Universidade de Brasília
Departamento de Ciência da Computação

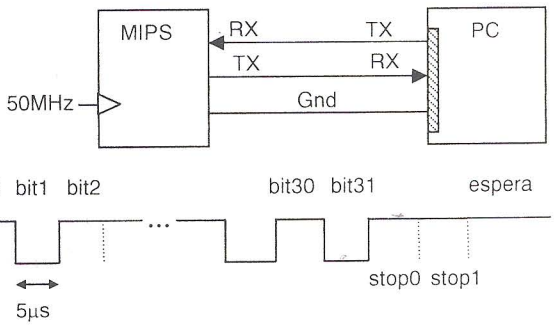
Disciplina: CIC 116394 – Organização e Arquitetura de Computadores – Turma A 2014/2
Prof. Marcus Vinicius Lamar

d0 d1 / d2 d3 d4 d5 d6 d7 d8

Nome: GABARITO Matrícula: □□/□□□□□□□□

Prova 2

1)(3.0) A comunicação entre máquinas abriu o caminho para o desenvolvimento da rede mundial de computadores – a internet. No entanto, tudo se iniciou com formas de comunicação serial bastante simples. A UART (*Universal Asyncronous Receiver/Transmitter)* foi uma das primeiras e continua sendo bastante popular. Considere uma interface serial assíncrona direta (*bit-banging*) em níveis lógicos (0 e +3.3V) composta dos terminais RX, TX e GND, diretamente ligada à interface do PC. O protocolo definido é composto por 1 bit de start, 32 bits de dados, 2 bits de stop, sem bit de paridade, e uma taxa de transmissão fixa de 200 kbauds de acordo com a figura ao lado.



Suponha que o PC (Processador Core i7, 4GHz, com 32KiB L1, 512KiB L2, 8MiB L3, 8GiB RAM, 1TB de HD) já possua em sua *South Bridge*, todo o hardware necessário para a comunicação com esta interface. No entanto, o processador MIPS UNICICLO possui apenas os terminais RX e TX diretamente ligados aos bits 0 e 1, respectivamente, de um registrador mapeado (MMIO) no endereço 0xFFFF0200, que é atualizado a cada borda de subida do sinal de clock de 50MHz. Considere que quando não há transmissão os terminais permanecem em nível lógico 1, e a comunicação se inicia pelo start bit seguido do bit menos significativo da word.

a)(1.5) Escreva uma rotina syscall UART_RX que, ao ser chamada, monitore o sinal RX e retorne no registrador $v0, a word recebida pela interface serial.

b)(1.5) Escreva uma rotina syscall UART_TX que transmita serialmente a word presente no registrador $a0.

2) (3.0) Dado o processador MIPS UNICICLO em anexo, acrescente mais um módulo de memória RAM dedicado ao código do sistema (.ktext 0x80000000) (1.0). Modifique adequadamente o Caminho de Dados e o Bloco Controlador de modo a implementar as instruções privilegiadas do Modo Kernel lwu e swu, capazes de ler e escrever na memória de código do usuário (.text 0x00400000).

Dica: O Modo Kernel pode ser automaticamente setado ao executar instruções lidas do segmento .ktext.

a)(1.0) lwu $rt,Imm($rs) #Instrução tipo-I Opcode=0x44 $rt=MemCodeUser[$rs+SignExtImm]

b)(1.0) swu $rt,Imm($rs) #Instrução tipo-I Opcode=0x45 MemCodeUser[$rs+SignExtImm]=$rt

Obs.: Caso o processador não esteja em Modo Kernel, as instruções devem ser tratadas como nop.

3) (5.0) Seja o Caminho de Dados e o Bloco Controlador do MIPS PIPELINE fornecidos em anexo, modifique-os adequadamente **E** determine as condições dos blocos Forward e Hazard de modo a detectar possíveis hazards e acelerar (caso seja possível) a execução das seguintes sequências de instruções MIPS:

