Escola Politécnica da Universidade de São Paulo
PCS 3216 – Sistemas de Programação

Simulador de Máquina de Von Neumann, Loader e Montador

Bernardo Marcelino do Nascimento – 9836197

Professor João José Neto

São Paulo

2020

Sumário

1. Introdução

O objetivo deste projeto é construir uma máquina de Von Neumann e seus mecanismos relacionados, como montador e loader, para aprofundar o entendimento de sistemas de programação e do funcionamento de computadores no geral.

O projeto se baseia na estrutura definida na Figura 1, que mostra os 3 componentes principais: o montador, o loader e a máquina de Von Neumann (MVN). O montador é o responsável por pegar um código escrito em linguagem Assembly e transformá-lo em um código binário compatível com a máquina. O loader então é o responsável por carregar esse código binário na memória da máquina, e iniciar a execução do programa, feita pela MVN.

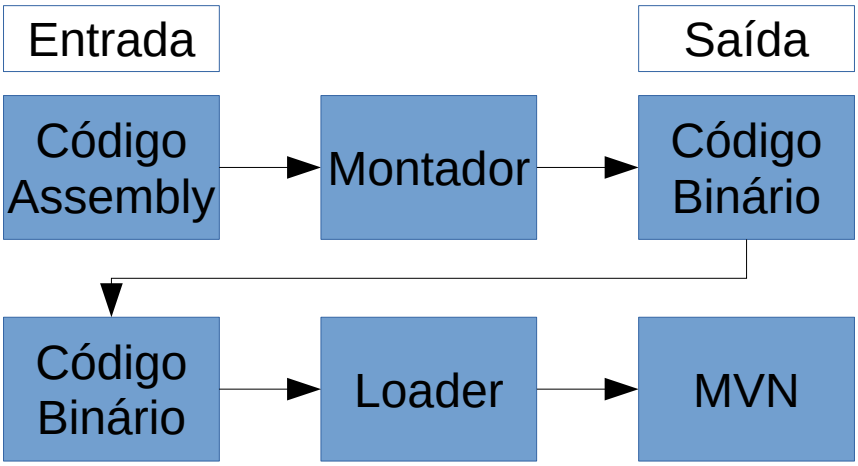

Figura 1 – Visão geral do projeto

A implementação da MVN foi feita na linguagem Python, versão 3.7.3, e toda a interação com o programa é feita através do terminal. O código está disponível em https://sites.google.com/usp.br/2020-pcs3216-9836197/ e em https://github.com/bmnascimento/von-neumann-simulator.

A MVN foi implementada no arquivo vm.py e para utilizar a máquina, basta chamar o programa através do terminal com o comando "python3 vm.py <arquivo_hexa>" onde arquivo_hexa é o nome do arquivo em hexadecimal.

O loader é carregado automaticamente na memória da MVN dentro do arquivo vm.py, mas seu código está descrito no arquivo loader.asm para melhor entendimento.

O montador foi implementado no arquivo montador.py e sua utilização é pelo comando "python3 montador.py <arquivo_input> <arquivo_output>" onde arquivo_input é um arquivo em Assembly e arquivo_output é o nome do arquivo em hexadecimal gerado pelo montador.

2. Máquina de Von Neumann

2.1. Definição

A máquina de Von Neumman é um modelo de máquina capaz de executar instruções e de realizar computações. A máquina consiste de uma memória, onde são armazenados os programas a serem executados assim como os dados que os programas utilizam. Além disso, também há uma unidade que decodifica as instruções e as executa e por fim há os registradores, que auxiliam a execução das instruções.

O funcionamento da máquina consiste em ler um endereço de memória, executar a instrução lida, ler o próximo endereço e executar a instrução e assim por diante, até atingir uma instrução de HALT.

2.2. Motor de Eventos

No projeto, será feita uma simulação de uma MVN e para isso será o utilizado o conceito de motor de eventos. O motor de eventos está esquematizado na Figura 2 e consiste em um loop que extrai uma instrução da memória, decodifica essa instrução e a executa de acordo com o procedimento específico, altera o ponteiro para a próxima instrução e repete o loop.

