Parte IV: DataPath e Controle

Introdução

Introdução

- Objetivo: Construir um processador para execução de instruções.
  - Caminho de dados
  - Unidade de controle

- O tempo do ciclo de clock e o número de ciclos por instrução são determinados pela implementação do processador.

- Projeto baseado no conjunto de instruções básicas do MIPS
  - Instruções de acesso à memória (lw e sw)
  - Instruções aritméticas (add, sub, and, or e slt)
  - Instruções de branch e jump (beq e j)

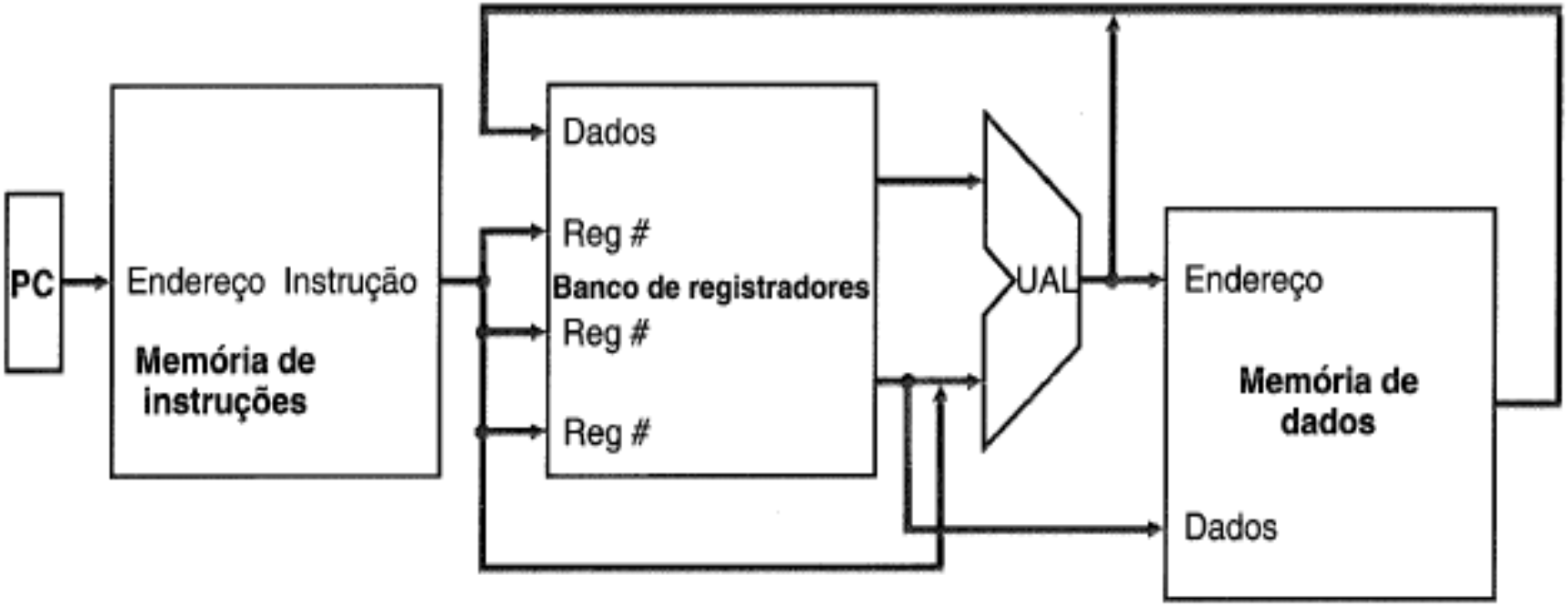
- Conjunto completo de instruções podem ser implementadas de forma similar.
- Outras arquiteturas seguem os mesmos fundamentos.

Visão Geral

- Para qualquer instrução:
  - Passo 1: Enviar PC (Program counter) para a memória que contém o programa, e trazer para o caminho de dados a instrução armazenada nesse endereço da memória.
  - Passo 2: Ler um (ex. lw, sw) ou dois registradores (ex. Add, sub), usando os campos correspondentes da instrução lida.

- Outras ações dependem da classe da instrução
  - Há muitas semelhanças entre elas.
  - Todas usam ULA após lidos os registradores.
    - As de acesso à mem. para cálculo do endereço
    - As aritméticas para a própria operação
    - As de desvios condicionais para a comparação
  - Similaridade facilita o projeto do HW!
- Depois da ULA, ações mudam um pouco
  - Acesso à mem endereça a mesma
  - Aritmética tem retorno em registradores
  - Desvios alteram o PC para a próxima instrução

Visão Geral


PC
Endereço Instrução
Memória de instruções
Dados
Reg #
Banco de registradores
Reg #
Reg #
UAL
Endereço
Memória de dados
Dados

Considerações sobre Lógica e Clock

- Sistemas digitais são compostos por elementos:
  - Combinacionais, cujas saídas só dependem dos sinais de entrada
  - De estado, que têm capacidade de armazenar estado

- Clocks servem para determinar o momento de atualização de estado.

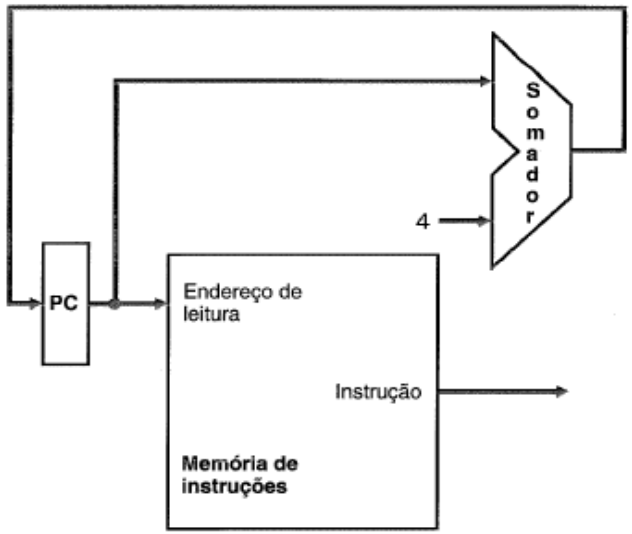
- Política de temporização para clocks é necessária para que circuitos funcionem com precisão.

- O processador pode ser implementado em um único ciclo de clock ou em vários
  - Um ciclo é mais simples, mas mostrou-se mais lento.
  - Vários ciclos exige maior controle.

MIPS Uniciclo

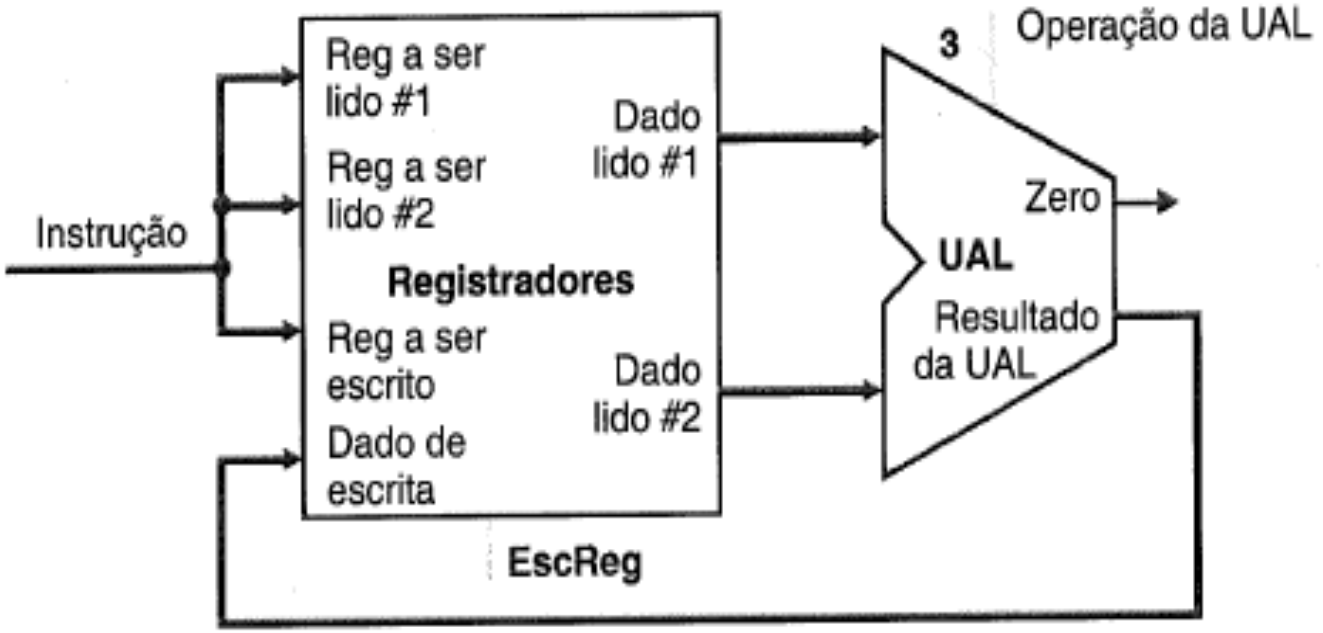
- Caminho de Dados
  - Interligação dos componentes básicos

- Busca de instruções necessita de:
  - Memória, onde instruções são armazenadas
  - PC, registrador para manter estado da próxima instrução a ser executada
  - Somador, para inc. do PC (+4 p/ restrição de alinham.)



