

Y86:
Encadeamento de Instruções (PIPE)

Arquitetura de Computadores
Lic. em Engenharia Informática

Y86: Encadeamento de instruções (*pipeline*)

3 – Organização do Processador	
Conteúdos	3.2 – <i>Datapath</i> encadeado (<i>pipeline</i>)
	3.3 – Dependências de Dados e Controlo
Resultados de Aprendizagem	R3.2 – Analisar e descrever organizações encadeadas de processadores elementares
	R3.3 – Caracterizar limitações inerentes a organizações encadeadas (dependências) e conceber potenciais soluções

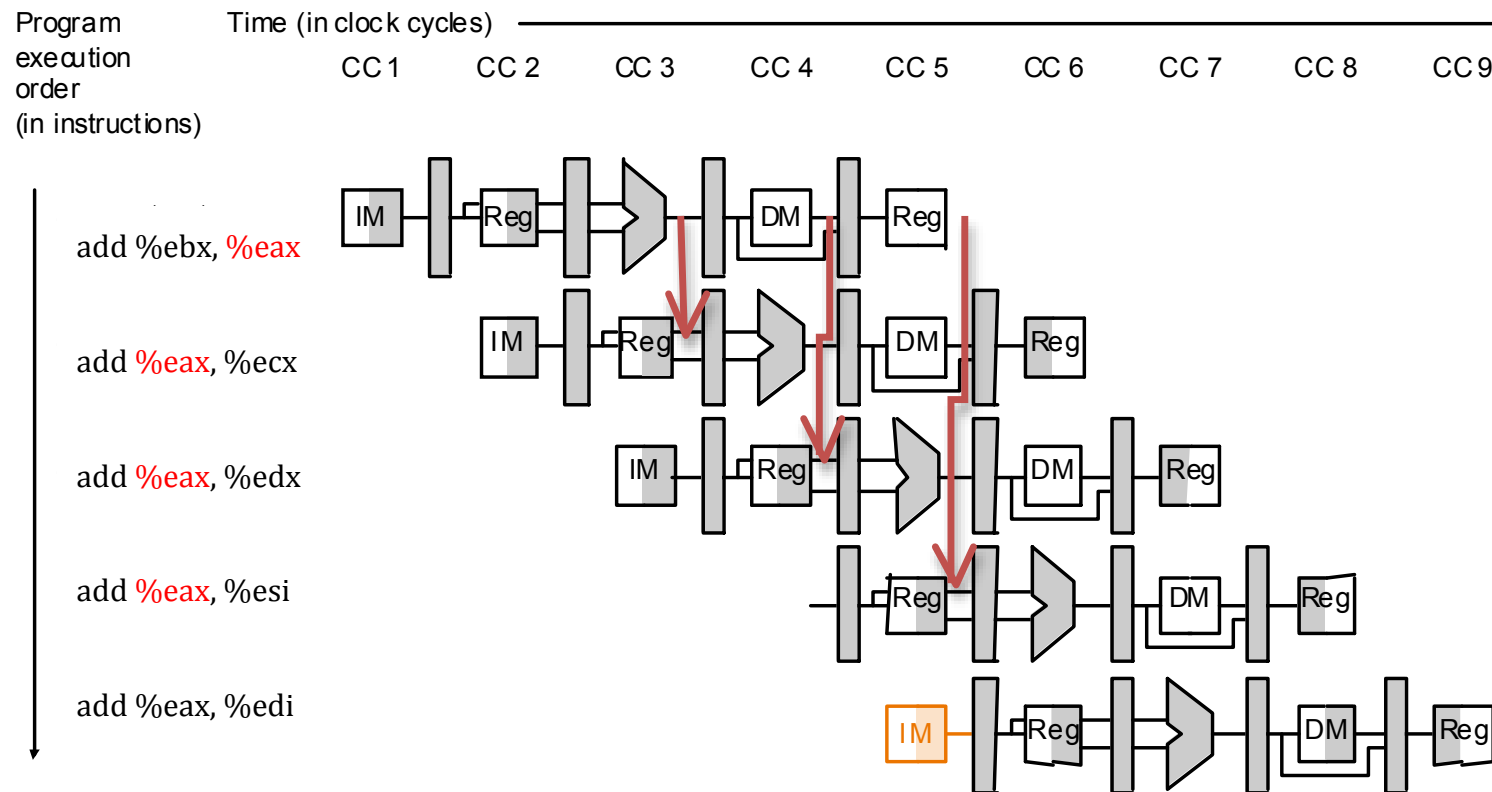
Y86 PIPE-: Limitações

- Dependências de Dados
 - Uma leitura de um registo precedida de uma escrita no mesmo registo constitui uma dependência de dados
 - Se a leitura ocorre antes da conclusão da escrita ocorre uma anomalia
 - Na versão PIPE- estas anomalias são corrigidas empatando (*stall*) o *pipeline* através da injeção de “bolhas” (`nops`)
- Dependências de controlo
 - O desfecho dos saltos condicionais só é conhecido depois da fase de execução. O Y86 prevê que o salto é tomado, executando as instruções no alvo de forma especulativa. Previsões erradas são corrigidas inserindo “bolhas”.
 - O destino de um `ret` só é conhecido depois da fase de leitura de memória. O Y86 resolve esta anomalia inserindo “bolhas” até que o endereço da próxima instrução seja conhecido.

