

A photograph of a server room with rows of black server racks. A semi-transparent blue rectangle is overlaid on the right side of the image, containing white text. The text reads 'Arquitetura do processador' in a large font, followed by '(Introdução)' in a smaller font. Below this, it says 'Cap. 4: “The Processor”'.

Arquitetura do processador

(Introdução)

Cap. 4: “The Processor”



Desenho do processador

Suporte para o conjunto de instruções

Desenho do processador

Suporte para o ISA do RISC-V (instruções de utilizador)

| Tipo de instruções | Mnemónicas |
|--------------------|--|
| Aritméticas (INT) | add, addi, sub, mul, mulh, mulsu, mulu, div, divu, rem, remu |
| Aritméticas (FP) | fsgnj_, fmin, fmax, fadd, fsub, fmul, fdiv, fsqrt, fmadd, fmsub, fnmadd, fnsub |
| Convert (INT↔FP) | fcvt, fmv |
| Lógicas | and, andi, or, ori, xor, xori |
| Deslocamento | sll, slli, srl, srli, sra, srai |
| Transferência | lui, auipc |
| Acesso à memória | lb, lbu, lh, lhu, lw, sb, sh, sw, flw, fld, fsw, fsd |
| Comparação | slt, slti, sltu, sltiu, feq, flt, fle, fclass |
| Controlo de fluxo | beq, bne, bge, bgeu, blt, bltu, jal, jalr |

Desenho do processador

Suporte para o ISA do RISC-V

| Tipo de instruções | Mnemónicas |
|----------------------|--|
| Aritméticas (INT) | add, addi, sub, mul, mulh, mulsu, mulu, div, divu, rem, remu |
| Aritméticas (FP) | fsgnj_, fmin, fmax, fadd, fsub, fmul, fdiv, fsqrt, fmadd, fmsub, fnmadd, fnsub |
| Conversão (INT ↔ FP) | fcvt, fmv |
| Lógicas | and, andi, or, ori, xor, xori |
| Deslocamento | sll, slli, srl, srli, sra, srai |
| Transferência | lui, auipc |
| Acesso à memória | lb, lbu, lh, lhu, lw, sb, sh, sw, flw, fld, fsw, fld |
| Comparação | slt, slti, sltu, sltiu, feq, flt, fle, fclass |
| Controlo de fluxo | beq, bne, bge, bgeu, blt, bltu, jal, jalr |

Aritméticas

Lógicas

Deslocamento

Comparação (INT)



RV32: 32-bit ALU

RV64: 64-bit ALU

Aritméticas

Conversão (INT ↔ FP)

Comparação (FP)



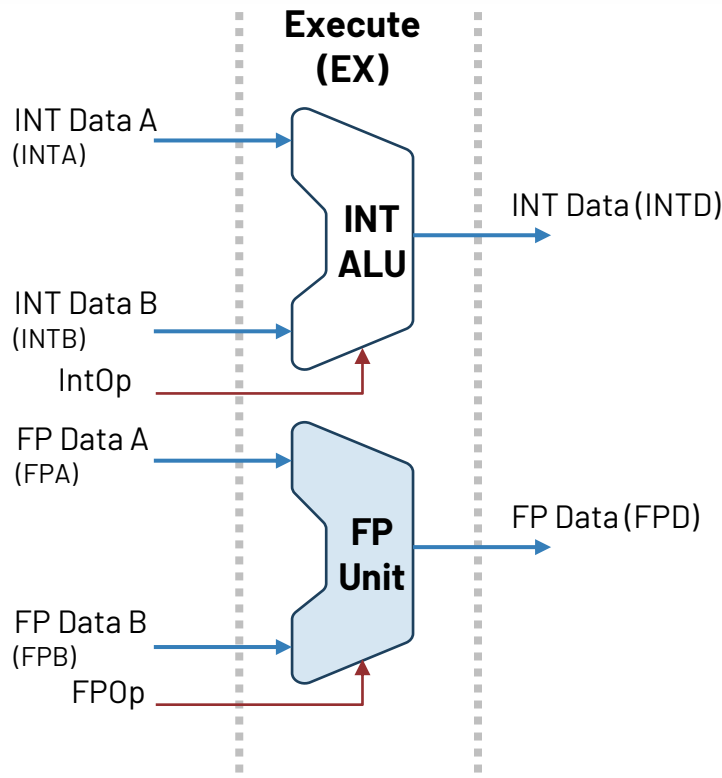
"F" Extension: float

"D" Extension: double



Desenho do processador

Suporte para o ISA do RISC-V

| Sinal | Significado |
|--------------|--|
| INT Data A/B | Entrada de dados (inteiro) A e B |
| INTD | Resultado produzido pela ALU de inteiros |
| IntOp | Seleção da operação a realizar na ALU de INT |
| FP Data A/B | Entrada de dados (FP) A e B |
| FPD | Resultado produzido pela ALU de FP |
| FPOp | Seleção da operação a realizar na ALU de FP |



Legenda:

-  Sinal de dados
-  Sinal de controlo (gerado através da decodificação da instrução)

A definição concreta das ALUs será explicada mais à frente

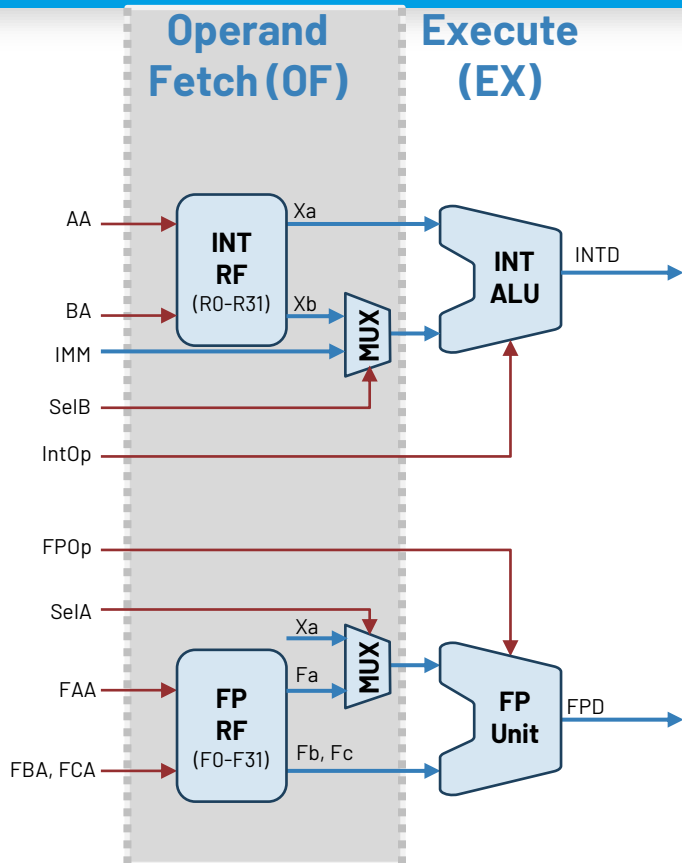
Desenho do processador

Circuito de dados (datapath)

Exemplos:

add x5, x3, x0

addi x7, x1, -4



Legenda:

- > Sinal de dados
- > Sinal de controlo (gerado através da decodificação da instrução)

| Sinal | Significado |
|--------|-----------------------------|
| AA,FAA | Source register A Address |
| BA,FBA | Source register B Address |
| FCA | Source register C Address |
| IMM | Imediato |
| SelB | SElect input B: INTB ou IMM |

Desenho do processador

Suporte para o ISA do RISC-V

| Tipo de instruções | Mnemónicas |
|--------------------|--|
| Aritméticas (INT) | add, addi, sub, mul, mulh, mulsu, mulu, div, divu, rem, remu |
| Aritméticas (FP) | fsgnj_, fmin, fmax, fadd, fsub, fmul, fdiv, fsqrt, fmadd, fmsub, fnmadd, fnsub |
| Convert (INT↔FP) | fcvt, fmv |
| Lógicas | and, andi, or, ori, xor, xori |
| Deslocamento | sll, slli, srl, srli, sra, srai |
| Transferência | lui, auipc |
| Acesso à memória | lb, lbu, lh, lhu, lw, sb, sh, sw, flw, fld, fs |
| Comparação | slt, slti, sltu, sltiu, feq, flt, fle, fclass |
| Controlo de fluxo | beq, bne, bge, bgeu, blt, bltu, jal, jalr |

Requer :

- Os operandos das instruções podem ser um registo ou um imediato codificado diretamente a partir da palavra de instrução.
- Há dois bancos de registos: INT (X0-X31) e FP (F0-F31)

Desenho do processador

Circuito de dados (datapath)

Exemplos:

add x5, x3, x0

addi x7, x1, -4

Nota :

O banco de registos de FP tem três entradas:

FBA, FBB e FBC

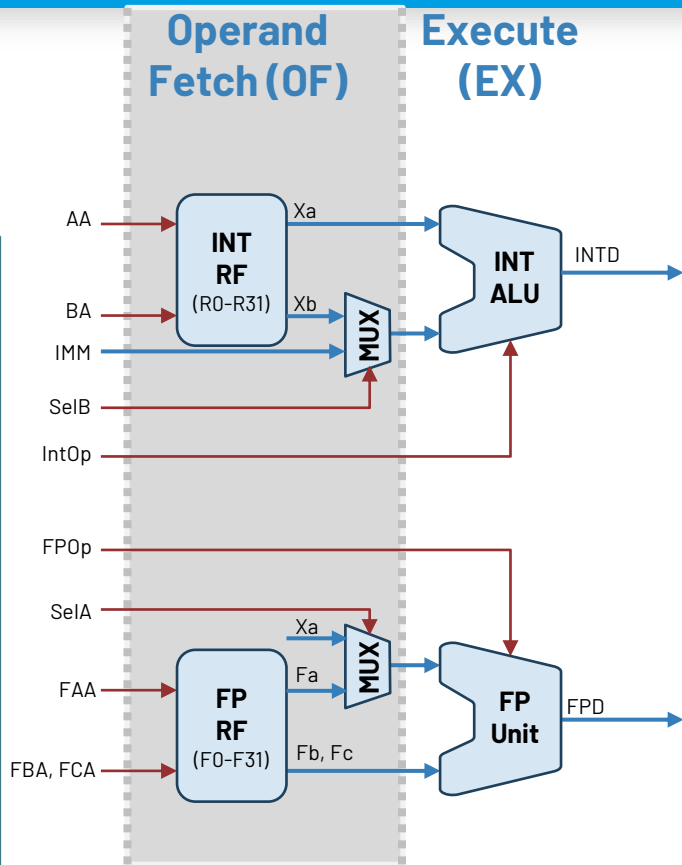
que servem para indicar os três registos lidos

Fa, Fb e Fc

O registo Fc é apenas usado nas operações de *fused multiply add/subtract*, e.g.

FMADD.d fd,fa,fb,fc

$(fd \leftarrow fa \times fd + fc)$



Legenda:

- Sinal de dados
- Sinal de controlo (gerado através da decodificação da instrução)

| Sinal | Significado |
|--------|-----------------------------|
| AA,FAA | Source register A Address |
| BA,FBA | Source register B Address |
| FCA | Source register C Address |
| IMM | Imediato |
| SelB | SElect input B: INTB ou IMM |

Desenho do processador

Suporte para o ISA do RISC-V

| Tipo de instruções | Mnemónicas |
|---------------------------|--|
| Aritméticas (INT) | add, addi, sub, mul, mulh, mulsu, mulu, div, divu, rem, remu |
| Aritméticas (FP) | fsgnj_, fmin, fmax, fadd, fsub, fmul, fdiv, fsqrt, fmadd, fmsub, fnmadd, fnsub |
| Convert (INT ↔ FP) | fcvt, fmv |
| Lógicas | and, andi, or, ori, xor, xori |
| Deslocamento | sll, slli, srl, srli, sra, srai |
| Transferência | lui, auipc |
| Acesso à memória | lb, lbu, lh, lhu, lw, sb, sh, sw, flw, fld, fs |
| Comparação | slt, slti, sltu, sltiu, feq, flt, fle, fclass |
| Controlo de fluxo | beq, bne, bge, bgeu, blt, bltu, jal, jalr |

Convert :

- A conversão é efetuada na unidade de FP. Esta tem de conseguir receber um valor inteiro.

Desenho do processador

Circuito de dados (datapath)

Legenda:

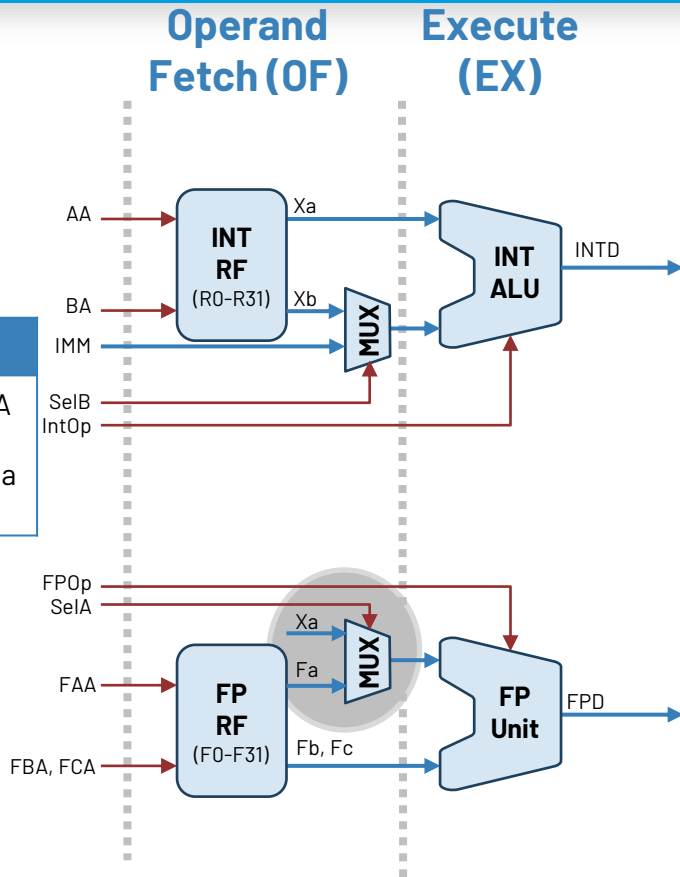
- Sinal de dados
- Sinal de controlo
(gerado através
da decodificação
da instrução)

| Sinal | Significado |
|-------|--|
| SelA | Escolhe o operando A da unidade de FP. Escolhe Xa se for uma conversão INT→FP |

Exemplos:

lw x5, 4(x3)

fsd f7, 8(x1)



[illegible]

Desenho do processador

Circuito de dados (datapath)

Legenda:

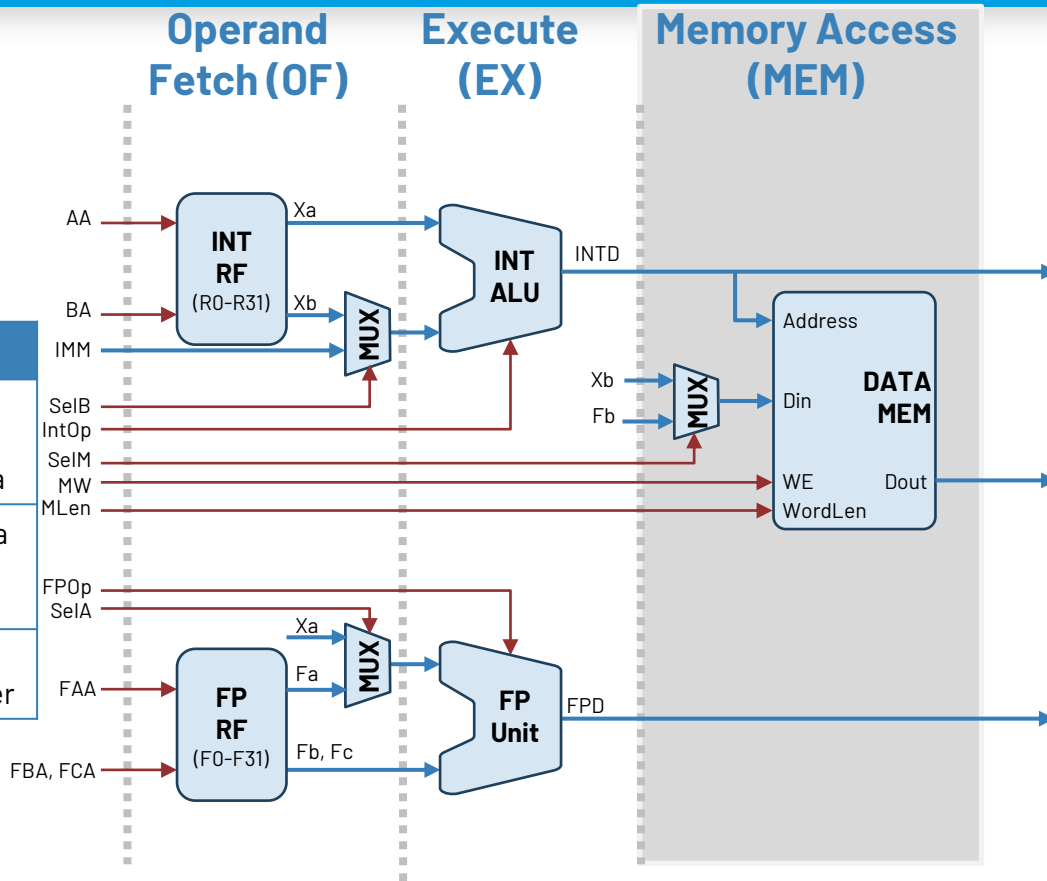
- > Sinal de dados
- > Sinal de controlo
(gerado através
da decodificação
da instrução)

| Sinal | Significado |
|---------|---|
| SeIM | Select Memory Data: seleciona o valor a escrever na memória |
| MW | Memory Write: Indica uma escrita na memória |
| WordLen | Indica o tamanho da palavra a ler/escrever |

Exemplos:

lw x5, 4(x3)

fsw f7, 8(x1)



Desenho do processador

Circuito de dados (datapath)

Legenda:

