André Fillipi de Góes Silva
João Gustavo Rogel de Oliveira

Arquitetura xAmpa

Santa Rita do Sapucaí

Novembro de 2019

Sumário

Introdução

Este documento foi elaborado como parte do projeto final da disciplina Arquiteturas de Computadores, a fim de esclarecer detalhes sobre o funcionamento de uma nova arquitetura computacional proposta e desenvolvida a partir dos conhecimentos adquiridos nas aulas.

O desenvolvimento da arquitetura foi baseado em arquiteturas existentes, como a MIPS, e teve um caráter essencialmente prático. O projeto contou com a programação de uma máquina virtual capaz de ler arquivos no formato definido (no caso deste projeto, arquivos *.xampa) e realizar a execução baseada nas instruções definidas previamente.

O principal objetivo do projeto é mostrar, de forma simplificada, como é o processo de desenvovimento de uma nova arquitetura de computadores, bem como mostrar quais devem ser as características consideradas durante o planejamento e a melhor forma de otimizar a solução. Além disso, pode-se colocar em prática conhecimentos que até então tinham sido estudados apenas de forma teórica.

1 Descrição Geral

A arquitetura xAmpa (Arquitetura mãozinha pra cima) é uma arquitetura computacional de 12 bits, composta inicialmente por 8 registradores e 7 instruções. Seu uso principal é voltado para operações simples entre registradores ou entre um registrador e uma constante (valor imediato ou endereço), como operações de soma, multiplicação, carregamento de valores em um registrador e leitura da memória de dados. A máquina virtual responsável pela execução dos comandos foi desenvolvida na linguagem de programação Python.

1.1 Registradores

A arquitetura possui 8 registradores, sendo 2 deles de uso específico e os outros 6 de uso geral. A tabela a seguir apresenta os registradores e seu tipo:

Registrador	**Tipo**	**Descrição**
a0	Específico	Program Counter
a1	Específico	Zero imediato
a2	Geral	Dados gerais
a3	Geral	Dados gerais
a4	Geral	Dados gerais
a5	Geral	Dados gerais
a6	Geral	Dados gerais
a7	Geral	Dados gerais

Tabela 1 – Registradores disponíveis

1.2 Instruções

A arquitetura possui, a princípio, apenas 7 instruções. As instruções do tipo R permitem operações entre dois registradores, enquanto as instruções do tipo I permitem operações entre um registrador e uma constante. A tabela a seguir mostra o nome das instruções, seu tipo e uma breve descrição de seu funcionamento:

Nome	**Tipo**	**Descrição**
SPC	I	Armazena uma constante em um registrador.
TCHAN	R	Move o valor de um registrador para outro.
ARAKETU	R	Realiza o "ou-exclusivo"entre dois registradores.
PSIRICO	R	Realiza a soma entre dois registradores.
EVA	I	Realiza a multiplicação entre um registrador e uma constante.
ASA	I	Salva um registrador na memória de dados.
IVETE	I	Carrega um valor da memória de dados.

Tabela 2 – Instruções disponíveis

1.3 Memória Cache

A memória cache utilizada na arquitetura é uma memória de instruções, que permite uma velocidade maior na consulta das instruções por dispensar a memória principal caso a instrução desejada esteja salva na memória cache. A cache guarda o endereço da instrução em 4 blocos. Além disso, cada bloco é composto de dois campos: um deles armazena a instrução, enquanto o outro registra a tag da instrução (seu endereço na memória principal).

O ciclo de busca da instrução funciona da seguinte forma: após a identificação da instrução, a máquina virtual acessa a memória cache e checa se a instrução se encontra lá. Em caso afirmativo, a instrução é executada, e em caso negativo, a memória principal é consultada e um bloco com 4 instruções (incluindo a instrução desejada) é carregado na memória cache.

2 Principais Componentes

2.1 Registradores

A arquitetura xAmpa possui 8 registradores, sendo 2 deles de uso específico e os outros 6 de uso geral. Os dois registradores de uso específico disponíveis são o a0 e o a1, sendo o primeiro responsável por registrar o Program Counter e o segundo por possuir o zero imediato. Os outros registradores são de uso geral, para quaisquer dados que sejam armazenados.

A estrutura dos registradores consiste em um dicionário, cujas chaves são os nomes dos registradores e os valores são variáveis inteiras que armazenam os valores da forma decimal. Além disso, um dicionário auxiliar possui as informações de conversão do nome do registrador para seu identificador no sistema binário.

