EE-222: Microprocessor Systems

Nested Loops
and
Stack & Call

Instructor: Dr. Arbab Latif

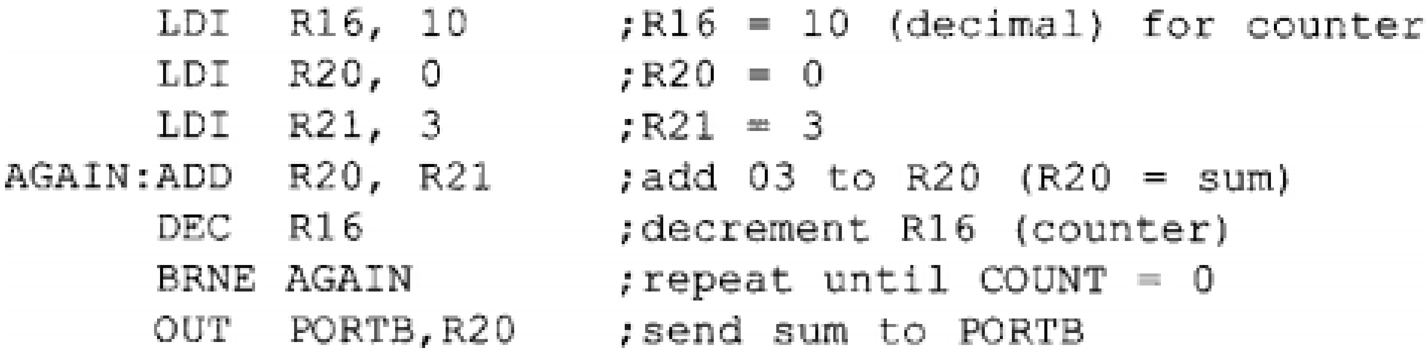
NUST

SCHOOL OF ELECTRICAL ENGINEERING &
COMPUTER SCIENCE (SEECS)

Nested Loops

Looping: Max you can run

- What is the maximum number of times that following loop can be repeated?

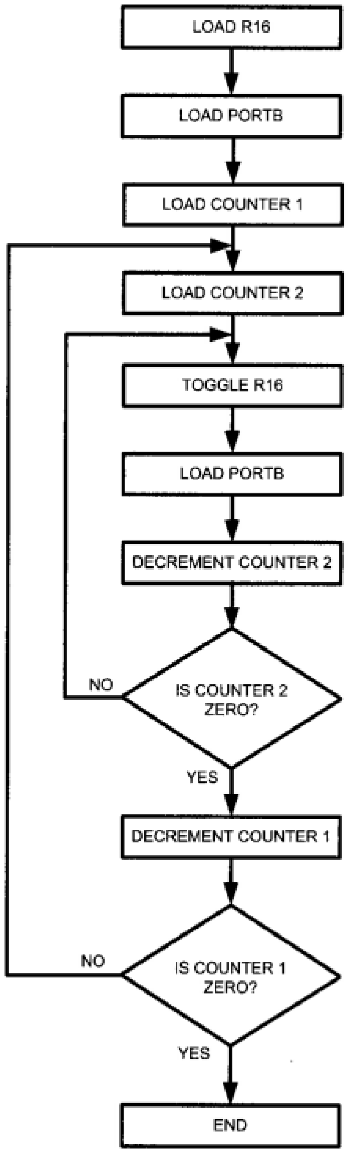


```
        LDI   R16, 10       ;R16 = 10 (decimal) for counter
        LDI   R20, 0        ;R20 = 0
        LDI   R21, 3        ;R21 = 3
AGAIN:ADD   R20, R21      ;add 03 to R20 (R20 = sum)
        DEC   R16           ;decrement R16 (counter)
        BRNE  AGAIN         ;repeat until COUNT = 0
        OUT   PORTB,R20     ;send sum to PORTB
```

Looping Example

- Write a program to:
  a. Load PORTB register with the value 0x55
  b. Complement Port B 700 times

