UMA IMPLEMENTAÇÃO DE META ASSEMBLER EM PASCAL

Flávio Soibelmann Glock
Porto Alegre - *Brasil*

- RESUMO

O Meta Assembler é um software que possibilita gerar montadores assembly para diferentes processadores.

A partir da definição de um formato generalizado obtido através do estudo das linguagens dos principais processadores o autor desenvolveu a Linguagem de Definição de Assembler (LDA). Com ela o usuário define o código de máquina a ser gerado e a sintaxe da linguagem que será utilizada.

Aplicando o formalismo LDA é feita a descrição da linguagem assembly do processador Z80.

- ABSTRACT

The Meta Assembler is an utility software which generates assemblers for any microprocessor. The user writes the specifications which define the machine-code to be generated and the language sintax to be used.

The author designed a generalized format to describe the most popular microprocessor languages. It is the Assembly Language Definition (LDA) format.

The LDA language implementation of Zilog's Z80 microprocessor assembly language is shown.

1. INTRODUÇÃO

O estado atual da informática exige que os computadores sejam capazes de "se comunicarem" com o programador em linguagens dirigidas a facilitar o trabalho humano. No entanto, os computadores desenvolvidos para funções específicas não são acessíveis ao programador, uma vez que normalmente não estão conectados a teclados e vídeos. Além disso, quando o programador fornece instruções, estas só podem ser executadas após a tradução para a linguagem de máquina dos processadores usados.

A construção de novos computadores impõe então recursos especiais para desenvolver o software necessário. O META ASSEMBLER é um desses recursos.

2. OBJETIVO

Para desenvolver equipamentos baseados em novos microprocessadores são necessárias ferramentas de software adequadas, as quais nem sem-

pre são disponíveis. O desenvolvimento dessas ferramentas leva à sobrecarga de trabalho e aumenta os prazos e custos dos projetos.

A solução encontrada para minimizar essas dificuldades foi a criação de um software genérico. O objetivo desse software é gerar montadores dedicados para cada microprocessador, a partir das especificações fornecidas.

O estudo das linguagens assembly dos microprocessadores existentes no mercado permitiu a identificação dos seus elementos comuns, fornecendo o princípio geral deste meta assembler.

Entende-se por meta assembler o software capaz de, com as especificações fornecidas, gerar montadores assembly. Estes, identificando uma linguagem, geram código de máquina.

Este software foi desenvolvido com o objetivo de estimular a utilização de microprocessadores novos em projetos de instrumentação no Centro de Engenharia Biomédica da Pontifícia Universidade Católica do Rio Grande do Sul (CEB-PUCRS), bem como fornecer suporte a outros setores de desenvolvimento dentro e fora da Universidade.

3. ASPECTOS GERAIS DO SOFTWARE

3.1. VISÃO GERAL DE UM ASSEMBLER

A expressão "assembler" traduzida literalmente significa "montador". É assim chamada porque se refere a um programa que traduz uma linguagem simbólica simples com a qual representamos operações (ex: LD A,B quando queremos transferir uma informação do registrador B para o registrador A) em linguagem de máquina (ex: 01111000 que representa a operação de transferência efetiva do registrador B para A). O assembler deve compor ou "montar" um código de bits, a partir de tabelas específicas, inerentes a cada processador.

Cada instrução em linguagem simbólica corresponde a um número determinado de bytes em linguagem de máquina. O assembler mantém um contador para registrar a posição onde ficarão esses bytes na memória do processador. Esta posição é chamada de endereço de memória. O assembler permite que o usuário guarde esses endereços colocando nomes ou "etiquetas" específicas para cada um. Esta forma de registro constitui os LABELS.

Além de permitir a definição de labels, o assembler permite a definição do próprio contador de memória, indicando assim onde colocar o programa. Podem ser definidos espaços que serão ocupados por variáveis e também valores iniciais para locais de memória.

O assembler deve ser capaz de interpretar expressões aritméticas e lógicas. Muitas vezes o usuário precisa definir certos operandos a partir de dados que serão estabelecidos durante a montagem do programa. Utiliza-se nesse caso, além das quatro operações, as operações com bits

(shift, separação de bytes) e operações lógicas (e, ou, não, ou-exclusivo). Além disso, o assembler deve aceitar dados em vários sistemas numéricos tais como binário, octal, hexadecimal e caracteres ASCII.

