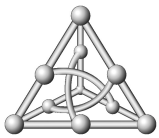


UNIVERSIDADE FEDERAL DE MATO GROSSO DO SUL
Faculdade de Computação

# Interpretador de linguagem baixo-nível

# Trabalho 2

**Algoritmos e Programação II**

## 1 Descrição

Figura 1: Capivara codificando em Capivariton.

Capivaras são nossas velhas conhecidas. Passam os dias e noites na FACOM, onde aproveitam para estudar. Ao contrário das linguagens de programação de alto-nível (como C ou Python), linguagens de baixo-nível (ou linguagens de montagem/*assembly languages*), possuem muito menos abstrações e possuem operadores/instruções muito próximos das instruções executadas pelos processadores.

Estudando linguagens de baixo-nível para operar o novíssimo microcontrolador *CapyAwesome*, desenvolvido pela *Capybara Coorp.*, as capivaram chegaram à conclusão de que todas linguagens existentes são desnecessariamente complicadas. Então resolveram definir sua própria linguagem de progamação de baixo-nível, chamada *Capivariton*, projetada especificamente para operar aquele microcontrolador. Contudo, ainda não há nenhum compilador ou mesmo interpretador para testar o Capivariton, e por isso elas precisam de sua ajuda.

Sua tarefa será escrever um interpretador para o Capivariton. As capivaras escreveram o conjunto de especificações da linguagem, transcrito a seguir com explicações adicionais.

### Registradores

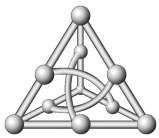
Nessa linguagem (assim como em outras de baixo-nível), não temos variáveis. Temos apenas um pequeno conjunto de **registradores**, que são pequenas unidades de memória dentro do processador. Os registradores da Capivariton **armazenam sempre inteiros** e **são inicializados com 0 no início do programa**. Eles são:

**`acc`** (*accumulator*) é o principal registrador de uso geral usado para computação interna nos microcontroladores CapiAwesome. **Todas as operações aritméticas usam e modificam implicitamente o valor de `acc`**.

**`dat`** (*data*) é um segundo registrador disponível na linguagem, que pode ser utilizado para armazenar qualquer valor.

**`ext`** (*extra*) é um registrador extra, que também pode armazenar qualquer valor.

**`pc`** (*program counter*) armazena o número da instrução atual na sequência do programa, começando em 0 (1ª instrução do programa). O valor em `pc` nunca é manipulado diretamente.





## INSTRUÇÕES

Programas em Capivariton são construídos a partir de um pequeno, mas versátil conjunto de instruções. Cada linha de um programa em Capivariton pode conter uma instrução, que consiste no nome da instrução seguido por um ou dois operandos, dependendo da instrução. Além disso, cada linha possui no máximo 128 caracteres. A seguir descreveremos detalhadamente as instruções disponíveis e como elas funcionam.

Cada tipo de instrução requer um número fixo de operandos. Se uma instrução possuir operandos associados, eles devem aparecer após o nome da instrução, separados por espaços. Operandos de instruções são descritos com a seguinte notação:

| Notação | Significado |
|---|---|
| `R` | Registrador |
| `I` | Inteiro |
| `R/I` | Registrador ou inteiro |

As instruções são:

**`mov`** `R/I R`
(*move*) Copia o valor do 1º operando para o 2º operando.

**`add`** `R/I`
(*add*) Adiciona o valor do 1º operando ao valor no registrador `acc` e armazena o resultado de volta em `acc`.

**`sub`** `R/I`
(*subtract*) Subtrai do valor no registrador `acc` o valor do 1º operando e armazena o resultado de volta em `acc`.

**`mul`** `R/I`
(*multiply*) Multiplica o valor do 1º operando pelo valor no registrador `acc` e armazena o resultado de volta em `acc`.

**`div`** `R/I`
(*divide*) Divide o valor no registrador `acc` pelo valor do 1º operando e armazena o resultado de volta em `acc`.

