Principles and Applications of Microcontrollers

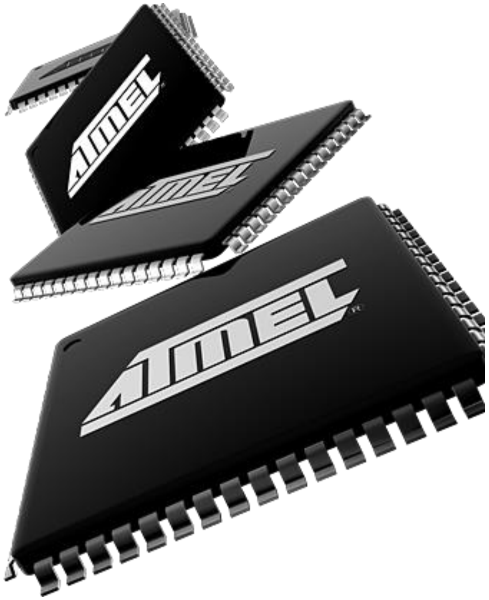
Yan-Fu Kuo

Dept. of Biomechatronics Engineering

National Taiwan University

Today:

- Structured programming



Outline

- Structured programming
  - Jump
  - Calling a function
- Getting started

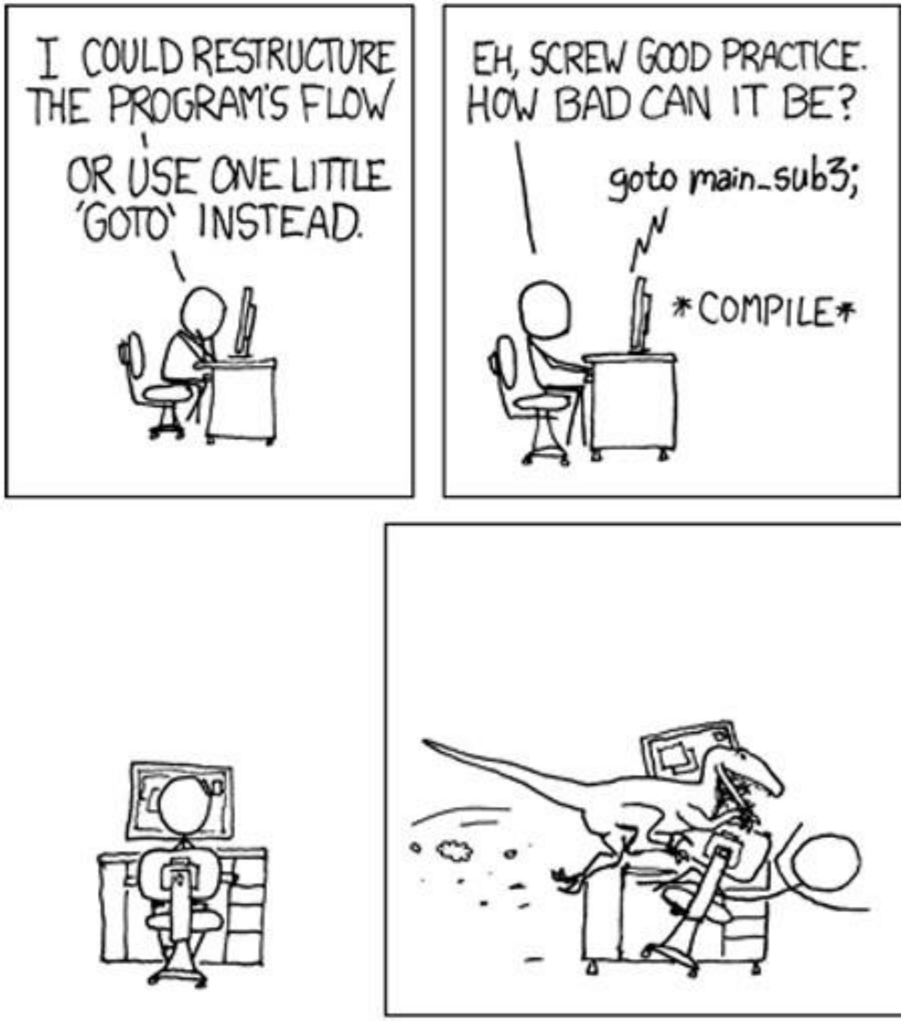
Structured Programming

- CPU executes instructions one after another
- However, sometimes we need to execute an instruction other than the next instruction
- For example:
  - Conditional instruction (if)
  - A loop (while, for)
  - A sub-routine or function

```
void main ()
{
   int a = 2;
   int c = 3;
   if (a == 8)
      c = 6;
   else
      c = 7;
   c = a + 3;
}
```

Jump and Call

- Program counter (PC) increases automatically after an instruction
- Two exceptions:
    - [Jump]: used for loop and condition
    - [Call]: used for function calls


I COULD RESTRUCTURE THE PROGRAM'S FLOW
OR USE ONE LITTLE 'GOTO' INSTEAD.
EH, SCREW GOOD PRACTICE. HOW BAD CAN IT BE?
goto main_sub3;
COMPILE

Jump

- [Jump] changes the PC and causes the CPU to execute an instruction at a target location assigned by a `label`

```
[Jump] label
```

```
        LDI R20, 0b00000001
        LDI R21, 0b00000000
        OUT DDRD, R20
L1:     OUT PORTD, R20
        CALL DELAY
        OUT PORTD, R21
        CALL DELAY
        JMP L1
```

- Two kinds of [Jump]:
  - Unconditional: the program jumps anyway
  - Conditional: the program jumps if the condition is true; otherwise, it executes the next instruction

