Percobaan IV

SYNTHESIZABLE MIPS32® MICROPROCESSOR BAGIAN II : ARITHMETIC AND LOGICAL UNIT (ALU) DAN CONTROL UNIT (CU)

Ahmad Aziz (13220034)

Asisten: Oktavian Putra Masyiakh (13219039)

Tanggal Percobaan: 11/11/2022

EL3111 Praktikum Arsitektur Sistem Komputer

Laboratorium Sinyal dan Sistem – Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung

***Abstrak*— Pada praktikum modul 4 yaitu SYNTHESIZABLE MIPS32® MICROPROCESSOR BAGIAN II : ARITHMETIC AND LOGICAL UNIT (ALU) DAN CONTROL UNIT (CU), dilakukan beberapa perancangan untuk membuat ALU dan juga CU. Tujuan percobaan pada praktikum modul 4 ini adalah memahami arsitektur mikroprosesor MIPS32® beserta datapath eksekusinya, dapat membuat Arithmetic and Logical Unit (ALU) dari MIPS32® dalam kode VHDL, serta dapat membuat Control Unit (CU) dari MIPS32® dalam kode VHDL. Ada enam tugas yang akan dilakukan pada praktikum modul 4 yaitu Perancangan Program Counter, Perancangan Left Shifter Dua Kali, Perancangan Carry-Lookahead Adder 32-bit, Sign Extender dan Control Unit (CU). Semua percobaan pada praktikum ini dilakukan pada perangkat computer dengan sistem operasi windows dengan software yang digunakan adalah Altera® Quartus® II v9.1sp2 serta code editor menggunakan Visual Studio Code dan juga Notepad++.**

***Kata Kunci*—ALU, control unit, counter, adder.**

I. PENDAHULUAN

Pada praktikum modul 4 yaitu SYNTHESIZABLE MIPS32® MICROPROCESSOR BAGIAN II : ARITHMETIC AND LOGICAL UNIT (ALU) DAN CONTROL UNIT (CU) dilakukan sebanyak 6 percobaan yang bertujuan diantaranya sebagai berikut:

- Praktikan memahami arsitektur mikroprosesor MIPS32® beserta datapath eksekusinya.
- Praktikan dapat membuat Arithmetic and Logical Unit (ALU) dari MIPS32® dalam kode VHDL yang synthesizable dan dapat disimulasikan dengan Altera® Quartus® II v9.1sp2
- Praktikan dapat membuat Control Unit (CU) 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. Perancangan Program Counter
b. Perancangan Left Shifter Dua Kali.
c. Perancangan Carry-Lookahead Adder 32-bit.
d. Sign Extender.
e. Arithmetic and Logical Unit (ALU).
f. Control Unit (CU).

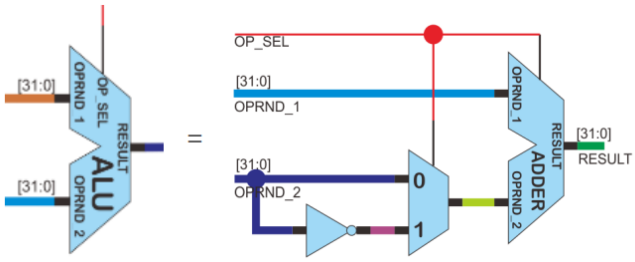
Dalam melakukan percobaan dan analisis pada praktikum modul ini, perangkat lunak dan alat yang digunakan adalah sebagai berikut:

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

II. LANDASAN TEORETIS

A. *Arithmetic and Logical Unit (ALU)*

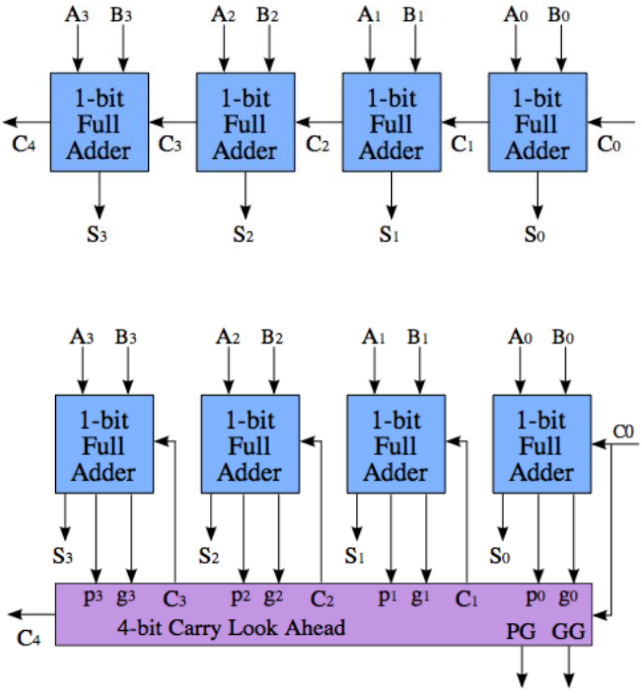
Dalam sistem elektronik digital, sebuah arithmetic and logical unit (ALU) adalah rangkaian digital yang berfungsi untuk melakukan perhitungan integer dan operasi logika. ALU merupakan blok pembangun dasar dari sebuah mikroprosesor. Mikroprosesor modern meliputi central processing unit dan graphics processing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



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

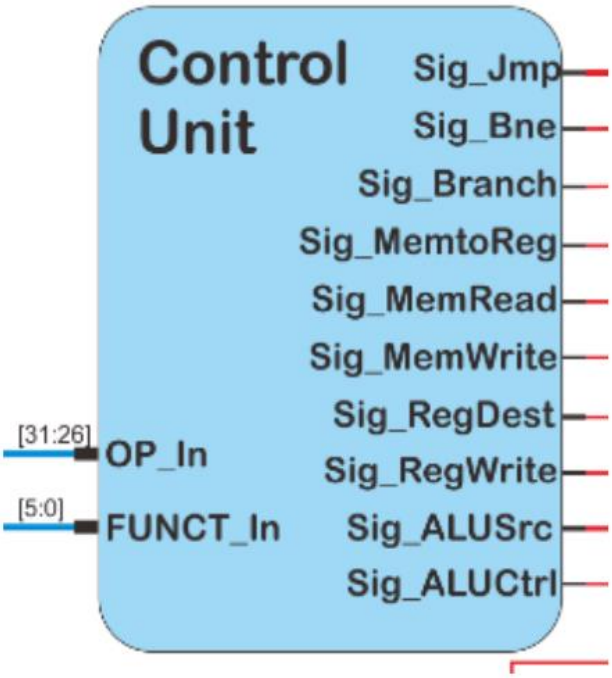
blok adder. Sedangkan untuk operasi pengurangan, ALU memanfaatkan sifat bilangan two's complement. Dengan demikian, pengurangan merupakan penjumlahan dengan bilangan negatif. Oleh karena itu, operand kedua dapat diubah menjadi bilangan negatif dengan memanfaatkan prinsip two's complement yaitu rumus $-X = \sim X + 1$. 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.



