UFFS – Universidade Federal da Fronteira Sul
Curso de Ciência da Computação
Disciplina: Organização de Computadores
Professor: Luciano L. Caimi **Data: 12/07/2023**
Aluno: _____ **Nota: _____**

1. (**3,0**) Suponha um sistema computacional com um espaço de endereçamento de 64 Kcélulas. A cache associada a este sistema possui 32 linhas. Sabendo que cada linha contém 32 células, pergunta-se:

MP = B * K

64K = B * 32

$2^{16} = B * 2^5$

$B = 2^{26} / 2^5 \Rightarrow 2^{16-5} \Rightarrow 2^{11} = 2048$ blocos

ou

B = 64 Kcélulas / 32 = 2 Kblocos = 2048 blocos

a) Qual é o número do bloco correspondente ao endereço $ABCD_{16}$?

$End = ABCD_{16} = 1010101111001101_2 = 43981_{10}$

Nro Bloco = End / K = 43981 / 32 = 1374

ou dado o endereço em binário de 16 bits: 1010101111001101

Dos 16 bits de endereço teremos:

Número do Bloco	Deslocamento no bloco

Nro do bloco usa 11 bits ($2^{11} = 2048$)e deslocamento deslocamento usa 5 bits ($2^5 = 32$)

10101011110	01101

Assim:

$10101011110_2 = 1374_{10}$

b) Considerando o mapeamento direto:

b1) Qual a divisão do endereço do ponto de vista da cache? Justifique.

No mapeamento direto temos a seguinte divisão de endereçamento do ponto de vista da cache

Rótulo	Linha de destino	Deslocamento na linha

Como temos 32 células em cada linha precisamos de 5 bits para o deslocamento ($2^5 = 32$)

Como temos 32 linhas na MC precisamos de 5 bits para o linha de destino ($2^5 = 32$)

Como temos uma proporção de 64 blocos da MP para cada linha da MC (2048 / 32 = 64) precisamos de 6 bits para rótulo ($2^6 = 64$)

rótulo	linha de destino	deslocamento
6 bits	5 bits	5 bits

b2) Qual é a linha de destino do endereço $ABCD_{16}$?

$ABCD_{16} = 1010101111001101_2$

separando os bits conforme item b1

rótulo	linha de destino	deslocamento
101010	11110	01101

Convertendo 11110_2 para decimal temos: 30

Ou seja a linha de destino é a 30

Outra maneira é fazer:

linha de destino = nro do bloco MOD L

linha de destino = 1374 MOD 32 = 30

c) Considerando o mapeamento associativo por conjuntos e que o sistema acima possui 4 conjuntos:

c1) Qual a divisão do endereço do ponto de vista da cache? Justifique.

No mapeamento associativo por conjuntos temos a seguinte divisão de endereçamento do ponto de vista da cache

Rótulo	conjunto de destino	Deslocamento na linha

Como temos 32 células em cada linha precisamos de 5 bits para o deslocamento (2^5 = 32)

Como temos 4 conjuntos na MC precisamos de 2 bits para o conjunto de destino (2^2 = 4)

Como temos 2048 blocos disputando 4 conjuntos a proporção é de 512 blocos destinados a cada conjunto, precisando então de um rótulo de 9 bits (2^9 = 521)

rótulo	conjunto de destino	deslocamento
9 bits	2 bits	5 bits

c2) Qual é o conjunto de destino do endereço $ABCD_{16}$?

$ABCD_{16}$ = 1010101111001101_2

separando os bits conforme item c2

rótulo	conjunto de destino	deslocamento
101010111	10	01101

Convertendo 10_2 para decimal temos: 2

Ou seja o conjunto de destino é o conjunto 2

d) Considerando o mapeamento associativo por conjuntos, é possível haver dois rótulos idênticos (explique e exemplifique sua resposta):

d1) em linhas pertencentes ao mesmo conjuntos?

Não é possível pois isto significaria que o mesmo bloco estaria carregado na cache em duas linhas diferentes do mesmo conjunto. Por exemplo: bloco 1374 está destinado ao conjunto 2 (conforme c2), o rótulo correspondente não pode aparecer em outra linha da cache do mesmo conjunto pois uma vez que ele seja carregado, a busca pelo bloco vai indicar que ele já se encontra na cache, no conjunto 2.

d2) em linhas pertencentes a conjuntos diferentes?

Sim, pois blocos com mesmo rótulo foram destinados a conjuntos diferentes e foram carregados ao mesmo tempo na cache. Por exemplo, no mapeamento apresentado em c1

bloco 1 = 00000000000_2

bloco 2 = 00000000011_2

os dois blocos possuem mesmo rótulo mas estão destinados a conjuntos diferentes na cache (conjunto 0 e conjunto 3 respectivamente).

