O Processador - Parte 3 - Pipelining no MIPS

Arquitetura de Computadores

Emilio Francesquini
e.francesquini@ufabc.edu.br

2020.Q1

Centro de Matemática, Computação e Cognição
Universidade Federal do ABC

UFABC

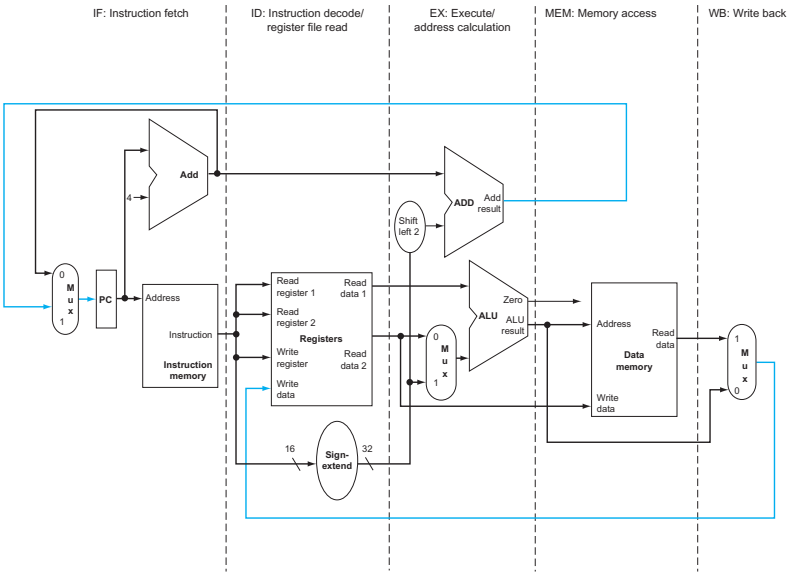
Disclaimer

UFABC

- Estes slides foram preparados para o curso de Arquitetura de Computadores na UFABC.
- Este material pode ser usado livremente desde que sejam mantidos, além deste aviso, os créditos aos autores e instituições.
- O conteúdo destes slides foi baseado no conteúdo do livro *Computer Organization And Design: The Hardware/Software Interface*, 5th Edition.

