Assembly AVR

SRAM

Prof. Roberto de Matos

roberto.matos@ifsc.edu.br

INSTITUTO FEDERAL
Santa Catarina
Câmpus São José

Objetivos

Objetivo

- Entender os modos de endereçamento da SRAM (Direto, Indireto, Indireto com pós-incremento, pré-decremento e deslocamento)
- Entender as instruções `lds`, `sts`, `ld`, `st`, `ldd` e `std`
- Utilizar o montador de forma mais sofisticada para facilitar a codificação:
  - Diretivas `.BYTE`, `.ORG`, `.DSEG`
  - Operadores ($+$, $-$) e funções (`LOW()` e `HIGH()`)

Objetivo

- Entender os modos de endereçamento da SRAM (Direto, Indireto, Indireto com pós-incremento, pré-decremento e deslocamento)
- Entender as instruções `lds`, `sts`, `ld`, `st`, `ldd` e `std`
- Utilizar o montador de forma mais sofisticada para facilitar a codificação:
  - Diretivas `.BYTE`, `.ORG`, `.DSEG`
  - Operadores ($+$, $-$) e funções (`LOW()` e `HIGH()`)

Referências:

- Datasheet do ATmega328p
- Manual online do montador
- Manual do conjunto de instruções do AVR

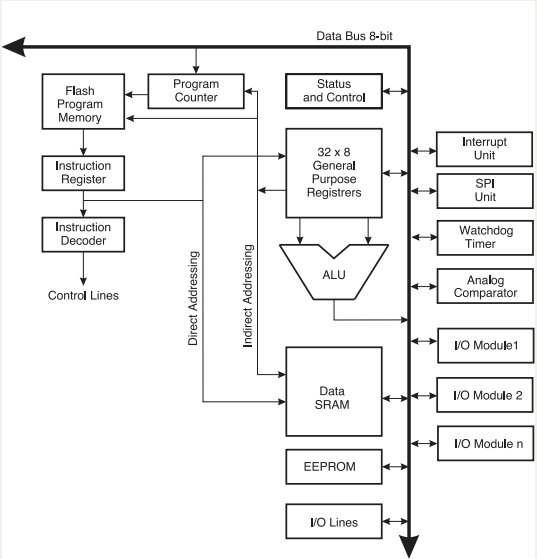
Motivação

Registradores e periféricos bastam para um MCU?

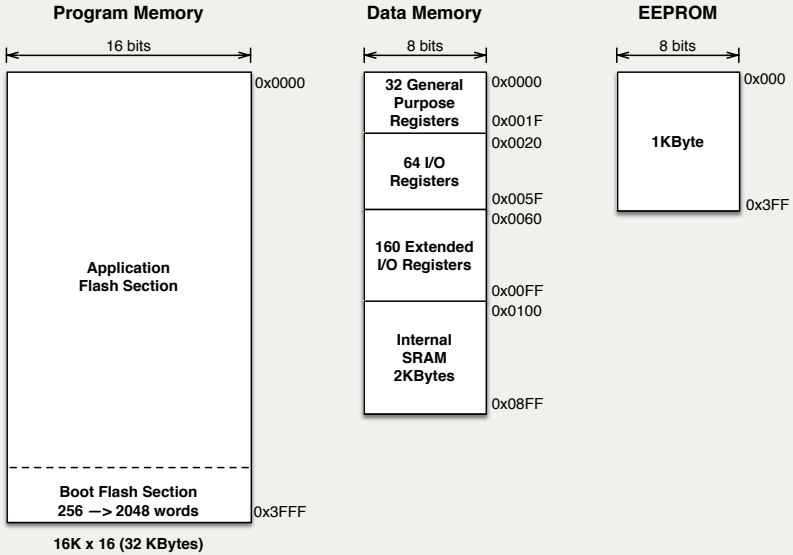
- Onde armazenar variáveis de forma temporária?
- Como processar vetores de dados?

Relembrando ...

