MC851 Projeto em Computação

Relatório Entrega 2

Equipe "RISC-VI":
RA 169374, Daniel Paulo Garcia
RA 182783, Lucca Costa Piccolotto Jordão
RA 185447, Paulo Barreira Pacitti
RA 198435, Guilherme Tavares Shimamoto
RA 216116, Gabriel Braga Proença
RA 221859, Mariana Megumi Izumizawa

Comunicação da equipe

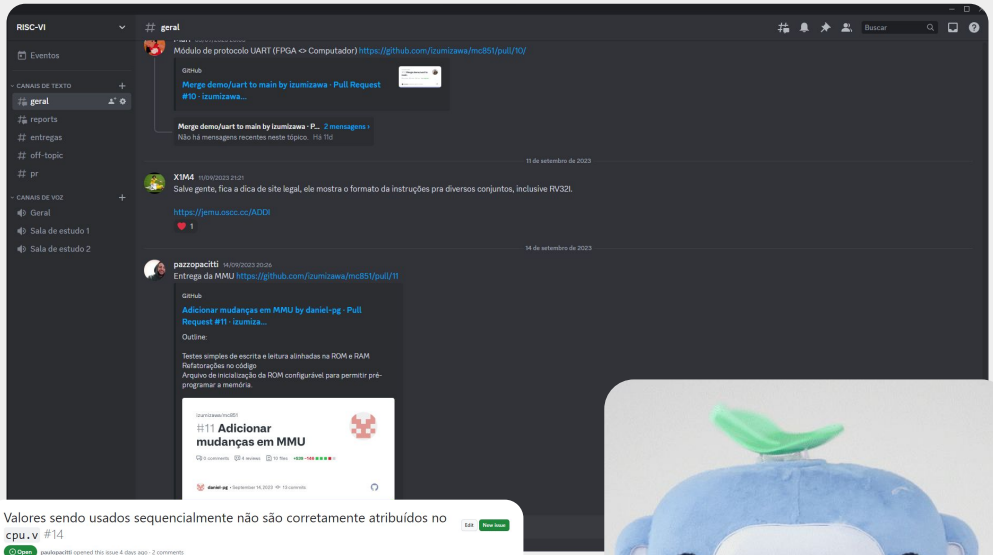
Antes

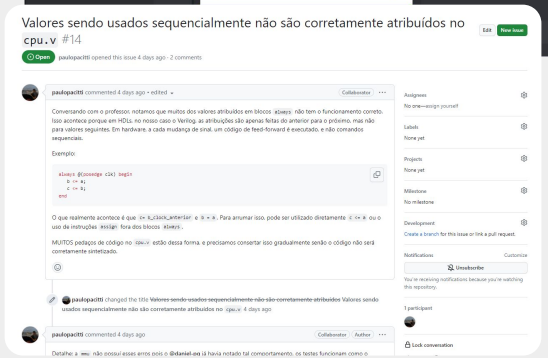
- Reuniões **semanais** no horário da aula
  - Dúvidas e discussões aguardavam até o momento do encontro
- Mensagens em grupo de **WhatsApp**
  - Documentação perdida
  - Acompanhamento difícil
  - Discussões diferentes atravessadas no mesmo canal

Comunicação da equipe

Hoje

- Reuniões **2x por semana** (terça e sexta)
- Mensagens em grupo de **WhatsApp**
  - Assuntos pontuais ou “rápidos”
- Servidor no **Discord**:
  - Canal de voz
  - Mensagens separadas por tópicos (PRs, dúvidas, entregas, reports)
- **Github**
  - Pull requests (comentários, discussões)
  - Issues com bugs





Organização

Dividimos novamente as duplas:

