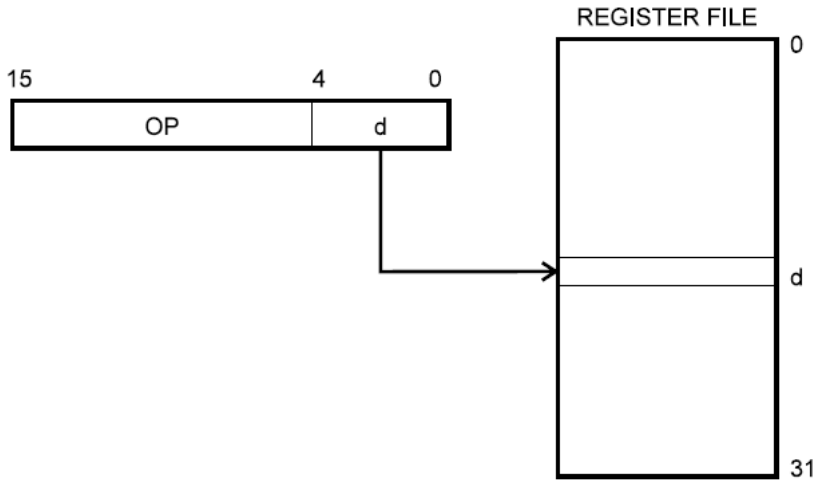
Adresovací módy paměti programu a dat

- AVR architektura obsahuje několik adresovacích módů pro přístup k paměti programu (FLASH) a k datové paměti (SRAM, pracovní registry a I/O registry)
- Na následujících obrázcích, které popisují jednotlivé adresovací módy, znamená zkratka OP kód instrukce v instrukčním slově.

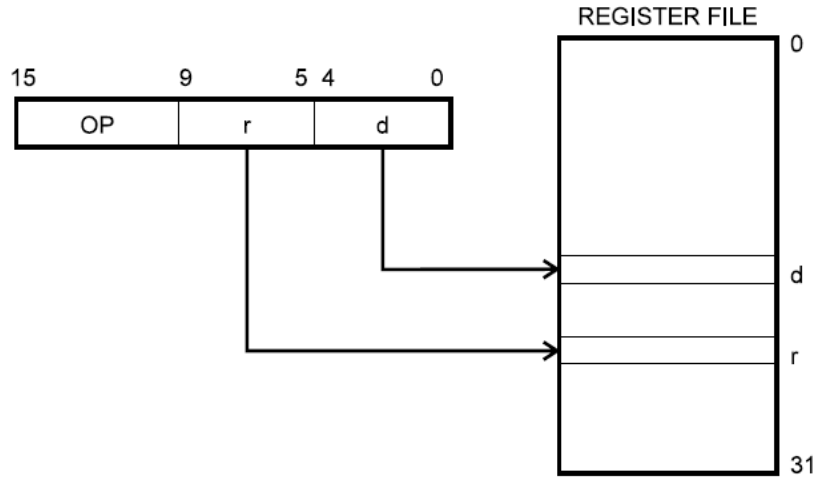
Přímé adresování jednoho registru

- Operand je uložen v registru Rd
- Např. instrukce INC Rd nebo ROL Rd



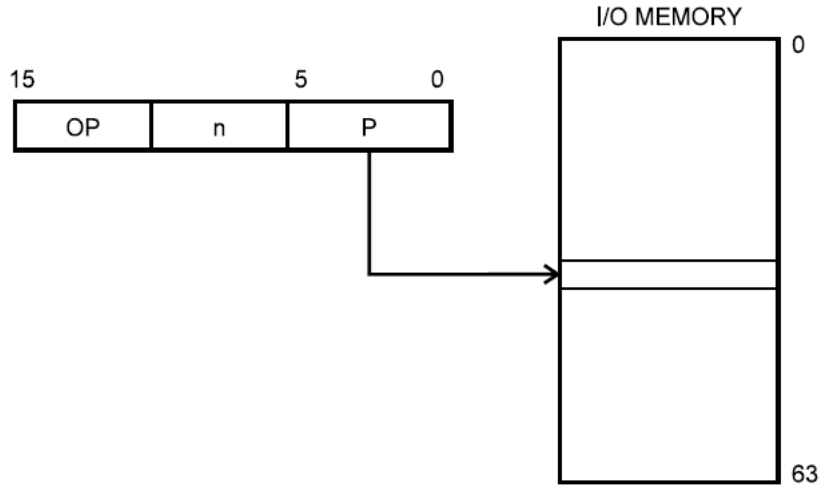
Přímé adresování dvou registrů

- Operand je uložen v registru Rr a Rd, výsledek je uložen do registru Rd
- Např. instrukce MOV Rd,Rr nebo ADD Rd,Rr



