Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Disciplina Software Básico (DCC008) - 2011_01

Trabalho Prático 03

1 Descrição Geral

Este trabalho envolve a implementação de um expansor de macros para o montador implementado no trabalho prático 2.

2 Informações Importantes

- O trabalho deve ser feito **individualmente**, podendo ser discutido entre os colegas, mas código fonte não poderá ser trocado.
- A data de entrega será divulgada no **Moodle** por meio da criação do recurso para a entrega do trabalho.
- **Política de Atrasos:** A entrega de cada trabalho prático deve ser realizada até a data estipulada na tarefa correspondente do Moodle. As tarefas foram programadas para aceitar submissões atrasadas, mas estas serão penalizadas. A penalização pelo atraso será geométrica com o mesmo conforme a fórmula abaixo:

$$Desconto(\%) = \frac{2^{d-1}}{0,32}$$

  Essa fórmula dá a porcentagem de desconto para d dias de atraso.

  ATENÇÃO: Note que depois de 6 dias o trabalho é avaliado em 0 pontos. Com base na política acima, é altamente recomendável que se esforcem para entregar o TP dentro do prazo para que não tenhamos problemas futuros.
- O trabalho deverá ser implementado **obrigatoriamente na linguagem C**.
- Deverá ser entregue exclusivamente o código fonte com os arquivos de dados necessários para a execução e um arquivo Makefile que permita a compilação do programa nas máquinas UNIX do departamento.
- Além disso, deverá ser entregue uma pequena documentação contendo todas as decisões de projeto que forem tomadas durante a implementação, sobre aspectos não contemplados na especificação, assim como uma justificativa para essas decisões. Esse documento não precisa ser extenso (entre 3 e 5 páginas).
- A ênfase do trabalho está no funcionamento do sistema e não em aspectos de programação ou interface com o usuário. Assim, não deverá haver tratamento de erros, ou seja, pode-se considerar que o programa tratado esteja correto. As operações de entrada e saída podem ser implementadas da forma mais simples possível.

- Todas as dúvidas referentes ao Trabalho Prático 03 serão esclarecidas por meio do fórum, devidamente nomeado, criado no ambiente **Moodle** da disciplina.
- A entrega do trabalho deverá ser realizada por meio do **Moodle**, na tarefa criada especificamente para tal. O que deve ser entregue e em que formato:
  - Pasta contendo os arquivos do trabalho. A pasta deve ser nomeada como `tp3_seulogin`, onde `seulogin` refere-se ao seu login no DCC. Esta pasta deve ser compactada no formato `.tar.gz`. Assim, o pacote final a ser enviado é `tp3_seulogin.tar.gz`
  - Estrutura de diretórios da pasta `tp3_seulogin`: Arquivo `Makefile`, arquivo `documentacao.pdf`, pasta `src` contendo o código fonte e pasta `test` contendo os programas de teste.

3 Características da Máquina Virtual

Como os trabalhos práticos da disciplina são dependentes das etapas anteriores, convém relembrar algumas informações sobre a máquina virtual:

- A menor unidade endereçável nessa máquina é um inteiro.
- Os tipos de dados tratados pela máquina também são somente inteiros.
- A máquina possui uma memória de não menos que 1000 posições e 3 registradores: o `PC` (contador de programas), o `AC` (acumulador) e o `SP` (apontador da pilha).
- As instruções são codificadas em um inteiro. A maioria delas possui um operando, que é uma posição de memória (`M`), especificada conforme o modo de endereçamento existente, também codificados em inteiros. As exceções são as instruções `RET` e `HALT`, que não possuem operandos. Relembrando, o conjunto de instruções da máquina está detalhado na Tabela 1.

Código	Símbolo	Significado	Ação
01	LOAD	Carrega AC	AC ← Memória[PC+M]
02	STORE	Armazena AC	Memória[PC+M] ← AC
03	JMP	Desvio incondicional	PC ← PC + M
04	JPG	Desvia se maior que 0	Se AC > 0, PC ← PC + M
05	JPE	Desvia se igual a 0	Se AC = 0, PC ← PC + M
06	JPL	Desvia se menor que 0	Se AC < 0, PC ← PC + M
07	JPNE	Desvia se diferente de 0	Se AC ≠ 0, PC ← PC + M
08	PUSH	Empilha valor	SP ← SP - 1; Memória[SP] ← Memória[PC+M]
09	POP	Desempilha valor	Memória[PC+M] ← Memória[SP]; SP ← SP + 1;
10	READ	Lê valor para a memória	Memória[PC+M] ← "Valor lido"
11	WRITE	Escreve conteúdo da memória	"Imprime" Memória[PC+M]
12	CALL	Chamada de subrotina	SP ← SP - 1; Memória[SP] ← PC; PC ← PC + M
13	RET	Retorno de subrotina	PC ← Memória[SP]; SP ← SP + 1
14	ADD	Soma operando em AC	AC ← AC + Memória[PC+M]
15	SUB	Subtrai operando de AC	AC ← AC - Memória[PC+M]
16	XOR	XOR lógico	AC ← AC **xor** Memória[PC+M]
17	AND	AND lógico	AC ← AC **and** Memória[PC+M]
18	OR	OR lógico	AC ← AC **or** Memória[PC+M]
19	HALT	Parada	

Tabela 1: Conjunto de instruções da máquina virtual.

