Aula 4

Organização de Computadores

Caminho de dados do conjunto de instruções do MIPS

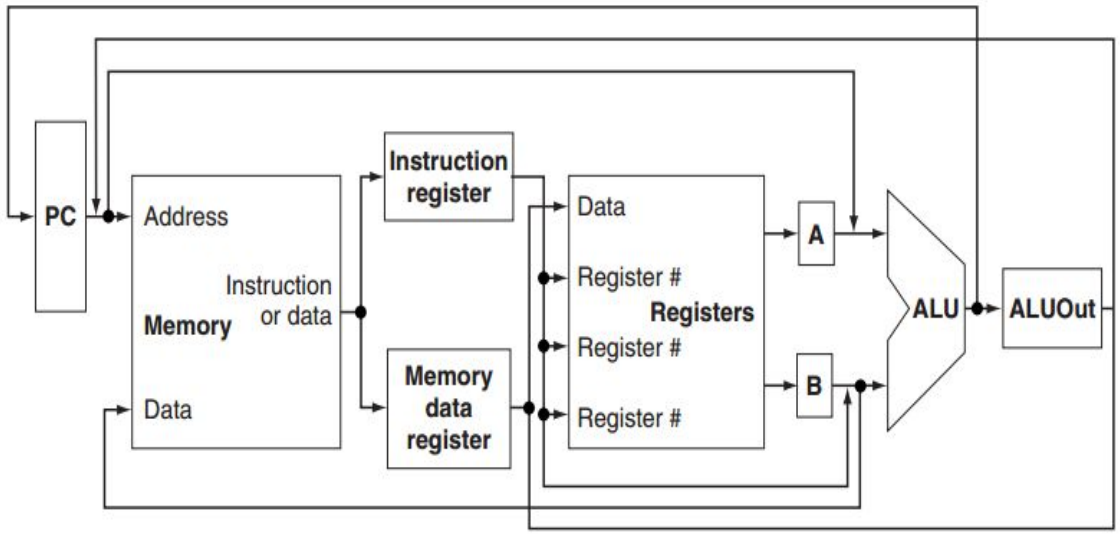
Profa. Débora Matos

Caminho de dados

Registradores acrescentados na implementação multiciclo:

- Registrador de Instrução – Instruction Register (RI)
- Registrador de dados da memória – Memory Data Register (MDR).

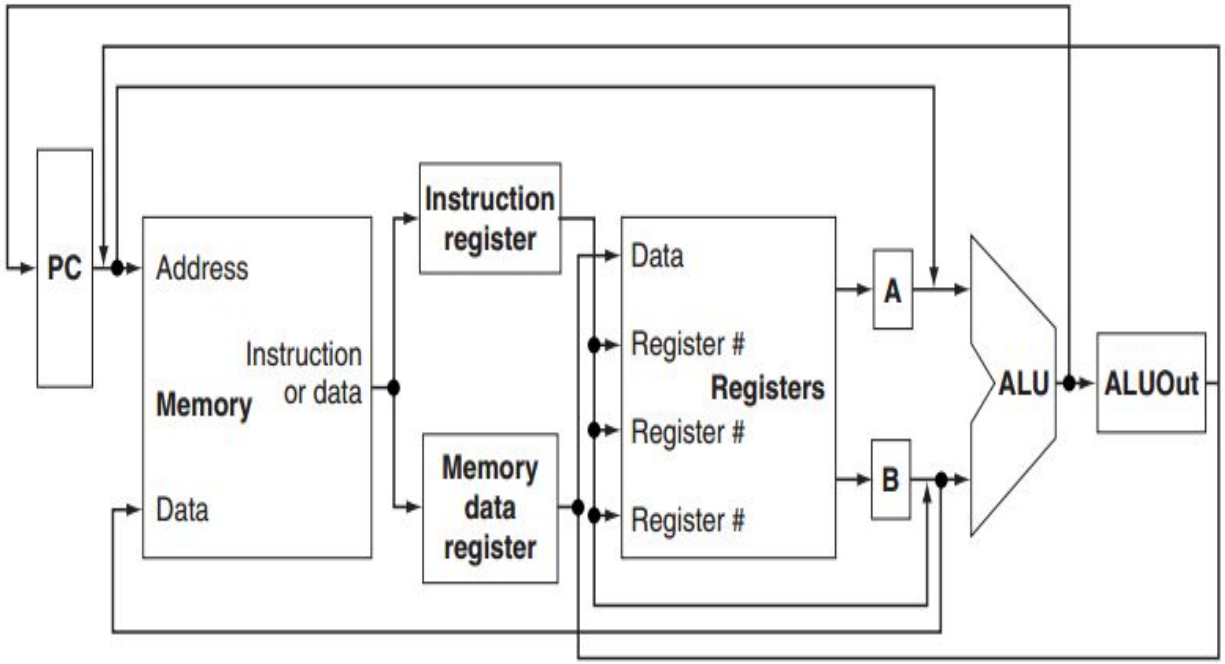
RI e MDR são usados para salvar a saída da memória referentes a uma leitura de instrução e uma leitura de dados, respectivamente.



Caminho de dados

Registradores acrescentados na implementação multiciclo:

- **Registradores A e B** usados para conter os valores dos registradores operandos lidos do banco de registradores.
- **Registrador ALUOut** – registrador de saída da ALU



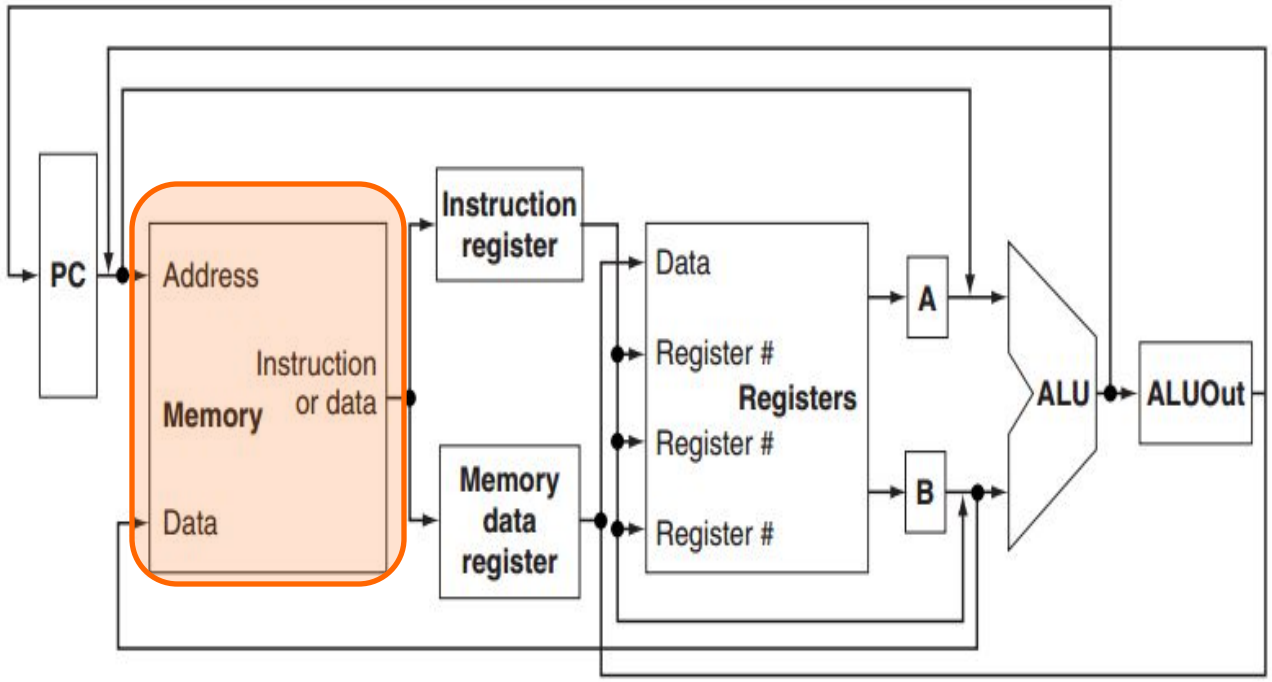
Caminho de dados

Exemplo do caminho de dados da seguinte instrução:

add $s3, $s2, $s1

1°) Busca da instrução na memória de instruções:

PC contém o endereço da próxima instrução. A partir desse endereço a instrução é lida e salva no RI.

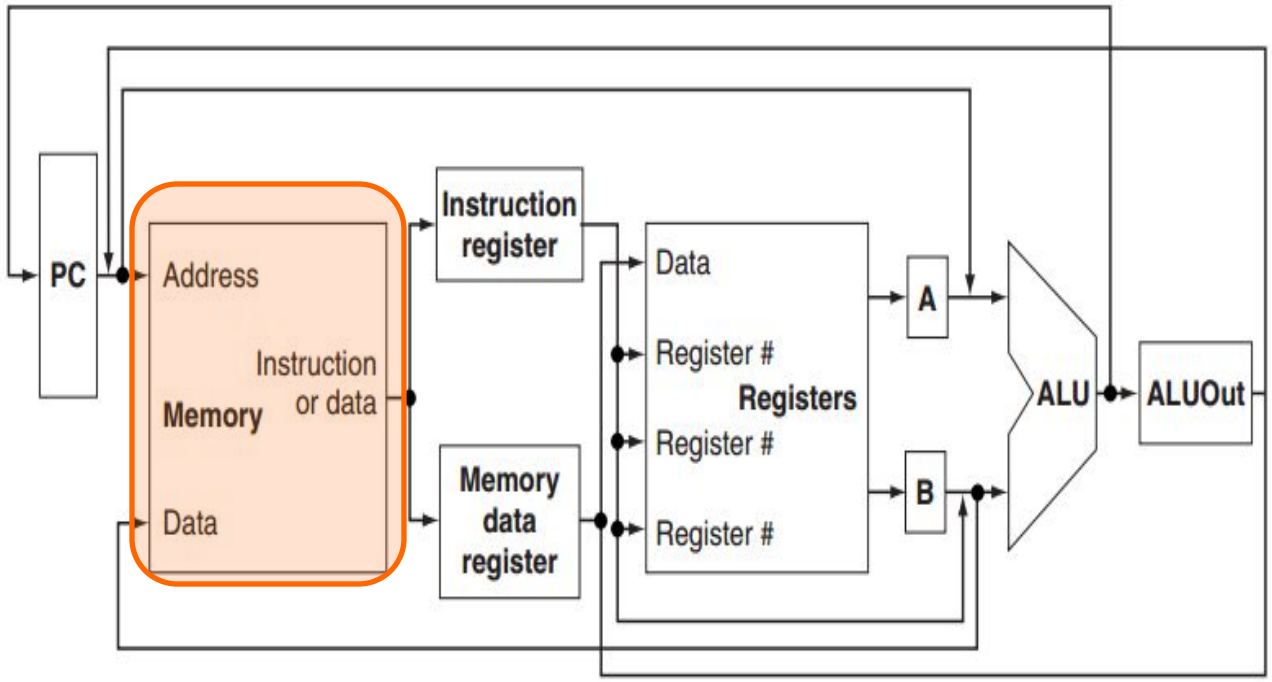


Caminho de dados

Exemplo do caminho de dados da seguinte instrução:

add $s3, $s2, $s1

2°) Próximo passo, a instrução é decodificada. Nesse caso é verificada a operação a ser realizada de forma a ativar a operação e os módulos requeridos em cada ciclo pela unidade de controle.