Y86 PIPE: Motivação

- As dependências de dados são demasiado comuns
- Resolvê-las recorrendo à injeção de “bolhas” resulta no desperdício de um elevado número de ciclos, comprometendo o desempenho do *pipeline*
- A versão PIPE do Y86 propõe-se resolver estas dependências de dados, diminuindo o número de bolhas injectadas (logo o número de ciclos desperdiçados)
- As dependências de controlo não sofrem qualquer alteração relativamente a PIPE-

Y86-PIPE- Condições para *stall*



Data Forwarding

- Problema
 - Um registo é lido na fase de DECODE
 - A escrita só ocorre na fase de WRITEBACK
- Observação
 - O valor a escrever no registo é gerado na fase de execução ou memória
- Resolução do problema
 - Passar o valor necessário directamente do estágio onde está disponível (E, M ou W) para o estágio de DECODE

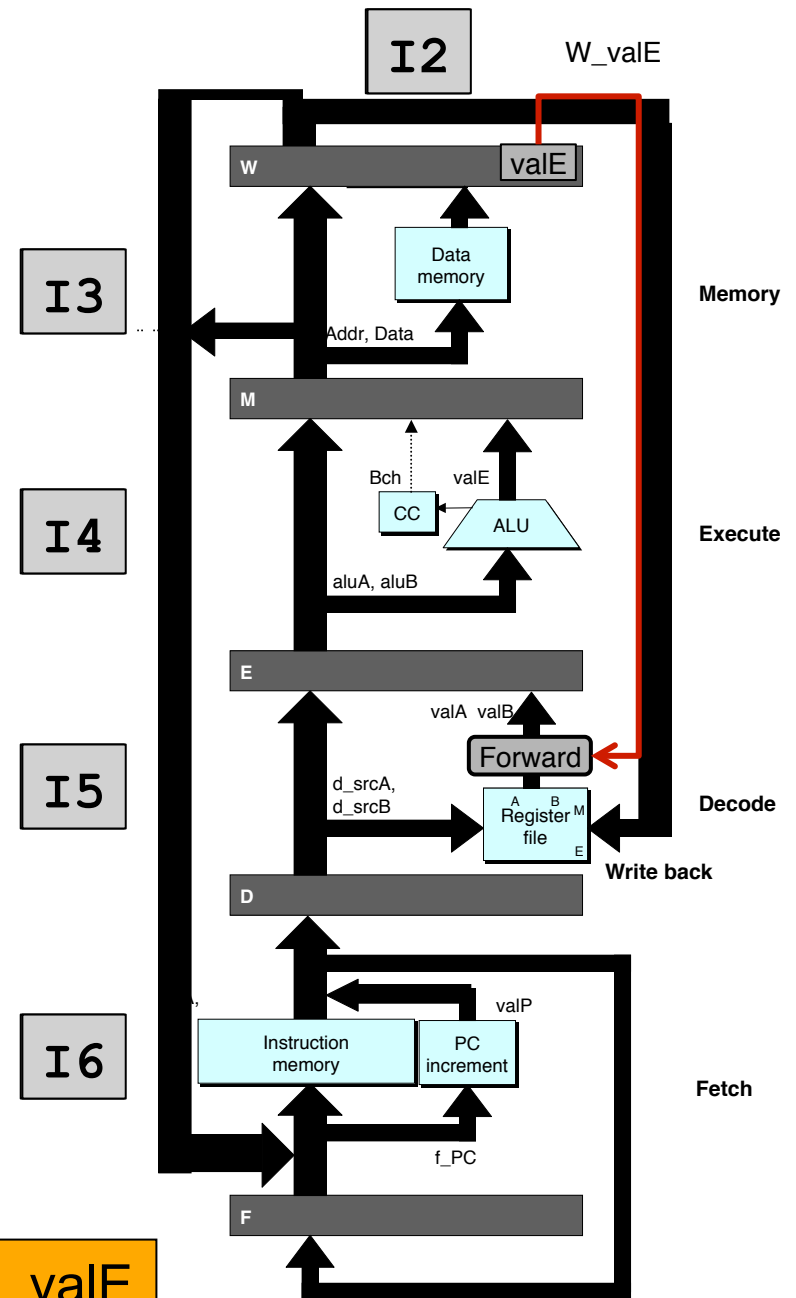
Y86 PIPE: Exemplo de Forwarding (1)

```

I1: irmovl $10, %eax
I2: mrmovl 30(%ebx), %ecx
I3: addl %esi, %edi
I4: addl %esi, %eax
I5: jmp MAIN
    
```

