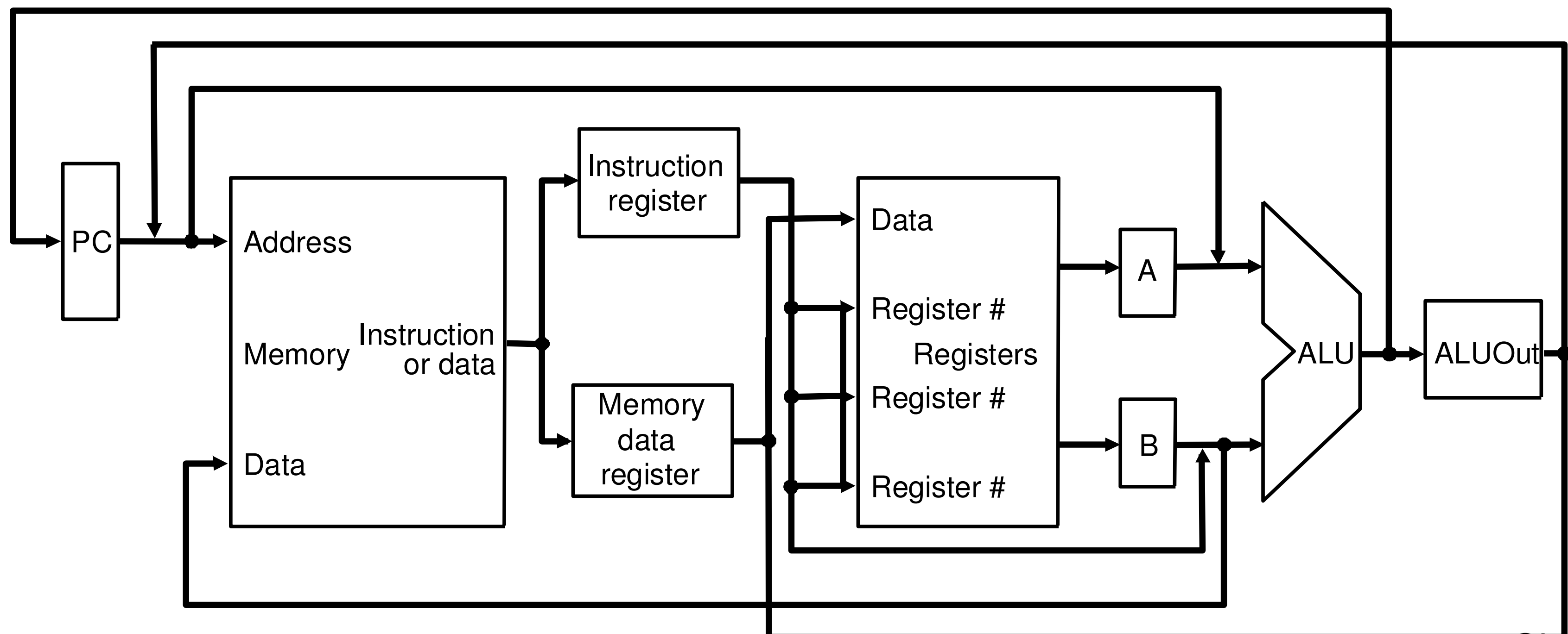
Chapter Five

The Processor: Datapath and Control (Parte B: multiciclo)