a)(1.0)
```
...
sw $t0,0($t1)
lw $t2,0($t1)
...
```

b)(1.5)
```
...
addi $t0,$t0,0x1234
ori $t1,$t0,0x5678
...
```

c)(1.5)
```
...
lui $t0,0xFFFF
addi $t0,$t0,0x1234
...
```

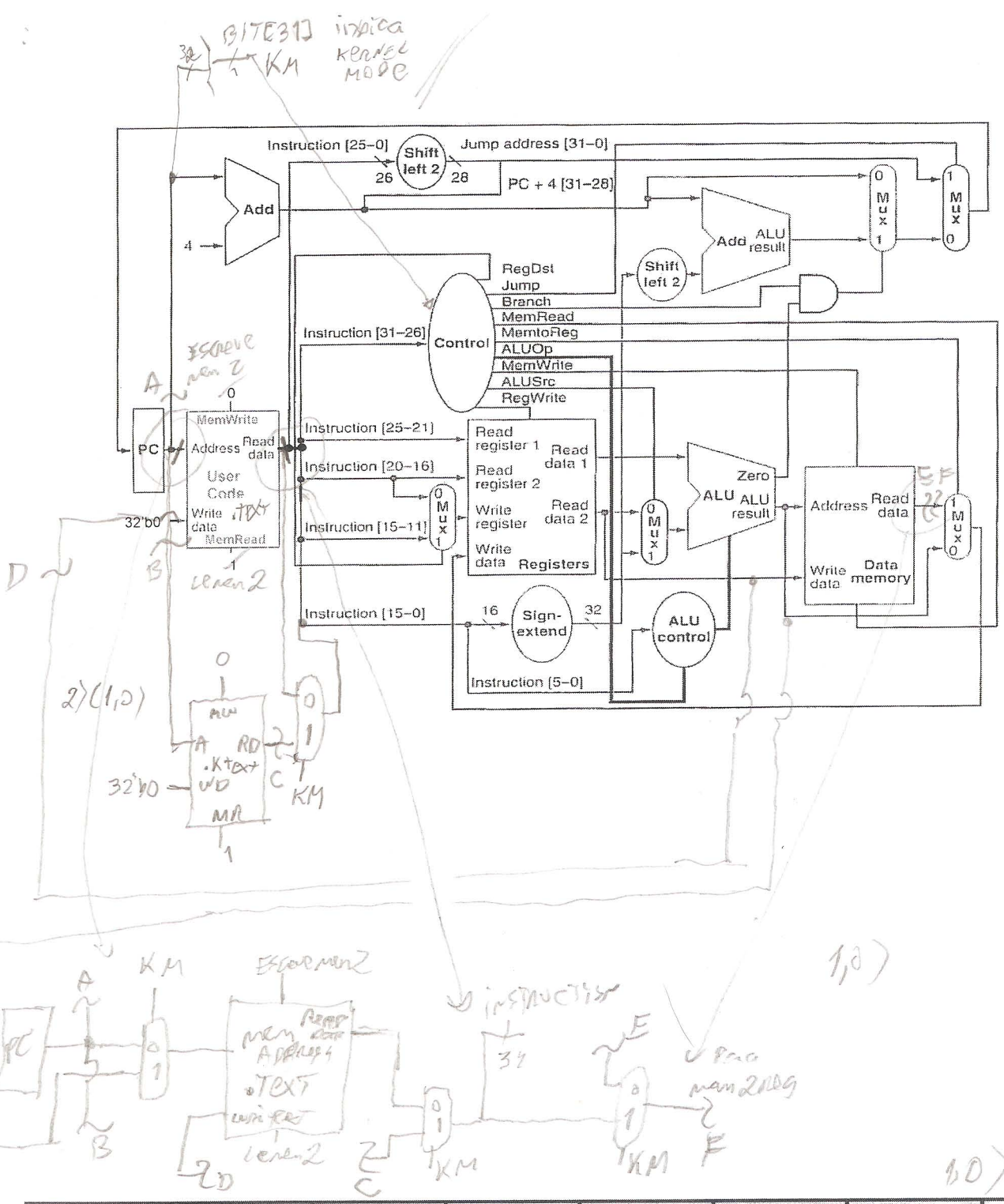
d)(1.0)
```
...
lw $t0,0($t1)
beq $t0,$t2,LABEL
...
```

4) (1.0) Supondo que você dispõe de 300Kibits para construir uma memória cache diretamente mapeada. Qual o percentual máximo dessa memória poderá ser usada para dados, caso se use blocos de 8 words e endereçamento a byte de 36 bits?

Boa Sorte!!!

