EE-222: Microprocessor Systems

AVR Microcontroller: Jump (aka Branch)

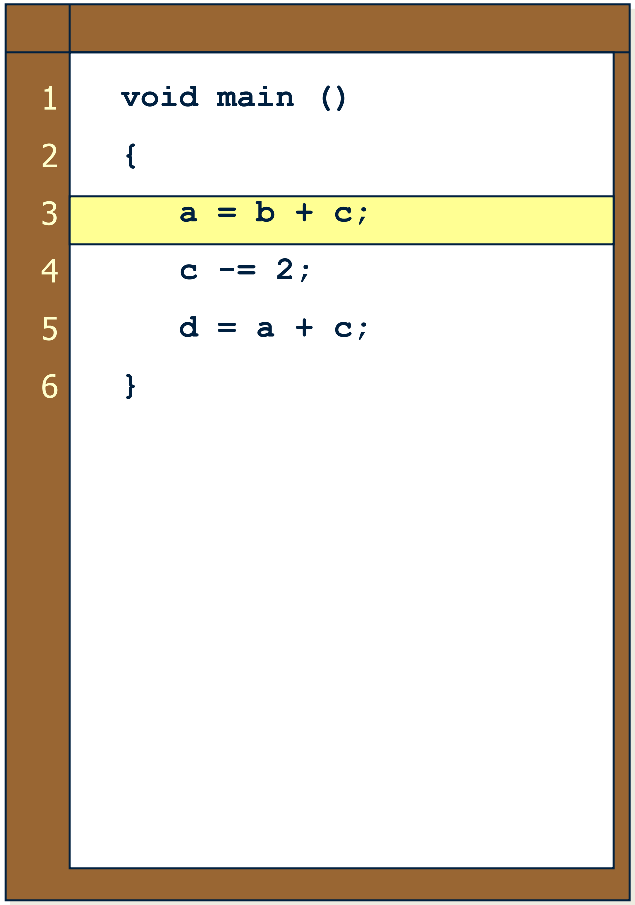
Instructor: Dr. Arbab Latif

NUST

SCHOOL OF ELECTRICAL ENGINEERING &
COMPUTER SCIENCE (SEECS)

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.



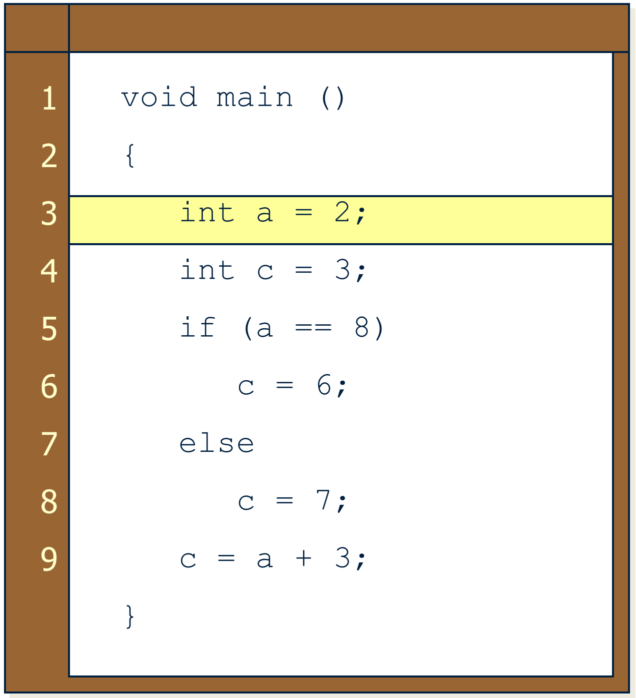
```
1   void main ()
2   {
3       a = b + c;
4       c -= 2;
5       d = a + c;
6   }
```

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)

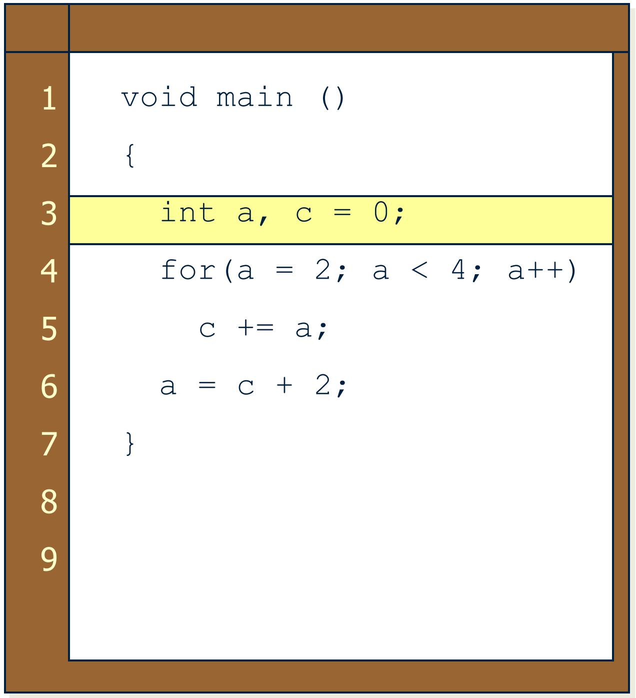
- Example 1: Not executing the next instruction, because of condition.
  - In the following example, the instruction of line 6 is not executed.



