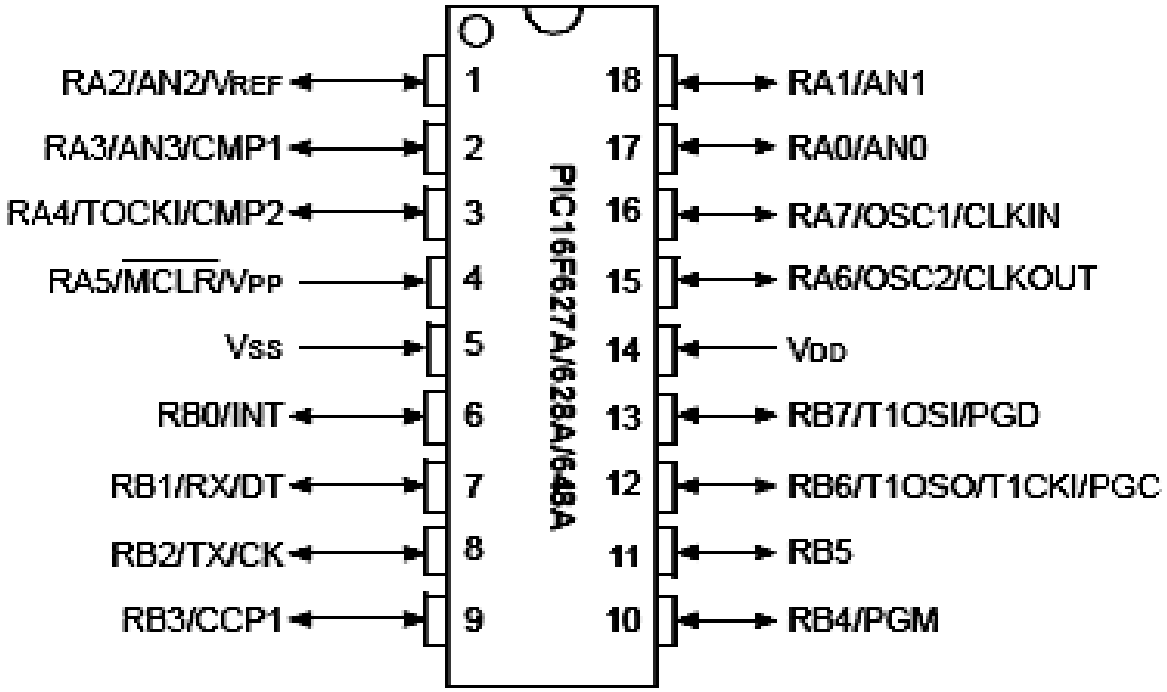
Programas de gravação e simulação:

Os programas (e versões) são:
gravação : MPLAB (v8.46) e Winpic800 (OS: x86)
simulação: Real Pic Simulator(v 1.1.0.0) e Picsimlab(v0.5.0)

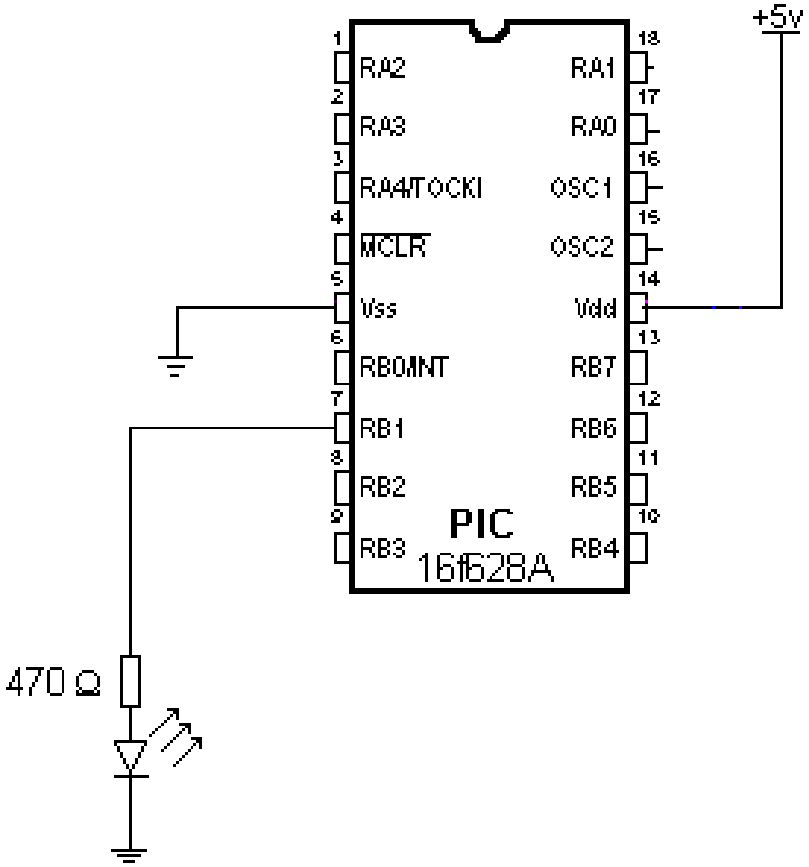
Os programas também podem ser obtidos livremente na WEB.

Também no site:
ftp://ftp.pucmg.br/Computacao/Mais_Utilizados/Programacao_Hardware/

Microcontrolador da família 16F6xxA com *core* de 14 bits



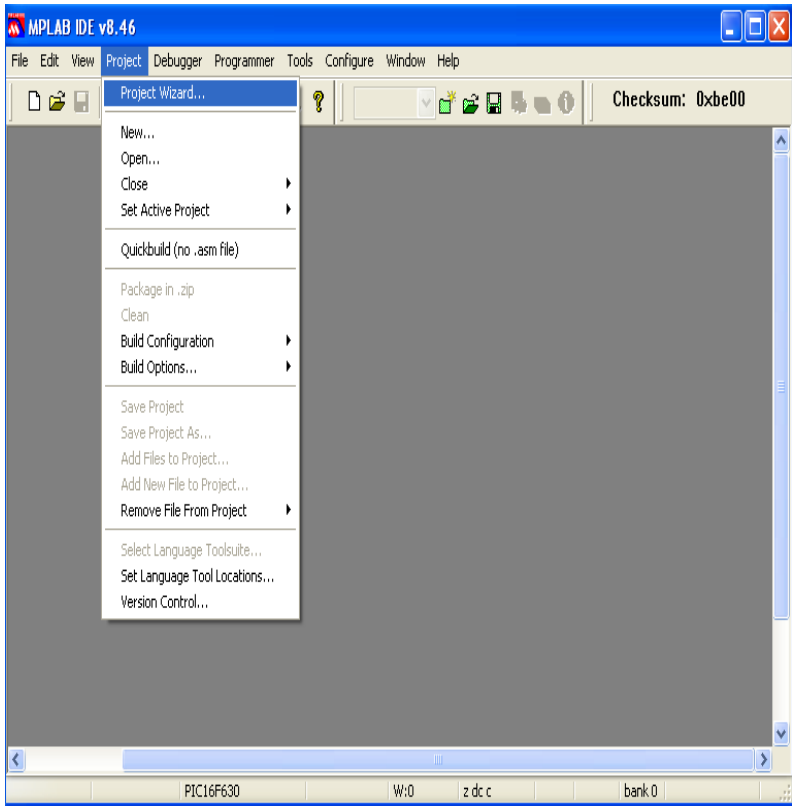
Circuito a ser utilizado para aula 1

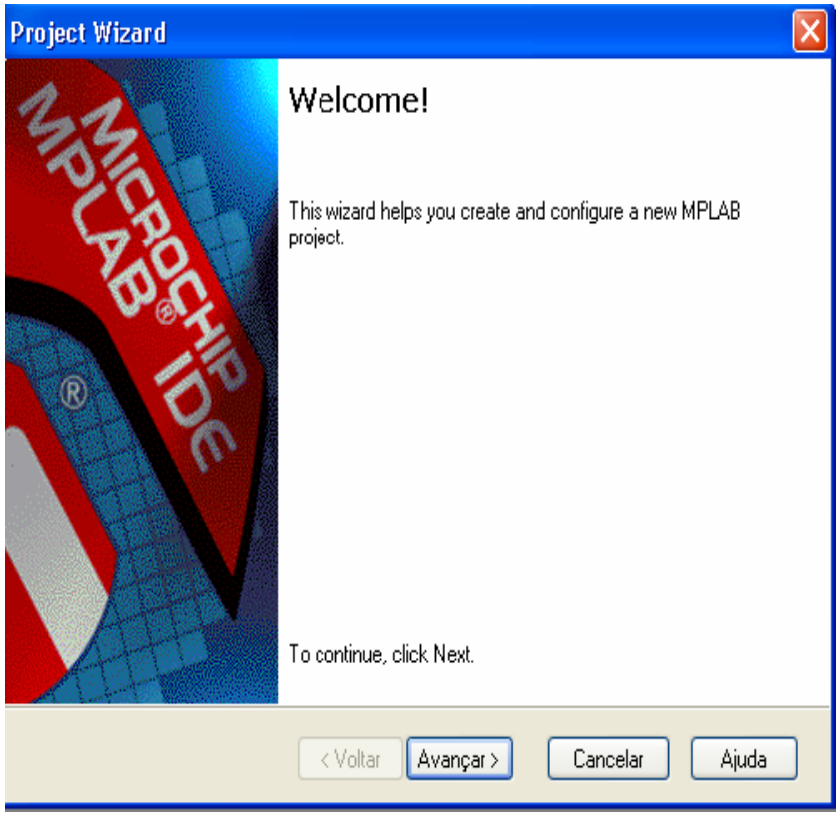


Programando – MPLAB 8.46 (Microchip)

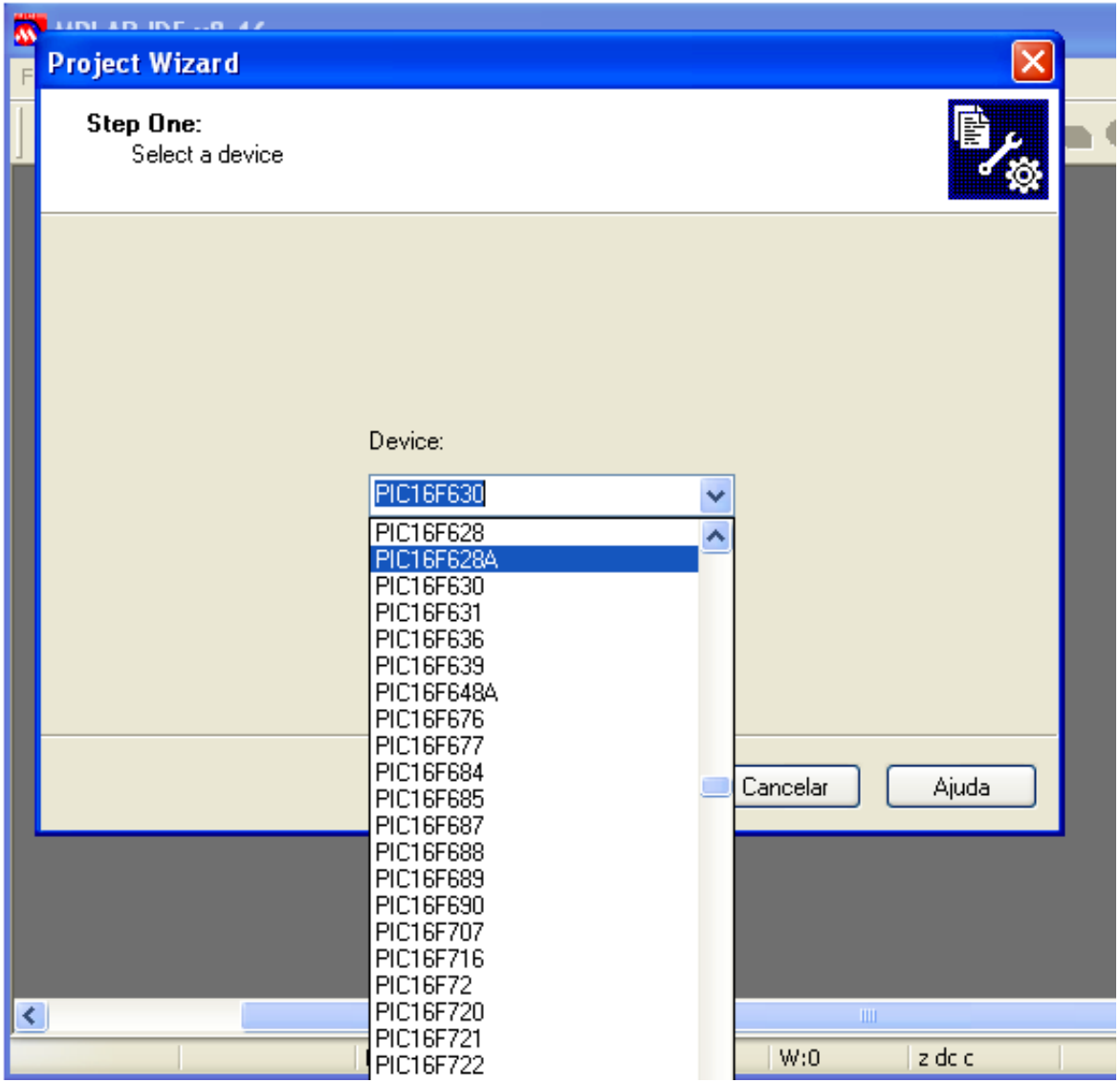
CRIAÇÃO DO PROJETO - PRIMEIRO PASSO

Uma vez que você abriu o MPLAB a primeira coisa a se fazer é criar um projeto ou abrir um projeto criado anteriormente. O projeto que vamos criar terá o nome de AULA01.MCP. ***É importante ressaltar que o projeto e o código-fonte PISCALED.ASM têm que ser salvos na mesma pasta.***





A seguir, precisamos definir com qual microcontrolador vamos trabalhar. Neste exemplo será selecionado o PIC16F628A.



