

SEL-433 APLICAÇÕES DE MICROPROCESSADORES I

8051

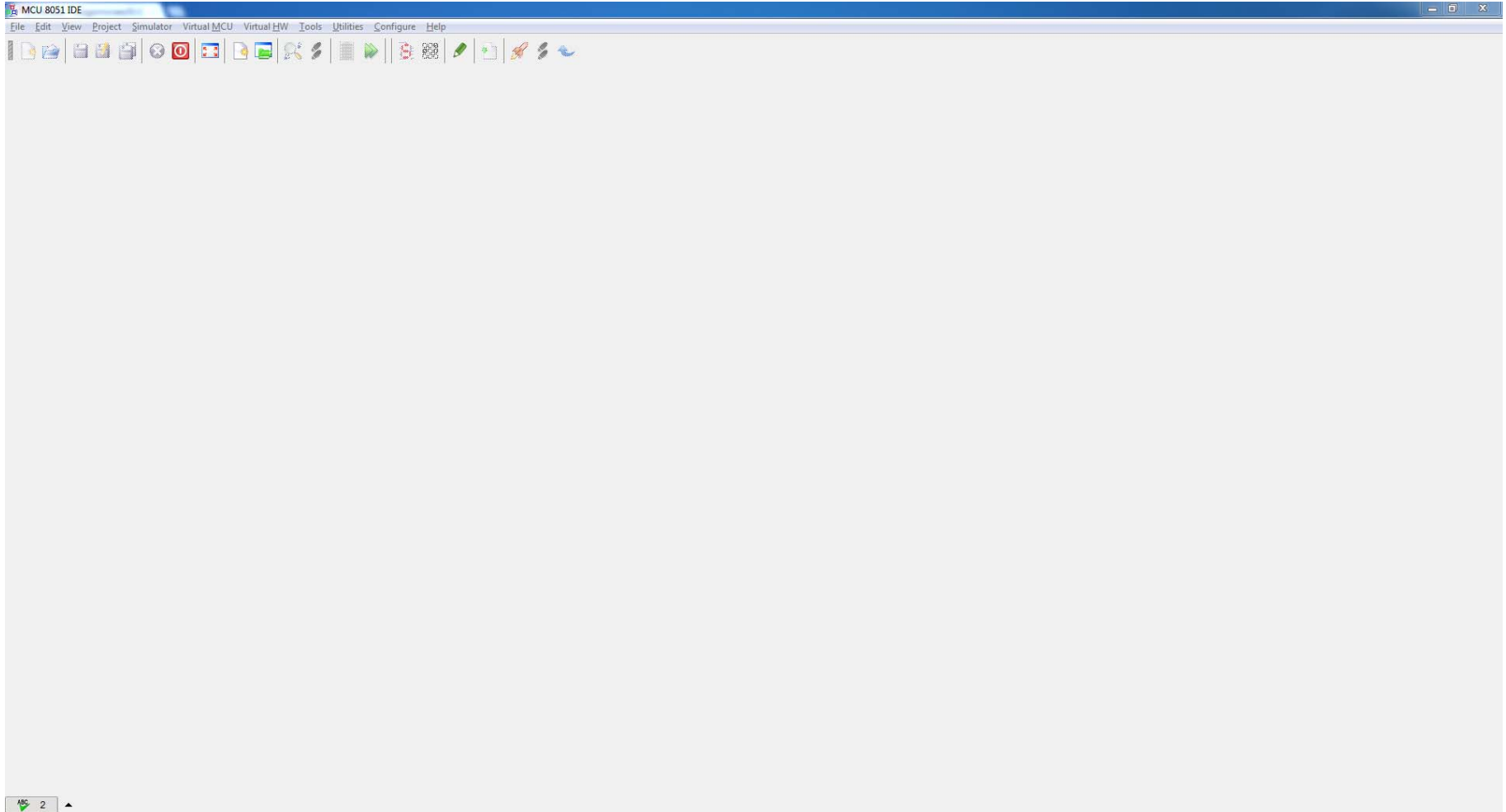
AULA 3

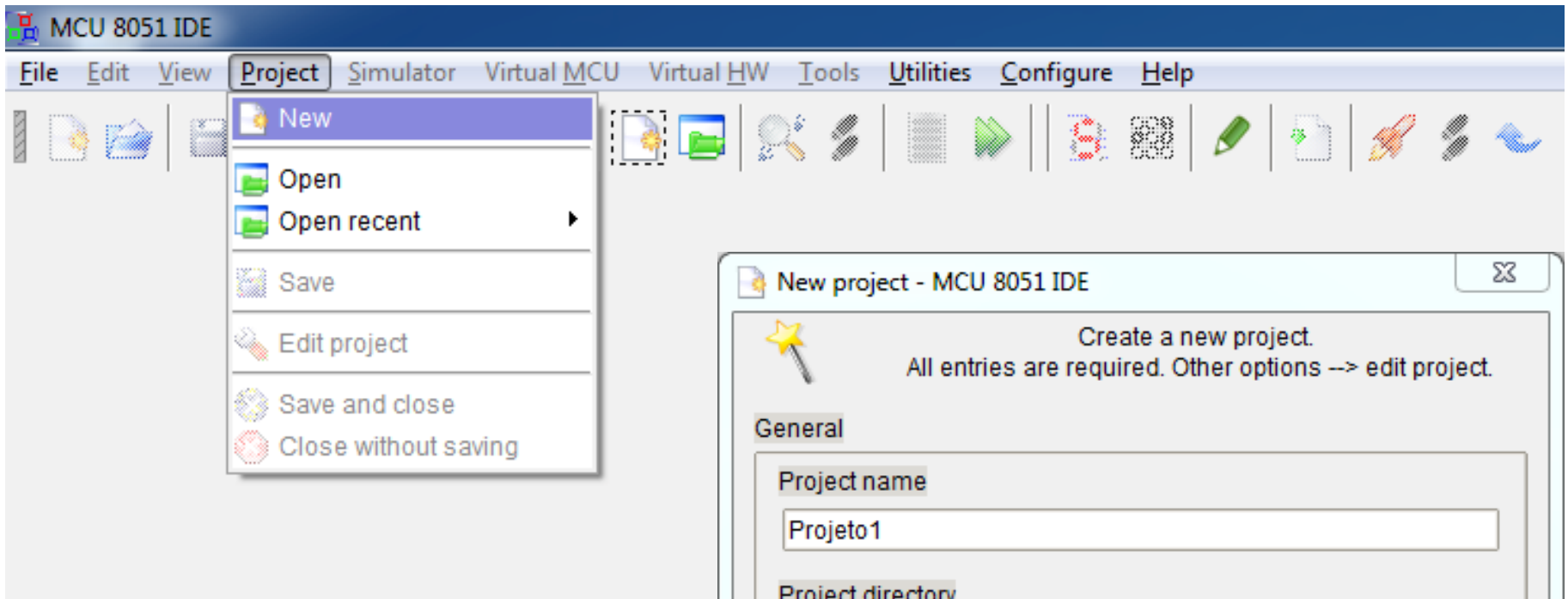
Ambiente de desenvolvimento de
Programação Assembly