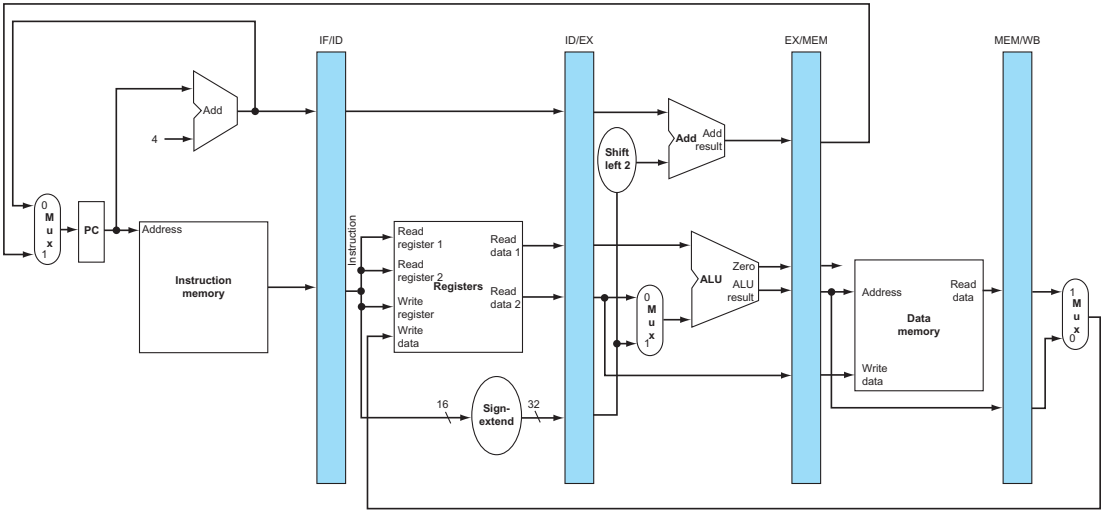
Pipelining no MIPS

MIPS Pipelined Datapath



- Fluxo direita → esquerda pode causar hazards

Registradores de pipeline

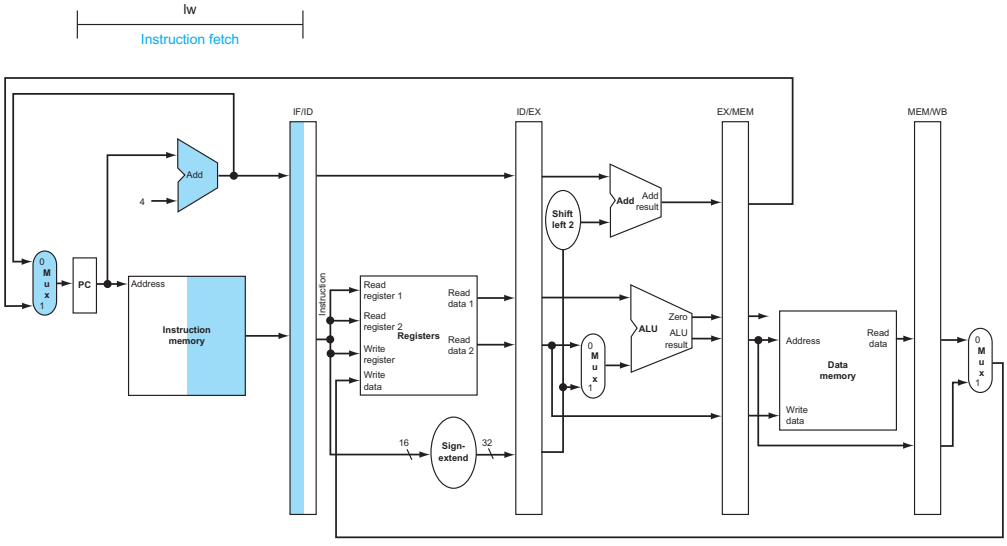


Operação do pipeline

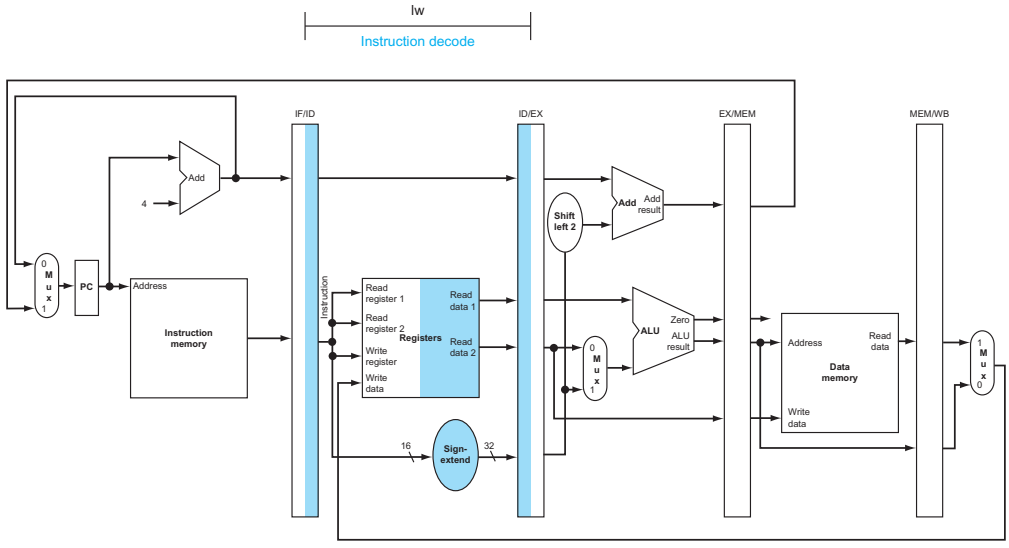
UFABC

- Ciclo a ciclo as instruções são passadas pelos estágios do datapath
- Há duas maneiras comuns de se mostrar o funcionamento do pipeline
  - *"Single-clock-cycle"pipeline diagram*
    - Mostra o uso do pipeline em um único ciclo
    - Destaca os recursos em uso
  - *"Multi-clock-cycle"pipeline diagram*
    - Mostra a operação durante o tempo
- Vamos começar dando uma olhada no *single-clock-cycle* para as operações de *load* e *store*

