School of Engineering
InES Institute of Embedded Systems

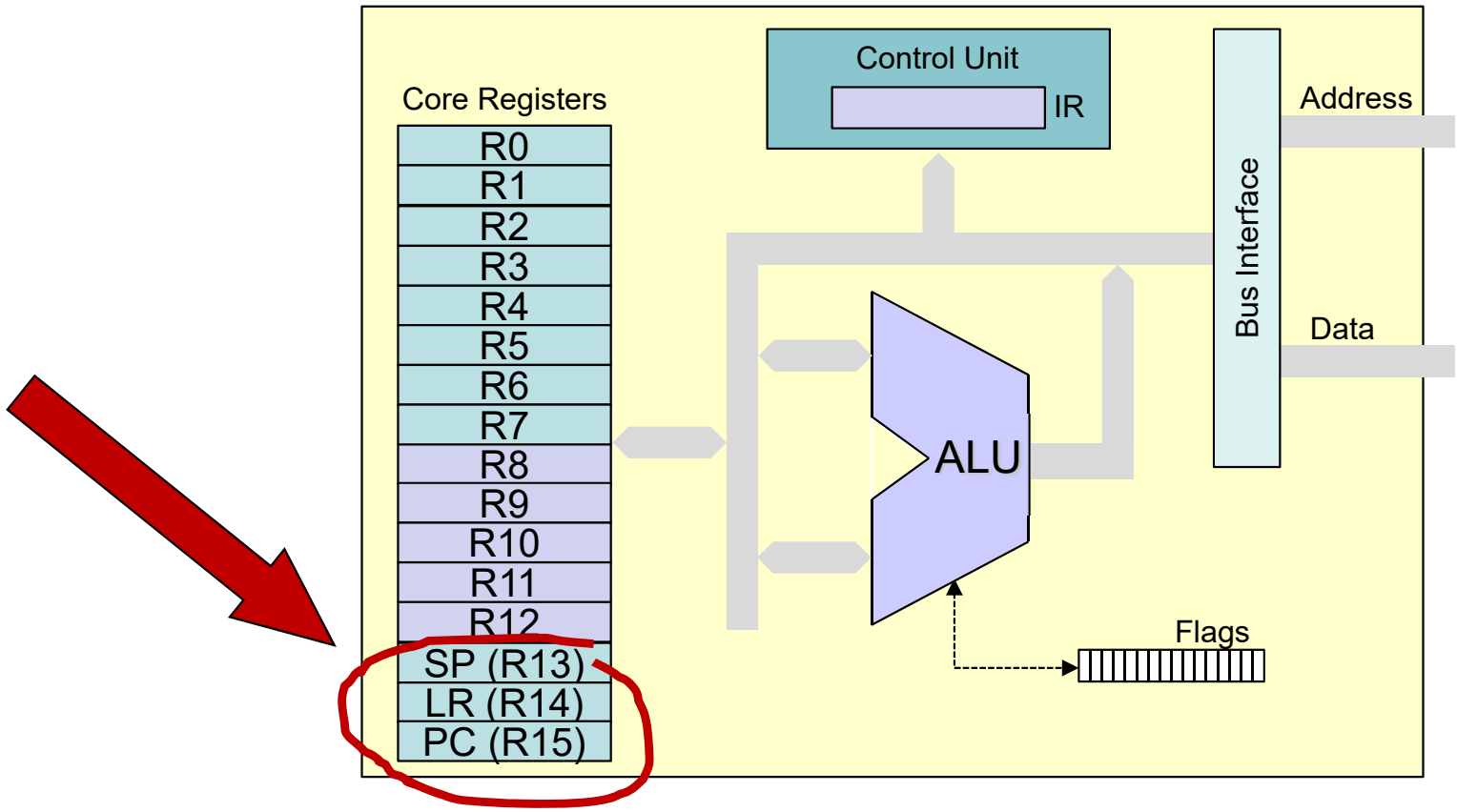
Subroutines and Stack

Computer Engineering 1

CT Team: A. Gieriet, J. Gruber, B. Koch, M. Loeser, M. Meli,
M. Rosenthal, M. Ostertag, A. Rüst, J. Scheier, T. Welti

Motivation

- **Do you remember?**



Motivation

School of Engineering
InES Institute of Embedded Systems

main program

subroutine

```
...
B   proc_a   ; 1. call
...          ; next instruction 1
...
...
...
B   proc_a   ; 2. call
...          ; next instruction 2
...
...
```

```
proc_a
        ...
        ...
        B   ???
```

Agenda

School of Engineering
InES Institute of Embedded Systems

- **Terminology**
- **Subroutine Call and Return**
- **Nested Subroutine Calls**
- **Stack**
- **ARM: PUSH and POP**
- **Nested Subroutines (revisited)**
- **Instructions using SP**
- **Assembler Directives**

Learning Objectives

School of Engineering
InES Institute of Embedded Systems

At the end of this lesson you will be able

- to explain and discuss the term subroutine
- to comprehend and explain how a subroutine call and return are implemented on ARM Cortex-M
- to implement (nested) subroutines in assembly
- to explain how a processor stack works
- to determine the content of the stack for a given assembly program with nested subroutine calls

