Program Studi Teknik Elektro ITB

Nama Kuliah (Kode) : Arsitektur Sistem Komputer (EL3011)
Tahun / Semester : 2023-2024 / Ganjil
Modul : SYNTHESIZABLE RISC-V (RV32I) MICROPROCESSOR BAGIAN I: INSTRUCTION SET, REGISTER, DAN MEMORY
Nama Asisten / NIM : _____
Nama Praktikan / NIM : William Anthony

Tugas Pendahuluan

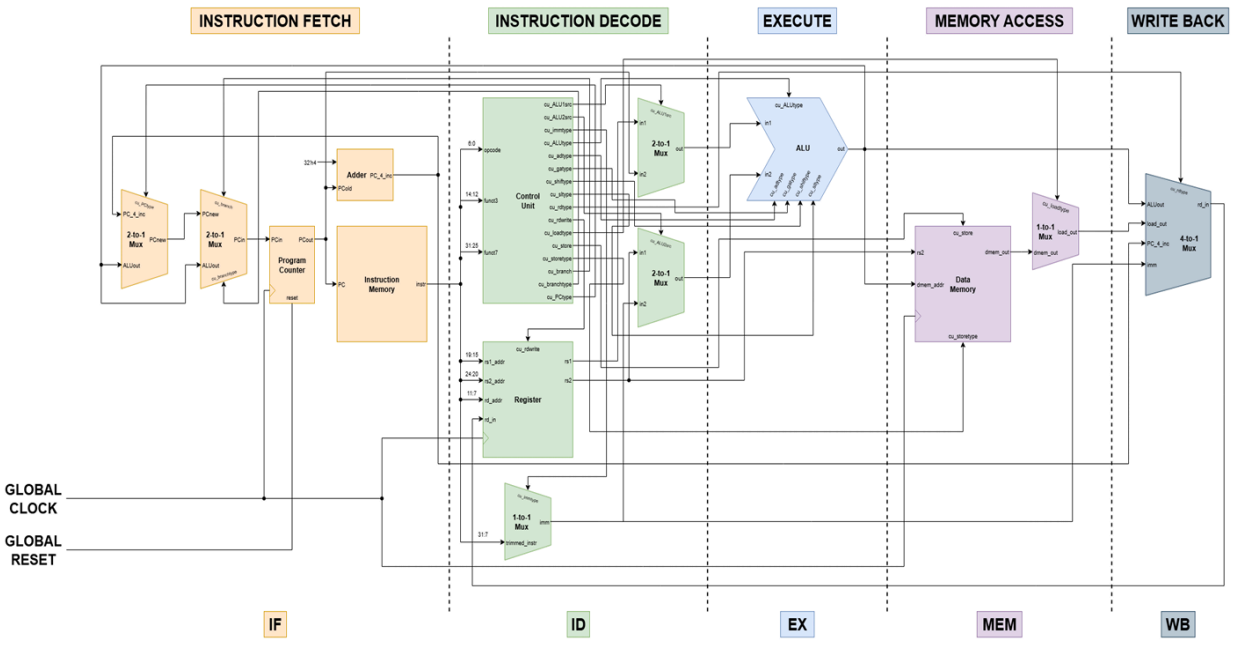
1. Jelaskan alur eksekusi instruksi pada prosesor single-cycle: IF, ID, EX, MEM, WB (meski pada single-cycle semua diselesaikan dalam 1 siklus)! Jelaskan semua format instruksi RV32I dan arti setiap field bit-nya!

Pada prosesor **single-cycle**, seluruh instruksi diselesaikan hanya dalam **satu siklus clock**, Ini akan melalui 5 tahap utama sebagai berikut.

Tahap	Nama Tahap	Fungsi Lengkap
1. IF (Instruction Fetch)	Pengambilan Instruksi	Tahap instruction fetch berfungsi mengatur aliran instruksi yang akan diolah pada tahap berikutnya. Instruksi yang sedang dijalankan merupakan instruksi yang berasal dan disimpan dari memory. Pada arsitektur ini, memory lalu dipisahkan menjadi dua bagian yaitu instruction memory yang berfungsi menyimpan instruksi-instruksi yang akan dieksekusi dan data memory yang berfungsi untuk menyimpan data untuk menghindari structural hazard. Singkatnyam prosesor membaca instruksi dari Instruction Memory menggunakan alamat dari Program Counter (PC). Setelah instruksi diambil, nilai PC akan bertambah 4 (karena setiap instruksi 32 bit = 4 byte) untuk menyiapkan pengambilan instruksi berikutnya.
2. ID (Instruction Decode & Register Fetch)	Penerjemahan dan Pembacaan Register	Instruksi yang telah diambil dipecah menjadi bagian-bagian seperti opcode, rs1, rs2, rd, funct3, funct7, dan immediate. Pada tahap ini juga dilakukan pembacaan nilai dari register file berdasarkan rs1 dan rs2 untuk dijadikan input ALU atau alamat memori. Tahap berikutnya, instruksi yang telah diambil (fetched) dari instruction memory berpindah ke tahap instruction decode. Untuk perjelas, lebar 32-bit akan dipecah sesuai format instruksi yang digunakan. Penjelasan mengenai decoding instruksi ini dapat dilihat pada bagian selanjutnya, tetapi secara umum akan dipecah menjadi bagian alamat register destinasi (rd), alamat register sumber/source (rs1 dan rs2), dan opcode atau kode yang menandakan operasi yang akan dijalankan (misalkan, instruksi add dan sub memiliki opcode yang sama karena