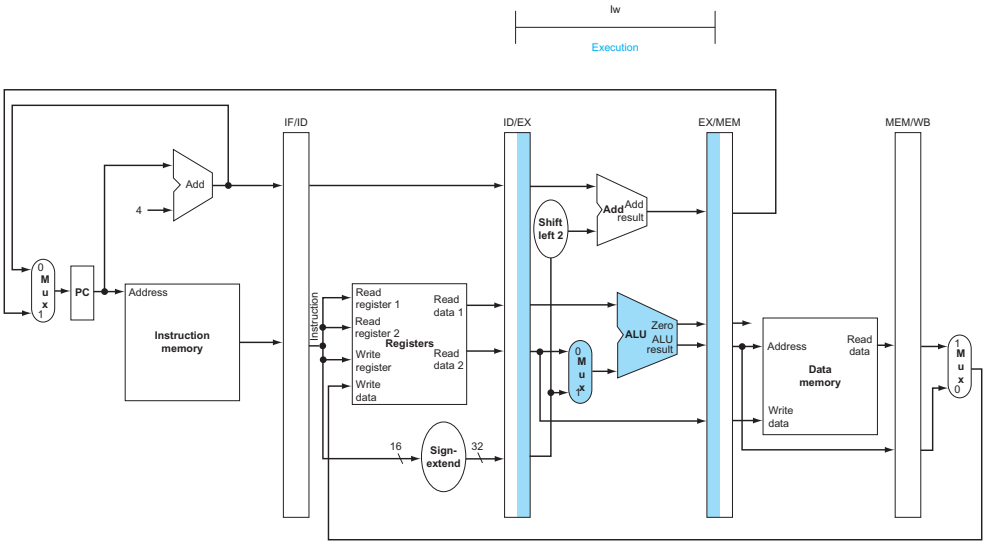
IF para *load* e *store*



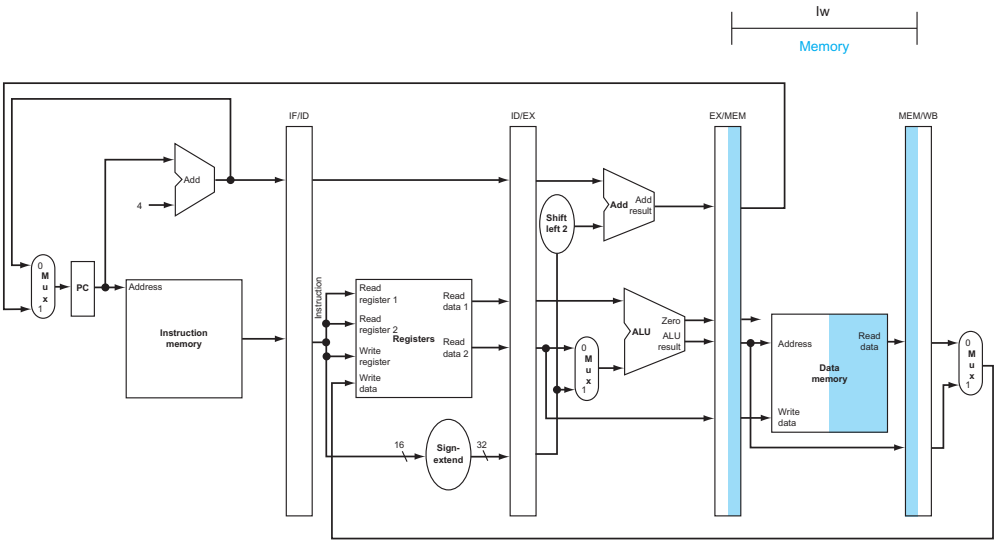
ID para *load* e *store*



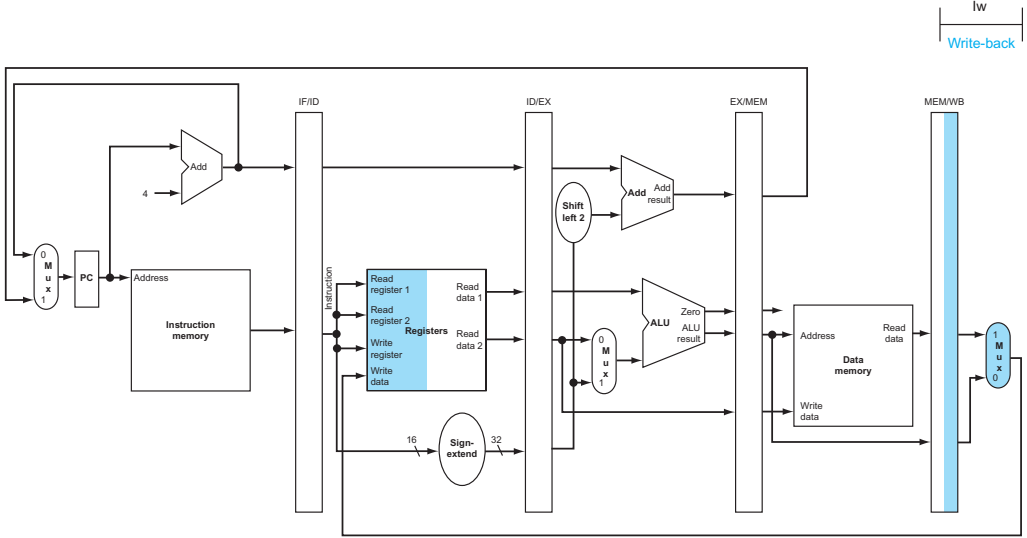
EX para *load*


lw
Execution
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
M
u
x
0
1
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
Shift left 2
Add
Add result
ALU
Zero
ALU result
Address
Data memory
Read data
Write data
16
Sign-extend
32

