BCC266 – Organização de Computadores (2022-02)

Departamento de Computação - Universidade Federal de Ouro Preto - MG
Professor: **Pedro Silva** (www.decom.ufop.br/)

Trabalho Prático I (TPI) - 10 pontos, peso 3.

- Data de entrega: 16/01/2023 até 23:55. O que vale é o horário do Moodle, e não do *seu*, ou do *meu* relógio!!!

- Clareza, identação e comentários no código também vão valer pontos. Por isso, escolha cuidadosamente o nome das variáveis e torne o código o mais legível possível.

- O padrão de entrada e saída deve ser respeitado exatamente como determinado no enunciado. Parte da correção é automática, não respeitar as instruções enunciadas pode acarretar em perda de pontos.

- Durante a correção, os programas serão submetidos a vários casos de testes, com características variadas.

- A avaliação considerará o tempo de execução e o percentual de respostas corretas.

- Eventualmente serão realizadas entrevistas sobre os estudos dirigidos para complementar a avaliação;

- O trabalho é em grupo de até 3 (três) pessoas.

- Entregar um relatório.

- Os códigos fonte serão submetidos a uma ferramenta de detecção de plágios em software.

- Códigos cuja autoria não seja do aluno, com alto nível de similaridade em relação a outros trabalhos, ou que não puder ser explicado, acarretará na perda da nota.

- Códigos ou funções prontas específicas de algoritmos para solução dos problemas elencados não são aceitos

- Não serão considerados algoritmos parcialmente implementados.

- Procedimento para a entrega:.

1. Submissão: via **Moodle**.
2. Os nomes dos arquivos e das funções devem ser especificados considerando boas práticas de programação.
3. Funções auxiliares, complementares aquelas definidas, podem ser especificadas e implementadas, se necessário.
4. A solução deve ser devidamente modularizada e separar a especificação da implementação em arquivos *.h* e *.c* sempre que cabível.
5. Os arquivos a serem entregues, incluindo aquele que contém *main()*, devem ser compactados (*.zip*), sendo o arquivo resultante submetido via **Moodle**.
6. Você deve submeter os arquivos fonte e o *.pdf* (relatório) na raiz do arquivo *.zip*.
7. Caracteres como acento, cedilha e afins não devem ser utilizados para especificar nomes de arquivos ou comentários no código.

- Bom trabalho!

Máquinas Universais

Neste trabalho, o aluno entrará em contato com máquinas universais e a ideia de controle por meio de instruções. Para isto, o aluno irá construir uma máquina em qualquer linguagem de programação que interpreta instruções muito simples, tais como SOMAR, SUBTRAIR, LEVAR DADOS PARA MEMÓRIA, TRAZER DADOS DA MEMÓRIA e PARAR.

A estrutura da instrução segue o formato:

```
OPCODE | ADDRESS ONE | ADDRESS TWO | ADDRESS THREE | etc...
OPCODE | CONTENT ONE | ADDRESS ONE | CONTENT TWO | ADDRESS TWO | etc...
```

`OPCODE`: significa o que a instrução deve fazer;

`ADDRESS X`: significa o endereço onde um conteúdo reside na memória;

`CONTENT X`: significa um conteúdo armazenado na memória, seja este um inteiro, um ponto flutuante, um vetor de bytes ou qualquer outro conteúdo.

A máquina possui uma estrutura de memória simples, conforme ilustra a variável abaixo denominada RAM:

```
1  typedef struct {
2      int opcode;
3      int info1;
4      int info2;
5      int info3;
6  } Instruction;
7
8  typedef struct {
9      float *items;
10     int size;
11 } Memory;
12
13 typedef struct {
14     Instruction* instructions;
15     Memory RAM;
16 } Machine;
```

Código 1: Estrutura de memória e instruções de uma máquina hipotética.

Além da memória, a máquina também possui um programa que será interpretado. No exemplo acima, a variável `instructions` representa tal programa, ou seja, um vetor representa um programa a ser interpretado pela máquina hipotética sendo criada. Cada linha deste vetor corresponde a uma instrução no formato explicado anteriormente, portanto cada linha é um *struct* de inteiros, por exemplo.

Toda a construção da máquina e alguns programas para esta máquina são detalhados durante as aulas com o professor da disciplina. Em suma, o professor irá construir boa parte do TP1 em conjunto com a turma durante as aulas. Chamaremos a linguagem que a máquina hipotética consegue interpretar de `CAVE LANGUAGE` (possuindo pelo menos **oito** tipos de instrução).

A operação de `MULTIPLICAÇÃO` será implementada com o aluno em sala, além das operações de `SOMA`, `SUBTRAÇÃO` e operações de controle, tais como `LEVAR PARA MEMÓRIA` e `FINALIZAR MÁQUINA` que já fazem parte da linguagem. Ao todo, já estarão implementadas **cinco** instruções. Sua tarefa é implementar **pelo menos** mais **duas** instruções que utilizam as quatro instruções básicas e **pelo menos** mais **uma** que utiliza uma das instruções que você criou, ou a operação de `MULTIPLICAÇÃO`.

A Figura 1 ilustra a máquina e os diversos programas que ela poderá executar.