Além das instruções acima, existem também as pseudo-instruções `WORD` e `END`:

- `WORD A` → Usada para alocar uma posição de memória cujo valor será `A`, ou seja, quando houver tal instrução, a posição de memória que está sendo referenciado pelo `PC` deverá receber o valor `A`
- `END` → Indica o final do programa para o montador

4 Descrição do Expansor de Macros

O Expansor de Macros deve ser, basicamente, um programa que receberá um arquivo de entrada, identificará as macros que foram definidas e as substituirá nos locais em que elas foram utilizadas, gerando um arquivo que pode ser utilizado como entrada para o montador implementado no Trabalho Prático 2.

A seguir serão descritos alguns detalhes para a implementação do expansor de macros.

4.1 Pseudo-instruções

Para a implementação do expansor de macros são necessárias duas pseudo-instruções:

- `BEGINMACRO` → Indica o início da definição de uma macro.
- `ENDMACRO` → Indica o fim da definição de uma macro.

4.2 Características das Macros

- As macros podem ser definidas em qualquer ponto do programa. O seu expansor será responsável pela interpretação correta do que é definição de macro, o que é código direto e o que é chamada de macro;
- As macros podem ter **um parâmetro** ou não ter parâmetro;
- **Não ocorre** definição de uma macro dentro de outra macro;
- O nome da macro será definido antes da pseudo-instruções BEGINMACRO seguido por dois pontos (e.g. $< nome\_macro >$: BEGINMACRO).

4.2.1 Macro sem Parâmetro

Exemplo:

```
MACRO1: BEGINMACRO
LOAD A
ADD B
STORE A
ENDMACRO
MACRO1
MACRO1
```

Ao ser processado pelo expansor de macros, deverá ser gerado o código:

```
LOAD A
ADD B
STORE A
LOAD A
ADD B
STORE A
```

4.2.2 Macro com Parâmetro

Exemplo:

```
MACRO2: BEGINMACRO Y
LOAD Y
ADD B
STORE Y
ENDMACRO
MACRO2 A
MACRO2 J
```

Ao ser processado pelo expansor de macros, deverá ser gerado o código:

```
LOAD A
ADD B
STORE A
LOAD J
ADD B
STORE J
```

5 Descrição da Tarefa

Os alunos deverão implementar o expansor de macros definido acima. Além disso, deverão ser criados dois programas de testes, **obrigatoriamente** contendo macros (definidas pelo aluno), a saber:

- **Exponenciação:** Programa que leia dois números a e b e faça a elevado a b. Exemplo:

$$2^3 = 8$$

- **Mediana:** Programa que leia 7 números e imprima a mediana deles. Exemplo:

$$valores = 1, 2, 3, 4, 5, 6, 7$$

$$mediana = 4$$

.

6 Formato da Entrada de Dados

A entrada do expansor de macros deve seguir o mesmo formato especificado para o montador implementado anteriormente. A única diferença é a presença da definição e utilização de macros.

Exemplo:

```
READ A
READ B
LOAD A
STORE C
SUB B
JPG L
LOAD B
STORE C
L: PRINT
HALT
A: WORD 0
B: WORD 0
C: WORD 0
PRINT: BEGINMACRO
WRITE A
```

```
WRITE B
WRITE C
ENDMACRO
END
```

Para um teste inicial do seu expansor de macros, utilize o programa acima.
Atenção: Tal programa não testa todas as instruções e não deve ser utilizado como único teste do seu expansor.

7 Formato da Saída de Dados

Corresponde ao formato do arquivo de entrada do Trabalho Prático 2, já que o montador será utilizado para passar o código de saída do expansor de macros para o executável que a máquina virtual reconhece.

8 Formato de Chamada do Expansor de Macros

O expansor de macros receberá apenas 2 argumentos quando da sua chamada:

- Nome do arquivo de entrada: contendo o programa a ter macros expandidas – informado como **primeiro argumento** na chamada da aplicação.
- Nome do arquivo de saída: informado como **segundo argumento** na chamada da aplicação.

Exemplo:

```
./expansor teste1 saida1
```

A chamada acima tem a seguinte semântica: executar o expansor para expandir as macros contidas no programa do arquivo `teste1` para o formato aceito pelo montador e gravar o resultado no arquivo `saida1`. Note que a única saída de dados do expansor é o arquivo de saída. Não é necessário imprimir informações na tela.

9 Sobre a Documentação

- Deve conter as decisões de projeto.
- Deve conter as informações de como executar o programa. Obs.: é necessário cumprir os formatos definidos acima para a execução, mas tais informações devem estar presentes também na documentação.
- Não incluir o código fonte no arquivo de documentação.
- Deve conter elementos que comprovem que o programa foi testado (e.g. entradas, saídas, imagens das telas de execução). Os arquivos relativos a testes devem ser enviados no pacote do trabalho, conforme descrito na Seção 2. A documentação deve conter as referências a esses arquivos, o que eles fazem e os resultados obtidos.

10 Considerações Finais

É obrigatório o cumprimento fiel de todas as especificações de interface descritas neste documento. As decisões de projeto devem fazer parte apenas da estrutura interna do expansor, não podendo afetar a interface de entrada e saída.