AVR – Instrukční soubor

Aritmetické instrukce

- připomenutí: RISC =
  - sada GPR (AVR R0 až R31), každý je akumulátor
  - všechny ALU instrukce výhradně nad GPR
- chybí dělení, násobení pouze typy ATmega
- dále symbolika:
  - Rr, Rd = kterýkoli registr R0 až R31 (až na výjimky)
  - Re = kterýkoli registr R16 až R31, při immediate adresování (jeden operand je konstanta)
  - K6 = 6b konstanta 0-63 (nebo konstantní výraz)
  - K8 = 8b konstanta 0-255 (nebo konstantní výraz)
  - P = registry pro nepřímé adresování, P může být kterýkoli z X, Y ,Z
  - Q = registry pro nepřímé adresování, Q může být Y nebo Z
  - k = přímá adresa v RAM
  - (P) = nepřímá adresa v RAM, P (X, Y ,Z) je pointer ukazující do RAM
  - b = číslo bitu (0 až 7)

8 bitové sčítání, odčítání

ADD	Rd,Rr	Add without Carry	Rd = Rd + Rr	Z,C,N,V,H,S	1
ADC	Rd,Rr	Add with Carry	Rd = Rd + Rr + C	Z,C,N,V,H,S	1
SUB	Rd,Rr	Subtract without Carry	Rd = Rd - Rr	Z,C,N,V,H,S	1
SUBI	Re,K8	Subtract Immediate	Re = Re - K8	Z,C,N,V,H,S	1
SBC	Rd,Rr	Subtract with Carry	Rd = Rd - Rr - C	Z,C,N,V,H,S	1
SBCI	Re,K8	Subtract with Carry Immedtiate	Re = Re - K8 - C	Z,C,N,V,H,S	1
INC	Rd	Increment Register	Rd = Rd + 1	Z,N,V,S	1
DEC	Rd	Decrement Register	Rd = Rd -1	Z,N,V,S	1

16 bitové sčítání, odčítání

ADIW	Rdl,K6	Add Immediate to Word	Rd+1:Rd = Rd+1:Rd + K6	Z,C,N,V,S	2
SBIW	Rdl,K6	Subtract Immediate from Word	Rd+1:Rd = Rd+1:Rd - K6	Z,C,N,V,S	2

- jako Rd pouze R24, R26, R28, R30 (nebo XL, YL, ZL)
- pro výpočet adres při nepřímém adresování
- příklady

```
adiw  r25:r24,1   ; Add 1 to r25:r24
adiw  r24,1       ; Add 1 to r25:r24
adiw  ZH:ZL,60    ; Add 60 to the Z pointer(r31:r30)
```

Další operace

SER	Re	Set Register	Re = 0xFF	None	1
CLR	Rd	Clear Register	Rd = 0	Z,C,N,V,S	1
COM	Rd	One's Complement	Rd = 0xFF - Rd	Z,C,N,V,S	1
NEG	Rd	Two's Complement	Rd = 0x00 - Rd	Z,C,N,V,H,S	1

8 bitové násobení

- jen ATmega!!!
- R1 → b15 až b8, R0 → b7 až b0 výsledku
- signed → ve dvojkovém doplňku
- fractional → ve formátu 1.7, provádí normalizaci výsledku (2.14 na 1.15, což je to << 1 v tabulce)

MUL	Rd,Rr	Multiply Unsigned	R1:R0 = Rd * Rr	Z,C	2
MULS	Rd,Rr	Multiply Signed	R1:R0 = Rd * Rr	Z,C	2
MULSU	Rd,Rr	Multiply Signed with Unsigned	R1:R0 = Rd * Rr	Z,C	2
FMUL	Rd,Rr	Fractional Multiply Unsigned	R1:R0 = (Rd * Rr) << 1	Z,C	2
FMULS	Rd,Rr	Fractional Multiply Signed	R1:R0 = (Rd * Rr) << 1	Z,C	2
FMULSU	Rd,Rr	Fractional Multiply Signed with Unsigned	R1:R0 = (Rd * Rr) << 1	Z,C	2

- `fmuls`: 10 * -10 (246 ve 2D) = (R1: 0xFF, R0: 0x9C), což je 65436 (-100 v doplňku do 2^{16}-1 = 65536)

