Arquitetura de Computadores
ACH2055
Aula 06 – *Datapath* de Ciclo Único e *Pipeline*

Norton Trevisan Roman
(norton@usp.br)

30 de setembro de 2019

Construindo o *Datapath*

Preparação

- Nosso *datapath* deve executar cada instrução num único ciclo do *clock*
  - Nenhum de seus recursos podem ser usados mais que $1\times$ por instrução

Construindo o *Datapath*

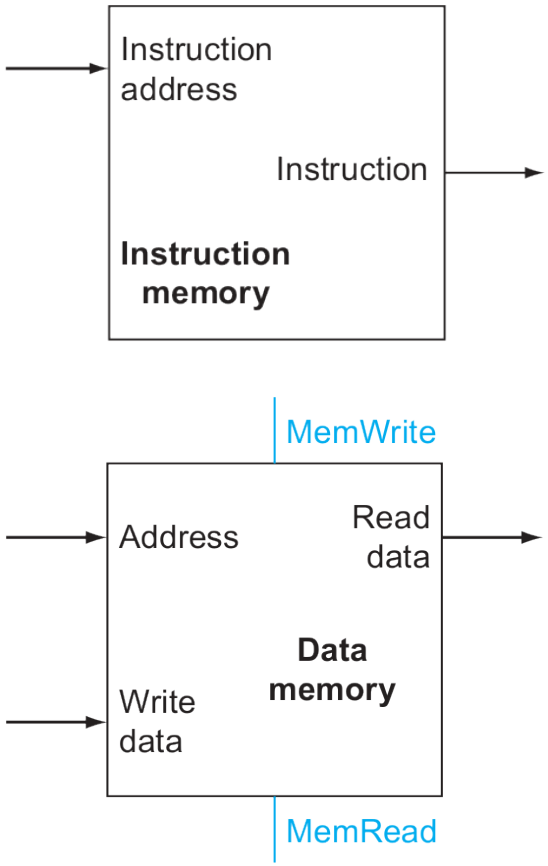
Preparação

- Nosso *datapath* deve executar cada instrução num único ciclo do *clock*
  - Nenhum de seus recursos podem ser usados mais que $1\times$ por instrução
- Por isso precisaremos de 2 memórias
  - Uma para as instruções e outra para os dados

Construindo o *Datapath*

Preparação

- Nosso *datapath* deve executar cada instrução num único ciclo do *clock*
  - Nenhum de seus recursos podem ser usados mais que 1× por instrução
- Por isso precisaremos de 2 memórias
  - Uma para as instruções e outra para os dados



Fonte: [1]

Construindo o *Datapath*

Preparação

- Além disso, os componentes do *datapath* precisarão ser compartilhados entre as diferentes famílias de instruções
  - Precisaremos de uma maneira de controlar quando um componente será usado

Construindo o *Datapath*

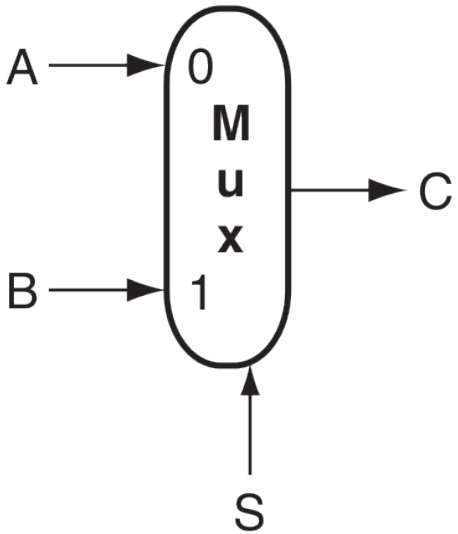
Preparação

- Além disso, os componentes do *datapath* precisarão ser compartilhados entre as diferentes famílias de instruções
  - Precisaremos de uma maneira de controlar quando um componente será usado
- Precisaremos então de um multiplexador
  - Além de sinais de controle, para selecionar dentre as múltiplas entradas

Construindo o *Datapath*

Preparação: Multiplexador

- Trata-se de um seletor
  - Sua saída (C) é uma das entradas (A ou B), selecionada por uma linha de controle (S)

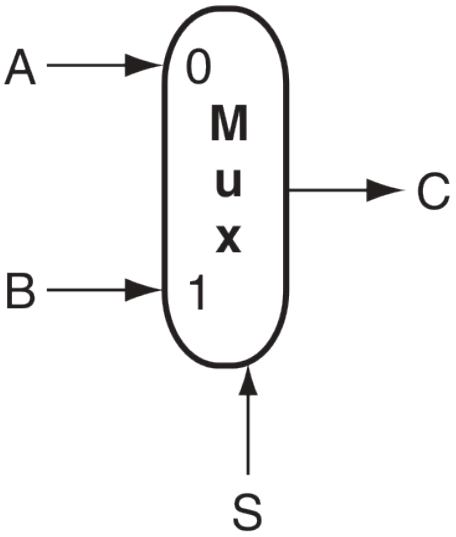


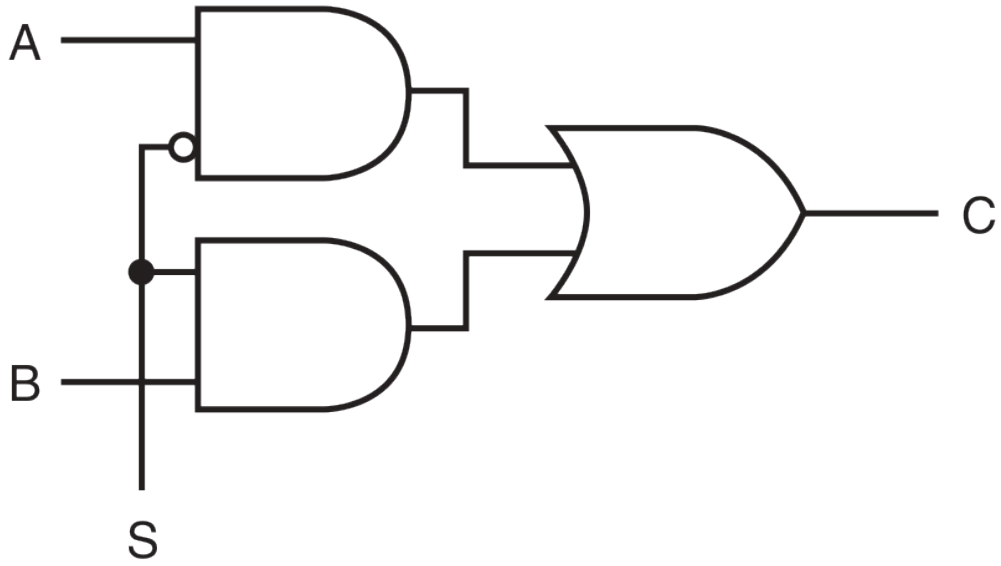
Fonte: [1]

Construindo o *Datapath*

Preparação: Multiplexador

- Trata-se de um seletor
  - Sua saída (C) é uma das entradas (A ou B), selecionada por uma linha de controle (S)





Fonte: [1]

Construindo o *Datapath*

Unindo instruções de memória e tipo-R

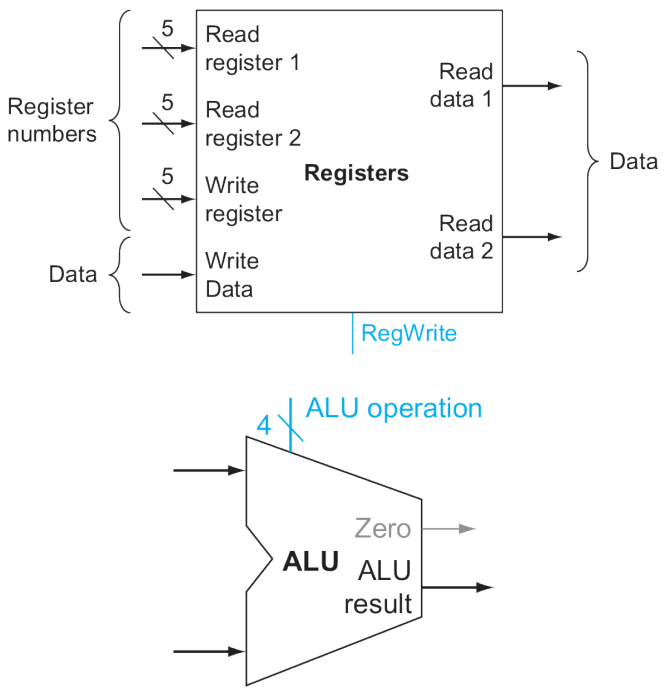
- Do que precisávamos para Tipo-R e Memória?

Construindo o *Datapath*

Unindo instruções de memória e tipo-R

- Do que precisávamos para Tipo-R e Memória?

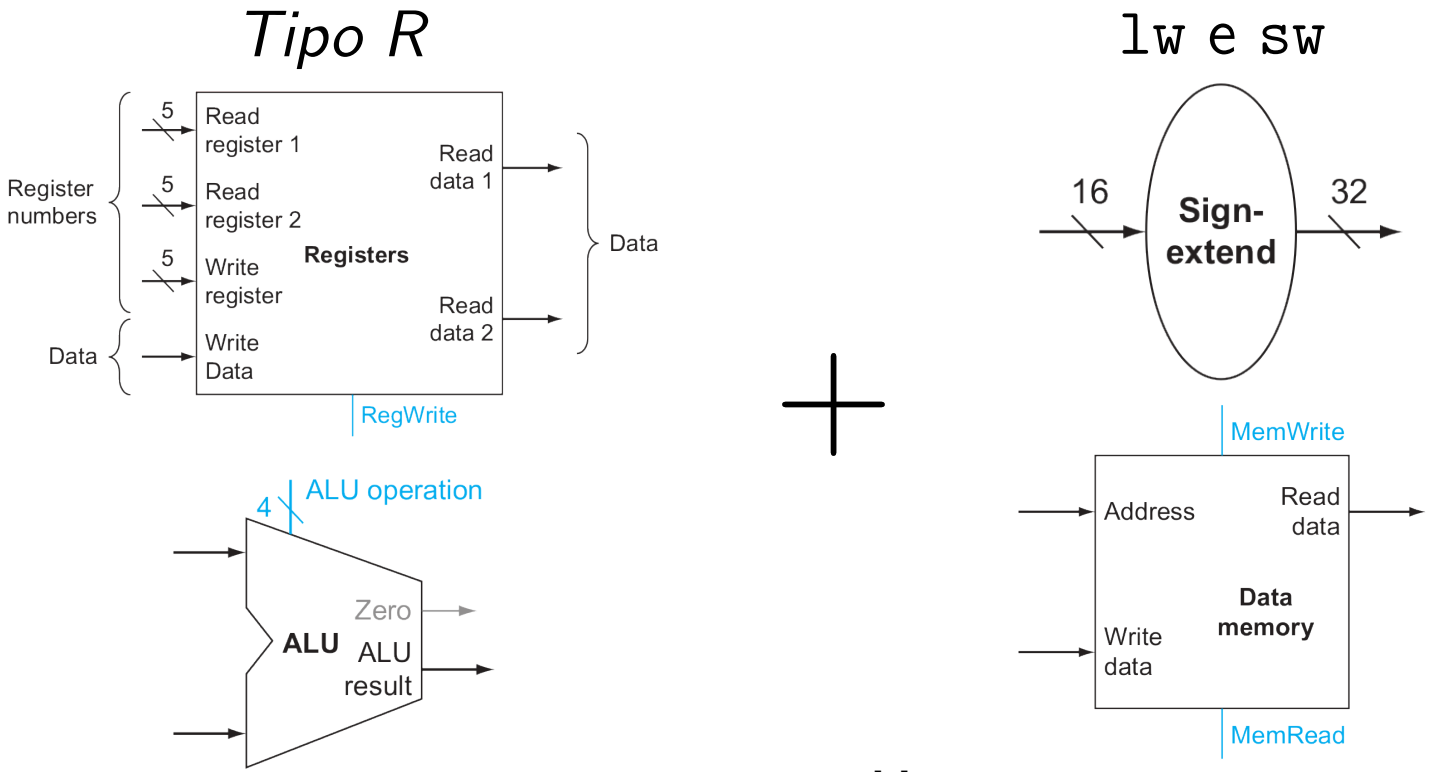
Tipo R



Construindo o *Datapath*

Unindo instruções de memória e tipo-R

- Do que precisávamos para Tipo-R e Memória?

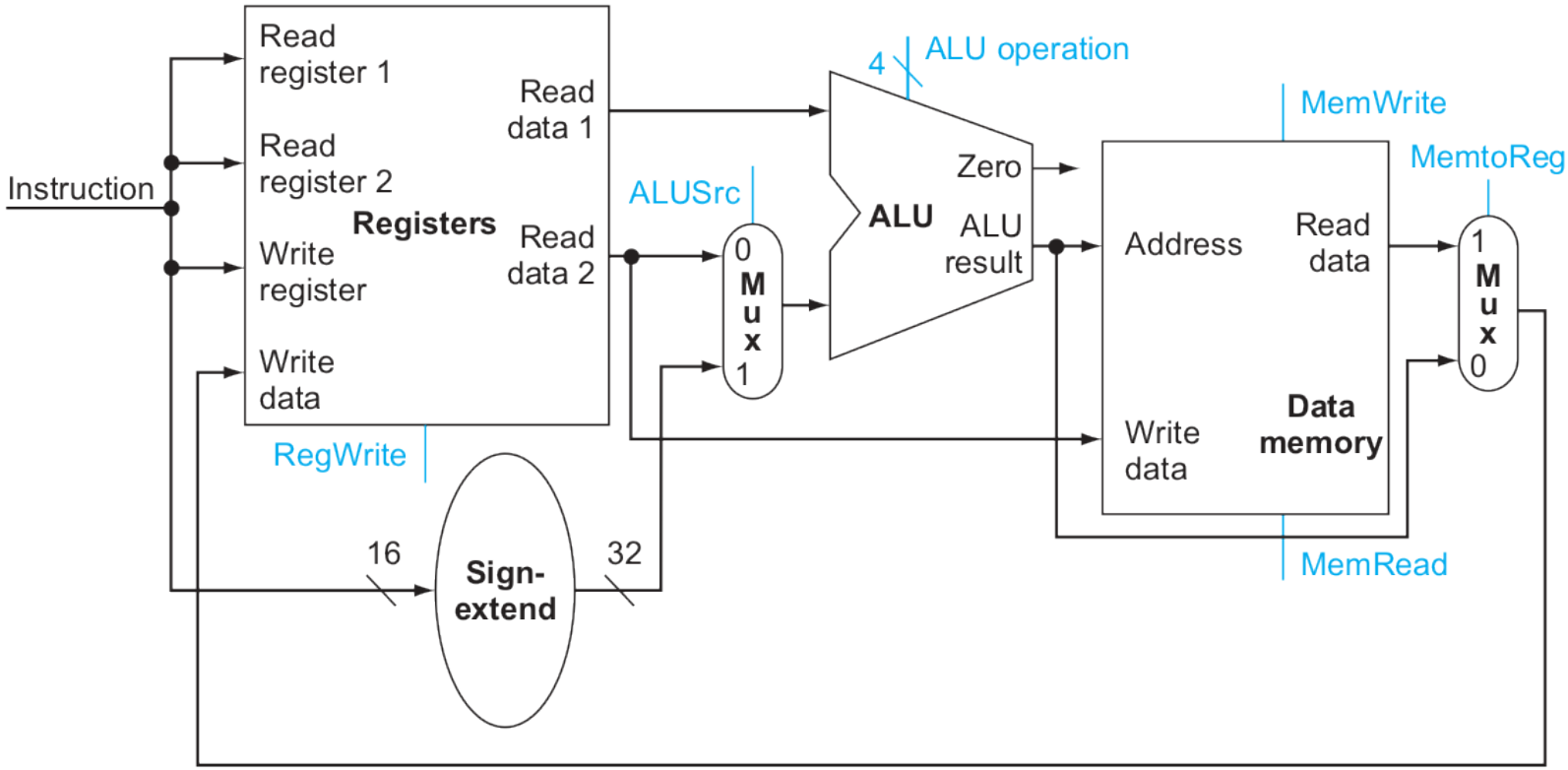


Fonte: [1]

Construindo o *Datapath*

Unindo instruções de memória e tipo-R

- Então...

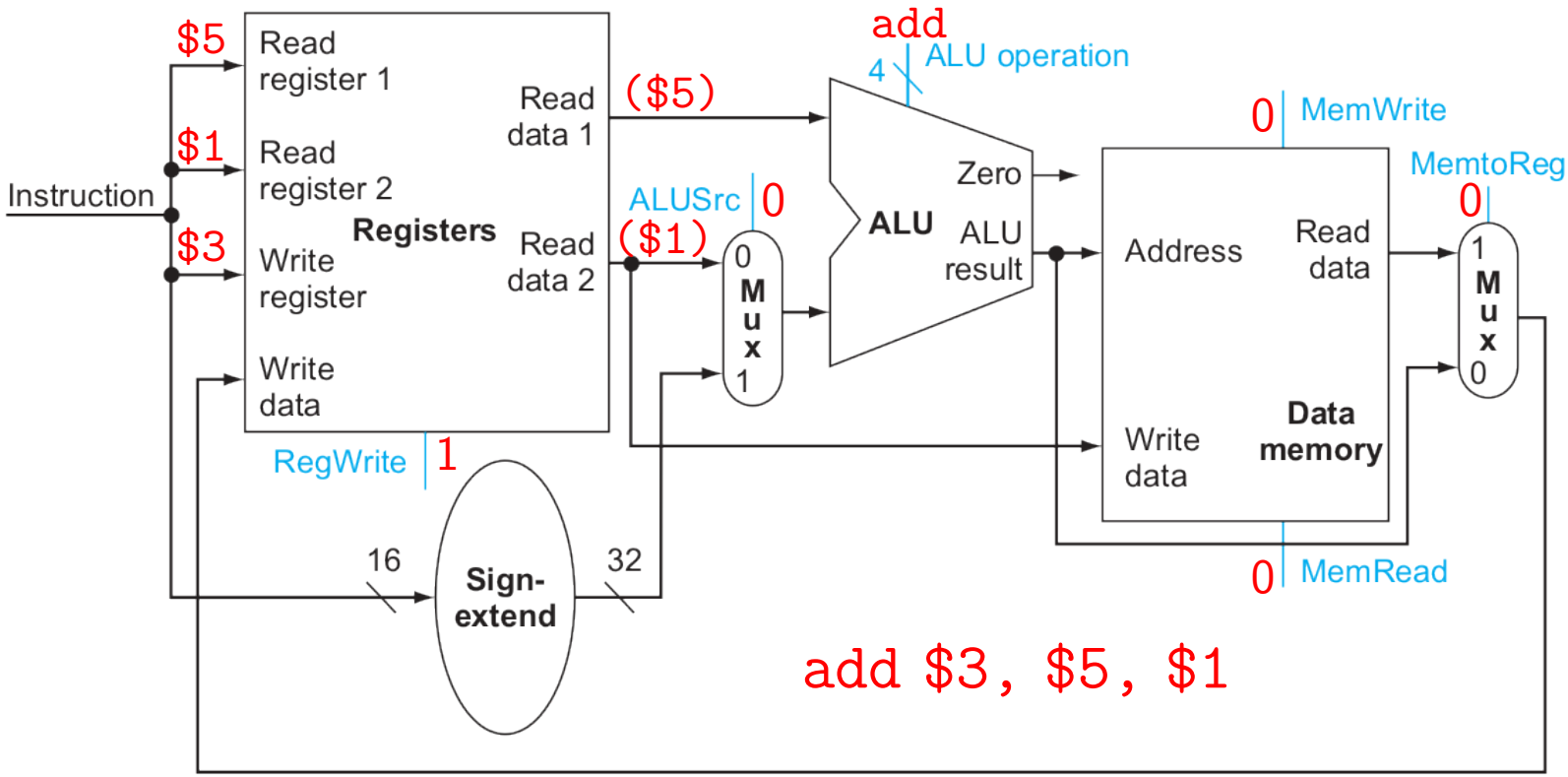


Fonte: Adaptado de [1]

Construindo o *Datapath*

Unindo instruções de memória e tipo-R

- Então...

