به نام خدا

آشنایی با زبان اسمبلی AVR

دستورات پایه (2)

Dr. Aref Karimiafshar
A.karimiafshar@ec.iut.ac.ir

CLR – Clear Register

- Clears a register. This instruction performs an Exclusive OR between a register and itself. This will clear all bits in the register.

Operation:

(i) Rd ← Rd ⊕ Rd

	Syntax:	Operands:	Program Counter:
(i)	CLR Rd	0 ≤ d ≤ 31	PC ← PC + 1

16-bit Opcode: (see EOR Rd,Rd)

0010	01dd	dddd	dddd

Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	0	0	0	1	–

Words	1 (2 bytes)
Cycles	1

دستور CLR: این دستور محتویات یک رجیستر رو صفر میکنه که این کارو با xor کردن اون رجیستر با خودش انجام میده
z رو همیشه یک می کنه چون مقدار این رجیستر همیشه صفر میشه

INC – Increment

- Adds one -1- to the contents of register Rd and places the result in the destination register Rd. The C Flag in SREG is not affected by the operation.

Operation:

(i) Rd ← Rd + 1

	Syntax:	Operands:	Program Counter:
(i)	INC Rd	0 ≤ d ≤ 31	PC ← PC + 1

16-bit Opcode:

1001	010d	dddd	0011

Status Register and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	⇔	⇔	⇔	⇔	–

Words	1 (2 bytes)
Cycles	1

این دستور مقدار یک رجیستر رو یک واحد افزایش میده
این دستور روی فلگ کری تاثیرگذار نیست: نکته

INC – Increment

S $N \oplus V$, for signed tests.

V $R7 \cdot \overline{R6} \cdot \overline{R5} \cdot \overline{R4} \cdot \overline{R3} \cdot \overline{R2} \cdot \overline{R1} \cdot \overline{R0}$

Set if two's complement overflow resulted from the operation; cleared otherwise. Two's complement overflow occurs if and only if Rd was $7F before the operation.

N R7

Set if MSB of the result is set; cleared otherwise.

Z $\overline{R7} \cdot \overline{R6} \cdot \overline{R5} \cdot \overline{R4} \cdot \overline{R3} \cdot \overline{R2} \cdot \overline{R1} \cdot \overline{R0}$

Set if the result is $00; cleared otherwise.

فرمول رجیستر وضعیت:

DEC – Decrement

- Subtracts one -1- from the contents of register Rd and places the result in the destination register Rd. The C Flag in SREG is not affected by the operation.

Operation:

(i) Rd ← Rd - 1

	Syntax:	Operands:	Program Counter:
(i)	DEC Rd	0 ≤ d ≤ 31	PC ← PC + 1

16-bit Opcode:

1001	010d	dddd	1010

Status Register and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	⇔	⇔	⇔	⇔	–

Words	1 (2 bytes)
Cycles	1

این دستور مقدار رجیستر رو یک واحد کاهش میده
نکته: روی فلگ کری هم تاثیر نمی ذاره

DEC – Decrement

S $N \oplus V$, for signed tests.

V $\overline{R7} \cdot R6 \cdot R5 \cdot R4 \cdot R3 \cdot R2 \cdot R1 \cdot R0$

Set if two's complement overflow resulted from the operation; cleared otherwise. Two's complement overflow occurs if and only if Rd was $80 before the operation.

N R7

Set if MSB of the result is set; cleared otherwise.

Z $\overline{R7} \cdot \overline{R6} \cdot \overline{R5} \cdot \overline{R4} \cdot \overline{R3} \cdot \overline{R2} \cdot \overline{R1} \cdot \overline{R0}$

Set if the result is $00; cleared otherwise.

فرمول رجیستر وضعیت:

COM – One's Complement

- This instruction performs a One's Complement of register Rd.

Operation:

(i) Rd ← $FF - Rd

	Syntax:	Operands:	Program Counter:
(i)	COM Rd	0 ≤ d ≤ 31	PC ← PC + 1

16-bit Opcode:

1001	010d	dddd	0000

Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	⇔	0	⇔	⇔	1

Words	1 (2 bytes)
Cycles	1

این دستور روی محتوای یک رجیستر انجام میشه
و کاری که انجام می ده اینه که مکمل یک محتوای رجیستر Rd رو محاسبه می کنه

COM – One's Complement

S N ⊕ V, for signed tests.

V 0

Cleared.

N R7

Set if MSB of the result is set; cleared otherwise.

Z $\overline{R7} \cdot \overline{R6} \cdot \overline{R5} \cdot \overline{R4} \cdot \overline{R3} \cdot \overline{R2} \cdot \overline{R1} \cdot \overline{R0}$

Set if the result is $00; cleared otherwise.

C 1

Set.

فرمول رجیستر وضعیت:

NEG – Two's Complement

- Replaces the contents of register Rd with its two's complement; the value $80 is left unchanged.

Operation:

(i) Rd ← $00 - Rd

	Syntax:	Operands:	Program Counter:
(i)	NEG Rd	0 ≤ d ≤ 31	PC ← PC + 1

16-bit Opcode:

1001	010d	dddd	0001

Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	⇔	⇔	⇔	⇔	⇔	⇔

Words	1 (2 bytes)
Cycles	1

این دستور مکمل دو محتوای یک رجیستر رو حساب میکنه

NEG – Two's Complement

H $P3 + \overline{Rd3}$

Set if there was a borrow from bit 3; cleared otherwise.