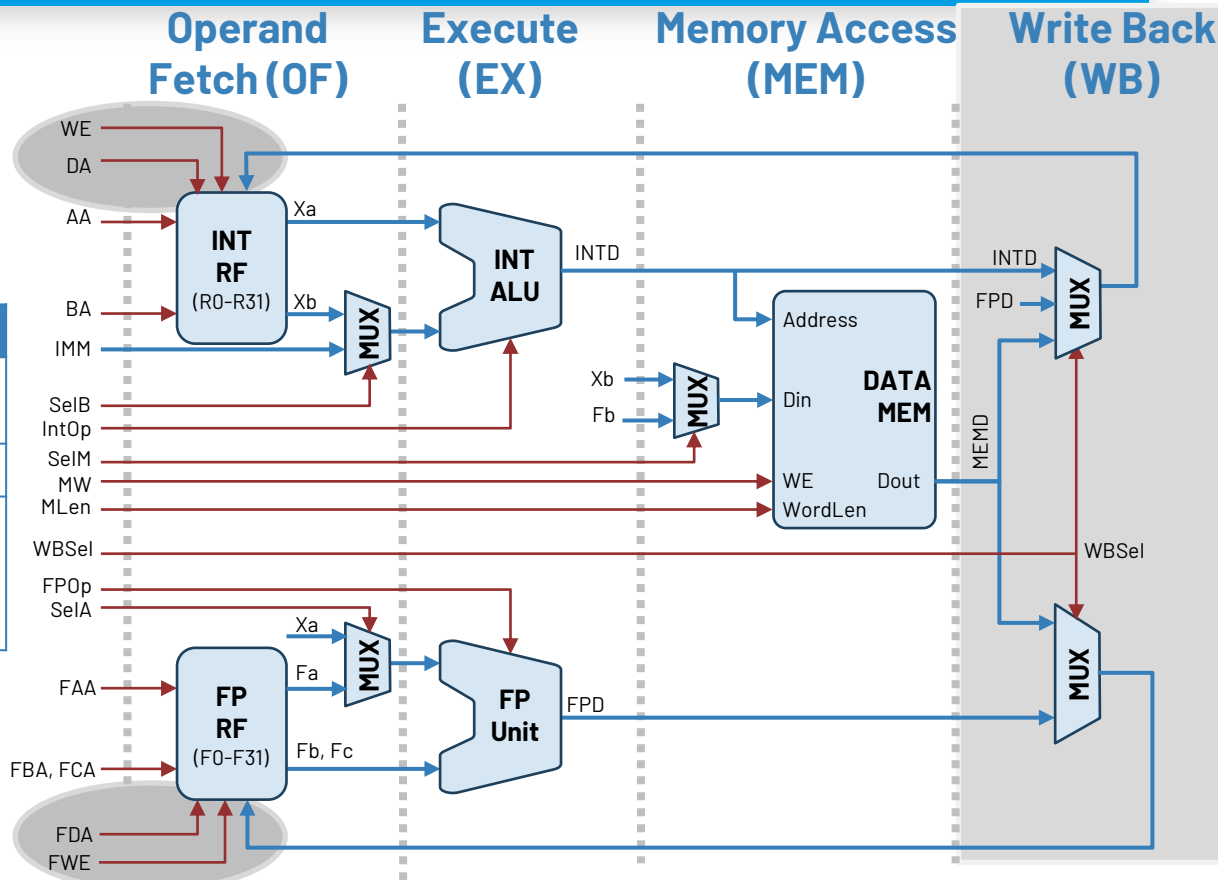
- Sinal de dados
- Sinal de controlo (gerado através da decodificação da instrução)

| Sinal | Significado |
|--------|---|
| DA,FDA | Destination register Address |
| WE,FWE | Write Enable |
| WBSel | Seleção do valor a escrever no registo (proveniente da ALU ou da memória) |

Exemplos:

lw x5, 4(x3)

fsw f7, 8(x1)



Desenho do processador

Circuito de dados (datapath)

Legenda:

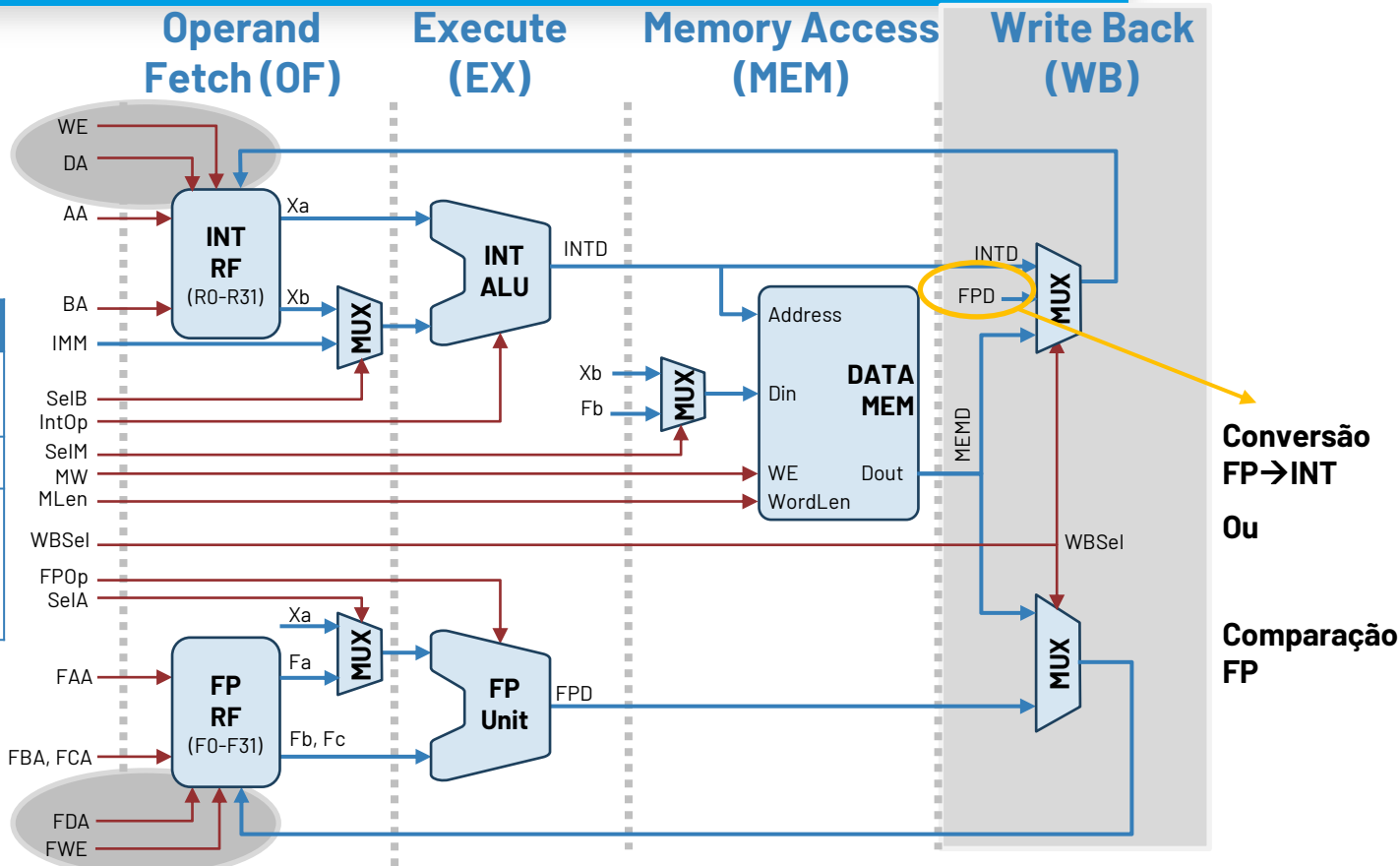
- Sinal de dados
- Sinal de controlo (gerado através da decodificação da instrução)

| Sinal | Significado |
|--------|---|
| DA,FDA | Destination register Address |
| WE,FWE | Write Enable |
| WBSel | Seleção do valor a escrever no registo (proveniente da ALU ou da memória) |

Exemplos:

lw x5, 4(x3)

fsd f7, 8(x1)



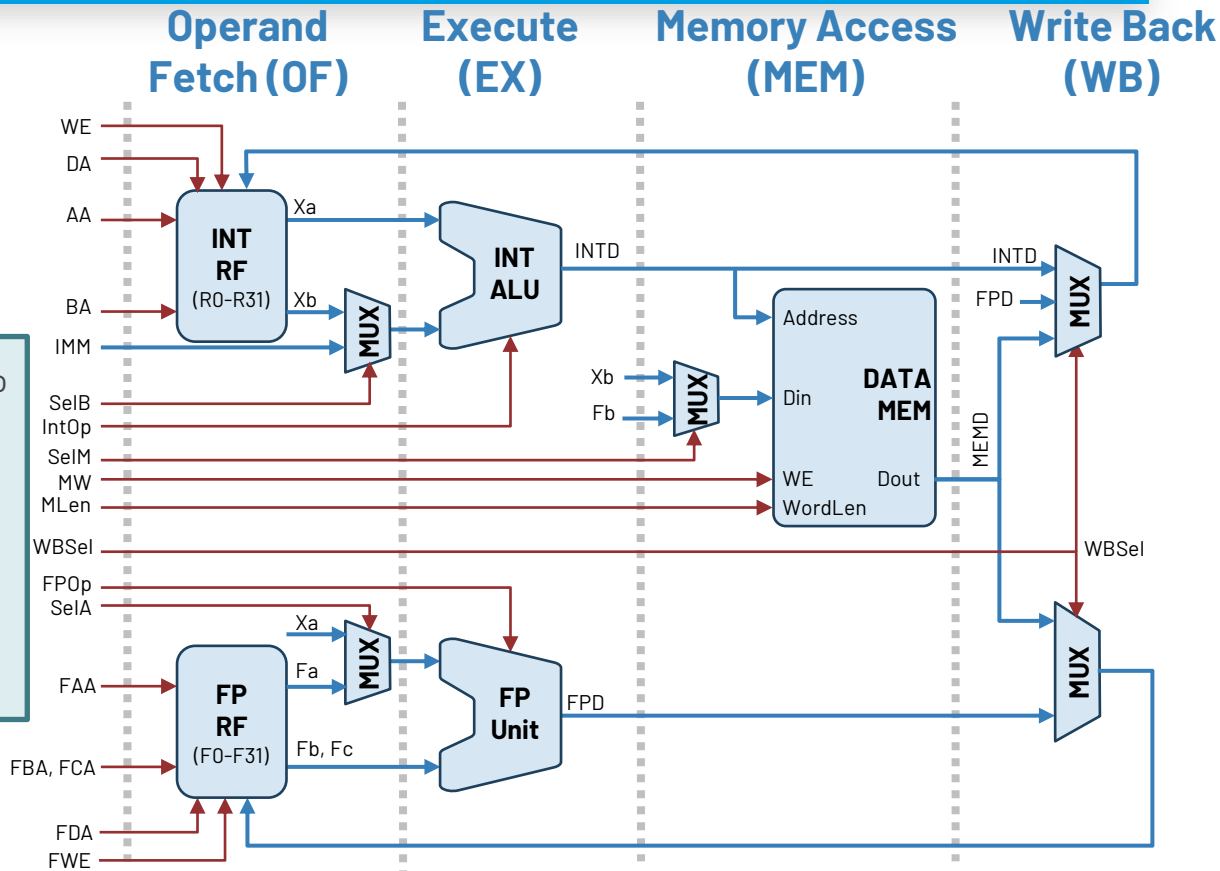
Desenho do processador

Circuito de dados (datapath)

Legenda:

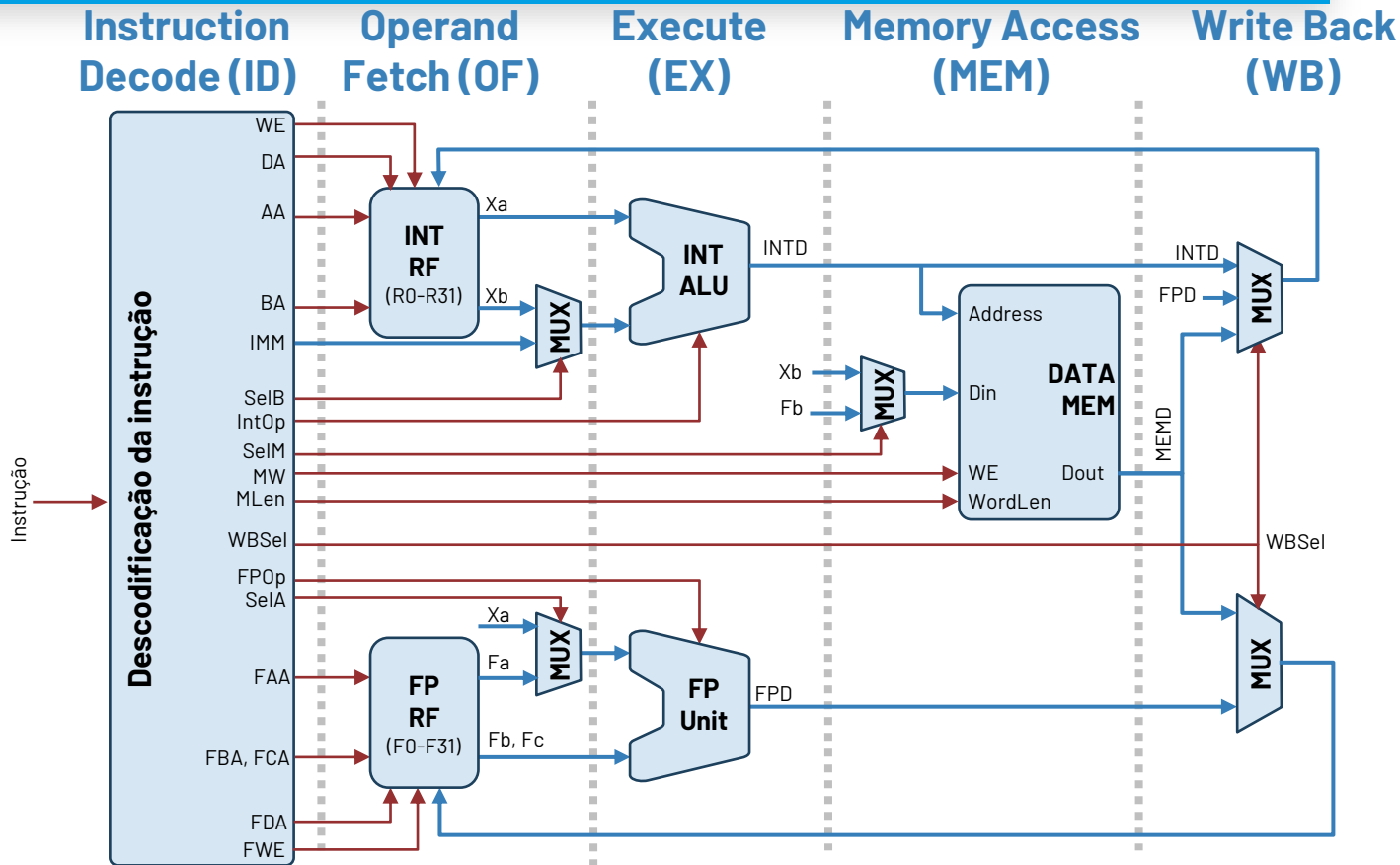
- Sinal de dados
- Sinal de controlo (gerado através da decodificação da instrução)

Para garantir o funcionamento “regular” do processador, é ainda necessário definir o circuito de controlo, nomeadamente a leitura da instrução da memória, a implementação do PC e as instruções de controlo de fluxo.



Desenho do processador

Sinais de controlo e descodificação da instrução



Desenho do processador

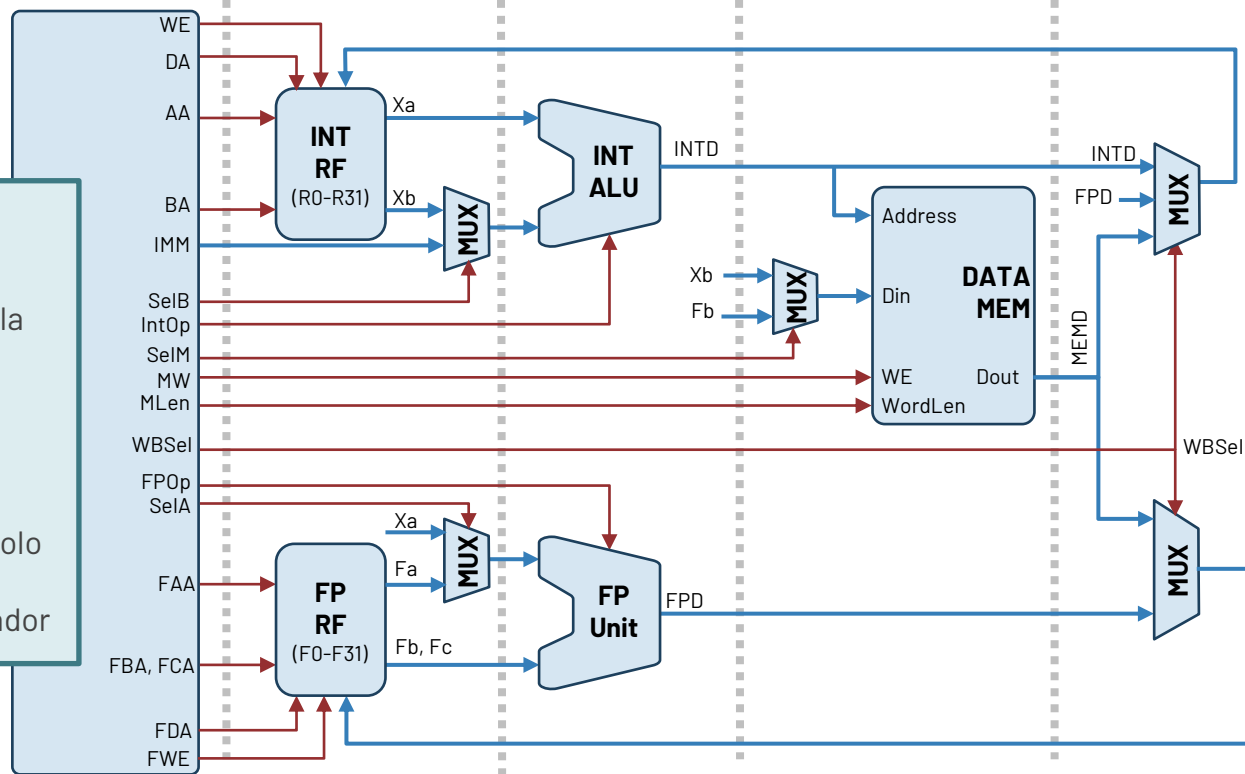
Sinais de controlo e descodificação da instrução

Instruction Decode (ID) Operand Fetch (OF) Execute (EX) Memory Access (MEM) Write Back (WB)

Palavra de controlo:

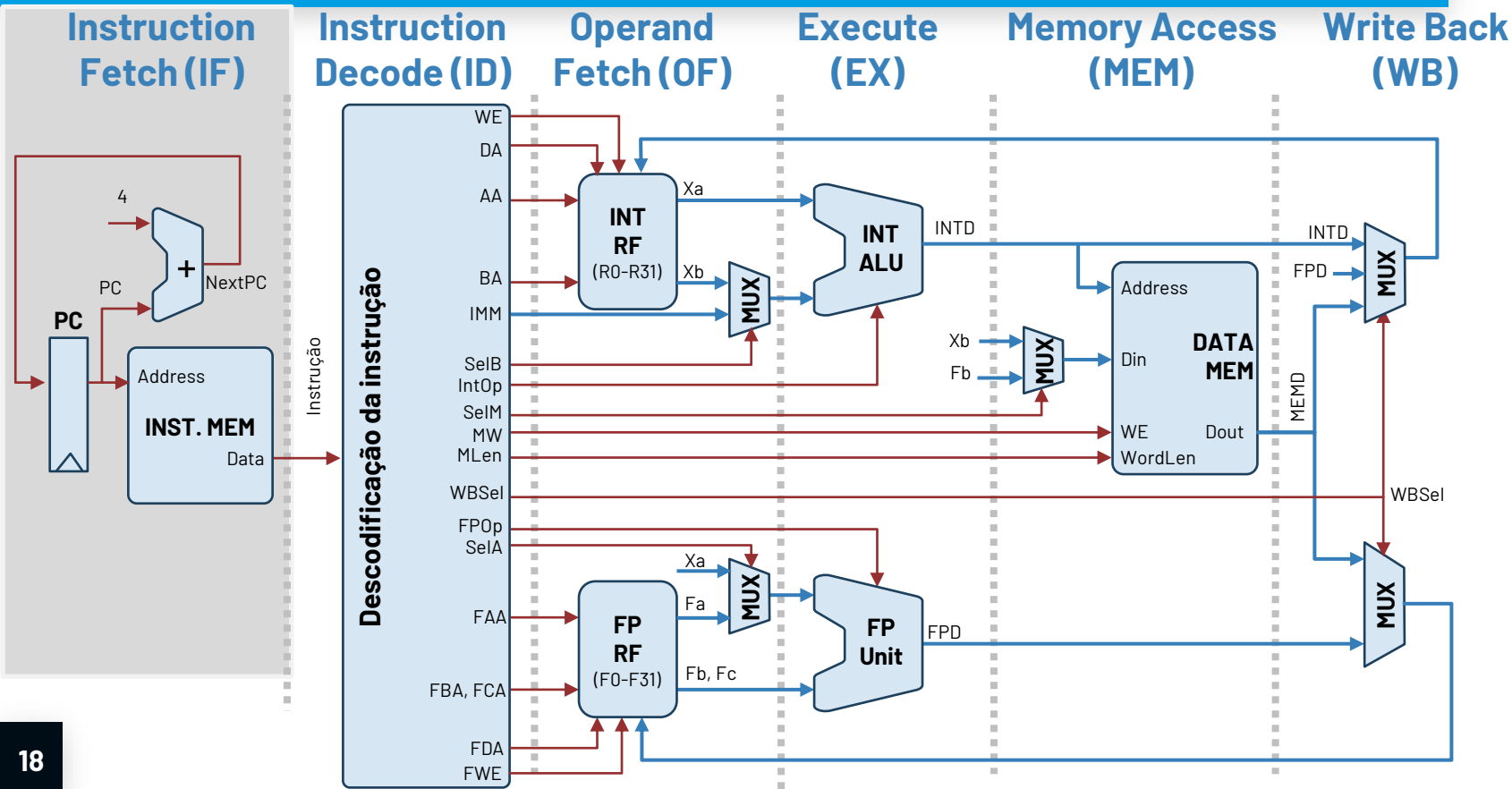
conjunto de sinais de controlo produzidos pela descodificação da instrução

Na prática a *Palavra de Controlo* realiza o controlo da execução das instruções no processador



Desenho do processador

Sinais de controlo e descodificação da instrução



Desenho do processador

Suporte para o ISA do RISC-V

Tipo de instruções Mnemónicas

Aritméticas (INT)

Aritméticas (FP)

Convert (INT ↔ FP)

Lógicas

Deslocamento

Transferência

Acesso à memória

Comparação

Controlo de fluxo

Requer :

- Forma de carregar o PC com um valor diferente de PC+4
- Forma de fazer depender o carregamento do PC através da condição
- Forma de salvaguardar o endereço de retorno (*Link*), correspondente a PC+4

slr, slri, srl, srli, sra, srai

lui, auipc

lb, lbu, lh, lhu, lw, sb, sh, sw, flw, fld, fsw, fsd

slt, slti, sltu, sltiu, feq, flt, fle, fclass

beq, bne, bge, bgeu, blt, bltu, jal, jalr

Instruction
Fetch (IF)

Instruction
Decode (ID)

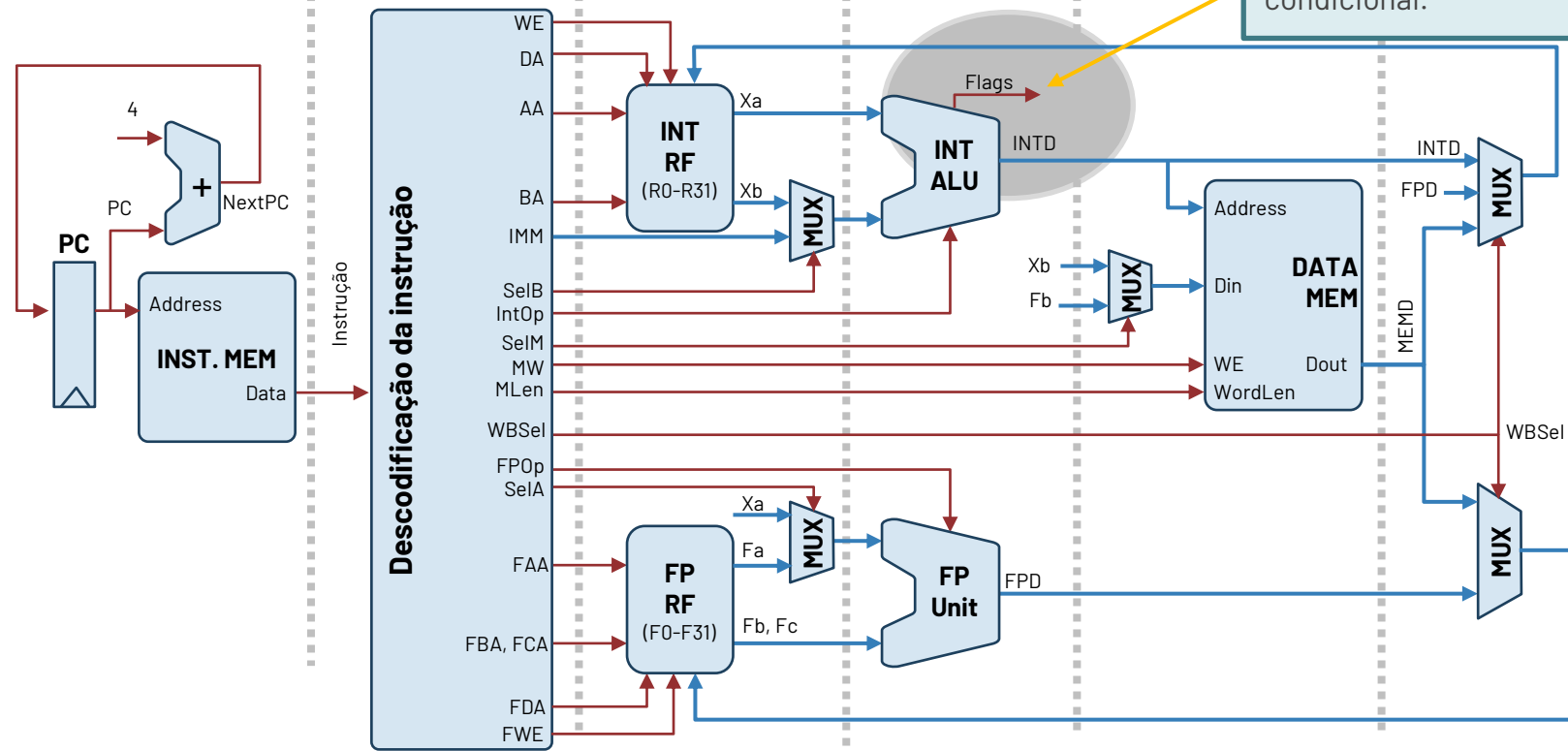
Operand
Fetch (OF)

Execute
(EX)

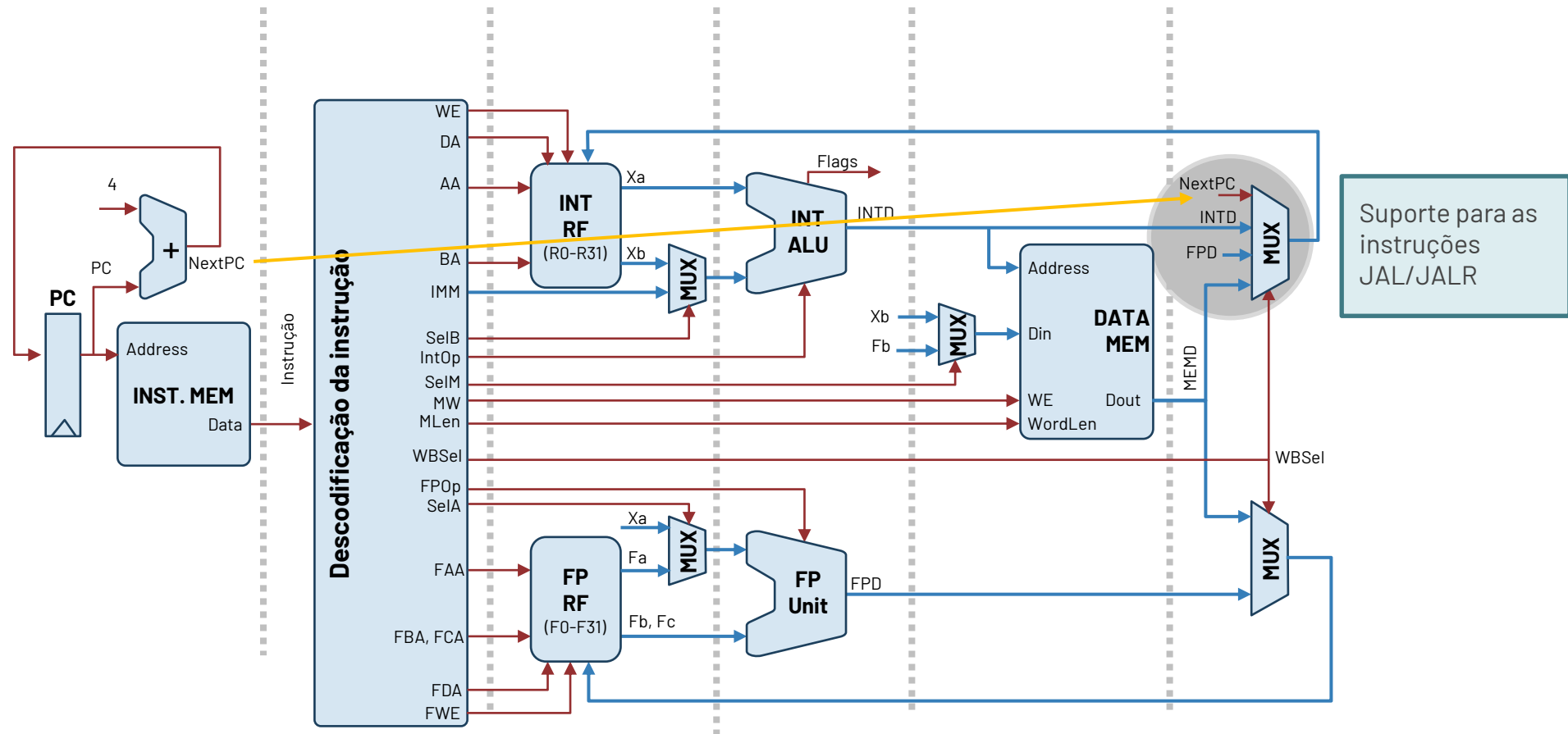
Memory Access
(MEM)

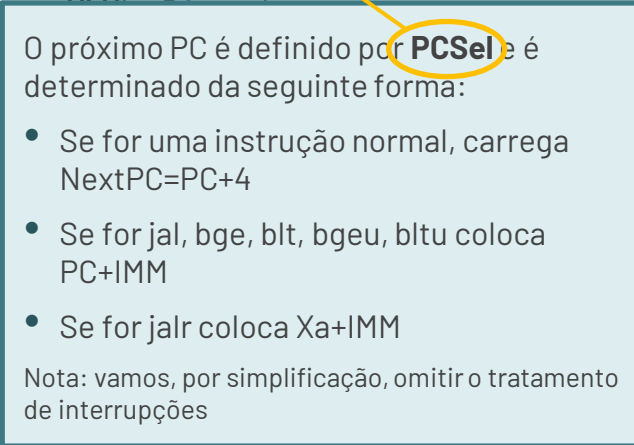
Write Back
(WB)

As **Flags** (e.g., ZCNV) são importantes para implementação das operações de salto condicional.



| Instruction Fetch (IF) | Instruction Decode (ID) | Operand Fetch (OF) | Execute (EX) | Memory Access (MEM) | Write Back (WB) |
|---------------------------|----------------------------|-----------------------|-----------------|------------------------|--------------------|
|---------------------------|----------------------------|-----------------------|-----------------|------------------------|--------------------|





Instruction
Fetch (IF)

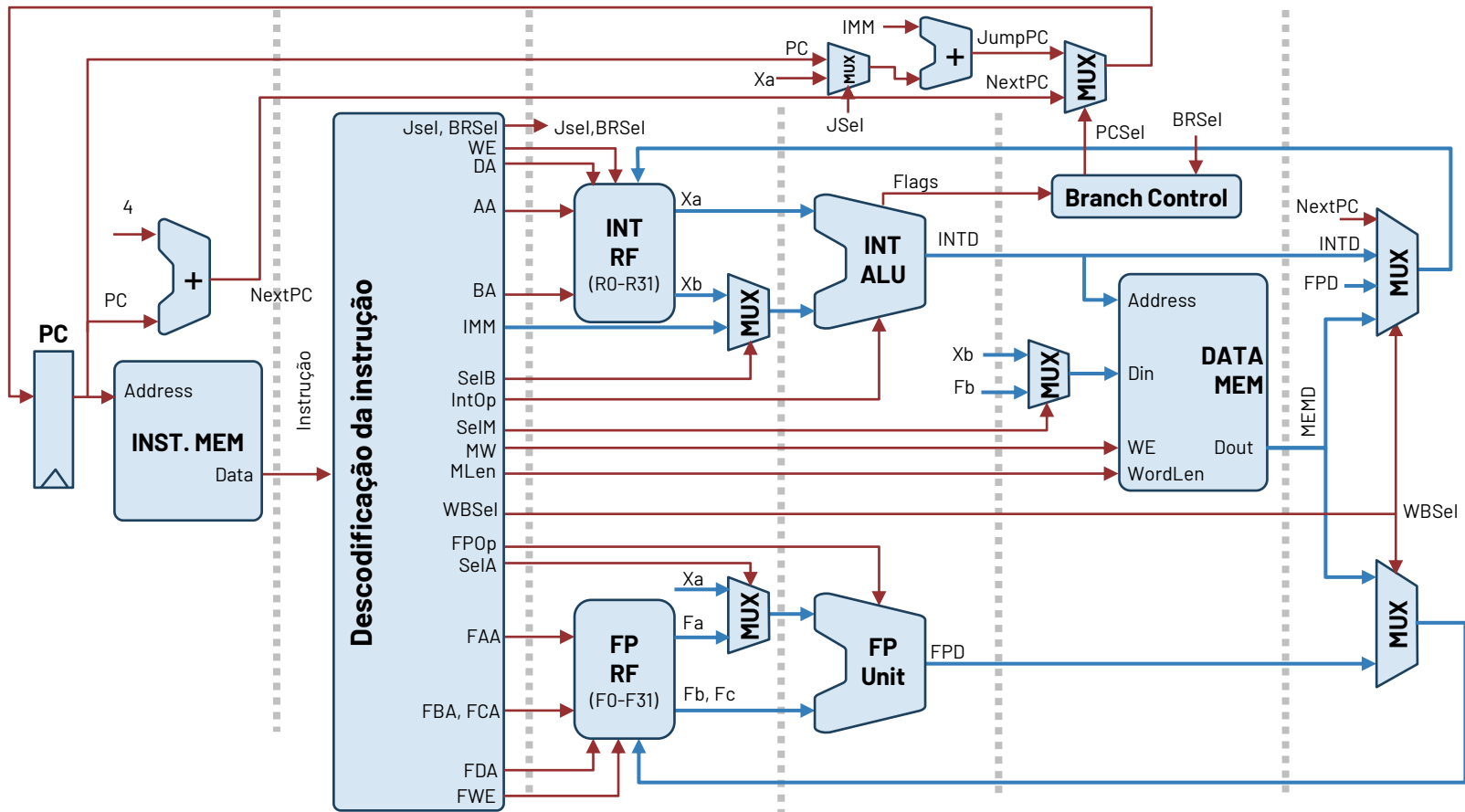
Instruction
Decode (ID)

Operand
Fetch (OF)

Execute
(EX)

Memory Access
(MEM)

Write Back
(WB)



Instruction
Fetch (IF)