Terminology

School of Engineering
InES Institute of Embedded Systems

- Subroutine / Procedure / Function / Method
  - Sequence of instructions to solve a subtask
  - Called by "name"
  - Interface and functionality known
  - Internal design and implementation are hidden
    → information hiding
  - Can be called from miscellaneous places in the program
- Why Subroutines?
  - Basic element of structured programing
  - Reuse of the same implementation → less mistakes
  - Simplifies verification and maintenance
  - Requires less memory
    - only one instance for several calls

Terminology

School of Engineering
InES Institute of Embedded Systems

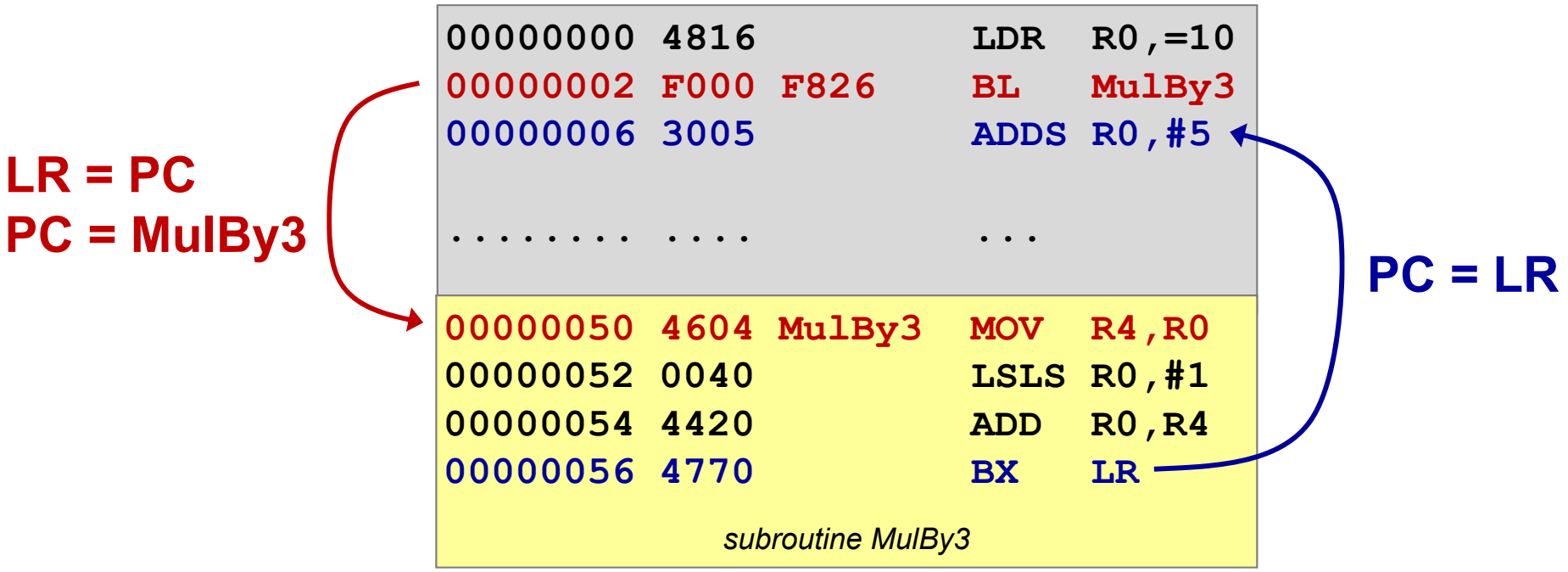
- **Terms used by ARM**
  - Routine, subroutine
    - A fragment of program to which control can be transferred that, on completing its task, returns control to its caller at an instruction following the call. *Routine* is used for clarity where there are nested calls: a routine is the *caller* and a subroutine is the *callee*.
  - Procedure
    - A routine that returns no result value.
  - Function
    - A routine that returns a result value.

source: Procedure Call Standard for the ARM Architecture
ARM IHI 0042E, 30th November 2012

Subroutine Call and Return

School of Engineering
InES Institute of Embedded Systems

- **Change of control flow**
  - Call Save PC to Link Register (LR)
  - Return Restore PC from LR



Subroutine Call and Return

School of Engineering
InES Institute of Embedded Systems

Structure of Subroutine

- Label with Name
  - e.g. MulBy3
- Return Statement
  - **BX LR**

```
00000050 4604 MulBy3   MOV   R4,R0
00000052 0040          LSLS  R0,#1
00000054 4420          ADD   R0,R4
00000056 4770          BX    LR
```

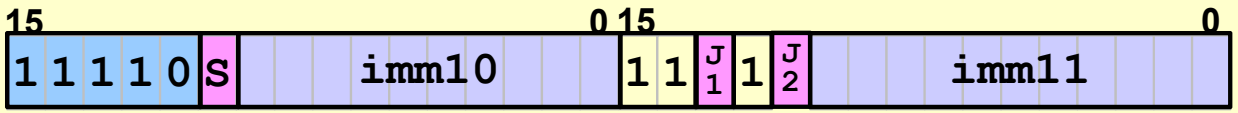
Subroutine Call and Return

School of Engineering
InES Institute of Embedded Systems

- **BL <label>**
  - Store current PC in LR
  - Branch to <label>
    - PC = PC +/- offset
    - offset range -16'777'216 to 16'777'214

Subroutine call through a label
- unconditional
- relative
- direct

```
BL <label>
```




```
I1 = NOT(J1 EOR S); I2 = NOT (J2 EOR S)
<imm> = S:I1:I2:imm10:imm11:0
LR = PC (LSB set to '1')
PC = PC + <imm>
```