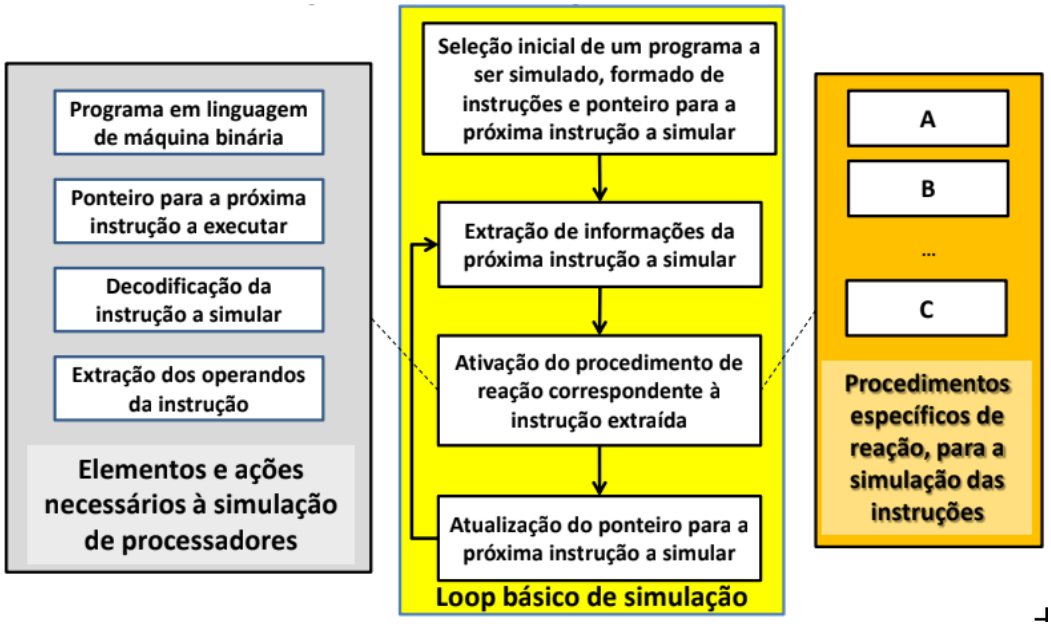

Figura 2 – Loop básico de simulação

2.3. Especificação

A MVN do projeto consiste em uma memória principal de 8 bits com 12 bits de endereçamento (4k x 8 bits), um contador de instruções de 12 bits e um acumulador de 8

bits. Além das memórias são especificadas as instruções que a máquina deve ser capaz de executar, apresentadas na Figura 3. O formato que a instrução deve assumir na memória é 4 bits para o código de operação e 12 bits para o operando, totalizando 16 bits. Portanto, cada instrução ocupa duas posições de memória.

```
; MNEMÔNICOS    CÓDIGO    INSTRUÇÃO / PSEUDO-INSTRUÇÃO
;
;    JP    J     /0xxx     JUMP INCONDICIONAL
;    JZ    Z     /1xxx     JUMP IF ZERO
;    JN    N     /2xxx     JUMP IF NEGATIVE
;    LV    V     /3xxx     LOAD VALUE
;    +     +     /4xxx     ADD
;    -     -     /5xxx     SUBTRACT
;    *     *     /6xxx     MULTIPLY
;    /     /     /7xxx     DIVIDE
;    LD    L     /8xxx     LOAD FROM MEMORY
;    MM    M     /9xxx     MOVE TO MEMORY
;    SC    S     /Axxx     SUBROUTINE CALL
;    RS    R     /Bxxx     RETURN FROM SUBROUTINE
;    HM    H     /Cxxx     HALT MACHINE
;    GD    G     /Dxxx     GET DATA
;    PD    P     /Exxx     PUT DATA
;    OS    O     /Fxxx     OPERATING SYSTEM CALL
;
;    @     @               ORIGIN
;    #     #               END
;    K     K               CONSTANT
```

Aula 07 - Exemplo de um simulador simples 29

Figura 3 – Instruções da MVN

O funcionamento da máquina e a implementação de cada instrução está descrita na Figura 4. Primeiramente o programa é carregado na memória simulada, então o contador de instruções é ajustado para apontar para a instrução inicial, em seguida é feita a extração do código de operação e do operando da instrução, e por fim é feito o tratamento adequado àquela instrução e o loop recomeça.