Os programadores de microcontroladores usam basicamente dois tipos de código-fonte: ASSEMBLY e LINGUAGEM C.

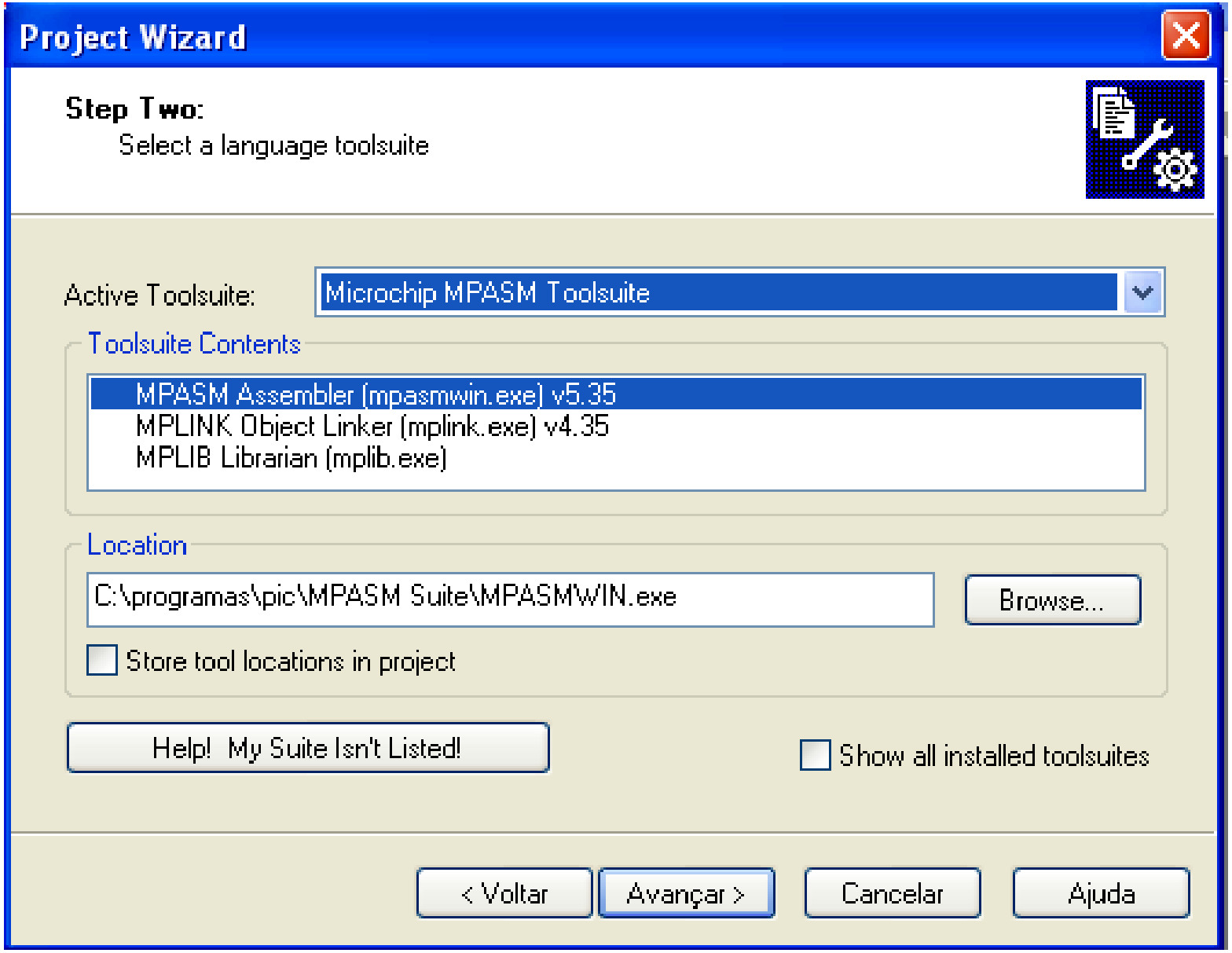
O código *Assembly* usa mnemônicos para trabalhar cada operação (instrução). As instruções atuam nos bits e bytes dos registradores internos do PIC.

Já a linguagem C, tem prontos comandos que realizam, muitas vezes, mais de uma instrução *assembly*.

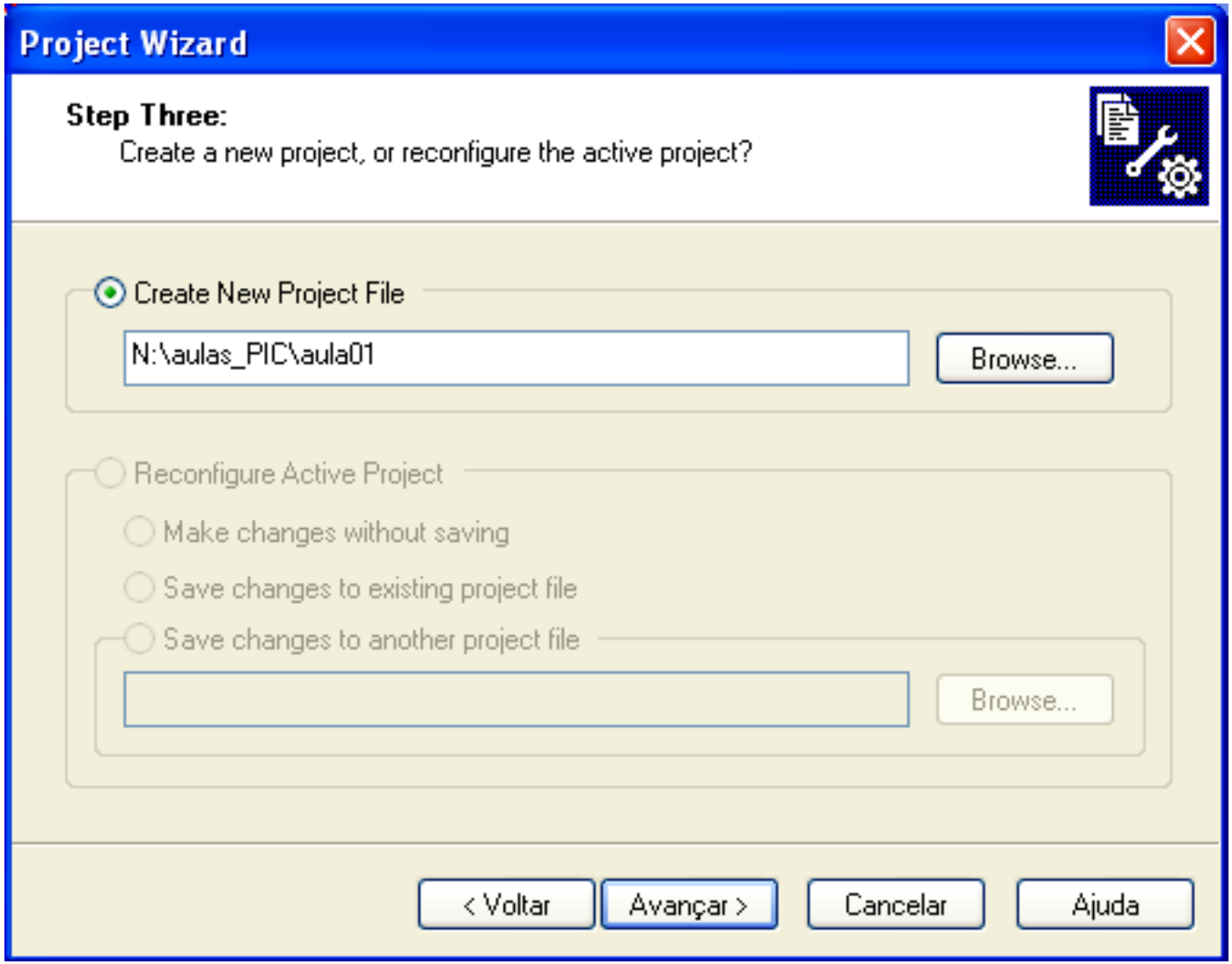
A vantagem de usar *assembly* é que a arquitetura interna do microcontrolador estudado fica muito clara, além do desenvolvimento de programas menores e mais rápidos.

A linguagem C traz a vantagem da rapidez no desenvolvimento dos programas.

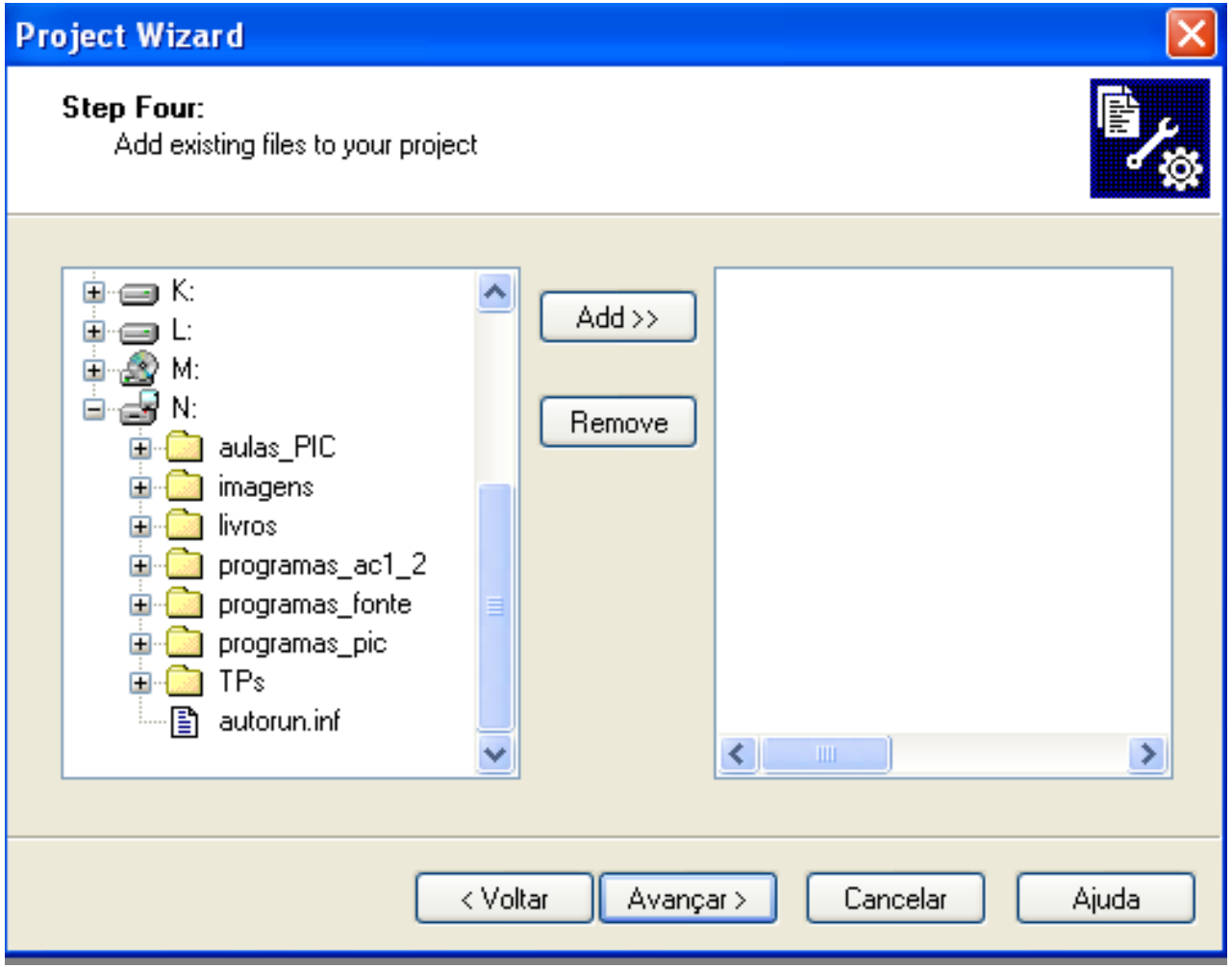
A seguir, a próxima janela do *Project Wizard*. Nesta etapa, selecionamos o compilador que iremos trabalhar e verificamos o local onde o mesmo está gravado. Para trabalharmos com a programação *Assembly*, selecionaremos a opção MPASM, como mostra a figura. Em *Active ToolSuite* estão as outras opções de compiladores.