S $N \oplus V$, for signed tests.

V $R7 \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$

Set if there is a two's complement overflow from the implied subtraction from zero; cleared otherwise. A two's complement overflow will occur if and only if the contents of the Register after operation (Result) is $80.

N R7

Set if MSB of the result is set; cleared otherwise.

Z $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$

Set if the result is $00; cleared otherwise.

C $R7 + R6 + R5 + R4 + R3 + R2 + R1 + R0$

Set if there is a borrow in the implied subtraction from zero; cleared otherwise. The C Flag will be set in all cases except when the contents of Register after operation is $00.

فرمول رجیستر وضعیت:

ROL – Rotate Left trough Carry

- Shifts all bits in Rd one place to the left. The C Flag is shifted into bit 0 of Rd. Bit 7 is shifted into the C Flag.

	Syntax:	Operands:	Program Counter:
(i)	ROL Rd	0 ≤ d ≤ 31	PC ← PC + 1

16-bit Opcode: (see ADC Rd,Rd)

0001	11dd	dddd	dddd

Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	⇔	⇔	⇔	⇔	⇔	⇔

Words	1 (2 bytes)
Cycles	1

این دستور میاد چرخش انجام میده روی محتوا با استفاده از کری ینی چرخش رو با استفاده از کری انجام میده و نحوه کارش به این صورت است که محتوای رجیستر رو یک بیت به سمت چپ از طریق کری انجام میده ینی کری میاد توی بیت صفر قرار می گیره و همینجور بیت به بیت این ها به سمت چپ منتقل میشه و بیت هفتم هم به کری منتقل میشه

ROL – Rotate Left trough Carry

H Rd3

S N ⊕ V, for signed tests.

V N ⊕ C, for N and C after the shift.

N R7

Set if MSB of the result is set; cleared otherwise.

Z $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$

Set if the result is $00; cleared otherwise.

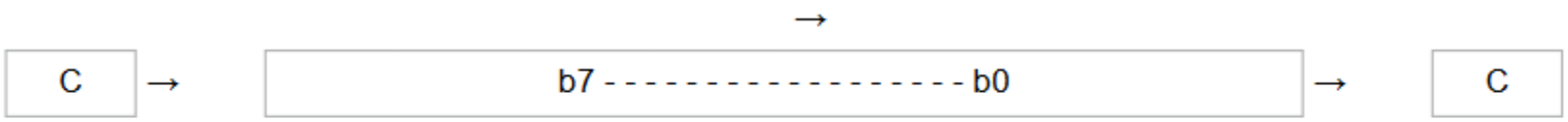
C Rd7

Set if, before the shift, the MSB of Rd was set; cleared otherwise.

فرمول رجیستر وضعیت:

ROR – Rotate Right through Carry

- Shifts all bits in Rd one place to the right. The C Flag is shifted into bit 7 of Rd. Bit 0 is shifted into the C Flag.



Syntax:	Operands:	Program Counter:
(i) ROR Rd	0 ≤ d ≤ 31	PC ← PC + 1

16-bit Opcode:

1001	010d	dddd	0111

Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	⇔	⇔	⇔	⇔	⇔

Words	1 (2 bytes)
Cycles	1

این دستور میاد چرخش رو به سمت راست انجام میده
این بار محتوای کری به بیت هفتم منتقل میشه و هر بیت به سمت راست شیفت پیدا میکنه و بیت صفرم به کری منتقل خواهد شد

ROR – Rotate Right through Carry

S $N \oplus V$, for signed tests.

V $N \oplus C$, for N and C after the shift.

N R7

Set if MSB of the result is set; cleared otherwise.

Z $\overline{R7} \cdot \overline{R6} \cdot \overline{R5} \cdot \overline{R4} \cdot \overline{R3} \cdot \overline{R2} \cdot \overline{R1} \cdot \overline{R0}$

Set if the result is $00; cleared otherwise.

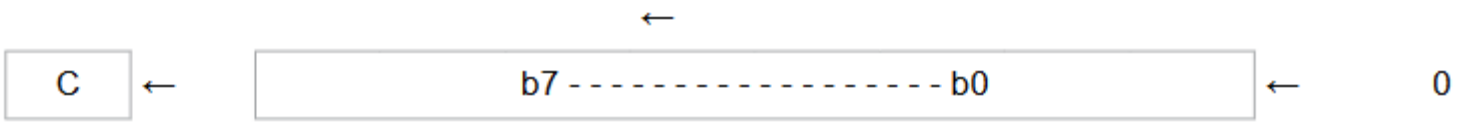
C Rd0

Set if, before the shift, the LSB of Rd was set; cleared otherwise.

فرمول رجیستر وضعیت:

LSL – Logical Shift Left

- Shifts all bits in Rd one place to the left. Bit 0 is cleared. Bit 7 is loaded into the C Flag of the SREG. This operation effectively multiplies signed and unsigned values by two.



