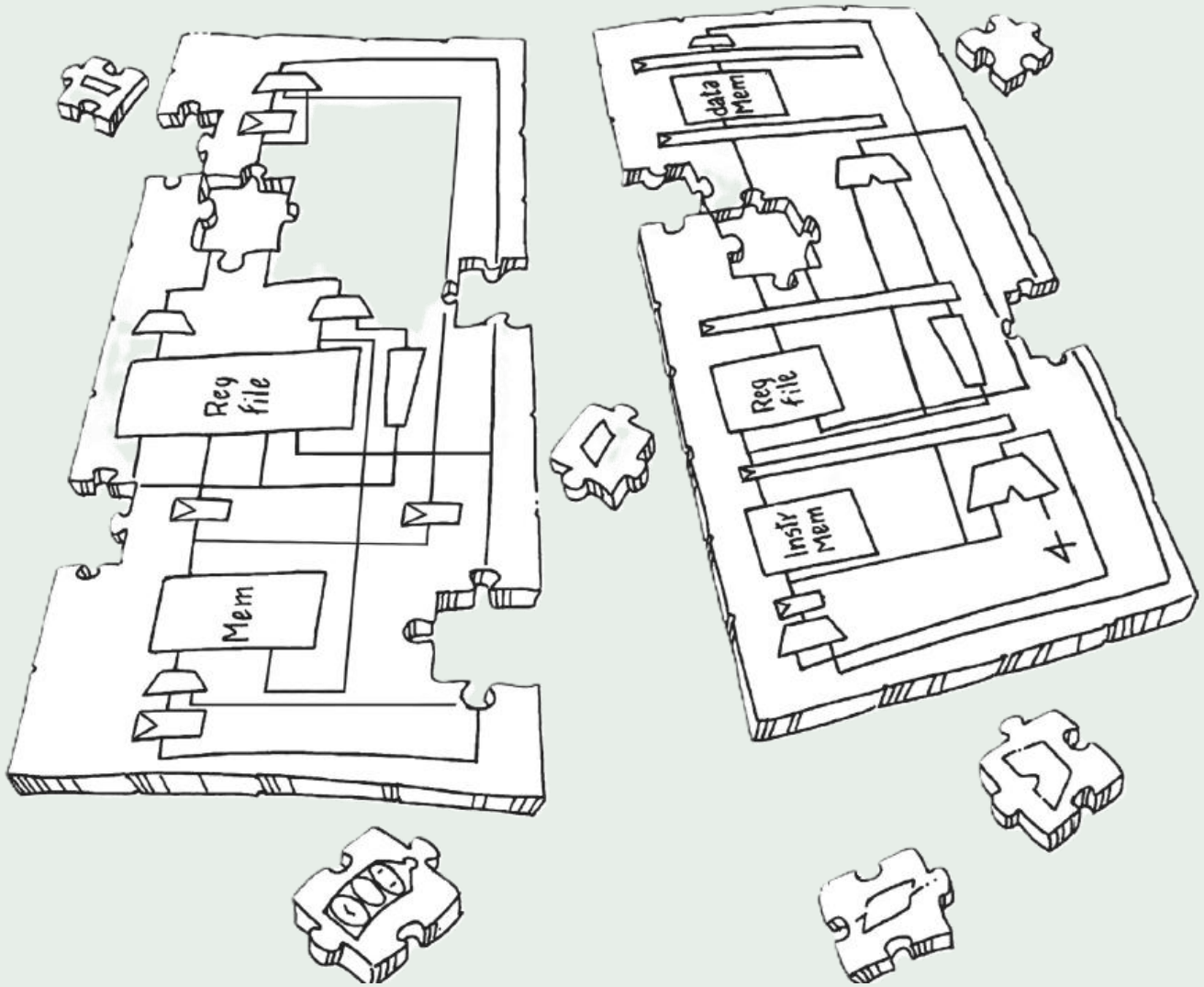


Fonte: Adaptado de Digital Design and Computer Architecture RISC-V Edition Sarah L. Harris David Harris

Arquitetura e Organização de Computadores 1

Prof. Luciano de Oliveira Neris
luciano@dc.ufscar.br

Construindo um Datapath e Controle RISC-V

Processador Simplificado

- Vamos ver como um processador é implementado
  - Iniciamos c/ um processador muito simples, e adicionamos recursos extras passo-a-passo

- ISA adotada: subconjunto do RISC-V:
  - Instruções de acesso à memória: lw and sw
  - Instruções da ALU: add, sub, or, slt
  - Instruções p/ fluxo de controle: beq and jal

Processador Simplificado

- Subconjunto RISC-V adotado:

lw: load word
sw: store word

add: adição
sub: subtração
or: ou bit-a-bit
slt: set if less than

beq: branch if equal
jal: desvio incondicional

Processador Simplificado

- Três Passos:
  - (FETCH / Busca): Acessar o endereço de memória correspondente para carregar a próxima instrução (32 bits)
  - (DECODE / Decodificação): Ler 0, 1 ou 2 registradores de acordo com os campos correspondentes na instrução
  - (EXECUTE / Execução): Executar a instrução

Processador Simplificado

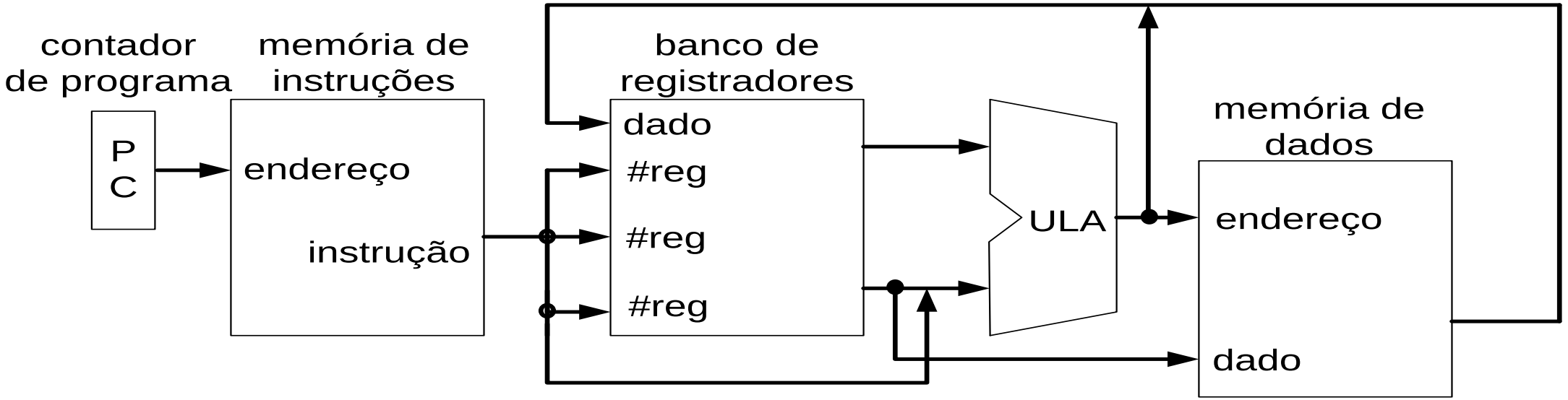
- Executar a instrução
  - Felizmente, existem algumas similaridades na execução de diferentes instruções: a maioria delas utiliza a ULA p/ calcular um valor numérico, ou um endereço de memória.

- Também existem diferenças:
  - Apenas duas delas acessam a memória
  - Store não escreve em nenhum registrador
  - Apenas branch e jal modificam o valor do PC

Processador Simplificado

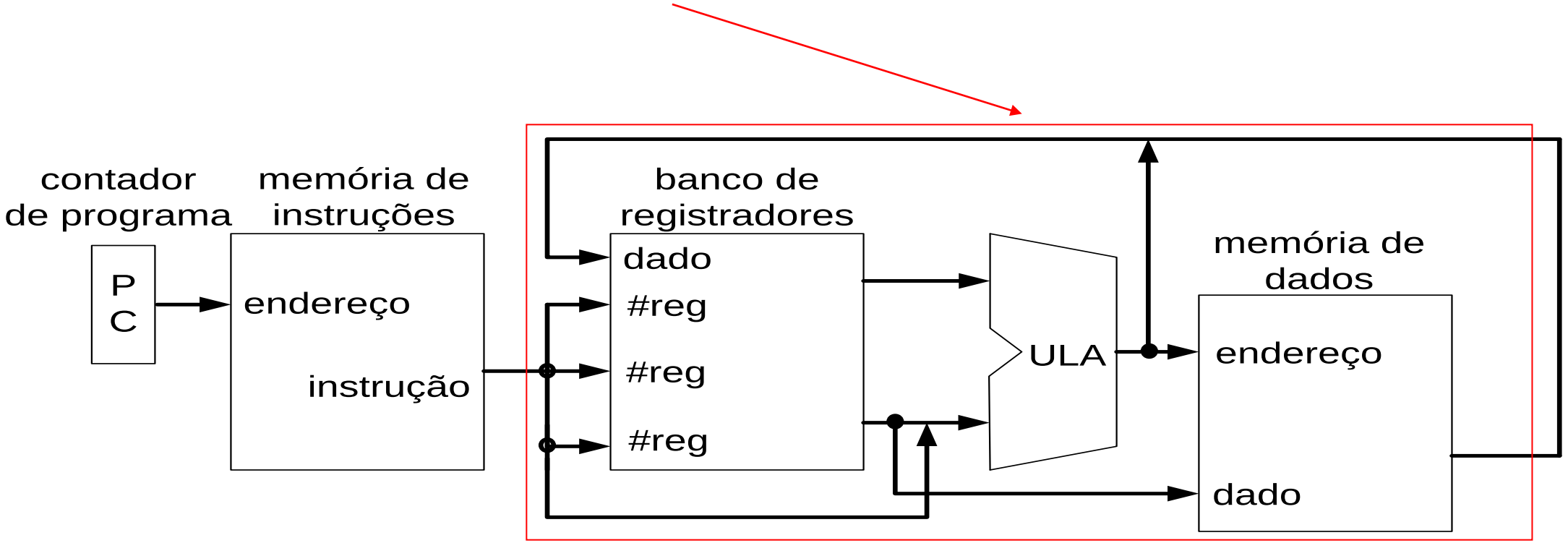
- Para implementar os 3 passos anteriores, nosso processador terá 5 componentes:
  - Registrador PC (Program Counter) fornece o endereço da instrução
  - Memória* onde as *instruções* são lidas (o campo de opcode é usado para executar a operação correspondente)
  - Memória* onde os *dados* são lidos ou gravados
  - ULA
  - Conjunto de registradores
- **Obs: Computadores c/ Memórias separadas p/ instruções e dados são conhecidos como "Arquitetura Harvard"*

Processador Simplificado

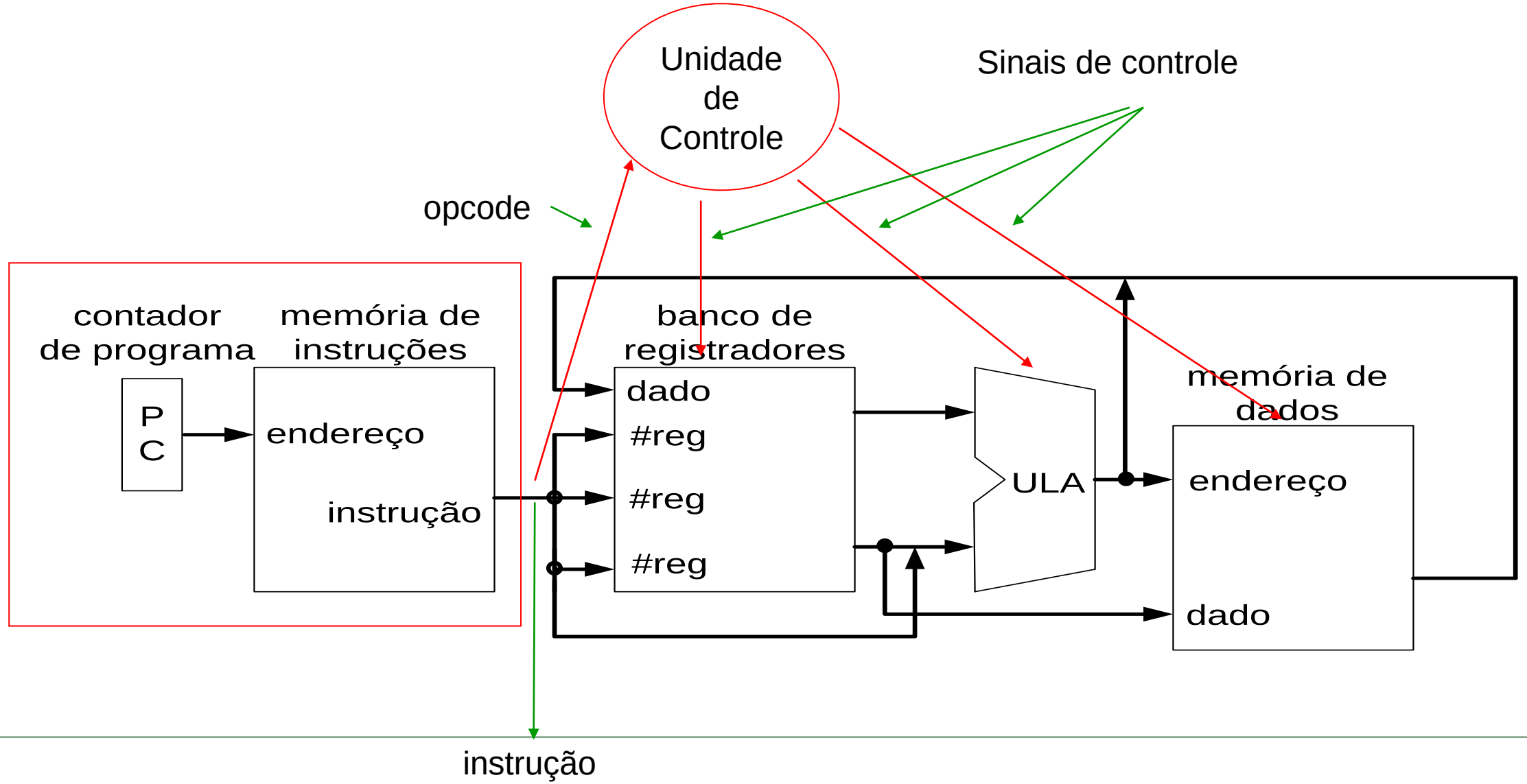


Processador Simplificado

À direita da figura vemos os elementos principais do fluxo de dados do RISC-V



Processador Simplificado



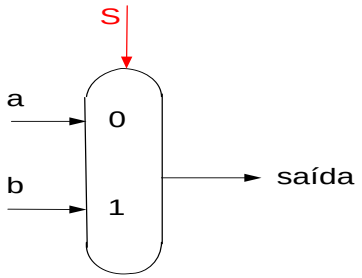
Unidade de Controle

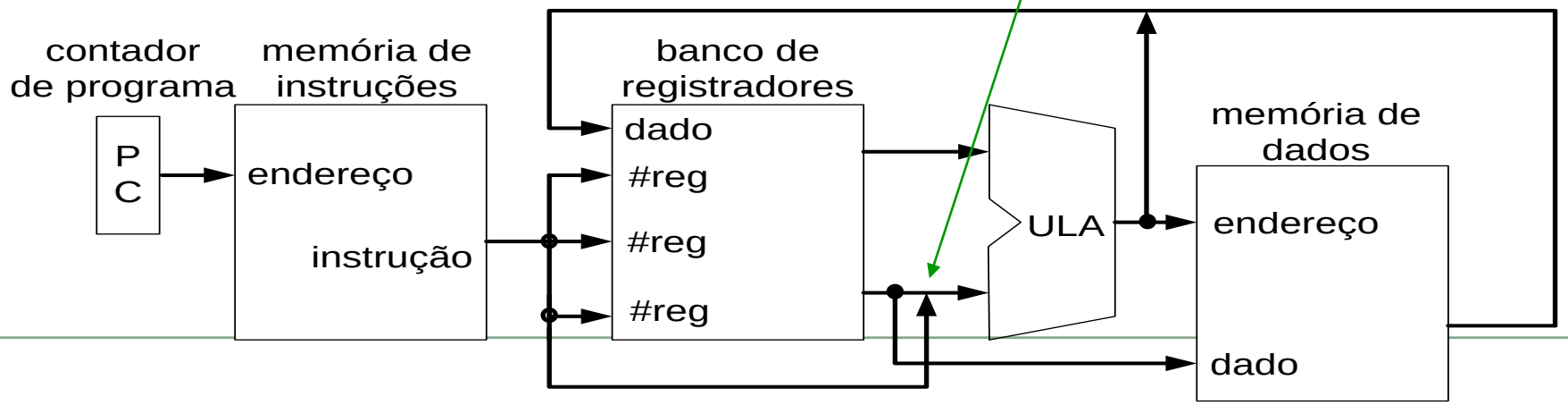
- Envia os bits de controle apropriados p/ todos os multiplexadores e componentes da CPU, de acordo c/ a instrução sendo executada.

- Isso é feito baseado nos bits do opcode da instrução

Unidade de Controle

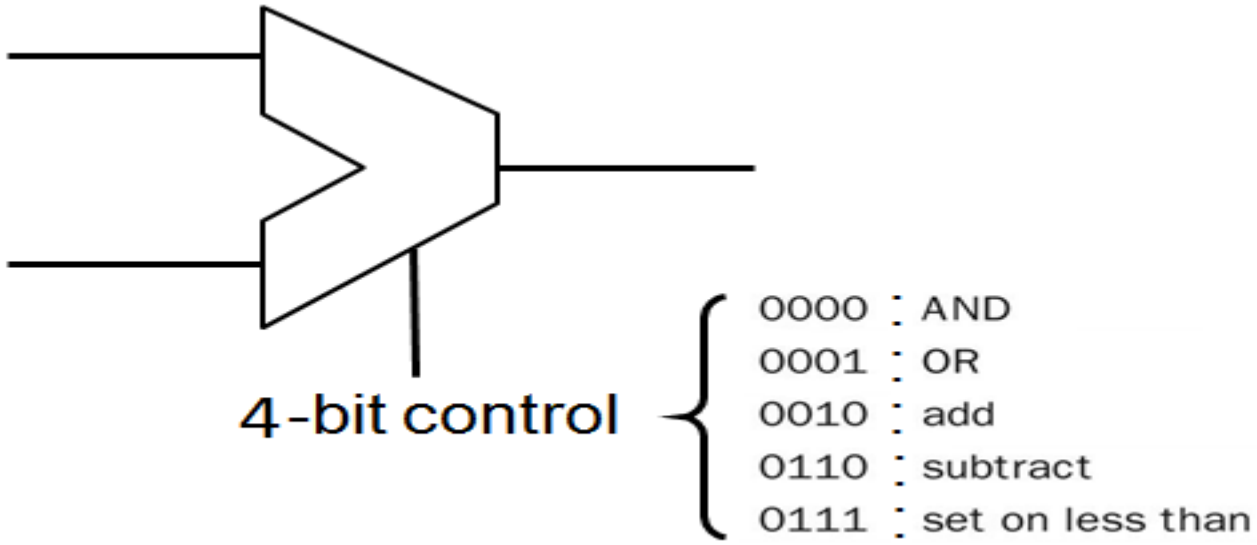
- Dependendo da instrução sendo executada, diferentes sinais alimentam o componente.
- Assim, dependendo da instrução, devemos decidir qual entrada será selecionada
- Isto é feito pelo Multiplexador (MUX)





Unidade de Controle

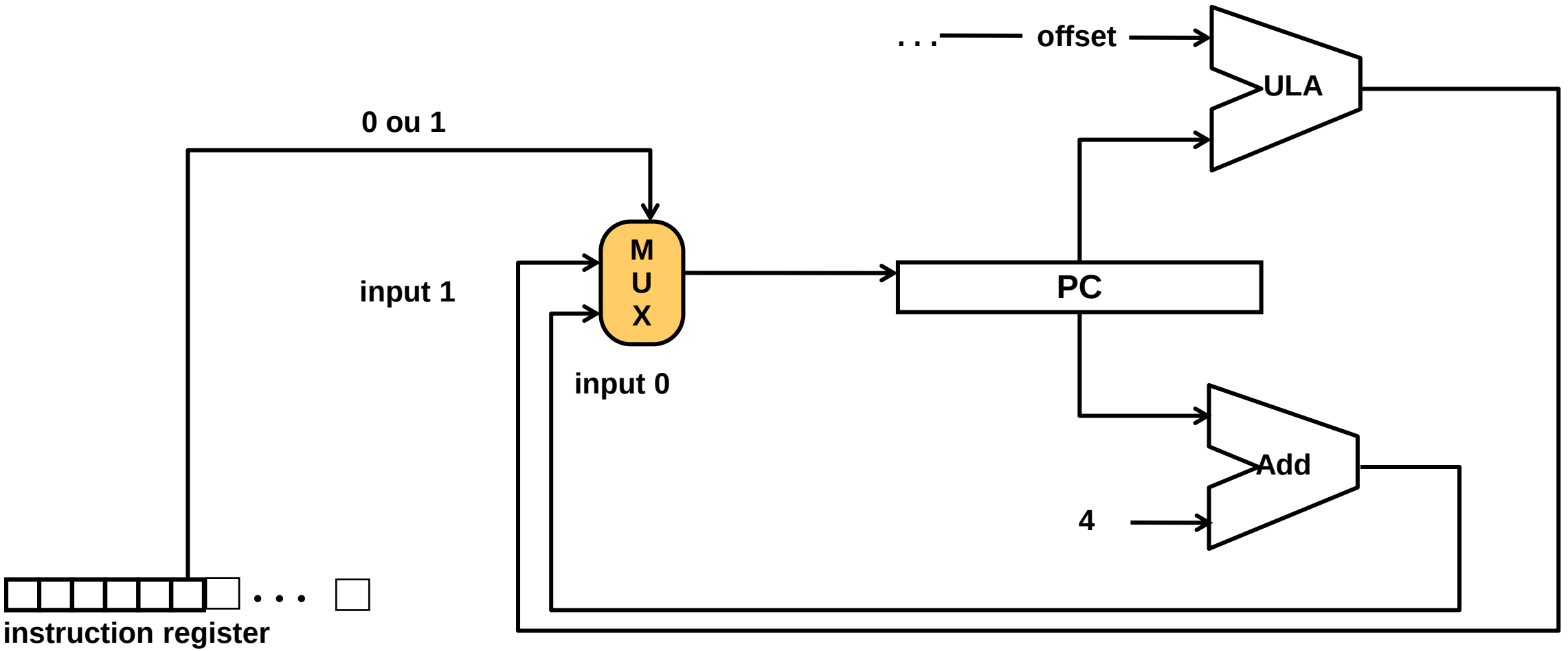
- Controladores
  - A ULA possui uma série de bits de controle p/ determinar o que ela deve fazer



Unidade de Controle

- Exemplo
  - Imagine que a instrução branch tem um opcode do tipo XXXXX1, e todos os demais opcodes são do tipo XXXXX0
  - Assim, a unidade de controle pode decidir se o valor do PC será atualizado c/ o valor "default" = (PC+4), ou da saída da ULA

Unidade de Controle

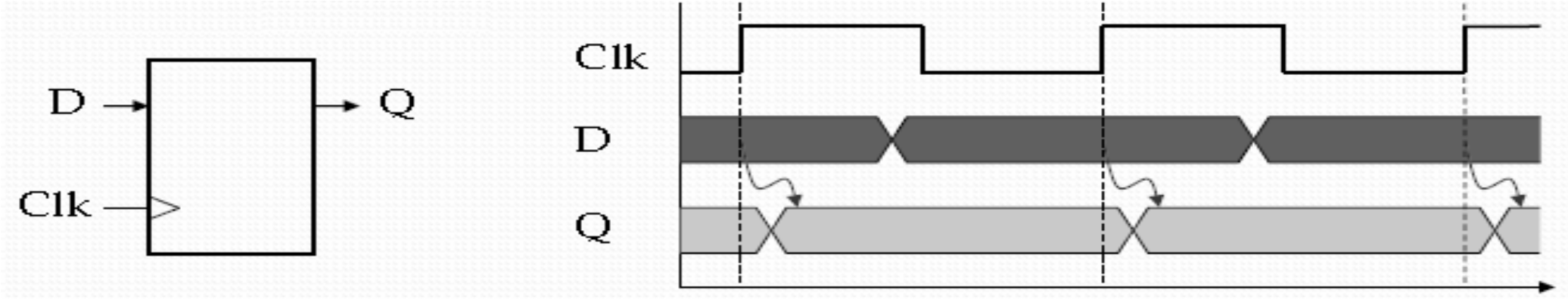


Circuitos Lógicos Digitais

- Circuito combinacional: as saídas dependem apenas dos níveis lógicos atuais das entradas.
  - Dada a mesma entrada, sempre produzem a mesma saída
  - Não possuem armazenamento interno, estado (memória)
- Circuito sequencial: as saídas dependem dos níveis lógicos das entradas e das informações armazenadas na memória interna (estado).

