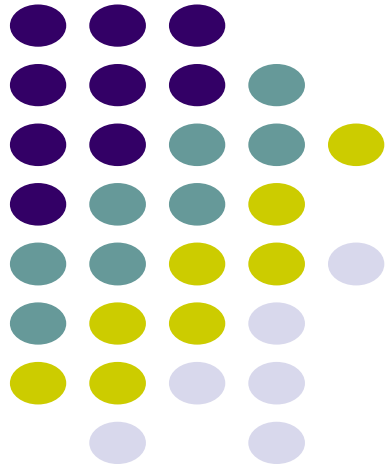
UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO
DEPARTAMENTO DE CIÊNCIAS EXATAS E NATURAIS
CURSO DE CIÊNCIA DA COMPUTAÇÃO

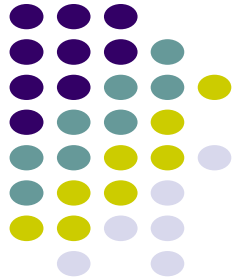
Arquitetura e Organização de Computadores

O Processador: caminho de dados



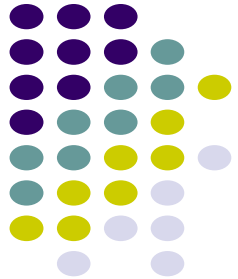
Prof. Sílvio Fernandes

Introdução



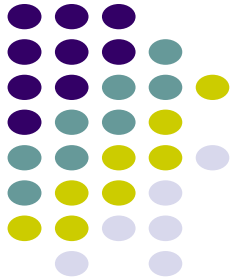
- Uma implementação MIPS básica
  - Instruções de referência a memória (lw e sw)
  - Instruções lógicas e aritméticas (add, sub, and, or e slt)
  - Instruções branch equal (beq) e jump (j)
- Os princípios básicos usados para criação do caminho de dados de outras instruções são semelhantes a estas
- Examinando várias estratégias de implementação, teremos a oportunidade de ver como o conj. Inst. afeta a velocidade de clock e o CPI da máquina

Uma sinopse da implementação



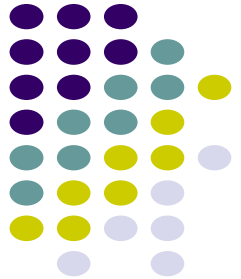
- Muito do que precisa ser feito para implementar as 3 classes de instruções citadas antes, é igual
- Para cada instrução
  1. Enviar o contador de programa (PC) à memória que contém o código e buscar a instrução dessa memória
  2. Ler um ou mais registradores, usando campos da instrução para selecionar os registradores a serem lidos.
- Em seguida, as ações para completar a instrução dependem da classe da instrução

Uma sinopse da implementação



- Mesmo entre diferentes classes, há algumas semelhanças
  - Todas as classes de instrução (exceto jump), usam a ALU após a leitura dos registradores
- Após usar a ALU, as ações necessárias para completar várias classes de instruções diferem
- Vamos entender como é a **microarquitetura** do MIPS: quais as "peças" lógicas (Alu, registradores, máquinas de estado finito, memórias e outros circuitos combinacionais) e como elas são conectadas para executar as instruções

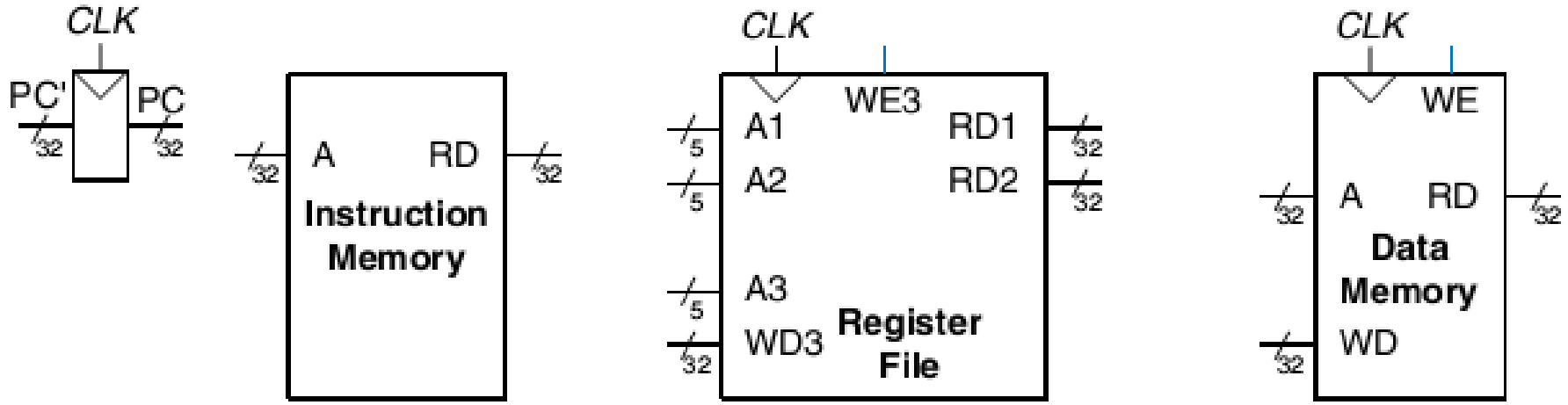
Uma sinopse da implementação



- Dividiremos nosso projeto em 2 partes
  - Caminho de dados (*datapath*)
  - Controle

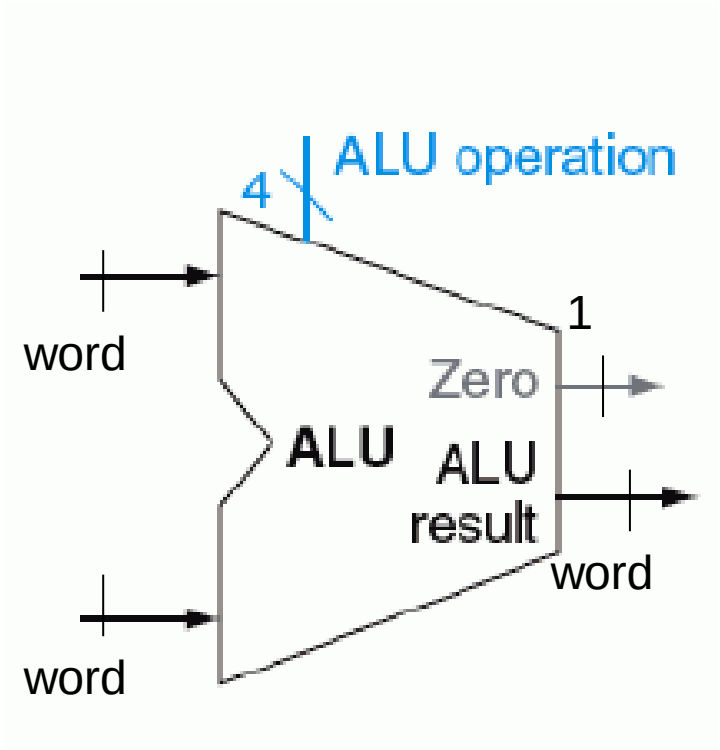
Componentes do Datapath

- Um bom início é começar pelos elementos de estado
  - Contém estado se tiver armazenamento interno
  - As entradas necessárias são os valores dos dados a serem escritos e o clock, que determina quando o valor dos dados deve ser escrito
  - São elementos *sequenciais*: suas saídas dependem de suas entradas e do conteúdo do estado interno

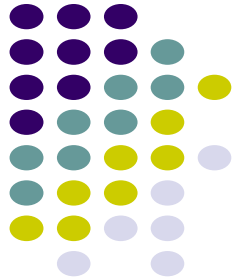


Componentes do Datapath

- A ALU é um circuito combinacional que realiza operações lógicas ou aritméticas no dados de entrada. A escolha do tipo da operação é indicada na entrada "ALU operation"