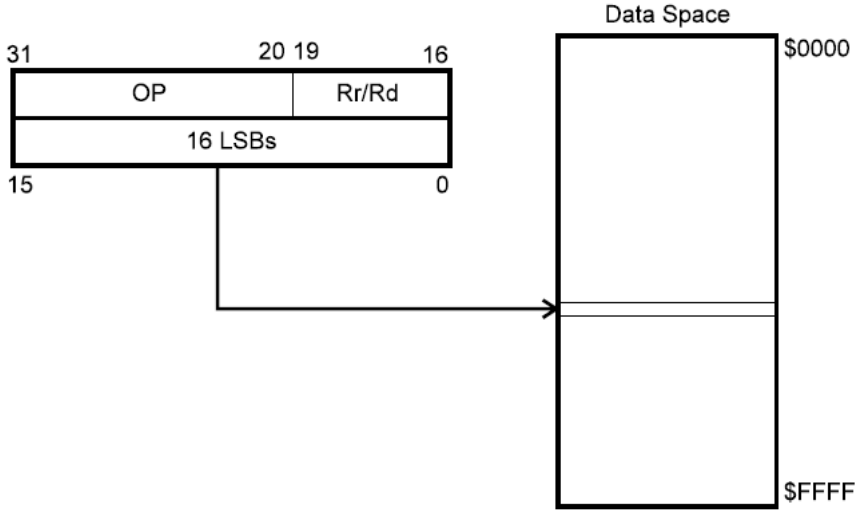
Přímé adresování I/O prostoru

- Jedním z operandů instrukce je adresa v I/O prostoru – viz obrázek, znak P
- Druhým operandem je adresa zdrojového nebo cílového registru – obrázek , znak n
- Např. instrukce IN Rd, P nebo OUT P, Rr



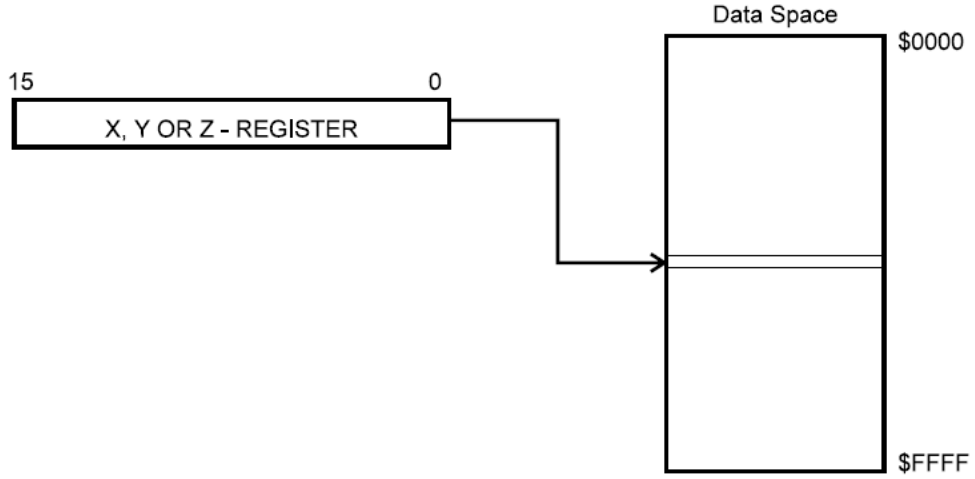
Přímé adresování dat

- 16-ti bitová adresa je obsažena v dolních 16-ti bitech (LSB) dvouslovové instrukce. Rd/Rr je cílový nebo zdrojový registr
- Např. instrukce LDS Rd, k nebo STS k, Rr , kde k je v rozsahu 0 až 65535



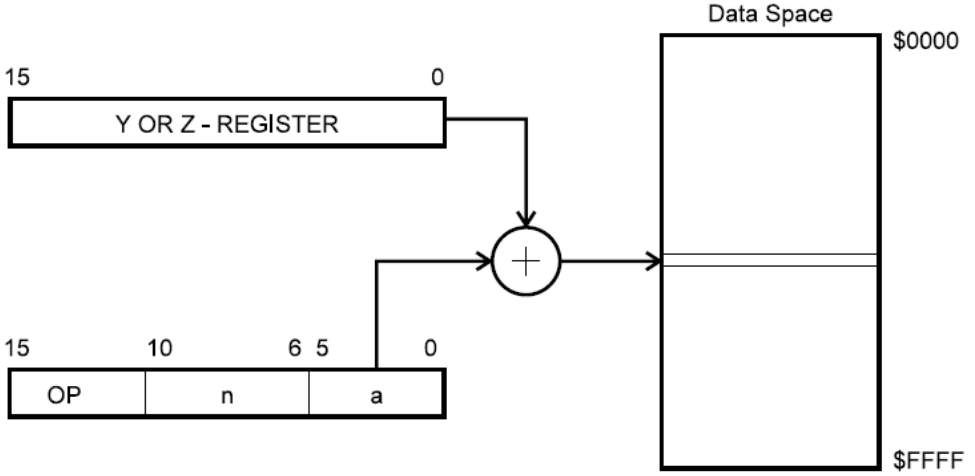
Nepřímé adresování dat

- Adresa operandu je obsahem registru X, Y nebo Z.
- Např. instrukce LD Rd,X nebo ST X,Rr , stejně pro Y a Z
- Využití pro přístup k polím, tabulkám a ukazateli na zásobník.



