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

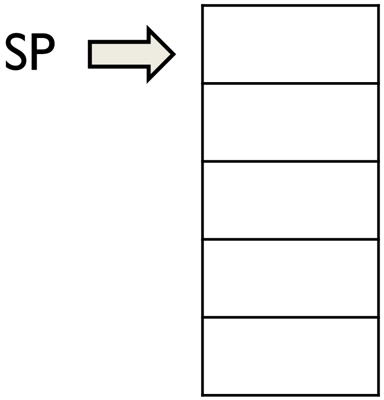
Chapter 8
Subroutines
Exercises

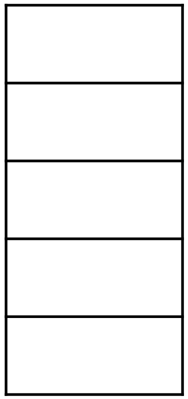
Z. Gu

Fall 2025

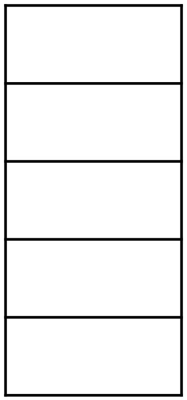
Stack

- Initially, let r0=0, r1=1, r2=2.
- a) Execute PUSH {r1,r2}. Draw stack.
- b) Execute POP {r0,r1}. Draw stack.





After PUSH {r1,r2}



After POP {r0,r1},
r0=?, r1=?

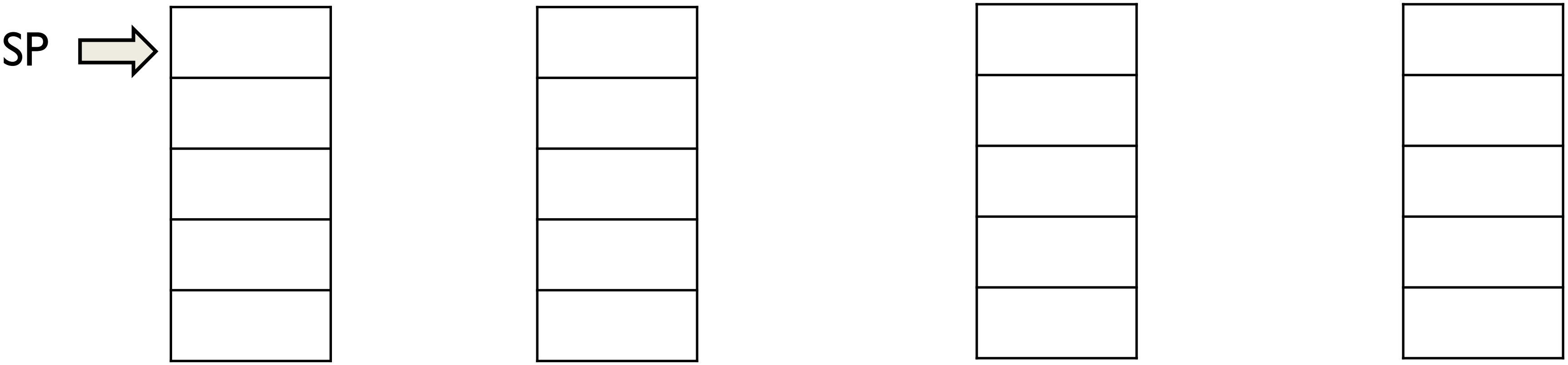
Stack

- Initially, let r0=0, r1=1, r2=2, r3=3
- Execute

  PUSH {r1,r2}

  PUSH {r3,r0}

  POP {r0-r3} (same as POP {r0, r1, r2, r3})
- Draw stack after each instruction. What is in registers after execution?



After PUSH {r1,r2}

After PUSH {r3,r0}

After POP {r0-r3}
r0=?, r1=?, r2=?, r3=?

