


Design de Computadores

Aula 11

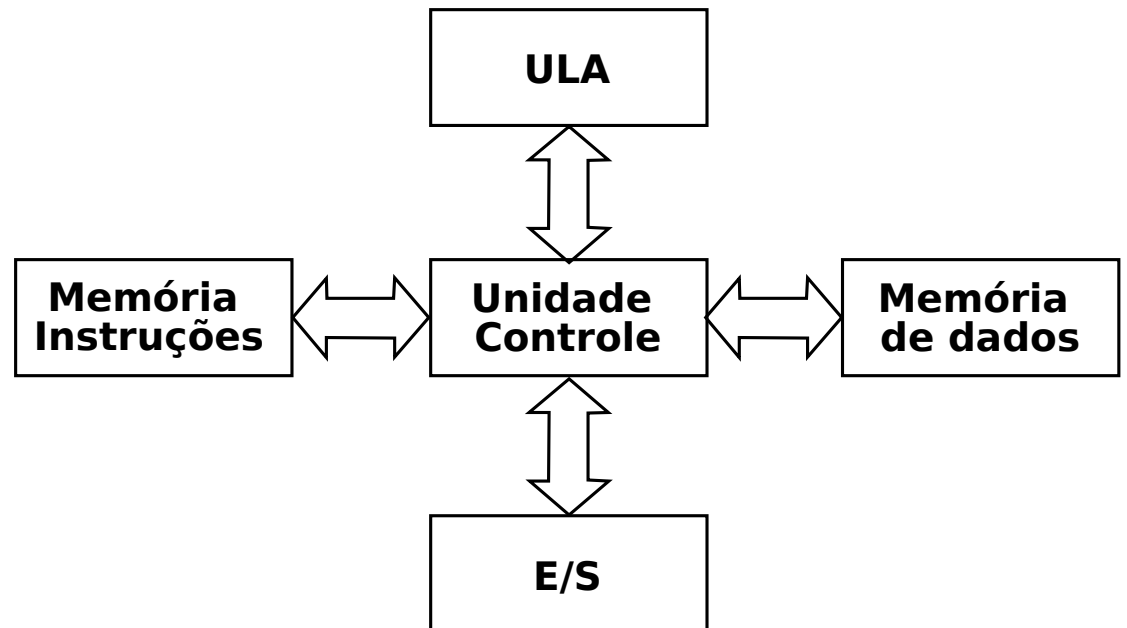
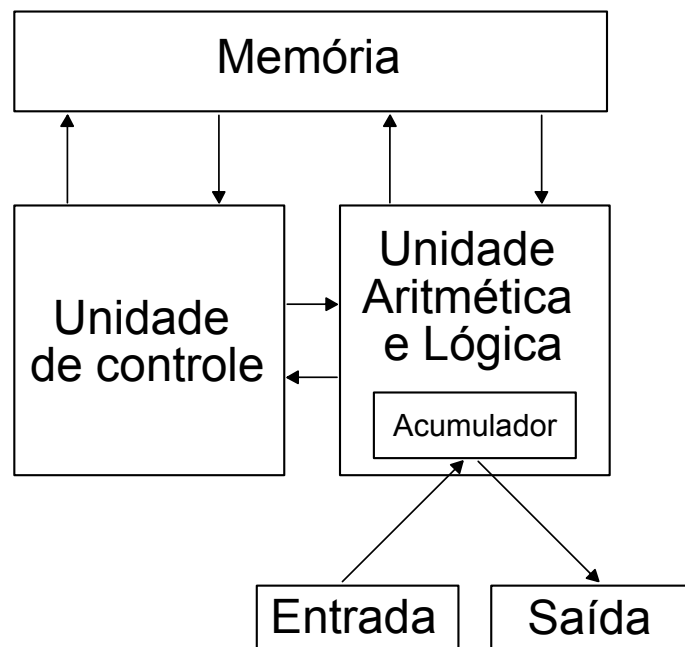
Insper