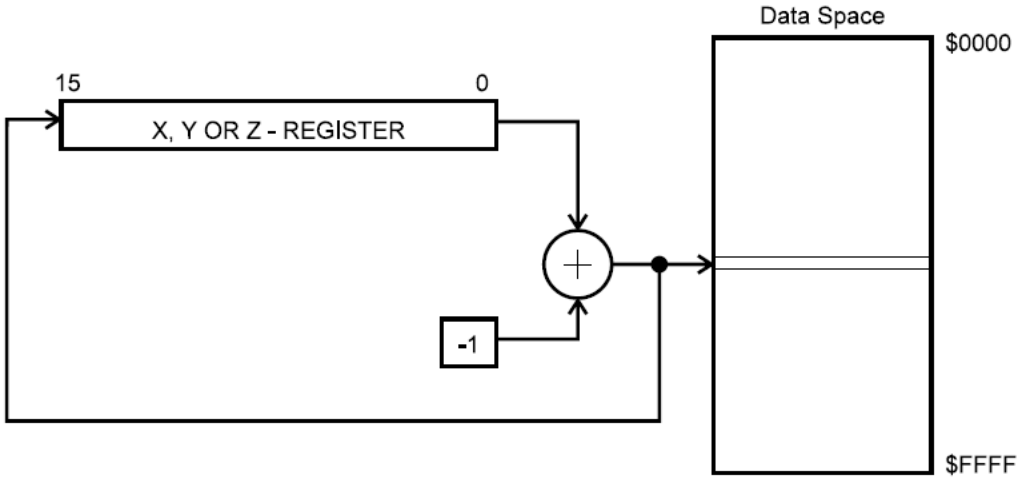
Nepřímé adresování dat s posunem

- Adresa operandu se získá přičtením obsahu registru Y nebo Z k adrese umístěné v dolních 6-ti bitech instrukčního slova.
- Druhým operandem je adresa zdrojového nebo cílového registru – znak n.
- Např. instrukce LDD Rd,Y+a nebo STD Y+a,Rr , stejně pro Z
- Využití pro přístup k polím, tabulkám a ukazateli na zásobník.



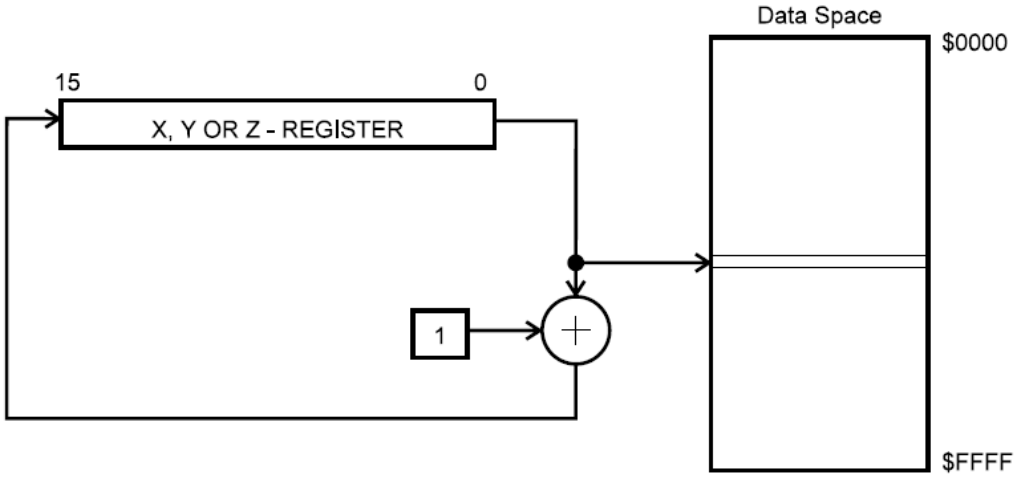
Nepřímé adresování dat s pre-dekrementací

- Registr X, Y nebo Z je před provedením operace dekrementován. Adresa v operandu je dekrementovaný obsah registru X, Y nebo Z.
- Např. instrukce LD Rd,-X nebo ST –X,Rr , stejně pro Y a Z
- Využití pro přístup k polím, tabulkám a ukazateli na zásobník.



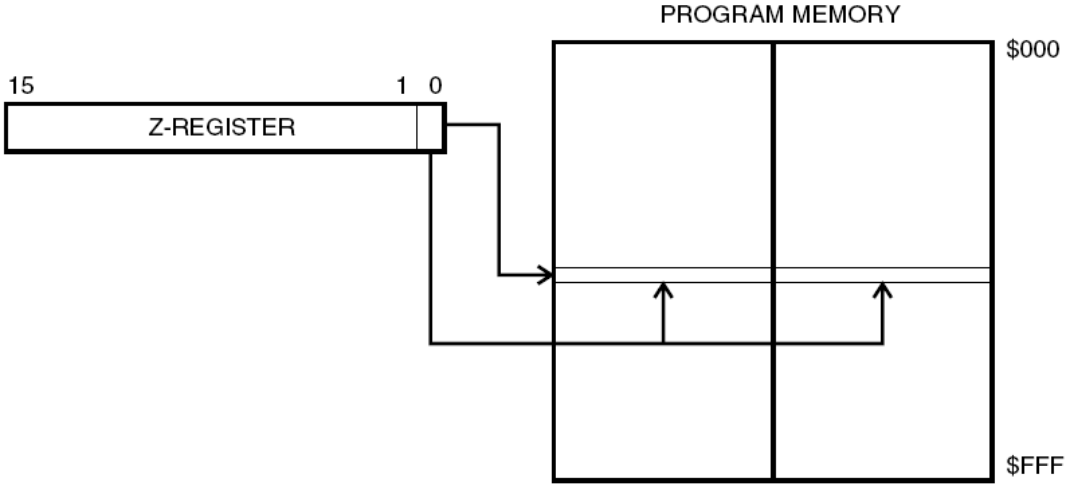
Nepřímé adresování dat s post-inkrementací

- Registr X, Y nebo Z je inkrementován po provedení operace. Adresa v operandu je obsah registru X, Y nebo Z před inkrementací.
- Např. instrukce LD Rd,X+ nebo ST X+,Rr , stejně pro Y a Z
- Využití pro přístup k polím, tabulkám a ukazateli na zásobník.