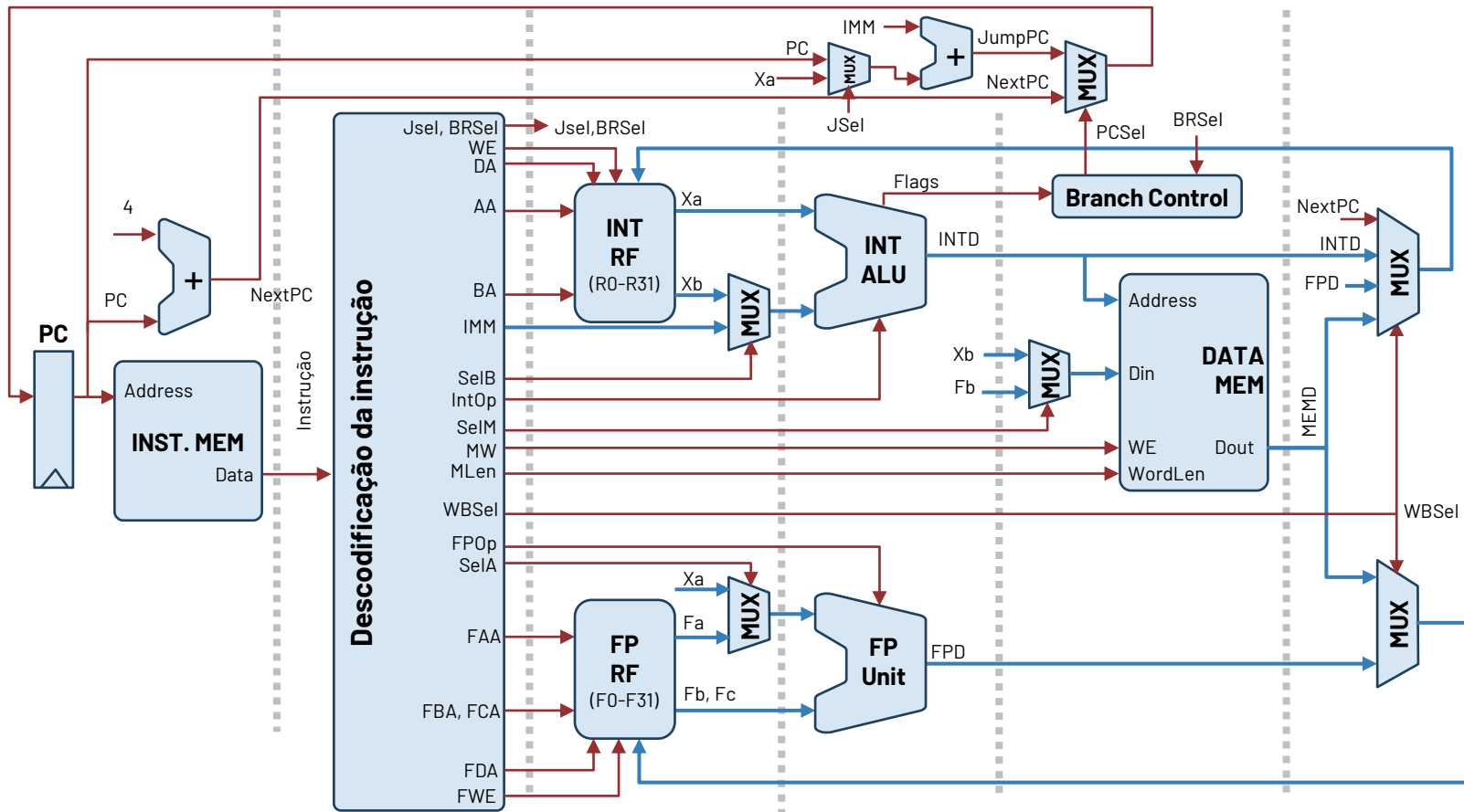
Instruction
Decode (ID)

Operand
Fetch (OF)

Execute
(EX)

Memory Access
(MEM)

Write Back
(WB)





Desenho do processador

Unidades fundamentais do processador

Instruction
Fetch (IF)

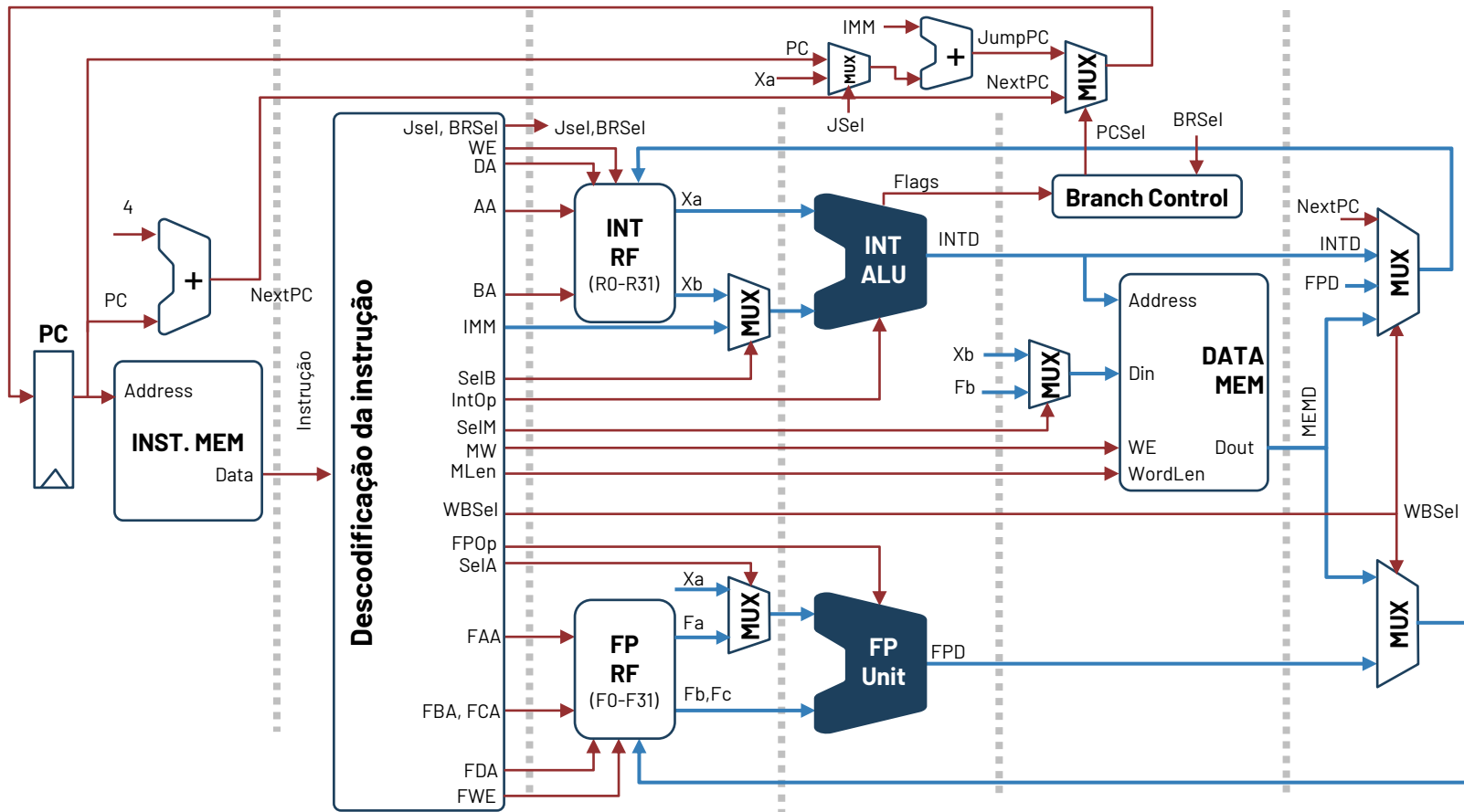
Instruction
Decode (ID)

Operand
Fetch (OF)

Execute
(EX)

Memory Access
(MEM)

Write Back
(WB)

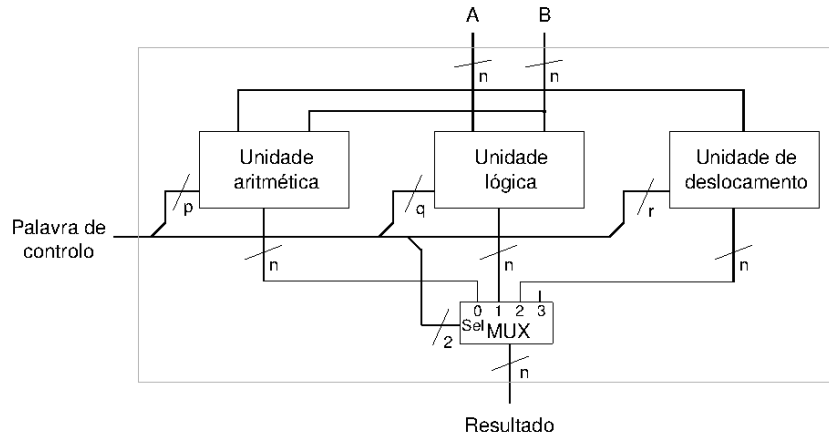


Unidades fundamentais do processador

Arithmetic and Logic Unit (ALU)



Estrutura interna da ALU de inteiros:



Unidades fundamentais do processador

Arithmetic and Logic Unit (ALU)

Existem várias codificações possíveis. A codificação é definida tendo em atenção as operações que a arquitetura necessita de suportar.

Para evitar um segundo passo de decodificação prefere-se a utilização de codificações esparsas, p. ex.:

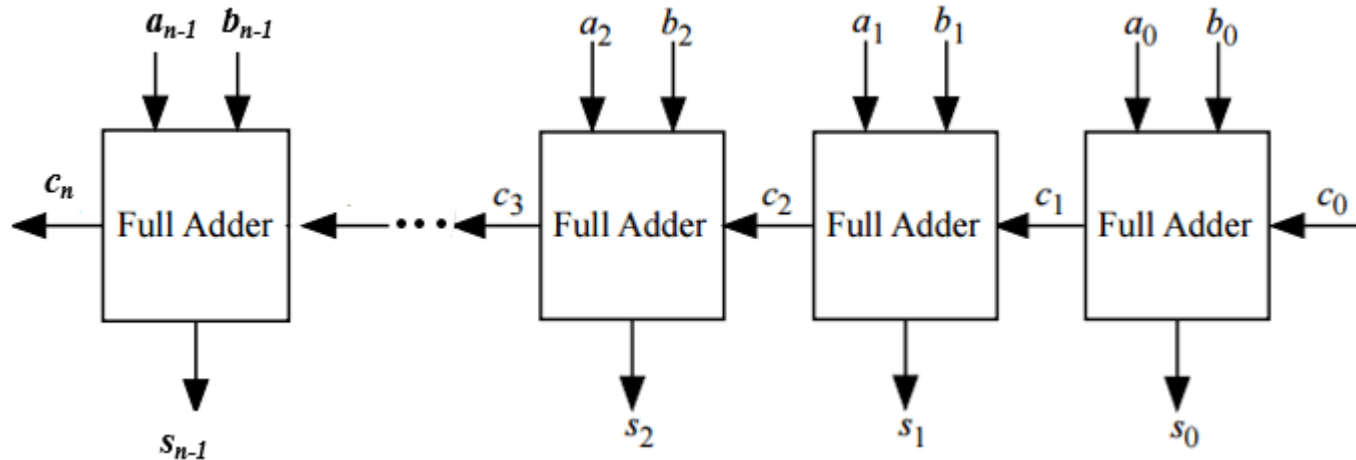
- $S_4, S_3 = 00 \rightarrow$ unidade aritmética;
- $S_4, S_3 = 01 \rightarrow$ unidade lógica;
- $S_4, S_3 = 10 \rightarrow$ unidade de deslocamento;
- $S_4, S_3 = 11 \rightarrow$ outras operações (ex: divisão, seno, coseno).

| | S_{4-0} | | Operação |
|-------------------------|-----------|---|---|
| Unidade aritmética | 00000 | $R \leftarrow A + B$ | Soma |
| | 00001 | $R \leftarrow A - B$ | Subtração |
| | 00010 | $R \leftarrow A + B + C$ | Soma com bit de transporte |
| | 00011 | $R \leftarrow A - B - C$ | Subtração com transporte negado |
| | 00100 | $R \leftarrow A \times B$ (low) | Multiplicação (parte menos significativa) |
| | 00101 | $R \leftarrow A \times B$ (unsigned high) | Multiplicação (parte mais significativa) |
| | 00110 | $R \leftarrow A \times B$ (signed high) | Multiplicação (parte mais significativa) |
| Unidade lógica | 00111 | $R \leftarrow A$ | Transferência de A |
| | 01-00 | $R \leftarrow B$ | Transferência de B |
| | 01-01 | $R \leftarrow A \& B$ | Conjunção |
| | 01-10 | $R \leftarrow A B$ | Disjunção |
| Unidade De deslocamento | 01-11 | $R \leftarrow A \oplus B$ | Disjunção exclusiva |
| | 10000 | $R \leftarrow \text{SHR } A$ | Deslocamento lógico à direita |
| | 10001 | $R \leftarrow \text{SHL } A$ | Deslocamento lógico à esquerda |
| | 10010 | $R \leftarrow \text{SHRA } A$ | Deslocamento aritmético à direita |
| | 10011 | $R \leftarrow \text{SHLA } A$ | Deslocamento aritmético à esquerda |
| | 10100 | $R \leftarrow \text{ROR } A$ | Rotação à direita |
| | 10101 | $R \leftarrow \text{ROL } A$ | Rotação à esquerda |
| | 10110 | $R \leftarrow \text{RORC } A$ | Rotação à direita com transporte |
| | 10111 | $R \leftarrow \text{ROLC } A$ | Rotação à esquerda com transporte |
| | ... | ... | Operações complexas (ex: DIV, SIN) |

Unidades fundamentais do processador

Somador

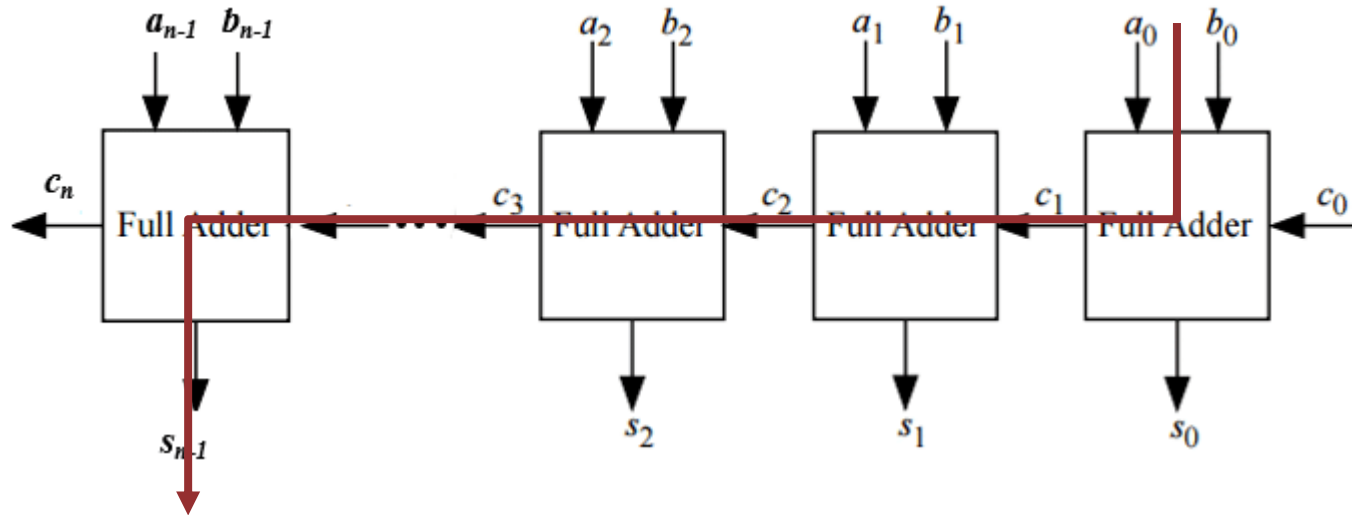
A. Ripple-Carry Adder



Unidades fundamentais do processador

Somador

A. Ripple-Carry Adder

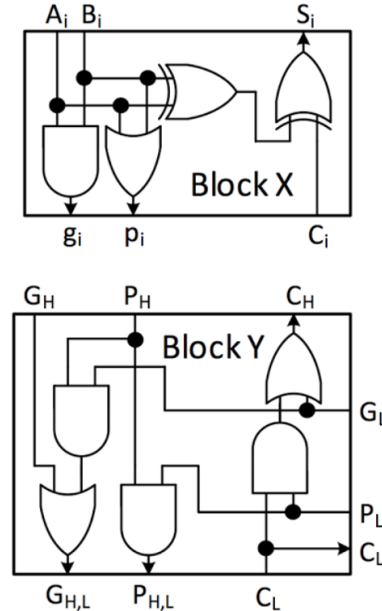
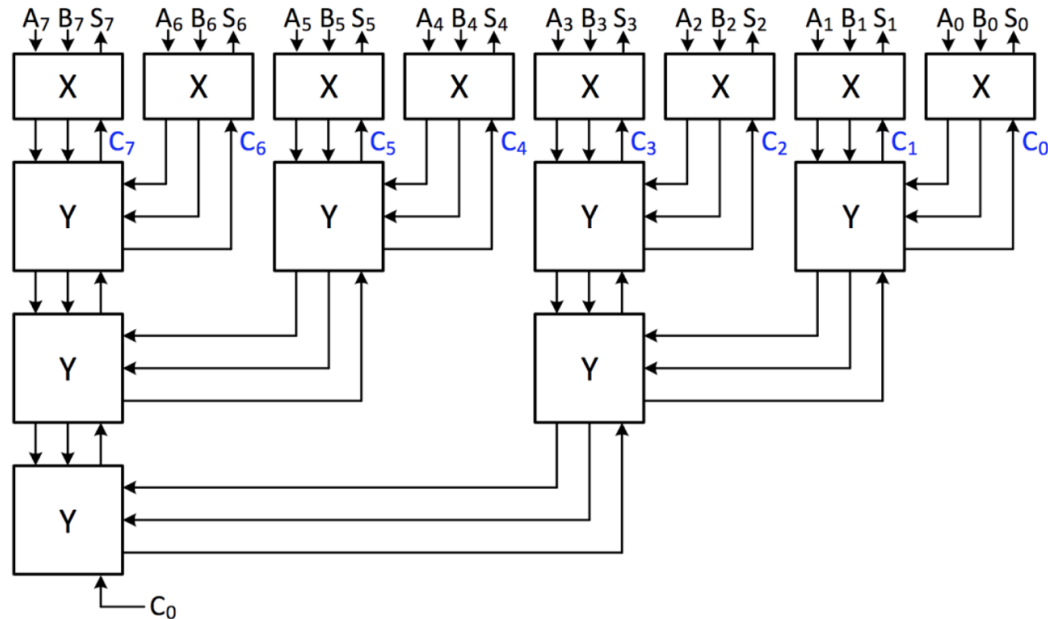