Lucca && Gabriel

Paulo && Daniel

Mariana && Guilherme

Planilha com sinais

MMU

Decode de instruções

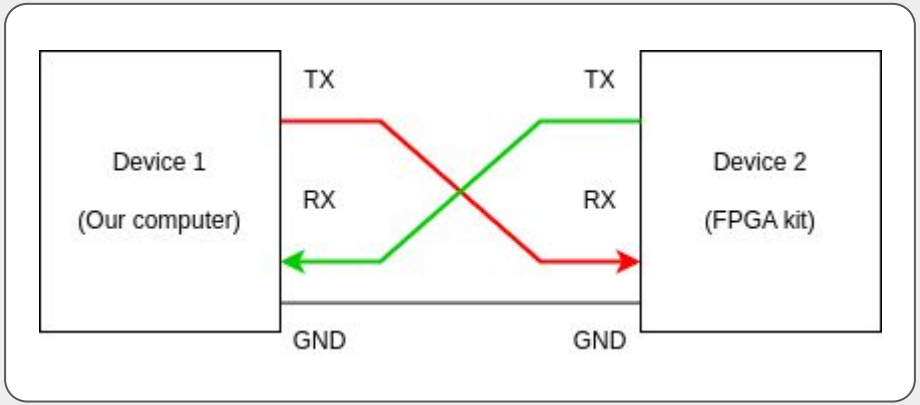
Periféricos

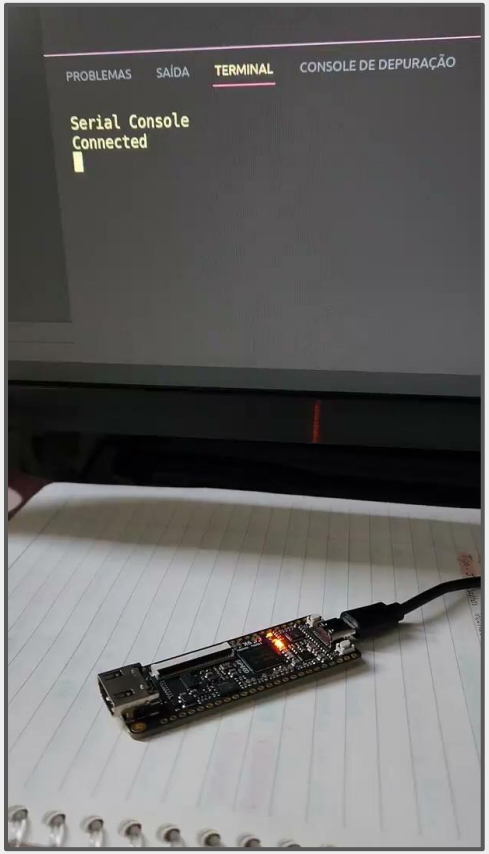
Periféricos

- Pesquisa sobre opções
- Protocolo UART
- Serial Console
- Wavetrace
- Serial Ports

Periféricos

Protocolo UART (transmissor/receptor assíncrono universal), para a troca de dados seriais entre dois dispositivos.





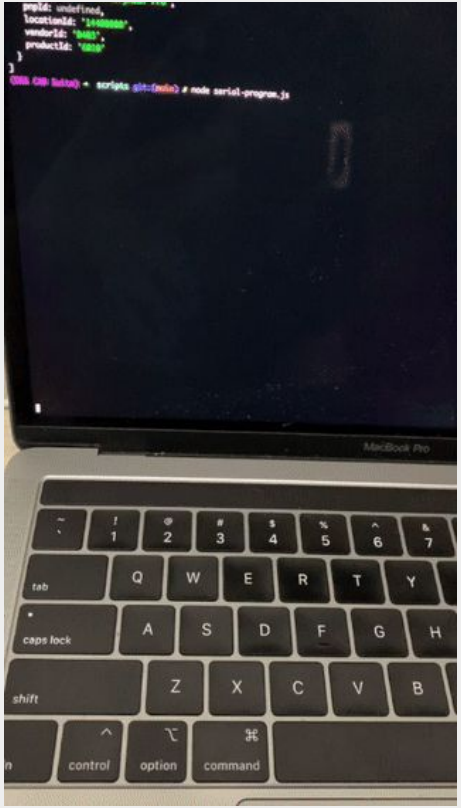
Circuito lógico implementado com Verilog, módulo com protocolo UART. Implantado no Tang Nano 9k.

