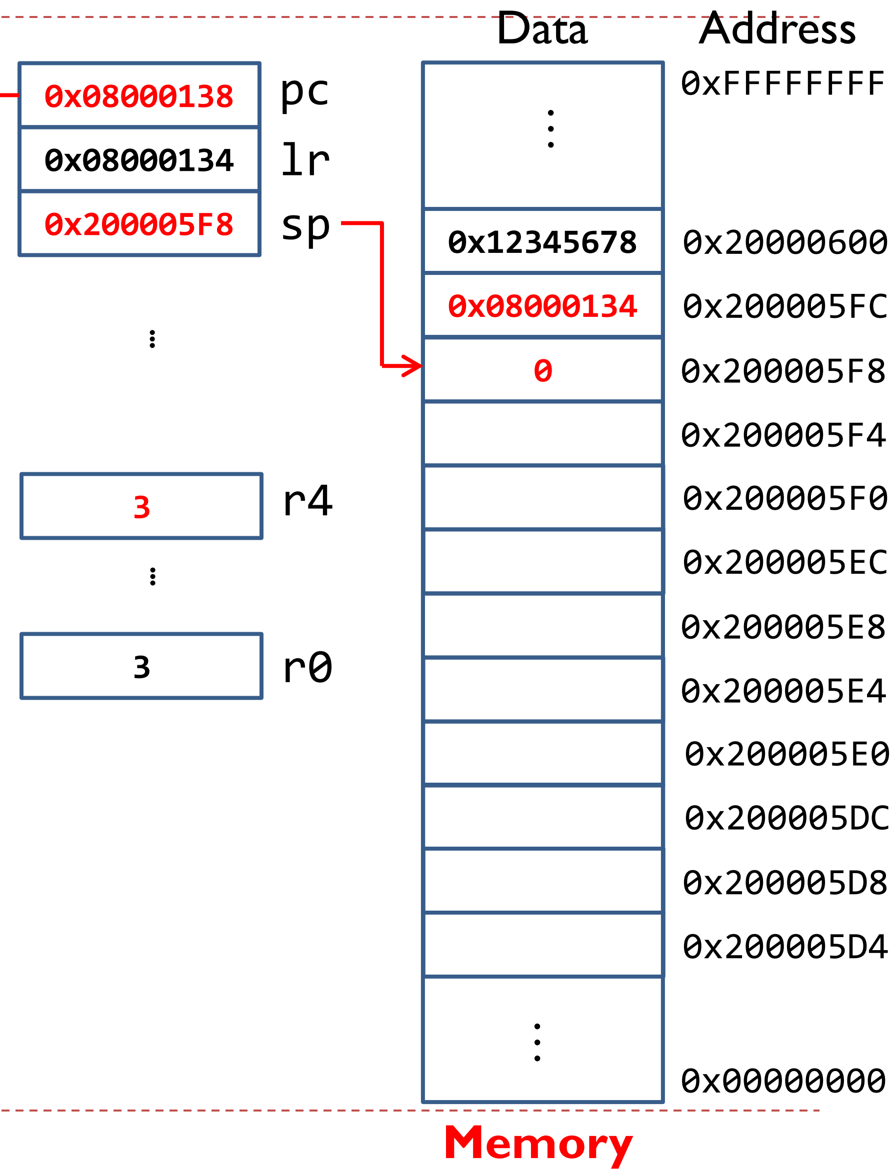
Recursive Factorial in Assembly

```
            AREA main, CODE, READONLY
            EXPORT   __main

            ENTRY

       __main PROC
            MOV r0, #0x03
0x08000130  BL  factorial
0x08000134 stop B   stop
            ENDP

       factorial
0x08000136  PUSH {r4, lr}
0x08000138  MOV  r4, r0
0x0800013A  CMP  r4, #0x01
0x0800013C  BNE  NZ
0x0800013E  MOVS r0, #0x01
0x08000140 loop POP  {r4, pc}
0x08000142 NZ   SUBS r0, r4, #1
0x08000144  BL   factorial
0x08000148  MUL  r0, r4, r0
0x0800014C  B    loop

            END
```



Recursive Factorial in Assembly

```
                AREA main, CODE, READONLY
                EXPORT  __main

                ENTRY

           __main PROC
                MOV r0, #0x03
0x08000130      BL  factorial
0x08000134 stop B   stop
                ENDP

           factorial
0x08000136      PUSH {r4, lr}
0x08000138      MOV  r4, r0
0x0800013A      CMP  r4, #0x01
0x0800013C      BNE  NZ
0x0800013E      MOVS r0, #0x01
0x08000140 loop POP  {r4, pc}
0x08000142 NZ   SUBS r0, r4, #1
0x08000144      BL   factorial
0x08000148      MUL  r0, r4, r0
0x0800014C      B    loop

                END
```



Recursive Factorial in Assembly

```
                  AREA main, CODE, READONLY
                  EXPORT   __main

                  ENTRY

           __main PROC
                  MOV r0, #0x03
0x08000130        BL  factorial
0x08000134 stop B    stop
                  ENDP

           factorial
0x08000136        PUSH {r4, lr}
0x08000138        MOV  r4, r0
0x0800013A        CMP  r4, #0x01
0x0800013C        BNE  NZ
0x0800013E        MOVS r0, #0x01
0x08000140 loop POP  {r4, pc}
0x08000142 NZ     SUBS r0, r4, #1
0x08000144        BL   factorial
0x08000148        MUL  r0, r4, r0
0x0800014C        B    loop

                  END
```

