INSTITUTO NACIONAL DE TELECOMUNICAÇÕES - INATEL

AGUINALDO DE SOUZA JÚNIOR - 1339

GABRIEL CLARET DE REZENDE - 1328

HENRIQUE BARCIA LANG - 1378

PROJETO DE ARQUITETURAS DE COMPUTADORES:

Máquina Virtual Inassembly

Santa Rita do Sapucaí – Minas Gerais

2019

SUMÁRIO

1 INTRODUÇÃO

O projeto desenvolvido é uma máquina virtual que emula o processador de uma arquitetura hipotética e sua interação com memórias virtuais (programa, dados e cache). Para seu desenvolvimento, foi utilizada a linguagem de programação Python, sendo o ambiente de desenvolvimento o editor de texto Visual Studio Code e o terminal do Windows/Linux para execução.

Com este projeto, deseja-se comprovar os conhecimentos adquiridos ao longo do semestre, demonstrando a funcionalidade de memória de programa, memória de dados, memória cache, ciclo de fetch, decodificação e execução de instruções e arquitetura de um processador.

2 DESCRIÇÃO GERAL

A máquina virtual traduz código fonte da linguagem de baixo nível Inassembly para código de máquina e, em seguida, processa o código gerado.

A arquitetura virtual desenvolvida apresenta palavra de 32 bits, 16 instruções e 8 registradores. Dentre esses registradores, 6 são de propósito geral e 2 de propósito específico.

Para a formulação da memória cache, utilizou-se o princípio da localidade – isto é, após a busca por uma certa instrução, a probabilidade de instruções próximas a ela serem solicitadas em seguida é alta. A memória cache apresenta uma estrutura 4x4, isto é, 4 linhas com um bloco de 4 instruções cada. O tamanho da memória cache é 64 bytes.

3 REGISTRADORES

Existem 8 registradores distintos na arquitetura, sendo 6 de uso geral e 2 de uso específico.

Registradores de **uso geral**: r0 (000), r1 (001), r2 (010), r3 (011), r4 (100), r5 (101)

Registradores de **uso específico**: zero (110) [zero imediato], r7 (111) [*Program Counter*]

4 INSTRUÇÕES

- (**add**) Adição aritmética entre dois registradores:

0000	0 (19)	rdest (3)	rsrc1 (3)	rsrc2 (3)

- (**addi**) Adição aritmética entre registrador e valor imediato positivo:

0001	rdest (3)	rsrc1 (3)	imm (22)

- (**sub**) Subtração aritmética entre dois registradores:

0010	0 (19)	rdest (3)	rsrc1 (3)	rsrc2 (3)

- (**subi**) Subtração aritmética entre registrador e valor imediato positivo:

0011	rdest (3)	rsrc1 (3)	imm (22)

- (**mult**) Multiplicação aritmética entre dois registradores:

0100	0 (19)	rdest (3)	rsrc1 (3)	rsrc2 (3)

- (**multi**) Multiplicação aritmética entre registrador e valor imediato positivo:

0101	rdest (3)	rsrc1 (3)	imm (22)

- (**div**) Divisão aritmética entre dois registradores:

0110	0 (19)	rdest (3)	rsrc1 (3)	rsrc2 (3)

- (**divi**) Divisão aritmética entre registrador e valor imediato positivo:

0111	rdest (3)	rsrc1 (3)	imm (22)

- (**load**) Carregar um valor instantâneo da memória de dados:

1000	0 (3)	rdest (3)	address (22)

- (**store**) Armazenar o valor contido em um registrador na memória de dados:

1001	0 (3)	rsrc (3)	address (22)

- (**jump**) Pular para um endereço de memória:

1010	0 (6)	address (22)

- (**bgt**) Pula para endereço de memória caso rsrc1 seja maior que rsrc2:

1011	rsrc1 (3)	rsrc2 (3)	address (22)