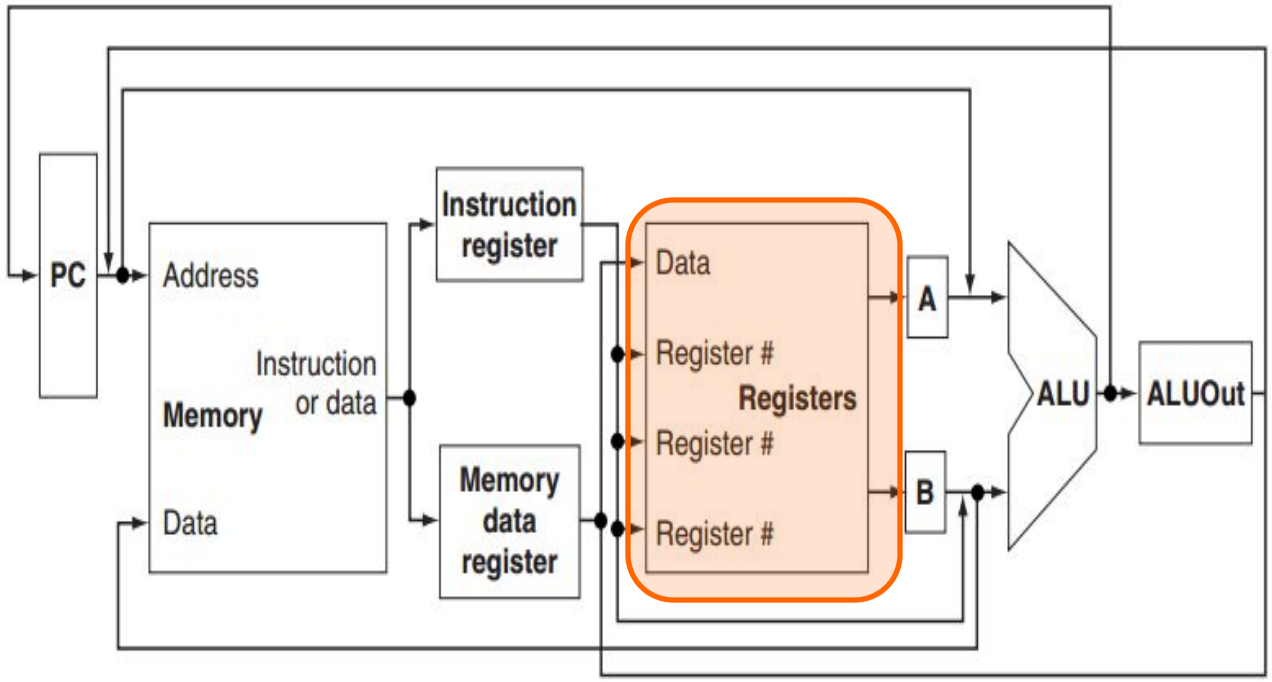


Caminho de dados

Exemplo do caminho de dados da seguinte instrução:

add $s3, $s2, $s1

3°) Como é uma instrução do tipo R, os operandos estão no banco de registradores. Os campos referentes aos registradores usados são informados e os dados a serem somados são lidos e escritos nos registradores A e B.

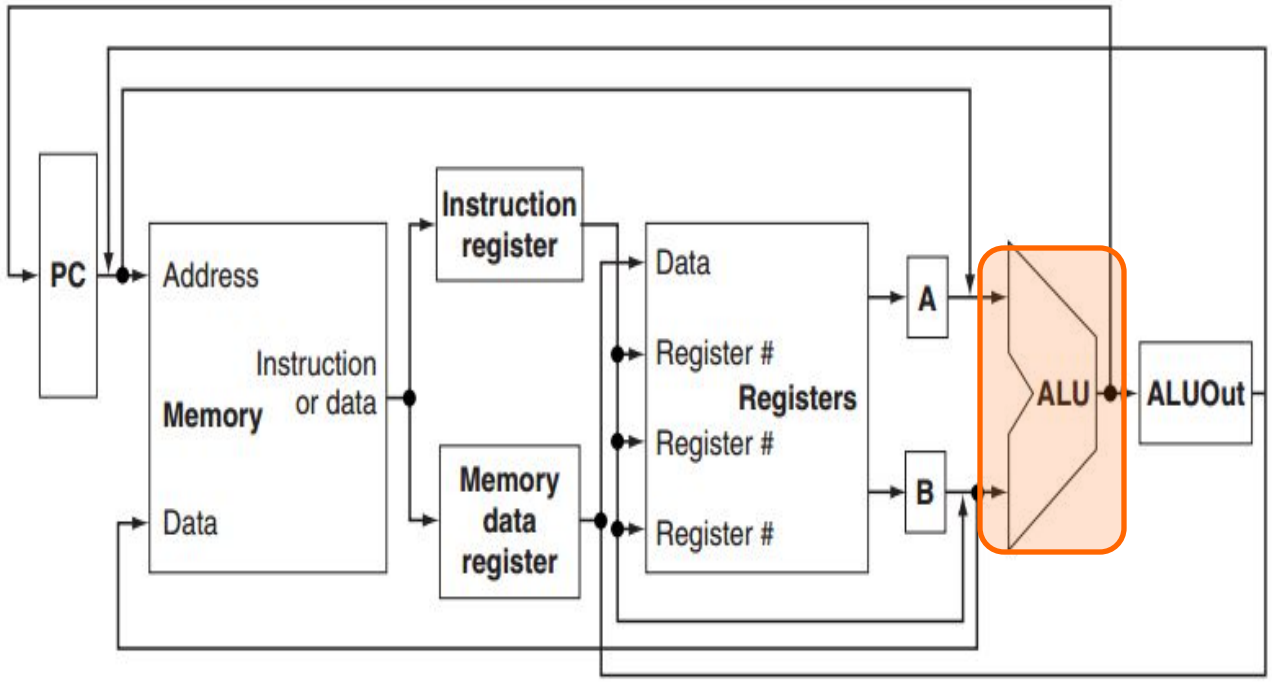


Caminho de dados

Exemplo do caminho de dados da seguinte instrução:

add $s3, $s2, $s1

4°) A soma entre os registradores A e B (referentes aos registradores $s1 e $s2) é realizada na ALU e o resultado é salvo no registrador de saída da ALU (ALUOut).

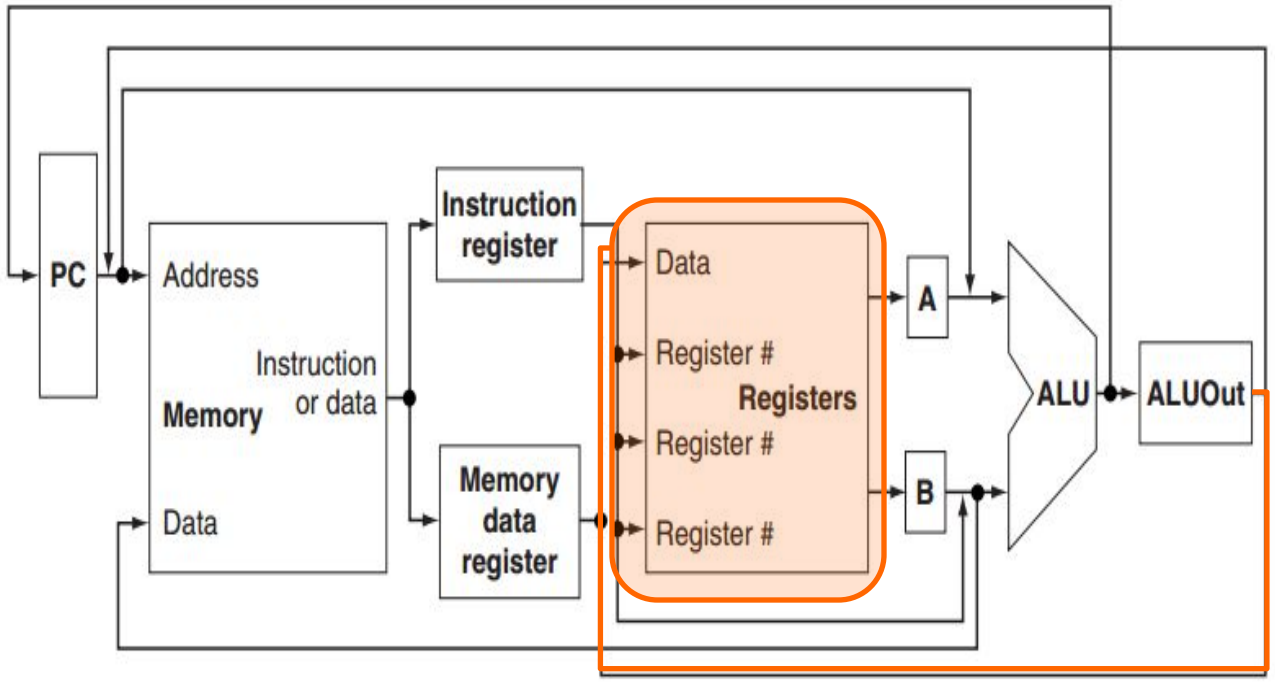


Caminho de dados

Exemplo do caminho de dados da seguinte instrução:

add $s3, $s2, $s1

5°) O resultado da operação precisa ser escrito no registrador de escrita o banco de registradores, nesse caso, o $s3. Para isso, o registrador de escrita é identificado na instrução (no IR) e o resultado da soma escrito.



Caminho de dados das instruções

Analisaremos o caminho de dados das seguintes instruções:

- Instruções de referência a memória: lw e sw
- Instruções com a ULA: add, sub, and, or e slt;
- Instruções beq e j (jump)

As instruções do MIPS apresentam regularidade. A maioria das instruções implementam etapas semelhantes. Para todas as classes de instruções os próximos 2 passos são executados:

- Ler o valor do endereço que consta no PC e buscar a próxima instrução da memória.
- Ler os registradores do banco que serão utilizados na instrução.