Decomposição das Instruções do
Tipo I

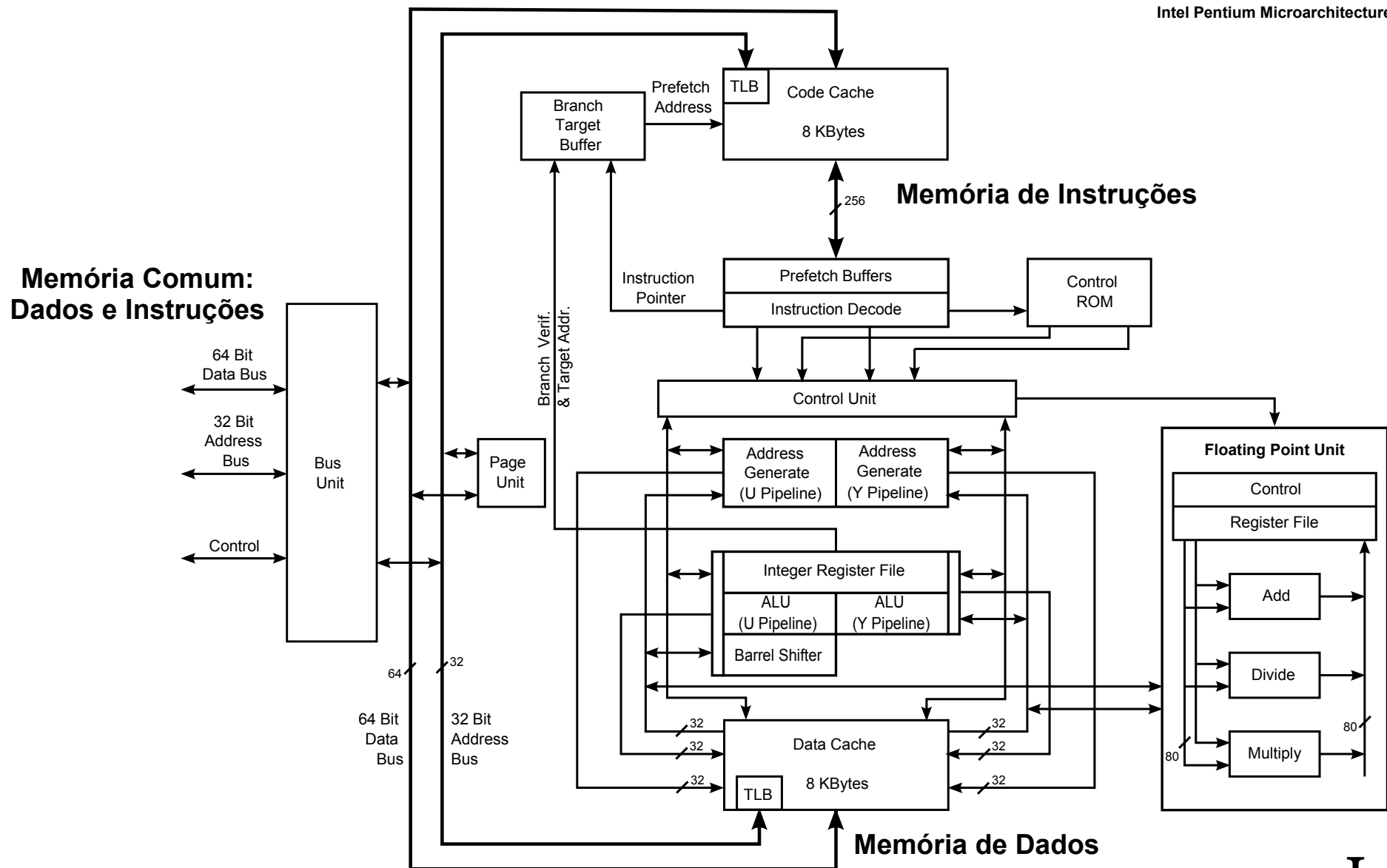
- 
- Tópicos:
 - Revisão;
 - Formato das instruções do tipo I;
 - Análise do funcionamento dessas instruções;
 - Esboço de fluxo de dados:
 - Que execute essas instruções.
 - Simulação manual do funcionamento do FD;
 - Implementação do FD em VHDL.

