LAMPIRAN-LAMPIRAN

Deskripsi Jenis-jenis Instruksi Mikro

Penjumlahan *Register* dengan *Register*

Mnemonic : ADD
Operand : RD, RA, RB
Format : ADD RD, RA, RB
Operasi : RD ← RA + RB
Tujuan : Melakukan operasi penjumlahan antara *register* dengan *register*.
Perancangan : Dapat menggunakan skematik *ripple carry adder*, *carry lookahead adder*, *carry selector adder*, maupun menggunakan *library* yang telah disediakan. Perancangan skematik diprogram menggunakan VHDL dengan metoda perancangan *structural*. Sedangkan perancangan dengan *library* menggunakan metoda perancangan *behavioral*.

Penjumlahan *Register* dengan *Immediate Sign*

Mnemonic : ADI
Operand : RD, RA, Imm
Format : ADI RD, RA, Imm
Operasi : RD ← RA + Imm(S)
Tujuan : Melakukan operasi penjumlahan antara *register* dengan *immediate* bertanda.
Deskripsi : Jangkauan nilai *immediate* antara -32768 sampai dengan 32767.
Perancangan : Nilai *immediate* diambil dari instruksi sebesar 16 bit LSB. Karena merupakan bilangan bertanada, maka 16 bit MSB harus bernilai sama dengan bit ke 16 LSB agar tanda dan nilai dari bilangan tersebut sama saat dikonfersi dari 16 bit ke 32 bit. Untuk itu dibangun sebuah komponen yang melakukan tugas tersebut yang disebut *Constant Unit*. Satu bit (CS) sebagai *input* pada *Constant Unit* berfungsi untuk memberitahu apakan nilai *immediate* yang masuk tersebut merupakan bilangan bertanda atau tidak.

Penjumlahan *Register* dengan *Immediate Unsign*

Mnemonic : ADIU
Operand : RD, RA, Imm
Format : ADIU RD, RA, Imm
Operasi : RD ← RA + Imm(U)
Tujuan : Melakukan operasi penjumlahan antara *register* dengan *immediate* tidak bertanda.
Deskripsi : Jangkauan nilai *immediate* antara 0 sampai dengan 65535.
Perancangan : Nilai *immediate* diambil dari instruksi sebesar 16 bit LSB dan untuk nilai MSB-nya diisi dengan nol.

Pengurangan *Register* dengan *Register*

Mnemonic : SUB
Operand : DA, RA, RB
Format : SUB RD, RA, RB
Operasi : RD ← RA - RB
Tujuan : Melakukan pengurangan antara *register* dengan *register*
Perancangan : Untuk instruksi pengurangan, digunakan metode *two's complement* pada nilai yang akan mengurangkan (RB) agar nilai tersebut menjadi negatif, selanjutnya nilai tersebut akan kita tambahkan dengan nilai yang akan dikurangkan (RA) dan hasilnya disimpan pada *register* tujuan (RD).

$RD \leftarrow RA - RB$
$RD \leftarrow RA + (-RB)$
$RD \leftarrow RA + (\overline{RB} + 1)$

Oleh karena itu dibutuhkan penjumlahan yang memiliki *carry in* dimana nilainya sama dengan satu pada operasi pengurangan dan nol pada operasi penjumlahan. Kemudian dibutuhkan satu bit lagi untuk menetukan apakah RB akan dikomplemen atau tidak. Bit ini disebut *SubAdd*. Jika *SubAdd* bernilai satu, maka RB akan dikomplemen, jika nol maka RB tidak dikomplemen.

Pengurangan *Register* dengan *Immediate Sign*

Mnemonic : SUBI
Operand : DA, RA, Imm
Format : SUBI RD, RA, Imm
Operasi : RD ← RA – Imm(S)
Tujuan : Melakukan pengurangan antara *register* dengan *immediate* bertanda.
Deskripsi : Jangkauan nilai *immediate* antara -32768 sampai dengan 32767.
Perancangan : Nilai *immediate* diambil menggunakan komponen *Constant Unit* seperti yang dilakukan pada penjumlahan *register* dengan *immediate sign*.

Pengurangan *Register* dengan *Immediate Unsign*

Mnemonic : SUBIU
Operand : DA, RA, Imm
Format : SUBIU DA,RA, Imm
Operasi : RD ← RA – Imm(U)
Tujuan : Melakukan pengurangan antara *register* dengan *immediate* tidak bertanda.
Deskripsi : Jangkauan nilai *immediate* antara 0 sampai dengan 65535.
Perancangan : Nilai *immediate* diambil dari instruksi sebesar 16 bit LSB dan untuk nilai MSB-nya diisi dengan nol.

Logika AND

Mnemonic : AND
Operand : DA, RA, RB
Format : AND DA, RA, RB
Operasi : DA ← RA ^ RB
Tujuan : Menghasilkan logika AND dari dua *input register*. Digunakan juga untuk seleksi bit.
Perancangan : Menggunakan gerbang logika AND untuk kedua *input*.

Logika AND *Immediate*

Mnemonic : ANDI
Operand : DA, RA, Imm
Format : ANDI DA, RA, Imm
Operasi : DA ← RA ^ Imm
Tujuan : Menghasilkan logika AND dari *input ragister* dan *immediate*. Digunakan juga untuk seleksi bit.
Perancangan : Nilai *immediate* juga berasal dari *Constant Unit*. 16 bit MSB-nya diisi dengan nol. Kemudian digunakan gerbang AND untuk nilai *immediate* yang telah dirubah menjadi nilai 32 bit dan nilai dari *register*.

Logika OR

Mnemonic : OR
Operand : DA, RA, RB
Format : OR DA, RA, RB
Operasi : DA ← RA ^{V} RB
Tujuan : Menghasilkan logika OR dari dua *input register*.
Perancangan : Menggunakan gerbang logika OR untuk kedua *input*.

Logika OR *Immediate*

Mnemonic : ORI
Operand : DA, RA, Imm
Format : ORI DA, RA, Imm
Operasi : DA ← RA ^{V} Imm
Tujuan : Menghasilkan logika OR dari *input ragister* dan *immediate*.
Perancangan : Nilai *immediate* juga berasal dari *Constant Unit*. 16 bit MSB-nya diisi dengan nol. Kemudian digunakan gerbang OR untuk nilai *immediate* yang telah dirubah menjadi nilai 32 bit dan nilai dari *register*.

Logika *Exclusive* OR

Mnemonic : XOR
Operand : DA, RA, RB
Format : XOR DA ,RA, RB

Operasi : DA ← RA ⊕ RB
Tujuan : Menghasilkan logika XOR dari dua *input register*. Digunakan juga untuk melakukan togel bit.
Perancangan : Menggunakan gerbang logika XOR untuk kedua *input*.

Logika *Exclusive* OR Immediate

Mnemonic : XORI
Operand : DA, RA, Imm
Format : XORI DA, RA, Imm
Operasi : DA ← RA ⊕ Imm
Tujuan : Menghasilkan logika XOR dari *input ragister* dan *immediate*. Digunakan juga untuk melakukan togel bit.
Perancangan : Nilai *immediate* juga berasal dari *Constant Unit*. 16 bit MSB-nya diisi dengan nol. Kemudian digunakan gerbang XOR untuk nilai *immediate* yang telah dirubah menjadi nilai 32 bit dan nilai dari *register*.

Logika *Not* OR

Mnemonic : NOR
Operand : DA, RA, RB
Format : NOR DA, RA, RB
Operasi : DA ← $\overline{RA \vee RB}$
Tujuan : Menghasilkan logika NOR dari dua *input register*. Digunakan juga untuk melakukan instruksi logika NOT.
Perancangan : Menggunakan gerbang logika NOR untuk kedua *input*.

Logika *Nor* OR Immediate

Mnemonic : NORI
Operand : DA, RA, Imm
Format : NORI DA, RA, Imm
Operasi : DA ← $\overline{RA \vee Imm}$
Tujuan : Menghasilkan logika NOR dari *input ragister* dan *immediate*. Digunakan juga melakukan instruksi logika NOT.
Perancangan : Nilai *immediate* juga berasal dari *Constant Unit*. 16 bit MSB-nya diisi dengan nol. Kemudian digunakan gerbang NOR untuk nilai *immediate* yang telah dirubah menjadi nilai 32 bit dan nilai dari *register*.

Shift Logical Right

Mnemonic : SLR
Operand : DA, RA, Imm
Format : SLR DA, RA, Imm
Operasi : DA ← sr(RA, Imm)
Tujuan : Menggeser RA ke kanan sebanyak lima bit LSB *immediate*.

Perancangan : Menggunakan *barrel shifter* maupun *shifter* biasa.

Shift Logical Right Variable

Mnemonic : SLRV
Operand : DA, RA, RB
Format : SLRV DA, RA, RB
Operasi : DA ← sr(RA, RB)
Tujuan : Menggeser RA ke kanan sebanyak lima bit LSB *register*.
Perancangan : Menggunakan *barrel shifter* maupun *shifter* biasa.

Shift Logical Left

Mnemonic : SLL
Operand : DA, RA, Imm
Format : SLL DA, RA, Imm
Operasi : DA ← sl(RA, Imm)
Tujuan : Menggeser RA ke kiri sebanyak lima bit LSB *immediate*.
Perancangan : Menggunakan *barrel shifter* maupun *shifter* biasa.

Shift Logical Left Variable

Mnemonic : SLLV
Operand : DA, RA, RB
Format : SLLV DA, RA, RB
Operasi : DA ← sl(RA, RB)
Tujuan : Menggeser RA ke kiri sebanyak lima bit LSB *register*.
Perancangan : Menggunakan *barrel shifter* maupun *shifter* biasa.

Shift Arithmetic Right

Mnemonic : SAR
Operand : DA, RA, Imm
Format : SAR DA, RA, Imm
Operasi : DA ← sar(RA, Imm)
Tujuan : Digunakan pada perancangan instruksi perkalian dengan algoritma *booth*.
Perancangan : Menggunakan *barrel shifter* maupun *shifter* biasa.

Shift Arithmetic Right Variable

Mnemonic : SARV
Operand : DA, RA, RB
Format : SARV DA, RA, RB
Operasi : DA ← sar(RA, RB)
Tujuan : Digunakan pada perancangan instruksi perkalian dengan algoritma *booth*.

Perancangan : Menggunakan *barrel shifter* maupun *shifter* biasa.

Load Upper Immediate

Mnemonic : LUI
Operand : RD, Imm
Format : LUI RD, Imm
Operasi : RD ← Imm(15:0) || X"0000"
Tujuan : Memindahkan nilai *immediate* (16 bit LSB instuksi) ke 16 bit MSB *register* yang dituju.
Perancangan : Menindahkan 16 bit LSB dari *immediate* ke dalam 16 bit MSB *register* yang dituju, sedangkan 16 bit LSB *register* yang dituju diberikan nilai 0.

Load Address

Mnemonic : LA
Operand : RD
Format : LA RD
Operasi : DA ← PC
Tujuan : Menyimpan nilai PC untuk instruksi berikutnya ke *register* yang dituju.
Perancangan : Melewatkan nilai PC untuk instruksi berikutnya melalui *function unit* ke *register* tujuan dimana *function unit* hanya berfungsi melewatkan atau melakukan operasi *pass*.

Store Byte

Mnemonic : SB
Operand : RA, RB
Format : SB RA, RB
Operasi : DataMem[RA] ← RB
Tujuan : Menyimpan data sebesar satu *byte* dari sebuah *register* ke alamat yang ditunjuk pada *data memory*.
Perancangan : Menggunakan MCO (Memory Control Out) untuk menangani cara penyimpanannya. Alamat diambil dari *bus* A dan data dari *bus* B.

Store Half Word

Mnemonic : SH
Operand : RA, RB
Format : SH RA, RB
Operasi : DataMem[RA] ← RB
Tujuan : Menyimpan data sebesar dua *byte* (setengah *word*) dari sebuah *register* ke alamat yang ditunjuk pada *data memory*.
Perancangan : Menggunakan MCO (Memory Control Out) untuk menangani cara penyimpanannya. Alamat diambil dari *bus* A dan data dari *bus* B.

Store Word

Mnemonic : SW
Operand : RA, RB
Format : SW RA, RB
Operasi : DataMem[RA] ← RB
Tujuan : Menyimpan data sebesar empat *byte* (satu *word*) dari sebuah *register* ke alamat yang ditunjuk pada *data memory*.
Perancangan : Menggunakan MCO (Memory Control Out) untuk menangani cara penyimpanannya. Alamat diambil dari *bus* A dan data dari *bus* B.

Load Byte

Mnemonic : LB
Operand : RD, RA
Format : LB RD, RA
Operasi : RD ← DataMem[RA]
Tujuan : Mengambil data sebesar satu *byte* dari alamat yang ditunjuk pada *data memory* kemudian disimpan pada *register* tujuan.
Perancangan : Menggunakan MCI (Memory Control In) untuk menangani cara penyimpanannya. Alamat diambil dari *bus* A dan data langsung menuju *bus* D.

Load Half Word

Mnemonic : LH
Operand : RD, RA
Format : LH RD, RA
Operasi : RD ← DataMem[RA]
Tujuan : Mengambil data sebesar dua *byte* (setengah *word*) dari alamat yang ditunjuk pada *data memory* kemudian disimpan pada *register* tujuan.
Perancangan : Menggunakan MCI (Memory Control In) untuk menangani cara penyimpanannya. Alamat diambil dari *bus* A dan data langsung menuju *bus* D.

Load Word

Mnemonic : LW
Operand : RD, RA
Format : LW RD, RA
Operasi : RD ← DataMem[RA]
Tujuan : Mengambil data sebesar empat *byte* (satu *word*) dari alamat yang ditunjuk pada *data memory* kemudian disimpan pada *register* tujuan.
Perancangan : Menggunakan MCI (Memory Control In) untuk menangani cara penyimpanannya. Alamat diambil dari *bus* A dan data langsung menuju *bus* D.

Set if Less Then

Mnemonic : SLT
Oerand : DA, RA, RB
Format : SLT DA, RA, RB
Operasi : if (RA < RB) then
DA ← X"00000001"
else
DA ← X"00000000"
Tujuan : SLT digunakan untuk menyimpan hasil perbandingan dua buah *register*.
Deskripsi : Kedua *register* yang dibandingkan merupakan bilangan bertanda.
Perancangan : Melakukan pengurangan antara RA dengan RB kemudian hasil XOR antara *flag Overflow* dan *flag Negative* ditaruh pada *register* tujuan.

Set if Less Then Immediate

Mnemonic : SLTI
Operand : DA, RA, Imm
Format : SLTI DA, RA, Imm
Operasi : if (RA < Imm) then
DA ← X"00000001"
else
DA ← X"00000000"
Tujuan : SLT digunakan untuk menyimpan hasil perbandingan antara *register* dengan nilai *immediate*.
Deskripsi : Jangkauan nilai *immediate* antara -32768 sampai dengan 32767.
Perancangan : Melakukan pengurangan antara RA dengan nilai *immediate* kemudian hasil XOR antara *flag Overflow* dan *flag Negative* ditaruh pada *register* tujuan.

Disable Interrupt

Mnemonic : DI
Operand : -
Format : DI
Operasi : E_Int ← '0'
Tujuan : Tidak mengizinkan terjadinya *interrupt*.
Perancangan : Menggunakan *interrupt control* dimana *disable interrupt* terjadi diketahui melalui nilai Opcode.

Enable Interrupt

Mnemonic : EI
Operand : -
Format : EI
Operasi : E_Int ← '1'

Tujuan : Mengizinkan terjadinya *interrupt*.
Perancangan : Menggunakan *interrupt control* dimana *enable interrupt* terjadi diketahui melalui nilai Opcode.

Branch if Equal

Mnemonic : BE
Operand : RA, RB, Imm
Format : BE RA, RB, Imm
Operasi : if (RA = RB) then
PC ← PC + Target
else
PC ← PC + 1
Tujuan : Melakukan percabangan relatif bersyarat jika RA sama dengan RB.
Deskripsi : Jangkauan instruksi yang dapat dicapai adalah 32767 instruksi ke atas atau 32768 instruksi ke bawah.
Perancangan : Menggunakan komponen *branch control* dimana percabangan dilakukan jika pengurangan menghasilkan nilai 1 untuk *flag Zero*.

Branch if Higher

Mnemonic : BH
Operand : RA, RB, Imm
Format : BH RA, RB, Imm
Operasi : if (RA(U) > RB(U)) then
PC ← PC + Target
else
PC ← PC + 1
Tujuan : Melakukan percabangan relatif bersyarat jika RA lebih besar dari pada RB dimana RA dan RB merupakan bilangan tidak bertanda.
Deskripsi : Jangkauan instruksi yang dapat dicapai adalah 32767 instruksi ke atas atau 32768 instruksi ke bawah.
Perancangan : Menggunakan komponen *branch control* dimana percabangan dilakukan jika pengurangan menghasilkan nilai 1 untuk *flag Carry* dan 0 untuk *flag Zero*.

Branch if Higher Equal

Mnemonic : BHE
Operand : RA, RB, Imm
Format : BHE RA, RB, Imm
Operasi : if (RA(U) >= RB(U)) then
PC ← PC + Target
else
PC ← PC + 1
Tujuan : Melakukan percabangan relatif bersyarat jika RA lebih besar atau sama dengan RB dimana RA dan RB merupakan bilangan tidak bertanda.

Deskripsi : Jangkauan instruksi yang dapat dicapai adalah 32767 instruksi ke atas atau 32768 instruksi ke bawah.
Perancangan : Menggunakan komponen *branch control* dimana percabangan dilakukan jika pengurangan menghasilkan nilai 1 untuk *flag Carry*.

Branch if Greater

Mnemonic : BG
Operand : RA, RB, Imm
Format : BG RA, RB, Imm
Operasi : if (RA(S) > RB(S)) then
PC ← PC + Target
else
PC ← PC + 1
Tujuan : Melakukan percabangan relatif bersyarat jika RA lebih besar dari pada RB dimana RA dan RB merupakan bilangan bertanda (sign).
Deskripsi : *Flag* XNV merupakan XOR dari *flag Negative* dengan *flag Overflow*. Jangkauan instruksi yang dapat dicapai adalah 32767 instruksi ke atas atau 32768 instruksi ke bawah.
Perancangan : Menggunakan komponen *branch control* dimana percabangan dilakukan jika pengurangan menghasilkan nilai 0 untuk *flag* XNV dan *flag Zero*.

Branch if Greater Equal

Mnemonic : BGE
Operand : RA, RB, Imm
Format : BGE RA, RB, Imm
Operasi : if (RA(S) >= RB(S)) then
PC ← PC + Target
else
PC ← PC + 1
Tujuan : Melakukan percabangan relatif bersyarat jika RA lebih besar atau sama dengan RB dimana RA dan RB merupakan bilangan bertanda (sign).
Deskripsi : *Flag* XNV merupakan XOR dari *flag Negative* dengan *flag Overflow*. Jangkauan instruksi yang dapat dicapai adalah 32767 instruksi ke atas atau 32768 instruksi ke bawah.
Perancangan : Menggunakan komponen *branch control* dimana percabangan dilakukan jika pengurangan menghasilkan nilai 0 untuk *flag* XNV.

Jump

Mnemonic : JMP
Operand : Target
Format : JMP Target
Operasi : PC ← PC + Target
Tujuan : Melakukan percabangan relatif tidak bersyarat.

Deskripsi : Jangkauan instruksi yang dapat dicapai adalah 33554431 instruksi ke atas atau 33554432 instruksi ke bawah.
Perancangan : Menggunakan komponen *branch control*.

Jump and Link

Mnemonic : JL
Operand : Target
Format : JL Target
Operasi : R31 ← PC + 1
PC ← PC + Target
Tujuan : Digunakan pada operasi pemanggilan prosedur.
Deskripsi : Jangkauan instruksi yang dapat dicapai adalah 33554431 instruksi ke atas atau 33554432 instruksi ke bawah.
Perancangan : Menggunakan komponen *branch control* seperti pada instruksi *jump* digabungkan dengan instruksi *load address* dimana R31 digunakan sebagai *register* tujuan.

Jump Register

Mnemonic : JR
Operand : RB
Format : JR RB
Operasi : PC ← RB
Tujuan : Digunakan pada operasi kembali dari prosedur.
Deskripsi : Jangkauan percabangan dapat mengapai seluruh memori instruksi.
Perancangan : Menggunakan komponen *branch control* dimana nilai PC diambil dari nilai *register* yang melalui JRA (*bus* B).