A tabela a seguir apresenta os identificadores de cada registrador, junto com os dados apresentados anteriormente:

Registrador	**Tipo**	**Identificador**	**Descrição**
a0	Específico	000	Program Counter
a1	Específico	001	Zero imediato
a2	Geral	010	Dados gerais
a3	Geral	011	Dados gerais
a4	Geral	100	Dados gerais
a5	Geral	101	Dados gerais
a6	Geral	110	Dados gerais
a7	Geral	111	Dados gerais

Tabela 3 – Detalhes dos registradores

2.2 Instruções

A arquitetura possui 7 instruções, responsáveis por operações de registro, carregamento, modificação, soma e multiplicação. As instruções do tipo R permitem operações entre dois registradores, enquanto as instruções do tipo I permitem operações entre um registrador e uma constante.

A tabela a seguir mostra a estrutura das instruções do tipo I e como deve ser feita a conversão entre a linha de comando em xAmpa e a palavra binária que define a instrução:

op-code	**Rd**	**Imm/Adress**
op-code - 3 bits	Registrador destino - 3 bits	Constante ou Endereço - 6 bits

Tabela 4 – Instruções do tipo I

A próxima tabela mostra a estrutura das instruções do tipo R e como deve ser feita a conversão entre a linha de comando em xAmpa e a palavra binária que define a instrução:

op-code	**Rd**	**Rs**	**Rt**
op-code - 3 bits	Registrador destino - 3 bits	Registrador 1 - 3 bits	Registrador 2 - 3 bits

Tabela 5 – Instruções do tipo R

Na máquina virtual, as instruções estão salvas em um dicionário, cujas chaves são o nome da própria instrução, e o valor é uma lista que possui como primeiro elemento o op-code da instrução e como segundo elemento o tipo da instrução. A tabela a seguir apresenta os op-codes de cada instrução, junto com as informações apresentadas anteriormente:

Nome	**OP-CODE**	**Tipo**	**Descrição**
SPC	000	I	Armazena uma constante em um registrador.
TCHAN	001	R	Move o valor de um registrador para outro.
ARAKETU	010	R	Realiza o "ou-exclusivo"entre dois registradores.
PSIRICO	011	R	Realiza a soma entre dois registradores.
EVA	100	I	Realiza a multiplicação entre um registrador e uma constante.
ASA	101	I	Salva um registrador na memória de dados.
IVETE	110	I	Carrega um valor da memória de dados.

Tabela 6 – Detalhes das instruções

2.3 Memória Cache

A máquina virtual possui uma memória cache de instruções, responsável por facilitar o ciclo de busca durante a execução do código e melhorar o desempenho. A cache guarda as instruções em blocos, na máquina virtual foi implementada uma cache com 4 blocos. Além disso, para cada bloco existem mais dois campos: um deles armazena a instrução, enquanto o outro registra a tag da instrução (seu endereço na memória principal).

O mapeamento utilizado na implementação da cache é o totalmente associativo, sendo assim cada bloco possui um lugar específico.

A implementação da memória cache foi baseada em duas classes, uma delas responsável por definir o armazenamento e outra responsável pelos blocos de controle. A classe Cache também é responsável pelas funções de busca e carregamento das informações.

3 Funcionamento

3.1 Ferramentas

O desenvolvimento do projeto contou com o uso da linguagem de programação Python em conjunto com a IDE Microsoft Visual Studio Code.

3.2 Execução da Máquina Virtual

O código da máquina virtual é dividido em várias funções, cada uma responsável por uma das etapas da execução. No início do processo, é solicitado ao usuário o caminho do arquivo e se existe a necessidade de mostrar o debug. Após isso, a memória de dados é iniciada e o ciclo de busca se inicia.

No ciclo de busca, o programa é lido e os comandos são separados de acordo com o espaçamento do programa. Em seguida, as palavras binárias para a execução são montadas através de consultas aos registradores, que indicam qual é o código binário correspondente a cada um, bem como o op-code de cada comando, e a memória de programa é atualizada antes do retorno à função principal.

Após a busca, o ciclo de execução é iniciado, a memória cache é configurada e o Program Counter é inicializado com o primeiro endereço para execução. Enquanto houverem dados na memória de programa, o Program Counter é atualizado e a instrução é executada. A palavra binária é dividida em partes menores, correspondentes a cada informação presente na linha de código, e uma série de comandos if-else é responsável por identificar qual é a operação que deve ser realizada. Essa busca da instrução é feita primeiramente na memória cache. Se a instrução desejada estiver carregada (cache hit), ela é executada a partir dali, mas em caso negativo (cache miss), a instrução é buscada na memória principal e carregada na memória cache. Quando ela é identificada, o código identifica os registradores e/ou constantes que serão utilizados e lê os valores que estão salvos. Por fim, a máquina virtual realiza de fato a operação, alterando os registradores necessários.

