Percobaan I
Synthesizable RISC-V (RV32I) Microprocessor Bagian I: Instruction Set, Register, dan Memory

Muhammad Falih Rosyidi (13223095)
Asisten : Michael Reynaldi Tamara (13222055)
Tanggal Percobaan : 06/11/2025
EL3011 Praktikum Arsitektur Sistem Komputer
Laboratorium Sinyal dan Sistem – Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung

***Abstrak*—These instructions give you guidelines for preparing papers for IEEE Transactions and Journals. Use this document as a template if you are using Microsoft *Word* 6.0 or later. Otherwise, use this document as an instruction set. The electronic file of your paper will be formatted further at IEEE. Paper titles should be written in uppercase and lowercase letters, not all uppercase. Avoid writing long formulas with subscripts in the title; short formulas that identify the elements are fine (e.g., "Nd–Fe–B"). Do not write "(Invited)" in the title. Full names of authors are preferred in the author field, but are not required. Put a space between authors' initials. Define all symbols used in the abstract. Do not cite references in the abstract. Do not delete the blank line immediately above the abstract; it sets the footnote at the bottom of this column.**

***Kata Kunci*—Enter key words or phrases in alphabetical order, separated by commas.**

I. PENDAHULUAN

Percobaan pada modul 1 kali ini akan menjalankan program-program untuk mengukur kinerja instruction memory baik dengan memori manual maupun ALTSYNCRAM, mengukur kinreja data memori dan Register, serta mengukur hasil instruction set. Percobaan 1 adalah mengukur kinerja fungsional dan timing dari penggunaan memori manual pada data instruction set yang sudah dibuat. Percobaan 2 adalah mengukur kinerja penggunaan ALTSYNCRAM sebagai instruksi memori. Percobaan 3 adalah mengukur kinerja ALTSYNCRAM sebagai data memory. Dan percobaan 4 adalah mengukur kinerja Register yang ada pada RV32I. Semua percobaan yang dilakukan pada modul ini memiliki beberapa tujuan yakni:

1. Praktikan memahami arsitektur mikroprosesor RISC-V (RV32I) beserta datapath eksekusinya (single-cycle untuk implementasi HDL praktikum).
2. Praktikan memahami instruction set inti RV32I dan mampu menulis program assembly sederhana untuk dieksekusi pada simulator/assembler RISC-V.
3. Praktikan dapat melakukan simulasi eksekusi program RISC-V pada simulator/assembler RISC-V dan memahami cara setiap instruksi dieksekusi.
4. Praktikan dapat membuat instruction memory, data memory, dan register file RISC-V dalam kode Verilog yang dapat disintesis dan disimulasikan dengan Intel® Quartus® Prime dan ModelSim.

II. LANDASAN TEORETIS

A. *Verilog HDL, Venus, dan Altera® Quartus II*

Verilog adalah bahasa pemrograman Hardware Description Language (HDL) yang berfungsi untuk merealisasikan sirkuit digital dengan kode [1]. Verilog biasanya digunakan untuk desain RTL dan verifikasi desain baik pada FPGA maupun pada ASICs.

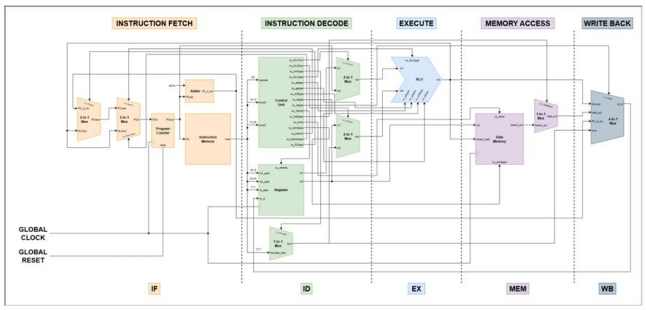
Venus adalah simulator instruksi set RISC-V yang dibangun untuk tujuan pembelajaran [2]. Venus akan mengubah kode-kode assembly kita menjadi machine code sesuai dengan format instruksi set yang ada pada RISC-V, sehingga memudahkan kita karena tidak perlu lagi mengubahnya menjadi machine code setiap instruksi.

Altera Quartus II adalah aplikasi lingkungan terintegrasi kuat yang dirancang untuk menyeseuaikan berbagai FPGA [3]. Quartus mendukung setiap tahap pengembangan baik itu design, sintesis, optimisasi, verifikasi, simulation dan analisis [3]. Sehingga Altera Quartus menjadi aplikasi yang berguna dalam menguji program atau desain HDL kita dalam hal analisis fungsional dan timing.

B. *Microprocessor RISC-V (RV32I)*

RISC-V adalah sebuah Instruction Set Architecture (ISA) yang open-source untuk semua orang [4]. RISC-V singkatan dari Reduces Instruction Set Computing sedangkan V berarti edisi ke lima dari RISC [4]. Terdapat berbagai tipe RISC-V tetapi yang digunakan pada praktikum ini adalah RV32I yakni RISC-V dengan base 32 bit. Karena ISA-nya yang simple dan mudah, hal ini membuat RISC-V banyak dikembangkan baik dalam penelitian, development atau bahkan dalam SoC. RISC-

V terdiri dari Lima tahap pada eksekusi instruksi.



Gambar 1 Datapath 5 Tahap Eksekusi Instruksi pada RISC-V Single Cycle [5]

Lima tahap ini terdiri dari:

1. Instruction Fetch (IF):
   Tujuan utama tahap instruksi fetch adalah untuk mengambil instruksi berikutnya dari memori sehingga dapat didekode dan dieksekusi oleh prosesor [6].
2. Instruction Decode (ID):
   Tujuan utama tahap ini adalah untuk menginterpretasikan instruksi yang diambil dan menyiapkan masukan yang diperlukan (register, sinyal kontrol) untuk tahap-tahap berikutnya [6].
3. Exceute/Addres Calculation (EX):
   Peran utamanya adalah melaksanakan operasi aritmatika atau logika yang ditentukan oleh instruksi, menghitung alamat memori untuk operasi load/store, atau menentukan hasil dari cabang [6].
4. Memory Access (MEM):
   Tahap ini bertanggung jawab untuk berinteraksi dengan memori data selama instruksi load atau store. Jika instruksi tersebut bukan operasi memori, tahap ini dilewati, dan prosesor beralih ke tahap writeback [6].
5. Write Back (WB):
   Tujuan utama tahap ini adalah untuk menulis hasil dari suatu instruksi (baik itu dari operasi aritmatika atau pemuatan memori) kembali ke register tujuan [6].

C. *Instruction Set dan Register RV32I*

RV32I memiliki 32 buah register didalamnya yang setiap register tersebut memiliki panjang 32 bit.

Register	ABI Name	Description	Saver
x0	zero	Zero constant	—
x1	ra	Return address	Callee
x2	sp	Stack pointer	Callee
x3	gp	Global pointer	—
x4	tp	Thread pointer	—
x5-x7	t0-t2	Temporaries	Caller
x8	s0 / fp	Saved / frame pointer	Callee
x9	s1	Saved register	Callee
x10-x11	a0-a1	Fn args/return values	Caller
x12-x17	a2-a7	Fn args	Caller
x18-x27	s2-s11	Saved registers	Callee
x28-x31	t3-t6	Temporaries	Caller
f0-7	ft0-7	FP temporaries	Caller
f8-9	fs0-1	FP saved registers	Callee
f10-11	fa0-1	FP args/return values	Caller
f12-17	fa2-7	FP args	Caller
f18-27	fs2-11	FP saved registers	Callee
f28-31	ft8-11	FP temporaries	Caller

Gambar 2 RISC-V List Register [7]

Selain itu, pada RISCV terdapat 6 jenis format instruksi yakni R, I, S, B, U, dan J.

31 27	26 25	24 20	19 15	14 12	11 7	6 0	
funct7		rs2	rs1	funct3	rd	opcode	R-type
imm[11:0]			rs1	funct3	rd	opcode	I-type
imm[11:5]		rs2	rs1	funct3	imm[4:0]	opcode	S-type
imm[12|10:5]		rs2	rs1	funct3	imm[4:1|11]	opcode	B-type
imm[31:12]					rd	opcode	U-type
imm[20|10:1|11|19:12]					rd	opcode	J-type

Gambar 3 Format Instruksi pada RISC-V [7]

Untuk keterangan masing-masing komponen pada format tersebut adalah berikut.

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).

Gambar 4 Deskripsi Komponen pada Format Instruksi [5]

Perbedaan tiap format instruksi itu karena tiap format memiliki fungsi yang berbeda-beda. Berikut penjelasan tiap formatnya.