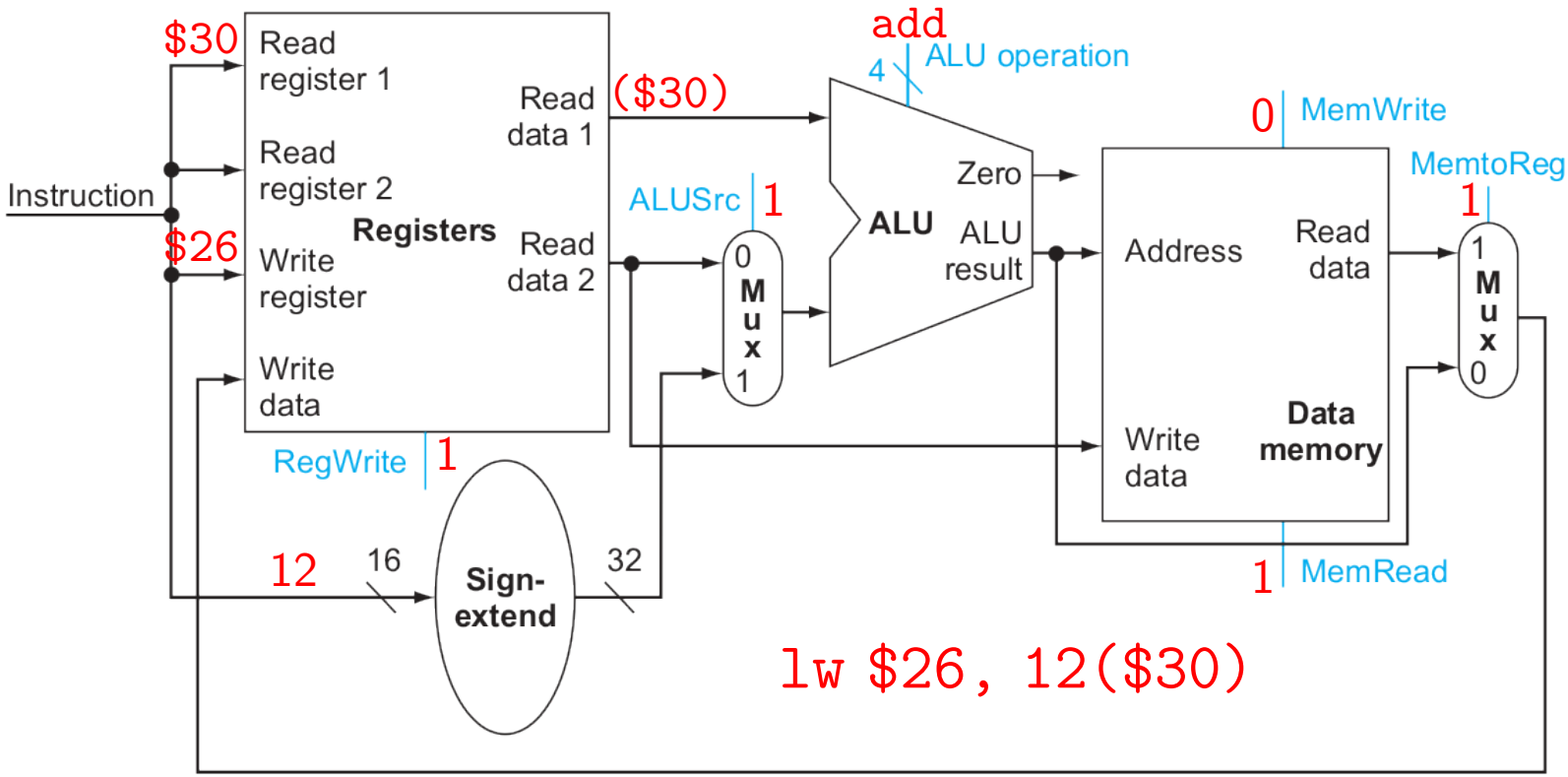


Fonte: Adaptado de [1]

Construindo o *Datapath*

Unindo instruções de memória e tipo-R

- Então...

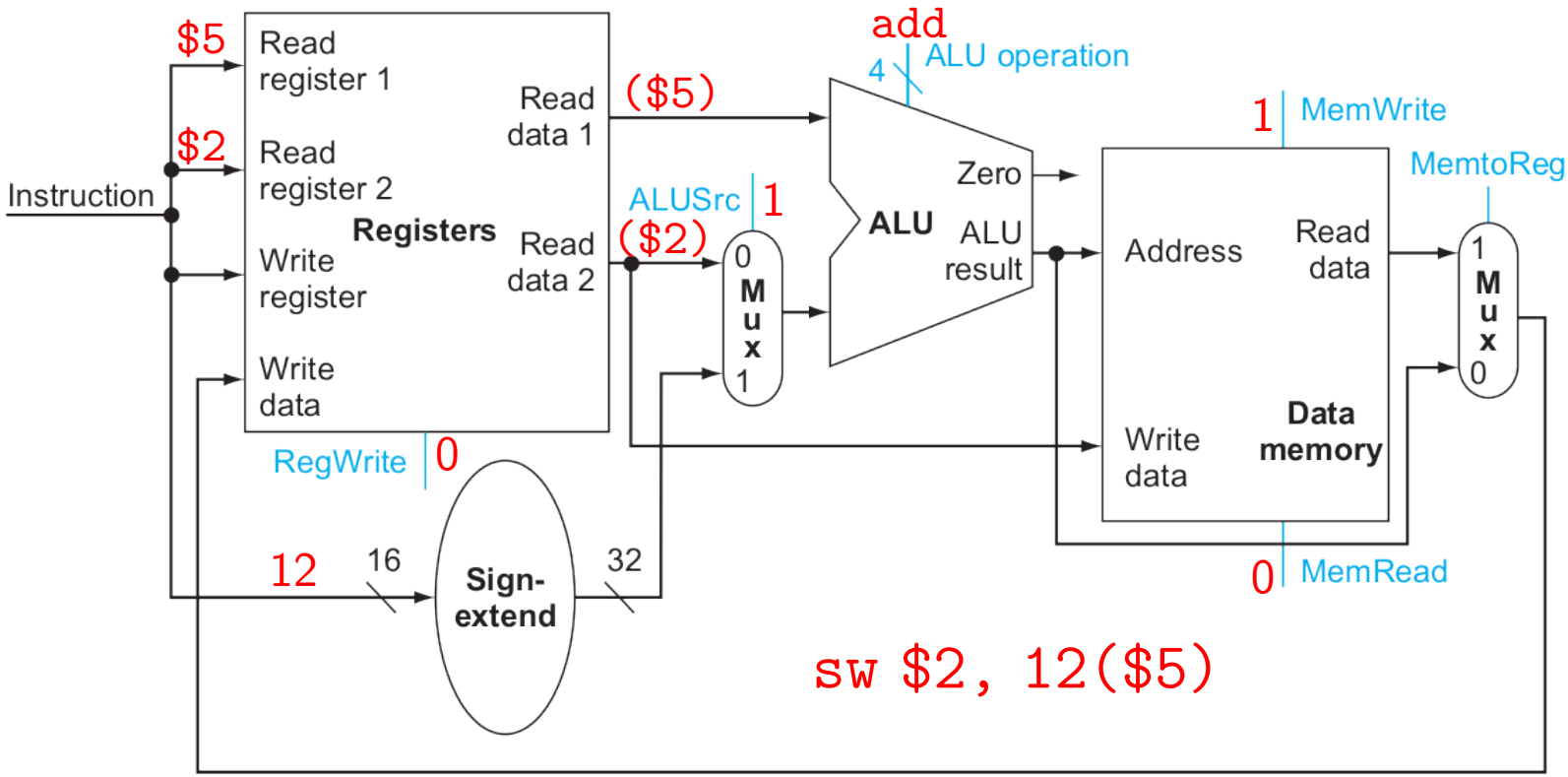


Fonte: Adaptado de [1]

Construindo o *Datapath*

Unindo instruções de memória e tipo-R

- Então...



Fonte: Adaptado de [1]

Construindo o *Datapath*

Unindo tudo

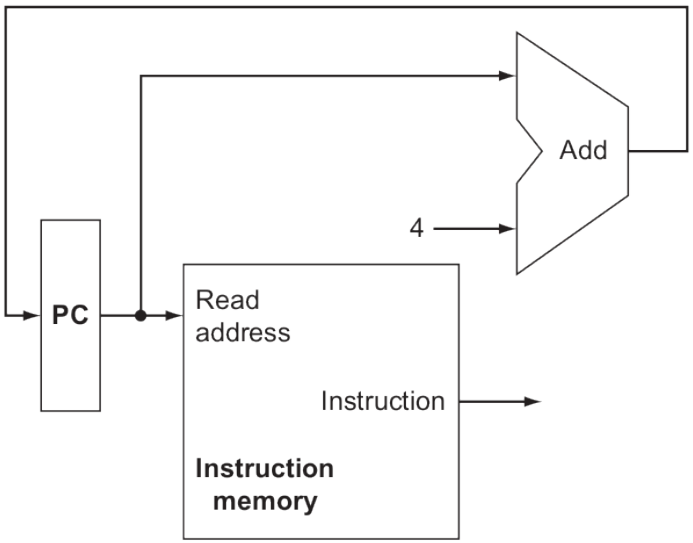
- Que mais falta?

Construindo o *Datapath*

Unindo tudo

- Que mais falta?

Buscar as instruções



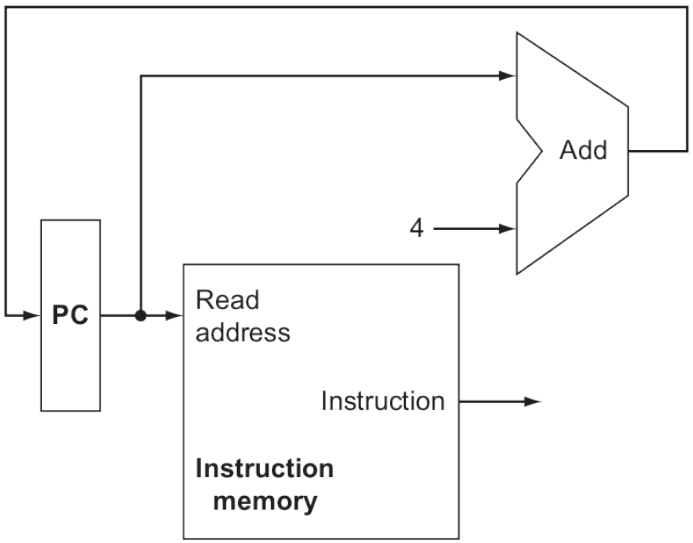
Fonte: [1]

Construindo o *Datapath*

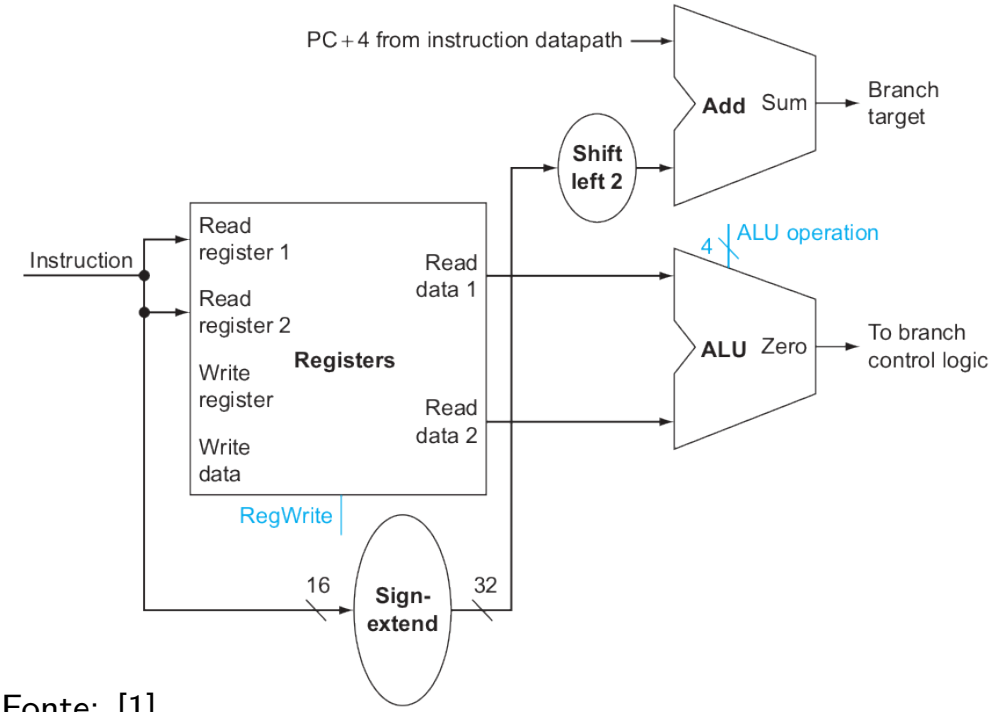
Unindo tudo

- Que mais falta?

Buscar as instruções



beq



Fonte: [1]

Construindo o *Datapath*

Unindo tudo

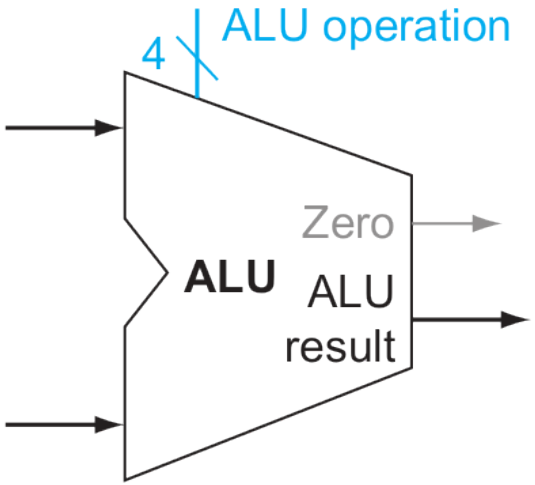
- Para unir estes ao circuito das instruções tipo-R e de memória, contudo, precisamos da unidade de controle
- Esta deve ser capaz de:
  - Gerar um sinal de escrita para cada elemento de estado
  - Gerar um sinal de controle para cada multiplexador
  - Controlar a ALU

Construindo o *Datapath*

Controle da ALU

- Vamos implementar as seguintes funções lógico-aritméticas:

Linhas de Controle da ALU	*Função*
0000	and
0001	or
0010	add
0110	sub
0111	slt

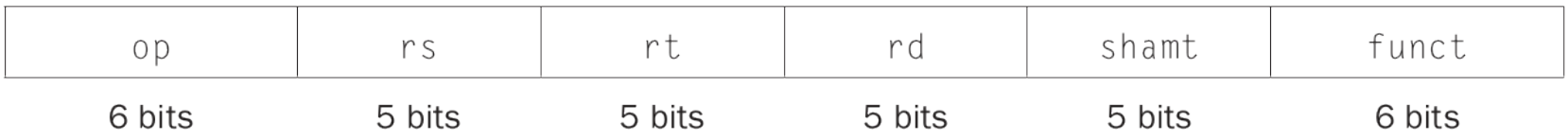


Fonte: [1]

Construindo o *Datapath*

Controle da ALU

- Podemos gerar esses sinais usando como entrada
  - O campo *function* da instrução
  - E um campo de controle de 2 bits, denominado *ALUOp*
    - Indicando se a operação é um `add` (00), `sub` (01), ou determinada pela operação codificada no campo *funct* (10)



Fonte: [1]

Construindo o *Datapath*

Controle da ALU

- Podemos gerar esses sinais usando como entrada
  - O campo *function* da instrução
  - E um campo de controle de 2 bits, denominado *ALUOp*
    - Indicando se a operação é um `add` (00), `sub` (01), ou determinada pela operação codificada no campo *funct* (10)

Fonte: [1]

- O sinal de 4 bits é usado então para controlar diretamente a ALU

Construindo o *Datapath*

Controle da ALU

Instruction opcode	ALUOp	Instruction operation	Funct field	Desired ALU action	ALU control input
LW	00	load word	XXXXXX	add	0010
SW	00	store word	XXXXXX	add	0010
Branch equal	01	branch equal	XXXXXX	subtract	0110
R-type	10	add	100000	add	0010
R-type	10	subtract	100010	subtract	0110
R-type	10	AND	100100	AND	0000
R-type	10	OR	100101	OR	0001
R-type	10	set on less than	101010	set on less than	0111

Fonte: [1]

Construindo o *Datapath*

Controle da ALU

Instruction opcode	ALUOp	Instruction operation	Funct field	Desired ALU action	ALU control input
LW	00	load word	XXXXXX	add	0010
SW	00	store word	XXXXXX	add	0010
Branch equal	01	branch equal	XXXXXX	subtract	0110
R-type	10	add	100000	add	0010
R-type	10	subtract	100010	subtract	0110
R-type	10	AND	100100	AND	0000
R-type	10	OR	100101	OR	0001
R-type	10	set on less than	101010	set on less than	0111

Fonte: [1]

- Note que `lw` e `sw` usam add, `beq` usa `sub`, e as demais operações são definidas por *funct*

Construindo o *Datapath*

Controle da ALU

Instruction opcode	ALUOp	Instruction operation	Funct field	Desired ALU action	ALU control input
LW	00	load word	XXXXXX	add	0010
SW	00	store word	XXXXXX	add	0010
Branch equal	01	branch equal	XXXXXX	subtract	0110
R-type	10	add	100000	add	0010
R-type	10	subtract	100010	subtract	0110
R-type	10	AND	100100	AND	0000
R-type	10	OR	100101	OR	0001
R-type	10	set on less than	101010	set on less than	0111

Fonte: [1]

- Os sinais de *ALUOp* são gerados na unidade de controle principal (mais adiante veremos)

Construindo o *Datapath*

Unindo tudo

- Temos então o controle da ALU
  - Que recebe sua entrada da instrução e do controle principal

Construindo o *Datapath*

Unindo tudo

- Temos então o controle da ALU
  - Que recebe sua entrada da instrução e do controle principal
- E como cada instrução se conecta ao *datapath*?

Construindo o *Datapath*

Unindo tudo

- Temos então o controle da ALU
  - Que recebe sua entrada da instrução e do controle principal
- E como cada instrução se conecta ao *datapath*?

Field	0	rs	rt	rd	shamt	funct
Bit positions	31:26	25:21	20:16	15:11	10:6	5:0

a. R-type instruction

Field	35 or 43	rs	rt	address
Bit positions	31:26	25:21	20:16	15:0

b. Load or store instruction

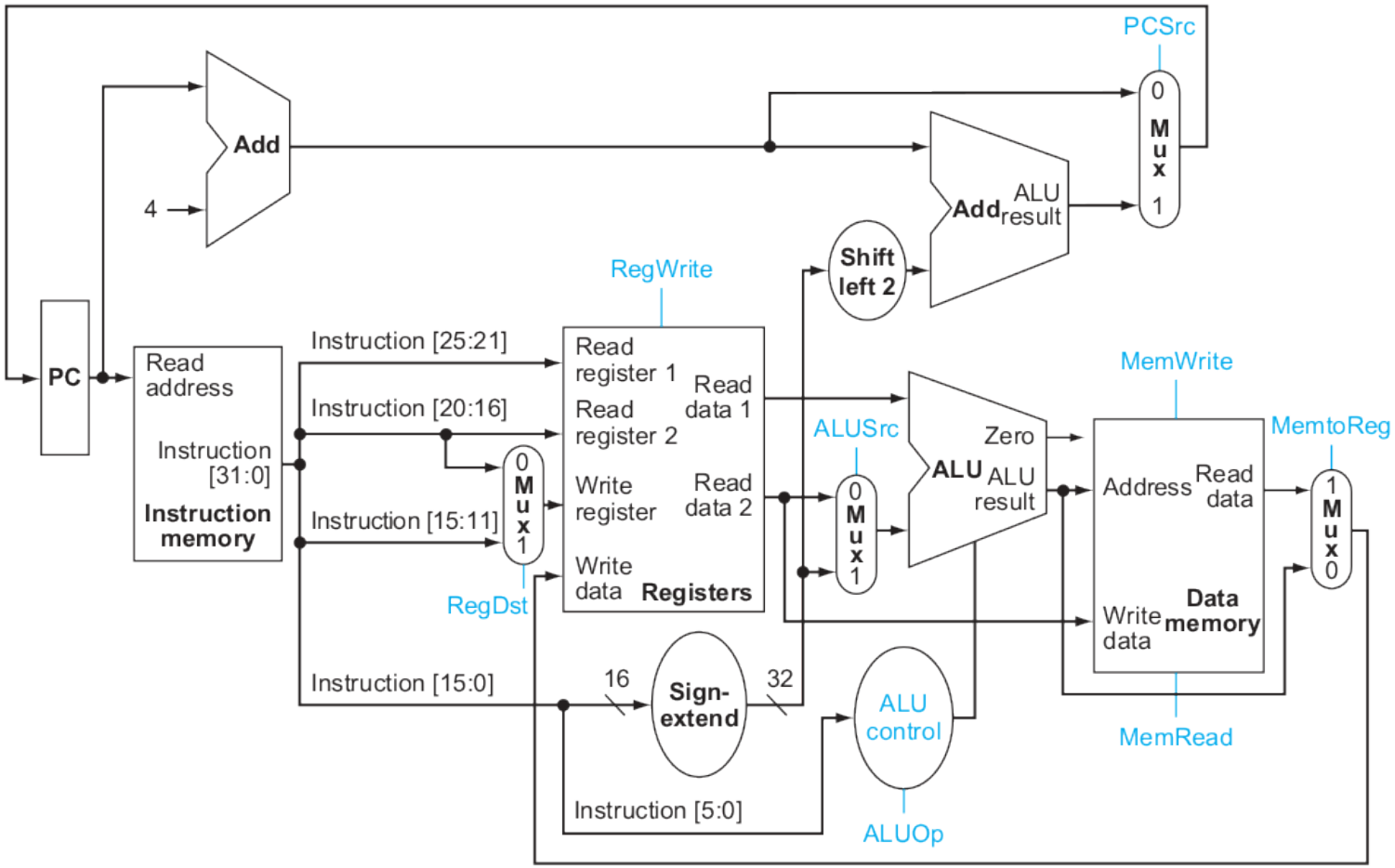
Field	4	rs	rt	address
Bit positions	31:26	25:21	20:16	15:0

c. Branch instruction

Fonte: [1]

