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

Projetando uma ISA

BCC266 - Organização de Computadores

Gabriel Vilas Novas Sousa
Thiago Ayolphi Liuth
Professor: Mateus Coelho Silva

Ouro Preto
14 de janeiro de 2024

Sumário

Lista de Figuras

Lista de Tabelas

1 Introdução

O trabalho consiste na criação de um programa que simule o funcionamento de uma ISA (Instruction Set Architecture). Uma Instruction Set[2][1], é descrita por Paul Kirvan como um conjunto de comandos para uma Unidade de Processamento Central (CPU) em linguagem de máquina. O termo pode ser referir a todas as possíveis instruções para uma CPU ou um subconjunto de instruções que tenham o objetivo de melhorar sua performance em certas situações.

Uma Instruction Set Architecture[1] seria, então, parte do modelo abstrato de um computador que define como a CPU é controlado pelo software. Ela cumpre o papel de interface entre o hardware e o software, especificando tanto o que o processador é capaz de fazer, quanto como deve ser feito.

1.1 Especificações do problema

A ISA simulada possui uma série de condições:

- Máximo de 16 instruções.
- Operações aritméticas podem ser soma, subtração, multiplicação, divisão ou shifts.
- Máximo de 16 registradores
- Instruções de no máximo 32 bits
- Existência de memória principal e banco de registradores, sem cache.
- Memória principal com até 16MB
- Programa deve ser carregado em uma unidade de memória especial.
- O programa deve ter pelo menos um dispositivo de saída para visualizar o resultado ao final da execução.
- O programa deve ser realizado em C, C++ ou em Python

Os conjuntos de instruções lidos pela ISA devem ser capazes de resolver os seguintes problemas:

1. Encontrar todos os números primos no intervalo de 1 a 100, os armazenar em memória, e em seguida os imprimir;
2. Dado um valor em radianos, calcular seu seno e o seu cosseno.

1.2 Considerações iniciais

A dupla optou por elaborar o código no VSCode, utilizando a linguagem Python.

- Ambiente de desenvolvimento do código fonte: VSCode. [2]
- Linguagem utilizada: Python.
- Ambiente de desenvolvimento da documentação: Overleaf LaTeX. [3]
- Bibliotecas utilizadas: struct, math, time.

O programa consiste em:

- Um arquivo principal.py, responsável por inicializar as variáveis, manipular as instruções para serem utilizadas, e controlar o fluxo dos scripts.
- Um arquivo operacoes.py, que conta com uma classe Operacoes(), a qual contém as operações aritméticas e as de comparação.

[1] do inglês, conjunto de instruções
[2] VSCode está disponível em https://code.visualstudio.com/
[3] Disponível em https://www.overleaf.com/

- Um arquivo registradores.py, com as classes Registradores() e Memoria(), que simulam ambas essas formas de armazenamento no formato de dicionários Python.
- Um arquivo conversor.py, que contém as funções binary32_to_float e float_to_binary32

O programa é acompanhado de dois arquivos .txt (primos.txt e sencons.txt), que contém os conjuntos de instruções para resolver cada um dos algoritmos.

1.3 Especificações da máquina

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

- Modelo: Dell Inspiron 15 5510.
- Processador: 11th Gen Intel® Core™ i7-11390H @ 3.40GHz × 8.
- Memória RAM: 16Gb.
- Sistema Operacional: Ubuntu 22.04.3 LTS.

1.4 Instruções de execução

Para a execução do projeto, garante que todos os arquivos se encontram na mesma pasta, incluindo os arquivos .txt, Então, digite no terminal:

```
make
```

Será oferecida a opção de escolher qual algoritmo será lido. A opção 1 se refere ao problema dos números primos. A opção 2 se refere ao problema do seno e cosseno. Caso opte pela opção dois, será requisitado ao usuário entrar o valor do qual será calculado o seno e cosseno. Esse valor deve ser um inteiro.

2 Implementação

O primeiro passo foi a estruturação da memória principal e dos registradores. Optamos por utilizar dicionários python por acharmos intuitivo, embora nesse caso específico utilizar vetores teria efeito muito similar (pois as chaves consistem em números inteiros crescentes iniciados em 0). A princípio, cada espaço de memória receberia strings de 8 bits. Mais tarde, na execução do segundo script, as memórias passaram a receber strings de 32 bits, para poderem receber valores double. Além disso, optamos por armazenar todos os valores no formato float 32 bits e realizar conversões para inteiros quando necessário.