B. Control Unit (CU)

Control Unit (CU) merupakan komponen dari sebuah mikroprosesor yang berfungsi untuk mengarahkan operasioperasi 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.



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 sinyalsinyal 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.

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

Untuk mengatur sinyal-sinyal kontrol tersebut, control unit mendeteksi setiap instruksi menggunakan 64 opcode dan funct.

III. Hasil dan Analisis

Setelah melakukan percobaan pada semua tugas didapatkan hasil sebagai brerikut:

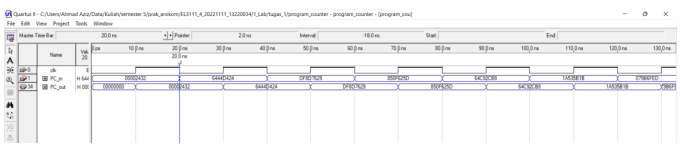
A. Tugas 1: Perancangan Program Counter

Program counter dalam hal ini untuk implementasi pada arsitektur MIPS32 berfungsi sebagai komponen register untuk meregist alamat instruksi yang akan dieksekusi sehingga sistem dapat mendeteksi jenis instruksi yang akan dieksekusi. Program counter terdiri atas flip flop yang berjenis D flip flop untuk dapat meregist input yang masuk pada rising edge. Pada percobaan ini dibuat program counter dengan lebar 32 bit sehingga dibutuhkan 32 flip flop.

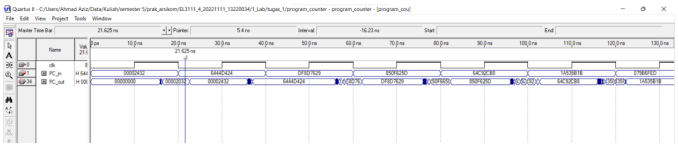
Implementasi program counter pada VHDL cukup sederhana yaitu dengan membuat output mengeluarkan input pada rising edge clock, sama seperti D flip flop. Berikut adalah implementasi program counter dalam VHDL

```
ARCHITECTURE behavior OF program_counter
IS
  BEGIN
    PROCESS (clk,PC_in)
      BEGIN
      IF (rising_edge(clk)) THEN
        PC_out <= PC_in;
      END IF;
    END PROCESS;
END behavior;
```

Kode VHDL berhasil decompile tanpa error sehingga dapat disimulasikan. Berikut ini adalah hasil simulasi program counter secara functional



Dan berikut hasil simulasi timing dari program counter



Dapat dilihat pada hasil simulasi tersebut, output dikeluarkan sesuai dengan input pada rising edge clock. Nilai PC_in degenerate secara random dengan representasi hexadecimal. Dengan hasil simulasi tersebut dikonfirmasi bahwa implementasi program counter berjalan dengan baik dan sesuai dengan fungsinya.

B. Tugas 2: Perancangan Left Shifter Dua Kali

Pada percobaan kedua ini dilakukan implementasi perancangan left shifter dua kali. Left shift merupakan komponen yang berfungsi untuk menggeser bit ke kiri, dalam hal ini menggeser 2 kali. Ada 2 left shifter yang dibuat pada percobaan ini yaitu 32 bit dan 26 bit dengan output yang diextend jadi 28 bit.

Implementasi left shifter dalam VHDL adalah dengan memasukkan jumlah bit LSB yang masuk dalam range shift pada input ke MSB pada output sehingga 2 bit terakhir pada MSB akan hilang atau tergeser. Selanjutnya tambahkan bit 00 pada LSB sehingga lebar data tetap sama.

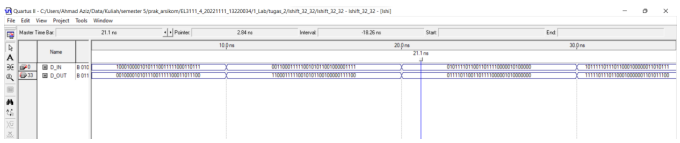
Berikut ini adalah implementasi left shifter pada VHDL untuk 32 bit

```
ARCHITECTURE behavior OF lshift_32_32 IS
  BEGIN
  PROCESS (D_IN)
    BEGIN
    D_OUT(31 DOWNTO 2) <= D_IN(29 DOWNTO
0);
    D_OUT(1 DOWNTO 0) <= "00";
  END PROCESS;
END behavior;
```

Dan untuk 26 bit dilakukan dengan cara yang sama, namun outputnya berbeda dimana diextend sebanyak 2 bit dari 26 bit jadi 28 bit. Jadi secara sederhana shifter ini hanya menambahkan 2 bit pada LSB. Berikut ini adalah implementasinya pada VHDL

```
ARCHITECTURE behavior OF lshift_26_28 IS
  BEGIN
  PROCESS (D_IN)
    BEGIN
    D_OUT(27 DOWNTO 2) <= D_IN(25 DOWNTO
0);
    D_OUT(1 DOWNTO 0) <= "00";
  END PROCESS;
END behavior;
```

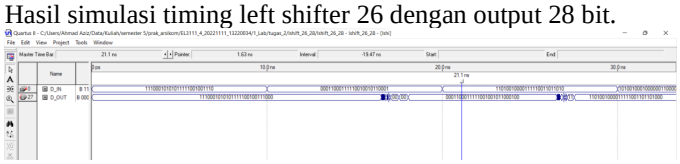
Program implementasi pada VHDL tersebut berhasil dikompilasi tanpa error sehingga implementasi sudah benar secara penulisan kode dan implementasinya. Selanjutnya kode tersebut dapat disimulasikan untuk melihat hasilnya. Berikut ini adalah hasil simulasi program left shift 32 bit secara functional



Dan berikut ini adalah hasil simulasi shifter 32 bit dengan simulasi timing.

Dan berikut untuk left shift 26 bit dengan output 28 bit secara functional

Hasil simulasi timing left shifter 26 dengan output 28 bit.

