PUCRS - Escola Politécnica
Disciplina: Sistemas Operacionais - 2020/2 - Trabalho Prático
Prof. Fernando Luís Dotti

1. Definição da Máquina Virtual (MV)

Nossa máquina virtual (MV) tem CPU e Memória.

1.1 CPU

O processador possui os seguintes registradores:

- um contador de instruções (PC)
- Um registrados de instruções (IR)
- oito registradores, 0 a 7

O conjunto de instruções é apresentado na tabela a seguir, adaptado de [1].
A tabela atual está revisada para não ter mais operações a nível de bit. As colunas em vermelho substituem a codificação em bits para designar os registradores e parâmetros utilizados, e compõem os "campos" de uma posicaoDeMemoria vide 1.2.

No.	Mnemonic / OPCODE	Description	Syntax	Micro- operation	REG	REG	PAR
J - Type Instructions							
1	JMP	Direct Jump	JMP k	PC ← k			
2	JMPI		JMPI Rs	PC ← Rs	Rs		k
3	JMPIG		JMPIG Rs, Rc	if Rc > 0 then PC ← Rs Else PC ← PC +1	Rs	Rc	
4	JMPIL		JMPIL Rs, Rc	if Rc < 0 then PC ← Rs Else PC ← PC +1	Rs	Rc	
5	JMPIE		JMPIE Rs, Rc	if Rc = 0 then PC ← Rs Else PC ← PC +1	Rs	Rc	
I - Type Instructions							
6	ADDI	Immediate addition	ADDI Rd, k	Rd ← Rd + k	Rd		k
7	SUBI	Immediate subtraction	SUBI Rd, k	Rd ← Rd – k	Rd		k
8	ANDI	Immediate AND	ANDI Rd, k	Rd ←Rd AND k	Rd		k
9	ORI	Immediate OR	ORI Rd, k	Rd ←Rd OR k	Rd		k
10	LDI	Load immediate	LDI Rd, k	Rd ← k	Rd		k
11	LDD	Load direct from data memory	LDD Rd,[A]	Rd ← [A]	Rd		A
12	STD	Store direct to data memory	STD [A],Rs	[A] ← Rs	Rs		A
R2 – Type Instructions							
13	ADD	Addition	ADD Rd, Rs	Rd ← Rd + Rs	Rd	Rs	
14	SUB	Subtraction	SUB Rd, Rs	Rd ← Rd - Rs	Rd	Rs	
15	MULT	multiplication	MULT Rd, Rs	Rd ← Rd * Rs	Rd	Rs	
17	LDX	Indirect load from memory	LDX Rd,[Rs]	Rd ← [Rs]	Rd	Rs	
18	STX	Indirect storage to memory	STX [Rd],Rs	[Rd] ←Rs	Rd	Rs	
R1 – Type Instructions							
22	SWAP	Swap nibbles	SWAP	Rd7←Rd3, Rd6←Rd2, Rd5←Rd1, Rd4←Rd0	Rd		
23	STOP						

1.2 Memória

Considere a memória como um array contíguo de posições de memória. Cada posição de memória são 4 bytes. A memória tem 1024 posições.

```
tamMemoria = 1024
array mem[tamMemoria] of posicaoDeMemoria
```

Cada posiçãoDeMemória codifica **[** OPCODE, 1 REG de 1..8, 1 REG de 1..8, PARAMETRO K OU A conforme operação **]**. Em um sistema real estes dados são codificados em bits de uma palavra. No nosso trabalho, podemos adotar que uma posicaoDeMemoria é um registro (objeto) com estes atributos. Note que no caso da posição de memória não ter uma instrução, temos que adotar então uma codificação própria. Neste caso podemos ter um OPCODE especial para significar uma posição de dados, e, no campo de K ou A, temos um valor inteiro. Um valor inteiro é suficiente pois a nossa arquitetura manipulará inteiros apenas.

1.3 Funcionamento da CPU: o ciclo de instruções

A CPU executa o ciclo de instruções. Dado um valor no PC, ela faz:

```
Loop
    busca a posição de memória apontada por PC, carrega no RI
    Decodifica o RI
        se RI inválido - sinaliza interrupção, acaba esta instrução
        executa operação
                Se erro: sinaliza interrupcao, acaba esta instrução
                conforme operação efetua a mesma e atualiza PC
    se existe interrupção
        desvia para rotina de tratamento
fimLoop
```

Um vetor de interrupções associa o código da interrupção com a rotina a ser executada.

1.4 Programas

Neste momento não temos um Sistema Operacional. Para fazer a Máquina Virtual funcionar, você deve carregar um programa no início da memória, atribuir um valor ao PC (o início do código do seu programa), e liberar a CPU para executar. A CPU vai executar até parar, encontrar um erro, ou então vai entrar em loop conforme o programa estiver correto ou não.

Você deve criar formas de acompanhar esta evolução.

Nossos programas podem ser escritos em TXT e lidos para a memória, ou então eles podem ser codificados em Java como a criação de um vetor de posicaoDeMemoria inicializado em cada posição do vetor como uma linha do programa.

A seguir o programa P1, que escreve em posições sabidas de memória os 10 números da sequência de Fibonacci. Ou seja, ao final do programa a memória tem estes 10 números em posições convencionadas no programa. Avalie se P1 está correto.

Como parte do exercício, construa P2, um programa que le um valor de uma determinada posição (carregada no inicio),

```
se o número for menor que zero coloca -1 no início da posição de memória para saída;
se for maior que zero este é o número de valores
    da sequencia de fibonacci a  serem escritos em sequencia a partir de uma posição de
    memoria;
```

Outros programas

```
P3:  dado um inteiro em alguma posição de memória,
        se for negativo armazena  -1 na saída;
        se for positivo responde o fatorial do número na saída

P4:  para um N definido (5 por exemplo)
        o programa ordena um vetor de N números em alguma posição de memória;
        ordena usando bubble sort
                loop ate que não swap nada
                        passando pelos N valores
                        faz swap de vizinhos se da esquerda maior que da direita
```

Possível solução para P1 (página a seguir):

P1: Verifique se funciona!

```
0    LDI  R1, 0          // 1ro valor
1    STD[50], R1         // de 50 a 60 estarao os numeros de fibonacci
2    LDI R2, 1           // 2o valor da sequencia
3    STD[51], R2
4    LDI R8, 52          // proximo endereco a armazenar proximo numero
5    LDI R6, 6           // 6   é proxima posição de mem (para pular para ela depois)
6    LDI R7, 61          // final
7    LDI R3, 0
8    ADD R3, R1          // R3 =+R2
9    LDI R1,0
10   ADD R1, R2
11   ADD R2, R3
12   STX R8, R2
13   ADD R8, 1
14   SUB R7, R8
15   JMPIG R6, R7
16   STOP
17
…
50  DATA   0
51  DATA   1
52  DATA   1
53  DATA   2
54  DATA   3
55  DATA   5
56  DATA   8
57  DATA   13
58  DATA   21
59  DATA   34
60  DATA   55
…
```