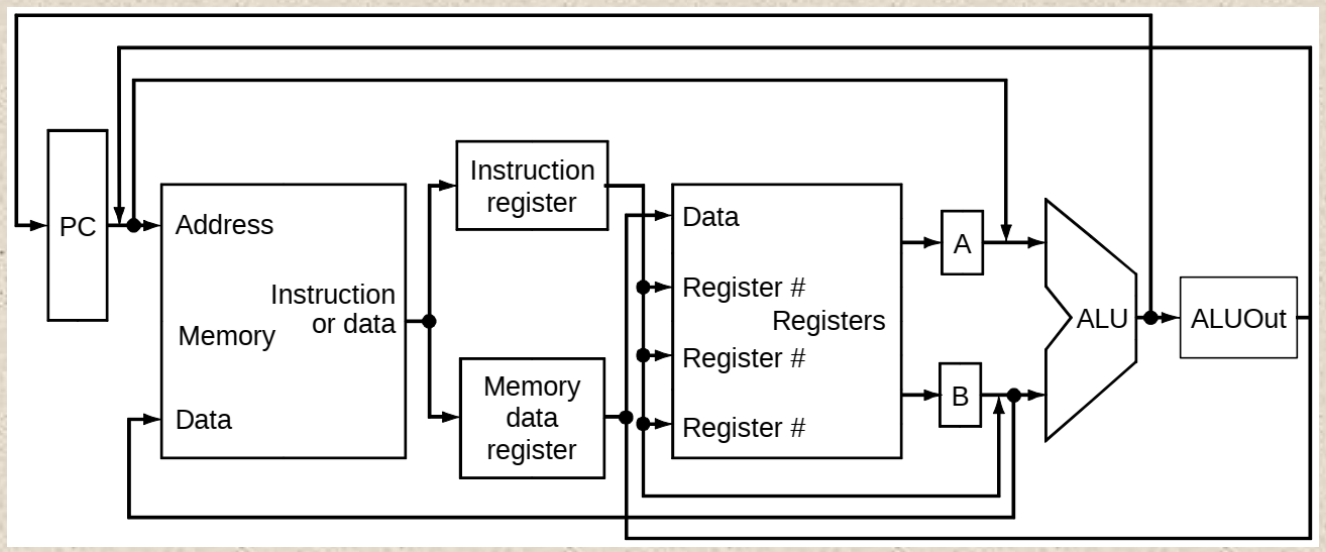
MIPS multiciclo

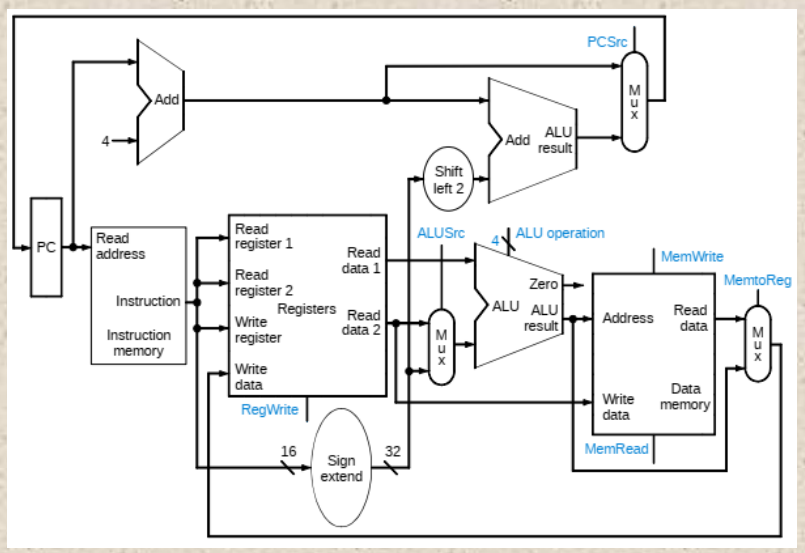
Para onde estamos indo

- Problemas do monociclo:
  - Se tivéssemos uma instrução mais complicada como de pontos flutuantes?
  - Alto tempo de execução e desperdício de uso de elementos.
- Uma solução:
  - Usar um tempo de ciclo de clock "menor";
  - Instruções diferentes levam diferentes números de ciclos;
  - Um caminho de dados "multiciclo":



Abordagem multiciclo

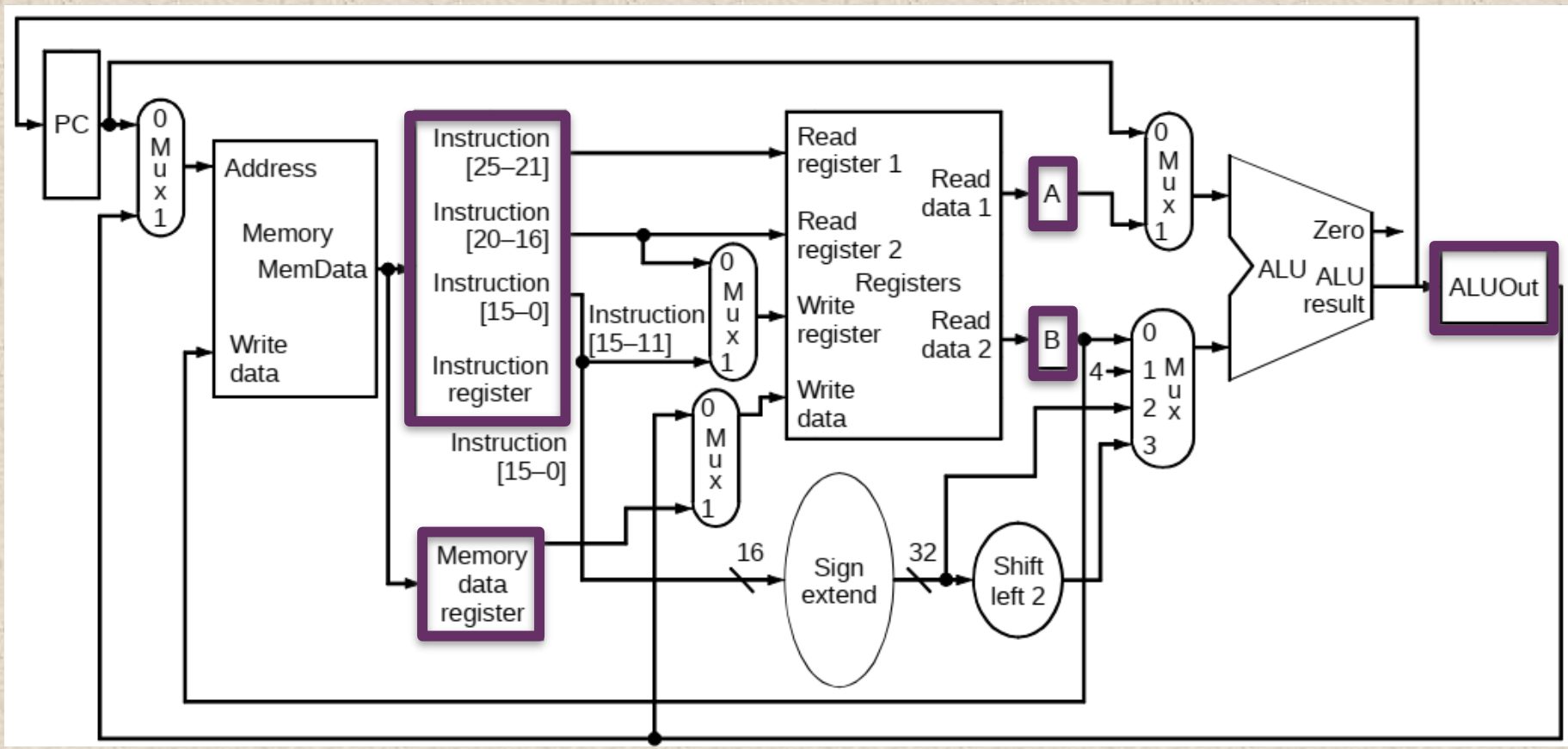
- Estaremos reutilizando unidades funcionais
  - ULA usada para calcular o endereço e incrementar o PC;
  - Memória usada para instrução e dados (dois acessos em ciclos diferentes: um para instrução e outro para dados).
- Nossos sinais de controle não serão determinados diretamente por instrução, mas também pelo ciclo de execução
  - por exemplo, o que a ULA deve fazer para uma instrução de "subtrair"? PC + 4 e subtração
- Usaremos uma máquina de estados finitos para controle.