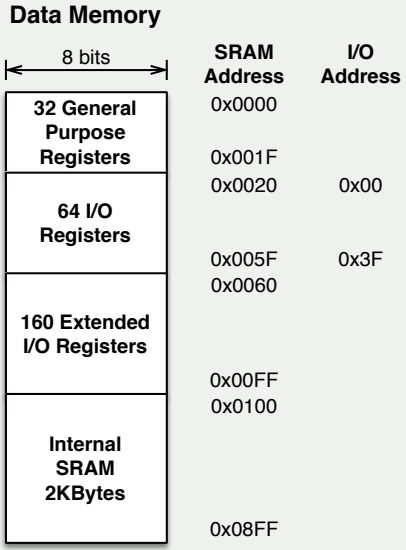
Núcleo do AVR



Memórias ATmega328p

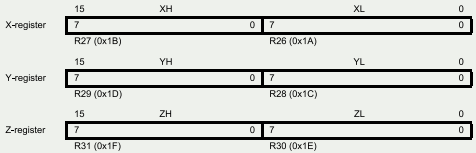


Endereçamento memória de dados



Registradores de Uso Geral

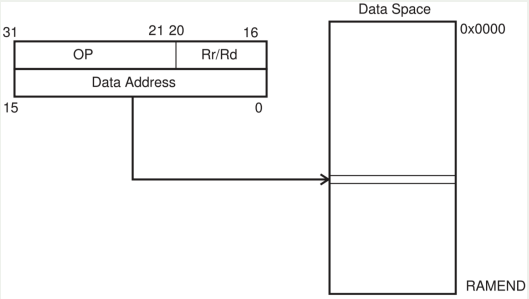
7	0	Addr.	
R0		0x00	
R1		0x01	
R2		0x02	
…			
R13		0x0D	
R14		0x0E	
R15		0x0F	
R16		0x10	
R17		0x11	
…			
R26		0x1A	X-register Low Byte
R27		0x1B	X-register High Byte
R28		0x1C	Y-register Low Byte
R29		0x1D	Y-register High Byte
R30		0x1E	Z-register Low Byte
R31		0x1F	Z-register High Byte



Modos de Endereçamento: Memória de Dados

Direto

- O conteúdo de uma posição da memória de dados é acessada usando seu endereço (direto).



- Sintaxe:

  `lds` Rd, k ; $Rd \leftarrow (k)$

  `sts` k, Rr ; $(k) \leftarrow Rr$

- Onde:

  $0 \leq r/d \leq 31$

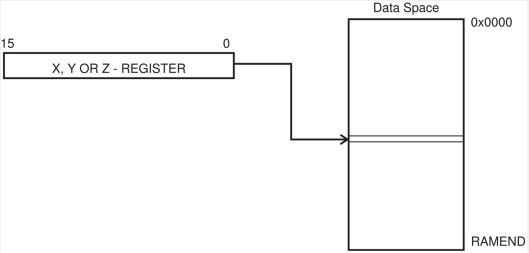
  $0 \leq k \leq 65535$

- Exemplo:

```
1  LDS R0, 0x0100
2  STS 0x0101, R0
```

Indireto

- O conteúdo de uma posição da memória de dados é acessada usando os ponteiros X, Y ou Z.



- Sintaxe:

  `ld` Rd, index ; $Rd \leftarrow (index)$

  `st` index, Rr ; $(index) \leftarrow Rr$

- Onde:

  $0 \leq r/d \leq 31$

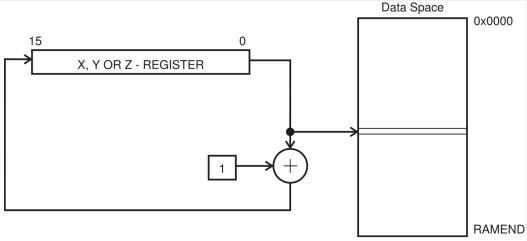
  $index = X, Y \text{ ou } Z$

- Exemplo:

```
1  LDI r27, 0x01   ; Byte mais significativo de X (XH)
2  LDI r26, 0x00   ; Byte menos significativo de X (XL)
3  LD r0, X        ; X está apontando o endereço 0x0100
4  INC r0
5  ST X, r0
```


Indireto com Pós-Incremento

- O conteúdo de uma posição da memória de dados é acessada usando os ponteiros X, Y ou Z, e o ponteiro é incrementado após o acesso.