MEM para *load*

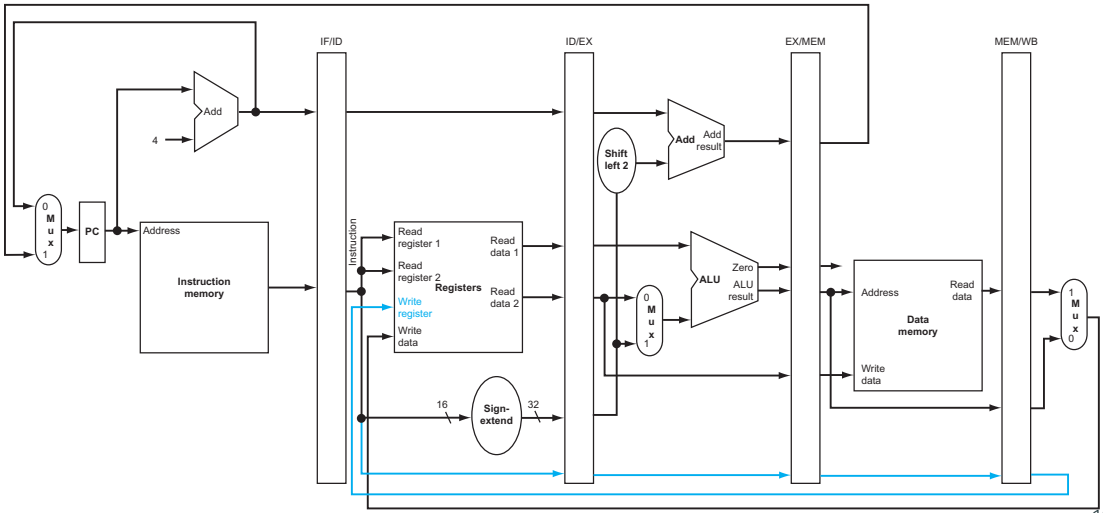


WB para *load*

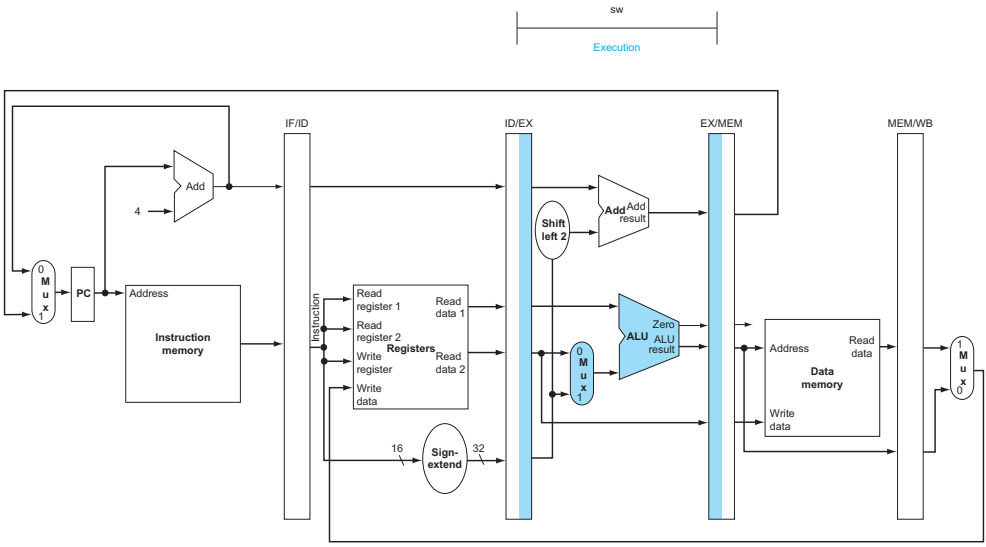


Possível número de registrador incorreto

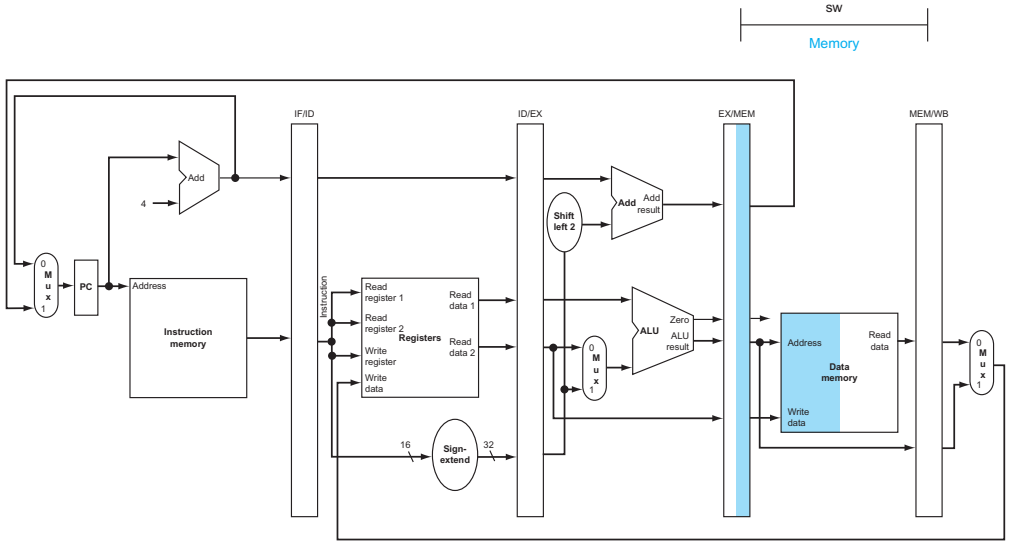
Datapath corrigido para *load*



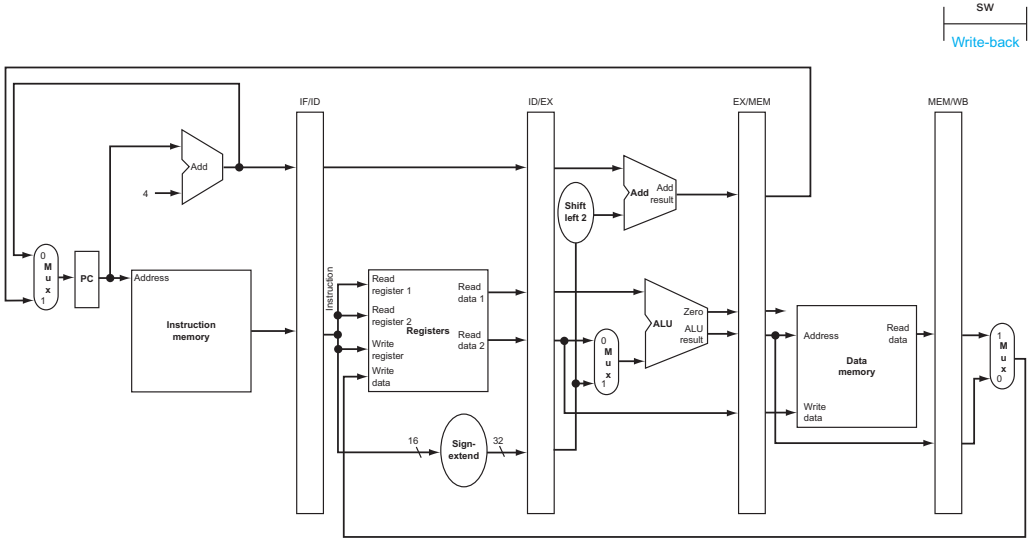
EX para *store*



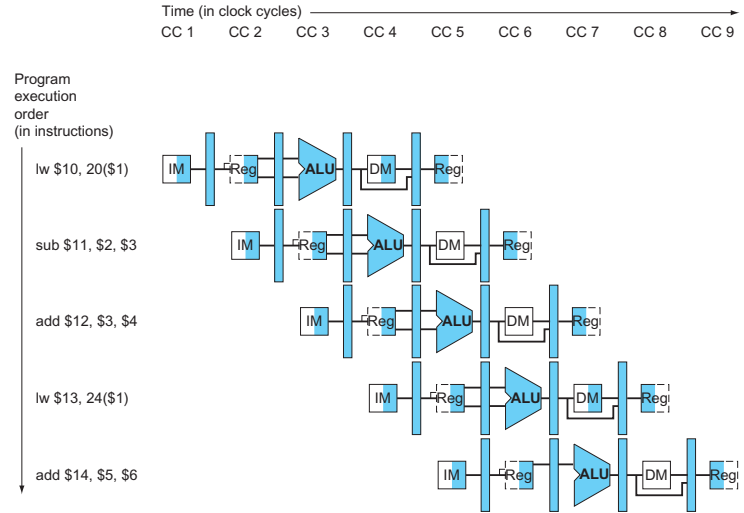
MEM para *store*



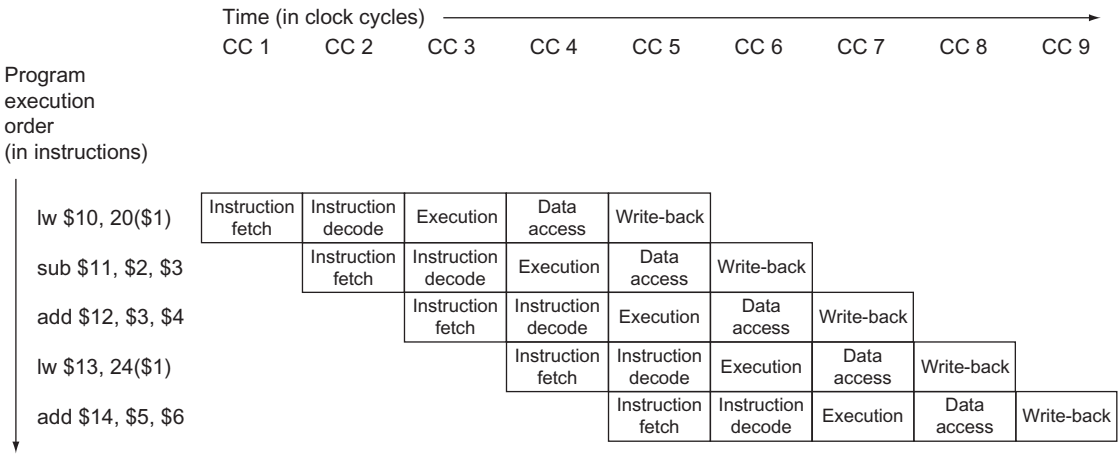
WB para *store*



Pode ser desenhado mostrando o uso dos recursos

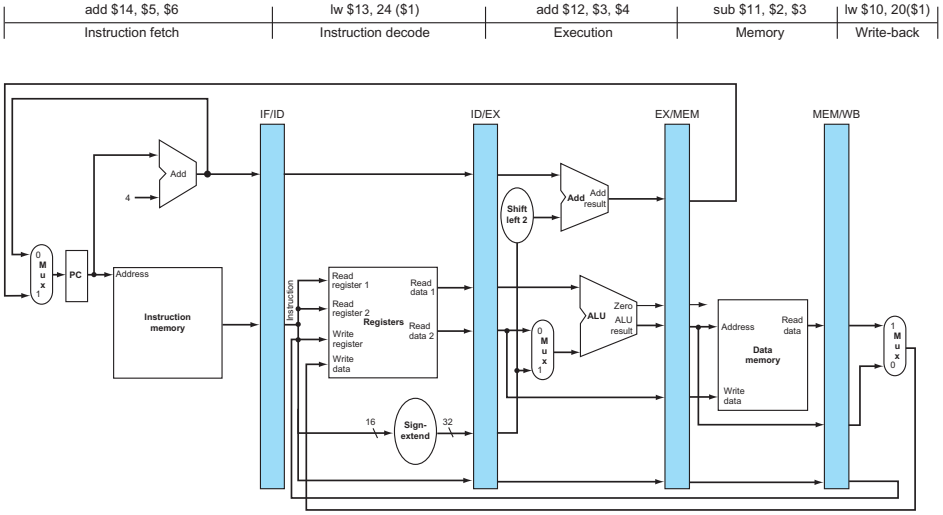


Ou mostrando o uso dos estágios

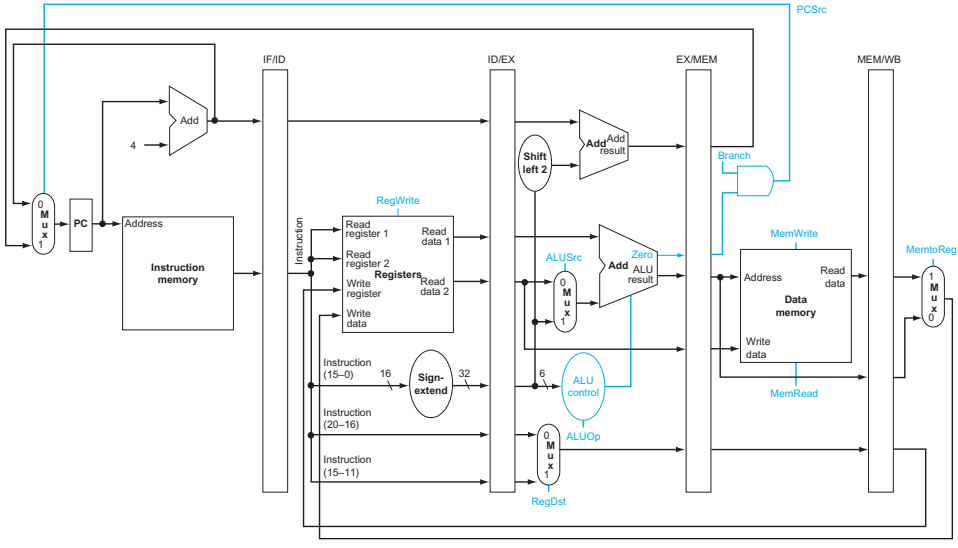


UFABC

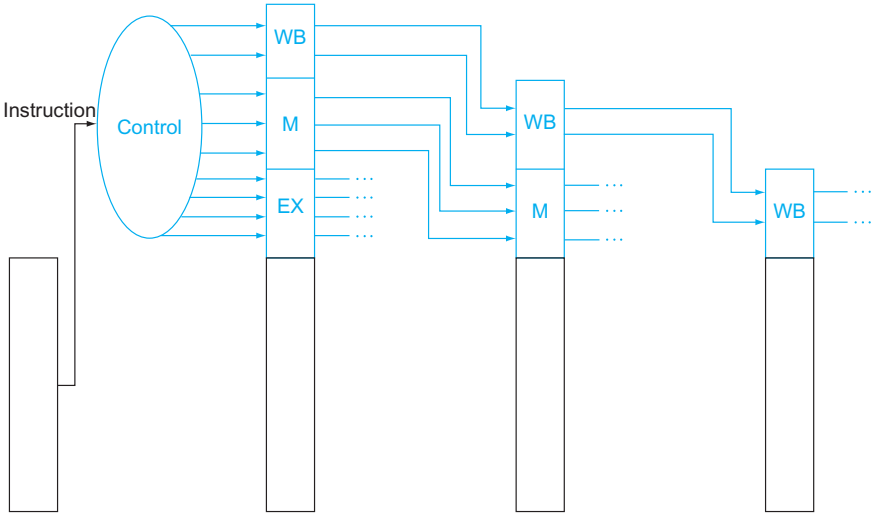
Mostrando o ciclo 5 das figuras anteriores