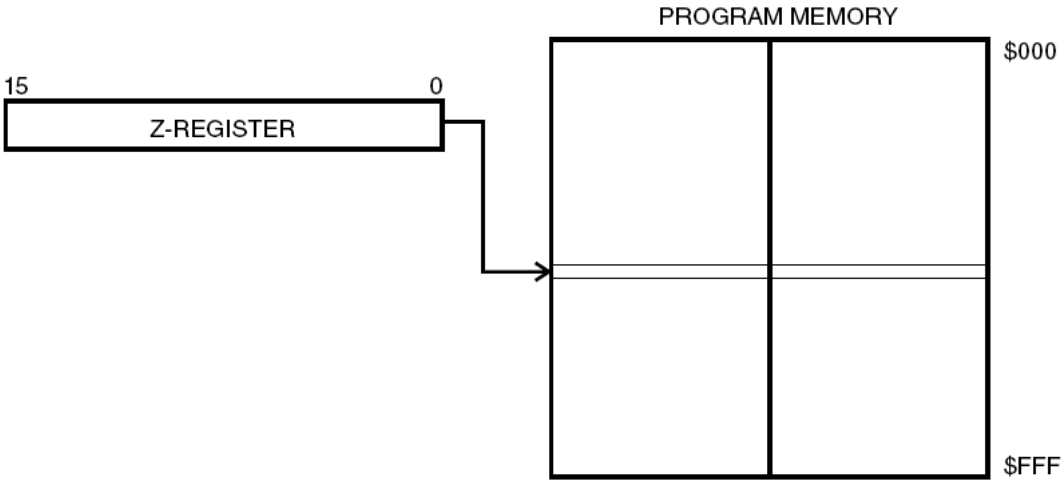
Přímé adresování konstanty v paměti programu

- Adresa konstanty je určená obsahem registru Z.
- 15 horních bitů adresuje slovo v paměti programu (určuje řádek), nultý bit vybírá horní (lsb = 1) nebo dolní (lsb = 0) bajt slova (určuje sloupec).
- Např. instrukce LPM
- Mimo AVR základní řady jsou možné další instrukce:
  LPM Rd,Z nebo LPM Rd,Z+ nebo SPM – zápis jednoho slova do paměti programu - viz datasheet
- Využití pro vyčítání tabulek, řetězců apod. uložených v paměti programu.



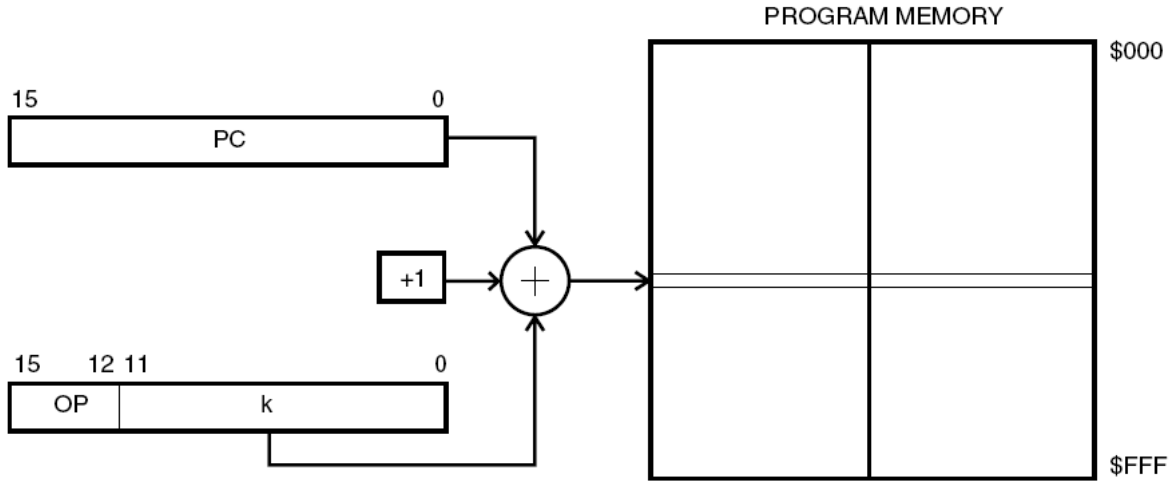
Nepřímé adresování paměti programu

- Provádění programu pokračuje na adrese obsažené v registru Z, tj. programový čítač PC je naplněn obsahem registru Z.
- instrukce IJMP a ICALL.



Relativní adresování v paměti programu

- Provádění programu pokračuje na adrese PC + k + 1. Relativní adresa k může být v rozsahu -2048 až +2047.
- instrukce RJMP k nebo RCALL k



Souhrn

- registry X, Y, Z slouží pro nepřímé adresování paměti dat (slouží jako ukazatele)
- registr Z slouží také pro nepřímé adresování paměti programu
- lze využít pre-dekrementaci a post-inkrementaci ukazatele
- pro přístup k I/O registrům, paměti dat a pro přesun dat mezi pracovními registry se používají rozdílné instrukce – odlišnost oproti 8051, kde jsme si vystačili s MOV

Atmel AVR assembler

- integrován do AVR studia
- v následujících odstavcích budou uvedeny pouze nejzákladnější informace, podrobnější informace lze získat v literatuře nebo datasheetech firmy Atmel

Direktivy assembleru

- direktivy se nepřekládají do strojového kódu, slouží k nastavení umístění v programové paměti, definují makra, inicializují paměť atd. Přehled direktiv je

uveden v následující tabulce.