MCU 8051 IDE

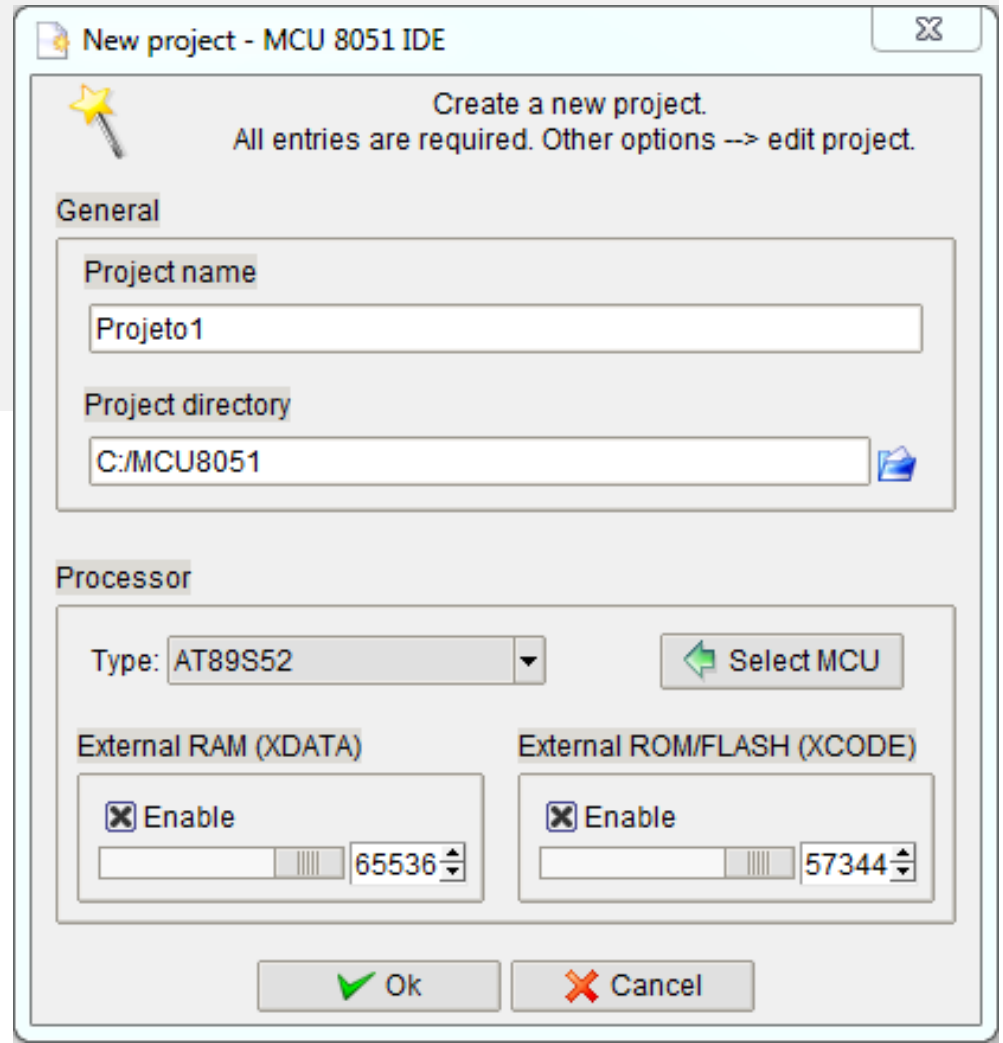
<http://mcu8051ide.sourceforge.net/>

Tela inicial





Criar novo projeto:
Project → New



Criar novo programa em Assembly

File → New

The screenshot shows the MCU 8051 IDE interface. The 'File' menu is open, and the 'New' option is highlighted. The main workspace is empty, and the status bar at the bottom indicates 'Line: 1 Column: 1 Total: 1 INS NORM'. The bottom panel displays various system registers and their values.

File Edit View Project Simulator Virtual MCU Virtual HW Tools Utilities Configure Help

New Ctrl+N
Open Ctrl+O
Open recent
Save Ctrl+S
Save as Ctrl+Shift+S
Save all Ctrl+L
Close Ctrl+W
Close all
File statistics
Save session
Quit Ctrl+Q

untitled

Line: 1 Column: 1 Total: 1 INS NORM

untitled ASM

Simulator C variables IO Ports Messages Notes Calculator Find in files Hide

TIMERS 0 & 1

	TH1	TL1	TH0	TL0	T1	T0	TCON	TMOD
P0:	11111111	FF	DPTR0:	00	00	SP:	07	
P1:	11111111	FF	DPTR1:	00	00	SBUF R:	94	
P2:	11111111	FF	Clock:	12000	SBUF T:	14		
P3:	11111111	FF	PC:	HEX 00	DEC 0			

