Universidade Federal do Espírito Santo

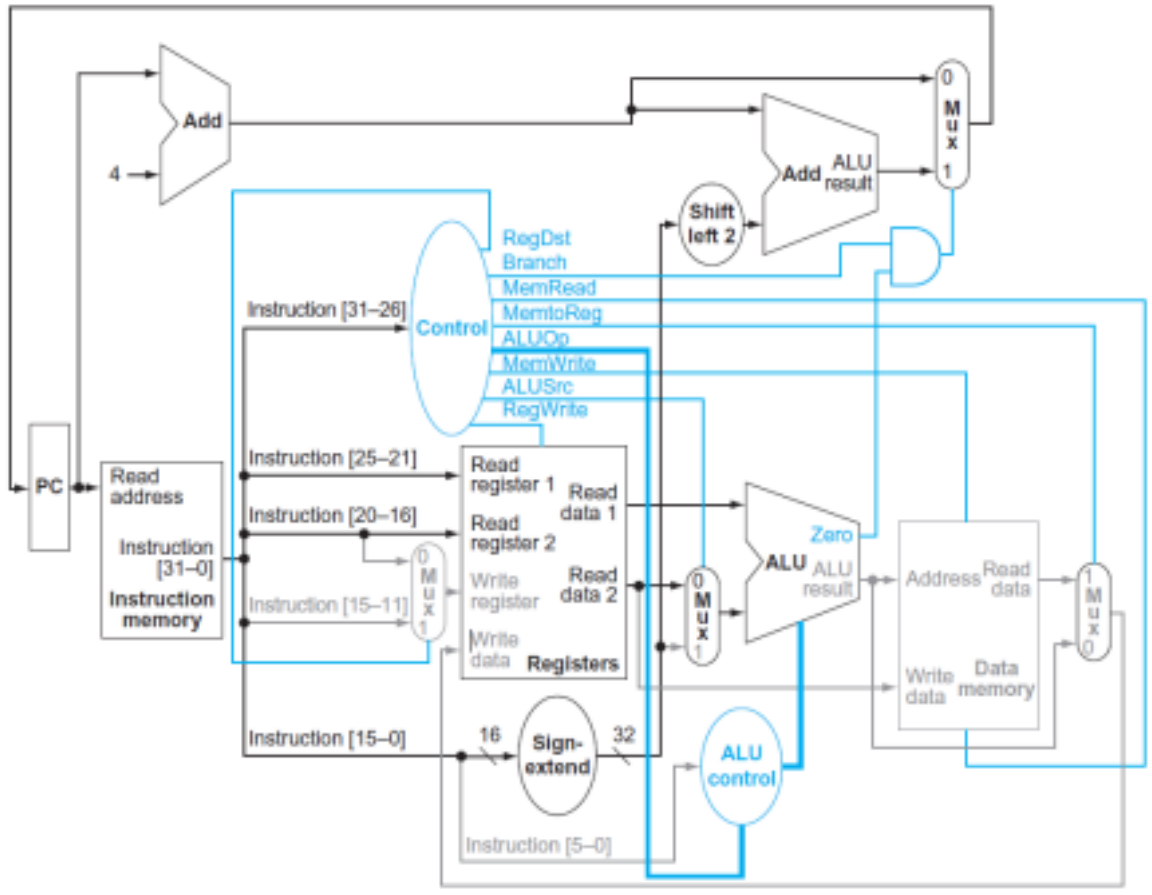
CENTRO TECNOLÓGICO
DEPARTAMENTO DE INFORMÁTICA

Arquitetura de Computadores I – Turmas 01 e 02 (EARTE) – 2021/2
Prof. Rodolfo da Silva Villaça – rodolfo.villaca@ufes.br
Prova P2 – 22 de março de 2022

Nome: Dionatas Santos Brito
Matrícula: 2019202307

Para esta prova considere que o seu número de matrícula na UFES pode ser representado pelo formato 20******XY_{10}, sendo que XY corresponde a um valor inteiro no intervalo [0..99].

