Trabalho de Arquiteturas de Computador – EC 208

Alunos: Bruno Balestra de Carvalho Ferreira

Felipe de Cássio Rocha Santos

Matheus de Oliveira e Silva Lourenço Lage

Data de Entrega:29/06/2018

OBJETIVO

Implementar uma máquina virtual a partir de um interpretador de instruções que possua memória cache. O código do interpretador deverá ser implementado em linguagem de alto nível.

ESPECIFICAÇÃO

Nossa arquitetura possui 16 bits de instrução, divididos em dois tipos de instruções: Add, Sub e Load, Store.

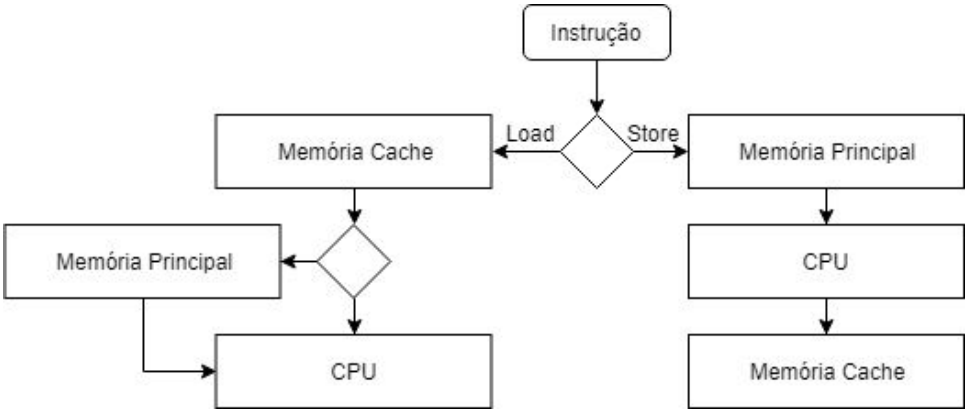
As instruções Add e Sub são divididas da seguinte forma:

|Operação(2bits)|Registrador1(4bits)|Registrador2(4bits)|EndRegistradorDest(6bits)|

e as instruções de Load e Store estão divididas a seguir:
|Operação(2bits)|Registrador(4bits)|EndereçoMemoriaDados(10bits)|

O programa está salvo no arquivo intrucoes.txt e o acesso de memória através do arquivo memoria.txt e da cache. O programa faz o acesso a memória cache antes de buscar na memória principal. A memória cache é feita a partir de uma Struct da seguinte forma:

|Validação(1bit)|TAG(5bits)|Dados|



Nossa cache possui 9 posições e inicializa com seus campos vazios, são preenchidos no primeiro

comando, baseado nas posições próximas do dado utilizado.

As instruções utilizadas são as seguintes, e a cache se movimenta da seguinte forma:

A cache é atualizada com referência do endereço inicial (TAG = [00000]), portanto ela atualiza a posição 0 a posição 7 da memória de dados.

0000000000001010 - Load no Registrador 0 da Memória na posição 10
Executando instrução [0] do tipo... load
Tentando buscar dados na cache (TAG = [01010])... MISS!

Atualizando cache (TAG de referência = [01010])...
TAG inicial = [00111] TAG final = [01110]
Cache atualizada.

Retornando CONTEÚDO = [8]

0000010000001011 – Load no Registrador 1 da Memória na posição 11
Executando instrução [1] do tipo... load
Tentando buscar dados na cache (TAG = [01011])... HIT!
Retornando CONTEÚDO = [6]

0001000000001100 – Load no Registrador 4 da Memória na posição 12
Executando instrução [2] do tipo... load
Tentando buscar dados na cache (TAG = [01100])... HIT!
Retornando CONTEÚDO = [4]

1000000001000010 – Add do Registrador 0 com 1 salvo no registrador 2
Executando instrução [3] do tipo... soma

1100100100000011 – Sub do Registrador 2 com 4 salvo no registrador 3
Executando instrução [4] do tipo... subtração

0100100000001010 – Store do Registrador 2 na Memória 10
Executando instrução [5] do tipo... store
Salvando dados na memória (TAG = [01010])... salvo

0100110000000001 – Store do Registrador 3 na Memória 1
Executando instrução [6] do tipo... store
Salvando dados na memória (TAG = [00001])... salvo

0000000000001010 - Load no Registrador 0 da Memória na posição 10
Executando instrução [7] do tipo... load
Tentando buscar dados na cache (TAG = [01010])... MISS!

Atualizando cache (TAG de referência = [01010])...
TAG inicial = [00111] TAG final = [01110]
Cache atualizada.

Retornando CONTEÚDO = [14]

0000000000001010 - Load no Registrador 0 da Memória na posição 10
Executando instrução [8] do tipo... load
Tentando buscar dados na cache (TAG = [01010])... HIT!
Retornando CONTEÚDO = [14]

As duas últimas instruções servem como exemplo para a verificação de HIT e MISS da cache de acordo com o bit de validação, pois após fazer o Store na posição 10 da memória, a cache passa a estar fora de sincronia com a memória principal (Bit de Validação = false)

Repositório GitHub: https://github.com/felipecassiors/high-level-interpreter