		keduanya instruksi tipe-R, dijelaskan berikutnya).
3. EX (Execute / Effective Address Calculation)	Eksekusi Operasi Logika / Aritmatika	Tahap ini melakukan operasi utama instruksi di ALU (Arithmetic Logic Unit). Contohnya: Jika instruksi aritmatika (ADD, SUB, AND, SLT), ALU akan menghitung hasil operasi. Jika instruksi load/store, ALU menghitung alamat efektif dari data di memori (rs1 + immediate). Jika instruksi branch, ALU membandingkan dua operand dan menentukan apakah lompat (branch) dilakukan atau tidak.
4. MEM (Memory Access)	Akses Data Memory (bila diperlukan)	Pada tahap ini, data disimpan dan/atau diambil dari data memory. Data memory hanya dapat disimpan atau dibaca jika ada sinyal MemRead dan/atau MemWrite yang sesuai sehingga operasi baca dan/atau tulis pada data memory dapat dilakukan. Jika instruksi adalah load (LW), data dibaca dari alamat hasil perhitungan ALU. Jika instruksi adalah store (SW), data dari register akan disimpan ke alamat tersebut. Instruksi selain itu akan melewati tahap ini tanpa perubahan.
5. WB (Write Back)	Menulis Kembali ke Register	Hasil akhir dari ALU (atau data yang dibaca dari memori) lalu akan dikembalikan ke register tujuan (rd) agar dapat digunakan oleh instruksi berikutnya. Pada instruksi SW atau BEQ, tahap ini dilewati karena tidak menghasilkan nilai ke register. Sehingga tahap terakhir ini digunakan untuk mengalirkan data dari data memory atau hasil perhitungan arithmetic and logical unit (ALU) ke register untuk disimpan, guna dipakai untuk instruksi-instruksi berikutnya.

Dapat diamati dari diagram tahap-tahap dibawah ini. Intinya **prosesor single-cycle** dijalankan **secara paralel dalam satu siklus clock**, sehingga hasil operasi langsung diperoleh di akhir siklus tersebut. *Hence* penamaannya single-cycle.



RISC-V RV32I memiliki lebar instruksi 32-bit. Format atau tipe dasarnya ada enam: R, I, S, B, U, J. Dengan ini komponen dari format dasar instruksi dijelaskan pada tabel selanjutnya.

Format	Struktur Bit	Keterangan & Contoh
R-type	`funct7 [31:25]	rs2 [24:20]
I-type	`imm[11:0]	rs1
S-type	`imm[11:5]	rs2
B-type	`imm[12]	imm[10:5]
U-type	`imm[31:12]	rd
J-type	`imm[20]	imm[10:1]

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	funct7							rs2					rs1					funct3			rd					opcode						
I	imm												rs1					funct3			rd					opcode						
S	imm[11:5]							rs2					rs1					funct3			imm[4:0]					opcode						
B	imm[12] imm[10:5]							rs2					rs1					funct3			imm[4:1] imm[11]					opcode						
U	imm[31:25]							imm[24:20]					imm[19:15]					imm[14:12]			rd					opcode						
J	imm[20] imm[10:5]							imm[4:1] imm[11]					imm[19:15]					imm[14:12]			rd					opcode						

Untuk keterangan masing-masing *field*, ada di sini :

Komponen	Keterangan
`opcode [6:0]`	Menentukan kelas/format instruksi dan keluarga operasinya.
`rd/rs1/rs2`	Indeks register tujuan (rd) dan sumber (rs1, rs2).
`funct3/funct7`	Sub-opcode untuk membedakan varian operasi dalam satu opcode.
`immediate`	Konstanta bertanda; tata letak tergantung format (PC-relative untuk B/J, upper-immediate untuk U).

2. Tentukan tipe instruksi, opcode[6:0], funct3, funct7 (jika ada), dan arti singkat untuk instruksi-instruksi RV32I berikut.

Instruksi	Format	opcode[6:0]	funct3	funct7	Arti Singkat
SLLI	I	10011	001	0	Shift kiri logis immediate
SRLI	I	10011	101	0	Shift kanan logis immediate
JALR	I	1100111	000	–	Lompat ke alamat (rs1 + imm), simpan PC+4 ke rd
SUB	R	110011	000	100000	Pengurangan antar register
AND	R	110011	111	0	Operasi AND
OR	R	110011	110	0	Operasi OR