2.1 Instruções

Todas as nossas instruções tem 28 bits, quatro que caracterizam o OPCODE e três bytes de valores int. Os valores int são usados como endereços de registradores em quase todas as circustâncias, e diretamente como inteiro na instrução STRS.

Trechos da instrução

[OPCODE — adress1 — adress2 — adress3]

Além disso, há uma variável de armazenamento temporário, responsável por receber o resultado de operações aritméticas, chamada de *holder*. A lista completa de instruções é a seguinte:

Comandos de manipulação das memórias

- STRS (Store in Register): Armazena no adress1 o valor inteiro do binário adress2, convertido para float.
- STHR (Store Holder in Register): Armazena no adress1 o valor em holder.
- STME (Store in Memory): Guarda o valor armazenado no registrador adress1 no endereço de memória apontado pelo conteúdo do registrador adress3
- LOAD: Carrega da memória para o registrador adress1 o valor apontado pelo conteúdo do registrador adress3.

Comandos de operação aritmética

- ADD_: Adiciona os valores dos registradores adress1 e adress2 e guarda em holder.
- SUB_: Subtrai do valor do registrador adress1 o valor do registrador adress2 e guarda em holder.
- MULT: Multiplica os valores dos registradores adress1 e adress2 e guarda em holder.
- RMND (Remainder): Pega o resto da divisão entre o valor do registrador adress1 pelo do registrador adress2, converte em float, e guarda em holder.
- DIV_: Divide o valor do registrador adress1 pelo de adress2 e guarda em holder.
- POW_ (Power): Eleva o valor armazenado no registrador adress1 ao valor armazenado em adress2 e guarda em holder.
- FAC_ (Factorial): Faz o fatorial do valor armazenado em adress1 e guarda em holder.

Comandos de controle de fluxo

- JUMP: Pula para a linha do valor de adress1
- JPIE (Jump if Equal): Pula para a linha do valor em adress3 caso o valor em adress1 seja igual ao valor em adress2.
- JPIG (Jump if Greater): Pula para a linha do valor em adress3 caso o valor em adress1 seja maior que o valor em adress2.
- PRNT (Print): Imprime o valor armazenado em adress1.
- END_: Encerra todos os comandos.

2.2 Planilha de montagem das instruções

Para facilitar a escrita das listas de instruções, criamos uma planilha que realiza a conversão de inteiros para seu valor em binário. Dessa forma, o planejamento não precisa ser feito em binário.

	COMANDO	ADRESS1	ADRESS2	ADRESS3	>>	COMANDO	ADRESS1	ADRESS2	ADRESS3
0	STRS	3	8		>>	STRS	00000011	00001000	00000000
1	STRS	4	1		>>	STRS	00000100	00000001	00000000
2	STRS	5	7		>>	STRS	00000101	00000111	00000000
3	STME	5		4	>>	STME	00000101	00000000	00000100
4	ADD_	5	1		>>	ADD_	00000101	00000001	00000000
5	ADD_	5	1		>>	ADD_	00000101	00000001	00000000
6	STHR	5			>>	STHR	00000101	00000000	00000000
7	ADD_	4	1		>>	ADD_	00000100	00000001	00000000
8	STHR	4			>>	STHR	00000100	00000000	00000000
9	JPIE	4	3	11	>>	JPIE	00000100	00000011	00001011
10	JUMP	3			>>	JUMP	00000011	00000000	00000000
11	END_				>>	END_	00000000	00000000	00000000
12					>>		00000000	00000000	00000000
13					>>		00000000	00000000	00000000
14					>>		00000000	00000000	00000000
15					>>		00000000	00000000	00000000

Figura 1: Planilha de montagem das instruções

3 Scripts

Estabelecida as memórias, as funções das operações, e as instruções, escrevemos os scripts para a resolução dos problemas. A utilização de comentários na planilha de montagem e a simulação do fluxo mentalmente foram essenciais para suas elaborações.

Além disso, o arquivo principal.py conta com uma função capaz de imprimir os conteúdos dos registradores e dos dez primeiros espaços da memória, essencial na depuração das operações e dos scripts.

3.1 O problema dos números primos

Para resolver o problema dos números primos, o script percorre todos os números de 2 a 100 (assume-se que o número 1 não precisa ser verificado pois por sua própria natureza não pode ter dois divisores distintos), verificando se são primos, e os armazena em memória. Ao atingir o valor 101, a verificação se encerra, são impressos os números primos armazenados, e o script é encerrado.