What’s wrong? Passing arguments and Returning Value

```
uint32_t sum(uint8_t a8, uint8_t b8, uint16_t c16, uint16_t d16,
uint32_t e32);
```

```
s = sum(1, 2, 3, 4, 5);
```

Caller

```
   MOV r0, #5 ; e32
   MOV r0, #1 ; a8
   MOV r1, #2 ; b8
   MOV r2, #3 ; c16
   MOV r3, #4 ; d16
   BL  sum
   ...
```

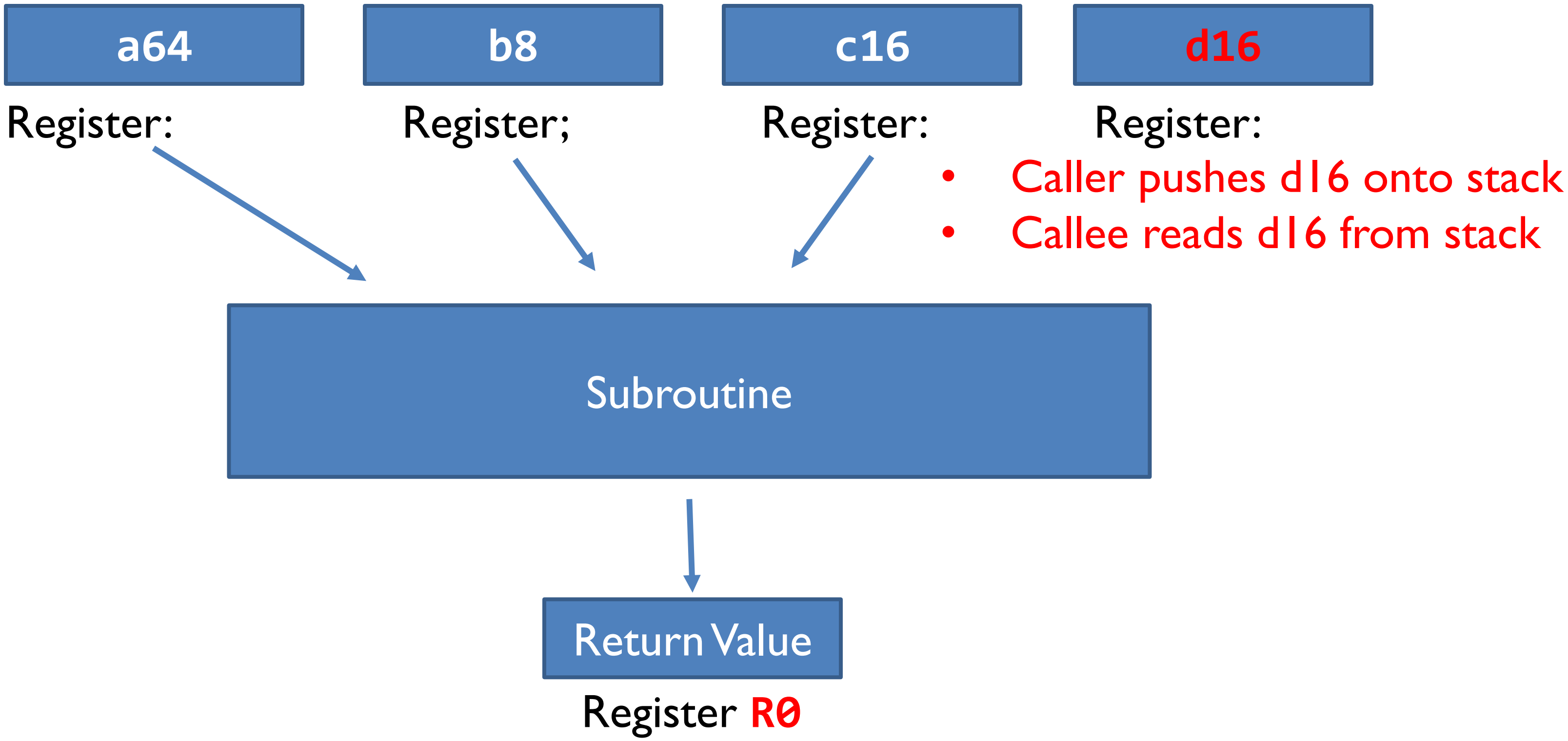
Callee

```
sum PROC
   ADD r0, r0, r1    ; a8 + b8
   ADD r0, r0, r2    ; add c16
   ADD r0, r0, r3    ; add d16
   ADD r0, r0, r1    ; add e32
   BX  LR
   ENDP
```

Passing arguments and Returning Value

```
uint64_t sum(uint64_t a64, uint8_t b8, uint16_t c16, uint16_t d16);
```

- Fill in register names.



a64

Register:

b8

Register;

c16

Register:

d16

Register:

- Caller pushes d16 onto stack
- Callee reads d16 from stack

Subroutine

Return Value

Register **R0**

What is Wrong?