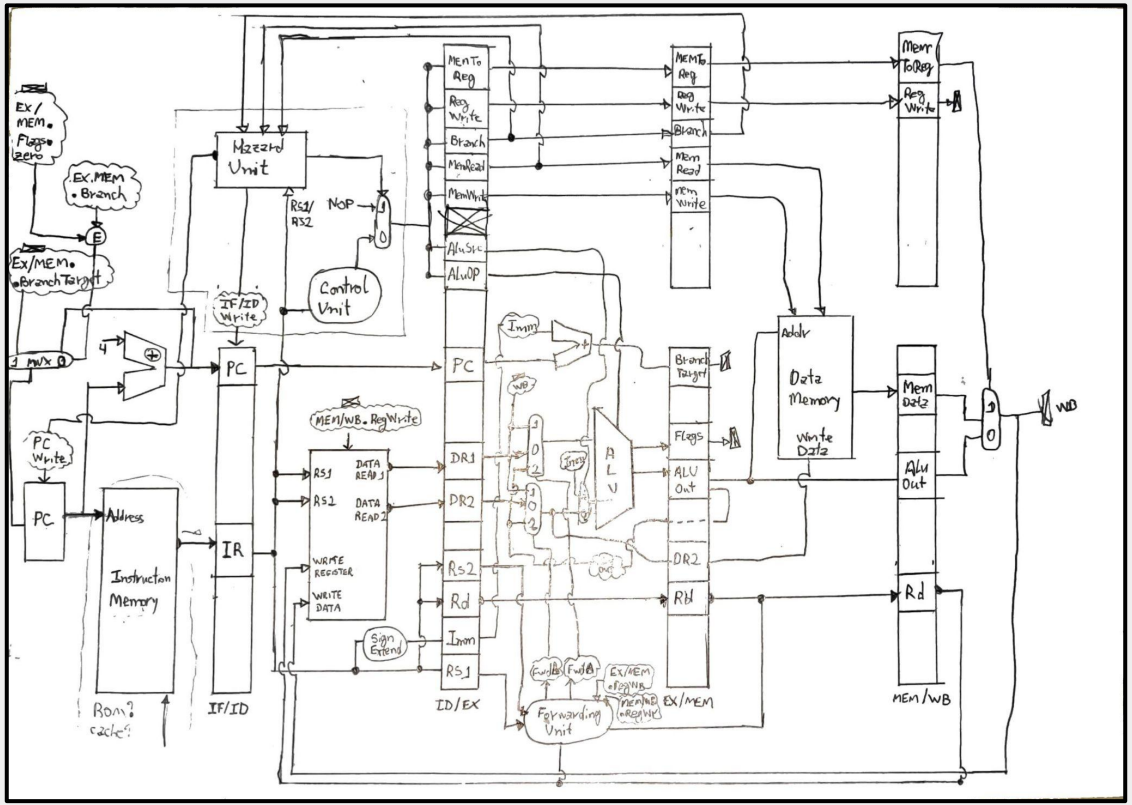
Ao inserir os caracteres no teclado do computador, acende os LEDs do FPGA.

E ao apertar o botão, imprime a mensagem que estava gravada na placa.

Serial Ports

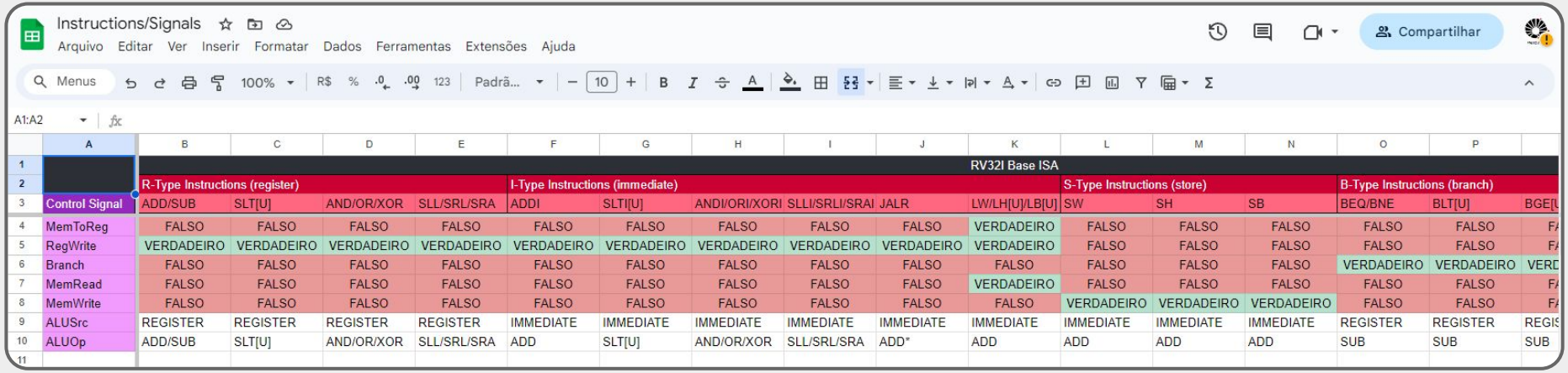
- Script em *NodeJS* utilizando a biblioteca *serialport*
- Placa envia dados para o computador
- Computador recebe e envia dados para placa
- Possibilidade de tratar dados do lado do computador





"Danielpath": datapath utilizado no nosso sistema computacional.

