SEL-433 APLICAÇÕES DE MICROPROCESSADORES I

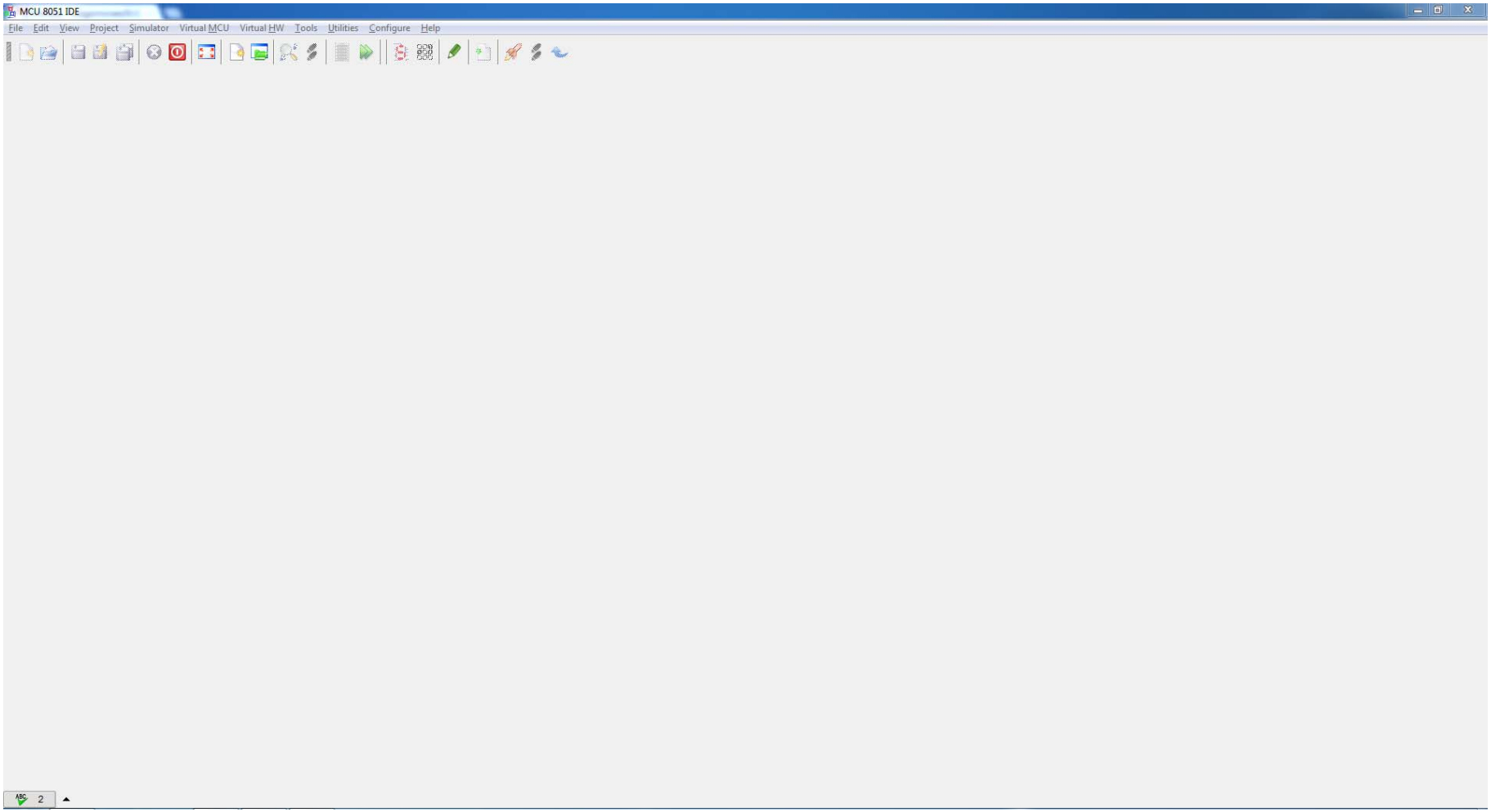
Ambiente de desenvolvimento de Programação Assembly