Revisão

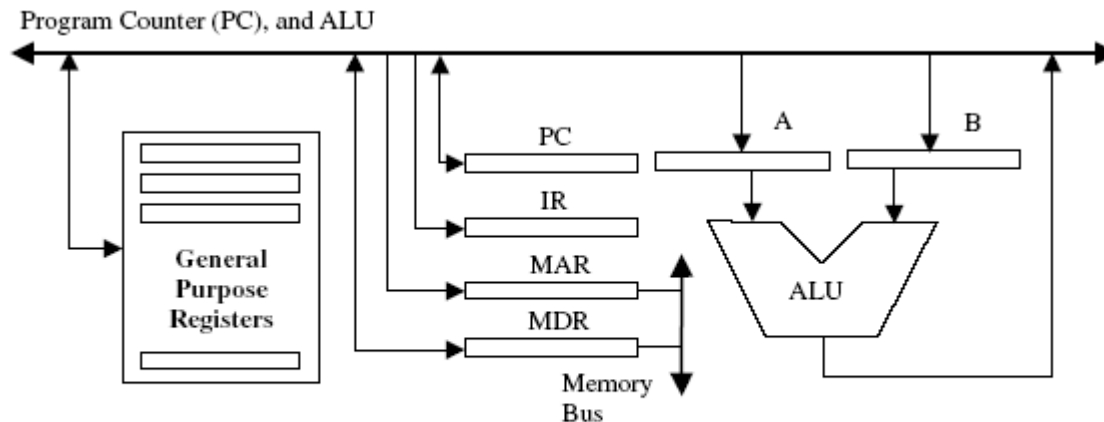
- Arquiteturas:
 - von Neumann (Princeton) e Harvard.



- Os processadores atuais são:
 - Uma arquitetura intermediária.

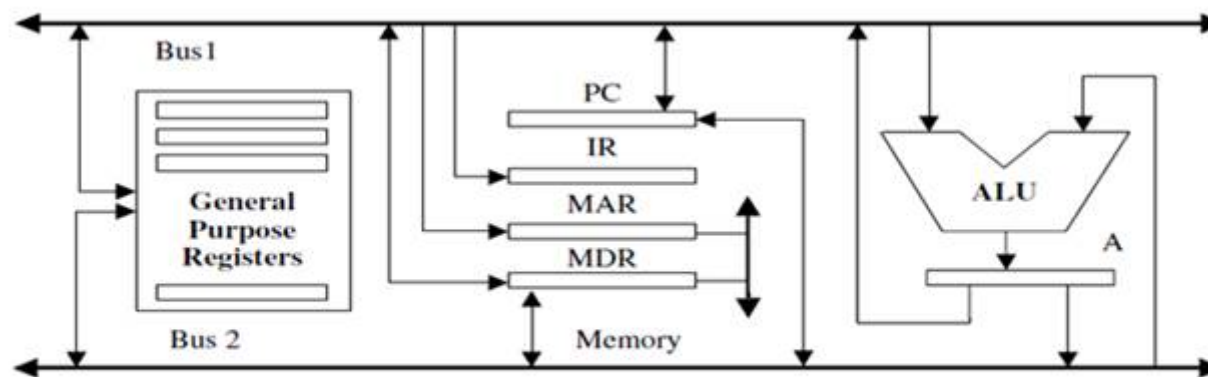


- Para aumentar o desempenho:
 - Pode-se aumentar a quantidade de barramentos.