**`mod`** `R/I`
(*modulo*) Calcula o resto da divisão do valor no registrador `acc` pelo valor do 1º operando e armazena o resultado de volta em `acc`.

**`jmp`** `I`
(*jump*) Avança no programa a quantidade de instruções definida no 1º operando (se positivo) ou retrocede (se negativo), atualizando indiretamente o registrador `pc`. O 1º operando nunca será 0.

**`jeq`** `I`
(*jump if equal to 0*) Como `jmp`, mas apenas se o valor no registrador `acc` é igual a 0 .

**`jlt`** `I`
(*jump if less than 0*) Como `jmp`, mas apenas se o valor no registrador `acc` é menor que 0.

**`jgt`** `I`
(*jump if greater than 0*) Como `jmp`, mas apenas se o valor no registrador `acc` é maior que 0.

**`prt`** `R/I`
(*print*) Exibe na tela em uma linha o valor do 1º operando.

### COMENTÁRIOS E ESPAÇOS

Em programas escritos na linguagem Capivariton, múltiplos espaços são ignorados, portanto uma instrução `mov  8  acc` é equivalente a `mov 8 acc`. Outro ponto importante é que qualquer texto após um símbolo "**#**"durante a leitura do programa é considerado um comentário e faz com que o restante da linha seja ignorado. Além disso, linhas vazias ou com apenas comentários não contam como instruções e são completamente ignoradas na leitura. Veja um exemplo abaixo (os números das linhas **não** fazem parte do programa e estão lá apenas para você compreender como as **instruções** são numeradas, as linhas sem instruções não estão numeradas):

```
0.  mov 0 acc
    # abaixo realizaremos uma soma (esta linha não conta)
1.  add 8
2.  prt acc
```

### EXECUÇÃO DO PROGRAMA

O programa inicia na instrução 0 (`pc` $= 0$). A cada instrução, o interpretador (sempre mantendo `pc` atualizado):

- Se a instrução atual **não** é uma instrução de *jump*, executa a operação e avança para a próxima instrução;
- Se a instrução atual é `jmp I`, avança ou retrocede `I` instruções;
- Se a instrução atual é `jeq I`, `jlt I`, `jgt I`, avança ou retrocede I instruções apenas se o valor de `acc` é igual, menor ou maior que 0 respectivamente, caso contrário avança para a próxima instrução.

O programa termina após passar pela última instrução. Em outras palavras, se seu programa tem $n$ instruções, elas são numeradas de 0 a $n - 1$ e seu programa termina quando o registrador `pc` chegar em $n$.

### PROGRAMAS INVÁLIDOS

Embora um interpretador completo deva ser capaz de identificar e reportar erros, iremos implementar apenas uma versão básica inicial do interpretador. Portanto, vamos supor que não serão fornecidos programas com instruções inválidas ou com *jumps* para antes de 0 ou após $n$ (onde $n$ é a quantidade de instruções do programa).

## 2 Exigências

Queremos ler o programa Capivariton do disco **apenas uma vez**. Dessa forma, seu programa interpretador em C DEVE receber por **linha de comando** um único parâmetro, que corresponde a um arquivo (geralmente com extensão `.cap`) contendo o programa em Capivariton a ser interpretado, por exemplo, `./interpretador teste.cap`. Seu interpretador deve ler todas as instruções do arquivo passado, DEVE **armazená-las em uma lista duplamente encadeada e só então realizar a interpretação do código Capivariton**. Essa exigência se deve ao fato de não haver limite para o tamanho do programa e não sabermos quantas instruções ele têm, portanto não há como alocar um vetor para armazená-las. Além disso, o encadeamento duplo se deve ao fato de podermos precisar avançar ou retroceder na sequência de instruções pelo uso de *jumps*.

