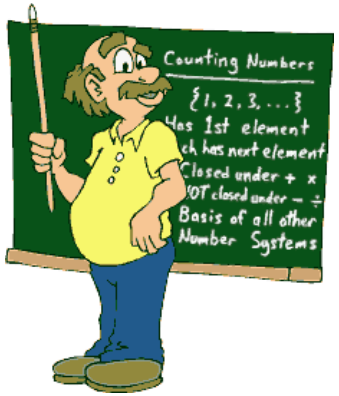
IECA

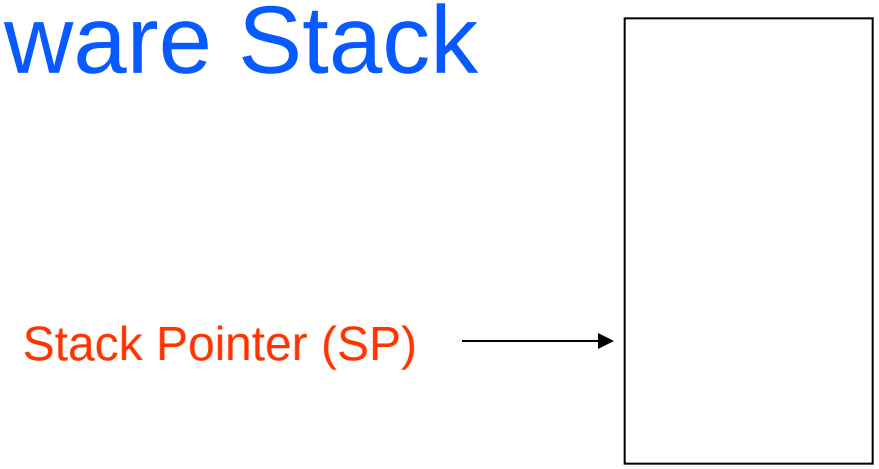
Embedded Computer Architecture

Lesson 7: Stack



Version: 1-11-2015, Henning Hargaard

Hardware Stack



- The stack pointer is an important register, keeping track of where the ”top of the stack” is (i.e. next free location (address)).
- Each time, data is written to the stack, the SP is automatically **decremented**.
- Each time, data is fetched from the stack, the SP is automatically **incremented**.

Stack pointer (SP) = SPH og SPL

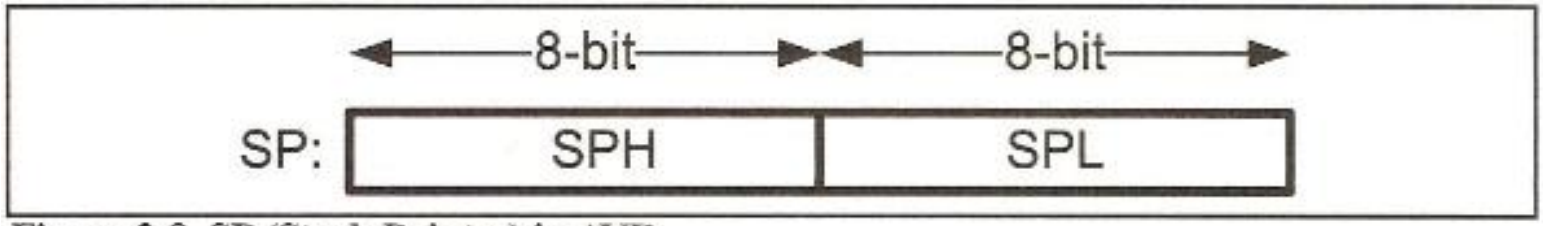


Figure 3-8. SP (Stack Pointer) in AVR

IMPORTANT:

Before using the stack, the stack pointer has to be initialized (point to the ”top of stack”) !

In assembly, you can do it this way:

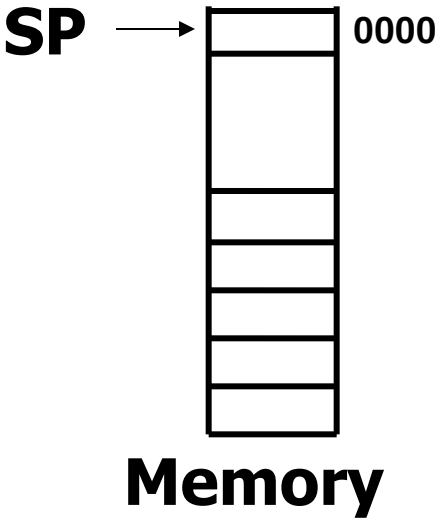
```
LDI   R16, HIGH(RAMEND)
OUT   SPH,R16
LDI   R16, LOW(RAMEND)
OUT   SPL,R16
```

PUSH register / POP register

- We may as (assembly-) programmers use the stack to store data temporarily.
- ”PUSH register” (f.ex. PUSH R17) copies the register contents to the top of stack (and automatically **decrements** the SP).
- ”POP register” (f.ex. POP R4) copies from the top of stack to the register (and automatically **increments** the SP).