	COMANDO	ADRESS1	ADRESS2	ADRESS3	>>	COMANDO	ADRESS1	ADRESS2	ADRESS3
0	STRS	2	2		>>	STRS	00000010	00000010	00000000
1	STRS	3	2		>>	STRS	00000011	00000010	00000000
2	STRS	4	0		>>	STRS	00000100	00000000	00000000
3	STRS	5	2		>>	STRS	00000101	00000010	00000000
4	STRS	14	101			STRS	00001110	01100101	00000000
5	RMND	5	3		>>	RMND	00000101	00000011	00000000
6	STHR	7			>>	STHR	00000111	00000000	00000000
7	JPIE	3	5	32	>>	JPIE	00000011	00000101	00100000
8	JPIE	7	0	16	>>	JPIE	00000111	00000000	00010000
9	ADD_	3	1		>>	ADD_	00000011	00000001	00000000
10	STHR	3			>>	STHR	00000011	00000000	00000000
11	JPIE	5	14	18	>>	JPIE	00000101	00001110	00010010
12	JUMP	5			>>	JUMP	00000101	00000000	00000000
13	ADD_	2	0		>>	ADD_	00000010	00000000	00000000
14	STHR	3			>>	STHR	00000011	00000000	00000000
15	JUMP	5			>>	JUMP	00000101	00000000	00000000
16	JPIE	5	14	18	>>	JPIE	00000101	00001110	00010010
17	JUMP	29			>>	JUMP	00011101	00000000	00000000
18	ADD_	4	0		>>	ADD_	00000100	00000000	00000000
19	STHR	15			>>	STHR	00001111	00000000	00000000
20	JPIE	15	0	28	>>	JPIE	00001111	00000000	00011100
21	STRS	4	0		>>	STRS	00000100	00000000	00000000
22	LOAD	6		4	>>	LOAD	00000110	00000000	00000100
23	PRNT	6			>>	PRNT	00000110	00000000	00000000
24	ADD_	4	1		>>	ADD_	00000100	00000001	00000000
25	STHR	4			>>	STHR	00000100	00000000	00000000
26	JPIE	4	15	28	>>	JPIE	00000100	00001111	00011100
27	JUMP	22			>>	JUMP	00010110	00000000	00000000
28	END_				>>	END_	00000000	00000000	00000000
29	ADD_	5	1		>>	ADD_	00000101	00000001	00000000
30	STHR	5			>>	STHR	00000101	00000000	00000000
31	JUMP	13			>>	JUMP	00001101	00000000	00000000
32	STME	5		4	>>	STME	00000101	00000000	00000100
33	ADD_	4	1		>>	ADD_	00000100	00000001	00000000
34	STHR	4			>>	STHR	00000100	00000000	00000000
35	JUMP	29			>>	JUMP	00011101	00000000	00000000

Figura 2: Planilha com o script para o problema dos números primos

Inicializando os registradores

- 0: Inicia o registrador 2 com o valor dois, para fins de reset do divisor no ciclo.
- 1: Inicia o registrador 3 com o valor 2, como primeiro divisor.

- 2: Inicia o registrador 4 com o valor 0, que aponta para qual endereço da memória será armazenado o primeiro primo encontrado.
- 3: Inicia o registrador 5 com o valor 2, que é o primeiro valor a ser testado.
- 4: Inicia o registrador 14 com o valor 101, valor que determina o fim do ciclo.

0	0	
1	1	
2	2	Dois para reset
3	2	Divisor
4	0	Endereço para storage
5	2	Valor em teste
6	X	Valor a ser impresso
7	/	Resultado da divisão
8		
9		
10		
11		
12		
13		
14	101	Teto do intervalo testado
15		Endereço do último valor armazenado em memória

Figura 3: Função dos registradores no problema dos números primos

Verificação do status de primo

- 5: Calcula o resto da divisão do valor em teste pelo divisor e armazena no registrador 7.
- 6: Armazena o resultado da operação no registrador 7.
- 7: Se o divisor for igual ao valor em teste, pula para a linha 32 (trecho de armazenamento)
- 8: Se o resultado armazenado no registrador 7 for igual a 0, pula para a linha 16, onde é verificado se os testes devem encerrar ou se deve-se testar o próximo número.
- 9: Soma um no valor do divisor.
- 10: Armazena o novo valor do divisor no registrador 3.
- 11: Se o valor em teste for igual ao valor no registrador 14 (fim do intervalo), pula para a linha 18, para iniciar as impressões.
- 12: Pula para a linha 5 e testa o valor com o novo divisor.
- 16: Se o valor em teste for igual ao valor no registrador 14 (fim do intervalo), pula para a linha 18, para iniciar as impressões.
- 17: Pula para o início dos resets (linha 29).