- (**blt**) Pula para endereço de memória caso rsrc1 seja menor que rsrc2:

1100	rsrc1 (3)	rsrc2 (3)	address (22)

- (**beq**) Pula para endereço de memória caso rsrc1 seja igual a rsrc2:

1101	rsrc1 (3)	rsrc2 (3)	address (22)

- (**move**) Atribui o valor armazenado no registrador rsrc em rdest:

1110	0 (22)	rdest (3)	rsrc (3)

- (**inout**) Imprime valor do registrador rdest ou lê uma entrada e armazena em rdest:

1111	0 (25)	rdest (3)

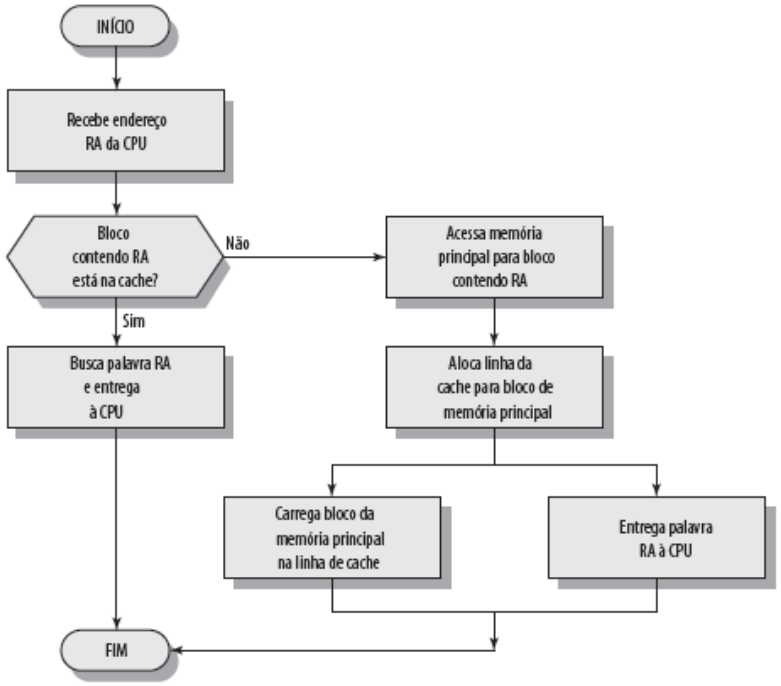
5 MEMÓRIA CACHE

A memória cache, por estar fisicamente mais próxima ao processador, garante um tempo de busca das instruções menor comparado à buscas na memória principal. A memória cache implementada baseia-se no princípio da localidade – isto é, após a busca por uma certa instrução, a probabilidade de instruções próximas a ela serem solicitadas em seguida é alta.

A memória cache apresenta uma estrutura 4x4, isto é, 4 linhas com 4 instruções por linha. O tamanho da memória cache é 64 bytes.

Para armazenar um bloco de instruções na memória cache, faz-se uso do valor do Program Counter. Os primeiros 28 bits determinam a tag, os próximos 2 bits a linha e os últimos 2 bits a coluna.

Cada linha da memória cache apresenta um campo denominado *tag*, que é responsável por discernir blocos de instruções distintos.



6 FUNCIONAMENTO

Para o desenvolvimento do projeto, foi utilizada a linguagem de programação Python, sendo o ambiente de desenvolvimento o editor de texto Visual Studio Code e o terminal do Windows/Linux para execução.

