PCS-3225 Sistemas Digitais II — 17 de abril de 2024 - v1

Codificação de Instruções do RISC-V

Aula 10 - Exercício

Bruno Albertini, Edson Gomi e Ricardo Lera

Neste exercício usaremos as seguintes instruções do processador RISC-V: lw, sw, add e sub. A codificação das instruções está descrita na Figura 1.

Instruction	Format	funct7	rs2	rs1	funct3	rd	opcode
add (add)	R	0000000	reg	reg	000	reg	0110011
sub (sub)	R	0100000	reg	reg	000	reg	0110011
Instruction	**Format**	**immediate**		**rs1**	**funct3**	**rd**	**opcode**
addi (add immediate)	I	constant		reg	000	reg	0010011
lw (load word)	I	address		reg	010	reg	0000011
Instruction	**Format**	**immed -iate**	**rs2**	**rs1**	**funct3**	**immed -iate**	**opcode**
sw (store word)	S	address	reg	reg	010	address	0100011

Figura 1: Codificação das Instruções do RISC-V

As Figuras 2, 3 e 4 apresentam os formatos R, I e S das instruções do RISC-V.

funct7	rs2	rs1	funct3	rd	opcode
7 bits	5 bits	5 bits	3 bits	5 bits	7 bits

Figura 2: Formato R

immediate	rs1	funct3	rd	opcode
12 bits	5 bits	3 bits	5 bits	7 bits

Figura 3: Formato I

immediate[11:5]	rs2	rs1	funct3	immediate[4:0]	opcode
7 bits	5 bits	5 bits	3 bits	5 bits	7 bits

Figura 4: Formato S

A operação $f = (a+b)-(c+d)$ pode ser descrita com as seguintes instruções em assembly do RISC-V de 32 bits. As variáveis f, a, b, c e d estão a partir do endereço $0x00$ da memória RAM (que tem palavras de 32 bits de tamanho) e as instruções a partir do endereço $0x08$ da memória.

```
# Assumindo que a, b, c e d sao armazenados em sequencia
   comecando no endereco de memoria 0x00
# O resultado f sera armazenado no endereco 0x04

# Carregue os dados da memoria RA
lw x5, 0x01(x0)    # x5 = a, carregue do endereco 0x01
lw x6, 0x02(x0)    # x6 = b, carregue do endereco 0x02
lw x7, 0x03(x0)    # x7 = c, carregue do endereco 0x03
lw x8, 0x04(x0)    # x8 = d, carregue do endereco 0x04

# Calcule a + b and c + d
add x9, x5, x6      # x9 = a + b
add x10, x7, x8     # x10 = c + d

# Calcule (a + b) - (c + d)
sub x11, x9, x10    # x11 = (a + b) - (c + d)

# Armazene o resultado f no endereco 0x110 da memoria RAM
sw x11, 0x00(x0)
```

O circuito calc.v apresentado a seguir permitirá a inicialização da memória RAM. Você deve completar a descrição do circuito com o mapeamento de dados e instruções descrito acima. Inicialize as variáveis a, b, c e d com 33_{10} ,58_{10}, 47_{10} e 159_{10}, respectivamente.

```
module calc(
  input  [2:0]  ram_addr_juiz,
  input         clk,
  output [31:0] ram_out_juiz,
  output reg    done
);
  reg [2:0]  ram_addr;
  reg [31:0] ram_in;
  reg        ram_rw;

  wire [2:0] ram_addr_mux;
  assign ram_addr_mux = (ram_rw ? ram_addr : ram_addr_juiz);

  ram #(.W (32), .L (16)) mem
  (
    .addr     (ram_addr_mux),
    .data_in  (ram_in),
    .rw       (ram_rw),
    .oe       (1'b1),
    .data_out (ram_out_juiz)
  );

  initial
  begin
    done = 0;
    ram_rw = 1;
    ram_addr = 3'h00;
  end

  always@(posedge clk)
  begin
    if (done !== 1)
    begin
      if (ram_addr == 3'h01)
        ram_in = 32'hxxxxxxxx;

// continue o preenchimento da RAM aqui.
```

```
      end
    end
    else
      ram_rw = 0;

  end

endmodule
```

A descrição ram.v apresentada a seguir representa a memória RAM que será utilizada neste exercício.

```
module ram
#(parameter W=32, L=16)
(
  input  [$clog2(L)-1:0] addr,
  input  [W-1:0] data_in,
  input  rw, oe,
  output [W-1:0] data_out
);
  reg [W-1:0] mem[L-1:0];

  assign data_out = (rw==1'b0 & oe==1'b1) ? mem[addr] : {W{1'bz}};

  always@(*)
    if(rw===1'b1)
      mem[addr] = data_in;

endmodule
```

Faça o upload apenas do arquivo calc.v no Juiz. Não é preciso submeter o arquivo ram.v.

Cada integrante do time terá 10 tentativas no Juiz.