

Y86:
Encadeamento de Instruções (PIPE)

Arquitectura de Computadores
Lic. em Engenharia Informática
Luís Paulo Santos

Created with

 **nitro**^{PDF} professional

download the free trial online at nitropdf.com/professional

Y86: Encadeamento de instruções (*pipeline*)

9 – Organização do Processador	
Conteúdos	9.2 – <i>Datapath</i> encadeado (<i>pipeline</i>)
	9.3 – Dependências de Dados e Controlo
Resultados de Aprendizagem	R9.2 – Analisar e descrever organizações encadeadas de processadores elementares
	R9.3 – Caracterizar limitações inerentes a organizações encadeadas (dependências) e conceber potenciais soluções

Y86 PIPE-: Limitações

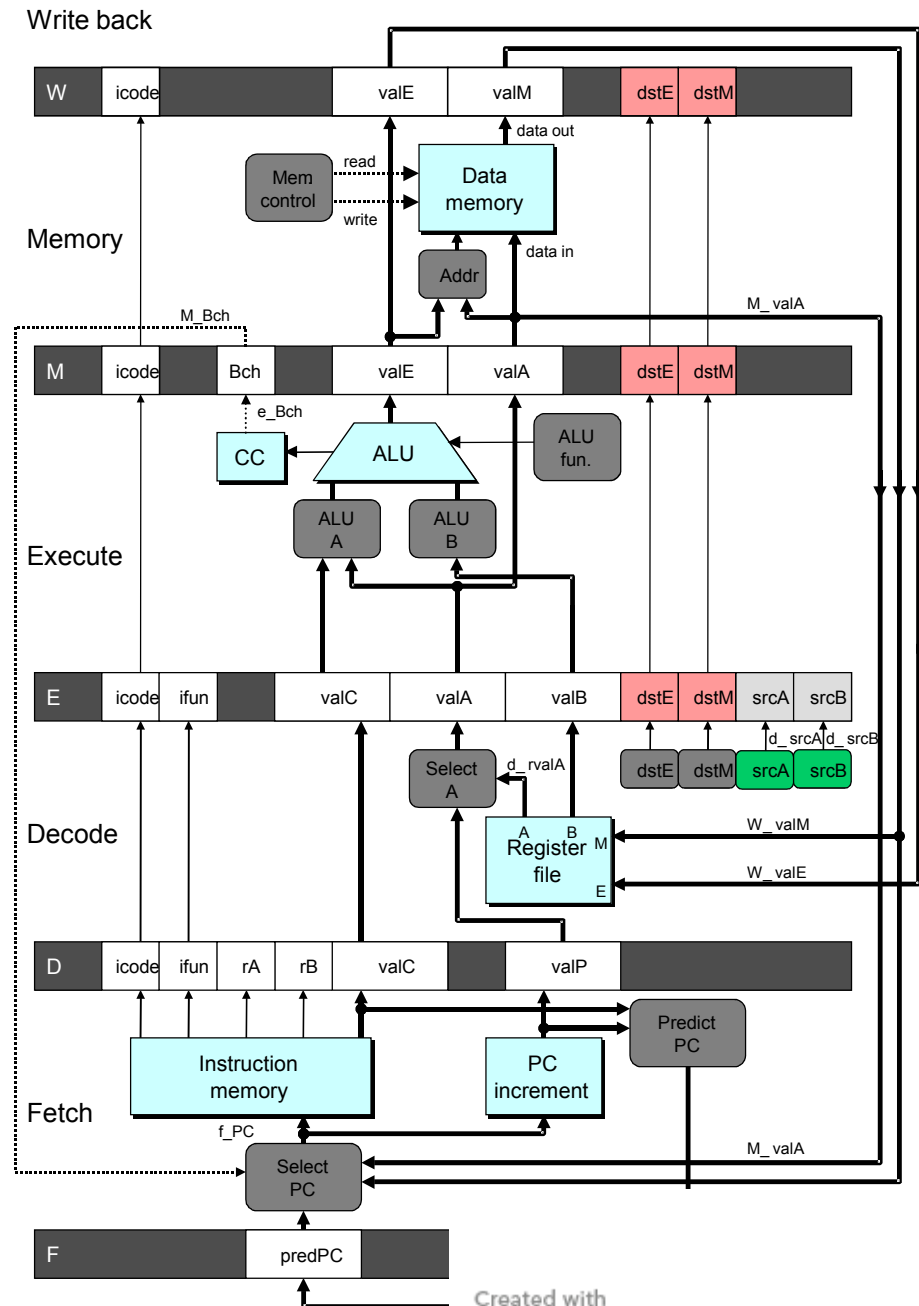
- Dependências de Dados
 - Uma leitura de um registo precedida de uma escrita no mesmo registo constitui uma dependência de dados
 - Se a leitura ocorre antes da conclusão da escrita ocorre uma anomalia
 - Na versão PIPE- estas anomalias são corrigidas empatando o *pipeline* através da injeção de “bolhas” (`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-

