4 stage Pipelined Processor Instruction Set RISC-V RV32I

Izzan Nawa Syarif

Link Github : https://github.com/0TheMagik/PSD_RISC-V_Core_1.0
konversi RISC-V to hexcode : https://luplab.gitlab.io/rvcodecjs/#q=bne+x0,+x2,+-56&abi=false&isa=AUTO
RISC-V ISA Manual : https://drive.google.com/file/d/1uviu1nH-tScFfgrovvFCrj7Omv8tFtkp/view

Abstrak

RISC-V adalah sebuah Instruction Set open source yang dirancang untuk memberikan standar yang terbuka dan mudah di implementasikan dan modular. Meningkatnya penggunaan processor dengan instruction set RISC-V menjadikan instruction set ini banyak di pelajari dengan melakukan implementasi

Latar Belakang

Latar belakang dari proyek ini adalah jumlah dari implementasi Instruction Set RISC-V menggunakan Hardware Description Language (HDL) VHDL masih lebih sedikit dibandingkan dengan Implementasi menggunakan Verilog. Kebutuhan akademik yang menggunakan VHDL dalam proses perancangan sistem digital menjadi latar belakang utama di lakukannya proyek ini.

Tujuan

Tujuan dari proyek ini adalah merancang dan mengimplementasikan Instruction Set RISC-V RV32I dengan menggunakan Hardware Description Language (HDL) VHDL. Implementasi akan dilakukan dengan membuat pipeline 4 tahapan (Fetch, Decode, Execute, Write Back).

Inspirasi Proyek

Inspirasi dari proyek ini didasarkan dari perkembangan jumlah penggunaan Processor berbasis Instruction Set RISC-V di indusri maupun akademik di dunia. Instrution Set RISC-V yang bersifat terbuka (open source) memiliki potensi untuk menggantikan dan menyatukan standar industri serta akademik secara global.

Metode

instruction Set dan instruction Type

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

RV32I Base Instruction Set						
imm[31:12]				rd	0110111	LUI
imm[31:12]				rd	0010111	AUIPC
imm[20|10:1|11|19:12]				rd	1101111	JAL
imm[11:0]		rs1	000	rd	1100111	JALR
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	BEQ
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	BNE
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011	BLT
imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011	BGE
imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011	BLTU
imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011	BGEU
imm[11:0]		rs1	000	rd	0000011	LB
imm[11:0]		rs1	001	rd	0000011	LH
imm[11:0]		rs1	010	rd	0000011	LW
imm[11:0]		rs1	100	rd	0000011	LBU
imm[11:0]		rs1	101	rd	0000011	LHU
imm[11:5]	rs2	rs1	000	imm[4:0]	0100011	SB
imm[11:5]	rs2	rs1	001	imm[4:0]	0100011	SH
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	SW
imm[11:0]		rs1	000	rd	0010011	ADDI
imm[11:0]		rs1	010	rd	0010011	SLTI
imm[11:0]		rs1	011	rd	0010011	SLTIU

imm[11:0]		rs1	100	rd	0010011	XORI
imm[11:0]		rs1	110	rd	0010011	ORI
imm[11:0]		rs1	111	rd	0010011	ANDI
0000000	shamt	rs1	001	rd	0010011	SLLI
0000000	shamt	rs1	101	rd	0010011	SRLI
0100000	shamt	rs1	101	rd	0010011	SRAI
0000000	rs2	rs1	000	rd	0110011	ADD
0100000	rs2	rs1	000	rd	0110011	SUB
0000000	rs2	rs1	001	rd	0110011	SLL
0000000	rs2	rs1	010	rd	0110011	SLT
0000000	rs2	rs1	011	rd	0110011	SLTU
0000000	rs2	rs1	100	rd	0110011	XOR
0000000	rs2	rs1	101	rd	0110011	SRL
0100000	rs2	rs1	101	rd	0110011	SRA
0000000	rs2	rs1	110	rd	0110011	OR
0000000	rs2	rs1	111	rd	0110011	AND

Seluruh instruksi yang di tampilkan pada gambar di implementasikan dalam Proyek kecuali AUIPC.

Decoder merujuk pada Instruction Type untuk mendapatkan informasi dari sebuah instruksi yang di fetch

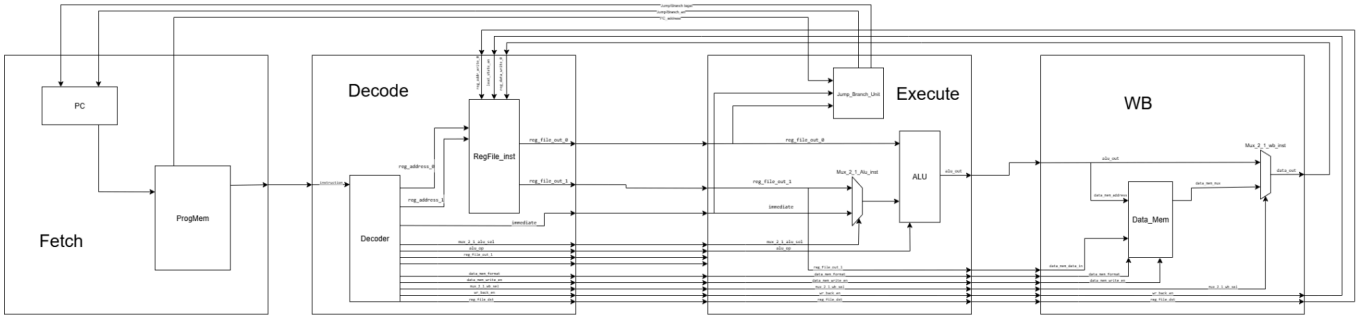
Kebanyakan dari instruksi kecuali Branch dan Jump selalu menggunakan ALU sehingga Line dari RegFile ke State Execute bisa menjadi sedikit dan tidak Terlalu banyak menggunakan

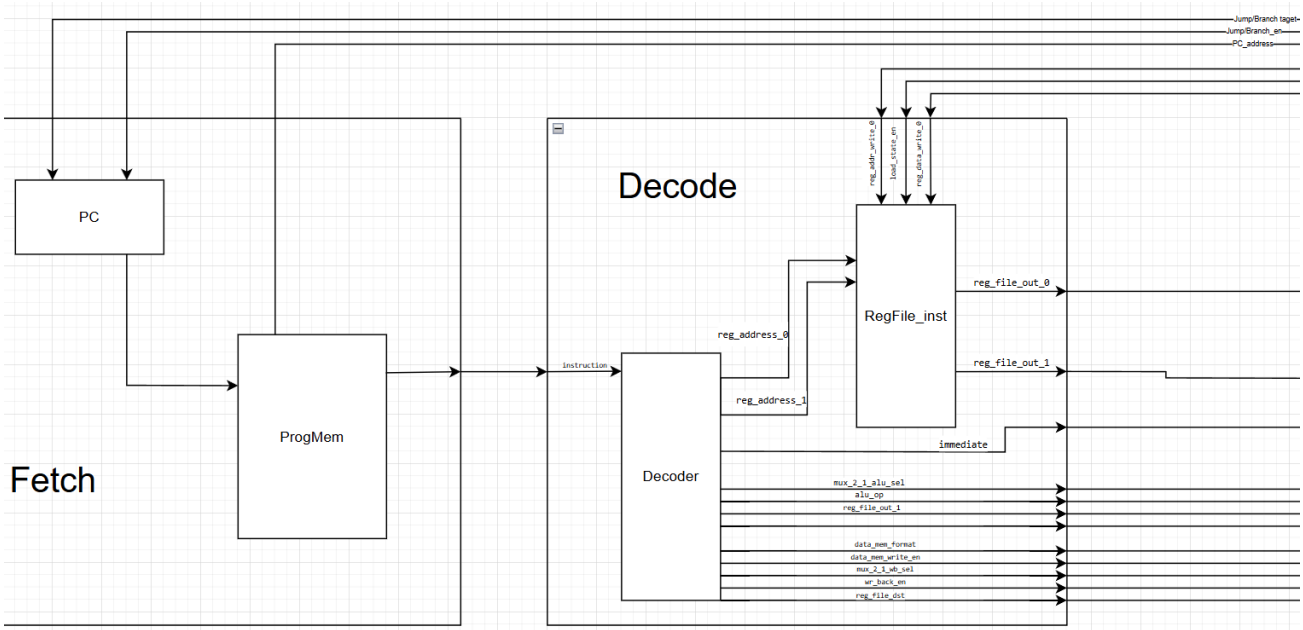
Dalam implementasinya Instruksi Store menggunakan Cycle yang lebih lama dibanding Instruksi lain sehingga mengakibatkan Data corruption karena instruksi belum selesai. untuk mengatasinya Perlu dibuat sebuah hazard unit untuk mendeteksi dan melakukan stall ke pipeline.