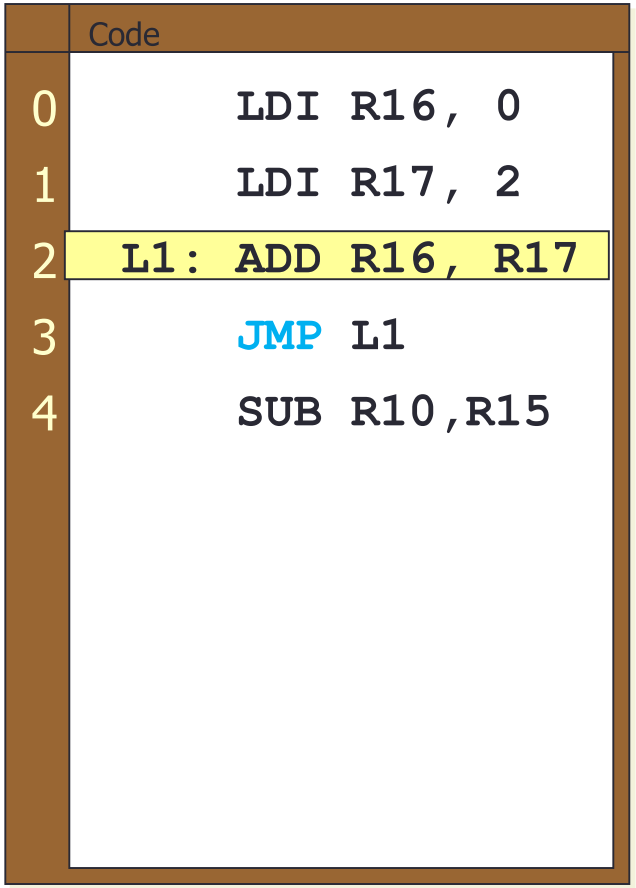
Unconditional Jump

- Three unconditional jump instructions in AVR:
  - **JMP** – jump

    PC = operand
  - **RJMP** – relative jump

    PC = PC + operand
  - **IJMP** – indirect jump

    PC = Z register

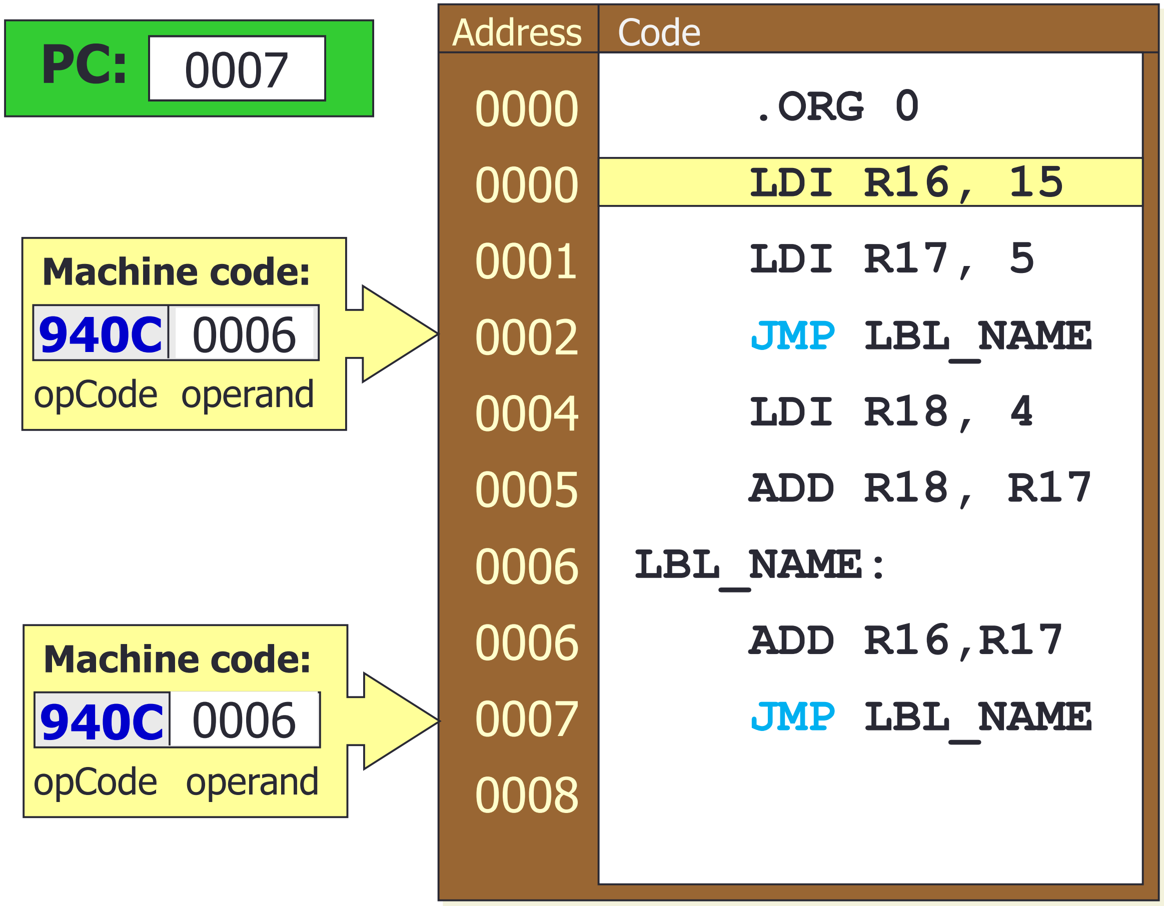


Code

```
0         LDI R16, 0
1         LDI R17, 2
2   L1:   ADD R16, R17
3         JMP L1
4         SUB R10,R15
```

Jump – JMP

- In JMP, the operand contains the ‘absolute’ address of the destination
- A 4-byte instruction, with 22 bits for address
- JMP allows a memory address from $000000 to $3FFFFF



PC: 0007

Machine code:

940C 0006

opCode operand

Machine code:

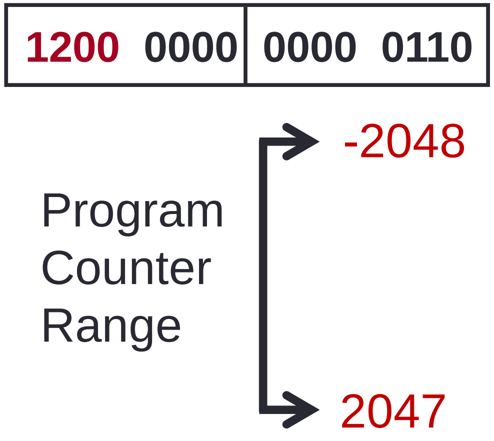
940C 0006

opCode operand

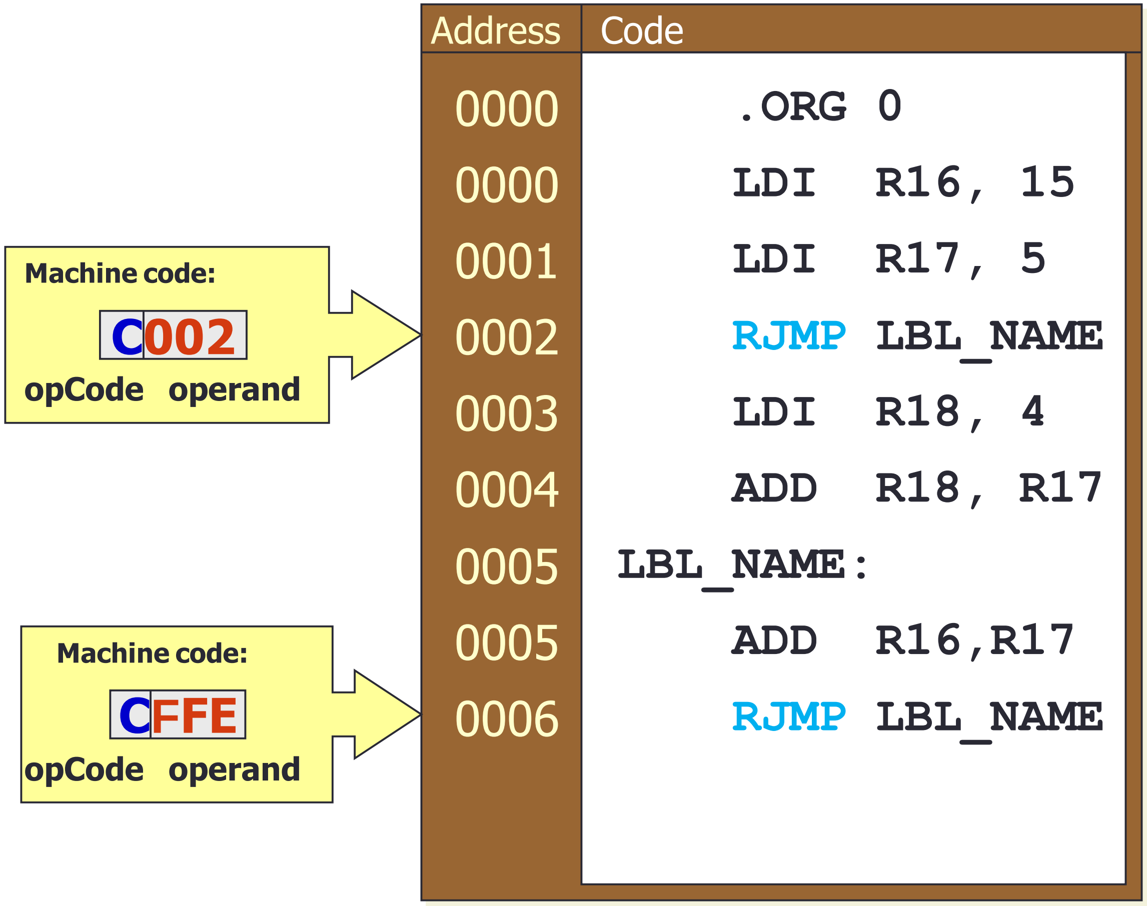
Address	Code
0000	.ORG 0
0000	LDI R16, 15
0001	LDI R17, 5
0002	JMP LBL_NAME
0004	LDI R18, 4
0005	ADD R18, R17
0006	LBL_NAME:
0006	ADD R16,R17
0007	JMP LBL_NAME
0008	

Relative Jump – RJMP

- In RJMP, the operand contains the ‘relative’ address of the destination
- A 2-byte instruction, with 12 bits for address



1200 0000	0000 0110

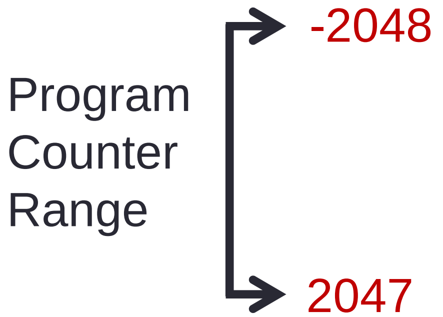


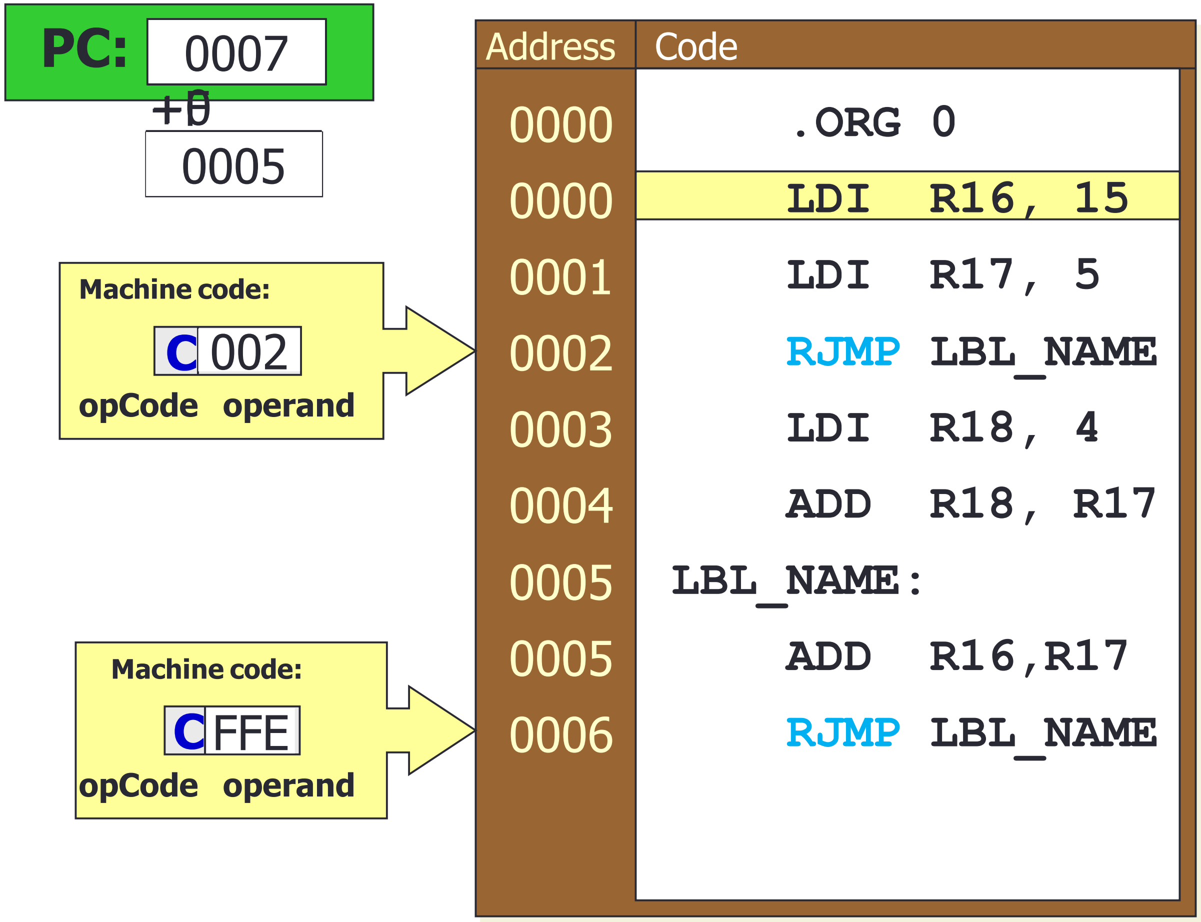
Address	Code
0000	.ORG 0
0000	LDI R16, 15
0001	LDI R17, 5
0002	RJMP LBL_NAME
0003	LDI R18, 4
0004	ADD R18, R17
0005	LBL_NAME:
0005	ADD R16,R17
0006	RJMP LBL_NAME