Project Wizard
Step Two:
Select a language toolsuite
Active Toolsuite:
Microchip MPASM Toolsuite
Toolsuite Contents
MPASM Assembler (mpasmwin.exe) v5.35
MPLINK Object Linker (mplink.exe) v4.35
MPLIB Librarian (mplib.exe)
Location
C:\programas\pic\MPASM Suite\MPASMWIN.exe
Browse...
Store tool locations in project
Help! My Suite Isn't Listed!
Show all installed toolsuites
< Voltar
Avançar >
Cancelar
Ajuda

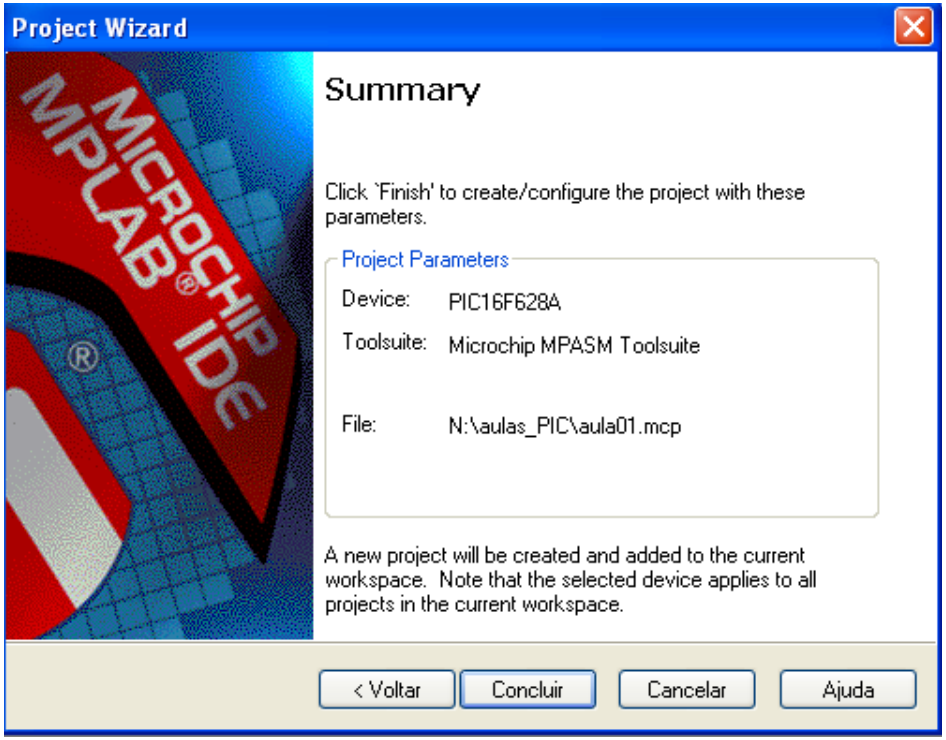
Nessa etapa, definiremos o nome do projeto e o local de gravação dos arquivos do mesmo. O projeto e o código-fonte têm que ser salvos na mesma pasta. A Figura mostra a janela que controla esta etapa.

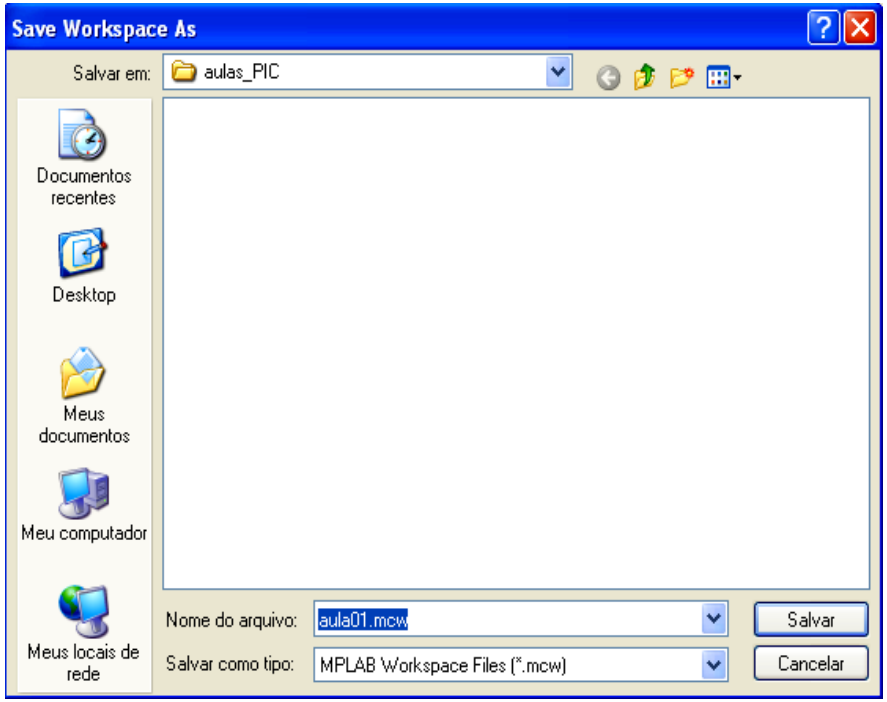


Agora transferiremos o código-fonte *Assembly* (ASM) para o projeto como mostrado na Figura. Se não existir nenhum ou iremos criar a partir do início, apenas “avançar”.

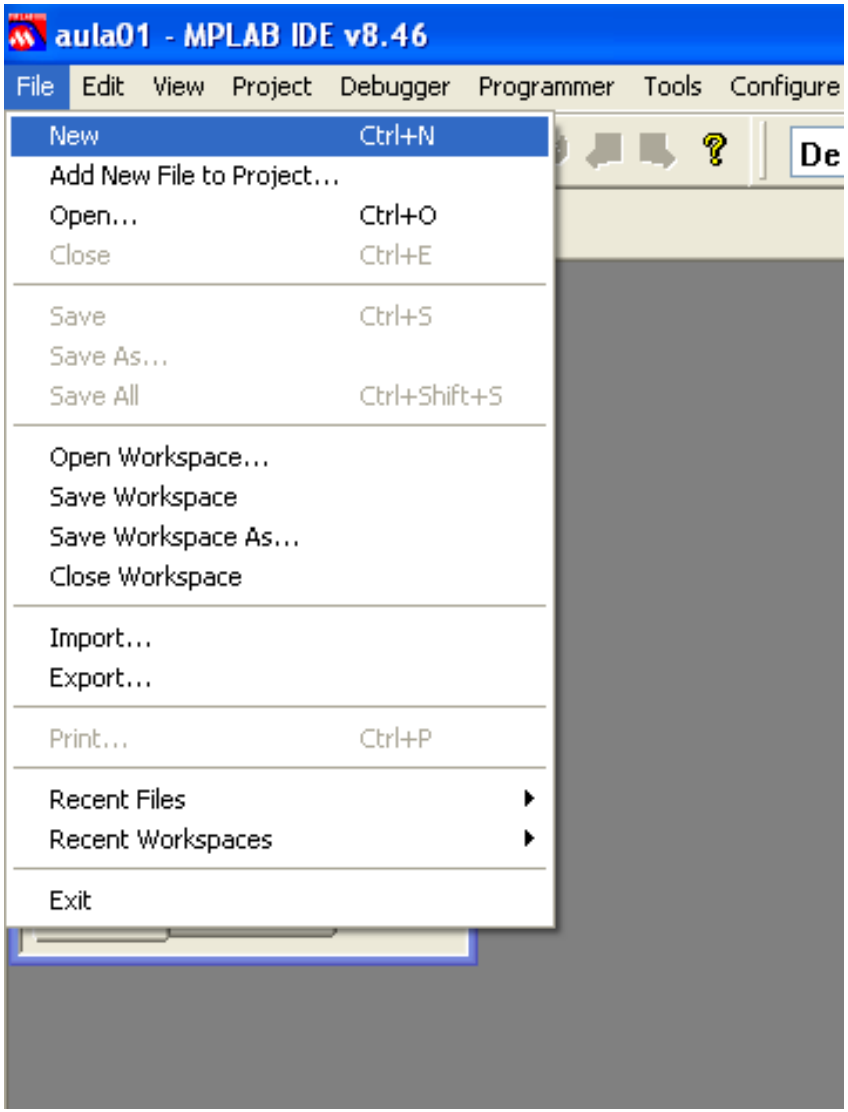


Na última janela temos um resumo do projeto, ao concluir, salvamos o projeto conforme criado

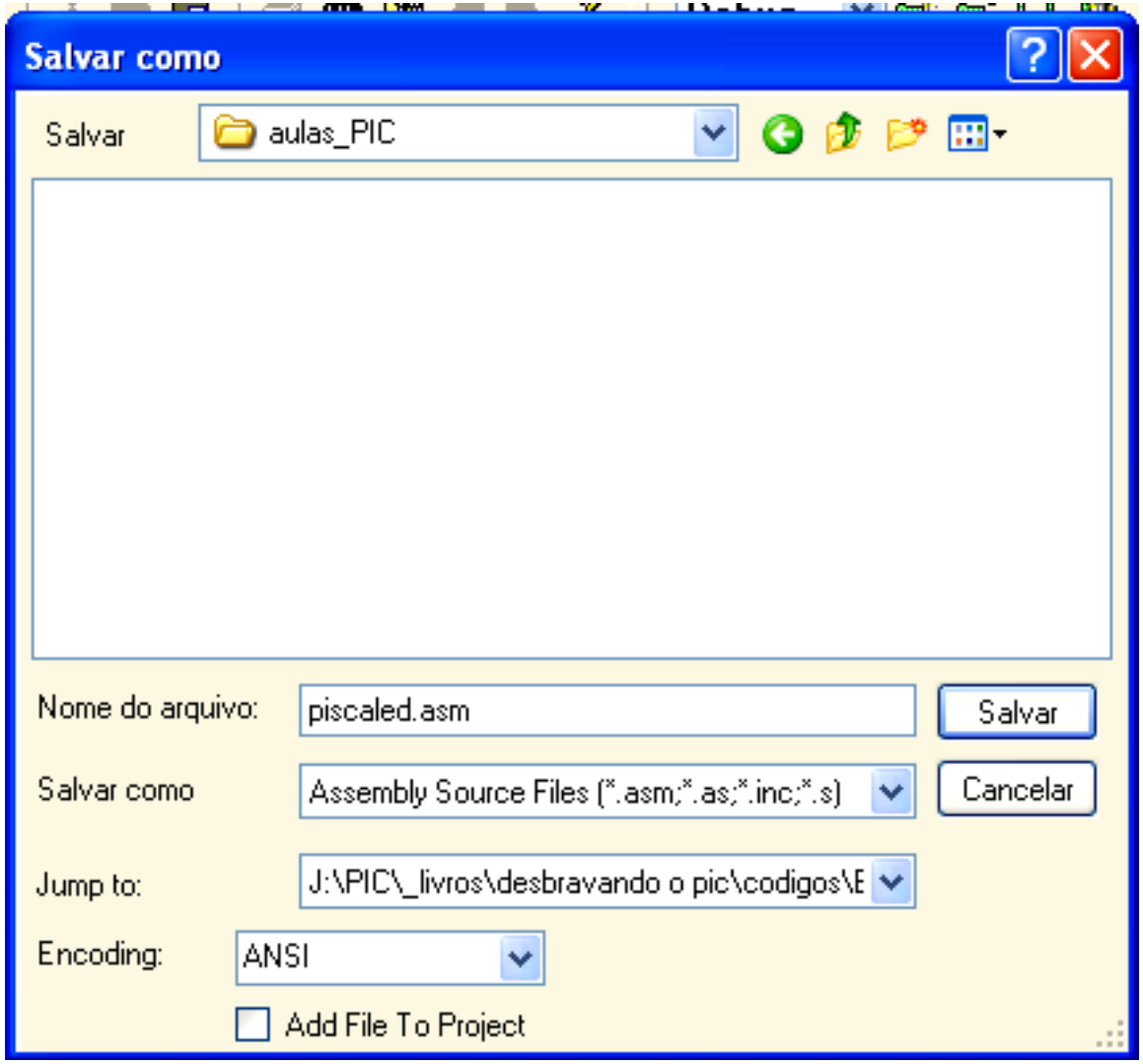




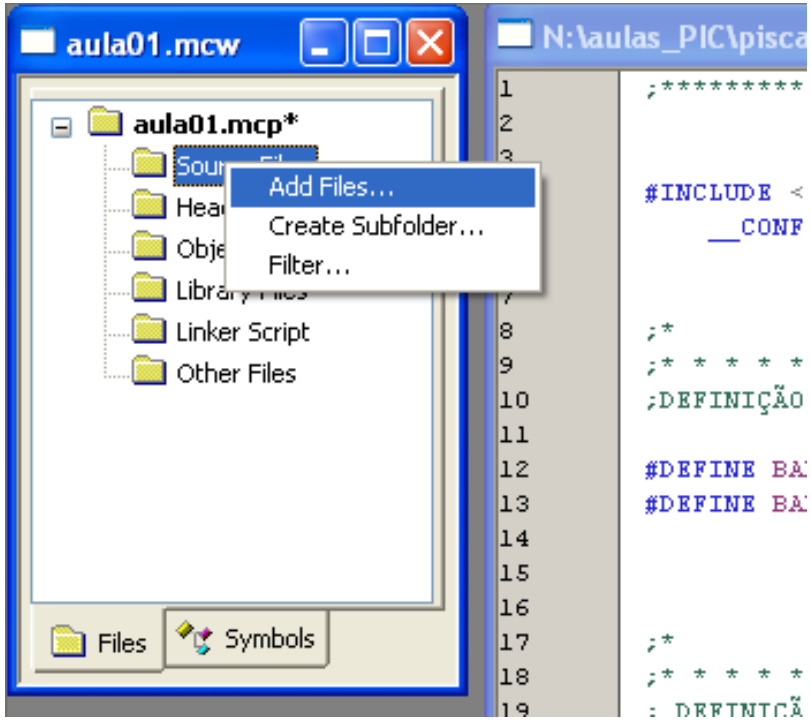
Inserindo um novo arquivo ao projeto, conforme a figura a seguir.

