

# Microcontroladores

## Conjunto de Instruções do 8051 - Lab

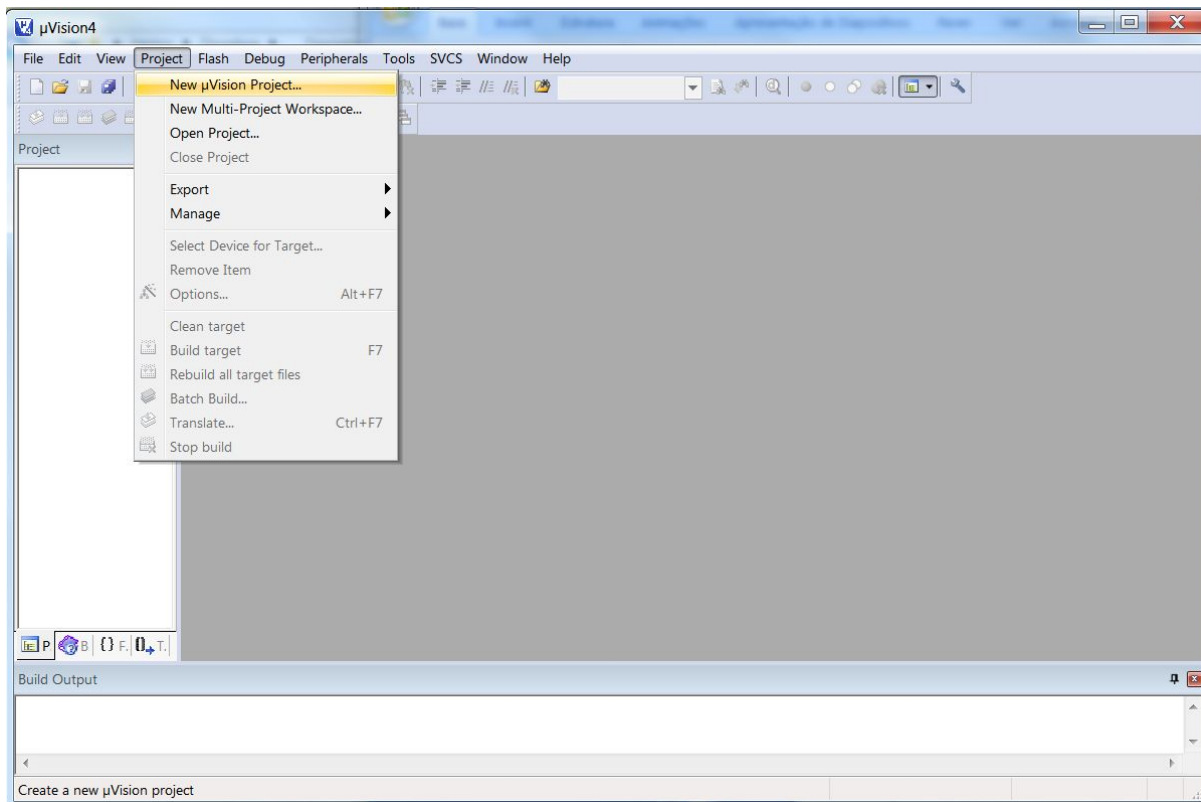
Prof. Guilherme Peron

Prof. Ronnier Rohrich

Prof. Rubão

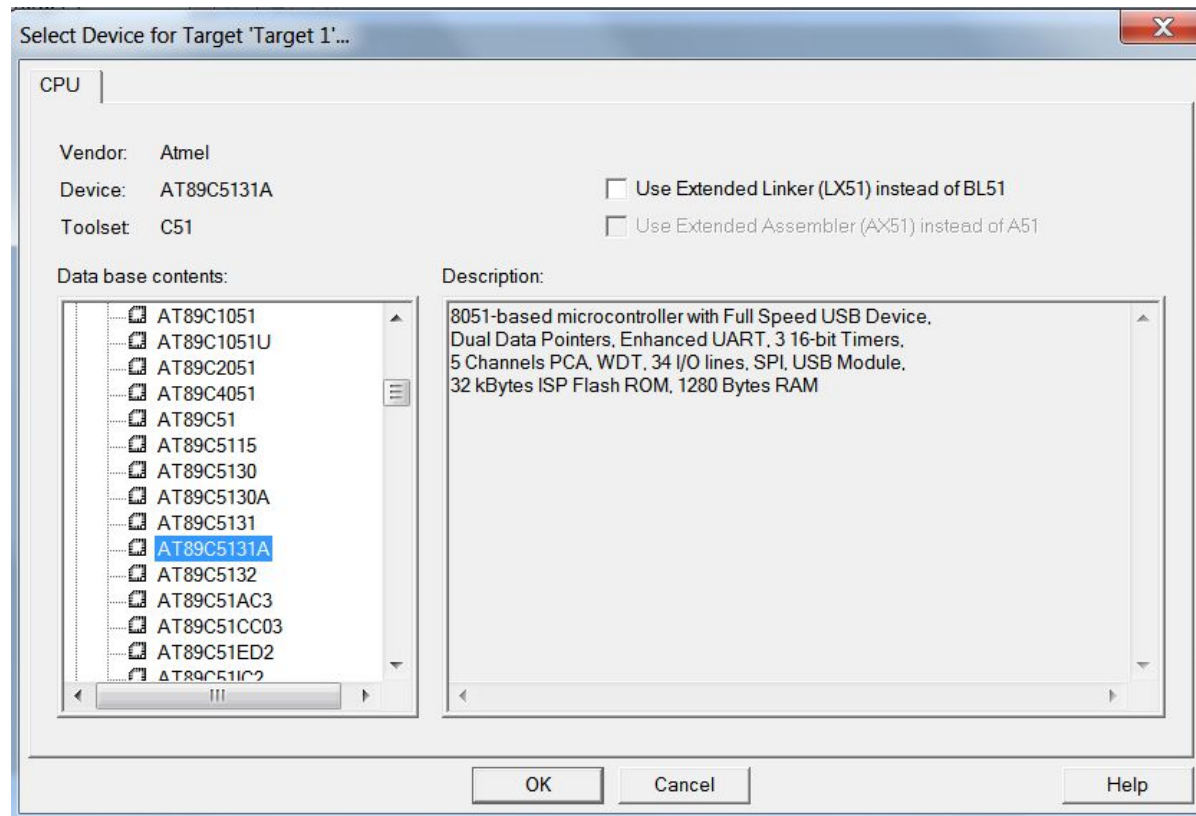
# Criação de um projeto 8051 asm no Keil

