Anais do XXXIV COBENGE. Passo Fundo: Ed. Universidade de Passo Fundo, Setembro de 2006.
ISBN 85-7515-371-4

# LBM-UMA PROPOSTA DE LINGUAGEM BÁSICA ESTRUTURADA PARA PROGRAMAÇÃO DE MICROCONTROLADORES NOS CURSOS DE ENGENHARIA.

**Fretz Sievers Junior** - fretz@comp.ita.br
ITA–Instituto Tecnológico de Aeronáutica , Departamento de Eng. Elet. e Computação
Pç Marechal Eduardo Gomes, n 50 – Campus do CTA, 12228-900, São José dos Campos – SP

**José Silvério Edmundo Germano**, silverio@fis.ita.br
ITA–Instituto Tecnológico de Aeronáutica , Departamento de Física – IEFF
Pç. Marechal Eduardo Gomes, n 50 – Campus do CTA, 12228-900, São José dos Campos – SP

**Felipe de Almeida**, felal@comp.ita.br
ITA–Instituto Tecnológico de Aeronáutica , Departamento de Engenharia de Software
Pç. Marechal Eduardo Gomes, n 50 – Campus do CTA, 12228-900, São José dos Campos – SP

**Resumo:** *Este artigo descreve uma Linguagem de Programação estruturada com poucos comandos de programação para um microcontrolador PIC16F84 com uma linguagem de alto nível a fim de introduzir alunos de engenharia sem nenhum conhecimento em programação para microcontroladores. Através de um kit de experimentos, o aprendiz poderá realizar a montagem de vários circuitos simples é realizar a programação para controlar componentes eletrônicos, não trabalhando com registradores como é feito na linguagem assembler, pois a linguagem proposta encapsula essas funcionalidades, facilitando o aprendizado do aluno é diminuindo a complexibilidade da linguagem. A linguagem possui um editor de comandos para facilitar na implementação dos comandos, mas poderá ser utilizado um editor de texto para realizar a programação. A idéia é introduzir a lógica de linguagem de programação em alto nível com comandos simples para que o aluno possa ter maior facilidade em linguagens mais complexas como assembly.*

***Palavras-chave:*** *Linguagem de Programação, Microcontroladores, Ensino a Engenharia*

## 1. INTRODUÇÃO

Nos cursos de Engenharia Elétrica, Eletrônica, Computação e Automação, abordam matérias de programação para microcontroladores. Alguns alunos quando começam a cursar matérias de microcontroladores sentem dificuldades referente a lógica dos comandos e os registradores do microcontrolador que são requisitos básicos para iniciar a programação.

A linguagem Assembly é vista para alguns alunos como uma linguagem complicada e de difícil assimilação pois requer muitos comandos para realizar uma determinada tarefa. O objetivo da linguagem proposta é retirar alguns fatores complicadores da linguagem assembler e incentivando o aluno a ter contato com uma linguagem mais básica o que não exige conhecimento da estrutura do microcontrolador, facilitando o seu aprendizado, e em seguida apresentar a linguagem assembly com seus comandos minemônicos e sua lógica, porém o aluno já terá uma base de programação para assimilar novos conhecimentos, facilitando seu aprendizado.

Este artigo apresenta um estudo com o objetivo de definir uma linguagem de programação estruturada para a programação de microcontroladores e um ambiente de execução. Em nosso estudo, iremos utilizar o microcontrolador PIC16F628A, sendo microcontrolador versátil, compacto e por suas características que serão mostradas na seção 5.0. É apresentada uma solução parcial denominada LBM (Linguagem Básica para microcontroladores), na qual comandos simples são definidos com as suas funções onde o ambiente de desenvolvimento ajuda o programador com a sintaxe dos comandos. O ambiente possui interpretador que converte os comandos da linguagem LBM para linguagem assembly e grava no microcontrolador. Através da linguagem proposta o programador não necessita saber sobre os registradores da linguagem, pois o interpretador analisará a sintaxe e a semântica e converterá para a linguagem assembler.

O restante do artigo esta organizada da seguinte forma: A seção 2 discute como desenvolver uma linguagem de programação, a seção 3 interpretador da linguagem, a seção 4 os comandos da linguagem, a seção 5 comenta a Interface do Editor, na seção 6 mostra o kit de experimentos implementado, na seção 7 o código implementado da linguagem LBM finalmente a seção 8 comenta sobre algumas conclusões e possíveis trabalhos futuros.

## 2. COMO DESENVOLVER UMA LINGUAGEM DE PROGRAMAÇÃO.

Criar uma linguagem de programação não é uma tarefa fácil. No início da implementação da linguagem LBM foi investigado uma lista de discussão [SULLIVAN, 2006] que ajudou na realização de alguns questionamentos tais como:

1. Como o programador irá se sentir diante da linguagem?

2. O programador deverá se adaptar a linguagem ou a linguagem deverá ser algo natural para ele?

3. A produtividade do programador e um fator importante?
4. O código deverá ser portável?

A linguagem proposta foi desenvolvida, visando ser o mais natural possível da linguagem humana, pois implementamos comandos que são muito utilizados e fáceis de visualizar no microcontrolador tais como portas, se porta, vai, nível alto, nível baixo, etc. A produtividade e um fator importante, pois diminuindo sua preocupação em gerenciamento de memória e registradores facilitam na implementação do código fonte. Acreditamos que uma das qualidades de uma linguagem de programação e sua portabilidade e produtividade. Criando uma linguagem que seja portável para vários microcontroladores, deixamos a escolha do aluno, qual microcontrolador se encaixa a suas necessidades e com isso ajudando em seu projeto e na sua aprendizagem.

