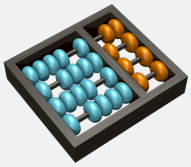
Instituto de Computação - UNICAMP

MC102 - Algoritmos e Programação de Computadores



Laboratório 09: A entrada do meu programa é um programa?

Segunto Semestre de 2017 - Turmas Coordenadas

Peso da Atividade: 3

Prazo de Entrega: 03 de novembro de 2017 às 23:59:59

Conteúdo

Contexto

Sim.

Resposta à pergunta do título

Figura 1: Apontadores. Extraído daqui.

Um sistema simples de computação pode ser composto dos seguintes itens:

- **ULA** - **U**nidade **L**ógica **A**ritmética, responsável por realizar operações aritméticas simples, por exemplo, somas e subtrações;
- **Memória de programa** - onde estão armazenados os programas que são executados pelo computador, por exemplo, o navegador que você está utilizando para ler este enunciado é um dos programas armazenados nessa memória;
- **PC** - **P**rogram **C**ounter é um apontador que diz qual será a próxima instrução executada pelo computador. Qualquer desvio condicional altera o estado do **PC**;
- **Memória de dados** - onde estão armazenados alguns valores temporários, por exemplo, as variáveis declaradas no seu laboratório de MC102; e
- **Dispositivos de entrada e saída** - periféricos do computador, como mouse, teclado, monitor, impressora, leitor biométrico, etc.

As unidades lógicas aritméticas são circuitos eletrônicos muito eficientes, que realizam apenas operações aritméticas. Essas unidades geralmente recebem um ou dois valores de entrada e um código que representa a operação que deve ser executada. Com a operação definida, a unidade produz um resultado na saída.

A memória de programa armazena cada uma das instruções que devem ser executadas pelo computador. Os programas são armazenados como um conjunto de `0`s e `1`s. Como é feita a transformação de um programa em linguagem C, por exemplo, nesse conjunto de `0`s e `1`s? O compilador é responsável por esse processo. Nesta disciplina é utilizado o `gcc` para compilar os seus programas e transformá-los em uma linguagem conhecida pelo computador.

Durante o processo de compilação o programa é submetido a diversas etapas. Uma dessas etapas transforma instruções da linguagem C em conjuntos de instruções que o computador é capaz de executar. Essas instruções formam a *Linguagem de Montagem* ou *Assembly* que um determinado computador é capaz de executar. Em seguida, o código resultante da linguagem de montagem é submetido a um *montador* que executa a transformação dessas instruções em um conjunto de `0`s e `1`s. Uma visão bem simplificada do processo pode ser vista na Figura 2.

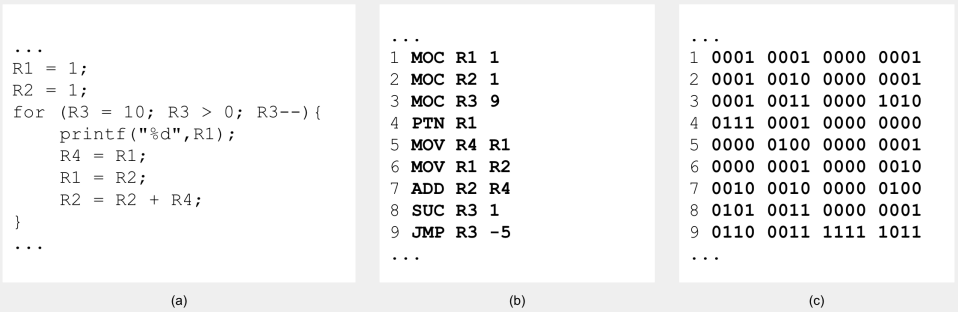


Figura 2: Exemplo de compilação de um programa em C. (a) Trecho de um código em linguagem C. (b) O programa transcrito em linguagem de montagem (Assembly). (c) A representação em linguagem de máquina do programa.

