CIÊNCIA DA COMPUTAÇÃO

Emulador de ISA na linguagem de programação C

Victor Emmanuel Susko Guimarães
Cristiano Augusto Dias Mafuz

I. Introdução

Os computadores modernos seguem um padrão de arquitetura criado por John von Neumann, importante matemático e revolucionário cientista na área da computação, que tornou possível os computadores serem programados para executarem diferentes funções. Tal arquitetura é constituída pelos seguintes componentes: Unidade central de processamento (CPU), Memória, sistema de entrada e saída, Unidade Lógica e Aritmética (ALU) e registradores. A CPU é responsável por buscar instruções na memória, decodificá-las, executá-las e armazenar os resultados. A memória é responsável por guardar os dados e as instruções do programa, o sistema de entrada e saída é responsável por receber os dados dos usuários ou de outros dispositivos e mostrar os resultados do programa, a ALU realiza operações de cálculo lógico e numérico e, por fim, os registradores são locais de armazenamento de dados, podendo estes armazenar valores, endereços de memória ou códigos de instrução. A figura 1 representa um esquema contendo os componentes do processador segundo a arquitetura de Neumann.

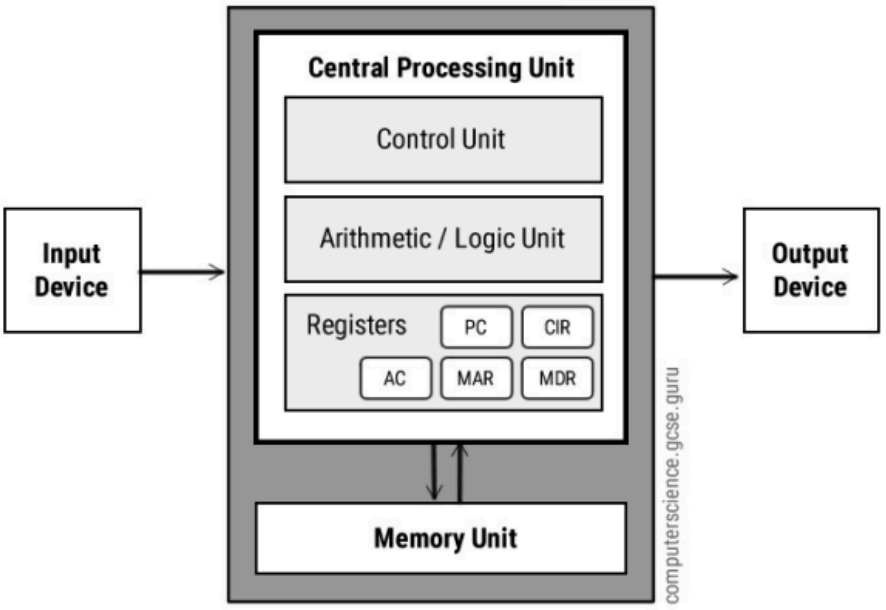

Figura 1 - arquitetura de Neumann

Nesse contexto, é importante citar que a ISA (Instruction Set Architecture) de um processador é o documento que define o conjunto de instruções que ele pode executar, o tamanho das instruções, o número de registradores, o tipo das instruções, como essas instruções são organizadas na memória e como o processador lê e executa as instruções.

Desse modo, o objetivo deste trabalho é construir um algoritmo em C que consiga emular o comportamento de um processador, com uma ISA cujas instruções e funcionamento são definidos pelos membros do grupo. Além disso, o emulador deve ser capaz de reproduzir dois algoritmos a partir de um arquivo de texto: o primeiro algoritmo deve identificar os números primos de 1 a 100, armazená-los na memória principal e imprimí-los, enquanto o segundo deve encontrar o seno e o cosseno de um ângulo dado em radianos.

Das disposições do trabalho, a seção II possui o referencial teórico usado na solução dos dois algoritmos, a seção III aborda todos os procedimentos da construção do emulador e as métricas avaliativas dos testes, a seção IV traz os resultados dos testes, bem como gráficos que avaliam dos resultados dos algoritmos, a seção V mostra uma síntese geral de todo o conteúdo apresentado, juntamente com uma análise crítica dos resultados e, por fim, a seção VI mostra um breve manual de como utilizar o emulador.

II. Referencial Teórico

Para o desenvolvimento dos algoritmos em questão, primeiramente é fundamental destacar as 16 funções básicas do emulador:

ADD #R1 #R2 - realiza a soma dos valores de dois registradores #R1 e #R2, armazenando o valor resultante no registrador de saída ACC

SUB #R1 #R2 - realiza a subtração dos valores de dois registradores #R1 e #R2, armazenando o valor resultante em ACC

MUL #R1 #R2 - realiza o produto dos valores de dois registradores #R1 e #R2, armazenando o valor resultante em ACC

DIV #R1 #R2 - realiza o quociente dos valores de dois registradores #R1 e #R2, armazenando o valor resultante em ACC

REM #R1 #R2 - realiza o resto da divisão dos valores de dois registradores #R1 e #R2, armazenando o valor resultante em ACC

ADDI #R1 CONST - realiza a adição de uma constante no valor de um registrador #R1, armazenando o valor resultante no próprio registrador e no ACC

SUBI #R1 CONST - realiza a subtração de uma constante no valor de um registrador #R1, armazenando o valor resultante no próprio registrador e no ACC

MULI #R1 CONST - realiza o produto de uma constante no valor de um registrador #R1, armazenando o valor resultante no próprio registrador e no ACC

STO #R1 - armazena o valor de ACC no registrador #R1

STI #R1 - atribui o valor de uma constante no registrador #R1

DSP - imprime o valor de ACC

STOM - armazena o valor de ACC na memória principal, tal processo é feito de maneira sequencial

DSPM - imprime todos os valores armazenados na memória principal

J Linha - salto incondicional para a linha

BEQ #R1 #R2 Linha - compara o valor do registrador #R1 com o registrador #R2, se o valor do primeiro registrador é igual ao do segundo, salta para a linha definida

BLT #R1 #R2 Linha - compara o valor do registrador #R1 com o registrador #R2, se o valor do primeiro registrador é menor que o do segundo, salta para a linha definida

Para o desenvolvimento do arquivo de texto dos números primos, o seguinte raciocínio foi utilizado:

Sabe-se que, pelo algoritmo intitulado “Crivo de Eratóstenes”, para verificar se um número é primo, é preciso fazer a divisão pelos números apenas até a raiz quadrada do limite estipulado, que no caso, é igual a 100 (pois são todos os números primos de 0 até 100), portanto, é necessário apenas fazer a divisão até raiz quadrada de 100, que é 10, no caso fazer as divisões pelos números primos nesse intervalo, pois os outros não são úteis.