Relative Jump – RJMP

- In RJMP, the operand contains the 'relative' address of the destination
- A 2-byte instruction, with 12 bits for address

1200 0000	0000 0110

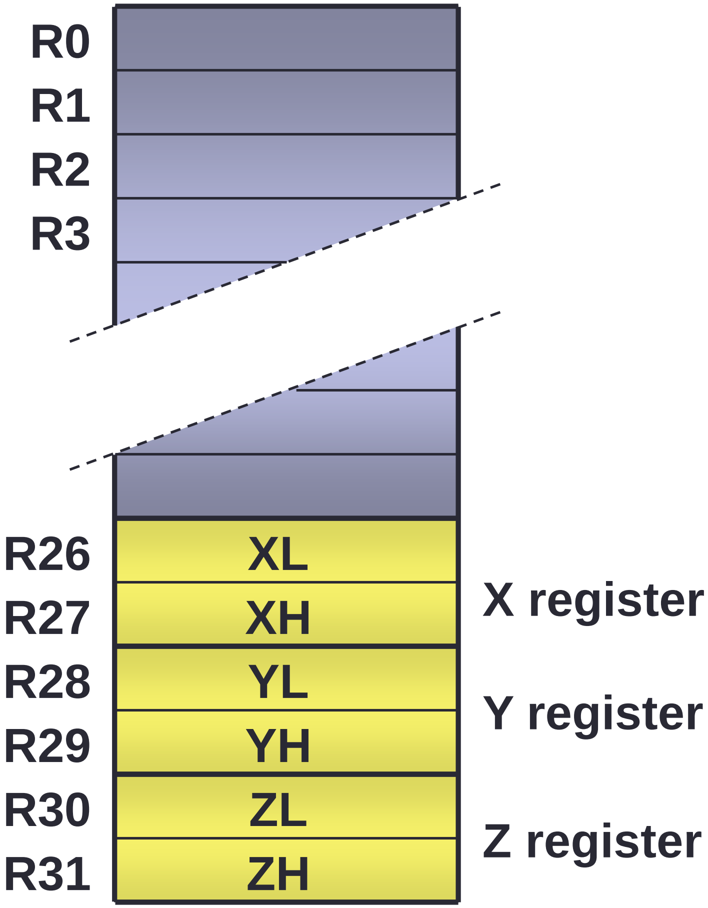




Address	Code
0000	.ORG 0
0000	LDI R16, 15
0001	LDI R17, 5
0002	RJMP LBL_NAME
0003	LDI R18, 4
0004	ADD R18, R17
0005	LBL_NAME:
0005	ADD R16,R17
0006	RJMP LBL_NAME

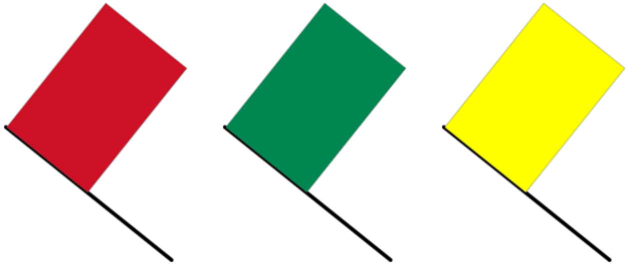
Indirect Jump – `IJMP`

- The instruction `IJMP` has no operand
- the program counter is loaded with the contents of Z register
- For example, if Z points to location $0100, by executing IJMP, the CPU jumps to location $0100



Conditional Jump

- Usage: branching and looping
- Examples in C language:
  - if-then-else
  - for
  - while
  - switch
- Jump is performed when a flag in the status register is at a specific value



Review of Status Register (SREG)

[Arithmetic: ADD or SUB]

Half Carry	H	Set if an Add/Sub. has Carry between Bits 4&3
Signed Flag	S	Used for Signed Tests
Overflow Flag	V	Set if an Add/Sub Results in Signed Overflow
Neagative Flag	N	Set if a Result is Negative
Zero Flag	Z	**Set if a Add/Subtract result is Zero**
Carry Flag	C	**Set if an Add/Subtract has Carry**

BREQ and BRNE

Instruction	Abbreviation of	Comment
BREQ	Branch if Equal	Jump if Z == 1
BRNE	Branch if Not Equal	Jump if Z == 0

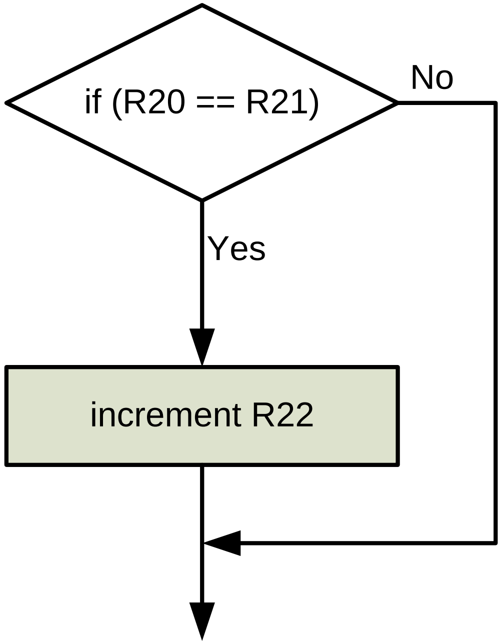
```
        [Arithmetic: ADD or SUB]





        BREQ L1
        OUT PORTD, R20
L1:     OUT PORTD, R21
```