Componentes do Datapath

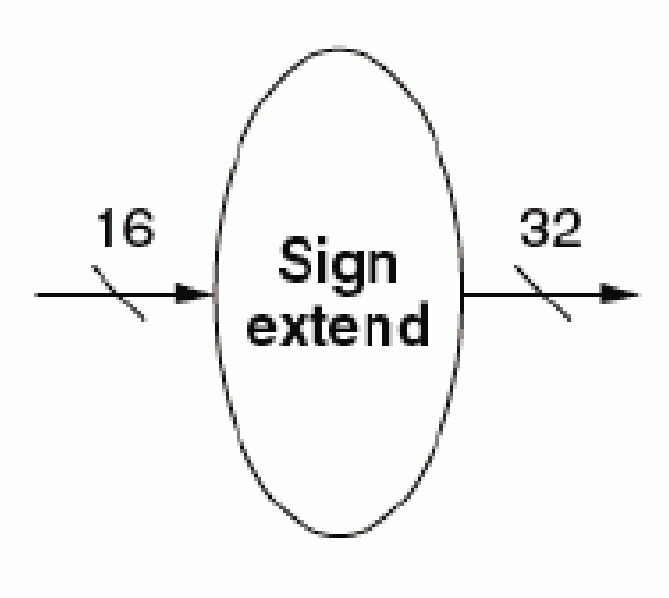


- A ALU usa 2 entradas de 32 bits e produz um resultado de 32 bits, bem como um sinal de 1 bit se o resultado for 0
- O sinal de controle (4 bits) da ALU define a operação a ser realizada por ela

Linhas de controle da ALU	Função
0000	AND
0001	OR
0010	add
0110	subtract
0111	set on less than
1100	NOR

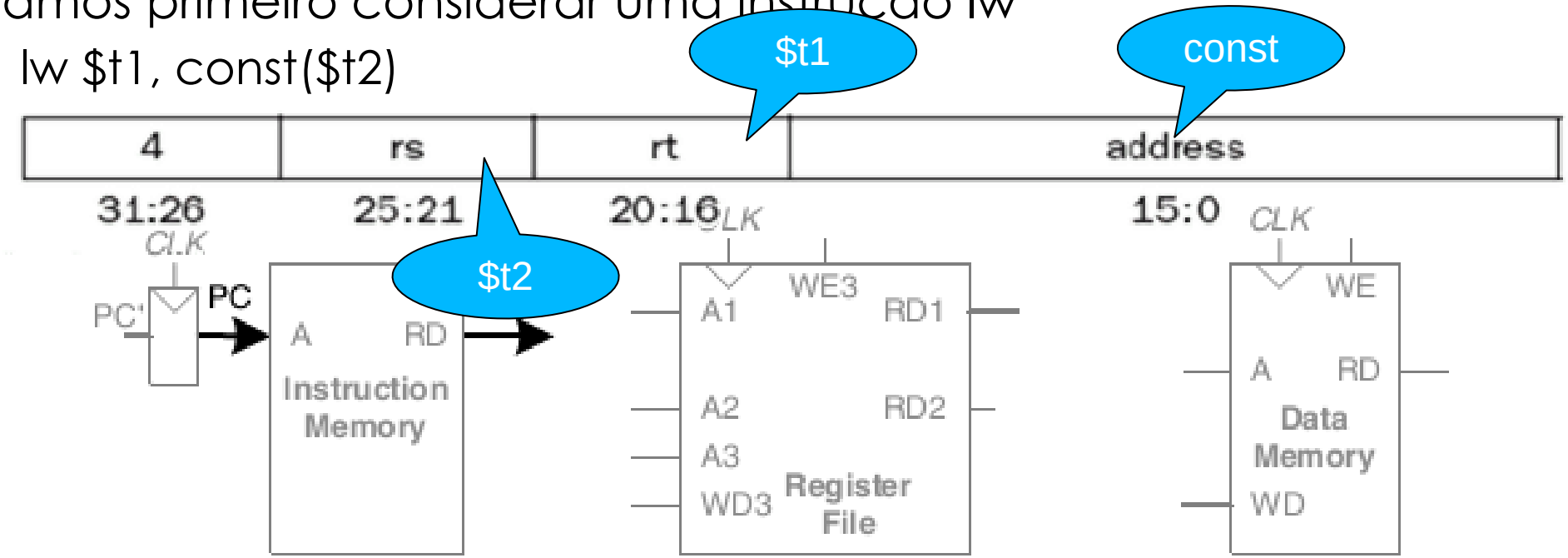
Componentes do Datapath

- Outro circuito combinacional é o extensor de sinal. Ele transforma número de 16 bits em 32 bits mantendo o sinal



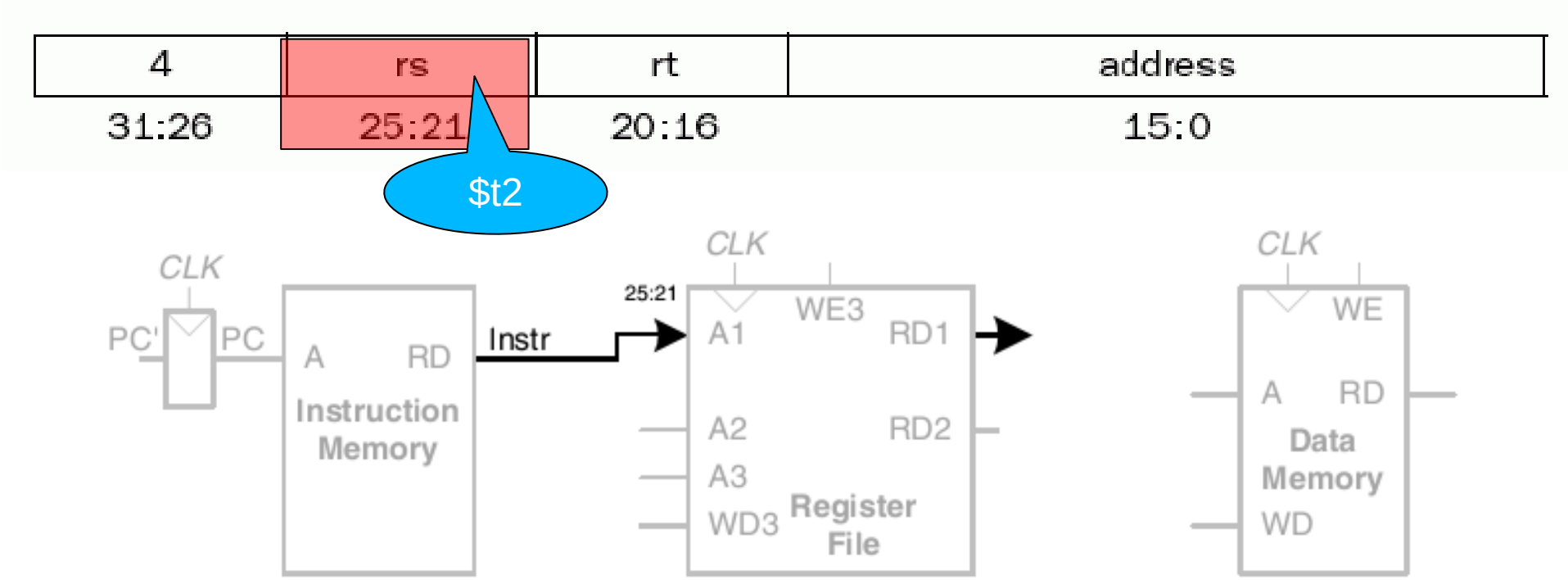
Componentes do Datapath

- Buscar instrução da memória
  - O valor armazenado no registrador PC é enviado como endereço para memória
  - A memória devolve uma instrução
  - Vamos primeiro considerar uma instrução lw
    - lw $t1, const($t2)



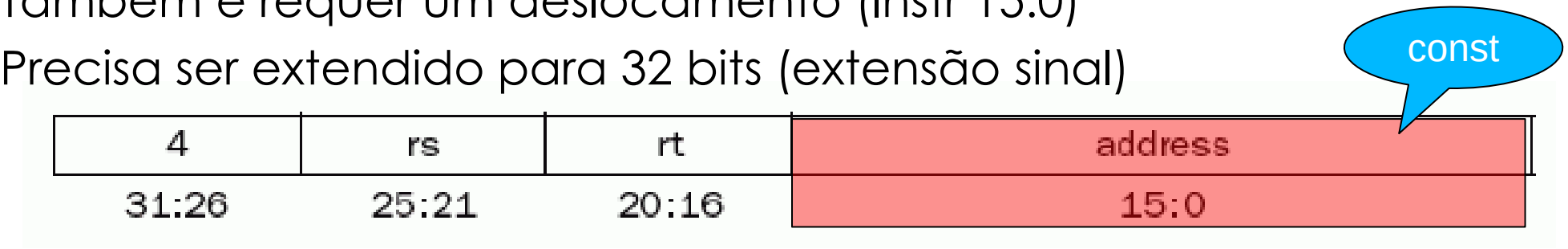
Componentes do Datapath

- Para instrução **lw**
  - Próximo passo é ler o reg. fonte no campo *rs* da instrução (Instr 25:21)
    - lw $t1, const($t2)