## 3. COMANDOS DA LINGUAGEM.

Nesta primeira versão foram criados 7 comandos para a linguagem LBM, apesar do número reduzido de comandos, podemos criar várias aplicações. Na seção 7 iremos mostrar um código de exemplo da linguagem LBM. Na tabela 1.0 apresentamos a lista de comandos da linguagem LBM.

| Comando | VAR1 | VAR 2 | Função |
|---|---|---|---|
| PORTA | Indica a porta de 0 a 7 | Estado lógico da Porta: Alto=1(próximo da tensão de alimentação) Baixo=0 (Próximo a 0v) | Acionar o nível lógico de (0/1) a uma determinada porta |
| SE PORTA | Indica a porta de 0 a 7 | Sinal lógico de entrada: ALTO=1 (Se sinal 1 pula para a próxima instrução) BAIXO=0 (se sinal 0 pula para a próxima instrução) | Compara o nível lógico, com o aplicado na porta, sendo iguais pulam para a próxima instrução. |
| ESPERA | Valores de tempo | Estabelece a grandeza onde: Mili s = 0,001 Segundos Micro s= 0,000001 Segundos | Espera durante o tempo selecionado |
| VAI | Linha de 01 a 64 | - | Continua a execução na linha determinada |
| REPETE | Número de vezes 01 a 15 | - | Repete o número de vezes, os comandos existentes entre o comando repetem e o termine (limite de 7 comandos um dentro do outro) e similar ao comando while |
| TERMINE | - | - | Determina o fim do conjunto de comandos a serem repetidos. |
| FIM | - | - | Fim da execução do programa |

Tabela 1.0 – Lista de comandos da linguagem LBM.

## 4. INTERFACE DO EDITOR

O editor LBM versão 1.5 possui 3 combo box para o programador implementar seu programa. A figura 1.0 mostra a interface do Editor LBM V1.5.

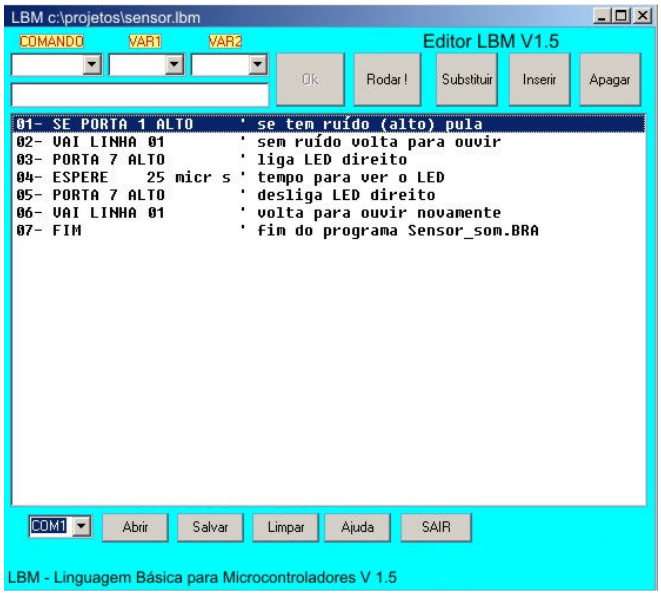

Figura 1.0 – Editor LBM V 1.5

Abaixo descrevemos a interface de entrada de dados:

**Comando**: Local onde se encontram os comandos da linguagem LBM. A linguagem possui 7 comandos: Porta, Se Porta, Espera, Vai Linha, Repete, Termine e Fim. Para cada comando escolhido, o combo box VAR 1 e VAR 2 aparecem opções diferentes ou podem ser desabilitados de acordo com a escolha do comando.

**VAR 1**: Neste combo box, o programador poderá escolher o número da porta do microcontrolador, tempo, número de linha do programa ou quantidade de repetições. Essas opções variam de acordo com o comando escolhido.

**VAR 2**: Neste combo box, o programador poderá assumir os níveis da porta (auto e baixo), a unidade de tempo (micro segundos ou mili segundos).

**Comentários**: Permite adicionar comentários nos comandos inseridos.

Para inserir uma linha de programa, o programador deverá escolher o comando que deseja adicionar e escolher as opções que aparecem no combo box VAR1. Exemplo caso seja escolhido o comando *Se Porta,* as opções de VAR1, será o número da porta e as opções de VAR2 será se o nível e baixo e alto como mostra a figura 2.0.

Figura 2.0 – Inserindo uma linha de comando.

A seguir, descrevemos os botões da interface:

**Rodar**: permite que o programa seja transferido para o Kit de experimentos para o microcontrolador PIC16F84 para realizar o controle nos dispositivos eletrônicos.

