"Se não der certo da primeira vez, chame de versão 1.0."

CPU Monociclo - Construção

Paulo Ricardo Lisboa de Almeida

DSBD

UFPR
UNIVERSIDADE FEDERAL DO PARANÁ

Implementação Básica

Construir um processador MIPS32 básico, que implementa o seguinte.

- Instruções de referência a memória: **sw** e **lw**
- Instruções lógicas e aritméticas: **add**, **sub**, **and**, **or** e **slt**
- Instruções de desvio: **beq** e **j**

Essa implementação segue a descrita no livro base da disciplina: Patterson e Henessy (2014).

Quem usa MIPS?

Quem usa MIPS?

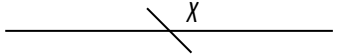
Quem usa MIPS?

Convenções Iniciais

O tamanho da palavra no MIPS32 é de 32 bits.

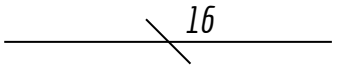
Assume-se que as linhas de conexão no projeto são barramentos de 32 bits.
Como se tivéssemos 32 fios em paralelo.

Caso o barramento tenha uma largura diferente de 32, isso será indicado da seguinte forma (*x* indica a largura do barramento).



Exemplos:

Um barramento de 32 bit:s

Um barramento de 16 bits: 

O Caminho de Dados

Analisando os principais componentes necessários para executar um programa são:

Uma **memória**, para armazenar as **instruções** do programa.

Entrada: O endereço da instrução desejada.

Saída: A instrução no endereço apontado.

Um registrador que vai armazenar o endereço da próxima instrução a ser executada.

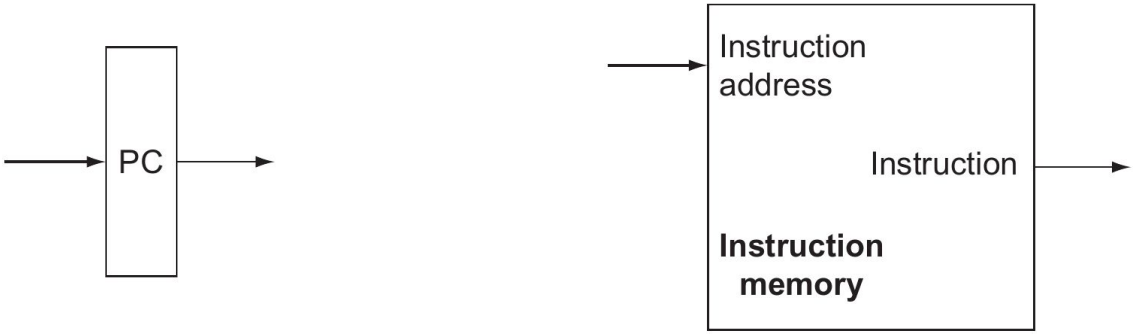
Registrador PC no MIPS.

Entrada: O próximo endereço a ser armazenado no registrador.

Saída: O endereço corrente.

O Caminho de Dados

Dica: é comum iniciar um valor com 0x para indicar que ele está sendo representado em hexadecimal.

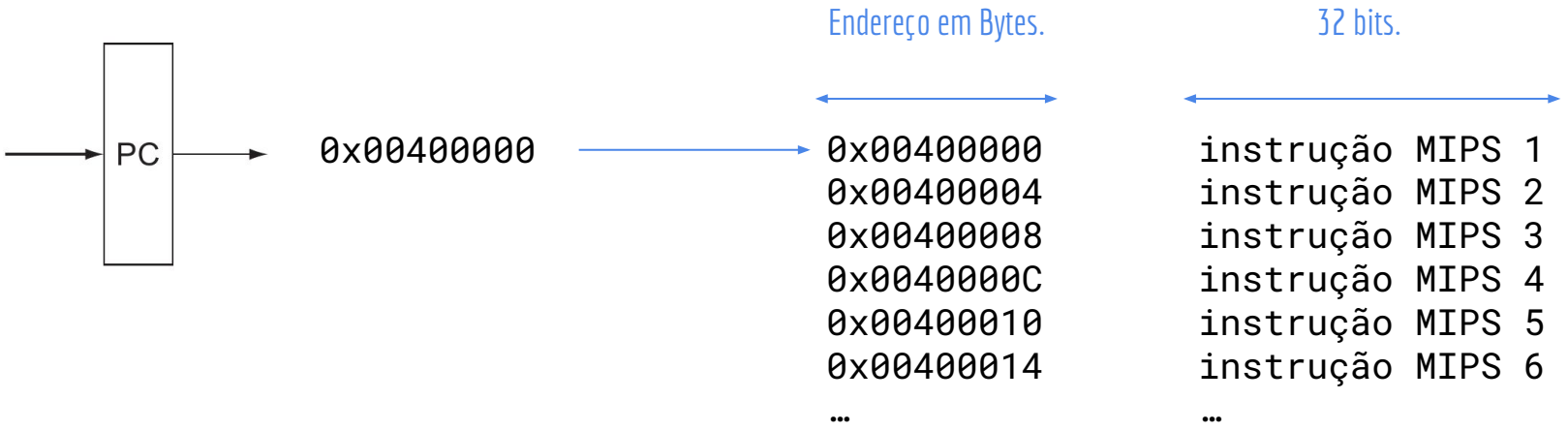



```
pc = 0x00400000 ──────────►  0x00400000    instrução MIPS 1
                             0x00400004    instrução MIPS 2
                             0x00400008    instrução MIPS 3
                             0x0040000C    instrução MIPS 4
                             0x00400010    instrução MIPS 5
                             0x00400014    instrução MIPS 6
                             …             …
```

Incrementando o PC

Caso não hajam desvios, após a execução da instrução atual, executar a próxima instrução.

O que fazer com o PC? Como?



Incrementando o PC

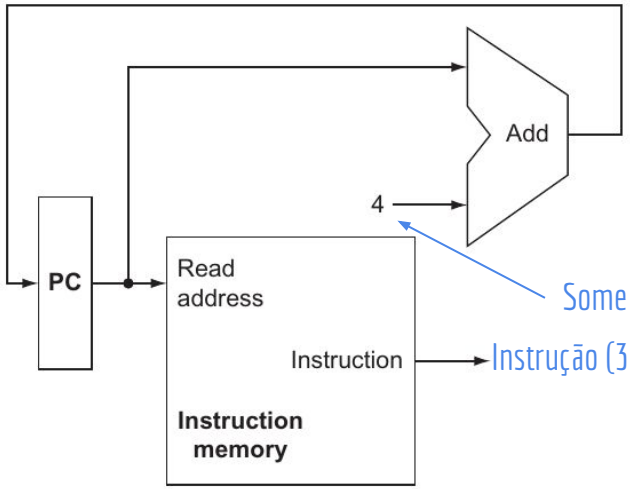
Caso não hajam desvios, após a execução da instrução atual, executar a próxima instrução.