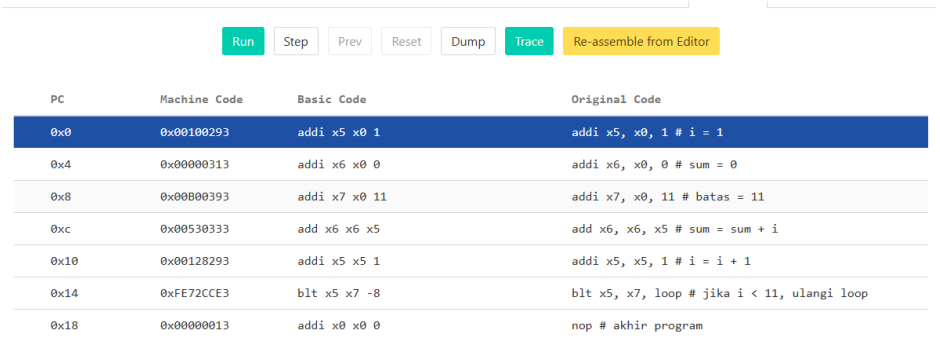
NOR	–	–	–	–	Tidak tersedia di RV32I (ekuivalen dengan XORI + NOT)
SLT	R	110011	010	0	Set rd = 1 jika rs1 < rs2 (signed)
ADDI	I	10011	000	–	Penjumlahan register + immediate
SLTI	I	10011	10	–	Set rd = 1 jika rs1 < imm (signed)
BEQ	B	1100011	000	–	Lompat jika rs1 == rs2
ANDI	I	10011	111	–	AND dengan immediate
XORI	I	10011	100	–	XOR dengan immediate
LUI	U	110111	–	–	Load upper immediate (20 bit tinggi)
LW	I	11	010	–	Load word dari memori
JAL (rd=x0)	J	1101111	–	–	Lompat tanpa menyimpan PC+4
JAL	J	1101111	–	–	Lompat dan simpan PC+4 ke rd
ADD	R	110011	000	0	Penjumlahan antar register
XOR	R	110011	100	0	XOR antar register
BNE	B	1100011	001	–	Lompat bila rs1 ≠ rs2
ORI	I	10011	110	–	OR dengan immediate
SW	S	100011	010	–	Simpan word ke memory
SLTIU	I	10011	011	–	Set rd=1 jika rs1 < imm (unsigned)

3. Buat test vector sederhana yang akan digunakan untuk menguji instruction memory pada Tugas 1 dan Tugas 2 dengan melakukan tugas-tugas di bawah ini!

a. Terjemahkan program di atas ke dalam bahasa assembly RISC-V (RV32I). Hanya gunakan instruksi-instruksi ADDI, ADD, dan BLT, kemudian lengkapi dengan label untuk percabangan.

```
# 3a Kode
.text
.globl main
main:
    addi x5, x0, 1      # i = 1
    addi x6, x0, 0      # sum = 0
    addi x7, x0, 11     # batas = 11
loop:
    add  x6, x6, x5     # sum = sum + i
    addi x5, x5, 1      # i = i + 1
    blt  x5, x7, loop   # jika i < 11, ulangi loop
done:
    nop                 # akhir program
```

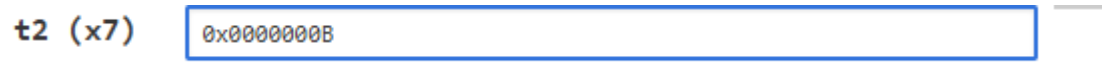
b. Lakukan verifikasi di Venus dengan menjalankan program hingga mencapai label akhir, lalu ambil satu screenshot saja yang sekaligus menampilkan panel Text (deretan instruksi) dan panel Registers pada kondisi akhir. Pada screenshot tersebut, tunjukkan dengan jelas bahwa nilai register yang Anda tetapkan sebagai i bernilai 10 dan register yang Anda tetapkan sebagai sum bernilai 55; beri penanda singkat (misalnya keterangan, panah, atau highlight) untuk menyebutkan register mana yang dipakai sebagai i dan sum. Ikuti arahan penghematan kertas dengan hanya menyertakan screenshot pada keadaan akhir eksekusi, tampilan Venus light mode, gambar harus jelas, dan bulak-balik.



addi x5, x0, 1

addi x6 x0 0

addi x7 x0 11 (Masukkan nilai 11 dalam register x7)



add x6 x6 x5 (Mulai inisiasi counter loop)