Syntax:	Operands:	Program Counter:
(i) LSL Rd	0 ≤ d ≤ 31	PC ← PC + 1

16-bit Opcode: (see ADD Rd,Rd)

0000	11dd	dddd	dddd

Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	⇔	⇔	⇔	⇔	⇔	⇔

Words	1 (2 bytes)
Cycles	1

این دستور میاد یک شیفت روی محتوای یک رجیستر به سمت چپ انجام میده ینی صفر میاد در بیت صفرم این رجیستر قرار میگیره و محتوای هر بیت به بیت بعدی منتقل میشه و اخرین بیت هم به کری منتقل می شه
از این دستور می تونیم برای اینکه یک عدد رو در 2 ضرب بکنیم استفاده میکنیم عملا شیفت به سمت چپ با استفاده از این دستور مثل این می مونه که انگار اون عدد رو داریم در 2 ضرب میکنیم

LSL – Logical Shift Left

H Rd3

S N ⊕ V, for signed tests.

V N ⊕ C, for N and C after the shift.

N R7

Set if MSB of the result is set; cleared otherwise.

Z $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$

Set if the result is $00; cleared otherwise.

C Rd7

Set if, before the shift, the MSB of Rd was set; cleared otherwise.

فرمول رجیستر وضعیت:

LSR – Logical Shift Right

- Shifts all bits in Rd one place to the right. Bit 7 is cleared. Bit 0 is loaded into the C Flag of the SREG. This operation effectively divides an unsigned value by two. The C Flag can be used to round the result.

Syntax:	Operands:	Program Counter:
(i) LSR Rd	0 ≤ d ≤ 31	PC ← PC + 1

16-bit Opcode:

1001	010d	dddd	0110

Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	⇔	⇔	0	⇔	⇔

Words	1 (2 bytes)
Cycles	1

این دستور شیفت می ده به سمت راست به این صورت که صفر از سمت راست وارد میشه و هر بیت به سمت راست شیفت پیدا می کنه و اخرین بیت به کری منتقل میشه

LSR – Logical Shift Right

S $N \oplus V$, for signed tests.

V $N \oplus C$, for N and C after the shift.

N 0

Z $\overline{R7} \cdot \overline{R6} \cdot \overline{R5} \cdot \overline{R4} \cdot \overline{R3} \cdot \overline{R2} \cdot \overline{R1} \cdot \overline{R0}$

Set if the result is $00; cleared otherwise.

C Rd0

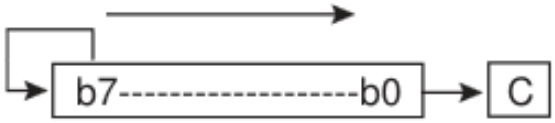
Set if, before the shift, the LSB of Rd was set; cleared otherwise.

فرمول رجیستر وضعیت:

ASR – Arithmetic Shift Right

- Shifts all bits in Rd one place to the right. Bit 7 is held constant. Bit 0 is loaded into the C Flag of the SREG. This operation effectively divides a signed value by two without changing its sign. The Carry Flag can be used to round the result.

(i)



	Syntax:	Operands:	Program Counter:
(i)	ASR Rd	0 ≤ d ≤ 31	PC ← PC + 1

16-bit Opcode:

1001	010d	dddd	0101

Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	⇔	⇔	⇔	⇔	⇔

Words	1 (2 bytes)
Cycles	1

این دستور میاد بیت هفتم رو به خودش منتقل می کنه و بیت هفتم هم معمولا برای علامت هم مورد استفاده قرار می گیره و بقیه بیت ها هم به سمت راست شیفت پیدا می کنند و بیت اخر هم به کری منتقل میشه
شیفت به سمت راست در اعداد علامت دار و بدون علامت می تونه به معنای تقسیم بر 2 باشه
از ASR که استفاده میکنیم می تونه به این منظور باشه که یک عدد علامت دار رو داریم بر 2 تقسیم می کنیم

ASR – Arithmetic Shift Right

S N ⊕ V, for signed tests.

V N ⊕ C, for N and C after the shift.

N R7

Set if MSB of the result is set; cleared otherwise.

Z $\overline{R7} \cdot \overline{R6} \cdot \overline{R5} \cdot \overline{R4} \cdot \overline{R3} \cdot \overline{R2} \cdot \overline{R1} \cdot \overline{R0}$

Set if the result is $00; cleared otherwise.

C Rd0

Set if, before the shift, the LSB of Rd was set; cleared otherwise.

فرمول رجیستر وضعیت:

SWAP – Swap Nibbles

- Swaps high and low nibbles in a register.

(i) R(7:4) ← Rd(3:0), R(3:0) ← Rd(7:4)

	Syntax:	Operands:	Program Counter:
(i)	SWAP Rd	0 ≤ d ≤ 31	PC ← PC + 1

16-bit Opcode:

1001	010d	dddd	0010

Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Words	1 (2 bytes)
Cycles	1

این دستور میاد نیمه پایین و نیمه بالا رو در یک رجیستر همه منظوره با هم جا به جا میکنه ینی اگر یک رجیستر مثل Rd داشته باشیم میاد جای اینارو با هم عوض میکنه و داخل اون رجیستر قرار میده

Summary

Some Instructions Using a GPR as Operand

Instruction		
CLR	Rd	Clear Register Rd
INC	Rd	Increment Rd
DEC	Rd	Decrement Rd
COM	Rd	One's Complement Rd
NEG	Rd	Negative (two's complement) Rd
ROL	Rd	Rotate left Rd through carry
ROR	Rd	Rotate right Rd through carry
LSL	Rd	Logical Shift Left Rd
LSR	Rd	Logical Shift Right Rd
ASR	Rd	Arithmetic Shift Right Rd
SWAP	Rd	Swap nibbles in Rd