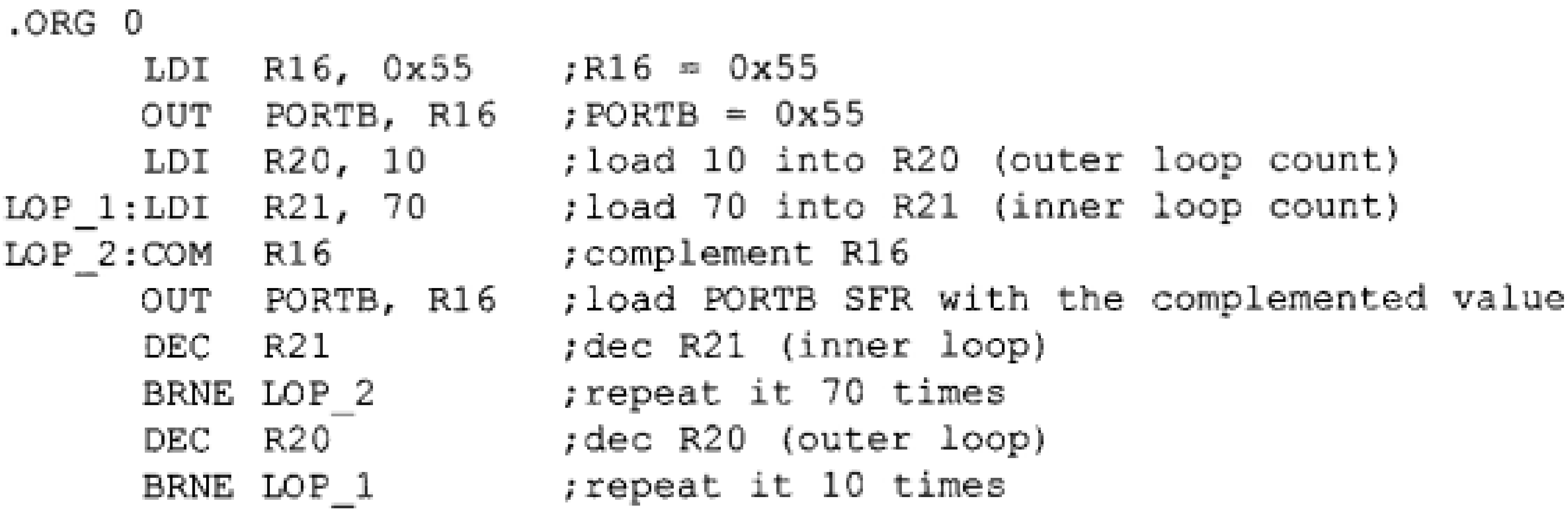
Nested Loops: Loop inside a Loop -> Flowchart



INSTRUCTIONS

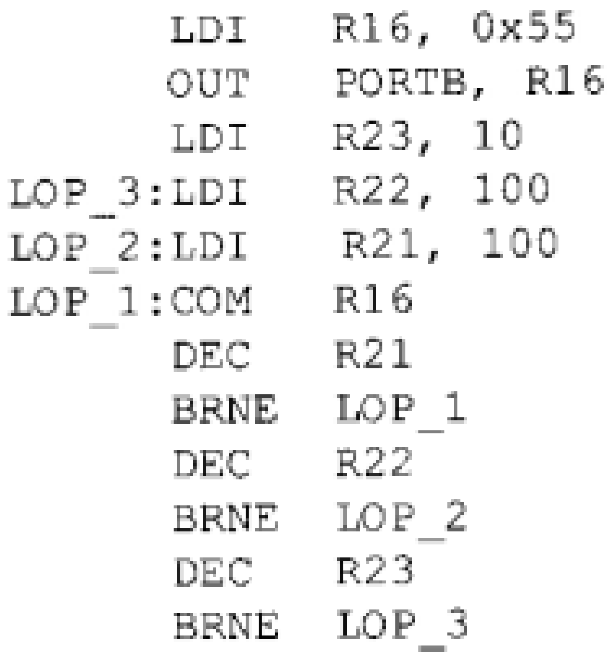
```
       LDI  R16, 0x55
       OUT  PORTB, R16
       LDI  R20, 10
LOP_1: LDI  R21, 70
LOP_2: COM  R16
       OUT  PORTB, R16
       DEC  R21
       BRNE LOP_2
       DEC  R20
       BRNE LOP_1
```

Nested Loops: Loop inside a Loop - Overall



```
.ORG 0
        LDI   R16, 0x55      ;R16 = 0x55
        OUT   PORTB, R16     ;PORTB = 0x55
        LDI   R20, 10        ;load 10 into R20 (outer loop count)
LOP_1:LDI   R21, 70        ;load 70 into R21 (inner loop count)
LOP_2:COM   R16            ;complement R16
        OUT   PORTB, R16     ;load PORTB SFR with the complemented value
        DEC   R21            ;dec R21 (inner loop)
        BRNE  LOP_2          ;repeat it 70 times
        DEC   R20            ;dec R20 (outer loop)
        BRNE  LOP_1          ;repeat it 10 times
```

Nested Loops: Loop in a Loop in a Loop



```
        LDI   R16, 0x55
        OUT   PORTB, R16
        LDI   R23, 10
LOP_3:LDI   R22, 100
LOP_2:LDI    R21, 100
LOP_1:COM   R16
        DEC   R21
        BRNE  LOP_1
        DEC   R22
        BRNE  LOP_2
        DEC   R23
        BRNE  LOP_3
```