**Substituir:** permite substituir uma linha de programa selecionado por uma outra inserida pelo programador.

**Inserir**: Permite inserir uma linha de programa selecionado.

**Apagar**: Permite apagar uma linha de programa selecionado.

**Abrir**: Permite escolher um código para ser carregado na interface do programa.

**Salvar**: Permite salvar o programa desenvolvido. O programa recebe a extensão LBM.

**Limpar**: Limpa a interface para inserir um novo programa. Atua como um comando novo.

**Sair**: Sai da interface.

**Ajuda**: Mostra um descritivo com todos os comandos da linguagem.

## 5. KIT DE EXPERIMENTOS PARA TESTE DA LINGUAGEM LBM.

A grande vantagem da família PIC é que todos os modelos possuem um set de instruções bem parecido, assim como mantém muitas semelhanças entre suas características básicas [PEREIRA, 2002]. Desta forma ao implementarmos a linguagem LBM no PIC16F628A, poderemos implementar em outros microcontroladores da Microchip como e o caso dos microcontroladores 16F627 e 16F628, pois a linguagem proposta LBM, o programador não necessita saber quais são os novos registradores, interrupção de Timer e interrupção de fim de escrita de EPROM, pois quem esta responsável por essa tarefa e o interpretador da linguagem LBM. Estas características são interessantes a engenheiros nas áreas de computação, elétrica e automação, porém a utilização da linguagem LBM e para introduzir a lógica de programação e não a estrutura do microcontrolador. Neste artigo utilizamos o microcontrolador PIC16F628A que possui as seguintes características [SOUZA,2003]:

- Microcontrolador de 18 pinos, o que facilita a montagem de hardwares experimentais.
- 13 portas configuráveis com entrada e saída
- 4 interrupções disponíveis (TMR0, Externa, Mudança de Estado e EEPROM)
- Memória de programação Flash, que permite a gravação do programa diversas vezes no mesmo chip, sem a necessidade de apagá-lo por meio de luz ultravioleta, como acontece nos microcontroladores de janela.
- Memória EEPROM não volátil Interna;
- Via de Programação de 14 bits e 35 instruções.

Na figura 3.0 e mostrado a implementação do circuito para a utilização do PIC 16F84.

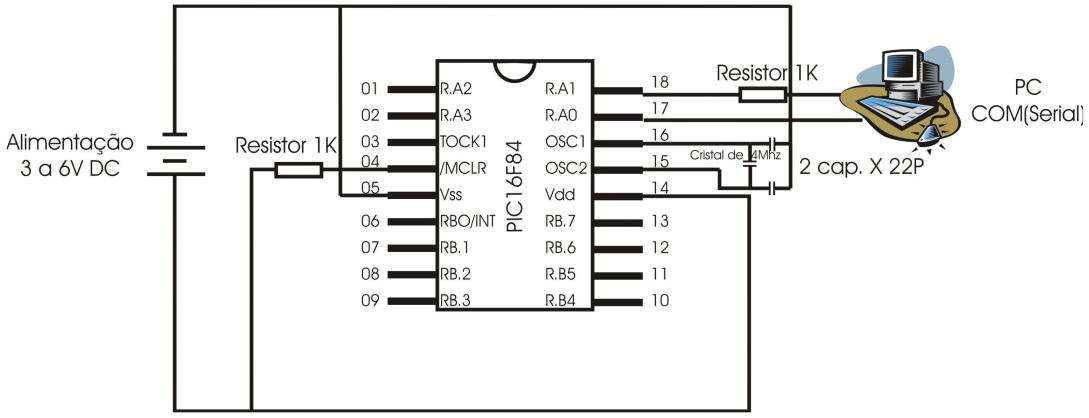


Figura 3.0 – Circuito básico para o funcionamento do PIC16F84.

A tabela 2.0 descreva a função de cada pino do PIC16F84. Maiores informações poderão ser encontradas em [MICROCHIP, 2006].

| Pino | Nome | Função | Observação |
|---|---|---|---|
| 1 | RA2 | Status do proc. do PIC16F84 por som | Uso restrito do sistema |
| 2;3 | NC | Nada Conectado | Nada deverá ser conectado |
| 4 | MCLR | Reset do PIC16F84 | Acionando em zero |
| 5 | -V | Pólo negativo da fonte de alimentação 3 à 6V DC | Pólo Negativo. Não inverter. |
| 6 | P0 | Led Indicativo esquerdo | Porta0 representado pela cor preta |
| 7 | P1 | Sensor/ Emissor de som | Porta1 representado pela cor marrom |
| 8 | P2 | Motor Esquerdo | Porta2 representado pela cor vermelho |
| 9 | P3 | Led amarelo Emissor / Sensor de Sombra | Porta3 representado pela cor laranja |
| 10 | P4 | Sensor/ Emissor faixa esquerdo | Porta4 representado pela cor amarelo |
| 11 | P5 | Motor Direito | Porta5 representado pela cor verde |
| 12 | P6 | Sensor/ Emissor faixa Direito | Porta6 Sensor/ Emissor de Faixa Direito |
| 14 | +V | Polo positivo da fonte de alimentação 3 à 6V DC | Polo positivo, não inverter |
| 15 | OSC | Circuito de geração de clock( Oscilação) | Cristal de 4Mhz e Cap. De Disco de 22P |
| 16 | OSC | Circuito de geração de clock( Oscilação) | Cristal de 4Mhz e Cap. De Disco de 22P |
| 17 | PCabo | Porta de com. entre o PC e o PIC16F84 | Fio branco do cabo |
| 18 | PCabo | Porta de com. entre o PC e o PIC16F84 | Fio Vermelho do cabo |

Tabela 2.0 – Funções dos pinos do microcontrolador.

