

Platforma lab 1 + lab 2

1. Introducere

Primele 2 sedinte de laborator au urmatoarele roluri:

1. Recapitularea sintaxei Verilog / SystemVerilog;
2. Recapitularea notiunilor fundamentale legate de procesoare;
3. Scrierea unui procesor functional minimal, care va constitui baza pentru dezvoltarile ulterioare;
4. Testarea procesorului facut prin scrierea de cod masina (tradus din instructiuni ASM).

Sintaxa Verilog poate fi gasita la:

https://wiki.dcae.pub.ro/index.php/Circuite_Integrate_Digitale

Pentru a usura procesul de proiectare se vor folosii si cateva elemente de sintaxa specifice SystemVerilog (SV) (observatie: cand faceti fisierele, sa nu uitati sa le setati ca SV), anume:

1. Orice element de tip “wire” / “reg” va deveni “logic”. “logic” este un tip de date specific SV ce ajuta la evitarea confuziilor privind fire si registre;
2. Structurile de tip always combinational: “always@(*)” vor deveni “always_comb”;
3. Structurile de tip always secvential: “always@(posedge clock)” vor deveni “always_ff@(posedge clock)”.

Elementele fundamentale ale unui procesor sunt:

1. Ralu: alcatuit din ALU (unitatea aritmetica si logica) si un bloc de registri (register file);
2. PC – Program Counter, un numarator a carui valoare este adresa instructiunii curente (ce este citita si executata);
3. Blocuri de control ce genereaza semnalele de control necesare: selectii pentru multiplexoare, enable pentru scrieri si numaratoare.

Suplimentar, pe langa procesor vor mai fi necesare o memorie de instructiuni si o memorie de date ca sa obtinem un calculator functional.

Pentru ca procesorul sa faca ceva, este nevoie ca in memoria de instructiuni sa fie scrise instructiunile ce se vor executa. Aceste instructiuni vor fi scrise in cod masina (binar), pornind de la setul de instructiuni al procesorului (ISA, Instruction Set Architecture), punand valorile dorite (cod de operatie, adrese, valoare) in campurile corespunzatoare (conform ISA).

2. Etapele lucrării

Lucrarea are următoarele etape:

1. Citirea documentației și înțelegerea setului de instrucțiuni și al arhitecturii;
2. Scrierea ALU;
3. Scrierea RALU (integrare ALU);
4. Scrierea procesorului (integrare Ralu);
5. Scrierea calculatorului (integrare procesor);
6. Scrierea testbench-ului și testarea calculatorului prin scrierea unui program simplu.

Procesorul va fi un procesor RISC, fără pipeline, calculatorul va fi de tip arhitectura Harvard (memorii separate de instrucțiuni și de date).

Instrucțiunile vor fi pe 32b, datele pe 16b.

Procesorul va fi cu RegisterFile, nu cu acumulator.

ISA

Toate instrucțiunile sunt pe 32b și au următorul format

Command				Value
Opcode	Destination addr	Operand0 addr	Operand1 addr	Value
4b	4b	4b	4b	16b

Câmpul “Opcode” specifică operația pe care procesorul o va efectua. Pentru majoritatea operațiilor este nevoie de o adresă destinație (unde se salvează rezultatul) și adrese pentru operandii ce constituie datele procesate.

Unele coduri de operație nu sunt implementate.

Instructiunile suportate sunt prezentate mai jos:

Opcode	Operation	Description	Misc
0	NOP	---	
1	ADD	$R[\text{dest}] = R[\text{op0_addr}] + R[\text{op1_addr}]$	
2	SUB	$R[\text{dest}] = R[\text{op0_addr}] - R[\text{op1_addr}]$	
3	MULT	$R[\text{dest}] = R[\text{op0_addr}] * R[\text{op1_addr}]$	
4	SHIFT 1 RIGHT	$R[\text{dest}] = R[\text{op1_addr}] \gg 1;$	
5			
6	AND	$R[\text{dest}] = R[\text{op0_addr}] \& R[\text{op1_addr}]$	
7	OR	$R[\text{dest}] = R[\text{op0_addr}] R[\text{op1_addr}]$	
8	XOR	$R[\text{dest}] = R[\text{op0_addr}] \wedge R[\text{op1_addr}]$	
9			
10	VALUE LOAD	$R[\text{dest}] = \text{value_from_instr}$	alu passes operand1
11	JMP	$\text{PC} = \text{value_from_instr}$	alu passes operand1
12	JMPZ	if Z_flag: $\text{PC} = \text{value_from_instr}$	alu passes operand1
13	WRITE MEM (STORE)	$\text{mem}[R[\text{op1_addr}]] = R[\text{op0_addr}]$	alu passes operand 0
14	READ MEM (LOAD)	$R[\text{dest}] = \text{mem}[R[\text{op0_addr}]]$	alu passes operand1
15	HALT	$\text{PC} = \text{PC};$	

Schema bloc

Calculatorul este construit dupa schemele bloc prezentate in continuare.

Blocul de registri (Register File) este o memorie RAM de 16 locatii x 16b.

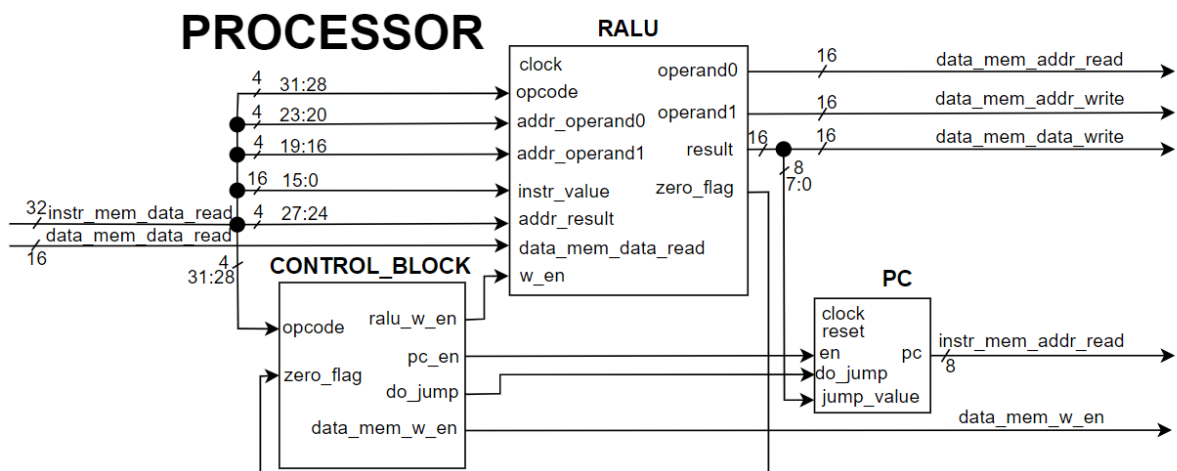
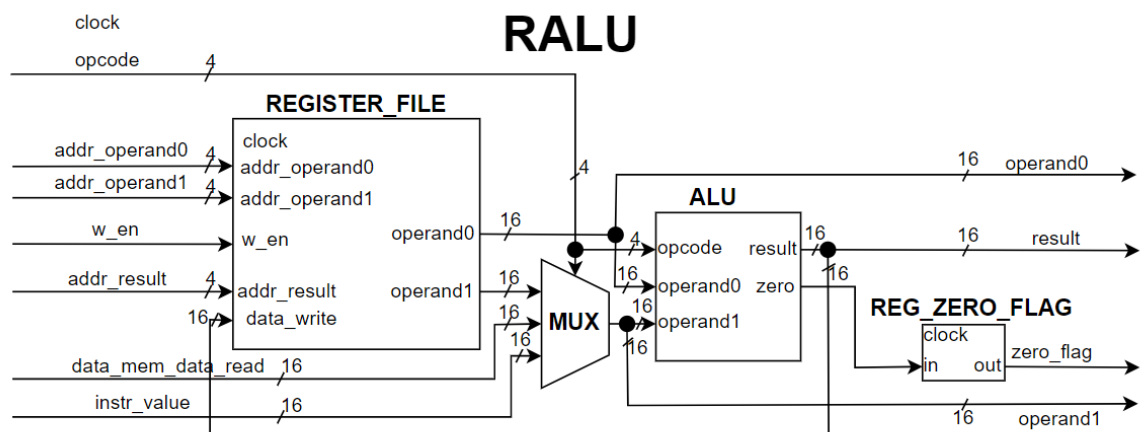
Pentru simplitate, firul de clock si firul de reset nu au mai fost trase pe desen.