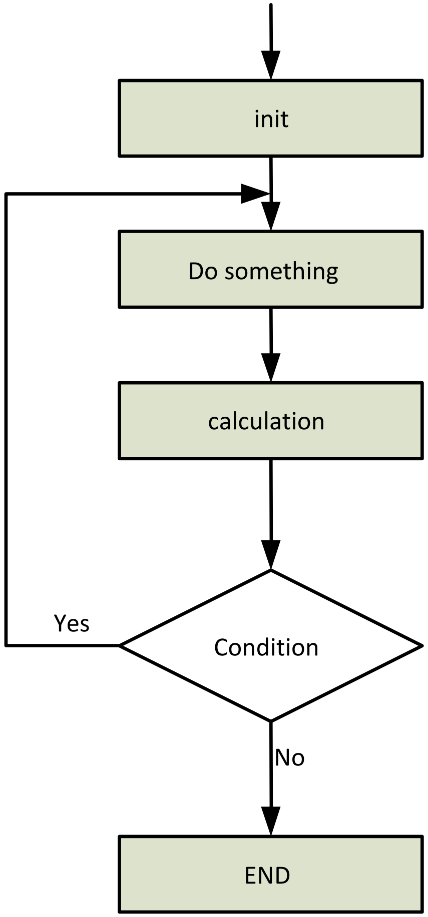
How many times would this be repeated?

Loops Examples: DIY

Also make sure you attempt the book examples

Loop

```
for (init; condition; calculation)
{
    do something
}
```

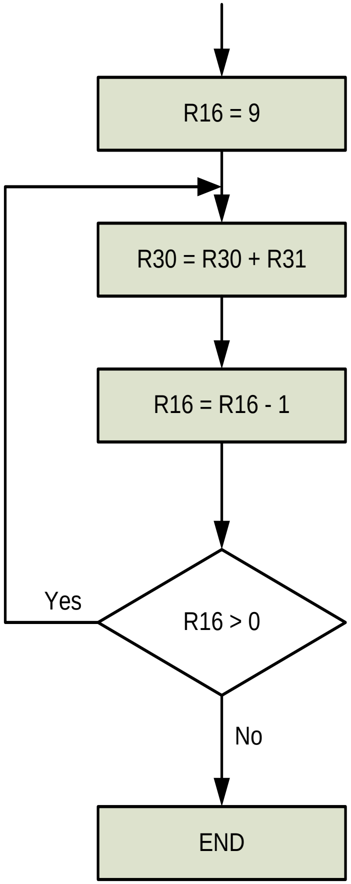


Loop

- Write a program that executes the instruction “ADD R30,R31” 9 times.

- **Solution:**

```
     .ORG 00
     LDI  R16,9           ;R16 = 9
L1:  ADD  R30,R31
     DEC  R16             ;R16 = R16 - 1
     BRNE L1              ;if Z = 0
L2:  RJMP L2              ;Wait here forever
```

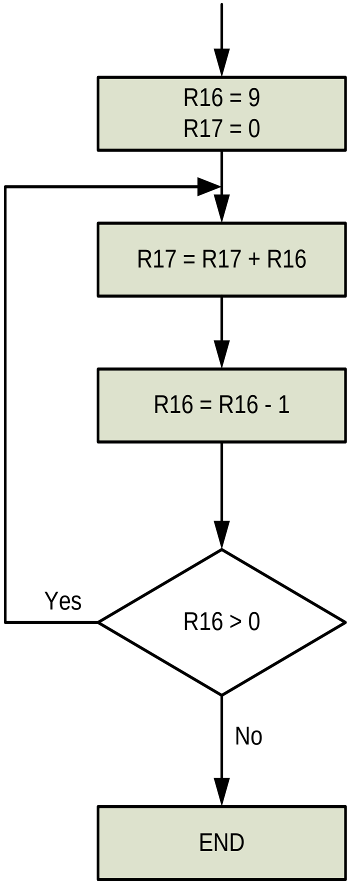


Loop

- Write a program that calculates the result of 9+8+7+…+1

- **Solution:**

```
     .ORG 00
     LDI  R16, 9          ;R16 = 9
     LDI  R17, 0          ;R17 = 0
L1:  ADD  R17,R16         ;R17 = R17 + R16
     DEC  R16             ;R16 = R16 - 1
     BRNE L1              ;if Z = 0
L2:  RJMP L2              ;Wait here forever
```