Instructions that affect Flag Bits

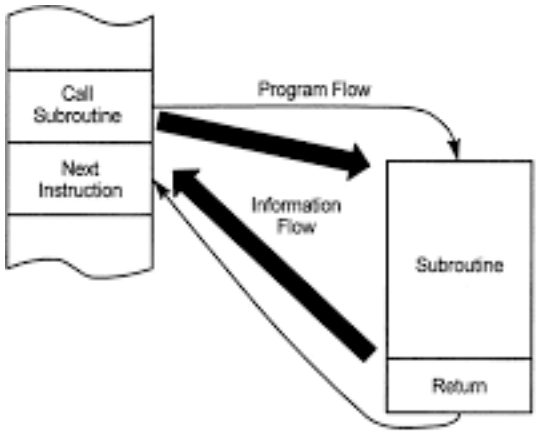
یکسری دستورات داریم که باعث میشه این پرچم ها هم تغییر بکنند

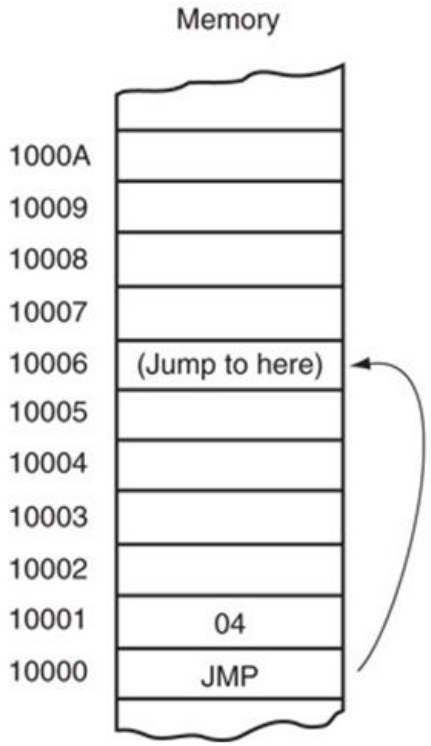
Instruction	C	Z	N	V	S	H
ADD	X	X	X	X	X	X
ADC	X	X	X	X	X	X
ADIW	X	X	X	X	X	
AND		X	X	X	X	
ANDI		X	X	X	X	
CBR		X	X	X	X	
CLR		X	X	X	X	
COM	X	X	X	X	X	
DEC		X	X	X	X	
EOR		X	X	X	X	
FMUL	X	X				
INC		X	X	X	X	
LSL	X	X	X	X		X
LSR	X	X	X	X		
OR		X	X	X	X	
ORI		X	X	X	X	
ROL	X	X	X	X		X
ROR	X	X	X	X		
SEN			1			
SEZ		1				
SUB	X	X	X	X	X	X
SUBI	X	X	X	X	X	X
TST		X	X	X	X	

Note: X can be 0 or 1.

دستورات پرش

- In the sequence of instruction to be executed, it is often necessary to transfer program control to a different location.
- Instruction in AVR to achieve this:
  - Branches (JUMP)
  - CALL





دستورات پرش رو کجاها داریم؟
دستورات پرش اون روال عادی اجرای برنامه ها رو تغییر میدن در حالت عادی به صورت ترتیبی از اول حافظه دستورات یکی یکی شروع می کنن به اجرا شدن و این ادامه پیدا میکنه ولی زمانی که از دستور پرش استفاده میکنیم زمان اجرای برنامه رو تغییر میدیم
دوتا دستوری که اینجا داریم:
1- branches
2- CALL
در حالت Branches (JUMP) روال عادی برنامه که داره طی میکشه به دستور پرش که می رسیم این PC ما رو تغییر میده و به یک نقطه مشخصی از حافظه منتقل میشه و از اونجا دستورات شروع می کنن به اجرا شدن
توی CALL روال اجرای برنامه منتقل میشه به یک subrotin اونجا یکسری کارهایی انجام میشه و دوباره برگشت داده می شیم به همون روال عادی مثلا دستور CALL رو که داشتیم اجرا می کردیم به دستور بعدیش منتقل می شیم و روال عادی ادامه پیدا خواهد کرد توی این حالت باید ادرس بازگشت رو در جایی ذخیره بکنیم تا بتونیم ادامه اون برنامه ای که داشته اجرا میشده رو پی بگیریم

حلقه

- Repeating a sequence of instructions or an operation a certain number of times.
  1. Repeat the instructions over and over!

```
LDI R16,0          ;R16 = 0
LDI R17,3          ;R17 = 3
ADD R16,R17        ;add value 3 to R16 (R16 = 0x03)
ADD R16,R17        ;add value 3 to R16 (R16 = 0x06)
ADD R16,R17        ;add value 3 to R16 (R16 = 0x09)
ADD R16,R17        ;add value 3 to R16 (R16 = 0x0C)
ADD R16,R17        ;add value 3 to R16 (R16 = 0x0F)
ADD R16,R17        ;add value 3 to R16 (R16 = 0x12)
```

Too much code space would be needed!

