Arquitetura de Computadores
ACH2055
Aula 05 – *Datapath* de Ciclo Único

Norton Trevisan Roman
(norton@usp.br)

19 de setembro de 2019

Construindo o *Datapath*

Implementando MIPS

- Vamos construir um *datapath* que rode MIPS

Construindo o *Datapath*

Implementando MIPS

- Vamos construir um *datapath* que rode MIPS
- Especificamente, implementaremos:
  - Referências à memória (`lw` e `sw`)
  - Instruções lógico-aritméticas (`add`, `sub`, `AND`, `OR` e `slt`)
  - Desvios (`beq` e `j`)

Construindo o *Datapath*

Implementando MIPS

- Cada instrução começará com os mesmos passos:
  - Armazenamos no contador de programa (PC) o endereço de memória contendo a instrução
  - Buscamos a instrução na memória
  - Lemos campos específicos da instrução, para determinar que registradores ler
  - Lemos os registradores selecionados

Construindo o *Datapath*

Implementando MIPS

- Cada instrução começará com os mesmos passos:
  - Armazenamos no contador de programa (PC) o endereço de memória contendo a instrução
  - Buscamos a instrução na memória
  - Lemos campos específicos da instrução, para determinar que registradores ler
  - Lemos os registradores selecionados
- Após esses passos, cada instrução segue seu caminho

Construindo o *Datapath*

Implementando MIPS

- Para simplificar, faremos com que cada instrução rode em um único ciclo de *clock*
  - Começando em uma borda do *clock* e finalizando na borda seguinte

Construindo o *Datapath*

Implementando MIPS

- Para simplificar, faremos com que cada instrução rode em um único ciclo de *clock*
  - Começando em uma borda do *clock* e finalizando na borda seguinte
- Precisamos então definir uma metodologia de temporização
  - Definindo assim quando os sinais podem ser lidos e quando podem ser escritos

Construindo o *Datapath*

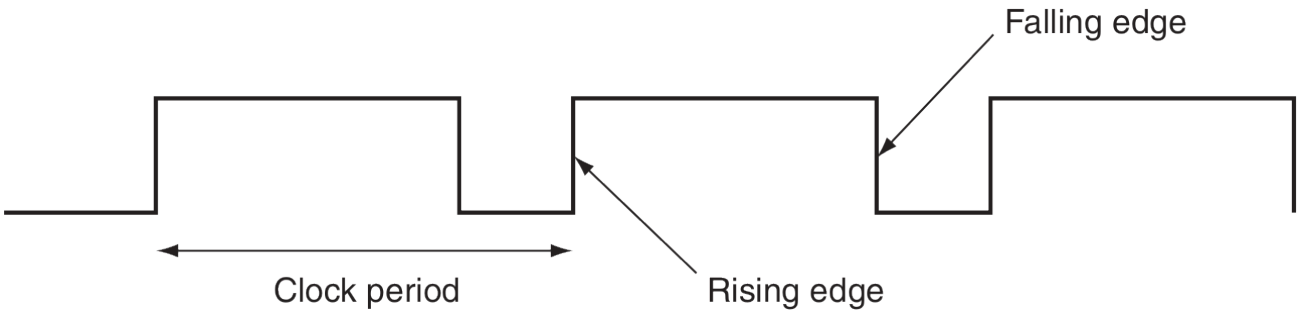
Metodologia de Temporização

- Vamos assumir um modelo ativado pela borda do *clock*
  - Valores são atualizados apenas nas bordas do sinal $\rightarrow$ a transição rápida de voltagem alta para baixa ou vice-versa
  - No nosso caso, na borda crescente

Construindo o *Datapath*

Metodologia de Temporização

- Vamos assumir um modelo ativado pela borda do *clock*
  - Valores são atualizados apenas nas bordas do sinal $\rightarrow$ a transição rápida de voltagem alta para baixa ou vice-versa
  - No nosso caso, na borda crescente

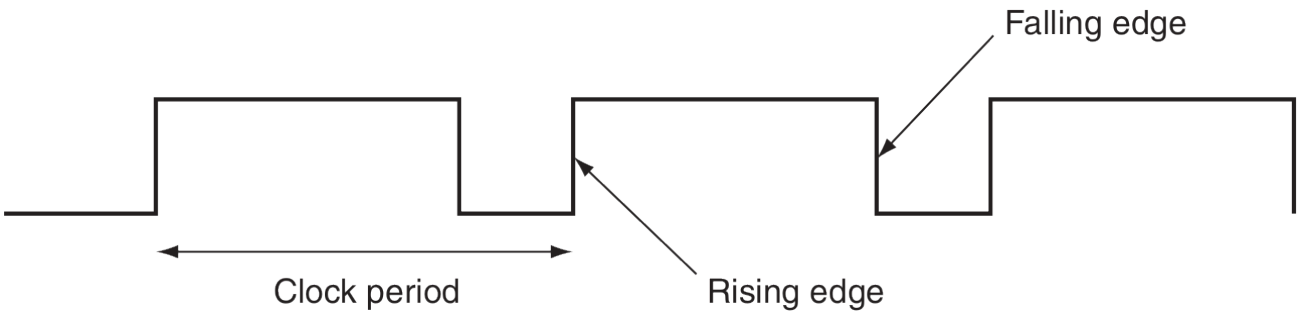


Fonte: [1]

Construindo o *Datapath*

Metodologia de Temporização

- As mudanças podem ocorrer tanto na borda ascendente quanto descendente
  - Dependerá do projeto
  - O conteúdo somente será atualizado na borda ativa do *clock*



Fonte: [1]

Construindo o *Datapath*

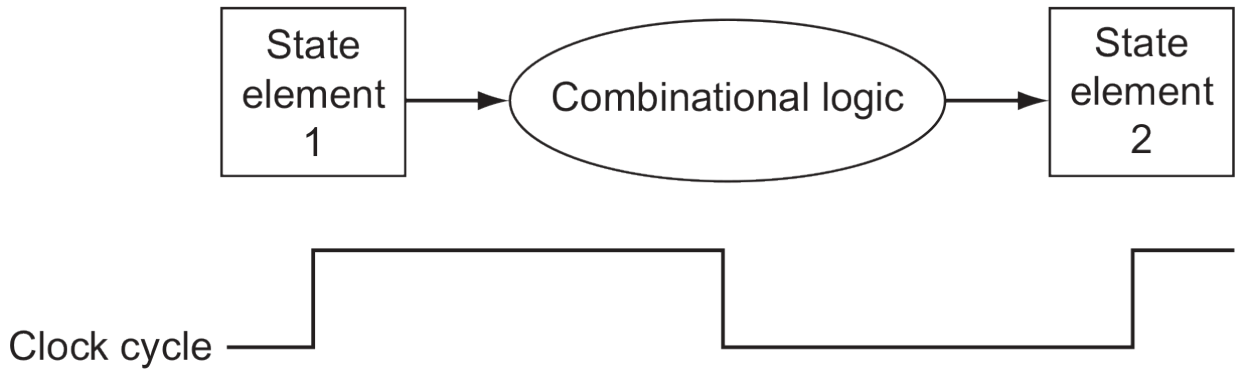
Metodologia de Temporização

- Todo bloco lógico recebe seus dados de um elemento de memória
  - Também conhecidos como **elementos de estado**, por conterem um estado

Construindo o *Datapath*

Metodologia de Temporização

- Todo bloco lógico recebe seus dados de um elemento de memória
  - Também conhecidos como **elementos de estado**, por conterem um estado
- Escrevendo sua saída em outro desses elementos

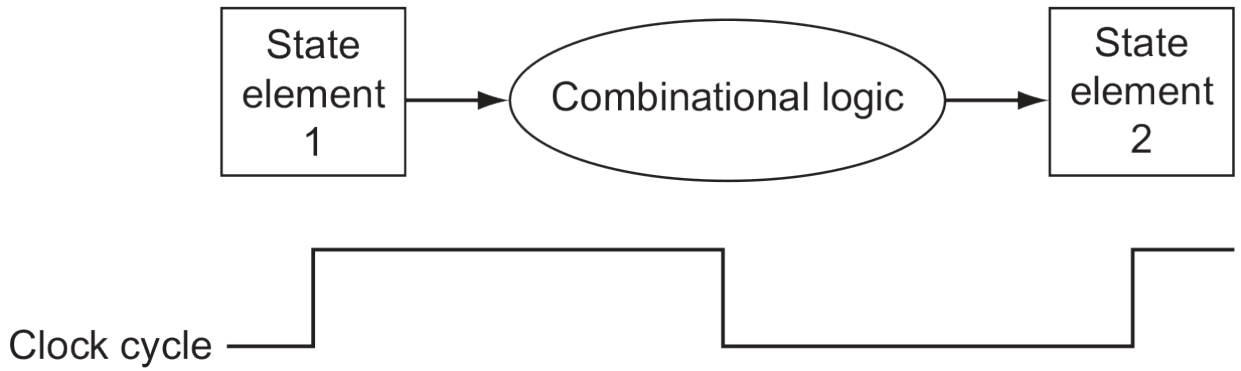


Fonte: [1]

Construindo o *Datapath*

Metodologia de Temporização

- Sua entrada corresponde então aos valores escritos em um ciclo de *clock* anterior
  - Enquanto que suas saídas poderão ser usadas no ciclo seguinte

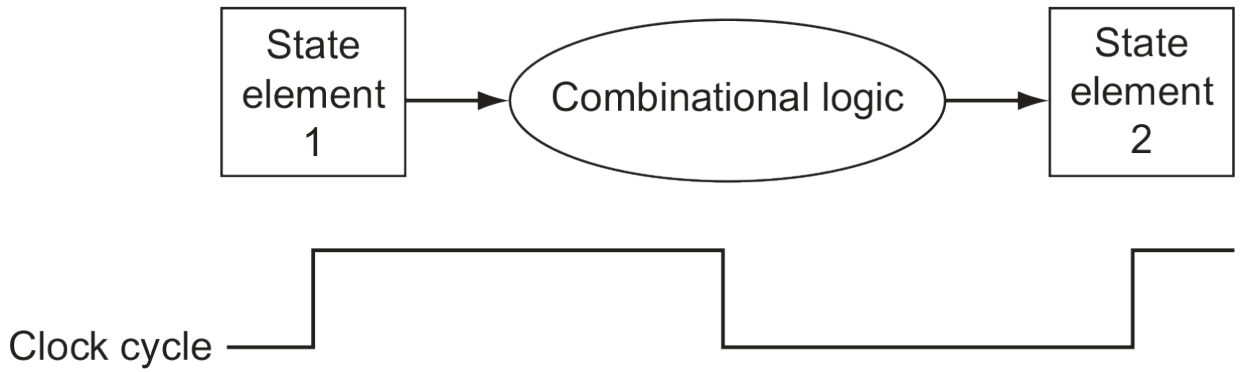


Fonte: [1]

Construindo o *Datapath*

Metodologia de Temporização

- Todos os sinais devem se propagar do elemento 1, através do bloco lógico, para o elemento 2 no tempo de um período do *clock*
  - O tempo necessário para os sinais chegarem ao elemento 2 define então o comprimento do ciclo do *clock*

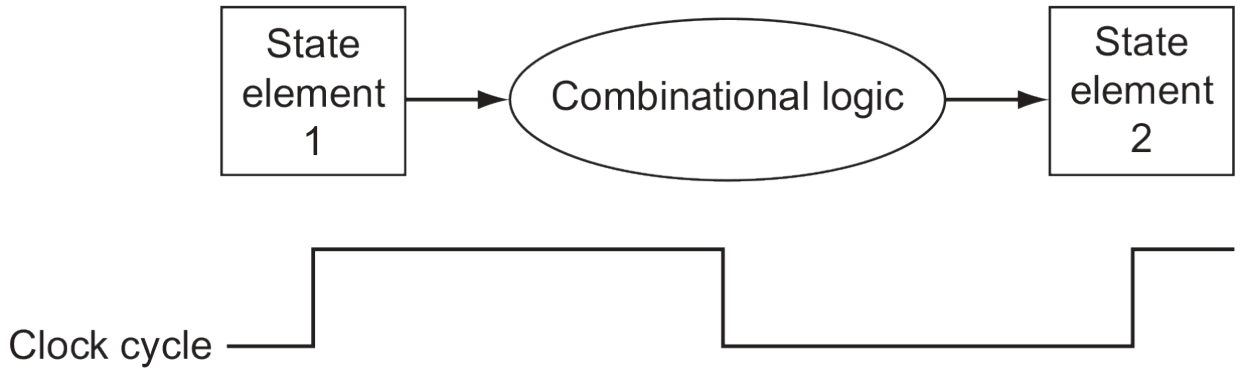


Fonte: [1]

Construindo o *Datapath*

Metodologia de Temporização

- Mais do que isso, os sinais escritos em um elemento devem ser válidos quando a borda ativa ocorrer
  - Um sinal é **válido** se for estável (seu valor não mudará até nova mudança na entrada)

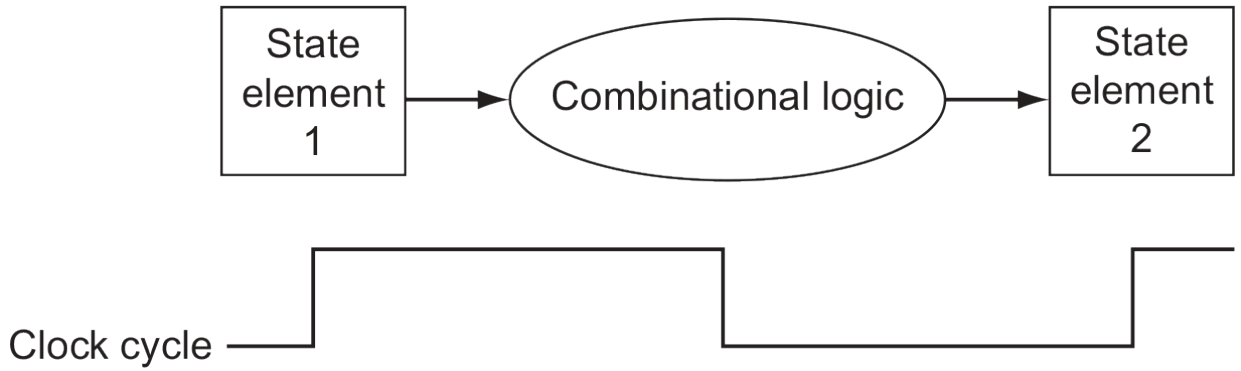


Fonte: [1]

Construindo o *Datapath*

Metodologia de Temporização

- Mais do que isso, os sinais escritos em um elemento devem ser válidos quando a borda ativa ocorrer
  - Um sinal é **válido** se for estável (seu valor não mudará até nova mudança na entrada)
  - Fornecendo assim entradas válidas ao bloco lógico seguinte

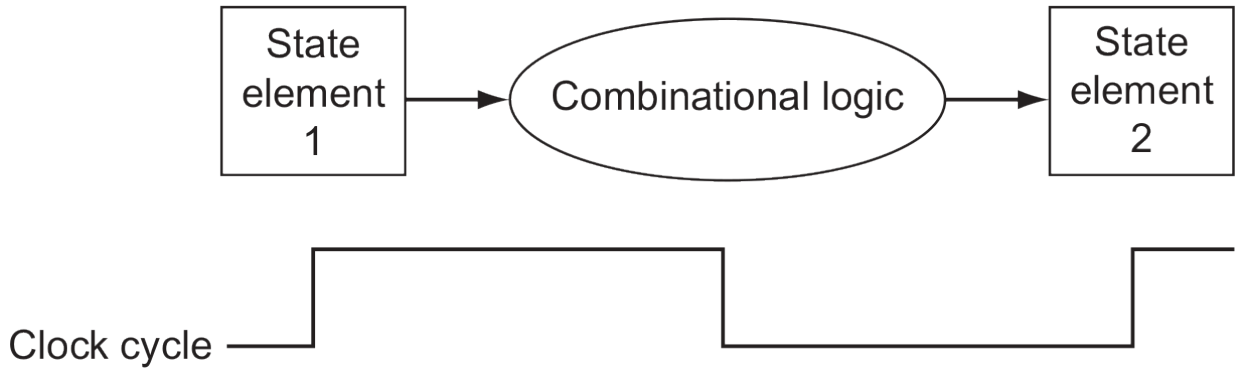


Fonte: [1]

Construindo o *Datapath*

Metodologia de Temporização

- Isso exige que o período do *clock* também seja longo o suficiente para que os sinais no bloco lógico se estabilizem
  - Para que possam ser então armazenados nos elementos com estado

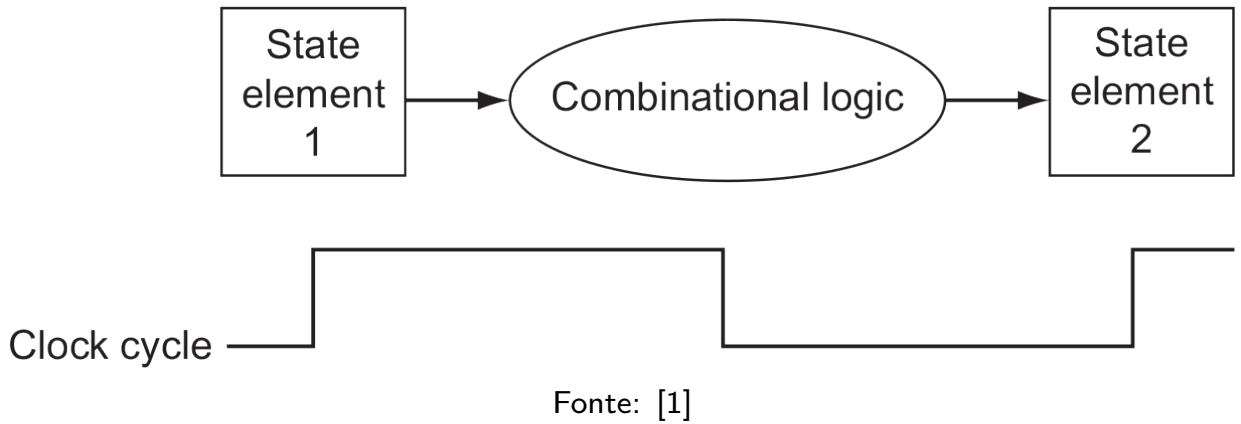


Fonte: [1]

Construindo o *Datapath*

Metodologia de Temporização

- Temos então um limite inferior para o período do *clock*
  - Deve ser longo o suficiente para que todas as entradas aos elementos com estado sejam válidas

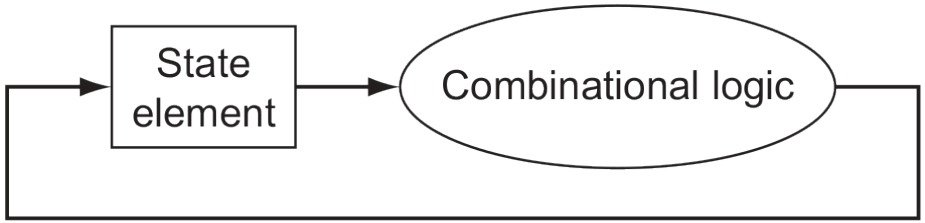


Fonte: [1]

Construindo o *Datapath*

Metodologia de Temporização

- Com isso, podemos ler um registrador, passar seu conteúdo por um bloco lógico, e escrever no mesmo registrador
  - Tudo no mesmo ciclo de *clock*
  - Isso porque as mudanças ocorrem apenas na borda do sinal



Fonte: [1]

Construindo o *Datapath*

Buscando uma Instrução

- Agora que temos a metodologia de temporização, vamos ao *datapath*

Construindo o *Datapath*

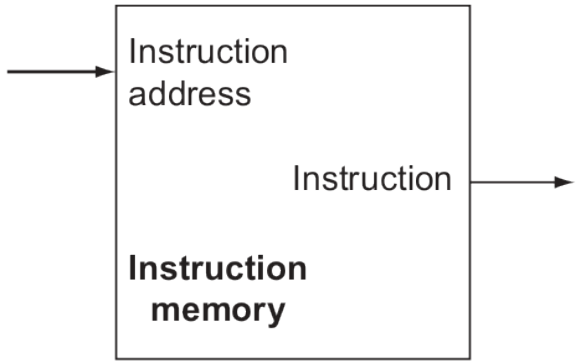
Buscando uma Instrução

- Agora que temos a metodologia de temporização, vamos ao *datapath*
- Antes de mais nada, precisamos armazenar instruções e dados
  - Precisamos de uma memória

Construindo o *Datapath*

Buscando uma Instrução

- Agora que temos a metodologia de temporização, vamos ao *datapath*
- Antes de mais nada, precisamos armazenar instruções e dados
  - Precisamos de uma memória

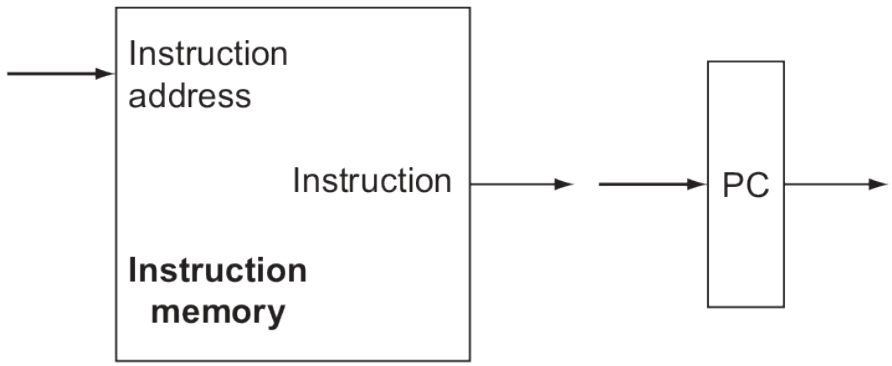


Fonte: [1]

Construindo o *Datapath*

Buscando uma Instrução

- Agora que temos a metodologia de temporização, vamos ao *datapath*
- Antes de mais nada, precisamos armazenar instruções e dados
  - Precisamos de uma memória



Fonte: [1]

- Além do contador de programa, para armazenar o endereço da instrução atual

Construindo o *Datapath*

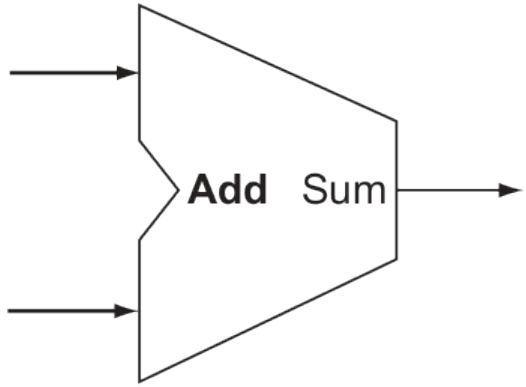
Buscando uma Instrução

- Precisamos também incrementar o PC em 4
  - O endereço da próxima instrução

Construindo o *Datapath*

Buscando uma Instrução

- Precisamos também incrementar o PC em 4
  - O endereço da próxima instrução
  - Precisamos de um adicionador

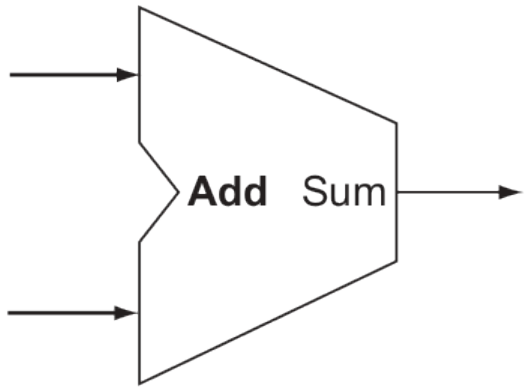


Fonte: [1]

Construindo o *Datapath*

Buscando uma Instrução

- Precisamos também incrementar o PC em 4
  - O endereço da próxima instrução
  - Precisamos de um adicionador

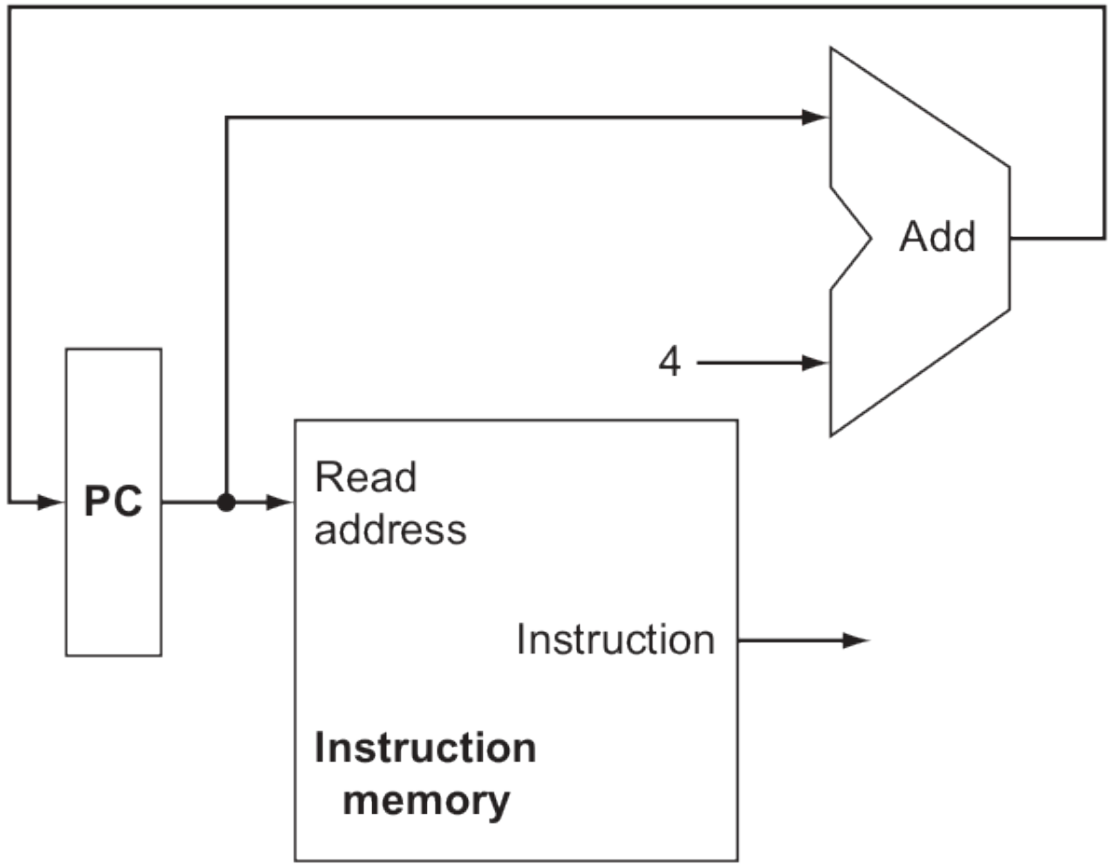


Fonte: [1]

- Para executar uma instrução, precisamos buscá-la na memória
  - E então incrementar o PC em 4, preparando assim para a execução da próxima instrução

Construindo o *Datapath*

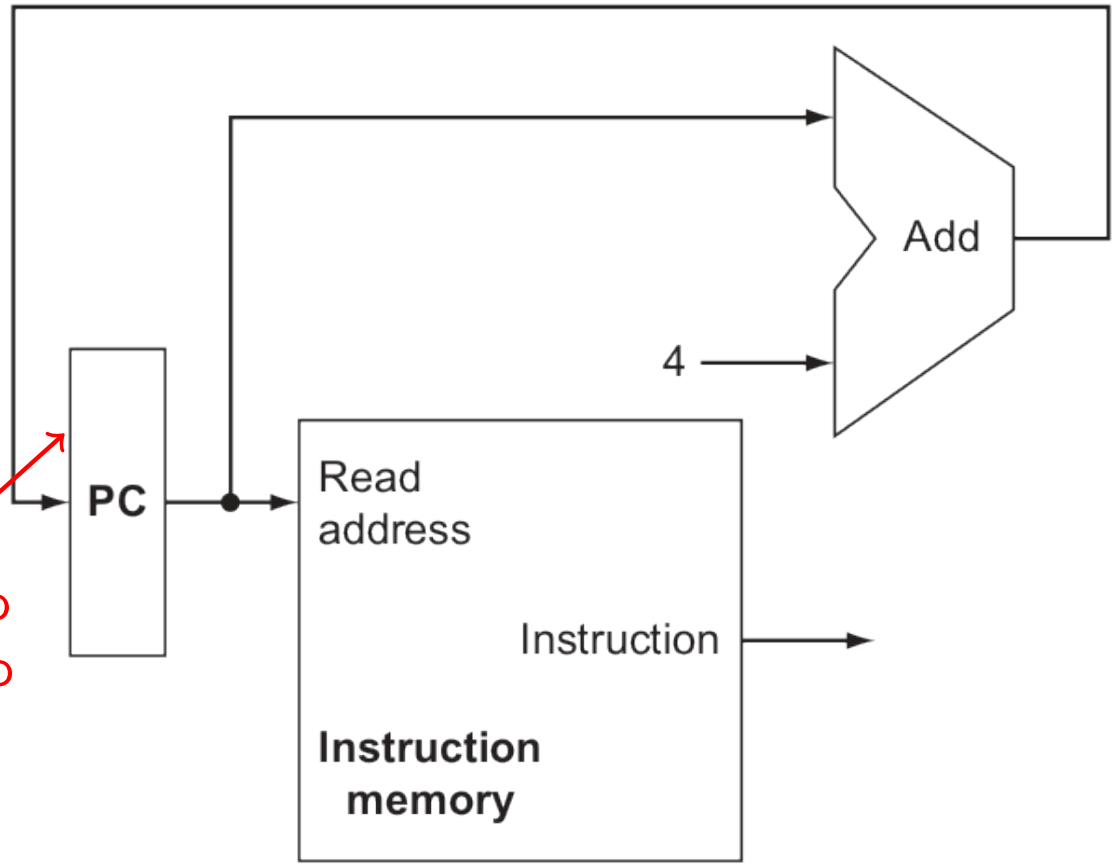
Buscando uma Instrução



Fonte: [1]

Construindo o *Datapath*

Buscando uma Instrução

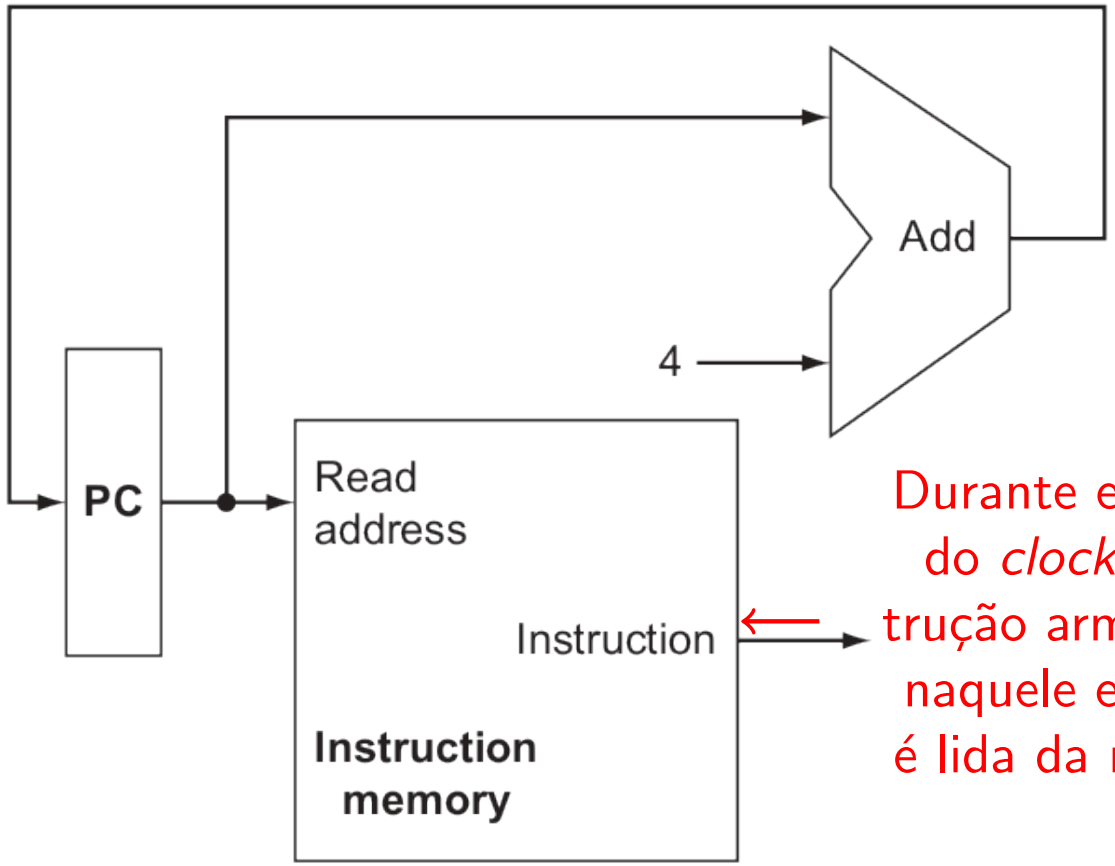


Na borda do *clock*, o endereço armazenado no PC é usado para acessar a memória

Fonte: [1]

Construindo o *Datapath*

Buscando uma Instrução

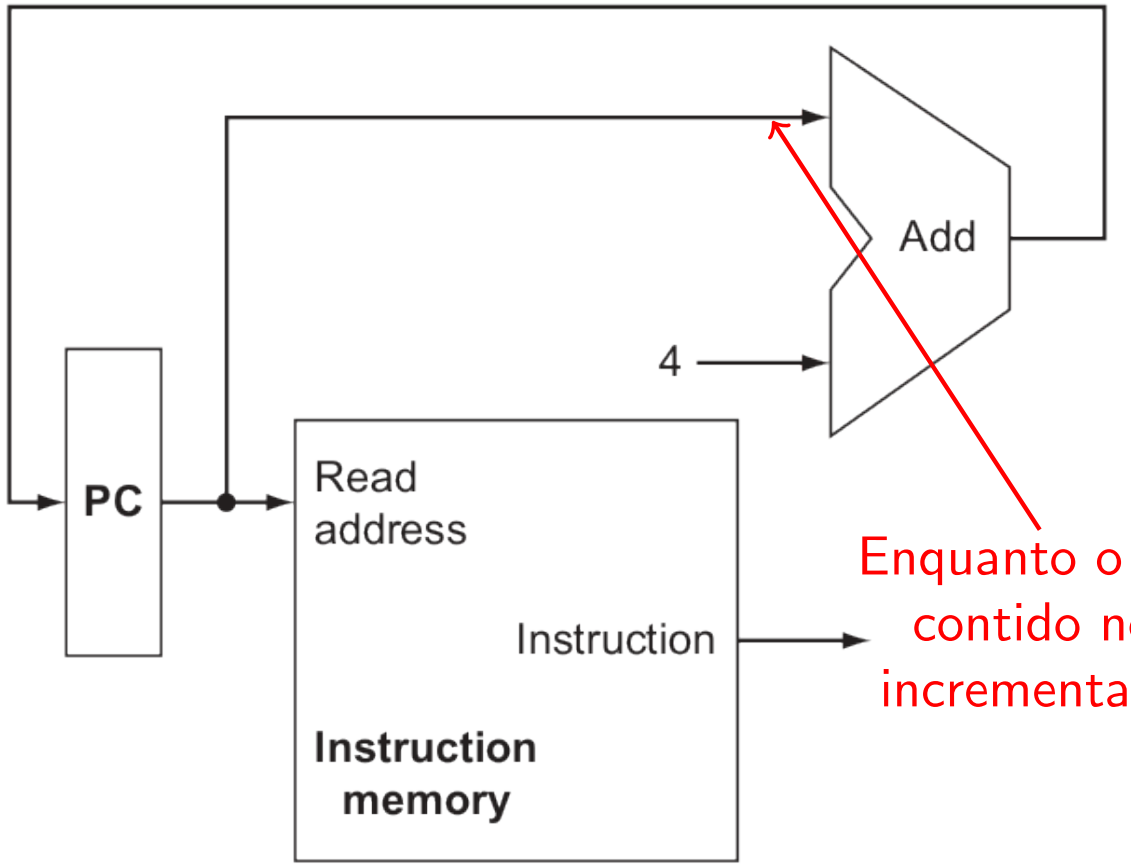


Durante esse ciclo do *clock*, a instrução armazenada naquele endereço é lida da memória

Fonte: [1]

Construindo o *Datapath*

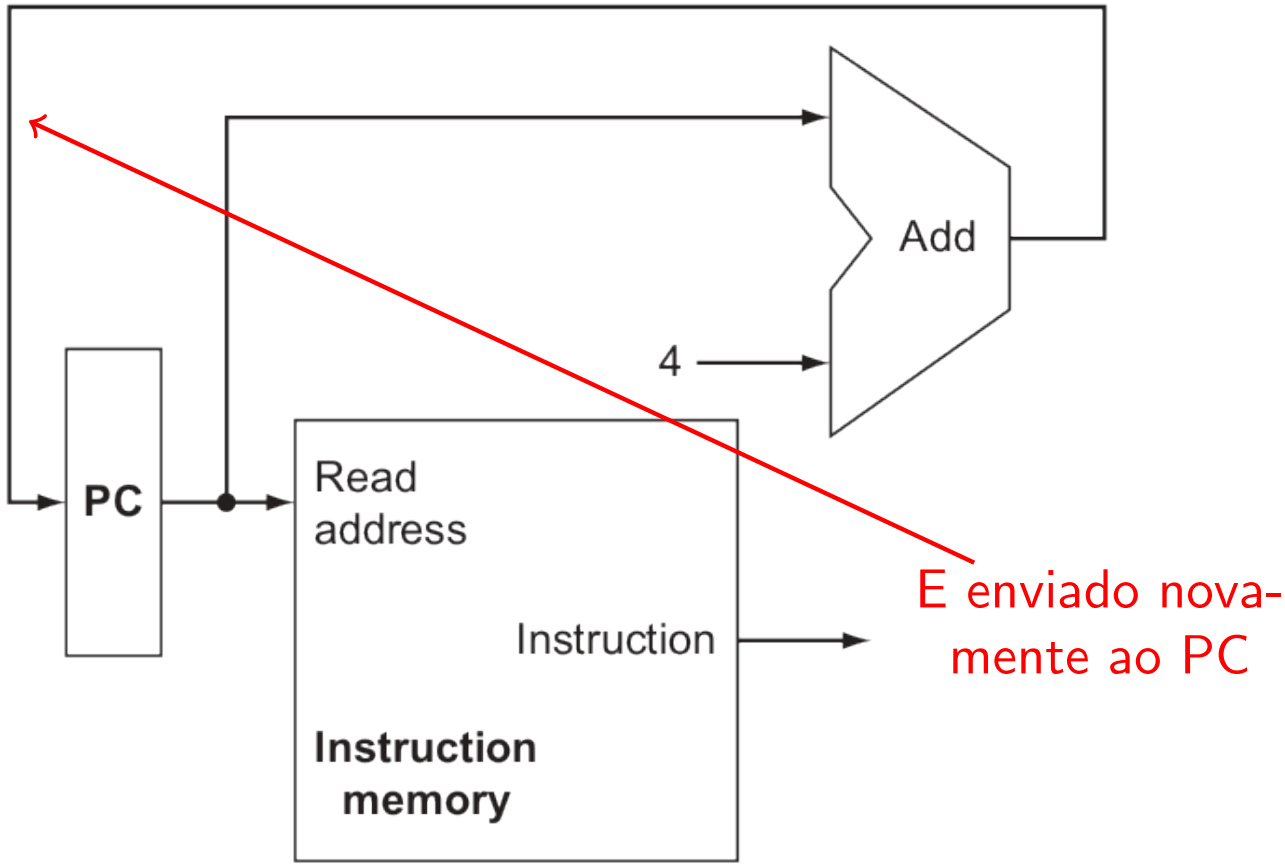
Buscando uma Instrução



Fonte: [1]

Construindo o *Datapath*

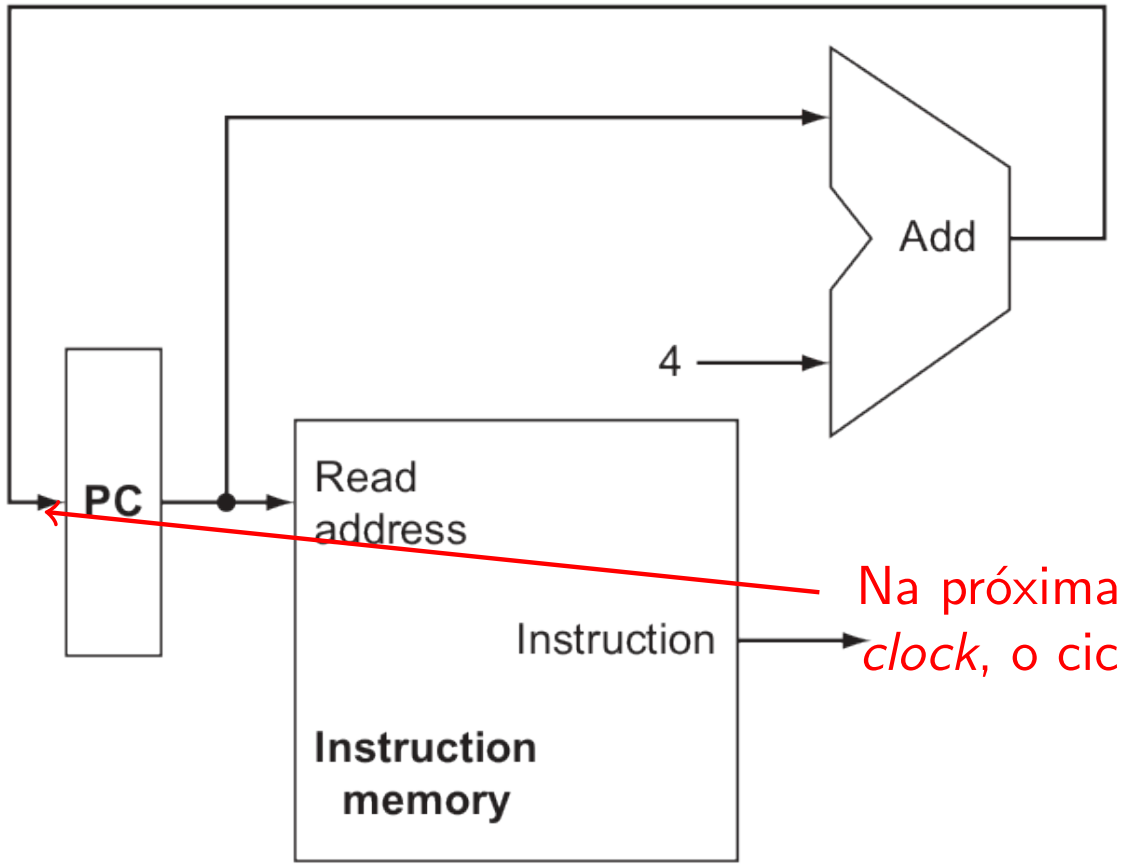
Buscando uma Instrução



Fonte: [1]

Construindo o *Datapath*

Buscando uma Instrução



Fonte: [1]

Construindo o *Datapath*

Implementando instruções do tipo R

Name	Fields						Comments
Field size	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	All MIPS instructions are 32 bits long
R-format	op	rs	rt	rd	shamt	funct	Arithmetic instruction format
I-format	op	rs	rt	address/immediate			Transfer, branch, imm. format
J-format	op	target address					Jump instruction format

Fonte: [1]

Construindo o *Datapath*

Implementando instruções do tipo R

Name	Fields						Comments
Field size	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	All MIPS instructions are 32 bits long
R-format	op	rs	rt	rd	shamt	funct	Arithmetic instruction format
I-format	op	rs	rt	address/immediate			Transfer, branch, imm. format
J-format	op	target address					Jump instruction format

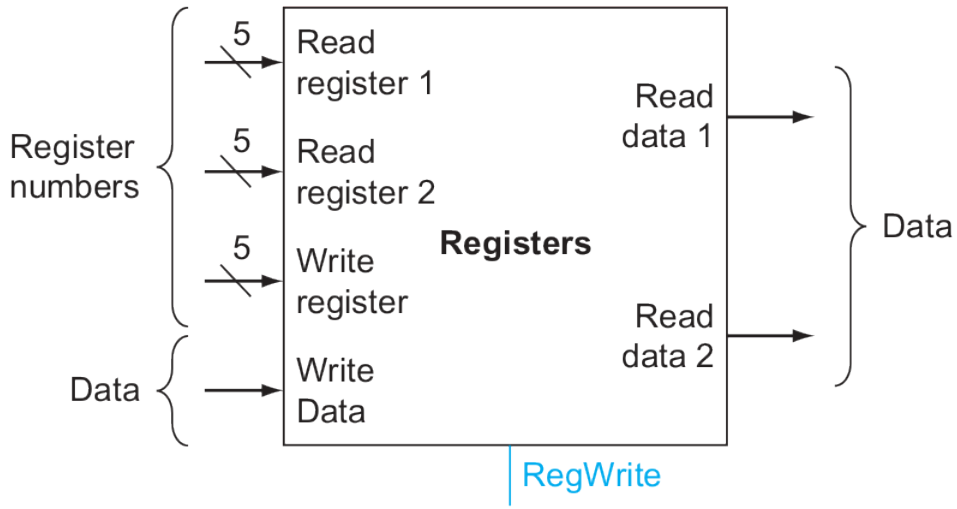
Fonte: [1]

- Todas as instruções do tipo R:
  - Leem 2 registradores (ou 1 registrador e o *shift amount*)
  - Usam seus valores para executar uma operação na ALU
  - Escrevem o resultado em um registrador

Construindo o *Datapath*

Implementando instruções do tipo R

- Os registradores são armazenados em um **Arquivo de registradores**
  - Conjunto de registradores que podem ser lidos e escritos a partir de um número de registro



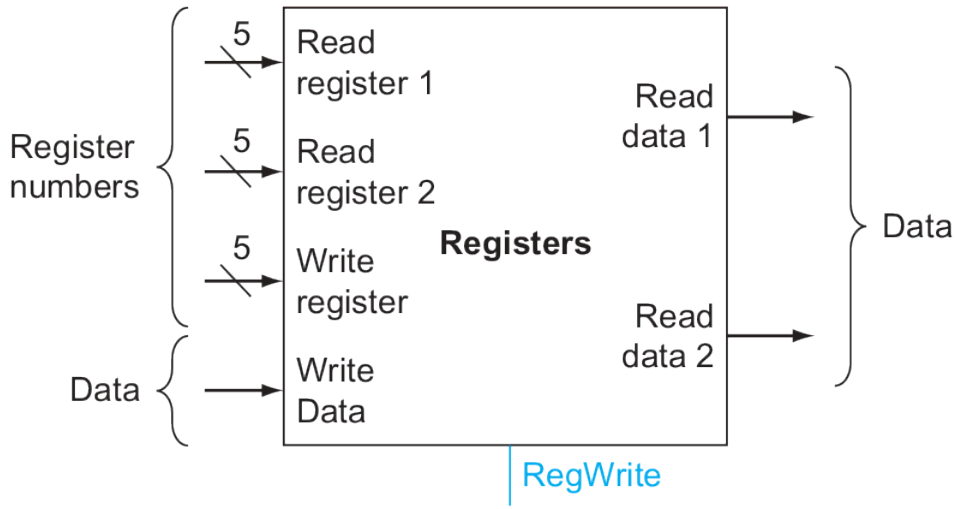
Fonte: [1]

Construindo o *Datapath*

Implementando instruções do tipo R

- Os registradores são armazenados em um **Arquivo de registradores**
  - Conjunto de registradores que podem ser lidos e escritos a partir de um número de registro

Arquivo com duas portas de leitura e uma de escrita



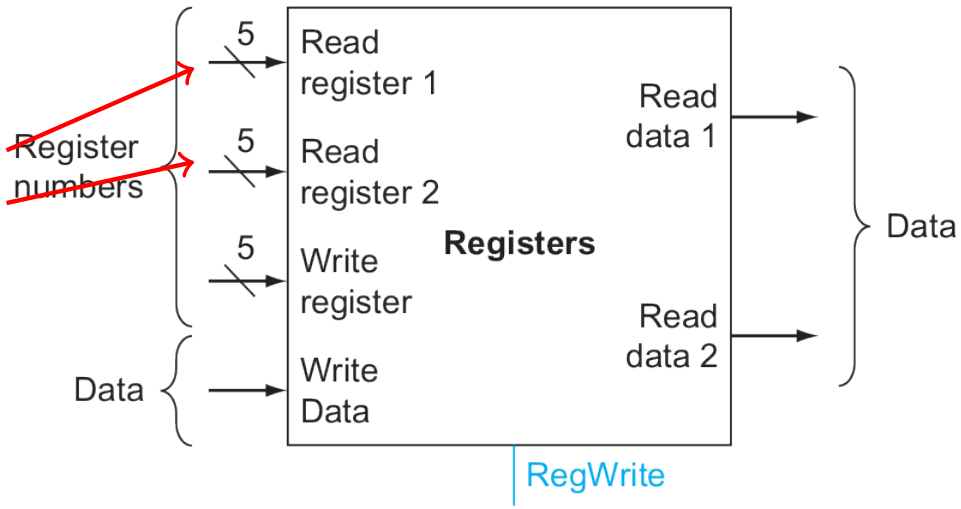
Fonte: [1]

Construindo o *Datapath*

Implementando instruções do tipo R

- Os registradores são armazenados em um **Arquivo de registradores**
  - Conjunto de registradores que podem ser lidos e escritos a partir de um número de registro

Para leitura, fornecemos o número do registrador (até 2 registradores diferentes)



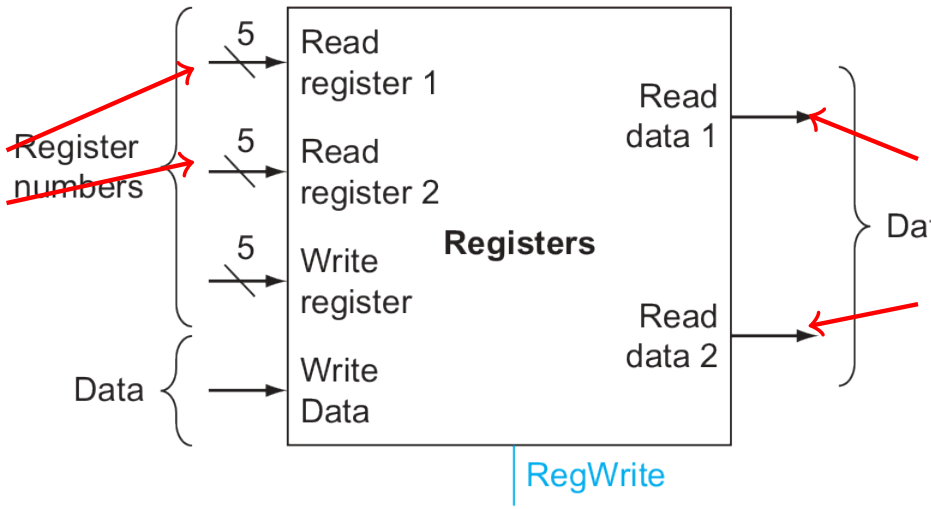
Fonte: [1]

Construindo o *Datapath*

Implementando instruções do tipo R

- Os registradores são armazenados em um **Arquivo de registradores**
  - Conjunto de registradores que podem ser lidos e escritos a partir de um número de registro

Para leitura, fornecemos o número do registrador (até 2 registradores diferentes)



E o conteúdo desse(s) registrador(es) é retornado

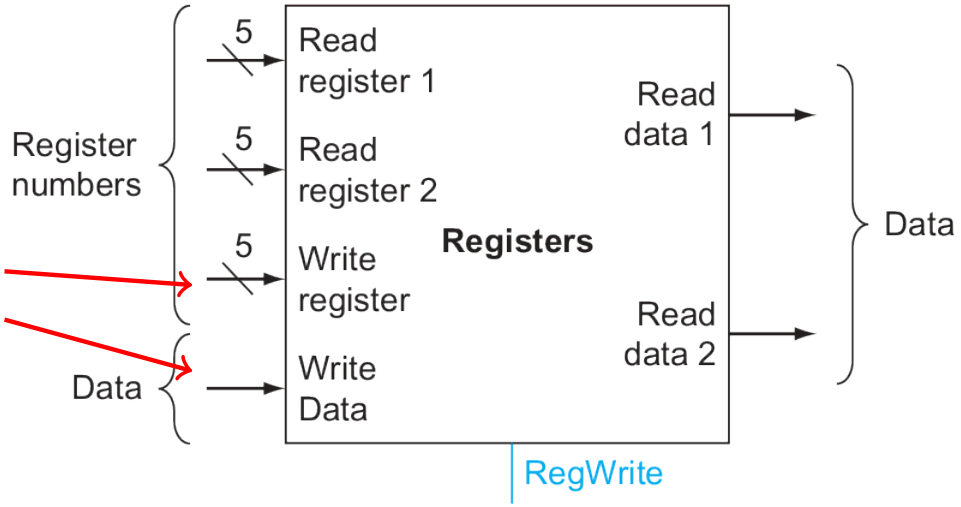
Fonte: [1]

Construindo o *Datapath*

Implementando instruções do tipo R

- Os registradores são armazenados em um **Arquivo de registradores**
  - Conjunto de registradores que podem ser lidos e escritos a partir de um número de registro

Para escrita, fornecemos o número do registrador e o dado



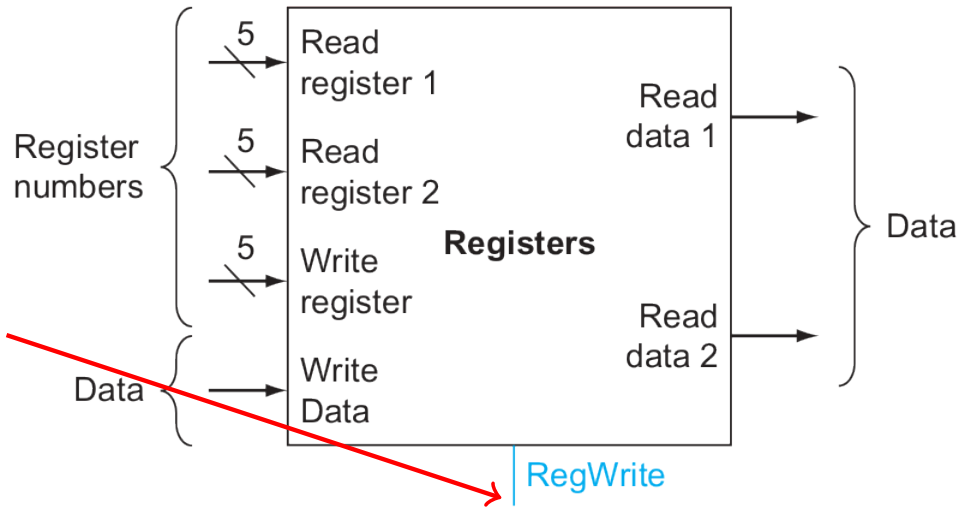
Fonte: [1]

Construindo o *Datapath*

Implementando instruções do tipo R

- Os registradores são armazenados em um **Arquivo de registradores**
  - Conjunto de registradores que podem ser lidos e escritos a partir de um número de registro

Além de um sinal de *clock* para controlar a escrita no registrador

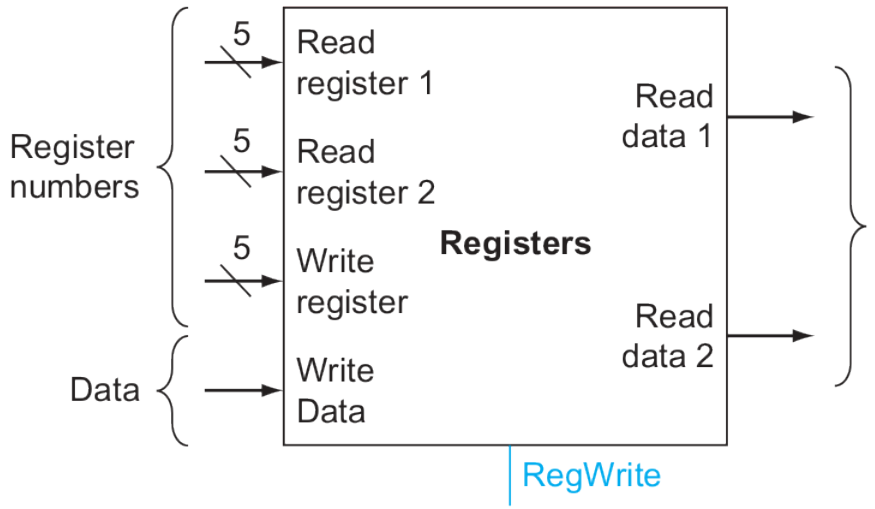


Fonte: [1]

Construindo o *Datapath*

Implementando instruções do tipo R

- Os registradores são armazenados em um **Arquivo de registradores**
  - Conjunto de registradores que podem ser lidos e escritos a partir de um número de registro



O arquivo sempre retorna o conteúdo dos registradores cujos números estão na entrada

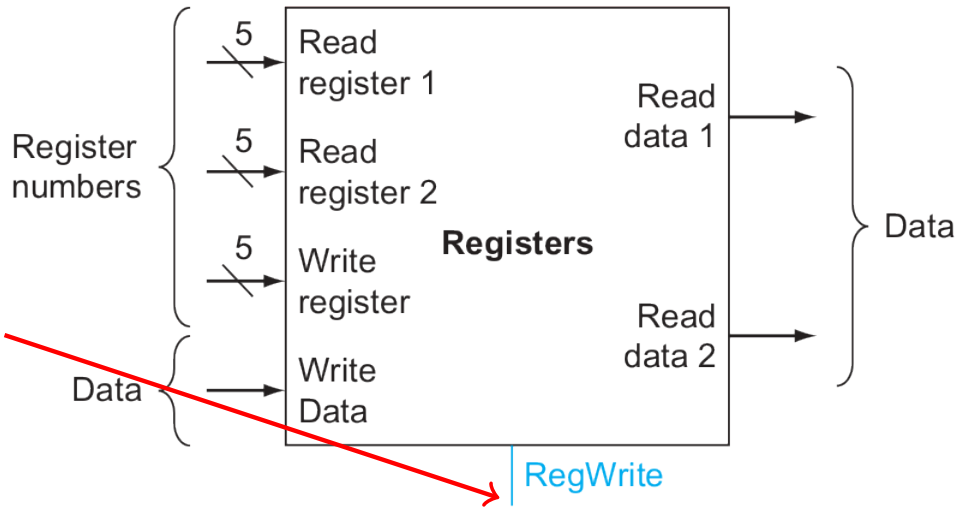
Fonte: [1]

Construindo o *Datapath*

Implementando instruções do tipo R

- Os registradores são armazenados em um **Arquivo de registradores**
  - Conjunto de registradores que podem ser lidos e escritos a partir de um número de registro

Escritas, contudo, são controladas pelo sinal de escrita



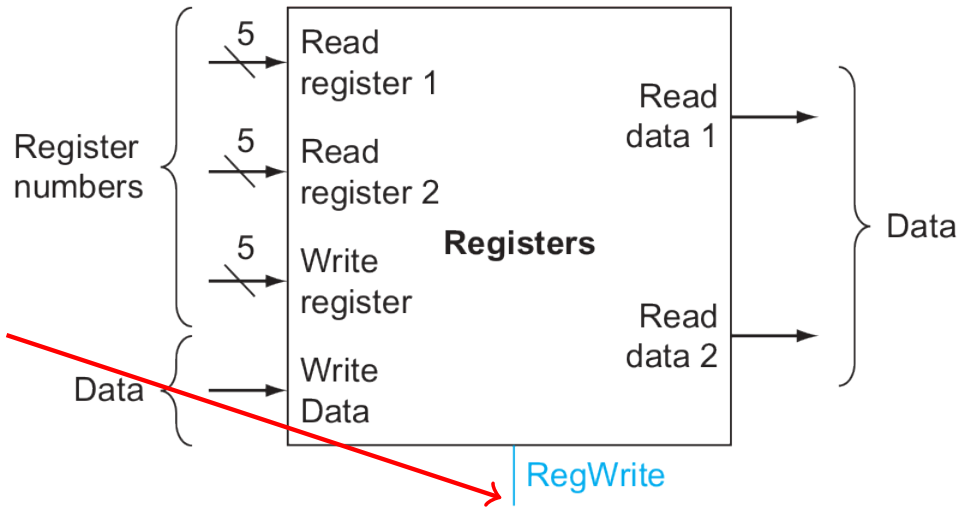
Fonte: [1]

Construindo o *Datapath*

Implementando instruções do tipo R

- Os registradores são armazenados em um **Arquivo de registradores**
  - Conjunto de registradores que podem ser lidos e escritos a partir de um número de registro

Que precisa ser ligado, para que uma escrita ocorra na próxima borda do sinal de clock

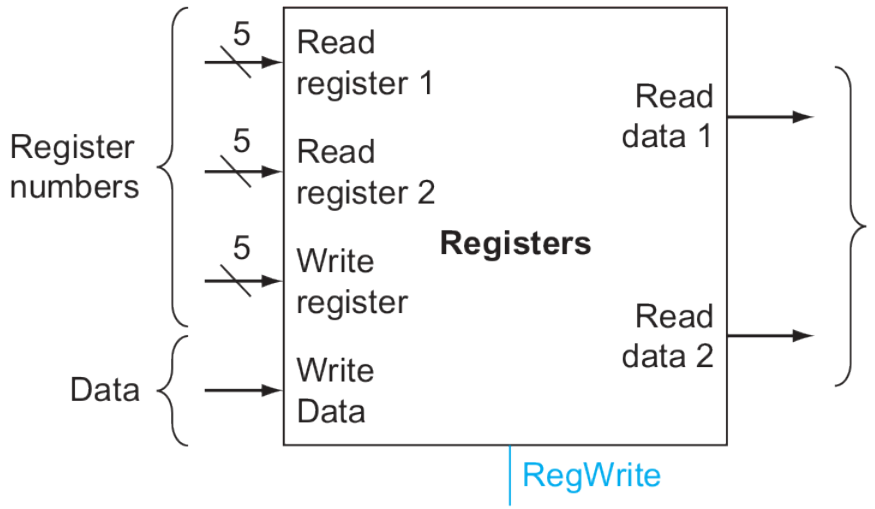


Fonte: [1]

Construindo o *Datapath*

Implementando instruções do tipo R

- Os registradores são armazenados em um **Arquivo de registradores**
  - Conjunto de registradores que podem ser lidos e escritos a partir de um número de registro



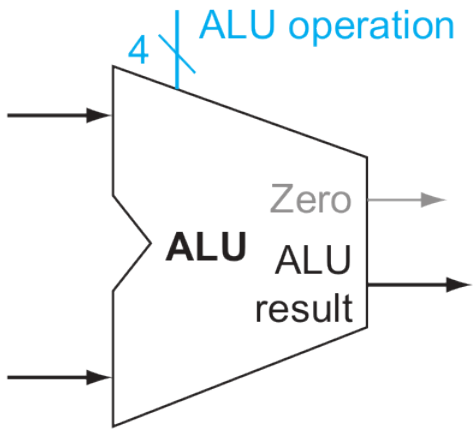
Até essa borda ocorrer, a saída continua sendo o que foi armazenado no ciclo anterior

Fonte: [1]

Construindo o *Datapath*

Implementando instruções do tipo R

- Precisaremos também de uma ALU, para operar nos valores dos registradores

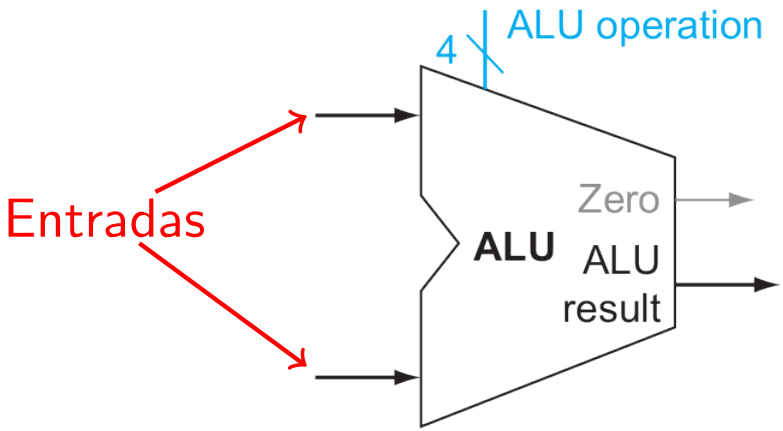


Fonte: [1]

Construindo o *Datapath*

Implementando instruções do tipo R

- Precisaremos também de uma ALU, para operar nos valores dos registradores

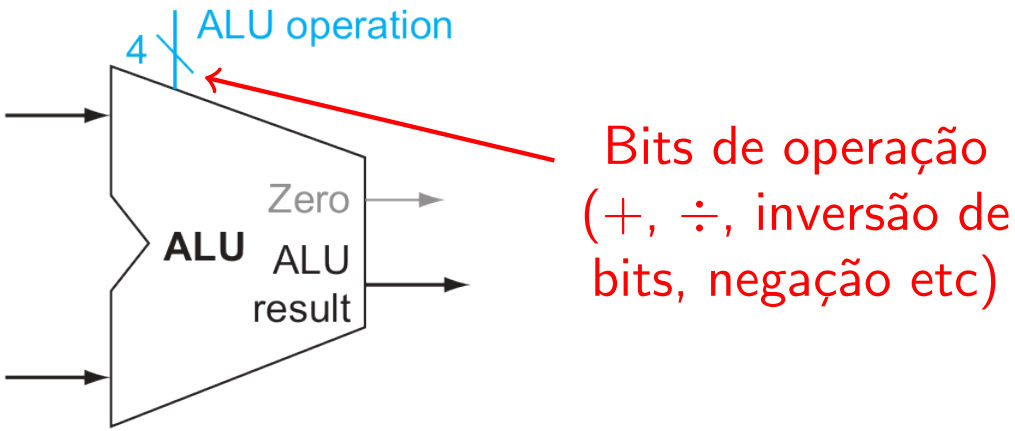


Fonte: [1]

Construindo o *Datapath*

Implementando instruções do tipo R

- Precisaremos também de uma ALU, para operar nos valores dos registradores

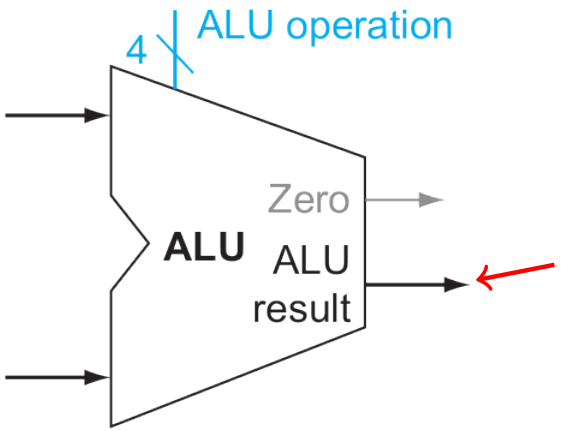


Fonte: [1]

Construindo o *Datapath*

Implementando instruções do tipo R

- Precisaremos também de uma ALU, para operar nos valores dos registradores



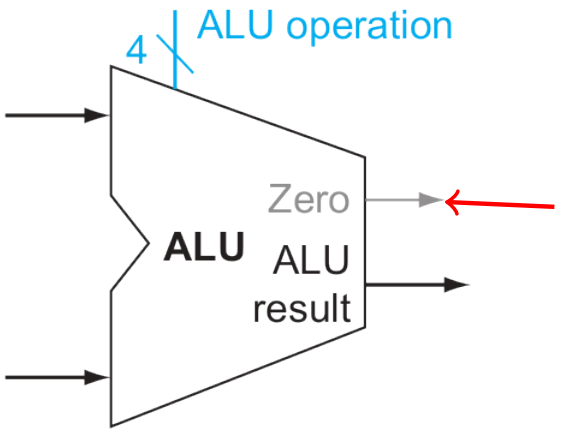
Resultado da operação

Fonte: [1]

Construindo o *Datapath*

Implementando instruções do tipo R

- Precisaremos também de uma ALU, para operar nos valores dos registradores



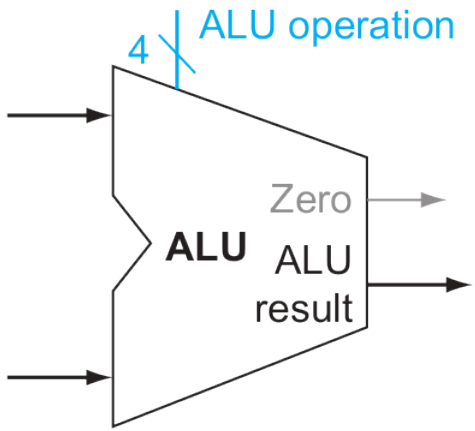
Indica que todos os bits do resultado são zero (útil para implementar *branches*)

Fonte: [1]

Construindo o *Datapath*

Implementando instruções do tipo R

- Precisaremos também de uma ALU, para operar nos valores dos registradores



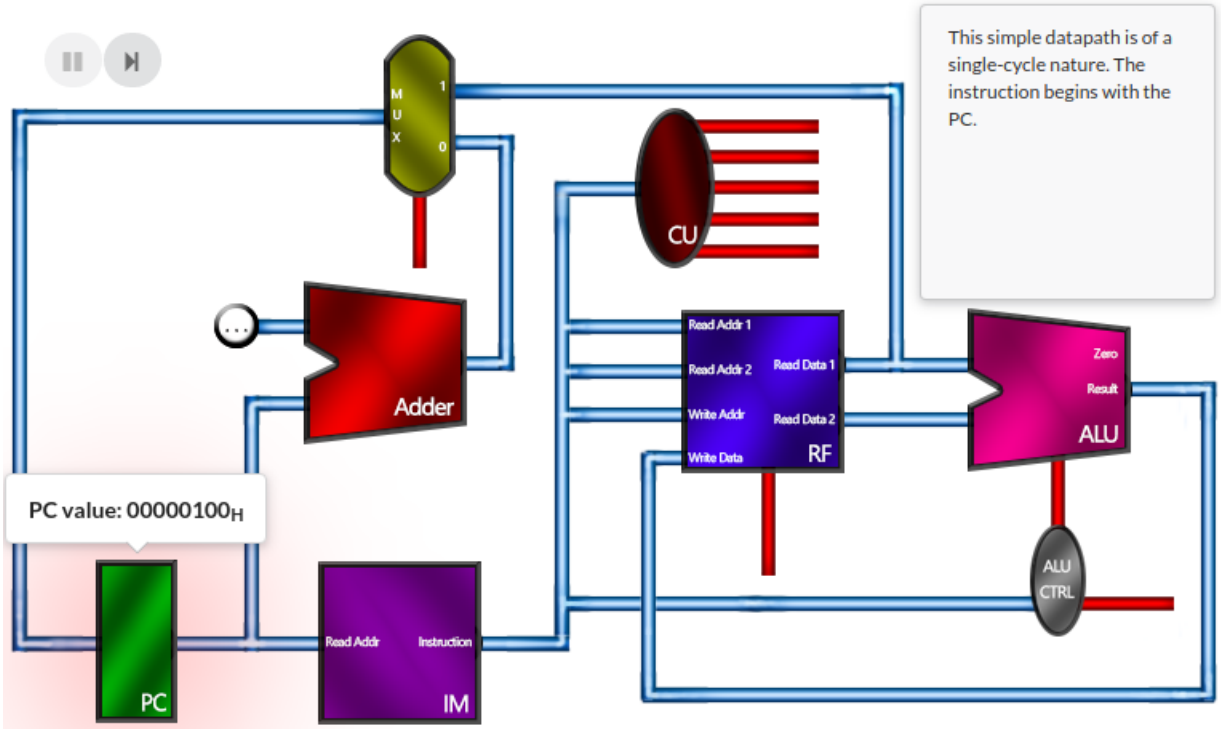
Fonte: [1]

Há mais sinais e saídas, que veremos mais adiante

Construindo o *Datapath*

Implementando instruções do tipo R

```
add $3, $5, $1
```



https://www3.ntu.edu.sg/home/smitha/FYP_Gerald/addInstruction.html

Construindo o *Datapath*

Implementando instruções do tipo I

Name	Fields						Comments
Field size	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	All MIPS instructions are 32 bits long
R-format	op	rs	rt	rd	shamt	funct	Arithmetic instruction format
I-format	op	rs	rt	address/immediate			Transfer, branch, imm. format
J-format	op	target address					Jump instruction format

Fonte: [1]

Construindo o *Datapath*

Implementando instruções do tipo I

Name	Fields						Comments
Field size	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	All MIPS instructions are 32 bits long
R-format	op	rs	rt	rd	shamt	funct	Arithmetic instruction format
I-format	op	rs	rt	address/immediate			Transfer, branch, imm. format
J-format	op	target address					Jump instruction format

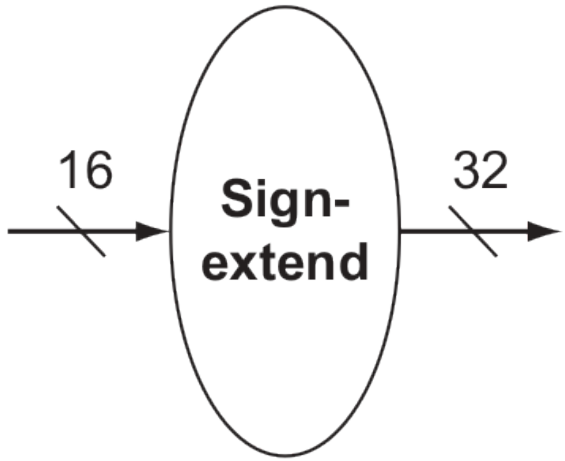
Fonte: [1]

- Vamos implementar `lw` e `sw`
  - `lw $s0, desl($s1)`
  - `sw $s0, desl($s1)`
  - Ambas usam o conjunto de registradores e a ALU, para poderem armazenar dados e calcular endereços

Construindo o *Datapath*

Implementando `lw` e `sw`

- Também precisam de uma unidade para estender o deslocamento de 16 bits para 32 (preservando o sinal)
  - Para que possa ser somado ao endereço no registrador base
  - Essa extensão se dá replicando-se o bit de sinal do dado de origem nos bits de maior ordem do dado de destino

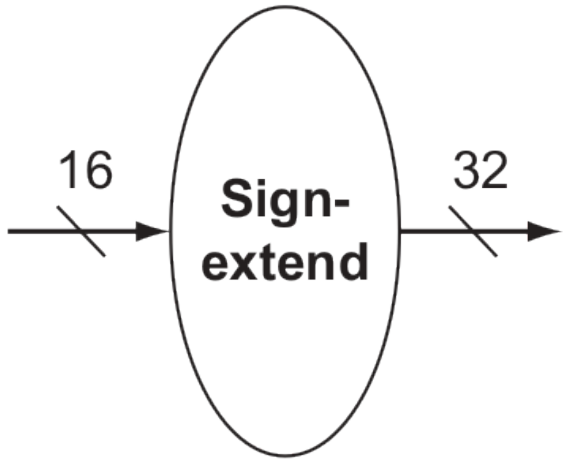


Fonte: [1]

Construindo o *Datapath*

Implementando `lw` e `sw`

- Também precisam de uma unidade para estender o deslocamento de 16 bits para 32 (preservando o sinal)
  - Para que possa ser somado ao endereço no registrador base
  - Essa extensão se dá replicando-se o bit de sinal do dado de origem nos bits de maior ordem do dado de destino
- Além, obviamente, de uma memória



Fonte: [1]

Construindo o *Datapath*

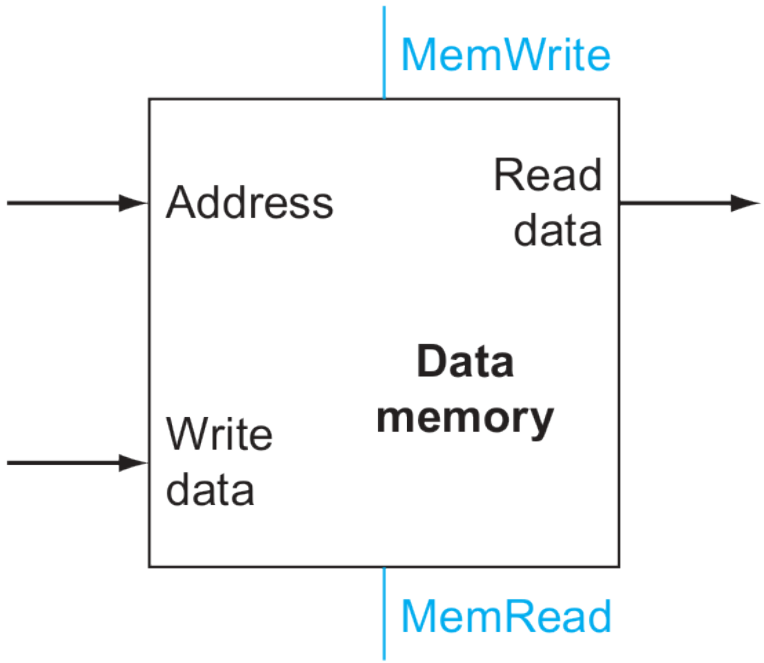
Implementando `lw` e `sw`

- Iremos ler e escrever na memória
  - Precisamos de sinais de controle para leitura e escrita
  - Além do endereço e dado de entrada a ser escrito

Construindo o *Datapath*

Implementando `lw` e `sw`

- Iremos ler e escrever na memória
  - Precisamos de sinais de controle para leitura e escrita
  - Além do endereço e dado de entrada a ser escrito

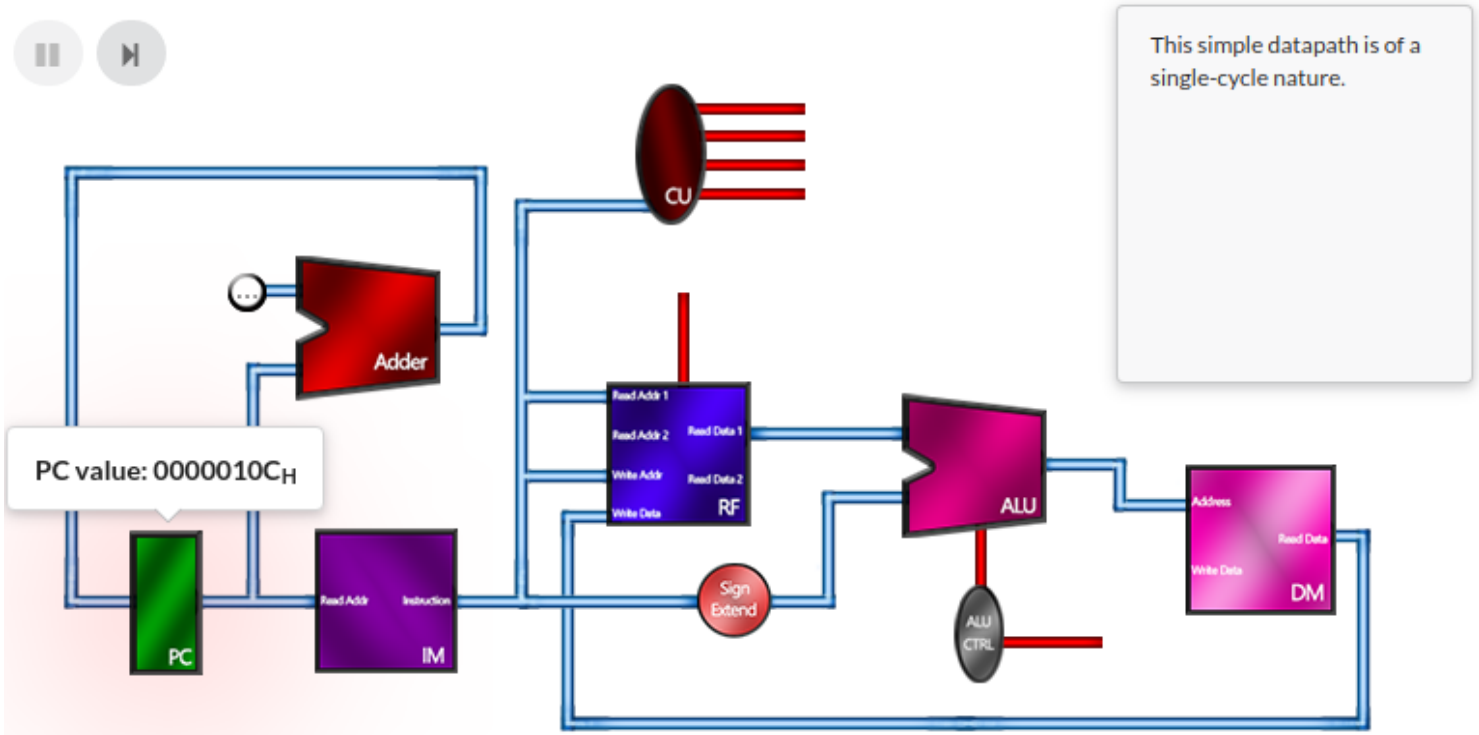


Fonte: [1]

Construindo o *Datapath*

Implementando `lw` e `sw`

```
lw $26, 12($30)
```

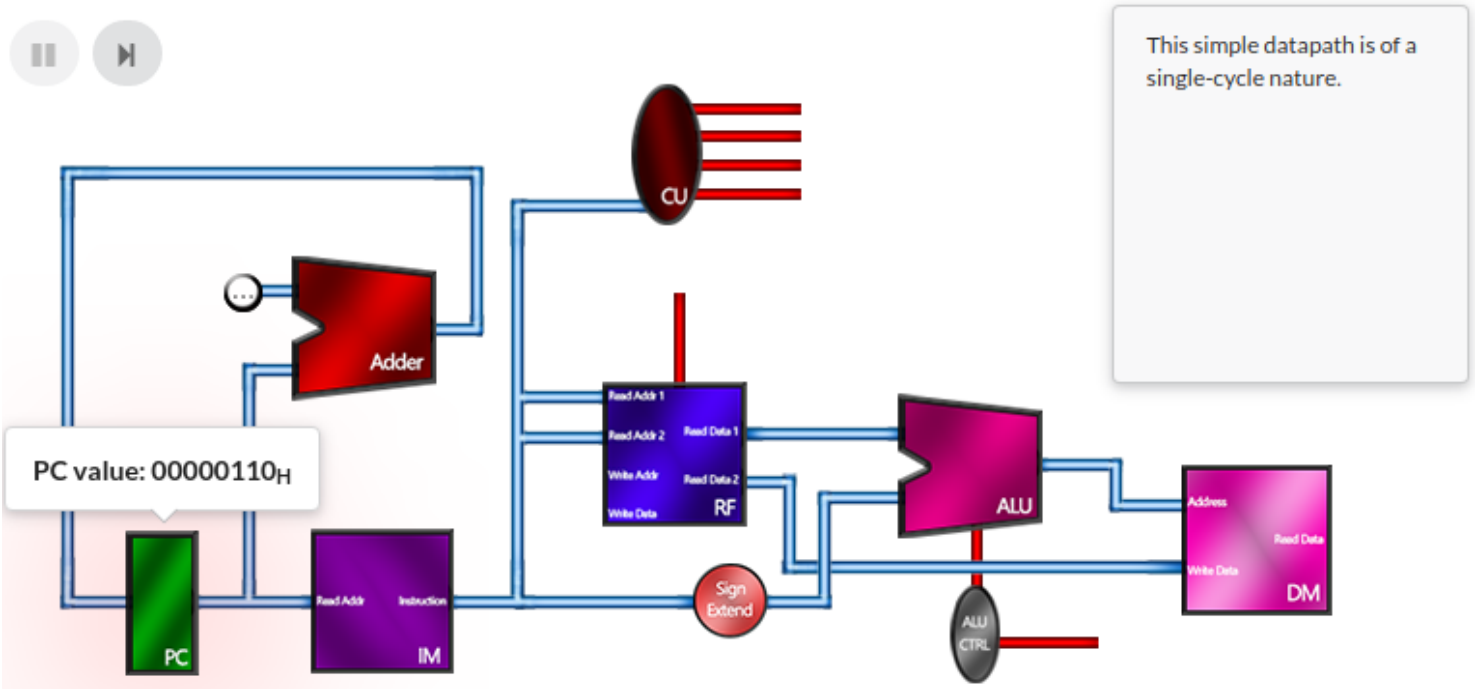


https://www3.ntu.edu.sg/home/smitha/FYP_Gerald/lwInstruction.html

Construindo o *Datapath*

Implementando `lw` e `sw`

```
sw $2, 12($5)
```



https://www3.ntu.edu.sg/home/smitha/FYP_Gerald/swInstruction.html

Construindo o *Datapath*

Implementando beq

- E agora implementemos beq
  - `beq` $\texttt{reg}_1$`,` $\texttt{reg}_2$`, rótulo`

Construindo o *Datapath*

Implementando `beq`

- E agora implementemos `beq`
  - `beq` $\texttt{reg}_1$`,` $\texttt{reg}_2$`,` `rótulo`
- Como calcular o rótulo?

Construindo o *Datapath*

Implementando `beq`

- E agora implementemos `beq`
  - `beq` $\texttt{reg}_1$`,` $\texttt{reg}_2$`, rótulo`
- Como calcular o rótulo?
  - Ocorre que a forma real da instrução é
    `beq` $\texttt{reg}_1$`,` $\texttt{reg}_2$`, deslocamento`
  - Então o rótulo é calculado como um deslocamento (16 bits) a partir do endereço seguinte à instrução `beq`

Construindo o *Datapath*

Implementando `beq`

- `beq` funciona então de modo semelhante a `lw` e `sw`
  - Precisamos estender o deslocamento, para somar ao registrador base (no caso, `PC+4`)

Construindo o *Datapath*

Implementando `beq`

- `beq` funciona então de modo semelhante a `lw` e `sw`
  - Precisamos estender o deslocamento, para somar ao registrador base (no caso, `PC+4`)
- MIPS contudo exige que, antes da extensão, o deslocamento seja deslocado à esquerda em 2 bits
  - Fazendo desse um deslocamento de palavra, não bytes
  - E aumentando assim o alcance do deslocamento por um fator de 4

Construindo o *Datapath*

Implementando `beq`

- Além disso, precisamos também saber qual será a próxima instrução executada

Construindo o *Datapath*

Implementando `beq`

- Além disso, precisamos também saber qual será a próxima instrução executada
  - Se a seguinte ao `beq` (os argumentos eram distintos)
    - Caso em que `PC = PC + 4`

Construindo o *Datapath*

Implementando `beq`

- Além disso, precisamos também saber qual será a próxima instrução executada
  - Se a seguinte ao `beq` (os argumentos eram distintos)
    - Caso em que `PC = PC + 4`
  - Ou a correspondente a endereço_base + deslocamento (os argumentos eram iguais)
    - Caso em que esse novo endereço se torna o novo `PC`

Construindo o *Datapath*

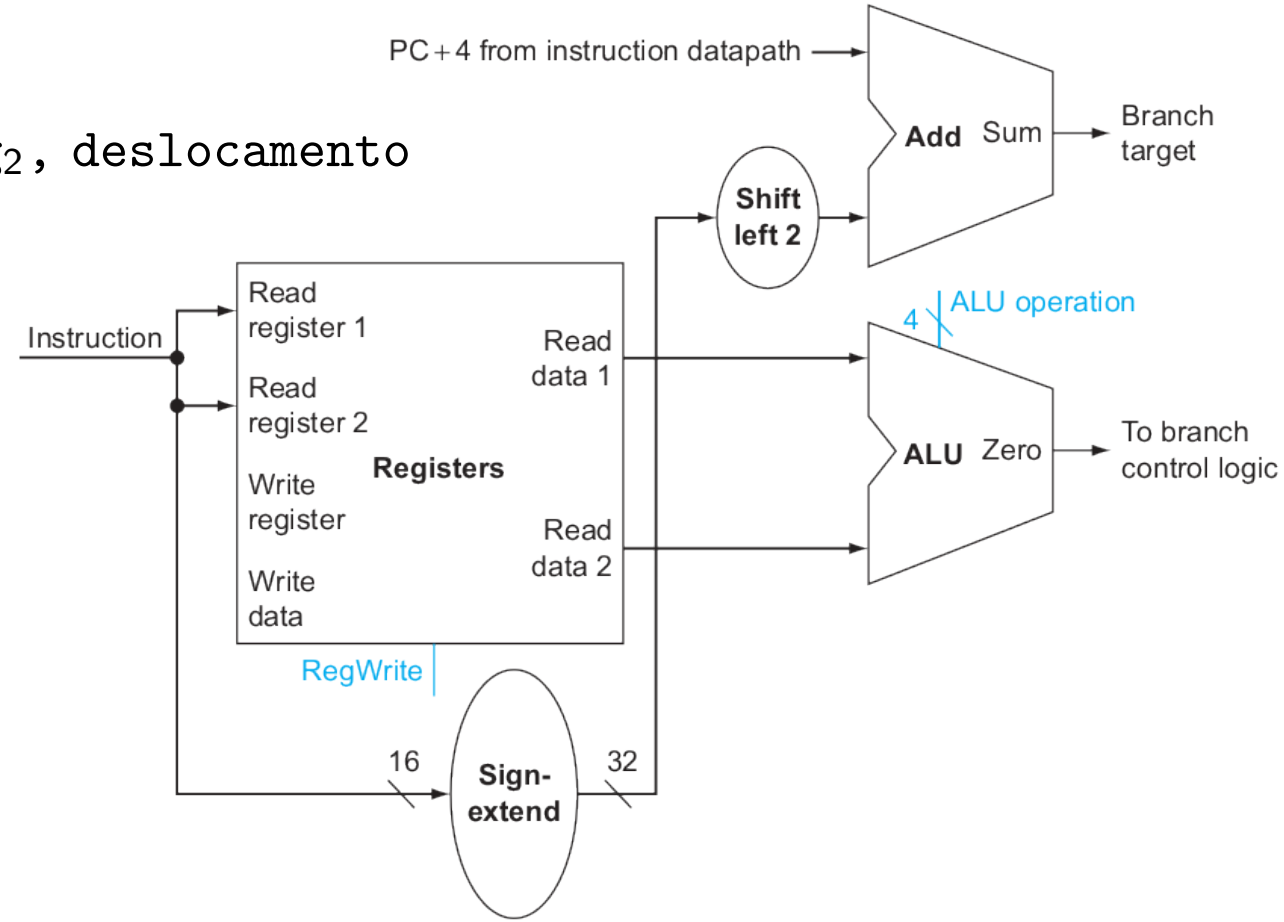
Implementando `beq`

- Além disso, precisamos também saber qual será a próxima instrução executada
  - Se a seguinte ao `beq` (os argumentos eram distintos)
    - Caso em que `PC = PC + 4`
  - Ou a correspondente a endereço_base + deslocamento (os argumentos eram iguais)
    - Caso em que esse novo endereço se torna o novo `PC`
- Temos então 2 operações
  - Cálculo do endereço alvo e comparação dos registradores

Construindo o *Datapath*

Implementando beq

`beq` $\texttt{reg}_1$`,` $\texttt{reg}_2$`, deslocamento`



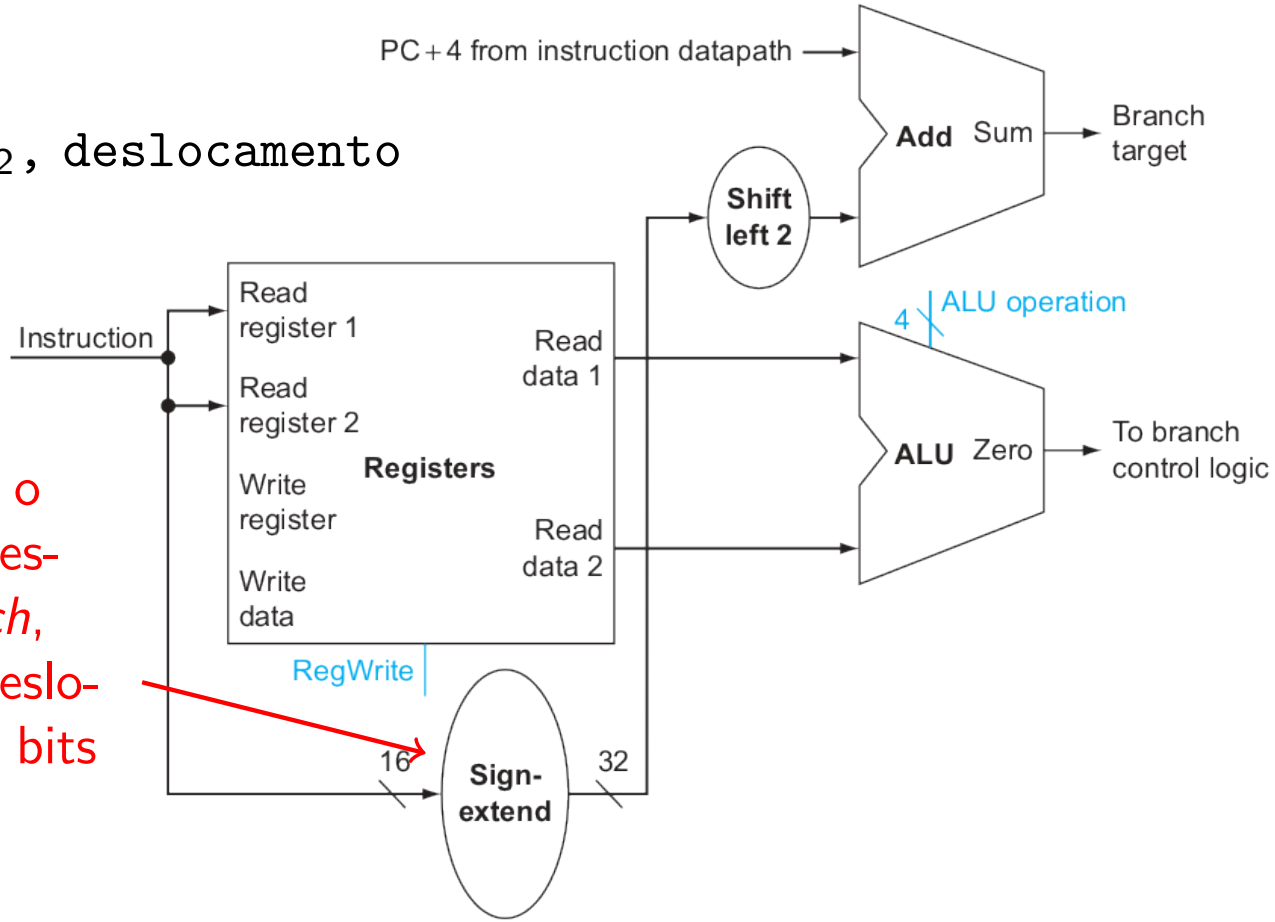
Fonte: [1]

Construindo o *Datapath*

Implementando beq

`beq` $\texttt{reg}_1$`,` $\texttt{reg}_2$`, deslocamento`

Para calcular o endereço de destino do *branch*, expandimos o deslocamento em 32 bits



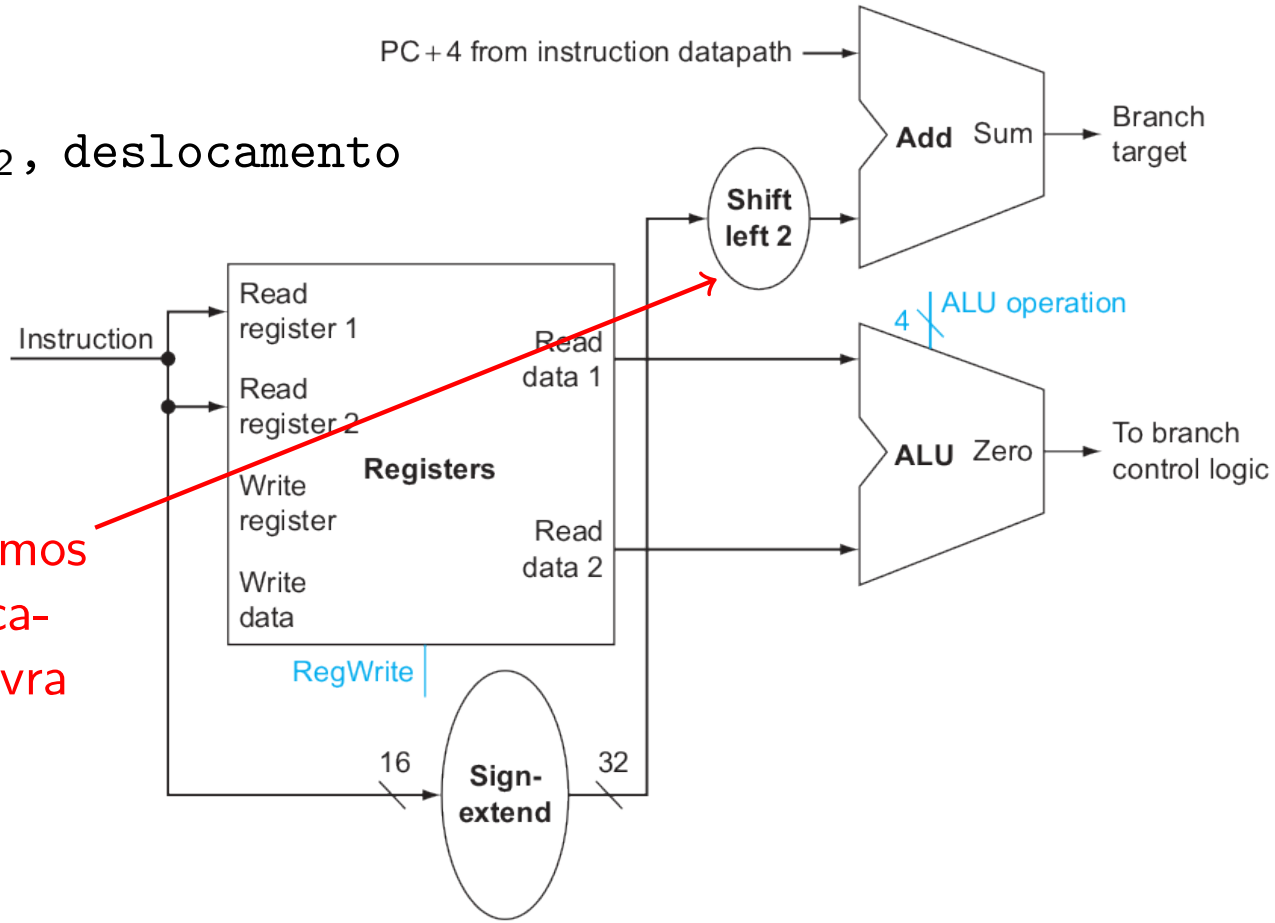
Fonte: [1]

Construindo o *Datapath*

Implementando beq

`beq` $\texttt{reg}_1$`,` $\texttt{reg}_2$`, deslocamento`

E o transformamos em um deslocamento de palavra



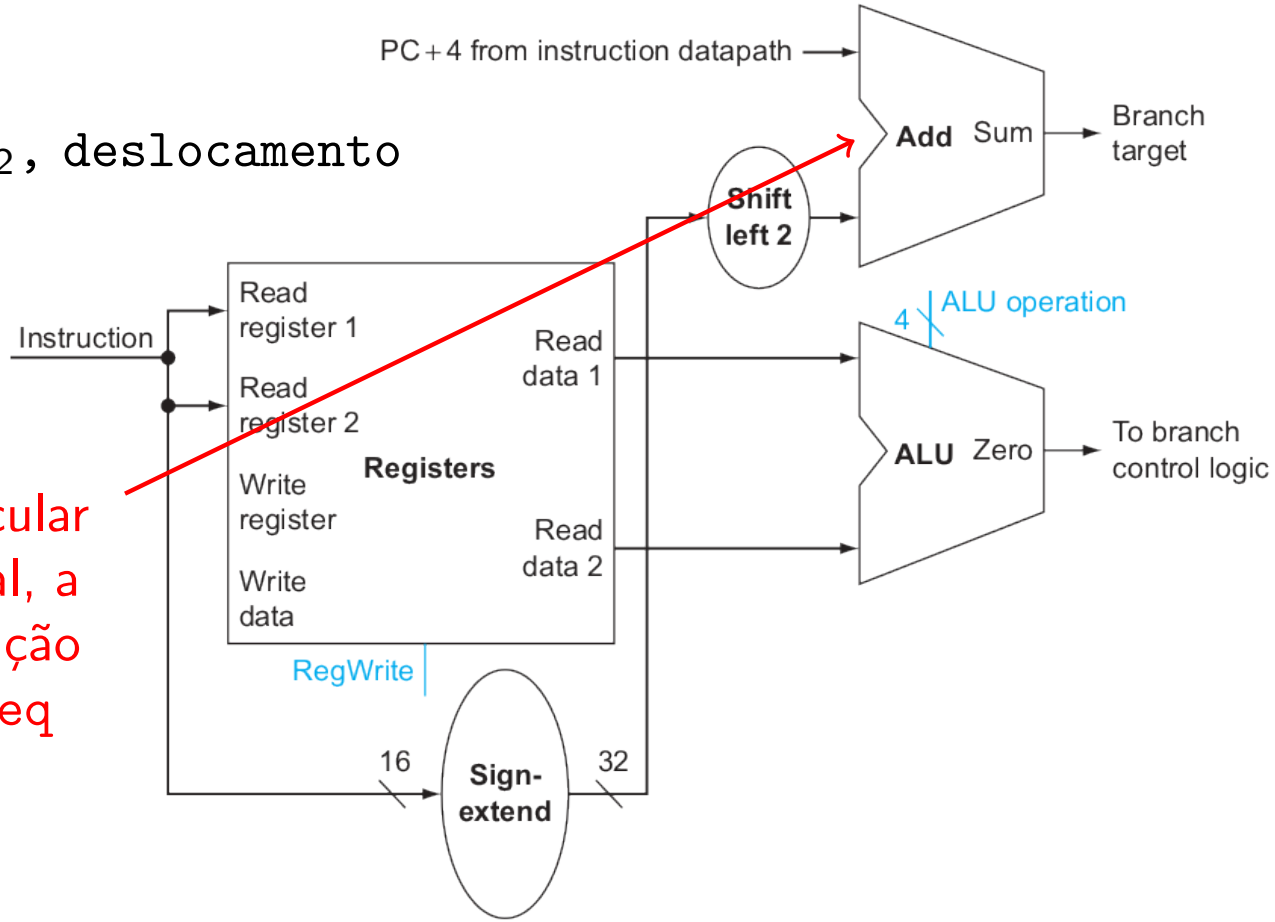
Fonte: [1]

Construindo o *Datapath*

Implementando beq

`beq` $\texttt{reg}_1$`,` $\texttt{reg}_2$`, deslocamento`

Para então calcular o endereço final, a partir da instrução seguinte ao beq

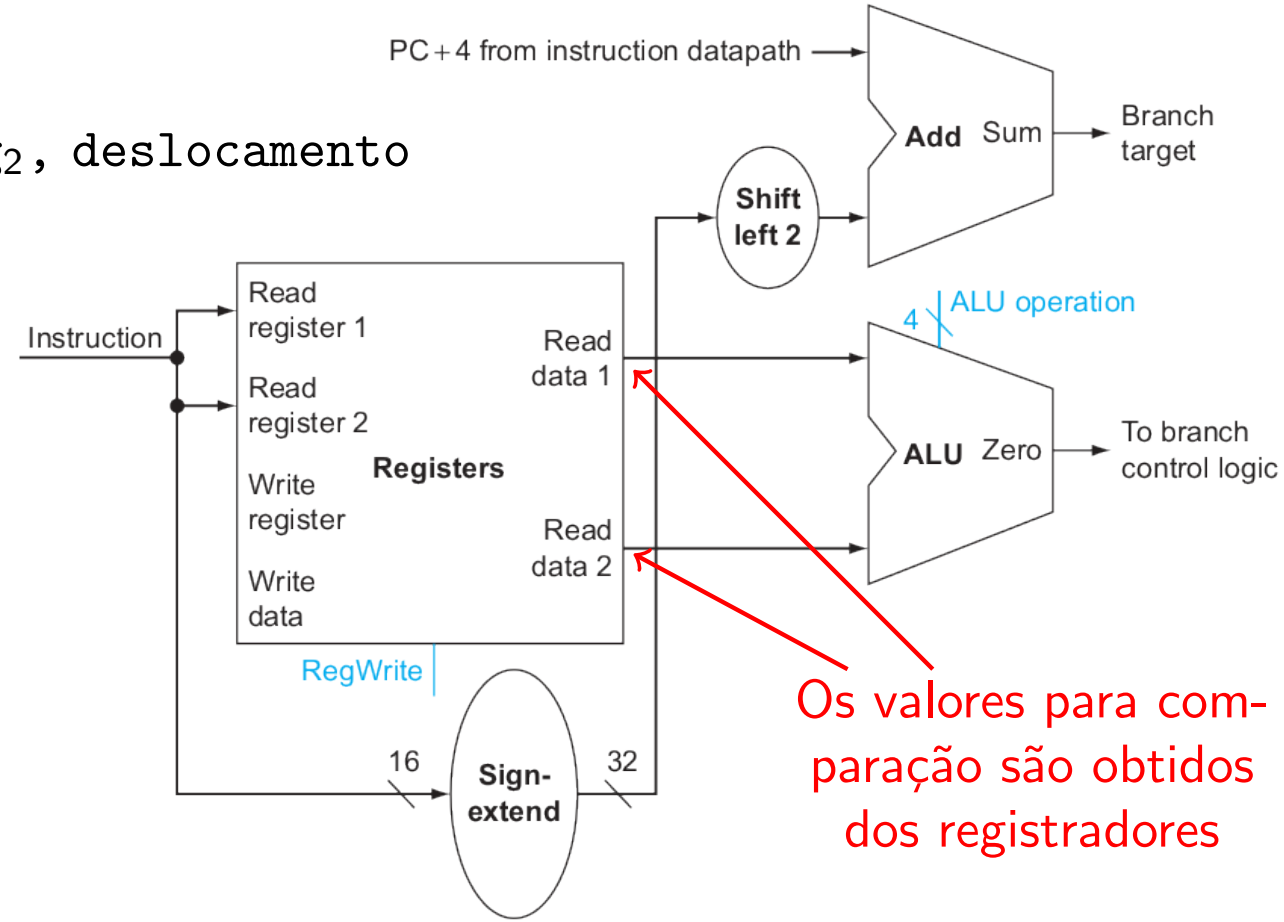


Fonte: [1]

Construindo o *Datapath*

Implementando beq

`beq` $\texttt{reg}_1$`,` $\texttt{reg}_2$`, deslocamento`

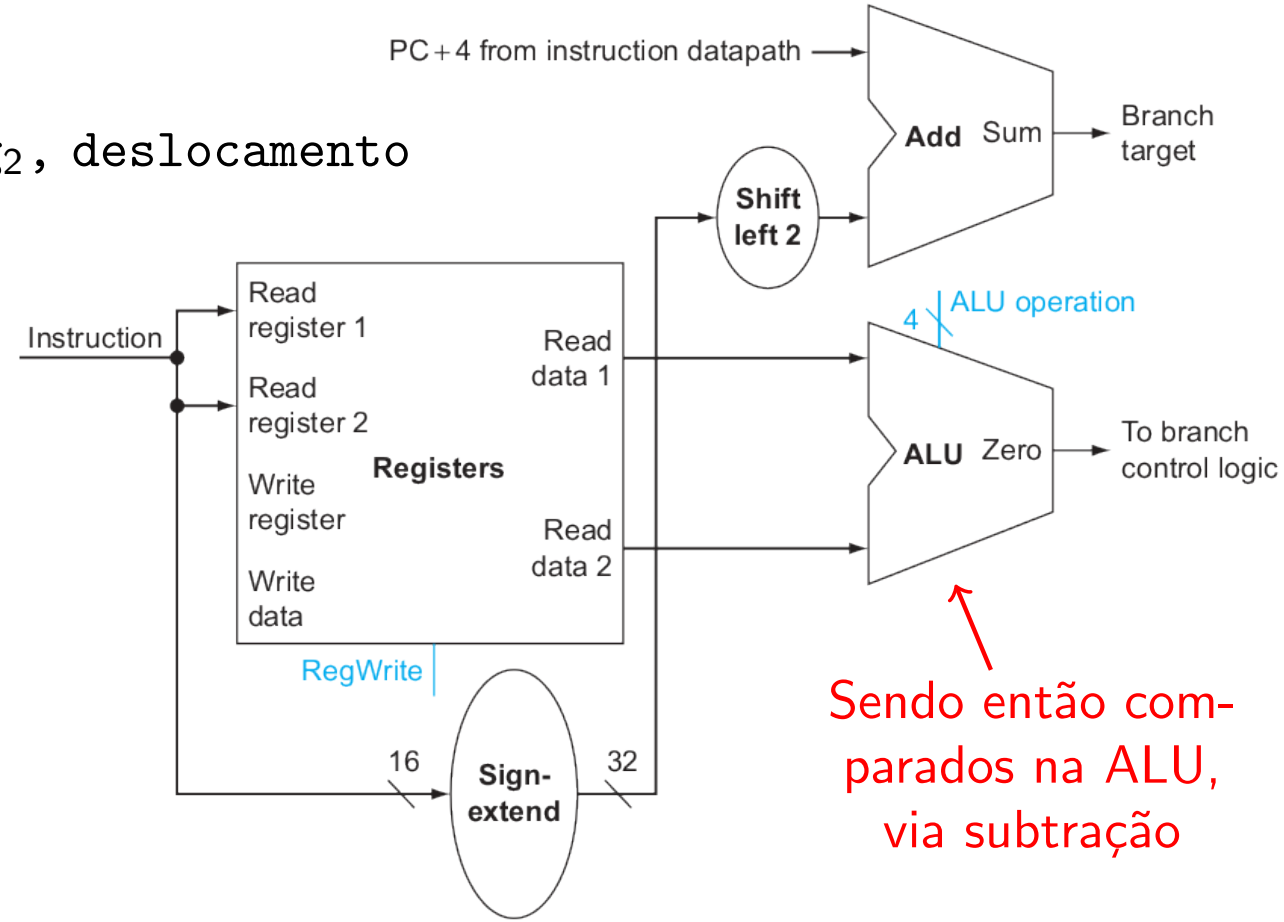


Fonte: [1]

Construindo o *Datapath*

Implementando beq

`beq` $\texttt{reg}_1$`,` $\texttt{reg}_2$`, deslocamento`

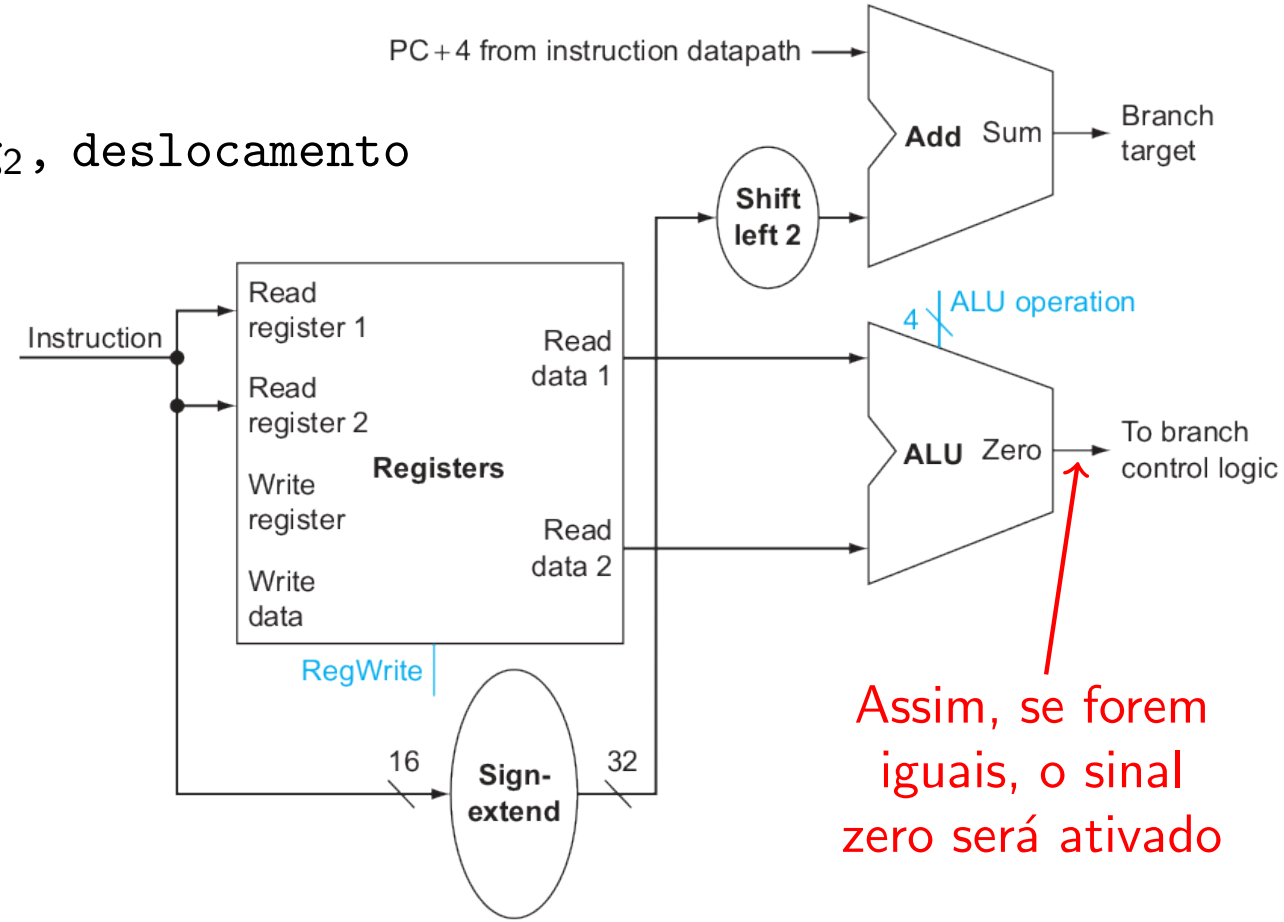


Fonte: [1]

Construindo o *Datapath*

Implementando beq

`beq` $\texttt{reg}_1$`,` $\texttt{reg}_2$`, deslocamento`

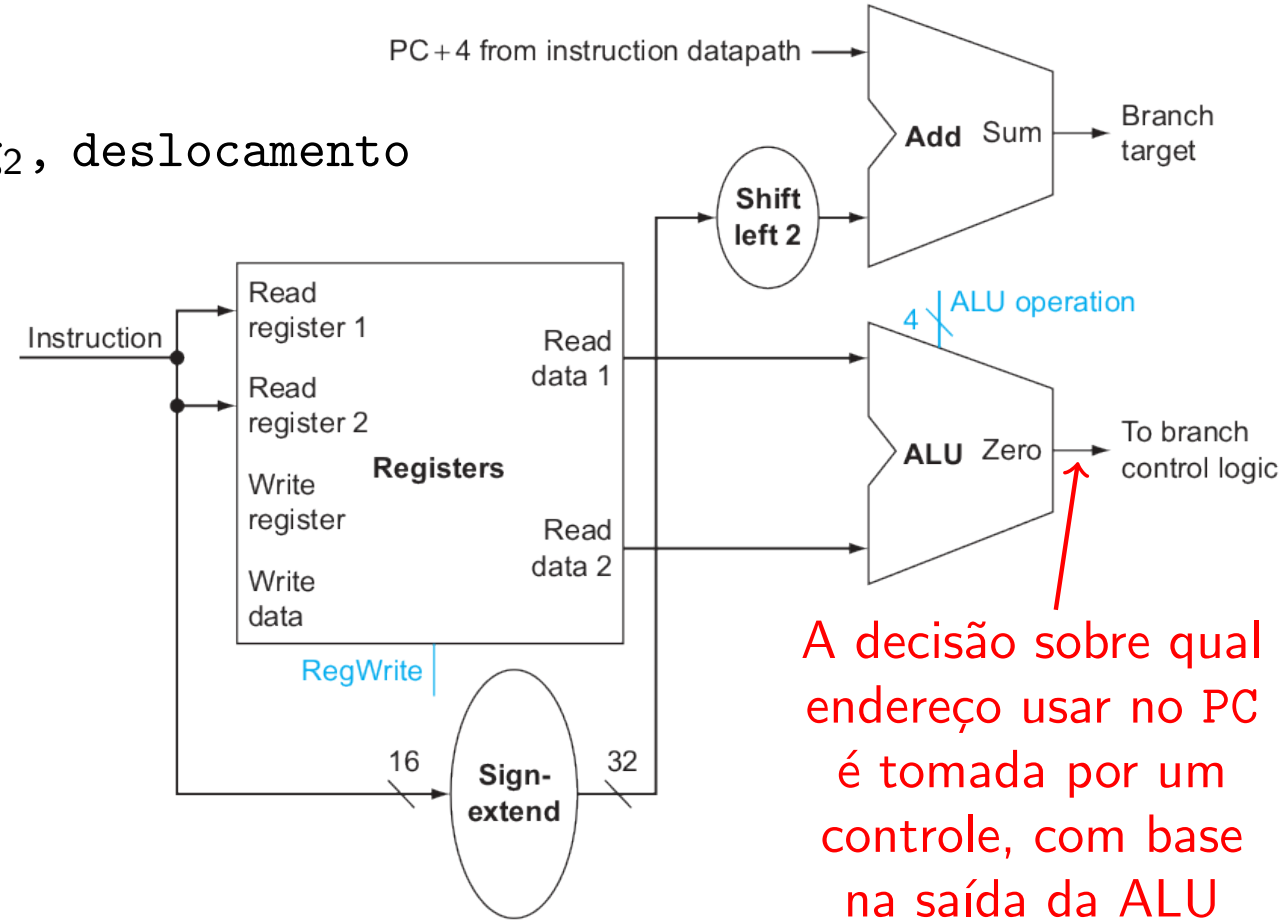


Fonte: [1]

Construindo o *Datapath*

Implementando beq

`beq` $\texttt{reg}_1$`,` $\texttt{reg}_2$`, deslocamento`

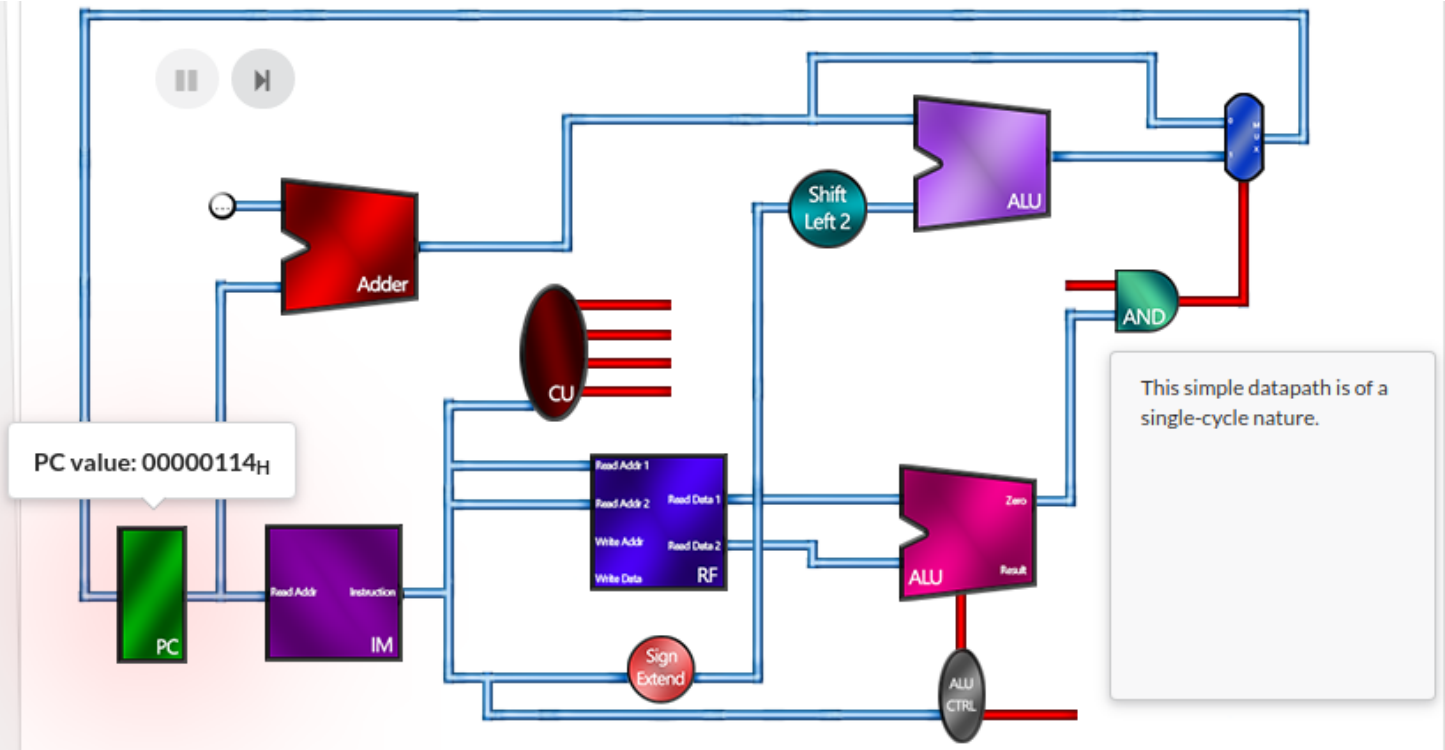


Fonte: [1]

Construindo o *Datapath*

Implementando beq

```
beq $9, $11, 12
```



https://www3.ntu.edu.sg/home/smitha/FYP_Gerald/beqInstruction.html

Construindo o *Datapath*

Implementando instruções do tipo J

Name	Fields						Comments
Field size	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	All MIPS instructions are 32 bits long
R-format	op	rs	rt	rd	shamt	funct	Arithmetic instruction format
I-format	op	rs	rt	address/immediate			Transfer, branch, imm. format
J-format	op	target address					Jump instruction format

Fonte: [1]

Construindo o *Datapath*

Implementando instruções do tipo J

Name	Fields						Comments
Field size	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	All MIPS instructions are 32 bits long
R-format	op	rs	rt	rd	shamt	funct	Arithmetic instruction format
I-format	op	rs	rt	address/immediate			Transfer, branch, imm. format
J-format	op	target address					Jump instruction format

Fonte: [1]

- Assim como com `beq`, "`j rótulo`" é, na verdade, "`j deslocamento`"
  - Deslocamos, em 2 bits à esquerda, os 26 bits do campo de endereço da instrução (vira deslocamento de palavra)
  - Substituímos os 28 bits menos significativos de `PC+4` (endereço seguinte ao `j`) por esses bits deslocados

Construindo o *Datapath*

Implementando `j`

- Diferentemente de `beq`, contudo, o deslocamento, num *jump* não é relativo
  - Não há a soma ao `PC+4`
  - Apenas são mantidos seus 4 bits mais significativos

Construindo o *Datapath*

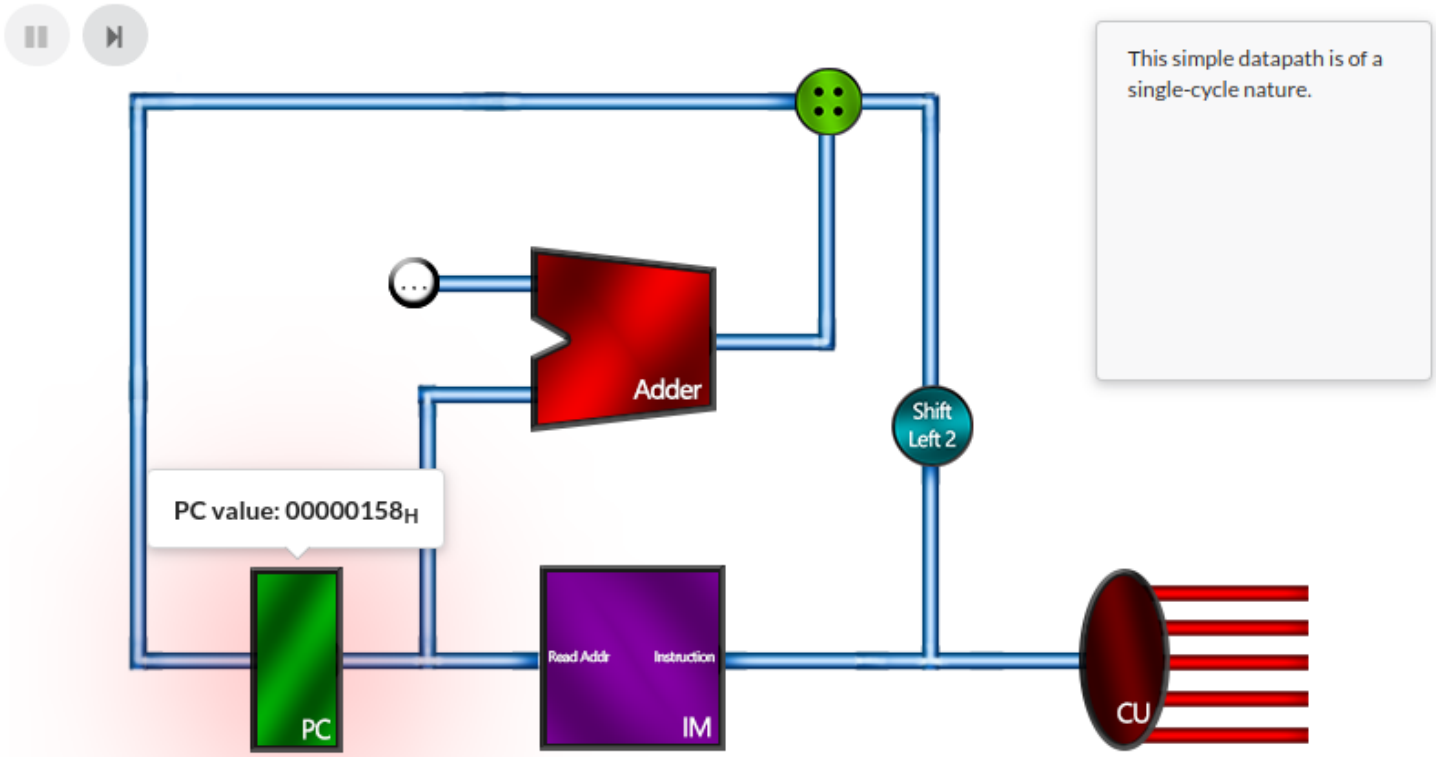
Implementando `j`

- Diferentemente de `beq`, contudo, o deslocamento, num *jump* não é relativo
  - Não há a soma ao `PC+4`
  - Apenas são mantidos seus 4 bits mais significativos
- No entanto, ainda depende do endereço do `j`
  - Justamente por manter os 4 bits do `PC+4`
  - Isso permite também que endereços anteriores ao `j` possam ser atingidos

Construindo o *Datapath*

Implementando `j`

```
j 512
```



https://www3.ntu.edu.sg/home/smitha/FYP_Gerald/jmpInstruction.html

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
3. `https://en.wikipedia.org/wiki/Arithmetic_logic_unit`
   - Apanhado interessante sobre a ALU