Percobaan 4
Arithmetic And Logical Unit (Alu) Dan Control Unit (Cu)

Dimas Ridhwana Shalsareza (13221076)
Asisten : Muhammad Daffa Rasyid (13220059)
Tanggal Percobaan : 09/11/2023
EL3111 Praktikum Arsitektur Sistem Komputer
Laboratorium Sinyal dan Sistem – Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung

***Abstrak*— Pada praktikum modul 4 yang berjudul "Arithmetic and Logical Unit dan Control Unit", praktikan akan melanjutkan percobaan pada modul sebelumnya yakni percobaan modul 3 dengan melakukan implementasi komponen Algorithmic Logic Unit dan Control Logic Unit menggunakan beberapa tools seperti Quartus Altera. Praktikan akan melaksanakan 6 percobaan diantaranya perancangan program counter, perancangan left shifter dua kali, perancangan carry look ahead adder 32-bit, perangcangan sign extender, perancangan ALU, dan perancangan control unit. Dengan melaksanakan praktikum ini, praktikan memperoleh kesimpulan bahwa setiap blok yang praktikan buat dapat disusun sedemikian sehingga menghasilkan processor MIPS32 yang dibutuhkan.**

***Kata Kunci*— Operand, Propagation delay, Rangkaian Digital**

I. PENDAHULUAN

Praktikum modul 4 yang berjudul "Arithmetic and Logical Unit dan Control Unit" ini memiliki beberapa tujuan sebagai berikut:

i) Praktikan memahami arsitektur mikroprosesor MIPS32® beserta datapath eksekusinya.

ii) Praktikan dapat membuat Arithmetic and Logical Unit (ALU) dari MIPS32® dalam kode VHDL yang synthesizable dan dapat disimulasikan dengan Altera® Quartus® II v9.1sp2.

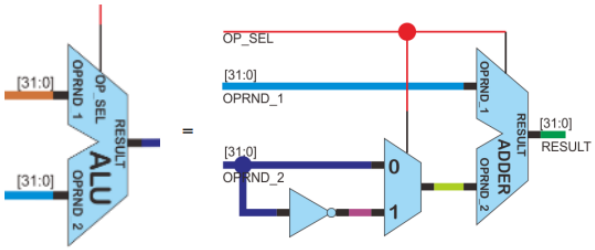
iii) Praktikan dapat membuat Control Unit (CU) dari MIPS32® dalam kode VHDL yang synthesizable dan dapat disimulasikan dengan Altera® Quartus® II v9.1sp2.

Pada percobaan ini, praktikan akan melakukan beberapa percobaan yang menunjang terlaksananya tujuan praktikum tersebut. Praktikan dituntut untuk paham dan terampil dalam menyusun kode VHDL dengan blok yang sudah diperoleh pada pengerjaan tugas pendahulian sedemikian hingga sesuai sesuai dengan teorema yang akan dibuktikan.

II. LANDASAN TEORETIS

A. *Arithmetic and Logical Unit*

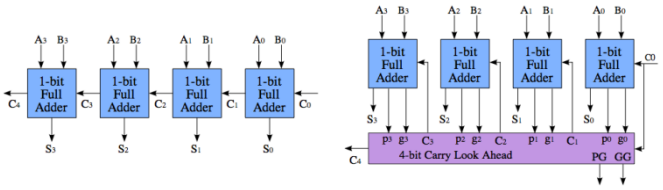
Dalam sistem elektronik digital, sebuah arithmetic and logical unit (ALU) adalah rangkaian digital yang berfungsi untuk melakukan perhitungan integer dan operasi logika. ALU merupakan blok pembangun dasar dari sebuah mikroprosesor. Mikroprosesor modern meliputi central processing unit dan graphics processing unit memiliki ALU yang sangat kompleks untuk melakukan perhitungan. Dalam mikroprosesor modern, digunakan sistem representasi bilangan two's complement



Gambar 2.1.1 Arithmetic Logical Unit.[1]

Pada mikroprosesor Single-Cycle MIPS32® yang akan kita realisasikan dalam praktikum ini, terdapat arithmetic and logical unit (ALU) yang sangat sederhana. ALU ini memiliki lebar data input sebesar 32-bit untuk memasukkan dua buah operand dan memiliki lebar data output sebesar 32-bit untuk mengeluarkan hasil komputasi. ALU ini hanya dapat menangani dua operasi matematika saja yaitu penjumlahan dan pengurangan. Untuk operasi penjumlahan, ALU memanfaatkan blok adder. Sedangkan untuk operasi pengurangan, ALU memanfaatkan sifat bilangan two's complement. Dengan demikian, pengurangan merupakan penjumlahan dengan bilangan negatif. Oleh karena itu, operand kedua dapat diubah menjadi bilangan negatif dengan memanfaatkan prinsip two's complement yaitu rumus $-X = \sim X + 1$. Setelah itu, adder akan menjumlahkan kedua operand tersebut seperti biasa.

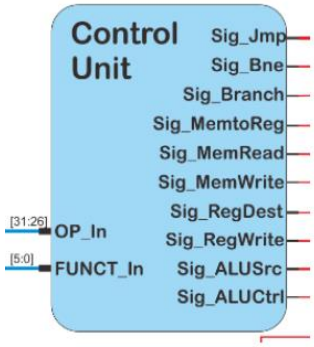
Untuk memilih operasi penjumlahan dan pengurangan, terdapat 2-to-1 multiplexer yang akan memilih arah operand kedua berasal. Untuk penjumlahan, selektor multiplexer bernilai 0 sedangkan untuk pengurangan selektor multiplexer bernilai 1. Selain itu, carry-in untuk
adder juga ditentukan dari operasi yang dilakukan. Untuk penjumlahan, carry-in bernilai 0 sedangkan untuk pengurangan, carry-in untuk bernilai 1. Dengan demikian, kedua sinyal ini (carry in dan selektor multiplexer) dapat dihubungkan menjadi satu sinyal yaitu OP_SEL. Untuk
melakukan inverting operand kedua, digunakan gerbang NOT dengan lebar data 32-bit. Untuk mendesain adder, ada beberapa arsitektur adder yang dapat dipilih. Masing-masing
arsitektur memiliki kelebihan dan kekurangan yang dapat ditinjau dari segi kecepatan, konsumsi daya, dan konsumsi area. Dua contoh arsitektur adder adalah ripple carry adder dan carry-lookahead adder. Ripple carry adder merupakan adder yang relatif sederhana. Kelemahan adder ini adalah dari
segi kecepatan karena setiap bit tidak dapat dijumlahkan secara bersamaan. Tahap adder yang lebih tinggi harus menunggu carry yang dibawa dari tahap adder yang lebih rendah. Pada carry-lookahead adder, setiap tahap adder dapat menghitung carry yang dia terima sehingga tidak perlu menunggu propagasi carry dari tahap sebelumnya. Kelebihan carry-lookahead adder harus dibayar dengan penambahan rangkaian logika yang akan mengkonsumsi luas area.



