Universidade Federal de Ouro Preto - UFOP
Instituto de Ciências Exatas e Biológicas - ICEB
Departamento de Computação - DECOM
Ciência da Computação

TP1

BCC206 - Organização de Computadores

Gustavo Zacarias Souza
Professor: Pedro Silva

Ouro Preto
12 de janeiro de 2023

Sumário

Lista de Códigos Fonte

1 Introdução

Este código é um programa em C que simula o funcionamento de uma máquina de computação de propósito geral. Ele simula o funcionamento de um processador com sua memória RAM e as instruções de operação.

1.1 Considerações iniciais

Algumas ferramentas foram utilizadas durante a criação deste projeto:

- Ambiente de desenvolvimento do código fonte: VScode.
- Linguagem utilizada: C.
- Ambiente de desenvolvimento da documentação: Overleaf LaTeX.

1.2 Ferramentas utilizadas

Algumas ferramentas foram utilizadas para testar a implementação, como:

- *Valgrind*: ferramentas de análise dinâmica do código.

1.3 Especificações da máquina

A máquina onde o desenvolvimento e os testes foram realizados possui a seguinte configuração:

- Processador: Intel(R) Core(TM) i5-9300H CPU @ 2.40GHz 2.40 GHz.
- Memória RAM: 8,00 Gb.
- Sistema Operacional: Sistema operacional de 64 bits.

1.4 Instruções de compilação e execução

Para a compilação do projeto, basta digitar:

Compilando o projeto

```
gcc main.c -c -Wall
gcc generator.c -c -Wall
gcc cpu.c -c -Wall
gcc op.c -c -Wall
gcc main.o cpu.o generator.o op.o -o exe
rm -r *.o
./exe ...
```

Usou-se para a compilação as seguintes opções:

- *-g*: para compilar com informação de depuração e ser usado pelo Valgrind.
- *-Wall*: para mostrar todos os possível *warnings* do código.

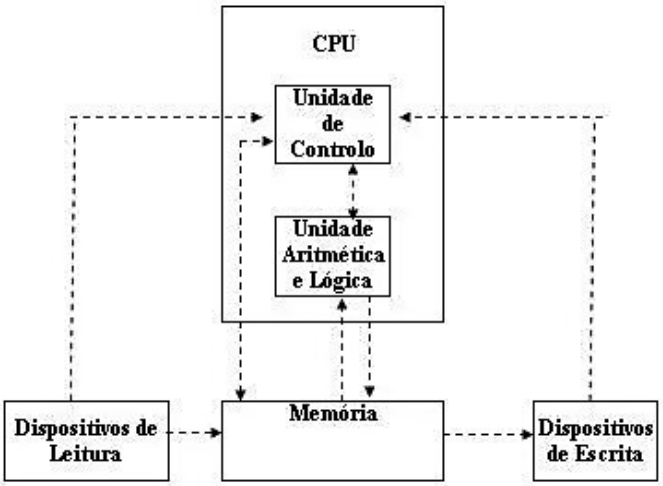
2 Desenvolvimento

Este código é um programa em C que simula o funcionamento de uma máquina de computação de propósito geral. Ele simula o funcionamento de um processador com sua memória RAM e as instruções de operação.

O principal arquivo deste programa é o main.c, que contém a função principal do programa. Esta função inicializa a máquina, lê as instruções e as executa, então imprime a memória RAM e finaliza a máquina. O gerador, contém funções para ler instruções de um arquivo de texto ou gerar instruções aleatórias. A CPU contém funções para iniciar, parar e executar a máquina. Estas funções gerenciam o processador, memória RAM e instruções, e também imprimem a memória RAM. Também existe um arquivo denominado "op"que contém funções para as operações básicas realizadas pelo processador, como soma, subtração, multiplicação, divisão, fatoração, exponenciação, sequência de Fibonacci e radicalização aproximada. O programa foi escrito para ser executado a partir da linha de comando, com três argumentos. O primeiro argumento é o tipo de instrução que será usada, podendo ser "random", "file"ou "menu". Ao ser executado, o programa inicializa a máquina e a memória RAM, gerando, lendo de um arquivo ou assumindo as instruções que o usuário passar. Em seguida, o programa executa as instruções e imprime a memória RAM. Por fim, o programa finaliza a máquina.

