Subroutines

R. Ferrero

Politecnico di Torino

Dipartimento di Automatica e Informatica (DAUIN)

Torino - Italy

Subroutine

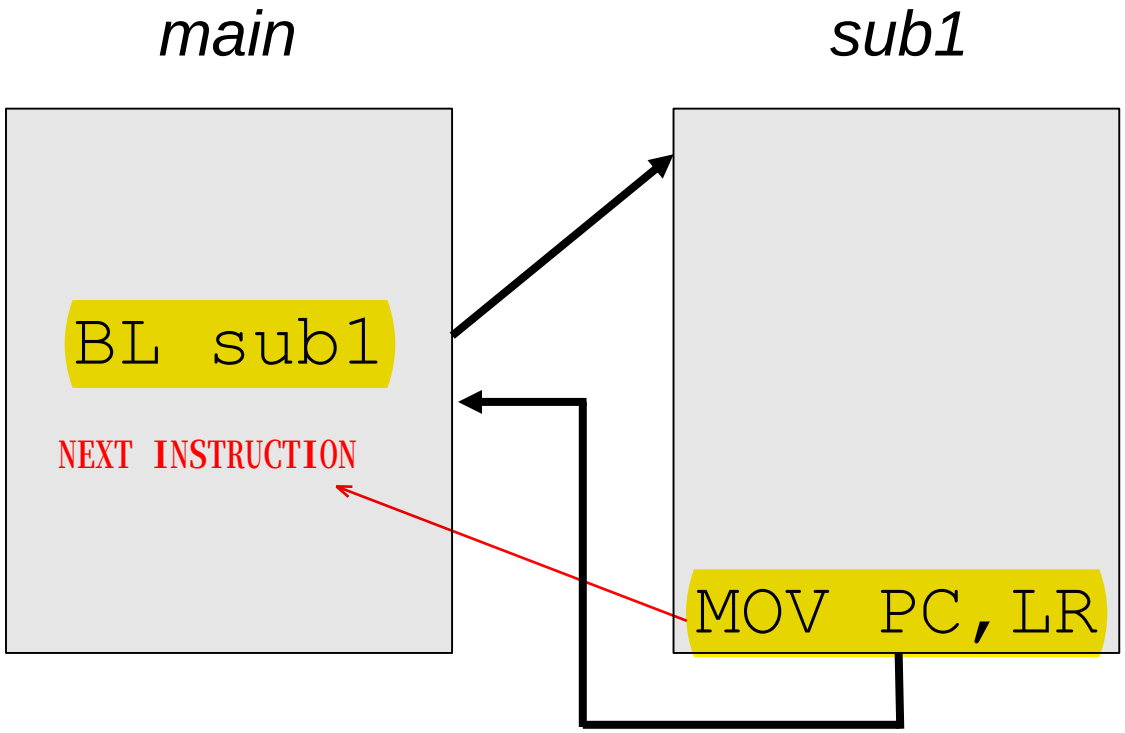
子程序

BL= BRANCH LINK
BLX = BRANCH LINK REGISTER

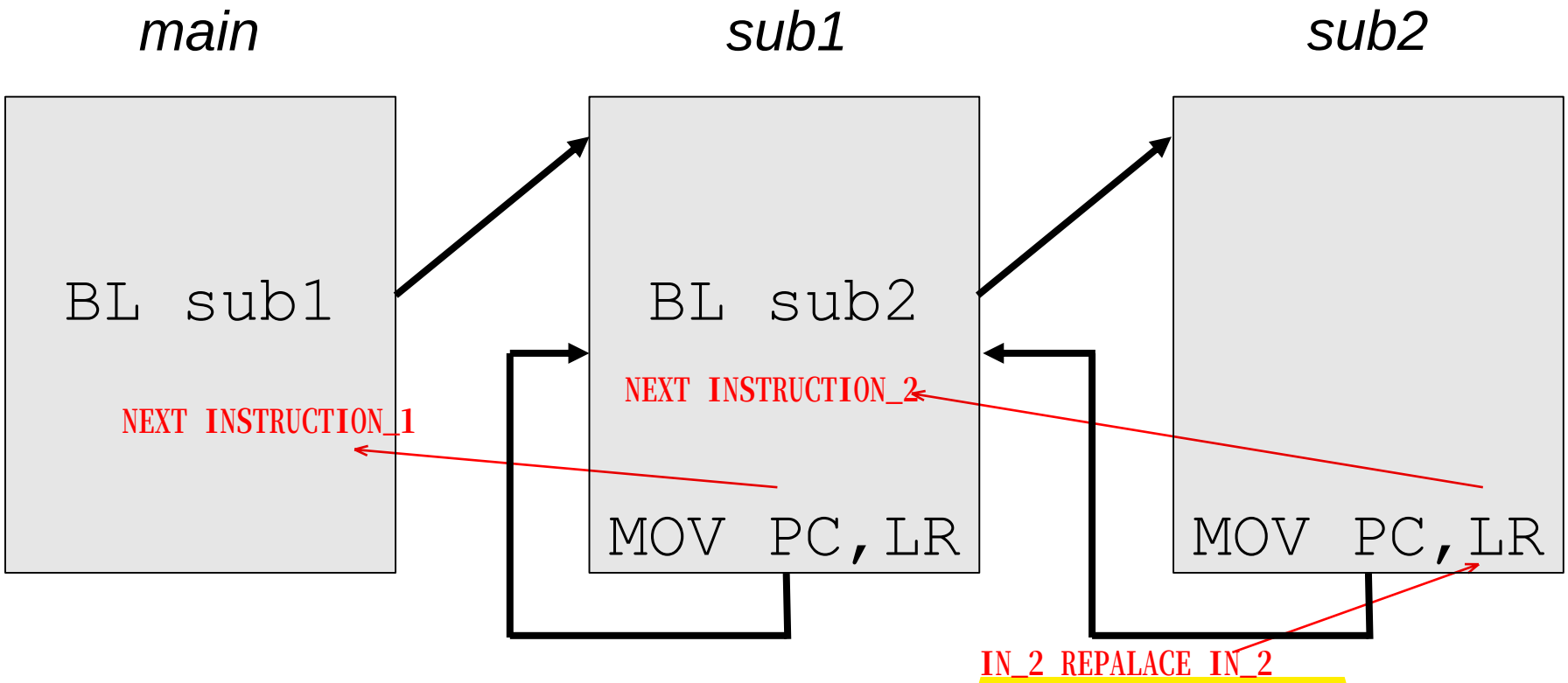
- A subroutine is called with `BL` and `BLX`.
- `BL <label>` and `BLX <Rn>`:
  - write the address of the next instruction to `LR`
  - write the value of `label` or `Rn` to `PC`
- A reentrant procedure ends with a branch to the address stored in `LR`.
- Optionally, the begin and end of a subroutine can be indicated with the directives `PROC`/`FUNCTION` and `ENDP`/`ENDFUNC`.

NOTE: PC = R15 LR=R14 SP=R13

Call to subroutine



Nested calls to subroutines



- When *sub1* calls *sub2*, `LR` is overwritten.
- *sub1* is not able to return to *main*.

Nested calls to subroutines

嵌套的

- Besides changing `LR` when called, *sub2* may also change the value of registers used in *sub1*.
- Every subroutine should save `LR` and the other used registers as first instruction:

```
PUSH {regList, LR}
```

- At the end, the subroutine restores `PC` and the initial value of the used registers:

```
POP {regList, PC}
```

LR's value lord in PC

Passing parameters and result

- There are three approaches:
  - in registers
  - by reference, i.e., a register with an address in memory
  - on the stack
- Example: a main routine calls a subroutine for computing the absolute difference of two unsigned numbers.

