PUCRS - Escola Politécnica
Disciplina: Sistemas Operacionais - Trabalho Prático
Hardware Simulado
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 programa (PC - program counter)
- Um registrados de instruções (IR - instruction register)
- Oito registradores, 0 a 7 (R_)

O conjunto de instruções é apresentado na tabela a seguir, adaptado de [1][1]. A máquina manipula somente inteiros. Com isto queremos focar no funcionamento geral (ciclo de instruções, etc), e não na complexidade do que é computado.
Ainda, a tabela de instruções está simplificada para não ter mais operações a nível de bit. As colunas em vermelho substituem a codificação em bits de [1] para designar os registradores e parâmetros utilizados, e compõem os "campos" de uma posicaoDeMemoria vide 1.2. Assim, representamos uma instrução de forma abstrata, com suas partes acessáveis em uma estrutura de dados.

No.	OPCODE	Descrição	Syntax	Micro-operation (significado)	Ra	Rb	P
Instruções JUMP. (primeira posição é para onde pula. Se Rc usado, é sempre segunda posição							
1	JMP	Direct Jump, absoluto	JMP k	PC ← k			k
2	JMPI	Direct com registrador	JMPI Rs	PC ← Rs	Rs		
3	JMPIG	Condicional, com registrador	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	
6	JMPIM	Condicional com memória	JMPIM [A]	PC ← [A]			A
7	JMPIGM		JMPIGM [A], Rc	if Rc > 0 then PC ← [A] Else PC ← PC +1		Rc	A
8	JMPILM		JMPILM [A], Rc	if Rc < 0 then PC ← [A] Else PC ← PC +1		Rc	A
9	JMPIEM		JMPIEM [A], Rc	if Rc = 0 then PC ← [A] Else PC ← PC +1		Rc	A
10	JMPIGT		JMPIGT k,Rc,Rs	If Rs > Rc then PC <- k else PC++	Rs	Rc	K
11	JMPIGK		JMPIGK k, Rc	If RC > 0 then PC <- k else PC++		Rc	K
12	JMPILK		JMPILK k, Rc	If RC < 0 then PC <- k else PC++		Rc	K
13	JMPIEK		JMPIEK k, Rc	If RC = 0 then PC <- k else PC++		Rc	K
14	STOP	Parada do programa					
Instruções Aritméticas							
15	ADDI	Adição Imediata	ADDI Rd, k	Rd ← Rd + k	Rd		k
16	SUBI	Subtração imediata	SUBI Rd, k	Rd ← Rd – k	Rd		k
17	ADD	Addição	ADD Rd, Rs	Rd ← Rd + Rs	Rd	Rs	
18	SUB	Subtração	SUB Rd, Rs	Rd ← Rd - Rs	Rd	Rs	
19	MULT	Multiplicação	MULT Rd, Rs	Rd ← Rd * Rs	Rd	Rs	
Instruções de Movimentação							
20	LDI	Carga imediata	LDI Rd, k	Rd ← k	Rd		k
21	LDD	Carga de memória	LDD Rd,[A]	Rd ← [A]	Rd		A
22	STD	Store em memória	STD [A],Rs	[A] ← Rs	Rs		A
23	LDX	Indirect load from memory	LDX Rd,[Rs]	Rd ← [Rs]	Rd	Rs	
24	STX	Indirect storage to memory	STX [Rd],Rs	[Rd] ←Rs	Rd	Rs	
25	MOVE		MOVE Rd,Rs	Rd ← Rs	Rd	Rs	
26	SYSCALL	Desvia para sistema					

1.2 Memória

Considere a memória como um array contíguo de posições de memória. A memória tem 1024 posições.

tamMemoria = 1024
array mem[tamMemoria] of posicaoDeMemoria

Cada posiçãoDeMemória codifica **[OPCODE**; **R1**: 1 REG de 0..7; **R2**: 1 REG de 0..7, **PARAMETRO**: K ou A conforme OPCODE], K significa constante, A é endereço. Ou seja cada posição tem a estrutura das colunas em vermelho da tabela. Em um sistema real estes dados são codificados em bits de uma palavra. No nosso trabalho, adotamos que uma posicaoDeMemoria é um registro (objeto) com estes atributos. Assim, opcode é um tipo ou enumeração da coluna opcode da tabela. Note que no caso da posição de memória não ter uma instrução, terá um dado (ou não está em utilização). No caso de dado, adotamos um OPCODE "DATA" especial para significar uma posição de dados, e, no campo P de uma instrução de opcode "DATA", temos um valor inteiro como dado. Um valor inteiro é suficiente pois a nossa arquitetura manipulará inteiros apenas. A constante K ou o endereço A usados em outras instruções, também são inteiros e codificadas no campo P.