©1998 Morgan Kaufmann Publishers

Multicycle Approach

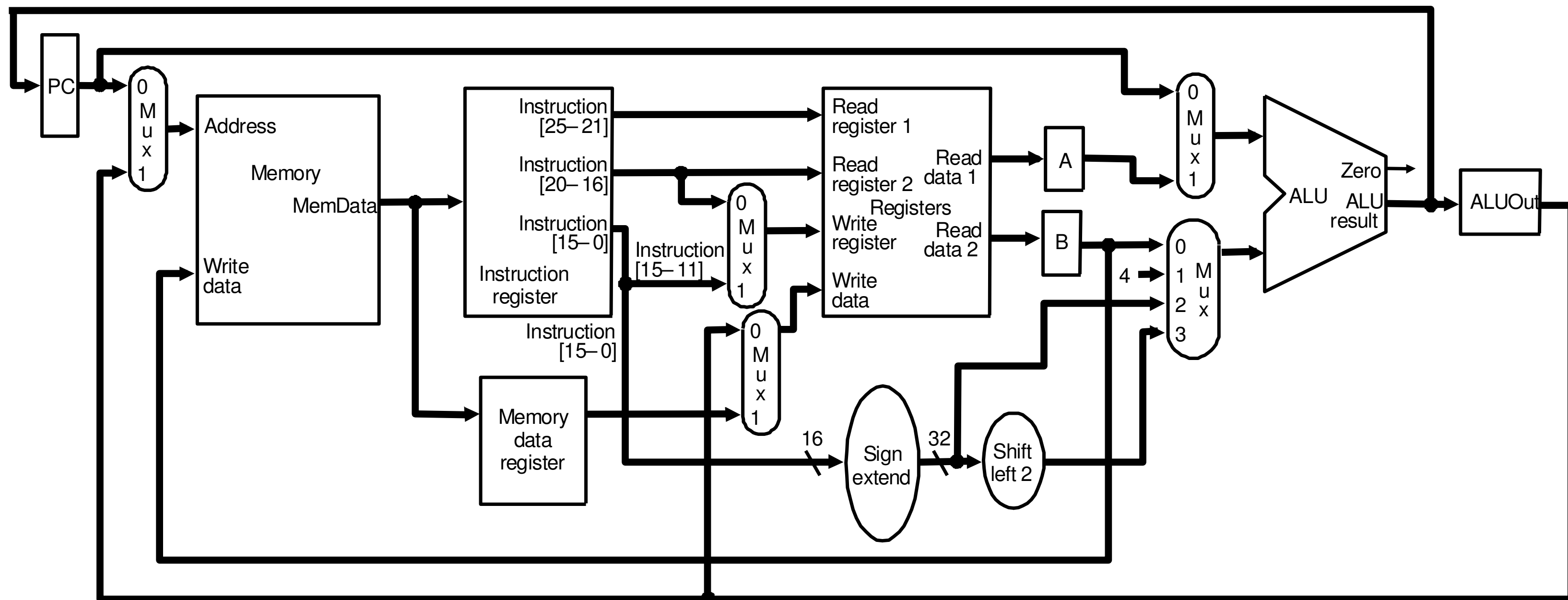
- **Break up the instructions into steps, each step takes a cycle**
  - **balance the amount of work to be done**
  - **restrict each cycle to use only one major functional unit**
    - 1 ALU, 1 Memória, 1 Banco de Registradores
- **At the end of a cycle**
  - **store values for use in later cycles (easiest thing to do)**
  - **introduce additional “internal” registers**



