Universidade de Brasília
Departamento de Ciência da Computação

Disciplina: CIC 116394 – Organização e Arquitetura de Computadores – Turma A
Prof. Marcus Vinicius Lamar

2016/1

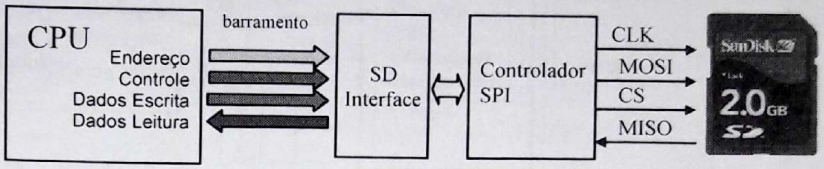
Nome: GABARITO

d0 d1 / d2 d3 d4 d5 d6 d7 d8
Matrícula: 16/0123456

Prova 2

1)(4.0) Nos dispositivos móveis, tais como *smartphones* e *tablets*, o consumo de energia de um dispositivo de armazenamento em massa é um fator primordial. A memória *flash* veio de encontro a esta necessidade, permitindo o armazenamento de grande quantidade de dados de modo não-volátil e com baixo consumo.

Considere que o cartão SD (*Secure Digital*) de 2GiB seja composto de 4194304 setores de 512 bytes, linearmente endereçados (0 a 4194303). O controlador de acesso ao cartão SD utiliza uma interface SPI, capaz de ler e gravar um bloco de dados (setor), disponibilizado através de 128 registradores de 32 bits (SD_BUFFER) contidos na interface. A interface SD possui ainda um registrador de 32 bits que armazena o endereço do setor (SD_ADDRESS) a ser acessado no cartão SD e um byte de controle contendo os sinais SD_READ, SD_READ_DONE, SD_WRITE, e SD_WRITE_DONE. A interface SD comanda o controlador do cartão SD da seguinte forma:



- Quando SD_READ é setado o controlador realiza o processo de armazenamento de um bloco de dados presente no cartão no endereço SD_ADDRESS nos registradores do SD_BUFFER. Ao fim do processo de leitura, o sinal SD_READ_DONE é setado pela interface e permanece setado até SD_READ ser resetado.
- Quando SD_WRITE é setado o controlador realiza o processo de gravação de um bloco de dados presente nos registradores SD_BUFFER para o cartão no endereço SD_ADDRESS. Ao fim do processo de escrita, o sinal SD_WRITE_DONE é setado pela interface e permanece setado até SD_WRITE ser resetado.

Os endereços MMIO da interface SD são definidos na tabela ao lado, e os bits de SD_CONTROL são definidos abaixo:

SD_CONTROL[0]=SD_READ
SD_CONTROL[1]=SD_READ_DONE
SD_CONTROL[2]=SD_WRITE
SD_CONTROL[3]=SD_READ_DONE

Endereço	Tamanho	Função
0xFFFF0250	1 word	SD_BUFFER[0]
...	...	...
0xFFFF044C	1 word	SD_BUFFER[127]
0xFFFF0450	1 word	SD_ADDRESS
0xFFFF0454	1 byte	SD_CONTROL

a) (2.0) Escreva um procedimento `SDRead` que leia $a2 words a partir do endereço $a0 (endereço virtual em bytes alinhado) do cartão SD, e escreva a partir do endereço $a1(alinhado) da memória de dados.

