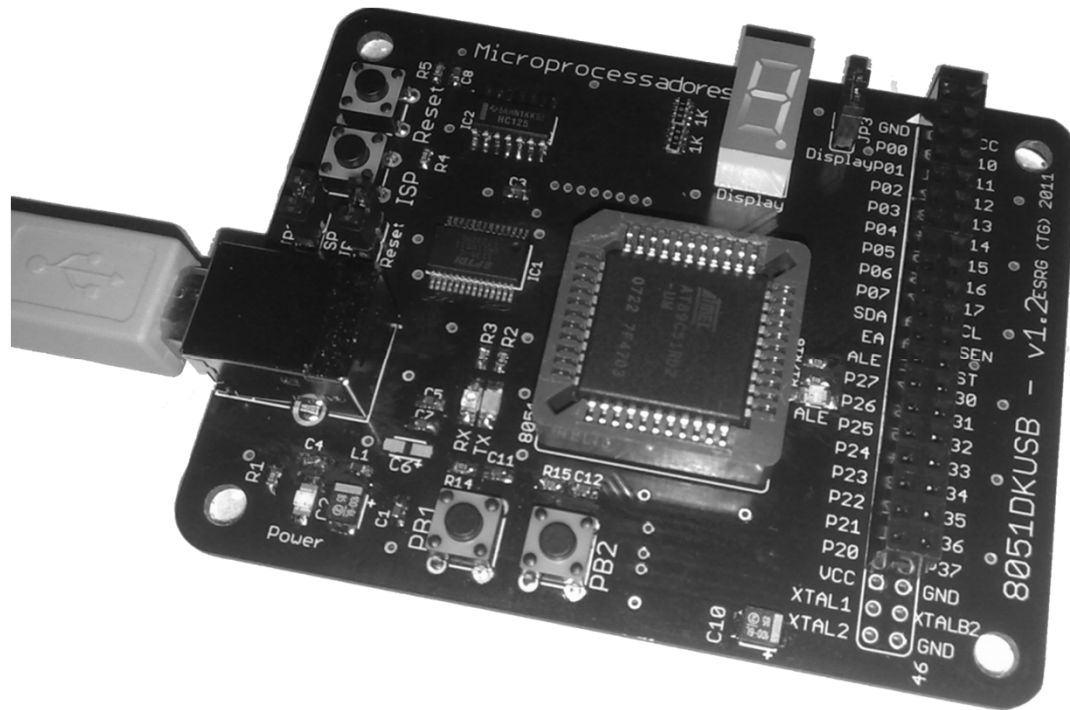
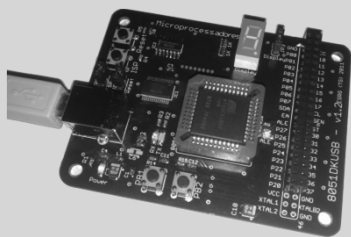


Mestrado Integrado em Eng. Electrónica Industrial e Computadores



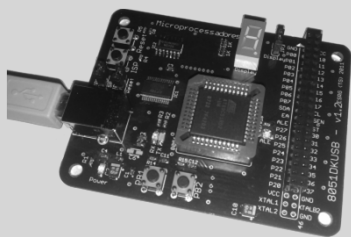
Instruction Set

Microcontroladores
2º Ano – A07



Modos de endereçamento

- Especificam o modo como os operandos de uma instrução são especificados e acedidos.
- O 8051 possui oito modos de endereçamento:
 - Endereçamento por Registo
 - Endereçamento Directo
 - Endereçamento Indirecto
 - Endereçamento Imediato
 - Endereçamento Relativo
 - Endereçamento Absoluto
 - Endereçamento Longo
 - Endereçamento Indexado
- Uma instrução pode envolver um ou mais modos de endereçamento.



Programa em assembly

1. Um programa *assembly* contém os seguintes elementos:

1. Instruções máquina
2. Directivas para o *assembler*
3. Controlos do *assembler*
4. Comentários

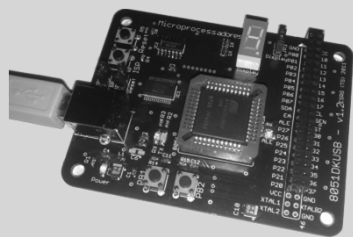
2. O formato genérico para cada linha é o seguinte:

[símbolo] mnemónica [operando] [,operando] [...] [;comentário]

1. A mnemónica pode ser uma instrução ou uma directiva para o *assembler*
2. O operando contém o endereço ou dado usado pela instrução
 - Registos, constantes, endereços de memória, apontadores;
3. O primeiro carácter de um símbolo deve ser sempre uma letra, '?', ou '_' que será seguida por letras, dígitos, '?', '_'.
Ex: `label`, `12345`, `_12345`, `?12345`
4. Uma etiqueta (*label*) é um símbolo que termina obrigatoriamente com o carácter ':'.
Ex: `label:`

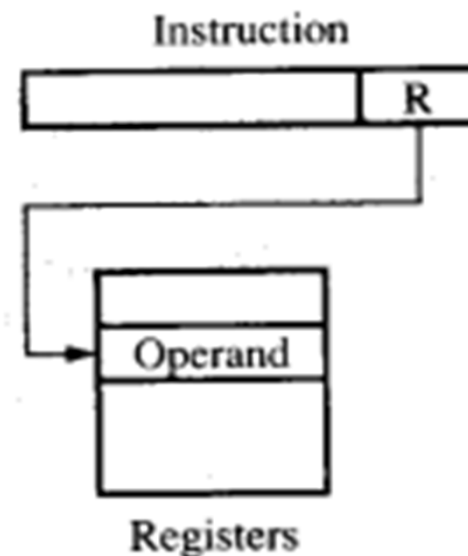
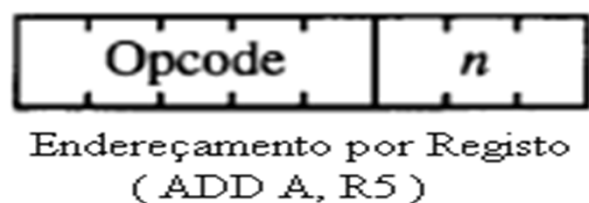
3. Qualificadores dos operandos:

