A person wearing a plaid shirt

Description automatically generated with low confidence

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*.

# Pendahuluan

P

ada 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:

1. Bahasa VHDL
2. Mikroprosesor MIPS32®.
3. Instruction Set dan Register Mikroprosesor MIPS32®.
4. Simulasi MIPS32® menggunakan PCSpim.

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

1. Quartus v9
2. PCSpim
3. Code editor Visual Studio Code

# Landasan Teoretis

## 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.

Table

Description automatically generated

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.

Diagram, schematic

Description automatically generated Chart, diagram, schematic, box and whisker chart

Description automatically generated

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**;**

Diagram, schematic

Description automatically generated

* 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**;**

Diagram, schematic

Description automatically generated

* 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;**

Diagram

Description automatically generated

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.

Diagram

Description automatically generated A picture containing text, clock

Description automatically generated

**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.

Graphical user interface, text, application

Description automatically generated

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.

Diagram, box and whisker chart

Description automatically generated

Diagram

Description automatically generated with medium confidence

## 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.

## 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.

A picture containing chart

Description automatically generated

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®).

Diagram, schematic

Description automatically generated

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.

Diagram, schematic

Description automatically generated

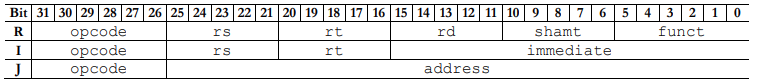
## Instruction Set dan Register Mikroprosesor MIPS32®

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.

Table

Description automatically generated

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.



Text

Description automatically generated

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

Table

Description automatically generated

## 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

A picture containing text

Description automatically generated

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.

# Hasil dan Analisis

Setelah melakukan percobaan pada semua tugas didapatkan hasil sebagai brerikut:

## 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:

Graphical user interface

Description automatically generated

Graphical user interface

Description automatically generated

Graphical user interface

Description automatically generated with low confidence

Graphical user interface

Description automatically generated

Graphical user interface, application

Description automatically generated

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"**;**

## 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:

Graphical user interface

Description automatically generated with medium confidence

A picture containing chart

Description automatically generated

Graphical user interface

Description automatically generated with low confidence

Graphical user interface

Description automatically generated with low confidence

Graphical user interface

Description automatically generated

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

## 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:

Graphical user interface, application

Description automatically generated

Graphical user interface, application

Description automatically generated with medium confidence

Graphical user interface, application

Description automatically generated

Graphical user interface

Description automatically generated with medium confidence

Graphical user interface, application

Description automatically generated

Graphical user interface

Description automatically generated

Graphical user interface

Description automatically generated with medium confidence

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.

## 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**;**

# 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**

### 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**;** |

### 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; |

### 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; |

### 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**;** |