	1	2	3	4	5	6
I1	F	D	E	M	W	
I2		F	D	E	M	W
I3			F	D	E	M
I4				F	D	E
I5					F	D
I6						F

valB = W_valE



Y86 PIPE: Exemplo de Forwarding (2)

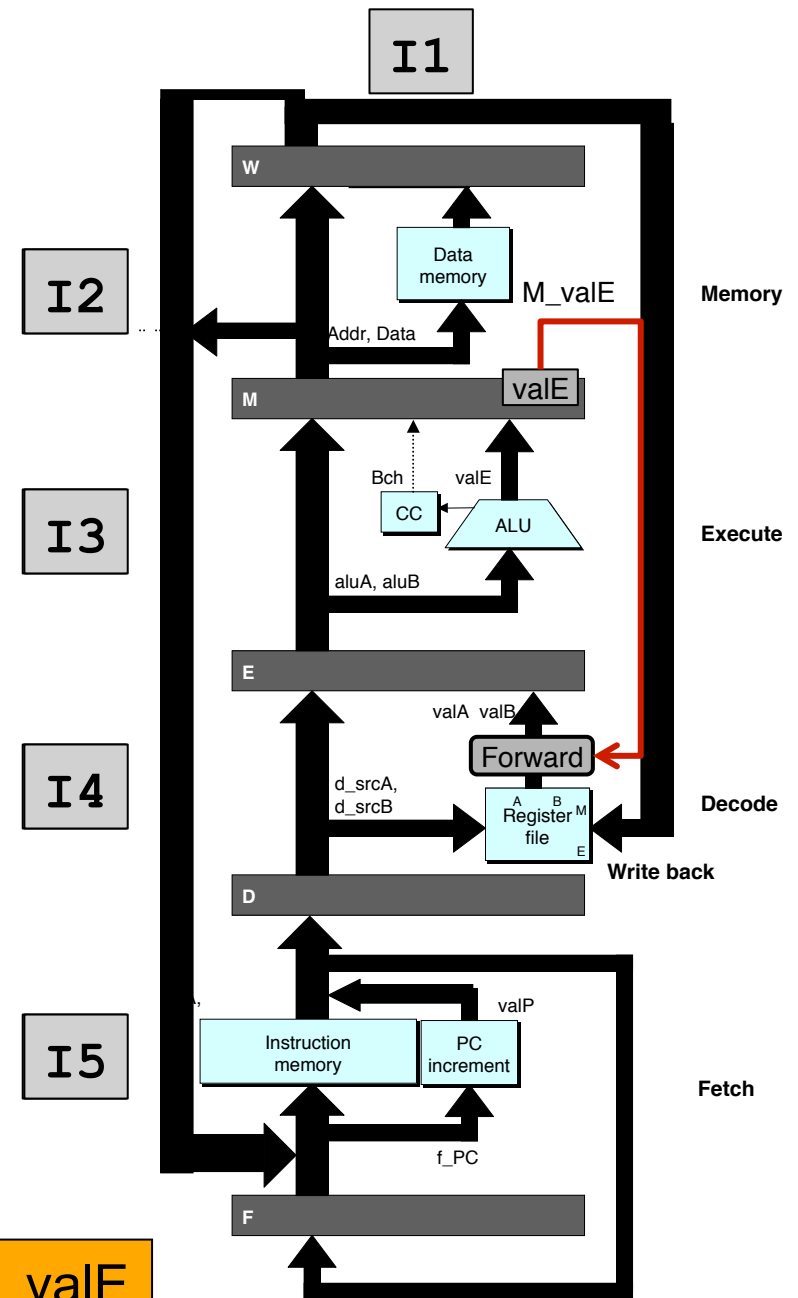
```

I1: irmovl $10, %eax
I2: mrmovl 30(%ebx), %ecx
I3: addl %esi, %eax
I4: ...
    
```

	1	2	3	4	5	6
I1	F	D	E	M	W	
I2		F	D	E	M	W
I3			F	D	E	M
I4				F	D	E
I5					F	D
I6						F

AC – Y86: PIPE

valB = M_valE



Y86 PIPE: Exemplo de Forwarding (3)