Logické instrukce

AND	Rd,Rr	Logical AND	Rd = Rd & Rr	Z,N,V,S	1
ANDI	Re,K8	Logical AND with Immediate	Re = Re & K8	Z,N,V,S	1
OR	Rd,Rr	Logical OR	Rd = Rd | Rr	Z,N,V,S	1
ORI	Re,K8	Logical OR with Immediate	Re = Re | K8	Z,N,V,S	1
EOR	Rd,Rr	Logical Exclusive OR	Rd = Rd EOR Rr	Z,N,V,S	1
SBR	Re,K8	Set Bit(s) in Register	Re = Re | K8	Z,C,N,V,S	1
CBR	Re,K8	Clear Bit(s) in Register	Re = Re & (0xFF - K8)	Z,C,N,V,S	1

- SBR , CBR → manipulace s bity přes maskování, tam, kde je v K8:
  - 1 → příslušný bit bude nastaven (SBR), nebo smazán (CBR)
  - 0 → příslušný bit bude ponechán beze změny
  - pozn. maskování – využívá se vlastností log. fcí:

Bit	maska	AND	OR	XOR
0	0	0	0	0
1	0	0	1	1
0	1	0	1	1
1	1	1	1	0

AND: maska = 0 → bit = 0, maska = 1 → bit ponechán beze změny
OR: maska = 1 → bit = 1, maska = 0 → bit ponechán beze změny
XOR: maska = 1 → bit negován, maska = 0 → bit ponechán beze změny

  - často se používá

```
.equ  Bit0 = 0
.equ  Bit1 = 1
.equ  Bit2 = 2
.equ  Bit3 = 3
.equ  Bit4 = 4
.equ  Bit5 = 5
.equ  Bit6 = 6
.equ  Bit7 = 7

; nastavení bitů Bit2, Bit3 a Bit7 v R16, ostatní beze změny
sbr   r16,(1 << Bit2) | (1 << Bit3) | (1 << Bit7)
      ; 1 << Bit2 = bitový posun 1 o 2 bity doleva = 0x04
      ; (1 << Bit2) | (1 << Bit3) | (1 << Bit7) =
      ;    0x04     |    0x08     |    0x80     =
```

```
                    ; = 0x8C = 0b10001100
```

- TST → pouze nastaví příznaky stavového slova CPU dle obsahu Rd

Instrukce přesunů

- dále symbolika:

Registr – registr

MOV	Rd,Rr	Copy register	Rd = Rr	None	1
MOVW	Rd,Rr	Copy register pair	Rd+1:Rd = Rr+1:Rr, r,d even	None	1

Registr <–> RAM

- z RAM nebo konstanta do registrů; load = nahraj (z RAM)

LDI	Re,K8	Load Immediate	Re = K	None	1
LDS	Rd,k	Load Direct	Rd = (k)	None	2*
LD	Rd,P	Load Indirect	Rd = (P)	None	2*
LD	Rd,P+	Load Indirect and Post-Increment	Rd = (P), P=P+1	None	2*
LD	Rd,-P	Load Indirect and Pre-Decrement	P=P-1, Rd = (P)	None	2*
LDD	Rd,Q+K6	Load Indirect with displacement	Rd = (Q+K6)	None	2*

- Z registrů do RAM; store = ulož (do RAM)

STS	k,Rr	Store Direct	(k) = Rr	None	2*
ST	P,Rr	Store Indirect	(P) = Rr	None	2*
ST	P+,Rr	Store Indirect and Post-Increment	(P) = Rr, P=P+1	None	2*
ST	-P,Rr	Store Indirect and Pre-Decrement	P=P-1, (P)=Rr	None	2*
STD	Q+K6,Rr	Store Indirect with displacement	(Q+K6) = Rr	None	2