Gambar 2.1.2 Ripple Carry dan Carry Look Ahead Adder[1]

B. *Control unit*

Control Unit (CU) merupakan komponen dari sebuah mikroprosesor yang berfungsi untuk mengarahkan operasi-operasi yang dilakukan oleh mikroprosesor tersebut. CU mengatur komunikasi dan koordinasi antarkomponen mikroprosesor menggunakan sinyal-sinyal kontrol. CU juga membaca dan menerjemahkan instruksi-instruksi yang diproses untuk menentukan urutan pemrosesan data.



Gambar 2.2.1 Block Control Unit[1]

Pada mikroprosesor Single-Cycle MIPS32® yang akan kita realisasikan dalam praktikum ini, terdapat control unit (CU) yang sangat sederhana. CU menerima opcode dan funct dari instruksi setelah di-decode untuk menentukan nilai dari sinyal-sinyal kontrol yang dikeluarkan. Terdapat sepuluh sinyal kontrol yang keluar dari CU ini yang dijelaskan sebagai berikut

Nama Sinyal	Lebar	Tujuan	Fungsi
Sig_Jmp	2 bit	*2-to-1 Mux* pada *Program Counter*	Menunjukkan adanya instruksi *jump* sehingga *program counter* dapat diset sesuai dengan *address* hasil kalkulasi.
Sig_Bne	1 bit	Gerbang OR 2 *Input*	Menunjukkan adanya instruksi `bne` untuk memilih hasil pencabangan pada *program counter.*
Sig_Branch	1 bit	Gerbang OR 2 *Input*	Menunjukkan adanya instruksi `beq` untuk memilih hasil pencabangan pada *program counter.*
Sig_MemtoReg	1 bit	*2-to-1 Mux* pada *Data Memory*	Memilih data untuk *writeback*, apakah berasal dari *data memory* atau ALU.
Sig_MemRead	1 bit	*Data Memory*	Sinyal yang mengaktifkan operasi baca pada *data memory.*
Sig_MemWrite	1 bit	*Data Memory*	Sinyal yang mengaktifkan operasi tulis pada *data memory.*
Sig_RegDest	2 bit	*4-to-1 Mux* pada *Register*	Memilih *register* yang akan dijadikan sebagai *destination register.*
Sig_RegWrite	1 bit	*Register*	Sinyal yang mengaktifkan operasi tulis pada *register.*
Sig_ALUSrc	2 bit	*4-to-1 Mux* pada ALU	Memilih data *operand* kedua yang akan masuk ke ALU, apakah dari *register* atau dari *immediate.*
Sig_ALUCtrl	2 bit	ALU	Memilih operasi yang akan dilakukan pada ALU apakah penjumlahan atau pengurangan.

Gambar 2.2.2 Gambar tabel sinyal kontrol[1]

Terdapat sembilan instruksi yang dapat dieksekusi oleh mikroprosesor Single-Cycle MIPS32® yang akan kita realisasikan dalam praktikum ini. Kesembilan instruksi tersebut akan menentukan nilai sinyal yang dikeluarkan oleh control unit karena setiap instruksi membutuhkan penanganan dan aliran data yang berbeda-beda. Berikut ini tabel nilai sinyal control unit untuk setiap instruksi yang dapat dieksekusi.

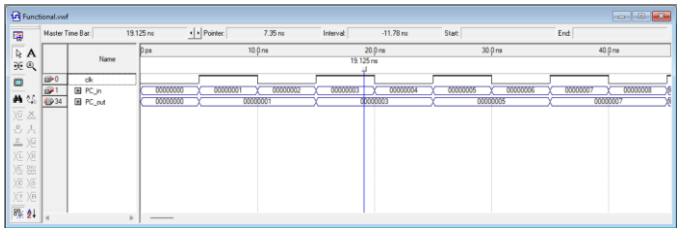
Instruksi	Tipe	Sig_Jmp	Sig_Bne	Sig_Branch	Sig_MemtoReg	Sig_MemRead
add	R	00	0	0	0	0
sub	R	00	0	0	0	0
beq	I	00	0	1	0	0
bne	I	00	1	0	0	0
addi	I	00	0	0	0	0
lw	I	00	0	0	1	1
sw	I	00	0	0	0	0
jmp	J	01	0	0	0	0
nop	-	00	0	0	0	0

Instruksi	Tipe	Sig_MemWrite	Sig_RegDest	Sig_RegWrite	Sig_ALUSrc	Sig_ALUCtrl
add	R	0	01	1	00	00
sub	R	0	01	1	00	01
beq	I	0	--	0	-	--
bne	I	0	--	0	-	--
addi	I	0	00	1	01	00
lw	I	0	00	1	01	00
sw	I	1	00	0	01	00
jmp	J	0	--	0	-	--
nop	-	0	00	0	00	00

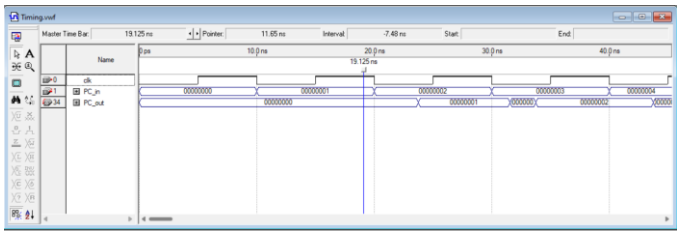
Gambar 2.2.2 Gambar tabel instruction Set[1]