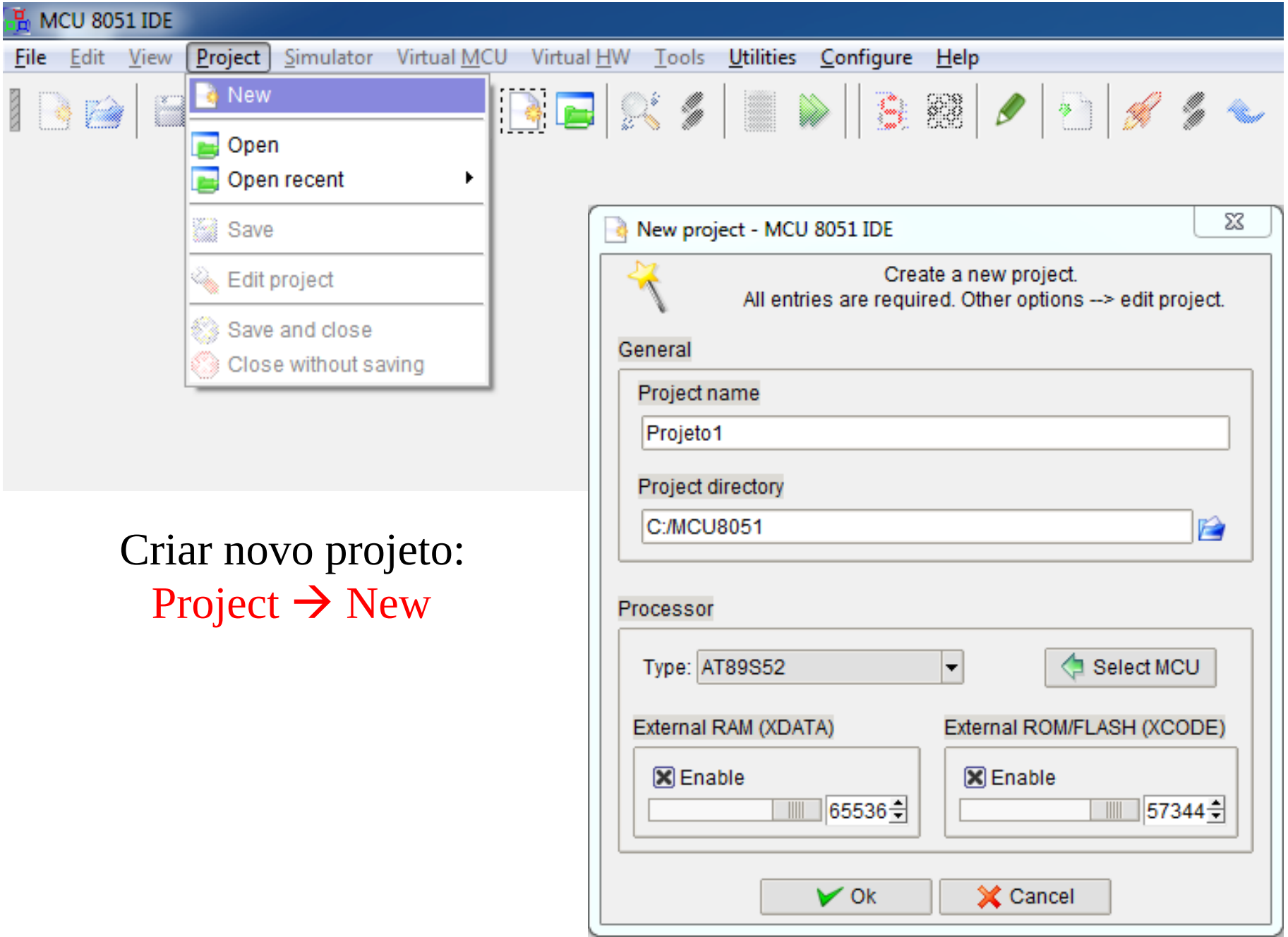
MCU 8051 IDE

http://mcu8051ide.sourceforge.net/

Prof. Evandro L. L. Rodrigues

Tela inicial



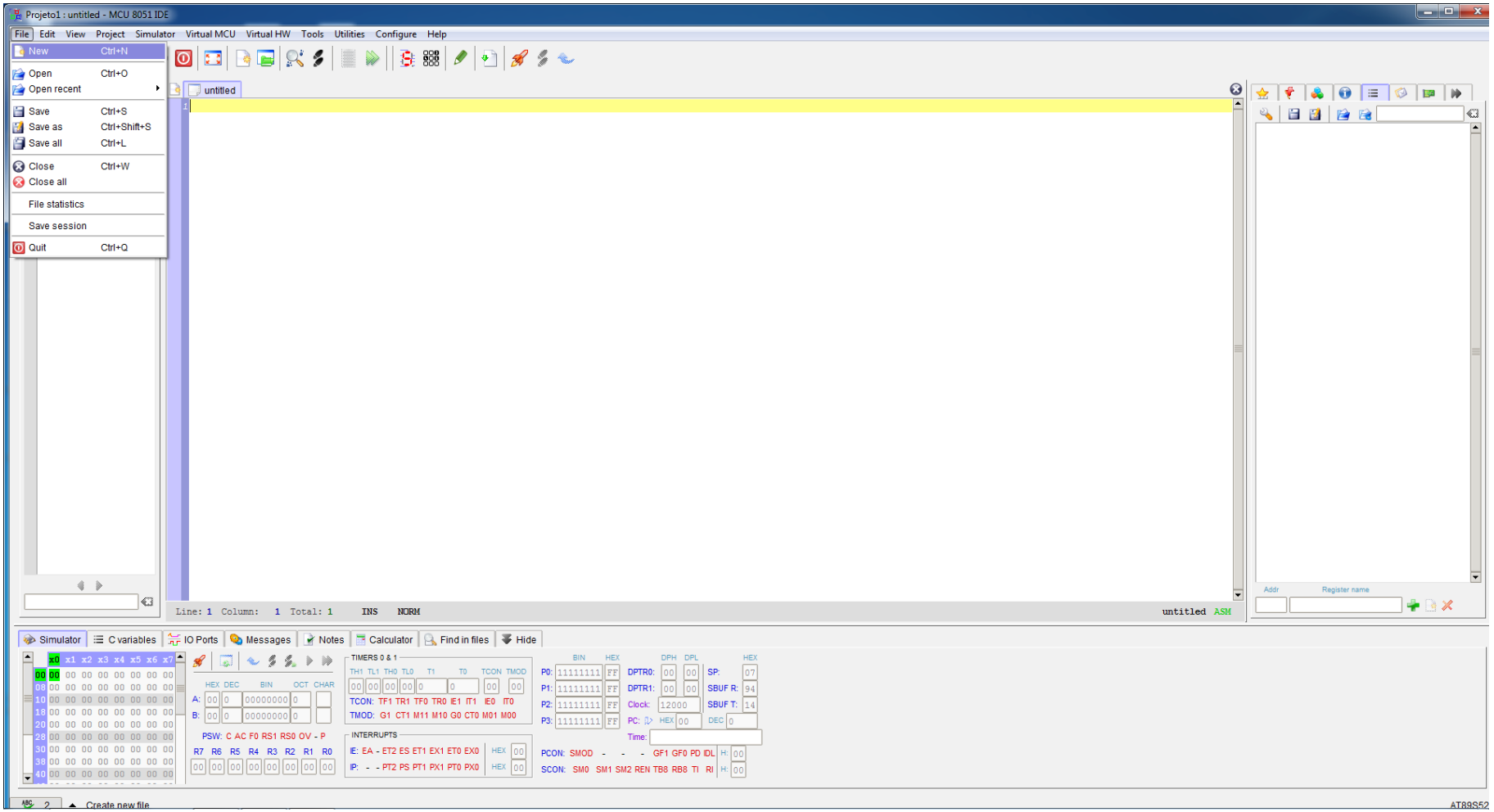


Criar novo projeto:

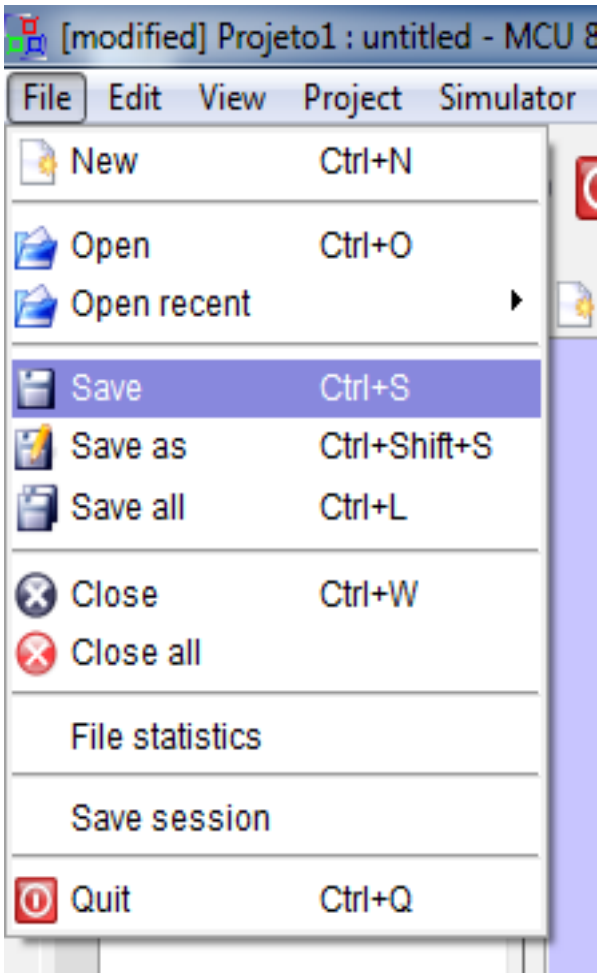
Project → New

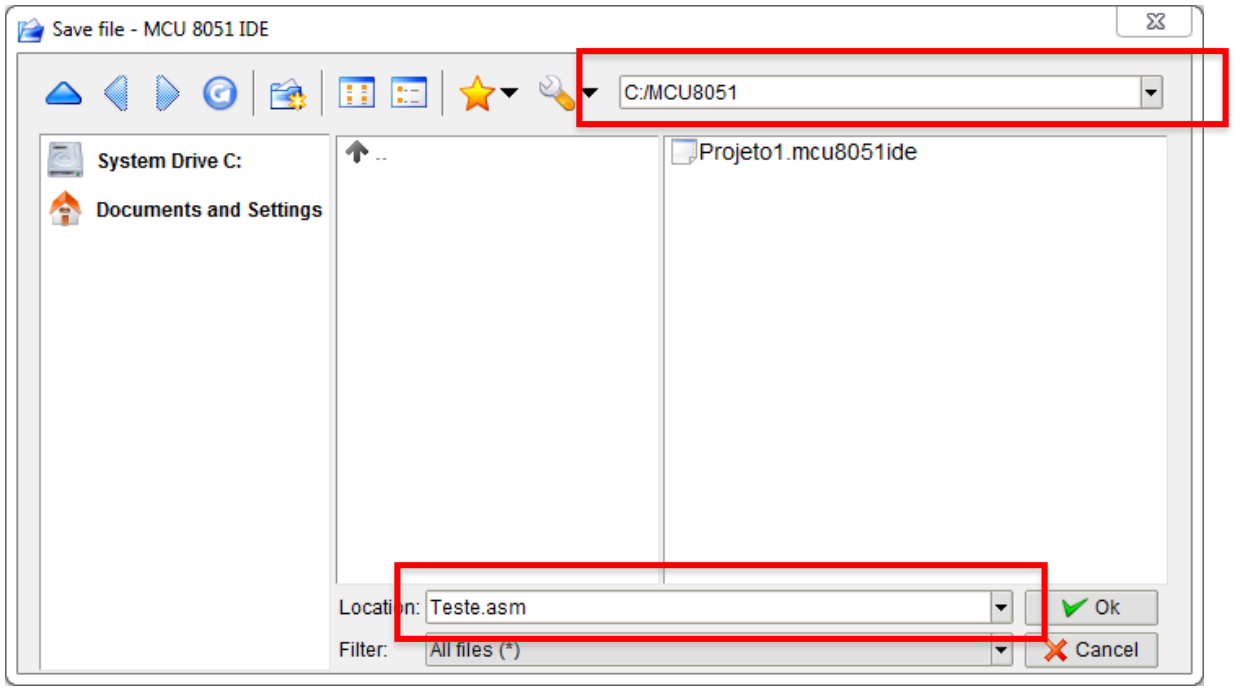
Criar novo programa em Assembly

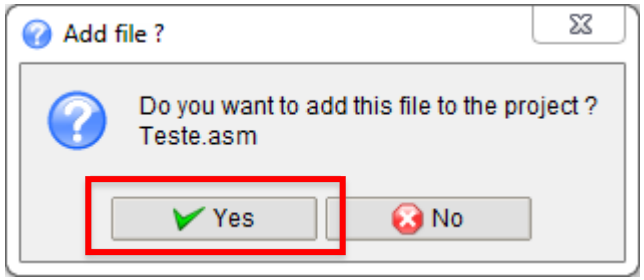
File → New



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>

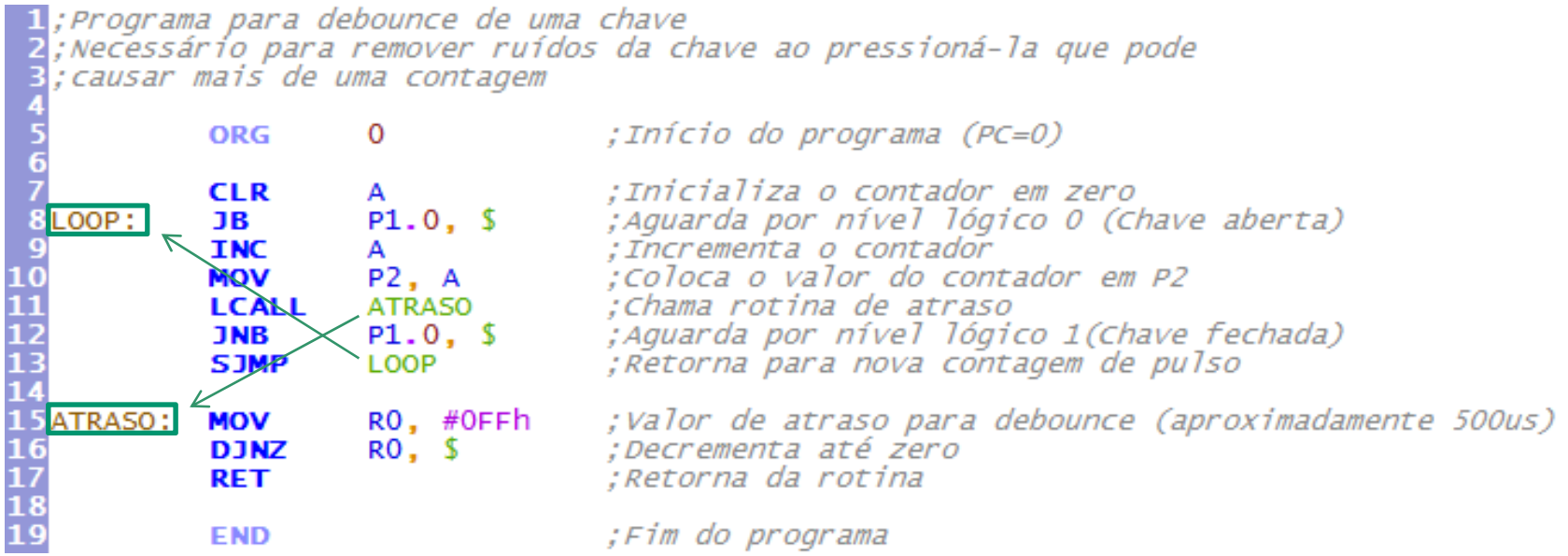
Rótulo:

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

```
7          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
```

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