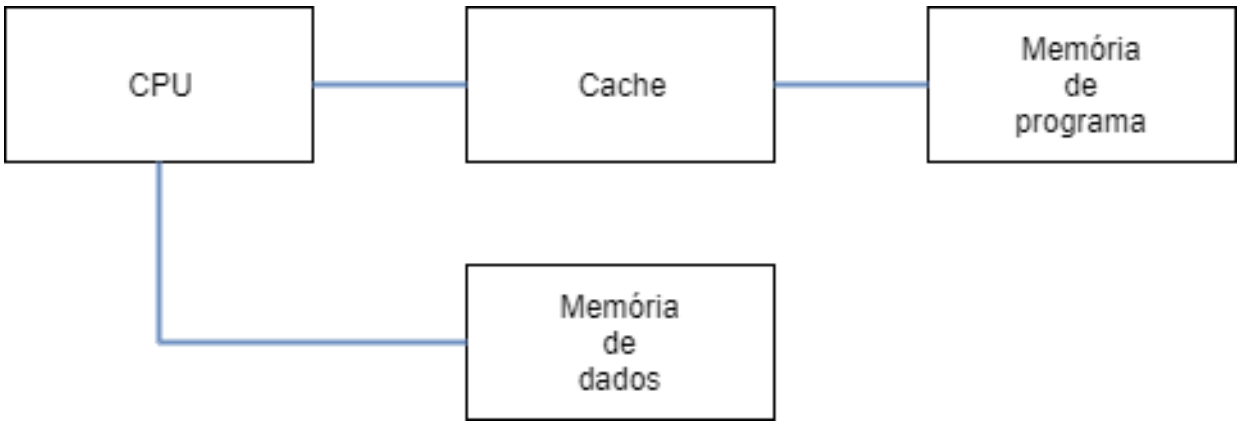
Ao carregar um código fonte da linguagem Inassembly, a máquina virtual primeiramente remove comentários e espaços desnecessários. Em seguida, realiza a tradução para linguagem de máquina e armazena as instruções na memória de programa.

Feito este processo de limpeza e tradução inicial, a máquina virtual processa as instruções da memória de programa da seguinte forma: atualiza o *Program Counter* (PC) e busca na memória cache a instrução. Caso haja um *miss* (ou seja, quando a linha está vazia ou a *tag* da linha é diferente da *tag* obtida pelo PC), um bloco de 4 instruções é carregado da memória principal para a memória cache.