Para exemplificar, toma-se como exemplo, o número 29:

Como já dito, o intervalo é 100, então é necessário dividir 29 por todos os primos de 0 a raiz de 100, sendo então:

29 / 2 = (não é inteiro)
29 / 3 = (não é inteiro)
29 / 5 = (não é inteiro)
29 / 7 = (não é inteiro)

Desse modo, já é possível concluir que 29 é primo, pois qualquer divisor maior que esse, vai resultar em um quociente com parte inteira menor que o próprio divisor, o que é incoerente, pois caso fosse divisível, já se revelaria em um desses resultados acima.

Assim, o seguinte texto foi produzido:

```
1 STI #0 0
2 STI #1 1
3 STI #2 0
4 STI #3 1
5 STI #4 100
6 ADDI #2 1
7 DIV #4 #2
8 STO #5
9 BLT #2 #5 6
10 ADDI #3 1
11 BEQ #3 #4 22
12 STI #2 1
13 ADDI #2 1
14 BEQ #3 #2 19
15 REM #3 #2
16 STO #6
17 BEQ #6 #0 10
18 BLT #2 #5 13
19 ADDI #3 0
20 STOM
21 J 10
22 DSPM
```

Na linha 4, há o limite estipulado (100), porém poderia ser feito com qualquer outro número como limite.

Da linha 6 à linha 9, é apenas para descobrir-se a raiz aproximada desse limite. Daí, com o auxílio de um contador, tem-se o quociente entre o limite e o valor atual do contador, que é comparado com ele próprio, a fim de, como dito, chegar em um valor igual aproximado da raiz quadrada do limite (registrador 5).

Da linha 10 até o final do texto, é o cálculo dos números primos, que são feitos com o uso de dois contadores, um que vai de número a número até o limite (registrador 3), e outro que, reseta sempre que esse registrador citado muda para um novo valor (registrador 2), e serve de divisor até se igualar ao valor do registrador 5 , que já foi explicado acima.

Ao final de cada iteração, se for primo, é armazenado na memória principal pela função “STOM”, e ao final da execução, impresso na tela pela função “DSPM”.

Já para o desenvolvimento do arquivo de texto do cálculo de seno e cosseno de um ângulo em radianos, a série de MacLaurin foi implementada. Vale ressaltar que existem várias formas diferentes de calcular o seno e o cosseno de um ângulo, como a série de Fourier, a fórmula de Euler, transformações trigonométricas, a relação dos catetos com a hipotenusa, etc. Entretanto, alguns métodos requerem o conhecimento prévio de outros senos ou cossenos de ângulos que também não se sabe quais são, como as transformações trigonométricas, outros são mais complexos de serem implementados por levarem em consideração números imaginários nos cálculos, como a fórmula de Euler. Por isso, a resolução escolhida foi a série de MacLaurin, pois o somatório da série pode ser encontrado sem a necessidade da implementação dos números imaginários e do conhecimento de outros senos. Apesar dos pontos positivos, este método também possui a desvantagem de que não é possível encontrar precisões maiores do que 6 casas decimais porque o limite de memória do emulador é atingido.

Desse modo, será demonstrado como chega-se ao somatório que fornece o seno e o cosseno de um ângulo.

A série de Taylor de uma função f(x) em torno de um ponto a é dada pela fórmula:

$$f(x) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \frac{f'''(a)}{3!}(x-a)^3 + \dots$$

A série de MacLaurin é o caso da série de Taylor em que o ponto de expansão, representado pela variável ***a*** é igual à zero. Assim, substituindo o valor de ***a*** por 0 na fórmula, tem-se que:

$$f(x) = f(0) + f'(0)(x) + \frac{f''(0)}{2!}(x)^2 + \frac{f'''(0)}{3!}(x)^3 + \dots$$

Agora, atribuindo a f(x) a função sen(x), é possível calcular as derivadas das funções de seno e encontrar seus valores quando x = 0. Portanto, tem-se que:

$$f(x) = sen(x)$$
$$f'(x) = cos(x)$$
$$f''(x) = -sen(x)$$
$$f'''(x) = -cos(x)$$
$$f''''(x) = sen(x)$$

Desse modo, as derivadas repetem este padrão infinitamente. Em seguida, substituindo 0 no valor de x, tem-se que:

$$f(0) = sen(0) = 0$$
$$f'(0) = cos(0) = 1$$
$$f''(0) = -sen(0) = 0$$
$$f'''(0) = -cos(0) = -1$$
$$f''''(0) = sen(0) = 0$$

Substituindo estes valores na série, tem-se:

$$sen(x) = 1 * x + \frac{0}{2!}(x)^2 + \frac{-1}{3!}(x)^3 + \frac{0}{4!}(x)^4 + \frac{1}{5!}(x)^5 + \ldots$$

É possível notar que todas as n-ésimas derivadas, tal que n é par, são excluídas do cálculo pois a soma parcial delas na série resultará em zero. Logo, simplificando, chega-se ao seguinte resultado:

$$sen(x) = x - \frac{(x)^3}{3!} + \frac{(x)^5}{5!} - \frac{(x)^7}{7!} + \ldots$$

Este cálculo pode ser expandido infinitas vezes, sendo que quanto mais parcelas da soma são calculadas, mais próximo do valor do seno é o resultado obtido.