pc	0x08000144
lr	0x08000134
sp	0x200005F8
⋮	
r4	3
⋮	
r0	2

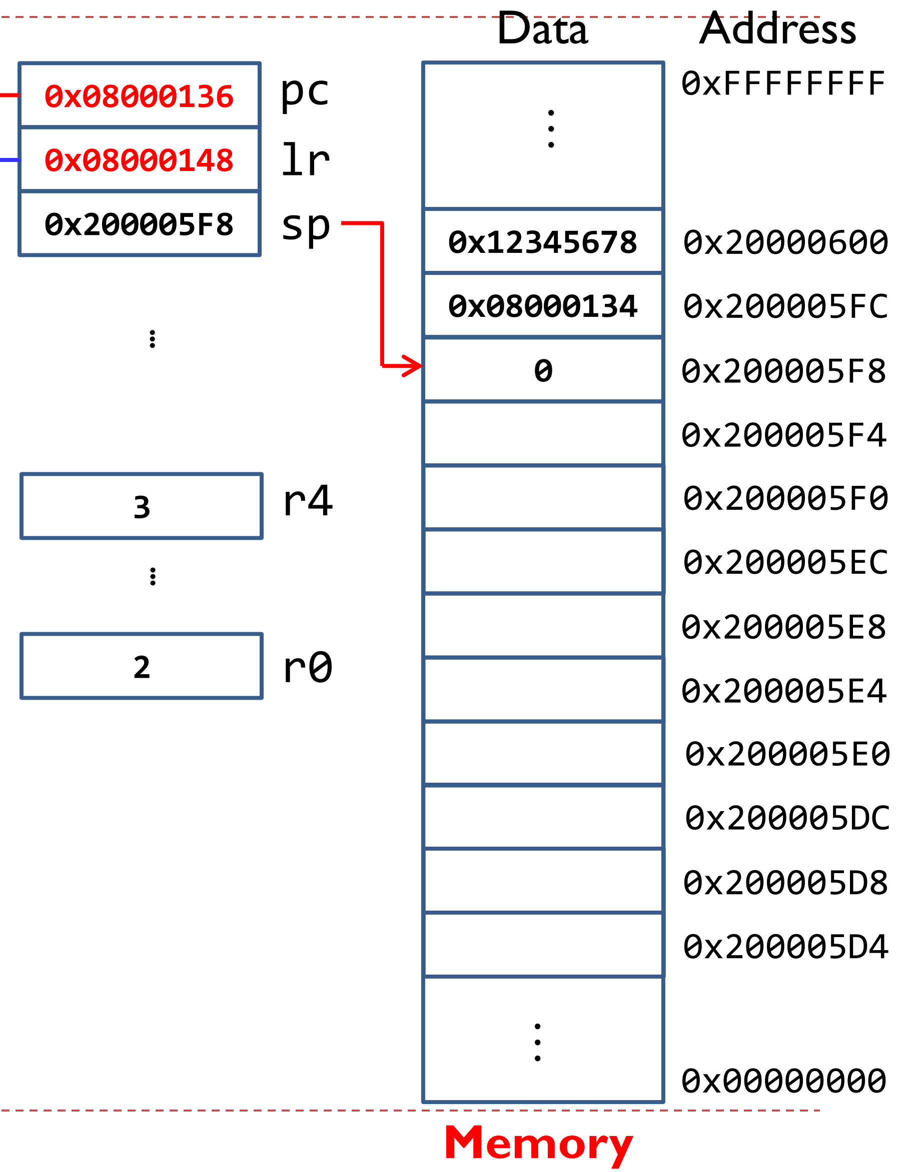
2nd recursive call factorial(2), with argument r0 = 2

Data	Address
⋮	0xFFFFFFFF
0x12345678	0x20000600
0x08000134	0x200005FC
0	0x200005F8
	0x200005F4
	0x200005F0
	0x200005EC
	0x200005E8
	0x200005E4
	0x200005E0
	0x200005DC
	0x200005D8
	0x200005D4
⋮	0x00000000

Memory

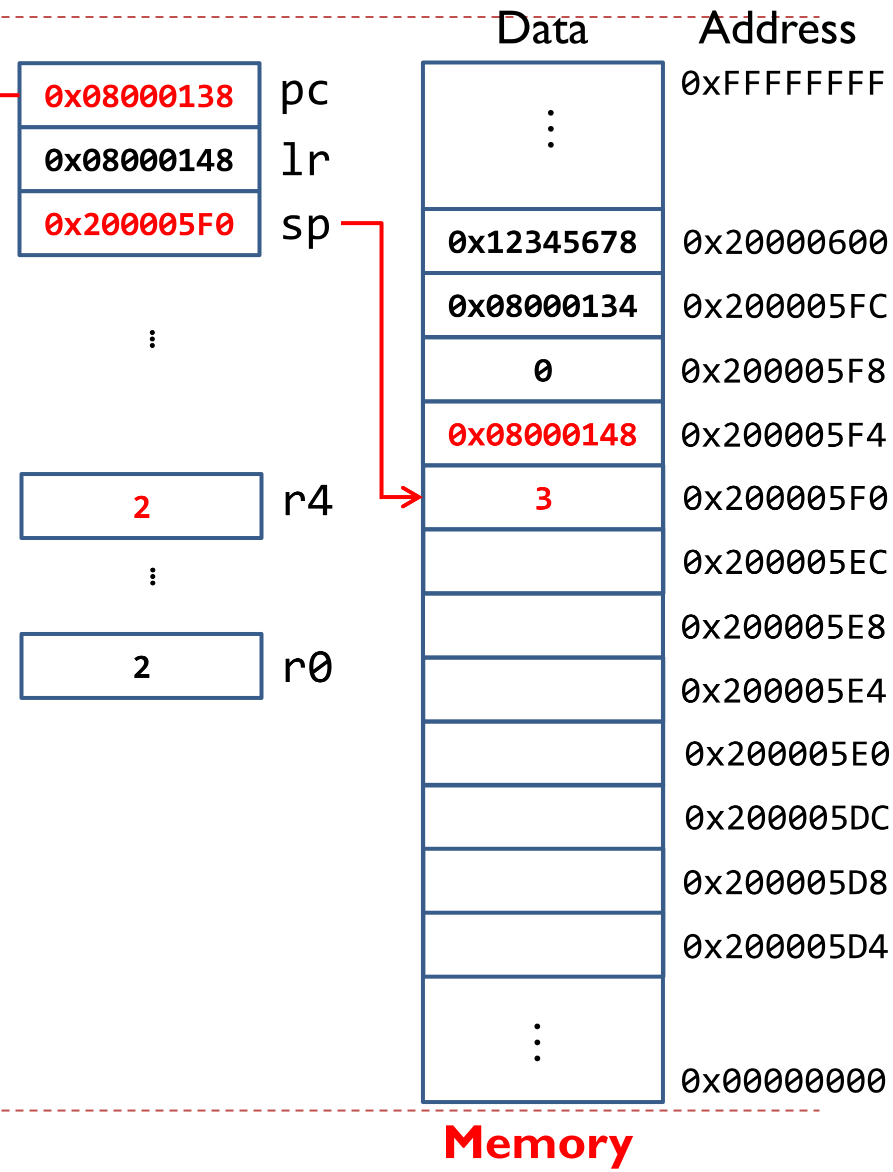
Recursive Factorial in Assembly

```
                AREA main, CODE, READONLY
                EXPORT  __main

                ENTRY

           __main PROC
                MOV r0, #0x03
0x08000130      BL  factorial
0x08000134 stop B   stop
                ENDP

           factorial
0x08000136      PUSH {r4, lr}
0x08000138      MOV  r4, r0
0x0800013A      CMP  r4, #0x01
0x0800013C      BNE  NZ
0x0800013E      MOVS r0, #0x01
0x08000140 loop POP  {r4, pc}
0x08000142 NZ   SUBS r0, r4, #1
0x08000144      BL   factorial
0x08000148      MUL  r0, r4, r0
0x0800014C      B    loop

                END
```