Planilha com sinais



Instructions/Signals

Arquivo Editar Ver Inserir Formatar Dados Ferramentas Extensões Ajuda

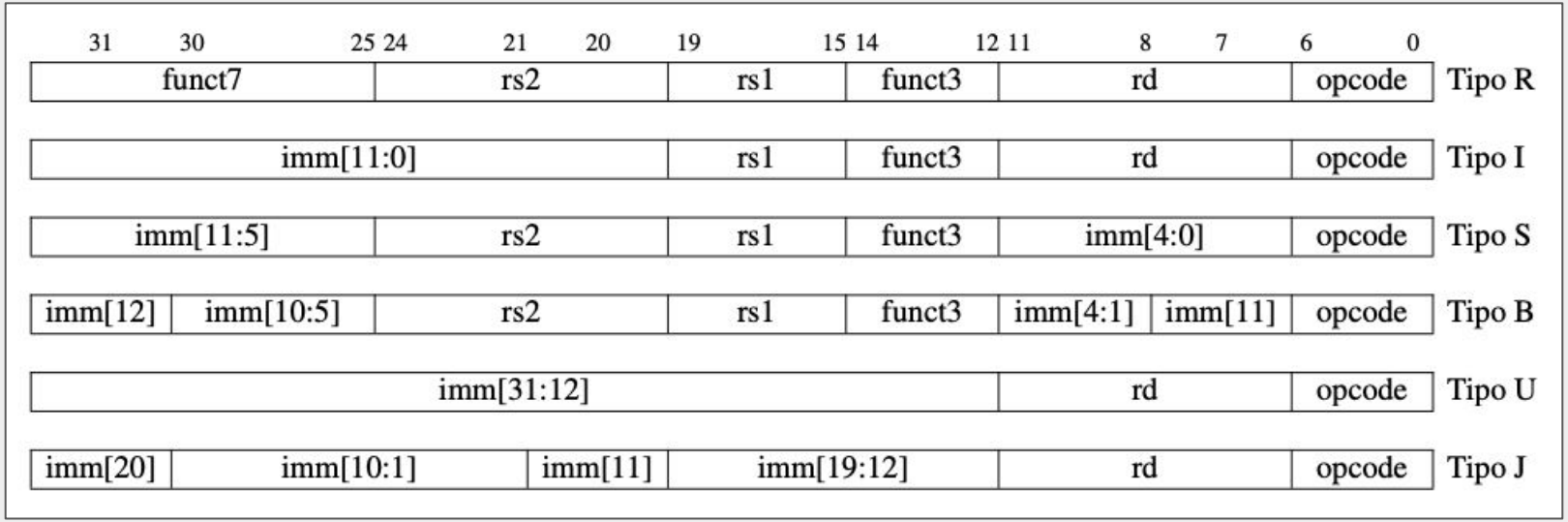
Compartilhar

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	
1		RV32I Base ISA															
2		R-Type Instructions (register)				I-Type Instructions (immediate)						S-Type Instructions (store)			B-Type Instructions (branch)		
3	Control Signal	ADD/SUB	SLT[U]	AND/OR/XOR	SLL/SRL/SRA	ADDI	SLTI[U]	ANDI/ORI/XORI	SLLI/SRLI/SRAI	JALR	LW/LH[U]/LB[U]	SW	SH	SB	BEQ/BNE	BLT[U]	BGE[U
4	MemToReg	FALSO	FALSO	FALSO	FALSO	FALSO	FALSO	FALSO	FALSO	FALSO	VERDADEIRO	FALSO	FALSO	FALSO	FALSO	FALSO	FA
5	RegWrite	VERDADEIRO	VERDADEIRO	VERDADEIRO	VERDADEIRO	VERDADEIRO	VERDADEIRO	VERDADEIRO	VERDADEIRO	VERDADEIRO	VERDADEIRO	FALSO	FALSO	FALSO	FALSO	FALSO	FA
6	Branch	FALSO	FALSO	FALSO	FALSO	FALSO	FALSO	FALSO	FALSO	FALSO	FALSO	FALSO	FALSO	FALSO	VERDADEIRO	VERDADEIRO	VERD
7	MemRead	FALSO	FALSO	FALSO	FALSO	FALSO	FALSO	FALSO	FALSO	FALSO	VERDADEIRO	FALSO	FALSO	FALSO	FALSO	FALSO	FA
8	MemWrite	FALSO	FALSO	FALSO	FALSO	FALSO	FALSO	FALSO	FALSO	FALSO	FALSO	VERDADEIRO	VERDADEIRO	VERDADEIRO	FALSO	FALSO	FA
9	ALUSrc	REGISTER	REGISTER	REGISTER	REGISTER	IMMEDIATE	IMMEDIATE	IMMEDIATE	IMMEDIATE	IMMEDIATE	IMMEDIATE	IMMEDIATE	IMMEDIATE	IMMEDIATE	REGISTER	REGISTER	REGIS
10	ALUOp	ADD/SUB	SLT[U]	AND/OR/XOR	SLL/SRL/SRA	ADD	SLT[U]	AND/OR/XOR	SLL/SRL/SRA	ADD*	ADD	ADD	ADD	ADD	SUB	SUB	SUB
11																	

```v
`include "define.v"
`include "../components/register_file.v"
`include "../components/alu_module.v"

module cpu (
    input         clk,
    input         reset_n,
    input         mmu_mem_ready,
    input [31:0]  mmu_data_out,
    output        mmu_write_enable,
    output        mmu_read_enable,
    output        mmu_mem_signed_read,
    output [ 1:0] mmu_mem_data_width,
    output [31:0] mmu_address,
    output [31:0] mmu_data_in
);
    reg [31:0] ifid_pc;
    reg [31:0] ifid_ir;

    reg [31:0] idex_pc;
    reg idex_reset;
    reg [2:0] idex_branch_op;
    reg idex_reg_write;
    reg idex_mem_read;
    reg idex_mem_write;
    reg idex_mem_to_reg;
    reg idex_alu_src;
    reg [ 3:0] idex_alu_op;
    reg [31:0] idex_data_read_1;
    reg [31:0] idex_data_read_2;
    reg [ 4:0] idex_rs1;
    reg [ 4:0] idex_rs2;
    reg [ 4:0] idex_rd;
    reg [31:0] idex_imm;

    reg exmem_reset;
    reg [2:0] exmem_branch_op;
    reg exmem_mem_to_reg;
    reg exmem_reg_write;
    reg exmem_mem_read;
    reg exmem_mem_write;
    reg [31:0] exmem_branch_target;
    reg [ 3:0] exmem_flags;
    reg [31:0] exmem_alu_out;
    reg [31:0] exmem_data_read_2;
    reg [ 4:0] exmem_rd;

    reg [31:0] memwb_mem_data_read;
    reg [31:0] memwb_alu_out;
    reg [ 4:0] memwb_rd;
    reg memwb_reg_write;
    reg memwb_mem_to_reg;

...
endmodule
```