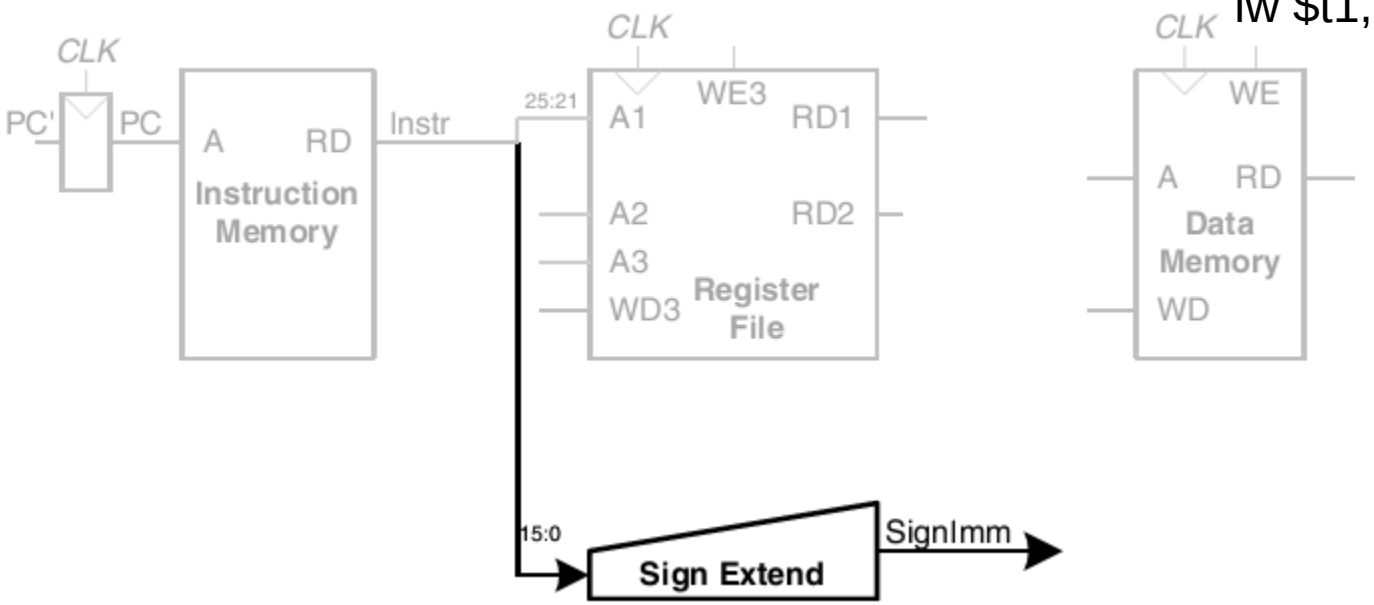


Componentes do Datapath

- Para instrução **lw**
  - Também é requer um deslocamento (Instr 15:0)
  - Precisa ser extendido para 32 bits (extensão sinal)

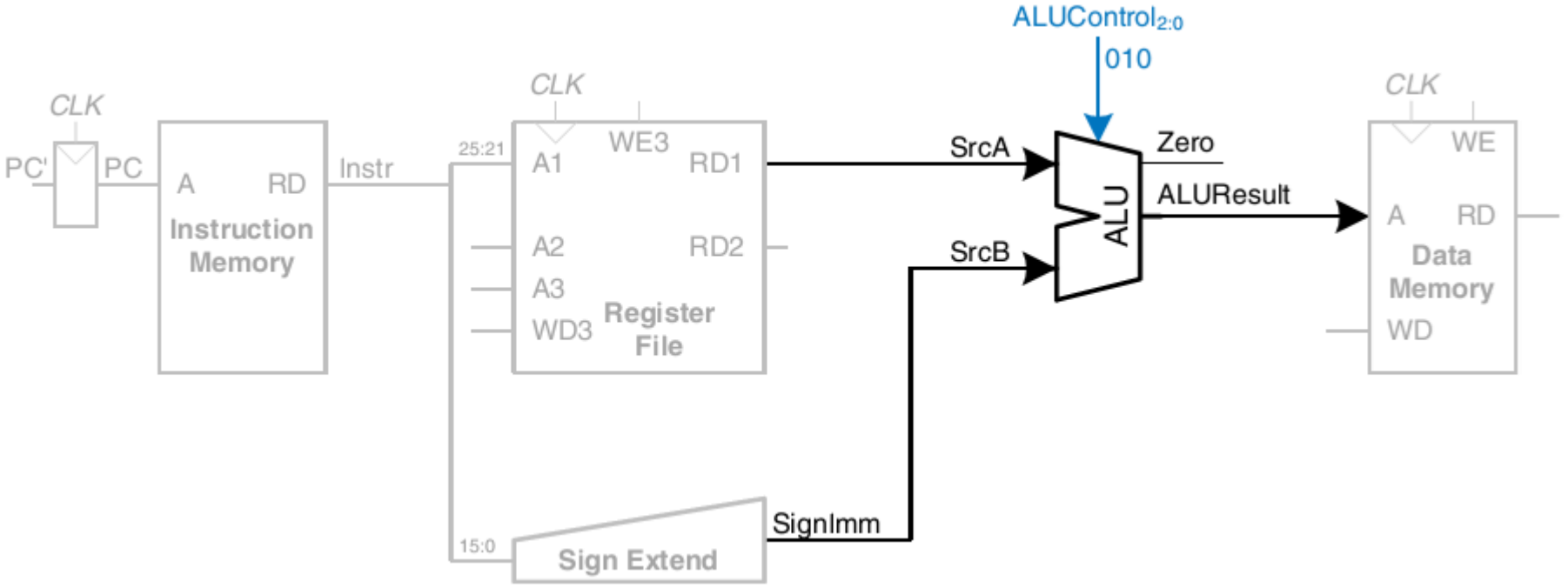


lw $t1, const($t2)



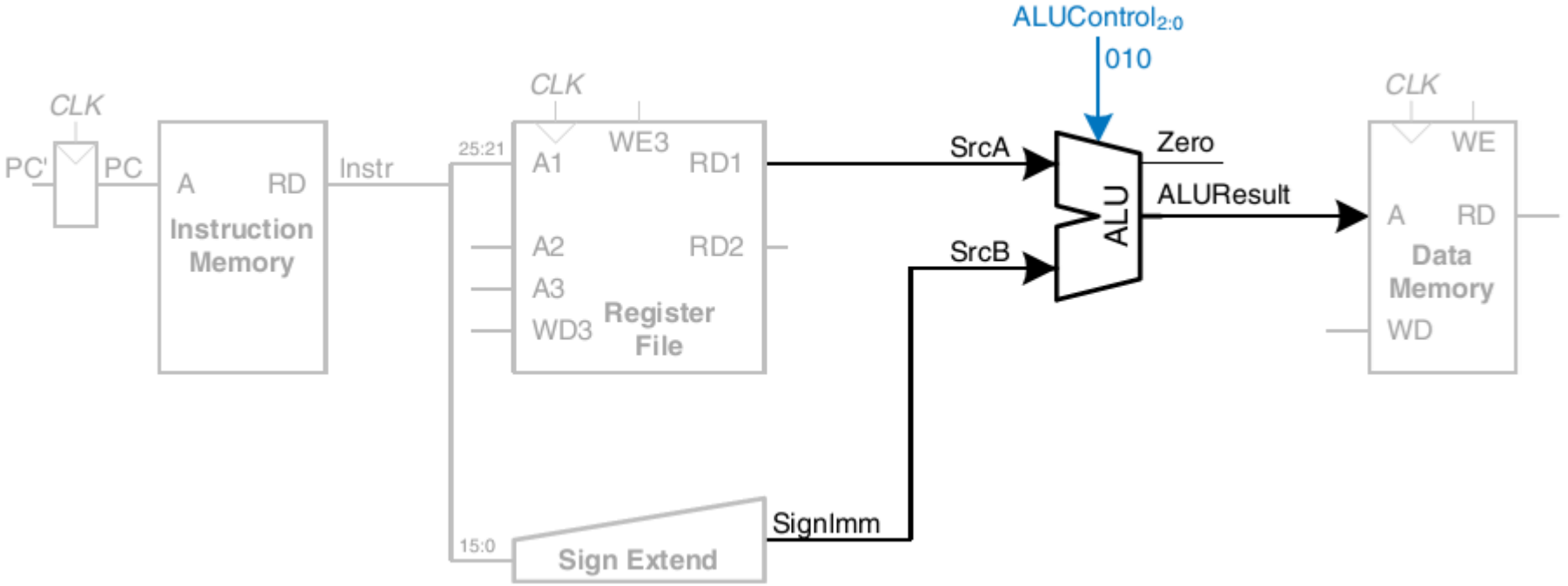
Componentes do Datapath

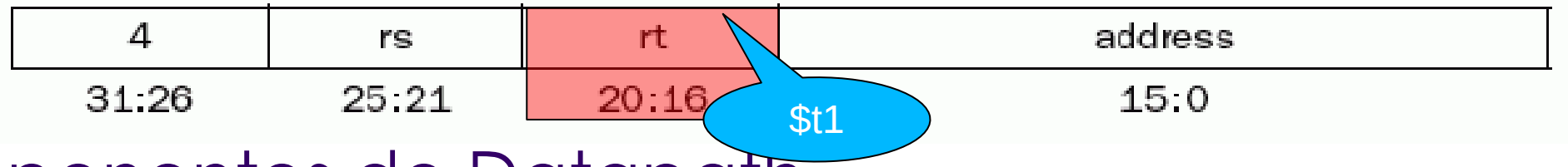
- Para instrução **lw**
  - O processador precisa somar o endereço de base com o *offset* para encontrar o endereço de mem.
  - Acrescentamos uma ALU (unidade lógica e aritmética)