Recursive Factorial in Assembly

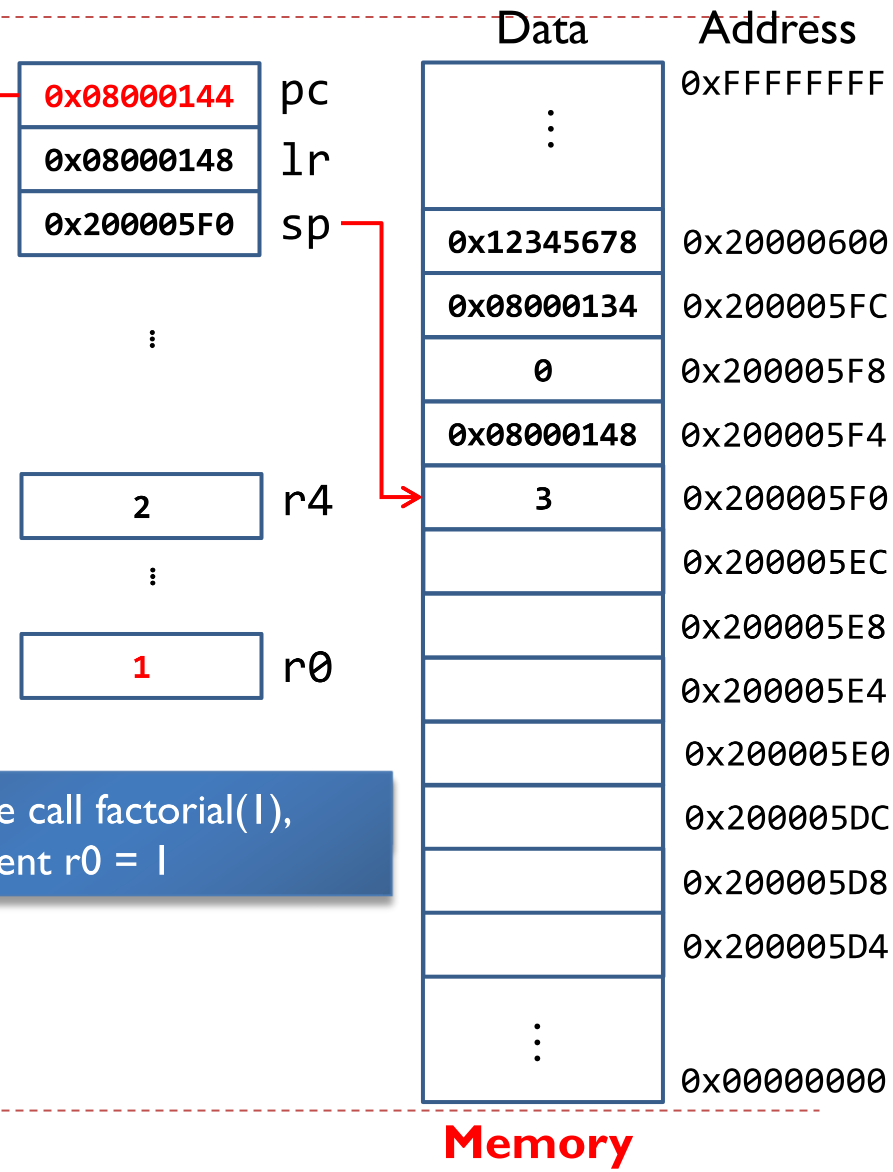
```
                   AREA main, CODE, READONLY
                   EXPORT   __main

                   ENTRY

          __main PROC
                   MOV r0, #0x03
0x08000130         BL   factorial
0x08000134  stop B     stop
                   ENDP

          factorial
0x08000136         PUSH {r4, lr}
0x08000138         MOV   r4, r0
0x0800013A         CMP   r4, #0x01
0x0800013C         BNE   NZ
0x0800013E         MOVS r0, #0x01
0x08000140  loop POP   {r4, pc}
0x08000142  NZ     SUBS r0, r4, #1
0x08000144         BL    factorial
0x08000148         MUL   r0, r4, r0
0x0800014C         B     loop

                   END
```



Recursive Factorial in Assembly

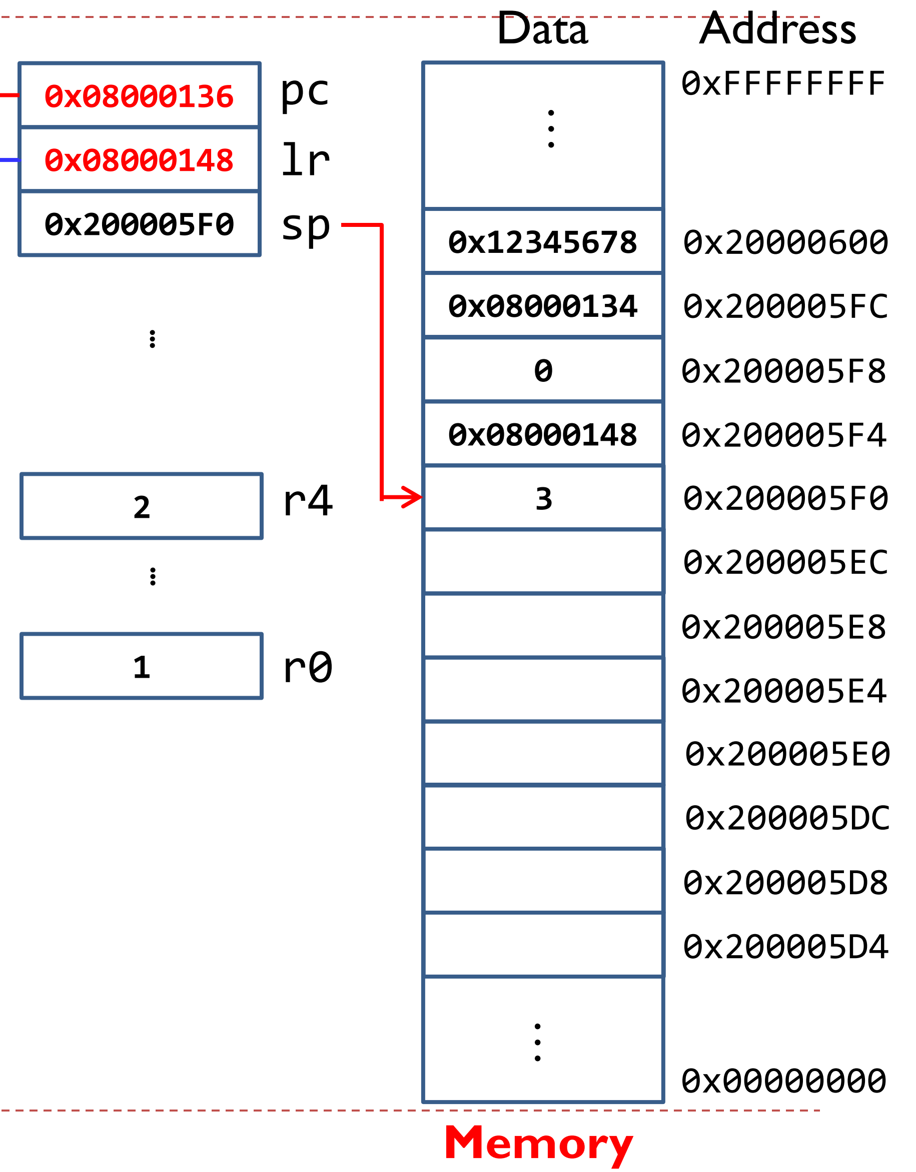
```
                 AREA main, CODE, READONLY
                 EXPORT   __main

                 ENTRY

           __main PROC
                 MOV r0, #0x03
0x08000130       BL   factorial
0x08000134 stop  B    stop
                 ENDP

           factorial
0x08000136       PUSH {r4, lr}
0x08000138       MOV  r4, r0
0x0800013A       CMP  r4, #0x01
0x0800013C       BNE  NZ
0x0800013E       MOVS r0, #0x01
0x08000140 loop  POP  {r4, pc}
0x08000142 NZ    SUBS r0, r4, #1
0x08000144       BL   factorial
0x08000148       MUL  r0, r4, r0
0x0800014C       B    loop

                 END
```