Na tabela 2.0 podemos observar que o microcontrolador PIC16F84 está conectado a vários dispositivos nas portas P0 a P7. Esses dispositivos estão conectados para podermos realizar os testes na linguagem LBM para desenvolvimento de aplicações. A figura 4.0 mostra kit de experimentos. Este kit de experimento possui todos os componentes descritos na tabela 2.0. Na próxima seção veremos um comparativo com a linguagem assembly e a linguagem LBM.

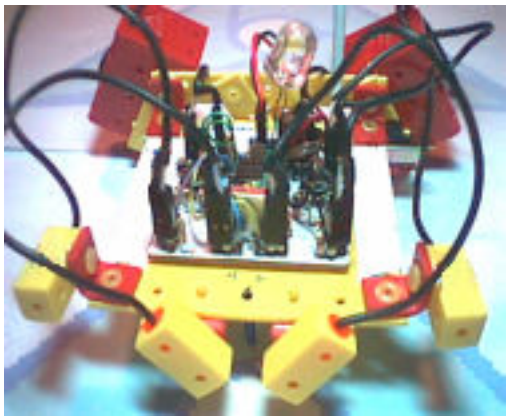

Figura 4.0 – Kit de experimento  para testar a linguagem LBM.

## 6. O INTERPRETADOR DA LINGUAGEM.

A linguagem LBM interage com um interpretador para a linguagem assembly do microcontrolador específico. O princípio de compilação, ou seja, passando da linguagem natural LBM para a linguagem de máquina será realizado por um interpretador da linguagem assembler do microcontrolador escolhido. A figura 5.0 apresenta esta arquitetura.

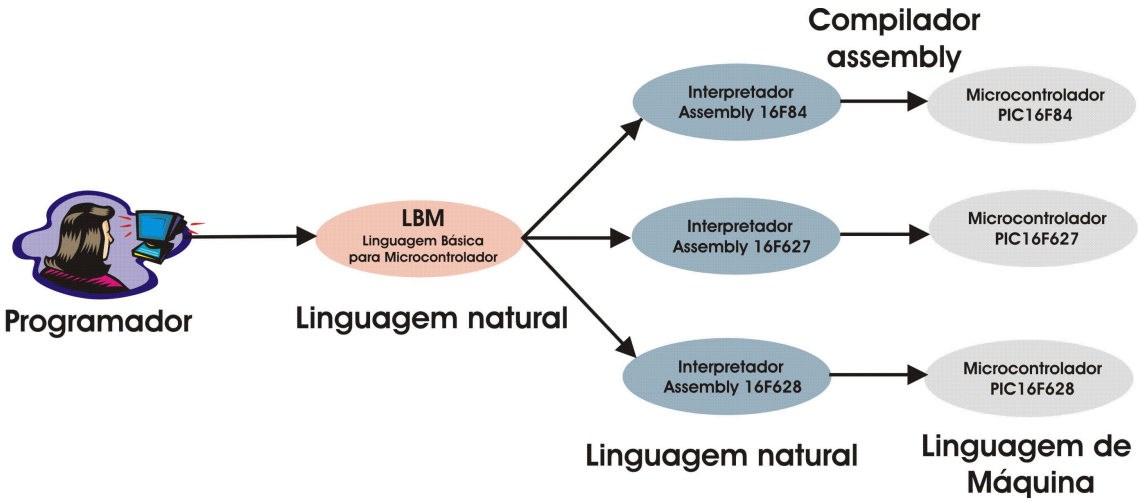


Figura 5.0 Arquitetura da linguagem LBM.

O interpretador passa por 13 estados. Foi determinado um autômato finito determinístico, segundo [HOPCROFT, 1939] pode ser definido por: A= (φ, ∑,δ, q0, F) onde A e o nome do autômato finito determinístico, φ e seu conjunto de estados, ∑ e seu conjuntos de símbolos de entrada, δ sua função de transição, F seu conjunto de estados de aceitação. Em nossa linguagem proposta fica: LBM = ( {inicio, arquivos de definições, paginação de memória, variáveis, flags internos, constantes, entradas, saídas, vetor de reset, inicio do programa inicialização das variáveis, rotina principal, fim do programa}, {0,1}, δ, inicio, fim do programa) diagrama de transição de estados na figura 2.0.

Na função de transição será aceita 0 quando ocorrer algum erro no estado corrente, portando o erro ao programador. Será aceito 1 quando o estado foi concluído com êxito passando para o próximo. A figura 6.0 mostra o diagrama de transição de estados.

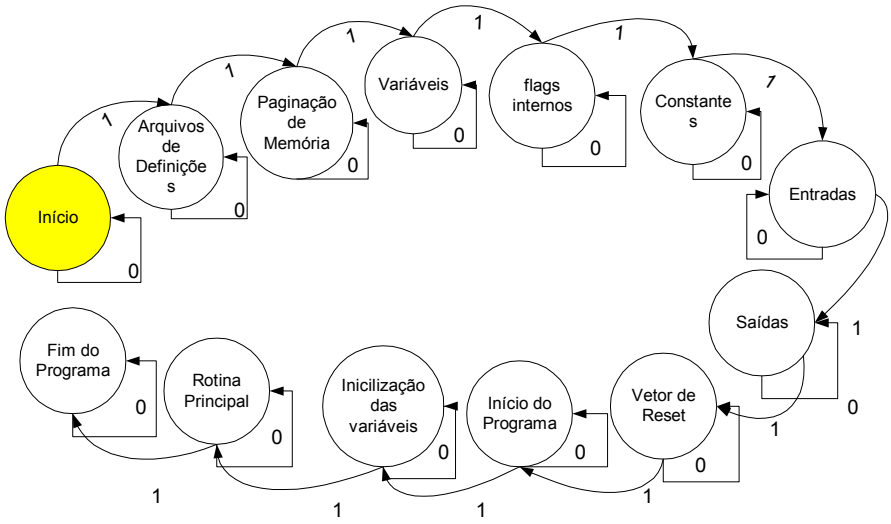


Figura 6.0 – Diagrama de transição de estados do interpretador da linguagem LBM.