Reset dos registradores para testagem dos próximos números

- 29 e 30: Incrementa o valor em teste em 1.
- 31: Pula para a linha 13.
- 13 e 14: Define o registrador do divisor como 2 nova ente
- 15: Volta para o trecho de testes, testando agora o valor seguinte no intervalo.

Trecho de armazenamento

- 32: Constatado que um número é primo, o guarda na memória apontada pelo registrador 4.
- 33 e 34: Incrementa em um o registrador que aponta onde será armazenado o próximo primo.
- 35: Pula para o início dos resets (linha 29).

Trecho de impressão

- 18 e 19: Armazena o último endereço de memória em que foi armazenado um primo no registrador 15.
- 20: Se não tiver nada armazenado, encerra o script.
- 21: Define o registrador 4 como 0 (ele vai apontar para o valor a ser impresso).
- 22: Busca na memória o valor armazenado no endereço apontado pelo registrador 4 e o armazena no registrador 6.
- 23: Imprime o conteúdo do registrador 6.
- 24 e 25: Incrementa em um o registrador que aponta para o endereço da memória.
- 26: Caso o conteúdo do registrador 4 chegue ao fim do intervalo de impressões, pula para a linha 28 (Encerra o script).
- Reinicia o ciclo de impressão.

A linha 28 contém a instrução END_, que encerra o script.

3.2 O problema do seno e cosseno

Para resolver o problema do cálculo do seno e do cosseno de um valor em radianos, o algoritimo se utiliza da expansão das séries de MacLaurin que definem o seno e o cosseno.

$$sen\,(x) = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n+1}}{(2n+1)!} = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots$$

$$\cos(x) = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n}}{(2n)!} = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \dots$$

Figura 4: Expansão das séries de MacLaurin que definem o seno e o cosseno.

Basicamente, o script calcula uma aproximação do seno e do cosseno utilizando 6 ou 8 ciclos (os resultados em ambos foram tabelados). Em cada ciclo são realizadas uma operação de adição e uma operação de subtração da série.

O valor em radianos é entrado pelo usuário e armazenado no registrador 2.

	COMANDO	ADRESS1	ADRESS2	ADRESS3	>>	COMANDO	ADRESS1	ADRESS2	ADRESS3
0	STRS	15	6		>>	STRS	00001111	00000110	00000000
1	STRS	5	1		>>	STRS	00000101	00000001	00000000
2	STRS	14	0		>>	STRS	00001110	00000000	00000000
3	POW_	2	5		>>	POW_	00000010	00000101	00000000
4	STHR	6			>>	STHR	00000110	00000000	00000000
5	FACT	5			>>	FACT	00000101	00000000	00000000
6	STHR	7			>>	STHR	00000111	00000000	00000000
7	DIV_	6	7		>>	DIV_	00000110	00000111	00000000
8	STHR	8			>>	STHR	00001000	00000000	00000000
9	ADD_	9	8		>>	ADD_	00001001	00001000	00000000
10	STHR	9			>>	STHR	00001001	00000000	00000000
11	ADD_	5	1		>>	ADD_	00000101	00000001	00000000
12	STHR	5			>>	STHR	00000101	00000000	00000000
13	ADD_	5	1		>>	ADD_	00000101	00000001	00000000
14	STHR	5			>>	STHR	00000101	00000000	00000000
15	POW_	2	5		>>	POW_	00000010	00000101	00000000
16	STHR	6			>>	STHR	00000110	00000000	00000000
17	FACT	5			>>	FACT	00000101	00000000	00000000
18	STHR	7			>>	STHR	00000111	00000000	00000000
19	DIV_	6	7		>>	DIV_	00000110	00000111	00000000
20	STHR	8			>>	STHR	00001000	00000000	00000000
21	SUB_	9	8		>>	SUB_	00001001	00001000	00000000
22	STHR	9			>>	STHR	00001001	00000000	00000000
23	ADD_	14	1		>>	ADD_	00001110	00000001	00000000
24	STHR	14			>>	STHR	00001110	00000000	00000000
25	JPIE	14	15	31	>>	JPIE	00001110	00001111	00011111
26	ADD_	5	1		>>	ADD_	00000101	00000001	00000000
27	STHR	5			>>	STHR	00000101	00000000	00000000
28	ADD_	5	1		>>	ADD_	00000101	00000001	00000000
29	STHR	5			>>	STHR	00000101	00000000	00000000
30	JUMP	3			>>	JUMP	00000011	00000000	00000000
31	JPIE	13	1	38	>>	JPIE	00001101	00000001	00100110
32	STRS	14	0		>>	STRS	00001110	00000000	00000000
33	STRS	5	0		>>	STRS	00000101	00000000	00000000
34	STRS	13	1		>>	STRS	00001101	00000001	00000000
35	PRNT	9			>>	PRNT	00001001	00000000	00000000
36	STRS	9	0		>>	STRS	00001001	00000000	00000000
37	JUMP	3			>>	JUMP	00000011	00000000	00000000
38	PRNT	9			>>	PRNT	00001001	00000000	00000000
39	END_				>>	END_	00000000	00000000	00000000