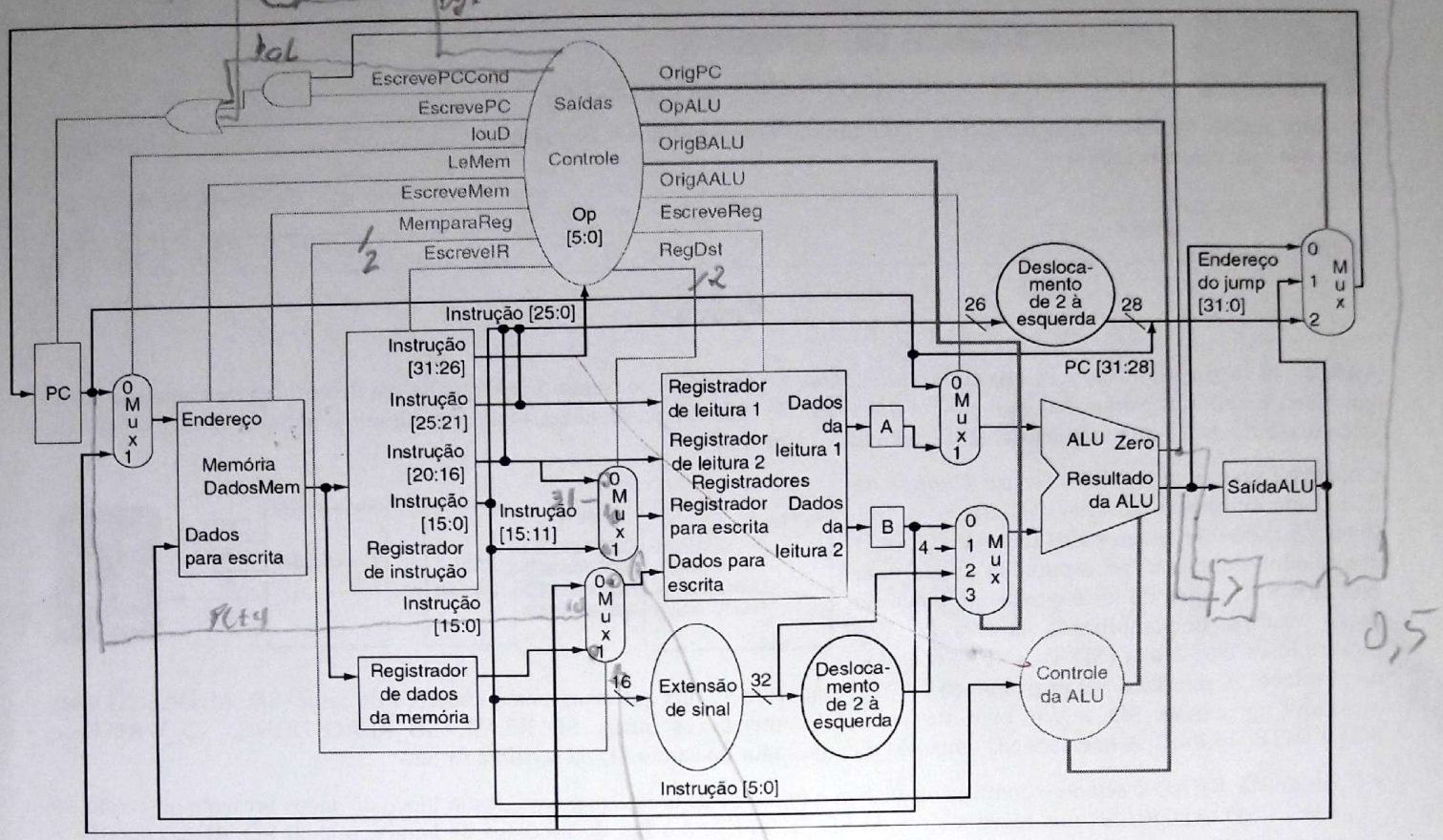
b) (2.0) Escreva um procedimento `SDWrite` que leia $a2 words a partir do endereço $a0 (alinhado) da memória de dados, e escreva a partir do endereço $a1 (endereço virtual em bytes alinhado) do cartão SD. Lembre-se que apenas os endereços solicitados é que devem ser modificados no cartão!