INTERRUPTS

	E:	EA	ET2	ES	ET1	EX1	ET0	EX0
IP:	-	-	PT2	PS	PT1	PK1	PT0	PX0

PCON: SMOD - - - GF1 GF0 PD DL H: 00

SCON: SM0 SM1 SM2 REN TB8 RB8 TI RI H: 00

PSW: C AC F0 RS1 RS0 OV - P

R7 R6 R5 R4 R3 R2 R1 R0

HEX DEC BIN OCT CHAR

A: 00 0 00000000 0

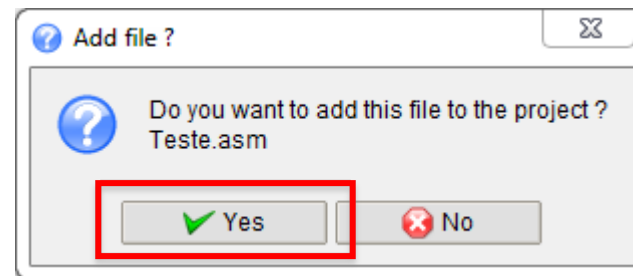
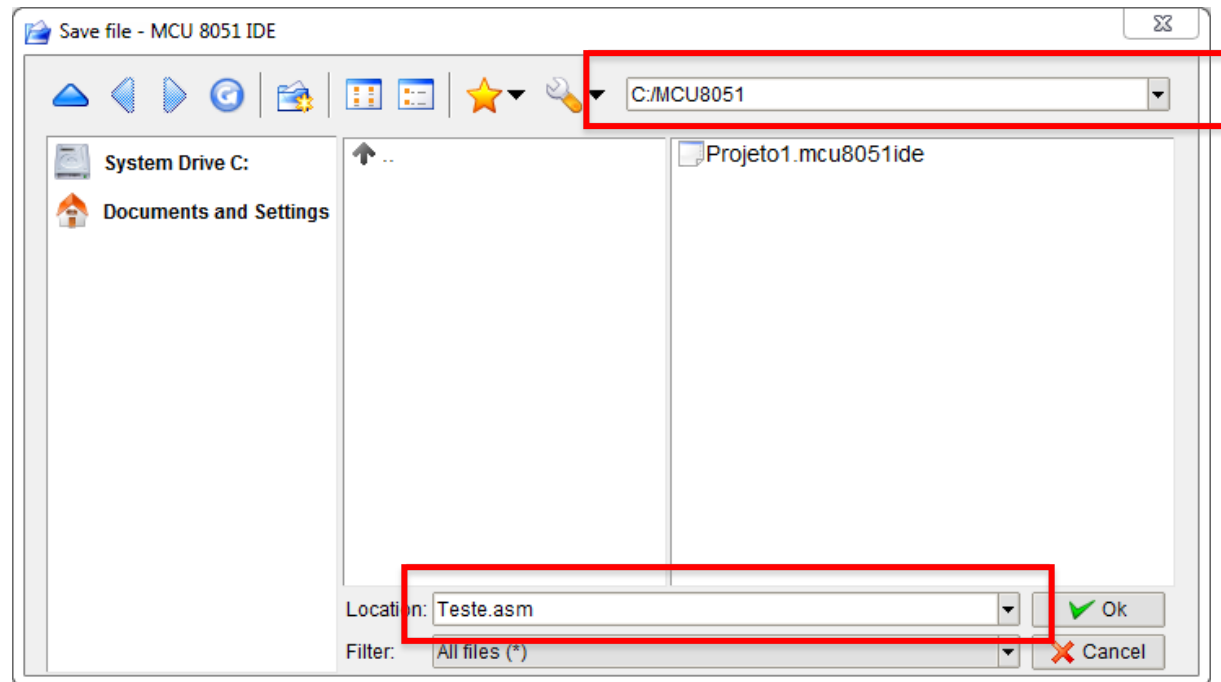
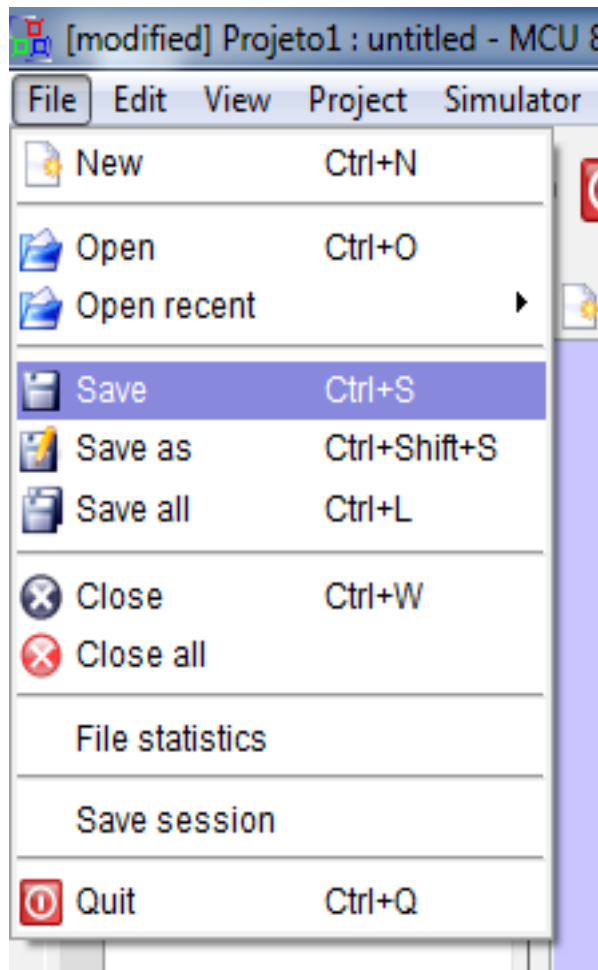
B: 00 0 00000000 0

Time: []

2 Create new file

AT89S52

Salvar um programa File → Save



Obs: O arquivo deve ser salvo como “Nome.asm” em um diretório próximo ao raiz. O Nome deve ser curto.