Passing parameters in registers

```
    MOV r0, #0x34
    MOV r1, #0xA3
    BL sub1
    ; r2 contains the result
...
stop B stop
...
```

Passing parameters in registers

```
sub1    PROC
        PUSH {LR}
        CMP r0, r1
        ITE HS
        SUBHS r2, r0, r1
        SUBLO r2, r1, r0
        POP {PC}
        ENDP
```

Passing parameters by reference

```
    MOV r0, #0x34
    MOV r1, #0xA3
    LDR r3, =mySpace
    STMIA r3, {r0, r1}
    BL sub2
    LDR r2, [r3]
    ; r2 contains the result
...
stop B stop
...
```

Passing parameters by reference

```
sub2    PROC
        PUSH {r2, r4, r5, LR}
        LDMIA r3, {r4, r5}
        CMP r4, r5
        ITE HS
        SUBHS r2, r4, r5
        SUBLO r2, r5, r4
        STR r2, [r3]
        POP {r2, r4, r5, PC}
        ENDP
```

Passing parameters on the stack

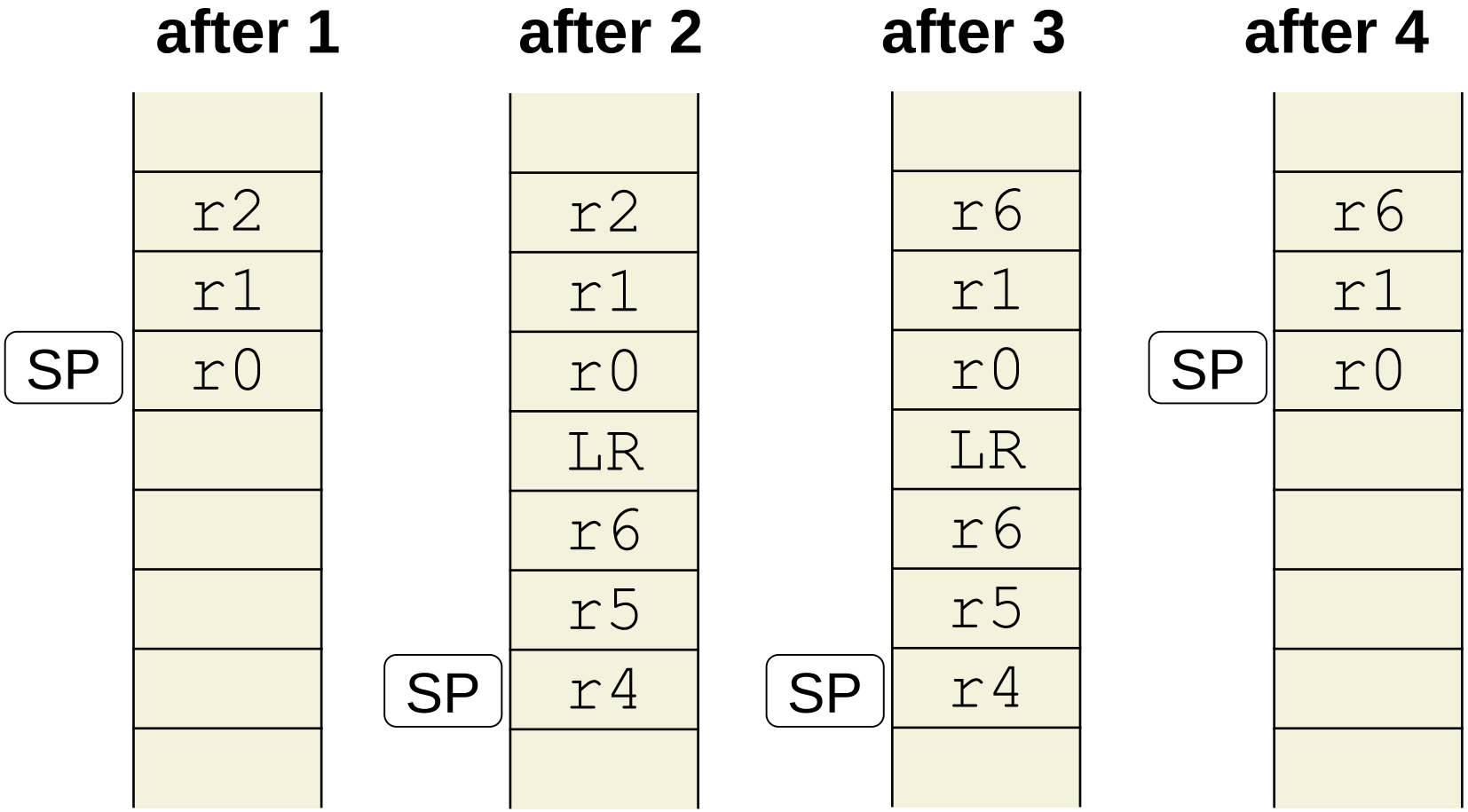
```
    MOV r0, #0x34
    MOV r1, #0xA3
1   PUSH {r0, r1, r2}
    BL sub3
    POP {r0, r1, r2}
    ; r2 contains the result
...
stop B stop
...
```