3rd recursive call factorial(1), with argument r0 = 1



Recursive Factorial in Assembly

```
                AREA main, CODE, READONLY
                EXPORT   __main

                ENTRY

           __main PROC
                MOV r0, #0x03
0x08000130      BL   factorial
0x08000134 stop B    stop
                ENDP

           factorial
0x08000136      PUSH {r4, lr}
0x08000138      MOV  r4, r0
0x0800013A      CMP  r4, #0x01
0x0800013C      BNE  NZ
0x0800013E      MOVS r0, #0x01
0x08000140 loop POP  {r4, pc}
0x08000142 NZ   SUBS r0, r4, #1
0x08000144      BL   factorial
0x08000148      MUL  r0, r4, r0
0x0800014C      B    loop

                END
```



Recursive Factorial in Assembly

```
                  AREA main, CODE, READONLY
                  EXPORT  __main

                  ENTRY

           __main PROC
                  MOV r0, #0x03
0x08000130        BL  factorial
0x08000134 stop   B   stop
                  ENDP

           factorial
0x08000136        PUSH {r4, lr}
0x08000138        MOV  r4, r0
0x0800013A        CMP  r4, #0x01
0x0800013C        BNE  NZ
0x0800013E        MOVS r0, #0x01
0x08000140 loop   POP  {r4, pc}
0x08000142 NZ     SUBS r0, r4, #1
0x08000144        BL   factorial
0x08000148        MUL  r0, r4, r0
0x0800014C        B    loop

                  END
```

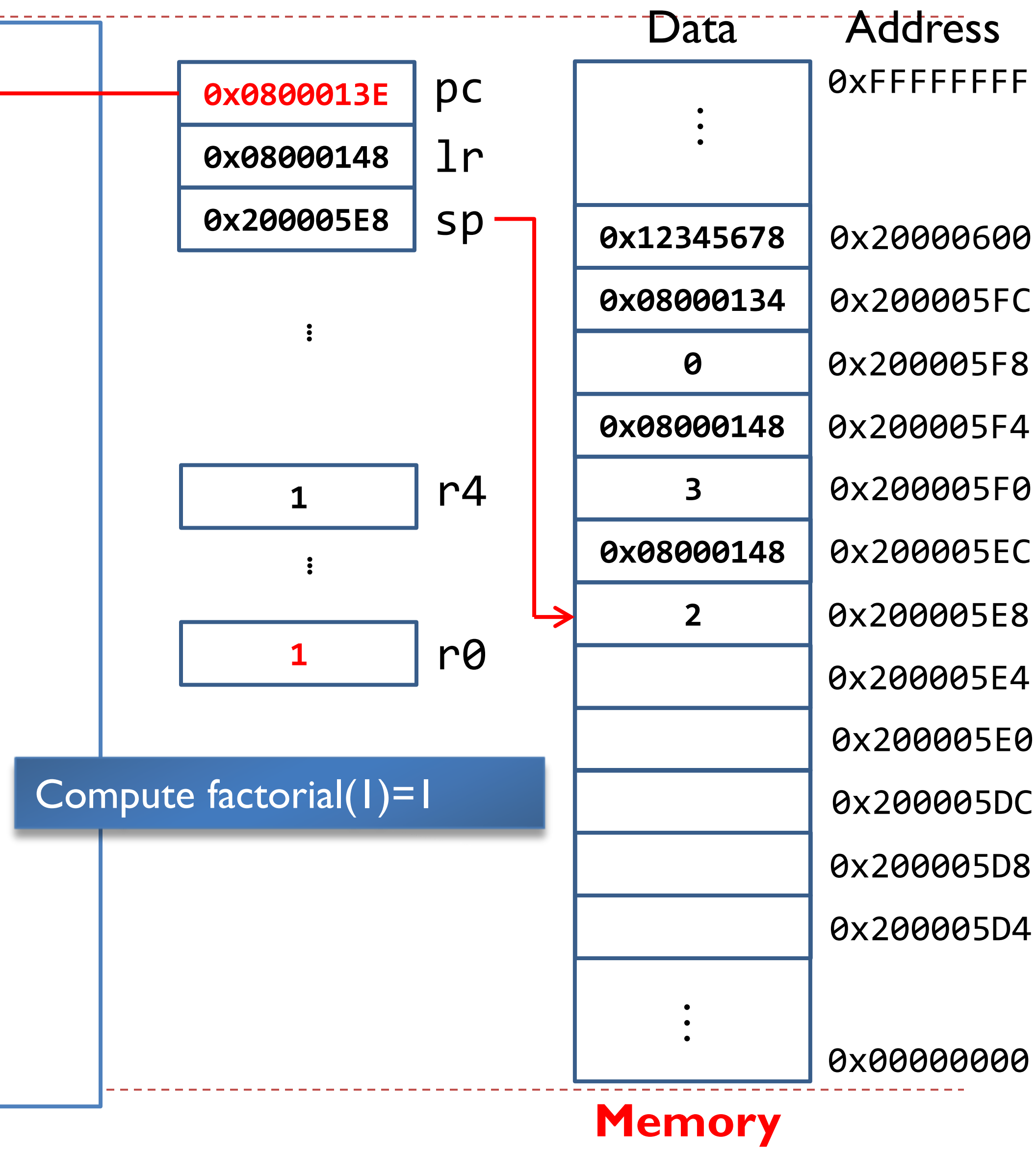
pc	0x08000138
lr	0x08000148
sp	0x200005E8
r4	1
r0	1

Memory

Data	Address
⋮	0xFFFFFFFF
0x12345678	0x20000600
0x08000134	0x200005FC
0	0x200005F8
0x08000148	0x200005F4
3	0x200005F0
0x08000148	0x200005EC
2	0x200005E8
	0x200005E4
	0x200005E0
	0x200005DC
	0x200005D8
	0x200005D4
⋮	0x00000000