Figura 5: Planilha com o script para o problema do seno e cosseno

Inicializando os registradores

- 0: Inicializa o registrador 15 com o valor 6 (número de ciclos do script)
- 1: Inicializa o registrador 5 com o valor 1 (primeiro expoente e primeiro fatorial). Será referido daqui para frente como n.
- 2: Inicializa o registrador 14 com o valor 0 (número da iteração atual)

Ciclo de cálculo

- 3 e 4: Calcula a potência do valor em radianos elevado a n e armazena no registrador 6.
- 5 e 6: Calcula o fatorial de n e armazena no registrador 7.
- 7 e 8: Calcula a divisão da potência pelo fatorial e armazena no registrador 8.
- 9 e 10: Soma o resultado da divisão ao total acumulado, no registrador 9.
- 11 a 14: n é incrementado em 2.

- 15 e 16: Calcula a potência do valor em radianos elevado a n e armazena no registrador 6.
- 17 e 18: Calcula o fatorial de n e armazena no registrador 7.
- 19 e 20: Calcula a divisão da potência pelo fatorial e armazena no registrador 8.
- 21 e 22: Subtrai o resultado da divisão do total acumulado, no registrador 9.
- 23 e 24: Soma um ao contador do número de ciclo atual.
- 25: Se o número de ciclos completos for igual ao número total de ciclos, pula para o reset dos registradores.
- 26 a 29: n é incrementado em 2.
- 30: Reinicia o ciclo.

Reset dos registradores para cálculo do cosseno

- 31: Se o valor no registrador 13 for igual a 1, pula para a linha 38, onde será impresso o total acumulado do cosseno e encerrado o código.
- 32: Reseta o número do ciclo atual para 0.
- 33: Define n como 0.
- 34: Define 13 como 1 (flag para indicar que o ciclo do cosseno já foi iniciado).
- 35: Imprime o total acumulado (até o momento, é o valor calculado para o seno).
- 36: Reseta o total acumulado para 0.
- 37: Pula para o início do ciclo de cálculo.
- 38: Imprime o total acumulado (dessa vez, será o valor calculado para o cosseno).
- 39: Encerra o script.

4 Testes

4.1 Números Primos

O script é bem sucedido em calcular, armazenar e imprimir todos os números primos entre 1 e 100, levando em torno de 0.02 e 0.03 segundos para executar.