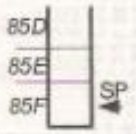
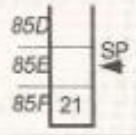
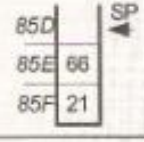
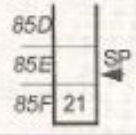
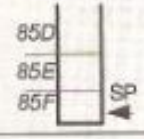
Stack (PUSH and POP)

R20:	$10	R22:	$30
R21:	$20	R0:	$00



Address	Code
	ORG 0
0000	LDI R16,HIGH(RAMEND)
0001	OUT SPH,R16
0002	LDI R16,LOW(RAMEND)
0003	OUT SPL,R16
0004	LDI R20,0x10
0005	LDI R21,0x20
0006	LDI R22,0x30
0007	PUSH $10
0008	PUSH $20
0009	PUSH $30
000A	POP R21
000B	POP R0
000C	POP R20
000D	L1: RJMP L1

Example 3-8 (page 120): PUSH/POP

After the execution of	Contents of some of the registers				Stack
	R20	R22	R31	SP	
OUT SPL,R16	$0	$0	0	$085F	
LDI R22, 0x66	$21	$66	0	$085F	
PUSH R20	$21	$66	0	$085E	
PUSH R22	$21	$66	0	$085D	
LDI R22, 0	$0	$0	0	$085D	
POP R22	$0	$66	0	$085E	
POP R31	$0	$66	$21	$085F	