Recursive Factorial in Assembly

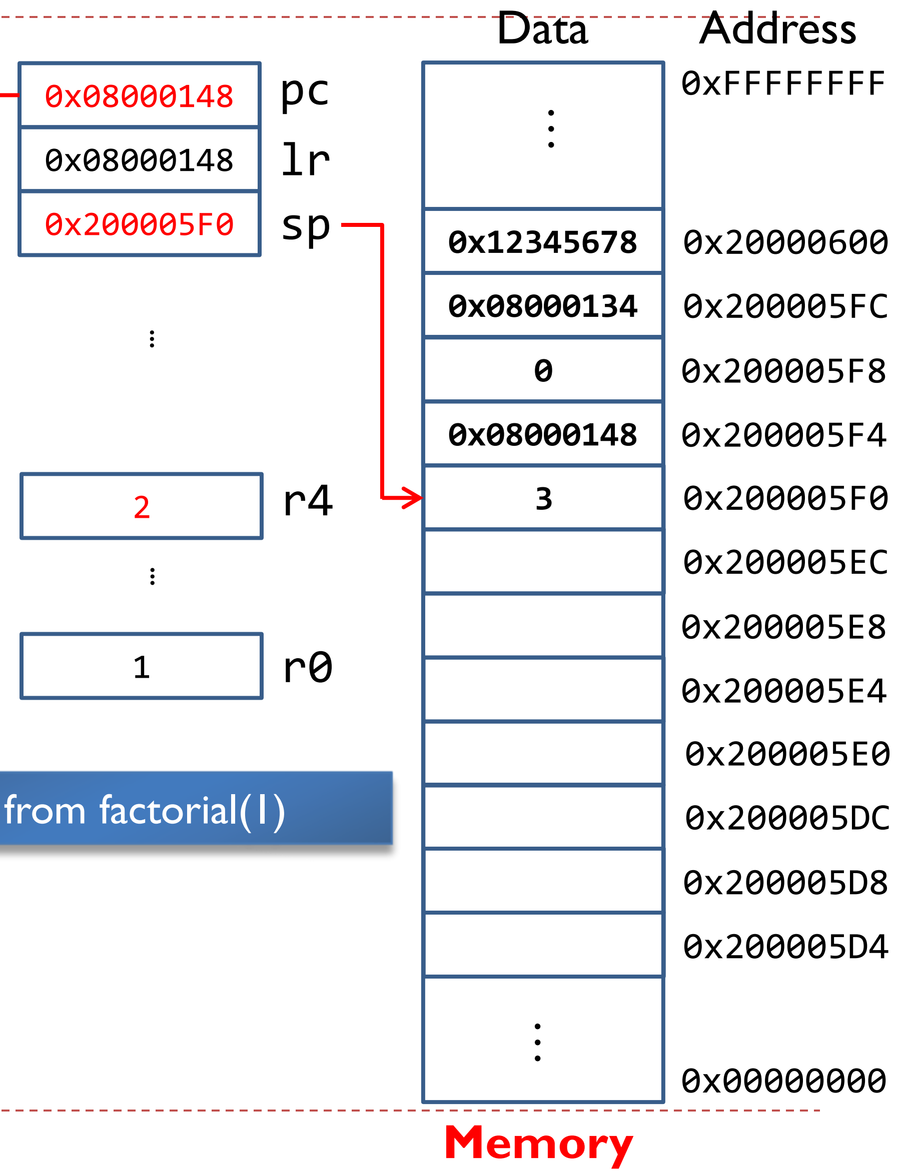
```
                 AREA main, CODE, READONLY
                 EXPORT   __main

                 ENTRY

            __main PROC
                 MOV r0, #0x03
0x08000130       BL  factorial
0x08000134  stop B   stop
                 ENDP

            factorial
0x08000136       PUSH {r4, lr}
0x08000138       MOV  r4, r0
0x0800013A       CMP  r4, #0x01
0x0800013C       BNE  NZ
0x0800013E       MOVS r0, #0x01
0x08000140  loop POP  {r4, pc}
0x08000142  NZ   SUBS r0, r4, #1
0x08000144       BL   factorial
0x08000148       MUL  r0, r4, r0
0x0800014C       B    loop

                 END
```



Recursive Factorial in Assembly

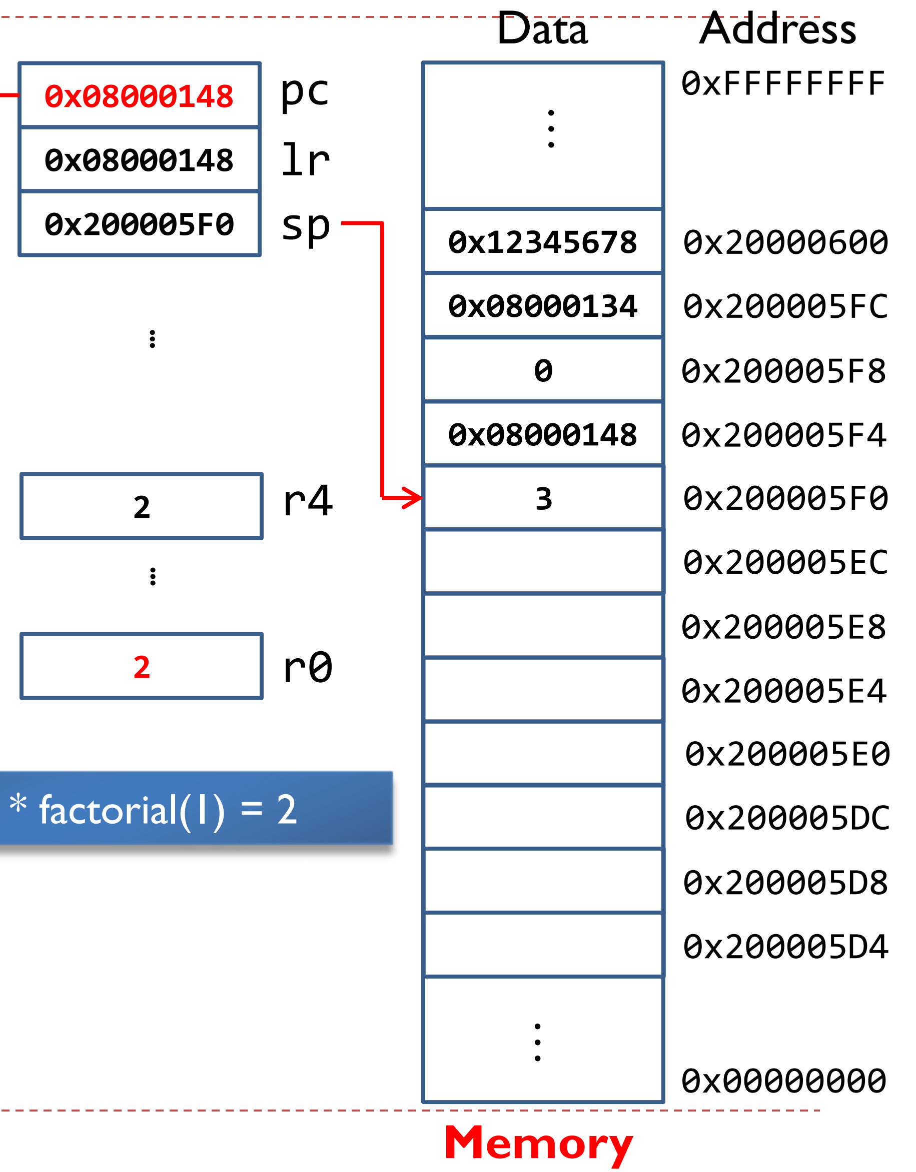
```
                 AREA main, CODE, READONLY
                 EXPORT  __main

                 ENTRY

            __main PROC
                 MOV r0, #0x03
0x08000130       BL  factorial
0x08000134  stop B   stop
                 ENDP

            factorial
0x08000136       PUSH {r4, lr}
0x08000138       MOV  r4, r0
0x0800013A       CMP  r4, #0x01
0x0800013C       BNE  NZ
0x0800013E       MOVS r0, #0x01
0x08000140  loop POP  {r4, pc}
0x08000142  NZ   SUBS r0, r4, #1
0x08000144       BL   factorial
0x08000148       MUL  r0, r4, r0
0x0800014C       B    loop

                 END
```