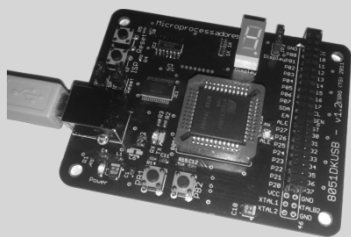
1. Constante precedida de '#'
2. Número sem qualquer prefixo é um endereço de memória
3. Apontador é precedido de '@'



Modos de endereçamento

- **Endereçamento por registo:**
 - Um dos operandos é um registo (R0-R7).
 - A informação do registo é codificada no *opcode* da instrução:
 - Os 3 *bits* menos significativos do *opcode* das instruções são usados para especificar um dos 8 registo R0 – R7 do banco actual;

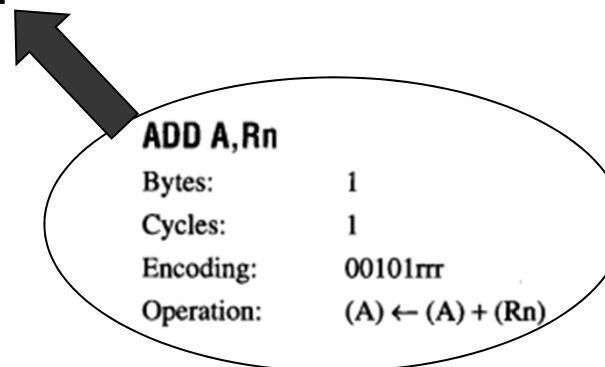




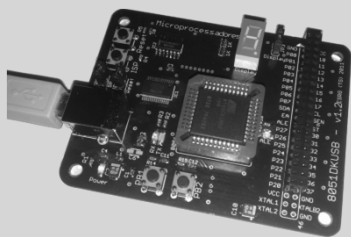
Modos de endereçamento

- Por exemplo para **ADD A, R7** :

Opcode: 00101 111 B



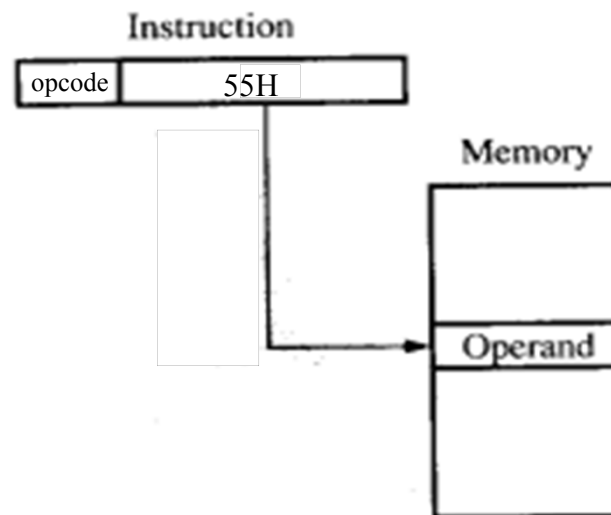
O *opcode* 2Fh será armazenado no endereço da memória de código (CODE) onde se encontra a instrução.



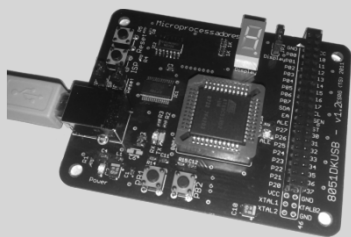
Modos de endereçamento

- **Endereçamento Directo**

- Pode aceder qualquer variável ou registo da RAM interna.
- A instrução tem 2-3 bytes de comprimento, tendo um dos bytes a localização a ser usado.



- São armazenados na CODE:
 - 25h e 55h



Modos de endereçamento

- **Endereçamento Imediato**

- O operando está armazenado na posição seguinte à do *opcode*.
- Apenas para constantes – 2 bytes:
 - Constante numérica
 - Variável simbólica
 - Expressão aritmética contendo apenas constantes, símbolos e operadores
- # é o símbolo indicador do operando imediato.
- A constante pode ter um comprimento de 8 ou 16 *bits*, dependendo do tipo de instrução.
- Exemplos:

Constante de 16 bits:

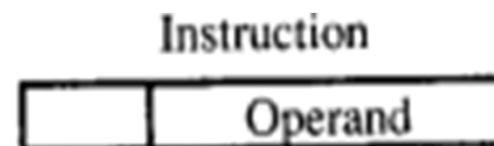
MOV DPTR, #800H

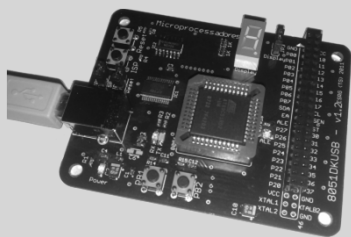
Constante de 8 bits:

MOV A, #10H



Endereçamento Imediato (ADD A, #44H)



