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

Chapter 8
Subroutines
Exercises

Z. Gu

Fall 2025

Review

Stack

PUSH {*Rd*}

- SP = SP-4 ⟶ descending stack
- (*SP) = *Rd* ⟶ full stack

Push multiple registers

They are equivalent.

```
PUSH {r6, r7, r8}   ⟺   PUSH {r8, r7, r6}   ⟺   PUSH {r8}
                                                PUSH {r7}
                                                PUSH {r6}
```

- SP is decremented before PUSH (pre-decrement), and incremented after POP (post-increment).
- The order in which registers listed in the register list does not matter.
- When pushing multiple registers, these registers are automatically sorted by name and the lowest-numbered register is stored to the lowest memory address, *i.e.* is stored last.

Review

Stack

POP {*Rd*}

- *Rd* = (*SP) ⟶ full stack
- SP = SP + 4 ⟶ Stack shrinks

Pop multiple registers

They are equivalent.

`POP {r6, r7, r8}` ⟺ `POP {r8, r7, r6}` ⟺

```
POP {r6}
POP {r7}
POP {r8}
```

- SP is decremented before PUSH (pre-decrement), and incremented after POP (post-increment).
- The order in which registers listed in the register list does not matter.
- When popping multiple registers, these registers are automatically sorted by name and the lowest-numbered register is loaded from the lowest memory address, *i.e.* is loaded first.

Review

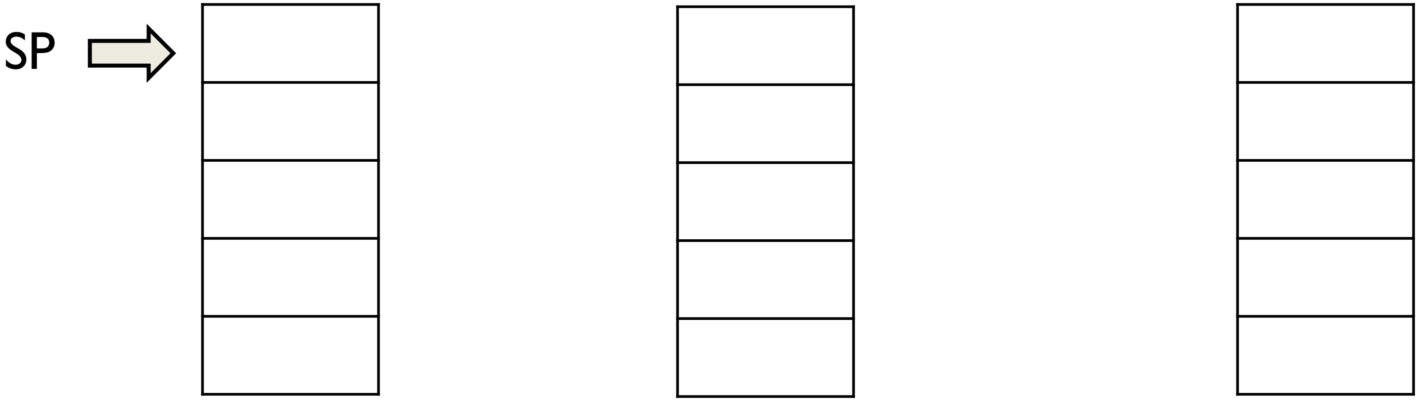
Summary: Condition Codes

Suffix	Description	Flags tested
EQ	EQual	Z=1
NE	Not Equal	Z=0
CS/HS	Unsigned Higher or Same	C=1
CC/LO	Unsigned LOwer	C=0
MI	MInus (Negative)	N=1
PL	PLus (Positive or Zero)	N=0
VS	oVerflow Set	V=1
VC	oVerflow Cleared	V=0
HI	Unsigned HIgher	C=1 & Z=0
LS	Unsigned Lower or Same	C=0 or Z=1
GE	Signed Greater or Equal	N=V
LT	Signed Less Than	N!=V
GT	Signed Greater Than	Z=0 & N=V
LE	Signed Less than or Equal	Z=1 or N!=V
AL	ALways	

Note AL is the default and does not need to be specified

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=?

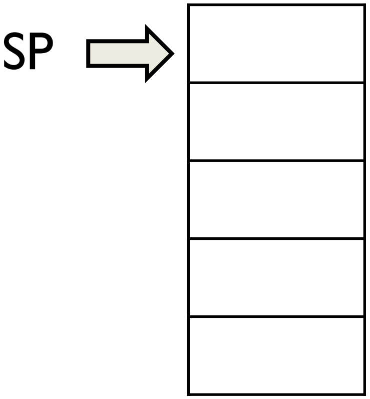
Stack

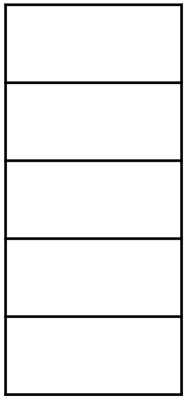
- Initially, let r0=0, r1=1, r2=2, r3=3
- Execute

  PUSH {r1,r2}

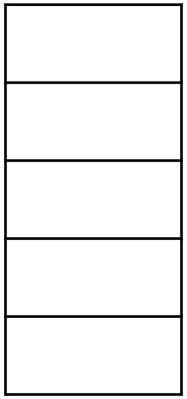
  PUSH {r3,r0}

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

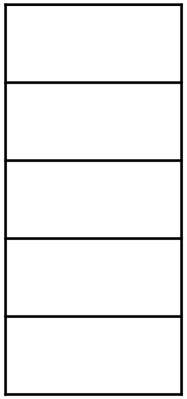




After PUSH {r1,r2}



After PUSH {r3,r0}



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

What 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.
  - int foo(int x, int y) {
  - if ((x+y) < 0)
  - return 0;
  - else return 1;
  - }
- ANS:
  - @ 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

- Write an assembly program to calculate the factorial of a number, corresponding to the following C programs. One recursive version, one iterative version. (In the exams, I may provide most of the code and let you fill in the blanks.)

```
//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);
}
```