3.2. IMPLEMENTAÇÃO DO META ASSEMBLER NO COMPUTADOR CEB-286-PUCRS

O meta assembler foi escrito em linguagem Pascal. O código objeto foi gerado através de um compilador TURBO Pascal implementado em sistema operacional MS/DOS 3.0.

O Software é constituído de 4 partes:

- módulo de interpretação de expressões numéricas;
- módulo de formatação de saída e de mensagens de erro;
- módulo de interpretação da Linguagem de Definição de Assembler;
- módulo principal. Este módulo implementa um assembler em duas etapas (2-pass assembler).

O código de máquina é gerado em formato hexadecimal, compatível com o programador de EPROM que está sendo desenvolvido no CEB-PUCRS.

3.3. FUNCIONAMENTO DO ASSEMBLER EM DUAS ETAPAS

Para "aprender" a linguagem a ser utilizada o software lê as especificações que estão armazenadas em disco.

O programa a ser montado é lido uma primeira vez. Durante esta primeira passagem o software cria uma lista de labels usados (tabela de labels).

O software lê o programa pela segunda vez, executando as seguintes tarefas:

- indica eventuais erros;
- grava no disco o programa em linguagem de máquina, em formato hexadecimal;
- lista o programa na tela ou impressora.

3.4. UTILIZAÇÃO DO META ASSEMBLER

A especificação da linguagem para o processador é feita através de um editor de texto, de acordo com as regras da Linguagem de Definição de Assembler (LDA).

O programa a ser montado é escrito utilizando-se o editor de de texto ou então é transmitido de outro computador.

Inicia-se a operação do meta assembler indicando as especificações a serem utilizadas e o nome do programa que se deseja montar.

O operador deve optar por saída na impressora/disco ou tela/dis-

co para obter a listagem.

O código de máquina gerado será transmitido para o equipamento objeto da montagem através de programador de EPROM ou via interface serial ou paralela.

3.5. VANTAGENS E LIMITAÇÕES DO SOFTWARE

O META ASSEMBLER permite a montagem de programas para processadores de 8, 16 e 32 bits, sendo que o limite de precisão para os operandos numéricos é de 32 bits mais sinal. O tempo de implementação de um montador para um novo processador é estimado em 40 horas.

O programa é compatível com diversos computadores, uma vez que é escrito em linguagem Pascal.

O tamanho máximo do programa a ser montado depende da capacidade da memória principal, que guarda a tabela de labels mais a definição do processador.

A velocidade de montagem varia dependendo do processador definido, mas situa-se em torno de 10 linhas por segundo utilizando-se o computador CEB-286-PUCRS.

4. A LINGUAGEM DE DEFINIÇÃO DE ASSEMBLER

A linguagem de definição de assembler serve para a codificação das especificações em formato capaz de ser interpretado pelo meta assembler. Para escrever em LDA o programador utiliza um editor de texto, criando um arquivo do tipo PROC.DEF no disco, sendo PROC o nome do processador a ser definido.

Uma linguagem fica definida pela criação de 5 conjuntos de informações:

- tabelas de constantes;
- tabelas de opcodes;
- limites para operandos numéricos;
- nome dos pseudo-opcodes utilizados;
- tabelas de sintaxe para as operações.

4.1. TABELAS DE CONSTANTES

Os operandos do mesmo tipo, como condições e sets de registradores, são utilizados várias vezes com o mesmo opcode, modificando o código de máquina gerado de acordo com tabelas definidas. Estes operandos são colocados para a LDA como tabelas de constantes.

4.2. TABELAS DE OPCODES

As operações que tem a mesma sintaxe, como as operações aritméticas e lógicas, são agrupadas através de tabelas de opcodes. Nestas ta-

belas consta o nome de cada operação e o valor numérico que o mesmo assume na montagem.

4.3. LIMITES PARA OPERANDOS NUMÉRICOS

A verificação da sintaxe em determinadas operações requer o teste de limites das expressões utilizadas a fim de evitar o overflow na montagem. Por exemplo, uma instrução que utiliza um byte de dados não deve ser montada se o valor dado for maior que 255. O usuário define os valores mínimos e máximos para as expressões.