The CALL instruction

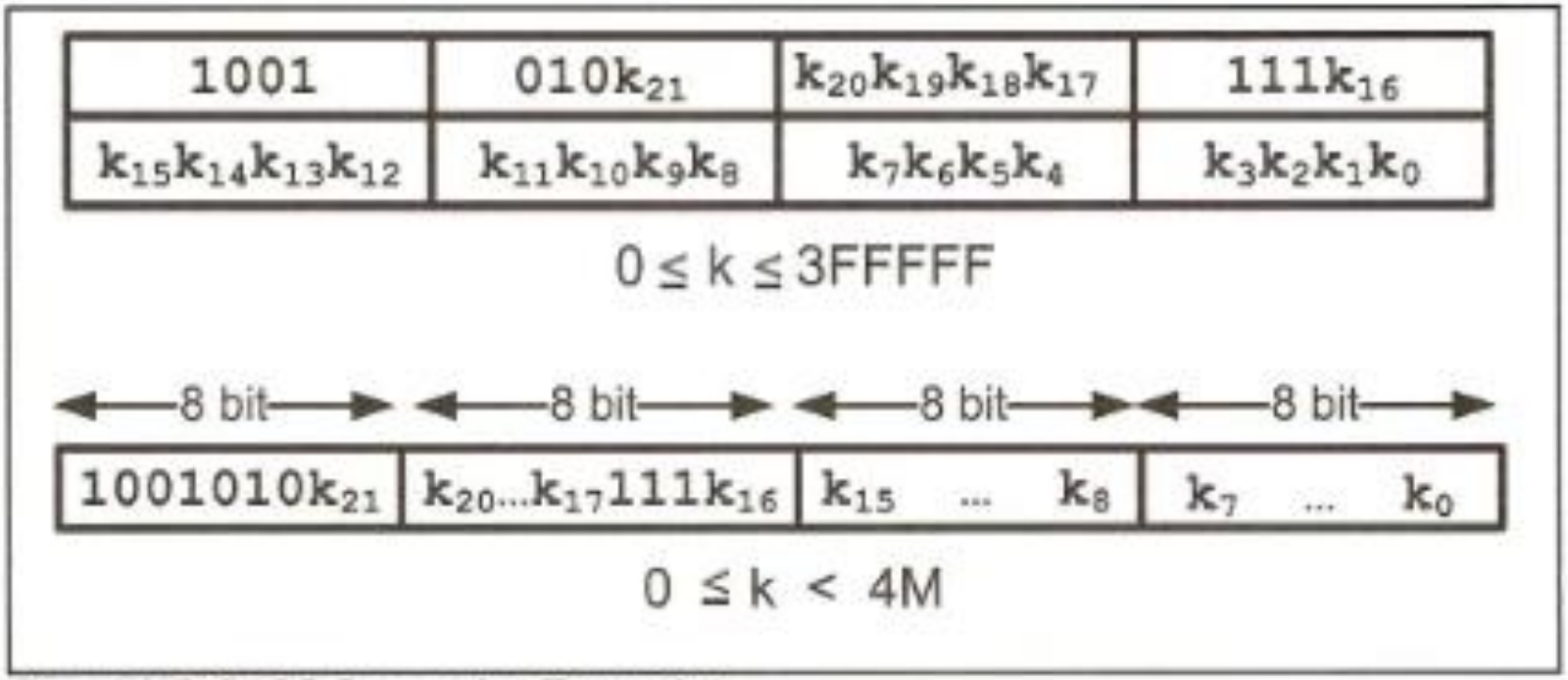
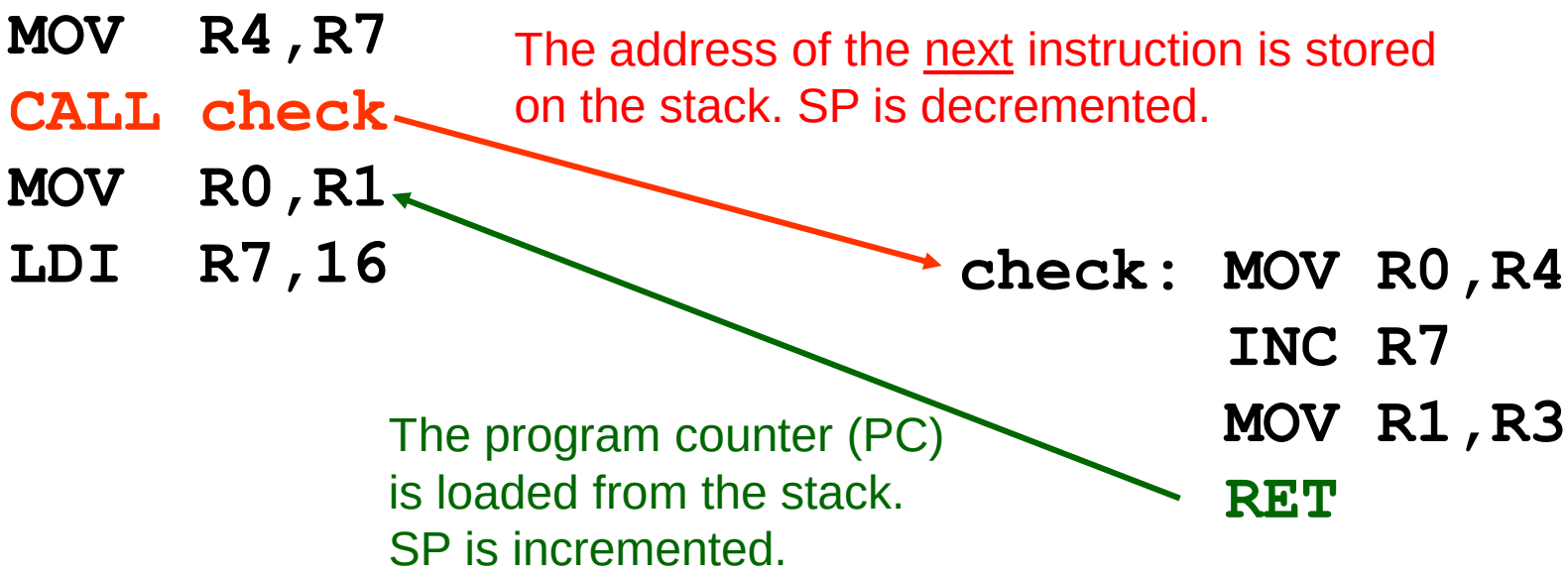