Example 1: `if(X==Y)`

	R20==R21	R20≠R21
Z		
Jump		

- Write a program that

1. Increases the value of R26, if R20 is equal to R21
2. Otherwise do nothing



- Solution:

```
        SUB R20, R21  ;Z==1 if R20 == R21
        BRNE L1       ;jump to L1 if Z==0
        INC R26
   L1:
        ...
```

Z==0

BRCS and BRCC

Instruction	Abbreviation of	Comment
BREQ	Branch if Equal	Jump if Z == 1
BRNE	Branch if Not Equal	Jump if Z == 0
BRCS	Branch if Carry Set	Jump if C == 1
BRCC	Branch if Carry Cleared	Jump if C == 0

```
      [Arithmetic: ADD or SUB]




      BRCS L1
      OUT PORTD, R20
L1:   OUT PORTD, R21
```

Example 2: if(X<Y)

	R26<R24	R26≥R24
C		
Jump		

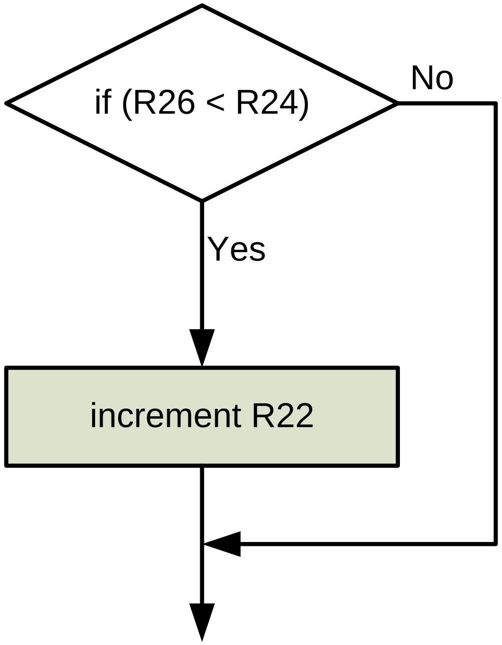
- Write a program that

1. Increases the value of R22 if R26 < R24
2. Otherwise do nothing

- Solution:

```
        SUB R26, R24  ;C==1 if R26 < R24
        BRCC L1       ;jump to L1 if C==0
        INC R22
   L1:
        ...
```

C==0



Example 3: if(X>=Y)

	R26<R24	R26≥R24
C		
Jump		

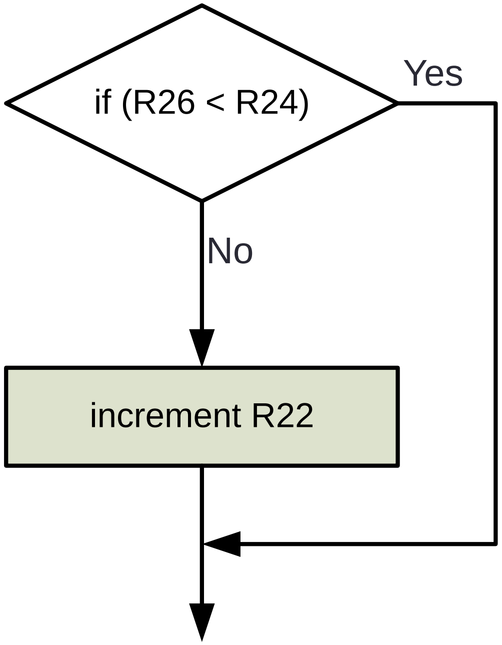
- Write a program that

1. Increases the value of R22 if R26 ≥ R24
2. Otherwise do nothing

- Solution:

```
        SUB R26, R24   ;C==0 if R26 >= R24
        BRCS L1        ;jump to L1 if C==1
        INC R22
L1:
        ...
```

C==1



Example 4: `if/else`

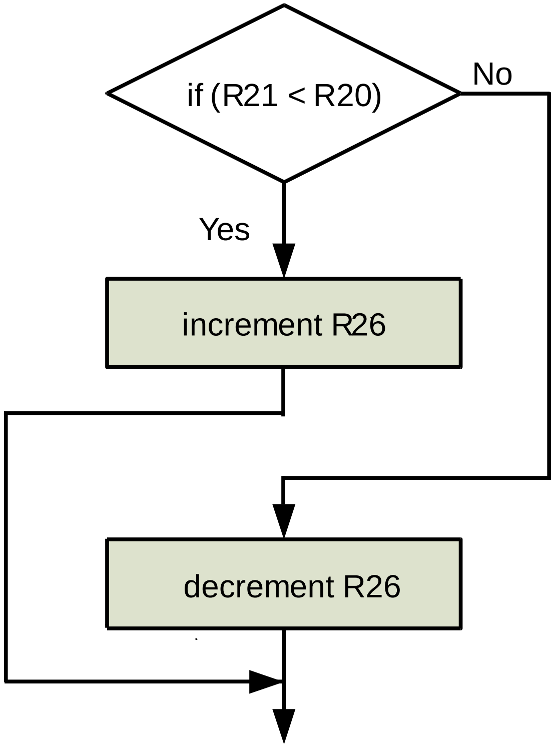
	R21<R20	R21≥R20
C		
Jump		

- Re-write this into assembly:

```
if(R20 > R21)
     R26++;
else
     R26--;
```

```
       SUB  R21, R20
       BRCC L1
       INC  R26
       JMP  L2
L1:    DEC  R26
L2:
```

