

① Fetch $icode : ifun \leftarrow M_1[PC]$
 $1B \leftarrow M_1[PC+1]$
 $val P \leftarrow PC+1$ $VALC \leftarrow M_4[PC+2]$

Decode $valB \leftarrow R[1B]$

Execute $valE \leftarrow valC + valB$
Set CC

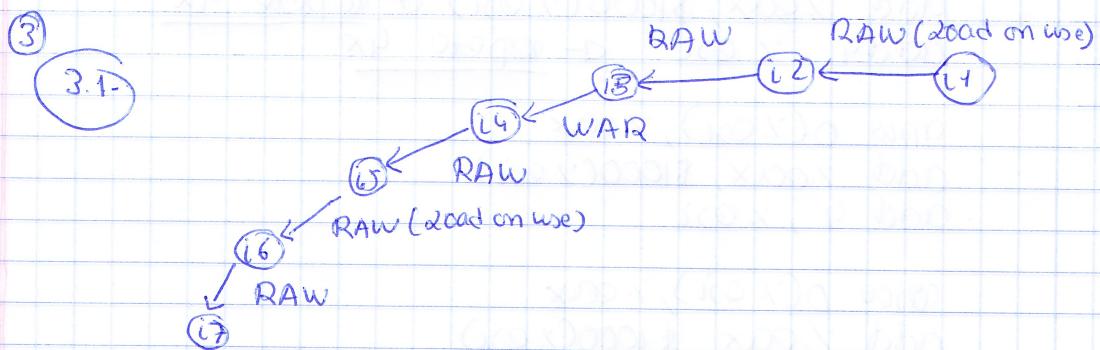
Memory (NADA)

write Back $R[1B] \leftarrow valE$

PC update $PC \leftarrow valP$

②	i ₁	FDEM _W
	i ₂	FDEM _W
	i ₃	FDEM _W
	i ₆ /Boh _a	FDE E M _W
	i ₇ /Boh _a	F D E E M _W
	i ₄	FDEM _W
	i ₅	FDEM _W
	i ₆	FDEM _W

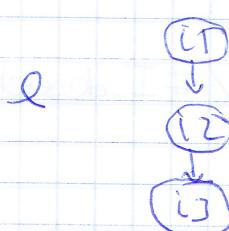
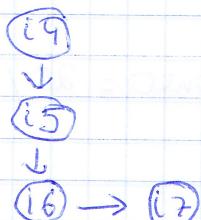
é preciso injetar bolhas pois o jump não devia ser tomado logo temos de impactar as instruções 6 e 7 de executação



R: Bonsando este esquema podemos constatar os varios tipos de dependencia de dados que temos, neste exercicio temos RAW, e WAR, através da remoção de registros podemos remover estas dependencias WAR:

- i3 - `removemov %eax, 1000(%ecx)`
 - i4 - `addl %esi, %ecx, %2`
 - i5 - `removemov 0(%a1), %edx`

Passando onto a tor o seguinte grafo:



3.1) Com o resultado obtido anteriormente é fácil fazer o desenrolamento dinâmico dessa instrução

ciclo	Aritmética / salto	Acesso à memória
1	addl %esi, %eax	memore 0(%ecx), %edx
2	addl %eax, %eax	memore 0(%eax), %edx
3	addl %eax, %eax	memore %eax, 1000(%ecx)
4	1 1 1 1 1 1	memore 0(%eax) , 1000(% ecx)%21
5		
6		
7		

2. Assim conseguimos obter executar 7 instruções em apenas 4 ciclos. Alterei ainda o registo da instrução 16 e 17 para %eax embora não fosse necessário apenas fiz porque esse é o professor na resolução de um exercício que também o fez.

4

4.1

Desde bramdo com o ciclo 4x obtemos o seguinte resultado:

mas 0(%esi), %eax & repete 4x
 add %eax, \$1000(%esi) & repete 4x
 add 4, %esi & repete 4x

mas 0(%esi), %eax
 add %eax, \$1000(%esi)
 add 4, %esi

mas 0(%esi), %eax
 add %eax, \$1000(%esi)
 add 4, %esi

mas 0(%esi), %eax
 add %eax, \$1000(%esi)
 add 4, %esi

1x \rightarrow sub 4, %eax & sub 4 em vez de 1
 1x \rightarrow gny ciclo

N = Número de instruções com o código original

$$\#I_{\text{desdobramento}} = N / 4 \times (4 \times 3 + 2)$$

$$\#I_{\text{original}} = N \times 5$$

$$\text{ganho} = \#I_{\text{original}} / \#I_{\text{desdobramento}} = 20 / 14 = \underline{\underline{1,428}}$$

4.2) O processamento vetorial tem vantagem nas aplicações em que o mesmo cálculo está a ser adicionado ou subtraído num grande com diferentes pontos de dados. Um exemplo de uma aplicação deste tipo seria alterar o brilho do ecrã visto que cada pixel é composto por três valores, vermelho, verde e azul e alterar o brilho significa adicionar ou remover sobre estes valores. Com este tipo de processamento, uma série de valores pode ser calculada de uma só vez. Com vez de uma ter uma série de instruções para obter os diferentes pixels este que se possa ter uma instrução que transfere os pixels em blocos. Outra vantagem são estes processamentos possuirem apenas instruções que podem ser aplicadas em todos os dados numa única operação. O nível de paralelismo nestes processamentos é muito elevado.

A introdução da instrução AUX é adequada para cálculos que usam floating-point, ou seja para aplicações multimedia (jogos, vídeos), financeiros ou científicos. Aumenta também o paralelismo existindo instruções de vírgula flutuante e diminui o uso de registos devido às operações "não destrutivas" instruções.