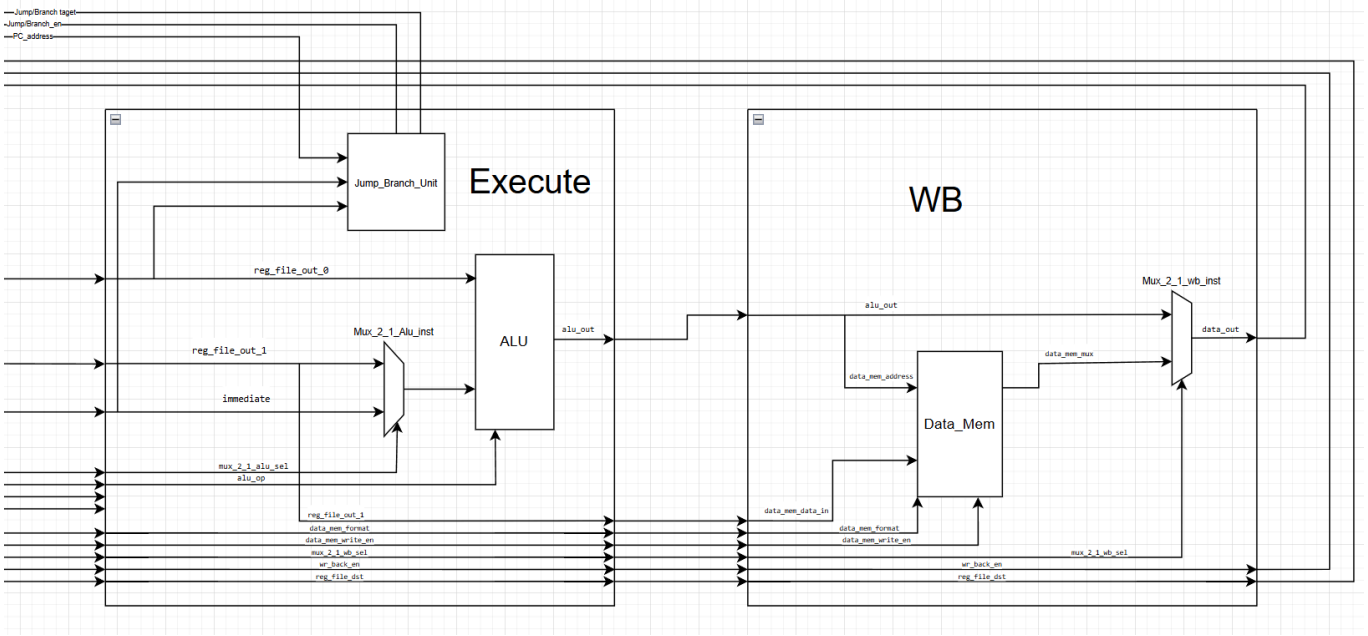
Instruksi Load dapat melakukan Load namun karena Store yang bermasalah pengisian data di Data_Memory perlu dilakukan dengan Manual di software simulasi.

Untuk melakukan Testing Testbench dengan nama `Core_tb.vhd` yang di compile dengan VHDL-2008 digunakan untuk melakukan Pengecekkan jalannya Processor.

Diagram Komponen







Modul yang digunakan

- **Microprogramming**

  Microprogramming di gunakan untuk mengubah sebuah instruksi menjadi sebuah kontrol ke komponen lain sehingga menentukan bagaimana kerja instruksi.
  Komponen tempat Microprogramming:
  - Decoder.vhd

- **FSM**

  FSM (Finite State Machine) digunakan untuk mengkontrol Keadaan dari Processor. Processor ini menggunakan 3 state Mealy machine dimana mempertimbangkan input dan current state untuk menentukan state berikutnya. penggunaan FSM ada pada Core.vhd yang memiliki 3 state yaitu LOAD_PC, IDLE, dan PIPELINE_RUN.
  Komponen dengan FSM:
  - Core.vhd

- **Data Flow style**

  Data flow style digunakan untuk assigment yang cepat dan tidak memerlukan pemeriksaan keadaan. salah satu penggunaannya ada pada State_fetch dimana address instruksi di assign ke output setelah dari Program Counter
  Komponen dengan Data Flow Style:
  - Mux2to1.vhd
  - State_Fetch.vhd
  - State_Decode.vhd
  - State_Execute.vhd
  - State_WB.vhd

- **Behavioral Style**

  Behavioral style digunakan untuk mendeskripsikan Sebuah perilaku untuk komponen. Penulisan dilakukan dengan menggunakan blok process yang akan di gunakan untuk signal assignment berdasarkan kodisi tertentu
  Komponen dengan Behavioral Syle:
  - Decoder.vhd
  - RegFile.vhd
  - Data_Memory.vhd
  - Program_Memory.vhd
  - Program_counter.vhd
  - Core.vhd
  - Core_tb.vhd

- **Structural Style**

  Structural style digunakan dengan menghubungkan Instace komponen dengan komponen lain sehingga menjadi sebuah Modul

Komponen dengan Structural Syle:

- Core.vhd
- State_Fetch.vhd
- State_Decode.vhd
- State_Execute.vhd
- State_WB.vhd

- **Looping**

  Looping di gunakan untuk melakukan cycle dari sebuah angka i sampai angka yang di tentukan atau kondisi tertentu

  Komponen yang menggunakan Looping:

  - Core_tb.vhd
  - RegFile.vhd

- **Function**

  Function digunakan untuk mengurangi penulisan dari bagian yang sering berulang

  komponen yang menggunakan Function:

  - RegFile.vhd
  - Program_Memory.vhd
  - Data_Memory.vhd

Komponen

1. **Core.vhd**

   komponen ini menyatukan seluruh bagian dari setiap tahapan yang akan dilalui instruksi. Bagian ini juga menyimpan Register yang menyimpan data dari setiap state. Register pada komponen ini di sinkronkan dengan clock. Setiap Register berada dalam process dan data hanya akan berubah jika ada clock berjalan

   Behavioral Style digunakan untuk penggunaan FSM karena memerlukan pengecekkan kondisi stiap risng_edge pada clock.

2. **State_Fetch.vhd**

   Komponen ini menyatukan Program counter dengan Program memory dan memberikan instruksi ke State Decode

   Structural style digunakan untuk menghubungkan Address untuk di fetch dari PC ke ProgMem, Output dari progmem akan terhubung dengan out pada State Fetch yang terhubung ke State Decode.

- **Program_Counter.vhd**

  Komponen ini melakukan +4 ke address progmem jika ada branch/jump maka address yang di berikan ke progmem untuk di fetch adalah address dari jump/branch tersebut.

  Behavioral style digunakan untuk mendeskripsikan perilaku dari PC karena di sinkronkan dengan clk. setiap clk rising_edge PC akan memberikan address+4 ke Progmem untuk di fetch.

- **Program_Memory.vhd**

  komponen ini adalah memory utama processor yang berisi instruksi yang akan di jalankan. Memory di buat menggunakan array berjumlah 1024 dan masing-masing berukuran 8-bit. Komponen ini memiliki input eksternal untuk melakukan load instruksi yang akan di jalankan.

  Behavioral style digunakan untuk melakukan penulisan instruksi ke memory dan pengambilan instruksi dari memory untuk di fetch ke Decoder.

3. **State_Decode.vhd**

   komponen ini menyatukan RegFile dan decoder. komponen ini akan menerima data dari State_Fetch yang berupa instruksi dari ProgMem dan dihubungkan ke Decoder.

- **RegFile.vhd**

  Komponen ini berperan sebagai register dari Processor, Register dibuat dengan menggunakan array berjumlah 32 dengan ukuran 32 bit. Register x0 akan selalu 0 menyesuaikan dengan spesifikasi RISC-V.

  Data flow style digunakan untuk konversi address ke integer dengan function convert_addr(), hasil konversi akan dia assign ke signal internal. Serta akses keluar data dari register.

  Behavioral style digunakan untuk melakukan penulisan data ke register yang address nya sudah di konversi ke integer.

  Looping For loop digunakan untuk clear Register, For loop di pilih karena lebih mudah di sintesis.

- **Decoder.vhd**

  Komponen ini berperan sebagai pengatur utama bagaimana sebuah instruksi akan berjalan melalui pipeline.

Behavioral style digunakan pada komponen ini untuk mendeskripsikan perilaku dari komponen dengan blok process. Setiap line pada blok process berjalan sekuensial.

Microprogramming di gunakan pada komponen ini dengan memberikan sinyal kekomponen lain untuk setiap instruksi yang di decode. Sinyal yang di kirimkan dari komponen ini akan mengontrol alur dari instruksi yang berjalan

Instruksi yang di Decode akan memberikan Sebuah value yang akan di gunakan untuk mengatur komponen lain pada Core.

