

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

Trabalho Prático 1

BCC266 - Organização de Computadores I

Felipe Braz Marques
Matheus Peixoto Ribeiro Vieira
Pedro Henrique Rabelo Leão de Oliveira
Professor: Pedro Henrique Lopes Silva

Ouro Preto
4 de janeiro de 2023

Sumário

1	Introdução	1
1.1	Especificações do problema	1
1.2	Considerações iniciais	1
1.3	Ferramentas utilizadas	1
1.4	Especificações da máquina	1
1.5	Instruções de compilação e execução	1
2	Desenvolvimento	2
2.1	generator.h	2
2.2	generator.c	2
2.3	main.c	4
3	Impressões Gerais e Considerações Finais	5

Lista de Figuras

1	Exemplo de dois elevado a quatro	3
---	--	---

1 Introdução

Para este trabalho é necessário entregar um código em C para simular o funcionamento de um computador, e como a memória principal funciona.

1.1 Especificações do problema

Utilizando a linguagem de programação C, é proposto construir uma máquina que funciona por meio de instruções, como soma, subtração, levar dados para memória e halt, que são as básicas, e a multiplicação, divisão, fatorial e exponencial, que são realizadas a partir das básicas. As instruções possuem uma estrutura, com código para identificar do que se trata, e possuem informações para que seja possível a realização das mesmas. E, tudo acontece por meio dos dados gerados e salvos na memória RAM da máquina.

1.2 Considerações iniciais

Algumas ferramentas foram utilizadas durante a criação deste projeto:

- Ambiente de desenvolvimento do código fonte: Visual Studio Code.
- Linguagem utilizada: C.
- Ambiente de desenvolvimento da documentação: Overleaf L^AT_EX.¹

1.3 Ferramentas utilizadas

Algumas ferramentas foram utilizadas para testar a implementação, como:

- GCC: versão 11.3.0.
- Valgrind: versão 3.18.1.

1.4 Especificações da máquina

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

- Processador: Intel i5-9300H.
- Memória RAM: 16Gb.
- Sistema Operacional: Ubuntu 22.04.1 LTS.

1.5 Instruções de compilação e execução

Para a compilação do projeto, basta digitar:

```
Compilando o projeto
```

```
gcc -Wall cpu.c generator.c main.c -o exe
```

Usou-se para a compilação as seguintes opções:

- *-Wall*: para mostrar todos os possível *warnings* do código.

Para a execução do programa basta digitar:

```
./exe instrucao-desejada tam-RAM
```

¹Disponível em <https://www.overleaf.com/>

2 Desenvolvimento

2.1 generator.h

Neste arquivo foram adicionados os protótipos para as funções `generateMultiplicationInstructions`, recebendo quatro inteiros, a `generateDivisionInstructions` e a `generateExponentiationInstructions`, recebendo dois inteiros como parâmetros, a `generateFactorialInstructions` recebendo um inteiro e, por fim, a `generateArithmeticProgressionInstructions` recebendo três valores inteiros.

2.2 generator.c

Dentro do arquivo `generator.c` foi desenvolvido o corpo das funções prototipadas no `generator.h`

A função **"generateMultiplicationInstructions"** recebe quatro elementos como parâmetros, sendo eles o `n1` e `n2` para executar a multiplicação, e dois valores de controle, para verificar se o programa deverá gerar a instrução de `Halt`, a partir da variável `execHalt`, e se os valores serão salvos na RAM, com a variável `salvarValorNaRam`.

Primeiramente, verificamos se haverá instruções extras, como o `halt` e salvar os valores na ram. Depois é gerado um vetor `instrucoes` do tipo `Instruction`, com o tamanho do `n2`, acrescido da quantidade de instruções extras, que serão 3 caso o programa execute o `Halt` caso contrário será dois caso vá salvar na ram, ou zero, caso nenhum dos parametros sejam verdadeiros.

Caso salvar na RAM seja verdadeiro, são geradas instruções para levar o `n1` para a RAM na posição 0 e outra para levar o valor 0 para a posição 1, pois como a multiplicação é uma sequência de somas, o termo neutro da adição é o zero.

Logo após, é verificado qual será a instrução inicial. Se o `salvarNaRam` for verdadeiro, a instrução inicial será a 2, caso contrário será a 0. Após isso é executado um `for` que vai do valor da instrução inicial até `n2` mais o valor de instrução inicial gerando somas da posição 0 da RAM com a 1 e salvando novamente na 1.

Por fim, caso o `halt` venha a ser executado, é gerada a instrução para tal e o vetor de instruções é retornado, encerrando, assim, a função.