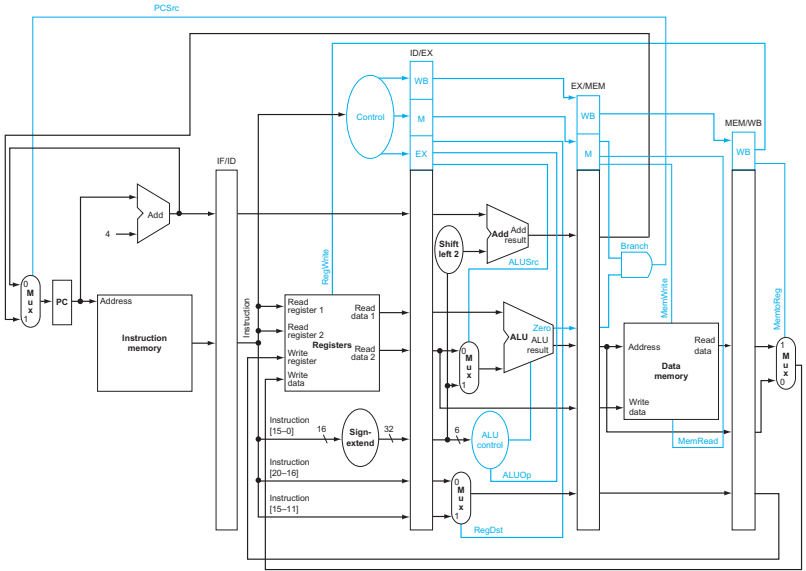
Controle do pipeline (simplificado)



Controle do pipeline



Controle do pipeline



Para saber mais sobre pipeline e hazards de dados, controle e exceções

UFABC

- O livro traz na sua seção 4.7 como o pipeline deve ser alterado para levar em consideração os hazards e fazer o forwarding/bypassing.
- A seção 4.9 descreve o tratamento de interrupções e exceções.
- Aqui não vamos entrar nestes detalhes, mas caso queira ainda mais detalhes do que no livro [PH] vejam o livro [HP] disponível na bibliografia da disciplina.
- Vamos dar apenas uma rápida olhada sobre o branch predictor.

Dynamic branch predictor