- A much better way is to use a *LOOP*

یکی از عملکرد ها و کاربردهای مهمی که دستورات پرش برای ما دارند اینه که وقتی که می خوایم یک دنباله ای از دستورات رو اجرا بکنیم که تکراری اند ینی یک عملیاتی رو می خوایم انجام بدیم مثلا 10 بار انجام دادن یک جمع مشخص برای این کار از حلقه ها استفاده می کنند
حلقه ها برمبنای دستورات پرش ساخته می شن
یک حلقه از یک نقطه ای شروع میشه و یک بدنه داره و دوباره بازگشت پیدا میکنه و این کار رو تکرار میکنه --> صفحه بعدی

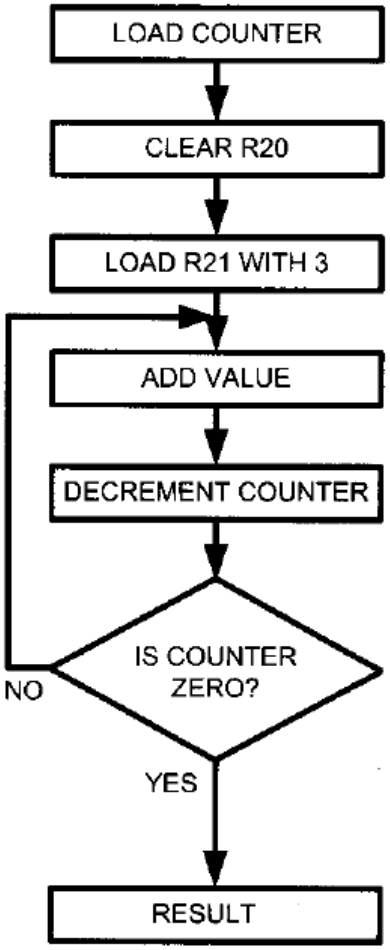
حلقه

- Structure

```
BACK: .........   ;start of the loop
      .........   ;body of the loop
      .........   ;body of the loop
      DEC Rn      ;decrement Rn, Z = 1 if Rn = 0
      BRNE BACK   ;branch to BACK if Z = 0
```

- Example

```
      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
```



BACK شروع حلقه رو مشخص میکنه که یک label است و می تونه هر چیز دیگه ای هم باشه
بعد اعمالی که میخوایم انجام بدیم
بعد یک کانتر یا شمارنده ای داریم که یک عملیاتی وابسته به نوع حلقمون روش انجام میدیم:
DEC Rn
بعد چک میکنیم یک سری پرچم ها رو و بعد براساس موقعیت هایی که این پرچم ها و شرایطی که دارند می تونیم انشعاب بدیم به اون نقطه اولیه که حلقه ادامه پیدا بکنه یا اینکه از حلقه خارج بشیم:
BRNE BACK

مثال: می خوایم 10 بار یک جمعی رو انجام بدیم:
R16 کانتر ما است و در هر بار عملیاتی که انجام میدیم از R16 یک واحد کم میکنیم و چک میکنیم اگر R16 که اخرین عملیات روش انجام شده باشه صفر شده باشه از این حلقه خارج میشیم در غیر اینصورت این حلقه ادامه پیدا میکنه
تعداد باری که می تونیم این حلقه رو تکرار بکنیم وابسته به رجیستر R16 است که روش داریم کنترل می ذاریم تا روی اون پرچم تاثیر بذاره

BRNE – Branch if Not Equal

- Conditional relative branch. Tests the Zero Flag (Z) and branches relatively to PC if Z is cleared.

(i) If Rd ≠ Rr (Z = 0) then PC ← PC + k + 1, else PC ← PC + 1

	Syntax:	Operands:	Program Counter:
(i)	BRNE k	-64 ≤ k ≤ +63	PC ← PC + k + 1 PC ← PC + 1, if condition is false

16-bit Opcode:

1111	01kk	kkkk	k001

Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Words	1 (2 bytes)
Cycles	1 if condition is false 2 if condition is true