Exemplo de um Programa em Assembly: (Código Fonte)

```
1 ;Programa para debounce de uma chave
2 ;Necessário para remover ruídos da chave ao pressioná-la que pode
3 ;causar mais de uma contagem
4
5     ORG     0           ;Início do programa (PC=0)
6
7     CLR     A           ;Inicializa o contador em zero
8 LOOP: JB     P1.0, $     ;Aguarda por nível lógico 0 (Chave aberta)
9         INC     A           ;Incrementa o contador
10        MOV     P2, A     ;Coloca o valor do contador em P2
11        LCALL   ATRASO    ;Chama rotina de atraso
12        JNB     P1.0, $   ;Aguarda por nível lógico 1 (Chave fechada)
13        SJMP    LOOP      ;Retorna para nova contagem de pulso
14
15 ATRASO: MOV     R0, #0FFh ;Valor de atraso para debounce (aproximadamente 500us)
16         DJNZ   R0, $     ;Decrementa até zero
17         RET              ;Retorna da rotina
18
19        END              ;Fim do programa
```

Formato de um programa

<Rótulo> <Operação> <Operandos> <Comentários>

Rótulo:

- Primeiro caractere alfabético, limitado a 13 caracteres
- “espaço”, “tab” e “:” são considerados fim do rótulo
- Corresponde ao endereço da instrução seguinte
- É opcional

```
7  
8 LOOP: CLR A ;Inicializa o contador em zero  
9 JB P1.0, $ ;Aguarda por nível lógico 0 (Chave aberta)  
INC A ;Incrementa o contador
```

Formato de um programa

Utilização de rótulos

```
1 ;Programa para debounce de uma chave
2 ;Necessário para remover ruídos da chave ao pressioná-la que pode
3 ;causar mais de uma contagem
4
5     ORG     0           ;Início do programa (PC=0)
6
7     CLR     A           ;Inicializa o contador em zero
8 LOOP: JB     P1.0, $    ;Aguarda por nível lógico 0 (Chave aberta)
9         INC     A       ;Incrementa o contador
10        MOV     P2, A   ;Coloca o valor do contador em P2
11        LCALL  ATRASO   ;Chama rotina de atraso
12        JNB    P1.0, $  ;Aguarda por nível lógico 1 (Chave fechada)
13        SJMP   LOOP     ;Retorna para nova contagem de pulso
14
15 ATRASO: MOV    R0, #0FFh ;Valor de atraso para debounce (aproximadamente 500us)
16         DJNZ   R0, $    ;Decrementa até zero
17         RET                ;Retorna da rotina
18
19     END                ;Fim do programa
```


Formato de um programa

<Rótulo> <Operação> <Operandos> <Comentários>

Operação:

- Contém o mnemônico da instrução ou diretivas do programa
- Não diferencia letras maiúsculas e minúsculas

7
8
9

LOOP:

CLR
JB
INC

A
P1.0, \$
A

*;Inicializa o contador em zero
;Aguarda por nível lógico 0 (Chave aberta)
;Incrementa o contador*

Formato de um programa

<Rótulo> <Operação> <Operandos> <Comentários>

Operandos:

- Especifica o dado a ser operado pela instrução

7
8
9

LOOP: CLR
JB
INC

A
P1.0, \$
A

*;Inicializa o contador em zero
;Aguarda por nível lógico 0 (Chave aberta)
;Incrementa o contador*

Formato de um programa

<Rótulo> <Operação> <Operandos> <Comentários>

Comentários:

- É utilizado para comentar o que está sendo feito para facilitar o entendimento
- É opcional
- Iniciado com “;”

7
8
9

```
LOOP: CLR A  
      JB P1.0, $  
      INC A
```

```
;Inicializa o contador em zero  
;Aguarda por nível lógico 0 (Chave aberta)  
;Incrementa o contador
```

Valores numéricos

- Base binária: #11110000**b**
- Base decimal: #255**d**
- Base hexadecimal: #4F**h**
- Base octal: #20**q**

IMPORTANTE!

Ao utilizar a base hexadecimal, se o número começar com uma letra (Ex: ABh), deve-se colocar um 0 (zero) antes da letra.

Exemplo: #0ABh

Outros valores para operandos

- Caractere ASCII: #'A'

Exemplo: MOV R0, #'0'

'0' (zero) corresponde a 30h ou 48d na tabela ASCII

- Endereço da instrução atual: \$

Exemplo: JB P1.0, \$

Se o bit 0 de P1 estiver em nível lógico 1, pula para o mesmo lugar (faz a mesma instrução novamente)

Sai desta posição quando P1.0 estiver em nível lógico 0

Diretivas do compilador

São utilizadas para complementar as informações para permitir a montagem efetiva do programa.

- Indicar o Endereço Inicial do Programa.
- Reservar área de Dados
- Definir equivalência entre valores
- Etc...

Diretivas do compilador

Diretiva ORG – Origem do programa

ORG endereço

Utilizada para instruir o Assembler em qual endereço deve começar a colocar o código.

Por padrão, na ausência desta diretiva, o código começa no endereço 0.

ORG 0 → Inicia o código no endereço zero

ORG 10h → Inicia o código no endereço 10h

Diretivas do compilador

Diretiva DB – Define byte

DB [bytes]

Utilizada para inserir bytes de dados diretamente na memória de programa.

DB 10h ;Coloca o byte 10h na posição atual do código

DB 20h, 30h, 40h ;Coloca os 3 bytes em ordem a partir da posição atual do código

DB 20h, 'teste', '\$' ;Coloca o byte 20h seguido dos caracteres de 'teste' seguido pelo caractere '\$'

Diretivas do compilador

Diretiva DB – Define byte

DB [bytes]

Utilizada para inserir bytes de dados diretamente na memória de programa.

ORG 100h ;Seleciona a posição 100h do código

DB 01h, 02h ;Coloca os bytes em ordem

DB '0123' ;Coloca os caracteres ASCII da mensagem

End.	100h	101h	102h	103h	104h	105h
Dado	01h	02h	30h ('0')	31h ('1')	32h ('2')	33h ('3')

Diretivas do compilador

Diretiva DW – Define word

DW dado[16bits]

Utilizada para inserir palavras de 16 bits (2 bytes) de dados diretamente na memória de programa.