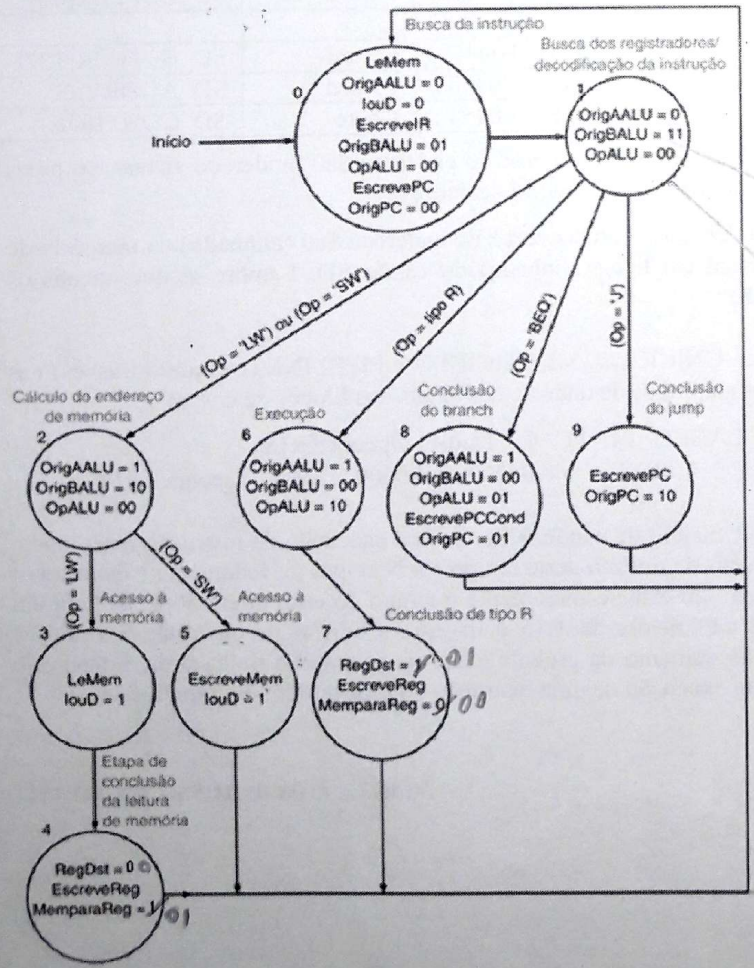
2) (6.0) Implemente as instruções abaixo nos processadores UNICICLO, MULTICICLO e PIPELINE (incluindo flushes) nas folhas em anexo, definindo as modificações necessárias nos caminhos de dados e nos sinais dos blocos de controle:

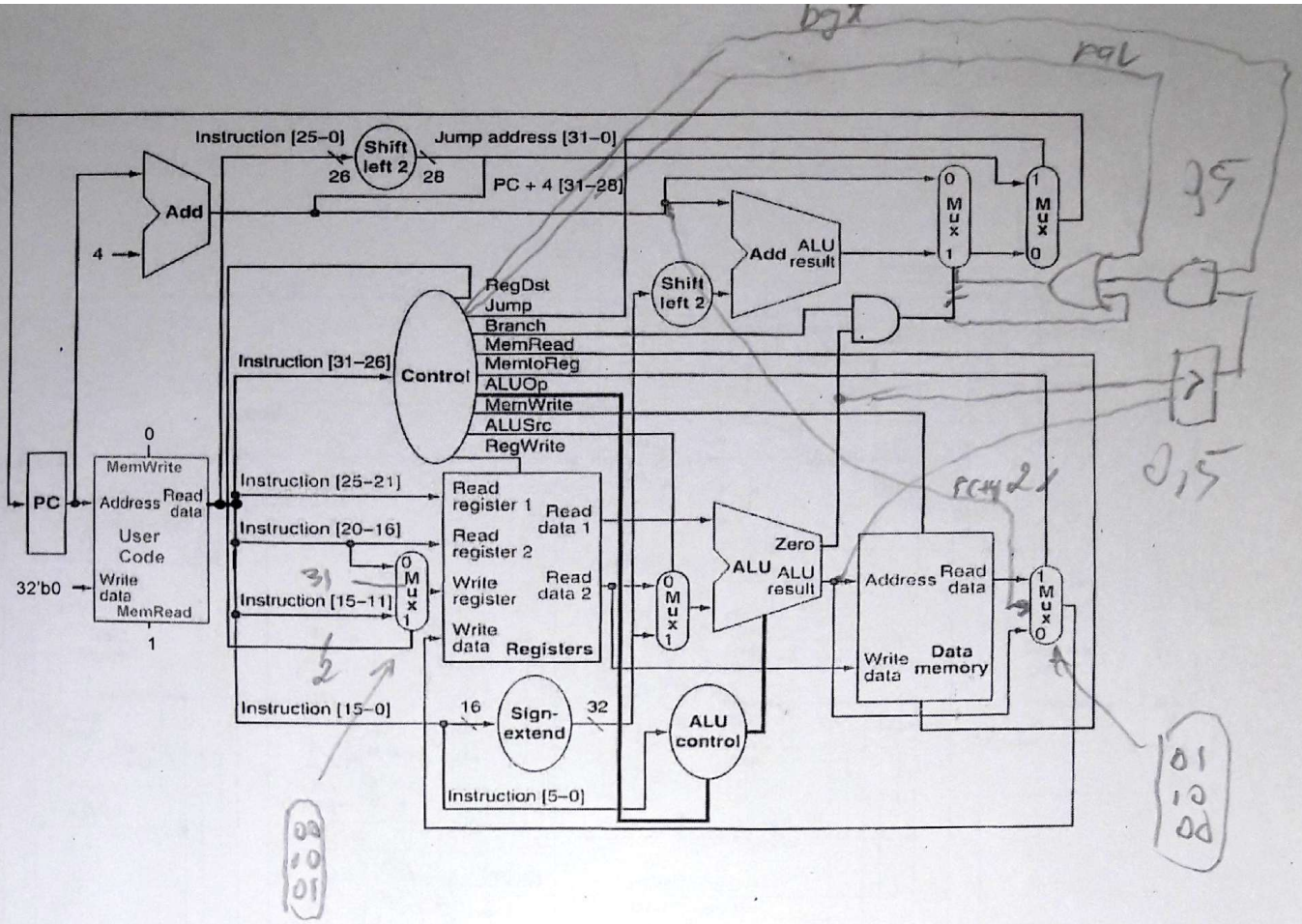
a) (3.0) `bgt rs,rt,LABEL` # rs>rt ? PC=LABEL : PC=PC+4 Tipo-I Opcode:0x18
b) (3.0) `bal LABEL` # $ra=PC+4 e PC=LABEL rs=00000 rt=00000 Tipo-I Opcode:0x19

