

# MC404

---

## ORGANIZAÇÃO BÁSICA DE COMPUTADORES E LINGUAGEM DE MONTAGEM

2009

Prof. Paulo Cesar Centoducatte

[ducatte@ic.unicamp.br](mailto:ducatte@ic.unicamp.br)

[www.ic.unicamp.br/~ducatte](http://www.ic.unicamp.br/~ducatte)

# MC404

---

## ORGANIZAÇÃO BÁSICA DE COMPUTADORES E LINGUAGEM DE MONTAGEM

**“Subrotinas  
e  
Pilha”**

# Subrotinas

## Sumário

---

- Instruções
- Uso da Pilha
- Regras Básicas

# Instruções

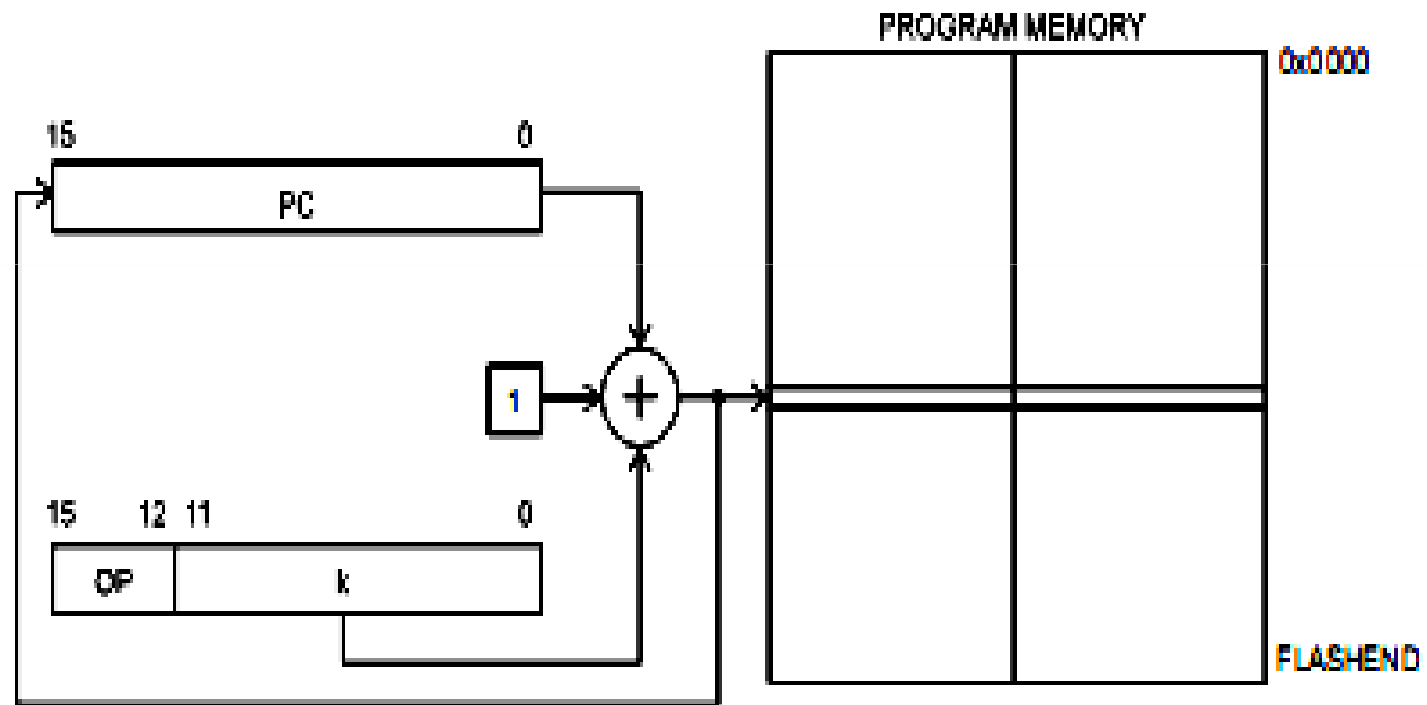
---

- **Rcall**
- **Icall**
- **Call**
- **Ret**
- **Reti**
- **Push**
- **Pop**

# RCALL

## Relative Program Addressing, RJMP and RCALL

Figure 13. Relative Program Memory Addressing



Program execution continues at address  $PC + k + 1$ . The relative address  $k$  is from -2048 to 2047.