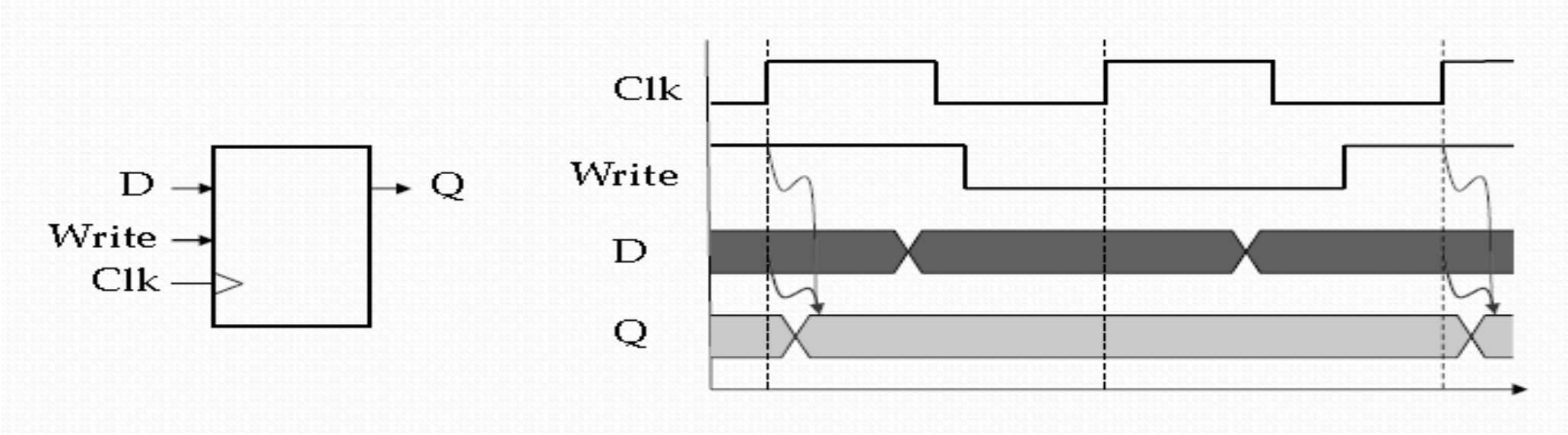
Circuitos Lógicos Digitais

- Unidade de armazenamento:
  - Sensível à borda: as saídas são atualizadas quando o clock muda de 0 para 1.

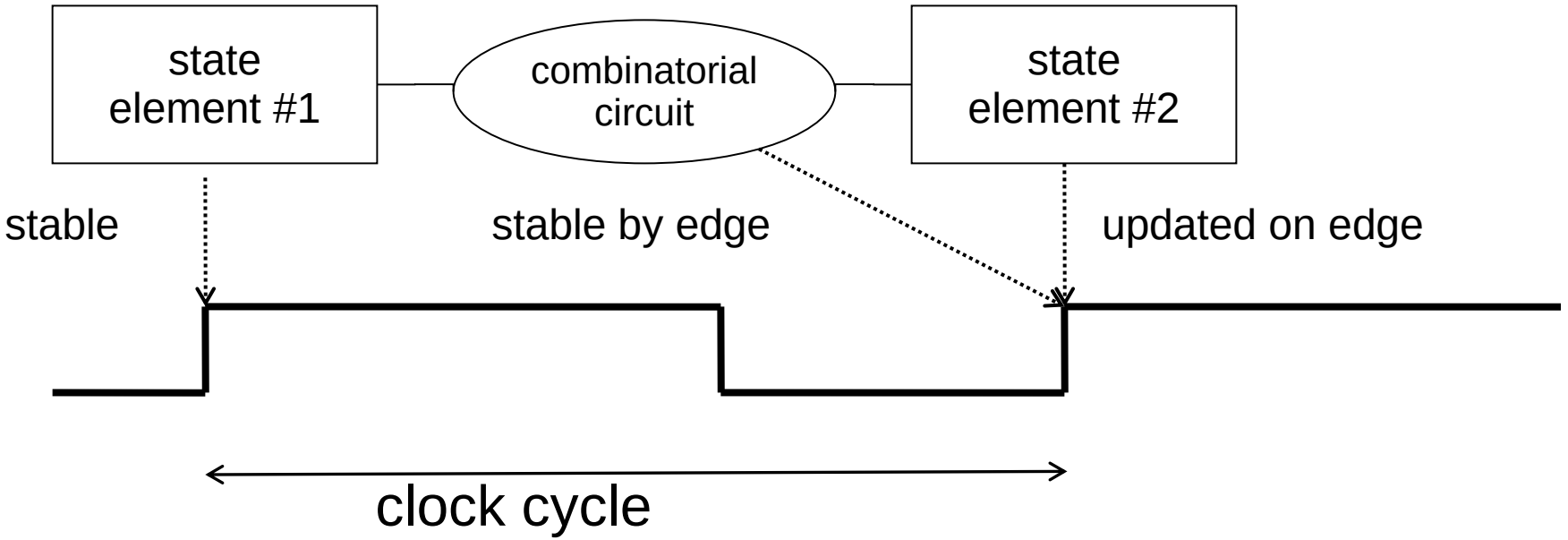


Circuitos Lógicos Digitais

- Registrador com entrada de controle:
  - Atualiza o valor armazenado apenas na subida de relógio quando a entrada de controle (Write) está ativa (nesse caso, no nível ALTO).



Clock



- Podemos usar o clock para controlar a leitura de um dado, seu uso em um circuito combinacional, e gravar a saída desse circuito
- O maior atraso determina o período do relógio.

Clock

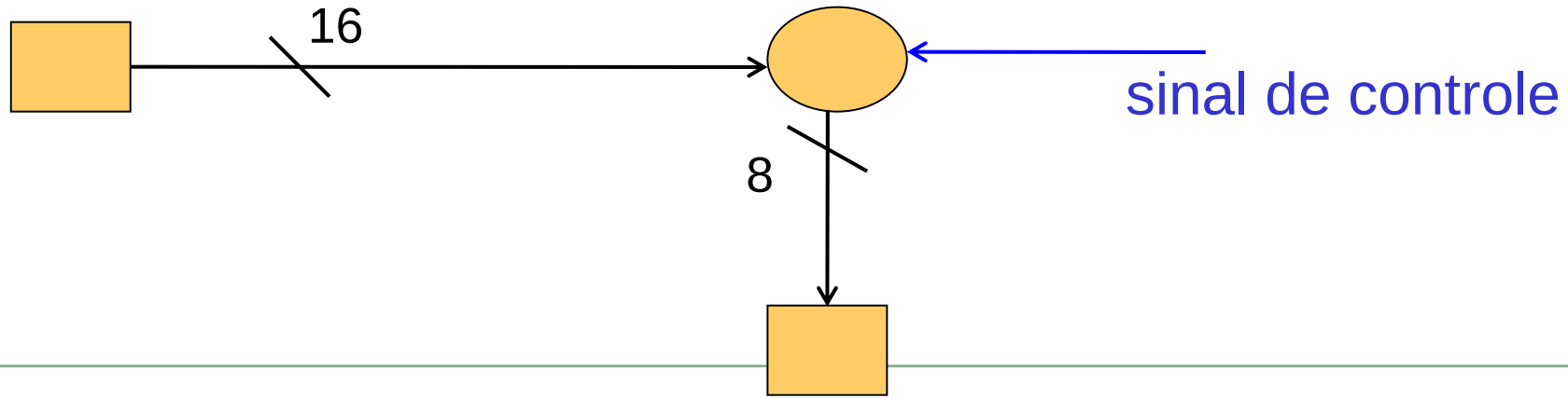
- Leitura e Escrita no mesmo Ciclo
  - Esse esquema permite que um mesmo elemento de estado possa ser lido e escrito no mesmo ciclo, sem que ocorram "condições de corrida".
    - Ex: leituras ocorrem na subida de borda, e escritas na descida.

Clock

- Opção: implementação monociclo (uniciclo)
  - Toda instrução inicia sua execução em uma borda ativa do sinal de relógio e termina na próxima borda.
  - Vantagem: simples de entender.
  - Desvantagem: o ciclo de relógio precisa ser alongado o suficiente para acomodar a instrução mais longa.

Barramentos

- A maioria dos elementos combinacionais e sequenciais possuem vários bits de entrada (ex: 32-bit inputs)
- O termo "barramento" (bus) refere-se a um fio composto por vários bits em paralelo.
- Indicamos a largura do barramento como se segue:

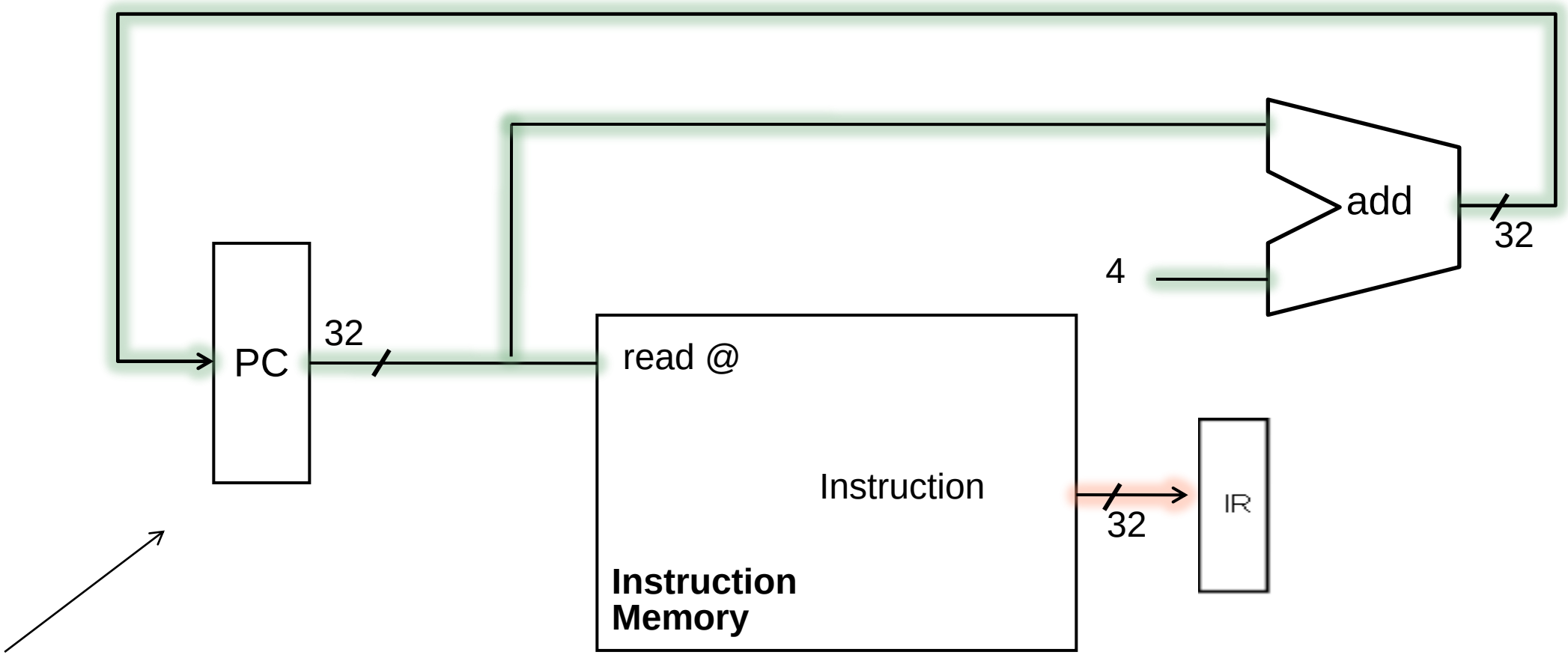


Construindo um Processador Simplificado

- 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

Construindo um Processador Simplificado

- Busca de Instrução (Fetch)



O PC é atualizado em 1 ciclo de clock

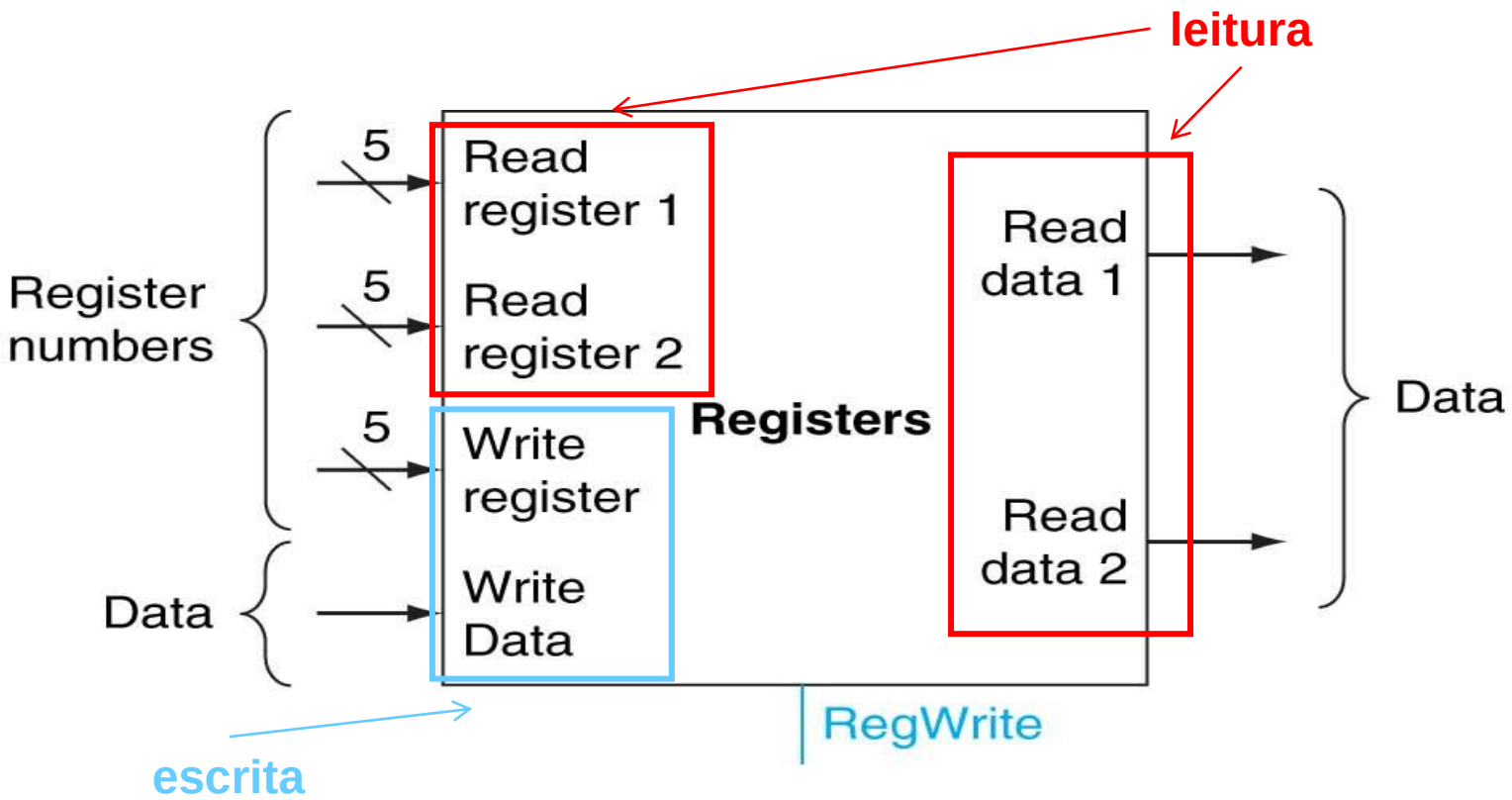
Construindo um Processador Simplificado

- Instruções do Tipo-R
  - Possuem 3 registradores como operandos:
    - 2 registradores para leitura (Rs1, Rs2)
    - 1 registrador para escrita (Rd)
  - Ex: add t1, t1, t2

- Banco de Registradores
  - Leitura:
    - Endereços dos registradores a serem lidos
    - Saída de dados para os conteúdos lidos
  - Escrita
    - Endereço do registrador a ser escrito
    - Entrada de dados a serem escritos

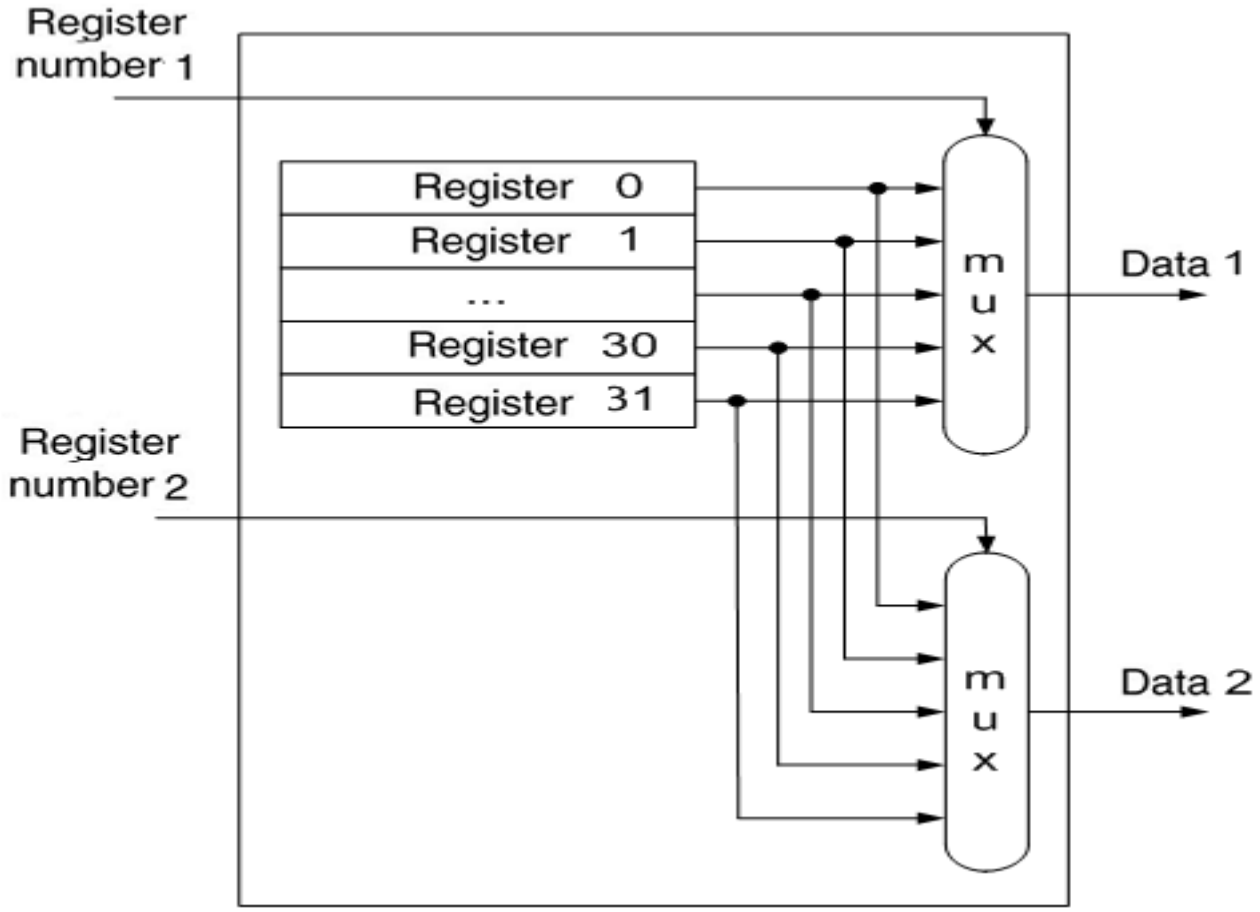
Construindo um Processador Simplificado

- Banco de Registradores
  - Cada registrador é representado por um código de 5-bits, que é extraído da instrução de 32 bits.