3) (2.0) Suponha que a implementação de um processador Uniciclo demande 900ps para a execução da instrução mais longa. A criação de um processador em Pipeline pode ser feita a partir da divisão deste tempo em N etapas perfeitamente balanceadas. Sabendo que: i) A inclusão de um registrador de pipeline em um estágio incrementa o tempo do estágio em 50ps. ii) Para um dado *workload*, o processador de 5 estágios possui uma CPI média de 1.15 e iii) que a adição de um estágio a mais é penalizado pelo incremento de 0.1 na CPI média, devido ao aumento da probabilidade de ocorrência de hazards. Encontre o número de etapas (N) ótimo que minimiza o tempo médio de execução de uma instrução no processador em pipeline.

Muita hora nessa calma!!!

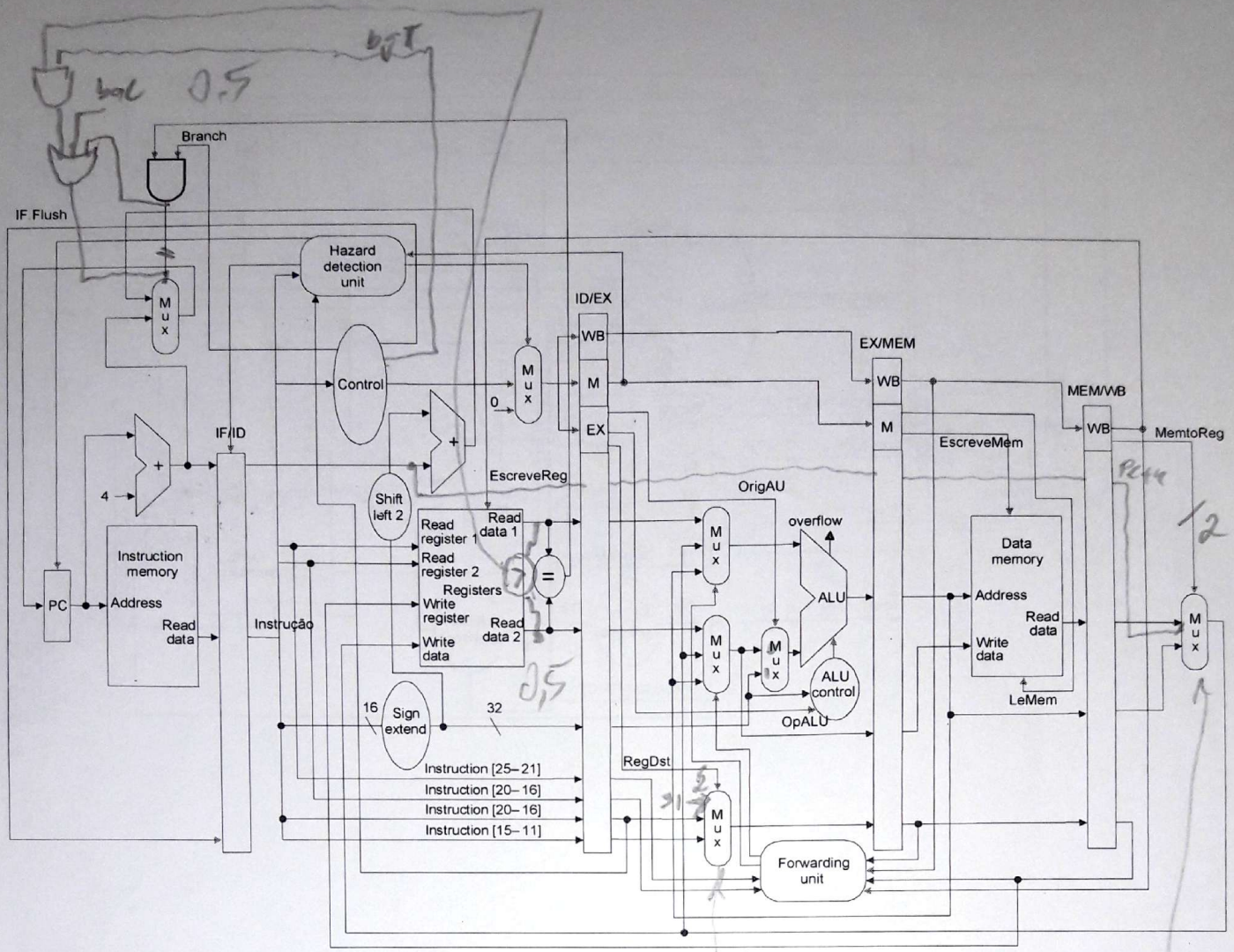

EscrevePCCond
EscrevePC
IouD
LeMem
EscreveMem
MemparaReg
EscreveIR
Saídas
Controle
Op [5:0]
OrigPC
OpALU
OrigBALU
OrigAALU
EscreveReg
RegDst
Instrução [25:0]
Instrução [31:26]
Instrução [25:21]
Instrução [20:16]
Instrução [15:0]
Registrador de instrução
Instrução [15:11]
Instrução [15:0]
Registrador de dados da memória
PC
Endereço
Memória
DadosMem
Dados para escrita
Registrador de leitura 1
Registrador de leitura 2
Registradores
