Software Básico – Turma B 2011/1
Trabalho 1: Pré-processador em um Montador Gerando Arquivos ELF
Trabalho Individual

Objetivos

O objetivo deste trabalho é implementar um pré-processador em um montador já existente. O pré-processador deve ser responsável por alterar o código fonte processando as diretivas define, para incluir tanto constantes quanto macros com parâmetros; e include, para incluir arquivos texto no código fonte.

O montador utilizado é um Montador de Assembly 8085, gerando arquivo ELF e símblos de depuração. Este montador deve ser melhorado com:

1) A inclusão do pré-processador com tratamento de macros.
2) Adicionando a possibilidade de usar números negativos e em 4 bases numéricas:

```
mov A, -13 ; -13 em decimal
mov C, 44d ; 44 em decimal
mov B,25h ; 25 em hexadecimal
mov H,333o;  333 em octal
mov HL, 1110111100001010b ; valor em binário
```

3) Possibilitando ao usuário chamar o programa sem passar os nomes dos arquivos. O programa deve solicitar caso o usuário não especifique os nomes na linha de comando.

4) A inclusão de erros para labels mal formadas: compostas de somente números, começando com números, labels iguais a instruções ou diretivas, contendo somente caracteres de pontuação, por exemplo.

5) A execução pelo montador de cálculos simples com constantes, tanto em diretivas quanto em instruções. Conversões de bases nos valores das constantes também devem ser realizadas.

```
.define MAX 10*2 ;constante MAX deve ter valor 20
mov A, 35-17 ; equivalente a mov A, 18
mov B, 55H- 21o ; equivalente a mov B, 68
```

IMPORTANTE:

- Embora o aluno possa definir os algoritmos e estruturas de dados da sua parte do montador, os arquivos que servem de interface entre os módulos devem obedecer a mesma estrutura, definida no formato ELF, e o programa básico do Murilo deve ser mantido inalterado sempre que possível: para tanto, procure trabalhar sempre na etapa de pré-processamento, alterando o código para que o montador do murilo realiza a montagem.
- As especificações do trabalho não cobrem todos os detalhes da implementação: sinta-se a vontade para melhorar ou sofisticar os programas, mas nunca simplificar. O que está aqui descrito é o mínimo a ser feito.
- Em caso de dúvidas, use o fórum do Moodle
- É desejável que os alunos troquem programas fontes e arquivos objeto e executáveis (via fórum de preferência) para possibilitar testes mais amplos e completos, mas cada um deve desenvolver seu próprio montador melhorado, sem compartilhamento de código.

O que deve ser entregue:

1. Um único arquivo .ZIP contendo o código fonte dos programas (ANSI C OBRIGATÓRIO) e arquivos de exemplos, o arquivo README.TXT e arquivos de documentação.
2. Código em 8085 (exemplos):
   a) Cada programa/exemplo deverá conter um cabeçalho explicativo
   b) Os exemplos devem salientar as características dos seus programas do ambiente de desenvolvimento. Exemplo: O código de exemplo deverá mostrar como são tratadas as questões de: símbolos não definidos, símbolos duplicados, instrução não definida ou com operandos incorretos, símbolo externo não definido, símbolo público inexistente ou duplicado etc.
   c) As alterações sugeridas acima devem ser testadas nos programas de teste.
3. Código Fonte (em linguagem C):
   a) Cada arquivo fonte deve conter um cabeçalho explicativo do seu conteúdo
   b) As funções do código fonte devem conter a descrição do que a mesma trata, do algoritmo usado e parâmetros de entrada e valores de retorno.
   c) É desejado que boas práticas de programação sejam empregadas, ou seja: evitar o uso de variáveis globais, nomes expressivos para variáveis e funções, comentários sobre o funcionamento do programa (técnicas aplicadas, algoritmos, etc.), e módulos agrupados em arquivos fontes separados (em vez de um único arquivo fonte enorme e desorganizado).
4. Texto explicativo (README.TXT) deve conter:
   a) Forma de compilação dos programas, explicitando como gerar os programas que compõe o ambiente.
   b) Identificação do autor.
   c) Descrição dos arquivos de exemplo (o que cada exemplo faz e qual característica ele mostra)
   d) Outras informações a respeito da utilização dos programas: parâmetros do programa, localização de arquivos, bugs identificados, etc.
5. Arquivos de documentação devem conter:
   a) Estrutura do código fonte, com descrição das funções, procedimentos, bibliotecas externas utilizadas, estruturas de dados, etc. No caso deste trabalho com o montador do Murilo, tenha o cuidado de identificar sempre as partes que foram feitas pelo autor original e pelo autor atual do trabalho, sendo que tudo que foi executado pelo autor atual deve estar obrigatoriamente documentado.
   b) Identificação das técnicas utilizadas na construção dos programas: algoritmos, fontes de informação externa, código reaproveitado, caso exista, entre outras informações relevantes sobre como os programas foram construídos.