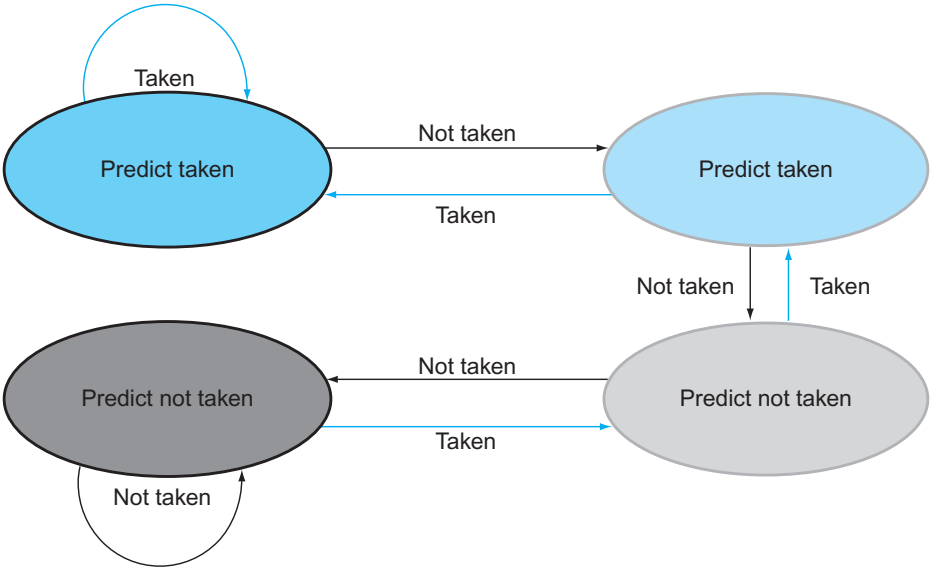
UFABC

- Em processadores com um pipeline longo, o custo de chutar um branch incorreto pode ser muito caro
- A solução é usar algo mais certeiro que sempre chutar que um branch será tomado ou não
- Dynamic branch predictors usam uma predição dinâmica
  - Branch predictor buffer (ou branch predictor table)
  - Indexado pelos endereços dos branches mais recentes
  - Guarda histórico do branch: se foi tomado ou não
- Para executar um branch
  - Verifica a tabela, chuta o mesmo resultado
  - Começa a execução da instrução apontada pelo chute
  - Se incorreto, limpa (*flush*) o pipeline e atualiza a tabela