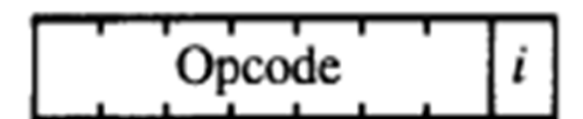


Modos de endereçamento

- **Endereçamento Indirecto**

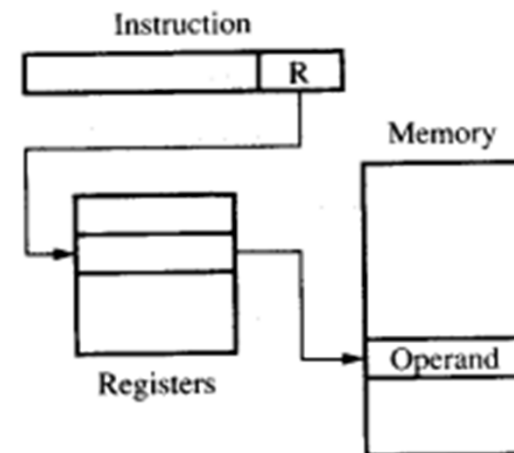
(apontador)

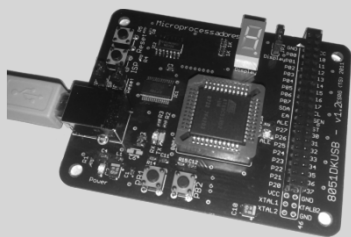
- Usado na manipulação sequencial de localizações de memória, entradas indexadas numa tabela na RAM e cadeias de caracter.
 - Endereços dos operandos conhecidos apenas no instante da execução.
- Normalmente, os registos R0 e R1 são usados como apontadores, exceptuando as instruções MOVX onde também pode ser usado o DPTR.



Endereçamento Indirecto
(ADD A, @R0)

→
Símbolo indicador do endereçamento indirecto





Modos de endereçamento

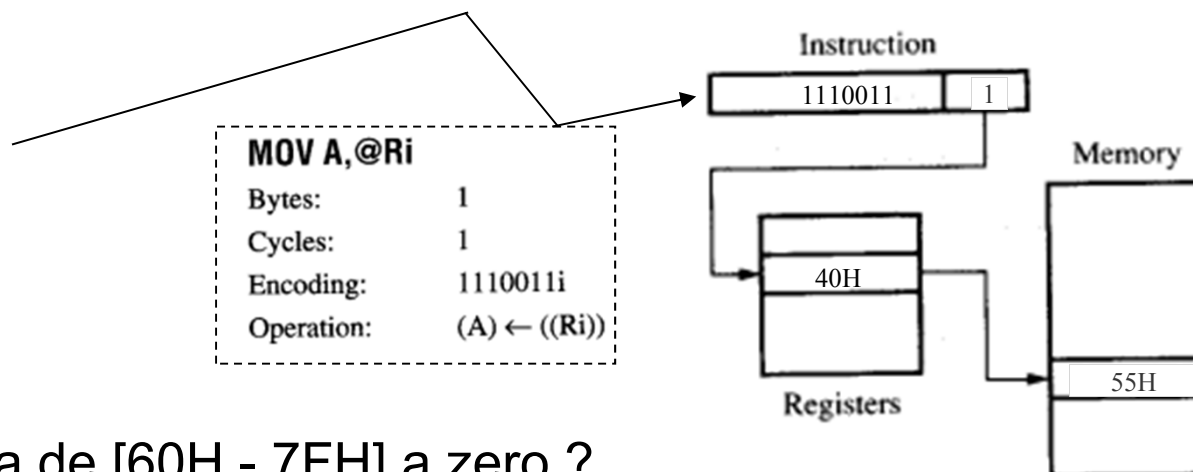
MOV A, @R1

R1 = 40H

(40H) = 55H

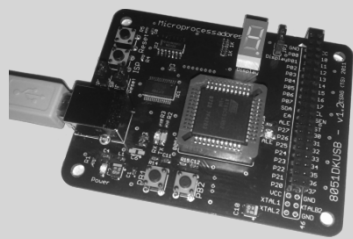
A = ?

Opcode = ?



Inicializar a RAM interna de [60H - 7FH] a zero ?

	MOV R0, #60H	; carrega R0 com o endereço da 1ª posição
CICLO:	MOV @R0, #0	; escreve zero na posição actual
	INC R0	; salta para a próxima posição
	CJNE R0, #80H, CICLO	; se não atingiu a última posição (7FH) continua
	...	; caso contrário, acabou e abandona o ciclo

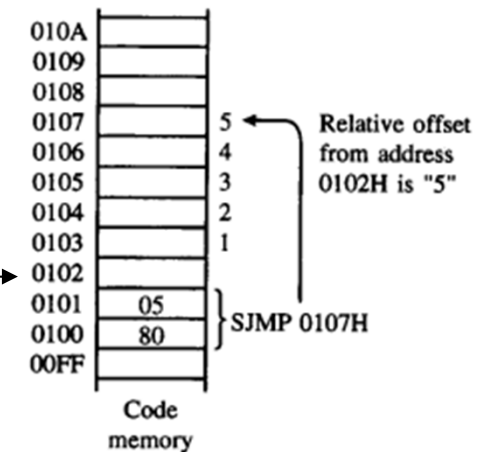


Modos de endereçamento

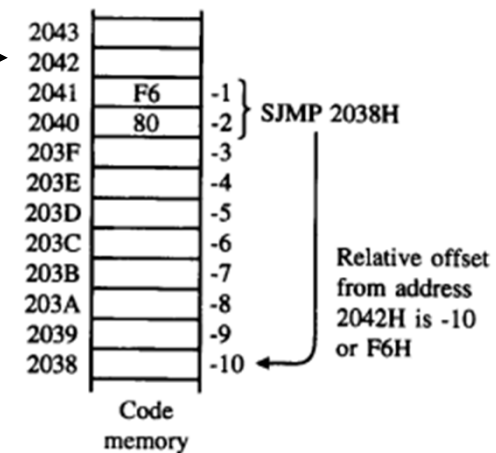
• Endereçamento Relativo



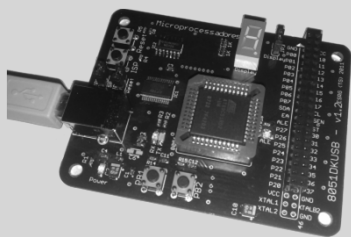
- Usado apenas com algumas instruções de salto;
- O *offset* é um valor com sinal de 8 bits: [-128 a 127] .
- O *offset* é somado ao $(PC) + 2$ para formar o endereço da próxima instrução a ser executada.



Salto para frente (no máximo 127 bytes)



Salto para trás (no máximo 128 bytes)



Modos de endereçamento

- **Endereçamento Absoluto**



Endereçamento Absoluto (AJMP, BACK)

- Usado apenas com as instruções ACALL e AJMP
- Os 5 MSB do PC especificam uma das 32 páginas de código
- 11 bits de endereço destino formado pelos 3 MSBits do código da operação (A10 –A8) mais o segundo byte da instrução (A7 – A0)
 - Permite saltos dentro da página corrente de 2K de memória de código

AJMP addr11

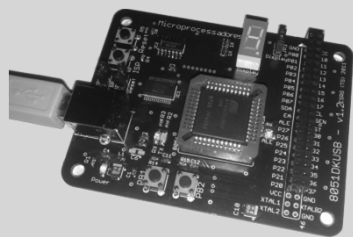
Bytes: 2

Cycles: 2

Encoding: aaa00001 aaaaaaaa

Note: aaa = A10–A8 and aaaaaaaa = A7–A0 of the destination address.

Operation: $(PC) \leftarrow (PC) + 2$
 $(PC_{10}-PC_0) \leftarrow \text{page address}$

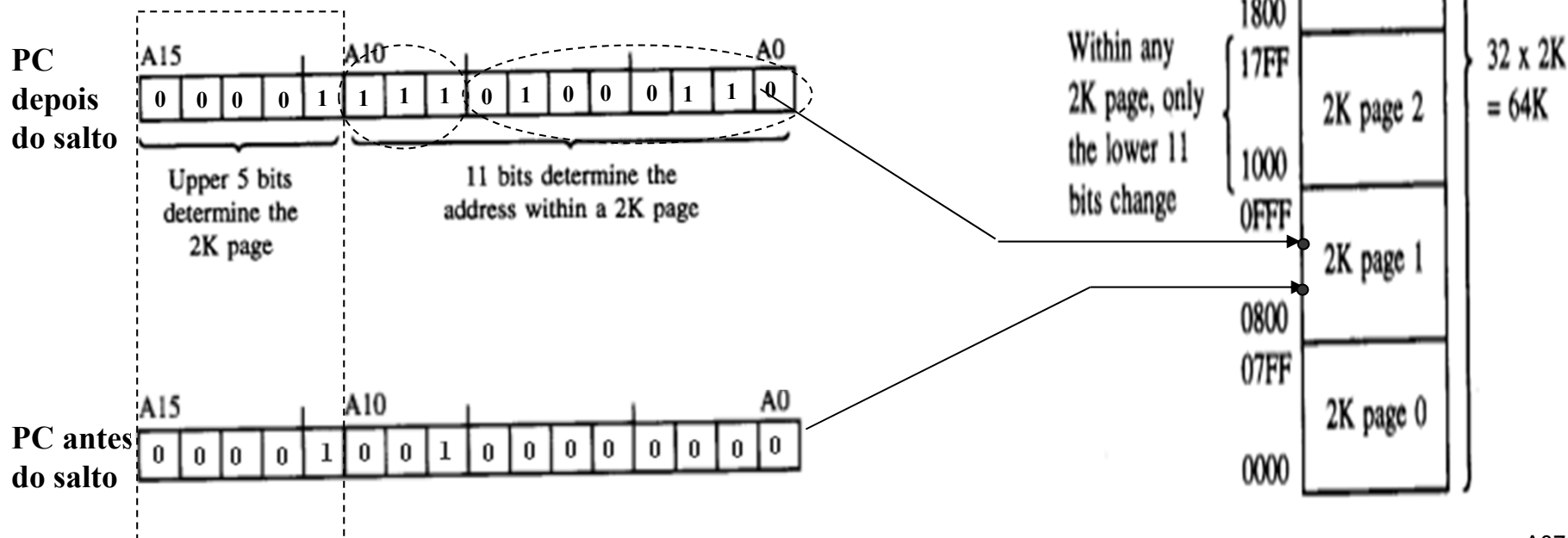


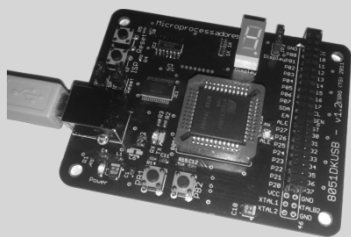
Modos de endereçamento

- Endereçamento Absoluto

AJMP 0f46H

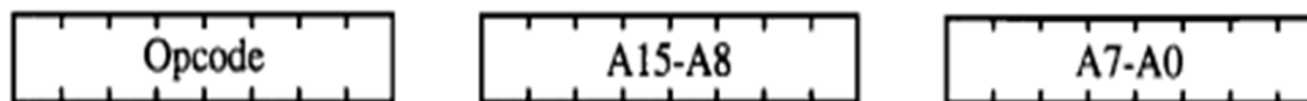
PC = 0900H





Modos de endereçamento

- Endereçamento Longo



Endereçamento Longo (LJMP, FAR_AHEAD)

- Usado apenas com as instruções LCALL e LJMP
- O endereço destino é de 16 *bits*, permitindo endereçar todo espaço da memória de código (64K)
- Instruções de 3 *bytes* de comprimento e dependentes da posição

Pg. 268: errado

LJMP addr16

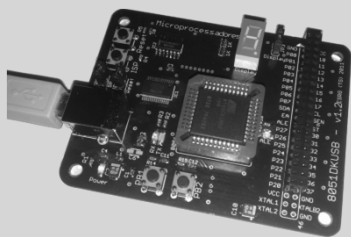
Bytes: 3

Cycles: 2

Encoding: 00000010 aaaaaaaaa aaaaaaaaa

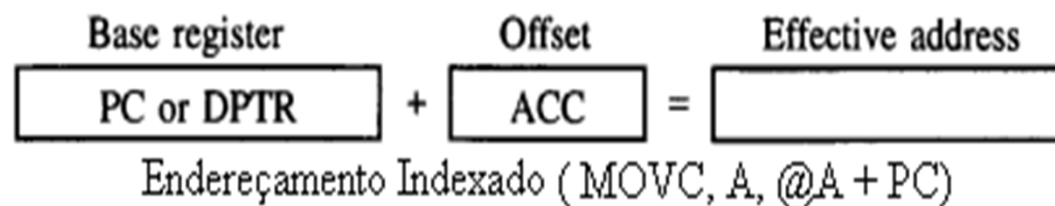
Note: Byte 2 contains address bits 15–8, byte 3 contains address bits 7–0.

Operation: (PC) ← addr15–addr0



Modos de endereçamento

- Endereçamento Indexado – apenas CODE ou XCODE



- Usa o PC ou o DPTR como registo base e o acumulador como offset.
- Ideal na implementação de tabelas de salto e de pesquisa.

MOV C, @A+PC

Bytes: 1
 Cycles: 2
 Encoding: 10000011
 Operation: (PC) ← (PC) + 1
 (A) ← ((A) + (PC))

REL_PC: INC A

MOV C, @A + PC

RET

DB 66H

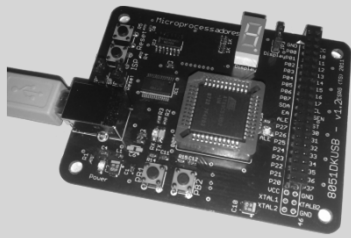
DB 77H

DB 88H

DB 99H

PC →

...
99H
88H
77H
66H
22H
83H
04H
..



Modos de endereçamento

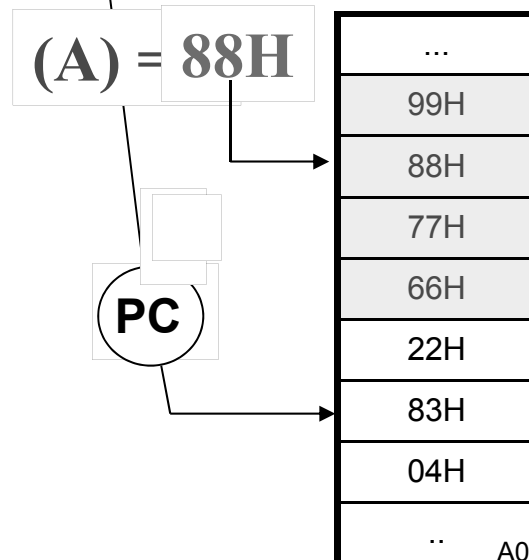
MOVC A, @A+PC

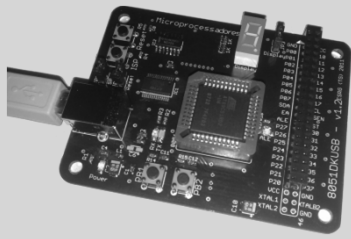
Bytes: 1
Cycles: 2
Encoding: 10000011
Operation: $(PC) \leftarrow (PC) + 1$
 $(A) \leftarrow ((A) + (PC))$

Que alterações efectuará ao programa com a inserção da instrução **MOV DPTR, #1234H** entre as instruções **MOVC A, @A+PC** e **RET**?

REL_PC: **INC A**
MOVC A, @A+PC
RET
DB 66H
DB 77H
DB 88H
DB 99H
...
MOV A, #02H
LCALL REL_PC
...

Toda a subrotina tem um nome dado por uma etiqueta e termina com a execução da instrução **RET**





Modos de endereçamento

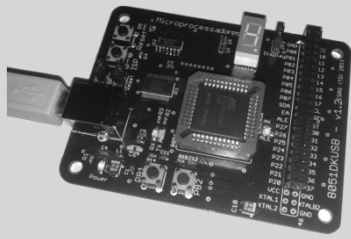
```
REL_PC: ADD A, #4
        MOVC A, @A + PC
        MOV DPTR, #1234H
        RET
        DB      66H
        DB      77H
        DB      88H
        DB      99H
        ...
        MOV A, #02H
        LCALL REL_PC
        ...
```

Deve-se somar ao acumulador o número de *bytes* que separa a instrução `MOVC A, @A + PC` do início da tabela

(A) = 88

PC

...
99H
88H
77H
66H
22H
12
34
90H
83H
04H
24H
..



Modos de endereçamento

JMP @A + DPTR

Bytes: 1

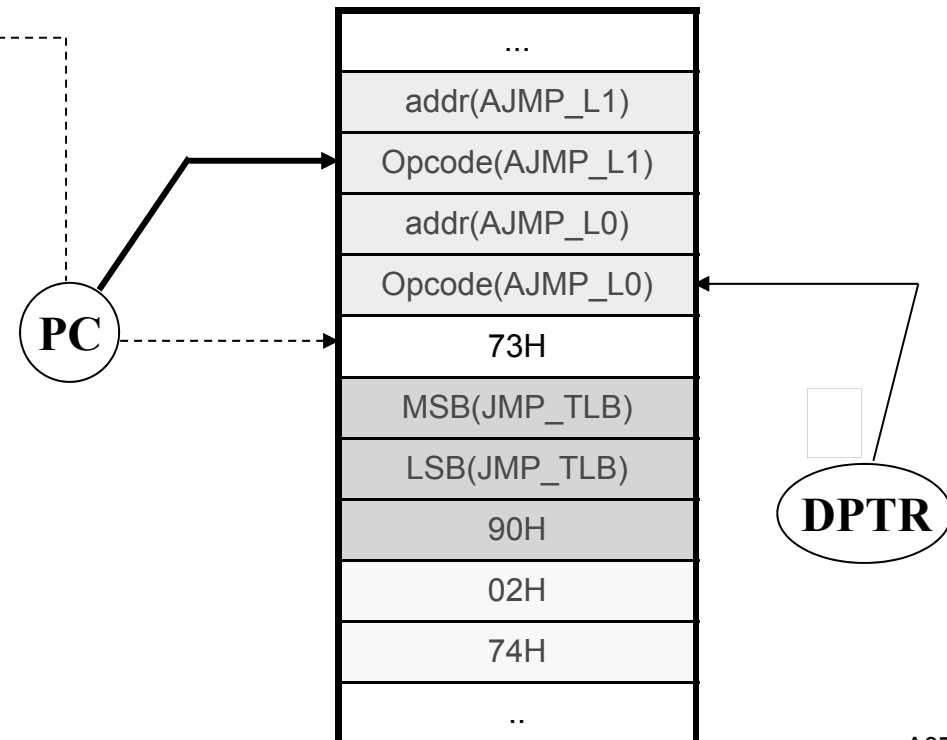
Cycles: 2

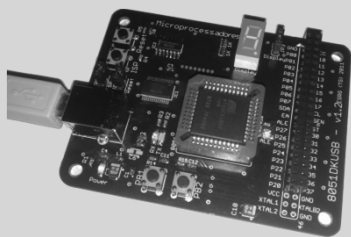
Encoding: 01110011

Operation: $(PC) \leftarrow (A) + (DPTR)$

```

MOV A, #02H
MOV DPTR, #JMP_TBL
JMP @A + DPTR
JMP_TBL: AJMP LABEL0
          AJMP LABEL1
          AJMP LABEL2
          AJMP LABEL3
          ...
LABEL0:   ...
LABEL1:   ...
LABEL2:   ...
LABEL3:   ...
  
```



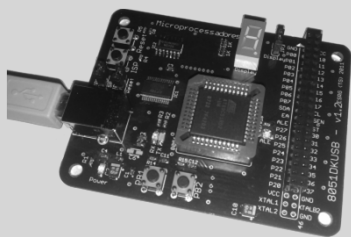


Modo específico

- Para instruções específicas que envolvem determinados operandos;
- Codificação num único byte;
- O *opcode* identifica implicitamente o operando.

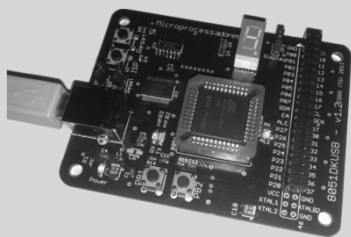
- Exemplos:

Mnemónica	Opcode
DIV AB	84h
SWAP A	C4h
DA A	D4h



Linguagem *assembly*

- Nenhum programador utiliza a linguagem máquina;
- Utilizam as mnemónicas da linguagem assembler;
- Um “assemblador” converte essas mnemónicas em código máquina;
- A linguagem assembler é muito próxima da linguagem máquina;
- Cada instrução é convertida directamente em bytes de código máquina (*opcode*);
- Produz código mais eficiente mas é mais trabalhoso para o programador.



Linguagem *Assembly*

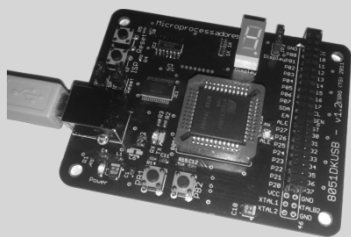
i. *Apresenta um nível de abstracção intermédia entre os dois extremos: linguagem máquina e linguagem alto-nível*

ii. *Facilita a programação pela substituição do código binário da linguagem máquina com símbolos*

Escrita usando labels (etiquetas), mnemónicas, etc.

iii. *Um programa em assembly não é executável*

Todos os símbolos tais como mnemónicas, etiquetas devem ser traduzidos para código binário



Assembly->Código Máquina

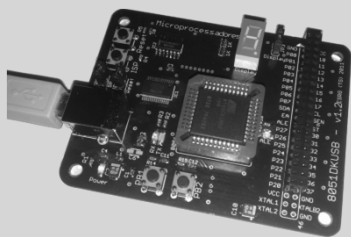
Dependendo da complexidade do ambiente de programação pode envolver várias etapas até a produção do código executável

i. Assembler

- a. Traduz um programa em assembly para programa em linguagem máquina (código objecto)*
- b. O código objecto pode estar na forma absoluta ou forma relocatable*

ii. Linker

- a. Combina vários programas objectos na forma relocatable, produzindo um programa executável através da atribuição de endereços absolutos*
- b. Produz também um ficheiro contendo o mapa de memória e tabela de símbolos*

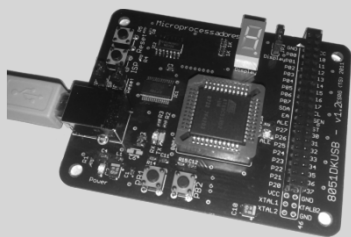


Contador de Localização

- O assembler possui um contador de localização para cada um dos cinco segmentos

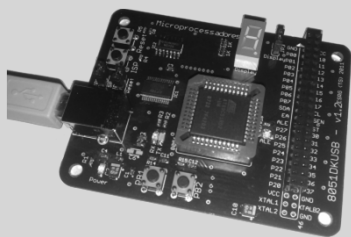
1. CODE	(0000h - FFFFH)	Código
2. DATA	(00H – FFH)/(00H – 7FH)	Interna de Dados
3. IDATA	(00H – FFH)/(00H – 7FH)	Interna de Dados Indirecta
4. BIT	(00H – FFH)/(20H – 2FH)	Interna de Dados: Bit Space
5. XDATA	(0000H – FFFFH)	Externa de Dados
- Cada contador de localização é inicializado com o valor zero e pode posteriormente ser alterado usando as directivas para o assembler
- O símbolo \$ pode ser usado para especificar o valor do contador de localização do segmento activo:

```
TABLE:    DB    1,2,3,4,5,6,7,8,9
LEN       EQU   $-TABLE
```