```
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 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)

- In the assembly language, there are 2 groups of instructions that make the CPU execute an instruction other than the next instruction.
  - These 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

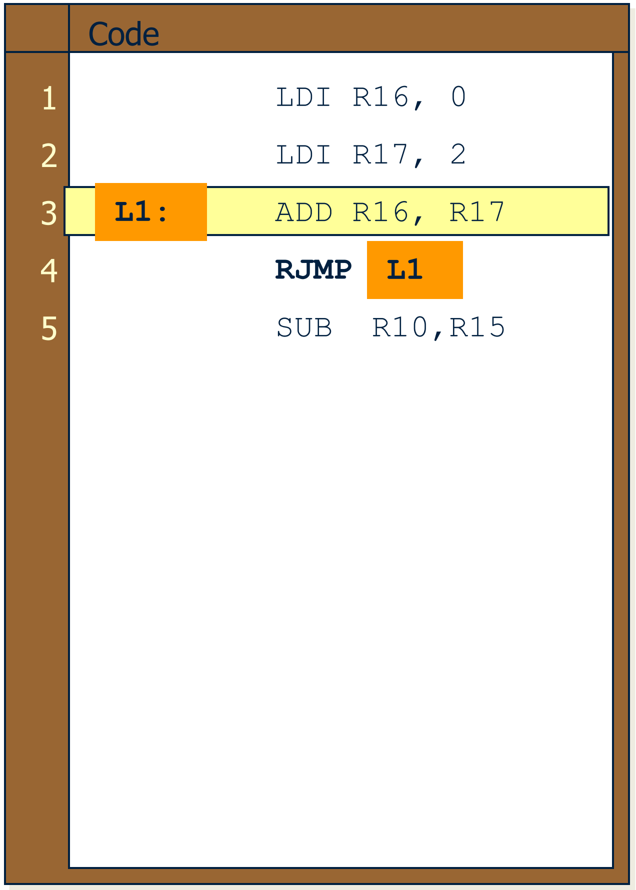
There are 2 kinds of Jump:

- 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

Unconditional Jump in AVR

- There are 3 unconditional jump instructions in AVR: **RJMP**, **JMP**, and **IJMP**
- 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.