# RCALL

## RCALL – Relative Call to Subroutine

### Description:

Relative call to an address within  $PC - 2K + 1$  and  $PC + 2K$  (words). The return address (the instruction after the RCALL) is stored onto the Stack. (See also CALL). In the assembler, labels are used instead of relative operands. For AVR microcontrollers with Program memory not exceeding 4K words (8K bytes) this instruction can address the entire memory from every address location. The Stack Pointer uses a post-decrement scheme during RCALL.

### Operation:

- (i)  $PC \leftarrow PC + k + 1$  Devices with 16 bits PC, 128K bytes Program memory maximum.
- (ii)  $PC \leftarrow PC + k + 1$  Devices with 22 bits PC, 8M bytes Program memory maximum.

### Syntax:

(i) RCALL k

### Operands:

$-2K \leq k < 2K$

### Program Counter:

$PC \leftarrow PC + k + 1$

### Stack:

$STACK \leftarrow PC + 1$   
 $SP \leftarrow SP - 2$  (2 bytes, 16 bits)

(ii) RCALL k

$-2K \leq k < 2K$

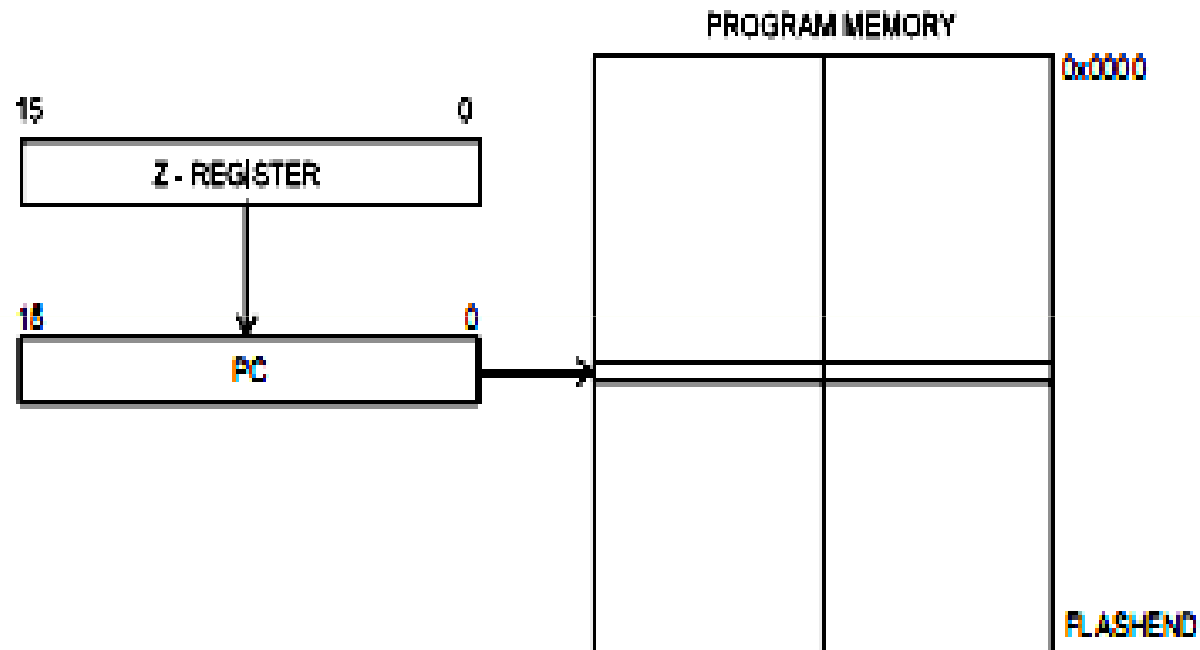
$PC \leftarrow PC + k + 1$

$STACK \leftarrow PC + 1$   
 $SP \leftarrow SP - 3$  (3 bytes, 22 bits)

# ICALL

## Indirect Program Addressing, JMP and ICALL

Figure 12. Indirect Program Memory Addressing



Program execution continues at address contained by the Z-register (i.e., the PC is loaded with the contents of the Z-register).

# ICALL

## ICALL – Indirect Call to Subroutine

### Description:

Indirect call of a subroutine pointed to by the Z (16 bits) Pointer Register in the Register File. The Z-pointer Register is 16 bits wide and allows call to a subroutine within the lowest 64K words (128K bytes) section in the Program memory space. The Stack Pointer uses a post-decrement scheme during ICALL.

This instruction is not available in all devices. Refer to the device specific instruction set summary.