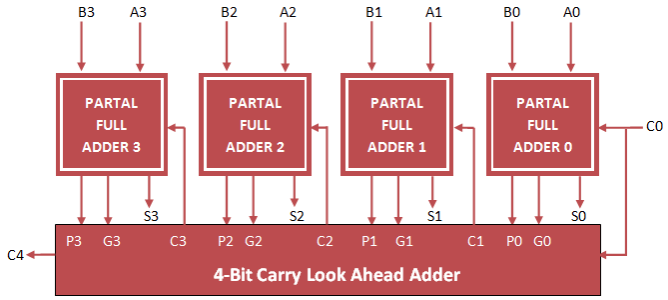


Dari hasil simulasi yang direpresentasikan dalam binary tersebut dapat dilihat data bergeser ke kiri sebanyak 2 bit dengan nilai yang masuk pada LSB yaitu bernilai 00. Dari hasil simulasi tersebut implementasi left shifter pada VHDL bekerja dengan baik dan sesuai dengan apa yang diinginkan.

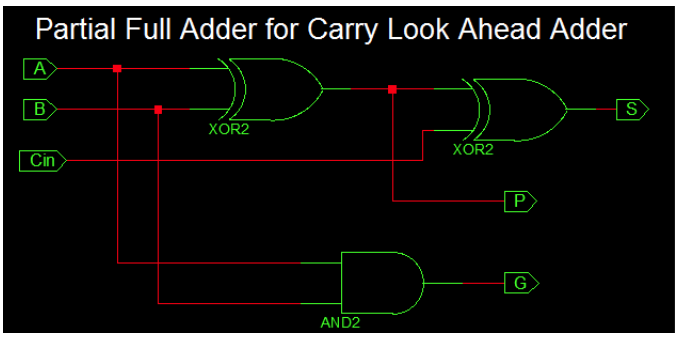
C. Tugas 3: Perancangan Carry-Lookahead Adder 32-bit

Pada tugas 3 dilakukan perancangan Carry-Lookahead Adder atau CLA dengan lebar data 32 bit. CLA merupakan komponen untuk melakukan penjumlahan 2 input dengan konfigurasi Carry-Lookahead. Komponen ini juga menerima 1 bit carry yang akan berguna dari operasi sebelumnya jika terdapat carry.

CLA dibuat dengan full adder sebanyak jumlah bitnya namun dengan mengambil propagate dan generate dalam setiap addernya sehingga proses operasi tidak perlu menunggu satu persatu operasi selesai sehingga operasi lebih cepat. Berikut ini adalah diagram komponen CLA



Dengan komponen full adder diagram logic gatenya sebagai berikut

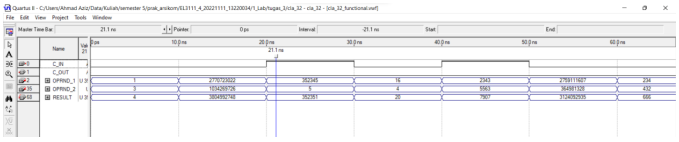


Komponen CLA dapat diimplementasikan dalam VHDL sebagai berikut

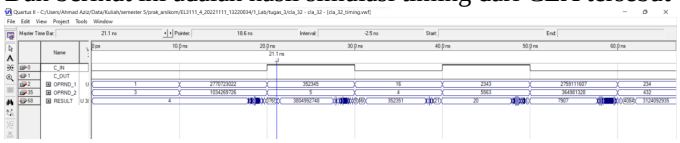
```
ARCHITECTURE behavior OF cla_32 IS
  SIGNAL SUM  :    STD_LOGIC_VECTOR(31
DOWNTO 0); -- Sum
  SIGNAL G    :    STD_LOGIC_VECTOR(31
DOWNTO 0); --generate
  SIGNAL P    :    STD_LOGIC_VECTOR(31
DOWNTO 0);--propagate
  SIGNAL C    :    STD_LOGIC_VECTOR(31
DOWNTO 1); --carry

  BEGIN
  SUM<= OPRND_1 XOR OPRND_2;
  G   <= OPRND_1 AND OPRND_2;
  P   <= OPRND_1 OR OPRND_2;
  PROCESS (OPRND_1, OPRND_2, C_IN)
    BEGIN
    C(1) <= G(0) OR (P(0) AND C_IN);
        FOR i IN 1 TO 30 LOOP
          C(i+1) <= G(i) OR (P(i) AND
C(i));
        END LOOP;
      C_OUT <= G(31) OR (P(31) AND C(31));
      RESULT(0) <= SUM(0) XOR C_IN;
      RESULT(31 DOWNTO 1) <= SUM(31 DOWNTO
1) XOR C(31 DOWNTO 1);
  END PROCESS;
END behavior;
```

Pada implementasinya tersebut, dideklarasikan sinyal untuk menyimpan value dari generate, propagate, dan juga carry secara sementara sebelum dikeluarkan ke output karena tidak menggunakan port khusus untuk sinyal tersebut. Sinyal input selanjutnya diproses pada full adder sesuai dengan rangkaian gate untuk full adder kemudian carry, popagate dan generatenya disimpan pada sinyal yang sudah dideklarasikan. Selanjutnya dilakukan looping untuk semua bit pada data yang dalam kasus ini berjumlah 32 bit.

Program tersebut dapat dikompilasi tanpa error pada software sehingga program sudah benar dan dapat disimulasikan. Berikut ini adalah hasil simulasi CLA 32 bit secara functional



Dan berikut ini adalah hasil simulasi timing dari CLA tersebut



Dari hasil simulasi tersebut, dapat dilihat hasil penjumlah bernilai benar secara functional dan timing meskipun pada simulasi timing terdapat pergeseran output yang cukup signifikan, namun hasil penjumlahannya masih bernilai benar. Sehingga implementasi CLA pada VHDL berjalan dengan benar.

D. Tugas 4: Sign Extender

Pada percobaan tugas ini akana dilakukan implementasi sign extender. Sign extender merupakan komponen untuk menambah lebar data dengan representasi dengan bilangan signed dalam hal ini yaitu 2's complement. Sign extender akan menambah bit pada MSB data tanpa mengubah sign atau nilai dari data yang dimasukkan.

Implementasi sign extender pada VHDL dengan representasi bilangan 2's complement dapat dilakukan dengan menambahkan bit complent dari data input sehingga tidak mengubah datanya.

Berikut ini adalah implementasi sign extender dalam bahasa VHDL

```
ARCHITECTURE behavior OF sign_extender IS
  BEGIN
  PROCESS (D_in)
    BEGIN
    D_out(31 DOWNTO 16) <=
"1111111111111111";
    D_out(15 DOWNTO 0) <= D_IN(15 DOWNTO
0);
  END PROCESS;
END behavior;
```

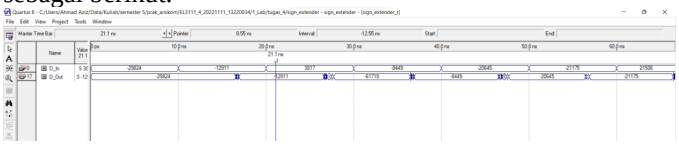
Karena jumlah bit data terdefinisi, maka kita dapat langsung menambahkan nilai 1 pada belakang MSB data yang akan kita extend. Program tersebut behasil dikompilasi dengan sempurna sehingga kode tersebut berjalan dengan baik dan dapat dilakukan simulasi pada kode tersebut

Berikut ini adalah hasil simulasi sign extender secara functional dalam representasi desimal

Dapat dilihat dari hasil simulasi tersebut nilai input dan output tidak mengalami perubahan nilai sehingga sign extender tersebut berjalan dengan benar. Selanjutnya kita lihat hasilanya dalam representasi binary untuk melihat apakah nilai output sudah mengalami extend, Berikut ini adalah hasil simulasi functional dalam representasi binary.