Componentes do Datapath

- Para instrução **lw**
  - A ALU faz a operação especificada por "ALUControl" sobre os dados (SrcA e SrcB) de 32 bits
  - Gera um resultado de 32 bits e uma flag indicando se o resultado é igual a zero

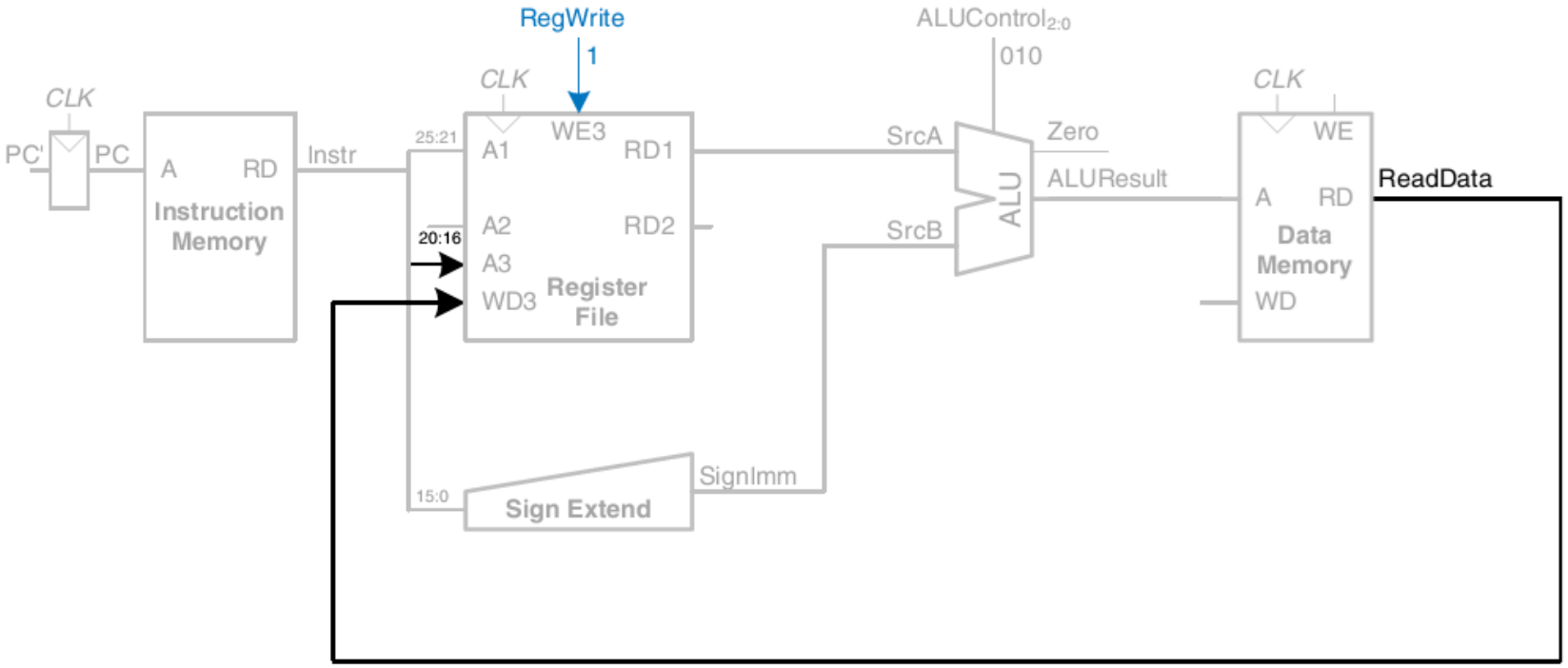




Componentes do Datapath

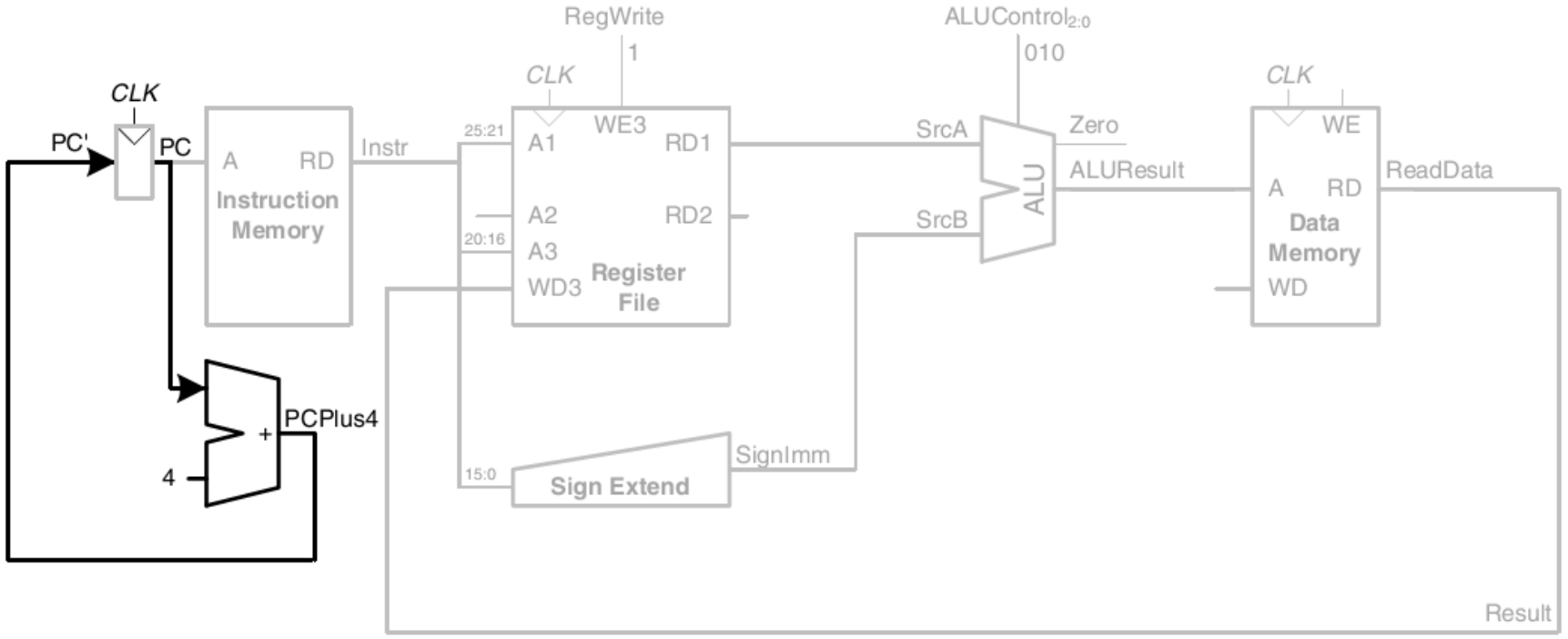
lw $t1, const($t2)

- Para instrução **lw**
  - ALUResult é enviado para memória como endereço
  - O valor lido é enviado para o banco de registradores – campo rt de lw (Intr 20:16) no final do ciclo



