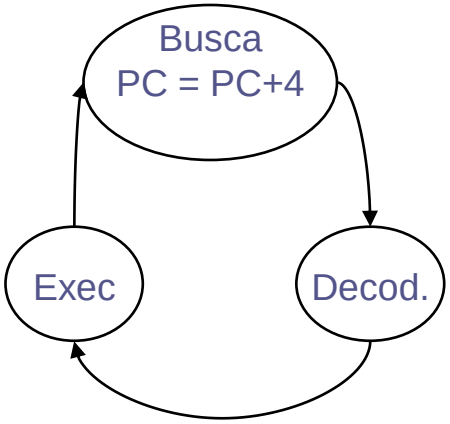
Arquitetura de Computadores

Arquitetura MIPS – caminho de dados

Prof. Tiago Gonçalves Botelho

Processador: Caminho de dados e Controle

- Considerando uma implementação simplificada do MIPS com as instruções...
  - de referência à memória: **`lw, sw`**
  - lógica-aritméticas: **`add, sub, and, or, slt`**
  - de controle: **`beq, j`**
- Implementação genérica
  - (PC) disponibiliza o endereço da instrução a ser buscada na memória. PC é atualizado.
  - Decodificação de instrução (e leitura de registradores)
  - Executa a instrução

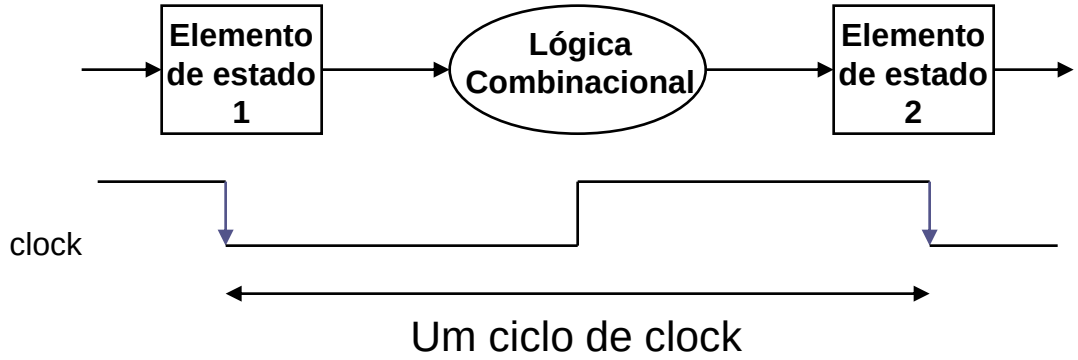


- Todas instruções (exceto **`j`**) usam a ALU após a leitura dos registradores. Como?

Referência à memória? Aritmética? Controle de fluxo?

Metodologias de Clock

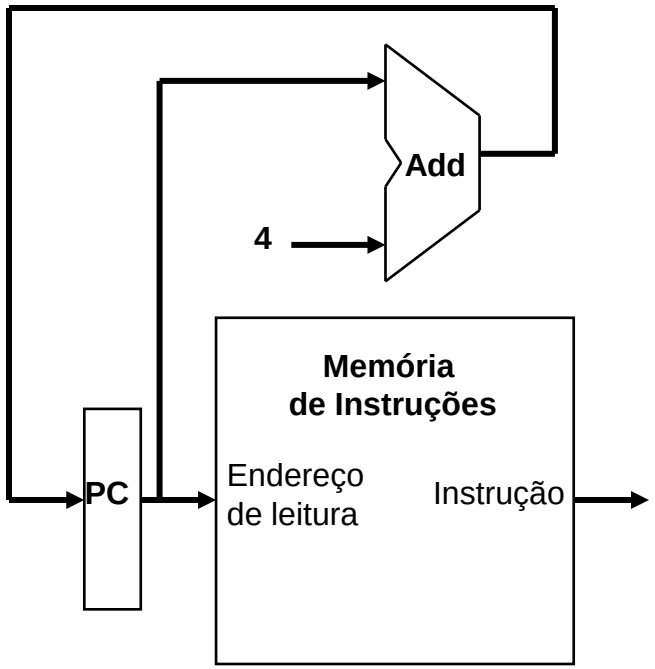
- As metodologias de clock definem quando os sinais podem ser lidos e quando podem ser escritos
  - Metodologia de mudança de estado
- Execução típica
  - lê conteúdos de elementos de estado
  - envia valores pela lógica combinacional
  - escreve resultados em um ou mais elementos de estado



- Assume-se que elementos de estado são escritos em todo ciclo de clock; caso contrário, é necessário um sinal de controle para escrita explícito
  - A escrita ocorre quando ambos, sinal de controle é setado e a mudança de estado de clock acontece

Buscando instruções

- A busca de instruções envolve
  - ler a instrução da memória de instruções
  - atualizar o PC para guardar o endereço da próxima instrução

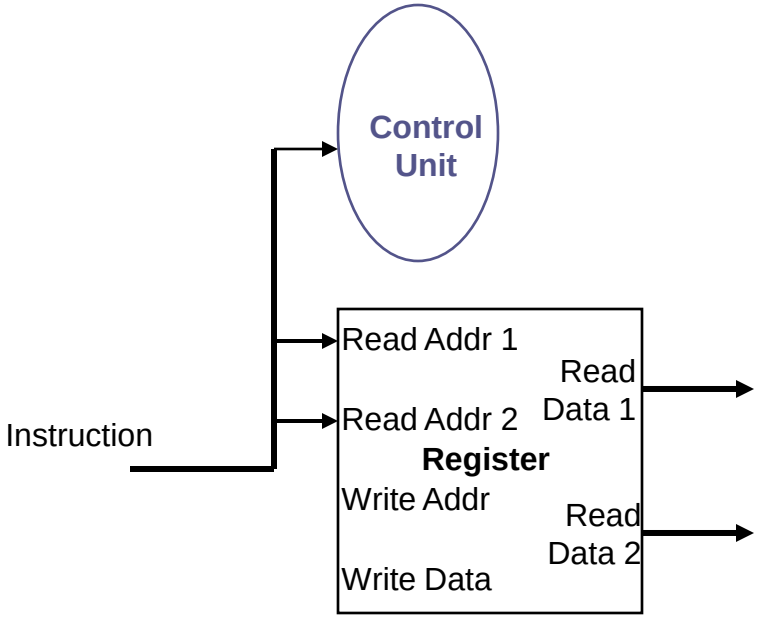


- PC é atualizado em todo ciclo. Nesse caso, não é necessário um controle explícito de escrita
- A memória de instruções é lida em todo ciclo. Da mesma forma, não é necessário um controle explícito de leitura

Decodificando instruções

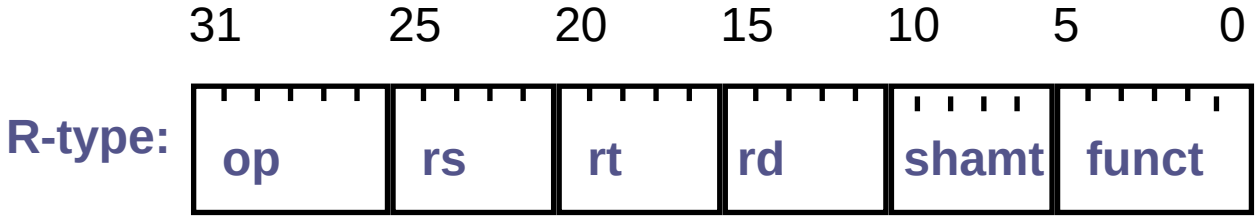
- Decodificação de instruções envolve
  - enviar os campos opcode e function para a unidade de controle

- "lendo" dois valores do banco de registradores
  - Os endereços dos registradores estão contidos na instrução



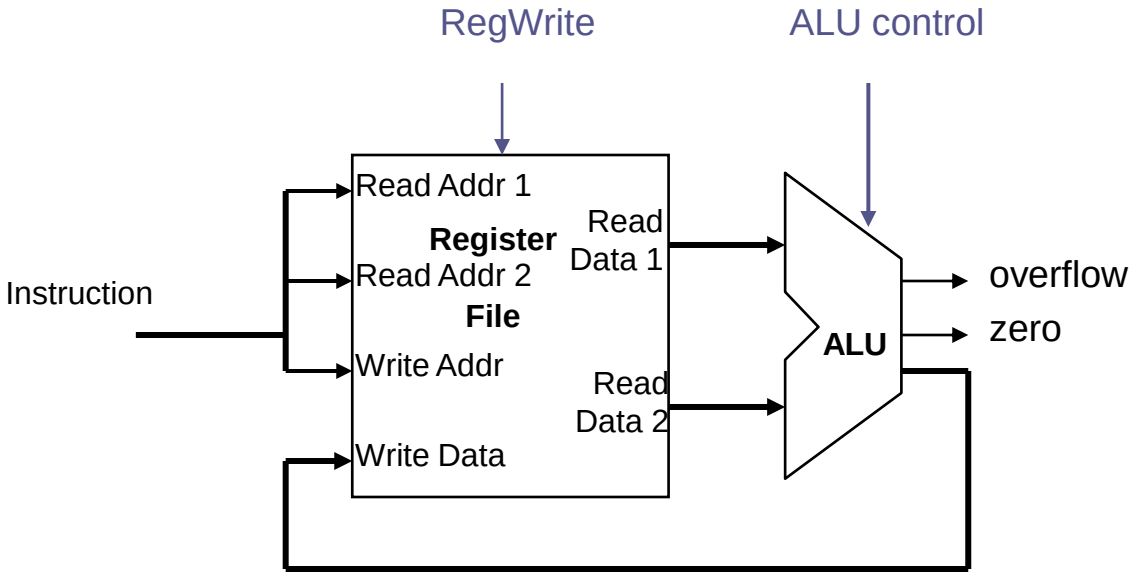
Execução de instruções do tipo R

- Instruções do tipo R (`add, sub, and, or`)



