

Microcontroladores

Conjunto de Instruções do 8051

Prof. Guilherme Peron

Prof. Ronnier Rohrich

Prof. Rubão

Introdução

Os microcontroladores:

- Têm instruções limitadas
- Precisam tratar os dados da forma correta

RISC x CISC

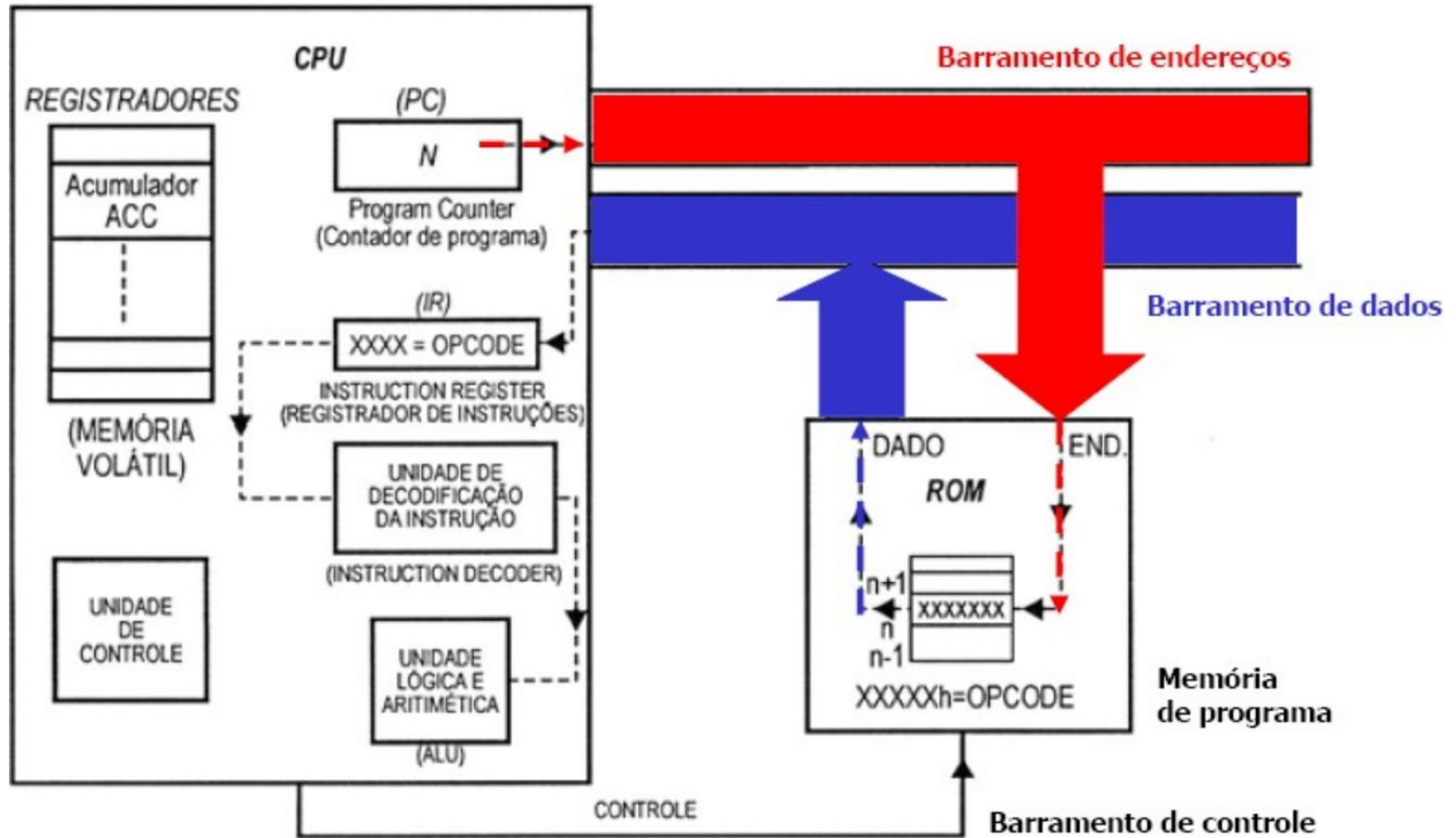
- RISC => Reduced Instruction Set Computer
 - Poucas instruções
 - Instruções ocupam um ciclo de clock
 - Instruções com o mesmo tamanho
 - PIC, ARM, PowerPC
 - Número de Instruções PIC: (~30)
- CISC => Complex Instruction Set Computer
 - Muitas instruções
 - Instruções podem ocupar mais de um ciclo de clock
 - Instruções de tamanho variado
 - PC (386, 486), **8051**
 - Número de Instruções 8051: (~255)

Linguagem de Máquina

- Um microcontrolador executa comandos específicos, que são constituídos de números binários.
- Estes comandos ou opcodes constituem a linguagem de máquina.
- Uma instrução do 8051 é constituída de um opcode e um ou mais operandos, neste caso o comprimento das instruções é variável.