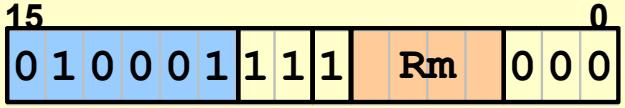
Subroutine Call and Return

School of Engineering
InES Institute of Embedded Systems

- **BLX (register)**
  - Store current PC in LR
  - Address of subroutine in register
  - Branch
    - PC = register
    - Branch address from 0 to 2^{32}

Subroutine call with address in register
- unconditional
- absolute
- indirect

```
BLX <Rm>
```

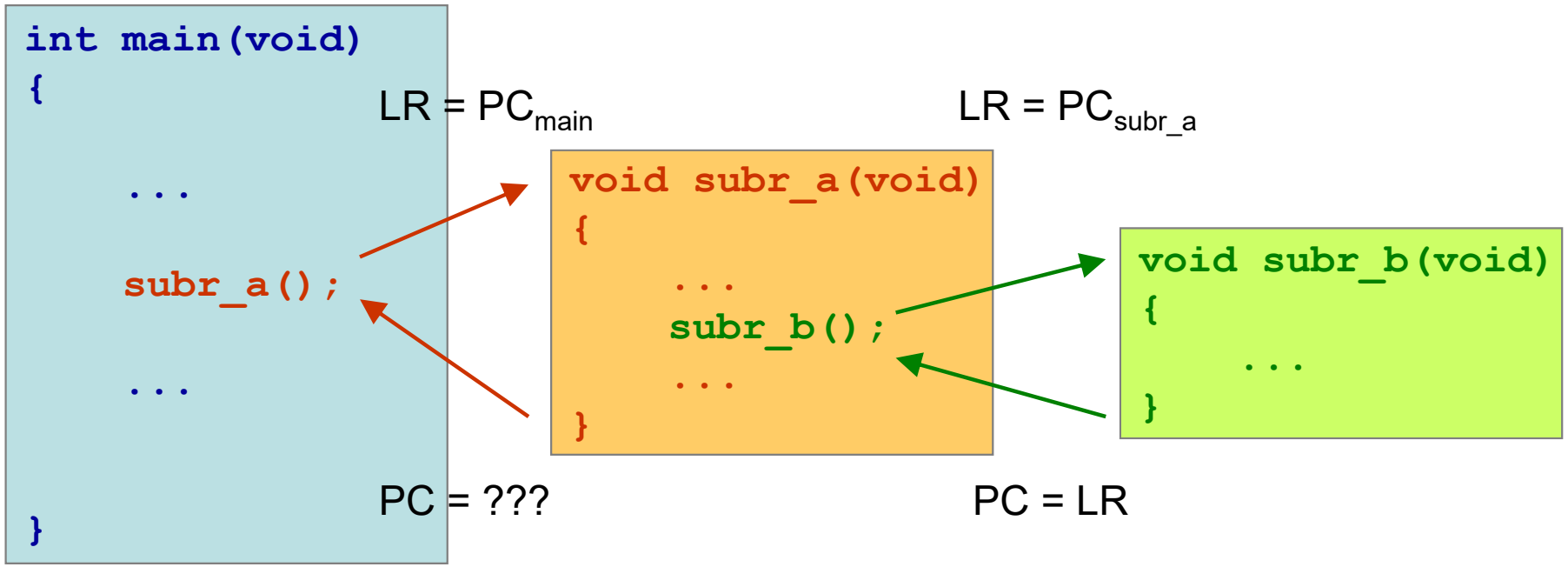



```
LR = PC - 2 (LSB set to '1')
PC = Rm
```

Nested Subroutine Calls

School of Engineering
InES Institute of Embedded Systems

- **Nested Subroutine (Function) Calls**
  - How do we do that with a single LR?

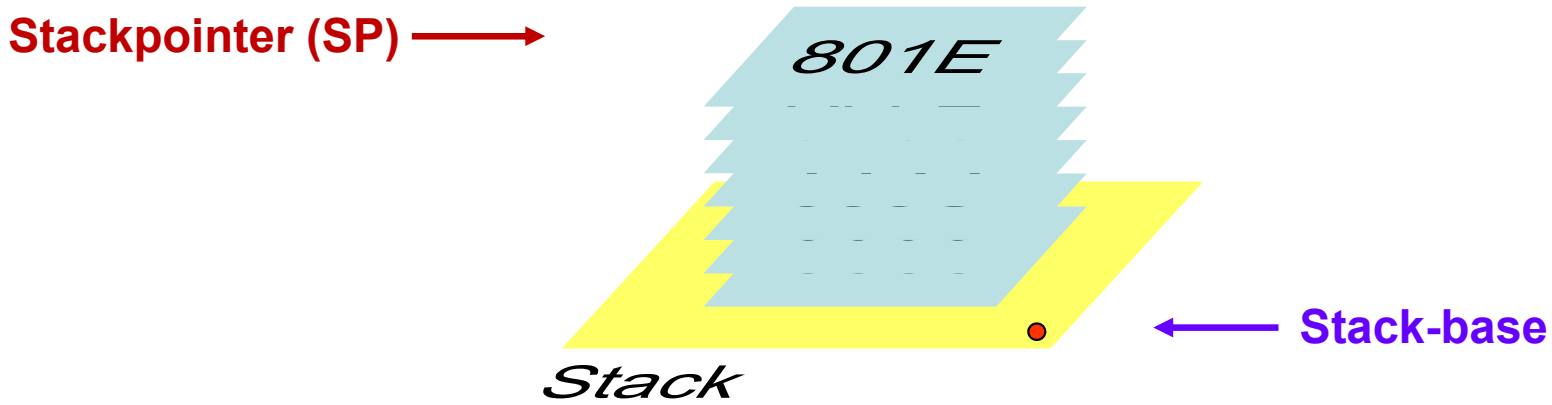


Animation

Stack

School of Engineering
InES Institute of Embedded Systems

- **Stack as Object**
  - Methods
    - **`PUSH()`** and **`POP()`**
  - Data
    - pushed (written) on top of the stack
    - popped (fetched, read) from the top of the stack → LIFO[1)]