C==0



Example 5: `for`

	R16==0	R16≠0
Z		
Jump		

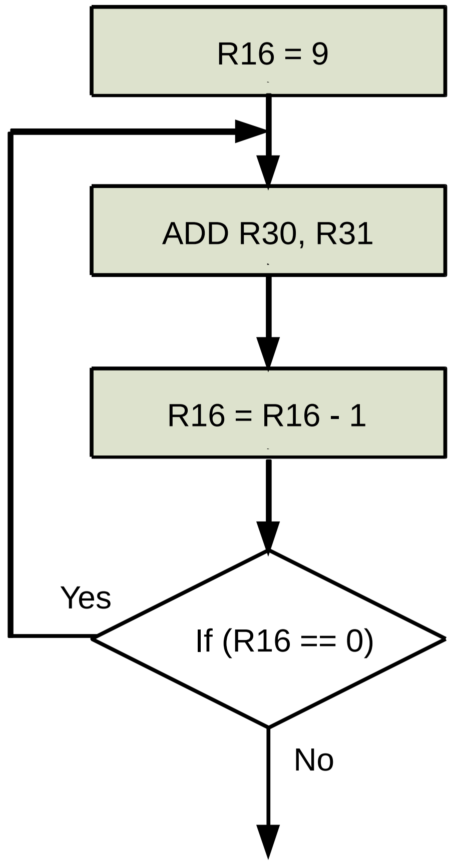
- Write a program that executes the instruction

  `ADD R30, R31`

  for 9 times

- Solution:

```
      LDI  R16, 9
L1:   ADD  R30, R31
      DEC  R16
      BRNE L1      ;if Z==0
      ...
```

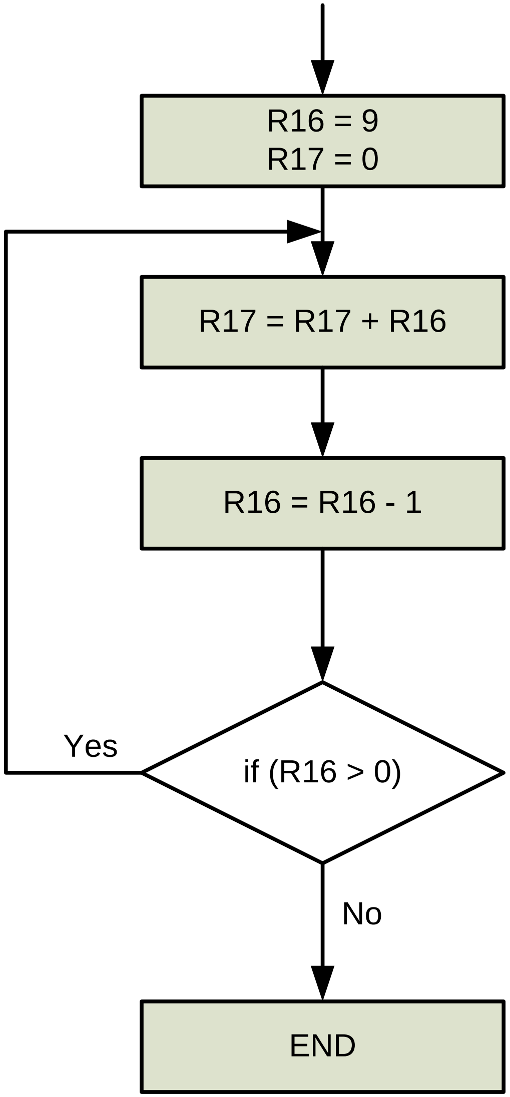


Example 6: for/while

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

- Solution:

```
        LDI   R16, 9
        LDI   R17, 0
L1:     ADD   R17, R16
        DEC   R16
        BRNE  L1          ;if Z==0
L2:     RJMP  L2          ;wait here
```



R16 = 9
R17 = 0

R17 = R17 + R16

R16 = R16 - 1

if (R16 > 0)

Yes

No

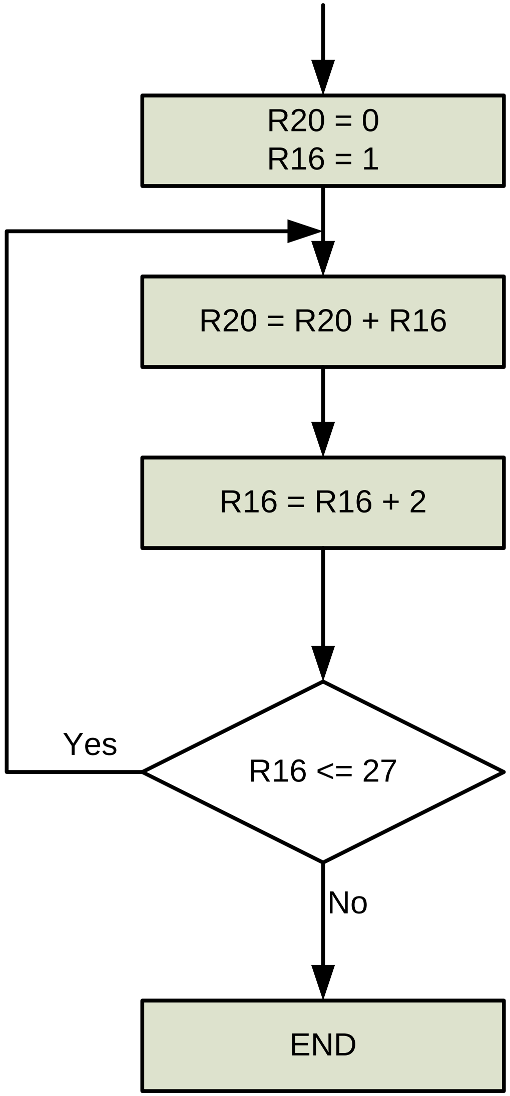
END

Example 7: `for/while`

- 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
     LDI R17, 27
     SUB R17, R16
     BRCC L1       ;jump if R16<=27
```



Example 8: switch

```
M_LOOP: ..
      CPI ch, 65   ;compare
      BREQ L1     ;branch if eq
      CPI ch, 66
      BREQ L2
      ...
      JMP EXIT
L1:   ...
      JMP EXIT
L2:   ...
      JMP EXIT
EXIT: ...
```

```
switch (ch)
{
  case 65: (L1)
    ...
    break;
  case 66: (L2)
    ...
    break;
    ...
}
...
```

Note: CP can only compare the values of two registers

Summary of Conditional Jump for `if`

BREQ
BRNE
BRCS
BRCC

- Increases the value of R1 only if R20 is equal to R21

```
if(R20 == R21)
    R1++;
```

- Increases the value of R1 only if R20 is smaller than R21