1. R type: adalah instruksi yang melakukan operasi dari semua data yang sudah ada pada register
2. I type: adalah instruksi yang melakukan operasi yang tidak hanya data dari register tetapi ada data dari konstanta/immediate
3. S type: adalah instruksi yang menyimpan data pada data memori
4. B type: adalah instruksi yang melompati urutan instruksi selanjutnya ketika suatu kondisi terpenuhi biasanya pada if, atau loop
5. U type: adalah instruksi yang menyimpan nilai immediate ke dalam register.
6. J type: adalah instruksi yang melompati instruksi urutan selanjutnya tanpa kondisi apapun, biasanya untuk memanggin fungsi

D. *Altera® MegaFunction ALTSYNCRAM*

Altera menyediakan beragam mode RAM untuk memenuhi kebutuhan memori desain system-on-a-programmable-chip (SOPC) masa kini. Altera menyediakan mode RAM melalui megafungsi memori yang dapat diparameterisasi dan dioptimalkan untuk arsitektur perangkat Altera®. Menggunakan megafungsi, alih-alih mengodekan logika Anda sendiri, menghemat waktu desain yang berharga dan menawarkan sintesis logika serta implementasi perangkat yang lebih efisien. Ada beberapa jenis megafungsi memori, termasuk megafungsi ALTSYNCRAM, ALTDPRAM, ALT3PRAM, dan LPM_RAM_DQ. Perangkat lunak Quartus® II secara otomatis memilih salah satu megafungsi untuk mengimplementasikan fungsi RAM. Pemilihan tersebut bergantung pada perangkat target, mode RAM, dan fitur fungsi RAM [8].

Sehinga karena sudah terdapat memori dedicated pada FPGA yang ada, menggunakan ALTSYNCRAM tidak aka membuat LUT yang ada untuk design memori saja. Akan tetapi, ini keefektidan ALTSYNCRAM akan sangat terasa jika data-nya cukup besar karena lebih efektif dengan desain yang terpisah tanpa menggunakan LUT (seperti pada memori manual).

III. Hasil dan Analisis

A. Tugas I : Perancangan Instruction Memory (RV32I

Pada tugas 1 ini kita mengaplikasikan instruction set yang sudah kita dapatkan setelah melakukan simulasi RISC-V program penjumlahan nilai ‘i’ pada looping while ‘i’ <= 10. Didapat machine code dan output data pada lampiran. Dapat dilihat bahwasanya hasil machine code yang didapat sudah sesuai dengan RISC-V REFERENCE CARD pada referensi nomor [7]. Setiap kode memiliki penulisan yang berbeda sesuai dengan formatnya dan datanya. Sehingga nilai opcode yang berbeda untuk beda format instruksi, nilai funct3/funct7 yang berbeda juga tergantung instruksinya. Begitu pula untuk rd/rs1/rs2 dan immediate yang berbeda nilainya untuk tiap data berdasarkan format instruksinya. Cara mendecode assembly ke machine code ini yakni dengan mengecek format dari instruksi pada line kode saat itu. Kemudian kita ubah nilai register ke binary dan nilai immediate ke binary (jika ada immediate), ditambah juga komponen lainnya yang binary-nya berdasarkan RISC-V REFERENCE CARD. Sehingga kita tulis bersama, berdasarkan urutan pada format instruksinya lalu ubah Kembali ke HEX, dengan begitu kita mendapati machine code untuk kode tersebut, salah satu contohnya ada pada line 1 dan line 3 yang memiliki instruksi yang sama yakni addi sehingga machine code-nya pun sangat mirip seperti pada di lampiran.

Bisa dilihat pada landasan teori serta pada kode untuk tugas 1 ini kita membuat instruction memory. Karena instruction memory dan data memory terpisah sehingga RISC-V RV32I yang dibuat adalah single cycle atau semua operasi berjalan bersamaan dalam satu siklus clock (IF – ID – EX – MEM – WB). Tiap tahap tersebut bekerja bersamaan/paralel dalma single cycle, dimulai dari IF yang mengeluarkan instruction code ke pada ID, kemudian ID yang mendapatkan instruction code dan mengolah decodenya menjadi signal-signal instruksi pada komponen lain, lalu di jalankan apada tahap EX di ALU, dan kemudian di tulis kembali ke memori dan di tulis lagi datanya pada register destinasi di WB. Jika seperti itu tahap dalam single cycle, RISC-V generik pada umumnya yang sudah ter-pipelined dalam 5 stage yakni stage IF - ID – EX – MEM – WB yang terpisah sehingga proses tiap stagenya berbeda 1 clock, misal jika IF sedang mengambil instruksi saat ini, maka ID sedang mengolah instruksi yang dikasih sebelumnya. Oleh karena itu, latency atau siklus clock yang dibutuhkan tiap data semakinya banyak. Akan tetapi karena critical path delay dalam 1 siklus nyam akin kecil membuat clock yang bisa dipakai lebih besar sehingga throughput-nya makin besar.

Pada hasil percobaan fungsional dan timing dalam tugas 1 ini didapati hasil terlihat pada gambar yang ada di lampitan. Berdasarkan timing analisis didapat data delay dengan periode clock 10ns sebagai berikut.

no	start(ns)	delay(ns)	std	avg
1	5	6.243	0.3877874049	6.22375
2	15	6.2		
3	25	5.658		
4	35	5.746		
5	45	6.309		
6	55	6.65		
7	65	6.2		
8	75	6.784		

Dapat dilihat bahwa rata-rata delay pada periode clock 10 ns adalah 6.22 ns hal ini menunjukkan bahwa perlu sekitat 6.22 ns untuk instruksi memori manual mengeluarkan data yang ada di dalamnya. Selain itu, dapat dilihat juga pada hasil simulasi fungsional dan timing di lampiran bahwa pembacaan datanya sudah sesuai denga nisi yang ada. NOP pada memori manual yakni addi x0, x0, 0 (0x00000013) juga ada pada hasil simulasi sehingga memori manual sudah bekerja dengan baik.

B. Tugas 2 : Perancangan Instruction Memory dengan Altera® MegaFunction ALTSYNCRAM

Seperti pada tugas 1, tugas 2 juga mengecek fungsional dan timing daring instruction memory di RV32I. Diperlukannya instruction memory yang terpisah dengan data memory yakni karena RISC-V berarsitektur Harvard, sehingga itu memudahkan dalam merancang microprocessor RISC-V. Pada kode ini juga diperlukan adanya file .mif karena untuk setup data pada ALTSYNCRAM. Tanpa file .mif maka tidak ada data yang ada di dalam ALTSYNCRAM sehingga yang dikeluarkan selalu 0. Sehingga ini adalah salah satu kelebihan ALTSYNCRAM disbanding memori manual pada tugas 1 yakni memudahkan kita menginisialisasi data dengan file .mif, jadi tidak perlu mengubah kode Verilog-nya.

Hasil yang didapat pada tugas ini berupa gambar pengujian fungsional dan timing yang terdapat pada lampiran. Dari pengujian timing didapat data delay-nya yakni dalam tabel berikut.

no	start(ns)	delay(ns)	std	avg
1	5	9.292	0.5776165312	9.013
2	15	8.348		
3	25	8.322		
4	35	9.576		
5	45	8.332		
6	55	9.295		
7	65	9.644		
8	75	9.295		

Didapati bahwa rata-rata delay di sekitar 9 ns. Hal ini menunjukkan bahwa diperlukan sekitar 9 ns untuk ALTSYNCRAM mengeluarkan data yang ada didalamnya. Hasil simulasi fungsional dan timing menunjukkan kecocokan data yang ada pada imemory.mif dengan hasil keluarannya. Selain itu, karena menggunakan ALTSYNCRAM maka default

untuk memorinya adalah 0 dan sudah seseuai dengan hasil simulasinya. Hal ini membuktikan bahwa instruction memory dengan ALTSYNCRAM sudah dapat bekerja dengan baik.