Registrador para escrita
Dados para escrita
Dados da leitura 1
Dados da leitura 2
A
B
4
Extensão de sinal
16
32
Deslocamento de 2 à esquerda
26
28
PC [31:28]
Endereço do jump [31:0]
ALU
Zero
Resultado da ALU
SaídaALU
Controle da ALU
Instrução [5:0]


Busca da instrução
Busca dos registradores/ decodificação da instrução
Início
0
LeMem
OrigAALU = 0
IouD = 0
EscreveIR
OrigBALU = 01
OpALU = 00
EscrevePC
OrigPC = 00
1
OrigAALU = 0
OrigBALU = 11
OpALU = 00
(Op = 'LW') ou (Op = 'SW')
(Op = tipo R)
(Op = 'BEQ')
(Op = 'J')
Cálculo de endereço de memória
Execução
Conclusão do branch
Conclusão do jump
2
OrigAALU = 1
OrigBALU = 10
OpALU = 00
6
OrigAALU = 1
OrigBALU = 00
OpALU = 10
8
OrigAALU = 1
OrigBALU = 00
OpALU = 01
EscrevePCCond
OrigPC = 01
9
EscrevePC
OrigPC = 10
(Op = 'LW')
(Op = 'SW')
Acesso à memória
Acesso à memória
Conclusão de tipo R
3
LeMem
IouD = 1
5
EscreveMem
IouD = 1
7
RegDst = 1
EscreveReg
MemparaReg = 0
Etapa de conclusão da leitura de memória
4
RegDst = 0
EscreveReg
MemparaReg = 1



Instrução	RegDst	OrigALU	Mempara Reg	Escreve Reg	Le Mem	Escreve Mem	Branch
formato R	01	0	00	1	0	0	0
lw	00	1	01	1	1	0	0
sw	XX	1	XX	0	0	1	0
beq	XX	0	XX	0	0	0	1
bgt	XX	0	XX	0	0	0	0
bal	10	X	10	1	0	0	0