Caminho de dados das instruções

A instrução load precisa ler apenas 1 registrador, mas a maioria das instruções leem 2 registradores.

lw $s0, 0($s2)

Após as 2 etapas idênticas para todas as instruções, as demais etapas dependem da classe de cada instrução:

- Instruções de referência a memória (lw, sw);
- Instruções lógicas e aritméticas;
- Instruções de desvios.

Caminho de dados das instruções

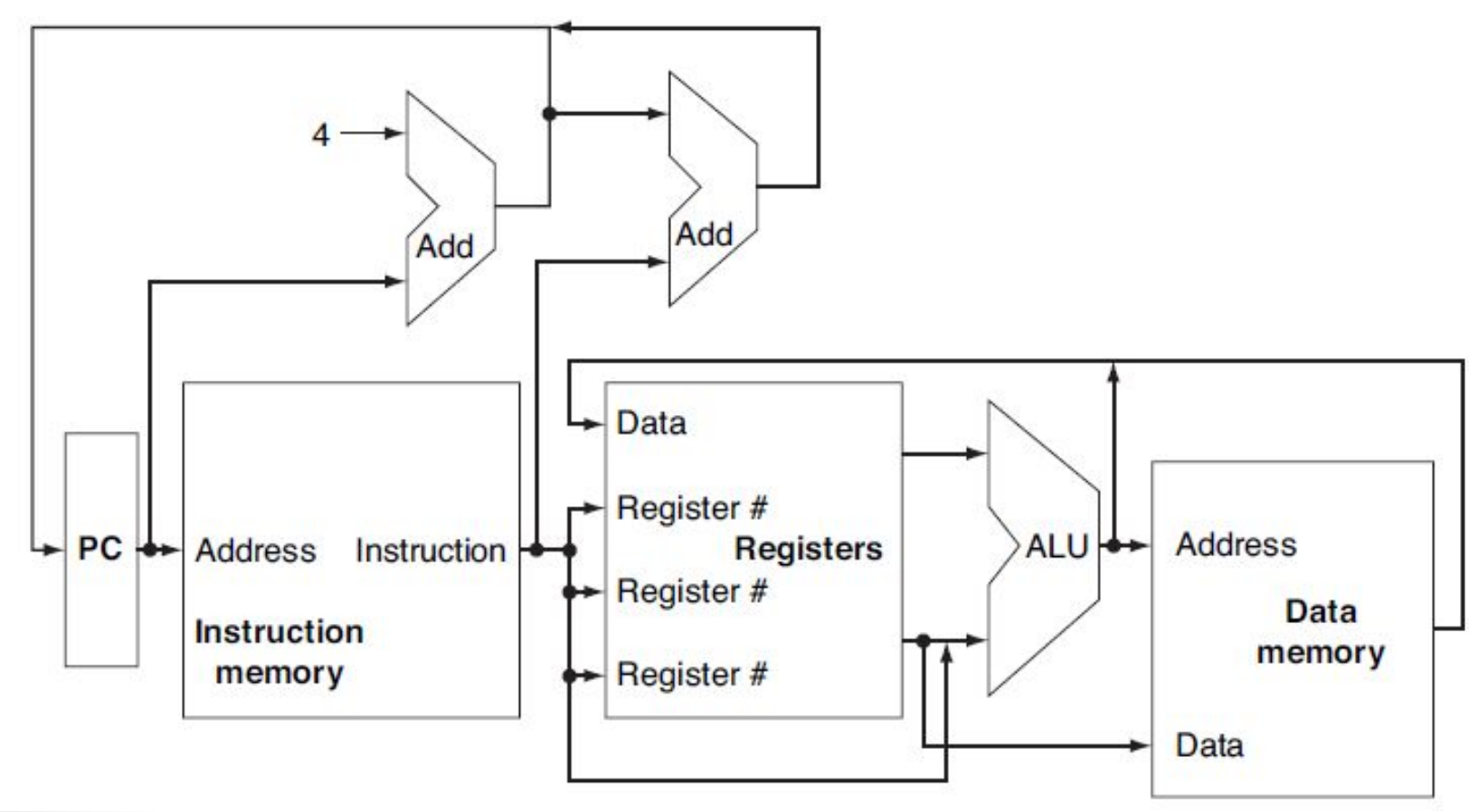
•Na terceira etapa, com exceção da instrução de jump, todas as demais utilizam a ULA.

Para as seguintes instruções, que operação é realizada na ULA: beq, slt, lw, sw?

•Após a terceira etapa, as ações para completar a execução da instrução dependem da sua classe:

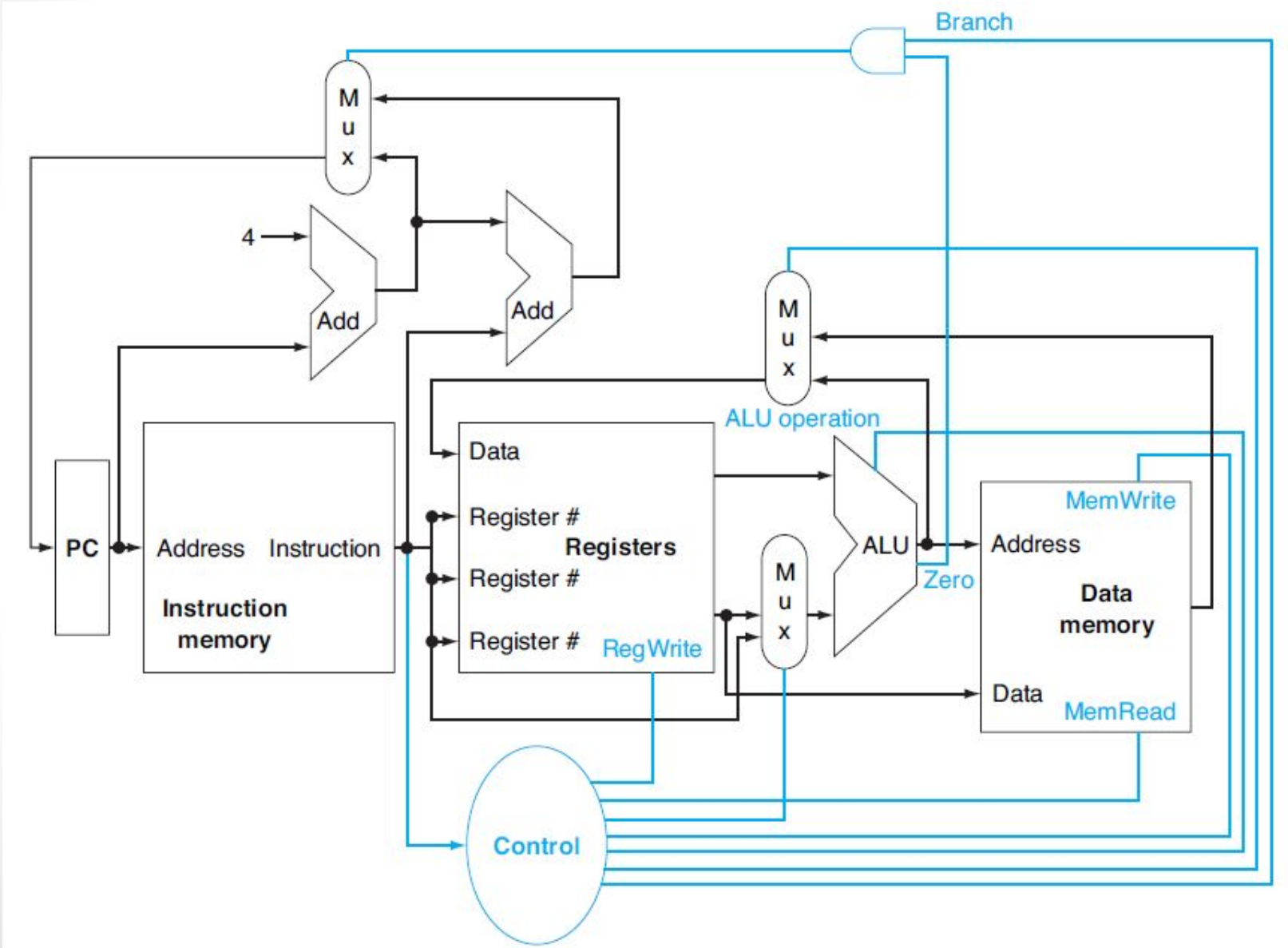
- Instruções de referência à memória irão acessá-la.
- Instruções do tipo R, precisam escrever o resultado da ALU no registrador de destino.
- A instrução de desvio, ou mudará o valor do PC com base na comparação ou irá incrementá-lo em 4 posições.

Caminho de dados das instruções



O que está faltando no projeto para a correta execução do grupo de instruções definido?

Caminho de dados das instruções



Caminho de dados das instruções

- Analisando a instrução beq:

  beq, $t1, $t2, offset

- O endereço de desvio precisa ser calculado. No entanto o campo offset possui apenas 16 bits.

- O endereço de destino é calculado somando o campo de offset estendido com o PC.