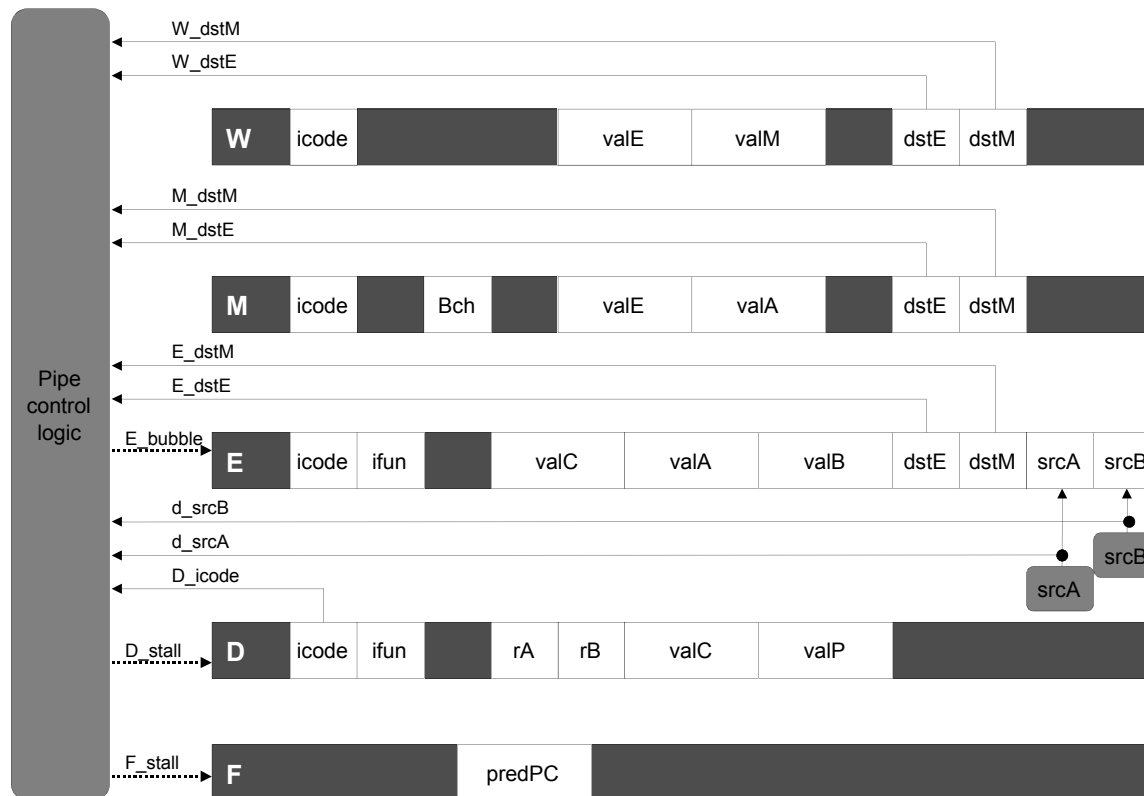
PIPE-: Condições para *stall*

- Registo a ler
 - `d_srcA` e `d_srcB`
(instrução no estágio de decode)
- Registos destino
 - `dstE` e `dstM` nos estágios E, M e W
- Dependência Dados
 - `d_srcA` ou `d_srcB` == `E_dst?` ou `M_dst?` Ou `W_dst?` (`? = E ou M`)
- Ignorar se `RegID==8`



PIPE- : Implementação do *stalling*

```
Se (d_srcA in {E_dstE, E_dstM, M_dstE, M_dstM, W_dstE, W_dstM} ||
    d_srcB in {E_dstE, E_dstM, M_dstE, M_dstM, W_dstE, W_dstM} )
Então
    E_bubble = D_stall = F_stall =1;
```



E_bubble

“Injecção” de um nop no estágio E (`icode=0`, `E_dstE=E_dstM=8`)