Walaupun begitu dapat dilihat bahwa delay pada ALTSYNCRAM lebih lama disbandingkan pada memori manual. Hal ini dapat terjadi karena pada kode ALTSYNCRAM menggunakan tipe UNREGISTERED yang membuat ALTSYNCRAM bekerja secara kombinasional. Sedangkan pada memori manual pembacan sinkron saat posedge clock. Hal ini membuat rangkaian memori manual lebih cepat karena delay clock to outputnya yang kecil. Pada AlTSYNCRAM menggunakan asinkron data sehingga perubahan input maka langsung akan mengeluarkan datanya, hal ini membuat rangakaian full kombinasional walau ada clocknya, sehingga tidak hanya perlu waktu untuk mengolah data tapi juga perlu waktu untuk mengakses RAM dari ALTSYNCRAM itu sendiri, hal ini lah yang membuat delay pada ALTSYNCRAM lebih lama. Selain itu penggunaan file .mif juga menambah delay karena perlu waktu untuk sistem mebaca dan menstabilkan data dari file .mif. Akan tetapi, untuk sistem yang lebih bsar penggunaan ALTSYNCRAM jauh lebih efisien (jika masih dalam implementasi FPGA) karena FPGA sudah mempunyai block RAM dedicated sendiri untuk ALTYNCRAM sehingga tidak banyak LUT yang terbuang sia-sia untuk memori saja.

C. Tugas 3 : Perancangan Data Memory dengan Altera® MegaFunction ALTSYNCRAM

Pada tugas 3 ini kita memakai ALTSYNCRAM Sebagai data memory. Dilakukan pengujian fungsional dan timing untuk data memory ini, pengujian dilakukan 2 tipe yakni membaca dememory.mif dan menulis data baru di baca juga. Hasil delay dari timing analisis ada pada tabel berikut.

timing tugas 3 keadaan 1 (pakai .mif)				
no	start(ns)	delay(ns)	std	avg
1	10	9.151	0.2864431819	9.071625
2	20	9.156		
3	30	9.151		
4	40	8.803		
5	50	8.491		
6	60	9.3		
7	70	9.156		
8	80	9.365		

Didapat delay rata-rata untuk mebaca data pada file .mif adalah 9.07 ns. Kemudian untuk delay dengan write dan read adalah berikut.

timing tugas 3 (pakai write dan read)				
no	start(ns)	delay(ns)	std	avg
1	10	7.463	0.3431525067	7.25975
2	20	6.902		
3	30	7.566		
4	40	7.463		
5	50	6.753		
6	60	7.566		
7	70	7		
8	80	6.902		

Didapati bahwa rata-rata delay untuk read dan write data memory adalah7.26 ns. Dari data tersebut kita dapati bahwa delay-nya tidak sama antara penggunaan file .mif dan dengan write kemudian read. Hal ini berbeda karena perlu ada waktu untuk sistem menstabilkan data yang dibaca pada file .mif atau cold start sedangkan pada kasus write kemudian read, sistem sudah siap karena sudah menjalankan write terlebih dahulu ataun warm. Walaupun berbeda delaynya, data yang berhasil dikeluarkan atau diuji sudah sesuai dimana pada kasus file .mif sudah berhasil mengeluarkan data pada file, serta pada kasus write kemudian read data random yang di write dan di read sama nilainya. Walau pada kedua kasus alamat ke 0 tidak terbaca secara fungsional karena Data Memory ini mengeluarkan saat negedge atau falling edge dari clock sedangkan sejak awal clock bernilai 0 dan ketika falling edge/negedge selanjutnya sudah masuk ke alamat berikutnya.

D. Tugas 4 : Perancangan Register

Pada tugas 4 ini menguji komponen register yang ada pada RV32I yakni 32 buah register dengan 32bit lebarnya masing-masing. Berikut adalah tabel delay timing analisis pada register.

no	start(ns)	delay(ns)	std	avg
1	70	7.523	0.4206100162	7.36175
2	110	6.407		
3	150	7.621		
4	190	7.1		
5	230	7.523		
6	270	7.576		
7	310	7.621		
8	350	7.523		

Didapati bahwa delay untuk membaca pada register ada pada 7.36 ns. Sehingga perlu delay sebesar 7.36 ns untuk mengeluarkan data yang ada register. Selain itu, didapati bahwa pada clock yang kecil yakni 10 ns – 30 ns, tidak semua data sesuai dengan yang diinginkan, sehingga ada banyak ketidak sesuain antara output simulasi dengan idealnya. Ketika mengunakan clock 40 ns barulah data yang di dapatkan sesuai dengan ideal outputnya seperti pada tabel dan gambar di lampiran. Hal ini dapat terjadi karena register ini memiliki banyak loop atau wire addres yang berbeda walau block

registernya sebenarnya sama, hal ini menyebabkan looping dimana rd harus menulis pada register tersebut sedangkan rs harus membaca pada register tersebut. Sehingga agar tidak bentrok dipakai lah posedge untuk menulis dan negedge untuk membaca. Akan tetapi, hal ini membatasi critical path delay menjadi setengah clock sehingga jika clock 10 ns maka proses menulis dengan rd harus selesai dan stabil kurang dari 5ns agar ketika negedge data yang dibaca rs1/rs2 sudah aman. Karena banyaknya wire addres dan jumlah register 32 sehingga banyak rangkaian mux yang terpakai pada komponen ini, hal ini menyebabkan delay yang cukup besar unutk mengakses dan menulis data. Oleh karena itu, sistem dapat bekerja dengan baik saat clock 40 ns. Data yang didapat pun sudah sesuai yakni saat negedge data yang dikeluarkan adalah data yang baru diinput saat posedge. Kecuali pada kasus khusus yakni x0 karena x0 harus selalu ground. Cara memastikannya yakni dengan selalu menset x0 = 0 saat ada posedge clk atau tiap membaca ini adalah implementasi pada kode walau kenayataanya x0 inputnya akan selalu terhubung dengan ground.

Dari komponen register pada tugas 4 dan data memory pada tugas 3 kita dapati beberapa perbedaan fungsional dan timingnya. Didapati bahwa latensi pada register lebih cepat dibandingkan pada tugas 3 jika memakai file .mif tetapi sama jika write dan read. Selain itu, clock pada data memori lebih kecil periodenya karena read nya asynchronous sehingga tidak bergantung pada clock sedangkan pada register write dan read syncornous dengan clock sehingga delay untuk stabil write dan read harus lebih cepat dibandingkan setengan dari periode clock. Walau periode clock nya harus lebih lama tetapi penerapan disiplin clock posedge dan negedege yang berbeda fungsi ini sangat penting karena memastikan agar fungsi yang berbeda tidak bekerja bersamaan apda register yang sama, yang mana akan memunculkan error karena jika hanya pada posedge maka mungkin saja sistem sedang menulis pada register x6 sambil sistem membaca pada register x6. Hal itu akan membuat data salah karena write belum selesai dengan stabil tetapi sudah dibaca oleh sistem.

IV. SIMPULAN

Berdasarkan hasil percobaan yang telah dilakukan pada modul 1 ini, dapat disimpulkan beberapa hal sebagai berikut:

- Arsitektur RISC-V (RV32I) menggunakan datapath single-cycle yang terdiri dari 5 tahap eksekusi (IF-ID-EX-MEM-WB) yang bekerja secara paralel dalam satu siklus clock. Perbedaan fundamental dengan pipeline adalah pada single-cycle semua tahap bekerja bersamaan, sedangkan pada pipeline setiap tahap terpisah dengan latency lebih besar namun throughput lebih tinggi.
- Instruction Set RV32I memiliki 6 format instruksi (R, I, S, B, U, J) yang masing-masing memiliki struktur berbeda sesuai fungsinya. Proses decode dari assembly ke machine code bergantung pada format instruksi, dimana komponen opcode, funct3/funct7, rd/rs1/rs2, dan immediate dikombinasikan sesuai urutan format kemudian dikonversi ke hexadecimal. Program assembly sederhana untuk penjumlahan iteratif (sum dari i pada loop $i \leq 10$) berhasil dikonversi dan dieksekusi dengan benar.
- Simulasi fungsional dan timing analysis menggunakan Quartus II dan ModelSim menunjukkan bahwa program RISC-V dapat dieksekusi sesuai ekspektasi. Hasil simulasi memverifikasi bahwa setiap instruksi berjalan dengan benar dan output sesuai dengan nilai yang diharapkan.
- Instruction Memory dengan memori manual memiliki delay rata-rata 6.22 ns dengan pembacaan sinkron di posedge clock. Implementasi ini lebih cepat karena menggunakan output register dengan clock-to-output delay yang kecil (~1-2 ns).
- Instruction Memory dengan ALTSYNCRAM (UNREGISTERED) memiliki delay rata-rata 9.01 ns, lebih lambat karena bersifat kombinasional (async read) dengan access time RAM yang lebih besar (~3-5 ns) ditambah overhead initialization dari file .mif. Keuntungan ALTSYNCRAM adalah efisiensi area pada desain besar karena menggunakan dedicated RAM blocks tanpa mengkonsumsi LUT.
- Data Memory menggunakan ALTSYNCRAM dengan karakteristik write di negedge (inverted clock) dan read asynchronous menunjukkan perbedaan delay antara pembacaan data dari file .mif (9.07 ns - cold start) dan setelah operasi write-read (7.26 ns - warm state). Perbedaan ini disebabkan oleh memory initialization state dimana cold start memerlukan waktu tambahan untuk setup, dan stabilisasi data, sedangkan warm state sudah dalam kondisi siap operasi.
- Register file 32×32-bit dengan 2 read ports dan 1 write port menggunakan dual-edge clocking (write @posedge, read @negedge) memerlukan periode clock minimum 40 ns (25 MHz) karena adanya feedback loop (rd → rf → rs → ALU → rd) dan half-cycle timing constraint. Keuntungan dual-edge adalah eliminasi data hazard secara otomatis, sehingga desain lebih sederhana meskipun clock frequency lebih rendah. Delay pembacaan register rata-rata adalah 7.36 ns, dengan register x0 dijaga selalu bernilai 0 sesuai spesifikasi RV32I.
- Perbandingan karakteristik timing menunjukkan trade-off antara performance, kompleksitas, dan efisiensi area: Instruction Memory Manual (Fmax ~100-150 MHz, delay 6.22 ns - tercepat), Instruction Memory ALTSYNCRAM (Fmax ~50-80 MHz, delay 9.01 ns), Data Memory ALTSYNCRAM (Fmax ~40-70 MHz, delay 7.26-9.07 ns), dan Register File (Fmax ~25 MHz, delay 7.36 ns - bottleneck karena dual-edge constraint). ALTSYNCRAM lebih lambat namun efisien untuk desain besar, sedangkan memori manual lebih cepat namun konsumsi LUT tinggi. Register file dengan dual-edge clocking mengorbankan clock speed untuk kesederhanaan desain dan eliminasi hazard detection logic.