Também são definidos o tamanho da palavra e o tamanho do endereçamento do processador. A definição do tamanho da palavra é importante pois alguns processadores possuem contador de endereços que incrementa 2 bytes de cada vez, enquanto que outros incrementam apenas 1 byte. O tamanho do endereçamento é utilizado pelo módulo de formatação de saída, colocando 2 ou 4 bytes de endereço antes de cada linha de código.

4.4. NOME DOS PSEUDO OPCODES UTILIZADOS

O meta assembler possui diversos opcodes definidos. No entanto, cada fabricante define nomes para as operações executadas pelo assembler de seu processador, tornando necessária a flexibilidade do sistema neste ponto. Por isso a LDA permite a redefinição dos nomes dos opcodes.

4.5. TABELAS DE SINTAXE PARA AS OPERAÇÕES

Através da LDA o usuário define as operações que o assembler deve ser capaz de montar. A definição é feita a partir dos nomes das tabelas já definidas e utilizando símbolos, tais como vírgulas, parênteses e outros, que descreverão a sintaxe da operação.

5. EXEMPLO DE APLICAÇÃO: DEFINIÇÃO DO PROCESSADOR Z80

A equipe do Centro de Engenharia Biomédica definiu o assembler do processador Z80 em LDA, para possibilitar o prosseguimento de um trabalho já iniciado em um computador CP/M onde era utilizado o montador MACRO-80. A vantagem da implementação em LDA é o aproveitamento do hardware existente no computador CEB-286, que possui maior capacidade de armazenamento em disco e onde está sendo desenvolvido um programador de EPROM.