1ª Questão – (2,5 pontos) – Considere o programa prova2.asm em linguagem de montagem MIPS em 32 bits, cujos segmentos de código (.text) e de dados (.data) da memória do computador MARS são mostrados a seguir. Considere também que o programa dado será inicialmente executado na CPU que segue o esquema abaixo (Figura 4.21 do livro texto – 5ª Edição). Note que esta CPU não dá suporte à instrução jump label diretamente, mas um comportamento similar pode ser obtido com uma instrução do tipo beq $0, $0, label, como no programa dado.



```
#prova2.asm

 .data
count: .word XY #insira os dígitos da sua matrícula aqui!
res: .space 4
numbers:.word 1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33,
35, 37, 39, 41, 43, 45, 47, 49, 51, 53, 55, 57, 59, 61, 63, 65, 67, 69, 71,
73, 75, 77, 79, 81, 83, 85, 87, 89, 91, 93, 95, 97, 99, 101, 103, 105, 107,
109, 111, 113, 115, 117, 119, 121, 123, 125, 127, 129, 131, 133, 135, 137,
139, 141, 143, 145, 147, 149, 151, 153, 155, 157, 159, 161, 163, 165, 167,
169, 171, 173, 175, 177, 179, 181, 183, 185, 187, 189, 191, 193, 195, 197,
199, 201, 203, 205

 .text
  la $s0, numbers
  lw $s1, count
  addi $s1, $s1, 1
  addi $t2, $t2, 1
sum:
  bge $t2, $s1, end
  addi $t2, $t2, 1
  lw $t0, 0($s0)
  add $t1, $t1, $t0
  addi $s0, $s0, 4
  beq $zero, $zero, sum
end:
  sw $t1, res
  add $t2, $zero, $zero
  add $s1, $t2, $zero
  add $s0, $s1, $zero
```

Para dar maior flexibilidade ao programador MIPS, foi decidido que nossa CPU deveria permitir a mudança no fluxo de execução de um programa MIPS de forma similar a um "jump reg" (salta para o endereço indicado no registrador reg). Usando uma estratégia similar ao que foi feito para a instrução jump label, modifique a CPU monociclo dada (via de dados e sinais de controle) para que ela suporte a instrução beq $0, $0, reg, sendo reg o número do registrador (codificado nos bits 15-11 da instrução) que irá conter o endereço do desvio.

Explique e identifique todas as mudanças necessárias no controle e na via de dados para que a CPU suporte a nova instrução.

2ª Questão – (2,5 pontos) – Considere agora que o programa prova2.asm vai ser executado numa CPU pipeline conforme ilustrada na figura a seguir (Figura 4.46 do livro texto – 5ª Edição). Note que ela não tem tratamento de conflitos (hazards), nem permite adiantamento de dados. Também não há nem um hardware extra para permitir a realização do teste de igualdade dos valores dos registradores lidos no 2º ciclo do pipeline (para reduzir a penalidade do beq).

a) (1,5) Indique todos os possíveis conflitos no código dado e as bolhas que seriam geradas durante a execução. Insira as instruções nop (bolhas) necessárias para tratar os conflitos de forma que o código continue sendo executado corretamente.

Código com instruções nop (bolhas) necessárias para tratar os conflitos estão no código abaixo:

```
-  la $s0, numbers        #(linha 1)
-  lw $s1, count          #(linha 2)
-  nop
-  nop
-  addi $s1, $s1, 1       #(linha 3)
-  addi $t2, $t2, 1       #(linha 4)
-  sum:                    #(linha 5)
-  #bge $t2, $s1, end     #(linha 6)
-  nop
-  nop
-  slt $1, $10, $17
-  beq $1, $0, end
-  nop
-  nop
-  nop
-  addi $t2, $t2, 1       #(linha 7)
-  lw $t0, 0($s0)         #(linha 8)
-  nop
-  nop
-  add $t1, $t1, $t0      #(linha 9)
-  addi $s0, $s0, 4       #(linha 10)
-  beq $zero, $zero, sum  #(linha 11)
-  end:                    #(linha 12)
-  sw $t1, res            #(linha 13)
-  add $t2, $zero, $zero  #(linha 14)
-  nop
-  nop
-  add $s1, $t2, $zero    #(linha 15)
-  nop
-  nop
-  add $s0, $s1, $zero    #(linha 16)
```

Conflitos estruturados:

- `bge $t2, $s1, end` `#(linha 6)`
- `addi $t2, $t2, 1` `#(linha 7)`

  Nessas linhas o registrador $t2 é usado tanto para a instrução condicional (bge) e a instrução de operação (addi).

Conflitos de dados:

- `lw $s1, count` `#(linha 2)`

Na linha 2 a valor de count é registrada em $s1 e em seguida esse ele é usado para fazer a adição traduzida na linguagem c (var = var + 1), com base no pipeline de 5 estágios, que foi estudada em sala de aula, podemos ver que vai da conflito pois a execução do lw ainda não acabou.

