UNISINOS - **UNIVERSIDADE DO VALE DO RIO DOS SINOS**
CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS (C6/6) – Curso: Informática

LABORATÓRIO II

Disciplina: Linguagem de Programação PASCAL **E-mail:** ***osorio@exatas.unisinos.br***
Professor responsável: ***Fernando Santos Osório*** **Web:**
Semestre: 2002/2 ***http://www.inf.unisinos.br/~osorio/lab2.html***
Horário: 41 e 51 **Xerox : *Pasta 54* – LAB. II (Xerox do C6/6)**

TRABALHO PRÁTICO 2002/2 – GRAU A

(Versão 1.0 – 02/09/2002)

Faça um programa para "simular" um computador baseado em uma arquitetura tipo RISC. O computador, denominado SUPER-CRIATURA (***SU**per **P**rocessador e **E**xecuto**R** de um **C**onjunto **R**eduzido de **I**nstruções **A**dotado como **T**rabalho na **U**nisinos e **R**ealizado por **A**lunos*), deverá conter uma memória de programa, uma memória de dados, uma pilha de dados, uma pilha de controle, registradores (RegA, entrada e saída) e um registrador de controle de instruções (PC). O computador SUPER-CRIATURA, seus componentes e suas instruções serão detalhados a seguir:

1. Ler o nome de um arquivo fornecido pelo usuário e depois os dados contidos neste arquivo texto, armazenando os dados lidos na memória de programa e na memória de dados. O arquivo possui a seguinte estrutura indicada abaixo (máximo: 40 comandos e 10 dados com valores inteiros).

Linhas do Arquivo:	Comentário:
VAR	=> Define os dados
VAL1 10	=> Variável VAL1:=10
VAL2 15	=> Variável VAL2:=15
PROGRAM	=> Define os comandos
INPA #	=> RegA:=Entrada_Teclado
ADDA VAL1	=> A:=A+VAL1;
OUTA #	=> Saída_Tela:=A;
STOP #	=> Fim da execução

2. A memória de programa e a memória de dados devem ser implementadas como listas seqüenciais, onde abaixo está ilustrado o resultado da leitura do arquivo descrito acima:

MEMÓRIA DE PROGRAMAS

Endereço	Comando	Parâmetro
1	INPA	#
2	ADDA	VAL1
3	OUTA	#
4	STOP	#
...		
40		

MEMÓRIA DE DADOS

Nome da Variável	Valor Armazenado
VAL1	10
VAL2	15
(max. 10 vars.)	

3. Os comandos que poderão ser executados pelo SUPER-CRIATURA são os seguintes:

COMANDO	PARÂMETRO	DESCRIÇÃO	
SETA	<valor>	Atribui um valor inteiro ao Registrador_A	RegA:=Valor
LOAD	<variável>	Copia o valor da variável para o Reg. A	RegA:=Var
SAVE	<variável>	Copia o valor do Reg. A para a variável	Var:=RegA
PUSH	#	Salva o Reg. A no topo da pilha de dados	RegA => Pilha
POPA	#	Lê o topo da pilha de dados e coloca no Reg. A	RegA <= Pilha
INPA	#	Lê o dado digitado da entrada para o Reg. A	RegA <= Entrada
OUTA	#	Coloca o valor do Reg. A no Registrador de Saída	RegA => Saída
INCA	#	Incrementa o valor do Registrador A	RegA:=RegA+1
DECA	#	Decrementa o valor do Registrador A	RegA:=RegA-1
ADDA	<variável>	Soma o valor da variável ao valor do Reg. A	RegA:=RegA+Var
SUBA	<variável>	Subtrai o valor da variável do valor do Reg. A	RegA:=RegA-Var
MULT	<variável>	Multiplica o valor de Reg. A pelo valor da variável	RegA:=RegA*Var
DIVA	<variável>	Divide o valor de Reg. A pelo valor da variável	RegA:=RegA/Var
JUMP	<+/-desvio>	Pula para cima (+) ou para baixo (-) "n" comandos. Desvio incondicional na execução do programa.	JUMP +N JUMP –N
JMPZ	<+/-desvio>	Pula para cima ou para baixo "n" comandos se o Registrador A for igual a zero, caso contrário segue para a instrução seguinte.	A=0? PC:=PC+N A=0? PC:=PC-N PC=Contr. instruc.
JPNZ	<+/-desvio>	Pula para cima ou para baixo "n" comandos se o Registrador A for diferente de zero, caso contrário segue para a instrução seguinte.	Se A<>0 Então PC:=PC+N Senão PC:=PC+1
SUBR	<nro_instr>	Desvia a execução do programa para a subrotina que começa na posição de memória "nro_instr"	PC:=Nro_instrução
RETS	#	Retorna da execução de uma subrotina	PC:=Retorno
NULL	#	Instrução sem efeito (não executa nada)	PC:=PC+1
STOP	#	Para a execução do programa (fim da execução)	STOP

- Indica que este comando não possui um parâmetro (parâmetro pode ser ignorado).
Comandos possuem sempre 4 letras seguidas de um espaço em branco e de seu parâmetro