Construindo um Processador Simplificado

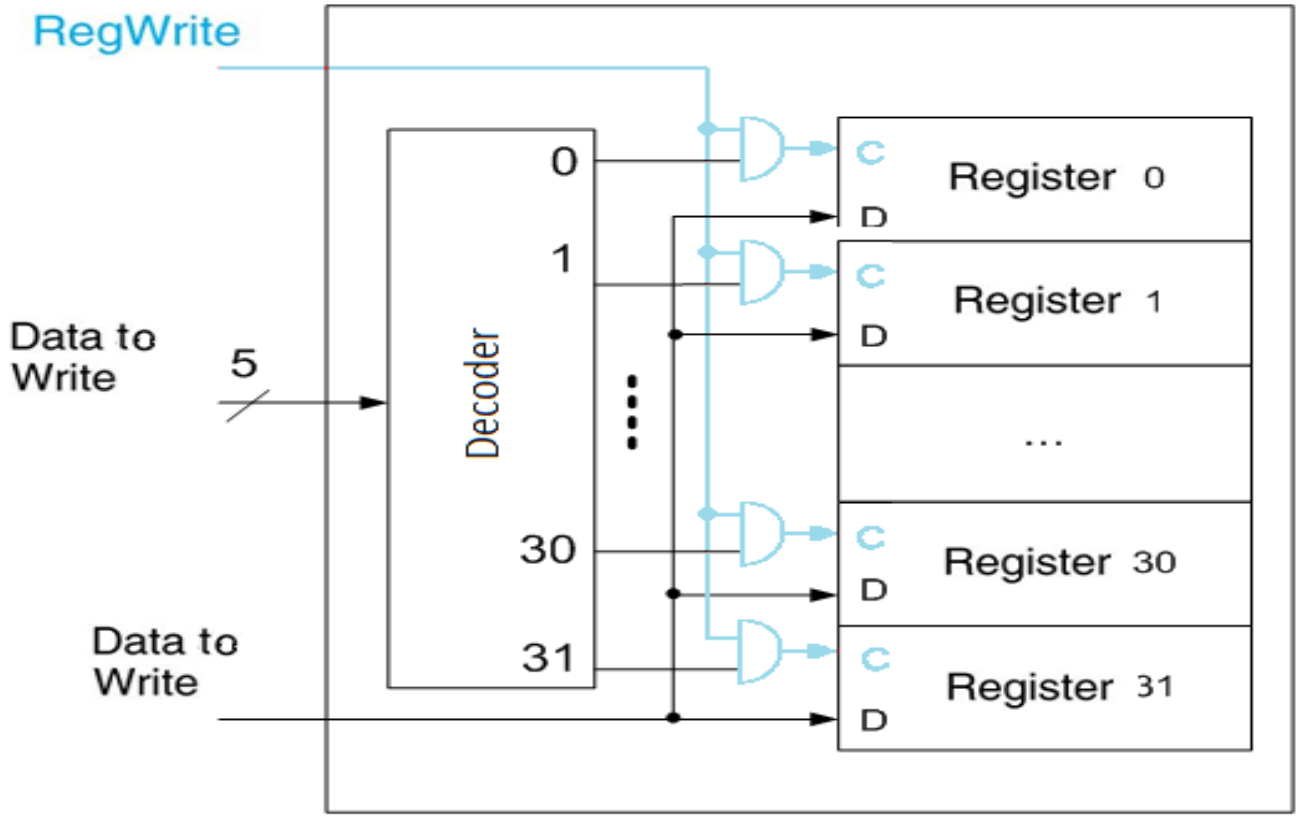
- Banco de Registradores
  - Diagrama do circuito de leitura do banco de registradores



Construindo um Processador Simplificado

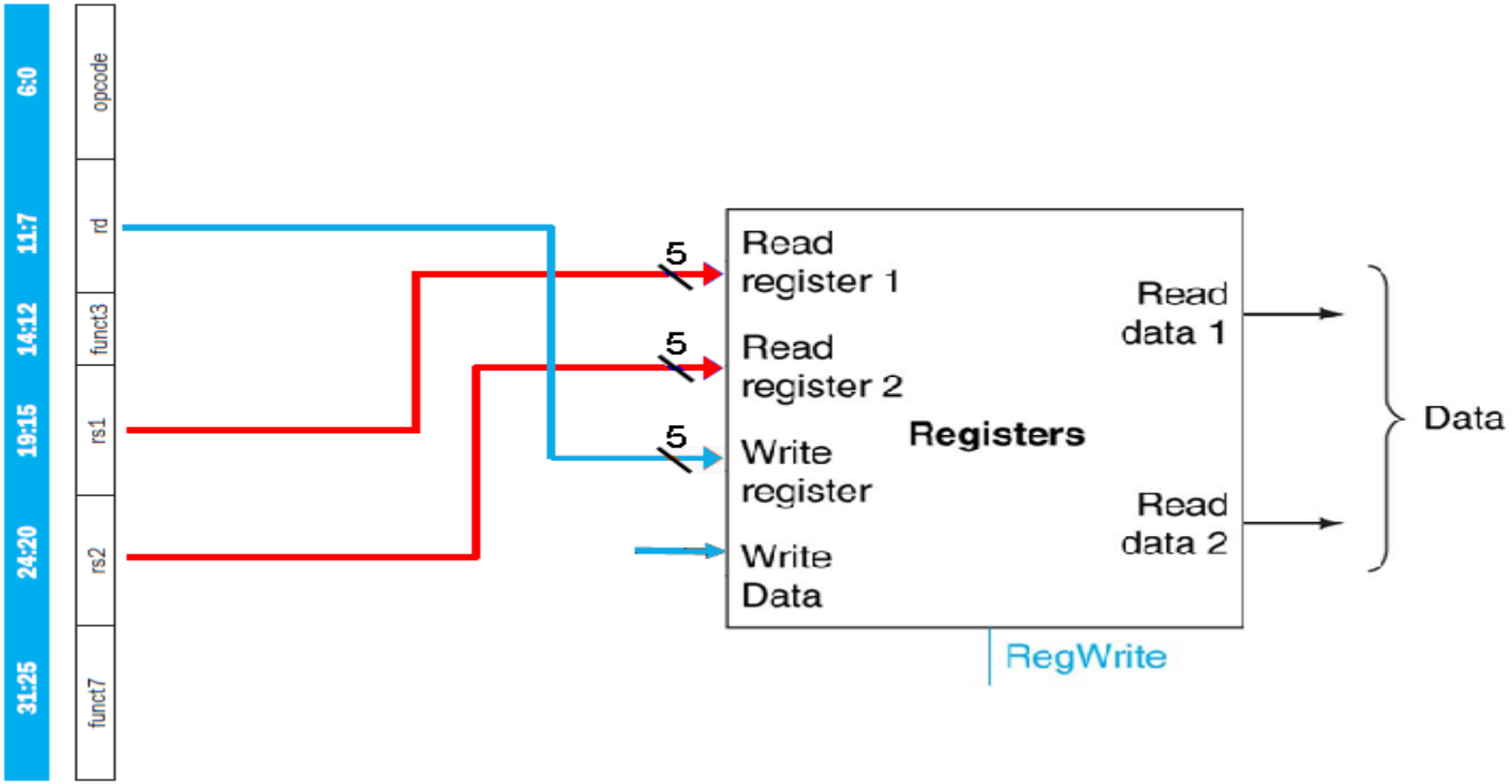
- Banco de Registradores
  - Diagrama do circuito de escrita no banco de registradores

sinal de controle para determinar quando escrever



Construindo um Processador Simplificado

- Instruções do Tipo-R

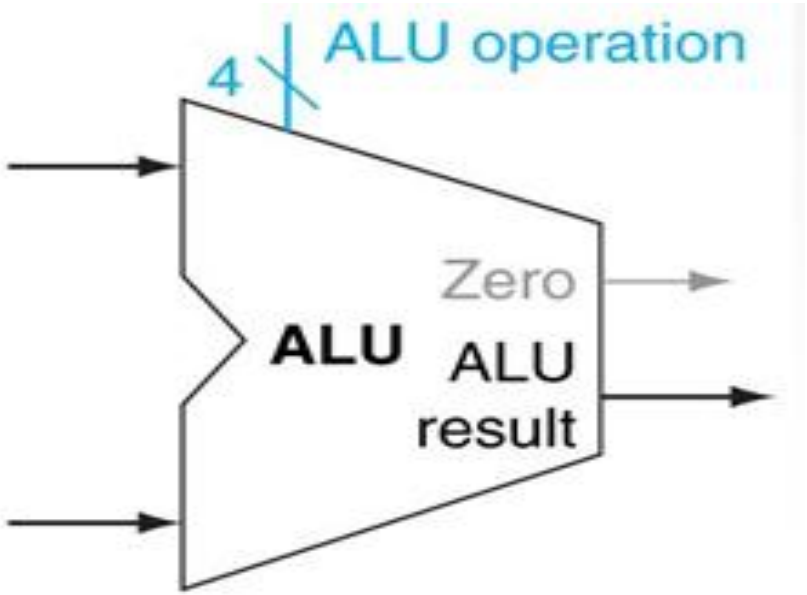


Construindo um Processador Simplificado

- Instruções do Tipo-R
  - Precisamos de um componente que forneça os dados a serem gravados no registrador indicado para a escrita
  - Esse componente frequentemente é a ULA
  - Precisamos de um sinal de escrita para disparar a gravação do registrador de saída na próxima mudança de borda do ciclo de clock

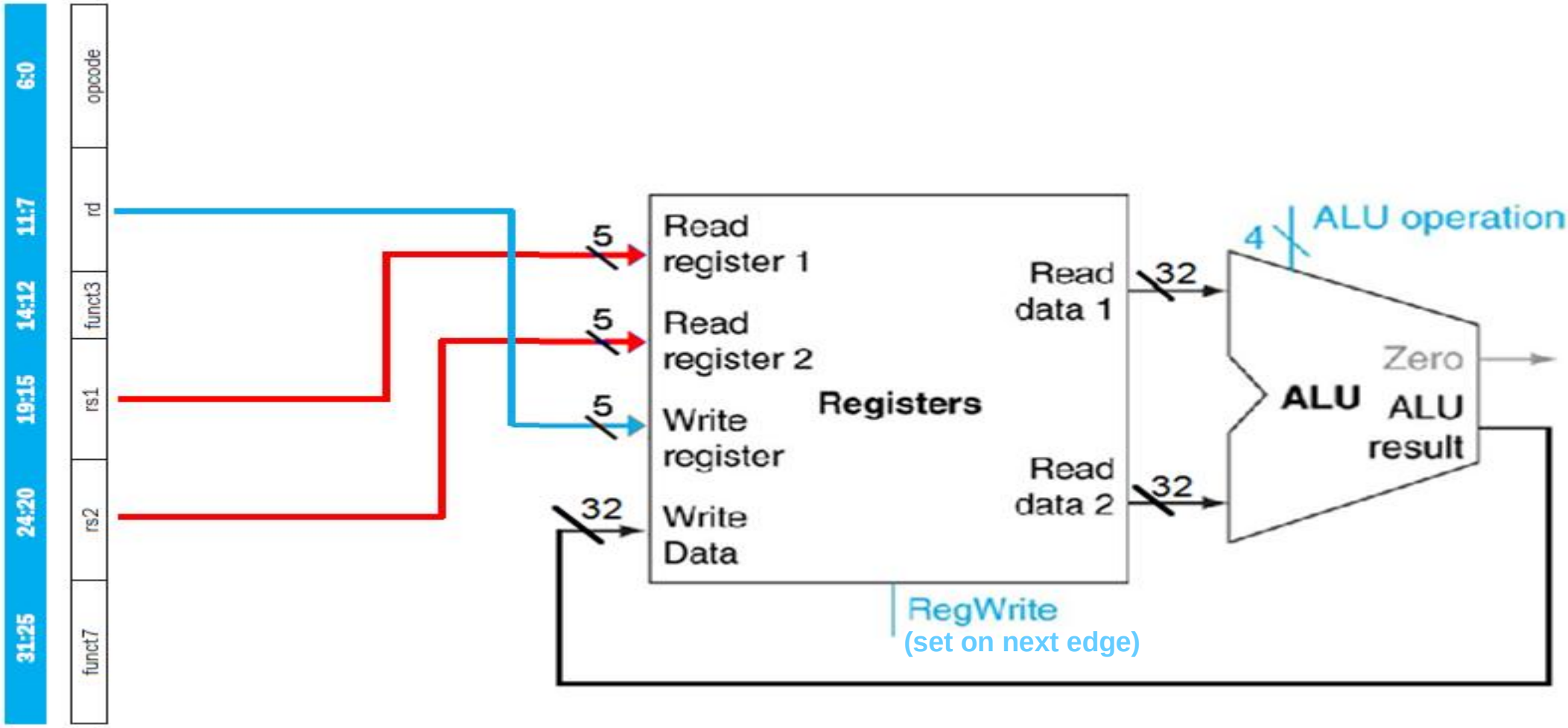
Construindo um Processador Simplificado

- Instruções do Tipo-R



Construindo um Processador Simplificado

- Instruções do Tipo-R

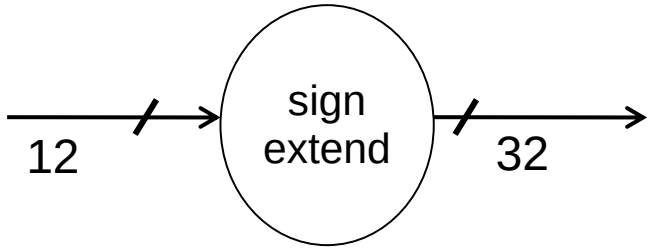


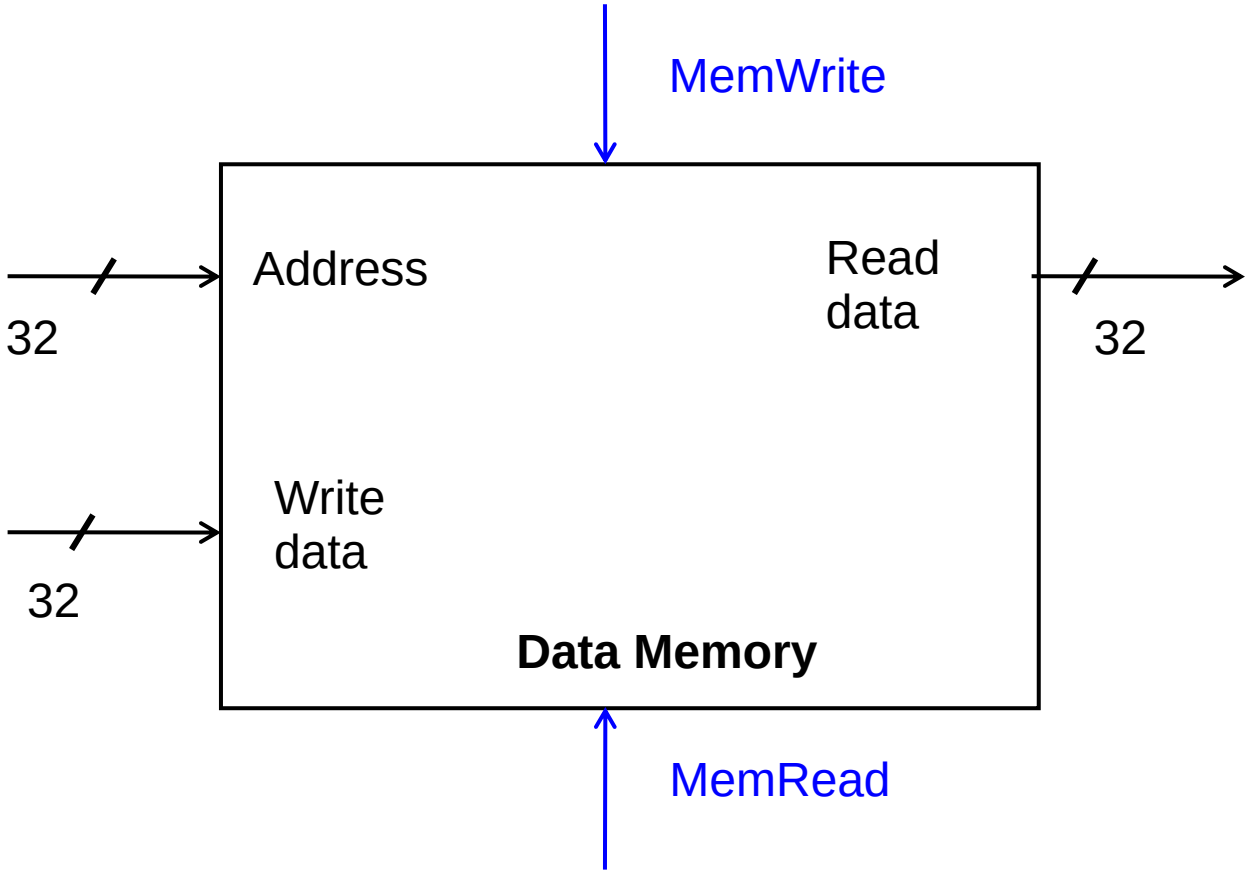
Construindo um Processador Simplificado

- Instrução Load (Tipo I)
  - O endereço de memória é computado somando o campo offset de 12 bits ao registrador de entrada (base) de 32 bits (rs1)
  - O offset possui 12-bits, mas os endereços de memória são de 32 bits
    - Assim, o offset deve ser estendido (sign-extended) p/ 32 bits, antes de ser somado ao registrador de entrada
  - A memória de dados tem sinais de controle p/ leitura e escrita
  - Essa instrução utiliza o banco de registradores (ler e armazenar dados) e a ULA para o cálculo do endereço

Construindo um Processador Simplificado

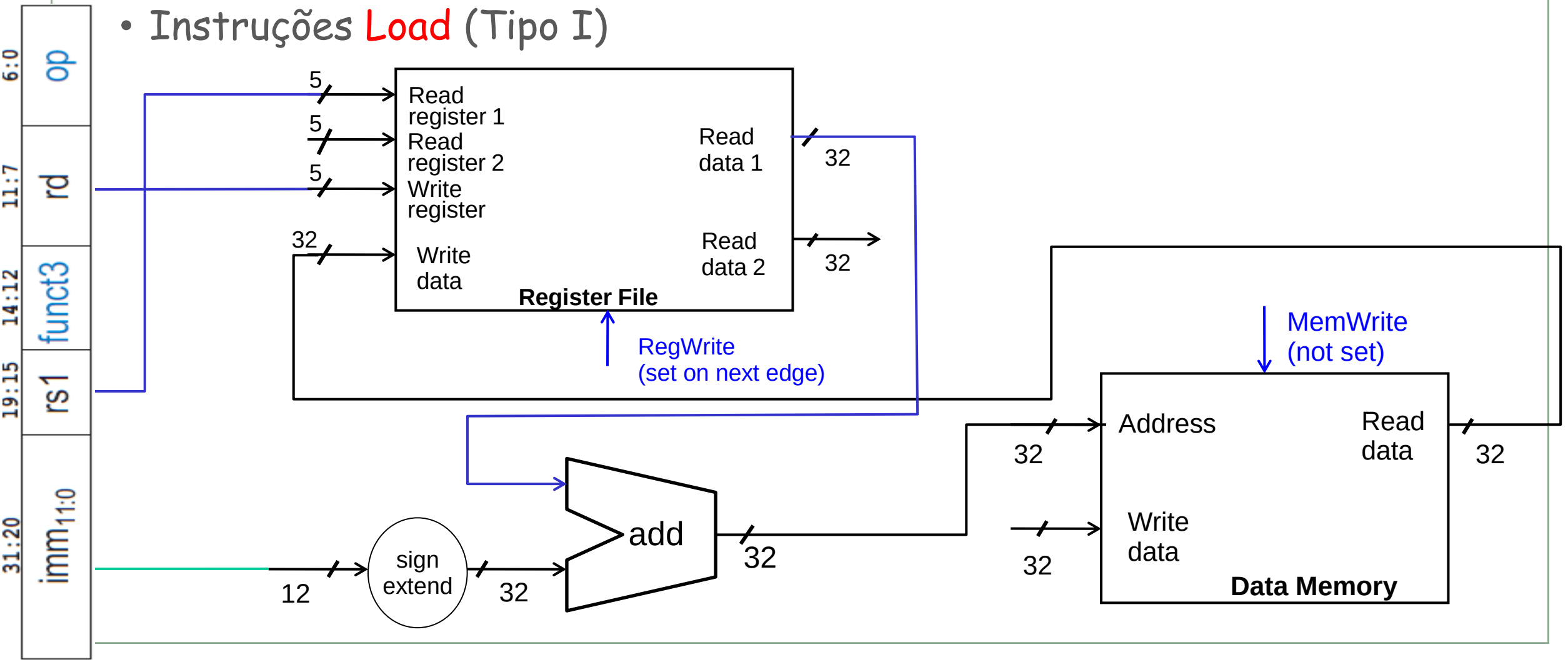
- Instruções Load (Tipo I)





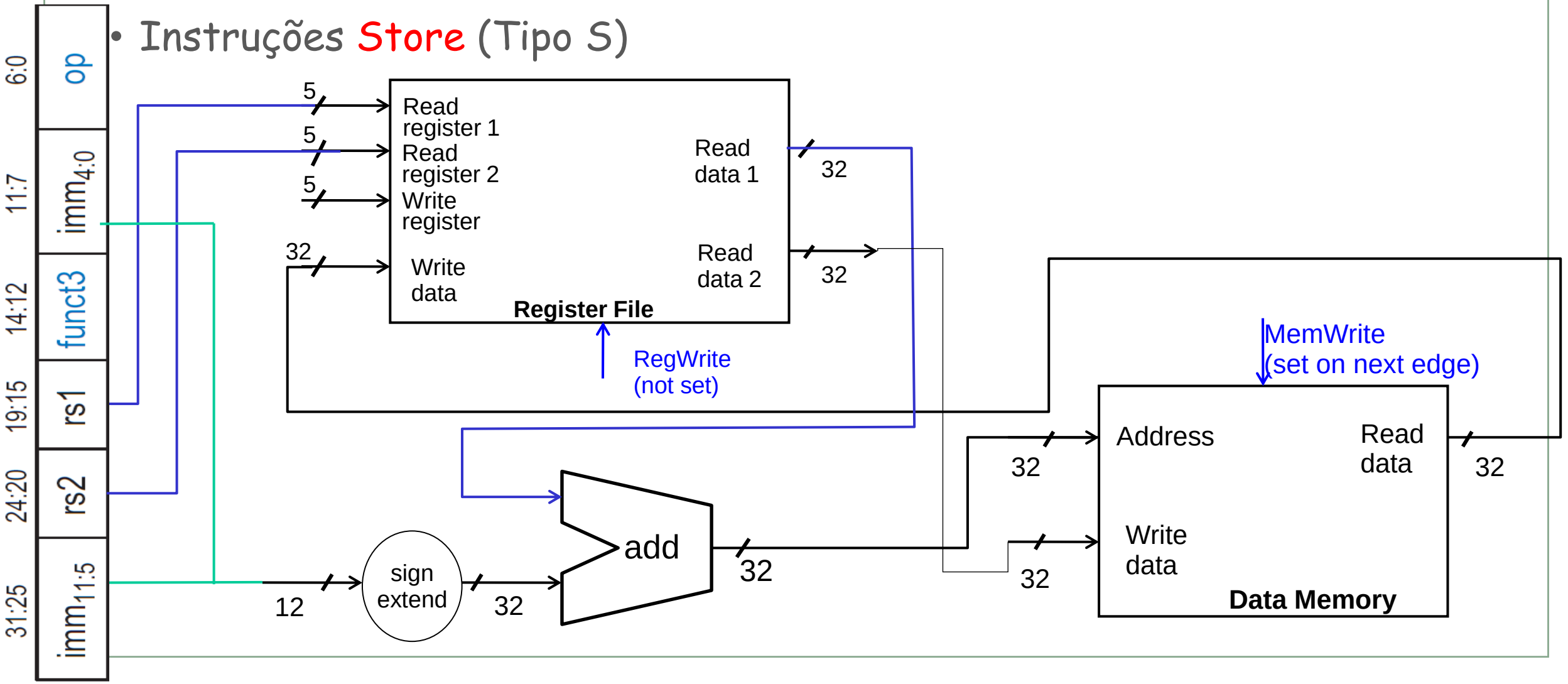
Construindo um Processador Simplificado

- Instruções Load (Tipo I)



Construindo um Processador Simplificado

- Instruções **Store** (Tipo S)

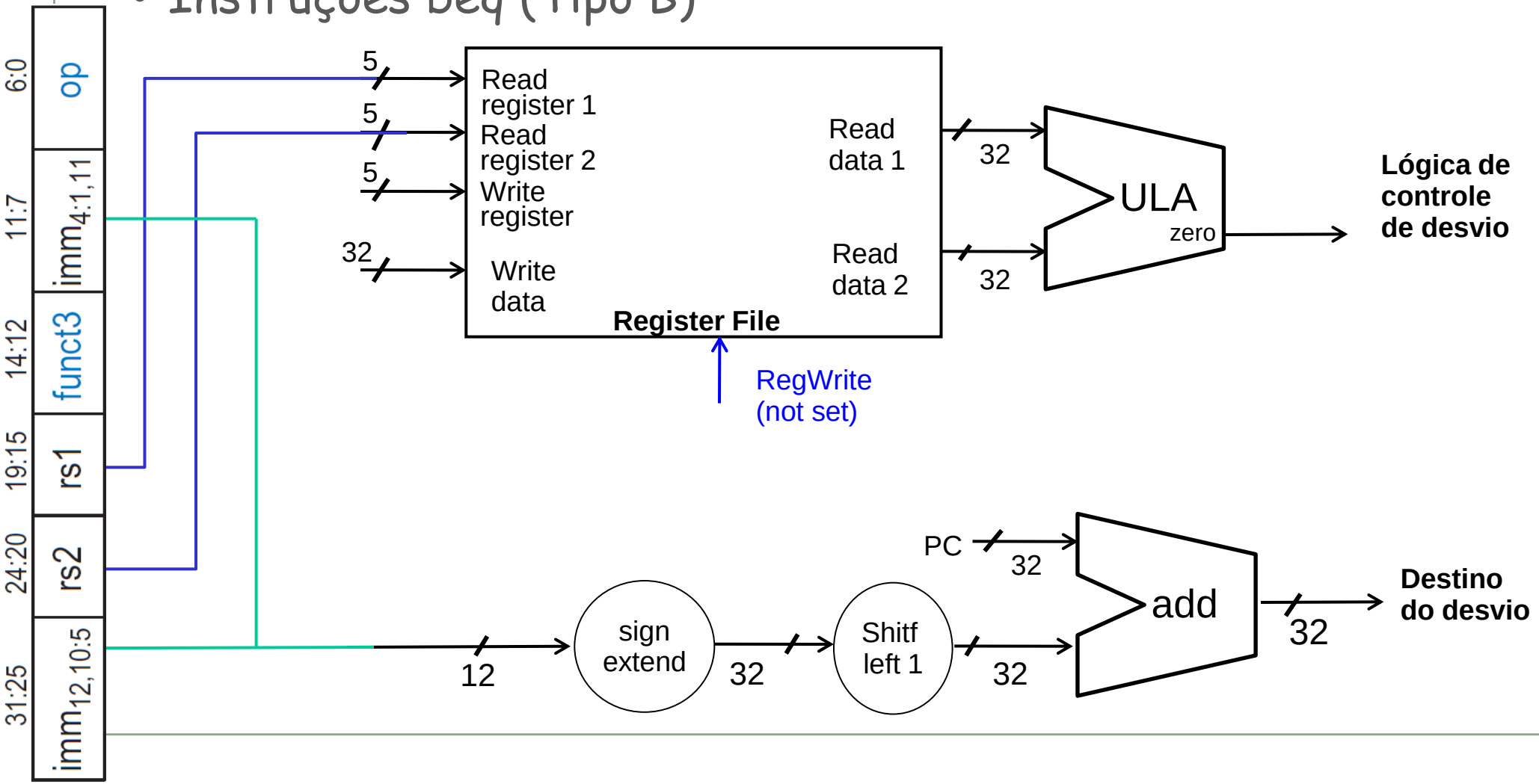


Construindo um Processador Simplificado

- Instruções Branch (Desvio Condicional)
  - Possuem três operandos:
    - Dois registradores a serem comparados
    - Offset de 16 bits para calcular o endereço de destino
    - Ex.: beq t1, t2, offset (Se t1 = t2, desviar p/ a instrução no endereço (PC + offset))
  - Como o deslocamento é constante, não se utiliza um circuito shifter, mas sim um deslocamento de sinais que acrescenta 0 à extremidade direita
  - A comparação é implementada como uma subtração
    - Para operandos iguais, a ALU sinaliza resultado ZERO
  - A depender do resultado da comparação, o novo valor de PC pode ser (PC + 4) ou (PC + 4 + (offset << 1))