Construindo o *Datapath*

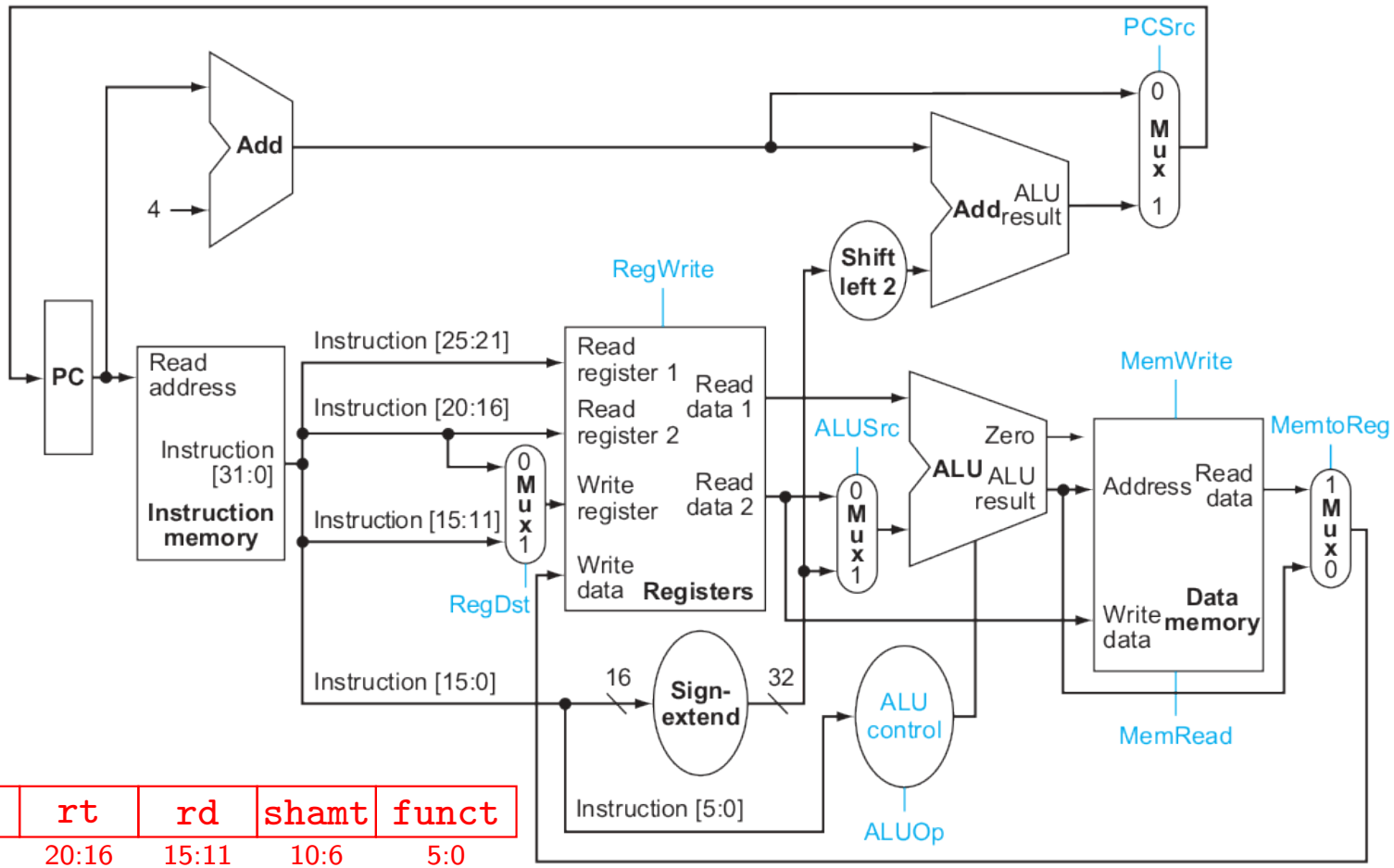
Unindo tudo



Fonte: Adaptado de [1]

Construindo o *Datapath*

Unindo tudo



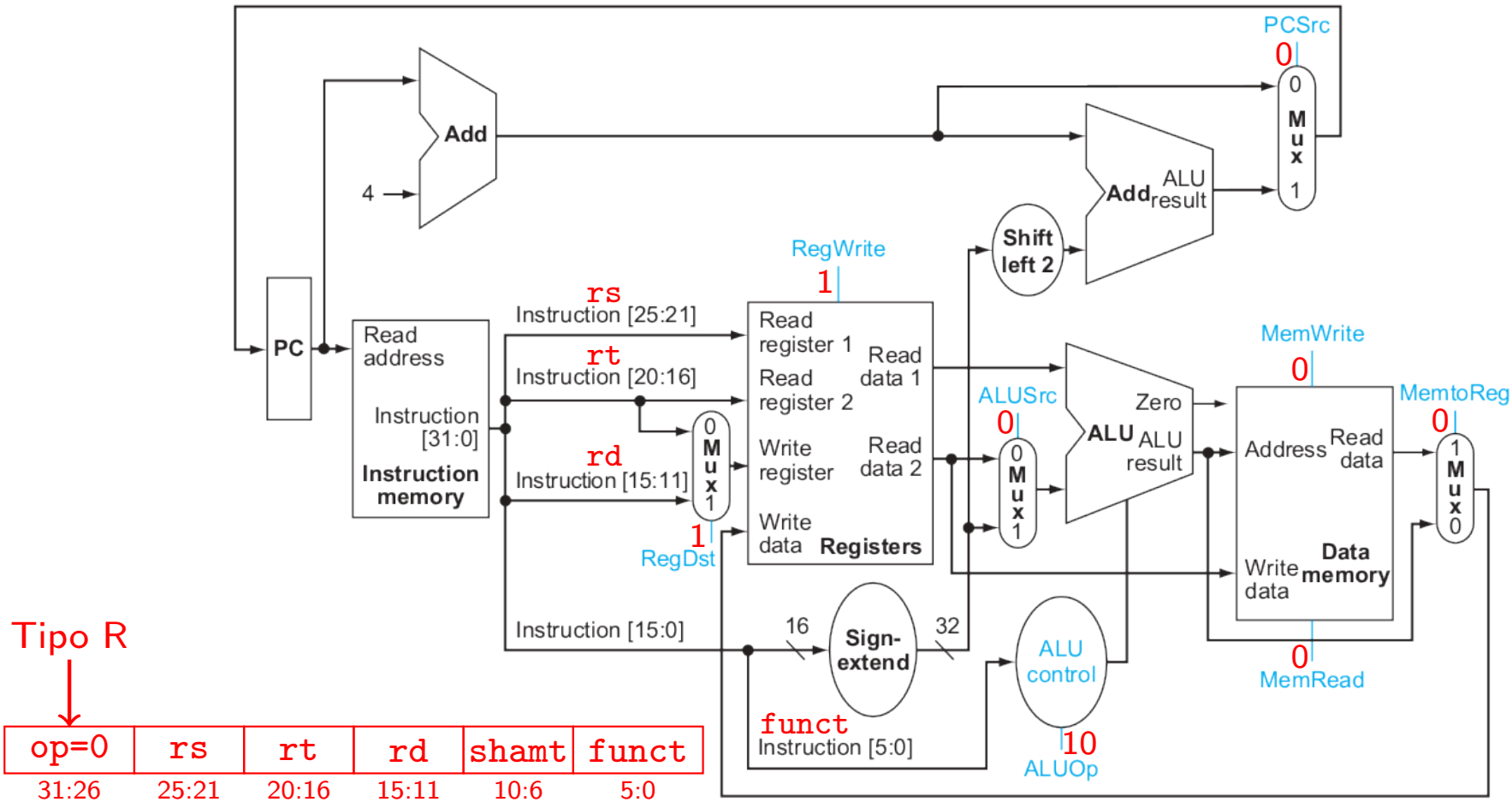
Tipo R

op=0	rs	rt	rd	shamt	funct
31:26	25:21	20:16	15:11	10:6	5:0

Fonte: Adaptado de [1]

Construindo o *Datapath*

Unindo tudo



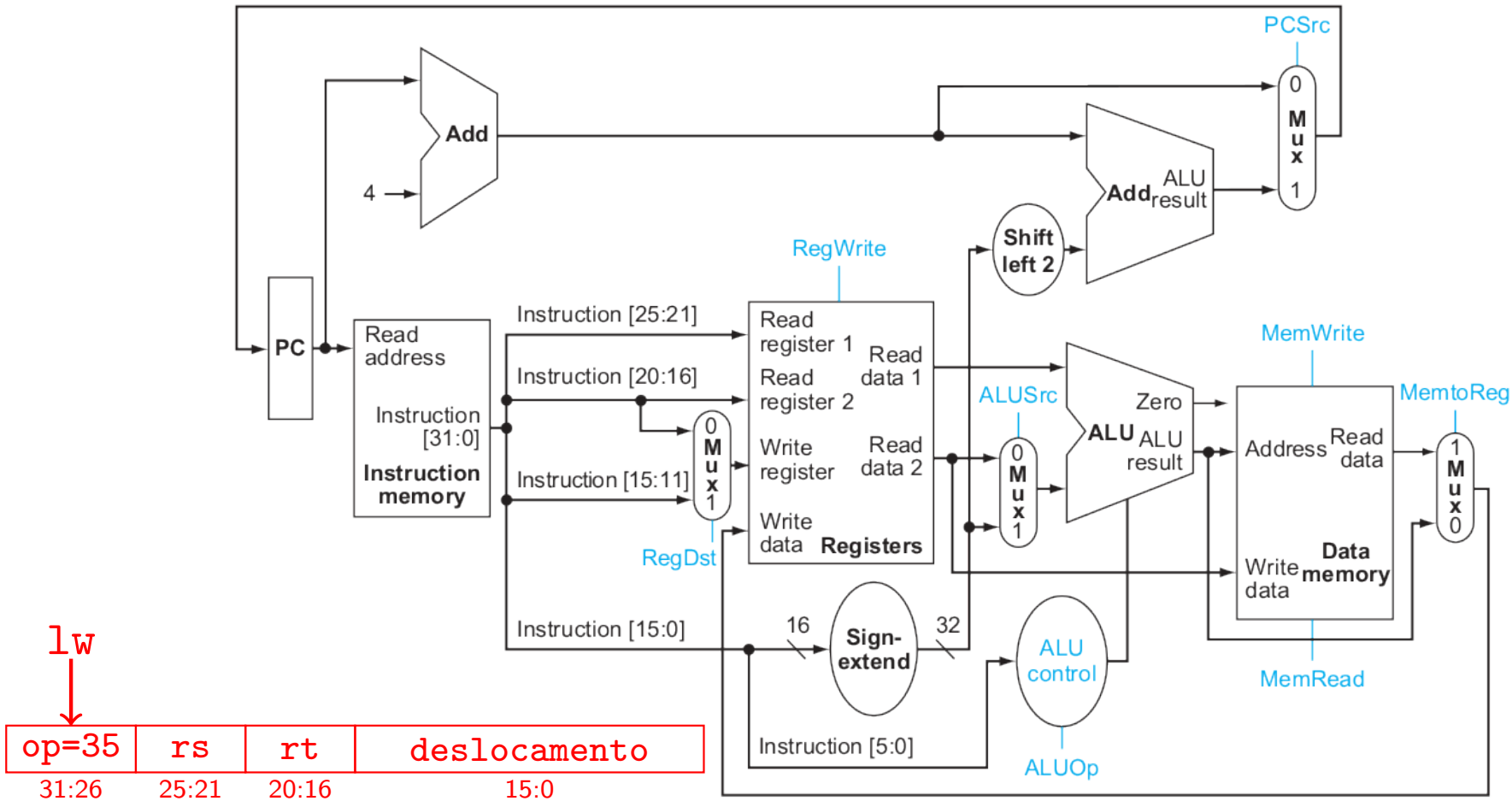
Tipo R

op=0	rs	rt	rd	shamt	funct
31:26	25:21	20:16	15:11	10:6	5:0

Fonte: Adaptado de [1]

Construindo o *Datapath*

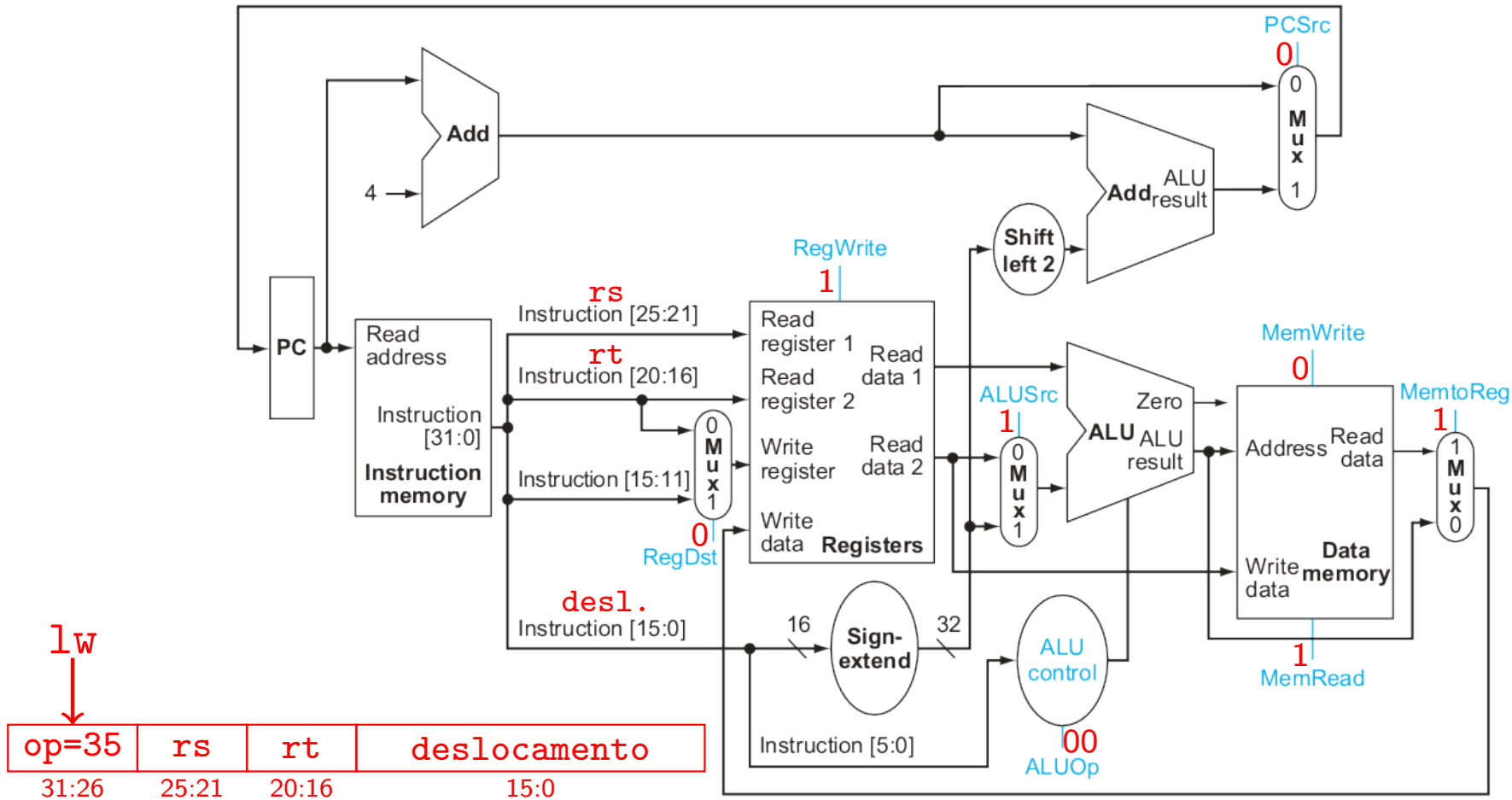
Unindo tudo



Fonte: Adaptado de [1]

Construindo o *Datapath*

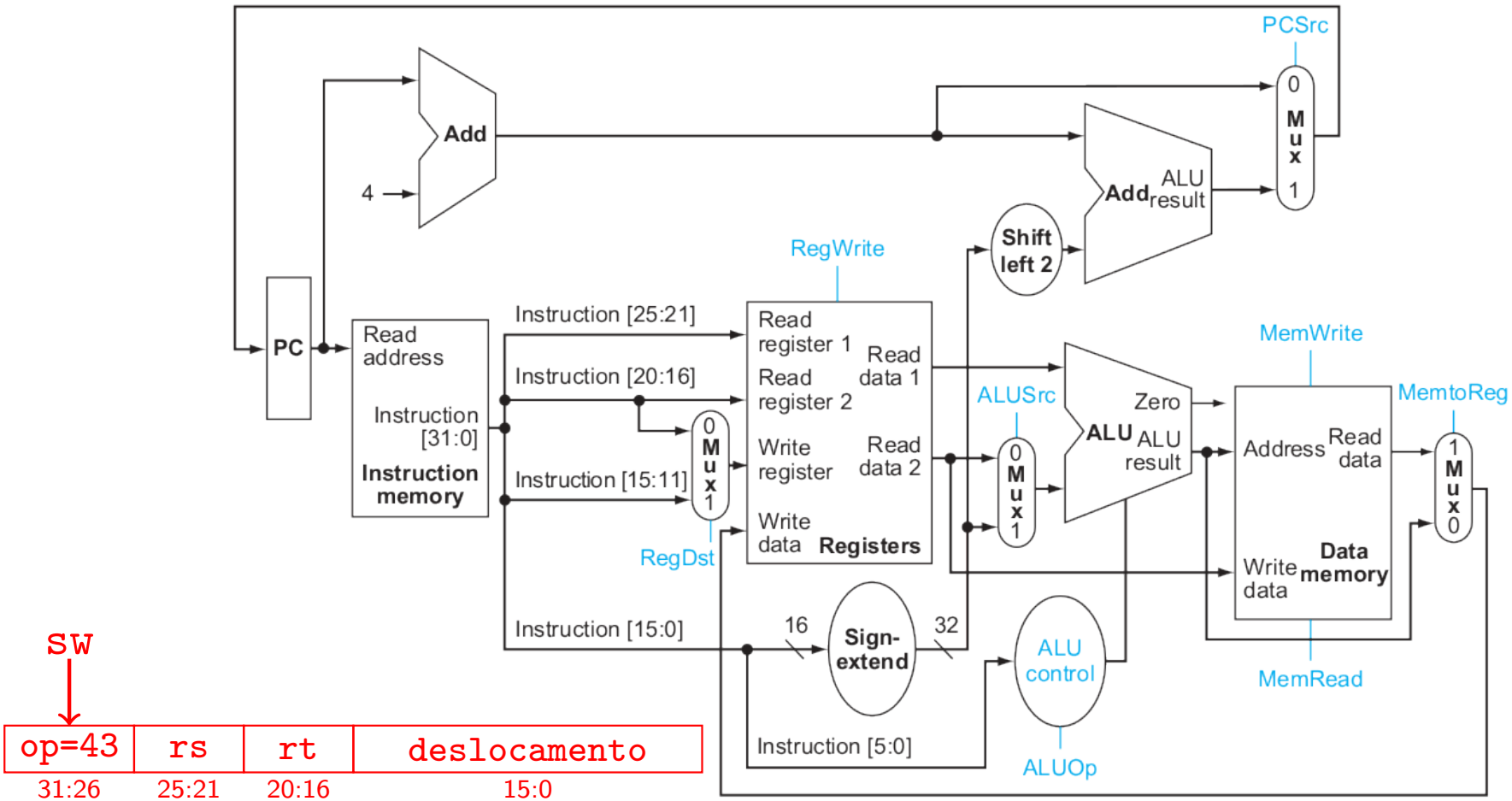
Unindo tudo



Fonte: Adaptado de [1]

Construindo o *Datapath*

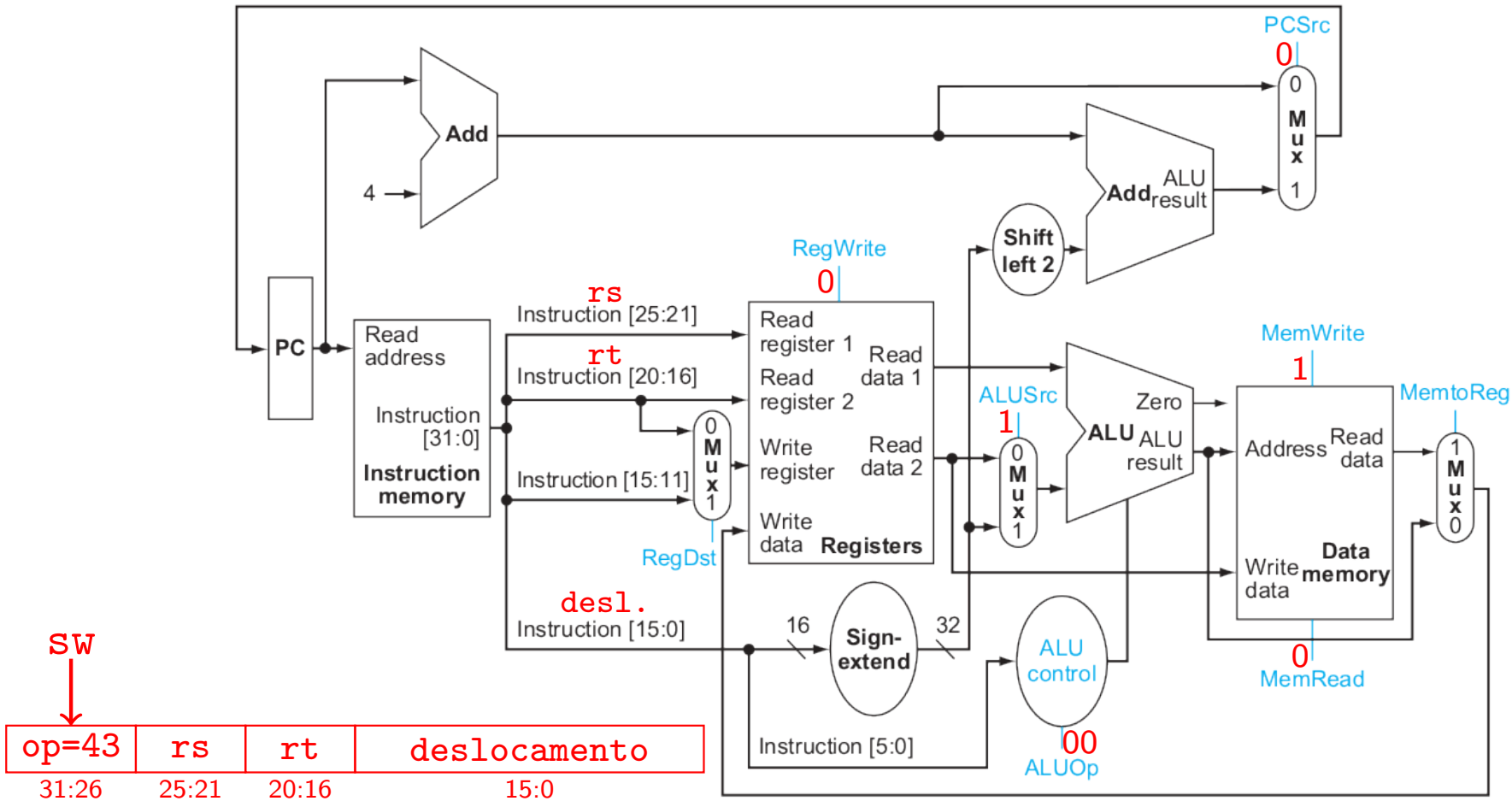
Unindo tudo



Fonte: Adaptado de [1]