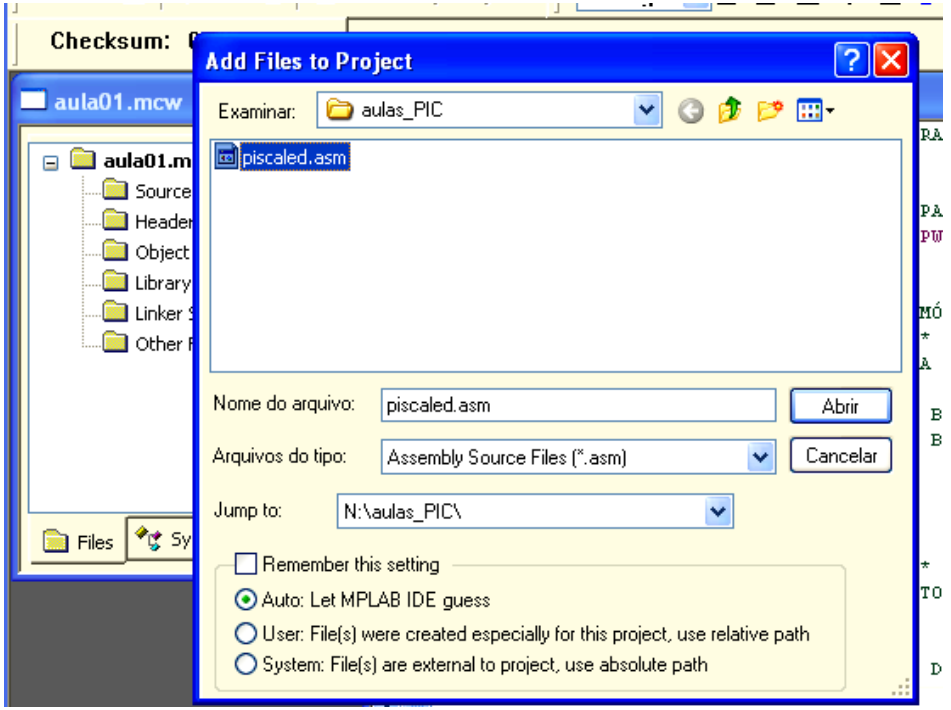


Editar o texto na janela e salvar com a extensão .asm na mesma pasta do projeto (vamos criar o arquivo PISCALED.ASM).

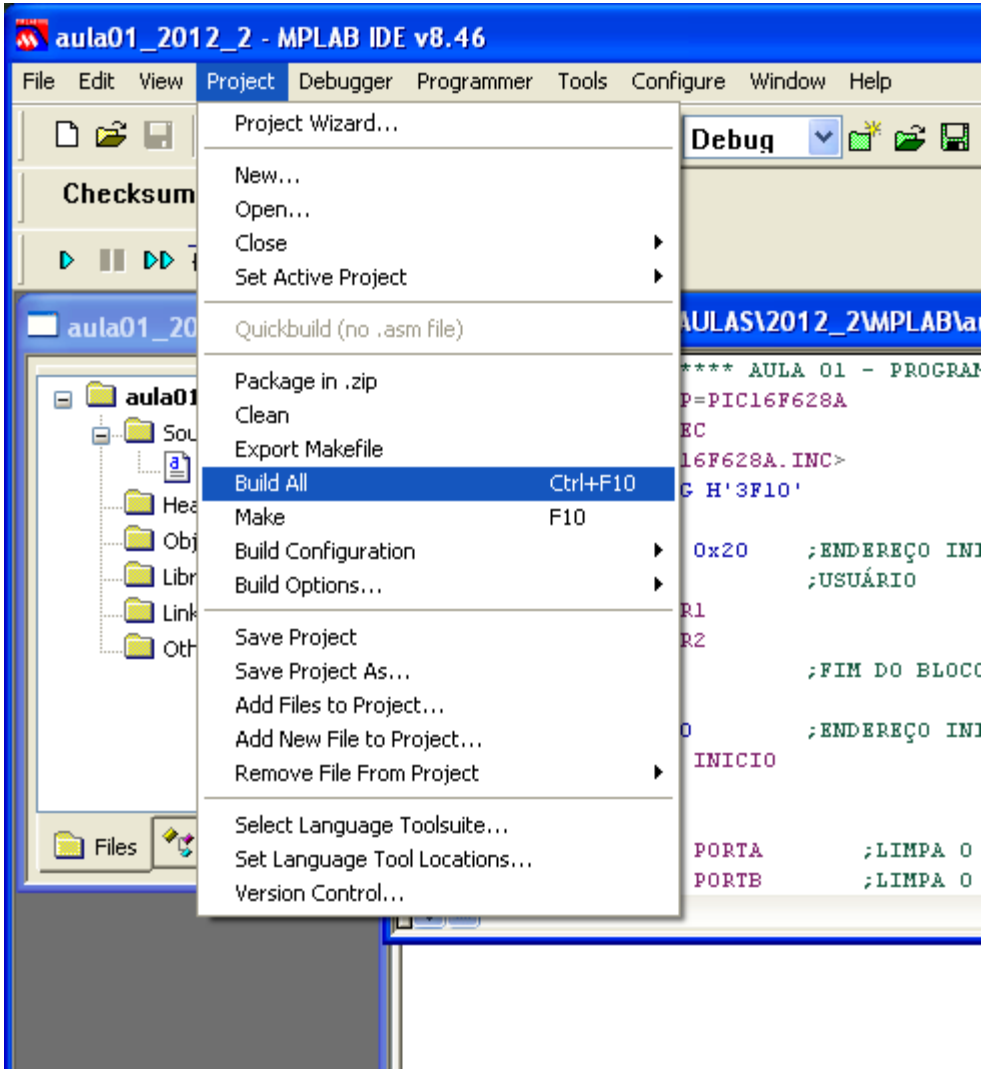


Com o arquivo gravado, adicioná-lo ao projeto como um fonte (Source File).



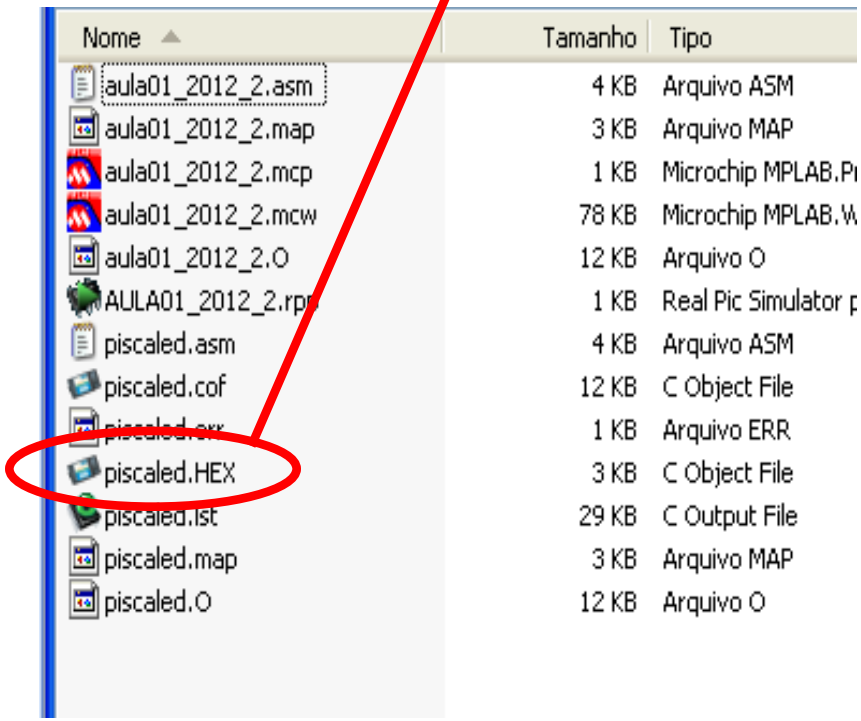


Compilar o programa.



Verificar se a compilação foi OK (na janela OUTPUT) e o arquivo com a linguagem de máquina foi gerado (.hex).

Na pasta do projeto



Considere agora o programa a seguir (PISCALED0.asm):

```
;************** AULA 01 - PROGRAMA PARA PISCAR LED *****************
                LIST   P=PIC16F628A
                RADIX DEC
                #INCLUDE <P16F628A.INC>
                __CONFIG _INTRC_OSC_NOCLKOUT & _WDT_OFF & _PWRTE_ON & _BODEN_OFF & _LVP_OFF &
_CP_OFF & _MCLRE_OFF & _DATA_CP_OFF
                ;ou __CONFIG H'3F10'

                CBLOCK      0x20            ;ENDEREÇO INICIAL DA MEMÓRIA DE
                                                                ;USUÁRIO
                CONTADOR1
                CONTADOR2
                ENDC                                    ;FIM DO BLOCO DE MEMÓRIA

                ORG         0x00                        ;ENDEREÇO INICIAL DE PROCESSAMENTO
                GOTO        INICIO

INICIO
                CLRF        PORTA                       ; LIMPA O PORTA
                CLRF        PORTB                       ; LIMPA O PORTB
                BSF STATUS, RP0     ; MUDAR PARA BANCO 1 DA MEMÓRIA
                CLRF TRISB          ; TODO TRISB =0 -> SAÍDA
                BCF STATUS, RP0     ; MUDAR PARA BANCO 0 DA MEMÓRIA

REPETE
                BSF PORTB, 1
                BCF PORTB, 1
                GOTO REPETE


                END
```

Explicação do programa

Diretivas de compilação

LIST P=PIC16F628A

Informa ao compilador para converter o código texto fonte para o padrão do PIC16F128A

RADIX DEC

Informa ao compilador para converter todo número que não possuir nenhuma “marca especial”para decimal

#INCLUDE <P16F628A.INC>

Informa ao compilador para incluir no nosso código fonte, um arquivo da microchip chamado P16F628A.inc, que está no diretório onde o Mplab está instalado. Nesse arquivo temos as equivalências dos nomes dos registradores e os respectivos endereços físicos, possibilitando nomes e abreviações mais amigáveis do que ficar guardando números.

Algumas linhas do arquivo P16F628A.INC

```
PORTA           EQU     H'0005'
PORTB           EQU     H'0006'
PCLATH          EQU     H'000A'
INTCON          EQU     H'000B'
PIR1            EQU     H'000C'
TMR1L           EQU     H'000E'
TMR1H           EQU     H'000F'
T1CON           EQU     H'0010'
TMR2            EQU     H'0011'
T2CON           EQU     H'0012'
CCPR1L          EQU     H'0015'
CCPR1H          EQU     H'0016'
CCP1CON         EQU     H'0017'
RCSTA           EQU     H'0018'
TXREG           EQU     H'0019'
RCREG           EQU     H'001A'
CMCON           EQU     H'001F'
OPTION_REG      EQU     H'0081'
TRISA           EQU     H'0085'
TRISB           EQU     H'0086'
PIE1            EQU     H'008C'
```

Obs.: o termo **EQU**,
serve para definir uma
constante em assembler.
Exemplos:

```
X               EQU 5
TESTE           EQU 15
PORTA           EQU H'0005'
```

__CONFIG _INTRC_OSC_NOCLKOUT & _WDT_OFF & _PWRTE_ON & _BODEN_OFF & _LVP_OFF & _CP_OFF & _MCLRE_OFF & _DATA_CP_OFF

Esta instrução configura o hardware interno do PIC, no nosso caso (__config, "dois traços, ou underline, + config) é a instrução ‘configurar’ as strings, grupo de letras entre os &.
(Obs: barra de espaço + & + barra de espaço)

BROWN OUT DETECT Trata-se de um sistema de detecção automática de baixa tensão capaz de resetar o PIC. Isso significa que, se a tensão de alimentação (V_{DD}) for menor que 4V (típico) por mais de 100μs, o sistema será reiniciado. Esses dados foram retirados do data sheet, da seção de especificações elétricas são valores fixos. Para trabalhar com valores diferentes; será necessário desabilitar este recurso e montar um circuito externo de BOR ligado ao pino MCLR.

POWER UP TIMER

O PIC16F628A possui o Power Up Timer (uma espécie de POR melhorado - veja no próximo capítulo) interno que pode ser habilitado ou não na hora da gravação. Esta opção irá fazer com que o PIC só comece a operar cerca de 72 ms após o pino MCLR ser colocado em nível alto. Se você estiver utilizando um circuito de POR melhorado (externo), então essa opção deve estar desativada.

CÓDIGO DE PROTEÇÃO

Para a gravação em série é muito importante que essa opção esteja ativada, pois isso impedirá que qualquer pessoa consiga ler o programa gravado dentro do PIC. Esta é a única proteção que você terá para que ninguém possa "copiar" o seu sistema. No caso do PIC16F628A, que é regravável eletronicamente, não há problemas em deixar essa opção sempre ativa, mesmo durante a fase de desenvolvimento, pois esse código impedirá que você leia a memória (inclusive para o comando "Verify"), mas não impedirá que você grave outro programa por cima (desde que grave toda a memória de programa, e não só parte dela para agilizar o processo). Entretanto, tome muito cuidado se você estiver trabalhando com PICs janelados (apagáveis por luz ultravioleta), porque a gravação de um componente com essa opção ligada pode significar sua perda, pois pode ser que você nunca mais consiga regravá-lo.

WATCHDOG TIMER

O WDT também pode ser ativado ou não na hora da gravação, e esta configuração não poderá ser alterada posteriormente pelo programa. Por isso, lembre-se de que, para ativar essa opção, seu programa deve estar preparado para limpar o contador do WDT periodicamente. Caso contrário, seu programa será resetado toda vez que esse contador estourar.

LOW VOLTAGE PROGRAM

Este também é um recurso relativamente novo para muitos modelos de PIC. Trata-se do sistema de programação do PIC (gravação da memória de programa) em baixa tensão: 5V. Normalmente essa programação é habilitada por uma alta tensão (13V) no pino MCLR. Acontece que hoje é possível criarmos sistemas onde um PIC possa gravar o programa de outro PIC, ou então efetuarmos um upgrade remoto. Para facilitar esta implementação, a Microchip elaborou um sistema onde não é necessário os 13V, raramente disponíveis na maioria dos projetos. Assim, todo o processo utiliza somente o nível TTL, customizando o hardware. Entretanto, nem tudo são flores. Para que esse sistema funcione de forma robusta e eficaz, um pino deve ser dedicado à função de entrar no modo de programação. Quando habilitada esta opção, o pino 10 deixa de ser o RB4 e passa a operar como PGM.

TIPO DE OSCILADOR

Existem dois grupos básicos de osciladores para uso com o PIC16F628A: internos e externos. Este modelo possui dois osciladores internos (37 KHz e 4 MHz - seleção por software) e capacidade para operar com vários tipos de osciladores externos. A escolha deve ser feita levando-se em conta o hardware do projeto.
Veja agora as opções disponíveis:

- **RC_CLKOUT:** Para oscilador externo tipo RC com o pino 15 operando como CLKOUT, isto é, com uma onda quadrada de ¼ da freqüência.
- **RC_I/O:** Para oscilador externo tipo RC com o pino 15 operando como I/O (RA6).
- **INTOSC_CLKOUT:** Para oscilador interno com o pino 15 operando como CLKOUT, isto é, com uma onda quadrada de ¼ da freqüência.
- **INTOSC_I/O:** Para oscilador interno com o pino 15 operando como I/O (RA6).
- **EC_I/O:** Para clock externo (circuito auto-oscilante) com o pino 15 operando como I/O (RA6).
- **XT:** Para osciladores externos tipo cristal ou ressoadores.
- **HS:** Para cristais ou ressoadores externos com freqüências elevadas (acima de 4 MHz).
- **LP:** Para cristais ou ressoadores externos com baixas freqüências (abaixo de 200 KHz). Utilizado para minimizar o consumo.

MASTER CLEAR ENABLE

Esta é opção que define o uso do pino 4, que pode ser I/O ou Master Clear externo (MCLR). Ao habilitar esta opção, o pino 4 funciona como MCLR.

DEFININDO AS CONFIGURAÇÕES NO PRÓPRIO PROGRAMA

Para evitarmos a chata tarefa de termos de configurar essas opções todas as veze que vamos gravar um PIC, e também para evitarmos dúvidas sobre o correto estad de cada uma delas, é possível especificarmos esta escolha no próprio código d programa, por meio de uma diretriz de compilação. A diretriz **__CONFIG** (com underlines) configura diretamente as opções de gravação. Para facilitar o trabalh com ela, os arquivos de include já definem nomes para as diversas opções. No cas do PIC16F628A, teremos as seguintes opções:

_BOREN_ON: Para BOR ligado.
_BOREN_OFF: Para BOR desligado.
_CP_ON: Para code protection ligado.
_CP_OFF: Para code protection desligado.
_DATA_CP_ON: Para acesso externo à EEPROM habilitado.
_DATA_CP_OFF: Para acesso externo à EEPROM desabilitado.
_PWRTE_ON: Para Power Up ligado.
_PWRTE_OFF: Para Power Up desligado.
_WDT_ON: Para WatchDog ligado.
_WDT_OFF: Para WatchDog desligado.
_LVP_ON: Para sistema de programação em baixa tensão ativado.
_LVP_OFF: Para sistema de programação em baixa tensão desativado.
_MCLRE_ON: Para Master Clear externo ativado.
_MCLRE_OFF: Para Master Clear externo desativado.
_RC_OSC_CKOUT: Para RC externo com saída CKOUT.
_RC_OSC_NOCKOUT: Para RC externo sem saída CKOUT (com I/O).
_INTOSC_OSC_CKOUT: Para oscilador interno com saída CKOUT.
_INTOSC_OSC_NOCKOUT: Para oscilador interno sem saída CKOUT (com I/O).
_EXTCLK_OSC: Para clock externo sem saída CKOUT (com I/O).
_LP_OSC : Para oscilador tipo LP.
_XT_OSC : Para oscilador tipo XT.
_HS_OSC : Para oscilador tipo HS.

A combinação destas opções deve ser feita por intermédio do operador & ("E").

Desta forma, a sintaxe da diretriz __CONFIG é a seguinte:

__CONFIG _CP_ON & _PWRTE_ON & _WDT_OFF & _INTOSC_OSC_NOCKOUT

```
CBLOCK      0x20    ;ENDEREÇO INICIAL DA MEMÓRIA DE
                    ;USUÁRIO
CONTADOR1
CONTADOR2

ENDC                ;FIM DO BLOCO DE MEMÓRIA
```

Definição das variáveis utilizadas pelo programador e o local onde tais variáveis estarão. Normalmente é bom definirmos esses endereços já que podem variar de PIC para PIC.

```
CBLOCK      0x20    ; Endereço inicial disponível para o PIC16f628A
CONTADOR1           ; variável
CONTADOR2           ; variável
```

```
ORG     0x00
GOTO INICIO
```

A diretiva ORG define em qual linha do programa estamos,
em geral o início do processamento que é onde o programa começa.

Vale acrescentar que, quando acontece uma interrupção, o programa é
Imediatamente encaminhado para o endereço 0x04.

Como nesse primeiro programa não iremos tratar de nenhuma
Interrupção, usamos a instrução GOTO seguido do label, para
nos encaminharmos para a primeira instrução do programa.

```
INICIO
        CLRF    PORTA
        CLRF    PORTB
```

Esta instrução é o "CLeaR File" clrf é o mnemônico coloca zeros num registro inteiro. No pic os registros são de 8 bits, isto é um byte, a Microchip chama esses registros de file, então as instruções do pic que se referem a registros de memória possuem sempre a letra f.

(porta ou portb) é o argumento dessa instrução, é nome do file que a instrução vai "encher de zeros", na verdade esse argumento tinha que ser um número, o número do endereço da memória ou registro, mas com aquele arquivo do "include", 16f628a.inc, podemos escrever portb ou porta, que isso equivale a 06, depois vamos ver isso...

O menemônico portb se refere então ao registro de memória que controla uma porta de entrada e saída, a PORTa B, do PIC 16F628A.

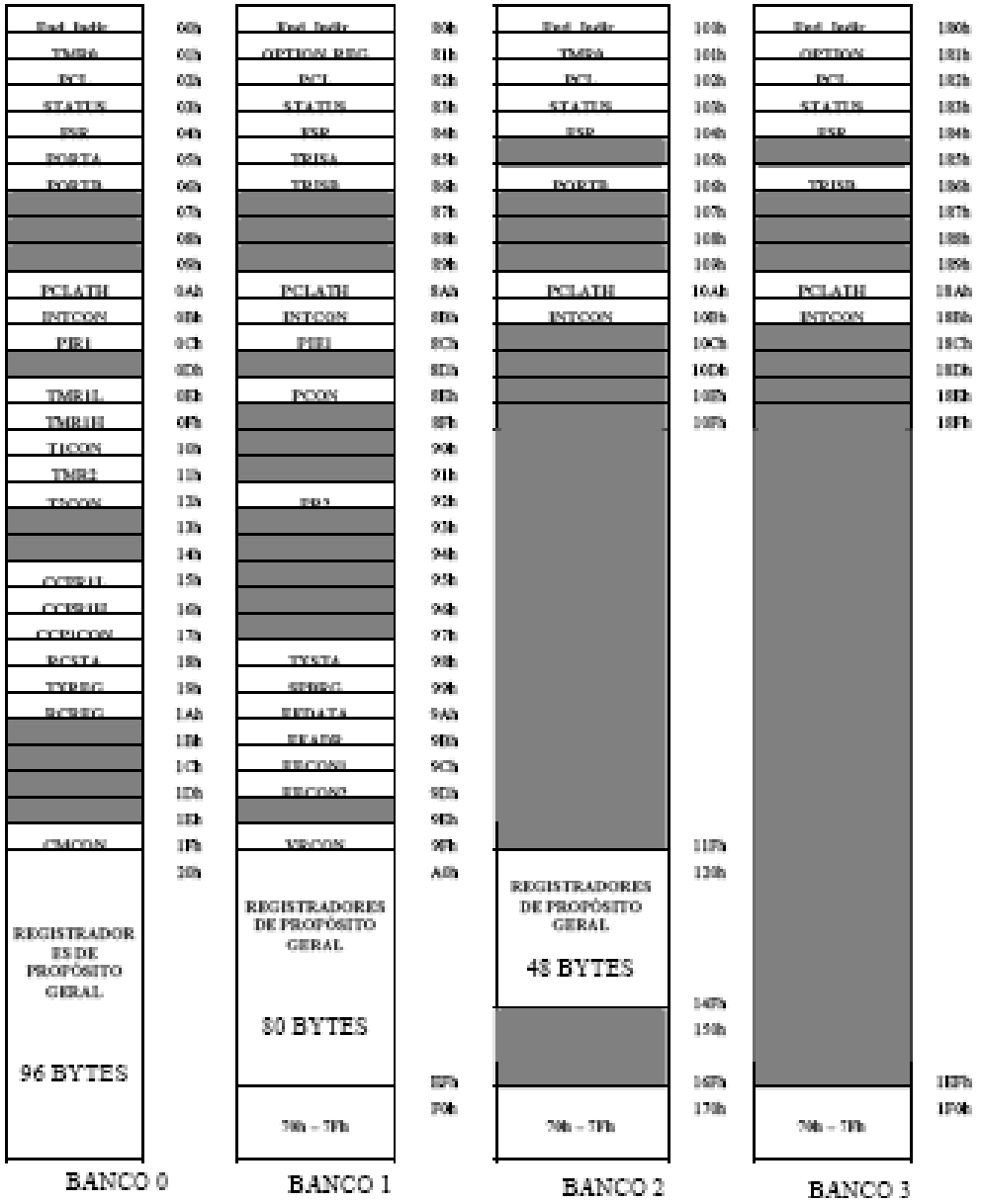
BSF STATUS, RP0

Esta instrução é o "Bit Set File" bsf faz com que um único bit pertencente a um registro seja alterado para 1.

O status e o rp0 são argumentos da instrução, a instrução tem a seguinte sintaxe: bsf f,b onde o f é o endereço do registro (file) e b é o número do bit a ser alterado, como podemos trabalhar com mnemônico, não precisamos decorar números, então status é um registro especial no pic em que cada bit está relacionado com uma parte do hardware do microcontrolador, depois vamos estudar esses registros;

O rp0 é o nome de um bit dentro do status, que altera o banco de memórias de dados que o PIC vai endereçar, no pic16f628A temos 4 bancos e o RPO=0 endereça o banco 0, e quando RP0=1 endereça para o banco 1, depois explicaremos melhor o que é isso.

