Percobaan III
SYNTHESIZABLE MIPS32® MICROPROCESSOR BAGIAN I : INSTRUCTION SET, REGISTER, DAN MEMORY

Ahmad Aziz (13220034)
Asisten: Syafiyatulqulub Soka Nugroho (13219056)
Tanggal Percobaan: 28/10/2022
EL3111 Praktikum Arsitektur Sistem Komputer
Laboratorium Sinyal dan Sistem – Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung

***Abstrak*— Pada praktikum modul 3 yaitu SYNTHESIZABLE MIPS32® MICROPROCESSOR BAGIAN I : INSTRUCTION SET, REGISTER, DAN MEMORY dilakukan untuk mewahami aristektur prosesor MIPS32 dan juga datapath pada eksekusinya. Pada praktikum ini juga dilakukan percobaan untuk memahami instruction set dari MIPS32® dan dapat membuat program sederhana. Dilakukan juga simulasi eksekusi program MIPS32® pada program simulasi SPIM dan memahami cara setiap instruksi dieksekusi. Serta, pada praktikum ini juga akan membuat t instruction memory, data memory dan register dari MIPS32® dalam kode VHDL yang synthesizable dan dapat disimulasikan dengan Altera® Quartus® II v9.1sp2. Pada praktikum ini percobaan dilakukan dengan Altera® Quartus® II v9.1sp2 Web Edition atau Altera® Quartus® II v9.1sp2 Subscription Edition untuk simulasi, PCSpim atau QtSpim sebagai simulator MIPS32® serta notepad++ dan VScode sebagai code editor.**

Kata Kunci—MIPS32, register, VHDL.

I. PENDAHULUAN

Pada praktikum modul 2 yaitu Synthesizable Mips32® Microprocessor Bagian I : Instruction Set, Register, Dan Memory dilakukan sebanyak 4 percobaan yang bertujuan diantaranya sebagai berikut:

- Praktikan memahami arsitektur mikroprosesor MIPS32® beserta datapath eksekusinya.
- Praktikan memahami instruction set dari MIPS32® dan dapat membuat program sederhana dalam bahasa assembly yang dapat dieksekusi pada MIPS32®.
- Praktikan dapat melakukan simulasi eksekusi program MIPS32® pada program simulasi SPIM dan memahami cara setiap instruksi dieksekusi.
- Praktikan dapat membuat instruction memory, data memory dan register dari MIPS32® dalam kode VHDL yang synthesizable dan dapat disimulasikan dengan Altera® Quartus® II v9.1sp2.

Praktikum pada modul ini menggunakan software Quartus v9 untuk membuat percobaan arsitektur dan melakukan simulasi. Ada beberapa topik yang dibahas pada modul praktikum ini yaitu sebagai berikut:

a. Bahasa VHDL
b. Mikroprosesor MIPS32®.
c. Instruction Set dan Register Mikroprosesor MIPS32®.
d. Simulasi MIPS32® menggunakan PCSpim.

Dalam melakukan percobaan dan analisis pada praktikum modul ini, perangkat lunak dan alat yang digunakan adalah sebagai berikut:

a. Quartus v9
b. PCSpim
c. Code editor Visual Studio Code

II. LANDASAN TEORETIS

A. Bahasa VHDL

VHDL (Very High Speed Integrated Circuit Hardware Description Language) atau VHSIC Hardware Description Language merupakan bahasa untuk mendeskripsikan perangkat keras yang digunakan dalam desain elektronik digital dan mixed-signal, contohnya Field-Programmable Gate Array (FPGA) atau Integrated Circuit (IC). Sistem digital sangat erat kaitannya dengan sinyal. Sinyal dapat dianalogikan sebagai wire dan dapat berubah ketika input berubah. Dalam VHDL, terdapat definisi sinyal bernama std_logic yang sesuai dengan standar IEEE 1164. Terdapat sembilan jenis nilai sinyal yang didefinisikan dalam std_logic. Untuk menggunakan nilai sinyal standar std_logic, kita dapat menggunakan library yang telah tersedia yaitu ieee.std_logic_1164.all. Operator Bitwise dalam Bahasa C

Bahasa C mendukung pengolahan informasi dalam level bit menggunakan operator bitwise. Berbeda dengan operator level byte, operator bitwise akan mengoperasikan data untuk setiap bit. Sedangkan operator level byte, data akan diolah dalam bentuk 1 byte (1 byte = 8 bit). Operator bitwise dapat digunakan pada berbagai tipe data seperti char, int, short, long, atau unsigned. Operator-operator bitwise dalam bahasa C didefinisikan sebagai berikut.

Simbol	Arti
U	Unknown
X	Forcing Unknown
0	Forcing 0
1	Forcing 1
Z	High Impedance
W	Weak Unknown
L	Weak 0
H	Weak 1
-	Don't Care

Tidak seperti bahasa Verilog HDL, VHDL merupakan bahasa yang case insensitive. Abstraksi utama dalam bahasa VHDL disebut entitas desain (design entity) yang terdiri atas input, output, dan fungsi yang didefinisikan secara benar. Entitas desain dalam VHDL terdiri atas dua bagian.

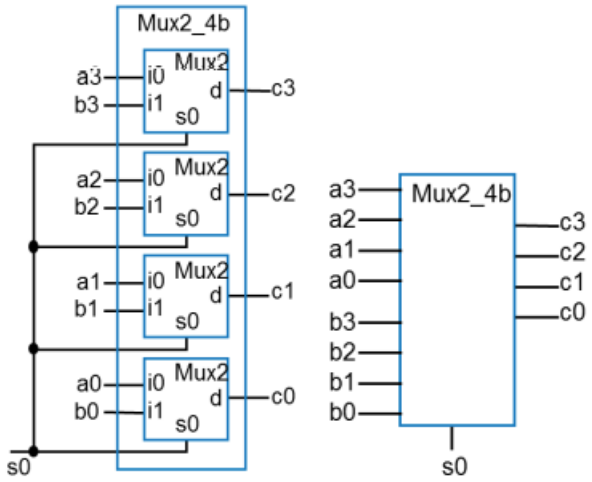
- Deklarasi Entitas (entity declaration) yang mendefinisikan antarmuka entitas tersebut terhadap dunia luar (contohnya port input dan port output).
- Arsitektur Entitas (entity architecture) yang mendefinisikan fungsi dari entitas (contohnya rangkaian logika di dalam entitas tersebut). Pendefinisian arsitektur dapat dilakukan secara behavioral maupun secara structural.

Setiap entitas desain harus disimpan dalam file VHDL yang terpisah dengan nama file sesuai dengan nama entitas yang dideklarasikan (contohnya nama_entity.vhd). Berikut ini template deklarasi sebuah entitas dan arsitektur entitas tersebut dalam bahasa VHDL.