D_stall

Escrita no registo D inibida

F_stall

Escrita no registo F inibida

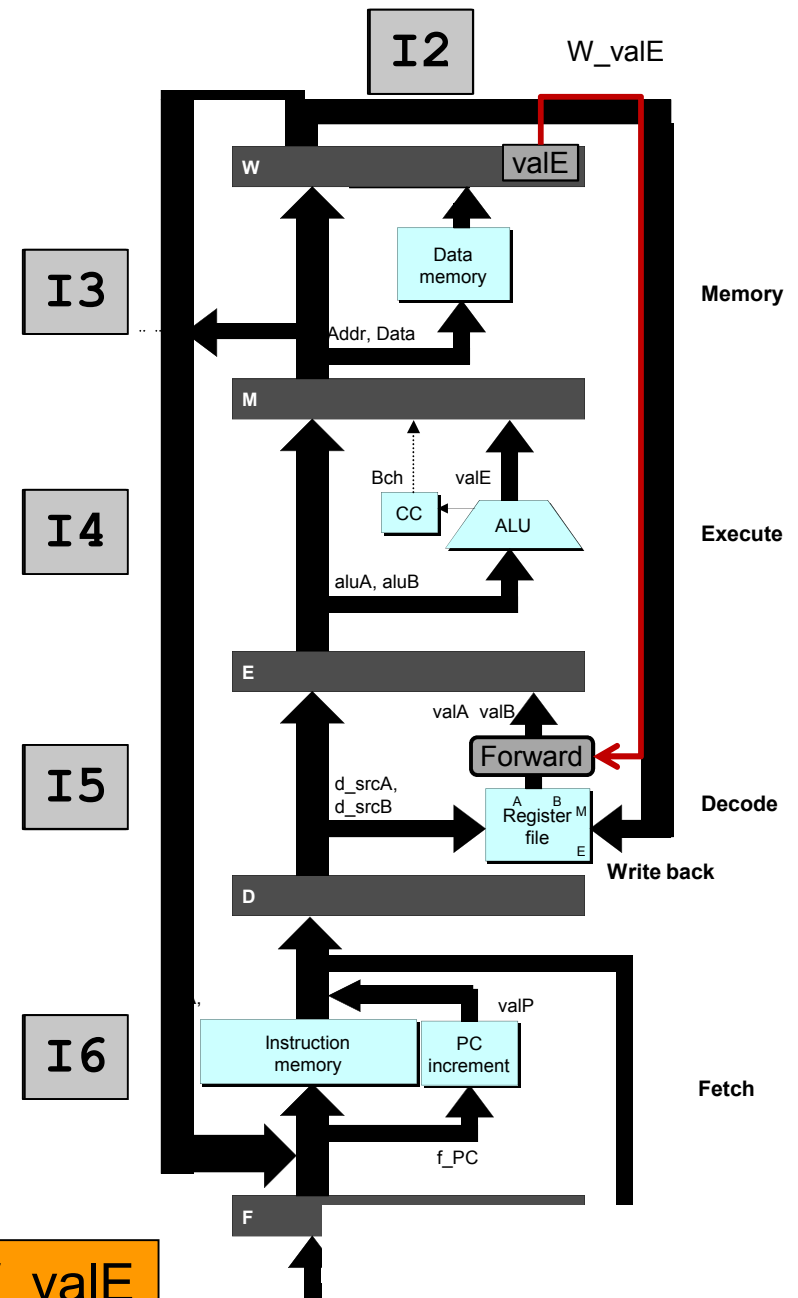
Data Forwarding

- 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