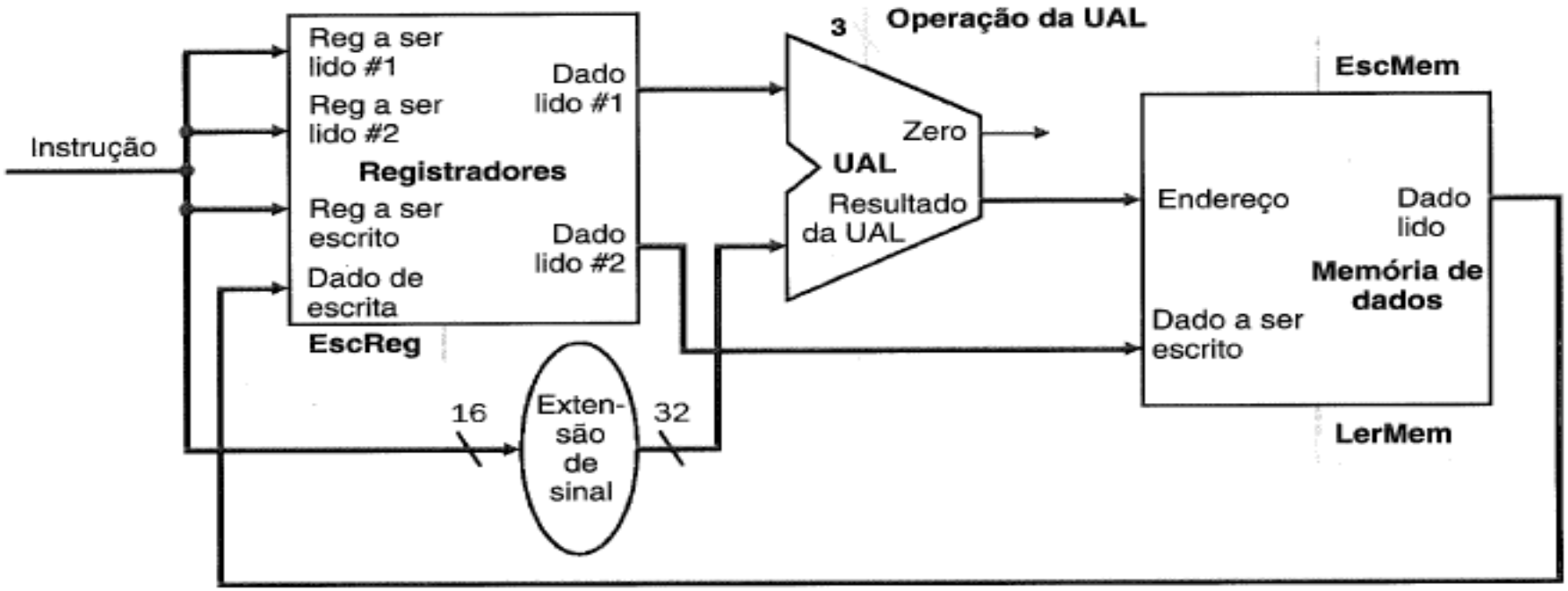
MIPS Uniciclo

- Instruções do tipo R
  - Lê registradores, operação ULA e escreve em registrador.
  - Registradores ficam em um banco de registradores
    - são acessados pelo seu número
    - Todo reg. De entrada gera seu dado na saída
  - Registrado a ser escrito precisa do número e do dado.
  - A ULA tem sua operação controlada por 3 bits (parte III)



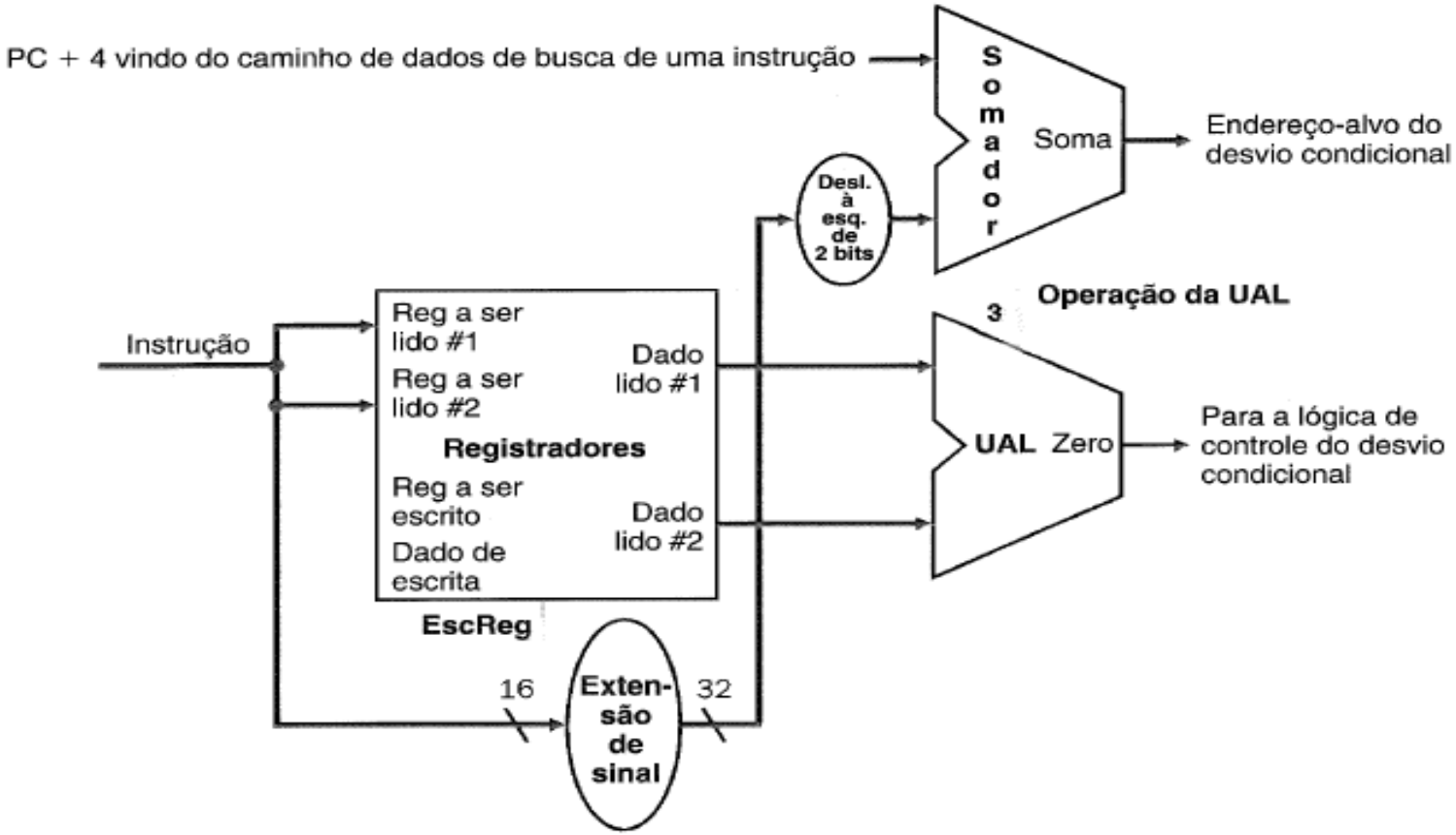
MIPS Uniciclo

- Operações de memória (ex. Lw $t1, desl($t2))
  - Calculam o end. De memória somando o conteúdo do registrador com o campo de deslocamento.
  - Se for store, valor a ser armazenado precisa ser lido do banco de registradores.
  - Se for load, o valor da memória deve ser escrito no banco de registradores.
  - O campo de desloc. precisa ser estendido de 16 para 32 bits.