III. HASIL DAN ANALISIS

A. *Perancangan Program Counter*



Gambar 3.1.1 Program Counter Simulasi Functional



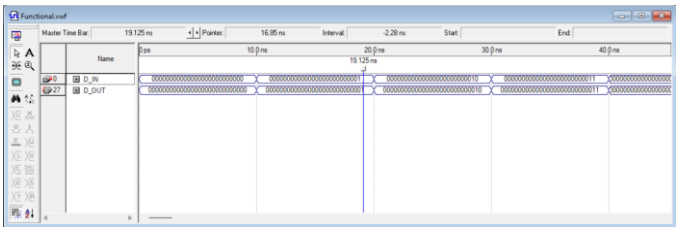
Gambar 3.1.2 Program Counter Simulasi Timing

Analisis:
Pada percobaan tugas 1 mengenai perancangan program counter, praktikan melakukan penyusunan berupa kode VHDL yang tertera pada lampiran 1. Dengan menggunakan kode tersebut, praktikan berhasil merepresentasikan sebuah program counter yang sesuai dengan spesifikasi yang dibutuhkan. Praktikan melanjutkan percobaan dengan melakukan analisis simulasi secara functional dan timing seperti yang tertera pada gambar 3.1.1 dan gambar 3.1.2. Program counter ini akan bekerja dengan merealisasikan program flipflop dengan suatu register 32-bit sehingga menggunakan 32 flip-flop. Jenis counter yang digunakan adalah tipe rising sehingga akan menerima data ketika clock sedang rising.

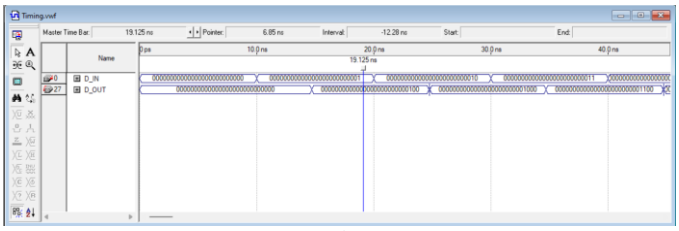
Sesuai yang terbaca pada gambar 3.1.1, pada simulasi functional terlihat bahwa hasil yang diperoleh sudah sesuai dimana program akan membawa informasi memory terakhir ketika clock rising. Informasi terakhir tersebut akan berjalan hingga kondisi clock rising kembali terpenuhi. Dengan terpenuhinya kondisi clk rising tersebut, program akan menyimpan kembali nilai terbaru dan mengulangi proses yang sama.

Hal yang berbeda terjadi pada penggunaan simulasi timing sesuai yang tertera pada gambar 3.1.2. Pada simulasi timing, hasil yang tebaca ketika clock rising tidak secara langsung disimpan, namun terdapat suatu delay propagation. Kondisi dengan menggunakan analisis timing juga memberikan output yang berbeda dengan hasil simulasi fungsional dikarenakan delay propagasi yang memberikan efek perlambatan pembacaan sehingga nilai yang terbaca tidak tepat sesuai dengan nilai aktual.

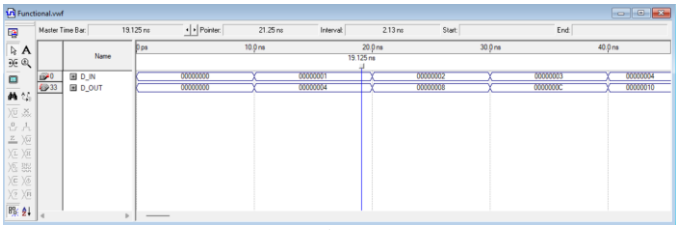
B. *Perancangan Left Shifter Dua Kali*



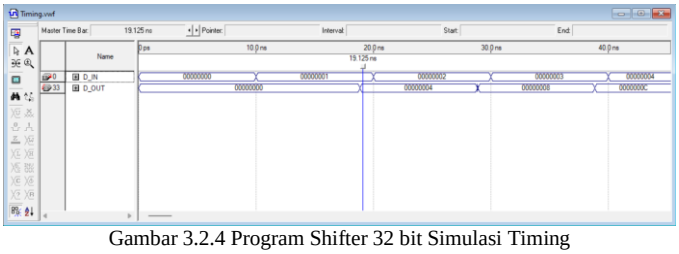
Gambar 3.2.1 Program Shifter 26 bit Simulasi Functional



Gambar 3.2.2 Program Shifter 26 bit Simulasi Timing



Gambar 3.2.3 Program Shifter 32 bit Simulasi Functional



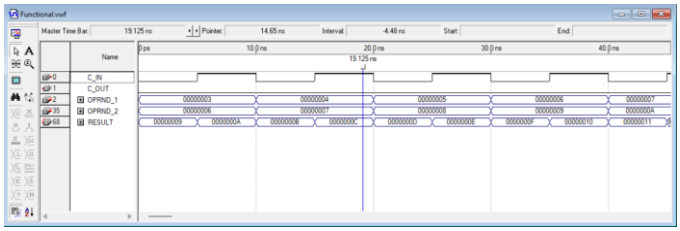
Gambar 3.2.4 Program Shifter 32 bit Simulasi Timing

Pada percobaan tugas 2 perancangan left shifter dua kali, praktikan pertama-tama melakukan pernyusunan program menggunakan bahasa VHDL. Pada percobaan ini, praktikan menggunakan dua jenis shifter dengan input berbeda yakni 26 bit dan 32 bit. Dengan kedua variasi tersebut, praktikan melakukan simulasi secara fungsional dan timing sehingga diperoleh hasil seperti yang tertera pada gambar 3.2.1 hingga gammbar 3.2.4.