Adicionar 4 no valor atual do PC.

Memória endereçada em bytes, logo “saltamos” 4 bytes = 32 bits

Somador

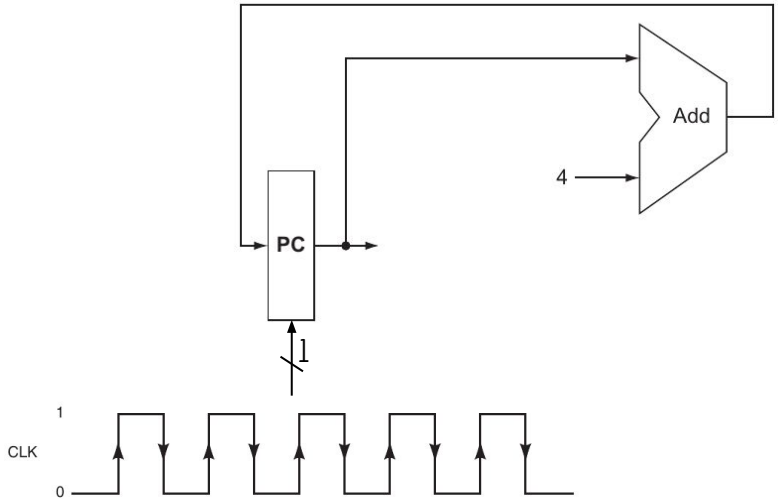
Ligando Tudo



Some 4_{10} no endereço atual de PC.

Instrução (32 bits) armazenada no endereço apontado por PC.

Sincronização



Sinais de **clock** são adicionados nos elementos de estado (sequenciais).

Ex.: O registrador PC só vai ler a entrada na transição de clock.

Enquanto não há transição, o valor antigo de PC é enviado para a saída, e lido pelo adder.

O sinal de clock vai ser **omitido para simplificar o raciocínio.**

Mas **sempre assuma que existe sincronismo**, para que o sinal anterior não seja "atropelado" pelo próximo.

Ligando Tudo

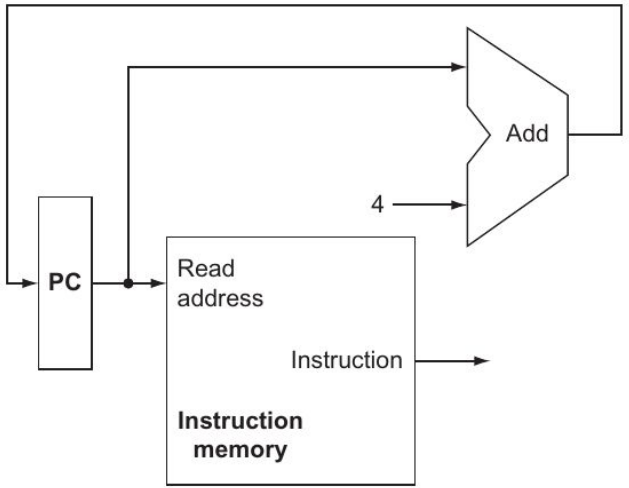
O “loop” principal está pronto.

A instrução é enviada para execução, e o PC é incrementado em 4 para apontar para a próxima.

O que é feito com a instrução agora depende do seu formato.

Começando com instruções básicas do tipo-R.

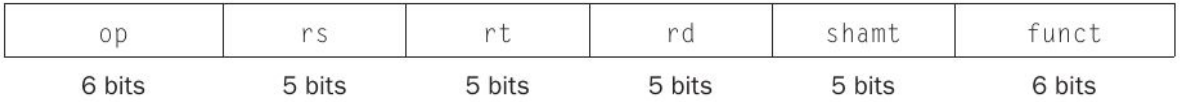
Como são as instruções do tipo-R?



Instruções do Tipo-R

Exemplo: `add $a0,$a1,$a2`

Leem **dois registradores**, executam uma operação aritmética **em uma ALU** (soma, subtração, deslocamento,...), e **armazenam** o resultado em um **terceiro registrador.**



Banco de Registradores

Necessária uma estrutura denominada banco de registradores.

Contém os 32 registradores do MIPS32.

Entradas: Endereço do registrador de leitura 1;
Endereço do registrador de leitura 2;
Endereço do registrador de escrita;
Dados a serem escritos no registrador de escrita.

Saídas: Dados do registrador de leitura 1;
Dados do registrador de leitura 2.

Pergunta: Qual o tamanho de cada um dos barramentos de entrada e saída no banco de registradores?

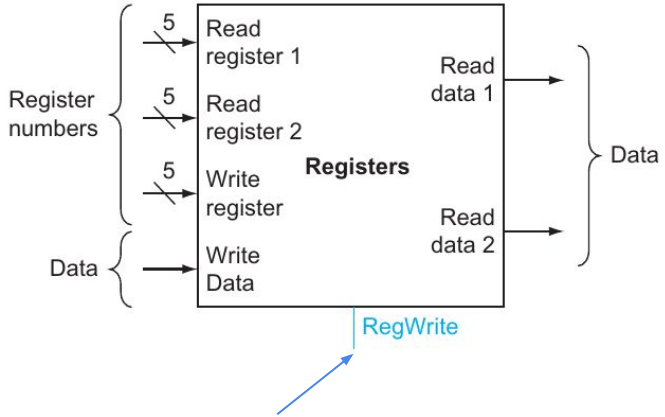
Banco de Registradores

Necessária uma estrutura denominada banco de registradores.

Contém os 32 registradores do MIPS32.

Entradas: Endereço do registrador de leitura 1;
Endereço do registrador de leitura 2;
Endereço do registrador de escrita;
Dados a serem escritos no registrador de escrita.

Saídas: Dados do registrador de leitura 1;
Dados do registrador de leitura 2.



Veremos adiante o uso desse sinal de 1 bit.