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.all;
ENTITY <nama_entity> IS
  PORT(
    <nama_port_1> : <tipe_port> STD_LOGIC;
    <nama_port_2> : <tipe_port>
STD_LOGIC_VECTOR(n DOWNTO 0)
    );
END <nama_entity>;
ARCHITECTURE <nama_arsitektur> OF
<nama_entity> IS
BEGIN
  <fungsi yang didefinisikan>
END <nama_arsitektur>;
```

Setiap entitas desain dalam file VHDL yang berbeda dapat dipanggil dan dirangkai menjadi rangkaian yang lebih besar. Hal ini sangat penting dilakukan dalam melakukan desain hardware. Pertama, hardware yang akan didesain harus kita pecah-pecah menjadi komponen-komponen logika yang cukup kecil, contohnya menjadi multiplexer, adder, flip-flop, dan sebagainya. Kemudian, kita mendesain masing-masing komponen logika tersebut dan melakukan simulasi fungsional dan simulasi timing untuk setiap komponen untuk meyakinkan bahwa setiap komponen dapat berfungsi dengan baik. Setelah itu, kita menggabungkan masing-masing komponen untuk membentuk entitas desain yang lebih besar (top level entity).

Langkah pertama dalam membentuk top level entity adalah dengan mendefinisikan top level entity tersebut seperti halnya kita membuat entitas desain biasa. Kemudian, pada arsitektur top level entity, kita memanggil desain entitas lain menggunakan construct component. Construct component ini memiliki isi yang sama persis dengan deklarasi entitas desain yang akan dipanggil oleh top level entity. Kemudian, kita harus melakukan instansiasi masing-masing komponen dan menghubungkan port input dan port output dari masing-masing komponen dengan top level design atau dengan komponen lain.



Contoh berikut digunakan untuk merealisasikan 2-to-1 multiplexer 4-bit dari empat buah 2-to-1 multiplexer 1-bit.

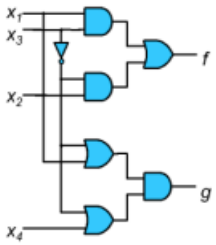
```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY Mux2_4b IS
  PORT (
      A_IN : IN STD_LOGIC_VECTOR (3 DOWNTO
0);
      B_IN : IN STD_LOGIC_VECTOR (3 DOWNTO
0);
      S_IN : IN STD_LOGIC;
      C_OUT : OUT STD_LOGIC_VECTOR (3
DOWNTO 0)
    );
END Mux2_4b;
ARCHITECTURE Structural OF Mux2_4b IS
  COMPONENT Mux2 IS
    PORT (
        A : IN STD_LOGIC;
        B : IN STD_LOGIC;
        S : IN STD_LOGIC;
        D : OUT STD_LOGIC
      );
  END COMPONENT;
BEGIN
  mux2_0 : Mux2
  PORT MAP
    (
    A => A_IN(0),
    B => B_IN(0),
    S => S_IN,
    D => C_OUT(0)
    );
  mux2_1 : Mux2
  PORT MAP
```

```
    (
    A => A_IN(1),
    B => B_IN(1),
    S => S_IN,
    D => C_OUT(1)
    );
  mux2_2 : Mux2
  PORT MAP
    (
    A => A_IN(2),
    B => B_IN(2),
    S => S_IN,
    D => C_OUT(2)
    );
  mux2_3 : Mux2
  PORT MAP
    (
    A => A_IN(3),
    B => B_IN(3),
    S => S_IN,
    D => C_OUT(3)
    );
END Structural;
```

Terdapat tiga jenis concurent signal assignment (CSA) dalam bahasa VHDL, yaitu simple CSA, conditional CSA, dan selected CSA. Ketiga jenis concurent signal assignment tersebut dijelaskan menggunakan contoh sebagai berikut.

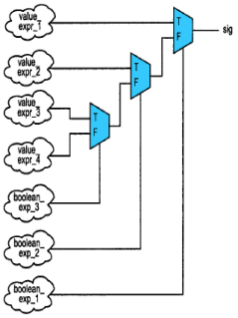
- Simple CSA. Assignment sinyal dilakukan dengan ekspresi logika biasa. Hasil implementasi Simple CSA akan berupa gerbang logika biasa.

```
ARCHITECTURE Behavioral OF Example IS
  BEGIN
    f <= (x1 AND x3) OR (NOT x3 AND x2);
    g <= (NOT x3 OR x1) AND (NOT x3 OR
x4);
END Behavioral;
```



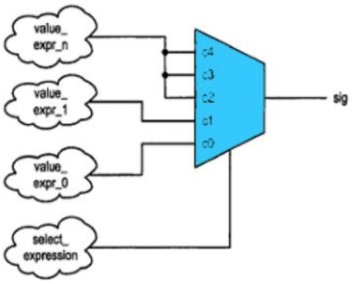
- Conditional CSA. Assignment sinyal dilakukan dengan construct WHEN-ELSE. Hasil implementasi Conditional CSA akan berupa kumpulan 2-to-1 multiplexer yang disusun secara bertahap dengan boolean_expr sebagai selektor dan value_expr sebagai nilai sinyal yang dapat dipilih.

```
signal_name <= value_expr_1 WHEN
boolean_expr_1 ELSE
  value_expr_2 WHEN boolean_expr_2 ELSE
  value_expr_3 WHEN boolean_expr_3 ELSE
  ...
    value_expr_n;
```



- Selected CSA. Assignment sinyal dilakukan dengan construct WITH-SELECT. Hasil implementasi Selected CSA akan berupa satu buah n-to-1 multiplexer dengan select_expression sebagai selektor dan value_expr_3 sebagai nilai sinyal yang dapat dipilih.

```
WITH select_expression SELECT
  signal_name <= value_expr_1 WHEN
choice_1,
    value_expr_2 WHEN choice_2,
    value_expr_3 WHEN choice_3,
    ...
    value_expr_n WHEN OTHERS;
```



Selain concurent signal assignment, dalam bahasa VHDL juga dikenal dengan construct PROCESS yang berfungsi melakukan assignment sinyal secara se kuensial. Sebuah proses (PROCESS) dapat dianalogikan sebagai bagian dari rangkaian yang dapat aktif dan dapat nonaktif. Sebuah proses akan diaktifkan ketika sinyal-sinyal (SIGNAL) dalam daftar sensitivitas (sensitivity list) mengalami perubahan nilai. Ketika diaktifkan, semua ekspresi dan pernyataan (statement) akan dieksekusi secara sekuensial hingga akhir dari proses tersebut.

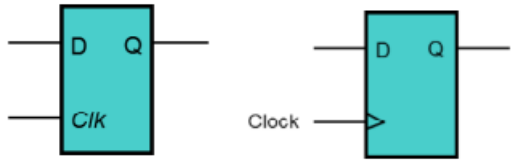
```
PROCESS (sensitivity_list)
  declarations;
  BEGIN
    sequential_statement_1;
    sequential_statement_2;
    ...
    sequential_statement_n;
END PROCESS;
```

Terdapat dua jenis construct yang digunakan dalam construct PROCESS, yaitu construct IF-THEN-ELSE dan construct CASE. Kedua jenis construct tersebut diberikan sebagai contoh berikut ini.

```
ARCHITECTURE Behavioral OF mux2to1 IS
  BEGIN
    PROCESS (w0, w1, s)
      BEGIN
        IF s = '0' THEN
          f <= w0;
        ELSE
          f <= w1;
        END IF;
    END PROCESS;
END Behavioral;
```

```
ARCHITECTURE Behavioral OF mux2to1 IS
  BEGIN
    PROCESS (w0, w1, s)
        BEGIN
          CASE s IS
            WHEN '0' =>
              f <= w0;
            WHEN OTHERS =>
              f <= w1;
          END CASE;
    END PROCESS;
END Behavioral;
```

Dalam bahasa VHDL, kita juga dapat mendefinisikan beberapa jenis elemen memory. Dua jenis elemen memory yang sering digunakan dalam bahasa VHDL adalah Gated D Latch dan D Flip-flop. Gated D Latch memiliki karakteristik yaitu output akan berubah mengikuti input saat clock high (atau clock low, tergantung implementasi). Sedangkan D Flip-flop memiliki karakteristik yaitu output akan berubah mengikuti input saat transisi clock dari low ke high (atau high ke low, tergantung implementasi). Untuk elemen memory lain seperti Gated S-R Latch, T Flip-flop, dan JK Flip-flop juga dapat diimplementasikan pada bahasa VHDL namun mereka jarang digunakan.



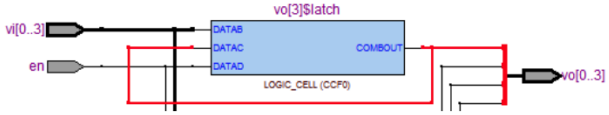