Jump Register and Link

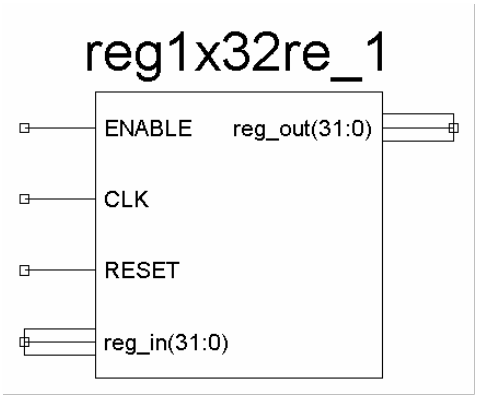
Mnemonic : JRL
Operand : RB
Format : JRL RB
Operasi : R31 ← PC
PC ← RB
Tujuan : Digunakan pada operasi pemanggilan prosedur.
Deskripsi : Jangkauan percabangan dapat mengapai seluruh memori instruksi.
Perancangan : Menggunakan komponen *branch control* seperti pada instruksi *jump register* digabungkan dengan instruksi *load address* dimana R31 digunakan sebagai *register* tujuan.

Program VHDL

Listing Komponen Prosesor RISC

Register 32-bit Asynchronous Reset Synchronous Enable Negatif Edge Clock

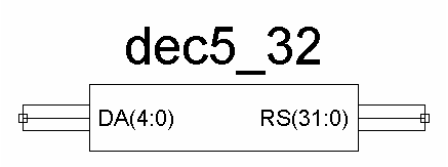
```
Nama File   :     reg1x32RE_1.vhd
Hirarki     :     # reg1x32RE_1
Program     :

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity reg1x32RE_1 is
    Port ( reg_in : in std_logic_vector(31 downto 0);
           reg_out : out std_logic_vector(31 downto 0);
           ENABLE : in std_logic;
           CLK : in std_logic;
           RESET : in std_logic);
end reg1x32RE_1;

architecture Behavioral of reg1x32RE_1 is

begin

      process(CLK, RESET, ENABLE) begin
            if(RESET = '1') then
                  reg_out <= X"00000000";
            elsif(CLK'EVENT and CLK = '0') then
                  if(ENABLE = '1') then
                        Reg_Out <= Reg_In;
                  end if;
            end if;
      end process;

end Behavioral;

Simbol      :
```



Decoder 5 to 32

```
Nama File   :     dec5_32.vhd
Hirarki     :     # dec_5_32
Program     :

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity dec5_32 is
    Port ( DA : in std_logic_vector(4 downto 0);
           RS : out std_logic_vector(31 downto 0));
end dec5_32;

architecture Behavioral of dec5_32 is

begin

      DEC_D : process(DA) begin
            if   (DA = "00000") then RS <= X"00000001";
            elsif(DA = "00001") then RS <= X"00000002";
            elsif(DA = "00010") then RS <= X"00000004";
            elsif(DA = "00011") then RS <= X"00000008";
            elsif(DA = "00100") then RS <= X"00000010";
            elsif(DA = "00101") then RS <= X"00000020";
            elsif(DA = "00110") then RS <= X"00000040";
            elsif(DA = "00111") then RS <= X"00000080";
            elsif(DA = "01000") then RS <= X"00000100";
            elsif(DA = "01001") then RS <= X"00000200";
            elsif(DA = "01010") then RS <= X"00000400";
            elsif(DA = "01011") then RS <= X"00000800";
            elsif(DA = "01100") then RS <= X"00001000";
            elsif(DA = "01101") then RS <= X"00002000";
            elsif(DA = "01110") then RS <= X"00004000";
            elsif(DA = "01111") then RS <= X"00008000";
            elsif(DA = "10000") then RS <= X"00010000";
            elsif(DA = "10001") then RS <= X"00020000";
            elsif(DA = "10010") then RS <= X"00040000";
            elsif(DA = "10011") then RS <= X"00080000";
            elsif(DA = "10100") then RS <= X"00100000";
            elsif(DA = "10101") then RS <= X"00200000";
            elsif(DA = "10110") then RS <= X"00400000";
            elsif(DA = "10111") then RS <= X"00800000";
            elsif(DA = "11000") then RS <= X"01000000";
            elsif(DA = "11001") then RS <= X"02000000";
            elsif(DA = "11010") then RS <= X"04000000";
            elsif(DA = "11011") then RS <= X"08000000";
            elsif(DA = "11100") then RS <= X"10000000";
            elsif(DA = "11101") then RS <= X"20000000";
            elsif(DA = "11110") then RS <= X"40000000";
            else                     RS <= X"80000000";
            end if;
      end process DEC_D;

end Behavioral;
```

Simbol :

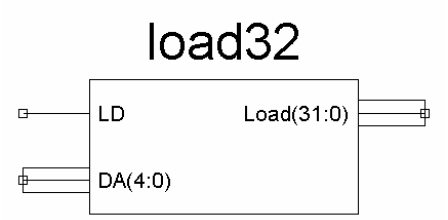


Load 32 bit

```
Nama File   :     load32.vhd
Hierarchy   :     # load_32
                  |
                  +-> # dec_5_32
Fungsi      :     Memilih register yang akan ditulis
Program     :
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity load32 is
    Port ( LD : in std_logic;
           DA : in std_logic_vector(4 downto 0);
           Load : out std_logic_vector(31 downto 0));
end load32;

architecture Behavioral of load32 is

      component dec5_32
            Port( DA : in std_logic_vector(4 downto 0);
                  RS : out std_logic_vector(31 downto 0));
      end component;

      signal RS : std_logic_vector(31 downto 0);

begin

      decoder : dec5_32 port map (DA=>DA, RS=>RS);

      process(LD, RS) begin
            for i in 31 downto 0 loop
                  Load(i) <= LD AND RS(i);
            end loop;
      end process;

end Behavioral;
```

Simbol :

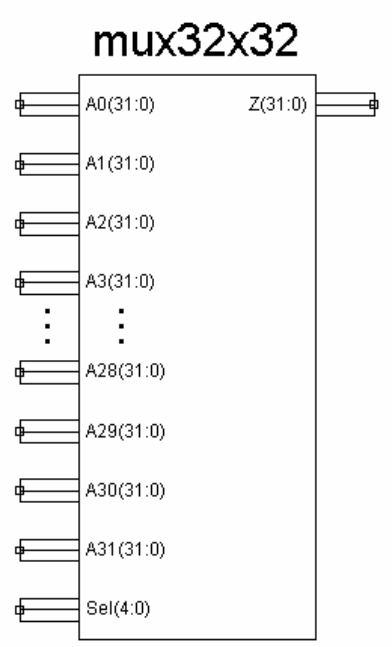


Multiplexer 32 input 32 bit

```
Nama File   :     mux32x32.vhd
Hierarchy   :     # mux32x32
Program     :

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity mux32x32 is
   Port ( A0, A1, A2, A3, A4, A5, A6, A7, A8, A9, A10, A11, A12, A13,
          A14, A15, A16, A17, A18, A19, A20, A21, A22, A23, A24, A25,
          A26, A27, A28, A29, A30, A31 :
          in std_logic_vector(31 downto 0);
          Sel : in std_logic_vector(4 downto 0);
          Z : out std_logic_vector(31 downto 0));
end mux32x32;

architecture Behavioral of mux32x32 is
begin

   process(Sel, A0, A1, A2, A3, A4, A5, A6, A7, A8, A9, A10, A11, A12,
           A13, A14, A15, A16, A17, A18, A19, A20, A21, A22, A23, A24,
           A25, A26, A27, A28, A29, A30, A31) begin

      if   (sel = "00000") then Z <= A0;
      elsif(sel = "00001") then Z <= A1;
      elsif(sel = "00010") then Z <= A2;
      elsif(sel = "00011") then Z <= A3;
      elsif(sel = "00100") then Z <= A4;
      elsif(sel = "00101") then Z <= A5;
      elsif(sel = "00110") then Z <= A6;
      elsif(sel = "00111") then Z <= A7;
      elsif(sel = "01000") then Z <= A8;
      elsif(sel = "01001") then Z <= A9;
      elsif(sel = "01010") then Z <= A10;
      elsif(sel = "01011") then Z <= A11;
      elsif(sel = "01100") then Z <= A12;
      elsif(sel = "01101") then Z <= A13;
      elsif(sel = "01110") then Z <= A14;
      elsif(sel = "01111") then Z <= A15;
      elsif(sel = "10000") then Z <= A16;
      elsif(sel = "10001") then Z <= A17;
      elsif(sel = "10010") then Z <= A18;
      elsif(sel = "10011") then Z <= A19;
      elsif(sel = "10100") then Z <= A20;
      elsif(sel = "10101") then Z <= A21;
      elsif(sel = "10110") then Z <= A22;
      elsif(sel = "10111") then Z <= A23;
      elsif(sel = "11000") then Z <= A24;
      elsif(sel = "11001") then Z <= A25;
      elsif(sel = "11010") then Z <= A26;
      elsif(sel = "11011") then Z <= A27;
      elsif(sel = "11100") then Z <= A28;
      elsif(sel = "11101") then Z <= A29;
```

```
      elsif(sel = "11110") then Z <= A30;
      else                      Z <= A31;
      end if;
   end process;

end Behavioral;
```

Simbol :



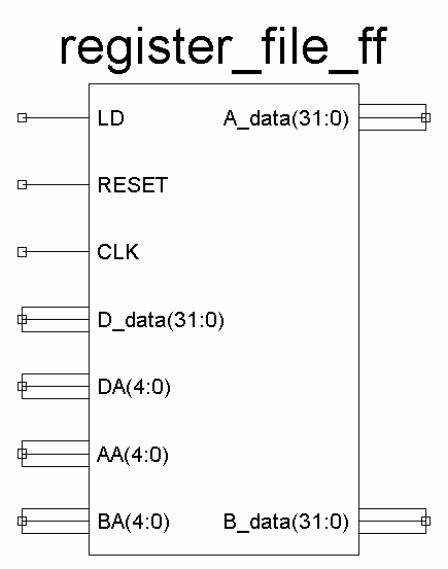
Register File 32x32-bit dengan Flip-flop

```
Nama File   :     register_file_FF.vhd
Hirarki     :     # register_file_FF
                  |
                  +-> # load32
                  |   |
                  |   +-> # dec5_32
                  |
                  +-> # mux32x32
                  |
                  +-> # reg1x32RE_1
Program     :

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity register_file_FF is
   Port ( D_data : in std_logic_vector(31 downto 0);
          DA, AA, BA : in std_logic_vector(4 downto 0);
          LD, RESET, CLK : in std_logic;
          A_data, B_data : out std_logic_vector(31 downto 0));
end register_file_FF;
```

```
architecture Behavioral of register_file_FF is

   component mux32x32
      Port ( A0, A1, A2, A3, A4, A5, A6, A7, A8, A9, A10, A11, A12,
             A13, A14, A15, A16, A17, A18, A19, A20, A21, A22, A23,
             A24, A25, A26, A27, A28, A29, A30, A31 :
             in std_logic_vector(31 downto 0);
             Sel : in std_logic_vector(4 downto 0);
             Z : out std_logic_vector(31 downto 0));
   end component;

   component load32
      Port ( LD : in std_logic;
             DA : in std_logic_vector(4 downto 0);
             Load : out std_logic_vector(31 downto 0));
   end component;

   component reg1x32RE_1
      Port ( Reg_In : in std_logic_vector(31 downto 0);
             Reg_Out : out std_logic_vector(31 downto 0);
             ENABLE, CLK, RESET : in std_logic);
   end component;

   signal R0, R1, R2, R3, R4, R5, R6, R7, R8, R9, R10, R11, R12, R13,
          R14, R15, R16, R17, R18, R19, R20, R21, R22, R23, R24, R25,
          R26, R27, R28, R29, R30, R31 : std_logic_vector(31 downto 0);
   signal Load : std_logic_vector (31 downto 0);

begin

   R0 <= X"00000000";

   mult_1 : mux32x32 port map (A0=>R0, A1=>R1, A2=>R2, A3=>R3, A4=>R4,
            A5=>R5, A6=>R6, A7=>R7, A8=>R8, A9=>R9, A10=>R10, A11=>R11,
            A12=>R12, A13=>R13, A14=>R14, A15=>R15, A16=>R16, A17=>R17,
            A18=>R18, A19=>R19, A20=>R20, A21=>R21, A22=>R22, A23=>R23,
            A24=>R24, A25=>R25, A26=>R26, A27=>R27, A28=>R28, A29=>R29,
            A30=>R30, A31=>R31, Sel=>AA, Z=>A_data);

   mult_2 : mux32x32 port map (A0=>R0, A1=>R1, A2=>R2, A3=>R3, A4=>R4,
            A5=>R5, A6=>R6, A7=>R7, A8=>R8, A9=>R9, A10=>R10, A11=>R11,
            A12=>R12, A13=>R13, A14=>R14, A15=>R15, A16=>R16, A17=>R17,
            A18=>R18, A19=>R19, A20=>R20, A21=>R21, A22=>R22, A23=>R23,
            A24=>R24, A25=>R25, A26=>R26, A27=>R27, A28=>R28, A29=>R29,
            A30=>R30, A31=>R31, Sel=>BA, Z=>B_data);

   load_map : load32 port map (LD=>LD, DA=>DA, Load=>Load);

   reg_32_1 :  reg1x32RE_1 port map (Reg_In=>D_data, Reg_Out=>R1,
               ENABLE=>Load(1), CLK=>CLK, RESET=>RESET);
   reg_32_2 :  reg1x32RE_1 port map (Reg_In=>D_data, Reg_Out=>R2,
               ENABLE=>Load(2), CLK=>CLK, RESET=>RESET);
   reg_32_3 :  reg1x32RE_1 port map (Reg_In=>D_data, Reg_Out=>R3,
               ENABLE=>Load(3), CLK=>CLK, RESET=>RESET);
   reg_32_4 :  reg1x32RE_1 port map (Reg_In=>D_data, Reg_Out=>R4,
               ENABLE=>Load(4), CLK=>CLK, RESET=>RESET);
```

```
   reg_32_5 :  reg1x32RE_1 port map (Reg_In=>D_data, Reg_Out=>R5,
               ENABLE=>Load(5), CLK=>CLK, RESET=>RESET);
   reg_32_6 :  reg1x32RE_1 port map (Reg_In=>D_data, Reg_Out=>R6,
               ENABLE=>Load(6), CLK=>CLK, RESET=>RESET);
   reg_32_7 :  reg1x32RE_1 port map (Reg_In=>D_data, Reg_Out=>R7,
               ENABLE=>Load(7), CLK=>CLK, RESET=>RESET);
   reg_32_8 :  reg1x32RE_1 port map (Reg_In=>D_data, Reg_Out=>R8,
               ENABLE=>Load(8), CLK=>CLK, RESET=>RESET);
   reg_32_9 :  reg1x32RE_1 port map (Reg_In=>D_data, Reg_Out=>R9,
               ENABLE=>Load(9), CLK=>CLK, RESET=>RESET);
   reg_32_10 : reg1x32RE_1 port map (Reg_In=>D_data, Reg_Out=>R10,
               ENABLE=>Load(10), CLK=>CLK, RESET=>RESET);
   reg_32_11 : reg1x32RE_1 port map (Reg_In=>D_data, Reg_Out=>R11,
               ENABLE=>Load(11), CLK=>CLK, RESET=>RESET);
   reg_32_12 : reg1x32RE_1 port map (Reg_In=>D_data, Reg_Out=>R12,
               ENABLE=>Load(12), CLK=>CLK, RESET=>RESET);
   reg_32_13 : reg1x32RE_1 port map (Reg_In=>D_data, Reg_Out=>R13,
               ENABLE=>Load(13), CLK=>CLK, RESET=>RESET);
   reg_32_14 : reg1x32RE_1 port map (Reg_In=>D_data, Reg_Out=>R14,
               ENABLE=>Load(14), CLK=>CLK, RESET=>RESET);
   reg_32_15 : reg1x32RE_1 port map (Reg_In=>D_data, Reg_Out=>R15,
               ENABLE=>Load(15), CLK=>CLK, RESET=>RESET);
   reg_32_16 : reg1x32RE_1 port map (Reg_In=>D_data, Reg_Out=>R16,
               ENABLE=>Load(16), CLK=>CLK, RESET=>RESET);
   reg_32_17 : reg1x32RE_1 port map (Reg_In=>D_data, Reg_Out=>R17,
               ENABLE=>Load(17), CLK=>CLK, RESET=>RESET);
   reg_32_18 : reg1x32RE_1 port map (Reg_In=>D_data, Reg_Out=>R18,
               ENABLE=>Load(18), CLK=>CLK, RESET=>RESET);
   reg_32_19 : reg1x32RE_1 port map (Reg_In=>D_data, Reg_Out=>R19,
               ENABLE=>Load(19), CLK=>CLK, RESET=>RESET);
   reg_32_20 : reg1x32RE_1 port map (Reg_In=>D_data, Reg_Out=>R20,
               ENABLE=>Load(20), CLK=>CLK, RESET=>RESET);
   reg_32_21 : reg1x32RE_1 port map (Reg_In=>D_data, Reg_Out=>R21,
               ENABLE=>Load(21), CLK=>CLK, RESET=>RESET);
   reg_32_22 : reg1x32RE_1 port map (Reg_In=>D_data, Reg_Out=>R22,
               ENABLE=>Load(22), CLK=>CLK, RESET=>RESET);
   reg_32_23 : reg1x32RE_1 port map (Reg_In=>D_data, Reg_Out=>R23,
               ENABLE=>Load(23), CLK=>CLK, RESET=>RESET);
   reg_32_24 : reg1x32RE_1 port map (Reg_In=>D_data, Reg_Out=>R24,
               ENABLE=>Load(24), CLK=>CLK, RESET=>RESET);
   reg_32_25 : reg1x32RE_1 port map (Reg_In=>D_data, Reg_Out=>R25,
               ENABLE=>Load(25), CLK=>CLK, RESET=>RESET);
   reg_32_26 : reg1x32RE_1 port map (Reg_In=>D_data, Reg_Out=>R26,
               ENABLE=>Load(26), CLK=>CLK, RESET=>RESET);
   reg_32_27 : reg1x32RE_1 port map (Reg_In=>D_data, Reg_Out=>R27,
               ENABLE=>Load(27), CLK=>CLK, RESET=>RESET);
   reg_32_28 : reg1x32RE_1 port map (Reg_In=>D_data, Reg_Out=>R28,
               ENABLE=>Load(28), CLK=>CLK, RESET=>RESET);
   reg_32_29 : reg1x32RE_1 port map (Reg_In=>D_data, Reg_Out=>R29,
               ENABLE=>Load(29), CLK=>CLK, RESET=>RESET);
   reg_32_30 : reg1x32RE_1 port map (Reg_In=>D_data, Reg_Out=>R30,
               ENABLE=>Load(30), CLK=>CLK, RESET=>RESET);
   reg_32_31 : reg1x32RE_1 port map (Reg_In=>D_data, Reg_Out=>R31,
               ENABLE=>Load(31), CLK=>CLK, RESET=>RESET);

end Behavioral;
```

Simbol :