Return from factorial(1)



Recursive Factorial in Assembly

```
                AREA main, CODE, READONLY
                EXPORT  __main

                ENTRY

           __main PROC
                MOV r0, #0x03
0x08000130      BL  factorial
0x08000134 stop B   stop
                ENDP

           factorial
0x08000136      PUSH {r4, lr}
0x08000138      MOV  r4, r0
0x0800013A      CMP  r4, #0x01
0x0800013C      BNE  NZ
0x0800013E      MOVS r0, #0x01
0x08000140 loop POP  {r4, pc}
0x08000142 NZ   SUBS r0, r4, #1
0x08000144      BL   factorial
0x08000148      MUL  r0, r4, r0
0x0800014C      B    loop

                END
```

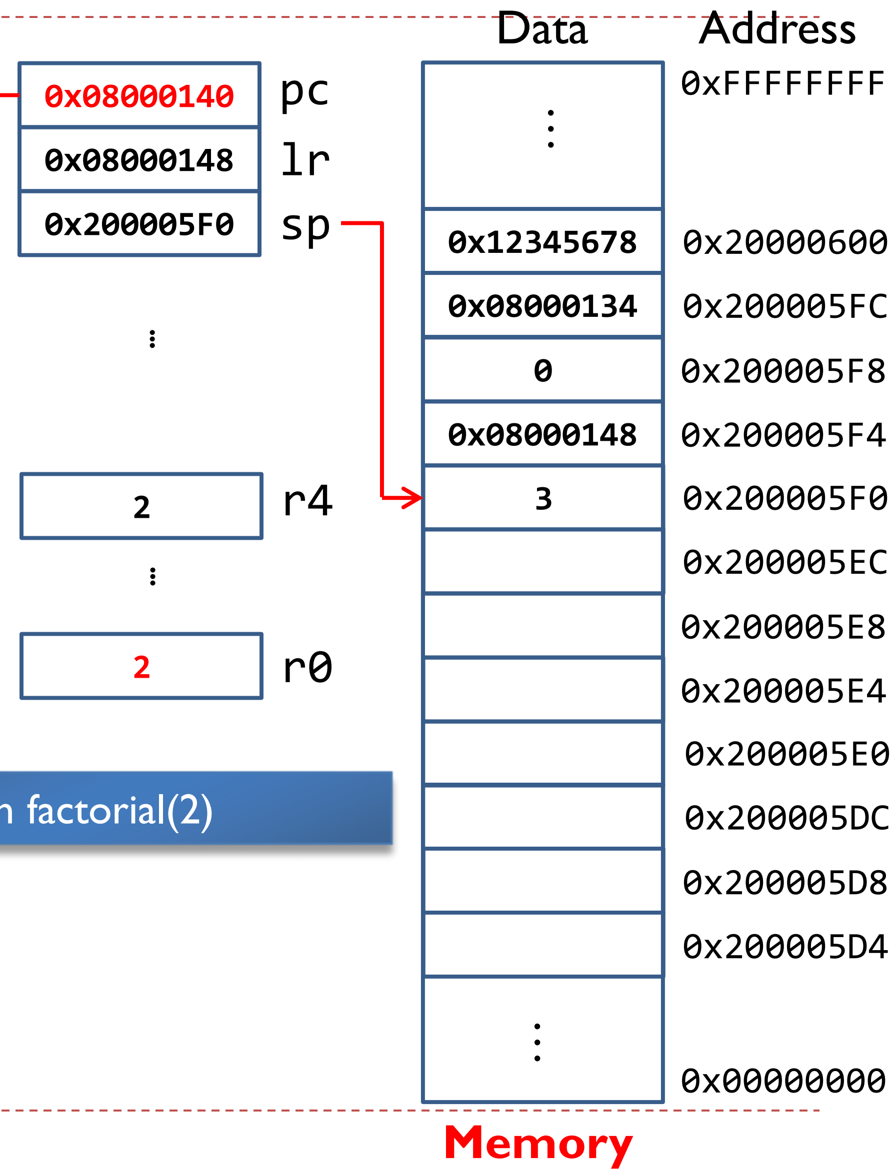


Compute 2 * factorial(1) = 2

Recursive Factorial in Assembly

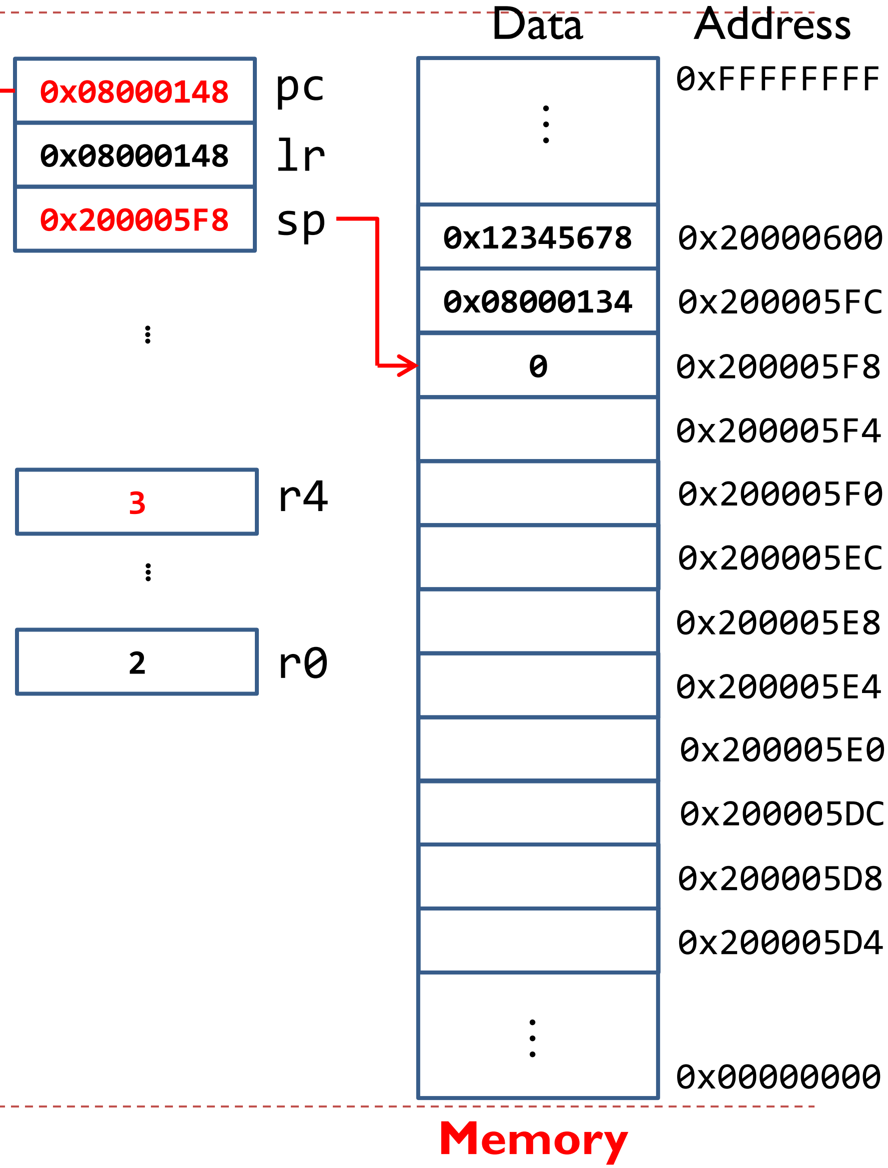
```
                  AREA main, CODE, READONLY
                  EXPORT   __main

                  ENTRY

            __main PROC
                  MOV r0, #0x03
0x08000130        BL   factorial
0x08000134  stop  B    stop
                  ENDP

            factorial
0x08000136        PUSH {r4, lr}
0x08000138        MOV  r4, r0
0x0800013A        CMP  r4, #0x01
0x0800013C        BNE  NZ
0x0800013E        MOVS r0, #0x01
0x08000140  loop  POP  {r4, pc}
0x08000142  NZ    SUBS r0, r4, #1
0x08000144        BL   factorial
0x08000148        MUL  r0, r4, r0
0x0800014C        B    loop

                  END
```

Return from factorial(2)



Recursive Factorial in Assembly

```
                    AREA main, CODE, READONLY
                    EXPORT   __main

                    ENTRY

           __main PROC
                    MOV r0, #0x03
0x08000130          BL  factorial
0x08000134 stop B   stop
                    ENDP

           factorial
0x08000136          PUSH {r4, lr}
0x08000138          MOV  r4, r0
0x0800013A          CMP  r4, #0x01
0x0800013C          BNE  NZ
0x0800013E          MOVS r0, #0x01
0x08000140 loop POP  {r4, pc}
0x08000142 NZ       SUBS r0, r4, #1
0x08000144          BL   factorial
0x08000148          MUL  r0, r4, r0
0x0800014C          B    loop

                    END
```



Recursive Factorial in Assembly

```
                 AREA main, CODE, READONLY
                 EXPORT   __main

                 ENTRY

            __main PROC
                 MOV r0, #0x03
0x08000130       BL   factorial
0x08000134  stop B    stop
                 ENDP

            factorial
0x08000136       PUSH {r4, lr}
0x08000138       MOV   r4, r0
0x0800013A       CMP   r4, #0x01
0x0800013C       BNE   NZ
0x0800013E       MOVS  r0, #0x01
0x08000140  loop POP   {r4, pc}
0x08000142  NZ   SUBS  r0, r4, #1
0x08000144       BL    factorial
0x08000148       MUL   r0, r4, r0
0x0800014C       B     loop

                 END
```

