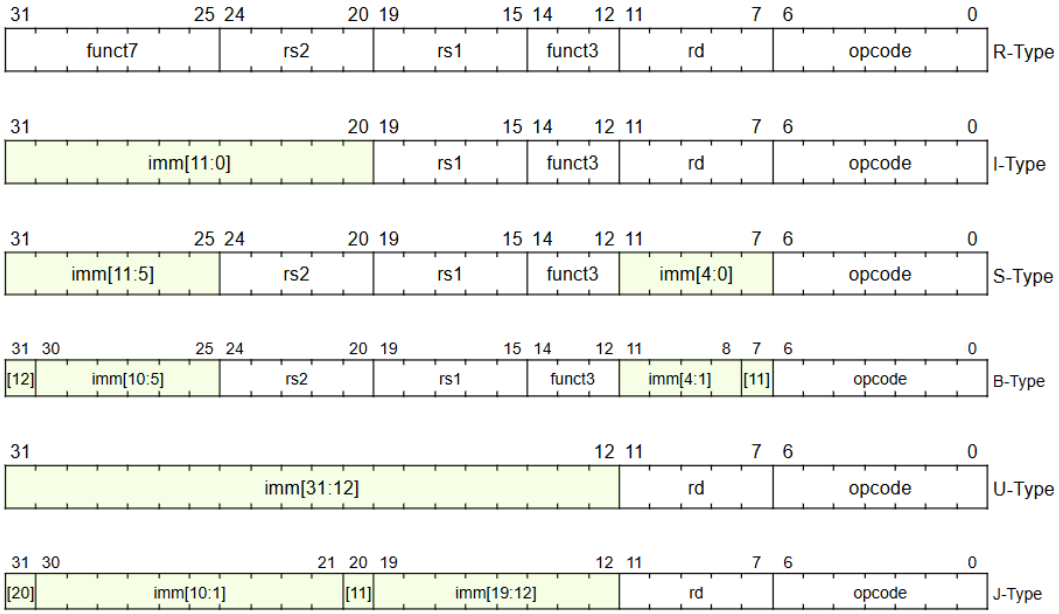
Program Studi Teknik Elektro ITB

Nama Kuliah (Kode) : Praktikum Arsitektur Sistem Komputer (EL3111)
Tahun / Semester : 2023-2024 / Ganjil
Modul : 1 RV32I INSTRUCTION SET,REGISTER,DAN MEMORY
Nama Asisten / NIM :
Nama Praktikan / NIM : Muhammad Falih Rosyidi / 13223095

Tugas Pendahuluan

1. Alur Eksekusi Intruksi:
    a. IF (Instruction Fetch) : adalah tahap dimana berbagai alur data instruksi di olah, baik alur intruksi berikutnya ataupun alur saat ini. Data instruksi yang dijalankan saat ini di ambil dari instruksi memori.
    b. ID (Instruction Decode) : adalah tahap mememecah kode instruksi saat ini, menjadi sesuai formatnya sehingga program yang diinginkan bisa berjalan.
    c. EX (Execute) : adalah tahap dimana berbagai data dari kode instruksi di olah pada Arithmatic Logic Unit atau ALU.
    d. MEM (Data Memory) : adalah tahap dimana data disimpan atau diambil dari data memory untuk instruksi tertentu
    e. WB (Write Back) : adalah tahap dimana data hasil akhir setelah di olah disimpan kembali baik pada register destinasi

    Format Instruksi RV32I:



    a. R type: adalah instruksi yang memiliki 2 register source atau datanya semua ada dalam register
    b. I type: adalah instruksi yang memili data berupa nilai immediate atau konstan dalam kodenya
    c. S type: adalah instruksi yang berfungsi untuk menyimpan data ke memori data
    d. B type: adalah instruksi conditional jump dimana akan berpindah ke instruksi tertentu jika memenuhi kondisinya
    e. U type: adalah instruksi memasuki nilai immediate ke register seperti LUI
    f. J type: adalah instruksi unconditional jump yakni pasti akan berpindah ke instruksi tertentu tanpa perlu kondisi tertentu

    Komponen yang ada pada setiap instruksi set adalah:

a. Opcode: adalah kode yang menjelaskan tipe intruksi dan keluarga interuksi yang ada di dalamnya
b. Rd: adalah register destination atau register tujuan data disimpan
c. Rs1 dan Rs2: adalah register source yakni register dimana data yang diambil
d. Funct3 dan Funct7: adalah sub-opcode atau kode yang menentukan operasi dalam varian opcode dari instruksi kode ini
e. Immediate: adalah nilai konstanta yang ditentukan dari kode program