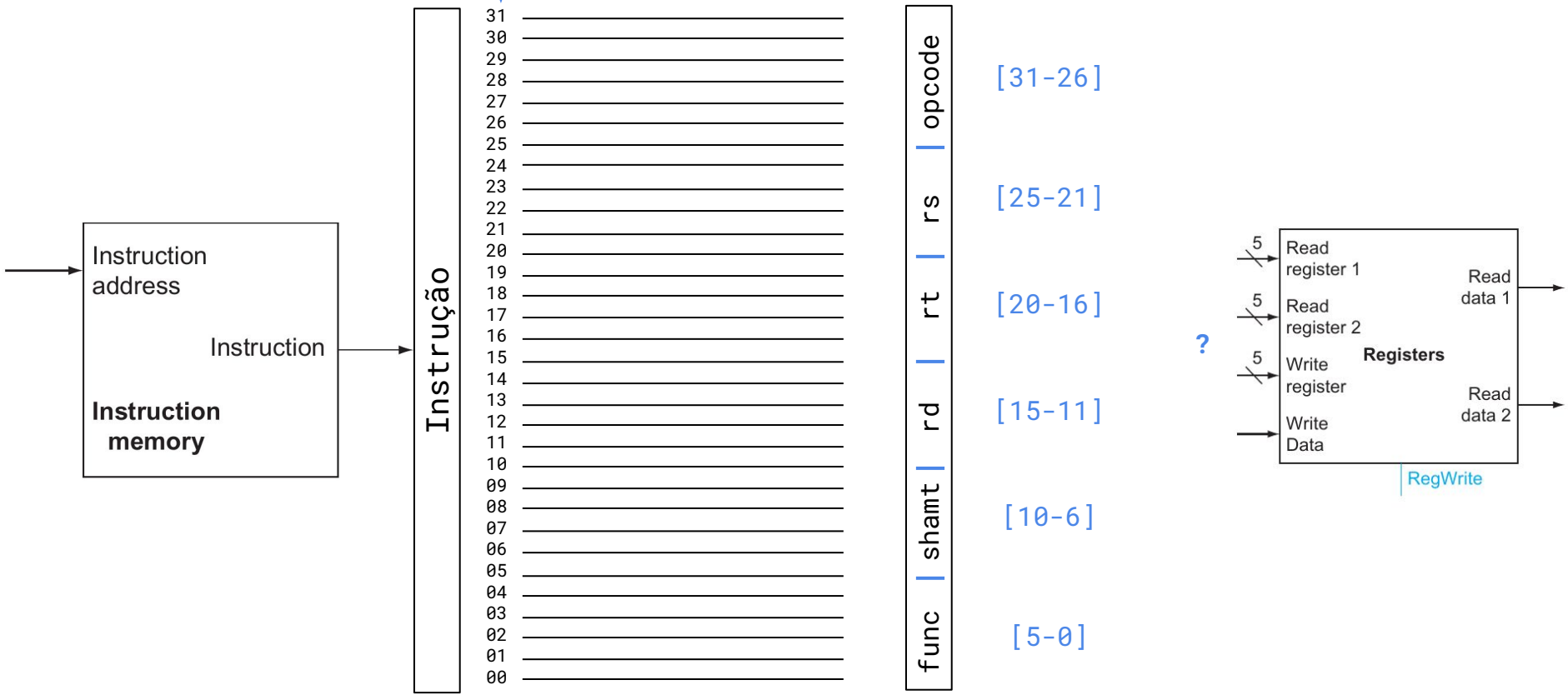
Fonte dos bits

Como definido, uma linha sem marcação tem largura de 32 bits.

Então na verdade são 32 linhas (barramento), endereçadas de 0 a 31.

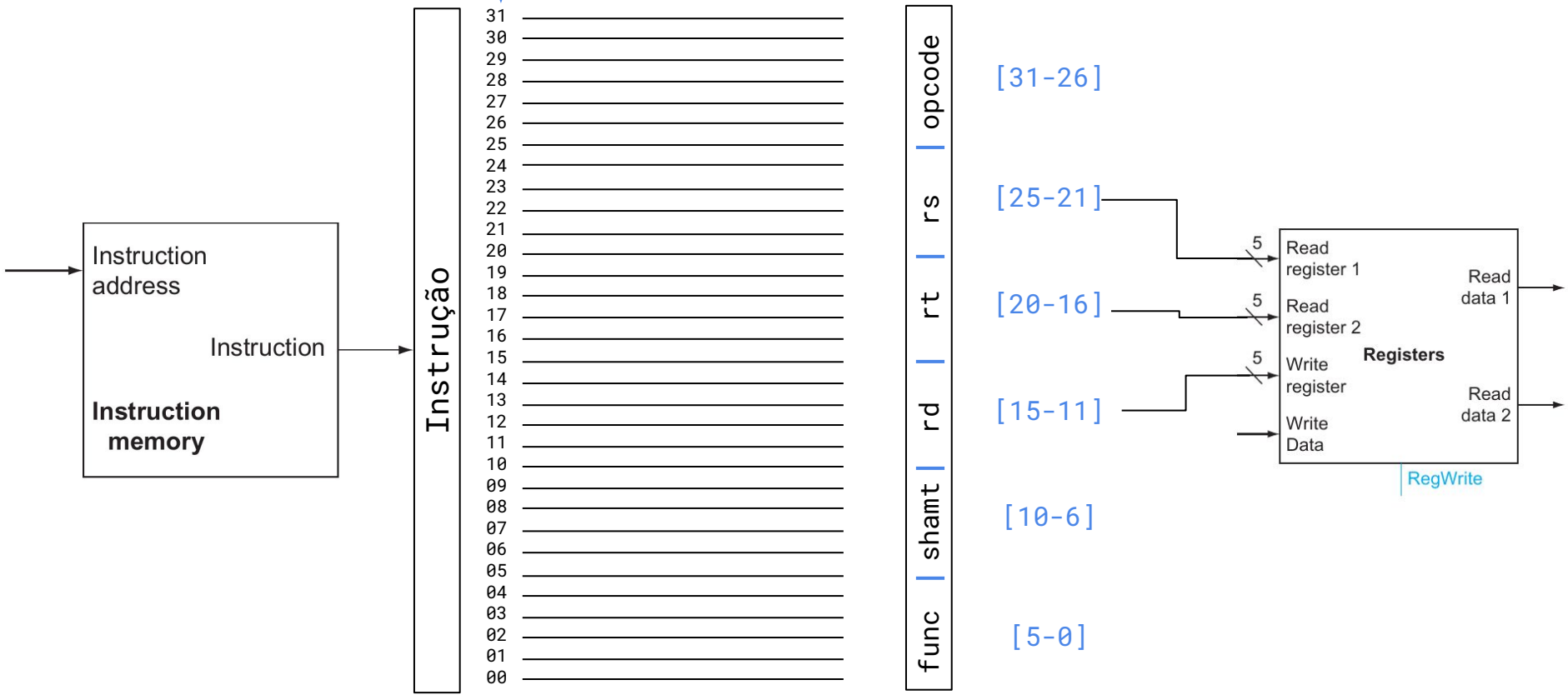
Fonte dos bits

Número da linha. Obs.: Para relacionar com o livro, vamos seguir uma abordagem little-endian, onde a instrução fica "ao contrário" no endereçamento.



Fonte dos bits

Número da linha. Obs.: Para relacionar com o livro, vamos seguir uma abordagem little-endian, onde a instrução fica "ao contrário" no endereçamento.



ALU

É necessário executar a operação especificada pela instrução.

Soma, subtração, deslocamento, ...

Arithmetic **L**ogic **U**nit - ALU.

Entradas: Dois valores de 32 bits.