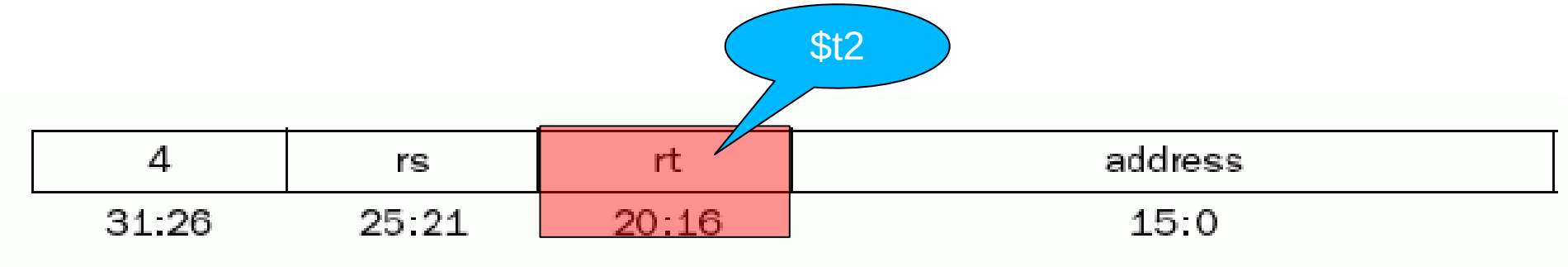
Componentes do Datapath

- Para instrução **lw**
  - Enquanto a instrução é executada (dentro de 1 ciclo) o processador deve calcular o end. da próxima instrução: PC'



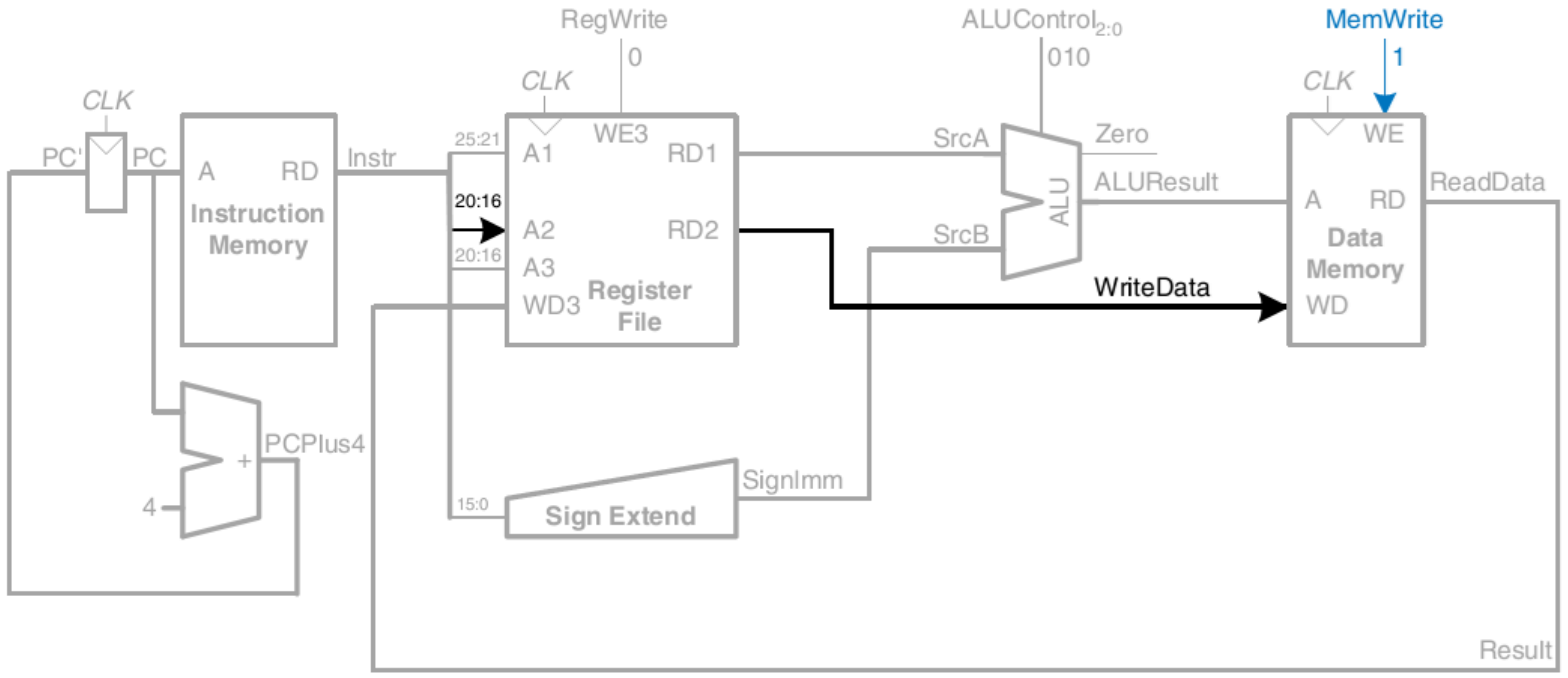
Componentes do Datapath

- Para instrução **sw**
  - **sw $t1, const($t2)**
  - Ler o endereço base do banco de reg., extende o sinal, usa a ALU para somá-los e usar como endereço de mem.
  - O 2º reg. (Instr 20:16) do banco informa o dado a ser escrito



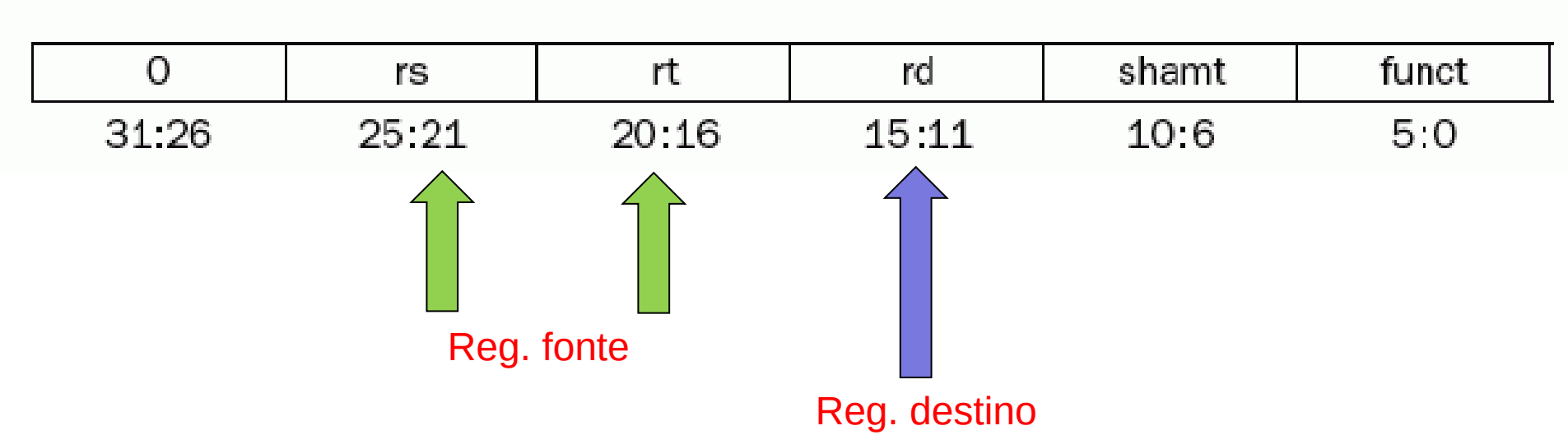
Componentes do Datapath

- Para instrução **sw**
  - Ler o endereço base do banco de reg., extende o sinal, usa a ALU para somá-los e usar como endereço de mem.
  - O 2º reg. (Instr 20:16) do banco informa o dado a ser escrito