Construindo o *Datapath*

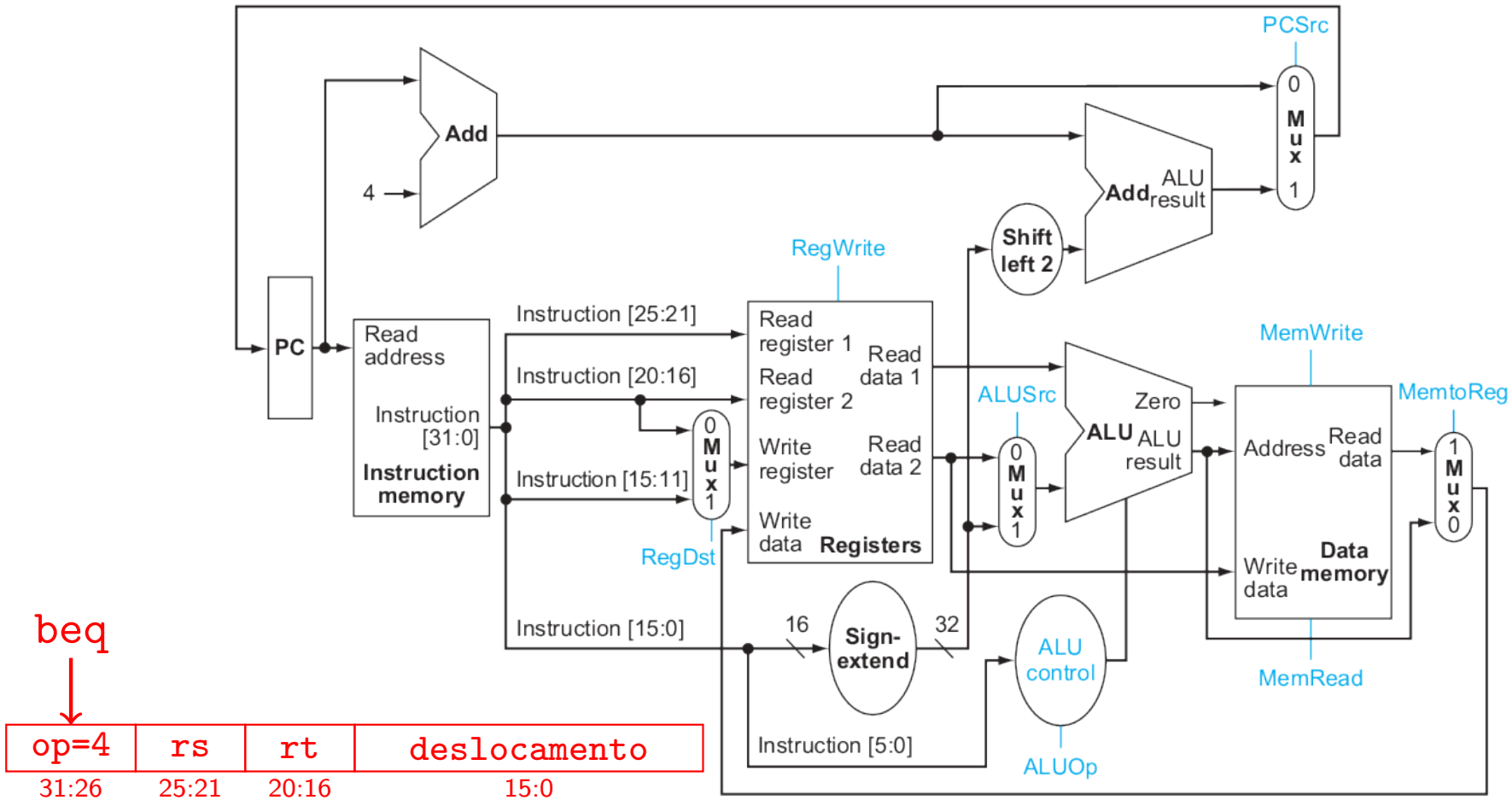
Unindo tudo



Fonte: Adaptado de [1]

Construindo o *Datapath*

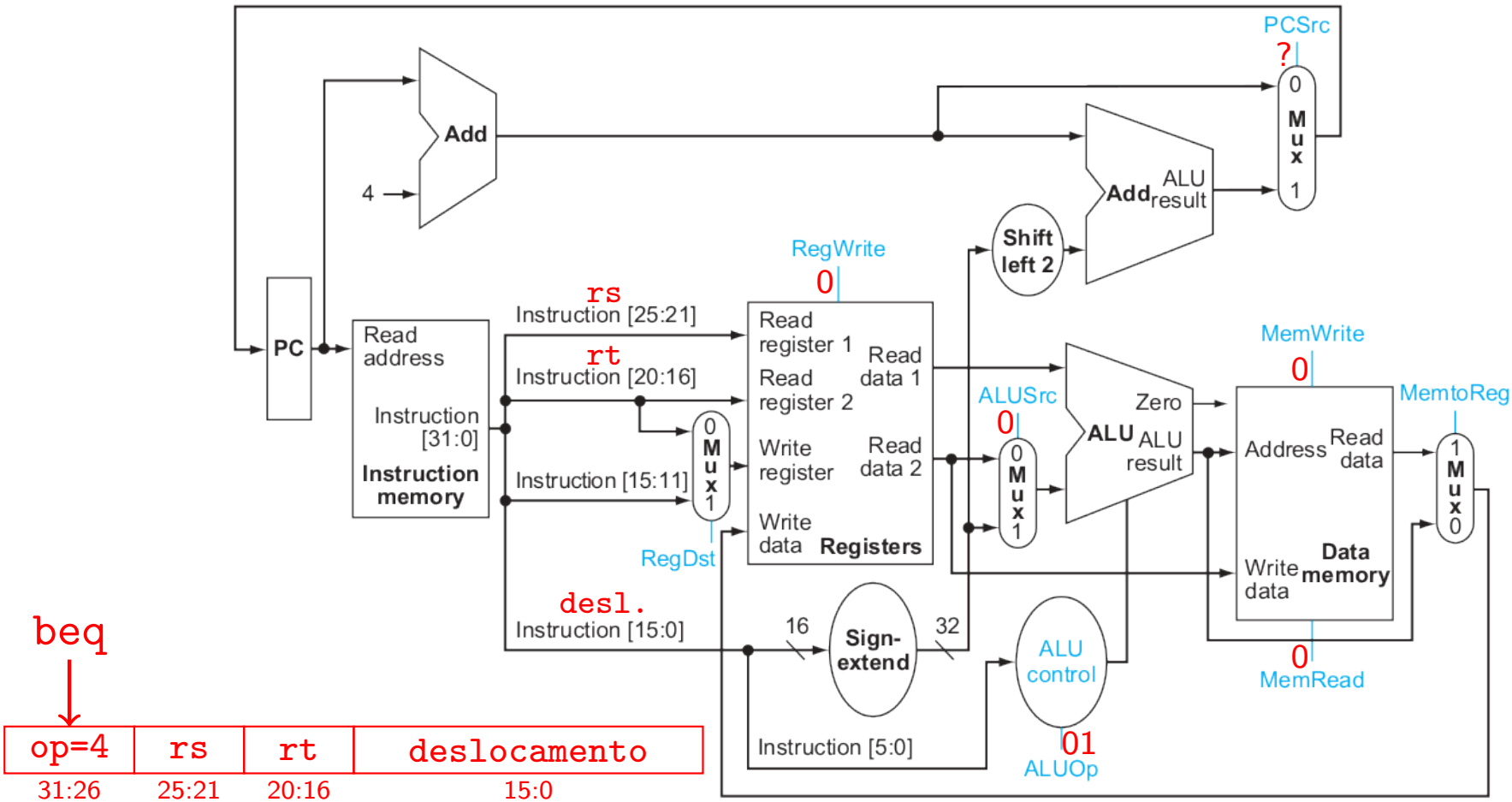
Unindo tudo



Fonte: Adaptado de [1]

Construindo o *Datapath*

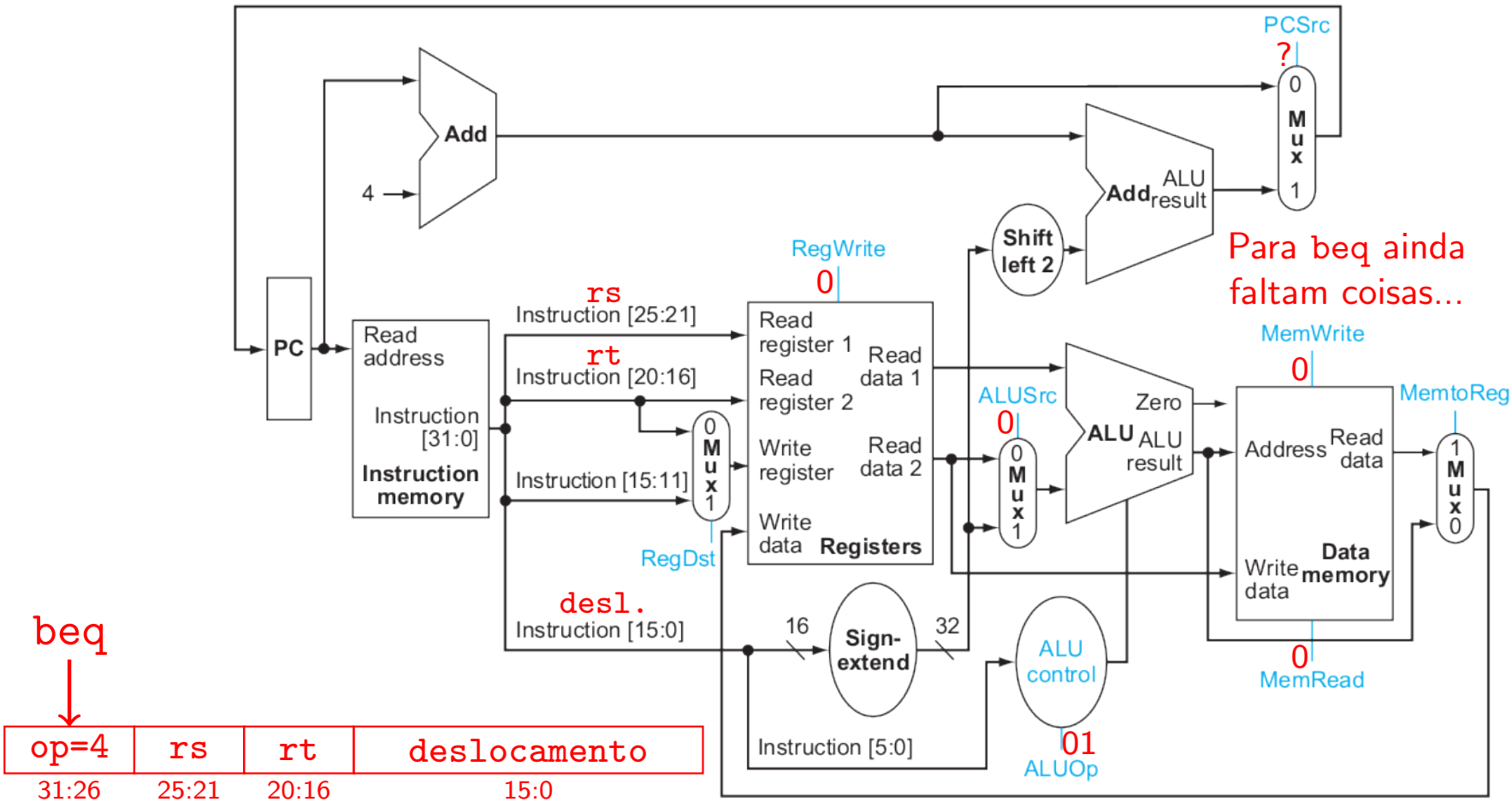
Unindo tudo



Fonte: Adaptado de [1]

Construindo o *Datapath*

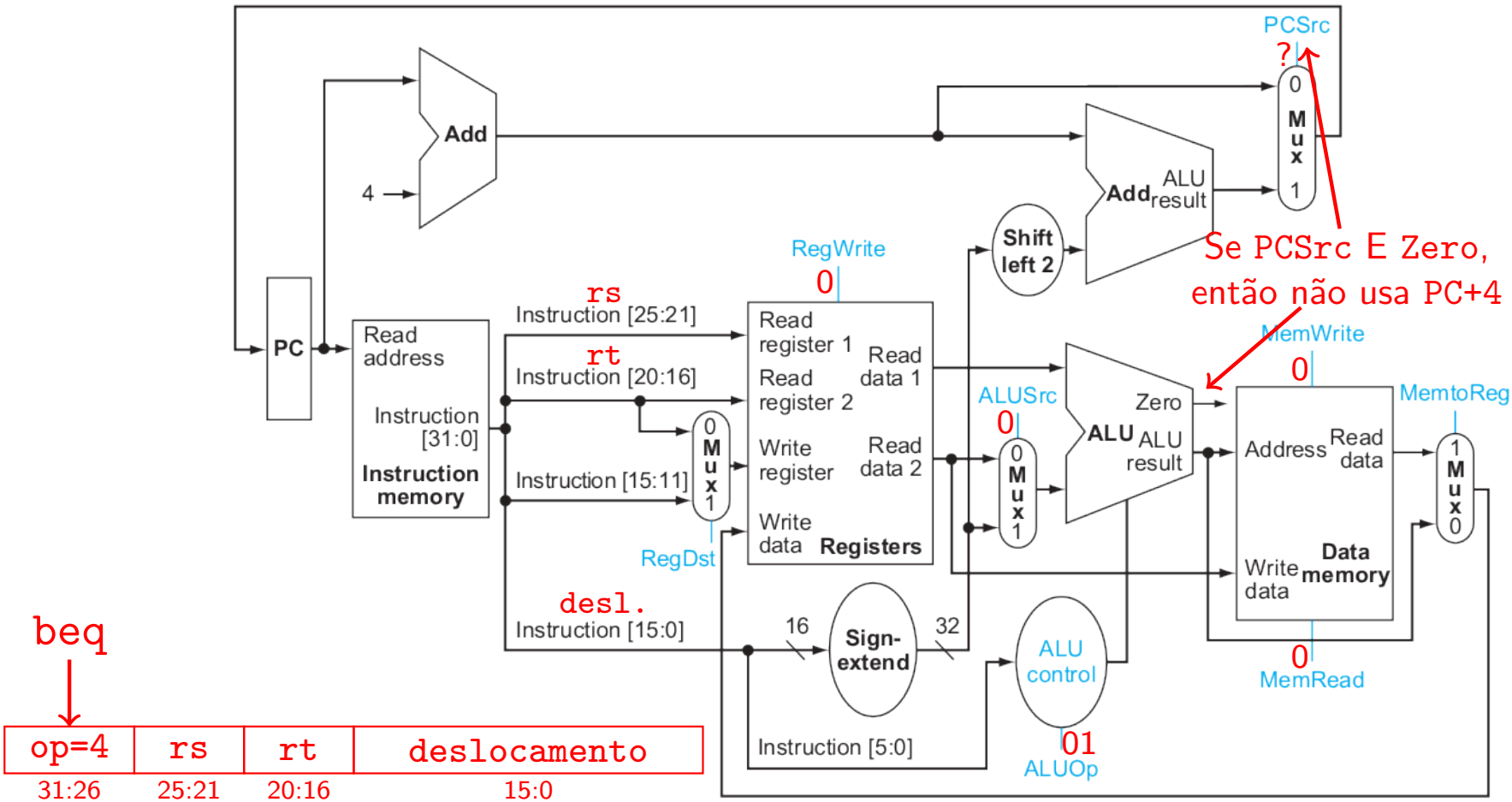
Unindo tudo



Fonte: Adaptado de [1]

Construindo o *Datapath*

Unindo tudo

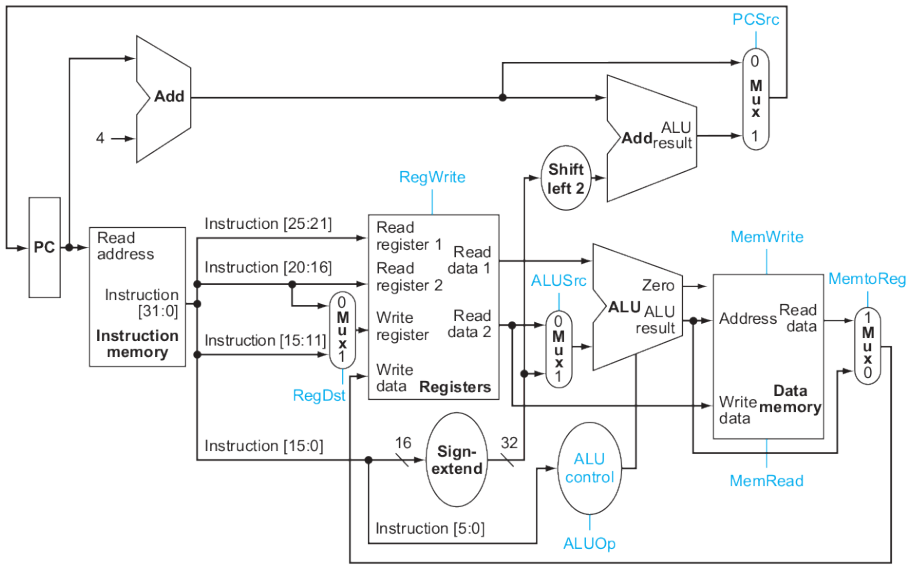


Fonte: Adaptado de [1]

Construindo o *Datapath*

Unindo tudo: unidade de controle principal

- Há também uma série de sinais de controle no *datapath*
  - *PCSrc, RegWrite, ALUSrc, MemWrite, MemRead, MemtoReg, RegDst e ALUOp*

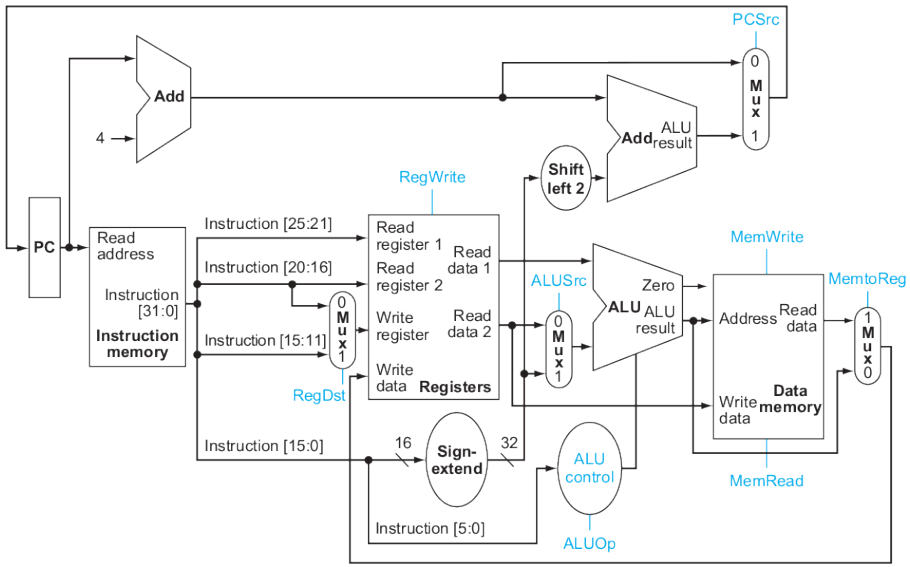


Fonte: Adaptado de [1]

Construindo o *Datapath*

Unindo tudo: unidade de controle principal

- Há também uma série de sinais de controle no *datapath*
  - *PCSrc*, *RegWrite*, *ALUSrc*, *MemWrite*, *MemRead*, *MemtoReg*, *RegDst* e *ALUOp*