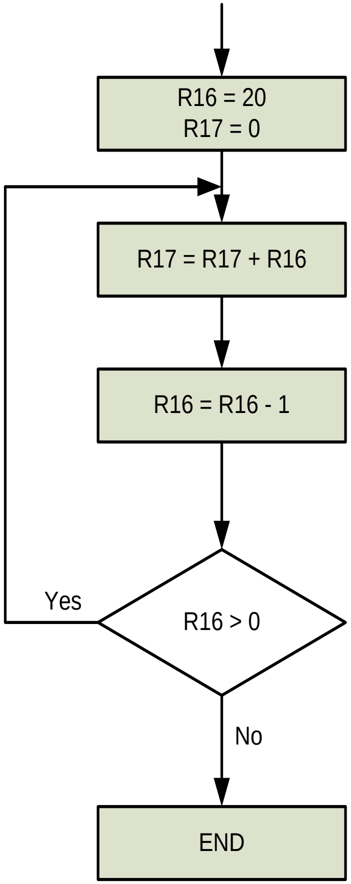


Loop

- Write a program that calculates the result of 20+19+18+17+…+1

- **Solution:**

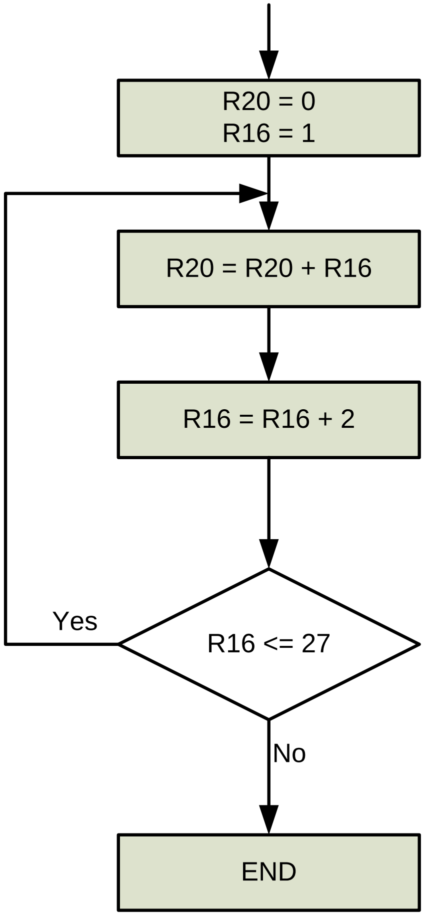
```
     .ORG 00
     LDI  R16, 20          ;R16 = 20
     LDI  R17, 0           ;R17 = 0
L1:  ADD  R17,R16          ;R17 = R17 + R16
     DEC  R16              ;R16 = R16 - 1
     BRNE L1               ;if Z = 0
L2:  RJMP L2               ;Wait here forever
```



Loop

- Write a program that calculates 1+3+5+…+27
- Solution:

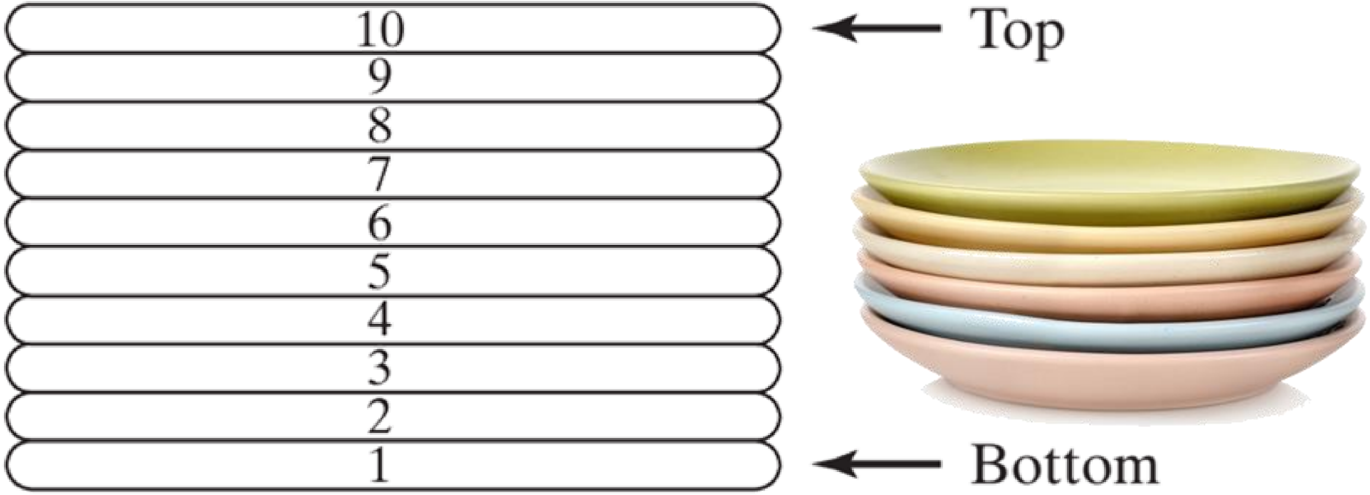
```
    LDI R20,0
    LDI R16,1
 L1:ADD R20,R16
    LDI R17,2
    ADD R16,R17 ;R16 = R16 + 2
    LDI R17,27  ;R17 = 27
    SUB R17,R16
    BRCC L1     ;if R16 <= 27 jump L1
```



Stack

Stack: LIFO

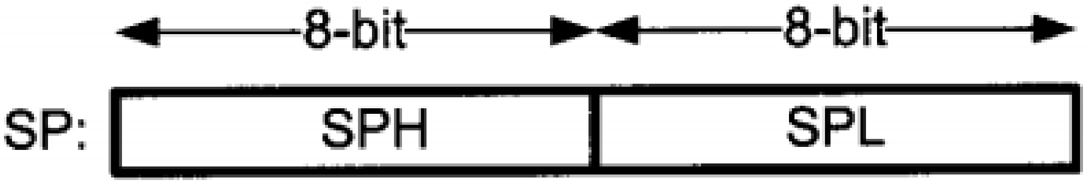
Stack of plates: **Easy to remove and add plate from top.**