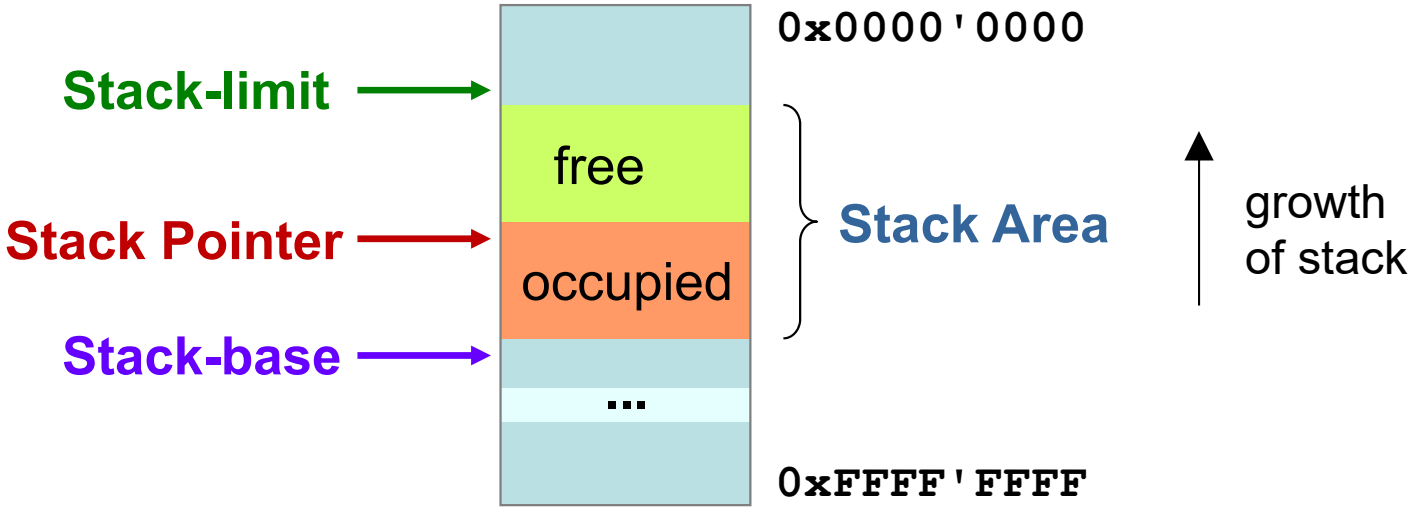
1) Last In First Out

Stack

School of Engineering
InES Institute of Embedded Systems

Implementation

Stack Area (Section)	Continuous area of RAM
Stack Pointer (SP)	R13 → points to last written data value
PUSH { … }	Decrement SP and store word(s)
POP { … }	Read word(s) and increment SP
Direction on ARM	"grows" from higher towards lower addresses → full-descending stack
Alignment	Stack operations are **word-aligned**