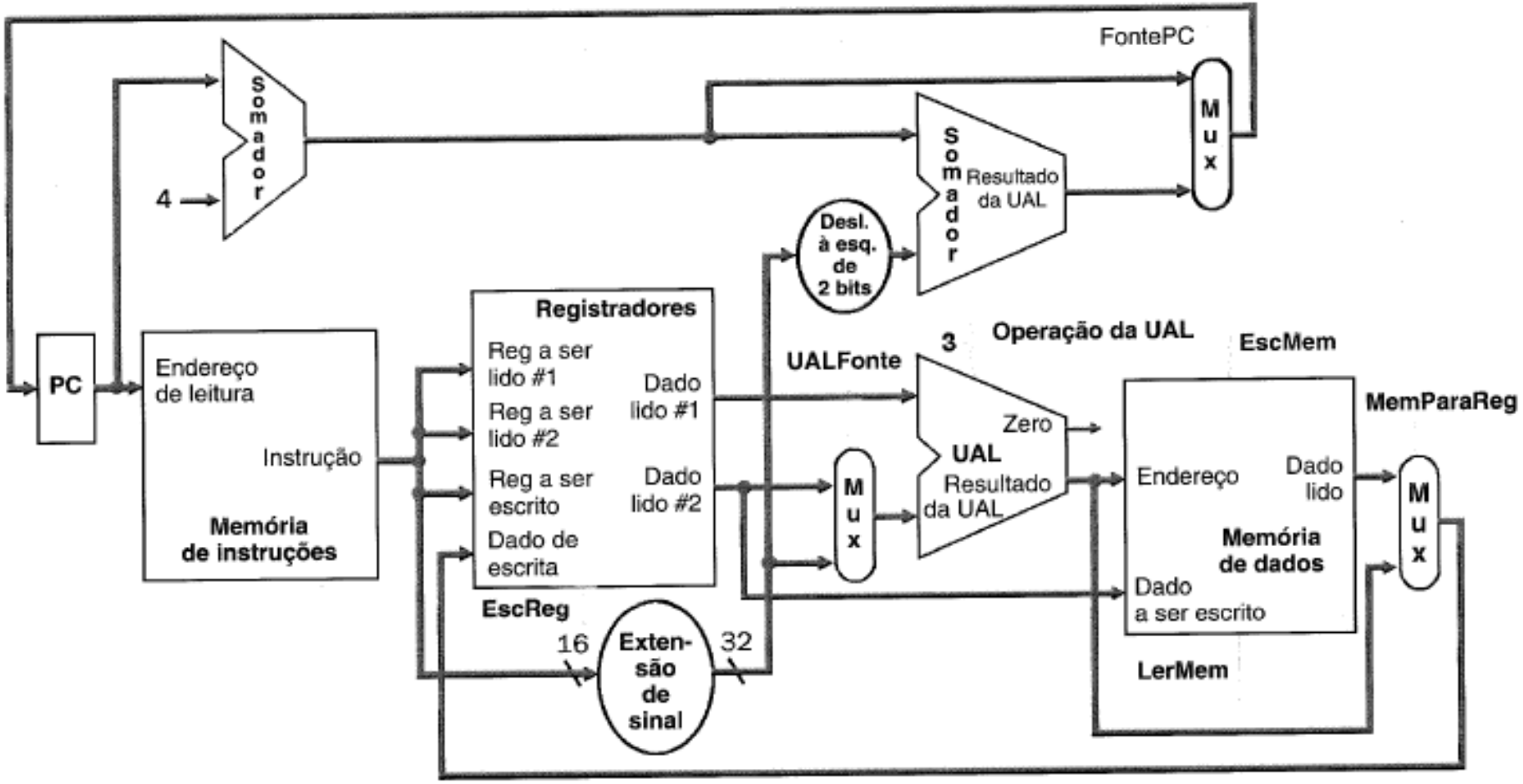
MIPS Uniciclo

- Instrução de branch condicional (beq)
  - Um desloc. (16 bits) como end. alvo deve ser estendido e somado ao PC
    - Como a ref é a próxima instrução, PC+4
    - Desloc. Deve ser deslocado 2 bits à esquerda (x4)
  - Próximo end. depende da comparação de 2 regs
    - Subtração na ALU com indicador de 0



MIPS Uniciclo

- Composição
  - Há reaproveitamento de componentes comuns, o que gera múltiplas entradas e saídas dos mesmos.
    - Seleção feita por multiplexadores (seletor de dados)



MIPS Uniciclo

- Controle da ULA
  - Já vimos anteriormente (Parte III) que a ULA tem entradas de controle que define a operação a ser executada.
  - São usadas diretamente pelas instruções aritméticas, e a operação é determinada pelo campo funct (função).
  - Operações de memória usam ULA para somar end. De memória.
  - Para o brach, ULA faz uma subtração (comparação se 0)

Entrada de controle da UAL	Função
000	AND
001	OR
010	Soma
110	Subtração
111	Set less than

MIPS Uniciclo

- Controle ULA
  - Uma unidade de controle pode gerar esses 3 bits
  - Entradas UALOp e campo funct
  - O código da instrução determina UALOp

Código de operação da instrução	UALOp	Operação da instrução	Campo de Função	Operação desejada da UAL	Entrada de controle da UAL
LW	00	load word	XXXXXX	soma	010
SW	00	store word	XXXXXX	soma	010
Branch equal	01	branch equal	XXXXXX	subtração	110
Tipo R	10	add	100000	soma	010
Tipo R	10	subtract	100010	subtração	110
Tipo R	10	AND	100100	and	000
Tipo R	10	OR	100101	or	001
Tipo R	10	set less than	101010	set less than	111

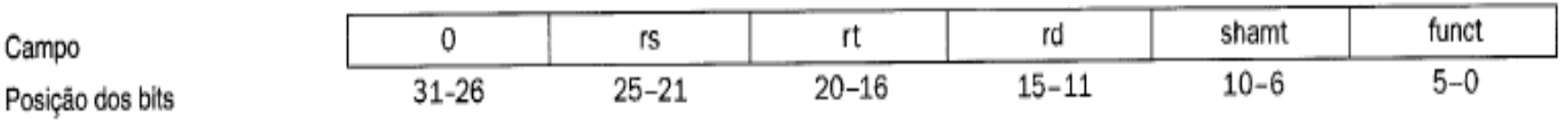
MIPS Uniciclo

- Mapeamento de UALOp e funct para controle da ULA
  - Campo de função só e usado quando UALOp = 10
  - Somente poucas combinações dos bits de funct são de interesse
    - Funct sempre inicia com 10, então não interessa
    - UALOp 11 não é usado, logo, X1 e 1X bastam

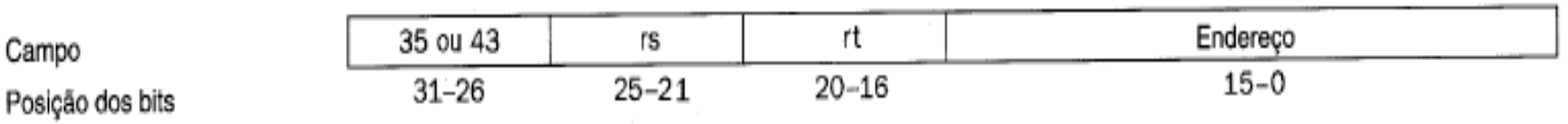
UALOp		Campo de função						Operação da UAL
UALOp1	UALOp2	F5	F4	F3	F2	F1	F0	
0	0	X	X	X	X	X	X	010
X	1	X	X	X	X	X	X	110
1	X	X	X	0	0	0	0	010
1	X	X	X	0	0	1	0	110
1	X	X	X	0	1	0	0	000
1	X	X	X	0	1	0	1	001
1	X	X	X	1	0	1	0	111

MIPS Uniciclo

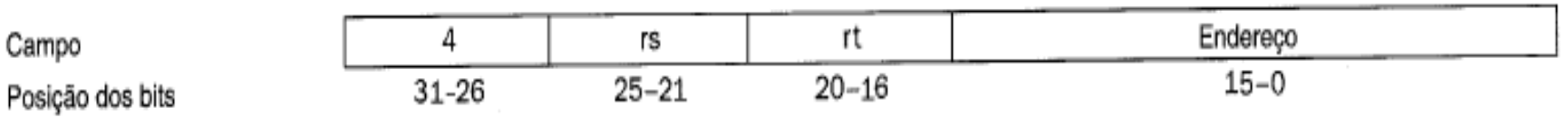
- Controle Principal
  - Extraídos dos campos da instrução
  - Revisão de formatos do MIPS:
    - Opcode nos bits 31-26
    - 2 Regs a serem lidos nas posições 25-21 e 20-16
    - Reg. Base p/ load e store sempre em 25-21
    - 16 bits de desloc. P/ branch, load e store em 15-0
    - Reg destino está de 15-11 se for instrução aritmética, senão, está em 20-16 no caso de load (uso de um MUX)