2.1 CPU

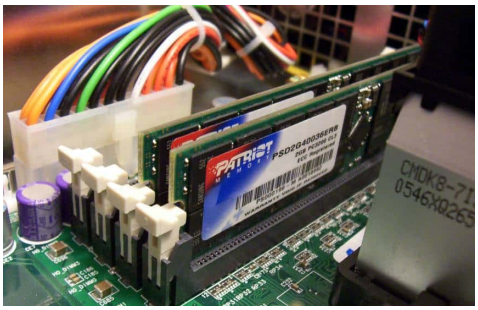
A CPU (Unidade Central de Processamento) é o "cérebro"de um computador, responsável pelas operações de processamento e lógica. Ela executa instruções de programa, processando dados, controlando o hardware e gerenciando informações.



A função start inicializa a máquina criando um vetor de instruções e uma RAM que armazena valores float. O tamanho da RAM pode ser especificado como um parâmetro na chamada da função. Além disso, essa função também inicializa a RAM com valores aleatórios entre -10 e 10. A função run executa as instruções armazenadas na RAM e atualiza os registradores conforme necessário. Os registradores são usados para armazenar o endereço de memória, o código de operação, o valor e o resultado da operação. A função printRAM imprime o conteúdo da RAM, listando o endereço de memória e o valor armazenado nele.

2.2 RAM

A memória RAM (Random Access Memory) é usada para armazenar instruções e dados temporariamente enquanto o computador está funcionando. Ela é mais rápida do que a memória de armazenamento a longo prazo, como o disco rígido, e é usada para acessar e manipular os dados que são usados atualmente. A memória RAM é limpa quando o computador é desligado, portanto, os dados não são armazenados permanentemente.



A simulação da RAM nesse código funciona armazenazenando os valores numéricos necessários para o cálculo de operações matemáticas. O tamanho da RAM é definido pelo usuário. Os valores da RAM são inicializados com valores aleatórios e podem ser alterados através da instrução 0. As instruções 1-7 acessam os valores da RAM para realizar operações matemáticas e armazenar os resultados na RAM.

2.3 Operações e Equações matemáticas utilizadas

```
1     float sum(float parcela1, float parcela2);
2     float sub(float minuendo, float subtraeno);
3     float mult(float fator1, float fator2);
4     float divi(float dividendo, float divisor);
5     float fact(float fator);
6     float expo(float base, float expoente);
7     float fibonacci(int elemento);
8     float raiz(float valor);
```

Código 1: Protótipos das funções matemáticas utilizadas

A imagem a seguir, apresenta a fórmula da sequência de Fibonacci.

$$F(n) = \frac{\sum_{p=0}^{(n-2)/2} \binom{n}{2p+1} 5^p}{2^{n-1}}$$

A imagem a seguir, apresenta a fórmula da combinação fatorial.

$$C_{p,n} = \frac{n!}{p!(n-p)!}$$

2.4 Incluindo fragmento de códigos

```
        machine->instructions = instructions;
        machine->RAM.items = (float*) malloc(sizeof(float) * RAMSize);
        machine->RAM.size = RAMSize;
        for (int i=0;i<RAMSize;i++){
            int valores;
            machine->RAM.items[i] = (valores = (int) (rand() % 21 - 10));
        }
```

Código 2: Exemplo de randomização de -10 a 10

```
  for(int i=2;;i++){
    raizaux=divi(valor,i);

    if(expo(raizaux,2)==valor)
      return raizaux;
    else if(expo(raizaux,2)<valor){
        for(int i=raizaux; i<=valormax; i++){
          result= expo(i,2);
          if(result>=valor){
            valorprox=valorprox-valor;
            result=result-valor;
            result = (result<0)? -result : result;
            valorprox = (valorprox<0)? -valorprox : valorprox;
            if(valorprox < result)
              return i-1;
            return i;
          }
          valorprox=result;
        }
    }
    else
      valormax=raizaux;
  }
}
```

Código 3: Após umas pré-verificação, o valor entra no for apresentado, que retornará uma raiz quadrada aproximada ou exata.