Caminho crítico
mais provável

Unidades fundamentais do processador

Somador

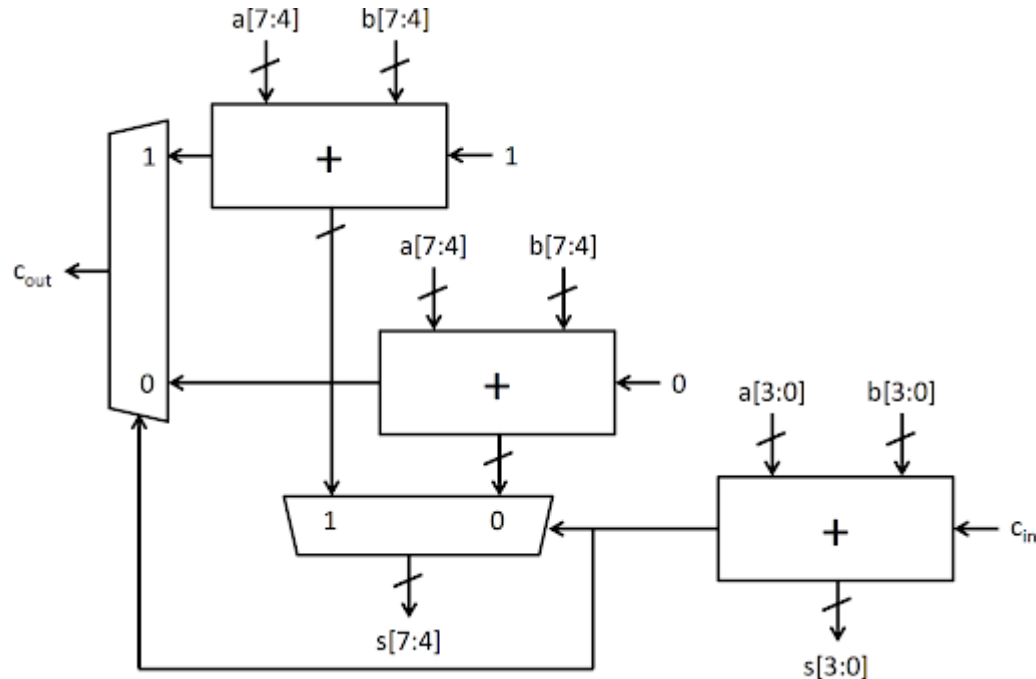
B. Carry Look-Ahead Adder



Unidades fundamentais do processador

Somador

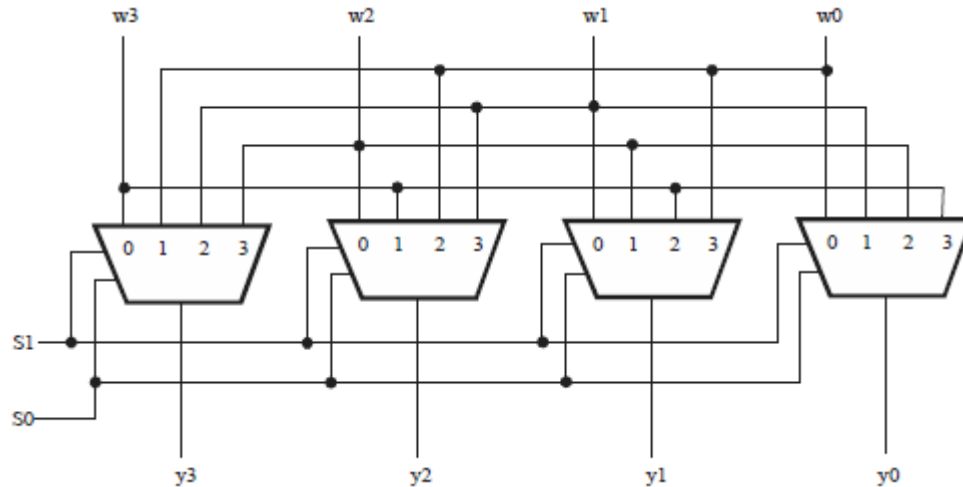
C. Carry Select Adder



Unidades fundamentais do processador

Barrel Shifter

B. Barrel Shifter



Cada entrada dos multiplexers corresponde a um deslocamento diferente

Os multiplexers selecionam o deslocamento certo.

Unidades fundamentais do processador

Multiplicador

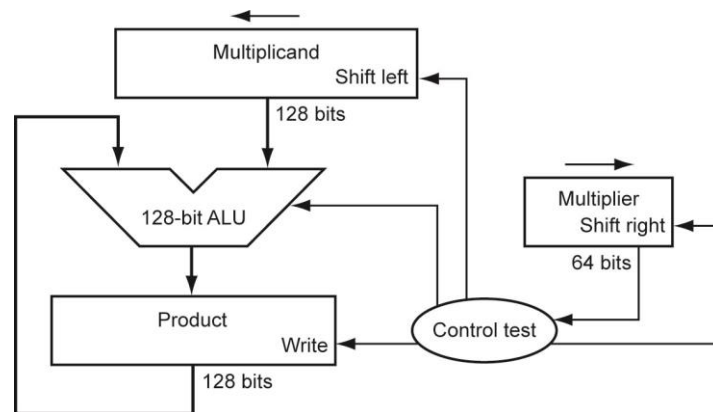
C. Multiplicação com/sem sinal

- É necessário conhecer o tipo de operandos (signed vs unsigned)
 - Em *unsigned* é feita uma extensão com zeros para determinar o resultado
 - Em *signed* é feita a extensão com o bit de sinal

1. Inicializar o multiplicando e o multiplicador
2. Se o bit menos significativo do multiplicador for 1, soma ao produto o multiplicando
3. Deslocar o multiplicando para a esquerda e o multiplicador para a direita
4. Repetir os pontos 2 e 3 até percorrer todos os bits do multiplicado

$$-114 \times 67 = 1000\ 1110_2 \times 0100\ 0011_2$$

| | |
|----------------------|--|
| Multiplicand: | 1 1 1 1 1 1 1 1 0 0 0 1 1 1 0 |
| Multiplier: | x 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 1 |
| | 1 1 1 1 1 1 1 1 1 0 0 0 1 1 1 0 |
| | 1 1 1 1 1 1 1 1 0 0 0 1 1 1 0 |
| + | 1 1 1 0 0 0 1 1 1 0 |
| | 1 1 1 0 0 0 1 0 0 0 1 0 1 0 1 0 |



Considerando um multiplicador de 64-bits

Unidades fundamentais do processador

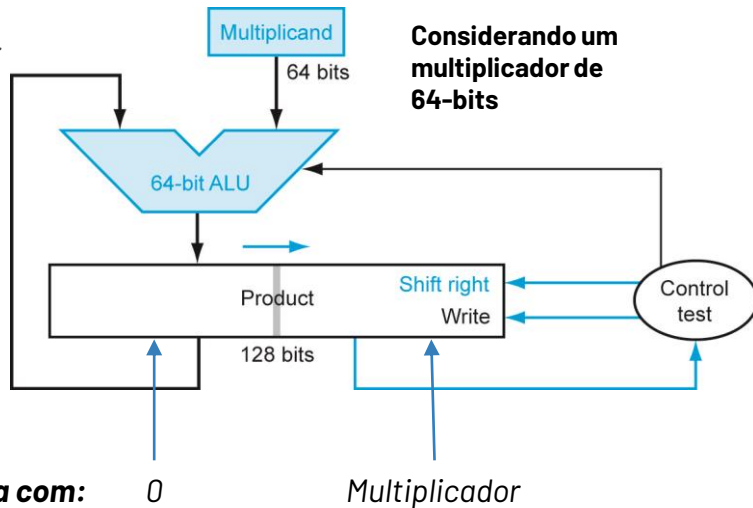
Multiplicador (versão refinada)

C. Multiplicação com/sem sinal

- É necessário conhecer o tipo de operandos (signed vs unsigned)
 - Em *unsigned* é feita uma extensão com zeros para determinar o resultado
 - Em *signed* é feita a extensão com o bit de sinal
1. Inicializar o multiplicando e o produto (parte alta com zeros, parte baixa com o multiplicado)
 2. Se o bit menos significativo do produto (multiplicador) for 1, soma ao produto o multiplicando
 3. Deslocar o produto para a direita
 4. Repetir os pontos 2 e 3 até percorrer todos os bits do multiplicador

$$-114 \times 67 = 1000\ 1110_2 \times 0100\ 0011_2$$

| | | | | | | | | | | | | | | | | | |
|----------------------|----------|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Multiplicand: | | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | |
| Multiplier: | x | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | |
| | | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | |
| | | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | | |
| | + | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | | | | | | |
| | | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |



Unidades fundamentais do processador

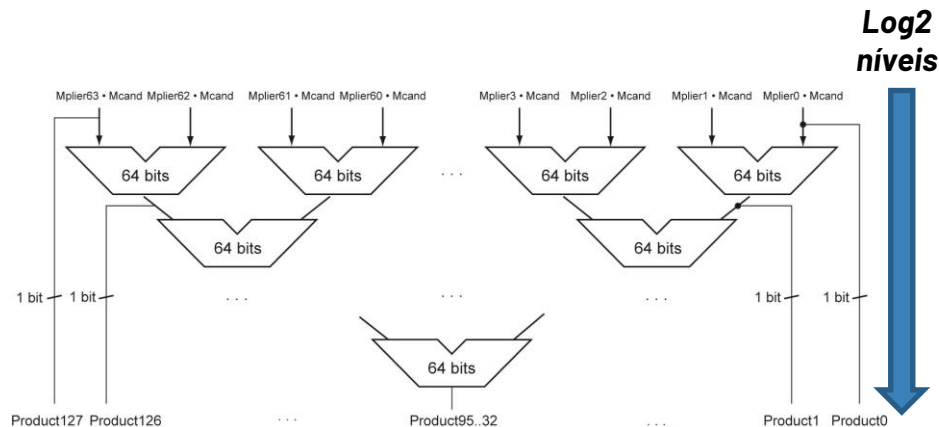
Multiplicador (versão mais rápida)

C. Multiplicação com/sem sinal

- É necessário conhecer o tipo de operandos (signed vs unsigned)
 - Em *unsigned* é feita uma extensão com zeros para determinar o resultado
 - Em *signed* é feita a extensão com o bit de sinal

$$-114 \times 67 = 1000\ 1110_2 \times 0100\ 0011_2$$

| | | | | | | | | | | | | | | | | | |
|----------------------|----------|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Multiplicand: | | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | |
| Multiplier: | x | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | |
| | | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | |
| | | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | | |
| | + | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | | | | | | |
| | | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |



Considerando um multiplicador de 64-bits

Divisão sem sinal

- Geralmente os números inteiros com sinal são representados em formato complemento para dois → facilita a implementação do somador!

$$173 / 4 = 1010\ 1101_2 / 100_2$$

$$100 \overline{) 10101101}$$

$$1 > 100?$$

Divisão sem sinal

- Geralmente os números inteiros com sinal são representados em formato complemento para dois → facilita a implementação do somador!

$$173 / 4 = 1010\ 1101_2 / 100_2$$

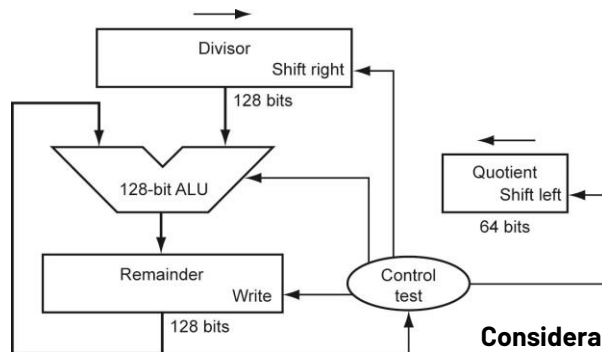
$$\begin{array}{r} 00101011 \\ 100 \overline{) 10101101} \\ \underline{-100} \\ 101 \\ \underline{-100} \\ 110 \\ \underline{-100} \\ 101 \\ \underline{-100} \\ 1 \end{array}$$

Resultado: $101011_2 = 43$

Resto: 1

D. Divisão (sem sinal)

- É necessário conhecer o tipo de operandos (signed vs unsigned)
- 1. Inicializar o registo divisor (parte alta com o divisor, parte baixa com zeros) e o resto com o dividendo
- 2. Deslocar o divisor para a direita e o quociente para a esquerda
- 3. Se $\text{divisor} \leq \text{resto}$ (ou seja, $\text{resto} - \text{divisor} > 0$)
 - $\text{resto} \leftarrow \text{resto} - \text{divisor}$; $\text{quociente (bit menos significativo)} \leftarrow 1$
 - Else
 - $\text{quociente (bit menos significativo)} \leftarrow 0$
- 4. Repetir mais 63 vezes os pontos 2 e 3



Considerando um divisor de 64-bits

$$173 / 4 = 1010\ 1101_2 / 100_2$$

$$\begin{array}{r} 00101011 \\ 100 \overline{) 10101101} \\ \underline{-100} \\ 101 \\ \underline{-100} \\ 110 \\ \underline{-100} \\ 101 \\ \underline{-100} \\ 1 \end{array}$$

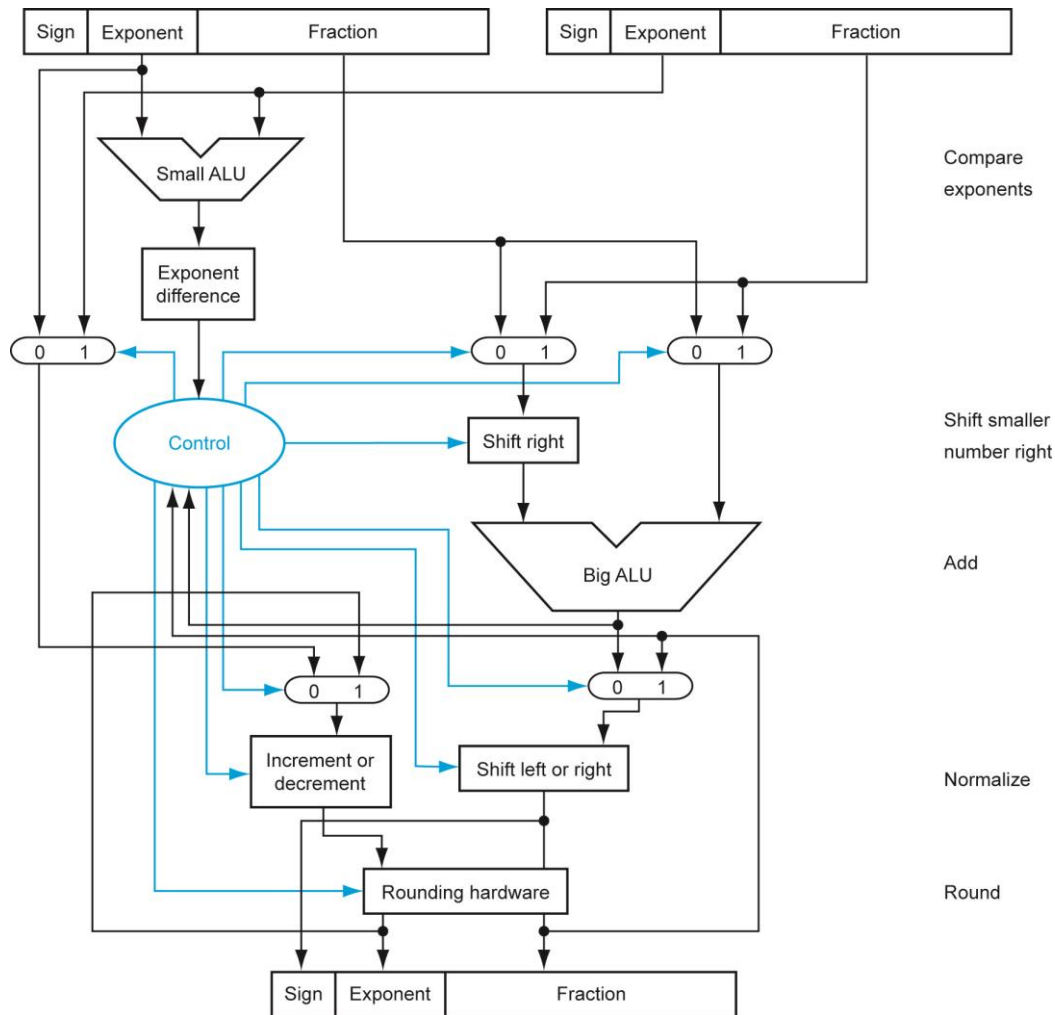
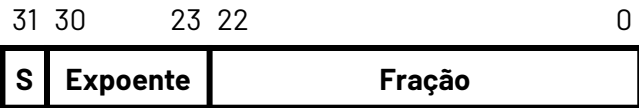
Resultado: $101011_2 = 43$

Resto: 1

Unidades fundamentais do processador

E. Soma em vírgula flutuante

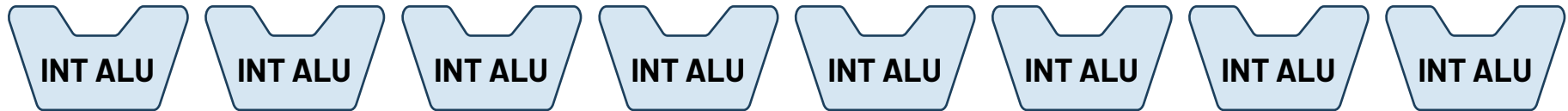
Formato de precisão simples (SP FP)



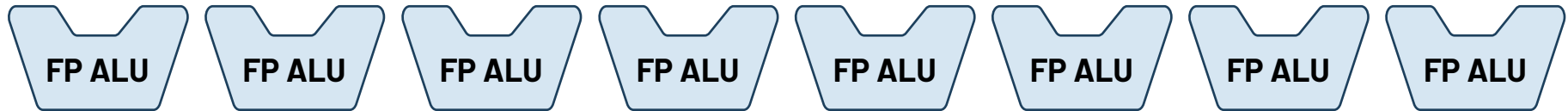
Unidades fundamentais do processador

Single Instruction, Multiple Data

F. Unidades SIMD (Single Instruction, Multiple Data) – inteiros



G. Unidades SIMD (Single Instruction, Multiple Data) – vírgula flutuante



Instruction
Fetch (IF)