from r2 to r0 store into the stack according to the number of register

Passing parameters on the stack

```
sub3    PROC
(2)     PUSH {r6, r4, r5, LR}
        LDR r4, [sp, #16]      mov R0 to R4
        LDR r5, [sp, #20]
        CMP r4, r5
        ITE HS
        SUBHS r6, r4, r5
        SUBLO r6, r5, r4
(3)     STR r6, [sp, #24]
(4)     POP {r6, r4, r5, PC}
        ENDP
```

Elements in the stack



ARM Architecture Procedure Call Standard (AAPCS)

- It regulates the interaction between a calling program and the called subroutine:
  - obligations on the caller: proper program state
  - obligations on the called subroutine: which parts of the program state must be preserved
  - rights of the called subroutine: which parts of the program state can be changed.
- It is part of Application Binary Interface (ABI).

Registers and AAPCS usage

Register	Synonym	Special	Role in the procedure call standard
r15		PC	The Program Counter
r14		LR	The Link Register
r13		SP	The Stack Pointer
r12		IP	The Intra-Procedure-call scratch register
r11	v8		Variable-register 8
r10	v7		Variable-register 7
r9		v6, SB, TR	Platform register. Meaning defined by platform standard
r8	v5		Variable-register 5
r7	v4		Variable-register 4
r6	v3		Variable-register 3
r5	v2		Variable-register 2
r4	v1		Variable-register 1
r3	a4		Argument / scratch register 4
r2	a3		Argument / scratch register 3
r1	a2		Argument / scratch register 2
r0	a1		Argument / scratch register 1

Obligations on the caller

- The first 4 parameters are passed in `r0`-`r3`.
- Further parameters are passed in the stack.
- After returning from the called subroutine, parameters must be removed from the stack
  - i.e., the stack pointer must have the same value as before the call.

Obligations & rights of the called

- The subroutine must preserve the contents of the registers `r4`-`r8`, `r10`, `r11` and `SP`.
- The subroutine can use `r0`-`r3` and `r12` as scratch registers.
- The return value is passed in `r0`-`r3`:
  - 32-bit sized type -> `r0`
  - 64-bit sized type -> `r0`-`r1`
  - 128-bit sized type -> `r0`-`r4`

Size of the data types

Type Class	Machine Type	Byte size	Byte alignment	Note
Integral	Unsigned byte	1	1	Character
	Signed byte	1	1	
	Unsigned half-word	2	2	
	Signed half-word	2	2	
	Unsigned word	4	4	
	Signed word	4	4	
	Unsigned double-word	8	8	
	Signed double-word	8	8	
Floating Point	Half precision	2	2	See *§4.1.1, Half-precision Floating Point.*
	Single precision (IEEE 754)	4	4	The encoding of floating point numbers is described in [ARM ARM] chapter C2, *VFP Programmer's Model*, *§2.1.1 Single-precision format*, and *§2.1.2 Double-precision format.*
	Double precision (IEEE 754)	8	8	
Containerized vector	64-bit vector	8	8	See *§4.1.2, Containerized Vectors.*
	128-bit vector	16	8	
Pointer	Data pointer	4	4	Pointer arithmetic should be unsigned.
	Code pointer	4	4	Bit 0 of a code pointer indicates the target instruction set type (0 ARM, 1 Thumb).

Table 1, Byte size and byte alignment of fundamental data types