Estrutura do Arquivo Fonte em Assembly

Para simplificar a tarefa de "Parsing", vamos definir alguns blocos para o nosso programa em linguagem de montagem. Esse blocos serão definidos de acordo com as seguintes diretivas:

`.begin <nome>`

Diretiva que marca o início de um programa/módulo (obrigatório para todos os módulos) , onde nome é o nome do módulo/programa.

`.macro`

- diretiva que define a área no programa para o código de macros. Todas as macros utilizadas no programa em questão deverão estar definidas neste bloco.

- A definição da macro obedece ao formato desenvolvido em aula.

NOME_DA_MACRO : MACRO ARG_1, ARG_2, ...

CÓDIGO_DA_MACRO
END_MACRO

- A forma de processar a macro, isto é, como implementar o processador de macros, deve seguir os princípios apresentados em sala de aula.

.data

Todos os símbolos (variáveis e constantes) devem ser definidas nesta área, incluindo símbolos externos e símbolos exportados pelas diretivas EXTERN e PUBLIC.

.text

aqui vai o código do programa. Dentro desse bloco, pode estar definida a diretiva .start, se o módulo contiver o ponto de início (entrada) do programa.

.start <rótulo>

diretiva que indica o rótulo do início do programa. Somente uma diretiva .start em um programa executável, mesmo que o programa seja composto de vários módulos.

.end

diretiva que marca o fim de um programa/módulo

Observações:

1. Considere que o código fonte deve obedecer a estrutura definida acima. Por exemplo, não devem haver símbolos definidos fora a área **.data**: sua existência deve acarretar em erros.
2. O fonte pode estar escrito em caixa alta (maiúsculas) ou caixa baixa (minúsculas).
3. Pode haver linhas em branco no texto.
4. Pode haver linhas apenas com comentários, identificados em seu início por ";"
5. Caso hajam vários arquivos fonte para gerar o mesmo executável, somente um símbolo pode ser definido como operando da diretiva **.start**.

Conjunto de Instruções e Diretivas

Instruções

Ver documentos disponíveis no moodle com detalhes das instruções 8085.

Diretivas

DB n(,n)	Define Byte(s)
DB 'string'	Define Byte ASCII character string
DS nn	Define Storage Block
DW nn(,nn)	Define Word(s)
public label	Símbolos Exportados, disponíveis a outros módulos
extern label	Símbolos exportados de outros módulos
define label valor	Definição de uma constante
include arquivo	Inclui arquivo texto no código fonte

Programas de exemplo

Os programas a seguir são exemplos, alguns foram estudados em sala de aula, e mostram a estrutura básica de um programa em 8085. Eles não devem ser usados como únicos programas de teste pra os

programas do trabalho: outros programas de teste devem ser criados, inclusive com erros para testar a verificação do código pelo programa. Sempre que possível, compartilhe esse códigos via fórum de dúvidas no moodle para que todos os grupos possam também realizar seus testes usando os mesmos programas. A correção dos programas também é feita com base nos exemplos postados: quanto mais exemplos comuns conhecidos de todos existirem, mais justa será a correção.