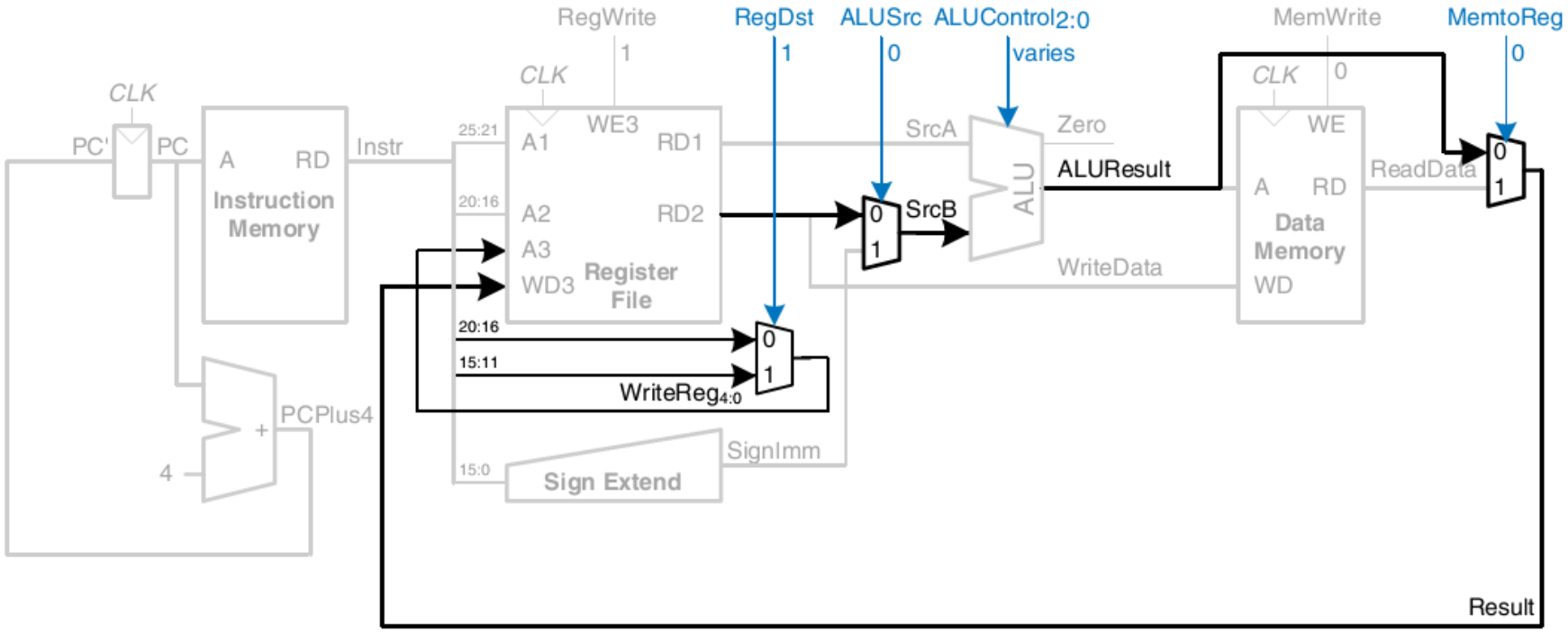
Componentes do Datapath

- Para instruções do tipo R
  - Instruções **add**, **sub**, **and**, **or** e **slt**
  - Ler 2 reg. do banco, faz alguma operação na ALU e escreve o resultado no banco reg.
    - add $t1, $t2, $t3

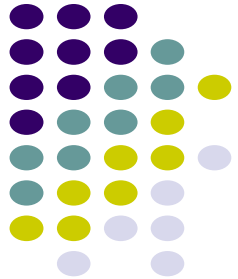


Componentes do Datapath

- Para instruções do tipo R
  - Instruções **add**, **sub**, **and**, **or** e **slt**
  - Ler 2 reg. do banco, faz alguma operação na ALU e escreve o resultado no banco reg.



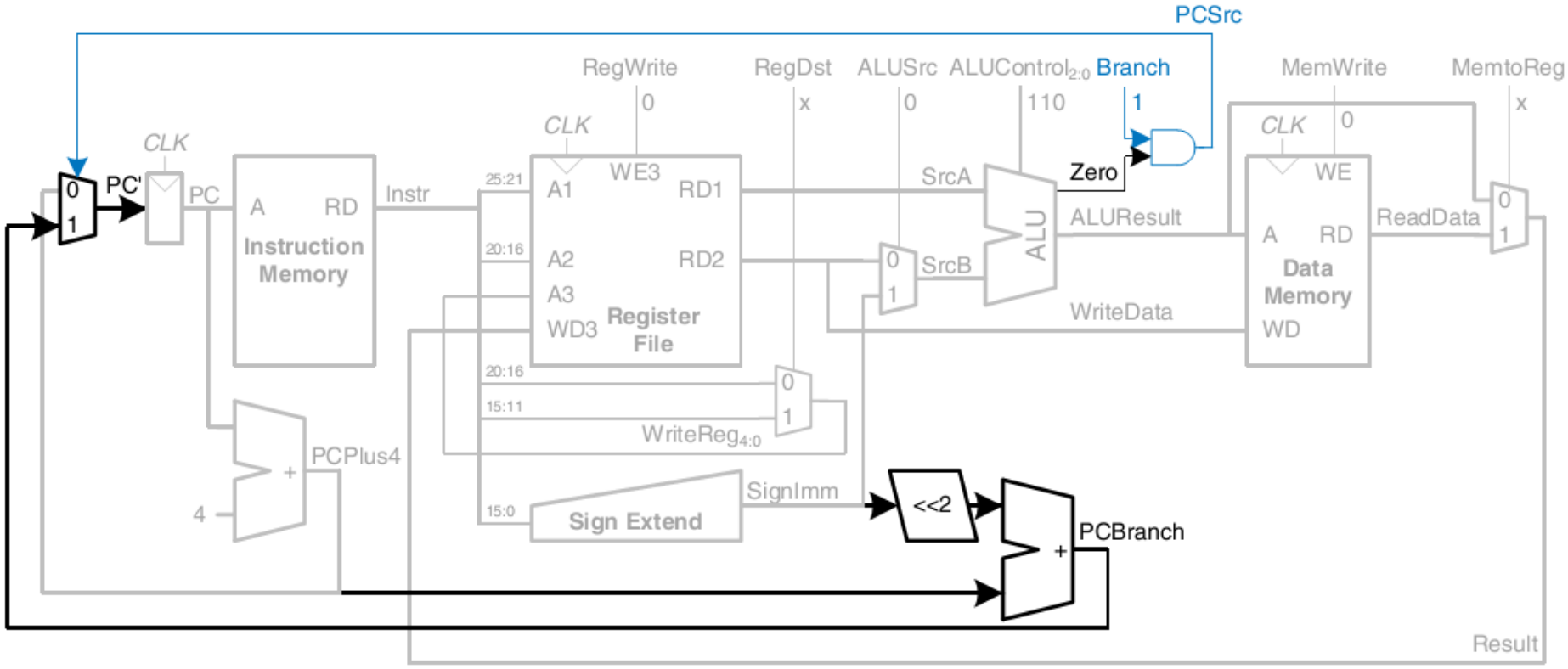
Componentes do Datapath



- Para instruções **beq** e **bne**
  - Compara 2 reg. e se for verdade realiza o salto modificando PC a partir do *label*
  - O valor imediato armazena a quantidade de palavras e por isso deve ser multiplicado por 4 antes de ser somado ao PC
    - beq $t1, $t2, label

Componentes do Datapath

- Para instruções **beq** e ~~**bne**~~



Componentes do Datapath

- Para instrução **addi**
  - addi $t1, $t2, const

codop	rs	rt	address
31:26	25:21	20:16	15:0

  - Nada precisa ser modificado
    - O 1° reg. é lido do banco de reg.
    - A const. passa pelo extensor de sinal
    - A operação é feita na ALU
    - O resultado é escrito no banco no end. do reg. Indicado pela instrução

