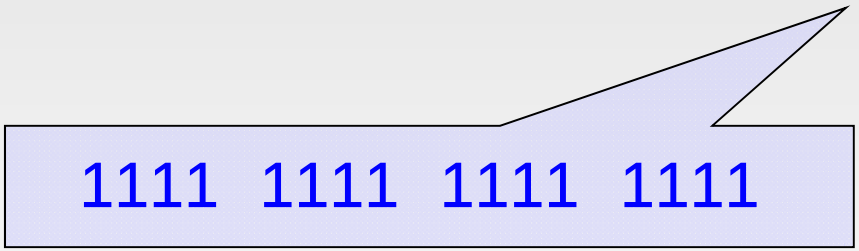
Sistemas Processadores e Periféricos
Aula 5 - Revisão

Prof. Frank Sill Torres
DELT – Escola de Engenharia
UFMG

Adaptado a partir dos Slides de Organização de Computadores 2006/02 do professor Leandro Galvão DCC/UFAM - galvao@dcc.ufam.edu.br e do Prof. Ricardo de Oliveira Duarte (DELT/UFMG)

Ponto fixo
:: Extensão de sinal :: Exemplo

-4_{dec} (16 bits) para 32 bits:



1111 1111 1111 1100 bin

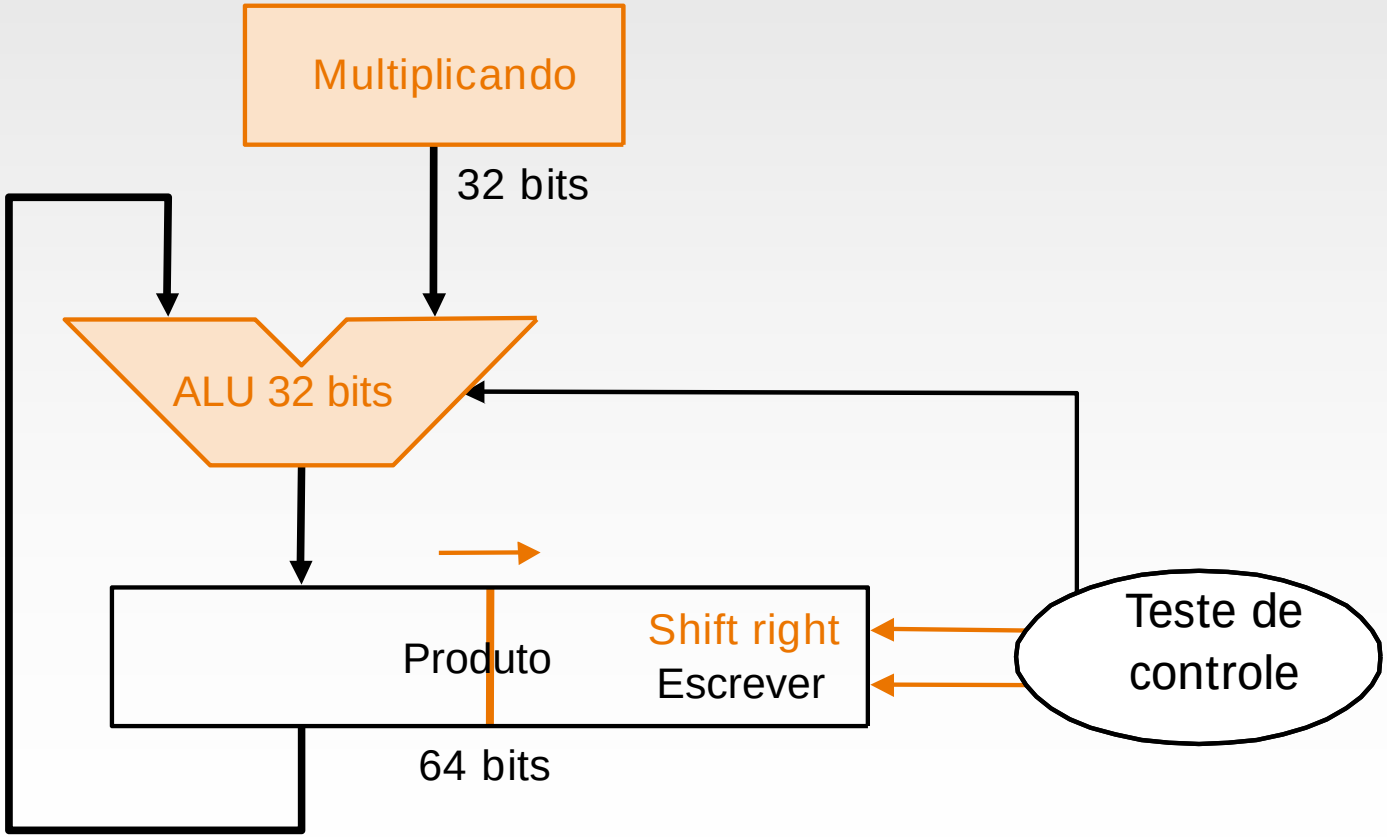
1111 1111 1111 1100 bin

Operações com ponto fixo :: Overflow

- Solução MIPS:
  - Causam exceções no overflow:
    - Adição (**`add`**)
    - Adição imediata (**`addi`**)
    - Subtração (**`sub`**)
  - Não causam exceções no overflow:
    - Adição sem sinal (**`addu`**)
    - Adição imediata sem sinal (**`addiu`**)
    - Subtração sem sinal (**`subu`**)

Operações com ponto fixo :: Multiplicação

- O multiplicador inicia na metade direita do produto



Operações com ponto fixo :: Multiplicação no MIPS

- Produto (64 bits) é colocado em um par de registradores de 32 bits:
  - Hi – armazena a parte mais significativa
  - Lo – armazena a parte menos significativa

- Não gera exceção de overflow

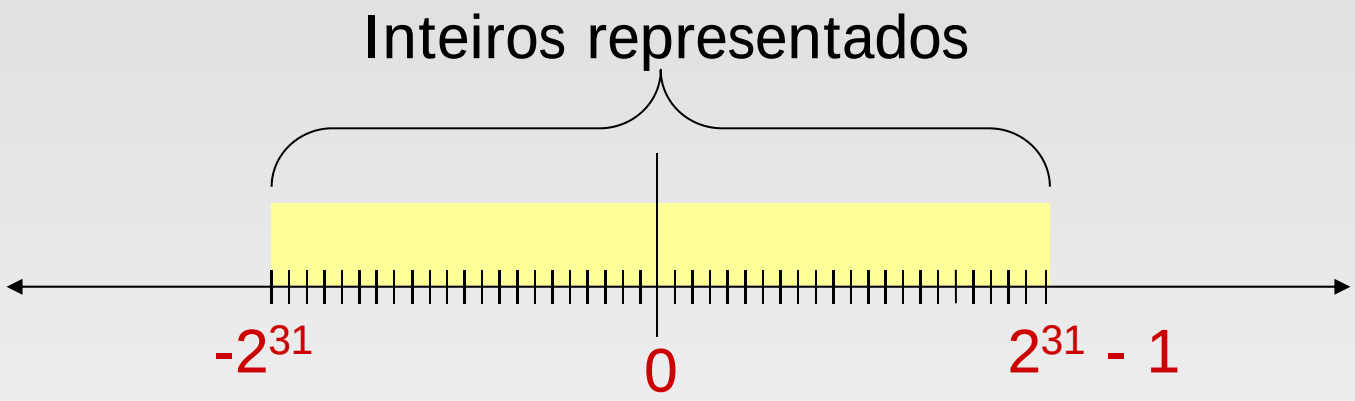
Ponto flutuante (Padrão IEEE 754)

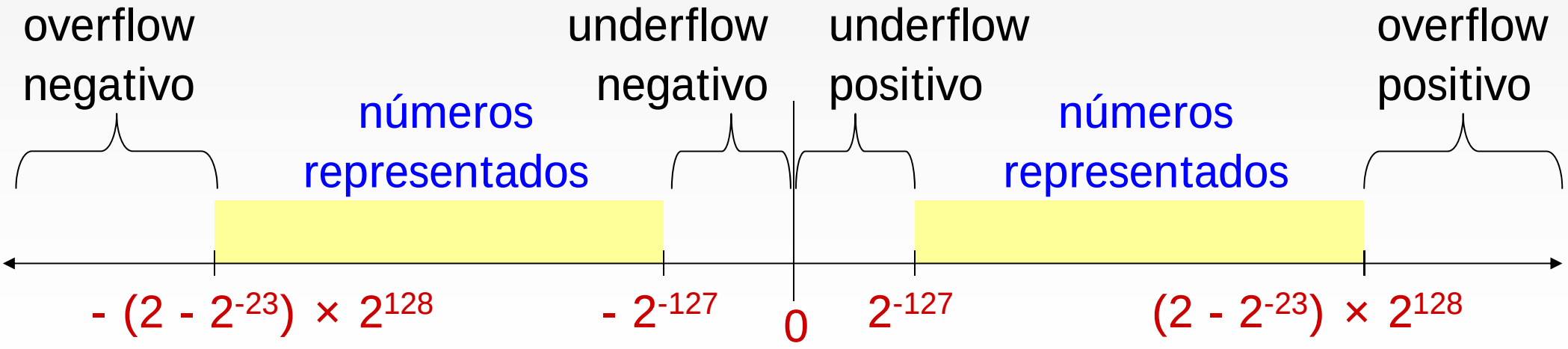
- Um número real pode ser representado no seguinte formato:

$$(-1)^s \times m \times B^e$$

  - s – sinal
  - m – significando (mantissa)
  - B – base
  - e – expoente

Ponto flutuante × Ponto fixo





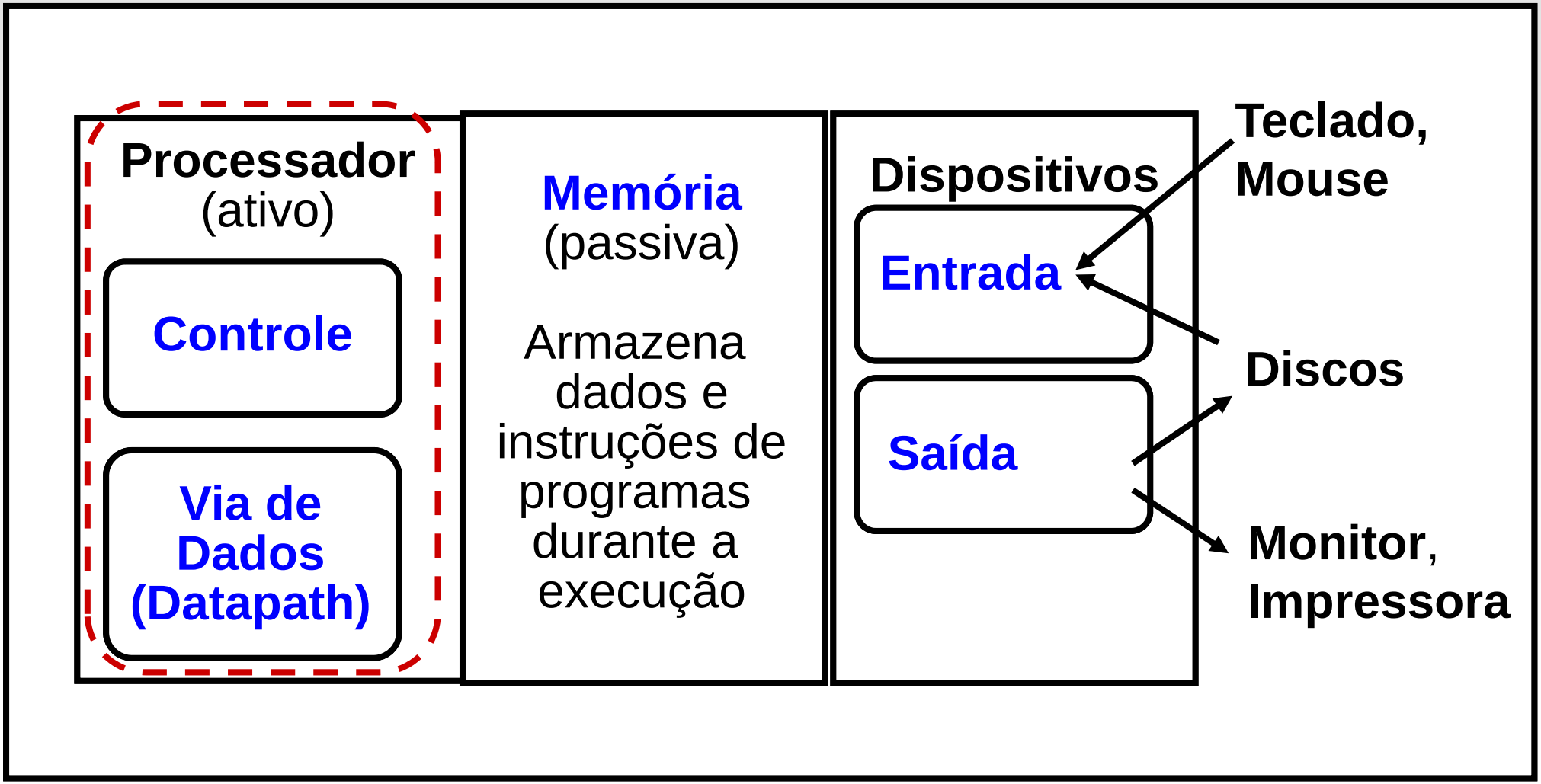
Sistemas Processadores e Periféricos
Aula 6 - Via de Dados e Controle (cap. 5)