©1998 Morgan Kaufmann Publishers

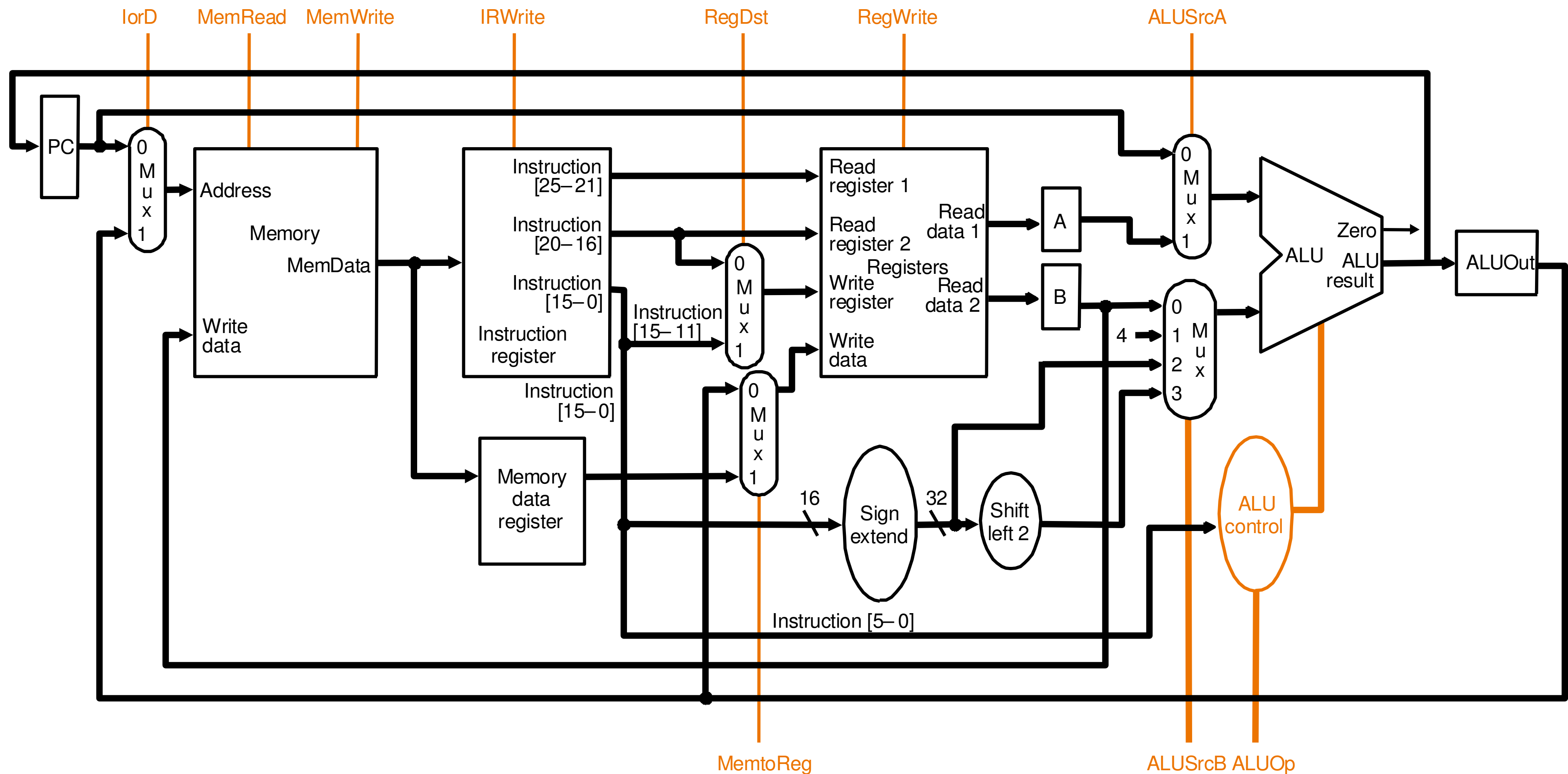
Multicycle Approach

- **IR (Instruction Register) e MDR (Memory Data Register) salvam saída da memória**
- **Registradores A e B salvam saída do banco de registradores**
- **ALUout salva saída da ALU**
- **Todos, exceto IR, guardam dados por um clock ⇒ controle de escrita é desnecessário**
- **novos MUXes: endereço da memória, operandos da ALU**



©1998 Morgan Kaufmann Publishers

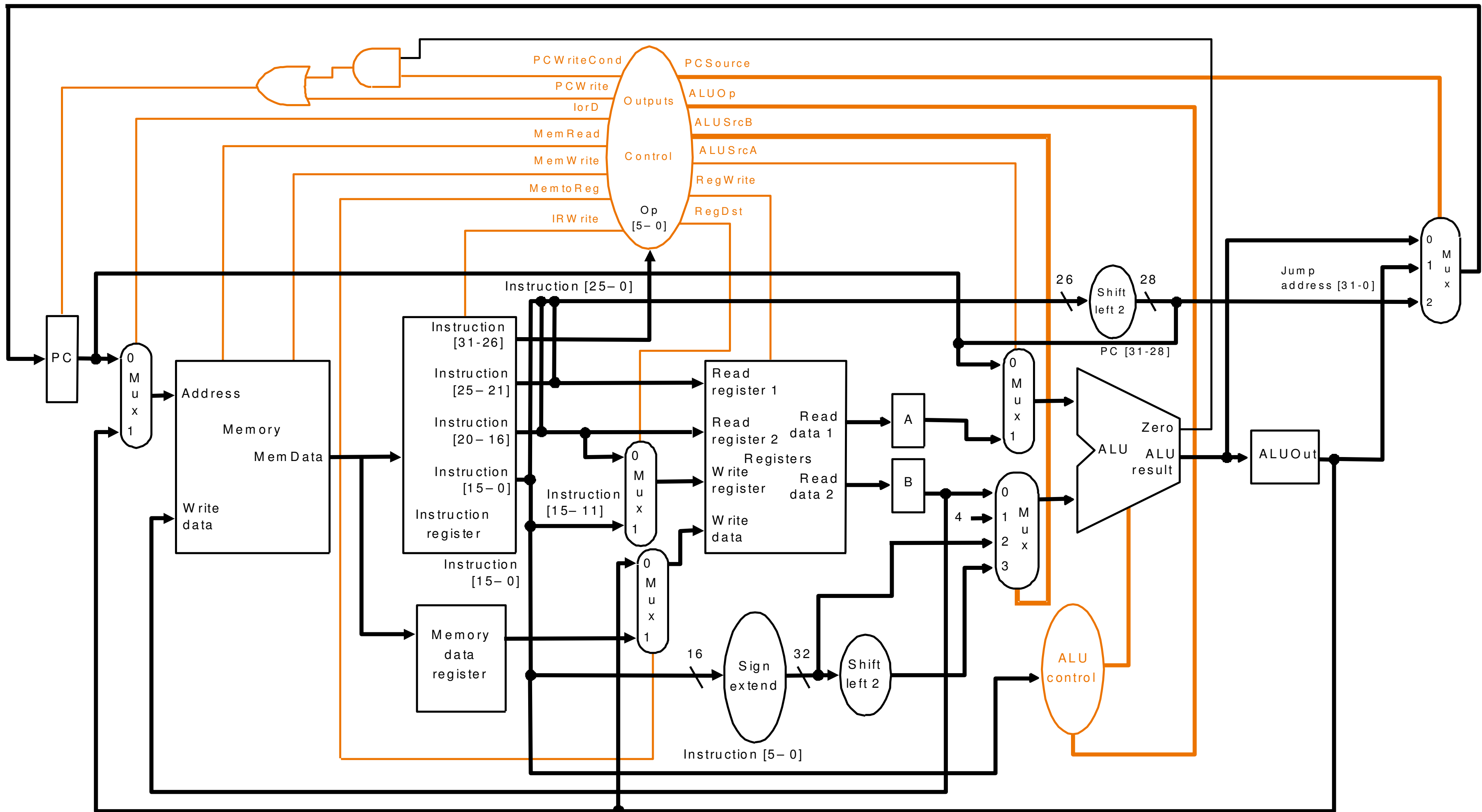
Via de dados multiciclo e sinais de controle