A função **"generateExponentiationInstructions"** recebe como parâmetro dois inteiros, um sendo a base e o outro o expoente. O expoente será utilizado no número de multiplicações com um valor a menos, pois, tomando como exemplo 2^3 , são feitas duas multiplicações, $2 \times 2 = 4$ e $4 \times 2 = 8$. Dessa forma, o `for` para definir a quantidade de instruções irá variar de 0 até expoente-1, sempre acrescentando à variável `qtdInstrucoes` o valor de base + 3, tendo em vista que, a cada chamada da função `generateMultiplicationInstructions`, são feitas base's somas para ter o resultado da multiplicação e o seu resultado deve ser enviado para a posição 0 da RAM, pois ele estará salvo na posição 1. Para isso, então, é colocado o valor zero na posição zero, depois é somado o valor da posição 1, e, por fim, é colocado o valor zero na posição 1. Essa troca segue exemplificada na figura 1, que representa 2^4 .

Assim, após o `for`, é adicionado mais três à variável inteira `qtdInstrucoes` para que sejam incluídos o `halt` e o movimento de levar os valores iniciais para a RAM. E, logo em seguida, é iniciado o vetor `instrucoes` do tipo `Instruction` do tamanho `qtdInstrucoes * sizeof(Instruction)`.

No vetor, a primeira instrução é para levar até a posição 0 da RAM o valor da base caso o expoente seja diferente de zero, ou 1, caso o expoente seja zero. Depois é levado o valor zero para a posição 1 da RAM, pois serão feitas multiplicações.

Em seguida, é iniciado um `for` que vai de 2 até `qtdInstrucoes-1`, onde no início é sempre feita uma chamada para a `generateMultiplicationInstructions`, passando a base como `n1` e `n2`, e `execHalt` e `salvarNaRam` como 0, pois esses valores já estão salvos e ainda há mais instruções para serem feitas antes de encerrar o programa.

O retorno da função de multiplicação é salvo em um vetor auxiliar chamado de `instrucoesTemp`, que é, então, adicionado ao vetor de instruções a partir de um `for` que vai de 0 até base, pois são feitas base's multiplicações. Assim, o vetor temporário que recebe o retorno é liberado e são geradas mais três instruções para a troca dos valores da posição 1 e 0 da RAM, como já foi explicado acima.

0	1	2	3
2	0		
2	2		
2	4		
4	0		
4	4		
4	8		
8	0		
8	8		
8	16		

Figura 1: Exemplo de dois elevado a quatro

O valor i do `for` é incrementado em mais três + base, pois é a quantidade de itens retornados da multiplicação.

Por fim, é executado o `halt` e retornado o vetor de instruções, finalizando, então a função.

A função **"generateDivisionInstructions"** recebe como parâmetro um dividendo e um divisor ($n1$ e $n2$ respectivamente para os comentários), e a lógica para operação de divisão se baseia em uma sequência de instruções de subtração.

Inicialmente, é alocado um vetor de instruções com o tamanho `qtdInstrucoes*sizeof(Instruction)`, que será o necessário para as primeiras 4 instruções. Primeiro, o dividendo é levado para a RAM no endereço 0, e o endereço 1 recebe o divisor. Em seguida, o valor 0 é atribuído para a posição 2 da RAM, que será onde ficará armazenado o quociente da divisão, e o valor 1 para a posição 3 da RAM, para que seja feita a soma de mais um ao quociente a cada subtração feita posteriormente.

Em seguida, é executado um `for` dentro dessa função que se repetirá o número de vezes que o divisor cabe no dividendo. Dentro desse `for`, a cada vez que ele se repete, é aumentado +2 na `qtdInstrucoes` e o vetor de instruções é realocado. Depois disso, é feita a operação de subtração do valor armazenado na posição 0 (dividendo) menos o valor armazenado na posição 1 (divisor), armazenando o resultado na própria posição 0. E é feito também uma operação de soma com o valor armazenado na posição 2 (quociente) mais o valor armazenado na posição 3 (um), pois, dessa forma, a repetição do `for`, o quociente será aumentado em 1, e posteriormente, teremos o quociente final.

Finalizado a execução do `for`, o vetor de instruções é realocado com o tamanho de `(qtdInstrucoes+1)* sizeof(Instruction)` para que seja incluída a instrução de `halt` para máquina. E, assim, é retornado o vetor de instruções feito para que seja executado o cálculo de um fatorial a partir das 4 funções básicas da máquina.

A função **"generateFactorialInstructions"** recebe um número inteiro dado para o fatorial como parâmetro, sendo chamado de $n1$.