Register File 32x32-bit dengan DP RAM

```
Nama File   :     register_file_DPRAM.vhd
Hirarki     :     # register_file_DPRAM
Program     :

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Register_File_DPRAM is
   port ( AA, BA, DA : in STD_LOGIC_vector(4 downto 0);
          D_Data : in std_logic_vector(31 downto 0);
          CLK, LD : in STD_LOGIC;
          A_Data, B_Data : out STD_LOGIC_vector(31 downto 0)
        );
end Register_File_DPRAM;

architecture Behavioral of Register_File_DPRAM is

   component RAM16X1D_1
      generic (INIT: bit_vector := X"16");
      port ( DPO : out STD_ULOGIC;
             SPO : out STD_ULOGIC;
             A0 : in STD_ULOGIC;
             A1 : in STD_ULOGIC;
             A2 : in STD_ULOGIC;
             A3 : in STD_ULOGIC;
             D : in STD_ULOGIC;
             DPRA0 : in STD_ULOGIC;
             DPRA1 : in STD_ULOGIC;
             DPRA2 : in STD_ULOGIC;
             DPRA3 : in STD_ULOGIC;
             WCLK : in STD_ULOGIC;
             WE : in STD_ULOGIC);
```

```
    end component;

    signal LD0, LD1 : std_logic;
    signal A_Data_tmp0, A_Data_tmp1, B_Data_tmp0, B_Data_tmp1 :
           std_logic_vector(31 downto 0);

begin

    LD0 <= LD AND (NOT DA(4));
    LD1 <= LD AND DA(4);

    RAM16X1D_1_16A_0  : RAM16X1D_1 generic map (INIT => X"16") port map
       (DPO => A_Data_tmp0(0),  A0 => DA(0), A1 => DA(1), A2 => DA(2),
       A3 => DA(3), D => D_Data(0),  DPRA0 => AA(0), DPRA1 => AA(1),
       DPRA2 => AA(2), DPRA3 => AA(3), WCLK => CLK, WE => LD0);
    RAM16X1D_1_16A_1  : RAM16X1D_1 generic map (INIT => X"16") port map
       (DPO => A_Data_tmp0(1),  A0 => DA(0), A1 => DA(1), A2 => DA(2),
       A3 => DA(3), D => D_Data(1),  DPRA0 => AA(0), DPRA1 => AA(1),
       DPRA2 => AA(2), DPRA3 => AA(3), WCLK => CLK, WE => LD0);
    RAM16X1D_1_16A_2  : RAM16X1D_1 generic map (INIT => X"16") port map
       (DPO => A_Data_tmp0(2),  A0 => DA(0), A1 => DA(1), A2 => DA(2),
       A3 => DA(3), D => D_Data(2),  DPRA0 => AA(0), DPRA1 => AA(1),
       DPRA2 => AA(2), DPRA3 => AA(3), WCLK => CLK, WE => LD0);
    RAM16X1D_1_16A_3  : RAM16X1D_1 generic map (INIT => X"16") port map
       (DPO => A_Data_tmp0(3),  A0 => DA(0), A1 => DA(1), A2 => DA(2),
       A3 => DA(3), D => D_Data(3),  DPRA0 => AA(0), DPRA1 => AA(1),
       DPRA2 => AA(2), DPRA3 => AA(3), WCLK => CLK, WE => LD0);
    RAM16X1D_1_16A_4  : RAM16X1D_1 generic map (INIT => X"16") port map
       (DPO => A_Data_tmp0(4),  A0 => DA(0), A1 => DA(1), A2 => DA(2),
       A3 => DA(3), D => D_Data(4),  DPRA0 => AA(0), DPRA1 => AA(1),
       DPRA2 => AA(2), DPRA3 => AA(3), WCLK => CLK, WE => LD0);
    RAM16X1D_1_16A_5  : RAM16X1D_1 generic map (INIT => X"16") port map
       (DPO => A_Data_tmp0(5),  A0 => DA(0), A1 => DA(1), A2 => DA(2),
       A3 => DA(3), D => D_Data(5),  DPRA0 => AA(0), DPRA1 => AA(1),
       DPRA2 => AA(2), DPRA3 => AA(3), WCLK => CLK, WE => LD0);
    RAM16X1D_1_16A_6  : RAM16X1D_1 generic map (INIT => X"16") port map
       (DPO => A_Data_tmp0(6),  A0 => DA(0), A1 => DA(1), A2 => DA(2),
       A3 => DA(3), D => D_Data(6),  DPRA0 => AA(0), DPRA1 => AA(1),
       DPRA2 => AA(2), DPRA3 => AA(3), WCLK => CLK, WE => LD0);
    RAM16X1D_1_16A_7  : RAM16X1D_1 generic map (INIT => X"16") port map
       (DPO => A_Data_tmp0(7),  A0 => DA(0), A1 => DA(1), A2 => DA(2),
       A3 => DA(3), D => D_Data(7),  DPRA0 => AA(0), DPRA1 => AA(1),
       DPRA2 => AA(2), DPRA3 => AA(3), WCLK => CLK, WE => LD0);
    RAM16X1D_1_16A_8  : RAM16X1D_1 generic map (INIT => X"16") port map
       (DPO => A_Data_tmp0(8),  A0 => DA(0), A1 => DA(1), A2 => DA(2),
       A3 => DA(3), D => D_Data(8),  DPRA0 => AA(0), DPRA1 => AA(1),
       DPRA2 => AA(2), DPRA3 => AA(3), WCLK => CLK, WE => LD0);
    RAM16X1D_1_16A_9  : RAM16X1D_1 generic map (INIT => X"16") port map
       (DPO => A_Data_tmp0(9),  A0 => DA(0), A1 => DA(1), A2 => DA(2),
       A3 => DA(3), D => D_Data(9),  DPRA0 => AA(0), DPRA1 => AA(1),
       DPRA2 => AA(2), DPRA3 => AA(3), WCLK => CLK, WE => LD0);
    RAM16X1D_1_16A_10 : RAM16X1D_1 generic map (INIT => X"16") port map
       (DPO => A_Data_tmp0(10), A0 => DA(0), A1 => DA(1), A2 => DA(2),
       A3 => DA(3), D => D_Data(10), DPRA0 => AA(0), DPRA1 => AA(1),
       DPRA2 => AA(2), DPRA3 => AA(3), WCLK => CLK, WE => LD0);
    RAM16X1D_1_16A_11 : RAM16X1D_1 generic map (INIT => X"16") port map
       (DPO => A_Data_tmp0(11), A0 => DA(0), A1 => DA(1), A2 => DA(2),
```

```
   A3 => DA(3), D => D_Data(11), DPRA0 => AA(0), DPRA1 => AA(1),
   DPRA2 => AA(2), DPRA3 => AA(3), WCLK => CLK, WE => LD0);
RAM16X1D_1_16A_12 : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => A_Data_tmp0(12), A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(12), DPRA0 => AA(0), DPRA1 => AA(1),
   DPRA2 => AA(2), DPRA3 => AA(3), WCLK => CLK, WE => LD0);
RAM16X1D_1_16A_13 : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => A_Data_tmp0(13), A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(13), DPRA0 => AA(0), DPRA1 => AA(1),
   DPRA2 => AA(2), DPRA3 => AA(3), WCLK => CLK, WE => LD0);
RAM16X1D_1_16A_14 : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => A_Data_tmp0(14), A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(14), DPRA0 => AA(0), DPRA1 => AA(1),
   DPRA2 => AA(2), DPRA3 => AA(3), WCLK => CLK, WE => LD0);
RAM16X1D_1_16A_15 : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => A_Data_tmp0(15), A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(15), DPRA0 => AA(0), DPRA1 => AA(1),
   DPRA2 => AA(2), DPRA3 => AA(3), WCLK => CLK, WE => LD0);
RAM16X1D_1_16A_16 : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => A_Data_tmp0(16), A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(16), DPRA0 => AA(0), DPRA1 => AA(1),
   DPRA2 => AA(2), DPRA3 => AA(3), WCLK => CLK, WE => LD0);
RAM16X1D_1_16A_17 : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => A_Data_tmp0(17), A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(17), DPRA0 => AA(0), DPRA1 => AA(1),
   DPRA2 => AA(2), DPRA3 => AA(3), WCLK => CLK, WE => LD0);
RAM16X1D_1_16A_18 : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => A_Data_tmp0(18), A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(18), DPRA0 => AA(0), DPRA1 => AA(1),
   DPRA2 => AA(2), DPRA3 => AA(3), WCLK => CLK, WE => LD0);
RAM16X1D_1_16A_19 : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => A_Data_tmp0(19), A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(19), DPRA0 => AA(0), DPRA1 => AA(1),
   DPRA2 => AA(2), DPRA3 => AA(3), WCLK => CLK, WE => LD0);
RAM16X1D_1_16A_20 : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => A_Data_tmp0(20), A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(20), DPRA0 => AA(0), DPRA1 => AA(1),
   DPRA2 => AA(2), DPRA3 => AA(3), WCLK => CLK, WE => LD0);
RAM16X1D_1_16A_21 : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => A_Data_tmp0(21), A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(21), DPRA0 => AA(0), DPRA1 => AA(1),
   DPRA2 => AA(2), DPRA3 => AA(3), WCLK => CLK, WE => LD0);
RAM16X1D_1_16A_22 : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => A_Data_tmp0(22), A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(22), DPRA0 => AA(0), DPRA1 => AA(1),
   DPRA2 => AA(2), DPRA3 => AA(3), WCLK => CLK, WE => LD0);
RAM16X1D_1_16A_23 : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => A_Data_tmp0(23), A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(23), DPRA0 => AA(0), DPRA1 => AA(1),
   DPRA2 => AA(2), DPRA3 => AA(3), WCLK => CLK, WE => LD0);
RAM16X1D_1_16A_24 : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => A_Data_tmp0(24), A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(24), DPRA0 => AA(0), DPRA1 => AA(1),
   DPRA2 => AA(2), DPRA3 => AA(3), WCLK => CLK, WE => LD0);
RAM16X1D_1_16A_25 : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => A_Data_tmp0(25), A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(25), DPRA0 => AA(0), DPRA1 => AA(1),
```

```
   DPRA2 => AA(2), DPRA3 => AA(3), WCLK => CLK, WE => LD0);
RAM16X1D_1_16A_26 : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => A_Data_tmp0(26), A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(26), DPRA0 => AA(0), DPRA1 => AA(1),
   DPRA2 => AA(2), DPRA3 => AA(3), WCLK => CLK, WE => LD0);
RAM16X1D_1_16A_27 : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => A_Data_tmp0(27), A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(27), DPRA0 => AA(0), DPRA1 => AA(1),
   DPRA2 => AA(2), DPRA3 => AA(3), WCLK => CLK, WE => LD0);
RAM16X1D_1_16A_28 : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => A_Data_tmp0(28), A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(28), DPRA0 => AA(0), DPRA1 => AA(1),
   DPRA2 => AA(2), DPRA3 => AA(3), WCLK => CLK, WE => LD0);
RAM16X1D_1_16A_29 : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => A_Data_tmp0(29), A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(29), DPRA0 => AA(0), DPRA1 => AA(1),
   DPRA2 => AA(2), DPRA3 => AA(3), WCLK => CLK, WE => LD0);
RAM16X1D_1_16A_30 : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => A_Data_tmp0(30), A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(30), DPRA0 => AA(0), DPRA1 => AA(1),
   DPRA2 => AA(2), DPRA3 => AA(3), WCLK => CLK, WE => LD0);
RAM16X1D_1_16A_31 : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => A_Data_tmp0(31), A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(31), DPRA0 => AA(0), DPRA1 => AA(1),
   DPRA2 => AA(2), DPRA3 => AA(3), WCLK => CLK, WE => LD0);

RAM16X1D_1_32A_0  : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => A_Data_tmp1(0),  A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(0),  DPRA0 => AA(0), DPRA1 => AA(1),
   DPRA2 => AA(2), DPRA3 => AA(3), WCLK => CLK, WE => LD1);
RAM16X1D_1_32A_1  : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => A_Data_tmp1(1),  A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(1),  DPRA0 => AA(0), DPRA1 => AA(1),
   DPRA2 => AA(2), DPRA3 => AA(3), WCLK => CLK, WE => LD1);
RAM16X1D_1_32A_2  : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => A_Data_tmp1(2),  A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(2),  DPRA0 => AA(0), DPRA1 => AA(1),
   DPRA2 => AA(2), DPRA3 => AA(3), WCLK => CLK, WE => LD1);
RAM16X1D_1_32A_3  : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => A_Data_tmp1(3),  A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(3),  DPRA0 => AA(0), DPRA1 => AA(1),
   DPRA2 => AA(2), DPRA3 => AA(3), WCLK => CLK, WE => LD1);
RAM16X1D_1_32A_4  : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => A_Data_tmp1(4),  A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(4),  DPRA0 => AA(0), DPRA1 => AA(1),
   DPRA2 => AA(2), DPRA3 => AA(3), WCLK => CLK, WE => LD1);
RAM16X1D_1_32A_5  : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => A_Data_tmp1(5),  A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(5),  DPRA0 => AA(0), DPRA1 => AA(1),
   DPRA2 => AA(2), DPRA3 => AA(3), WCLK => CLK, WE => LD1);
RAM16X1D_1_32A_6  : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => A_Data_tmp1(6),  A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(6),  DPRA0 => AA(0), DPRA1 => AA(1),
   DPRA2 => AA(2), DPRA3 => AA(3), WCLK => CLK, WE => LD1);
RAM16X1D_1_32A_7  : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => A_Data_tmp1(7),  A0 => DA(0), A1 => DA(1), A2 => DA(2),
```

```
   A3 => DA(3), D => D_Data(7),  DPRA0 => AA(0), DPRA1 => AA(1),
   DPRA2 => AA(2), DPRA3 => AA(3), WCLK => CLK, WE => LD1);
RAM16X1D_1_32A_8  : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => A_Data_tmp1(8),  A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(8),  DPRA0 => AA(0), DPRA1 => AA(1),
   DPRA2 => AA(2), DPRA3 => AA(3), WCLK => CLK, WE => LD1);
RAM16X1D_1_32A_9  : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => A_Data_tmp1(9),  A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(9),  DPRA0 => AA(0), DPRA1 => AA(1),
   DPRA2 => AA(2), DPRA3 => AA(3), WCLK => CLK, WE => LD1);
RAM16X1D_1_32A_10 : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => A_Data_tmp1(10), A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(10), DPRA0 => AA(0), DPRA1 => AA(1),
   DPRA2 => AA(2), DPRA3 => AA(3), WCLK => CLK, WE => LD1);
RAM16X1D_1_32A_11 : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => A_Data_tmp1(11), A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(11), DPRA0 => AA(0), DPRA1 => AA(1),
   DPRA2 => AA(2), DPRA3 => AA(3), WCLK => CLK, WE => LD1);
RAM16X1D_1_32A_12 : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => A_Data_tmp1(12), A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(12), DPRA0 => AA(0), DPRA1 => AA(1),
   DPRA2 => AA(2), DPRA3 => AA(3), WCLK => CLK, WE => LD1);
RAM16X1D_1_32A_13 : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => A_Data_tmp1(13), A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(13), DPRA0 => AA(0), DPRA1 => AA(1),
   DPRA2 => AA(2), DPRA3 => AA(3), WCLK => CLK, WE => LD1);
RAM16X1D_1_32A_14 : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => A_Data_tmp1(14), A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(14), DPRA0 => AA(0), DPRA1 => AA(1),
   DPRA2 => AA(2), DPRA3 => AA(3), WCLK => CLK, WE => LD1);
RAM16X1D_1_32A_15 : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => A_Data_tmp1(15), A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(15), DPRA0 => AA(0), DPRA1 => AA(1),
   DPRA2 => AA(2), DPRA3 => AA(3), WCLK => CLK, WE => LD1);
RAM16X1D_1_32A_16 : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => A_Data_tmp1(16), A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(16), DPRA0 => AA(0), DPRA1 => AA(1),
   DPRA2 => AA(2), DPRA3 => AA(3), WCLK => CLK, WE => LD1);
RAM16X1D_1_32A_17 : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => A_Data_tmp1(17), A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(17), DPRA0 => AA(0), DPRA1 => AA(1),
   DPRA2 => AA(2), DPRA3 => AA(3), WCLK => CLK, WE => LD1);
RAM16X1D_1_32A_18 : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => A_Data_tmp1(18), A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(18), DPRA0 => AA(0), DPRA1 => AA(1),
   DPRA2 => AA(2), DPRA3 => AA(3), WCLK => CLK, WE => LD1);
RAM16X1D_1_32A_19 : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => A_Data_tmp1(19), A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(19), DPRA0 => AA(0), DPRA1 => AA(1),
   DPRA2 => AA(2), DPRA3 => AA(3), WCLK => CLK, WE => LD1);
RAM16X1D_1_32A_20 : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => A_Data_tmp1(20), A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(20), DPRA0 => AA(0), DPRA1 => AA(1),
   DPRA2 => AA(2), DPRA3 => AA(3), WCLK => CLK, WE => LD1);
RAM16X1D_1_32A_21 : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => A_Data_tmp1(21), A0 => DA(0), A1 => DA(1), A2 => DA(2),
```

```
   A3 => DA(3), D => D_Data(21), DPRA0 => AA(0), DPRA1 => AA(1),
   DPRA2 => AA(2), DPRA3 => AA(3), WCLK => CLK, WE => LD1);
RAM16X1D_1_32A_22 : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => A_Data_tmp1(22), A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(22), DPRA0 => AA(0), DPRA1 => AA(1),
   DPRA2 => AA(2), DPRA3 => AA(3), WCLK => CLK, WE => LD1);
RAM16X1D_1_32A_23 : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => A_Data_tmp1(23), A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(23), DPRA0 => AA(0), DPRA1 => AA(1),
   DPRA2 => AA(2), DPRA3 => AA(3), WCLK => CLK, WE => LD1);
RAM16X1D_1_32A_24 : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => A_Data_tmp1(24), A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(24), DPRA0 => AA(0), DPRA1 => AA(1),
   DPRA2 => AA(2), DPRA3 => AA(3), WCLK => CLK, WE => LD1);
RAM16X1D_1_32A_25 : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => A_Data_tmp1(25), A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(25), DPRA0 => AA(0), DPRA1 => AA(1),
   DPRA2 => AA(2), DPRA3 => AA(3), WCLK => CLK, WE => LD1);
RAM16X1D_1_32A_26 : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => A_Data_tmp1(26), A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(26), DPRA0 => AA(0), DPRA1 => AA(1),
   DPRA2 => AA(2), DPRA3 => AA(3), WCLK => CLK, WE => LD1);
RAM16X1D_1_32A_27 : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => A_Data_tmp1(27), A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(27), DPRA0 => AA(0), DPRA1 => AA(1),
   DPRA2 => AA(2), DPRA3 => AA(3), WCLK => CLK, WE => LD1);
RAM16X1D_1_32A_28 : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => A_Data_tmp1(28), A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(28), DPRA0 => AA(0), DPRA1 => AA(1),
   DPRA2 => AA(2), DPRA3 => AA(3), WCLK => CLK, WE => LD1);
RAM16X1D_1_32A_29 : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => A_Data_tmp1(29), A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(29), DPRA0 => AA(0), DPRA1 => AA(1),
   DPRA2 => AA(2), DPRA3 => AA(3), WCLK => CLK, WE => LD1);
RAM16X1D_1_32A_30 : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => A_Data_tmp1(30), A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(30), DPRA0 => AA(0), DPRA1 => AA(1),
   DPRA2 => AA(2), DPRA3 => AA(3), WCLK => CLK, WE => LD1);
RAM16X1D_1_32A_31 : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => A_Data_tmp1(31), A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(31), DPRA0 => AA(0), DPRA1 => AA(1),
   DPRA2 => AA(2), DPRA3 => AA(3), WCLK => CLK, WE => LD1);

RAM16X1D_1_16B_0  : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => B_Data_tmp0(0),  A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(0),  DPRA0 => BA(0), DPRA1 => BA(1),
   DPRA2 => BA(2), DPRA3 => BA(3), WCLK => CLK, WE => LD0);
RAM16X1D_1_16B_1  : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => B_Data_tmp0(1),  A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(1),  DPRA0 => BA(0), DPRA1 => BA(1),
   DPRA2 => BA(2), DPRA3 => BA(3), WCLK => CLK, WE => LD0);
RAM16X1D_1_16B_2  : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => B_Data_tmp0(2),  A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(2),  DPRA0 => BA(0), DPRA1 => BA(1),
   DPRA2 => BA(2), DPRA3 => BA(3), WCLK => CLK, WE => LD0);
RAM16X1D_1_16B_3  : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => B_Data_tmp0(3),  A0 => DA(0), A1 => DA(1), A2 => DA(2),
```

