Toy Assembly

Programação de Computadores I

Ciência da Computação

Prof. Daniel Saad Nogueira Nunes

1 Introdução

Se denominam linguagens assembly aquelas linguagens de baixo nível, que estão em um nível de abstração bem próximo a da arquitetura da máquina programada.

O objetivo deste projeto é construir um interpretador de uma linguagem assembly simples (toy assembly).

2 Especificação

Neste *assembly*, os programas são escritos com uma instrução por linha, que são numeradas a partir de 0. O assembly possui acesso a 32 registradores de inteiros de 32-bits, identificados de `R0` a `R31`, que devem ser inicializados com zero na execução de um programa. Além disto, os programas têm acesso a uma memória com capacidade para 1000 inteiros com sinal de 32-bits, que também deve ser inicializada com zeros.

Considerando `RX`, `RY` e `RZ` ($0 \leq \texttt{X}, \texttt{Y}, \texttt{Z} < 32$) três registradores, `INTEIRO` um número inteiro com sinal de 32-bits e `ENDERECO` um inteiro sem sinal representando o número da linha ocupada por uma instrução. Temos que o conjunto de instruções que deve ser suportada pelo interpretador é o seguinte:

- `MOV RX INTEIRO`: atribui a `RX` o valor do `INTEIRO`
- `MOV RX RY`: atribui a `RX` o valor de `RY`.
- `ADD RX RY RZ`: soma os valores de `RY` e `RZ` e armazena o resultado em `RX`.
- `SUB RX RY RZ`: subtrai `RZ` de `RY` e armazena o resultado em `RX`.
- `MUL RX RY RZ`: multiplica `RY` por `RZ` e armazena o resultado em `RX`.
- `DIV RX RY RZ`: divide `RY` por `RZ` e armazena o resultado em `RX`. A divisão é inteira, isto é, a parte fracionária é desprezada.
- `MOD RX RY RZ`: toma o resultado de `RY` *mod* `RZ` e armazena o resultado em `RX`. O resultado `RX` da operação de mod é sempre $0 \leq \texttt{RX} < \texttt{RZ}$.
- `BEQ RX RY ENDERECO`: o fluxo do programa é direcionado para a instrução que ocupa a linha de número `ENDERECO` caso `RX` seja igual a `RY`.
- `BLT RX RY ENDERECO`: o fluxo do programa é direcionado para a instrução que ocupa a linha de número `ENDERECO` caso `RX` seja menor que `RY`.
- `JMP ENDERECO`: o fluxo do programa é direcionado para a instrução que ocupa a linha de número `ENDERECO`.
- `LOAD RX RY`: carrega o conteúdo de memoria[`RY`] para o registrador `RX`.
- `STORE RX RY`: carrega o conteúdo de `RX` para memoria[`RY`].
- `PRINT RX`: imprime uma linha na tela com o o valor de `RX`.

- `EXIT`: encerra o programa.

Observação: se após executar 10^5 instruções e o programa não tiver alcançado a instrução `EXIT`, ele deverá ser encerrado mesmo assim.

2.1 Modularização

O sistema deverá ser dividido em módulos, cada qual com uma tarefa. Estes módulos podem ser organizados internamente através de várias funções e eles correspondem aos seguintes:

- Módulo de leitura: efetua a leitura do programa de entrada.
- Módulo de saída: efetua a impressão dos dados.
- Módulo de controle: responsável por decodificar e executar as instruções.
- Módulo de operações aritméticas: realiza as operações aritméticas.
- Módulo de operações lógicas: realiza as operações lógicas (`BEQ`, `BLT` e `JMP`).
- Módulo de operações em memória: realiza as operações sobre a memória (`LOAD` e `STORE`).
- Módulo principal: contém a função `main` e as chamadas das funções exportadas pelos outros módulos. O ideal é que este módulo possua uma quantidade muito pequena de código, já que ele vai utilizar funções que estão presentes nos outros módulos.

Os módulos devem ser organizados em arquivos separados, com seus respectivos arquivos de cabeçalho e implementação.

2.2 Construção do sistema

Um `Makefile` deverá ser produzido para a compilação dos códigos-fontes no executável e deverá ser distribuído junto ao código.

2.3 Documentação

O código deve ser bem documentado, com presença de comentários explicando os trechos mais complexos do código. Além disso, um arquivo `README.md` deve ser providenciado com a devida identificação do autor descrevendo o projeto e instruindo como o código deve ser compilado através da ferramenta `make`.

2.4 Entrada

A entrada deve ser lida do teclado (`stdin`).

A primeira linha da entrada possui um inteiro N ($1 \leq N \leq 100$), que indica o número de linhas do programa.

As próximas N linhas contém as instruções do programa. É garantido que o programa é sintaticamente correto.

2.5 Saída

Para cada comando `PRINT` deverá ser impresso na tela (`stdout`) o valor do registrador correspondente em uma linha.

2.6 Exemplos

O programa abaixo corresponde a um programa que soma $1+1$ e imprime o resultado na tela:

- Entrada:

```
5
MOV R0 1
MOV R1 1
ADD R0 R1 R1
PRINT R0
EXIT
```

- Saída:

```
2
```

O programa abaixo imprime os 10 primeiros números da sequência de Fibonacci:

- Entrada:

```
16
MOV R0 1
MOV R1 1
MOV R2 2
MOV R3 2
MOV R4 10
MOV R5 1
PRINT R0
PRINT R1
BEQ R3 R4 15
ADD R2 R1 R0
PRINT R2
MOV R0 R1
MOV R1 R2
ADD R3 R3 R5
JMP 8
EXIT
```

- Saída:

```
1
1
2
3
5
8
13
21
34
55
```

O programa abaixo insere os valores 1 e 2 nas posições 0 e 1, imprime o conteúdo destas posições, troca os conteúdos destas posições (swap), e os imprime novamente .

- Entrada:

```
15
MOV R4 0
MOV R5 1
MOV R0 1
MOV R1 2
STORE R0 R4
STORE R1 R5
PRINT R0
PRINT R1
LOAD R0 R5
LOAD R1 R4
STORE R0 R4
STORE R1 R5
PRINT R0
PRINT R1
EXIT
```

- Saída:

```
1
2
2
1
```

3 Critérios de correção

Deve ser utilizada a linguagem de programação `C` para a implementação do interpretador.

Para validação da correção do algoritmo, testes automatizados serão realizados, então é **crucial** que a saída esteja conforme o especificado.

Serão descontados pontos dos códigos que não possuírem indentação.

Os códigos deverão ser documentados e obrigatoriamente devem possuir a presença de um arquivo `Makefile`.

3.1 Ambiente de Correção

Para a correção dos projetos, será utilizada uma máquina de 64-bits com sistema operacional `GNU/LINUX` e compilador `GCC 10.2.0`, logo é imprescindível que o sistema seja capaz de ser compilado e executado nesta configuração.

4 Considerações

- `GDB`, `Valgrind` e ferramentas gráficas associadas podem ajudar na depuração do código.
- Este projeto deve ser executado **individualmente**.
- Não serão avaliados códigos que não compilem ou sem a presença de um `Makefile`.
- Como a correção é automatizada, deverá ser impresso apenas o que a especificação pede. Atente-se para a formatação da saída.
- A incidência de plágio será avaliada automaticamente com nota 0 para os envolvidos. Medidas disciplinares também serão tomadas.
- O trabalho deve ser entregue dentro de uma pasta zipada com a devida identificação do aluno no prazo combinado pelo ambiente virtual de aprendizagem da disciplina.