Linguagem de Máquina

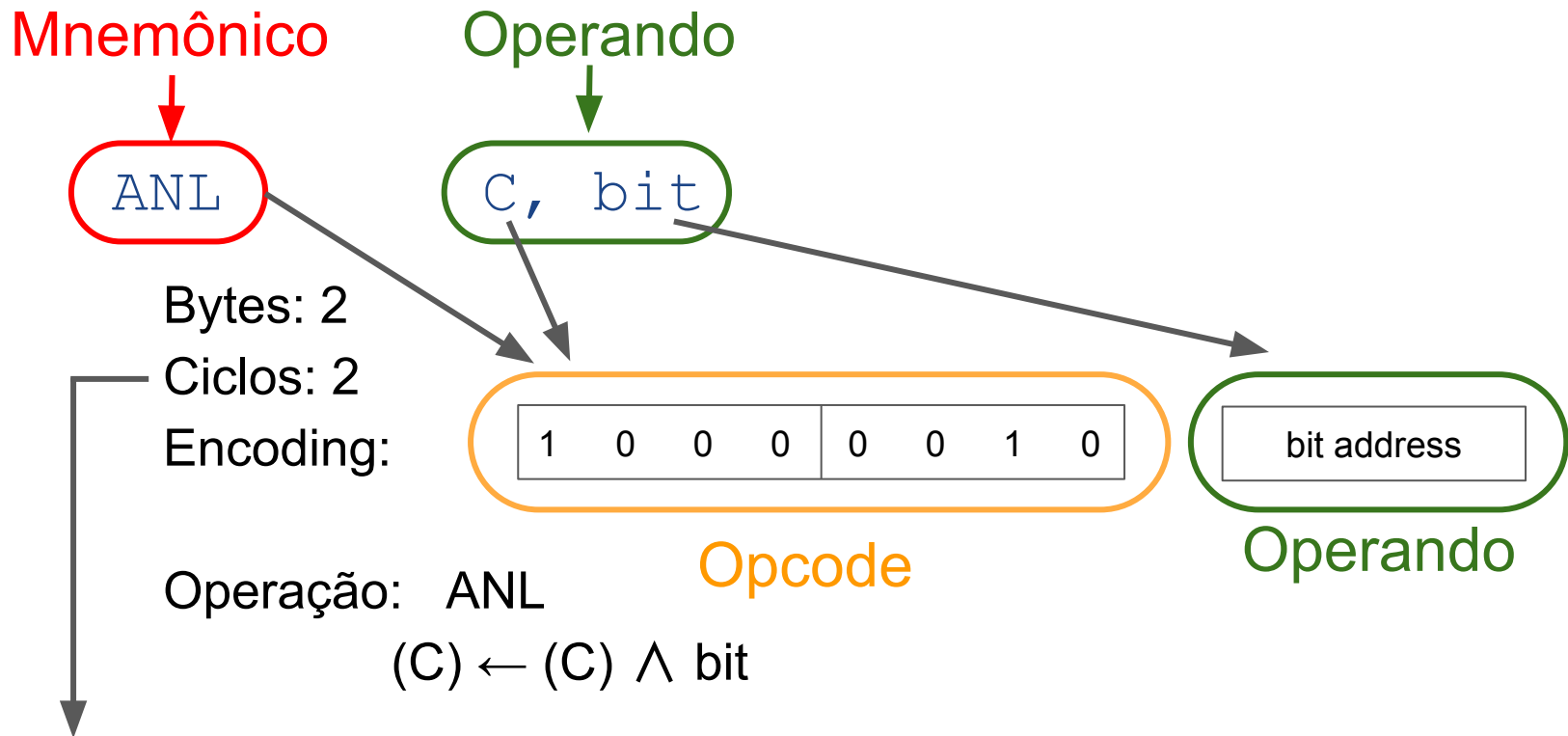


Linguagem de *Assembly*

- Para facilitar a vida do programador, criou-se a Linguagem *Assembly*, que possui o mesmo conjunto de instruções, porém utiliza símbolos (mnemônicos) no lugar dos números.
- A conversão da linguagem *assembly* para a linguagem de máquina é feita pelo *assembler* (montador). NUNCA CONFUNDIR!
- Entretanto ainda é específico para cada tipo de CPU, sendo considerada uma linguagem de baixo nível.

Linguagem de *Assembly*

- Formato de uma instrução assembly



No 8051, um ciclo de máquina é realizado em 12 períodos de clock. Se o clock for de 12 MHz, um ciclo de máquina tem 1us. Então, esta instrução é executada em 2us.

Linguagem de *Assembly*

Ah, professor! Vou ter que decorar todos estes tais de mnemônicos?



Linguagem de Alto Nível

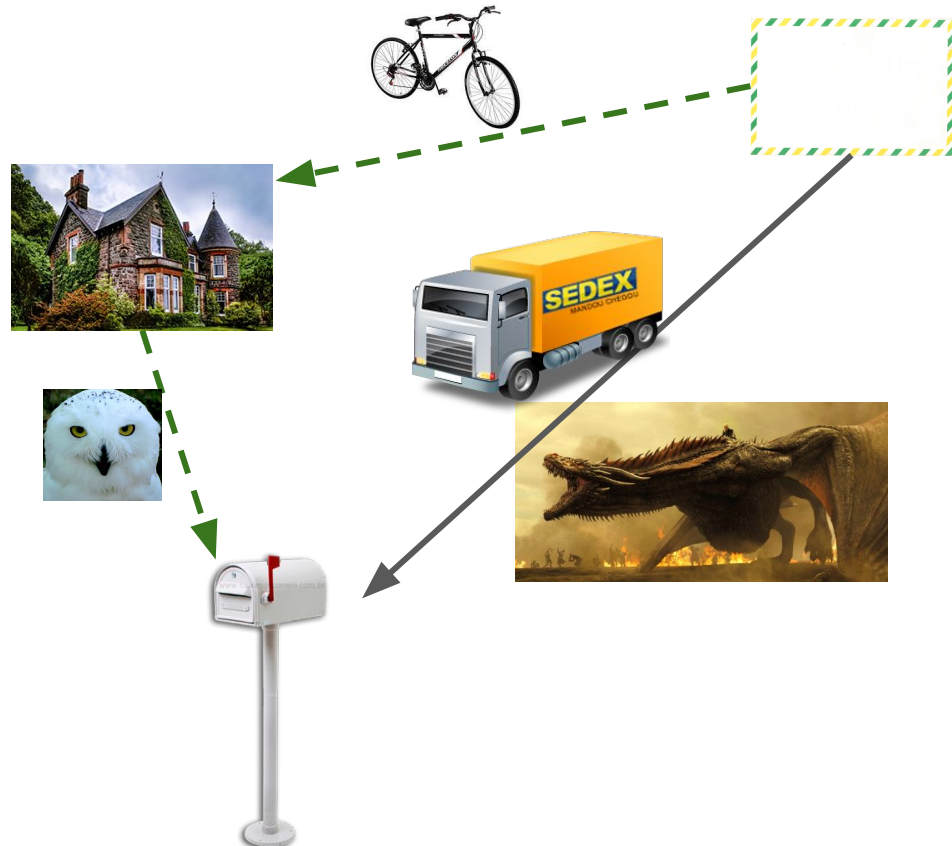
- Há algumas linguagens mais próximas à linguagem humana:
 - C, C++, Pascal, Java etc
- A conversão destas linguagens para a linguagem de máquina é feita pelo compilador / *cross* compilador

Modos de Endereçamento

Modos de Endereçamento

- As instruções operam com dados.
- Os dados podem ser acessados (endereçados) de diversos modos:

1. Registrador
2. Direto
3. Indireto
4. Imediato
5. Relativo
6. Absoluto
7. Longo
8. Indexado



Endereçamento por registrador

- O código da instrução contém um campo com três bits que identifica qual dos registradores R0 a R7 (Rn) deve ser usado (dentro do banco selecionado). É codificado apenas em um byte.

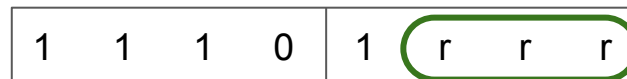
Ex: `MOV A, R3 ;move para o acumulador o
 ;conteúdo de R3`

`MOV A, Rn`

Bytes: 1

Ciclos: 1

Encoding:



Operação: `MOV`
 $(A) \leftarrow (Rn)$

Endereçamento por registrador

Mnemônico	Operação	<i>Flags</i>	Código	Bytes	Ciclos
INC Rn	$(Rn) \leftarrow (Rn) + 1$	nenhum	00001nnn	1	1
ADD A, Rn	$(A) \leftarrow (A) + (Rn)$	C,OV,AC	00111nnn	1	1

Endereçamento direto

- Permitem o acesso a qualquer posição de memória de dados interna ou SFR. O operando (endereço) é especificado por um campo de 8 bits na instrução.

Ex: `MOV A, 1Fh ;move o conteúdo do ender
 ;1Fh da mem int para o acc`

`ADD A, DPH ;A ← A+DPH`

Endereçamento direto

Ex: `MOV A, 1Fh ;move o conteúdo do ender
 ;1Fh da mem int para o acc`

MOV A, direto

Bytes: 2

Ciclos: 1

Encoding:

1	1	1	0	1	1	0	1
---	---	---	---	---	---	---	---

end. direto

Operação: **MOV**
(A) ← (direto)

Mnemônico	Operação	Flags	Código	Bytes	Ciclos
MOV A, P3	(A)←(P3)	nenhum	E5 B0	2	1
MOV R0, 0C1h	(R0)←(0C1h)	nenhum	A8 C1	2	2

Endereçamento indireto

- Na instrução está especificado um registrador cujo conteúdo é um endereço para o operando, como um **ponteiro**.
- Só podem atuar como ponteiros os registradores R0, R1 e SP (para end de 8 bits) e DPTR (para end. de 16 bits).
- O símbolo **@** indica “**o endereço apontado por**”.

Ex: MOV A, @R0h ;move para o acc o conteúdo do
 ;end de mem interna apontado
 ;por R0. (R0 guarda um end da
 ;memória interna)

 MOVX @DPTR,A ;move o acc para o end de mem
 ;externa apontado por DPTR

Endereçamento indireto

- Algumas famílias possuem acessos específicos da memória.
- Exemplo: No 80xx52, os 128 bytes superiores da memória RAM interna só podem ser acessados usando este modo de endereçamento.