Prof. Frank Sill Torres
DELT – Escola de Engenharia
UFMG

Adaptado a partir dos Slides de Organização de Computadores 2006/02 do professor

Leandro Galvão DCC/UFAM - galvao@dcc.ufam.edu.br

Cinco Componentes Clássicos



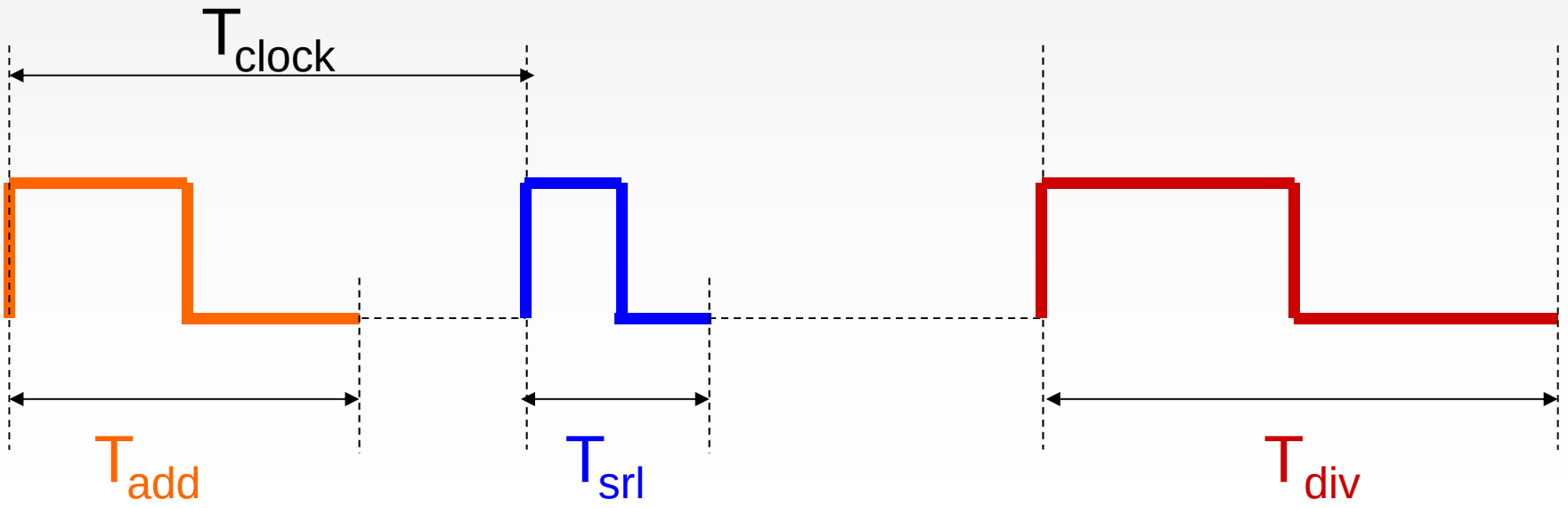
Componentes do Processador

- Via de Dados (datapath)
  - Parte do processador que contem o hardware necessário para execução de todos as operações requeridas pelo computador

- Controle
  - Parte do processador que comanda as ações da via de dados

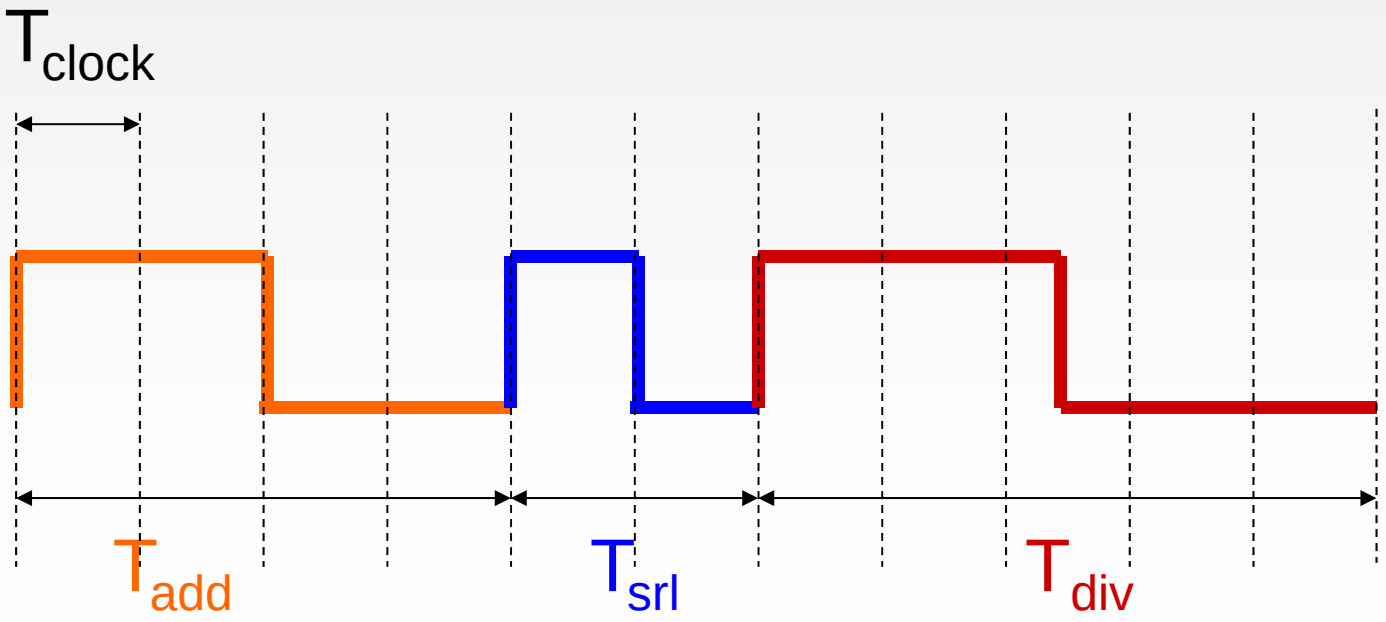
Implementação de Instruções no Processador

- Arquitetura tipo Ciclo Único
  - Cada instrução é executada em um 1 ciclo de clock
  - Ciclo de clock deve ser longo o suficiente para executar a instrução mais longa
  - Desvantagem: velocidade global limitada à velocidade da instrução mais lenta



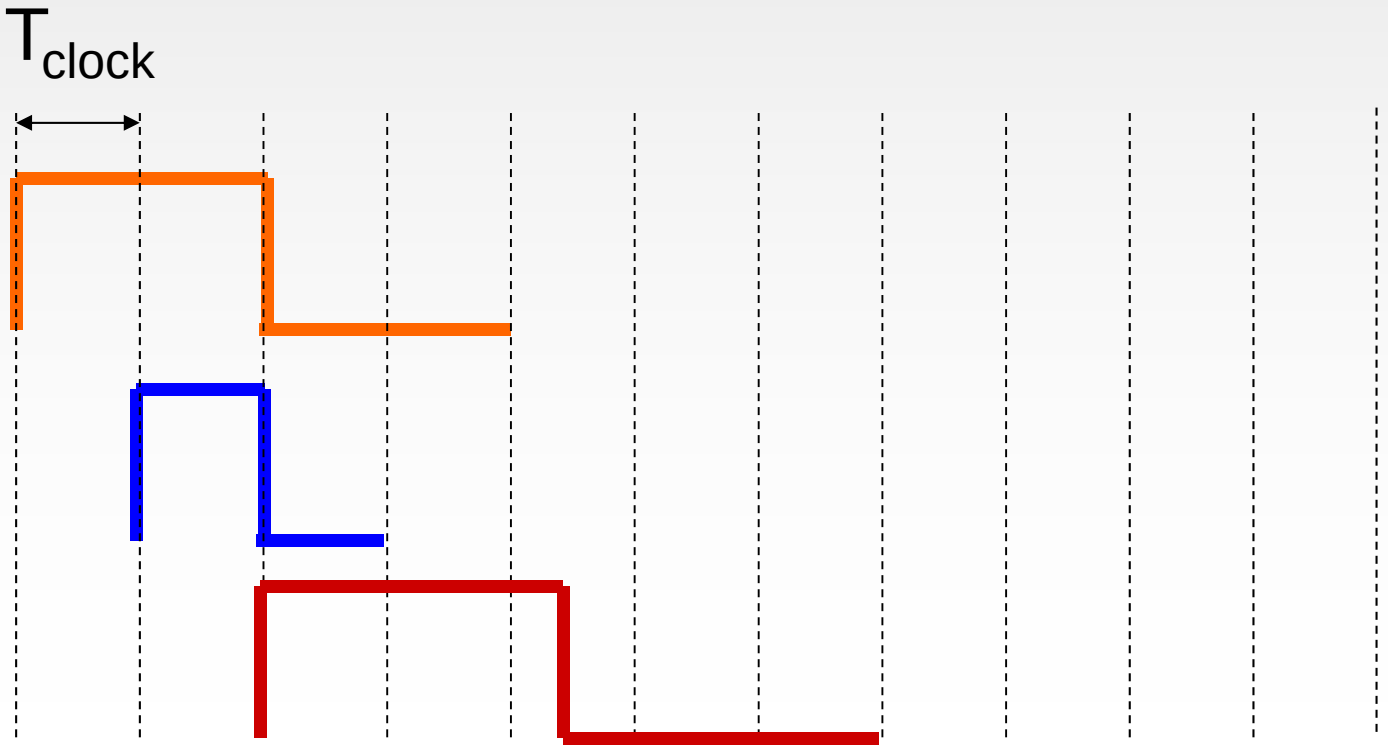
Implementação de Instruções no Processador

- Arquitetura tipo Multi-ciclo
  - Quebra o ciclo de execução em vários passos
  - Executa cada passo em um ciclo de clock
  - Vantagem: cada instrução usa apenas o número de ciclos que ela necessita



Implementação de Instruções no Processador

- Arquitetura tipo Pipelined (linha de montagem)
  - Cada instrução é executada em múltiplos ciclos
  - Executa uma etapa de cada instrução em cada ciclo
  - Processa múltiplas instruções em paralelo



