Elementos de Sistema – Avaliação Prática - “10”

Nome completo:

25 pts de HW	50 pts de SW

Instruções:

1. A avaliação tem duração total de 120 minutos.
2. **Você não pode consultar a internet, apenas seu repositório LOCAL**
3. Você deve editar esse documento quando necessário.

1) (5 HW, 5 SW) Conceitos

a) (3 HW, 3 SW) Explique como uma operação de jump ocorre no nosso hardware (tanto a nível de SW quanto a nível de HW)

b) (2 HW, 2 SW) Explique quais são os papeis dos registradores, memória ROM, memoryIO e ULA na execução de um programa que desenha na tela (LCD).

2) (0 HW, 25 SW) pseudocódigo

Arquivo	***F-Assembly/src/nasm/p3Pseudo.nasm***		
Teste	Simulação	**./testeAssembly.py**	***p3Pseudo***`.nasm 2 1000`

Transcreva o pseudocódigo a seguir para assembly do Z01.1:

```
WHILE(True):
  if    ( RAM[1] == 1 && RAM[2] == 2 ):
     RAM[5] = 1
  else
     RAM5[5] = !RAM[1]
```

- Para testar, descomentar linha do **/**`F-Assembly/tests/config.txt`
  - `p3Pseudo.nasm 2 1000`

Teste 0 (10 pts)	O teste 0 verifica o **else**.
Teste 1 (15 pts)	O teste 1 verifica o **if**.

3) (0 HW, 20 SW) VectorMax

Arquivo	Projetos/F-Assembly/src/nasm/**p3VectorMax.nasm**		
Teste	Simulação	**./testeAssembly.py**	`p3VectorMax.nasm 2 1000`

Buscar pelo valor máximo de um vetor e atualizar a RAM 2 com o valor encontrado

Assuma que:
- O endereço **0** da RAM indica a **posição inicial de um vetor**
- O endereço **1** da RAM indica o **tamanho do vetor**
- O vetor é uma região contínua da RAM

Considere o exemplo na qual um vetor de tamanho 5 está armazenado na memória RAM começando no endereço 4.

Vector = [15, 11, 15, 20, 12]

```
----x    RAM[ 0] : 4
|        RAM[ 1] : 5   x----
|        RAM[ 2] : MAX     |
|        RAM[ 3] : 1       |
------->RAM[ 4] : 15   ---|---
         RAM[ 5] : 11      |
         RAM[ 6] : 15      | Tamanho do vetor = 5
         RAM[ 7] : 20      |
         RAM[ 8] : 12   ---|---
         RAM[ 9] : 0
```

- Para testar: descomentar a linha do `/F-Assembly/tests/config.txt`
  - `p3VectorMax.nasm 2 1000`

Teste 0 (10 pts)	O teste 0 é o exemplo mostrado a seguir (vetor começa em 4 e tem 5 termos)
Teste 1 (15 pts)	O teste 1 é genérico, você deve ler os valores em RAM[0] e RAM[1]

4) (5 HW, 0 SW) LEDs

Arquivo	**Projetos/G-CPU/src/rtl/MemoryIO.vho**		
Teste	Simulação	**./testeAssemblyMyCPU.py**	`p3TestLed.nasm 1 1000`

No Z01.1 não somos capazes de realizar a leitura do valor do periférico **LED** do MemoryIO, isso as vezes dificulta o desenvolvimento de um software. Modifique o MemoryIO para possibilitar que o endereço referente aos LEDs possa ser do tipo **READ/WRITE,** possibilitando a operação a seguir:

```
leaw $21184, %A
movw (%A), %D
```

5) (15 HW, 0 SW) Melhorando o HW

O *instruction set* do nosso Z01.1 não diferencia operações entre números do tipo sem sinalização (**unsigned)** e com sinalização (complemento de dois, **signed**). Isso pode causar alguns problemas.