4. **State_Execute.vhd**

Komponen ini menerima hasil dari state decode dan melakukan eksekusi bersarkan data yang dikirim. Data yang dikirim tersebut akan masuk ke dalam komponen yang berada pada state tersebut dan di execute lalu keluar dari state tersebut.

Structural Style digunakan pada komponen ini dengan membuat instance komponen ALU, Mux_2_1, dan Jump_branch_unit. ketigaa komponen tesebut kemudian di hubungkan dengan satu sama lain dengan komponen State_Execute sebagai Top level yang menyatukan ketiganya.

- **Mux2to1.vhd**

Komponen ini akan diatur oleh sinyal dari decoder, dimana jika decoder memberikan `0` maka output dari mux adalah input dari `in_mux_0` atau `in_mux_1` jika decoder memberikan `1`.

Dataflow style digunakan pada komponen ini dengan mendeskripsikan bagaimana data akan melalui sebuah hardware. Semua sinyal yang melalui komponen ini akan berjalan secara konkuren dan tidak ada delay karena berada pada blok utama di architecture .

- **ALU.vhd**

Komponen ini berfungsi untuk melakukan arithmathic antara `input_0` dan `input_1`. input_1 bergantung dengan mux_2_1 sehingga dapat berupa immediate atau nilai dari register.

Behavioral style di gunakan dalam komponen ini dengan penggunaan blok process untuk melakukan eksekusi dengan sekuensial karena melakukan pengecekkan data opcode yang dikirim dari decoder untuk melakukan eksekusi.

- **Jump_Branch_Unit.vhd**

komponen ini di gunakan untuk melakukan kalkulasi dan perbandingan pada Jump dan Branch. Saat akan melakukan Brach atau jump PC akan di jumlahkan dengan offset yang berasal dari immediate+nilai pada reg_file.

Hasil dari kalkulasi akan dimasukkan kembali ke PC sebagai address yang akan di Fetch ke Decoder.

Behavioral digunakan dalam komponen ini sama seperti alu dengan blok process untuk pengecekkan data opcode yang di terima dari decoder.

5. **State_WB.vhd**

Komponen ini menyatukan seluruh komponen dalam state Write Back. Output dari komponen ini akan di tuliskan kembali ke Reg_file pada state Decode

Structural style di gunakan dalam komponen ini dengan membuat instance komponen Data Memory_dan Mux_2_1 yang kemudian dihubungkan satu sama lain. input dari State_WB.vhd di hubungkan ke Data memory dan Mux.

- **Data_Memory.vhd**

  komponen ini adalah controller untuk mengakses Data memory tempat Store dan Load dari data akses ke data memory dilakukan dengan menggunakan address+offset yang di kalkulasikan dari ALU. Memory di buat sebagai array 2048 dengan ukuran 8 bit.

  Behavioral style dan function di gunakan pada komponen ini dengan menggunakan blok process untuk mengidentifikasi mode store atau load dan format apa yang di gunakan. Untuk mendapatkan nomor array yang akan di akses function `convert_addr()` akan melakukan konversi data `std_logic_vector` ke integer angka integer tersebut akan digunakan untuk menunjuk ke address yang akan di akses.

- **Mux2to1.vhd**

  komponen Mux pada State ini di gunakan sebagai pemilih dari mana data berasal untuk di teruskan ke Reg_file

  Dataflow style digunakan pada komponen ini dengan mendeskripsikan bagaimana data akan melalui sebuah hardware. Semua sinyal yang melalui komponen ini akan berjalan secara konkuren dan tidak ada delay karena berada pada blok utama di architecture.

6. **Core_tb.vhd**

Komponen ini adalah cloked testbench yang digunakan untuk melakukan pengetesan input secara otomatis dan berjalan selama 30 rising_edge. Terdapat 3 test input pada program Test pertama adalah kondisi dimana processor mendapatkan

program yang tidak memiliki data hazard sama sekali. Test kedua processor mendapatkan Data Hazard dan store yang mengakibatkan Kesalahan output ke register tujuan. Tes ketiga Processor mendapatkan Branch, namun karena koneksi jump_branch_unit.vhd tidak sempurna (bugged) branch gagal di laksanakan.

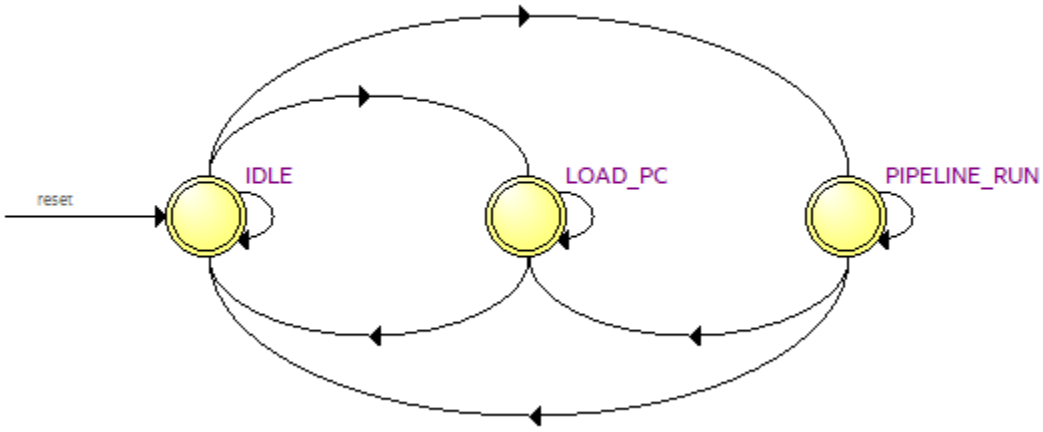
Testbench ini menggunakan Behavioral Style untuk menjalankankan input dan menjalankan processor.

Dokumentasi

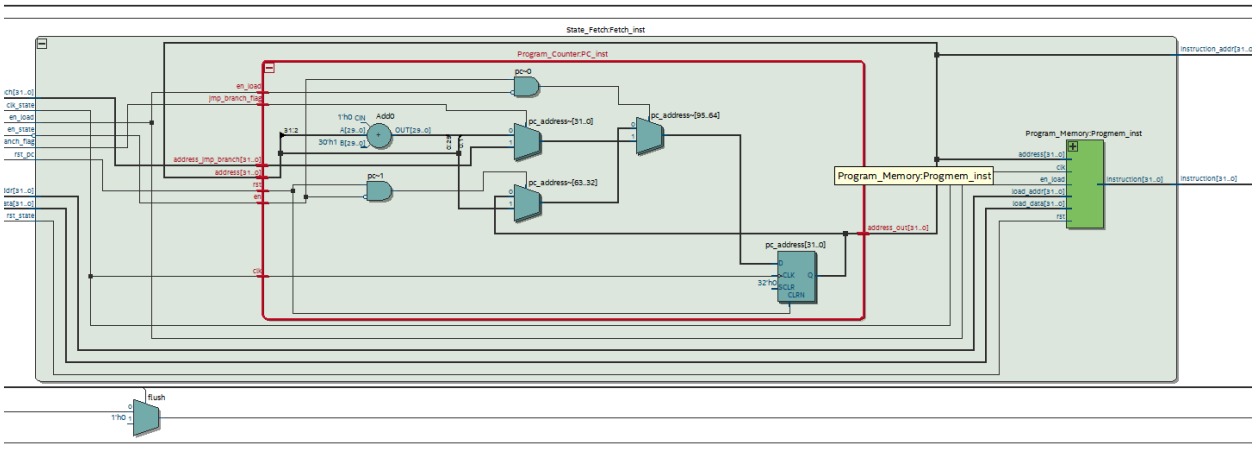
* Hasil sintes yang lebih besar bisa di lihat pada folder Synthesizeable