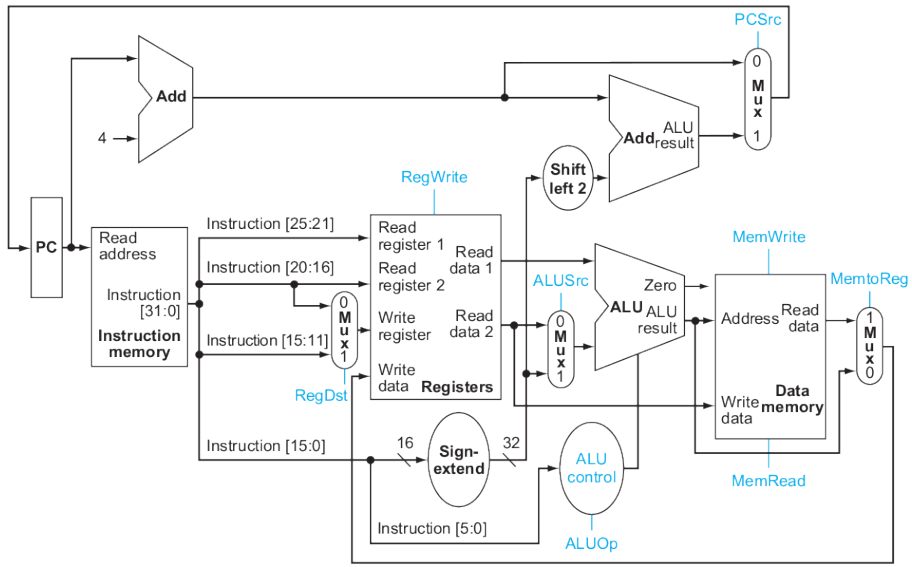
Fonte: Adaptado de [1]

- Além do sinal do *clock*, implícito em todo elemento de estado

Construindo o *Datapath*

Unindo tudo: unidade de controle principal

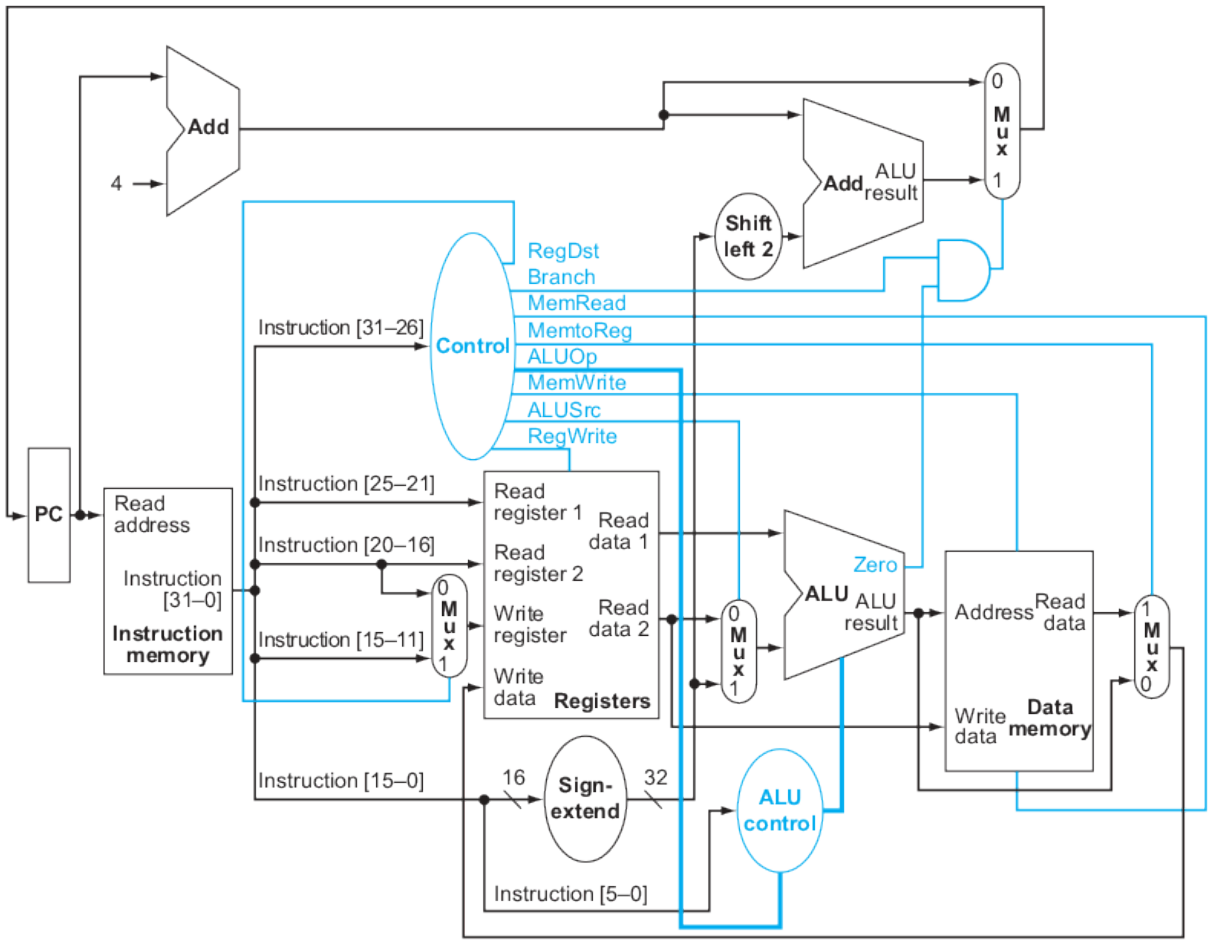
- À exceção desse sinal do *clock*, todos os demais são controlados pela unidade de controle principal
  - Usando como entrada os 6 bits de *opcode* da instrução (não mostrados até agora)



Fonte: Adaptado de [1]

Construindo o *Datapath*

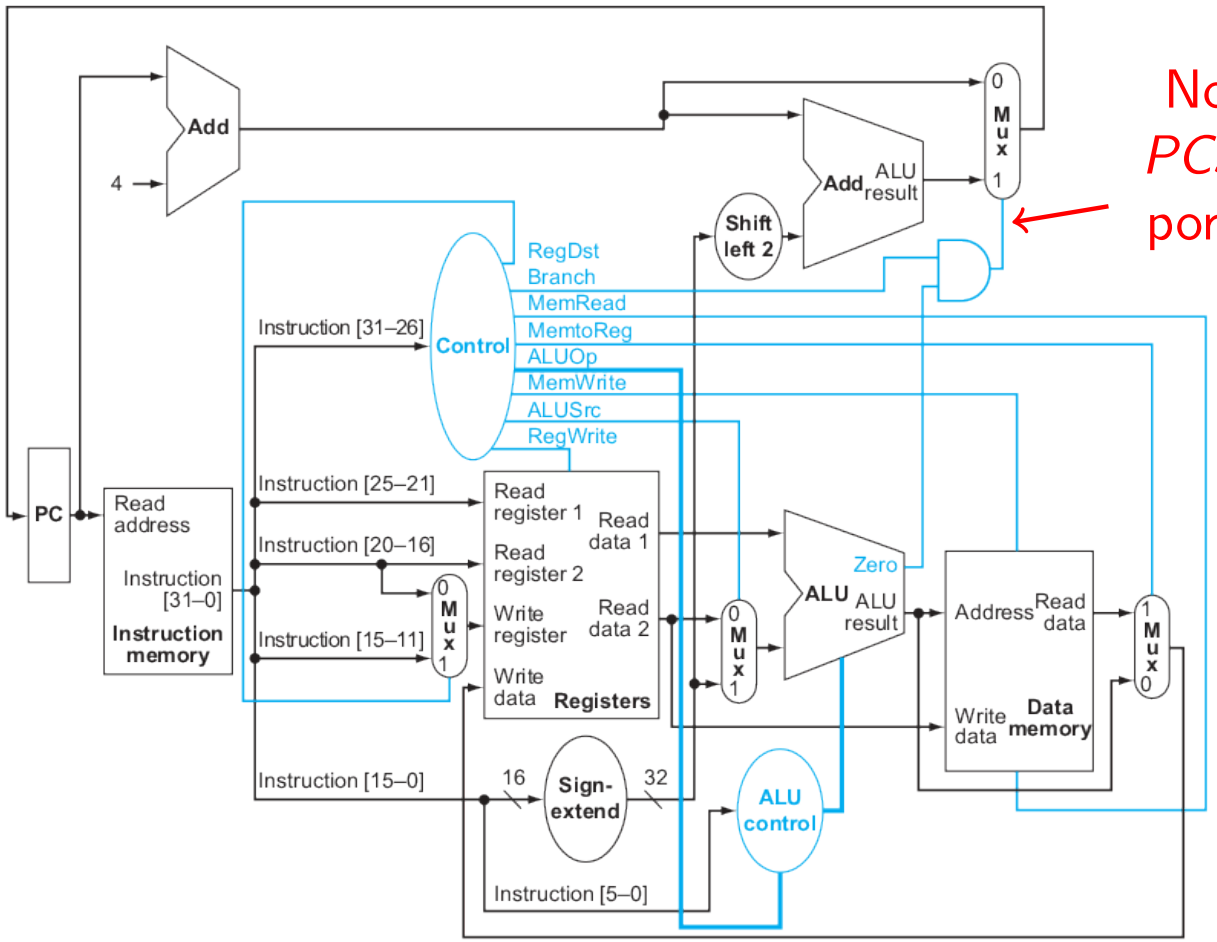
Unindo tudo: unidade de controle principal



Fonte: [1]

Construindo o *Datapath*

Unindo tudo: unidade de controle principal

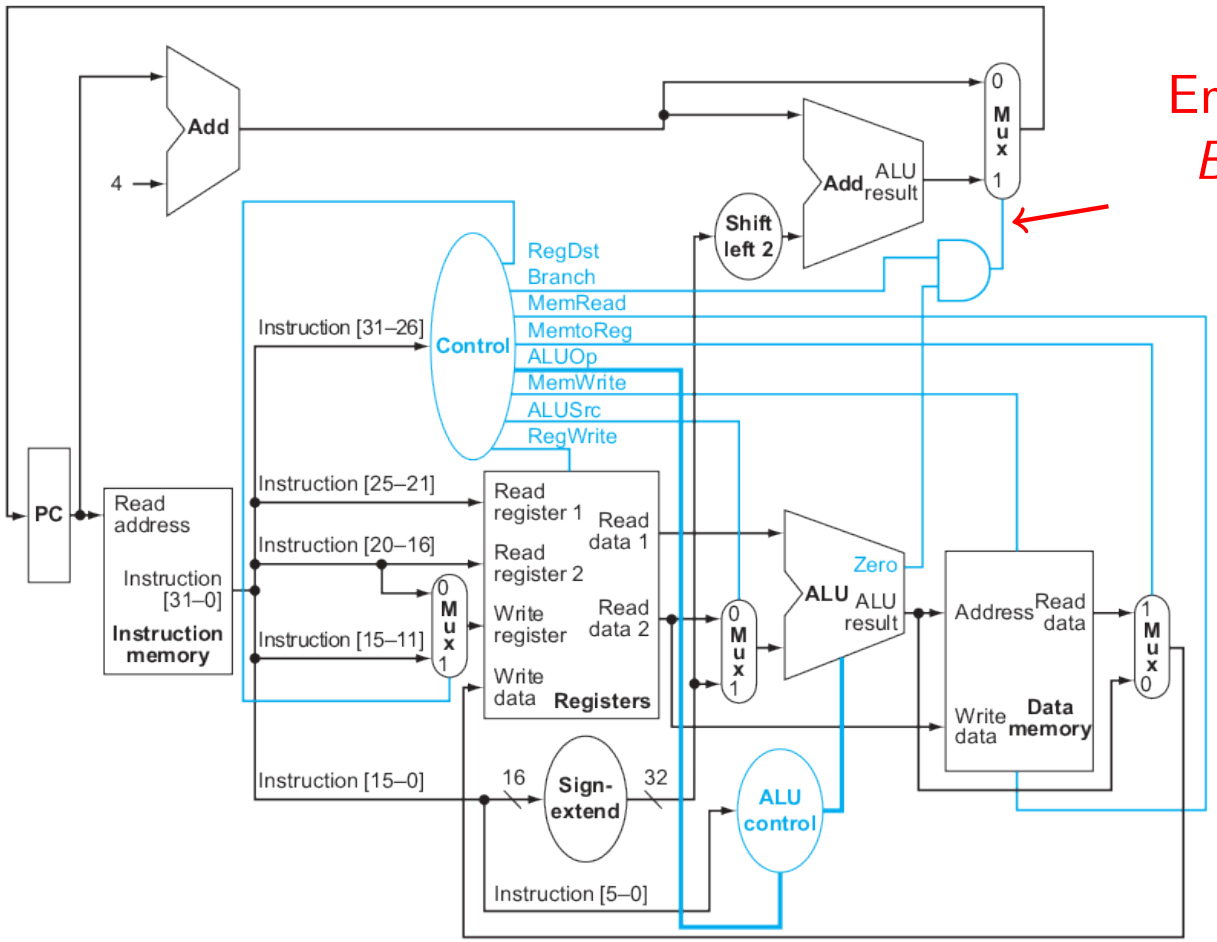


Note que o sinal de *PCSrc* foi substituído por *Branch && Zero*

Fonte: [1]

Construindo o *Datapath*

Unindo tudo: unidade de controle principal

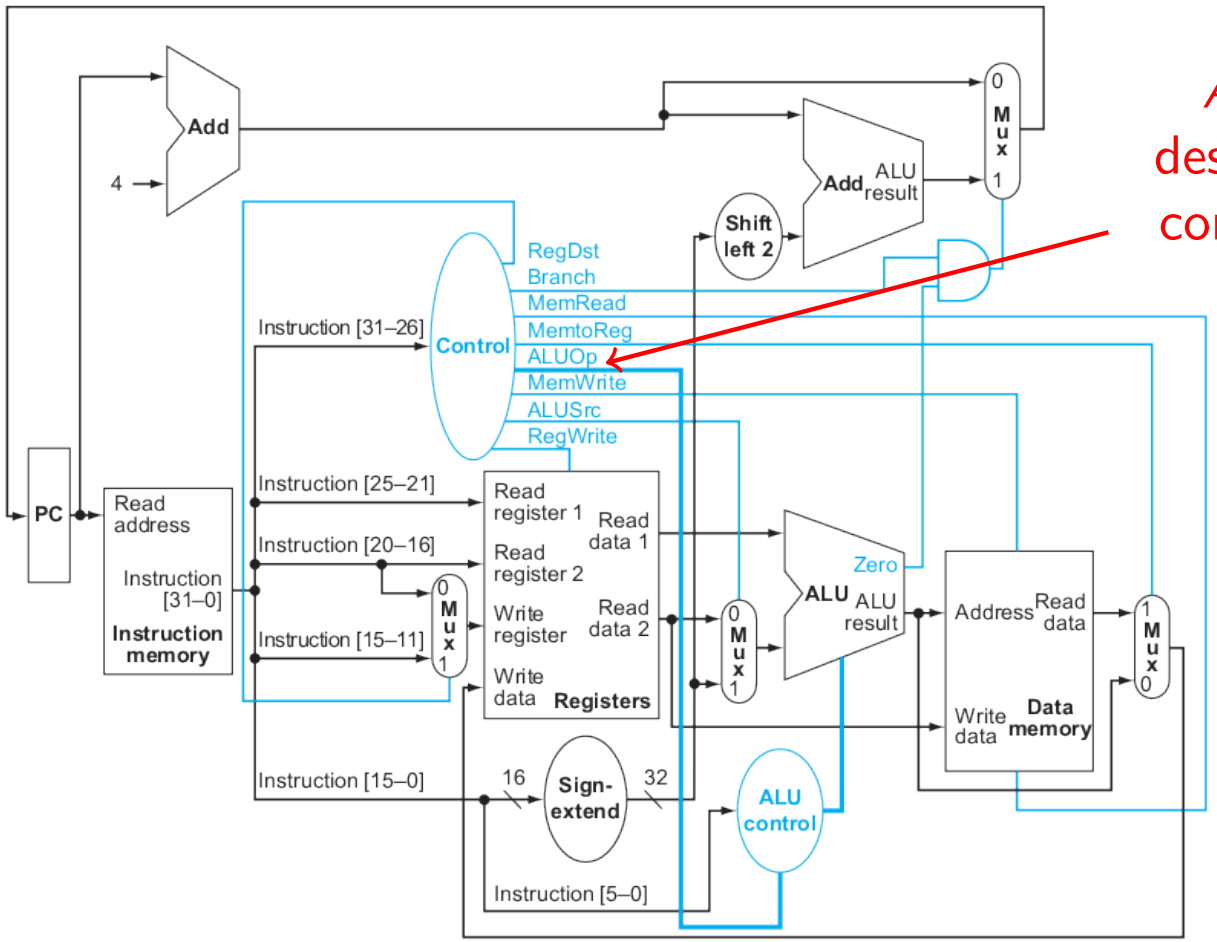


Em instruções beq, *Branch* é feito 1, nas demais é 0

Fonte: [1]

Construindo o *Datapath*

Unindo tudo: unidade de controle principal



ALUOp está em destaque por ser um conjunto de 2 sinais

Fonte: [1]

Construindo o *Datapath*

Unidade de controle: Tabela verdade

Input or output	Signal name	R-format	lw	sw	beq
Inputs	Op5	0	1	1	0
	Op4	0	0	0	0
	Op3	0	0	1	0
	Op2	0	0	0	1
	Op1	0	1	1	0
	Op0	0	1	1	0
Outputs	RegDst	1	0	X	X
	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

Fonte: [1]

Construindo o *Datapath*

Unidade de controle: Tabela verdade

Input or output	Signal name	R-format	lw	sw	beq
Inputs Op[5:0] corresponde aos bits 31:26 (*opcode*)	Op5	0	1	1	0
	Op4	0	0	0	0
	Op3	0	0	1	0
	Op2	0	0	0	1
	Op1	0	1	1	0
	Op0	0	1	1	0
Outputs	RegDst	1	0	X	X
	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

Fonte: [1]

Construindo o *Datapath*

Unidade de controle: Tabela verdade

Input or output	Signal name	R-format	lw	sw	beq
Inputs	Op5	0	1	1	0
	Op4	0	0	0	0
	Op3	0	0	1	0
	Op2	0	0	0	1
	Op1	0	1	1	0
	Op0	0	1	1	0
Outputs	RegDst	1	0	X	X
	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

X significa que não importa o valor do sinal

Fonte: [1]

Construindo o *Datapath*

Unindo tudo: jumps

- Falta ainda implementar `j` (*Opcode* 2)

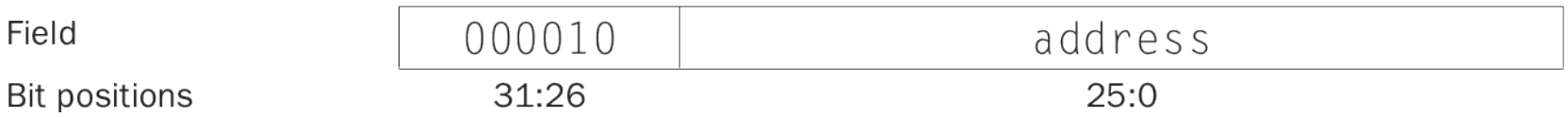
Field	000010	address
Bit positions	31:26	25:0

Fonte: [1]

Construindo o *Datapath*

Unindo tudo: jumps

- Falta ainda implementar `j` (*Opcode* 2)

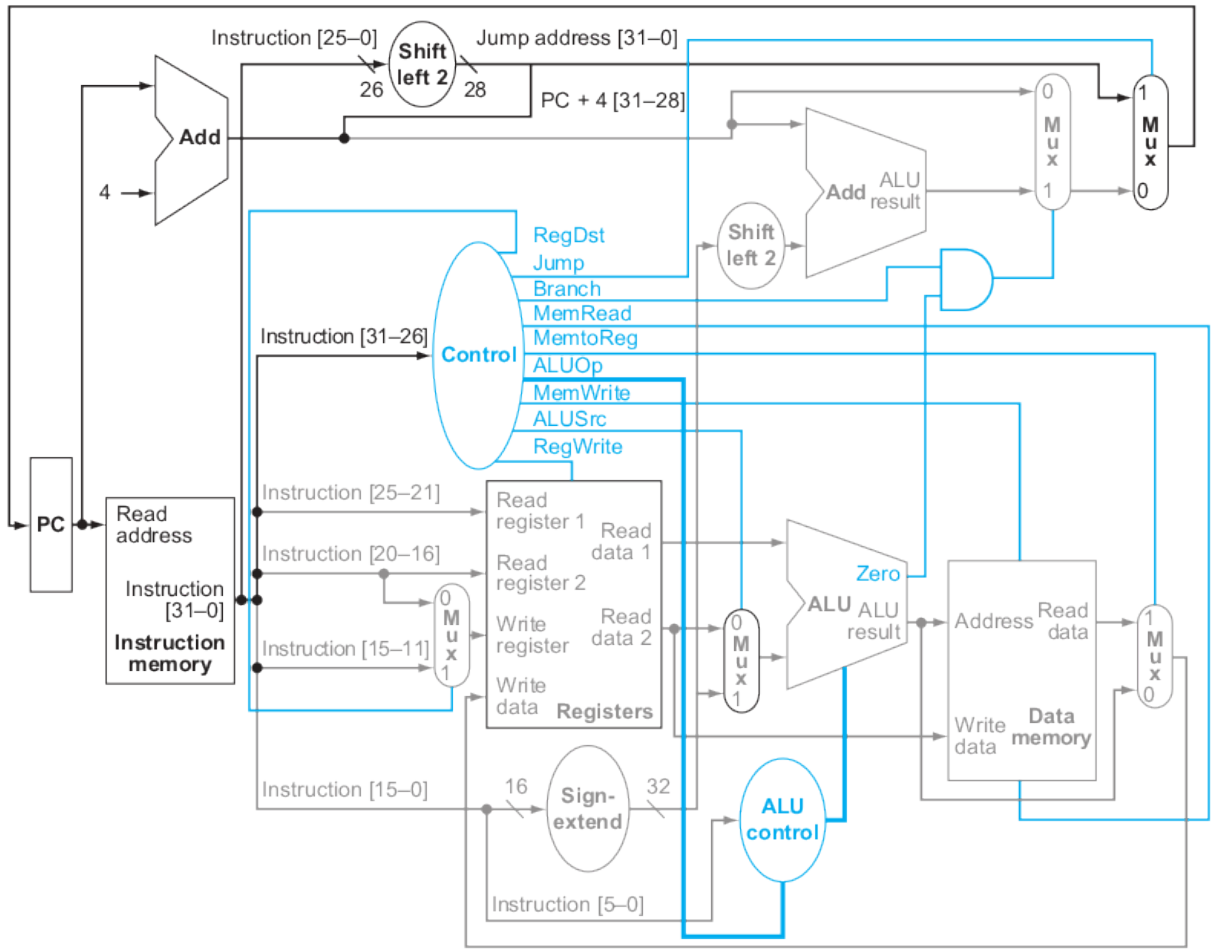


Fonte: [1]