Cabeçalho da implementação da CPU. É possível notar a declaração dos registradores de *pipeline* pelos prefixos nas variáveis.

Decode de instruções



Decode de instruções

```
case (opcode)
    // R-type instructions
    7'b0110011: begin
        idex_reg_write <=  1; // True
        idex_alu_src <= `ALU_SRC_FROM_REG;
        if (funct3 == 3'b000) begin
            if (funct7 == 7'b0000000) begin
                idex_alu_op <= `ALU_ADD;
            end else begin
                idex_alu_op <= `ALU_SUB;
            end
        end else if (funct3 == 3'b001) begin
            idex_alu_op <= `ALU_SLL;
        end else if (funct3 == 3'b010) begin
            idex_alu_op <= `ALU_SLT;
        end else if (funct3 == 3'b011) begin
            idex_alu_op <= `ALU_SLTU;
        end else if (funct3 == 3'b100) begin
            idex_alu_op <= `ALU_XOR;
        end else if (funct3 == 3'b101) begin
            if (funct7 == 7'b0000000) begin
                idex_alu_op <= `ALU_SRL;
            end else begin
                idex_alu_op <= `ALU_SRA;
            end
        end else if (funct3 == 3'b110) begin
            idex_alu_op <= `ALU_OR;
        end else if (funct3 == 3'b111) begin
            idex_alu_op <= `ALU_AND;
        end
    end
```

```
// I-type instructions
7'b0010011: begin
    idex_reg_write <=  1; // True
    idex_alu_src <= `ALU_SRC_FROM_IMM;
    idex_imm <= i_imm_extended;
    if (funct3 == 3'b000) begin
        idex_alu_op <= `ALU_ADD;
    end else if (funct3 == 3'b010) begin
        idex_alu_op <= `ALU_ADD;
    end else if (funct3 == 3'b011) begin
        idex_alu_op <= `ALU_SLT;
    end else if (funct3 == 3'b100) begin
        idex_alu_op <= `ALU_XOR;
    end else if (funct3 == 3'b110) begin
        idex_alu_op <= `ALU_OR;
    end else if (funct3 == 3'b111) begin
        idex_alu_op <= `ALU_AND;
    end else if (funct3 == 3'b001) begin
        idex_imm <= shamt_extended;
        idex_alu_op <= `ALU_SLL; //slli
    end else if (funct3 == 3'b101) begin
        idex_imm <= shamt_extended;
        if (funct7 == 7'b0000000) begin
            idex_alu_op <= `ALU_SRL; //srli
        end else begin
            idex_alu_op <= `ALU_SRA; //srai
        end
    end
end
```