Construindo um Processador Simplificado

- Instruções beq (Tipo B)

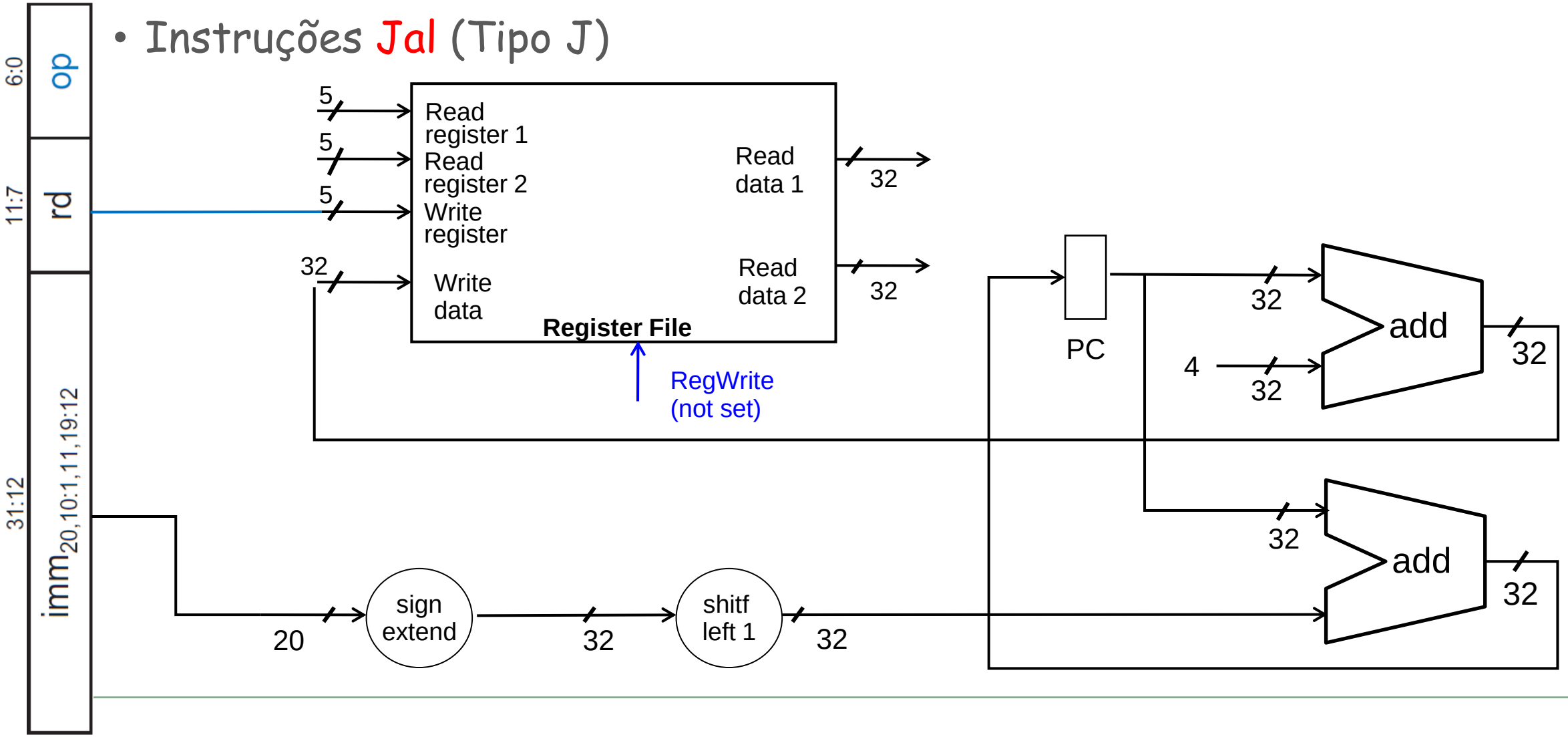


Construindo um Processador Simplificado

- Instruções Jal (Desvio Incondicional)
  - É preciso adicionar ao PC os 20 bits do campo imediato da instrução deslocados um dígitos à esquerda (i.e., concatenado com 0) e estendida 32 bits (sinalizado).
  - O PC adicionado de 4 (próxima instrução após jal) deve ser salvo no registrado indicado pelo campo rd da instrução

Construindo um Processador Simplificado

- Instruções Jal (Tipo J)



Construindo um Processador Simplificado

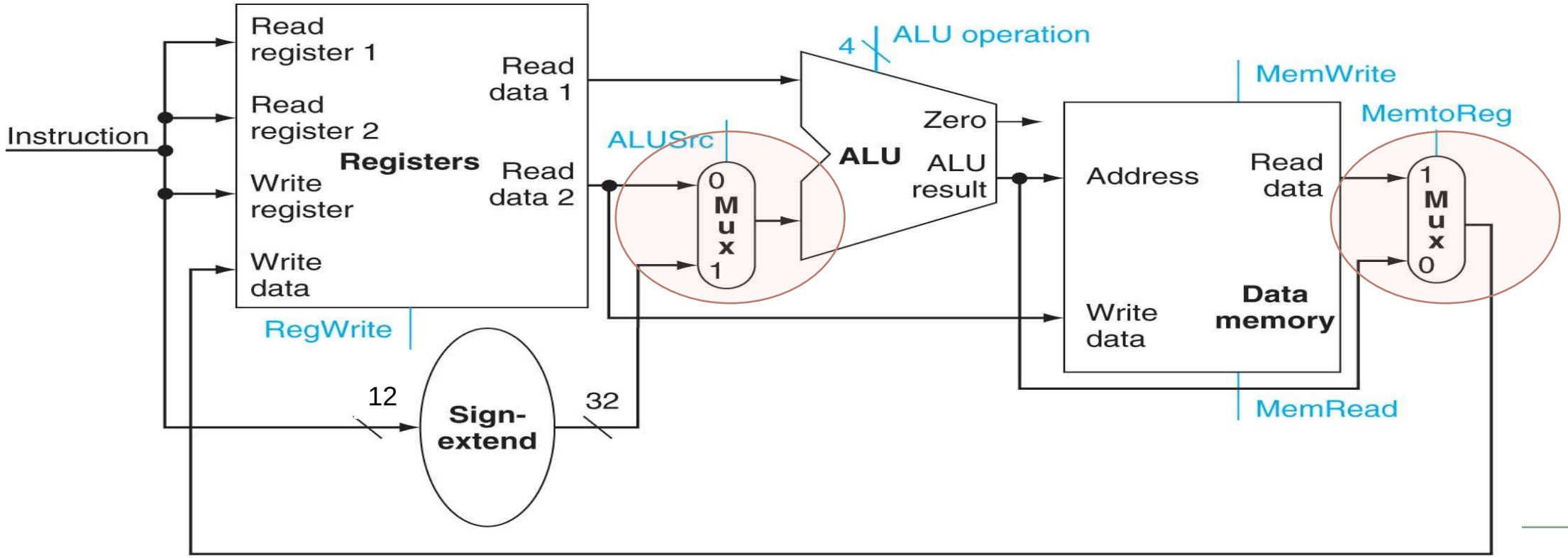
- Combinando Todas as Instruções
  - Os projetos anteriores podem ser combinados para formar um único datapath
  - Em sua forma mais simples, ele será capaz de executar qualquer uma das instruções em um único ciclo.
  - Nesse caso, todo elemento do datapath é usado no máximo uma vez a cada ciclo de clock
  - Nenhuma duplicação de hardware é necessária (talvez apenas alguns somadores)

Construindo um Processador Simplificado

- Combinando Todas as Instruções
  - Inicialmente, vamos combinar os datapaths p/ instruções R-Type (ALU) e de acesso à memória (Load / Store)
  - Para isso, adicionamos um multiplexador para escolher o datapath p/ a ALU, ou p/ a Memória (determinando ainda os sinais de controle correspondentes)

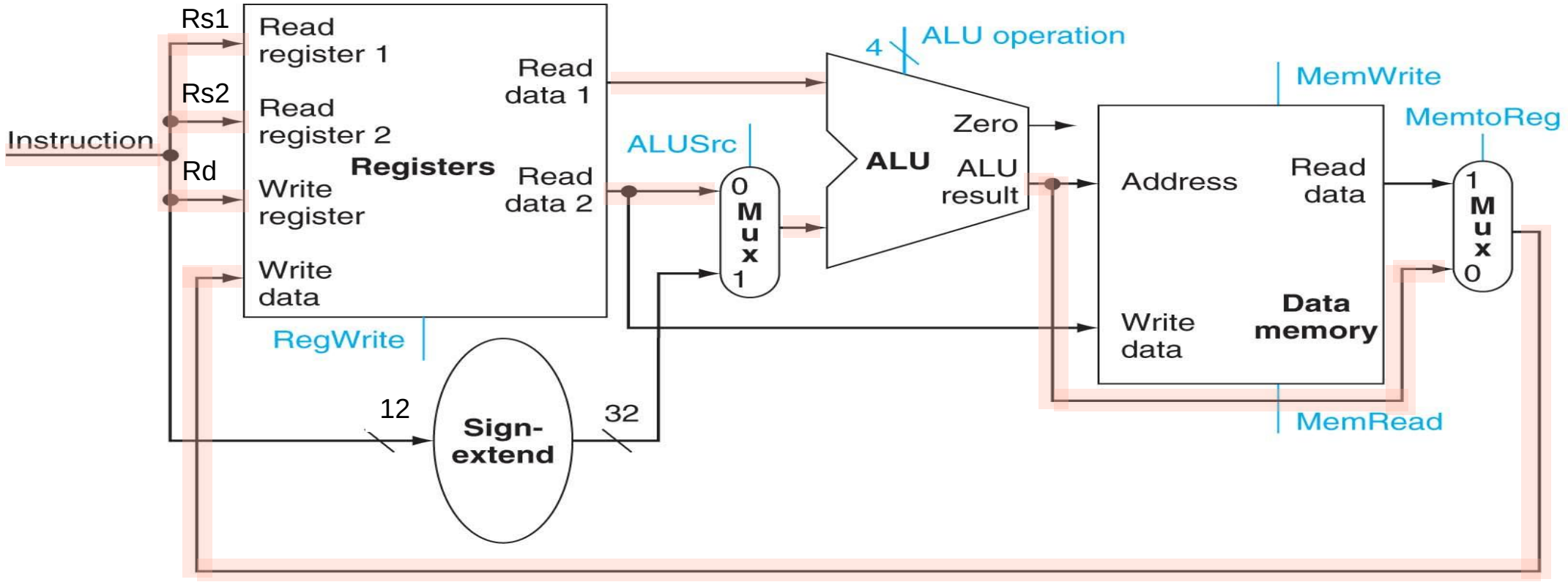
Construindo um Processador Simplificado

- Combinando Todas as Instruções
  - R-Type (ALU) e de acesso à memória (Load/Store)



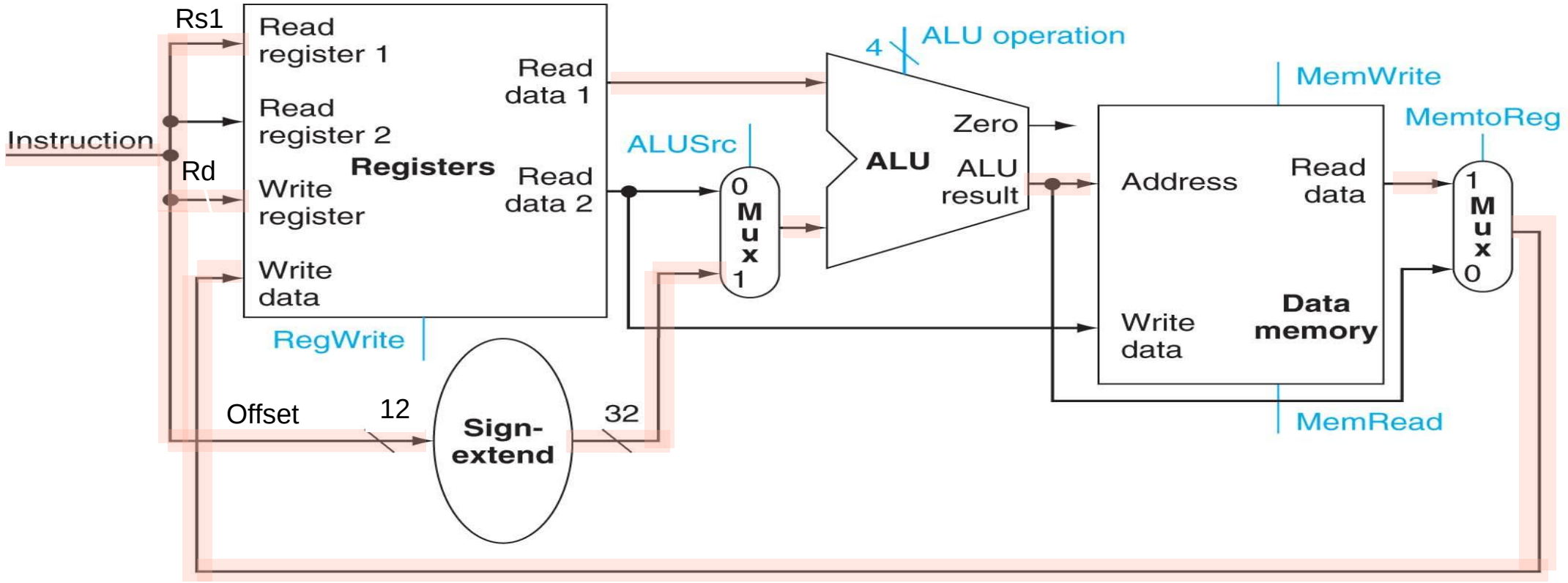
Construindo um Processador Simplificado

- Combinando Todas as Instruções
  - add, t0, s1, s2



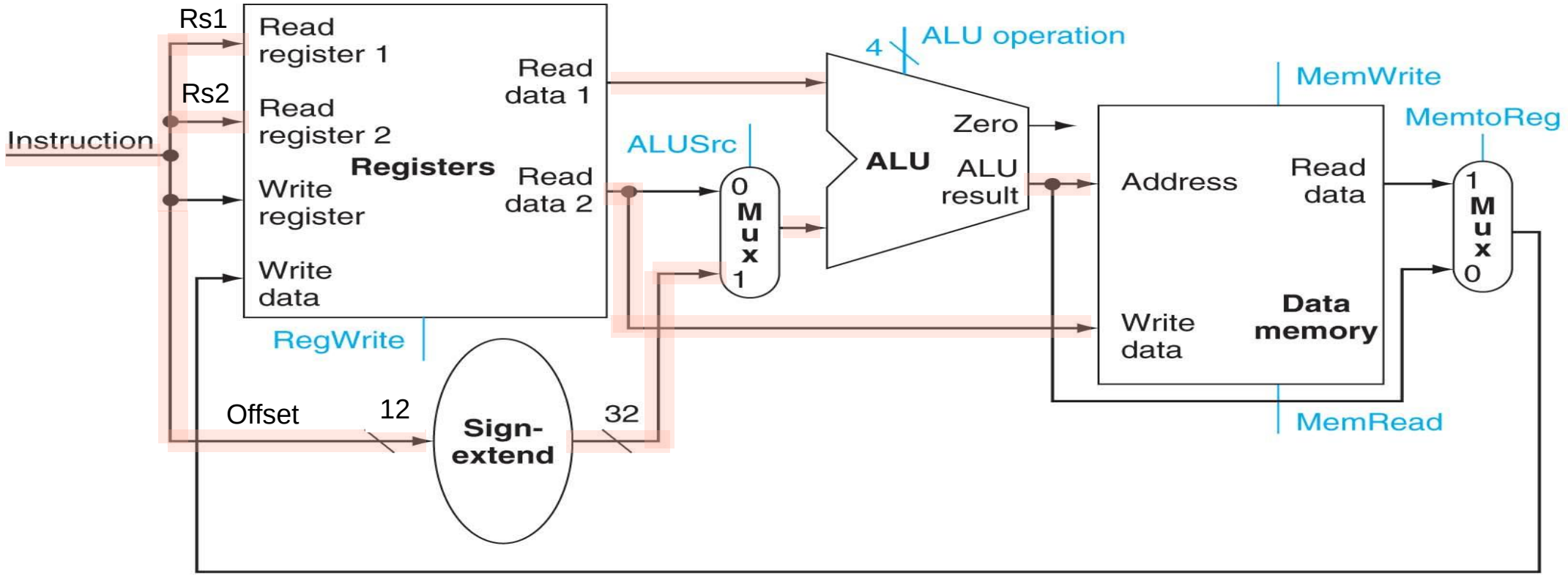
Construindo um Processador Simplificado

- Combinando Todas as Instruções
  - lw, t0, 16(s2)



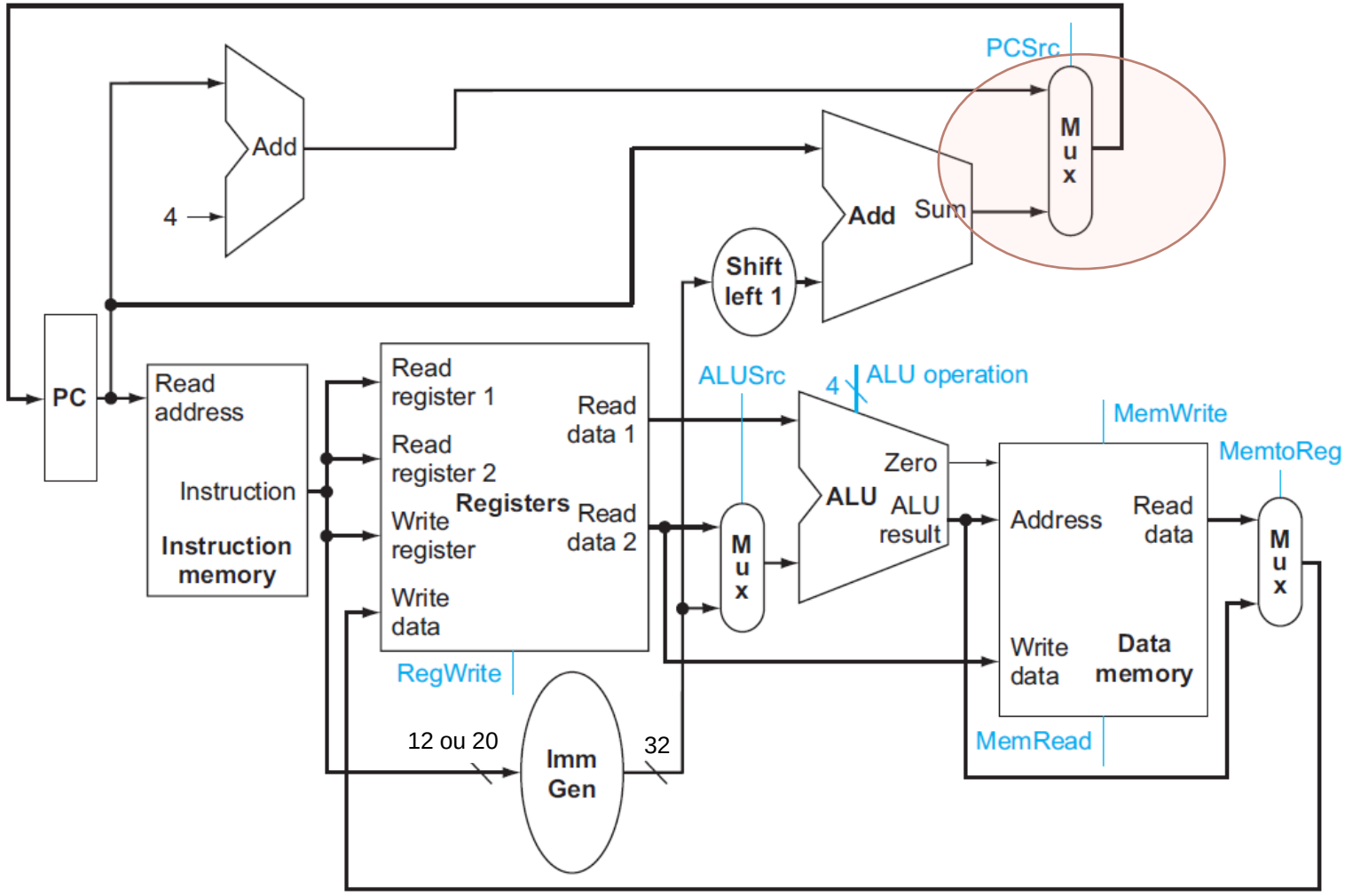
Construindo um Processador Simplificado

- Combinando Todas as Instruções
  - sw, t0, 16(s2)



