

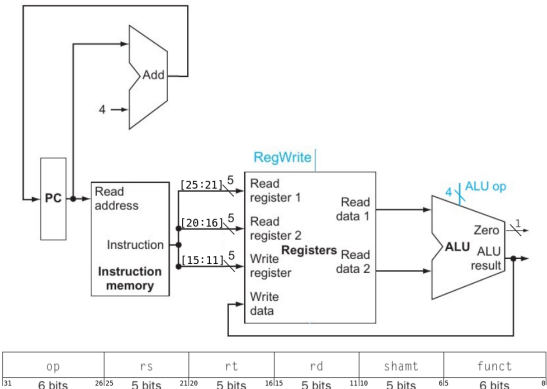
Construindo a CPU: Parte 2

Yuri Kaszubowski Lopes
UDESC

Anotações

Revisão: parte 1

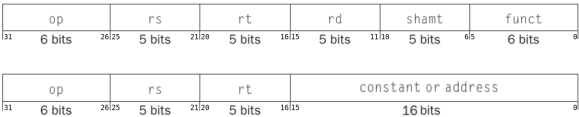
- Caminho de dados inicial para instruções do tipo-R



Anotações

Loads e Stores

- Vamos adicionar instruções para loads (lw) e stores (sw) de palavras (words)
 - Instruções do tipo-I

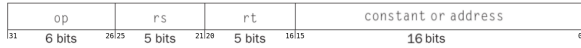


Anotações

Loads e Stores

- `lw $regDestino, deslocamento($regBase)`
 - ▶ `$regDestino = MEM[$regBase + deslocamento]`
- Deslocamento é um **imediato**
 - ▶ Pode ser positivo ou negativo
- `sw $regFonte, deslocamento($regBase)`
 - ▶ `MEM[$regBase + deslocamento] = $regFonte`

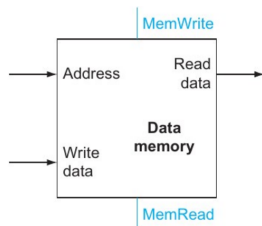
1 `lw $t0, 32($s3)`



Anotações

Loads e Stores

- Vamos precisar de 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
 - ▶ Saídas:
 - ★ Dado lido pela memória



Anotações

Loads e Stores

- O campo de constante (imediato) contém 16 bits
- Será 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)
- Simplesmente colocar zeros à esquerda não funciona em complemento a 2

Anotações

Extensão de sinal em complemento a 2

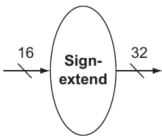
Valor base 10	Valor binário com 16 bits	Valor binário com 32 bits
73	00000000 01001001	00000000 00000000 00000000 01001001
-73	11111111 10110111	11111111 11111111 11111111 10110111

Não podemos simplesmente "completar com zeros" em complemento de 2. Depende se o número é positivo ou negativo!

Anotações

Extensão de sinal em complemento a 2

- 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



Anotações

Multiplexadores extras

Exemplos de opcode

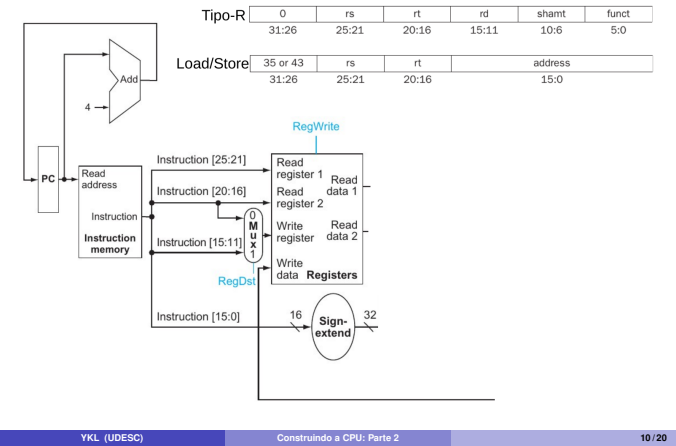
Tipo-R	0 31:26	rs 25:21	rt 20:16	rd 15:11	shamt 10:6	funct 5:0
Loads/Stores	35 or 43 31:26	rs 25:21	rt 20:16	address 15:0		
Branches	4 31:26	rs 25:21	rt 20:16	address 15:0		

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)

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

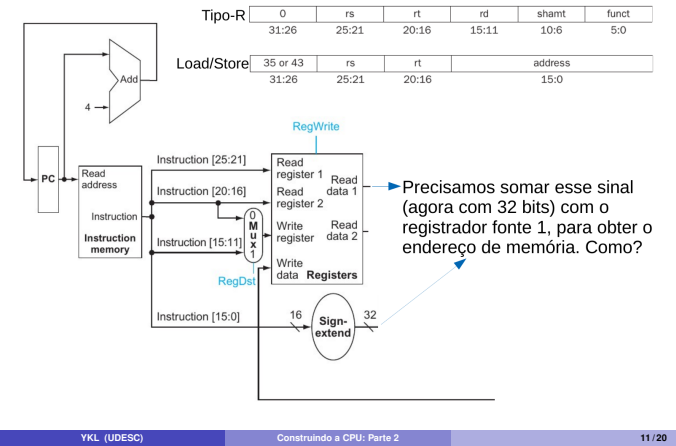
Anotações

Caminho de dados (datapath)