```
ENTITY latch IS
  PORT (
    D : IN STD_LOGIC;
    Clk : IN STD_LOGIC;
    Q : OUT STD_LOGIC
  );
END latch;
```

```
ARCHITECTURE Behavioral OF latch IS
  BEGIN
    PROCESS ( D, Clk )
      BEGIN
      IF Clk = '1' THEN
        Q <= D;
      END IF;
  END PROCESS;
END Behavioral;
```

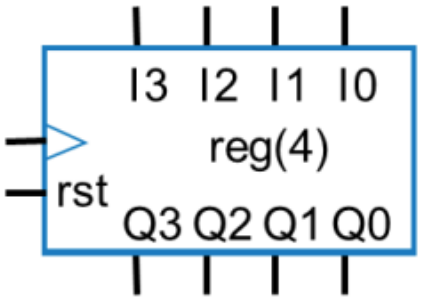
```
ENTITY flipflop IS
  PORT (
  D : IN STD_LOGIC ;
  Clk : IN STD_LOGIC ;
  Q : OUT STD_LOGIC
  );
END flipflop ;
```

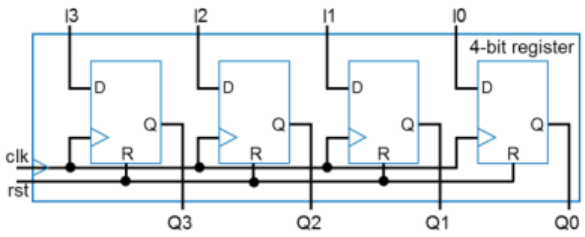
```
ARCHITECTURE Behavior OF flipflop IS
  BEGIN
    PROCESS ( Clock )
    BEGIN
      IF Clock'EVENT AND Clock='1' THEN
      Q <= D ;
      END IF ;
  END PROCESS ;
END Behavior ;
```

Penggunaan Latch dalam implementasi rangkaian menggunakan bahasa VHDL sebaiknya dihindari kecuali kita mengetahui apa yang kita lakukan. Dalam proses sintesis, implementasi Latch ini akan memberikan kita warning. Sebagian besar perangkat FPGA milik Altera tidak memiliki elemen dasar berupa Latch. Dengan demikian sebuah Latch harus dibuat menggunakan Logic Cell. Sayangnya, hal ini membutuhkan sebuah feedback pada Logic Cell untuk mengimplementasikan fungsi memory. Hal ini akan menyebabkan analisis timing statis tidak dapat dilakukan.



Salah satu komponen memory yang paling sering digunakan adalah register. Register terdiri atas beberapa buah flip-flop yang disusun sedemikian rupa sehingga membentuk elemen penyimpanan. Register juga dipakai untuk mengimplementasikan rangkaian sekuensial contohnya finite state machine.





B. Altera® Quartus® II

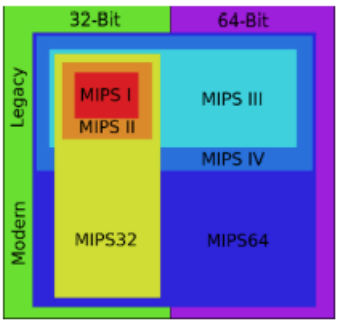
Pada modul praktikum ini tidak akan dibahas terlalu dalam cara-cara melakukan simulasi pada Altera® Quartus® II karena diasumsikan praktikan telah memperoleh pengalaman menggunakan program ini baik untuk simulasi fungsional dan simulasi timing saat mengambil Praktikum Sistem Digital pada tingkat II. Versi Altera® Quartus® II yang disarankan untuk digunakan dalam praktikum ini adalah Altera® Quartus® II v9.1sp2 karena pada versi ini terdapat simulator fungsional dan timing yang telah terintegrasi. Versi Altera® Quartus® II yang lebih baru tidak terdapat simulator fungsional dan timing sehingga praktikan harus menggunakan Mentor Graphics® ModelSim® untuk melakukan simulasi.

Langkah pertama untuk menggunakan Altera® Quartus® II adalah membuat project terlebih dahulu. Untuk membuat project, gunakan new project wizard kemudian ikuti petunjuk-petunjuk yang ada. Beri lokasi dan nama project yang diinginkan. Pilih dokumen-dokumen yang akan dimasukkan ke dalam project (kita dapat melewati langkah ini terlebih dahulu). Kemudian pilih device yang akan digunakan. Untuk praktikum ini, kita tidak akan melakukan implementasi pada FPGA karena praktikum ini hanya berupa simulasi saja. Oleh karena itu, kita dapat memilih FPGA dengan spesifikasi tertinggi baik untuk Altera® Cyclone™ maupun Altera® Stratix™. Setelah project dibuat, kita dapat mulai bekerja di dalamnya.

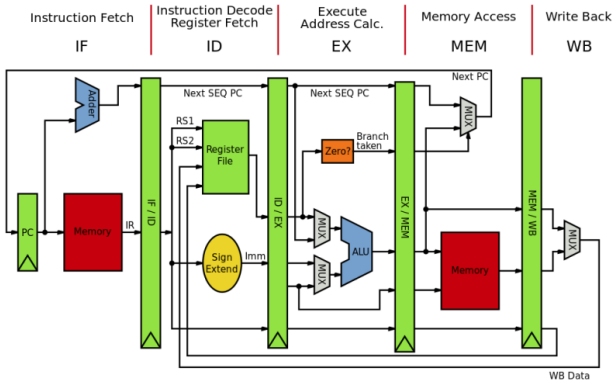
Untuk melakukan simulasi, kita harus melakukan kompilasi terhadap project yang kita buat. Kompilasi yang dilakukan bisa kompilasi penuh maupun hanya Analysis & Synthesis saja. Kompilasi penuh akan memakan waktu yang lebih lama karena semua proses meliputi Analysis & Synthesis, Fitter, dan Assembler akan dilakukan. Kompilasi penuh ini akan memberi kita gambaran terutama dari sisi timing analysis. Sedangkan dengan Analysis & Synthesis, kita telah mendapat rangkaian yang kita buat dan dapat dilakukan simulasi fungsional.

C. Mikroprosesor MIPS32®

MIPS32® (Microprocessor without Interlocked Pipeline Stages) merupakan sebuah mikroprosesor 32-bit yang dikembangkan oleh MIPS Technologies. Mikroprosesor ini merupakan reduced instruction set computer (RISC). Mikroprosesor ini sering digunakan sebagai bahan pembelajaran mata kuliah Arsitektur Sistem Komputer diberbagai universitas dan sekolah teknik.



Dalam kehidupan nyata, arsitektur mikroprosesor MIPS® sering digunakan dalam sistem embedded seperti perangkat Windows™ CE, router, residential gateway, dan konsol video game seperti Sony® PlayStation®, Sony® PlayStation® 2 (PS2™), dan Sony® PlayStation® Portable (PSP®).

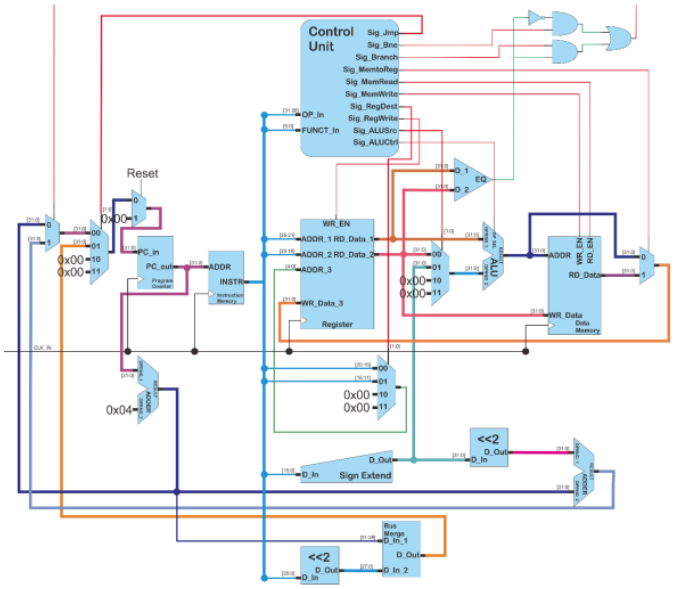


Terdapat lima tahap yang dilakukan ketika mikroprosesor MIPS32® melakukan eksekusi suatu instruksi. Kelima tahap tersebut adalah sebagai berikut.

- Instruction Fetch (IF)
  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 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. Dengan demikian, arsitektur ini menganut Harvard Architecture.
- Instruction Decode (ID)
  Tahap berikutnya, instruksi yang telah diambil (fetched) dari instruction memory berpindah ke tahap instruction decode. Pada tahap ini, instruksi dengan lebar 32-bit akan dipecah sesuai format instruksi yang digunakan. Penjelasan mengenai decoding instruksi ini dapat dilihat pada bagian selanjutnya.
- Execute / Address Calculation (EX)
  Tahap ini merupakan tahap sebagian besar operasi aritmatika dan logika pada arithmetic and logical unit (ALU) dilakukan. Pada tahap ini juga terdapat tempat untuk meneruskan alamat register kembali ke tahap instruction decode sebagai deteksi hazard.
- Data Memory (MEM)
  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.
- Write Back (WB)
  Tahap terakhir ini digunakan untuk mengalirkan data dari data memory atau hasil perhitungan arithmetic and logical unit (ALU) ke register untuk dapat menjalankan instruksi selanjutnya.

Dalam praktikum ini, kita akan melakukan implementasi mikroprosesor MIPS32® yang sederhana. Mikroprosesor MIPS32® yang akan diimplementasikan tidak memiliki pipeline dan semua instruksi selesai dieksekusi dalam satu siklus clock. Dengan demikian, kita akan membuat mikroprosesor Single-Cycle MIPS32® menggunakan bahasa VHDL yang synthesizable. Diagram arsitektur mikroprosesor

Single-Cycle MIPS32® yang akan kita buat diberikan sebagai berikut.



D. *Instruction Set dan Register Mikroprosesor MIPS32®*