---

[1] [1] "Design of a General Purpose 8-bit RISC Processor for Computer Architecture Learning". Antonio Hernández Zavala , Oscar Camacho Nieto , Jorge A. Huerta Ruelas , Arodí R. Carvallo Domínguez. Computación y Sistemas, Vol. 19, No. 2, 2015, pp. 371–385 ISSN 1405-5546 doi: 10.13053/CyS-19-2-1941 http://www.scielo.org.mx/pdf/cys/v19n2/v19n2a13.pdf

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
executa operação
atualiza PC conforme operação.
Se STOP, pára
fimLoop

2. Programas

Neste momento não temos um Sistema Operacional. Para fazer a Máquina Virtual funcionar, deve-se carregar um programa no início da memória, atribuir ao PC o início do código do seu programa (0), e liberar a CPU para executar. A CPU vai executar até parar, encontrar um erro, ou então vai entrar em loop caso o programa assim o faça. 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. Veja no código um exemplo.

2.1 Exemplo

EXERCÍCIO: EMULE A EXECUÇÃO DO PROGRAMA
CONSIDERE O MESMO CARREGADO EM MEMÓRIA, E PC=0

FATORIAL

```
                          // linha   comentário
 0  LDI  R0, 4            // 0       r0 é valor a calcular fatorial
 1  LDI  R1, 1            // 1       r1 é 1 para multiplicar (por r0)
 2  LDI  R6, 1            // 2       r6 é 1 para ser o decrementO
 3  LDI  R7, 8            // 3       r7 tem posicao de stop do programa = 8
 4  JMPIE R7, R0          // 4       se r0=0 pula para r7(=8)
 5  MULT  R1, R0          // 5       r1 = r1 * r0
 6  SUB   R0, R6          // 6       decrementa r0 1
 7  JMP  4                // 7       vai p posicao 4
 8  STD R1, 10            // 8       coloca valor de r1 na posição 10
 9  STOP                  // 9      STOP
10  DATA                  // 10      armazena resultado  quando faz stop
```

ESTADO DA CPU

R[0] :

R[1] :

R[2] :

R[3] :

R[4] :

R[5] :

R[6] :

R[7] :

PC: 0

IR:

INSTRUÇÕES UTILIZADAS

No.	OPCODE	Descrição	Syntax	Micro-operation (significado)	Ra	Rb	P
Instruções JUMP. (primeira posição é para onde pula. Se Rc usado, é sempre segunda posição							
1	JMP	Direct Jump, absoluto	JMP k	PC ← k			k
5	JMPIE		JMPIE Rs, Rc	if Rc = 0 then PC ← Rs Else PC ← PC +1	Rs	Rc	
14	STOP	Parada do programa					
Instruções Aritméticas							
18	SUB	Subtração	SUB Rd, Rs	Rd ← Rd - Rs	Rd	Rs	
19	MULT	Multiplicação	MULT Rd, Rs	Rd ← Rd * Rs	Rd	Rs	
Instruções de Movimentação							
20	LDI	Carga imediata	LDI Rd, k	Rd ← k	Rd		k
22	STD	Store em memória	STD [A],Rs	[A] ← Rs	Rs		A