Ciclo único

Construindo a via de dados

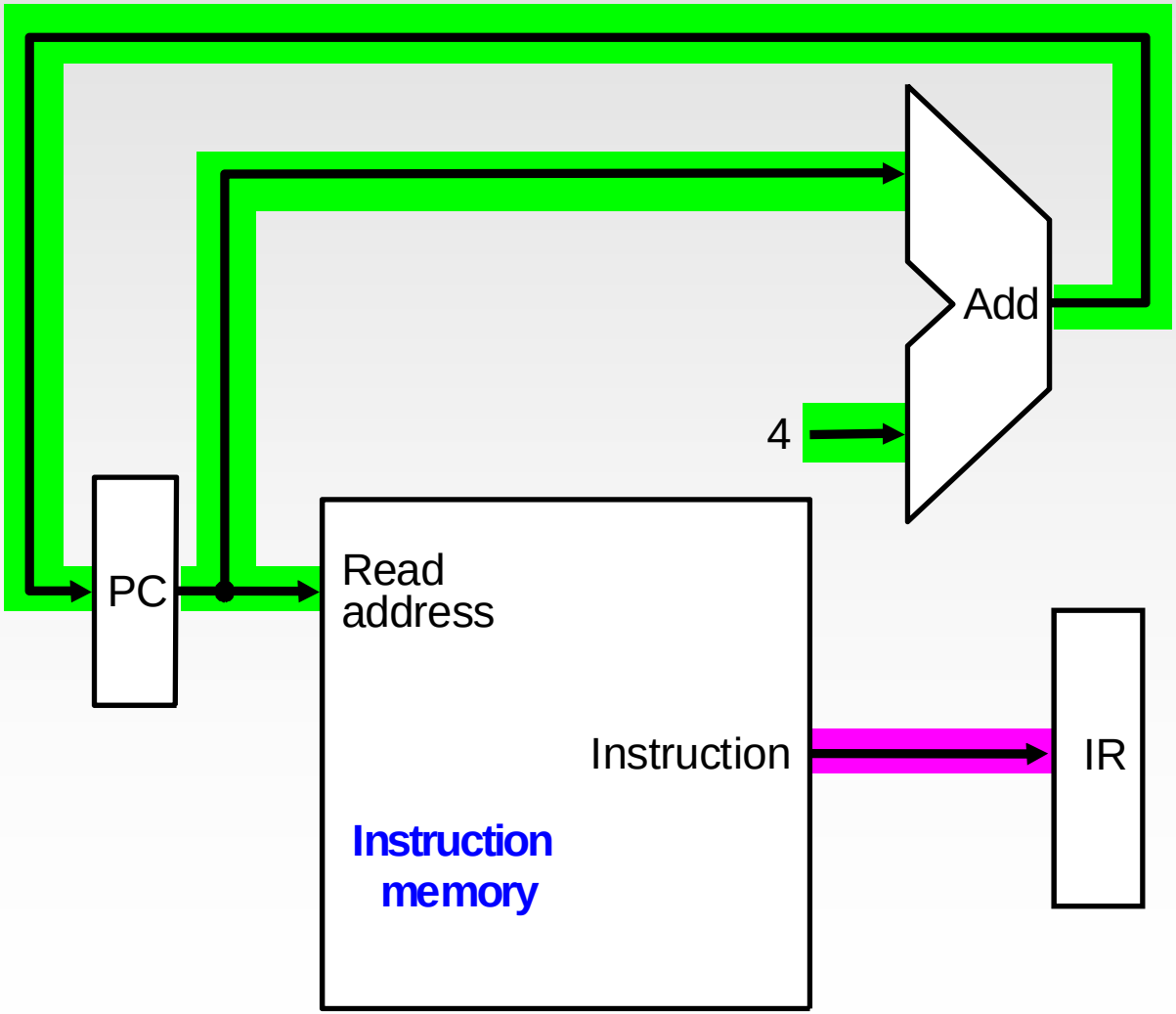
- Para facilitar o entendimento da construção do caminho de dados, vamos seguir a abordagem de ciclo único

- Depois, mostraremos as modificações necessárias para tornar um projeto de ciclo único em um projeto multi-ciclo

Implementação de ciclo único

- Para executar qualquer instrução, precisamos buscá-la na memória
- O registrador Program Counter (PC) é utilizado para ler a instrução da memória e armazená-la no Registrador de Instrução (IR)
- Um somador incrementa PC em 4 (uma word) e coloca o resultado de volta em PC

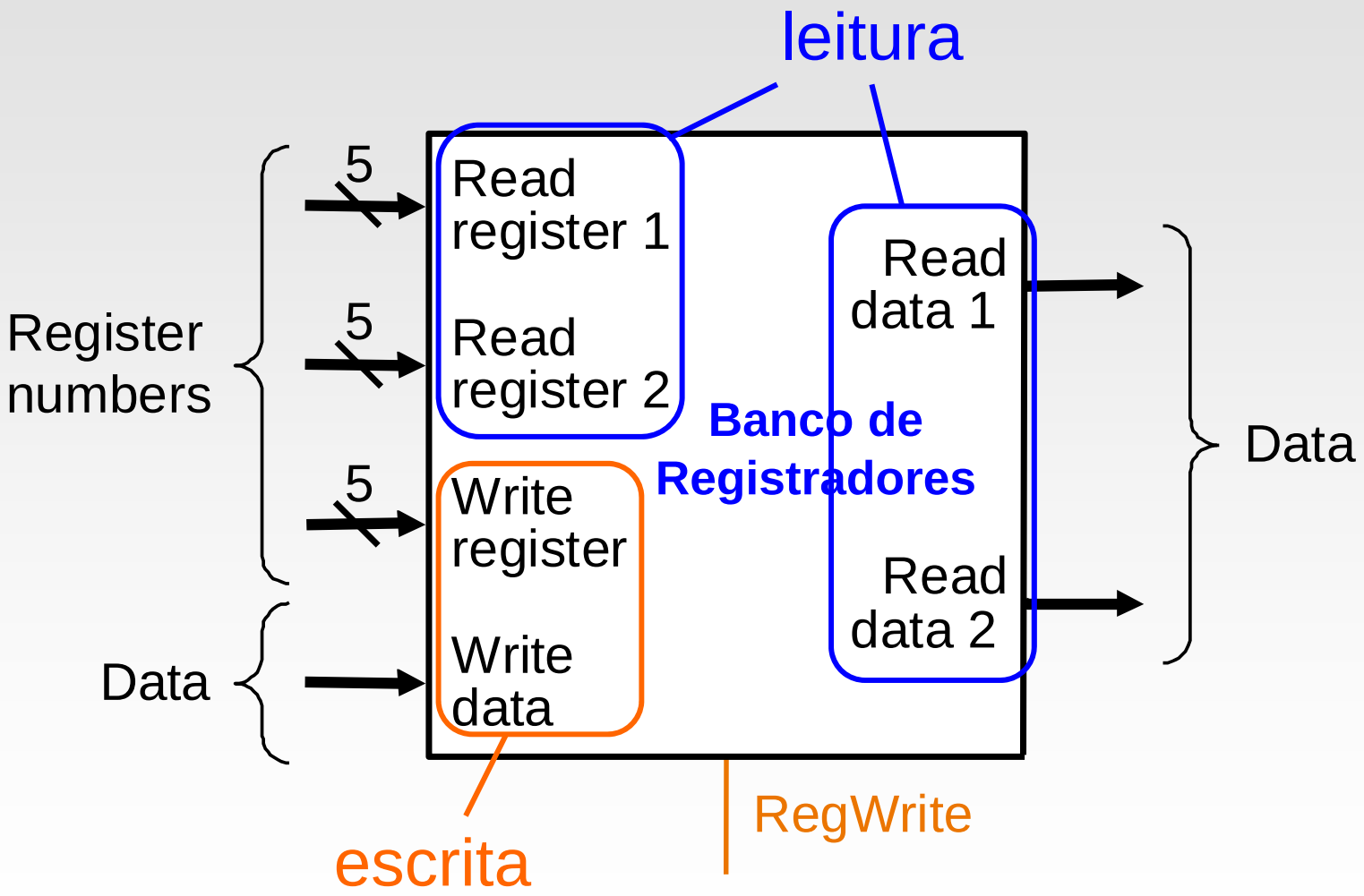
Implementação de ciclo único



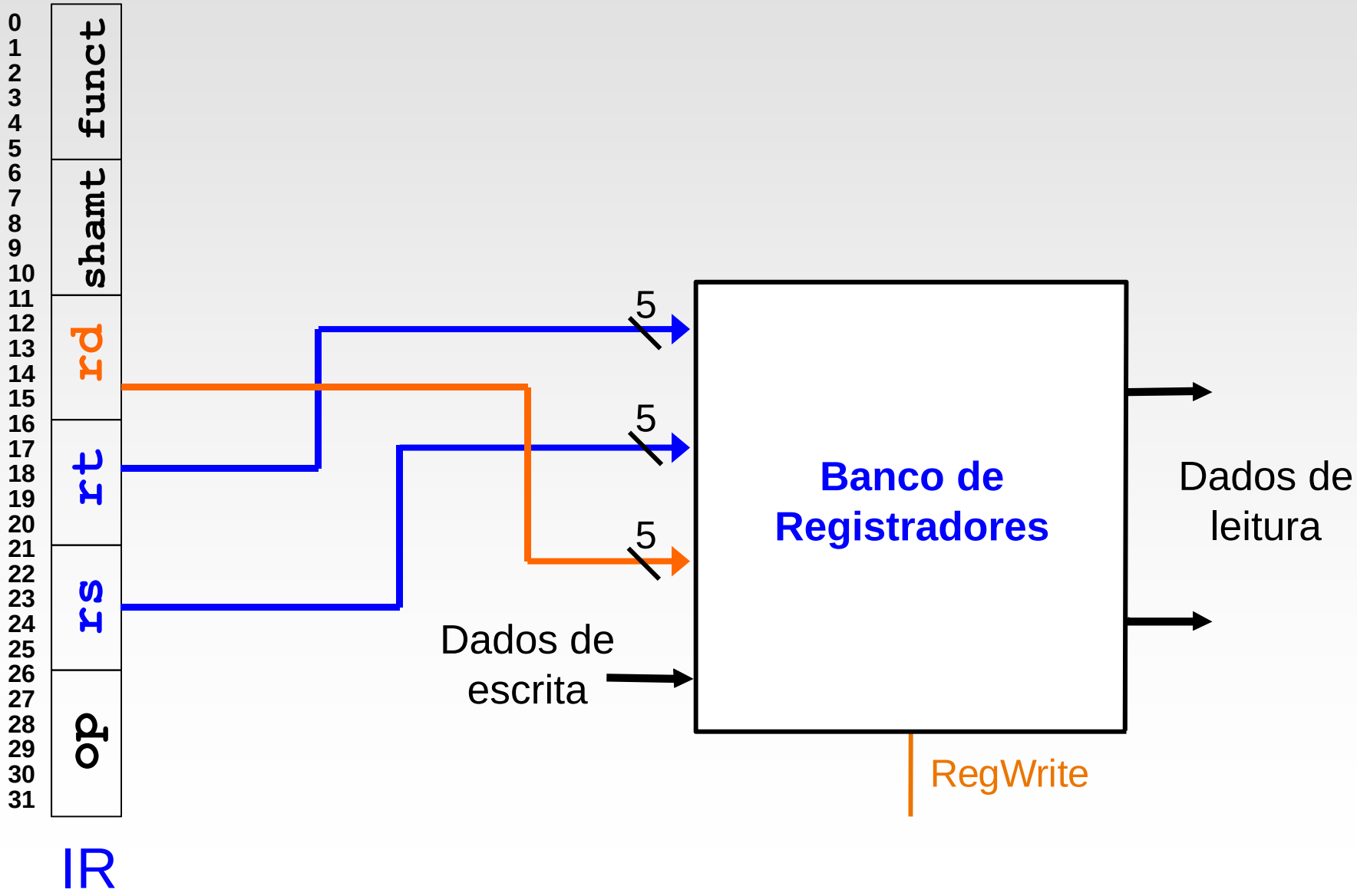
Implementação de ciclo único :: Tipo R

- Instruções do tipo R:
  - Leem dois registradores (**`Rs`**, **`Rt`**)
  - Escrevem em um registrador (**`Rd`**)
- Ponto de vista do Banco de Registradores:
  - Leitura:
    - Números (endereços) dos registradores a serem lidos
    - Saída de dados para os conteúdos lidos
  - Escrita
    - Número (endereço) do registrador a ser escrito
    - Entrada de dados a serem escritos

Implementação de ciclo único
:: Tipo R



Implementação de ciclo único
:: Tipo R



Implementação de ciclo único :: Load / Store

- Calculam um endereço de memória somando:
  - O registrador base (**`Rs`**) de 32 bits
  - Campo de offset de 16 bits