a. Instrução de tipo R



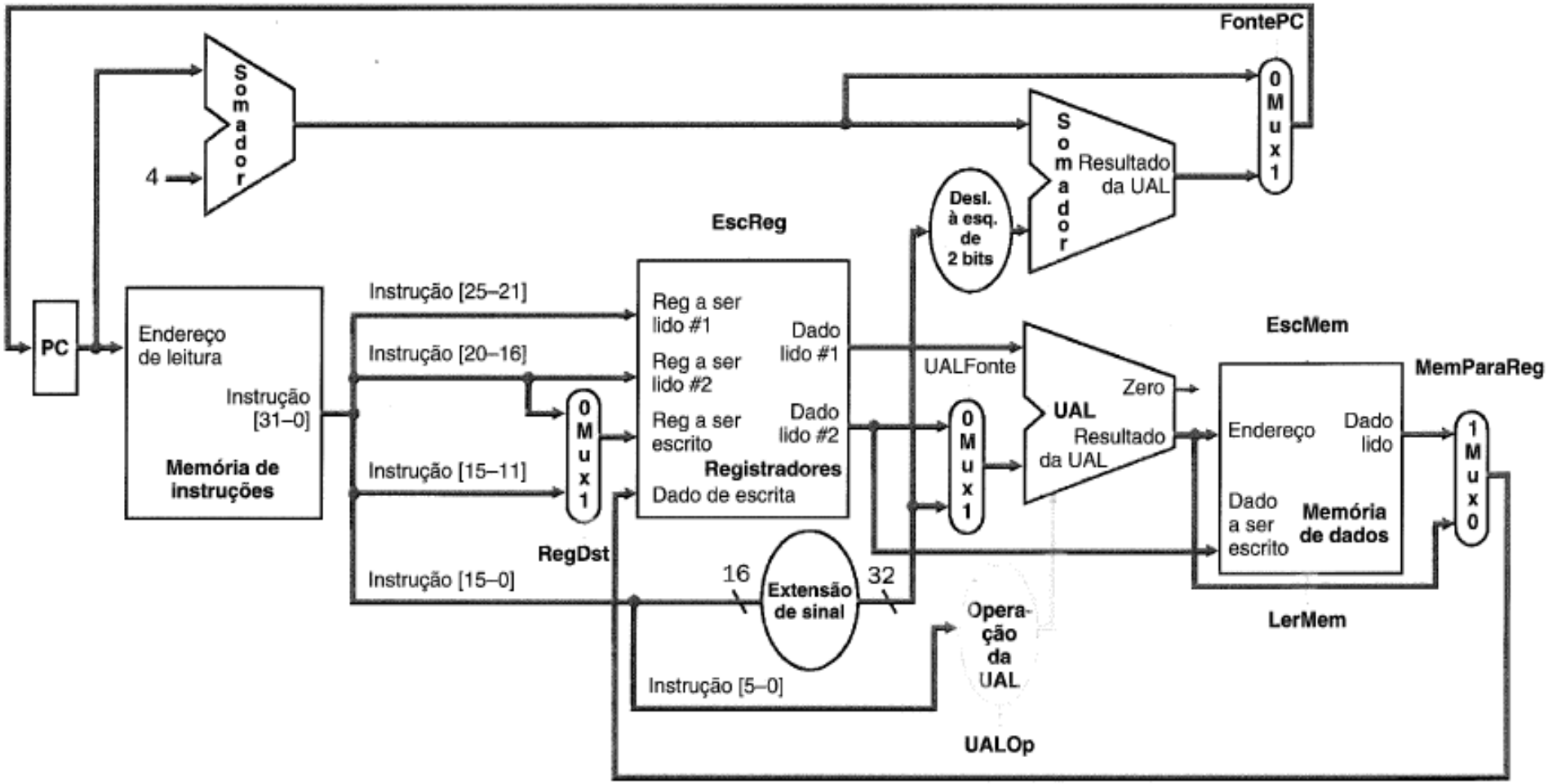
b. Instrução load word ou store word



c. Instrução de desvio condicional

MIPS Uniciclo

- Controle parcial



MIPS Uniciclo

- Outras linha de controle

Nome do sinal	Efeito quando ativo	Efeito quando inativo
RegDst	O número do registrador-destino onde será escrito o resultado da operação (Reg a ser Escrito) vem do campo rt (bits 20–16).	O número do registrador-destino onde será escrito o resultado da operação (Reg a ser Escrito) vem do campo rt (bits 15–11).
EscReg	Nenhum	O registrador na entrada Reg a ser Escrito é escrito com o valor presente na entrada Dado de Escrita.
UALFonte	O segundo operando da UAL vem do segundo registrador do banco de registradores (Dado lido #2).	O segundo operando da UAL é o resultado da extensão de sinal dos 16 bits menos significativos da instrução.
FontePC	O PC é substituído pelo valor presente na saída do somador que calcula PC + 4.	O PC é substituído pelo valor presente na saída do somador que calcula o endereço-alvo do desvio condicional.
LerMem	Nenhum	O conteúdo da memória designado pela entrada de endereço é colocado na saída Dado Lido.
EscMem	Nenhum	O conteúdo da memória designado pela entrada de endereço é substituído pelo valor presente na entrada Dado a ser Escrito.
MemParaReg	O valor na entrada do registrador de escrita vem da UAL.	O valor na entrada do registrador de escrita vem da memória de dados.

MIPS Uniciclo

- Sinais de Controle
  - P/ determiná-los, basta opcode da instrução, exceto para FontePC
  - FontePC é 1, se instrução de branch e saída Zero da ULA.

Instrução	RegDst	UALFonte	MemParaReg	EscReg	LerMem	EscMem	DvC	UALOp1	UALOp0
Formato R	1	0	0	1	0	0	0	1	0
lw	0	1	1	1	1	0	0	0	0
sw	X	1	X	0	0	1	0	0	0
beq	X	0	X	0	0	0	1	0	1

MIPS Uniciclo

- Sinais de controle
  - Opcodes:

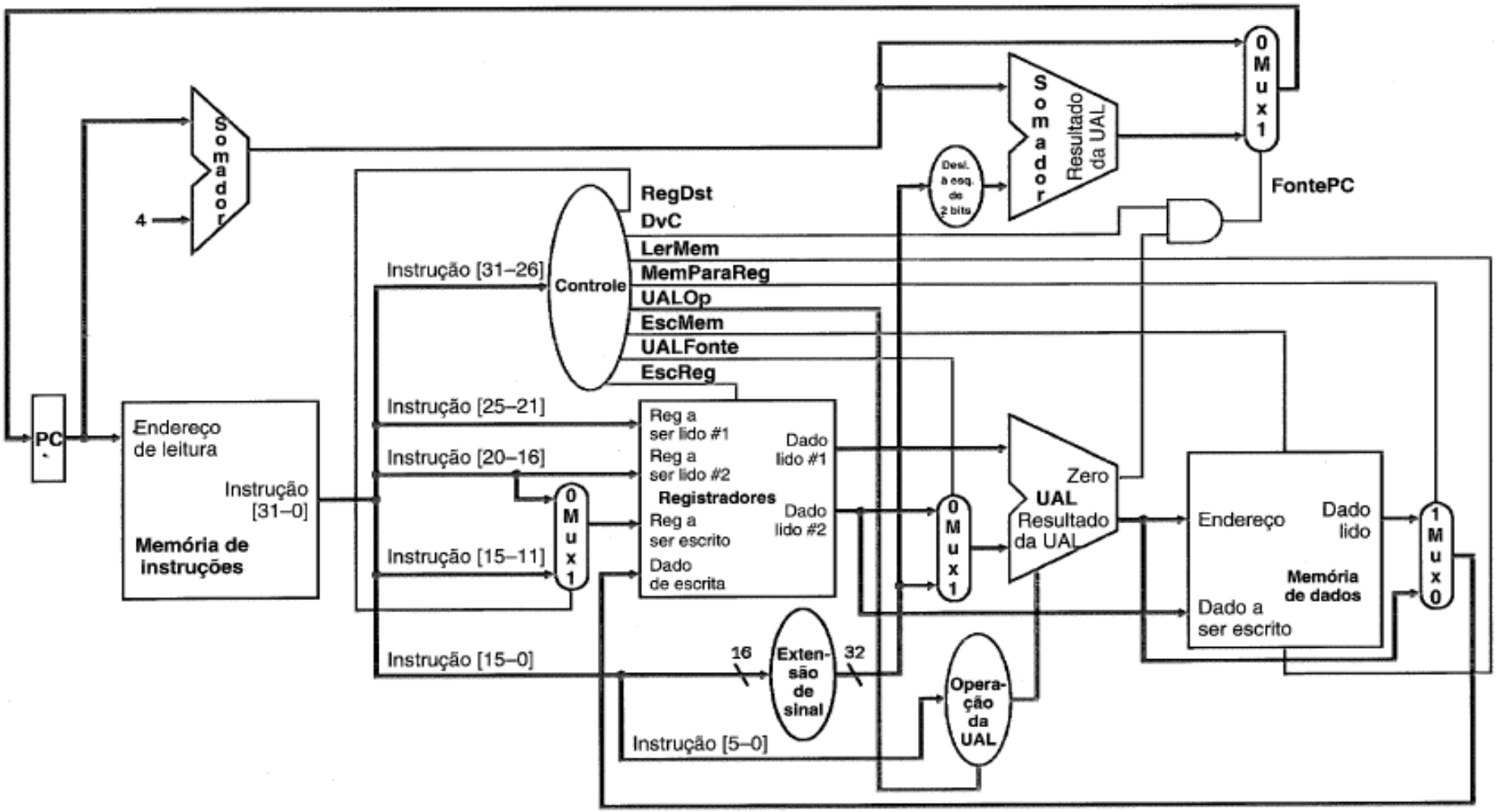
Nome	Opcode em decimal	Opcode em binário					
		Op5	Op4	Op3	Op2	Op1	Op0
Formato R	0_{dez}	0	0	0	0	0	0
lw	35_{dez}	1	0	0	0	1	1
sw	43_{dez}	1	0	1	0	1	1
beq	4_{dez}	0	0	0	1	0	0

  - Determinação dos sinais:

Entrada ou saída	Nome do sinal	Formato R	lw	sw	beq
Entradas	Op5	0	1	1	0
	Op4	0	0	0	0
	Op3	0	0	1	0
	Op2	0	0	0	1
	Op1	0	1	1	0
	Op0	0	1	1	0
Saídas	RegDst	1	0	X	X
	UALFonte	0	1	1	0
	MemParaReg	0	1	X	X
	EscReg	1	1	0	0
	LerMem	0	1	0	0
	EscMem	0	0	1	0
	DvC	0	0	0	1
	UALOp1	1	0	0	0
	UALOp0	0	0	0	1