Direktiva	Popis
.BYTE	Rezervuje byte pro proměnnou
.CSEG	Kódový segment
.DB	Definuje konstantní byte
.DEF	Definuje symbolické jméno pro registr
.DEVICE	Definuje pro který uC se má provádět překlad
.DSEG	Datový segment
.DW	Definuje konstantní slovo
.ENDM nebo .ENDMACRO	Konec makra
.EQU	Přiřazuje symbol výrazu
.ESEG	EEPROM segment
.EXIT	Opuštění souboru
.INCLUDE	Čtení zdrojového kódu z dalšího souboru
.LIST	Zapíná generování listingu
.LISTMAC	Zapíná rozvinutí makra v listingu
.NOLIST	Vypíná generování listingu
.ORG	Nastavuje základní adresu programu
.SET	Přiřazuje symbol výrazu

Komentáře

- pomocí znaku „středník“
  např.: `ldi temp,0xFF  ;nastavení portu A jako výstupního`

Začátek programu

- vložení definičního souboru pro daný typ AVR
  - tento hlavičkový soubor definuje pomocí direktivy `.EQU` jména I/O registrů, jména jednotlivých bitů I/O registrů, definuje registry XH,XL,YH,YL,ZH,ZL pro nepřímé adresování, dále pojmenovává adresy vektorů přerušení a další konstanty (např. název použitého AVRka, velikost RAM, koncovou adresu RAM apod.)
  - hlavičkové soubory jsou pojmenovány `xxxxdef.inc`
  - např.: `.INCLUDE "m32def.inc"`
- definice konstant, tabulek, nových jmen registrů apod.
  např.:
  ```
  .EQU PI = 3.14
  konst: .DB 0, 1, 255, 34, 25, -5
  .DEF temp = R16
  ```

Registry

- 32 pracovních registrů R0 až R31

- Pomocí direktivy `.DEF` jim lze přiřadit jména a ty používat v programu – výhoda.
  např.: `.DEF temp = R16`

Naplnění registru konstantou

- Instrukce `ldi` (load immediate)
- Např.: `ldi temp,10  ;naplní registr R16 číslem 10`
- **Pracuje pouze s registry R16 až R31 !!**

Přesun mezi dvěma registry

- Instrukce `mov`
- Např.: `mov temp,r2  ;naplní registr R16 hodnotu z reg. R2`

Registry X,Y,Z

- Slouží pro nepřímé adresování paměti dat (X,Y,Z) nebo paměti programu (Z)
- Horní byte příslušného registru je označen xH, dolní byte xL (např.: ZH a ZL)
- Naplnění registru X, Y nebo Z adresou se provede následovně:

```
.EQU Adresa = 0x100        ;adresa pocatku tabulky
;nejaky kod
ldi XH,HIGH(Adresa)        ;horni byte adresy
ldi XL,LOW(Adresa)         ;dolni bajt adresy
```

- Vlastní nepřímé adresování se provede pomocí instrukcí ld (Load) a st (Store)

Ukazatel	Popis funkce	Příklad zápisu v asm
X	Zápis/čtení do/z adresy X	`st x,r1` `ld r1,x`
X+	Zápis/čtení do/z adresy X s post-inkrementací ukazatele	`st x+,r1` `ld r1,x+`
-X	Zápis/čtení do/z adresy X s pre-dekrementací ukazatele	`st -x,r1` `ld r1,-x`

- Registr Z lze použít pro nepřímé adresování paměti programu
  - Paměť programu je organizována po slovech (po 16-ti bitech) – adresu nutno násobit dvěma - potom horních 15 bitů udává řádek a nultý bit vybírá horní či dolní bajt ve slově (určuje sloupec) – viz kapitola adresovací módy

```
ldi ZH,HIGH(2*Adresa)        ;horni byte adresy
ldi ZL,LOW(2*Adresa)         ;dolni bajt adresy
```

  - Vlastní nepřímé adresování se provede pomocí instrukce `lpm` (Load program memory), která načte do registru R0 obsah na adrese uložené v reg. Z

- Využití pro načtení tabulek či řetězců z paměti programu. Abychom se mohli pohybovat po tabulce, nutno inkrementovat či dekrementovat ukazatel (obsah registru Z) – použití instrukce `adiw` nebo `sbiw` , které přičítají nebo odečítají konstantu v rozsahu 0 až 63 k/od registrů X,Y,Z a pro registrovou dvojici R25:R24. To je provedeno jako 16-ti bitová operace!
  Instrukce adiw a sbiw přebírají jako svůj první parametr dolní bajt registrové dvojice!
  např.: `adiw ZL,1 ;pricteni 1 k registru Z`
- AVR typu ATmega mají navíc dvě varianty instrukce `lpm` umožňující načtení hodnoty z adresy do jiného registru než R0 a post-inkrementaci Z. Dále mají i instrukci `spm`, která zapisuje do paměti programu slovo v R1:R0.