Abordagem multiciclo

- Divida as instruções em etapas, cada etapa leva um ciclo
  - Equilibrar a quantidade de trabalho a ser feito;
    - Se acesso a memória e a registradores leva muito tempo, não usar ULA no mesmo ciclo.
  - Restringir cada ciclo para usar apenas uma unidade funcional principal.
- No fim de um ciclo
  - Armazenar valores para uso em ciclos posteriores (resultados parciais);
  - Introduzir registradores "internos" adicionais, diferente dos de propósito geral.

Abordagem multiciclo



Ideia por trás da abordagem multiciclo

- Definimos cada instrução a partir da perspectiva ISA;
- Divida em etapas/ciclos seguindo nossa regra de que os dados fluem em mais de uma unidade funcional principal (banco de registradores, memória e ULA);
- Introduzir novos registradores conforme necessário (por exemplo, A, B, ALUOut, MDR, etc.)
- Por fim, tente incluir o máximo de trabalho em cada etapa (evite ciclos desnecessários) enquanto também tenta compartilhar etapas sempre que possível
  - (minimiza o controle, ajuda a simplificar a solução)
- Resultado: Livro de implementação multiciclo (regra de onde o dado sai, por onde ele passa, e onde ele chega)

Instruções pela perspectiva ISA

- Considere cada instrução da perspectiva do ISA;
- Exemplo:

op	rs	rt	rd	shm	fn
6	5	5	5	5	6
31-26	25-21	20-16	15-11	10-6	5-0

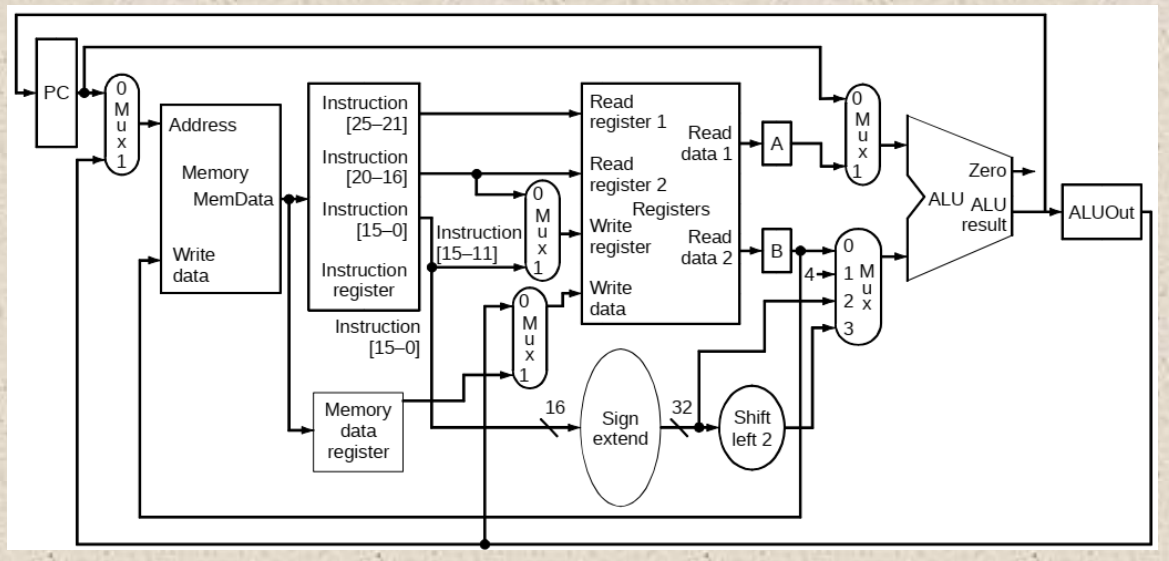
  - A instrução add altera um registrador;
    - Registrador especificado pelos bits 15:11 da instrução;
    - Instrução especificada pelo PC;
  - Novo valor é a soma (“op”) de dois registradores;
  - Registradores especificados pelos bits 25:21 e 20:16 da instrução.
  - Reg[Memoria[PC][15:11]] <= Reg[Memoria[PC][25:21]] op Reg[Memoria[PC][20:16]]
  - Para conseguir isso, devemos dividir a instrução.
    - (como introduzir variáveis ao programar)

Decompondo uma instrução

- Definição ISA de aritmética:

  Reg[Memoria[PC][15:11]] <= Reg[Memoria[PC][25:21]] op Reg[Memória[PC][20:16]]

- Pode se dividir em:
  - IR <= Memoria[PC]
  - A <= Reg[IR[25:21]]
  - B <= Reg[IR[20:16]]
  - ALUOut <= A op B
  - Reg[IR[15:11]] <= ALUOut



- Esquecemos uma parte importante da definição de aritmética!
  - PC <= PC + 4

Cinco etapas de execução

1. Busca de instruções na memória para registrador intermediário;
2. Decodificação de instruções e busca de registradores;
3. Execução, cálculo de endereço de memória ou desvio;
4. Acesso à memória ou conclusão de instruções do tipo R;
5. Etapa de *write-back.*

INSTRUÇÕES LEVAM DE 3 - 5 CICLOS!