Dapat dilihat dari hasil simulasi tersebut nilai LSB pada output bernilai sama dengan input dengan pada MSBnya terdapat penambahan atau pelebaran data tanpa mengubah nilainya dalam ini dengan 2's complement ditambahkan bit 1.

Kemudian jika disimulasikan dengan timing didapatkan hasil sebagai berikut.



Data output cukup baik dengan pergeseran hanya setengah periode dari masuknya input pada sistem.

E. Tugas 5: Arithmetic and Logical Unit (ALU)

Pada percobaan 5 akan dilakukan implementasi Arithmetic and Logical Unit atau ALU dengan VHDL. ALU merupakan komponen yang berfungsi untuk melakukan operasi aritmatika dan logika dari input yang dimasukkan yaitu operannya. Pada ALU juga terdapat selector yang berfungsi untuk melakukan pemilihan operasi yang akan dijalankan pada operan.

Pada percobaan ini dibuat ALU yang sederhana dengan operasi penjumlahan dan pengurangan dengan CLA yang dibuat pada percobaan sebelumnya.

Berikut ini adalah implementasi ALU sederhana dalam bahasa VHDL

```
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 OPRND2_comp : STD_LOGIC_VECTOR
(31 DOWNTO 0);

  BEGIN ADDER : cla_32
  PORT MAP(
    OPRND_1 => OPRND_1,
    OPRND_2 => OPRND2_comp,
    C_IN => OP_SEL,
    RESULT => RESULT
  );

  PROCESS (OP_SEL, OPRND_2)
    BEGIN
    IF OP_SEL = '0' THEN
      OPRND2_comp <= OPRND_2;
```

```
    ELSE
      OPRND2_comp <= not OPRND_2;
    END IF;
  END PROCESS;
END behavioral;
```

CLA pada percobaan sebelumnya tidak memiliki selector untuk melakukan pengurangan, dalam ALU ini kita memerlukan operasi pengurangan. Operasi pengurangan sendiri mudah dilakukan jika kita sudah memiliki komponen adder yaitu dengan menjadikan nilai pada operan kedua menjadi minusnya atau dikali minus 1. Sehingga dalam implementasi ini kita harus dapat menajdikan nilai operan kedua menjadi minusnya, dalam hal pada 2's complement dengan mengcomplemen semua bit dan ditambah dengan 1. Proses ini dilakukan setelah kita mengetahui operasi yang akan dilakukan, sehingga dibuat sebuah sinyal untuk mendapatkan nilai minus dari operan kedua.

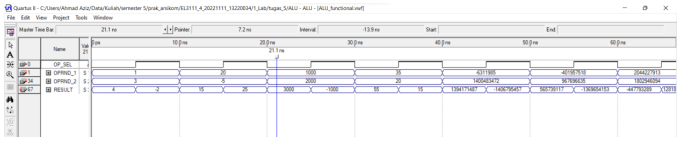
```
  SIGNAL OPRND2_comp : STD_LOGIC_VECTOR
  (31 DOWNTO 0);
```

Selanjutnya kita mengecek selector dari input selector pada ALU untuk menentukan operasi yang akan dilakukan

```
  IF OP_SEL = '0' THEN
      OPRND2_comp <= OPRND_2;
  ELSE
      OPRND2_comp <= not OPRND_2;
  END IF;
```

Kode implementasi dalam VHDL tersebut berhasil dikompilasi sehingga kode tersebut sudah benar dan dapat dilakukan simulasi.

Berikut ini adalah hasil simulasi ALU yang sudah dibuat secara functional



Dan berikut ini adalah hasil simulasi timing pada ALU

Dari hasil simulasi tersebut dapat dilihat dalam representasi decimal nilai output atau hasil operasi bernilai benar. Untuk selector dari simulasi ini dibuat dengan clock sederhana dengan periode sama dengan pergantian nilai input sehingga untuk setiap nilai input kita dapat melihat hasil outputnya untuk operasi penjumlahan dan pengurangan. Operasi penjumlahan dilakukan ketika selector bernilai 0 dan ketika bernilai 1 akan dilakukan operasi pengurangan.

F. Tugas 6: Control Unit (CU)

Pada percobaan ke 6 yaitu implementasi Control Unit atau CU. Control unit adalah komponen untuk melakukan control terhadap instruksi dan operasi yang akan dilakukan. Control unit bekerja dengan input dari opcode dan function input dari komponen. Output dari control unit ini akan masuk ke komponen-komponen lain yang digunakan seperti komponen adder yang menggunakan sinyal output tipe R.

Pada percobaan ini akan dibuat Control unit sederhana dengan 10 function saja. Implementasi CU pada percobaan ini cukup sederhana dengan mengeluarkan output berdasarkan input opcode dan function yang diberikan pada input. Sehingga dalam VHDL dapat dilakukan dengan switch case sederhana yang implementasinya ada pada kode berikut

```
ARCHITECTURE behavior OF cu IS
  SIGNAL OUT_sig : STD_LOGIC_VECTOR(10
DOWNTO 0);
  BEGIN
  PROCESS (OP_In, FUNCT_In) IS
    BEGIN
    IF OP_In = "000000" THEN
      CASE FUNCT_In IS
        WHEN "100000"  => OUT_sig <=
"00000001100"; -- ADD
        WHEN "100010"  => OUT_sig <=
"00000001101"; -- SUB
        WHEN OTHERS   => OUT_sig <=
"00000000000"; -- NOP
      END CASE;
    ELSE
      CASE OP_In IS
        WHEN "000100"  => OUT_sig <=
"00100000000"; -- BEQ
        WHEN "000101"  => OUT_sig <=
"01000000010"; -- BNE
        WHEN "001000"  => OUT_sig <=
"00000000110"; -- ADD
        WHEN "100011"  => OUT_sig <=
"00011000110"; -- LW
        WHEN "101011"  => OUT_sig <=
"00000100010"; -- SW
        WHEN "000010"  => OUT_sig <=
"10000000000"; -- JMP
        WHEN others   => OUT_sig <=
"00000000000"; -- NOP
      END CASE;
    END IF;

    Sig_Jmp     <= OUT_sig(10);
    Sig_Bne     <= OUT_sig(9);
    Sig_Branch  <= OUT_sig(8);
    Sig_MemtoReg <= OUT_sig(7);
    Sig_MemRead  <= OUT_sig(6);
    Sig_MemWrite <= OUT_sig(5);
    Sig_RegDest  <= OUT_sig(4 DOWNTO 3);
    Sig_RegWrite <= OUT_sig(2);
    Sig_ALUSrc  <= OUT_sig(1);
    Sig_ALUCtrl  <= OUT_sig(0);
  END PROCESS;
END behavior;
```