Figure 3-7. CALL Instruction Formation

There is also a “relative CALL” of smaller size.
Example: RCALL HUGO

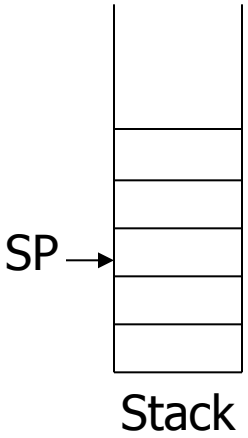
Calling subroutines

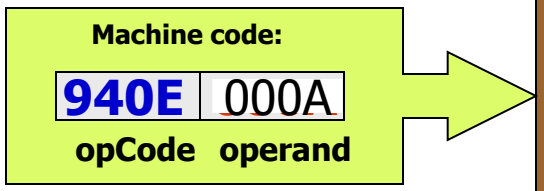


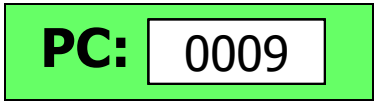
- When using subroutines, the stack is (automatically) used to ”find the way back” till the instruction after ”CALL” : Literally a ”bookmark” is saved at the stack.

Calling a Function

- To execute a call:
  - Address of the next instruction is saved
  - PC is loaded with the appropriate value

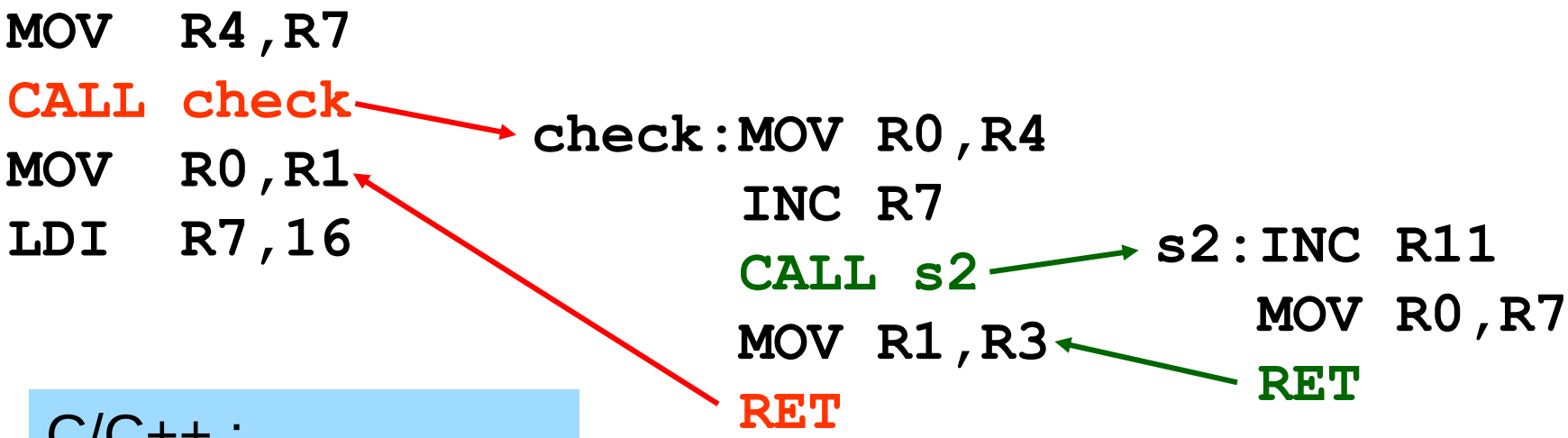






Address	Code
0000	LDI R16,HIGH(RAMEND)
0001	OUT SPH,R16
0002	LDI R16,LOW(RAMEND)
0003	OUT SPL,R16
0004	LDI R20,15
0005	LDI R21,5
0006	CALL FUNC_NAME
00 08	INC R20
0009	L1: RJMP L1
000A	FUNC_NAME:
000A	ADD R20,R21
000B	SUBI R20,3
000C	RET
000D	

Multi-level CALLs



C/C++ :
check(s2());

- When calling routines in multiple levels, the SP still keeps track of the actual adresses to return to.
- Risc of ”stack overflow” !

Typical use of CALL

```
.INCLUDE "M32DEF.INC"   ;Modify for your chip

;MAIN program calling subroutines
             .ORG   0
MAIN:        CALL   SUBR_1
             CALL   SUBR_2
             CALL   SUBR_3
             CALL   SUBR_4
HERE:        RJMP   HERE          ;stay here
;------------end of MAIN
;
SUBR_1:      ....
             ....
             RET
;------------end of subroutine 1
;
SUBR_2:      ....
             ....
             RET
;------------end of subroutine 2

SUBR_3:      ....
             ....
             RET
;------------end of subroutine 3

SUBR_4:      ....
             ....
             RET
;------------end of subroutine 4
```

Figure 3-9. AVR Assembly Main Program That Calls Subroutines

End of lesson 7