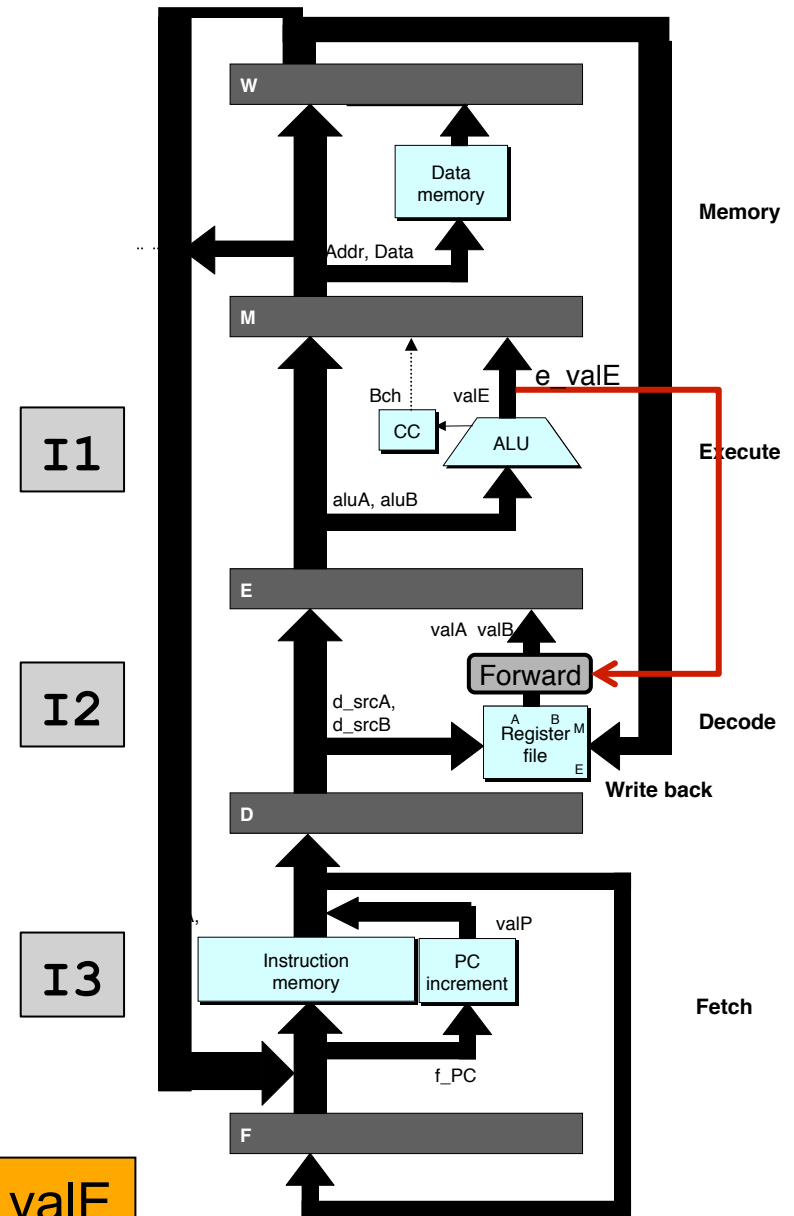
```

I1: irmovl $10, %eax
I2: addl %esi, %eax
I3: ...
    
```

	1	2	3	4	5	6
I1	F	D	E	M	W	
I2		F	D	E	M	W
I3			F	D	E	M
I4				F	D	E
I5					F	D
I6						F

AC – Y86: PIPE

valB = e_valE



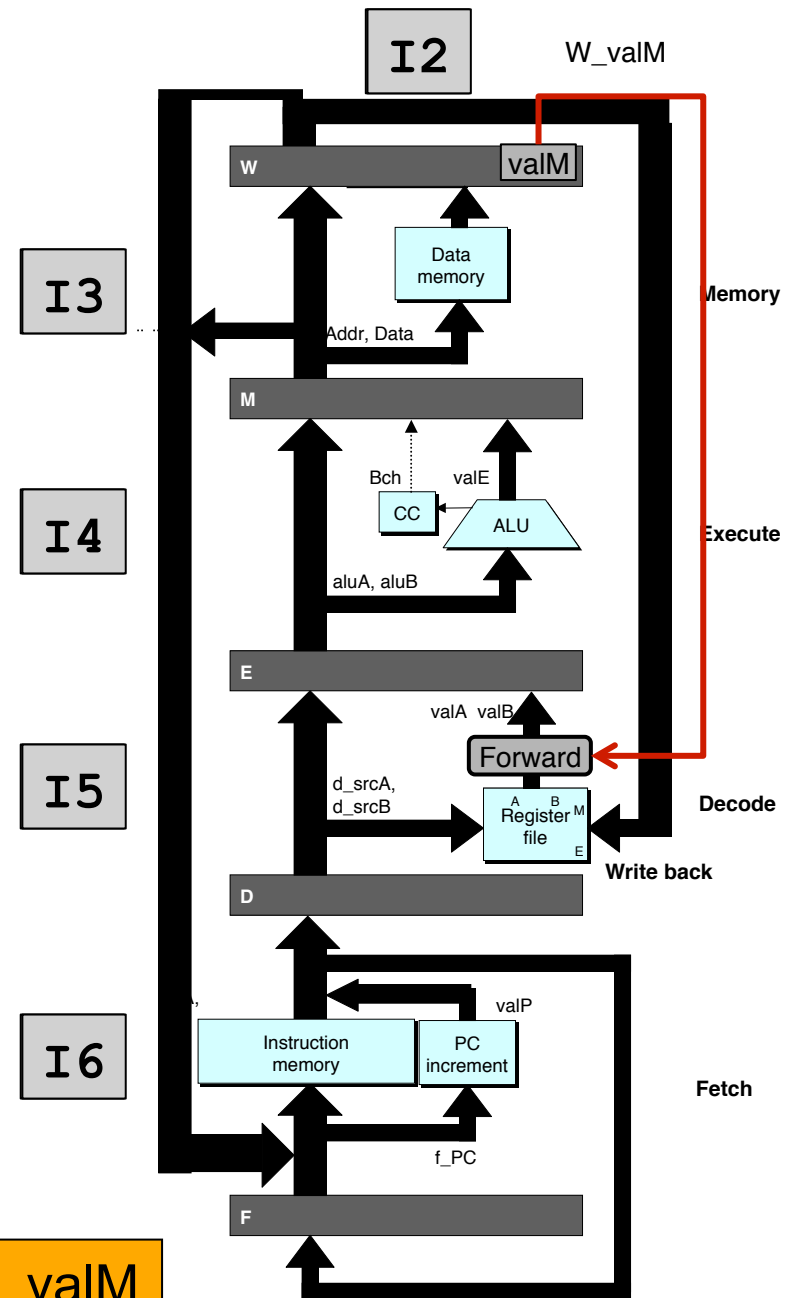
Y86 PIPE: Exemplo de Forwarding (4)

```

I1: mrmovl 30(%ebx), %ecx
I3: addl %esi, %ebx
I3: addl %esi, %edi
I4: addl %ecx, %eax
I5: jmp MAIN
    
```