Instrução	ALUOp1	ALUOp0	bgt	bal	JUMP=0			
formato R	1	0	0	0				
lw	0	0	0	0				
sw	0	0	0	0				
beq	0	1	0	0				
bgt	0	1	1	0				
bal	X	X	0	1				



NOVO BLOCO

Rs —
Rt —

Instrução	Linhas de controle do estágio de cálculo de endereço/execução						
	RegDst	OpALU1	OpALU0	OrigALU			
Formato R	1	1	0	0			
lw	0	0	0	1			
sw	X	0	0	1			
beq	X	0	1	0			

Instrução	Linhas de controle do estágio de acesso à memória					
	Branch	LeMem	Escreve Mem			
Formato R	0	0	0			
lw	0	1	0			
sw	0	0	1			
beq	1	0	0			

Instrução	Linhas de controle do estágio de escrita do resultado				
	Escreve Reg	Mem para Reg			
Formato R	1	0			
lw	1	1			
sw	0	X			
beq	0	X			

OAC-A 2016/1

Prova 2

Gabarito

1) As ligações estão em Anexo

Pontos avaliados:

a) SD-Read

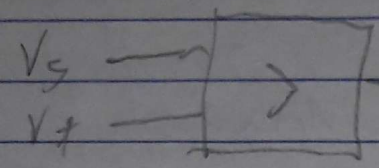
- Converte end. virtual → END. do BLOCO 0,2
  Nº Bytes → Nº BLOCOS 0,2
- Escreve END no SD_ADDRESS 0,2
- Acionar BIT SD_read 0,2
- Polling no BIT SD_Read_DONE 0,2
* LOOP transferir do BUFFER p/ MEMÓRIA 0,5
→ Se coloou início e final 0,5

b) SD-WRITE

- Converter End Virtual → END. BLOCO 0,2
  Nº Bytes → Nº BLOCOS
* LER BLOCO p/ SD_BUFFER 0,5
*LOOP p/ Gravar Endereço [illegible] no SD BUFFER 0,5
- Escrever SD_ADDRESS 0,2
- Acionar SD_WRITE 0,2
- Polling do BIT SD_WRITE_DONE 0,2
- voltar p/ Nº BLOCO 0,2
→ se colocou início e final 0,5

2)

a) bgt Vs, Vt, LABEL Vs > Vt?