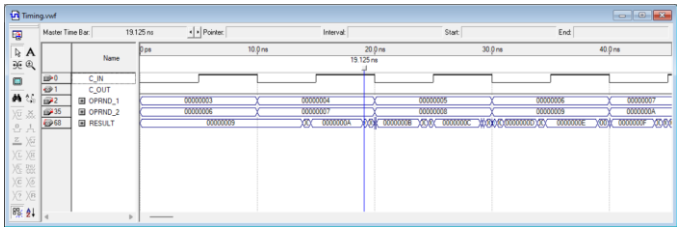
Dengan mengamati hasil pada gambar 3.2.1, praktikan memperoleh hasil simulasi shifting 26 bit secara fungsional. Pada hasil tersebut terlihat perbedaan dengan gambar 3.2.2 dimana pada gambar 3.2.2 dilakukan simulasi timing. Pada simulasi timing, praktikan memperoleh hasil shifting 2 bit yang sudah sesuai namun terdapat delay propagasi. Hal tersebut terjadi diakibatkan penggunaan simulasi timing mempertimbangkan kondisi delay propagasi yang dihasilkan pada perancangan program. Keberadaan delay propagasi ini dapat mengakibatkan hasil yang terjadi pada simulasi timing tidak sesuai dengan kenyataan yang diinginkan pada simulasi fungsional.

Dengan mengamati hasil pada gambar 3.2.3 dan gambabr 3.2.4, praktikan memperoleh hasil simulasi shifting 32 bit secara fungsional. Pada percobaan tersebut, praktikan menggunakan representasi hexadesimal sehingga tidak terlihat secara jelas shifting yang terjadi didalamnya. Namun apabila dianalisis nilai hasil perhitungan tersebut, praktikan memperoleh bahwa hasil hexadesimal sesuai dengan nilai yang diharapkan.

C. Perancangan Carry Look Ahead Adder 32-bit



Gambar 3.3.1 Program Carry look Ahead adder Simulasi Functional

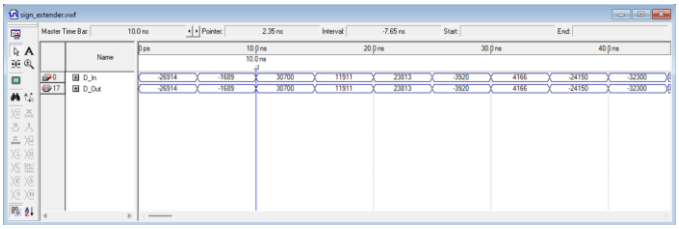


Gambar 3.3.2 Program Carry Look Ahead adder Simulasi Timing

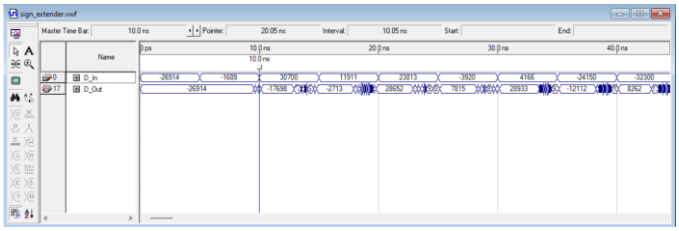
Analisis:
Dengan mengamati gambar 3.3.1 yang merupakan hasil simulasi fungsional carry look ahead adder, terlihat bahwa hasil sudah sesuai dengan yang diharapkan dimana program akan melakukan penjumlahan setiakali C_IN bernilai LOW. Sebagai contoh pada kondisi Opreand 1 bernilai 3 dan operand 2 bernilai 6, praktikan memperoleh output sebesar 9 dalam hexadesimal sesuai dengan hasil yang diharapkan. Hal menarik terjadi ketika C_IN bernilai HIGH, program secara otomatis melakukan penambahan sebagai carry sehingga memperoleh hexadesimal bernilai A. Hal ini menandakan skema carry adder dimana penjumlahan dilakukan dengan mempertimbangkan nilai Carry yang diperoleh secara langsung tanpa perlu menunggu program ssebelumnya selesai. Analisis tersebut secara tidak langsung menjelaskan perbedaan antara carry look ahead adder dengan ripple carry adder dimana pada ripple carry adder, diperlukan hasil carry yang dihasilkan dari penjumlahan adder 1 bit least significant.

Praktikan juga memperoleh hasil simulasi timing dimana sesuai dengan skema simulasi timing yang memperhatikan delay propagasi, praktikan memperoleh hasil seperti gambar 3.3.2 dengan delay propagasi. Walaupun terdapat nilai delay propagasi tersebut, hasil yang diperoleh pada output tetap menunjukkan nilai yang sama.

D. Perangcangan Sign Extender



Gambar 3.4.1 Program Sign Extender Simulasi Functional

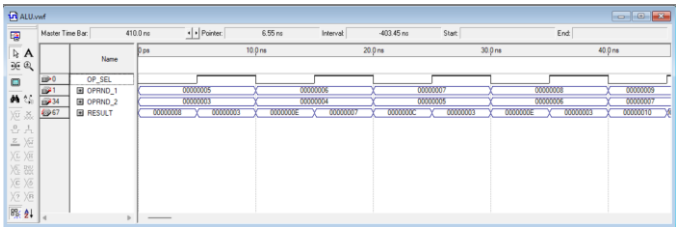


Gambar 3.4.2 Program Sign Extender Simulasi Timing

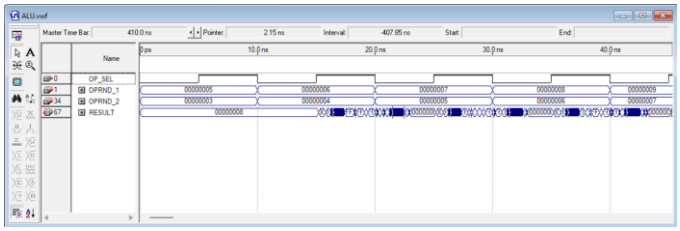
Dengan mengamati gambar 3.4.1, praktikan memperoleh output D_Out yang sama dengan D_in. Hal ini mengindikasikan bahwa Sign Extender yang praktikum telah rancang sudah sesuai dengan nilai yang seharusnya terjadi. Hal ini dapat diamati lebih jelas apabila praktikan melakukan perubahan nilai radix yang merupakan desimal menjadi representasi dalam bit. Apabila dilakukan pengamatan dengan representasi bit, akan terlihat bahwa panjang bit yang diperoleh akan semakin besar namun nilai representasi bit yang dihasilkan akan tetap sama. Keberadan sign extender ini krusial pada processor MIPS32 dimana akan terdapat left shifting sebanyak 2 kali. Dengan terdapatnyaa sign extender, maka data yang diperoleh atau dilakukan operasi shifting akan tetap sesuai dan tidak mengalami data hilang.