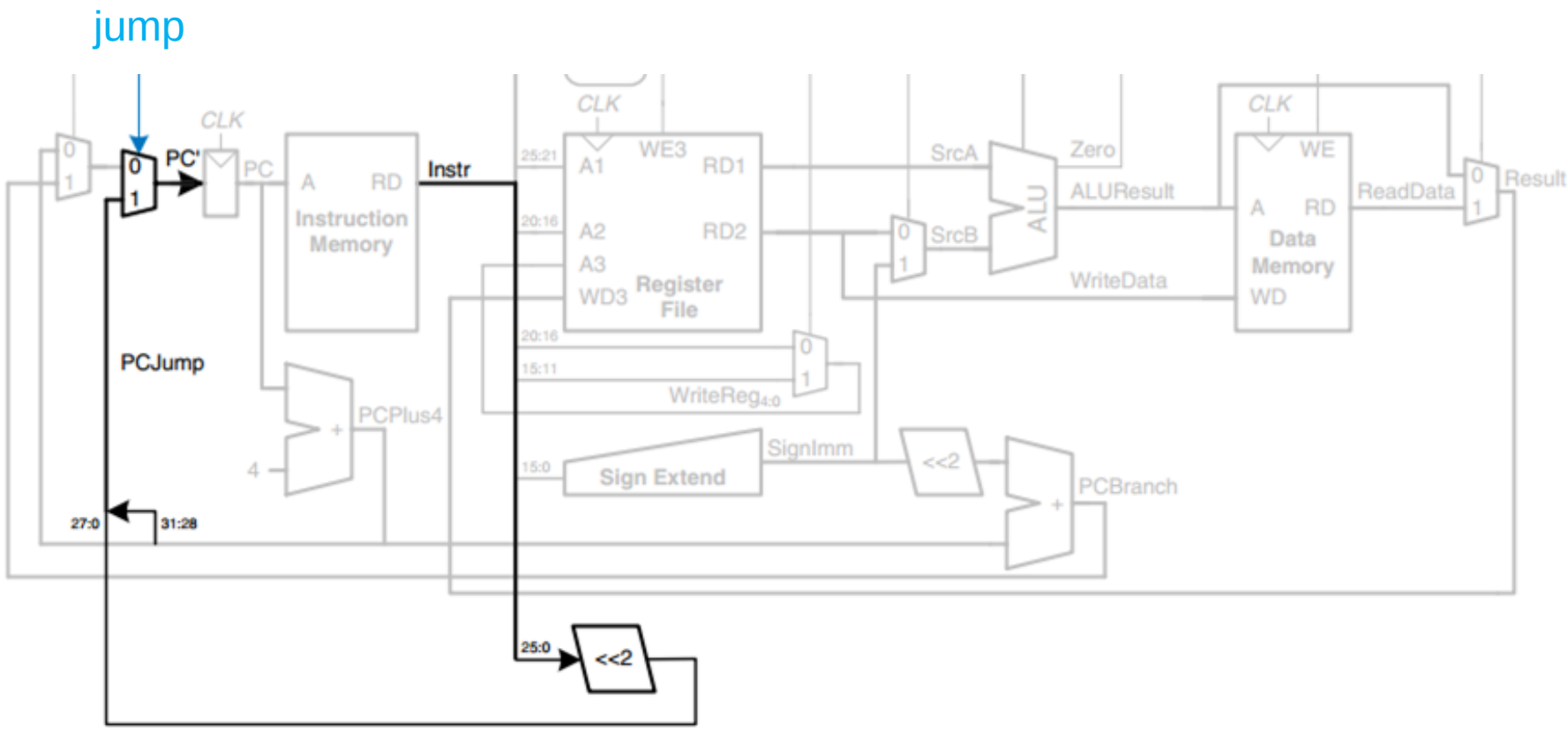
Componentes do Datapath

- Para instrução **j**
  - j label

  - Funciona substituindo os 28 bits menos significativos do PC pelos 26 menos significativos da instrução deslocados de 2 bits á esquerda
  - Esse deslocamento é realizado simplesmente concatenando "00" ao offset do jump