Tabela 2 – Mapa da memória de dados do PIC16F628



CLRF TRISB

Esta instrução é o "CLeaR File" clrf faz com que todos os bits de um file vão pra zero, o trisb é o argumento da instrução, no caso o nome de um file ou registro especial do pic, que controla todos os pinos da porta B, ou portb.

O registro TRIS indica a direção com que os dados irão fluir (ou seja, o pino será uma entrada ou saída).

Então, esta instrução vai garantir que no início do programa todos os pinos da porta B serão saídas. Se colocarmos um 1 em algum desses bits, o pino será uma entrada.

Vale acrescentar que TRIS está no banco 0 da memória.

BCF STATUS, RP0

Esta instrução é o "Bit Clear File" bsf faz com que um único bit pertencente a um registro seja alterado para Zero.
O status e o rp0 são argumentos da instrução, a instrução tem a seguinte sintaxe: bsf f,b onde o f é o endereço do registro (file) e b é o número do bit a ser alterado, verifique que é o inverso de bsf.

Com esta instrução voltamos para o banco Zero de dados.

BSF PORTB, 1

Já vimos a instrução bsf, aqui ela manda o bit 1 do file portb ir para nível lógico 1, no nosso circuito esse bit 1
do file portb é o RB1, que está ligado no led.

Analisando o circuito elétrico, esse nível 1, leva o pino (7) à 5V o que vai fazer ACENDER O LED.

BCF PORTB, 1

Já vimos esta instrução também, o bcf, aqui manda o bit 1 do file portb ir para nível lógico zero.
Analisando o circuito elétrico, esse nível zero, leva o pino (7) à 0V o que vai fazer APAGAR O LED.

GOTO REPETE

Instrução de desvio incondicional para o label repete do programa.

END

Esta é uma informação ao compilador dizendo que o código fonte terminou.
Não é instrução do PIC.

Agora vamos digitar, compilar e gravar no PIC

Utilizaremos o winpic800 para a gravação e o gravador JDM serial

Por que o led não pisca e só fica aceso ?

Resp.

Porque estamos trabalhando a 4MHz e não a 1 Hz !

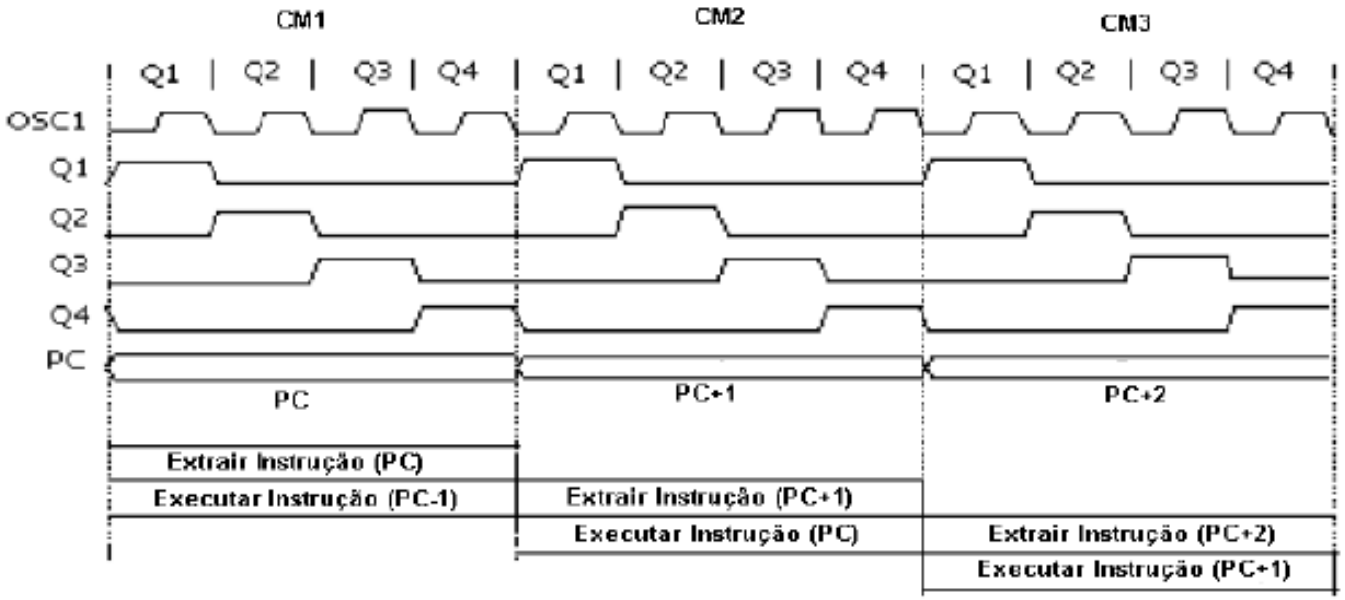
Compare com o programa original

Explicação do programa inicial

CICLO DE MÁQUINA

O oscilador externo (geralmente um cristal) ou o interno (circuito RC) é usado para fornecer um sinal de clock ao microcontrolador. O clock é necessário para que o microcontrolador possa executar as instruções de um programa.

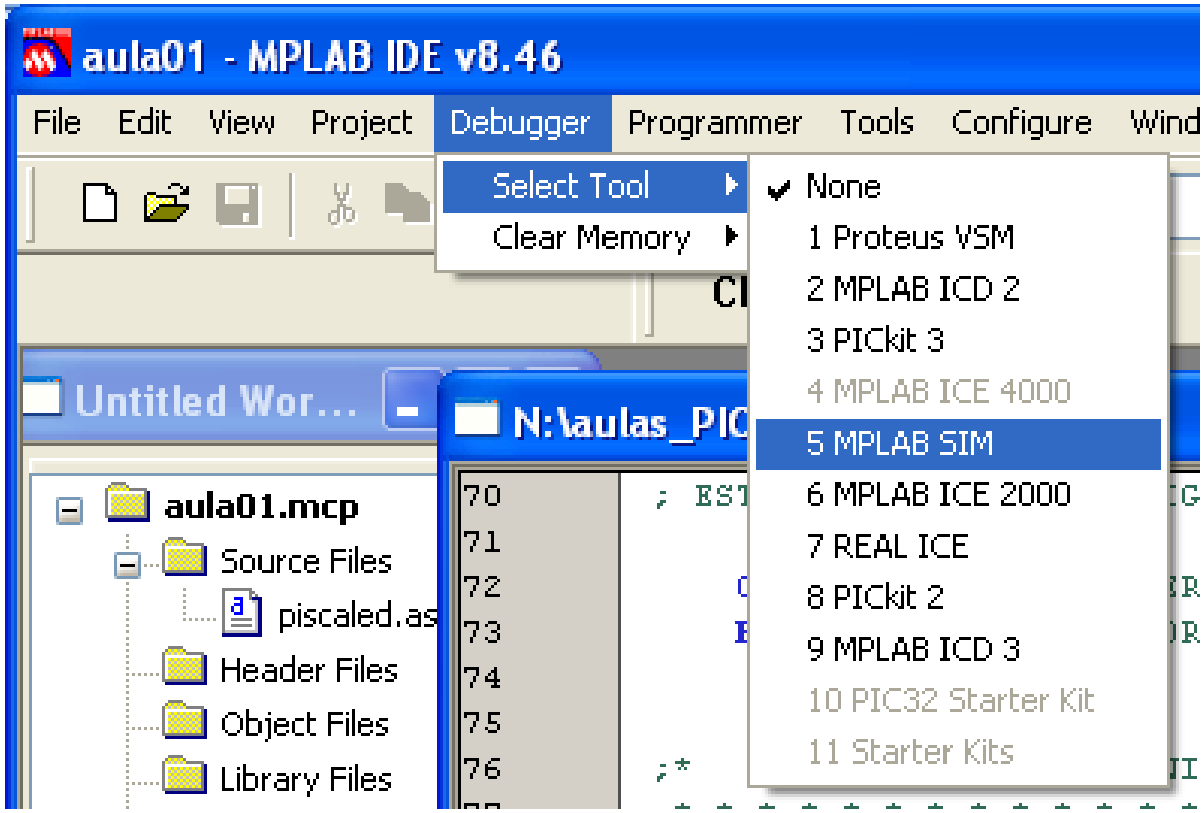
Nos microcontroladores PIC, um ciclo de máquina (CM) possui quatro fases de clock que são Q1, Q2, Q3 e Q4. Dessa forma, para um clock externo de 4MHz, temos um ciclo de máquina (CM=4 x 1/F) igual a 1µs (ou 1MHz)



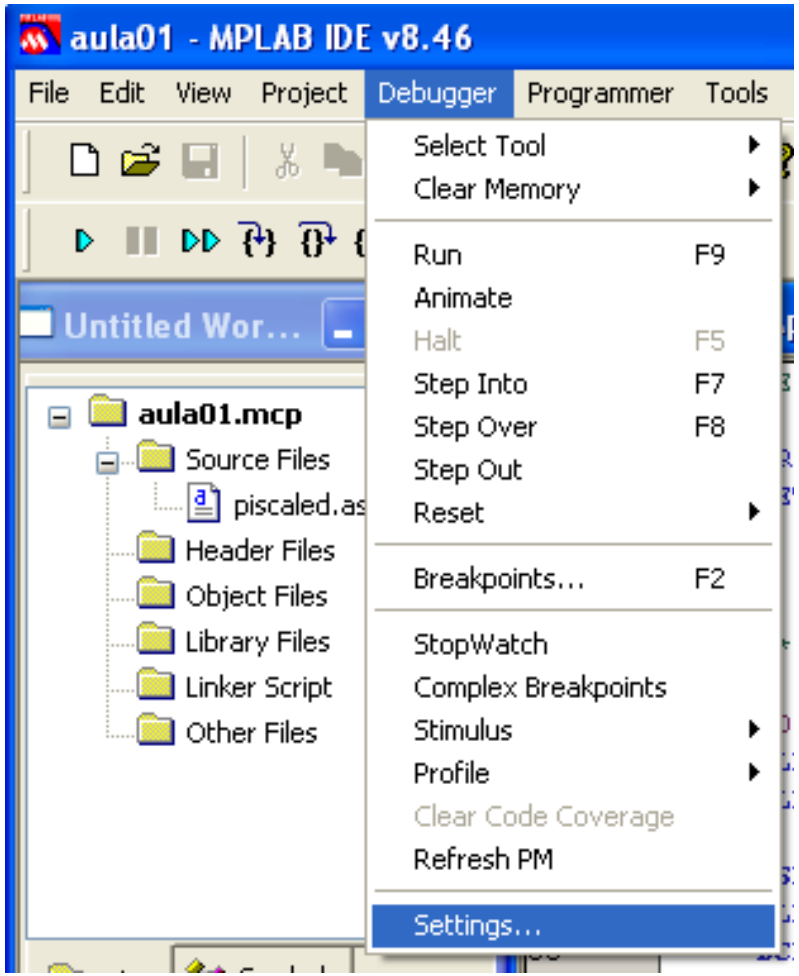
•O Contador de Programa (PC) é incrementado na fase Q1 do ciclo de máquina e a instrução seguinte é resgatada da memória de programa e armazenada no registro de instruções da CPU no ciclo Q4.
Ela é decodificada e executada no próximo ciclo, no intervalo de Q1 e Q4.
•O PIPELINE (sobreposição) permite que quase todas as instruções sejam executadas em apenas um ciclo de máquina, gastando assim 1 μs (para um clock de 4 MHz).
•As únicas exeções referem-se às instruções que geram “saltos” no contador de programa, como chamadas de funções em outro local da memória de programa e os retornos dessas funções.