Caminho de dados das instruções

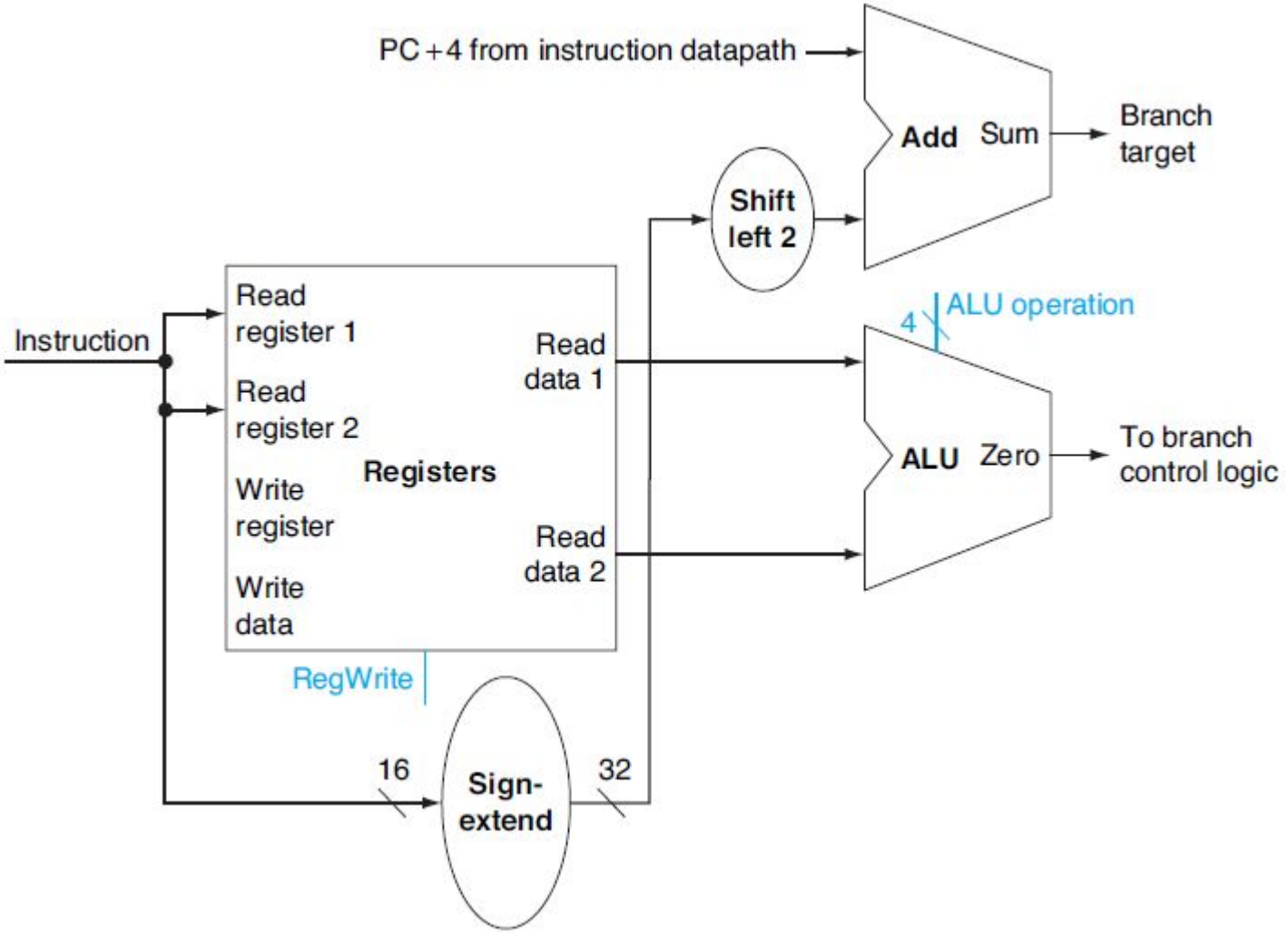
- Analisando a instrução beq:

  beq, $t1, $t2, offset

- Observações:
  - A base para o cálculo do endereço de desvio é o endereço da instrução seguinte ao desvio (PC + 4);
  - O campo offset é deslocado em 2 bits para a esquerda para compor um offset de 1 palavra.

- Sendo assim, o caminho de dados de desvio precisa de 2 operações: calcular o endereço de destino e comparar o conteúdo do registrador.

Caminho de dados das instruções



Caminho de dados das instruções

- No conjunto de instruções do MIPS os desvios são atrasados, ou seja a instrução imediatamente posterior ao desvio é sempre executada.

- Essa solução facilita a implementação das técnicas de pipeline.

Faça o esquemático para a instrução de *branch* condicional e explique seu funcionamento.

Caminho de dados das instruções

Caminho de dados *single cycle* (ciclo único):

- Quando as instruções são projetadas para operar em apenas 1 ciclo de clock, nenhum recurso do caminho de dados pode ser reutilizado em cada instrução.
- Sendo assim, cada elemento que precisa ser acessado mais de uma vez em cada instrução precisa ser replicado.

Quais elementos precisam ser replicados?

- É preciso ter uma memória de dados separada da memória de instruções. Adicionar somadores, etc.

Caminho de dados das instruções

- **Instruções do tipo R x instruções de acesso à memória:**
  - As instruções do tipo R enviam para a ALU dados vindos de 2 registradores.
  - As instruções de acesso à memória usam a ALU para o cálculo do endereço, mas uma das entradas é o campo offset de 16 bits com sinal estendido da instrução.
  - O valor armazenado no registrador de destino vem da ALU (para instruções do tipo R) ou da memória (para instrução load).

Caminho de dados das instruções

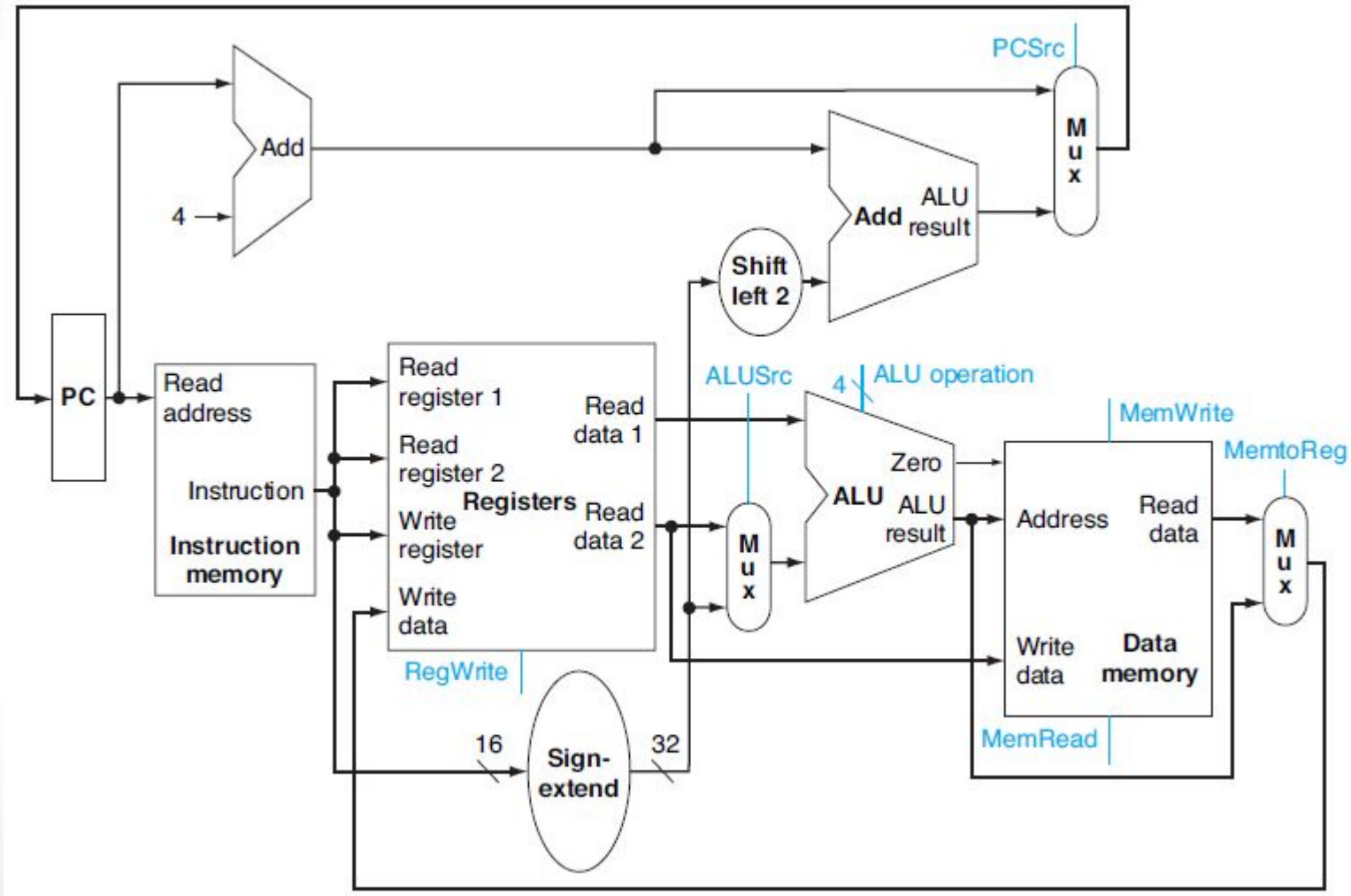
Porque na implementação de ciclo único as memórias de dados e instruções precisam ser separadas?

1)Dentro de um mesmo ciclo não é possível ler dois dados da memória.

2)Ter memórias separadas é menos dispendioso.

3)Permite tamanho da palavra diferente que o tamanho da instrução.

Caminho de dados das instruções



Caminho de dados das instruções

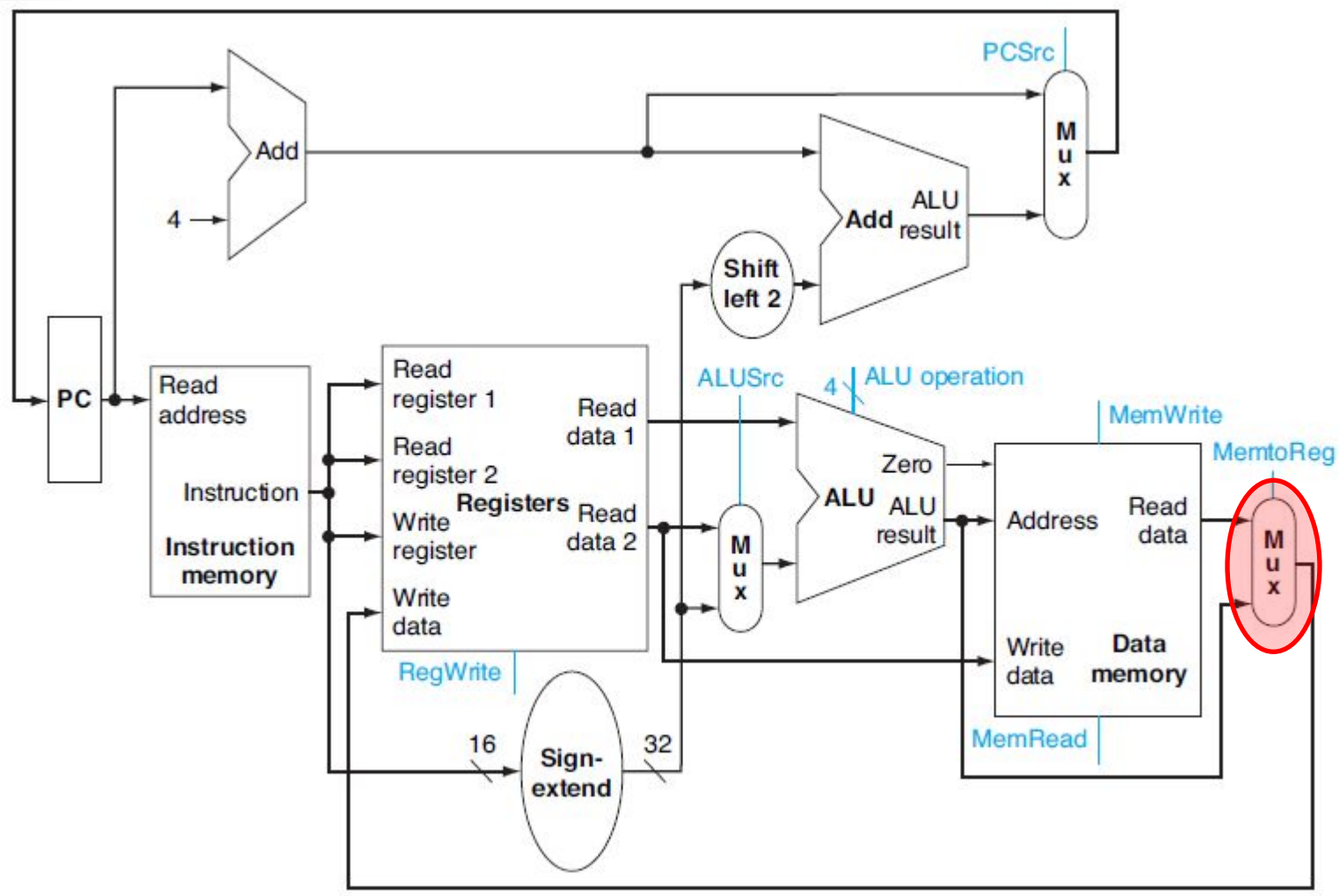
Exercício:

a) Considerando as observações mencionadas, construa um caminho de dados para instruções do tipo R e de acesso à memória considerando que as instruções levam 1 ciclo de clock.

b) O que precisaria mudar nessa organização para implementar a seguinte instrução (soma e salva o resultado em uma posição de memória)? Que HW a mais precisa ser adicionado:

addm $s1, $s2, offset

Caminho de dados das instruções



Em que situação este MUX é utilizado?