Você DEVE usar a seguinte estrutura de dados em seu trabalho para implementar a lista duplamente encadeada:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_OP 3
#define MAX_LINE 128

typedef struct instr {
   char op[MAX_OP+1];   /* Operação */

   char reg1[MAX_OP+1]; /* Registrador do operando 1, se registrador */
   int val1;            /* Valor do operando 1, se inteiro */

   char reg2[MAX_OP+1]; /* Registrador do operando 2, se houver */

   struct instr *prev;  /* Anterior */
   struct instr *next;  /* Próximo */
} instruction;
```

Por fim, considere que uma linha do arquivo de entrada possui no máximo 128 caracteres.

## 3 Entrada e saída

**Não haverá entrada habitual**[1] no sentido de valores digitados para o seu interpretador. Como descrito anteriormente, seu interpretador deve ler dados de um arquivo passado por linha de comando, que contém código em Capivariton.

Como **saída**[2], seu interpretador deve imprimir apenas operandos de instruções `prt`.

[1] A entrada habitual consiste naquilo que seu programa lê com *scanf*
[2] A saída consiste apenas naquilo que o programa escreve, utilizando *printf* e similares

# 4 Exemplos de programas em Capivariton e as saídas corretas

Veja a seguir alguns exemplos de programas em Capivariton e suas saídas corretas. **Os números das linhas NÃO fazem parte dos programas e estão lá apenas para você compreender como as instruções são numeradas**, as linhas sem instruções não estão numeradas.

### Exemplo 1 (programa em Capivariton)

```
0.  mov 0 acc
    # um comentário (esta linha não conta)
    # abaixo realizaremos uma soma (esta linha não conta)
1.    add   8 # pode ter espacos antes da instrução, abaixo tem uma linha vazia

2.  prt acc
```

### Exemplo 1 (saída correta)

```
8
```

### Exemplo 2 (programa em Capivariton)

```
0.  jmp 2
1.  jmp 5
2.  jmp +1 # não faz diferença ter um + antes do numero positivo
3.  prt pc
4.  prt pc
5.  jmp -4
6.  prt pc
```

### Exemplo 2 (saída correta)

```
3
4
6
```

### Exemplo 3 (programa em Capivariton)

```
     # calcula o fatorial de n

     # acc guarda o n atual
     # dat guarda o fatorial atual

 0.  mov 5 acc   # define n = 5 e guarda em acc
 1.  mov 1 dat   # fatorial inicial = 1

 2.  jeq 8       # while n != 0
 3.  mov acc ext #   faz copia de n atual
 4.  mov dat acc #   coloca fat atual em acc
 5.  mul ext     #   multiplica fat por n
 6.  mov acc dat #   copia fat para dat
 7.  mov ext acc #   volta n para acc
 8.  sub 1       #   faz n--
 9.  jmp -7      #   volta p/ a comparação do laço

10.  prt dat     # escreve resposta
```

### Exemplo 3 (saída correta)

```
120
```

## Exemplo 4 (programa em Capivariton)

```
     # calcula e escreve o ciclo de n
     # dat guarda o n atualizado
     # acc guarda sempre (n-1) atualizado p/ comparação do while
 0.  mov 22 dat  # define n = 22 e guarda em dat
 1.  mov dat acc # copia para acc o valor n e...
 2.  sub 1       # subtrai 1

 3.  jeq 16      # while (n-1) != 0, ou seja, while n != 1

 4.  prt dat     #   escreve o n atual
 5.  mov dat acc #   copia o n atual para acc
 6.  mod 2       #   calcula o n%2 em acc

 7.  jeq +7      #   se n%2 != 0, prossegue, senão vai p/ else
                 #   (IF: caso onde n é ímpar, vamos fazer n = n*3 + 1)
 8.  mov dat acc #     coloca o n atual em acc
 9.  mul 3       #     multiplica n por 3 (n = n*3)
10.  add 1       #     adiciona 1 ao n (n = n+1)
11.  mov acc dat #     copia novo n para dat
12.  sub 1       #     atualiza acc com n-1
13.  jmp 5       #     vai p/ final do if/else
                 #   (ELSE: caso onde n é par, vamos fazer n = n/2)
14.  mov dat acc #     coloca o n atual em acc
15.  div 2       #     divide n por 2 (n = n/2)
16.  mov acc dat #     copia novo n para dat
17.  sub 1       #     atualiza acc com n-1