A memória de dados contém circuitos ligados às entradas e saída da ULA que armazenam valores temporários (como se fossem variáveis do seu programa em C). Chamamos a memória de dados que está ligada diretamente à ULA de registrador (ou banco de registradores).

Como podem perceber, podemos ter um computador bastante simples (poucas instruções), que seja capaz de realizar muitas operações. Assim, o poder de transformar simples manipulações de bits em tudo o que temos nos nossos computadores está nas mãos de quem escreve os programas, ou seja, nas suas mãos.

Imagine um computador bastante simples, como o mostrado na Figura 3. Ele contém todos os itens exibidos na lista acima. Vamos chamá-lo de MC102.

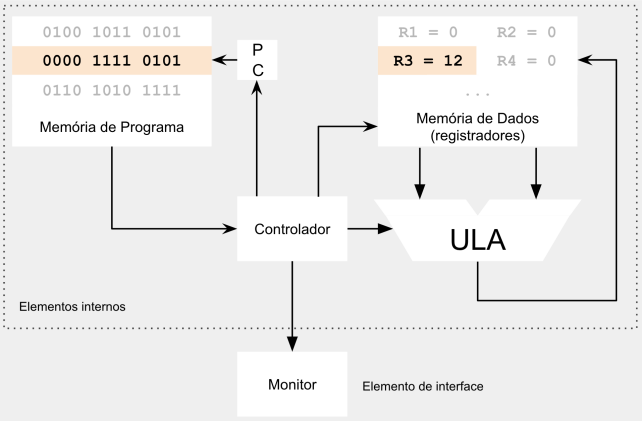


Figura 3: Arquitetura do computador MC102.

Por ser bastante simples, esse computador realiza apenas operações de soma e subtração e cópias de valores sobre seus registradores. Ao todo, a linguagem de montagem que é aceita por esse computador, e é chamada de *MCASSEMBLY102*, possui nove instruções. Todas elas são compostas por três letras, que representam de forma mnemônica a operação realizada pelo computador. Dependendo da operação ela pode receber um ou dois parâmetros, conforme a Tabela 1, apresentada a seguir.

#	Mnemônico	Parâmetro 1	Parâmetro 2	Descrição
1	MOV	Rd	Ro	O registrador de destino Rd recebe o valor armazenado no registrador de origem Ro (Rd ← Ro)
2	MOC	Rd	N	Rd recebe o valor do inteiro N (Rd ← N)
3	ADD	Rd	Ro	Rd recebe a soma dele mesmo com o valor de Ro (Rd ← Rd + Ro)
4	ADC	Rd	N	Rd recebe a soma dele mesmo com o valor do inteiro N (Rd ← Rd + N)
5	SUB	Rd	Ro	Rd recebe a subtração dele mesmo com o valor de Ro (Rd ← Rd - Ro)
6	SUC	Rd	N	Rd recebe a subtração dele mesmo com o valor do inteiro N (Rd ← Rd - N)
7	JMP	Ro	N	Se o valor armazenado em Ro for maior ou igual a zero, realiza um salto de N instruções. Caso contrário continua a execução do programa. Observe que N pode ser positivo ou negativo. Se N é positivo, o PC avança N instruções. Caso contrário, o PC retrocede N instruções. (IF (Ro ≥ 0) pula N instruções ELSE continua)

8	PTN	Ro	Imprime o inteiro armazenado no registrador Ro seguido de um espaço em branco
9	PTC	Ro	Imprime o caracter correspondente ao valor do registrador Ro sem adição de espaços adicionais

Tabela 1: Conjunto de instruções da linguagem MCASSEMBLY102.

Dessa forma, vamos analisar o programa em linguagem de montagem exibido na Figura 2(b):

```
1 MOC R1 1
2 MOC R2 1
3 MOC R3 9
4 PTN R1
5 MOV R4 R1
6 MOV R1 R2
7 ADD R2 R4
8 SUC R3 1
9 JMP R3 -5
```

No início, o **PC** aponta para a primeira linha do programa (linha 1).

