

# Arquitetura de Computadores

---

PROF. DR. ISAAC

# Instruções do MSC-51

Tipo	Quantidade
Aritméticas	24
Lógicas	25
Transferência (cópia) de Dados	28
Booleanas	17
Saltos	17

**Instruções aritméticas:** envolvem operações do tipo soma, subtração, multiplicação, divisão, incremento e decremento.

**Instruções lógicas:** fazem operações bit a bit com registradores e também rotações.

# Instruções do MSC-51

**Instruções de transferência (cópia) de dados:** copiam bytes entre os diversos registradores e a RAM interna.

**Instruções booleanas:** essas instruções são denominadas booleanas porque trabalham com variáveis lógicas (variável de 1 bit). Como o próprio nome sugere, elas são talhadas para resolver expressões booleanas.

**Instruções de salto:** desviam o fluxo de execução do programa, chamam subrotinas, fazem desvios condicionais e executam laços de repetição.

# Instruções do 8051

Lógica	Aritmética	Memória	Outros
ANL ✓	ADD ✓	MOV ✓	NOP
ORL ✓	ADDC ✓	MOVC ✓	RET e RETI
XRL ✓	SUBB ✓	MOVX ✓	ACALL e LCALL
CLR ✓	MUL ✓	PUSH	JMP
CPL	DIV ✓	POP	AJMP
RL	INC ✓	XCH ✓	LJMP
RLC	DEC ✓	XCHD ✓	SJMP
RR	DA		JB e JNB
RRC			JZ e JNZ
SWAP			JC e JNC
SETB			JBC
			DJNZ
			CJNE

# Instruções do 8051

Lógica	Aritmética	Memória	Outros
ANL ✓	ADD ✓	MOV ✓	NOP
ORL ✓	ADDC ✓	MOVC ✓	RET e RETI
XRL ✓	SUBB ✓	MOVB ✓	ACALL e LCALL
CLR ✓	MUL ✓	PUSH	JMP
CPL	DIV ✓	POP	AJMP
RL	INC ✓	XCH ✓	LJMP
RLC	DEC ✓	XCHD ✓	SJMP
RR	DA		JB e JNB
RRC			JZ e JNZ
SWAP			JC e JNC
SETB			JBC
			DJNZ
			CJNE

# Instruções Lógicas:

## Operações Lógicas com o Acumulador

---

# Instruções de operações lógicas com o acumulador

---

**CLR A:** inicializa o acumulador com zeros;

**CPL A:** calcula o complemento 1 do acumulador (inverter todos os bits);

**RL A:** rotaciona o acumulador à esquerda;

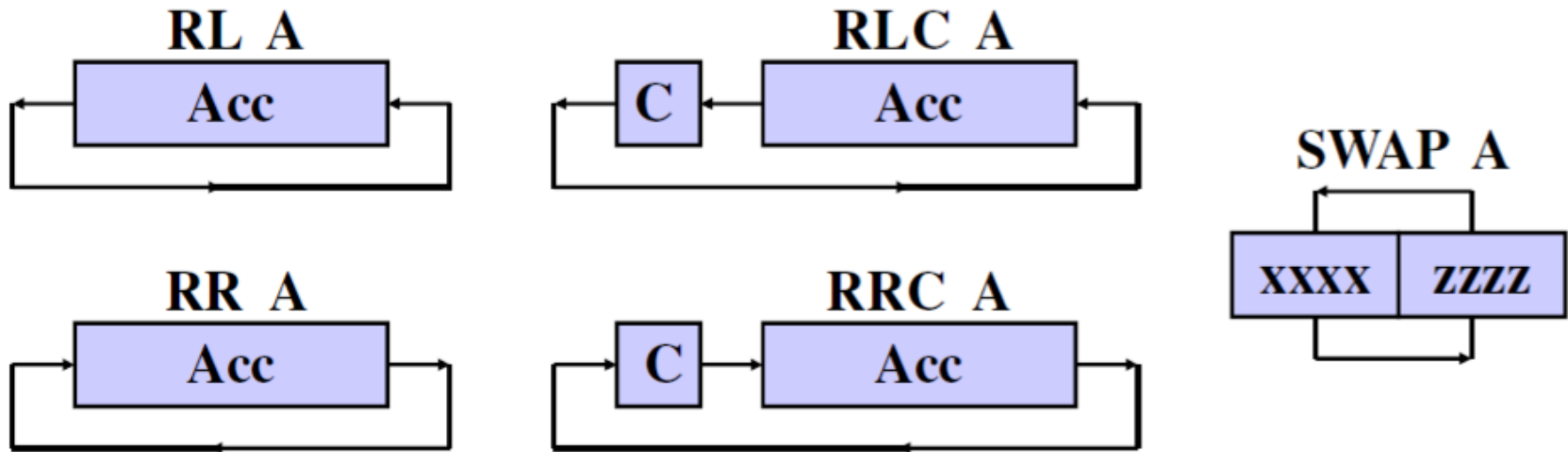
**RLC A:** rotaciona o acumulador à esquerda, usando o carry;