Obtenção de imediato - BEQ

```
task test_imm_beq();
  begin
      $write("  test imm: ");

      // Instrucao
      // 0 101011 10011 00101 000 1101 1 1100011
      // Resultado
      // 00000000000000000000 1 101011 1101 0
      //
      #10
      ifid_ir <= 32'b01010111001100101000110111100011;

      #10
      imm <= { ifid_ir[31], ifid_ir[7], ifid_ir[30:25], ifid_ir[11:8], 1'b0 };

      #10
      idex_imm <= { { 20 { imm[12] } }, imm[11:0] };

      #10
      $display("idex_imm is %b", idex_imm);
  end
endtask
```

MMU

- Definição da ROM;
- Definição da RAM;
- Unidade de gerenciamento de memória -> MMU;
  - Máquina de estados;
  - Endereçamento virtual;
  - Sinais de escrita/leitura;

```
module rom #(
    parameter ADDR_WIDTH = 8,  // 256×4B = 1 KiB
    parameter ROMFILE="test.mem"
) (
    input clk,
    input read_enable,
    input [ADDR_WIDTH-1:0] address,
    output reg [31:0] data_out
);

    reg [31:0] mem [0:2**ADDR_WIDTH-1];

    initial begin
        $readmemh(ROMFILE, mem);
    end

    always @(posedge clk) begin
        if (read_enable)
            data_out <= mem[address];
        else
            data_out <= 0;
    end

endmodule
```

```
module ram #(
    parameter ADDR_WIDTH = 8    // 256×4B = 1 KiB
) (
    input clk,
    input write_enable,
    input read_enable,
    input [ADDR_WIDTH-1:0] address,
    input [31:0] data_in,
    output reg [31:0] data_out
);

    reg [31:0] mem [0:2**ADDR_WIDTH-1];

    always @(posedge clk) begin
        if (read_enable)
            data_out <= mem[address];
        else
            data_out <= 0;

        if (write_enable) mem[address] <= data_in;
    end

endmodule
```

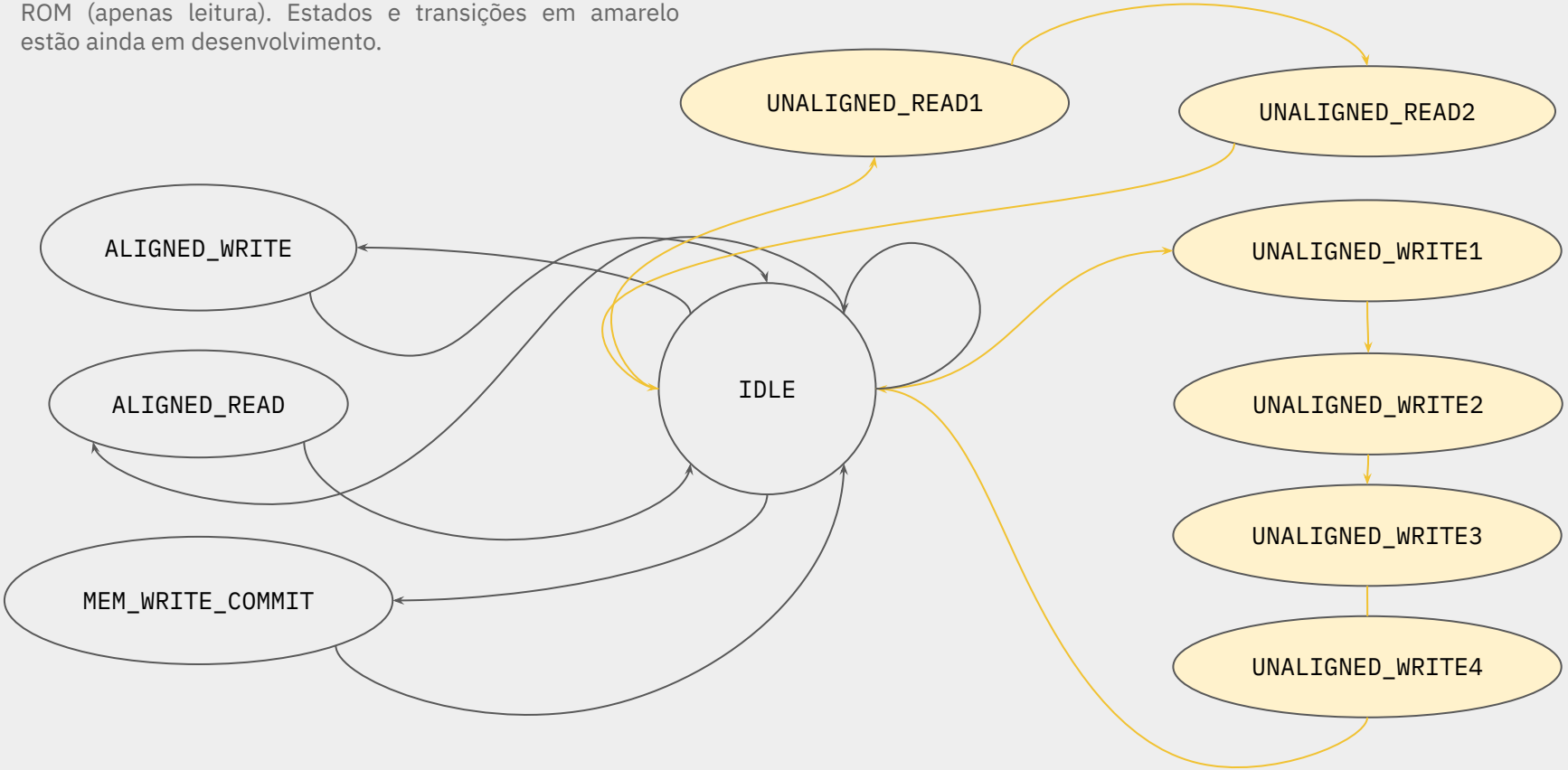