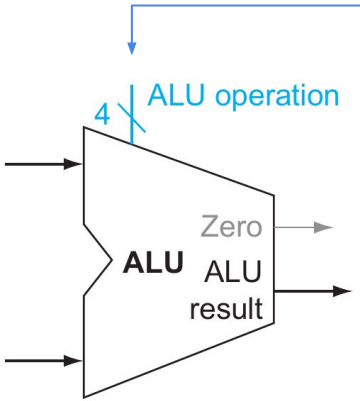
Entrada ALUop de 4 bits, que define qual a operação deve ser realizada com os valores.

Saídas: Uma saída de 32 bits com o resultado da operação.

Uma saída de 1 bit indicando se o resultado da operação foi zero.

Simplificará a construção de outros trechos do circuito.

ALU

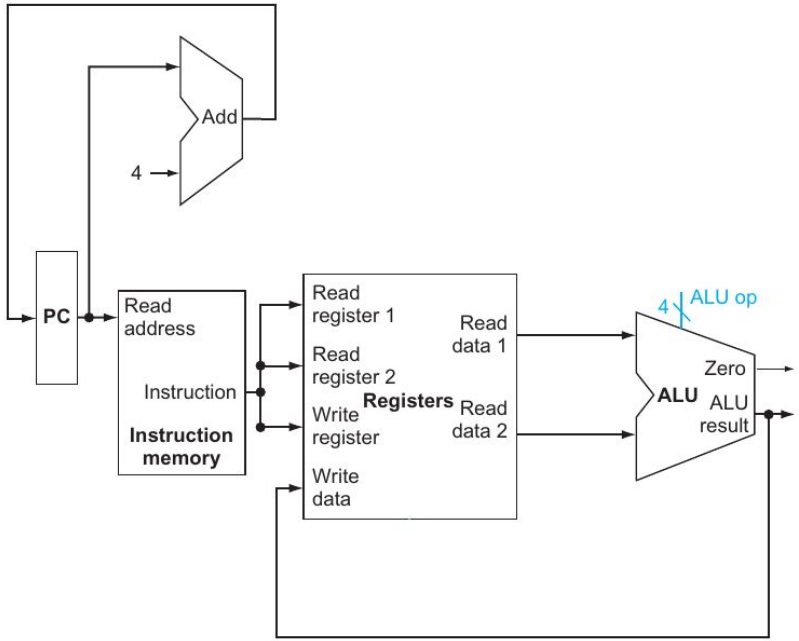


ALU control lines	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	set on less than
1100	NOR

Dentro da ALU podem existir diversos circuitos especialistas
- Somadores
- Operadores Lógicos
- ...

Qual desses circuitos será ativado depende do **ALUOp**

Ligando Tudo



Loads e Stores

Vamos adicionar instruções para *loads* e *stores*.

Instruções do tipo-I.

Qual a diferença dessas instruções para o tipo-R?

Loads e Stores

`lw $t1, valor_offset($t2)`

Carrega para $t1 o conteúdo apontado pelo endereço de memória [$t2 + valor_offset].

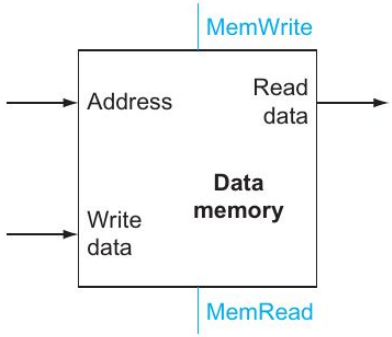
`sw $t1, valor_offset($t2)`

Armazena no endereço apontado por ($t2 + valor_offset) o conteúdo de $t1.

O Offset é representado em complemento de 2.

Pode ser positivo ou negativo.

Loads e Stores



Necessária uma **memória para dados.**

Entradas: Endereço de memória.

Dados a ser escrito.

Sinais de Controle MemWrite e MemRead.

Indica se a memória deve escrever na posição, ou;

Se deve ler o dado especificado na posição e direcioná-lo para a saída.

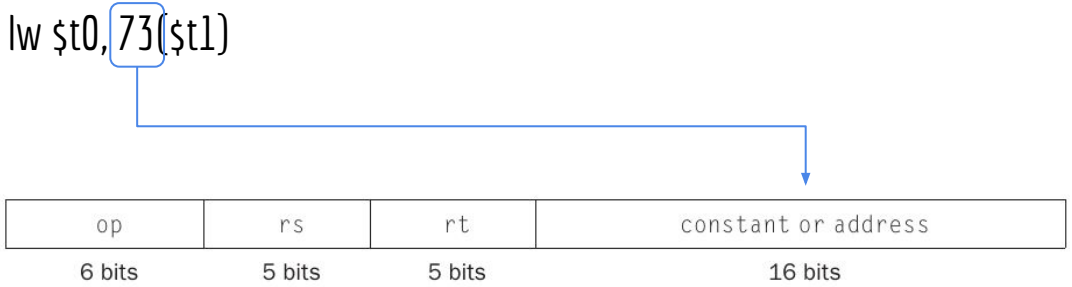
Saída: Dado lido pela memória.

Loads e Stores

O campo de constante contém 16 bits.

Somado com o registrador para obter o endereço de memória a ser lido/escrito.

Ex.: lw $t0, 73($t1)



Problema: estamos somando um valor de 16 bits(do imediato) com um de 32 (do registrador).

Podemos completar com zeros a esquerda?

Extensão de Sinal

Exemplo em complemento de 2:

Valor na base 10	Em binário com 16 bits	Em binário com 32 bits
73	00000000 01001001	00000000 00000000 00000000 01001001
-73	11111111 10110111	11111111 11111111 11111111 10110111

Extensão de Sinal

Exemplo em complemento de 2:

Valor na base 10	Em binário com 16 bits	Em binário com 32 bits
73	00000000 01001001	00000000 00000000 00000000 01001001
-73	11111111 10110111	11111111 11111111 11111111 10110111

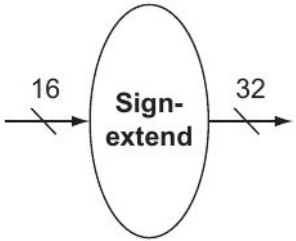
Não podemos simplesmente “completar com zeros”.
Depende se o número é positivo ou negativo!

Extensão de Sinal

Vamos utilizar um componente para extensão de sinal.

Dado um sinal de 16 bits, gera o seu correspondente em 32 bits.

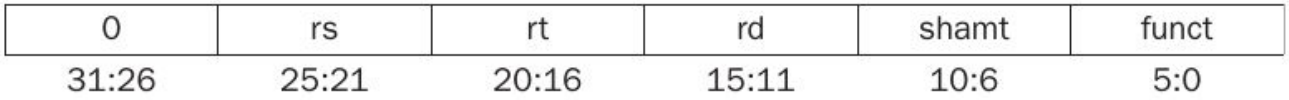
Leva em consideração o complemento de 2 para gerar o sinal correto.