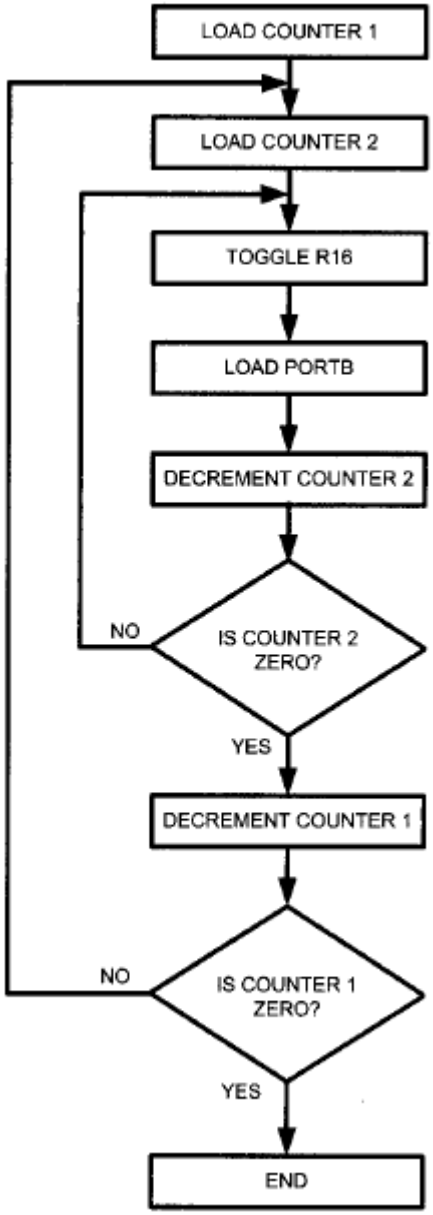
این دستور میاد مقایسه میکنه مثلا اگر این دو تا عدد با هم برابر نباشند: PC ما با یک مقدار مشخصی اضافه میشه و منتقل میشه به اون نقطه
پس BRNE یک دستور پرش شرطی هستش و میاد مقدار Z رو چک میکنه و براساس اون تصمیم میگیره که ایا روند عادی اجرای برنامه ادامه پیدا کنه و یا اینکه از روند عادی اجرا خارج بشه و به یک نقطه دیگری منتقل بشیم این کار با چک کردن پرچم Z انجام میشه و اگر Z برابر با 0 باشه باعث میشه PC با اون عددی که مشخص شده جمع میشه و اون نقطه مورد نظر را به ما میده
K یک عدد 7 بیتی علامتدار است
با دستور BRNE می تونیم به 64 خونه جلوتر یا 64 خونه عقب تر جابه جا بشیم
اگر شرط برقرار باشه PC ما با K جمع میشه و +1 و منتقل میشه به اون خونه در غیر اینطورت مثل قبل عمل میکنه
نکته ای که وجود داره اینه که باید یک منطق برای تاثیرگذاری بر روی این پرچم Z قبل از چک کردن و رسیدن به دستور BRNE K دیده باشیم توی کد

حلقه‌های تو در تو

- Maximum number of times a loop can be repeated
  - Limited

- Example

```
      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
```

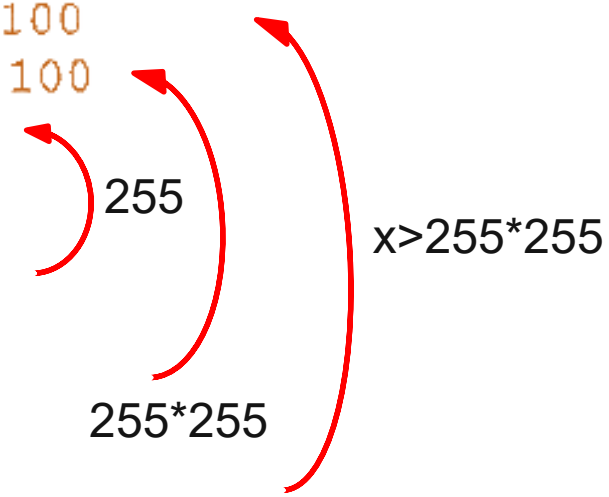


با یک حلقه چون فقط یک رجیستر داریم محدودیم به 255 و اگر بخوایم بیشتر از این تعداد بار تکرار داشته باشیم می تونیم از حلقه های تودرتو استفاده بکنیم
چون الان دوتا حلقه تودرتو داریم پس 255*255 بار می تونیم این حلقه تودرتو رو تکرار بکنیم

حلقه‌های تو در تو

- Number of times > 255* 255 (=65,025)
  - Three nested loop
- Example

```
        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
```



اگر بخوایم تعداد تکرار یک حلقه رو افزایش بدیم تعداد حلقه های تودرتو رو باز زیاد میکنیم

دستورات پرش

- AVR conditional branch (jump) instructions

Branch (Jump) Instructions

Instruction	Action
BRLO	Branch if C = 1
BRSH	Branch if C = 0
BREQ	Branch if Z = 1
BRNE	Branch if Z = 0
BRMI	Branch if N = 1
BRPL	Branch if N = 0
BRVS	Branch if V = 1
BRVC	Branch if V = 0

دستورات صفحه روبرو رو دستورات پرش شرطی میگیم چون برمبنای یک شرطی این ها تصمیم میگیرند پرش داشته باشند یا نداشته باشند
صفحه های بعدی اینارو توضیح میدیم

BRLO – Branch if Lower (Unsigned)

- Conditional relative branch. Tests the Carry Flag (C) and branches relatively to PC if C is set.

(i) If Rd < Rr (C = 1) then PC ← PC + k + 1, else PC ← PC + 1

	Syntax:	Operands:	Program Counter:
(i)	BRLO k	-64 ≤ k ≤ +63	PC ← PC + k + 1 PC ← PC + 1, if condition is false

16-bit Opcode:

1111	00kk	kkkk	k000

Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Words 1 (2 bytes)

Cycles 1 if condition is false

2 if condition is true

BRLO: انشعاب در صورتی که کوچکتر باشد
توی این دستور میاد پرچم C رو چک میکنه پس قبل از اینکه این دستور بخواد اجرا بشه باید یک کنترل روی C داشته باشیم و مشخص بکنیم که C چجوری باید کنترل بشه
اگر C فعال شده باشه اون پرش رو انجام میده
K یک عدد 7 بیتی علامتدار است و می تونیم به 64 خونه جلوتر یا 64 خونه عقب تر پرش بکنیم

BRSH – Branch if Same or Higher (Unsigned)

- Conditional relative branch. Tests the Carry Flag (C) and branches relatively to PC if C is cleared.