Abaixo e definido cada um dos estados:

O primeiro estado é definido os arquivos de definição do microcontrolador. Cada microntrolador possui um arquivo de definição um exemplo e o microcontrolador PIC16F84 que possui o arquivo "P16F84.inc". Maiores detalhes pode ser vistos em [MICROCHIP,2006]

O estado de paginação de memória define os comandos de usuário para alteração da página de memória. O PIC possui dois bancos de memória para registradores SFR (Special Function Registers) [MARTINS,2005] que servem exatamente para guardar a configuração e o estado de funcionamento da máquina atual. A memória para variáveis do sistema e disponível somente no banco 0, quando queremos acessar algum registrador SFR que esta no banco 1, devemos primeiro informar ao sistema que queremos trabalhar com esse banco. Para tal deve-se alterar o valor do bit RP0 no registrador STATUS. Com o objetivo de tornar essa tarefa muito mais fácil, o interpretador assembler cria dois comandos virtuais chamados BANK0 e BANK1 que são definidos no arquivo gerado da seguinte maneira como mostrado na listagem 1:

```
# DEFINE    BANK0    BCF      STATUS, RP0    ; Seta bank 0 de memória
# DEFINE    BANK1    BSF      STATUS, RP0    ;Seta bank 1 de memória
```

Listagem 1.0 Definição dos comandos no estado de paginação de memória.

O estado de variáveis trata da definição de todos os nomes e endereço de todas as variáveis do sistema. Seguindo a estrutura existente em nosso modelo listado na figura 2.0 neste estado e criado o espaço e nome para todas as variáveis que utilizaremos no programa. As variáveis são criadas como var1, var2 e assim sucessivamente sempre utilizando os comandos CBLOCK e ENDC para facilitar o endereçamento das variáveis. A listagem 2.0 mostra um exemplo de definição de variáveis.

```
;*************************************
;*              VARIÁVEIS                *
;*************************************
; DEFINIÇÃO DOS NOMES E ENDEREÇOS DE TODAS AS VARIÁVEIS UTILIZADAS  PELO SISTEMA
          CBLOCKDx0C ;ENDEREÇO INICIAL DA MEMÓRIA DE ;USUÁRIO
          W_TEMP;    REGISTRADORES TEMPORÁRIOS PARA USO  STATUS_TEMP;JUNTO ÀS INTERRUPÇÕES
          FLAG; REGISTRADOR PARA FLAG
          VAR1;
          VAR2;
          VARN;
          ENDC;  FIM DO BLOCO DE MEMÓRIA
```

Listagem 2.0 Definição das variáveis utilizadas no sistema.

O estado flags internos define todos os flags utilizados pelo sistema. Flags são bits que definimos dentro de um byte para serem utilizados como chaves on/off. Desta forma, em um único endereço de memória (registrador) poderemos guardar ate 8 flags que registrarão 8 flags diferentes. Por exemplo, um flag pode marcar se um byte já foi transmitido ou não, outro pode marcar se existe um dado recebido ou não. Nesta parte o interpretador cria dois flags. Um flag transmitido e um flag recebido. Pois os experimentos propostos em nossos estudos somente necessitam de dois flags nesta versão. Na listagem 3.0 mostramos uma pequena listagem de código gerado por este estado.

```
;*************************************
;*              FLAGS INTERNOS           *
;*************************************
; DEFINIÇÃO DE TODOS OS FLAGS UTILIZADOS PELO SISTEMA

# DEFINE     TRANSMITIDO          FLAG,0      ; FLAG PARA INFORMAR QUE O DADO FOI TRANSMITIDO
                                              ;1 -> TRANSMITIDO  E 0 -> NÃO TRANSMITIDO
# DEFINE     RECEBIDO             FLAG,1      ; FLAG PARA INFORMAR QUE O DADO FOI RECEBIDO
                                              ;1 -> RECEBIDO  E 0 -> NÃO RECEBIDO
```

Listagem 3.0 Definição dos flags internos.

No estado constante, são muito úteis para simplificar alterações de valores no sistema, por exemplo, imagine que seu sistema deva possuir vários delays de atraso durante a execução. O interpretador LBM cria todos os delays baseado em uma constante de tempo quando o programador usa o comando ESPERE na linguagem LBM e criado uma constante para delimitar o tempo, como mostra a listagem 4.0.

```
;*************************************
;*              CONSTANTES               *
;*************************************
; DEFINIÇÃO DE TODAS AS CONSTANTES UTILIZADAS PELO SISTEMA
TEMPO_DELAY  QUE   .00020  ;TEMPO DE DELAY DO SISTEMA EM mili SEG. PODE VARIAR DE 10 A 50 mili SEG.
```

Listagem 4.0 Definição das Constantes.

O estado entradas e saídas definem o nome das portas que serão de entrada e saída, somente os labels. Os periféricos conectados a porta podem ser observadas na tabela 2.0.

