UNIVERSIDADE DE SÃO PAULO
Instituto de Ciências Matemáticas e de Computação

USP – ICMC – SSC

SSC0902 – Organização e Arquitetura de Computadores (OrgArq)

Professor responsável: *Fernando Santos Osório*
Semestre: 2020/2
Monitores: Gabriell Tavares Luna

Horário: Seg. 10h10 e Quinta 08h10
Wiki: SSC-902 e Facebook: SSC0902-FOsorio
Web: ***http://www.icmc.usp.br/~fosorio /***

NRO. USP: < Colocar o NUSP nos programas fonte> DATA**: 30 / 11 / 2020**

NOME: < Colocar o NOME nos programas fonte>

>> COLOCAR SEU NOME E NRO. USP COMO COMENTÁRIO DOS PROGRAMAS ENTREGUES!

PROVA FINAL – SSC0902 OrgArq (Programação)
MIPS – Simuladores MARS ou Qt-SPIM (Proc. RISC 32bits Mips)

Implemente o programa em Linguagem de Montagem do Processador **MIPS** (.S ou .ASM – Arquivo texto com programa), usando o montador e simulador disponibilizado nas aulas e usado para fazer trabalhos e exercícios durante o semestre, **MARS ou QtSPIM**, seguindo exatamente a especificação e instruções dadas abaixo => Entregar 2 programas (Questão 1 e Questão 2).

QUESTÃO 1 – Q1) Faça um programa para o MIPS que Processe uma Imagem contida em um arquivo em disco. Siga os passos indicados abaixo. [Q1 vale 5.0 pts]

(A) Ler o arquivo do disco "image.pgm" (procure colocar este arquivo no mesmo diretório do executável do simulador), contendo uma imagem. Esta imagem está em um formato chamado, PGM, mas não se preocupe, basta ler do disco um total de 100x100 (10.000 pixels, 1 byte por pixel em tons de cinza) + 37 bytes cabeçalho da imagem => Total 10.038 bytes a serem lidos. **A imagem começa em +38 bytes do início do arquivo.** Este arquivo deve ser lido para a memória usando as funções do sistema operacional "SysCalls".

(B) Aplicar uma "convolução" com um vetor de apenas 3 valores (-3, 10, -3) sobre todos os pixels da imagem. A convolução implica em multiplicar todos os valores da imagem por este vetor, por exemplo, dado o Pixel (X) da imagem (que possui 10.000 pixels), devemos fazer a seguinte operação:

Novo_Pixel(X) = Pixel (X-1)*-3 + Pixel(X)*10 + Pixel(X)+1*-3 .:. Para cada um dos 10.000 pixels
A convolução acessa os dados da imagem lida e gera uma NOVA imagem

(C) Salvar em disco o arquivo "conv.pgm" que é a imagem resultado da convolução realizada. Este arquivo deve ter os 38 bytes iniciais do cabeçalho (copiar os MESMOS 38 bytes iniciais do arquivo lido), seguido dos 10.000 novos bytes resultantes da operação de convolução realizada.

IMPORTANTE Q1:

- Antes de declarar a áreas (buffer) de dados de entrada e saída, use sempre a diretiva ".align 2" – Exemplo:

.align 2
Texto: .asciiz "Texto, Bytes, Words, Dados do Arquivo devem estar alinhados com Words na memoria"

- Os arquivo de imagens com extensão ".pgm" podem ser abertos diretamente usando o IrfanView para Windows (https://www.irfanview.com/) ou o software ImageJ que roda em Java – Win/Linux (http://imagej.nih.gov/ij)
- Você pode "olhar dentro" do conteúdo do arquivo de imagem neste site: https://hexed.it/

RESUMO Q1:
- Ler arquivo para memória. [Note que ler 10.000 bytes no simulador pode demorar uns 10 segundos na simulação]
- Aplicar a operação byte_novo = (byte-1)*(-3) + (byte)*10 + (byte+1)*-3 para os 10.000 bytes da imagem, gerando um novo bloco de memória com os bytes novos.
- Copiar o cabeçalho inicial da imagem (38 bytes) no cabeçalho da nova imagem.
- Salvar arquivo em disco com a nova imagem.

>> Para os curiosos, uma boa leitura **DEPOIS** de fazer a prova, é o site: https://setosa.io/ev/image-kernels/ que trata sobre convoluções, onde em vez de usar uma matriz 3x3 usamos um vetor 3x1, que equivale a uma matriz 3x3 com a linha superior e inferior com zeros (só usamos a linha central do kernel).

Aula sobre arquivos:
Aula 16.11 Arquivos 11-16 (S13) - Programação MIPS: Leitura/Escrita de Arquivos

==//==//==//==

QUESTÃO 2 – Q2) Faça um programa para o MIPS que implemente um "Neurônio Artificial" (Inspirado no Perceptron) calculando a ativação do neurônio, considerando Ponto-Fixo (inteiros) e Ponto-Flutuante (float). Siga os passos indicados abaixo. [Q2 vale 5.0 pts]

(2.1) Ler do teclado uma sequência de 3 números inteiros (com sinal): X1, X2, X3 (Neuron inputs)
(2.2) Ler do teclado uma sequência de 4 números inteiros (com sinal): W0, W1, W2, W3 (Neuron synaptic weights)
(2.3) Chamar uma sub-rotina para calcular a saída de ativação do neurônio. A saída do neurônio é a soma ponderada de peso(i) * entrada(i) + w0 => [s = w0 + x1*w1 + x2*w2 + x3*w3] (Neuron weighted sum)
Sub-rotina: CNI (Calculate Neuron Integer)
(2.4) Exibir o resultado da soma ponderada na tela: exibir o valor de saída int ("s" na equação acima) na tela
(2.5) Se o resultado da soma ponderada for maior que zero escrever na tela: "ativado". Se o valor for igual a zero, escrever na tela "zero" e se for menor que zero, escrever "inativo".