4. O simulador do SUPER-CRIATURA deve ler o arquivo, colocar o programa e os dados na memória, inicializar a pilha de dados e a pilha de controle (ambas começam vazias), e inicializar o registrador de controle de instruções como valor 1 (PC = endereço da próxima instrução a ser executada). Os registradores A, de entrada e de saída devem também ser inicializados com zero. O programa deve realizar a simulação executando os comandos e incrementando o PC a medida que as instruções são executadas, onde no exemplo abaixo mostramos a execução e a evolução do estados dos registradores:

```
PC=1  REG_A=0  INPUT=0  OUTPUT=0  =>  INPA  #  (usuário digita 7)
PC=2  REG_A=7  INPUT=7  OUTPUT=0  =>  ADDA  VAL1
PC=3  REG_A=17 INPUT=7  OUTPUT=0  =>  OUTA  #
PC=4  REG_A=17 INPUT=7  OUTPUT=17 => STOP  #
```

5. A execução deverá poder ser realizada de 2 formas diferentes, de acordo com a opção do usuário, sendo exibido na tela o estado atual dos diversos componentes do sistema:

   Opção 1: Executar todo o programa até encontrar o comando STOP.

   Opção 2: Executar passo-à-passo o programa. Usar o "readkey" para uma pausa entre comandos.

Tela de Simulação do SUPER-CRIATURA:

Estado dos Registradores			
PC: 10	REG_A: 5	INPUT: 0	OUTPUT: 0

Endereço	Comando	Parâmetro
10	LOAD	CNT
11	ADDA	N1
12	OUTA	#
13	DECA	#
14	JPNZ	-2

Exibir só 5 comandos (os 5 próximos)

Nome da Variável	Valor Armazenado
CNT	10
N1	2
N2	25
N3	-10
N4	12

Exibir só 5 variáveis (as 5 primeiras)

Pilha de Dados		Pilha de Controle	
4	Topo		
6			
-3		8	Topo
1	Base	3	Base

<= Máximo de dados nas pilhas: 10
(Exibir todos os 10 dados)

ATUALIZAR ESTES DADOS APÓS CADA COMANDO!

6. Funcionamento da pilha de dados: A pilha de dados é manipulada exclusivamente pelas instruções PUSH e POPA, que colocam o registrador A no topo da pilha (PUSH) e retiram o conteúdo do topo da pilha para o registrador A (POP). O tamanho da pilha de dados é de no máximo 10 dados.
7. A entrada de dados durante a execução do programa é realizada através da instrução INPA, onde o programa deve parar sua execução, solicitar ao usuário para que digite um dado (nro. inteiro), ler este dado e armazená-lo no registrador de entrada e no registrador A.
8. A saída de dados durante a execução do programa é realizada através da instrução OUTA que permite que o registrador A seja copiado para o registrador de saída.
9. Desvio da ordem de execução das instruções de um programa (laços e subrotinas): a ordem seqüencial da execução das instruções do programa pode ser alterada apenas pelas instruções: JUMP, JMPZ, JPNZ, SUBR ou RETS. As instruções de JUMP (pulo) podem ser do tipo:

   A) Desvio incondicional relativo: desvia a execução para uma instrução seguinte (se o valor do desvio for positivo) ou para uma instrução anterior (se o valor do desvio for negativo). Exemplo:
   JUMP <valor_desvio> onde é realizado PC:=PC + <valor_desvio>
   JUMP 5 => PC:=PC+5 JUMP –4 => PC:=PC+(-4)

   B) Desvio condicional relativo: desvia para uma instrução posterior ou anterior, se a condição for satisfeita, caso contrário prossegue a execução normalmente (instrução seguinte). Exemplo:
   JMPZ 10 => Se o Reg_A for igual a zero Então PC:=PC+10 Senão PC:=PC+1
   JPNZ -6 => Se o Reg_A for diferente de zero Então PC:=PC-6 Senão PC:=PC+1

10. Funcionamento da pilha de controle: A pilha de controle permite que sejam executadas as subrotinas, armazenando o número do comando a ser colocado no registrador de controle após retornar da execução da subrotina. Esta pilha é afetada por dois comandos apenas: SUBR e RETS. Exemplo:

    SUBR <instr> => Empilha na pilha de controle o número (endereço) da próxima instrução e desvia para a instrução indicada no comando: pilha <= PC+1; PC:=<instr>.

    RETS # => Desempilha da pilha de controle o número da próxima instrução a ser executada, colocando no registrador de controle de instruções: PC <= pilha.

As estruturas de dados usadas neste programa devem ser baseadas (similares) as rotinas de manipulação de estruturas de dados que estudamos e implementamos na nossa disciplina (LISTA, FILA, PILHA e DEQUE). Faça um prograna MODULAR e SEM USAR VARIÁVEIS GLOBAIS.

Consistência e Verificação de Erros:

- O programa de simulação do SUPER-CRIATURA deve ter um mínimo de consistência, detectando erros do tipo: pilha vazia (impossível de executar uma operação), pilha ou lista cheia (estouro da capacidade de armazenamento de uma estrutura de dados), comando inexistente/inválido. Entretanto, nós assumiremos um "sistema bem comportado", ou seja, assumimos que o usuário não tem a intenção de fornecer dados inválidos ao sistema, sendo obrigação deste prever principalmente as situações de erro ligadas próprio sistema e erros "acidentais".

  - Lista de ***erros que tem que ser detectados***:

  - Estouro (overflow / underflow) da pilha e/ou lista;
  - Desvio para nro. de comando inválido (endereço da instrução inválido)
  - Acesso a uma variável inexistente (leitura ou escrita)

Teste do SUPER-CRIATURA:

- Implemente um programa que calcule o fatorial de um número usando este simulador. Faça uma versão normal e uma versão recursiva.

BOM TRABALHO!