Dengan mengamati gambat 3.4.2, praktikan memperoleh D out yang bernilai sama dengan D_in namun terjadi perubahan pada D_out tahap selanjutnya. Hal ini terjadi akibat terdapatnya delay propagasi pada program. Dengan mengamati delay tersebut, praktikan akan memperoleh lost data yang dapat dijelaskan secara teoritis sehingga dapat dilakukan penyetaraan data kembali.

E. Perancangan ALU



Gambar 3.5.1 Program ALU Simulasi Functional

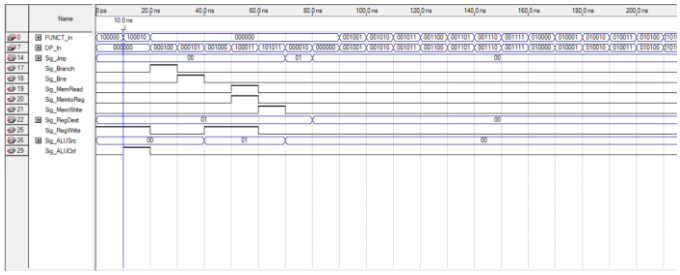


Gambar 3.5.2 Program ALU Simulasi Timing

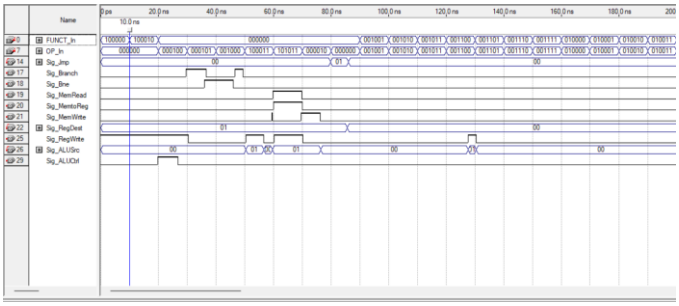
Pada percobaan 5 ini, praktikan merancang program ALU dan melakukan simulasi secara timing dan fungsional. Praktikan memanfaatkan carry look ahead adder yang sudah diperoleh pada tugas sebelumnya dan memberikan selektor operasi berupa 1 bit. Sesuai dengan konsep ALU, praktikan merancang program tersebut dengan terlebih dahulu menentukan dua jenis input. Input pertama akan langsung memasukin blok CLA sedangkan input kedua akan dilakukan selektor terlebih dahulu untuk menentukan jenis operasi yang akan dilakukan. Apabila selektor menandakan nilai "0" yang bermakna penjumlahan, makan program akan langsung menjumlahkan kedua input dengan CLA. Sedangkan ketika selektor menandakan nilai "1" yang bermakna pengurangan, program akan mengkonvesi nilai input 2 menggunakan notgate dan melakuakn representasi twos complement sebagai operasi pengurangan. Hasil yang diperoleh kemudian dianjutkan melalui Carry Look Ahead Adder.

Dengan mengamati hasil simulasi functional program ALU seperti yangt ertera pada gambar 3.5.1, praktikan memperoleh bahwa hasil yang diperoleh sudah sesuai dengan nilai yang diharapkan. Dimana pada penggunaan operand 1 sebesar 5 dalam hexadesimal dan operand 2 bernilai 3 dalam hexadesimal dan memanfaatkan seletor berniali 0 yang menandakan operasi yang dilakukan merupakan penjumlahan, akan diperoleh hasil bernilai 8 dalam hexadesimal. Representasi tersebut sudah sesuai dengan hasil yang diharapkan. Dengan membandingkan hasil simulasi fungsional dan timing, praktikan mmeperoleh hasil yang sama untuk nilai operand_1 dan opreand_2 dengan selektor penjumlahan. Namun terdapat perbedaan dimana dengan simulasi timing, hasil selanjutnya yang diperoleh mengalami delay propagasi yang cukup besar. Hal ini dapat terjadi akibat penggunaan Carry look ahead adder yang juga memiliki karakteristik delay propagasi yang besar.

F. Perancangan Control Unit



Gambar 3.6.1 Program Control Unit Simulasi Functional



Gambar 3.6.2 Program Control Unit Timing

Analisis:

Pada percobaan 6 ini, praktikan merancang program Control Unit yang menerima dua input dengan panjang masing-masing 6 bit sebagai opcode dan funct. Praktikan merancang suatu kode VHDL yang dapat membaca nilai input tersebut dan melakukan penyebaran output operasi. Hasil dari penyebaran output operasi tersesbut akan menunjukkan jenis instruksi sehingga jenis instruksi tersebut dapat terdeteksi. Dalam penggunaan Control Unit ini, praktikan dapat mengatur komponen MIPS32 seperti ALU denganoperasi penjumlahan dan penguranagan, operasi lain seperti shifting dengan output-output instruksi yang bervariasi.

Sesuai dengan hasil percobaan yang terlihat pada gambar 3.6.1 dengan simulasi fungsional, praktikan memperoleh hasil yang sesuai dengan yang diharapkan dimana program berhasil mengurai input berupa funct dan opcode sesuai dengan tipe-tipe instruksi. Dengan mengamati hasil pada gambar 3.6.2, praktikan juga memperoleh bahwa output sesuai dengan hasil percobaan simulasi fungsional namun dengan suatu nilai delay propagasi. Delay propagasi ini terjadi akibat penggnuaan blok lain yang memiliki delay sehingga menghasilkan pergeseran waktu yang mengakibatkan proses yang terjadi tidak terjadi secara realtime. Hal ini sudah sesuai dengan karakteristik penggunaan simulasi timing dimana simulasi timing akan memperhitungkan delay propagasi dari masing-masing program yang diekekusi.