O programa a seguir implementa o assembler Z80.
Uma descrição da linguagem LDA encontra-se no apêndice.

```
; --------------------------------
; definição do processador Z80
; --------------------------------

; definição do pseudo opcodes.

pseudo ORG =        ORG
```

```
pseudo EQU =        EQU
pseudo END =        END
pseudo PHASE =      .PHASE      ; para compatibilidade com Macro-80
pseudo DEPHASE =    .DEPHASE
pseudo DIW =        DEFW
pseudo DEFS1 =      DEFS
pseudo DB1 =        DEFB
pseudo SET =        SET
pseudo TITLE =      TITLE

; definição de tamanho de endereçamento.

word = 1      ;(contador de endereços incrementa para cada 1 bytes)
address = 2   ;(2 bytes de endereçamento)

; definição de intervalos numéricos

range d:    - 127 .. 127
range adrs: 0 .. 0FFFFH
range n:    0 .. 0FFH
range nn:   0 .. 0FFFFH
range p:    0 .. 38H

; definição de tabelas de constantes

qq:       ;a tabela 'qq' e utilizada na montagem
 BC=00H   ; das operações 'PUSH' e 'POP'
 DE=10H
 HL=20H
 AF=30H

rega:
 B=0
 C=8
 D=10H
 E=18H
 H=20H
 L=28H
 A=38H

regb:
 B=0
 C=1
 D=2
 E=3
 H=4
 L=5
 A=7

pp:
 BC=00H
 DE=10H
```

```
 IX=20H
 SP=30H


rr:
 BC=00H
 DE=10H
 IY=20H
 SP=30H

rbd:
 BC=00H
 DE=10H

dd:
 BC=00H
 DE=10H
 HL=20H
 SP=30H

cc:
 NZ=00H
 Z=08H
 NC=10H
 C=18H
 PO=20H
 PE=28H
 P=30H
 M=38H

ix:
 IX=0DDH
 IY=0FDH

cjr:
 C = 38H
 NC = 30H
 Z = 28H
 NZ = 20H

bit:
 0=0H
 1=8H
 2=10H
 3=18H
 4=20H
 5=28H
 6=30H
 7=38H

; definição de tabelas de opcodes
```

```
opcode calc:
 INC=04H
 DEC=05H

opcode arita:
 AND=20H
 CP=38H
 OR=30H
 XOR=28H

opcode aritb:
 ADD=00H
 ADC=08H
 SUB=10H
 SBC=18H

opcode seta:
 BIT=40H
 RES=80H
 SET=0C0H

opcode blt:
 NEG =  $44
 LDI =  $A0
 LDIR = $B0
 LDD =  $AB
 LDDR = $BB
 CPI =  $A1
 CPIR = $B1
 CPD =  $A9
 CPDR = $B9
 RLD =  $6F
 RRD =  $67
 RETI = $4D
 RETN = $45
 INI =  $A2
 INIR = $B2
 IND =  $AA
 INDR = $BA
 OUTI = $A3
 OTIR = $B3
 OUTD = $AB
 OTDR = $CC

opcode rot:
 RLC=0H
 RRC=8H
 RL=10H
 RR=18H
 SLA=20H
 SRA=28H
 SRL=38H
```

```
; definição da tabela de sintaxe e de montagem:
;   a primeira linha define a sintaxe da operação;
;   as linhas seguintes dão instruções para montagem.

JP (HL)    ;representa um instrução com sintaxe rígida
 0E9H      ;monta 1 byte = E9 hexa.

             ;caso a sintaxe 'JP' acima não sirva, então continua
             ;  procurando outra forma da instrução:

JP (ix)    ;representa uma instrução com opções: usa a tabela 'ix'
  ix       ;monta 1 byte = DD ou FD hexa,
 0E9H      ;  e o segundo byte = E9 hexa.

JP cc,nn
 0C2H + cc
 LOW nn
 HIGH nn

JP nn
 0C3H
 LOW nn
 HIGH nn

arita (HL)    ;usa a tabela de opcodes 'arita'
 arita+86H    ;soma o valor encontrado com 86 hexa.
arita regb       ;a ausência de tabulador marca início de definição
 arita+regb+80H ;a tabela de montagem tem sempre tabulador antes.
arita (ix d)
 ix
 arita+86H
 d
arita n
 0C6H+arita
 n

aritb A,(HL)
 aritb+86H
aritb A,regb
 aritb+regb+B0H
aritb A,(ix d)
 ix
 aritb+86H
 d
aritb A,n
 0C6H+aritb
 n
IN rega,(C)
 0EDH
 40H+rega
IN A,(n)
```

```
 0DBH
 n
OUT (C),rega
 0EDH
 41H+rega
OUT (n),A
 0D3H
 n
; ********* LD simples ************
LD A,I
 0EDH
 57H
LD A,R
 0EDH
 5FH
LD I,A
 0EDH
 47H
LD R,A
 0EDH
 4FH
LD SP,HL
 OF9H
LD SP,ix
 ix
 OF9H
LD rega (HL)
 46H + rega
LD rega,regb
 40H+rega+regb
LD (HL),regb
 70H+regb
LD A,(rbd)
 rbd+0AH
LD (rbd),A
 rbd + 2
; ********* LD com o primeiro operando simples **************
LD HL,(nn)
 2AH
 LOW nn
 HIGH nn
LD dd,(nn)
 0EDH
 5BH+dd
 LOW nn
 HIGH nn
LD dd,nn
 dd + 1
 LOW nn
 HIGH nn
LD rega,(ix d)
 ix
```

```
  46H+rega
  d
 LD A,(nn)
  3AH
  LOW nn
  HIGH nn
 LD rega,n
  rega + 6
  n
 LD (HL),n
  36H
  n
 LD ix,(nn)
  ix
  2AH
  LOW nn
  HIGH nn
 LD ix,nn
  ix
  21H
  LOW nn
  HIGH nn
 ; *********** algum cálculo no primeiro operando ***********
 LD (ix d),regb
  ix
  70H+regb
  d
 LD (ix d),n
  ix
  36H
  d
  n
 ; *********** outros LD *************
 LD (nn),A
  32H
  LOW nn
  HIGH nn
 LD (nn),HL
  22H
  LOW nn
  HIGH nn
 LD (nn),dd
  0EDH
  43H+dd
  LOW nn
  HIGH nn
 LD (nn),ix
  ix
  22H
  LOW nn
  HIGH nn
 ; **************
```

```
calc (ix d)
 ix
 calc+30H
 d
calc rega
 rega+calc
calc (HL)
 48+calc
PUSH qq
 0C5H+qq
PUSH ix
 ix
 0E5H
POP qq
 0C1H+qq
POP ix
 ix
 0E1H
EX DE,HL
 0EBH
EX AF,AF'
 08H
EXX
 0D9H
EX (SP),HL
 0E3H
EX (SP),ix
 ix
 0E3H
JR cjr,adrs   ;'$' representa o contador de programa.
 cjr
 adrs-2-$
JR adrs
 18H
 adrs-2-$
DJNZ adrs
 10H
 adrs-$-2
CALL cc,nn
 cc+0C4H
 LOW nn
 HIGH nn
CALL nn
 0CDH
 LOW nn
 HIGH nn
RET
 0C9H
RET cc
 cc+0C0H
CPL
 2FH
```

```
CCF
 3FH
NOP
 00H
RLCA
 7H
RLA
 17H
RRCA
 0FH
RRA
 1FH
SCF
 37H
RST p
 0C7H+p
ADD HL,dd
 9H+dd
ADC HL,dd
 0EDH
 4AH+dd
SBC HL,dd
 0EDH
 42H+dd
DAA
 27H
HALT
 76H
DI
 0F3H
EI
 0FBH
IM 0
 0EDH
 46H
IM 1
 0EDH
 56H
IM 2
 0EDH
 5EH
ADD IX,pp
 ODDH
 49H+pp
ADD IY,rr
 0FDH
 9H+rr
INC dd
 3H+dd
INC ix
 ix
 23H
```

```
DEC dd
 0BH+dd
DEC ix
 ix
 2BH
seta bit,(HL)
 0CBH
 seta+bit+6
seta bit,regb
 0CBH
 seta+bit+regb
seta bit,(ix d)
 ix
 0CBH
 d
 seta+bit+6H
blt
 0EDH
 blt
rot (HL)
 0CBH
 rot+6
rot regb
 0CBH
 rot+regb
rot (ix d)
 ix
 0CBH
 d
 rot+6
```