- Exemplo:

```
1   LDI r27, 0x01   ; Byte mais significativo de X (XH)
2   LDI r26, 0x00   ; Byte menos significativo de X (XL)
3 Loop:
4    INC r0
5    ST X+, r0
6    RJMP loop
```

- Sintaxe:

`ld` Rd, index+ ; $Rd \leftarrow (index)$
; $index \leftarrow index + 1$

`st` index+, Rr ; $(index) \leftarrow Rr$
; $index \leftarrow index + 1$

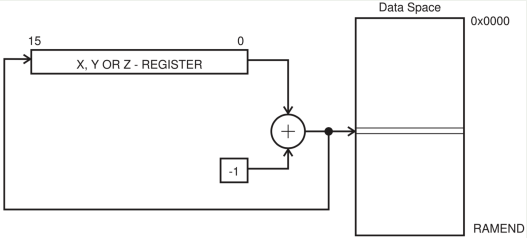
- Onde:

$0 \leq r/d \leq 31$

$index = X, Y \text{ ou } Z$

Indireto com Pré-Decremento

- O ponteiro X, Y ou Z é decrementado e depois é usado para acessar o conteúdo de uma posição da memória de dados.



- Exemplo:

```
1   LDI r27, 0x01   ; Byte mais significativo de X (XH)
2   LDI r26, 0x0B   ; Byte menos significativo de X (XL)
3 Loop:
4   INC r0
5   ST -X, r0
6   RJMP loop
```

- Sintaxe:

`ld` Rd, −index ; $index \leftarrow index - 1$
; $Rd \leftarrow (index)$

`st` −index, Rr ; $index \leftarrow index - 1$
; $(index) \leftarrow Rr$

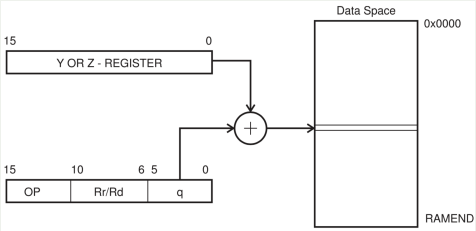
- Onde:

$0 \leq r/d \leq 31$

$index = X, Y \text{ ou } Z$

Indireto com Deslocamento

- O endereço do operando é o resultado do conteúdo do registro Y ou Z adicionado ao valor contido em seis bits da palavra de instrução. O ponteiro não sofre alteração.



- Exemplo:

```
1 LDI r29, 0x01  ; Byte mais significativo de Y  (YH = r29)
2 LDI r28, 0x00  ; Byte menos significativo de Y (YL = r28)
3 INC r0
4 STD Y+2, r0    ; Y = 0x0100 "+2" = 0x0102
5 INC r0
6 STD Y+3, r0    ; Y continua sendo 0x0100. Agora com + 3,
7                ; a operação acontece no endereço 0x0103
```

- Sintaxe:

  `ldd` Rd, index+q ; $Rd \leftarrow (index + q)$

  `std` index+q, Rr ; $(index + q) \leftarrow Rr$

- Onde:

  $0 \leq r/d \leq 31$

  $index = Y \text{ ou } Z$

  $0 \leq q \leq 63$

Memória de dados: Resumo Modos de Endereçamento

- 5 modos de endereçamento: Direto, Indireto, Indireto com Pós-incremento, Indireto com Pré-decremento e Indireto com Deslocamento.
- Os registradores R26 ao R31 são utilizados como ponteiros para o endereçamento indireto.
- O endereçamento indireto com deslocamento alcança 63 endereços a partir do endereço base (Y+ ou Z+).
- Ao usar os modos de endereçamento indireto com pré-decremento e pós-incremento automático, os ponteiros X, Y e Z são modificados.
- Toda a memória dados pode ser acessada com esses modos de endereçamentos.

Diretivas do montador[a]

[a] Manual online do montador

Alocando espaço na SRAM

- **.BYTE** reserva bytes para variáveis na RAM.

- Diretivas de Segmentos:

  .CSEG ; Flash (programa)

  .DSEG ; RAM (dados)

  .ESEG ; EEPROM

- **.ORG** permite definir um endereço absoluto.

Alocando espaço na SRAM

- **.BYTE** reserva bytes para variáveis na RAM.

- Diretivas de Segmentos:

  .CSEG ; Flash (programa)

  .DSEG ; RAM (dados)

  .ESEG ; EEPROM

- **.ORG** permite definir um endereço absoluto.