18.  jmp -15     #   volta p/ a comparação do laço

19.  prt 1       # escreve último elemento do ciclo
```

## Exemplo 4 (saída correta)

```
22
11
34
17
52
26
13
40
20
10
5
16
8
4
2
1
```

# 5 Entrega

Instruções para entrega do seu trabalho:

1. **Cabeçalho**

   Seu trabalho deve ter um cabeçalho com o seguinte formato:

```
/*************************************************
 *
 * Nome do(a) estudante
 * Trabalho 2
 * Professor(a): Nome do(a) professor(a)
 *
 */
```

2. **Compilador**

   Os(as) professores(as) usam o compilador da linguagem C da coleção de compiladores GNU `gcc`, com as opções de compilação `-Wall -std=c99 -pedantic` para corrigir os programas. Se você usar algum outro compilador para desenvolver seu programa, antes de entregá-lo verifique se o seu programa tem extensão `.c`, compila sem mensagens de alerta e executa corretamente.

3. **Forma de entrega**

   A entrega será realizada diretamente na página da disciplina no AVA/UFMS. Um fórum de discussão deste trabalho já se encontra aberto. Após abrir uma sessão digitando seu *login* e sua senha, vá até o tópico "Trabalhos", e escolha "T2 - Entrega". Você pode fazer o upload de vários rascunhos, mas **o envio definitivo deve ser feito até a data indicada no AVA**. Apenas com o envio definitivo seu trabalho será corrigido. Encerrado o prazo, não serão mais aceitos trabalhos.

4. **Atrasos**

   Trabalhos atrasados não serão aceitos. Não deixe para entregar seu trabalho na última hora. Para prevenir imprevistos como queda de energia, problemas com o sistema, falha de conexão com a internet, sugerimos que a entrega do trabalho seja feita pelo menos um dia antes do prazo determinado.

5. **Erros**

   Trabalhos com erros de compilação receberão nota ZERO. Faça todos os testes necessários para garantir que seu programa está livre de erros de compilação.

6. **O que entregar?**

   Você deve entregar um único arquivo contendo APENAS o seu programa fonte com o mesmo nome de seu login no passaporte UFMS, como por exemplo, `fulano.silva.c`. NÃO entregue qualquer outro arquivo, tal como o programa executável, já compilado.

7. **Verificação dos dados de entrada**

   Não se preocupe com a verificação dos dados de entrada do seu programa. Seu programa não precisa fazer consistência dos dados de entrada. Isto significa que se, por exemplo, o seu programa pede um número entre 1 e 10 e o usuário digita um número negativo, uma letra, um cifrão, etc, o seu programa pode fazer qualquer coisa, como travar o computador ou encerrar a sua execução abruptamente com respostas erradas.

8. **Arquivo com o programa fonte**

   Seu arquivo contendo o programa fonte na linguagem C deve estar bem organizado. Um programa na linguagem C tem de ser muito bem compreendido por uma pessoa. Verifique se seu programa tem a indentação adequada, se não tem linhas muito longas, se tem variáveis com nomes significativos, entre outros. Não esqueça que um programa bem descrito e bem organizado é a chave de seu sucesso. Não esqueça da documentação de seu programa e de suas funções.

   Dê o nome do seu usuário do Passaporte UFMS para seu programa e adicione a extensão `.c` a este arquivo. Por exemplo, `fulano.silva.c` é um nome válido.

9. **Conduta Ética**

   O trabalho deve ser feito INDIVIDUALMENTE. Cada estudante tem responsabilidade sobre cópias de seu trabalho, mesmo que parciais. Não faça o trabalho em grupo e não compartilhe seu programa ou trechos de seu programa. Você pode consultar seus colegas para esclarecer dúvidas e discutir idéias sobre o trabalho, ao vivo ou no fórum de discussão da disciplina, mas NÃO copie o programa!

   Trabalhos envolvidos em plágio, mesmo que parcial, terão nota ZERO.