Referensi

[1] https://www.geeksforgeeks.org/electronics-engineering/getting-started-with-verilog/, diakses pada 7 November 2025, 16.45.

[2] https://github.com/kvakil/venus, diakses pada 7 November 2025, 16.50.

[3] https://www.altera.com/products/development-tools/quartus, diakses pada 7 November 2025, 17.00.

[4] https://riscv.org/blog/what-is-risc-v-and-why-is-it-important/, diakses pada 7 November 2025, 17.10.

[5] Bagus Hanindhito et al. *Petunjuk Praktikum EL3011 Arsitektur Komputer Edisi 2025*. 2025. Bandung: LDTE ITB

[6] https://github.com/Varunkumar0610/RISC-V-32I-5-stage-Pipeline-Core, diaskes pada 7 Novermber 2025, 17.22.

[7] *RISC-V REFERENCE CARD.* *https://www.cs.sfu.ca/~ashriram/Courses/CS295/assets/notebooks/RISCV/RISCV_CARD.pdf*, *diakses pada 17.40*

[8] *RAM Megafunction User Guide, https://cdrdv2-public.intel.com/654455/ug_ram1.pdf, diakses pada 18.00*

Lampiran

1. Source Code untuk tugas 1

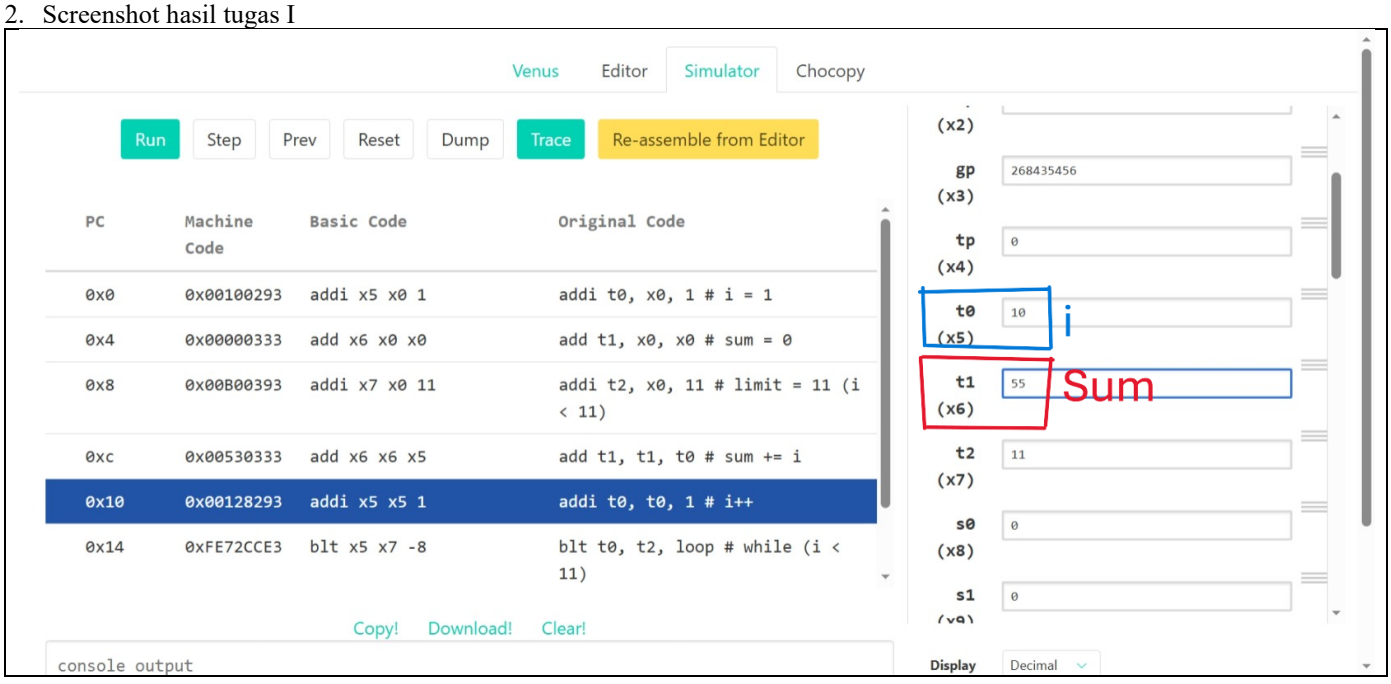
```
# Praktikum EL3011 Arsitektur Sistem Komputer
# Modul         : 1
# Percobaan     : 1 (Tugas Pendahuluan)
# Tanggal       : 4 November 2025
# Kelompok      :
# Rombongan     :
# Nama (NIM) 1 : Muammar Qadafi (13223094)
# Nama (NIM) 2 : Muhammad Falih Rosyidi (13223095)
# Nama File     : soal3.s
# Deskripsi     : Assembly dari tugas pendahuluan no 3 mengenai sum dari i pada looop
<= 10
.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
```