Caso o usuário opte por exibir as ferramentas de debug, elas serão mostradas ao final da execução de cada linha, mostrando o estado atual de todos os registradores. Ao final da execução do programa, os registradores são apresentados novamente, indicando qual foi o estado final do processo.

3.3 Fluxograma

O fluxograma a seguir mostra a execução da máquina virtual, a fim de tornar mais simples a visualização do processo:

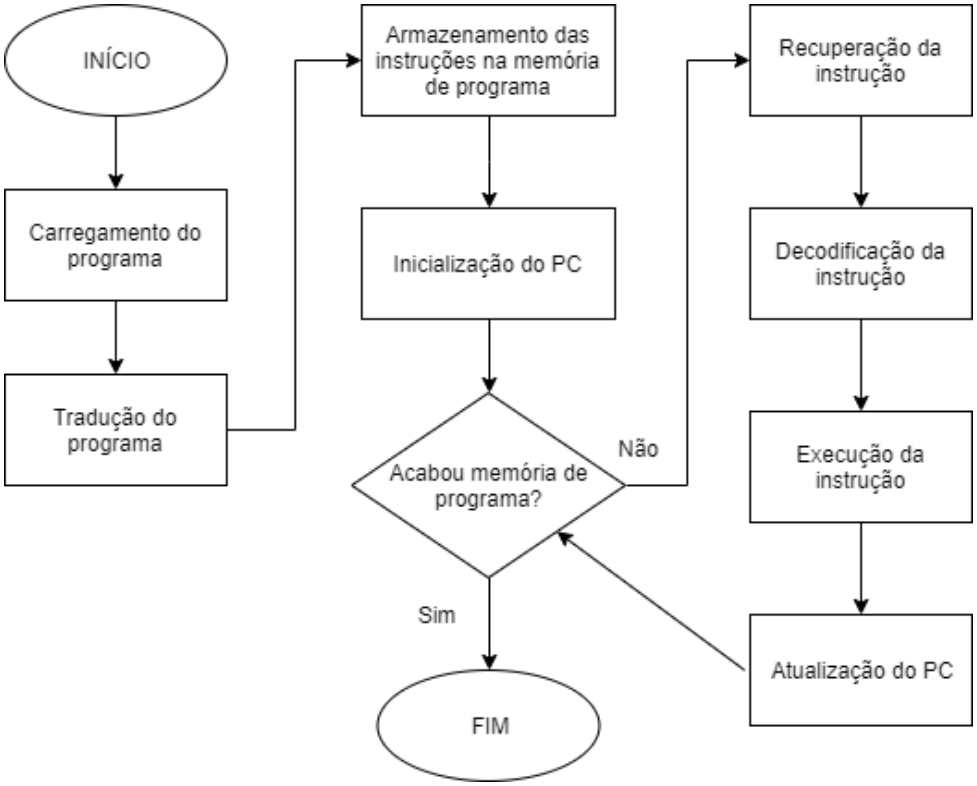

Figura 1 – Fluxograma do ciclo de execução

Conclusão

Durante a execução do projeto final da disciplina de Arquiteturas de Computadores, foi possível ter uma visão nova de como uma linguagem de programação é executada atráves do desenvolvimento de uma máquina virtual capaz de interpretar um código, e não apenas escrever esse código com uma linguagem de baixo nível. Desta forma, o conhecimento teórico foi posto em prática, e foi possível compreender de forma satisfatória como fatores relacionados ao tamanho da palavra da arquitetura ou a forma como as instruções são implementadas alteram a complexidade da implementação dessa arquitetura.

A implementação foi realizada com sucesso para as primeiras instruções, e existe uma possibilidade de expansão dos comandos caso seja necessário ou desejado no futuro. Além disso, as instruções disponíveis já permitem que programas simples possam ser executados corretamente, mostrando que o projeto pode sim ser utilizado no mundo real, com as adaptações necessárias.

Por fim, é possível dizer que a experiência adquirida com este projeto é de grande importância para a vida acadêmica e profissional dos alunos, por trazer um problema do mundo real e propor uma solução eficaz e capaz de resolvê-lo.

ANEXO A – Código Fonte da Máquina Virtual

O código fonte com a máquina virtual se encontra no GitHub, que pode ser acessado com o link <https://github.com/Orange-Caramel/xAmpa>.