IV. SIMPULAN

- Dengan melakukan percobaan pada modul 4 ini, praktikan dapat menyimpulkan bahwa processor MIPS32 disusun oleh suatu susunan blok yang terdapat salah satunya ALU dan CU yang masing-masing memiliki kegunaan yang krusial untuk pengaturan nilai dan memory pada processor MIPS32. Pemanfaatan blok tersebut dapat disusun sedemikian dengan memanfaatkan kode VHDL untuk memperoleh output yang sesuai dengan kebutuhan.
- Dengan meakukan percobaan modul 4 ini dan merujuk pada tugas 5, praktikan dapat mengambil kesimpulan bahwa penyusunan Arithmetic dan Logical Unit pada VHDL dapat menggunakan diagram yang lebih sederhana yakni memanfaatkan konsep penjumlahan dengan sebuah adder dan pengaturan nilai positif ataupun negatif pada salah satu input sesuai dengan multiplexer 2 bit untuk menghasilkan operasi penjumlahan maupun pengurangan.
- Dengan melaksanakan tugas 6, praktikan dapat mengambil kesimpulan bahwa proses pembuatan control unit membutuhkan 2 input dengan masing-masing bernilai 6 bit. Nilai yang diperoleh pada input tersebut akan diproses untuk menunjukkan jenis instruksi yang akan dieksekusi pada tahap selanjutnya

REFERENSI

[1] R. Bryant, D. O'Hallaron, “Computer Systems - A Programmer's Persp.” 2nd ed., Pearson, 2010, BBS

[2] Andri H.dkk, “Modul EL-3111 Arsikom 2023-2024”, LDTE, 2017

[3] R. Bryant, D. O'Hallaron, “Computer Systems - A Programmer's Persp.” 2nd ed., Pearson, 2010, BBS

[4] Andri H.dkk, “Modul EL-3111 Arsikom 2023-2024”, LDTE, 2017

Lampiran

1. Source code untuk tugas 1 (program_counter.vhd)

```
-- Praktikum EL3111 Arsitektur Sistem Komputer
-- Modul         : 4
-- Tanggal       : 9 November 2023
-- Kelompok      : 9
-- Rombongan     : D
-- Nama (NIM)    : Dimas Ridhwana Shalsareza (13221076)
-- Nama File     : program_counter.vhd

LIBRARY IEEE;  -- Mengimport library IEEE untuk menggunakan tipe data dan fungsi
standar VHDL
USE IEEE.STD_LOGIC_1164.ALL;  -- Menggunakan paket STD_LOGIC_1164 untuk tipe data
logika standar
USE IEEE.STD_LOGIC_ARITH.ALL;  -- Menggunakan paket STD_LOGIC_ARITH untuk operasi
aritmetika standar
USE IEEE.STD_LOGIC_UNSIGNED.ALL;  -- Menggunakan paket STD_LOGIC_UNSIGNED untuk
operasi unsigned standar
USE IEEE.NUMERIC_STD.ALL;
LIBRARY altera_mf;
USE altera_mf.altera_mf_components.ALL;


ENTITY program_counter IS
  PORT (
    clk : IN STD_LOGIC;  -- Port input clock.
    PC_in : IN STD_LOGIC_VECTOR (31 DOWNTO 0);  -- Port input data 32-bit.
    PC_out : OUT STD_LOGIC_VECTOR (31 DOWNTO 0)  -- Port output data 32-bit.
  );
END program_counter;

ARCHITECTURE behavior OF program_counter IS
  SIGNAL PC_reg : STD_LOGIC_VECTOR (31 DOWNTO 0) := (OTHERS => '0');  -- Sinyal
internal untuk merepresentasikan nilai register 32-bit, diinisialisasi dengan
nilai 0.

BEGIN
  PROCESS(clk)  -- Proses yang akan dieksekusi ketika ada perubahan pada clock.
  BEGIN
    IF RISING_EDGE(clk) THEN  -- Pengecekan apakah clock berada pada rising edge.
      PC_reg <= PC_in;  -- Jika ya, update nilai register dengan nilai input.
    END IF;
  END PROCESS;

  PC_out <= PC_reg;  -- Keluaran mengikuti nilai dari register PC_reg.
END behavior;
```

2. Source code untuk tugas 2.1 (lshift_26_28.vhd)

```
-- Praktikum EL3111 Arsitektur Sistem Komputer
-- Modul         : 4
-- Tanggal       : 9 November 2023
-- Kelompok      : 9
-- Rombongan     : D
-- Nama (NIM)    : Dimas Ridhwana Shalsareza (13221076)
-- Nama File     : lshift_26_28.vhd

LIBRARY IEEE;  -- Mengimport library IEEE untuk menggunakan tipe data dan fungsi
standar VHDL
```

```
USE IEEE.STD_LOGIC_1164.ALL;  -- Menggunakan paket STD_LOGIC_1164 untuk tipe data
logika standar
USE IEEE.STD_LOGIC_ARITH.ALL;  -- Menggunakan paket STD_LOGIC_ARITH untuk operasi
aritmetika standar
USE IEEE.STD_LOGIC_UNSIGNED.ALL;  -- Menggunakan paket STD_LOGIC_UNSIGNED untuk
operasi unsigned standar
USE IEEE.NUMERIC_STD.ALL;
LIBRARY altera_mf;
USE altera_mf.altera_mf_components.ALL;

ENTITY lshift_26_28 IS
PORT (
  D_IN : IN STD_LOGIC_VECTOR (25 DOWNTO 0); -- Input 32-bit;
  D_OUT : OUT STD_LOGIC_VECTOR (27 DOWNTO 0) -- Output 32-bit;
);
END lshift_26_28;

ARCHITECTURE behavior OF lshift_26_28 IS
BEGIN
    D_OUT <= D_IN(25 DOWNTO 0) & "00"; -- Menggeser input sebanyak dua bit ke
kiri, dan menambahkan dua bit 0 di kanan.
END behavior;
```