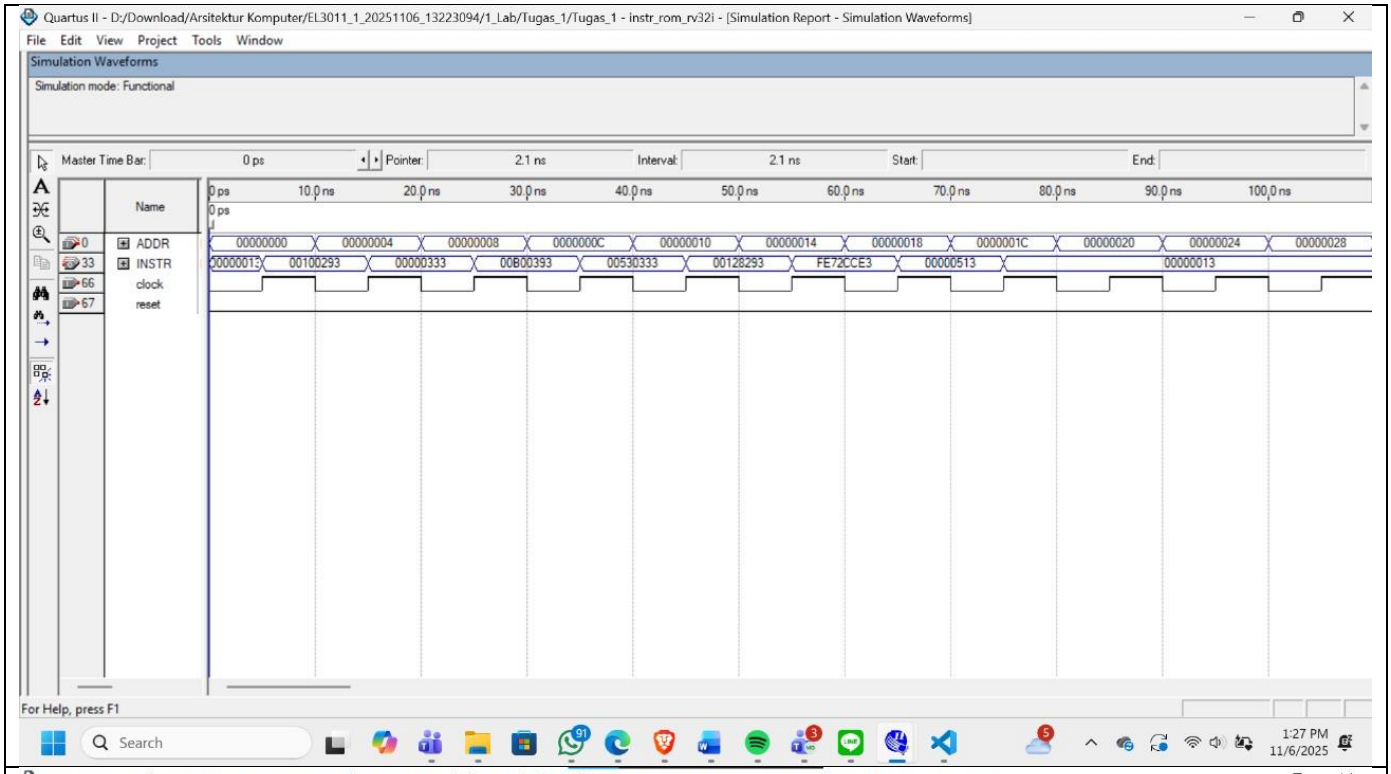
```
# Praktikum EL3011 Arsitektur Sistem Komputer
# Modul         : 1
# Percobaan     : 1 (Tugas Pendahuluan)
# Tanggal       : 4 November 2025
# Kelompok      :
# Rombongan     :
# Nama (NIM) 1 : Muammar Qadafi (13223094)
# Nama (NIM) 2 : Muhammad Falih Rosyidi (13223095)
# Nama File     : soal3.out
# Deskripsi     : Machine Code dari Assembly kode tugas pendahuluan no 3 mengenai sum
dari i pada looop  <= 10

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

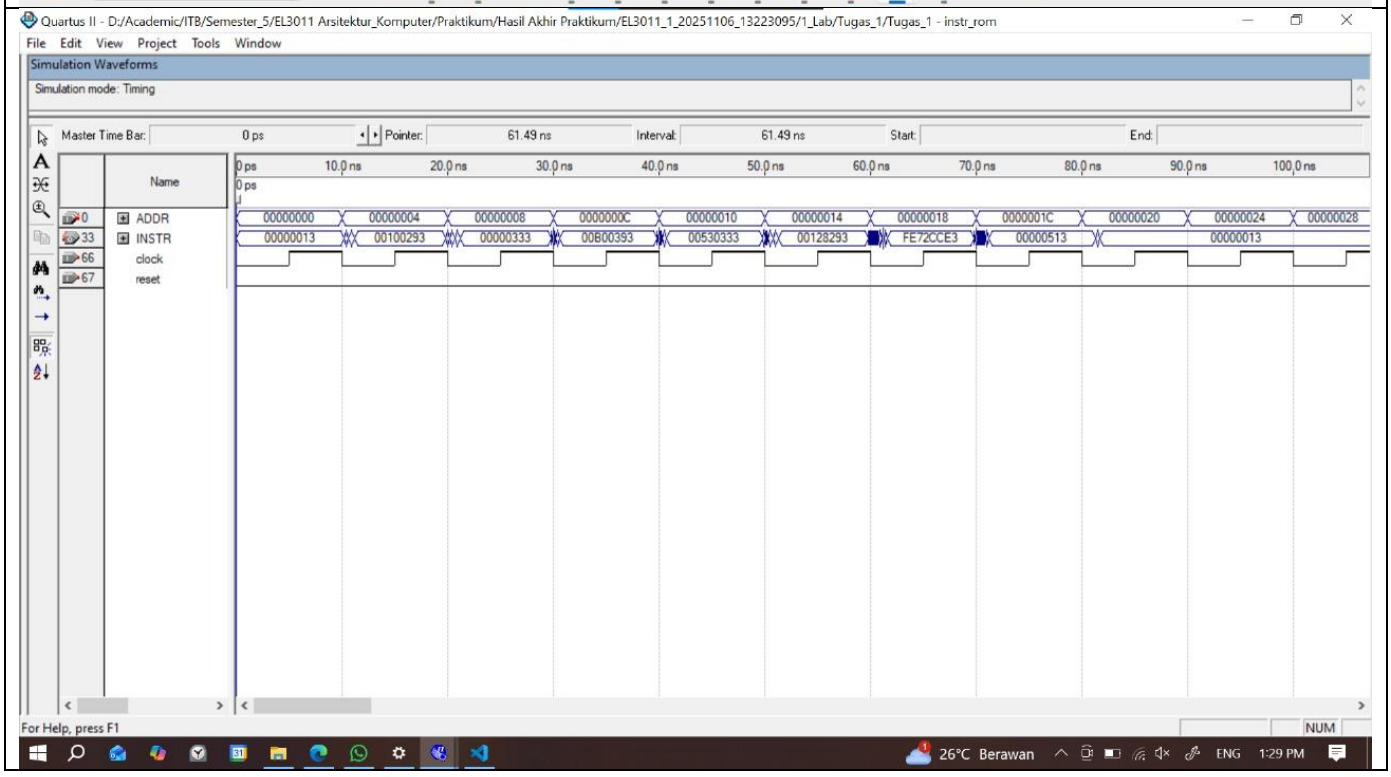
```
// Praktikum EL3011 Arsitektur Sistem Komputer
// Modul         : 1
// Percobaan     : 1
// Tanggal       : 4 November 2025
// Kelompok      :
// Rombongan     :
// Nama (NIM) 1 : Muammar Qadafi (13223094)
// Nama (NIM) 2 : Muhammad Falih Rosyidi (13223095)
// Nama File     : instr_rom_rv32i.v
// Deskripsi     : Instruction ROM 32x32 Single-Cycle RISC-V (RV32I)
module instr_rom_rv32i (
    input  wire [31:0] ADDR,   // byte address dari PC
    input  wire        clock,
    input  wire        reset,
```

```
    output reg  [31:0] INSTR   // instruksi 32-bit
);
  // 32 word x 32-bit
    reg [31:0] mem [0:31];
  // Word index = PC[6:2] (bit [1:0] = 2'b00)
    wire [4:0] waddr = ADDR[6:2];

    integer i;
    initial begin
    // Default: semua NOP (addi x0,x0,0)
    for (i = 0; i < 32; i = i + 1) mem[i] = 32'h00000013;

    // Isi program (ganti sesuai dengan "Machine Code" dari Venus pada Tugas
Pendahuluan nomor 3!):
    mem[ 0] = 32'h00100293; // addi x5, x0, 1
    mem[ 1] = 32'h00000333; // add  x6, x0, x0
    mem[ 2] = 32'h00B00393; // addi x7, x0, 10
    mem[ 3] = 32'h00530333; // add  x6, x6, x5
    mem[ 4] = 32'h00128293; // addi x5, x5, 1
    mem[ 5] = 32'hFE72CCE3; // addi x5, x7, -8
    mem[ 6] = 32'h00000513; // add  x6, x6, x5
    end