- Para isso, temos que
  - Deslocar, em 2 bits à esquerda, os 26 bits do campo de endereço da instrução
  - Substituir os 28 bits menos significativos de `PC+4` por esses bits deslocados

Construindo o *Datapath*

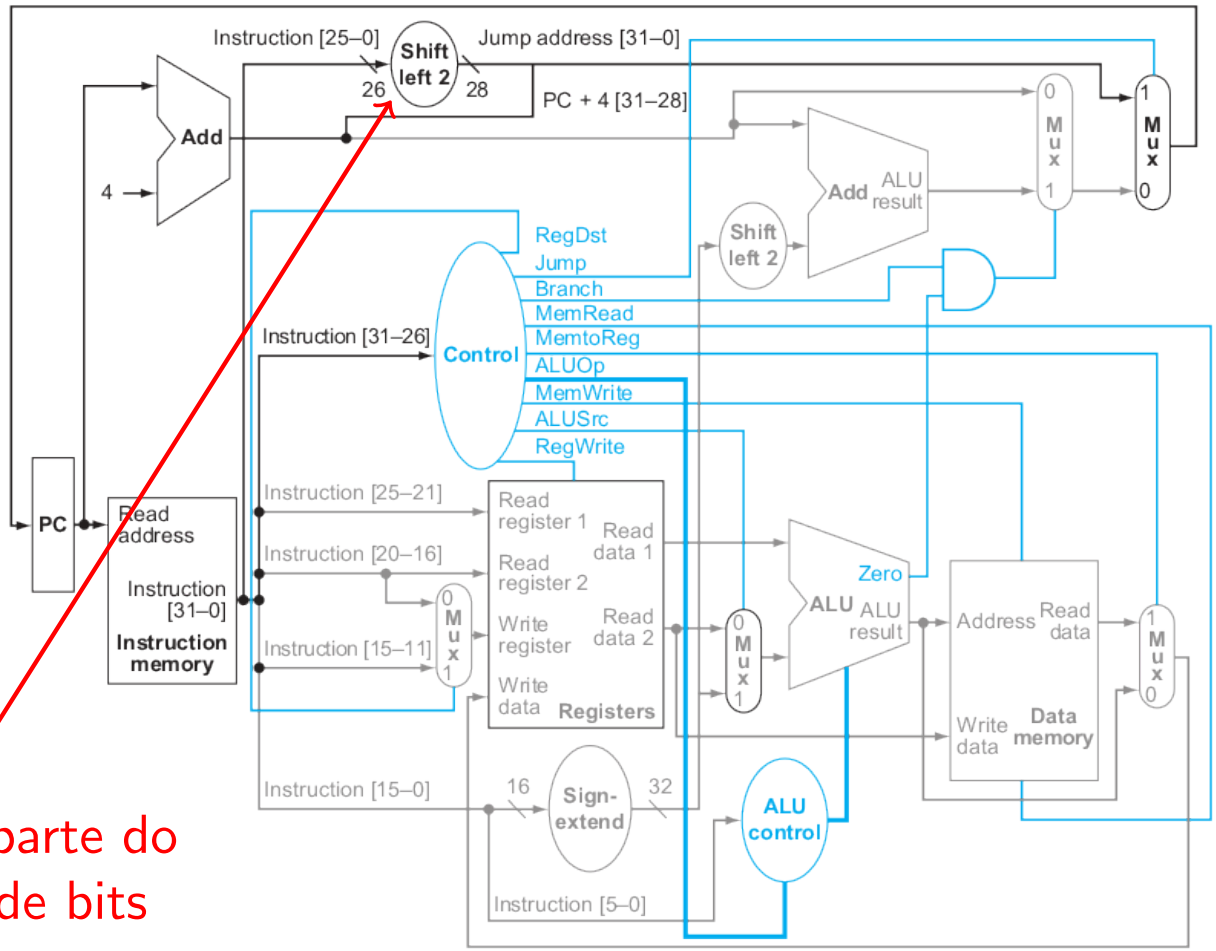
Unindo tudo: jumps



Fonte: [1]

Construindo o *Datapath*

Unindo tudo: jumps

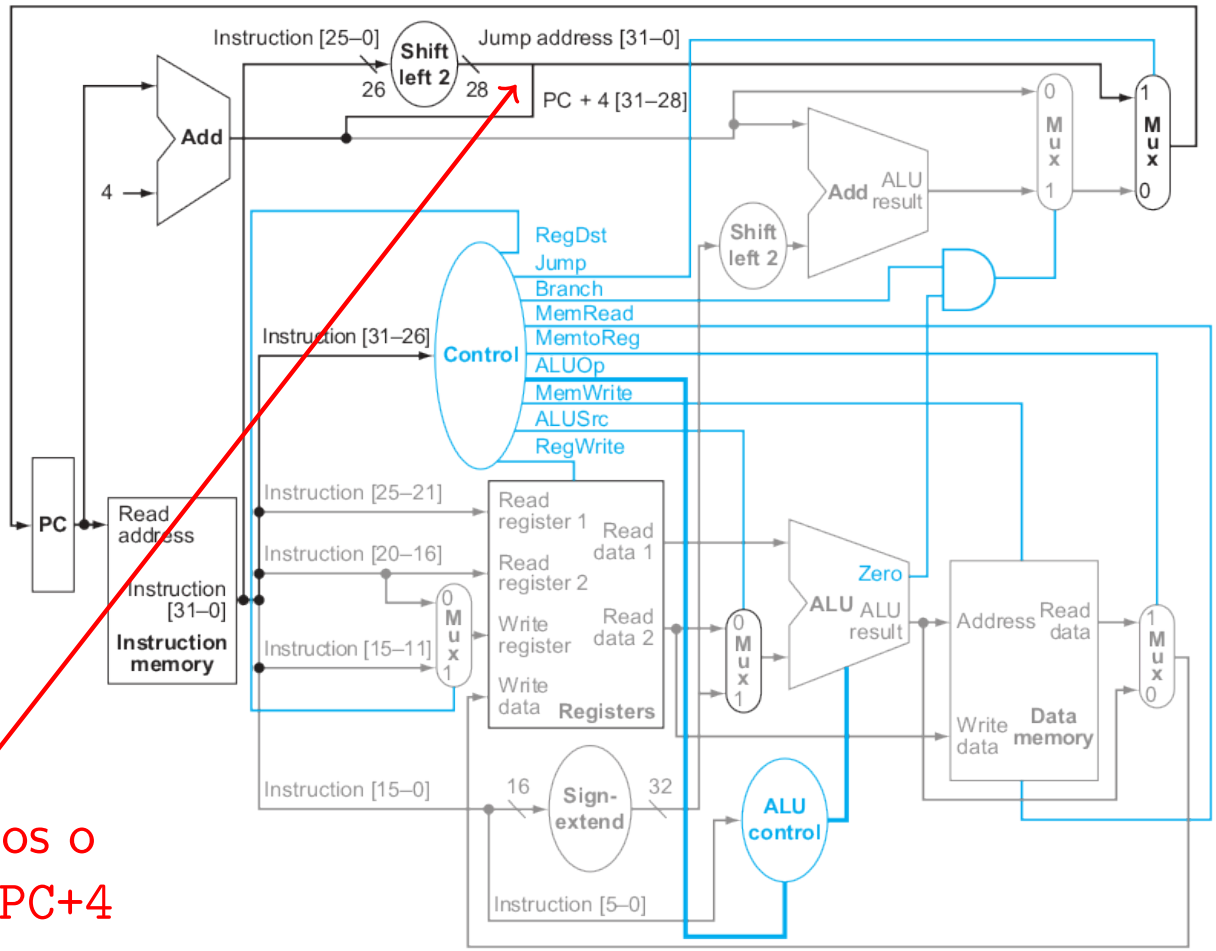


Adicionamos a parte do deslocamento de bits

Fonte: [1]

Construindo o *Datapath*

Unindo tudo: jumps

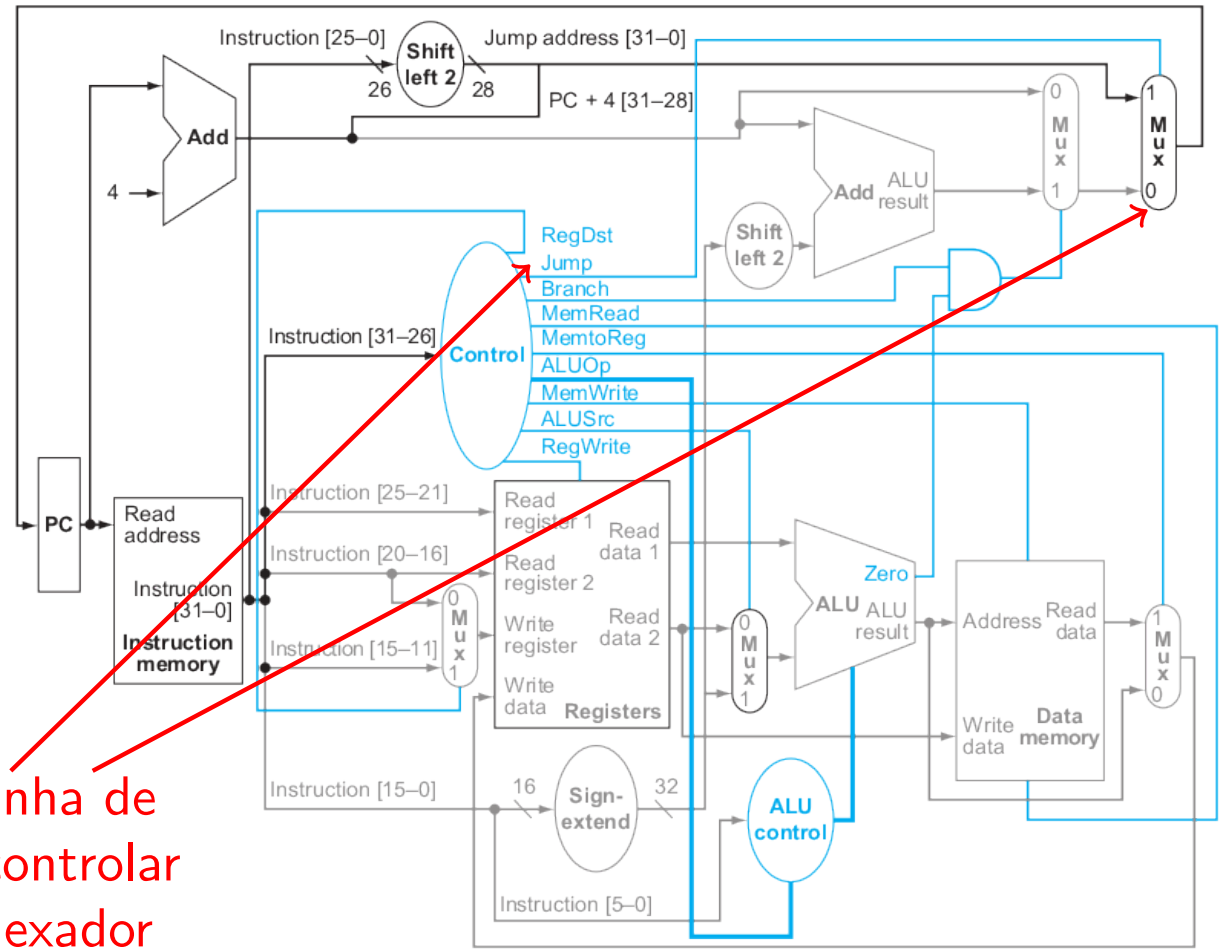


Concatenamos o resultado ao PC+4

Fonte: [1]

Construindo o *Datapath*

Unindo tudo: jumps

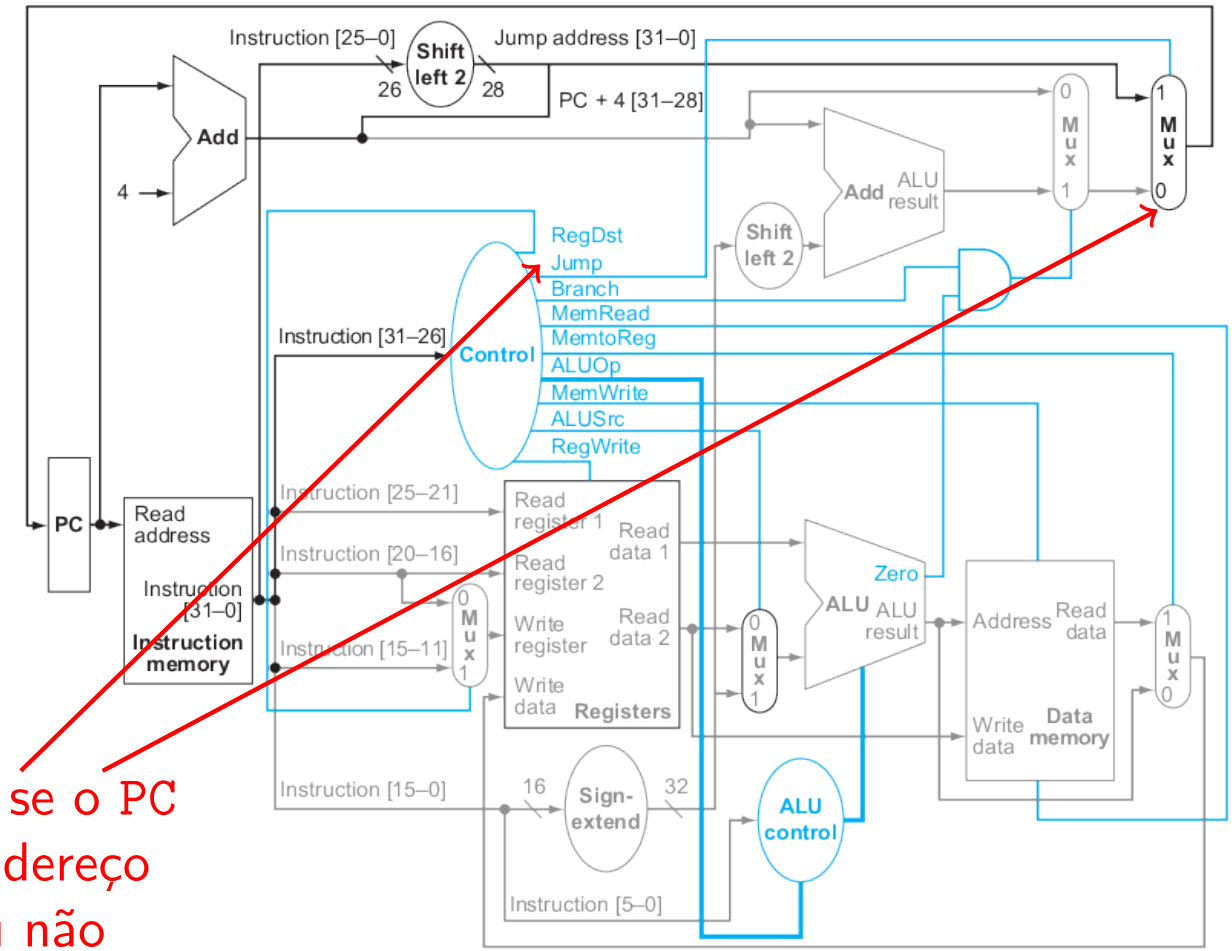


Adicionamos linha de controle, para controlar o novo multiplexador

Fonte: [1]

Construindo o *Datapath*

Unindo tudo: jumps

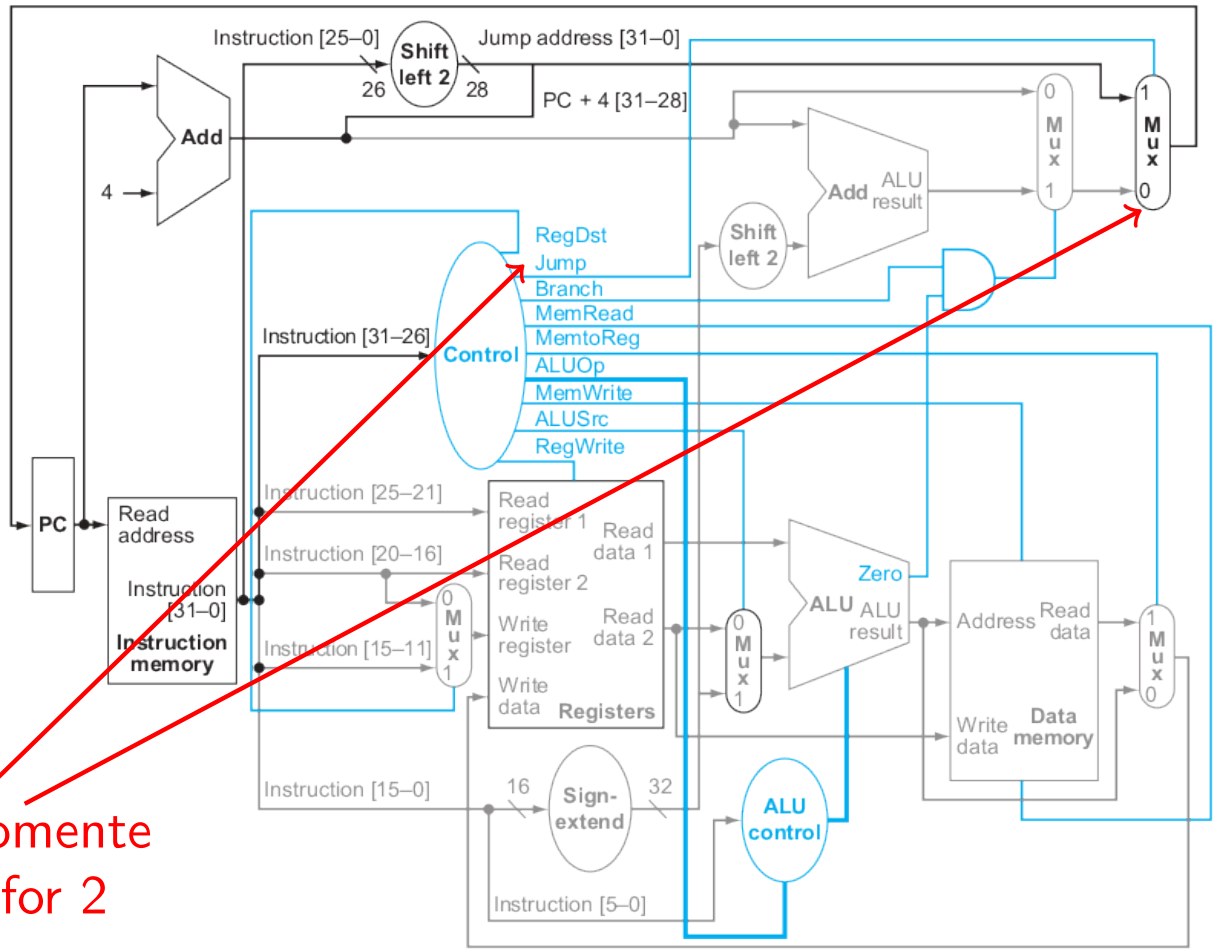


Determinando se o PC receberá o endereço do *jump* ou não

Fonte: [1]

Construindo o *Datapath*

Unindo tudo: jumps



Jump será 1 somente se o *opcode* for 2

Fonte: [1]

Datapath de Ciclo Único

Problemas

- Embora correto, esse tipo de projeto é ineficiente
  - O ciclo de *clock* precisa ter o mesmo comprimento, para toda instrução

Datapath de Ciclo Único

Problemas

- Embora correto, esse tipo de projeto é ineficiente
  - O ciclo de *clock* precisa ter o mesmo comprimento, para toda instrução
- Então o caminho mais longo possível no processador irá determinar seu tamanho
  - E este será certamente um `lw`, uma vez que ele usa a memória de instruções, o arquivo de registradores (para leitura e escrita final), a ALU e a memória de dados

Datapath de Ciclo Único

Solução

- *Pipelining*

Fonte: `https://asia.nikkei.com/Business/Business-deals/Australia-rejects-Hong-Kong-group-s-bid-for-pipeline-operator`