`1 MOC R1 1` : a constante 1 é copiada para o registrador R1. O **PC** é incrementado e agora aponta para a segunda linha do programa (PC ← PC + 1).
`2 MOC R2 1` : a constante 1 é copiada para o registrador R2 e PC ← PC + 1.
`3 MOC R3 9` : a constante 9 é copiada para o registrador R3 e PC ← PC + 1.
`4 PTN R1` : é impresso no monitor o número contido no registrador R1 (valor 1) e um espaço em branco e PC ← PC + 1.
`5 MOV R4 R1`: o valor do registrador R1 é copiado para o registrador R4 e PC ← PC + 1.
`6 MOV R1 R2`: o valor do registrador R2 é copiado para o registrador R1 e PC ← PC + 1.
`7 ADD R2 R4`: o valor do registrador R4 é somado com o valor do registrador R2 e o resultado é armazenado no registrador R2 e PC ← PC + 1.
`8 SUC R3 1` : o valor do registrador R3 é subtraído de uma unidade e o resultado é armazenado no registrador R3 e PC ← PC + 1.
`9 JMP R3 -5`: verifica se R3 é maior ou igual a zero. Como R3 é igual a oito, PC é atualizado. Então, PC ← PC - 5 e o novo comando que será executado é o da quarta linha.

`4 PTN R1` : é impresso no monitor o número contido no registrador R1 (valor 1) e um espaço em branco e PC ← PC + 1.
`5 MOV R4 R1`: o valor do registrador R1 é copiado para o registrador R4 e PC ← PC + 1.
`6 MOV R1 R2`: o valor do registrador R2 é copiado para o registrador R1 e PC ← PC + 1.
`7 ADD R2 R4`: o valor do registrador R4 é somado com o valor do registrador R2 e o resultado é armazenado no registrador R2 e PC ← PC + 1.
`8 SUC R3 1` : o valor do registrador R3 é subtraído de uma unidade e o resultado é armazenado no registrador R3 e PC ← PC + 1.
`9 JMP R3 -5`: verifica se R3 é maior ou igual a zero. Como R3 é igual a sete, PC é atualizado. Então, PC ← PC - 5 e o novo comando que será executado é o da quarta linha.

...

`4 PTN R1` : é impresso no monitor o número contido no registrador R1 (valor 55) e um espaço em branco e PC ← PC + 1.
`5 MOV R4 R1`: o valor do registrador R1 é copiado para o registrador R4 e PC ← PC + 1.
`6 MOV R1 R2`: o valor do registrador R2 é copiado para o registrador R1 e PC ← PC + 1.
`7 ADD R2 R4`: o valor do registrador R4 é somado com o valor do registrador R2 e o resultado é armazenado no registrador R2 e PC ← PC + 1.
`8 SUC R3 1` : o valor do registrador R3 é subtraído de uma unidade e o resultado é armazenado no registrador R3 e PC ← PC + 1.
`9 JMP R3 -5`: verifica se R3 é maior ou igual a zero. Como R3 é igual a -1, o salto não é realizado. Então, PC ← PC + 1.

Como não há mais instruções que devem ser executadas, o programa pára.

MCASSEMBLY102 também possui suporte a comentários. Todas as linhas que começam com o caractere `#` são ignoradas pelo montador e **não** são transformadas na linguagem que a máquina lê.

Tarefa

Você deve escrever um programa em linguagem C que leia um programa em MCASSEMBLY102 e simule a sua execução no computador MC102. A saída do seu programa deve ser o resultado das operações de impressão.

A primeira linha da entrada do seu programa é um inteiro `n > 1`, que representa o número de linhas do programa que será interpretado. Em seguida as `n` linhas, tal que cada uma delas contém uma instrução dentre as propostas na Tabela 1 ou um comentário.

