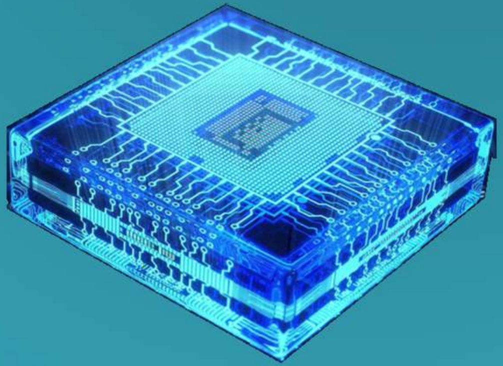
Microprocessors and Assembly language

Isfahan University of Technology (IUT)



Jump And Call

Dr. Hamidreza Hakim
hakim@iut.ac.ir

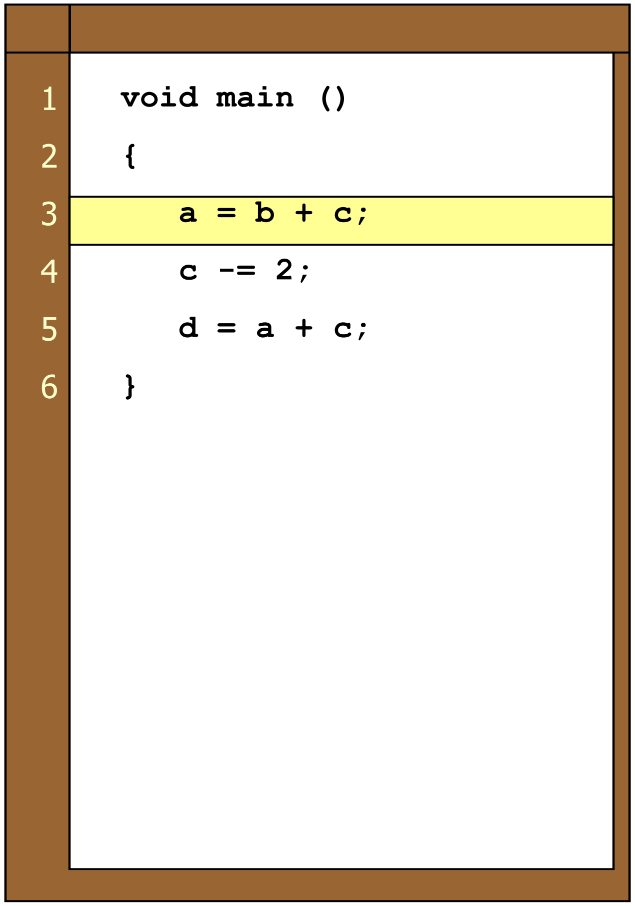
بسم الله الرحمن الرحیم

Topics

- Introduction to jump and call
- Jump
- Call
  - Stack
  - Calling a function
- Time Delay

Jump and Call

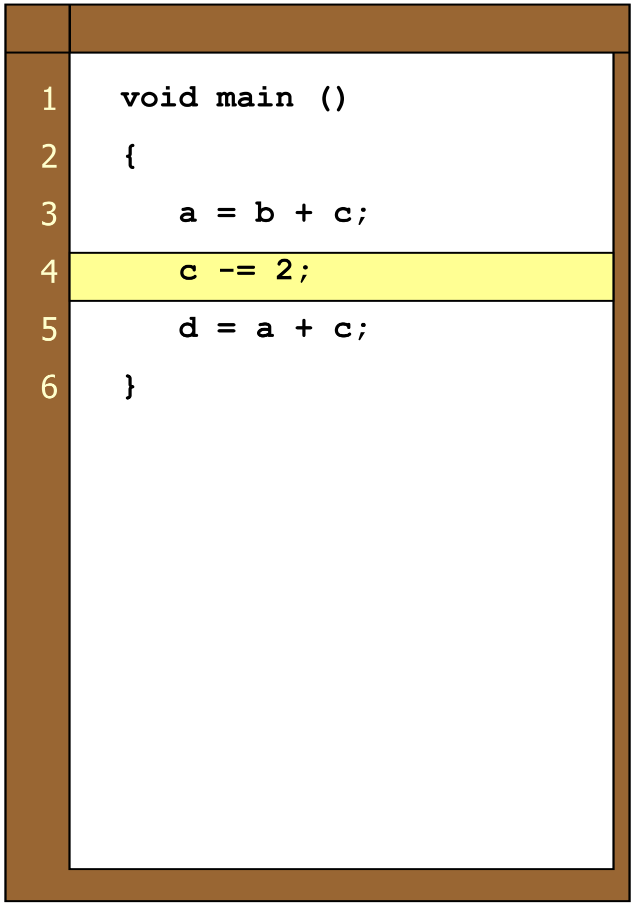
- CPU executes instructions one after another.
  - For example in the following C program, CPU first executes the instruction of line 3 (adds b and c), then
    executes the instruction of line 4.



```
1   void main ()
2   {
3       a = b + c;
4       c -= 2;
5       d = a + c;
6   }
```

Jump and Call

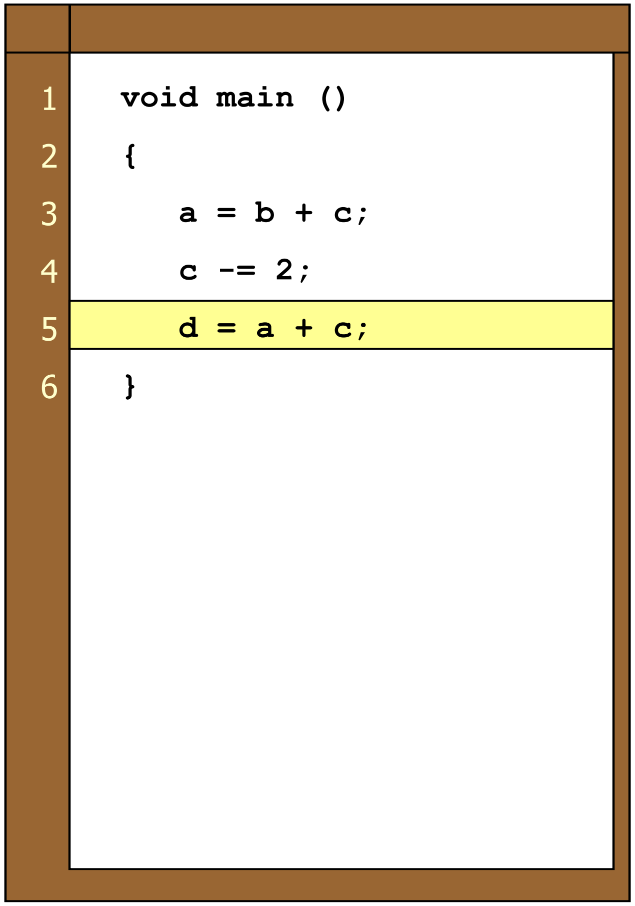
- CPU executes instructions one after another.
  - For example in the following C program, CPU first executes the instruction of line 3 (adds b and c), then
    executes the instruction of line 4.



```
1   void main ()
2   {
3      a = b + c;
4      c -= 2;
5      d = a + c;
6   }
```

Jump and Call

- CPU executes instructions one after another.
  - For example in the following C program, CPU first executes the instruction of line 3 (adds b and c), then executes the instruction of line 4.



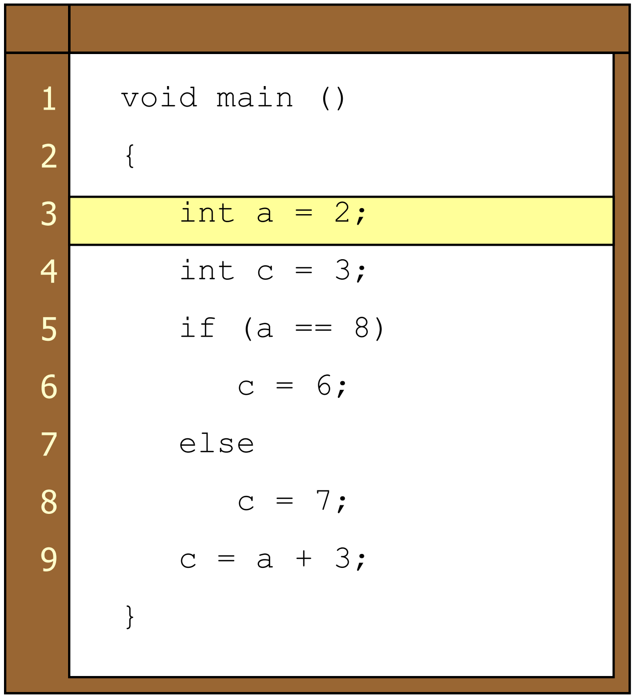
```
1   void main ()
2   {
3       a = b + c;
4       c -= 2;
5       d = a + c;
6   }
```

Jump and Call (Continued)

- But sometimes we need the CPU to execute, an instruction other than the next instruction. For example:
  - When we use a conditional instruction (if)
  - When we make a loop
  - When we call a function

Jump and Call (Continued)

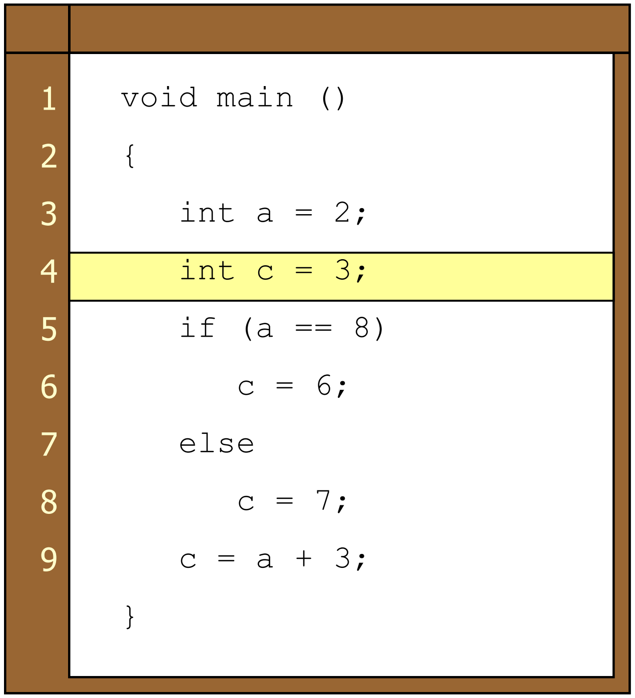
- Example 1: Not executing the next instruction, because of condition.
  - In the following example, the instruction of line 6 is not executed.



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

Jump and Call (Continued)

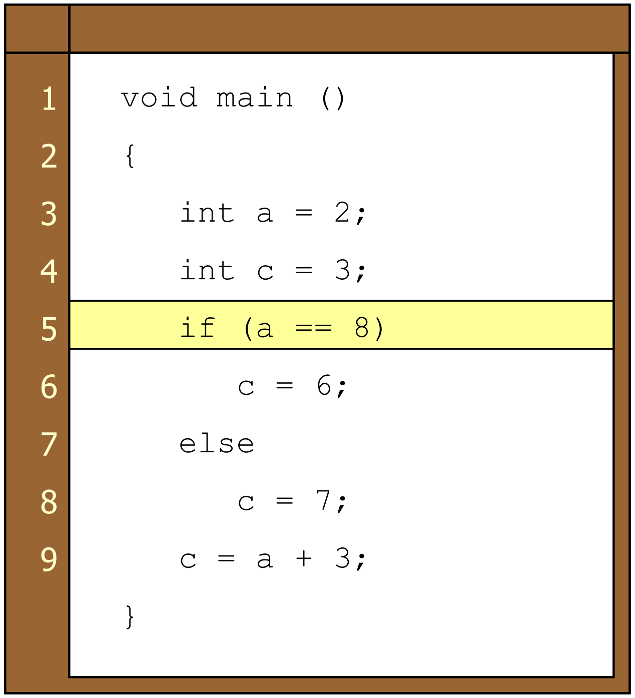
- Example 1: Not executing the next instruction, because of condition.
  - In the following example, the instruction of line 6 is not executed.



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

Jump and Call (Continued)

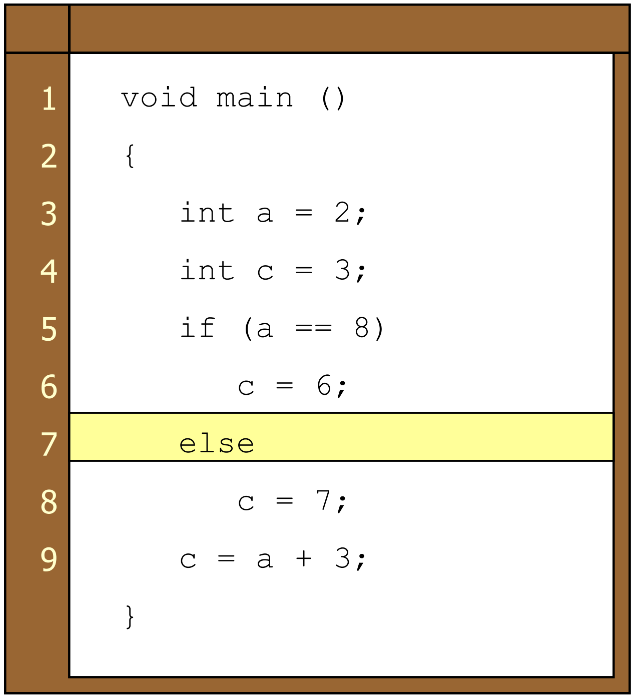
- Example 1: Not executing the next instruction, because of condition.
  - In the following example, the instruction of line 6 is not executed.



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

Jump and Call (Continued)

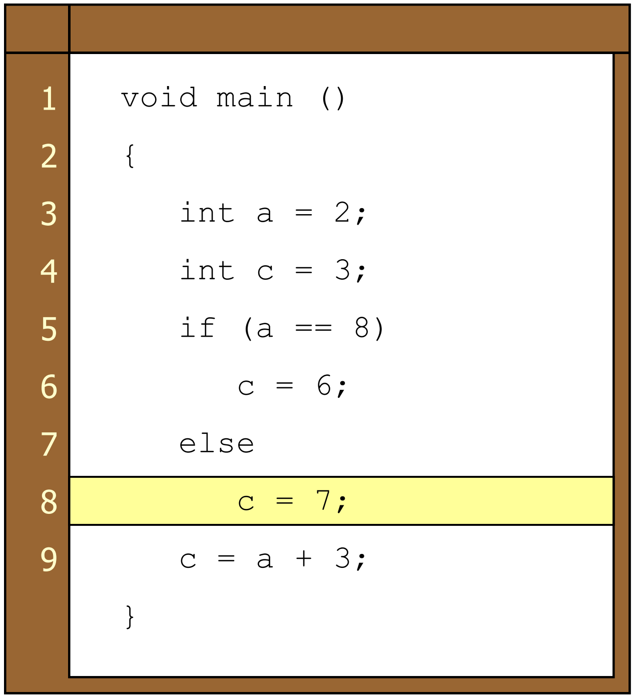
- Example 1: Not executing the next instruction, because of condition.
  - In the following example, the instruction of line 6 is not executed.



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

Jump and Call (Continued)

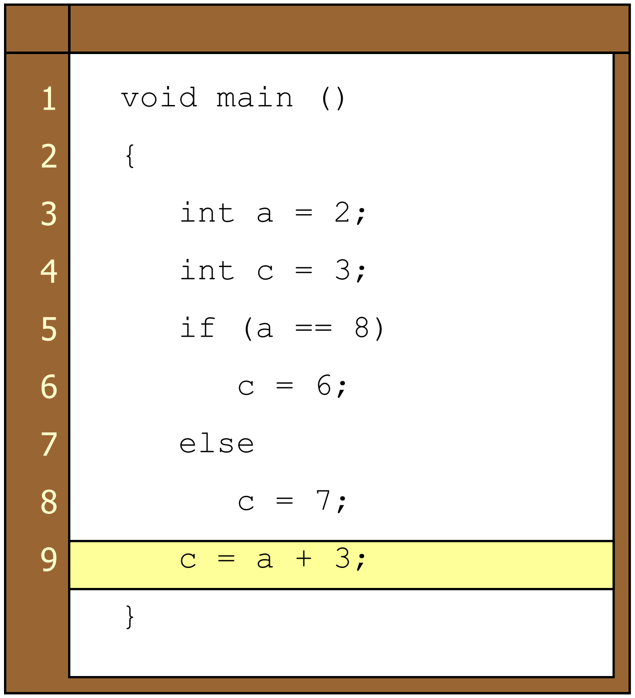
- Example 1: Not executing the next instruction, because of condition.
  - In the following example, the instruction of line 6 is not executed.



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

Jump and Call (Continued)

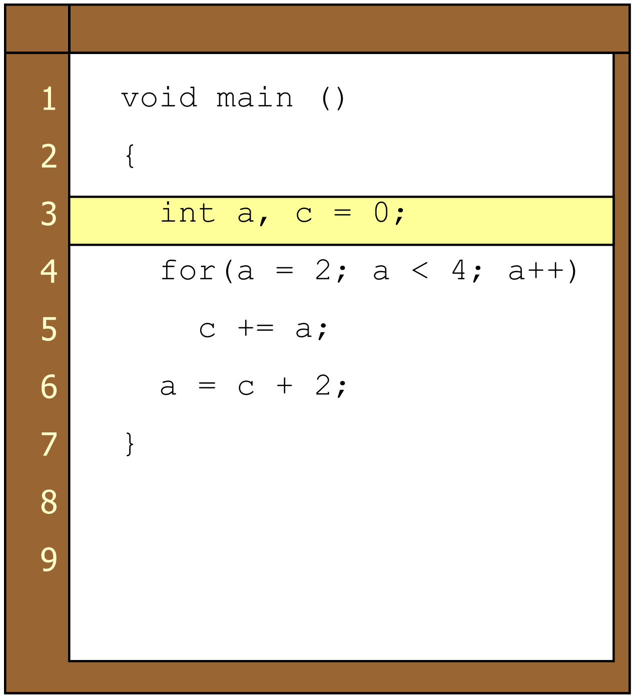
- Example 1: Not executing the next instruction, because of condition.
  - In the following example, the instruction of line 6 is not executed.