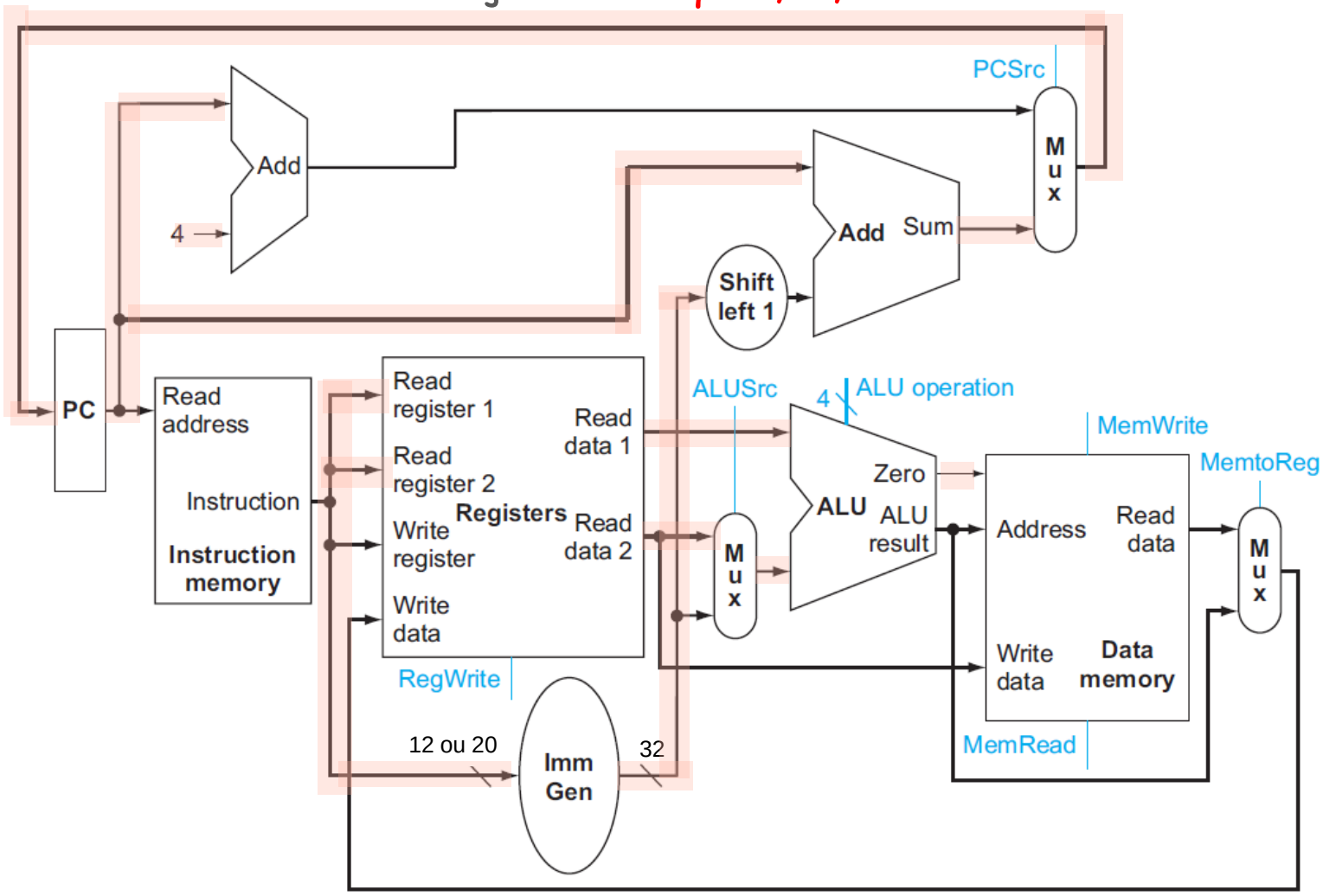
Construindo um Processador Simplificado

- Combinando Todas as Instruções: R-Type, L/S e beq



Construindo um Processador Simplificado

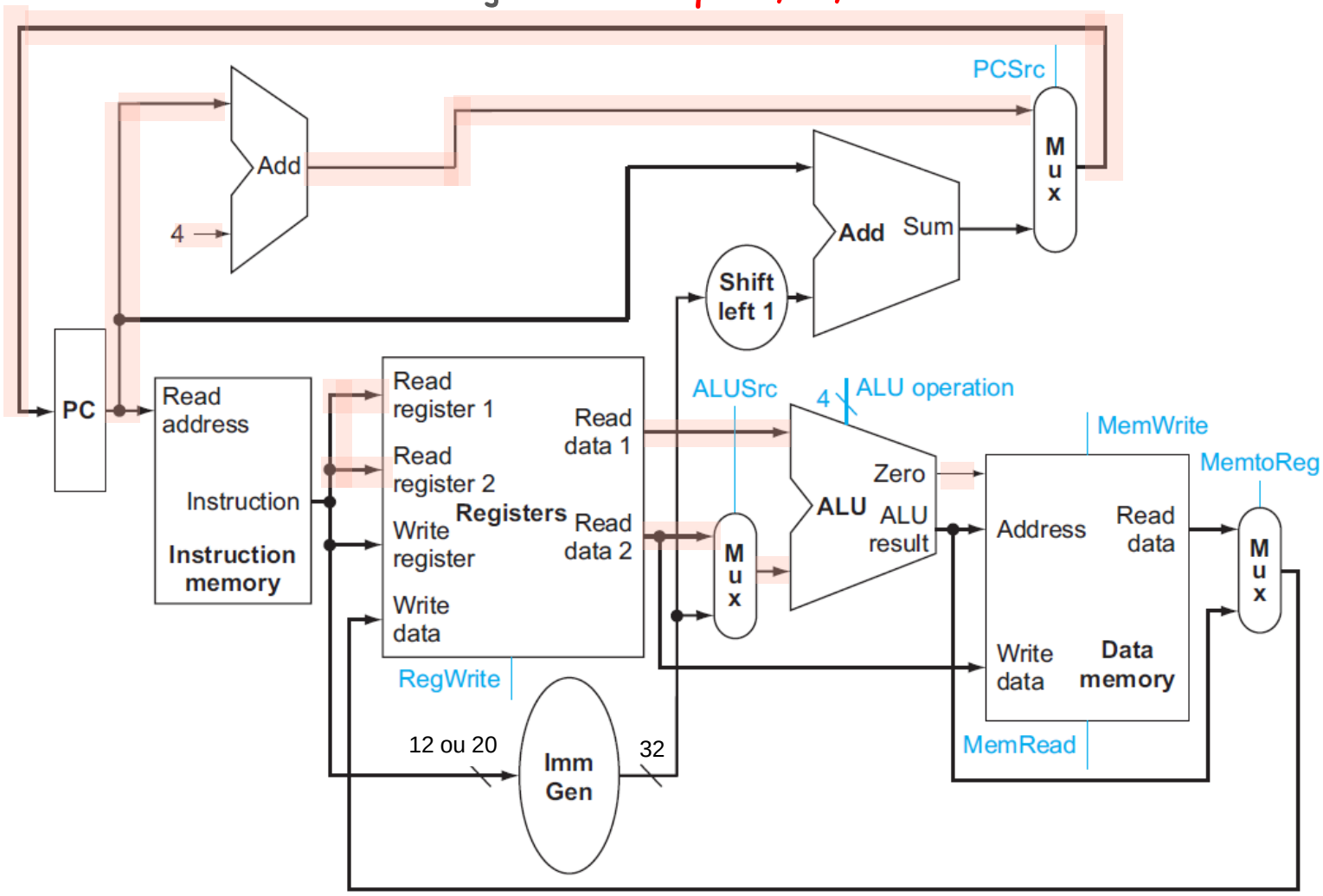
- Combinando Todas as Instruções: beq t0,t1,label



t0 = t1

Construindo um Processador Simplificado

- Combinando Todas as Instruções: beq t0,t1,label



t0 != t1

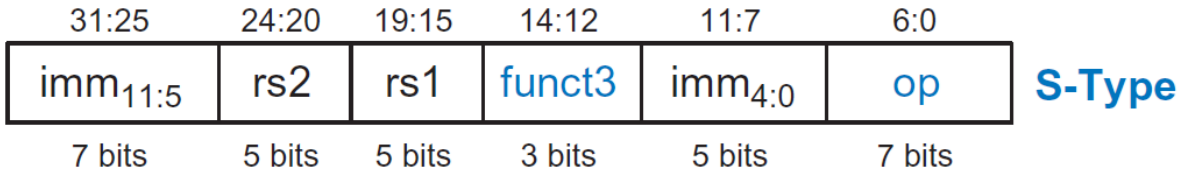
Construindo um Processador Simplificado

- Relembrando...

7 bits	5 bits	5 bits	3 bits	5 bits	7 bits	
funct7	rs2	rs1	funct3	rd	op	**R-Type**
$imm_{11:0}$		rs1	funct3	rd	op	**I-Type**
$imm_{11:5}$	rs2	rs1	funct3	$imm_{4:0}$	op	**S-Type**
$imm_{12,10:5}$	rs2	rs1	funct3	$imm_{4:1,11}$	op	**B-Type**
$imm_{31:12}$				rd	op	**U-Type**
$imm_{20,10:1,11,19:12}$				rd	op	**J-Type**
20 bits				5 bits	7 bits	

Construindo um Processador Simplificado

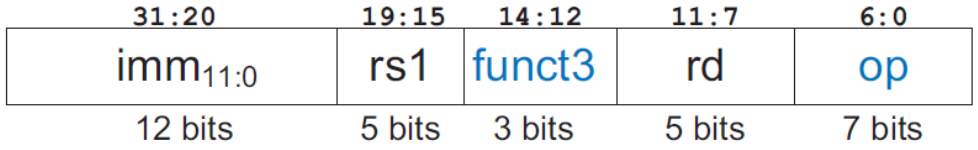
- Relembrando...
  - Para sw, campo rs2 (bits 24-20) designam registrador cujo conteúdo será escrito na memória de dados e rs1 (bits 19-15) o registrador contendo o endereço base da posição de memória a ser escrita



S-Type

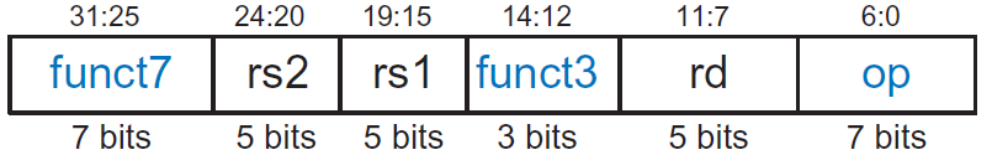
  - Para lw, rd (bits 11-7) designa o registrador que será carregado com valor lido da memória de dados e o registrador contendo o endereço base da posição de memória a ser lida está no campo rs1 (bits 19-15)

I-Type



  - Para instruções tipo R, o endereço do registrador a ser escrito está no campo rd (bits 11-7)

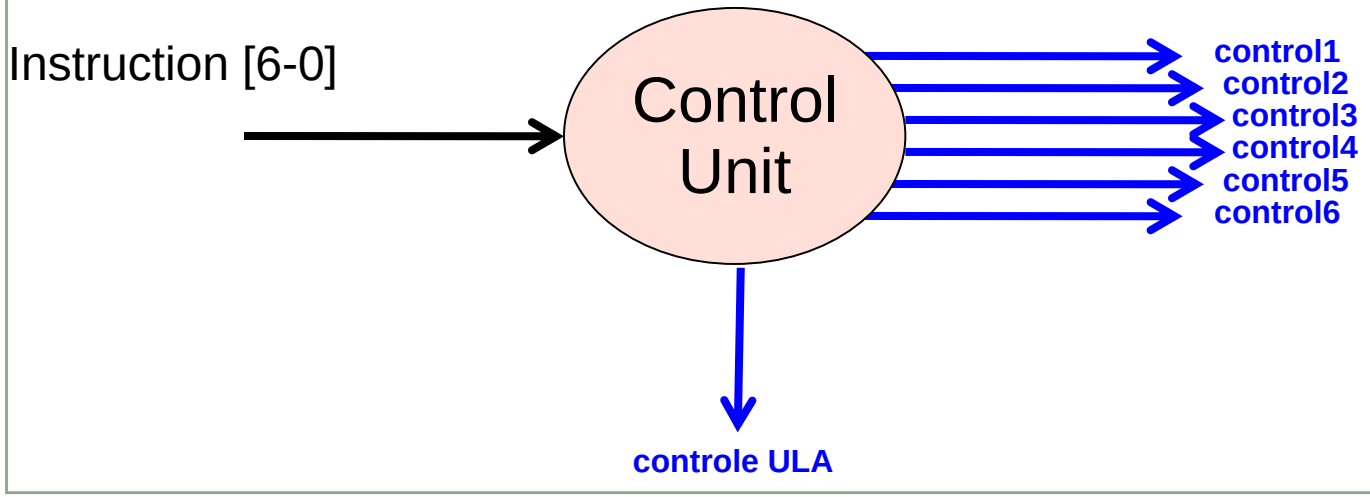
R-Type



Unidade de Controle

Unidade de Controle

- A Unidade de Controle deve através dos sinais de controle:
  - Selecionar as operações para executar (ULA, leitura/escrita, etc.)
  - Controlar o fluxo de dados (entradas para o multiplexador)
- Baseia-se no opcode da instrução para determinar o valor de uma série de bits de controle
- Sua operação é definida por uma tabela verdade



opcode	c1	c2	c3	c4	c5	c6
000000	1	X	0	X	0	1
000001	1	1	X	0	0	0
. . . . . .						
111110	1	x	0	x	x	0
111111	0	0	x	0	1	x

X = não importa

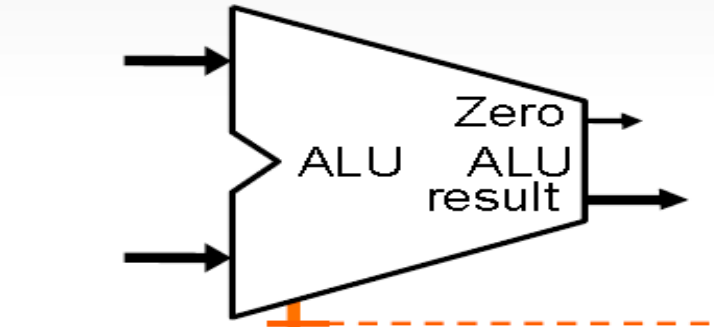
Unidade de Controle

- Controle da ULA
  - Instruções do tipo R: há cinco ações possíveis para a ALU.
    - Todas possuem opcode igual a 51.
    - Diferem no campo funct3 e funct7 (10 bits).
  - Instruções load/store: usam a ALU para uma adição (soma conteúdo do registrador base com offset para obter o endereço de memória)
  - Instrução branch: usa a ALU para uma subtração – que, por sua vez, implementa uma comparação.

Unidade de Controle

- Controle da ULA
  - Dois bits (ULAop) são usados para indicar se a operação esperada da ULA é uma soma (load/store - 00), subtração (branch - 01) ou se é determinada pelo campo funct (10).

$ALUControl_{2:0}$	Function
000	add
001	subtract
010	and
011	or
101	SLT

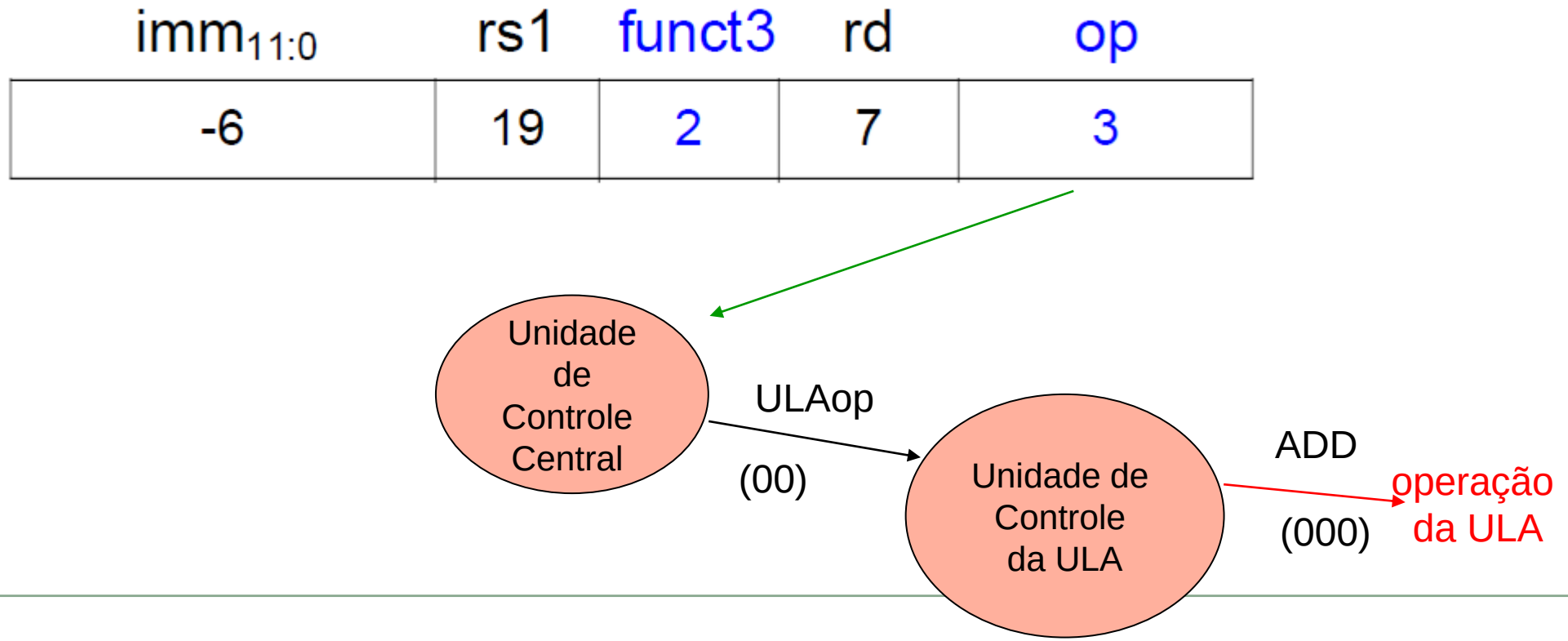


Instrução	opALU	Entrada de controle na ALU
Load/Store	00	add
Desvio	01	sub
Tipo R	10	funct

Unidade de Controle

- Controle da ULA
  - Ex 1.: Para a instrução load-word a operação da ULA só depende do opcode.

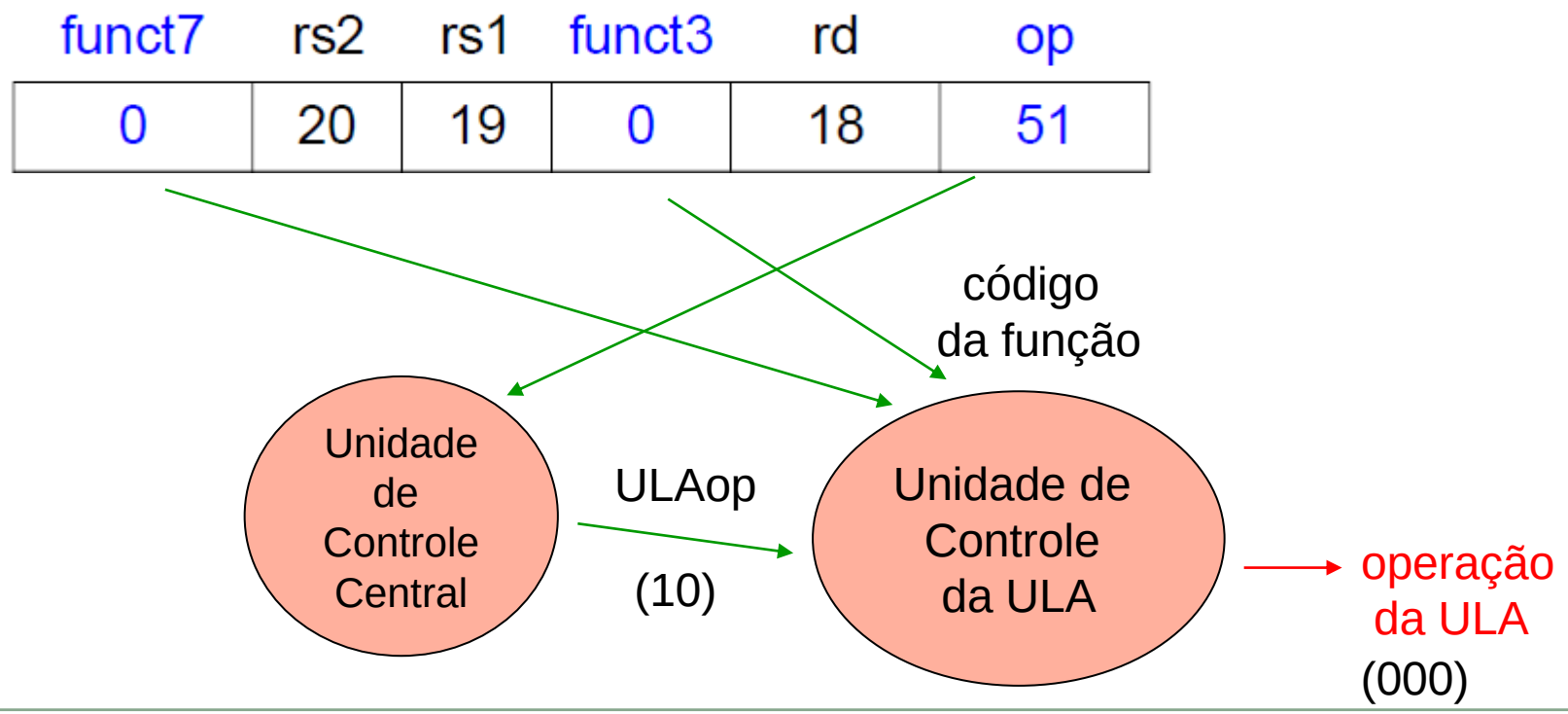
    lw t2, -6(s3)



Unidade de Controle

- Controle da ULA
  - Ex 2.: Para a instrução add a operação da ULA depende também do código da função

add s2, s3, s4



Unidade de Controle

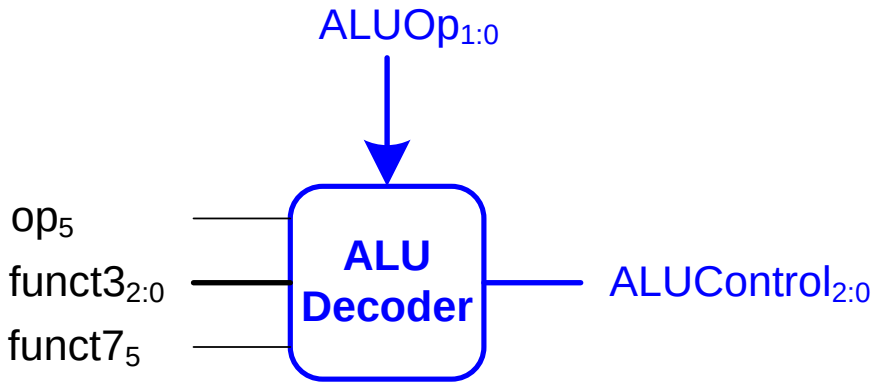
- Controle da ULA
  - ALUOp e o campo funct.

Table B.1 RV32I: RISC-V integer instructions

op	funct3	funct7	Type	Instruction	Description	Operation
0000011 (3)	000	–	I	`lb rd, imm(rs1)`	load byte	`rd = SignExt([Address]7:0)`
0000011 (3)	001	–	I	`lh rd, imm(rs1)`	load half	`rd = SignExt([Address]15:0)`
0000011 (3)	010	–	I	`lw rd, imm(rs1)`	load word	`rd = [Address]31:0`
0000011 (3)	100	–	I	`lbu rd, imm(rs1)`	load byte unsigned	`rd = ZeroExt([Address]7:0)`
0000011 (3)	101	–	I	`lhu rd, imm(rs1)`	load half unsigned	`rd = ZeroExt([Address]15:0)`
0010011 (19)	000	–	I	`addi rd, rs1, imm`	add immediate	`rd = rs1 + SignExt(imm)`
0110011 (51)	000	0000000	R	`add rd, rs1, rs2`	add	`rd = rs1 + rs2`
0110011 (51)	000	0100000	R	`sub rd, rs1, rs2`	sub	`rd = rs1 – rs2`
0110011 (51)	001	0000000	R	`sll rd, rs1, rs2`	shift left logical	`rd = rs1 << rs24:0`
0110011 (51)	010	0000000	R	`slt rd, rs1, rs2`	set less than	`rd = (rs1 < rs2)`
0110011 (51)	011	0000000	R	`sltu rd, rs1, rs2`	set less than unsigned	`rd = (rs1 < rs2)`
0110011 (51)	100	0000000	R	`xor rd, rs1, rs2`	xor	`rd = rs1 ^ rs2`

Unidade de Controle

- Controle da ULA



ALUOp	funct3	op_5 , $funct7_5$	Instruction	$ALUControl_{2:0}$
00	x	x	`lw, sw`	000 (add)
01	x	x	`beq`	001 (subtract)
10	000	00, 01, 10	`add`	000 (add)
	000	11	`sub`	001 (subtract)
	010	x	`slt`	101 (set less than)
	110	x	`or`	011 (or)
	111	x	`and`	010 (and)

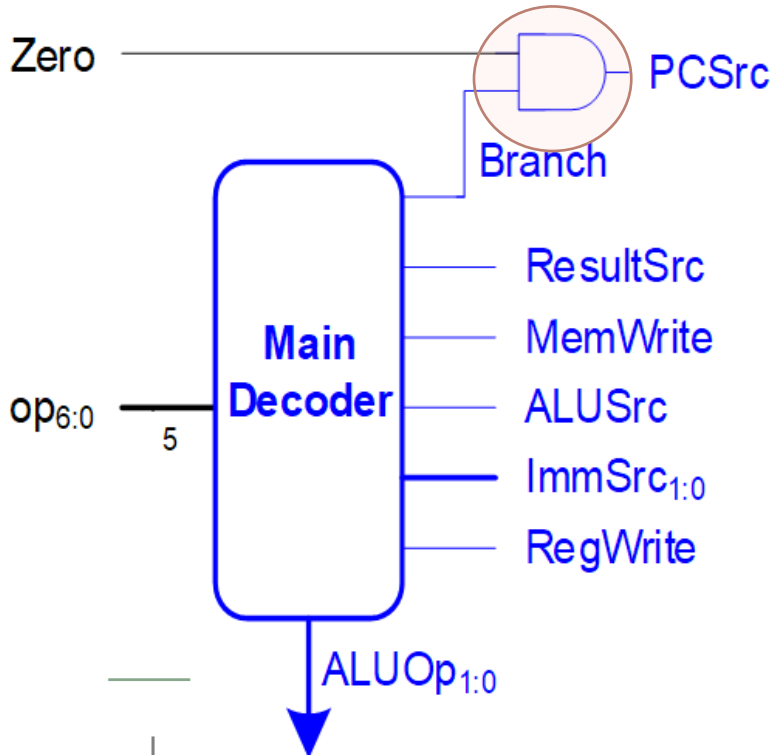
Unidade de Controle

- Controle da ULA

ALUOp	funct3	op_5 , $funct7_5$	$ALUControl_{2:0}$
00	xxx	xx	000
01	xxx	xx	001
10	000	00	000
10	000	01	000
10	000	10	000
10	000	11	001
10	010	xx	101
10	110	xx	011
10	111	xx	010

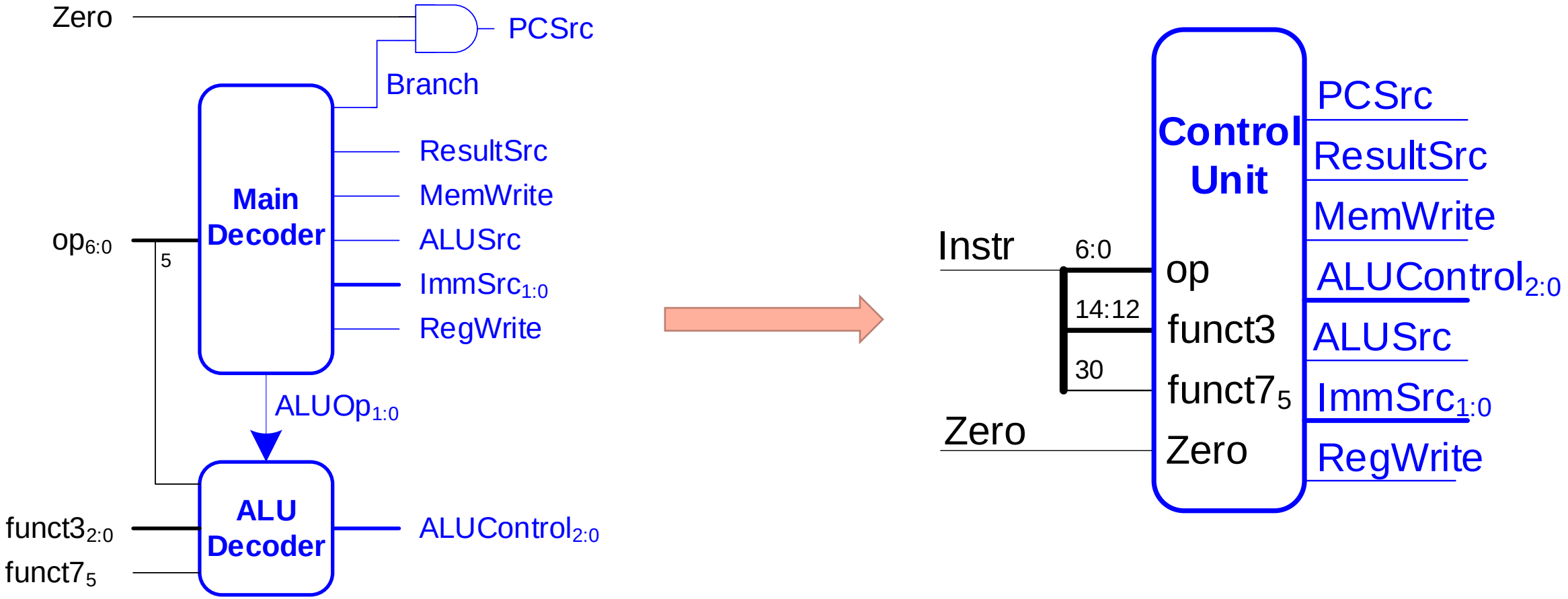
Unidade de Controle

- Linhas de Controle: valores
  - 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

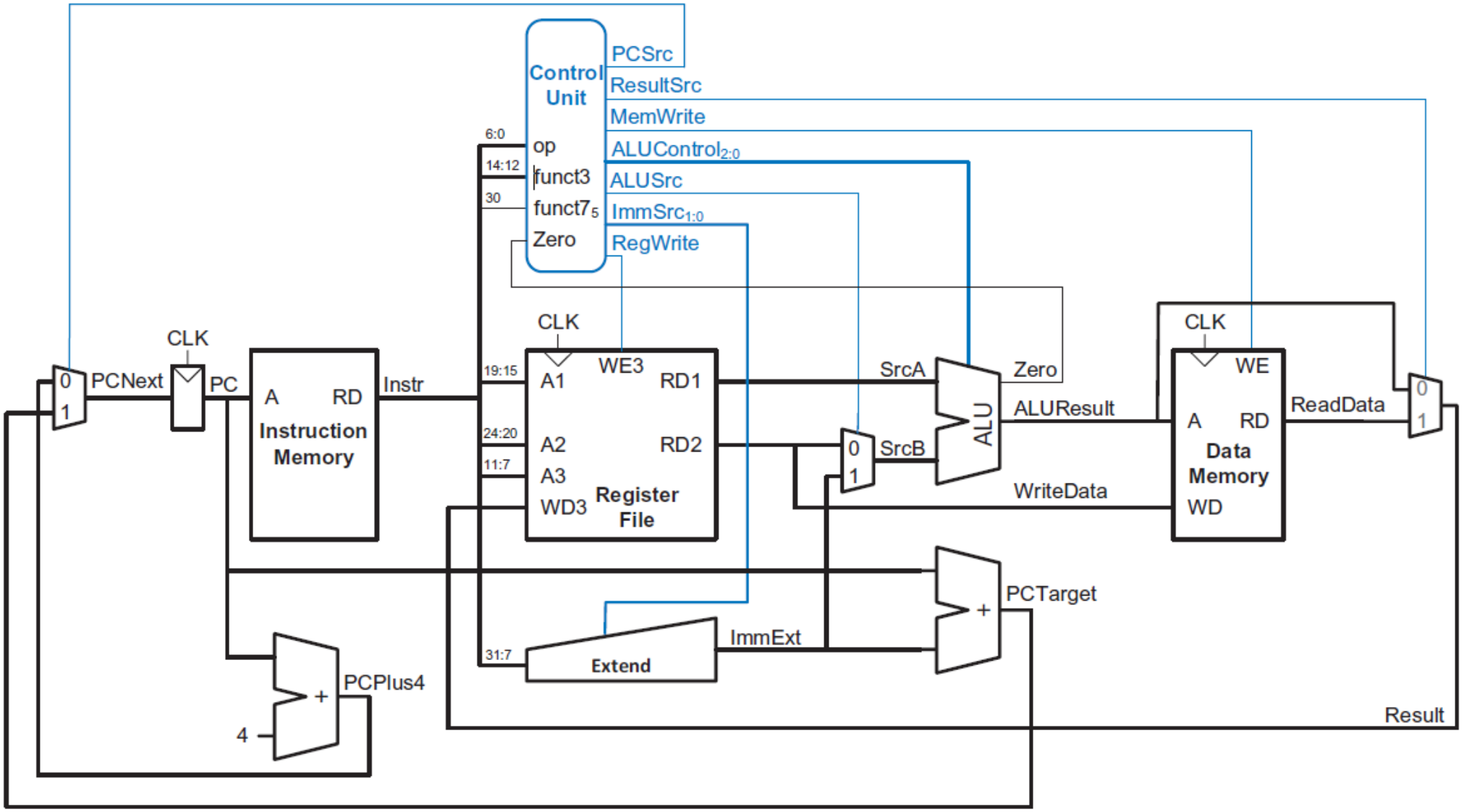


Unidade de Controle

- Combinando...

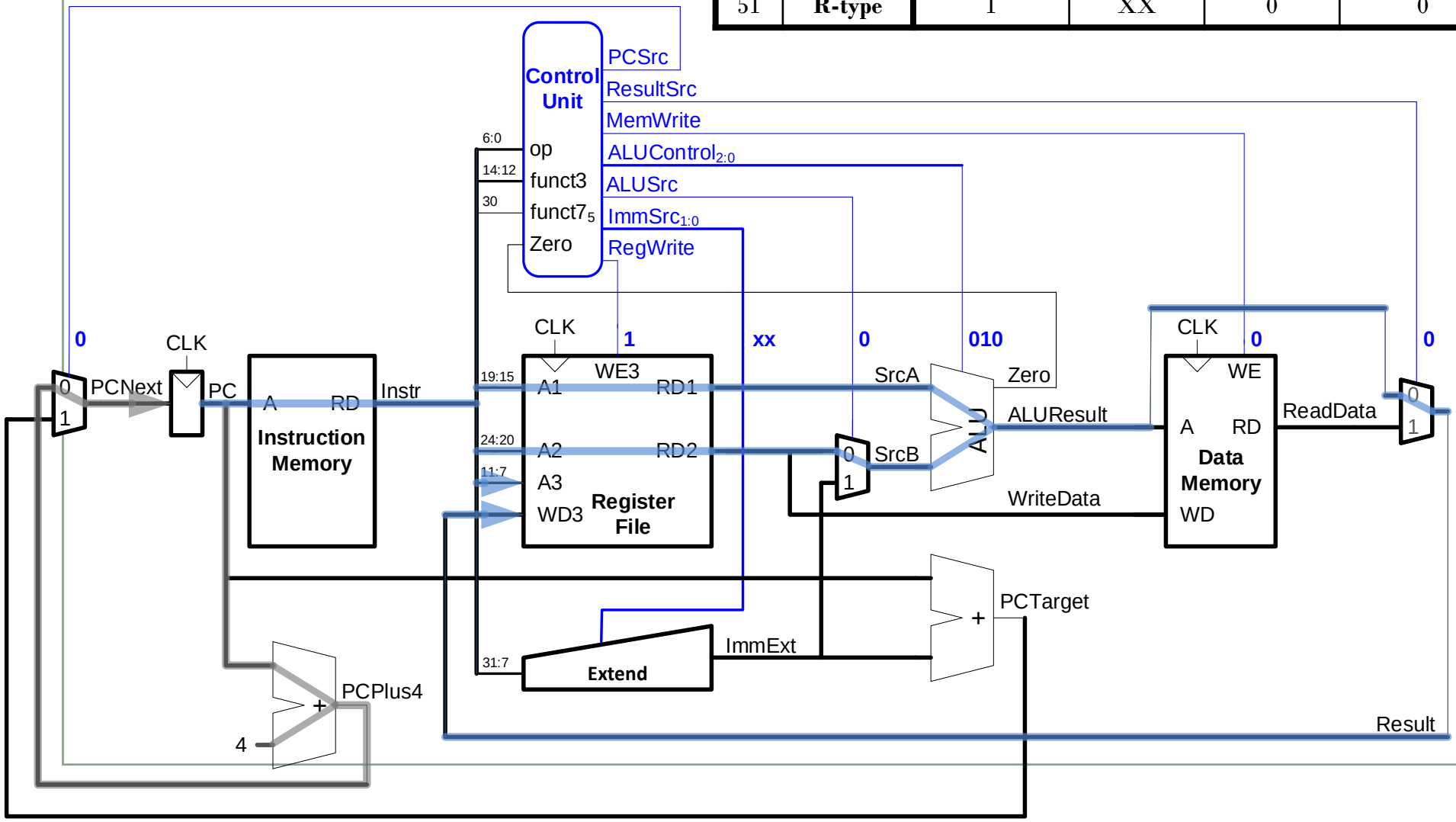


Unidade de Controle



Datapath Utilizado p/ Instruções do Tipo-R

op	Instruct	RegWrite	ImmSrc	ALUSrc	MemWrite	ResultSrc	Branch	ALUOp
51	**R-type**	1	XX	0	0	0	0	10



Unidade de Controle

- Linhas de Controle: valores

op	Instr.	RegWrite	ImmSrc	ALUSrc	MemWrite	ResultSrc	Branch	ALUOp
3	**`lw`**	1	00	1	0	1	0	00
35	**`sw`**	0	01	1	1	X	0	00
51	**R-type**	1	XX	0	0	0	0	10
99	**`beq`**	0	10	0	0	X	1	01

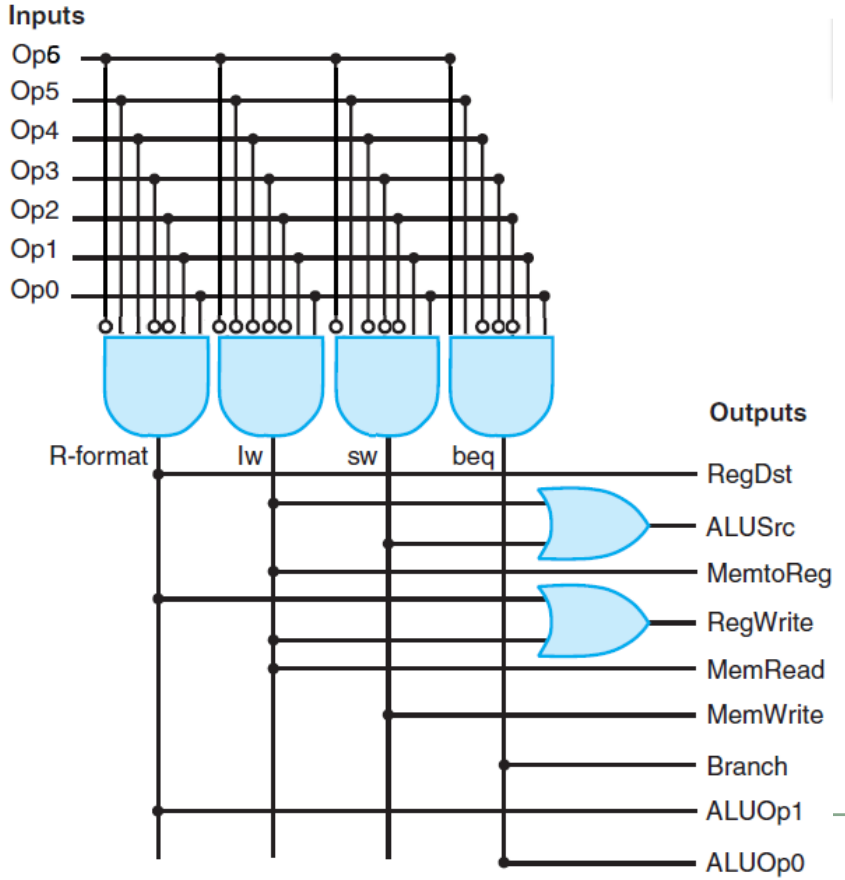
Unidade de Controle

- Linhas de Controle
  - Os níveis lógicos dos sinais de controle gerados pela unidade central dependem exclusivamente do opcode da instrução.

Input or output	Signal name	R-format	ld	sd	beq
Inputs	I[6]	0	0	0	1
	I[5]	1	0	1	1
	I[4]	1	0	0	0
	I[3]	0	0	0	0
	I[2]	0	0	0	0
	I[1]	1	1	1	1
	I[0]	1	1	1	1
Outputs	ALUSrc	0	1	1	0
	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

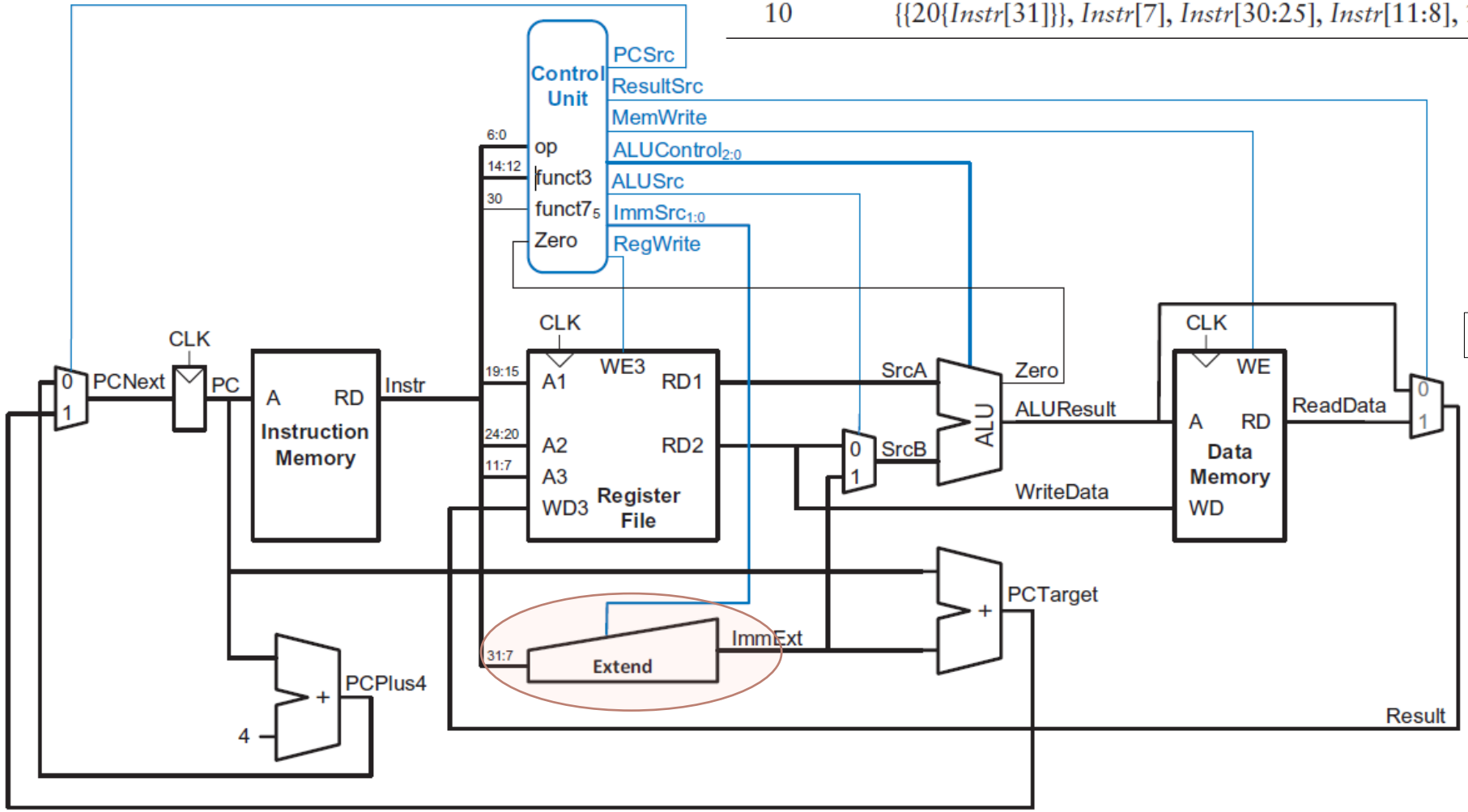
Unidade de Controle

- Linhas de Controle
  - A partir das informação da tabela verdade, é possível implementar diretamente a unidade central de controle usando portas lógicas:



Datapath Utilizado p/ Instruções do Tipo-R

ImmSrc	ImmExt	Type	Description
00	{{20{*Instr*[31]}}, *Instr*[31:20]}	I	12-bit signed immediate
01	{{20{*Instr*[31]}}, *Instr*[31:25], *Instr*[11:7]}	S	12-bit signed immediate
10	{{20{*Instr*[31]}}, *Instr*[7], *Instr*[30:25], *Instr*[11:8], 1'b0}	B	13-bit signed immediate



I-Type

31:20	19:15	14:12	11:7	6:0
$imm_{11:0}$	rs1	funct3	rd	op
12 bits	5 bits	3 bits	5 bits	7 bits

S-Type

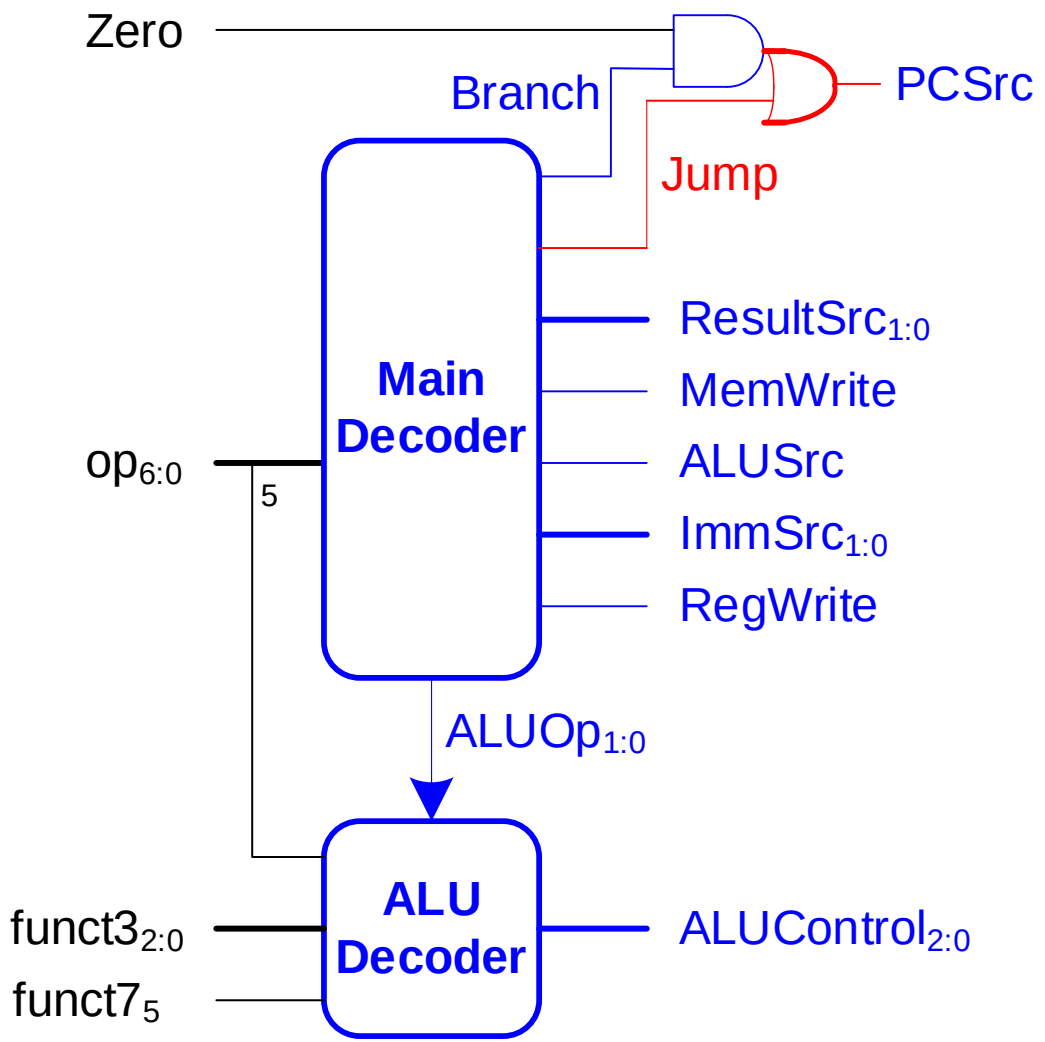
31:25	24:20	19:15	14:12	11:7	6:0
$imm_{11:5}$	rs2	rs1	funct3	$imm_{4:0}$	op
7 bits	5 bits	5 bits	3 bits	5 bits	7 bits