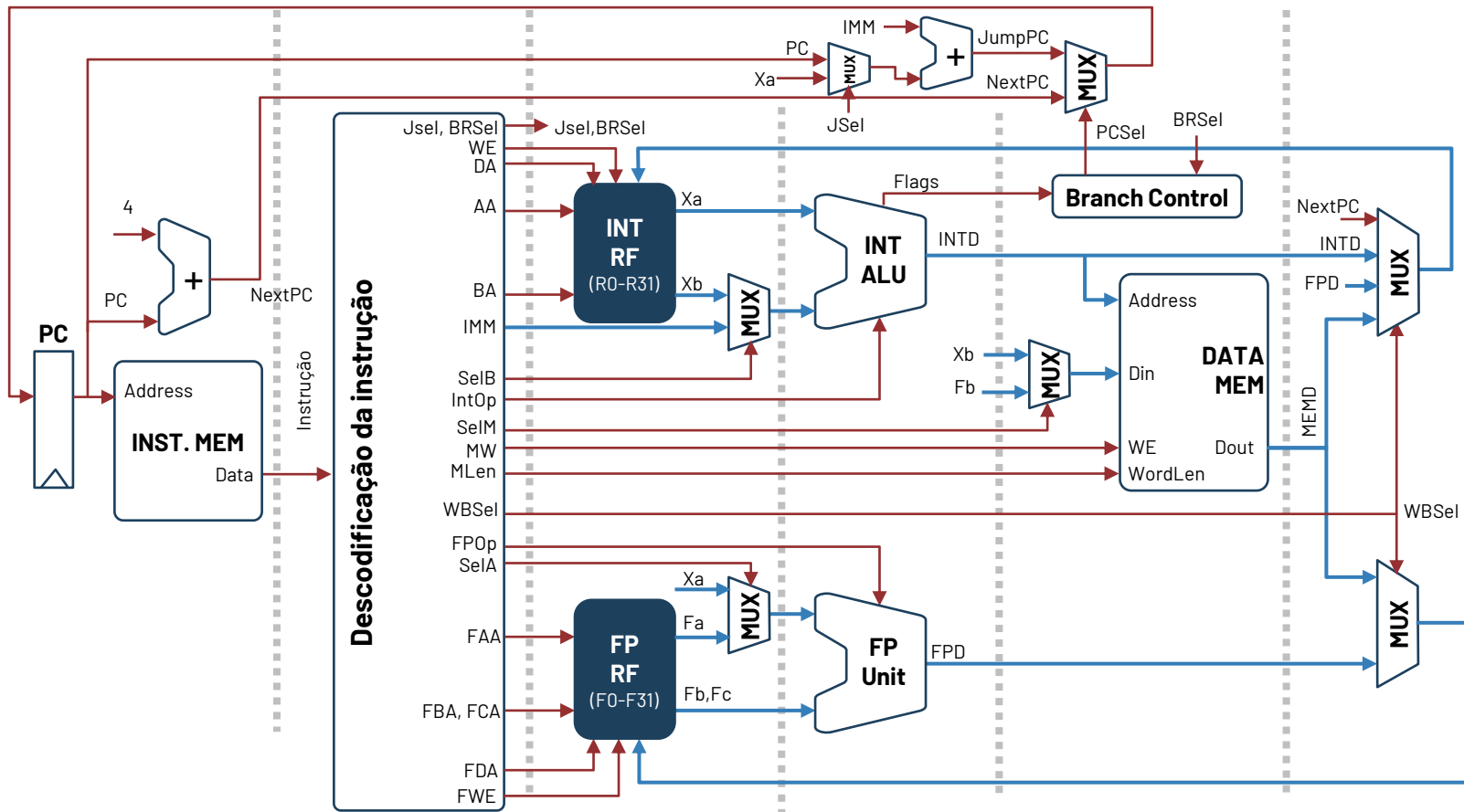
Instruction
Decode (ID)

Operand
Fetch (OF)

Execute
(EX)

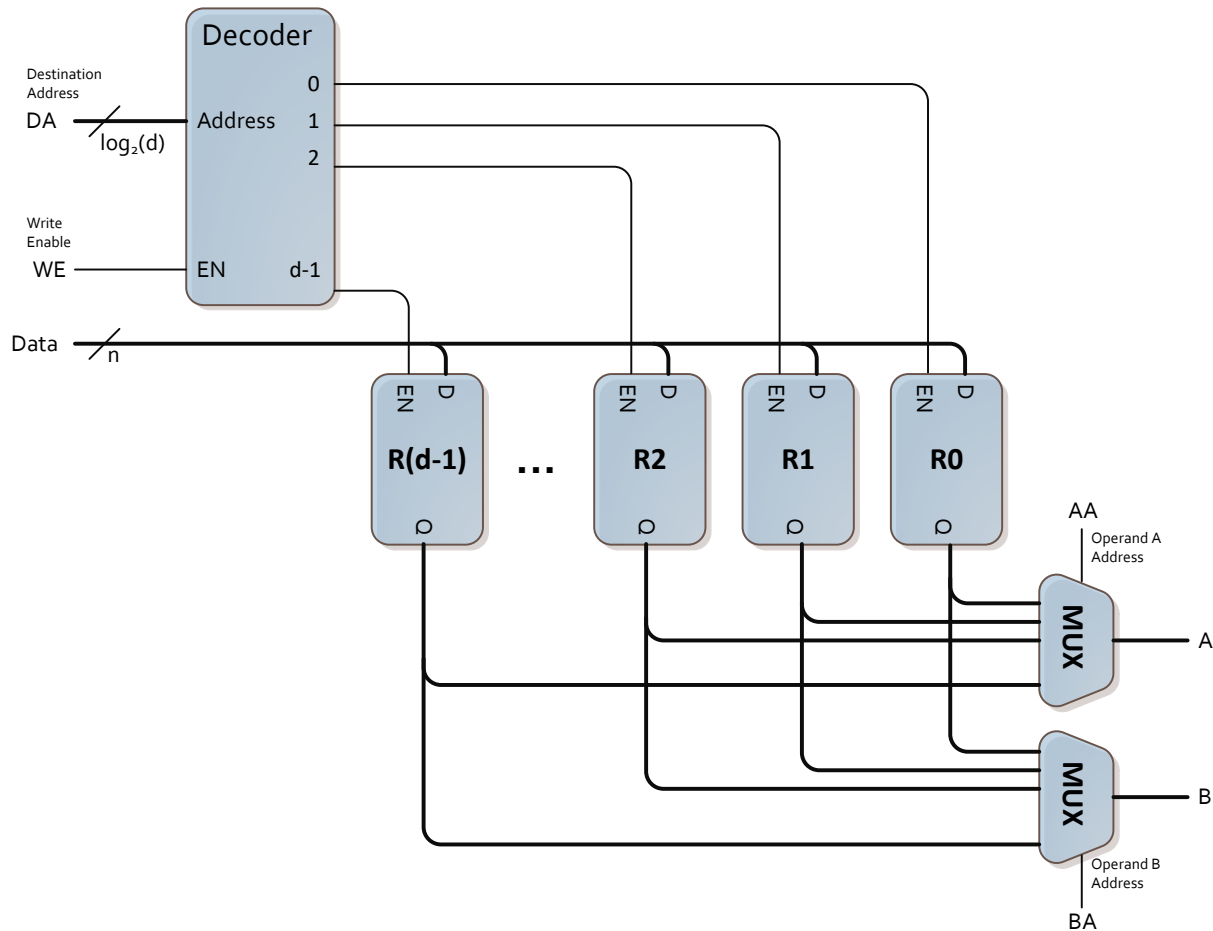
Memory Access
(MEM)

Write Back
(WB)



Banco de registros

Estrutura Interna



Instruction
Fetch (IF)

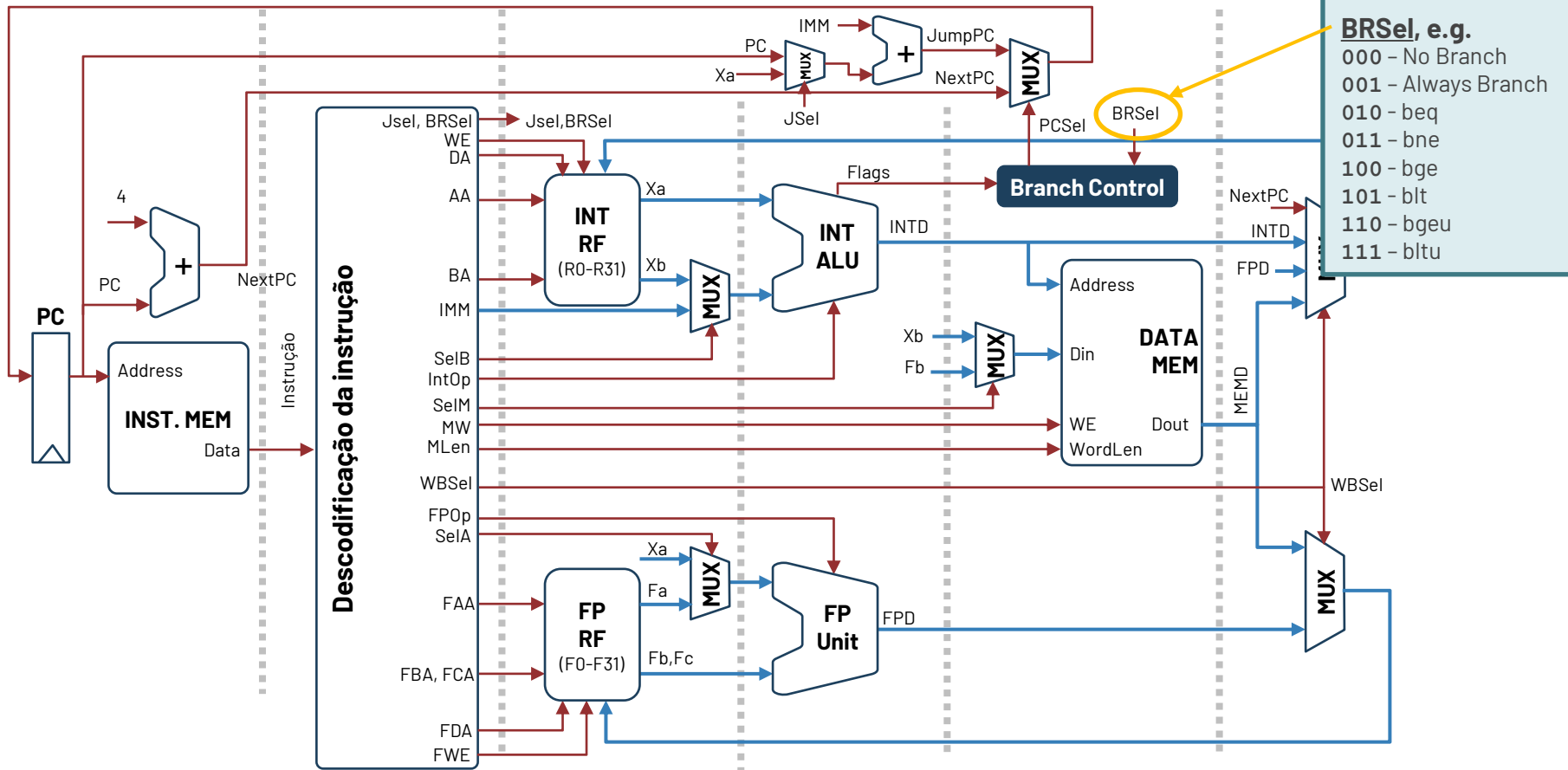
Instruction
Decode (ID)

Operand
Fetch (OF)

Execute
(EX)

Memory Access
(MEM)

Write Back
(WB)



Unidades fundamentais do processador

Controlo de salto

BRSel, e.g.

000 – No Branch
 001 – Always Branch
 010 – beq
 011 – bne
 100 – bge
 101 – blt
 110 – bgeu
 111 – bltu

| BRSel | Description | Flags | PCSel |
|-------|-----------------------------|-------------|--------------------------|
| 000 | No Branch | X | 0 – NextPC |
| 001 | Always Branch | X | 1 – JumpPC |
| 010 | Branch Equal | Z=1 Z=0 | 1 – JumpPC 0 – NextPC |
| 011 | Branch Not Equal | Z=0 Z=1 | 1 – JumpPC 0 – NextPC |
| 100 | Branch Greater or Equal | N=V N!=V | 1 – JumpPC 0 – NextPC |
| 101 | Branch Lower Than | N!=V N=V | 1 – JumpPC 0 – NextPC |
| 110 | Branch Greater or Equal (U) | C=0 C=1 | 1 – JumpPC 0 – NextPC |
| 111 | Branch Lower Than (U) | C=1 C=0 | 1 – JumpPC 0 – NextPC |

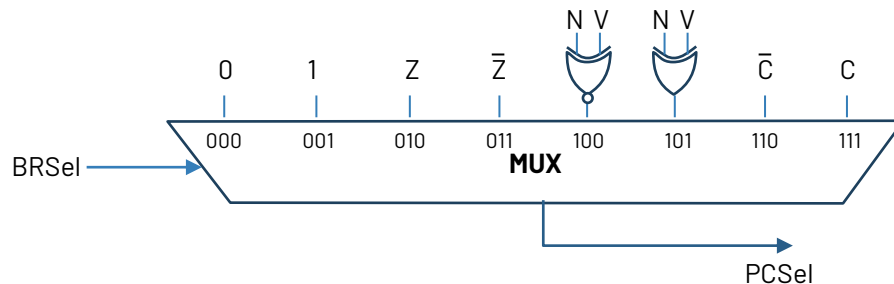
Unidades fundamentais do processador

Controlo de salto

BRSel, e.g.

000 - No Branch
 001 - Always Branch
 010 - beq
 011 - bne
 100 - bge
 101 - blt
 110 - bgeu
 111 - bltu

| BRSel | Description | Flags | PCSel |
|-------|-----------------------------|-------------|--------------------------|
| 000 | No Branch | X | 0 - NextPC |
| 001 | Always Branch | X | 1 - JumpPC |
| 010 | Branch Equal | Z=1 Z=0 | 1 - JumpPC 0 - NextPC |
| 011 | Branch Not Equal | Z=0 Z=1 | 1 - JumpPC 0 - NextPC |
| 100 | Branch Greater or Equal | N=V N!=V | 1 - JumpPC 0 - NextPC |
| 101 | Branch Lower Than | N!=V N=V | 1 - JumpPC 0 - NextPC |
| 110 | Branch Greater or Equal (U) | C=0 C=1 | 1 - JumpPC 0 - NextPC |
| 111 | Branch Lower Than (U) | C=1 C=0 | 1 - JumpPC 0 - NextPC |



Unidades fundamentais do processador

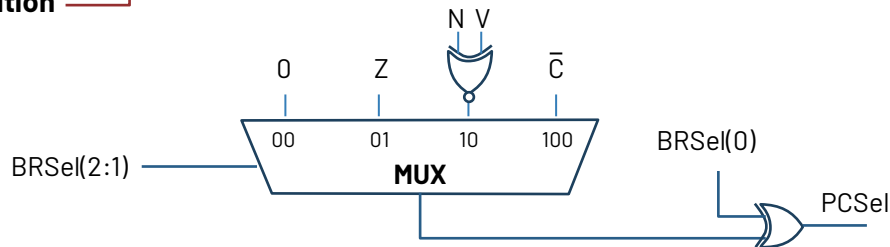
Controlo de salto

BRSel, e.g.

000 - No Branch
 001 - Always Branch
 010 - beq
 011 - bne
 100 - bge
 101 - blt
 110 - bgeu
 111 - bltu

| BRSel | Description | Flags | PCSel |
|-------|-----------------------------|-------------|--------------------------|
| 000 | No Branch | X | 0 - NextPC |
| 001 | Always Branch | X | 1 - JumpPC |
| 010 | Branch Equal | Z=1 Z=0 | 1 - JumpPC 0 - NextPC |
| 011 | Branch Not Equal | Z=0 Z=1 | 1 - JumpPC 0 - NextPC |
| 100 | Branch Greater or Equal | N=V N!=V | 1 - JumpPC 0 - NextPC |
| 101 | Branch Lower Than | N!=V N=V | 1 - JumpPC 0 - NextPC |
| 110 | Branch Greater or Equal (U) | C=0 C=1 | 1 - JumpPC 0 - NextPC |
| 111 | Branch Lower Than (U) | C=1 C=0 | 1 - JumpPC 0 - NextPC |

Invert condition



Instruction
Fetch (IF)

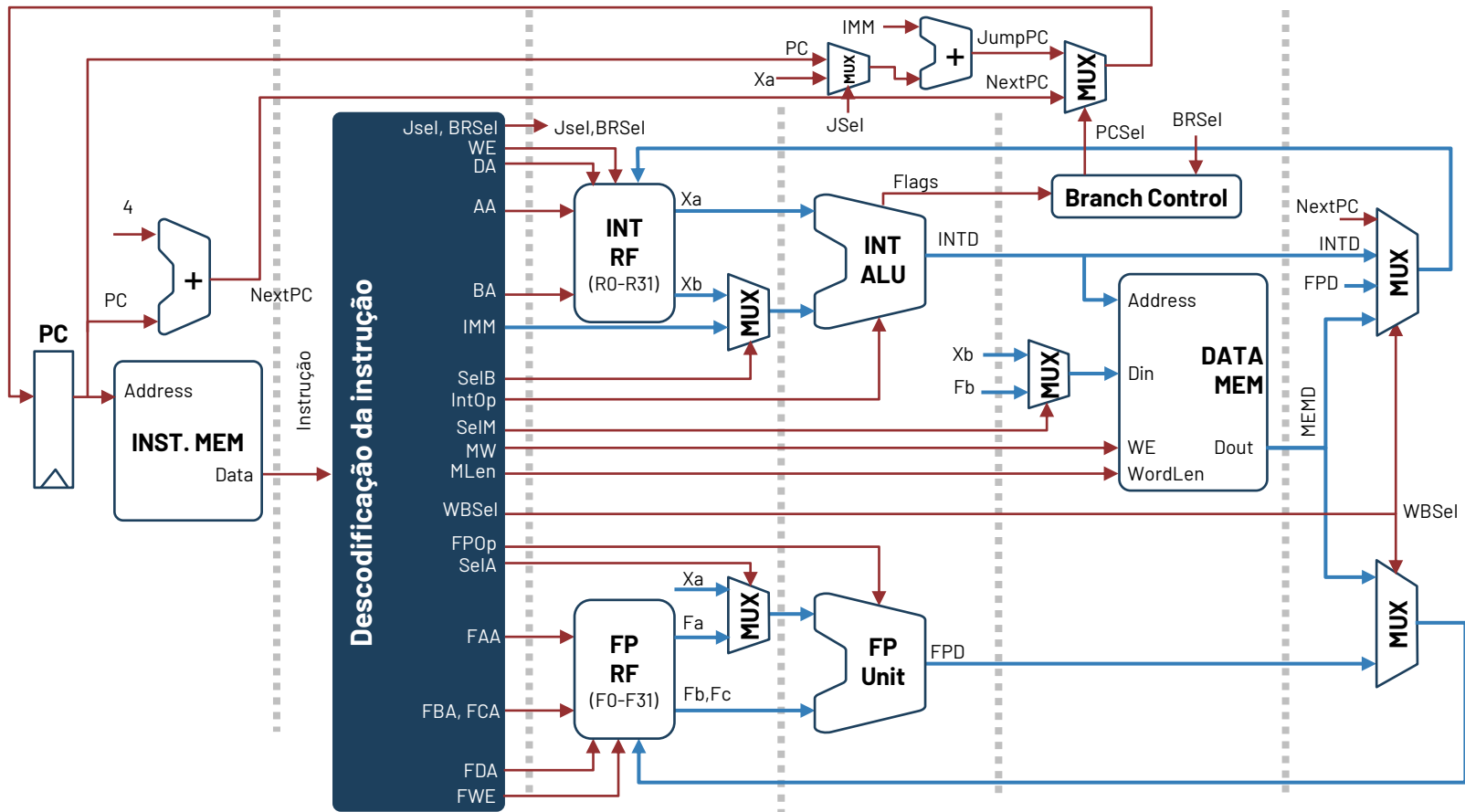
Instruction
Decode (ID)