Datapath de Ciclo Único

Solução

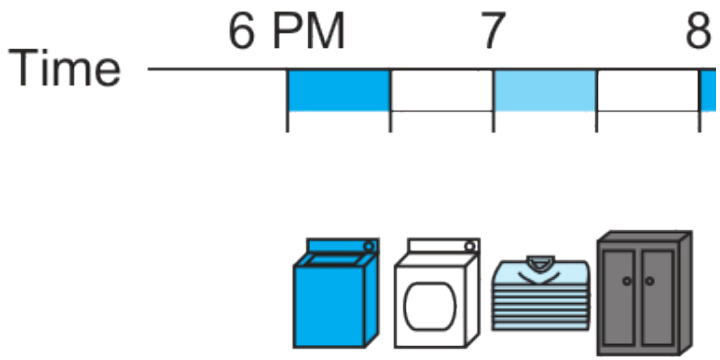
- *Pipelining*
  - Técnica de implementação na qual múltiplas instruções são sobrepostas durante a execução
  - Melhora assim a eficiência pela execução de múltiplas instruções simultaneamente

Fonte: `https://asia.nikkei.com/Business/Business-deals/Australia-rejects-Hong-Kong-group-s-bid-for-pipeline-operator`

Pipelining

Analogia – Lavar roupas

- Estágios da tarefa:
  - Coloca-se a roupa na lava-roupas
  - Quando pronta, passa-se à secadora
  - Quando pronta, dobra-se a roupa
  - Ao final guarda-se a roupa



Fonte: Adaptado de [1]

Pipelining

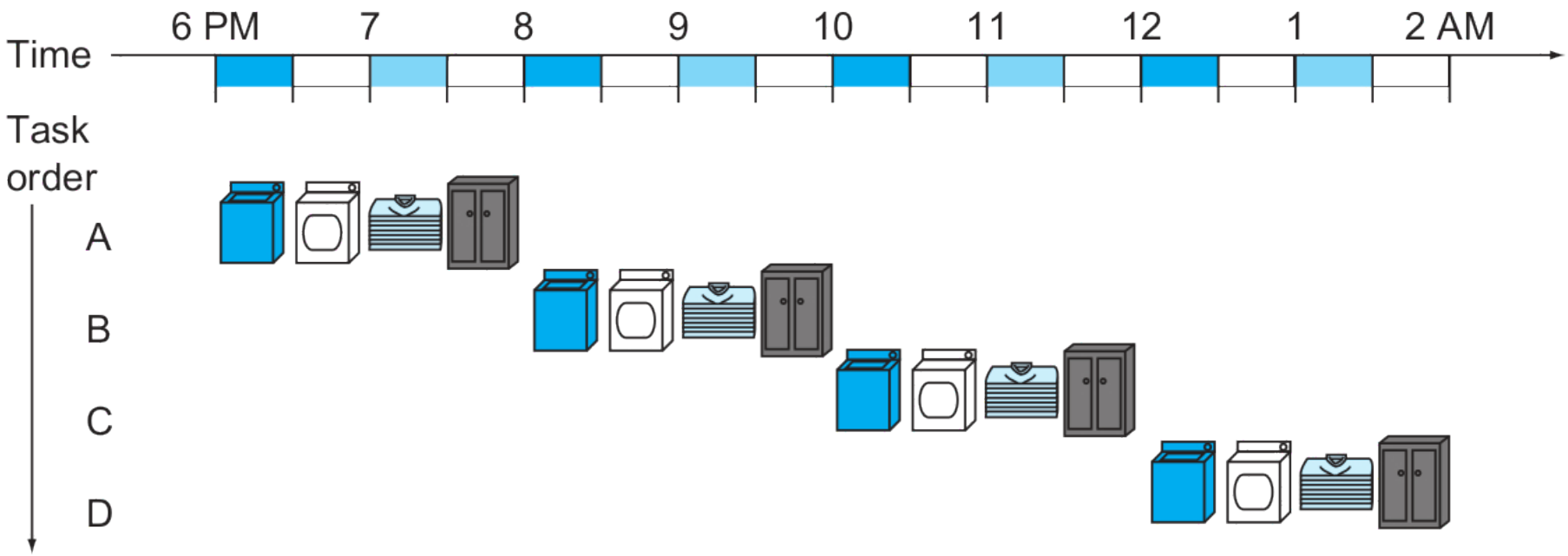
Analogia – Lavar roupas

- E o que acontece quando precisamos repetir esse procedimento?

Pipelining

Analogia – Lavar roupas

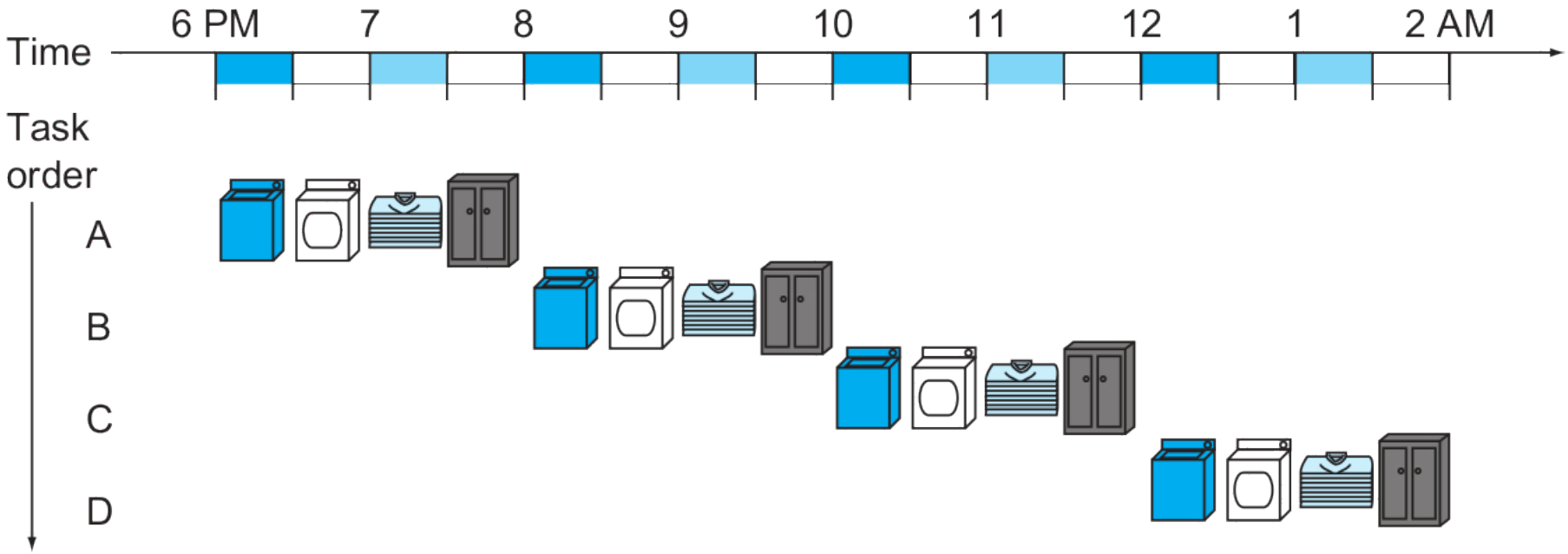
- E o que acontece quando precisamos repetir esse procedimento?



Fonte: [1]

Pipelining

Analogia – Lavar roupas

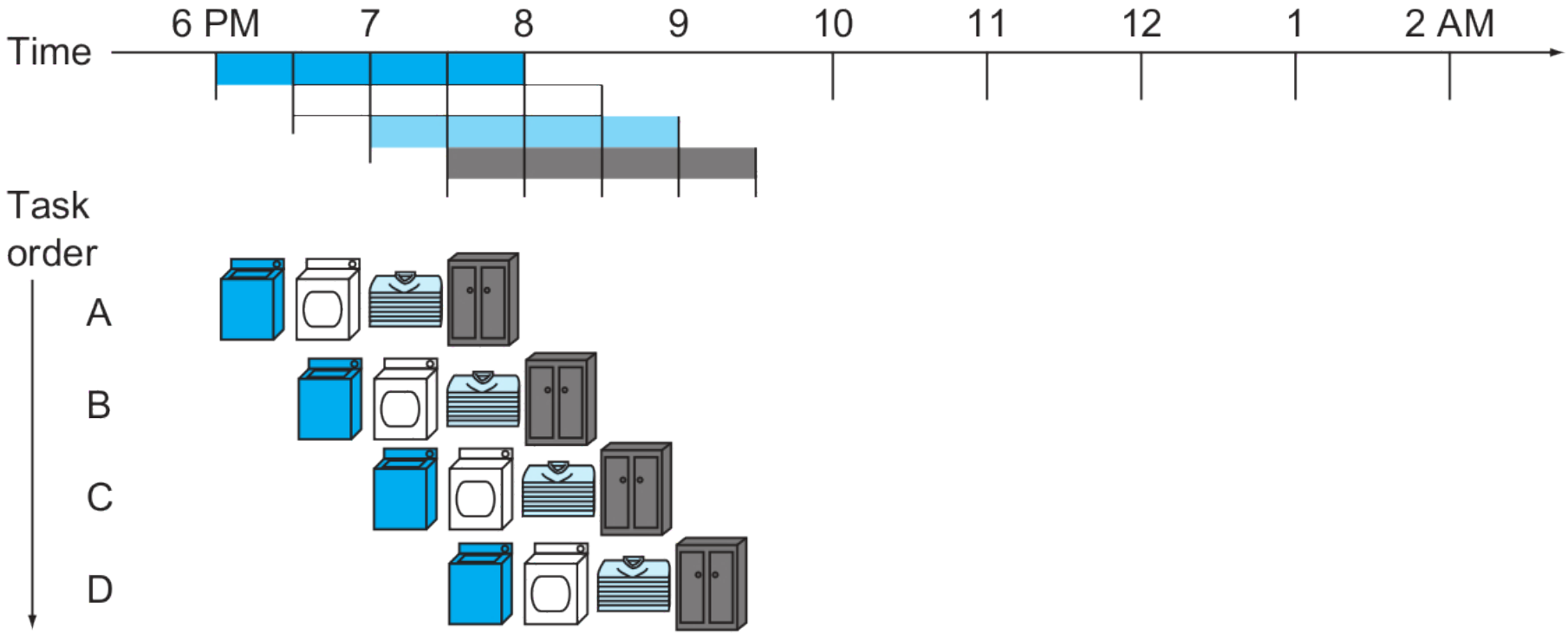


Fonte: [1]

- Não teria como deixar isso mais eficiente?

Pipelining

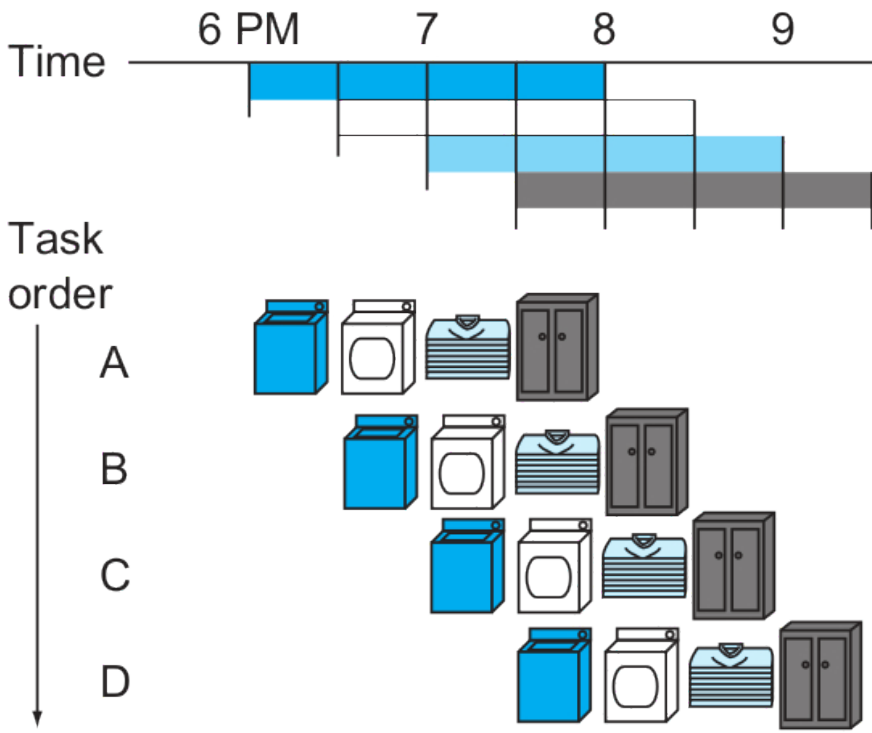
Analogia – Lavar roupas



Fonte: [1]

Pipelining

Analogia – Lavar roupas

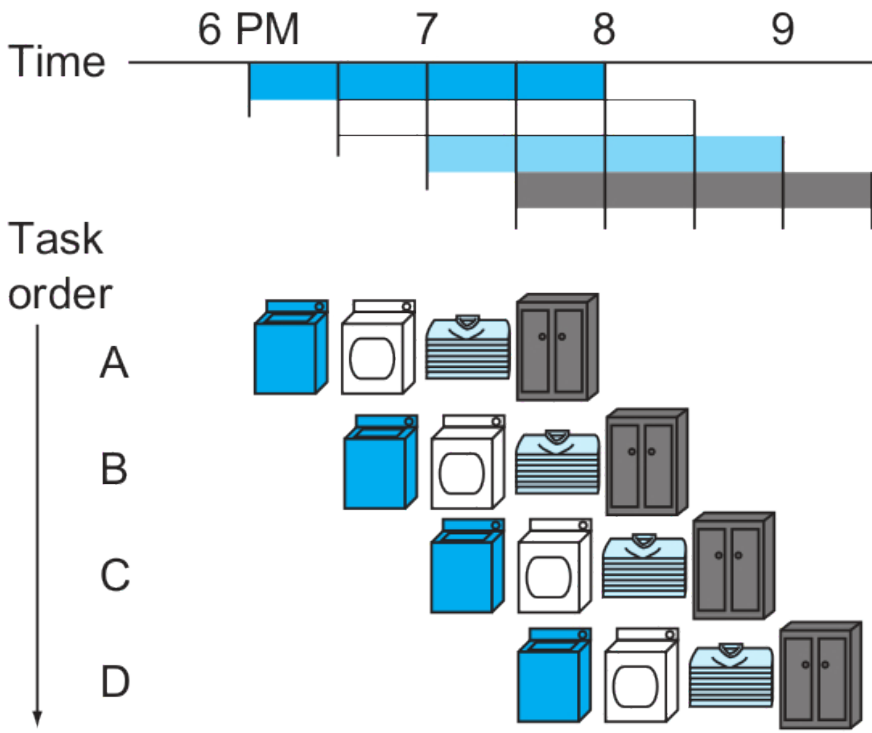


Aproveitamos os recursos liberados em uma tarefa para a seguinte

Fonte: [1]

Pipelining

Analogia – Lavar roupas

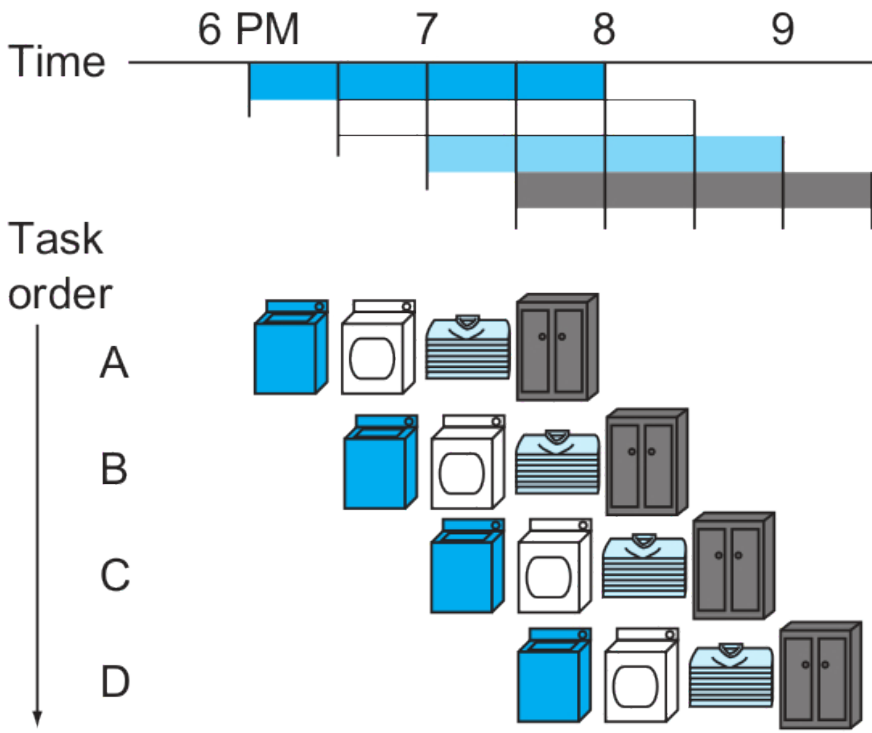


Note que isso não reduz o tempo individual de cada tarefa

Fonte: [1]

Pipelining

Analogia – Lavar roupas

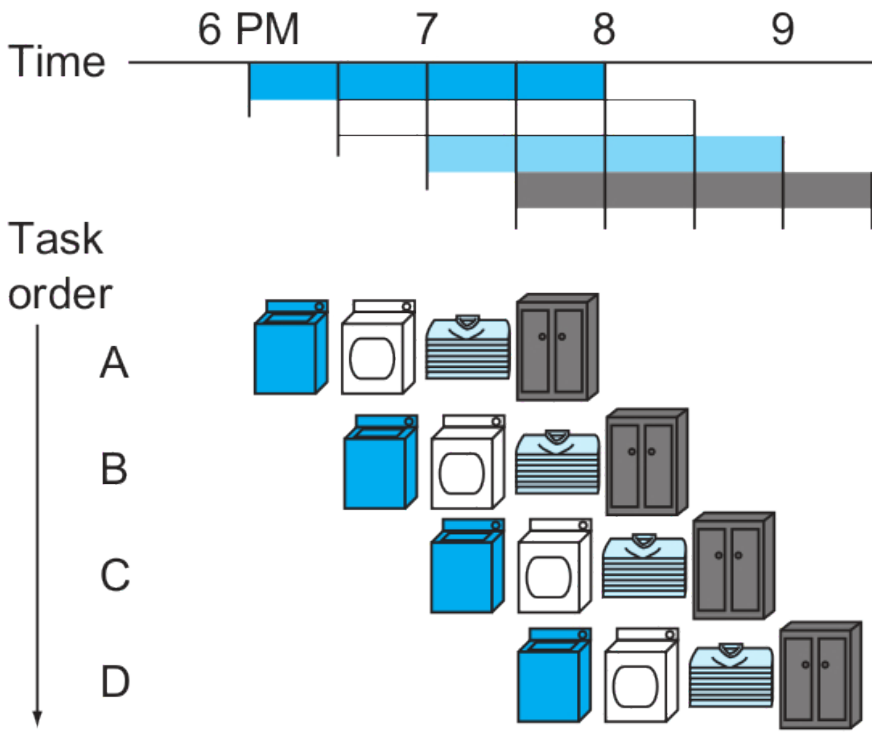


Cada uma ainda precisa completar seus 4 estágios (sua latência)

Fonte: [1]

Pipelining

Analogia – Lavar roupas

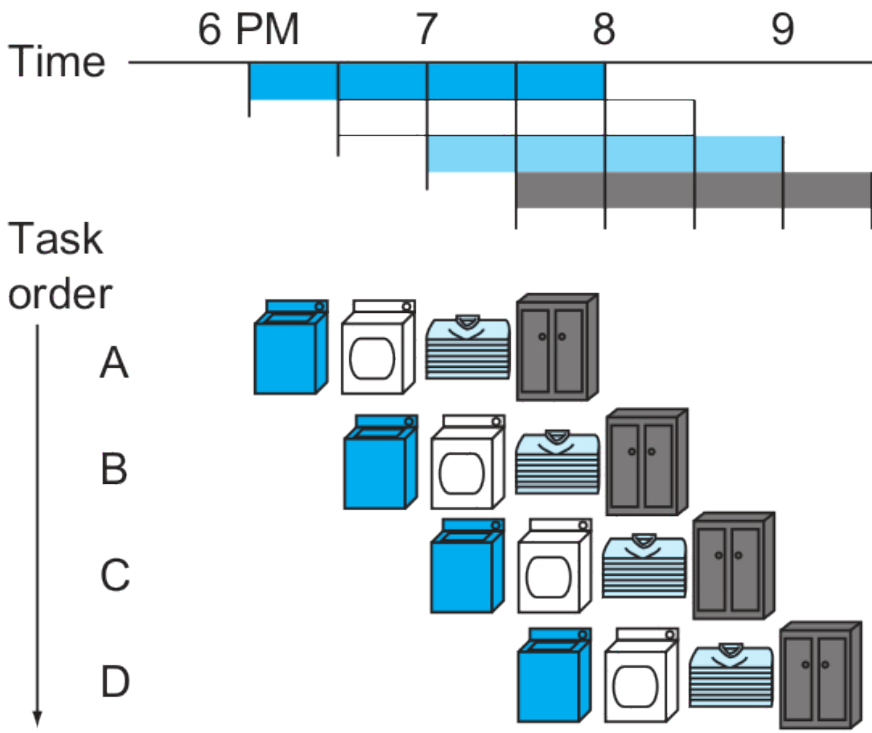


No entanto, pela sobreposição de tarefas, na média elas ficaram mais rápidas

Fonte: [1]