- příklad:

```
        ; konstantní data
        ldi   r16,13            ; r16 = 13
        ; přímé adresování (prom je proměnná v RAM)
        lds   r0,prom           ; r0 = prom
        sts   prom,r1           ; prom = r1
        ; nepřímé adresování
        ldi   XL,LOW(prom)      ; do X uložím adresu prom (16b adresy)
        ldi   XH,HIGH(prom)     ; horní byte adresy
        ld    r2,X              ; r2 = (X), neboli r2 = prom
        st    X,r3              ; (X) = r3, neboli prom = r3
        ; nepřímé s postinkrementací
        ; v cyklu pole[i] = 0xAA+i, i = 0 až 33, pole je proměnná v RAM
        ldi   YL,LOW(pole)
        ldi   YH,HIGH(pole)
        ldi   r18,33            ; čítač opakování, bude dekrementován
        ldi   r17,0xAA          ; do r5 chci 0xAA, nelze přímo!!!
        mov   r5,r17            ; r5 = 0xAA
smycka:
        st    Y+,r5             ; pole[i++] = r5
        inc   r5                ; 0xAA+i
        dec   r18               ; dec a brne je vlastně DJNZ
        brne  smycka            ; r18<>0 skočí (popis brne viz. níže)
        ; náhodný přístup do pole
        ; r10 = pole[3]
        ldi   YL,LOW(pole)
        ldi   YH,HIGH(pole)
        ldd   r10,Y+3
        ; pole[19] = 89
        movw  ZL:ZL,YH:YL       ; kopie Y do Z (není nutné, …
                                ; … std by mohlo pracovat i s Y)
        ldi   r18,89
```

```
std   Z+19,r18
```

Obsluha SFR

- opak: SFR je v RAM – adresy 0x20–0x5F (64 adres)

IN	Rd,P	In Port	Rd = P	None	1
OUT	P,Rr	Out Port	P = Rr	None	1

- in → kopie registru v oblasti SFR do Rd
- out → kopie Rr do registru v oblasti SFR
- příklady:

```
in    r10, PINB                ; přečte bránu B a uloží ji do R10
; nastavení UART (v reg. UCSRB chceme nahodit bity RXEN a TXEN)
ldi r16,(1<<RXEN)|(1<<TXEN)   ; RXEN a TXEN definovány přes EQU
out UCSRB,r16
```

Zásobník

PUSH	Rr	Push register on Stack	STACK = Rr	None	2
POP	Rd	Pop register from Stack	Rd = STACK	None	2

- obsluha → nepřímé adr. přes SP
- SP je obecně 16-ti bitový, rozdělen na dva registry SPH a SPL. Nalézá se v I/O prostoru.
  Nutno ho na začátku programu nastavit např. na konec SRAM:

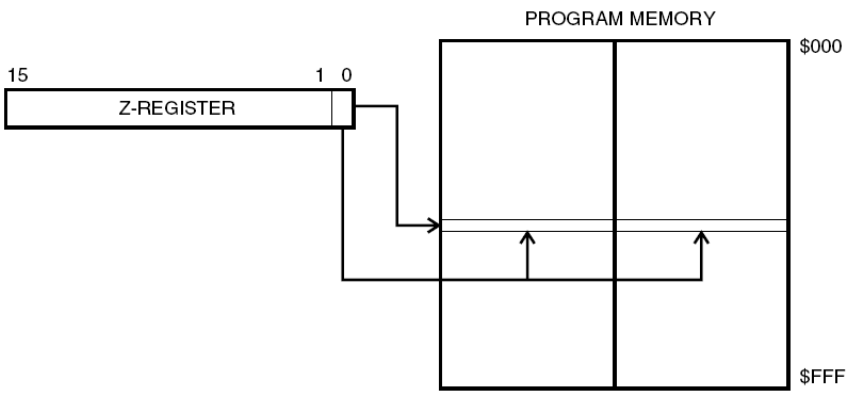
```
ldi R16,HIGH(RAMEND)
out SPH,R16
ldi R16,LOW(RAMEND)
out SPL,R16
;Konstanta RAMEND určuje poslední adresu ve SRAM, je ;definovaná
v hlavičkovém souboru a je závislá na typu ;AVR
```

Z a do paměti programu

- Load Program Memory = čtení z paměti programu (do GPR)
  - (u 8051 movc A,@A+DPTR)
- Store Program Memory = zápis do paměti programu (AVR je schopno přeprogramovat samo sebe)