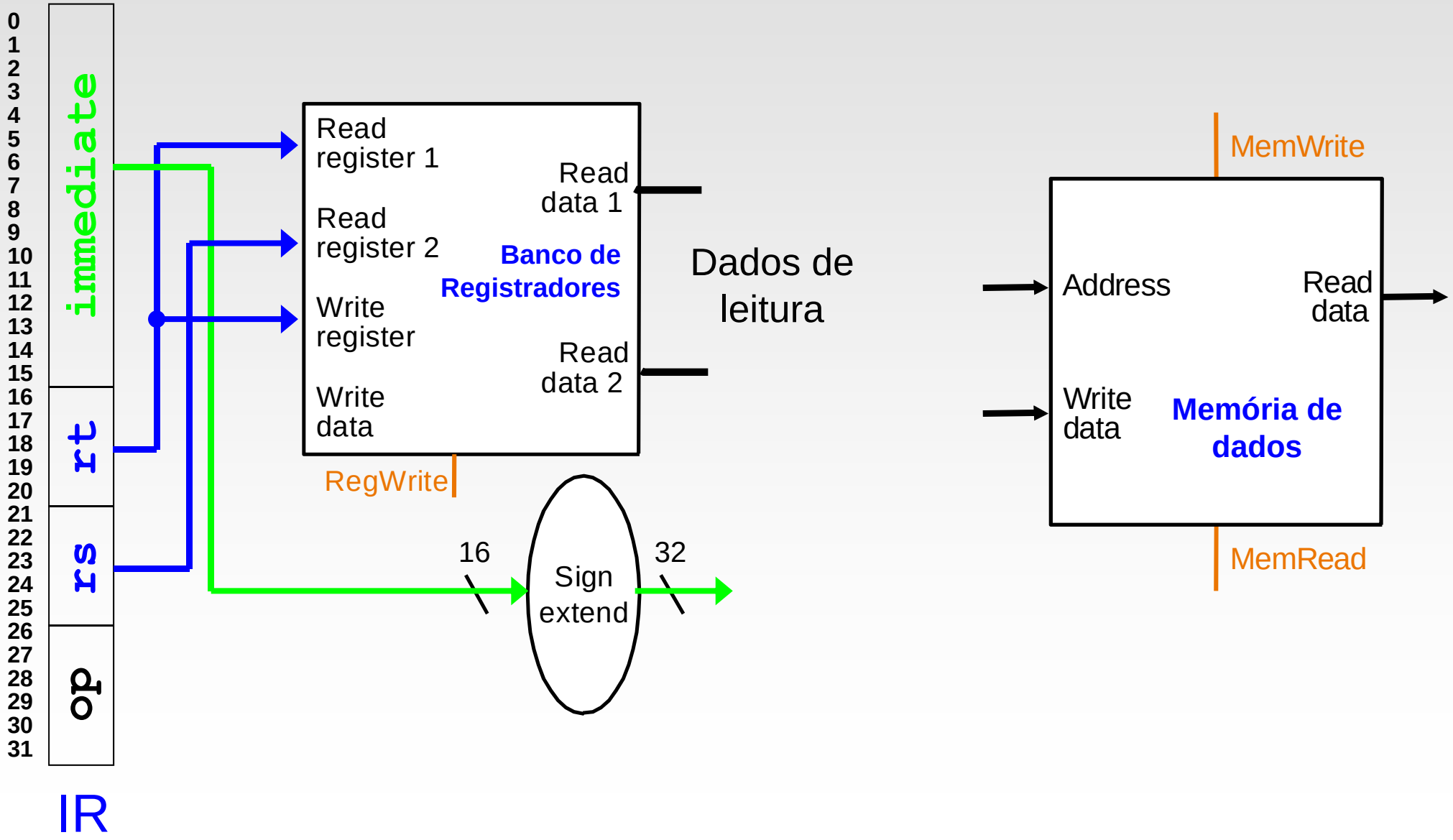
- Precisamos de uma unidade para estender o sinal do campo de offset para um valor de 32 bits

Implementação de ciclo único :: Load / Store

- Manipulam memória de dados:
  - Instruções **`Store`** escrevem dados
  - Instruções **`Load`** leem dados

- Precisamos de uma memória de dados com:
  - Sinal de controle para escrita
  - Sinal de controle para leitura
  - Entrada de endereço para leitura/escrita
  - Entrada de dados para escrita
  - Saída de dados de leitura

Implementação de ciclo único :: Load / Store



Implementação de ciclo único :: Desvios (branch)

- Possuem três operandos:
  - Dois registradores a serem comparados
  - Offset de 16 bits para calcular o endereço de destino

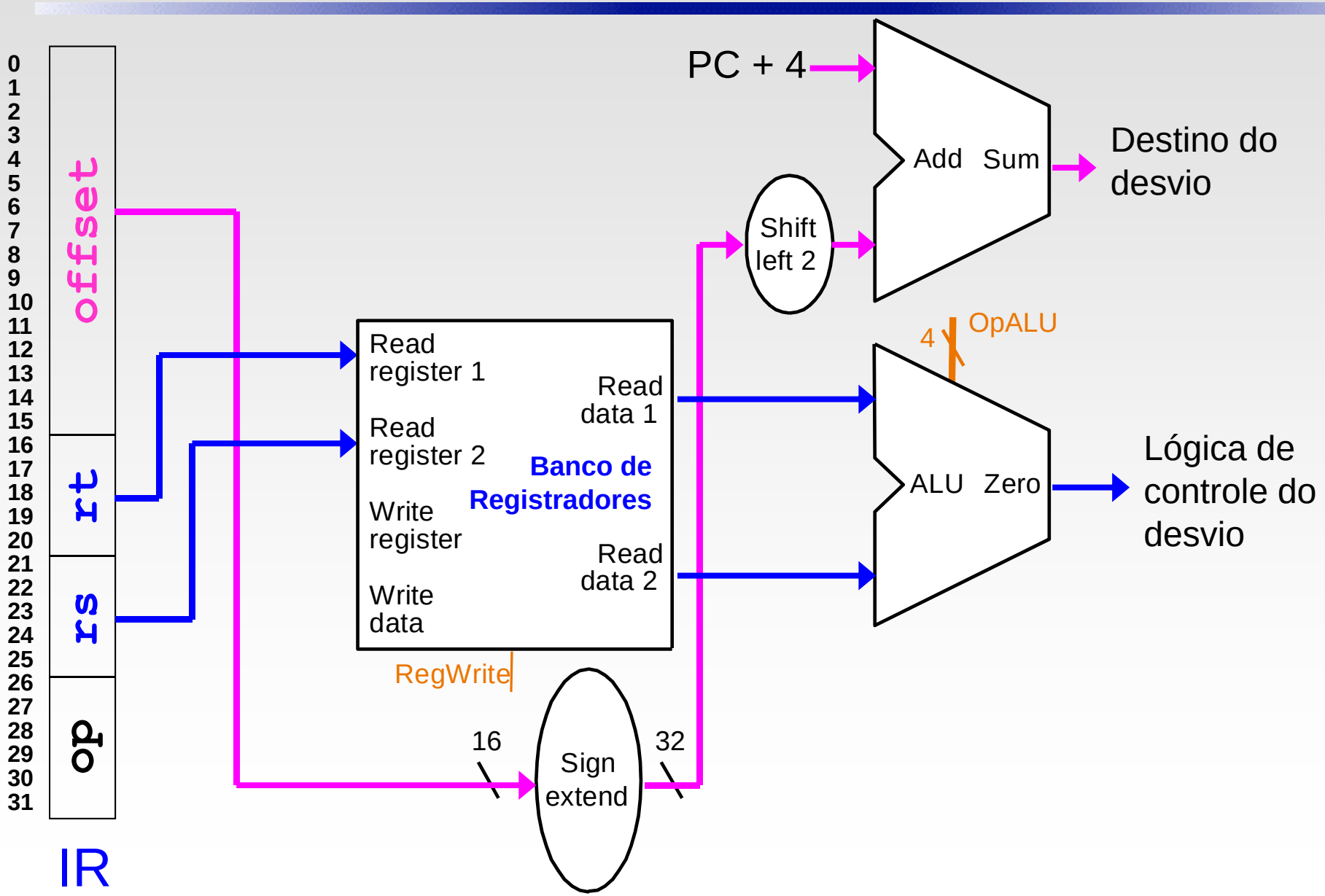
- Dois detalhes:
  - Offset deve ser deslocado 2 bits para a esquerda, para ser expresso em bytes
  - A depender do resultado da comparação, o novo valor de PC pode ser (PC + 4) ou (PC + 4 + (offset << 2))

Implementação de ciclo único :: Desvios (branch)

- Como o deslocamento é constante, não se utiliza um circuito shifter, mas sim um deslocamento de sinais que acrescenta 00 à extremidade direita

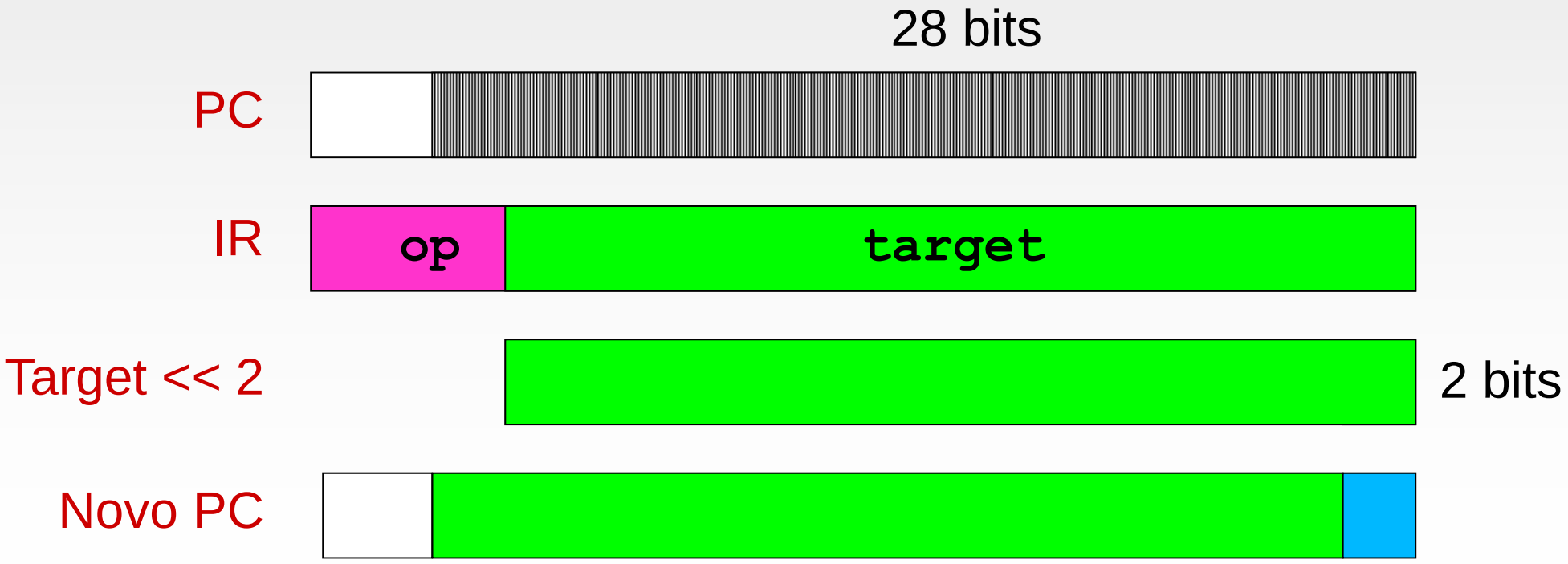
- A comparação é implementada como uma subtração
  - Para operandos iguais, a ALU sinaliza resultado ZERO

Implementação de ciclo único :: Desvios (branch)