- `lw $t0, 0($s0)` `#(linha 8)`
  - Nessa linha um valor do vetor é armazenado em $t0 e logo em seguida usado no comando de adição add.

- `add $t2, $zero, $zero` `#(linha 14)`
- `add $s1, $t2, $zero` `#(linha 15)`
- `add $s0, $s1, $zero` `#(linha 16)`

Nessas linhas os três registradores são zerados antes de acabar o programa, porém os registradores são usados sequencialmente nas operações, logo fazendo o erro

Conflitos de controle:

- `bge $t2, $s1, end` `#(linha 6)`
- `beq $zero, $zero, sum` `#(linha 11)`

Nessas linhas, em caso de saltos há necessidade de esvaziar o pipeline e zerar os comandos de controle dos 2 ciclos seguintes.

b) (1,0) Qual o ganho de desempenho para a execução do programa Prova2.asm corrigido nesta CPU pipeline comparada com a CPU monociclo da questão anterior, considerando que o clock da CPU monociclo é igual ao clock de cada um dos estágios da CPU pipeline?

Dado o tempo do monociclo ser igual a:
sendo a quantidade de instrução 18(contando as sub instruções)

$T_{mono} = (CicloClock/\ qtdInstrução)$
=C/18

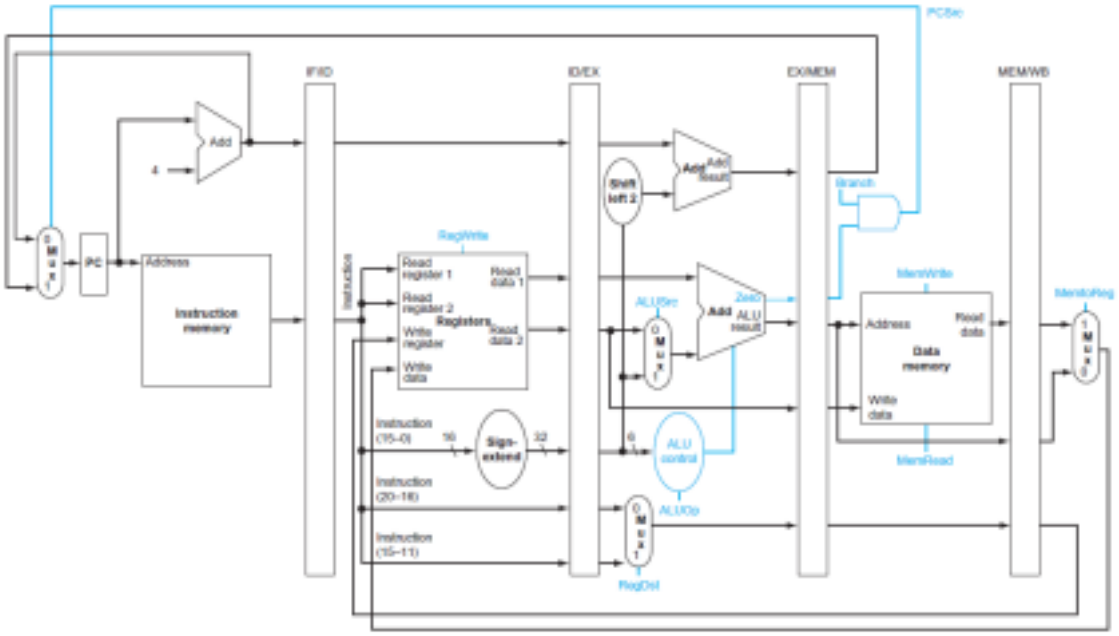
Pipeline igual a:
Sendo a quantidade de instruções 27(contando os nops e sub instruções)