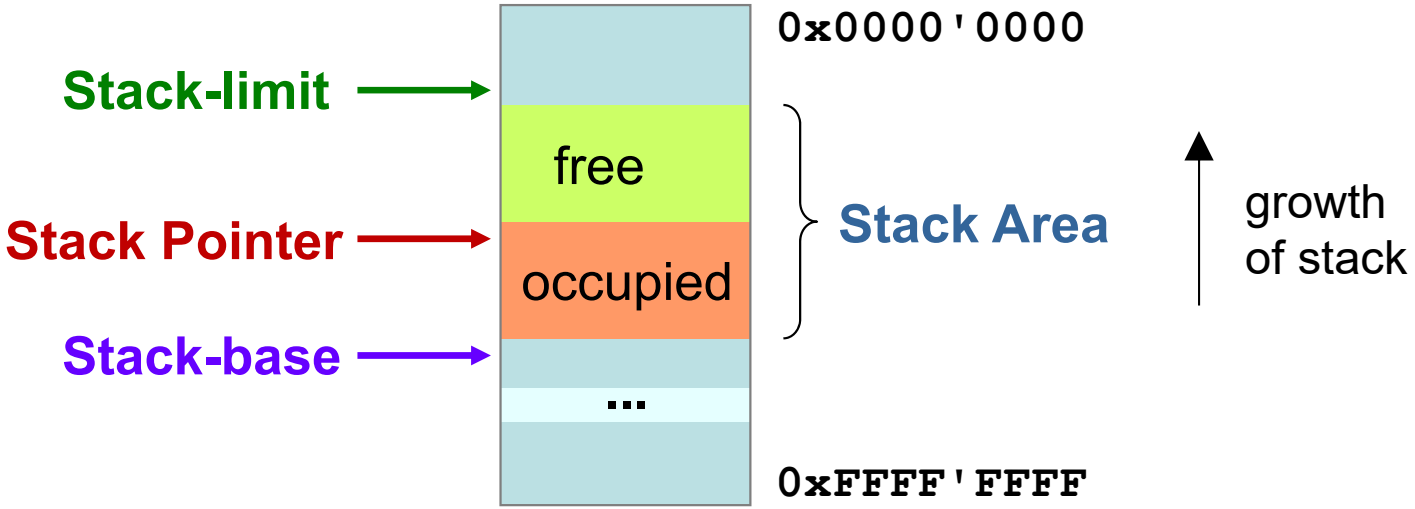


Stack

School of Engineering
InES Institute of Embedded Systems

Initialization

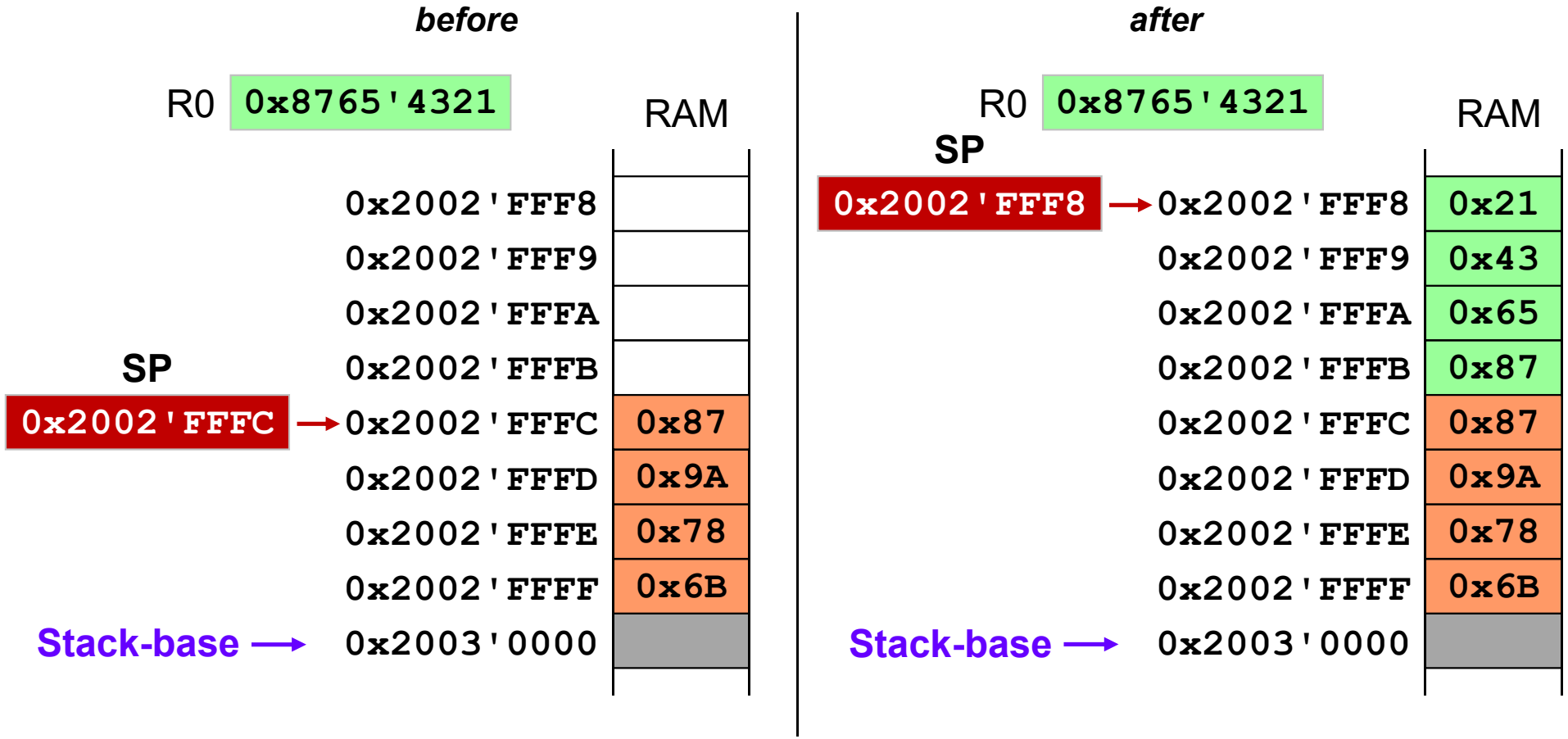
- Processor fetches initial value of SP (**Stack-base**) at reset
  - from address 0x0000'0000
- **Stack-base** is right above the stack area
  - SP is decremented before writing the first word