Stack in AVR

- Stack is a section of RAM used by the CPU to store information temporarily:
  - Info could be Data or Address

- Stack is accessed by a register called Stack Pointer (SP):
  - Composed of two registers SPL and SPH



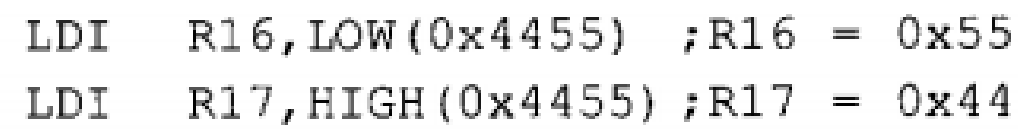
- Stack grows from higher memory location to lower memory location:
  - Therefore its common to initialize SP to the uppermost memory location

STACK Operations

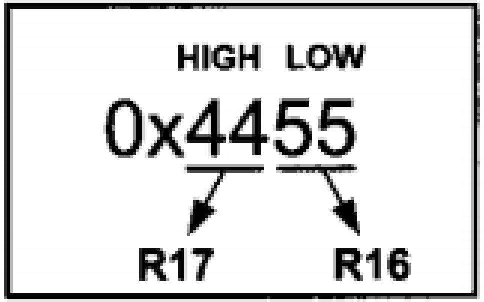
- PUSH
  - stores register on the top of the stack (TOS)
    - Data is saved where the SP points to
    - And the SP is decremented by one
  - `PUSH Rr ; Rr = (R0 – R31)`
    - `Example: PUSH R10 ; store R10 onto the stack`

- POP
  - loads stack contents back into a CPU register
    - Top location of the stack is copied back to a register
    - SP is incremented by one
  - `POP Rr ; Rr = (R0 – R31)`
    - `Example: POP R16 ; load TOS to R16`

STACK Example: HIGH and LOW Functions

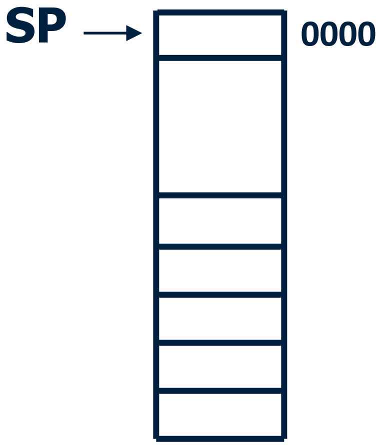


```
LDI   R16,LOW(0x4455)  ;R16 = 0x55
LDI   R17,HIGH(0x4455) ;R17 = 0x44
```



Stack

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



Memory

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

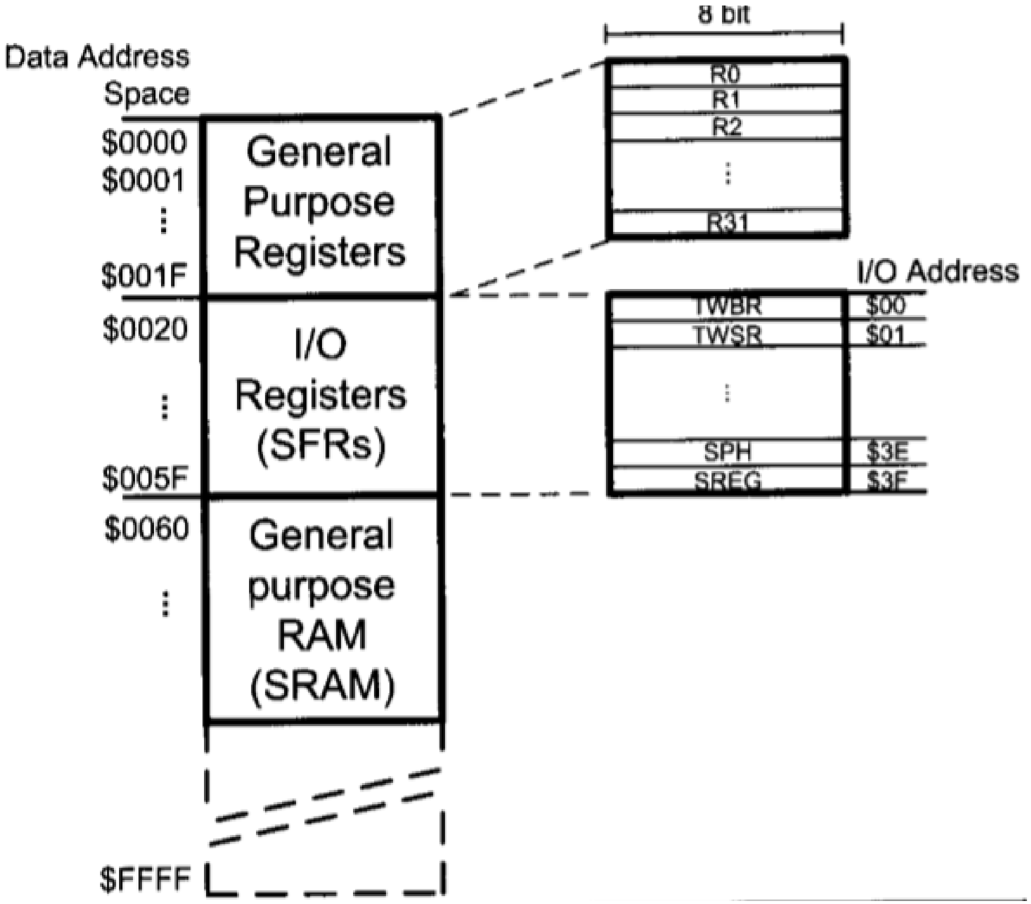
STACK Example

```
.ORG 0
;initialize the SP to point to the last location of RAM (RAMEND
LDI   R16, HIGH(RAMEND)          ;load SPH
OUT   SPH, R16
LDI   R16, LOW(RAMEND)            ;load SPL
OUT   SPL, R16

LDI   R31, 0
LDI   R20, 0x21
LDI   R22, 0x66

PUSH   R20
PUSH   R22

LDI    R20, 0
LDI    R22, 0

POP    R22
POP    R31
```