Para quais instruções?

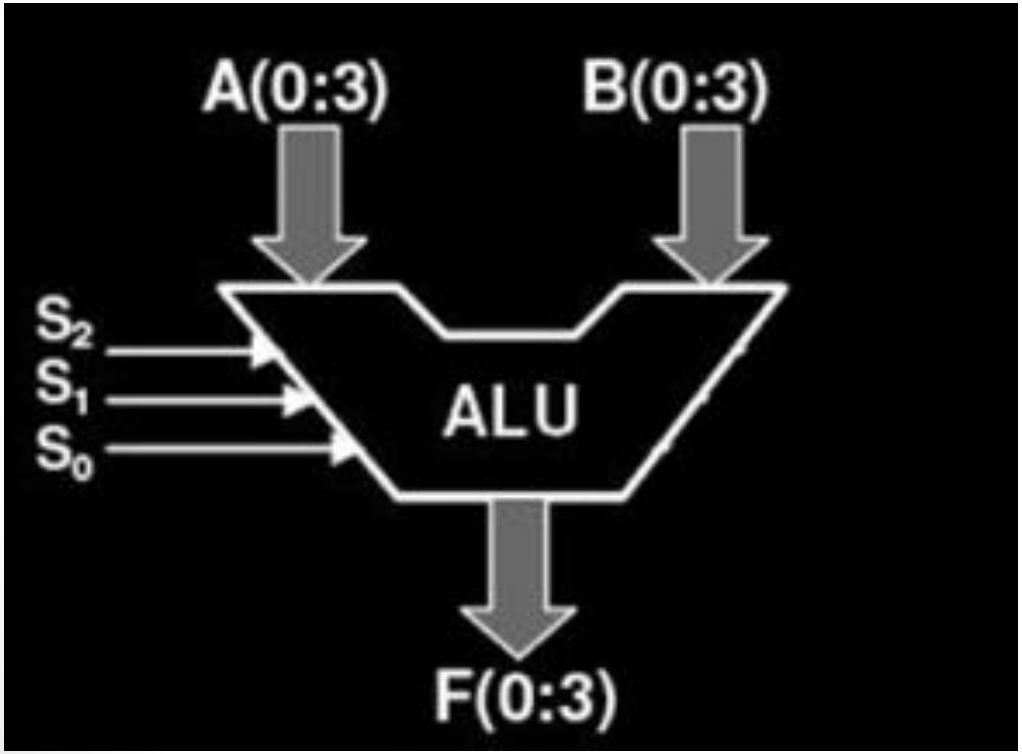
Caminho de dados das instruções

Funções da ALU:

ALU control lines	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	set on less than
1100	NOR

Caminho de dados das instruções

Implementação da ALU em VHDL:



S_2	S_1	S_0	Function (F)
0	0	0	A+B
0	0	1	A-B
0	1	0	A-1
0	1	1	A+1
1	0	0	A $\wedge$ B
1	0	1	A $\vee$ B
1	1	0	NOT A
1	1	1	A $\oplus$ B

Caminho de dados das instruções

Implementação da ALU em VHDL:

VHDL Code for 4-bit ALU

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3   use IEEE.NUMERIC_STD.ALL;
4
5   entity alu is
6   port(   Clk : in std_logic; --clock signal
7           A,B : in signed(3 downto 0); --input (
8           S : in unsigned(2 downto 0); --Operat
9           F : out signed(3 downto 0)  --output (
10          );
11  end alu;
12
13  architecture Behavioral of alu is
14
15  signal t1,t2,t3: signed(3 downto 0) := (other
16
17  begin
18
19  t1<= A;
20  t2<= B;
21  F<= t3;
22
```

Caminho de dados das instruções

Implementação da ALU em VHDL:

```
23  process(Clk)
24  begin
25
26      if(rising_edge(Clk)) then --Do the calcul
27          case S is
28              when "000" =>
29                  t3<= t1 + t2;  --addition
30              when "001" =>
31                  t3<= t1 - t2; --subtraction
32              when "010" =>
33                  t3<= t1 - 1;  --sub 1
34              when "011" =>
35                  t3<= t1 + 1; --add 1
36              when "100" =>
37                  t3<= t1 and t2; --AND gate
38              when "101" =>
39                  t3<= t1 or t2;  --OR gate
40              when "110" =>
41                  t3<= not t1 ;  --NOT gate
42              when "111" =>
43                  t3<= t1 xor t2; --XOR gate
44              when others =>
45                  NULL;
46          end case;
47      end if;
48
49  end process;
50
51  end Behavioral;
```

Caminho de dados das instruções

Funções 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

Caminho de dados das instruções

Analisando o formato das instruções:

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

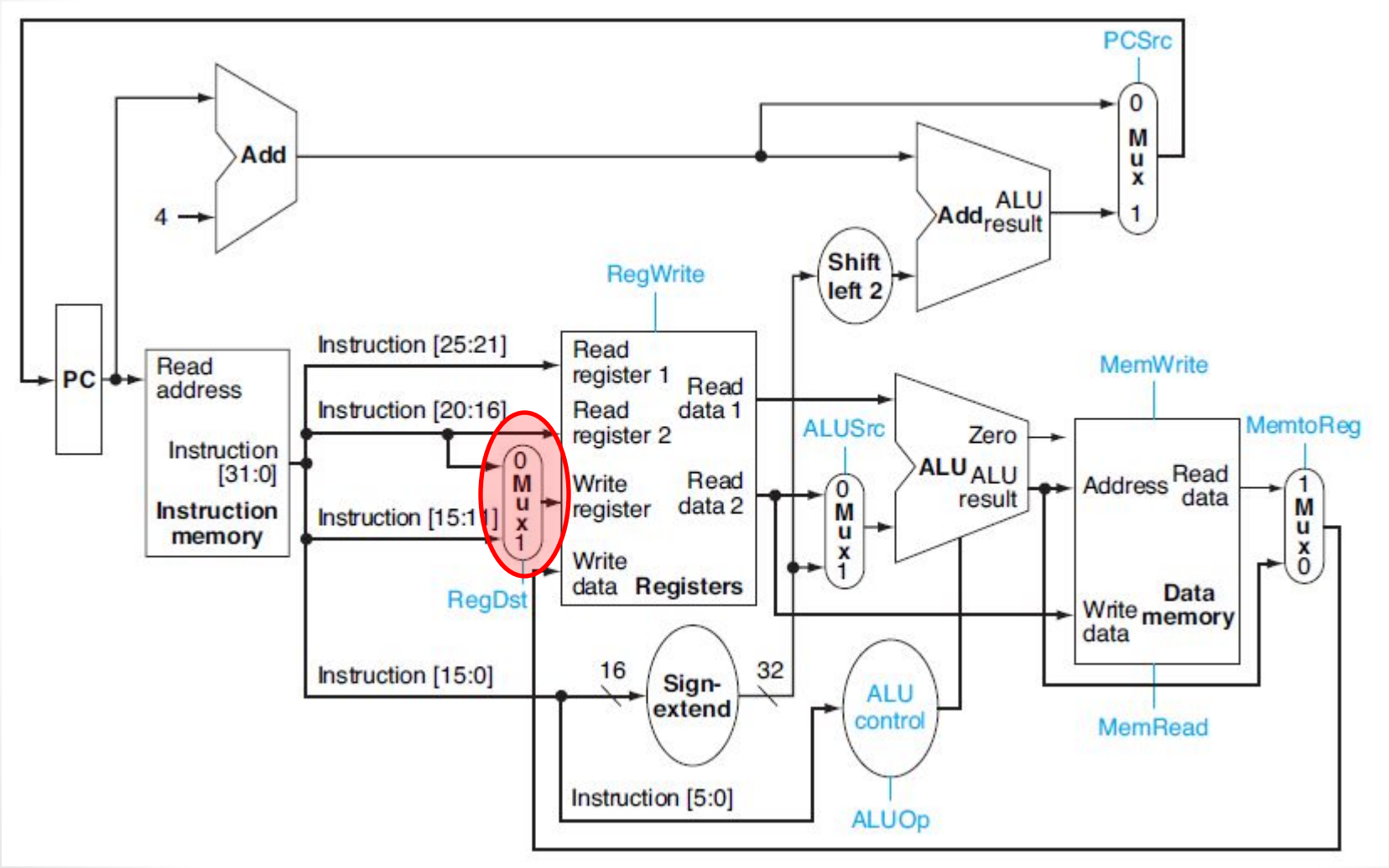
Field	4	rs	rt	address
Bit positions	31:26	25:21	20:16	15:0

c. Branch instruction

- Registrador de destino é o *rt* para operações de load e para instruções do tipo R, ele é o *rd*.
- Assim, é preciso usar mais um multiplexador para definir que campo será usado para o registrador de destino.

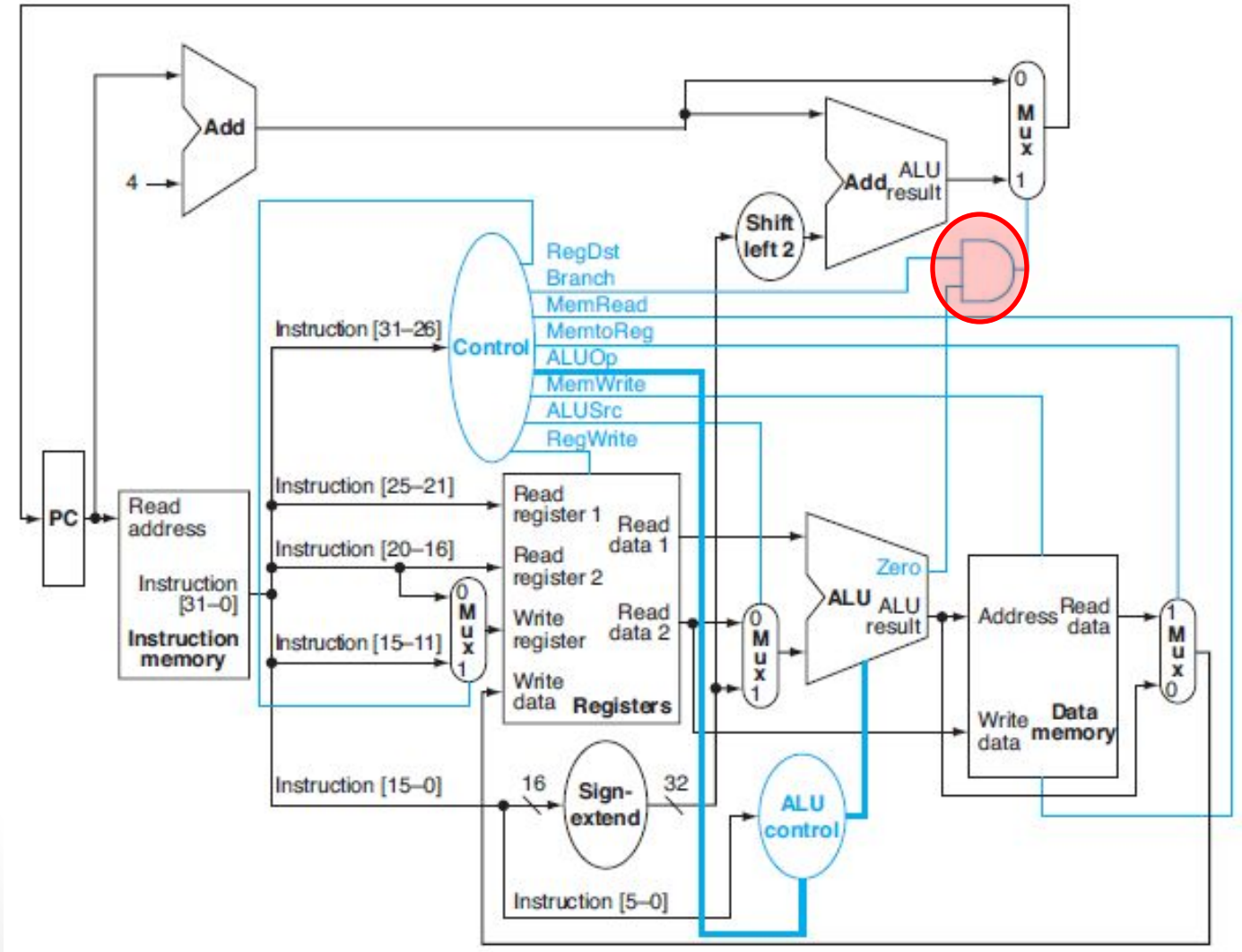
Caminho de dados das instruções

Atualizando o novo caminho de dados:



Caminho de dados das instruções

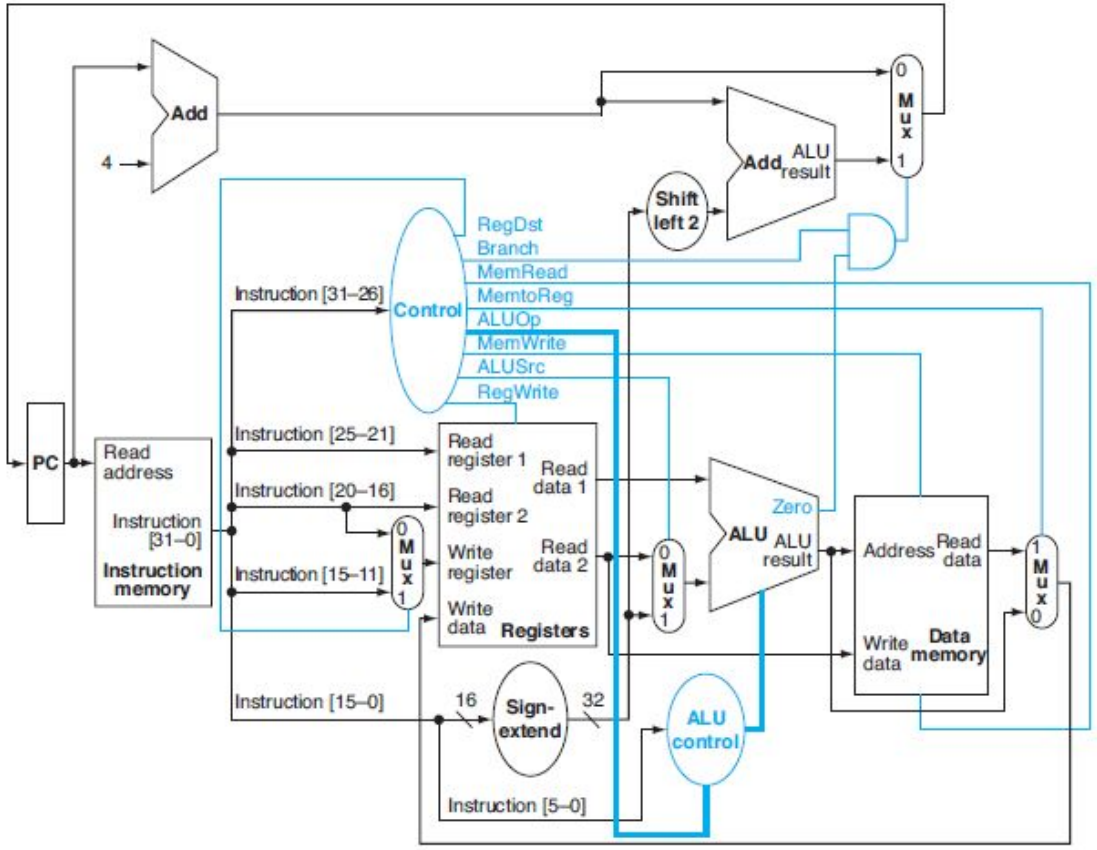
Porque é preciso adicionar essa AND?



Caminho de dados das instruções

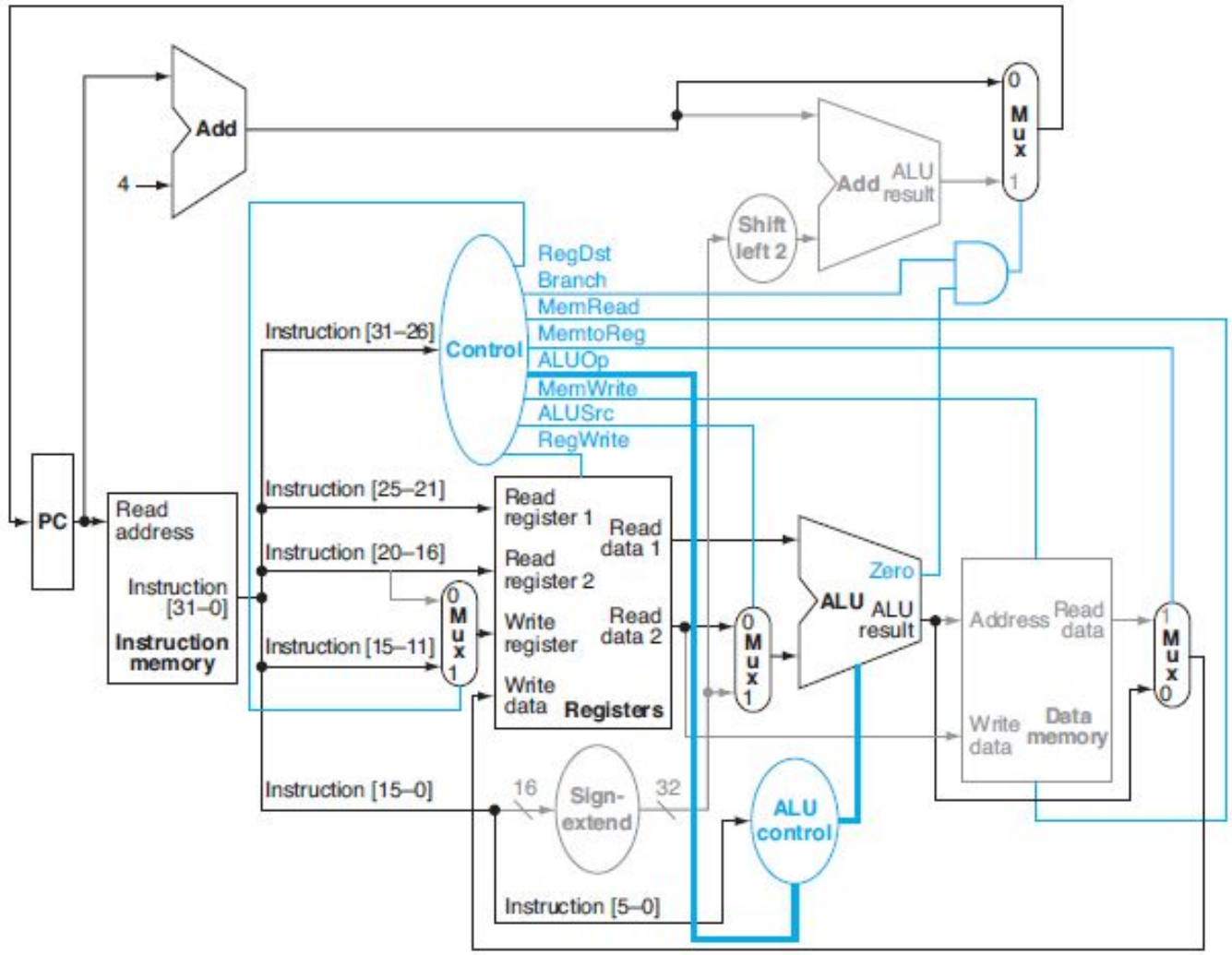
Definições dos sinais de controle conforme o tipo de instrução

Instruction	RegDst	ALUSrc	Memto-Reg	Reg-Write	Mem-Read	Mem-Write	Branch	ALUOp1	ALUOp0
R-format	1	0	0	1	0	0	0	1	0
lw	0	1	1	1	1	0	0	0	0
sw	X	1	X	0	0	1	0	0	0
beq	X	0	X	0	0	0	1	0	1



Caminho de dados das instruções

Instruções tipo R



Caminho de dados das instruções

Unidade de Controle

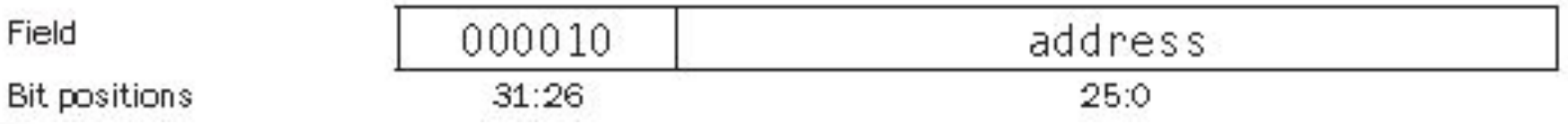
Instruction	RegDst	ALUSrc	Memto-Reg	Reg-Write	Mem-Read	Mem-Write	Branch	ALUOp1	ALUOp0
R-format	1	0	0	1	0	0	0	1	0
lw	0	1	1	1	1	0	0	0	0
sw	X	1	X	0	0	1	0	0	0
beq	X	0	X	0	0	0	1	0	1

Sinal	Efeito quando inativo	Efeito quando ativo
RegDst	Reg de escrita vem do campo rt (bits 20:16)	Reg de escrita vem do campo rd (bits 15:11)