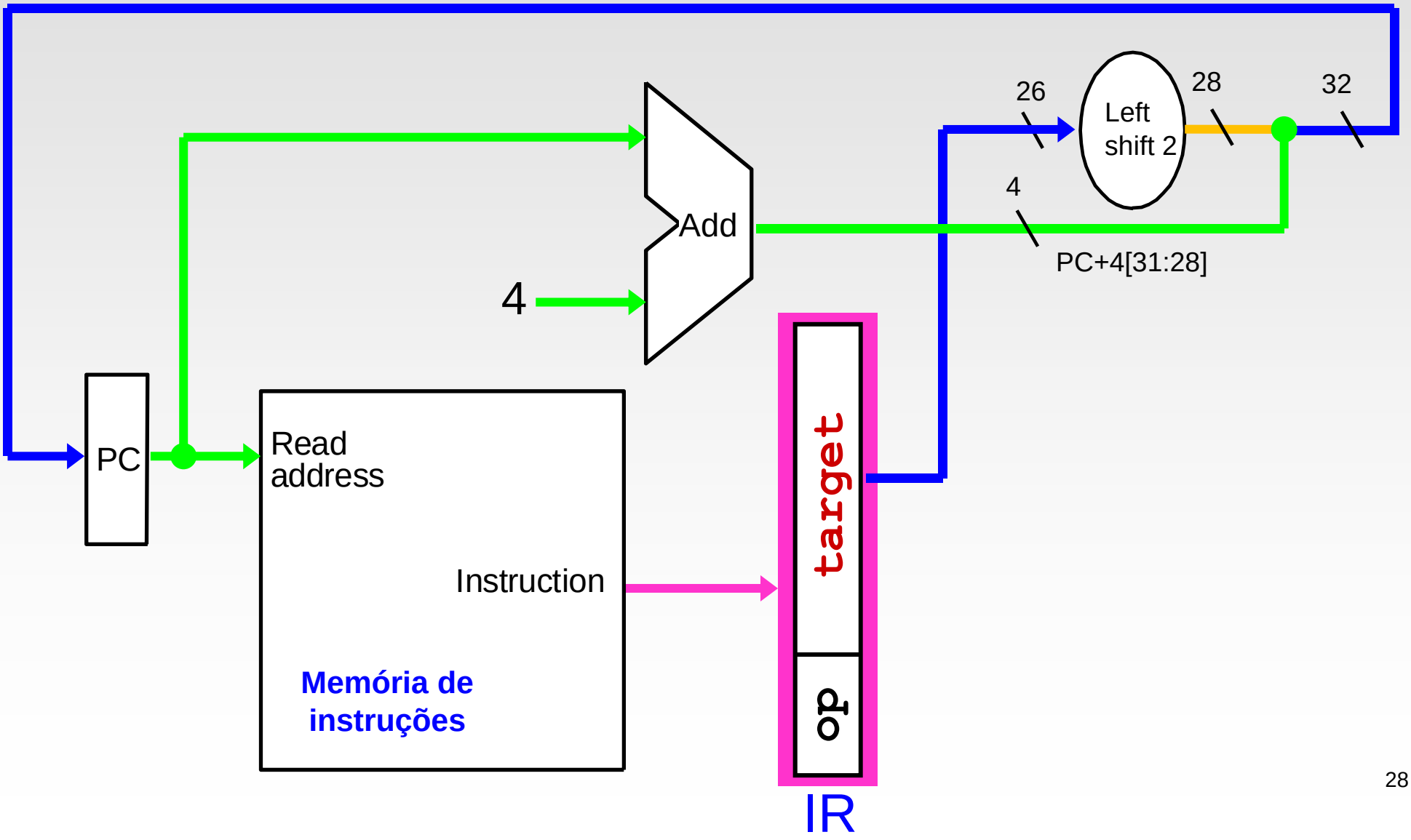


Implementação de ciclo único :: Saltos (jump)

- Substitui os 28 bits menos significativos do PC pelos 26 bits menos significativos do IR deslocados 2 bits à esquerda



Implementação de ciclo único :: Saltos (jump)

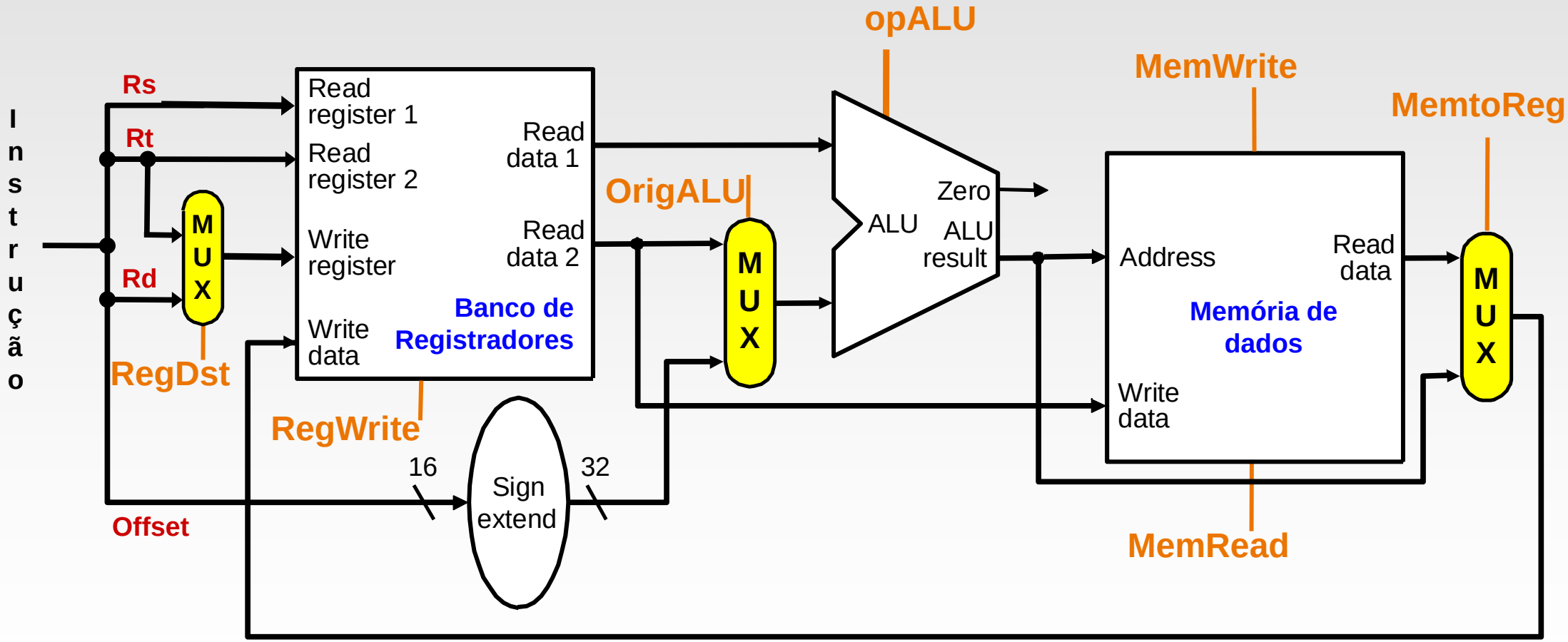


Combinando instruções :: Tipo R + Load/Store

- Para criar um caminho de dados com hardware compartilhado, deve-se suportar 2 origens para:
  - Segunda entrada da ALU:
    - **Tipo R:** registrador
    - **Load/Store:** campo immediate
  - Dados escritos no banco de registradores:
    - **Tipo R:** saída da ALU
    - **Load:** memória de dados
  - Registrador a ser escrito no banco:
    - **Tipo R:** Rd
    - **Load:** Rt

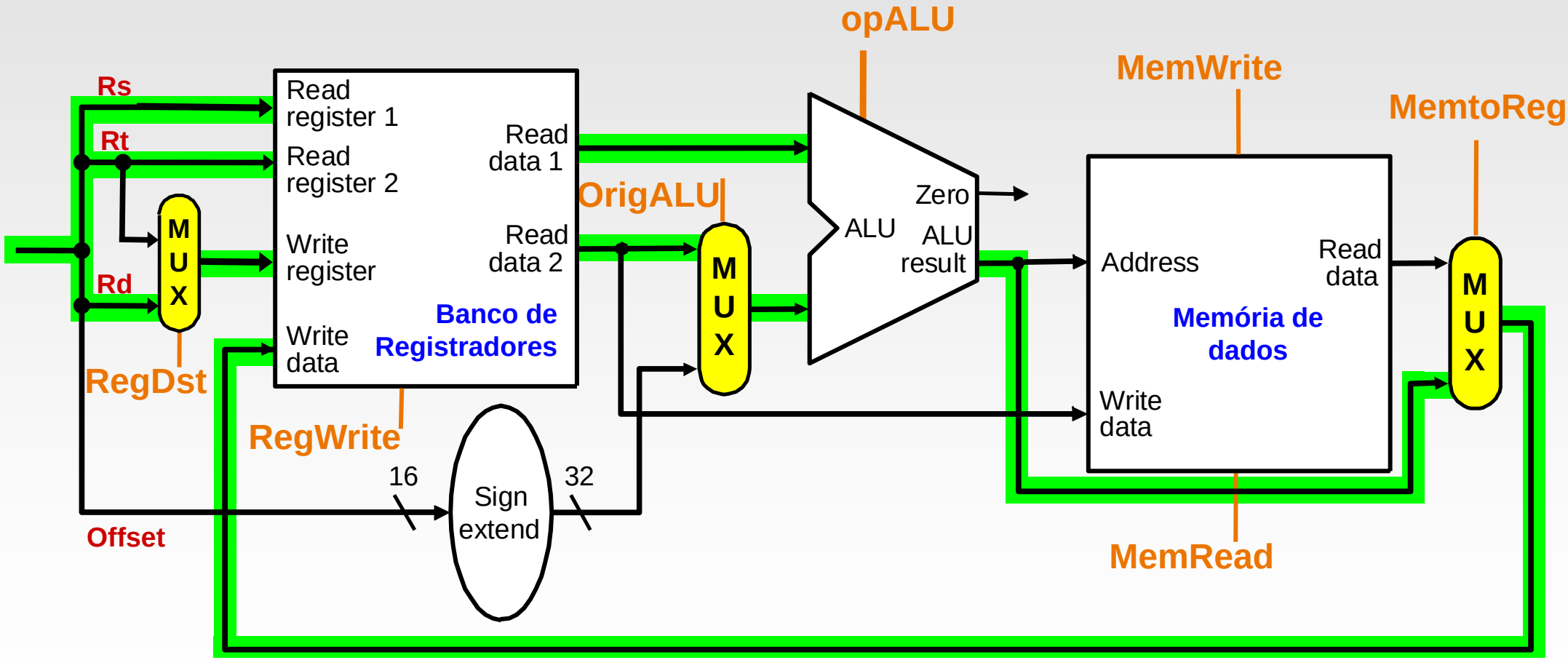
Uso de multiplexadores (MUX)

Combinando instruções :: Tipo R + Load/Store



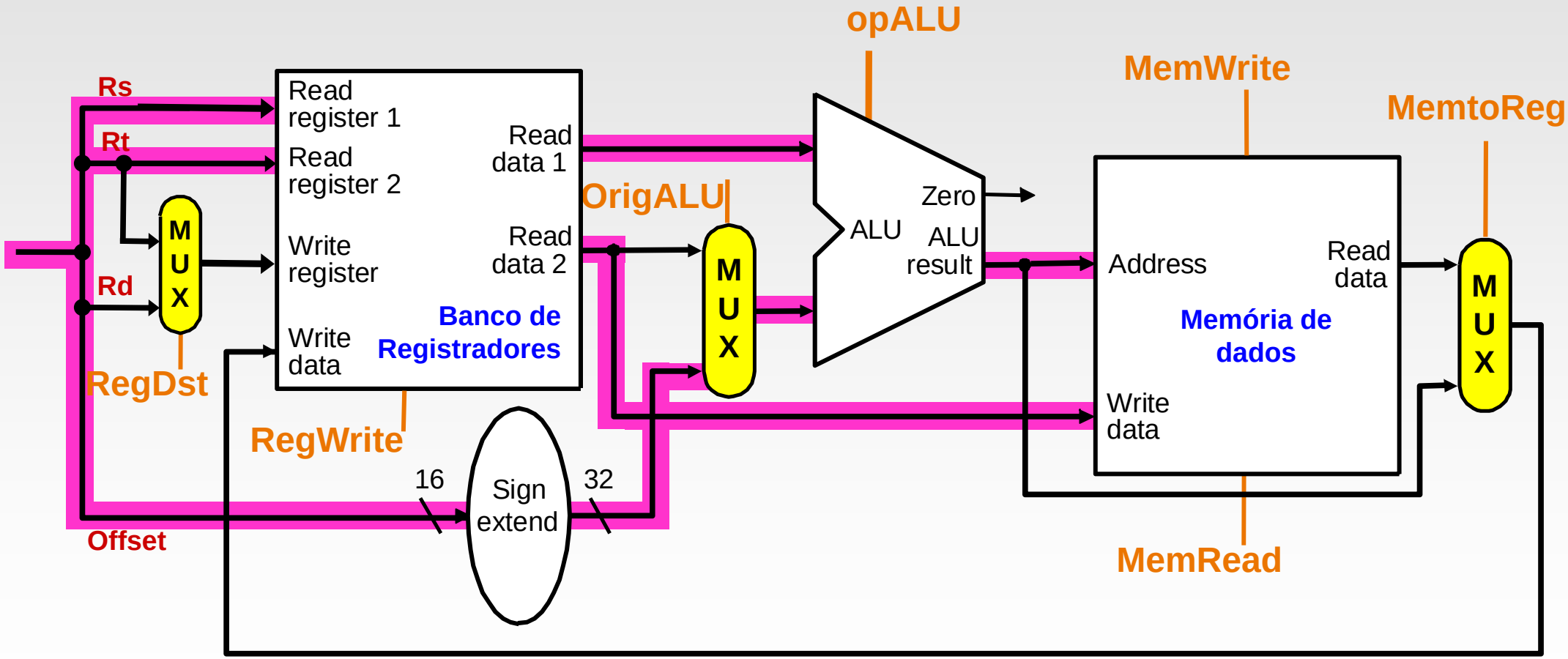
Combinando instruções :: Tipo R + Load/Store

```
add $t0, $s1, $s2
```