Multiplexadores extras

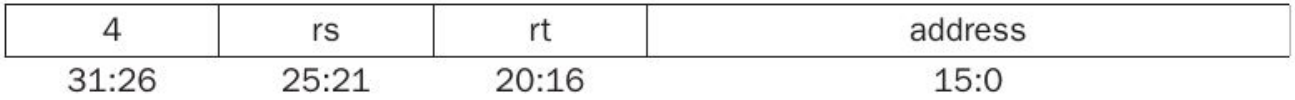
Exemplos de opcode

Tipo-R



Loads e Stores

Branches



Multiplexadores extras

Exemplos de opcode

Tipo-R

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

Loads e Stores

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

Branches

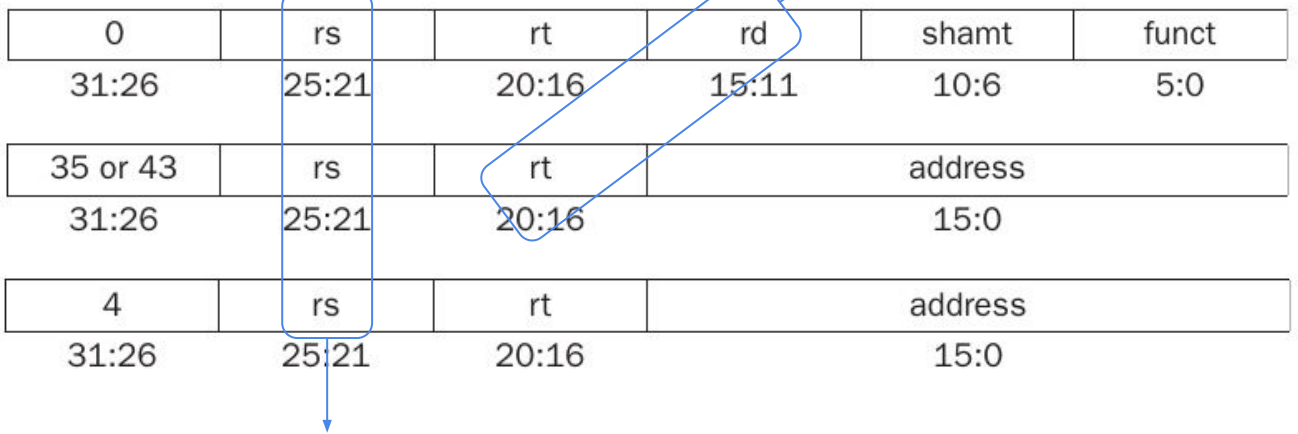
4	rs	rt	address
31:26	25:21	20:16	15:0

Primeiro fonte sempre está na mesma posição. Serve como 1° fonte para tipo-R, ou registrador base para lw e sw (de qualquer forma, deve ser enviado para a ALU).

Multiplexadores extras

Registrador de destino muda entre loads, e tipo-R. Para stores, não temos destino, e sim dois fontes (endereço base em rs, e o registrador com o valor a ser escrito rt).

Exemplos de opcode



Primeiro fonte sempre está na mesma posição. Serve como 1° fonte para tipo-R, ou registrador base para lw e sw (de qualquer forma, deve ser enviado para a ALU).

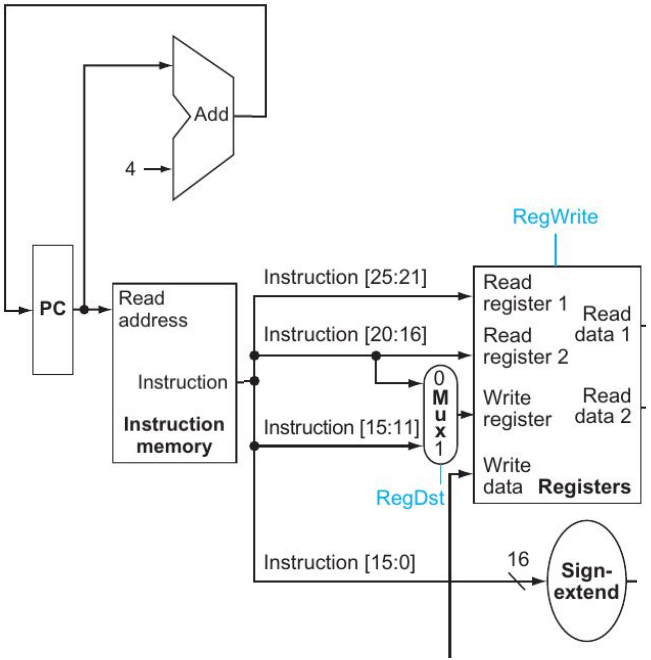
Ligando Tudo

Tipo-R

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

Load/Store

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



Necessário somar esse sinal (agora com 32 bits) com o registrador fonte 1, para obter o endereço de memória. Como?

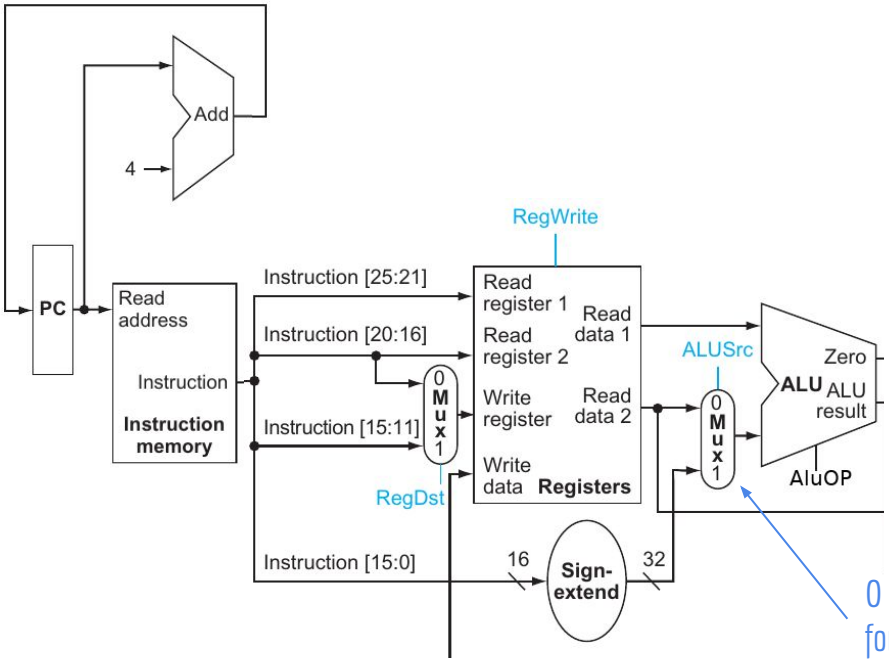
Ligando Tudo

Tipo-R

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

Load/Store

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



O segundo operando pode vir do segundo registrador fonte, ou do campo imediato, dependendo da instrução.