**RR A:** rotaciona o acumulador à direita;

**RRC A:** rotaciona o acumulador à direita, usando o carry;

**SWAP A:** troca de posição as nibbles do acumulador.

# Instruções de operações lógicas com o acumulador



Operações Lógicas com o acumulador.



# Instruções de operações lógicas com o acumulador

		Bytes	MC	Op
CLR	A	1	1	E4
CPL				F4
RL				23
RLC				33
RR				03
RRC				13
SWAP				C4

Instruções para operações lógicas com o acumulador.

**\*MC (Machine Cycle) ciclos de máquina.**

# Instruções Booleanas

---

# Instrução - CLR

---

**Operação:** CLR

**Função:** zera (colocar em 0) o valor do bit.

**Sintaxe:** CLR bit

**Descrição :** Zera (coloca 1) o valor do bit especificado.

**Exemplo:**

- CLR RS0
- CLR C
- CLR 20h.1

# Instrução - CPL

---

**Operação:** CPL

**Função:** Realiza o complemento (inverter o nível lógico) do bit.

**Sintaxe:** CPL bit

**Descrição :** CPL realiza o complemento do bit, que pode ser uma flag, bit de memória, porta ou Carry (C).

**Exemplo:**

- CPL P1.3
- CPL RS1
- CPL 20h.7

# Instrução - SETB

---

## **Operação: SETB**

**Função:** ativa (colocar em 1) o valor do bit.

**Sintaxe:** SETB *bit*

**Descrição :** Seta (coloca 1) no bit especificado, ou no Carry (C).

## **Exemplo:**

- SETB P1.3
- SETB C
- SETB RS0
- SETB 20h.0

# Instruções de operações lógicas com o acumulador

		Bytes	MC	Op1	Op2
CLR	C	1	1	C3	-
	bit	2	1	C2	bit
SETB	C	1	1	D3	-
	bit	2	1	D2	bit
CPL	C	1	1	B3	-
	bit	2	1	B2	bit

Instruções zerar/ativar/complementar um bit.

**\*MC (Machine Cycle) ciclos de máquina.**

# Instrução - AND / OR Booleano

---

A instrução **ANL C, bit** executa a operação “ $C \leftarrow C \text{ AND bit}$ ”.

A instrução **ORL C, bit** executa a operação “ $C \leftarrow C \text{ ORL bit}$ ”.

## Exemplo:

- ANL C, P0.1
- ANL C, 25h.1
- ORL C, /P0.1
- ORL C, B.2

# Instrução - AND / OR Booleano

---

			Bytes	MC	Op1	Op2
ANL	C,	bit	2	2	82	bit
		/bit	2	2	B0	bit
ORL	C,	bit	2	2	72	bit
		/bit	2	2	A0	bit

Instruções de AND e OR com um bit.



# Instrução - Transferência (Cópia) de Bits

---

Nas instruções de transferência (cópia) de bits, a flag carry (C) sempre está envolvida. Não é permitida a transferência (cópia) direta entre bits, mas sim apenas por intermédio do carry.

## **Exemplo:**

- `mov C, 20h.0`
- `mov Acc.0, C`
- `mov RS0, C`

# Instrução - Transferência (Cópia) de Bits

---

				Bytes	MC	Op1	Op2
MOV	C	,	bit	2	2	A2	bit
MOV	bit	,	C	2	2	92	bit

Instruções de transferência de bits.

# Instruções de Desvio:

## Salto Incondicionais

---

# Instrução - AJMP

## Operação: AJMP

---

**Função:** Desvio absoluto dentro do bloco de 2K da memória de programa

**Sintaxe:** AJMP *endereço*

**Descrição :** AJMP desvia o programa para o endereço indicado no parâmetro. Apenas código localizado no bloco de 2k do programa pode ser alvo deste desvio (um campo de 11 bits para o endereço de destino).

## Exemplo:

- AJMP LABEL
- AJMP 002h

# Instrução - LJMP

## Operação: LJMP

---

**Função:** Salto longo (long jump).

**Sintaxe:** LJMP *endereço*

**Descrição :** LJMP é capaz de desviar a execução para qualquer posição da memória de programa, pois oferece um campo de 16 bits para a especificação do endereço de destino.