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

Jump and Call (Continued)

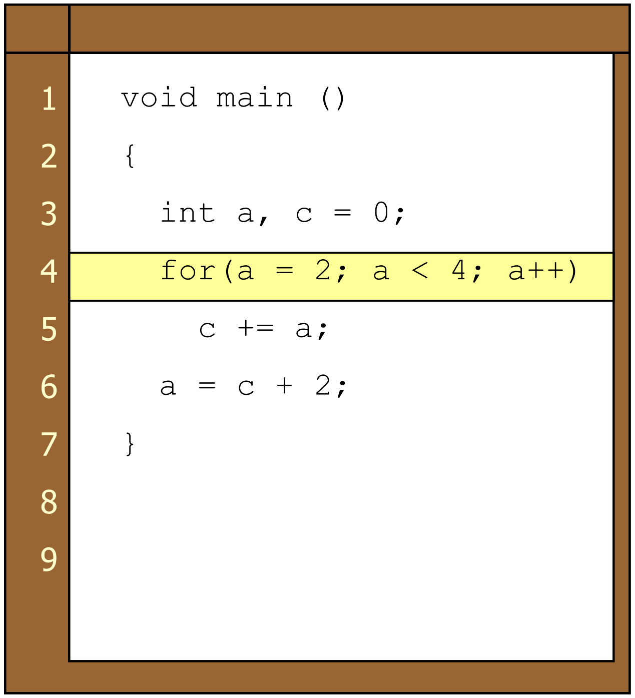
- Example 2: In this example the next instruction will not be executed because of loop.
  - In the following example, the order of execution is as follows:
    - Line 4
    - Line 5
    - Again, line 4
    - Again line 5
    - Line 6



```
1  void main ()
2  {
3    int a, c = 0;
4    for(a = 2; a < 4; a++)
5      c += a;
6    a = c + 2;
7  }
8
9
```

Jump and Call (Continued)

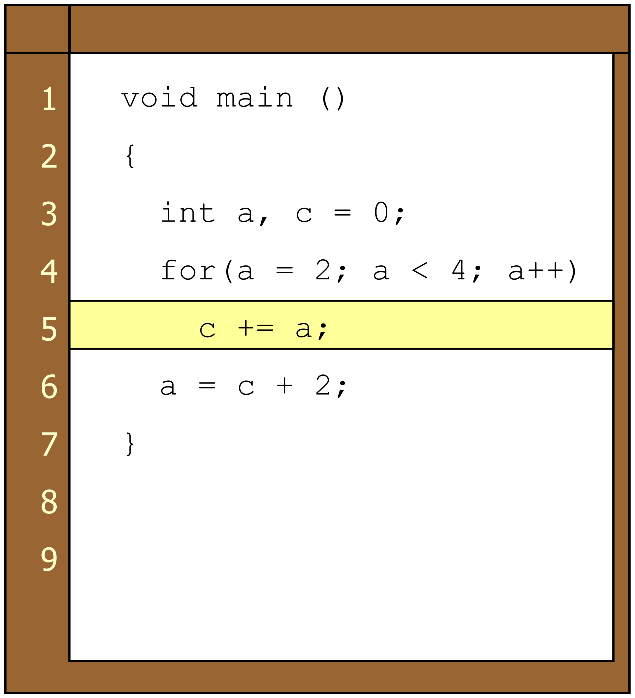
- Example 2: In this example the next instruction will not be executed because of loop.
  - In the following example, the order of execution is as follows:
    - Line 4
    - Line 5
    - Again, line 4
    - Again line 5
    - Line 6



```
1  void main ()
2  {
3    int a, c = 0;
4    for(a = 2; a < 4; a++)
5      c += a;
6    a = c + 2;
7  }
8
9
```

Jump and Call (Continued)

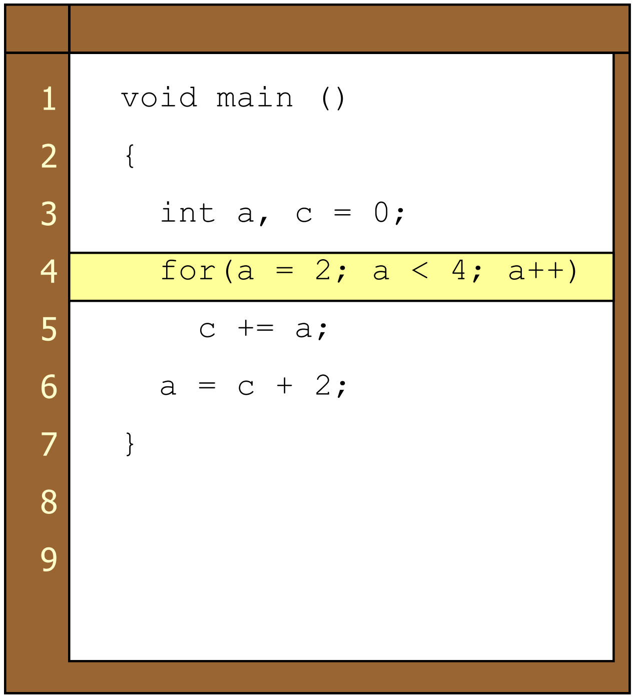
- Example 2: In this example the next instruction will not be executed because of loop.
  - In the following example, the order of execution is as follows:
    - Line 4
    - Line 5
    - Again, line 4
    - Again line 5
    - Line 6



```
1   void main ()
2   {
3     int a, c = 0;
4     for(a = 2; a < 4; a++)
5       c += a;
6     a = c + 2;
7   }
8
9
```

Jump and Call (Continued)

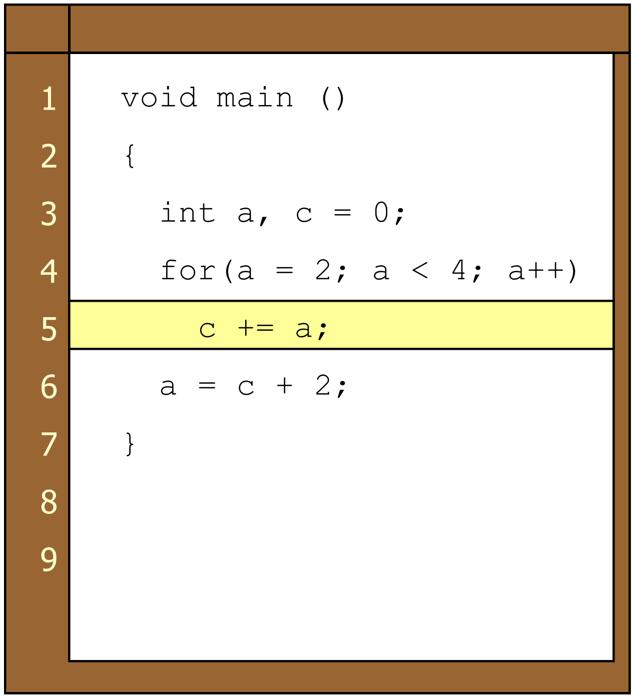
- Example 2: In this example the next instruction will not be executed because of loop.
  - In the following example, the order of execution is as follows:
    - Line 4
    - Line 5
    - Again, line 4
    - Again line 5
    - Line 6



```
1  void main ()
2  {
3    int a, c = 0;
4    for(a = 2; a < 4; a++)
5      c += a;
6    a = c + 2;
7  }
8
9
```

Jump and Call (Continued)

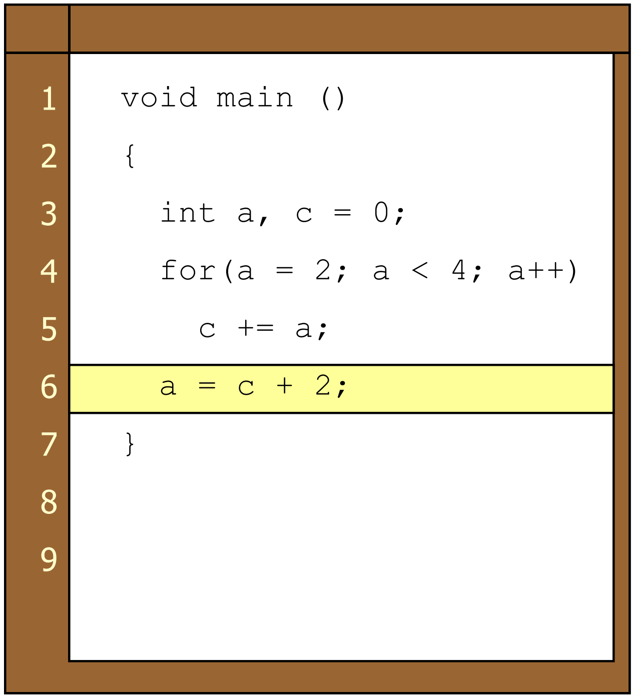
- Example 2: In this example the next instruction will not be executed because of loop.
  - In the following example, the order of execution is as follows:
    - Line 4
    - Line 5
    - Again, line 4
    - Again line 5
    - Line 6



```
1  void main ()
2  {
3    int a, c = 0;
4    for(a = 2; a < 4; a++)
5      c += a;
6    a = c + 2;
7  }
8
9
```

Jump and Call (Continued)

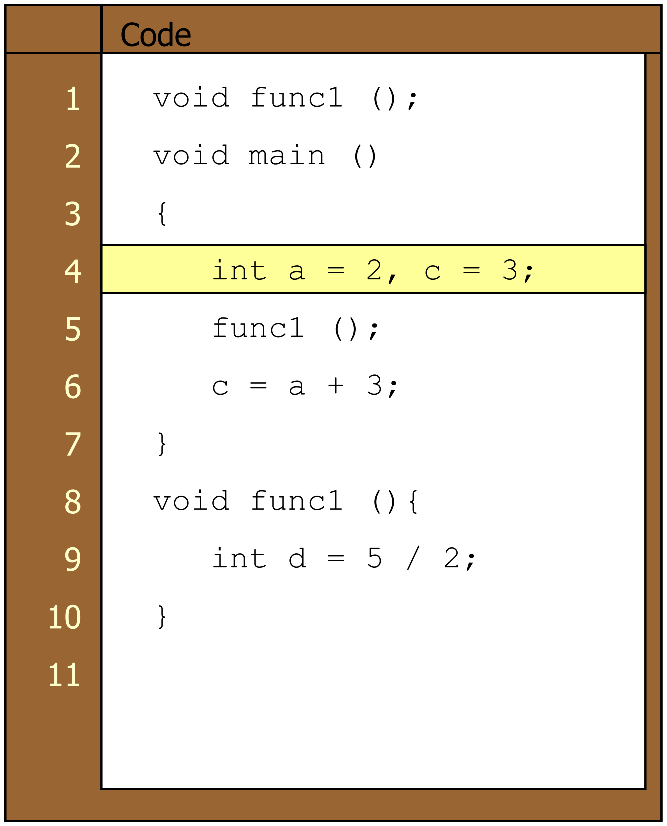
- Example 2: In this example the next instruction will not be executed because of loop.
  - In the following example, the order of execution is as follows:
    - Line 4
    - Line 5
    - Again, line 4
    - Again line 5
    - Line 6



```
1  void main ()
2  {
3    int a, c = 0;
4    for(a = 2; a < 4; a++)
5      c += a;
6    a = c + 2;
7  }
8
9
```

Jump and Call (Continued)

- Example 3: Not executing the next instruction, because of calling a function.
  - In the following example, the instruction of line 6 is not executed after line 5.



Code

```
1   void func1 ();
2   void main ()
3   {
4      int a = 2, c = 3;
5      func1 ();
6      c = a + 3;
7   }
8   void func1 (){
9      int d = 5 / 2;
10  }
11
```

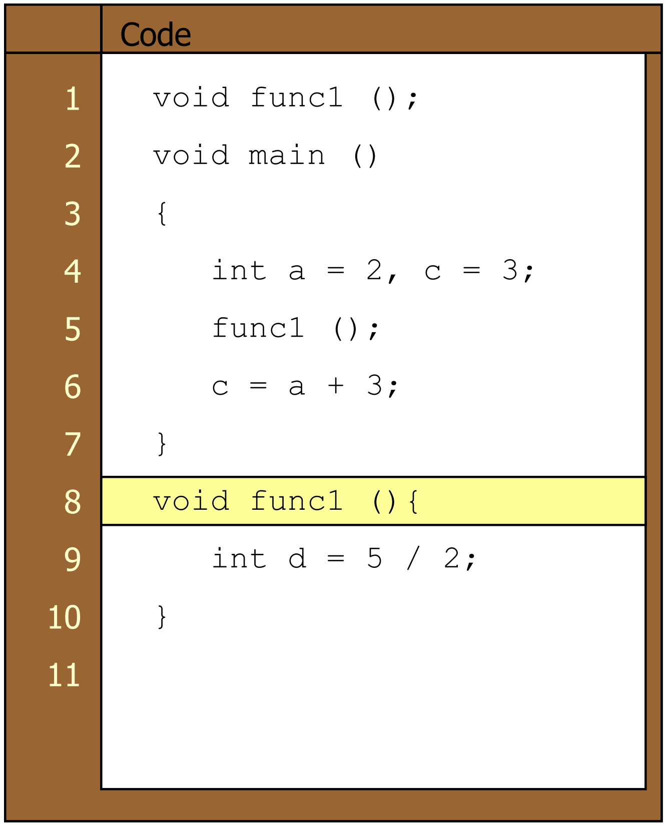
Jump and Call (Continued)

- Example 3: Not executing the next instruction, because of calling a function.
  - In the following example, the instruction of line 6 is not executed after line 5.

```
     Code
 1   void func1 ();
 2   void main ()
 3   {
 4      int a = 2, c = 3;
 5      func1 ();
 6      c = a + 3;
 7   }
 8   void func1 (){
 9      int d = 5 / 2;
10   }
11
```

Jump and Call (Continued)

- Example 3: Not executing the next instruction, because of calling a function.
  - In the following example, the instruction of line 6 is not executed after line 5.

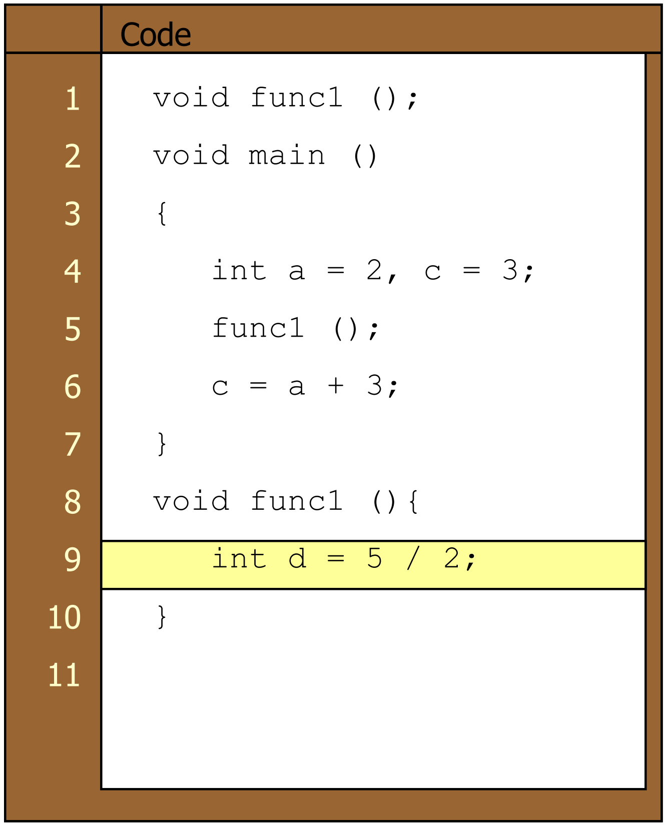


Code

```
1   void func1 ();
2   void main ()
3   {
4       int a = 2, c = 3;
5       func1 ();
6       c = a + 3;
7   }
8   void func1 (){
9       int d = 5 / 2;
10  }
11
```

Jump and Call (Continued)

- Example 3: Not executing the next instruction, because of calling a function.
  - In the following example, the instruction of line 6 is not executed after line 5.



Code

```
1   void func1 ();
2   void main ()
3   {
4      int a = 2, c = 3;
5      func1 ();
6      c = a + 3;
7   }
8   void func1 (){
9      int d = 5 / 2;
10  }
11
```

Jump and Call (Continued)

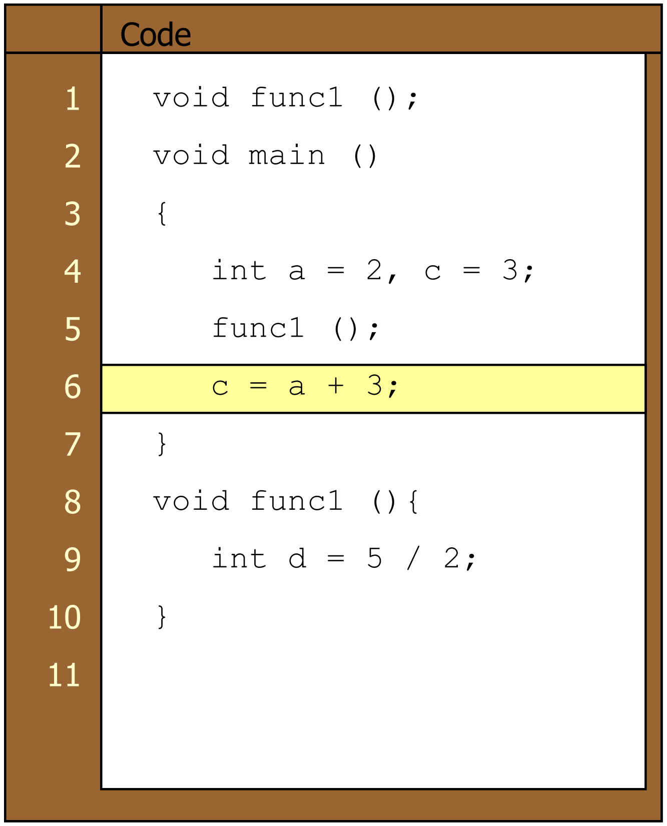
- Example 3: Not executing the next instruction, because of calling a function.
  - In the following example, the instruction of line 6 is not executed after line 5.

Code

```
1   void func1 ();
2   void main ()
3   {
4      int a = 2, c = 3;
5      func1 ();
6      c = a + 3;
7   }
8   void func1 (){
9      int d = 5 / 2;
10  }
11
```

Jump and Call (Continued)

- Example 3: Not executing the next instruction, because of calling a function.
  - In the following example, the instruction of line 6 is not executed after line 5.



Code

```
1   void func1 ();
2   void main ()
3   {
4      int a = 2, c = 3;
5      func1 ();
6      c = a + 3;
7   }
8   void func1 (){
9      int d = 5 / 2;
10  }
11
```

Jump and Call (Continued)

- In the assembly language, there are 2 groups of instructions that make the CPU execute an instruction other than the next instruction.
- The instructions are:
  - Jump: used for making loop and condition
  - Call: used for making function calls

Jump

- Jump changes the Program Counter (PC) and causes the CPU to execute an instruction other than the next instruction.

Jump

There are 2 kinds of Jump

- Unconditional Jump: When CPU executes an unconditional jump, it jumps unconditionally (without checking any condition) to the target location.
  - Example: RJMP and JMP instructions
- Conditional Jump: When CPU executes a conditional jump, it checks a condition, if the condition is true then it jumps to the target location; otherwise, it executes the next instruction.