Etapa 1: Busca de instruções

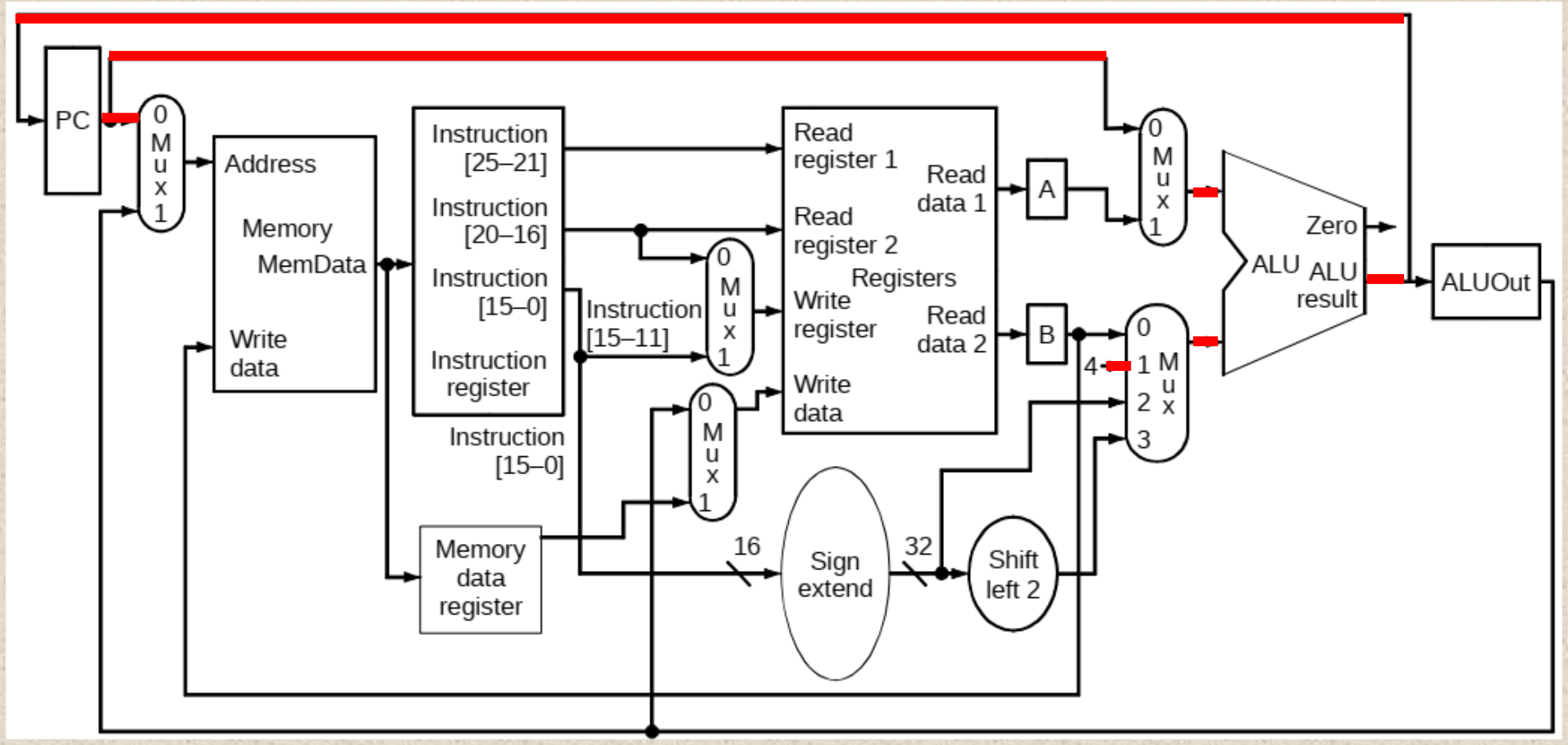
- Use o PC para obter as instruções e coloque-as no Registro de Instruções (IR, do inglês *instruction register*);
  - Uso da memória
- Incremente o PC em 4 e coloque o resultado de volta no PC;
  - Uso da ULA
- Pode ser descrito sucintamente usando RTL "*Register-Transfer Language*"

IR <= Memória[PC];

PC <= PC + 4;

Qual é a vantagem de atualizar o PC agora?

Etapa 1: Busca de instruções



Etapa 2: Decodificação de instruções e busca de registradores

- Leia os registradores rs e rt caso precisemos deles;
- Espera necessária para instruções do tipo R (para fazer a operação) e para instruções lw e sw (soma para cálculo de endereço);
- Calcular o endereço da ramificação caso a instrução seja uma ramificação
- RTL:

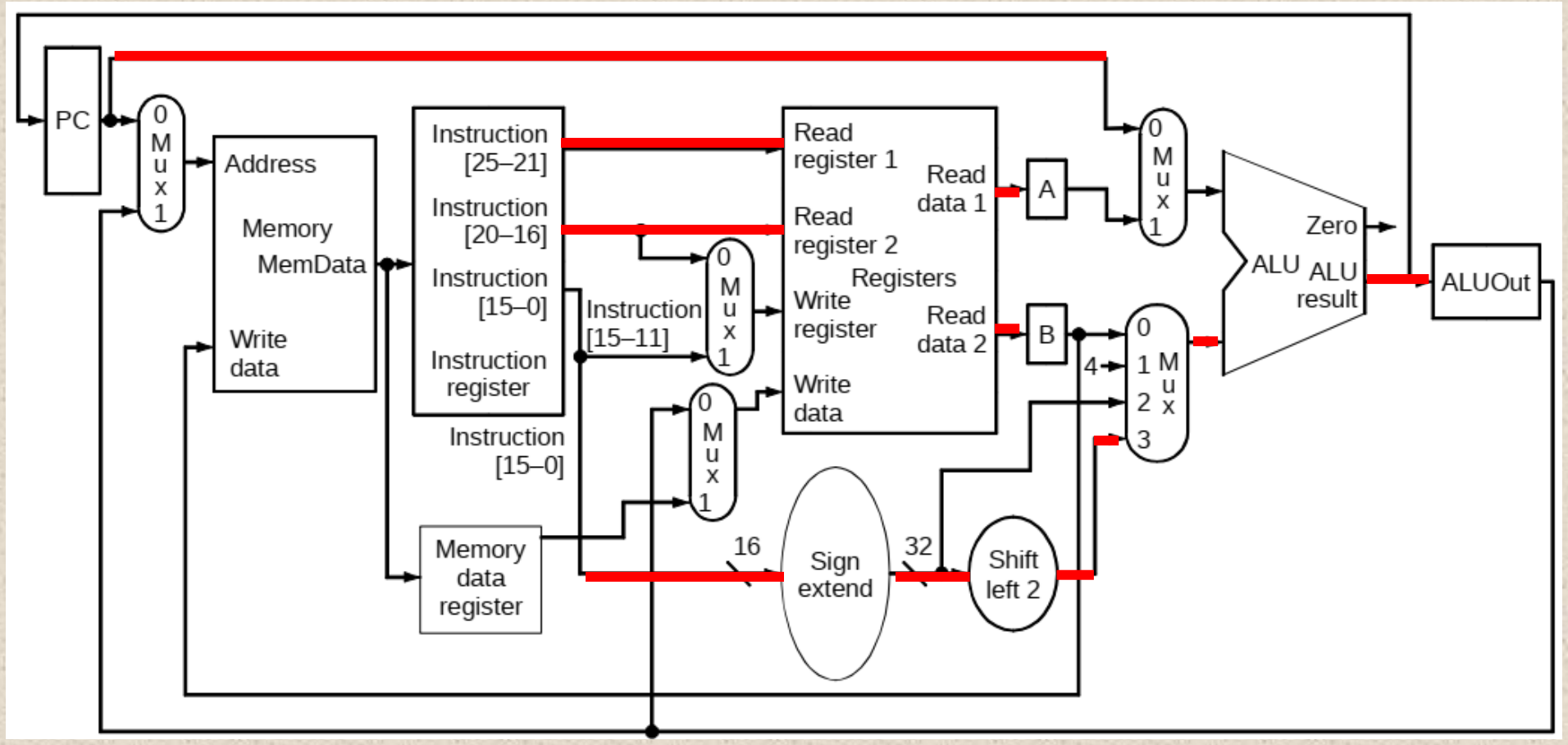
  A <= Reg[IR[25:21]];

  B <= Reg[IR[20:16]];

  ALUOut <= PC + (sinal-extend(IR[15:0]) << 2);

- Não estamos definindo nenhuma linha de controle com base no tipo de instrução
  - (estamos ocupados "decodificando" nossa lógica de controle).

Etapa 2: Decodificação de instruções e busca de registradores



Etapa 3 (dependente da instrução)

- ALU está executando uma das três funções, com base no tipo de instrução
- Referência de Memória:

  ALUOut <= A + sinal-extend(IR[15:0]);

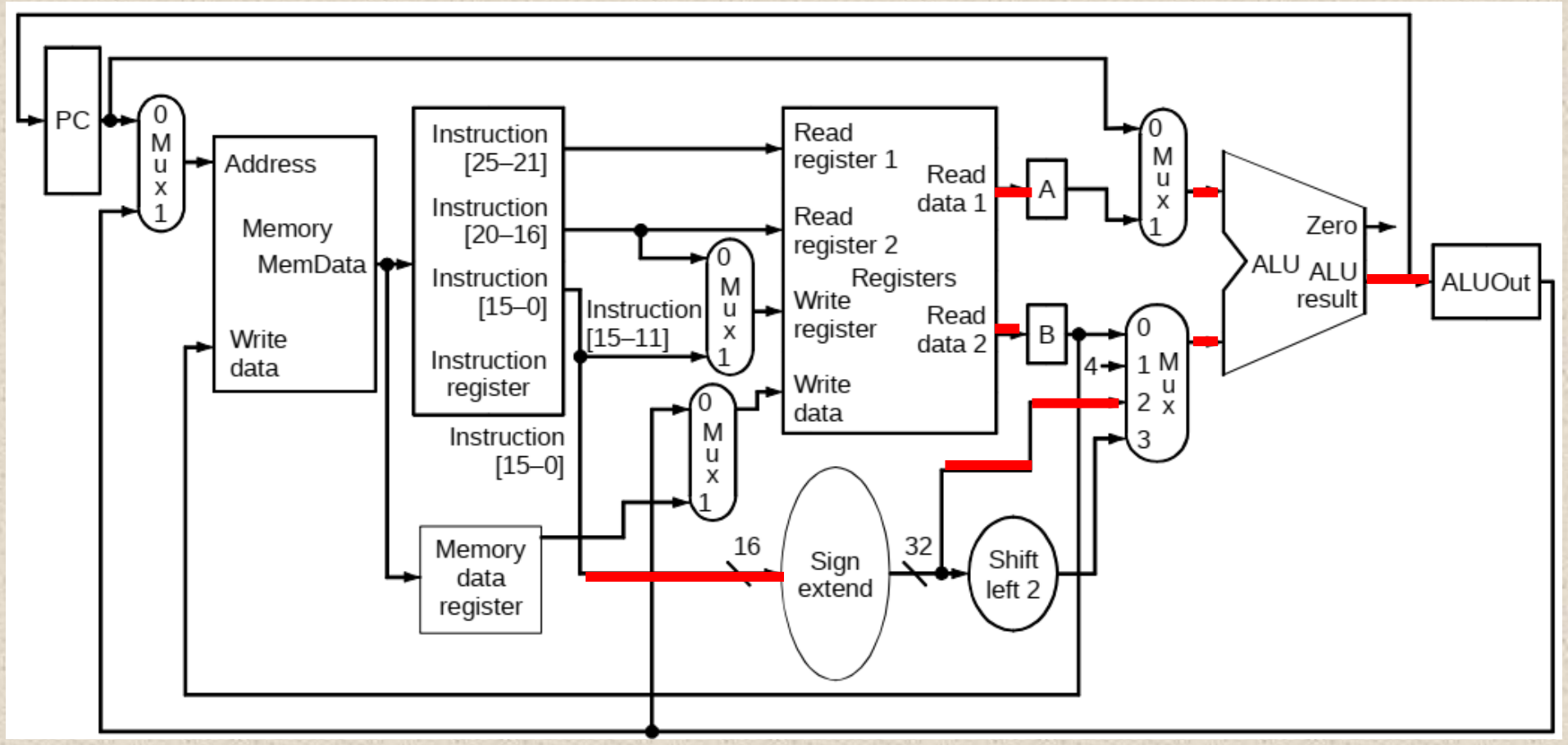
- Tipo R:

  ALUOut <= A op B;

- Desvio:

  se (A==B) PC <= ALUOut;

Etapa 3 (dependente da instrução)



Etapa 4 (Tipo R ou de acesso à memória)