$$\text{valB} = W_ \text{valE}$$


Y86 PIPE: Exemplo de *Forwarding* (2)

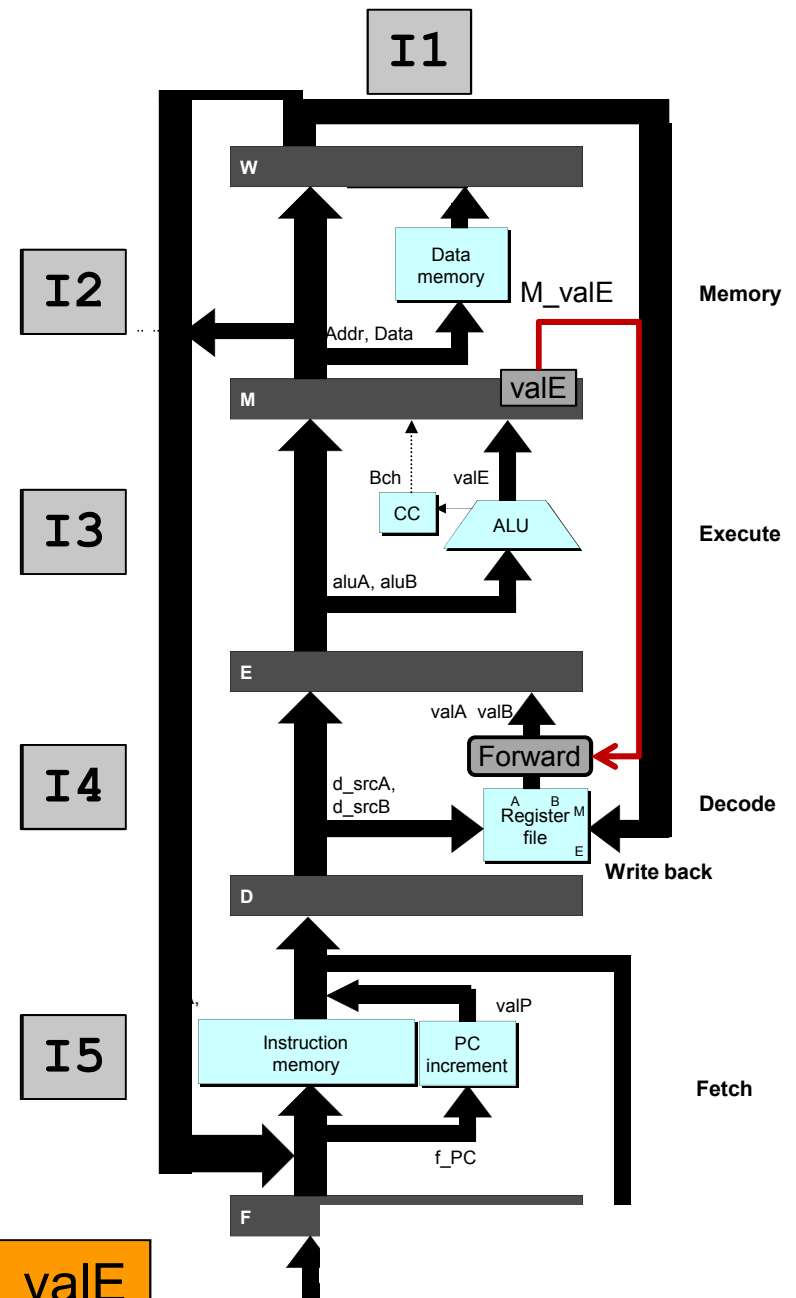
```

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)

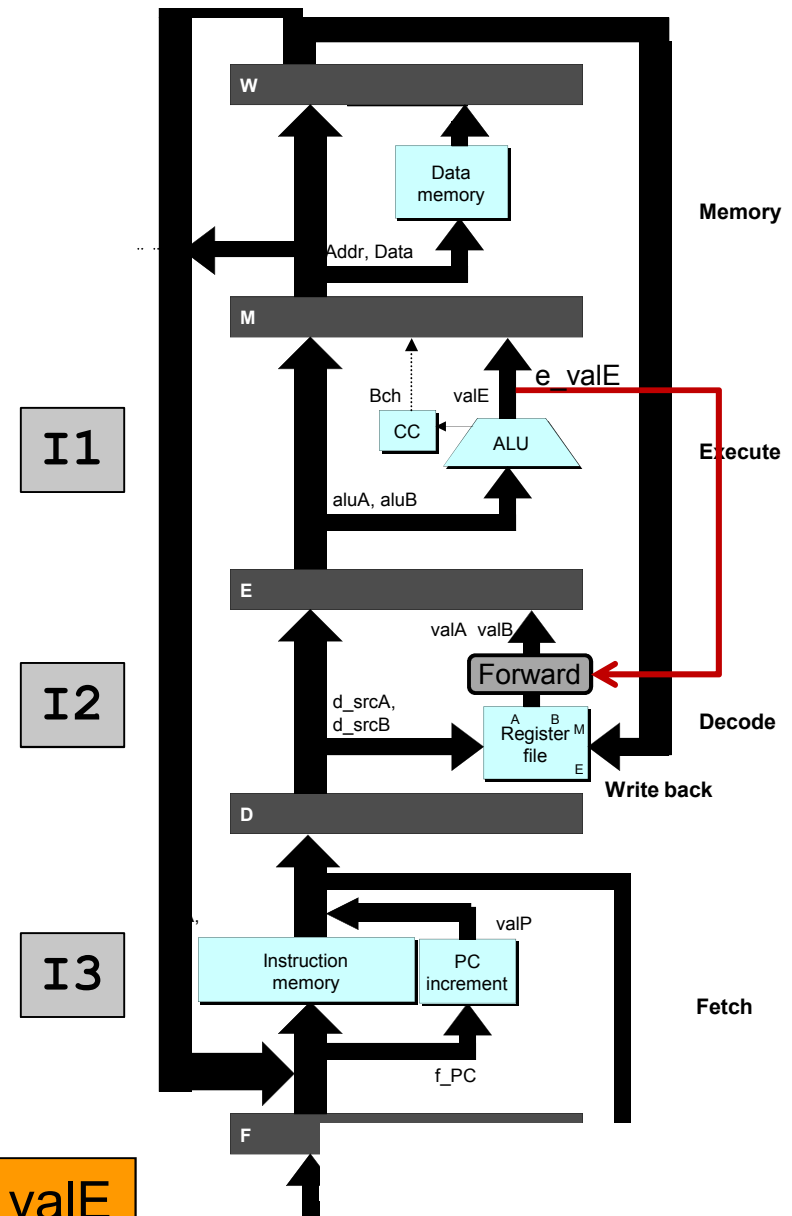
```