  // Baca sinkron; reset keluarkan NOP
    always @(posedge clock or posedge reset) begin
    if (reset) INSTR <= 32'h00000013;   // NOP RV32I
    else       INSTR <= mem[waddr];
    end
endmodule
```

2. Screenshot hasil tugas I







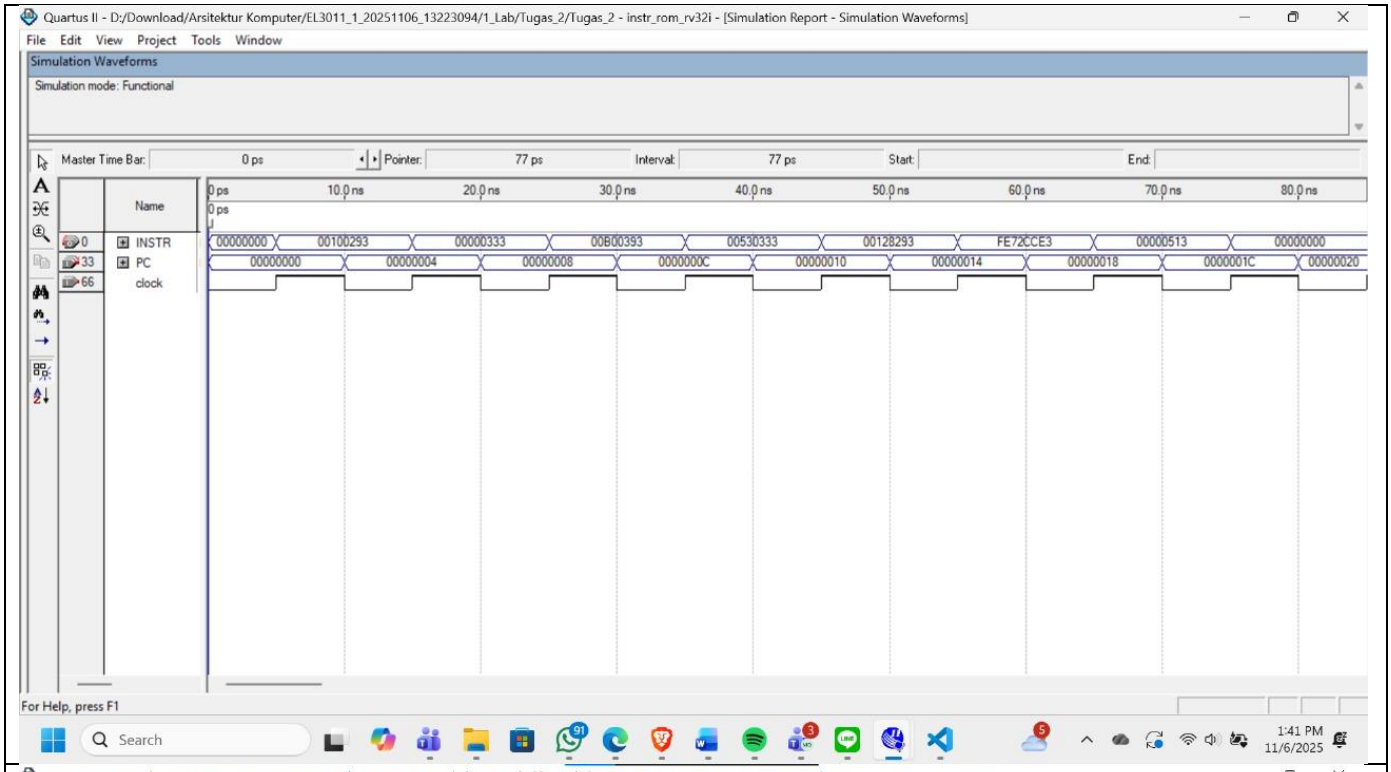
3. Source code untuk tugas 2

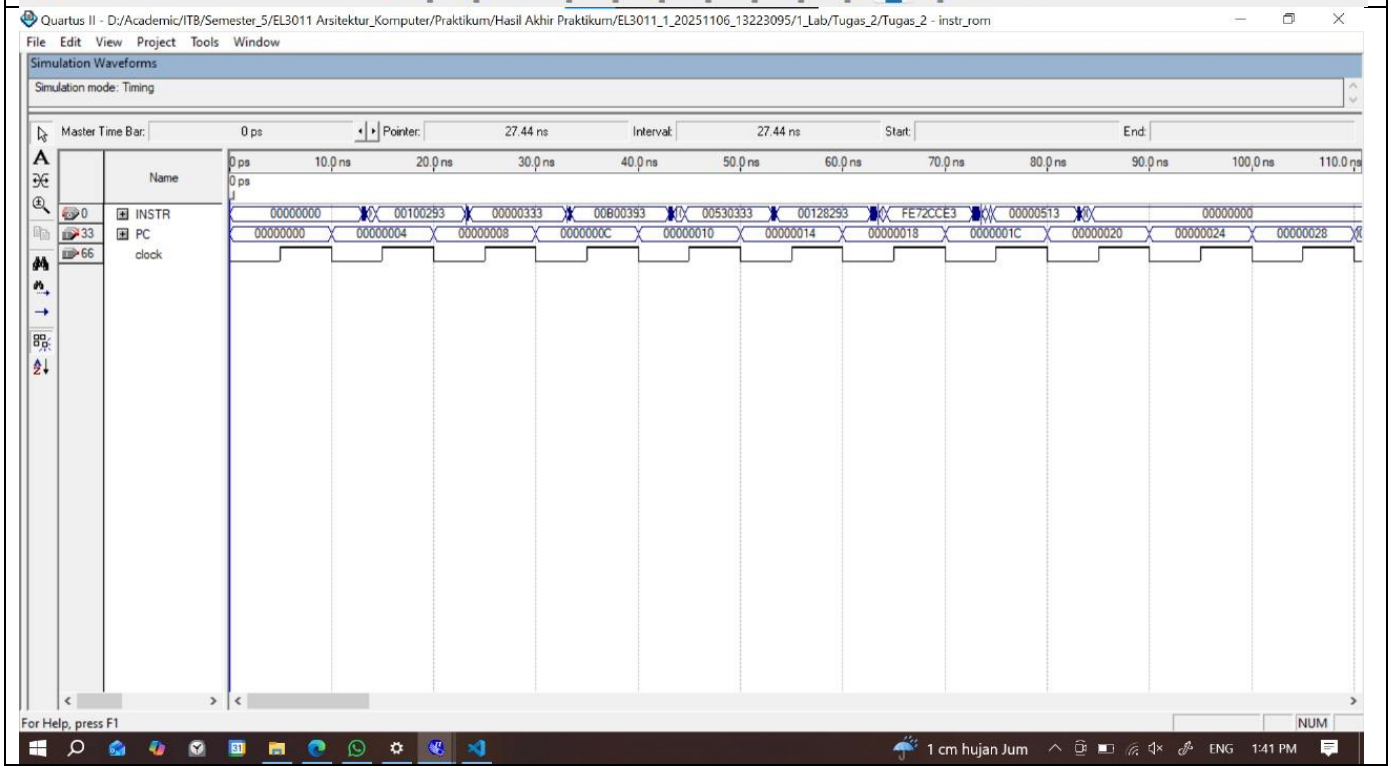
```
// Praktikum EL3011 Arsitektur Sistem Komputer
// Modul         : 1
// Percobaan     : 1
// Tanggal       : 6 November 2025
// Kelompok      :
// Rombongan     :
// Nama (NIM) 1 : Muammar Qadafi (13223094)
// Nama (NIM) 2 : Muhammad Falih Rosyidi (13223095)
```

```
// Nama File    : instr_rom_rv32i.v
// Deskripsi    : Instruction ROM 32x32 (RV32I) via ALTSYNCRAM + .mif
`timescale 1ns/1ps
module instr_rom_rv32i (
    input  wire        clock,
    input  wire [31:0] PC,      // byte address
    output wire [31:0] INSTR
);
  // Word index untuk 32 word
    wire [4:0] waddr = PC[6:2];

    altsyncram #(
    .operation_mode("ROM"),
    .width_a(32),
    .widthad_a(5),                  // 32 word
    .init_file("imemory.mif"),
    .outdata_reg_a("UNREGISTERED")
    ) rom (
    .clock0   (clock),
    .address_a(waddr),
    .q_a      (INSTR),
    .wren_a   (1'b0),
    .data_a   (32'b0)
    );
endmodule
```