Falta implementar 3 possíveis fontes de carga para PC: PC+4, beq e j

©1998 Morgan Kaufmann Publishers

Via de dados completa



©1998 Morgan Kaufmann Publishers

Sinais de controle (1 bit)

	Desligado	Ligado
RegDst	Reg de escrita rt	Reg de escrita rd
RegWR		Escreve no banco
ALUSrcA	operando é o PC	operando é o reg A
MemRD		Lê a memória
MemWR		Escreve na memória
Memto Reg	Reg WR data ALUout	Reg WR data MDR
IorD	Endereço do PC	Endereço de ALUout
IRWrite		Escreve no IR
PCWrite		Escreve no PC
PCWriteCond		Escr PC se ALU=0

©1998 Morgan Kaufmann Publishers

Sinais de controle (2 bits)

ALUop	00	Soma
	01	Subtração
	10	Function
ALUSrcB	00	Segundo operando é o registrador B
	01	“ constante 4
	10	“ sign-extend, 16 bits do IR
	11	“ idem acima, deslocado 2 bits à esquerda
PCSrc	00	PC + 4
	01	ALUout (target address)
	10	Jump (4 bits PC & 26 bits ender & 00)

©1998 Morgan Kaufmann Publishers

Five Execution Steps

- **Instruction Fetch**
- **Instruction Decode and Register Fetch**
- **Execution, Memory Address Computation, or Branch Completion**
- **Memory Access or R-type instruction completion**
- **Write-back step**

INSTRUCTIONS TAKE FROM 3 - 5 CYCLES!

©1998 Morgan Kaufmann Publishers

Step 1: Instruction Fetch

- **Use PC to get instruction and put it in the Instruction Register.**
- **Increment the PC by 4 and put the result back in the PC.**
- **Can be described succinctly using RTL "Register-Transfer Language"**

```
IR = Memory[PC];
PC = PC + 4;
```

Can we figure out the values of the control signals?

What is the advantage of updating the PC now?

©1998 Morgan Kaufmann Publishers

Step 2: Instruction Decode and Register Fetch

- **Read registers rs and rt in case we need them**
- **Compute the branch address in case the instruction is a branch**
- **RTL:**

```
A = Reg[IR[25-21]];
B = Reg[IR[20-16]];
ALUOut = PC + (sign-extend(IR[15-0]) << 2);
```

- **We aren't setting any control lines based on the instruction type (we are busy "decoding" it in our control logic)**

©1998 Morgan Kaufmann Publishers

Step 3 (instruction dependent)

- ALU is performing one of three functions, based on instruction type
- Memory Reference:

```
ALUOut = A + sign-extend(IR[15-0]);
```

- R-type:

```
ALUOut = A op B;
```

- Branch:

```
if (A==B) PC = ALUOut;
```


©1998 Morgan Kaufmann Publishers

Step 4 (R-type or memory-access)

- **Loads and stores access memory**

```
MDR = Memory[ALUOut];
       or
Memory[ALUOut] = B;
```

- **R-type instructions finish**

```
Reg[IR[15-11]] = ALUOut;
```

The write actually takes place at the end of the cycle on the edge

©1998 Morgan Kaufmann Publishers

Write-back step

- `Reg[IR[20-16]]= MDR;`

What about all the other instructions?