## Exemplo:

- LJMP LABEL
- LJMP 0002h

# Instrução - SJMP

## Operação: SJMP

---

**Função:** Short Jump

**Sintaxe:** SJMP *valor*

**Descrição :** O salto curto, SJMP (short jump), toma como base a posição atual da instrução para desviar o fluxo de execução do programa e, por isso, é denominado de salto relativo. Este *valor* precisa estar entre -128 ou +127 bytes de distância da instrução que segue o SJMP.

## Exemplo:

- SJMP LABEL
- SJMP -8

# Instrução - JMP

## Operação: JMP

---

**Função:** Desvia o programa para DPTR+A

**Sintaxe:** JMP @A+DPTR

**Descrição :** JMP desvia incondicionalmente para o endereço representado pela soma de DPTR e do valor do acumulador.

## Exemplo:

- JMP LABEL
- JMP @A+DPTR

# Instruções LJMP, AJMP, SJMP e JMP

---

		Bytes	MC	Op1	Op2	Op3
LJMP	end16	3	2	02	MSB(end16)	LSB(end16)
AJMP	end11	2	2	[(MSB(end11))<<5]OU1	LSB(end11)	-
SJMP	rel	2	2	80	rel	-
JMP	@A+DPTR	1	2	73	-	-

Instruções de saltos incondicionais.