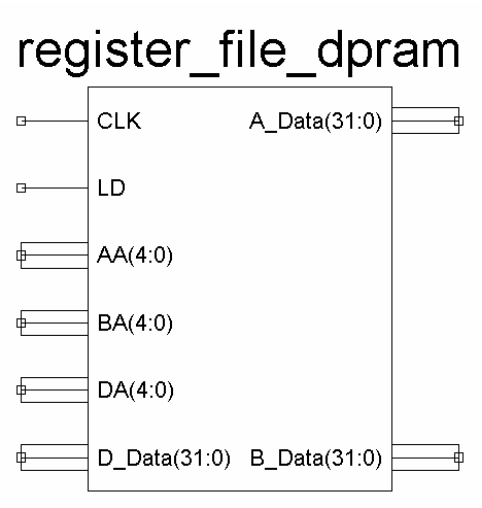
```
   A3 => DA(3), D => D_Data(3),  DPRA0 => BA(0), DPRA1 => BA(1),
   DPRA2 => BA(2), DPRA3 => BA(3), WCLK => CLK, WE => LD0);
RAM16X1D_1_16B_4  : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => B_Data_tmp0(4),  A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(4),  DPRA0 => BA(0), DPRA1 => BA(1),
   DPRA2 => BA(2), DPRA3 => BA(3), WCLK => CLK, WE => LD0);
RAM16X1D_1_16B_5  : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => B_Data_tmp0(5),  A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(5),  DPRA0 => BA(0), DPRA1 => BA(1),
   DPRA2 => BA(2), DPRA3 => BA(3), WCLK => CLK, WE => LD0);
RAM16X1D_1_16B_6  : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => B_Data_tmp0(6),  A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(6),  DPRA0 => BA(0), DPRA1 => BA(1),
   DPRA2 => BA(2), DPRA3 => BA(3), WCLK => CLK, WE => LD0);
RAM16X1D_1_16B_7  : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => B_Data_tmp0(7),  A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(7),  DPRA0 => BA(0), DPRA1 => BA(1),
   DPRA2 => BA(2), DPRA3 => BA(3), WCLK => CLK, WE => LD0);
RAM16X1D_1_16B_8  : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => B_Data_tmp0(8),  A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(8),  DPRA0 => BA(0), DPRA1 => BA(1),
   DPRA2 => BA(2), DPRA3 => BA(3), WCLK => CLK, WE => LD0);
RAM16X1D_1_16B_9  : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => B_Data_tmp0(9),  A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(9),  DPRA0 => BA(0), DPRA1 => BA(1),
   DPRA2 => BA(2), DPRA3 => BA(3), WCLK => CLK, WE => LD0);
RAM16X1D_1_16B_10 : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => B_Data_tmp0(10), A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(10), DPRA0 => BA(0), DPRA1 => BA(1),
   DPRA2 => BA(2), DPRA3 => BA(3), WCLK => CLK, WE => LD0);
RAM16X1D_1_16B_11 : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => B_Data_tmp0(11), A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(11), DPRA0 => BA(0), DPRA1 => BA(1),
   DPRA2 => BA(2), DPRA3 => BA(3), WCLK => CLK, WE => LD0);
RAM16X1D_1_16B_12 : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => B_Data_tmp0(12), A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(12), DPRA0 => BA(0), DPRA1 => BA(1),
   DPRA2 => BA(2), DPRA3 => BA(3), WCLK => CLK, WE => LD0);
RAM16X1D_1_16B_13 : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => B_Data_tmp0(13), A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(13), DPRA0 => BA(0), DPRA1 => BA(1),
   DPRA2 => BA(2), DPRA3 => BA(3), WCLK => CLK, WE => LD0);
RAM16X1D_1_16B_14 : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => B_Data_tmp0(14), A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(14), DPRA0 => BA(0), DPRA1 => BA(1),
   DPRA2 => BA(2), DPRA3 => BA(3), WCLK => CLK, WE => LD0);
RAM16X1D_1_16B_15 : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => B_Data_tmp0(15), A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(15), DPRA0 => BA(0), DPRA1 => BA(1),
   DPRA2 => BA(2), DPRA3 => BA(3), WCLK => CLK, WE => LD0);
RAM16X1D_1_16B_16 : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => B_Data_tmp0(16), A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(16), DPRA0 => BA(0), DPRA1 => BA(1),
   DPRA2 => BA(2), DPRA3 => BA(3), WCLK => CLK, WE => LD0);
RAM16X1D_1_16B_17 : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => B_Data_tmp0(17), A0 => DA(0), A1 => DA(1), A2 => DA(2),
```

```
   A3 => DA(3), D => D_Data(17), DPRA0 => BA(0), DPRA1 => BA(1),
   DPRA2 => BA(2), DPRA3 => BA(3), WCLK => CLK, WE => LD0);
RAM16X1D_1_16B_18 : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => B_Data_tmp0(18), A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(18), DPRA0 => BA(0), DPRA1 => BA(1),
   DPRA2 => BA(2), DPRA3 => BA(3), WCLK => CLK, WE => LD0);
RAM16X1D_1_16B_19 : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => B_Data_tmp0(19), A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(19), DPRA0 => BA(0), DPRA1 => BA(1),
   DPRA2 => BA(2), DPRA3 => BA(3), WCLK => CLK, WE => LD0);
RAM16X1D_1_16B_20 : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => B_Data_tmp0(20), A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(20), DPRA0 => BA(0), DPRA1 => BA(1),
   DPRA2 => BA(2), DPRA3 => BA(3), WCLK => CLK, WE => LD0);
RAM16X1D_1_16B_21 : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => B_Data_tmp0(21), A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(21), DPRA0 => BA(0), DPRA1 => BA(1),
   DPRA2 => BA(2), DPRA3 => BA(3), WCLK => CLK, WE => LD0);
RAM16X1D_1_16B_22 : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => B_Data_tmp0(22), A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(22), DPRA0 => BA(0), DPRA1 => BA(1),
   DPRA2 => BA(2), DPRA3 => BA(3), WCLK => CLK, WE => LD0);
RAM16X1D_1_16B_23 : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => B_Data_tmp0(23), A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(23), DPRA0 => BA(0), DPRA1 => BA(1),
   DPRA2 => BA(2), DPRA3 => BA(3), WCLK => CLK, WE => LD0);
RAM16X1D_1_16B_24 : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => B_Data_tmp0(24), A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(24), DPRA0 => BA(0), DPRA1 => BA(1),
   DPRA2 => BA(2), DPRA3 => BA(3), WCLK => CLK, WE => LD0);
RAM16X1D_1_16B_25 : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => B_Data_tmp0(25), A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(25), DPRA0 => BA(0), DPRA1 => BA(1),
   DPRA2 => BA(2), DPRA3 => BA(3), WCLK => CLK, WE => LD0);
RAM16X1D_1_16B_26 : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => B_Data_tmp0(26), A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(26), DPRA0 => BA(0), DPRA1 => BA(1),
   DPRA2 => BA(2), DPRA3 => BA(3), WCLK => CLK, WE => LD0);
RAM16X1D_1_16B_27 : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => B_Data_tmp0(27), A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(27), DPRA0 => BA(0), DPRA1 => BA(1),
   DPRA2 => BA(2), DPRA3 => BA(3), WCLK => CLK, WE => LD0);
RAM16X1D_1_16B_28 : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => B_Data_tmp0(28), A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(28), DPRA0 => BA(0), DPRA1 => BA(1),
   DPRA2 => BA(2), DPRA3 => BA(3), WCLK => CLK, WE => LD0);
RAM16X1D_1_16B_29 : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => B_Data_tmp0(29), A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(29), DPRA0 => BA(0), DPRA1 => BA(1),
   DPRA2 => BA(2), DPRA3 => BA(3), WCLK => CLK, WE => LD0);
RAM16X1D_1_16B_30 : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => B_Data_tmp0(30), A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(30), DPRA0 => BA(0), DPRA1 => BA(1),
   DPRA2 => BA(2), DPRA3 => BA(3), WCLK => CLK, WE => LD0);
RAM16X1D_1_16B_31 : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => B_Data_tmp0(31), A0 => DA(0), A1 => DA(1), A2 => DA(2),
```

```
   A3 => DA(3), D => D_Data(31), DPRA0 => BA(0), DPRA1 => BA(1),
   DPRA2 => BA(2), DPRA3 => BA(3), WCLK => CLK, WE => LD0);

RAM16X1D_1_32B_0  : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => B_Data_tmp1(0),  A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(0),  DPRA0 => BA(0), DPRA1 => BA(1),
   DPRA2 => BA(2), DPRA3 => BA(3), WCLK => CLK, WE => LD1);
RAM16X1D_1_32B_1  : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => B_Data_tmp1(1),  A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(1),  DPRA0 => BA(0), DPRA1 => BA(1),
   DPRA2 => BA(2), DPRA3 => BA(3), WCLK => CLK, WE => LD1);
RAM16X1D_1_32B_2  : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => B_Data_tmp1(2),  A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(2),  DPRA0 => BA(0), DPRA1 => BA(1),
   DPRA2 => BA(2), DPRA3 => BA(3), WCLK => CLK, WE => LD1);
RAM16X1D_1_32B_3  : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => B_Data_tmp1(3),  A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(3),  DPRA0 => BA(0), DPRA1 => BA(1),
   DPRA2 => BA(2), DPRA3 => BA(3), WCLK => CLK, WE => LD1);
RAM16X1D_1_32B_4  : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => B_Data_tmp1(4),  A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(4),  DPRA0 => BA(0), DPRA1 => BA(1),
   DPRA2 => BA(2), DPRA3 => BA(3), WCLK => CLK, WE => LD1);
RAM16X1D_1_32B_5  : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => B_Data_tmp1(5),  A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(5),  DPRA0 => BA(0), DPRA1 => BA(1),
   DPRA2 => BA(2), DPRA3 => BA(3), WCLK => CLK, WE => LD1);
RAM16X1D_1_32B_6  : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => B_Data_tmp1(6),  A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(6),  DPRA0 => BA(0), DPRA1 => BA(1),
   DPRA2 => BA(2), DPRA3 => BA(3), WCLK => CLK, WE => LD1);
RAM16X1D_1_32B_7  : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => B_Data_tmp1(7),  A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(7),  DPRA0 => BA(0), DPRA1 => BA(1),
   DPRA2 => BA(2), DPRA3 => BA(3), WCLK => CLK, WE => LD1);
RAM16X1D_1_32B_8  : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => B_Data_tmp1(8),  A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(8),  DPRA0 => BA(0), DPRA1 => BA(1),
   DPRA2 => BA(2), DPRA3 => BA(3), WCLK => CLK, WE => LD1);
RAM16X1D_1_32B_9  : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => B_Data_tmp1(9),  A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(9),  DPRA0 => BA(0), DPRA1 => BA(1),
   DPRA2 => BA(2), DPRA3 => BA(3), WCLK => CLK, WE => LD1);
RAM16X1D_1_32B_10 : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => B_Data_tmp1(10), A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(10), DPRA0 => BA(0), DPRA1 => BA(1),
   DPRA2 => BA(2), DPRA3 => BA(3), WCLK => CLK, WE => LD1);
RAM16X1D_1_32B_11 : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => B_Data_tmp1(11), A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(11), DPRA0 => BA(0), DPRA1 => BA(1),
   DPRA2 => BA(2), DPRA3 => BA(3), WCLK => CLK, WE => LD1);
RAM16X1D_1_32B_12 : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => B_Data_tmp1(12), A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(12), DPRA0 => BA(0), DPRA1 => BA(1),
   DPRA2 => BA(2), DPRA3 => BA(3), WCLK => CLK, WE => LD1);
RAM16X1D_1_32B_13 : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => B_Data_tmp1(13), A0 => DA(0), A1 => DA(1), A2 => DA(2),
```

```
   A3 => DA(3), D => D_Data(13), DPRA0 => BA(0), DPRA1 => BA(1),
   DPRA2 => BA(2), DPRA3 => BA(3), WCLK => CLK, WE => LD1);
RAM16X1D_1_32B_14 : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => B_Data_tmp1(14), A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(14), DPRA0 => BA(0), DPRA1 => BA(1),
   DPRA2 => BA(2), DPRA3 => BA(3), WCLK => CLK, WE => LD1);
RAM16X1D_1_32B_15 : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => B_Data_tmp1(15), A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(15), DPRA0 => BA(0), DPRA1 => BA(1),
   DPRA2 => BA(2), DPRA3 => BA(3), WCLK => CLK, WE => LD1);
RAM16X1D_1_32B_16 : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => B_Data_tmp1(16), A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(16), DPRA0 => BA(0), DPRA1 => BA(1),
   DPRA2 => BA(2), DPRA3 => BA(3), WCLK => CLK, WE => LD1);
RAM16X1D_1_32B_17 : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => B_Data_tmp1(17), A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(17), DPRA0 => BA(0), DPRA1 => BA(1),
   DPRA2 => BA(2), DPRA3 => BA(3), WCLK => CLK, WE => LD1);
RAM16X1D_1_32B_18 : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => B_Data_tmp1(18), A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(18), DPRA0 => BA(0), DPRA1 => BA(1),
   DPRA2 => BA(2), DPRA3 => BA(3), WCLK => CLK, WE => LD1);
RAM16X1D_1_32B_19 : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => B_Data_tmp1(19), A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(19), DPRA0 => BA(0), DPRA1 => BA(1),
   DPRA2 => BA(2), DPRA3 => BA(3), WCLK => CLK, WE => LD1);
RAM16X1D_1_32B_20 : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => B_Data_tmp1(20), A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(20), DPRA0 => BA(0), DPRA1 => BA(1),
   DPRA2 => BA(2), DPRA3 => BA(3), WCLK => CLK, WE => LD1);
RAM16X1D_1_32B_21 : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => B_Data_tmp1(21), A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(21), DPRA0 => BA(0), DPRA1 => BA(1),
   DPRA2 => BA(2), DPRA3 => BA(3), WCLK => CLK, WE => LD1);
RAM16X1D_1_32B_22 : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => B_Data_tmp1(22), A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(22), DPRA0 => BA(0), DPRA1 => BA(1),
   DPRA2 => BA(2), DPRA3 => BA(3), WCLK => CLK, WE => LD1);
RAM16X1D_1_32B_23 : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => B_Data_tmp1(23), A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(23), DPRA0 => BA(0), DPRA1 => BA(1),
   DPRA2 => BA(2), DPRA3 => BA(3), WCLK => CLK, WE => LD1);
RAM16X1D_1_32B_24 : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => B_Data_tmp1(24), A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(24), DPRA0 => BA(0), DPRA1 => BA(1),
   DPRA2 => BA(2), DPRA3 => BA(3), WCLK => CLK, WE => LD1);
RAM16X1D_1_32B_25 : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => B_Data_tmp1(25), A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(25), DPRA0 => BA(0), DPRA1 => BA(1),
   DPRA2 => BA(2), DPRA3 => BA(3), WCLK => CLK, WE => LD1);
RAM16X1D_1_32B_26 : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => B_Data_tmp1(26), A0 => DA(0), A1 => DA(1), A2 => DA(2),
   A3 => DA(3), D => D_Data(26), DPRA0 => BA(0), DPRA1 => BA(1),
   DPRA2 => BA(2), DPRA3 => BA(3), WCLK => CLK, WE => LD1);
RAM16X1D_1_32B_27 : RAM16X1D_1 generic map (INIT => X"16") port map
   (DPO => B_Data_tmp1(27), A0 => DA(0), A1 => DA(1), A2 => DA(2),
```

```
      A3 => DA(3), D => D_Data(27), DPRA0 => BA(0), DPRA1 => BA(1),
      DPRA2 => BA(2), DPRA3 => BA(3), WCLK => CLK, WE => LD1);
   RAM16X1D_1_32B_28 : RAM16X1D_1 generic map (INIT => X"16") port map
      (DPO => B_Data_tmp1(28), A0 => DA(0), A1 => DA(1), A2 => DA(2),
      A3 => DA(3), D => D_Data(28), DPRA0 => BA(0), DPRA1 => BA(1),
      DPRA2 => BA(2), DPRA3 => BA(3), WCLK => CLK, WE => LD1);
   RAM16X1D_1_32B_29 : RAM16X1D_1 generic map (INIT => X"16") port map
      (DPO => B_Data_tmp1(29), A0 => DA(0), A1 => DA(1), A2 => DA(2),
      A3 => DA(3), D => D_Data(29), DPRA0 => BA(0), DPRA1 => BA(1),
      DPRA2 => BA(2), DPRA3 => BA(3), WCLK => CLK, WE => LD1);
   RAM16X1D_1_32B_30 : RAM16X1D_1 generic map (INIT => X"16") port map
      (DPO => B_Data_tmp1(30), A0 => DA(0), A1 => DA(1), A2 => DA(2),
      A3 => DA(3), D => D_Data(30), DPRA0 => BA(0), DPRA1 => BA(1),
      DPRA2 => BA(2), DPRA3 => BA(3), WCLK => CLK, WE => LD1);
   RAM16X1D_1_32B_31 : RAM16X1D_1 generic map (INIT => X"16") port map
      (DPO => B_Data_tmp1(31), A0 => DA(0), A1 => DA(1), A2 => DA(2),
      A3 => DA(3), D => D_Data(31), DPRA0 => BA(0), DPRA1 => BA(1),
      DPRA2 => BA(2), DPRA3 => BA(3), WCLK => CLK, WE => LD1);

   process(AA(4), BA(4), A_Data_tmp0, A_Data_tmp1, B_Data_tmp0,
           B_Data_tmp1) begin
      if(AA = "00000") then
         A_Data <= X"00000000";
      else
         if(AA(4) = '0') then
            A_Data <= A_Data_tmp0;
         else
            A_Data <= A_Data_tmp1;
         end if;
      end if;

      if(BA = "00000") then
         B_Data <= X"00000000";
      else
         if(BA(4) = '0') then
            B_Data <= B_Data_tmp0;
         else
            B_Data <= B_Data_tmp1;
         end if;
      end if;
   end process;

end Behavioral;

Simbol      :
```

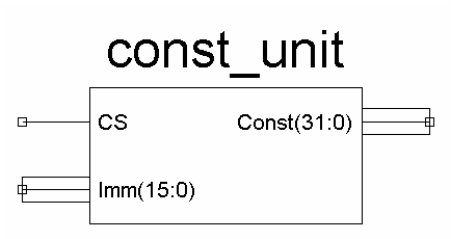


Constant Unit

```
Nama File   :     Constant_Unit.vhd
Hirarki     :     # Constant_Unit
Fungsi      :     Konversi 16 bit ke 32 bit

                    CS    Operation
                  ------------------------------
                    0     X"0000" || IR(15:0)
                    1     IR(15) || IR(15:0)
Program     :
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Const_Unit is
    Port ( Imm : in std_logic_vector(15 downto 0);
           CS : in std_logic;
           Const : out std_logic_vector(31 downto 0));
end Const_Unit;

architecture Behavioral of Const_Unit is

begin

      Const(15 downto 0) <= Imm(15 downto 0);

      process (CS, Imm) begin
            if (CS = '0') then
                  Const(31 downto 16) <= X"0000";
            else
                  for i in 31 downto 16 loop
                        Const(i) <= Imm(15);
                  end loop;
            end if;
      end process;
```

```
end Behavioral;

Simbol      :
```

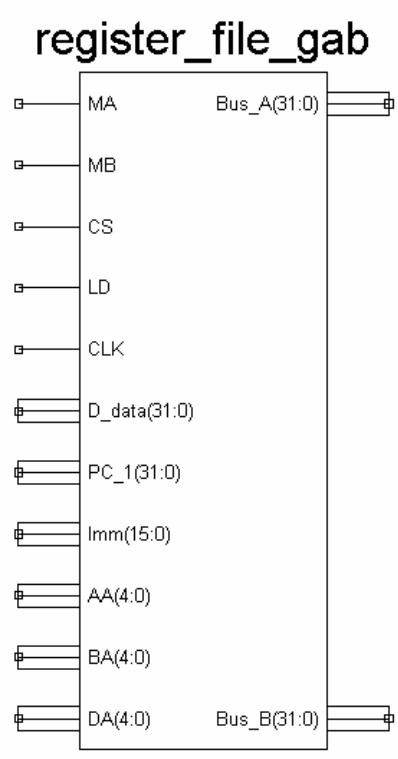


Register File Gabungan

```
Nama File   :     Register_File_Gab.vhd
Hirarki     :     # register_file_gab
                  |
                  +-> # const_unit
                  |
                  +-> # register_file_DPRAM
Fungsi      :     Terdiri dari Register File, Constant Unit, MUX_A, dan
                  MUX_B
Program     :

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Register_File_Gab is
    Port( D_data, PC_1 : in std_logic_vector(31 downto 0);
          Imm : in std_logic_vector(15 downto 0);
          AA, BA, DA : in std_logic_vector(4 downto 0);
          MA, MB, CS, LD, CLK : in std_logic;
          Bus_A, Bus_B : out std_logic_vector(31 downto 0));
end Register_File_Gab;

architecture Behavioral of Register_File_Gab is

      component Register_File_DPRAM
            Port( D_data : in std_logic_vector(31 downto 0);
                  DA, AA, BA : in std_logic_vector(4 downto 0);
                  LD, CLK : in std_logic;
                  A_data, B_data : out std_logic_vector(31 downto 0));
      end component;

      component Const_Unit
            Port( Imm : in std_logic_vector(15 downto 0);
                  CS : in std_logic;
                  Const : out std_logic_vector(31 downto 0));
      end component;

      signal A_data, B_data, Const : std_logic_vector(31 downto 0);

begin
```

```
      register_file_PM : Register_File_DPRAM port map (D_data=>D_data,
                         DA=>DA, AA=>AA, BA=>BA, LD=>LD, CLK=>CLK,
                         A_data=>A_data, B_data=>B_data);
      constant_unit : Const_Unit port map (Imm=>Imm, CS=>CS,
                      Const=>Const);

      MUX_A : process(MA, PC_1, A_data) begin
            if    (MA = '0') then
                  Bus_A <= A_data;
            else
                  BUS_A <= PC_1;
            end if;
      end process;

      MUX_B : process(MB, Const, B_data) begin
            if    (MB = '0') then
                  Bus_B <= B_data;
            else
                  BUS_B <= Const;
            end if;
      end process;

end Behavioral;
```