6. CONCLUSÕES

A utilização do meta assembler pelo grupo de desenvolvimento aumentou as possibilidades de utilização dos recursos de hardware disponíveis.

Estão sendo implementados em LDA montadores para os microprocessadores MC68000, 80286 e o microcomputador Z8.

Está em estudo a viabilidade de definição de uma linguagem assembler padronizada para vários microprocessadores, e a implementação de uma extensão da LDA que permita escrever os programas em uma linguagem estruturada do tipo PL/M e PL/Z.

- APÊNDICE

SINTAXE DA LINGUAGEM LDA: Linguagem de Descrição de Assembler

O formalismo BNF (Bankus Naur Form) é utilizado a seguir para descrever a sintaxe de presente linguagem.

```
< definição de assembler > ::= < declaração >
                                  <definição de assembler>

  declaração    ::= < declaração de nome de pseudo-opcode > |
                    < declaração de tabela de opcode > |
                    < declaração de tabela de constante > |
                    < declaração de limite numérico > |
                    < declaração de tamanho de palavra > |
                    < declaração de tamanho de endereçamento > |
                    < declaração de sintaxe > |
                    < comentário >

< declaração de nome de pseudo-opcode > ::=
                                pseudo < pseudo > = < labelpseudo >

< pseudo > ::= EQU | SET | ORG | PHASE | DEPHASE |
               DS1 | DS2 | DS4 | DEFB1 | DEFB2 | DEFB4 | DIW

< labelpseudo > ::= <label >

< declaração de tabela de opcode > ::= opcode < labelopcode > :
                                        < lista de opcodes >

< labelopcode > ::= <label minúsculo >

< lista de opcodes > ::=
            < tabulador > < nomeopcode > = < expressão >
               < lista de opcodes >

< nomeopcode > ::= < label >

< declaração de tabel de constante > ::=
              < label tabela de constante > :
                 < lista de constantes >

< label tabela de constante > ::= < label minúsculo >

< lista de constantes > ::=
           < tabulador > < nome constante > = <expressão >
               <lista de constantes >

< nome constante > ::= < label >
```

```
< declaração de limite numérico >  ::= range   < labelrange >  =
                 < expressão >< tabulador > .. < tabulador>< expressão >

< labelrange >    ::=  < label minúsculo >

< declaração de tamanho de palavra >  ::= word = < expressão >

< declaração de sintaxe> ::=< nome da operação> <tabulador><operando >

< nome da operação >  ::=  < labelopcode>   |  < label >

< operando>  ::=  (  < label tabela de constante >  ,
                     < labelrange > ,
                     < símbolo maiúsculo> ,
                     < símbolo decimal > ,
                     < outros símbolos > ) [ < operando > ]

< expressão de montagem >  ::=  <tabulador>< expressão >
                                  < expressão de montagem >

< comentário > ::=  ( ; | * )[<comentário>]

< expressão > ::=  <operação unária >  < expressão > |
                   <expressão > < operação > <expressão > |
                   <constante decimal > |
                   <constante hexadecimal > |
                   <constante octal > |
                   <constante binária > |
                   <constante alfanumérica > |
                   <label > :
                   <pc >

<pc >  ::= $

<constante alfanumérica >  ::= ( ' , " )  <string > ( ' , " )

<string >  ::=  <símbolo maiúsculo > |
                <símbolo minúsculo > |
                <símbolo decimal  > |
                <outros símbolos> |
                ''' | """ | ;
                  [ <string> ]

<constante decimal>   ::= < símbolo decimal>

<constante hexadecimal >  ::= $ < símbolo hexa > |
                                < símbolo hexa > H

<constante octal >   ::= < símbolo octal >  (O, Q)

<constante binária >   ::= <símbolo binário >  B
```