# Portas Digitais do 8051:

P1, P2, P3 e P4

---

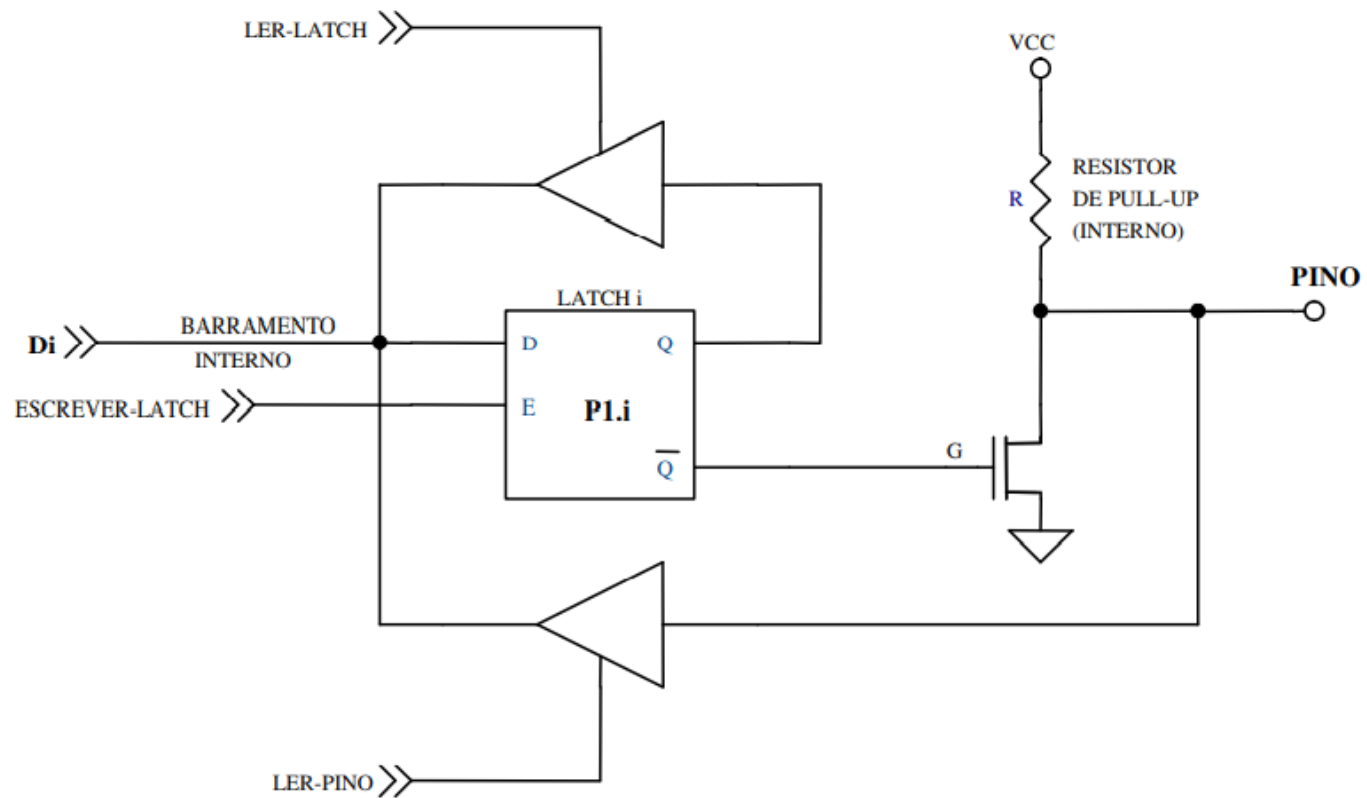
# Portas

---

- P0 → barramento de dados (D0, ...,D7) e parte baixa do barramento de endereços (A0, ...,A7);
- P1 → porta paralela livre.
- P2 → parte alta do barramento de endereços (A8, ...,A15);
- P3 → Sinais de controle e de comunicação.

# Portas

- P1 → porta paralela livre.



# Portas

---

Pino P3	Função
P3.0	RX
P3.1	TX
P3.2	/INT0
P3.3	/INT1
P3.4	T0
P3.5	T1
P3.6	/WR
P3.7	/RD

# EdiSim51 - Portas

A tela preta ao lado apresenta todos os dispositivos ligados aos 32 pinos das 4 portas.

EdSim51DI - Version 2.1.20 | aula03-ex5.asm

System Clock (MHz) 12.0 1000 Update Freq.

SBUF

R/O W/O TH0 TL0 R7 0x00 B 0x00

0x00 0x00 0x00 0x00 R6 0x00 ACC 0x00

RXD TXD TMOD 0x00 R5 0x00 PSW 0x00

1 1 TCON 0x00 R4 0x00 IP 0x00

SCON 0x00 R3 0x00 IE 0x00

R2 0xFD PCON 0x00

pins bits TH1 TL1 R1 0xFE DPH 0x00

0xFF 0xFF P3 0x00 0x00 R0 0x00 DPL 0x00

0xFF 0xFF P2 PC 8051 SP 0x07

0xFF 0xFF P1 0x0000 PSW 0 0 0 0 0 0 0 0

0xFF 0xFF P0 Modify RAM

Data Memory addr 0x00 0x00 value

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	00	00	FE	FD	00	00	00	00	00	00	00	00	00	00	00	00
10	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
20	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
30	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
40	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
50	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
60	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
70	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Copyright ©2005-2016 James Rogers Remove All Breakpoints

RST Assm Run New Load Save Copy Paste

Reset: PC = 0x0000

P0.7 1 Display-select Decoder CS|DAC WR

P0.6 1 Keypad Column 2

P0.5 1 Keypad Column 1

P0.4 1 Keypad Column 0

P0.3 1 Keypad Row 3

P0.2 1 Keypad Row 2

P0.1 1 Keypad Row 1

P0.0 1 Keypad Row 0

P1.7 1 LED 7|Seg. dp|DAC DB7|LCD DB7

P1.6 1 LED 6|Seg. g|DAC DB6|LCD DB6

P1.5 1 LED 5|Seg. f|DAC DB5|LCD DB5

P1.4 1 LED 4|Seg. e|DAC DB4|LCD DB4

P1.3 1 LED 3|... d|...DB3|...DB3|... RS

P1.2 1 LED 2|... c|...DB2|...DB2|LCD E

P1.1 1 LED 1|Seg. b|DAC DB1|LCD DB1

P1.0 1 LED 0|Seg. a|DAC DB0|LCD DB0

P2.7 1 SW 7|ADC DB7

P2.6 1 SW 6|ADC DB6

P2.5 1 SW 5|ADC DB5

P2.4 1 SW 4|ADC DB4

P2.3 1 SW 3|ADC DB3

P2.2 1 SW 2|ADC DB2

P2.1 1 ADC DB1

P2.0 1 ADC DB0

P3.7 1 ADC RD|Comparator Output

P3.6 1 ADC WR

P3.5 1 Motor Sensor

P3.4 1 Display-select Input 1

P3.3 1 AND G..put|SW 1|Displ..t 0

P3.2 1 SW 0|ADC INTR

P3.1 1 Motor Control Bit 1|Ext. UART Rx

P3.0 1 Motor Control Bit 0|Ext. UART Tx

DI LD 1 2 3 AND Gate Disabled U No Parity 8-bit UART @ 4800 Baud 0.0 V MAX

# Exercícios

---

# Exercício 1

## Exercício 1:

---

Crie um programa que inicie o port P1 com valor Feh, e contenha uma rotina de repetição que fique rotacionando a direita o valor do port P1 (neste exercício use a instrução **ajmp** com um **LABEL**).