Pada implementasi VHDL tersebut dideklarasikan sebuah sinyal dalam bentuk vector 10 bit untuk menampung nilai dari output sebelum dikeluarkan. Selanjutnya ada 2 case yang perlu dicek untuk kondisi opcodenya ketika bernilai 0 pada operasi tipe R. Kemudian dilakukan switch case berdasarkan input functionnya dan nilai dari sinyal output dapat di define sesuai dengan table fungsinya untuk arsitektur MIPS32.

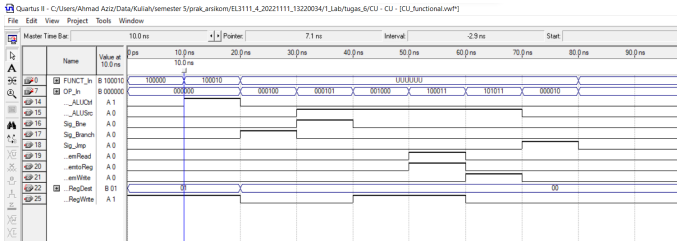
```
IF OP_In = "000000" THEN
     CASE FUNCT_In IS
       WHEN "100000"   => OUT_sig <=
"00000001100"; -- ADD
       WHEN "100010"   => OUT_sig <=
"00000001101"; -- SUB
       WHEN OTHERS    => OUT_sig <=
"00000000000"; -- NOP
     END CASE;
```

Selanjutnya untuk kasus selain opcode bernilai 0 yaitu operasi selain tipe R dengan cara yang sama dengan switch case.

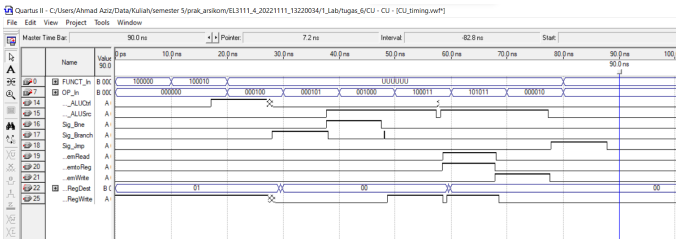
```
   ELSE
     CASE OP_In IS
       WHEN "000100"   => OUT_sig <=
"00100000000"; -- BEQ
       WHEN "000101"   => OUT_sig <=
"01000000010"; -- BNE
       WHEN "001000"   => OUT_sig <=
"00000000110"; -- ADD
       WHEN "100011"   => OUT_sig <=
"00011000110"; -- LW
       WHEN "101011"   => OUT_sig <=
"00000100010"; -- SW
       WHEN "000010"   => OUT_sig <=
"10000000000"; -- JMP
       WHEN others    => OUT_sig <=
"00000000000"; -- NOP
     END CASE;
   END IF;
```

Setelah sinyal output didefine dengan input dari opcode dan functionnya, nilai sinyal output dari vectornya dipecah untuk masing masih output. Dengan cara ini kita dapat menghemat memory dan kode yang lebih ringkas.

Kode implementasi dalam VHDL ini berhasil dikompilasi tanpa error sehingga dapat disimulasikan. Berikut ini adalah hasil simulasi Control unit secara functional



Dan berikut ini adalah hasil simulasi timing dari control unitnya



Dari hasil simulasi tersebut, dapat dilihat bahwa outputnya sesuai dengan seharusnya yang terdapat pada table opcode dan function. Dengan hasil simulasi tersebut dapat disimpulkan control unit yang diimplementasikan sudah bekerja dengan baik sebagaimana mestinya.

IV. SIMPULAN

- Control unit mendapatkan input dari function dan opcode untuk menentukan outputnya yang akan digunakan untuk melakukan eksekusi perintah yang akan dijalankan.
- Control unit memiliki nilai yang sudah didefinisikan, dan akan bernilai 0 jika tidak terdapat dalam nilai terdefinisi yang akan dianggap sebagai jeda dalam siklusnya.
- Adder dengan jenis CLA dapat diimplementasikan dengan full adder dengan menambahkan sinyal carry, propagate dan generate dalam sistem secara parallel dalam setiap komponen full addernya.
- CLA dapat melakukan operasi penjumlahan lebih cepa dengan RCA karena adanya sinyal propagate untuk carry setiap addernya sehingga tidak perlu menunggu hasil operasi dari masing-masing full addernya.
- Operasi pengurangan dapat dilakukan dengan komponen adder yaitu dengan mengubah operan kedua menjadi negasinya atau dikalikan dengan minus 1.
- Dalam VHDL, sinyal dalam bentuk vector dapat dimanfaatkan layaknya array dalam bahasa tingkat tinggi sebagai penyiimpan variable dalam proses pengolahan sinyal yang kemudian dipecah setiap bitnya sesuai dengan format yang ditentukan sehingga dapat menghemat memory dengan program yang ringkas.

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

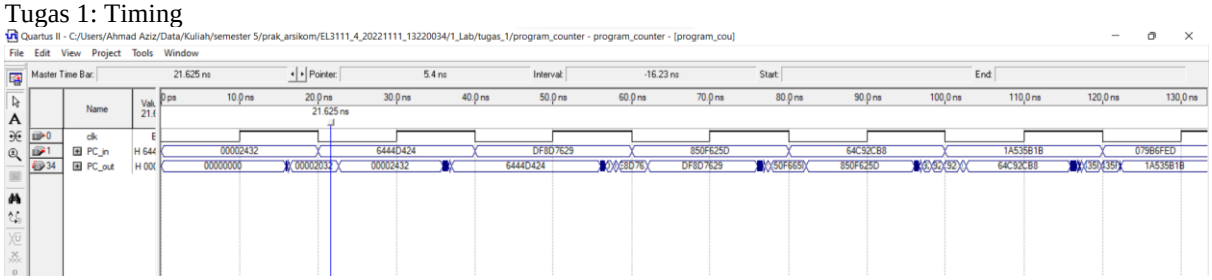
1) *Source code dan simulasi untuk tugas 1*

program_counter.vhd

```
-- Praktikum EL3111 Arsitektur Sistem Komputer
-- Modul    : 4
-- Percobaan  : 1
-- Tanggal    : 11 November 2022
-- Kelompok   : 10
-- Rombongan  : B
-- Nama (NIM) 1 : Ahmad Aziz (13220034)
-- Nama (NIM) 2 :
-- Nama File  : program_counter.vhd
-- Deskripsi  : Program Counter

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

ENTITY program_counter IS
PORT (
    clk : IN STD_LOGIC;
    PC_in : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
    PC_out : OUT STD_LOGIC_VECTOR (31 DOWNTO 0)
);
END program_counter;

ARCHITECTURE behavior OF program_counter IS
  BEGIN
    PROCESS (clk,PC_in)
      BEGIN
      IF (rising_edge(clk)) THEN
        PC_out <= PC_in;
      END IF;
    END PROCESS;
END behavior;
```