De forma similar, a série de Taylor para o cosseno pode ser calculada da seguinte forma:

$$f(x) = cos(x)$$
$$f'(x) = -sen(x)$$
$$f''(x) = -cos(x)$$
$$f'''(x) = sen(x)$$
$$f''''(x) = cos(x)$$

Substituindo 0 no valor de x, tem-se que:

$$f(0) = 1$$
$$f'(0) = 0$$
$$f''(0) = -1$$
$$f'''(0) = 0$$
$$f''''(0) = 1$$

Substituindo estes valores na série, tem-se:

$$cos(x) = 1x^0 + \frac{-1}{2!}x^2 + \frac{0}{3!}x^3 + \frac{1}{4!}x^4 + \frac{0}{5!}(x)^5 + \ldots$$

Dessa vez, todas as n-ésimas derivadas, tal que n é ímpar, são excluídas do cálculo pois a soma parcial delas na série resultará em zero. Logo, simplificando, chega-se ao seguinte resultado:

$$cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} + \frac{x^6}{6!} + \ldots$$

Uma vez demonstradas as duas séries, ambas foram implementadas no arquivo de texto com os comandos mostrados anteriormente. Desse modo, o cálculo do seno e do cosseno de um ângulo dado em radianos é calculado pelo emulador com os seguintes comandos:

```
1 STI #0 0
2 STI #11 1
3 STI #2 1
4 STI #1 0
5 STI #3 1.57
6 STI #8 6.283185307179586
7 STI #9 80
8 STI #13 2
9 STI #15 0
10
11 REM #3 #8
12 STO #10
13 MUL #10 #8
14 STO #10
```

```
15 SUB #3 #10
16 STO #3
17
18 J 35
19 J 44
20 DIV #5 #7
21 STO #12
22 REM #1 #13
23 STO #14
24 BEQ #14 #0 28
25 SUB #15 #12
26 STO #15
27 J 30
28 ADD #15 #12
29 STO #15
30 ADDI #1 1
31 ADDI #2 2
32 BEQ #1 #9 54
33 J 18
34
35 STI #4 0
36 STI #5 1
37 BEQ #4 #2 19
38 MUL #3 #5
39
40 STO #5
41 ADDI #4 1
42 J 37
43
44 SUB #2 #0
45 STO #6
46 STI #7 1
47 BEQ #6 #0 20
48 MUL #6 #7
49
50 STO #7
51 SUBI #6 1
52 J 47
53
54 ADDI #15 0
55 DSP
56 BEQ #0 #11 60
57 SUBI #11 1
58 STI #2 0
```

59 J 4
60

O algoritmo começa fazendo o cálculo de correspondência do ângulo desejado no círculo trigonométrico (linhas 11 a 16), representado pelo registrador 3, na linha 5. O resultado desse cálculo revela qual é o ângulo estabelecido entre 0 e 2π que corresponde ao ângulo escolhido. Em outras palavras, caso o ângulo escolhido já esteja nesse intervalo, ele permanece o mesmo, entretanto, se o ângulo é maior que 2π, o cálculo mostrará qual ângulo é o equivalente dentro do intervalo mencionado. O intuito do cálculo é fazer com que mais possibilidades de ângulos possam ser calculadas, uma vez que a série de MacLaurin possui termos em fatorial, que crescem de maneira exponencial com poucos valores de diferença. Caso o cálculo de correspondência não fosse aplicado, ângulos maiores que 15 radianos não poderiam ser encontrados, pois os termos em fatorial dessas somas teriam ultrapassado o limite do emulador. Assim, o cálculo consiste em encontrar o resto da divisão entre o ângulo pela constante 6.283185307179586, que equivale a aproximadamente 2π, e multiplicar este resto pela constante novamente. O resultado encontrado é o ângulo correspondente.

No exemplo acima, o ângulo é de 1,57 radianos (aproximadamente $\pi/2 = 90^{o}$), todavia poderia ser atribuído a este registrador qualquer ângulo o qual deseja-se saber o seno e o cosseno.

Já na linha 18 (começo da "função main"), é chamado o bloco de texto onde está localizado a "função exponencial" (linha 35 a 42), que funciona com o uso de dois contadores (registradores 2 e 4), um estático e o outro dinâmico, o estático (registrador 2), dita o limite de iterações do registrador dinâmico (4), e, a cada uma dessas iterações, o registrador 5, que começa com o valor 1, vai se multiplicando pelo valor do ângulo estipulado, localizado no registrador 3 (o nosso "x"), e sendo armazenado no próprio registrador 5.

Ao finalizar esse cálculo, é feito o retorno para a função "main", e a linha 19 usa um "jump" novamente, dessa vez redirecionando para a "função fatorial" (linha 44 a 52), que funciona, novamente, com o auxílio de um contador, que recebe o valor do registrador 2 manipulando o ACC (registrador 6), e fazendo com que, a cada iteração esse registrador subtraia em uma unidade, e antes disso, aplicando um método similar ao usado na "função exponencial", inicializando o registrador 7 com 1, e multiplicando pelo registrador 6 e armazenando no mesmo.

Então, voltando para a função "main", temos em mãos o valor exponencial e o fatorial daquela parcela em questão (registradores 5 e 7 respectivamente), a seguir, o quociente da divisão desses fatores é armazenado no registrador 12, e, logo em seguida é feita uma verificação para averiguar se o registrador 1 é par ou ímpar, caso seja par, a operação é de soma, caso contrário, de subtração, e ao final armazenado no registrador 15, o registrador 1 e 2 são acrescidos em 1 e 2 unidades respectivamente, e todo o processo (partindo da linha 18) é feito novamente, até se chegar no limite estipulado pelo registrador 9.