Mikroprosesor MIPS32® memiliki set instruksi yang sederhana dibandingkan dengan mikroprosesor milik Intel®. Sebelum kita melangkah lebih jauh untuk melihat instruksi-instruksi dasar pada MIPS32®, kita perlu melihat register yang tersedia pada MIPS32®. Terdapat 32 buah register pada MIPS32® yang masing-masing register memiliki kegunaannya masing-masing. Semua register pada MIPS32® dapat diakses menggunakan address dengan lebar 5-bit. Tabel berikut merupakan daftar register yang tersedia dalam MIPS32® beserta fungsinya masing-masing.

Nama *Register*	Alamat	Fungsi
`$zero`	0	Nilai konstan 0
`$at`	1	Penyimpan Sementara Assembler
`$v0-$v1`	2-3	Penyimpan nilai dari hasil fungsi dan penyelesaian ekspresi
`$a0-$a3`	4-7	Penyimpan Argumen
`$t0-$t7`	8-15	Penyimpan sementara
`$s0-$s7`	16-23	Penyimpan sementara untuk pemanggilan fungsi
`$t8-$t9`	24-25	Penyimpan sementara
`$k0-$k1`	26-27	Digunakan oleh Kernel OS
`$gp`	28	*Pointer* global
`$sp`	29	*Pointer stack*
`$fp`	30	*Pointer frame*
`$ra`	31	*Return Address*

MIPS32® memiliki instruksi dengan lebar 32-bit. Instruksi-instruksi yang dimiliki MIPS32® dapat dilihat lebih lengkap pada lembar lampiran. Terdapat tiga buah format dasar dari instruksi MIPS32®. Ketiga format dasar instruksi tersebut adalah instruksi tipe-R, instruksi tipe-I, dan insruksi tipe-J. Format ketiga instruksi dasar tersebut dapat dilihat pada gambar berikut. Komponen dari ketiga format dasar instruksi tersebut dijelaskan pada tabel selanjutnya.

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	opcode						rs					rt					rd					shamt					funct					
I	opcode						rs					rt					immediate															
J	opcode						address																									

Komponen	Keterangan
`opcode`	Menunjukkan jenis operasi yang akan dilakukan oleh instruksi tersebut. Khusus untuk instruksi tipe-R, opcode selalu bernilai 0x00.
`rs, rt, rd`	Menentukan alamat (nomor) dari *source register* (instruksi tipe-R dan instruksi tipe-I), *temporary register* (instruksi tipe-R dan instruksi tipe-I), dan *destination register* (instruksi tipe-R).
`shamt`	Menunjukkan jumlah penggeseran bit (*shift amount*) pada instruksi tipe-R.
`funct`	Memilih operasi matematika yang akan dilakukan pada instruksi tipe-R
`immediate`	Menentukan nilai konstanta yang menunjukkan *operand* yang konstan atau *address*.
`address`	Alamat tujuan pada *instruction memory* yang akan dieksekusi setelahnya.

Dalam praktikum ini, mikroprosesor Single-Cycle MIPS32® yang akan diimplementasikan harus dapat menjalankan sembilan buah instruksi sebagai berikut.

Instruksi	Tipe	`opcode`	`funct`	Keterangan
`add`	R	`000000`	`100000`	Operasi penjumlahan
`sub`	R	`000000`	`100010`	Operasi pengurangan
`beq`	I	`000100`	`------`	Pencabangan bila sama (*Branch-if-Equal*)
`bne`	I	`000101`	`------`	Pencabangan bila tidak sama (*Branch-if-Not-Equal*)
`addi`	I	`001000`	`------`	Operasi penjumlahan dengan konstanta
`lw`	I	`100011`	`------`	Mengambil data dari *data memory* (*load word*)
`sw`	I	`101011`	`------`	Menyimpan data ke *data memory* (*save word*)
`jmp`	J	`000010`	`------`	Menuju ke instruksi pada *address* tertentu (*jump*)
`nop`	-	`000000`	`000000`	Tidak ada operasi (*no operation*). Digunakan untuk menambahkan jeda satu siklus setelah instruksi *branching* dilakukan.

E. *Simulasi MIPS32® menggunakan PCSpim*

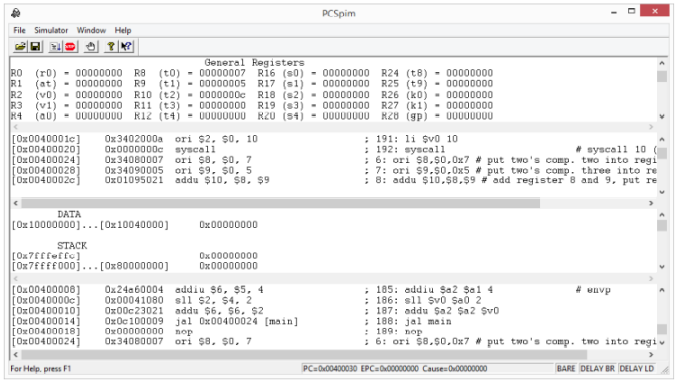
Sebelum kita mendesain mikroprosesor Single-Cycle MIPS32®, ada baiknya kita mempelajari terlebih dahulu bagaimana sebuah instruksi dieksekusi oleh mikroprosesor MIPS32® tersebut. Kita dapat menggunakan sebuah simulator untuk melakukan eksekusi program yang kita buat pada mikroprosesor MIPS32® lalu melihat hasilnya. Simulator MIPS32® yang akan digunakan dalam praktikum ini adalah PCSpim. Simulator PCSpim dapat diunduh di https://praktikum.ee.itb.ac.id/praktikum/el3111/.

Perhatikan bahwa program ini membutuhkan Microsoft® .Net Framework 2.0 untuk dapat berjalan. Bagi praktikan yang menggunakan Microsoft® Windows® XP, Microsoft® Windows® 8, dan Microsoft® Windows® 8.1 harus memasang Microsoft® .Net Framework 2.0 terlebih dahulu. Microsoft® .Net Framework 2.0 dapat diunduh di https://praktikum.ee.itb.ac.id/praktikum/el3111/ dalam paket instalasi Microsoft® .Net Framework 3.5.

Setelah instalasi selesai PCSpim, kita dapat langsung menjalankan PCSpim dari start menu. Khusus untuk pengguna Microsoft® Windows® edisi 64-bit, terkadang PCSpim akan memberikan pesan error karena tidak dapat menemukan file exceptions.s. Untuk mengatasinya, pilih menu Simulator lalu klik submenu Settings. Pada bagian Load exception file, ganti alamat file exceptions.s.

Sebelum : C:\Program Files\PCSpim\exceptions.s

Sesudah : C:\Program Files (x86)\PCSpim\exceptions.s



Jendela dari PCSpim dibagi menjadi empat bagian. Bagian pertama merupakan Register Display yang berisi isi dari setiap register pada MIPS32® meliputi 32 general purpose register dan beberapa floating point register serta beberapa register yang lain. Isi dari setiap register yang ditampilkan dalam format heksadesimal. Bagian kedua merupakan Text Display yang berisi program dalam bahasa assembly, kode instruksi dalam heksadesimal, dan alamat instruksi tersebut. Bagian ketiga merupakan Data and Stack Display yang berisi isi memory dalam MIPS32® yang menampung data-data serta stack. Bagian keempat merupakan SPIM Message yang berisi laporan dari simulator ketika terjadi galat.

Bila dalam program bahasa assembly yang kita buat terdapat perintah untuk menampilkan sesuatu ke layar (mirip dengan printf dalam bahasa C), maka output ke layar tersebut akan ditampilkan dalam jendela konsol termasuk apabila program meminta pengguna memasukkan input. Untuk memulai penggunaan PCSpim pertama kali, Anda akan diminta untuk menjalankan program sederhana.

Buatlah program dalam bahasa assembly dengan menyalin kode program di bawah ini menggunakan teks editor Notepad++. Simpan file tersebut dengan nama add.asm. Kalimat di sebelah kanan tanda # merupakan komentar dan tidak akan dieksekusi oleh simulator. Ubah konfigurasi PCSpim agar menjalankan simulasi menggunakan Bare Machine dengan membuka menu Simulator lalu submenu Settings.