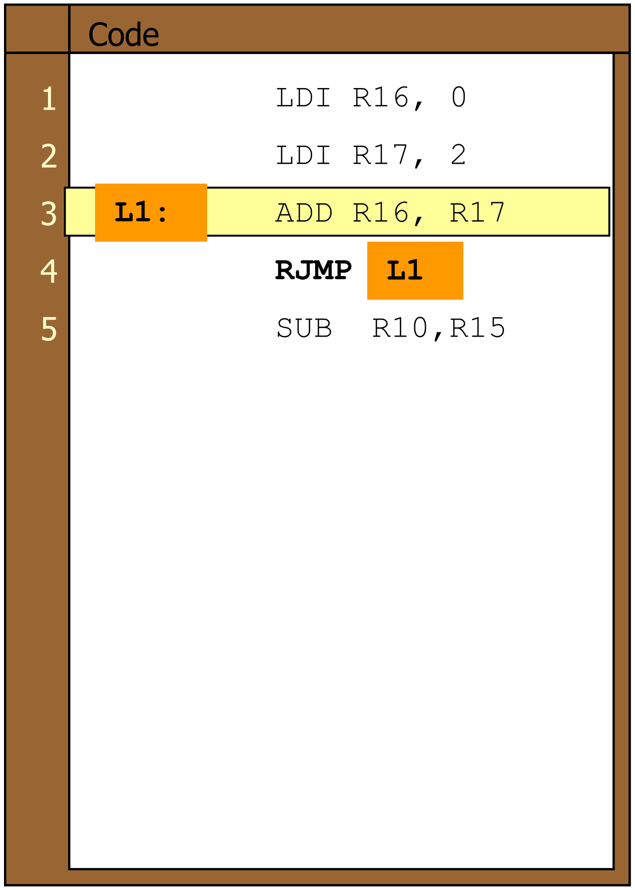
Unconditional Jump in AVR

- There are 3 unconditional jump instructions in AVR:

  JMP, **RJMP**, and **IJMP**

Note

- We label the location where we want to jump, using a unique name, followed by ‘:’
- Then, in front of the jump instruction we mention the name of the label.
- This causes the CPU to jump to the location we have labeled, instead of executing the next instruction.



```
            LDI  R16, 0
            LDI  R17, 2
L1:         ADD  R16, R17
            RJMP L1
            SUB  R10,R15
```

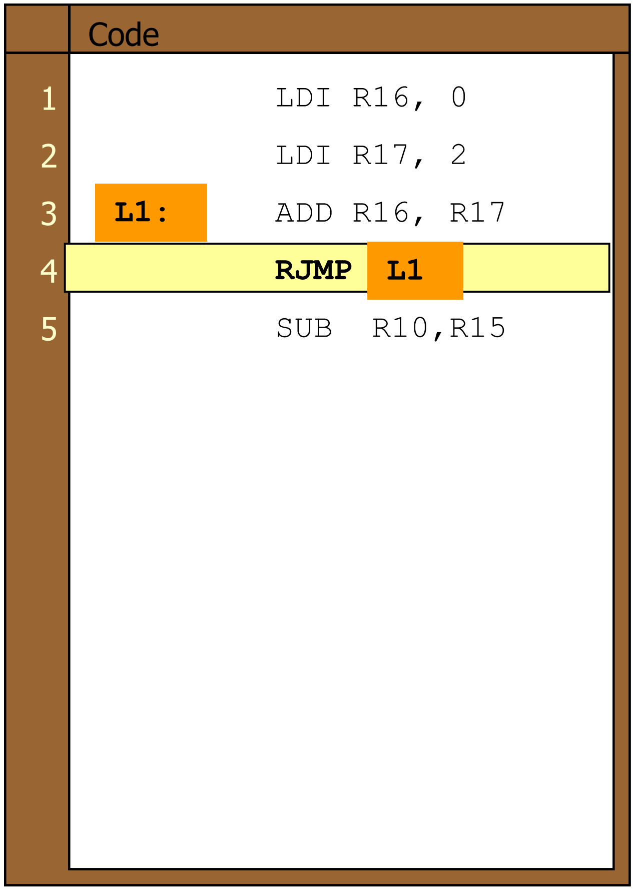
Unconditional Jump in AVR

- There are 3 unconditional jump instructions in AVR:

  JMP, **RJMP**, and **IJMP**

Note

- We label the location where we want to jump, using a unique name, followed by ':'
- Then, in front of the jump instruction we mention the name of the label.
- This causes the CPU to jump to the location we have labeled, instead of executing the next instruction.



Code

```
        LDI R16, 0
        LDI R17, 2
L1:     ADD R16, R17
        RJMP L1
        SUB  R10,R15
```

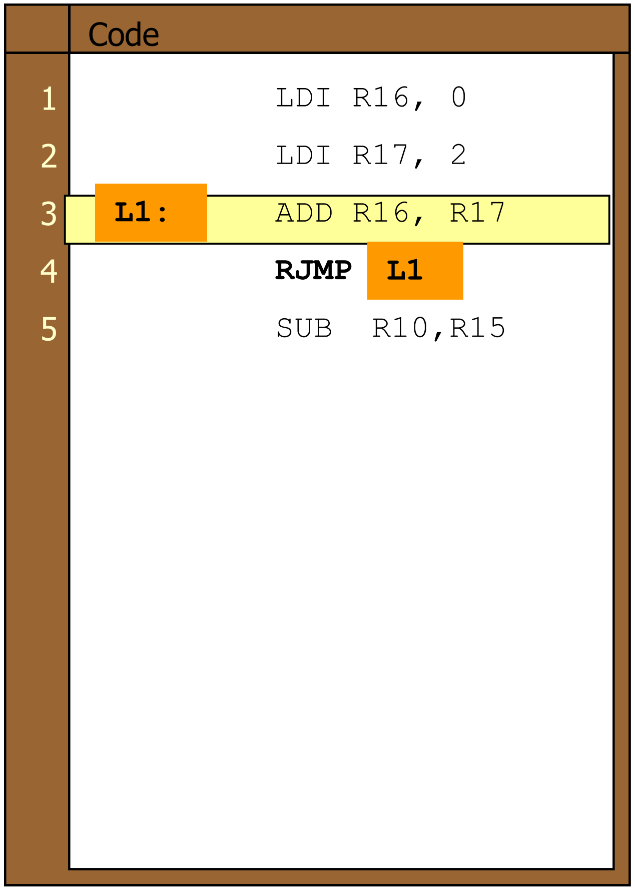
Unconditional Jump in AVR

- There are 3 unconditional jump instructions in AVR:

  JMP, **RJMP**, and **IJMP**

Note

- We label the location where we want to jump, using a unique name, followed by ‘:’
- Then, in front of the jump instruction we mention the name of the label.
- This causes the CPU to jump to the location we have labeled, instead of executing the next instruction.



Code

```
           LDI R16, 0
           LDI R17, 2
L1:        ADD R16, R17
           RJMP  L1
           SUB   R10,R15
```

Ways of specifying the jump target

- There are 3 ways to provide the jump address:
  - PC = operand
  - PC = PC + operand
  - PC = Z register

JMP

- JMP ⟷ PC = operand

1001 010X	XXXX 110X	XXXX XXXX	XXXX XXXX

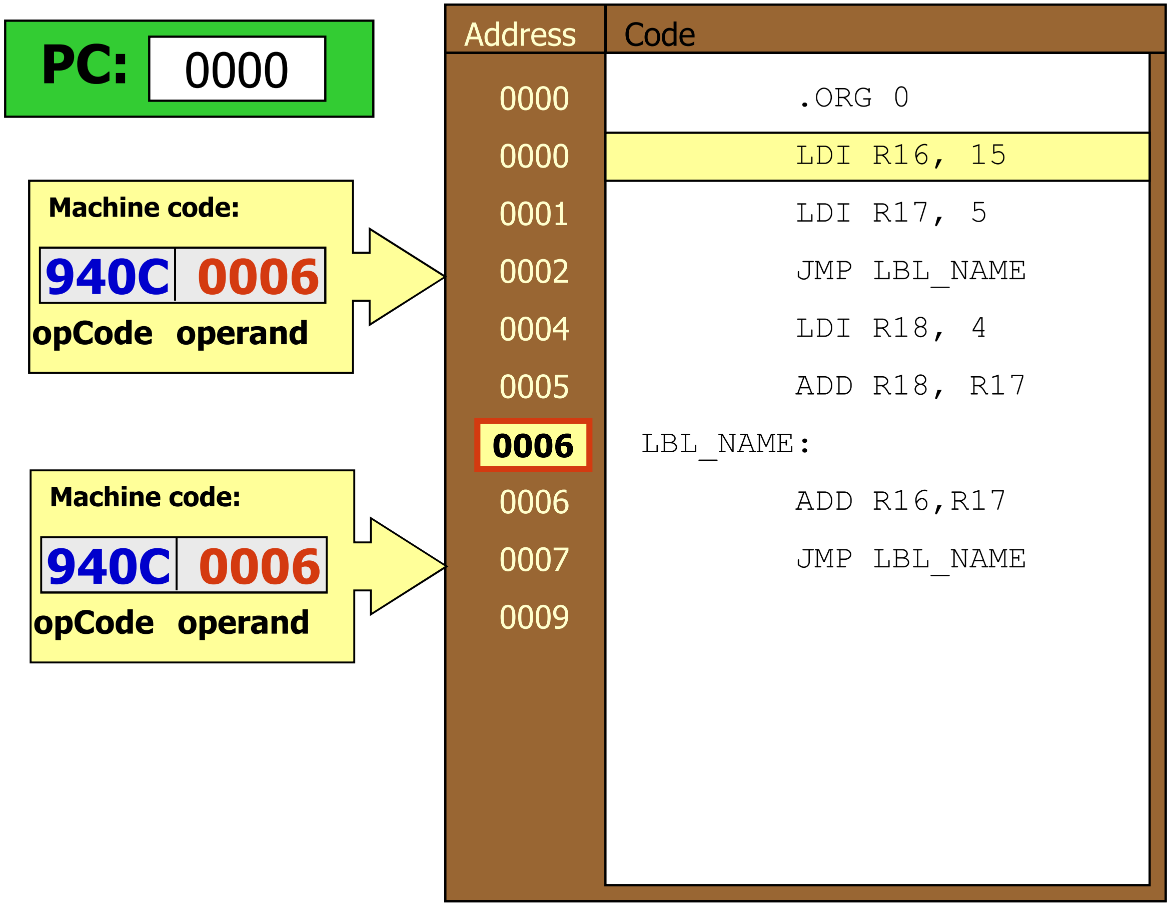
  - Example:

1001 0100	0000 1100	0000 0000	0000 0110

  - Operand = **0000000000000000000110**
  - **4 byte(why?)**

JMP

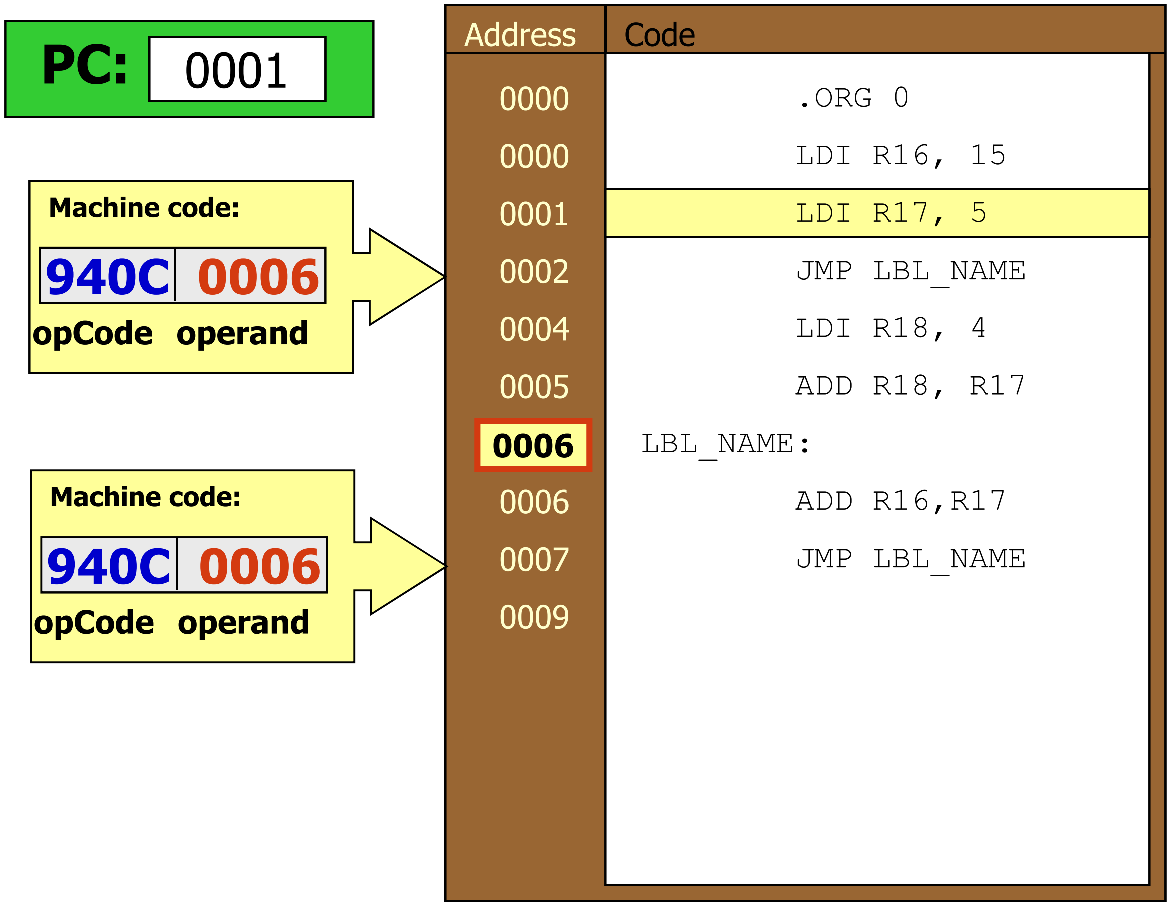
- In JMP, the operand, contains the address of the destination
- When an JMP is executed:
  - PC is loaded with the operand value



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
0009	

JMP

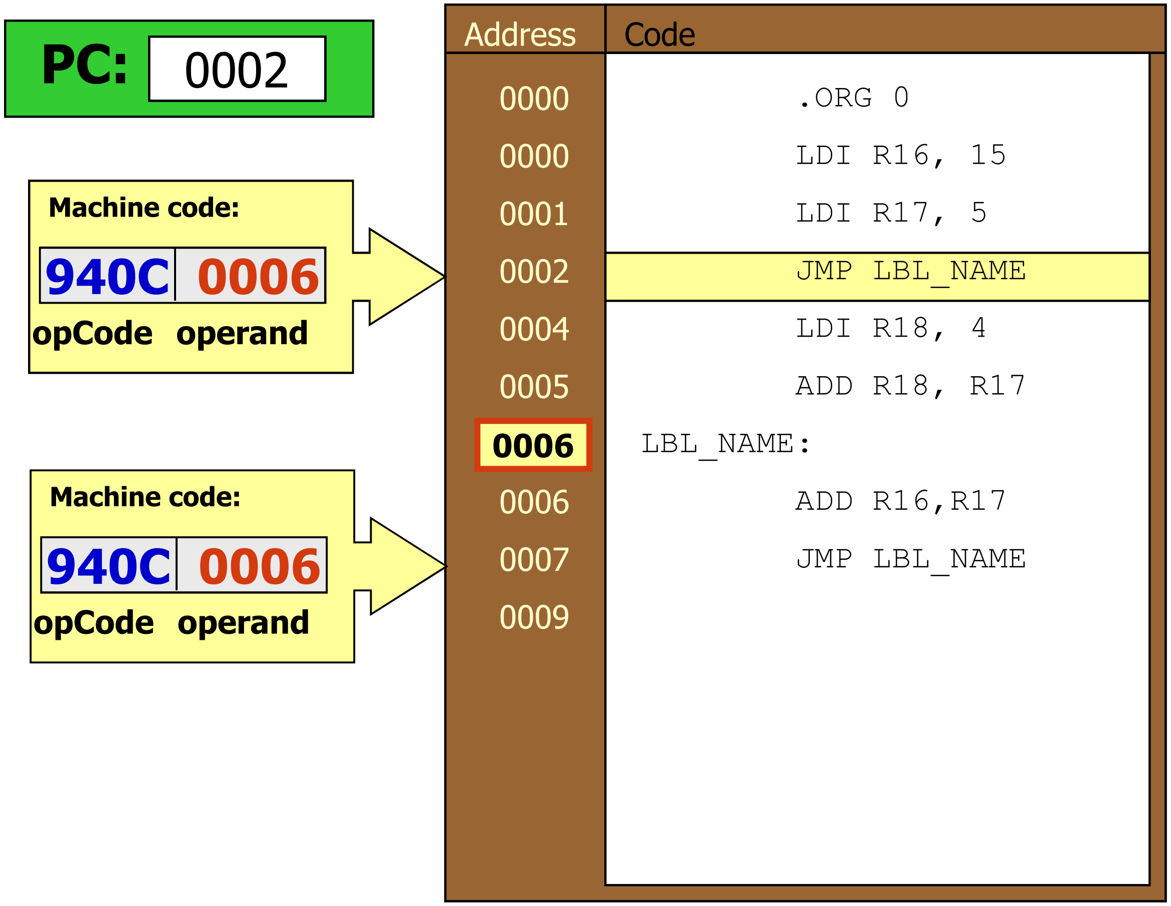
- In JMP, the operand, contains the address of the destination
- When an JMP is executed:
  - PC is loaded with the operand value



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
0009	

JMP

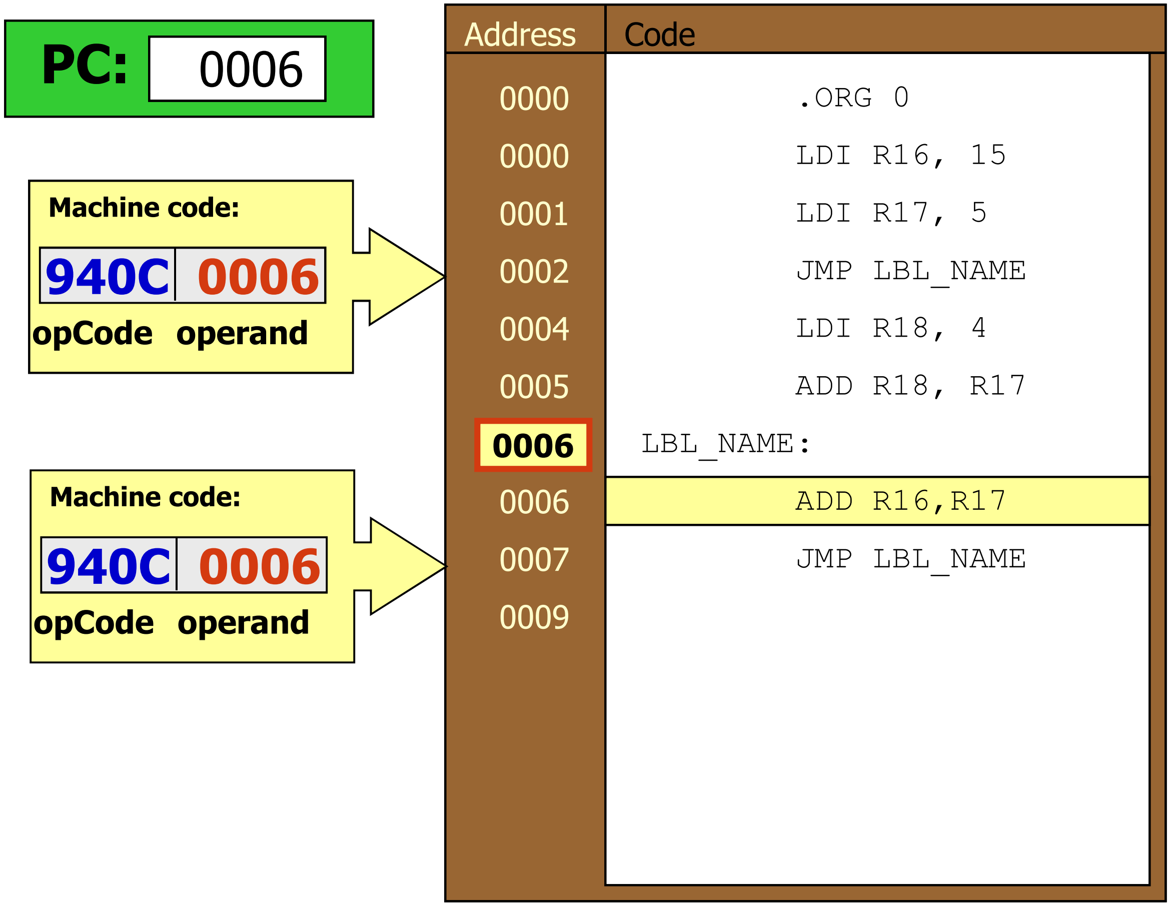
- In JMP, the operand, contains the address of the destination
- When an JMP is executed:
  - PC is loaded with the operand value