Simbol :



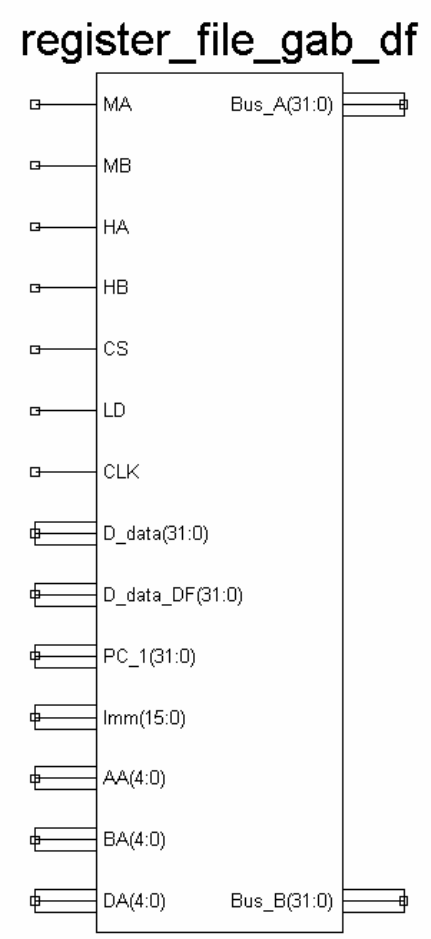
Register File Gabungan untuk Data Forwarding

Nama File : Register_File_Gab_DF.vhd

```
Hirarki     :     # register_file_gab_DF
                  |
                  +-> # const_unit
                  |
                  +-> # register_file_DPRAM
Fungsi      :     Terdiri dari Register File, Constant Unit, MUX_A, dan
                  MUX_B untuk Data Forwarding
Program     :

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Register_File_Gab_DF is
    Port( D_data, D_data_DF, PC_1 : in std_logic_vector(31 downto 0);
          Imm : in std_logic_vector(15 downto 0);
          AA, BA, DA : in std_logic_vector(4 downto 0);
          MA, MB, HA, HB, CS, LD, CLK : in std_logic;
          Bus_A, Bus_B : out std_logic_vector(31 downto 0));
end Register_File_Gab_DF;

architecture Behavioral of Register_File_Gab_DF is

      component Register_File_DPRAM
            Port( D_data : in std_logic_vector(31 downto 0);
                  DA, AA, BA : in std_logic_vector(4 downto 0);
                  LD, CLK : in std_logic;
                  A_data, B_data : out std_logic_vector(31 downto 0));
      end component;

      component Const_Unit
            Port( Imm : in std_logic_vector(15 downto 0);
                  CS : in std_logic;
                  Const : out std_logic_vector(31 downto 0));
      end component;

      signal A_data, B_data, Const : std_logic_vector(31 downto 0);
      signal MAtmp, MBtmp : std_logic_vector(1 downto 0);

begin

      register_file_PM : Register_File_DPRAM port map (D_data=>D_data,
                         DA=>DA, AA=>AA, BA=>BA, LD=>LD, CLK=>CLK,
                         A_data=>A_data, B_data=>B_data);
      constant_unit : Const_Unit port map (Imm=>Imm, CS=>CS,
                      Const=>Const);

      MAtmp <= HA & MA;
      MBtmp <= HB & MB;

      MUX_A : process(MAtmp, PC_1, A_data, D_data_DF) begin
            if    (MAtmp = "00") then
                  Bus_A <= A_data;
            elsif (MAtmp = "01") then
                  Bus_A <= PC_1;
            else
```

```
                BUS_A <= D_data_DF;
            end if;
        end process;

        MUX_B : process(MBtmp, Const, B_data, D_data_DF) begin
            if    (MBtmp = "00") then
                Bus_B <= B_data;
            elsif (MBtmp = "01") then
                Bus_B <= Const;
            else
                BUS_B <= D_data_DF;
            end if;
        end process;

end Behavioral;
```

Simbol :

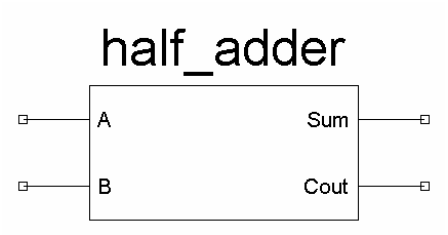


Half Adder

Nama File : half_adder.vhd
Hirarki : # half_adder
Fungsi : Penjumlahan 1 bit
Program :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity half_adder is
    Port ( A, B : in std_logic;
           Sum, Cout : out std_logic);
end half_adder;

architecture Behavioral of half_adder is

begin

            Sum <= A XOR B;
            Cout <= A AND B;

end Behavioral;
```

Simbol :

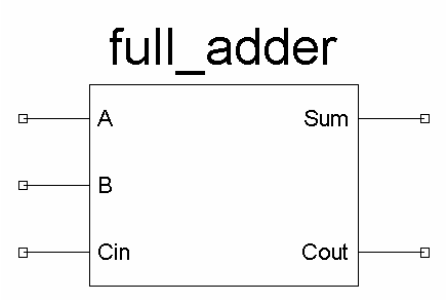


Full Adder

```
Nama File   :     full_adder.vhd
Hirarki     :     # full_adder
                  |
                  +-> # half_adder
Fungsi      :     Penjumlahan 1 bit
Program     :

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity full_adder is
    Port ( A, B, Cin : in std_logic;
           Sum, Cout : out std_logic);

end full_adder;

architecture Behavioral of full_adder is
      component half_adder
            port (  A, B : in std_logic;
                  Sum, Cout : out std_logic);
      end component;

      signal SumH1, CoutH1, CoutH2 : std_logic;
```

```
begin
      HA1 : half_adder port map (A, B, SumH1, CoutH1);
      HA2 : half_adder port map (SumH1, Cin, Sum, CoutH2);
      Cout <= CoutH2 OR CoutH1;

end Behavioral;
```

Simbol :

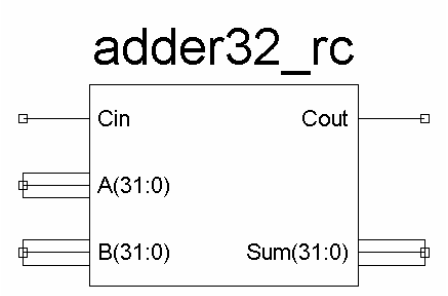


Ripple Carry Adder 32 bit

```
Nama File   :     adder32_rc.vhd
Hirarki     :     # adder32_rc
                  |
                  +-> # full_adder
                      |
                      +-> # half_adder
Fungsi      :     Penjumlahan 32 bit
Program     :

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Adder32_rc is
    Port ( A, B : in std_logic_vector(31 downto 0);
           Cin : in std_logic;
           Sum : out std_logic_vector(31 downto 0);
           Cout : out std_logic);
end Adder32_rc;

architecture Behavioral of Adder32_rc is
      component full_adder
            port( A, B, Cin : in std_logic;
                  Sum, Cout : out std_logic);
      end component;

      signal C : std_logic_vector( 31 downto 1);
begin

      FA1  : full_adder port map (A(0),  B(0),  Cin,   Sum(0),  C(1));
      FA2  : full_adder port map (A(1),  B(1),  C(1),  Sum(1),  C(2));
      FA3  : full_adder port map (A(2),  B(2),  C(2),  Sum(2),  C(3));
      FA4  : full_adder port map (A(3),  B(3),  C(3),  Sum(3),  C(4));
      FA5  : full_adder port map (A(4),  B(4),  C(4),  Sum(4),  C(5));
```

```
      FA6  : full_adder port map (A(5),  B(5),  C(5),  Sum(5),  C(6));
      FA7  : full_adder port map (A(6),  B(6),  C(6),  Sum(6),  C(7));
      FA8  : full_adder port map (A(7),  B(7),  C(7),  Sum(7),  C(8));
      FA9  : full_adder port map (A(8),  B(8),  C(8),  Sum(8),  C(9));
      FA10 : full_adder port map (A(9),  B(9),  C(9),  Sum(9),  C(10));
      FA11 : full_adder port map (A(10), B(10), C(10), Sum(10), C(11));
      FA12 : full_adder port map (A(11), B(11), C(11), Sum(11), C(12));
      FA13 : full_adder port map (A(12), B(12), C(12), Sum(12), C(13));
      FA14 : full_adder port map (A(13), B(13), C(13), Sum(13), C(14));
      FA15 : full_adder port map (A(14), B(14), C(14), Sum(14), C(15));
      FA16 : full_adder port map (A(15), B(15), C(15), Sum(15), C(16));
      FA17 : full_adder port map (A(16), B(16), C(16), Sum(16), C(17));
      FA18 : full_adder port map (A(17), B(17), C(17), Sum(17), C(18));
      FA19 : full_adder port map (A(18), B(18), C(18), Sum(18), C(19));
      FA20 : full_adder port map (A(19), B(19), C(19), Sum(19), C(20));
      FA21 : full_adder port map (A(20), B(20), C(20), Sum(20), C(21));
      FA22 : full_adder port map (A(21), B(21), C(21), Sum(21), C(22));
      FA23 : full_adder port map (A(22), B(22), C(22), Sum(22), C(23));
      FA24 : full_adder port map (A(23), B(23), C(23), Sum(23), C(24));
      FA25 : full_adder port map (A(24), B(24), C(24), Sum(24), C(25));
      FA26 : full_adder port map (A(25), B(25), C(25), Sum(25), C(26));
      FA27 : full_adder port map (A(26), B(26), C(26), Sum(26), C(27));
      FA28 : full_adder port map (A(27), B(27), C(27), Sum(27), C(28));
      FA29 : full_adder port map (A(28), B(28), C(28), Sum(28), C(29));
      FA30 : full_adder port map (A(29), B(29), C(29), Sum(29), C(30));
      FA31 : full_adder port map (A(30), B(30), C(30), Sum(30), C(31));
      FA32 : full_adder port map (A(31), B(31), C(31), Sum(31), Cout);

end Behavioral;
```

Simbol :

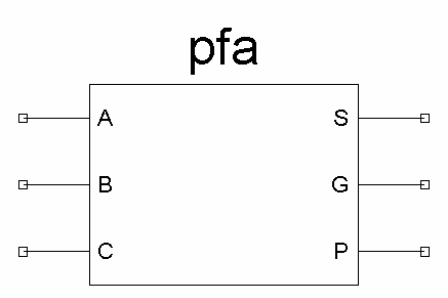


Partial Full Adder

```
Nama File   :     PFA.vhd
Hirarki     :     # PFA
Program     :

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity PFA is
    Port ( A, B, C : in std_logic;
         S, G : out std_logic;
```

```
         P : inout std_logic);
end PFA;

architecture Behavioral of PFA is

begin

      P <= B XOR A;
      S <= P XOR C;
      G <= A AND B;

end Behavioral;
```

Simbol :



Carry Lookahead Adder 32 bit

```
Nama File   :     adder32_cl.vhd
Hirarki     :     # adder32_cl
Fungsi      :     Penjumlahan 32 bit
Program     :

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity adder32_cl is
    Port ( A, B : in std_logic_vector(31 downto 0);
         Cin : in std_logic;
         Sum : out std_logic_vector(31 downto 0);
         Cout : out std_logic);
end adder32_cl;

architecture Behavioral of adder32_cl is
      component PFA
            port (  A, B, C : in std_logic;
                  S, G : out std_logic;
                  P : inout std_logic);
      end component;

      signal C : std_logic_vector(31 downto 1);
      signal G : std_logic_vector(31 downto 0);
      signal P : std_logic_vector(31 downto 0);

begin
```

```
PFA0  : PFA port map (A(0),  B(0),  Cin,   Sum(0),  G(0),  P(0));
PFA1  : PFA port map (A(1),  B(1),  C(1),  Sum(1),  G(1),  P(1));
PFA2  : PFA port map (A(2),  B(2),  C(2),  Sum(2),  G(2),  P(2));
PFA3  : PFA port map (A(3),  B(3),  C(3),  Sum(3),  G(3),  P(3));
PFA4  : PFA port map (A(4),  B(4),  C(4),  Sum(4),  G(4),  P(4));
PFA5  : PFA port map (A(5),  B(5),  C(5),  Sum(5),  G(5),  P(5));
PFA6  : PFA port map (A(6),  B(6),  C(6),  Sum(6),  G(6),  P(6));
PFA7  : PFA port map (A(7),  B(7),  C(7),  Sum(7),  G(7),  P(7));
PFA8  : PFA port map (A(8),  B(8),  C(8),  Sum(8),  G(8),  P(8));
PFA9  : PFA port map (A(9),  B(9),  C(9),  Sum(9),  G(9),  P(9));
PFA10 : PFA port map (A(10), B(10), C(10), Sum(10), G(10), P(10));
PFA11 : PFA port map (A(11), B(11), C(11), Sum(11), G(11), P(11));
PFA12 : PFA port map (A(12), B(12), C(12), Sum(12), G(12), P(12));
PFA13 : PFA port map (A(13), B(13), C(13), Sum(13), G(13), P(13));
PFA14 : PFA port map (A(14), B(14), C(14), Sum(14), G(14), P(14));
PFA15 : PFA port map (A(15), B(15), C(15), Sum(15), G(15), P(15));
PFA16 : PFA port map (A(16), B(16), C(16), Sum(16), G(16), P(16));
PFA17 : PFA port map (A(17), B(17), C(17), Sum(17), G(17), P(17));
PFA18 : PFA port map (A(18), B(18), C(18), Sum(18), G(18), P(18));
PFA19 : PFA port map (A(19), B(19), C(19), Sum(19), G(19), P(19));
PFA20 : PFA port map (A(20), B(20), C(20), Sum(20), G(20), P(20));
PFA21 : PFA port map (A(21), B(21), C(21), Sum(21), G(21), P(21));
PFA22 : PFA port map (A(22), B(22), C(22), Sum(22), G(22), P(22));
PFA23 : PFA port map (A(23), B(23), C(23), Sum(23), G(23), P(23));
PFA24 : PFA port map (A(24), B(24), C(24), Sum(24), G(24), P(24));
PFA25 : PFA port map (A(25), B(25), C(25), Sum(25), G(25), P(25));
PFA26 : PFA port map (A(26), B(26), C(26), Sum(26), G(26), P(26));
PFA27 : PFA port map (A(27), B(27), C(27), Sum(27), G(27), P(27));
PFA28 : PFA port map (A(28), B(28), C(28), Sum(28), G(28), P(28));
PFA29 : PFA port map (A(29), B(29), C(29), Sum(29), G(29), P(29));
PFA30 : PFA port map (A(30), B(30), C(30), Sum(30), G(30), P(30));
PFA31 : PFA port map (A(31), B(31), C(31), Sum(31), G(31), P(31));

C(1) <= G(0) or (P(0) and Cin);
C(2) <= G(1) or (P(1) and G(0)) or (P(1) and P(0) and Cin);
C(3) <= G(2) or (P(2) and G(1)) or (P(2) and P(1) and G(0)) or
   (P(2) and P(1) and P(0) and Cin);
C(4) <= G(3) or (P(3) and G(2)) or (P(3) and P(2) and G(1)) or
   (P(3) and P(2) and P(1) and G(0)) or
   (P(3) and P(2) and P(1) and P(0) and Cin);
C(5) <= G(4) or (P(4) and G(3)) or (P(4) and P(3) and G(2)) or
   (P(4) and P(3) and P(2) and G(1)) or
   (P(4) and P(3) and P(2) and P(1) and G(0)) or
   (P(4) and P(3) and P(2) and P(1) and P(0) and Cin);
C(6) <= G(5) or (P(5) and G(4)) or (P(5) and P(4) and G(3)) or
   (P(5) and P(4) and P(3) and G(2)) or
   (P(5) and P(4) and P(3) and P(2) and G(1)) or
   (P(5) and P(4) and P(3) and P(2) and P(1) and G(0)) or
   (P(5) and P(4) and P(3) and P(2) and P(1) and P(0) and Cin);
C(7) <= G(6) or (P(6) and G(5)) or (P(6) and P(5) and G(4)) or
   (P(6) and P(5) and P(4) and G(3)) or
   (P(6) and P(5) and P(4) and P(3) and G(2)) or
   (P(6) and P(5) and P(4) and P(3) and P(2) and G(1)) or
   (P(6) and P(5) and P(4) and P(3) and P(2) and P(1) and G(0)) or
   (P(6) and P(5) and P(4) and P(3) and P(2) and P(1) and P(0) and
   Cin);
```

```
C(8) <= G(7) or (P(7) and G(6)) or (P(7) and P(6) and G(5)) or
   (P(7) and P(6) and P(5) and G(4)) or
   (P(7) and P(6) and P(5) and P(4) and G(3)) or
   (P(7) and P(6) and P(5) and P(4) and P(3) and G(2)) or
   (P(7) and P(6) and P(5) and P(4) and P(3) and P(2) and G(1)) or
   (P(7) and P(6) and P(5) and P(4) and P(3) and P(2) and P(1) and
   G(0)) or
   (P(7) and P(6) and P(5) and P(4) and P(3) and P(2) and P(1) and P(0)
   and Cin);
C(9) <= G(8) or (P(8) and G(7)) or (P(8) and P(7) and G(6)) or
   (P(8) and P(7) and P(6) and G(5)) or
   (P(8) and P(7) and P(6) and P(5) and G(4)) or
   (P(8) and P(7) and P(6) and P(5) and P(4) and G(3)) or
   (P(8) and P(7) and P(6) and P(5) and P(4) and P(3) and G(2)) or
   (P(8) and P(7) and P(6) and P(5) and P(4) and P(3) and P(2) and
   G(1)) or
   (P(8) and P(7) and P(6) and P(5) and P(4) and P(3) and P(2) and P(1)
   and G(0)) or
   (P(8) and P(7) and P(6) and P(5) and P(4) and P(3) and P(2) and P(1)
   and P(0) and Cin);
C(10) <= G(9) or (P(9) and G(8)) or (P(9) and P(8) and G(7)) or
   (P(9) and P(8) and P(7) and G(6)) or
   (P(9) and P(8) and P(7) and P(6) and G(5)) or
   (P(9) and P(8) and P(7) and P(6) and P(5) and G(4)) or
   (P(9) and P(8) and P(7) and P(6) and P(5) and P(4) and G(3)) or
   (P(9) and P(8) and P(7) and P(6) and P(5) and P(4) and P(3) and
   G(2)) or
   (P(9) and P(8) and P(7) and P(6) and P(5) and P(4) and P(3) and P(2)
   and G(1)) or
   (P(9) and P(8) and P(7) and P(6) and P(5) and P(4) and P(3) and P(2)
   and P(1) and G(0)) or
   (P(9) and P(8) and P(7) and P(6) and P(5) and P(4) and P(3) and P(2)
   and P(1) and P(0) and Cin);
C(11) <= G(10) or (P(10) and G(9)) or (P(10) and P(9) and G(8)) or
   (P(10) and P(9) and P(8) and G(7)) or
   (P(10) and P(9) and P(8) and P(7) and G(6)) or
   (P(10) and P(9) and P(8) and P(7) and P(6) and G(5)) or
   (P(10) and P(9) and P(8) and P(7) and P(6) and P(5) and G(4)) or
   (P(10) and P(9) and P(8) and P(7) and P(6) and P(5) and P(4) and
   G(3)) or
   (P(10) and P(9) and P(8) and P(7) and P(6) and P(5) and P(4) and
   P(3) and G(2)) or
   (P(10) and P(9) and P(8) and P(7) and P(6) and P(5) and P(4) and
   P(3) and P(2) and G(1)) or
   (P(10) and P(9) and P(8) and P(7) and P(6) and P(5) and P(4) and
   P(3) and P(2) and P(1) and G(0)) or
   (P(10) and P(9) and P(8) and P(7) and P(6) and P(5) and P(4) and
   P(3) and P(2) and P(1) and P(0) and Cin);
C(12) <= G(11) or (P(11) and G(10)) or (P(11) and P(10) and G(9)) or
   (P(11) and P(10) and P(9) and G(8)) or
   (P(11) and P(10) and P(9) and P(8) and G(7)) or
   (P(11) and P(10) and P(9) and P(8) and P(7) and G(6)) or
   (P(11) and P(10) and P(9) and P(8) and P(7) and P(6) and G(5)) or
   (P(11) and P(10) and P(9) and P(8) and P(7) and P(6) and P(5) and
   G(4)) or
   (P(11) and P(10) and P(9) and P(8) and P(7) and P(6) and P(5) and
   P(4) and G(3)) or
```