```
# Program untuk menjumlahkan 7 dengan 5
  .text
  .globl main
main:
  ori $8,$0,0x07 # masukkan angka 7 ke
register 8
  ori $9,$0,0x05 # masukkan angka 5 ke
register 9
  addu $10,$8,$9 # jumlahkan dan simpan
hasilnya di register 10
# akhir dari program
```

Buka file add.asm menggunakan PCSpim dengan membuka menu File lalu Open. Bila terjadi kesalahan sintaks dalam pemrograman bahasa assembly, PCSpim akan mengeluarkan pesan galat. Periksa kembali program yang dibuat lalu simpan program tersebut sebelum dibuka kembali menggunakan PCSpim. Bila program berhasil dibuka, kita dapat melihat bahwa file bahasa assembly telah diterjemahkan menjadi instruksi-instruksi dalam bahasa heksadesimal dan disimpan dalam instruction memory.

Untuk memulai eksekusi, kita harus mengeset nilai program counter (PC). Program counter (PC) merupakan bagian dari mikroprosesor yang menyimpan address instruksi yang akan dieksekusi. Pada bagian Register Display, terlihat bahwa PC bernilai 0x00000000. Ubah nilai PC tersebut menjadi 0x00400000 dengan membuka menu Simulator, lalu submenu Set Value. Tuliskan PC pada kotak isian paling atas dan 0x00400000 pada kotak isian paling bawah. Hal ini dilakukan karena program yang kita buat dimulai pada address tersebut.

Tekan tombol F10 pada keyboard untuk melakukan eksekusi satu instruksi. Tekan tombol F10 hingga instruksi dari program yang kita tulis dapat dieksekusi. Perhatikan bahwa saat instruksi pertama dilakukan, nilai register 8 berubah menjadi 0x07 dan PC berubah menjadi 0x00400004. Tekan kembali tombol F10 pada keyboard untuk melakukan eksekusi satu instruksi berikutnya dan perhatikan yang terjadi pada register 9. Tekan kembali tombol F10 pada keyboard untuk melakukan eksekusi satu instruksi berikutnya dan perhatikan yang terjadi pada register 10. Hasil penjumlahan kedua bilangan tersebut disimpan pada register 10.

III. Hasil dan Analisis

Setelah melakukan percobaan pada semua tugas didapatkan hasil sebagai brerikut:

A. Tugas 1: Perancangan Instruction Memory

Pada tugas 1 dilakukan percobaan untuk membuat instruction memory untuk arsitektur MIPS dengan Quartus. instruction memory memiliki lebar data sebesar 32-bit dan lebar address sebesar 32-bit. Hanya 32 address paling awal saja yang dipakai. Instruction memory memiliki 54 sebuah port input yang menerima address dengan lebar 32-bit dan sebuah port output yang mengeluarkan instruksi dengan lebar data 32-bit. Terdapat pula port input clock untuk mengendalikan rangkaian ini.

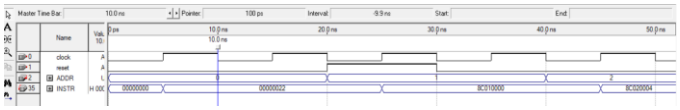
Berikut ini adalah kode VHDL untuk instruction memory yang dibuat yang akan di simulasikan:

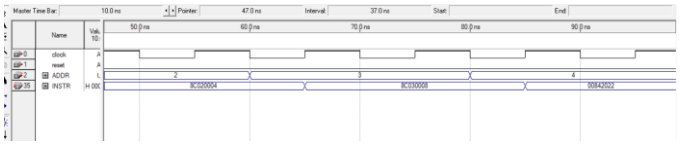
```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;
LIBRARY altera_mf;
USE altera_mf.altera_mf_components.ALL;

ENTITY instrucMEM IS
  PORT (
     ADDR : IN std_logic_vector (31
DOWNTO 0);
     clock : IN std_logic;
     reset : IN std_logic;
     INSTR : OUT std_logic_vector (31
DOWNTO 0)
   );
END ENTITY;

ARCHITECTURE behavior OF instrucMEM IS
TYPE ramtype IS ARRAY (31 DOWNTO 0) OF
std_logic_vector (31 DOWNTO 0);
```

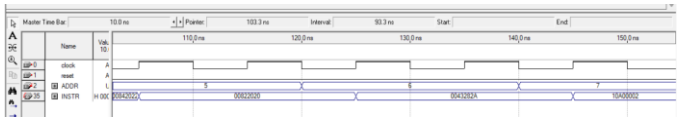
```
SIGNAL mem: ramtype;
  BEGIN
    PROCESS (clock,reset)
    BEGIN
      IF (reset='1') THEN
        INSTR <= (OTHERS => '0');
      ELSIF (rising_edge(clock)) THEN
        INSTR <=
mem(to_integer(unsigned(ADDR)));
      END IF;
    END PROCESS;

-- Isi dalam instruction memory
mem(0) <= X"00000022";
mem(1) <= X"8c010000";
mem(2) <= X"8c020004";
mem(3) <= X"8c030008";
mem(4) <= X"00842022";
mem(5) <= X"00822020";
mem(6) <= X"0043282a";
mem(7) <= X"10a00002";
mem(8) <= X"00411020";
mem(9) <= X"1000fffb";
mem(10) <= X"ac040000";
mem(11) <= X"1000ffff";
  END behavior;
```

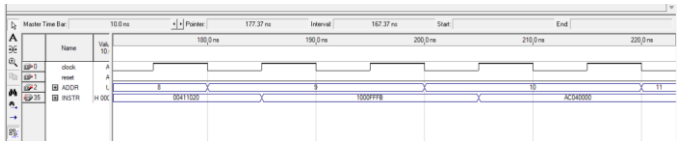
Pada program tersebut dapat dilihat bahwa setiap memory sudah kita assign dengan nilai tertentu.

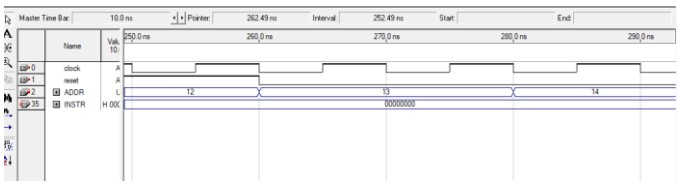
Berikut ini adalah hasil simulasi program VHDL untuk instruction memory diatas:











Pada hasil simulasi tersebut dapat dilihat pada output INSTR outputnya sudah sesuai dengan alamat yang kita masukkan pada ADDR. Outputnya pun sama dengan program VHDL dimana alamat di assign.