B-Type

31:25	24:20	19:15	14:12	11:7	6:0
$imm_{12,10:5}$	rs2	rs1	funct3	$imm_{4:1,11}$	op
7 bits	5 bits	5 bits	3 bits	5 bits	7 bits

Unidade de Controle

JAL

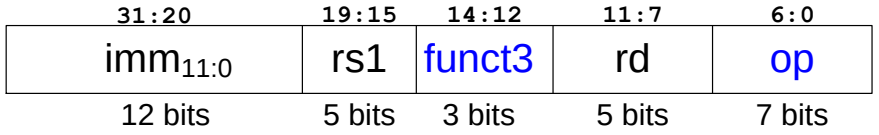


Unidade de Controle

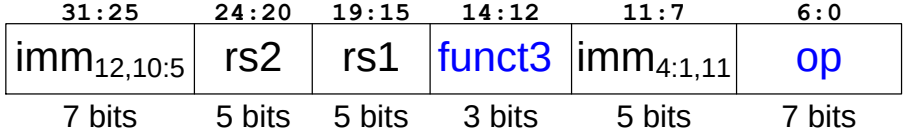
JAL

$ImmSrc_{1:0}$	ImmExt	Instruction Type
00	{{20{instr[31]}}, instr[31:20]}	I-Type
01	{{20{instr[31]}}, instr[31:25], instr[11:7]}	S-Type
10	{{19{instr[31]}}, instr[31], instr[7], instr[30:25], instr[11:8], 1'b0}	B-Type
11	**{{12{instr[31]}}, instr[19:12], instr[20], instr[30:21], 1'b0}**	**J-Type**

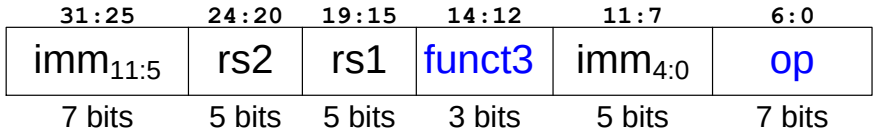
I-Type



B-Type



S-Type

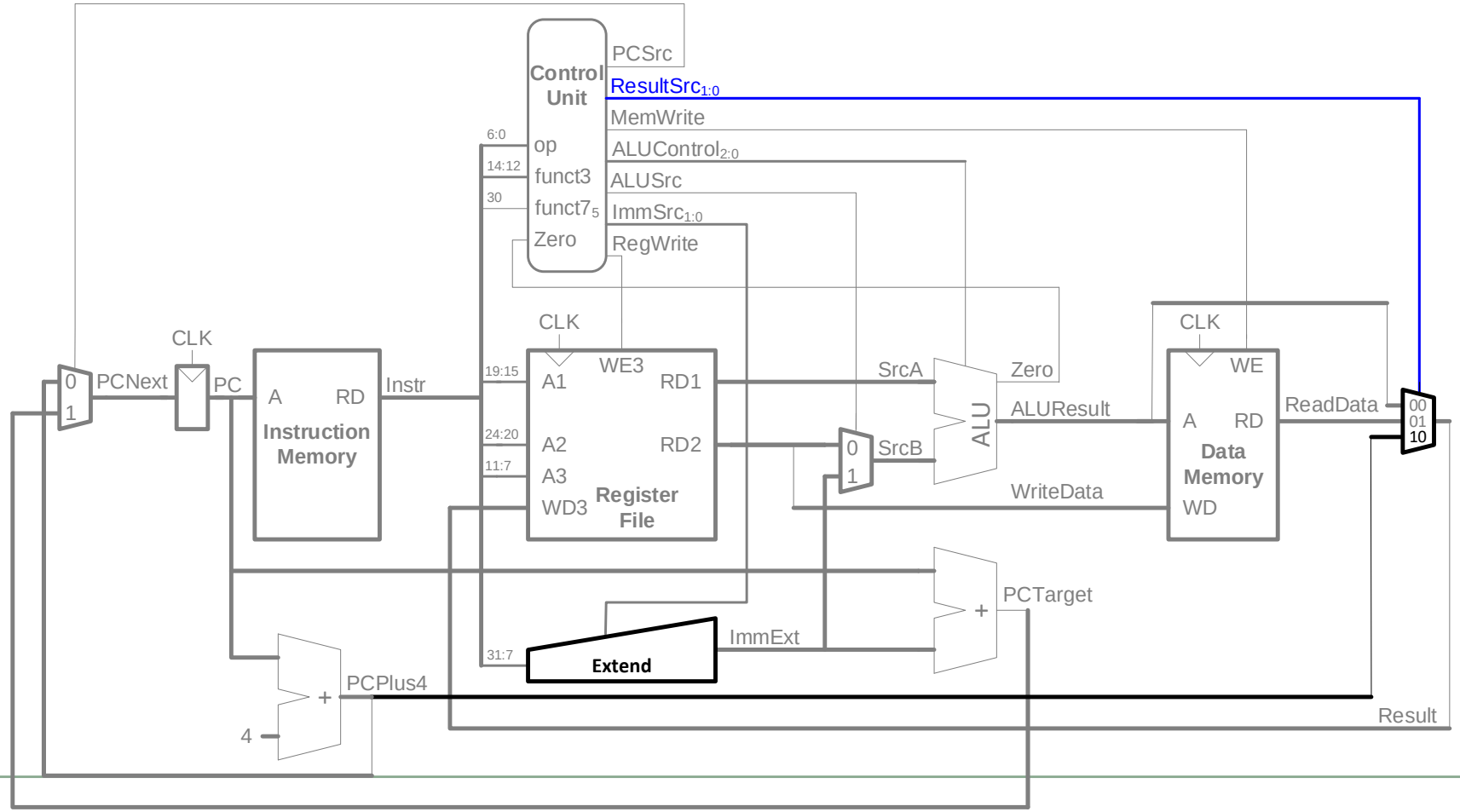


J-Type

Unidade de Controle

JAL

op	Instruct	RegWrite	ImmSrc	ALUSrc	MemWrite	ResultSrc	Branch	ALUOp	Jump
111	jal	1	11	X	0	10	0	XX	1



Execução de uma Instrução Tipo R

- Seja a instrução tipo R add t1, t2, t3, podemos imaginar que esta instrução é executada em 4 etapas:

    1. Busca da instrução (na memória de instruções) e incremento do PC

    2. Leitura de dois registradores (no caso, t2 e t3, ou rs1 e rs2) e geração dos sinais de controle para o resto do bloco operativo (decodificação da instrução)

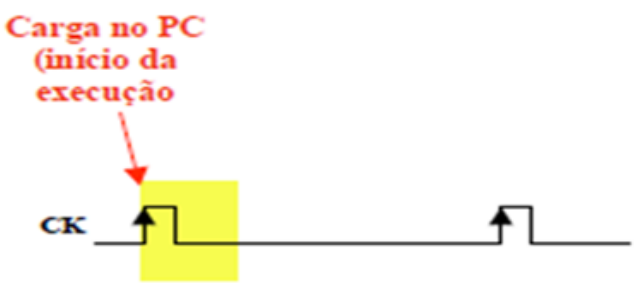
    3. Operação na ULA

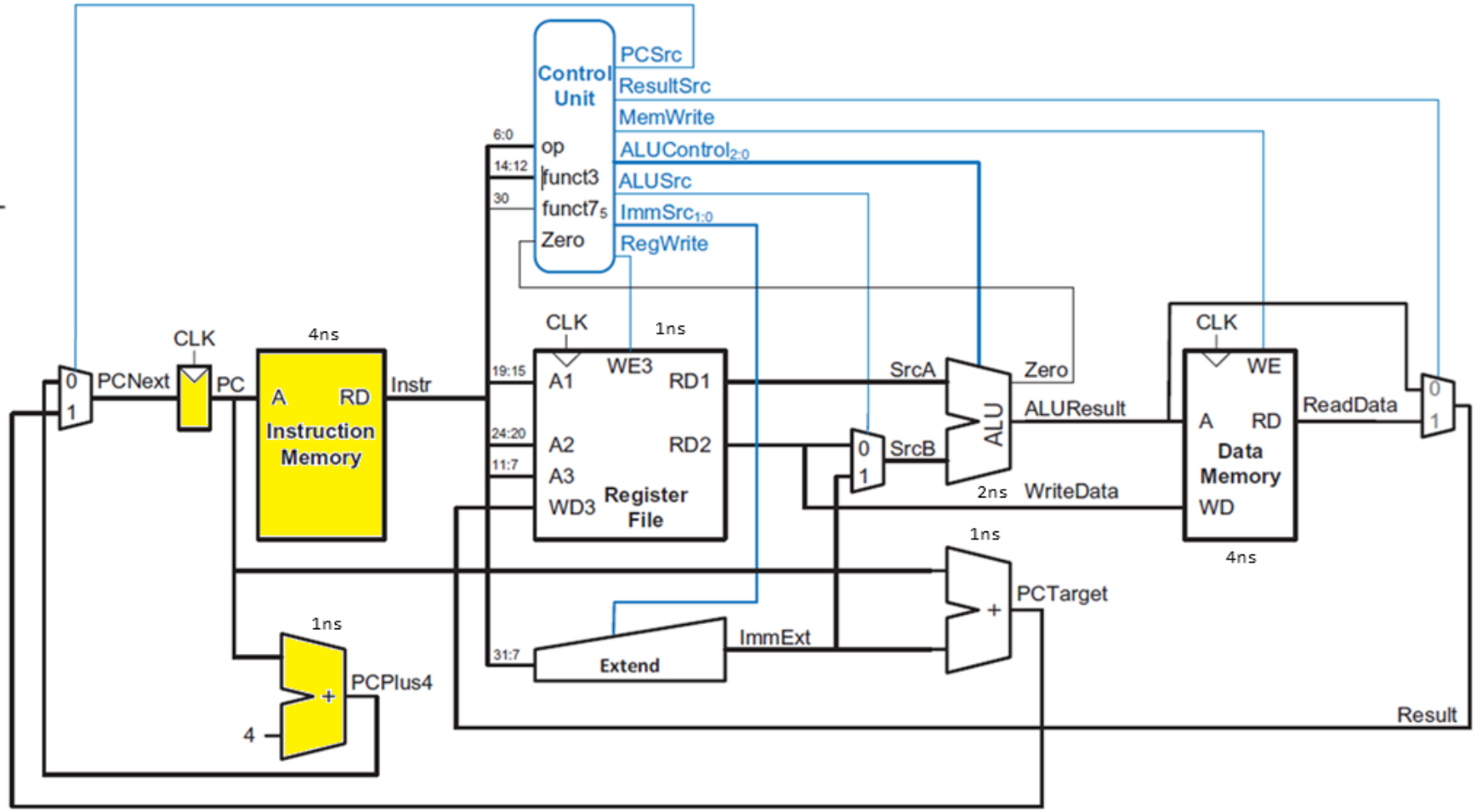
    4. Escrita (do resultado da operação realizada na ULA) no registrador destino (t1 ou rd)

Como estes passos ocorrem dentro do mesmo ciclo de relógio (regime monociclo), a ordem real irá depender do atraso de cada componente.

Execução de uma Instrução Tipo R

- Busca da instrução e cálculo de PC+4

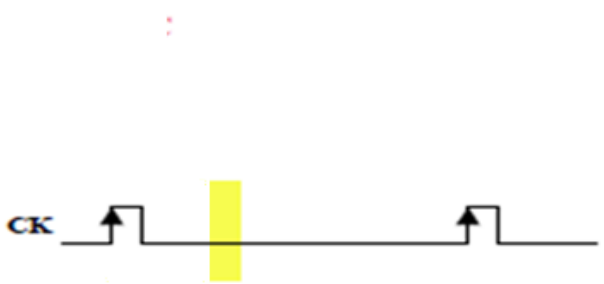


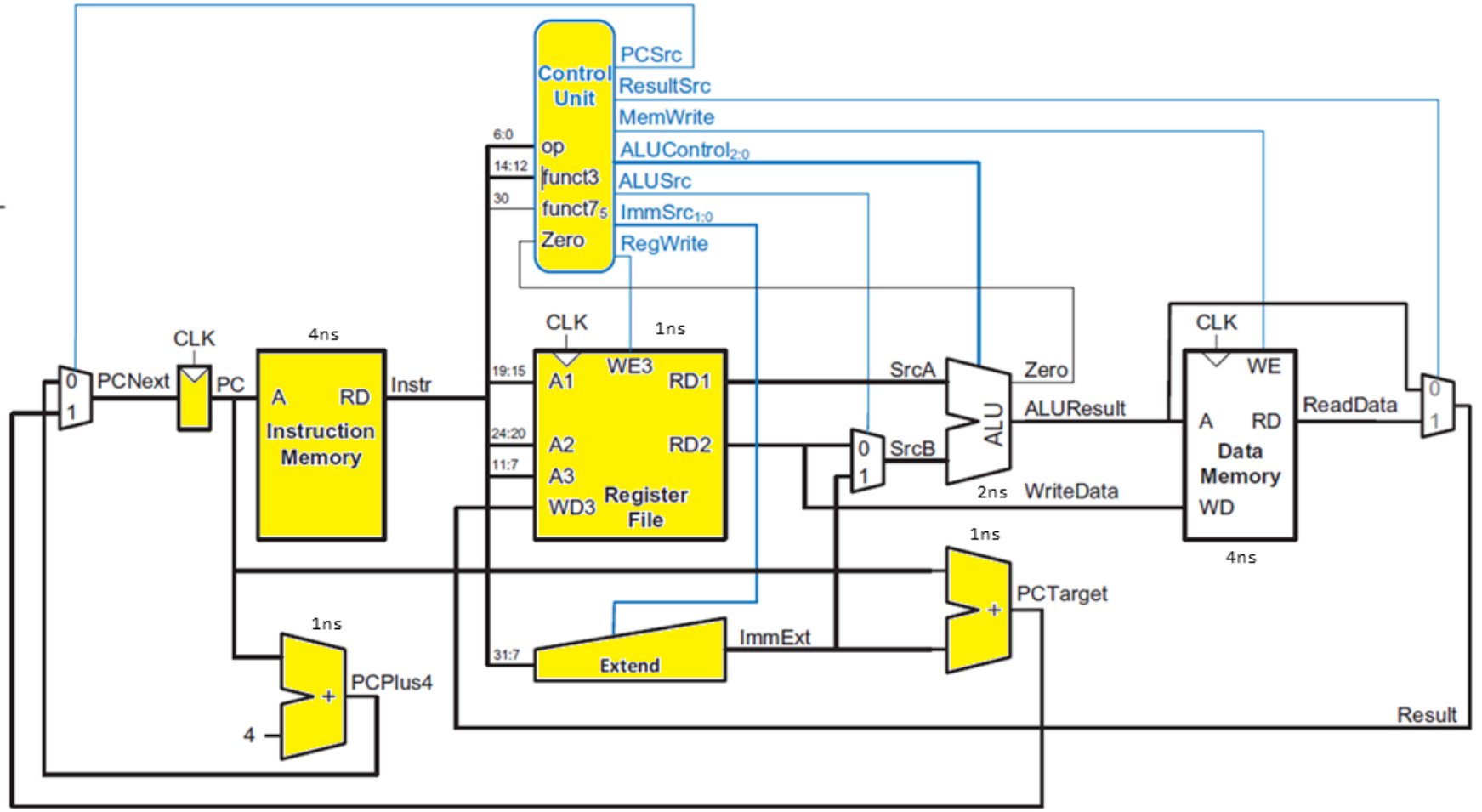


Fonte: Prof. José Luís Güntzel - Arquitetura e Organização de Computadores II

Execução de uma Instrução Tipo R

- Leitura de Rs e Rt e geração sinais de controle

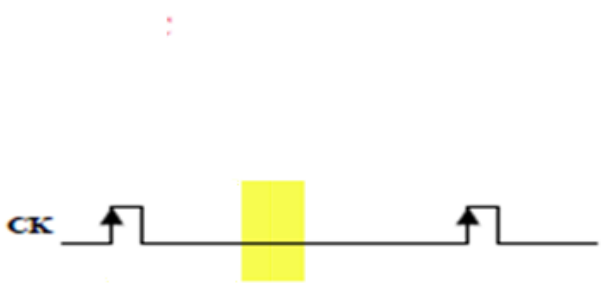


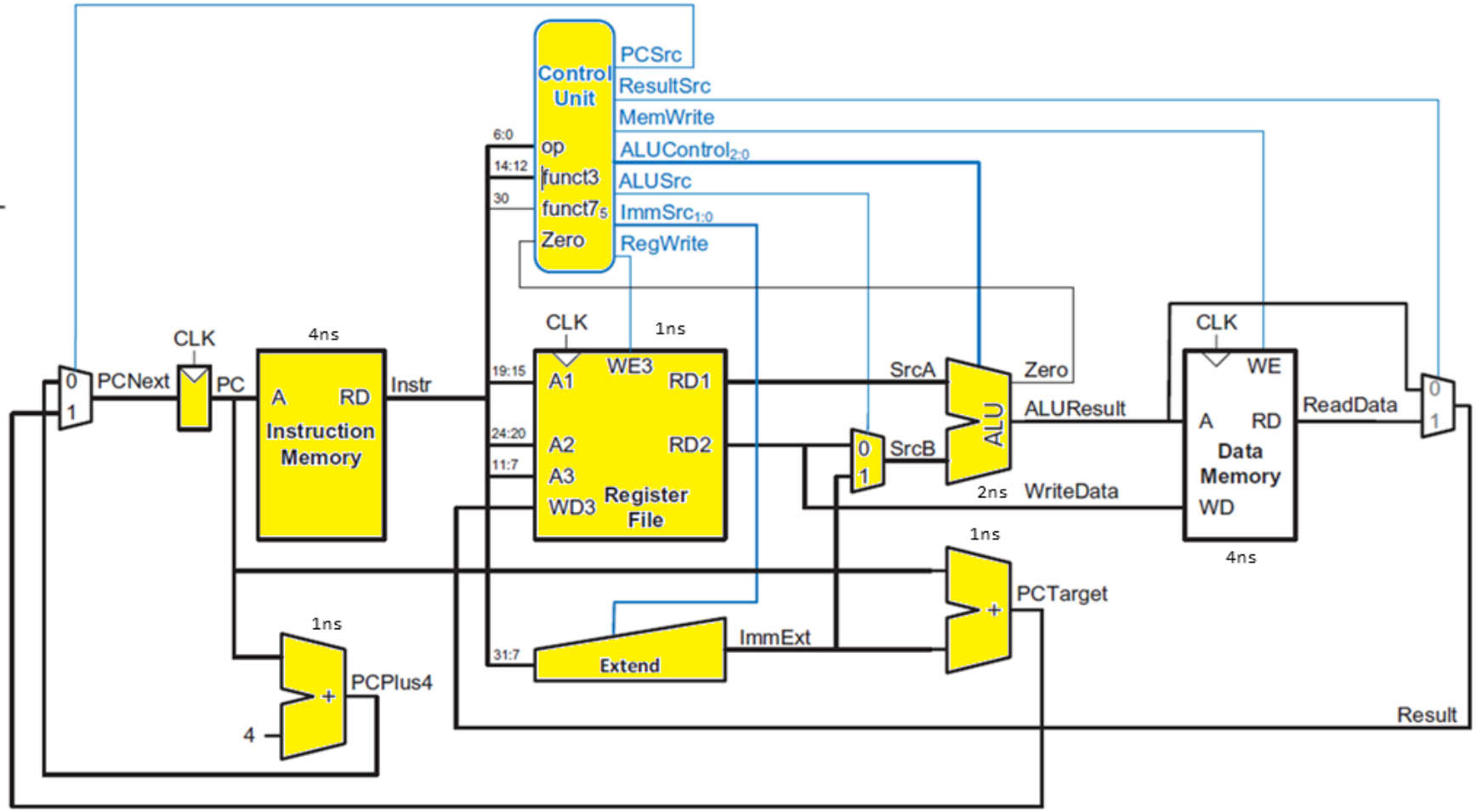


Fonte: Prof. José Luís Güntzel - Arquitetura e Organização de Computadores II

Execução de uma Instrução Tipo R

- Operação na ULA (depende de "funct")