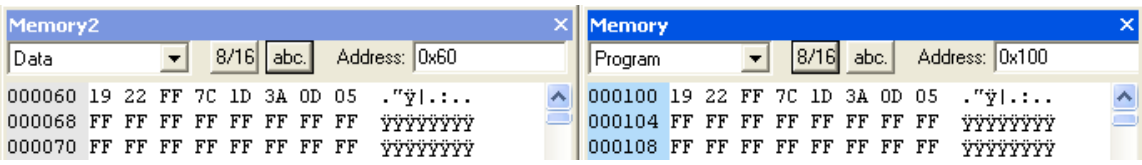
Příklad kopírování tabulky z paměti programu do paměti dat

```
.include "m32def.inc"

.EQU RAM_ADR = 0x0060
.DEF pocit = R16

.CSEG
.ORG 0x0100    ;tabulka ulozena v pameti prog. od adresy 0x0100
data: .DB 25,34,255,124,29,58,13,5

.ORG 0x0000
start:
          ldi pocit,8              ;8 prvku v tabulce
          ldi ZH,HIGH(2*data)      ;horni byte adresy v CODE
          ldi ZL,LOW(2*data)       ;dolni bajt adresy v CODE
          ldi XH,HIGH(RAM_ADR)     ;horni byte adresy v DATA
          ldi XL,LOW(RAM_ADR)      ;dolni bajt adresy v DATA
copy:     lpm            ;vycteni prvku z tabulky
          adiw Z,1       ;inkrementace ukazatele
          st x+,R0       ;ulozeni do SRAM a inkr. ukazatele
          dec pocit      ;dekrementace pocitadla
          brne copy
konec:    rjmp konec
```



Seznam instrukcí pracujících pouze s registry R16 až R31

```
ldi Rd,k  ;naplnění registru Rd konstantou k
andi Rd,k ;bitový součin registru Rd a konstanty k
```

```
cbr Rd,K  ;vymazání všech bitů v Rd, které jsou nastaveny
          ;na jedničku v konstantní masce hodnoty K
cpi Rd,k  ;porovnání Rd s konstantou k
sbci Rd,k ;odečtení konstanty k a aktuální hodnoty příznaku
          ;přenosu (C) od registru Rd a uložení výsledku do Rd
sbr Rd,K  ;bitový součet registru Rd a masky K
ser Rd    ;nastavení všch bitů v Rd na jedničku
subi Rd,k ;odečtení konstanty od obsahu registru Rd a uložení
          ;výsledku zpět do Rd
```

Souhrn

- Jednotlivé registry lze pojmenovat pomocí direktivy `.DEF`
- Potřebujeme-li přístup do paměti pomocí ukazatelů, použijeme registry X,Y,Z.
- Šestnáctibitovou proměnnou je vhodné umístit do R25:R24 – využijeme instrukcí `adiw` nebo `sbiw`
- Předpokládáme-li bitový přístup do registrů, vyhradíme si R16 až R23.
- Pro vyčítání tabulek z paměti programu použijeme registr Z a nepřímé adresování.

I/O porty (I/O registry)

- 64 I/O portů, ne všechny jsou dostupné (záleží na typu AVR)
- Definice názvů I/O portů jsou uvedeny v hlavičkovém souboru `xxxxdef.inc` – výhoda – nemusíme si pamatovat adresy. Taktéž jednotlivé bity I/O portů mají přidělen název. Pojmenování je totožné s názvy v datasheetu daného typu AVR.

Čtení a zápis z/do I/O portů (registrů)

- Přístup po bytech
  - Instrukce `in` a `out`
    Př.:
    ```
    in R16,SREG    ;nacteni obsahu stavoveho reg. do R16
    out SPL,R16    ;zapis do I/O registru (nastaveni
    out SPH,R17    ;Stack Pointeru)
    ```
  - Instrukce `lds` a `sts` – využíváme přístupu do SRAM
    Př.:
    ```
    ;nutno pricist 0x20 nebot I/O registry jsou
    ;mapovany do SRAM od adresy 0x20 (32 desitkove)
    lds R16,(SREG+0x20)
    sts (SPL+0x020),R16
    sts (SPH+0x020),R17
    ```

- Přístup po bitech
  - Instrukce `sbi` (Set Bit) a `cbi` (Clear Bit)
    Př.:
    ```
    sbi PORTB,0          ;nastaví bit 0 v reg. PORTB na 1
    sbi PORTB,PB5        ;nastaví bit 5 v reg. PORTB na 1
    cbi PORTB,0          ;vynuluje bit 0 v reg. PORTB
    cbi PORTB,PB5        ;vynuluje bit 5 v reg. PORTB
    ```
  - **Omezení - instrukce `sbi` a `cbi` pracují pouze s I/O registry, které leží v rozsahu adres 0x00 až 0x1F – tzn. prvních 32 I/O registrů !!!!**
  - Pro registry ležící od adresy 0x20 se musí použít přístup po bajtech.
    Př.:
    ```
    in R16,TCCR1A        ;načtení I/O registru do R0
    ;nastavení vybraných bitů do 1 pomocí maskování
    ori R16,(1<<FOC1A)|(1<<COM1A1)|(1<<COM1A0)
    out TCCR1A,R16       ;zápis zpět do I/O registru
    ```
  - Pro modifikaci jednotlivých bitů ve stavovém registru jsou zavedeny instrukce `SEx` a `CLx`, kde x je název bitu.
    Př.:
    ```
    sei  ;nastaví bit I do 1
    cli  ;vynuluje bit I
    set  ;nastaví bit T do 1
    clt  ;vynuluje bit T
    ```
- Maskování
  - Způsob nastavení vybraných bitů registru (obecně proměnné) na 1 nebo na 0
  - Využití bitového součtu (nastavení bitu do 1), bitového součinu + negace (nastavení bitu do 0) a bitového posuvu