Por exemplo, assuma o estado inicial da CPU com os valores a seguir nos registradores:

REG	Valor em binário	Valor em Decimal
S	1111111111111111	65535 ou -1
D	1000000000000000	3278 ou -32768
A	0000000000000010	2

E o que acontece se executamos uma operação de jump greater (**jg)** em **%S** ou **%D ?** Como a sabe CPU sabe se os valores salvos em **%S** e **%D** são maiores ou menores que ZERO?

```
jg %S
```

JG - Desvia Execução se Maior que Zero

jg reg

- **Descrição:** A instrução *jg* desvia, o fluxo de execução, para o endereço armazenado em %A, somente se o valor do reg. for maior que zero.

```
jg %s
```

O problema é que a operação de **jump** não sabe se deve considerar o valor salvo em **%S** como **signed** (complemento de dois) ou **unsigned** (todos os bits usados para gravar o valor) e realiza o salto sempre que zr = 0 e ng = 0:

```
zr = 0
ng = 0   (verifica apenas se o bit 15 é um ou zero)
```

E ignora o caso na qual o valor salvo no registrador é maior igual ou maior que 32769 (**1**000 0000 0000 0000) e o sinal é para ser considerado como **unsigned.**

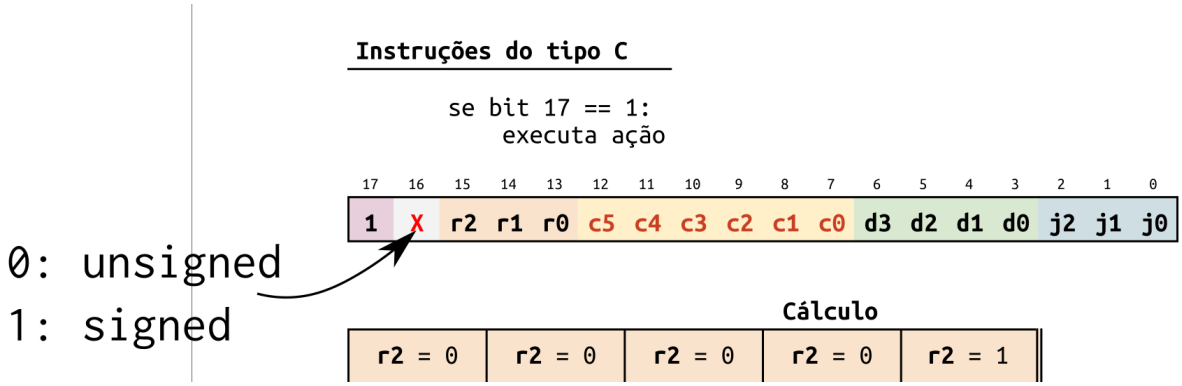
```
zr = 0
ng = 1
```

Processadores resolvem esse problema, informando na instrução de jump se o dado a ser verificado é to tipo **unsigned** ou **signed.**

Vamos modificar o hardware para possibilitar que a operação de **jg** (**jeg** sofre da mesma coisa, mas vamos ignorar) trate do sinal do registrador de duas maneiras diferentes:

`jgu %S`	Considera o registrador como sendo do tipo **unsigned**
`jgs %S`	Considera o registrador como sendo do tipo **signed**

Uma solução para implementarmos isso é a de utilizar o bit **16** das instruções do tipo **C** (que não tem uso no momento) para indicar se o número a ser operado é do tipo **unsigned** ou **signed**., como ilustrado a seguir:



Implementando

Arquivo:	Projetos/G-CPU/src/rtl/**ControlUnit.vhd**
Teste:	SiM: **./testeHW.py**

Encontre a tabela verdade e então equação lógica e implemente o VHDL para o novo instruction set que é referente somente ao jgX e que controla o sinal do **loadPC** (control unit).

5 HW pts	Tabela verdade
5 HW pts	Equação
5 HW pts	Implementação em VHDL (**./testeHW.py**