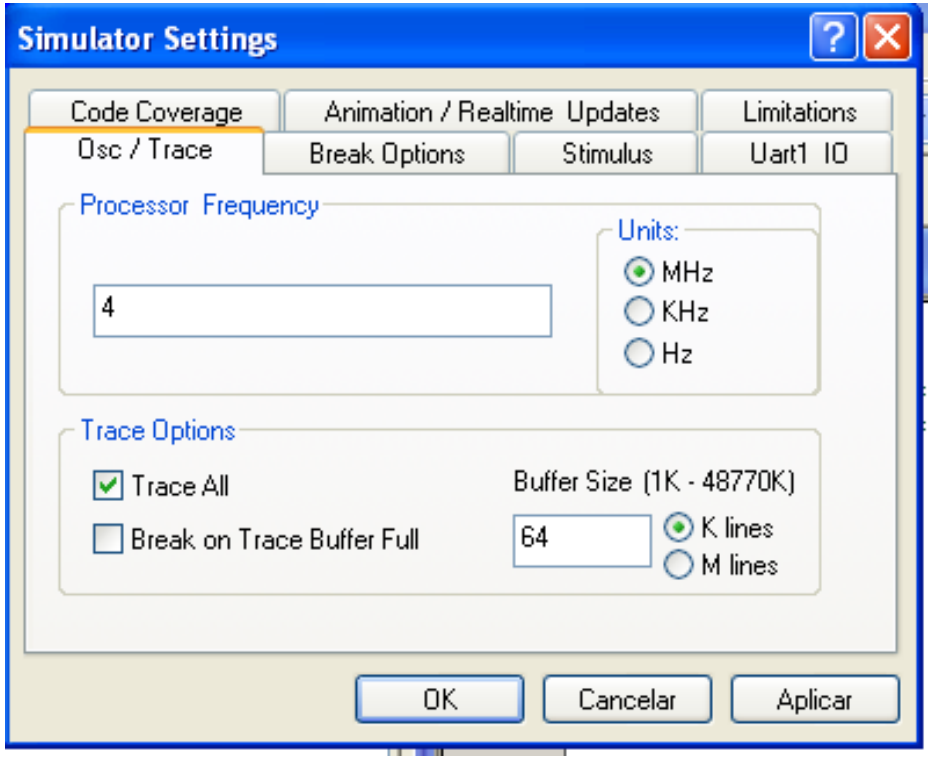
Analisando o programa

Selecionando o simulador MPLAB SIM

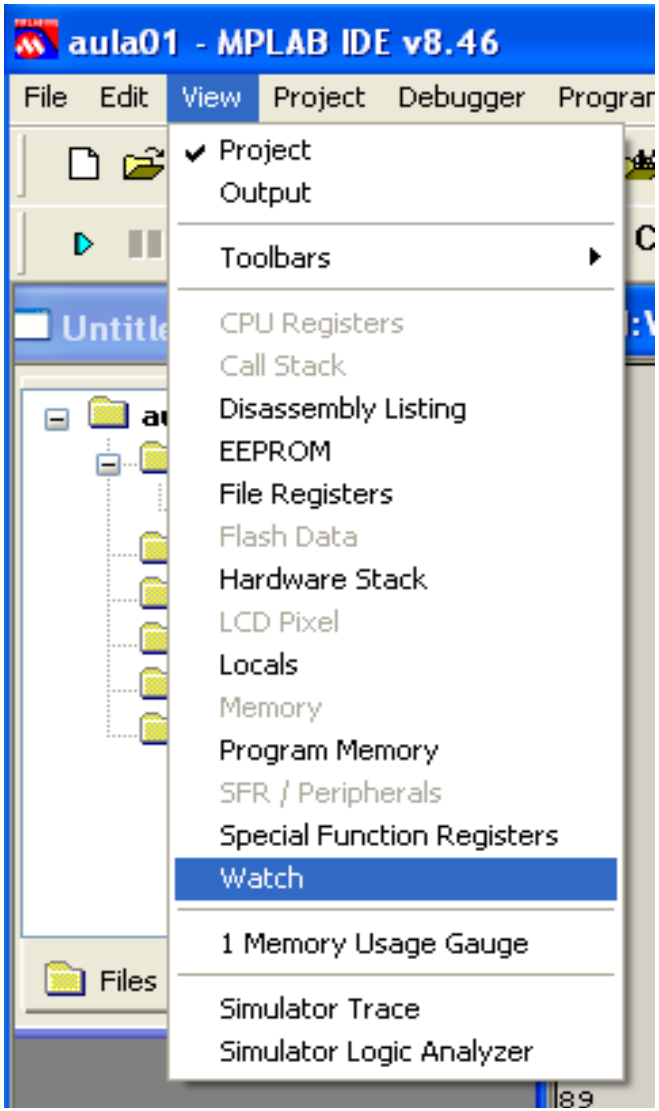


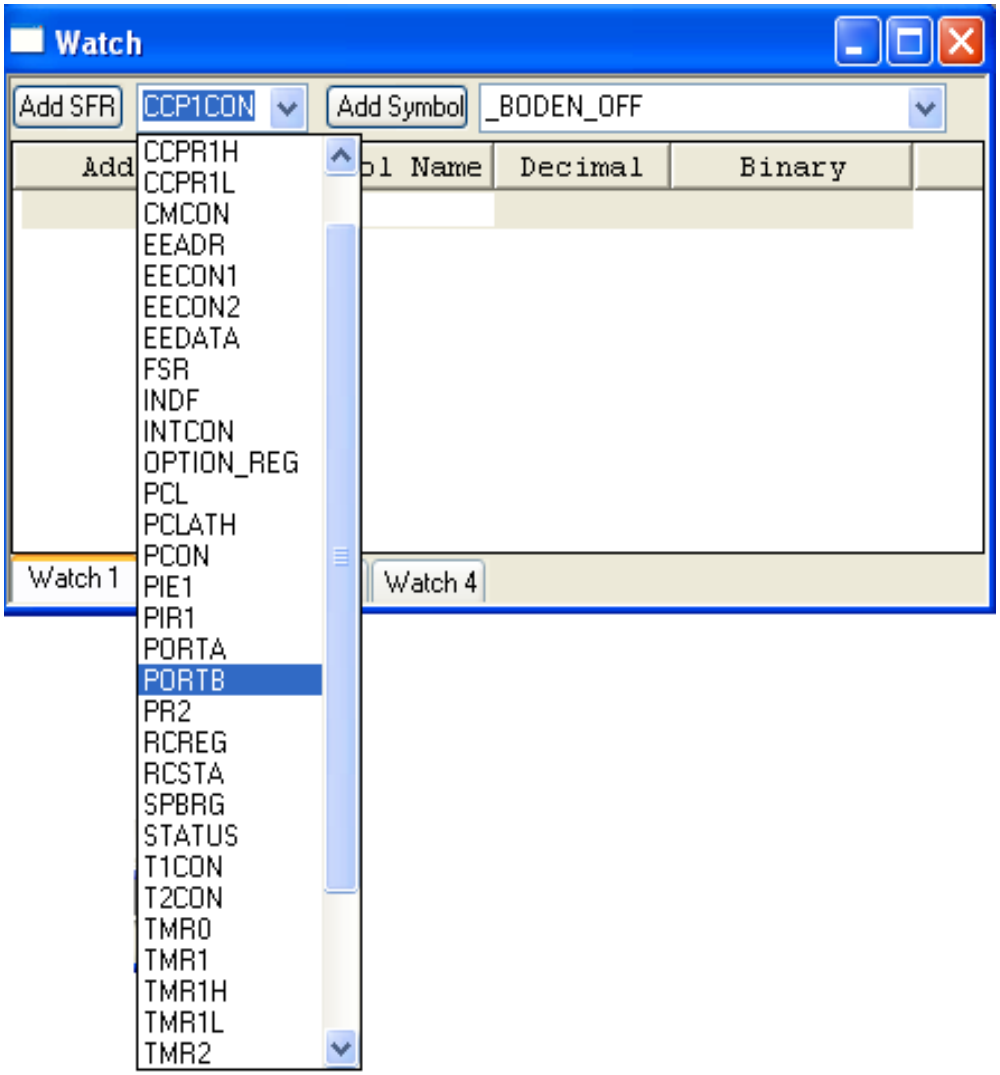
Alterando a freqüência de operação para 4 MHz
(ciclo de máquina de 1 MHz).



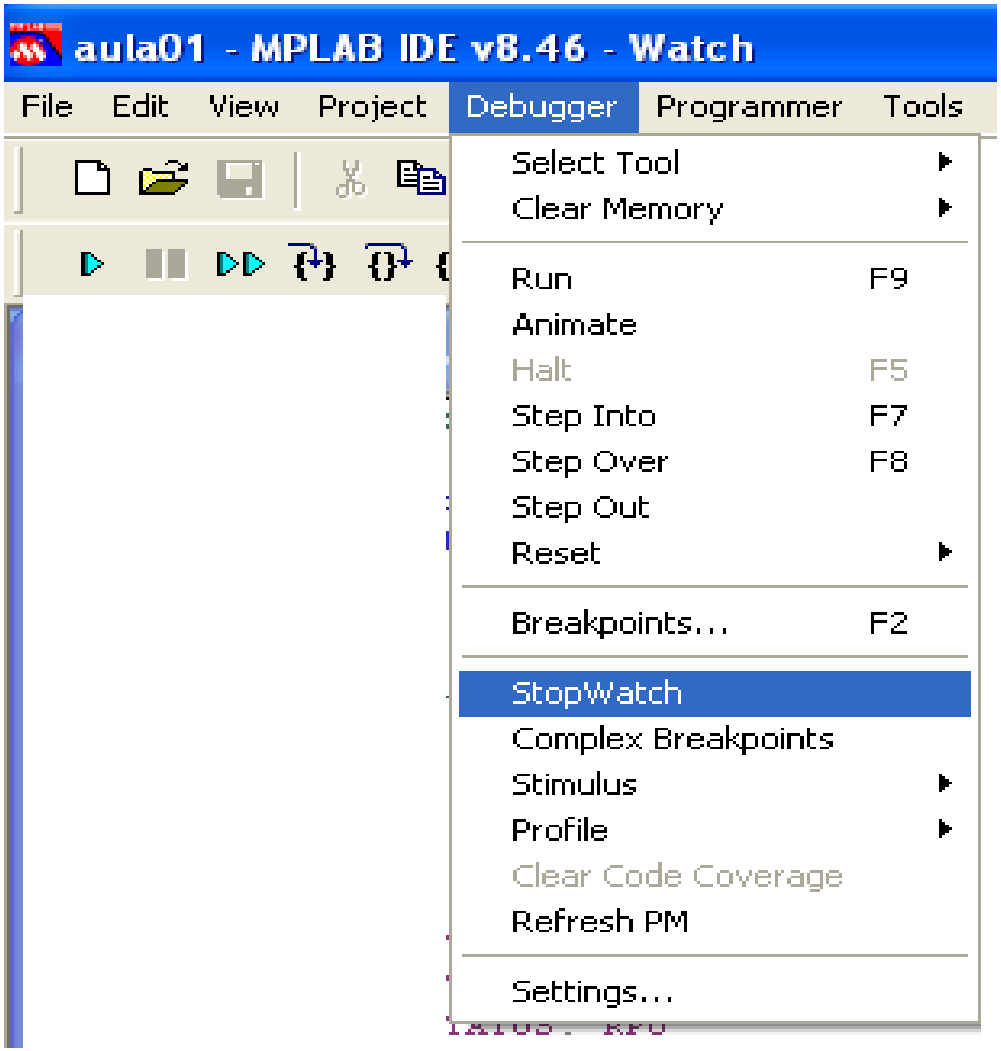


Adicionando elementos (registradores) a serem observados (PORTB, STATUS, TRISB)