After the execution of	Contents of some of the registers				Stack
	R20	R22	R31	SP	
OUT SPL,R16	$0	$0	0	$085F	85D 85E 85F ← SP
LDI R22, 0x66	$21	$66	0	$085F	85D 85E 85F ← SP
PUSH R20	$21	$66	0	$085E	85D 85E ← SP 85F 21
PUSH R22	$21	$66	0	$085D	85D ← SP 85E 66 85F 21
LDI R22, 0	$0	$0	0	$085D	85D ← SP 85E 66 85F 21
POP R22	$0	$66	0	$085E	85D 85E ← SP 85F 21
POP R31	$0	$66	$21	$085F	85D 85E 85F ← SP

The Upper Limit of the Stack

- SP must be set to point above 0x60
  - Must not define the stack in the register memory nor in the I/O memory



Call Instructions

Why Subroutines?

- Divide programs into subroutines to perform tasks that need to be performed frequently:
  - Makes program more structured
    - Saves memory space

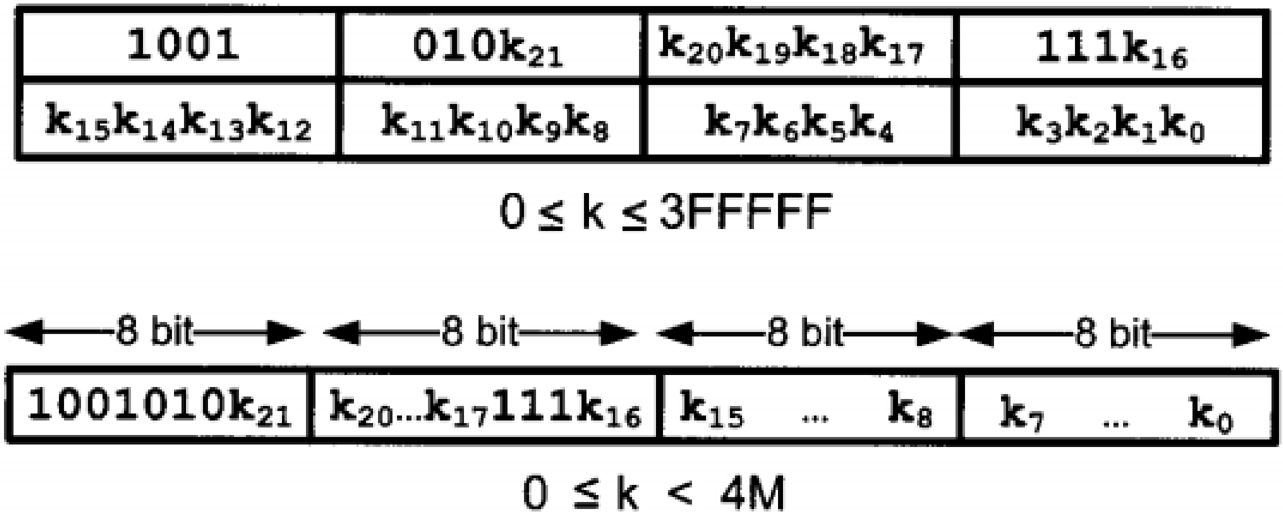
- A complicated problem is usually divided into separate subroutines.

How is a Subroutine Invoked?