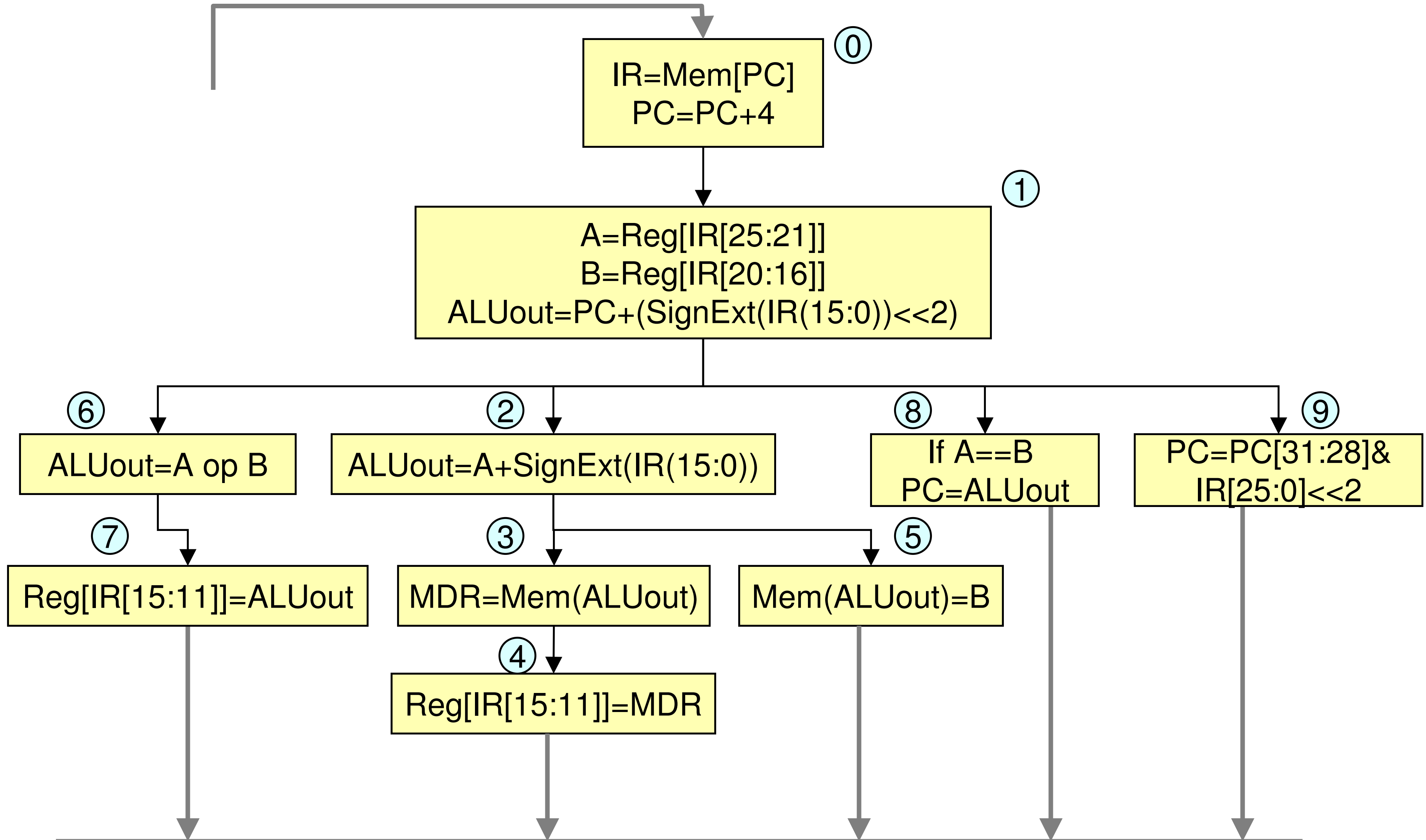
©1998 Morgan Kaufmann Publishers

Summary:

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) then PC = ALUOut	PC = PC [31-28] || (IR[25-0]<<2)
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		

©1998 Morgan Kaufmann Publishers

Outra visão, fluxograma



©1998 Morgan Kaufmann Publishers

Simple Questions

- **How many cycles will it take to execute this code?**

```
             lw $t2, 0($t3)
             lw $t3, 4($t3)
             beq $t2, $t3, Label        #assume not
             add $t5, $t2, $t3
             sw $t5, 8($t3)
Label:       ...
```

- **What is going on during the 8th cycle of execution?**
- **In what cycle does the actual addition of `$t2` and `$t3` takes place?**

©1998 Morgan Kaufmann Publishers

Implementing the Control

- **Value of control signals is dependent upon:**
  - **what instruction is being executed**
  - **which step is being performed**

- **Use the information we've acculumated to specify a finite state machine**
  - **specify the finite state machine graphically, or**
  - **use microprogramming**