Memoria de date si memoria de instructiuni nu au registru de pipeline pe iesire (pentru simplitatea sincronizarilor, in practica acel registru apare).

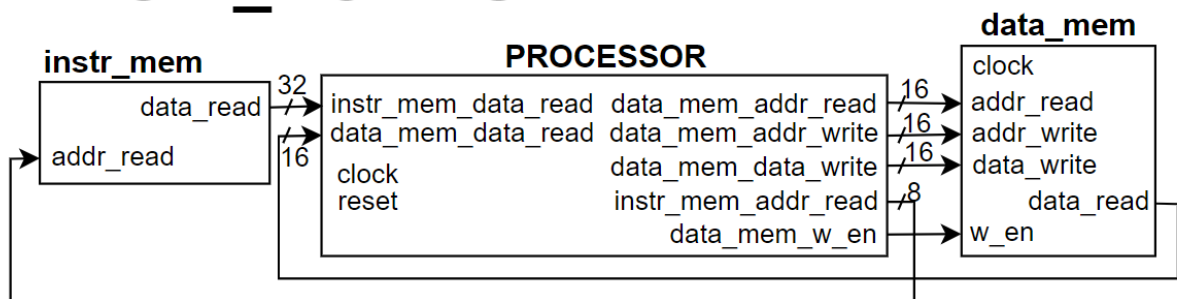
Memoria de date este de [0:65_535] locatii a cate 16b fiecare.

Memoria de instructiuni este de [0:255] locatii a cate 32b fiecare.

Multiplexorul pentru operand 1 va avea nevoie si de niste logica suplimentara pentru generarea semnalului de selectie in functie de opcode XOR se face un mux cu 16 intrari si se leaga toate cele 16 intrari posibile in functie de instructiunea ce se executa.



BASIC_COMPUTER



Deoarece nu avem un mecanism prin care sa scriem din exterior catre memoria de instructiuni, scrierea se va face printr-un bloc de tip “initial” plasat in testbench. Astfel scrierea se face ca in exemplul de mai jos:

```
initial
    begin
        dut.mem_instr_0.mem[0] = 32'b1010_0101_0000_0000__0000_0000_0000_0010;
        // value load: R[5] = 2
        dut.mem_instr_0.mem[1] = 32'b0000_0000_0000_0000__0000_0000_0000_0000;
        // NOP
    end
```

La adresa 0 din elementul de memorare “mem” care se afla in instanta “mem_instr_0” care se afla in instanta “dut” (care se afla in tb), se atribuie valoarea pe 32b data. Aceasta valoare este optinuta prin specificarea campului “opcode” cu valoarea binara 1010 (care conform tabelului ce descrie setul de instructiuni este functia “value load”), specificarea adresei destinatie cu valoarea 0101 adica registrul 5, si specificarea valorii ce se va incarca, anume 2. Instructiunea implementata este practic $R[5] = 2$;

La urmatoarea adresa se adauga instructiunea NOP.

Pentru testarea sa, memoria de instructiuni a calculatorului va fi scrisa cu instructiuni de timp “load value” care sa puna valori in registrii, apoi cateva operatii de calcul cu respectivele valori si la final testarea instructiunii de jump if zero. Programul se incheie prin instructiunea de “HALT”.