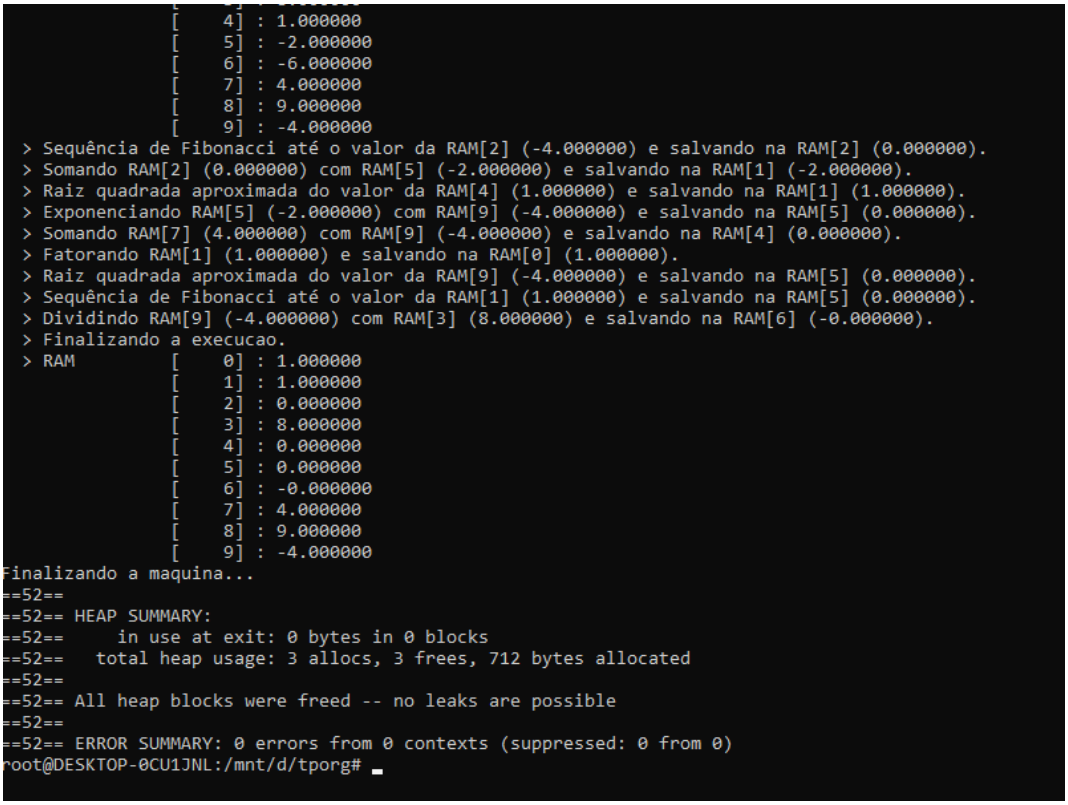
```
  float result = 0;
    int neg=0;

    if ((fator1<1 && fator1>=0)||(fator2<1 && fator2>=0)||(fator1>-1 && fator1
        <=0)||(fator2>-1 && fator2<=0))
        return 0;

    if ((fator1<0 && fator2>0) || (fator1>0 && fator2<0))
        neg=1;

    fator1 = (fator1<0)? -fator1 : fator1;
    fator2 = (fator2<0)? -fator2 : fator2;

    for (int i = 0; i < fator2; i++)
        result = sum(result, fator1);

    if(neg)
      result *= -1;

    return result;
}
```

Código 4: Função de multiplicação, usando for, válida para números positivos e negativos

3 Experimetos

Os testes feitos foram os de verificação de memory leaks e de memória alocada, tentei fazer um teste de tempo de execução, mas não consegui.

4 Resultado



Tentei efetuar uma medição usando a biblioteca time.h, mas o código executa tão rápido (ou fiz de modo errado), que o tempo continuava no 0.00000, então não foi possível verificar com precisão o tempo de execução. Porém, o teste de memory leaks fluiu muito bem, o programa está livre de memory leaks. Também fiz incessantes testes dos resultados de saída do programa, tentei precaver o programa de todas as formas possíveis de possibilidades de erro.

5 Considerações Finais

Gostei muito do trabalho, mesmo demandando um bom tempo, achei o trabalho bem interessante, não se trata somente de programar, a parte mais custosa é a lógica por trás de cada operação matemática. Contudo, foi possível entender o quão rápido um processador efetua uma quantidade massiva de comandos, além de, mostrar o quão importante é a memória RAM.