MIPS Uniciclo

- Visão Geral



Execução de Instrução R

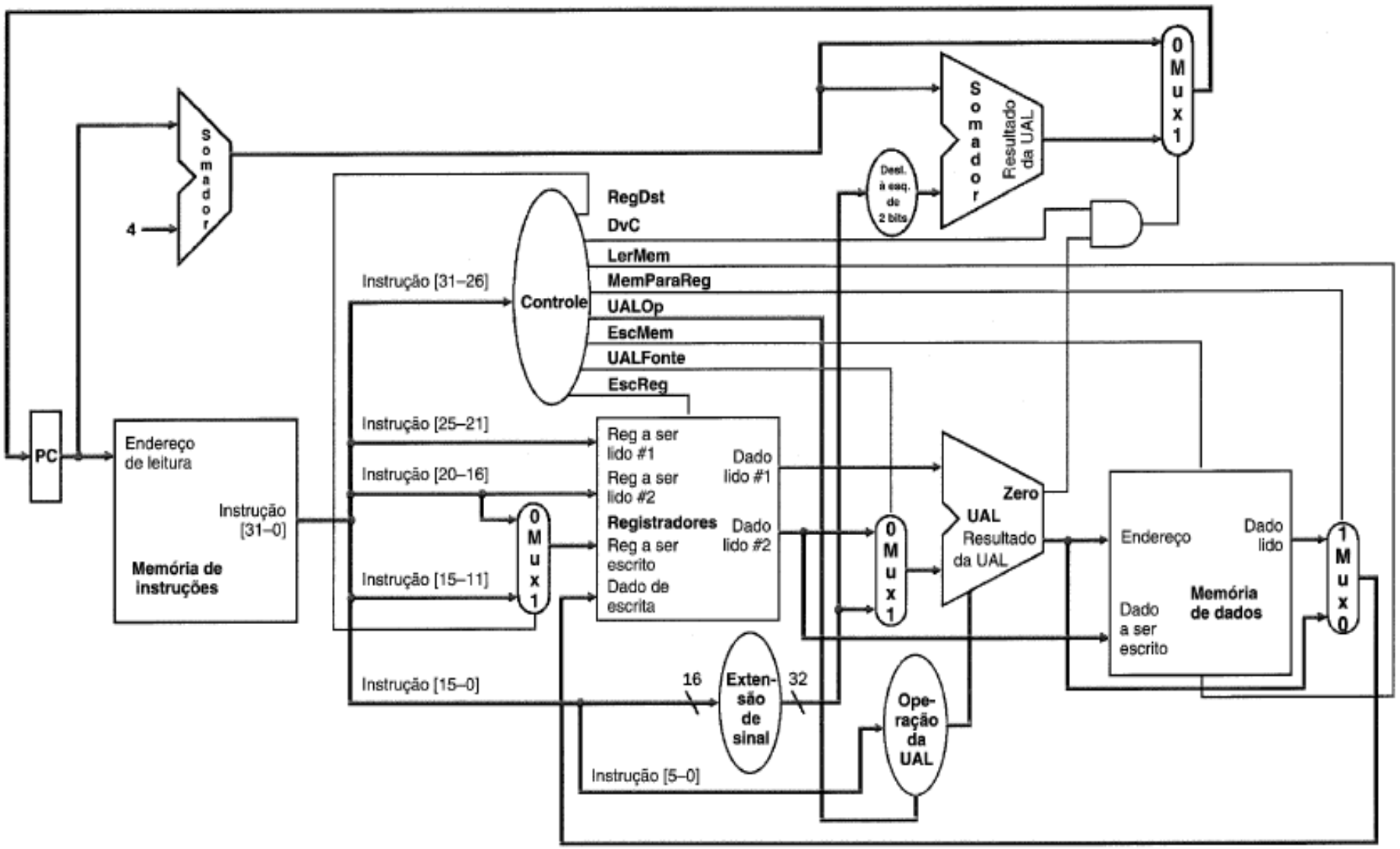


Figura 5.21 O primeiro passo de uma instrução de tipo R efetua a busca da instrução da memória de instruções e incrementa o PC. As partes ativas neste passo aparecem em destaque; as partes mais claras não estarão ativas neste passo, mas algumas delas poderão estar ativas ainda dentro deste ciclo.

Execução de Instrução R

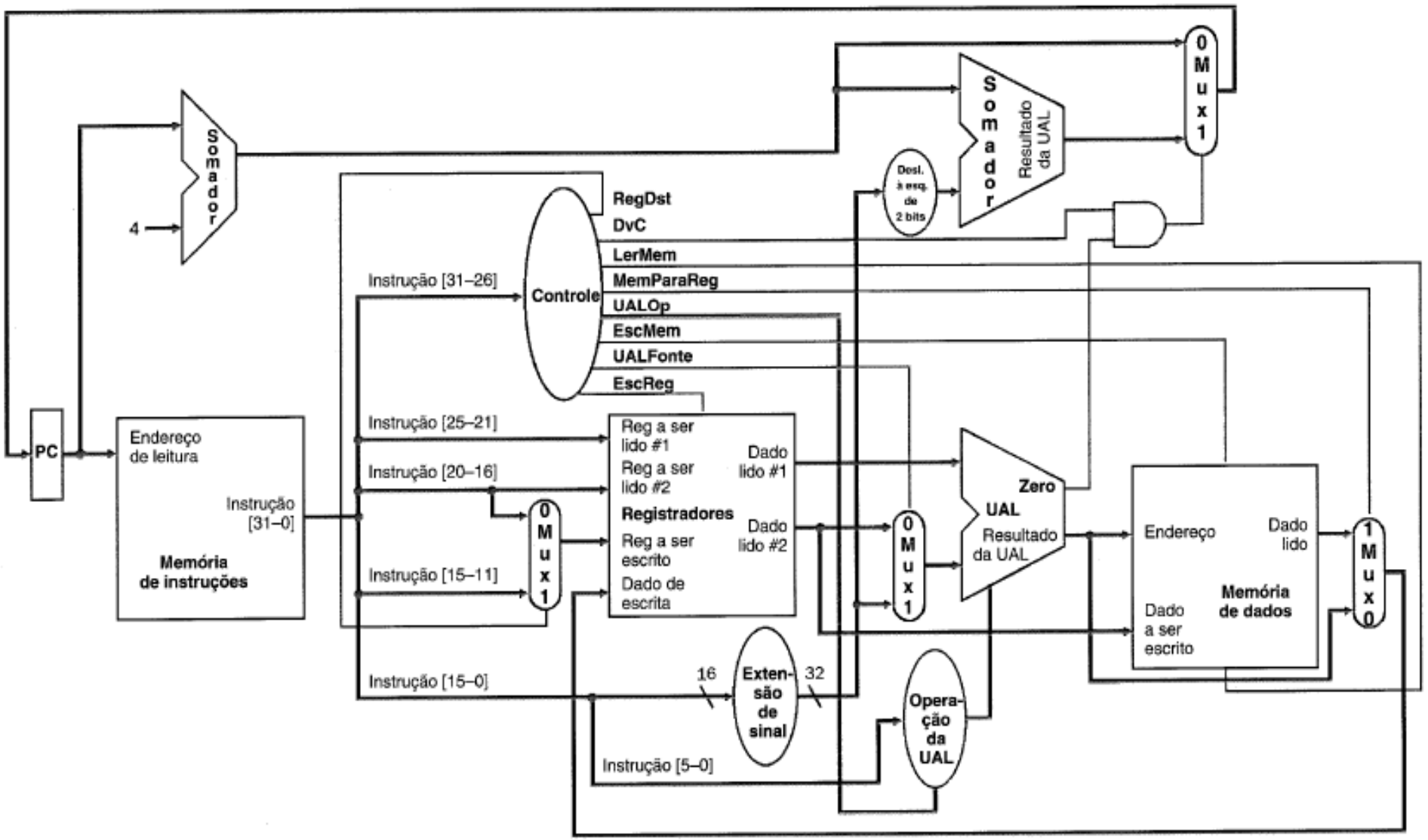


Figura 5.22 A segunda fase da execução de instruções de tipo R lê o conteúdo de registradores de duas fontes do banco de registradores. A unidade de controle principal também usa o campo do opcode para atribuir valores às linhas de controle. Essas unidades tornam-se ativas ao mesmo tempo que aquelas ativas durante a busca da instrução, mostrada na Figura 5.21.

Execução de Instrução R

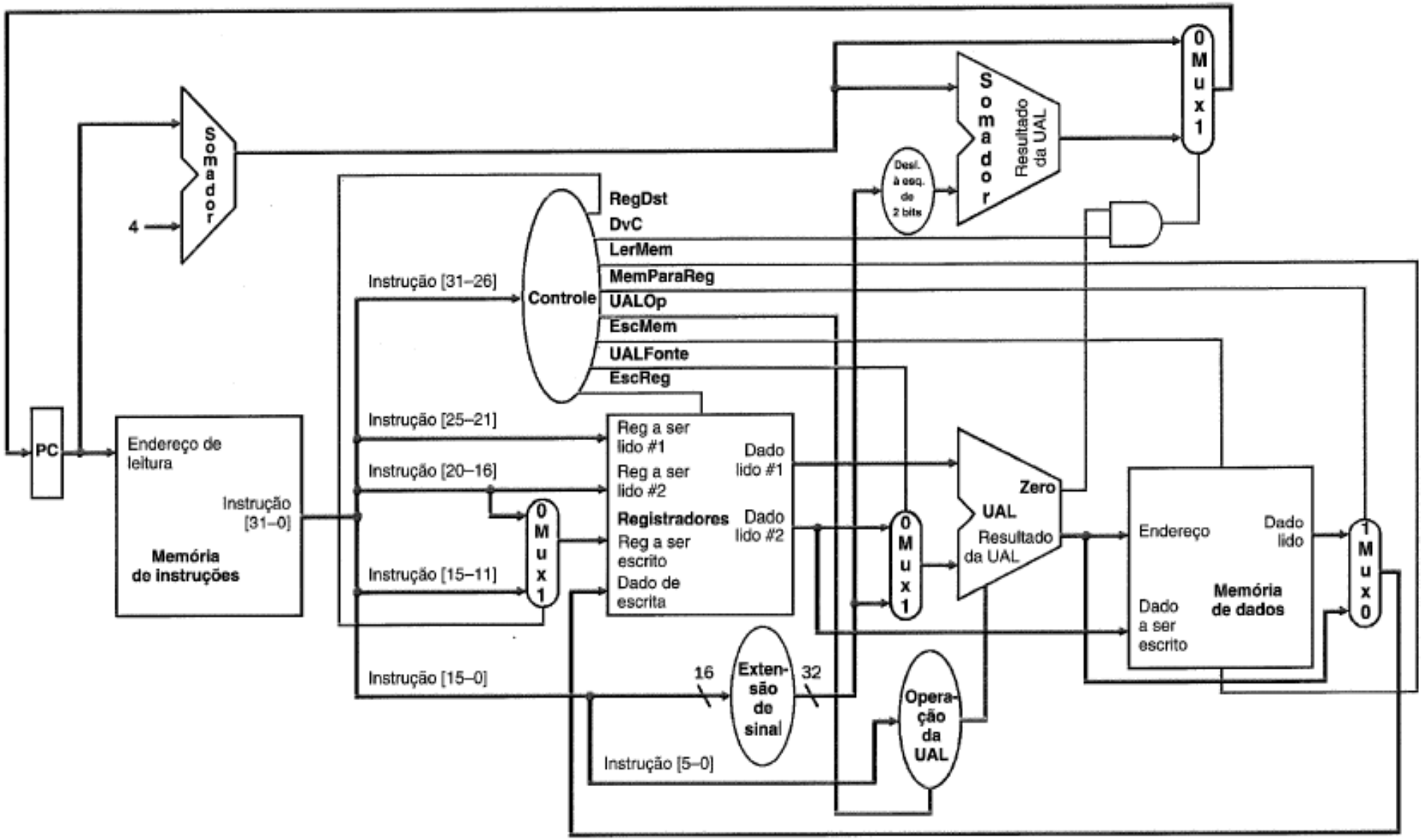


Figura 5.23 A terceira fase da execução de instruções de tipo R envolve a realização pela UAL da operação especificada, sobre o conteúdo dos registradores de dados. Os valores das linhas de controle são todos atribuídos, e o controle da UAL já havia sido calculado anteriormente. A UAL opera sobre dados.