	1	2	3	4	5	6
I1	F	D	E	M	W	
I2		F	D	E	M	W
I3			F	D	E	M
I4				F	D	E
I5					F	D
I6						F

valA = W_valM



Y86 PIPE: Exemplo de Forwarding (5)

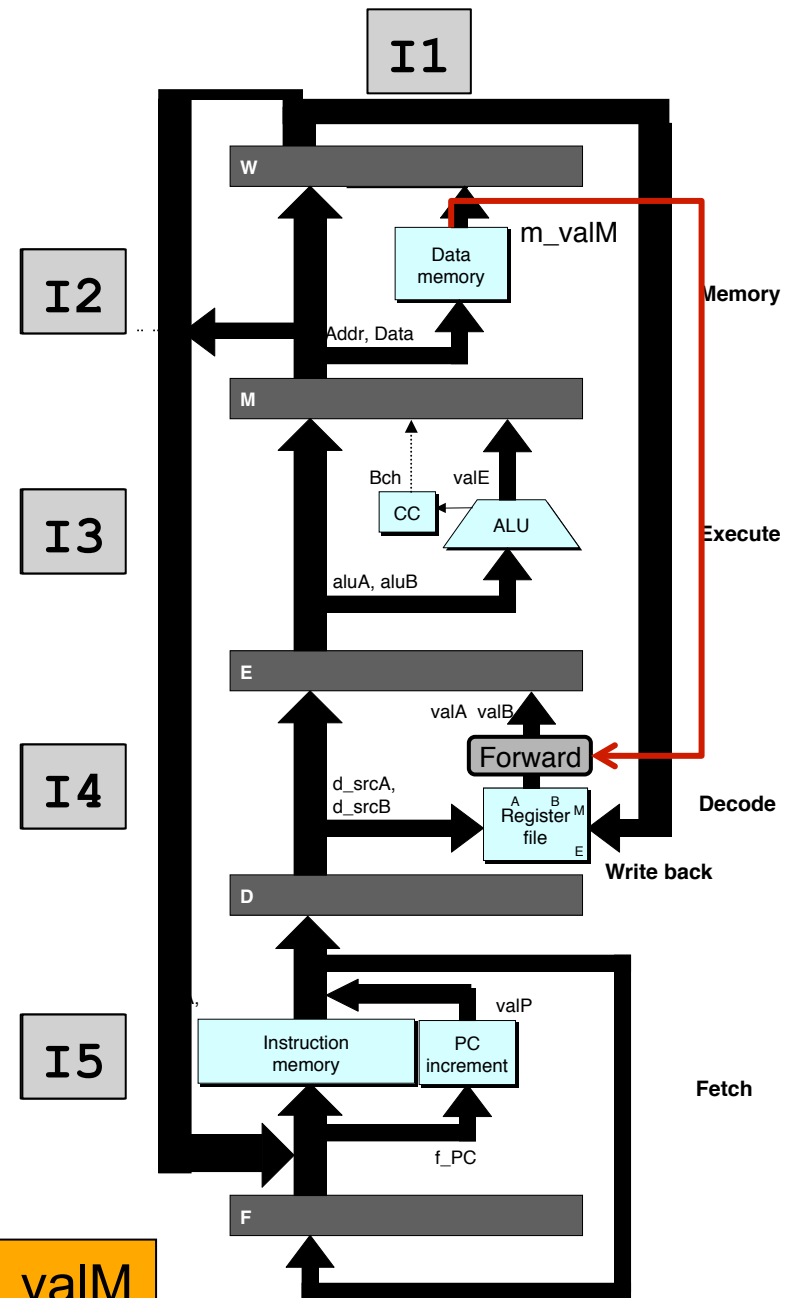
```

I1: mrmovl 30(%ebx), %ecx
I2: addl %esi, %edi
I3: addl %ecx, %eax
I4: ...
    
```