Tugas 1: Functional

Tugas 1: Timing



2) *Source code untuk tugas 2*

lshift_26_28.vhd

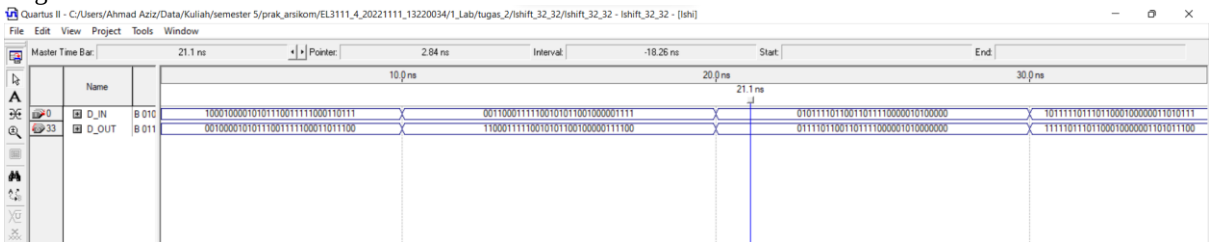
```
-- Praktikum EL3111 Arsitektur Sistem Komputer
-- Modul    : 4
-- Percobaan  : 2b
-- Tanggal    : 11 November 2022
-- Kelompok   : 10
-- Rombongan  : B
-- Nama (NIM) 1 : Ahmad Aziz (13220034)
-- Nama (NIM) 2 :
-- Nama File  : lshift_26_28.vhd
-- Deskripsi  : Program untuk melakukan operasi shift left pada 26 bit dengan
output 28 bit

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.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
  PROCESS (D_IN)
    BEGIN
    D_OUT(27 DOWNTO 2) <= D_IN(25 DOWNTO 0);
    D_OUT(1 DOWNTO 0) <= "00";
  END PROCESS;
END behavior;
```

lshift_32_32.vhd

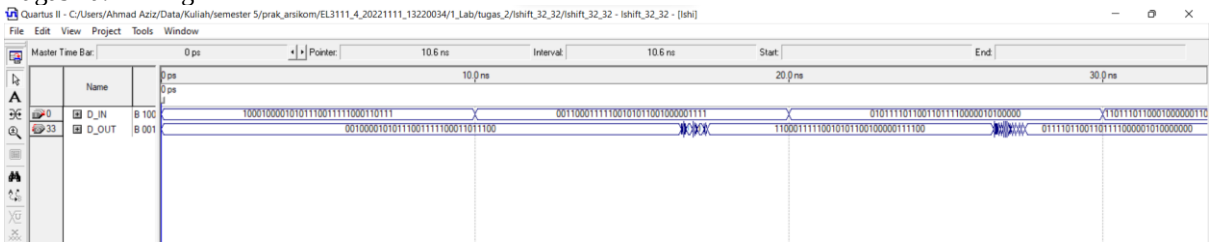
```
-- Praktikum EL3111 Arsitektur Sistem Komputer
-- Modul    : 4
-- Percobaan  : 2a
-- Tanggal    : 11 November 2022
-- Kelompok   : 10
-- Rombongan  : B
-- Nama (NIM) 1 : Ahmad Aziz (13220034)
-- Nama (NIM) 2 :
-- Nama File  : lshift_32_32.vhd
-- Deskripsi  : Program untuk melakukan operasi shift left 32 bit

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.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
  PROCESS (D_IN)
    BEGIN
    D_OUT(31 DOWNTO 2) <= D_IN(29 DOWNTO 0);
    D_OUT(1 DOWNTO 0) <= "00";
  END PROCESS;
END behavior;
```