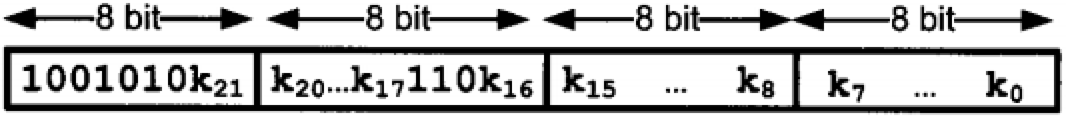
```
          LDI R16, 0
          LDI R17, 2
L1:       ADD R16, R17
          RJMP L1
          SUB  R10,R15
```

Ways of specifying the Jump Target

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

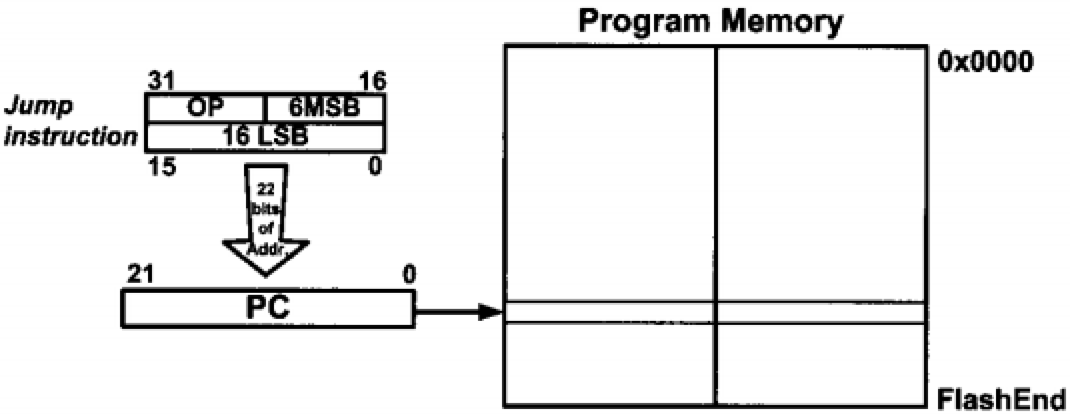
JMP

- JMP ⟷ PC = operand
  - Long Jump
  - 4-byte instruction
    - 10-bits for the opcode and rest 22-bit for the target address
    - 22-bit = 4M memory locations



8 bit	8 bit	8 bit	8 bit
1001010k_{21}	k_{20}…k_{17}110k_{16}	k_{15} … k_8	k_7 … k_0

$0 \le k < 4M$



JMP

- JMP ⟷ PC = operand