Operand
Fetch (OF)

Execute
(EX)

Memory Access
(MEM)

Write Back
(WB)



Unidades fundamentais do processador

Descodificação de instruções

- Em processadores com elevado número de instruções, a descodificação da instrução é realizada de forma hierárquica (2 ou mais níveis):
 - Classe de instrução (ex: processamento de dados, memória, controlo)
 - Sub-classe de instrução (ex: processamento de dados de inteiros/vírgula flutuante)
 - Operação a realizar

Exemplo genérico:

| Opcode(6:0) | Classe |
|-------------|--------------------------------|
| 000xxxx | Load/Store |
| 01xxxxx | Operações sobre inteiros |
| 10xxxxx | Operações sobre floating point |
| 10x0xxx | Controlo de fluxo |
| ... | ... |

Unidades fundamentais do processador

Descodificação de instruções

- Em processadores com elevado número de instruções, a descodificação da instrução é realizada de forma hierárquica (2 ou mais níveis):
 - **Classe de instrução:** bits 6:2
 - **Dimensão da instrução:** bits 1:0 (opcode(1:0)=00, 01 ou 10 → instruções de 16 bits)

RISC-V:

Opcode (4:2)

| | | 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 |
|--------------|----|--------|----------|----------|----------|--------|----------|----------------------|------------------------|
| Opcode (6:5) | 00 | LOAD | LOAD-FP | Custom-0 | MISC-MEM | OP-IMM | AUIPC | OP-IMM-32 (RV64I) | Reserved (48 bits) |
| | 01 | STORE | STORE-FP | Custom-1 | Atomic | OP | LUI | OP-32 (RV64I) | Reserved (64 bits) |
| | 10 | FMADD | FMSUB | FNMSUB | FNMADD | OP-FP | Reserved | Custom-2 | Reserved (48 bits) |
| | 11 | BRANCH | JALR | Reserved | JAL | SYSTEM | Reserved | Custom-3 | Reserved (≥80 bits) |

Unidades fundamentais do processador

Descodificação de instruções

- Em processadores com elevado número de instruções, a descodificação da instrução é realizada de forma hierárquica (2 ou mais níveis):
 - Classe de instrução:** bits 6:2
 - Dimensão da instrução:** bits 1:0 (opcode(1:0)=00, 01 ou 10 → instruções de 16 bits)

RISC-V:

Opcode (4:2)

| | | Load / Store | | Processadores Especializados | +Memoria | Process. | Transf. | Process. (RV64) | Instruções >32bits |
|------------------------------|----|--------------|----------|------------------------------|----------|----------|----------|-------------------|----------------------|
| Fused Multiply-add/sub (5:1) | 00 | LOAD | LOAD-FP | Custom-0 | MISC-MEM | OP-IMM | AUIPC | OP-IMM-32 (RV64I) | Reserved (48 bits) |
| | | STORE | STORE-FP | Custom-1 | Atomic | OP | LUI | OP-32 (RV64I) | Reserved (64 bits) |
| | | FMADD | FMSUB | FNMSUB | FNMADD | OP-FP | Reserved | Custom-2 | Reserved (48 bits) |
| | 11 | BRANCH | JALR | Reserved | JAL | SYSTEM | Reserved | Custom-3 | Reserved (>=80 bits) |

Controlo de fluxo

Sistema:

ecall, ebreak, csrrs, csrrc, csrrw

Processadores Especializados

Instruction
Fetch (IF)

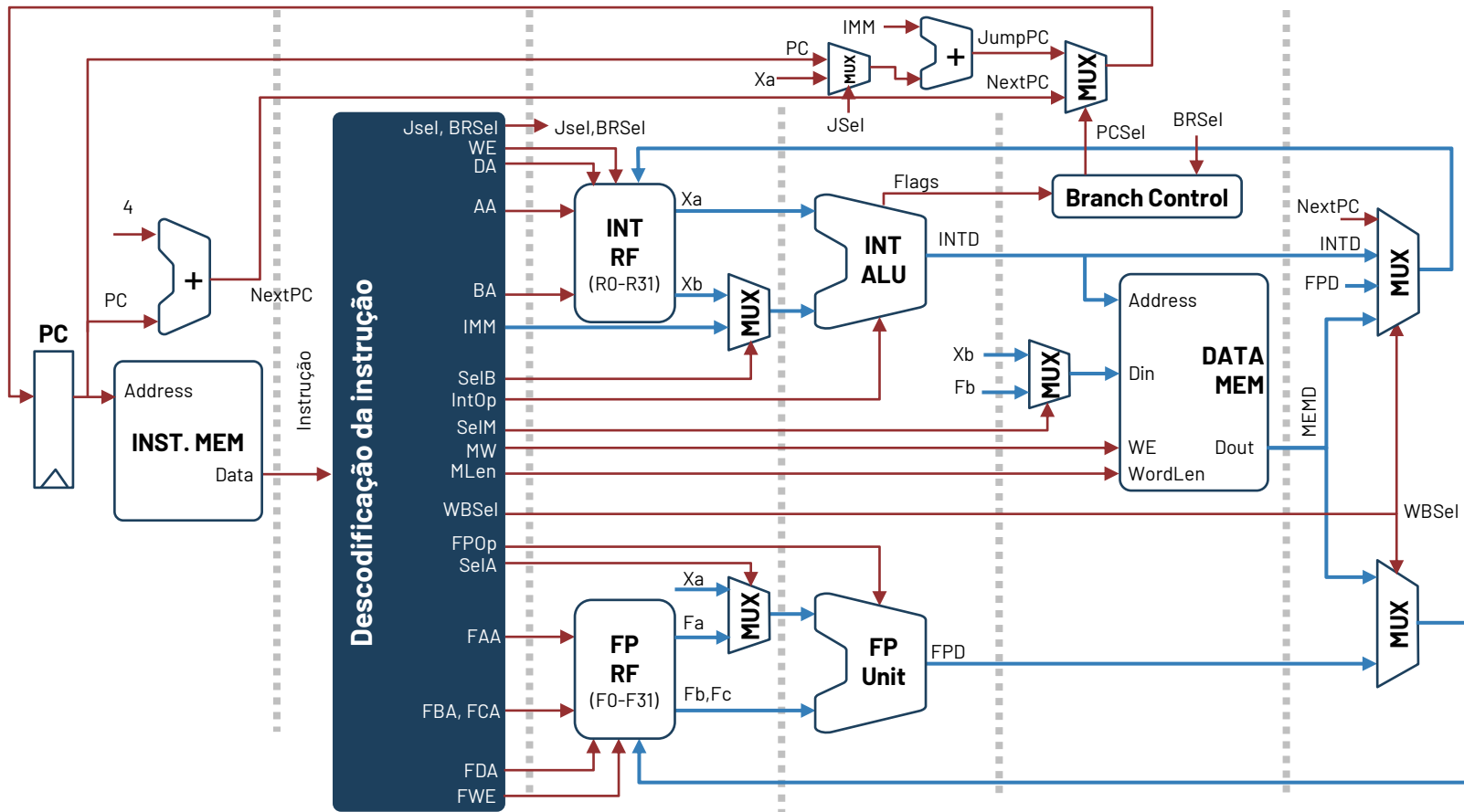
Instruction
Decode (ID)

Operand
Fetch (OF)

Execute
(EX)

Memory Access
(MEM)

Write Back
(WB)

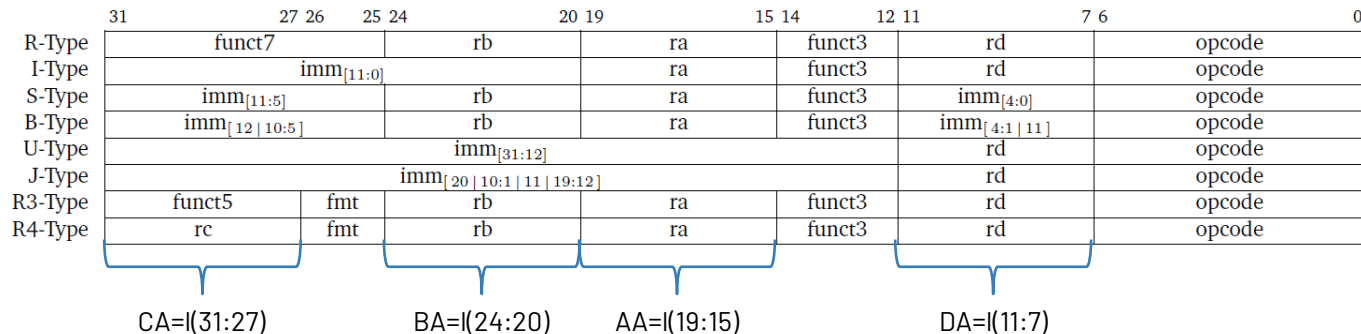


Unidades fundamentais do processador

Descodificação de instruções

- Dentro de uma dada sub-classe a descodificação é geralmente muito simples, obtida:
 - Diretamente a partir da palavra de instrução
- ou...

Instruction Formats



Note que neste caso nem é necessário saber qual o tipo de instrução para realizar a leitura dos operandos.

Permite aceder aos bancos de registos (OF – Operand Fetch) em paralelo com a descodificação de instruções (ID – Instruction Decode), reduzindo o caminho crítico. É por isso que em processadores simples, os estágios de OF e ID aparecem juntos.

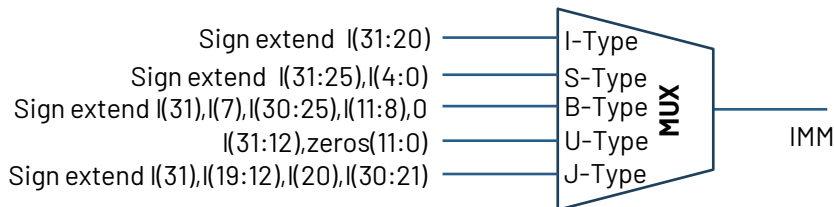
Unidades fundamentais do processador

Descodificação de instruções

- Dentro de uma dada sub-classe a descodificação é geralmente muito simples, obtida:
 - Diretamente a partir da palavra de instrução
 - Ou com recurso a expressões lógicas simples

Instruction Formats

| | 31 | 27 26 | 25 24 | 20 19 | 15 14 | 12 11 | 7 6 | 0 |
|---------|---|-------|-------|-------|--------|--------|---------------------------|--------|
| R-Type | funct7 | | | rb | ra | funct3 | rd | opcode |
| I-Type | imm _[11:0] | | | | ra | funct3 | rd | opcode |
| S-Type | imm _[11:5] | | | rb | ra | funct3 | imm _[4:0] | opcode |
| B-Type | imm _[12 10:5] | | | rb | ra | funct3 | imm _[4:1 11] | opcode |
| U-Type | imm _[31:12] | | | | | | rd | opcode |
| J-Type | imm _[20 10:1 11 19:12] | | | | | | rd | opcode |
| R3-Type | funct5 | fmt | rb | ra | funct3 | rd | opcode | |
| R4-Type | rc | fmt | rb | ra | funct3 | rd | opcode | |



Descodificação da instrução

WE
DA

AA

BA

IMM

SeIB

IntOp

SeIM

MW

MLen

WBSeI

FPOp

SeIA

FAA

FBA

DFA

FWE

Unidades fundamentais do processador

Descodificação de instruções

- Dentro de uma dada sub-classe a descodificação é geralmente muito simples, obtida:
 - Diretamente a partir da palavra de instrução
 - Ou com recurso a expressões lógicas simples

| | | Opcode (4:2) | | | | | | | |
|--------------|----|--------------|----------|----------|----------|--------|----------|-------------------|----------------------|
| | | 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 |
| Opcode (6:5) | 00 | LOAD | LOAD-FP | Custom-0 | MISC-MEM | OP-IMM | AUIPC | OP-IMM-32 (RV64I) | Reserved (48 bits) |
| | 01 | STORE | STORE-FP | Custom-1 | Atomic | OP | LUI | OP-32 (RV64I) | Reserved (64 bits) |
| | 10 | FMADD | FMSUB | FNMSUB | FNMADD | OP-FP | Reserved | Custom-2 | Reserved (48 bits) |
| | 11 | BRANCH | JALR | Reserved | JAL | SYSTEM | Reserved | Custom-3 | Reserved (>=80 bits) |

Se OP-IMM ou se OP-IMM-32 → SelB=1



SelB = $\overline{\text{Opcode}_6} \overline{\text{Opcode}_5} \overline{\text{Opcode}_4} \overline{\text{Opcode}_2}$

Descodificação da instrução

WE
DA

AA

BA

IMM

SelB

IntOp

SelM

MW

MLen

WBSEL

FP0p

SelA

FAA

FBA

DFA

FWE

- Dentro de uma dada sub-classe a descodificação é geralmente muito simples, obtida:
 - Diretamente a partir da palavra de instrução
 - Ou com recurso a expressões lógicas simples

Estratégia para gerar as expressões lógicas: construir a tabela de verdade com:

- Entrada: palavra de instrução (na maioria dos casos apenas é necessário considerar os bits do OPCODE)
- Saída: cada um dos sinais da palavra de controlo (saídas do descodificador de instruções)
- Método: extrair a expressão lógica (inspeção da tabela de verdade ou mapas de Karnaugh) para cada bit.

- Dentro de uma dada sub-classe a descodificação é geralmente muito simples, obtida:
 - Diretamente a partir da palavra de instrução
 - Ou com recurso a expressões lógicas simples
- Quando a codificação das instruções é demasiado complexa, pode ser realizada através de uma memória descodificação:



Na prática: corresponde a colocar a tabela de verdade anterior diretamente na memória.



Descodificação de instruções

Suporte para o conjunto de instruções

Instruction
Fetch (IF)

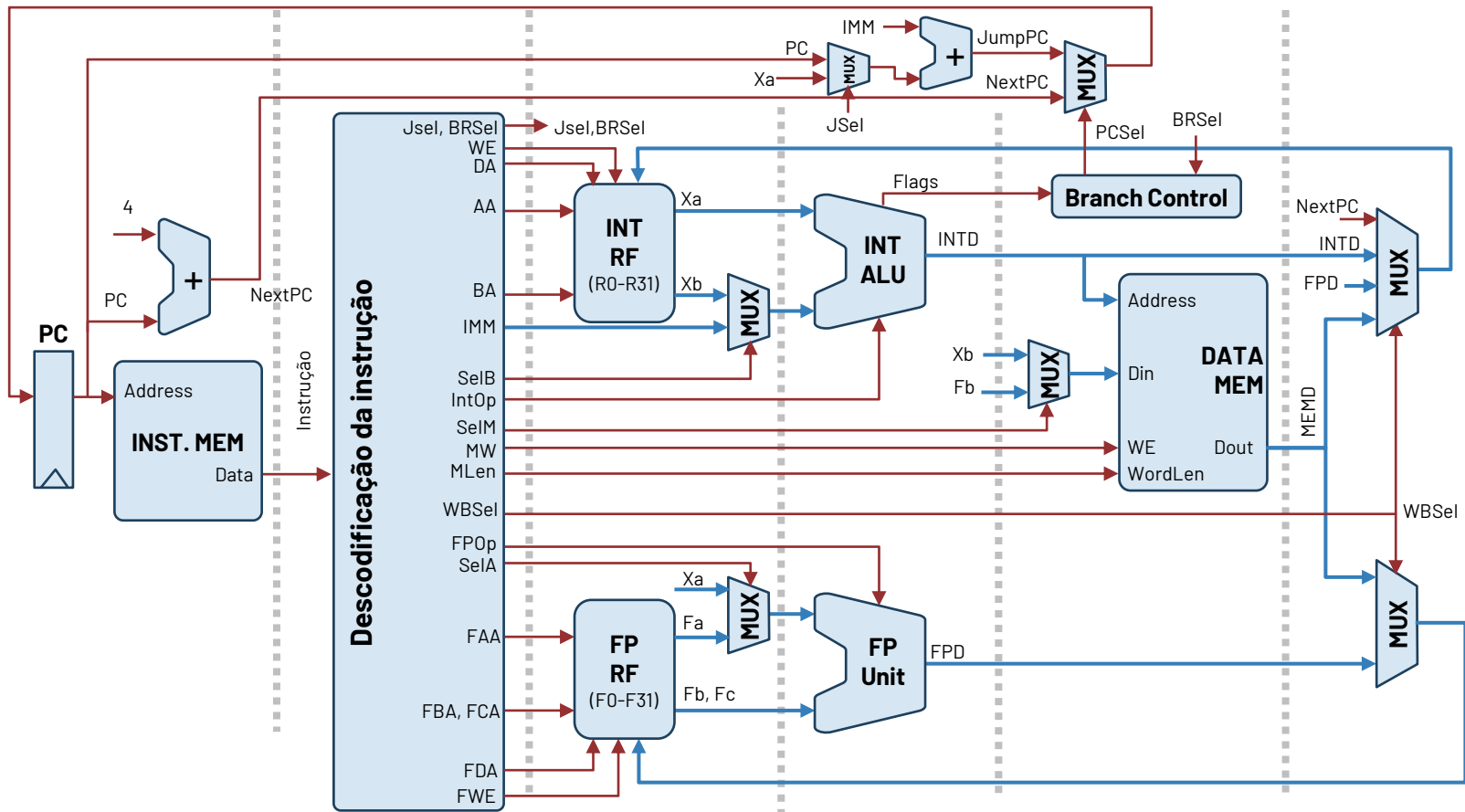
Instruction
Decode (ID)

Operand
Fetch (OF)

Execute
(EX)

Memory Access
(MEM)

Write Back
(WB)