Repetir a mesma implementação, porém agora com valores de ponto flutuante, ou seja, com as entradas e pesos representados por valores em ponto flutuante (float – single precision, com sinal):
(2.6) Ler do teclado uma sequência de 3 números float: X1, X2, X3 (Neuron inputs)
(2.7) Ler do teclado uma sequência de 4 números float: W0, W1, W2, W3 (Neuron synaptic weights)
(2.8) Chamar uma sub-rotina para calcular a saída de ativação do neurônio. A saída do neurônio é a soma ponderada de peso(i) * entrada(i) + w0 => [s = w0 + x1*w1 + x2*w2 + x3*w3] (Neuron weighted sum)
Sub-rotina: CNF (Calculate Neuron Float)
(2.9) Exibir o resultado da soma ponderada na tela: exibir o valor de saída float ("s" na equação acima) na tela
(2.10) Se o resultado da soma ponderada for maior que zero escrever na tela: "ativado". Se o valor for igual a zero, escrever na tela "zero" e se for menor que zero, escrever "inativo".

Lembre-se que o programa deve ter no MAIN pelo menos 2 chamadas de sub-rotinas, CNI e CNF, que executam o cálculo da saída do neurônio, e deve exibir na tela as mensagens conforme a saída da ativação do neurônio que for calculada. Respeitar o devido uso ("usage") de registradores quando usar sub-rotinas.

Aula sobre Aritmética de Ponto Flutuante no MIPS:
Aula 12.11 Arquivos 11-12 (S12) - Programação MIPS: Operações de Ponto-Flutuante e Alocação de Memória

>> Leitura para **DEPOIS** da prova sobre o Perceptron: https://en.wikipedia.org/wiki/Perceptron e https://www.neuroelectrics.com/blog/2016/08/02/artificial-neural-networks-the-rosenblatt-perceptron/

INFORMAÇÕES IMPORTANTES:

REGRAS EM RELAÇÃO REALIZAÇÃO DESTA PROVA

- A PROVA É **INDIVIDUAL.**
- PROVA DEVE SER REALIZADA CONSIDERANDO AS REGRAS DO "**Contrato**" QUE VOCÊ ACEITOU => https://forms.gle/mAaVRPo5jdRDbWr37

(É necessário aceitar os termos do contrato para que a prova seja corrigida!)

MIPS Register – Names and Usage

Register name	Number	Usage
$zero	0	constant 0
$at	1	reserved for assembler
$v0	2	expression evaluation and results of a function
$v1	3	expression evaluation and results of a function
$a0	4	argument 1
$a1	5	argument 2
$a2	6	argument 3
$a3	7	argument 4
$t0	8	temporary (not preserved across call)
$t1	9	temporary (not preserved across call)
$t2	10	temporary (not preserved across call)
$t3	11	temporary (not preserved across call)
$t4	12	temporary (not preserved across call)
$t5	13	temporary (not preserved across call)
$t6	14	temporary (not preserved across call)
$t7	15	temporary (not preserved across call)
$s0	16	saved temporary (preserved across call)
$s1	17	saved temporary (preserved across call)
$s2	18	saved temporary (preserved across call)
$s3	19	saved temporary (preserved across call)
$s4	20	saved temporary (preserved across call)
$s5	21	saved temporary (preserved across call)
$s6	22	saved temporary (preserved across call)
$s7	23	saved temporary (preserved across call)
$t8	24	temporary (not preserved across call)
$t9	25	temporary (not preserved across call)
$k0	26	reserved for OS kernel
$k1	27	reserved for OS kernel
$gp	28	pointer to global area
$sp	29	stack pointer
$fp	30	frame pointer
$ra	31	return address (used by function call)

MIPS – SPIM & MARS SYSCALLS (System Call code in $v0)

Service	System call code	Arguments	Result
print_int	1	$a0 = integer	
print_float	2	$f12 = float	
print_double	3	$f12 = double	
print_string	4	$a0 = string	
read_int	5		integer (in $v0)
read_float	6		float (in $f0)
read_double	7		double (in $f0)
read_string	8	$a0 = buffer, $a1 = length	
sbrk	9	$a0 = amount	address (in $v0)
exit	10		
print_char	11	$a0 = char	
read_char	12		char (in $a0)
open	13	$a0 = filename (string), $a1 = flags, $a2 = mode	file descriptor (in $a0)
read	14	$a0 = file descriptor, $a1 = buffer, $a2 = length	num chars read (in $a0)
write	15	$a0 = file descriptor, $a1 = buffer, $a2 = length	num chars written (in $a0)
close	16	$a0 = file descriptor	
exit2	17	$a0 = result	