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
0009	

JMP

- In JMP, the operand, contains the address of the destination
- When an JMP is executed:
  - PC is loaded with the operand value



PC: 0006

Machine code:

940C	0006
opCode	operand

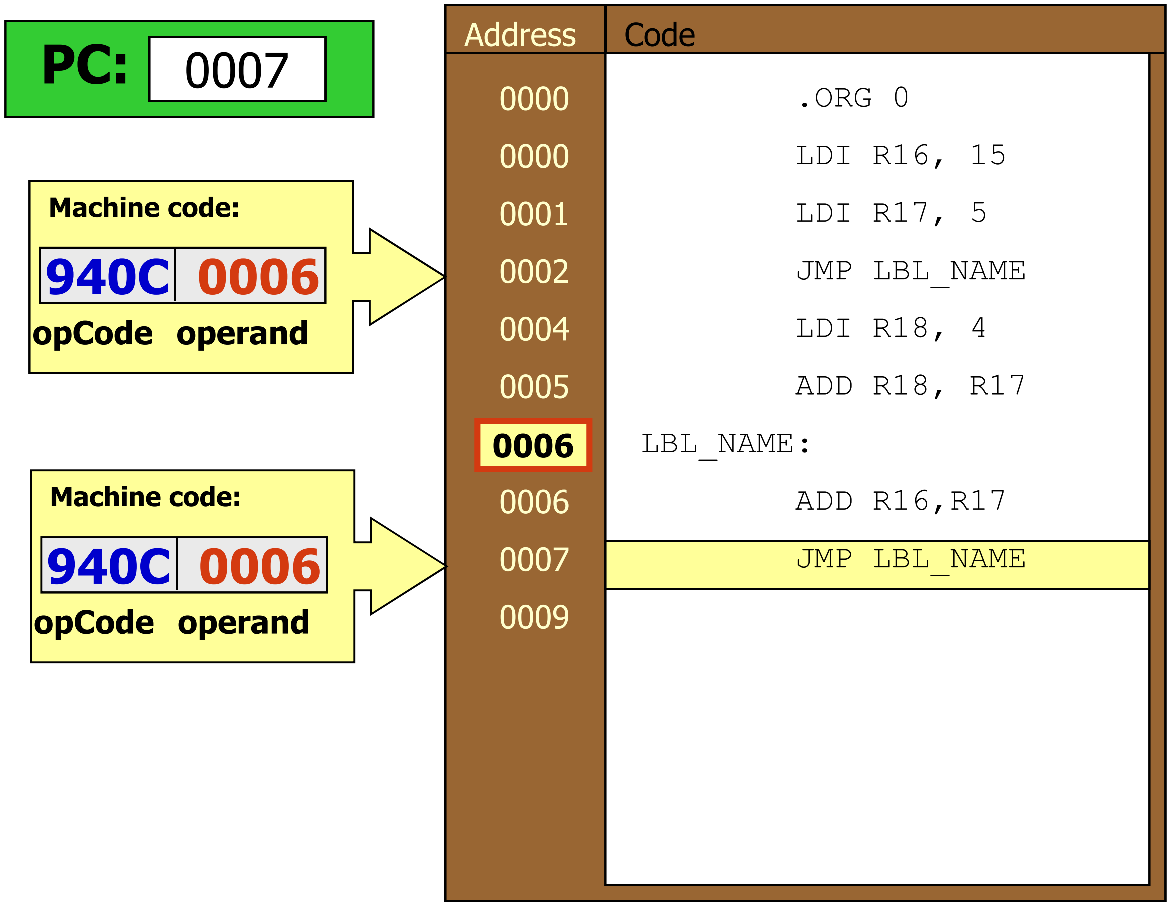
Machine code:

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
0009	

JMP

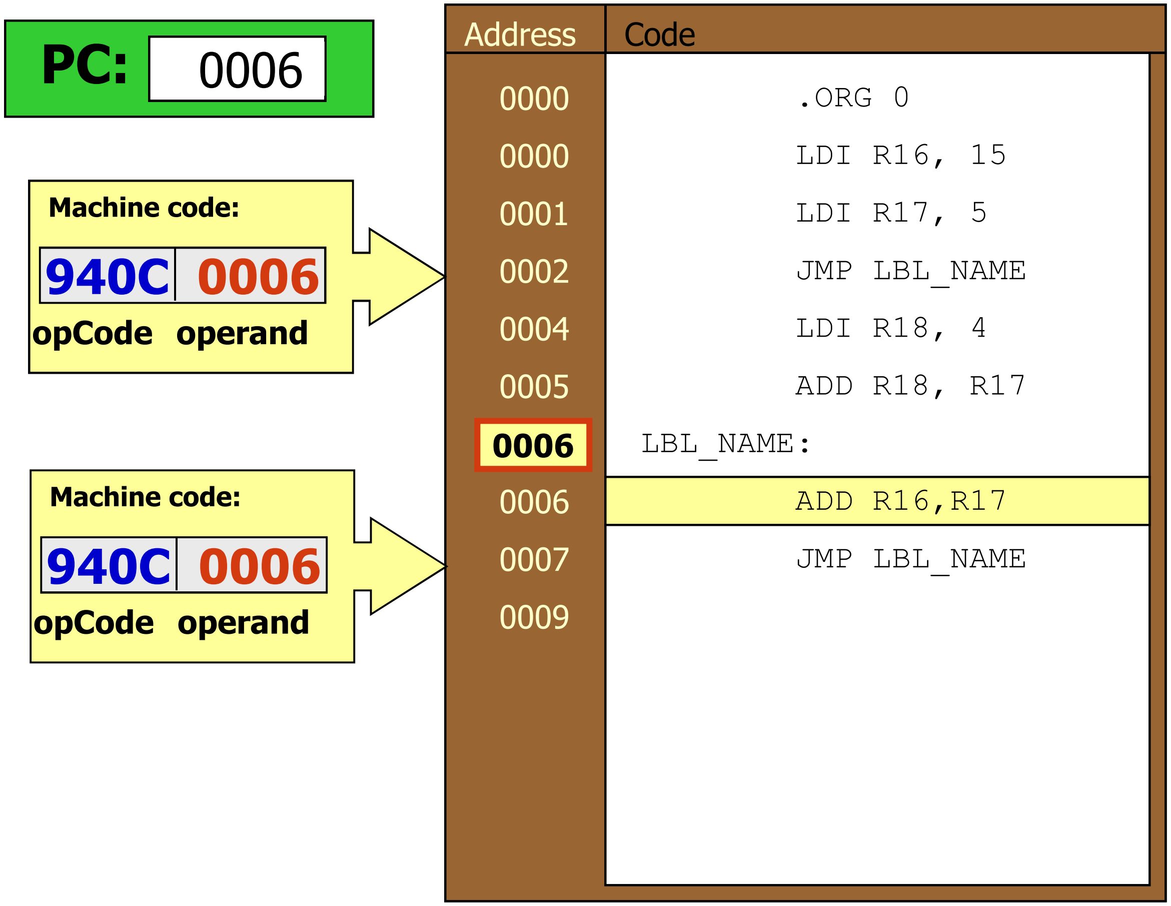
- In JMP, the operand, contains the address of the destination
- When an JMP is executed:
  - PC is loaded with the operand value



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
0009	

JMP

- In JMP, the operand, contains the address of the destination
- When an JMP is executed:
  - PC is loaded with the operand value



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
0009	

RJMP (Relative jump)

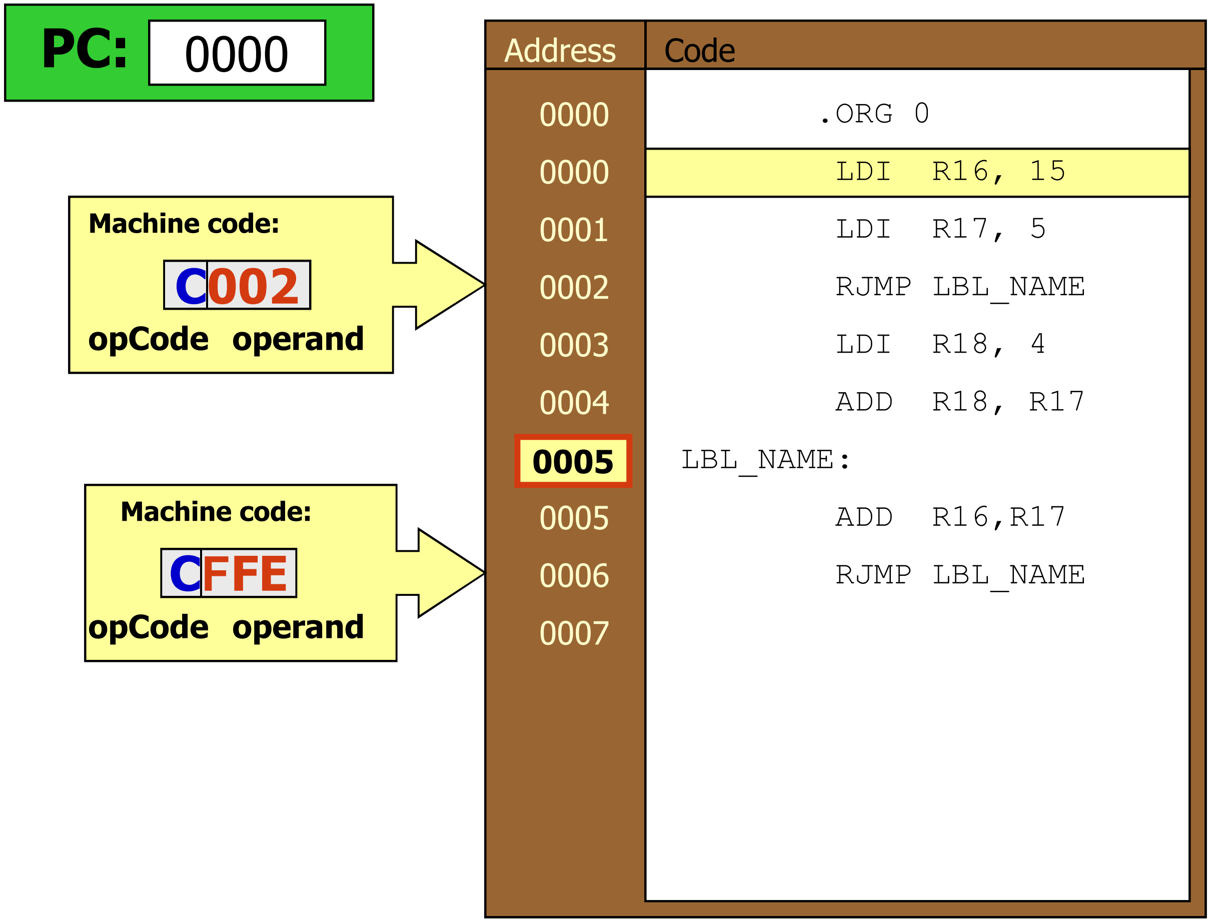
- RJMP ⟺ PC = PC + operand

1100 XXXX XXXX XXXX

- Example: **1100 0000 | 0000 0110**
  - Operand = **000000000110**
  - **PC = PC + 000000000110**

RJMP

- When RJMP is executed:
  - The operand will be added to the current value of PC



PC: 0000

Machine code: C002

opCode operand

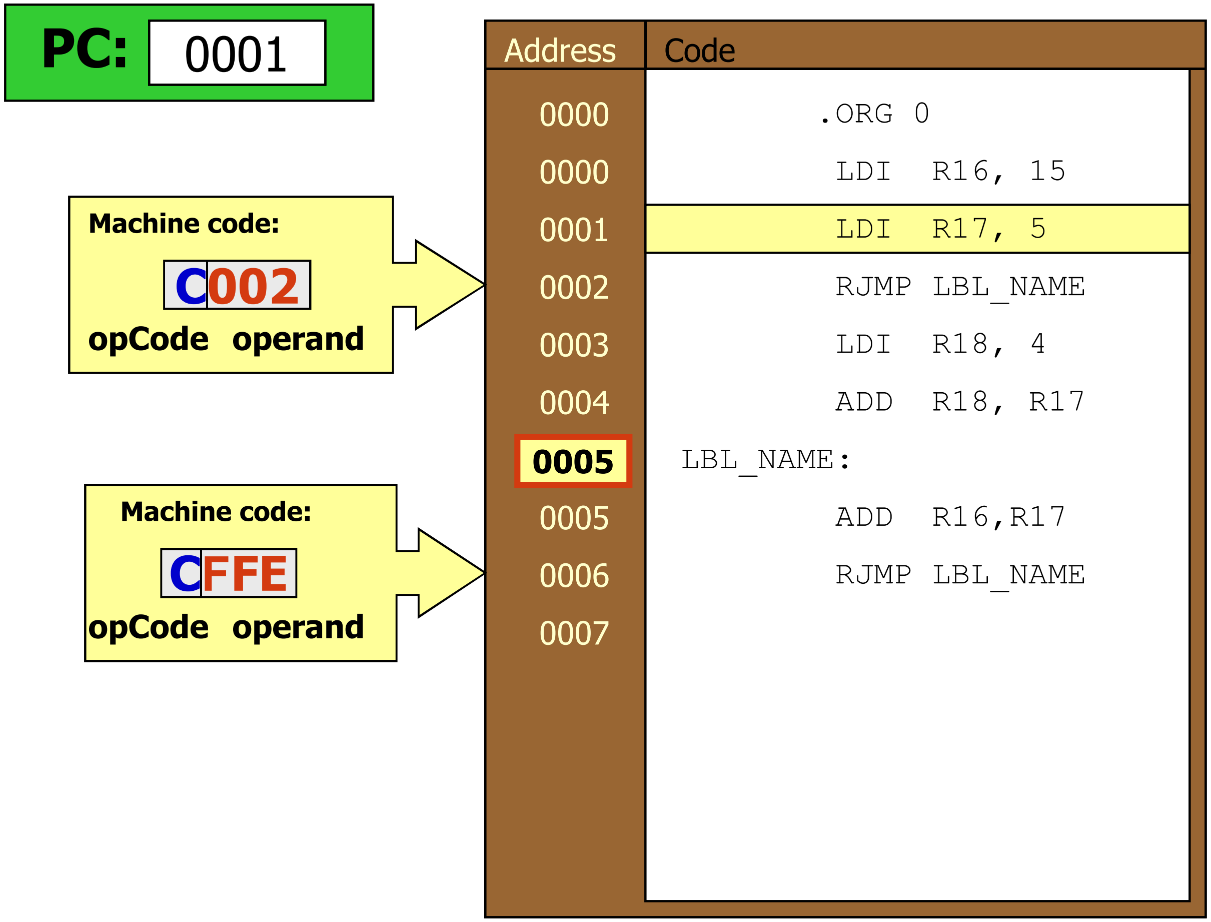
Machine code: CFFE

opCode operand

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
0007	

RJMP

- When RJMP is executed:
  - The operand will be added to the current value of PC



PC: 0001

Machine code:

C002

opCode operand

Machine code:

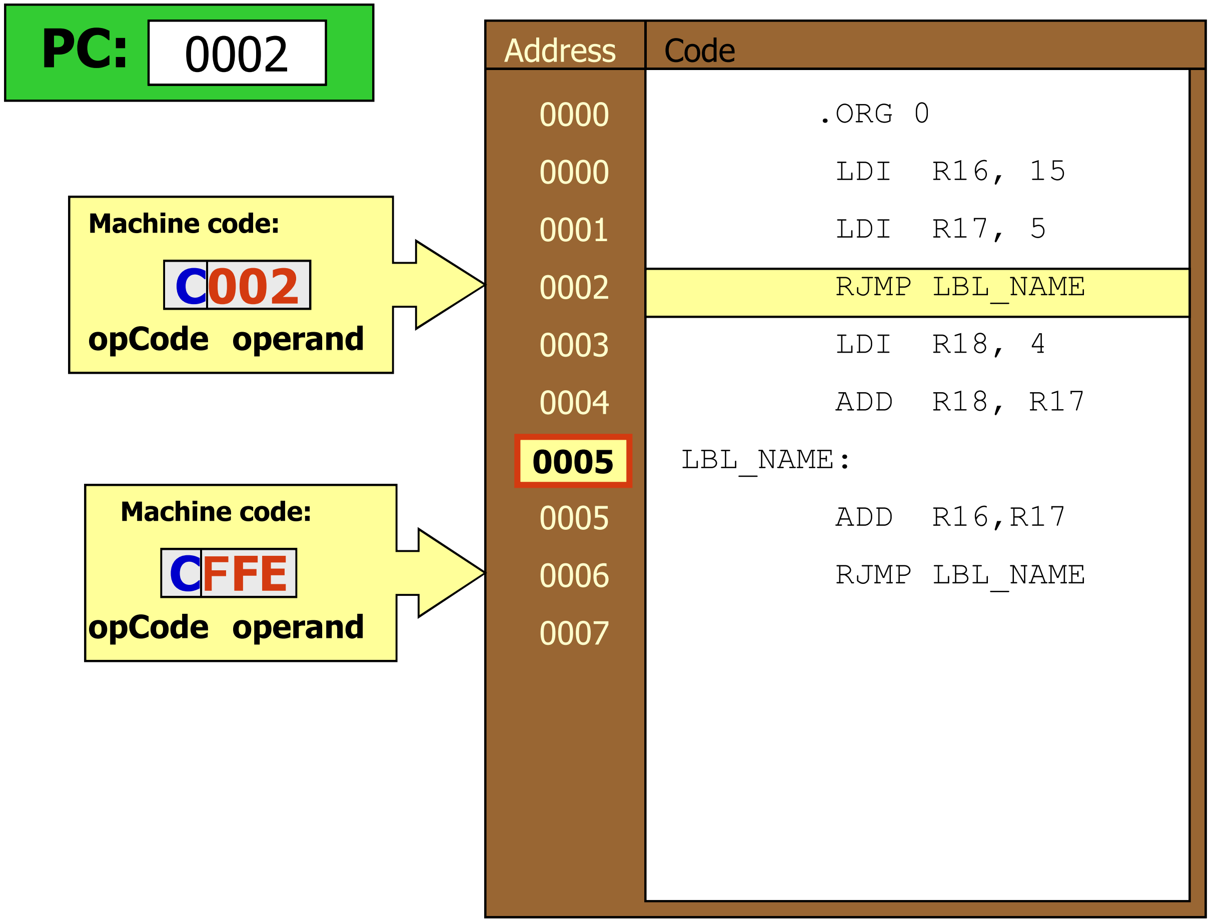
CFFE

opCode operand

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
0007	

RJMP

- When RJMP is executed:
  - The operand will be added to the current value of PC



PC: 0002

Machine code:

C002

opCode operand

Machine code:

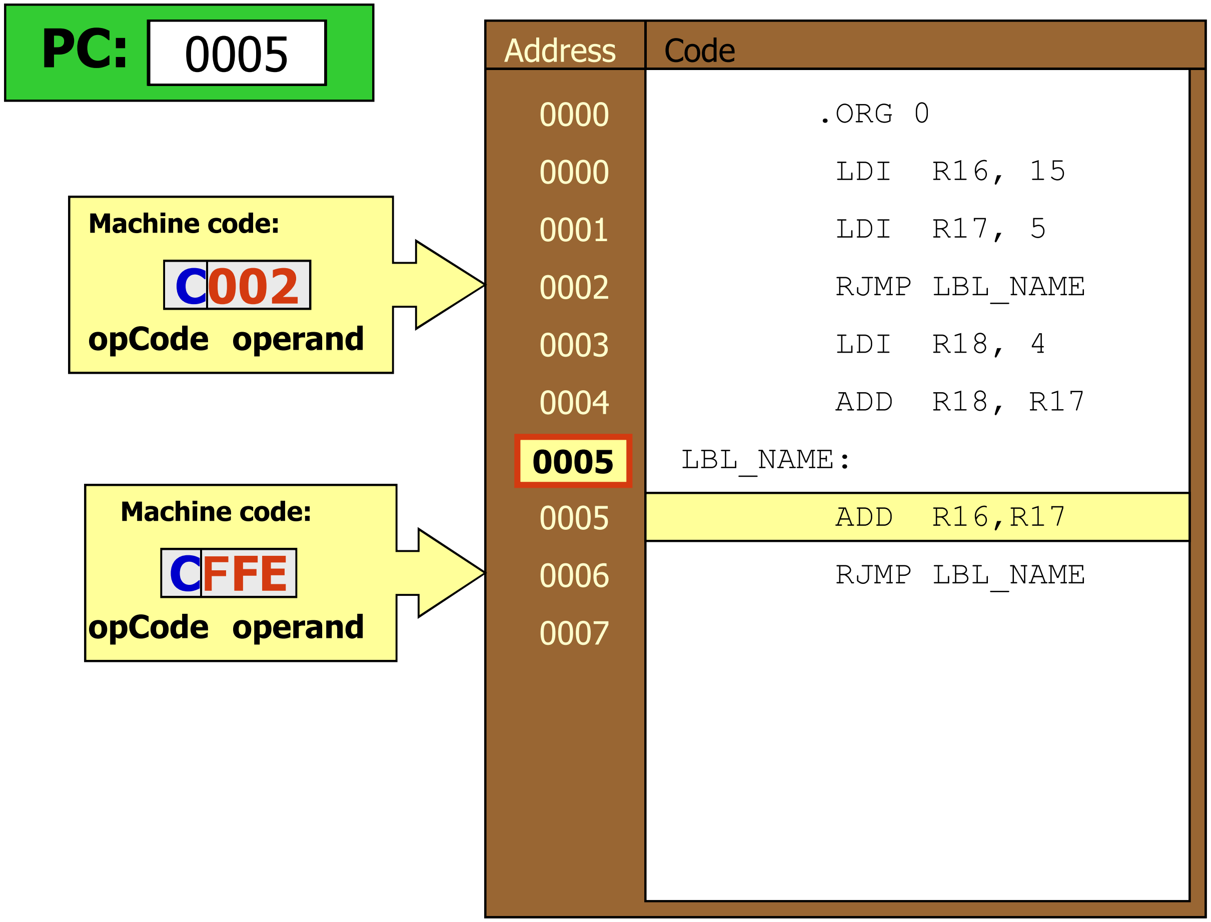
CFFE

opCode operand

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
0007	

RJMP

- When RJMP is executed:
  - The operand will be added to the current value of PC



PC: 0005

Machine code:

C002

opCode operand

Machine code:

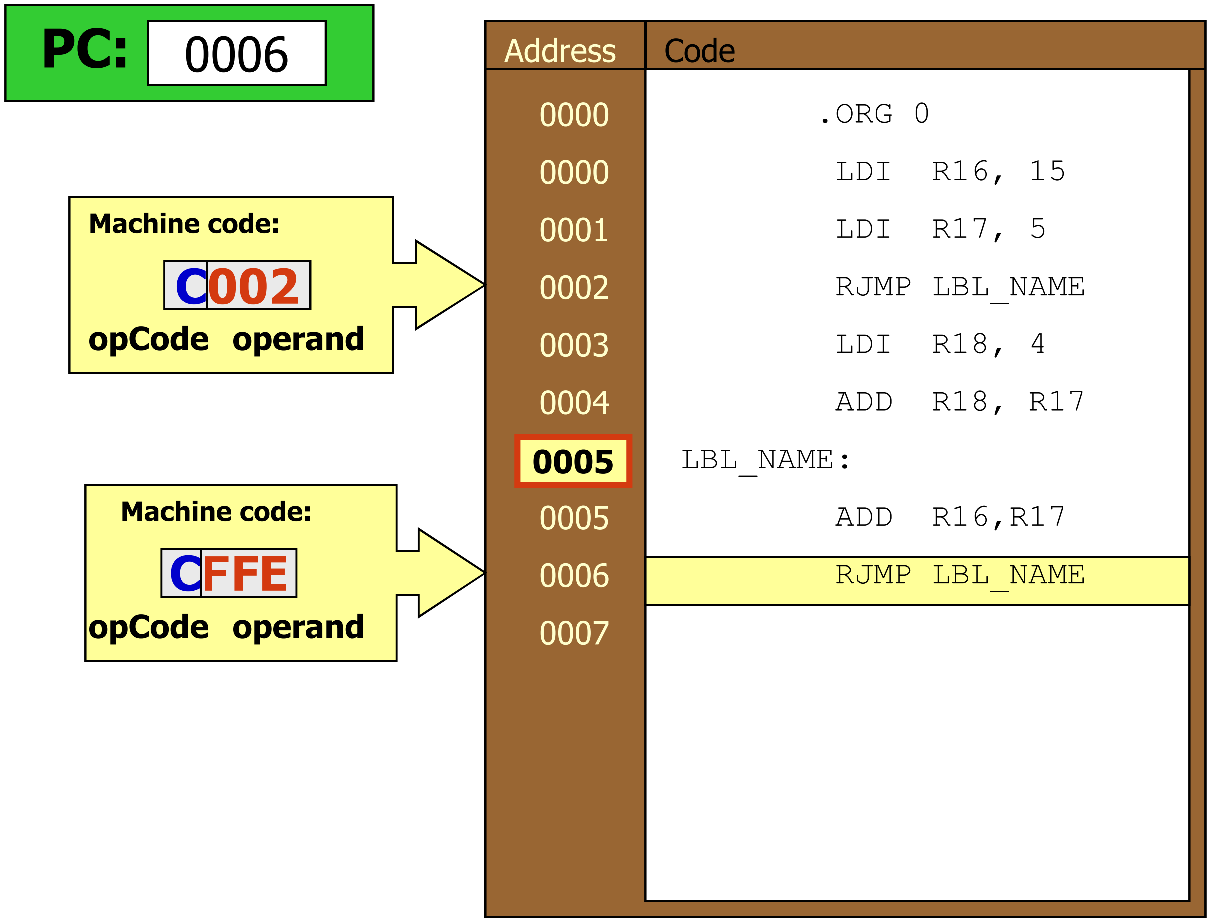
CFFE

opCode operand

```
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
0007
```

RJMP

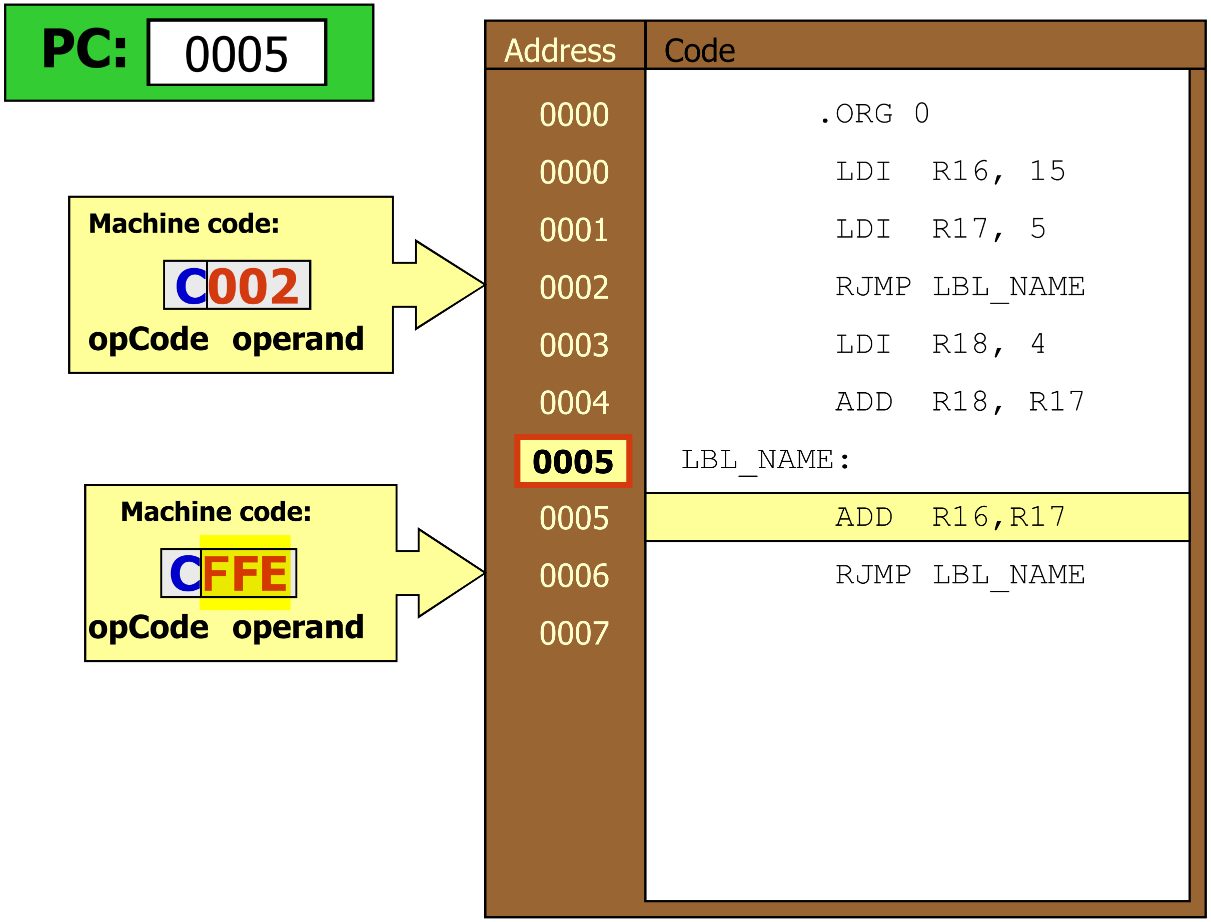
- When RJMP is executed:
  - The operand will be added to the current value of PC



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
0007	

RJMP

- When RJMP is executed:
  - The operand will be added to the current value of PC



PC: 0005

Machine code: C002

opCode operand

Machine code: CFFE

opCode operand

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
0007	

RJMP Backward

- RJMP ⟺ PC = PC + operand

1100 XXXX XXXX XXXX

- Negative Operand(Range Address?)

IJMP (Indirect jump)

- IJMP ⟺ PC = Z register (R30:R31)

1001 0100 0000 1001

- The instruction has no operand.
- the Program counter is loaded with the contents of Z register.
  - For example, if Z points to location 100, by executing IJMP, the CPU jumps to location 100.

Conditional Jump in AVR

SREG:	I	T	H	S	V	N	Z	C

- Relative Jump
- 2 Byte jump
- Jump Address (-64 : +64)

1111 01XX XXXX X000

Conditional Jump in AVR

SREG:	I	T	H	S	V	N	Z	C

- The conditional jump instructions in AVR are as follows:

Instruction	**Abbreviation of**	**Comment**
BREQ *lbl*	Branch if Equal	Jump to location *lbl* if Z = 1,
BRNE *lbl*	Branch if Not Equal	Jump if Z = 0, to location *lbl*
BRCS *lbl* BRLO *lbl*	Branch if Carry Set Branch if Lower	Jump to location *lbl*, if C = 1
BRCC *lbl* BRSH *lbl*	Branch if Carry Cleared Branch if Same or Higher	Jump to location *lbl*, if C = 0
BRMI *lbl*	Branch if Minus	Jump to location lbl, if N = 1
BRPL *lbl*	Branch if Plus	Jump if N = 0
BRGE *lbl*	Branch if Greater or Equal	Jump if S = 0
BRLT*lbl*	Branch if Less Than	Jump if S = 1
BRHS *lbl*	Branch if Half Carry Set	If H = 1 then jump to *lbl*
BRHC lbl	Branch if Half Carry Cleared	if H = 0 then jump to lbl
BRTS	Branch if T flag Set	If T = 1 then jump to lbl
BRTC	Branch if T flag Cleared	If T = 0 then jump to lbl
BRIS	Branch if I flag set	If I = 1 then jump to lbl
BRIC	Branch if I flag cleared	If I = 0 then jump to lbl

Usages of Conditional jump

- Conditions
- Loop

Conditions

- When b is subtracted from a:
  - The result is zero, when a is equal to b
  - Carry will be set when a < b
    - Note: Carry set when we borrow in high level concept

$$\begin{array}{r} a \\ -\ b \\ \hline \end{array}$$

SREG:

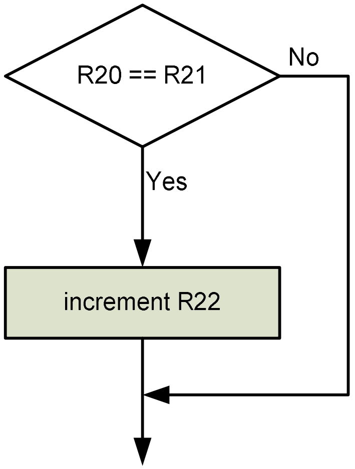
I	T	H	S	V	N	Z	C

Example 1

- Write a program that if R20 is equal to R21 then R22 increases.

- Solution:

```
    SUB R20,R21     ;Z will be set if R20 == R21
    BRNE NEXT       ;if Not Equal jump to next
    INC R22
NEXT:
```

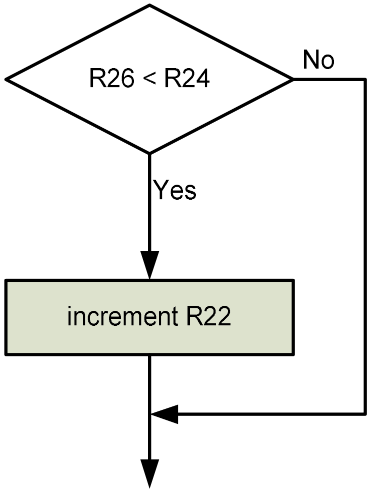


Example 2

- Write a program that if R26 < R24 then R22 increases.

- Solution:

```
    SUB R26,R24     ;C will be set when R26 < R24
    BRCC L1         ;if Carry cleared jump to L1
    INC R22
L1:
```

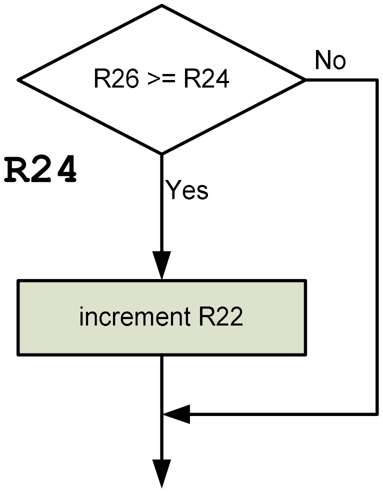


Example 3

- Write a program that if R26 >= R24 then R22 increases.

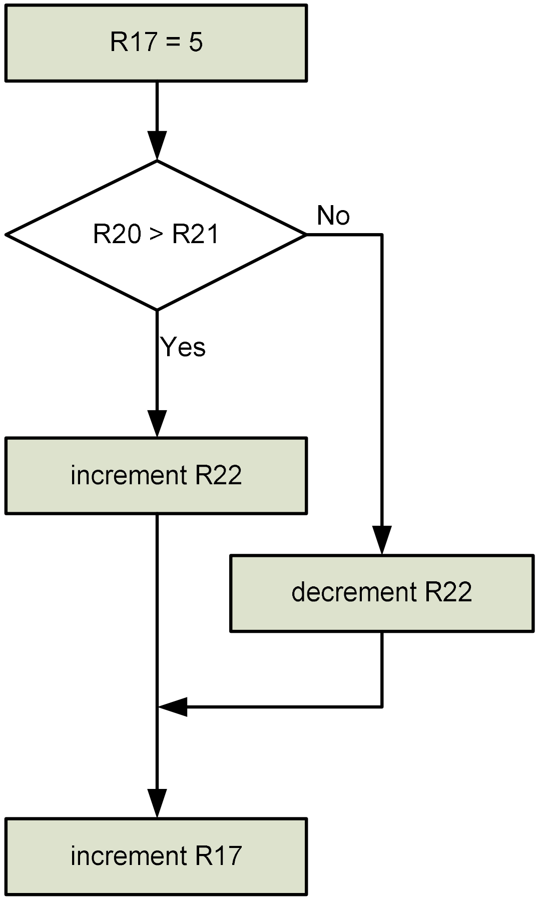
- Solution:

```
        SUB R26,R24      ;C will be cleared when R26 >= R24
        BRCS L1          ;if Carry set jump to L1
        INC R22
    L1:
```



Example 4: IF and ELSE

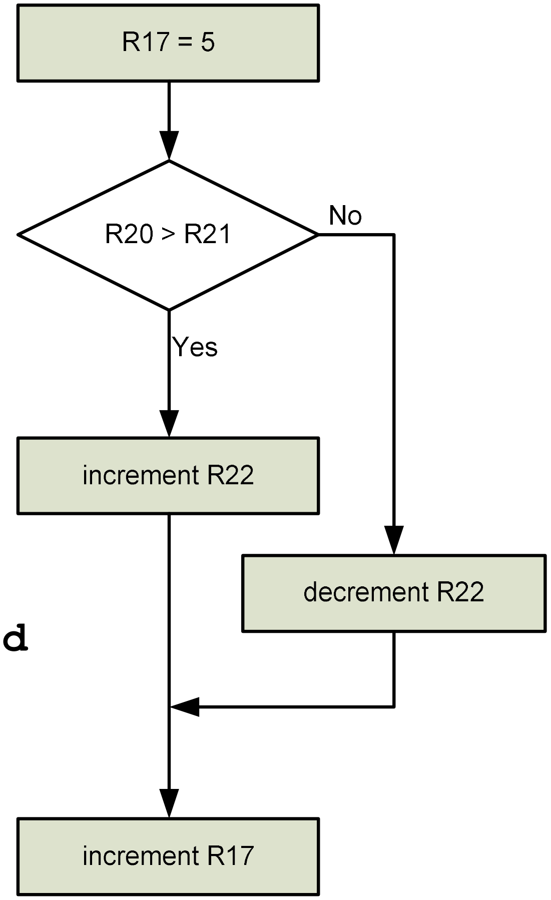
```
int main ( )
{
   R17 = 5;
   if (R20 > R21)
     R22++;
   else
     R22--;
   R17++;
}
```



Example 4: IF and ELSE

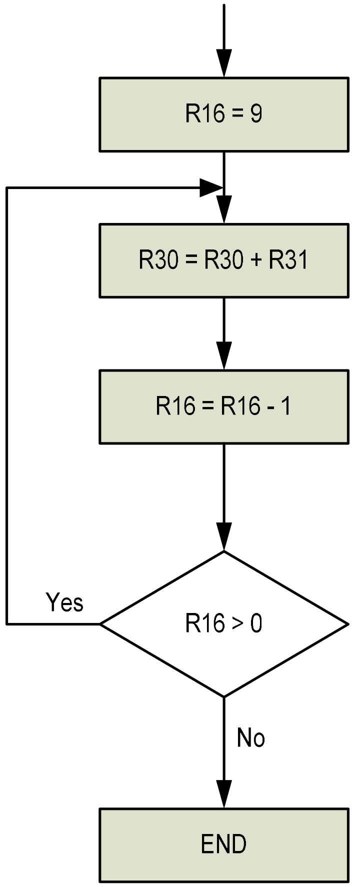
```
int main ( )
{
   R17 = 5;
   if (R20 > R21)
     R22++;
   else
     R22--;
   R17++;
}
```

```
   LDI  R17,5
   SUB  R21, R20        ;C is set when R20>R21
   BRCC ELSE_LABEL      ;jump to else if cleared
   INC  R22
   JMP  NEXT
ELSE_LABEL:
   DEC  R22
NEXT:
   INC  R17
```



Loop

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

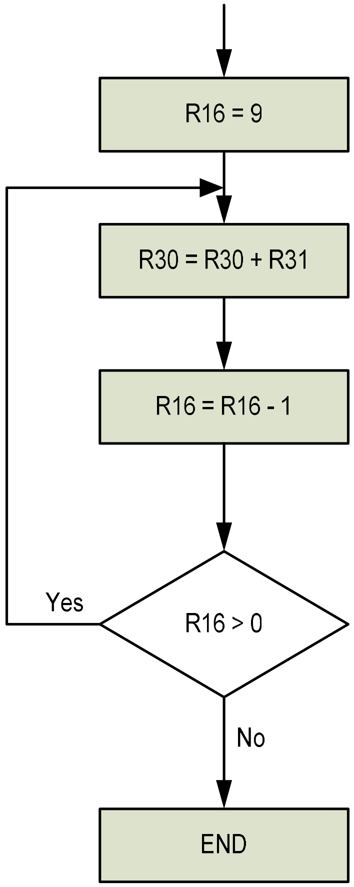


Loop

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

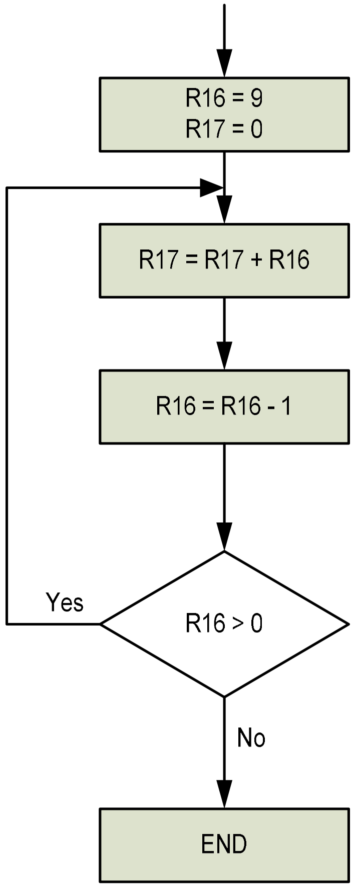
- **Solution:**

```
     .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



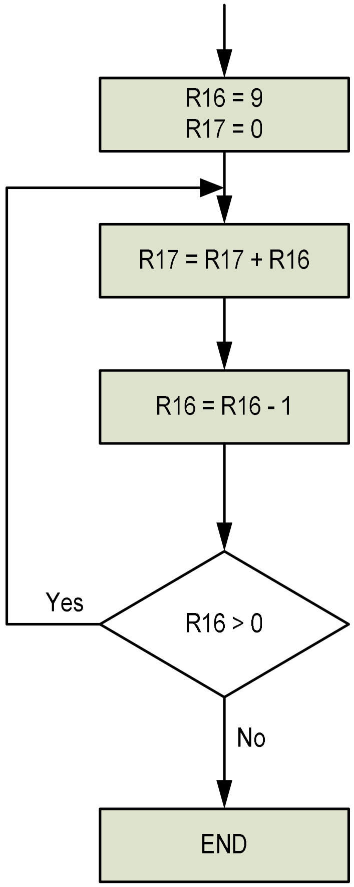
www.araxai.ir

Loop

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

- **Solution:**

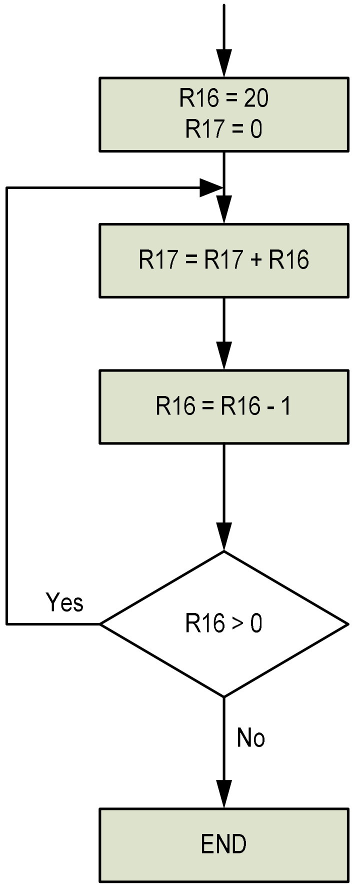
```
     .ORG 00
     LDI  R16, 9          ;R16 = 9
     LDI  R17, 0          ;R17 = 0
L1:  ADD  R17,R16         ;R17 = R17 + R16
     DEC  R16             ;R16 = R16 - 1
     BRNE L1              ;if Z = 0
L2:  RJMP L2              ;Wait here forever
```



www.araxai.ir

Loop

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

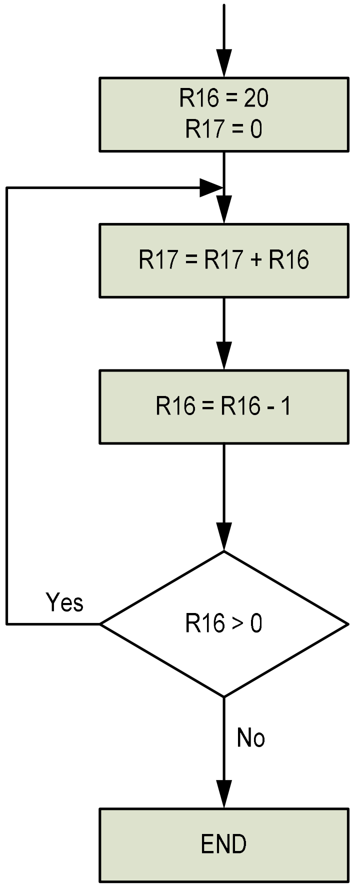


Loop

- 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
```

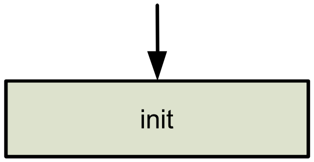


Loop

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

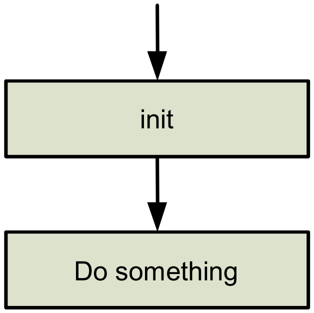
Loop

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



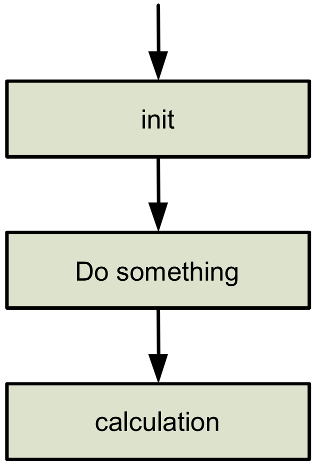
Loop

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



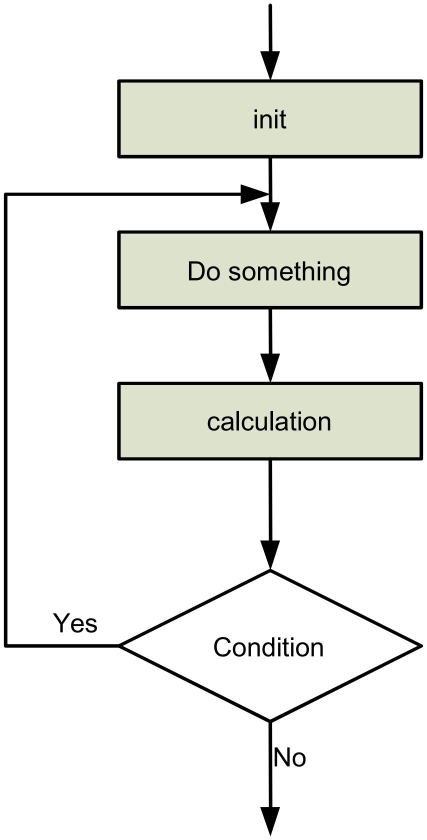
Loop

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



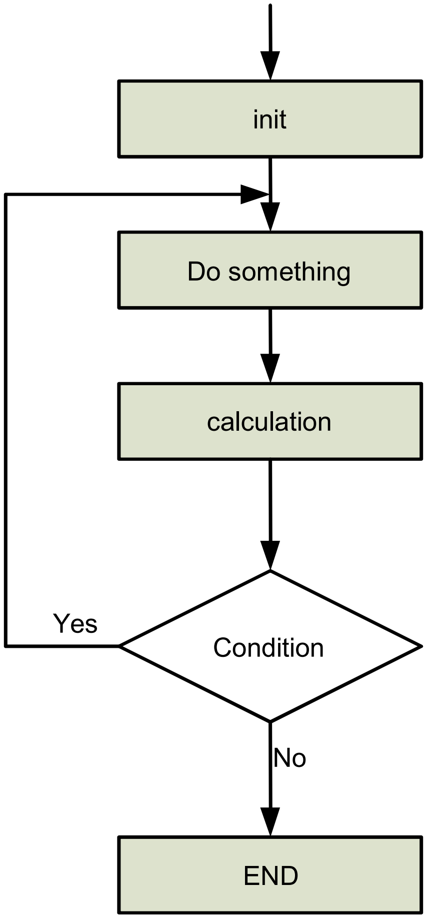
Loop

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



Loop

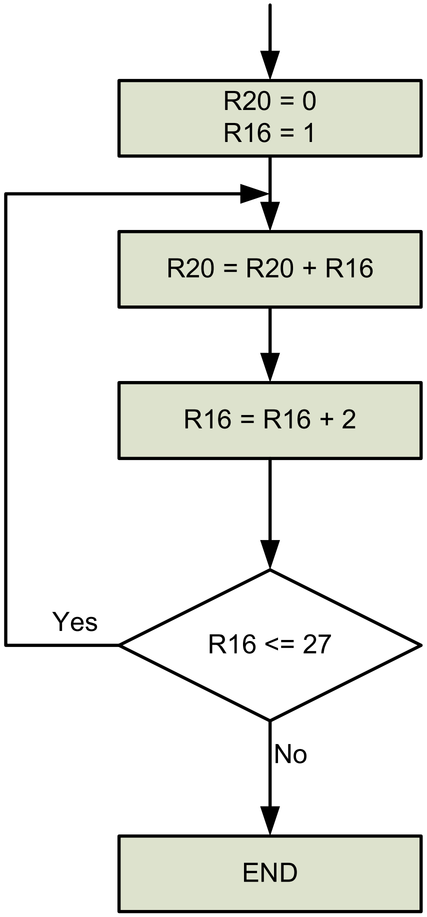
```
for (init; condition; calculation)
{
    do something
}
```



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
```



More Points

- TST Rd
  - Update Statues Register based of Rd

Call Topics

- Stack, Push and Pop
- Calling a function

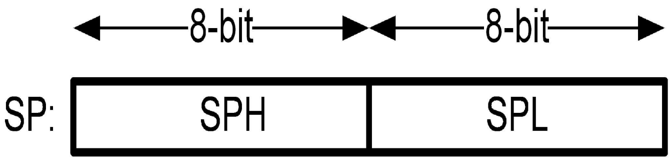
Stack

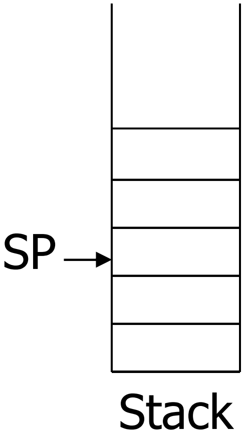
- PUSH *Rr*