Formato de um programa

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

Operandos:

- Especifica o dado a ser operado pela instrução

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

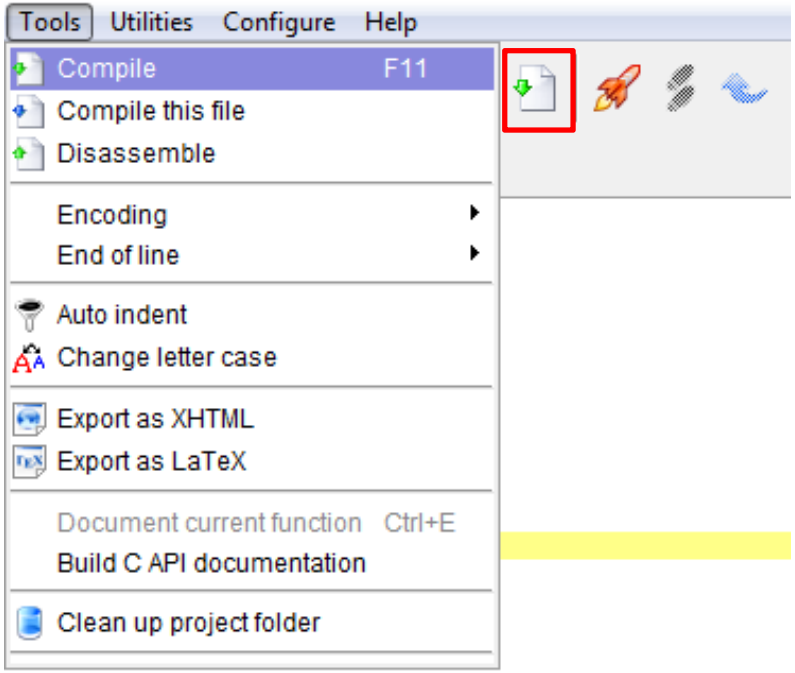
Atribui o valor ‘value’para uma variável ‘var’

Exemplo:

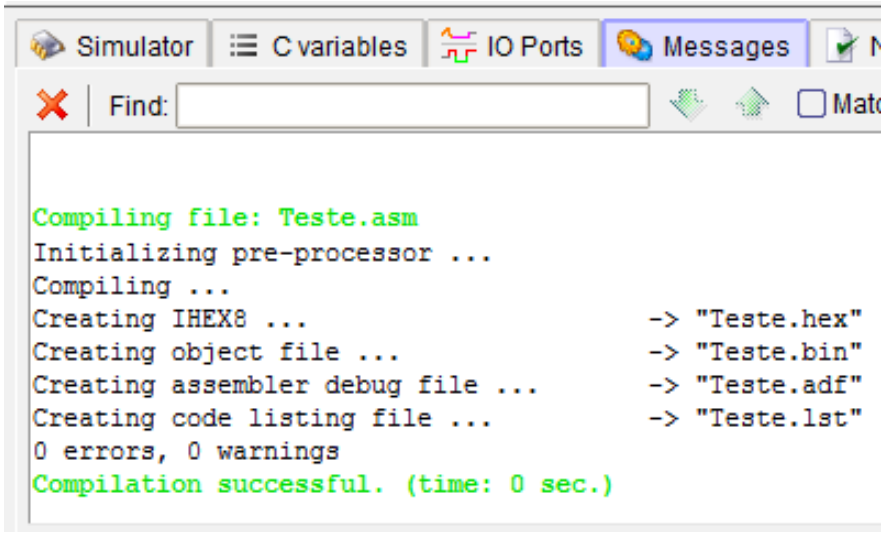
```
Controle    EQU     10h             ;atribui 10h para Controle
            ORG 0
            MOV     A, #Controle    ;Acumulador = 10h
```