LPM	None	Load Program Memory	R0 = (Z)	None	3
LPM	Rd,Z	Load Program Memory	Rd = (Z)	None	3
LPM	Rd,Z+	Load Program Memory and Post-Increment	Rd = (Z), Z=Z+1	None	3
ELPM	None	Extended Load Program Memory	R0 = (RAMPZ:Z)	None	3
ELPM	Rd,Z	Extended Load Program Memory	Rd = (RAMPZ:Z)	None	3
ELPM	Rd,Z+	Extended Load Program Memory and Post Increment	Rd = (RAMPZ:Z), Z = Z+1	None	3
SPM	None	Store Program Memory	(Z) = R1:R0	None	-
ESPM	None	Extended Store Program Memory	(RAMPZ:Z) = R1:R0	None	-

- Princip práce lpm
  - 15 horních bitů adresuje slovo v paměti programu (určuje řádek), nultý bit vybírá horní (lsb = 1) nebo dolní (lsb = 0) bajt slova (určuje sloupec).



- V asm je nutné adresu násobit 2

- Příklad: kopie tabulky dat `tab` v CSEG do RAM (pole `tabRam`) na ATmega32

```
.INCLUDE "m32def.inc"

        .DSEG
tabRam:     .BYTE 10

        .CSEG
        .ORG  0
        rjmp  start
start:
        ldi   ZL, LOW(2*tab)
        ldi   ZH, HIGH(2*tab)

        ldi   r16, 6
        mov   r1, r16

        ldi   XL, LOW(tabRam)
        ldi   XH, HIGH(tabRam)
loop:   lpm   r2, Z+
        st    X+, r2
        dec   r1
        brne  loop
end:    rjmp  end

tab:    .DB   10,245,30,47,75,96
```

Bitové operace

Základní

LSL	Rd	Logical shift left	Rd(n+1)=Rd(n), Rd(0)=0, C=Rd(7)	Z,C,N,V,H,S	1
LSR	Rd	Logical shift right	Rd(n)=Rd(n+1), Rd(7)=0, C=Rd(0)	Z,C,N,V,S	1
ROL	Rd	Rotate left through carry	Rd(0)=C, Rd(n+1)=Rd(n), C=Rd(7)	Z,C,N,V,H,S	1
ROR	Rd	Rotate right through carry	Rd(7)=C, Rd(n)=Rd(n+1), C=Rd(0)	Z,C,N,V,S	1
ASR	Rd	Arithmetic shift right	Rd(n)=Rd(n+1), n=0,...,6	Z,C,N,V,S	1
SWAP	Rd	Swap nibbles	Rd(3..0) = Rd(7..4), Rd(7..4) = Rd(3..0)	None	1
NOP	None	No operation	None	None	1
SLEEP	None	Sleep	See instruction manual	None	1
WDR	None	Watchdog Reset	See instruction manual	None	1

Manipulace s SREG

- SREG = název pro PSW u AVR

BSET	b	Set flag	SREG(b) = 1	SREG(b)	1
BCLR	b	Clear flag	SREG(b) = 0	SREG(b)	1

BST	Rr,b	Bit store from register to T	T = Rr(b)	T	1
BLD	Rd,b	Bit load from register to T	Rd(b) = T	None	1

- příklady:

  (u každého AVR platí definice):

```
; SREG - Status Register
.equ  SREG_C = 0  ; Carry Flag
.equ  SREG_Z = 1  ; Zero Flag
.equ  SREG_N = 2  ; Negative Flag
.equ  SREG_V = 3  ; Two's Complement Overflow Flag
.equ  SREG_S = 4  ; Sign Bit
.equ  SREG_H = 5  ; Half Carry Flag
.equ  SREG_T = 6  ; Bit Copy Storage
.equ  SREG_I = 7  ; Global Interrupt Enable


bset  6           ; Set T flag
bset  SREG_T      ; totéž
bst   r16,Bit2    ; Store bit 2 of r1 in T flag
bld   r0,4        ; Load T into bit 4 of r0
```

SEC	None	Set carry flag	C =1	C	1
CLC	None	Clear carry flag	C = 0	C	1
SEN	None	Set negative flag	N = 1	N	1
CLN	None	Clear negative flag	N = 0	N	1
SEZ	None	Set zero flag	Z = 1	Z	1
CLZ	None	Clear zero flag	Z = 0	Z	1
SEI	None	Set interrupt flag	I = 1	I	1
CLI	None	Clear interrupt flag	I = 0	I	1
SES	None	Set signed flag	S = 1	S	1
CLN	None	Clear signed flag	S = 0	S	1
SEV	None	Set overflow flag	V = 1	V	1
CLV	None	Clear overflow flag	V = 0	V	1
SET	None	Set T-flag	T = 1	T	1
CLT	None	Clear T-flag	T = 0	T	1
SEH	None	Set half carry flag	H = 1	H	1
CLH	None	Clear half carry flag	H = 0	H	1