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



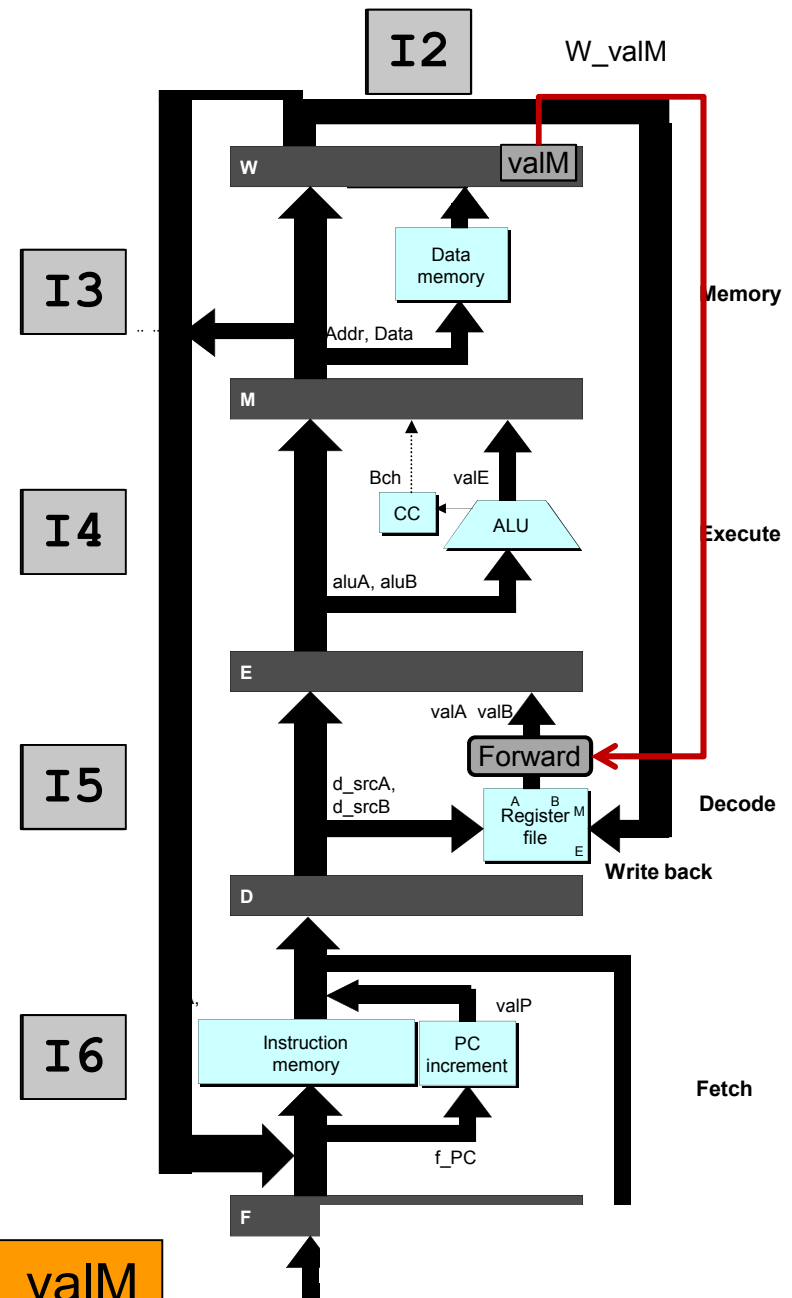
Y86 PIPE: Exemplo de Forwarding (4)

```

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)

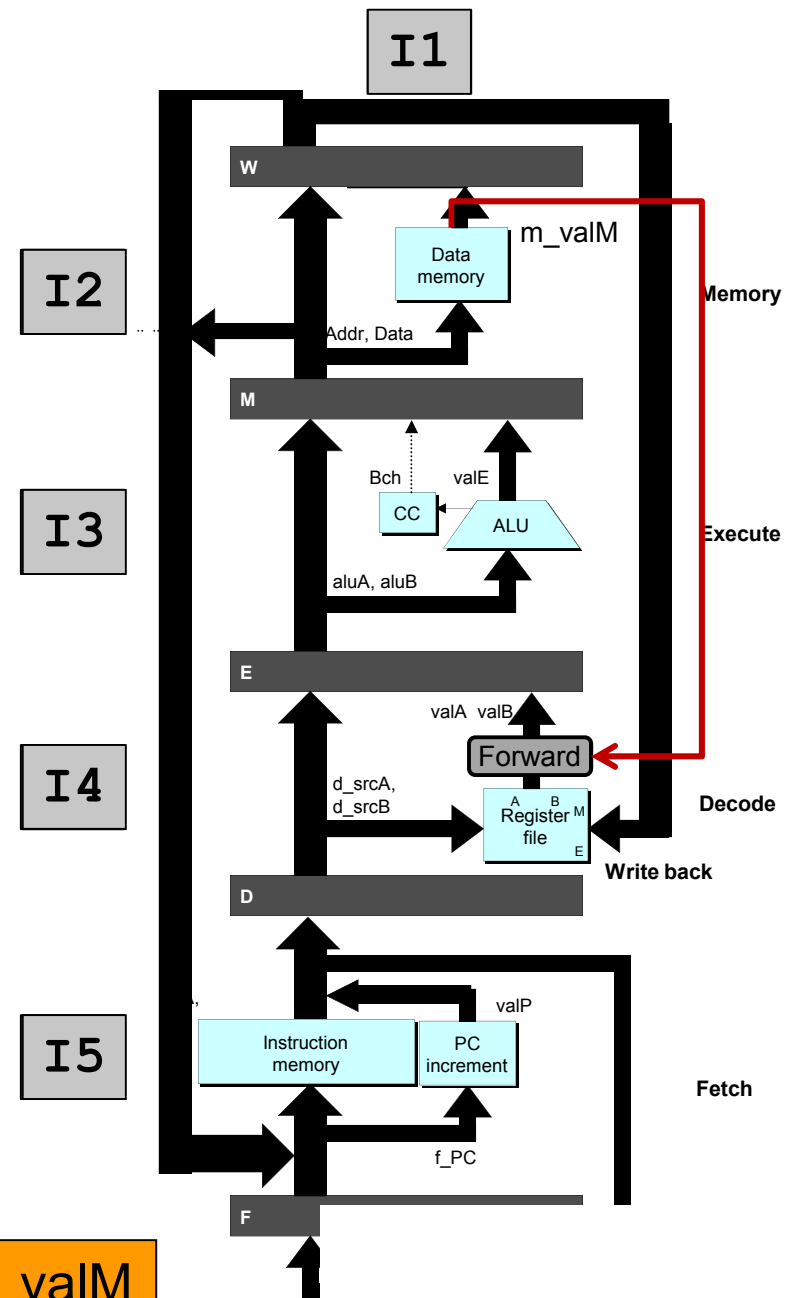
```

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



Created with

Y86 PIPE: Exemplo de Forwarding (6)