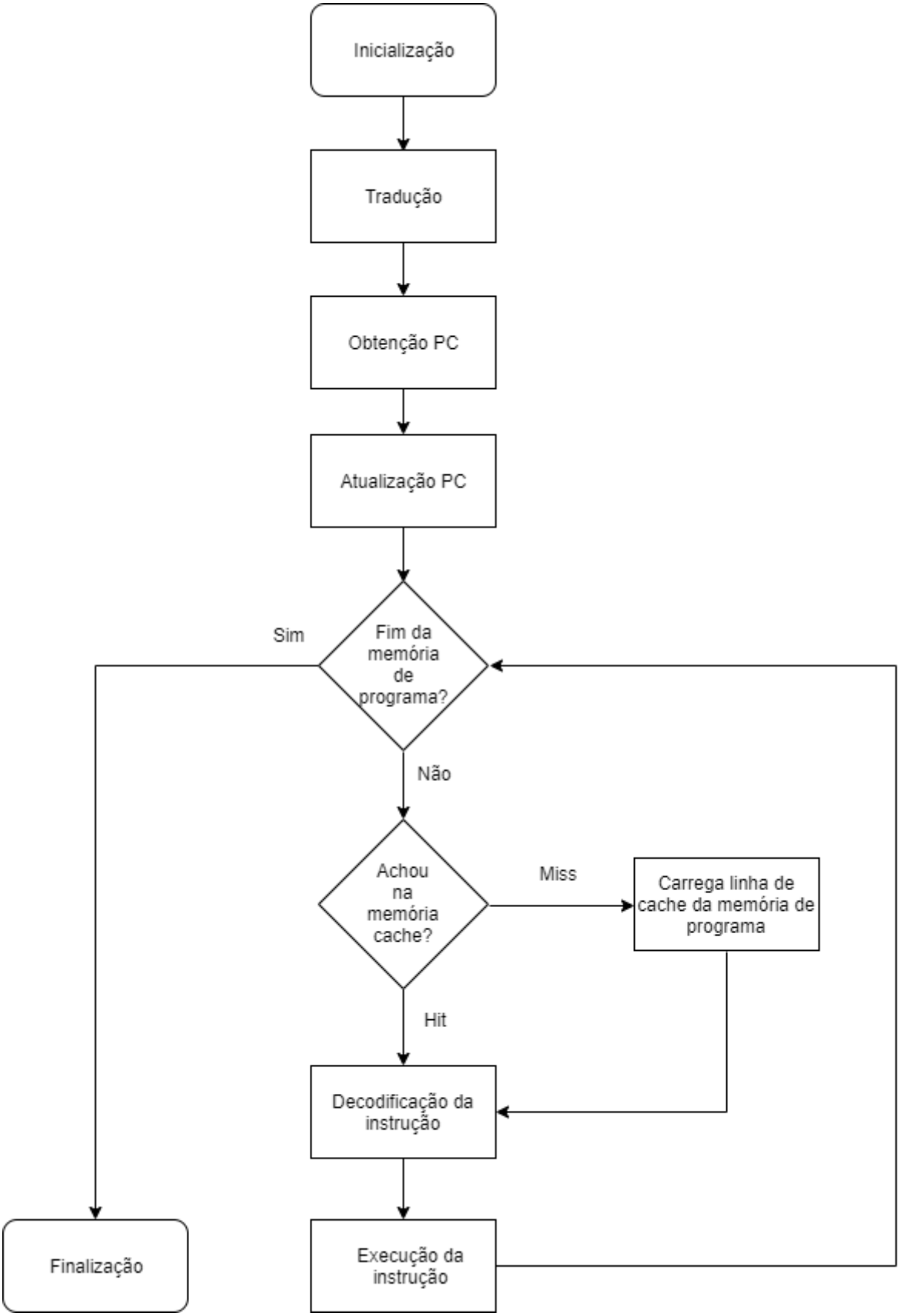
Com a instrução recuperada, o método em Python correspondente é chamada com base no op-code da instrução (isto é, primeiros 4 bits). No método, a instrução é primeiro decodificada e em seguida a operação análoga é executada.

Por exemplo, caso a operação add seja recuperada (op-code 0), o registrador de destino é obtido pelos 3 bits seguintes ao op-code, o registrador fonte 1 pelos próximos 3 bits e o registrador fonte 2 pelos últimos 3 bits. Na operação, soma-se o valor do registrador fonte 1 e 2 e armazena-se no registrador fonte.

Por último, compara-se o valor do PC com o tamanho da memória de programa: caso seja maior, o programa foi concluído.



7 FLUXOGRAMA



8 CONCLUSÃO

Com a máquina virtual desenvolvida, foi possível comprovar os conhecimentos adquiridos no laboratório de C208 e nas aulas teóricas.

Ao implementar uma memória cache, pode-se ver na prática como funciona uma política de leitura *look-through*.

Foi visto e comprovado o ciclo de *fetch*, decodificação e execução de uma instrução, bem como o funcionamento geral de uma arquitetura de computador convencional.

Para futuras modificações na máquina virtual, a implementação de mais níveis de cache seria interessante para aprofundar os conhecimentos em memória cache.

9 BIBLIOGRAFIA

Documentação de Python. Disponível em: <https://docs.python.org/3.7/>. Acesso em: 26 nov. 2019.

Cache Memory in Computador Organization. Disponível em: <https://www.geeksforgeeks.org/cache-memory-in-computer-organization/>. Acesso em: 26 nov. 2019.

Object-Oriented Programming (OOP) in Python 3. Disponível em: <https://realpython.com/python3-object-oriented-programming/>. Acesso em: 26 nov. 2019.

10 APÊNDICE

https://github.com/henriqueblang/virtual-machine-inassembly