$T_{pipe} = (CicloClock/\ qtdInstrução)/5$
=C/5,4

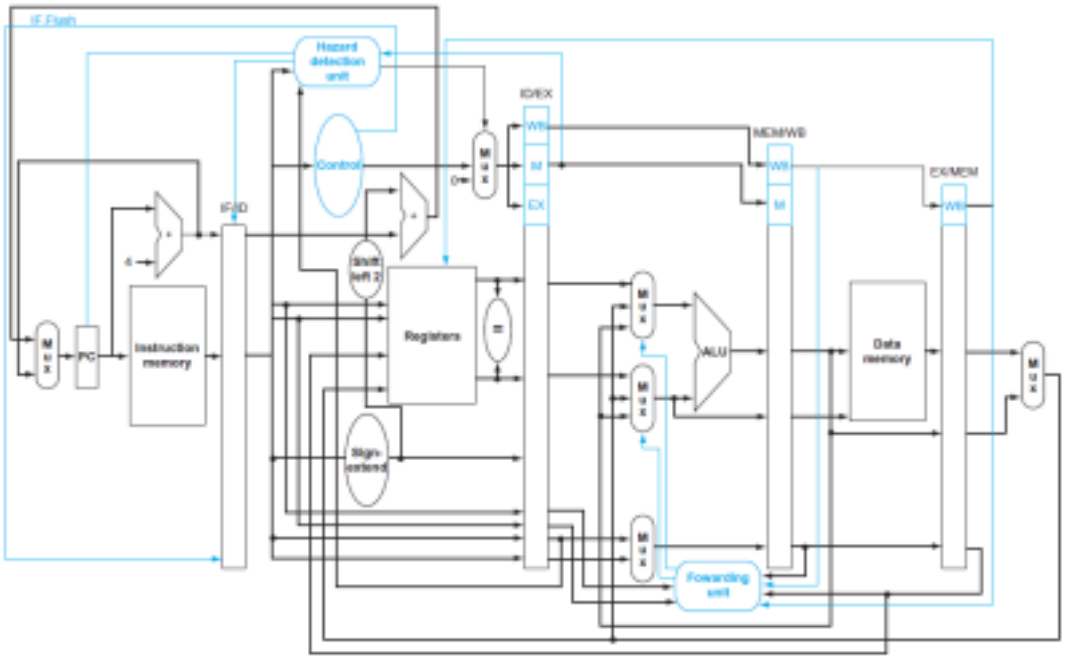
Ganho:

Ganho(%) =[(C/18) / (C/5,4)]*100
=0,3*100
=30%

Executando o programa *prova2.asm* na CPU monociclo 419.430ps obtivemos um tempo de execução de e na CPU pipeline no DrMips, obtivemos um tempo de execução de 400400 . Assim teremos um ganho = 419.430/400400 = 1,047 vezes. Está batendo o limite de ciclos



3ª Questão – (2,0 pontos) – Considere agora que o programa prova2.asm vai ser executado numa CPU pipeline conforme ilustrada na figura a seguir (Figura 4.65 do livro texto – 5ª Edição). Note que ela tem suporte para tratamento de conflitos (hazards) e para adiantamento de dados, além de possuir um hardware extra que permite a realização do teste de igualdade dos valores dos registradores lidos no 2º ciclo do pipeline (para reduzir a penalidade do beq). Qual o ganho de desempenho para a execução do programa Prova2.asm na CPU pipeline comparada com a CPU monociclo dada, considerando que o clock da CPU monociclo é igual ao clock de cada um dos estágios do pipeline?



Executando o programa *prova2.asm* na CPU monociclo 419.430ps obtivemos um tempo de execução de e na CPU pipeline no DrMips, obtivemos um tempo de execução de 284.400ps. Assim teremos um ganho = 419.430/284.400 = 1,475 vezes.

4ª Questão – (3,0 pontos) – Considere o programa prova3.asm em linguagem de montagem MIPS em 32 bits, cujos segmentos de código (.text) e de dados (.data) da memória do computador MARS são mostrados a seguir. Considere também que o programa dado será inicialmente executado numa CPU MIPS similar àquela projetada durante o curso, mas considerando que as memórias de instruções e de dados são, de fato, duas caches. Considere ainda que a CPU só acessa estes caches e que estas últimas acessam a memória principal.

```
## Prova3.asm
#data segment

 .data
count: .word XY #Alterar para XY da matrícula
res: 0
numbers:  .word 1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27,
29, 31, 33, 35, 37, 39, 41, 43, 45, 47, 49, 51, 53, 55, 57, 59, 61,
63, 65, 67, 69, 71, 73, 75, 77, 79, 81, 83, 85, 87, 89, 91, 93, 95,
97, 99, 101, 103, 105, 107, 109, 111, 113, 115, 117, 119, 121, 123,
125, 127, 129, 131, 133, 135, 137, 139, 141, 143, 145, 147, 149,
151, 153, 155, 157, 159, 161, 163, 165, 167, 169, 171, 173, 175,
177, 179, 181, 183, 185, 187, 189, 191, 193, 195, 197, 199, 201,
203, 205, 207, 209, 211, 213, 215

#text segment
 .text
  la $s0, numbers
  lw $s1, count
  addi $s1, $s1, 1
  add $t2, $0, $0
sum:
  bge $t2, $s1, end
  addi $t2, $t2, 1
  lw $t0, 0($s0)
  add $t1, $t1, $t0
 addi $s0, $s0, 4
  beq $zero, $zero, sum
end:
  sw $t1, res
```

