Arquitetura e Organização de computadores

Aluno: Felipe Derkian de Sousa Freitas

Matrícula: 1201424418

Prof: Herbert Rocha

Lista I

Boa Vista, 13 de Dezembro de 2016

(Questão 1) A arquitetura de Von Neumann é uma arquitetura de computador que se caracteriza pela possibilidade de uma máquina digital armazenar seus programas no mesmo espaço de memória que os dados, podendo assim manipular tais programas.

(Questão 2) Princeton: foi um computador que tinha memória comum para armazenar o programa de controle, bem como variáveis e outras estruturas de dados. A maior vantagem da arquitetura Princeton é que simplifica o projeto do micro controlador porque somente uma memória é acessada.

Harvard: foi um projeto que usava bancos de memória separados para armazenamento de programas, a pilha de processos e a RAM variável. A vantagem da arquitetura Harvard é que o conteúdo da RAM possa ser usado tanto para armazenamento de variáveis (dados) como para o armazenamento de instruções de programas tendo vantagem para algumas aplicações e o conteúdo da pilha do contador de programas, isso permite uma maior flexibilidade no desenvolvimento de software, principalmente na área de sistemas operacionais em tempo real. A arquitetura de Harvard executa instruções em menos ciclos que a de Princeton.

(Questão 3)

WR = 3.14 \* r²

WR = 3.14 \* 20² = 1256cm²

A = 1 cm²

B = 4 cm²

C = 9 cm²

1. Rendimento = 1 / (1 + ( 0.5 \* ½))² = 0.64 = 64%.

Chips p/ Wafer = 1256 cm²/1 cm² = 2156 chips.

Custo p/ chip = 100/1256\*0.64 = U$ 0.12440287.

1. Rendimento = 1 / (1+( 0.5 \* 4/2))² = 0.25 ou 25%

Chips p/ Wafer = 1256 cm²/4 cm² = 314 chips.

Custo p/ chip = 100/1256\*0.25 = U$ 0.31847134.

1. Rendimento = 1 / (1 + (0.5 \*9/2)) ² = 0.09467 ou 9,467%.

Chips p/ wafer = 1256 cm²/9 cm² = 139,555 chips.

Custo p/ chip = 100 / 139,555 \* 0.0946 = U$ 7,569.

(Questão 4)   
Performance = 1/frequências

P = 7/500\*106 = 2 nano-segundo(ns)  
  
A)

CPI = 5\*1 + 1\*2 + 1\*3/7 = 1,42857143.  
tempo = I \* 1.428 \* 2ns = 2.856 \* I \* ns

B)  
CPI = 10\*1 + 1\*2 + 1\*3/12 = 1,25.  
tempo = I \* 1.25 \*2 ns = 2.5 \* I \* ns

Tempo execução (a) / Tempo execução (b) = 2.856 \* I \* ns / 2.5 \* I \* ns = 1.1424 vezes

B é mais rápido que A 1.1424 vezes, pois leva menos ciclos para toda a execução.

MIPS(1) = 7 / 2.856 \* 106 = 2,45 x 10-6

MIPS(2) = 12/2.5\*106 = 4.8x10-6

4.8x10-6 / 2,45 x 10-6 = 1,9591 vezes

No MIPS, o compilador 1 leva menos tempo 2,45 x 10-6 e o compilador 2 demora 4.8x10-6 sendo mais rápido 1,9591 vezes ou seja quase 2 vezes mais rápido.

(Questão 5) performance = 1/tempo de execução

Programa 1:

Pm1 = 1/10 =0.1 Pm2 = 1/5 = 0.2

Tempo execução m2 > Tempo execução m1

Pm1/Pm2 = 0.2/0.1 = 2 vezes mais rápido.

Programa 2:

Pm1 = 1/3 =0.333 Pm2 = 1/4 = 0.25

Tempo execução m1 > Tempo execução m2

Pm2/Pm1 = 0.333/0.25 = 1.332 vezes mais rápido.

(Questão 6)

RISC: Reduced Instruction Set Computer, Poucas instruções simples executadas diretamente pelo hardware. Ex.: Sparc, Alpha, MIPS

\*Número reduzido de instruções  
\*Instruções com mesmo tamanho  
\*Muitos registradores  
\*Operações somente entre registradores  
\*Instruções executadas diretamente pelo hardware

CISC: Complex Instruction Set Computer, muitas instruções complexas interpretadas por microprogramas. Ex.: x86

\*Grande variedade de instruções  
\*Instruções de tamanho variado  
\*Poucos registradores  
\*Operações em memória  
\*Utiliza microcódigos

(Questão 7)

Assume, $s0-$s5 contém f, g, h, i, j, k.  
Assume $t2 contém 4.

slt $t3, $s5, $zero # if k< 0 then $t3 = 1 else $t3=0

bne $t3, $zero, Exit # if k< 0 then Exit

slt $t3, $s5, $t2 # if k<4 then $t3 = 1 else $t3=0

beq $t3, $zero, Exit # if k≥4 the Exit

add $t1, $s5, $s5 # t1 = 2\*k

add $t1, $t1, $t1 # t1 = 4\*k

add $t1, $t1, $t4 # t1 = base address + 4\*k

lw $t0, 0($t1) # load the address pointed

# by t1 into register t0

jr $t0 # jump to addr pointed by

L0: add $s0, $s3, $s4 # f = i + j

J Exit

L1: add $s0, $s1, $s2 # f = g+h

J Exit

L2: sub $s0, $s1, $s2 # f = g-h

J Exit

L3: sub $s0, $s3, $s4 # f = i - j

(Questão 8)  
  
 insertion\_sort:

# Caller RTE store

addi $sp, $sp, -20

sw $fp, 20($sp)

sw $ra, 16($sp)

sw $a0, 12($sp)

sw $a1, 8($sp)

addi $fp, $sp, 20

# Implement insertion sort

add $s0, $zero, $zero # i = 0

fLoop:

addi $s0, $s0, 1 # i += 1

beq $s0, $a1, insertion\_sort\_end # br if i = arrayLength

addi $s1, $s0, -1 # j = i - 1

wLoop:

beq $s0, $a1, fLoop # br if i = arrayLength

blt $s1, $s0, swap # br if j < i

addi $s1, $s1, -1 # j = j - 1

j wLoop # continue while loop

swap:

add $t0, $s0, $a0

add $t1, $s1, $a0

lb $t2, ($t0)

lb $t3, ($t1)

sb $t3, ($t0)

sb $t2, ($t1)

j wLoop # continue while loop

insertion\_sort\_end:

# Caller RTE restore

lw $fp, 20($sp)

lw $ra, 16($sp)

lw $a0, 12($sp)

lw $a1, 8($sp)

addi $sp, $sp, 20

# Return to Caller

jr $ra

(Questão 9)

\*Op: A operação básica a ser realizada pela instrução, tradicionalmente chamada de código de operação(opcode).  
\*Rs: O registrador contendo o primeiro operando fonte.  
\*Rt: O registrador contendo o segundo operando fonte.  
\*Rd: O registrador que guarda o resultado da operação, também conhecido como registrador destino.  
\*Shamt: de quantidade de bits a serem deslocados.  
\*Funct: Este campo seleciona uma variação especifica da operação apontada no campo Op, sendo as vezes chamado de código de função.

(Questão 10)