Mnemônico	Operação	Flags	Código	Bytes	Ciclos
MOV A, @Ri i=0 ou 1	$(A) \leftarrow ((Ri)_{MDI})$	nenhum	1110011i	1	1
MOVX @R1, A	$((Ri))_{MDE} \leftarrow (A)$	nenhum	F3	1	2
MOV @R0, A	$((Ri))_{MDI} \leftarrow (A)$	nenhum	F6	1	1
MOVX @DPTR,A	$((DPTR))_{MDE} \leftarrow (A)$	nenhum	F0	1	2

Endereçamento imediato

- O valor do operando está contido no próprio código da instrução, como uma constante.
- Obs.: “Quando o número hexadecimal começar com A-F o número 0 (zero) deve ser adicionado antes do operando”.

Ex: ANL A, #0Fh ;AND lógico bit a bit entre
 ;o conteúdo do acc e a
ANL A, #data ;constante 0Fh

ANL A, #data

Bytes: 2

Ciclos: 1

Encoding:

0 1 0 1 | 0 1 0 0

dato immediato

Operação: ANL

$$(A) \leftarrow (A) \wedge \#data$$

Endereçamento imediato

- Não existe MOVX e MOVC com este modo de endereçamento.

Mnemônico	Operação	Flags	Código	Bytes	Ciclos
MOV A, #25	$(A) \leftarrow 25$	nenhum	74 19	2	1
MOV R1, #25h	$(R1) \leftarrow 25h$	nenhum	79 25	2	1
MOV DPTR, #1FFFh	$(DPTR) \leftarrow 1FFFh$	nenhum	90 1F FF	3	2

Endereçamento relativo

- Permite realizar um salto em relação à posição do PC.
- O operando é um **offset** (diferença) entre o endereço de destino e o endereço atual.
- O *offset* está em complemento de dois, com tamanho de 8 bits, ou seja, no intervalo de **+127** a **-128** bytes do PC.



Decimal	Complemento de dois
-128	10000000 (80h)
-127	10000001 (81h)
-126	10000010 (82h)
...	...
-2	11111110 (FEh)
-1	11111111 (FFh)
0	00000000 (00h)
1	00000001 (01h)
2	00000010 (02h)
...	...
126	01111110 (7Eh)
127	01111111 (7Fh)

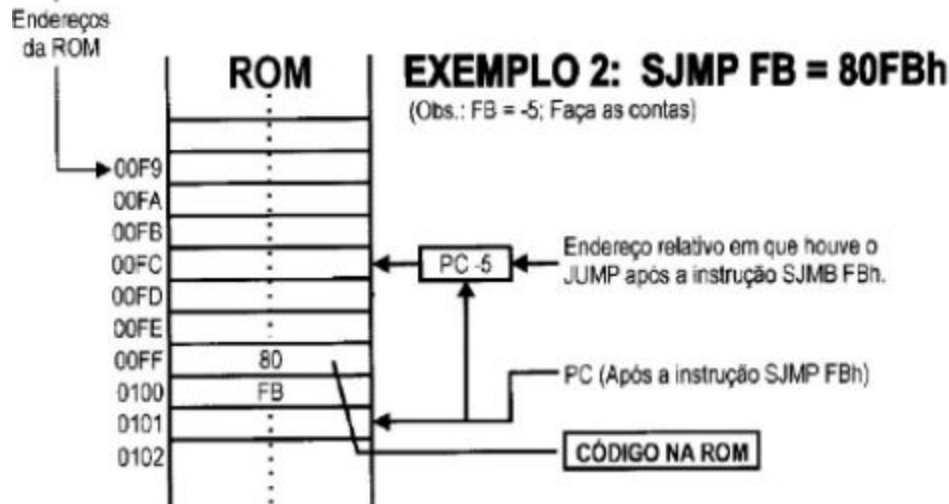
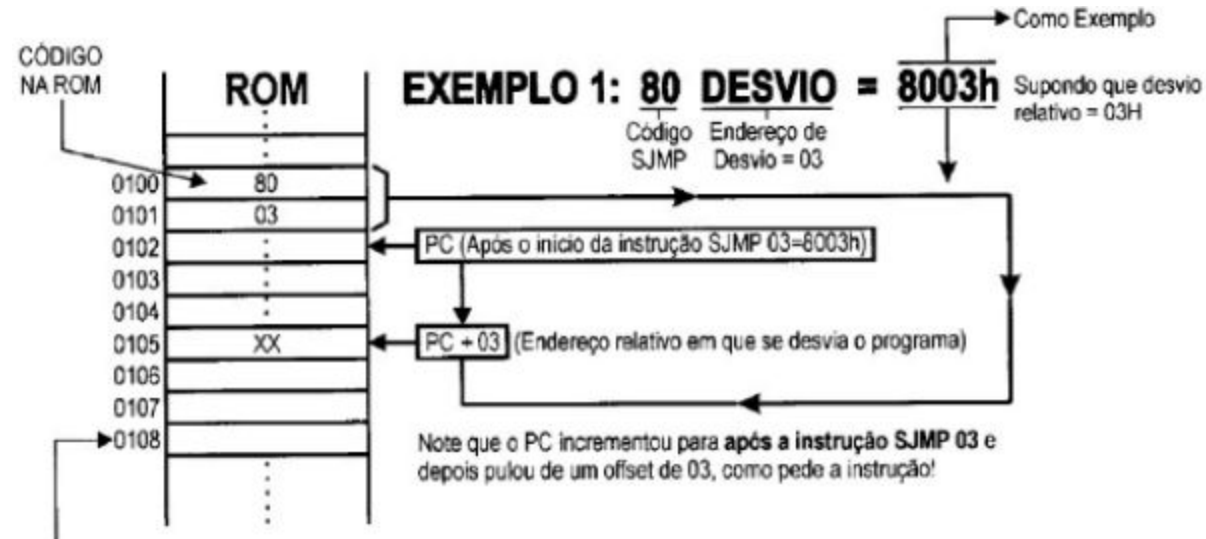
Endereçamento relativo

- Instruções que utilizam endereçamento relativo:
DJNZ, CJNE, JB, JNB, JBC, JC, JNC, SJMP, JZ, JNZ
- Permitem apenas saltos curtos (**-128** a **+127**)
- É **muito utilizado** e, se os limites forem extrapolados, o compilador acusa erro de sintaxe.

Mnemônico	Operação	Código	Bytes	Ciclos
SJMP 0FBh	$(PC) \leftarrow PC - 5$	80 FB	2	2
JB P3.1, 10	Se P3.1=1, $(PC) \leftarrow (PC) + 0Ah$	20 B1 10	3	2

Endereçamento relativo

Ex.



Endereçamento absoluto

- A memória ROM é dividida em até 32 páginas de 2kB
- Permite saltos **dentro da mesma** página
- Instruções **ACALL** e **AJMP** (2 bytes)
- Normalmente **NÃO** é utilizado.

Endereçamento longo

- Permite saltos **dentro de toda a faixa de endereçamento** possível
- Instruções **LCALL** e **LJMP** (3 bytes)
- **Muito utilizado.**



Mnemônico	Operação	Código	Bytes	Ciclos
LCALL 5F3Ch	$((SP)) \leftarrow PC_L$ $(SP) \leftarrow SP + 1$ $((SP)) \leftarrow PC_H$ $(SP) \leftarrow SP + 1$ $(PC) \leftarrow 5F3Ch$	12 3C 5F	3	2
LJMP 77ACh	$(PC) \leftarrow 77ACh$	02 AC 77	3	2

Endereçamento indexado

- O endereço de destino depende não só do endereço inserido na instrução, mas também do conteúdo do **acc** naquele instante, que serve como um índice;
- Usado para leitura de valores armazenados na ROM. São utilizados apontadores de 16 bits (DPTR e PC). Instruções de salto também utilizam;
- Útil para **busca em tabelas** armazenadas na ROM;
- Geralmente o PC não é utilizado.