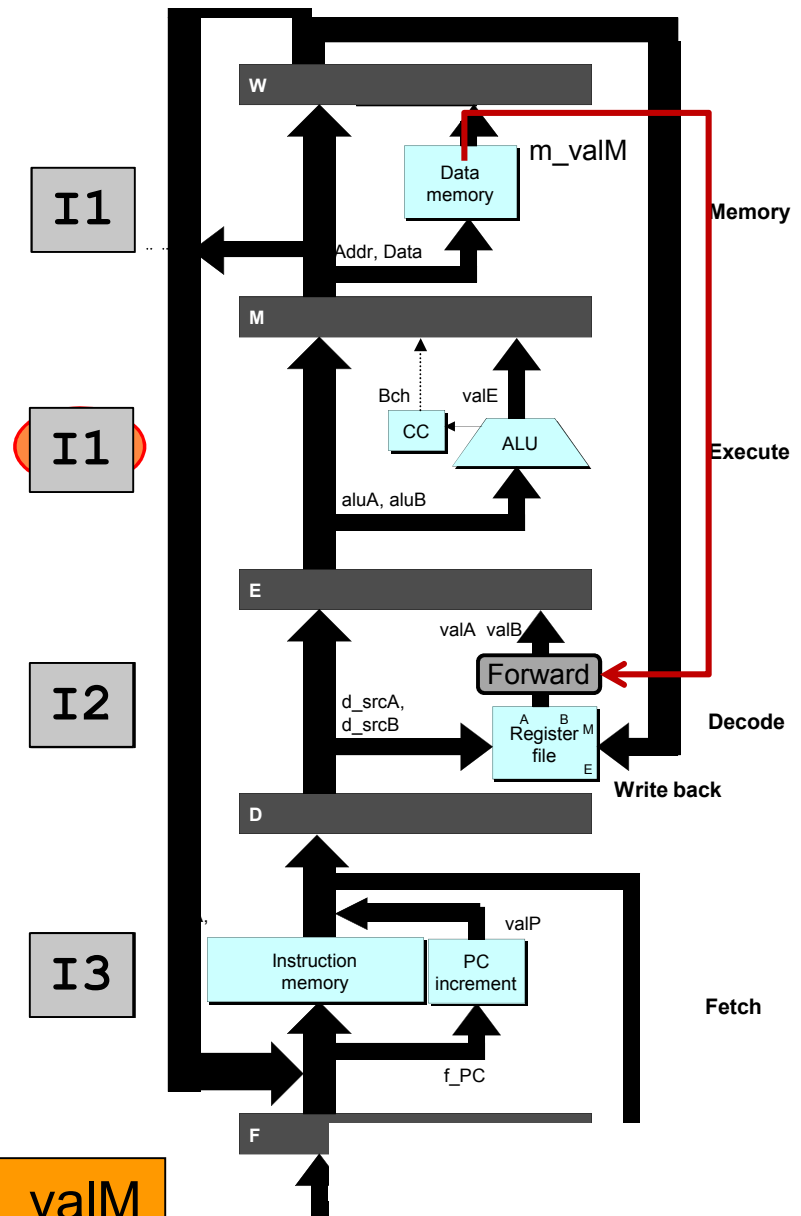
```

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

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

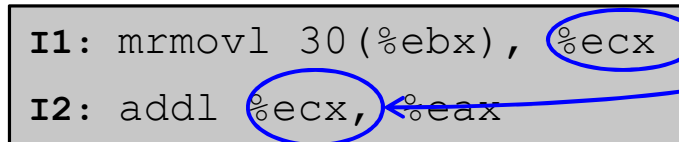
AC – Y86: PIPE

valA = m_valM



Load /Use

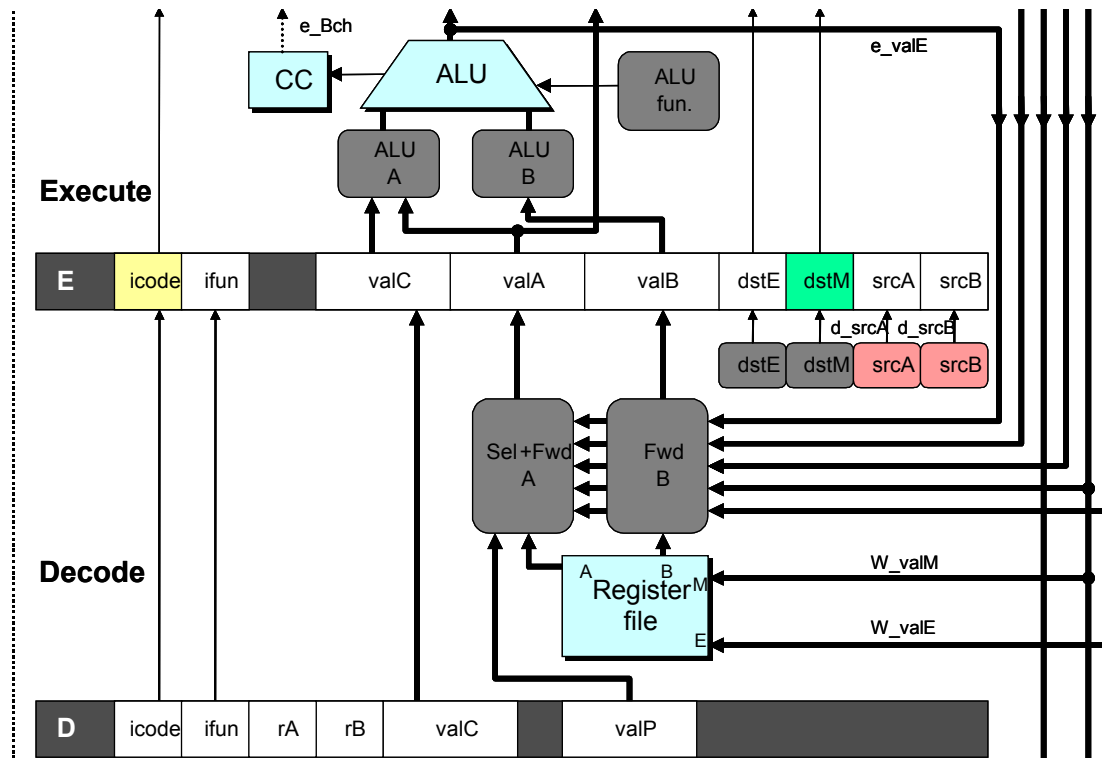
- Uma situação de *load/use* ocorre quando uma leitura de memória para registo é seguida de uma leitura do mesmo registo