One-bus datapath

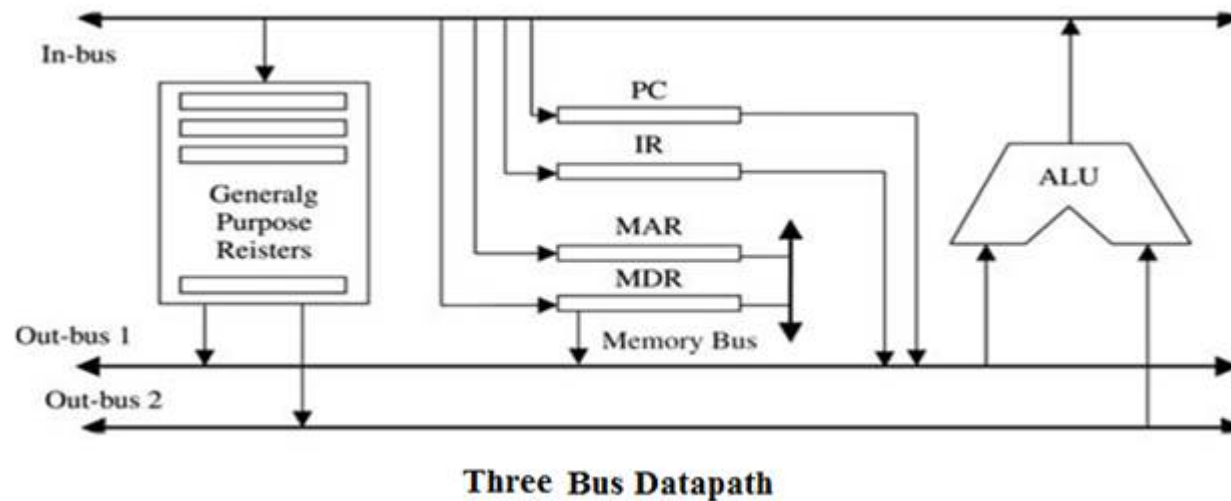
A ALU recebe os operandos em Sequência. Os registradores A e B são necessários.



Two Bus Datapath

A ALU recebe os operandos ao mesmo tempo. Não precisa dos registradores.

- Arquitetura com 3 barramentos:
 - A ALU recebe os operandos simultaneamente;
 - E também pode escrever o resultado no mesmo ciclo de clock;
 - De forma similar ao DLX.