- Instruções de desvio não chegam a esse passo;
- Carrega e armazena a memória de acesso

  MDR <= Memoria[ALUOut]; (MDR – registrador intermediário: *memory data register*)

  ou

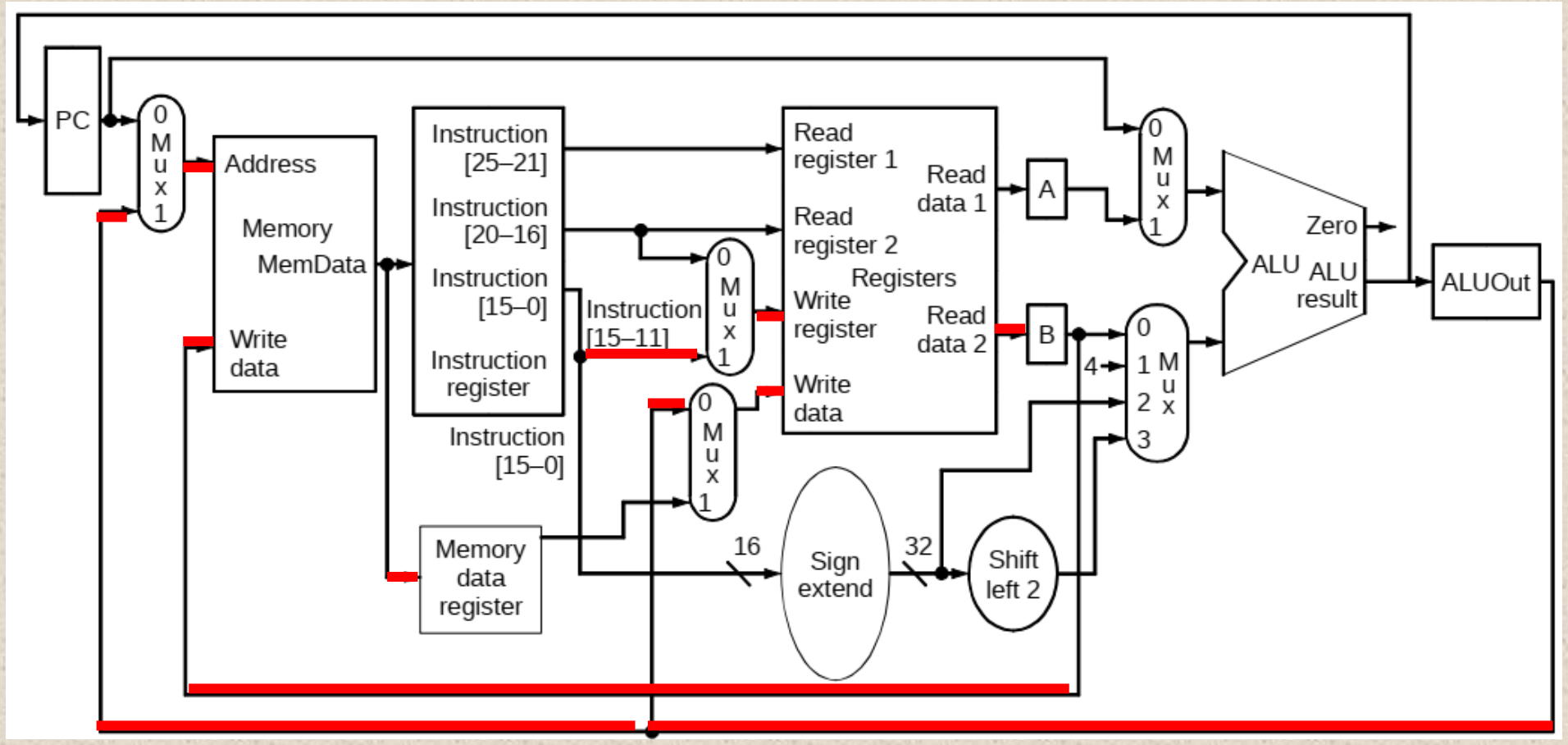
  Memoria[ALUOut] <= B;

- Fim das instruções do tipo R

  Reg[IR[15:11]] <= ALUOut;

- A gravação realmente ocorre no final do ciclo na borda

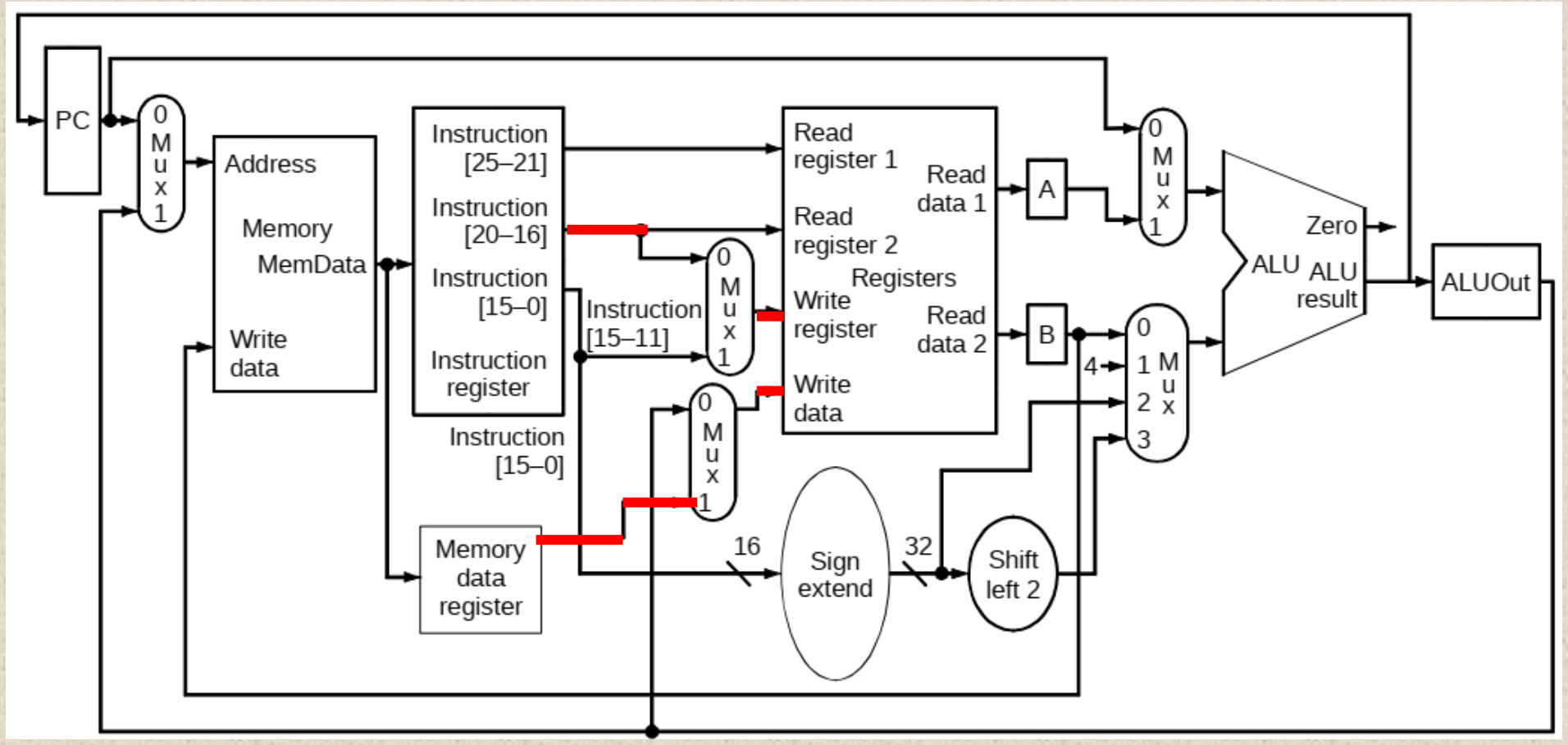
Etapa 4 (Tipo R ou de acesso à memória)