### Operation:

- (i)  $PC(15:0) \leftarrow Z(15:0)$  Devices with 16 bits PC, 128K bytes Program memory maximum.
- (ii)  $PC(15:0) \leftarrow Z(15:0)$  Devices with 22 bits PC, 8M bytes Program memory maximum.  
 $PC(21:16) \leftarrow 0$

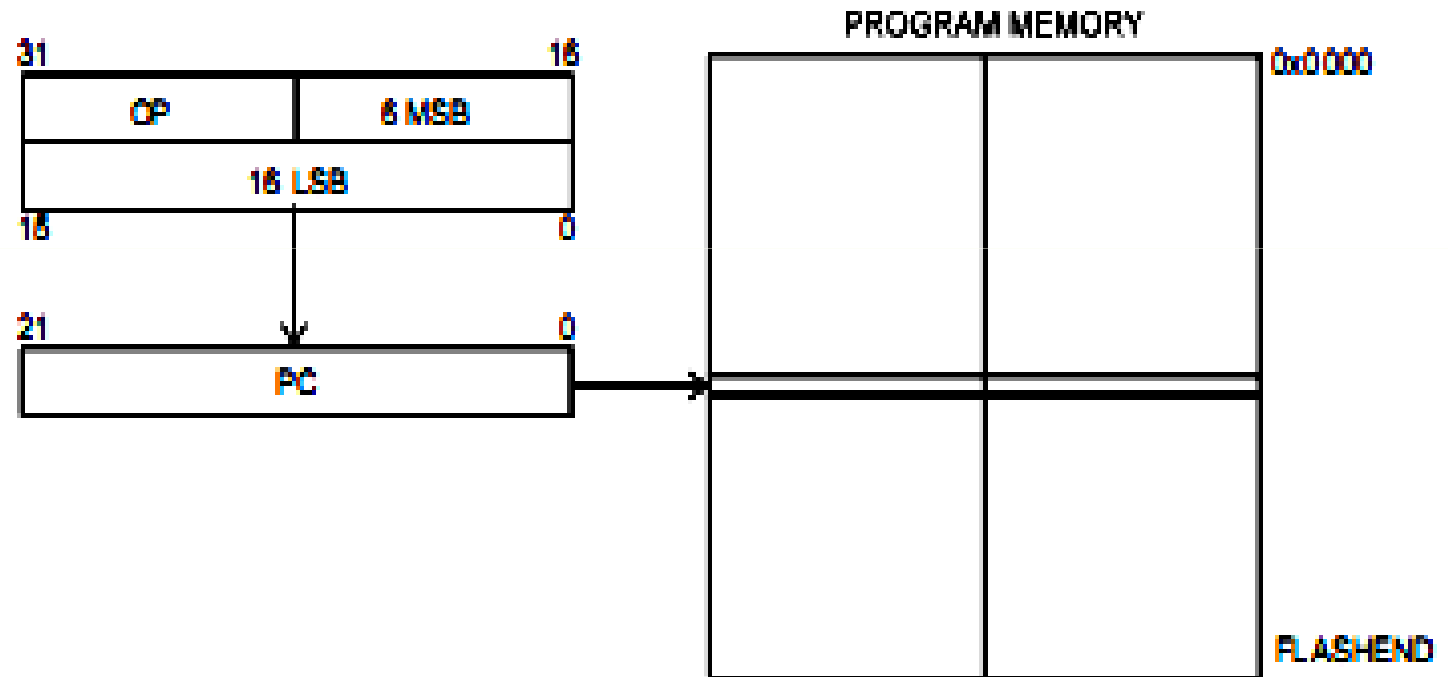
	Syntax:	Operands:	Program Counter:	Stack:
(i)	ICALL	None	See Operation	$STACK \leftarrow PC + 1$ $SP \leftarrow SP - 2$ (2 bytes, 16 bits)
(ii)	ICALL	None	See Operation	$STACK \leftarrow PC + 1$ $SP \leftarrow SP - 3$ (3 bytes, 22 bits)



# CALL

## Direct Program Addressing, JMP and CALL

Figure 11. Direct Program Memory Addressing



Program execution continues at the address immediate in the instruction word.

# CALL

## CALL – Long Call to a Subroutine

### Description:

Calls to a subroutine within the entire Program memory. The return address (to the instruction after the CALL) will be stored onto the Stack. (See also RCALL). The Stack Pointer uses a post-decrement schema during CALL.

This instruction is not available in all devices. Refer to the device specific instruction set summary.

### Operation:

- (i)  $PC \leftarrow k$       Devices with 16 bits PC, 128K bytes Program memory maximum.
- (ii)  $PC \leftarrow k$       Devices with 22 bits PC, 8M bytes Program memory maximum.