3. Source code untuk tugas 2.2 (lshift_32_32.vhd)

```
-- Praktikum EL3111 Arsitektur Sistem Komputer
-- Modul         : 4
-- Tanggal       : 9 November 2023
-- Kelompok      : 9
-- Rombongan     : D
-- Nama (NIM)    : Dimas Ridhwana Shalsareza (13221076)
-- Nama File     : lshift_32_32.vhd

LIBRARY IEEE;  -- Mengimport library IEEE untuk menggunakan tipe data dan fungsi
standar VHDL
USE IEEE.STD_LOGIC_1164.ALL;  -- Menggunakan paket STD_LOGIC_1164 untuk tipe data
logika standar
USE IEEE.STD_LOGIC_ARITH.ALL;  -- Menggunakan paket STD_LOGIC_ARITH untuk operasi
aritmetika standar
USE IEEE.STD_LOGIC_UNSIGNED.ALL;  -- Menggunakan paket STD_LOGIC_UNSIGNED untuk
operasi unsigned standar
USE IEEE.NUMERIC_STD.ALL;
LIBRARY altera_mf;
USE altera_mf.altera_mf_components.ALL;

ENTITY lshift_32_32 IS
PORT (
  D_IN : IN STD_LOGIC_VECTOR (31 DOWNTO 0); -- Input 32-bit;
  D_OUT : OUT STD_LOGIC_VECTOR (31 DOWNTO 0) -- Output 32-bit;
);
END lshift_32_32;

ARCHITECTURE behavior OF lshift_32_32 IS
BEGIN
    D_OUT <= D_IN(29 DOWNTO 0) & "00"; -- Menggeser input sebanyak dua bit ke
kiri, dan menambahkan dua bit 0 di kanan.
END behavior;
```

4. Source code untuk tugas 3 (cla_32.vhd)

```vhdl
-- Praktikum EL3111 Arsitektur Sistem Komputer
-- Modul         : 4
-- Tanggal       : 9 November 2023
-- Kelompok      : 9
-- Rombongan     : D
-- Nama (NIM)    : Dimas Ridhwana Shalsareza (13221076)
-- Nama File     : cla_32.vhd

LIBRARY IEEE;  -- Mengimport library IEEE untuk menggunakan tipe data dan fungsi standar VHDL
USE IEEE.STD_LOGIC_1164.ALL;  -- Menggunakan paket STD_LOGIC_1164 untuk tipe data logika standar
USE IEEE.STD_LOGIC_ARITH.ALL;  -- Menggunakan paket STD_LOGIC_ARITH untuk operasi aritmetika standar
USE IEEE.STD_LOGIC_UNSIGNED.ALL;  -- Menggunakan paket STD_LOGIC_UNSIGNED untuk operasi unsigned standar
USE IEEE.NUMERIC_STD.ALL;
LIBRARY altera_mf;
USE altera_mf.altera_mf_components.ALL;

ENTITY cla_32 IS
  PORT (
    OPRND_1    : IN STD_LOGIC_VECTOR(31 DOWNTO 0); -- Operand 1
    OPRND_2    : IN STD_LOGIC_VECTOR (31 DOWNTO 0); -- Operand 2
    C_IN       : IN STD_LOGIC; -- Carry In
    RESULT     : OUT STD_LOGIC_VECTOR (31 DOWNTO 0); -- Result
    C_OUT      : OUT STD_LOGIC -- Overflow
  );
END cla_32;

ARCHITECTURE behavioral OF cla_32 IS
  SIGNAL c_int : STD_LOGIC_VECTOR(31 DOWNTO 1);
  SIGNAL sum : STD_LOGIC_VECTOR(31 DOWNTO 0);
  SIGNAL car_generate : STD_LOGIC_VECTOR(31 DOWNTO 0);
  SIGNAL car_propagate : STD_LOGIC_VECTOR(31 DOWNTO 0);
BEGIN
  PROCESS(OPRND_1, OPRND_2, C_IN)
  BEGIN
    sum <= OPRND_1 XOR OPRND_2;
    car_generate <= OPRND_1 AND OPRND_2;
    car_propagate <= OPRND_1 OR OPRND_2;

    c_int(1) <= car_generate(0) OR (car_propagate(0) AND C_IN);

    FOR i IN 1 TO 30 LOOP
      c_int(i+1) <= car_generate(i) OR (car_propagate(i) AND c_int(i));
    END LOOP;

    C_OUT <= car_generate(31) OR (car_propagate(31) AND c_int(31));
  END PROCESS;

  RESULT(0) <= sum(0) XOR C_IN;
  RESULT(31 DOWNTO 1) <= sum(31 DOWNTO 1) XOR c_int(31 DOWNTO 1);

END behavioral;
```

5. Source code untuk tugas 4 (sign_extender.vhd)

```vhdl
-- Praktikum EL3111 Arsitektur Sistem Komputer
-- Modul         : 4
```

```
-- Tanggal       : 9 November 2023
-- Kelompok      : 9
-- Rombongan     : D
-- Nama (NIM)    : Dimas Ridhwana Shalsareza (13221076)
-- Nama File     : sign_extender.vhd

LIBRARY ieee;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_unsigned.all;
USE IEEE.NUMERIC_STD.ALL;
LIBRARY altera_mf;
USE altera_mf.altera_mf_components.ALL;

ENTITY sign_extender IS
PORT (
    D_In  : IN std_logic_vector (15 DOWNTO 0); -- Data Input 1
    D_Out :OUT std_logic_vector (31 DOWNTO 0) -- Data Input 2
);
END sign_extender;

ARCHITECTURE Behavioral OF sign_extender IS
BEGIN
    D_OUT(15 DOWNTO 0) <= D_IN(15 DOWNTO 0);
    D_OUT(31 DOWNTO 16) <= (OTHERS => D_IN (15));
END Behavioral;
```