Análise das Instruções tipo R

- Como vimos, essas instruções possuem a seguinte expressão geral:
 - $R[rd] \leftarrow R[s] \text{ operação } R[t];$
- Utilizando a RTN abstrata, poderíamos decompor em:
 - $IR \leftarrow M[PC] : PC \leftarrow PC + 4;$
 - $Inst(opc:=xx) \rightarrow R[rd] \leftarrow R[rs] \text{ operação } R[rt]$

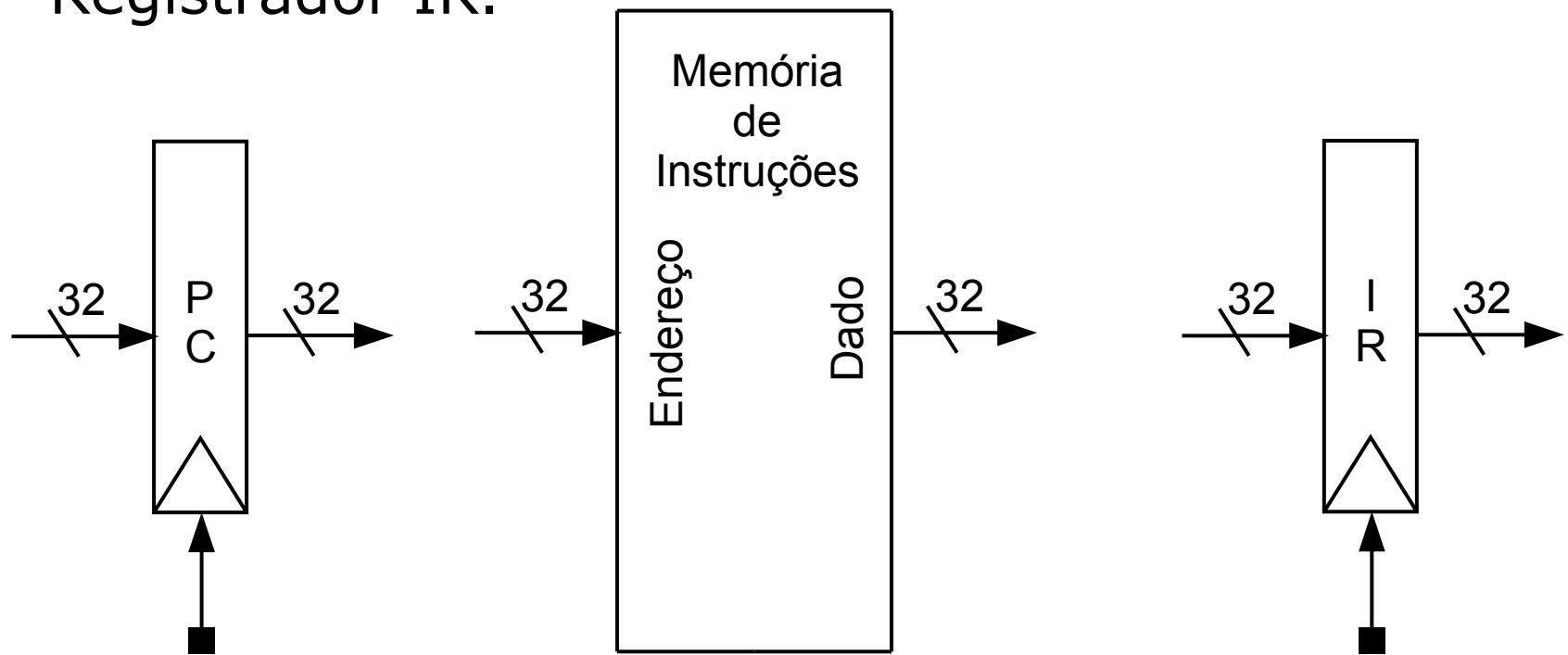
Onde: inst = instrução executada;