ORG 100h ;Seleciona a posição 100h do código

DW 1234h ;Coloca os bytes em ordem

DW 40h

DW 'A'

End.	100h	101h	102h	103h	104h	105h
Dado	12h	34h	00h	40h	41h ('A')	00h

Diretivas do compilador

É possível utilizar labels para DB e DW

Exemplo:

```
mensagem:  DB  'texto'
```

```
          MOV  DPTR, #mensagem
```

Utilizado para ler dados da memória de programa

Faz DPTR guardar o endereço do primeiro byte de mensagem

Diretivas do compilador

Diretiva EQU – Equate

var EQU value

Atribui o valor 'value' para uma variável 'var'

- A variável só pode receber um único valor.
- O valor pode ser um valor numérico ou uma expressão.
- Uma vez declarado o valor da variável este não poderá mudar.

Diretivas do compilador

Diretiva EQU – Equate

var EQU value

Atribui o valor 'value' para uma variável 'var'

Exemplo:

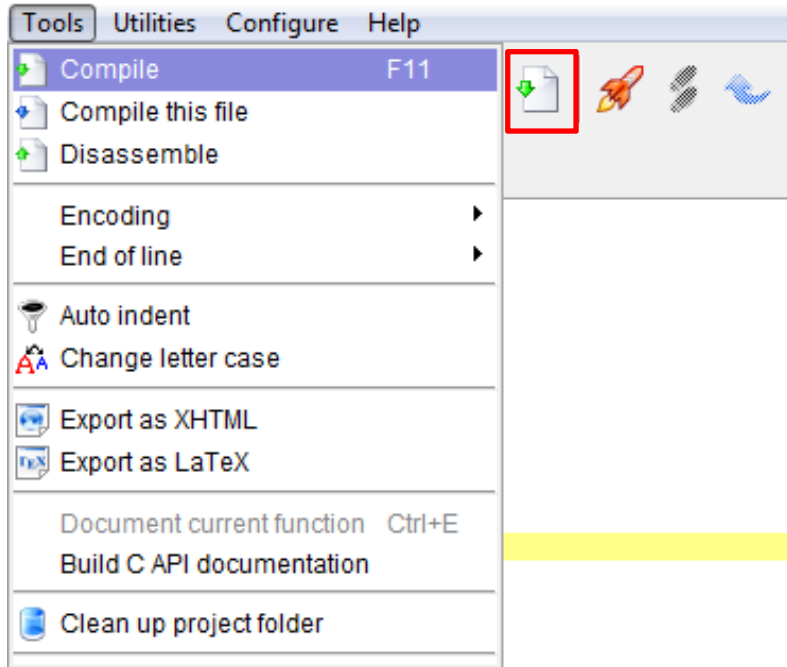
Controle EQU 10h ;atribui 10h para Controle

ORG 0

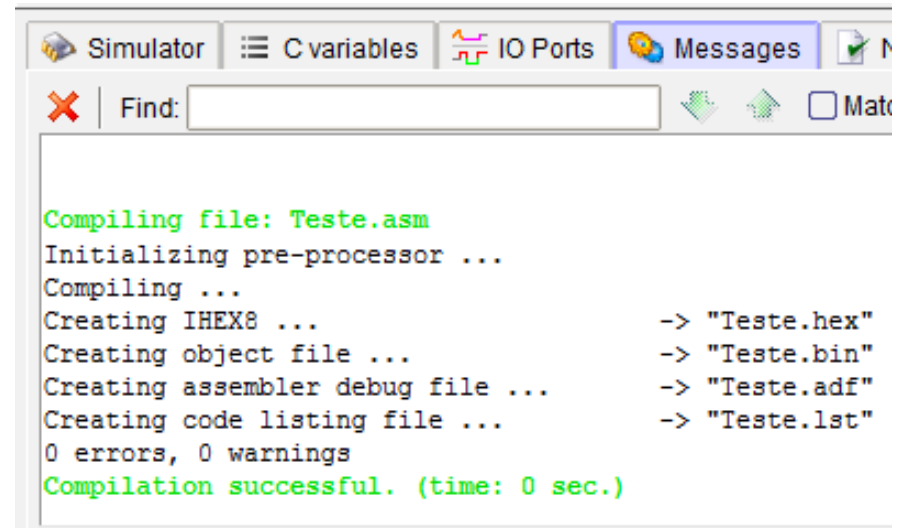
MOV A, #Controle ;Acumulador = 10h

Para Compilar o
Código Fonte e gerar o
Código Objeto

Tools → Compile

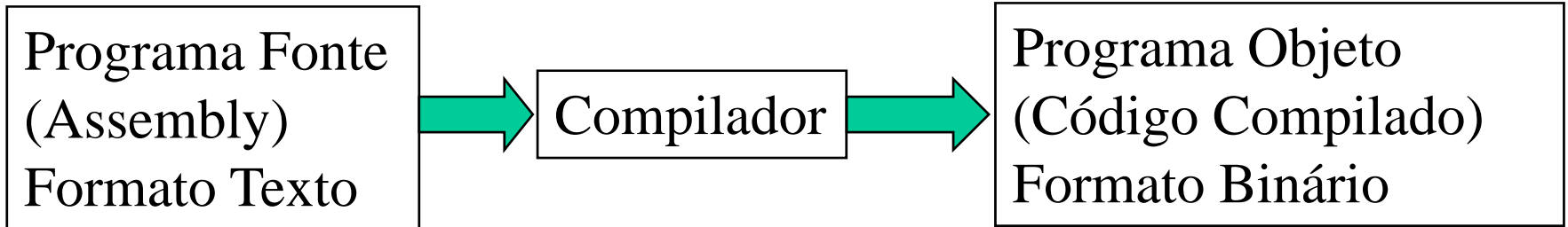


O Código Objeto é gerado no formato
.HEX no mesmo diretório do arquivo
do Código Fonte



A janela “Messages” mostra se a
compilação não teve erros ou em que
linhas do Código Fonte existem erros