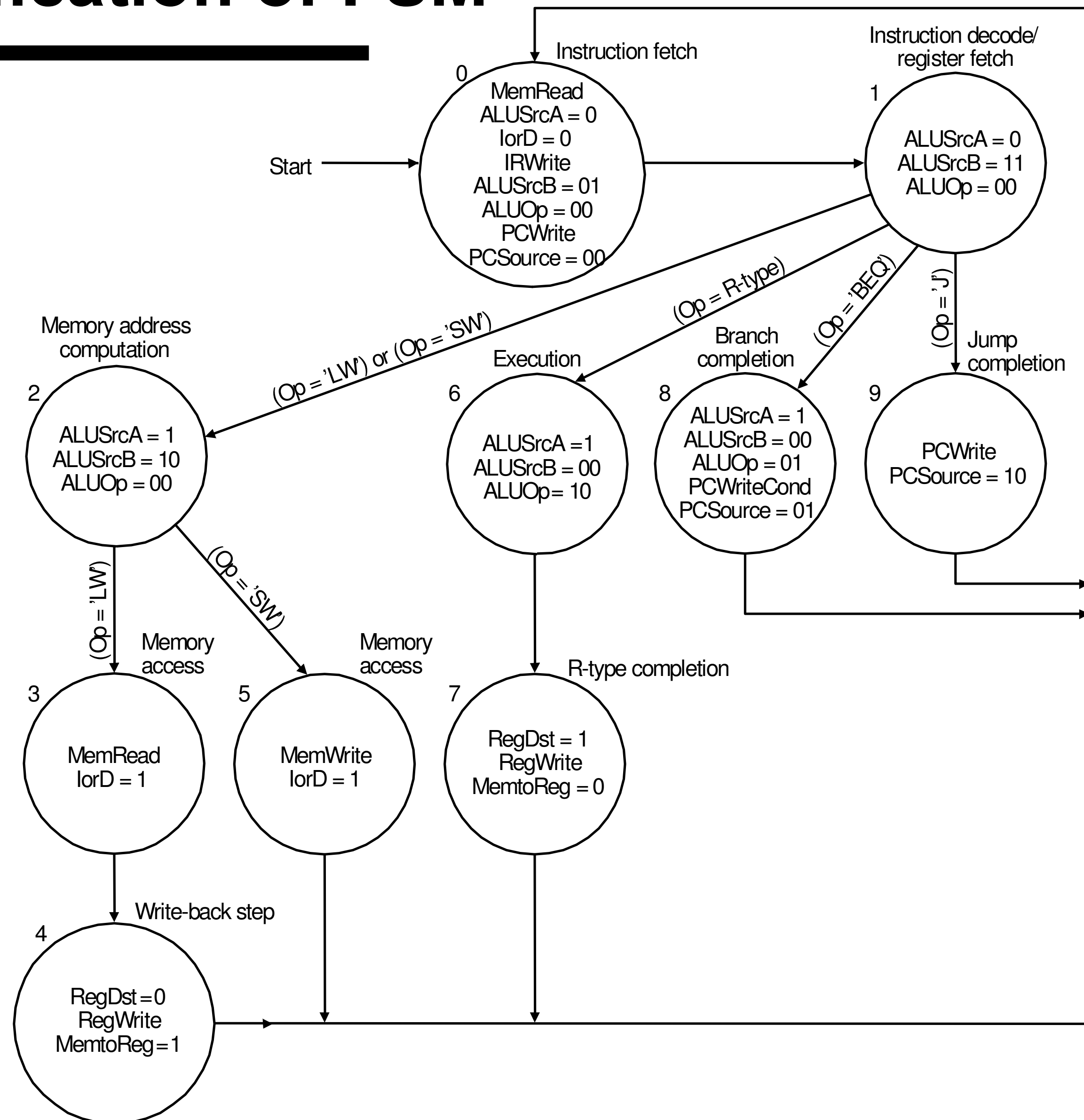
- **Implementation can be derived from specification**