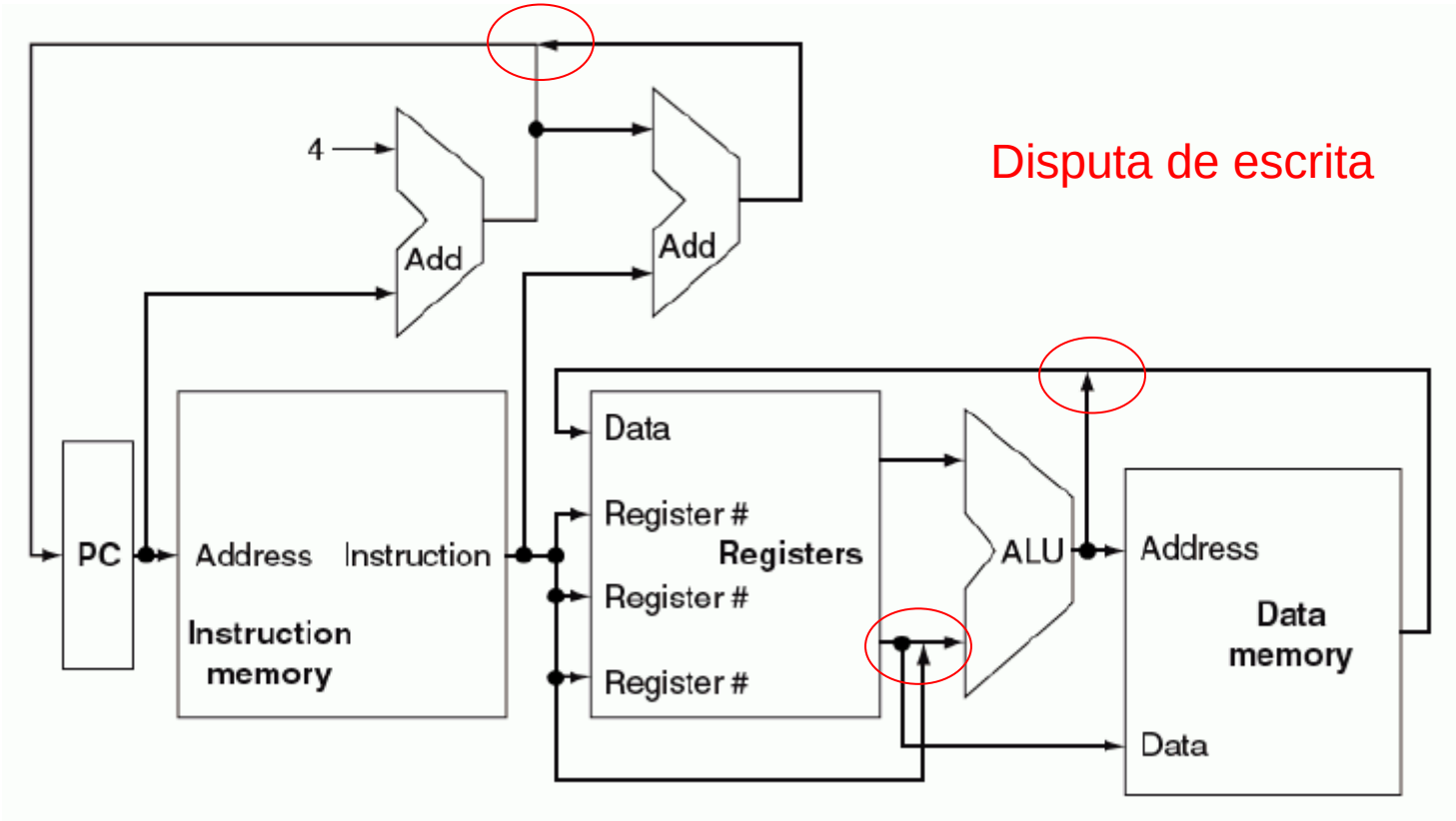
Componentes do Datapath

- Para instrução **j**

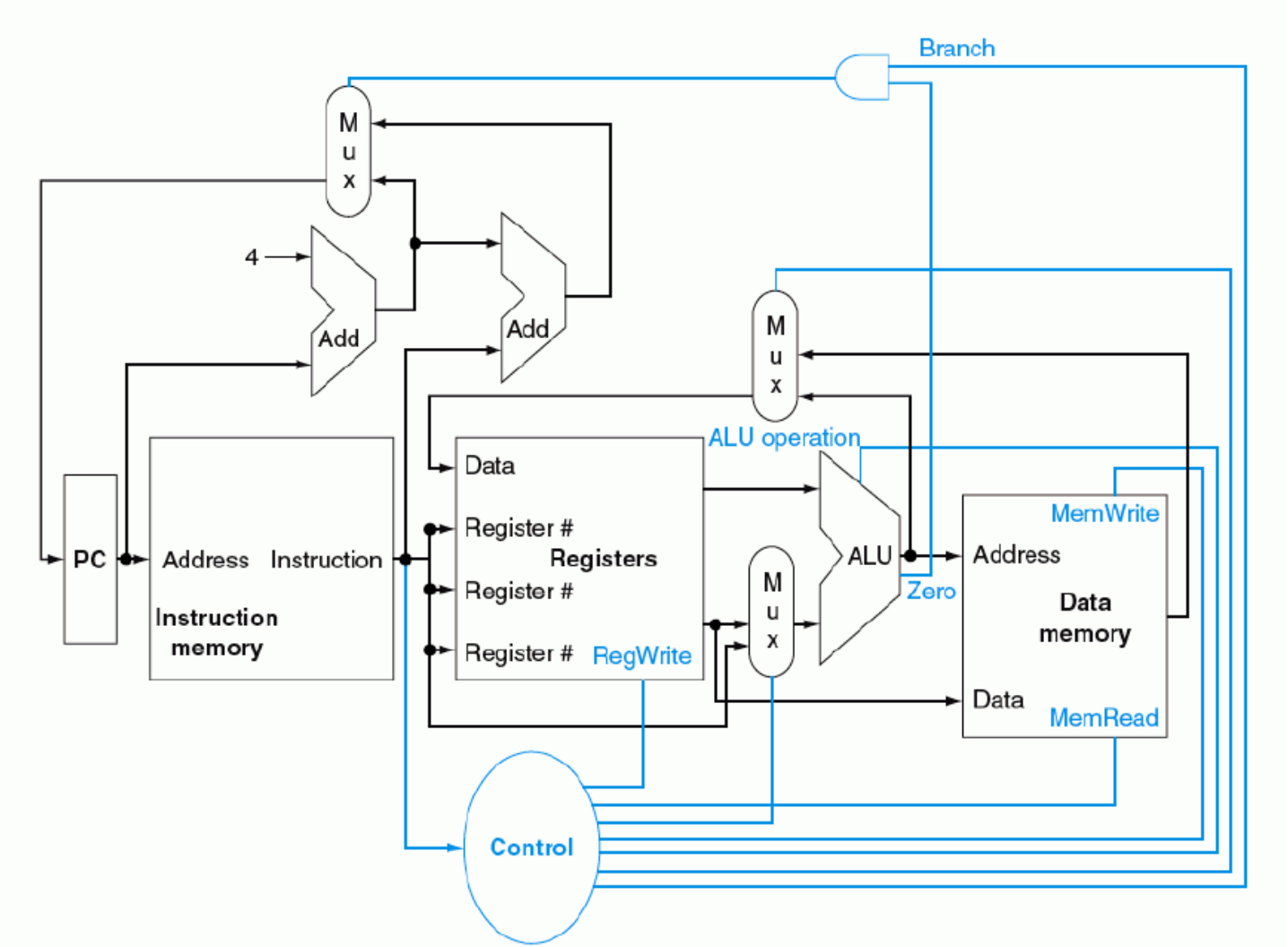


Uma sinopse da implementação

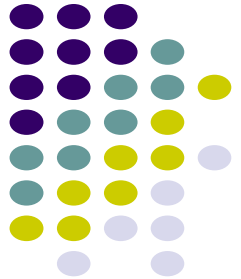
- Uma visão abstrata da implementação do subconjunto MIPS



Uma sinopse da implementação

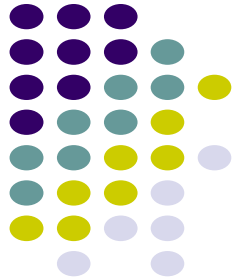


Convenções lógicas de projeto



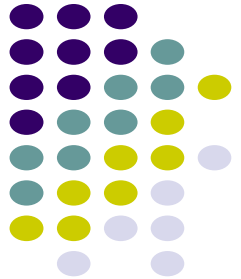
- Unidades funcionais MIPS
  - Elementos que operam nos valores dos dados
    - Combinacionais: suas saídas dependem apenas das entradas atuais (ALU)
  - Elementos que contêm estado
    - Também chamado de **elementos de estado**
    - Contém estado se tiver armazenamento interno
    - As entradas necessárias são os valores dos dados a serem escritos e o clock, que determina quando o valor dos dados deve ser escrito
    - São elementos *sequenciais*: suas saídas dependem de suas entradas e do conteúdo do estado interno

Convenções lógicas de projeto



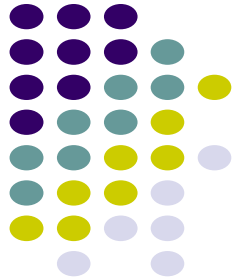
- Unidades funcionais MIPS
  - Elementos que contêm estado
    - Ex: memórias de instruções e dados e registradores
- Usaremos o termo *ativo* para indicar um sinal que está logicamente *alto* e *ativar* para especificar que um sinal deve ser determinado logicamente alto, e *inativo* ou *desativar* para representar o que é logicamente baixo

Metodologia de clocking



- Uma **metodologia de clocking** define quando os sinais podem ser lidos e quando podem ser escritos
- Ela é importante para sincronização das leituras e escritas
- Uma metodologia de **sincronização acionada por transição** significa que quaisquer valores armazenados em um elemento lógico sequencial são atualizados apenas em uma transição do clock

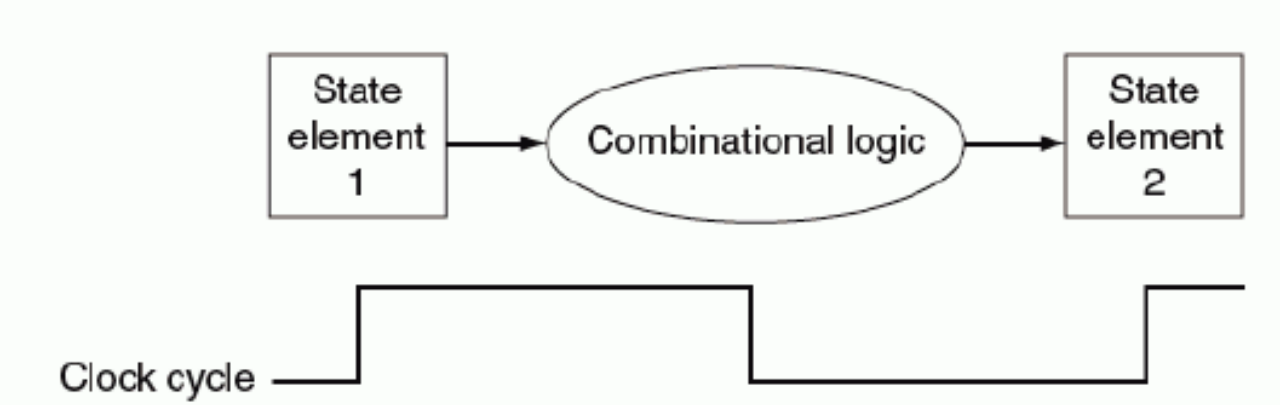
Metodologia de clocking



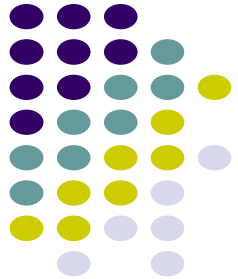
- Como apenas os elementos de estado podem armazenar valores de dados, qualquer coleção de lógica combinatória precisa ter suas entradas vindas de um conjunto de elementos de estado e suas saídas escritas em um conjunto de elementos de estado
- As entradas são valores escritos em um ciclo de clock anterior, enquanto as saídas são valores que podem ser usados em um ciclo de clock seguinte

Metodologia de clocking

- A lógica combinatória, os elementos de estado e o clock estão intimamente relacionados



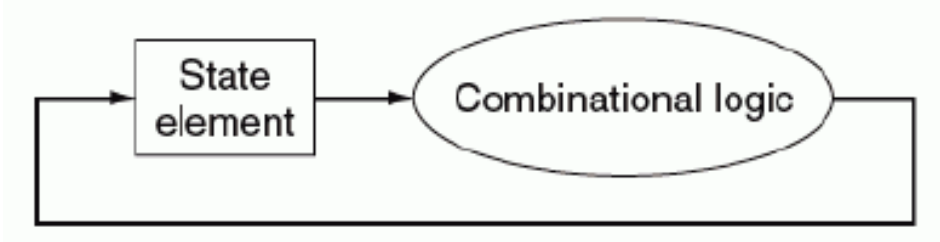
Metodologia de clocking



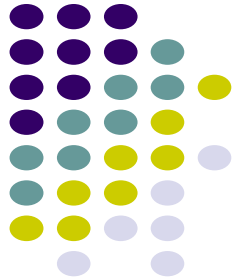
- Para simplificar, não foi mostrado um **sinal de controle** de escrita quando um elemento de estado é escrito a cada transição ativa do clock
- Por outro lado, se um elemento de estado não for atualizado em cada clock, um sinal de controla de escrita explícito é necessário

Metodologia de clocking

- Essa metodologia permite ler o conteúdo de um registrador, enviar o valor por meio de alguma lógica combinatória e escrever nesse registrador no mesmo ciclo de clock

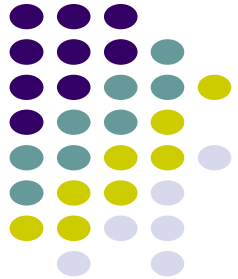


Verifique você mesmo



1. O caminho de dados de ciclo único conceitualmente descrito até aqui *precisa* ter memórias de instrução e de dados separadas porque
   a) O formato dos dados e das instruções é diferente no MIPS, e, portanto, são necessárias diferentes memórias
   b) Ter memórias separadas é menos dispendioso
   c) O processador opera em um ciclo e não pode usar uma memória de porta única para dois acessos diferentes dentro desse ciclo

Verifique você mesmo



2. Verdadeiro ou falso: como o banco de registradores é lido e escrito no mesmo ciclo de clock, qualquer caminho de dados MIPS usando escritas acionadas por transição precisa ter mais de uma cópia do banco de registradores.

Respostas

- 1) a
- 2) falso

Referências

- PATTERSON, D. A. ; HENNESSY, J.L. Organização e projeto de computadores – a interface hardware software. 3. ed. Editora Campus, 2005.
- HARRIS, David M; HARRIS, Sarah L. **Digital Design and Computer Architecture.** 2ed. Elsevier, 2013.