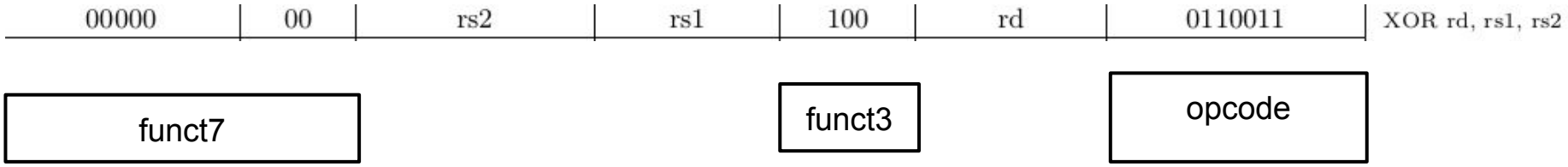
Adicionando XOR ao caminho de dados



Alterando o módulo *ControlUnit*

- Recebe como uma das entradas o **opcode** da instrução
- Determina o tipo da instrução e quais os sinais de controle precisam estar ativados para tal instrução.
- XOR é uma **instrução do tipo R** (identificada pelo opcode igual a 0110011). Logo, segue o código responsável por lidar com isso no módulo *ControlUnit* :

```
60 module ControlUnit (input [6:0] opcode, input [31:0] inst, output reg alusrc, memtoreg
61
62   always @(opcode) begin
63     alusrc   <= 0;
64     memtoreg <= 0;
65     regwrite <= 0;
66     memread  <= 0;
67     memwrite <= 0;
68     branch   <= 0;
69     aluop    <= 0;
70     ImmGen   <= 0;
71     case(opcode)
72       7'b0110011: begin // R type == 51
73         regwrite <= 1;
74         aluop    <= 2;
75       end
```

Alterando o módulo *alucontrol*

- No passo anterior note que atribuímos o valor 2 (binário 10) ao **aluop**
- Esse valor é uma entrada do módulo *alucontrol* e indica que a operação que a ALU deve executar será decidida com base nos campos *funct3* e/ou *funct7* da instrução.
- Caso o opcode seja 4 (100), que corresponde a XOR, o valor 3 é atribuído a saída *alucontrol*.

```
145  assign funct3 = funct[2:0];
146  assign funct7 = funct[9:3];
147
148  always @(aluop) begin
149    case (aluop)
150      0: alucontrol <= 4'd2; // ADD to SW and LW
151      1: alucontrol <= 4'd6; // SUB to branch
152      default: begin
153        case (funct3)
154          0: alucontrol <= (funct7 == 0) ? /*ADD*/ 4'd2 : /*SUB*/ 4'd6;
155          2: alucontrol <= 4'd7; // SLT
156          4: alucontrol <= 4'd3; // XOR
157          6: alucontrol <= 4'd1; // OR
158          //39: alucontrol <= 4'd12; // NOR
159          7: alucontrol <= 4'd0; // AND
160          default: alucontrol <= 4'd15; // Nop
161        endcase
```

Alterando o módulo *ALU*

- Por fim, temos que implementar a operação XOR na ALU.
- Caso a entrada *alucontrol* seja igual a 3 (tal como definimos no módulo *alucontrol*), executamos a operação de XOR bit a bit.

```
167 module ALU (input [3:0] alucontrol, input [31:0] A, B, output reg [31:0] aluout, outpu
168
169   assign zero = (aluout == 0); // Zero recebe um valor lógico caso aluout seja igual a
170
171   always @(alucontrol, A, B) begin
172       case (alucontrol)
173         0: aluout <= A & B; // AND
174         1: aluout <= A | B; // OR
175         2: aluout <= A + B; // ADD
176         3: aluout <= A ^ B; // XOR
177         6: aluout <= A - B; // SUB
178         //7: aluout <= A < B ? 32'd1:32'd0; //SLT
179         //12: aluout <= ~(A | B); // NOR
180       default: aluout <= 0; //default 0, Nada acontece;
181     endcase
182   end
183 endmodule
184
```

Testando a instrução

- Vamos utilizar a seguinte instrução para testar se a nossa implementação está correta.
  - xor x4, x3, x2
- Precisamos inserir a instrução na memória de instruções, ela está no módulo *fetch.*
- Vamos executar uma *nop* e uma *xor*.

```
14   initial begin
15     // Exemplos
16
17     inst_mem[0] <= 32'h00000000; // nop
18     inst_mem[1] <= 32'h21C233; // xor x4, x3, x2
19     //inst_mem[1] <= 32'h00500113; // addi x2, x0, 5  ok
20     //inst_mem[2] <= 32'h00210233; // add  x4, x2, x2  ok
21     //inst_mem[1] <= 32'h00202223; // sw x2, 8(x0) ok
22     //inst_mem[1] <= 32'h0050a423; // sw x5, 8(x1) ok
23     //inst_mem[2] <= 32'h0000a003; // lw x1, x0(0) ok
24     //inst_mem[1] <= 32'hfff00113; // addi x2,x0,-1 ok
25     //inst_mem[2] <= 32'h00318133; // add x2, x3, x3 ok
26     //inst_mem[3] <= 32'h40328133; // sub x2, x5, x3 ok
27   end
28
29 endmodule
```

Testando a instrução

- Se o sinal de *reset* não estiver acionado, nada é executado no datapath. Lembre-se de clicar no *check box* responsável por habilitar esse sinal.
- Obs: inicialmente o registradores x2 e x3 armazenavam os valores 2 e 3, respectivamente.