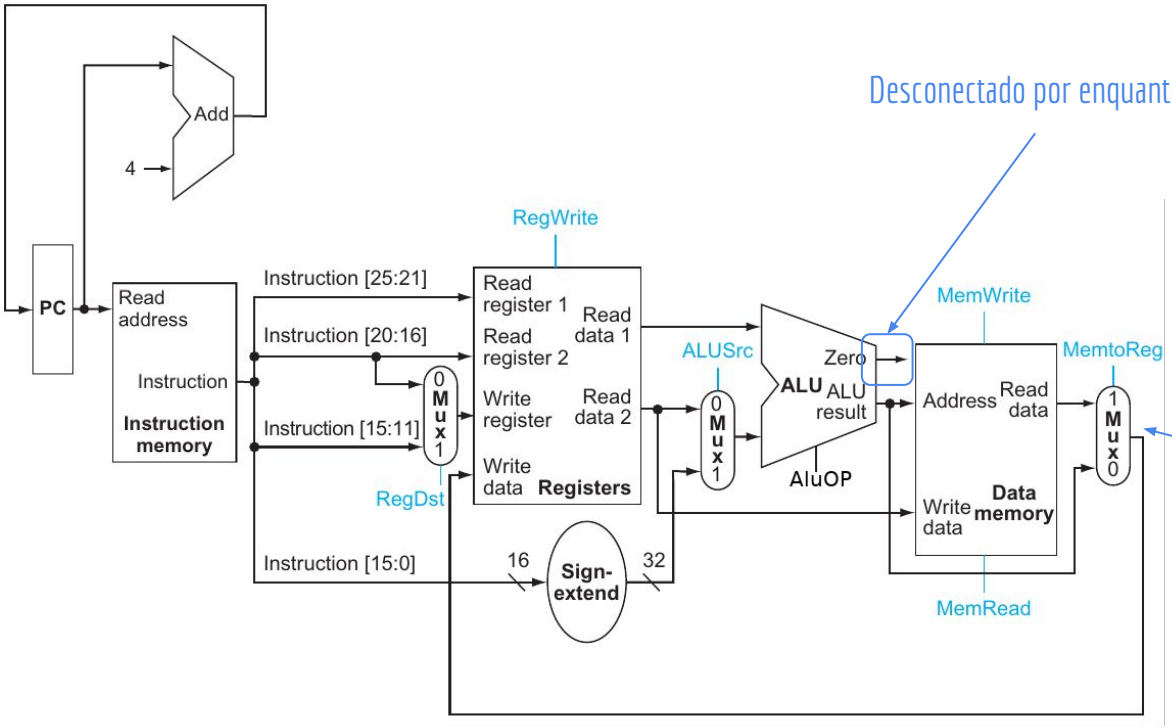
Ligando Tudo

Tipo-R

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

Load/Store

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



Desconectado por enquanto.

O dado a ser escrito no registrador pode vir da memória, ou do resultado da ALU, dependendo da instrução.

Branches

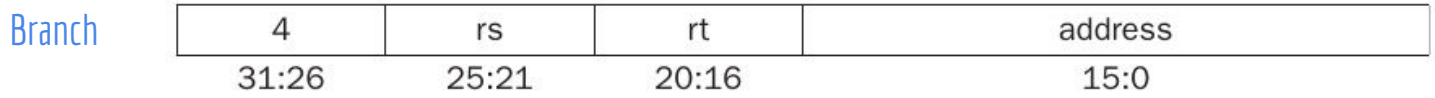
Um *branch* (desvio, galho) soma o valor de deslocamento (que está no campo imediato) ao **PC+4** caso os registradores rs e rt sejam iguais.

beq - branch if equal.

O endereço de deslocamento (**offset**) está em **palavras**, e não em bytes.

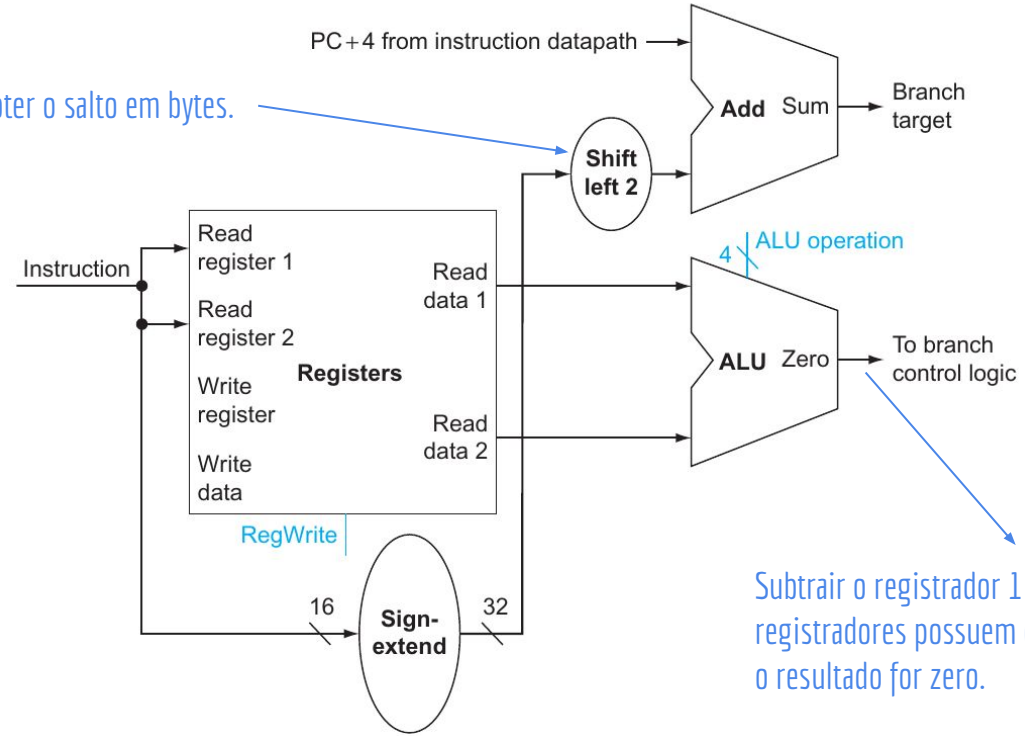
Deslocar 2x à esquerda para multiplicar por 4, para então obtermos o deslocamento em bytes.

Como comparar rs com rt?