Implementação dos módulos de RAM e ROM.

```
`include "define.v"
`include "components/ram.v"
`include "components/rom.v"

module mmu #(
    parameter ROMFILE="test.mem"
) (
    input clk, reset_n,
    input write_enable,
    input read_enable,
    input mem_signed_read,
    input [ 1:0] mem_data_width,
    input [31:0] address,
    input [31:0] data_in,
    output reg [31:0] data_out,
    output reg mem_ready
);


 /*****************************************************
     * ROM (Read-Only Memory)
     */
    localparam ROM_ADDR_WIDTH = 8;
    reg rom_read_enable;
    wire  [ROM_ADDR_WIDTH-1:0] rom_address;
    wire [31:0] rom_data_out;

    rom #(
        .ADDR_WIDTH(ROM_ADDR_WIDTH),
        .ROMFILE(ROMFILE)
    ) rom_inst (
        .clk            (clk            ),
        .read_enable    (rom_read_enable),
        .address        (rom_address    ),
        .data_out       (rom_data_out   )
    );
    // -------------------------------------------------
```

```
    /*****************************************************
     * RAM (Random Access Memory)
     */
    localparam RAM_ADDR_WIDTH = 8;
    reg ram_write_enable;
    reg ram_read_enable;
    wire  [RAM_ADDR_WIDTH-1:0] ram_address;
    wire  [31:0] ram_data_in;
    wire [31:0] ram_data_out;

    ram #( .ADDR_WIDTH(RAM_ADDR_WIDTH) ) ram_inst (
        .clk            (clk                ),
        .write_enable   (ram_write_enable   ),
        .read_enable    (ram_read_enable    ),
        .address        (ram_address        ),
        .data_in        (ram_data_in        ),
        .data_out       (ram_data_out       )
    );
    // -------------------------------------------------
 ...
endmodule
```