Processor

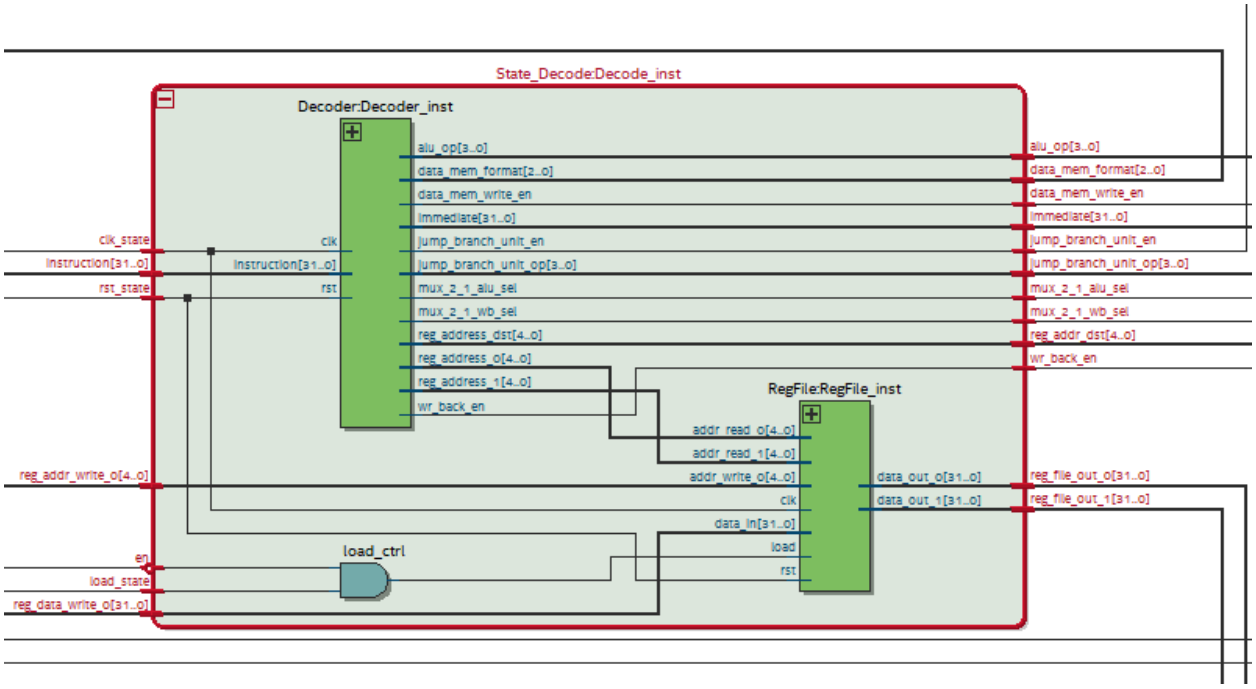
- FSM



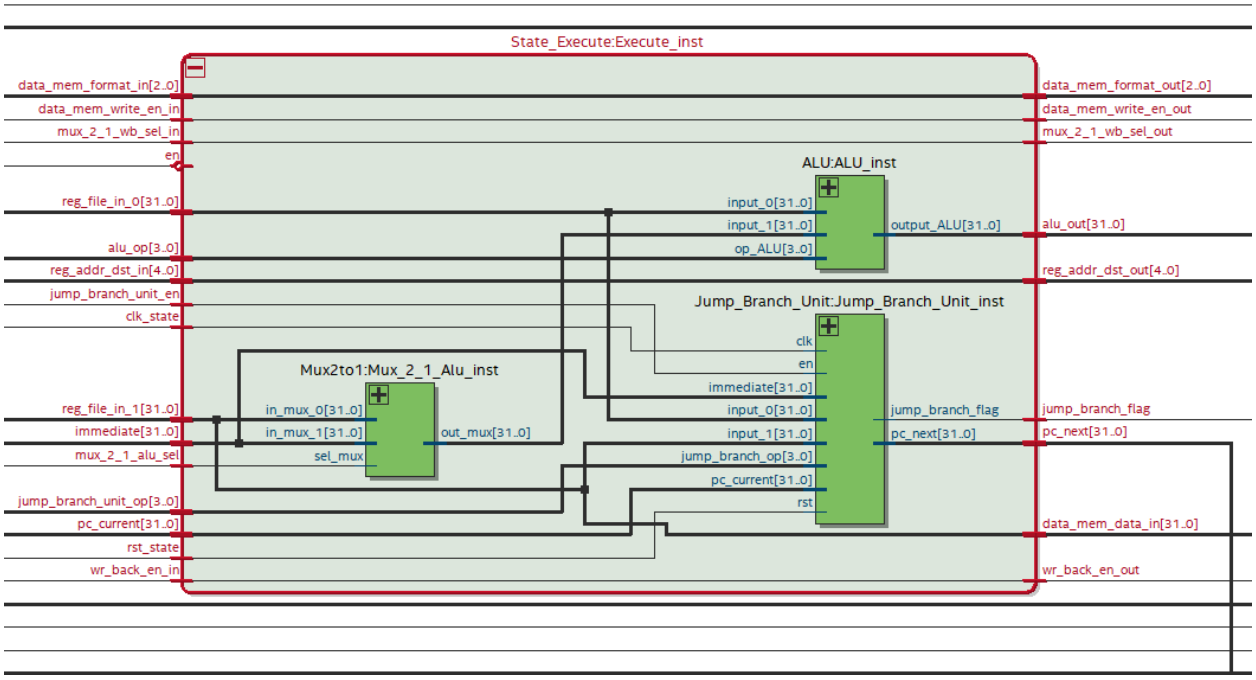
- State Fetch



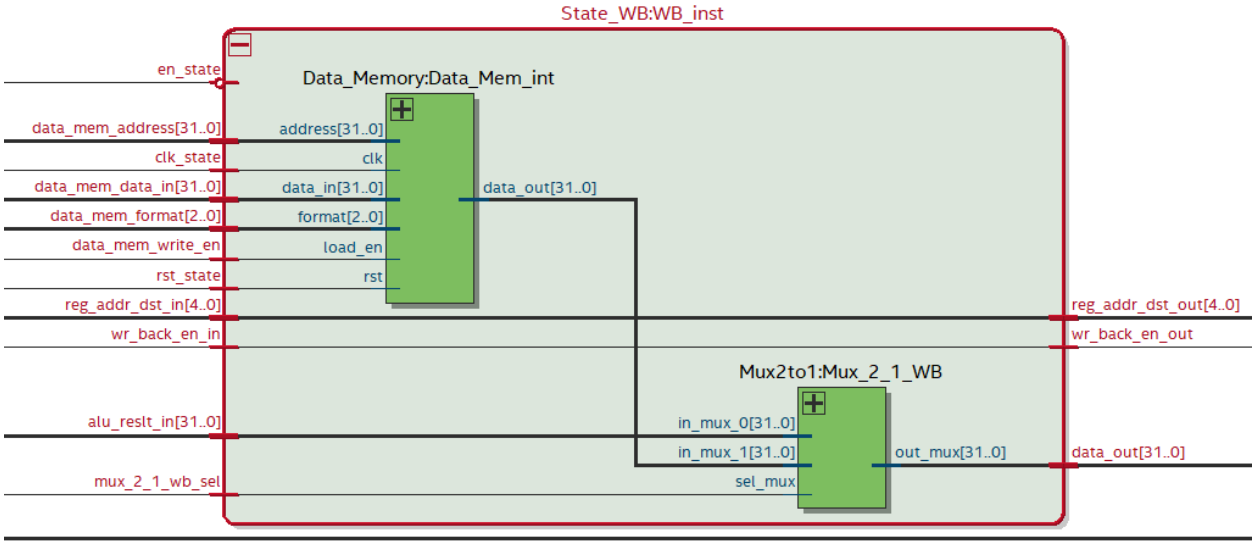
- State_Decode



- State_Execute

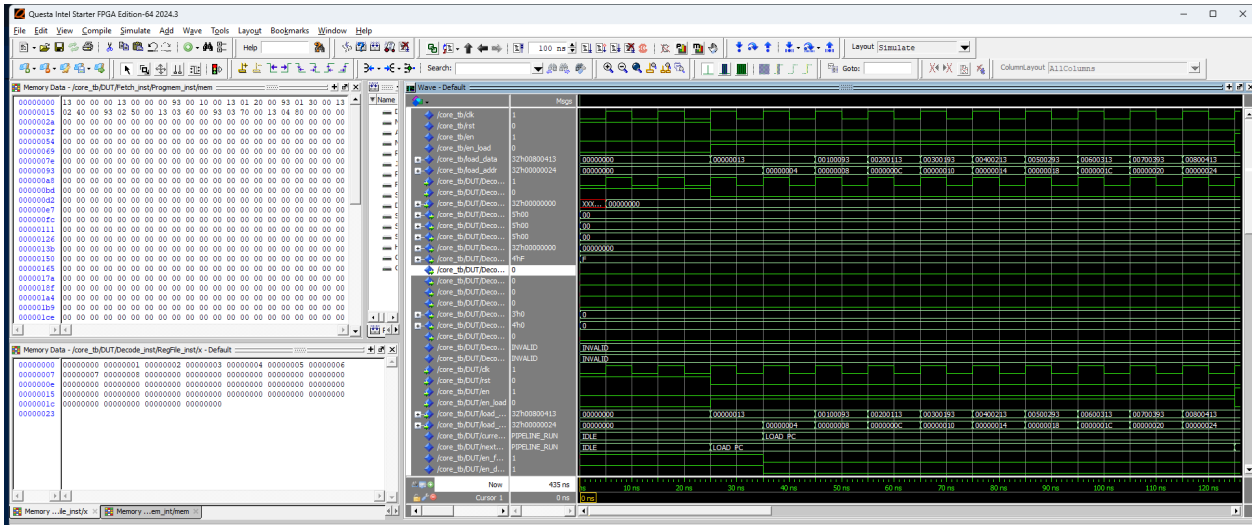


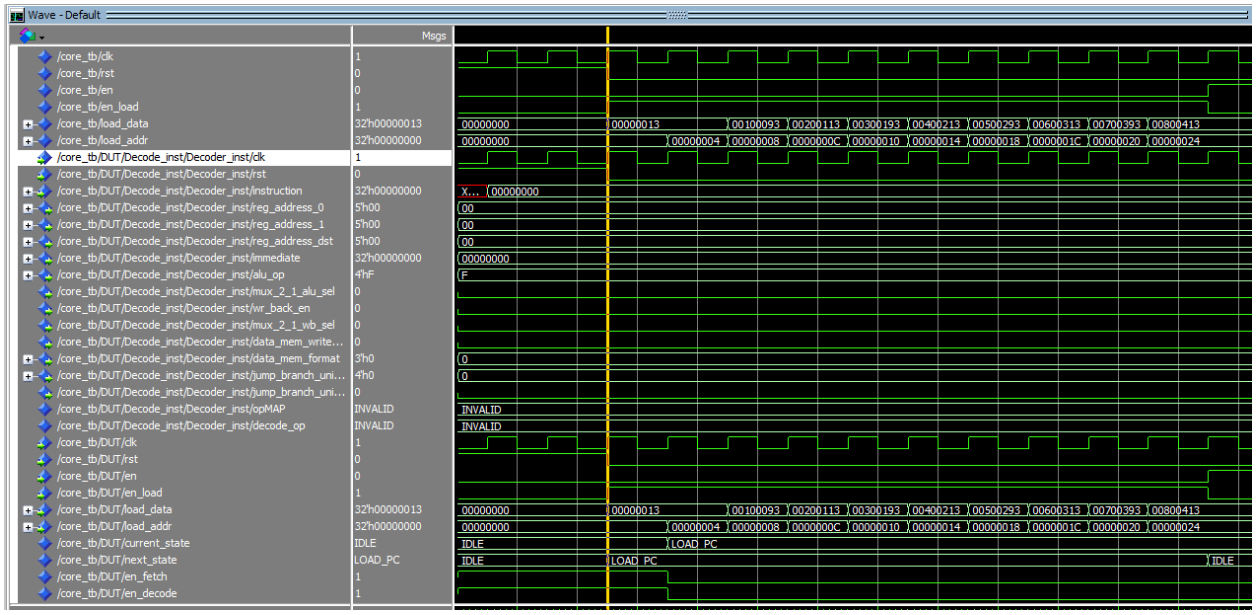
- State_WB

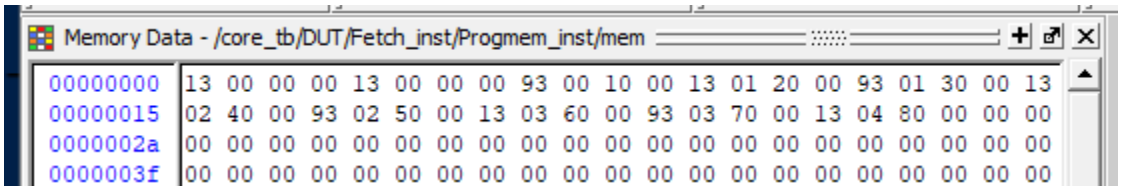


Test Case 1

- **Input instruksi**

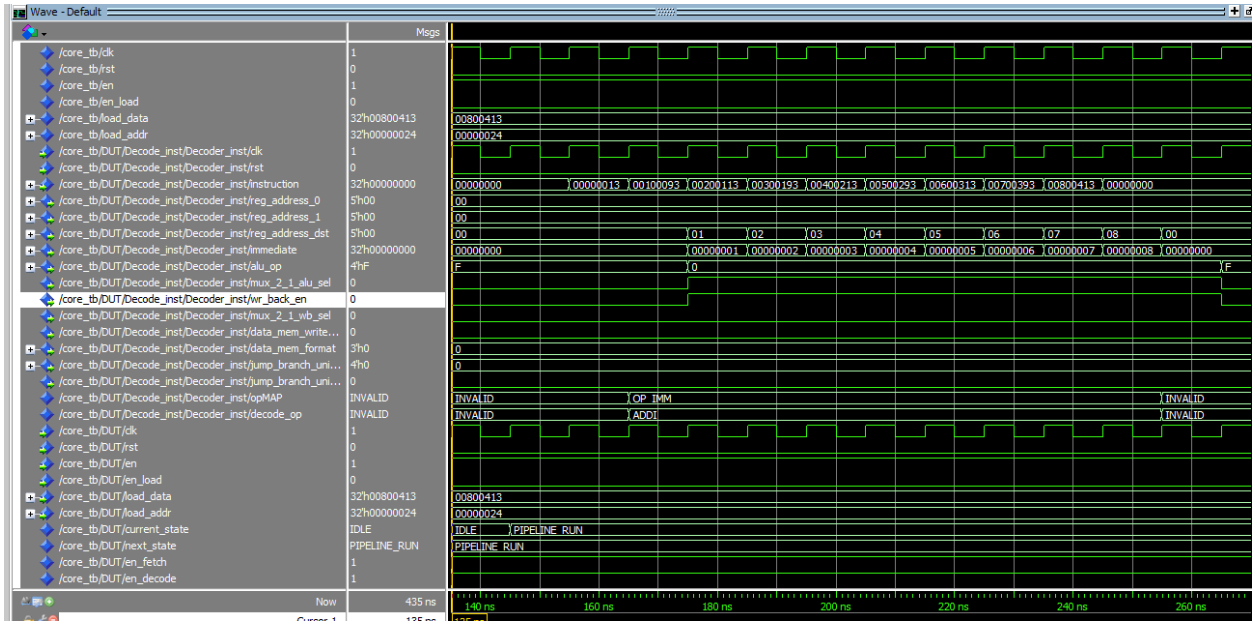


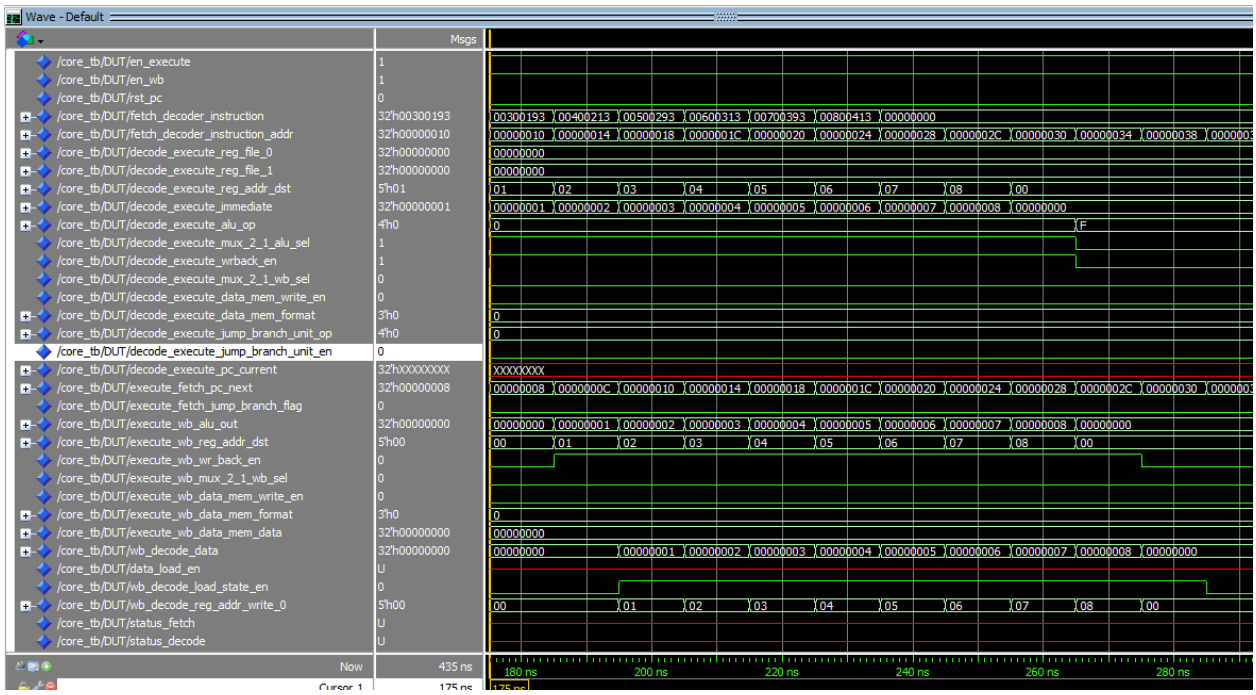




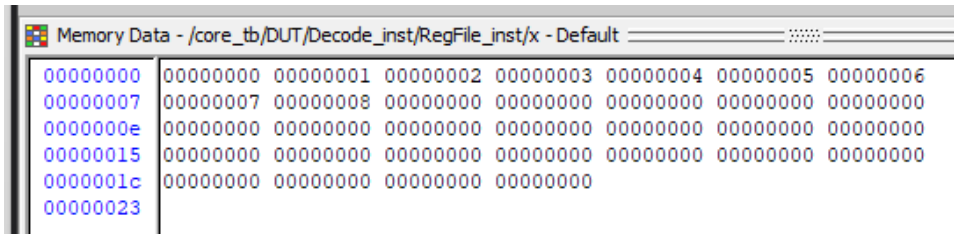
Seluruh nya mengiisi Progmem

- **Pipeline Terisi penuh**



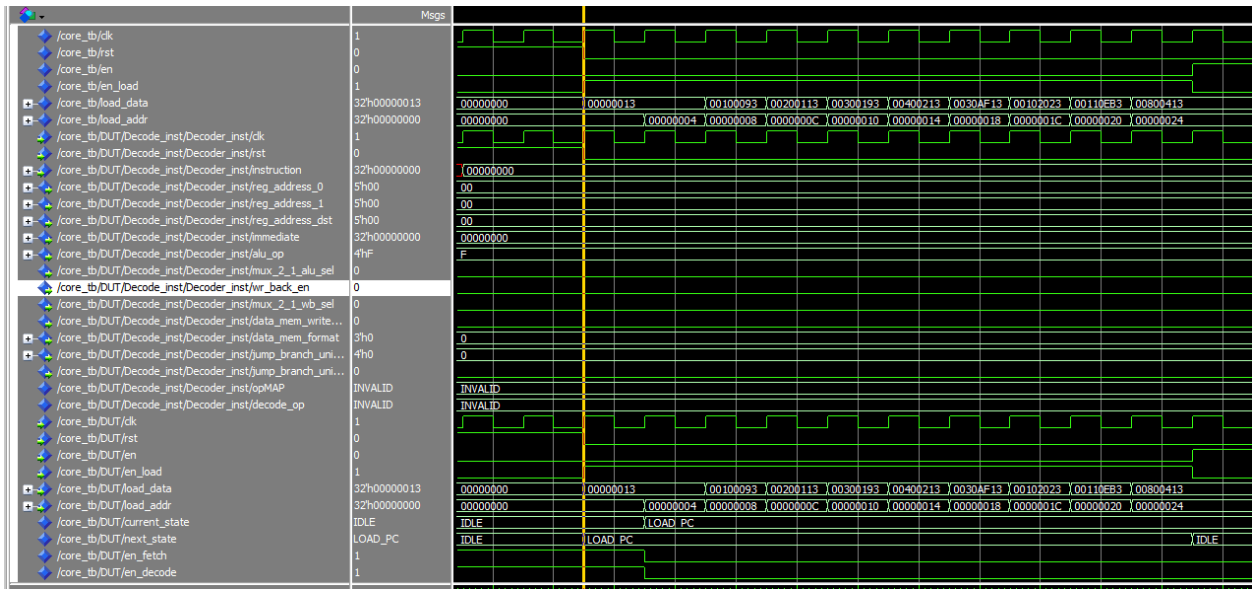


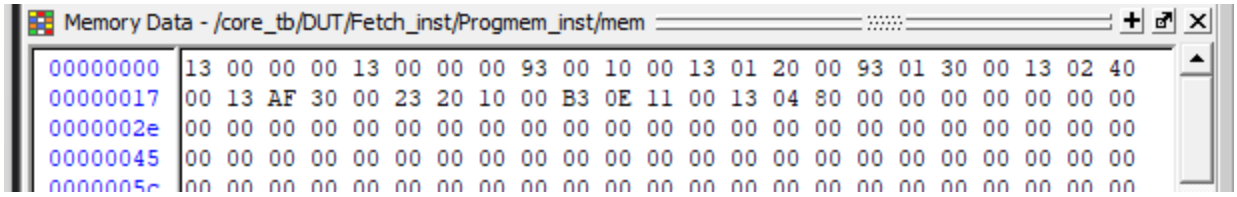
- **Hasil**



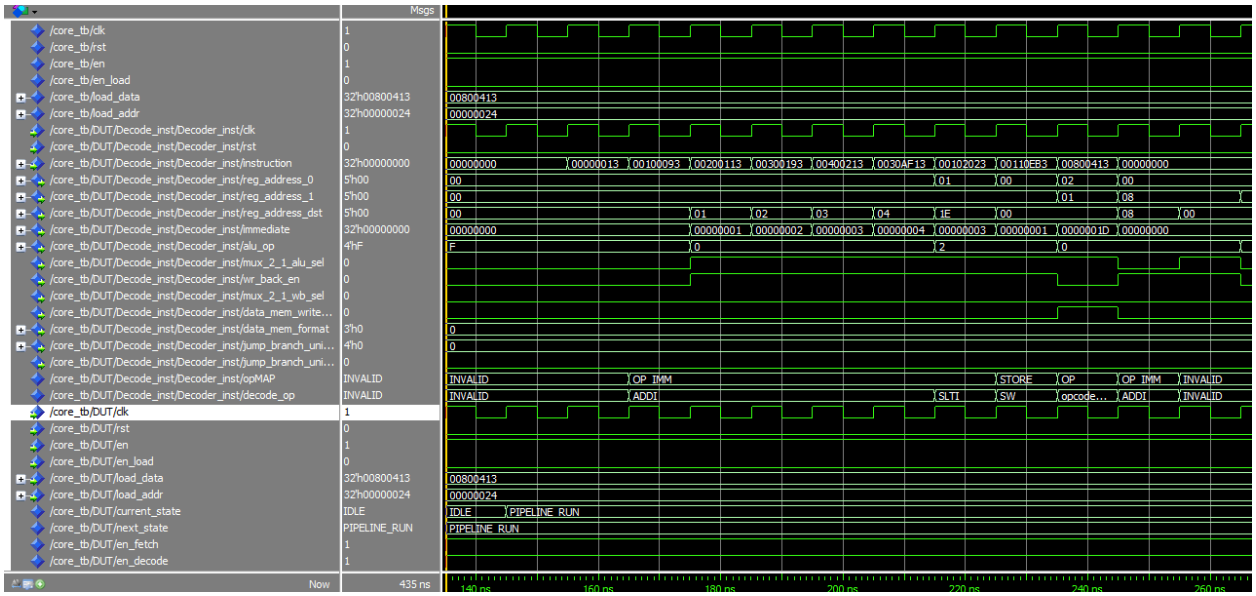