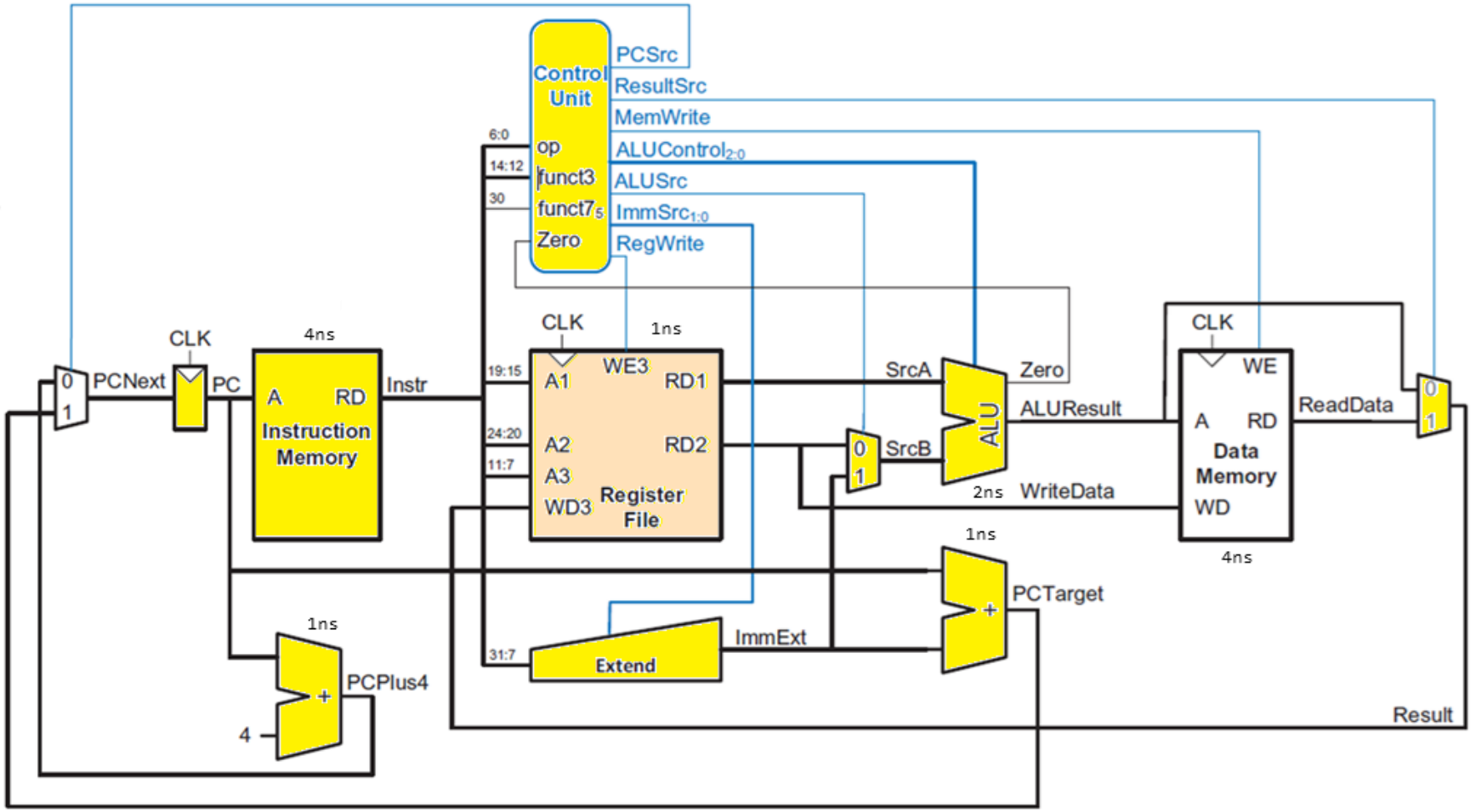




Fonte: Prof. José Luís Güntzel - Arquitetura e Organização de Computadores II

Execução de uma Instrução Tipo R

- Escrita no registrador-destino



Fonte: Prof. José Luís Güntzel - Arquitetura e Organização de Computadores II

Execução de uma Instrução lw

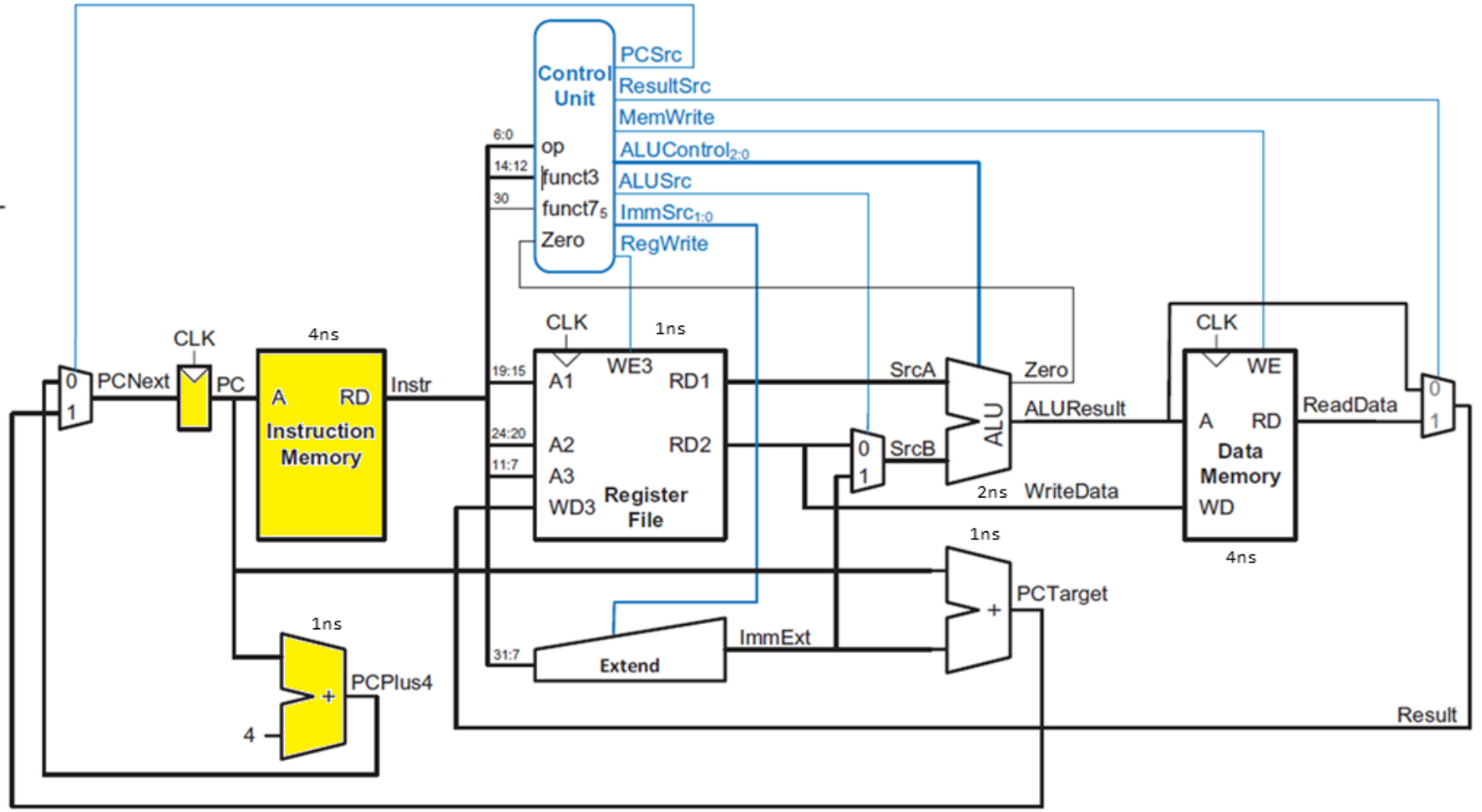
- Seja a instrução tipo I lw t1, deslocamento(t2), podemos imaginar que esta instrução é executada em 5 etapas:

1. Busca da instrução (na memória de instruções) e incremento do PC
2. Leitura de dois registradores (no caso, t1 e t2, ou rsd e rs1) e geração dos sinais de controle para o resto do bloco operativo (decodificação da instrução).
3. Cálculo do endereço usando a ULA (adição)
4. Acesso à memória de dados para uma leitura (endereço = resultado da ULA)
5. Escrita (do valor lido da memória de dados) no registrador destino (t1, que neste caso corresponde ao campo rd)

Fonte: Prof. José Luís Güntzel - Arquitetura e Organização de Computadores II

Execução de uma Instrução lw

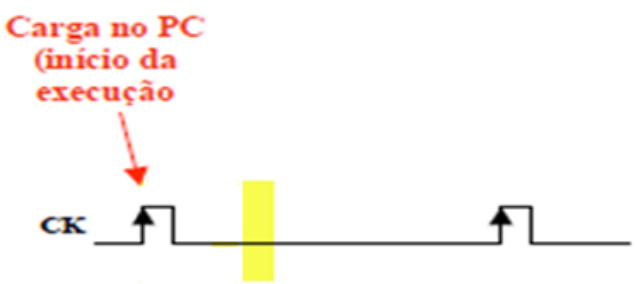
- Busca da instrução e cálculo de PC+4

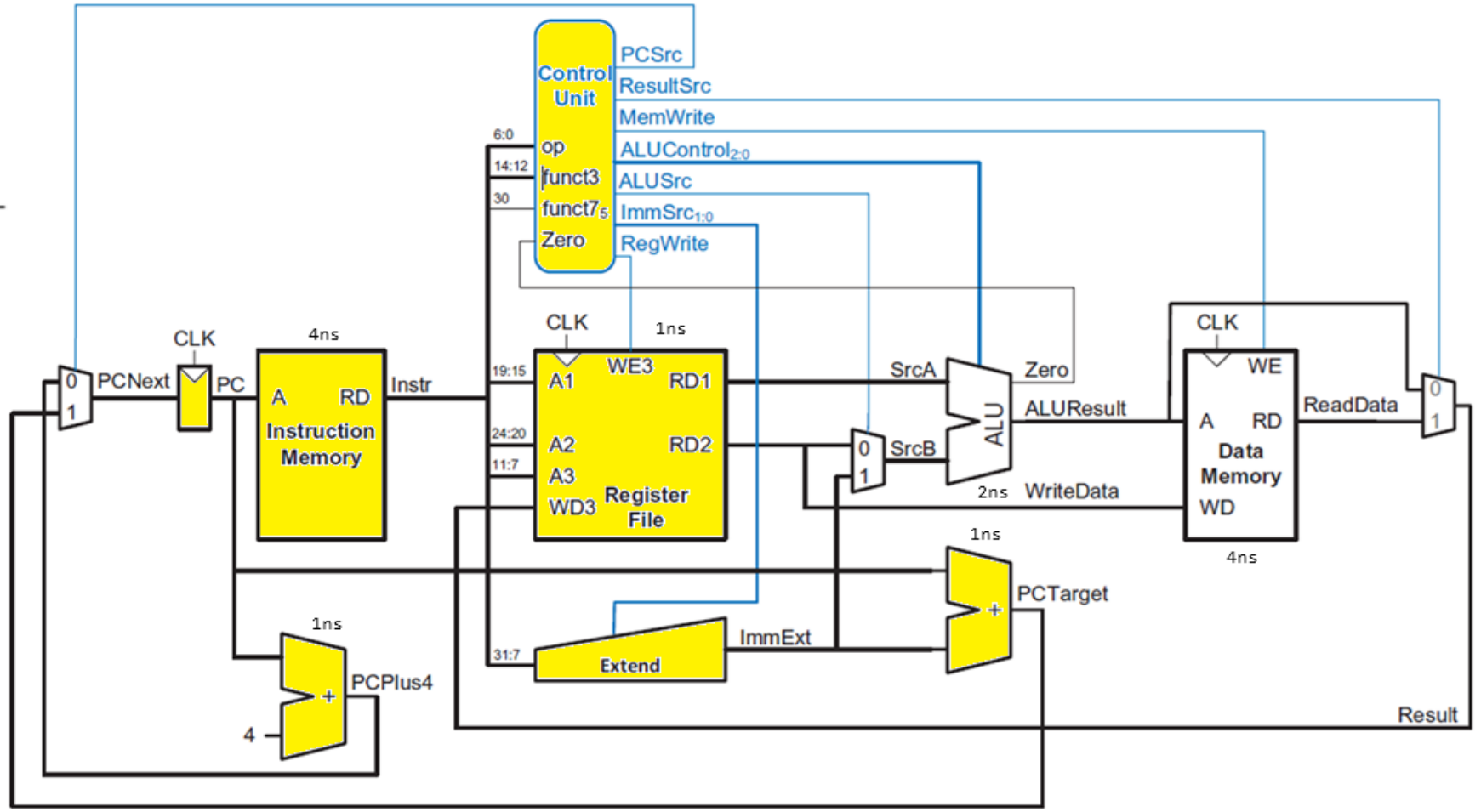


Fonte: Prof. José Luís Güntzel - Arquitetura e Organização de Computadores II

Execução de uma Instrução lw

- Leitura de Rs (e Rt) e geração sinais de controle

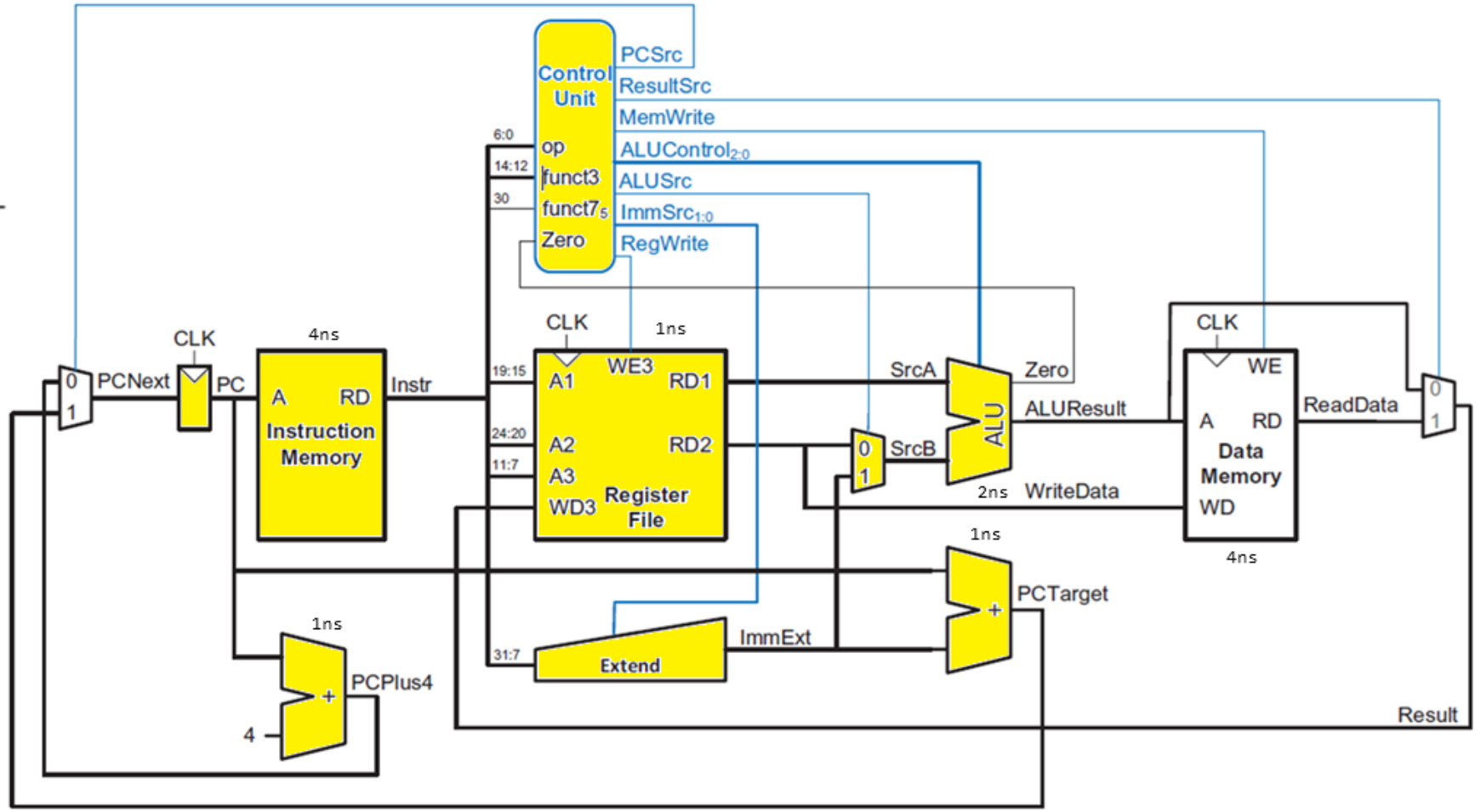




Fonte: Prof. José Luís Güntzel - Arquitetura e Organização de Computadores II

Execução de uma Instrução lw

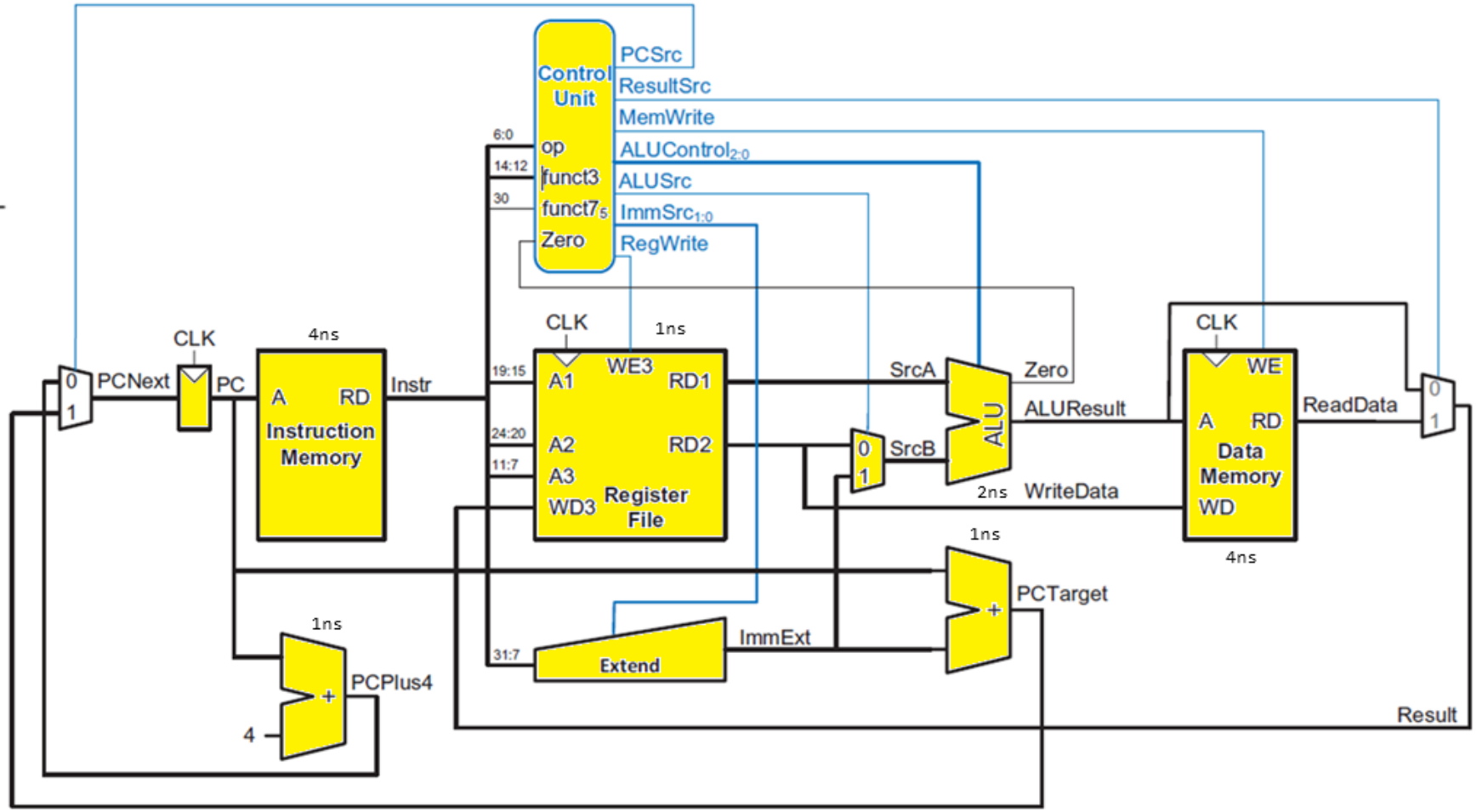
- Cálculo do endereço usando a ULA (adição)



Fonte: Prof. José Luís Güntzel - Arquitetura e Organização de Computadores II

Execução de uma Instrução lw

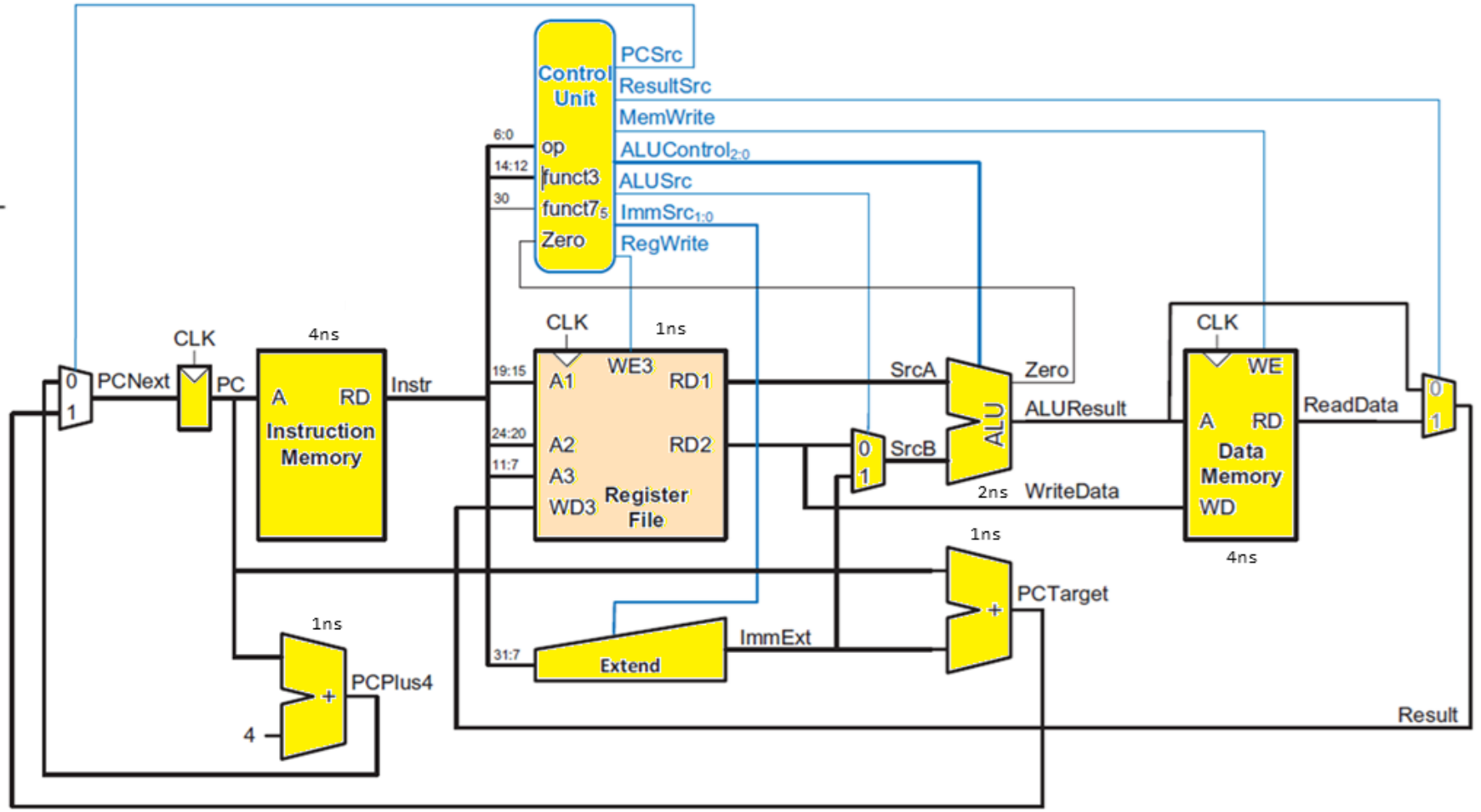
- Acesso à memória de dados para uma leitura



Fonte: Prof. José Luís Güntzel - Arquitetura e Organização de Computadores II

Execução de uma Instrução lw

- Escrita no registrador-destino



Fonte: Prof. José Luís Güntzel - Arquitetura e Organização de Computadores II

Desempenho de Máquinas Monociclo

- Unidades funcionais utilizadas por cada instrução

instrução	Etapa 1	Etapa 2	Etapa3	Etapa 4	Etapa 5
Tipo R	Busca da instrução	Lê registrador(es)	ULA	Escreve registrador	
lw	Busca da instrução	Lê registrador(es)	ULA	Lê memória	Escreve registrador
sw	Busca da instrução	Lê registrador(es)	ULA	Escreve na memória	
beq	Busca da instrução	Lê registrador(es)	ULA		
jump	Busca da instrução				

Fonte: Prof. José Luís Güntzel - Arquitetura e Organização de Computadores II

Desempenho de Máquinas Monociclo

- Tempo de execução de cada instrução (com valores hipotéticos de atraso para cada etapa)

instrução	Acesso à memória de instruções	Leitura de registradores	Operação na ULA	Acesso à memória de dados	Escrita no registrador	Total
Tipo R	4 ns	1 ns	2 ns	---	1 ns	8 ns
lw	4 ns	1 ns	2 ns	4 ns	1 ns	12 ns
sw	4 ns	1 ns	2 ns	4 ns	---	11 ns
beq	4 ns	1 ns	2 ns	---	---	7 ns
jump	4 ns	---	---	---	---	4 ns

Fonte: Prof. José Luís Güntzel - Arquitetura e Organização de Computadores II

Conclusões

- Vimos como é possível construir um datapath relativamente simples para executar instruções da arquitetura RiSC-V
- Os mesmos princípios se aplicam para se implementar outras instruções
- O datapath apresentado pode ser otimizado (para execução mais rápida) em vários aspectos
- Uma de suas limitações é o fato de ser mono-ciclo, ou seja, executar qualquer instrução em um ciclo de clock, porém esse ciclo é bastante demorado.
- Solução: implementação multi-ciclo desse datapath.

Conclusões

- A duração de um ciclo de clock deve ser longa o bastante para comportar o caminho mais longo no processador.
- A penalidade por utilizar um projeto monociclo é significativa, mas pode ser considerada aceitável para um conjunto bastante reduzido de instruções – foi explorada inicialmente em computadores com conjuntos bastante simples de instruções.
- Observação:
  - Como o ciclo de relógio é determinado pelo atraso de pior caso entre todas as instruções, é absolutamente inútil o uso de técnicas que reduzem o atraso do caso comum, mas que não trazem qualquer otimização do pior caso.