a) (2,0) Considere que as memórias de dados e de instruções do diagrama da CPU são, de fato, duas caches de 64 blocos com 4 palavras/bloco, com mapeamento direto e política LRU de substituição de blocos. Mostre o esquema de implementação (indicando bits de índice/linhas, tags/rótulos, dados, etc.) da cache de dados e o seu estado (o que está armazenado em cada posição da cache) quando a última instrução do programa prova3.asm estiver sendo executada para sua entrada de dados XY.

Entrada de dados XY = 07.
Representando um (1) endereço MIPS de 32 bits :

Tag (22 bits)	Index (6 bits)	Palavra (2 bit)	Offset (2 bits)

Descrição:

- [Tag] Identifica o prefixo do endereço armazenado na memória cache;
- [Index] Representa o bloco (ou linha) de memória cache que vai armazenar o endereço de 32 bits, pois estamos considerando endereços de 32 bits (MIPS). 2^6 = 64 blocos;
- [Palavra] Considerando que em cada bloco (linha) de memória cache serão armazenados 4 palavras, o bit Palavra serve para representar qual conjunto de 4 Bytes (tamanho de uma palavra na memória principal do projeto do MIPS) foi referenciado. 2^2 = 4 Palavras/Bloco (ou linha);
- [Offset] Serve para juntar 4 Bytes em uma mesma palavra. 2^2 = 4 Bytes/Palavra.

Considerando 1 bit de validade (V) para cada linha da memória cache, 22 bits de Tag, e cada linha contendo 4 palavras de 32 bits, o conteúdo de cada linha da memória cache é:

index (6 bits)

V (1 bit)	Tag (22 bits)	Palavra 0 (32 bits)	Palavra 1 (32 bit)	Palavra 2 (32 bits)	Palavra 3 (32 bits)

- Total de bits por bloco (ou linha) da memória cache é:
  - 1(V) + 22(Tag) + 32(Palavra 0) + 32(Palavra 1) + 32(Palavra 2) + 32(Palavra 3) = 151 bits/bloco
- Total de bits da memória cache é:
  - = 151 bits/bloco * 64 blocos = 9664 bits

O bit Offset seleciona a entrada de um MUX 4:1, selecionando Palavra 0-3 (Palavra1, Palavra1, Palavra2, Palavra3) de acordo com o endereço referenciado. Para um determinado bloco (ou linha) da memória cache (por mapeamento direto) a Tag armazenada na memória cache é comparada com os 22 bits mais significativos do endereço referenciado para identificar um cache hit ou um cache miss. O resultado dessa comparação também é comparado ao bit V. Se V=0, sempre haverá um miss.

Considerando que a variável count ocupa a posição #0 da memória de dados, ou

0x10010000, e a variável res ocupa a posição #1 da memória de dados, ou 0x10010004, acessos aos 62 primeiros números (numbers) ocuparão a cache sem conflito. Após isso haverão conflitos em cada acesso, começando o processo de substituição a partir da posição #0 (0x10010000). No meu caso, count = 7 (count=XY) o que ocasionará 8 loops (count+1) e todos os 8 primeiros números ficarão armazenados na memória cache

$a1	5	0
$a2	6	0
$a3	7	0
$t0	8	13
$t1	9	49
$t2	10	0
$t3	11	0
$t4	12	0
$t5	13	0

#bloco	V	Tag (23 bits)	Palavra 0 (32 bits)	Palavra 1 (32 bits)	Palavra 2 (32 bits)	Palavra 3 (32 bits)
0	1	0x100100	count	res	1	3
1	1	0x100100	5	7	9	11
2	1	0x100100	13	*	*	*