Loop 1

t1 (x6)	0x00000001

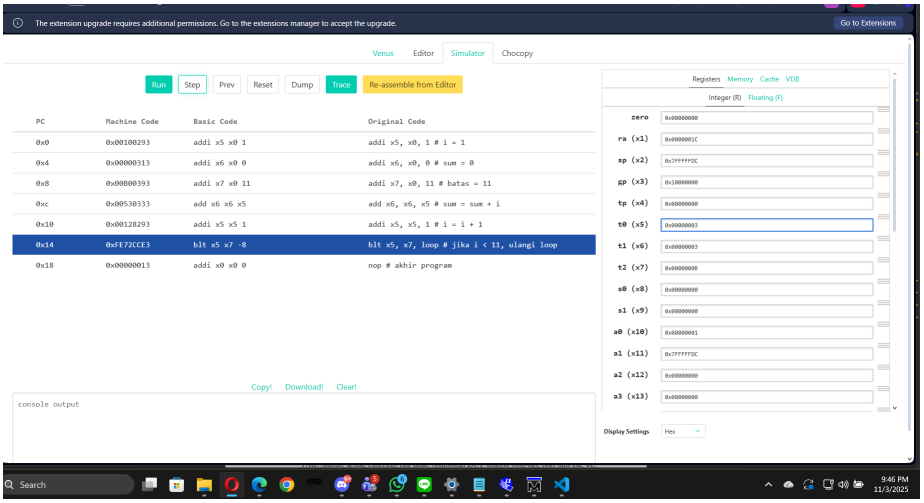
Loop 2

t0 (x5)	0x00000002

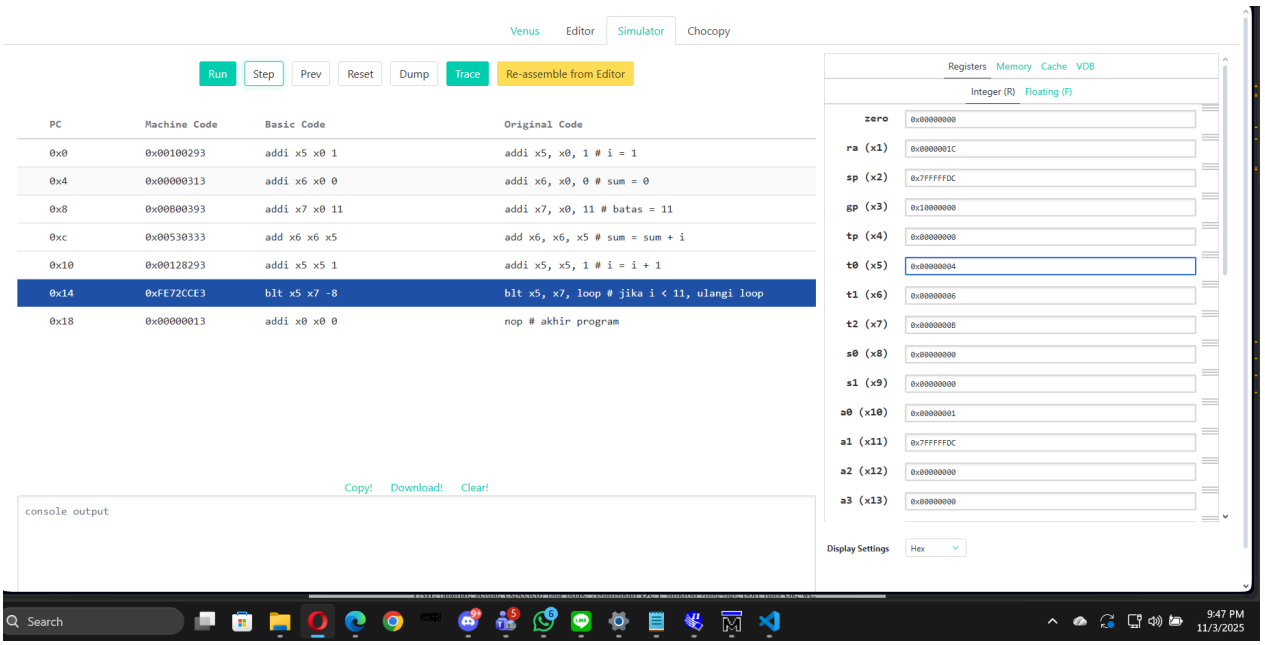
Loop 3

t0 (x5)	0x00000003

Dan seterusnya…

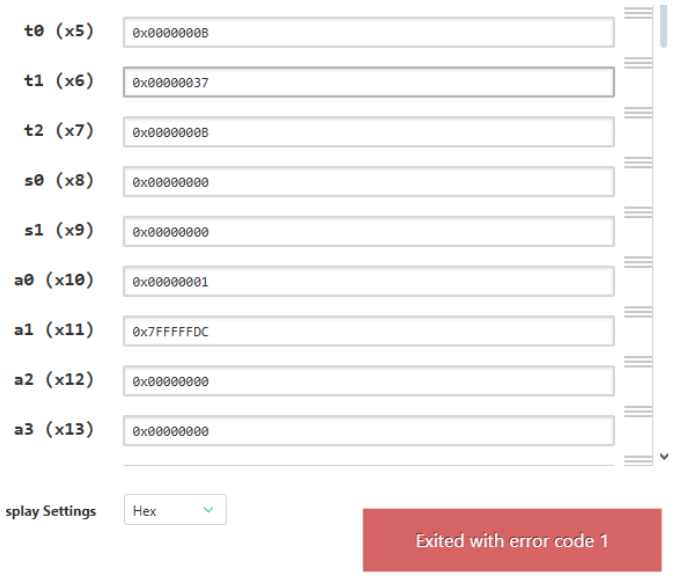


Loop 4



t0 (x5)	0x00000005
t1 (x6)	0x0000000A

Sehingga akan menjadi **t1 = 0x00000037** (32+16+4+2+1 = **55**) dan **t2 = 0x0000000B** (B = 11, 11 Iterasi). Bukan 10.



c. Catat hasil machine code (32-bit) dari setiap instruksi yang muncul pada panel Text Venus. (hint: gunakan opsi Dump untuk memberikan semua machine code yang dapat langsung disalin!)

```
0x00100293
0x00000313
0x00B00393
0x00530333
0x00128293
0xFE72CCE3
0x00000013
```

4. Buat testbench register file yang: (1) menulis nilai arbitrer bebas di 4 alamat berbeda (mis. x1, x2, x3, dan x4) dengan data yang mudah dicek (misalnya 0x11,0x22,0x33,0x44), satu siklus per penulisan; (2) setelah semua write selesai, membaca 10 alamat (mis. x0 – x9) termasuk 4 alamat tadi. Testbench self-checking: bandingkan hasil baca dengan expected, laporkan PASS jika semua cocok atau FAIL (alamat, actual, expected) bila tidak. Asumsikan DUT sinkron rising-edge, port tulis clk, we, waddr, wdata, port baca raddr*,rdata* (1 atau 2), data 32-bit, 32 register. Jika meniru RISC-V, verifikasi x0 selalu 0 dan mengabaikan write.