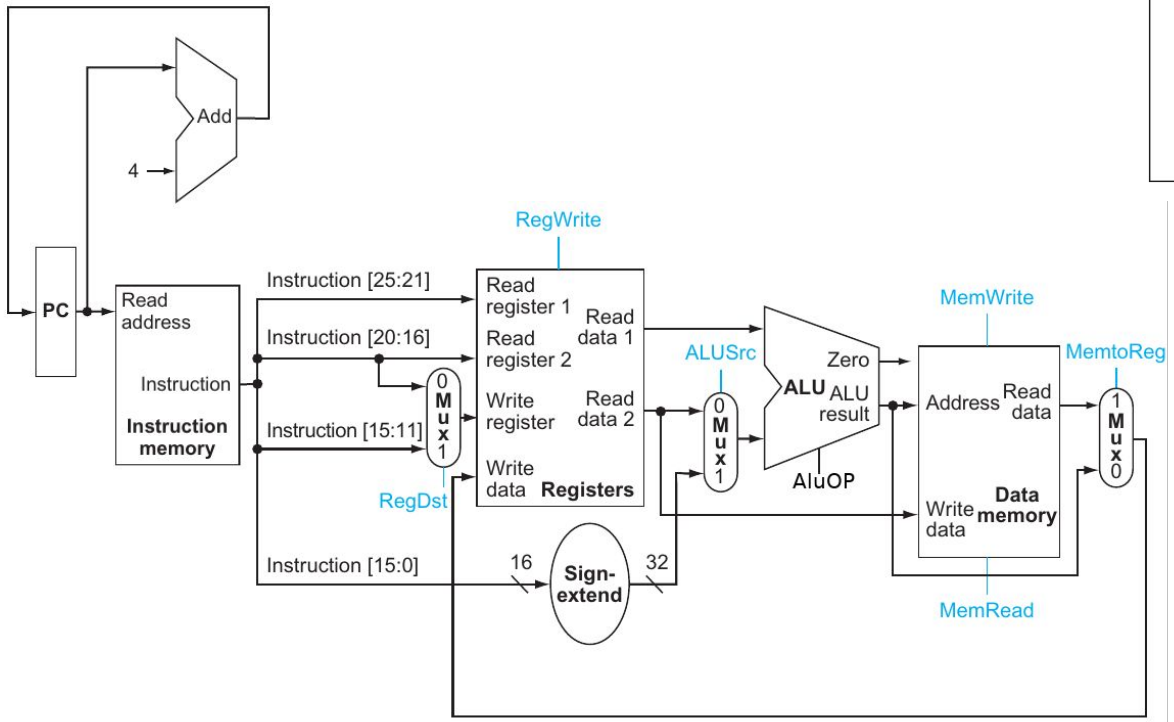
Branches

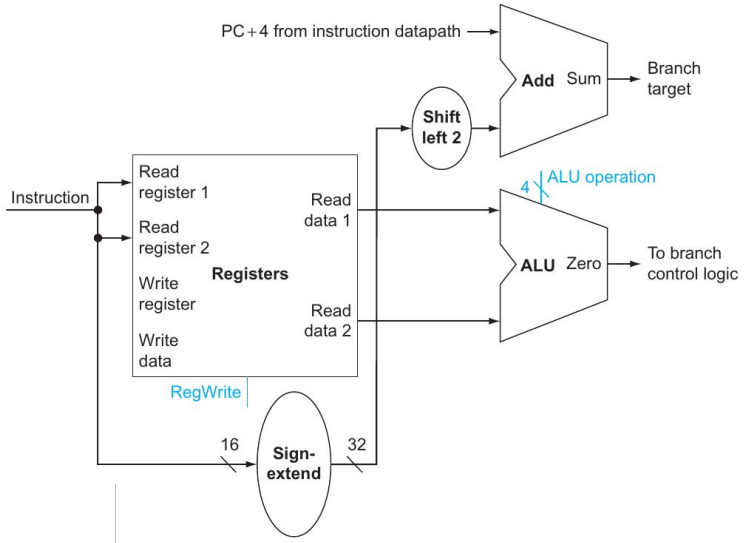
Deslocando 2x à esquerda para obter o salto em bytes.



Subtrair o registrador 1 do registrador 2. Os registradores possuem o mesmo conteúdo se o resultado for zero.

Faça você mesmo

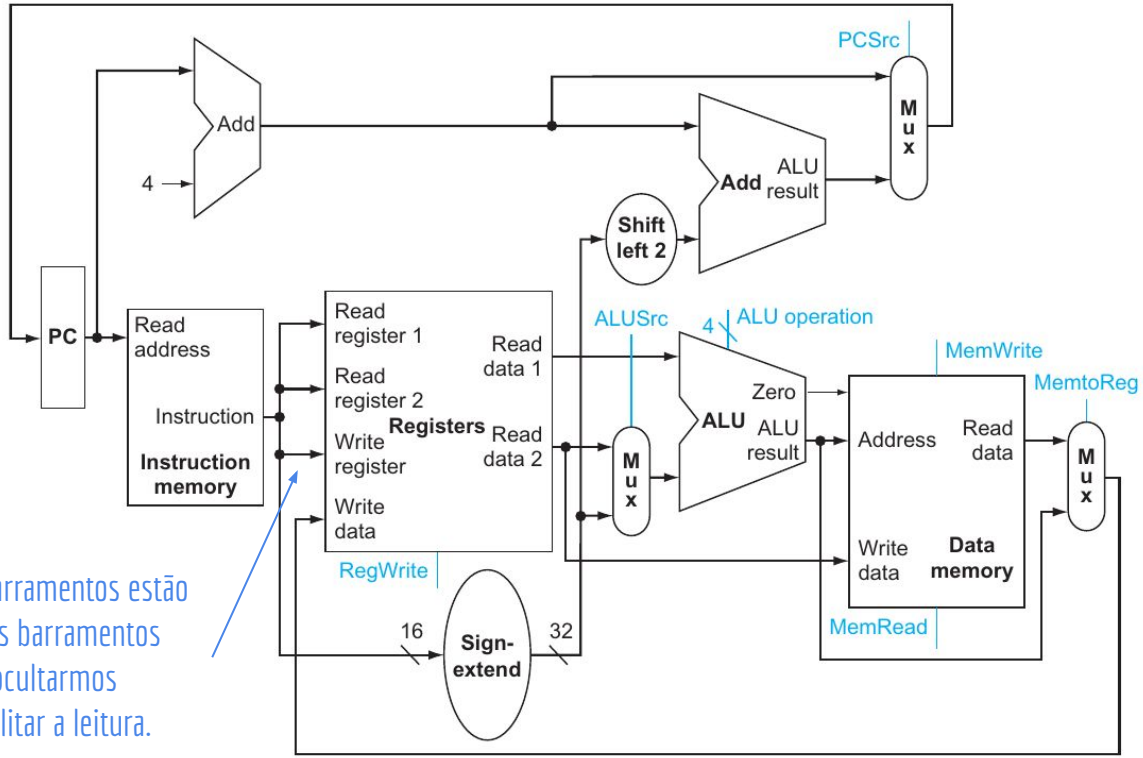




Dado o caminho de dados que inclui loads/stores, adicione o caminho de dados para branches. Ligue os componentes faltantes dos circuitos.