Etapa 5 (*write-back*)

- Reg[IR[20:16]] <= MDR;
- Qual instrução precisa disso?

Etapa 5 (*write-back*)

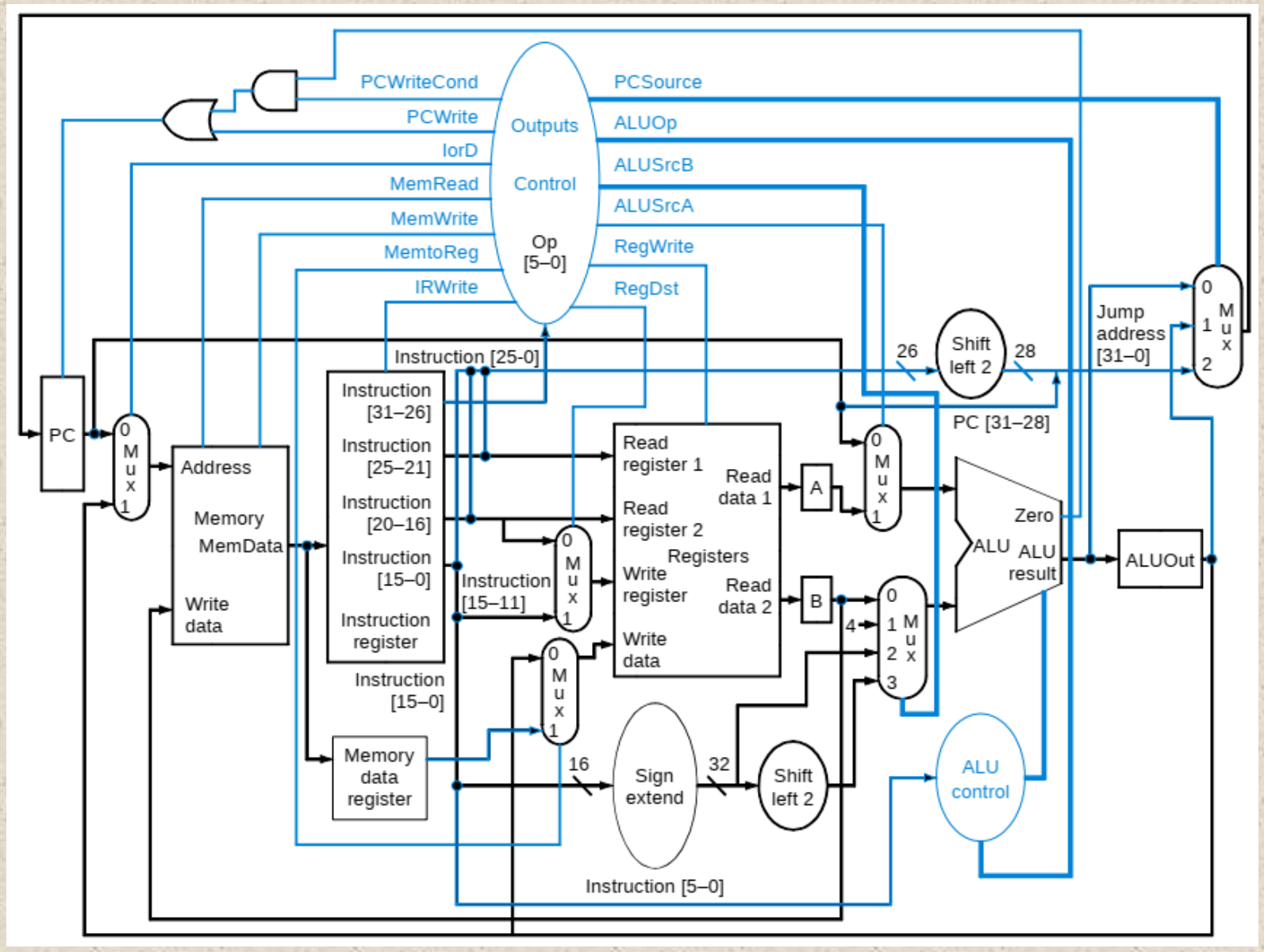


Resumo

Step name	Action for R-type instructions	Action for memory-reference instructions	Action for branches	Action for jumps
Instruction fetch	IR <= Memory[PC] PC <= PC + 4			
Instruction decode/register fetch	A <= Reg [IR[25:21]] B <= Reg [IR[20:16]] ALUOut <= PC + (sign-extend (IR[15:0]) << 2)			
Execution, address computation, branch/jump completion	ALUOut <= A op B	ALUOut <= A + sign-extend (IR[15:0])	If (A == B) PC <= ALUOut	PC <= {PC [31:28], (IR[25:0]),2'b00)}
Memory access or R-type completion	Reg [IR[15:11]] <= ALUOut	Load: MDR <= Memory[ALUOut] or Store: Memory [ALUOut] <= B		
Memory read completion		Load: Reg[IR[20:16]] <= MDR		

FIGURE 5.30 Summary of the steps taken to execute any instruction class. Instructions take from three to five execution steps. The first two steps are independent of the instruction class. After these steps, an instruction takes from one to three more cycles to complete, depending on the instruction class. The empty entries for the Memory access step or the Memory read completion step indicate that the particular instruction class takes fewer cycles. In a multicycle implementation, a new instruction will be started as soon as the current instruction completes, so these cycles are not idle or wasted. As mentioned earlier, the register file actually reads every cycle, but as long as the IR does not change, the values read from the register file are identical. In particular, the value read into register B during the Instruction decode stage, for a branch or R-type instruction, is the same as the value stored into B during the Execution stage and then used in the Memory access stage for a store word instruction.