- Exemplo:

```
1  .INCLUDE <m328Pdef.INC>
2
3  .DSEG            ; Segmento da Memória RAM (dados)
4  .ORG SRAM_START  ; 0x0100 para o ATmega328p
5
6    var1: .BYTE 1 ; aloca 1 byte no rótulo var1.
7                  ; var1 é o rótulo do endereço (0x0100)
8                  ; do byte alocado.
9
10   var2: .BYTE 2 ; aloca 2 bytes a partir do rótulo var2.
11                 ; var2 é o rótulo do endereço (0x0101)
12                 ; do PRIMEIRO byte  alocado.
13                 ; O endereço do segundo byte é var2+1 (0x0103)
14
15   var3: .BYTE 1 ; outra variável
16
17 .CSEG            ; Segmento da Memória de Flash (programa)
18 start:
19   LDI r19,15
20   STS var1,r19  ; inicializa o endereço var1 com 15
21
22   RJMP start
```


Alocando espaço na SRAM

- Explicando o exemplo:
  - Acima, a diretiva `.ORG` é usada para definir o início do `.DSEG` para o endereço `SRAM_START`, que é definido no arquivo de include. No nosso caso, o arquivo de include é o `m328Pdef.inc` e o `SRAM_START` é `0x0100`. Os rótulos `var1`, `var2` e `var3` são então usados como sinônimos para o primeiro endereço da memória de dados onde o espaço está sendo alocado.

  - Observe que não podemos gravar valores na memória de dados ao programar o chip, só podemos alocar espaço para eles e inicializá-los em tempo de execução. Como apresentado anteriormente, os valores podem ser gravados na SRAM sem alocar espaço, mas o método acima nos dá nomes (rótulos) significativos para os dados que estamos armazenando e nos permite controlar a quantidade de SRAM que estamos usando.

Expressões do Montador[a]

[a] Manual online do montador

Uso de operadores e funções

- Uso do operador +

```
1  .INCLUDE <m328Pdef.INC>
2
3  .DSEG
4  .ORG SRAM_START
5    uint: .BYTE 2
6
7  .CSEG
8  start:
9      LDI r16,0xCD      ; carrega 0xCD em r16
10     LDI r17,0xAB      ; carrega 0xAB em r17
11
12     STS uint,r16      ; inicializa uint com
13     STS uint+1,r17    ; o valor 0xABCD
14
15     LDS r0,uint       ; carrega o valor de
16     LDS r1,uint+1     ; int em r1:r0
17
18     RJMP start
```


Uso de operadores e funções

- Uso do operador +

```
1  .INCLUDE <m328Pdef.INC>
2
3  .DSEG
4  .ORG SRAM_START
5    uint: .BYTE 2
6
7  .CSEG
8  start:
9      LDI r16,0xCD      ; carrega 0xCD em r16
10     LDI r17,0xAB      ; carrega 0xAB em r17
11
12     STS uint,r16      ; inicializa uint com
13     STS uint+1,r17    ; o valor 0xABCD
14
15     LDS r0,uint       ; carrega o valor de
16     LDS r1,uint+1     ; int em r1:r0
17
18     RJMP start
```

- Uso das funções `LOW()` e `HIGH()`

```
1  .INCLUDE <m328Pdef.INC>
2
3  .DSEG
4  .ORG SRAM_START
5    char: .BYTE   1
6
7  .CSEG
8  start:
9    LDI   r16,0xFF       ; carrega 0xFF em r16
10
11   LDI   XL,LOW(char)   ; inicializa o ponteiro X
12   LDI   XH,HIGH(char)  ; com o endereço de char
13
14   ST    X,r16          ; armazena o valor de R16 no char
15
16   RJMP start
```


Experimentos

Experimentos no MPLAB IDE

- Simule todos os trechos de códigos apresentados. Modifique os valores das localizações de memória afetadas para verificar se o funcionamento está coerente.
- Faça um programa que some duas variáveis de 8 bits:

$$\mathbf{C = A + B}$$

- Refaça o exercício anterior, mas considerando variáveis de 16 bits.
- Observações:
  - Considere as variáveis A, B e C armazenadas sequencialmente a partir do primeiro endereço da memória SRAM.
  - Para o exercício com variáveis de 16 bits, considere a ordenação dos bytes como *little-endian*[1], ou seja, o byte menos significativo é armazenado no menor endereço.

[1] *Little-endian vs. Big-endian*