Observações da Tarefa

- Você *deve* inserir uma quebra de linha `\n` após o término da simulação do programa de entrada.
- Toda linha que começa com o caractere `#` representa um comentário e deve ser ignorada. Uma linha de comentário possui no máximo `50` caracteres. Não existem linhas em branco ou comentários em linhas que contém instruções.

- Um programa terá, no máximo, `150` **instruções**. Note que o valor de `n` pode ser maior que `150`.
- O número máximo de registradores suportados pelo seu interpretador é `12`, sendo numerados de `R1` até `R12`.
- MCASSEMBLY102 não é uma linguagem do tipo *case sensitive*. Contudo, considere que os programas estão escritos com todos os caracteres em maiúsculas.
- Para esta atividade não é permitido o uso de colchetes para acessar elementos de vetores ou matrizes. Você **deve** utilizar apontadores de forma explícita para isso.
- Cada instrução apresentada na Tabela 1 deve ser implementada como sendo uma função. Os cabeçalhos das funções a serem implementadas estão descritas no arquivo auxiliar `lab09.h`. Essas funções **devem** ser implementadas e invocadas pelo seu programa. Não é permitido alterar este arquivo, nem os parâmetros das funções.

Exemplos

> *Notas:*
>
> Textos em `azul` designam dados de entrada, isto é, que devem ser lidos pelo seu programa.
> Textos em `preto` designam dados de saída, ou seja, que devem ser impressos pelo seu programa.

Exemplo de execução 1:

```
10
# PROGRAMA 1 DA FIGURA 2B
MOC R1 1
MOC R2 1
MOC R3 9
PTN R1
MOV R4 R1
MOV R1 R2
ADD R2 R4
SUC R3 1
JMP R3 -5

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

Exemplo de execução 2:

```
23
# PROGRAMA FOFO DE EXEMPLO
MOC R1 69
PTC R1
MOC R1 117
PTC R1
MOC R1 32
PTC R1
MOC R1 115
PTC R1
MOC R1 50
PTC R1
MOC R1 32
PTC R1
MOC R1 77
PTC R1
MOC R1 67
PTC R1
MOC R1 49
PTC R1
MOC R1 48
PTC R1
MOC R1 50
PTC R1

Eu s2 MC102
```

Observações Gerais

- O número máximo de submissões é 20.
- O arquivo `lab09.c` deve conter todo o seu programa, incluindo o corpo de todas as funções.
- Para a realização dos testes automáticos, a compilação se dará da seguinte forma: `gcc lab09.c -o lab09 -Wall -Werror -ansi -pedantic` .
- Não se esqueça de incluir no início do programa uma breve descrição dos objetivos, da entrada, da saída, seu nome, RA e turma.
- Após cada submissão, você deve aguardar um minuto até poder submeter seu trabalho novamente.

- Ao final deste laboratório, você terá aprendido como utilizar apontadores.

Critérios Importantes

O **não** cumprimento dos critérios abaixo acarretará em **nota zero na atividade**, independentemente dos resultados dos testes do SuSy.

- Sua solução deve atender todos os requisitos definidos no enunciado.
- Não serão aceitas soluções contendo estruturas não vistas em sala (para este laboratório, poderão ser utilizadas apenas variáveis simples, vetores, matrizes, operações de entrada e saída, operações aritméticas, desvios condicionais, estruturas de repetição, strings, funções e apontadores).
- Não é permitido o uso de `continue` e `break` (exceto em estruturas do tipo *switch-case*).
- Não é permitido o uso de variáveis globais.
- Não é permitido o uso de colchetes para acessar elementos de vetores ou matrizes.
- Cada função deve conter apenas um único `return`.
- As funções indicadas no arquivo `lab09.h` devem ser implementadas.
- Os únicos cabeçalhos aceitos para inclusão são `stdio.h`, `string.h` e `lab09.h`.