```
WIDTH=32;
DEPTH=32;
ADDRESS_RADIX=HEX;
DATA_RADIX=HEX;
CONTENT
BEGIN
  00 : 00100293;  -- addi x1,x0,5
  01 : 00000333;  -- addi x2,x0,7
  02 : 00B00393;  -- add  x3,x1,x2
  03 : 00530333;  -- sw   x3,0(x0)
  04 : 00128293;  -- sw   x3,0(x0)
  05 : FE72CCE3;  -- sw   x3,0(x0)
  06 : 00000513;  -- sw   x3,0(x0)
  // Ganti dengan machine code dari Tugas Pendahuluan nomor 3!
END;
```

4. Screenshot hasil tugas 2





5. Source code untuk tugas 3

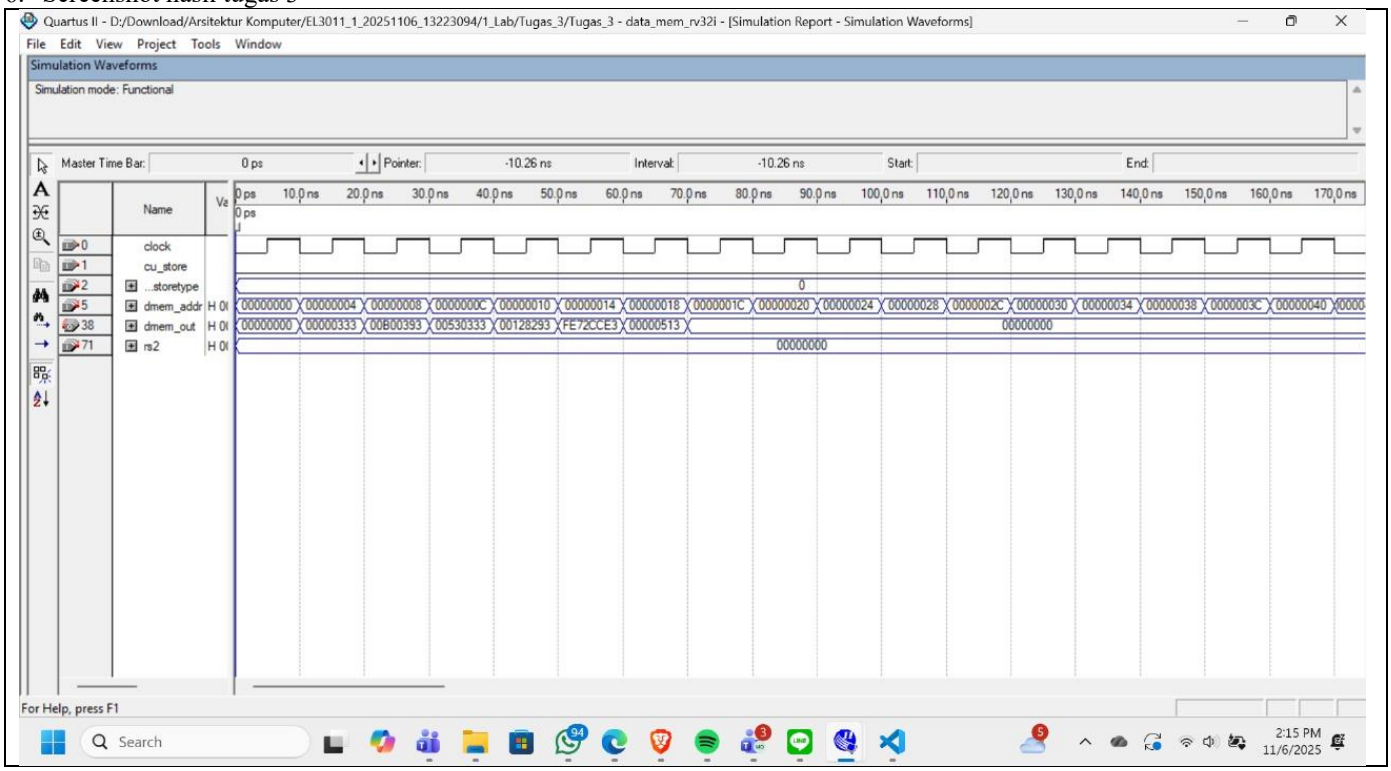
```
// Praktikum EL3011 Arsitektur Sistem Komputer
// Modul         : 1
// Percobaan     : 3
// Tanggal       : 6 November 2025
// Kelompok      :
// Rombongan     :
// Nama (NIM) 1 : Muammar Qadafi (13223094)
// Nama (NIM) 2 : Muhammad Falih Rosyidi (13223095
```

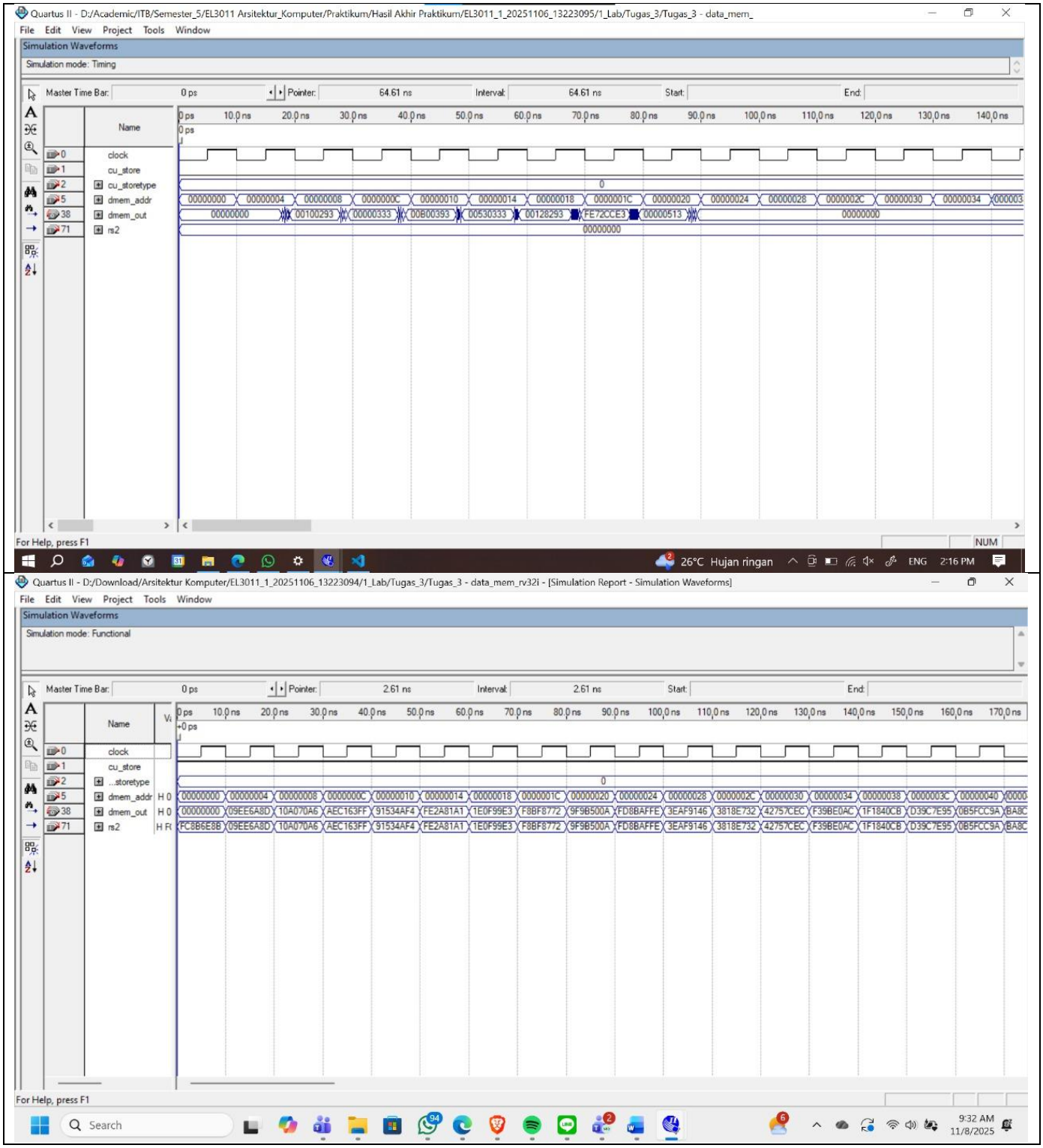
```
// Nama File     : data_mem_rv32i.v
// Deskripsi     : Data Memory 256 x 32-bit (RV32I) via ALTSYNCRAM