opc= opcode;

operação = função definida no campo funct.

- Considerando a RTN concreta e todas as etapas da execução:
 - Fetch: $M_{END} \leftarrow PC : IR \leftarrow M[PC];$
 - Decode:
 - $UC \leftarrow IR<31..26> : CTR_{ULA} \leftarrow IR<5..0> :$
 $RS_{END} \leftarrow IR<25..21> : RT_{END} \leftarrow IR<20..16> :$
 $RD_{END} \leftarrow IR<15..11>; C \leftarrow PC + 4;$
 - Execute: $ULA_A \leftarrow R[rs] : ULA_B \leftarrow R[rt]$
 - MemoryAccess: nada
 - WriteBack: $RD_{WR} \leftarrow True : R[rd] \leftarrow ULA_{OUT} : PC \leftarrow C$

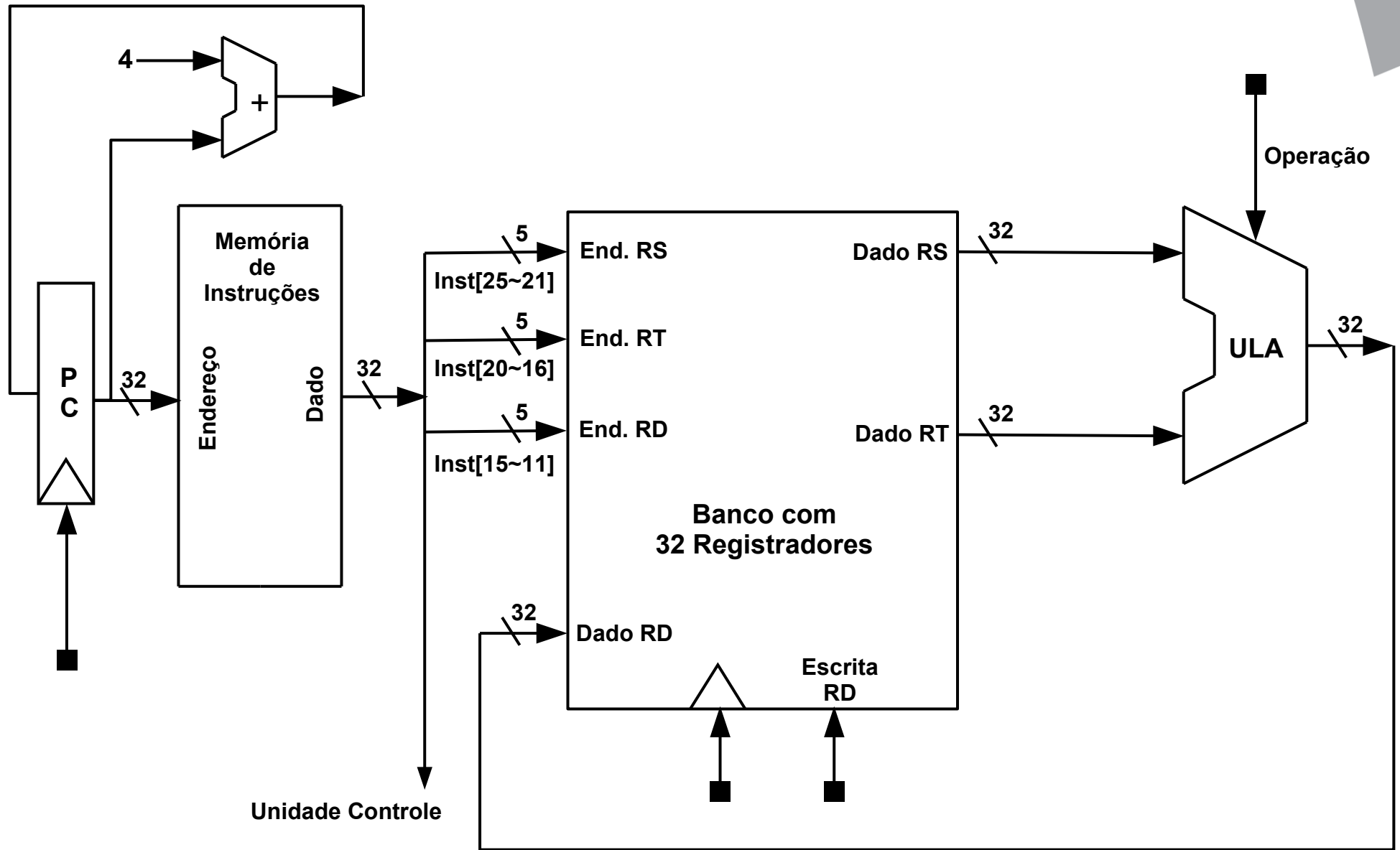
- Olhando apenas a etapa de “fetch”:
 - $M_{END} \leftarrow PC : IR \leftarrow M[PC];$
- Podemos definir as unidades funcionais:
 - Memória;
 - Registrador PC;
 - Registrador IR.



- Olhando apenas a etapa de “decode”:
 - $UC \leftarrow IR\langle 31..26 \rangle : CTR_{ULA} \leftarrow IR\langle 5..0 \rangle :$
 $RS_{END} \leftarrow IR\langle 25..21 \rangle : RT_{END} \leftarrow IR\langle 20..16 \rangle :$
 $RD_{END} \leftarrow IR\langle 15..11 \rangle ; C \leftarrow PC + 4 ;$
- Podemos definir as unidades funcionais:
 - Unidade de controle;
 - Banco de registradores;
 - ULA;
 - Somador para o PC.
- Como estamos trabalhando com:
 - Memórias separadas para instruções e dados;
 - Podemos retirar o IR e interligar direto no Banco de Registradores.

- A etapa de “execute”:
 - $ULA_A \leftarrow R[rs] : ULA_B \leftarrow R[rt]$
 - Não adiciona nenhum componente novo.
- A etapa de “memory access”:
 - Fica sem uso nesse tipo de instrução.
- A etapa de “write back”:
 - $RD_{WR} \leftarrow True : R[rd] \leftarrow ULA_{OUT} : PC \leftarrow C$
 - Só mostra as transferências:
 - Como o PC já é um registrador;
 - Não é necessário o registrador intermediário: C.