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

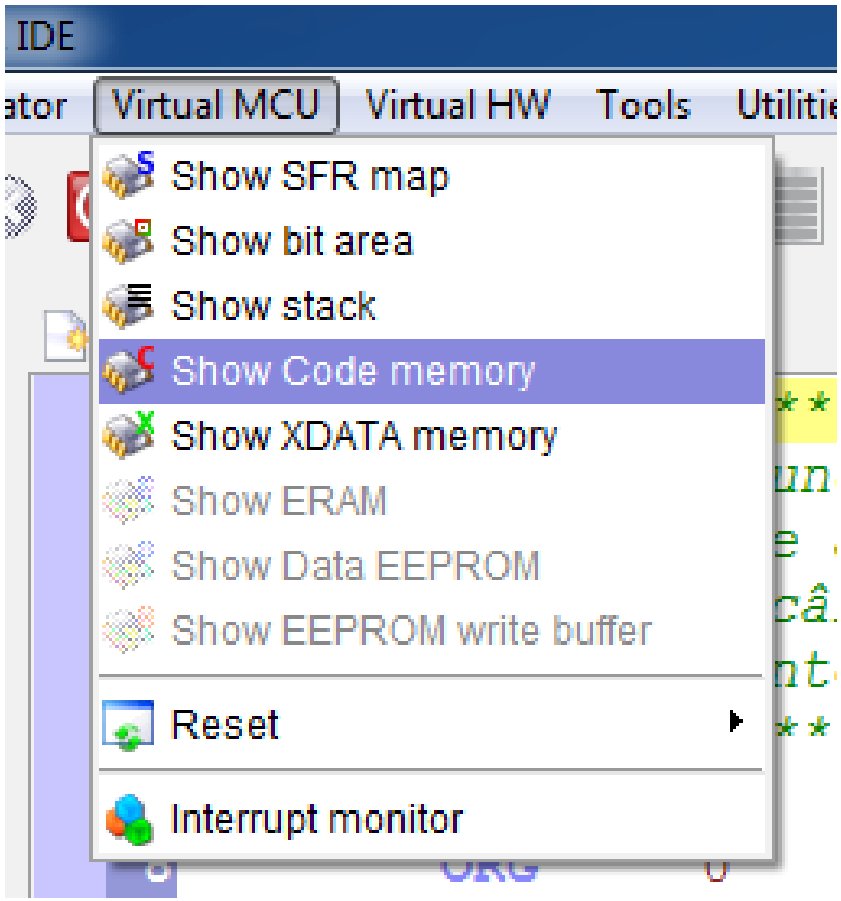
Compilação do código fonte

Programa Fonte (Assembly) Formato Texto → Compilador → Programa Objeto (Código Compilado) Formato Binário

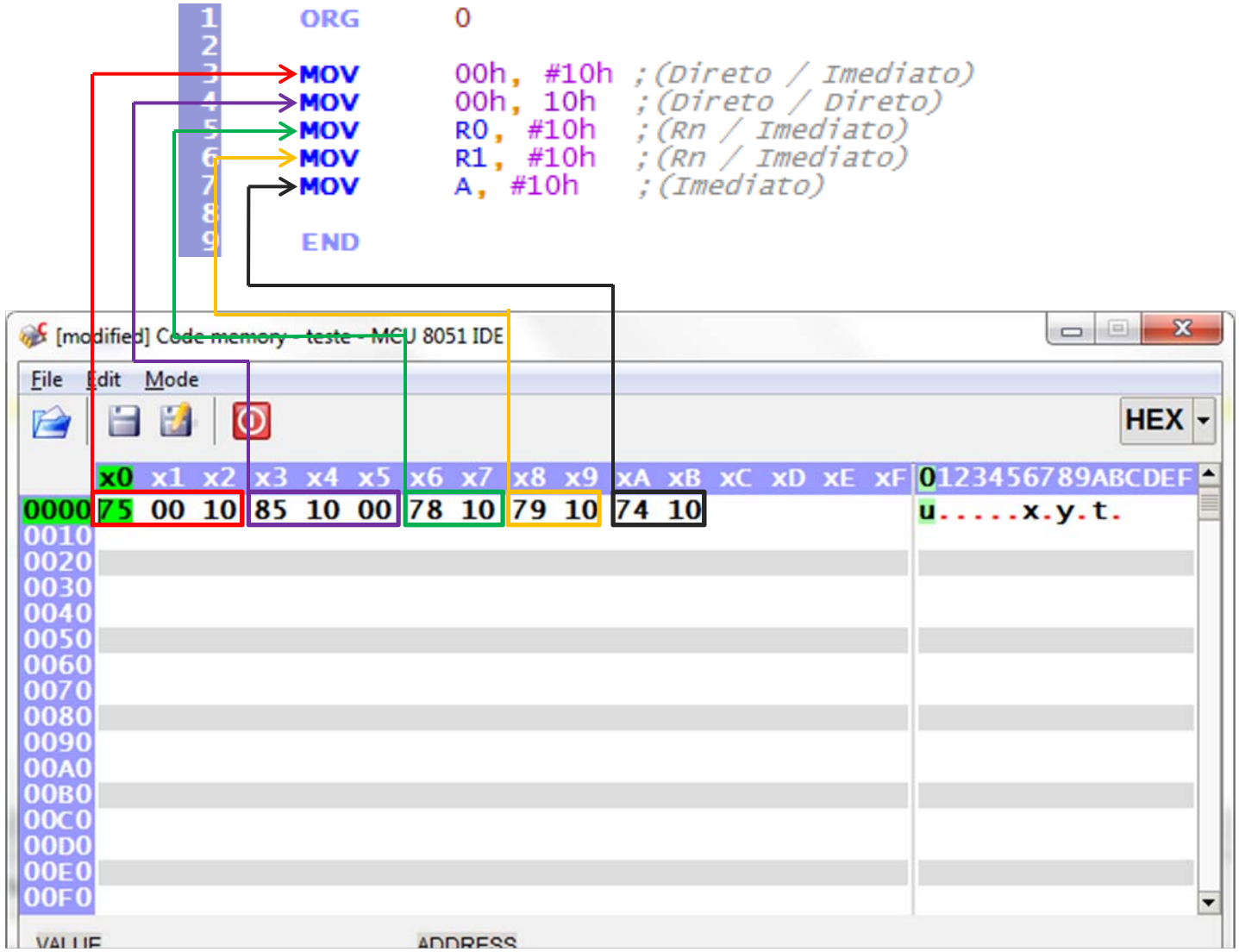
```
1  ;Programa para debounce de uma chave
2  ;Necessário para remover ruídos da chave ao pressioná-la que pode
3  ;causar mais de uma contagem
4
5           ORG     0               ;Início do programa (PC=0)
6
7           CLR     A               ;Inicializa o contador em zero
8  LOOP:    JB      P1.0, $         ;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
```