6. Source code untuk tugas 5 (ALU.vhd)

```
-- Praktikum EL3111 Arsitektur Sistem Komputer
-- Modul         : 4
-- Tanggal       : 9 November 2023
-- Kelompok      : 9
-- Rombongan     : D
-- Nama (NIM)    : Dimas Ridhwana Shalsareza (13221076)
-- Nama File     : ALU.vhd

LIBRARY ieee;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_unsigned.all;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.NUMERIC_STD.ALL;
LIBRARY altera_mf;
USE altera_mf.altera_mf_components.ALL;

ENTITY ALU IS
PORT (
    OPRND_1     : IN std_logic_vector (31 DOWNTO 0); -- Data Input 1
    OPRND_2     : IN std_logic_vector (31 DOWNTO 0); -- Data Input 2
    OP_SEL      : IN std_logic; -- Operation Select
    RESULT      : OUT std_logic_vector (31 DOWNTO 0) -- Data Output
);
END ALU;

ARCHITECTURE Behavioral OF ALU IS
    COMPONENT cla_32 IS
    PORT (
        OPRND_1 : IN STD_LOGIC_VECTOR(31 DOWNTO 0); -- Operand 1
        OPRND_2 : IN STD_LOGIC_VECTOR (31 DOWNTO 0); -- Operand 2
        C_IN    : IN STD_LOGIC;  -- Carry In
        RESULT  : OUT STD_LOGIC_VECTOR (31 DOWNTO 0); -- Result
```

```
        C_OUT   : OUT STD_LOGIC -- Overflow
    );
    END COMPONENT;
    SIGNAL RES: STD_LOGIC_VECTOR(31 DOWNTO 0);
    BEGIN
    ADDER: cla_32
    PORT MAP (
        OPRND_1 => OPRND_1,
        OPRND_2 => RES,
        C_IN => OP_SEL,
        RESULT => RESULT
    );
    PROCESS(OPRND_2, OP_SEL)
    BEGIN
        IF (OP_SEL = '0') THEN
            RES <= OPRND_2; -- nilai asli
        ELSIF (OP_SEL = '1') THEN
            RES <= NOT(OPRND_2) + 1; --NOT (2's complement)
        ELSE
            RES <= (OTHERS => '0');
        END IF;
    END PROCESS;
END Behavioral;
```

7. Source code untuk tugas 6 (cu.vhd)

```
-- Praktikum EL3111 Arsitektur Sistem Komputer
-- Modul         : 4
-- Tanggal       : 9 November 2023
-- Kelompok      : 9
-- Rombongan     : D
-- Nama (NIM)    : Dimas Ridhwana Shalsareza (13221076)
-- Nama File     : cu.vhd

LIBRARY IEEE;  -- Mengimport library IEEE untuk menggunakan tipe data dan fungsi
standar VHDL
USE IEEE.STD_LOGIC_1164.ALL;  -- Menggunakan paket STD_LOGIC_1164 untuk tipe data
logika standar
USE IEEE.STD_LOGIC_ARITH.ALL;  -- Menggunakan paket STD_LOGIC_ARITH untuk operasi
aritmetika standar
USE IEEE.STD_LOGIC_UNSIGNED.ALL;  -- Menggunakan paket STD_LOGIC_UNSIGNED untuk
operasi unsigned standar
USE IEEE.NUMERIC_STD.ALL;
LIBRARY altera_mf;
USE altera_mf.altera_mf_components.ALL;

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY cu IS
PORT (
  OP_In          : IN STD_LOGIC_VECTOR (5 DOWNTO 0);
  FUNCT_In       : IN STD_LOGIC_VECTOR (5 DOWNTO 0);
  Sig_Jmp        : OUT STD_LOGIC_VECTOR (1 DOWNTO 0);
  Sig_Bne        : OUT STD_LOGIC;
  Sig_Branch     : OUT STD_LOGIC;
  Sig_MemtoReg   : OUT STD_LOGIC;
  Sig_MemRead    : OUT STD_LOGIC;
  Sig_MemWrite   : OUT STD_LOGIC;
```

```
  Sig_RegDest    : OUT STD_LOGIC_VECTOR (1 DOWNTO 0);
  Sig_RegWrite   : OUT STD_LOGIC;
  Sig_ALUSrc     : OUT STD_LOGIC_VECTOR (1 DOWNTO 0);
  Sig_ALUCtrl    : OUT STD_LOGIC_VECTOR (1 DOWNTO 0)
);
END cu;

ARCHITECTURE behavioral OF cu IS
BEGIN
  PROCESS(OP_In, FUNCT_In)
  BEGIN
    Sig_Jmp       <= "00";
    Sig_Bne       <= '0';
    Sig_Branch    <= '0';
    Sig_MemtoReg  <= '0';
    Sig_MemRead   <= '0';
    Sig_MemWrite  <= '0';
    Sig_ALUSrc    <= "00";
    Sig_RegDest   <= "00";
    Sig_RegWrite  <= '0';
    Sig_ALUCtrl   <= "00";

    IF (OP_In = "000000") THEN  -- R-type instructions
      IF (FUNCT_In = "000000") THEN
        Sig_RegDest   <= "00";
        Sig_RegWrite  <= '0';
        Sig_ALUCtrl   <= "00";
      ELSIF (FUNCT_In = "100010") THEN
        Sig_ALUCtrl   <= "01";
      END IF;
    ELSE
      Sig_ALUCtrl <= "00";

      IF (OP_In = "000010") THEN  -- Jump instruction
        Sig_Jmp <= "01";
      ELSIF (OP_In = "000100") THEN  -- Branch instructions
        Sig_Branch <= '1';
      ELSIF (OP_In = "000101") THEN  -- Bne instruction
        Sig_Bne <= '1';
      ELSIF (OP_In = "001000" OR OP_In = "100011") THEN  -- Load and Store
instructions
        Sig_RegWrite <= '1';
        Sig_ALUSrc <= "01";
        Sig_MemtoReg <= '1';
        Sig_MemRead <= '1';
      ELSIF (OP_In = "101011") THEN  -- Store instruction
        Sig_MemWrite <= '1';
      END IF;
    END IF;
  END PROCESS;
END behavioral;
```