RegWrite	Nenhum	Registrador de escrita recebe o dado de escrita
ALUSrc	O segundo operando da ALU vem da segunda saída do banco de registradores	O segundo operando da ALU consiste nos 16 bits mais baixos da instrução com sinal estendido.
PCSrc	O PC é substituído pela saída do somador que calcula o valor de PC + 4	O PC é substituído pela saída do somador que calcula o destino do desvio.
MemRead	Nenhum	O conteúdo da memória de dados designado pela entrada Endereço é colocado na saída Dados de leitura.
EscreveMem	Nenhum	O conteúdo da memória de dados designado pela entrada Endereço é colocado na saída Dados da leitura.
MemparaReg	O valor enviado à entrada Dados para escrita do banco de registradores vem da ALU	O valor enviado à entrada Dados para escrita do banco de registradores vem da memória de dados.

Caminho de dados das instruções

Instrução jump

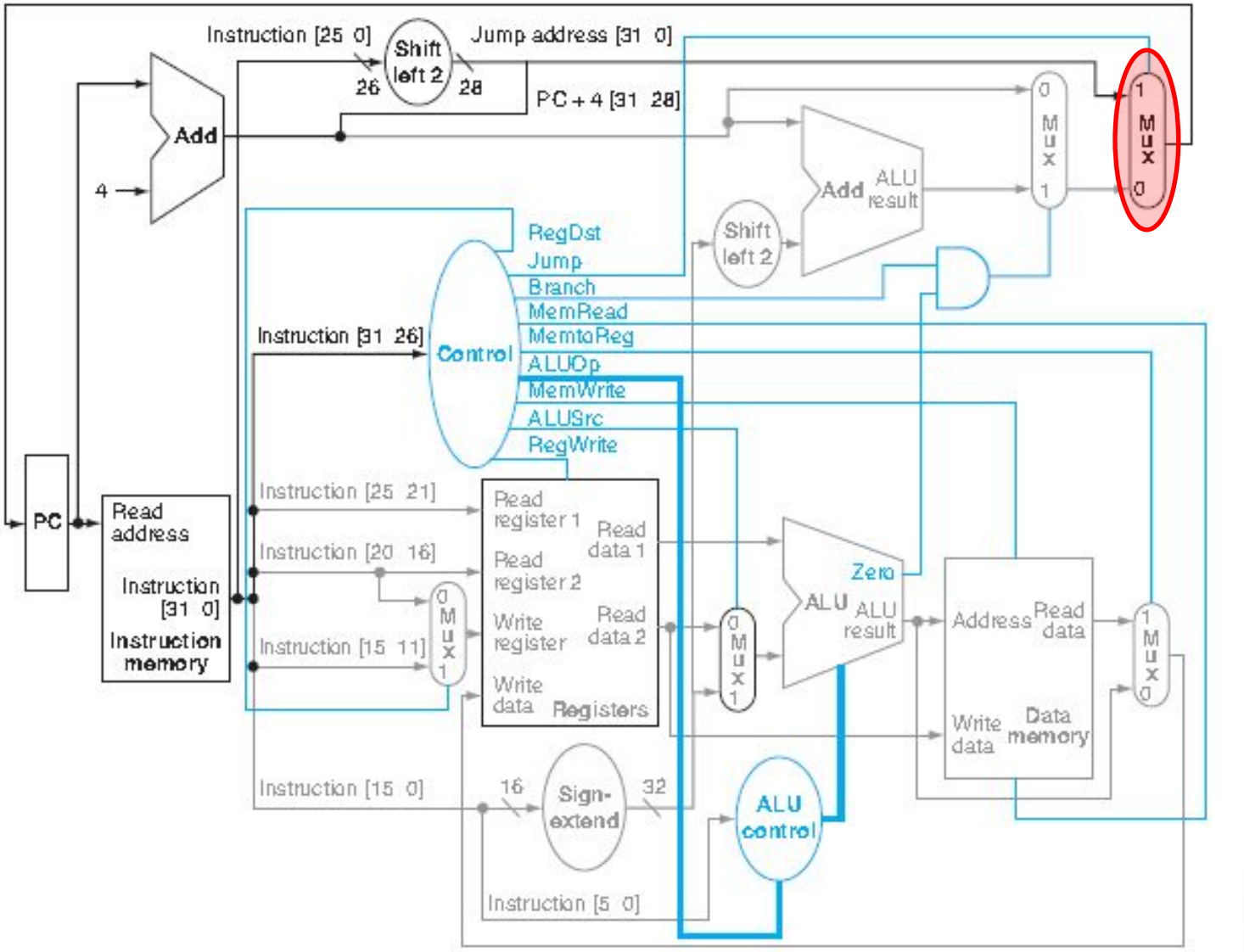
Relembrando:



1) 4 bits superiores do PC + 4 (bits 31:28 do endereço da instrução imediatamente seguinte);
2) Campo de 26 bits imediato da instrução jump;
3) Os 2 bits menos significativos: 00b

Caminho de dados das instruções

Instrução jump



Sinais de controle

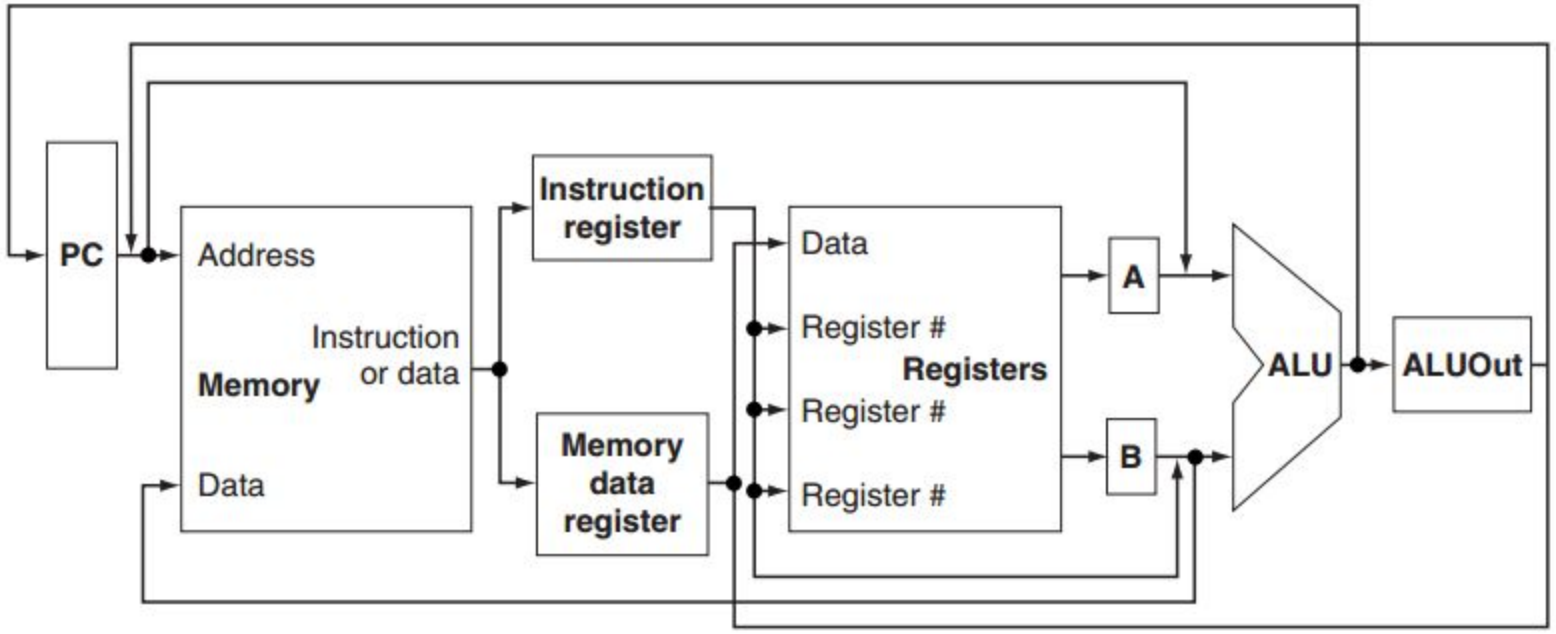
Input or output	Signal name	R-format	lw	sw	beq
Inputs	Op5	0	1	1	0
	Op4	0	0	0	0
	Op3	0	0	1	0
	Op2	0	0	0	1
	Op1	0	1	1	0
	Op0	0	1	1	0
Outputs	RegDst	1	0	X	X
	ALUSrc	0	1	1	0
	MemtoReg	0	1	X	X
	RegWrite	1	1	0	0
	MemRead	0	1	0	0
	MemWrite	0	0	1	0
	Branch	0	0	0	1
	ALUOp1	1	0	0	0
	ALUOp0	0	0	0	1

É possível combinar alguns dos sinais de controle.
Substituir um por outro ou pelo seu inverso?

Caminho de dados multiciclo

Arquiteturas multiciclo

Como as instruções são divididas em etapas similares, podemos usar cada etapa para construir uma **arquitetura multiciclo**.

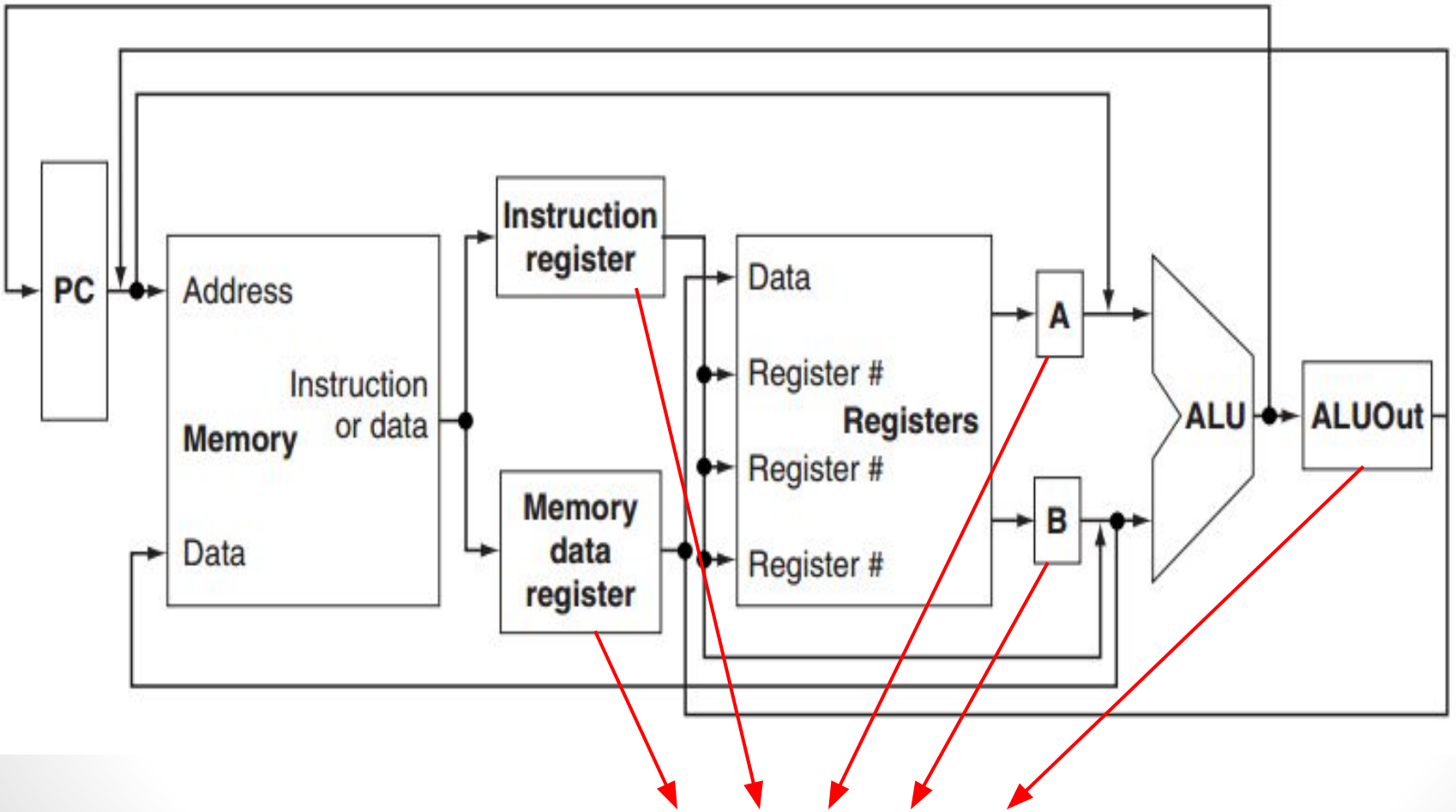


Caminho de dados multiciclo

Por que uma implementação de ciclo único não é utilizada atualmente?

1. A frequência de clock ficaria limitada pela pior instrução (caminho mais longo);
2. Embora CPI seja 1, o desempenho de uma proposta de ciclo 1 é sempre pior;
3. Quando possui várias instruções, as situações anteriores pioram consideravelmente;
4. Não permite tornar o caso comum mais rápido.

Caminho de dados multiciclo



Registradores requeridos em uma implementação multiciclo

Caminho de dados multiciclo

- Em uma implementação multiciclo cada etapa do circuito levará um ciclo de clock.

  Exemplos:
  - leitura/ escrita na memória - 1 ciclo