Ao chegar nesse valor, é impresso na tela o valor guardado pelo registrador 15, porém a execução só termina se ambas operações de cálculo de seno e cosseno forem feitas, caso tenha sido apenas o do seno, é verificado se o valor guardado pelo registrador 0 (guarda apenas o valor 0, sem qualquer alteração), é idêntico ao valor guardado pelo registrador 11, registrador esse que guarda valor 1 por toda a execução para encontrar o valor do cosseno, porém ao final, como essa verificação não é verdade e o jump não vai ocorrer, é subtraído de uma unidade, o registrador 2 na execução da função cosseno começa dessa vez de 0 ao invés de 1 (essa é a única diferença entre ela e a função seno), e é feito um "jump" para a linha 4, sendo assim, todo o cálculo é feito novamente e, dessa vez o jump da função BEQ na linha 56 é executado, encerrando a execução do algoritmo.

III. Metodologia

Para a construção do emulador, foram utilizados 7 arquivos: **isa.c, isa.h, alu.c, alu.h, memoria.c, memoria.h e main.c**, cujas funções e conteúdos são:

isa.c - é onde estão definidas as funções de alocação do banco de registradores e as funções de identificação da operação que a ALU realizará.

isa.h - possui os protótipos das funções do arquivo isa.c e as seguintes estruturas: *Instrução, Alu, Main_mem, Inst_mem, Registrador e Banco_reg*. Cada uma das estruturas representa o comportamento do componente com o nome correspondente, desse modo, a estrutura *Instrução* é utilizada para limitar o tamanho das instruções em até 32 bits, com o auxílio das bibliotecas <stdint.h> e <inttypes.h>, que permitem estipular limites às variáveis inteiras. Dentro dessa estrutura são esperados dois registradores de tamanho limitado a 4 bits, que são utilizados nas funções do emulador.

A estrutura *Alu* representa a ALU e contém o opcode, que indica qual função o emulador deve executar segundo uma determinada instrução. Esse opcode é uma variável do tipo inteiro que está restringida a 4 bits.

Já a estrutura *Main_mem* representa a memória principal do processador, e nela existe um ponteiro, que será alocado dinamicamente durante a execução do emulador para que contenha exatamente 16MB. Além disso, essa estrutura também possui uma variável inteira que armazenará o endereço de cada índice do vetor, para que ele seja preenchido de maneira sequencial.

Inst_mem é a estrutura que representa a memória de instruções do processador, e possui um vetor de strings, que armazenará todo o arquivo de texto que contém as instruções do processador, assim, é a partir dessa memória que o algoritmo definido no arquivo é executado. Além disso, essa estrutura também apresenta um inteiro que indica a linha para a qual o processador deve direcionar o fluxo da execução do algoritmo.

Por último, *Registrador* é a estrutura de dados que corresponde a um único registrador, contendo dentro dela uma variável do tipo **double**, que consegue carregar casas

decimais, e *Banco_reg* é a estrutura que um ponteiro, que será alocado para conter 16 registradores mais o registrador especial ACC, que é também chamado de registrador de saída. Este registrador é responsável por armazenar os resultados de algumas operações algébricas, e também pode ser utilizado para imprimir valores na tela do usuário, como mencionado na seção II deste relatório.

alu.c - nesse arquivo estão contidas todas as funções que podem ser desempenhadas pelo emulador, com ênfase nas operações algébricas da ALU. Para cada comando, indicado pelo opcode, há uma função correspondente, além de outras duas funções auxiliares "encontra_registrador" e "printBinary20Bits", que são, respectivamente, uma função que encontra um registrador para a realização de algum comando e uma função que auxiliará na impressão dos elementos da memória principal.

alu.h - armazena todos os protótipos das funções da alu.c

memoria.c - possui as funções de alocação e desalocação da memória de instrução e da memória principal, além de possuir a função de carregar a memória de instruções a partir do arquivo de texto e a função que irá executar o algoritmo contido no arquivo.

memoria.h - possui todos os protótipos das funções do arquivo memoria.c

main.c - é o arquivo onde ocorre a junção de todos os outros arquivos, e é por meio deste arquivo que o emulador é executado. Todas as variáveis respectivas às estruturas de dados mencionadas são declaradas e manipuladas neste arquivo.

Uma vez que as partes do emulador foram apresentadas, é possível descrever como ocorre a execução do emulador. Primeiramente, todas as variáveis são devidamente declaradas, os 16 registradores são alocados e as variáveis de memória são alocadas e inicializadas. Depois, o arquivo de texto é lido e carregado no vetor que contém todas as linhas do arquivo de texto, e o número de linhas total do arquivo é obtido nesse processo. Em seguida, inicia-se o processo de execução do algoritmo, em que a memória de instruções é lida, e conforme o comando requisitado, é atribuído um número ao opcode correspondente à função do comando. Posteriormente, tal função é chamada e as operações lógicas ou aritméticas são feitas, se o comando tiver uma condicional de salto e o resultado desta for verdadeiro, a memória de instruções recebe a linha em questão, caso contrário a memória de instruções receberá a linha seguinte, encerrando o algoritmo quando o último índice da memória de instruções é acessado. Por fim, todas as variáveis alocadas são desalocadas e a emulação termina. A figura 2 representa um fluxograma de execução do emulador, e a figura 3 mostra a etapa principal, que é a etapa de execução do algoritmo carregado na memória de instruções.