- Passo 1: criar um novo projeto e salvar em uma pasta nova, **Project / New  $\mu$ Vision Project**



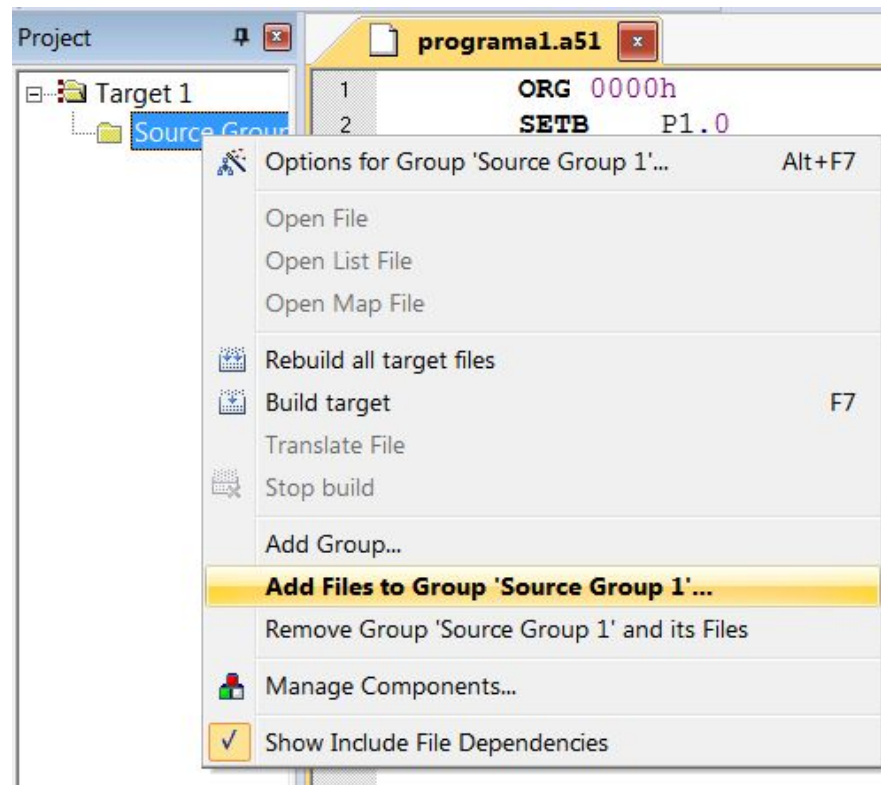
# Criação de um projeto 8051 asm no Keil

- Passo 2: seleccionar o dispositivo **Atmel / AT89C5131A**
  - Responder (**NO**) para a pergunta **Copy standard 8051 startup code to project folder and add file to project ?**



# Criação de um projeto 8051 asm no Keil

- 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.

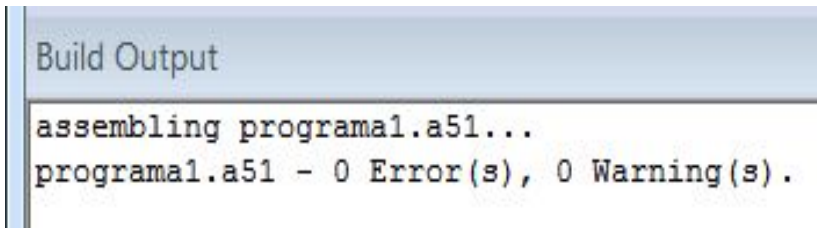


# Criação de um projeto 8051 asm no Keil

- Compilar o programa:



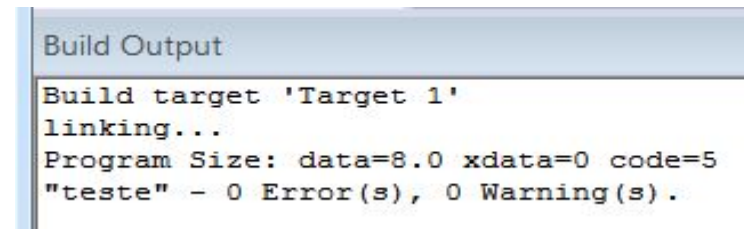
- A janela de saída apresenta o resultado, erros e warnings:



- Linkar o programa:

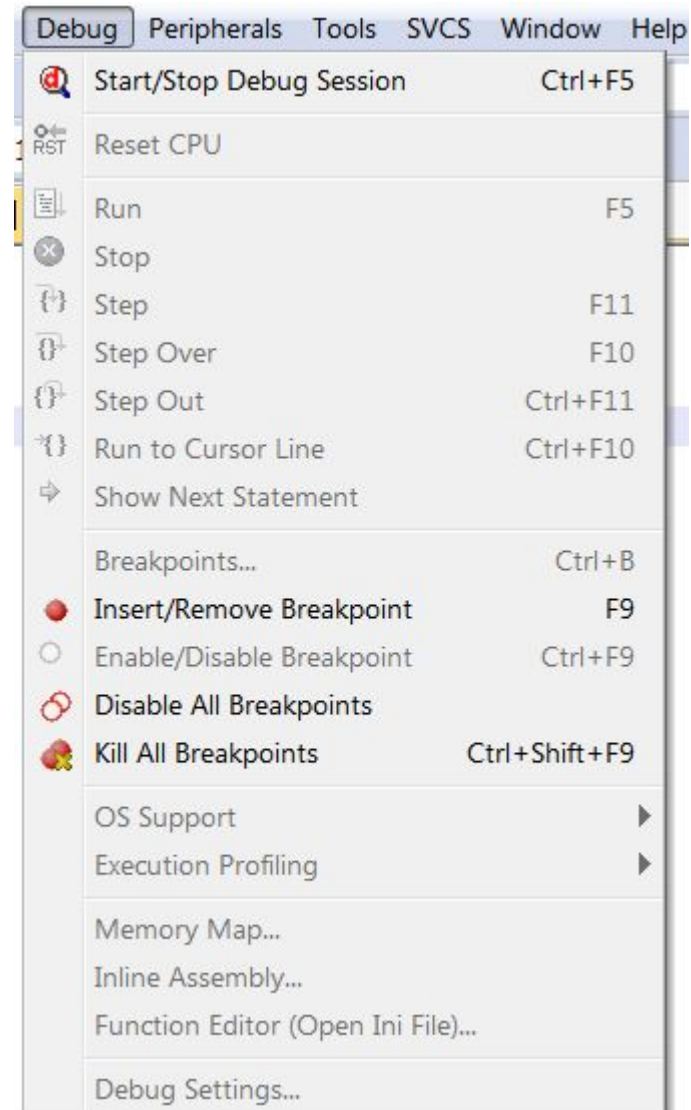


- A janela de saída apresenta o resultado:



# Criação de um projeto 8051 asm no Keil

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





# Resumo das instruções assembly 8051

## 8051 Instruction Set Summary

<b>Rn</b>	Register R7-R0 of the currently selected Register Bank.
<b>Data</b>	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)].
<b>@Ri</b>	8-bit Internal Data RAM location (0-255) addressed indirectly through register R1 or R0.
<b>#data</b>	8-bit constant included in instruction.
<b>#data16</b>	16-bit constant included in instruction.
<b>addr16</b>	16-bit destination address. Used by LCALL and LJMP. A branch can be anywhere within the 64k byte Program Memory address space.
<b>addr11</b>	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.
<b>rel</b>	Signed (two'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.
<b>bit</b>	Direct Addressed bit in Internal Data RAM or Special Function Register.

Instruction	Flag			Instruction	Flag		
	C	OV	AC		C	OV	AC
ADD	X	X	X	CLR C	O		
ADDC	X	X	X	CPL C	X		
SUBB	X	X	X	ANL C,bit	X		
MUL	O	X		ANL C,bit	X		
DIV	O	X		ORL C,bit	X		
DA	X	X		ORL C,bit	X		
RRC	X			MOV C,bit	X		
RLC	X			CJNE	X		
SETB C	1						

Note that operations on SFR byte address 200 or bit addresses 209-215 (i.e. the PSW or bits in the PSW) will also affect flag settings.

Mnemonic	Description	Byte	Cycle
<b>Arithmetic operations</b>			
ADD A,Rn	Add register to accumulator	1	1
ADD A,direct	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,direct	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,direct	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 direct	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 direct	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 hij], [A lo]	1	4
DIV AB	Divide A by B -> A= result, B= remainder	1	4
DA	Decimal adjust accumulator	1	1
CLR A	Clear accumulator	1	1

This paper was created by Štěpán Matějka alias Mates for anybody who needs it. Mates, Prague - Czech Republic 1998, 2002.