ARM: PUSH and POP

School of Engineering
InES Institute of Embedded Systems

- **Example: `PUSH {R0}`**

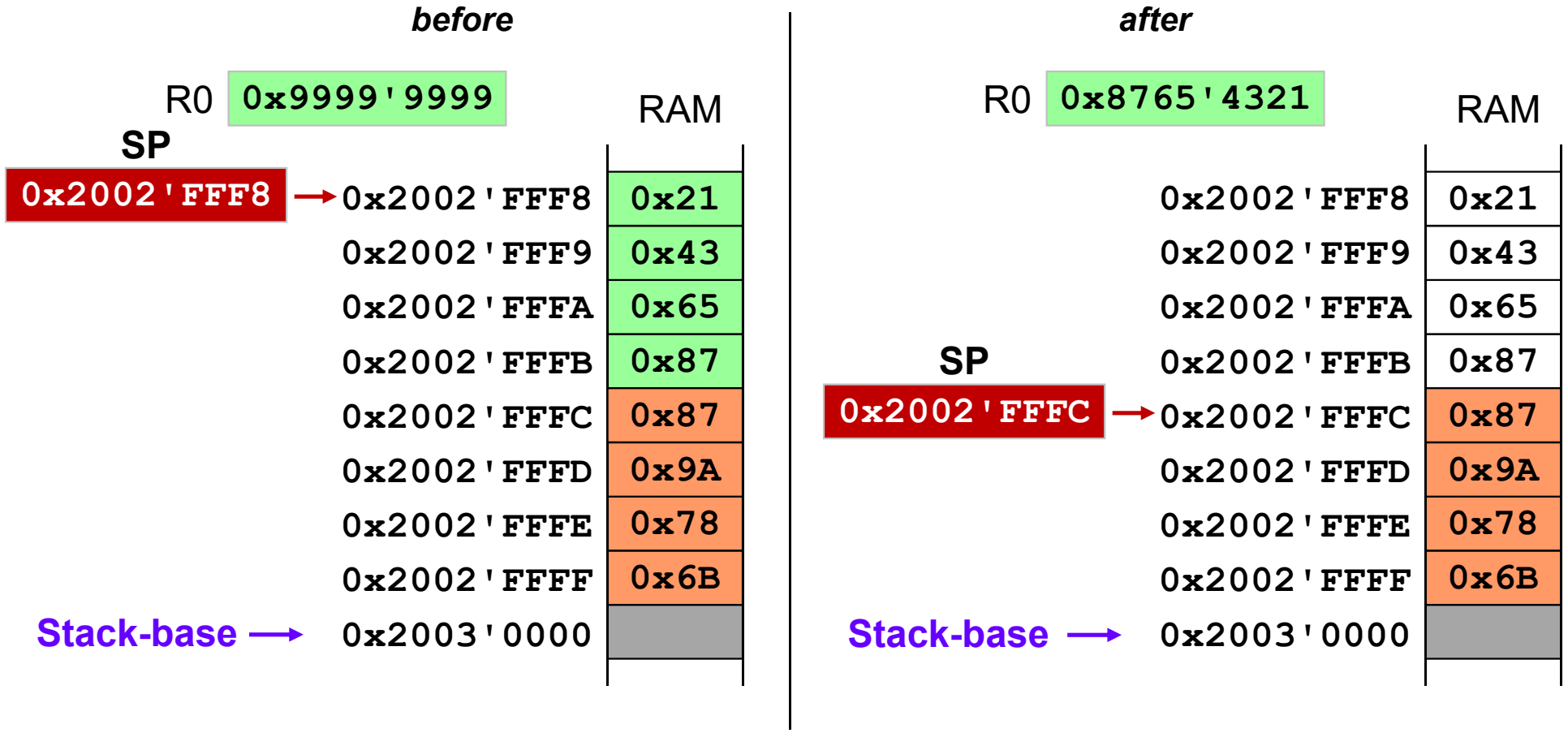


SP points to last value that has been written

ARM: PUSH and POP

School of Engineering
InES Institute of Embedded Systems

- **Example: `POP {R0}`**

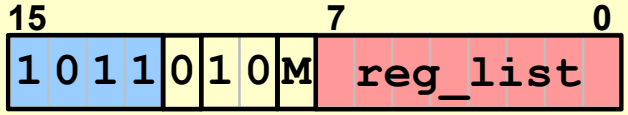


ARM: PUSH and POP

School of Engineering
InES Institute of Embedded Systems

- **PUSH**
  - **`registers`**
    - One or more registers to be stored
    - Low registers
      → **`reg_list`** = one bit per register
    - LR (R14) → M-bit
      → No other high registers
    - Lowest register stored first (lowest adress)

```
PUSH {registers}
```




```
addr = SP - 4*BitCount(M::reg_list)

for i = 0 to 7
    if reg_list<i> == '1' then
        Mem[addr,4] = R[i]
        addr = addr + 4

if (M == '1') then
     Mem[addr] = LR

SP = SP - 4*BitCount(M::reg_list)
```

```
00000000 B480      PUSH      {R7}
00000002 B43A      PUSH      {R1,R3,R4,R5}
00000004 B43A      PUSH      {R1,R3-R5}
00000006 B500      PUSH      {LR}
00000008 B580      PUSH      {R7,LR}
```

```
M::reg_list = 0x03A
 = 0'0011'1010b
```