```
   (P(11) and P(10) and P(9) and P(8) and P(7) and P(6) and P(5) and
   P(4) and P(3) and G(2)) or
   (P(11) and P(10) and P(9) and P(8) and P(7) and P(6) and P(5) and
   P(4) and P(3) and P(2) and G(1)) or
   (P(11) and P(10) and P(9) and P(8) and P(7) and P(6) and P(5) and
   P(4) and P(3) and P(2) and P(1) and G(0)) or
   (P(11) and P(10) and P(9) and P(8) and P(7) and P(6) and P(5) and
   P(4) and P(3) and P(2) and P(1) and P(0) and Cin);
C(13) <= G(12) or (P(12) and G(11)) or (P(12) and P(11) and G(10)) or
   (P(12) and P(11) and P(10) and G(9)) or
   (P(12) and P(11) and P(10) and P(9) and G(8)) or
   (P(12) and P(11) and P(10) and P(9) and P(8) and G(7)) or
   (P(12) and P(11) and P(10) and P(9) and P(8) and P(7) and G(6)) or
   (P(12) and P(11) and P(10) and P(9) and P(8) and P(7) and P(6) and
   G(5)) or
   (P(12) and P(11) and P(10) and P(9) and P(8) and P(7) and P(6) and
   P(5) and G(4)) or
   (P(12) and P(11) and P(10) and P(9) and P(8) and P(7) and P(6) and
   P(5) and P(4) and G(3)) or
   (P(12) and P(11) and P(10) and P(9) and P(8) and P(7) and P(6) and
   P(5) and P(4) and P(3) and G(2)) or
   (P(12) and P(11) and P(10) and P(9) and P(8) and P(7) and P(6) and
   P(5) and P(4) and P(3) and P(2) and G(1)) or
   (P(12) and P(11) and P(10) and P(9) and P(8) and P(7) and P(6) and
   P(5) and P(4) and P(3) and P(2) and P(1) and G(0)) or
   (P(12) and P(11) and P(10) and P(9) and P(8) and P(7) and P(6) and
   P(5) and P(4) and P(3) and P(2) and P(1) and P(0) and Cin);
C(14) <= G(13) or (P(13) and G(12)) or (P(13) and P(12) and G(11)) or
   (P(13) and P(12) and P(11) and G(10)) or
   (P(13) and P(12) and P(11) and P(10) and G(9)) or
   (P(13) and P(12) and P(11) and P(10) and P(9) and G(8)) or
   (P(13) and P(12) and P(11) and P(10) and P(9) and P(8) and G(7)) or
   (P(13) and P(12) and P(11) and P(10) and P(9) and P(8) and P(7) and
   G(6)) or
   (P(13) and P(12) and P(11) and P(10) and P(9) and P(8) and P(7) and
   P(6) and G(5)) or
   (P(13) and P(12) and P(11) and P(10) and P(9) and P(8) and P(7) and
   P(6) and P(5) and G(4)) or
   (P(13) and P(12) and P(11) and P(10) and P(9) and P(8) and P(7) and
   P(6) and P(5) and P(4) and G(3)) or
   (P(13) and P(12) and P(11) and P(10) and P(9) and P(8) and P(7) and
   P(6) and P(5) and P(4) and P(3) and G(2)) or
   (P(13) and P(12) and P(11) and P(10) and P(9) and P(8) and P(7) and
   P(6) and P(5) and P(4) and P(3) and P(2) and G(1)) or
   (P(13) and P(12) and P(11) and P(10) and P(9) and P(8) and P(7) and
   P(6) and P(5) and P(4) and P(3) and P(2) and P(1) and G(0)) or
   (P(13) and P(12) and P(11) and P(10) and P(9) and P(8) and P(7) and
   P(6) and P(5) and P(4) and P(3) and P(2) and P(1) and P(0) and Cin);

C(15) <=    G(14) or
   (P(14) and G(13)) or
   (P(14) and P(13) and G(12)) or
   (P(14) and P(13) and P(12) and G(11)) or
   (P(14) and P(13) and P(12) and P(11) and G(10)) or
   (P(14) and P(13) and P(12) and P(11) and P(10) and G(9)) or
   (P(14) and P(13) and P(12) and P(11) and P(10) and P(9) and G(8)) or
```

```
   (P(14) and P(13) and P(12) and P(11) and P(10) and P(9) and P(8) and
   G(7)) or
   (P(14) and P(13) and P(12) and P(11) and P(10) and P(9) and P(8) and
   P(7) and G(6)) or
   (P(14) and P(13) and P(12) and P(11) and P(10) and P(9) and P(8) and
   P(7) and P(6) and G(5)) or
   (P(14) and P(13) and P(12) and P(11) and P(10) and P(9) and P(8) and
   P(7) and P(6) and P(5) and G(4)) or
   (P(14) and P(13) and P(12) and P(11) and P(10) and P(9) and P(8) and
   P(7) and P(6) and P(5) and P(4) and G(3)) or
   (P(14) and P(13) and P(12) and P(11) and P(10) and P(9) and P(8) and
   P(7) and P(6) and P(5) and P(4) and P(3) and G(2)) or
   (P(14) and P(13) and P(12) and P(11) and P(10) and P(9) and P(8) and
   P(7) and P(6) and P(5) and P(4) and P(3) and P(2) and G(1)) or
   (P(14) and P(13) and P(12) and P(11) and P(10) and P(9) and P(8) and
   P(7) and P(6) and P(5) and P(4) and P(3) and P(2) and P(1) and G(0))
   or
   (P(14) and P(13) and P(12) and P(11) and P(10) and P(9) and P(8) and
   P(7) and P(6) and P(5) and P(4) and P(3) and P(2) and P(1) and P(0)
   and Cin);

C(16) <=    G(15) or
   (P(15) and G(14)) or
   (P(15) and P(14) and G(13)) or
   (P(15) and P(14) and P(13) and G(12)) or
   (P(15) and P(14) and P(13) and P(12) and G(11)) or
   (P(15) and P(14) and P(13) and P(12) and P(11) and G(10)) or
   (P(15) and P(14) and P(13) and P(12) and P(11) and P(10) and G(9))
   or
   (P(15) and P(14) and P(13) and P(12) and P(11) and P(10) and P(9)
   and G(8)) or
   (P(15) and P(14) and P(13) and P(12) and P(11) and P(10) and P(9)
   and P(8) and G(7)) or
   (P(15) and P(14) and P(13) and P(12) and P(11) and P(10) and P(9)
   and P(8) and P(7) and G(6)) or
   (P(15) and P(14) and P(13) and P(12) and P(11) and P(10) and P(9)
   and P(8) and P(7) and P(6) and G(5)) or
   (P(15) and P(14) and P(13) and P(12) and P(11) and P(10) and P(9)
   and P(8) and P(7) and P(6) and P(5) and G(4)) or
   (P(15) and P(14) and P(13) and P(12) and P(11) and P(10) and P(9)
   and P(8) and P(7) and P(6) and P(5) and P(4) and G(3)) or
   (P(15) and P(14) and P(13) and P(12) and P(11) and P(10) and P(9)
   and P(8) and P(7) and P(6) and P(5) and P(4) and P(3) and G(2)) or
   (P(15) and P(14) and P(13) and P(12) and P(11) and P(10) and P(9)
   and P(8) and P(7) and P(6) and P(5) and P(4) and P(3) and P(2) and
   G(1)) or
   (P(15) and P(14) and P(13) and P(12) and P(11) and P(10) and P(9)
   and P(8) and P(7) and P(6) and P(5) and P(4) and P(3) and P(2) and
   P(1) and G(0)) or
   (P(15) and P(14) and P(13) and P(12) and P(11) and P(10) and P(9)
   and P(8) and P(7) and P(6) and P(5) and P(4) and P(3) and P(2) and
   P(1) and P(0) and Cin);

C(17) <=    G(16) or
   (P(16) and G(15)) or
   (P(16) and P(15) and G(14)) or
   (P(16) and P(15) and P(14) and G(13)) or
```

```
   (P(16) and P(15) and P(14) and P(13) and G(12)) or
   (P(16) and P(15) and P(14) and P(13) and P(12) and G(11)) or
   (P(16) and P(15) and P(14) and P(13) and P(12) and P(11) and G(10))
   or
   (P(16) and P(15) and P(14) and P(13) and P(12) and P(11) and P(10)
   and G(9)) or
   (P(16) and P(15) and P(14) and P(13) and P(12) and P(11) and P(10)
   and P(9) and G(8)) or
   (P(16) and P(15) and P(14) and P(13) and P(12) and P(11) and P(10)
   and P(9) and P(8) and G(7)) or
   (P(16) and P(15) and P(14) and P(13) and P(12) and P(11) and P(10)
   and P(9) and P(8) and P(7) and G(6)) or
   (P(16) and P(15) and P(14) and P(13) and P(12) and P(11) and P(10)
   and P(9) and P(8) and P(7) and P(6) and G(5)) or
   (P(16) and P(15) and P(14) and P(13) and P(12) and P(11) and P(10)
   and P(9) and P(8) and P(7) and P(6) and P(5) and G(4)) or
   (P(16) and P(15) and P(14) and P(13) and P(12) and P(11) and P(10)
   and P(9) and P(8) and P(7) and P(6) and P(5) and P(4) and G(3)) or
   (P(16) and P(15) and P(14) and P(13) and P(12) and P(11) and P(10)
   and P(9) and P(8) and P(7) and P(6) and P(5) and P(4) and P(3) and
   G(2)) or
   (P(16) and P(15) and P(14) and P(13) and P(12) and P(11) and P(10)
   and P(9) and P(8) and P(7) and P(6) and P(5) and P(4) and P(3) and
   P(2) and G(1)) or
   (P(16) and P(15) and P(14) and P(13) and P(12) and P(11) and P(10)
   and P(9) and P(8) and P(7) and P(6) and P(5) and P(4) and P(3) and
   P(2) and P(1) and G(0)) or
   (P(16) and P(15) and P(14) and P(13) and P(12) and P(11) and P(10)
   and P(9) and P(8) and P(7) and P(6) and P(5) and P(4) and P(3) and
   P(2) and P(1) and P(0) and Cin);

C(18) <=    G(17) or
   (P(17) and G(16)) or
   (P(17) and P(16) and G(15)) or
   (P(17) and P(16) and P(15) and G(14)) or
   (P(17) and P(16) and P(15) and P(14) and G(13)) or
   (P(17) and P(16) and P(15) and P(14) and P(13) and G(12)) or
   (P(17) and P(16) and P(15) and P(14) and P(13) and P(12) and G(11))
   or
   (P(17) and P(16) and P(15) and P(14) and P(13) and P(12) and P(11)
   and G(10)) or
   (P(17) and P(16) and P(15) and P(14) and P(13) and P(12) and P(11)
   and P(10) and G(9)) or
   (P(17) and P(16) and P(15) and P(14) and P(13) and P(12) and P(11)
   and P(10) and P(9) and G(8)) or
   (P(17) and P(16) and P(15) and P(14) and P(13) and P(12) and P(11)
   and P(10) and P(9) and P(8) and G(7)) or
   (P(17) and P(16) and P(15) and P(14) and P(13) and P(12) and P(11)
   and P(10) and P(9) and P(8) and P(7) and G(6)) or
   (P(17) and P(16) and P(15) and P(14) and P(13) and P(12) and P(11)
   and P(10) and P(9) and P(8) and P(7) and P(6) and G(5)) or
   (P(17) and P(16) and P(15) and P(14) and P(13) and P(12) and P(11)
   and P(10) and P(9) and P(8) and P(7) and P(6) and P(5) and G(4)) or
   (P(17) and P(16) and P(15) and P(14) and P(13) and P(12) and P(11)
   and P(10) and P(9) and P(8) and P(7) and P(6) and P(5) and P(4) and
   G(3)) or
```

```
   (P(17) and P(16) and P(15) and P(14) and P(13) and P(12) and P(11)
   and P(10) and P(9) and P(8) and P(7) and P(6) and P(5) and P(4) and
   P(3) and G(2)) or
   (P(17) and P(16) and P(15) and P(14) and P(13) and P(12) and P(11)
   and P(10) and P(9) and P(8) and P(7) and P(6) and P(5) and P(4) and
   P(3) and P(2) and G(1)) or
   (P(17) and P(16) and P(15) and P(14) and P(13) and P(12) and P(11)
   and P(10) and P(9) and P(8) and P(7) and P(6) and P(5) and P(4) and
   P(3) and P(2) and P(1) and G(0)) or
   (P(17) and P(16) and P(15) and P(14) and P(13) and P(12) and P(11)
   and P(10) and P(9) and P(8) and P(7) and P(6) and P(5) and P(4) and
   P(3) and P(2) and P(1) and P(0) and Cin);

C(19) <=    G(18) or
   (P(18) and G(17)) or
   (P(18) and P(17) and G(16)) or
   (P(18) and P(17) and P(16) and G(15)) or
   (P(18) and P(17) and P(16) and P(15) and G(14)) or
   (P(18) and P(17) and P(16) and P(15) and P(14) and G(13)) or
   (P(18) and P(17) and P(16) and P(15) and P(14) and P(13) and G(12))
   or
   (P(18) and P(17) and P(16) and P(15) and P(14) and P(13) and P(12)
   and G(11)) or
   (P(18) and P(17) and P(16) and P(15) and P(14) and P(13) and P(12)
   and P(11) and G(10)) or
   (P(18) and P(17) and P(16) and P(15) and P(14) and P(13) and P(12)
   and P(11) and P(10) and G(9)) or
   (P(18) and P(17) and P(16) and P(15) and P(14) and P(13) and P(12)
   and P(11) and P(10) and P(9) and G(8)) or
   (P(18) and P(17) and P(16) and P(15) and P(14) and P(13) and P(12)
   and P(11) and P(10) and P(9) and P(8) and G(7)) or
   (P(18) and P(17) and P(16) and P(15) and P(14) and P(13) and P(12)
   and P(11) and P(10) and P(9) and P(8) and P(7) and G(6)) or
   (P(18) and P(17) and P(16) and P(15) and P(14) and P(13) and P(12)
   and P(11) and P(10) and P(9) and P(8) and P(7) and P(6) and G(5)) or
   (P(18) and P(17) and P(16) and P(15) and P(14) and P(13) and P(12)
   and P(11) and P(10) and P(9) and P(8) and P(7) and P(6) and P(5) and
   G(4)) or
   (P(18) and P(17) and P(16) and P(15) and P(14) and P(13) and P(12)
   and P(11) and P(10) and P(9) and P(8) and P(7) and P(6) and P(5) and
   P(4) and G(3)) or
   (P(18) and P(17) and P(16) and P(15) and P(14) and P(13) and P(12)
   and P(11) and P(10) and P(9) and P(8) and P(7) and P(6) and P(5) and
   P(4) and P(3) and G(2)) or
   (P(18) and P(17) and P(16) and P(15) and P(14) and P(13) and P(12)
   and P(11) and P(10) and P(9) and P(8) and P(7) and P(6) and P(5) and
   P(4) and P(3) and P(2) and G(1)) or
   (P(18) and P(17) and P(16) and P(15) and P(14) and P(13) and P(12)
   and P(11) and P(10) and P(9) and P(8) and P(7) and P(6) and P(5) and
   P(4) and P(3) and P(2) and P(1) and G(0)) or
   (P(18) and P(17) and P(16) and P(15) and P(14) and P(13) and P(12)
   and P(11) and P(10) and P(9) and P(8) and P(7) and P(6) and P(5) and
   P(4) and P(3) and P(2) and P(1) and P(0) and Cin);

C(20) <=    G(19) or
   (P(19) and G(18)) or
   (P(19) and P(18) and G(17)) or
```

```
   (P(19) and P(18) and P(17) and G(16)) or
   (P(19) and P(18) and P(17) and P(16) and G(15)) or
   (P(19) and P(18) and P(17) and P(16) and P(15) and G(14)) or
   (P(19) and P(18) and P(17) and P(16) and P(15) and P(14) and G(13))
   or
   (P(19) and P(18) and P(17) and P(16) and P(15) and P(14) and P(13)
   and G(12)) or
   (P(19) and P(18) and P(17) and P(16) and P(15) and P(14) and P(13)
   and P(12) and G(11)) or
   (P(19) and P(18) and P(17) and P(16) and P(15) and P(14) and P(13)
   and P(12) and P(11) and G(10)) or
   (P(19) and P(18) and P(17) and P(16) and P(15) and P(14) and P(13)
   and P(12) and P(11) and P(10) and G(9)) or
   (P(19) and P(18) and P(17) and P(16) and P(15) and P(14) and P(13)
   and P(12) and P(11) and P(10) and P(9) and G(8)) or
   (P(19) and P(18) and P(17) and P(16) and P(15) and P(14) and P(13)
   and P(12) and P(11) and P(10) and P(9) and P(8) and G(7)) or
   (P(19) and P(18) and P(17) and P(16) and P(15) and P(14) and P(13)
   and P(12) and P(11) and P(10) and P(9) and P(8) and P(7) and G(6))
   or
   (P(19) and P(18) and P(17) and P(16) and P(15) and P(14) and P(13)
   and P(12) and P(11) and P(10) and P(9) and P(8) and P(7) and P(6)
   and G(5)) or
   (P(19) and P(18) and P(17) and P(16) and P(15) and P(14) and P(13)
   and P(12) and P(11) and P(10) and P(9) and P(8) and P(7) and P(6)
   and P(5) and G(4)) or
   (P(19) and P(18) and P(17) and P(16) and P(15) and P(14) and P(13)
   and P(12) and P(11) and P(10) and P(9) and P(8) and P(7) and P(6)
   and P(5) and P(4) and G(3)) or
   (P(19) and P(18) and P(17) and P(16) and P(15) and P(14) and P(13)
   and P(12) and P(11) and P(10) and P(9) and P(8) and P(7) and P(6)
   and P(5) and P(4) and P(3) and G(2)) or
   (P(19) and P(18) and P(17) and P(16) and P(15) and P(14) and P(13)
   and P(12) and P(11) and P(10) and P(9) and P(8) and P(7) and P(6)
   and P(5) and P(4) and P(3) and P(2) and G(1)) or
   (P(19) and P(18) and P(17) and P(16) and P(15) and P(14) and P(13)
   and P(12) and P(11) and P(10) and P(9) and P(8) and P(7) and P(6)
   and P(5) and P(4) and P(3) and P(2) and P(1) and G(0)) or
   (P(19) and P(18) and P(17) and P(16) and P(15) and P(14) and P(13)
   and P(12) and P(11) and P(10) and P(9) and P(8) and P(7) and P(6)
   and P(5) and P(4) and P(3) and P(2) and P(1) and P(0) and Cin);

C(21) <=    G(20) or
   (P(20) and G(19)) or
   (P(20) and P(19) and G(18)) or
   (P(20) and P(19) and P(18) and G(17)) or
   (P(20) and P(19) and P(18) and P(17) and G(16)) or
   (P(20) and P(19) and P(18) and P(17) and P(16) and G(15)) or
   (P(20) and P(19) and P(18) and P(17) and P(16) and P(15) and G(14))
   or
   (P(20) and P(19) and P(18) and P(17) and P(16) and P(15) and P(14)
   and G(13)) or
   (P(20) and P(19) and P(18) and P(17) and P(16) and P(15) and P(14)
   and P(13) and G(12)) or
   (P(20) and P(19) and P(18) and P(17) and P(16) and P(15) and P(14)
   and P(13) and P(12) and G(11)) or
```