- Four instructions to invoke a subroutine:
  - CALL (long call)
  - RCALL (relative call)
  - ICAL (indirect call)
  - EICALL (extended indirect call)

CALL Instruction

- 4-byte (32-bit) instruction
  - 10-bits are used for the opcode
  - 22-bits are used for the target subroutine



1001	010k_{21}	$k_{20}k_{19}k_{18}k_{17}$	111k_{16}
$k_{15}k_{14}k_{13}k_{12}$	$k_{11}k_{10}k_9k_8$	$k_7k_6k_5k_4$	$k_3k_2k_1k_0$

$0 \le k \le 3FFFFF$

8 bit	8 bit	8 bit	8 bit
1001010k_{21}	k_{20}…k_{17}111k_{16}	k_{15} … k_8	k_7 … k_0

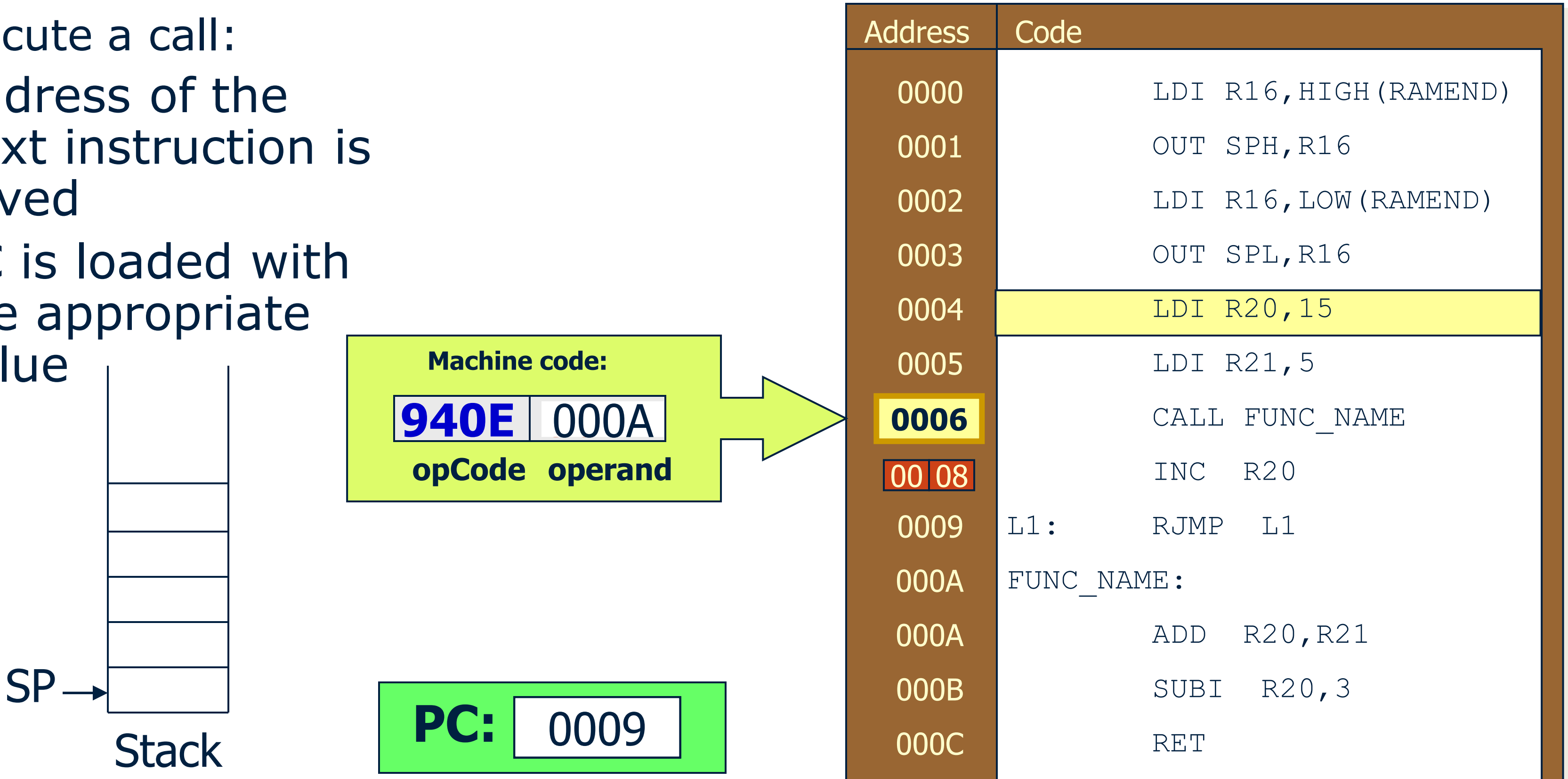
$0 \le k < 4M$

Steps in Calling a Function

- Following steps occur when a subroutine is called:
  1. Processor saves the PC of the next instruction on the stack
  2. Begins to fetch instructions from new location
  3. After finishing execution of the subroutine, the RET instruction transfers control back to the caller