```
if(R20 < R21)
    R1++;
```

http://www.youtube.com/watch?v=PIVRKyRzfW4

Conditional Jump in AVR

Instruction	Abbreviation of	Comment
BREQ	Branch if Equal	Jump if Z = 1
BRNE	Branch if Not Equal	Jump if Z = 0
BRCS	Branch if Carry Set	Jump if C = 1
BRCC	Branch if Carry Cleared	Jump if C = 0
BRMI	Branch if Minus	Jump if N = 1
BRPL	Branch if Plus	Jump if N = 0
BRGE	Branch if Greater or Equal	Jump if S = 0
BRLT	Branch if Less Than	Jump if S = 1
BRHS	Branch if Half Carry Set	Jump if H = 1
BRHC	Branch if Half Carry Cleared	Jump if H = 0
BRTS	Branch if T flag Set	Jump if T = 1
BRTC	Branch if T flag Cleared	Jump if T = 0
BRIS	Branch if I flag set	Jump if I = 1
BRIC	Branch if I flag cleared	Jump if I = 0

Outline (Cont’d)

- Structured programming
  - Jump
  - Calling a function
- Getting started

Calling A Function

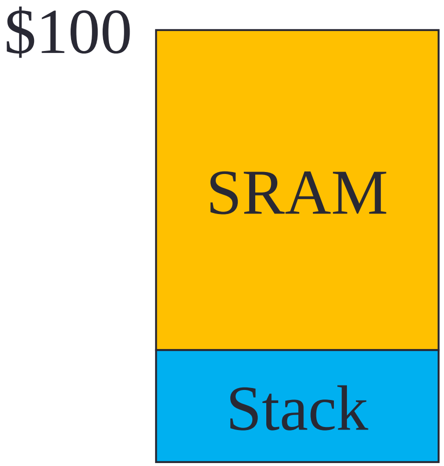
- A function is “called”
- The PC changes to the label being “called”
- The PC changes back at the end of the function
- What is missing here?

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
0008	INC R20
0009	L1: RJMP L1
000A	FUNC: ADD R20, R21
000B	SUBI R20, 3
000C	RET

Stack

- A section of RAM for temporarily storing PC
- Procedure:

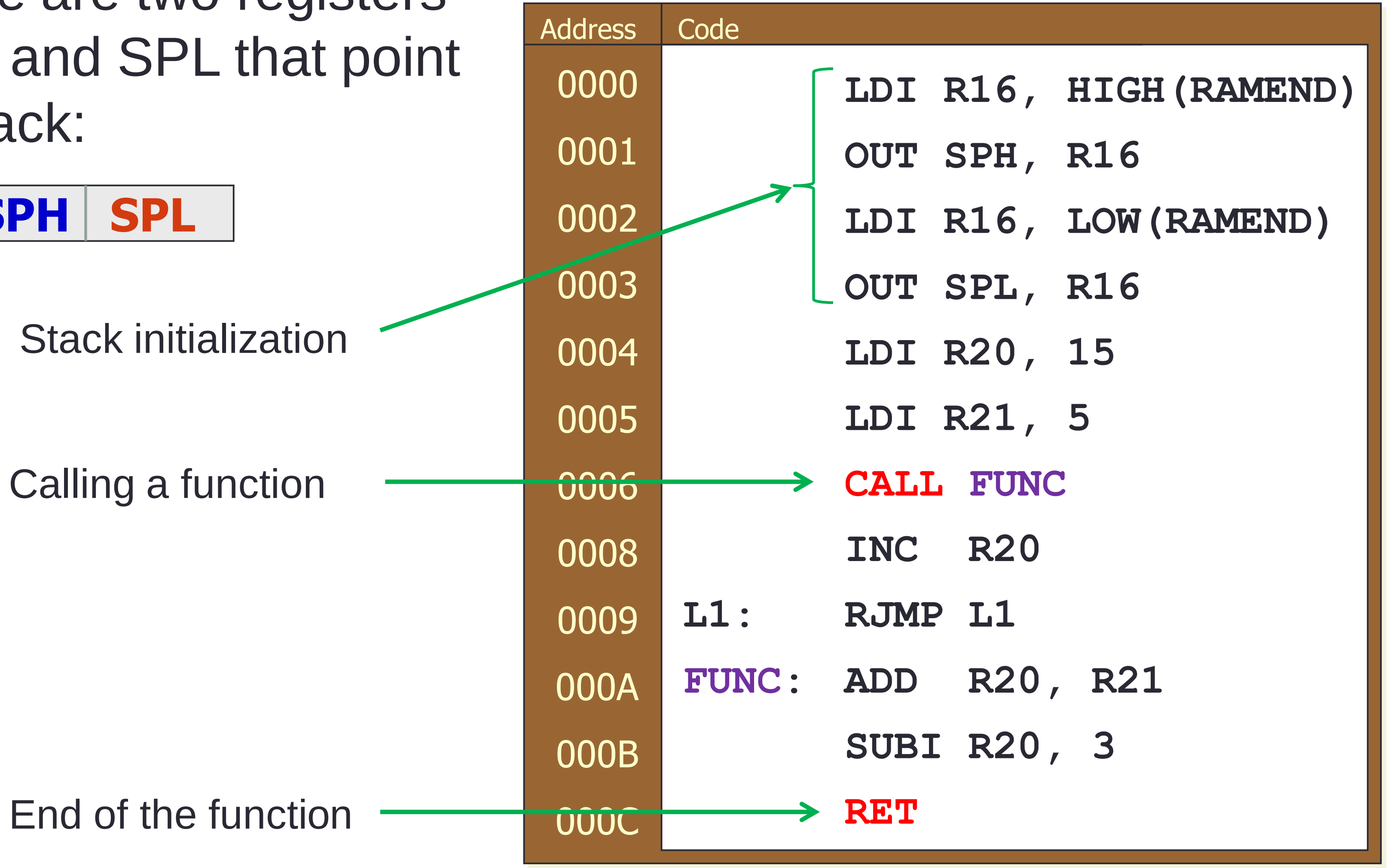
1. Save the address of instruction right below the **CALL** instruction on 'stack'
2. Change the PC to where the function to be called
3. When reaching the end of the function (**RET**), retrieve the address from 'stack'



Initializing Stack Pointer

- There are two registers SPH and SPL that point to stack:

SPH	SPL



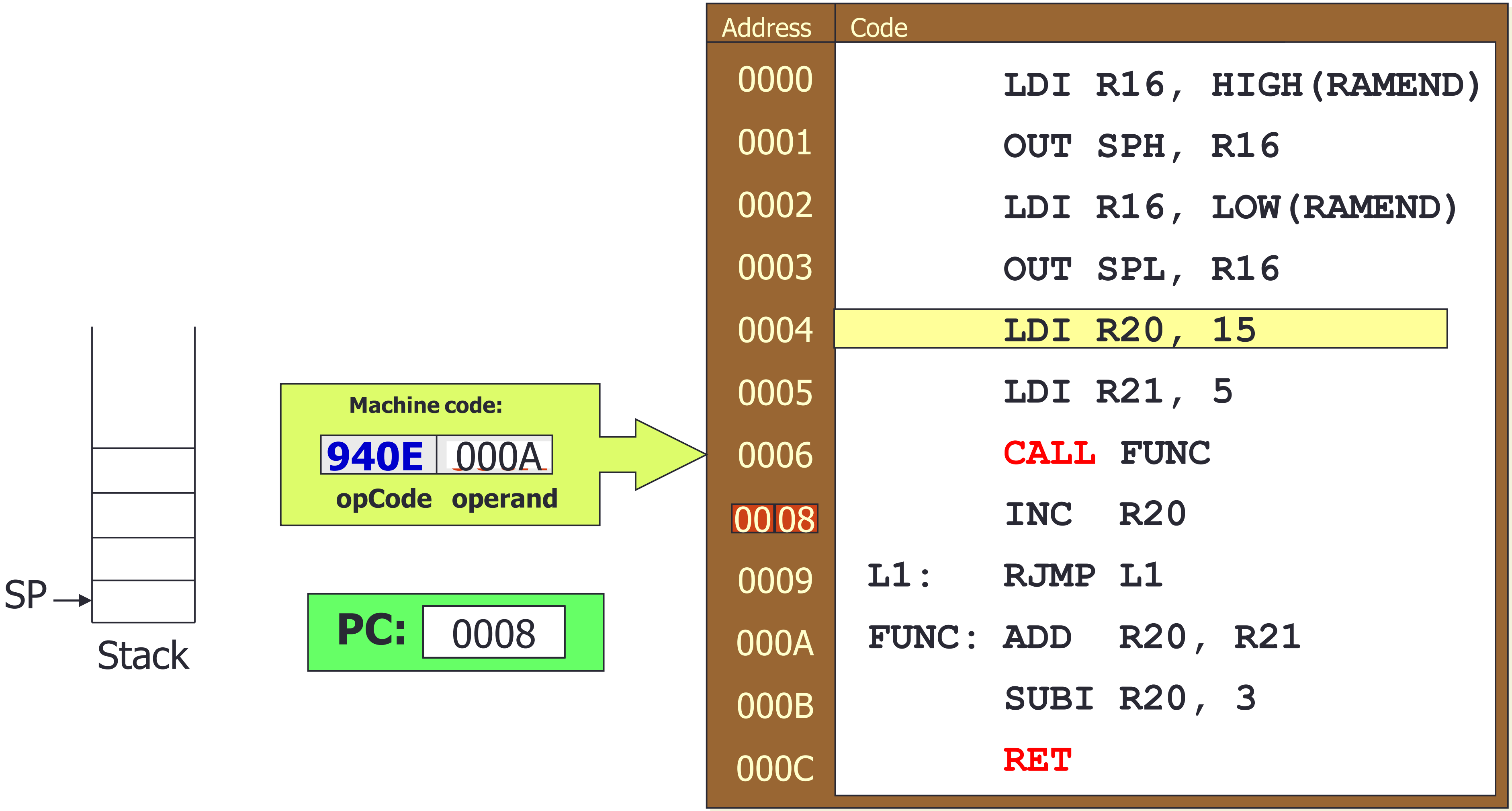
Stack initialization

Calling a function

End of the function