```
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

Detecção de *load/use*



Anomalia	Condição
Load/Use	E_icode in { IMRMOVL, IPOPL } && E_dstM in { d_srcA, d_srcB }

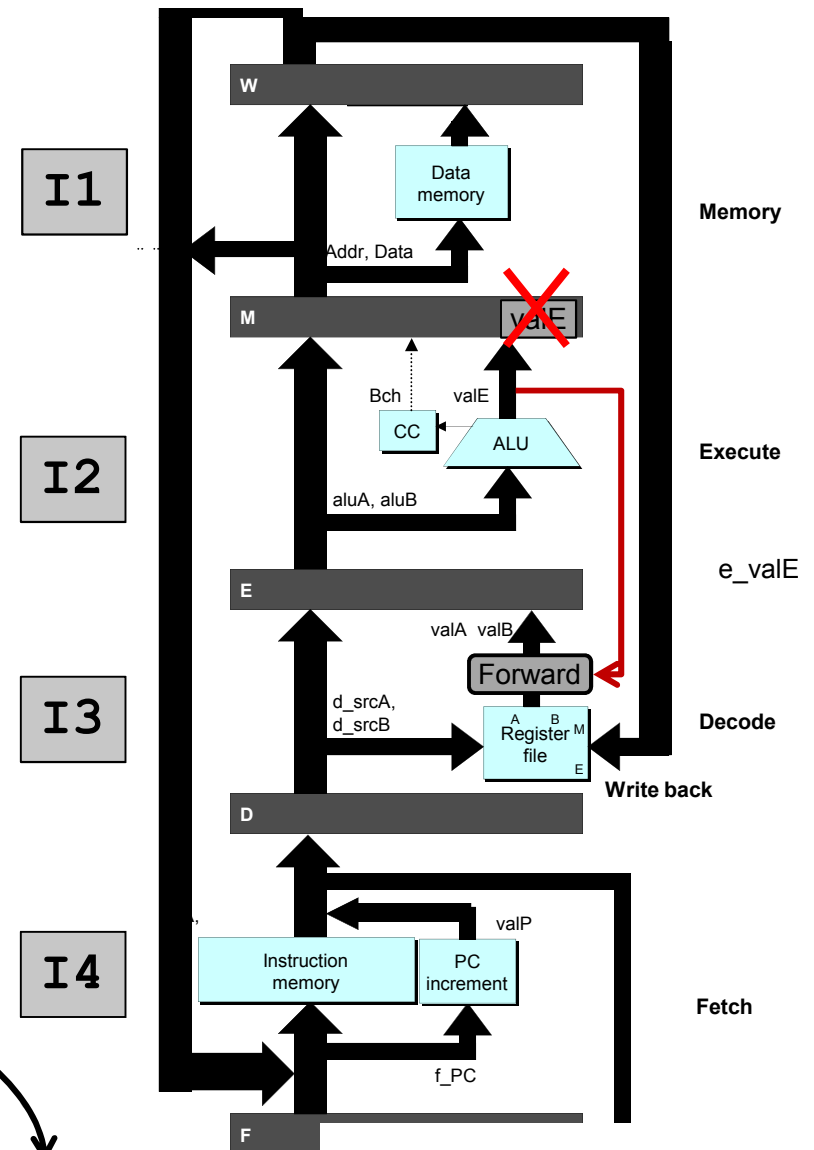
Y86 PIPE: Atalho a usar

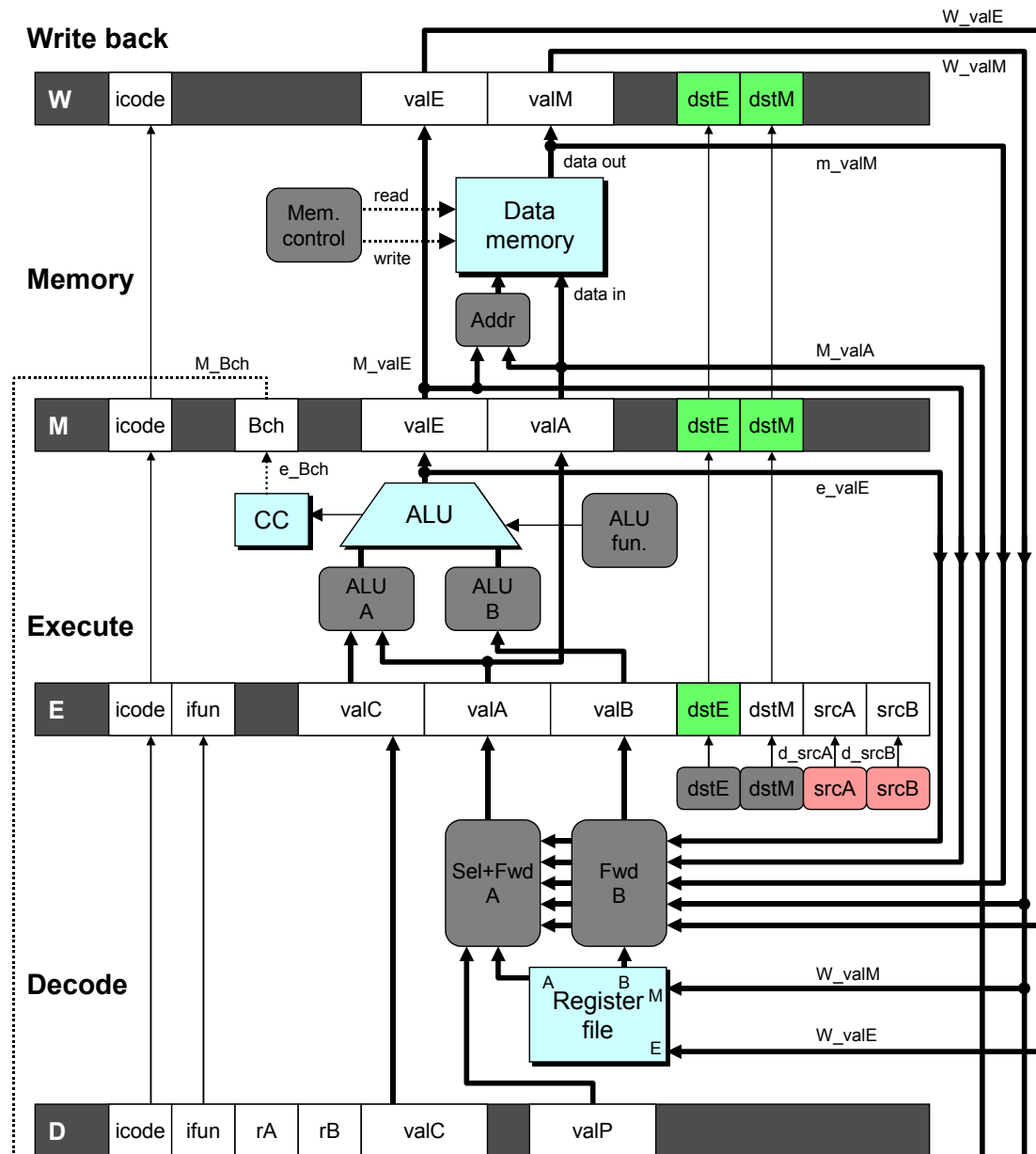
I1: irmovl \$10, %eax
I2: irmovl \$30, %eax
I3: addl %eax, %eax