©1998 Morgan Kaufmann Publishers

Tabela dos sinais de controle

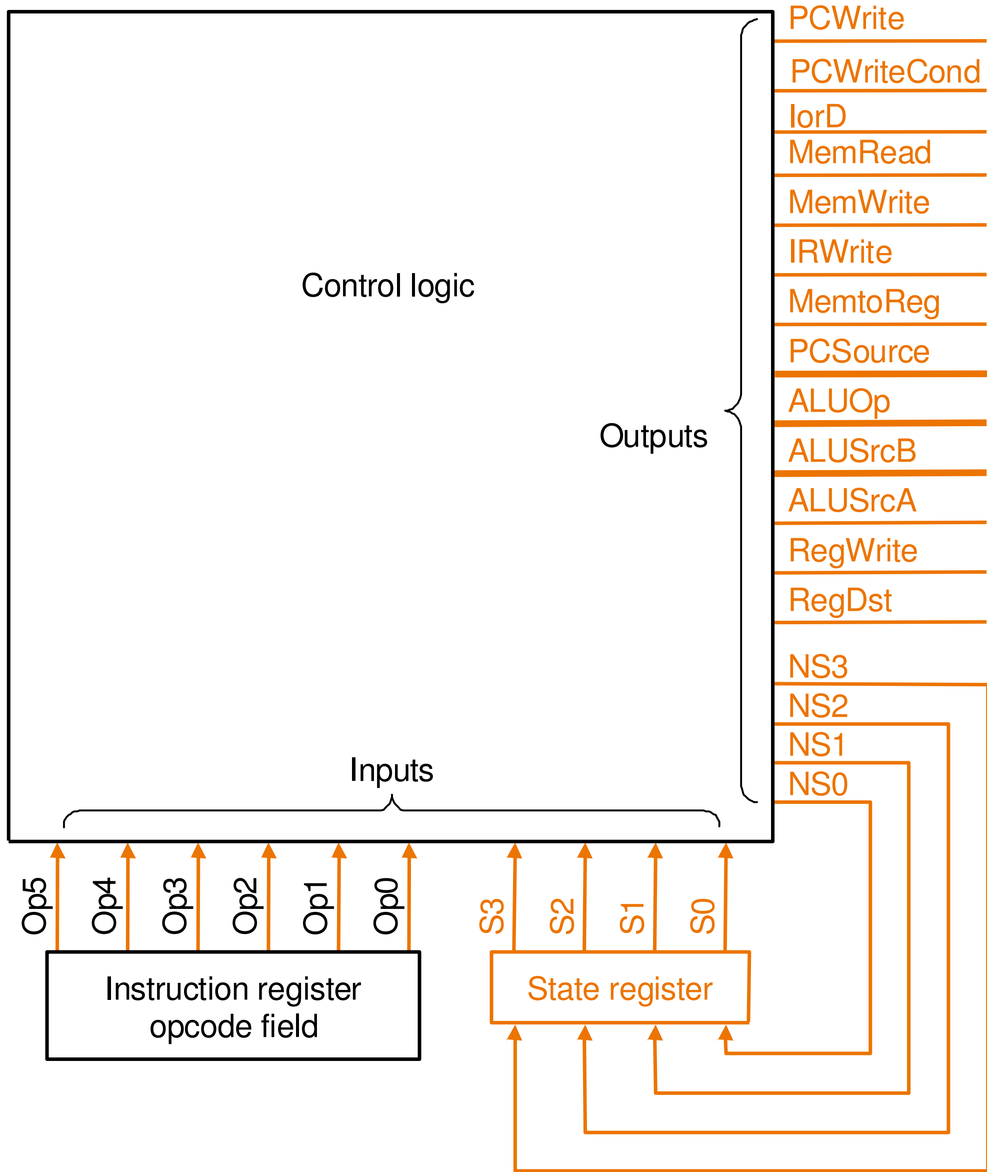
©1998 Morgan Kaufmann Publishers

Graphical Specification of FSM



- **How many state bits will we need?**

Finite State Machine for Control



©1998 Morgan Kaufmann Publishers

Desempenho desta máquina para o gcc

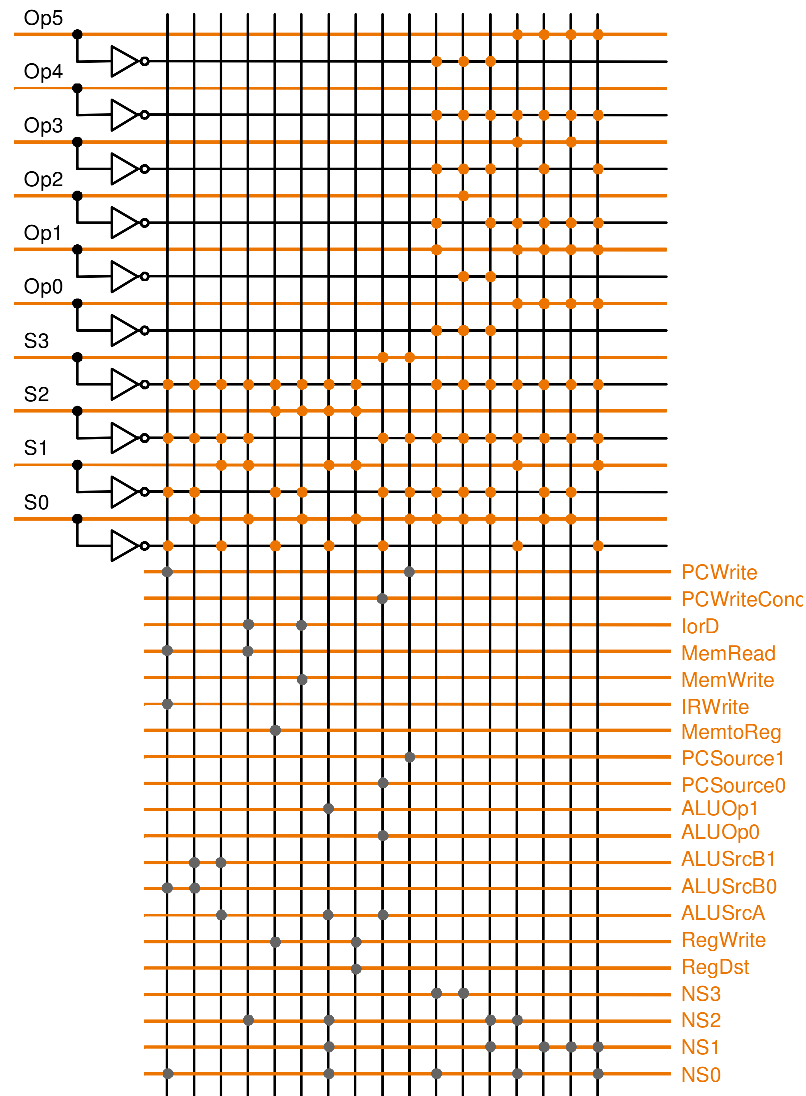
	lw	sw	Tipo R	beq	jump
%	23	13	43	19	2
CPI	5	4	4	3	3