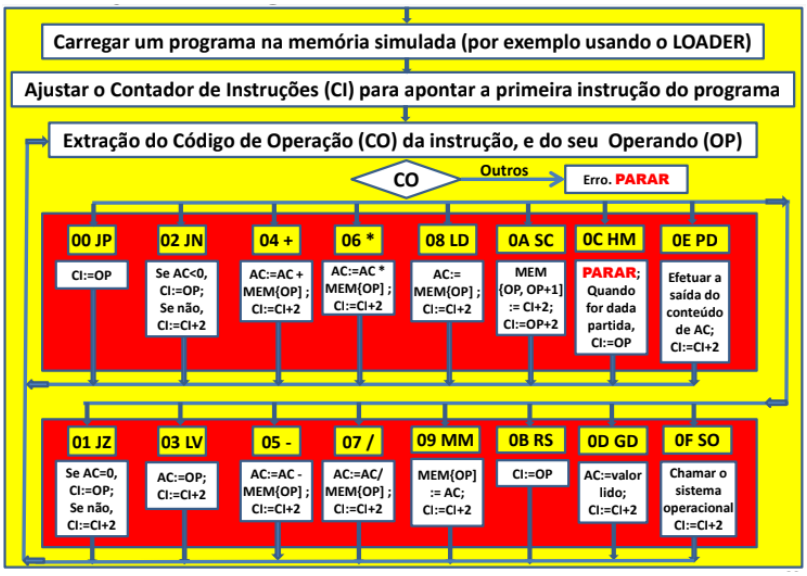

Figura 4 – Descrição das Instruções

2.4. Implementação

Na implementação da máquina, primeiramente é chamada a função boot, que vai inicializar o contador de instruções (CI), o acumulador (AC) e a memória (MEM). Logo em seguida é descrito o loop principal do motor de eventos, que executa infinitamente, primeiro chamando a função executeInstruction que vai decodificar e executar a instrução e se houver um erro ou um comando de halt, o loop é finalizado.

```
CI, AC, MEM = boot(code, debug)
while True:
      CI, AC, status = executeInstruction(CI, AC, MEM, debug)
      if status == 'halt':
            print('\nMáquina parada')
            break
      elif status == 'error':
            print('\nErro: Instrução desconhecida')
            break
```

É na função executeInstruction que acontece o trabalho de acessar, decodificar e executar a instrução. Primeiramente ocorre a decodificação da instrução, o código de operação (CO) consiste nos 4 bits mais significativos do primeiro endereço de memória e o operando são os 4 bits menos significativos do primeiro endereço concatenados com os 8 bits do segundo endereço. Essa operação foi feita nas linhas abaixo.

```
CO = MEM[CI] >> 4
OP = ((MEM[CI] % 0x10) << 8) + MEM[CI+1]
```

Com o código de operação e o operando decodificados, a simulação vai para um if que executa a instrução de acordo com seu CO, da forma que foi descrita na Figura 4. Aqui o código inteiro foi omitido por ser muito extenso.

```
if CO == 0x0: # JP
      CI = OP
elif CO == 0x1: # JZ
      if AC == 0:
            CI = OP
      else:
            CI += 2
...
```

Após executar a instrução, a máquina retorna os novos valores do contador de instruções, do acumulador, limitado a 8 bits em caso de overflow, e do status da máquina (erro ou halt).

```
return (CI, AC % 0x100, status)
```

3. Loader

3.1. Definição

O loader é um programa que é pré-carregado na memória da MVN, como mostra a Figura X e é o responsável por ler o código armazenado em um meio externo, que no caso da simulação aqui descrita é um arquivo em hexadecimal, análogo a uma fita perfurada.

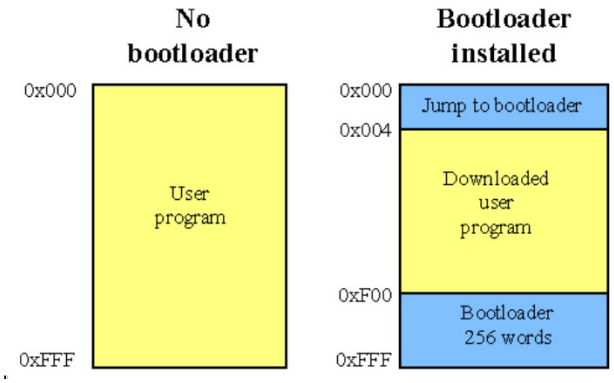

Figura X – Loader na memória

A lógica de funcionamento de um loader demonstrada na Figura X consiste em ler um byte do arquivo, guardar esse byte no local apropriado na memória da MVN, decrementar o ponteiro que indica o tamanho do arquivo, checar se é o último byte, se for deve mudar o contador de instruções para apontar para o programa carregado, caso contrário recomeça o loop lendo outro byte.

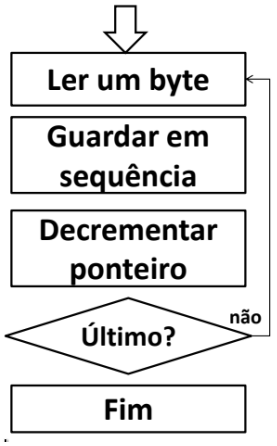

Figura X – Loader na memória

3.2. Especificação

O loader do projeto foi feito em linguagem de baixo nível e roda dentro da própria máquina virtual. Na MVN, a posição 0 de memória contém uma instrução de jump

incondicional para a posição F00, como mostra a Figura X, e o código do loader em si se encontra nos endereços de F00 a FFF.

3.3. Implementação

Primeiramente, o código indica o jump na posição 0 de memória e o resto do código se encontra da posição F00 em diante.

```
@  0
JP F00
```