(i) If Rd ≥Rr (C = 0) then PC ← PC + k + 1, else PC ← PC + 1

	Syntax:	Operands:	Program Counter:
(i)	BRSH k	-64 ≤ k ≤ +63	PC ← PC + k + 1 PC ← PC + 1, if condition is false

16-bit Opcode:

1111	01kk	kkkk	k000

Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Words 1 (2 bytes)

Cycles 1 if condition is false

2 if condition is true

شاخه نسبی مشروط. پرچم حمل (C) را آزمایش می کند و در صورت پاک شدن C، نسبتاً به رایانه منشعب می شود

BREQ – Branch if Equal

- Conditional relative branch. Tests the Zero Flag (Z) and branches relatively to PC if Z is set.

(i) If Rd = Rr (Z = 1) then PC ← PC + k + 1, else PC ← PC + 1

	Syntax:	Operands:	Program Counter:
(i)	BREQ k	-64 ≤ k ≤ +63	PC ← PC + k + 1 PC ← PC + 1, if condition is false

16-bit Opcode:

1111	00kk	kkkk	k001

Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Words	1 (2 bytes)
Cycles	1 if condition is false 2 if condition is true

یک دستور پرش شرطی بر مبنای کنترل پرچم Z هستش

BRMI – Branch if Minus

- Conditional relative branch. Tests the Negative Flag (N) and branches relatively to PC if N is set.

(i) If N = 1 then PC ← PC + k + 1, else PC ← PC + 1

	Syntax:	Operands:	Program Counter:
(i)	BRMI k	-64 ≤ k ≤ +63	PC ← PC + k + 1 PC ← PC + 1, if condition is false

16-bit Opcode:

1111	00kk	kkkk	k010

Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Words	1 (2 bytes)
Cycles	1 if condition is false 2 if condition is true

این دستور میاد پرچم N رو چک میکنه

BRPL – Branch if Plus

- Conditional relative branch. Tests the Negative Flag (N) and branches relatively to PC if N is cleared.

(i) If N = 0 then PC ← PC + k + 1, else PC ← PC + 1

	Syntax:	Operands:	Program Counter:
(i)	BRPL k	-64 ≤ k ≤ +63	PC ← PC + k + 1 PC ← PC + 1, if condition is false

16-bit Opcode:

1111	01kk	kkkk	k010

Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Words 1 (2 bytes)

Cycles 1 if condition is false

2 if condition is true

شاخه نسبی مشروط. پرچم منفی (N) را آزمایش می کند و در صورت پاک شدن N نسبتاً به PC منشعب می شود.

BRVC – Branch if Overflow Cleared

- Conditional relative branch. Tests the Overflow Flag (V) and branches relatively to PC if V is cleared.

(i) If V = 0 then PC ← PC + k + 1, else PC ← PC + 1

	Syntax:	Operands:	Program Counter:
(i)	BRVC k	-64 ≤ k ≤ +63	PC ← PC + k + 1 PC ← PC + 1, if condition is false

16-bit Opcode:

1111	01kk	kkkk	k011

Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Words	1 (2 bytes)
Cycles	1 if condition is false 2 if condition is true

این دستور برمبنای پرچم V عمل میکنه

BRVS – Branch if Overflow Set

- Conditional relative branch. Tests the Overflow Flag (V) and branches relatively to PC if V is set.

(i) If V = 1 then PC ← PC + k + 1, else PC ← PC + 1

	Syntax:	Operands:	Program Counter:
(i)	BRVS k	-64 ≤ k ≤ +63	PC ← PC + k + 1 PC ← PC + 1, if condition is false

16-bit Opcode:

1111	00kk	kkkk	k011

Status Register (SREG) and Boolean Formula

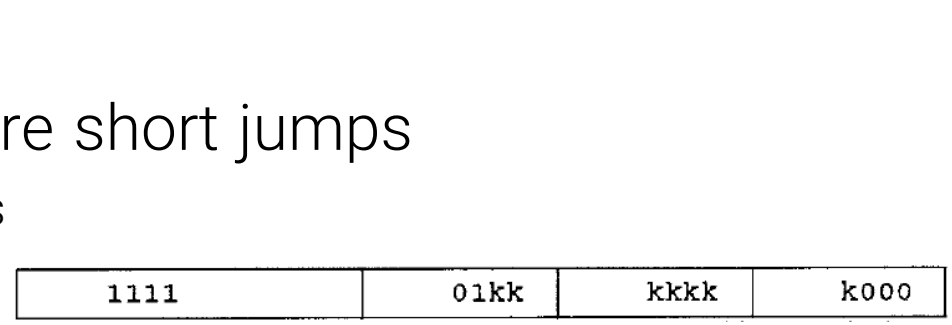
I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