# Resposta do exercício 1

## Exercício 1:

---

Crie um programa que inicie o port P1 com valor Feh, e contenha uma rotina de repetição que fique rotacionando a direita o valor do port P1 (neste exercício use a instrução **ajmp** com um **LABEL**).

```
mov A, #0FEh
```

**ROT:**

```
rr      A  
mov     P1, A  
ajmp    ROT
```



# Exercício 2

## Exercício 2:

---

Crie um programa que inicie o port P1 com valor Feh, e contenha uma rotina de repetição que fique rotacionando a direita o valor do port P1 (neste exercício use a instrução **ajmp** com o endereço da memória do programa).

# Resposta do exercício 2

## Exercício 2:

---

Crie um programa que inicie o port P1 com valor 0FEh, e contenha uma rotina de repetição que fique rotacionando a direita o valor do port P1 (neste exercício use a instrução **ajmp** com o endereço da memória do programa).

```
0000| mov A, #0FEh  
      rot:  
0002| rr A  
0003| mov P1, A  
0005| ajmp 002h
```

# Exercício 3

## Exercício 3:

---

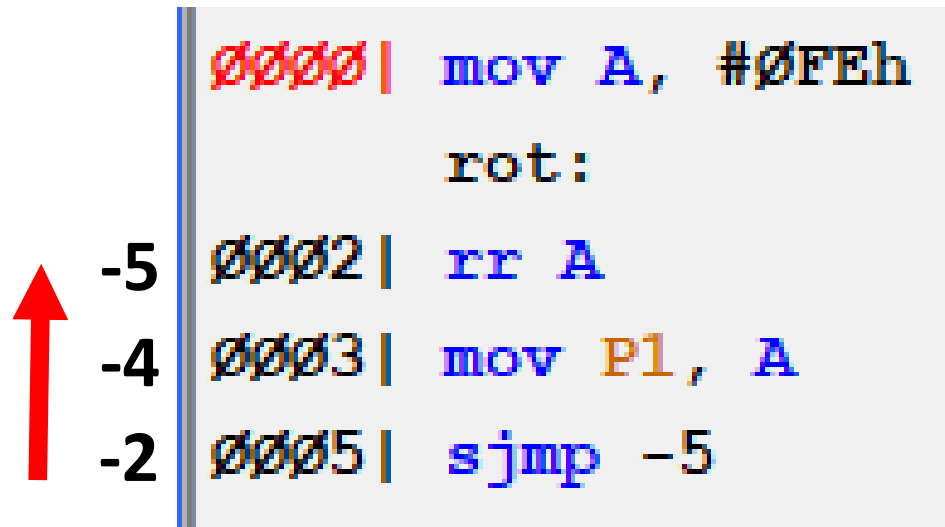
Crie um programa que inicie o port P1 com valor Feh, e contenha uma rotina de repetição que fique rotacionando a direita o valor do port P1 (neste exercício use a instrução **sjmp** com o valor necessário para voltar, ou seja, sem LABEL).

# Resposta do exercício 3

## Exercício 3:

---

Para a instrução **SJMP \$** desviar para si mesma, é necessário retroceder duas posições, ou seja **-2**, mais as posições necessárias para retroceder.



The diagram illustrates the assembly code for a self-jump instruction. It shows a sequence of instructions with their relative offsets and addresses. A red arrow points to the **SJMP -5** instruction, which is at address **-2**. The instruction **SJMP -5** is the instruction that jumps back to the instruction at address **-5**, which is **RR A**.

Offset	Address	Instruction
	<b>0000</b>	<b>mov A, #0FEh</b>
		<b>rot:</b>
<b>-5</b>	<b>0002</b>	<b>rr A</b>
<b>-4</b>	<b>0003</b>	<b>mov P1, A</b>
<b>-2</b>	<b>0005</b>	<b>sjmp -5</b>

# Exercício 4

Dado o programa abaixo, qual será o valor de R2 após a execução.

---

```
0000| MOV R1, #0FFh
0002| DEC R1
0003| SJMP TESTE
      EX01:
0005| DEC R1
0006| MOV A, R1
0007| DEC A
0008| SJMP FIM
      TESTE:
000A| INC R1
000B| SJMP EX01
000D| DEC R1
000E| MOV A, R1
      FIM:
000F| MOV R2, A
```

# Bibliografia

---

ZELENOVSKY, R.; MENDONÇA, A. Microcontroladores Programação e Projeto com a Família 8051. MZ Editora, RJ, 2005.

Gimenez, Salvador P. Microcontroladores 8051 - Teoria e Prática, Editora Érica, 2010.