Caller Program

```
Extern int32_t sum3(int32_t a1, int32_t a2, int32_t a3);

int main(void){
int32_t s
...
s = sum3(-1, -2, -3) + sum3(4, 5, 6);
...
```

Callee Program

```
sum3 PROC
EXPORT sum3
; r3 = sum
ADD r3, r0, r1 ; sum = a1 + a2
ADD r3, r0, r2 ; sum += a3
MOV r1, r3
BX pc
ENDP
```

toLower

Caller Program

```
#include <stdio.h>

extern int mystery(int); /* mystery assembler
routine */

int main(void)
{
    static const char str[] = "Hello, World!";

    const int len = sizeof(str)/sizeof(str[0]);
    char      newstr[len];
    int       i;

    for (i = 0; i < len; i++)
        newstr[i] = toLower (str[i]);

    printf("%s\n", newstr);

    return 0;
}
```

- Consider the following C program that converts all ASCII letters to lower case. Write the toLower function in ARMv7 assembly code.

Callee Program

```
int toLower (int c)
{
  if (c >= 'A' && c <= 'Z')
     c += 'a' - 'A';

  return c;
}
```

Callee Program Assembly

```
    .text
    .global toLower
toLower:
```

If Then Else

- Translate the following program into ARMv7 assembly.

C Program	Assembly Program

int foo(int x, int y) {
  if (x+y < 0)
  return 0;
else
  return 1;
}
	@ int foo(int x, int y) - returns 0 if (x+y) < 0, else 1
	@ x in r0, y in r1, return in r0
	foo:
	  …
	  BX lr

Factorial

- Fill in the blanks (TODO) for the assembly programs for calculating the factorial of a number, corresponding to the following C programs. One recursive version, one iterative version.

```
//Iterative algorithms for Factorial
#include <stdint.h>

uint32_t fact_iter(uint32_t n) {
    uint32_t acc = 1;
    if (n <= 1) {
        return 1;
    }
    while (n > 1) {
        acc *= n;
        n -= 1;
    }
    return acc;
}
```

```
//Recursive algorithms for Factorial
#include <stdint.h>

uint32_t fact_rec(uint32_t n) {
    if (n <= 1) {
        return 1;
    }
    return n * fact_rec(n - 1);
}
```

Factorial

```
% uint32_t fact_iter(uint32_t n);
% r0 = n, returns r0 = n!
    .global fact_iter
fact_iter:
    PUSH    {r4, lr}        % save callee-saved we'll use and return addr
    MOV     r1, r0          % r1 = n (loop counter)
    MOV     r0, #1          % r0 = acc = 1
    CMP     r1, #1
    BLS     .Ldone_iter  % if n <= 1, return 1

.Lloop_iter:
    % TODO
.Ldone_iter:
    POP     {r4, lr}
    BX      lr
```

```
% uint32_t factorial(uint32_t n);
% r0: n
% returns r0: n!

factorial:
    CMP     r0, #1          % if (n <= 1) ...
    BLE     base_case       %    ... return 1

    PUSH    {lr}            % save return address for this frame
    PUSH    {r0}            % save current n on stack (we'll need it after the recursive call)

    SUB     r0, r0, #1      % r0 = n - 1 (argument for recursive call)
    BL      factorial       % r0 = factorial(n - 1)

    POP     {r1}            % r1 = saved n (restore caller's n)
    MUL     r0, r0, r1      % r0 = factorial(n - 1) * n

    POP     {lr}            % restore return address
    BX      lr              % return with result in r0

base_case:
    % TODO
```

What is wrong?

```
int16_t sum_of_array(int16_t *pArray){
    uint32_t i;
    int32_t sum = 0;
    for(i=0; i<64; i++) // array size = 64
        sum += pArray[i];
    return (int16_t) sum;
}
```

```
sum_of_array PROC
      MOV   r2, #0   ; loop index
      MOV   r3, #0   ; sum
      B     check
loop  LDRSH r1, [r0], #2
      ADD   r3, r3, r1   ; sum += pArray[i]
      ADD   r2, r2, #1   ; i++
check CMP   r2, #64
      BLO   loop         ; branch if unsigned LOwer
      MOV   r0, r3       ; return result in r0
      BX    lr
      ENDP
```

Program Understanding

- Write out the sequence of values of r0 and r7 after running this program.

```
start:
        mov     r0, #1

main:
        add     r0, r0, #1
        cmp     r0, #5
        bne     skip
        bl      call

skip:
        b       main

call:
        add     r7, r7, #255
        mov     r0, #1
        bx      lr
```