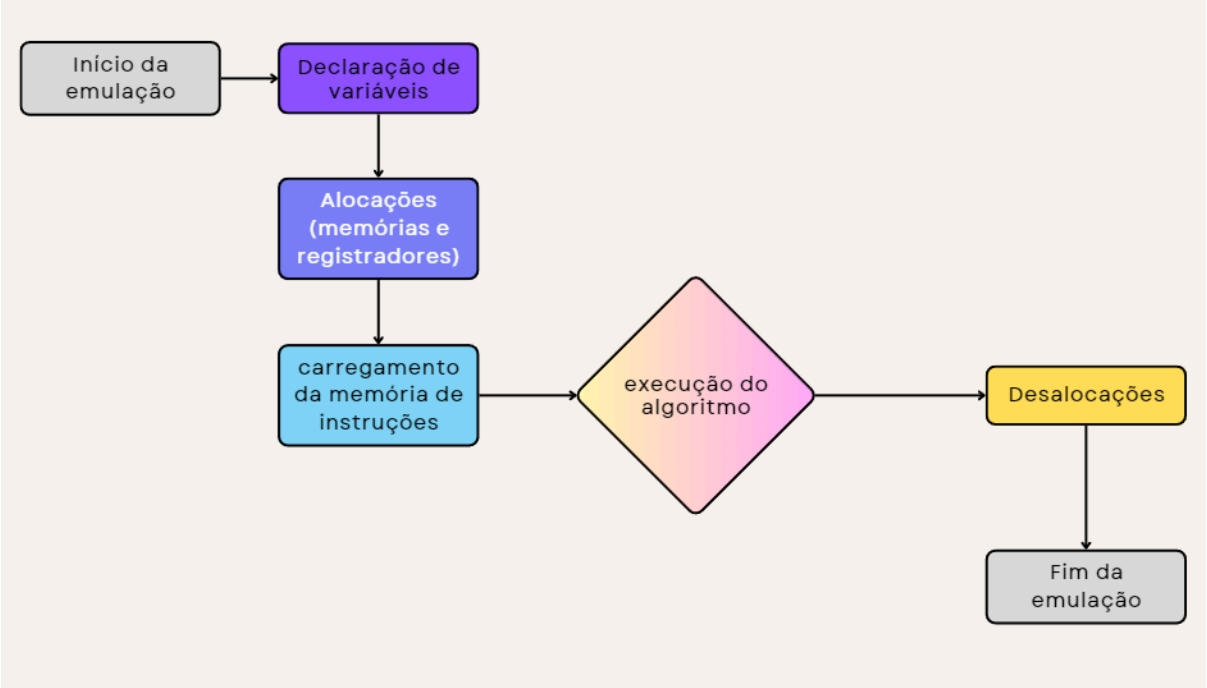

Figura 2 - fluxo de execução do emulador

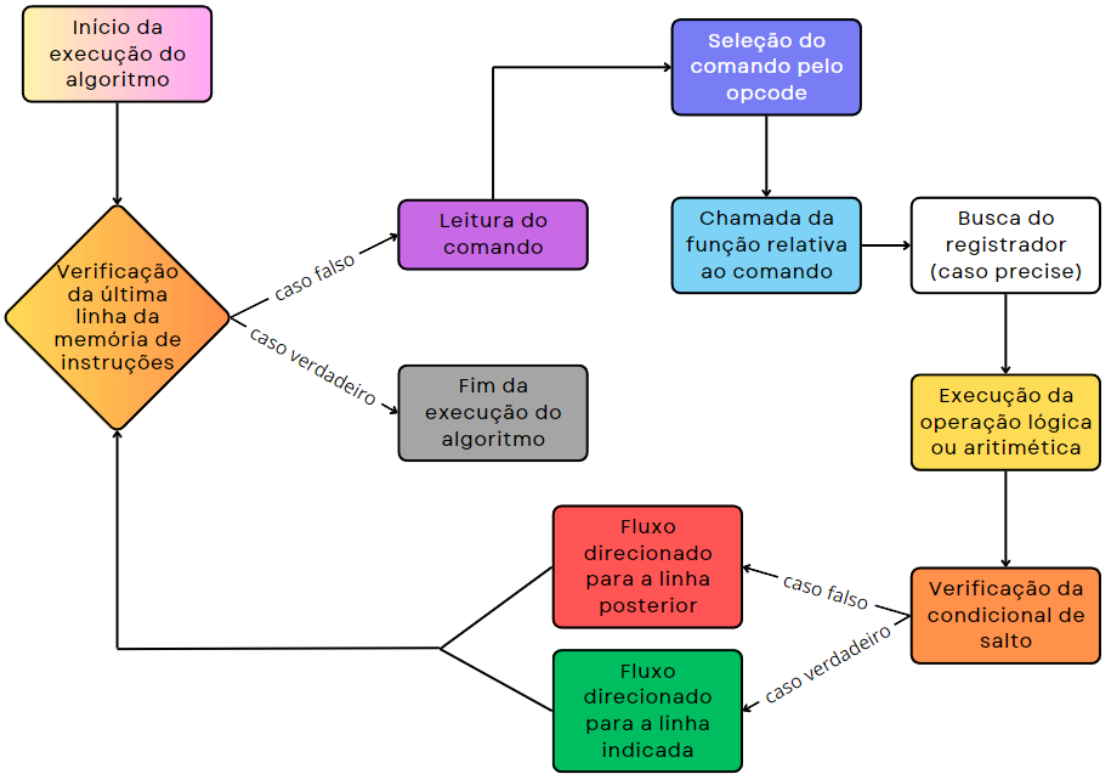

Figura 3 - fluxo de execução do algoritmo

Em relação à métrica utilizada para a avaliação dos dois algoritmos, cada algoritmo foi submetido a 6 testes, o primeiro algoritmo foi avaliado quanto ao armazenamento correto de todos os números primos de 1 a 100 nos endereços da memória principal de maneira sequencial e se esses números são impressos no terminal, e o segundo foi avaliado quanto à precisão das casas decimais do seno e do cosseno do ângulo e se o resultado é apresentado no terminal. Além disso, ambos foram avaliados segundo o tempo de execução em cada teste. Para o cálculo do tempo de execução do emulador, a biblioteca <sys/time.h> foi utilizada, e o cálculo do tempo decorrido é efetuado pela função “gettimeofday”, que marca a hora atual, com precisão de microssegundos, quando o emulador passa pela linha em que a função foi chamada. O tempo de execução, portanto, é encontrado pela subtração da hora final pela hora inicial e ele é convertido para milissegundos.