```
[SP] = Rr
SP = SP - 1
```

- POP *Rd*

```
SP = SP + 1
Rd = [SP]
```

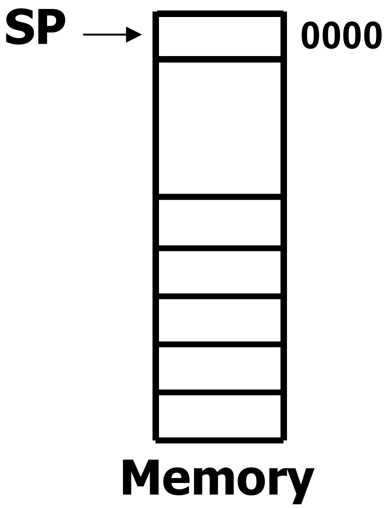




Stack

R20: $00 R22: $00

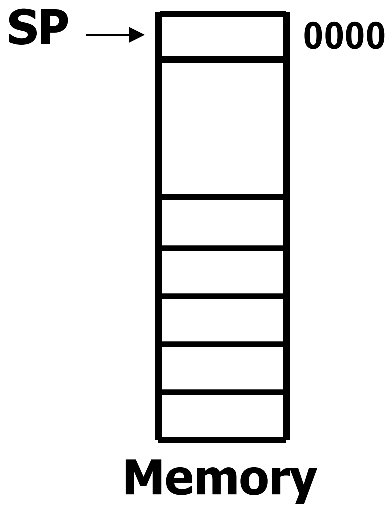
R21: $00 R0: $00



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 R20
0008	PUSH R21
0009	PUSH R22
000A	POP R21
000B	POP R0
000C	POP R20
000D	L1: RJMP L1

Stack

R20:	$00	R22:	$00
R21:	$00	R0:	$00

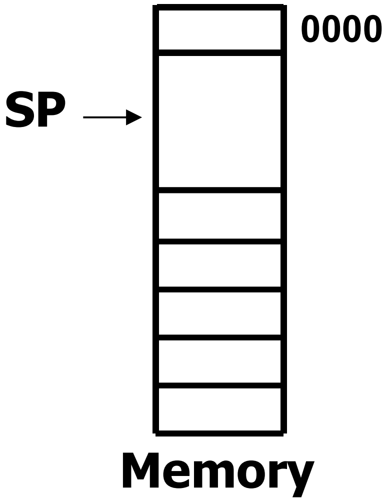


Address	Code
	ORG 0
0000	LDI R16,HIGH(RAMEND)
0001	OUT SPH,R16
0002	LDI R16,LOW(RAMEND)
0003	OUT SPL,R16
0004	LDI R20,0x10
0005	LDI R21, 0x20
0006	LDI R22,0x30
0007	PUSH R20
0008	PUSH R21
0009	PUSH R22
000A	POP R21
000B	POP R0
000C	POP R20
000D	L1: RJMP L1

Stack

R20: $00 **R22:** $00

R21: $00 **R0:** $00

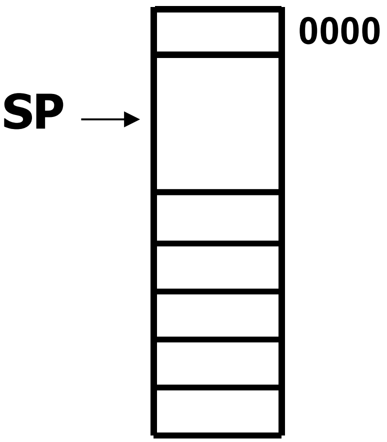


Address	Code
	ORG 0
0000	LDI R16,HIGH(RAMEND)
0001	OUT SPH,R16
0002	LDI R16,LOW(RAMEND)
0003	OUT SPL,R16
0004	LDI R20,0x10
0005	LDI R21, 0x20
0006	LDI R22,0x30
0007	PUSH R20
0008	PUSH R21
0009	PUSH R22
000A	POP R21
000B	POP R0
000C	POP R20
000D	L1: RJMP L1

Stack

R20: $00   R22: $00

R21: $00   R0: $00

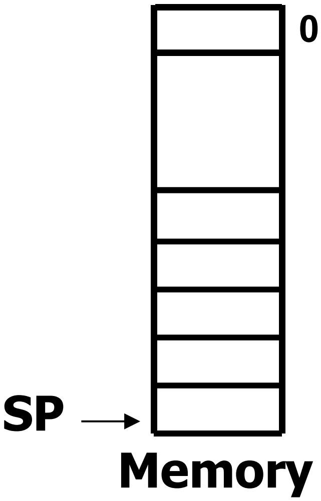


Memory

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 R20
0008	PUSH R21
0009	PUSH R22
000A	POP R21
000B	POP R0
000C	POP R20
000D	L1: RJMP L1

Stack

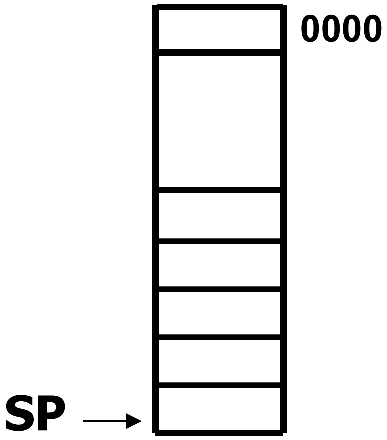
R20: $00	R22: $00
R21: $00	R0: $00



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 R20
0008	PUSH R21
0009	PUSH R22
000A	POP R21
000B	POP R0
000C	POP R20
000D	L1: RJMP L1

Stack

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

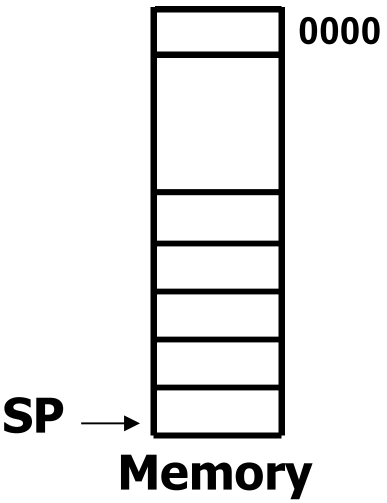


Memory

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 R20
0008	PUSH R21
0009	PUSH R22
000A	POP R21
000B	POP R0
000C	POP R20
000D	L1: RJMP L1

Stack

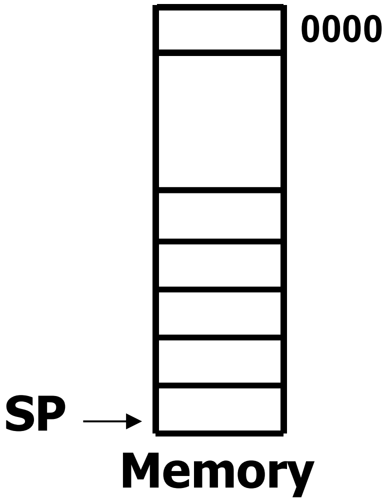
R20:	$10	R22:	$00
R21:	$20	R0:	$00



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 R20
0008	PUSH R21
0009	PUSH R22
000A	POP R21
000B	POP R0
000C	POP R20
000D	L1: RJMP L1

Stack

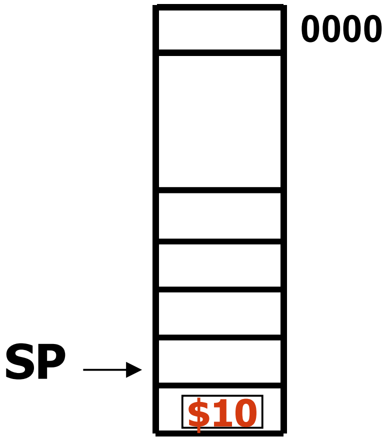
R20:	$10	R22:	$30
R21:	$20	R0:	$00



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 R20
0008	PUSH R21
0009	PUSH R22
000A	POP R21
000B	POP R0
000C	POP R20
000D	L1: RJMP L1

Stack

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

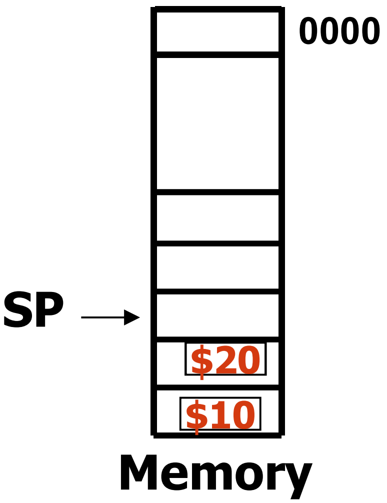


Memory

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 R20
0008	PUSH R21
0009	PUSH R22
000A	POP R21
000B	POP R0
000C	POP R20
000D	L1: RJMP L1

Stack

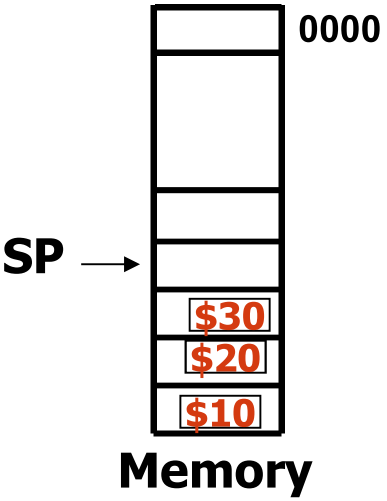
R20:	$10	R22:	$30
R21:	$20	R0:	$00



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 R20
0008	PUSH R21
0009	PUSH R22
000A	POP R21
000B	POP R0
000C	POP R20
000D	L1: RJMP L1

Stack

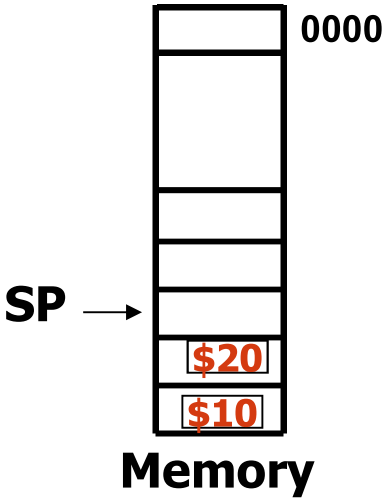
R20:	$10	R22:	$30
R21:	$20	R0:	$00



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 R20
0008	PUSH R21
0009	PUSH R22
000A	POP R21
000B	POP R0
000C	POP R20
000D	L1: RJMP L1

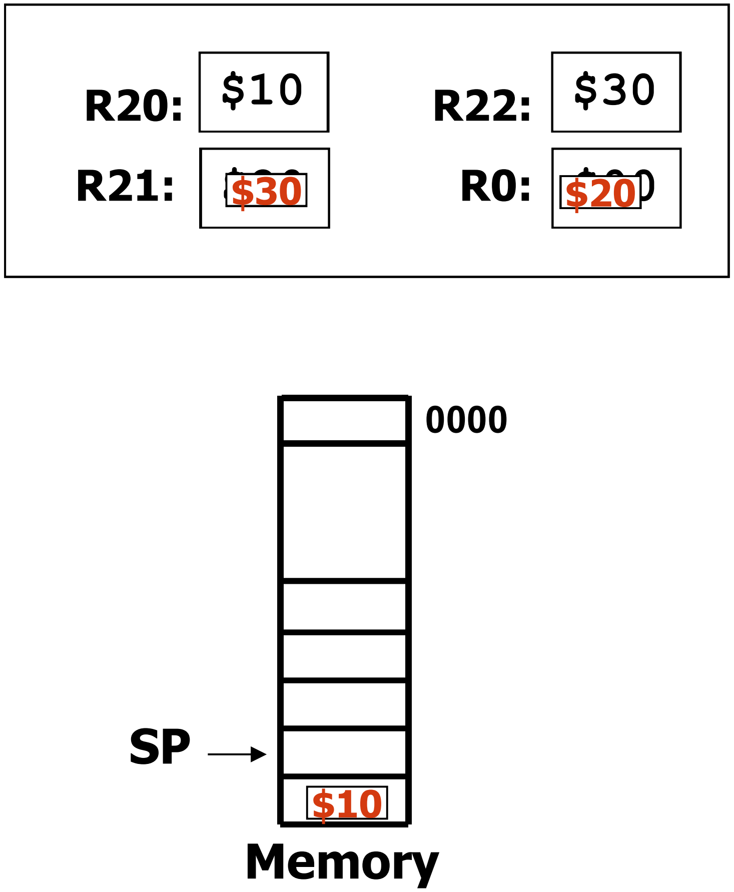
Stack

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



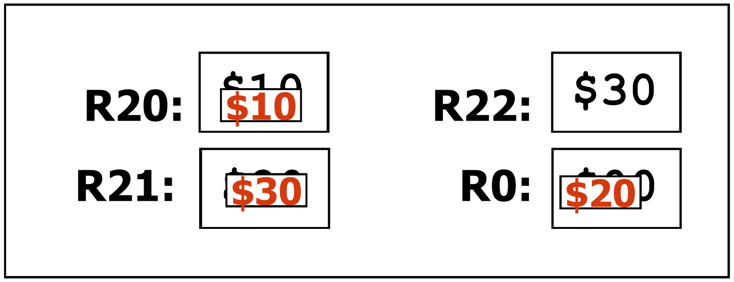
Address	Code
	ORG 0
0000	LDI R16,HIGH(RAMEND)
0001	OUT SPH,R16
0002	LDI R16,LOW(RAMEND)
0003	OUT SPL,R16
0004	LDI R20,0x10
0005	LDI R21, 0x20
0006	LDI R22,0x30
0007	PUSH R20
0008	PUSH R21
0009	PUSH R22
000A	POP R21
000B	POP R0
000C	POP R20
000D	L1: RJMP L1

Stack



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 R20
0008	PUSH R21
0009	PUSH R22
000A	POP R21
000B	POP R0
000C	POP R20
000D	L1: RJMP L1

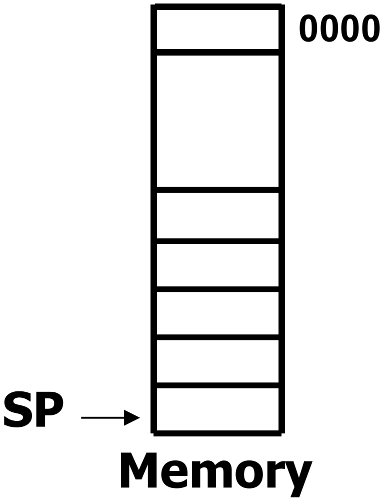
Stack


R20: $10

R22: $30

R21: $30

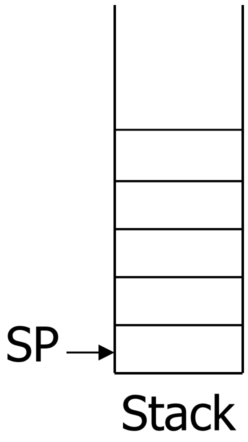
R0: $20



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 R20
0008	PUSH R21
0009	PUSH R22
000A	POP R21
000B	POP R0
000C	POP R20
000D	L1: RJMP L1

Calling a Function

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



Machine code:

940E **000A**

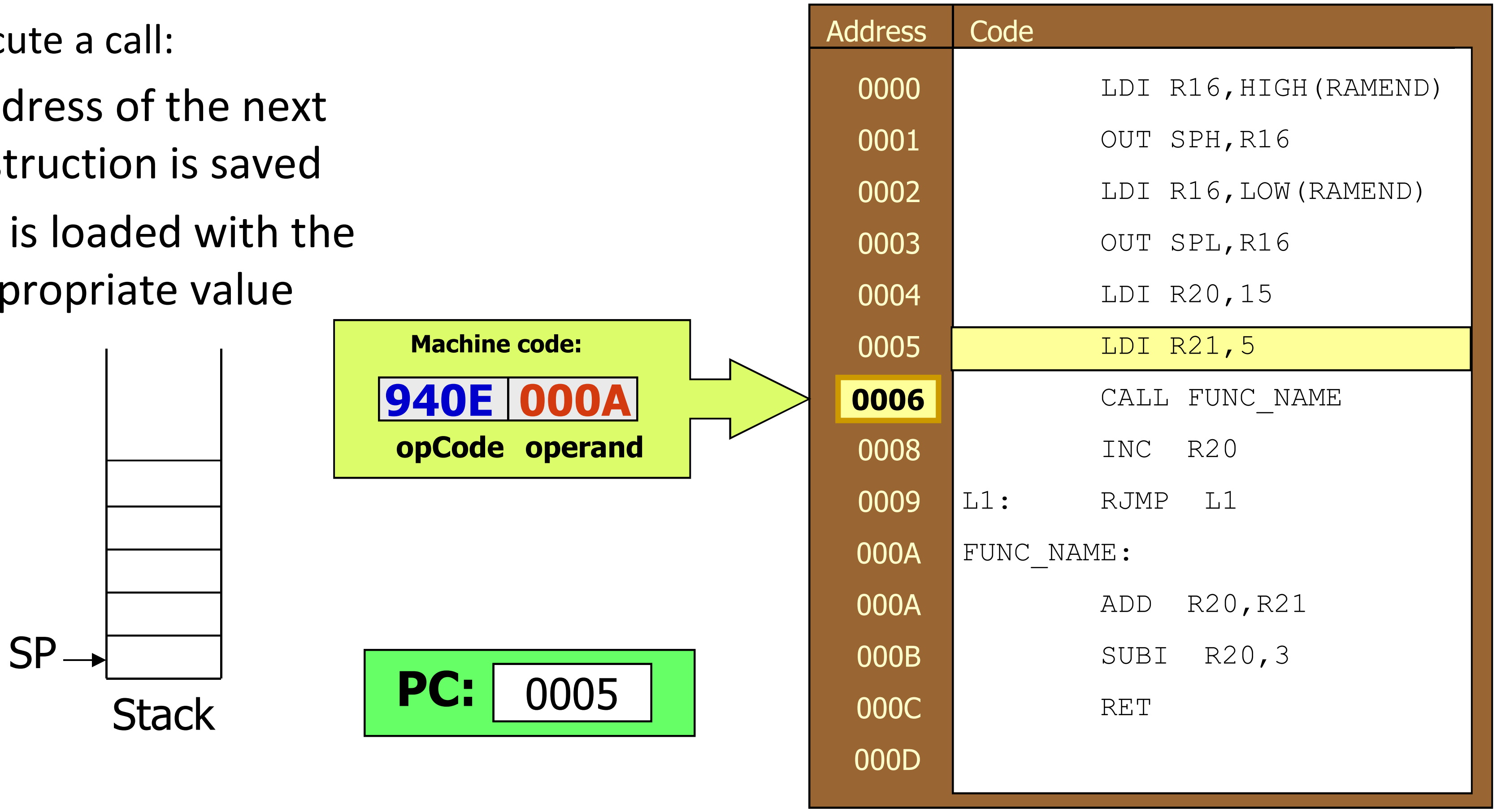
opCode **operand**

PC: 0004

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

Calling a Function

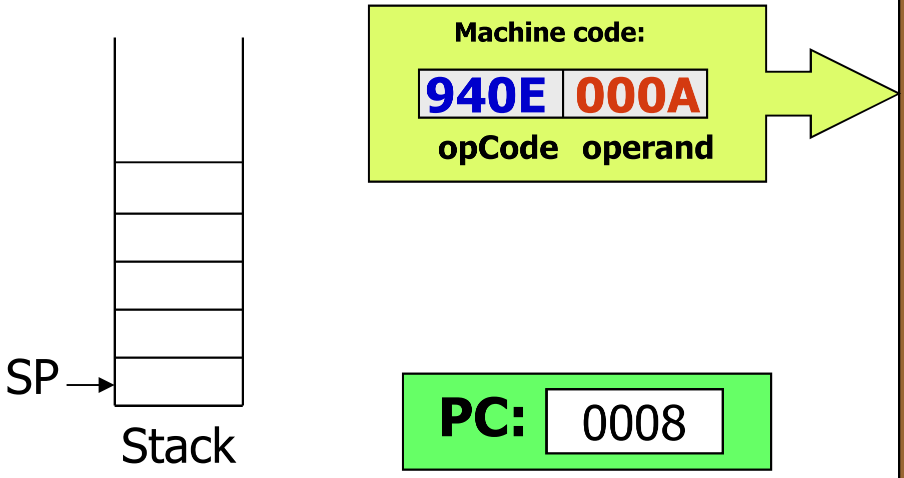
- To execute a call:
  - Address of the next instruction is saved
  - PC is loaded with the appropriate value



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

Calling a Function

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



Machine code:

940E **000A**

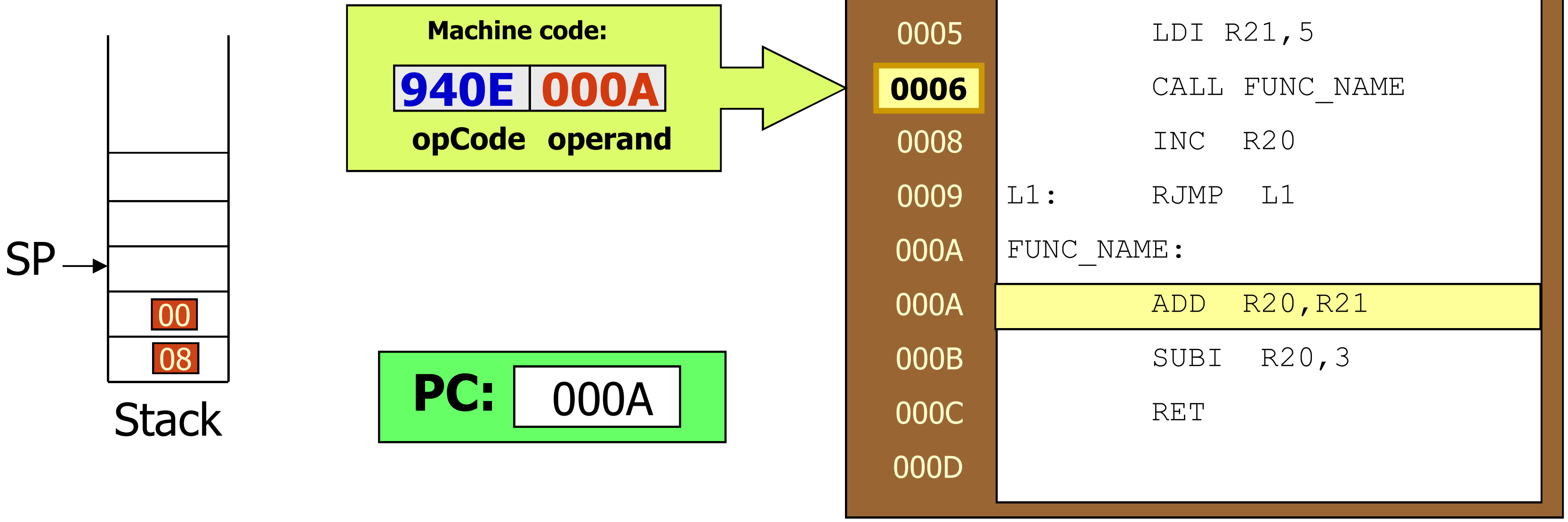
opCode operand

PC: 0008

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

Calling a Function

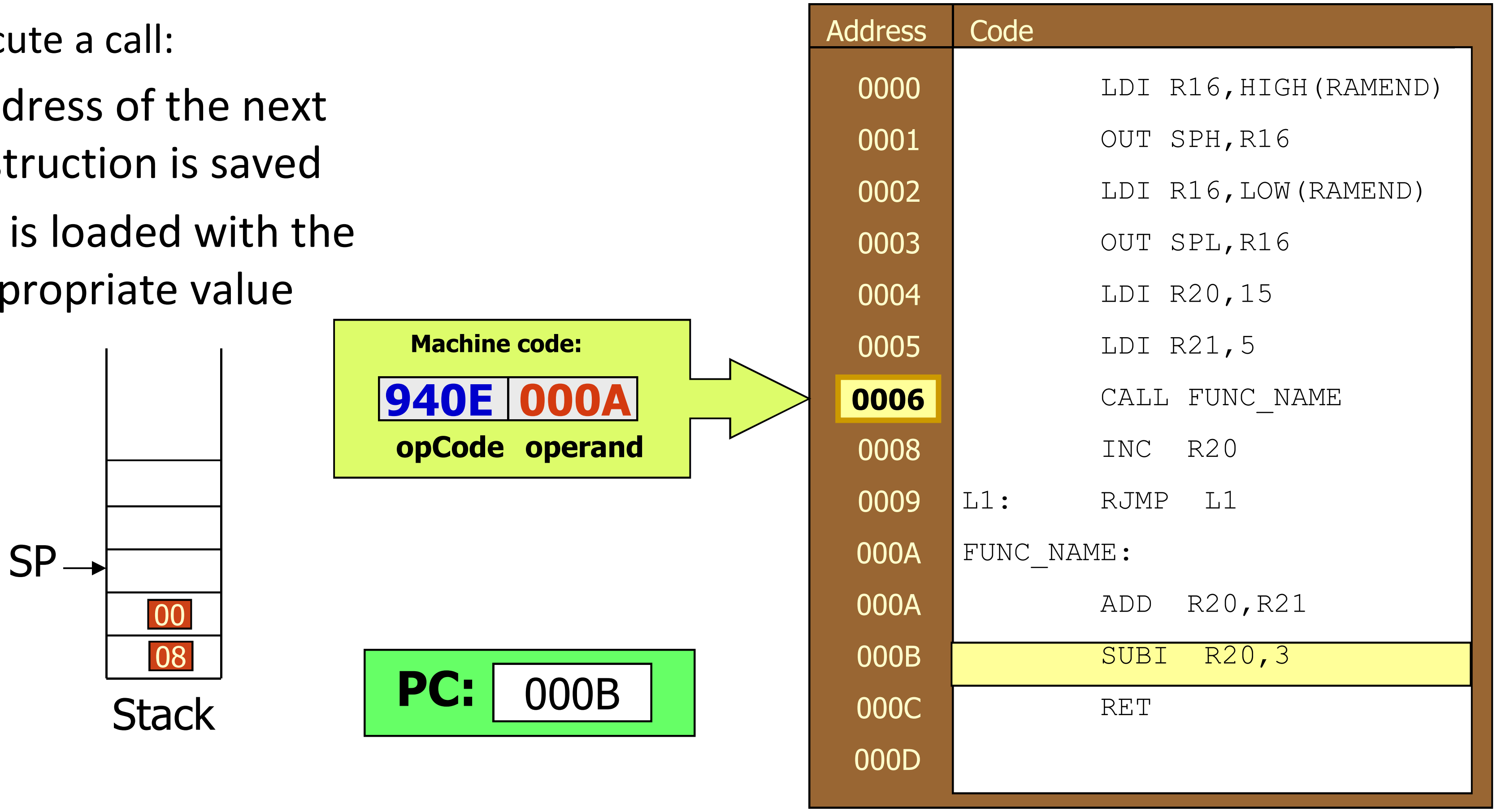
- To execute a call:
  - Address of the next instruction is saved
  - PC is loaded with the appropriate value



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

Calling a Function

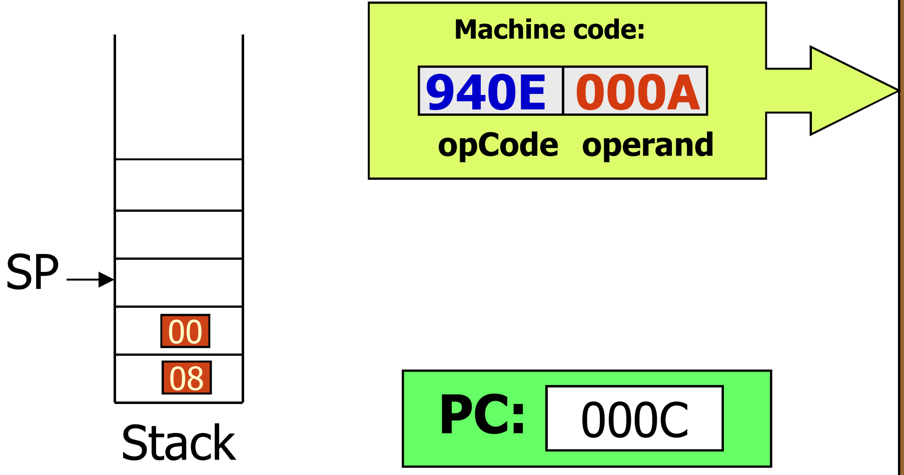
- To execute a call:
  - Address of the next instruction is saved
  - PC is loaded with the appropriate value



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

Calling a Function

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



Machine code:

940E 000A

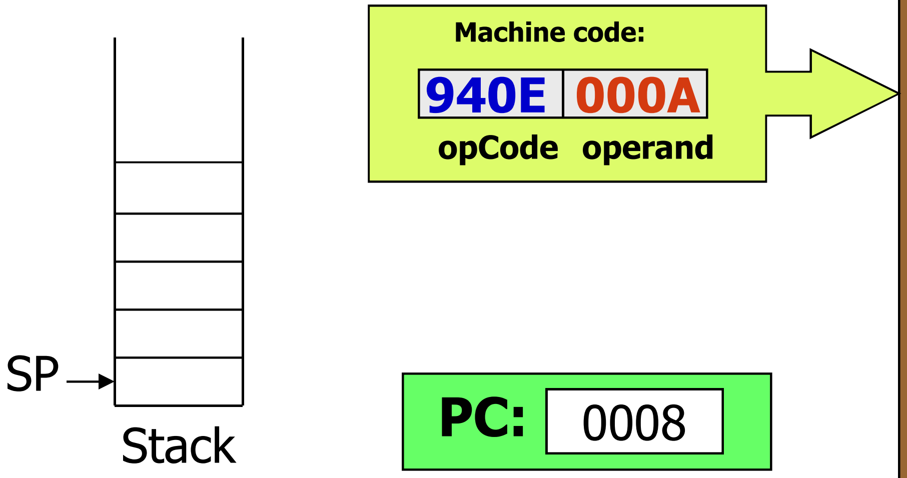
opCode operand

PC: 000C

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

Calling a Function

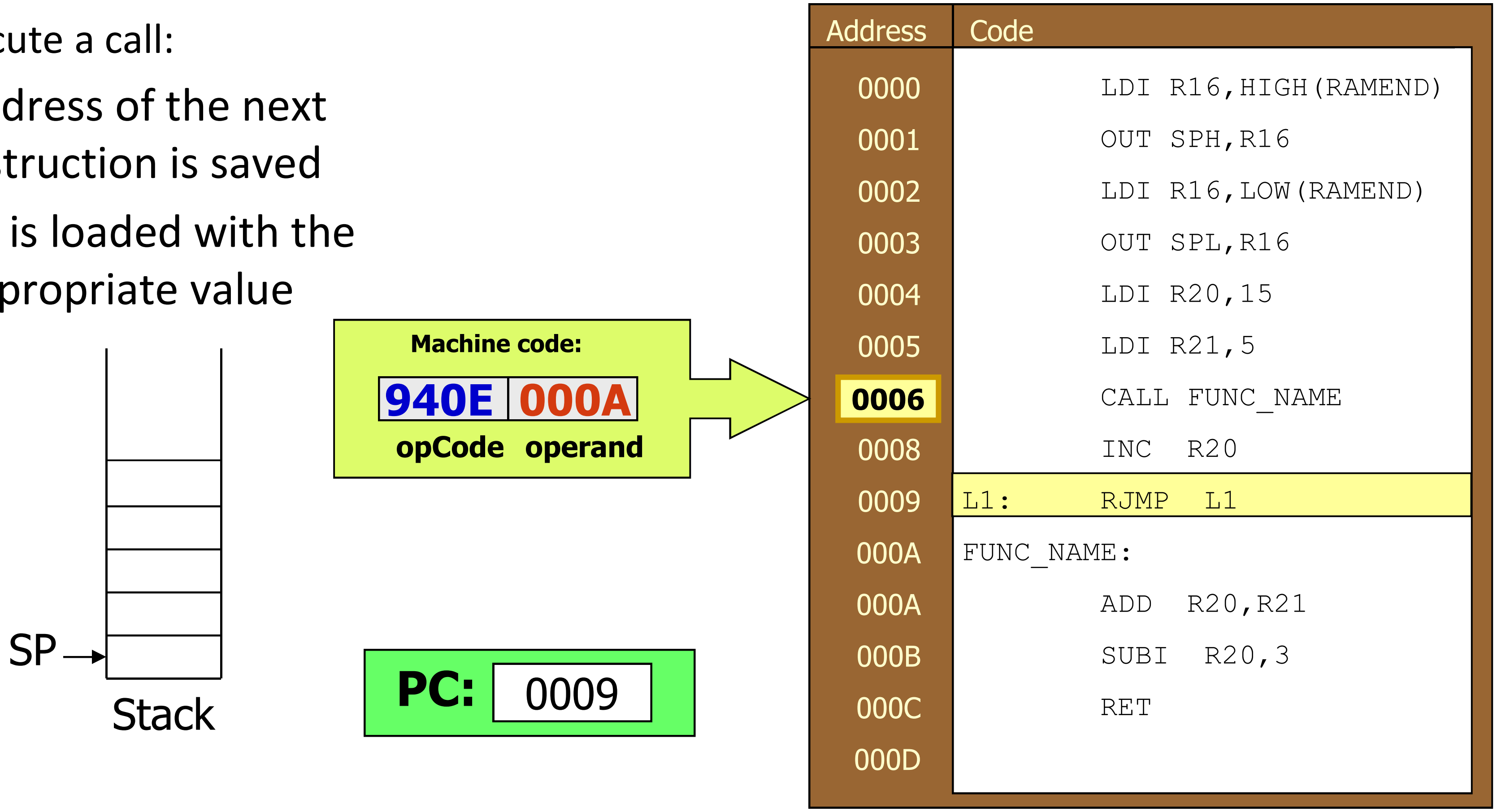
- To execute a call:
  - Address of the next instruction is saved
  - PC is loaded with the appropriate value



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

Calling a Function

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

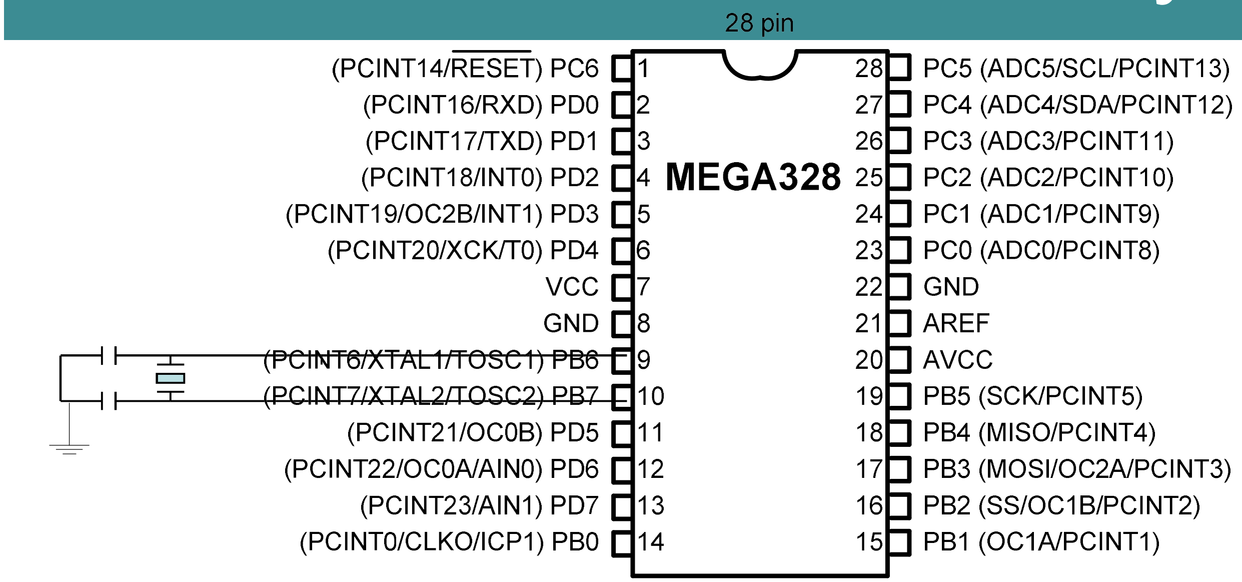


Address	Code
0000	LDI R16,HIGH(RAMEND)
0001	OUT SPH,R16
0002	LDI R16,LOW(RAMEND)
0003	OUT SPL,R16
0004	LDI R20,15
0005	LDI R21,5
0006	CALL FUNC_NAME
0008	INC R20
0009	L1: RJMP L1
000A	FUNC_NAME:
000A	ADD R20,R21
000B	SUBI R20,3
000C	RET
000D	

Some More Point

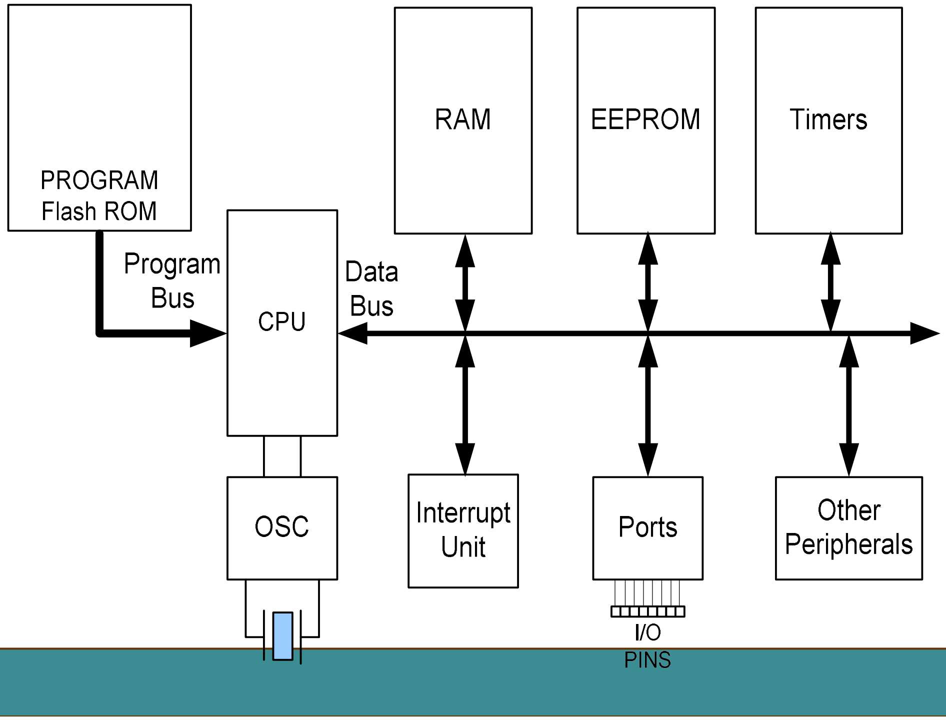
- RCALL (Relative CALL)
- ICALL (Indirect CALL)

Time delay



$$T_{\text{Machine cycle}} = \frac{1}{F_{\text{XTAL}}}$$

$$T_{\text{Machine cycle}} = \frac{1}{16\text{MHz}} = 62.5 \text{ ns}$$



Time delay