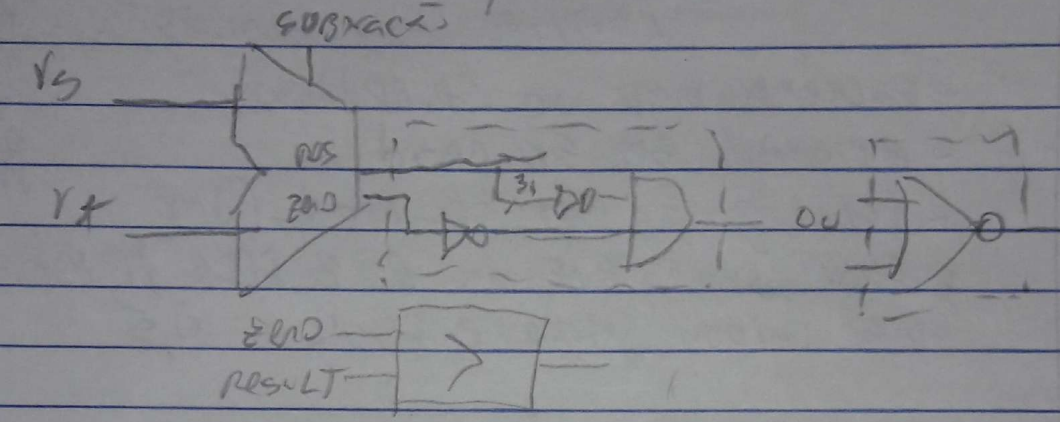


Vs > Vt

Vs − Vt > 0

Subtrai e verifica B.T mais significativo do resultado

Desde que Resultado ≠ 0



3) $t_{exec} = I \times CPI \times T$

$\frac{t}{I} = CPI \times T$ TEMPO MÉDIO

$t_M = (1.15 + 0.1(N-5)) \times \left(\frac{900}{N} + 50\right)$

$t_M = (0.1N + 0.65) \times (900N^{-1} + 50)$

$= 90 + 5N + 585N^{-1} + 32.5$

$= 5N + 585N^{-1} + 122.5$

PONTO onde t_M é MÍNIMO

$\frac{\partial t_M}{\partial N} = 0 \quad 5 - 585N^{-2} = 0$

$N^{-2} = \frac{5}{585}$

$N = 10,8 \rightarrow$ Logo $N = 11$ Estágios

```
.eqv SD_BUFFER  0xFF000250
.eqv SD_ADDRESS 0xFF00044C
.eqv SD_CONTROL 0xFF000454

.data
END_DATA: .space 1024   #Espaco para colocar os dados lidos do cartão SD  e também serve de
dados para serem gravados no SD

.text
MAIN:   la $a0,0x00000100   # endereco virtual de origem no cartão (256, meio do 1o bloco só
para complicar)
    la $a1,END_DATA     # endereco de destino
    li $a2,1024     # numero de bytes (1/2 bloco + 1 bloco + 1/2 bloco)
    jal SDRead

    la $a0,0x00000100   # endereco de origem
    la $a1,END_DATA     # endereco virtual de destino no cartão (256, meio do 1o bloco só
    para complicar)
    li $a2,1024     # numero de bytes (1/2 bloco + 1 bloco + 1/2 bloco)
    jal SDWrite

    li $v0,10
    syscall




SDWrite: beq $a2,$zero,FIM_SDWRITE # se quantidade de bytes é zero então FIM
    srl $t7,$a2,9       # calcula o número de blocos de 512 bytes a serem gravados dada a
    quantidade de bytes solicitada
    addi $t7,$t7,1
    srl $t6,$a0,9       # converte o endereço virtual (em bytes) a ser gravado em endereço
    real (em blocos) do Cartão
    andi $t5,$a0,0x000001FF # extrai o endereço inicial em bytes dentro do primeiro bloco
    add $a3,$a1,$a2     # calcula qual deve ser o último endereço a ser lido da memoria de
    dados
    # para cada bloco (setor)
    # le setor de destino do cartão para o SD_BUFFER
WLOOP1:  la $t2,SD_ADDRESS  # escreve o endereço real a ser lido na interface SD
    sw $t6,0($t2)

    li $t0,SD_CONTROL   # aciona o bit 0 do SD_READ indicando para a interface realizar a
    leitura (e zera o bit 1)
    li $t1,0x01
    sb $t1,0($t0)

WWAIT:  j SIMULA_READW      # esta chamada não existe na prática, só para simulação no Mars
WW1:    lw $t1,0($t0)       # le o bit 1 do SD_CONTROL.  SD_READ_DONE
    andi $t1,$t1,0x02
    beq $t1,$zero,WWAIT # espera o SD_READ_DONE ser setado

    # grava os dados da origem no SD_BUFFER no lugar correto
    li $t2,512      # numero de bytes nos registradore SD_BUFFER words
    li $t0,0
    li $t3,SD_BUFFER
WLOOP2: beq $t2,$t0,WFORA2  # transfere até 128 words da memoria $a1 para SD_BUFFER
```

```
    blt $t0,$t5,PULA2   # para pular até o endereço no primeiro bloco
    lw $t4,0($a1)
    sw $t4,0($t3)
    addi $t3,$t3,4      # incrementa posição do SD_BUFFER $t3
PULA2:  addi $a1,$a1,4      # incrementa endereço $a1
    addi $t0,$t0,4      # contador de bytes.
    bge $a1,$a3,WFORA2  # Se chegou ao último endereço antes de terminar um bloco então FIM
    j WLOOP2

    # grava SD_BUFFER no cartão
WFORA2: li $t0,SD_CONTROL   # aciona o bit 2 do SD_WRITE indicando para a interface realizar
a gravação (e zera o bit 3)
    li $t1,0x04
    sb $t1,0($t0)

WWAIT2: j SIMULA_WRITE      # esta chamada não existe na prática, só para simulação no Mars
WW2:    lw $t1,0($t0)       # le o bit 3 do SD_CONTROL.  SD_WRITE_DONE
    andi $t1,$t1,0x08
    beq $t1,$zero,WWAIT2    # espera o SD_READ_DONE ser setado

    #próximo bloco