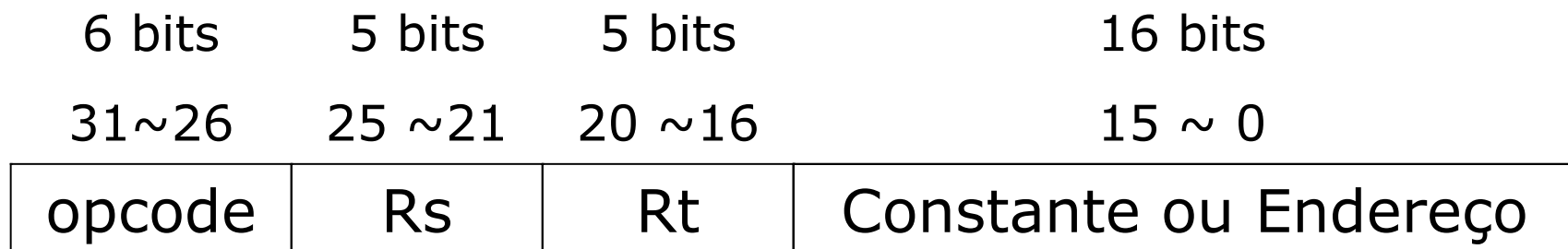
- A interligação final:



Instruções do tipo I

- As instruções desse tipo:
 - Não possuem só uma opção para o campo de opcode, como as do tipo R;
 - Possuem dois campos de registradores: Rs e Rt;
 - Possuem um campo com um valor imediato:
 - Esse valor, de 16 bits, está codificado na instrução;
 - E representa um inteiro com sinal.
 - São utilizadas para:
 - Carregar valores imediatos (constantes) nos registradores;
 - Fazer operações lógicas e aritméticas com valores imediatos (addi rs, rt, #valor);
 - Escrever na memória (sw);
 - Ler da memória (lw);
 - Fazer desvios condicionais (beq).

- As instruções desse tipo, para a nossa implementação, são:
 - Load word: lw;
 - Store word: sw;
 - Branch on equal: beq.
- O formato das instruções tipo I:



- Exemplo de Branch on Equal:

- ASM: beq \$rs, \$rt, imediato

- Sua operação:

- $PC += (R[rs] == R[rt] ? 4 + distancia : 4);$

- Onde:

- $distancia = (14 @ imediato < 15 > \#imediato < 15..0 > \#0 \#0);$

- Para a instrução: beq \$t0, \$t1, imediato:

- Qual seria a sua codificação?

6 bits

5 bits

5 bits

16 bits

31~26

25 ~21

20 ~16

15 ~ 0

opcode	Rs	Rt	Constante ou Endereço

- Exemplo de Load:

- ASM: `lw $rt, imediato($rs);`
- Operação:
 - $R[rt] = M[R[rs] + \text{extSinal}(\text{imediato})]$
 - Onde:
 - $\text{extSinal} = (16 @ \text{imediato} < 15 > \# \text{imediato} < 15..0 >)$
- Para a instrução: `lw $t0, -4($sp):`
 - Qual seria a sua codificação?

6 bits	5 bits	5 bits	16 bits
31~26	25 ~21	20 ~16	15 ~ 0
opcode	Rs	Rt	Constante ou Endereço

- Exemplo de Store:

- ASM: `sw $rt, imediato($rs);`

- Operação:

- $M[R[rs] + \text{extSinal}(\text{imediato})] = R[rt]$

- Onde:

- $\text{extSinal} = (16 @ \text{imediato} < 15 > \# \text{imediato} < 15..0 >)$

- Para a instrução: `sw $t1, 4($sp):`

- Qual seria a sua codificação?

6 bits

5 bits

5 bits

16 bits

31~26

25 ~21

20 ~16

15 ~ 0

opcode	Rs	Rt	Constante ou Endereço

- Faça a análise das instruções anteriores:
 - Na sequência: LW, SW e BEQ (fica mais fácil);
 - Descreva o funcionamento RTL da arquitetura para implementá-las;
 - Esboce o caminho de dados;
 - Simule o funcionamento de cada instrução;
 - Se não estiver bom, reinicie o processo.
- Com o caminho de dados pronto:
 - Implemente em VHDL e simule.
- Quando estiver funcional:
 - Mescle com o caminho de dados para as instruções do tipo R.

Insper

www.insper.edu.br