O estado vetor de reset possui um endereço para o qual o programa é desviado toda vez que um reset ocorre, seja ele pela energização do sistema ou pelo master clear externo (/MCLR). Neste estado e escrito um código para ir ao início do programa. A listagem 5.0 mostra o código escrito em assembler.

```
;*********************************
;*          VETOR DE RESET       *
;*********************************
;
        ORG     0x00    ;ENDEREÇO INICIAL DE PROCESSAMENTO
        GOTO    INICIO
```

Listagem 5.0 Definição do Vetor de Reset.

No estado inicio do programa e escrita as rotinas de desvio e as rotinas de chamada, que também podem ser consideradas de funções no assembler. As rotinas de desvio nada mais são do que "pulos" no programa por meio da instrução GOTO, esses pulos são identificados pelo comando "VAI" na linguagem LBM. Acontece que no assembler o numero de linha e substituído por um label. Já as rotinas de chamada são acessadas através do comando CALL. O comando CALL corresponde na linguagem LBM por REPETE. Esta instrução possibilita que o próximo ponto do programa (PC+1) seja guardado na pilha (stack) para que o sistema possa retornar a ele mais tarde por meio da instrução RETURN (ou similar) que é utilizado para encerrar a rotina. A listagem 6.0 mostra parte do código gerado pelo estado inicio do programa.

```
;*********************************
;*          INICIO DO PROGRAMA   *
;*********************************
;
INICIO
        BANK1                   ;ALTERA PARA O BANCO 1
        MOVLW  B'00000110'
        MOVWF  TRISA            ;DEFINE RA1 E 2 COMO ENTRADA E DEMAIS
                                ;COMO SAÍDAS
        MOVLW  B'00000000'
        MOVWF  TRISB            ;DEFINE TODO O PORTB COMO SAÍDA
        MOVLW  B'10000000'
        MOVWF  OPTION_REG       ;PRESCALER 1:2 NO TMR0
                                ;PULL-UPS DESABILITADOS
                                ;AS DEMAIS CONFG. SÃO IRRELEVANTES
        MOVLW  B'00000000'
        MOVWF  INTCON           ;TODAS AS INTERRUPÇÕES DESLIGADAS
        BANK0                   ;RETORNA PARA O BANCO 0
```

Listagem 6 – Estado Inicial do programa

O estado inicialização das variáveis coloca as variáveis em seu estado inicial, como mostra a listagem 7.

```
;*********************************
;*      INICIALIZAÇÃO DAS VARIÁVEIS  *
;*********************************
;
        CLRF    PORTA           ;LIMPA O PORTA
        MOVLW  DISPLAY
        MOVWF  PORTB            ;ACENDE O VALOR CERTO NO DISPLAY
        MOVLW  MIN
        MOVWF  CONTADOR         ;INICIA CONTADOR COM VALOR MIN.
```

Listagem 7 – Estado Inicialização das Variáveis.

O estado Rotina Principal é criada as rotinas que testam diferentes bits, podem ser portas ou flags, são: BTFSC e BTFSS em assembler. Esses comandos correspondem ao comando SE PORTA da linguagem LBM.

BTFSC = Testa (T) no bit (B) do registrador (F) e pula (S) a próxima linha se a resposta for 0 (C)
BTFSS= Testa (T) o bit (B) do registrador (F) e pula (S) a próxima linha se a resposta for 1 (S).

A listagem 8 mostra uma rotina criada pela linguagem LBM.

```
;**********************************
;*            ROTINA PRINCIPAL                *
;**********************************
MAIN

        BTFSC   BOTAO           ;O BOTÃO ESTÁ PRESSIONADO?
        GOTO    BOTAO_LIB       ;NÃO, ENTÃO TRATA BOTÃO LIBERADO
        GOTO    BOTAO_PRES      ;SIM, ENTÃO TRATA BOTÃO PRESSIONADO

BOTAO_LIB
        BCF     LED             ;APAGA O LED
        GOTO    MAIN            ;RETORNA AO LOOP PRINCIPAL

BOTAO_PRES
        BSF     LED             ;ACENDE O LED
        GOTO    MAIN            ;RETORNA AO LOOP PRINCIPAL
```

Listagem 8 – Código gerado pelo estado Rotina Principal.

O estado Fim do Programa e responsável pelo fechamento do programa em assembler e a compilação do código e a transferência para o microcontrolador PIC. Na listagem 9 e mostrado o fechamento do programa.

```
;**********************************
;*            FIM DO PROGRAMA                 *
;**********************************

        END                     ;OBRIGATÓRIO
```

Listagem 9 – Fechamento do Programa.

## 7. PROGRAMA DE EXEMPLO.

Nesta seção iremos mostrar um código simples que podem ser implementados com a linguagem LBM. Para criar um pisca-pisca com um led na linguagem LBM a implementação do código e mostrado na listagem 10

```
01- PORTA 7 ALTO          ' LED direito ligado (será um pisca-pisca)
02- ESPERE  1000 mili s   ' tempo de aceso (1 seg)
03- PORTA 7 BAIXO         ' LED direito apagado
04- ESPERE  1000 mili s   ' tempo de apagado (1 seg)
05- VAI LINHA 01          ' fica em loop piscando sem parar
06- FIM                   ' fim do programa PISCA_LED.LBM
```