2. Instruksi RV32I

Instruksi	Tipe	Opcode[6:0]	Funct3	Funct7	Arti singkat
SLLI	I	0010011	001	-	x[rd] = x[rs1] << imm[4:0]
SRLI	I	0010011	101	-	x[rd] = x[rs1] >>> imm[4:0]
JALR	I	1100111	000	-	x[rd] = pc+4; pc = (x[rs1]+sext(imm[11:0])
ADD	R	0110011	000	0000000	x[rd] = x[rs1] + x[rs2]
SUB	R	0110011	000	0100000	x[rd] = x[rs1] - x[rs2]
AND	R	0110011	111	0000000	x[rd] = x[rs1] & x[rs2]
OR	R	0110011	110	0000000	x[rd] = x[rs1] | x[rs2]
XOR	R	0110011	100	0000000	x[rd] = x[rs1] ^ x[rs2]
NOR	-	-	-	-	-
SLT	R	0110011	010	0000000	if (x[rs1] < x[rs2]) x[rd] = 1 else x[rd] = 0
BEQ	B	1100011	000	-	if (x[rs1] == x[rs2]) pc += sext({imm[12:1], 1'b0}) else pc += 4
BNE	B	1100011	001	-	if (x[rs1] != x[rs2]) pc += sext({imm[12:1], 1'b0}) else pc += 4
ADDI	I	0010011	000	-	x[rd] = x[rs1] + sext(imm[11:0])
SLTI	I	0010011	010	-	if (x[rs1] < sext(imm[11:0]) x[rd] = 1 else x[rd] = 0
ANDI	I	0010011	111	-	x[rd] = x[rs1] & sext(imm[11:0])
ORI	I	0010011	110	-	x[rd] = x[rs1] | sext(imm[11:0]
XORI	I	0010011	100	-	x[rd] = x[rs1] ^ sext(imm[11:0])
LUI	U	0110111	-	-	x[rd] = sext(imm[31:12] << 12)
LW	I	0000011	010	-	x[rd] = sext(M[x[rs1] + sext(imm[11:0])][31:0])
SW	S	0100011	010	-	M[x[rs1] + sext(imm[11:0])][31:0] = x[rs2][31:0]
JAL (rd=x0)	J	1101111	-	-	x[rd] = pc+4; pc += sext(imm[20:1]), walau terlaksana rx = x0 akan selalu bernilai 0
JAL	J	1101111	-	-	x[rd] = pc+4; pc += sext(imm[20:1])
ADDI	I	0010011	000	-	x[rd] = x[rs1] + sext(imm[11:0])
SLTIU	I	0010011	011	-	if (x[rs1] <u sext(imm[11:0]) x[rd] = 1 else x[rd] = 0

Note:

- sext adalah sign extended
- pc adalah program counter
- NOR dapat diperoleh dengan di OR kan kedua input baru di xori dengan -1 (xori -1 adalah operasi bitwiese not pada riscv)