1001 010X XXXX 110X XXXX XXXX XXXX XXXX XXXX XXXX

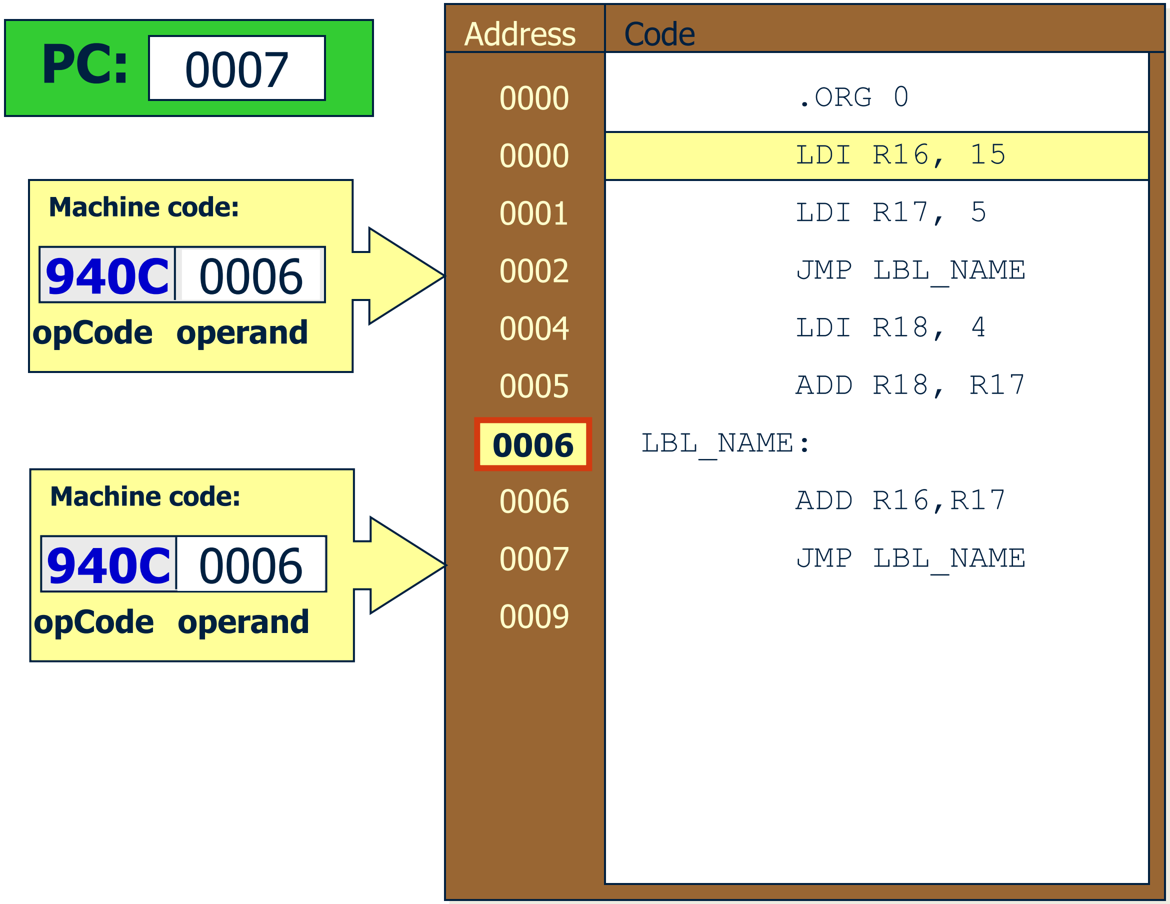
  - Example:

1001 0100 0000 1100 0000 0000 0000 0110

    - Operand = 0000000000000000000110

JMP

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



PC: 0007

Machine code:

940C 0006

opCode operand

Machine code:

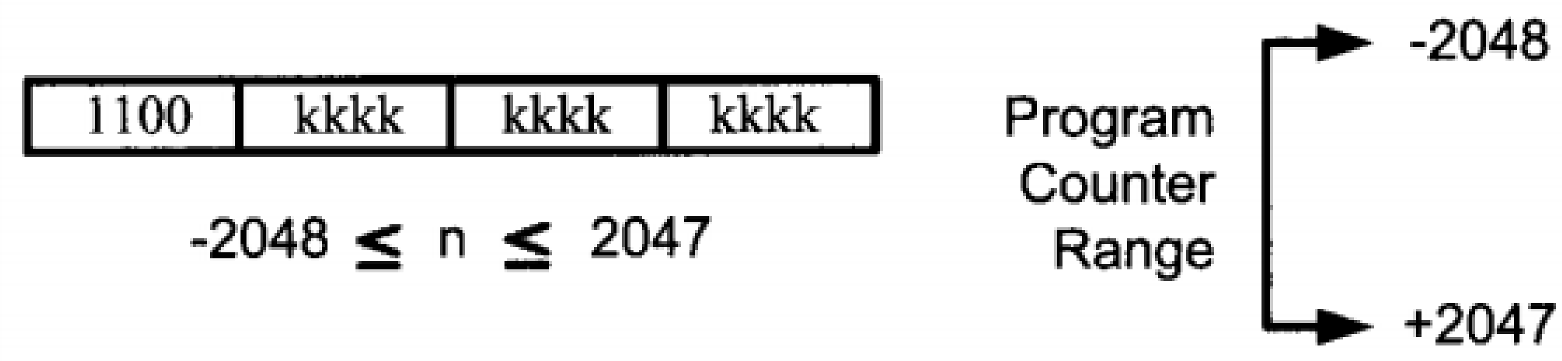
940C 0006

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
0009	

RJMP (Relative Jump)

- RJMP:
  - 2-byte instruction
  - Lower 12-bits for the relative address of the target
    - Range divided into forward and backward jumps.



