# MC404

# ORGANIZAÇÃO BÁSICA DE COMPUTADORES E LINGUAGEM DE MONTAGEM

**2010**

**Prof. Paulo Cesar Centoducatte**
**Prof. Mario Lúcio Côrtes**
**Prof. Ricardo Pannain**

# MC404

## ORGANIZAÇÃO BÁSICA DE COMPUTADORES E LINGUAGEM DE MONTAGEM

**"Macros,**

**Montagem Condicional de Código**

**e Uso da EEPROM"**

# Macros
## Sumário

- Macros
- Montagem Condicional de Código
- Uso da EEPROM

# Macro

**Código que é replicado a cada chamada, com parâmetros substituídos**

**Exemplo de definição de macro**

```
.MACRO ADDI ;(Rd, k)
        subi    @0, -(@1)
.ENDMACRO
```

**Exemplo de chamada**

```
addi , 0xA0
addi r20, 0x77
```

# Macros

**; Troca o conteúdo de dois registradores sem usar variável auxiliar!:**

```
.MACRO SWAP ;(Rd, Rs)
      eor    @0, @1
      eor    @1, @0
      eor    @0, @1
.ENDMACRO
```

| R0 | R1 |
|---|---|
| R0 | R1 |
| $R0 \oplus R1$ | R1 |
| $R0 \oplus R1$ | $(R0 \oplus R1) \oplus R1$ |
| $R0 \oplus R1$ | R0 |
| $(R0 \oplus R1) \oplus R0$ | R0 |
| $R0 \oplus (R0 \oplus R1)$ | R0 |
| R1 | R0 |

OBS: propriedades do XOR

$R0 \oplus (R1 \oplus R2) = (R0 \oplus R1) \oplus R2$
$R0 \oplus 0 = R0$
$R0 \oplus R1 = R1 \oplus R0$

# Macro

```
;Soma uma constante de 8 bits a um registrador

    .MACRO ADDI ;(Rd, k)
        subi    @0, -(@1)
    .ENDMACRO



; Soma uma constante de 16 bits a um par de
;registradores (X, Y ou Z)
    .MACRO ADDIW ;(RdL:RdH, k)
        subi    @0L, LOW(-@1)
        sbci    @0H, HIGH(-@1)
    .ENDMACRO
```

# Macro – Montagem Condicional

```
; I/O
    .macro input
      .if @1 < 0x40
            in  @0, @1
      .else
            lds @0, @1
      .endif
    .endmacro
```

# Macro – Montagem Condicional

```
; I/O

  .macro output
   .if @0 < 0x40
        out @0, @1
   .else
        sts @0, @1
   .endif
  .endmacro
```

# Macro – Montagem Condicional

```
; Branch if Bit in I/O-Register is Set
   .macro bbis ;port,bit,target
   .if @0 < 0x20
         sbic    @0, @1
         rjmp    @2
   .elif @0 < 0x40
         in      zl, @0
         sbrc    zl, @1
         rjmp    @2
    .else
         lds     zl, @0
         sbrc    zl, @1
         rjmp    @2
    .endif
   .endmacro
```

# Macro e Montagem Condicional

**Branch if Bit in I/O-Register is Cleared**

```
.macro bbic ;port,bit,target
  .if @0 < 0x20
        sbis  @0, @1
        rjmp  @2
 .elif @0 < 0x40
        in      zl, @0
        sbrs  zl, @1
        rjmp  @2
  .else
        lds     zl, @0
        sbrs   zl, @1
        rjmp   @2
  .endif
.endmacro
```

# Montagem Condicional

```
.EQU clock=40000000

.IF clock>4000000
    .EQU divider=4
.ELSE
    .EQU divider=2
.ENDIF
```

<u>Mais exemplos de uso de IF e ENDIF</u>

# Macros Aninhadas

```
.macro mult8  ; macro para multiplica dois números
              ; de 8 bits sem sinal
    ; Parâmetros de entrada:
        ; @0 e @1: dois registradores quaisquer
        ; (números para multiplcar)
    ; destroi: r0, r1, @2

        mul @0,@1
.endmacro
```

# Macros Aninhadas

```
.macro callmult8  ; macro para multiplicar dois números
                  ; de 8 bits sem sinal
    ; Parâmetros de entrada:
        ; @0 e @1: dois registradores quaisquer
        ; (números para multiplcar)
        ; @2 um dos pares X, Y ou Z: endereço na RAM
        ; para colocar o produto no formato little endian
        ; destroi: r0, r1, @2

          mult8 @0,@1   ;  invoca a macro mult8 repassando os
                        ; parametros @0 e @1
          st @2+,r0
          st @2, r1
.endmacro
```