Mnemonic	Description	Byte	Cycle
CPL A	Complement accumulator	1	1
RL A	Rotate accumulator left	1	1
RLC A	Rotate accumulator left through carry	1	1
RR A	Rotate accumulator right	1	1
RRC A	Rotate accumulator right through carry	1	1
SWAP A	Swap nibbles within the accumulator	1	1

### Logic operations

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

### Boolean variable manipulation

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

### Program and machine control

ACALL addr11	Absolute subroutines call	2	2
LCALL addr16	Long subroutines call	3	2
RET	Return from subroutines	1	2
RETI	Return from interrupt	1	2
AJMP addr11	Absolute jump	2	2
LJMP addr16	Long jump	3	2
SJMP rel	Short jump (relative address)	2	2
JMP @A+DPTR	Jump indirect relative to the DPTR	1	2
JZ rel	Jump if accumulator is zero	2	2
JNZ rel	Jump if accumulator is not zero	2	2
JC rel	Jump if carry flag is set	2	2
JNC rel	Jump if carry flag is not set	2	2
JB bit,rel	Jump if bit is set	3	2
JNB bit,rel	Jump if bit is not set	3	2
JBC bit,rel	Jump if direct bit is set and clear bit	3	2
CJNE A,direct,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 reg. and jump if not equal	3	2
CJNE @Rn,#data,rel	Compare immediate to ind. and jump if not equal	3	2
DJNZ Rn,rel	Decrement register and jump if not zero	2	2
DJNZ direct,rel	Decrement direct byte and jump if not zero	3	2
NOP	No operation	1	1

### Data transfer

MOV A,Rn	Move register to accumulator	1	1
MOV A,direct	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,direct	Move direct byte to register	2	2
MOV Rn,#data	Move immediate data to register	2	1
MOV direct,A	Move accumulator to direct byte	2	1
MOV direct,Rn	Move register to direct byte	2	2
MOV direct,direct	Move direct byte to direct byte	3	2
MOV direct,@Ri	Move indirect RAM to direct byte	2	2
MOV direct,#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
MOVC A,@Ri	Move external RAM (8-bit addr.) to A	1	2
MOVC A,@DPTR	Move external RAM (16-bit addr.) to A	1	2
MOVC @Ri,A	Move A to external RAM (8-bit addr.)	1	2
MOVC @DPTR,A	Move A to external RAM (16-bit addr.)	1	2
PUSH direct	Push direct byte onto stack	2	2
POP direct	Pop direct byte from stack	2	2
XCH A,Rn	Exchange register to accumulator	1	1
XCH A,direct	Exchange direct byte to accumulator	2	1
XCH A,@Ri	Exchange indirect RAM to accumulator	1	1
XCHD A,@Ri	Exchange low-order nibble indir. RAM with A	1	1

\*) MOV A,ACC is not a valid instruction

jne A,#data @ (jump if A != data)	cjne A,#data @ add A,#low(-data) or @ ne: jnc A,#(data+1) ne
je A,#data @ (jump if A == data)	jz @ ne: jnc A,#(data+1) ne
ja,jnbe A,#data @ (jump if A > data)	add A,#low(-data-1) or @ ne: jnc A,#(data+1) ne
jae,jnb A,#data @ (jump if A >= data)	add A,#low(-data) or @ ne: jnc A,#(data+1) ne
jb,jnae A,#data @ (jump if A < data)	add jnc A,#low(-data) or @ ne: jnc A,#(data+1) ne
jbe,jna A,#data @ (jump if A <= data)	add jnc A,#low(-data-1) or @ ne: jnc A,#(data+1) ne
switch A <=, > #data (no A modification)	cjne A,#data ne ... : execute code if A==data ne: jc is_below : jump if A<data jnc is_above : jump if A>data or exec. code



Enjoy It! Mates

# Exemplo 0 - Endereçamento

- Objetivo: analisar as diferentes formas de endereçamento

```
1 ORG 0000h
2
3 ; endereçamento por registrador
4 ; o conteúdo do registrador R0
5 ; é copiado para o acumulador
6 MOV A, R0
7
8 ; endereçamento direto
9 ; endereço do dado é carregado
10 ; diretamente na instrução
11 MOV A, 30h
12
13 ; endereçamento indireto
14 ; o dado é acessado pelo endereço
15 ; armazenado nos registradores R0 e R1
16 MOV R0, #44h
17 MOV A, @R0
18
19
20 ; endereçamento imediato
21 ; o valor do operando está na instrução
22 MOV A, #25
23
24
25
26
27
28
29
30
31
32
```

```
24 ; endereçamento relativo
25 ; permite um salto para um endereço
26 ; apenas saltos curtos de -128 a 127
27
28 linhadecima:
29 SJMP pulaum
30 NOP
31 pulaum:
32 SJMP linhadecima
33
34 ; endereçamento absoluto
35 ; permite o salto para um endereço
36 ; dentro da mesma página de 2kbytes
37 AJMP 03h
38
39 ; endereçamento longo
40 ; permite o salto para qualquer endereço
41 ; dentro da faixa de 64 KB
42 LJMP 77ACh
43
44 ; endereçamento indexado
45 ; endereçamento direto para acesso
46 MOVC A, @A+DPTR ;A=(A+DPTR)
47
48 SJMP $
49
50 END
```



# Exemplo 1 - Movimentação de dados

- Objetivo: analisar as diferentes formas de representação numérica e a movimentação de dados entre registradores.

```
1 ; Analisar as diferentes formas de representação
2 ; numérica e a movimentação de dados entre registradores
3
4 ORG 0000h
5
6     MOV R0, #11      ; movimenta o número 11 em decimal para o reg R0
7     MOV R1, #11h     ; movimenta o número 11 em hexa para o reg R0
8     MOV R2, #00001011b ;movimenta o número 11d na notação binária
9
10    ;Endereçamentos de registradores
11    MOV R3, #3h
12    MOV A, R3
13    MOV R7, A
14    MOV B, R1
15
16 END
```