IV. Resultados

Para o primeiro algoritmo, o emulador conseguiu encontrar todos os números primos de 1 a 100 com uma taxa de acerto de 100% em todos os testes. Os tempos de execução em cada teste, respectivamente, e em milissegundos, foram:

Teste 1: 3.866
Teste 2: 2.860
Teste 3: 3.344
Teste 4: 2.816
Teste 5: 2.915
Teste 6: 2.746

Observa-se que os tempos de execução estão próximos de 3 a 4 milissegundos para cada execução do teste. Além disso, a memória principal foi utilizada de forma sequencial, como esperado. A figura 4 mostra uma saída do emulador para esse algoritmo, nela tem-se qual memória foi utilizada para armazenar os números e o endereço onde cada número foi armazenado, em números binários.

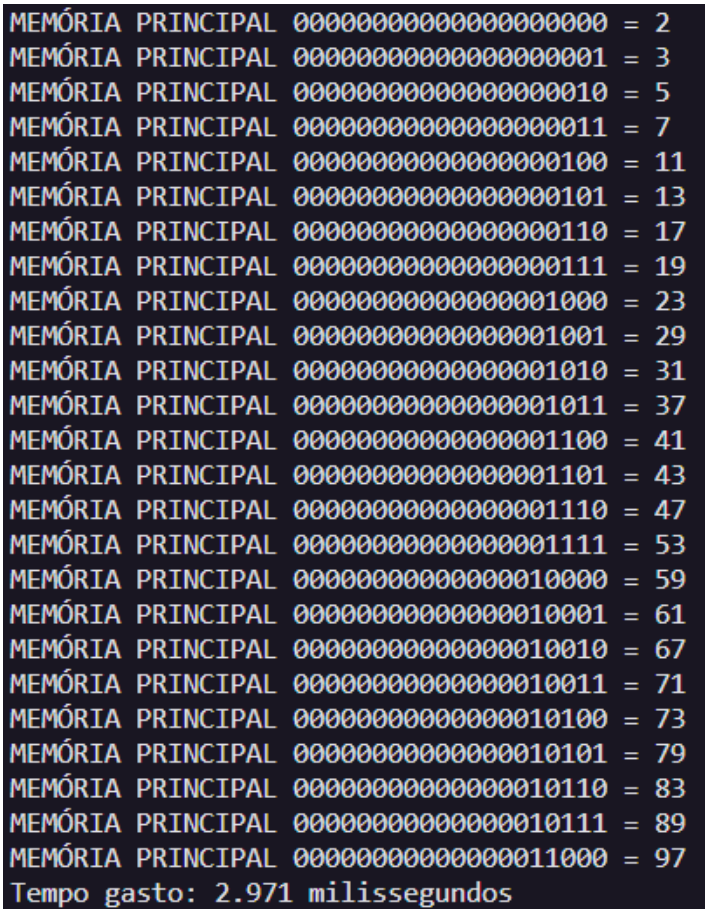

Figura 4 - exemplo de saída do algoritmo de números primos

Para o segundo algoritmo, foi observado que o emulador consegue executá-lo até ângulos de 53 radianos. O emulador não consegue calcular valores acima deste ângulo pois o limite do valor que um registrador consegue comportar é atingido, e nesses casos, os resultados obtidos são números muito diferentes dos esperados. Entretanto, para valores até 53 radianos, o emulador conseguiu calcular, na maioria dos testes, pelo menos 5 casas decimais de precisão para o seno e para o cosseno. Os tempos de execução desse algoritmo são maiores que o algoritmo 1, pois os cálculos são mais extensos e o arquivo de texto possui mais comandos. Dados também em milissegundos, são eles:

Teste 1 - ângulo de 9 radianos: 40.521
Teste 2 - ângulo de 18 radianos: 38.893
Teste 3 - ângulo de 27 radianos: 39.365
Teste 4 - ângulo de 36 radianos: 40.257
Teste 5 - ângulo de 45 radianos: 41.374
Teste 6 - ângulo de 53 radianos: 35.170

É perceptível que os tempos estão numa faixa próxima de 35 a 40 milissegundos por teste. A figura 5 ilustra um exemplo de saída do algoritmo 2 quando o ângulo é de 1.5707963 (aproximadamente 90°), sendo que o primeiro valor de ACC impresso na tela representa o seno do ângulo de interesse, e o segundo valor de ACC representa o cosseno do ângulo.

```
ACC: 1.000000
ACC: 0.000000
Tempo gasto: 26.001 milissegundos
```

Figura 5 - saída do algoritmo para o cálculo de seno e cosseno de 90°

Finalmente, as figuras de 6 a 9 expressam todos os resultados descritos graficamente: a figura 6 diz respeito à taxa de acerto do algoritmo 1, a figura 7 mostra o tempo de execução desse algoritmo, a figura 8 mostra quantas casas decimais o algoritmo 2 conseguiu acertar para cada um dos ângulos e a figura 9 mostra o tempo de execução do algoritmo 2.