Endereçamento indexado

Ex: `MOV A, #03`
 `MOV DPTR, #TABELA ; DPTR recebe o end de`
 `; mem cujo label TABELA`
 `MOVC A, @A+DPTR ; A= (A+DPTR)`

Mnemônico	Operação	Código	Bytes	Ciclos
MOVC A, @A+DPTR	$(A) \leftarrow ((A) + (DPTR))$	93	1	2
MOVC A, @A+PC	$(A) \leftarrow ((A) + (PC))$	83	1	2
JMP @A+DPTR	$(PC) \leftarrow ((A) + (DPTR))$	73	1	2

Instruções Sem Modo Endereçamento

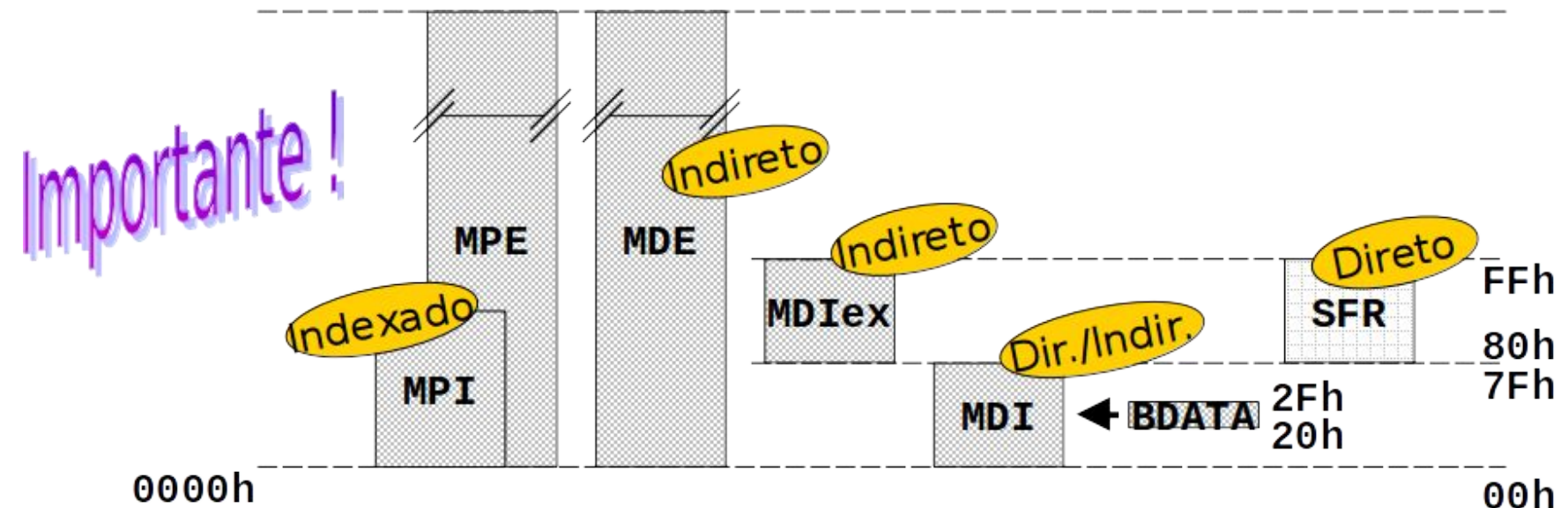
- Há instruções para as quais não se aplicam os modos de endereçamento;
- Instruções específicas de alguns registradores ou que não têm operandos

Mnemônico	Operação	Código	Bytes	Ciclos
CPL A	$(A) \leftarrow \neg(A)$	F4	1	1
INC DPTR	$(DPTR) \leftarrow (DPTR) + 1$	A3	1	2
RET	$(PC) \leftarrow ((SP-1))((SP))$ $(SP) \leftarrow (SP) - 2$	22	1	2

Visão Geral - Endereçamentos

Estas instruções são divididas em três grupos:

- Transferência de dados na MDI
- Transferência de dados na MDE
- Leitura de tabelas armazenadas na MP (I/E)



Tipos de Instruções

Tipos de Instruções

8051 Instruction Set Summary

- Rn** Register R7-R0 of the currently selected Register Bank.
Data 8-bit internal data location's address. This could be an internal data RAM location (0-127) or a SFR (i.e. I/O port, control register, status register, etc. (128-255)).
@Ri 8-bit internal data RAM location (0-255) addressed indirectly through register Ri or RD.
#data 8-bit constant included in instruction.
#data16 16-bit constant included in instruction.
addr16 16-bit destination address. Used by LCALL and LJMP. A branch can be anywhere within the 64K byte Program Memory address space.
addr11 11-bit destination address. Used by ACALL and AJMP. The branch will be within the same 2K byte page of Program Memory as the first byte of the following instruction.
rel Signed 8-bit's complement 8-bit offset byte. Used by SJMP and all conditional jumps. Range is -128 to +127 bytes relative to first byte of the following instruction.
bit Direct Addressed bit in internal data RAM or Special Function Register.

Instruction	C	OV	AC	Instruction	C	OV	AC
ADD	X	X	X	CLR C	0		
ADDC	X	X	X	CPL C	X		
SUBB	X	X	X	ANL C, bit	X		
MUL	0	X		ANL C, bit	X		
DIV	0	X		ORL C, bit	X		
DA	X			ORL C, bit	X		
RRC	X			MOV C, bit	X		
RLC	X			CJNE	X		
SETB C							

Note that operations on SFR byte addresses 200 or bit addresses 209-215 (i.e. the PSW) do not affect flag settings.

Mnemonic	Description	Byte	Cycle
Arithmetic operations			
ADD A, Rn	Add register to accumulator	1	1
ADD A, #data	Add direct byte to accumulator	2	1
ADD A, @Ri	Add indirect RAM to accumulator	1	1
ADD A, #data	Add immediate data to accumulator	2	1
ADDC A, Rn	Add register to accumulator with carry flag	1	1
ADDC A, #data	Add direct byte to A with carry flag	2	1
ADDC A, @Ri	Add indirect RAM to A with carry flag	1	1
ADDC A, #data	Add immediate data to A with carry flag	2	1
SUBB A, Rn	Subtract register to accumulator with borrow	1	1
SUBB A, #data	Subtract direct byte to A with carry borrow	2	1
SUBB A, @Ri	Subtract indirect RAM to A with carry borrow	1	1
SUBB A, #data	Subtract immediate data to A with carry borrow	2	1
INC A	Increment accumulator	1	1
INC Rn	Increment register	1	1
INC #data	Increment direct byte	2	1
INC @Ri	Increment indirect RAM	1	1
DEC A	Decrement accumulator	1	1
DEC Rn	Decrement register	1	1
DEC #data	Decrement direct byte	2	1
DEC @Ri	Decrement indirect RAM	1	1
INC DPTR	Increment data pointer	1	2
MUL AB	Multiply A and B → [B] [A] x [B]	1	4
DIV AB	Divide A by B → A ÷ B = result, B = remainder	1	4
DA A	Decimal adjust accumulator	1	1
CLR A	Clear accumulator	1	1

Mnemonic	Description	Byte	Cycle
PL A	Complement accumulator	1	1
L A	Rotate accumulator left	1	1
LC A	Rotate accumulator left through carry	1	1
R A	Rotate accumulator right	1	1
RC A	Rotate accumulator right through carry	1	1
RR A	Rotate right by the accumulator	1	1