Unidade de controle

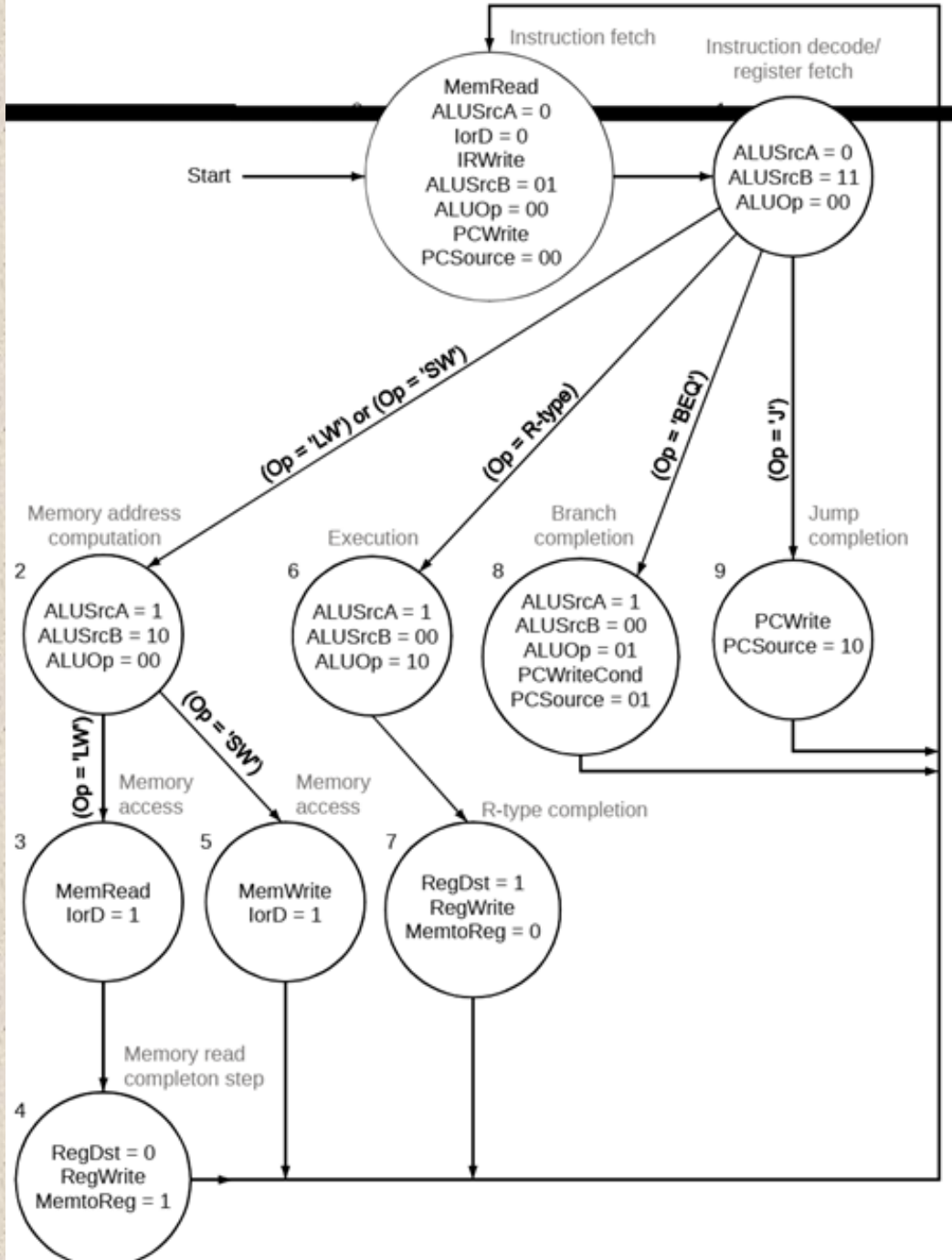


Implementando o controle

- O valor dos sinais de controle depende de:
  - qual instrução está sendo executada
  - qual etapa/ciclo está sendo executada
- Use as informações que acumulamos para especificar uma máquina de estado finito
  - especificar a máquina de estados finitos graficamente, ou
  - usar microprogramação
- A implementação pode ser derivada da especificação