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



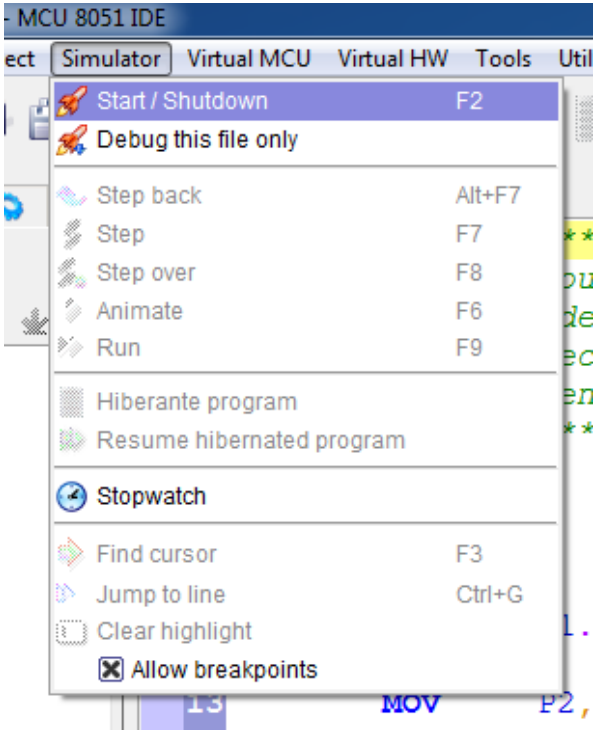
Virtual MCU → Show Code memory

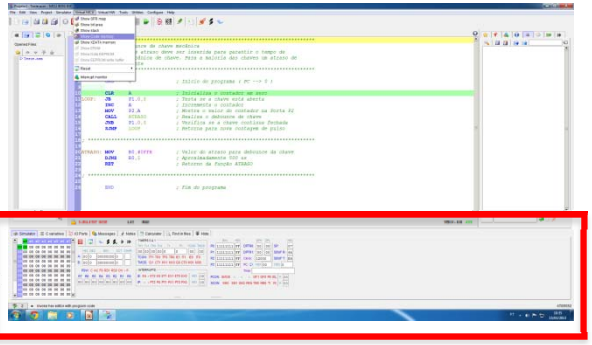


http://www.keil.com/support/man/docs/is51/is51_instructions.htm

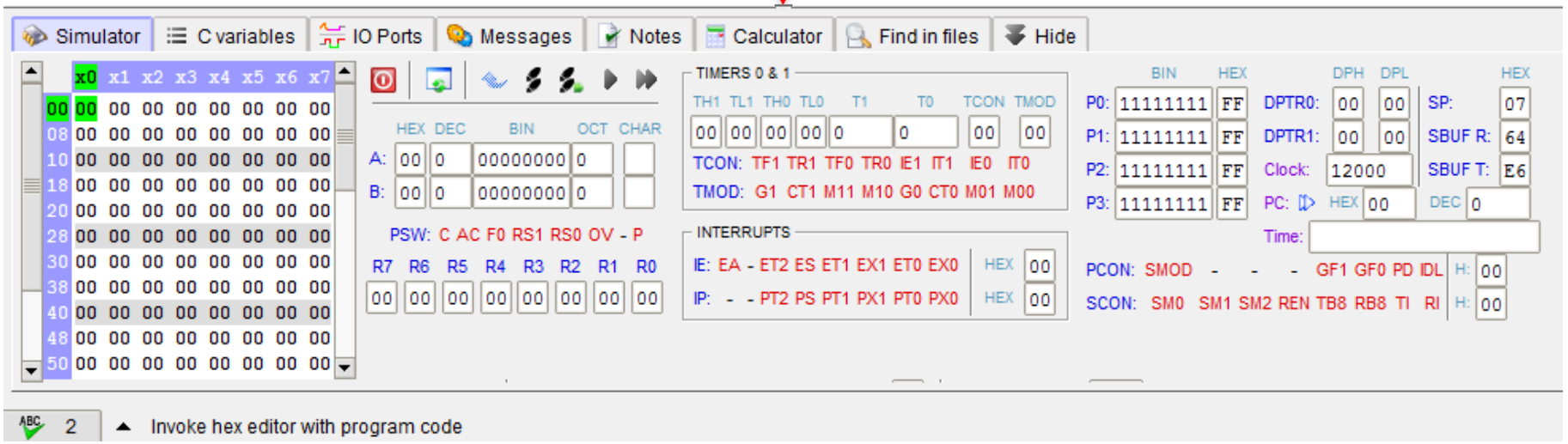
Simulador:
Pode ser utilizado como teste de mesa