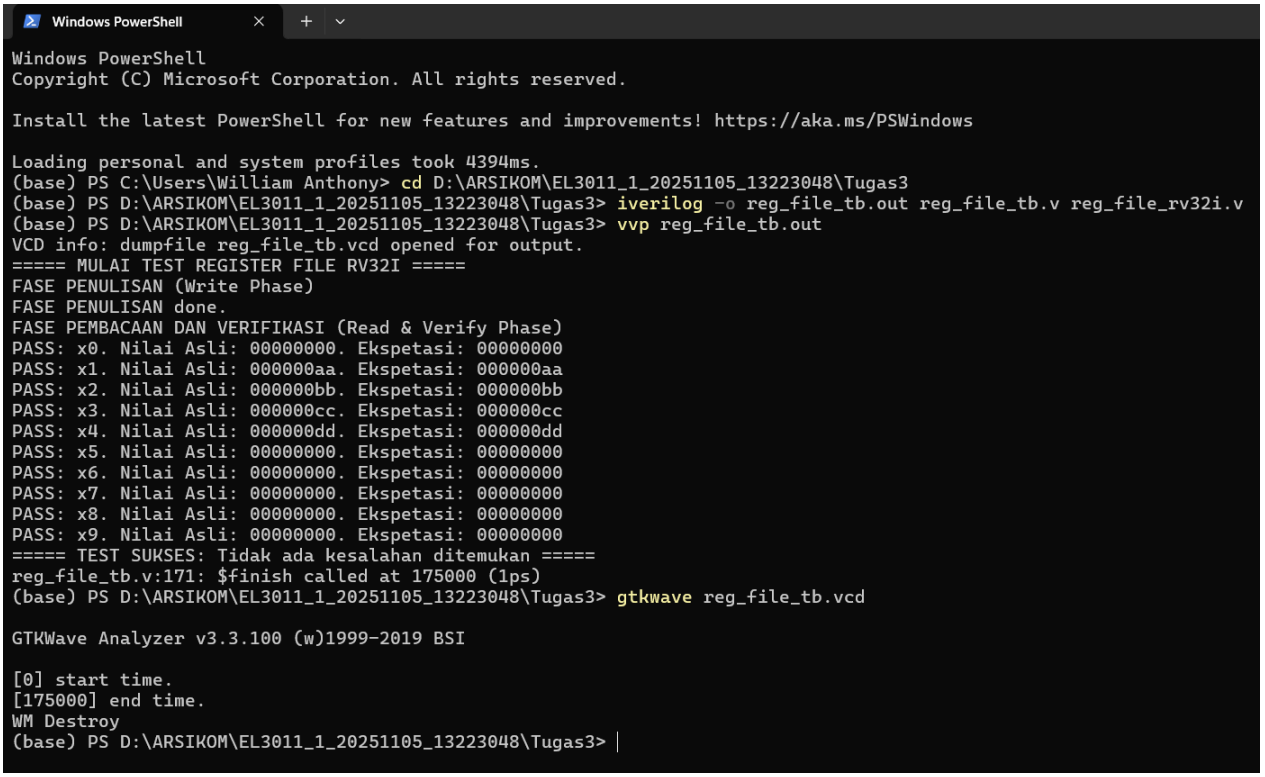
Kode testbench yang digunakan

```
// ============================================================================
// Praktikum EL3011 Arsitektur Sistem Komputer
// Modul        : 1
// Percobaan    : 1
// Tanggal      : 5 November 2025
// Kelompok     :
// Rombongan    : Rabu
// Nama (NIM) 1 : William Anthony (13223048)
// Nama (NIM) 2 : Darren Johan (13223032)
// Nama File    : reg_file_tb.v
// Deskripsi    : Testbench self-checking untuk reg_file_rv32i
//                Menulis 4 register (x1-x4) lalu membaca x0-x9
//                dengan sinkronisasi penuh rising edge-falling edge.
// ============================================================================


`timescale 1ns/1ps

module reg_file_tb;

    // Deklarasi sinyal-sinyal untuk DUT (Device Under Test)
```