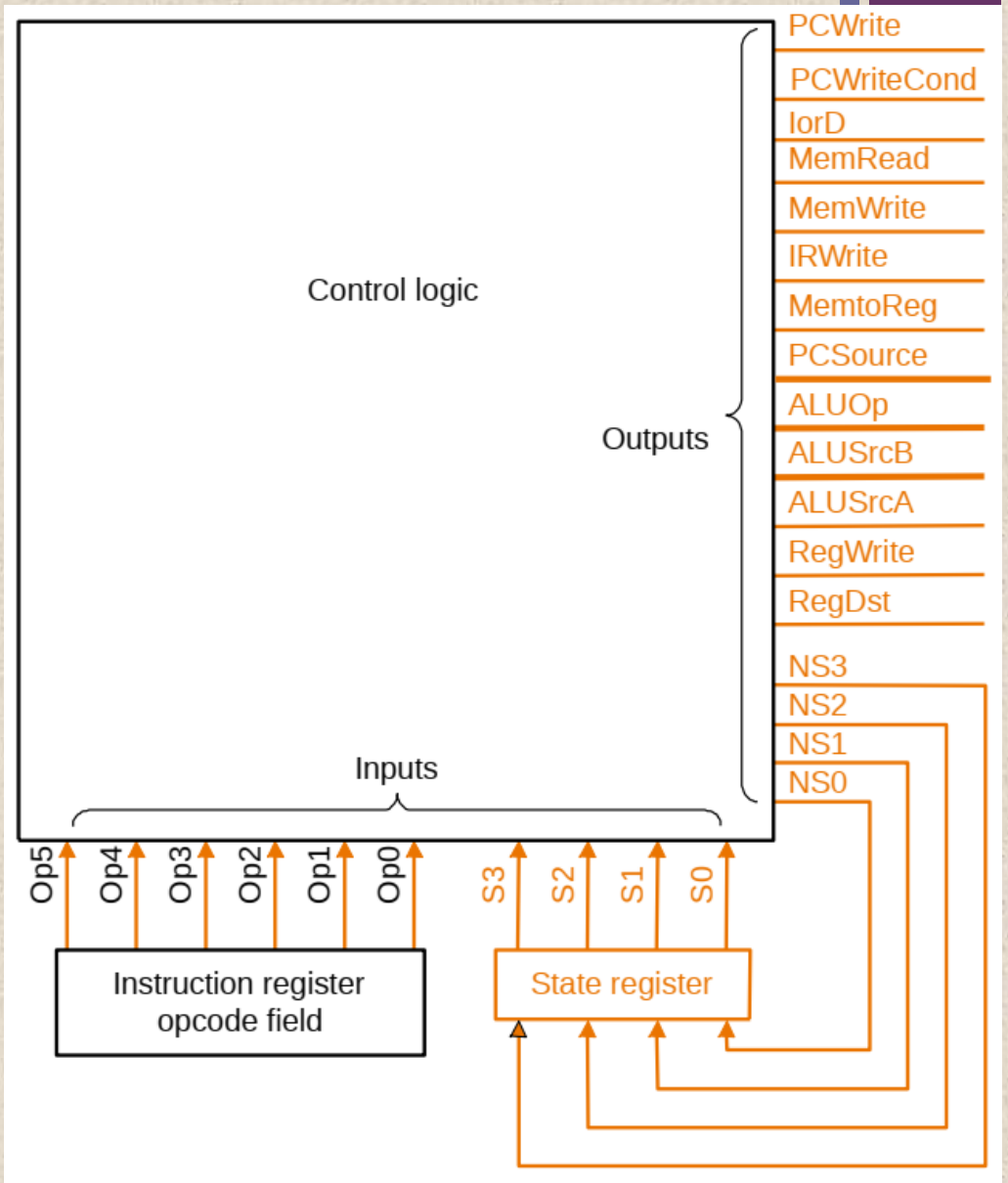
Especificação gráfica de MEF

- Observação:
  - não importa se não for mencionado
  - setado se apenas o nome
  - caso contrário, valor exato
- De quantos bits de estado precisaremos?



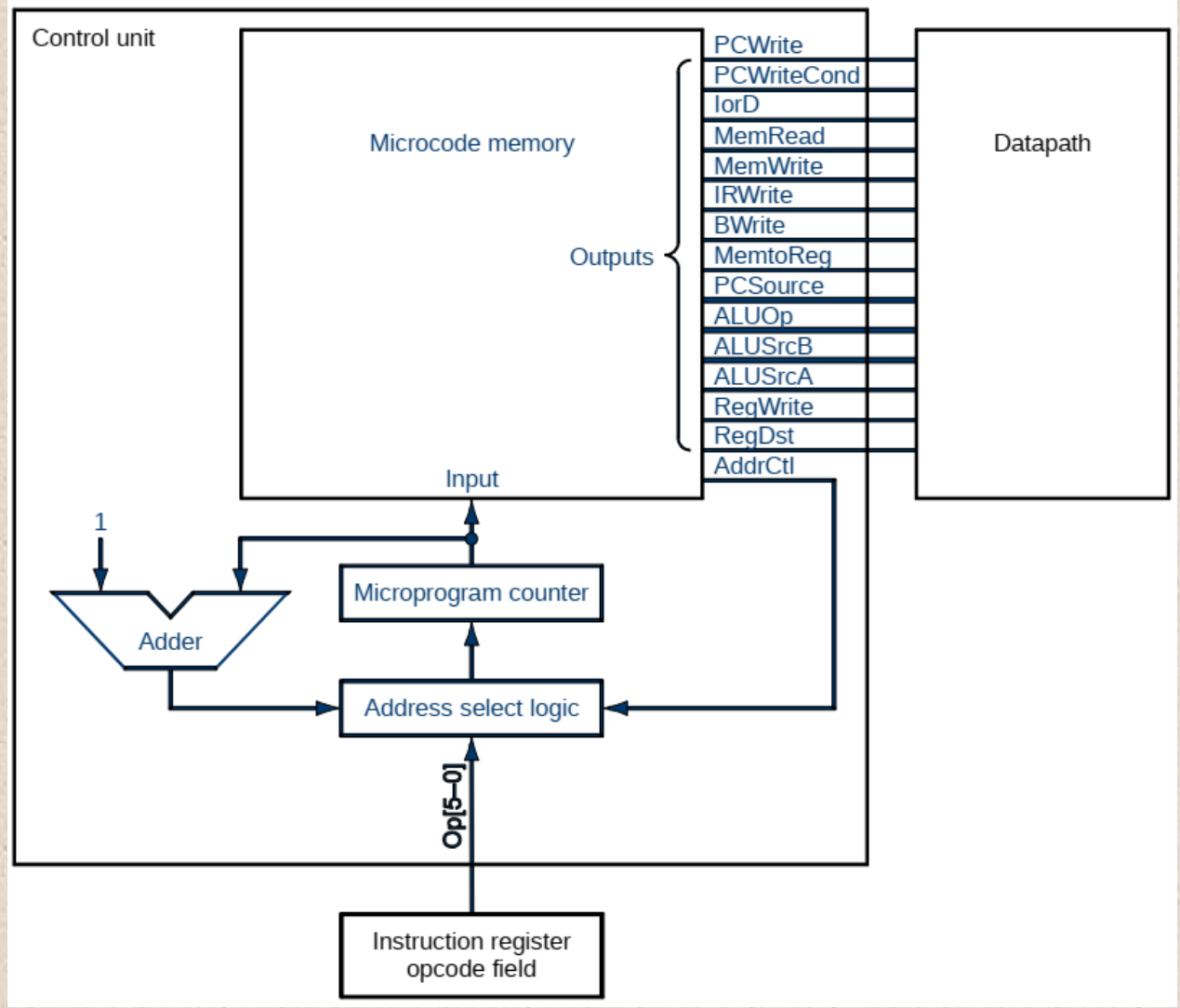
Máquina de estados finitos para controle

- Implementação



Microprogramação

- CPU dentro da CPU;
- Para CISC;
- O que são as "microinstruções"?



Codificação máxima *vs* mínima

- Sem codificação:
  - 1 bit para cada operação de caminho de dados
  - mais rápido, requer mais memória (lógica)
  - usado para Vax 780 – surpreendentes 400K de memória!
- Muita codificação:
  - enviar as microinstruções através da lógica para obter sinais de controle
  - usa menos memória, mais lento
- Contexto histórico do CISC:
  - Muita lógica para colocar em um único chip com todo o resto
  - Uso de uma ROM (ou mesmo RAM) para armazenar o microcódigo
  - É fácil adicionar novas instruções

Organização MIPS

Agradeço a Prof. Dr. Fábio A. M. Cappabianco, pelos materiais disponibilizados.