```
mem(0) <= X"00000022";
mem(1) <= X"8c010000";
mem(2) <= X"8c020004";
mem(3) <= X"8c030008";
mem(4) <= X"00842022";
mem(5) <= X"00822020";
mem(6) <= X"0043282a";
mem(7) <= X"10a00002";
mem(8) <= X"00411020";
mem(9) <= X"1000fffb";
mem(10) <= X"ac040000";
mem(11) <= X"1000ffff";
```

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

Pada percobaan tugas 2 ini akan memanfaatkan sebuah template desain yang telah tersedia dalam Altera® Quartus® II yaitu Altera® MegaFunction ALTSYNCRAM. Template ini dapat digunakan untuk merealisasikan synchronous RAM dan ROM dalam desain kita. Selain itu, kita dapat menggunakan inisialisasi isi memory dari file eksternal berformat .mif (memory initialization file).

Untuk merealisasikannya, terlebih dahulun include file mifnya dengan cara deklarasi berikut berikut:

```
  init_file : STRING; -- name of the .mif
file
```

Kemudian pada implementasinya sebagai berikut:

```
init_file => "imemory.mif",
```

Ukuran bit pada program ini juga disesuikan menjadi 8 bit saja sehingga kode programnya perlu dimosdifikasi. Berikut ini adalah kode VHDL lengkap yang digunakan:

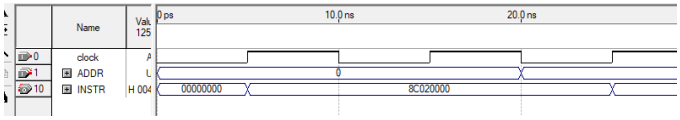
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
LIBRARY altera_mf;
USE altera_mf.all;

ENTITY instruction_memory IS
  PORT (
      ADDR : IN STD_LOGIC_VECTOR (7 DOWNTO
0); -- alamat
      clock : IN STD_LOGIC := '1'; --
clock
      INSTR : OUT STD_LOGIC_VECTOR (31
DOWNTO 0) -- output
    );
END ENTITY;

ARCHITECTURE structural OF
instruction_memory IS
```

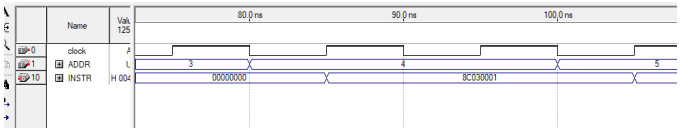
```
SIGNAL sub_wire0 : STD_LOGIC_VECTOR (31
DOWNTO 0);
-- signal keluaran output
COMPONENT altsyncram
-- komponen memori
  GENERIC
  (
    init_file : STRING; -- name of the
.mif file
    operation_mode : STRING; -- the
operation mode
    widthad_a : NATURAL; -- width of
address_a[]
    width_a : NATURAL -- width of data_a[]
  );

  PORT
  (
    clock0 : IN STD_LOGIC ;
    address_a : IN STD_LOGIC_VECTOR (7
DOWNTO 0);
    q_a : OUT STD_LOGIC_VECTOR (31 DOWNTO
0)
  );
END COMPONENT;

BEGIN
  INSTR <= sub_wire0;
  altsyncram_component : altsyncram
  GENERIC MAP
  (
    init_file => "imemory.mif",
    operation_mode => "ROM",
    widthad_a => 8,
    width_a => 32
  )
  PORT MAP
  (
    clock0 => clock,
    address_a => ADDR,
    q_a => sub_wire0
  );
  END structural;
```

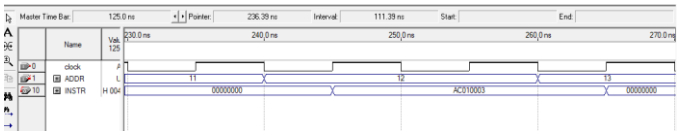
Kemudian buat file mif yang akan digunakan:

```
-- Praktikum EL3111 Arsitektur Sistem
Komputer
-- Modul : 4
-- Percobaan : 1
-- Tanggal : 28 Oktober 2022
-- Kelompok : 10
-- Rombongan : B
-- Nama (NIM) 1 : Gilbert Ng (13220032)
-- Nama (NIM) 2 : Ahmad Aziz (13220034)
-- Nama File : imemory.mif

WIDTH=32; -- number of bits of data per
word
DEPTH=256; -- the number of addresses
ADDRESS_RADIX=HEX;
DATA_RADIX=HEX;
CONTENT
BEGIN
  00 : 8c020000;
  04 : 8c030001;
  08 : 00430820;
  0C : ac010003;
  10 : 1022ffff;
  14 : 1021fffa;
  06 : 0043282a;
  07 : 10a00002;
  09 : 1000fffb;
  10 : ac040000;
  11 : 1000ffff;
END;
```

Berikut adalah hasil simulasi dari program dengan menggunakan mif file terpisah:







Dapat dilihat pada hasil simulasi semua nilai yang diassign pada file mif muncul pada alamat yang sesuai. Sehingga pembacaan dan penggunaan file mif sudah berhasil dan berjalan dengan baik.

Untuk sistem blok juga berjalan dengan baik dimana pembacaan dilakukan pada raising clock

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

Pada percobaan tugas 3 ini akan membuat Data memory untuk arsitektur MIPS32.

Berikut ini adalah kode VHDL yang digunakan dan sudah disesuaikan:

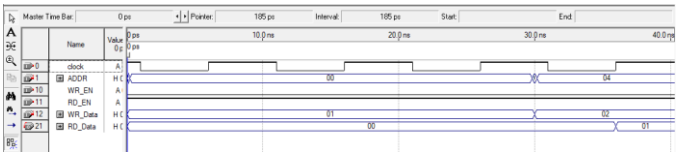
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
LIBRARY altera_mf;
```

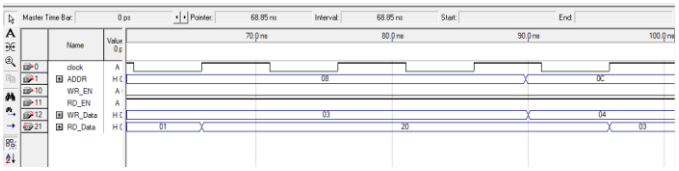
```
USE altera_mf.all;

ENTITY data_memory IS
  PORT (
      ADDR : IN STD_LOGIC_VECTOR (7 DOWNTO
0); -- alamat
      WR_EN : IN STD_LOGIC; --Indikator
Penulisan
      RD_EN : IN STD_LOGIC; --Indikator
Pembacaan
      clock : IN STD_LOGIC := '1'; --
clock
      RD_Data : OUT STD_LOGIC_VECTOR (7
DOWNTO 0);
      WR_Data : IN STD_LOGIC_VECTOR (7
DOWNTO 0)
    );
END ENTITY;

ARCHITECTURE structural of data_memory IS

COMPONENT altsyncram
-- komponen memori
GENERIC
  (
    init_file : STRING; -- name of the
.mif file
    operation_mode : STRING; -- the
operation mode
    widthad_a : NATURAL; -- width of
address_a[]
    width_a : NATURAL -- width of data_a[]
  );