# Exemplo 1 - Movimentação de dados

**Registers**

Register	Value
Regs	
r0	0x0b
r1	0x11
r2	0x0b
r3	0x03
r4	0x00
r5	0x00
r6	0x00
r7	0x03
Sys	
a	0x03
<b>b</b>	<b>0x11</b>
sp	0x07
sp_max	0x07
PC \$	C:0x000C
auxr1	0x00
dpnr	0x0000
states	8
sec	0.00000300
psw	0x00

**Disassembly**

```
C:0x0009  FF  MOV  R7,A
14:      MOV B, R1
C:0x000A  89F0  MOV  B(0xF0
C:0x000C  00  NOP
```

**TestExemplo1.a51**

```
1 ; Analisar as diferentes formas
2 ; numérica e a movimentação de
3
4 ORG 0000h
5
6 MOV R0, #11 ; moviment
7 MOV R1, #11h ; moviment
8 MOV R2, #00001011b ;movime
9
10 ;Endereçamentos de regist
11 MOV R3, #3h
12 MOV A, R3
13 MOV R7, A
14 MOV B, R1
15
16 END
```

# Exemplo 2 - Multiplicação

- Objetivo: realizar operações matemáticas.  
Multiplicar 2\*3 e armazenar o resultado em R1

```
1 ; Objetivo: Realizar operações matemáticas
2 ; Multiplicar 2 por 3 e armazenar o resultado em R1
3
4 ORG 0000h
5
6     MOV A, #2h ; Carrega o acc com 2
7     MOV B, #3h ; Carrega o B com 3
8     MUL AB      ; Realiza a operação de multiplicação
9     MOV R1, A   ; Copia o resultado para R1
10
11     SJMP $      ; Fica indefinidamente nesta linha
12 END
```

# Exemplo 2 - Multiplicação

Registers

Register	Value
Regs	
r0	0x00
r1	0x06
r2	0x00
r3	0x00
r4	0x00
r5	0x00
r6	0x00
r7	0x00
Sys	
a	0x06
b	0x00
sp	0x07
sp_max	0x07
PC \$	C:0x0007
aux1	0x00
dp1r	0x0000
states	8
sec	0.00000300
psw	0x00

Disassembly

11: SJMP \$ ; Fica indefinidamente nesta linha

C:0x0007 80FE SJMP C:0007

C:0x0009 00 NOP

C:0x000A 00 NOP

TestExemplo2.a51

1 ; Objetivo: Realizar operações matemáticas

2 ; Multiplicar 2 por 3 e armazenar o resultado em R1

3

4 ORG 0000h

5

6 MOV A, #2h ; Carrega o acc com 2

7 MOV B, #3h ; Carrega o B com 3

8 MUL AB ; Realiza a operação de multiplicação

9 MOV R1, A ; Copia o resultado para R1

10

11 SJMP \$ ; Fica indefinidamente nesta linha

12 END

# Exemplo 3 - Criação de um temporizador

- Objetivo: criar um temporizador e verificar seu tempo de execução.

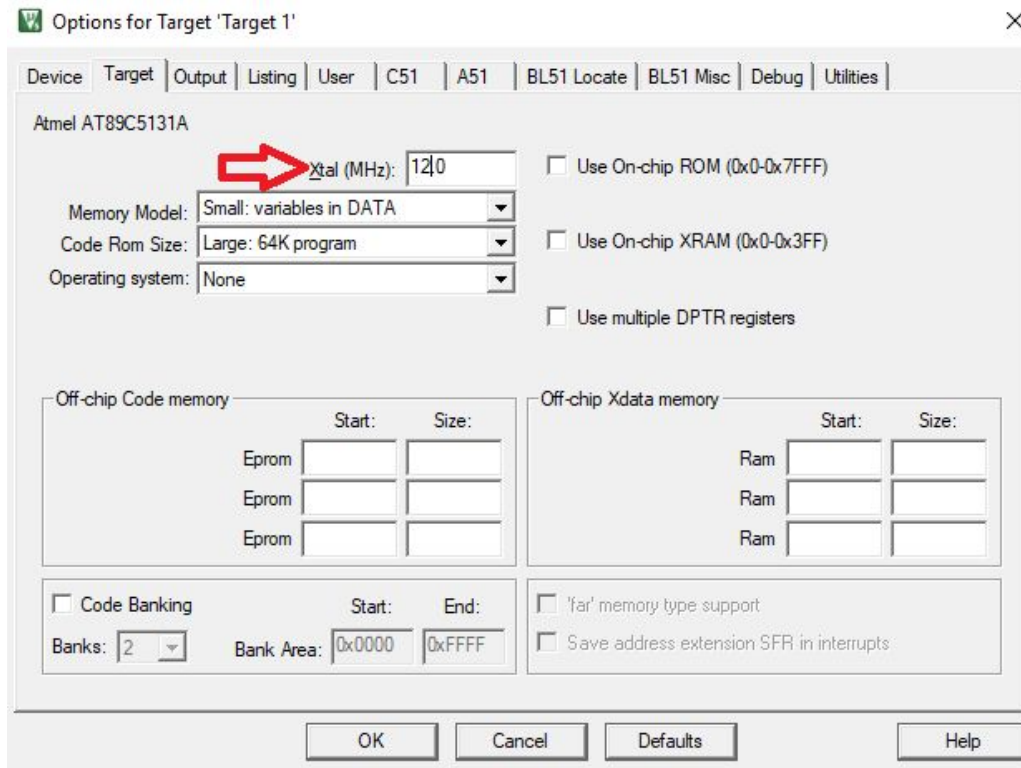
```
1  ; Objetivo: criar um temporizador e verificar
2  ; seu tempo de execução
3
4  ORG 0000h
5
6      ; Carrega o R6 com um valor para fazer um laço
7      MOV R6, #255
8
9      ; Faz um laço para fazer um delay
10     delay:
11         DJNZ R6, delay
12     NOP
13     JMP $      ; fica nesta linha indefinidamente
14     END
15
16     ; Sobre simulador de temporização no keil
17     ; http://www.keil.com/support/docs/971.htm
```