Salto/Invocação genérico

- O ASM51 permite o uso das mnemónicas genéricas JMP ou CALL. Evitando em muitas situações o uso de SJMP, AJMP, LJMP ou ACALL, LCALL;
- A conversão da mnemónica genérica para instrução real segue a seguinte regra:
 1. JMP é substituído por SJMP se não for usada *forward reference* e o destino do salto estiver na gama de -128 localizações
 - » *forward reference*: uso de um símbolo antes da respectiva definição
 2. JMP/CALL é substituído por AJMP/ACALL se não for usada *forward reference* e a instrução que segue JMP/CALL pertencer à mesma página de 2K que a instrução destino
 3. Caso contrário, será usado a conversão para LJMP/LCALL

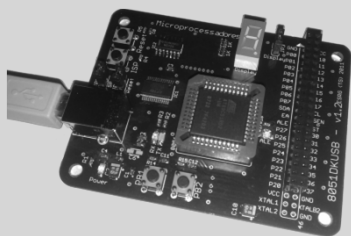


Endereçar ao bit

Muitas vezes é necessário realizarmos operações sobre bits.

A linguagem assembly permite o endereçamento ao bit utilizando uma das seguintes formas:

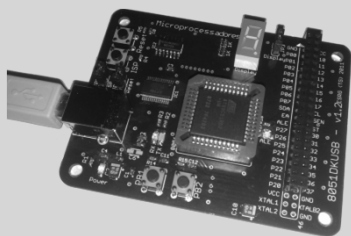
	ON	EQU	7
Explicitamente pelo endereço	SETB	0E7H	
	JNB	99H, \$	
Usando o operador .	SETB	ACC.7	
	SETB	224.ON	
	SETB	0E0H.7	
Usando um símbolo pré-definido	JNB	TI, \$	
	CLR	C	



Bases numéricas

Decimal	MOV	A, #15
	MOV	A, #15 D
Binário	MOV	#1111 B
Octal	MOV	A, #17 Q
	MOV	A, #17 O
Hexadecimal	MOV	A, #0F H
	MOV	A, #0A5 H
	MOV	A, #A5 H ?

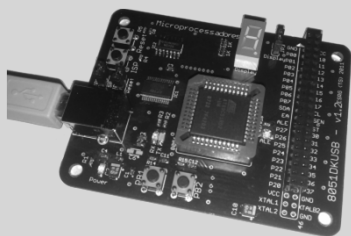
*Para diferenciar um
dado hexadecimal
imediato de um
símbolo*



Operadores de expressão

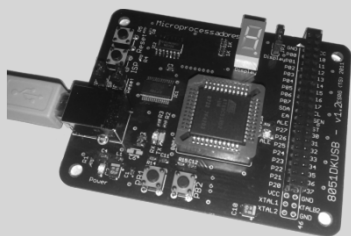
Símbolo do Operador	Operação
+, -,	Adição, subtração
/, *	Divisão, multiplicação
MOD	Resto da divisão
OR	Ou-lógico
AND	E-lógico
XOR	Ou-exclusivo
NOT	Complemento
SHR	Rodar à direita
SHL	Rodar à esquerda
HIGH	Obter o MSB
LOW	Obter o LSB
EQ, =	Igual a
NE, <>	Diferente
LT, <	Menor que
LE, <=	Menor ou igual
GT, >	Maior que
GE, >=	Maior ou igual

Expressão	Resultado
'A' - 'B'	0001H
HIGH(0AADDH)	0AAH
LOW(0AADDH)	0DDH
7 MOD 4	3
1000B SHR 2	0010B
NOT 1	0FFFEH
'A' SHL 8	4100H
5 EQ 8	0000H
'A' LT 'B'	0FFFFH
3 <= 3	0FFFFH
-1	0FFFFH
7 NE 4 ou 7 <> 4	0FFFFH
1101B XOR 0101B	1000B
HIGH('A' SHL 8)	0041H
HIGH 'A' SHL 8	0000H
'A' OR 'A' SHL 8	4141H
NOT 'A' - 1	0FFBFH



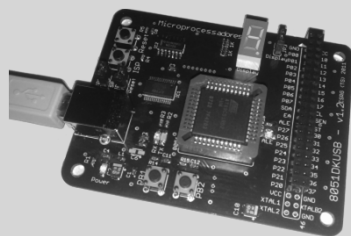
Directivas

Categoria	Directiva	Sintaxe			Função
Controlo do estado	ORG		ORG	expressão	Especifica um valor para contador de localização do segmento activo
	END		END		Indica ao <i>assembler</i> o fim do programa fonte
	USING		USING	expressão	Indica ao <i>assembler</i> o banco de registo usado no código que vem a seguir à directiva. Repare que a comutação do banco de registo deve ser efectuada usando apenas instruções do 8051
Definição de símbolos	SEGMENT	Símbolo	SEGMENT	tipo_de_segmento	Declara um símbolo como sendo um segmento <i>relocatable</i> de um dado tipo. Para começar a usar o segmento, deve-se usar a directiva RSEG
	EQU	Símbolo	EQU	expressão	Atribui um valor a um símbolo
	SET	Símbolo	SET	expressão	Igual ao EQU, exceptuando o facto de permitir a redefinição o símbolo
	DATA	Símbolo	DATA	expressão	Atribui ao símbolo um endereço directo da RAM interna
	IDATA	Símbolo	IDATA	expressão	Atribui um endereço indirectamente endereçável da RAM interna ao símbolo
	XDATA	Símbolo	XDATA	expressão	Atribui ao símbolo um endereço da memória externa
	BIT	Símbolo	BIT	expressão	Atribui um endereço directo da área de memória endereçável ao bit a um símbolo
	CODE	Símbolo	CODE	expressão	Atribui um endereço da memória de código ao símbolo