  - leitura/ escrita no banco de registradores – 1 ciclo
  - operação com a ULA - 1 ciclo

- No final do ciclo, todos os dados gerados precisam ser armazenados em um registrador.

- Uma implementação multiciclo é a base para aplicação de pipeline.

Caminho de dados multiciclo

- Os dados usados pela mesma instrução em um ciclo posterior precisam ser armazenados em um desses registradores adicionados na arquitetura.

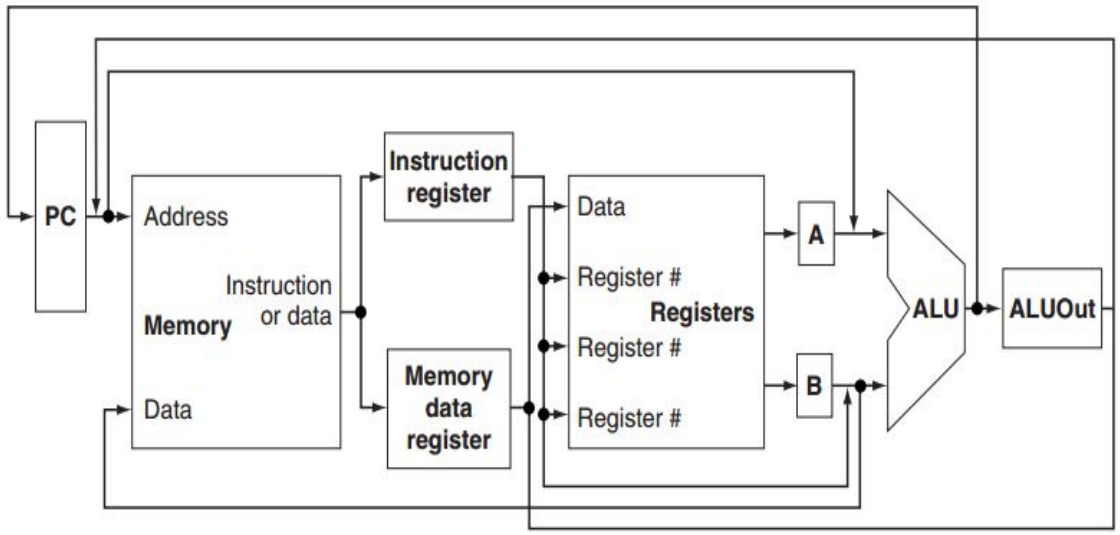
- Os dados usados pelas instruções subsequentes em um ciclo de clock posterior são armazenados em um dos elementos de estado visíveis ao programador: memória, banco de registradores, PC.

Caminho de dados multiciclo

Registradores acrescentados na implementação multiciclo:

- Registrador de Instrução – Instruction Register (RI)
- Registrador de dados da memória – Memory Data Register (MDR).

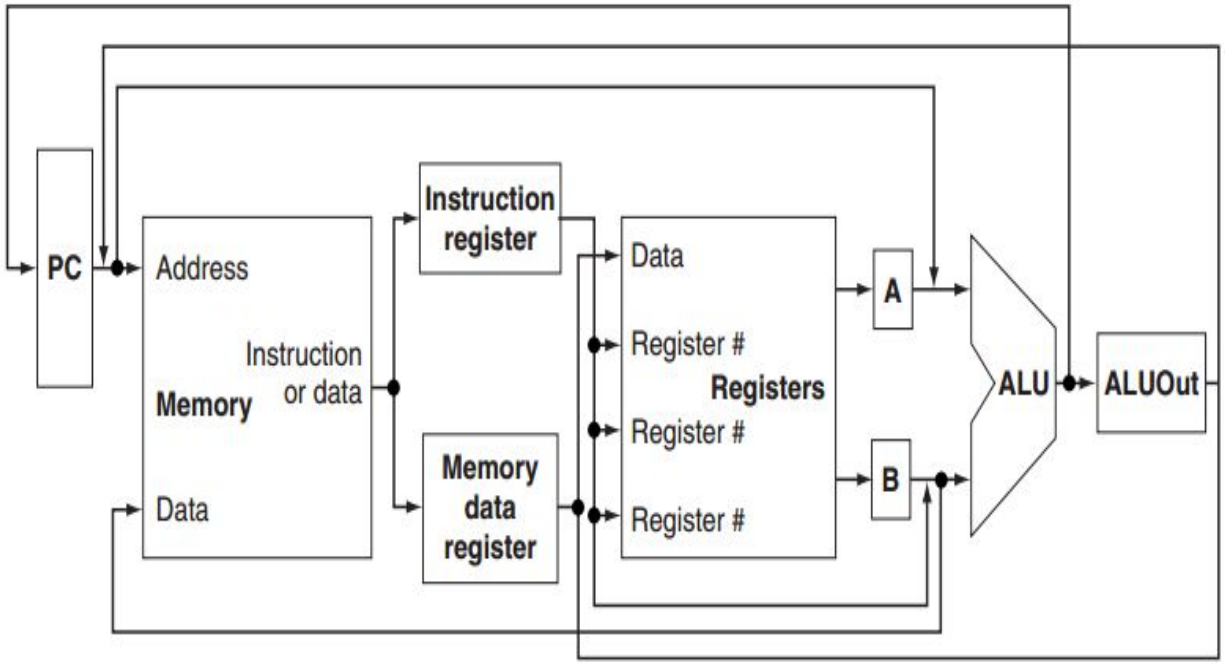
RI e MDR são usados para salvar a saída da memória referentes a uma leitura de instrução e uma leitura de dados, respectivamente.



Caminho de dados multiciclo

Registradores acrescentados na implementação multiciclo:

- Registradores A e B usados para conter os valores dos registradores operandos lidos do banco de registradores.
- Registrador ALUOut – registrador de saída da ALU



Arquiteturas multiciclo

- Em uma arquitetura **multiciclo,** cada etapa na execução levará 1 ciclo de clock. Só que agora um ciclo de clock é definido com um tempo muito menor.
- A vantagem agora é que **uma unidade funcional pode ser utilizada mais de uma vez** por instrução, desde que seja usada em diferentes ciclos de clock.
- Considerando a tabela abaixo, qual seria o ciclo de clock para uma arquitetura multiciclo?

Instruction	Instr fetch	Register read	ALU op	Memory access	Register write	Total time
lw	200 ps	100 ps	200 ps	200 ps	100 ps	800 ps
sw	200 ps	100 ps	200 ps	200 ps		700 ps
R-format	200 ps	100 ps	200 ps		100 ps	600 ps
beq	200 ps	100 ps	200 ps			500 ps
1 ps = one picosecond = 1 x 10^{-12}s - see S.I. prefixes						

Arquiteturas multiciclo

- Sendo assim, é possível reduzir a quantidade de hardware;
- É possível utilizar uma única unidade de memória para instruções e dados;
- Existe uma única ALU;
- Alguns registradores precisam ser adicionados após cada unidade funcional para conter a saída dessa unidade até o valor ser usado em um ciclo de clock subsequente.

O que muda no projeto de HW entre uma implementacão de ciclo único comparada a uma de multiciclo.