Pipelining

Analogia – Lavar roupas

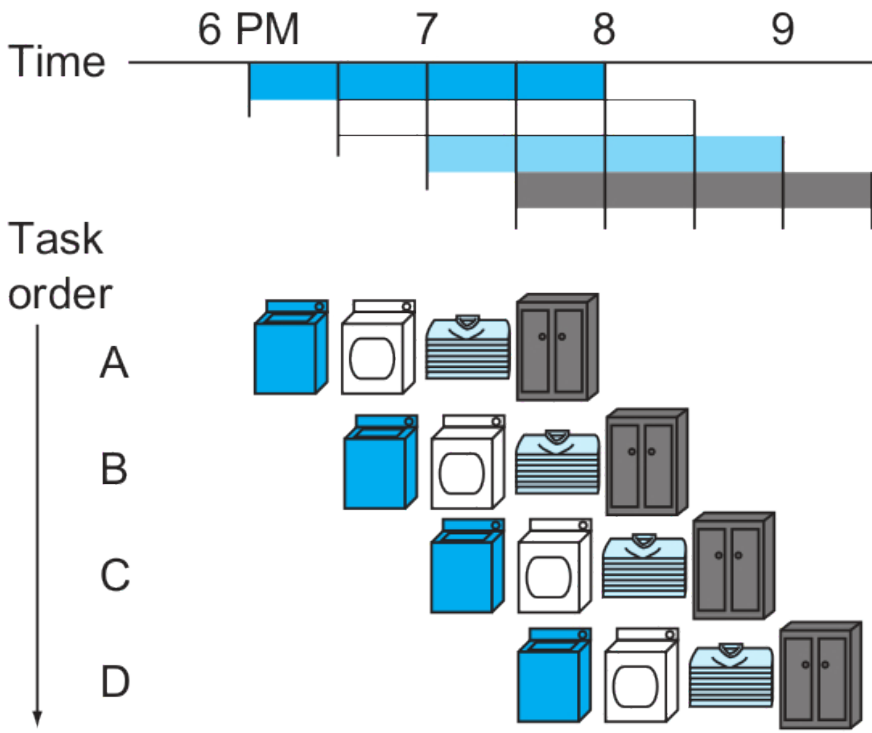


Pois foram paralelizadas pela sobreposição, aumentando assim a vazão

Fonte: [1]

Pipelining

Analogia – Lavar roupas



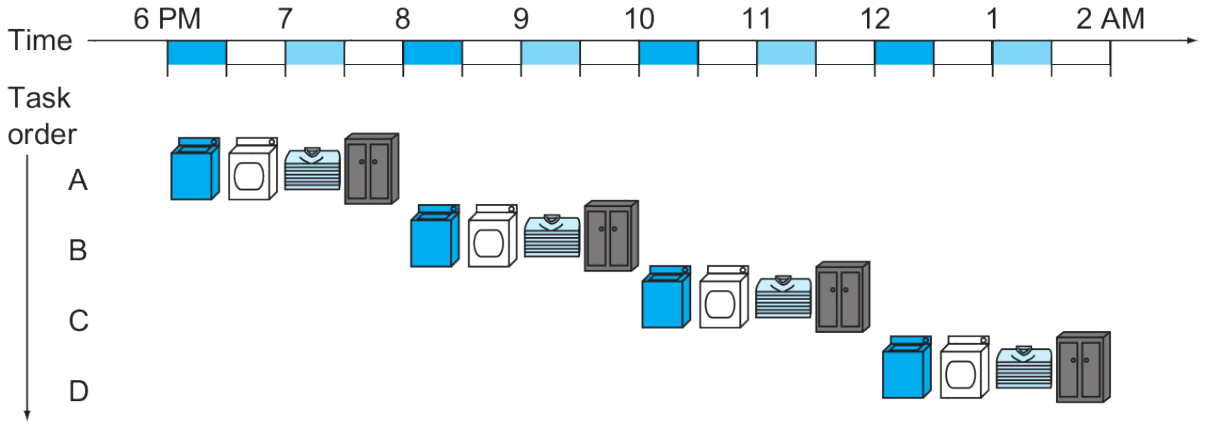
E reduzindo o tempo total para completar o trabalho

Fonte: [1]

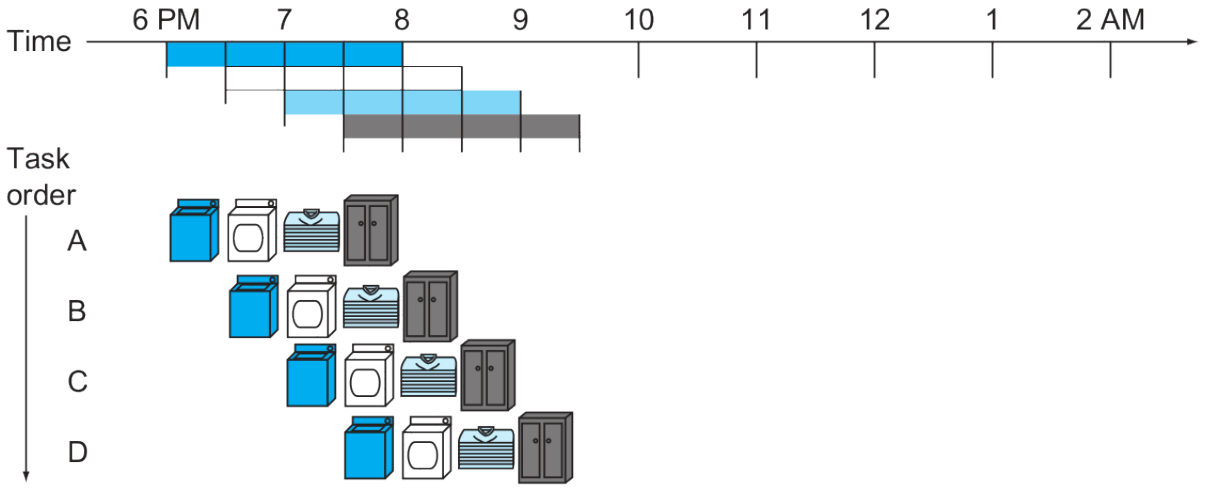
Pipelining

Analogia – Lavar roupas

Sem *pipelining*



Com *pipelining*

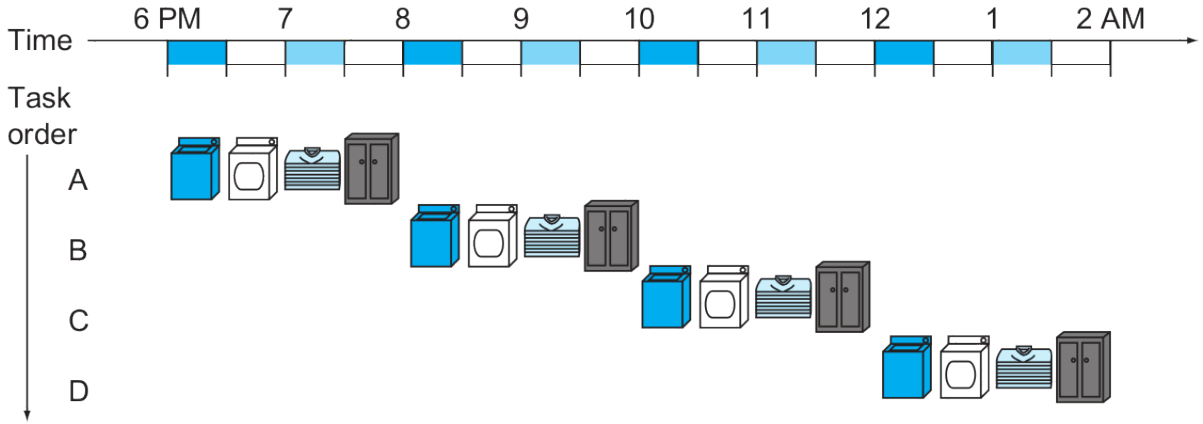


Fonte: [1]

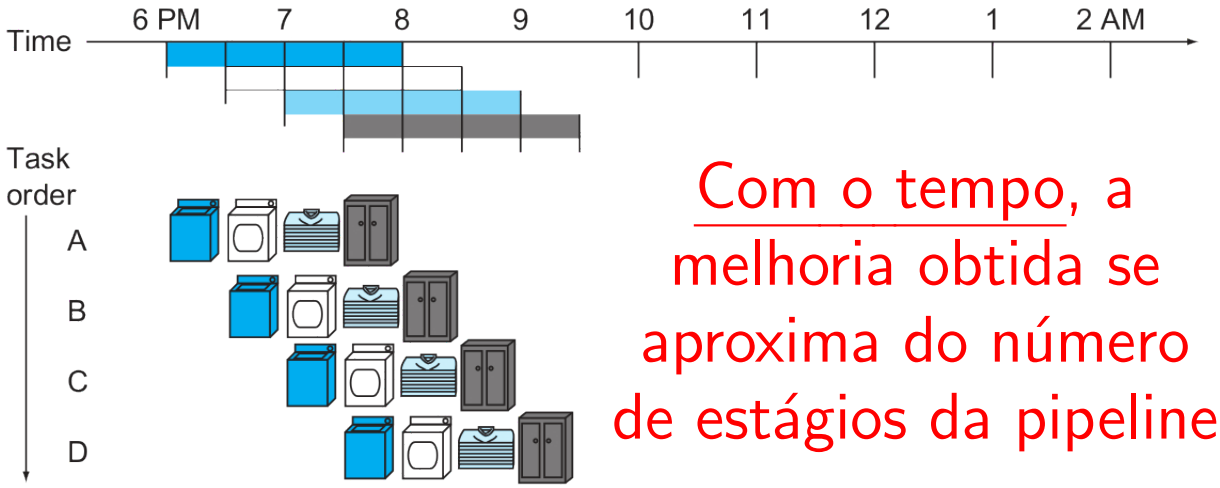
Pipelining

Analogia – Lavar roupas

Sem *pipelining*



Com *pipelining*



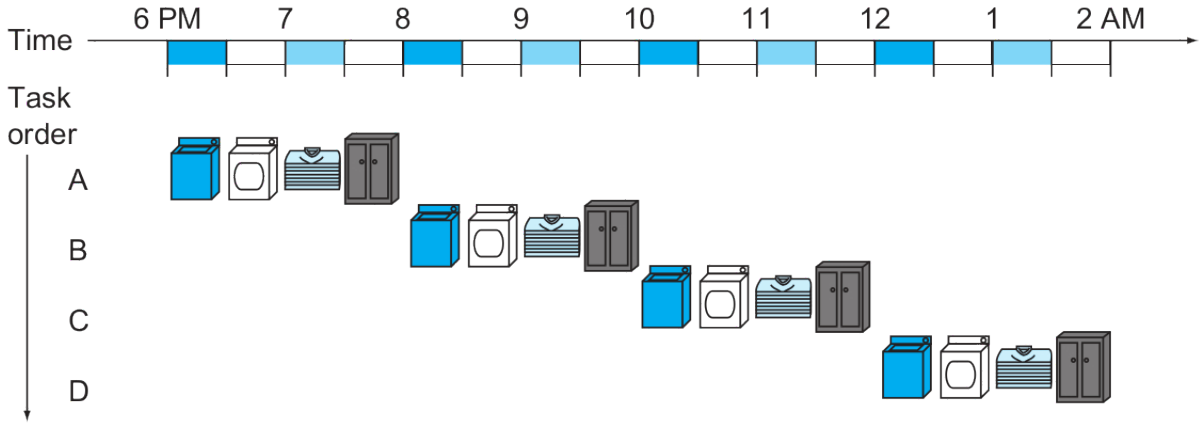
Com o tempo, a melhoria obtida se aproxima do número de estágios da pipeline

Fonte: [1]

Pipelining

Analogia – Lavar roupas

Sem *pipelining*



Com *pipelining*

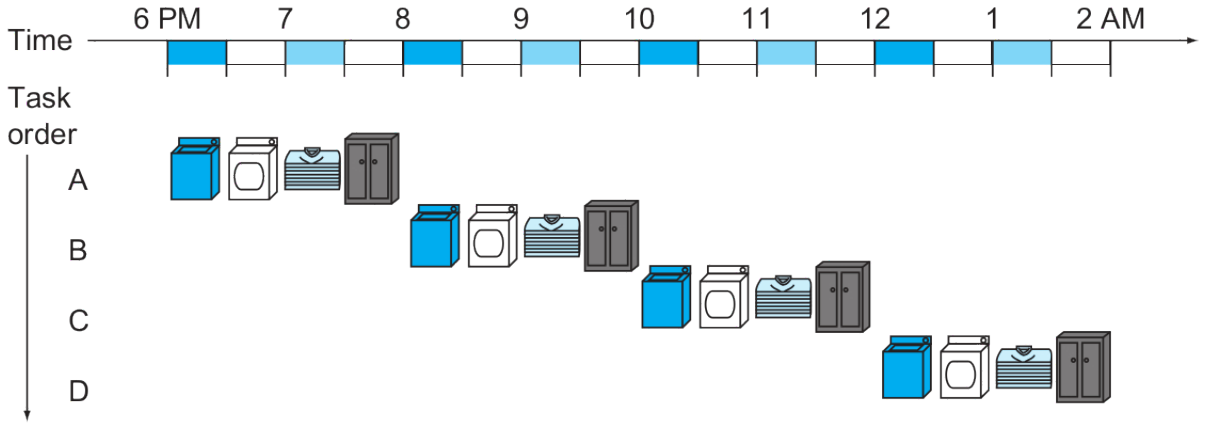
Isso se todos os estágios gastarem o mesmo tempo

Fonte: [1]

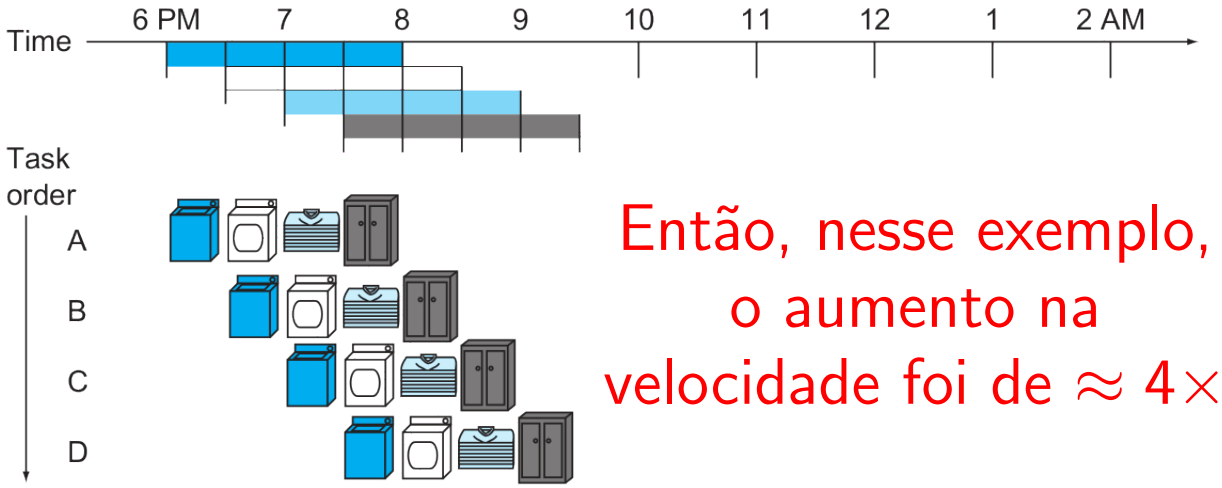
Pipelining

Analogia – Lavar roupas

Sem *pipelining*



Com *pipelining*



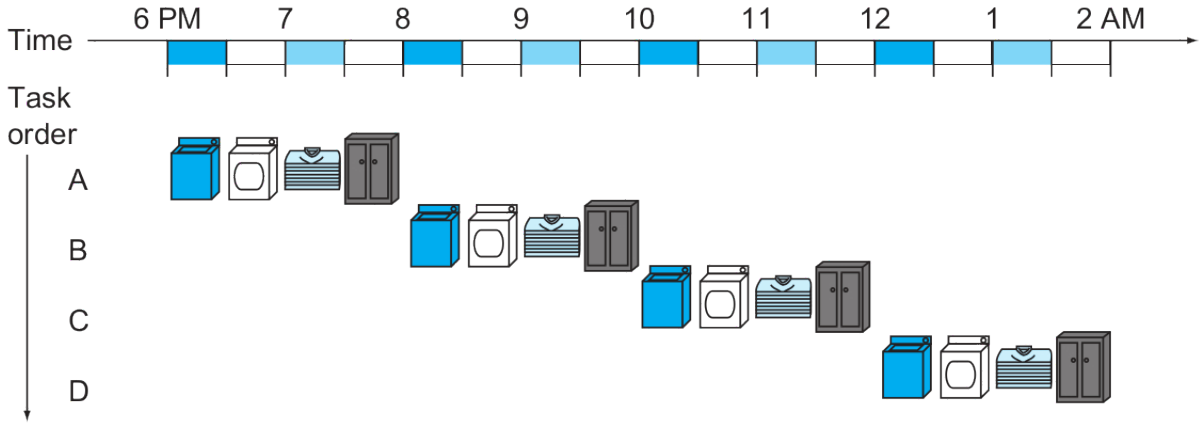
Então, nesse exemplo, o aumento na velocidade foi de $\approx 4\times$

Fonte: [1]

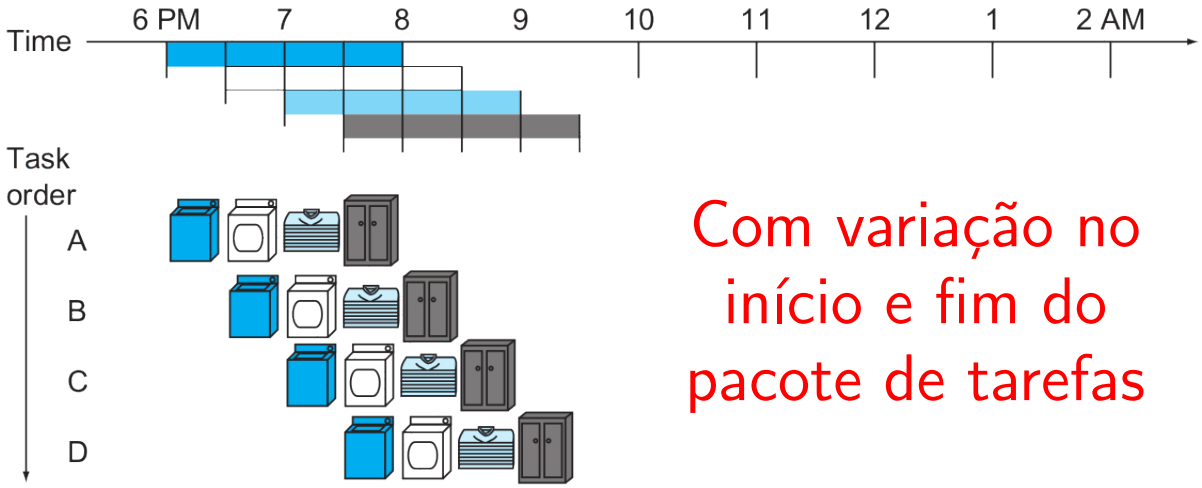
Pipelining

Analogia – Lavar roupas

Sem *pipelining*



Com *pipelining*



Com variação no início e fim do pacote de tarefas

Fonte: [1]

Pipelining

De volta a MIPS

- Será que haveria estágios também com as instruções MIPS?

Pipelining

De volta a MIPS

- Será que haveria estágios também com as instruções MIPS?
  1. Buscar a instrução da memória

Pipelining

De volta a MIPS

- Será que haveria estágios também com as instruções MIPS?
  1. Buscar a instrução da memória
  2. Ler registradores enquanto decodifica a instrução
     - O fato das instruções MIPS terem o mesmo tamanho torna fácil sua busca no primeiro estágio e sua decodificação no segundo
     - Por elas variarem pouco, podemos começar a ler os registradores ao mesmo tempo em que determinamos o tipo da instrução buscada
     - Sem essa simetria, a decodificação precisaria ser feita em um estágio à parte

De volta a MIPS

3. Executar a operação ou calcular um endereço
   - Operandos de memória somente aparecem em *loads* e *stores* em MIPS, tornando possível calcular o endereço de memória nesse estágio e acessá-la no seguinte
   - Operar nos operandos em memória (x86) exigiria 3 passos (endereço, memória e execução), em vez de 2 (execução e memória)

Pipelining

De volta a MIPS

3. Executar a operação ou calcular um endereço
   - Operandos de memória somente aparecem em *loads* e *stores* em MIPS, tornando possível calcular o endereço de memória nesse estágio e acessá-la no seguinte
   - Operar nos operandos em memória (x86) exigiria 3 passos (endereço, memória e execução), em vez de 2 (execução e memória)
4. Acessar operandos na memória
   - Uma vez que os operandos precisam estar alinhados na memória, não precisamos nos preocupar com uma instrução exigir 2 acessos à memória. Basta um estágio

De volta a MIPS

5. Escrever o resultado em um registrador

Pipelining

De volta a MIPS

5. Escrever o resultado em um registrador

- O conjunto de instruções MIPS foi projetado para execução em *pipeline*
  - Então com certeza podemos melhorar nosso *datapath* com *pipelining*

Referências

1. Patterson, D.A.; Hennessy, J.L. (2013): Computer Organization and Design: The Hardware/Software Interface. Morgan Kaufmann. 5ª ed.
   - Para detalhes sobre as partes do circuito consulte também o Apêndice B
2. `https://www3.ntu.edu.sg/home/smitha/FYP_Gerald/`
   - Simulador de uma arquitetura de ciclo único