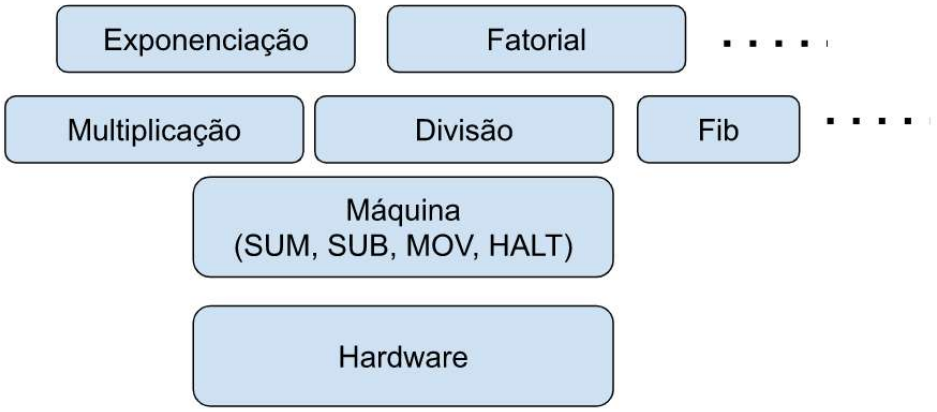

Figura 1: Máquina universal e os programas que ela pode executar.

Por fim, um exemplo em C de como seria a interpretação de uma instrução na máquina hipotética, precisamente uma instrução de SOMA:

```c
void run(Machine* machine) {
    // ...
    while(opcode != -1) {
        Instruction instruction = machine->instructions[PC];
        opcode = instruction.opcode;
        switch (opcode) {
            // ...
            case 1: // Sum
                address1 = instruction.info1;
                address2 = instruction.info2;
                RAMContent1 = machine->RAM.items[address1];
                RAMContent2 = machine->RAM.items[address2];
                result = RAMContent1 + RAMContent2;
                address3 = instruction.info3;
                machine->RAM.items[address3] = result;
                printf("Adding RAM[%d] (%f) and RAM[%d] (%f) and saving in RAM[%d] (%f)\n",
                                address1, RAMContent1, address2, RAMContent2, address3, result);
                break;
            // ...
        }
        // ...
    }
}
```

Código 2: Interpretando uma instrução de soma na máquina hipotética.

Ao final desta entrega será avaliado:

- A quantidade de programas escritos em `CAVE LANGUAGE` que fazem uso da máquina, sendo que esta apenas faz operações de `SOMA`, `SUBTRAÇÃO` e operações de controle, tais como `LEVAR PARA MEMÓRIA` e `FINALIZAR MÁQUINA`;
- A capacidade do aluno em entender e explicar os conceitos apresentados durante as aulas, tais como PIPELINE, INSTRUÇÃO, INTERPRETAÇÃO, TRADUÇÃO, UNDERFLOW, OVERFLOW, REPRESENTAÇÃO DE NÚMEROS, ARITMÉTICA COMPUTACIONAL, CICLO DE INSTRUÇÃO e outros contidos no plano de ensino da disciplina.

Imposições e comentários gerais

Neste trabalho, as seguintes regras devem ser seguidas:

- Seu programa não pode ter *memory leaks* (caso a sua linguagem tenha esse problema), ou seja, toda memória alocada pelo seu código deve ser corretamente liberada antes do final da execução. (Dica: utilize a ferramenta *valgrind* para se certificar de que seu código libera toda a memória alocada)
- Um grande número de *Warnings* ocasionará a redução na nota final.
- Implementações diferentes do que foi solicitado serão desconsideradas.

O que deve ser entregue

- Código fonte pode ser em C/C++, Java ou Python (bem identado e comentado).
- Documentação do trabalho (relatório). A documentação deve conter:
  1. **Implementação:** descrição sobre a implementação do programa. Não faça "print screens" de telas. Ao contrário, procure resumir ao máximo a documentação, fazendo referência ao que julgar mais relevante. É importante, no entanto, que seja descrito o funcionamento das principais funções e procedimentos utilizados, bem como decisões tomadas relativas aos casos e detalhes de especificação que porventura estejam omissos no enunciado. Muito importante: os códigos utilizados na implementação devem ser inseridos na documentação.
  2. **Impressões gerais**: descreva o seu processo de implementação deste trabalho. Aponte coisas que gostou bem como aquelas que o desagradou. Avalie o que o motivou, conhecimentos que adquiriu, entre outros.
  3. **Análise**: deve ser feita uma análise dos resultados obtidos com este trabalho.
  4. **Conclusão**: comentários gerais sobre o trabalho e as principais dificuldades encontradas em sua implementação.
  5. **Formato:** PDF ou HTML.

Exemplo de relatório: `https://www.overleaf.com/latex/templates/modelo-relatorio/vprmcsdgmcgd`.

Como deve ser feita a entrega

Verifique se seu programa compila e executa na linha de comando antes de efetuar a entrega. Quando o resultado for correto, entregue via *Moodle* até a 16/01/2023 até 23:55 um arquivo **.ZIP**. Esse arquivo deve conter: (i) os arquivos fonte utilizados na implementação, (ii) instruções de como compilar e executar o programa no terminal, e (iii) o relatório em **PDF** (não esqueça de colocar seu nome e do grupo no relatório pois o Moodle renomeia os arquivos enviados).

Detalhes da implementação

O código-fonte deve ser modularizado corretamente. A separação das operações em funções e procedimentos está a cargo do aluno, porém, não deve haver acúmulo de operações dentro uma mesma função/procedimento.