# Exemplo 3 - Criação de um temporizador

- Mudar o clock do microprocessador para 12 MHz.

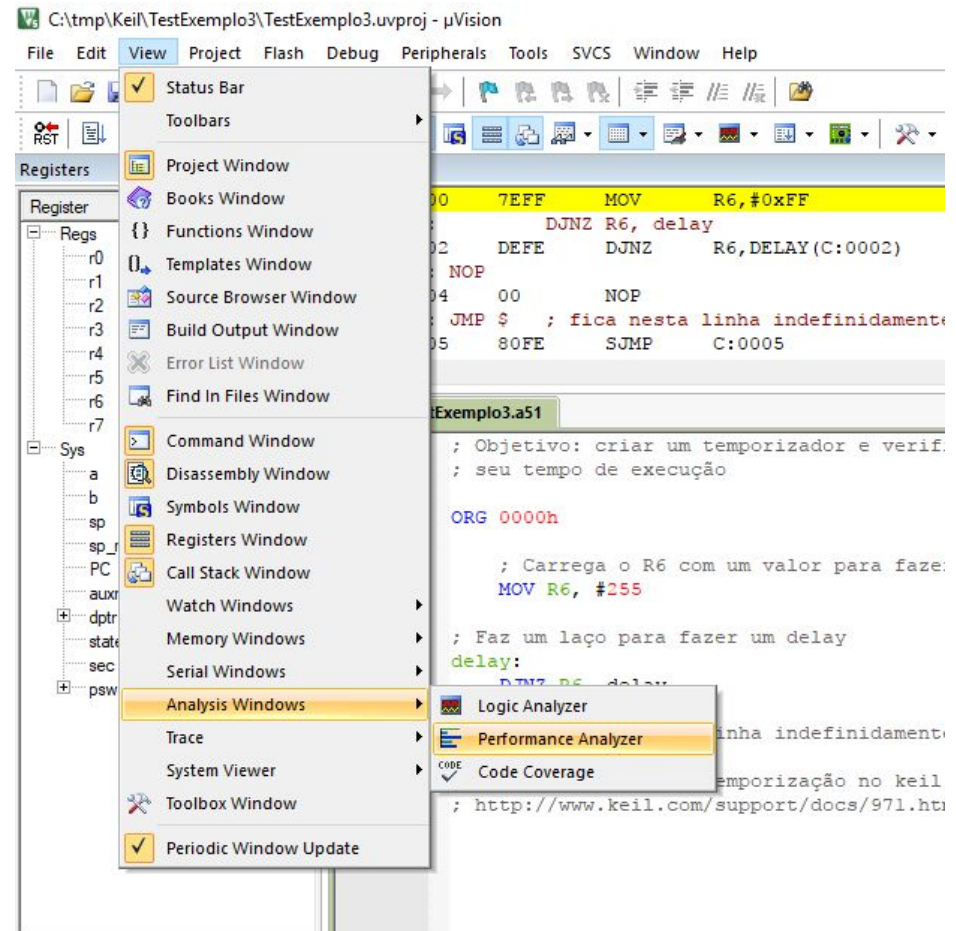
Clicar com o botão direito em **Target 1** na seção do projeto e depois **Options for Target Target 1**. E em seguida colocar o **Xtal** para **12 MHz** para facilitar as contas.





# Exemplo 3 - Criação de um temporizador

- Para verificar o tempo de execução, entre no modo **Debug**, no menu **View / Analysis Windows / Performance Analyzer**



# Exemplo 3 - Criação de um temporizador

- Colocar breakpoints antes e depois da rotina de temporização.

```
1 ; Objetivo: criar um temporizador e verificar
2 ; seu tempo de execução
3
4 ORG 0000h
5
6 ; Carrega o R6 com um valor para fazer um laço
7 MOV R6, #255
8
9 ; Faz um laço para fazer um delay
10 delay:
11     DJNZ R6, delay
12 NOP
13 JMP $ ; fica nesta linha indefinidamente
14 END
15
16 ; Sobre simulador de temporização no keil
17 ; http://www.keil.com/support/docs/971.htm
```

# Exemplo 3 - Criação de um temporizador

The screenshot displays the Keil Performance Analyzer interface. On the left, the 'Registers' window shows the state of various registers. The 'sec' register, located under the 'Sys' category, is highlighted with a red box and contains the value 0.00051100. The 'sp' register is also highlighted with a blue box and contains the value 0x07. The 'states' register contains the value 511. On the right, the 'Performance Analyzer' window shows a table with columns for 'min time:', 'max time:', 'avg time:', 'total time:', '%', and 'count:'. The 'total time:' column is highlighted with a red box and contains the value 0.000511. Below this, the 'Disassembly' window shows the assembly code for 'TestExemplo3.a51'. The code includes comments in Portuguese and assembly instructions: 'ORG 0000h', 'MOV R6, #255', 'DJSZ R6, delay', 'NOP', 'JMP \$', and 'END'. A green bar highlights the 'NOP' instruction at line 12.