Logic operations

ANL A, #data	AND direct byte to accumulator	2	1
ANL A, @Ri	AND indirect RAM to accumulator	1	1
ANL A, #data	AND immediate data to accumulator	2	1
ANL direct, A	AND accumulator to direct byte	2	1
ANL direct, #data	AND immediate data to direct byte	3	2
RL A, Rn	Rotate register to accumulator	1	1
RL A, #data	OR direct byte to accumulator	2	1
RL A, @Ri	OR indirect RAM to accumulator	1	1
RL A, #data	OR immediate data to accumulator	2	1
RL direct, A	OR accumulator to direct byte	2	1
RL direct, #data	OR immediate data to direct byte	3	2
RL A, Rn	Exclusive OR register to accumulator	1	1
RL A, #data	Exclusive OR direct byte to accumulator	2	1
RL A, @Ri	Exclusive OR indirect RAM to accumulator	1	1
RL A, #data	Exclusive OR immediate data to accumulator	2	1
RL direct, A	Exclusive OR accumulator to direct byte	2	1
RL direct, #data	Exclusive OR immediate data to direct byte	3	2

Boolean variable manipulation

CLR bit	Clear direct bit	2	1
CLR C	Set carry flag	1	1
CLR bit	Set direct bit	2	1
CPL C	Complement carry flag	1	1
PL bit	Complement direct bit	2	1
ANL C, bit	AND direct bit to carry flag	2	2
ANL C, #data	AND complement of direct bit to carry	2	2
ORL C, bit	OR direct bit to carry flag	2	2
ORL C, #data	OR complement of direct bit to carry	2	2
MOV C, bit	Move direct bit to carry flag	2	1

Program and machine control

CALL addr16	Long sub routine call	3	2
RET	Return from sub routine	1	2
RI	Return from interrupt	1	2
JMP addr11	Absolute jump	2	2
JMP addr16	Long jump	3	2
JMP rel	Short jump (relative address)	2	2
JP @A, DPTR	Jump indirect relative to the DPTR	1	2
JZ rel	Jump if accumulator is zero	2	2
JL rel	Jump if accumulator is not zero	2	2
C rel	Jump if carry flag is set	2	2
NC rel	Jump if carry flag is not set	2	2
JB bit, rel	Jump if bit is set	3	2
JB bit, rel	Jump if bit is not set	3	2
JB bit, rel	Jump if direct bit is set and clear bit	3	2
CJNE A, #data, rel	Compare direct byte to A and jump if not equal	3	2

Mnemonic	Description	Byte	Cycle
CJNE A, #data, rel	Compare immediate to A and jump if not equal	3	2
CJNE Rn, #data, rel	Compare immediate to register and jump if not equal	3	2
CJNE @Rn, #data, rel	Compare immediate to indirect and jump if not equal	3	2
DJNE Rn, rel	Decrement register and jump if not zero	2	2
DJNE direct, rel	Decrement direct byte and jump if not zero	3	2

MOV A, Rn	Move register to accumulator	1	1
MOV A, #data	Move direct byte to accumulator	2	1
MOV A, @Ri	Move indirect RAM to accumulator	1	1
MOV A, #data	Move immediate data to accumulator	2	1
MOV Rn, A	Move accumulator to register	1	1
MOV Rn, #data	Move direct byte to register	2	2
MOV Rn, direct	Move direct byte to register	2	1
MOV #data, A	Move accumulator to direct byte	2	1
MOV #data, Rn	Move register to direct byte	2	2
MOV #data, direct	Move direct byte to direct byte	3	2
MOV #data, @Ri	Move indirect RAM to direct byte	2	2
MOV #data, #data	Move immediate data to direct byte	3	2
MOV @Ri, A	Move accumulator to indirect RAM	1	1
MOV @Ri, direct	Move direct byte to indirect RAM	2	2
MOV @Ri, #data	Move immediate data to indirect RAM	2	1
MOV DPTR, #data16	Load data pointer with a 16-bit constant	3	2
MOVC A, @A, DPTR	Move code byte relative to DPTR to accumulator	1	2
MOVC A, @A, PC	Move code byte relative to PC to accumulator	1	2
MOVA A, @Ri	Move external RAM (8-bit address) to A	1	2
MOVA A, @DPTR	Move external RAM (16-bit address) to A	1	2
MOVA @Ri, A	Move A to external RAM (8-bit address)	1	2
MOVA @DPTR, A	Move A to external RAM (16-bit address)	1	2
PUSH #data	Push direct byte onto stack	2	2
POP #data	Pop direct byte from stack	2	2
XCH A, Rn	Exchange register to accumulator	1	1
XCH A, #data	Exchange direct byte to accumulator	2	1
XCH A, @Ri	Exchange indirect RAM to accumulator	1	1
XCHD A, @Ri	Exchange low-order nibble indirect RAM with A	1	1

* MOV A, ACC is not a valid instruction

(mov A, #data, @ (jump if A <= data)	one A, #data, @ add A, Rn(-data) or one A, #data, ne P @ ne, ..
(mov A, #data, @ (jump if A >= data)	add A, Rn(-data+1) or one A, #data+1, ne P @ ne, ..
(mov A, #data, @ (jump if A < data)	add A, Rn(-data) or one A, #data, ne P @ ne, ..
(mov A, #data, @ (jump if A > data)	add A, Rn(-data+1) or one A, #data+1, ne P @ ne, ..
(mov A, #data, @ (jump if A <= data)	add A, Rn(-data) or one A, #data, ne P @ ne, ..
(mov A, #data, @ (jump if A >= data)	add A, Rn(-data+1) or one A, #data+1, ne P @ ne, ..
(mov A, #data, @ (jump if A < data)	add A, Rn(-data) or one A, #data, ne P @ ne, ..
(mov A, #data, @ (jump if A > data)	add A, Rn(-data+1) or one A, #data+1, ne P @ ne, ..
(mov A, #data, @ (jump if A <= data)	add A, Rn(-data) or one A, #data, ne P @ ne, ..
(mov A, #data, @ (jump if A >= data)	add A, Rn(-data+1) or one A, #data+1, ne P @ ne, ..



Enjoy It! Mates

Tipos de Instruções

- Há 255 instruções *assembly* na arquitetura 8051 divididas basicamente em 5 grupos:
 - **Transferência de Dados** (*Data Transfer*)
 - Transferência de dados na memória RAM interna
 - Transferência de dados na memória RAM externa
 - Leitura de tabelas armazenadas na memória ROM
 - **Operações Lógicas** (*Logical*)
 - **Operações Aritméticas** (*Arithmetic*)
 - **Manipulação de Variáveis Booleanas** (Bits) (*Boolean*)
 - Booleanas de desvio
 - **Controle de Programa** (Desvios) (*Program branching*)
 - Salto incondicional
 - Salto condicional

Tipos de Instruções

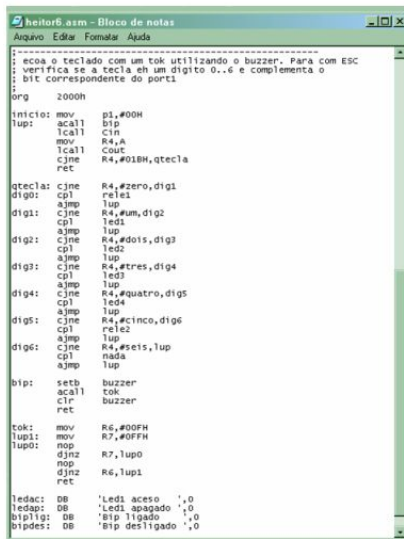
- O importante não é conhecer todas as instruções, mas sim as características dos seus grupos principais.
- A referência para o conjunto de instruções do 8051 se encontra disponível no moodle da disciplina na seção **Recursos**
 - Guia do Programador e de Instruções da Família 80C51
 - Referência completa
 - Tabela de Instruções do 8051
 - Tabela resumida
 - Mapa Mental de Instruções do 8051
- Na Internet uma ótima fonte é:
 - http://www.keil.com/support/man/docs/is51/is51_opcodes.htm

Diretivas de compilação

- Não são instruções, mas definem endereços, nomes ou outras informações importantes.
- Alguns compiladores exigem a definição prévia dos símbolos utilizados (bytes e bits dos SFRs) e seus respectivos endereços.