    addi $t7,$t7,-1     # contador de blocos
    add $t6,$t6,1       # endereço real do bloco no cartão
    li $t5,0        # não será mais o primeiro bloco
    bne $t7,$zero, WLOOP1

FIM_SDWRITE: li $t0,SD_CONTROL  # reseta o bit 2 do SD_WRITE
    li $t1,0x00
    sb $t1,0($t0)
    jr $ra
    mod $1,$t2,$t4




SIMULA_WRITE:   li $k1,0x08         # liga o bit 3 de SD_WRITE_DONE  (é zerado junto com o
set do bit 2)
        la $k0,SD_CONTROL
        sw $k1,0($k0)
        j WW2




SDRead: beq $a2,$zero,FIM_SDREAD # se quantidade de bytes é zero então FIM
    srl $t7,$a2,9       # calcula o número de blocos de 512 bytes a serem lidos dada a
    quantidade de bytes solicitada
    addi $t7,$t7,1
    srl $t6,$a0,9       # converte o endereço virtual (em bytes) a ser lido em endereço real
    (em blocos) do Cartão
    andi $t5,$a0,0x000001FF # extrai o endereço inicial em bytes dentro do primeiro bloco
    add $a3,$a1,$a2     # calcula qual deve ser o último endereço a ser preenchido na
    memoria de dados

RLOOP1:  la $t2,SD_ADDRESS  # escreve o endereço real a ser lido na interface SD
    sw $t6,0($t2)
```

```
    li $t0,SD_CONTROL   # aciona o bit 0 do SD_READ indicando para a interface realizar a
    leitura (e zera o bit 1)
    li $t1,0x01
    sb $t1,0($t0)

RWAIT:  j SIMULA_READ       # esta chamada não existe na prática, só para simulação no Mars
RW1:    lw $t1,0($t0)       # le o bit 1 do SD_CONTROL.  SD_READ_DONE
    andi $t1,$t1,0x02
    beq $t1,$zero,RWAIT # espera o SD_READ_DONE ser setado

    li $t2,512      # numero de bytes nos registradore SD_BUFFER words
    li $t0,0
    li $t3,SD_BUFFER
RLOOP2: beq $t2,$t0,RFORA2  # transfere até 128 words do SD_BUFFER para a memória em $a1
    blt $t0,$t5,PULA    # para pular até o endereço no primeiro bloco
    lw $t4,0($t3)
    sw $t4,0($a1)
    addi $a1,$a1,4      # incrementa endereco de destino $a1
PULA:   addi $t3,$t3,4      # incrementa posição no SD_BUFFER
    addi $t0,$t0,4      # contador de bytes.
    bge $a1,$a3,FIM_SDREAD  # Se chegou ao último endereço antes de terminar um bloco então
    FIM
    j RLOOP2

RFORA2: addi $t7,$t7,-1     # contador de blocos
    add $t6,$t6,1       # endereço real do bloco no cartão
    li $t5,0        # não será mais o primeiro bloco
    bne $t7,$zero, RLOOP1

FIM_SDREAD: li $t0,SD_CONTROL   # reseta o bit 0 do SD_READ
    li $t1,0x00
    sb $t1,0($t0)
    jr $ra




SIMULA_READ:    la $k0,SD_BUFFER        # simula a leitura do cartão para o SD_BUFFERs
            li $k1,128
LOOP_SIMREAD:   beq $k1, $zero, FIM_SIMREAD
        sw $k0,0($k0)           #Preenche o SD_BUFFER com o  próprio endereco só para teste!
        addi $k1,$k1,-1         # umero de word
        addi $k0,$k0,4          # endereço do SD_BUFFER[]
        j LOOP_SIMREAD
FIM_SIMREAD:    li $k1,0x02         # liga o bit 1 de SD_READ_DONE  (é zerado junto com o
set do bit 0)
        la $k0,SD_CONTROL
        sw $k1,0($k0)
        j RW1
```

```
SIMULA_READW:   la $k0,SD_BUFFER        # simula a leitura do cartão para o SD_BUFFERs
            li $k1,128
LOOP_SIMREADW:  beq $k1, $zero, FIM_SIMREADW
        sw $k0,0($k0)           #Preenche o SD_BUFFER com o  próprio endereco só para teste!
        addi $k1,$k1,-1         # umero de word
        addi $k0,$k0,4          # endereço do SD_BUFFER[]
        j LOOP_SIMREADW
FIM_SIMREADW:   li $k1,0x02         # liga o bit 1 de SD_READ_DONE  (é zerado junto com o
set do bit 0)
        la $k0,SD_CONTROL
        sw $k1,0($k0)
        j WW1
```