2. **(1,0)** Considerando a memória apresentada na questão acima, utilizando o mapeamento associativo por conjuntos com 4 conjuntos, descreva detalhadamente, a sequência de ações realizada pelo controlador da cache para fornecer o conteúdo do endereço de memória ABCD para o processador. Na descrição você deve apresentar os valores envolvidos (rótulo, conjunto, deslocamento, bloco) em cada ação realizada.

Funcionamento Básico:

1) O processador solicita o endereço $ABCD_{16}$

2) O controlador da memória cache verifica se o endereço está armazenado na cache, ou seja, vai verificar no conjunto 2 (conjunto de destino do bloco 1374) se o rótulo do endereço solicitado (rótulo 101010111) encontra-se em alguma das linhas do conjunto 2

2.1) Caso sim: ocorre um acerto (hit) - a cache entrega para o processador o endereço solicitado, neste caso, corresponde ao dado que está na célula 13 da linha (pois o deslocamento do endereço $ABCD_{16}$ é $01101_2 = 13_{10}$)

2.2) Caso não: ocorre uma falta (miss) – o controlador da cache acessa o nível inferior da hierarquia, ou seja, a memória principal, neste caso.

2.2.1) o bloco onde o endereço solicitado se encontra é carregado na cache, isto é, o bloco 1374 é lido na MP e carregado em uma linha da cache, atualizando o rótulo da linha.

2.2.2) a cache entrega o dado solicitado ao processador, neste caso, corresponde ao dado que está na célula 13 da linha (pois o deslocamento do endereço $ABCD_{16}$ é $01101_2 = 13_{10}$)

3. (**3,0)** Considerando o pipeline de 5 estágios (IF/OF/EX/MEM/WB) do RISC-V sem adiantamento, apresente a evolução das instruções do programa abaixo no pipeline usando bolhas (stalls) e eliminações (flush) para resolver os conflitos existentes.

```
      ADD  t3, t4, t2
      LW    t6, 20 (t3)
      BEQ  t6, zero, pula    # considere o beq verdadeiro
      ADDI t4, t6, -1
      J fim
pula:
      ADD t4, t6, t4
fim: nop
```

Instrução	CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8
ADD	IF	OF	EX	MEM	WB			
LW		IF	OF	-	-	EX	MEM	WB
BEQ			IF	-	-	OF	-	-
ADDI						IF	-	-

Instrução	CC9	CC10	CC11	CC12	CC13	CC14	CC15	CC16
BEQ	**EX**	**MEM**	**WB**					
ADDI	**OF**	**- FLUSH -**	**- FLUSH -**	**- FLUSH -**				
J	**IF**	**- FLUSH -**	**- FLUSH -**	**- FLUSH -**	**- FLUSH -**			
ADD		**IF**	**OF**	**EX**	**MEM**	**WB**		
NOP			**IF**	**OF**	**EX**	**MEM**	**WB**	

No CC4 tem que esperar o conflito de dados sobre t3 resolver para avançar

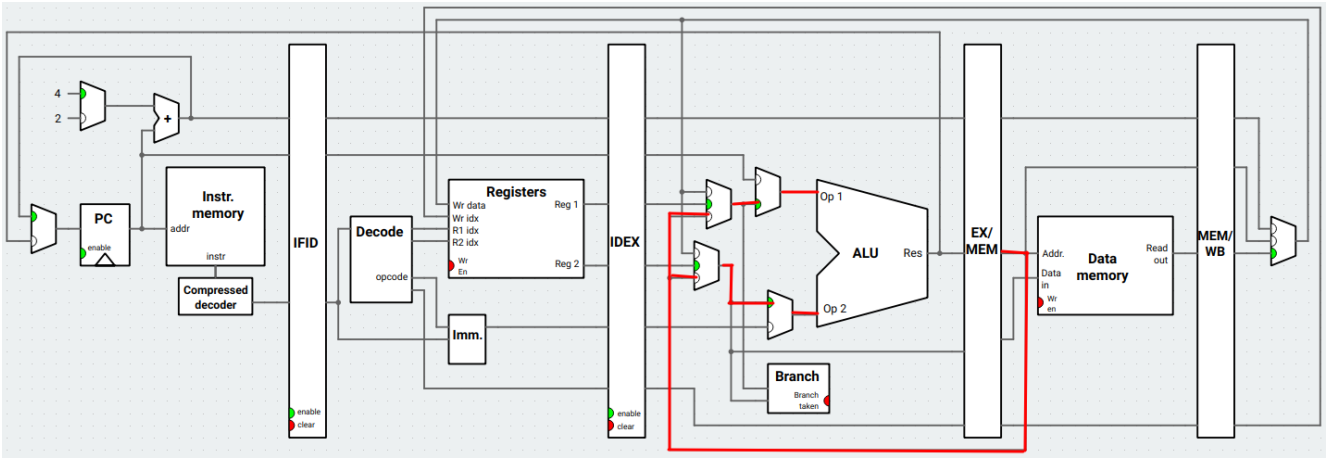
No CC7 tem que espera o conflito de dados sobre t6 resolver para avançar

