PUCRS - Escola Politécnica
Disciplina: Sistemas Operacionais - 2021/1 - Trabalho Prático - Fase 1
Prof. Fernando Luís Dotti

Trabalho Prático. Primeira parte. Em grupos de até 3 alunos.

O professor fornecerá um programa template de uma máquina virtual (MV) definida abaixo. Você deve:
1) entender e estender este template se necessário, para representar todo conjunto de instruções;
2) rodar programas fornecidos, escritos na linguagem definida;
3) escrever e rodar programas solicitados.

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.

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; 1 REG de 0..7; 1 REG de 0..7, PARAMETRO: K ou A ou DADO conforme OPCODE **]**, ou seja as 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.

1.3 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
```

---

[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

No.	Mnemonic / OPCODE	Descrição	Syntax	Micro-operation (significado)	R1	R2	P
Instruções JUMP							
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	STOP	Parada do programa					
Instruções Aritméticas							
11	ADDI	Adição Imediata	ADDI Rd, k	Rd ← Rd + k	Rd		k
12	SUBI	Subtração imediata	SUBI Rd, k	Rd ← Rd – k	Rd		k
13	ADD	Addição	ADD Rd, Rs	Rd ← Rd + Rs	Rd	Rs	
14	SUB	Subtração	SUB Rd, Rs	Rd ← Rd - Rs	Rd	Rs	
15	MULT	Multiplicação	MULT Rd, Rs	Rd ← Rd * Rs	Rd	Rs	
Instruções de Movimentação							
16	LDI	Carga imediata	LDI Rd, k	Rd ← k	Rd		k
17	LDD	Carga de memória	LDD Rd,[A]	Rd ← [A]	Rd		A
18	STD	Store em memória	STD [A],Rs	[A] ← Rs	Rs		A
19	LDX	Indirect load from memory	LDX Rd,[Rs]	Rd ← [Rs]	Rd	Rs	
20	STX	Indirect storage to memory	STX [Rd],Rs	[Rd] ←Rs	Rd	Rs	
21	SWAP	SWAP regs	SWAP Ra, Rb	T ← Ra Ra ← Rb Rb ←T			

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.

A seguir o programa P1. Seu código fica nas posições 0 a 16. Ele escreve nas posições 50 a 59 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 49 não são utilizadas. Consideramos todo intervalo de 0 a 59 como imagem do programa. Avalie se P1 está correto.

```
0   LDI  R1, 0          // 1ro valor
1   STD[50], R1         // de 50 a 59 estarão números da seq. 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 posição de mem do inicio do loop
6   LDI R7, 60          // 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
17  ...
```

```
17  _
... _
49  _
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  _
```

Como parte do exercício, construa os seguintes programas:

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;

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