Execução de Instrução R

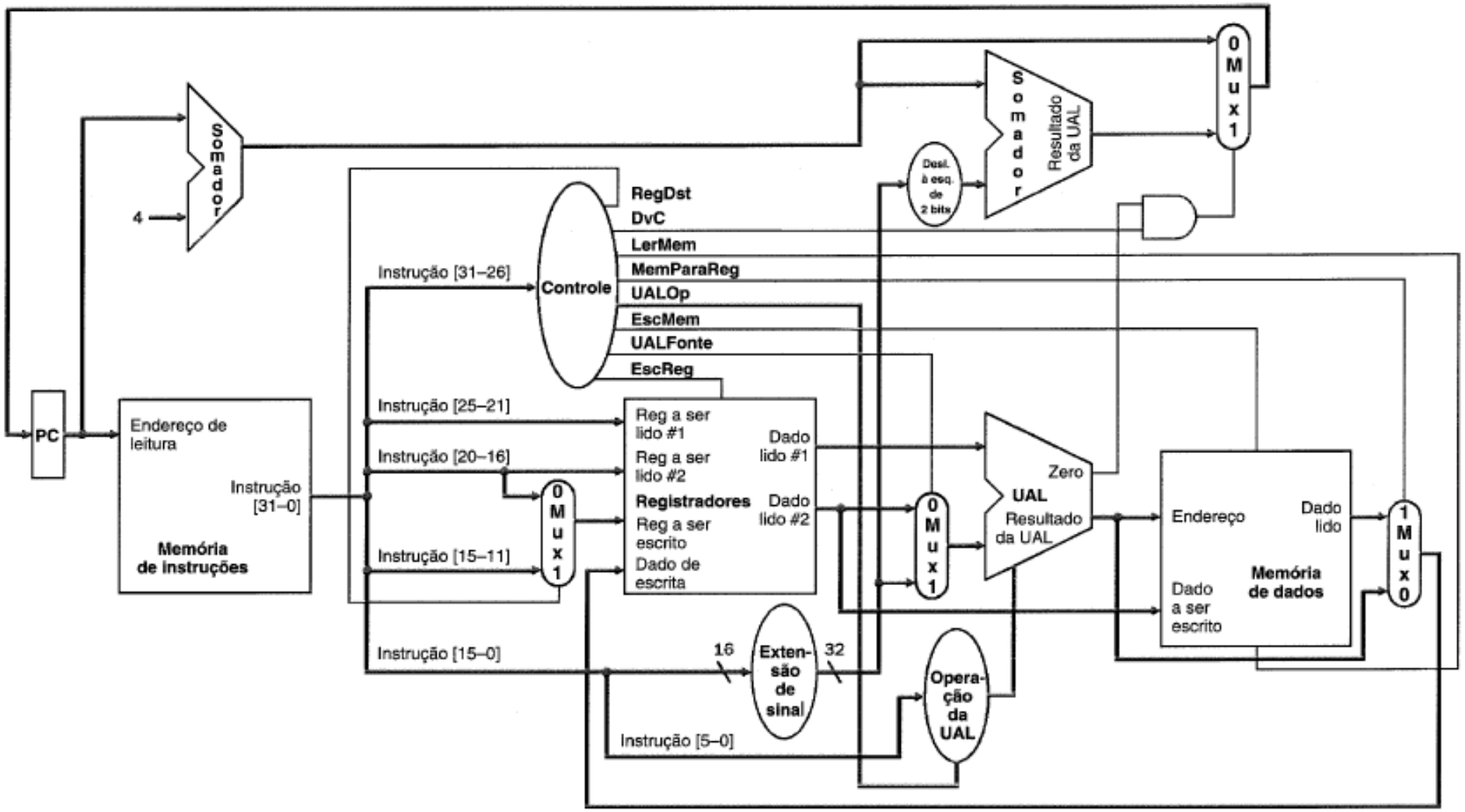


Figura 5.24 O passo final da execução de uma instrução do tipo R, a escrita do resultado, é realizado junto com a ativação das unidades mostradas nos últimos três passos descritos na Figura 5.23. O PC também é atualizado no final desta fase. Considerando que o caminho de dados é combinacional, este passo mostra todas as unidades e linhas de controle ativas, quando elas podem ser consideradas estáveis. Observe que, mesmo se a instrução for daquelas que usam o mesmo registrador tanto para leitura quanto para escrita (como, por exemplo, `add R1,R1,R1`), ela ainda assim funciona corretamente: o valor lido dos registradores é o valor de `R1`, que foi escrito no final de algum dos ciclos de clock anteriores, enquanto o valor escrito nos registradores pela instrução só é efetivamente escrito na transição do clock realizada no final do ciclo de clock atual.

Execução de Instrução Load

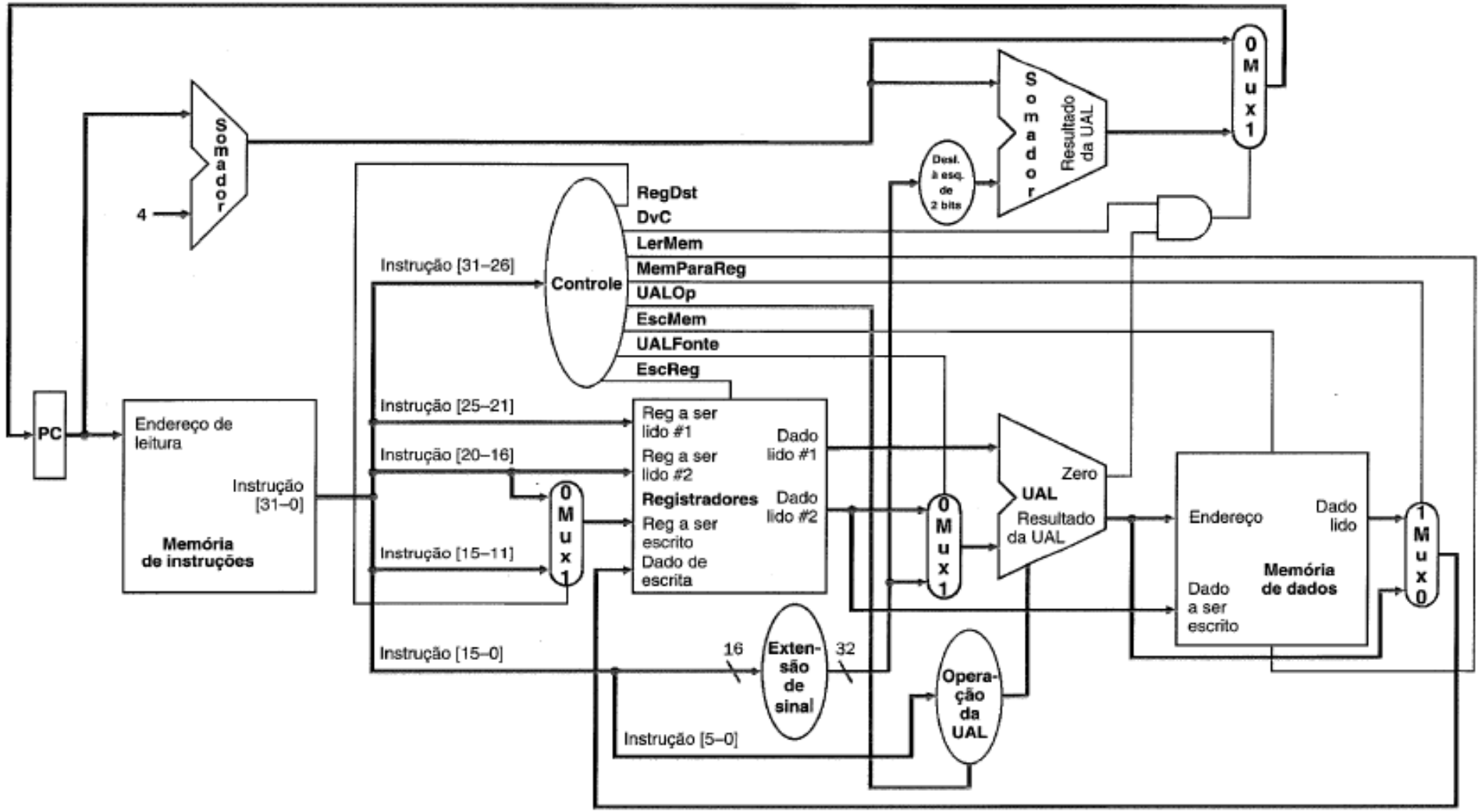


Figura 5.25 Operação de uma instrução de load word no caminho de dados definido, mostrando o esquema de controle utilizado. Uma instrução de store operaria neste caminho de dados de maneira muito semelhante à de load. A principal diferença seria no sinal de controle da memória que deveria indicar escrita no caso de uma instrução de store, em vez de leitura, como no caso da instrução de load word. Além disso, o valor do segundo registrador lido seria usado para armazenar o dado, e a operação de escrita em um registrador do dado da memória não ocorreria.