	1	2	3	4	5	6
I1	F	D	E	M	W	
I2		F	D	E	M	W
I3			F	D	E	M
I4				F	D	E
I5					F	D
I6						F

AC – Y86: PIPE

valA = m_valM



Y86 PIPE: Exemplo de Forwarding (6)

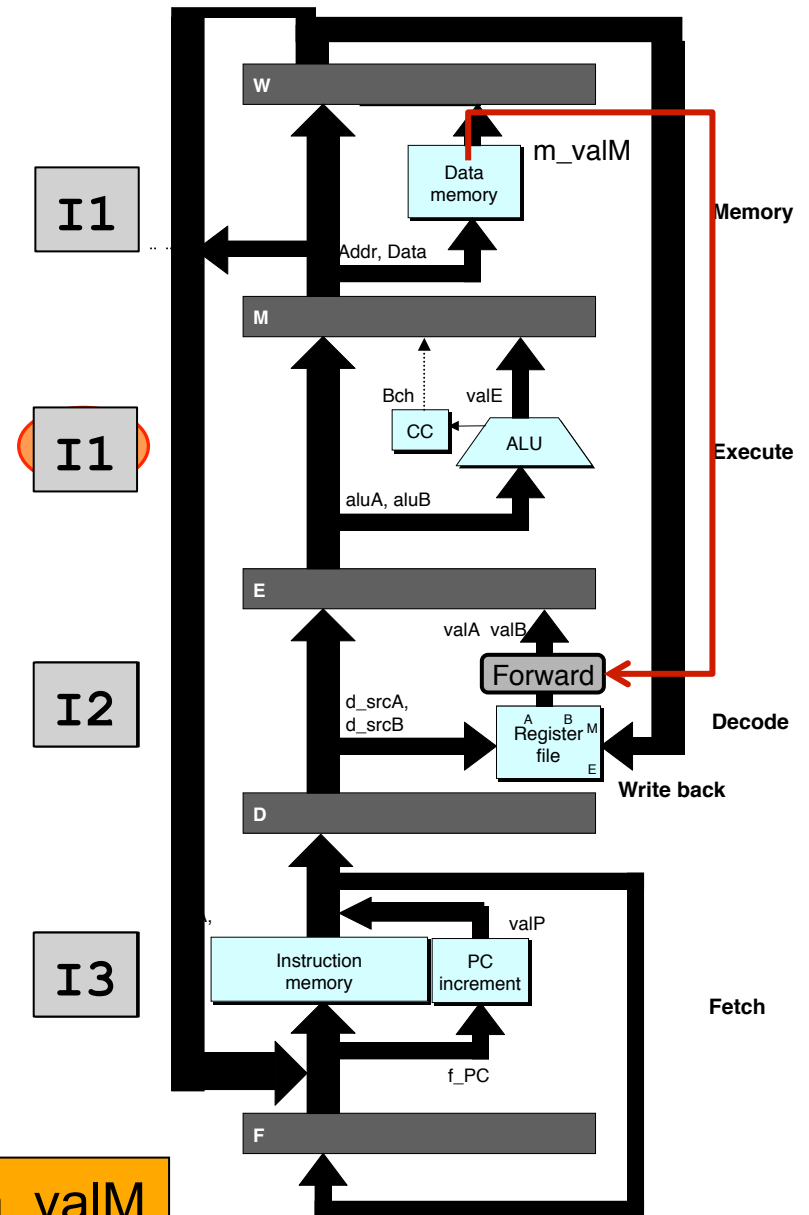
```

I1: mrmovl 30(%ebx), %ecx
I2: addl %ecx, %eax
I3: ...
    
```

	1	2	3	4	5	6
I1	F	D	E	M	W	
				E	M	W
I2		F	D	D	E	M
I3			F	F	D	E
I4					F	D
I5						F

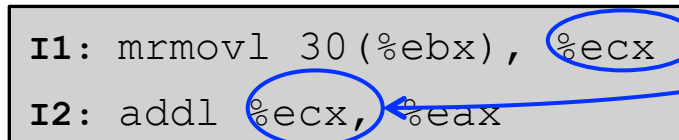
AC – Y86: PIPE

valA = m_valM



Load /Use

- Uma situação de *load/use* ocorre quando uma leitura de memória para registo é seguida de uma leitura do mesmo registo



```
I1: mrmovl 30(%ebx), %ecx
I2: addl %ecx, %eax
```