Exibe os valores atuais de:
registradores, acumulador, portas, memória etc

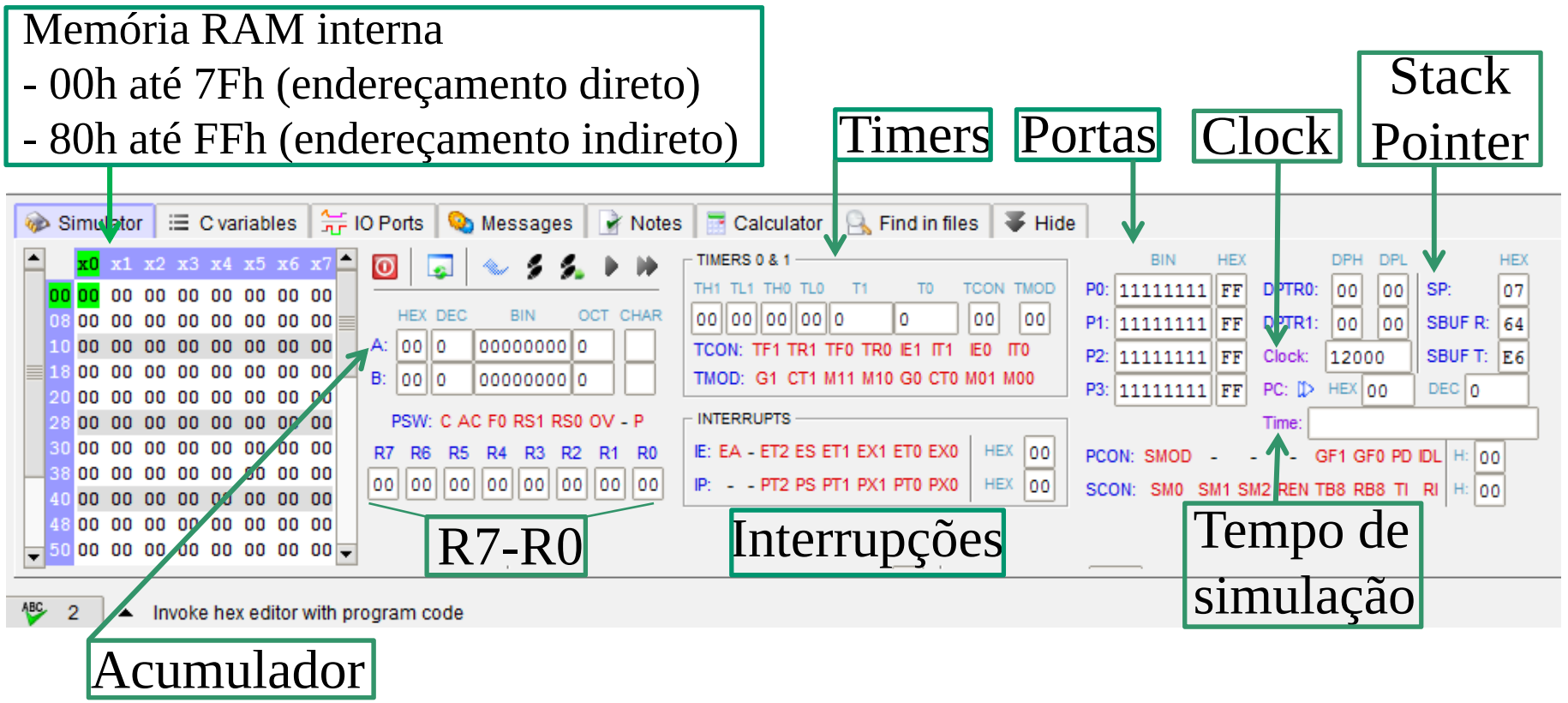




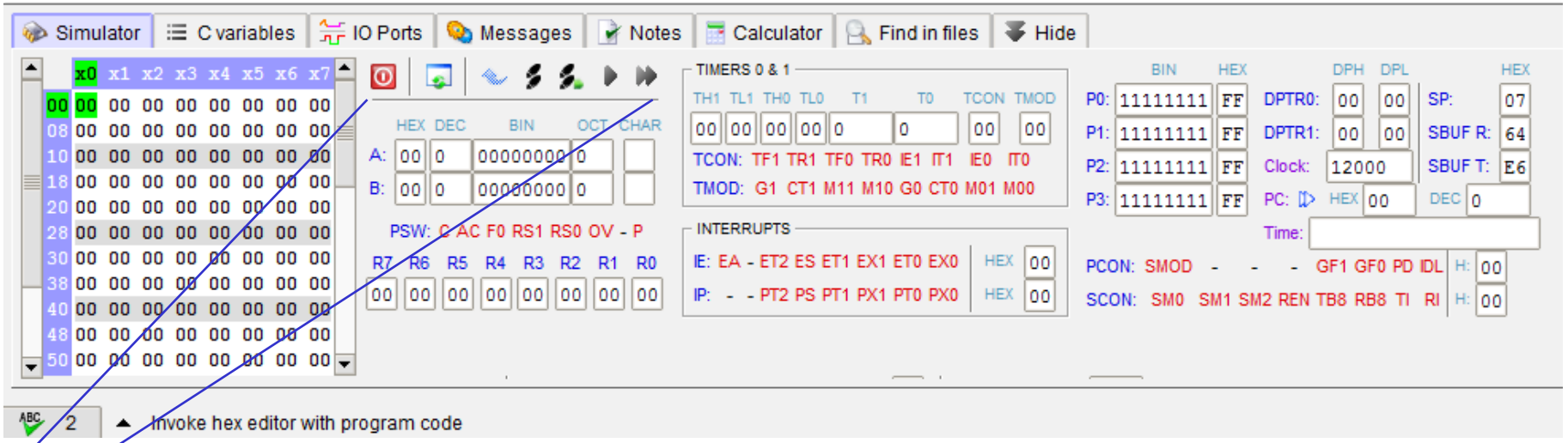
Simulator → Start/Shutdown



Interface do Simulador



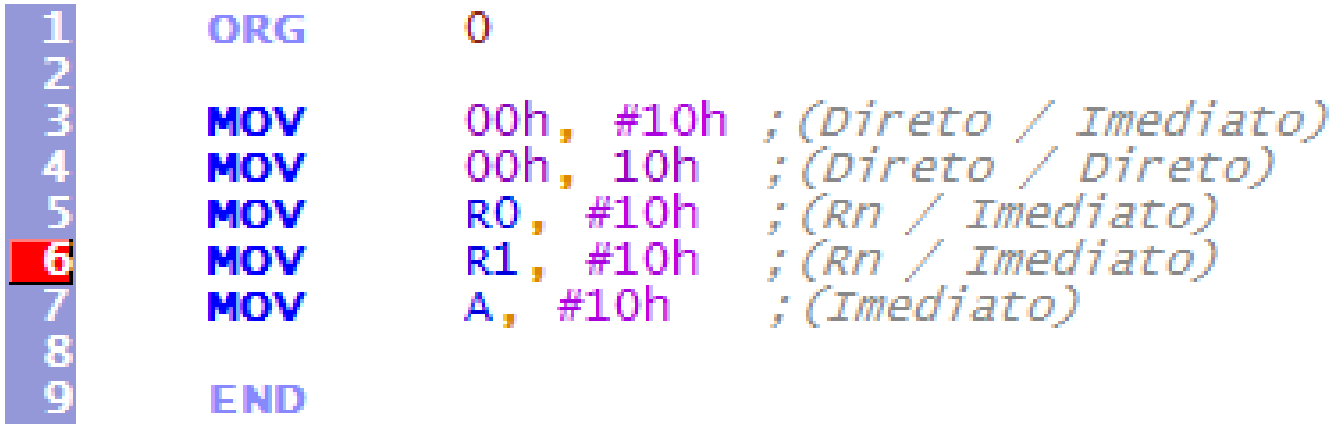
Interface do Simulador



- 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)

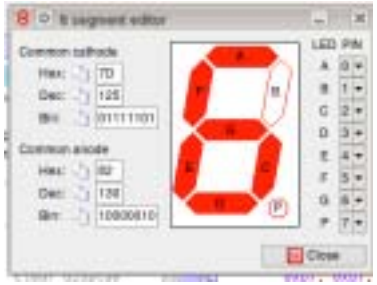
* Exemplo: DJNZ A, $;Decrementa o acumulador até este valer 0

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.