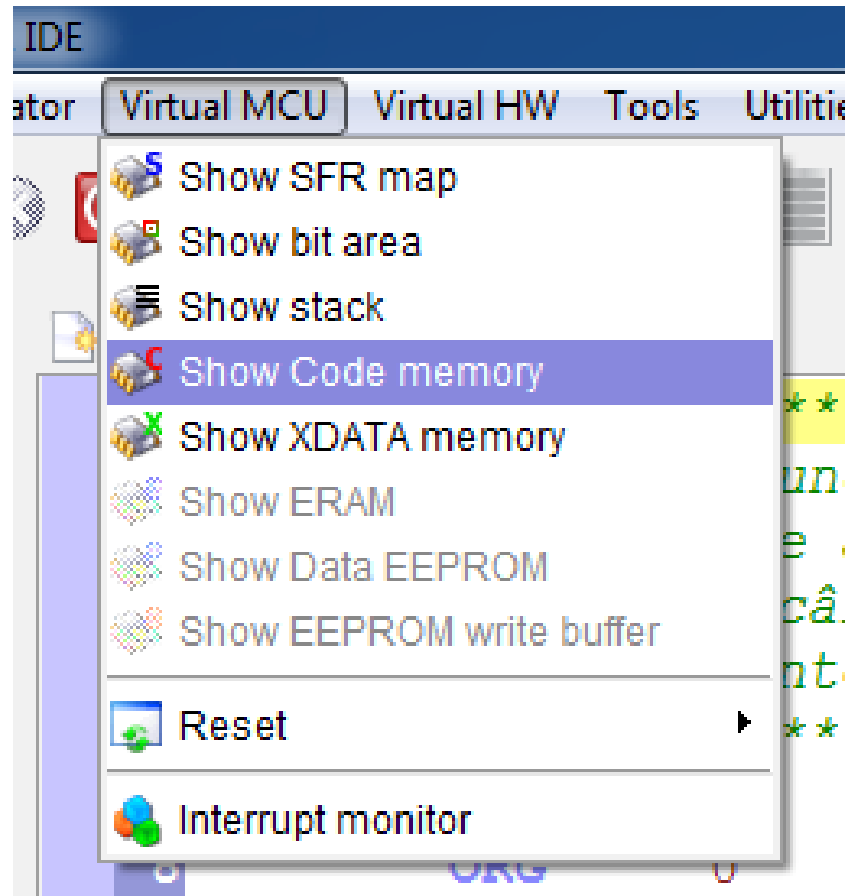
Compilação do código fonte



```
1 ;Programa para debounce de uma chave
2 ;Necessário para remover ruídos da chave ao pressioná-la que pode
3 ;causar mais de uma contagem
4
5     ORG     0           ;Início do programa (PC=0)
6
7     CLR     A           ;Inicializa o contador em zero
8 LOOP: JB     P1.0, $    ;Aguarda por nível lógico 0 (Chave aberta)
9         INC     A           ;Incrementa o contador
10        MOV     P2, A     ;Coloca o valor do contador em P2
11        LCALL   ATRASO    ;Chama rotina de atraso
12        JNB    P1.0, $    ;Aguarda por nível lógico 1(chave fechada)
13        SJMP   LOOP      ;Retorna para nova contagem de pulso
14
15 ATRASO: MOV     R0, #0FFh ;valor de atraso para debounce (aproximadamente 500us)
16         DJNZ   R0, $     ;Decrementa até zero
17         RET
18
19     END                ;Fim do programa
```

E4 20 90 FD 04 F5 A0 12 00 0F 30 90 FD 80 F2 78 FF D8 FE 22

Para visualizar a organização da memória de programa



Virtual MCU → Show Code memory


```

1  ORG      0
2
3  → MOV     00h, #10h ;(Direto / Imediato)
4  → MOV     00h, 10h  ;(Direto / Direto)
5  → MOV     R0, #10h  ;(Rn / Imediato)
6  → MOV     R1, #10h  ;(Rn / Imediato)
7  → MOV     A, #10h   ;(Imediato)
8
9  END

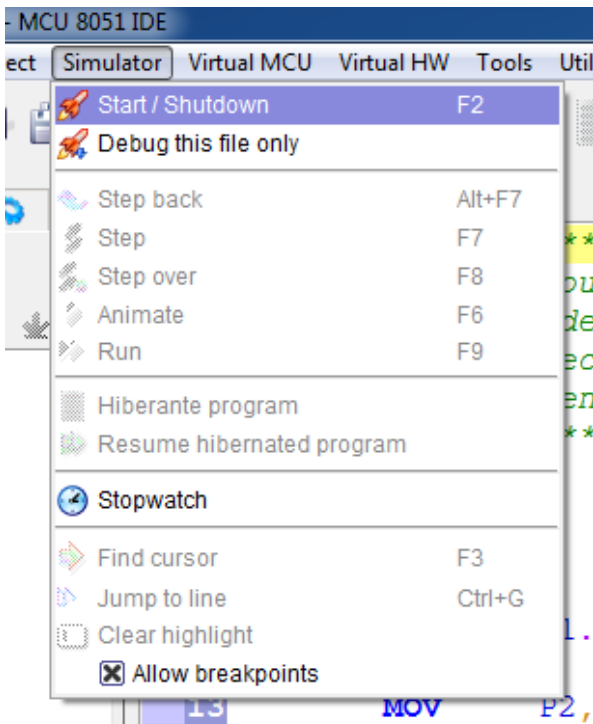
```

The diagram illustrates the mapping of assembly instructions to memory addresses in a hex editor. The instructions and their corresponding memory locations are as follows:

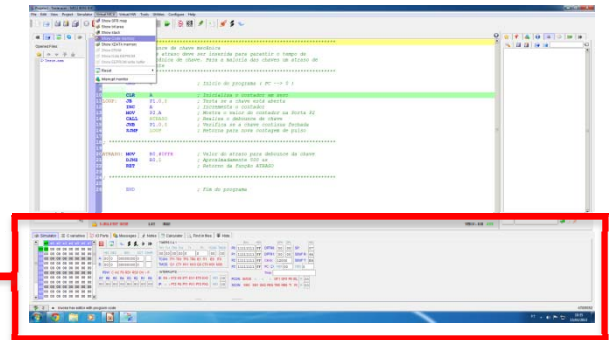
- Line 3: MOV 00h, #10h ;(Direto / Imediato) → Address 0000 (hex 75 00 10)
- Line 4: MOV 00h, 10h ;(Direto / Direto) → Address 0001 (hex 85 10 00)
- Line 5: MOV R0, #10h ;(Rn / Imediato) → Address 0002 (hex 78 10)
- Line 6: MOV R1, #10h ;(Rn / Imediato) → Address 0003 (hex 79 10)
- Line 7: MOV A, #10h ;(Imediato) → Address 0004 (hex 74 10)