Diretiva	Significado	Função	Exemplo
ORG	Origin	Define a posição da mem de prog de início do programa	ORG 2000h
EQU	Equate	Cria sinônimos para dar significado aos elementos do programa	sensor EQU P3.1 velocidade EQU 08h
DB <valor>	Define Byte	Cria variáveis em memória	contador: DB 00h
DB <valor>,...	Define Byte	Cria vetores em memória	Tabela: DB 15h,22h,35h
DB <string>	Define Byte	Cria strings ASCII em memória	MSG: DB "UTFPR"
END	Fim	Indica o final do programa	

Ciclo de desenvolvimento de firmware para microcontroladores



```
heitor.asm - Bloco de notas
Arquivo Editar Formatar Ajuda

: ecoa o teclado com um tok utilizando o buzzer. Para com ESC
: verifica se a tecla eh um digito 0..6 e complementa o
: bit correspondente do port1
:
org 2000h

intctio: mov p1,#00H
lup:      acall b1p
         call cin
         mov R4,A
         lcall cout
         cjne R4,#01BH,qtecla
         ret

qtecla:  cjne R4,#zero,dig1
dig0:    cpl
         ajmp lup
dig1:    cjne R4,#um,dig2
         cpl
         led1
         ajmp lup
dig2:    cjne R4,#dois,dig3
         cpl
         led2
         ajmp lup
dig3:    cjne R4,#tres,dig4
         cpl
         led3
         ajmp lup
dig4:    cjne R4,#quatro,dig5
         cpl
         led4
         ajmp lup
dig5:    cjne R4,#cinco,dig6
         cpl
         led5
         ajmp lup
dig6:    cjne R4,#seis,lup
         cpl
         nada
         ajmp lup

b1p:     setb buzzer
         acall tok
         clr buzzer
         ret

tok:     mov R6,#00FH
lup1:    mov R7,#0FFH
lup0:    nop
         djnz R7,lup0
         nop
         djnz R6,lup1
         ret

ledac:   db 'Led1 aceso ','0
ledap:   db 'Led1 apagado ','0
b1p1gi:  db 'B1p ligado ','0
b1pdes:  db 'B1p desligado ','0
```

Salva arquivo



```
C:\WINNT\system32\cmd.exe
O volume na unidade D é Winchester2
O número de série do volume é E532-FB21

Pasta de D:\Heitor\cefet\DISCIP\1\Digital2\Ntestes
13-01-05 02:09          10.362 ex1.asm.txt
13-01-05 02:09          1.406 ex1.2.hex.txt
24-01-05 19:29          4.383 HSL-LCD.ASM.txt
                   16.151 bytes
0 arquivo(s) 3.489.144.832 bytes disponíveis
D:\Heitor\cefet\DISCIP\1\Digital2\Ntestes>rename ex1.asm.txt ex1.asm
D:\Heitor\cefet\DISCIP\1\Digital2\Ntestes>asm51 ex1.asm
8051 Cross-Assembler, Version 1.2h
(c) Copyright 1984, 1985, 1986, 1987, 1988, 1989, 1990
by Metalink Corporation
First pass
Second pass
ASSEMBLY COMPLETE, 50 ERRORS FOUND
D:\Heitor\cefet\DISCIP\1\Digital2\Ntestes>_
```

Compilação c/ASM51

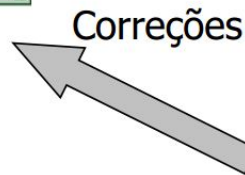
- Programa: **.HEX**
- Listagem: **.LST**



Gravação na memória de programas do 8051 (flash ou EEPROM) ou em ROM externa



Teste e depuração *in-circuit*



Correções

Edição

- Formato texto
- Extensão **.ASM**

Software μ Vision da Keil

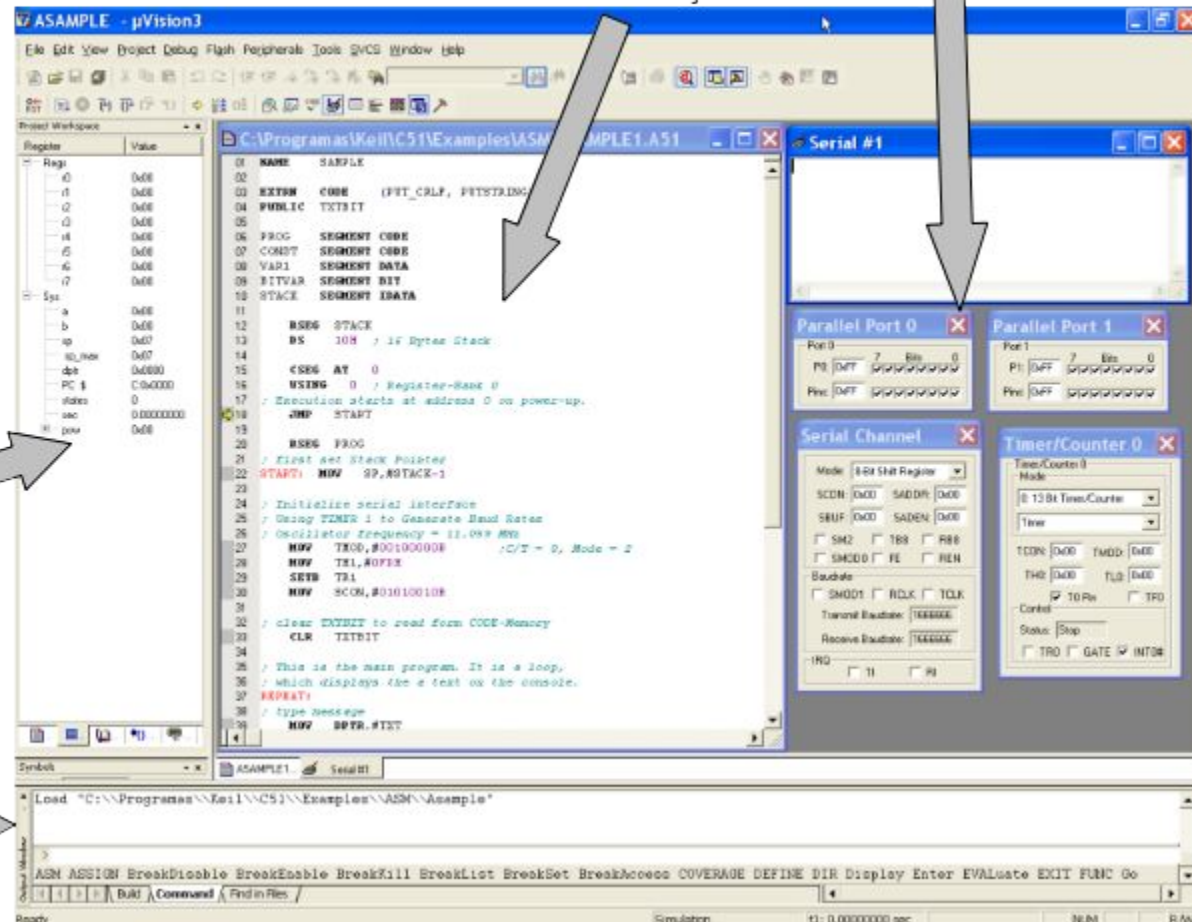
- Simulação
- Emulação *in-circuit*

Área de edição e simulação

Gerenciamento de arquivos do projeto (modo edição)

Registradores internos da CPU (modo execução)