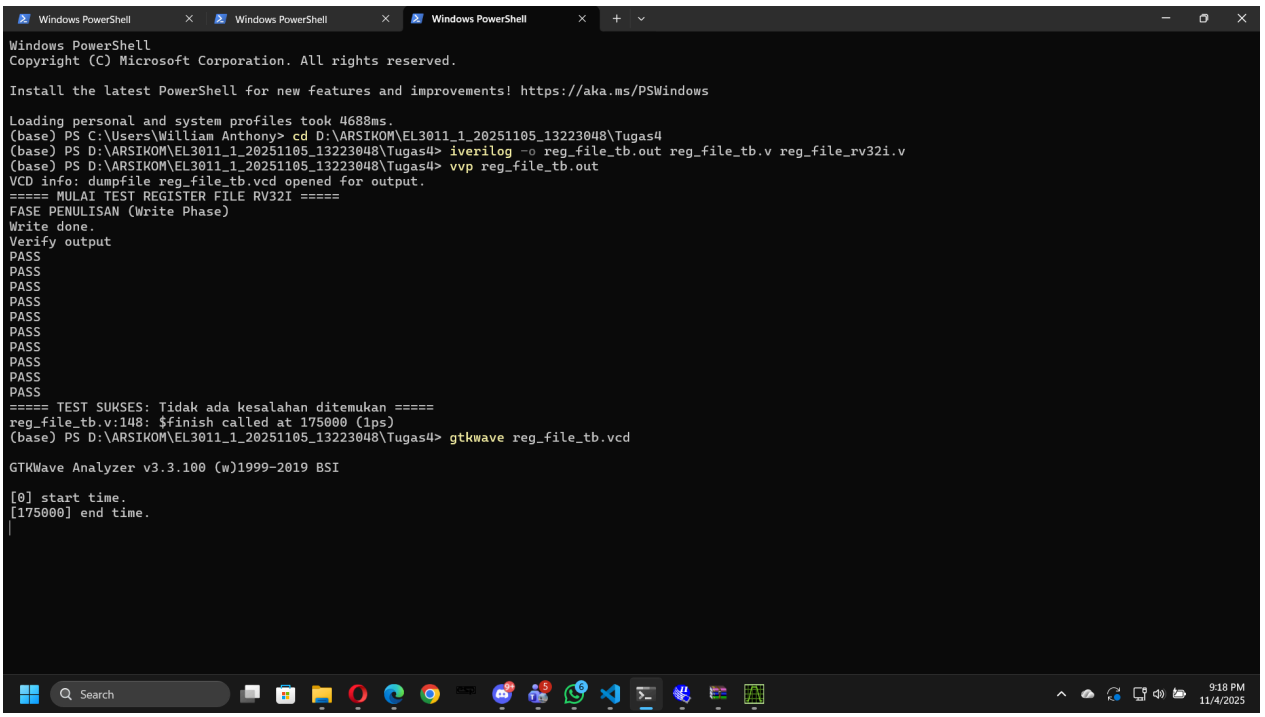
```
reg         clock;
reg         cu_rdwrite; // write enable
reg  [4:0]  rs1_addr;   // alamat baca port 1
reg  [4:0]  rs2_addr;   // alamat baca port 2
reg  [4:0]  rd_addr;    // alamat tulis
reg  [31:0] rd_in;      // data tulis
wire [31:0] rs1;        // data baca port 1
wire [31:0] rs2;        // data baca port 2

// Array untuk mempermudah perbandingan expected
reg [31:0] expected_rf [0:9];
integer errors;     // Deklarasi jumlah error di tingkat modul
integer k;          // Deklarasi variabel loop 'k' di tingkat modul
integer reg_idx;    // Deklarasi variabel loop 'reg_idx' di tingkat modul
reg [31:0] data_aktual; // Variabel untuk menyimpan data yang dibaca, deklarasi di tingkat modul

// Unit Under Test (UUT) - reg_file_rv32i
reg_file_rv32i uut (
    .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)
);

// Siapkan nilai diharapkan untuk register file agar bisa verifikasi.
initial begin
    expected_rf[0] = 32'h00000000; // x0 selalu 0
    expected_rf[1] = 32'h000000aa;
    expected_rf[2] = 32'h000000bb;
    expected_rf[3] = 32'h000000cc;
    expected_rf[4] = 32'h000000dd;
    // Register x5 sampai x9 tidak ditulis, jadi seharusnya 0
    for (k = 5; k <= 9; k = k + 1) begin // Gunakan int k yang sudah dideklarasikan di tingkat modul
        expected_rf[k] = 32'h00000000;
    end
end

// Clock generation
initial begin
    clock = 0;
    forever #5 clock = ~clock; // Clock period 10ns
end

// Test sequence utama
initial begin
    $dumpfile("reg_file_tb.vcd");
    $dumpvars(0, reg_file_tb);

    $display("===== MULAI TEST REGISTER FILE RV32I =====");

    errors = 0; // Inisialisasi errors di sini

    // Inisialisasi awal semua signal kontrol
    cu_rdwrite = 0;
    rd_addr    = 0;
    rd_in      = 0;
    rs1_addr   = 0; // Harus dimulai dari 0 untuk biar x0 diverif
    rs2_addr   = 0;

    #10; // Tunggu satu siklus clock awal agar sistem stabil (Kalau ga nanti aneh di perpindahan data)

    $display("FASE PENULISAN (Write Phase)");

    // Penulisan ke x1
    cu_rdwrite = 1; // Aktifkan write enable
    rd_addr    = 5'd1; // Alamat x1
    rd_in      = 32'h000000aa;
    #10;

    // Penulisan ke x2
    rd_addr    = 5'd2; // Alamat x2
    rd_in      = 32'h000000bb;
    #10;

    // Penulisan ke x3
    rd_addr    = 5'd3; // Alamat x3
    rd_in      = 32'h000000cc;
    #10;

    // Penulisan ke x4
    rd_addr    = 5'd4; // Alamat x4
    rd_in      = 32'h000000dd;
    #10;

    // Coba menulis ke x0 (seharusnya diabaikan)
    rd_addr    = 5'd0; // Alamat x0
    rd_in      = 32'hFFFFFFFF; // Masukkan datanya sembarang, seharusnya tidak berpengaruh samsek
                               ///ke x0
    #10;