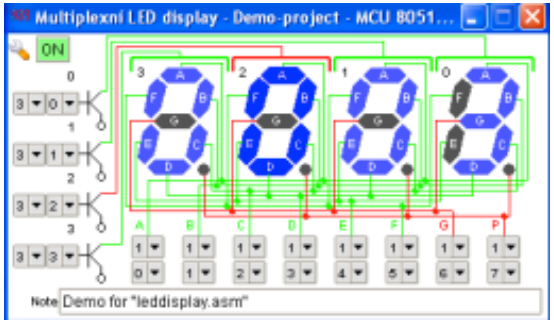


Útil para programas que tomam tempo com parada garantida ao atingir aquele ponto.

Ferramentas disponíveis que permitem simular várias aplicações

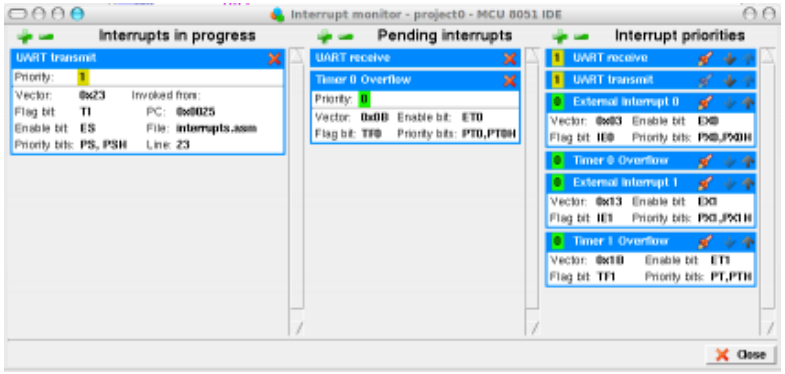


Display 8 segmentos

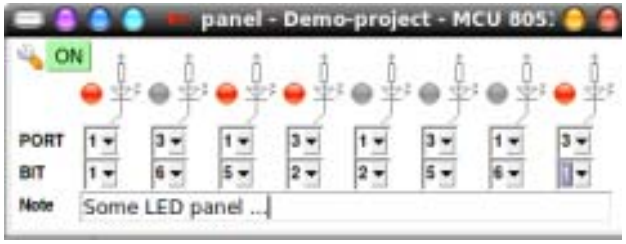


Display multiplexado

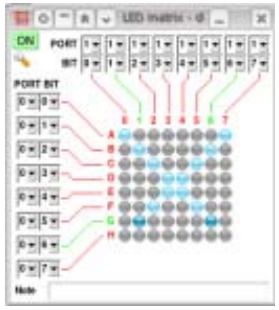
Display LCD



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
```

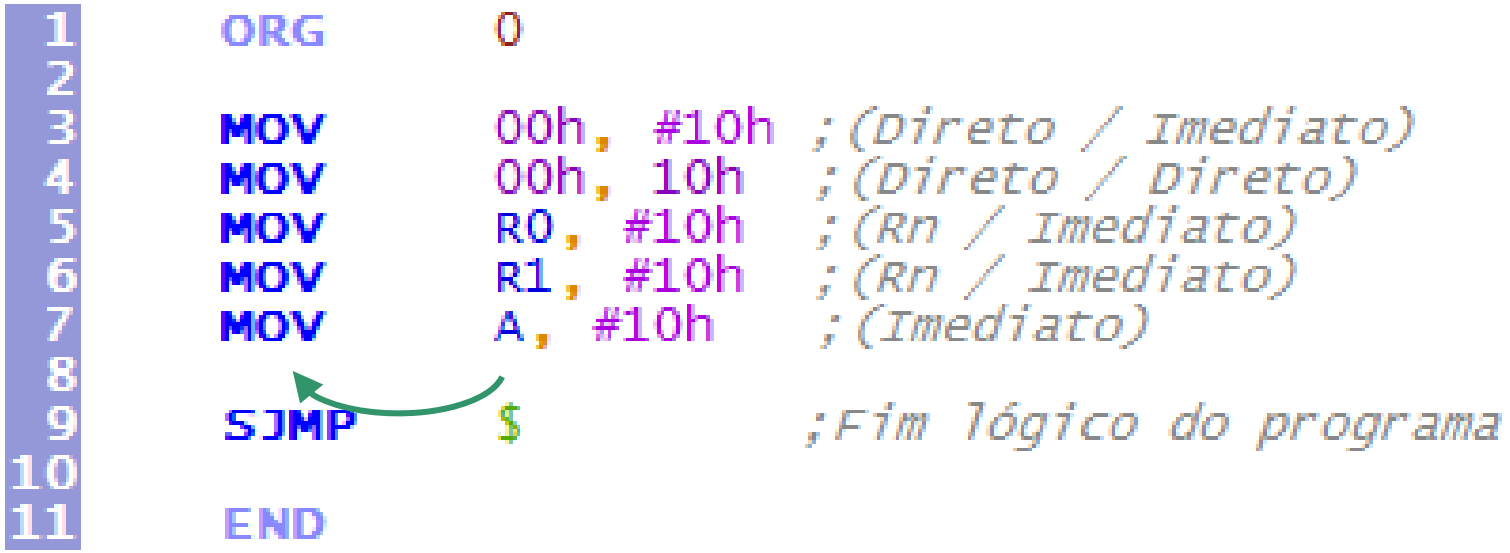
- Fim físico (diretiva END) não significa a parada do programa!
- Indica ao compilador que não há mais código após aquele ponto.
- O processador continuará executando instruções desconhecidas presentes após o fim físico do programa.
- É de responsabilidade do programador criar um fim lógico para o programa.

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                     ;Retorna da rotina
18
19          END                     ;Fim do programa
```

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