	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





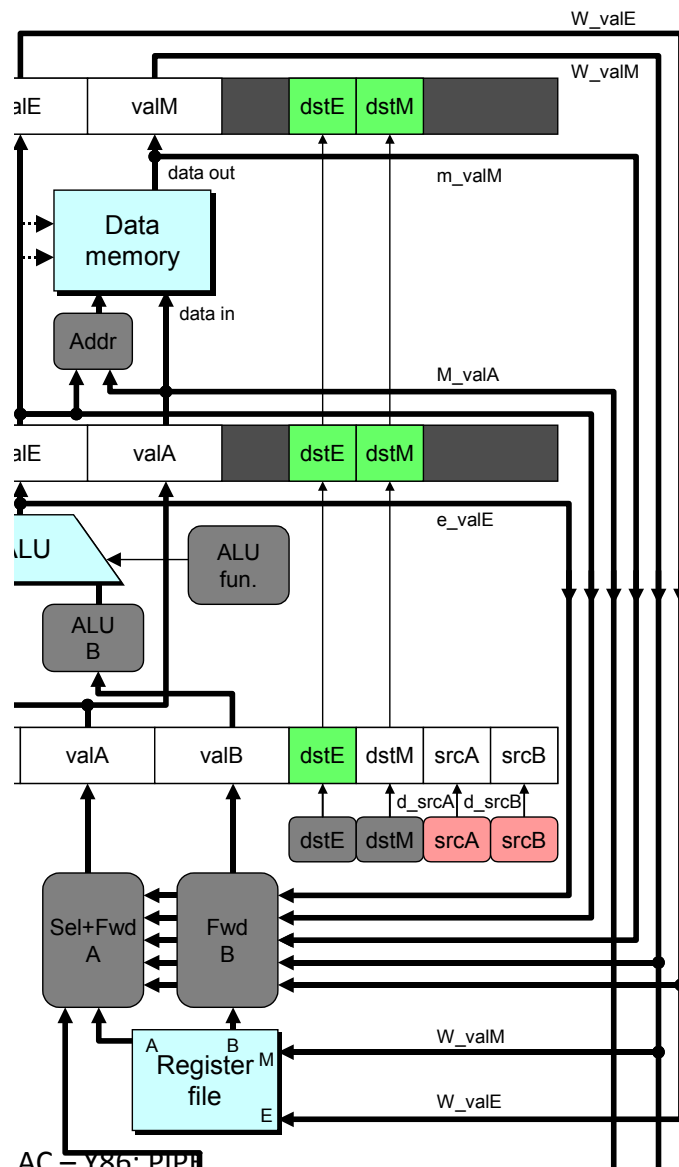
AC – Y86: PIPE

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

Created with

Y86 PIPE: implementação de *forwarding*



AC - Y86: PIPE

Qual o atalho a utilizar?
 ## NOTA: **X** representa A ou B conforme
 ## se trate de valA ou valB

```
int new_E_valX = [
    # Use incremented PC
    D_icode in { ICALL, IJXX } : D_valP;
    # Forward valE from execute
    d_srcX == E_dstE : e_valE;
    # Forward valM from memory
    d_srcX == M_dstM : m_valM;
    # Forward valE from memory
    d_srcX == M_dstE : M_valE;
    # Forward valM from write back
    d_srcX == W_dstM : W_valM;
    # Forward valE from write back
    d_srcX == W_dstE : W_valE;
    # Use value read from register file
    1 : d_rvalX;
];
```

Created with

nitroPDF[®] professional

download the free trial online at nitropdf.com/professional

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