```
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
0008                 INC  R20
0009     L1:         RJMP L1
000A     FUNC:       ADD  R20, R21
000B                 SUBI R20, 3
000C                 RET
```

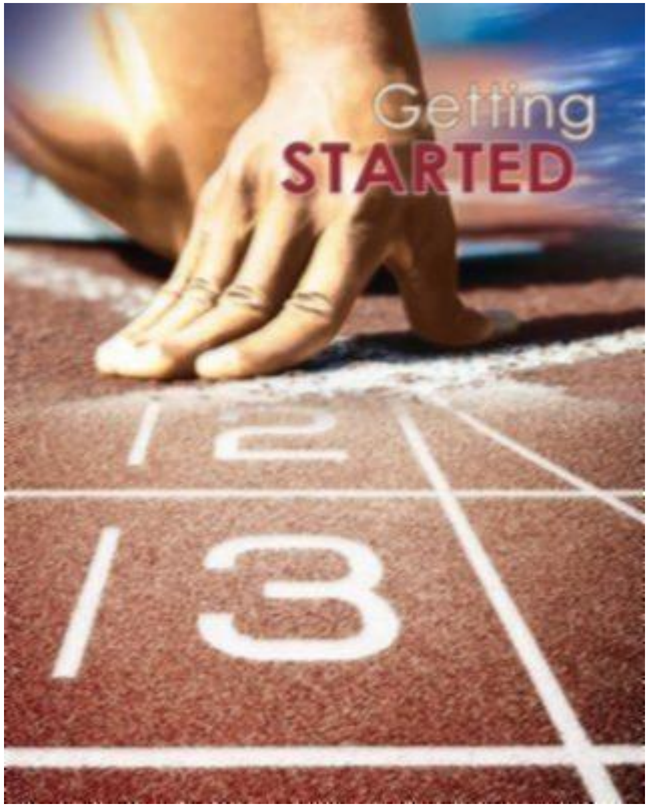
Calling a Function



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
0008	INC R20
0009	L1: RJMP L1
000A	FUNC: ADD R20, R21
000B	SUBI R20, 3
000C	RET

Outline (Cont'd)

- Structured programming
  - Jump
  - Calling a function
- Getting started



Reference

- ATmega328P datasheet
- AVR 8-bit instruction set
- AVR1022: assembler user guide
- M. A. Mazidi, S. Naimi, and S. Naimi, *The AVR Microcontroller and Embedded Systems: Using Assembly and C*, Prentice Hall, 2010
- AVR GCC library help http://nongnu.org/avr-libc/user-manual/modules.html