- Têm um problema de errar a predição duas vezes em laços aninhados!
  - Erram na última iteração do laço
  - Erram na primeira iteração do laço

```
1 outer: ...
2        ...
3 inner: ...
4        beq ..., ..., inner
5        ...
6        beq ..., ..., outer
```

Preditores de 2 bits



- Correlating predictors
  - Mantem 2 automatos, um para quando um branch foi tomado pela última vez e outro para quando não foi tomado. Mostra-se que isso traz um aumento significativo do acerto.
- Tournament predictors
  - Mantém 2 preditores para cada branch. Conforme o branch vai executando, os preditores são atualizados e a cada momento o que teve mais acertos históricos é utilizado para prever o resultado. Alguns processadores recentes usam este esquema.

UFABC

- Uma maneira de aumentar o paralelismo é aumentar o comprimento do pipeline (mas isto traz, como já vimos, outros problemas)
- Alguns processadores são capazes de executar mais de uma instrução ao mesmo tempo no mesmo estágio do pipeline
  - Para isto, eles incluem hardware adicional para evitar conflitos.
  - Isso, na nossa analogia, é equivalente a instalar diversas máquinas de lavar, secar, ...
  - O nome geral que se dá a essa técnica é *multiple issue*

Multiple issue

UFABC

- Issue slots são cada uma das posições nas quais as instruções podem ser selecionadas para execução em um único ciclo.
- Issue packet é o conjunto de instruções que será selecionada para execução em um único ciclo
- Estático
  - As decisões de quais instruções serão executadas ao mesmo tempo é definida em tempo de compilação.
  - Em arquiteturas VLIW (*very long instruction word*) as diversas instruções independentes são empacotadas em uma única instrução (tipicamente com diversos campos opcode diferentes)
- Dinâmico
  - As decisões de quais instruções serão executadas ao mesmo tempo é definida pelo processador em tempo de execução.
  - Normalmente o compilador já fez um primeiro passo de otimização.

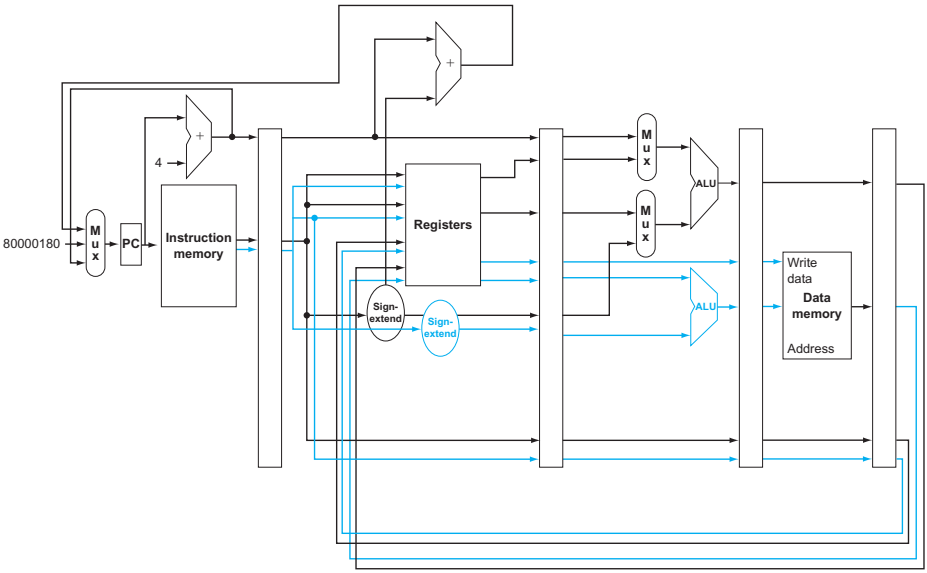
UFABC

- O compilador é o responsável por remover todos (ou quase) os hazards
  - Precisa reordenar instruções dentro de issue packets
  - Não pode haver dependêcias dentro de um pacote
  - Talvez seja possível haver dependências entre pacotes
    - Mas depende da ISA! O compilador precisa saber dos detalhes
  - Completa pacote com `nops` caso necessário

- Pacotes com dois slots
  - Um para instruções ALU/Branch
  - Um para instruções load/store
  - Precisam estar alinhados em fronteiras de 64 bits

Instruction type	Pipe stages							
ALU or branch instruction	IF	ID	EX	MEM	WB			
Load or store instruction	IF	ID	EX	MEM	WB			
ALU or branch instruction		IF	ID	EX	MEM	WB		
Load or store instruction		IF	ID	EX	MEM	WB		
ALU or branch instruction			IF	ID	EX	MEM	WB	
Load or store instruction			IF	ID	EX	MEM	WB	
ALU or branch instruction				IF	ID	EX	MEM	WB
Load or store instruction				IF	ID	EX	MEM	WB

MIPS com dual-issue estático



Hazards no MIPS com dual-issue

UFABC

- Mais instruções executando em paralelo.
- EX data hazard
  - Forwarding resolveu os stalls com um sistema de single-issue
  - Agora não podemos usar o resultado da ALU em um load/store no mesmo pacote

```
1 add  $t0, $s0, $s1
2 load $s2, 0($t0)
```