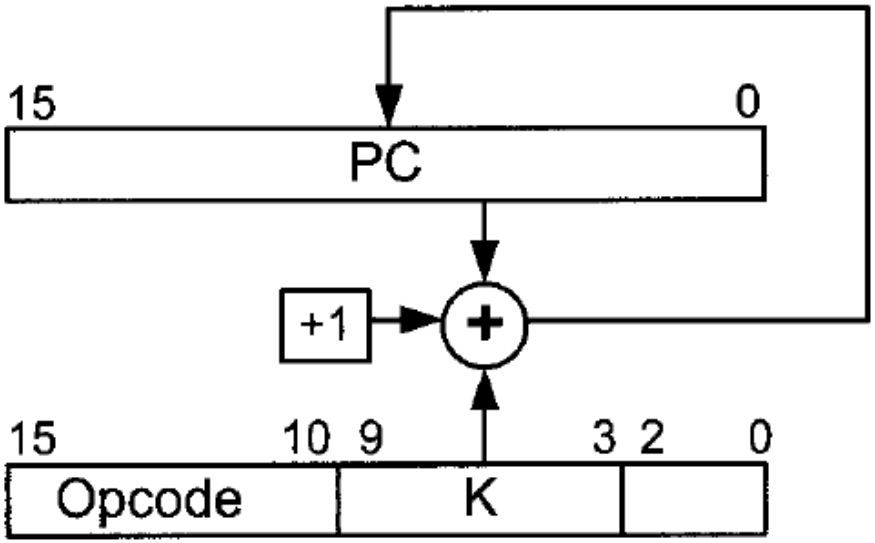
Words	1 (2 bytes)
Cycles	1 if condition is false 2 if condition is true

شاخه نسبی مشروط. پرچم سرریز (V) را آزمایش می کند و در صورت تنظیم V، نسبتاً به PC منشعب می شود

طول پرش

- All conditional branches are short jumps
  - They are all 2-byte instructions
    - The relative address is 7 bits
    - Relative to program counter
  - Address of target must be within 64 bytes of PC
    - Relative address, positive → jump forward
    - Relative address, negative→ jump backwards





تا اینجا پرش هایی که گفتیم پرش های شرطی بود و همشون اپکد 16 بیتی داشتن و اپرندی هم که دریافت می کرد یک اپرند 7 بیتی علامتدار بود
اگر K یک عدد مثبت باشه ما رو 64 خونه حداکثر می تونه جلو ببره و اگر این K منفی باشه ما می تونیم حداکثر 65 خونه به عقب برگردیم
پس عدد K هستش که تعیین میکنه ما چند خونه به جلو یا عقب بریم
اون منطقی که پشت تعیین ادرس جدیدمون هستش اینه که PC رو داریم مقدار این PC با K جمع میشه و به اضافه 1 میشه و چرا یک؟ به صورت عادی PC با یک جمع میشه و دستور بعدی رو به ما میده بنابراین اگر این K رو بهش اضافه بکنیم اون پرشی که ما میخوایم رو بهمون میده

پرش‌های غیر شرطی

- Unconditional branches (jumps)
  - Control unconditionally transferred to the target location
- AVR Unconditional branches
  - JMP
  - RJMP
  - IJMP

- Use depend on target address

پرش های غیرشرطی: این پرش ها در هر صورت پرش خودشون رو انجام میدن
اینکه از اون سه دستور که روبرو گفته در چه جایی استفاده بکنیم این وابسته به اون ادرس مقصدی هستش که ما می خوایم به اون پرش بکنیم و این سه دستور توی طول پرش هم با تفاوت دارند

JMP – Jump

- Jump to an address within the entire 4M (words) Program memory.

(i) PC ← k

	Syntax:	Operands:	Program Counter:	Stack:
(i)	JMP k	0 ≤ k < 4M	PC ← k	Unchanged

32-bit Opcode:

1001	010k	kkkk	110k
kkkk	kkkk	kkkk	kkkk

Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Words	2 (4 bytes)
Cycles	3

K اون محلی هستش که قصد داریم به اون پرش بکنیم
K به صورت مستقیم در PC جا می گیره
اپکد ما اینجا 32 بیتی هستش

RJMP – Relative Jump

- Relative jump to an address within PC - 2K +1 and PC + 2K (words).

(i) PC ← PC + k + 1

	Syntax:	Operands:	Program Counter:	Stack:
(i)	RJMP	k -2K ≤ k < 2K	PC ← PC + k + 1	Unchanged

16-bit Opcode:

1100	kkkk	kkkk	kkkk

Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Words 1 (2 bytes)

Cycles 2

این دستور به صورت نسبی پرش انجام میده
اپکد این دستور 16 بیتی هستش اما 12 بیت از اپکد برای مشخص کردن K به کار گرفته میشه پس دامنه پرش ما خیلی بزرگ است

IJMP – Indirect Jump

- Indirect jump to the address pointed to by the Z (16 bits) Pointer Register in the Register File.

(i)	PC ← Z(15:0)	Devices with 16-bit PC, 128KB Program memory maximum.
(ii)	PC(15:0) ← Z(15:0) PC(21:16) ← 0	Devices with 22-bit PC, 8MB Program memory maximum.

	Syntax:	Operands:	Program Counter:	Stack:
(i), (ii)	IJMP	None	See Operation	Not Affected

16-bit Opcode:

1001	0100	0000	1001

Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Words	1 (2 bytes)
Cycles	2

توی این حالت میاد از رجیستر خاص منظوره Z استفاده می کنه
این دستور اپرند نداره
وقتی این دستور می خواد اجرا بشه می ره سراغ Z و محتوای Z رو به عنوان PC کانتر جدید تلقی می کنه
پس دوتا ویژگی داره:
1- به اندازه 16 بیت ادرس بدیم و جابه جا بشیم
2- به صورت داینامیک می تونیم Z رو مشخص بکنیم نسبت به دو حالت قبل ولی اینجا Z هر مقداری که درونش باشه در PC ما قرار میگیره

پایان

موفق و پیروز باشید