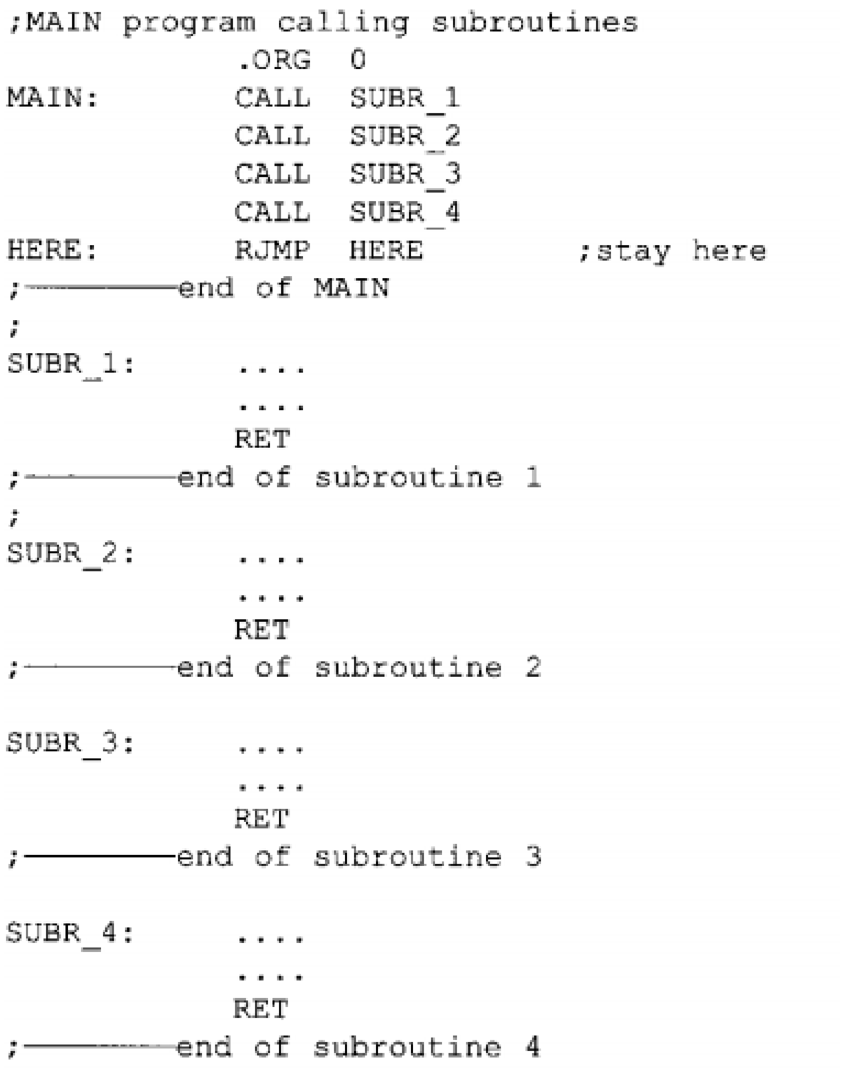
Calling a Function

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

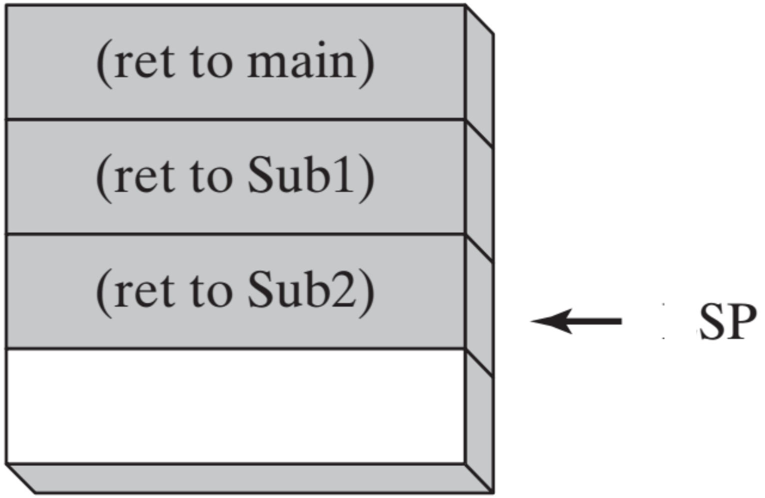


Address	Code
0000	LDI R16,HIGH(RAMEND)
0001	OUT SPH,R16
0002	LDI R16,LOW(RAMEND)
0003	OUT SPL,R16
0004	LDI R20,15
0005	LDI R21,5
0006	CALL FUNC_NAME
00 08	INC R20
0009	L1: RJMP L1
000A	FUNC_NAME:
000A	ADD R20,R21
000B	SUBI R20,3
000C	RET
000D	

Calling Many Subroutine from the Main Program



```
;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
```



Reading Assignment

- Read and explore:
  - RCALL
  - ICALL

Reading

- The AVR Microcontroller and Embedded Systems: Using Assembly and C by Mazidi et al., Prentice Hall
  - Chapter-3: 3.1 – 3.2

THANK YOU