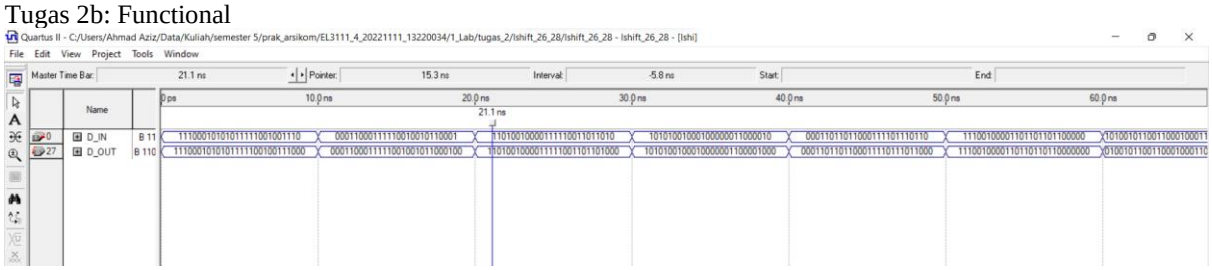
Tugas 2a: Functional



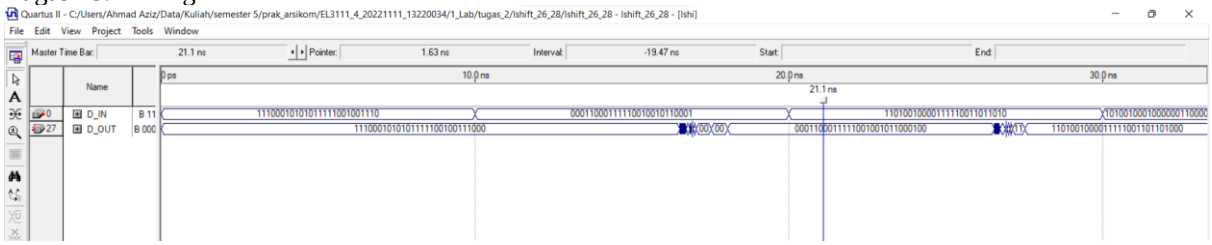
Tugas 2a: Timing



Tugas 2b: Functional



Tugas 2b: Timing

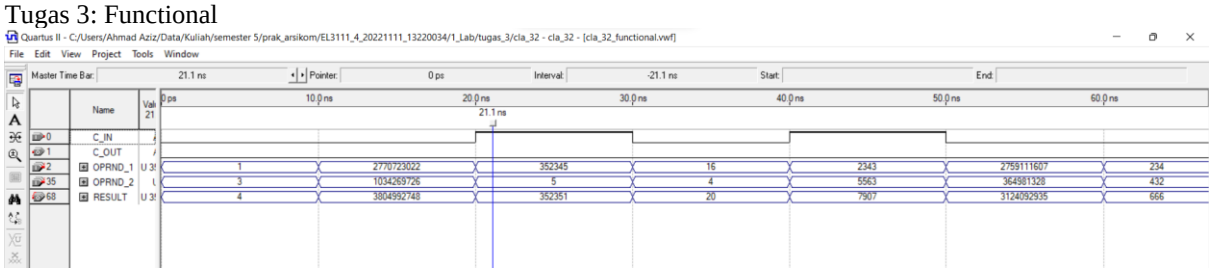


3) *Source code untuk tugas 3*

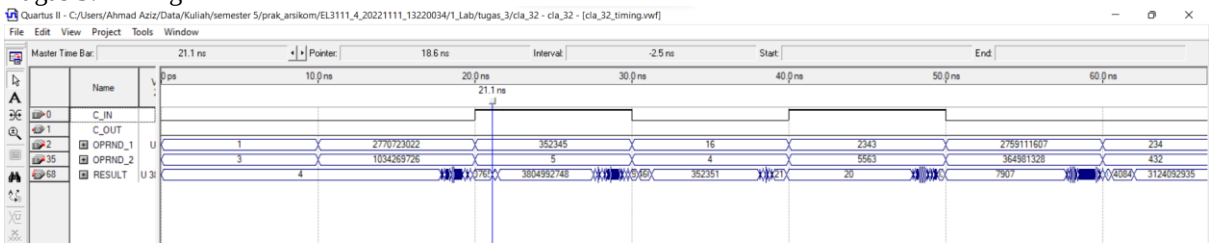
cla_32.vhd

```
-- Praktikum EL3111 Arsitektur Sistem Komputer
-- Modul    : 4
-- Percobaan  : 3
-- Tanggal    : 11 November 2022
-- Kelompok   : 10
-- Rombongan  : B
-- Nama (NIM) 1 : Ahmad Aziz (13220034)
-- Nama (NIM) 2 :
-- Nama File  : cla_32.vhd
-- Deskripsi  : Carry Look Ahead Adder 32 bit

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.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 behavior OF cla_32 IS
  SIGNAL SUM   :     STD_LOGIC_VECTOR(31 DOWNTO 0); -- Sum
  SIGNAL G     :     STD_LOGIC_VECTOR(31 DOWNTO 0); --generate
  SIGNAL P     :     STD_LOGIC_VECTOR(31 DOWNTO 0);--propagate
  SIGNAL C   :     STD_LOGIC_VECTOR(31 DOWNTO 1); --carry

  BEGIN
  SUM<= OPRND_1 XOR OPRND_2;
  G   <= OPRND_1 AND OPRND_2;
  P   <= OPRND_1 OR OPRND_2;
  PROCESS (OPRND_1, OPRND_2, C_IN)
    BEGIN
    C(1) <= G(0) OR (P(0) AND C_IN);
        FOR i IN 1 TO 30 LOOP
          C(i+1) <= G(i) OR (P(i) AND C(i));
        END LOOP;
      C_OUT <= G(31) OR (P(31) AND C(31));
      RESULT(0) <= SUM(0) XOR C_IN;
      RESULT(31 DOWNTO 1) <= SUM(31 DOWNTO 1) XOR C(31 DOWNTO 1);
  END PROCESS;
END behavior;
```

Tugas 3: Functional



Tugas 3: Timing



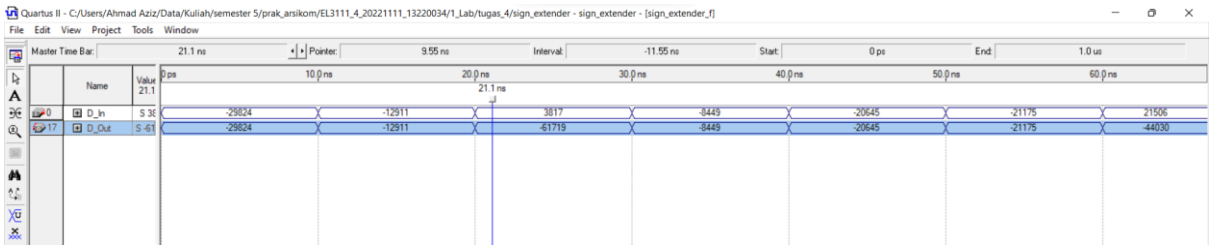
4) *Source code untuk tugas 4*

reverseByte.h

```
-- Praktikum EL3111 Arsitektur Sistem Komputer
-- Modul    : 4
-- Percobaan  : 4
-- Tanggal    : 11 November 2022
-- Kelompok   : 10
-- Rombongan  : B
-- Nama (NIM) 1 : Ahmad Aziz (13220034)
-- Nama (NIM) 2 :
-- Nama File  : sign_extender.vhd
-- Deskripsi  : Program sign extender pada arsitektur MIPS

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.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 behavior OF sign_extender IS
  BEGIN
  PROCESS (D_in)
    BEGIN
    D_out(31 DOWNTO 16) <= "1111111111111111";
    D_out(15 DOWNTO 0) <= D_IN(15 DOWNTO 0);
  END PROCESS;
END behavior;
```