pc	0x08000148
lr	0x08000148
sp	0x200005F8
⋮	
r4	3
⋮	
r0	6

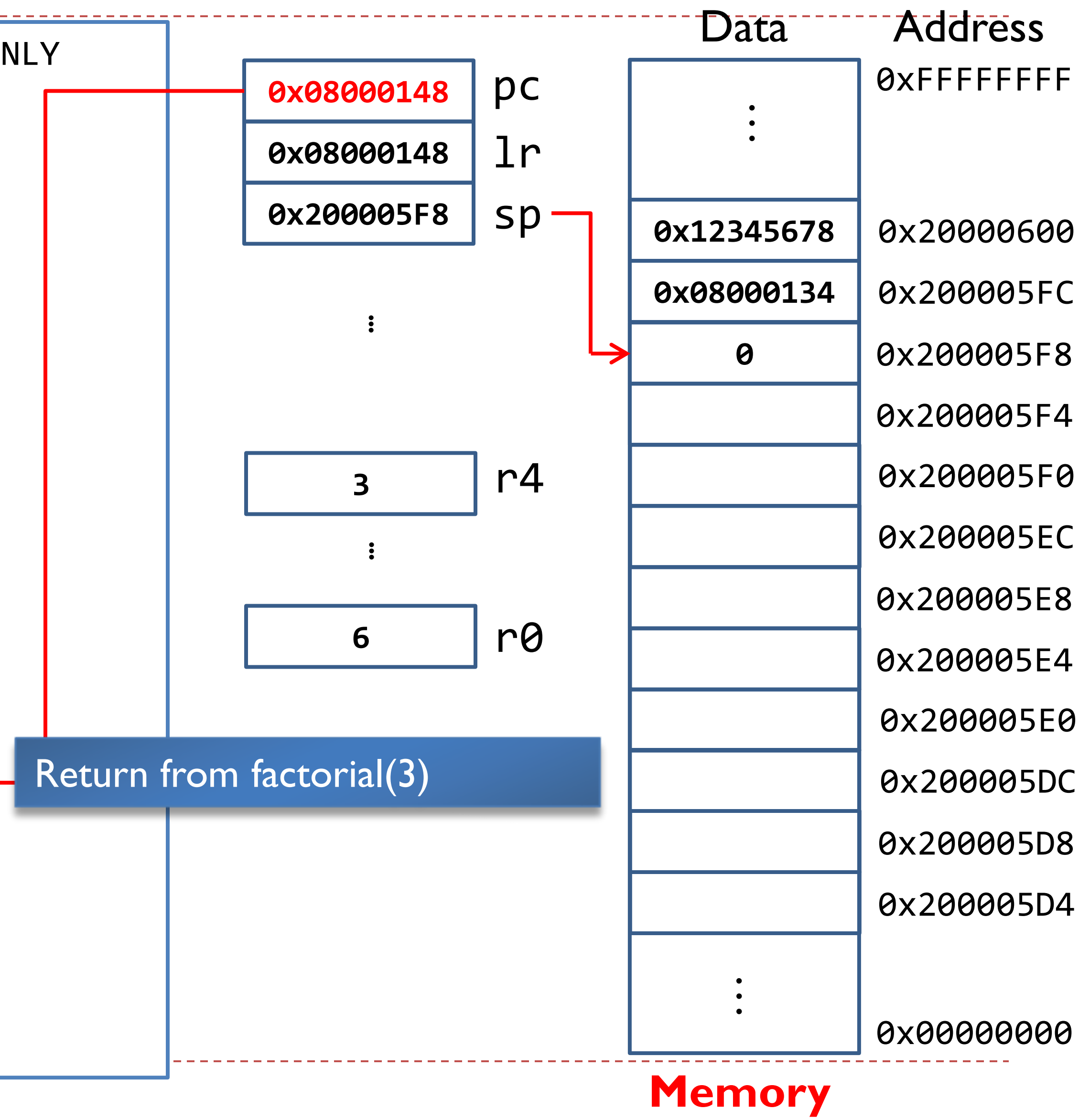
Compute 3 * factorial(2) = 6

Data	Address
⋮	0xFFFFFFFF
0x12345678	0x20000600
0x08000134	0x200005FC
0	0x200005F8
	0x200005F4
	0x200005F0
	0x200005EC
	0x200005E8
	0x200005E4
	0x200005E0
	0x200005DC
	0x200005D8
	0x200005D4
⋮	0x00000000

Memory

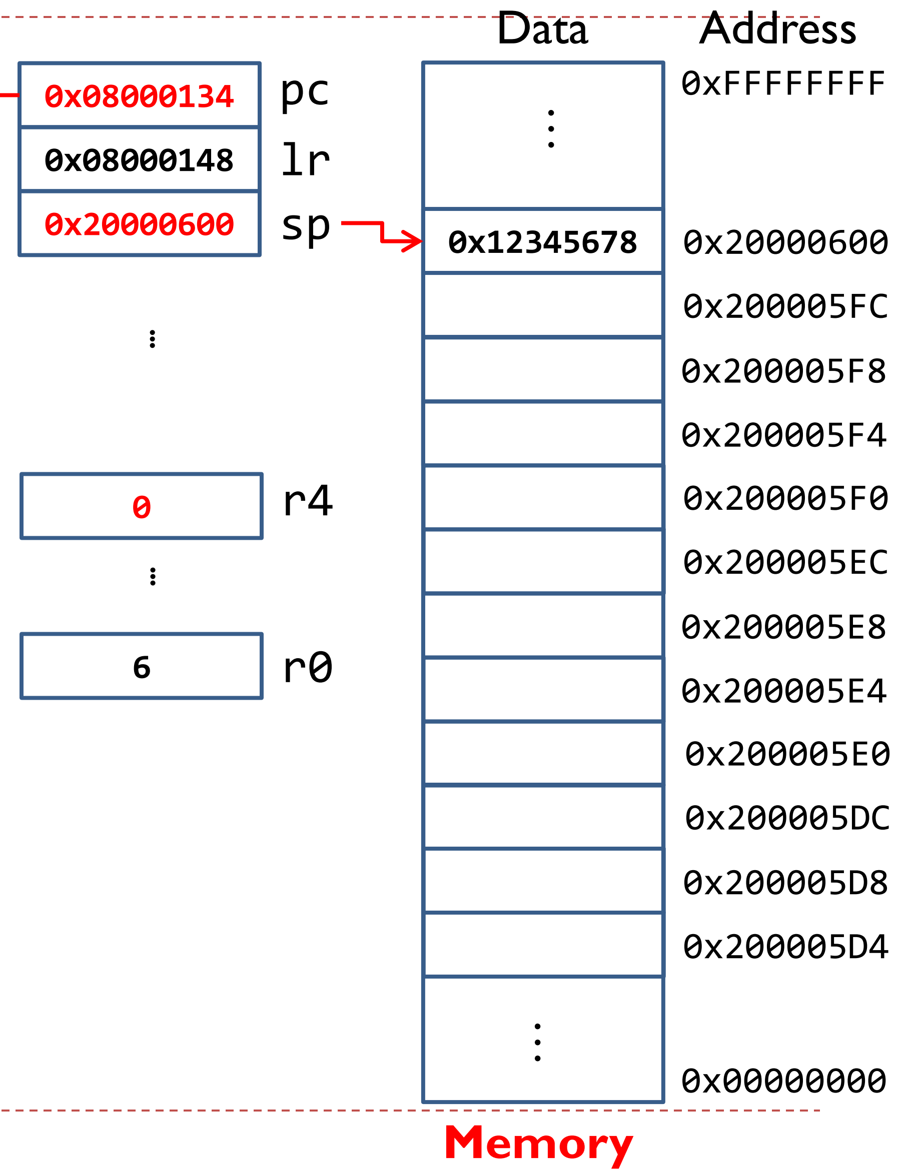
Recursive Factorial in Assembly

```
                AREA main, CODE, READONLY
                EXPORT  __main

                ENTRY

           __main PROC
                MOV r0, #0x03
0x08000130      BL  factorial
0x08000134 stop B   stop
                ENDP

           factorial
0x08000136      PUSH {r4, lr}
0x08000138      MOV  r4, r0
0x0800013A      CMP  r4, #0x01
0x0800013C      BNE  NZ
0x0800013E      MOVS r0, #0x01
0x08000140 loop POP  {r4, pc}
0x08000142 NZ   SUBS r0, r4, #1
0x08000144      BL   factorial
0x08000148      MUL  r0, r4, r0
0x0800014C      B    loop

                END
```



Recursive Factorial in Assembly

```
                  AREA main, CODE, READONLY
                  EXPORT   __main

                  ENTRY