De início, é instanciado um vetor de instruções e alocado dinamicamente com o tamanho necessário para 4 instruções iniciais. Instruções essas que realizam as seguintes funções: primeiro é levado o valor de $n1$ para a posição 0 da RAM e 0 para a posição 1 da RAM, estes que serão utilizados posteriormente nas chamadas da função `generateMultiplicationInstructions`. Depois, é salvo $n1-1$ na posição 2 da RAM, posição esta que servirá para representar os valores que estão sendo multiplicados no fatorial, e 1 na posição 3 da RAM, pois, dessa forma, a cada multiplicação que for feita, será feita, a partir de uma instrução, a subtração do valor que está salvo na posição 2 menos o valor que está salvo na posição 3, e o resultado é armazenado novamente na posição 2.

Depois disso, é instanciado um vetor temporário de instruções. Dentro de um for que se repetirá $n1-2$ vezes, a quantidade de instruções é aumentada $i+4$, e com isso o vetor de instruções é realocado. O vetor de instruções temporárias recebe o vetor retornado da função `generateMultiplicationInstructions`, e é feito um outro for para passar as instruções para o vetor principal. Após isso, são feitas 3 instruções, duas de levar um valor para certa posição da memória e uma de soma, com intuito de passar o resultado da multiplicação feita anteriormente para posição 0 da RAM e deixar a posição 1 com o valor 0 armazenado nela, dessa forma, as chamadas seguintes da função `generateMultiplicationInstructions` funcionará normalmente. E, por último, no for, tem a instrução de subtração, citada anteriormente, que subtrai o valor da posição 2 menos 1 (valor da posição 3).

Finalizado o loop do for, o vetor de instruções é realocado com o tamanho de $(qtdInstrucoes+1) * sizeof(Instruction)$ para que seja incluída a instrução de halt para máquina. E, assim, é retornado o vetor de instruções feito para que seja executado o cálculo de um fatorial a partir das 4 funções básicas da máquina.

A função **"generateAritmeticProgressionInstructions"** irá realizar a operação de progressão aritmética. Para isso, receberá três parâmetros inteiros, sendo eles o primeiro termo da progressão (a_1), a quantidade de termos (n) e a razão em que ocorre a progressão ($razao$). Sendo essas as variáveis da fórmula $a_n = (n - 1) * razao + a_1$.

É instanciado, então, um vetor do tipo instruções de tamanho sete (instruções padrões) mais o valor da razão (indica a quantidade de multiplicações que serão realizadas).

Realizando primeiramente $n - 1$, o valor de n é levado para posição 0 da RAM e o valor 1 é levado para a posição 1. Logo em seguida é realizada a instrução de subtração, salvando o resultado na posição 0. Depois, o valor 0 é levado para a posição 1, pois será realizada a multiplicação, fazendo necessário com que tenha um valor neutro para as somas que ocorrerão, pois elas são a base da multiplicação.

Um vetor temp do tipo `Instruction` é instanciado recebendo as instruções de multiplicação pela função `generateMultiplicationInstructions`, que tem os seus parâmetros n_1 sendo 0 (Esse valor não é importante pelo fato de já está salvo na RAM), n_2 sendo a *razao* (Valor responsável para saber a quantidade de somas que serão feitas), `execHalt` e `salvarValorNaRam` como 0.

Em seguida é realizado um loop de 0 a $razao-1$ com incremento de 1 para adicionar as instrucoes de multiplicação no vetor de instruções. Em seguida o vetor temp é liberado.

Por fim ocorre a soma dos dois termos para finalizar a operação. Então, a_1 é levado para a posição 0 da RAM, para que ocorrerá uma soma com o resultado da multiplicação que está salvo na posição 1, escrevendo a conclusão da progressão na posição 0 da RAM. Assim, a instrução de halt será feita e o vetor `instrucoes` será retornado, finalizando, então, a função.

2.3 main.c

No arquivo `main.c` foi adicionado quatro "else if" para verificar o tipo de instrução que será executado, sendo eles "multiply", "division", "expoente", "factorial" e "ap", passando os seus respectivos valores para as operações.

3 Impressões Gerais e Considerações Finais

Realizando o trabalho, foi possível entender mais sobre o funcionamento do processador, fazendo as aulas mais teóricas da disciplina se tornarem mais práticas e mais visíveis.

Uma das dificuldades que encontramos foi não poder utilizar os valores salvos na RAM para realizar as operações matemáticas, pois era necessário gerar as instruções antes, o que gerou muitas dúvidas sobre como seria feita a divisão, pois, por se tratar de subtrações, era necessário saber previamente quantas subtrações seriam feitas, o que já nos deixaria com o resultado, o que nos levaria a uma posição contrária à proposta do projeto.

Todavia conseguimos contornar este problema com um for que é incrementado em dividendos quantidades, seguindo, assim, a proposta do projeto.