Figura 6 - gráfico da taxa de acerto do algoritmo 1

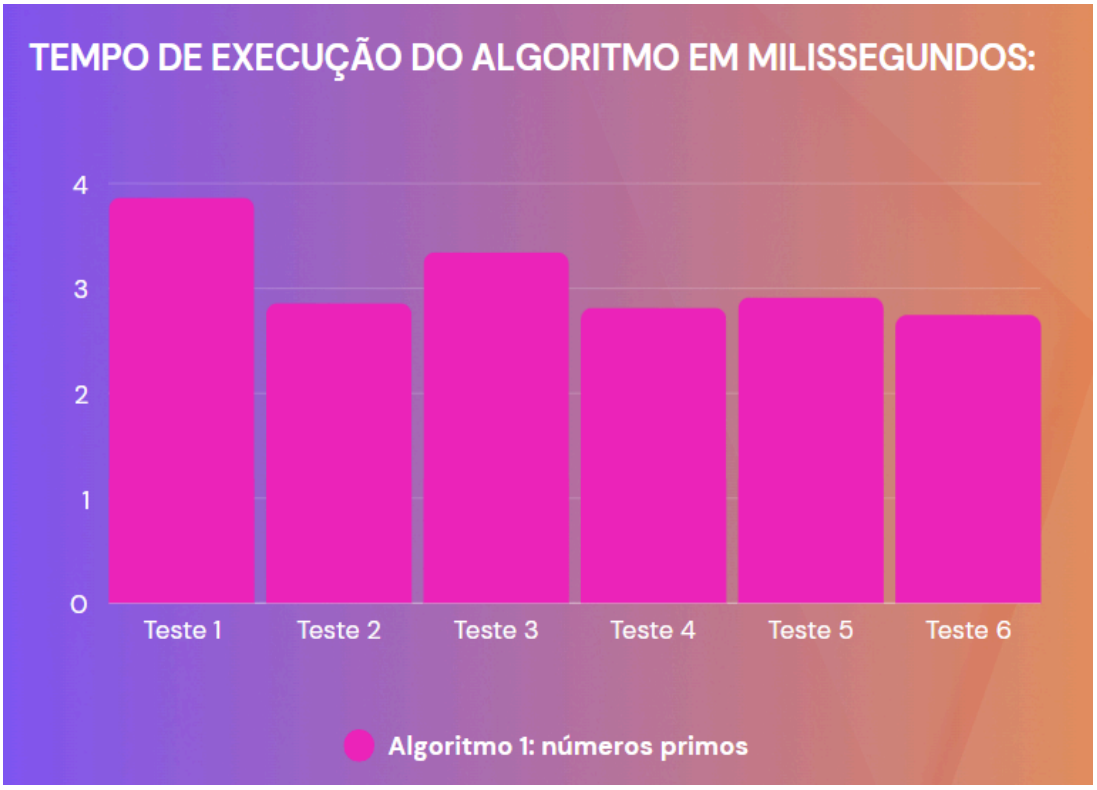

Figura 7 - gráfico do tempo de execução do algoritmo 1

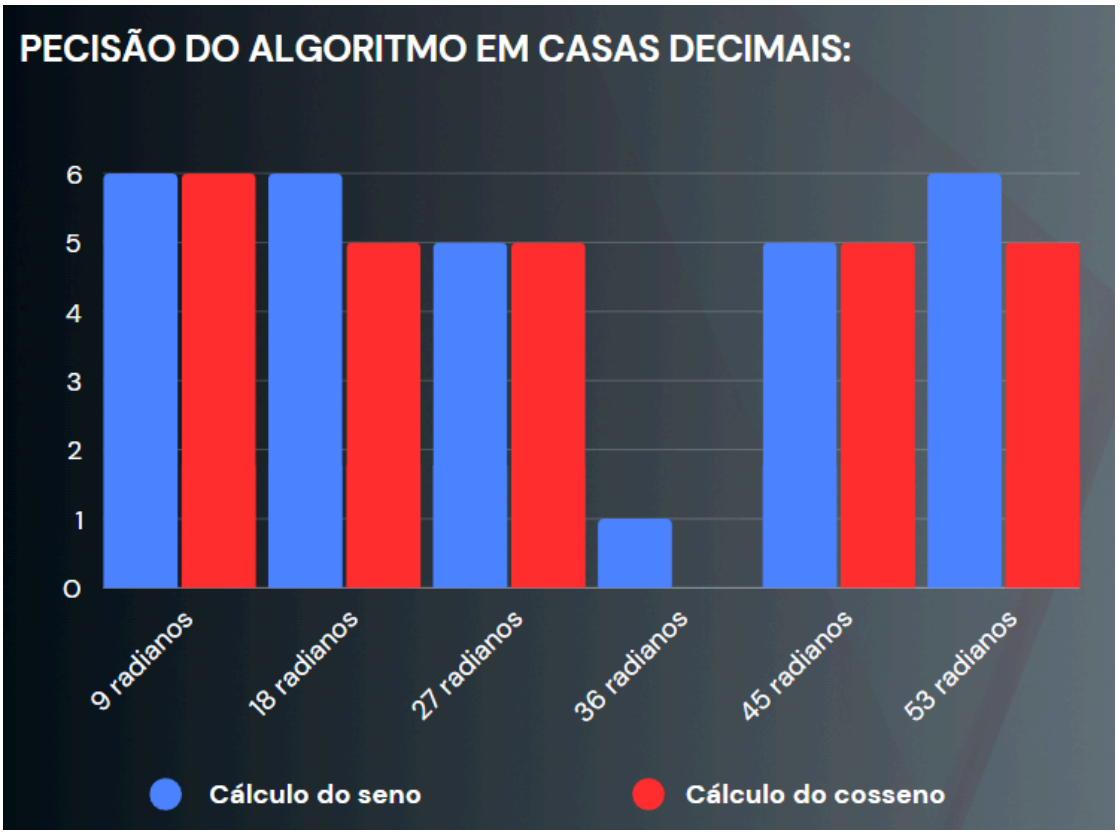

Figura 8 - gráfico da precisão de casas decimais do algoritmo 2

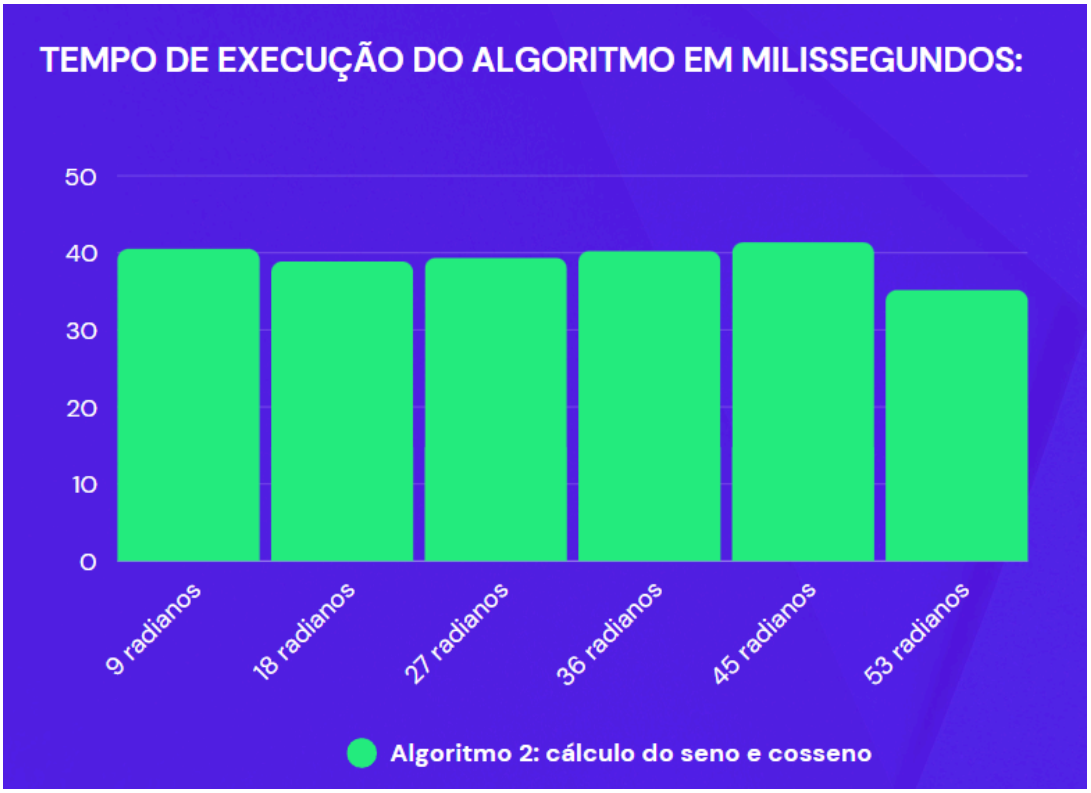

Figura 9 - gráfico do tempo de execução do algoritmo 2

V. Conclusão

Este trabalho possui a finalidade de emular um processador, com a consolidação de uma ISA cujas definições foram feitas pelos membros do grupo. O emulador deve ser capaz de, a partir de um arquivo de texto, produzir dois algoritmos: o primeiro irá encontrar, armazenar e imprimir os números primos de 1 a 100, enquanto o segundo deve calcular o seno e o cosseno de um ângulo de interesse, dado em radianos.

Para a construção do emulador, várias estruturas de dados foram implementadas, cada uma com uma função específica no código, a fim de emular os componentes de um processador da arquitetura de Von Neumann. O emulador consegue executar 16 comandos, possui 16 registradores, memória principal de 16MB, instruções limitadas até 32 bits, carregamento da memória de instruções para que os algoritmos sejam executados por ela e dois dispositivos de saída, um para o registrador especial de saída ACC e outro para a memória principal. Além disso, o emulador consegue mostrar o tempo de execução de um algoritmo em milissegundos. As métricas utilizadas foram, para o algoritmo 1, encontrar e armazenar todos os números primos de 1 a 100, e imprimir na tela do usuário todos eles e seus endereços. O algoritmo 2 foi avaliado em casas de precisão do cálculo do seno e do cosseno, e ambos os algoritmos foram testados quanto ao tempo de execução.

Sobre os resultados apresentados, o algoritmo 1 teve 100% de acerto em todos os testes e o tempo de execução esteve próximo do intervalo de 3 a 4 milissegundos, enquanto o algoritmo 2 conseguiu encontrar, na maioria dos testes, pelo menos 5 casas decimais de acerto, com tempo de execução entre 35 e 40 milissegundos.