Listagem 10 – Programa Pisca – Pisca na linguagem LBM

O mesmo programa em linguagem assembly utiliza mais de 20 linhas de código. Pois e necessário a definição de seu arquivo de definição, paginação de memória, variáveis, portas de entradas,saídas e vetor de reset. Na linguagem LBM isso fica  por conta do interpretador da

linguagem. O código abaixo foi retirado de [SOUZA, 2003] para realizar um comparativo. A listagem 11 mostra o código em assembler.

```
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*                     DESCRIÇÃO DO ARQUIVO                              *
;*-------------------------------------------------------------------*
;*    SISTEMA  MUITO  SIMPLES  PARA  REPRESENTAR  O  ESTADO  DE  UM  BOTÃO  ATRAVÉS  DE  UM  LED.
*
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*                     ARQUIVOS DE DEFINIÇÕES                            *
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
#INCLUDE <P16F628A.INC>             ;ARQUIVO PADRÃO MICROCHIP PARA 16F628A
__CONFIG  _BODEN_ON & _CP_OFF & _PWRTE_ON & _WDT_OFF & _LVP_OFF & _MCLRE_ON & _XT_OSC
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*                    PAGINAÇÃO DE MEMÓRIA                               *
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;DEFINIÇÃO DE COMANDOS DE USUÁRIO PARA ALTERAÇÃO DA PÁGINA DE MEMÓRIA

#DEFINE BANK0  BCF STATUS,RP0 ;SETA BANK 0 DE MEMÓRIA
#DEFINE BANK1  BSF STATUS,RP0 ;SETA BANK 1 DE MEMÓRIA
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*                         VARIÁVEIS                                     *
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; DEFINIÇÃO DOS NOMES E ENDEREÇOS DE TODAS AS VARIÁVEIS UTILIZADAS PELO SISTEMA
       CBLOCK 0x20   ;ENDEREÇO INICIAL DA MEMÓRIA DE USUÁRIO
       ENDC                 ;FIM DO BLOCO DE MEMÓRIA
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*                        FLAGS INTERNOS                                 *
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; DEFINIÇÃO DE TODOS OS FLAGS UTILIZADOS PELO SISTEMA
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*                         CONSTANTES                                    *
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; DEFINIÇÃO DE TODAS AS CONSTANTES UTILIZADAS PELO SISTEMA
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*                          ENTRADAS                                     *
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; DEFINIÇÃO DE TODOS OS PINOS QUE SERÃO UTILIZADOS COMO ENTRADA RECOMENDAMOS TAMBÉM COMENTAR O
SIGNIFICADO DE SEUS ESTADOS (0 E 1)

#DEFINE BOTAO  PORTA,2 ;PORTA DO BOTÃO       ; 0 -> PRESSIONADO  1 -> LIBERADO
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*                          SAÍDAS                                       *
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; DEFINIÇÃO DE TODOS OS PINOS QUE SERÃO UTILIZADOS COMO SAÍDA RECOMENDAMOS TAMBÉM COMENTAR O
; SIGNIFICADO DE SEUS ESTADOS (0 E 1)

#DEFINE LED    PORTB,0 ;PORTA DO LED  ; 0 -> APAGADO
                                      ; 1 -> ACESO
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*                       VETOR DE RESET                                  *
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
       ORG    0x00           ;ENDEREÇO INICIAL DE PROCESSAMENTO
       GOTO   INICIO
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*                    INÍCIO DA INTERRUPÇÃO                              *
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; AS INTERRUPÇÕES NÃO SERÃO UTILIZADAS, POR ISSO PODEMOS SUBSTITUIR TODO O SISTEMA EXISTENTE
NO ARQUIVO MODELO PELO APRESENTADO ABAIXO ESTE SISTEMA NÃO É OBRIGATÓRIO, MAS PODE EVITAR
PROBLEMAS FUTUROS


       ORG    0x04           ;ENDEREÇO INICIAL DA INTERRUPÇÃO
       RETFIE                ;RETORNA DA INTERRUPÇÃO
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*                     INICIO DO PROGRAMA                                *
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
INICIO
       CLRF   PORTA          ;LIMPA O PORTA
       CLRF   PORTB          ;LIMPA O PORTB
       BANK1                        ;ALTERA PARA O BANCO 1
       MOVLW  B'00000100'
       MOVWF  TRISA          ;DEFINE RA2 COMO ENTRADA E DEMAIS
                                    ;COMO SAÍDAS
       MOVLW  B'00000000'
```

```
        MOVWF   TRISB           ;DEFINE TODO O PORTB COMO SAÍDA
        MOVLW   B'10000000'
        MOVWF   OPTION_REG      ;PRESCALER 1:2 NO TMR0        ;PULL-UPS DESABILITADOS
                                        ;AS DEMAIS CONFG. SÃO IRRELEVANTES
        MOVLW   B'00000000'
        MOVWF   INTCON          ;TODAS AS INTERRUPÇÕES DESLIGADAS
        BANK0                           ;RETORNA PARA O BANCO 0
        MOVLW   B'00000111'
        MOVWF   CMCON           ;DEFINE O MODO DE OPERAÇÃO DO COMPARADOR ANALÓGICO
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*                      INICIALIZAÇÃO DAS VARIÁVEIS                      *
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*                      ROTINA PRINCIPAL                                 *
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
MAIN
        BTFSC   BOTAO           ;O BOTÃO ESTÁ PRESSIONADO?
        GOTO    BOTAO_LIB       ;NÃO, ENTÃO TRATA BOTÃO LIBERADO
        GOTO    BOTAO_PRES      ;SIM, ENTÃO TRATA BOTÃO PRESSIONADO
BOTAO_LIB
        BCF     LED                             ;APAGA O LED
        GOTO    MAIN            ;RETORNA AO LOOP PRINCIPAL
BOTAO_PRES
        BSF     LED                             ;ACENDE O LED
        GOTO    MAIN            ;RETORNA AO LOOP PRINCIPAL
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*                        FIM DO PROGRAMA                                *
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
        END                     ;OBRIGATÓRIO
```