PORT
  (
    wren_a : IN STD_LOGIC; -- Write Enable
Activation
    rden_a : IN STD_LOGIC; -- Read Enable
Activation
    clock0 : IN STD_LOGIC; -- Clock
    address_a : IN STD_LOGIC_VECTOR (7
DOWNTO 0); -- Address Input
    q_a : OUT STD_LOGIC_VECTOR (7 DOWNTO
0); -- Data Output
    data_a : IN STD_LOGIC_VECTOR (7 DOWNTO
0) -- Data Input
  );
END COMPONENT;

BEGIN
altsyncram_component : altsyncram
  GENERIC MAP
  (
    init_file => "dmemory.mif",
    operation_mode => "SINGLE_PORT",
    widthad_a => 8,
    width_a => 8
  )
  PORT MAP
  (
    wren_a => WR_EN,
    rden_a => RD_EN,
    clock0 => clock,
    address_a => ADDR,
    q_a => RD_Data,
    data_a => WR_Data
  );
END structural;
```

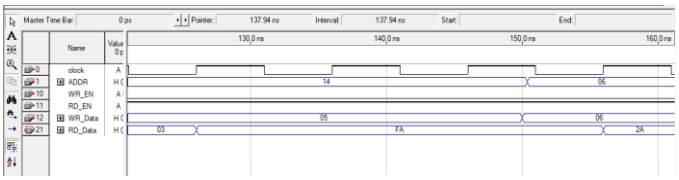
Isi data memory diambil dari file mif denga nisi sebgai berikut:

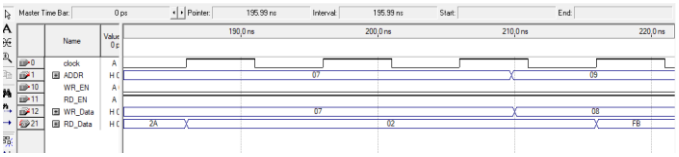
```
  00 : 8c020000;
  04 : 8c030001;
  08 : 00430820;
  0C : ac010003;
  10 : 1022ffff;
  14 : 1021fffa;
  06 : 0043282a;
  07 : 10a00002;
  09 : 1000fffb;
  10 : ac040000;
  11 : 1000ffff;
```

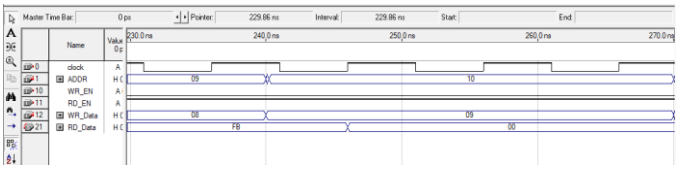
Berikut ini adalah hasil simulasi dari data memory yang telah dibuat:

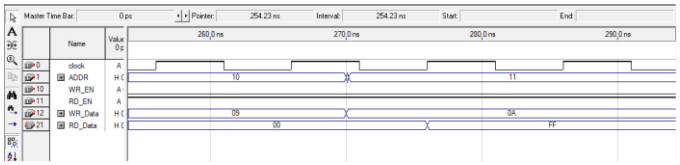


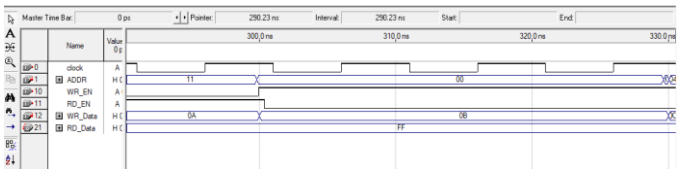












Pada data memory ini, hanya menggunakan 3 byte sehingga hanya 2 byte LSB data saja yang muncul pada saat disimulasikan sehingga hasil simulasi ini sudah sesai dan berjalan dengan semestinya.

D. Tugas 4: Perancangan Register

Pada percobaan tugas ke 4 ini akan membuat perancangan register untuk arsitektur MIPS32. Register ini memiliki dua buah port input untuk memasukkan address 32-bit dari data yang akan dibaca dan memiliki satu buah port input untuk memasukkan address 32-bit tempat data akan ditulis. Selain itu terdapat dua buah port output tempat keluarnya data 32-bit yang dibaca dan satu buah port input tempat masuknya data 32-bit yang akan ditulis. Terdapat pula port input untuk clock dan port input untuk sinyal untuk mengaktifkan mode tulis (write enable). Penggunaan urutan register ini sesuai dengan tabel register MIPS32® pada landasan teoretis praktikum. Perhatikan bahwa nilai register 0 harus tetaplah nol.

Berikut ini adalah kode VHDL yang dibuat untuk realisasi dan simulasi register pada quartus:

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE IEEE.NUMERIC_STD.ALL;
LIBRARY altera_mf;
USE altera_mf.altera_mf_components.ALL;

ENTITY reg_file IS
  PORT (
    clock : IN STD_LOGIC; -- clock
    WR_EN : IN STD_LOGIC; -- write enable
    ADDR_1 : IN STD_LOGIC_VECTOR (4 DOWNTO
0); -- Input 1
    ADDR_2 : IN STD_LOGIC_VECTOR (4 DOWNTO
0); -- Input 2
    ADDR_3 : IN STD_LOGIC_VECTOR (4 DOWNTO
0); -- Input 3
    WR_Data_3 : IN STD_LOGIC_VECTOR (31
DOWNTO 0);-- write data
    RD_Data_1 : OUT STD_LOGIC_VECTOR(31
DOWNTO 0);-- read data 1
    RD_Data_2 : OUT STD_LOGIC_VECTOR(31
DOWNTO 0) -- read data 2
    );
END ENTITY;

ARCHITECTURE behavior OF reg_file IS
TYPE ramtype IS ARRAY (31 DOWNTO 0) OF
STD_LOGIC_VECTOR(31 DOWNTO 0);
  SIGNAL mem: ramtype;

BEGIN
  PROCESS (clock, WR_EN, ADDR_1, ADDR_2,
ADDR_3, mem)
  BEGIN
    IF (rising_edge(clock) AND WR_EN =
'1') THEN
      mem(conv_integer(ADDR_3)) <=
WR_Data_3;
    ELSIF (falling_edge(clock)) THEN
      RD_DATA_1 <=
mem(conv_integer(ADDR_1));
      RD_DATA_2 <=
mem(conv_integer(ADDR_2));
    END IF;
  END PROCESS;
  END behavior;
```

IV. SIMPULAN

- Arsitektur MIPS32 mengeksekusi perintah dalam lima tahapan yaitu Instruction Fetch, Instruction Decode, Execute, Data Memory, dan Write Back.
- Inisiasi memory VHDL dapat dilakukan pada file external .mif

REFERENSI

[1] Bryant, Randal, dan David O'Hallaron. *Computer Systems: A Programmer's Perspective 2nd Edition*. 2011. Massachusetts: Pearson Education Inc.

[2] Patterson, David, dan John Hennessy. *Computer Organization and Design: The Hardware/Software Interface*. 2012. Waltham: Elsevier Inc.

[3] Kernighan, Brian, dan Dennis Ritchie. The C Programming Language 2nd edition. 1988. Englewood Cliffs : Prentice Hall.