Cabeçalho da implementação da MMU. Contém as instâncias da memória ROM e RAM e cria os sinais de controle interno e externo, para que a CPU consiga ler e escrever nos módulos de memória.

Máquina de estados da MMU para leitura e escrita na RAM e ROM (apenas leitura). Estados e transições em amarelo estão ainda em desenvolvimento.



SoC: *system-on-a-chip*

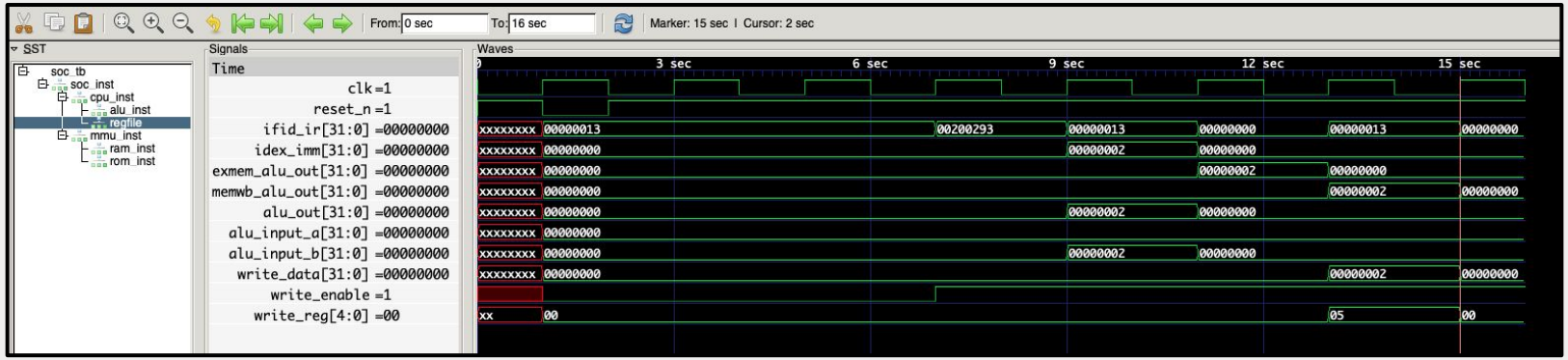
- *top module*;
- Conecta a CPU com a memória. Módulo central do sistema computacional implementado utilizando a arquitetura RISC-V

```
`include "cpu.v"
`include "mmu.v"

module soc (
    input clk,
    input reset_n
);
    wire         mmu_mem_ready;
    wire [31:0]  mmu_data_out;
    wire         mmu_write_enable;
    wire         mmu_read_enable;
    wire         mmu_mem_signed_read;
    wire [ 1:0]  mmu_mem_data_width;
    wire [31:0]  mmu_address;
    wire [31:0]  mmu_data_in;

    cpu cpu_inst (
        .clk (clk),
        .reset_n (reset_n),
        .mmu_mem_ready(mmu_mem_ready),
        .mmu_data_out(mmu_data_out),
        .mmu_write_enable(mmu_write_enable),
        .mmu_read_enable(mmu_read_enable),
        .mmu_mem_signed_read(mmu_signed_read),
        .mmu_mem_data_width(mmu_mem_data_width),
        .mmu_address(mmu_address),
        .mmu_data_in(mmu_data_in)
    );

    mmu #( .ROMFILE("../../src/memdump/beq.mem")) mmu_inst (
        .clk(clk),
        .reset_n(reset_n),
        .write_enable(mmu_write_enable),
        .read_enable(mmu_read_enable),
        .mem_signed_read(mmu_signed_read),
        .mem_data_width(mmu_mem_data_width),
        .address(mmu_address),
        .data_in(mmu_data_in),
        .data_out(mmu_data_out),
        .mem_ready(mmu_mem_ready)
    );
endmodule
```

Implementação do `soc.v`, mostrando a integração entre MMU e CPU.

À direita, a saída dos testes automatizados, incluindo o último que testa um programa em assembly. Abaixo está o *wavetrace* do programa em assembly que testa a instrução `addi`.

```
(OSS CAD Suite) → paulopacitti@loch  ~/Documents/unicamp/mc851/code/mc851/rtl/tests git:(if) × bash test.sh
===Testing alu_tb.v=======================================================
alu_module: starting tests
  test_add: passed!
  test_sub: passed!
  test_and: passed!
  test_or: passed!
  test_xor: passed!
  test_sll: passed!
  test_srl: passed!
  test_sra: passed!
  test_slt: passed!
  test_slt: passed!

===Testing mmu_tb.v=======================================================
VCD info: dumpfile mmu_wave.vcd opened for output.
memory_control_tb: starting tests
  test_ram_read_and_write: passed!
  test_rom_read: passed!
mmu_tb.v:98: $finish called at 31 (1s)

===Testing register_file_tb.v=============================================
SUCCESS: Test 1 - Escrita e leitura básica
SUCCESS: Test 2 - Escrita em um registrador diferente e leitura no mesmo registrador
SUCCESS: Test 3 - Leitura de registrador inexistente
SUCCESS: Test 4 - Leitura após uma escrita em um registrador diferente
SUCCESS: Test 5 - Escrita em dois registradores diferentes e leitura nos dois registradores
SUCCESS: Teste 6 - Tentativa de escrita no registrador x0
register_file_tb.v:124: $finish called at 1170 (1s)

===Testing soc_tb.v=======================================================
VCD info: dumpfile soc_wave.vcd opened for output.
soc_tb: starting tests
  test_write_and_read:  passed!
soc_tb.v:36: $finish called at 16 (1s)
```



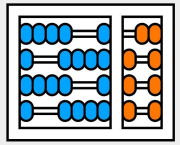
Futuro

- Implementação da cache L1: aumentar otimização do sistema computacional.
- Controle de acesso de memória: priorizar e controlar *pipeline* de acordo com o uso da MMU pela CPU.
- Implantação do sistema na placa FPGA.

Referências

D. A. Patterson and J. L. Hennessy, *Computer Organization and Design RISC-V Edition: The Hardware Software Interface*, 1st ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2017

Repositório com código do sitema computacional: https://github.com/izumizawa/mc851



Obrigado!

Equipe "RISC-VI":
RA 169374, Daniel Paulo Garcia
RA 182783, Lucca Costa Piccolotto Jordão
RA 185447, Paulo Barreira Pacitti
RA 198435, Guilherme Tavares Shimamoto
RA 216116, Gabriel Braga Proença
RA 221859, Mariana Megumi Izumizawa