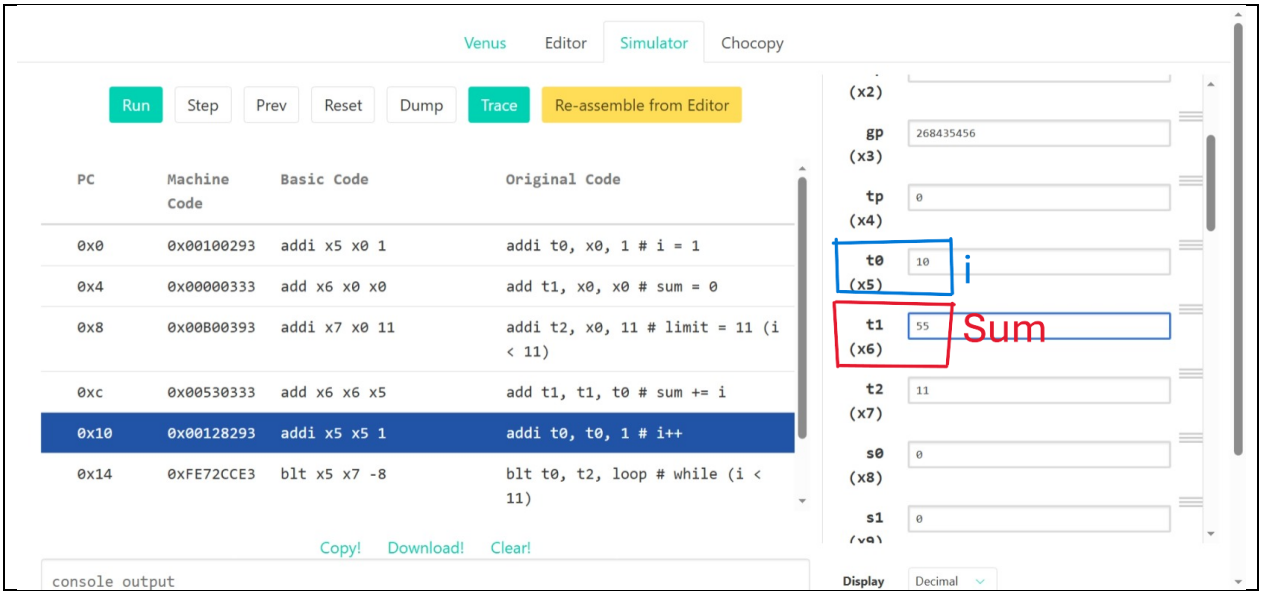
3. Assembly Code
   a. KODE

```
.text
.globl main
main:
    addi t0, x0, 1        # i = 1
    add  t1, x0, x0       # sum = 0
    addi t2, x0, 11       # limit = 11 (i < 11)
```

```
loop:
    add  t1, t1, t0       # sum += i
    addi t0, t0, 1        # i++
    blt  t0, t2, loop     # while (i < 11)

    # exit program
    addi a0, x0, 0        # return 0
```

b. VERIFIKASI



c. MACHINE CODE

```
0x00100293
0x00000333
0x00B00393
0x00530333
0x00128293
0xFE72CCE3
0x00000513
```