  - É preciso quebrar em dois pacotes (essencialmente estamos fazendo um stall)
- Load-use hazard
  - Ainda há uma latência para usarmos de um ciclo, mas passa a ser uma espera de 2 instruções
- É necessário um mecanismo de escalonamento mais agressivo por parte do compilador

Exemplo de escalonamento

UFABC

Queremos escalonar:

```
Loop: lw    $t0, 0($s1)    # $t0=array element
      addu  $t0,$t0,$s2# add scalar in $s2
      sw    $t0, 0($s1)# store result
      addi  $s1,$s1,-4# decrement pointer
      bne   $s1,$zero,Loop# branch $s1!=0
```

Versão otimizada:

	ALU or branch instruction	Data transfer instruction	Clock cycle
Loop:		lw $t0, 0($s1)	1
	addi $s1,$s1,-4		2
	addu $t0,$t0,$s2		3
	bne $s1,$zero,Loop	sw $t0, 4($s1)	4

- IPC = 5/4 = 1.25 (ideal IPC = 2)

Desenrolando laços

UFABC

- Desenrolar laços (*loop unroling*) permite que aproveitemos melhor os recursos do processador
- Replicamos o corpo do laço para expor mais paralelismo
  - Reduz o overhead do controle do laço
- Utilizamos diferentes registradores por iteração
  - Chamado de ”register renaming”
  - Evita anti-dependências *loop-carried*
    - Ex. Store seguido de load de um mesmo registrador
    - Também chamado de dependência de nome
    - Ocorre quando se reusa o nome de um registrador

Desenrolando laços - Exemplo

	ALU or branch instruction		Data transfer instruction		Clock cycle
Loop:	addi	$s1,$s1,-16	lw	$t0, 0($s1)	1
			lw	$t1,12($s1)	2
	addu	$t0,$t0,$s2	lw	$t2, 8($s1)	3
	addu	$t1,$t1,$s2	lw	$t3, 4($s1)	4
	addu	$t2,$t2,$s2	sw	$t0, 16($s1)	5
	addu	$t3,$t3,$s2	sw	$t1,12($s1)	6
			sw	$t2, 8($s1)	7
	bne	$s1,$zero,Loop	sw	$t3, 4($s1)	8

- IPC = 14/8 = 1.75 (ideal IPC = 2)
  - Estamos mais próximos de 2, mas pagamos em número de registradores usados e tamanho do código

Processadores superescalares

UFABC

- São processadores com multiple issue dinâmico
- O próprio processador decide se deve executar 0, 1, 2, ... instruções por ciclo
  - Também é responsável por cuidar de hazards estruturais e de dados
- Diminui a pressão no escalonamento feito pelo compilador
  - Apesar de ainda ser útil
  - A semântica do código é garantida pela CPU
- Algumas CPUs são capazes de escolher as instruções a serem executadas fora de ordem (*out-of-order*)
  - Mas ainda dão a ilusão ao usuário que as instruções estão sendo executadas em ordem

Para que fazer escalonamento dinâmico?

UFABC

- Se não fazer simplificaria o hardware e o escalonamento poderia ser feito pelo compilador?
  - Alguns stalls não podem ser determinados em tempo de compilação
    - Ex. Falhas de cache
  - Não é possível fazer o escalonamento no caso de branches
    - Já que por definição é determinado dinamicamente
  - Isto tudo é muito dependente da ISA em questão.

Multiple-issue funciona?

UFABC

- Sim! Mas não tanto quanto gostaríamos.
- Programas têm dependências que limitam o ILP.
  - Algumas são dificílimas de eliminar.
    - Ex. Pointer aliasing.
- Alguns tipos de paralelismo são difíceis de expor.
  - Tamanho de janela limitada durante o issue de instruções.
- Há atrasos no acesso à memória e uma banda limitada.
  - É difícil de manter o pipeline cheio.
- Especulação pode ajudar muito, se for bem feita.

- A complexidade de escalonamentos dinâmicos e especulação exigem gasto de energia.
- Dependendo da aplicação, múltiplos núcleos simples podem ser mais vantajosos.

Microprocessor	Year	Clock Rate	Pipeline Stages	Issue Width	Out-of-Order/ Speculation	Cores/ Chip	Power	
Intel 486	1989	25 MHz	5	1	No	1	5	W
Intel Pentium	1993	66 MHz	5	2	No	1	10	W
Intel Pentium Pro	1997	200 MHz	10	3	Yes	1	29	W
Intel Pentium 4 Willamette	2001	2000 MHz	22	3	Yes	1	75	W
Intel Pentium 4 Prescott	2004	3600 MHz	31	3	Yes	1	103	W
Intel Core	2006	2930 MHz	14	4	Yes	2	75	W
Intel Core i5 Nehalem	2010	3300 MHz	14	4	Yes	1	87	W
Intel Core i5 Ivy Bridge	2012	3400 MHz	14	4	Yes	8	77	W

Arm Cortex-A8 e Intel i7

UFABC

Processor	ARM A8	Intel Core i7 920
Market	Personal Mobile Device	Server, Cloud
Thermal design power	2 Watts	130 Watts
Clock rate	1 GHz	2.66 GHz
Cores/Chip	1	4
Floating point?	No	Yes
Multiple Issue?	Dynamic	Dynamic
Peak instructions/clock cycle	2	4
Pipeline Stages	14	14
Pipeline schedule	Static In-order	Dynamic Out-of-order with Speculation
Branch prediction	2-level	2-level
1st level caches / core	32 KiB I, 32 KiB D	32 KiB I, 32 KiB D
2nd level cache / core	128 - 1024 KiB	256 KiB
3rd level cache (shared)	–	2 - 8 MiB