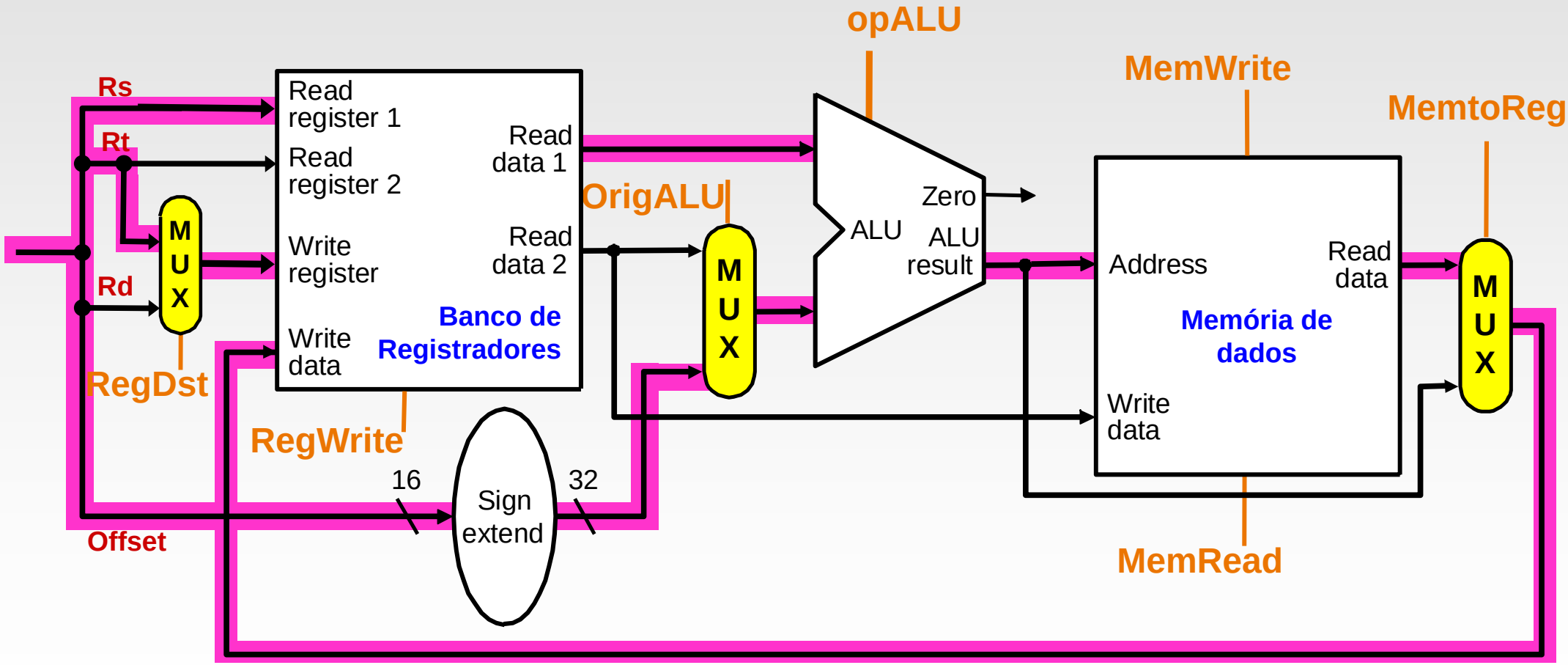
Combinando instruções :: Tipo R + Load/Store

```
sw $t0, 16($s2)
```



Combinando instruções :: Tipo R + Load/Store

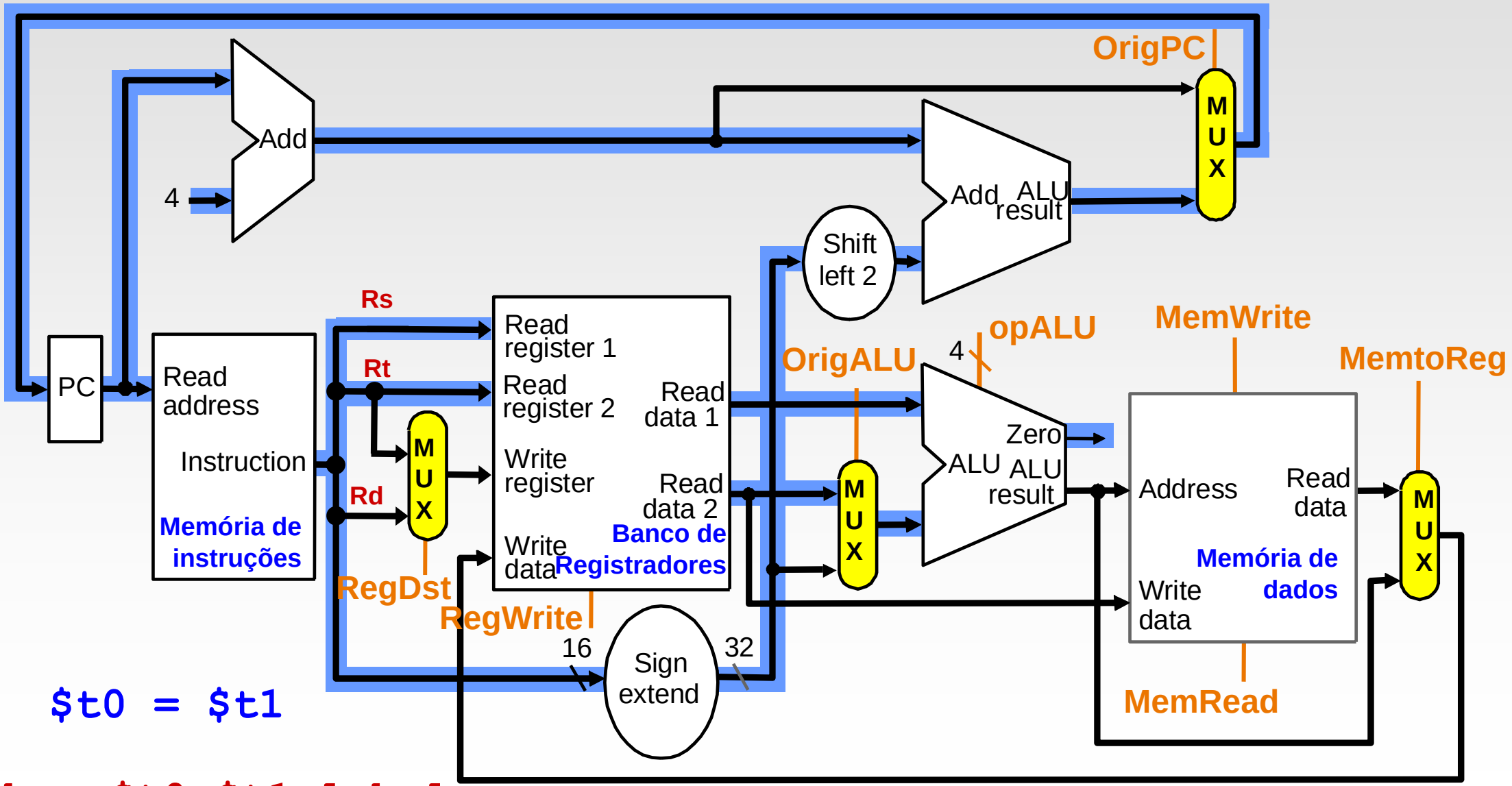
```
lw $t0, 16($s2)
```



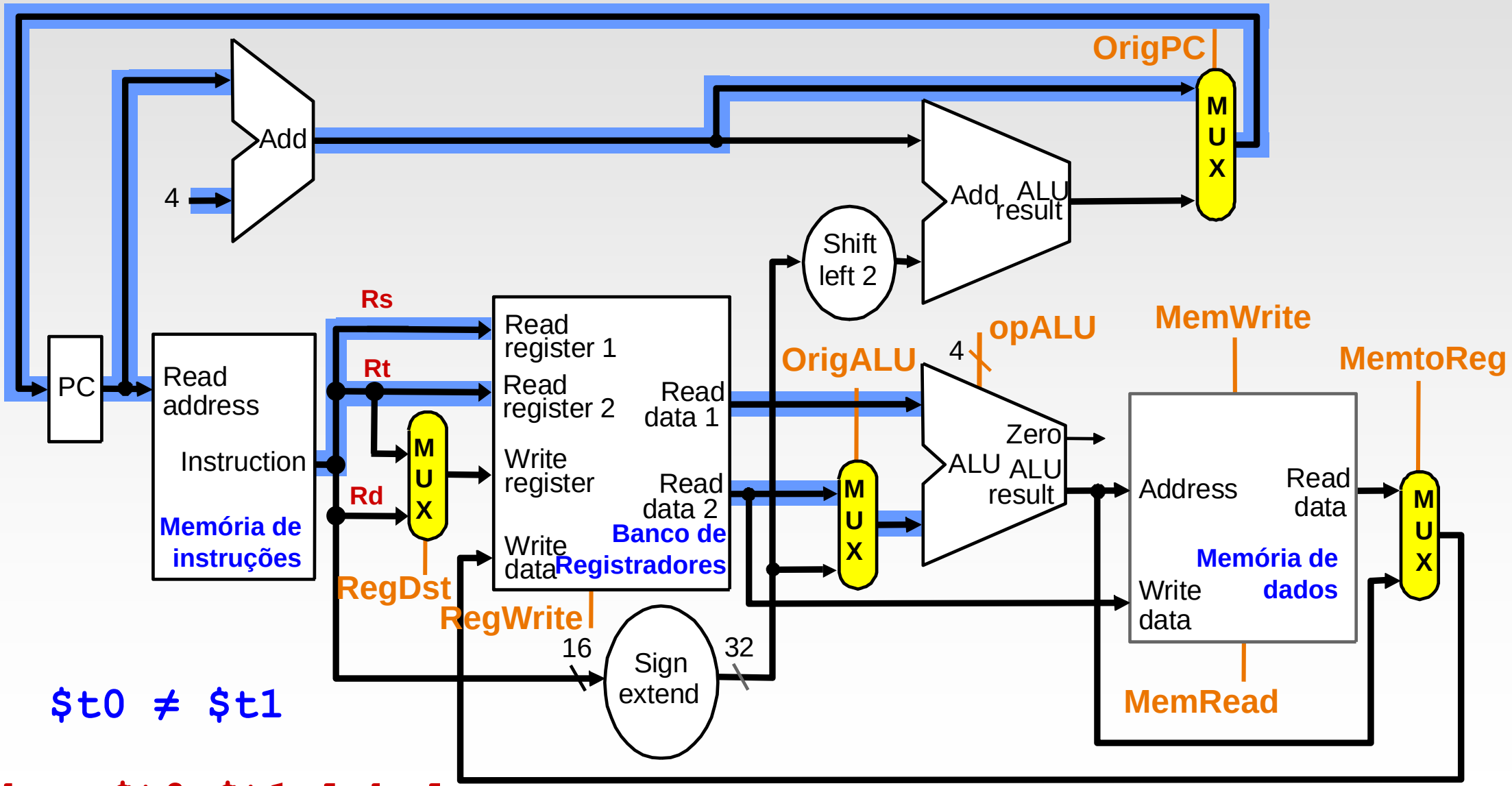
Combinando instruções :: Tipo R + Load/Store + Desvios

- Um multiplexador adicional é necessário para selecionar o valor a ser escrito em PC:
  - Endereço de instrução seguinte (PC + 4), ou
  - Endereço de destino do desvio (PC + 4 + (offset <<2))