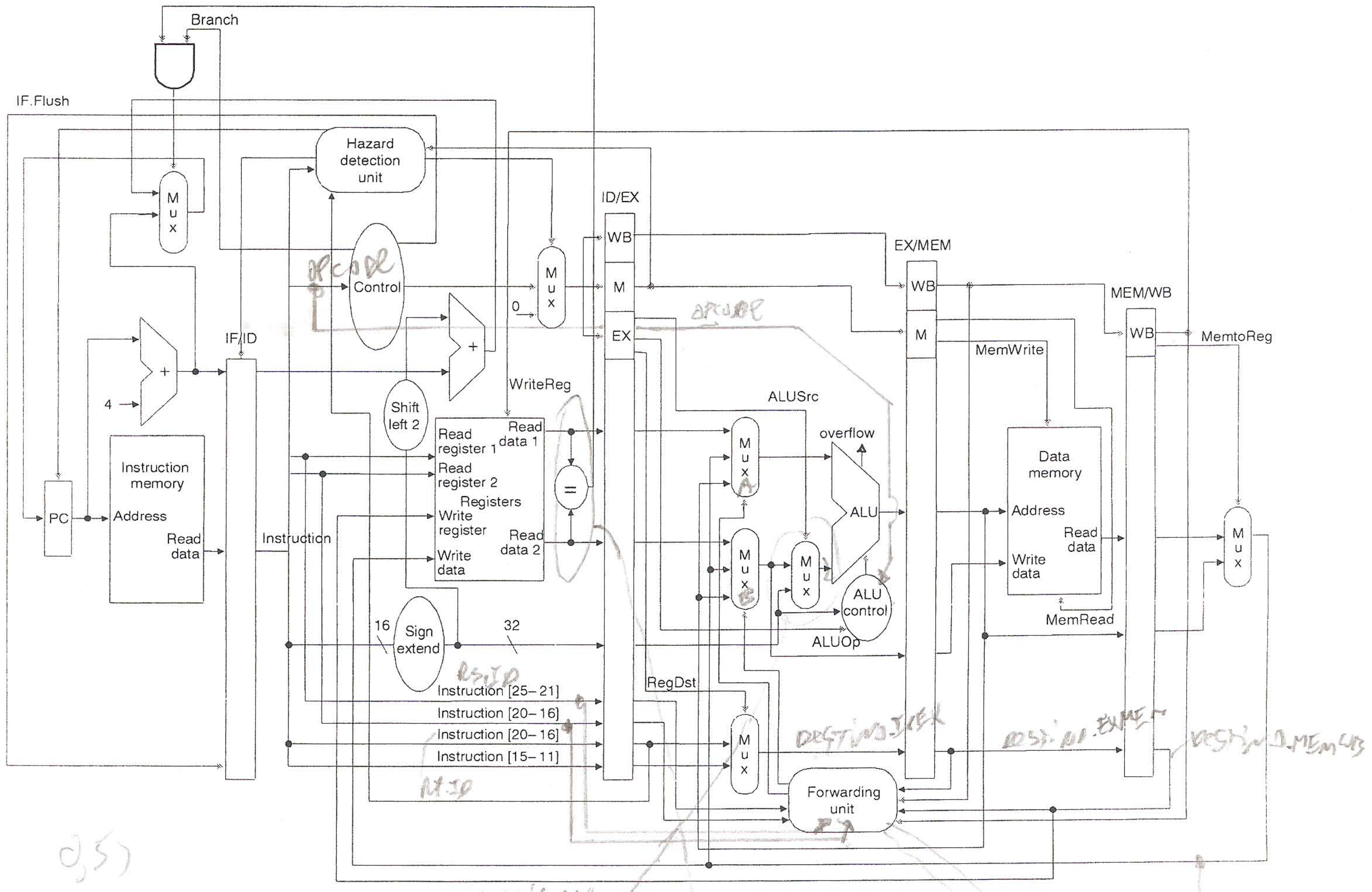
```
1:
2: .ktext
3:
4: #############  RX
5:
6: UART_RX: la $t0,0xFFFF0200   # endereco da UART
7:  li $a0,0        # onde deve estar o resultado
8:
9: LOOP1:   lw $t1,0($t0)           # carrega os bits RX e TX
10:     andi $t1,$t1,0x0001 # seleciona apenas o bit 0 (RX)
11:     bne $t1,$zero,LOOP1     # espera ate bit0==0  bit de start
12:
13: ### MIPS Uniciclo com 50MHz, necessário esperar inicialmente 7.5us apos o start bit pa
ra capturar o meio do primeiro bit de dados
14: ### logo necessita aguardar 7.5*50 = 375 ciclos, porem cada iteracao do laco correspon
te a 2 instrucoes
15:     li $t2,187          # 1+2*$t2 = 375   logo $t2 = 187
16: LOOP2:  addi $t2,$t2,-1
17:     bne $t2,$zero,LOOP2
18:
19:     li $t3,0        ### posicao do bits a receber
20: RXBIT:  lw $t1,0($t0)       # <<<<<<<<< le um bit
21:     andi $t1,$t1,0x0001     #seleciona o bit 0
22:     sllv $t1,$t1,$t3    ### desloca o bit recebido para a posicao correta
23:     or $a0,$a0,$t1      ### coloca bit deslocado em $a0
24:     addi $t3,$t3,1      # proxima localizacao do bit
25:     slti $t5,$t3,32     ## eh o ultimo bit ?
26:     beq $t5,$zero, FIM      # sim: Leu os 32 bits vai para FIM, nao: continua
27: ###### Necessita esperar 5 us depois da cada bit lido, 5 * 50 = 250 ciclos
28:     nop             ### so para ficar exata a divisao :)
29:     li $t2,121          # 6+2+2*$t2 = 250   logo $t2 = 121
30: LOOP3:  addi $t2,$t2,-1
31:     bne $t2,$zero,LOOP3
32:     j RXBIT         # volta a ler outro bit
33:
34: FIM:    eret            # FIM retorna do Syscall !!!!
35:
36:
37:
38:
39:
40:
41:
42:
43:
44:
45:
46:
47:
48:
49:
```

```
50: #############  TX
51:
52: UART_TX:   la $t0,0xFFFF0200    ## endereco da UART
53:     li $t1,2            ##### TX eh o segundo bit
54:
55:     li $t2,50       ### Gera 5us = 250 ciclos de startbit  5*$t2=250  logo $t2=50
56: ##### Cria start bit
57: START:  sw $t1,0($t0)       # coloca bit1 = 1
58:     addi $t2,$t2,-1
59:     nop             ### So para fazer a divisao exata
60:     nop
61:     bne $t2,$zero,START #Loop do bit de start
62:
63:     nop             ### para que fique um loop com 8 instrucoes abaixo
64:     nop
65:     nop
66:     li $t3,0        ### posicao do bit a enviar
67: TXBIT:  andi $t1,$a0,0x0001     ## seleciona o bit menos significativo bit0
68:     sll $t1,$t1,1       ## coloca na 2a posicao
69:
70:     li $t2,81       # envia por 5us=250 ciclos   7+3*$t2 = 250 logo  $t2=81
71: LOOP4:  sw $t1,0($t0)       ## envia bit1
72:     addi $t2,$t2,-1
73:     bne $t2,$zero,LOOP4 # loop de transmissao do bit
74:
75:     addi $t3,$t3,1      #proxima
76:     slti $t5,$t3,32     # eh o ultimo bit ?
77:     srl $a0,$a0,1       #Coloca proximo bit a transmitir no bit0
78:     bne $t5,$zero,TXBIT # Nao: volta para o loop de transmissao   sim: continua
79:
80:     li $t1,2            ##### TX eh o segundo bit
81:     li $t2,100      ### Aguarda 10us = 500 ciclos de stop bits  5*$t2=500  logo $t2 =
100
82: ##### Cria os dois stop bits
83: STOP:   sw $t1,0($t0)       ### coloca TX em alto para os stops bits
84:     addi $t2,$t2,-1
85:     nop             ### So para fazer a divisao exata
86:     nop
87:     bne $t2,$zero,STOP  ## fim da transmissao dos stops bits?
88:
89:     eret            # retorna do Syscall!!!
```