```
LDI     R16, 19
LDI     R20, 95
LDI     R21, 5
ADD     R16, R20
ADD     R16, R21
```

Delay = 5 x $T_{\text{machine cycle}}$ = 5 x 62.5 ns = 312.5 ns

Time delay

```
                             machine cycle
LDI     R16, 19                    1
LDI     R20, 95                    1
LDI     R21, 5                     1
ADD     R16, R20                   1
ADD     R16, R21                   1
                               ─────────
                                   5
```

$$\text{Delay} = 5 \times T_{\text{machine cycle}} = 5 \times 62.5\ \text{ns} = 312.5\ \text{ns}$$

Time delay

```
                                machine cycle
        LDI     R16, 100              1
AGAIN:  ADD     R17,R16               1
        DEC     R16                   1
        BRNE    AGAIN                1/2
```

Time delay

```
                                     machine cycle
        LDI     R16, 100                   1
AGAIN:  ADD     R17,R16                    1
        DEC     R16                        1
        BRNE    AGAIN                     1/2
```

Branch penalty

Time delay

```
                                    machine cycle
        LDI     R16, 100                  1
AGAIN:  ADD     R17,R16                   1
        DEC     R16                       1
        BRNE    AGAIN                    1/2
```

Time delay

```
                                  machine cycle
        LDI     R16, 100                1
AGAIN:  ADD     R17,R16                 1
        DEC     R16                     1
        BRNE    AGAIN                  1/2
```

Time delay

```
                                   machine cycle
          LDI     R16, 100               1
AGAIN:    ADD     R17,R16                1          *100
          DEC     R16                    1          *100
          BRNE    AGAIN                 1/2         *100
```

Time delay

```
                            machine cycle
        LDI     R16, 50           1
AGAIN:  NOP                       1
        NOP                       1
        DEC     R16               1
        BRNE    AGAIN            1/2
```

Time delay

```
                                  machine cycle
         LDI     R16, 50                1
AGAIN:   NOP                            1
         NOP                            1
         DEC     R16                    1
         BRNE    AGAIN                 1/2
```

Time delay

```
                                   machine cycle
        LDI     R16, 50                  1
AGAIN:  NOP                              1          *50
        NOP                              1          *50
        DEC     R16                      1          *50
        BRNE    AGAIN                   1/2         *50
                                     ___________
```

Time delay

```
                                 machine cycle
        LDI     R17, 20               1
L1:     LDI     R16, 50               1
L2:     NOP                           1
        NOP                           1
        DEC     R16                   1
        BRNE    L2                   1/2
        DEC     R17                   1
        BRNE    L1                   1/2
                                    ______
```

Time delay

```
                                   machine cycle
        LDI     R17, 20                  1
L1:     LDI     R16, 50                  1
L2:     NOP                              1
        NOP                              1
        DEC     R16                      1
        BRNE    L2                      1/2
        DEC     R17                      1
        BRNE    L1                      1/2
                                     ________
```

Time delay

```
                                     machine cycle
            LDI     R17, 20                1
L1:         LDI     R16, 50                1
L2:         NOP                            1
            NOP                            1
            DEC     R16                    1
            BRNE    L2                    1/2
            DEC     R17                    1
            BRNE    L1                    1/2
                                      ________
```

Time delay

```
                                    machine cycle
        LDI     R17, 20                  1
L1:     LDI     R16, 50                  1       *20
L2:     NOP                              1       *20 * 50
        NOP                              1       *20 * 50
        DEC     R16                      1       *20 * 50
        BRNE    L2                      1/2      *20 * 50
        DEC     R17                      1       *20
        BRNE    L1                      1/2      *20
                                    ________
```