4. Testbench register file

```
// Modul         : 1
// Percobaan     : TP
// Tanggal       : 4 November 2025
// Kelompok      :
// Rombongan     :
// Nama (NIM) 1  : Muammar Qadafi (13223094)
// Nama (NIM) 2  : Muhammad Falih Rosyidi (13223095)
// Nama File     : tb_reg_file_rv32i.v
// Deskripsi     : Testbench Register file RV32I, 32x32, 2 read ports, 1 write port.
// x0 selalu 0; write @posedge, read @negedge
// Testbench untuk reg_file_rv32i
// Deskripsi:
// 1. Menulis ke x1, x2, x3, x4
// 2. Mencoba menulis ke x0 (untuk verifikasi write-ignore)
// 3. Membaca x0-x9
// 4. Self-checking: membandingkan hasil baca dengan expected
// 5. Melaporkan PASS atau FAIL

`timescale 1ns/1ps
`include "../Tugas4/reg_file_rv32i.v"
module tb_reg_file_rv32i;
reg clock;

// Input ke DUT
reg        cu_rdwrite;    // write enable dari CU
reg [4:0]  rs1_addr;      // alamat baca port 1
reg [4:0]  rs2_addr;      // alamat baca port 2
reg [4:0]  rd_addr;       // alamat tulis (write-back)
reg [31:0] rd_in;         // data tulis (write-back data)
```

```
// Output dari DUT
wire [31:0] rs1;           // data baca port 1
wire [31:0] rs2;           // data baca port 2

// Sinyal Internal Testbench
integer fail_flag; // Penanda jika ada kegagalan
reg [31:0] expected_values [0:9]; // Array untuk menyimpan nilai yang diharapkan

// Instansiasi DUT
reg_file_rv32i DUT (
    .clock(clock), .cu_rdwrite(cu_rdwrite),
    .rs1_addr(rs1_addr), .rs2_addr(rs2_addr),
    .rd_addr(rd_addr), .rd_in(rd_in),
    .rs1(rs1), .rs2(rs2)
);

// Clock Generator
localparam CLK_PERIOD = 10;
always #(CLK_PERIOD/2) clock = ~clock;

// VCD Dump untuk GTKWave
initial begin
    $dumpfile("tb_reg_file_rv32i.vcd");
    $dumpvars(0, tb_reg_file_rv32i);
end

// SIMULASI UTAMA
initial begin
    // 1. Inisialisasi Sinyal
    clock = 0;
    cu_rdwrite = 0;
    rs1_addr = 5'b0;
    rs2_addr = 5'b0;
    rd_addr = 5'b0;
    rd_in = 32'b0;
    fail_flag = 0; // Set flag ke 0 (belum gagal)

    // Inisialisasi nilai yang diharapkan
    expected_values[0] = 0; // x0 selalu 0
    // Nilai yang kita tulis
    expected_values[1] = 32'h11;
    expected_values[2] = 32'h22;
    expected_values[3] = 32'h33;
    expected_values[4] = 32'h44;
    // Register lain (x5-x9) harus 0 (nilai inisialisasi DUT)
    expected_values[5] = 32'h0;
    expected_values[6] = 32'h0;
    expected_values[7] = 32'h0;
    expected_values[8] = 32'h0;
    expected_values[9] = 32'h0;

    #CLK_PERIOD; // Tunggu 1 siklus

    cu_rdwrite = 1;
    // Mulai menulis
    // Tulis x1
    @(posedge clock);
    rd_addr = 1;
    rd_in = expected_values[1];
    #1;
    // Tulis x2
    @(posedge clock);
    rd_addr = 2;
    rd_in = expected_values[2];
    #1;
    // Tulis x3
    @(posedge clock);
    cu_rdwrite = 1;
    rd_addr = 3;
    rd_in = expected_values[3];
    #1;
    // Tulis x4
    @(posedge clock);
    cu_rdwrite = 1;
    rd_addr = 4;
    rd_in = expected_values[4];
    #1;
```

```
    // Tes tambahan: Coba tulis ke x0 (seharusnya diabaikan oleh DUT)
    @(posedge clock);
    cu_rdwrite = 1;
    rd_addr = 0;
    rd_in = 32'hDEADBEEF;
    #1;

    @(posedge clock);
    cu_rdwrite = 0; // Matikan write enable setelah selesai
    rd_addr = 5'bx; // Set ke 'x' untuk deteksi bug
    rd_in   = 32'bx;
    #CLK_PERIOD;

    // 3. Fase Baca & Verifikasi
    @(posedge clock);
    rs1_addr = 0;
    rs2_addr = 1;
    @(negedge clock);
    #1;
    if (rs1 !== expected_values[0]) fail_flag = 1;
    if (rs2 !== expected_values[1]) fail_flag = 1;

    @(posedge clock);
    rs1_addr = 2;
    rs2_addr = 3;
    @(negedge clock);
    #1;
    if (rs1 !== expected values[2]) fail flag = 1;
    if (rs2 !== expected_values[3]) fail_flag = 1;

    @(posedge clock);
    rs1_addr = 4;
    rs2_addr = 5;
    @(negedge clock);
    #1;
    if (rs1 !== expected_values[4]) fail_flag = 1;
    if (rs2 !== expected_values[5]) fail_flag = 1;

    @(posedge clock);
    rs1 addr = 6;
    rs2_addr = 7;
    @(negedge clock);
    #1;
    if (rs1 !== expected_values[6]) fail_flag = 1;
    if (rs2 !== expected_values[7]) fail_flag = 1;

    @(posedge clock);
    rs1_addr = 8;
    rs2_addr = 9;
    @(negedge clock);
    #1;
    if (rs1 !== expected_values[8]) fail_flag = 1;
    if (rs2 !== expected_values[9]) fail_flag = 1;

    #CLK_PERIOD;

    // Laporan Hasil Akhir
    if (fail_flag == 0) $display("--- HASIL: SEMUA TES PASS! ---");
    else $display("--- HASIL: ADA TES YANG FAIL! ---");

    $finish;
end

endmodule
```

5. Kenggulan dan Trade Off penggunaan Altera®/Intel® ALTSYNCRAM

Karena dalam implementasi FPGA kemungkinan besar RAM manual akan memakai LUT dan itu akan memakan banyak area. Sedangkan dengan mengunakan ALTSYNCRAM kita menggunakan block RAM yang sudah terdapat pada FPGA. Oleh karenhala itu, maka menggunakan ALTSYNCRAM akan membuat pengggunaan resource pada FPGA lebih baik sehingga LUT nya tidak terpakai banyak. Selain itu, akan membuat kinerja menjadi lebih optimal karena timing clock yang lebih presisi. Dan terakhir yakni memudahkan dalam meng-inisialisasi memori.

REFERENSI:

[1] https://danielmangum.com/posts/risc-v-bytes-intro-instruction-formats/#r-format

[2] https://docs.openhwgroup.org/projects/cva6-user-manual/01_cva6_user/RISCV_Instructions_RV32I.html
[3] https://dejazzer.com/coen2710/lectures/RISC-V-Reference-Data-Green-Card.pdf