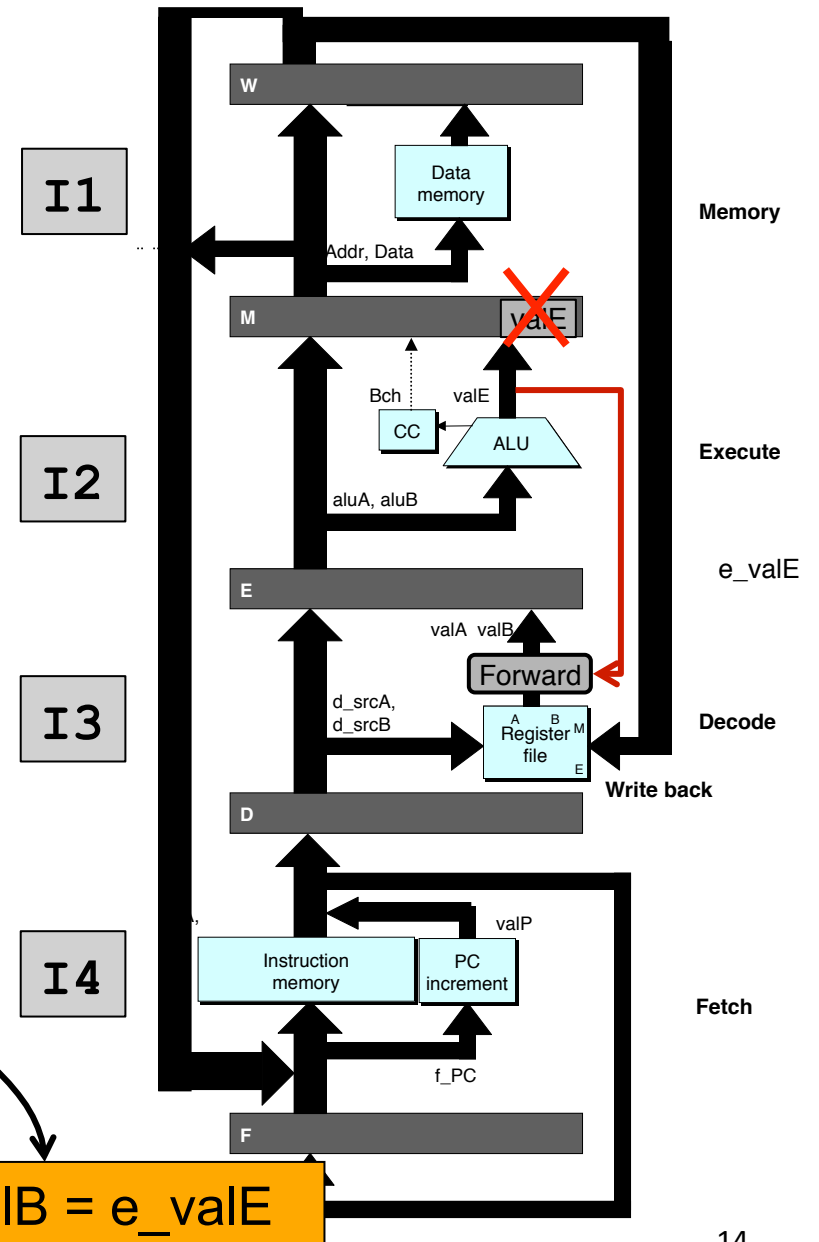
- Como I1 ainda está no estágio de Execute quando I2 pede o valor do registo, este valor ainda não foi lido e não pode ser encaminhado (*forwarded*) para o Decode
- A resolução da anomalia passa por injectar uma “bolha”, dando assim tempo para que a memória seja lida