```
   (P(20) and P(19) and P(18) and P(17) and P(16) and P(15) and P(14)
   and P(13) and P(12) and P(11) and G(10)) or
   (P(20) and P(19) and P(18) and P(17) and P(16) and P(15) and P(14)
   and P(13) and P(12) and P(11) and P(10) and G(9)) or
   (P(20) and P(19) and P(18) and P(17) and P(16) and P(15) and P(14)
   and P(13) and P(12) and P(11) and P(10) and P(9) and G(8)) or
   (P(20) and P(19) and P(18) and P(17) and P(16) and P(15) and P(14)
   and P(13) and P(12) and P(11) and P(10) and P(9) and P(8) and G(7))
   or
   (P(20) and P(19) and P(18) and P(17) and P(16) and P(15) and P(14)
   and P(13) and P(12) and P(11) and P(10) and P(9) and P(8) and P(7)
   and G(6)) or
   (P(20) and P(19) and P(18) and P(17) and P(16) and P(15) and P(14)
   and P(13) and P(12) and P(11) and P(10) and P(9) and P(8) and P(7)
   and P(6) and G(5)) or
   (P(20) and P(19) and P(18) and P(17) and P(16) and P(15) and P(14)
   and P(13) and P(12) and P(11) and P(10) and P(9) and P(8) and P(7)
   and P(6) and P(5) and G(4)) or
   (P(20) and P(19) and P(18) and P(17) and P(16) and P(15) and P(14)
   and P(13) and P(12) and P(11) and P(10) and P(9) and P(8) and P(7)
   and P(6) and P(5) and P(4) and G(3)) or
   (P(20) and P(19) and P(18) and P(17) and P(16) and P(15) and P(14)
   and P(13) and P(12) and P(11) and P(10) and P(9) and P(8) and P(7)
   and P(6) and P(5) and P(4) and P(3) and G(2)) or
   (P(20) and P(19) and P(18) and P(17) and P(16) and P(15) and P(14)
   and P(13) and P(12) and P(11) and P(10) and P(9) and P(8) and P(7)
   and P(6) and P(5) and P(4) and P(3) and P(2) and G(1)) or
   (P(20) and P(19) and P(18) and P(17) and P(16) and P(15) and P(14)
   and P(13) and P(12) and P(11) and P(10) and P(9) and P(8) and P(7)
   and P(6) and P(5) and P(4) and P(3) and P(2) and P(1) and G(0)) or
   (P(20) and P(19) and P(18) and P(17) and P(16) and P(15) and P(14)
   and P(13) and P(12) and P(11) and P(10) and P(9) and P(8) and P(7)
   and P(6) and P(5) and P(4) and P(3) and P(2) and P(1) and P(0) and
   Cin);

C(22) <=    G(21) or
   (P(21) and G(20)) or
   (P(21) and P(20) and G(19)) or
   (P(21) and P(20) and P(19) and G(18)) or
   (P(21) and P(20) and P(19) and P(18) and G(17)) or
   (P(21) and P(20) and P(19) and P(18) and P(17) and G(16)) or
   (P(21) and P(20) and P(19) and P(18) and P(17) and P(16) and G(15))
   or
   (P(21) and P(20) and P(19) and P(18) and P(17) and P(16) and P(15)
   and G(14)) or
   (P(21) and P(20) and P(19) and P(18) and P(17) and P(16) and P(15)
   and P(14) and G(13)) or
   (P(21) and P(20) and P(19) and P(18) and P(17) and P(16) and P(15)
   and P(14) and P(13) and G(12)) or
   (P(21) and P(20) and P(19) and P(18) and P(17) and P(16) and P(15)
   and P(14) and P(13) and P(12) and G(11)) or
   (P(21) and P(20) and P(19) and P(18) and P(17) and P(16) and P(15)
   and P(14) and P(13) and P(12) and P(11) and G(10)) or
   (P(21) and P(20) and P(19) and P(18) and P(17) and P(16) and P(15)
   and P(14) and P(13) and P(12) and P(11) and P(10) and G(9)) or
```

```
   (P(21) and P(20) and P(19) and P(18) and P(17) and P(16) and P(15)
   and P(14) and P(13) and P(12) and P(11) and P(10) and P(9) and G(8))
   or
   (P(21) and P(20) and P(19) and P(18) and P(17) and P(16) and P(15)
   and P(14) and P(13) and P(12) and P(11) and P(10) and P(9) and P(8)
   and G(7)) or
   (P(21) and P(20) and P(19) and P(18) and P(17) and P(16) and P(15)
   and P(14) and P(13) and P(12) and P(11) and P(10) and P(9) and P(8)
   and P(7) and G(6)) or
   (P(21) and P(20) and P(19) and P(18) and P(17) and P(16) and P(15)
   and P(14) and P(13) and P(12) and P(11) and P(10) and P(9) and P(8)
   and P(7) and P(6) and G(5)) or
   (P(21) and P(20) and P(19) and P(18) and P(17) and P(16) and P(15)
   and P(14) and P(13) and P(12) and P(11) and P(10) and P(9) and P(8)
   and P(7) and P(6) and P(5) and G(4)) or
   (P(21) and P(20) and P(19) and P(18) and P(17) and P(16) and P(15)
   and P(14) and P(13) and P(12) and P(11) and P(10) and P(9) and P(8)
   and P(7) and P(6) and P(5) and P(4) and G(3)) or
   (P(21) and P(20) and P(19) and P(18) and P(17) and P(16) and P(15)
   and P(14) and P(13) and P(12) and P(11) and P(10) and P(9) and P(8)
   and P(7) and P(6) and P(5) and P(4) and P(3) and G(2)) or
   (P(21) and P(20) and P(19) and P(18) and P(17) and P(16) and P(15)
   and P(14) and P(13) and P(12) and P(11) and P(10) and P(9) and P(8)
   and P(7) and P(6) and P(5) and P(4) and P(3) and P(2) and G(1)) or
   (P(21) and P(20) and P(19) and P(18) and P(17) and P(16) and P(15)
   and P(14) and P(13) and P(12) and P(11) and P(10) and P(9) and P(8)
   and P(7) and P(6) and P(5) and P(4) and P(3) and P(2) and P(1) and
   G(0)) or
   (P(21) and P(20) and P(19) and P(18) and P(17) and P(16) and P(15)
   and P(14) and P(13) and P(12) and P(11) and P(10) and P(9) and P(8)
   and P(7) and P(6) and P(5) and P(4) and P(3) and P(2) and P(1) and
   P(0) and Cin);

C(23) <=    G(22) or
   (P(22) and G(21)) or
   (P(22) and P(21) and G(20)) or
   (P(22) and P(21) and P(20) and G(19)) or
   (P(22) and P(21) and P(20) and P(19) and G(18)) or
   (P(22) and P(21) and P(20) and P(19) and P(18) and G(17)) or
   (P(22) and P(21) and P(20) and P(19) and P(18) and P(17) and G(16))
   or
   (P(22) and P(21) and P(20) and P(19) and P(18) and P(17) and P(16)
   and G(15)) or
   (P(22) and P(21) and P(20) and P(19) and P(18) and P(17) and P(16)
   and P(15) and G(14)) or
   (P(22) and P(21) and P(20) and P(19) and P(18) and P(17) and P(16)
   and P(15) and P(14) and G(13)) or
   (P(22) and P(21) and P(20) and P(19) and P(18) and P(17) and P(16)
   and P(15) and P(14) and P(13) and G(12)) or
   (P(22) and P(21) and P(20) and P(19) and P(18) and P(17) and P(16)
   and P(15) and P(14) and P(13) and P(12) and G(11)) or
   (P(22) and P(21) and P(20) and P(19) and P(18) and P(17) and P(16)
   and P(15) and P(14) and P(13) and P(12) and P(11) and G(10)) or
   (P(22) and P(21) and P(20) and P(19) and P(18) and P(17) and P(16)
   and P(15) and P(14) and P(13) and P(12) and P(11) and P(10) and
   G(9)) or
```

```
   (P(22) and P(21) and P(20) and P(19) and P(18) and P(17) and P(16)
   and P(15) and P(14) and P(13) and P(12) and P(11) and P(10) and P(9)
   and G(8)) or
   (P(22) and P(21) and P(20) and P(19) and P(18) and P(17) and P(16)
   and P(15) and P(14) and P(13) and P(12) and P(11) and P(10) and P(9)
   and P(8) and G(7)) or
   (P(22) and P(21) and P(20) and P(19) and P(18) and P(17) and P(16)
   and P(15) and P(14) and P(13) and P(12) and P(11) and P(10) and P(9)
   and P(8) and P(7) and G(6)) or
   (P(22) and P(21) and P(20) and P(19) and P(18) and P(17) and P(16)
   and P(15) and P(14) and P(13) and P(12) and P(11) and P(10) and P(9)
   and P(8) and P(7) and P(6) and G(5)) or
   (P(22) and P(21) and P(20) and P(19) and P(18) and P(17) and P(16)
   and P(15) and P(14) and P(13) and P(12) and P(11) and P(10) and P(9)
   and P(8) and P(7) and P(6) and P(5) and G(4)) or
   (P(22) and P(21) and P(20) and P(19) and P(18) and P(17) and P(16)
   and P(15) and P(14) and P(13) and P(12) and P(11) and P(10) and P(9)
   and P(8) and P(7) and P(6) and P(5) and P(4) and G(3)) or
   (P(22) and P(21) and P(20) and P(19) and P(18) and P(17) and P(16)
   and P(15) and P(14) and P(13) and P(12) and P(11) and P(10) and P(9)
   and P(8) and P(7) and P(6) and P(5) and P(4) and P(3) and G(2)) or
   (P(22) and P(21) and P(20) and P(19) and P(18) and P(17) and P(16)
   and P(15) and P(14) and P(13) and P(12) and P(11) and P(10) and P(9)
   and P(8) and P(7) and P(6) and P(5) and P(4) and P(3) and P(2) and
   G(1)) or
   (P(22) and P(21) and P(20) and P(19) and P(18) and P(17) and P(16)
   and P(15) and P(14) and P(13) and P(12) and P(11) and P(10) and P(9)
   and P(8) and P(7) and P(6) and P(5) and P(4) and P(3) and P(2) and
   P(1) and G(0)) or
   (P(22) and P(21) and P(20) and P(19) and P(18) and P(17) and P(16)
   and P(15) and P(14) and P(13) and P(12) and P(11) and P(10) and P(9)
   and P(8) and P(7) and P(6) and P(5) and P(4) and P(3) and P(2) and
   P(1) and P(0) and Cin);

C(24) <=    G(23) or
   (P(23) and G(22)) or
   (P(23) and P(22) and G(21)) or
   (P(23) and P(22) and P(21) and G(20)) or
   (P(23) and P(22) and P(21) and P(20) and G(19)) or
   (P(23) and P(22) and P(21) and P(20) and P(19) and G(18)) or
   (P(23) and P(22) and P(21) and P(20) and P(19) and P(18) and G(17))
   or
   (P(23) and P(22) and P(21) and P(20) and P(19) and P(18) and P(17)
   and G(16)) or
   (P(23) and P(22) and P(21) and P(20) and P(19) and P(18) and P(17)
   and P(16) and G(15)) or
   (P(23) and P(22) and P(21) and P(20) and P(19) and P(18) and P(17)
   and P(16) and P(15) and G(14)) or
   (P(23) and P(22) and P(21) and P(20) and P(19) and P(18) and P(17)
   and P(16) and P(15) and P(14) and G(13)) or
   (P(23) and P(22) and P(21) and P(20) and P(19) and P(18) and P(17)
   and P(16) and P(15) and P(14) and P(13) and G(12)) or
   (P(23) and P(22) and P(21) and P(20) and P(19) and P(18) and P(17)
   and P(16) and P(15) and P(14) and P(13) and P(12) and G(11)) or
   (P(23) and P(22) and P(21) and P(20) and P(19) and P(18) and P(17)
   and P(16) and P(15) and P(14) and P(13) and P(12) and P(11) and
   G(10)) or
```

```
   (P(23) and P(22) and P(21) and P(20) and P(19) and P(18) and P(17)
   and P(16) and P(15) and P(14) and P(13) and P(12) and P(11) and
   P(10) and G(9)) or
   (P(23) and P(22) and P(21) and P(20) and P(19) and P(18) and P(17)
   and P(16) and P(15) and P(14) and P(13) and P(12) and P(11) and
   P(10) and P(9) and G(8)) or
   (P(23) and P(22) and P(21) and P(20) and P(19) and P(18) and P(17)
   and P(16) and P(15) and P(14) and P(13) and P(12) and P(11) and
   P(10) and P(9) and P(8) and G(7)) or
   (P(23) and P(22) and P(21) and P(20) and P(19) and P(18) and P(17)
   and P(16) and P(15) and P(14) and P(13) and P(12) and P(11) and
   P(10) and P(9) and P(8) and P(7) and G(6)) or
   (P(23) and P(22) and P(21) and P(20) and P(19) and P(18) and P(17)
   and P(16) and P(15) and P(14) and P(13) and P(12) and P(11) and
   P(10) and P(9) and P(8) and P(7) and P(6) and G(5)) or
   (P(23) and P(22) and P(21) and P(20) and P(19) and P(18) and P(17)
   and P(16) and P(15) and P(14) and P(13) and P(12) and P(11) and
   P(10) and P(9) and P(8) and P(7) and P(6) and P(5) and G(4)) or
   (P(23) and P(22) and P(21) and P(20) and P(19) and P(18) and P(17)
   and P(16) and P(15) and P(14) and P(13) and P(12) and P(11) and
   P(10) and P(9) and P(8) and P(7) and P(6) and P(5) and P(4) and
   G(3)) or
   (P(23) and P(22) and P(21) and P(20) and P(19) and P(18) and P(17)
   and P(16) and P(15) and P(14) and P(13) and P(12) and P(11) and
   P(10) and P(9) and P(8) and P(7) and P(6) and P(5) and P(4) and P(3)
   and G(2)) or
   (P(23) and P(22) and P(21) and P(20) and P(19) and P(18) and P(17)
   and P(16) and P(15) and P(14) and P(13) and P(12) and P(11) and
   P(10) and P(9) and P(8) and P(7) and P(6) and P(5) and P(4) and P(3)
   and P(2) and G(1)) or
   (P(23) and P(22) and P(21) and P(20) and P(19) and P(18) and P(17)
   and P(16) and P(15) and P(14) and P(13) and P(12) and P(11) and
   P(10) and P(9) and P(8) and P(7) and P(6) and P(5) and P(4) and P(3)
   and P(2) and P(1) and G(0)) or
   (P(23) and P(22) and P(21) and P(20) and P(19) and P(18) and P(17)
   and P(16) and P(15) and P(14) and P(13) and P(12) and P(11) and
   P(10) and P(9) and P(8) and P(7) and P(6) and P(5) and P(4) and P(3)
   and P(2) and P(1) and P(0) and Cin);

C(25) <=    G(24) or
   (P(24) and G(23)) or
   (P(24) and P(23) and G(22)) or
   (P(24) and P(23) and P(22) and G(21)) or
   (P(24) and P(23) and P(22) and P(21) and G(20)) or
   (P(24) and P(23) and P(22) and P(21) and P(20) and G(19)) or
   (P(24) and P(23) and P(22) and P(21) and P(20) and P(19) and G(18))
   or
   (P(24) and P(23) and P(22) and P(21) and P(20) and P(19) and P(18)
   and G(17)) or
   (P(24) and P(23) and P(22) and P(21) and P(20) and P(19) and P(18)
   and P(17) and G(16)) or
   (P(24) and P(23) and P(22) and P(21) and P(20) and P(19) and P(18)
   and P(17) and P(16) and G(15)) or
   (P(24) and P(23) and P(22) and P(21) and P(20) and P(19) and P(18)
   and P(17) and P(16) and P(15) and G(14)) or
   (P(24) and P(23) and P(22) and P(21) and P(20) and P(19) and P(18)
   and P(17) and P(16) and P(15) and P(14) and G(13)) or
```

```
   (P(24) and P(23) and P(22) and P(21) and P(20) and P(19) and P(18)
   and P(17) and P(16) and P(15) and P(14) and P(13) and G(12)) or
   (P(24) and P(23) and P(22) and P(21) and P(20) and P(19) and P(18)
   and P(17) and P(16) and P(15) and P(14) and P(13) and P(12) and
   G(11)) or
   (P(24) and P(23) and P(22) and P(21) and P(20) and P(19) and P(18)
   and P(17) and P(16) and P(15) and P(14) and P(13) and P(12) and
   P(11) and G(10)) or
   (P(24) and P(23) and P(22) and P(21) and P(20) and P(19) and P(18)
   and P(17) and P(16) and P(15) and P(14) and P(13) and P(12) and
   P(11) and P(10) and G(9)) or
   (P(24) and P(23) and P(22) and P(21) and P(20) and P(19) and P(18)
   and P(17) and P(16) and P(15) and P(14) and P(13) and P(12) and
   P(11) and P(10) and P(9) and G(8)) or
   (P(24) and P(23) and P(22) and P(21) and P(20) and P(19) and P(18)
   and P(17) and P(16) and P(15) and P(14) and P(13) and P(12) and
   P(11) and P(10) and P(9) and P(8) and G(7)) or
   (P(24) and P(23) and P(22) and P(21) and P(20) and P(19) and P(18)
   and P(17) and P(16) and P(15) and P(14) and P(13) and P(12) and
   P(11) and P(10) and P(9) and P(8) and P(7) and G(6)) or
   (P(24) and P(23) and P(22) and P(21) and P(20) and P(19) and P(18)
   and P(17) and P(16) and P(15) and P(14) and P(13) and P(12) and
   P(11) and P(10) and P(9) and P(8) and P(7) and P(6) and G(5)) or
   (P(24) and P(23) and P(22) and P(21) and P(20) and P(19) and P(18)
   and P(17) and P(16) and P(15) and P(14) and P(13) and P(12) and
   P(11) and P(10) and P(9) and P(8) and P(7) and P(6) and P(5) and
   G(4)) or
   (P(24) and P(23) and P(22) and P(21) and P(20) and P(19) and P(18)
   and P(17) and P(16) and P(15) and P(14) and P(13) and P(12) and
   P(11) and P(10) and P(9) and P(8) and P(7) and P(6) and P(5) and
   P(4) and G(3)) or
   (P(24) and P(23) and P(22) and P(21) and P(20) and P(19) and P(18)
   and P(17) and P(16) and P(15) and P(14) and P(13) and P(12) and
   P(11) and P(10) and P(9) and P(8) and P(7) and P(6) and P(5) and
   P(4) and P(3) and G(2)) or
   (P(24) and P(23) and P(22) and P(21) and P(20) and P(19) and P(18)
   and P(17) and P(16) and P(15) and P(14) and P(13) and P(12) and
   P(11) and P(10) and P(9) and P(8) and P(7) and P(6) and P(5) and
   P(4) and P(3) and P(2) and G(1)) or
   (P(24) and P(23) and P(22) and P(21) and P(20) and P(19) and P(18)
   and P(17) and P(16) and P(15) and P(14) and P(13) and P(12) and
   P(11) and P(10) and P(9) and P(8) and P(7) and P(6) and P(5) and
   P(4) and P(3) and P(2) and P(1) and G(0)) or
   (P(24) and P(23) and P(22) and P(21) and P(20) and P(19) and P(18)
   and P(17) and P(16) and P(15) and P(14) and P(13) and P(12) and
   P(11) and P(10) and P(9) and P(8) and P(7) and P(6) and P(5) and
   P(4) and P(3) and P(2) and P(1) and P(0) and Cin);

C(26) <=    G(25) or
   (P(25) and G(24)) or
   (P(25) and P(24) and G(23)) or
   (P(25) and P(24) and P(23) and G(22)) or
   (P(25) and P(24) and P(23) and P(22) and G(21)) or
   (P(25) and P(24) and P(23) and P(22) and P(21) and G(20)) or
   (P(25) and P(24) and P(23) and P(22) and P(21) and P(20) and G(19))
   or
```