Register	Value
Regs	
r0	0x00
r1	0x00
r2	0x00
r3	0x00
r4	0x00
r5	0x00
r6	0x00
r7	0x00
Sys	
a	0x00
b	0x00
sp	0x07
sp_max	0x07
PC \$	C:0x0004
auxr1	0x00
dp1r	0x0000
states	511
sec	0.00051100
psw	0x00

min time:	max time:	avg time:	total time:	%	count:
			0.000511	100.0	

```
1 ; Objetivo: criar um temporizador e verificar
2 ; seu tempo de execução
3
4 ORG 0000h
5
6 ; Carrega o R6 com um valor para fazer um laço
7 MOV R6, #255
8
9 ; Faz um laço para fazer um delay
10 delay:
11     DJSZ R6, delay
12     NOP
13     JMP $ ; fica nesta linha indefinidamente
14     END
15
16 ; Sobre simulador de temporização no keil
17 ; http://www.keil.com/support/docs/971.htm
```

# Exemplo 4 - Fatorial

- Objetivo: Calcular o fatorial de um número até 5 e no final armazenar o resultado final em R0.

```
1 ; Calcular o fatorial de um número até 5
2 ; e no final armazenar o resultado final em R0
3 ; R0 -> guarda a saída
4 ; R1 -> guarda o número a calcular o fatorial
5
6 ORG 0000h
7
8 ; resolve o fatorial de 4
9 MOV R1, #4
10 MOV A, R1
11
12 fat:
13 DEC R1
14 MOV B, R1
15 MUL AB
16 ; compara R1 com 1
17 ; e pula se ainda não for igual
18 CJNE R1, #1h, fat
19 MOV R0, A
20
21 END
```

# Exemplo 4 - Fatorial

The screenshot displays a microcontroller development environment with two main panels: **Registers** and **Disassembly**.

**Registers Panel:**

Register	Value
<b>Regs</b>	
r0	0x00
<b>r1</b>	<b>0x01</b>
r2	0x00
r3	0x00
r4	0x00
r5	0x00
r6	0x00
r7	0x00
<b>Sys</b>	
<b>a</b>	<b>0x18</b>
b	0x00
<b>sp</b>	<b>0x07</b>
sp_max	0x07
PC \$	C:0x000A
auxr1	0x00
dpnr	0x0000
states	29
sec	0.00001088
psw	0x00

**Disassembly Panel:**

The disassembly window shows the assembly code for **TestExemplo4.a51**. The code is as follows:

```
7
8      ; resolve o fatorial de 4
9      MOV R1, #4
10     MOV A, R1
11
12     fat:
13     DEC R1
14     MOV B, R1
15     MUL AB
16     ; compara R1 com 1
17     ; e pula se ainda não for igual
18     CJNE R1, #1h, fat
19     MOV R0, A
20
21     END
```

The registers panel shows the state of the microcontroller's registers. The **Regs** section lists general-purpose registers r0 through r7, with r1 highlighted at 0x01. The **Sys** section lists system registers, with 'a' at 0x18, 'sp' at 0x07, and 'PC \$' at C:0x000A. The disassembly panel shows the assembly code for the factorial program, with the current instruction being **MOV R0, A** at address 19.