Branches

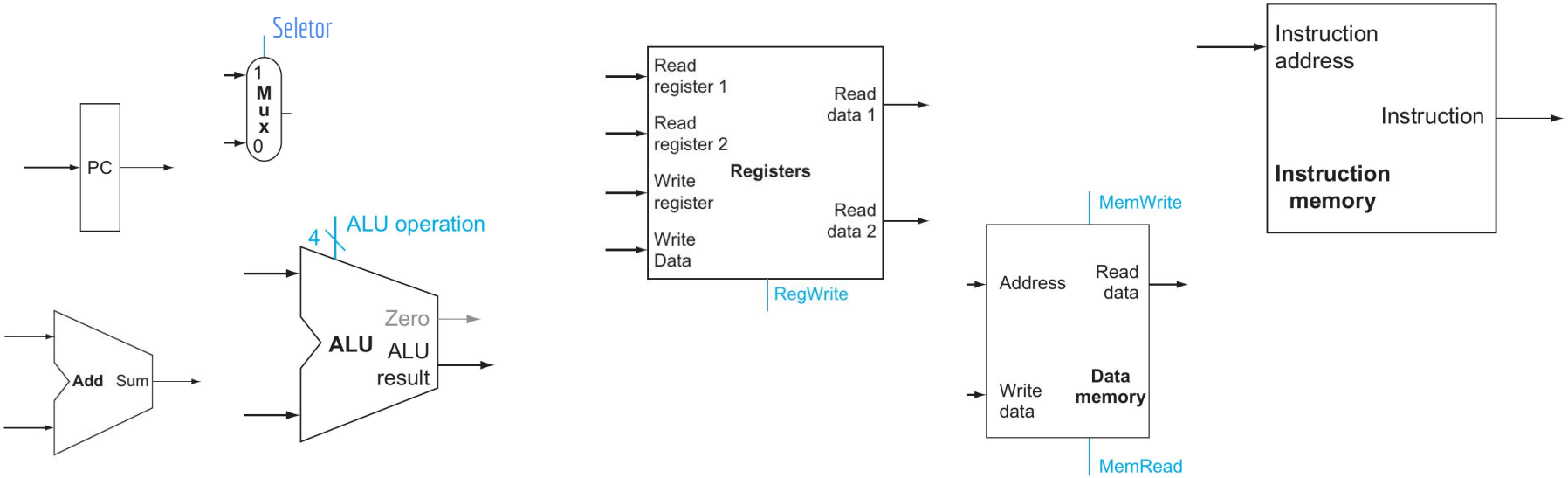
O novo endereço de PC pode ser PC+4, ou PC+4+deslocamento, dependendo da instrução. O sinal de controle PCSrc vai determinar isso.



Note que o mux e os tamanhos dos barramentos estão ocultos. Assuma que o mux existe e os barramentos de endereço são de 5 bits. É comum ocultarmos componentes nos diagramas para facilitar a leitura.

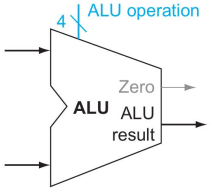
Exercícios

1. Sem olhar os slides anteriores, utilizando os seguintes componentes, monte novamente o processador. Marque no circuito a largura de cada barramento. Caso o barramento utilize somente algumas linhas de outro (ex.: n° do registrador de entrada), indique quais as linhas que são utilizadas por ele através da notação *[n-m]*.

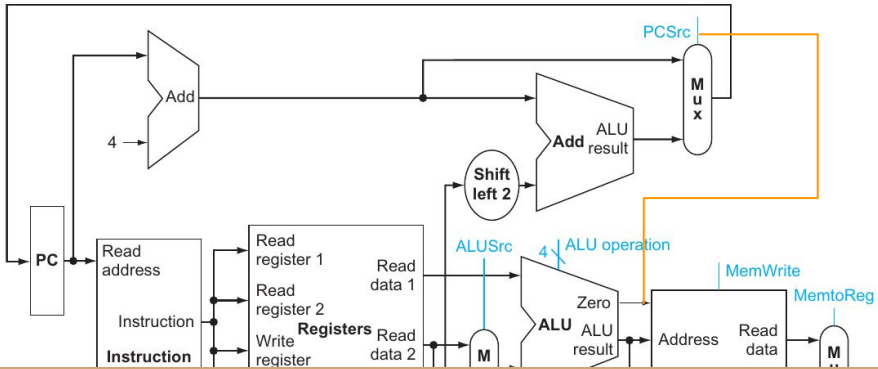


Exercícios

2. Os 4 bits que informam a ALU sobre qual a operação que deve ser executada podem vir de onde? Não precisa ligar no circuito, mas dê suas ideias sobre como poderíamos definir isso.



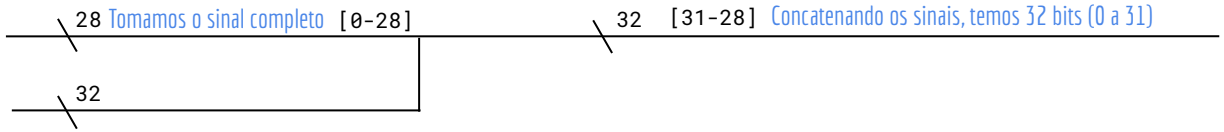
3. Considerando os branches, podemos ligar o seletor do multiplexador diretamente na saída zero da ALU, como feito em laranja no circuito (a saída tem apenas 1 bit, indicando se o resultado da operação foi zero)? Sim? Não? Por quê?



Exercícios

4. Considere as instruções de desvio incondicional (jumps).
    O endereço é especificado em palavras. Para obter o endereço em bytes é necessário multiplicar por 4 (deslocar para a esquerda 2x).
        Resultado é um valor com 28 bits.
        Emprestamos os 4 primeiros bits de PC+4, e os demais 28 bits do endereço do jump deslocado, para formar o endereço final.
            Endereçamento pseudodireto.
            Necessário concatenar a parte do sinal vêm do PC+4, com a parte do endereço do jump.

Utilize a seguinte notação para concatenar:

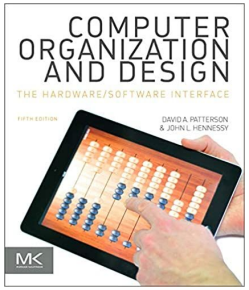


Dadas essas informações, implemente jumps no processador.
    Não precisa implementar os sinais de controle (pode deixar esses sinais “soltos”).

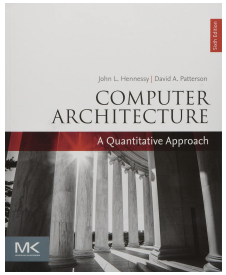
Importante: Lembre-se que os endereços estão invertidos por nossa máquina ser little-endian.

Referências

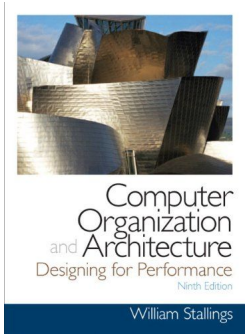
Patterson, Hennessy . Arquitetura e Organização de Computadores: A interface hardware/software. 2014.



Hennessy, Patterson. Arquitetura de Computadores: uma abordagem quantitativa. 2019.



Stallings, W. Organização de Arquitetura de Computadores. 2012.



Licença