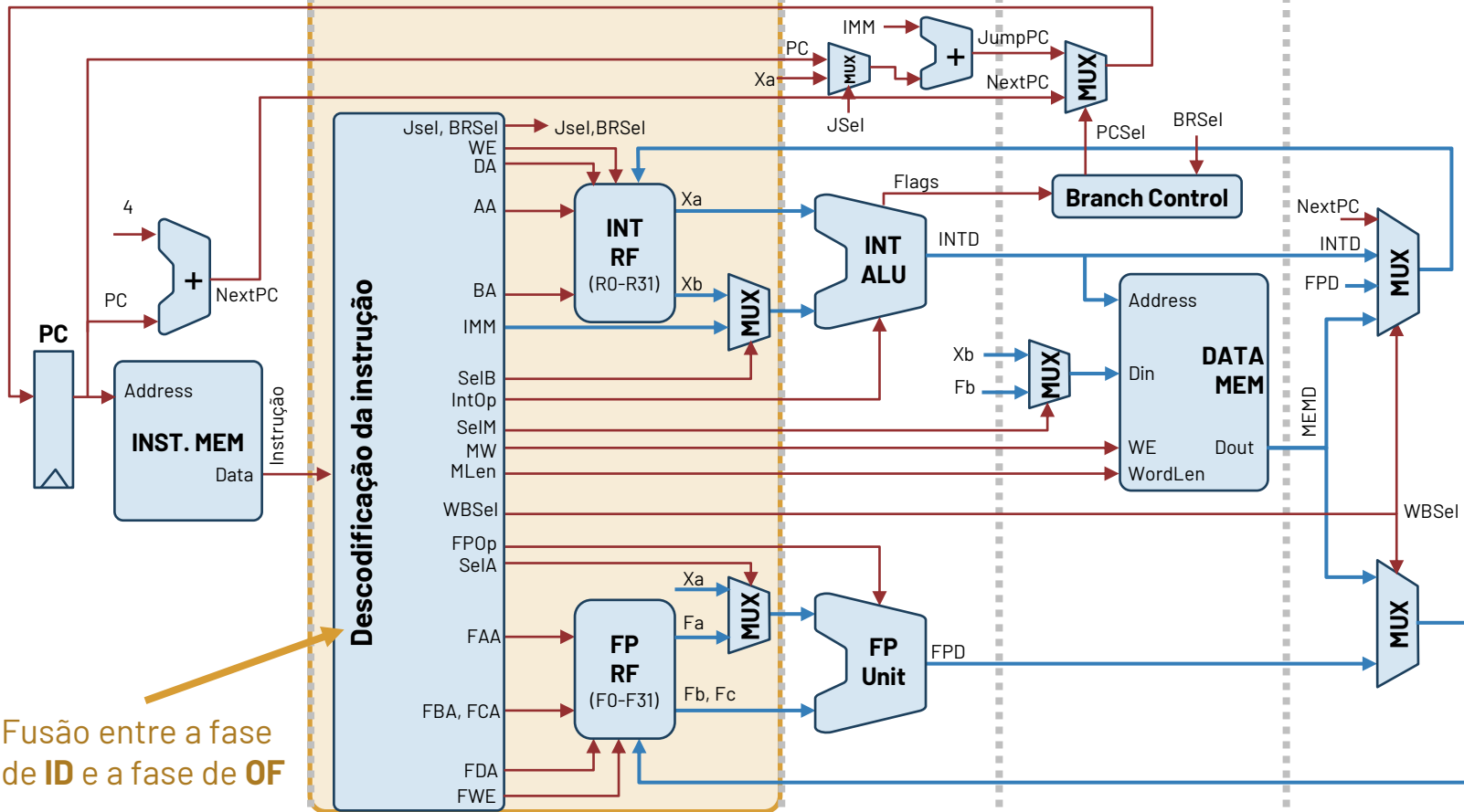
Instruction
Fetch (IF)

Instruction
Decode (ID)

Execute
(EX)

Memory Access
(MEM)

Write Back
(WB)



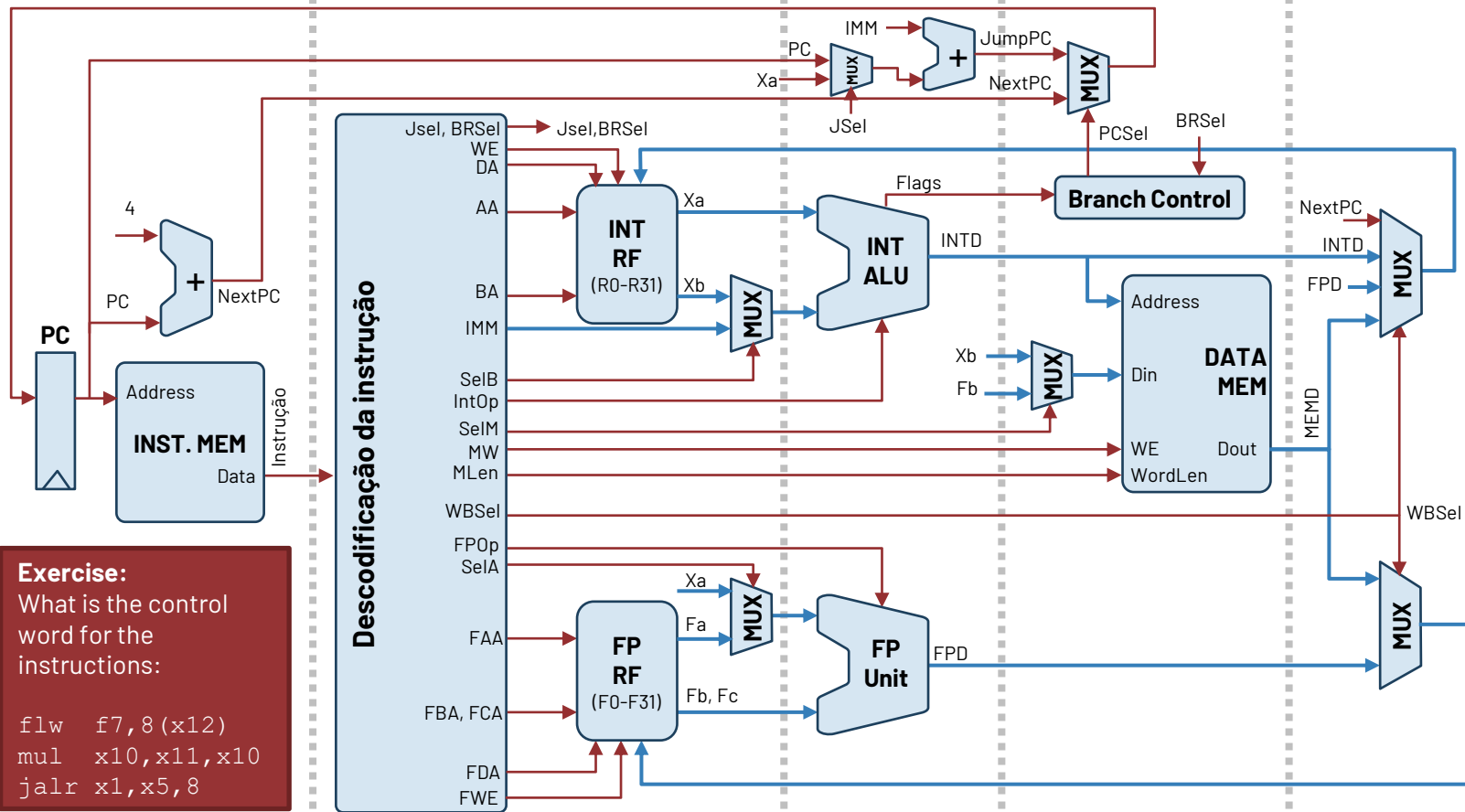
Instruction
Fetch (IF)

Instruction
Decode (ID)

Execute
(EX)

Memory Access
(MEM)

Write Back
(WB)



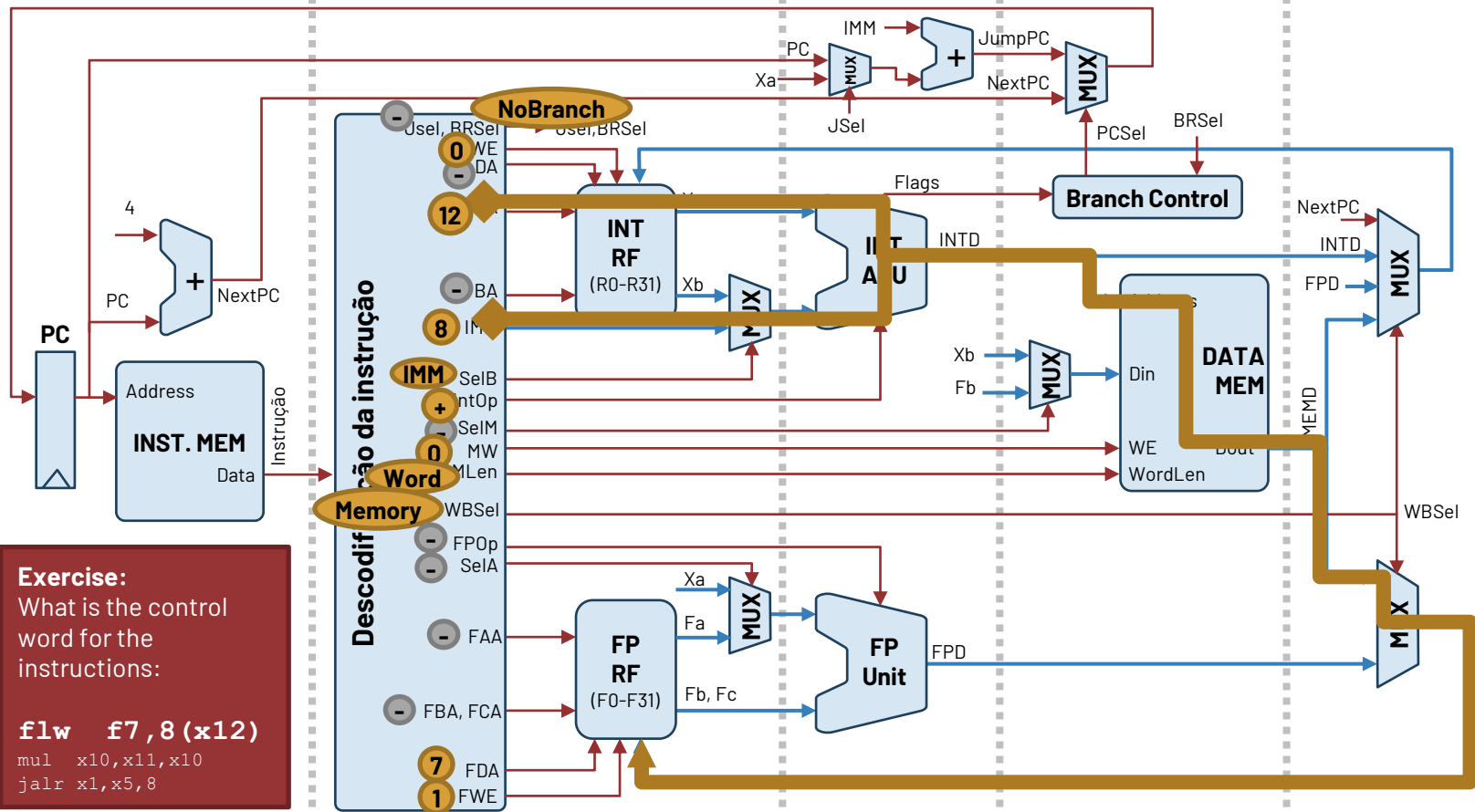
Instruction
Fetch (IF)

Instruction
Decode (ID)

Execute
(EX)

Memory Access
(MEM)

Write Back
(WB)



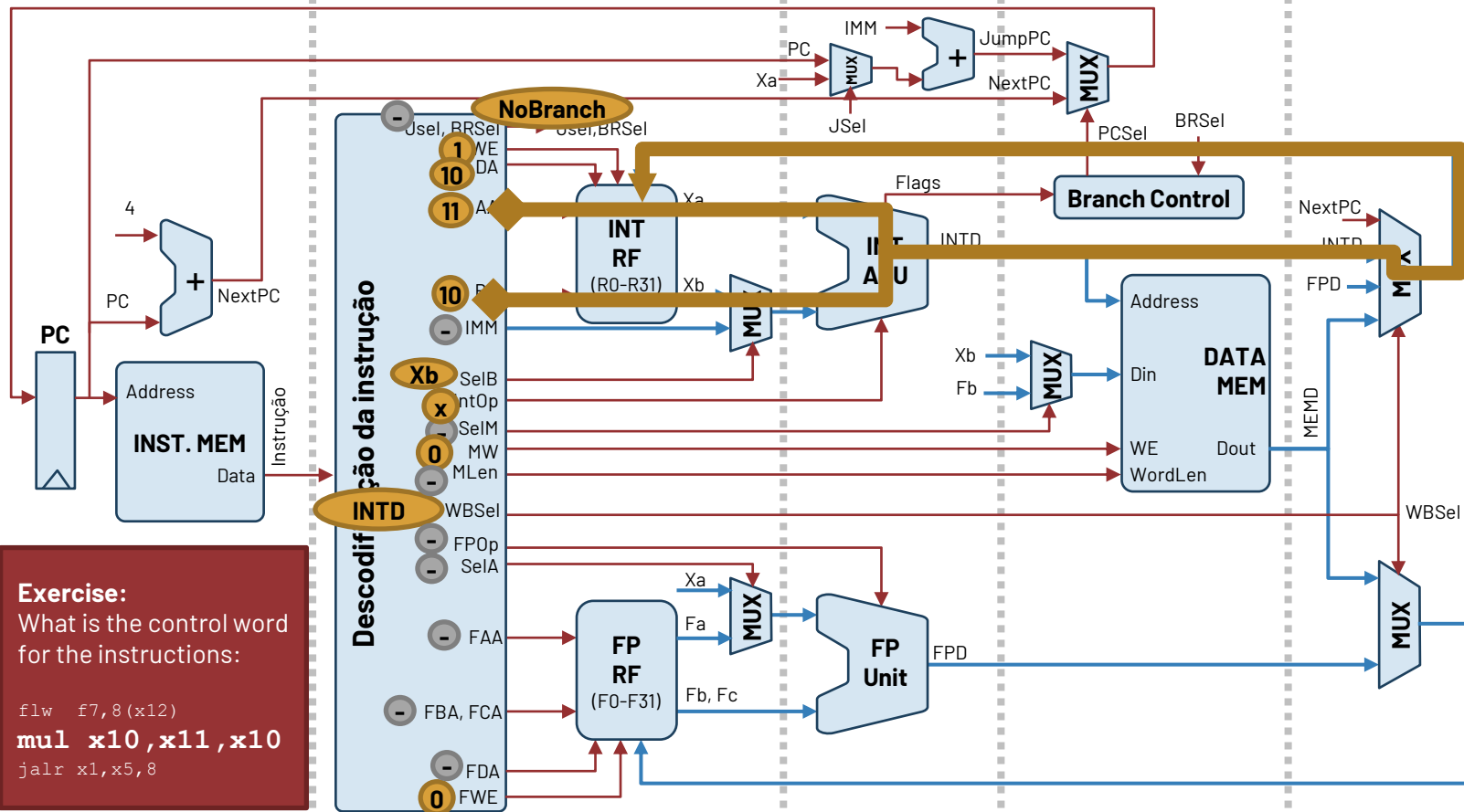
Instruction
Fetch (IF)

Instruction
Decode (ID)

Execute
(EX)

Memory Access
(MEM)

Write Back
(WB)



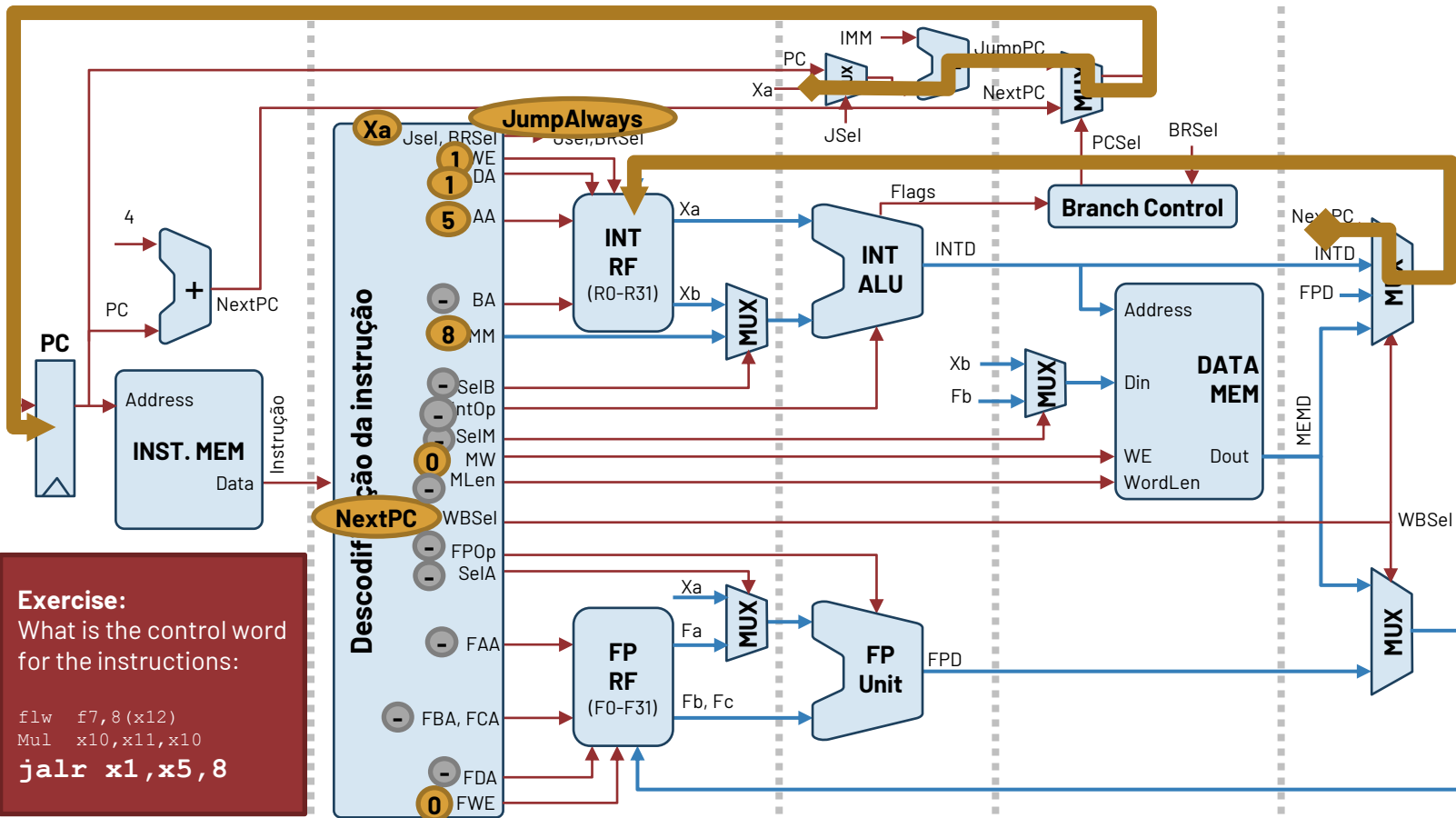
Instruction
Fetch (IF)

Instruction
Decode (ID)

Execute
(EX)

Memory Access
(MEM)

Write Back
(WB)



- **Observações:**
 - Os sinais que controlam a escrita em registos (incluindo PC) **nunca** são opcionais
 - É mais fácil determinar o valor dos sinais de controlo, percorrendo a arquitectura do fim (escrita/resultado) para o início (leitura dos operandos)