    cu_rdwrite = 0; // Nonaktifkan write enable setelah semua penulisan
    $display("FASE PENULISAN done.");
    #10; // Tunggu sebentar setelah penulisan selesai, pastikan tidak ada efek samping

    $display("FASE PEMBACAAN DAN VERIFIKASI (Read & Verify Phase)");

    // ********************************************************************
    // PENYESUAIAN TIMING PEMBACAAN
    // ********************************************************************
    // (1) Set alamat baca awal ke x0
```

```
    /*
        Sinkronisasikan proses baca tulis yang terjadi pada register.
        Proses pembacaan data pada register berlangsung pada saat falling edge clock sedangkan
        penulisan data pada register berlangsung pada saat rising edge clock.
    */
        rs1_addr = 0;
        #5;

        // Menunggu sampai falling edge clock pertama dari fase pembacaan

        // (2) Membaca 10 alamat (x0 - x9) dan melakukan self-checking
        for (reg_idx = 0; reg_idx <= 9; reg_idx = reg_idx + 1) begin
            // Pada titik ini (tepat setelah #5, yaitu di falling edge), rs1_addr sudah diatur
            // dan rs1 seharusnya sudah update.
            data_aktual = rs1; // Ambil data aktual dari port baca

            if (data_aktual == expected_rf[reg_idx]) begin
                $display("PASS: x%0d. Nilai Asli: %h. Ekspetasi: %h", reg_idx, data_aktual, expected_rf[reg_idx]);
            end else begin
                $display("FAIL: x%0d. Nilai Asli: %h. Ekspetasi: %h", reg_idx, data_aktual, expected_rf[reg_idx]);
                errors = errors + 1;
            end

            // Siapkan pengaturan alamat baca untuk iterasi berikutnya (kalo bukan yang terakhir)
            // dan tunggu satu siklus penuh (rising edge + falling edge) untuk pembacaan berikutnya.
            // Ini akan membuat rs1_addr stabil sebelum falling edge
            // dan memberi waktu bagi rs1 untuk update.
            if (reg_idx < 9) begin
                rs1_addr = reg_idx + 1;
                #10; // Tunggu satu siklus clock penuh (rising edge + falling edge)
            end
        end


        // ********************************************************************
        // VALIDASI AKHIRRRRRR!

        if (errors == 0) begin
            $display("===== TEST SUKSES: Tidak ada kesalahan ditemukan =====");
        end else begin
            $display("===== TEST GAGAL: %0d kesalahan ditemukan =====", errors);
        end

        #10; // Beri waktu sedikit sebelum $finish
        $finish; // Mengakhiri simulasi
    end

endmodule

/*
cd D:\ARSIKOM\EL3011_1_20251105_13223048\Tugas3
iverilog -o reg_file_tb.out reg_file_tb.v reg_file_rv32i.v
vvp reg_file_tb.out
gtkwave reg_file_tb.vcd
*/
```