Test Case 2

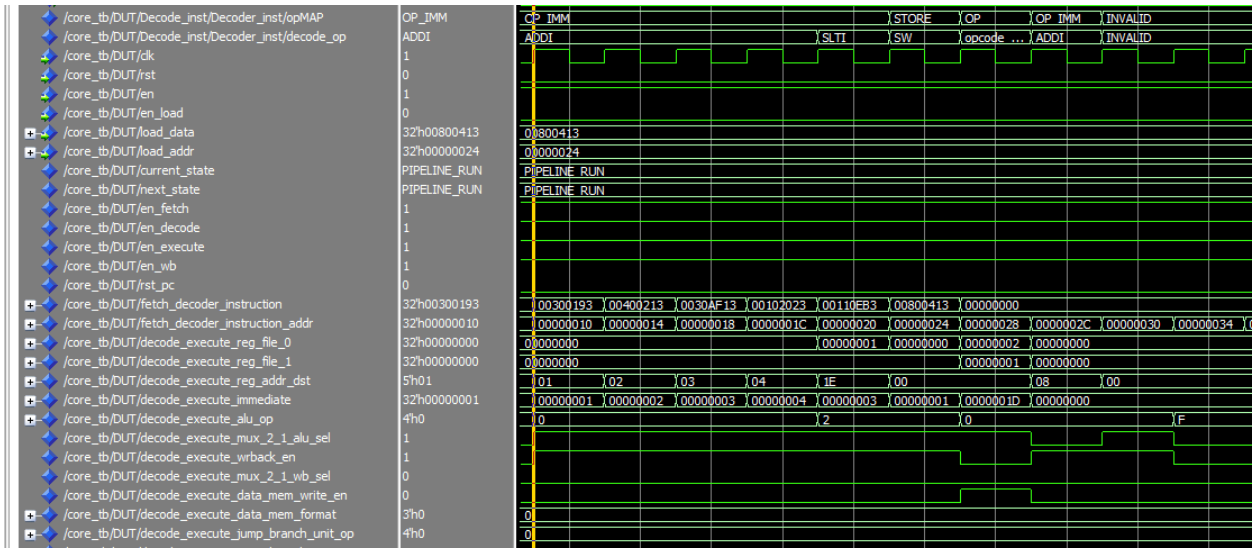
- **Input instruksi**

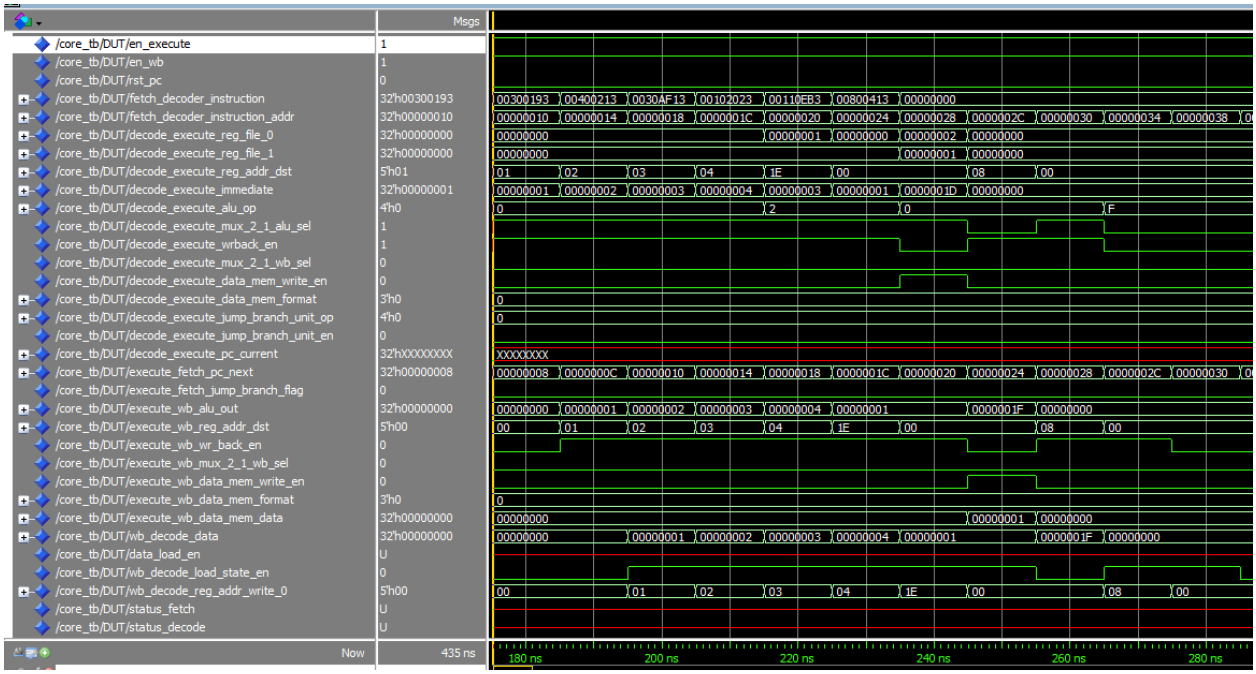




- **Pipeline terisi penuh (menjalankan program)**

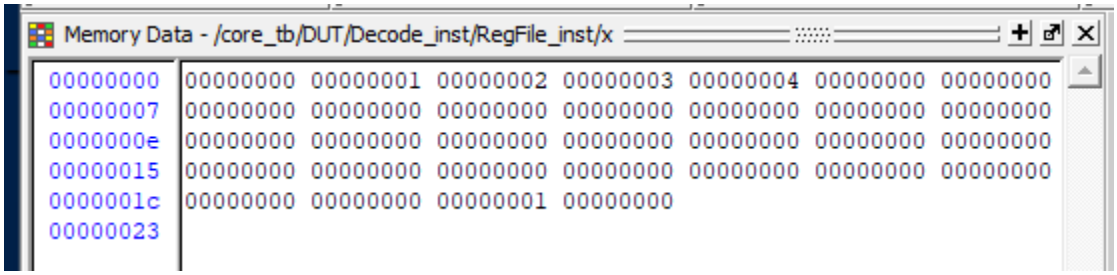




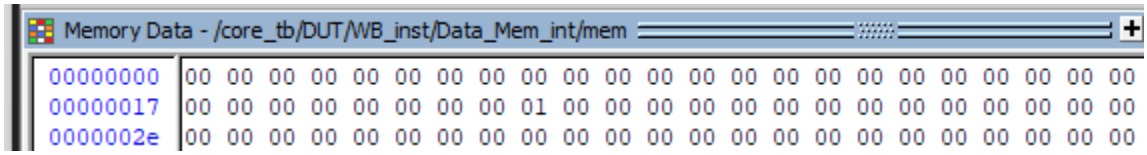


- **Hasil**

Register



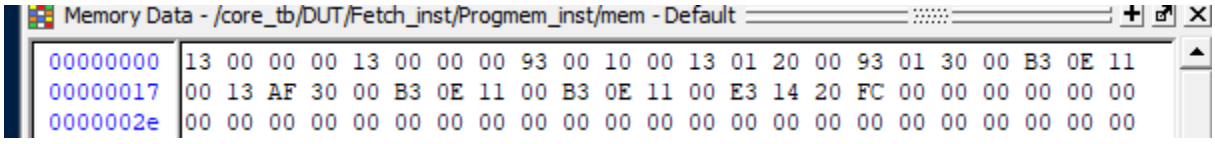
Data_memory

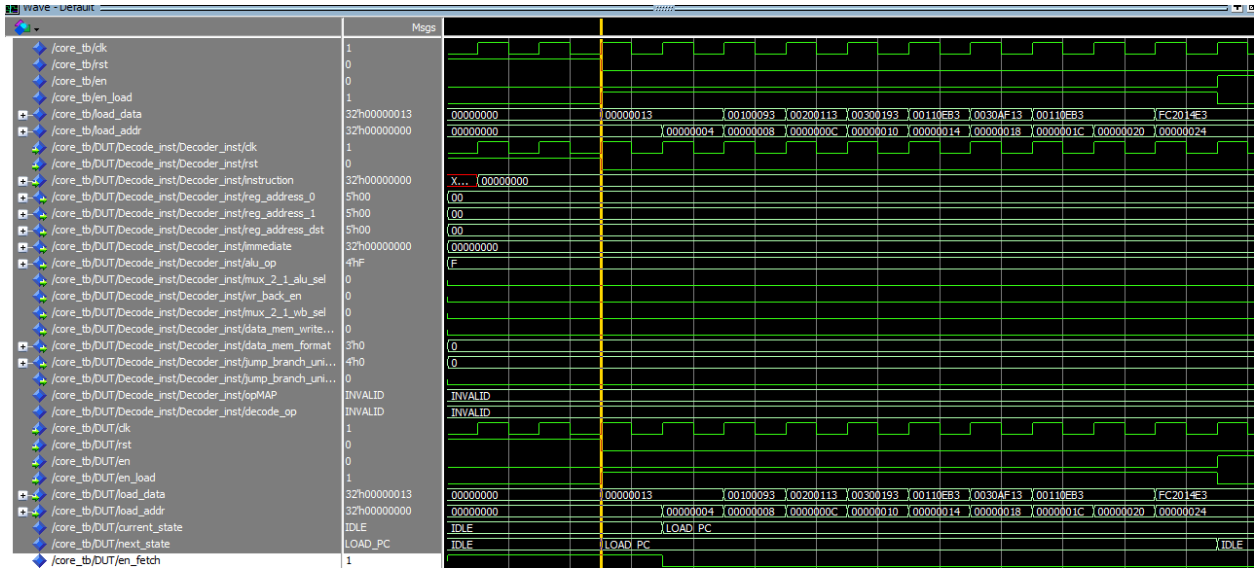


Instruksi Store memerlukan lebih banyak cycle, data hazard terjadi

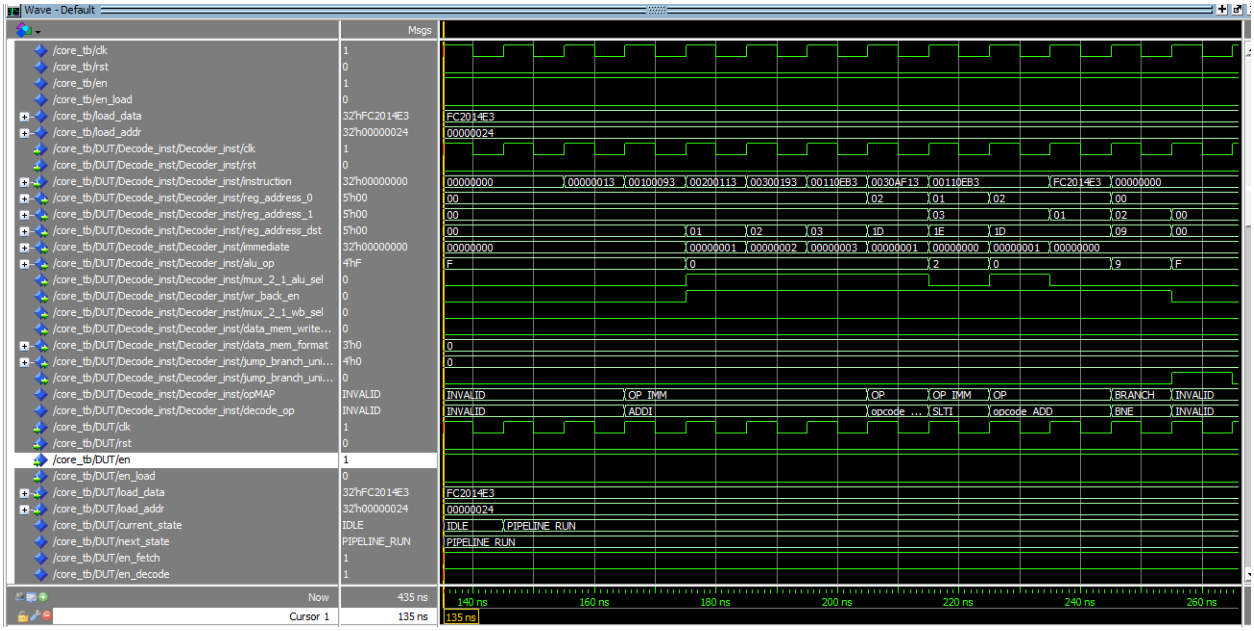
Test Case 3

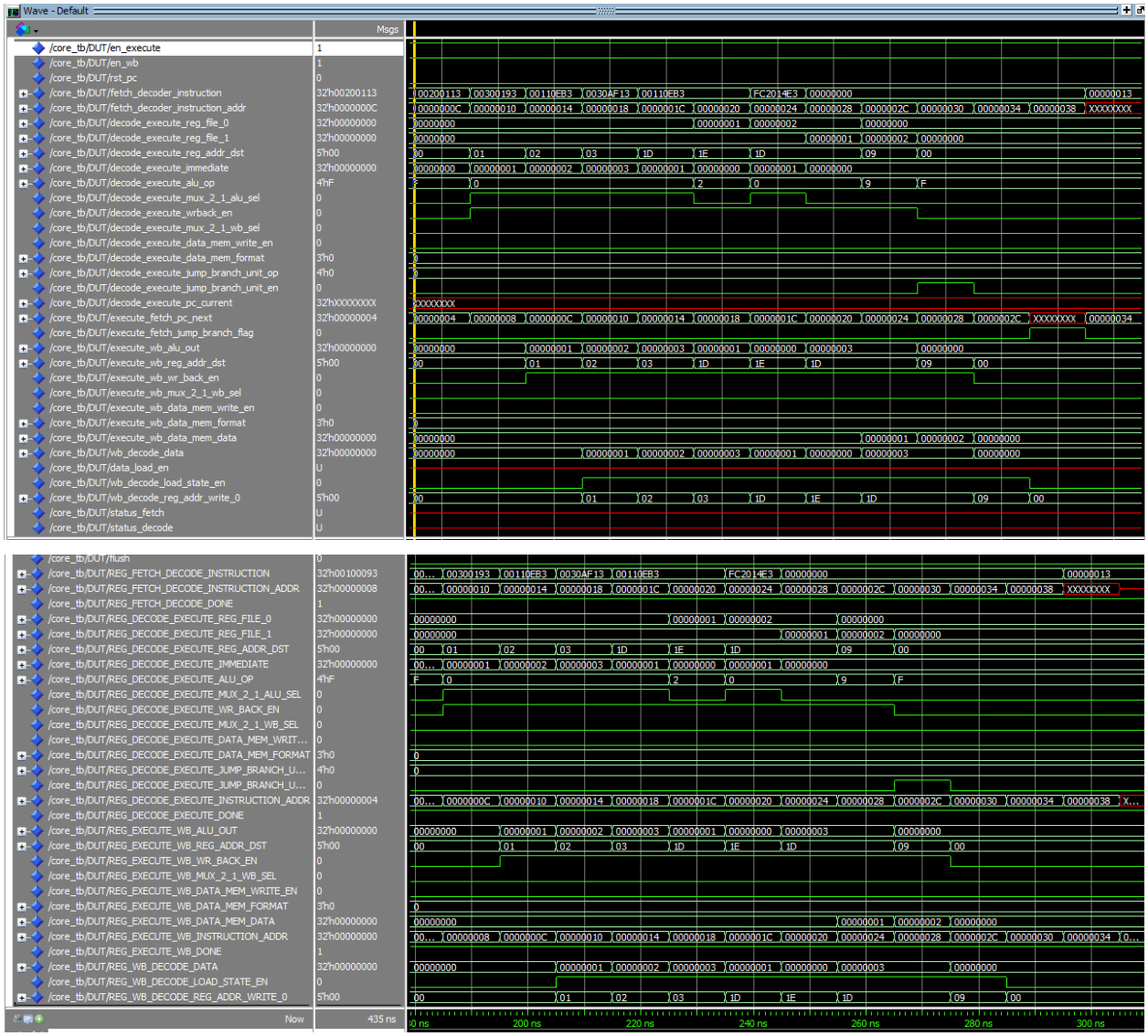
- **Input instruksi**



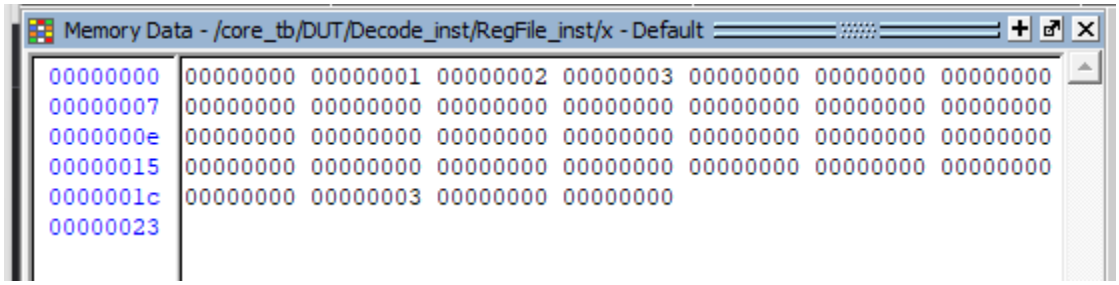


- **Pipeline terisi penuh (menjalankan program)**