Em conclusão, os resultados chegaram perto do esperado em todos os testes realizados no trabalho, portanto, o emulador consegue, na maioria das vezes, executar todas as funções propostas pela ISA do grupo. Além disso, os algoritmos são coerentes às suas respectivas propostas, pois as saídas do emulador comprovam sucesso na maior parte dos casos.

VI. Manual do emulador

O arquivo zip relativo a este trabalho possui o nome de "*tp2_org_comp*", e contém as bibliotecas mencionadas (isa.h, alu.h, memoria.h) com os arquivos .c respectivos e o arquivo principal main.c. Também será enviado junto a estas bibliotecas três arquivos de texto, um arquivo com o nome de "*texto_primos.txt*" que contém o algoritmo 1 mostrado neste relatório, um arquivo com o nome de "*texto_seno_cosseno.txt*" que possui o algoritmo 2 também mostrado neste relatório e um arquivo vazio com o nome de "*texto.txt*", que é o arquivo pelo qual o emulador executa os algoritmos. Para executar um desses dois algoritmos, é necessário que o algoritmo de interesse seja copiado e colado dentro do arquivo "*texto.txt*" e que este arquivo esteja na mesma pasta do executável do emulador.

O emulador foi criado e utilizado em ambiente Linux, e o compilador utilizado foi o gcc. Um arquivo do tipo makefile será enviado para que todas as bibliotecas e o arquivo principal sejam compilados apenas com o comando "make" no terminal, criando junto um executável com o nome de "exe". Entretanto, o emulador também pode ser compilado com as seguintes diretivas de compilação: **gcc alu.c alu.h isa.c isa.h memoria.c memoria.h main.c -g -Wall -o exe**

É importante destacar que todos esses arquivos precisam estar na mesma pasta.

Com respeito ao algoritmo 2, vale frisar que para trocar o ângulo a ser calculado, basta trocar o valor do registrador 3 na linha 5 para outro ângulo de interesse, e para alterar o número de parcelas de soma basta trocar o valor do registrador 9 na linha 7. Ao final do processo, o arquivo de texto com os novos valores precisa ser salvo e não é necessário compilar as bibliotecas novamente.

Por fim, para que o algoritmo seja executado junto com o emulador, basta digitar o comando **./exe** e o emulador será iniciado.