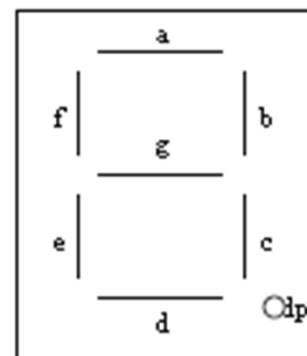
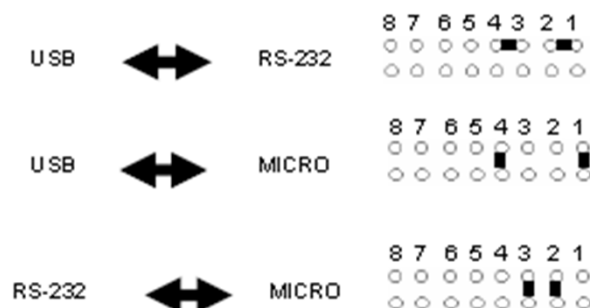
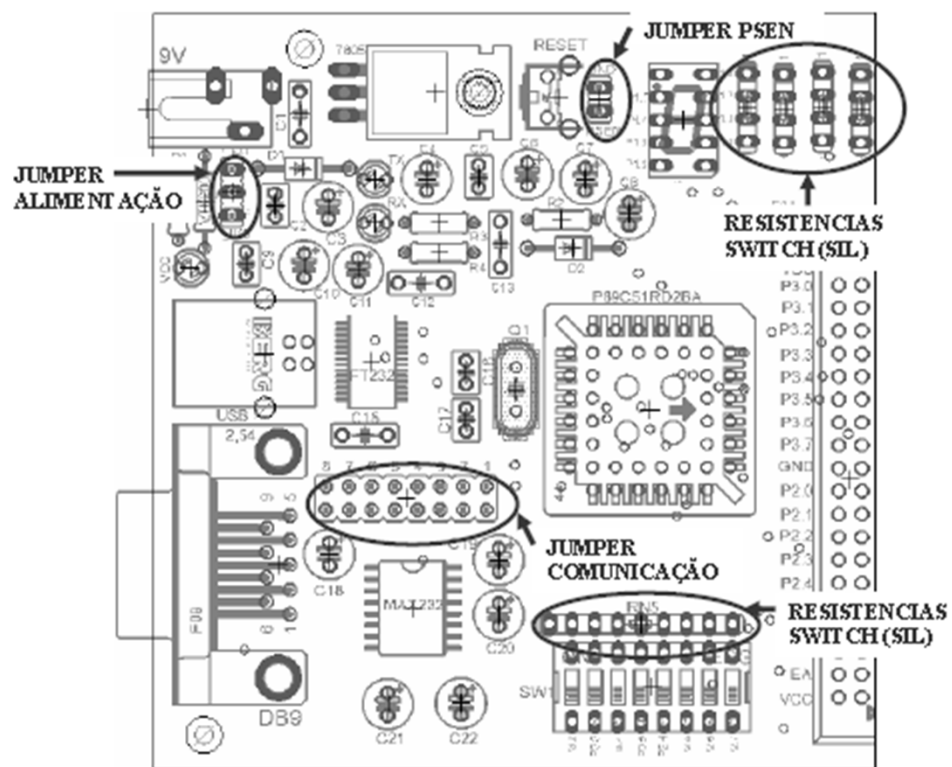


Directivas

Categoria	Directiva	Síntaxe	Função
Inicialização e reserva de armazenamento	DS	[LABEL:] DS expressão	Reserva espaços em múltiplos de <i>bytes</i> . Não pode ser utilizado com segmento do tipo BIT. O valor da expressão deve ser conhecida pelo <i>assembler</i>
	DBIT	[LABEL:] DBIT expressão	Reserva espaços em múltiplos de bits. O valor da expressão deve ser conhecida pelo <i>assembler</i>
	DB/DW	[LABEL:] DB/DW expressão	Inicializa a memória de código com valores do tipo byte/word
<i>Program linkage</i>	PUBLIC	PUBLIC Símbolo [, símbolo] [...]	Define uma lista de símbolos que tornam visíveis e utilizáveis a partir de outros módulos
	EXTRN	EXTRN Tipo_segmento(símbolo [,símbolo] [...], ...)	Informa o <i>assembler</i> da lista de símbolos definidos noutros módulos e que vão ser utilizados neste. O tipo de segmento pode ser CODE, DATA, XDATA, IDATA, BIT e um especial designado por NUMBER que especifica um símbolo definido por EQU
	NAME	NAME Nome_do_módulo	
Seleção de Segmentos	RSEG	RSEG Nome_do_segmento	Ao encontrar uma directiva de selecção de segmento, o <i>assembler</i> direcciona o código
	CSEG	CSEG [AT endereço]	ou dado que lhe segue para o segmento
	...	DSEG [AT endereço]	seleccionado até que seja seleccionado um
	XSEG	XSEG [AT endereço]	outro segmento



KIT8051- Esquema de Ligações



a	P1.3
b	P1.2
c	P1.0
d	P1.6
e	P1.5
f	P1.7
g	P1.4
dp	P1.1

Escrever 0 em P1.3 liga o segmento a.
Escrever 1 desliga