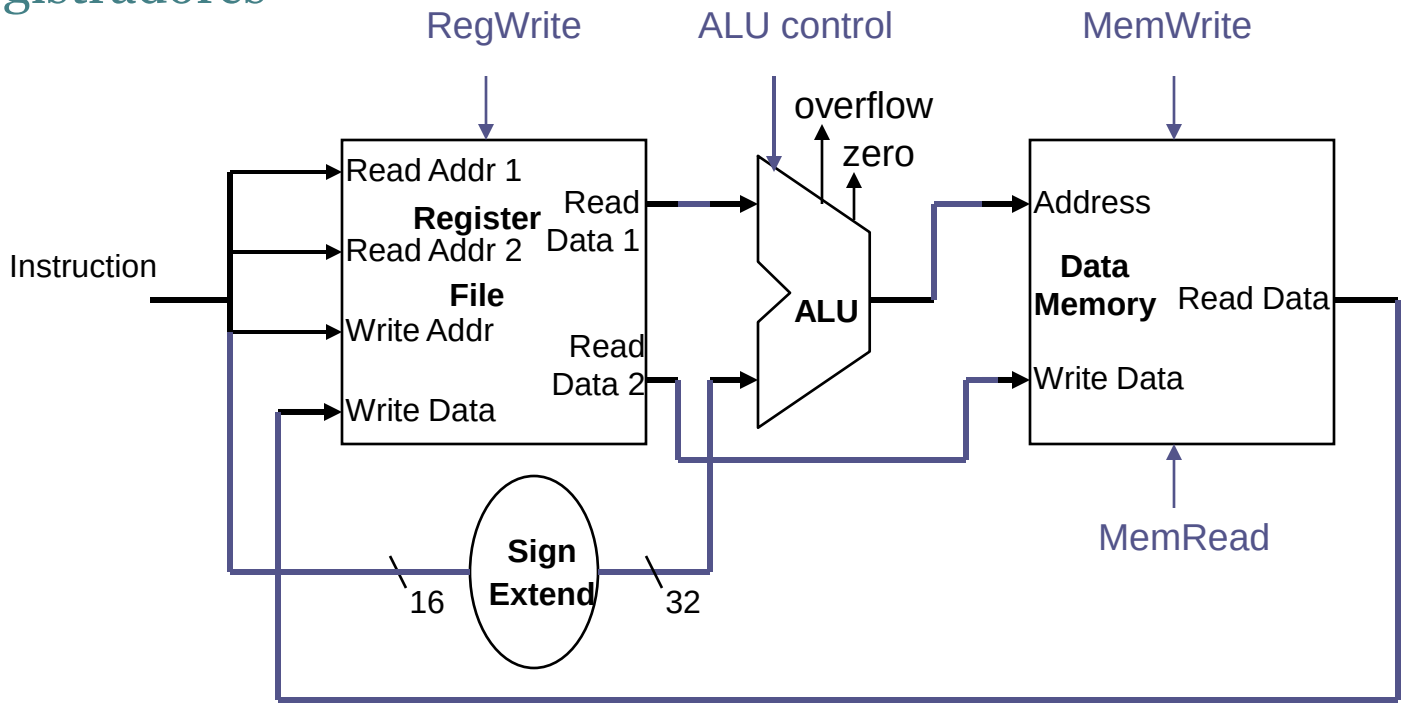
  - op + funct definem a operação executada com rs e rt
  - O resultado é armazenado em rd

- O banco de registradores não é escrito em todo ciclo (ex: `sw`). Assim, é necessário um sinal de controle explícito para escrita



Executando instruções Load e Store

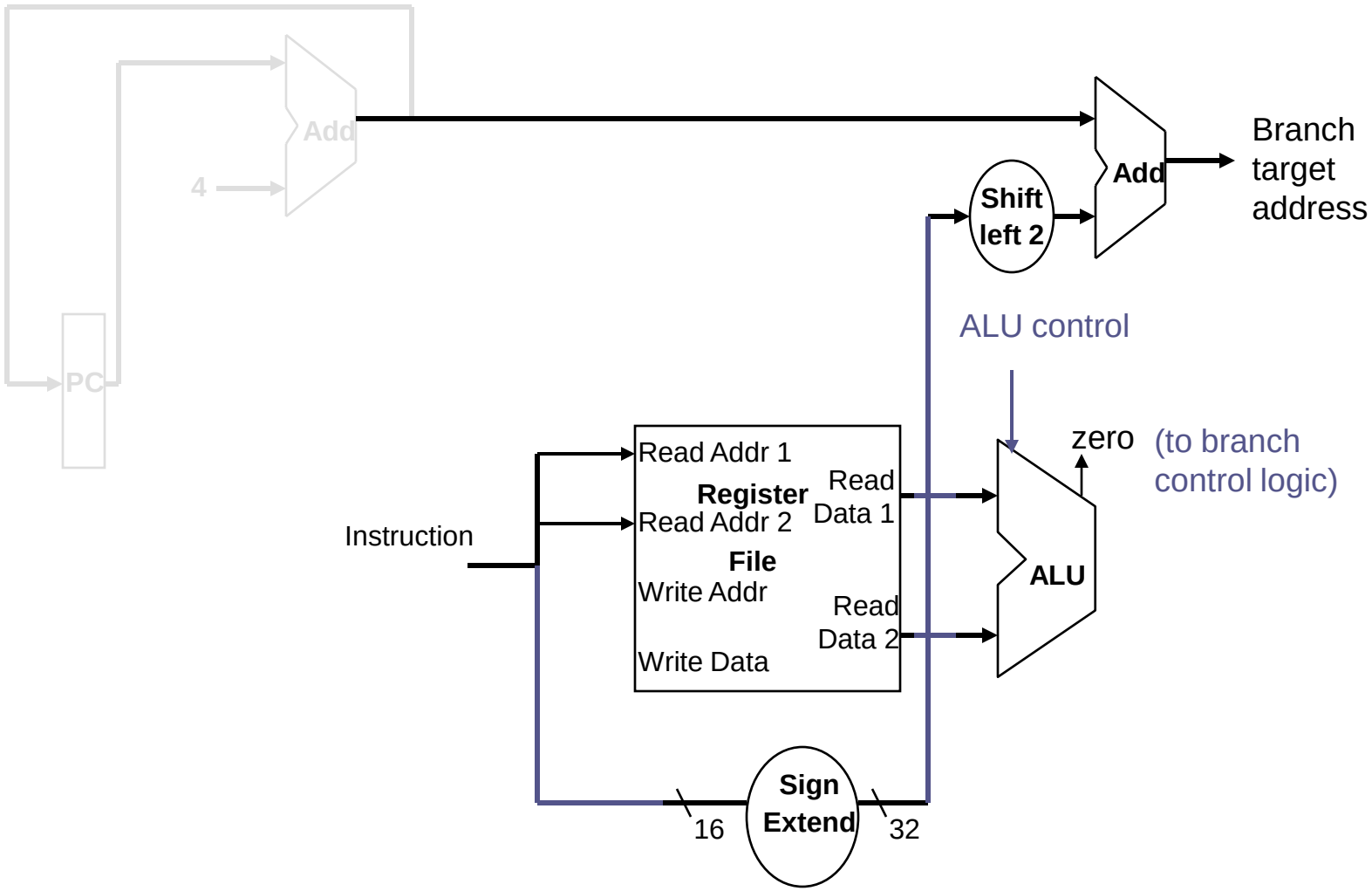
- Instruções Load e Store envolvem
  - Calcular endereço de memória somando o registrador base (lido durante a decodificação) ao campo offset de 16 bits extendido
  - store valor (lido durante a decodificação), escrito na memória de dados
  - load valor, lido da memória de dados, escrito no banco de registradores



Executando instruções de salto

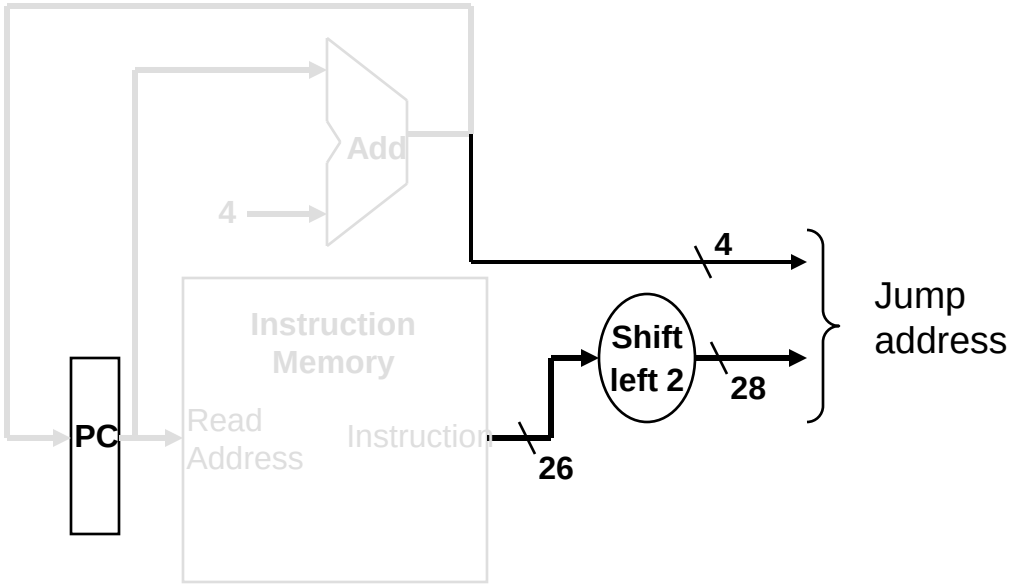
- Instruções de salto envolvem
  - Comparar os operandos lidos durante a decodificação (saída **`zero`** da ALU em caso de igualdade)
  - Calcular o endereço de salto somando o valor atualizado do PC ao campo offset de 16 bits extendido para 32 bits

Executando instruções de salto



Executando operações de salto

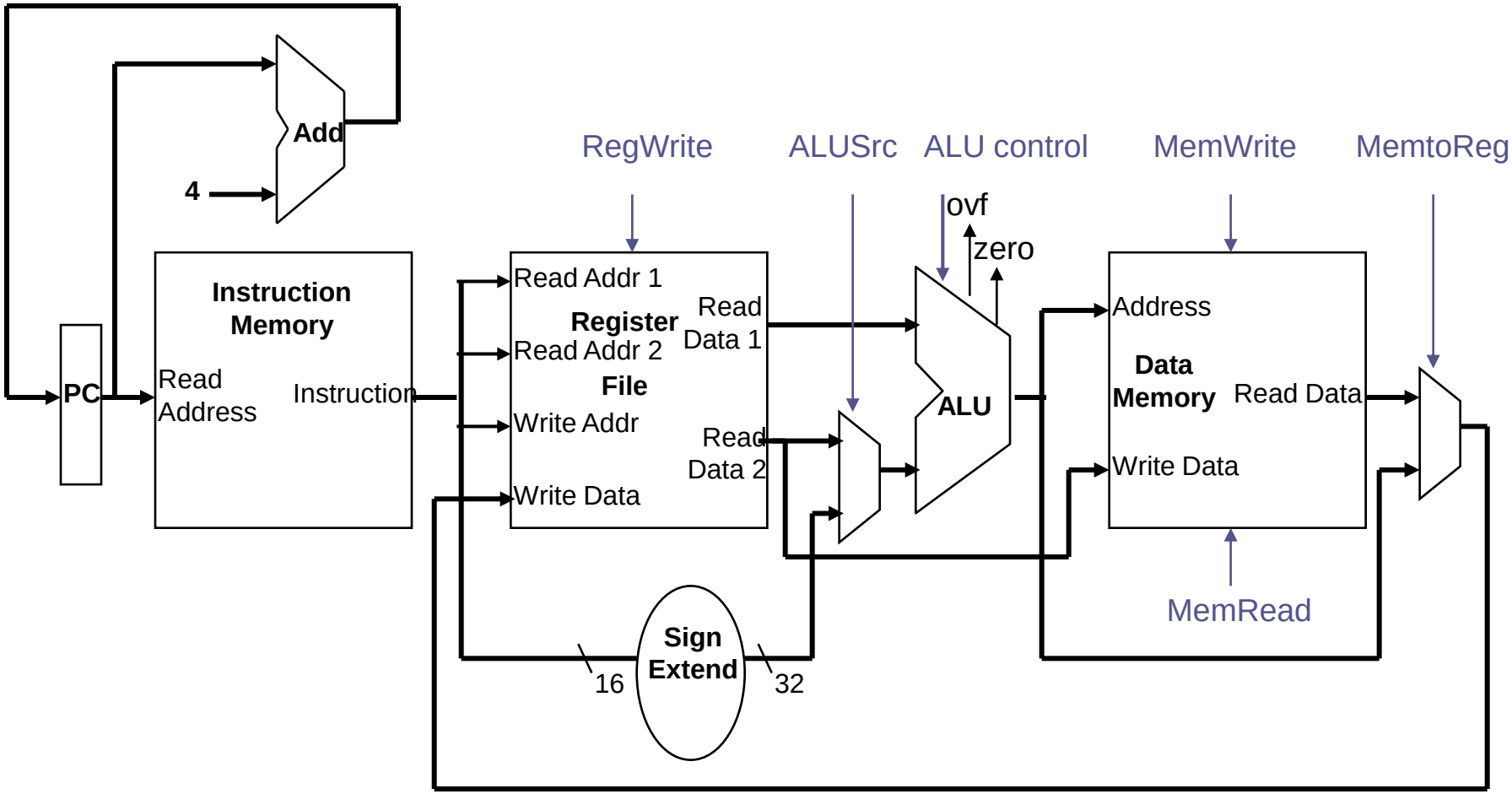
- Instruções de salto envolvem
  - Substituir os 28 bits menos significativos do PC pelos 26 bits menos significativos da instrução buscada deslocados à esquerda por 2 bits



Criando um caminho único a partir das partes

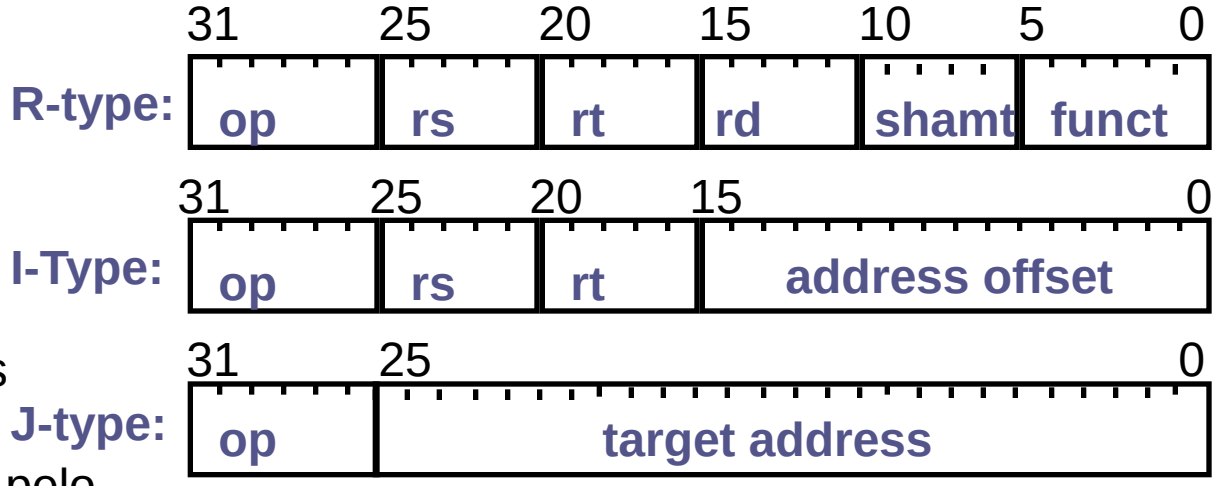
- Juntar os segmentos de caminhos de dados; adicionar as linhas de controle; adicionar multiplexadores
- Projeto de ciclo único – busca, decodificação e execução de cada instrução em apenas um ciclo de clock
  - Nenhum recurso do caminho pode ser usado mais de uma vez por instrução. Assim, precisam ser duplicados (ex., separar memória de instrução e memória de dados, e diversos somadores)
  - Multiplexadores são necessários nas entradas de elementos compartilhados com linhas de controle para seleção
  - Sinais de escrita para controle da escrita no banco de registradores e na memória de dados
- Tempo de ciclo é determinado pelo caminho mais longo

Caminho de dados para as instruções de acesso à memória e as instruções tipo R



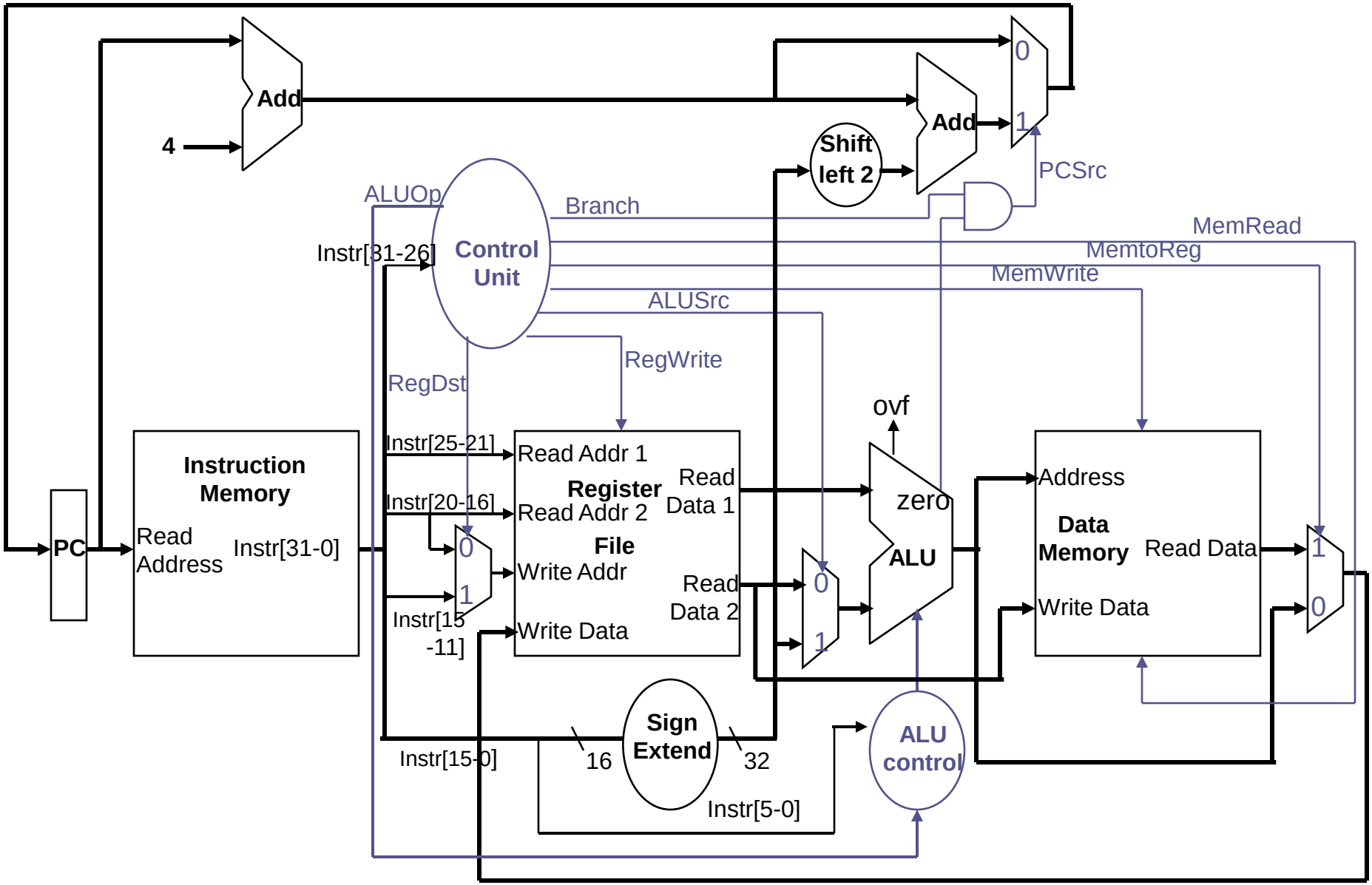
Acrescentando o controle

- Selecionando operações a realizar (ALU, Register File and Memory read/write)
- Controlando o fluxo de dados (multiplexadores)

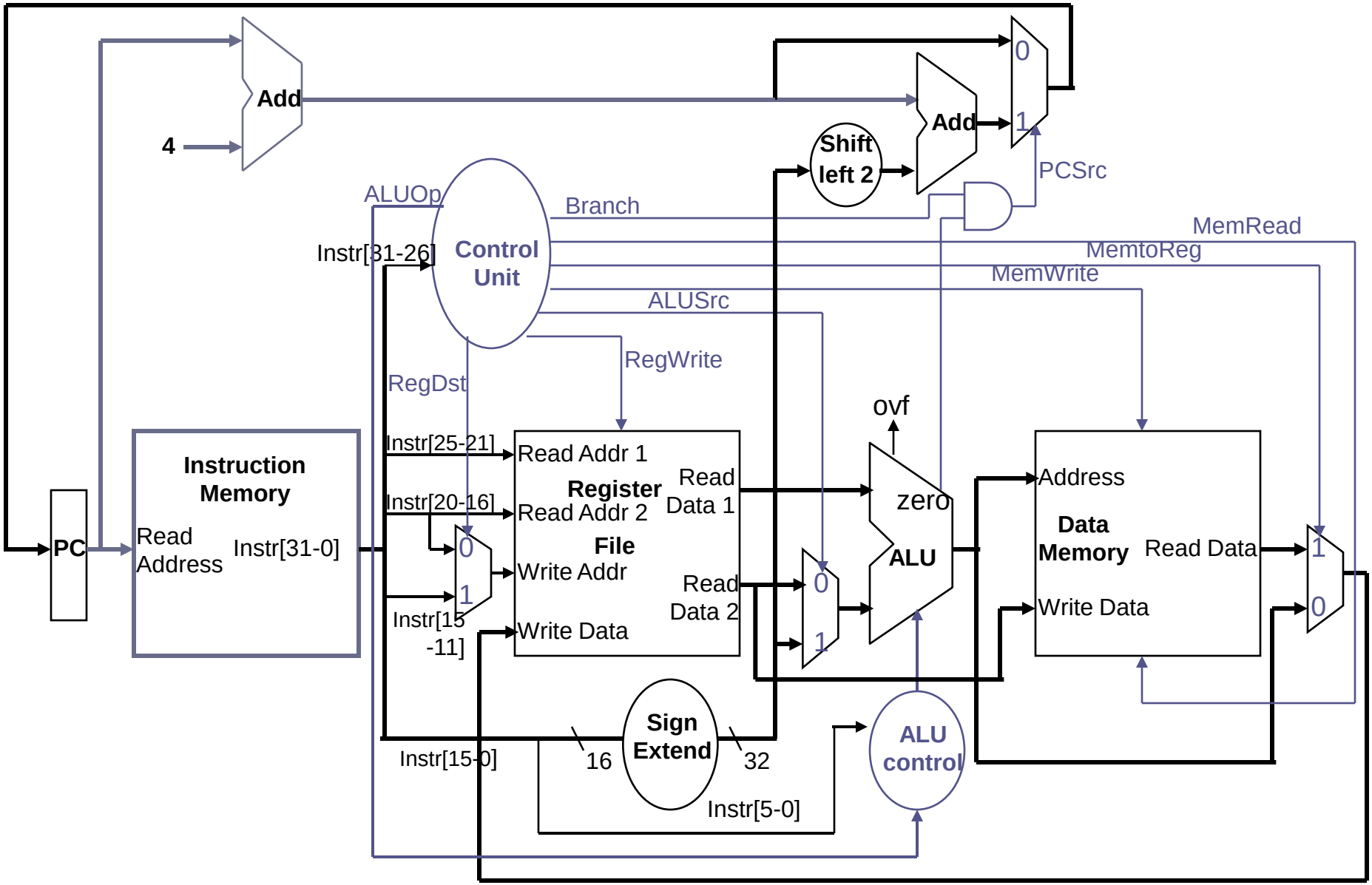


- Observações
  - Campos op sempre nos bits 31-26
  - Os dois registradoress a serem lidos são sempre especificados pelo campo rs (bits 25-21) e campo rt (bits 20-16); para lw e sw rs é o registrador base.
  - O registrador de destino está em um de dois lugares– em rt (bits 20-16) para lw; em rd (bits 15-11) para instruções do tipo R.
  - Deslocamento para beq, lw, e sw sempre nos bits 15-0

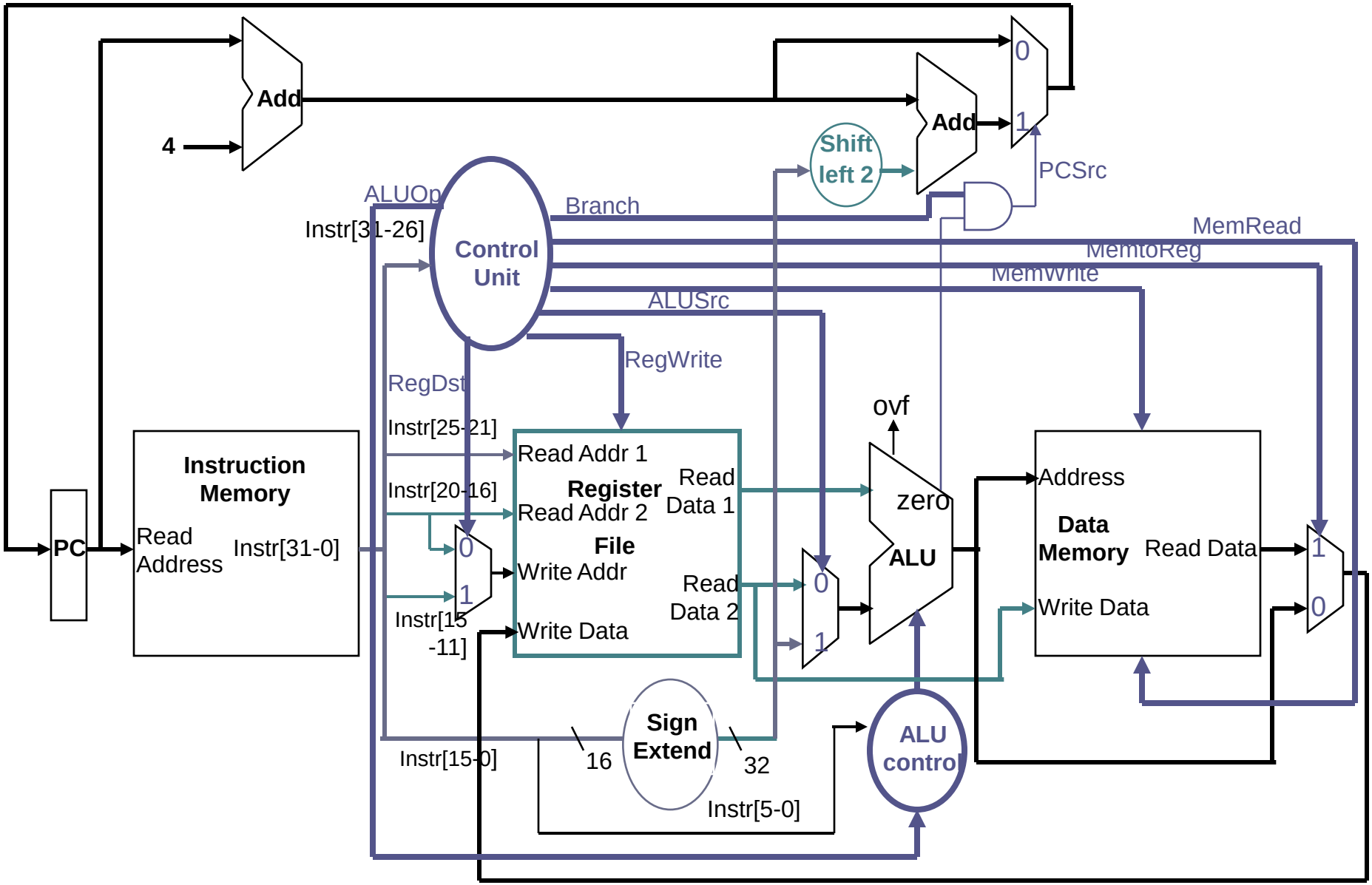
Acrescentando o controle



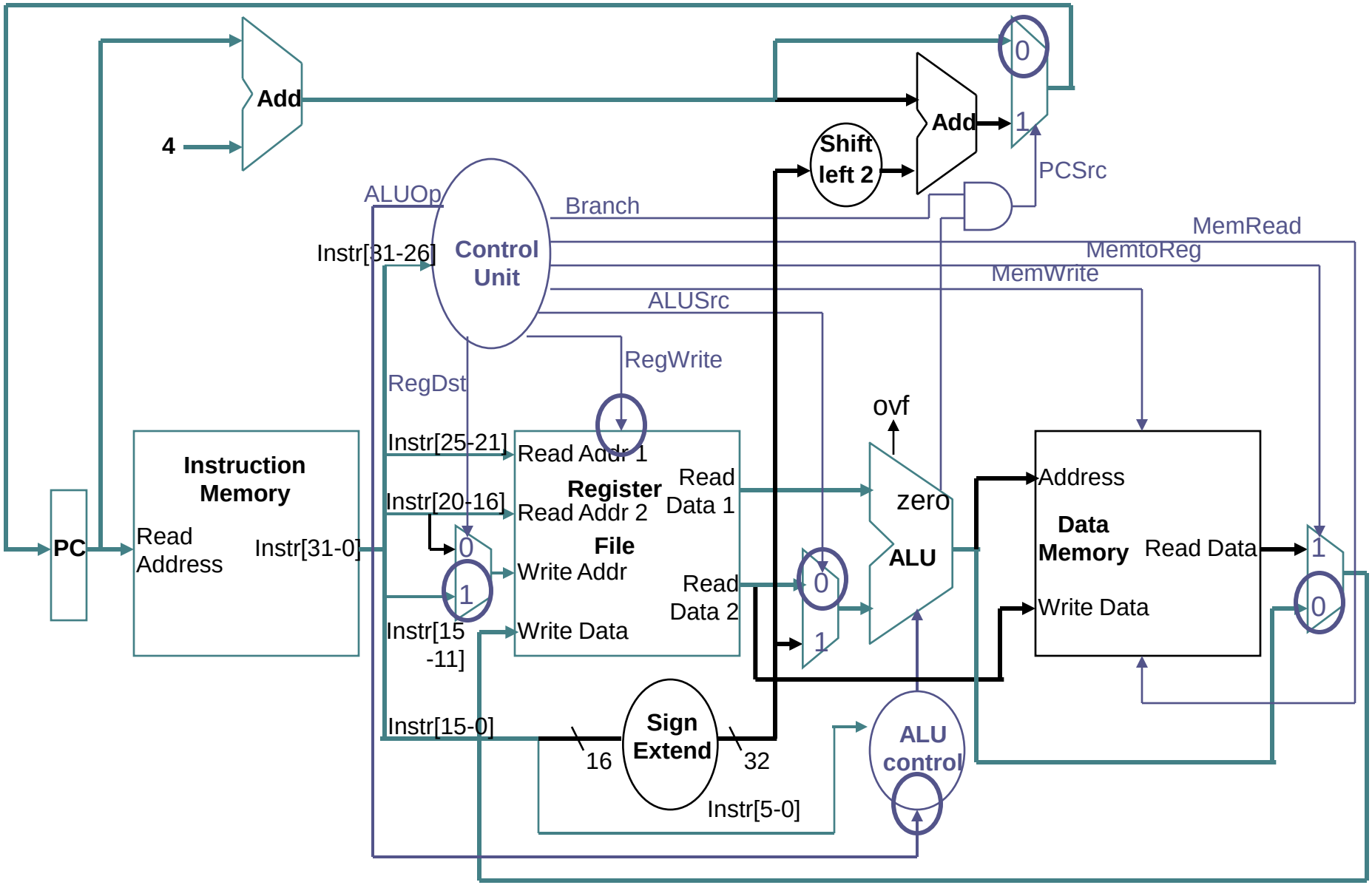
Busca de instrução



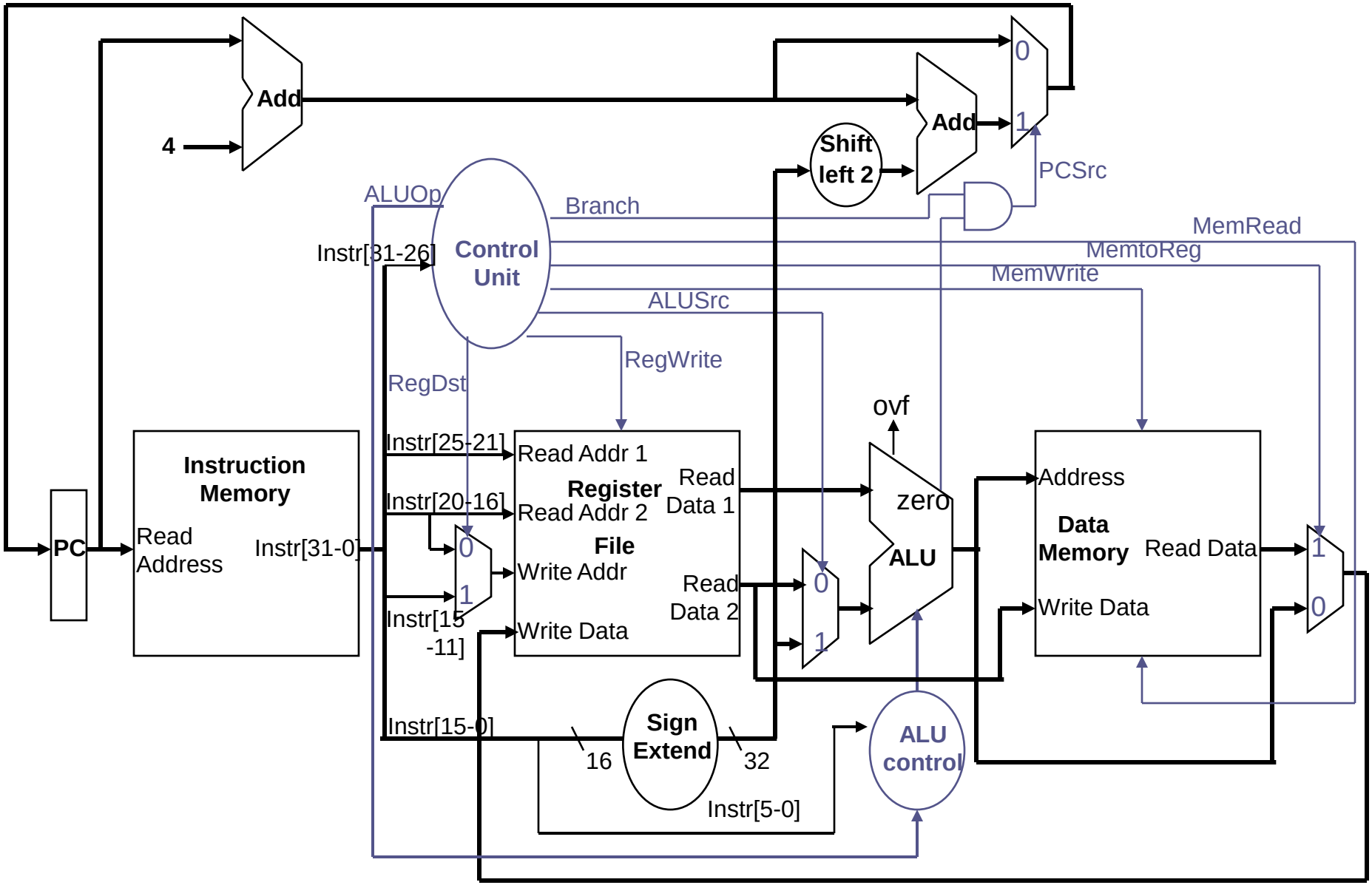
Decodificação de instrução



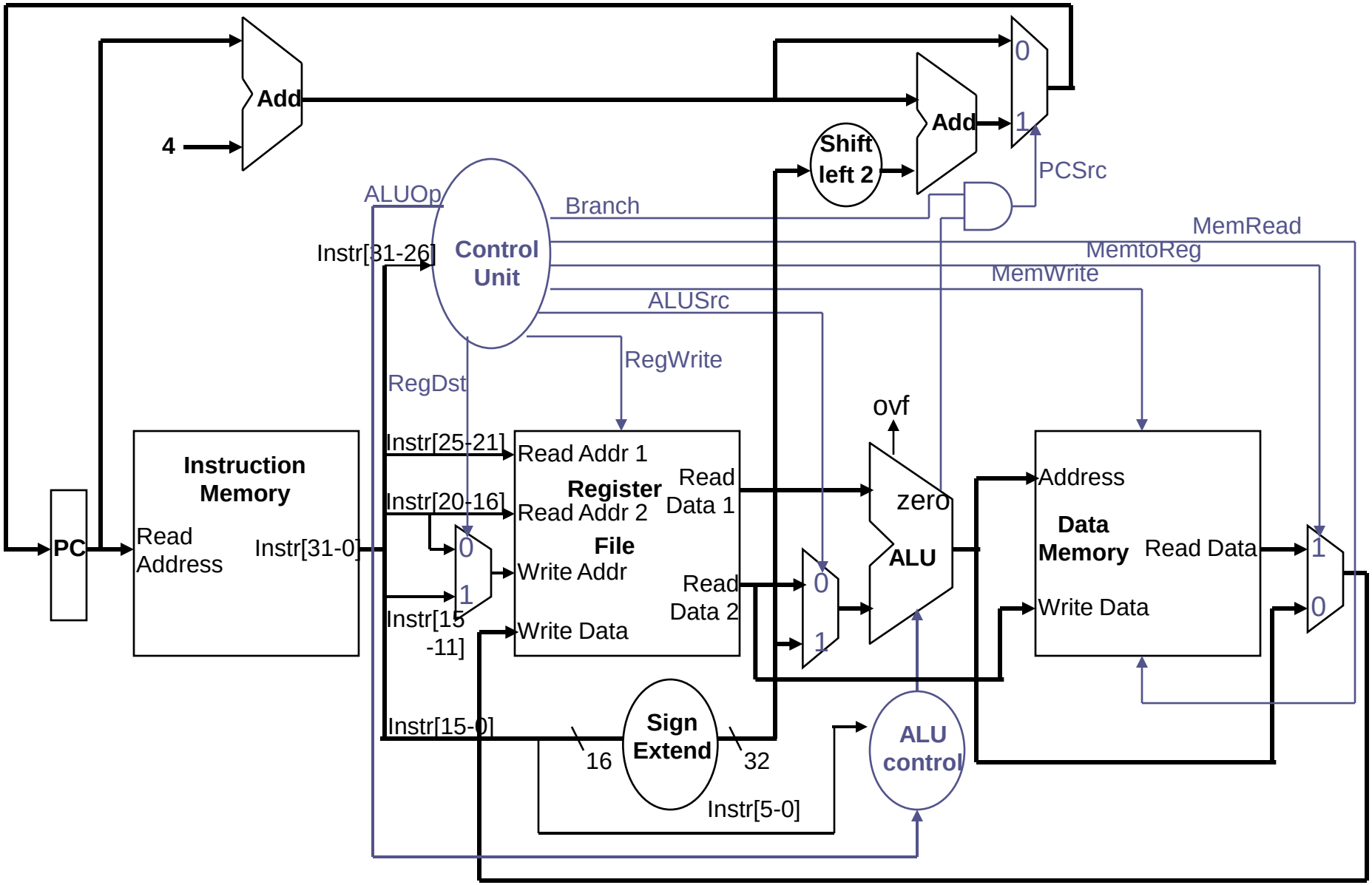
Instrução tipo R (Fluxo de dados e controle)



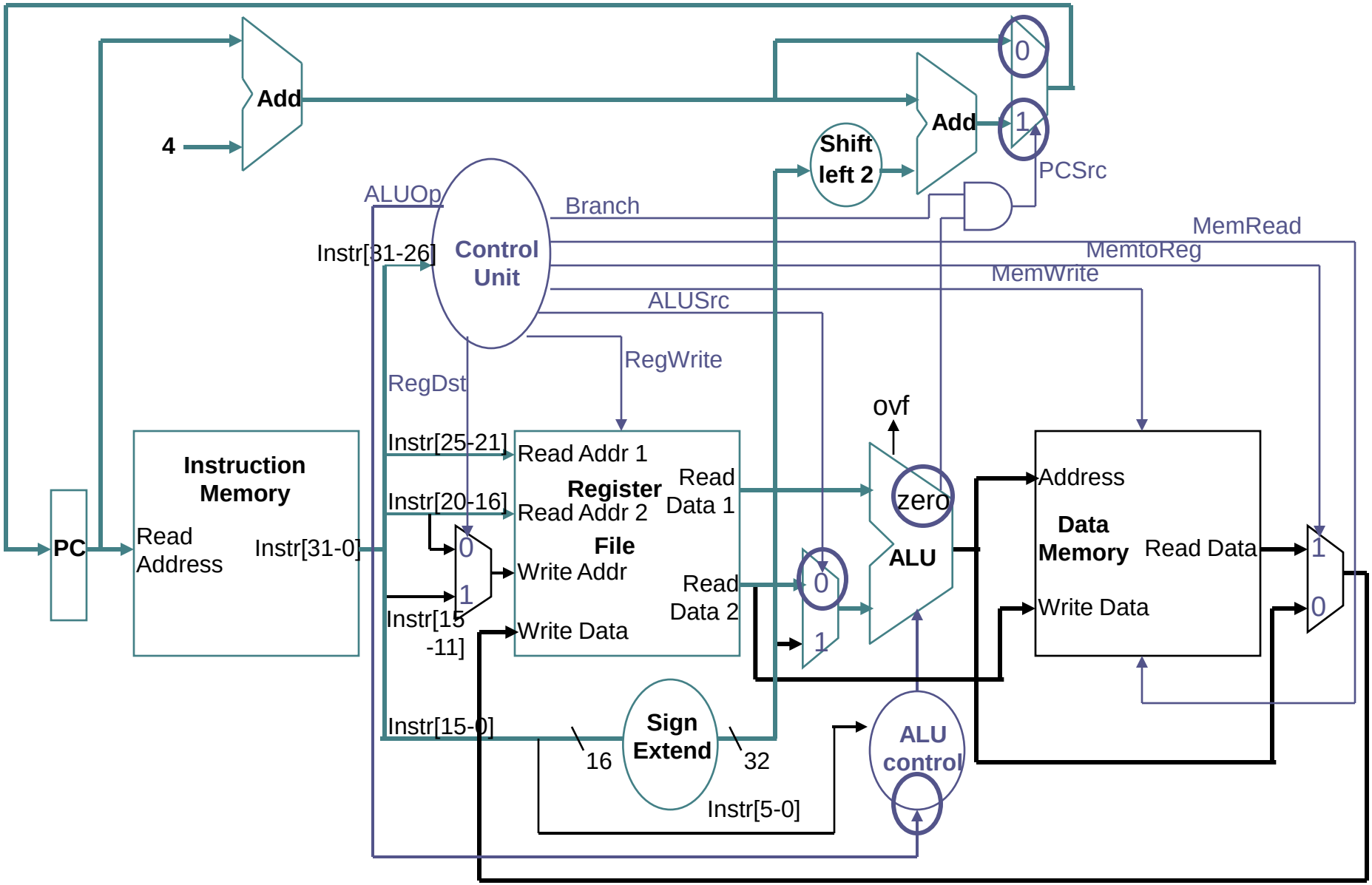
Instrução Load (Fluxo de dados e controle)



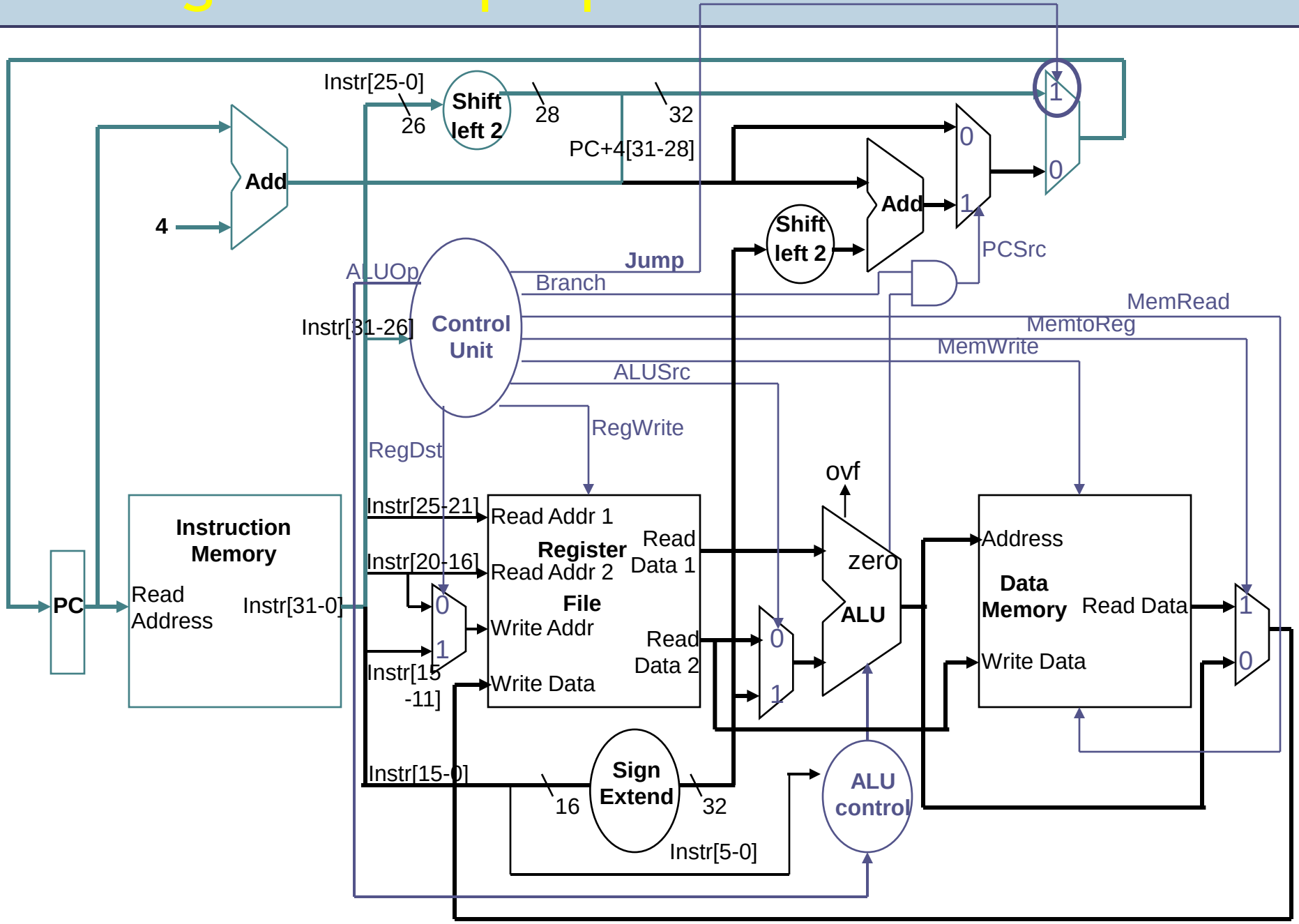
Instrução de salto (Fluxo de dados e controle)



Instrução de salto (Fluxo de dados e controle)



Adding the Jump Operation



Por que não usar uma implementação monociclo?

- Tempo de ciclo grande o suficiente para acomodar todas as instruções
- O CPI da máquina será igual a 1
- O ciclo é baseado no tempo da maior instrução: LW, que utiliza 5 unidades funcionais
- Todas as demais instruções podem ser executadas em um ciclo de clock menor → Monociclo pode se ineficiente

Exemplo

- Suponha que o tempo de operação das principais unidades funcionais de uma certa implementação são os seguintes:
  - Unidades de memória: 2 ns.
  - UAL e somadores: 2 ns.
  - Banco de registradores (leitura ou escrita): 1 ns.
- Admitindo que os multiplexadores, a unidade de controle, os acessos ao PC, a unidade de extensão de sinal e os fios não contribuem com qualquer retardo, verifique qual das duas implementações abaixo é mais rápida, informando também a relação de performance:
  - 1- Uma implementação na qual todas as instruções operam em 1 ciclo de clock de tamanho fixo.
  - 2 - Uma implementação na qual todas as instruções operam em 1 ciclo de clock de tamanho variável, usando um clock de tamanho variável, que assume o tamanho exato da duração de cada instrução

Exemplo

- Pressuponha o seguinte mix de instruções:
  - 24% de instruções de load word,
  - 12% de instruções de store word,
  - 44% de instruções do tipo R,
  - 18% de desvio condicional e
  - 2% de desvio incondicional

Exemplo

- Solução:

Tempo de processador = NI X CPI X Tclock

- Monociclo:
- O ciclo de clock definido pela instrução mais lenta o lw, que usa 5 unidades funcionais:
  - Busca de Instruções na memória: 2 ns
  - Acesso ao banco de registradores: 1ns
  - Execução na UAL: 2 ns
  - Acesso à memória de dados: 2 ns
  - Escrita em registrador: 1 ns
- **Total de 8 ns são gastos pela instrução de load**

Exemplo

- Tamanho exigido para cada classe de instrução:

Classe de instrução	Memória de Instrução	Leitura de registrador	Operação ALU	Memória de dados	Leitura de registrador	Total
Tipo R	2	1	2	0	1	6
lw	2	1	2	2	1	8
sw	2	1	2	2		7
branch	2	1	2	0		5
jump	2					2

Exemplo

- Clock variável:
  - Ciclo de clock = 8*24% + 7*12 % + 6*44% + 5*18% + 2*2% = 6,3

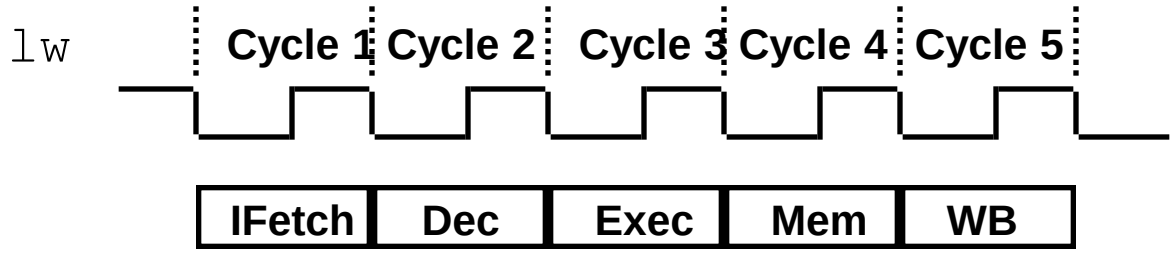
$$\frac{Performance_{clock\ fixo}}{Performance_{clock\ variável}} = \frac{tempo\ de\ execução_{clock\ fixo}}{tempo\ de\ execução_{clock\ variável}} =$$

$$\frac{IC * ciclo\ de\ clock_{clock\ fixo}}{IC * ciclo\ de\ clock_{clock\ variável}} = \frac{ciclo\ de\ clock_{clock\ fixo}}{ciclo\ de\ clock_{clock\ variável}} = \frac{8}{6,3} = 1,27$$

- A implementação de clock variável é 27% mais rápida que a implementação de clock fixo.

Implementação Multiciclo

- Monociclo → clock dado pelo LW
- Multiciclo: cada passo executado por uma unidade funcional gastará um ciclo de clock de forma que:
  - Balancear a quantidade de trabalho em cada estágio
  - O uso de cada uniade funcional seja restrito a cada ciclo de clock

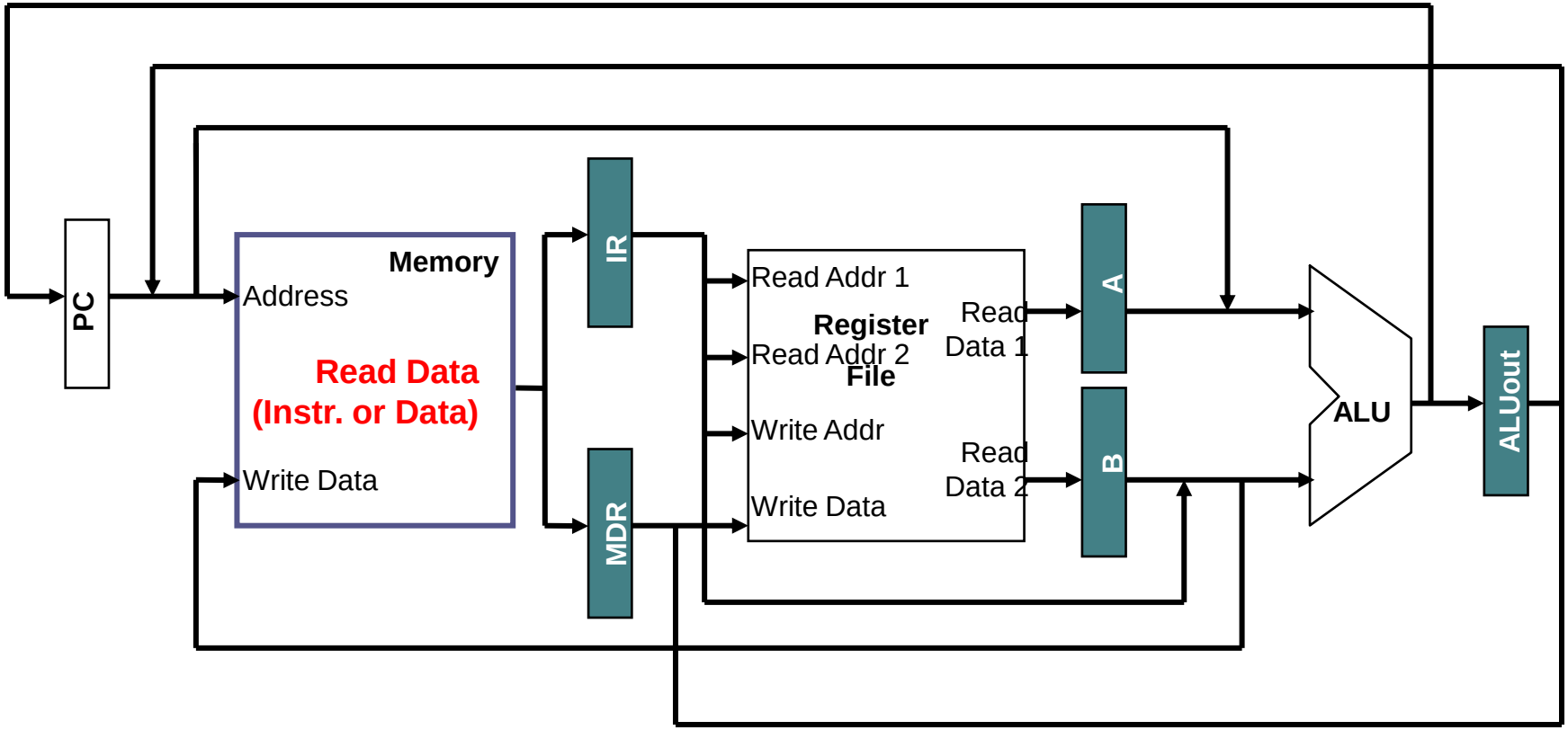


- Nem todas as instruções gastam o mesmo número de ciclos
- Taxa de clock mais rápidas
- Multiciclo permite que as unidades funcionais sejam usadas mais do que uma vez por instrução em diferentes ciclos

Caminho de dados: Multiciclo

- A inclusão de registradores no caminho de dados é determinada por dois fatores:
  - Quais as unidades funcionais ficarão restritas a um único ciclo
  - Quais os dados serão usados em ciclos posteriores de execução da instrução
- Na implementação multiciclo o ciclo de clock deve acomodar, quando muito, as seguintes operações:
  - Um acesso à memória
  - Acesso ao banco de registradores (duas leituras e uma escrita), ou
  - Uma operação da UAL
- Os dados nessas unidades funcionais precisam ser armazenados em registradores temporários para serem usados em ciclos posteriores:
  - IR (Instruction Register) – armazena uma instrução lida
  - MDR (Memory Data Register) – armazena um dado lido
  - A e B – armazenam os valores dos operandos lidos do banco de registradores
  - UALSaída – armazena o resultado da UAL

Caminho de dados: Multiciclo

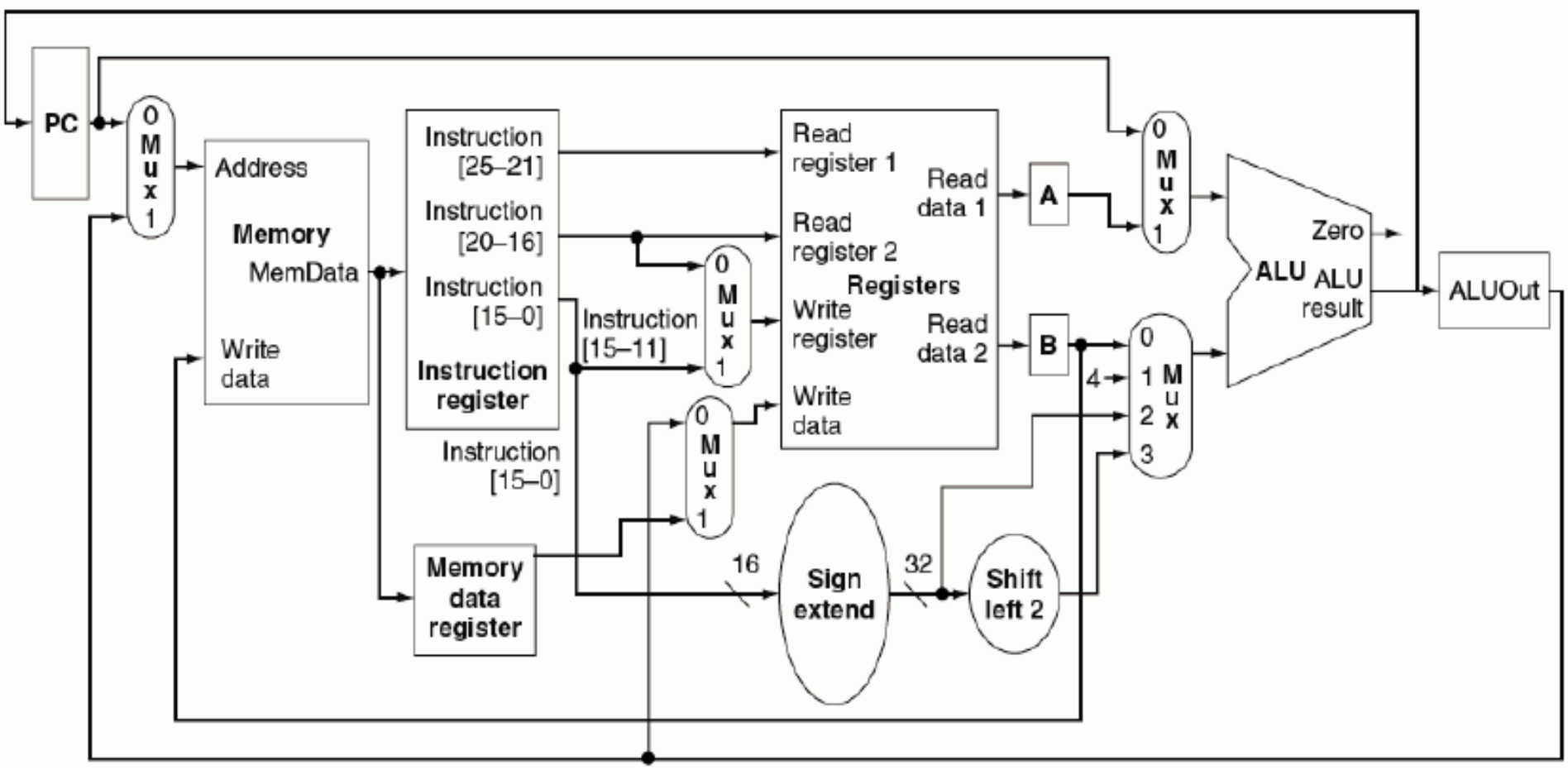


- Única **memória** para dados e instruções.
- **Uma UAL** em vez de uma UAL e dois somadores.
- Colocação de mais **um ou mais registradores** depois de cada uma das unidades funcionais, para reter o resultado calculado até que ele seja usado no ciclo de clock seguinte.

Caminho de dados: Multiciclo

- Para que as unidades funcionais possam ser compartilhadas e para a substituição dos dois somadores da implementação monociclo, foram introduzidos outros multiplexadores:
  - Entrada superior da UAL: multiplexador escolhe entre o reg. A e o PC
  - Segunda entrada da UAL ampliada de 2 para 4 entradas:
    - Incremento de 4 para o PC
    - Deslocamento usado para o cálculo do desvio condicional
    - Esse multiplexador necessitará de duas linhas de controle

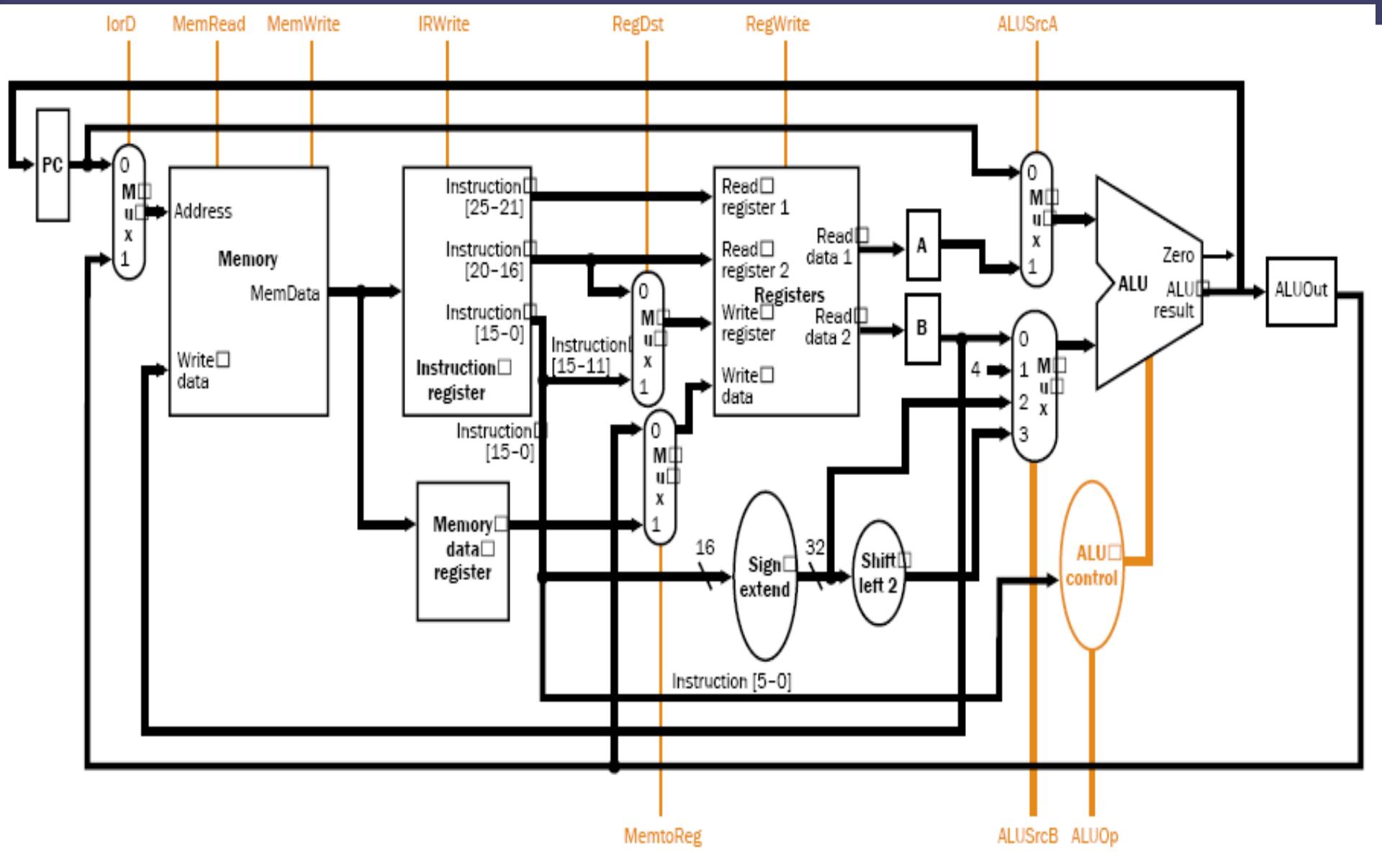
Caminho de dados: Multiciclo



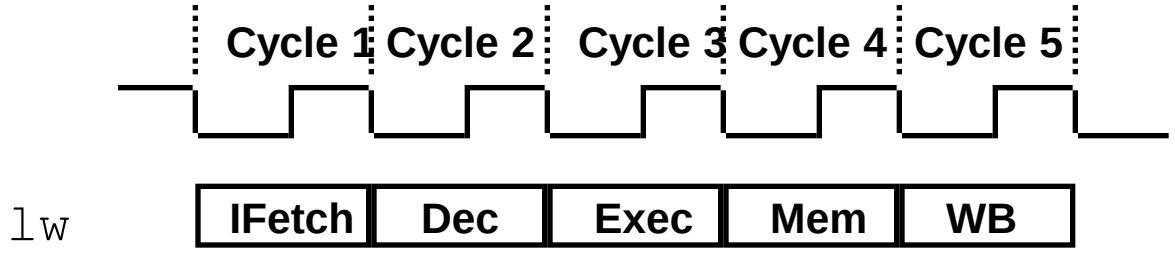
Caminho de dados: Multiciclo

- Os elementos de estados visíveis ao programador vão precisar de sinais de controle: PC, memória e registradores e IR
- Para as instruções de desvio existem três possíveis fontes para o valor a ser escrito no PC:
  - A saída da UAL assume o valor de PC + 4
    - Durante a busca da instrução, esse valor pode ser armazenado diretamente no PC: PC + 4 (mux)
  - A saída da UAL irá armazenar o endereço de desvio condicional após ele ter sido calculado, sinal estendido de 32 bits
  - Para uma instrução de jump, a UAL armazenará:
    - (PC + 4) + sinal estendido << 2 bits

Caminho de dados: Multiciclo



Os cinco passos da instrução Load

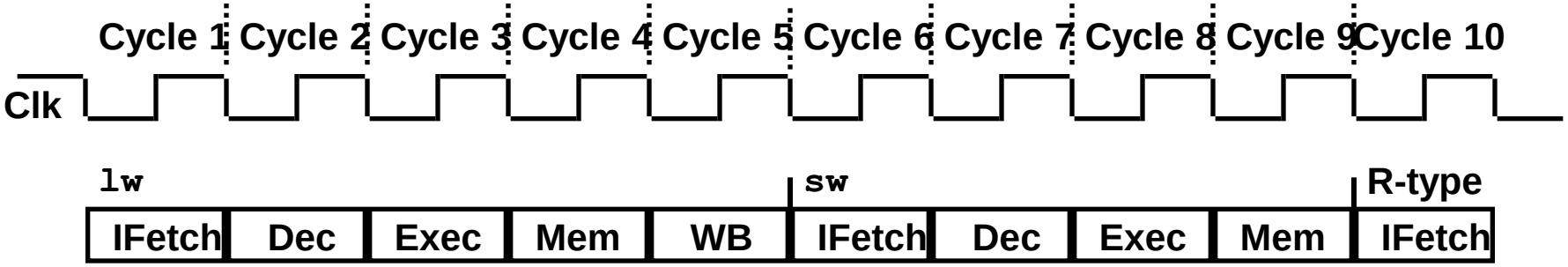


- IFetch: Busca de instrução e atualização do PC
- Dec: Decodificação de instrução, Leitura de registrador, Campo de offset de sinal estenddido
- Exec: Executa instruções tipo R; Calcula Endereço de memória; Comparação para desvio; Jump
- Mem: Leitura da memória; Escrita na memória; Instruções do tipo R (RegFile write)
- WB: Leitura da memória, loads são completados escrevendo os valores na memória (RegFile write)