# Uso de Macro

```
RESET:                            ; início do programa
    ldi   r16,low(RAMEND)         ; fim da RAM: definido em
                                  ; m88def.inc
    out   SPL,r16                 ; inicializa Stack Pointer
    ldi   r16,high(RAMEND)
    out   SPH, r16
    ldi   yh, high(SRAM_START)    ; Área de RAM onde será
                                  ; colocado o resultado
    ldi   yl, low(SRAM_START)
    ldi   r16,2
    ldi   r17,5
    callmult8  r16,r17, y         ;  Expande as duas macros

    rjmp PC
```

# Uso da EEPROM

- **EEPROM Data Memory**
  - **The ATmega48/88/168 contains 256/512/512 bytes of data EEPROM memory.**
  - **It is organized as a separate data space, in which single bytes can be read and written.**
  - **The EEPROM has an endurance of at least 100,000 write/erase cycles.**
  - **The access between the EEPROM and the CPU is described in the following, specifying the:**
    - **EEPROM Address Registers → EEAR**
    - **EEPROM Data Register → EEDR**
    - **EEPROM Control Register → EECR**

# EEPROM

## The EEPROM Address Register – EEARH and EEARL

| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | |
|---|---|---|---|---|---|---|---|---|---|
| | – | – | – | – | – | – | – | EEAR8 | EEARH |
| | EEAR7 | EEAR6 | EEAR5 | EEAR4 | EEAR3 | EEAR2 | EEAR1 | EEAR0 | EEARL |
| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| Read/Write | R | R | R | R | R | R | R | R/W | |
| | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | X | |
| | X | X | X | X | X | X | X | X | |

- **Bits 15..9 – Res: Reserved Bits**

- **Bits 8..0 – EEAR8..0: EEPROM Address**

# EEPROM

## The EEPROM Data Register – EEDR

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | MSB | | | | | | | LSB | EEDR |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

- **Bits 7..0 – EEDR7.0: EEPROM Data**

# EEPROM

## The EEPROM Control Register – EECR

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | – | – | EEPM1 | EEPM0 | EERIE | EEMPE | EEPE | EERE |
| Read/Write | R | R | R/W | R/W | R/W | R/W | R/W | R/W |
| Initial Value | 0 | 0 | X | X | 0 | 0 | X | 0 |

- **Bits 7..6 – Res: Reserved Bits**

- **Bits 5, 4 – EEPM1 and EEPM0: EEPROM Programming Mode Bits**

# EEPROM

**Bits 5, 4 – EEPM1 and EEPM0: EEPROM Programming Mode Bits**

**Table 5-1.** EEPROM Mode Bits

| EEPM1 | EEPM0 | Programming Time | Operation |
|---|---|---|---|
| 0 | 0 | 3.4 ms | Erase and Write in one operation (Atomic Operation) |
| 0 | 1 | 1.8 ms | Erase Only |
| 1 | 0 | 1.8 ms | Write Only |
| 1 | 1 | – | Reserved for future use |

# EEPROM

- **Bit 3 – EERIE: EEPROM Ready Interrupt Enable**

Writing EERIE to one enables the EEPROM Ready Interrupt if the I bit in SREG is set. Writing EERIE to zero disables the interrupt. The EEPROM Ready interrupt generates a constant interrupt when EEPE is cleared. The interrupt will not be generated during EEPROM write or SPM.

- **Bit 2 – EEMPE: EEPROM Master Write Enable**

The EEMPE bit determines whether setting EEPE to one causes the EEPROM to be written. When EEMPE is set, setting EEPE within four clock cycles will write data to the EEPROM at the selected address If EEMPE is zero, setting EEPE will have no effect. When EEMPE has been written to one by software, hardware clears the bit to zero after four clock cycles. See the description of the EEPE bit for an EEPROM write procedure.

# EEPROM

- **Bit 1 – EEPE: EEPROM Write Enable**

The EEPROM Write Enable Signal EEPE is the write strobe to the EEPROM. When address and data are correctly set up, the EEPE bit must be written to one to write the value into the EEPROM. The EEMPE bit must be written to one before a logical one is written to EEPE, otherwise no EEPROM write takes place. The following procedure should be followed when writing the EEPROM (the order of steps 3 and 4 is not essential):

1. Wait until EEPE becomes zero.
2. Wait until SELFPRGEN in SPMCSR becomes zero.
3. Write new EEPROM address to EEAR (optional).
4. Write new EEPROM data to EEDR (optional).
5. Write a logical one to the EEMPE bit while writing a zero to EEPE in EECR.
6. Within four clock cycles after setting EEMPE, write a logical one to EEPE.

# Uso da EEPROM

**e2prom-rotinas.asm**