Combinando instruções
:: Tipo R + Load/Store + Desvios



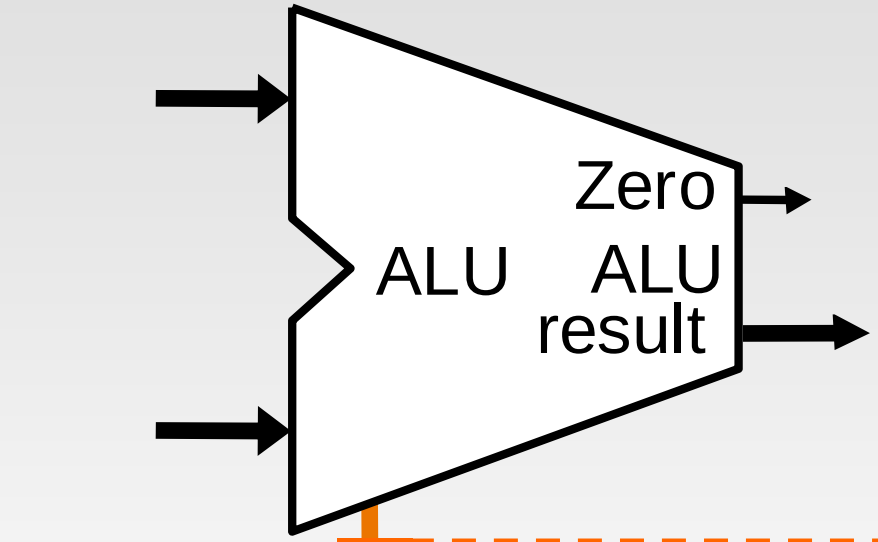
Combinando instruções
:: Tipo R + Load/Store + Desvios



Implementando o Controle :: Controle da ALU

- Instruções do tipo R
  - ALU realiza ação determinada pelo campo **`funct`**
- Load / Store
  - ALU soma conteúdo do registrador base com offset para obter o endereço de memória
- Desvios
  - ALU realiza uma subtração

Implementando o Controle :: Controle da ALU



Instrução	opALU	Entrada de controle na ALU
Load/Store	00	add
Desvio	01	sub
Tipo R	10	funct

Implementando o Controle :: Controle da ALU

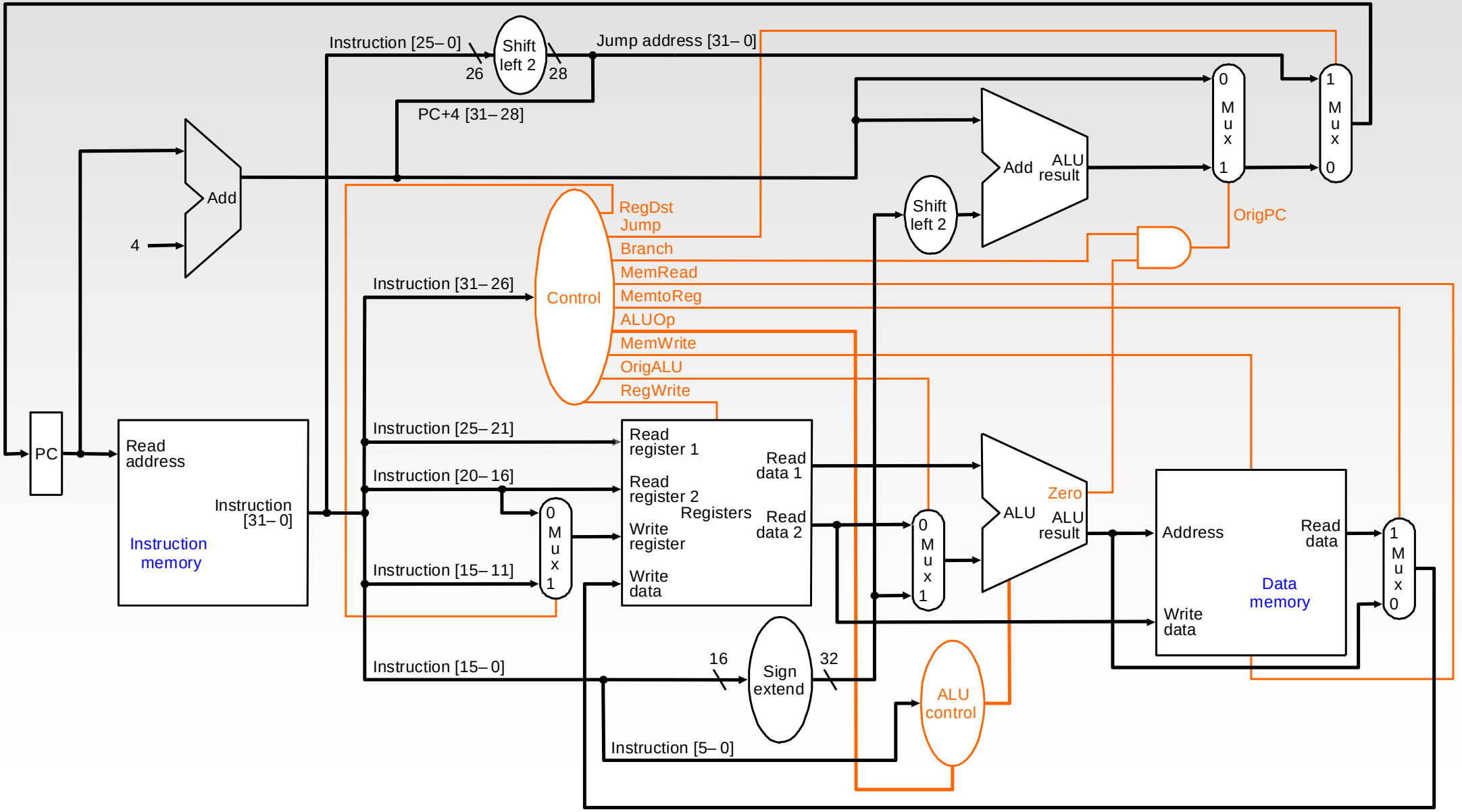
- O circuito lógico da unidade de controle da ALU é obtido a partir da tabela verdade construída para as combinações desejadas do campo funct com os bits de opALU

- A saída da tabela verdade (e do circuito lógico sintetizado a partir dela) será os bits de entrada do controle da ALU, os quais determinarão que operação será realizada (soma, and, shift, etc.)

Implementando o Controle :: Unidade de controle principal

- Deve-se criar uma tabela verdade para as combinações desejadas do campo `op` da instrução gravada em IR, a fim de produzir os correspondente sinais de controle em:
  - **RegDst**
  - **RegWrite**
  - **OrigALU**
  - **Branch**
  - **MemRead**
  - **MemWrite**
  - **MemtoReg**
  - **ALUop**
  - **JUMP**

Implementando o Controle
:: Unidade de controle principal



Implementando o Controle :: Unidade de controle principal

Sinal de Controle	Efeito quando inativo (zero)	Efeito quando ativo (um)
RegDst	O registrador de destino é indicado no campo **Rt** (bits 20:16)	O registrador de destino é indicado no campo **Rd** (bits 15:11)
RegWrite	Nenhum	O registrador destino é escrito com o valor da entrada Write data
OrigALU	O 2º operando da ALU é um registrador (Read data 2)	O 2º operando da ALU é o campo imm estendido para 32 *bits*
Branch	O PC é substituído pelo *output* do somador que calcula **PC+4**	O PC é substituído pelo *output* do somador que calcula o destino do salto
MemRead	Nenhum	O conteúdo da posição de memória endereçada é colocado nas saídas
MemWrite	Nenhum	O conteúdo da posição de memória endereçada é reescrito
MemtoReg	O valor a escrever no registrador é o resultado da ALU	O valor a escrever no registrador é lido da memória de dados

Implementando o Controle :: Unidade de controle principal

- Todos os elementos de estado (registradores, memória) têm o clock como uma entrada implícita

- Unidade de controle pode definir todos os sinais de controle baseada no campo opcode da instrução, exceto Branch, que depende da saída zero da ALU no caso de instruções de desvio

Implementando o Controle
:: Unidade de controle principal

	Opcode	0_h	23_h	$2B_h$	4_h
	Nome	R	Lw	Sw	Beq
Entrada opcode na unidade de controle	Bit31	0	1	1	0
	Bit30	0	0	0	0
	Bit29	0	0	1	0
	Bit28	0	0	0	1
	Bit27	0	1	1	0
	Bit26	0	1	1	0
Saídas da unidade de controle	RegDst	0	1	X	X
	OrigALU	1	0	0	1
	MemtoReg	0	1	X	X
	RegWrite	1	1	0	0
	MemRead	0	1	0	0
	MemWrite	0	0	1	0
	Branch	0	0	0	1
	ALUop1	1	0	0	0
	ALUop0	0	0	0	1

Desempenho Ciclo Único

- Ciclo de clock tem mesma duração para todas instruções: ciclos de clocks por instrução (CPI) = 1
- Ciclo de clock é longo suficiente para executar a instrução mais demorada, que neste caso é **`lw`**
- Apesar do CPI ser 1, o período de clock (T_{clock}) é elevado o que prejudica o desempenho do CPU

$$T_{exec} = N_{instr} \times CPI \times T_{clock}$$

Exemplo: Desempenho Ciclo Único

- Considere uma máquina com uma unidade de ponto flutuante adicional
- Assuma que os atrasos sejam os seguintes:

Componente	Atraso (ns)
Memória	2
ALU	2
Unidade de PF (+ / -)	8
Unidade de PF (× / ÷)	16
Acesso a registradores	1
MUX, controle, acesso ao PC, extensão de sinal e ligações	0

Exemplo: Desempenho Ciclo Único

- Assuma o seguinte mix de instruções:

Instrução	%
Load	31
Store	21
Tipo R	25
Branch	5
Jump	2
Operações PF (+ / -)	7
Operações PF (× / ÷)	7

Exemplo: Desempenho Ciclo Único

- Compare o desempenho de
  a) Uma implementação de ciclo único usando clock de período fixo
  b) Uma implementação de clock com período “variável”
     - Cada instrução executa em um ciclo de clock
     - O ciclo é tão longo quanto seja necessário para a executar a instrução
     - Na prática, só possível em sistemas assíncronas (que não têm um sinal de clock)!

Exemplo: Desempenho Ciclo Único

Tipo de Instrução	Mem. Instr.	Leitura Regist.	oper. ALU	Mem. Dados	Escr . Reg.	PF add/ sub	PF mul/ div	Tempo Total ns.
Load word	2	1	2	2	1			8
Store word	2	1	2	2				7
R-format	2	1	2		1			6
Branch	2	1	2					5
Jump	2							2
FP mul/div	2	1			1		16	20
FP add/sub	2	1			1	8		12

Exemplo: Desempenho Ciclo Único

- Período de clock considerando período fixo = tempo da instrução mais longa = 20 ns.