Execução de Instrução Branch-eq

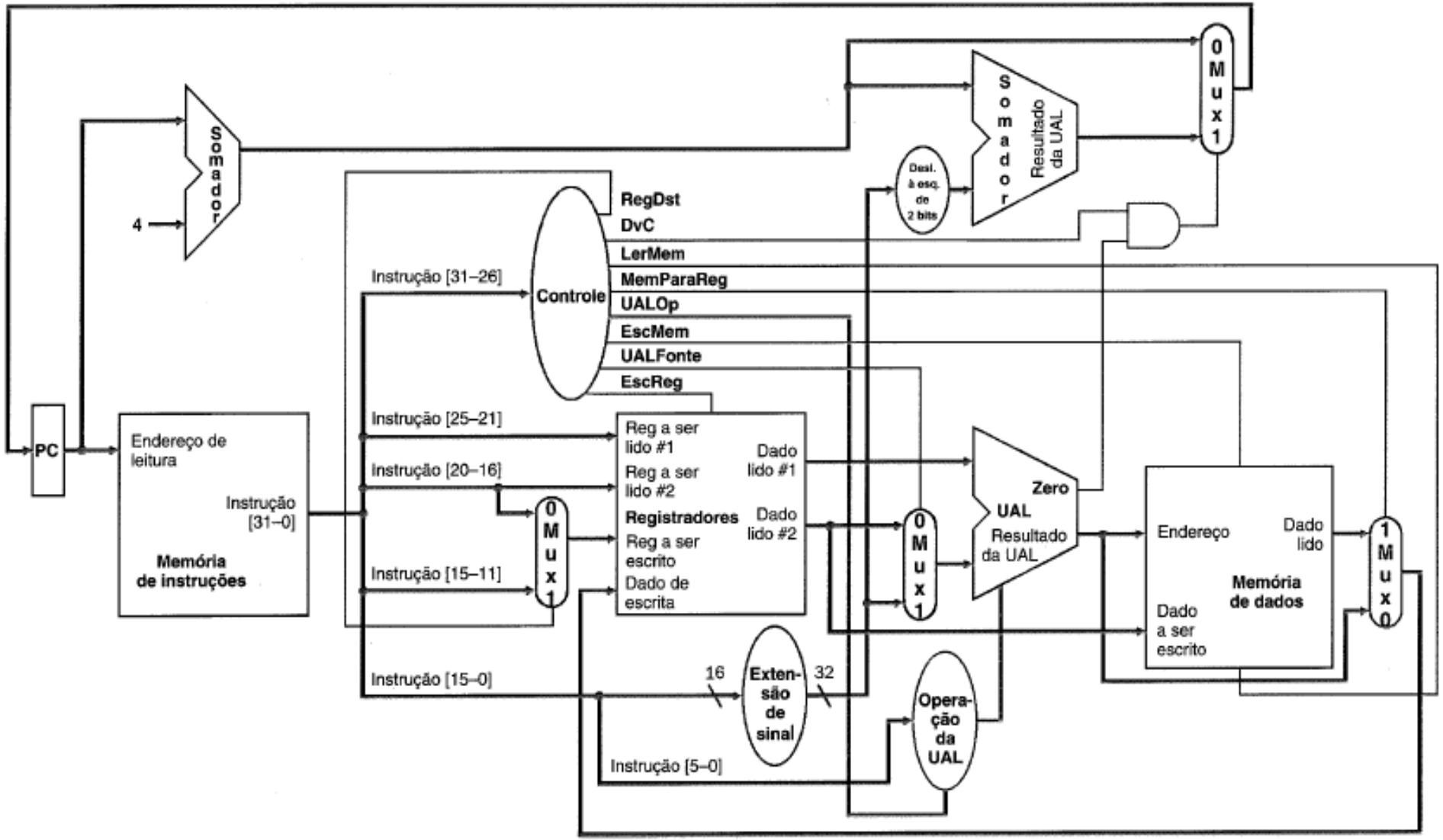


Figura 5.26 Caminho de dados em operação para a execução de uma instrução branch equal. Após usar o banco de registradores e a UAL para realizar a comparação, a saída Zero é usada para selecionar o valor a ser armazenado no PC, entre os dois valores candidatos.

MIPS Uniciclo

- Problemas
  - Todas as instruções deveriam funcionar em um único ciclo de clock (CPI = 1!)
  - O clock deverá ser ajustado para a instrução mais demorada
    - Ex. Load usa cinco unidades funcionais em série
    - Outras instruções poderiam ser executadas em ciclo menor
  - Ex. Se o tempo das unidade funcionais forem:
    - Memória: 2 ns
    - ULA: 2 ns
    - Banco de registradores: 1ns
    - Mux, controle e outros: Desprezíveis
    - Qual o clock para cada instrução?

MIPS Multiciclo

- Objetivo: Quebrar a execução de instruções em passos

- Cada passo é executado em um ciclo de clock

- Favorece o reuso de hardware
  - Ex. Mem de instruções e dados, ULA e somadores
  - Necessita mais MUXs

- Usa mais registradores para armazenar valores intermediários (não visíveis ao programador)

- Instruções diferentes poderão usar quantidades diferentes de ciclos de clock

MIPS Multiciclo

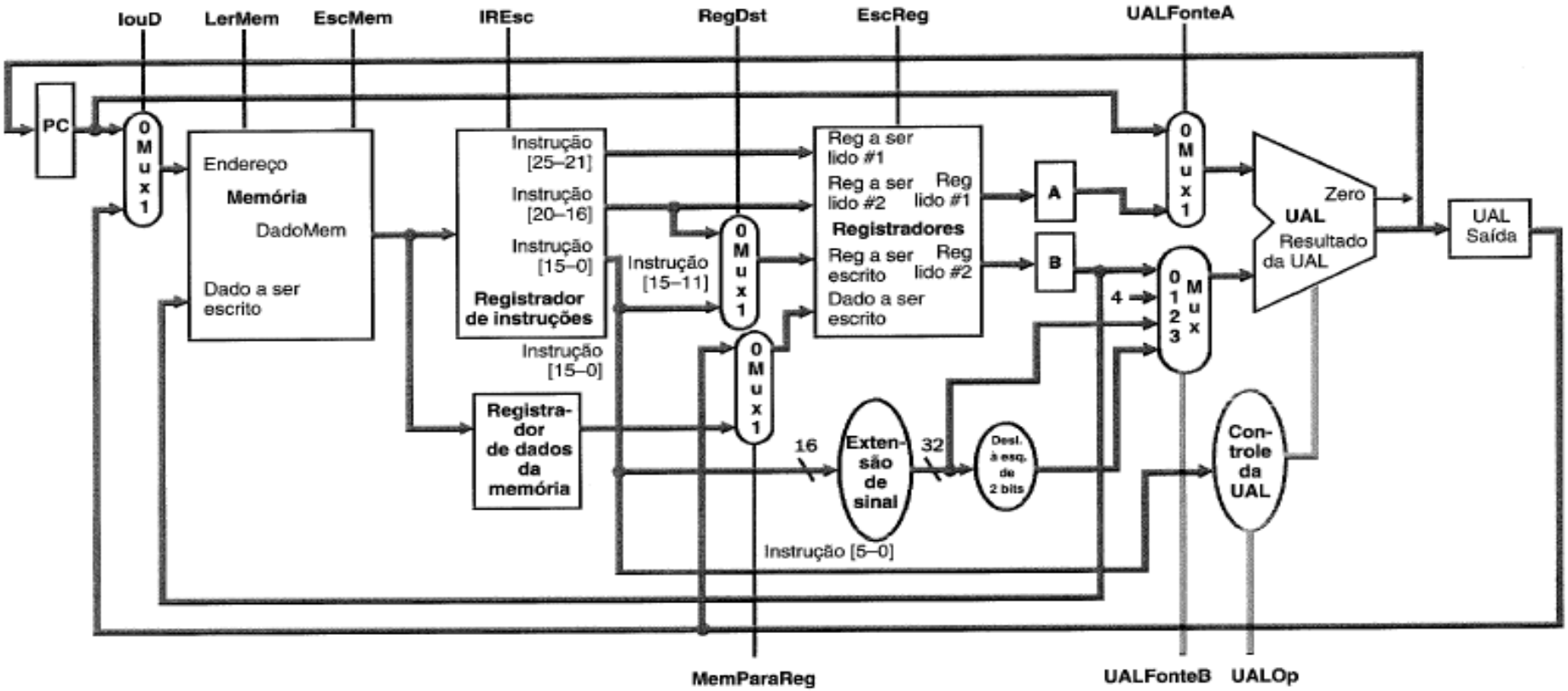


Figura 5.31 Caminho de dados multiciclo para execução das instruções básicas do MIPS. Apesar de este caminho de dados suportar o incremento normal do PC, serão necessárias algumas novas conexões e um novo multiplexador para suportar desvios condicionais e incondicionais. Em comparação com o caminho de dados para implementação monociclo, a implementação multiciclo inclui diversos registradores novos (IR, MDR, A, B, UALSaída), um multiplexador para o endereço de memória, um multiplexador para a entrada superior da UAL, e a expansão do multiplexador da entrada inferior da UAL de duas para quatro entradas. Essas pequenas modificações permitiram-nos eliminar dois somadores e uma memória.

Figura 5.32 Caminho de dados multiciclo da Figura 5.31 com as linhas de controle. Os sinais UALOp e UALSelB são sinais de controle de 2 bits, enquanto todos os demais são de 1 bit. Os registradores A e B não precisam de um sinal de escrita, uma vez que seus conteúdos são lidos somente em um ciclo imediatamente após eles terem sido escritos. O registrador de dados de memória foi adicionado para guardar o dado de um load, quando ele chega da memória. Os dados que chegam da memória não podem ser escritos diretamente no banco de registradores, pois a duração do ciclo de clock não é suficiente para acomodar um acesso à memória e outro ao banco de registradores. O sinal LerMem foi deslocado para o topo da unidade de memória para simplificar as figuras. O conjunto completo de unidades e linhas de controle para um caminho de dados que suporte desvios condicionais será mostrado em breve.