- **CPI = 0,23*5 + 0,13*4 + 0,43*4 + 0,19*3 + 0,02*3 = 4,02**
- **Melhor do que se todas as instruções tomassem 5 ciclos**
- **Melhoria em MIPS (supor ck = 100 MHz)**
  - **CPI = 4 -> 25 MIPS**
  - **CPI = 5 -> 20 MIPS**
  - **CPI = 1 (fazer tudo em um ciclo) -> 100 MIPS**

©1998 Morgan Kaufmann Publishers

PLA Implementation

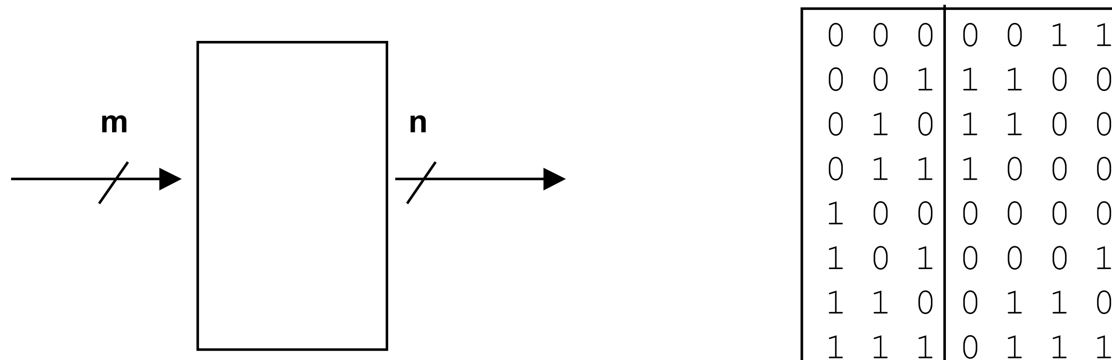
- **If I picked a horizontal or vertical line could you explain it?**



©1998 Morgan Kaufmann Publishers

ROM Implementation

- ROM = "Read Only Memory"
  - values of memory locations are fixed ahead of time
- A ROM can be used to implement a truth table
  - if the address is m-bits, we can address 2^m entries in the ROM.
  - our outputs are the bits of data that the address points to.



0	0	0	0	0	1	1
0	0	1	1	1	0	0
0	1	0	1	1	0	0
0	1	1	1	0	0	0
1	0	0	0	0	0	0
1	0	1	0	0	0	1
1	1	0	0	1	1	0
1	1	1	0	1	1	1

m is the "heigth", and n is the "width"

©1998 Morgan Kaufmann Publishers

ROM Implementation

- **How many inputs are there?**
  6 bits for opcode, 4 bits for state = 10 address lines
  (i.e., 2^{10} = 1024 different addresses)
- **How many outputs are there?**
  16 datapath-control outputs, 4 state bits = 20 outputs

- **ROM is 2^{10} x 20 = 1K x 20 bits (and a rather unusual size)**

- **Rather wasteful, since for lots of the entries, the outputs are the same**
  — i.e., opcode is often ignored

©1998 Morgan Kaufmann Publishers

ROM vs PLA

- Break up the table into two parts
  - — 4 state bits tell you the 16 outputs, 2^4 x 16 bits of ROM
  - — 10 bits tell you the 4 next state bits, 2^{10} x 4 bits of ROM
  - — Total: 4.3K bits of ROM
- PLA is much smaller
  - — can share product terms
  - — only need entries that produce an active output
  - — can take into account don't cares
- Size is (#inputs × #product-terms) + (#outputs × #product-terms)

  For this example = (10x17)+(20x17) = 460 PLA cells

- PLA cells usually about the size of a ROM cell (slightly bigger)

©1998 Morgan Kaufmann Publishers