Address	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xA	xB	xC	xD	xE	xF	HEX	ASCII
0000	75	00	10	85	10	00	78	10	79	10	74	10					0	u.....x.y.t.
0010																		
0020																		
0030																		
0040																		
0050																		
0060																		
0070																		
0080																		
0090																		
00A0																		
00B0																		
00C0																		
00D0																		
00E0																		
00F0																		

http://www.keil.com/support/man/docs/is51/is51_instructions.htm



Simulador:
 Pode ser utilizado como teste de mesa
 Exibe os valores atuais de:
 registradores, acumulador, portas, memória etc



Simulator → Start/Shutdown

The image shows the main interface of the simulator. It includes a memory dump on the left, a central control panel with buttons for Start, Stop, Step, and Run, and a right-hand panel displaying various registers and I/O ports. The registers shown include TIMERS 0 & 1, INTERRUPTS, and various I/O ports like P0, P1, P2, P3, PCON, and SCON. The status bar at the bottom indicates 'Invoke hex editor with program code'.

HEX	DEC	BIN	OCT	CHAR
A:	00	0	00000000	0
B:	00	0	00000000	0

TH1	TL1	TH0	TL0	T1	T0	TCON	TMOD
00	00	00	00	0	0	00	00

IE	EA	ET2	ES	ET1	EX1	ET0	EX0	HEX
00	00	00	00	00	00	00	00	00

P0	P1	P2	P3	PCON	SCON
11111111	11111111	11111111	11111111	SMOD - - - GF1 GF0 PD IDL	SM0 SM1 SM2 REN TB8 RB8 TI RI

Interface do Simulador

Memória RAM interna
- 00h até 7Fh (endereçamento direto)
- 80h até FFh (endereçamento indireto)

Timers

Portas

Clock

Stack
Pointer

The screenshot shows a simulator interface with several panels. On the left is a memory dump table with columns x0 through x7 and rows 00 through 50. Below it are registers A and B, and PSW. In the center are TIMERS 0 & 1 and INTERRUPTS sections. On the right are I/O PORTS (P0-P3) and simulation parameters like DPH, DPL, Clock, PC, and SCON. A 'Time' input field is also present.

x0	x1	x2	x3	x4	x5	x6	x7
00	00	00	00	00	00	00	00
08	00	00	00	00	00	00	00
10	00	00	00	00	00	00	00
18	00	00	00	00	00	00	00
20	00	00	00	00	00	00	00
28	00	00	00	00	00	00	00
30	00	00	00	00	00	00	00
38	00	00	00	00	00	00	00
40	00	00	00	00	00	00	00
48	00	00	00	00	00	00	00
50	00	00	00	00	00	00	00

Registers: A: 00 0, B: 00 0, PSW: C AC F0 RS1 RS0 OV - P, R7-R0: 00 00 00 00 00 00 00 00

Timers: TH1 TL1 TH0 TL0 T1 T0 TCON TMOD

Interrupts: IE: EA - ET2 ES ET1 EX1 ET0 EX0, IP: - - PT2 PS PT1 PX1 PT0 PX0

Ports: P0: 11111111 FF, P1: 11111111 FF, P2: 11111111 FF, P3: 11111111 FF

Simulation Parameters: DPH: 00, DPL: 00, Clock: 12000, PC: 00, SCON: SM0 SM1 SM2 REN TB8 RB8 TI RI

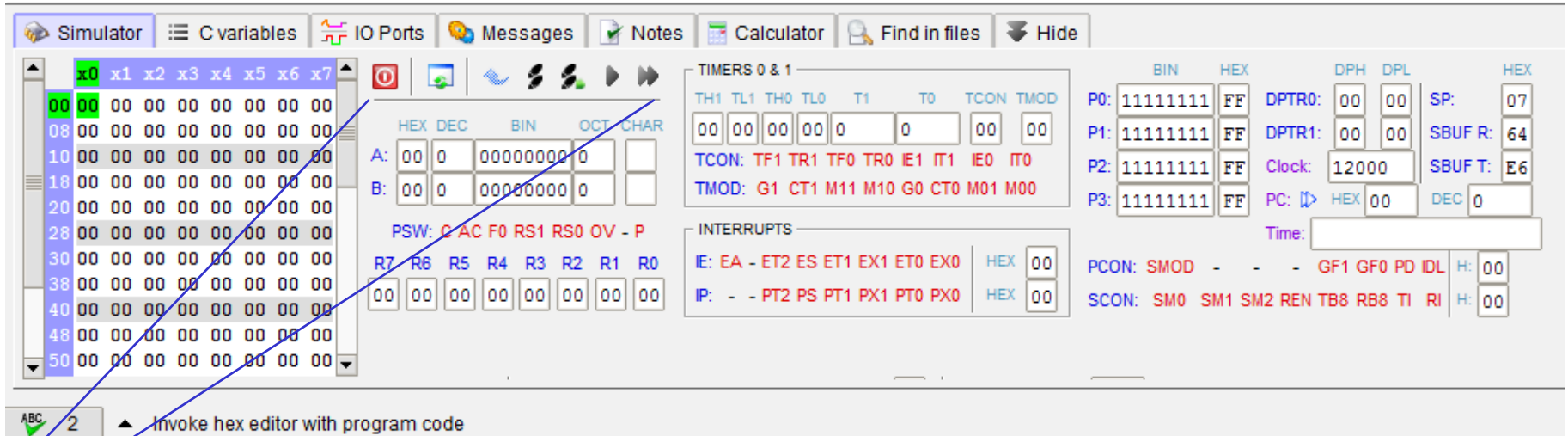
R7-R0








Interrupções

Tempo de
simulação

Acumulador

Interface do Simulador



-  Inicia/para simulação
-  Reset (PC=0). Mantém valores na memória
-  Volta 1 instrução
-  Executa 1 instrução
-  Executa até sair da linha atual*
-  Executa continuamente (lento)
-  Executa continuamente (rápido, não atualiza os valores até pausar)

* Exemplo: DJNZ A, \$;Decrementa o acumulador até este valer 0

É possível adicionar breakpoints clicando no número de uma linha do código. Deste modo a execução irá pausar antes de executar a instrução da linha selecionada.