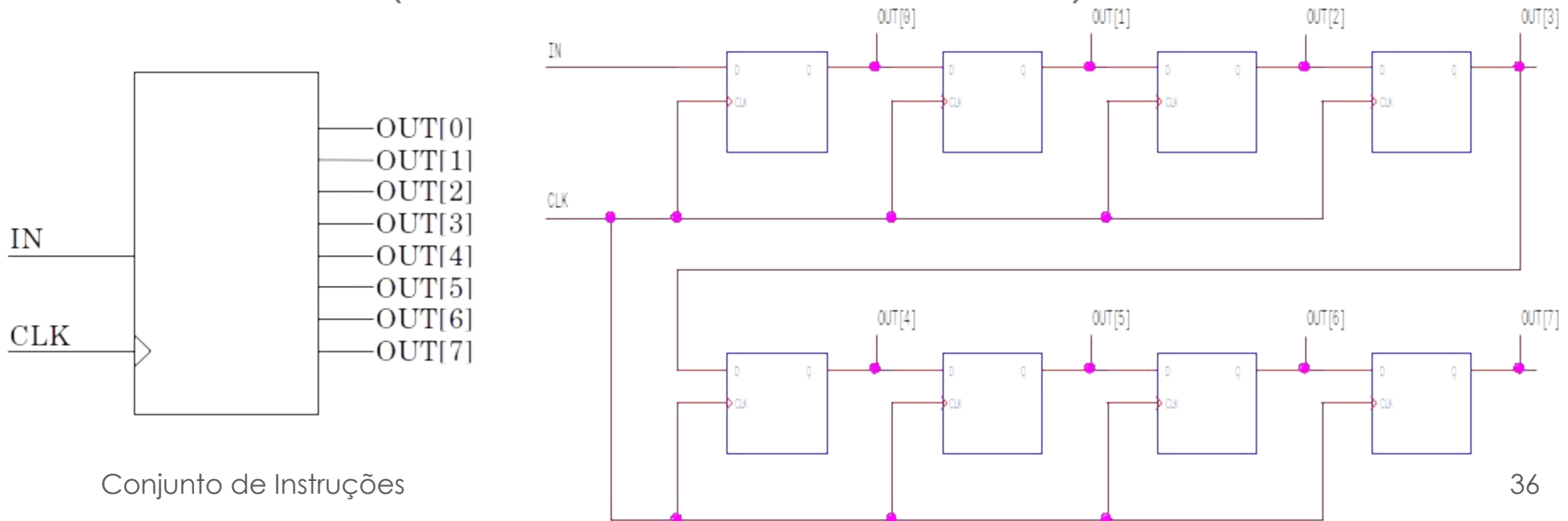
Mensagens de compilação, warnings, e



Exemplo de aplicação

Desenvolver por software um registrador de deslocamento de 8 bits, considerando:

- CLK: P0.0 (sensível à borda de subida)
- IN: P0.1
- OUT: P1 (inicialmente em 00000000b)



Sequência de operações

- Garantir a condição inicial do RESET
- Aguardar pela subida do CLK
- Após a subida em CLK, deslocar o conteúdo de OUT[7..0] uma posição, no sentido de OUT[7] OUT[0]
- Colocar o conteúdo de IN em OUT[0]

Codificação em assembly

```
1      $mod51
2
0000   3      ORG      0000h
4      ;
0000 E4   5      inicio: CLR      A          ;limpa acumulador
0001 F590  6              MOV      P1,A        ;coloca OUT[7..0] em 0
0003 3080FD 7      loop1: JNB      P0.0,loop1    ;espera subida do CLK
0006 A281   8              MOV      C,P0.1      ;coloca IN no Carry
0008 33     9              RLC      A          ;faz deslocamento c/ Carry
0009 F590  10             MOV      P1,A        ;atualiza saídas
000B 2080FD 11      loop2: JB       P0.0,loop2    ;espera
000E 80F3   12             SJMP     loop1        ;volta ao loop
13      ;
14      END
```

Questão: por quê é necessário a linha 11?

Sofisticação do projeto (1)

Incluir uma entrada de habilitação (Enable) em P0.2:

```

                ORG      0000h
inicio:  CLR      A              ;limpa acumulador
                MOV      P1,A      ;coloca OUT[7..0] em 0
loop1:   JNB      P0.0,loop1      ;espera subida do CLK
                MOV      C,P0.1    ;coloca IN no Carry
                JNB      P0.2,loop2 ;habilitação está ativa?
                RLC      A          ;faz deslocamento com Carry
                MOV      P1,A      ;atualiza saídas
loop2:   JB       P0.0,loop2      ;espera descida do CLK
                SJMP     loop1     ;volta ao loop
                END
```

Sofisticação do projeto (2)

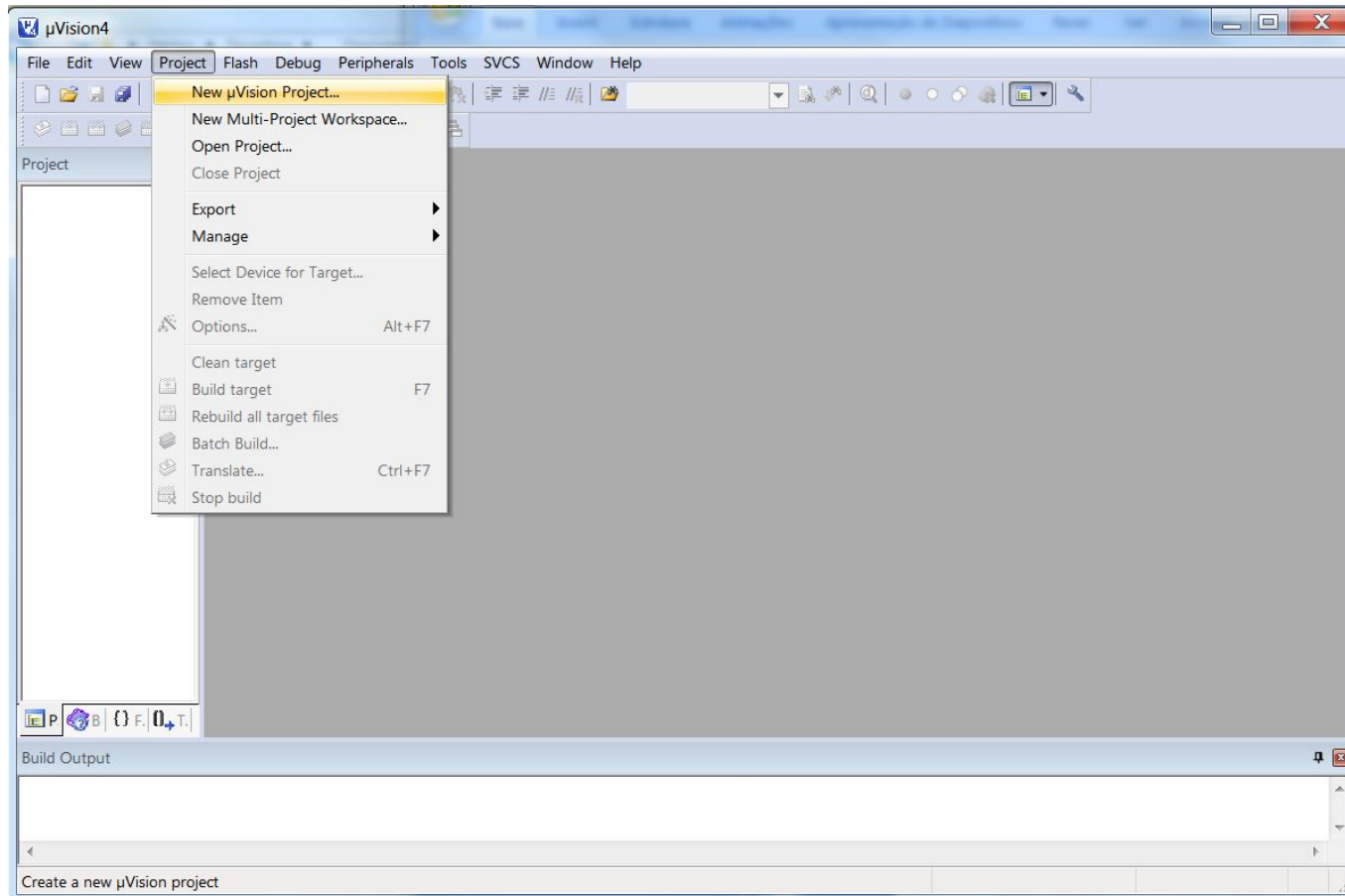
Incluir uma entrada de sentido (Up-Down) em P0.3:

```

                ORG      0000h
inicio:  CLR      A              ;limpa acumulador
                MOV      P1,A      ;coloca OUT[7..0] em 0
loop1:   JNB      P0.0,loop1      ;espera subida do CLK
                MOV      C,P0.1    ;coloca IN no Carry
                JNB      P0.2,loop2 ;habilitação está ativa?
                JNB      P0.3,dir   ;sentido para a direita ?
esq:     RLC      A              ;desloca p/esquerda c/Carry
                SJMP     atual
dir:     RRC      A              ;desloca p/direita c/Carry
atual:   MOV      P1,A      ;atualiza saídas
loop2:   JB       P0.0,loop2      ;espera descida do CLK
                SJMP     loop1     ;volta ao loop
                END
```