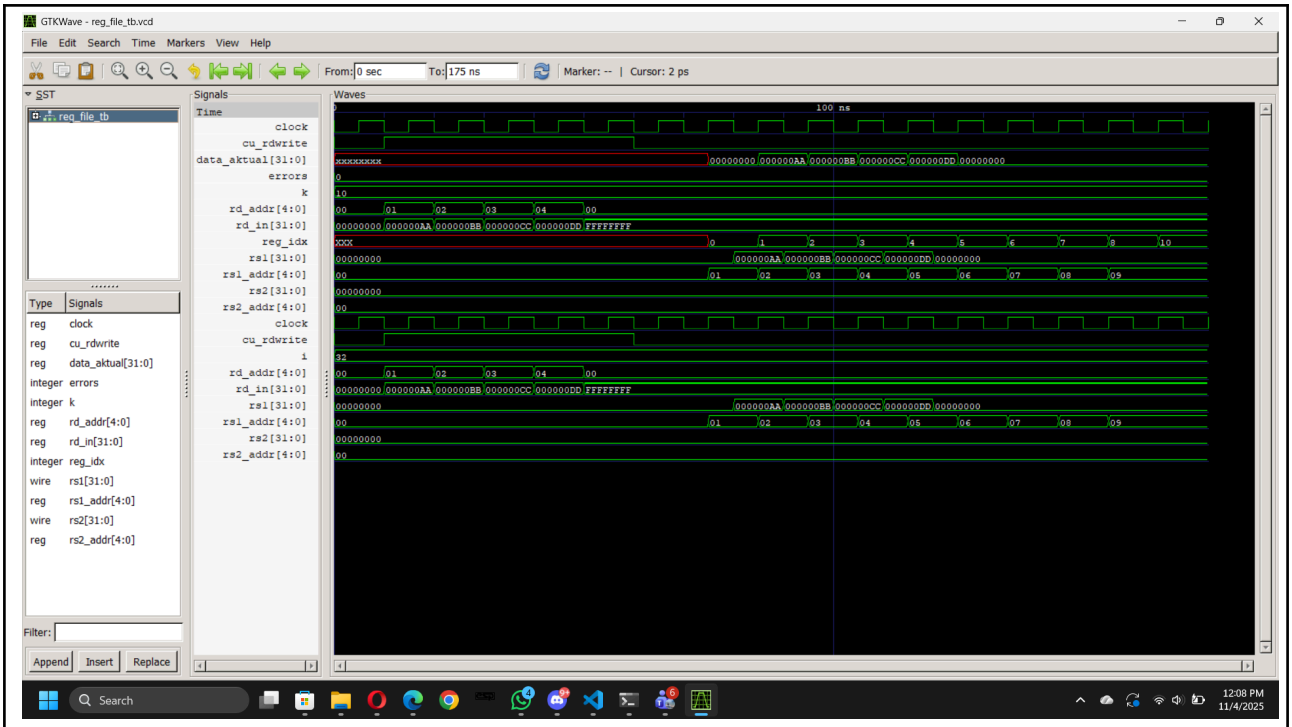
Bukti Berhasil



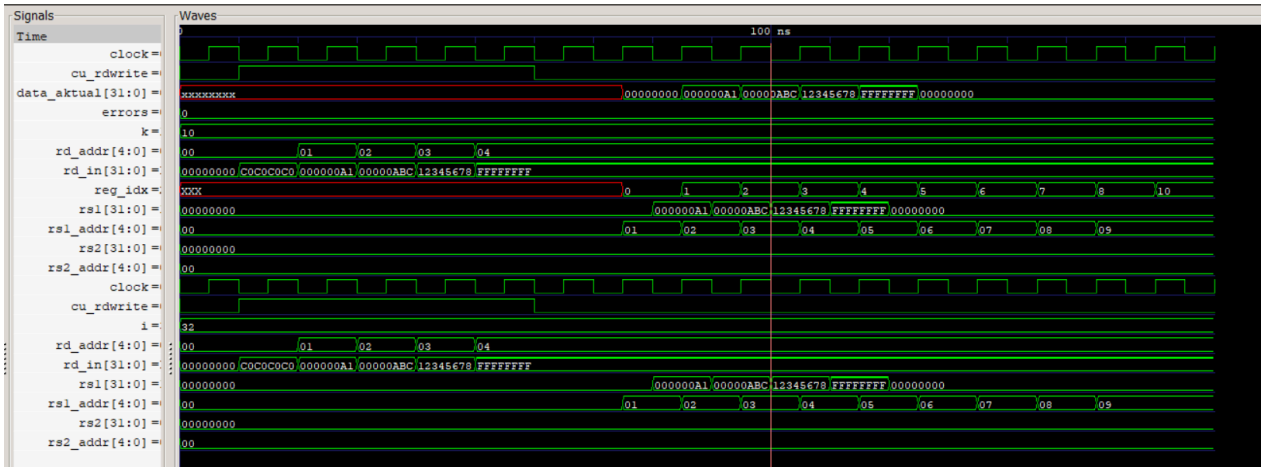
Versi 1 (Dengan komparasi Aktual secara langsung)



Versi 2 (Dipakai Praktikan)



Versi 1 (Dengan komparasi Aktual secara langsung)



Versi 2 (Dipakai Praktikan)

Dengan bukti-bukti diatas maka telah dikonfirmasi berhasil!

Berdasarkan output simulasi dan waveform GTKWave yang ditampilkan, dapat disimpulkan bahwa testbench berhasil dan module reg_file_rv32i berfungsi dengan benar sesuai spesifikasi. Pesan ***PASS*** untuk semua register x0 hingga x9 dalam output powershell tunjukkin nilai aktual yang dibaca dari register file sesuai dengan nilai yang diharapkan. Hal ini menunjukkan proses penulisan ke register x1–x4 telah berhasil, sementara register x0 serta x5–x9 tetap bernilai nol sebagaimana aturan pada arsitektur RISC-V.

Dengan pengamatan waveform hasil simulasi di GTKWave. Terlihat bahwa sinyal rs1[31:0] secara berurutan menampilkan nilai 00000000, 000000AA, 000000BB, 000000CC, dan 000000DD, diikuti oleh 00000000 untuk register x5 hingga x9. Pola ini identik dengan data yang ditulis pada fase penulisan dan sepenuhnya konsisten dengan hasil verifikasi *PASS* di konsol.

5. Jelaskan keunggulan dan trade-off penggunaan Altera®/Intel® ALTSYNCRAM dibandingkan memori "manual" berbasis array RTL pada desain instruction/data memory RV32I.

Aspek	ALTSYNCRAM (IP bawaan Quartus)	Array RTL Manual
Kelebihan	01. Menggunakan Block RAM internal FPGA (lebih cepat & hemat sumber daya). 02. Dapat diinisialisasi dengan file .mif atau .hex. 03. Lebih realistis terhadap implementasi FPGA sebenarnya.	01. Desain sederhana & mudah dipahami. 02. Tidak tergantung vendor (lebih portable). 03. Cocok untuk simulasi dan pembelajaran dasar.
Kekurangan	01. Tidak portable ke platform non-Intel/Altera. 02. Tidak transparan saat debugging RTL.	01. Tidak efisien (menggunakan banyak LUT/flip-flop). 02. Tidak bisa langsung load file .mif.
Trade-off	Untuk ALTSYNCRAM lebih efisien untuk implementasi nyata FPGA (kecepatan dan resource), sementara untuk array RTL lebih cocok untuk simulasi konsep karena mudah dimodifikasi dan dipahami.	

Daftar Referensi

Modul Praktikum EL3011 Arsitektur Sistem Komputer Edisi 2025 untuk digunakan pada Semester I Tahun Akademik 2025 / 2026. Stefen Sutandi, dkk. 2025. Ms Teams (Diakses 4 November 2025).

Venus. https://github.com/61c-teach/venus/wiki (Diakses 4 November 2025)

AMD Inc. Xilinx. *Choosing between Distributed RAM and Dedicated Block RAM*. UG901. 2018. https://docs.amd.com/r/en-US/ug901-vivado-synthesis/Choosing-Between-Distributed-RAM-and-Dedicated-Block-RAM. (Diakses 4 November 2025)

Teknologi pada FPGA (Field Programmable Gate Array). https://binus.ac.id/bandung/2019/12/teknologi-pada-fpga-field-programmable-gate-array (Diakses 4 November 2025)

Implementasi Sistem Kendali ON-OFF pada Field Programmable Gate Array (FPGA). https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&ved=2ahUKEwijo-bk39eQAxWq1TgGHV-SHX8QFnoECBgQAQ&url=https%3A%2F%2Fjtera.polteksmi.ac.id%2Findex.php%2Fjtera%2Farticle%2Fdownload%2F548%2F256&usg=AOvVaw3HRMjsHSvpioU64D1ru2xw&opi=89978449. (Diakses 4 November)

All About the RISC-V Processors. 2025. https://www.stromasys.com/resources/all-about-the-risc-v-processors (Diakses 4 November 2024)