1100	kkkk	kkkk	kkkk

$-2048 \leq n \leq 2047$

RJMP (Relative jump)

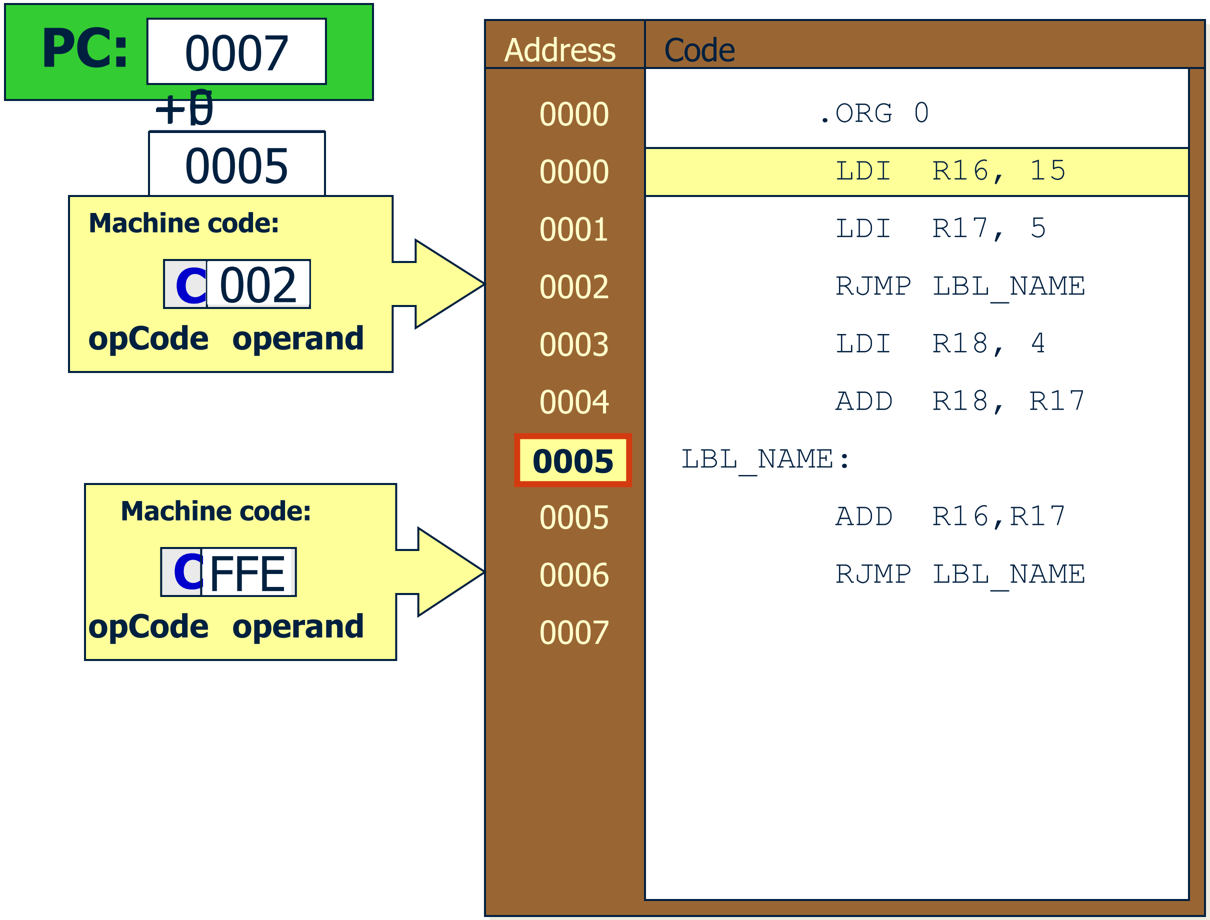
- RJMP ↔ PC = PC + operand

1100 xxxx xxxx xxxx

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

RJMP

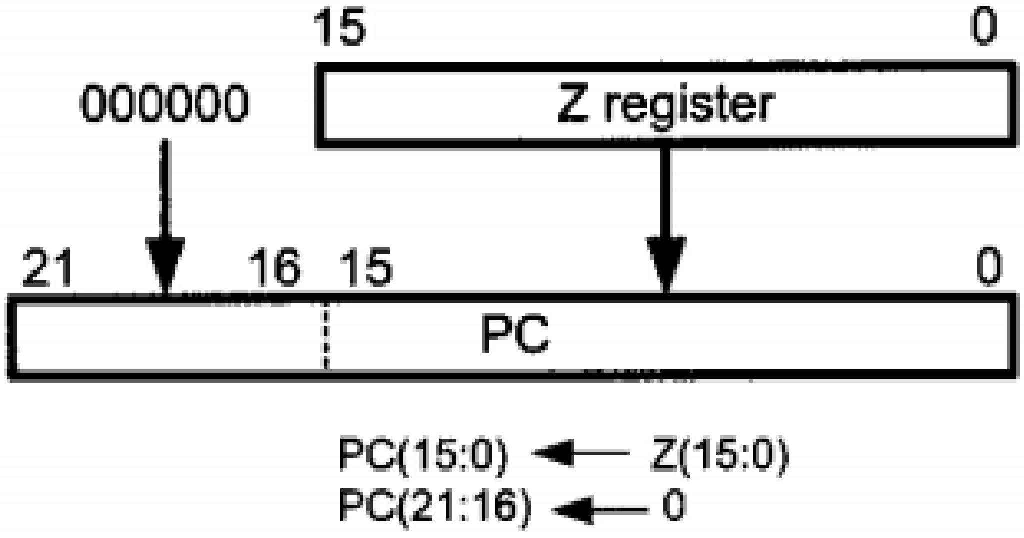
- When RJMP is executed:
  - The operand will be added to the current value of PC



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	

IJMP (Indirect Jump)

- IJMP: ⟷ PC = Z register
  - 2-byte instruction
  - PC is loaded with the contents of Z-register
    - So jumps to the address pointed to by the Z-register