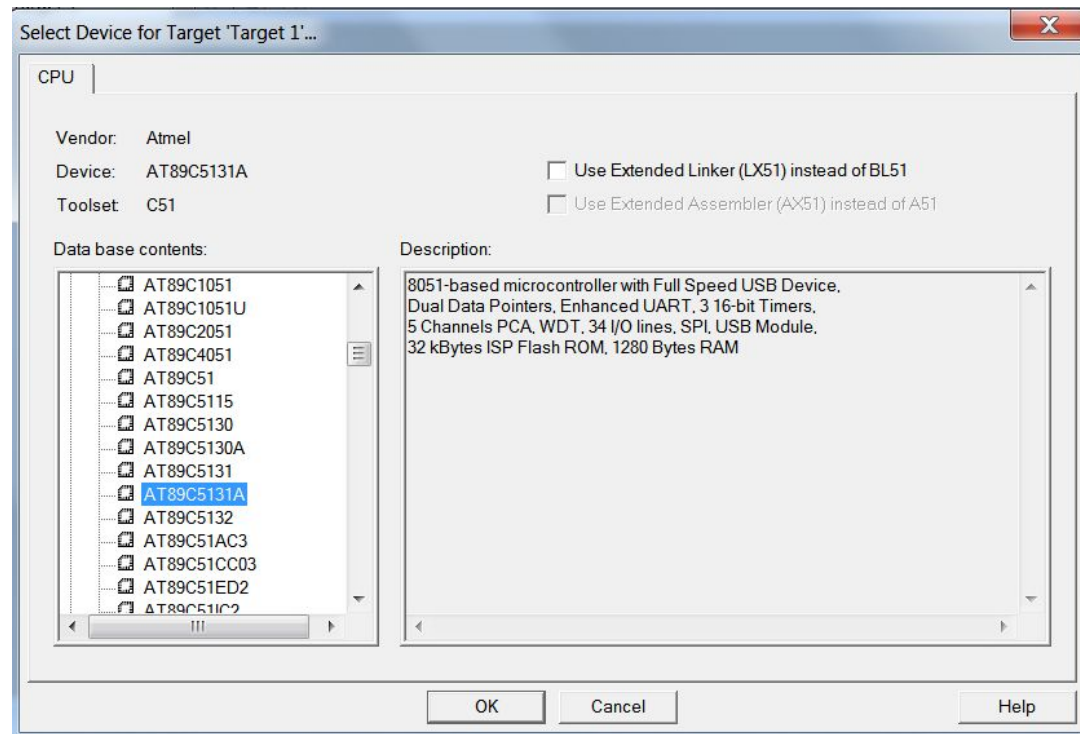

Passos para a utilização do Keil μ Vision - 1

Criar um novo projeto e salvar numa pasta de trabalho



Passos para a utilização do Keil μVision - 2

- Selecionar Atmel e o nome do processador:
AT89C5131A
- Copy standard 8051 startup code to project folder and
add file to project ? NÃO



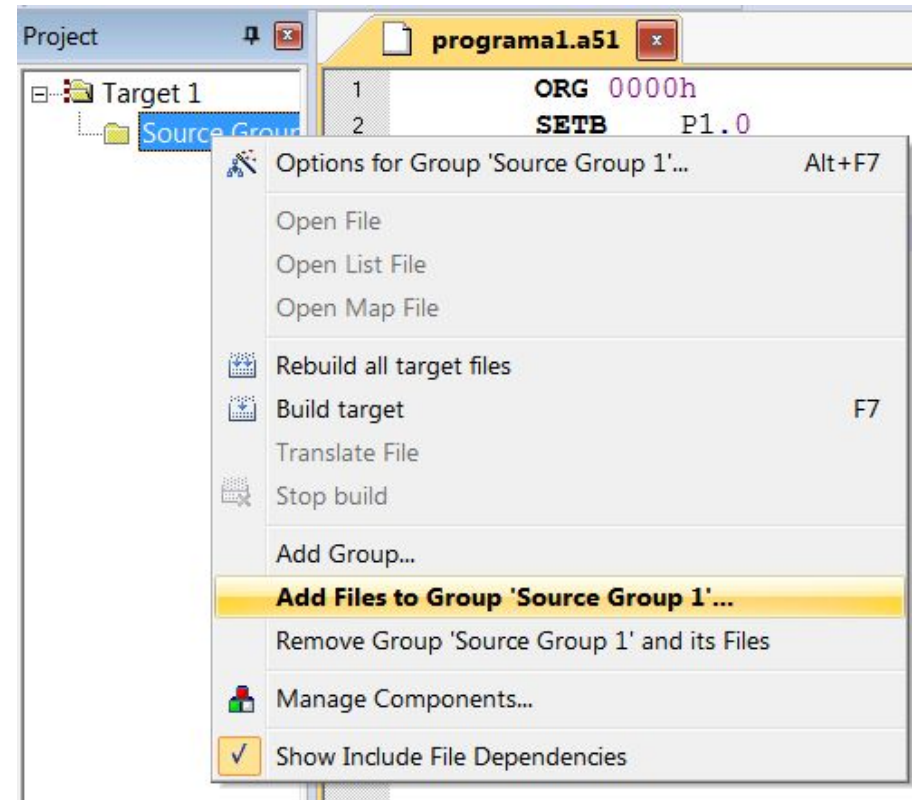
Passos para a utilização do Keil μVision - 3

Criar um novo arquivo com
FILE/NEW

Editar o arquivo, iniciando
com ORG e terminando
com END

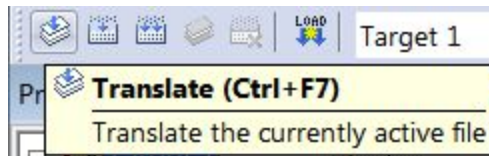
Salvar o arquivo com
extensão A51 na mesma
pasta do projeto.

Incluir o novo programa no
projeto.



Passos para a utilização do Keil μVision - 4

Compilar o Programa:



A janela de saída apresenta o resultado, erros e warnings:

```
Build Output
assembling program1.a51...
program1.a51 - 0 Error(s), 0 Warning(s).
```

Linkar o Programa:



A janela de saída apresenta o resultado:

```
Build Output
Build target 'Target 1'
linking...
Program Size: data=8.0 xdata=0 code=5
"teste" - 0 Error(s), 0 Warning(s).
```

Passos para a utilização do Keil μ Vision - 5

1. Executar o programa na aba DEBUG
2. Utilizar os comandos:
 - RUN
 - STOP
 - STEP
3. Acompanhar as janelas:
 - Disassembly
 - Registers
 - Memory