Lampiran

1) *Source code untuk tugas 1*

instrucMEM.vhd

```
-- Praktikum EL3111 Arsitektur Sistem Komputer
-- Modul : 4
-- Percobaan : 1
-- Tanggal : 18 November 2013
-- Kelompok : 10
-- Rombongan : B
-- Nama (NIM) 1 : Gilbert Ng (13220032)
-- Nama (NIM) 2 : Ahmad Aziz (13220034)
-- Nama File : instrucMEM.vhd
-- Deskripsi : Implementasi instruction memory

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;
LIBRARY altera_mf;
USE altera_mf.altera_mf_components.ALL;

ENTITY instrucMEM IS
  PORT (
      ADDR : IN std_logic_vector (31 DOWNTO 0);
      clock : IN std_logic;
      reset : IN std_logic;
      INSTR : OUT std_logic_vector (31 DOWNTO 0)
    );
END ENTITY;

ARCHITECTURE behavior OF instrucMEM IS
TYPE ramtype IS ARRAY (31 DOWNTO 0) OF std_logic_vector (31 DOWNTO 0);
SIGNAL mem: ramtype;
  BEGIN
    PROCESS (clock,reset)
    BEGIN
      IF (reset='1') THEN
        INSTR <= (OTHERS => '0');
      ELSIF (rising_edge(clock)) THEN
        INSTR <= mem(to_integer(unsigned(ADDR)));
      END IF;
    END PROCESS;

-- Isi dalam instruction memory
mem(0) <= X"00000022";
mem(1) <= X"8c010000";
mem(2) <= X"8c020004";
mem(3) <= X"8c030008";
mem(4) <= X"00842022";
mem(5) <= X"00822020";
mem(6) <= X"0043282a";
mem(7) <= X"10a00002";
mem(8) <= X"00411020";
mem(9) <= X"1000fffb";
mem(10) <= X"ac040000";
mem(11) <= X"1000ffff";
END behavior;
```

2) *Source code untuk tugas 2*

instruction_memory.vhd

```
-- Praktikum EL3111 Arsitektur Sistem Komputer
-- Modul : 4
-- Percobaan : 1
-- Tanggal : 28 Oktober 2022
-- Kelompok : 10
-- Rombongan : B
-- Nama (NIM) 1 : Gilbert Ng (13220032)
-- Nama (NIM) 2 : Ahmad Aziz (13220034)
-- Nama File : instruction_memory.vhd

LIBRARY ieee;
USE ieee.std_logic_1164.all;
LIBRARY altera_mf;
USE altera_mf.all;

ENTITY instruction_memory IS
  PORT (
      ADDR : IN STD_LOGIC_VECTOR (7 DOWNTO 0); -- alamat
      clock : IN STD_LOGIC := '1'; -- clock
      INSTR : OUT STD_LOGIC_VECTOR (31 DOWNTO 0) -- output
     );
END ENTITY;

ARCHITECTURE structural OF instruction_memory IS
SIGNAL sub_wire0 : STD_LOGIC_VECTOR (31 DOWNTO 0);
-- signal keluaran output
COMPONENT altsyncram
-- komponen memori
  GENERIC
  (
    init_file : STRING; -- name of the .mif file
    operation_mode : STRING; -- the operation mode
    widthad_a : NATURAL; -- width of address_a[]
    width_a : NATURAL -- width of data_a[]
  );

  PORT
  (
    clock0 : IN STD_LOGIC ;
    address_a : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
    q_a : OUT STD_LOGIC_VECTOR (31 DOWNTO 0)
  );
END COMPONENT;

BEGIN
  INSTR <= sub_wire0;
  altsyncram_component : altsyncram
  GENERIC MAP
  (
    init_file => "imemory.mif",
    operation_mode => "ROM",
    widthad_a => 8,
    width_a => 32
  )
  PORT MAP
  (
    clock0 => clock,
    address_a => ADDR,
    q_a => sub_wire0
  );
END structural;
```

3) *Source code untuk tugas 3*

data_memory.vhd

```
-- Praktikum EL3111 Arsitektur Sistem Komputer
-- Modul : 4
-- Percobaan : 1
-- Tanggal : 28 Oktober 2022
-- Kelompok : 10
-- Rombongan : B
-- Nama (NIM) 1 : Gilbert Ng (13220032)
-- Nama (NIM) 2 : Ahmad Aziz (13220034)
-- Nama File : data_memory.vhd

LIBRARY ieee;
USE ieee.std_logic_1164.all;
LIBRARY altera_mf;
USE altera_mf.all;

ENTITY data_memory IS
  PORT (
      ADDR : IN STD_LOGIC_VECTOR (7 DOWNTO 0); -- alamat
      WR_EN : IN STD_LOGIC; --Indikator Penulisan
      RD_EN : IN STD_LOGIC; --Indikator Pembacaan
      clock : IN STD_LOGIC := '1'; -- clock
      RD_Data : OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
      WR_Data : IN STD_LOGIC_VECTOR (7 DOWNTO 0)
    );
END ENTITY;

ARCHITECTURE structural of data_memory IS

COMPONENT altsyncram
-- komponen memori
GENERIC
  (
    init_file : STRING; -- name of the .mif file
    operation_mode : STRING; -- the operation mode
    widthad_a : NATURAL; -- width of address_a[]
    width_a : NATURAL -- width of data_a[]
  );

PORT
  (
    wren_a : IN STD_LOGIC; -- Write Enable Activation
    rden_a : IN STD_LOGIC; -- Read Enable Activation
    clock0 : IN STD_LOGIC; -- Clock
    address_a : IN STD_LOGIC_VECTOR (7 DOWNTO 0); -- Address Input
    q_a : OUT STD_LOGIC_VECTOR (7 DOWNTO 0); -- Data Output
    data_a : IN STD_LOGIC_VECTOR (7 DOWNTO 0) -- Data Input
  );
END COMPONENT;

BEGIN
altsyncram_component : altsyncram
  GENERIC MAP
  (
    init_file => "dmemory.mif",
    operation_mode => "SINGLE_PORT",
    widthad_a => 8,
    width_a => 8
  )
  PORT MAP
  (
    wren_a => WR_EN,
    rden_a => RD_EN,
    clock0 => clock,
```

```
    address_a => ADDR,
    q_a => RD_Data,
    data_a => WR_Data
  );
END structural;
```

4) *Source code untuk tugas 4*

reverseByte.h

```
-- Praktikum EL3111 Arsitektur Sistem Komputer
-- Modul : 4
-- Percobaan : 1
-- Tanggal : 28 Oktober 2022
-- Kelompok : 10
-- Rombongan : B
-- Nama (NIM) 1 : Gilbert Ng (13220032)
-- Nama (NIM) 2 : Ahmad Aziz (13220034)
-- Nama File : reg_file.vhd

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE IEEE.NUMERIC_STD.ALL;
LIBRARY altera_mf;
USE altera_mf.altera_mf_components.ALL;

ENTITY reg_file IS
  PORT (
    clock : IN STD_LOGIC; -- clock
    WR_EN : IN STD_LOGIC; -- write enable
    ADDR_1 : IN STD_LOGIC_VECTOR (4 DOWNTO 0); -- Input 1
    ADDR_2 : IN STD_LOGIC_VECTOR (4 DOWNTO 0); -- Input 2
    ADDR_3 : IN STD_LOGIC_VECTOR (4 DOWNTO 0); -- Input 3
    WR_Data_3 : IN STD_LOGIC_VECTOR (31 DOWNTO 0);-- write data
    RD_Data_1 : OUT STD_LOGIC_VECTOR(31 DOWNTO 0);-- read data 1
    RD_Data_2 : OUT STD_LOGIC_VECTOR(31 DOWNTO 0) -- read data 2
    );
END ENTITY;

ARCHITECTURE behavior OF reg_file IS
TYPE ramtype IS ARRAY (31 DOWNTO 0) OF STD_LOGIC_VECTOR(31 DOWNTO 0);
  SIGNAL mem: ramtype;
```

```
BEGIN
  PROCESS (clock, WR_EN, ADDR_1, ADDR_2, ADDR_3, mem)
  BEGIN
    IF (rising_edge(clock) AND WR_EN = '1') THEN
      mem(conv_integer(ADDR_3)) <= WR_Data_3;
    ELSIF (falling_edge(clock)) THEN
      RD_DATA_1 <= mem(conv_integer(ADDR_1));
      RD_DATA_2 <= mem(conv_integer(ADDR_2));
    END IF;
  END PROCESS;
END behavior;
```