Exemplo 1:

```
.begin SOMA
;soma dois numeros
.text
.start inicio
inicio: LXI       H,N1
        MOV       A,M
        LXI       H,N2
        MOV       B,M
        ADD       B
        LXI       H,N3
        MOV       M,A
        MVI       A,1
        DCR       A
        HLT
.data
N1: db 10
N2: db 3
N3: ds 1
.end
```

Exemplo 2:

```
.begin  MAX
;calcula o valor máximo em um vetor de
;inteiros
.text
.start max
max:  lxi h,ADD1
      mov c,m
      inx h
      mov a,m
ADD2: inx h
      cmp m
      jnc LESS
      mov a,m
LESS: dcr c
      jnz ADD2
      sta ADD_OUTPUT
      hlt
; data
ADD1:       db 4,1,2,3,15
ADD_OUTPUT: ds 1
.end
```

Exemplo 3:

```
.begin FAT
;fatorial de n
.text
.start start
start: lxi h,n
      mov c,m

      mov b,c
fat:  dcr c
      lxi h,aux
      mov m,c
      jz fim
      call Mult
      lxi h,n
      mov m,a

      mov b,a
      lxi h,aux
      mov c,m
      jmp fat
fim:  hlt
;A = B * C
Mult: MVI   A,0   ;A is product
;add x to accumulating product in A
Loop: ADD   B
      DCR   C     ;y counter--
;until count down to zero
      JNZ   Loop
      RET
.data
n:   db 5
aux: ds 1
.end
```

Exemplo 4:

```
.begin AreaTri
.text
.start start
start:
      LXI H,base    ;get x
      MOV   B,M     ;B is x
      LXI H,altura  ;get y
      MOV   C,M     ;C is y
      CALL  Mult
      LXI   H,area
      MOV   M,A     ;product to memory
      HLT
;subroutine: multiply
;two positive numbers x*y
; B holds x
; C holds y, decrements until 0
; A holds accumulating product
;Loop y times adding B(x) to A
Mult: MVI   A,0   ;A is product
    ;add x to accumulating product in A
Loop: ADD   B
      DCR   C     ;y counter--
      ;until count down to zero
      JNZ   Loop
      RET
.data
base: db 10
altura: db 3
area: ds 1
.end
```

Exemplo 5:

```
;calcula serie de fibonnaci
;ate valor limite
.begin      FIBONACCI
.text
.start   start
start:lxi h,n
      mov c,m
      dcr c
loop: lxi h,older
      mov a,m
      lxi h,old
      add m
      lxi h,new
      mov m,a
      dcr c
      jz final
      lxi h,old
      mov b,m
      lxi h,older
      mov m,b
      lxi h,new
      mov b,m
      lxi h,old
      mov m,b
      jmp loop
final:hlt
.data
n:     db 5
old:   db 1
older: db 0
new:   ds 1
.end
```

Exemplo 6:

Programa em dois arquivos max3.as, modB.as

max3.as

```
.begin MAX3
extern MAX
extern ADD1
public ADD_OUTPUT
.text
.start START
START: lxi h,VETOR1
      lxi d,ADD1
      mov c,m
      inr c
      call COPY
      lxi h,ADD_OUTPUT
      mov a,m
      lxi h,MAX1
      mov m,a
      lxi h,VETOR2
      lxi d,ADD1
      mov c,m
      inr c
      call COPY
      lxi h,ADD_OUTPUT
      mov a,m
```

```
      lxi h,MAX2
      mov m,a
      lxi h,VETOR3
      lxi d,ADD1
      mov c,m
      inr c
      call COPY
      lxi h,ADD_OUTPUT
      mov a,m
      lxi h,MAX3
      mov m,a
      hlt

COPY: mov a,m
      xchg
      mov m,a
      inx h
      xchg
      inx h
      dcr c
      jnz COPY
      call MAX
      ret

.data
ADD_OUTPUT: ds 1
VETOR1: db 10,1,2,3,4,5,5,4,3,2,1
MAX1:   ds 1
VETOR2: db 7,13,2,31,54,55,51,48
MAX2:   ds 1
VETOR3: db 5,11,29,36,4,6
MAX3:   ds 1
.end
```

modB.as

```
.begin MODB
public MAX
public ADD1
extern ADD_OUTPUT
.code
MAX:  lxi h,ADD1
      mov c,m
      inx h
      mov a,m
ADD2: inx h
      cmp m
      jnc LESS
      mov a,m
LESS: dcr c
      jnz ADD2
      sta ADD_OUTPUT
      ret
.data
ADD1: ds 20
.end
```