    Příklad – nastavení bitů do 1:
    ```
    in R16,TCCR1A        ;načtení I/O registru do R0
    ;nastavení vybraných bitů do 1 pomocí maskování
    ori R16,(1<<FOC1A)|(1<<COM1A1)|(1<<COM1A0)
    out TCCR1A,R16       ;zápis zpět do I/O registru
    ```

    Příklad – nastavení bitů do 0:
    ```
    in R16,TCCR1A        ;načtení I/O registru do R0
    ;nastavení vybraných bitů do 0 pomocí maskování
    andi R16,~((1<<FOC1A)|(1<<COM1A1)|(1<<COM1A0))
    out TCCR1A,R16       ;zápis zpět do I/O registru
    ```

Přístup do SRAM

- Přímé adresování
  - Pomocí instrukcí `lds` a `sts`

    Př.:

```
.EQU Adresa = 0x0070
lds R0,Adresa        ;čtení ze SRAM z dané adresy
sts Adresa,R0        ;zápis na adresu ve SRAM
```

- Nepřímé adresování
  - Pomocí instrukcí `ld` a `st`. Lze použít registry X, Y i Z.

Ukazatel	Popis funkce	Příklad zápisu v asm
X	Zápis/čtení do/z adresy X	`st X,R1` `ld R1,X`
X+	Zápis/čtení do/z adresy X s post-inkrementací ukazatele	`st X+,R1` `ld R1,X+`
-X	Zápis/čtení do/z adresy X s pre-dekrementací ukazatele	`st -X,R1` `ld R1,-X`

  - Pomocí instrukcí `ldd` a `std` – nepřímé adr. s posunem – bázová hodnota ukazatele se nemění, pouze se k němu dočasně přičítá konstanta. Lze použít pouze registry Y a Z.

    Př.:

```
.EQU Adresa = 0x0070
.DEF temp = R0
ldi YH,HIGH(Adresa)  ;nastavení ukazatele
ldi YL,LOW(Adresa)   ;nastavení ukazatele

;nějaký kód

std Y+2,temp         ;uložení hodnoty do SRAM na
                     ;adresu 0x0072
ldd temp,Y+2         ;načtení hodnoty ze SRAM z
                     ;adresy 0x0072
```

Zásobník a ukazatel zásobníku (Stack Pointer)

- Zásobník je typu LIFO.
- Ukládání návratových adres a lokálních proměnných.
- 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.
;HIGH() a LOW() jsou funkce definované překladačem, které
;vrací druhý bajt výrazu (HIGH) nebo dolní bajt výrazu
;(LOW).
```

- Uložení hodnoty do zásobníku a vyjmutí hodnoty ze zásobníku

```
push R0    ;uložení obsahu registru R0 do zásobníku
pop R0     ;vyjmutí hodnoty a uložení do R0
```

Vždy je nutno dbát na pravidlo kolik jsem toho uložil, tolik toho i vyberu. Přičemž vyčítání hodnot ze zásobníku je v opačném pořadí než ukládání.

```
push R0    ;uložení obsahu registru R0 do zásobníku
push R1    ;uložení obsahu registru R1 do zásobníku
pop R1     ;vyjmutí hodnoty a uložení do R1
pop R0     ;vyjmutí hodnoty a uložení do R0
```

- Vhodné použití zásobníku je v následujích případech:
    - Hodnoty budeme opět potřebovat po několika řádcích kódu.
    - Všechny registry jsou již použité.
    - Nemámě možnost je uložit jinde.

Řízení chodu programu

- Po resetu je programový čítač PC (Program Counter) nastaven na adresu 0x0000. Po provedení resetu se nastaví všem registrům a portům defaultní hodnoty a program se začne provádět od adresy 0x0000.
- Pomocí direktivy `.ORG` můžeme sdělit překladači na jaké adrese začíná náš kód nebo obecně nastavit adresu v daném segmentu.
- Direktivou `.CSEG`, `.DSEG` nebo `.ESEG` se přepínáme mezi kódovou, datovou a EEPROM sekcí. Ve spojení s direktivou `.ORG` lze např. uložit tabulku konstant do paměti programu na dané adrese nebo do paměti EEPROM.

  Př.:
  ```
  .CSEG
  .ORG 0x0100
  tab1: .DB 1,15,23,58,125;tabulka v paměti programu od
                          ;adresy 0x0100
  .ESEG
  .ORG 0x0000
  tab2: .DB 0,10,20,30,40  ;tabulka v EEPROM od adr. 0x0000

  .DSEG
  .ORG 0x0200
  tab3: .BYTE 10      ;rezervuje 10B v datové paměti od
                      ;adresy 0x0200
  ```

- Při použití přerušení musí náš program začínat až za tabulkou vektorů přerušení, která se nachází na začátku paměti programu. První instrukce musí tedy být skok na začátek našeho programu, který se nalézá až za tabulkou vektorů přerušení.
  Př.:
  ```
  .CSEG
  .ORG 0x0000
  rjmp start          ;odskok na hlavní program
  rjmp Int0_ISR       ;vektor přerušní od INT0 - odskok na
                      ;oblužnou rutinu
  rjmp Int1_ISR       ;vektor přerušní od INT1 - odskok na
                      ;oblužnou rutinu
  start:              ;začátek hlavního programu
  ;nějaký další kód
  ```

Podmíněné skoky

- Chceme aby se program větvil na základě nějakých podmínek.
- Použití instrukcí, které testují příznaky ve stavovém slově SREG nebo určitý bit v registru, a dle výsledku dojde ke skoku či nikoliv.
- To, který ze stavových příznaků se změní při provedení nejaké instrukce, je popsáno v seznamu instrukcí v datasheetu.
- Filozofie podmíněných skoků je taková, že se nejprve provede nějaká operace, která mění bity v SREG, a poté se zavolá instrukce, která tyto bity testuje a případně skáče.
- Existují dvě skupiny instrukcí skoku.
- **Instrukce, které při splněné podmínce přeskakují následující instrukci**
  - `sbrc Rr,b` (Skip If Bit In Register Cleared)
  - `sbrs Rr,b` (Skip If Bit In Register Set)
  - `sbic P,b` (Skip If Bit In I/O Register Cleared)
  - `sbis P,b` (Skip If Bit In I/O Register Set)
  - `cpse Rd,Rd` (Compare, Skip If Equal)
  - **Příklad:**