- např. `SEC` je vlastně `BSET SREG_C`
- `CLC` je obdoba CLR C u 8051

Bity v SFR

SBI	P,b	Set bit in I/O register	I/O(P,b) = 1	None	2
CBI	P,b	Clear bit in I/O register	I/O(P,b) = 0	None	2

- dolních 32 adres v SFR je bitově přístupných (adresy 0 až 31 v SFR jsou adresy 32 až 63 v RAM!!!)
- použití především při práci s porty
- jiný příklad

```
sbi   EECR,EEMWE        ; u ATmega64 je EEMWE 2.bit v registru EECR…
                        ; … adresa 0x1C; 0x1C < 0x20 (31 dekadicky))
in    r16,EECR          ; jiné řešení téhož problému
sbr   r16,(1 << EEMWE)  ; složitě proto, aby předchozí obsah EECR
out   EECR,r16          ; zůstal zachován
```

Větvení programu

Běžné instrukce

RJMP	k	Relative Jump	PC = PC + k +1	None	2
JMP	k	Jump	PC = k	None	3
RCALL	k	Relative Call Subroutine	STACK = PC+1, PC = PC + k + 1	None	3/4*
CALL	k	Call Subroutine	STACK = PC+2, PC = k	None	4/5*
RET	None	Subroutine Return	PC = STACK	None	4/5*
RETI	None	Interrupt Return	PC = STACK	I	4/5*

Neběžné skoky

IJMP	None	Indirect Jump to (Z)	PC = Z	None	2
EIJMP	None	Extended Indirect Jump (Z)	STACK = PC+1, PC(15:0) = Z, PC(21:16) = EIND	None	2
ICALL	None	Indirect Call to (Z)	STACK = PC+1, PC = Z	None	3/4*
EICALL	None	Extended Indirect Call to (Z)	STACK = PC+1, PC(15:0) = Z, PC(21:16) =EIND	None	4*

- ICALL je vlastně volání funkce přes pointer na funkci

Větvení

- u RISC poněkud složitější (nejsou sdružené instrukce typu DJNZ, CJNE apod.)
- někdy kombinace několika instrukcí

TST	Rd	Test for Zero or Negative	Rd = Rd & Rd	Z,C,N,V,S	1
CP	Rd,Rr	Compare	Rd - Rr	Z,C,N,V,H,S	1
CPC	Rd,Rr	Compare with Carry	Rd - Rr - C	Z,C,N,V,H,S	1
CPI	Re,K8	Compare with Immediate	Rd - K	Z,C,N,V,H,S	1
CPSE	Rd,Rr	Compare, Skip if equal	if (Rd ==Rr) PC = PC 2 or 3	None	1/2/3
SBRC	Rr,b	Skip if bit in register cleared	if(Rr(b)==0) PC = PC + 2 or 3	None	1/2/3
SBRS	Rr,b	Skip if bit in register set	if(Rr(b)==1) PC = PC + 2 or 3	None	1/2/3
SBIC	P,b	Skip if bit in I/O register cleared	if(I/O(P,b)==0) PC = PC + 2 or 3	None	1/2/3
SBIS	P,b	Skip if bit in I/O register set	if(I/O(P,b)==1) PC = PC + 2 or 3	None	1/2/3

- podmíněné větvení – dle výsledku předchozí operace v ALU (nastaví flagy v SREG)