Y86 PIPE: Atalho a usar

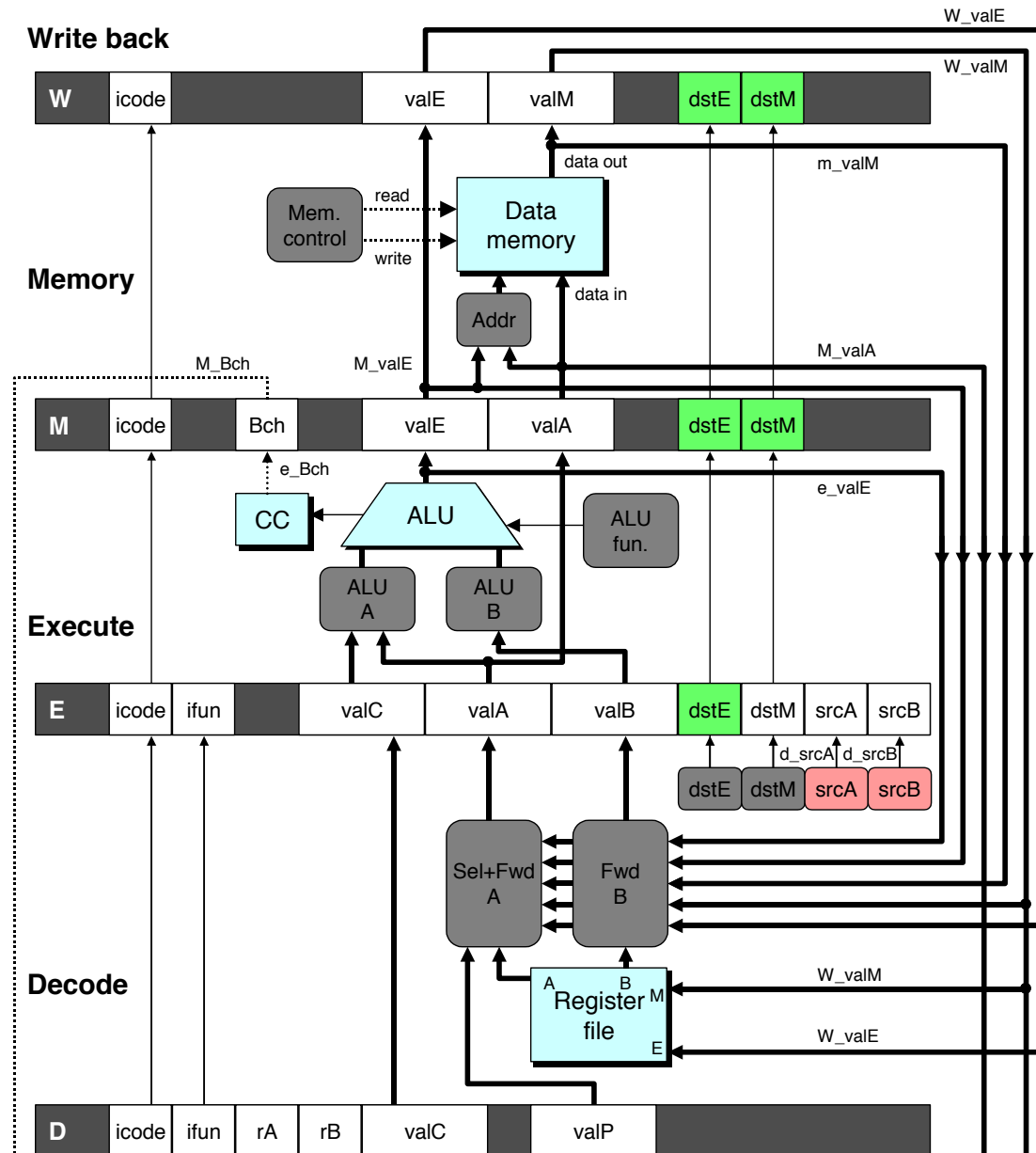
I1: irmovl \$10, %eax
I2: irmovl \$30, %eax
I3: addl %eax, %eax

	1	2	3	4	5	6
I1	F	D	E	M		
I2		F	D	E		
I3			F	D		
I4				F		
I5						
I6						

AC – Y86: PIPE



valA = valB = e_valE



AC – Y86: PIPE

Y86 PIPE: implementação de *forwarding*

- Adicionar 5 atalhos dos registos de *pipeline* E, M, e W para o estágio DECODE
- Adicionar 2 *multiplexers* para seleccionar `valA` e `valB` no DECODE

Y86 PIPE: Resumo

- Dependências de Dados
 - Tratadas maioritariamente com *forwarding*
 - Não há penalização no desempenho
 - Load/use exige que se empate o *pipeline* durante 1 ciclo
- Dependências de Controlo (não há alterações relativamente a PIPE-)
 - Salto condicional mal previsto: cancelar instruções em F e D
 - 2 ciclos do relógio desperdiçados
 - `ret`: Empatar o estágio F (injectando bolhas em E) até o endereço de retorno ser lido
 - 3 ciclos do relógio desperdiçados

Y86 PIPE: Desempenho

- Tempo de execução em PIPE (não considerando *stall*)
 - $\#CC \approx \#I$, logo $CPI \approx 1.0$
- Considerando *stalls* temos:
 - 1 ciclo adicional por cada Load/use (*LP – load penalty*)
 - 2 ciclos por cada salto mal previsto (*MP – mispredict penalty*)
 - 3 ciclos por cada instrução RET (*RP – return penalty*)
 - $\#CC_{pipe} = \#I + \#I_{LP} + 2 * \#I_{MP} + 3 * \#I_{RET}$, ou
 - $CPI_{pipe} = 1 + \%I_{LP} + 2 * \%I_{MP} + 3 * \%I_{RET}$
 - $\#I_{LP}$ = instruções que implicam load penalty, etc
 - $\%I_{LP}$ = %instruções que implicam load penalty, etc

Y86 PIPE: Desempenho

- Exemplo
 - Qual o CPI de um programa na arquitetura PIPE em que:
 - 25% das instruções são Load e em 20% dos casos o valor é utilizado pela instrução seguinte
 - 20% das instruções são saltos condicionais e 40% não são tomados
 - 2% das instruções são RET
 - $LP = 0,25 \times 0,2 = 0,05$ (acréscimo de 0,05 ciclos por instrução)
 - $MP = 2 \times 0,2 \times 0,4 = 0,16$
 - $RP = 3 \times 0,02 = 0,06$
 - $CPI = 1 + 0,05 + 0,16 + 0,06 = 1,27$