No CC10 descobre que o BEQ é verdadeiro e tem que fazer o FLUSH das instruções carregadas indevidamente (ADDI e J) e carregar a instrução correta ADD

4. (**3,0)** O caminho de dados mostrado abaixo permite fazer o adiantamento (forwading) quando encontra um conflito de dados do tipo RAW.

a1) Marque na figura abaixo o caminho percorrido pelo dado quando ocorre um adiantamento na execução das instruções abaixo:

```
ADD t0, t1, t2
ADD s1, t0, t0
```

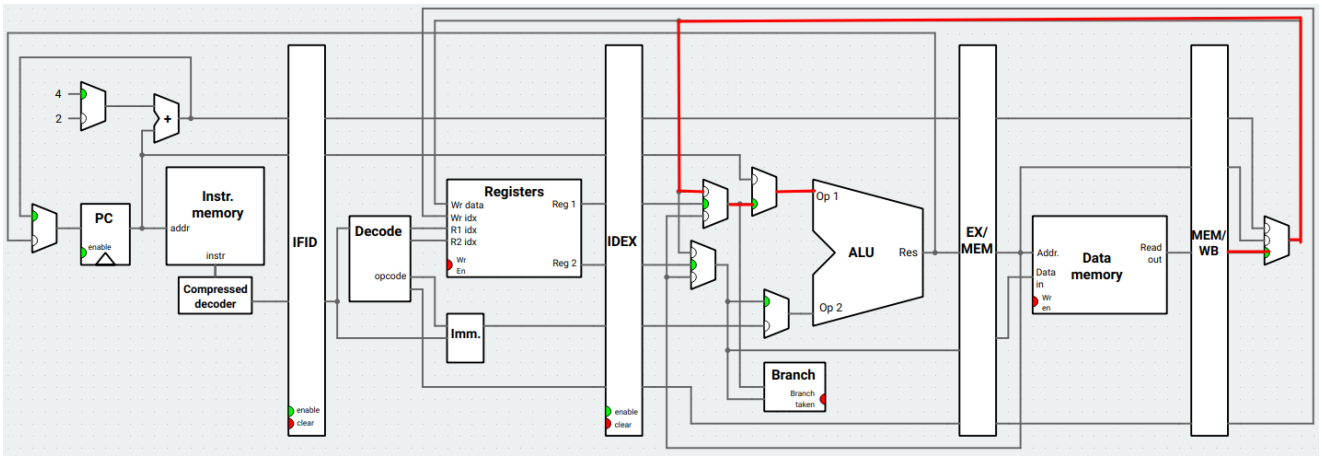


a2) Descreva o que está ocorrendo no Pipeline e o comportamento do mesmo no que diz respeito ao *forwarding*

O registrador t0 é escrito pelo primeiro ADD e lido pelo segundo ADD (tanto por RS1 quanto RS2). Quando o primeiro ADD entra no estágio de Mem. Access o novo valor de t0 já está calculado mas ainda não chegou no estágio de write back, assim é possível fazer o adiantamento do valor e fazendo com que o mesmo seja usado nas duas entradas da ULA (tendo em vista que o valor de to é usado em RS1 e RS2 no segundo ADD)

b1) Marque na figura abaixo o caminho percorrido pelo dado quando ocorre um adiantamento na execução das instruções abaixo:

```
LW  s0, 0(s2)
ADD t0, s2, s3
ADD s2, s0, t1
```



b2) Descreva o que está ocorrendo no Pipeline e o comportamento do mesmo no que diz respeito ao *forwarding*

O registrador s0 é escrito pela instrução LW e lido pela instrução (RS1 da instrução). Quando o LW entra no estágio WriteBack faz-se o adiantamento do valor de tal forma que, enquanto o valor está sendo gravado no banco de registradores ele também é adiantado como primeiro operando da ULA. Ou seja, em vez de pegar o valor antigo de s0 que vem do estágio de Busca de operando, pega-se o valor já calculado, adiantado do estágio de WriteBack.