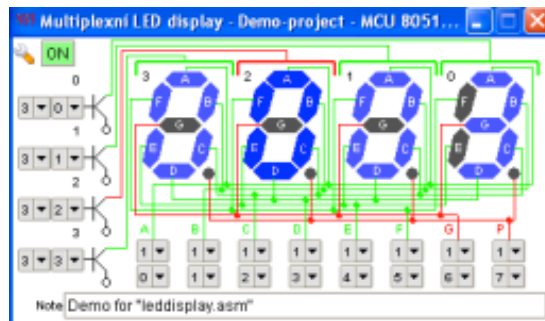
```
1      ORG      0
2
3      MOV      00h, #10h ;(Direto / Imediato)
4      MOV      00h, 10h  ;(Direto / Direto)
5      MOV      R0, #10h  ;(Rn / Imediato)
6      MOV      R1, #10h  ;(Rn / Imediato)
7      MOV      A, #10h   ;(Imediato)
8
9      END
```

Útil para programas que tomam tempo com parada garantida ao atingir aquele ponto.

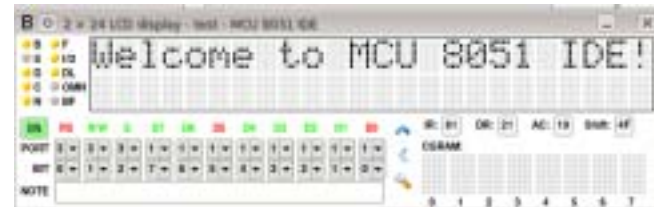
Ferramentas disponíveis que permitem simular várias aplicações



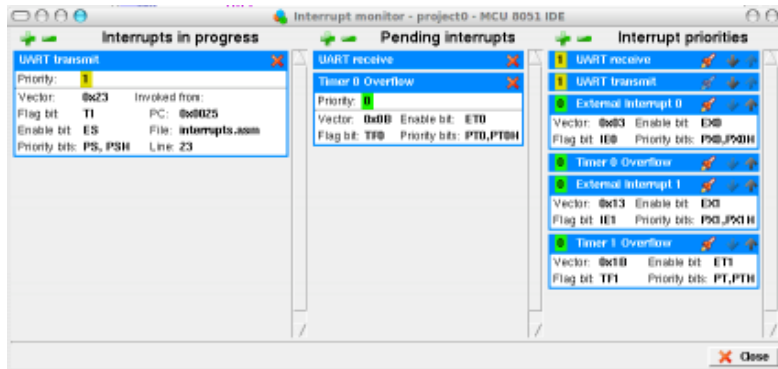
Display 8 segmentos



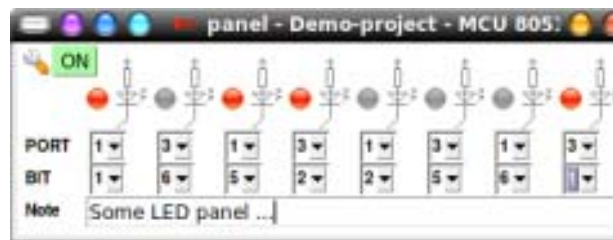
Display multiplexado



Display LCD



Monitor de interrupções



Painel de chaves/leds



Matriz de leds

Entre outras ferramentas:

Calculadora para conversão de bases, calculadora para rotinas de atrasos, timers, etc.

Fim de um programa

```
1  ORG      0
2
3  MOV      00h, #10h ;(Direto / Imediato)
4  MOV      00h, 10h  ;(Direto / Direto)
5  MOV      R0, #10h  ;(Rn / Imediato)
6  MOV      R1, #10h  ;(Rn / Imediato)
7  MOV      A, #10h   ;(Imediato)
8
9  END
```

- Fim físico (diretiva END) **não significa a parada do programa!**
- Indica ao compilador que não há mais código após aquele ponto.
- O processador continuará executando instruções desconhecidas presentes após o fim físico do programa.
- É de responsabilidade do programador criar um fim lógico para o programa.

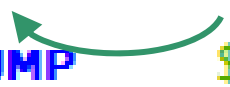
Fim lógico de um programa

- Sempre feito com um loop infinito
- Pode ser um loop que não faça nada ou que realize o mesmo procedimento várias vezes
- Necessário para impedir a execução de código desconhecido

Fim lógico de um programa

Programa fica parado sempre na mesma instrução

```
1  ORG      0
2
3  MOV      00h, #10h ;(Direto / Imediato)
4  MOV      00h, 10h  ;(Direto / Direto)
5  MOV      R0, #10h  ;(Rn / Imediato)
6  MOV      R1, #10h  ;(Rn / Imediato)
7  MOV      A, #10h   ;(Imediato)
8
9  SJMP     $          ;Fim lógico do programa
10
11  END
```



Programa não sairá desta posição.

Uma interrupção pode tirar o programa desta instrução, mas voltará após esta ser finalizada

Fim lógico de um programa

Programa em loop contínuo:

```
1 ;Programa para debounce de uma chave
2 ;Necessário para remover ruídos da chave ao pressioná-la que pode
3 ;causar mais de uma contagem
4
5     ORG     0           ;Início do programa (PC=0)
6
7     CLR     A           ;Inicializa o contador em zero
8 LOOP: JB     P1.0, $    ;Aguarda por nível lógico 0 (Chave aberta)
9         INC     A           ;Incrementa o contador
10        MOV     P2, A     ;Coloca o valor do contador em P2
11        LCALL   ATRASO    ;Chama rotina de atraso
12        JNB    P1.0, $    ;Aguarda por nível lógico 1(chave fechada)
13        SJMP   LOOP      ;Retorna para nova contagem de pulso
14
15 ATRASO: MOV     R0, #0FFh ;valor de atraso para debounce (aproximadamente 500us)
16         DJNZ   R0, $     ;Decrementa até zero
17         RET
18
19     END               ;Fim do programa
```

Processador nunca executará bytes desconhecidos pois sempre volta para uma parte conhecida do programa