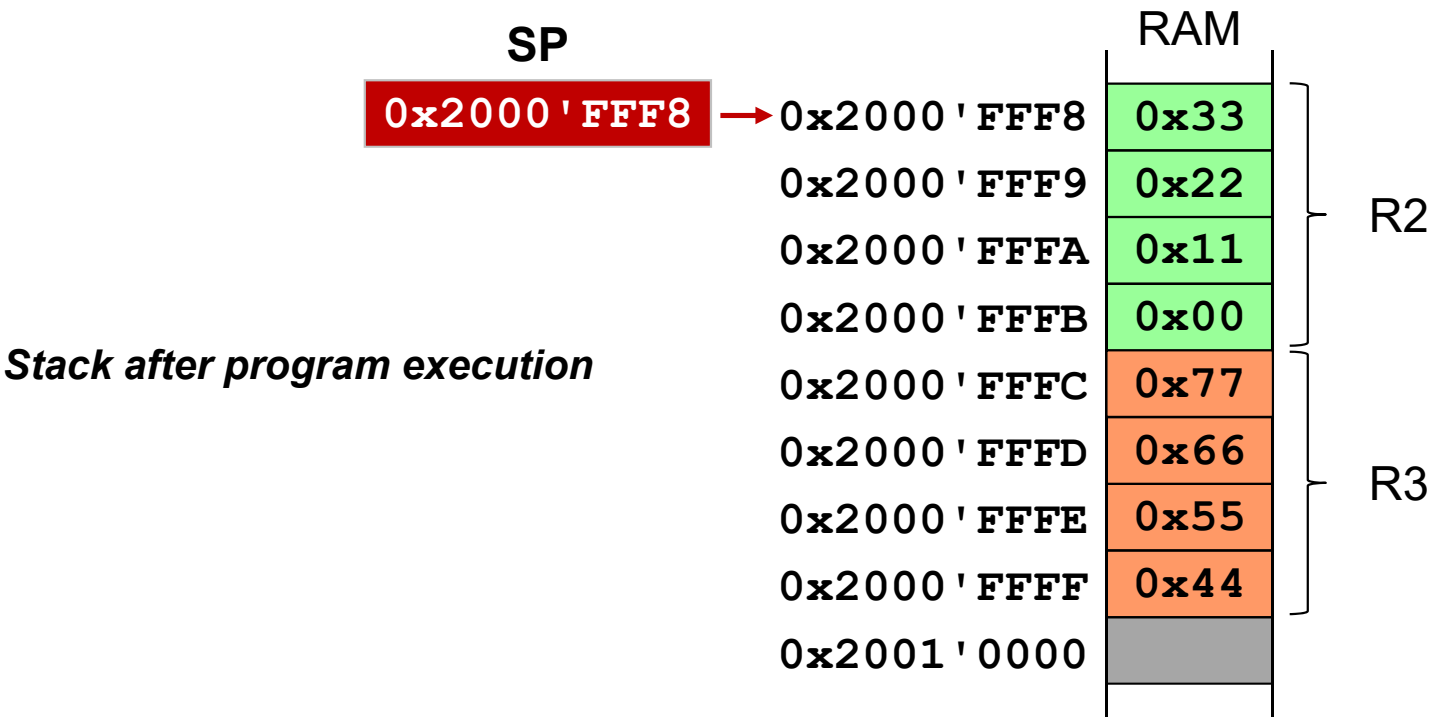
ARM: PUSH and POP

School of Engineering
InES Institute of Embedded Systems

- **Storage Order PUSH**
  - Lowest register
    - stored to lowest address [1)]

Example

```
LDR     R1,=0x20010000
MOV     SP,R1
LDR     R2,=0x00112233
LDR     R3,=0x44556677
PUSH    {R2,R3}
```



Stack after program execution

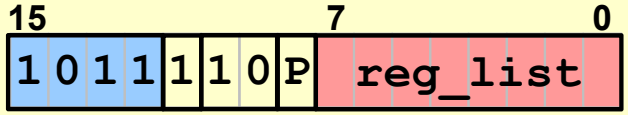
[1)] The lowest register is stored first.

ARM: PUSH and POP

School of Engineering
InES Institute of Embedded Systems

- **POP**
  - **`registers`**
    - One or more registers to be restored
    - Low registers
      → **`reg_list`** = one bit per register
    - PC (R15) → P-bit
      → No other high registers
    - Lowest register reloaded first

```
POP {registers}
```




```
addr = SP

for i = 0 to 7
    if reg_list<i> == '1' then
        R[i] = Mem[addr,4]
        addr = addr + 4

if (P == '1') then
     PC = Mem[addr]

SP = SP + 4*BitCount(P::reg_list)
```

```
00000000 BC80      POP      {R7}
00000002 BC3A      POP      {R1,R3,R4,R5}
00000004 BC3A      POP      {R1,R3-R5}
00000006 BD00      POP      {PC}
00000008 BD80      POP      {R7,PC}
```

```
P::reg_list = 0x03A
 = 0'0011'1010b
```

ARM: PUSH and POP

School of Engineering
InES Institute of Embedded Systems

- **ARM Stack**
  - Only Words → 32-bit
  - Pushing and popping of half-words and bytes not possible
  - I.e. SP mod 4 = 0 → word aligned
- **"Number of PUSHs" = "Number of POPs"**
- **Stack-limit < SP < stack-base**
  - Stack size has to fit program requirements

Nested Subroutines (revisited)

School of Engineering
InES Institute of Embedded Systems

Save LR on Stack

```
ADDR_LED_31_0   EQU     0x60000100
LED_PATTERN     EQU     0xA55A5AA5

subrExample     PUSH    {R4,R5,LR}

                ; write pattern to LEDs
                LDR     R4,=ADDR_LED_31_0
                LDR     R5,=LED_PATTERN
                STR     R5,[R4]

                BL      write7seg

                POP     {R4,R5,PC}
```