# Exemplo 5 - Entradas e Saídas

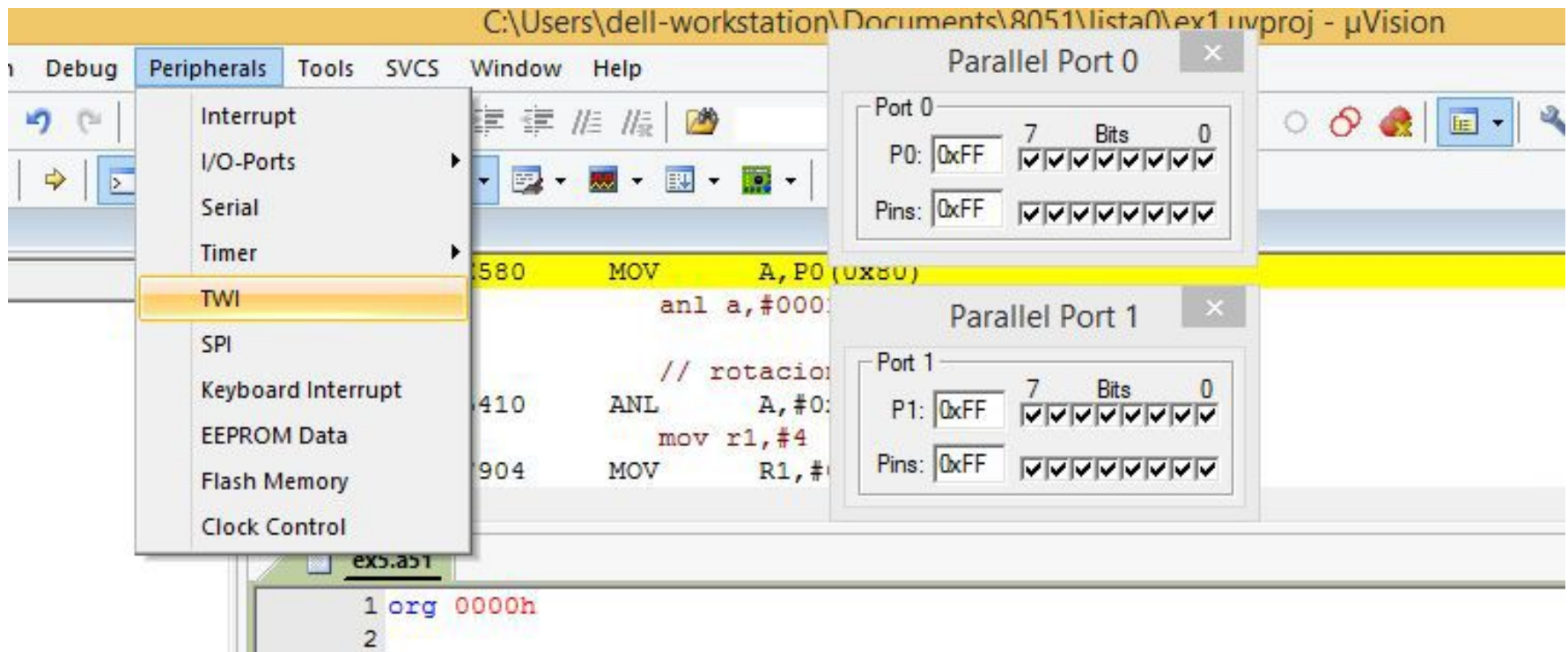
- Objetivo: realizar a leitura do status da entrada digital **P0.4** e enviar nível lógico **alto** (1) para saída **P1.0** e enviar nível lógico **baixo** (0) para as demais saídas da **Porta1**, quando a entrada for 1.
- Caso a entrada seja zero todos os bits da **Porta1** devem ser zero.

```
7  ORG 0000h
8
9      ;loop para verificar o estado do pino
10 loop:
11     MOV A, P0      ; copiar o valor do port para o Acc
12     ANL A, #00010000b ; aplica a máscara em Acc
13
14     ; rotaciona o bit do Acc de 5 para 0
15     MOV R1, #4
16 rot:  RR A
17     DJNZ R1, rot
18     CJNE A, #1, aux ;Se o Acc não for 1, então porta não está setada
19
20     MOV P1, #1     ;Porta está setada
21     JMP loop
22
23 aux:
24     MOV P1, #0
25     JMP loop
26
27     JMP $
28
29 END
```



# Exemplo 5 - Entradas e Saídas

- Para simular, abra o menu **Peripherals** => I/O Ports / Port 0 e Port 1



# Exemplo 5 - Entradas e Saídas

The screenshot displays a software development environment with several windows:

- Registers:** A table showing the state of registers. The 'Regs' section lists r0 through r7, all with a value of 0x00. The 'Sys' section lists a (0x04), b (0x00), sp (0x07), sp\_max (0x07), and PC (\$ C:0x0007).
- Disassembly:** A window showing assembly instructions. Line 26 is highlighted: `C:0x0014 80EA SJMP LOOP(C:0000)`. Other visible instructions include `JMP loop` at line 25 and `JMP S` at line 27.
- TestExemplo5.a51:** An assembly source file window showing the following code:

```
7 ORG 0000h
8
9 ;loop para verificar o estado do pino
10 loop:
11 MOV A, P0 ; copiar o valor do port para o Acc
12 ANL A, #00010000b ; aplica a máscara em Acc
13
14 ; rotaciona o bit do Acc de 5 para 0
15 MOV R1, #4
16 rot:
17 RR A
18 DJNZ R1, rot
19 CJNE A, #1, aux ;Se o Acc não for 1, então porta não está setada
20
21 MOV P1, #1 ;Porta está setada
22 JMP loop
```
- Parallel Port 0:** A configuration dialog box for Port 0. It shows 'P0: 0x10' and 'Pins: 0x10' with bit selection checkboxes for bits 0 through 7.
- Parallel Port 1:** A configuration dialog box for Port 1. It shows 'P1: 0x01' and 'Pins: 0x01' with bit selection checkboxes for bits 0 through 7.
- Memory 1:** A window showing the memory address field.

# Atividade 1

Desenvolver as três tarefas abaixo e apresentar ao professor até a próxima aula.

# Tarefa 1

- Crie uma rotina assembly que gere um atraso (*delay*) de 1 segundo
- Comprove o funcionamento pelo modo de depuração

# Tarefa 2

- Desenvolver um código assembly para "piscar" dois leds alternados, a cada 1 segundo, conectados aos pinos P1.7 e P2.1, quando a entrada P0.3 estiver ativa. Os demais pinos das Portas 1-2 devem permanecer inativos.

# Tarefa 3

- Objetivo: Fazer varredura de uma string armazenada na memória de programa e colocar cada caracter no acumulador