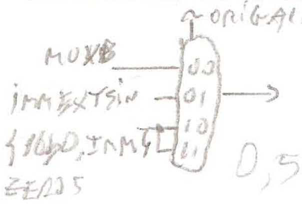
Instrução	RegDst	OrigALU	Mempara Reg	Escreve Reg	Le Mem	Escreve Mem	Branch
formato R	1	0	0	1	0	0	0
lw	0	1	1	1	1	0	0
sw	X	1	X	0	0	1	0
beq	X	0	X	0	0	0	1
LWM6KM	0	1	1	1	0	0	0
SWM6KM	X	1	X	0	0	0	0
LWM6KM	X	X	X	0	0	0	X
SWM6KM	X	X	X	0	0	0	X

Instrução	ALUOp1	ALUOp0	le mem2	Escreve mem2				
formato R	1	0	0	0				
lw	0	0	0	0				
sw	0	0	0	0				
beq	0	1	0	0				
LWM6KM	0	0	1	0				
SWM6KM	0	0	0	1				
LWM6KM	X	X	0	0				
SWM6KM	X	X	0	0				



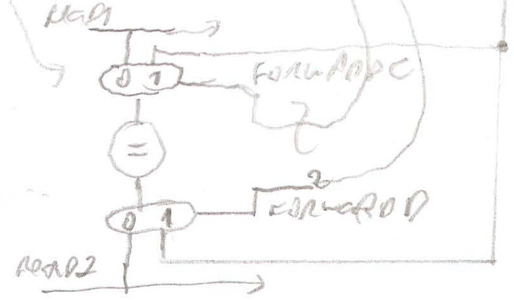
0,5)