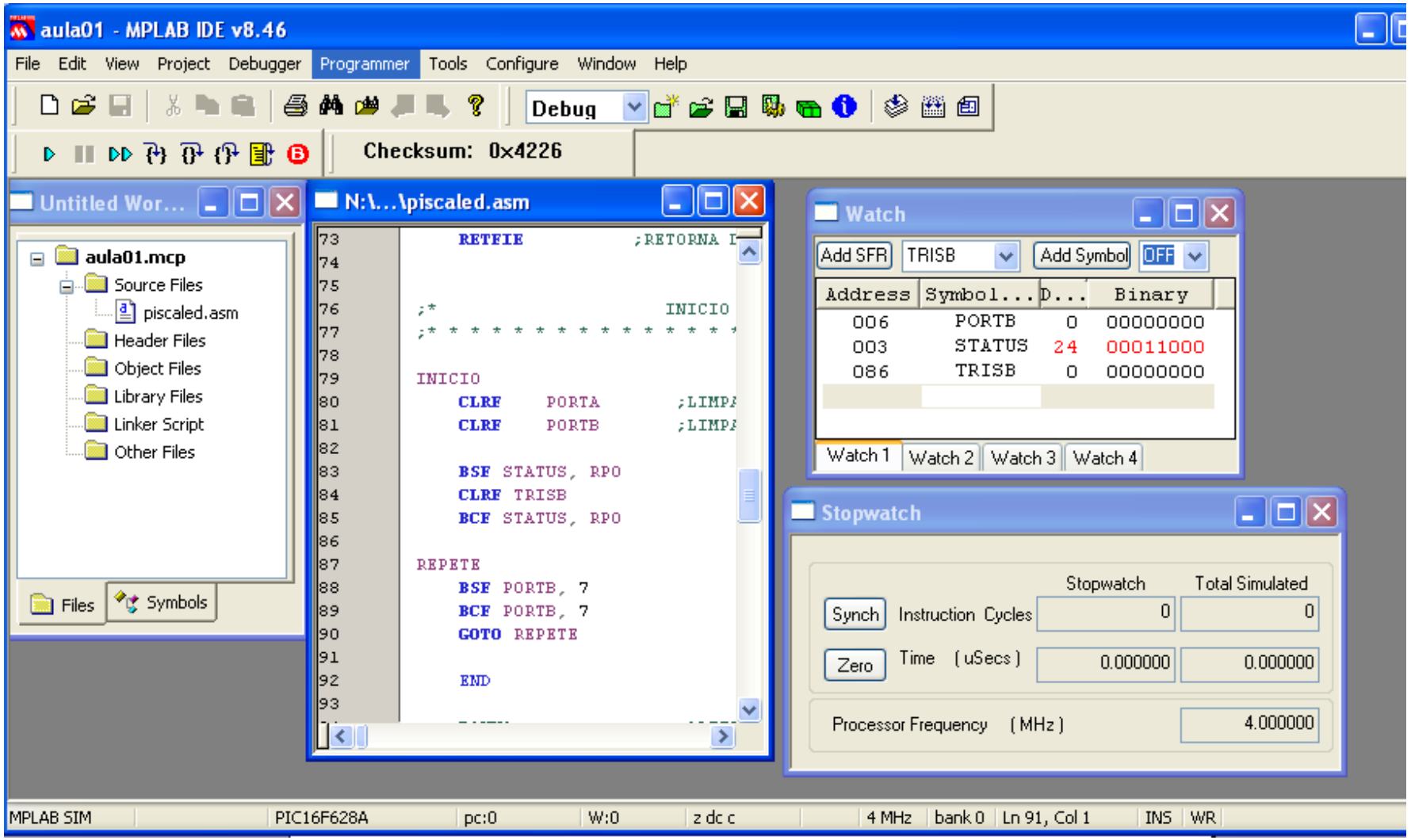




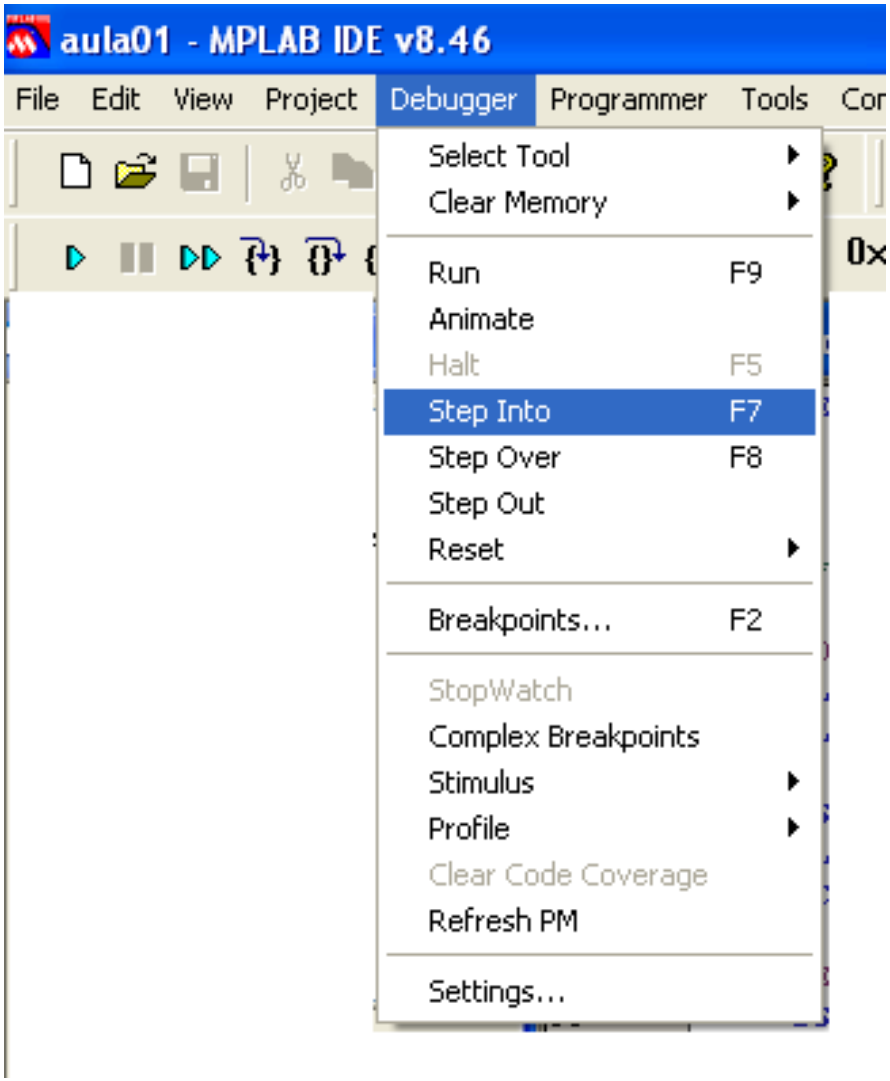
Adicionando o medidor de tempo



Tela final para a análise



Executar o programa linha a linha e
observar o tempo necessário para cada instrução.



Novas instruções

CALL k ; Chamada a uma subrotina

Descrição: Chama uma subrotina memorizada no endereço k. O parâmetro k pode ser especificado utilizando-se diretamente o valor numérico do endereço ou então o relativo label.

Exemplo:

```
#define LED1 1
org 00H
call ledOn ; Chama a rotina ledOn
ledOn
btfsc PORTB,LED1 ; testa o bit 1 da porta B
return
```

Quando a CPU do PIC encontra uma instrução CALL, memoriza no STACK o valor do registrador PC+1 de modo a poder retornar para instrução após o CALL, em seguida escreve no PC o endereço da subrotina pulando a execução desta ultima. O valor original do PC será recuperado pela subrotina com a execução da instrução de retorno RETURN ou RETLW.

No PIC16F628 estão disponíveis 8 níveis de stack (pilha), ou seja a instrução CALL dentro de uma subrotina pode ter no máximo 8 chamadas ou 8 níveis. As demais chamadas serão sobrepostas às primeiras.

Observe o programa PISCALED1.ASM.
Compile a avalie o tempo de execução.

```
REPETE
        BSF PORTB, 1
        CALL  ATRASO
        BCF PORTB, 1
        CALL  ATRASO
        GOTO REPETE

ATRASO
        NOP
        RETURN

        END
```

Observe o programa PISCALED2.ASM.
Compile a avalie o tempo de execução.

Responda: Quantos NOP's estão presentes?

```
REPETE
        BSF PORTB, 1
        CALL  ATRASO
        BCF PORTB, 1
        CALL  ATRASO
        GOTO REPETE

ATRASO
        NOP
        ...
        RETURN

        END
```

O registrador de trabalho W

Os membros da família 16FXXX podem acessar tanto direta como indiretamente qualquer posição de memória RAM ou de registros internos, pois estão todos mapeados no mesmo bloco de memória.

Qualquer operação pode ser feita com qualquer registro (de dados ou de controle).

As operações lógicas e aritméticas são realizadas pela ULA (unidade lógica e a aritmética) que possui um registro próprio chamado W (Working register - popular acumulador),

Vamos usar muito esse registrador, que não está presente na RAM e não é acessado por endereçamento. A ULA é de 8 bits e permite realizar somas, subtrações, deslocamento (shifts) e operações lógicas.

MOVLW k ; Copia para W o valor constante k

Descrição: Passa ao acumulador W um valor constante k.

Exemplo:

org 00H

movlw 20

Após ter executado este programa o acumulador W irá a 20.

MOVWF f ; Copia o conteúdo do registrador W para o registrador F

Descrição: Esta instrução copia o conteúdo do registrador W no registrador de parâmetro f.

Exemplo: Para copia o valor 10H no registrador TMR0. A instrução a se executar será a seguinte:

movlw 10H ;Escreve no registrador W o valor 10H

movwf TMR0 ;e o memoriza no registrador TMR0

NOP ; Nenhuma operação

Descrição: Esta instrução não executa nenhuma operação mas é útil para inserir atrasos de um ciclo de maquina ou mais.

Exemplo:

```
nop
nop
```

DECFSZ f,b ; Decrementa o valor do registrador f e pula a próxima instrução se o resultado for zero.

Descrição: Decrementa o valor de registrador do endereço f e se o resultado for zero pula a próxima instrução. **Exemplo**:

```
counter equ 0CH
org 00H
movlw 10 ;counter = 10
movwf counter
loop
```

Observe o programa PISCALED3.ASM.
Compile a avalie o tempo de execução.

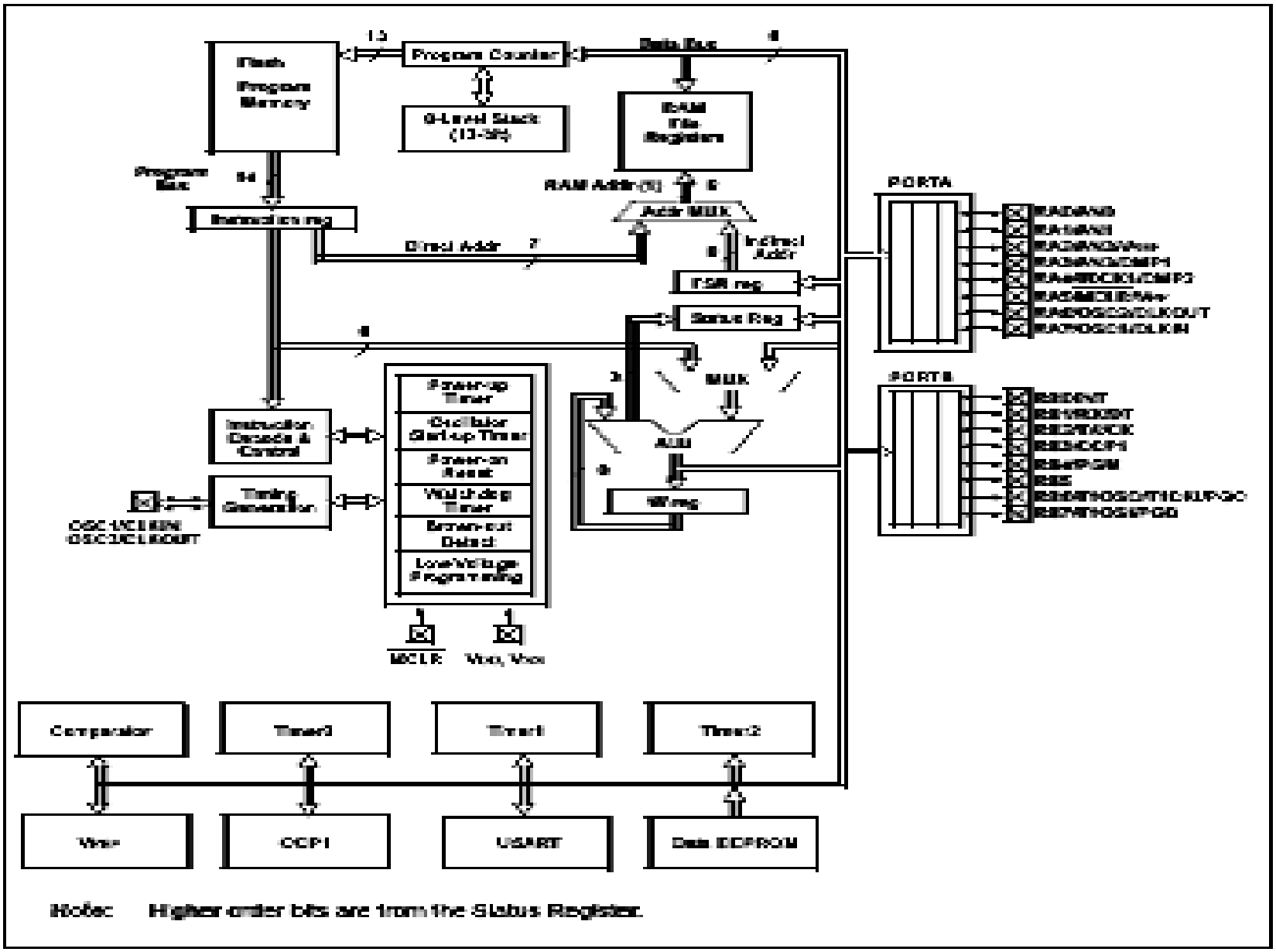
Responda: Qual o tempo máximo que podemos ter ?
- Altere o valor do CONTADOR1.

```
REPETE
        BSF PORTB, 1
        CALL ATRASO
        BCF PORTB, 1
        CALL ATRASO
        GOTO REPETE


ATRASO
        MOVLW   10
        MOVWF   CONTADOR1
ATRASO1
        DECFSZ  CONTADOR1
        GOTO    ATRASO1

        RETURN


END
```

Relatório a ser apresentado:

Implementar rotinas de delay de 0.5 segundo e 1 segundo.

Dica: Coloque um contador dentro de outro para multiplicar a contagem



Note: Higher order bits are from the Status Register.

Programa em C (compilador CCS)

```
#include <16F628A.h>

#FUSES NOWDT                    //No Watch Dog Timer
#FUSES INTRC_IO                 //Internal RC Osc, no CLKOUT
#FUSES NOPUT                    //No Power Up Timer
#FUSES NOPROTECT                //Code not protected from reading
#FUSES NOBROWNOUT               //No brownout reset
#FUSES NOMCLR                   //Master Clear pin used for I/O
#FUSES NOLVP                    //No low voltage prgming, B3(PIC16) or B5(PIC18) used for I/O
#FUSES NOCPD                    //No EE protection

#use delay(clock=4000000)
#use rs232(baud=9600,parity=N,xmit=PIN_B2,rcv=PIN_B1,bits=8)

void main()
{
        int tempo=500;
        while(true)
        {
                output_high(PIN_B0);
                delay_ms(tempo);
                output_low(PIN_B0);
                delay_ms(tempo);
        }
}
```