           __main PROC
                  MOV r0, #0x03
0x08000130        BL   factorial
0x08000134 stop B      stop
                  ENDP

           factorial
0x08000136        PUSH {r4, lr}
0x08000138        MOV   r4, r0
0x0800013A        CMP   r4, #0x01
0x0800013C        BNE   NZ
0x0800013E        MOVS  r0, #0x01
0x08000140 loop   POP   {r4, pc}
0x08000142 NZ     SUBS  r0, r4, #1
0x08000144        BL    factorial
0x08000148        MUL   r0, r4, r0
0x0800014C        B     loop

                  END
```



Recursive Factorial in Assembly

```
                AREA main, CODE, READONLY
                EXPORT   __main

                ENTRY

           __main PROC
                MOV r0, #0x03
0x08000130      BL   factorial
0x08000134 stop B    stop
                ENDP

           factorial
0x08000136      PUSH {r4, lr}
0x08000138      MOV  r4, r0
0x0800013A      CMP  r4, #0x01
0x0800013C      BNE  NZ
0x0800013E      MOVS r0, #0x01
0x08000140 loop POP  {r4, pc}
0x08000142 NZ   SUBS r0, r4, #1
0x08000144      BL   factorial
0x08000148      MUL  r0, r4, r0
0x0800014C      B    loop

                END
```