- Tempo médio de clock considerando período variável =
  - $8 \times 31\ \% + 7 \times 21\ \% + 6 \times 27\ \% + 5 \times 5\ \% + 2 \times 2\ \% + 20 \times 7\ \% + 12 \times 7\ \% = 7.0$ ns

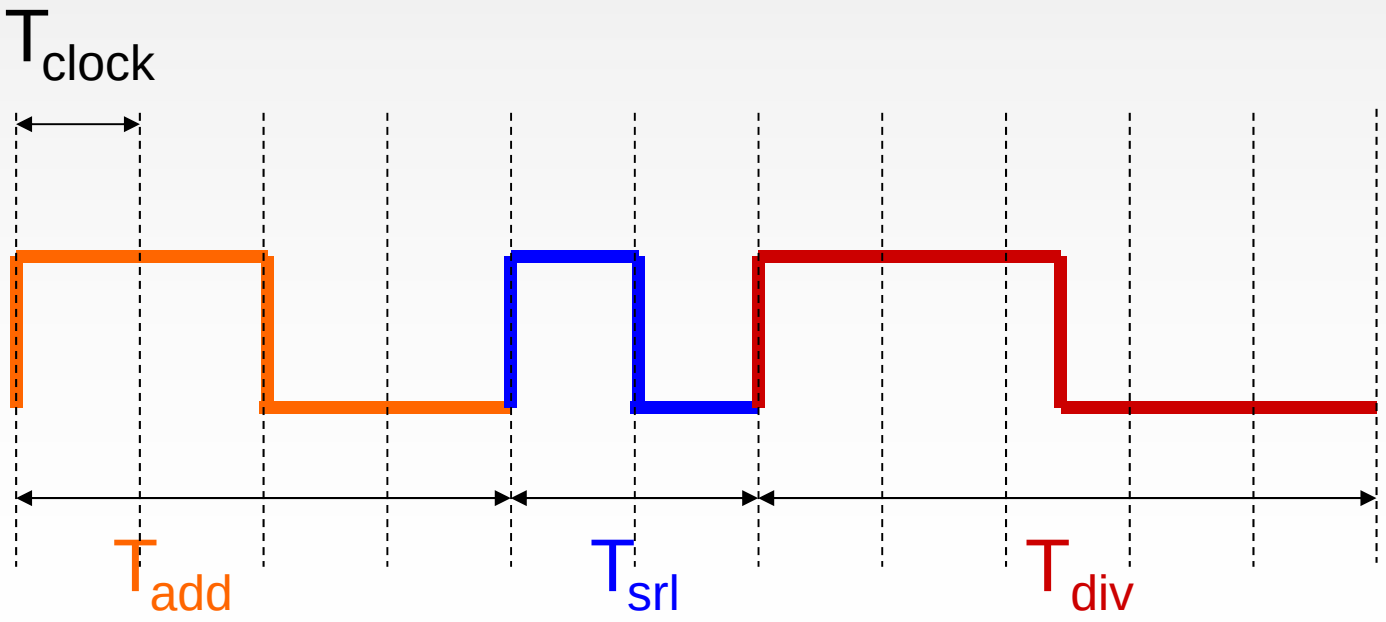
- Portanto:

$$\frac{Desempenho_{\text{ciclo único}}}{Desempenho_{\text{ciclo variável}}} = \frac{20}{7} \cong 2{,}9$$

Multi-ciclo

Revisão do slide anterior

- Multi-ciclo
  - Quebra o ciclo de execução em vários passos
  - Executa cada passo em um ciclo de clock
  - Vantagem: cada instrução usa apenas o número de ciclos que ela necessita



Dividindo Instruções em Estágios

- Possíveis estágios:
  - Instruction Fetch (IF) – Carga de instrução e incremento do registrador PC
  - Instruction Decode (ID) – Decodificação e carga de registrador(es) do banco
  - Execução (EX) – operação da ALU, cômputo de endereço de memória ou finalização de desvios
  - MEM – Acesso à memória ou finalização de instrução R
  - Write Back (WB) – Finalização de leitura de memória

Dividindo Instruções em Estágios

- Cada estágio toma um ciclo de clock
- Nem todas as instruções usam todos os estágios
- Em MIPS, as instruções tomam entre 3 – 5 ciclos (estágios)

Dividindo Instruções em Estágios

- Implementação multi-ciclo permite que uma unidade funcional seja usada mais de uma vez por instrução, desde que seja usada em diferentes ciclos de clock

- O compartilhamento de hardware e a redução do tempo de execução constituem as principais vantagens de um projeto multi-ciclo

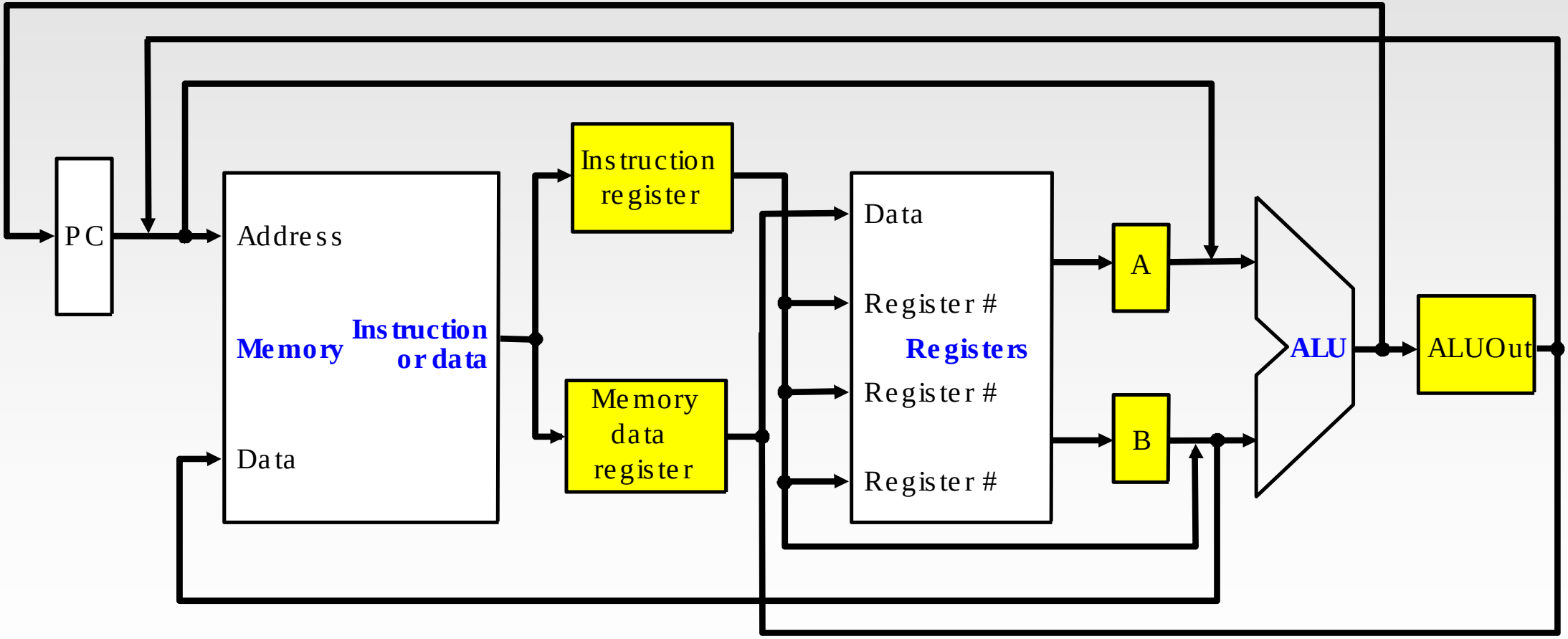
Dividindo Instruções em Estágios

- Registradores são adicionados após cada unidade funcional para conter a saída dessa unidade até o valor ser utilizado em um ciclo de clock subsequente
- Esses registradores são “invisíveis” ao programador
- Sua função é evitar perda de sincronização nas transições de clock

Dividindo Instruções em Estágios

- IR (registrador de instrução) – usado para guardar a saída da memória para uma leitura de instrução
- MDR (registrador de dados da memória) – usado para guardar a saída da memória para uma leitura de dados
- A e B – usados para conter os valores dos registradores operandos lidos do banco de registradores
- ALUOut – contém a saída da ALU

Dividindo Instruções em Estágios



Dividindo Instruções em Estágios :: Inclusão de MUX

- Como várias unidades funcionais são compartilhadas para diferentes finalidades, precisamos de:
  - Incluir multiplexadores
  - Expandir os multiplexadores existentes
- Como MUX e registradores são muito pequenos se comparados a uma unidade de memória ou ALU

  → Portanto, a economia na eliminação destes compensa os custos de adição daqueles

Dividindo Instruções em Estágios :: Inclusão de MUX

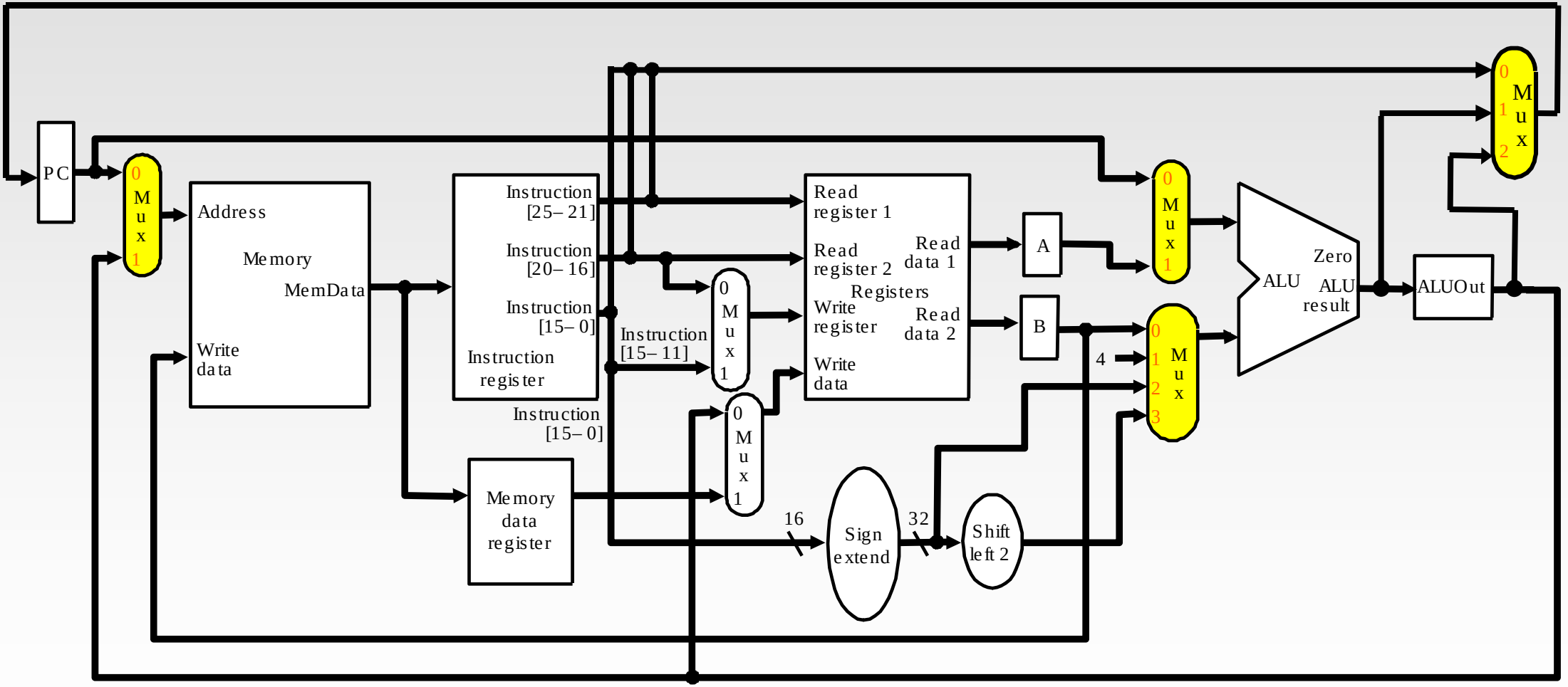
- Memória única:
  - Requer um MUX para selecionar se o endereço de acesso à memória vem de PC (instrução) ou de SaídaALU (dados)

- ALU única:
  - Um MUX adicional é incluído na primeira entrada para escolher entre o registrador A ou o PC
  - O MUX da segunda entrada da ALU é expandido para quatro entradas, a fim de poder selecionar a constante 4 (incremento do PC) e o campo offset estendido e deslocado (desvios)

Dividindo Instruções em Estágios :: Inclusão de MUX

- Três origens para o valor de PC:
  - Saída da ALU (PC + 4) (Entrada 1)
  - Registrador ALUOut, onde é armazenado o endereço de desvio, após ele ser calculado (Entrada 2)
  - 26 bits menos significativos do IR deslocados de 2 à esquerda e concatenados com os 4 bits mais significativos de PC + 4, no caso de jumps (Entrada 0)

Dividindo Instruções em Estágios :: Inclusão de MUX



O que vocês aprenderam hoje?

- Via de dados e Unidade de controle da arquitetura tipo ciclo único
  - Inclusão de multiplexadores
  - Linhas de controle
  - Implementação do controle
- Desempenho da arquitetura tipo ciclo único
- Via de dados de arquitetura tipo multi-ciclo

Questão

- Defina os estados dos sinais de controle para a arquitetura ciclo único para a instrução *add $s1, $s2, $s3*

Nome do sinal	add $s1, $s2, $s3
RegDst	1
OrigALU	0
MemtoReg	0
RegWrite	1
MemRead	0
MemWrite	0
Branch	0
ALUop1	1
ALUop0	0
Jump	0