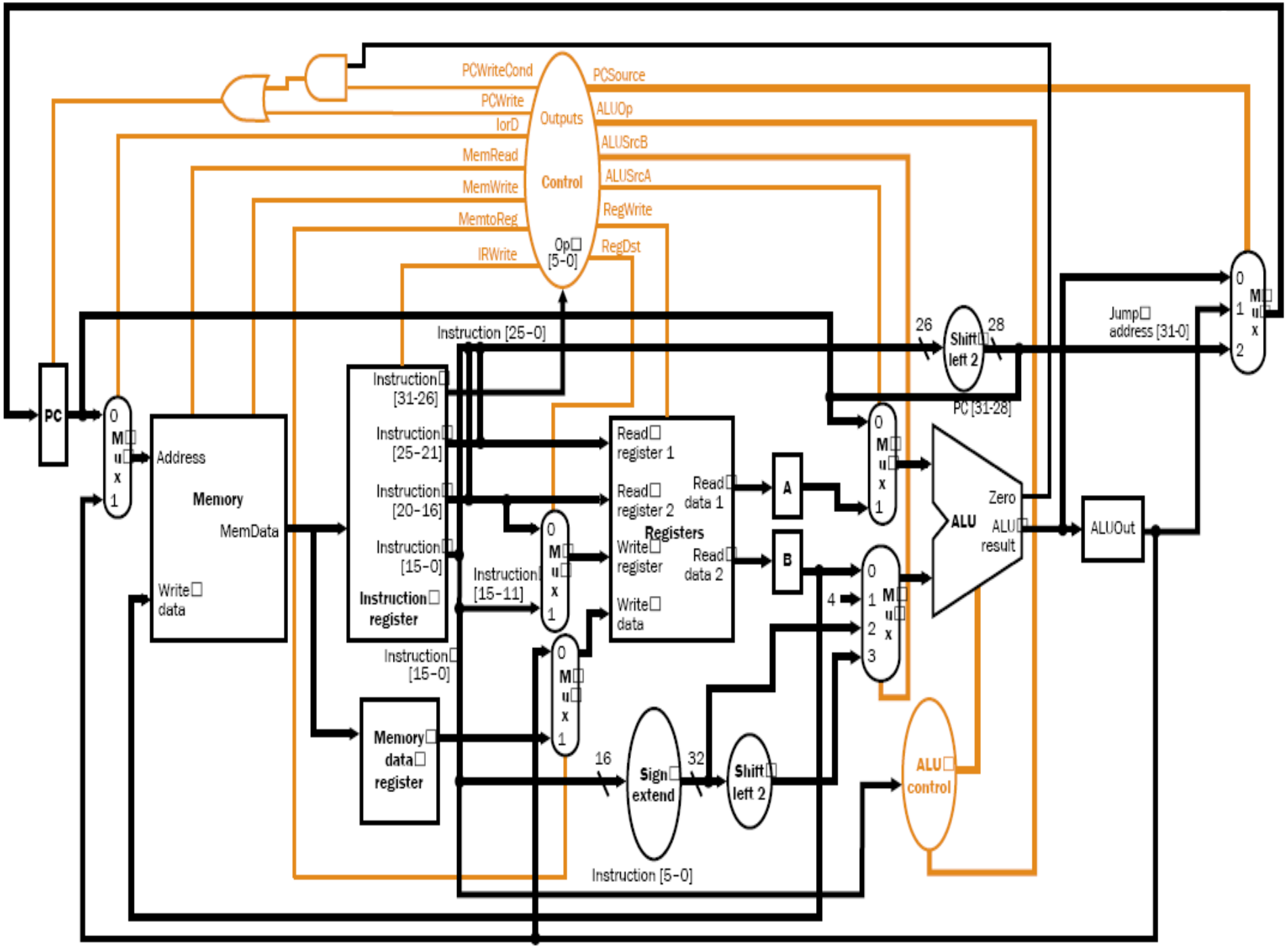
INSTRUÇÕES PRECISAM DE 3 - 5 ETAPAS!

Multiciclo: vantagens e desvantagens

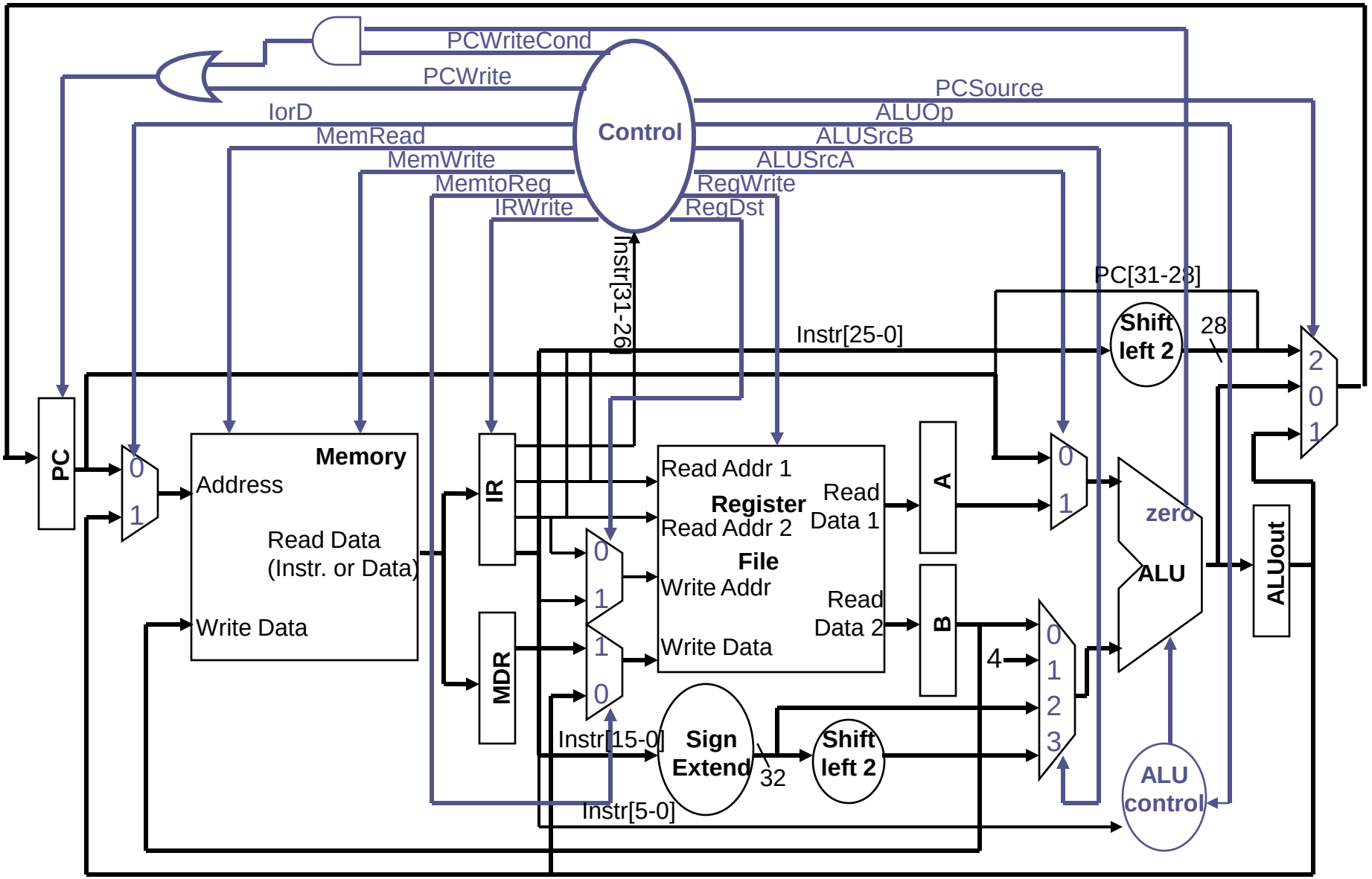
- Usa o ciclo de clock de forma eficiente - o ciclo de clock está programado para acomodar a etapa de instrução mais lenta.



- Implementações multiciclo permitem que as unidades funcionais sejam utilizadas mais de uma vez por instrução desde que sejam utilizados nos diferentes ciclos de clock, mas...
- Requer registros de estado adicionais internos, mais multiplexadores, e de controle mais complicado (FSM).


PCWriteCond
PCWrite
IorD
MemRead
MemWrite
MemtoReg
IRWrite
Outputs
Control
Op [5-0]
PCSource
ALUOp
ALUSrcB
ALUSrcA
RegWrite
RegDst
PC
0
M
u
x
1
Address
Memory
MemData
Write data
Instruction [31-26]
Instruction [25-21]
Instruction [20-16]
Instruction [15-0]
Instruction register
Instruction [15-0]
Instruction [15-11]
Memory data register
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
A
B
4
ALU
Zero
ALU result
ALUOut
26
Shift left 2
28
PC [31-28]
Jump address [31-0]
16
Sign extend
32
Shift left 2
ALU control
Instruction [5-0]

Caminho de dados multiciclo com sinais de controle



Nome do passo	Ação nas instruções R	Ação nas instruções de memória	Ação na instrução de desvio cond.	Ação na instrução de desvio incond.
Busca da instrução	IR = Memória[PC] PC = PC + 4			
Decodificação	A= Reg[IR[25-21]]] B = Reg[IR[20-16]] UALSaída = PC + extensão do sinal (IR[15-0]) <<2)			
Execução, cálculo do endereço, término de uma instrução de branch/jump	ALUOut = A op B	UALSaída = A + extensão do sinal (IR[15-0])	Se (A == B) então PC = UALSaída	PC = PC[31-28] || (IR[25-0] <<2)
Término de uma instrução de referência à memória ou do tipo R	Reg(IR[15-11]) = UALSaída	Load: MDR = memória[UALSaída] ou Store: Memória[UALOut] = B		
Término de leitura da memória		Load: Reg[IR{20-16]] = MDR		

CPI em uma CPU Multiciclo

- **Usando um mix de instruções com 25% de loads, 10% de stores, 11% de branches, 2% de jumps e 52% de ALU (todo o restante do mix) e considerando o número de ciclos de clock para cada classe de instrução:**
  - **Loads: 5**
  - **Stores: 4**
  - **Instruções da ALU: 4**
  - **Branches: 3**
  - **Jumps: 3**

CPI = 0,25*5+0,10*4+0,52*4+0,11*3+0,02*3 = 4,12

Esse CPI é melhor ou pior que uma instrução monociclo?

Implementação do Controle