	Syntax:	Operands:	Program Counter	Stack:
(i)	CALL k	$0 \leq k < 64K$	$PC \leftarrow k$	$STACK \leftarrow PC+2$ $SP \leftarrow SP-2$ , (2 bytes, 16 bits)
(ii)	CALL k	$0 \leq k < 4M$	$PC \leftarrow k$	$STACK \leftarrow PC+2$ $SP \leftarrow SP-3$ (3 bytes, 22 bits)

# RET

---

## RET – Return from Subroutine

---

### Description:

Returns from subroutine. The return address is loaded from the STACK. The Stack Pointer uses a pre-increment scheme during RET.

### Operation:

- (i)  $PC(15:0) \leftarrow \text{STACK}$  Devices with 16 bits PC, 128K bytes Program memory maximum.
- (ii)  $PC(21:0) \leftarrow \text{STACK}$  Devices with 22 bits PC, 8M bytes Program memory maximum.

	Syntax:	Operands:	Program Counter:	Stack:
(i)	RET	None	See Operation	$SP \leftarrow SP + 2$ , (2bytes, 16 bits)
(ii)	RET	None	See Operation	$SP \leftarrow SP + 3$ , (3bytes, 22 bits)

# PUSH

---

## PUSH – Push Register on Stack

---

### Description:

This instruction stores the contents of register  $R_r$  on the STACK. The Stack Pointer is post-decremented by 1 after the PUSH.

This instruction is not available in all devices. Refer to the device specific instruction set summary.

### Operation:

(i)  $STACK \leftarrow R_r$

### Syntax:

(i) PUSH  $R_r$

### Operands:

$0 \leq r \leq 31$

### Program Counter:

$PC \leftarrow PC + 1$

### Stack:

$SP \leftarrow SP - 1$

# POP

---

## POP – Pop Register from Stack

---

### Description:

This instruction loads register  $Rd$  with a byte from the STACK. The Stack Pointer is pre-incremented by 1 before the POP.

This instruction is not available in all devices. Refer to the device specific instruction set summary.

### Operation:

(i)  $Rd \leftarrow \text{STACK}$

### Syntax:

(i) POP  $Rd$

### Operands:

$0 \leq d \leq 31$

### Program Counter:

$PC \leftarrow PC + 1$

### Stack:

$SP \leftarrow SP + 1$

# Pilha

The AVR Stack Pointer is implemented as two 8-bit registers in the I/O space. The number of bits actually used is implementation dependent. Note that the data space in some implementations of the AVR architecture is so small that only SPL is needed. In this case, the SPH Register will not be present.

Bit	15	14	13	12	11	10	9	8	
	SP15	SP14	SP13	SP12	SP11	SP10	SP9	SP8	SPH
	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0	SPL
	7	6	5	4	3	2	1	0	
Read/Write	RW	RW	RW	RW	RW	RW	RW	RW	
	RW	RW	RW	RW	RW	RW	RW	RW	
Initial Value	RAMEND	RAMEND	RAMEND	RAMEND	RAMEND	RAMEND	RAMEND	RAMEND	
	RAMEND	RAMEND	RAMEND	RAMEND	RAMEND	RAMEND	RAMEND	RAMEND	

# Espaço de I/O

---

## IN - Load an I/O Location to Register

---

### Description:

Loads data from the I/O Space (Ports, Timers, Configuration Registers etc.) into register Rd in the Register File.

### Operation:

(i)  $Rd \leftarrow I/O(A)$

### Syntax:

(i) IN Rd,A

### Operands:

$0 \leq d \leq 31, 0 \leq A \leq 83$

### Program Counter:

$PC \leftarrow PC + 1$

# Espaço de I/O

---

## OUT – Store Register to I/O Location

---

### Description:

Stores data from register  $R_r$  in the Register File to I/O Space (Ports, Timers, Configuration Registers etc.).

### Operation:

(i)  $I/O(A) \leftarrow R_r$

### Syntax:

(i) OUT A,  $R_r$

### Operands:

$0 \leq r \leq 31, 0 \leq A \leq 63$

### Program Counter:

$PC \leftarrow PC + 1$



# Pilha

---

- **Inicialização do Apontador de Pilha**

**.def r =r16 ; usado na inicialização dos apontadores  
; da pilha e início da RAM**

**ldi r,low(RAMEND) ; fim da RAM: definido em m88def.inc  
out SPL,r ; inicializa Stack Pointer  
ldi r,high(RAMEND)  
out SPH, r**

# Regras Básicas para Programação de Subrotinas

---

- **NomeRotina:**
- **Salvar contexto (em geral na pilha)**
  - **Registradores utilizados na subrotina**
  - **Flags (se necessário) – Registrador SREG**
- **Corpo da Subrotina**
- **Restaurar registradores salvos**
- **RET**