```
;nejaky kod
sbic PINB,0     ;když PINB.0 == 0 tak přeskoč
                ;následující instrukci rjmp, jinak
                ;pokračuj normálně - dojde k vykonání
                ;instrukce rjmp
rjmp NejakeNavesti
;nejaky dalsi kod
```

  - **Omezení – bitový přístup (a tedy použití instrukcí `sbis` a `sbic`) je možný pouze k dolní polovině I/O registrů (prvních 32 – do adresy 0x1F).**

- **Instrukce, které při splněné podmínce provedou skok na danou adresu (relativní)**
  - Testují bit v SREG.
  - `brxy k` (Branch If x y), kde k je relativní adresa (PC = PC + k)
  - Např.:
    `breq k` (Branch If Equal) – testuje zda je bit Z v SREG roven 1
    a při splněné podmínce skáče
    `brne k` (Branch If Not Equal) – testuje zda je bit Z v SREG roven 0
    a při splněné podmínce skáče
    `brcs k` (Branch If Carry Set) – testuje zda je bit Z v SREG roven 1

`brcc k` (Branch If Carry Cleared) – testuje zda je bit C v SREG roven 0

`brsh k` (Branch If Same or Higher) – testuje zda je bit C v SREG roven 0

`brlo k` (Branch If Lower) – testuje zda je bit C v SREG roven 1

`brmi k` (Branch If Minus) – testuje zda je bit N v SREG roven 1

`brpl k` (Branch If Plus) – testuje zda je bit N v SREG roven 0

.

.

.

a další – viz datasheet

Příklad – čekací smyčka

```
.DEF delay = R18
.DEF delay2 = R19
cekej:    ldi delay,0xFF      ;naplneni registru konstantou
hop:      ldi delay2,0xFF     ;naplneni registru konstantou
hop2:     dec delay2          ;dekrementuj
          brne hop2           ;skakej pokud je delay2 != 0
          dec delay
          brne hop            ;skakej pokud je delay != 0
```

Ten samý program napsaný pro 8051

```
cekej:    mov R3,#FFH
hop:      mov R2,#FF
hop2:     djnz R2,hop2   ;dekrementuj a skoč není-li R2 == 0
          djnz R3,hop    ;dekrementuj a skoč není-li R3 == 0
```

Nepodmíněné skoky a volání podprogramů

- `rcall k`
  - relativní volání podprogramu, PC = PC + k + 1
  - je zakódované jako relativní odskok – relativní vzdálenost spočítá překladač
  - dovoluje volat podprogram v rozsahu paměti PC – 2K + 1 a PC + 2K (slov)
  - vykonání instrukce trvá 3 takty, délka instrukce 1 slovo (2 bajty)
  - při volání podprogramu se uloží návratová adresa do zásobníku
  - podprogram musí být ukončen instrukcí `ret`

    Př.:

    ```
    ;nejaky kod
    rcall cekej
    ;nejaky kod

    ;Podprogram cekej
    cekej:    ldi delay,0xFF
    ```

```
            ldi delay2,0xFF
hop2:       dec delay2
            brne hop2
            dec delay
            brne hop
            ret
```

- `call k`
  - absolutní volání podprogramu, PC = k
  - dovoluje volat podprogram v celém rozsahu paměti programu (4M slov nebo-li 8MB)
  - vykonání instrukce trvá 4 takty, délka instrukce 2 slova (4 bajty)
  - jen u řady ATmega (mimo ATmega8)
- `icall`
  - nepřímé volání podprogramu pomocí registru Z, PC = Z
  - dovoluje volat podprogram v rozsahu paměti 64k slov (128kB)
  - vykonání instrukce trvá 3 nebo 4 takty, délka instrukce 1 slovo (2 bajty)

- `rjmp k`
  - relativní odskok na adresu k, PC = PC + k + 1
  - absolutní adresu spočítá překladač
  - dovoluje skoky v rozsahu paměti PC – 2K + 1 a PC + 2K (slov)
  - vykonání instrukce trvá 2 takty, délka instrukce 1 slovo (2 bajty)
- `jmp k`
  - absolutní skok na adresu k, PC = k
  - dovoluje skákat v celém rozsahu paměti programu (4M slov nebo-li 8MB)
  - vykonání instrukce trvá 3 takty, délka instrukce 2 slova (4 bajty)
  - jen u řady ATmega (mimo ATmega8)
- `ijmp`
  - nepřímé skok registru Z, PC = Z
  - dovoluje skok v dolních 64k slovech (128kB) programové paměti
  - vykonání instrukce trvá 2 takty , délka instrukce 1 slovo (2 bajty)