- **Hasil**



Branch gagal terjadi karena koneksi jump_branch_unit yang belum Sempurna

Masalah Implementasi

1. Memaksimalkan throughput pada Pipeline

Untuk memaksimalkan instruksi yang bisa di proses dalam satu waktu, maka Pipeline digunakan untuk memperbanyak jumlah instruksi yang dapat di process. Sebelum pipeline digunakan instruksi di process dalam waktu 4 cycle untuk 1 instruksi. Setelah Pipeline di implementasikan dalam 4 cycle, 4 instruksi dapat di proses.

Implementasi Pipeline dilakukan dengan memasukkan setiap Output dari setiap komponen ke sebuah register yang terhubung ke tahapan selanjutnya. Register yang terhubung tersebut di sinkronkan dengan clk shingga akses hanya terjadi jika rising_edge. Setiap rising_edge maka isi dari register akan berganti menjadi isi dari stage sebelumnya

Status : Resolved, Processor dapat menjalankan instruksi dalam Pipeline

2. Unsable Space in memory

Karena penggunaan Pipeline dalam processor maka Processor memerlukan waktu untuk mengisi pipeline sehingga instruksi tidak bisa di eksekusi dengan benar untuk 2 instruksi pertama

Solusi dari masalah tersebut adalah mengisi 2 address pertama yang di fetch oleh PC dengan `addi x0, x0, 0` instruksi ini tidak mengubah apapun dalam data memory maupun program memory

Status : Resolved penulisan program sesuai dengan ketentuan, Side effect dari implementasi Program_couter dan pipeline

3. Data Hazard

Penggunaan pipeline mengizinkan akses ke register file secara bersamaan di 2 stage yaitu Execute dan Write Back.

akses yang dilakukan saat Execute dan akses baru pada Decode akan diperiksa. Jika akses yang dilakukan akan mengakibatkan penulisan ke register yang sama atau bergantung pada output instruksi tersebut maka pipeline akan di stall untuk menunggu instruksi pada stage Execute di selesaikan. Stelah instruksi pada Stage Execute di selesaikan maka instruksi pada Decode bisa masuk ke stage selanjutnya

Status : Unresolved pada Processor, Resolved dengan memberikan jarak pada Instruksi yang dapat terjadi data hazard sebanyak PC+(5x4)