BRBC	s,k	Branch if Status flag cleared	if(SREG(s)==0) PC = PC + k + 1	None	1/2
BRBS	s,k	Branch if Status flag set	if(SREG(s)==1) PC = PC + k + 1	None	1/2
BREQ	k	Branch if equal	if(Z==1) PC = PC + k + 1	None	1/2
BRNE	k	Branch if not equal	if(Z==0) PC = PC + k + 1	None	1/2
BRCS	k	Branch if carry set	if(C==1) PC = PC + k + 1	None	1/2
BRCC	k	Branch if carry cleared	if(C==0) PC = PC + k + 1	None	1/2
BRSH	k	Branch if same or higher	if(C==0) PC = PC + k + 1	None	1/2
BRLO	k	Branch if lower	if(C==1) PC = PC + k + 1	None	1/2
BRMI	k	Branch if minus	if(N==1) PC = PC + k + 1	None	1/2
BRPL	k	Branch if plus	if(N==0) PC = PC + k + 1	None	1/2
BRGE	k	Branch if greater than or equal (signed)	if(S==0) PC = PC + k + 1	None	1/2
BRLT	k	Branch if less than (signed)	if(S==1) PC = PC + k + 1	None	1/2
BRHS	k	Branch if half carry flag set	if(H==1) PC = PC + k + 1	None	1/2
BRHC	k	Branch if half carry flag cleared	if(H==0) PC = PC + k + 1	None	1/2
BRTS	k	Branch if T flag set	if(T==1) PC = PC + k + 1	None	1/2
BRTC	k	Branch if T flag cleared	if(T==0) PC = PC + k + 1	None	1/2
BRVS	k	Branch if overflow flag set	if(V==1) PC = PC + k + 1	None	1/2
BRVC	k	Branch if overflow flag cleared	if(V==0) PC = PC + k + 1	None	1/2
BRIE	k	Branch if interrupt enabled	if(I==1) PC = PC + k + 1	None	1/2
BRID	k	Branch if interrupt disabled	if(I==0) PC = PC + k + 1	None	1/2

typické situace

- cyklus

```
      ldi   r18,OPAK          ; čítač opakování, bude dekrementován
cyklus:
      ; operace v cyklu
      dec   r18               ; dec a brne je vlastně DJNZ
      brne  cyklus            ; r18<>0 skočí
```

- úplná podmínka

`if (PODMINKA) {`
`  // kod 1;`
`}`
`else {`
`  // kod 2;`
`}`
`// kod za if`
	`if:  PODMINKA`
	`  br??  skok`
	`  ; kod 1`
	`  rjmp  endif`
	`skok:`
	`  ; kod 2`
	`endif: ; kod za if`

1) `PODMINKA`: `CP`, `CPI`, `TST` (nastaví flagy)

2) `br??` → skok na základě flagů předchozí operace

Test jednoho čísla:

```
        tst   r0          ; skočí, když je r0:
        brne  navesti     ; != 0
        breq  navesti     ; == 0
        brmi  navesti     ; < 0 (>= 0x80 -> -1 ve 2D)
        brpl  navesti     ; > 0 (< 0x80)
```

Porovnání dvou čísel (se znaménkem, např. 165 (-91 ve 2D) < 50):

```
        cp    r0,r1       ; skočí, když je:
        brne  navesti     ; r0 != r1
        breq  navesti     ; r0 == r1
        brge  navesti     ; r0 >= r1
        brlt  navesti     ; r0 <= r1
```

Porovnání dvou čísel (bez znaménka, např. 165 > 50):

```
        cpi   r0,KONST    ; skočí, když je:
        brne  navesti     ; r0 != KONST
        breq  navesti     ; r0 == KONST
        brcs  navesti     ; r0 > KONST (C=1 při |r0| > |KONST|)
        brcc  navesti     ; r0 <= KONST
```

- neúplná podmínka

```
  if (r0 < r1) DELEJ;
        cp    r0,r1
        breq  endif       ; filtrace r0 == r1
        brcs  endif       ; filtrace r0 > r1
        ; DELEJ           ; zbývá r0 < r1
    endif:nop
```

- tlačítka na portu A, stisk (příslušný bit = 0) → akce (pro každé tl. jiná)
  - využijeme skip instrukce → přeskočí následující instrukci v případě splnění podmínky

```
        in    r0,PINA     ; přečteme port
        sbrs  r17,0       ; tl0 není stisklé -> nikam se neskočí
        rjmp  AkceTl0     ; tl. obsloužíme
        sbrs  r17,1       ; tl1 není stisklé -> nikam se neskočí
        rjmp  AkceTl1     ; tl. obsloužíme
        sbrs  r17,2       ; tl2 není stisklé -> nikam se neskočí
        rjmp  AkceTl2     ; tl. obsloužíme
```