MIPS Multiciclo

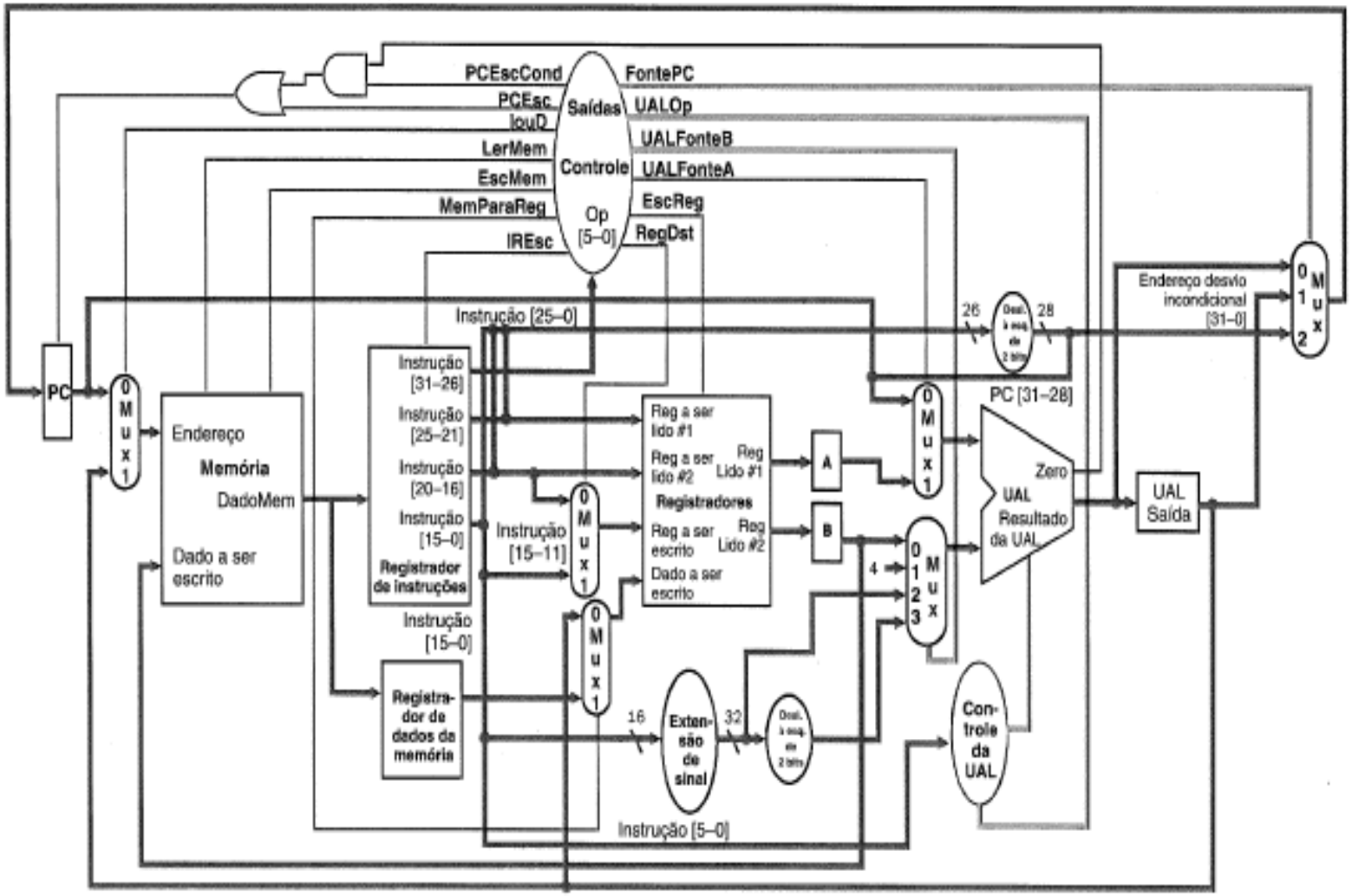


Figura 5.33 Caminho de dados completo para a implementação multiciclo junto com as linhas de controle necessárias ao seu funcionamento. As linhas de controle da Figura 5.32 foram incorporadas à unidade de controle, além de serem incluídos todos os elementos necessários ao suporte das mudanças no PC. Os principais acréscimos, comparando com o caminho de dados da Figura 5.32, foram o multiplexador utilizado para selecionar a fonte do novo valor do PC (em cima, à direita), duas portas usadas para relacionar os sinais de escrita do PC (em cima, à esquerda) e os sinais de controle FontePC, PCEsc e PCEscCond. Este último sinal passa por uma porta AND com a saída Zero da UAL, para decidir se deve ou não ser realizado um desvio condicional. O sinal resultante passa por um OR junto com o sinal de controle PCEsc para gerar o sinal de controle de escrita do PC. Além disso, a saída do IR deve ser rearrumada de maneira a mandar seus 26 bits menos significativos (o endereço do desvio incondicional) para a lógica usada para selecionar o próximo valor do PC. Estes 26 bits são deslocados 2 bits à esquerda, recebendo dois zeros em seus bits menos significativos. Estes 28 bits são então concatenados com os 4 bits de mais alta ordem do PC incrementado.

MIPS Multiciclo

- Controle
  - Sinais de controle para cada passo da execução, em vários ciclos.
  - Técnicas:
    - Máquina estados finitos: Estados e transições
    - Microprogramação: Representação por um programa
  - Cada estado gasta 1 ciclo de clock

MIPS Multiciclo

Estágios:

- Busca de instrução:
  - Busca instrução na MEM e incrementa PC
- Inst Dec & Reg.
  - Leituras dos regs A e B
  - Calculo do Destino do Branch (ALUout)
- Execution, Mem Address ou Branch Compl.
  - ALUout = A+end_mem, ou
  - ALUout = A op B, ou
  - Se (A=B) PC = ALUout, ou
  - PC = PC [31-28] || IR[0-25]<<2
- Mem access ou R-type compl.
  - MDR = Mem[ALUout],
  - Mem[ALUout] = B,
  - Reg[IR[15-11]] = ALUout
- Mem read compl.
  - Reg[IR[20-16]]=MDR

MIPS Multiciclo

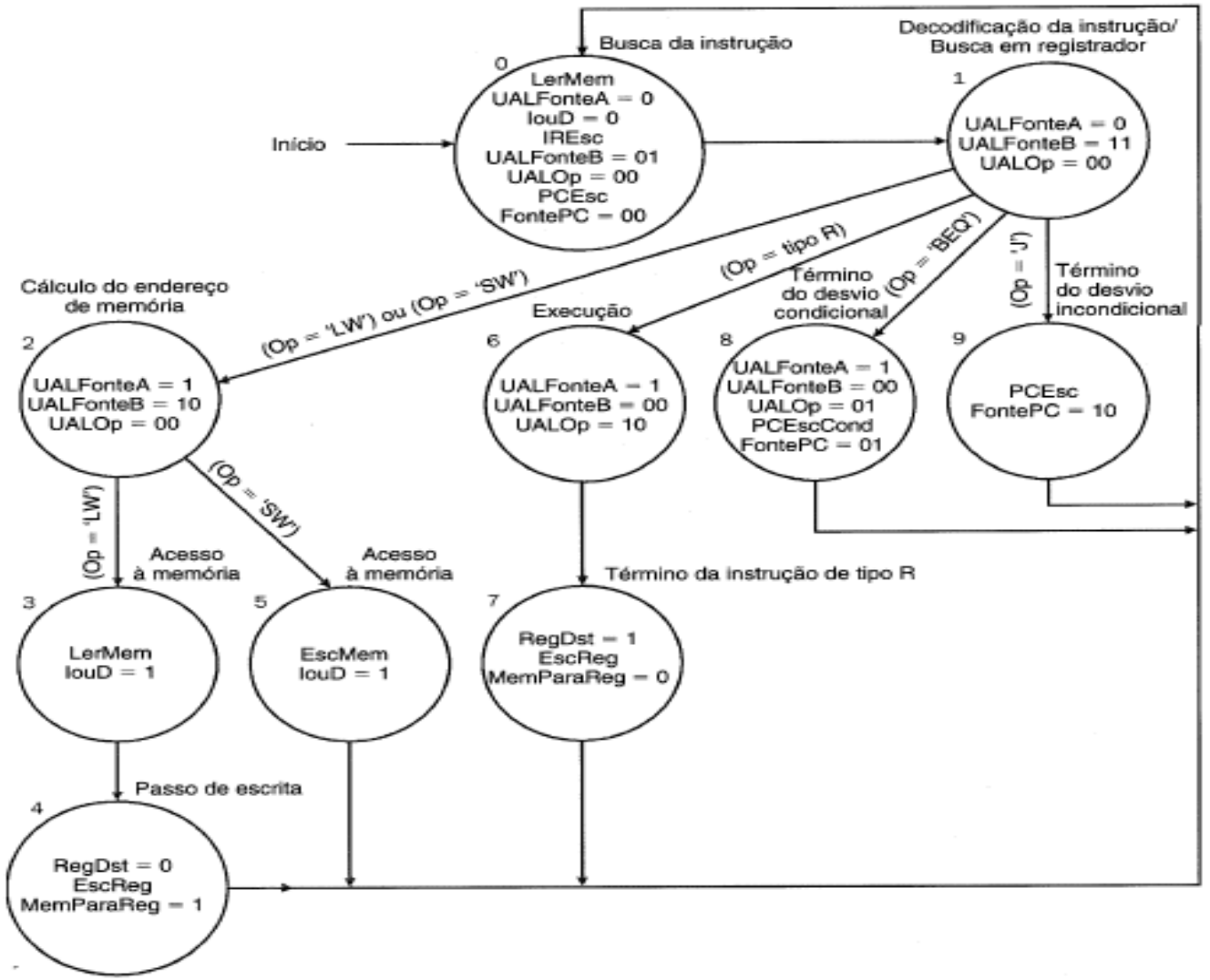


Figura 5.42 O controle completo, baseado em uma máquina de estados finitos para o caminho de dados mostrado na Figura 5.33. As identificações constantes dos arcos são condições que devem ser testadas para determinar qual dos estados é o próximo; quando a determinação do próximo estado for incondicional, não é colocada qualquer identificação. Os nomes dentro dos nós indicam os sinais de saída, ativos durante esse estado. Vamos sempre especificar os valores dos controles dos multiplexadores se a correta operação da estrutura assim determinar. Portanto, em alguns estados um determinado controle de multiplexador terá o valor 0 atribuído. No Apêndice C vamos examinar como implementar esta máquina de estados finitos a partir de equações lógicas, além de como implementar essas equações lógicas usando circuitos.