OPALU=11
↳ OPCODE define a operação da ULA



0,5)

- Acrescentar na ULA a operação
saída = {B[15:0], 16'b0}
// opcode do lui



0,5)

Instrução	Linhas de controle do estágio de cálculo de endereço/execução						
	RegDst	OpALU1	OpALU0	OrigALU			
Formato R	1	1	0	00			
lw	0	0	0	01			
sw	X	0	0	01			
beq	X	0	1	00			
addi	0	1	1	01			
ORi	0	1	1	10			
LUi	0	1	1	11			

↳ 11 [illegible]

0,5) 0,5)

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			
addi	0	0	0			
ORi	0	0	0			
LUi	0	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			
addi	1	0			
ORi	1	0			
LUi	1	0			

3)

a) `sw $t0,0($t1)`
`lw $t2,0($t1)` → Possibilidade de Hazard na escrita → leitura do mesmo endereço da memória?
NÃO HÁ! (1,0)

b) `addi $t0, $t0, 0x1234` (Rx, Rs)
`ori $t1, $t0, 0x5678` (Rx, Rs)
→ HAZARD DADOS em $t0
Condição do FORWARD: já tem os MUXs necessários
IF(DESTINO.EXMEM ≠ 0 E ESCREVEREG.EXMEM == 1)
IF(DESTINO.EXMEM == RT.IDEX)
FORWARDA = 10 (0,5)

c) `lui $t0, 0xFFFF` (Rx)
`addi $t0, $t0, 0x1234` (Rx, Rs)
→ IGUAL ao anterior ↑ (0,5)

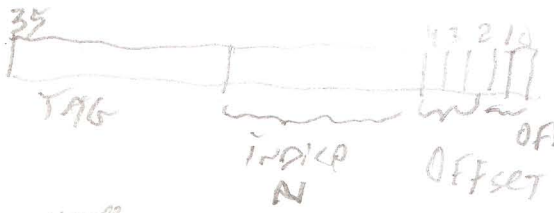
d) `lw $t0,0($t1)`
`beq $t0, $t2, LABEL` (Rs, Rx)
Precisa ADICIONAR os MUXs C e D
→ HAZARD DADOS $t0
Estágio MEM para estágio ID
entrada do comparador
→ 2 Bolhas.
IF(DESTINO.MEMWB == Rs.ID)
FORWARDC = 1 ELSE = 0 (0,5)

IF(DESTINO.MEMWB == RT.ID)
FORWARDD = 1 ELSE = 0

4) MEM TOTAL (MÁXIMA) = 300 Kibits

BLOCO = 8 x 32 = 256 BITS

Endereço = 36 bits



$$\begin{cases} (256 + TAG + 1) \times 2^N <= 300 \times 2^{10} \\ TAG = 36 - N - 3 - 2 = 31 - N \end{cases}$$

$257 + 31 - N <= 300 \times 2^{10-N}$

$288 - N <= 300 \times 2^{10-N}$

N deve ser inteira não negativa!

como resolver??

Se $N = 10$

$288 - 10 <= 300 \times 2^{10-10} \rightarrow 278 <= 300 \rightarrow \surd$

Se $N = 9$

$288 - 9 <= 300 \times 2^{10-9} \rightarrow 279 <= 600 \rightarrow \surd$

⋮

Se $N = 11$

$288 - 11 <= 300 \times 2^{10-11} \rightarrow 277 <= 150 \rightarrow F$

Logo $N_{MAX} = 10$

Assim: $TAM_{MAX\ DADOS} = 2^{10} \times 256 = 256$ Kibits

% MÁXIMA DE DADOS $= \frac{256K}{300K} = 85{,}33\%$ 1,0)

Universidade de Brasília
Departamento de Ciência da Computação
Disciplina: CIC 116394 – Organização e Arquitetura de Computadores
Prof. Marcus Vinicius Lamar

Nome: _____ Matrícula: _____

a)
sw $t0, 0($t1)
lw $t2, 0($t1)
- Não Há HAZARD
- Não precisa modificar o caminho de dados

b)
addi $t0, $t0, 0x1234
ori $t1, $t0, 0x5678
HAZARD em $t0
FORWARD EX/MEM → ULA
→ Necessita implementar instruções tipo-I

c) lui $t0, 0xFFFF
addi $t0, $t0, 0x1234

d) lw $t0, 0($t1)
beq $t0, $t2, LABEL
vs