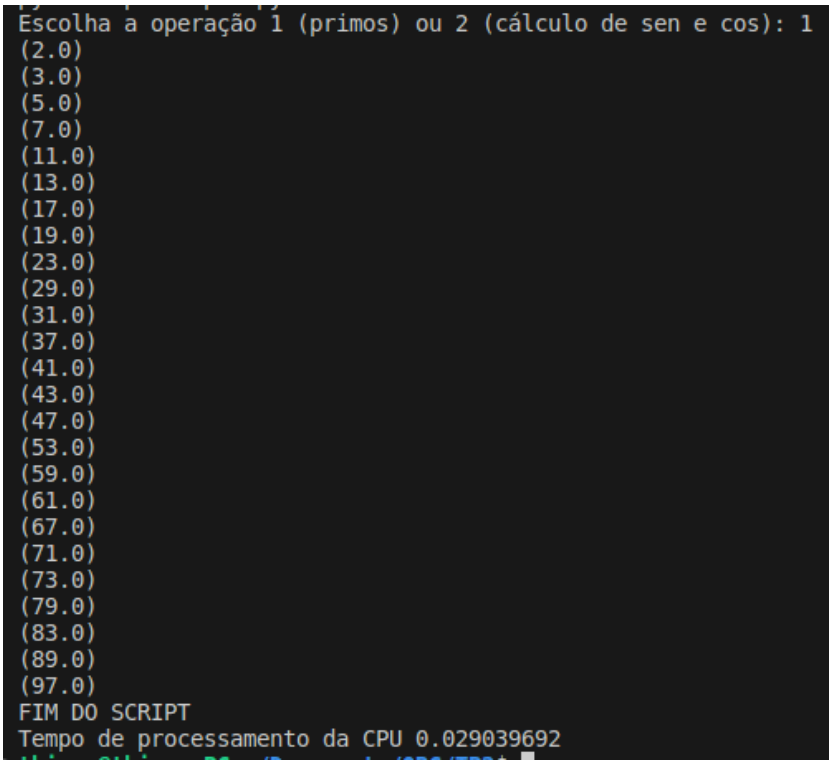

Figura 6: Impressão dos números primos

Veja que os números são impressos no formato double, pois é como são armazenados. Como dito anteriormente, os valores só são convertidos em inteiros pelo código nas operações onde necessário.

4.2 Seno e Cosseno

O código também se mostrou eficaz no cálculo do seno e do cosseno de um valor em radianos, como pode ser averiguado na tabela abaixo:

Teste com 6 ciclos

Radianos	Seno (script)	Seno (Calculadora)	Cosseno (script)	Cosseno (Calculadora)
0	0	0	1	1
1	0.841470956802	0.84147099	0.5403022766113	0.5403023
2	0.909297406673	0.90929713	-0.416146814823	-0.41614749
8	0.987129032611	0.98935834	-0.152437627315	-0.14549941
10	-1.1069567203	-0.54402118	-2.232305526733	-0.83907148

Tabela 1: Valores do seno e cosseno para diferentes radianos, 6 ciclos

Teste com 8 ciclos

Radianos	Seno (script)	Seno (Calculadora)	Cosseno (script)	Cosseno (Calculadora)
0	0	0	1	1
1	0.841470956802	0.84147099	0.54030227661132	0.5403023
2	0.909297406673	0.90929713	-0.41614681482315	-0.41614749
8	0.989358901977	0.98935834	-0.1455170363187	-0.14549941
10	-0.544004738330	-0.54402118	-0.83949863910675	-0.83907148

Tabela 2: Valores do seno e cosseno para diferentes radianos, 8 ciclos

O tempo de execução desse script foi em torno de 0.002 e 0.003 segundos.

5 Análise

A utilização do formato double no armazenamento dos números permite trabalhar com operações mais complexas, o que foi essencial para solucionar o problema do cálculo do seno e do cosseno. O script do problema dos números primos também funcionaria com memórias de 1 byte, armazenando inteiros. Porém, a ISA seria limitada.

Ao comparar o tempo de execução de ambos os algoritmos, é possível verificar que o dos números primos leva uma média de dez vezes mais tempo para executar. Isso provavelmente se deve à necessidade de múltiplos loops para testar cada número do intervalo. A quantidade de testes pode ser reduzida, limitando o teto de testagem como metade do valor em teste.

Na execução do segundo script, tomamos que a aproximação do seno e do cosseno por séries de MacLaurin oferecem uma fórmula de fácil implementação para o cálculo nesse contexto, mas ao analisar os valores tabelados, pode-se perceber que o resultado do algoritmo perde precisão conforme o valor em radianos aumenta, sendo necessários mais ciclos de execução para manter uma precisão aceitável.

Além disso, no teste com 8 ciclos, o algoritmo só conseguiu sustentar sua execução até o valor 17 em radianos. Após isso, os números flutuantes ficam grandes demais para serem armazenados na memória de 32 bits. Já com 6 ciclos, foi possível calcular até o seno e cosseno de 47 radianos, porém com péssima precisão. Isso mostra que há um equilíbrio entre precisão e capacidade de processamento que deve ser buscada de acordo com o contexto e a necessidade.

6 Conclusão

O trabalho mostrou que é possível simular o funcionamento de uma ISA em linguagem python, o que apresenta alto valor didático no aprendizado de seu funcionamento. Com ele, passamos a compreender melhor os Conjuntos de Instruções e como ocorre a comunicação entre software e hardware.

Acreditamos ter realizado um trabalho consistente com o que foi solicitado e que adquirimos uma boa base para estudarmos mais sobre o assunto futuramente.

Referências

[1] ARM. Instruction set architecture (isa). [Online; accessed 13-janeiro-2024].

[2] Paul Kirvan. Definition instruction set, 2022. [Online; accessed 12-janeiro-2024].