Representação para carga em memória.

```
        public Word[] f = new Word[] {
             // valores -1 indicam campo não usado pela instrução.
                                //Ra  Rb  P          linha   comment
             new Word(Opcode.LDI,   0,-1, 4),      // 0       r0 é valor a calcular f
             new Word(Opcode.LDI,   1,-1, 1),      // 1       r1 é 1 para multiplicar (por r0)
             new Word(Opcode.LDI,   6,-1, 1),      // 2       r6 é 1 para ser o decremento
             new Word(Opcode.LDI,   7,-1, 8),      // 3       r7 tem posicao de stop do programa = 8
             new Word(Opcode.JMPIE, 7, 0, 0),      // 4       se r0=0 pula para r7(=8)
             new Word(Opcode.MULT,  1, 0,-1),      // 5       r1 = r1 * r0
             new Word(Opcode.SUB,   0, 6,-1),      // 6       decrementa r0 1
             new Word(Opcode.JMP,  -1,-1, 4),      // 7       vai p posicao 4
             new Word(Opcode.STD,   1,-1, 10),     // 8       coloca valor de r1 na posição 10
             new Word(Opcode.STOP, -1,-1,-1),      // 9       stop
             new Word(Opcode.DATA, -1,-1,-1) };    // 10      resultado
```

2.2 Exemplo: Fibonacci

A seguir o programa P1. Seu código fica nas posições 0 a 16. Ele escreve nas posições 20 a 29 da memória os primeiros 10 números da sequência de Fibonacci. Ao final, para ver a resposta, deve ser feito um dump da memória. As posições 17 a 19 não são utilizadas. Consideramos todo intervalo de 0 a 29 como imagem do programa. Avalie se P1 está correto.

```
 0  LDI  R1, 0        // 1ro valor
 1  STD[20], R1       // de 20 a 29 estarão números da seq. de fibonacci
 2  LDI R2, 1         // 2o valor da sequencia
 3  STD[21], R2       // dois primeiros valores armazenados em 20 e 21
 4  LDI R8, 22        // proximo endereco a armazenar proximo numero
 5  LDI R6, 6         // 6 é posição de mem do inicio do loop
 6  LDI R7, 31        // final, restaura em R7 o valor final
 7  MOVE R3, R1
 8  MOVE R1, R2
 9  ADD R2, R3     // adiciona em R2 valor de R3 (R2 é o novo valor da serie)
10  STX R8, R2      // coloca o novo valor na posição informada em R8
11  ADDI R8, 1      // R8 tem nova posição a armazenar valor no proximo loop
12  SUB R7, R8      // subtrai R8 de R7 para ver se chegou ao final
13  JMPIG R6, R7  // se R7 for maior que zero jump para R6 (início do loop)
14  STOP            // senão acaba

15  DATA
16  DATA
17  DATA
18  DATA
19  DATA
20  DATA
21  DATA
22  DATA
23  DATA
24  DATA
25  DATA
26  DATA
27  DATA
28  DATA
29  DATA
```

```
20  DATA
21  DATA
22  DATA
23  DATA
24  DATA
25  DATA
26  DATA
27  DATA
28  DATA
29  DATA
```

ESTADO DA CPU

R[1] :

R[2] :

R[3] :

R[4] :

R[5] :

R[6] :

R[7] :

R[8]:

PC:

IR:

```
 0  LDI  R1, 0        // 1ro valor
 1  STD[20], R1      // de 20 a 29 estarão números da seq. de fibonacci
 2  LDI R2, 1        // 2o valor da sequencia
 3  STD[21], R2      // dois primeiros valores armazenados em 20 e 21
 4  LDI R8, 22       // proximo endereco a armazenar proximo numero
 5  LDI R6, 6        // 6 é posição de mem do inicio do loop
 6  LDI R7, 30       // final, restaura em R7 o valor final
 7  LDI R3, 0        // zera R3
 8  ADD R3, R1       // R3 =+R1, ou seja R3 = R1
 9  LDI R1,0         // zera R1
10   ADD R1, R2      // adiciona R2 em R1, ou seja, R1 <- R2
11   ADD R2, R3      // adiciona em R2 valor de R3 (R2 é o novo valor da serie)
12   STX R8, R2      // coloca o novo valor na posição informada em R8
13   ADDI R8, 1      // R8 tem nova posição a armazenar valor no proximo loop
14   SUB R7, R8      // subtrai R8 de R7 para ver se chegou ao final
15   JMPIG R6, R7    // se R7 for maior que zero jump para R6 (início do loop)
16   STOP            // senão acaba
```

ESTADO DA CPU

R[0] :

R[1] :

R[2] :

R[3] :

R[4] :

R[5] :

R[6] :

R[7] :

PC: 0

IR: