ORGANIZAÇÃO E ARQUITETURA DE COMPUTADORES

Arquitetura RISC-V Monociclo

Via de Dados e Unidade de Controle

Profª. Fabiana F F Peres

Apoio: Camile Bordini

Execução de Instruções

- É dividido em vários passos:
  - Busca a próxima instrução da memória;
  - Atualiza o **contador de programa** (Registrador) para que ele aponte para a próxima instrução;
  - Determina o **tipo da instrução** (Decodifica a instrução);
  - Acessa os dados contidos em registradores
  - Executa a instrução
    - faz a operação na **ULA**
    - Acessa a memória de dados, se necessário
  - Armazena o resultado em locais apropriados
  - Volta ao passo 1 (inicia a execução da próxima instrução)

Caminho de dados

- Lembrando
  - Um Sistema digital (computador simples) contém um **caminho de dados** e uma **unidade de controle**
- Caminho de dados:
  - Um conjunto de registradores
  - Micro operações nesses registradores
  - Interface de controle entre os registradores e outros components da CPU ou no caminho de dados

Implementações

- Há algumas formas de se implementar o conjunto de instruções de um processador:

1. **Monociclo**
2. **Multiciclo**
3. **Pipeline**

1. Monociclo

- **1 instrução** por ciclo de clock
- Para isso, o ciclo deve ser longo o suficiente para acomodar a **instrução mais lenta** (gargalo!)

2. Multiciclo

- Cada instrução é executada em **múltiplas etapas** (busca, decodificação, execução, acesso à memória e escrita de resultado)
- **Cada etapa** leva 1 ciclo
- O tempo de um ciclo deve acomodar a **etapa mais longa**
- Portanto, cada instrução pode ter uma quantidade de etapas diferentes
- Consequentemente, uma instrução pode levar menos ciclos que outras (usar apenas etapas necessárias)

Implementações

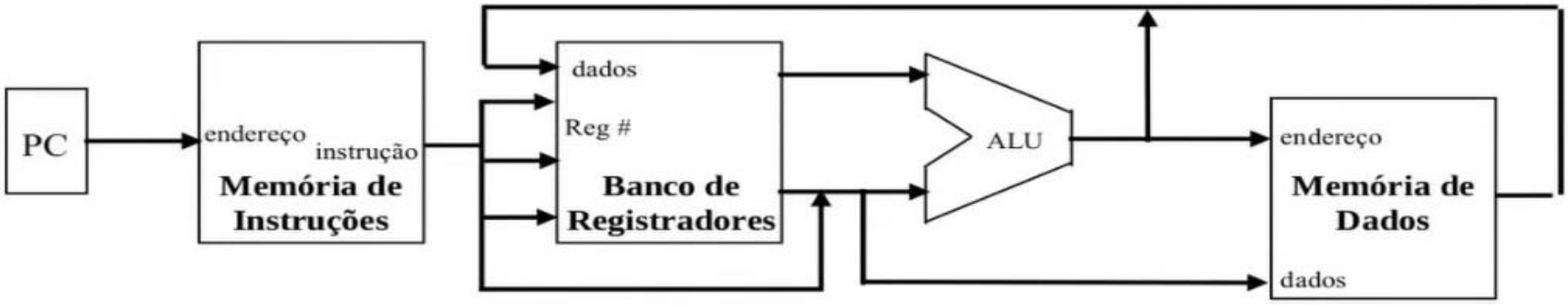
3. Pipeline

- Executa cada instrução em **múltiplas etapas** (como no **multiciclo**)
- Cada etapa executa uma instrução diferente – **etapas em paralelo** – como em uma linha de montagem

Monociclo

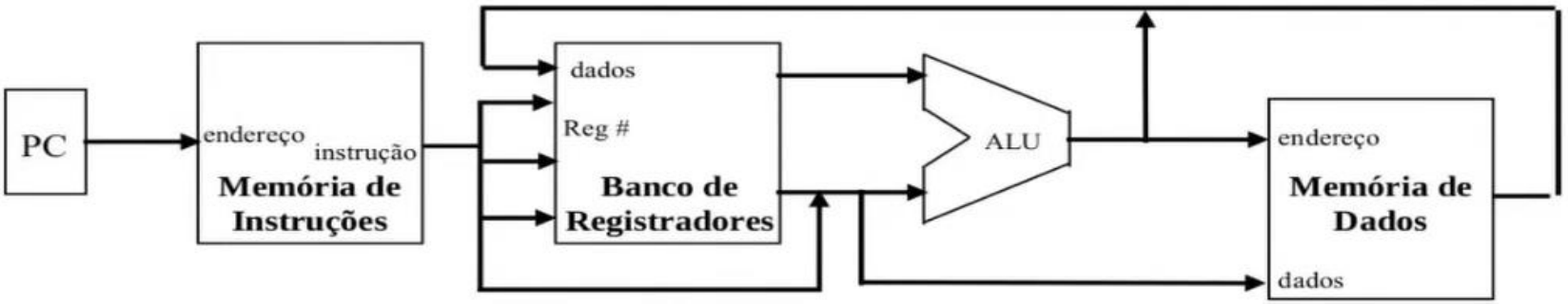
Via de Dados Monociclo

- Busca: **PC** precisa guardar o endereço da instrução a ser buscada
- A instrução precisa estar na **memória de instruções**
- A instrução pode ter operandos de entrada que são armazenados no **banco de registradores**
- Os quais serão processados pela **ULA**
- O resultado da ULA deve ser armazenado em uma **memória de dados**



Via de Dados Monociclo

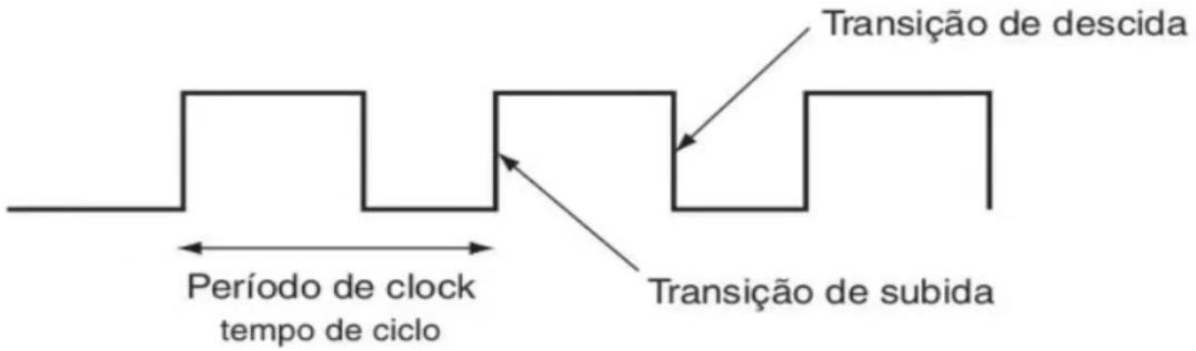
- Lembrando, há dois tipos de unidades funcionais:
  - Elementos que operam nos **valores dos dados** (**combinacionais**): para uma certa entrada, produz sempre a mesma saída
    - Ex: ULA
  - Elementos que contém **estado** (**sequenciais**): para uma certa entrada, a saída pode ser difente, depende do estado (necessário memória)
    - Ex: PC, memória de instruções, banco de registradores, memória de dados



Elementos de Estado

- Em relação aos elementos **sequenciais** (de **estado**), em que momento o **valor do estado** pode ser armazenado?
  - São baseados no **clock** (lógica síncrona)

- **Clock**: implementado através de um dispositivo que oscila (alto/baixo)

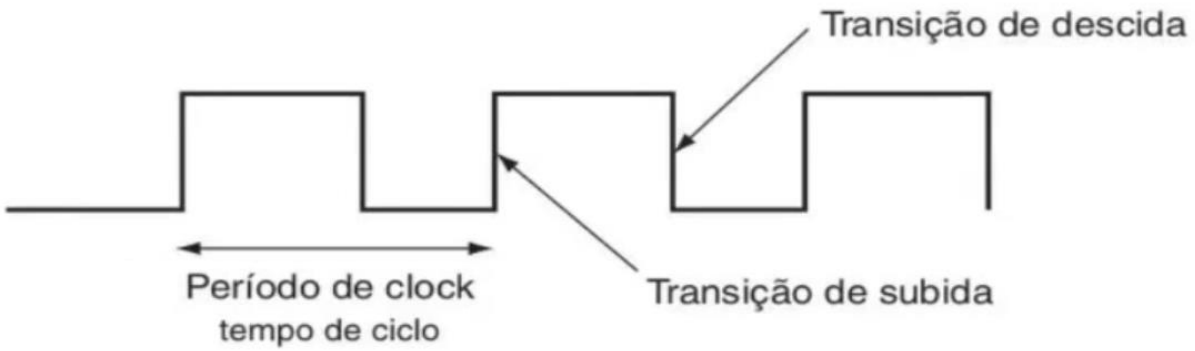


*Obs: elementos **combinacionais**, como produzem o mesmo resultado, não há necessidade de um controle por clock.*

Elementos de Estado

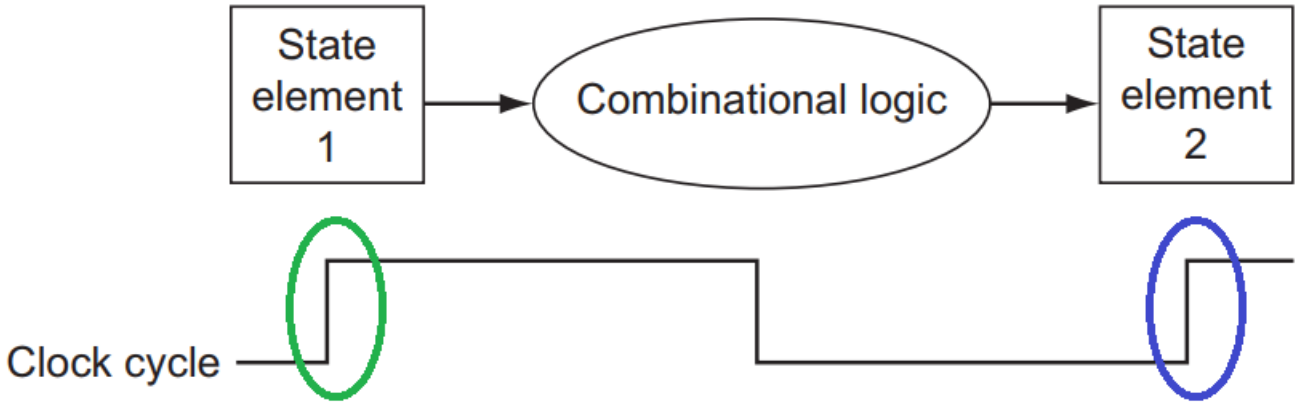
- **Métodos de clock (*clocking*)**: que define quando que o elemento de estado deve ser atualizado: Na transição de subida ou de descida?
  - Fabricante deve decidir!

- Supondo que os elementos de estado serão atualizados durante a **transição de subida**
- Assim, mesmo que algum dado seja alterado no meio do ciclo, ele só vai ser atualizado no momento da **subida**



Elementos de Estado

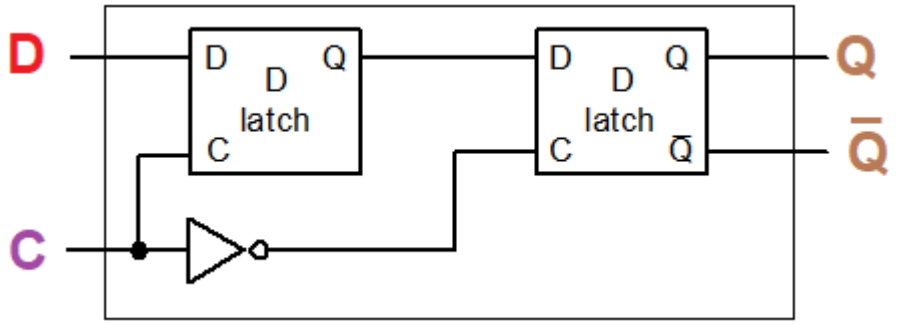
- **Execução típica:**
  - Leitura do conteúdo de alguns elementos de estado
  - Enviado valores (através de alguma lógica combinacional)
  - Escrita dos resultados em um ou mais elementos de estado

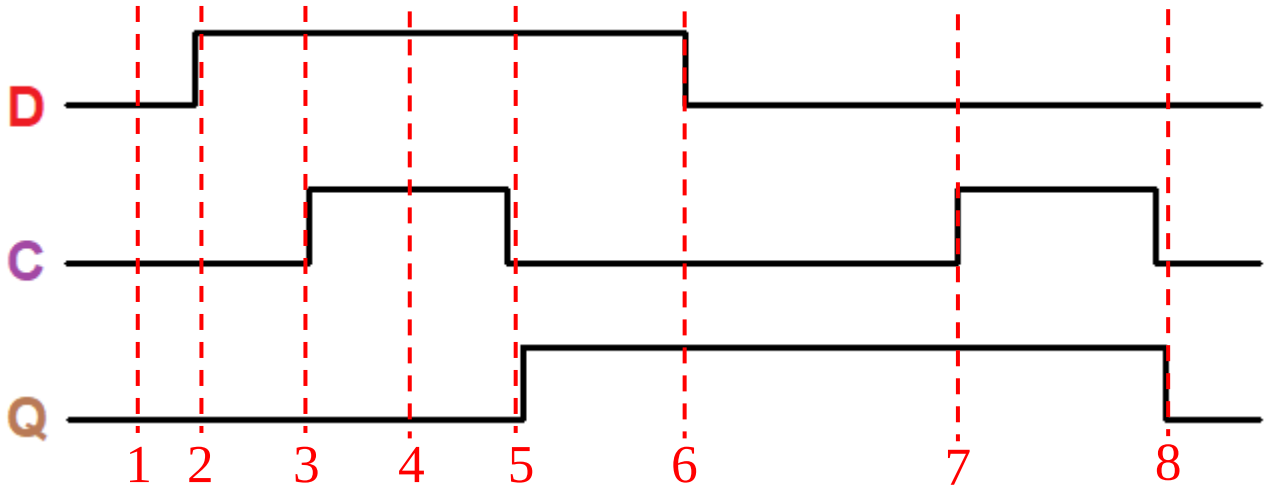


Elementos de Estado

Lembrando…

- **Flip-flop D**
  - Armazena 1 bit
  - o estado (**Q**) muda somente na transição do clock (**C**)…
    - …de 0 para 1,
    - …ou de 1 para 0 (exemplo abaixo)





Ex: Banco de registradores

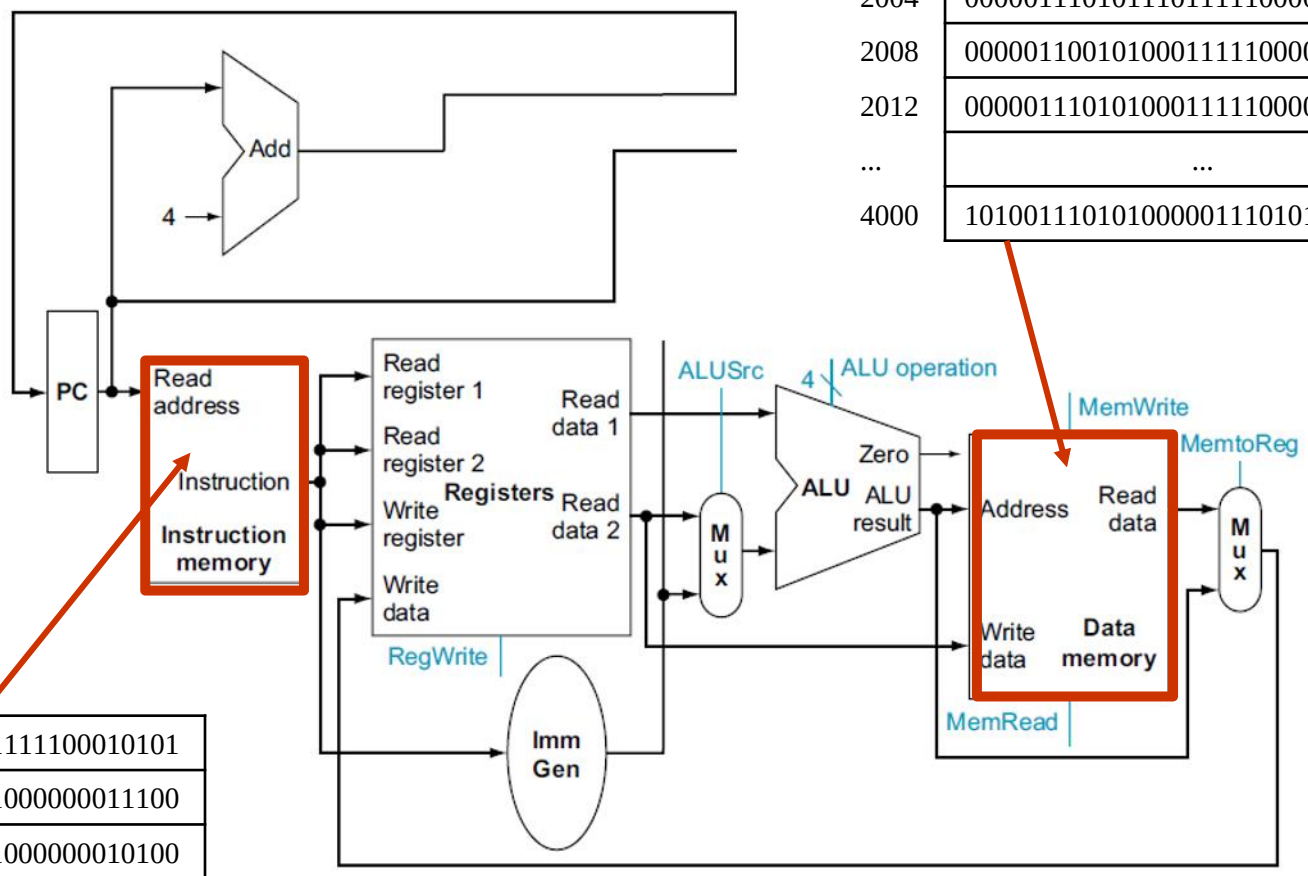
- São construídos utilizando vários **flip-flops D**
  - 32 registradores
- **Leituras**:
  - **2 registradores** para leituras
  - **2 dados de saída** a serem processados pela ULA
- **Escritas**:
  - **1 registrador**
  - **Dado a ser escrito**
- Sinal de controle ***write***: indicando se é para ser escrito algo em algum registrador

Caminho de dados por instrução

Via de Dados Monociclo

Memória de dados

2000	10110001001101000011111100010101
2004	00000111010111011111000001011101
2008	00000110010100011111000000010111
2012	00000111010100011111000011111000
...	...
4000	10100111010100000111010101011100



Memória de instruções

8000	10110000000101000011111100010101
8004	00000111010100011111000000011100
8008	00000110010100011111000000010100
8012	00000111010100011111000000011000
...	...
9000	00100111010100000111000000011100

Busca

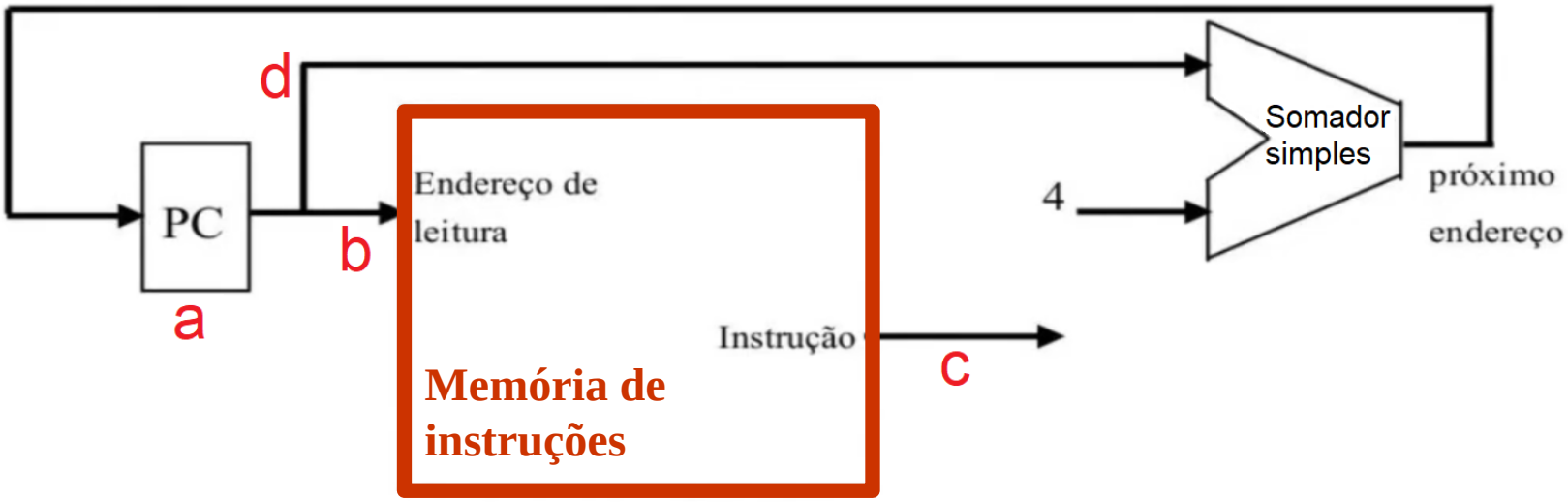
a. O endereço da instrução a ser buscada está em **PC**

b. O endereço é enviado à **memória de instruções**

`IR = Memória[PC]`

c. Instrução é armazenada em **IR**

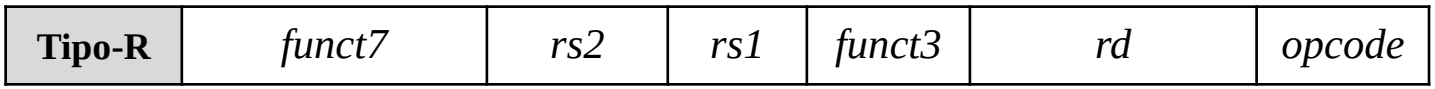
d. **PC** é atualizado para guardar a próxima instrução (incremento em 4 bytes) (`PC = PC + 4`)



Instruções Tipo R

Instruções Tipo R

- **Instruções Tipo R:** ***(add, sub, and, or)***



- **Instância da instrução Add:**

```
add rd, rs1, rs2      //rd = rs1 + rs2
```

- **Requisitos para execução:**

• Ler o conteúdo de dois **registradores** (`rs1` e `rs2`)

• Realizar uma *operação*, com o conteúdo dos **registradores**, na **ULA**

• Escrever o resultado da operação em um outro **registrador** (`rd`)

a. **Instrução** é acessada pelo registrador IR: endereços dos **3 registradores** (2 leitura e 1 de escrita).

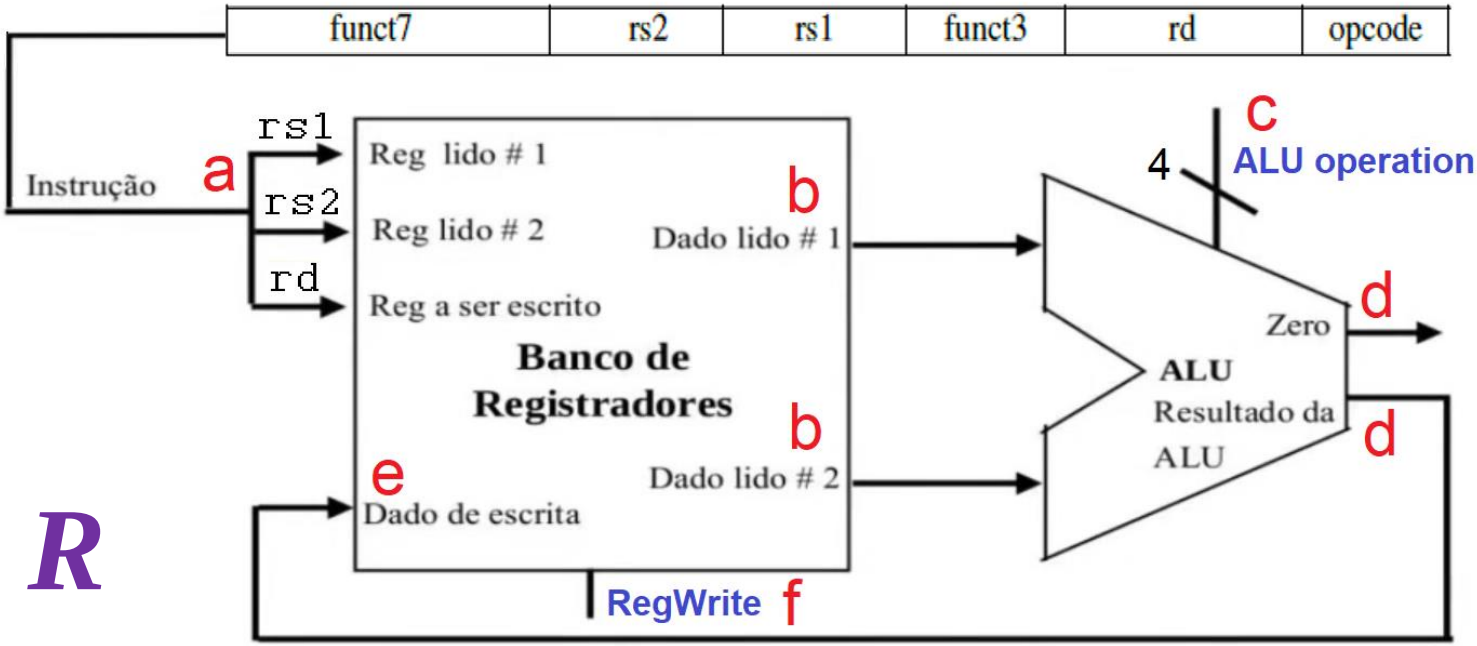
b. **Dados lidos** dos 2 registradores e enviados à ULA.

c. **ALU operation**: sinal de controle indicando qual operação é pra ser executada.

d. Dois sinais de saída: *flag* indicando se o resultado deu **zero**, e o **resultado da operação**.

e. **Resultado deve ser escrito** devolta no banco de registradores

f. **RegWrite**: sinal de escrita em registrador deve estar ativado



Tipo R

Sinais de Controle

- **ALU Operation**
  - Bits para identificação de qual operação deve ser executada na ULA
  - Ex: add, sub, and, or, lw, sw,…?

Sinais de Controle

Nome do sinal	Efeito quando desativado	Efeito quando ativado
RegWrite	Nenhum	É escrito em um registrador (*Write register)* o valor existente em *Write date*
ALUSrc	O segundo operando da ULA é oriundo do segundo registrador (*Read data 2*)	O segundo operando da ULA é a constante (imediato) estendido para 32 bits
PCSrc	PC é substituído pela saída do somador que calcula o valor de PC+4	PC é substituído pela saída do somador que calcula o alvo de um desvio (instruções de **Branch**)
MemRead	Nenhum	O conteúdo da **Memória de dados** armazenado no endereço de entrada (*Address*) é colocado na saída (*Read data*)
MemWrite	Nenhum	O conteúdo da **Memória de dados** armazenado no endereço de entrada (*Address*) é substituído pelo dado na entrada (*Write data*)
MemtoReg	O valor alimentado no **registrador** de entrada de dados (*Write data*) vem da **ULA**	O valor alimentado no **registrador** de entrada de dados (*Write data*) vem da **Memória de dados**

Instruções Tipo I

Instruções Tipo I

- **Instruções Tipo I:** ***(lw)***

Tipo-I	*imm[11:0]*	*rs1*	*funct3*	*rd*	*opcode*

- **Instância da instrução Load:**

```
lw rd, imm(rs1)   //rd=Mem[rs1+ImmGen(imm)]
```

- **Requisitos para execução:**
  - Ler o conteúdo do **registrador** de base (`rs1`)
  - Realizar a soma na **ULA**: do **registrador** de base com a **constante** `imm` para calcular o **endereço da memória**
  - Acessar a **memória**:
    - para ler o **dado** contido dentro do endereço calculado
  - Escrever o **dado** lido da memória dentro de um **registrador** (`rd`).

Instruções Tipo I

- **Instruções Tipo I:** ***(lw)***

Tipo-I	*imm[11:0]*	*rs1*	*funct3*	*rd*	*opcode*

- **Instância da instrução Load:**

```
lw rd, imm(rs1)   //rd=Mem[rs1+ImmGen(imm)]
```

- Observação:
  - O valor do offset da instrução (`imm`) pode ser positivo ou negativo
  - E como o endereço guardado por `rs1` é de 32 bits, para facilitar a soma, é necessário que o valor de imediato também tenha 32 bits
  - Assim, um **Gerador de Imediato** irá extender o sinal (bit mais à esquerda) replicando-o até atingir 32 bits

a. **Instrução** é acessada pelo registrador IR: `rs1`, `rd`, `imm`

b. **Gerador de imediato** extende o valor da constante de imediato (32bits)

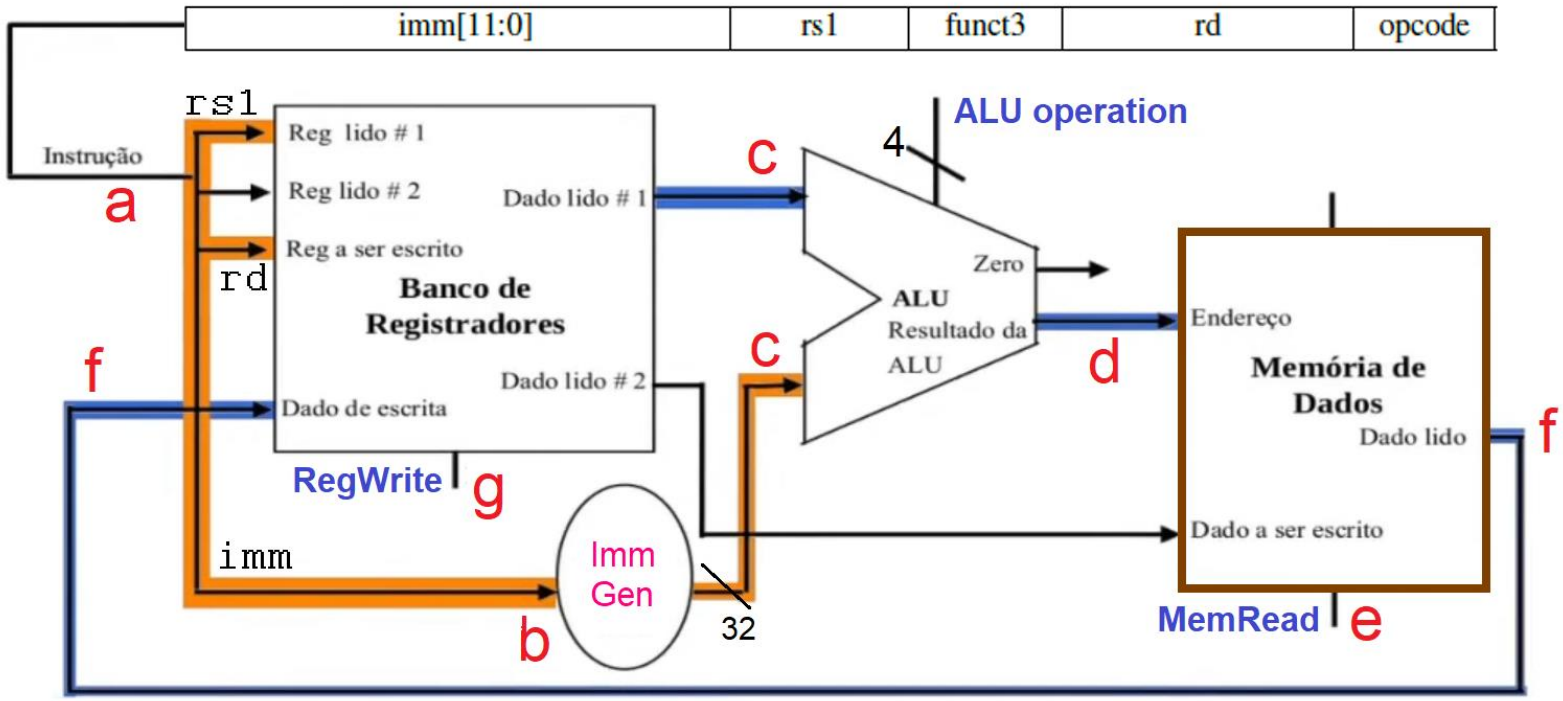
c. Entradas da ULA: **endereço base** (dado lido de `rs1`) e o **valor de imediato** já extendido (deslocamento) – **AluOp** indicando a operação de soma

d. **Endereço efetivo** é encaminhado à memória de dados

e. Sinal de controle **MemRead** deve estar ativo

f. **Dado é lido da memória** e deve ser **escrito no banco de registradores**

g. Sinal de controle **RegWrite** deve estar ativo



Tipo I

Sinais de Controle

Nome do sinal	Efeito quando desativado	Efeito quando ativado
RegWrite	Nenhum	É escrito em um registrador (*Write register)* o valor existente em *Write date*
ALUSrc	O segundo operando da ULA é oriundo do segundo registrador (*Read data 2*)	O segundo operando da ULA é a constante (imediato) estendido para 32 bits
PCSrc	PC é substituído pela saída do somador que calcula o valor de PC+4	PC é substituído pela saída do somador que calcula o alvo de um desvio (instruções de **Branch**)
MemRead	Nenhum	O conteúdo da **Memória de dados** armazenado no endereço de entrada (*Address*) é colocado na saída (*Read data*)
MemWrite	Nenhum	O conteúdo da **Memória de dados** armazenado no endereço de entrada (*Address*) é substituído pelo dado na entrada (*Write data*)
MemtoReg	O valor alimentado no **registrador** de entrada de dados (*Write data*) vem da **ULA**	O valor alimentado no **registrador** de entrada de dados (*Write data*) vem da **Memória de dados**

Instruções Tipo S

Instruções Tipo S

- **Instruções Tipo S:** ***(sw)***

Tipo-S	*imm[11:5]*	*rs2*	*rs1*	*funct3*	*imm[4:0]*	*opcode*

- **Instância da instrução Store:**

```
sw rs2, imm(rs1)   //Mem[rs1+ImmGen(imm)]=rs2
```

- **Requisitos para execução:**

  - Ler o conteúdo do **registradores** `rs1` e `rs2`
  - Realizar a soma na **ULA**: do **registrador** de base (`rs1`) com a **constante** `imm` para calcular o **endereço da memória**
  - Acessar a **memória**:
    - para escrever o valor contido em `rs2` no endereço calculado

a. **Instrução** é acessada pelo registrador IR: `rs1`, `rs2`, `imm`

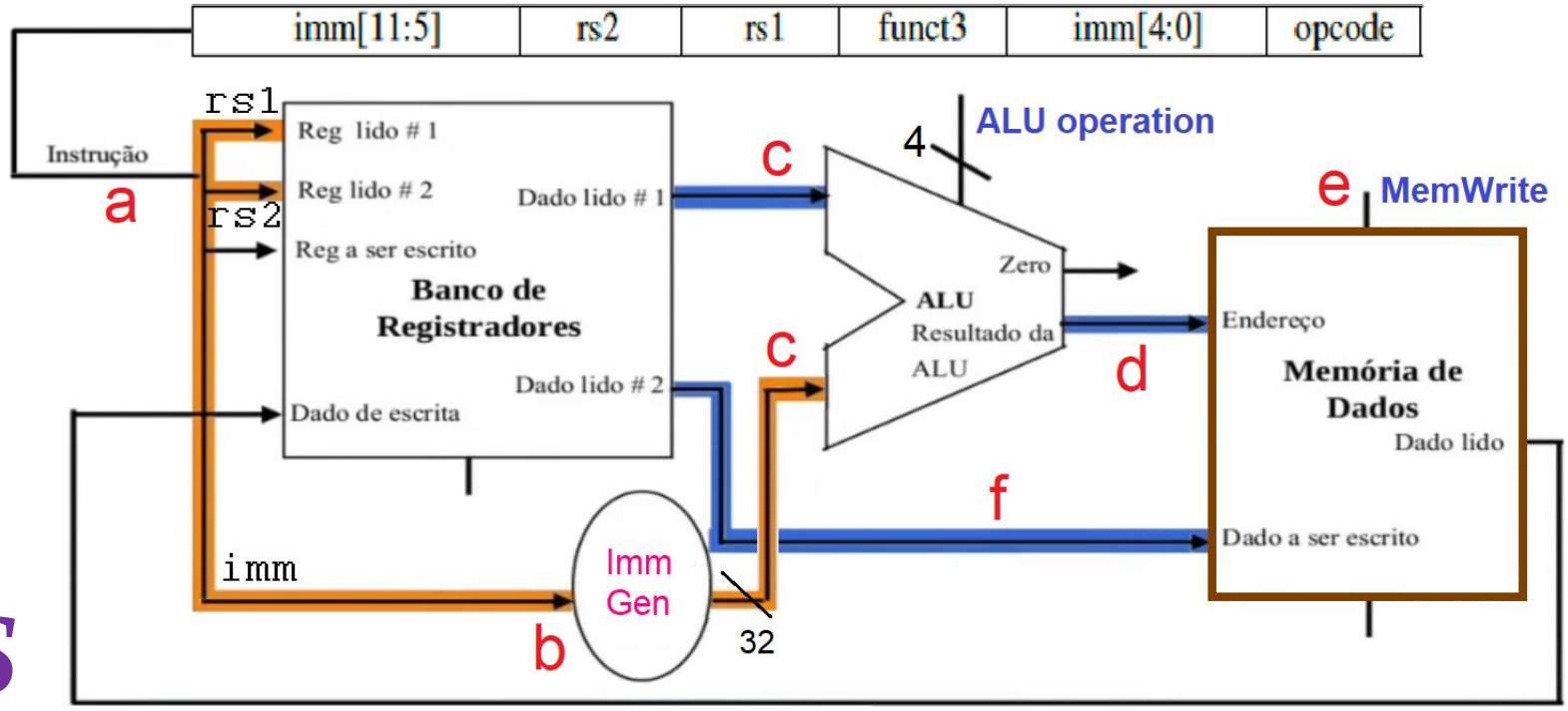
b. **Gerador de imediato** extende o valor da constante de imediato (32bits)

c. Entradas da ULA: **endereço base** (dado lido de `rs1`) e o **valor de imediato** já extendido (deslocamento) – **AluOp** indica a operação (soma)

d. **Endereço efetivo** é encaminhado à memória de dados

e. Sinal de controle **MemWrite** deve estar ativo

f. **Dado do registrador** `rs2` é trafegado para a **memória de dados**



Tipo S

Sinais de Controle

Nome do sinal	Efeito quando desativado	Efeito quando ativado
RegWrite	Nenhum	É escrito em um registrador (*Write register)* o valor existente em *Write date*
ALUSrc	O segundo operando da ULA é oriundo do segundo registrador (*Read data 2*)	O segundo operando da ULA é a constante (imediato) estendido para 32 bits
PCSrc	PC é substituído pela saída do somador que calcula o valor de PC+4	PC é substituído pela saída do somador que calcula o alvo de um desvio (instruções de **Branch**)
MemRead	Nenhum	O conteúdo da **Memória de dados** armazenado no endereço de entrada (*Address*) é colocado na saída (*Read data*)
MemWrite	Nenhum	O conteúdo da **Memória de dados** armazenado no endereço de entrada (*Address*) é substituído pelo dado na entrada (*Write data*)
MemtoReg	O valor alimentado no **registrador** de entrada de dados (*Write data*) vem da **ULA**	O valor alimentado no **registrador** de entrada de dados (*Write data*) vem da **Memória de dados**

Instruções Tipo B

Instrução Tipo B

- **Instruções Tipo B:** ***(beq)***

Tipo-B	*imm[12]|imm[10:5]*	*rs2*	*rs1*	***funct3***	*imm[4:1]|imm[11]*	***opcode***

- **Instância da instrução <u>Beq</u>:**

```
beq rs1, rs2, L
//if rs1==rs2 then PC = PC + ImmGen(imm)
```

- **Requisitos para execução:**
  - Ler o conteúdo de dois **registradores** (`rs1` e `rs2`)
  - Utilizar a **ULA** para identificar se armazenam o mesmo valor
  - Se armazenam o mesmo valor: o PC será atualizado com PC +`imm`

a. **Instrução** é acessada pelo registrador IR: `rs1, rs2, L`

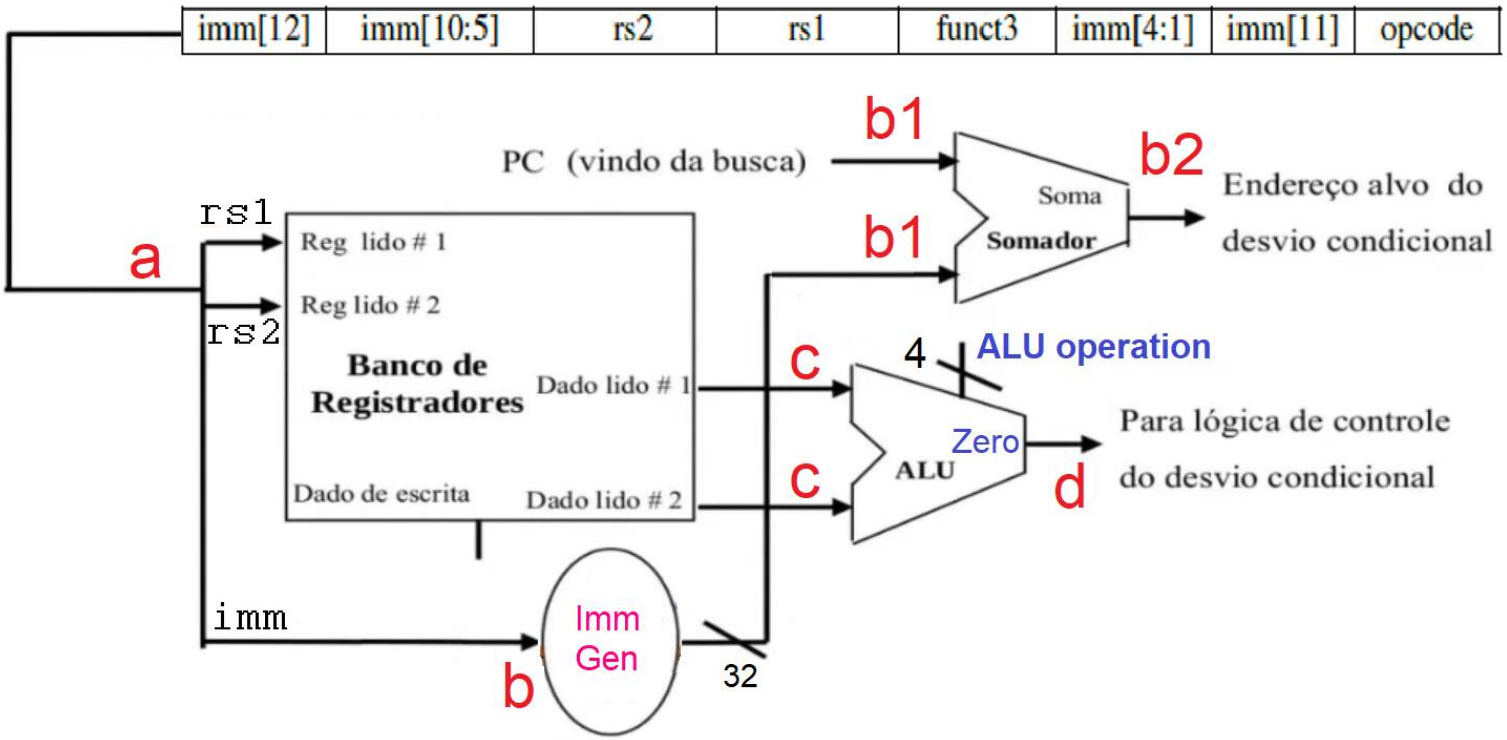
b. **Gerador de imediato** extende o valor da constante de imediato (32bits)

b1 O **imediato já estendido** é somado com o **PC**

b2 Produzindo como saída o **endereço do desvio** que (*talvez**) irá para o PC

c. Entradas da ULA: **valor de** `rs1` e **valor de** `rs2` –**AluOp** indica a operação

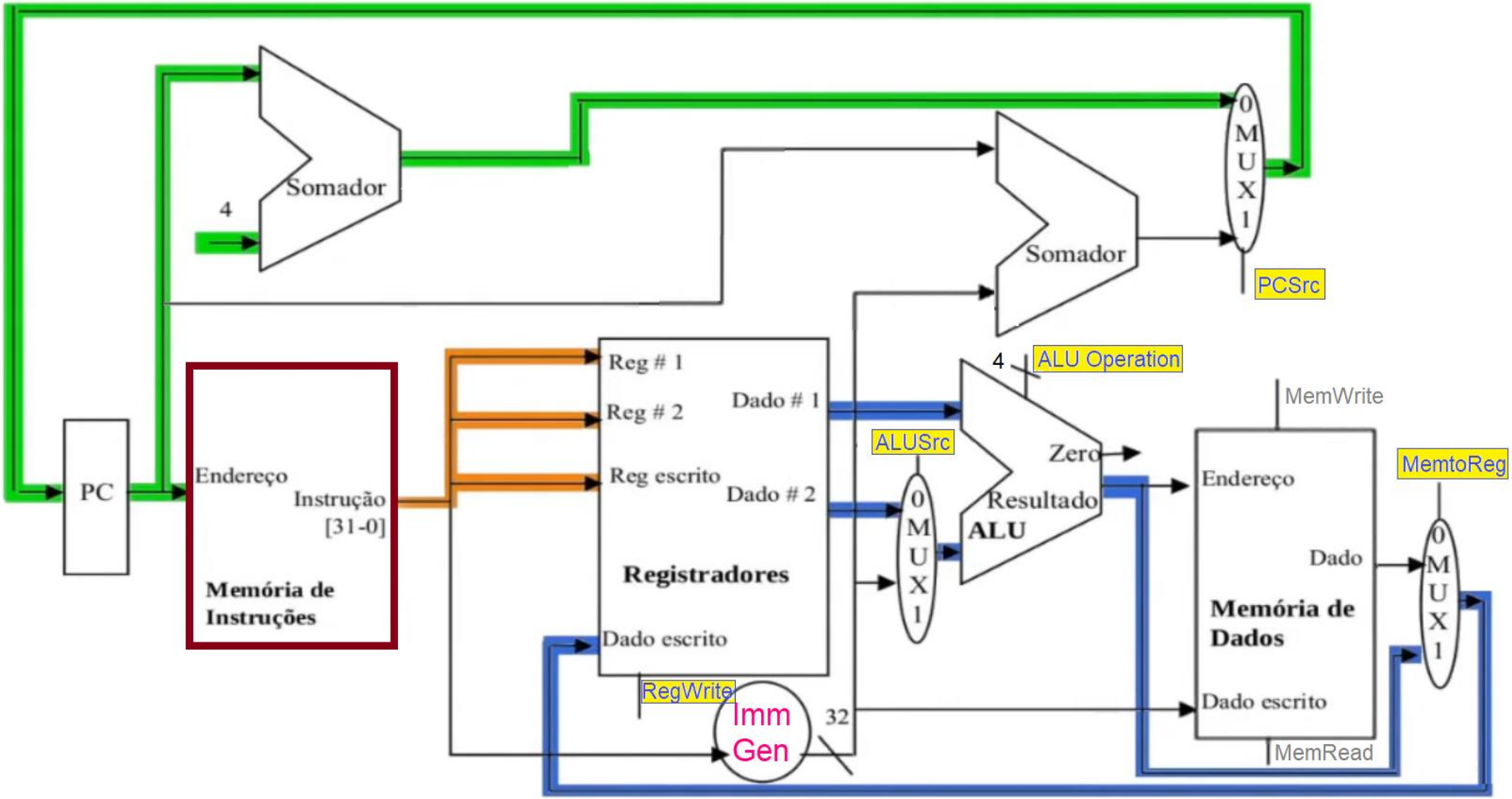
d. Valores são **subtraídos**: se resultar em 0 é porque são iguais e **sinal Zero** é ativado (*permitindo que o valor da soma anterior – endereço do desvio – seja atualizado em PC)



Tipo B

Caminho de dados completo

Caminho de dados: Instrução Tipo R

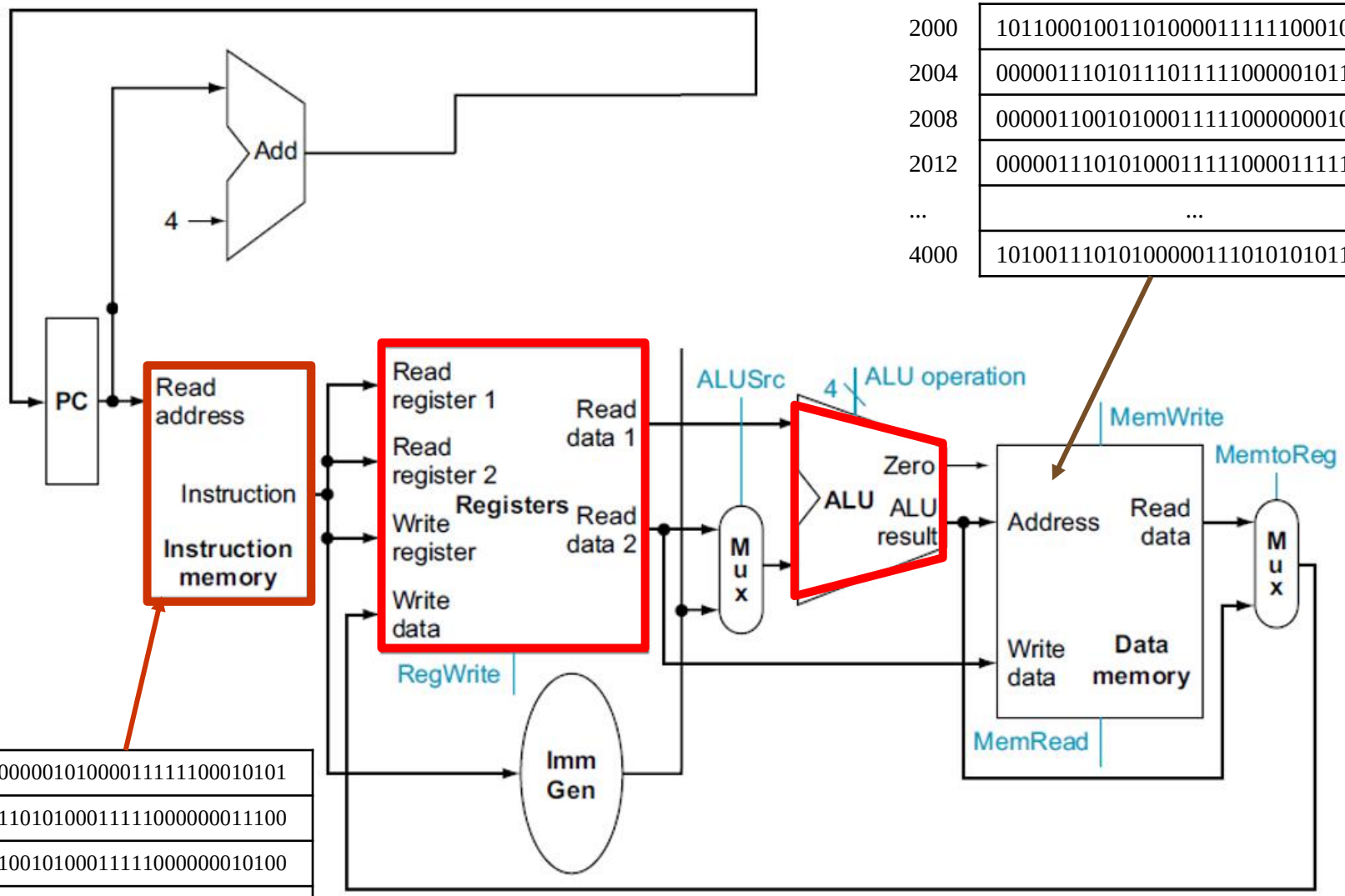


`add rd, rs1, rs2` `//rd = rs1 + rs2`

Sinais de Controle – Tipo R

Nome do sinal	Efeito quando desativado	Efeito quando ativado
RegWrite	Nenhum	É escrito em um registrador (*Write register)* o valor existente em *Write date*
ALUSrc	O segundo operando da ULA é oriundo do segundo registrador (*Read data 2*)	O segundo operando da ULA é a constante (imediato) estendido para 32 bits
PCSrc	PC é substituído pela saída do somador que calcula o valor de PC+4	PC é substituído pela saída do somador que calcula o alvo de um desvio (instruções de **Branch**)
MemRead	Nenhum	O conteúdo da **Memória de dados** armazenado no endereço de entrada (*Address*) é colocado na saída (*Read data*)
MemWrite	Nenhum	O conteúdo da **Memória de dados** armazenado no endereço de entrada (*Address*) é substituído pelo dado na entrada (*Write data*)
MemtoReg	O valor alimentado no **registrador** de entrada de dados (*Write data*) vem da **ULA**	O valor alimentado no **registrador** de entrada de dados (*Write data*) vem da **Memória de dados**

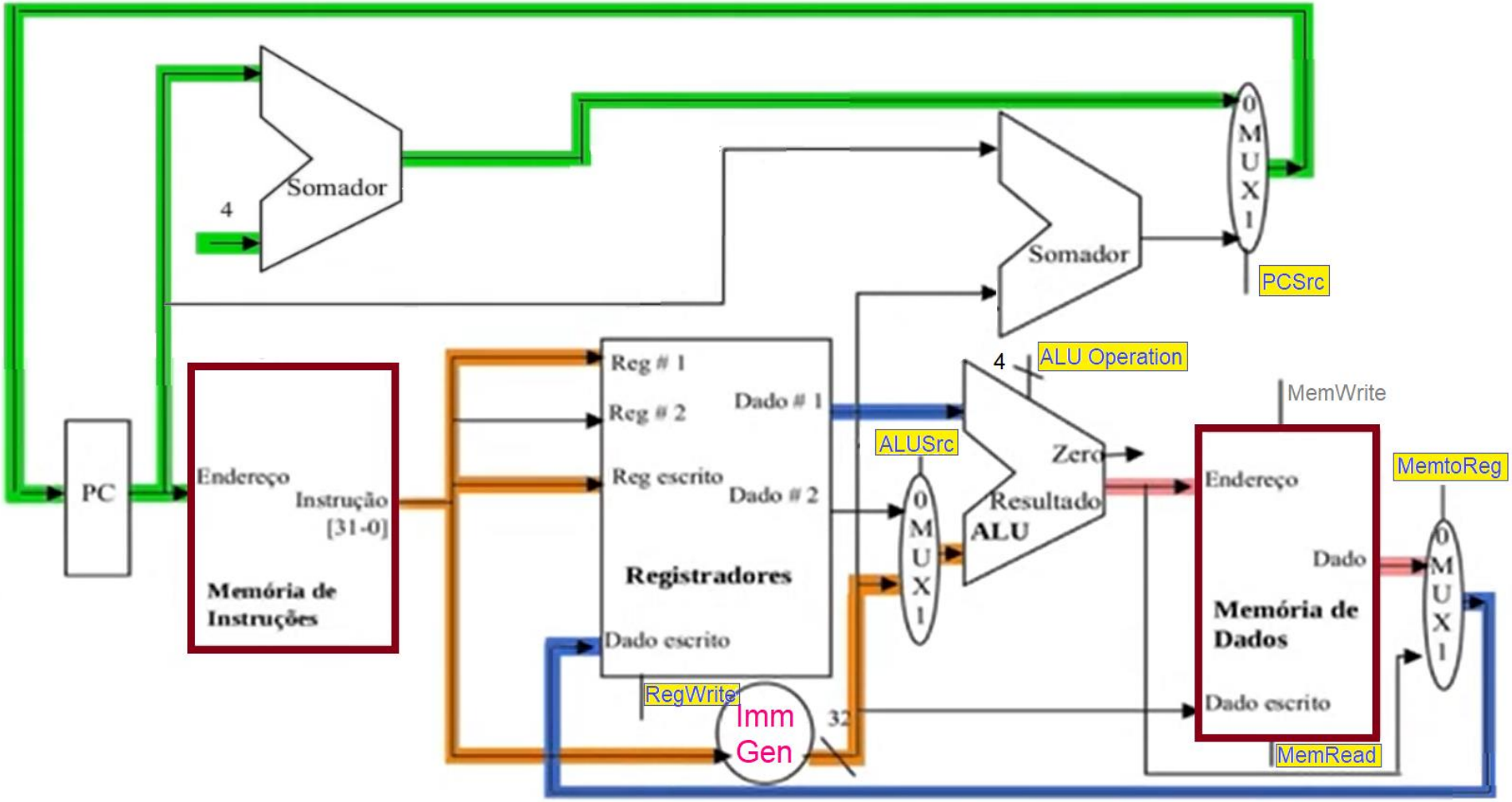
Tipo-R	*funct7*	*rs2*	*rs1*	*funct3*	*rd*	*opcode*



2000	10110001001101000011111100010101
2004	00000111010111011111000001011101
2008	00000110010100011111000000010111
2012	00000111010100011111000011111000
...	...
4000	10100111010100000111010101011100

8000	10110000000101000011111100010101
8004	00000111010100011111000000011100
8008	00000110010100011111000000010100
8012	00000111010100011111000000011000
...	...
9000	00100111010100000111000000011100

Caminho de dados: Instrução Tipo I

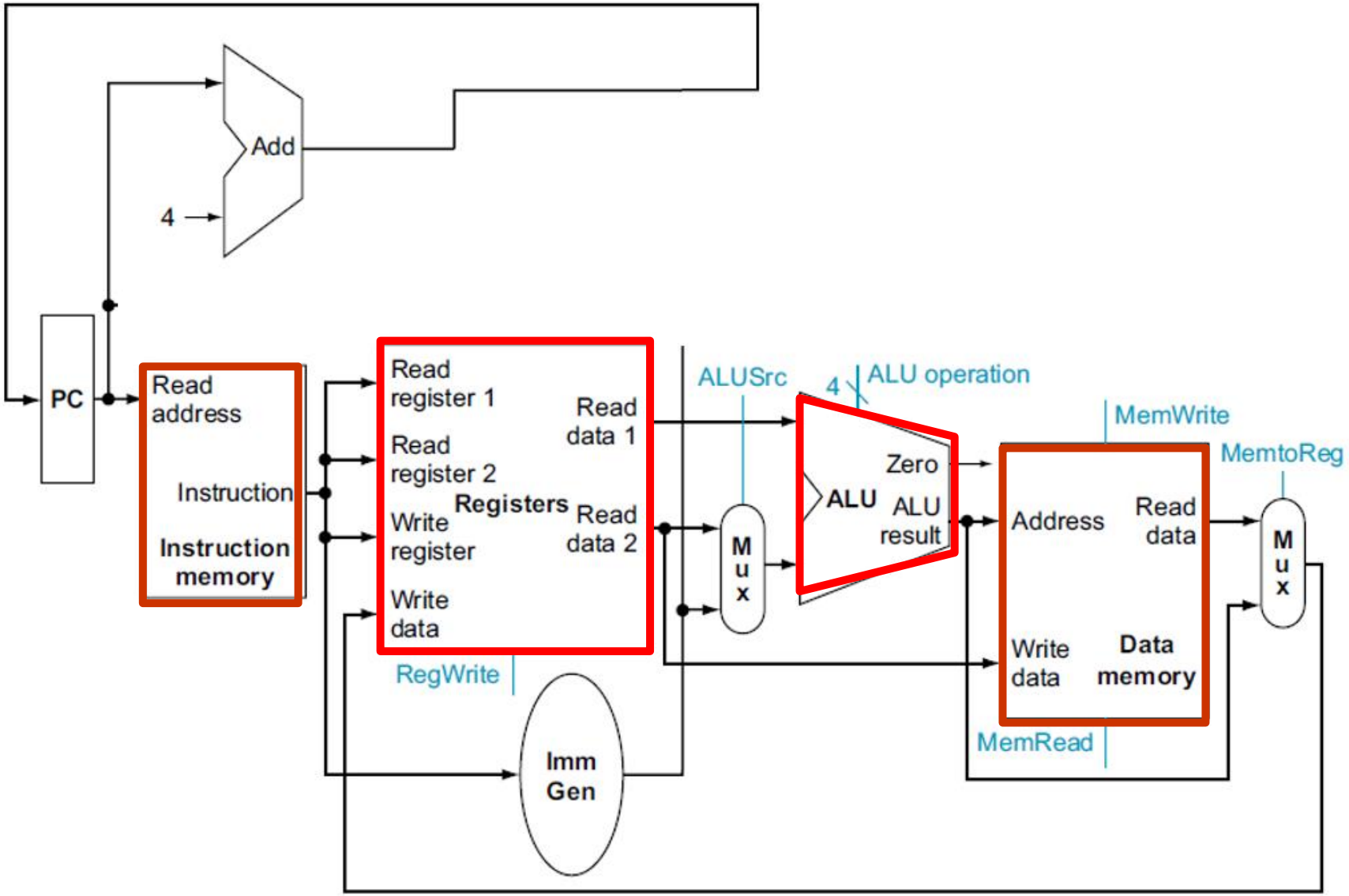


`lw rd, imm(rs1)` `//rd=Mem[rs1+ImmGen(imm)]`

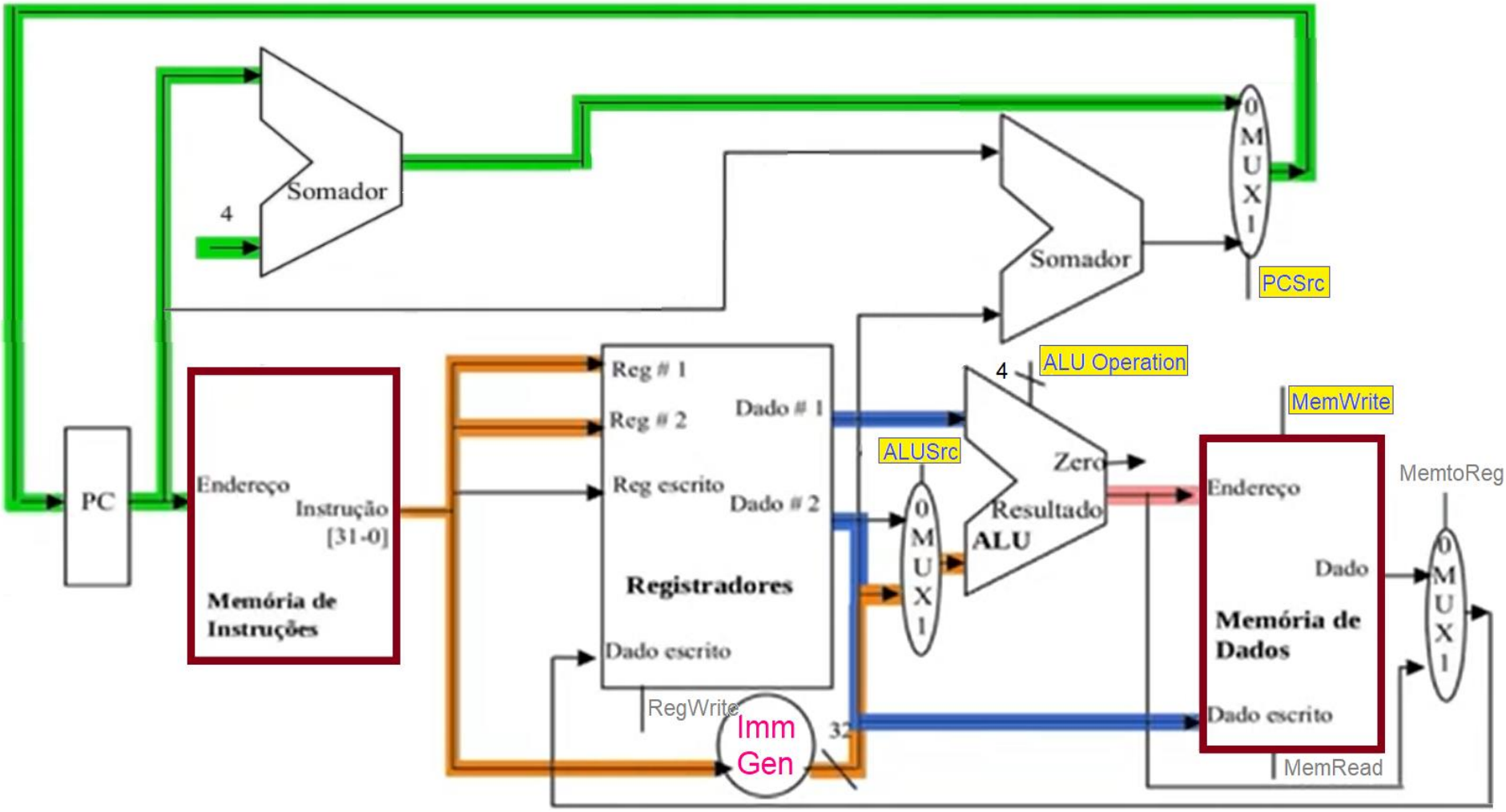
Sinais de Controle – Tipo I

Nome do sinal	Efeito quando desativado	Efeito quando ativado
RegWrite	Nenhum	É escrito em um registrador (*Write register)* o valor existente em *Write date*
ALUSrc	O segundo operando da ULA é oriundo do segundo registrador (*Read data 2*)	O segundo operando da ULA é a constante (imediato) estendido para 32 bits
PCSrc	PC é substituído pela saída do somador que calcula o valor de PC+4	PC é substituído pela saída do somador que calcula o alvo de um desvio (instruções de **Branch**)
MemRead	Nenhum	O conteúdo da **Memória de dados** armazenado no endereço de entrada (*Address*) é colocado na saída (*Read data*)
MemWrite	Nenhum	O conteúdo da **Memória de dados** armazenado no endereço de entrada (*Address*) é substituído pelo dado na entrada (*Write data*)
MemtoReg	O valor alimentado no **registrador** de entrada de dados (*Write data*) vem da **ULA**	O valor alimentado no **registrador** de entrada de dados (*Write data*) vem da **Memória de dados**

Tipo-I	*imm[11:0]*	*rs1*	*funct3*	*rd*	*opcode*



Caminho de dados: Instrução Tipo S

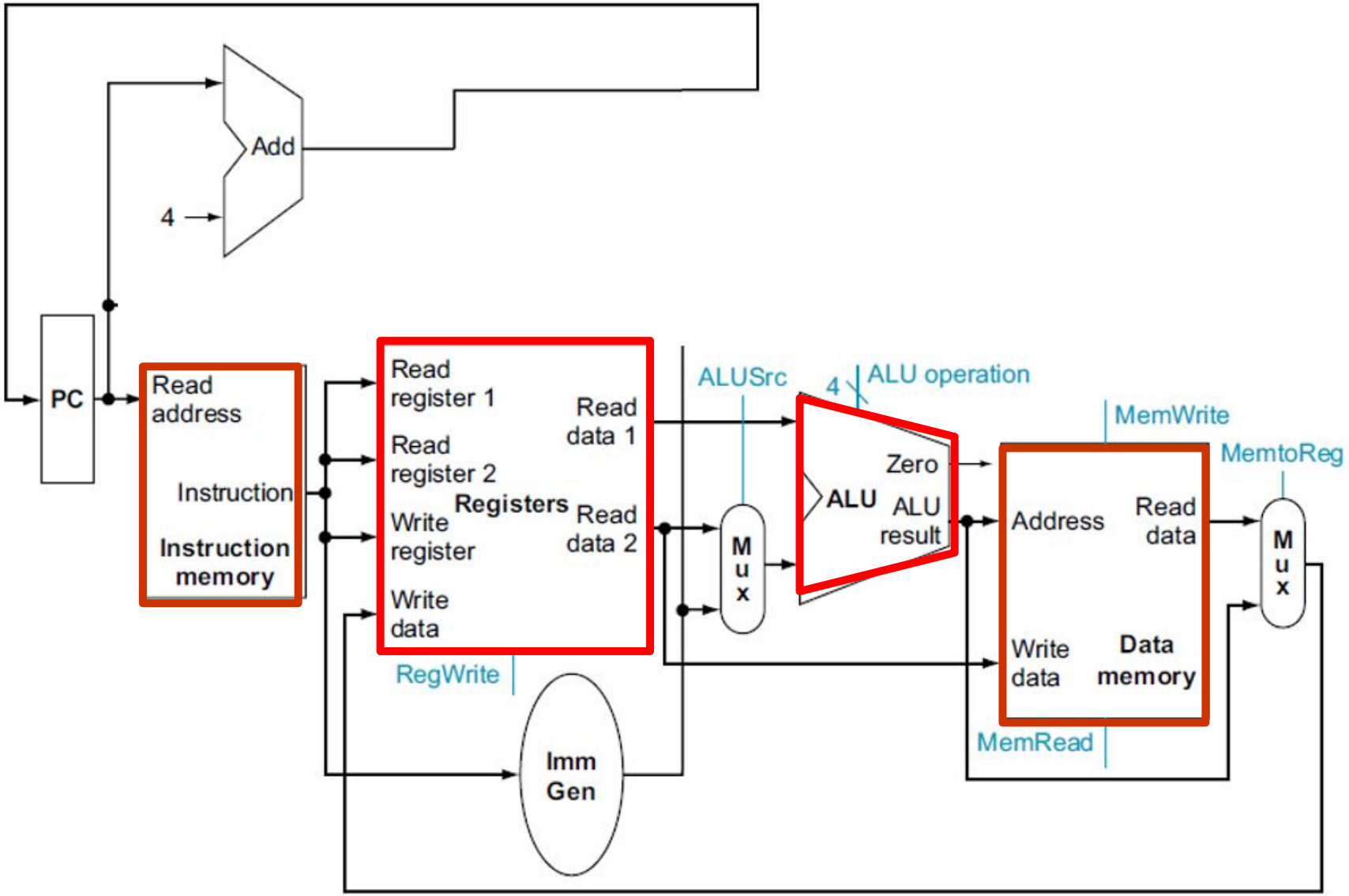


`sw rs2, imm(rs1)` `//Mem[rs1+ImmGen(imm)]=rs2`

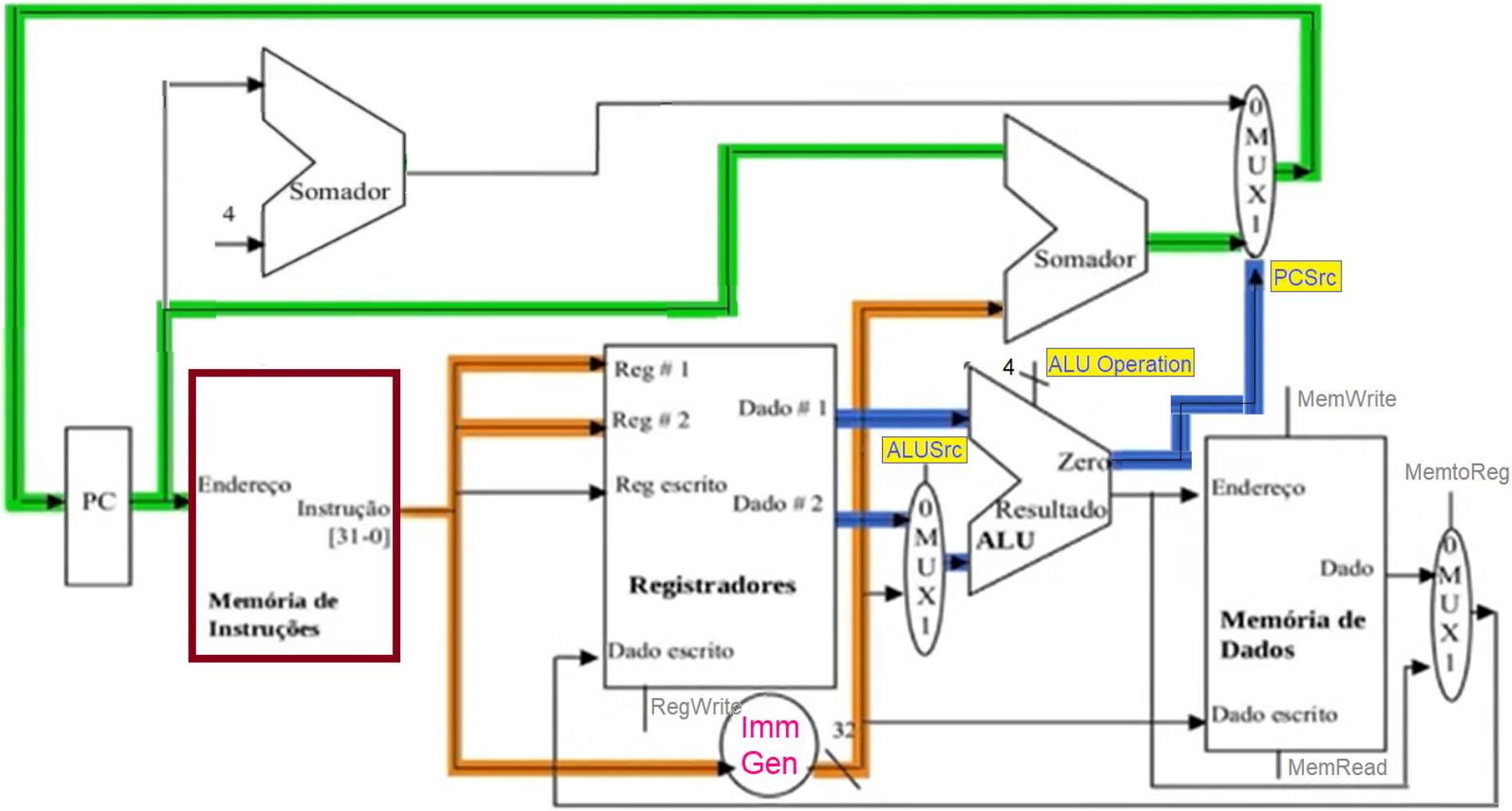
Sinais de Controle – Tipo S

Nome do sinal	Efeito quando desativado	Efeito quando ativado
RegWrite	Nenhum	É escrito em um registrador (*Write register)* o valor existente em *Write date*
ALUSrc	O segundo operando da ULA é oriundo do segundo registrador (*Read data 2*)	O segundo operando da ULA é a constante (imediato) estendido para 32 bits
PCSrc	PC é substituído pela saída do somador que calcula o valor de PC+4	PC é substituído pela saída do somador que calcula o alvo de um desvio (instruções de **Branch**)
MemRead	Nenhum	O conteúdo da **Memória de dados** armazenado no endereço de entrada (*Address*) é colocado na saída (*Read data*)
MemWrite	Nenhum	O conteúdo da **Memória de dados** armazenado no endereço de entrada (*Address*) é substituído pelo dado na entrada (*Write data*)
MemtoReg	O valor alimentado no **registrador** de entrada de dados (*Write data*) vem da **ULA**	O valor alimentado no **registrador** de entrada de dados (*Write data*) vem da **Memória de dados**

Tipo-S	*imm[11:5]*	*rs2*	*rs1*	*funct3*	*imm[4:0]*	*opcode*



Caminho de dados: Instrução Tipo B



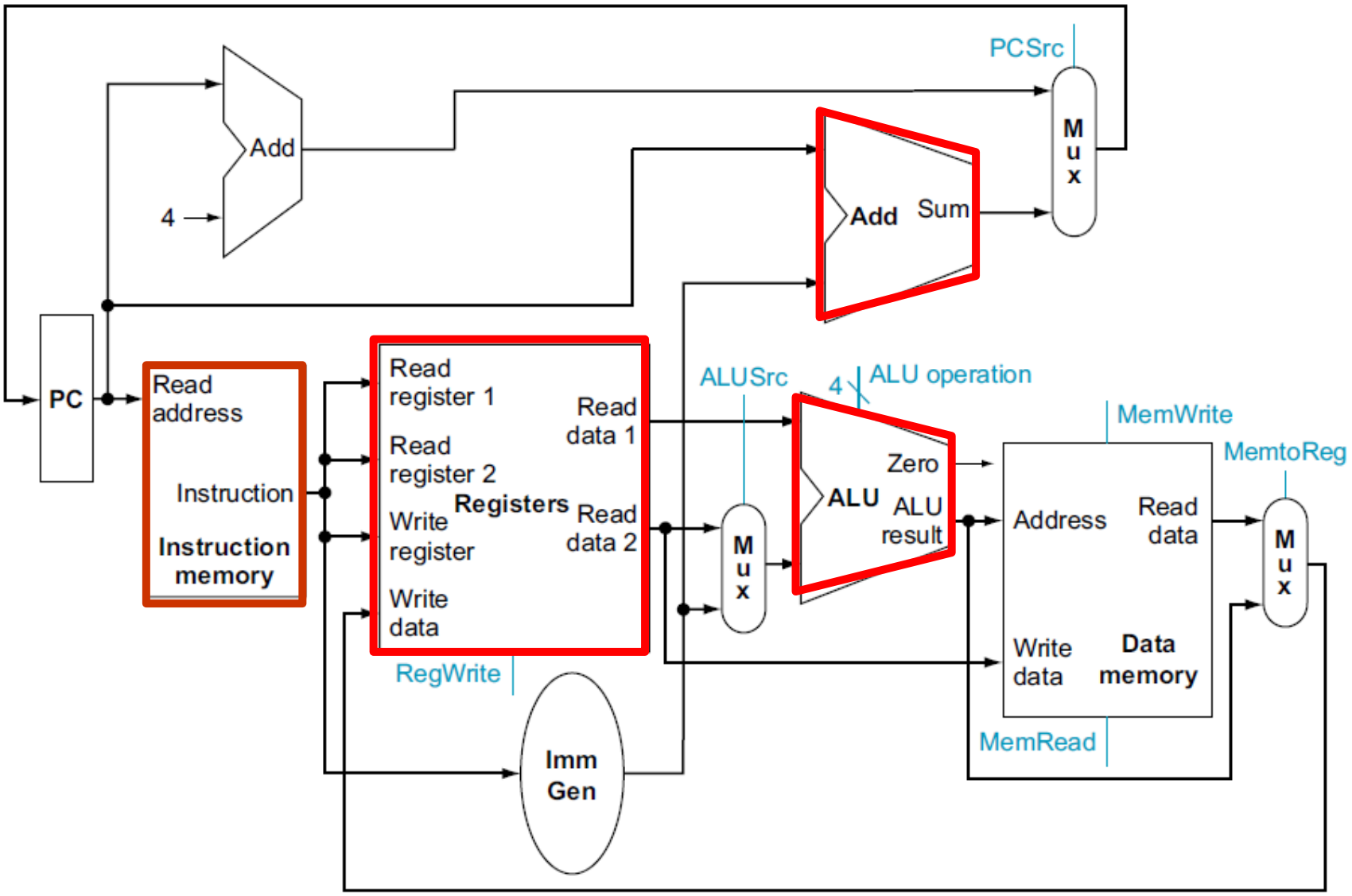
beq rs1, rs2, L

```
//if rs1==rs2   PC = PC+4 + ImmGen(imm)
```

Sinais de Controle – Tipo S

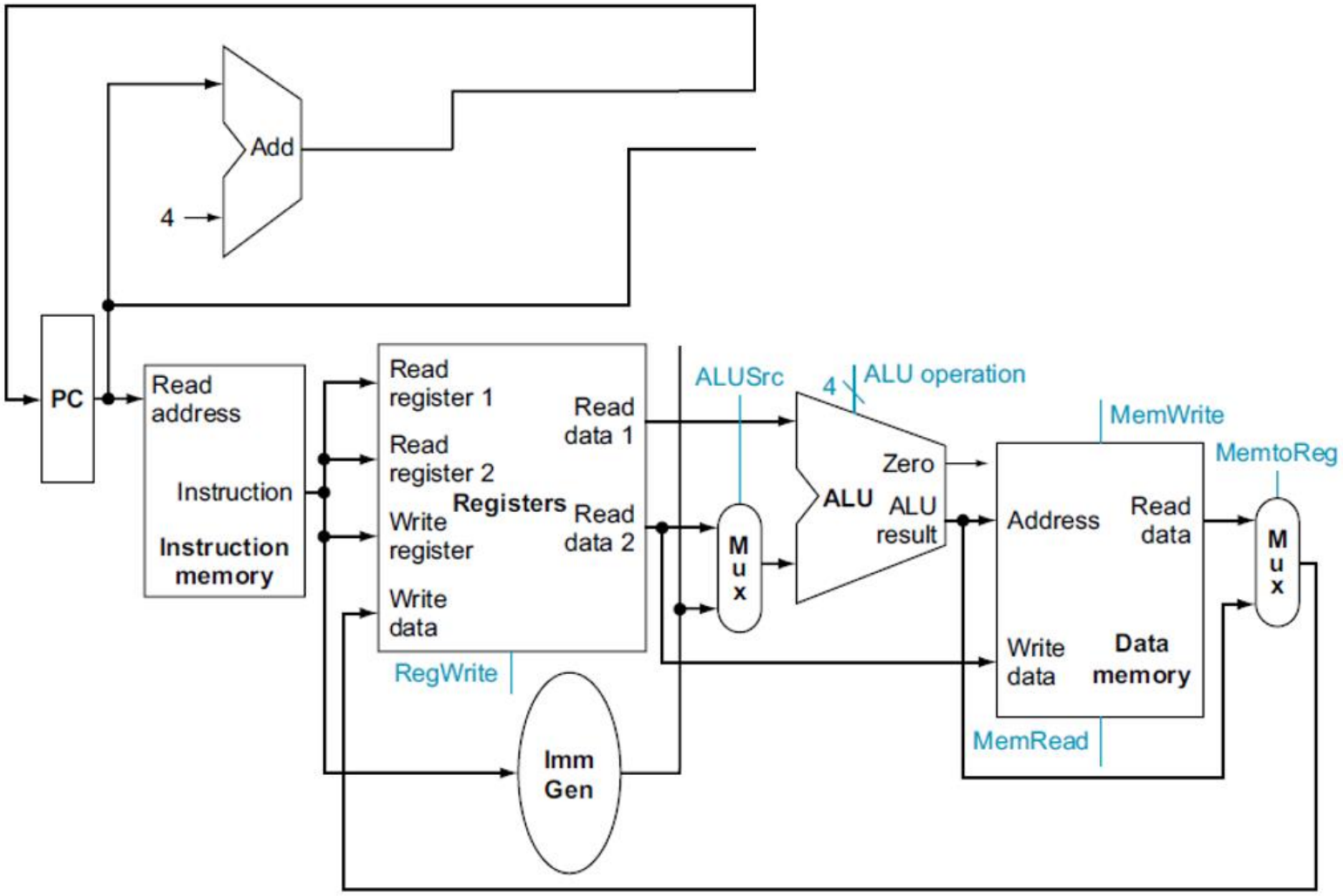
Nome do sinal	Efeito quando desativado	Efeito quando ativado
RegWrite	Nenhum	É escrito em um registrador (*Write register)* o valor existente em *Write date*
ALUSrc	O segundo operando da ULA é oriundo do segundo registrador (*Read data 2*)	O segundo operando da ULA é a constante (imediato) estendido para 32 bits
PCSrc	PC é substituído pela saída do somador que calcula o valor de PC+4	PC é substituído pela saída do somador que calcula o alvo de um desvio (instruções de **Branch**)
MemRead	Nenhum	O conteúdo da **Memória de dados** armazenado no endereço de entrada (*Address*) é colocado na saída (*Read data*)
MemWrite	Nenhum	O conteúdo da **Memória de dados** armazenado no endereço de entrada (*Address*) é substituído pelo dado na entrada (*Write data*)
MemtoReg	O valor alimentado no **registrador** de entrada de dados (*Write data*) vem da **ULA**	O valor alimentado no **registrador** de entrada de dados (*Write data*) vem da **Memória de dados**

Tipo-B	*imm[12]|imm[10:5]*	*rs2*	*rs1*	***funct3***	*imm[4:1]|imm[11]*	***opcode***

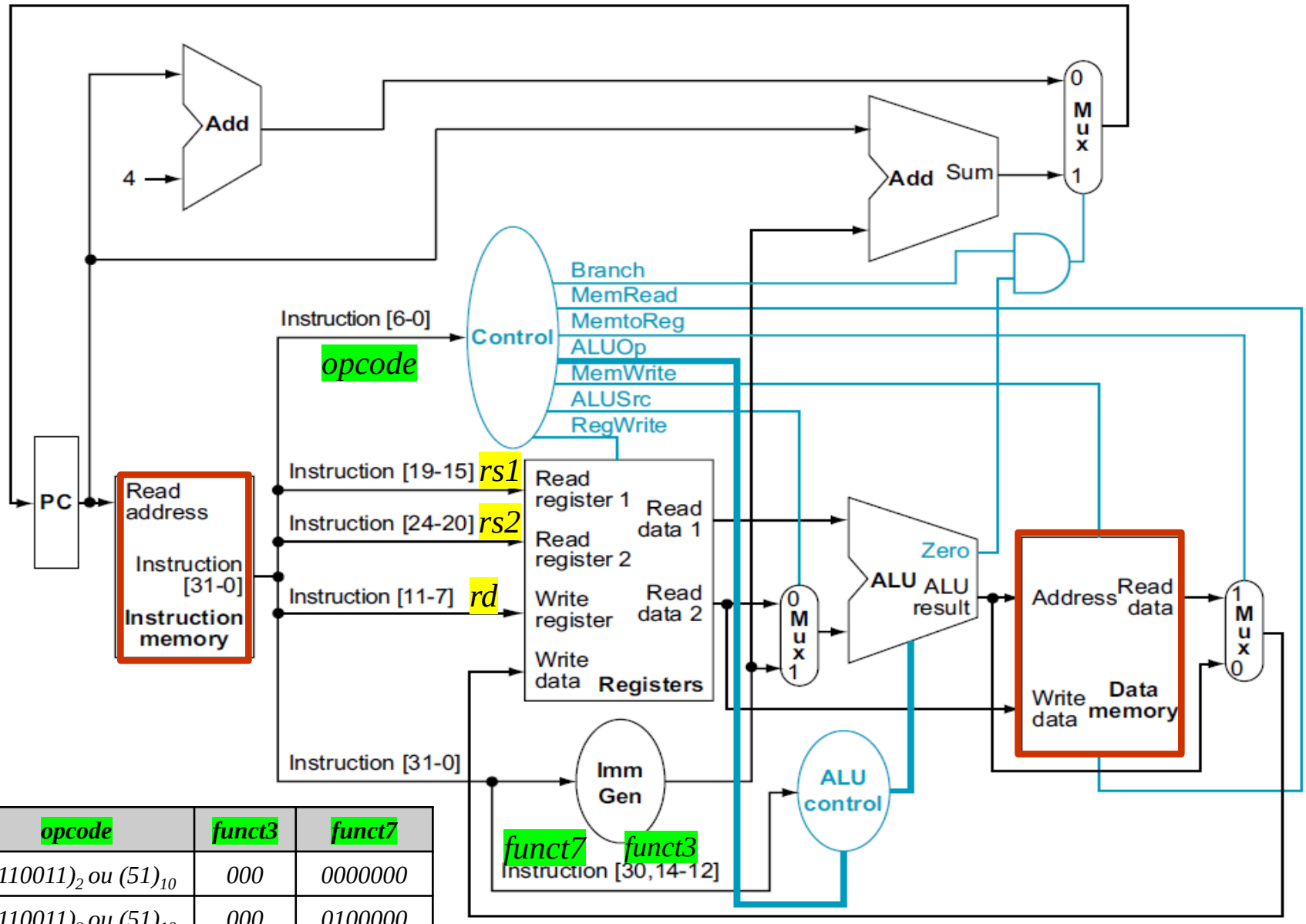


Caminho de dados
Monociclo

Via de Dados


Add
4
PC
Read address
Instruction
Instruction memory
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
RegWrite
ALUSrc
4
ALU operation
M u x
ALU
Zero
ALU result
MemWrite
MemtoReg
Address
Read data
Write data
Data memory
MemRead
M u x
Imm Gen

Tipo-R	*funct7*	*rs2*	*rs1*	*funct3*	*rd*	*opcode*



	opcode	*funct3*	*funct7*
add	$(0110011)_2$ *ou* $(51)_{10}$	*000*	*0000000*
sub	$(0110011)_2$ *ou* $(51)_{10}$	*000*	*0100000*

Name (Field size)	Field 7 bits	5 bits	5 bits	3 bits	5 bits	7 bits	Comments
R-type	funct7	rs2	rs1	funct3	rd	opcode	Arithmetic instruction format
I-type	immediate[11:0]		rs1	funct3	rd	opcode	Loads & immediate arithmetic
S-type	immed[11:5]	rs2	rs1	funct3	immed[4:0]	opcode	Stores
SB-type	immed[12,10:5]	rs2	rs1	funct3	immed[4:1,11]	opcode	Conditional branch format
UJ-type	immediate[20,10:1,11,19:12]				rd	opcode	Unconditional jump format
U-type	immediate[31:12]				rd	opcode	Upper immediate format

FIGURE 4.16 The actual RISC-V formats. Figure 4.16 introduces R-, I-, S-, and U-types, which are straightforward.

		Immediate Output Bit by Bit																													
		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Instruction	Format	Immediate Input Bit by Bit																															
Load, Arith. Imm.	I	i31	i31	i31	i31	i31	i31	i31	i31	i31	i31	i31	i31	i31	i31	i31	i31	i31	i31	i31	i31	i31	i30	i29	i28	i27	i26	i25	i24	i23	i22	i21	i20
Store	S	"	"	"	"	"	"	"	"	"	"	"	"	"	"	"	"	"	"	"	"	"	"	"	"	"	"	"	"	i11	i10	i9	i8	i7
Cond. Branch	S	"	"	"	"	"	"	"	"	"	"	"	"	"	"	"	"	"	"	"	"	i30	i29	i28	i27	i26	i25	i24	"	"	"	"	0
Uncond. Jump	U	"	"	"	"	"	"	"	"	"	"	"	"	i30	i29	i28	i27	i26	i25	i24	i23	i22	i21	i20	i19	i18	i17	i16	i15	i14	i13	i12	"
Load Upper Imm.	U	"	i30	i29	i28	i27	i26	i25	i24	i23	i22	i21	i20	i19	i18	i17	i16	i15	i14	i13	i12	0	0	0	0	0	0	0	0	0	0	0	"
Unique Inputs		1	2	2	2	2	2	2	2	2	2	2	2	3	3	3	3	3	3	3	3	4	4	4	4	4	4	4	4	4	4	4	3

FIGURE 4.17 Inputs to immediate if hypotheticaly conditional branches use the S format, and if jumps, use the U format.

		Immediate Output Bit by Bit																															
		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Instruction	Format	Immediate Input Bit by Bit																															
Load, Arith. Imm.	I	i31	i31	i31	i31	i31	i31	i31	i31	i31	i31	i31	i31	i31	i31	i31	i31	i31	i31	i31	i31	i31	i30	i29	i28	i27	i26	i25	i24	i23	i22	i21	i20
Store	S	"	"	"	"	"	"	"	"	"	"	"	"	"	"	"	"	"	"	"	"	"	"	"	"	"	"	"	"	"	i11	i10	i9	i8	i7
Cond. Branch	SB	"	"	"	"	"	"	"	"	"	"	"	"	"	"	"	"	"	"	"	"	"	i7	"	"	"	"	"	"	"	"	"	"	"	0
Uncond. Jump	UJ	"	"	"	"	"	"	"	"	"	"	"	"	i19	i18	i17	i16	i15	i14	i13	i12	i20	"	"	"	"	"	"	"	"	"	"	"	"	"
Load Upper Imm.	U	"	i30	i29	i28	i27	i26	i25	i24	i23	i22	i21	i20	"	"	"	"	"	"	"	"	0	0	0	0	0	0	0	0	0	0	0	"
Unique Inputs		1	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	4	2	2	2	2	2	2	3	3	3	3	3

FIGURE 4.18 Inputs to immediate given that branches use the SB format and jumps use the UJ format, which is what RISC-V uses.

Referências

- PATTERSON, David A; HENNESSY, John L; Computer Organization and Design – The hardware/software interface RISC-V edition; Elsevier – Morgan Kaufmann/ Amsterdam.
- PATTERSON, David; Waterman, Andrew; The RISC-V reader: an open architecture atlas; First edition. Berkeley, California: Strawberry Canyon LLC, 2017.