<label > ::= (<símbolo maiúsculo > , <caracteres de separação >)
[(<label > , $)]

<label maiúsculo > ::=
(<símbolo minúsculo > , < caracteres de separação >)
[(<label maiúsculo > , $)]

<operação unária > ::= - | NOT | LOW | HIGH | LOWW | HIGHW

<operação > ::= * | / | + | - | MOD | SHR | SHL | AND | OR | XOR

<símbolo maiúsculo > ::=
A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z

<símbolo minúsculo > ::=
a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z

<símbolo hexa> ::=
(A|B|C|D|E|F | <símbolo decimal>)| [<símbolo hexa>]

<símbolo octal> ::= (0|1|2|3|4|5|6|70) [<símbolo octal >]

<símbolo decimal > ::= (0|1|2|3|4|5|6|8|9) [< símbolo decimal>]

<símbolo binário > ::= 0|1 | <símbolo binário >

<caracteres de separação > ::= . | _

<tabulador ::= espaço | tabulador |[<tabulador>]

<outros símbolos > ::=
* | (|) | [|] | | # | . | _ | ~ | | & | ! | : |
+ | $ | % | ' | | ? | ^ | | <tabulador>

- **BIBLIOGRAFIA**

1 _____ **8080/8085 Assembly Language Programming,** Santa Clara CA, Intel, 1979.

2 _____ **A Programmer's Guide to the Z8 Microcomputer** Application Note, Cupertino CA, Zilog, October 1980

3 _____ **Microcomputer Catalog** Databook, Mountain View CA, NEC Eletronics, 1984.

4 _____ **Series 32000** Databook, Santa Clara CA, National Semiconductor Corporation, 1984.

5 _____ **Utility Software Manual** MACRO-80, Microsoft, 1978.

6 _____ **Z8 Microcomputer Technical Manual,** Campbell CA, Zilog, April 1983.

7 _____ **Z80 CPU Programmer's Reference Guide,** Campbell Ca, Zilog, October 1982.

8 _____ **Z80 CPU Z80A CPU Technical Manual,** Cupertino CA, Zilog, September 1978.

9 AHO, Alfred V.; ULLMAN, Jeffrey D. **Principles of Compiler Design,** Reading MA, Addison-Wesley, April 1979.

10 BARRON, D. W. **Assemblers and Loaders,** London, McDonald, 1972.

11 KANE, Gerry; HAWKINS, Doug & LEVENTHAL, Lance. **68000 Assembly Language Programming,** Berkeley CA, OSBORNE/McGraw-Hill, 1981.

12 CALINGAERT, Peter **Assemblers, Compilers, and Program Translation** Potomac, Computer Science Press, 1979.