Listagem 11- Código escrito na linguagem Assembler

## 8. CONCLUSÕES E TRABALHOS FUTUROS.

O estudo apresentado neste artigo demonstra que e possível conceber uma linguagem de programação estruturada para microcontroladores visando facilitar a programação criando um interpretador para linguagem assembly, com a implementação de automatas finitos determinísticos. O modelo proposto elimina a necessidade do programador conhecer as interrupções, registradores e outras particularidades do microcontrolador, bastando somente o programador conhecer a linguagem LBM e mudar o interpretador para o microcontrolador escolhido.

Ainda é necessário realizar comparações em diferentes linguagens com o modelo proposto a fim de se obter dados de desempenho e aprendizado com o protótipo atual. Um dos trabalhos futuros propostos e medir a eficiência da linguagem LBM com crianças em comparação a linguagem assembler e linguagem C. A metodologia a ser utilizada e traçar uma curva de aprendizagem comparando a linguagem LBM com as linguagens C e Assembler para microcontroladores PIC, crianças que cursam o ensino médio poderiam ser exposta a essa linguagem e realizar um comparativo com aprendizado com alunos universitário. Essa linguagem para alunos universitário e somente introdutório para que futuramente eles sejam expostos a linguagens C e assembler para entender o funcionamento dos registradores e particularidades dos microcontroladores.

O editor da linguagem ajuda o programador, a saber, quais os parâmetros que devem ser empregados para determinados comandos, evitando erro de sintaxe.

O estudo realizado neste artigo não só demonstrou a viabilidade da proposta como também abriram novas perspectivas de pesquisa e desenvolvimento.

**AGRADECIMENTOS**

Agradecemos a FINEP – Financiadora de Estudos e Projetos que nos concedeu o investimento para o projeto WEBLAB - Um Ambiente Computacional de Aprendizagem Interligada com Experimentos Reais de Física através de Sistemas de Aquisição de Dados para realização das pesquisas apresentadas neste artigo. A linguagem de programação LBM faz parte do projeto WEBLAB.

Agradecemos também a Universidade Braz Cubas e a Faculdade Bandeirantes que incentivou e forneceu subsídios para criação do trabalho e participação no congresso.

## 8. REFERÊNCIAS

[GOTTFRIED, 1993] Byron S. Gottfried, (1993) "Programando em C", São Paulo, Makron Books, 1993.

[HOPCROFT, 1939] HOPCROFT, John E. , Introdução a Teoria dos Automatos e Linguagens e Computação, 2º Ed, original de Vanderberg D. de Souza , Rio de janeiro, Campus, 2002

[SOUZA,2003] SOUZA, David José de, Desbravando o PIC ampliado e atualizado para PIC 16F628A, 9º Edição, São Paulo, Érica, 2003.

[SULLIVAN, 2006] Sullivan, Greg, Lista de discussão do professor do MIT Massachusetts Institute of Technology, http://people.csail.mit.edu/gregs/ll1-discuss-archive-html/msg04323.html, Acessado em 12/01/2006.

[MARTINS,2005] MARTINS, Nardenio Almeida , Sistemas Microcontrolados, São Paulo, Novatec, 2005.

[MENEZES,1998] Paulo Blauth, Linguagens Formais e Autômatos, Instituto de Informática da UFRGS, 2º Ed, Sagra Luzzatto, Porto Alegre, RS.

[MICROCHIP, 2006] http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=64. Acessado em 10/01/2006.

[MORKARZEL,2004] MORKAZEL, Marcos Perez, Internet Embedded: TCP/IP para Microcontroladores, 1ºed, São Paulo, Érica, 2004.

[NICOLOSI, 2000] NICOLOSI, Denys Emilio Campion, Microcontrolador 8051 Detalhado, São Paulo, Érica, 2000.

[PEREIRA, 2002] PEREIRA, Fabio, Microcontroladores PIC Técnicas Avançadas, São Paulo, Érica, 2002.

[PETROUTSOS, 2000] PETROUTSOS, Evangelos, Dominando o Visual Basic 6.0, São Paulo, Makron Books, 2000.

# LBM-UMA PROPOSTA DE LINGUAGEM BÁSICA ESTRUTURADA PARA PROGRAMAÇÃO DE MICROCONTROLADORES NOS CURSOS DE ENGENHARIA

***Abstract.*** *This meta-paper describes the Language Programs structure with few commands for microcontroller PIC16F628A with the language of the high level in order to introduce children or people with any programming knowledge for microcontroller. Through a kit of experiments the student it can accomplish the assembly of several simple circuits and realize the program for the electronics*

*components, not work with registers with do in language assembler, because the propose language encapsule this functionales facilitating the student's learning and reducing the difficult of the language. The language has to an editor of commands to facilitate in the implementation of the commands, but a text editor can be used to accomplish the programming.*

***Key-words:*** *Programming language, Microcontroler, Teach the Engineering*