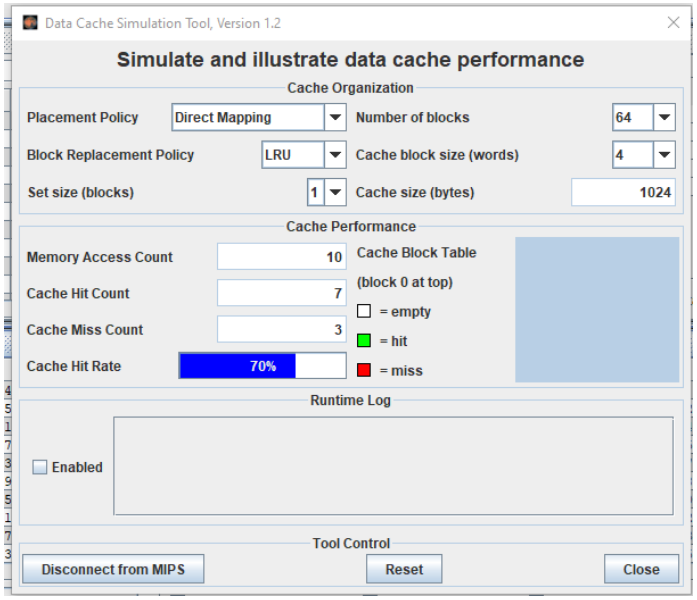
Data Segment

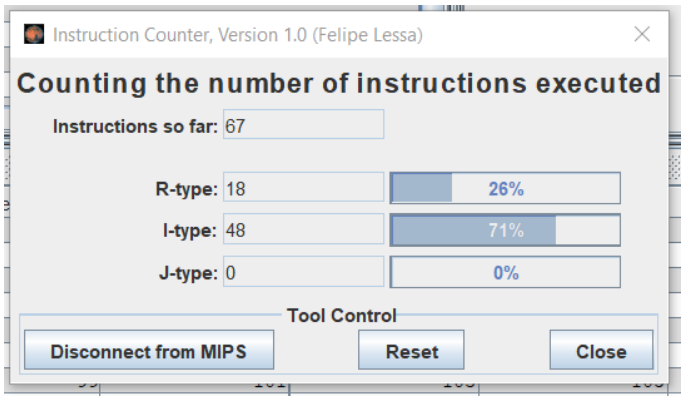
Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	7	0	1	3	5	7	9	11
0x10010020	13	15	17	19	21	23	25	27
0x10010040	29	31	33	35	37	39	41	43
0x10010060	45	47	49	51	53	55	57	59
0x10010080	61	63	65	67	69	71	73	75
0x100100a0	77	79	81	83	85	87	89	91
0x100100c0	93	95	97	99	101	103	105	107
0x100100e0	109	111	113	115	117	119	121	123
0x10010100	125	127	129	131	133	135	137	139
0x10010120	141	143	145	147	149	151	153	155
0x10010140	157	159	161	163	165	167	169	171
0x10010160	173	175	177	179	181	183	185	187
0x10010180	189	191	193	195	197	199	201	203
0x100101a0	205	0	0	0	0	0	0	0
0x100101c0	0	0	0	0	0	0	0	0
0x100101e0	0	0	0	0	0	0	0	0

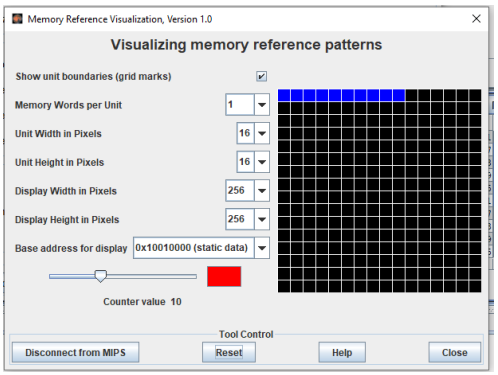
0x10010000 (.data) Hexadecimal Addresses Hexadecimal Values ASCII

b) (1.0) Calcule a taxa média de falhas da cache de dados quando a execução do programa prova3.asm estiver sendo executada para sua entrada de dados XY. Indique os cálculos feitos para chegar à sua resposta.

Resposta:
Segundo o MIPS e para a matrícula XY = 07 temos que a taxa de acertos para o cache de dados (Cache Hit Rate) é igual a 70%.







Como a taxa de falhas é dada por

1 - 100%
taxa de falhas - 70%

taxa de falhas = 30%

Logo a taxa de acertos é igual a 30%, pois se trata de um cache de 4 palavras por bloco, onde todo acesso a 1 endereço, como o acesso de memória sequencial, em com base nos 3 endereços seguintes, a cada 1 falha (primeiro acesso) tenho 3 acertos (acesso seguinte). Logo a taxa de falhas média é próxima de 25% e taxa de acertos próximo de 75%.