Save LR and registers used by subroutine

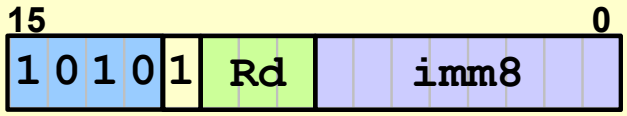
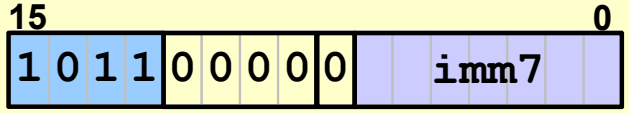
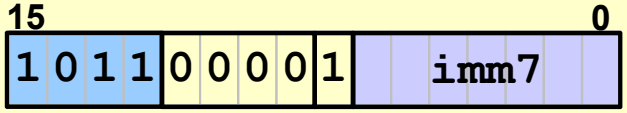
Call another subroutine

Restore registers and PC

Please note: **`BX LR`** is not required here, as we directly restore the PC using **`POP`**

Instructions using SP

- **Add to / subtract from SP**
  - Immediate offset <imm>
  - Offset range 0 – 1020d and 0 – 508d respectively

	ADD (SP plus immediate)	SUB (SP minus immediate)
load stack pointer plus offset into register	ADD <Rd>,SP,#<imm>  <imm> = imm8:00 Rd = SP + <imm>	high registers as destination are not supported
allocate (SUB) / deallocate (ADD) memory on stack	ADD SP,SP,#<imm>  <imm> = imm7:00 SP = SP + <imm>	SUB SP,SP,#<imm>  <imm> = imm7:00 SP = SP - <imm>

no flags will be changed

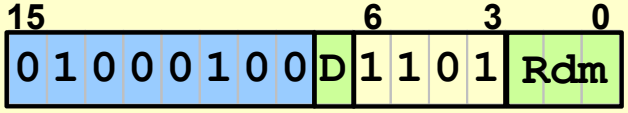
Instructions using SP

Add to SP (Register)

ADD (SP plus register)

load stack pointer plus offset into register

`ADD <Rdm>,SP,<Rdm>`

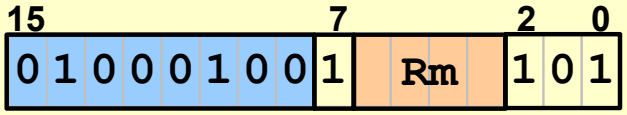


`Rdm = SP + Rdm`

Target → any register except SP
bits[6:3] = 1101b → SP

deallocate (ADD) memory on stack

`ADD SP,SP,<Rm>`



`SP = SP + Rm`

Target → SP
bits[7, 2:0] = 1101b → SP

Instructions using SP

School of Engineering
InES Institute of Embedded Systems

- **Instructions with Opcodes previously covered**

```
CMP SP,Rm
CMP Rn,SP
MOV SP,Rm
MOV Rd,SP
```

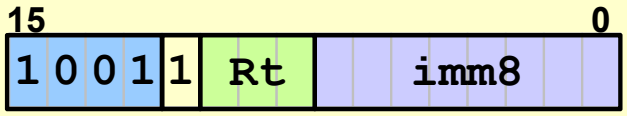
Instructions using SP

Accessing Memory using SP

- Immediate offset <imm>
- Offset range 0 – 1020d
- Word transfers

LDR (immediate) T2

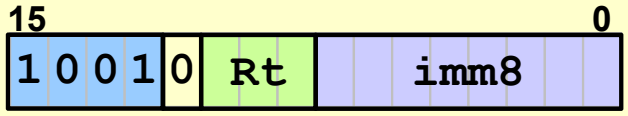
```
LDR <Rt>,[SP,#<imm>]
```




```
<imm> = imm8:00
Rt = Mem[SP + <imm>]
```

STR (immediate) T2

```
STR <Rt>,[SP,#<imm>]
```




```
<imm> = imm8:00
Mem[SP + <imm>] = Rt
```

Instructions using SP

School of Engineering
InES Institute of Embedded Systems

- **Using other instructions to implement** [1)]
  - **PUSH {R2,R3,R6}**

```
00000000 B083      SUB     SP,SP,#12
00000002 9200      STR     R2,[SP]
00000004 9301      STR     R3,[SP,#4]
00000006 9602      STR     R6,[SP,#8]
```

  - **POP {R2,R3,R6}**

```
00000008 9A00      LDR     R2,[SP]
0000000A 9B01      LDR     R3,[SP,#4]
0000000C 9B02      LDR     R6,[SP,#8]
0000000E B003      ADD     SP,SP,#12
```

[1)] Caveat: the alternate implementations are not atomic

Assembler Directives

School of Engineering
InES Institute of Embedded Systems

- **Assembler Directives**
  - PROC / ENDP
  - FUNCTION / ENDFUNC
- **Mark start and end of a procedure / function**
  - Used by debugger (tool)
    - Buttons "step over" and "step out"
  - Structure code for reader

```
subrExample     PROC
                PUSH    {...,LR}
                ...
                ...
                POP     {...,PC}
                ENDP
```

Conclusions

School of Engineering
InES Institute of Embedded Systems

- **Subroutines**
  - Structured programming
    - → Avoids duplicated code / clear interface
  - Call and return on ARM:
    - **`BL <label>`** and **`BX LR / POP {PC}`**
  - Nested subroutines → save LR on stack
- **Stack**
  - Continuous area of memory → Last-in First-Out
  - **`PUSH`** und **`POP`**
  - ARM
    - Full-descending stack
    - SP points to last entry that has been written
    - grows from higher towards lower addresses