`timescale 1ns/1ps
module data_mem_rv32i (
    input  wire        clock,
    input  wire        cu_store,        // WE dari CU
    input  wire [1:0]  cu_storetype,    // 00=SW, 01=SH, 10=SB
    input  wire [31:0] dmem_addr,       // byte address
    input  wire [31:0] rs2,             // data tulis
    output wire [31:0] dmem_out         // data baca (async)
);
    wire [7:0] waddr = dmem_addr[9:2];

    reg  [3:0] be;
    always @* begin
    case (cu_storetype)
      2'b00: be = 4'b1111;                                          // SW
      2'b01: be = (dmem_addr[1]) ? 4'b1100 : 4'b0011;               // SH
      2'b10: case (dmem_addr[1:0])                                   // SB
                2'b00: be = 4'b0001;
                2'b01: be = 4'b0010;
                2'b10: be = 4'b0100;
                default: be = 4'b1000;
            endcase
        default: be = 4'b0000;
    endcase
    end

    reg [31:0] wr_data_aligned;
    always @* begin
    case (cu_storetype)
        2'b00: wr_data_aligned = rs2;                                 // SW
        2'b01: wr_data_aligned = (dmem_addr[1]) ? {rs2[15:0],16'b0} :
{16'b0,rs2[15:0]};// SH
        2'b10: case (dmem_addr[1:0])                                  // SB
            2'b00: wr_data_aligned = {24'b0, rs2[7:0]};
            2'b01: wr_data_aligned = {16'b0, rs2[7:0], 8'b0};
            2'b10: wr_data_aligned = {8'b0,  rs2[7:0], 16'b0};
        default: wr_data_aligned = {rs2[7:0], 24'b0};
        endcase
    default: wr_data_aligned = 32'b0;
    endcase
    end
    wire clk_n = ~clock;          // <<< write @negedge sistem

    wire [31:0] q_ram;
    altsyncram #(
    .operation_mode   ("SINGLE_PORT"),
    .width_a          (32),
    .widthad_a        (8),
    .outdata_reg_a    ("UNREGISTERED"),  // read async
    .init_file        ("dememory.mif"),
    .width_byteena_a  (4)
    ) ram (
    .clock0    (clk_n),                 // <<< pakai clock terbalik
    .wren_a    (cu_store),
    .address_a (waddr),
    .data_a    (wr_data_aligned),
    .q_a       (q_ram),
```

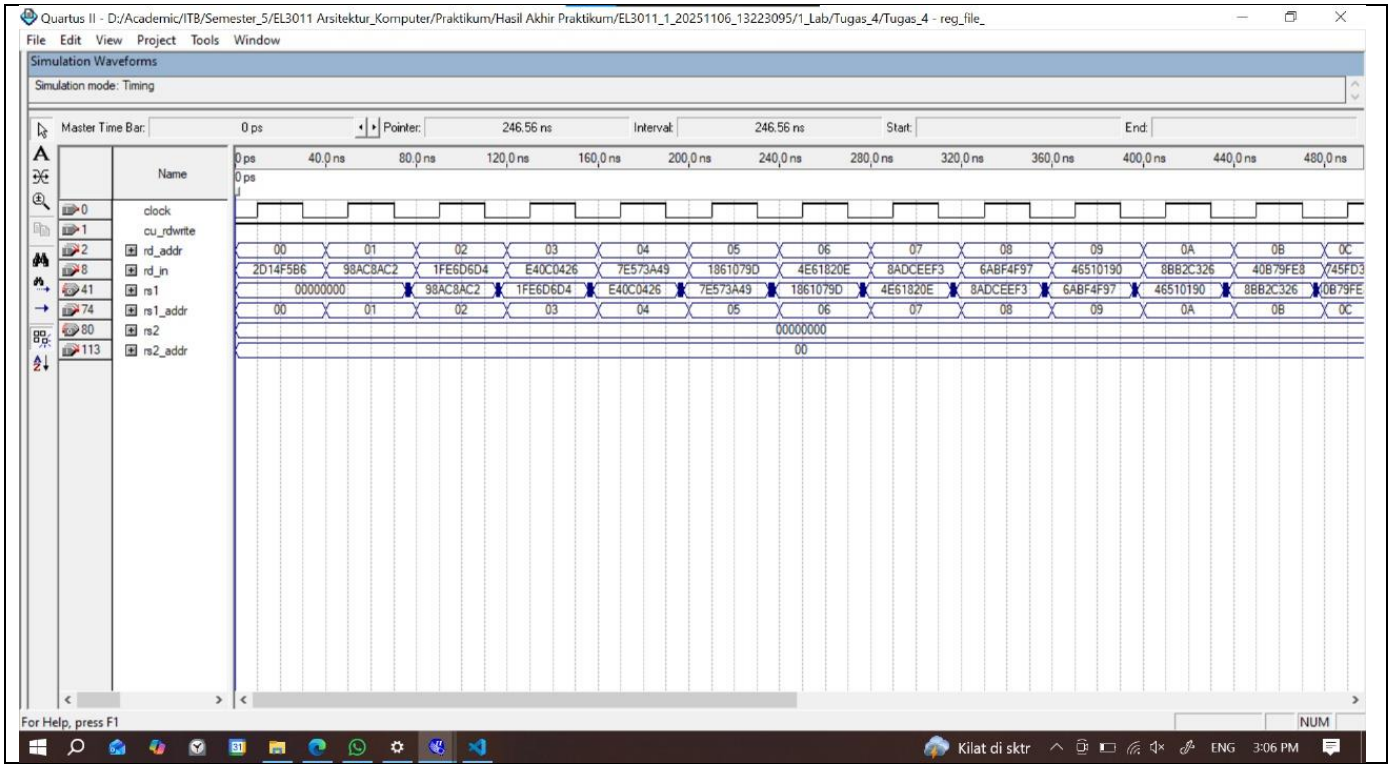
```
    .byteena_a  (be)
    );

    assign dmem_out = q_ram;      // <<< read "setiap saat"
endmodule
```

```
WIDTH=32;
DEPTH=32;
ADDRESS_RADIX=HEX;
DATA_RADIX=HEX;
CONTENT
BEGIN
  00 : 00100293;  -- addi x1,x0,5
  01 : 00000333;  -- addi x2,x0,7
  02 : 00B00393;  -- add  x3,x1,x2
  03 : 00530333;  -- sw   x3,0(x0)
  04 : 00128293;  -- sw   x3,0(x0)
  05 : FE72CCE3;  -- sw   x3,0(x0)
  06 : 00000513;  -- sw   x3,0(x0)
  // Ganti dengan machine code dari Tugas Pendahuluan nomor 3!
END;
```

6. Screenshot hasil tugas 3




Quartus II - D:/Academic/ITB/Semester_5/EL3011 Arsitektur_Komputer/Praktikum/Hasil Akhir Praktikum/EL3011_1_20251106_13223095/1_Lab/Tugas_3/Tugas_3 - data_mem_
File Edit View Project Tools Window
Simulation Waveforms
Simulation mode: Timing
Master Time Bar: 0 ps Pointer: 64.61 ns Interval: 64.61 ns Start: End:
Name
clock
cu_store
cu_storetype
dmem_addr
dmem_out
rs2
For Help, press F1
NUM
26°C Hujan ringan
ENG 2:16 PM
Quartus II - D:/Download/Arsitektur Komputer/EL3011_1_20251106_13223094/1_Lab/Tugas_3/Tugas_3 - data_mem_rv32i - [Simulation Report - Simulation Waveforms]
File Edit View Project Tools Window
Simulation Waveforms
Simulation mode: Functional
Master Time Bar: 0 ps Pointer: 2.61 ns Interval: 2.61 ns Start: End:
Name
clock
cu_store
...storetype
dmem_addr
dmem_out
rs2
For Help, press F1
Search
9:32 AM
11/8/2025



7. Source code untuk tugas 4

```
// Modul          : 1
// Percobaan      : 4
// Tanggal        : 6 November 2025
// Kelompok       :
// Rombongan      :
// Nama (NIM) 1 : Muammar Qadafi (13223094)
// Nama (NIM) 2 : Muhammad Falih Rosyidi (13223095)
// Nama File      : reg_file_rv32i.v
// Deskripsi      : Register file RV32I, 32x32, 2 read ports, 1 write port.
// x0 selalu 0; write @posedge, read @negedge
`timescale 1ns/1ps
module reg_file_rv32i (
input  wire        clock,          // global clock
input  wire        cu_rdwrite,     // write enable dari CU
input  wire [4:0]  rs1_addr,       // alamat baca port 1
input  wire [4:0]  rs2_addr,       // alamat baca port 2
input  wire [4:0]  rd_addr,        // alamat tulis (write-back)
input  wire [31:0] rd_in,          // data tulis (write-back data)
output reg  [31:0] rs1,            // data baca port 1
output reg  [31:0] rs2             // data baca port 2
);
// 32 register x 32-bit
reg [31:0] rf [0:31];
integer i;
initial begin
for (i = 0; i < 32; i = i + 1) rf[i] = 32'b0;  // untuk simulasi
end
// Tulis @posedge; tulis ke x0 diabaikan. Jaga x0 selalu nol.
always @(posedge clock) begin
if (cu_rdwrite && (rd_addr != 5'd0))
rf[rd_addr] <= rd_in;
rf[0] <= 32'b0;
end
```

```
// Baca @negedge (sinkron, sesuai instruksi praktikum)
always @(negedge clock) begin
rs1 <= (rs1_addr == 5'd0) ? 32'b0 : rf[rs1_addr];
rs2 <= (rs2_addr == 5'd0) ? 32'b0 : rf[rs2_addr];
end
endmodule
```

8. Screenshot hasil tugas 4