Tugas 4: Functional

Decimal



Binary

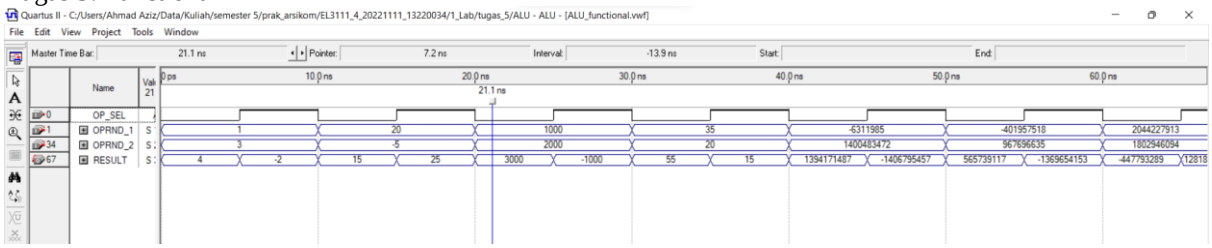
Tugas 4: Timing

5) *Source code untuk tugas 5*

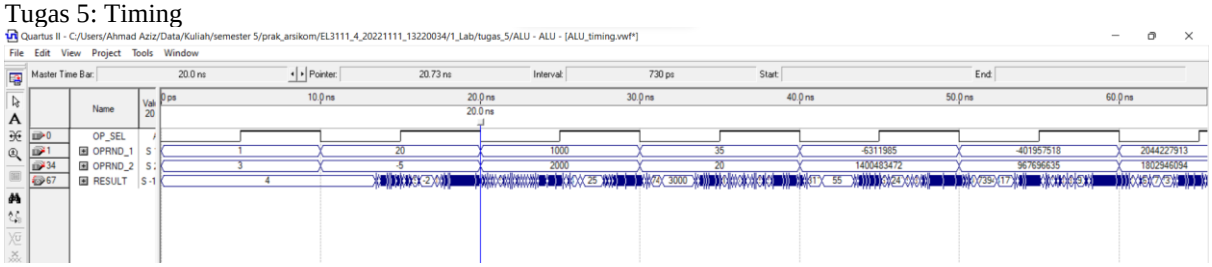
ALU.vhd

```
-- Praktikum EL3111 Arsitektur Sistem Komputer
-- Modul    : 4
-- Percobaan  : 5
-- Tanggal    : 11 November 2022
-- Kelompok   : 10
-- Rombongan  : B
-- Nama (NIM) 1 : Ahmad Aziz (13220034)
-- Nama (NIM) 2 :
-- Nama File  : ALU.vhd
-- Deskripsi  : Program Arithmetic Logic Unit

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.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 OPRND2_comp : STD_LOGIC_VECTOR (31 DOWNTO 0);

  BEGIN ADDER : cla_32
  PORT MAP(
    OPRND_1 => OPRND_1,
    OPRND_2 => OPRND2_comp,
    C_IN => OP_SEL,
    RESULT => RESULT
  );

  PROCESS (OP_SEL, OPRND_2)
    BEGIN
    IF OP_SEL = '0' THEN
      OPRND2_comp <= OPRND_2;
    ELSE
      OPRND2_comp <= not OPRND_2;
    END IF;
  END PROCESS;
END behavioral;
```

Tugas 5: Functional



Tugas 5: Timing



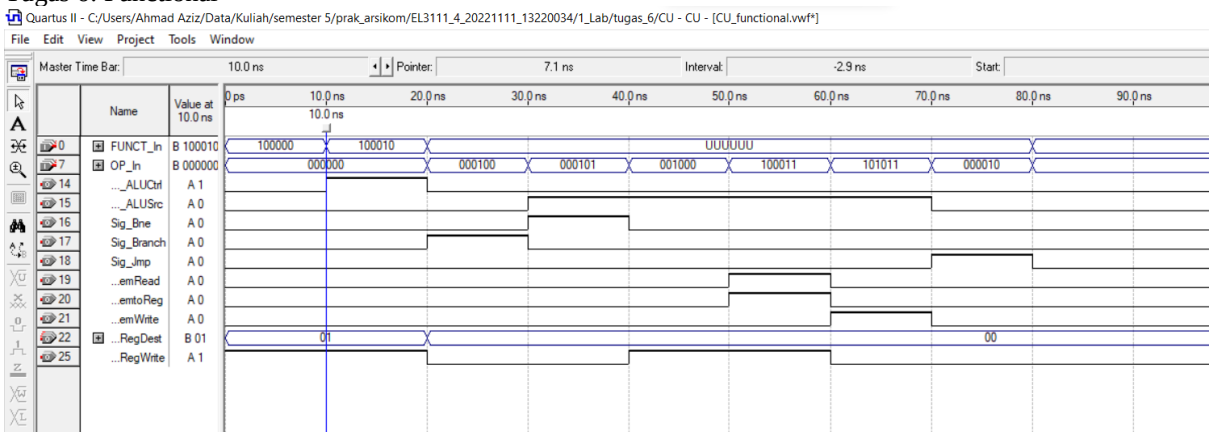
6) *Source code untuk tugas 6*

CU.vhd

```
-- Praktikum EL3111 Arsitektur Sistem Komputer
-- Modul    : 4
-- Percobaan  : 6
-- Tanggal    : 11 November 2022
-- Kelompok   : 10
-- Rombongan  : B
-- Nama (NIM) 1 : Ahmad Aziz (13220034)
-- Nama (NIM) 2 :
-- Nama File  : CU.vhd
-- Deskripsi  : Program control unit (CU)

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.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;
  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;
  Sig_ALUCtrl : OUT STD_LOGIC
  );
END cu;

ARCHITECTURE behavior OF cu IS
  SIGNAL OUT_sig : STD_LOGIC_VECTOR(10 DOWNTO 0);
  BEGIN
  PROCESS (OP_In, FUNCT_In) IS
    BEGIN
    IF OP_In = "000000" THEN
      CASE FUNCT_In IS
        WHEN "100000"   => OUT_sig <= "00000001100"; -- ADD
        WHEN "100010"   => OUT_sig <= "00000001101"; -- SUB
        WHEN OTHERS   => OUT_sig <= "00000000000"; -- NOP
      END CASE;
    ELSE
      CASE OP_In IS
        WHEN "000100"   => OUT_sig <= "00100000000"; -- BEQ
        WHEN "000101"   => OUT_sig <= "01000000010"; -- BNE
        WHEN "001000"   => OUT_sig <= "00000000110"; -- ADD
        WHEN "100011"   => OUT_sig <= "00011000110"; -- LW
        WHEN "101011"   => OUT_sig <= "00000100010"; -- SW
        WHEN "000010"   => OUT_sig <= "10000000000"; -- JMP
        WHEN others   => OUT_sig <= "00000000000"; -- NOP
      END CASE;
    END IF;

    Sig_Jmp     <= OUT_sig(10);
    Sig_Bne     <= OUT_sig(9);
    Sig_Branch  <= OUT_sig(8);
    Sig_MemtoReg  <= OUT_sig(7);
    Sig_MemRead   <= OUT_sig(6);
    Sig_MemWrite  <= OUT_sig(5);
```

```
    Sig_RegDest   <= OUT_sig(4 DOWNTO 3);
    Sig_RegWrite  <= OUT_sig(2);
    Sig_ALUSrc    <= OUT_sig(1);
    Sig_ALUCtrl   <= OUT_sig(0);
  END PROCESS;
END behavior;
```

Tugas 6: Functional



Tugas 6: Timing