      - For example, if Z points to location 100, by executing IJMP, the CPU jumps to location 100
    - The instruction has no operand



Conditional Jump

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 and
- Loop

Looping Instructions

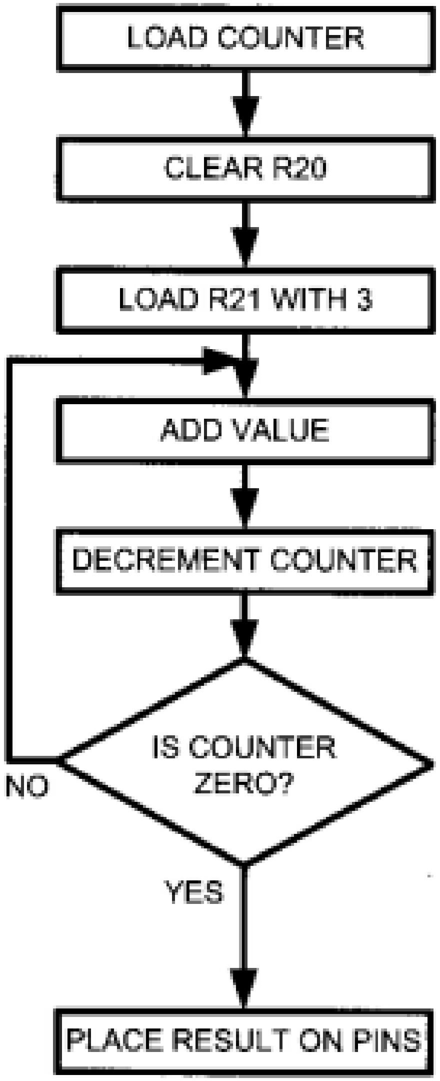
Looping: Using BRNE

- BRNE [BRnach if Not Equal]:
  - uses the *ZERO* flag [Z=0]

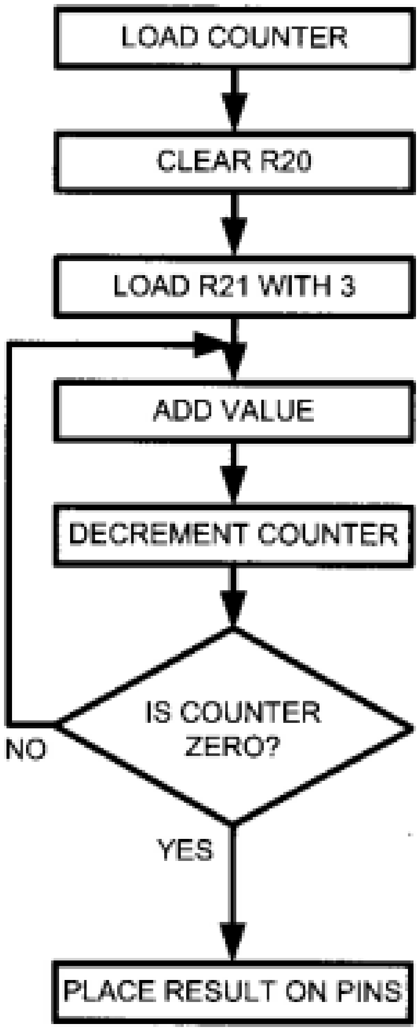
```
LDI R16, 0
LDI R17, 3
LDI R18, 5 ; counter
AGAIN: ADD R16, R17
       DEC R18
      BRNE AGAIN
```

Looping: Example

- Write a program to:
  a. Clear R20
  b. Add 3 to R20 ten times
  c. And the send the sum to PORTB



Looping: Example

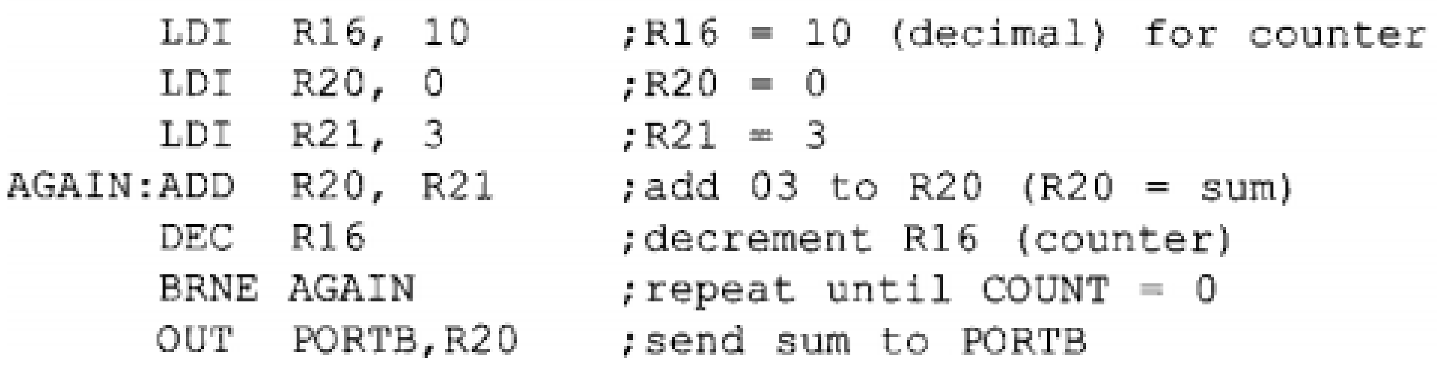


INSTRUCTIONS

```
        LDI  R16, 10
        LDI  R20, 0
        LDI  R21, 3
AGAIN:  ADD R20, R21
        DEC R16
        BRNE AGAIN
        OUT  PORTB, R20
```

Looping: Example Overall

- Write a program to:
  a. Clear R20
  b. Add 3 to R20 ten times
  c. And the send the sum to PORTB



```
      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
```

Reading

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

THANK YOU