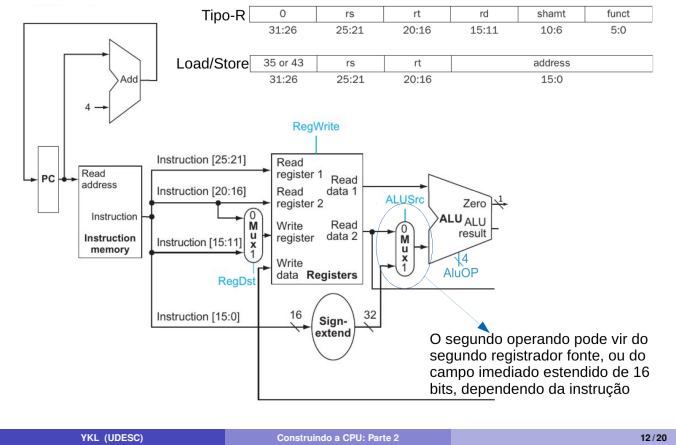
Anotações

Caminho de dados (datapath)



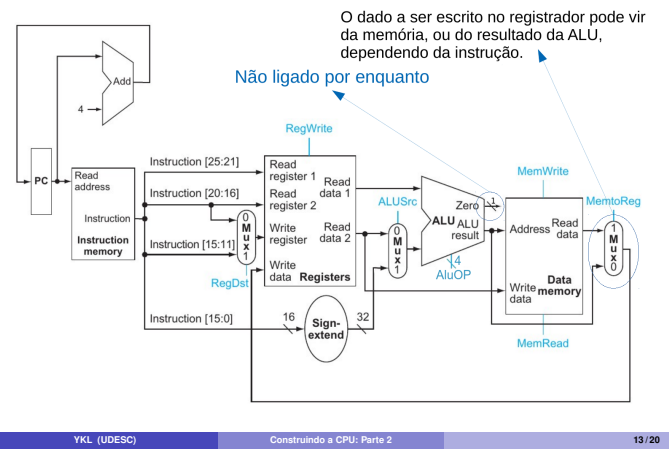
Anotações

Caminho de dados (datapath)



Anotações

Caminho de dados (datapath)



Anotações

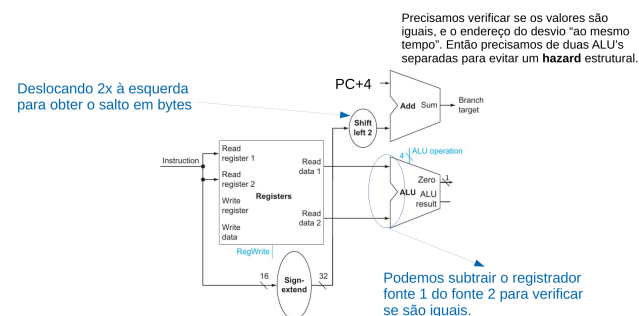
Desvio beq (Branch Equal)

- Um Desvio `beq` (Branch Equal) soma o valor de deslocamento (que está no campo imediato) ao `PC+4` caso os registradores `rs` e `rt` sejam iguais
- O valor de deslocamento (offset) está em **palavras** (de 4 bytes), e não em bytes
 - Devemos deslocar 2x à esquerda para multiplicar por 4, para então obtermos o deslocamento em bytes que será armazenado em PC
- Como podemos comparar `rs` com `rt`?

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

Anotações

Caminho de Dados: Branches



Anotações

Exercício

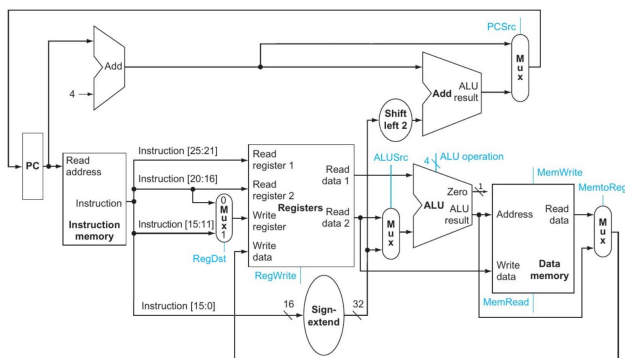
- ▶ Deve haver um multiplexador para escolher entre PC+4 ou PC+4+deslocamento
 - ▶ Este multiplexador é controlado pelo sinal $PCSrc$
 - ▶ O $PCSrc$ é definido pelo controlador com base na saída $Zero$ da ALU
 - Não é necessário mostrar o controlador e suas ligações neste exercício

Anotações

[illegible]

Resposta

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



Anotações

Exercício

- ➊ Descreva cada um dos sinais de controle (em azul) da resposta do exercício 1.
- ➋ Adicione o caminho de dados para as instruções de desvio incondicional `j` (jumps)
 - O endereço é especificado em palavras
 - Para obter o endereço em bytes, precisamos multiplicar por 4 (deslocar para a esquerda 2x)
 - Teremos como resultado um valor com 28 bits
 - Empréstamos os 4 primeiros bits de PC+4, e os demais 28 bits do endereço do jump (depois do deslocamento), para formar o endereço final
 - Precisamos concatenar o sinal (parte do sinal vêm do PC+4, e parte do endereço do jump). Veja a notação de concatenação abaixo
 - Não precisa implementar os sinais de controle (linhas em azul, pode deixar esses sinais "soltos")

28 Tomamos o sinal completo [0-28] 32 [0-31] Concatenando os sinais, temos 32 bits (de 0 a 31)

32 Tomamos os bits 28 a 31 [31-28]

Anotações

[illegible]

Referências

- D. Patterson; J. Henessy. **Organização e Projeto de Computadores: Interface Hardware/Software**. 5a Edição. Elsevier Brasil, 2017.
- Andrew S. Tanenbaum. **Organização estruturada de computadores**. 5. ed. São Paulo: Pearson, 2007.
- Harris, D. and Harris, S. **Digital Design and Computer Architecture**. 2a ed. 2012.
- courses.missouristate.edu/KenVollmar/mars/

Anotações

Anotações

Anotações