```
(P(25) and P(24) and P(23) and P(22) and P(21) and P(20) and P(19)
and G(18)) or
(P(25) and P(24) and P(23) and P(22) and P(21) and P(20) and P(19)
and P(18) and G(17)) or
(P(25) and P(24) and P(23) and P(22) and P(21) and P(20) and P(19)
and P(18) and P(17) and G(16)) or
(P(25) and P(24) and P(23) and P(22) and P(21) and P(20) and P(19)
and P(18) and P(17) and P(16) and G(15)) or
(P(25) and P(24) and P(23) and P(22) and P(21) and P(20) and P(19)
and P(18) and P(17) and P(16) and P(15) and G(14)) or
(P(25) and P(24) and P(23) and P(22) and P(21) and P(20) and P(19)
and P(18) and P(17) and P(16) and P(15) and P(14) and G(13)) or
(P(25) and P(24) and P(23) and P(22) and P(21) and P(20) and P(19)
and P(18) and P(17) and P(16) and P(15) and P(14) and P(13) and
G(12)) or
(P(25) and P(24) and P(23) and P(22) and P(21) and P(20) and P(19)
and P(18) and P(17) and P(16) and P(15) and P(14) and P(13) and
P(12) and G(11)) or
(P(25) and P(24) and P(23) and P(22) and P(21) and P(20) and P(19)
and P(18) and P(17) and P(16) and P(15) and P(14) and P(13) and
P(12) and P(11) and G(10)) or
(P(25) and P(24) and P(23) and P(22) and P(21) and P(20) and P(19)
and P(18) and P(17) and P(16) and P(15) and P(14) and P(13) and
P(12) and P(11) and P(10) and G(9)) or
(P(25) and P(24) and P(23) and P(22) and P(21) and P(20) and P(19)
and P(18) and P(17) and P(16) and P(15) and P(14) and P(13) and
P(12) and P(11) and P(10) and P(9) and G(8)) or
(P(25) and P(24) and P(23) and P(22) and P(21) and P(20) and P(19)
and P(18) and P(17) and P(16) and P(15) and P(14) and P(13) and
P(12) and P(11) and P(10) and P(9) and P(8) and G(7)) or
(P(25) and P(24) and P(23) and P(22) and P(21) and P(20) and P(19)
and P(18) and P(17) and P(16) and P(15) and P(14) and P(13) and
P(12) and P(11) and P(10) and P(9) and P(8) and P(7) and G(6)) or
(P(25) and P(24) and P(23) and P(22) and P(21) and P(20) and P(19)
and P(18) and P(17) and P(16) and P(15) and P(14) and P(13) and
P(12) and P(11) and P(10) and P(9) and P(8) and P(7) and P(6) and
G(5)) or
(P(25) and P(24) and P(23) and P(22) and P(21) and P(20) and P(19)
and P(18) and P(17) and P(16) and P(15) and P(14) and P(13) and
P(12) and P(11) and P(10) and P(9) and P(8) and P(7) and P(6) and
P(5) and G(4)) or
(P(25) and P(24) and P(23) and P(22) and P(21) and P(20) and P(19)
and P(18) and P(17) and P(16) and P(15) and P(14) and P(13) and
P(12) and P(11) and P(10) and P(9) and P(8) and P(7) and P(6) and
P(5) and P(4) and G(3)) or
(P(25) and P(24) and P(23) and P(22) and P(21) and P(20) and P(19)
and P(18) and P(17) and P(16) and P(15) and P(14) and P(13) and
P(12) and P(11) and P(10) and P(9) and P(8) and P(7) and P(6) and
P(5) and P(4) and P(3) and G(2)) or
(P(25) and P(24) and P(23) and P(22) and P(21) and P(20) and P(19)
and P(18) and P(17) and P(16) and P(15) and P(14) and P(13) and
P(12) and P(11) and P(10) and P(9) and P(8) and P(7) and P(6) and
P(5) and P(4) and P(3) and P(2) and G(1)) or
(P(25) and P(24) and P(23) and P(22) and P(21) and P(20) and P(19)
and P(18) and P(17) and P(16) and P(15) and P(14) and P(13) and
P(12) and P(11) and P(10) and P(9) and P(8) and P(7) and P(6) and
P(5) and P(4) and P(3) and P(2) and P(1) and G(0)) or
```

```
   (P(25) and P(24) and P(23) and P(22) and P(21) and P(20) and P(19)
   and P(18) and P(17) and P(16) and P(15) and P(14) and P(13) and
   P(12) and P(11) and P(10) and P(9) and P(8) and P(7) and P(6) and
   P(5) and P(4) and P(3) and P(2) and P(1) and P(0) and Cin);

C(27) <=    G(26) or
   (P(26) and G(25)) or
   (P(26) and P(25) and G(24)) or
   (P(26) and P(25) and P(24) and G(23)) or
   (P(26) and P(25) and P(24) and P(23) and G(22)) or
   (P(26) and P(25) and P(24) and P(23) and P(22) and G(21)) or
   (P(26) and P(25) and P(24) and P(23) and P(22) and P(21) and G(20))
   or
   (P(26) and P(25) and P(24) and P(23) and P(22) and P(21) and P(20)
   and G(19)) or
   (P(26) and P(25) and P(24) and P(23) and P(22) and P(21) and P(20)
   and P(19) and G(18)) or
   (P(26) and P(25) and P(24) and P(23) and P(22) and P(21) and P(20)
   and P(19) and P(18) and G(17)) or
   (P(26) and P(25) and P(24) and P(23) and P(22) and P(21) and P(20)
   and P(19) and P(18) and P(17) and G(16)) or
   (P(26) and P(25) and P(24) and P(23) and P(22) and P(21) and P(20)
   and P(19) and P(18) and P(17) and P(16) and G(15)) or
   (P(26) and P(25) and P(24) and P(23) and P(22) and P(21) and P(20)
   and P(19) and P(18) and P(17) and P(16) and P(15) and G(14)) or
   (P(26) and P(25) and P(24) and P(23) and P(22) and P(21) and P(20)
   and P(19) and P(18) and P(17) and P(16) and P(15) and P(14) and
   G(13)) or
   (P(26) and P(25) and P(24) and P(23) and P(22) and P(21) and P(20)
   and P(19) and P(18) and P(17) and P(16) and P(15) and P(14) and
   P(13) and G(12)) or
   (P(26) and P(25) and P(24) and P(23) and P(22) and P(21) and P(20)
   and P(19) and P(18) and P(17) and P(16) and P(15) and P(14) and
   P(13) and P(12) and G(11)) or
   (P(26) and P(25) and P(24) and P(23) and P(22) and P(21) and P(20)
   and P(19) and P(18) and P(17) and P(16) and P(15) and P(14) and
   P(13) and P(12) and P(11) and G(10)) or
   (P(26) and P(25) and P(24) and P(23) and P(22) and P(21) and P(20)
   and P(19) and P(18) and P(17) and P(16) and G(15) and P(14) and
   P(13) and P(12) and P(11) and P(10) and G(9)) or
   (P(26) and P(25) and P(24) and P(23) and P(22) and P(21) and P(20)
   and P(19) and P(18) and P(17) and P(16) and P(15) and P(14) and
   P(13) and P(12) and P(11) and P(10) and P(9) and G(8)) or
   (P(26) and P(25) and P(24) and P(23) and P(22) and P(21) and P(20)
   and P(19) and P(18) and P(17) and P(16) and P(15) and P(14) and
   P(13) and P(12) and P(11) and P(10) and P(9) and P(8) and G(7)) or
   (P(26) and P(25) and P(24) and P(23) and P(22) and P(21) and P(20)
   and P(19) and P(18) and P(17) and P(16) and P(15) and P(14) and
   P(13) and P(12) and P(11) and P(10) and P(9) and P(8) and P(7) and
   G(6)) or
   (P(26) and P(25) and P(24) and P(23) and P(22) and P(21) and P(20)
   and P(19) and P(18) and P(17) and P(16) and P(15) and P(14) and
   P(13) and P(12) and P(11) and P(10) and P(9) and P(8) and P(7) and
   P(6) and G(5)) or
   (P(26) and P(25) and P(24) and P(23) and P(22) and P(21) and P(20)
   and P(19) and P(18) and P(17) and P(16) and P(15) and P(14) and
```

```
   P(13) and P(12) and P(11) and P(10) and P(9) and P(8) and P(7) and
   P(6) and P(5) and G(4)) or
   (P(26) and P(25) and P(24) and P(23) and P(22) and P(21) and P(20)
   and P(19) and P(18) and P(17) and P(16) and P(15) and P(14) and
   P(13) and P(12) and P(11) and P(10) and P(9) and P(8) and P(7) and
   P(6) and P(5) and P(4) and G(3)) or
   (P(26) and P(25) and P(24) and P(23) and P(22) and P(21) and P(20)
   and P(19) and P(18) and P(17) and P(16) and P(15) and P(14) and
   P(13) and P(12) and P(11) and P(10) and P(9) and P(8) and P(7) and
   P(6) and P(5) and P(4) and P(3) and G(2)) or
   (P(26) and P(25) and P(24) and P(23) and P(22) and P(21) and P(20)
   and P(19) and P(18) and P(17) and P(16) and P(15) and P(14) and
   P(13) and P(12) and P(11) and P(10) and P(9) and P(8) and P(7) and
   P(6) and P(5) and P(4) and P(3) and P(2) and G(1)) or
   (P(26) and P(25) and P(24) and P(23) and P(22) and P(21) and P(20)
   and P(19) and P(18) and P(17) and P(16) and P(15) and P(14) and
   P(13) and P(12) and P(11) and P(10) and P(9) and P(8) and P(7) and
   P(6) and P(5) and P(4) and P(3) and P(2) and P(1) and G(0)) or
   (P(26) and P(25) and P(24) and P(23) and P(22) and P(21) and P(20)
   and P(19) and P(18) and P(17) and P(16) and P(15) and P(14) and
   P(13) and P(12) and P(11) and P(10) and P(9) and P(8) and P(7) and
   P(6) and P(5) and P(4) and P(3) and P(2) and P(1) and P(0) and Cin);

C(28) <=    G(27) or
   (P(27) and G(26)) or
   (P(27) and P(26) and G(25)) or
   (P(27) and P(26) and P(25) and G(24)) or
   (P(27) and P(26) and P(25) and P(24) and G(23)) or
   (P(27) and P(26) and P(25) and P(24) and P(23) and G(22)) or
   (P(27) and P(26) and P(25) and P(24) and P(23) and P(22) and G(21))
   or
   (P(27) and P(26) and P(25) and P(24) and P(23) and P(22) and P(21)
   and G(20)) or
   (P(27) and P(26) and P(25) and P(24) and P(23) and P(22) and P(21)
   and P(20) and G(19)) or
   (P(27) and P(26) and P(25) and P(24) and P(23) and P(22) and P(21)
   and P(20) and P(19) and G(18)) or
   (P(27) and P(26) and P(25) and P(24) and P(23) and P(22) and P(21)
   and P(20) and P(19) and P(18) and G(17)) or
   (P(27) and P(26) and P(25) and P(24) and P(23) and P(22) and P(21)
   and P(20) and P(19) and P(18) and P(17) and G(16)) or
   (P(27) and P(26) and P(25) and P(24) and P(23) and P(22) and P(21)
   and P(20) and P(19) and P(18) and P(17) and P(16) and G(15)) or
   (P(27) and P(26) and P(25) and P(24) and P(23) and P(22) and P(21)
   and P(20) and P(19) and P(18) and P(17) and P(16) and P(15) and
   G(14)) or
   (P(27) and P(26) and P(25) and P(24) and P(23) and P(22) and P(21)
   and P(20) and P(19) and P(18) and P(17) and P(16) and P(15) and
   P(14) and G(13)) or
   (P(27) and P(26) and P(25) and P(24) and P(23) and P(22) and P(21)
   and P(20) and P(19) and P(18) and P(17) and P(16) and P(15) and
   P(14) and P(13) and G(12)) or
   (P(27) and P(26) and P(25) and P(24) and P(23) and P(22) and P(21)
   and P(20) and P(19) and P(18) and P(17) and P(16) and P(15) and
   P(14) and P(13) and P(12) and G(11)) or
```

```
   (P(27) and P(26) and P(25) and P(24) and P(23) and P(22) and P(21)
   and P(20) and P(19) and P(18) and P(17) and P(16) and P(15) and
   P(14) and P(13) and P(12) and P(11) and G(10)) or
   (P(27) and P(26) and P(25) and P(24) and P(23) and P(22) and P(21)
   and P(20) and P(19) and P(18) and P(17) and P(16) and P(15) and
   P(14) and P(13) and P(12) and P(11) and P(10) and G(9)) or
   (P(27) and P(26) and P(25) and P(24) and P(23) and P(22) and P(21)
   and P(20) and P(19) and P(18) and P(17) and P(16) and P(15) and
   P(14) and P(13) and P(12) and P(11) and P(10) and P(9) and G(8)) or
   (P(27) and P(26) and P(25) and P(24) and P(23) and P(22) and P(21)
   and P(20) and P(19) and P(18) and P(17) and P(16) and P(15) and
   P(14) and P(13) and P(12) and P(11) and P(10) and P(9) and P(8) and
   G(7)) or
   (P(27) and P(26) and P(25) and P(24) and P(23) and P(22) and P(21)
   and P(20) and P(19) and P(18) and P(17) and P(16) and P(15) and
   P(14) and P(13) and P(12) and P(11) and P(10) and P(9) and P(8) and
   P(7) and G(6)) or
   (P(27) and P(26) and P(25) and P(24) and P(23) and P(22) and P(21)
   and P(20) and P(19) and P(18) and P(17) and P(16) and P(15) and
   P(14) and P(13) and P(12) and P(11) and P(10) and P(9) and P(8) and
   P(7) and P(6) and G(5)) or
   (P(27) and P(26) and P(25) and P(24) and P(23) and P(22) and P(21)
   and P(20) and P(19) and P(18) and P(17) and P(16) and P(15) and
   P(14) and P(13) and P(12) and P(11) and P(10) and P(9) and P(8) and
   P(7) and P(6) and P(5) and G(4)) or
   (P(27) and P(26) and P(25) and P(24) and P(23) and P(22) and P(21)
   and P(20) and P(19) and P(18) and P(17) and P(16) and P(15) and
   P(14) and P(13) and P(12) and P(11) and P(10) and P(9) and P(8) and
   P(7) and P(6) and P(5) and P(4) and G(3)) or
   (P(27) and P(26) and P(25) and P(24) and P(23) and P(22) and P(21)
   and P(20) and P(19) and P(18) and P(17) and P(16) and P(15) and
   P(14) and P(13) and P(12) and P(11) and P(10) and P(9) and P(8) and
   P(7) and P(6) and P(5) and P(4) and P(3) and G(2)) or
   (P(27) and P(26) and P(25) and P(24) and P(23) and P(22) and P(21)
   and P(20) and P(19) and P(18) and P(17) and P(16) and P(15) and
   P(14) and P(13) and P(12) and P(11) and P(10) and P(9) and P(8) and
   P(7) and P(6) and P(5) and P(4) and P(3) and P(2) and G(1)) or
   (P(27) and P(26) and P(25) and P(24) and P(23) and P(22) and P(21)
   and P(20) and P(19) and P(18) and P(17) and P(16) and P(15) and
   P(14) and P(13) and P(12) and P(11) and P(10) and P(9) and P(8) and
   P(7) and P(6) and P(5) and P(4) and P(3) and P(2) and P(1) and G(0))
   or
   (P(27) and P(26) and P(25) and P(24) and P(23) and P(22) and P(21)
   and P(20) and P(19) and P(18) and P(17) and P(16) and P(15) and
   P(14) and P(13) and P(12) and P(11) and P(10) and P(9) and P(8) and
   P(7) and P(6) and P(5) and P(4) and P(3) and P(2) and P(1) and P(0)
   and Cin);

C(29) <=    G(28) or
   (P(28) and G(27)) or
   (P(28) and P(27) and G(26)) or
   (P(28) and P(27) and P(26) and G(25)) or
   (P(28) and P(27) and P(26) and P(25) and G(24)) or
   (P(28) and P(27) and P(26) and P(25) and P(24) and G(23)) or
   (P(28) and P(27) and P(26) and P(25) and P(24) and P(23) and G(22))
   or
```

(P(28) and P(27) and P(26) and P(25) and P(24) and P(23) and P(22) and G(21)) or
(P(28) and P(27) and P(26) and P(25) and P(24) and P(23) and P(22) and P(21) and G(20)) or
(P(28) and P(27) and P(26) and P(25) and P(24) and P(23) and P(22) and P(21) and P(20) and G(19)) or
(P(28) and P(27) and P(26) and P(25) and P(24) and P(23) and P(22) and P(21) and P(20) and P(19) and G(18)) or
(P(28) and P(27) and P(26) and P(25) and P(24) and P(23) and P(22) and P(21) and P(20) and P(19) and P(18) and G(17)) or
(P(28) and P(27) and P(26) and P(25) and P(24) and P(23) and P(22) and P(21) and P(20) and P(19) and P(18) and P(17) and G(16)) or
(P(28) and P(27) and P(26) and P(25) and P(24) and P(23) and P(22) and P(21) and P(20) and P(19) and P(18) and P(17) and P(16) and G(15)) or
(P(28) and P(27) and P(26) and P(25) and P(24) and P(23) and P(22) and P(21) and P(20) and P(19) and P(18) and P(17) and P(16) and P(15) and G(14)) or
(P(28) and P(27) and P(26) and P(25) and P(24) and P(23) and P(22) and P(21) and P(20) and P(19) and P(18) and P(17) and P(16) and P(15) and P(14) and G(13)) or
(P(28) and P(27) and P(26) and P(25) and P(24) and P(23) and P(22) and P(21) and P(20) and P(19) and P(18) and P(17) and P(16) and P(15) and P(14) and P(13) and G(12)) or
(P(28) and P(27) and P(26) and P(25) and P(24) and P(23) and P(22) and P(21) and P(20) and P(19) and P(18) and P(17) and P(16) and P(15) and P(14) and P(13) and P(12) and G(11)) or
(P(28) and P(27) and P(26) and P(25) and P(24) and P(23) and P(22) and P(21) and P(20) and P(19) and P(18) and P(17) and P(16) and P(15) and P(14) and P(13) and P(12) and P(11) and G(10)) or
(P(28) and P(27) and P(26) and P(25) and P(24) and P(23) and P(22) and P(21) and P(20) and P(19) and P(18) and P(17) and P(16) and P(15) and P(14) and P(13) and P(12) and P(11) and P(10) and G(9)) or
(P(28) and P(27) and P(26) and P(25) and P(24) and P(23) and P(22) and P(21) and P(20) and P(19) and P(18) and P(17) and P(16) and P(15) and P(14) and P(13) and P(12) and P(11) and P(10) and P(9) and G(8)) or
(P(28) and P(27) and P(26) and P(25) and P(24) and P(23) and P(22) and P(21) and P(20) and P(19) and P(18) and P(17) and P(16) and P(15) and P(14) and P(13) and P(12) and P(11) and P(10) and P(9) and P(8) and G(7)) or
(P(28) and P(27) and P(26) and P(25) and P(24) and P(23) and P(22) and P(21) and P(20) and P(19) and P(18) and P(17) and P(16) and P(15) and P(14) and P(13) and P(12) and P(11) and P(10) and P(9) and P(8) and P(7) and G(6)) or
(P(28) and P(27) and P(26) and P(25) and P(24) and P(23) and P(22) and P(21) and P(20) and P(19) and P(18) and P(17) and P(16) and P(15) and P(14) and P(13) and P(12) and P(11) and P(10) and P(9) and P(8) and P(7) and P(6) and G(5)) or
(P(28) and P(27) and P(26) and P(25) and P(24) and P(23) and P(22) and P(21) and P(20) and P(19) and P(18) and P(17) and P(16) and P(15) and P(14) and P(13) and P(12) and P(11) and P(10) and P(9) and P(8) and P(7) and P(6) and P(5) and G(4)) or
(P(28) and P(27) and P(26) and P(25) and P(24) and P(23) and P(22) and P(21) and P(20) and P(19) and P(18) and P(17) and P(16) and P(15) and P(14) and P(13) and P(12) and P(11) and P(10) and P(9) and P(8) and P(7) and P(6) and P(5) and P(4) and G(3)) or

```
    (P(28) and P(27) and P(26) and P(25) and P(24) and P(23) and P(22)
    and P(21) and P(20) and P(19) and P(18) and P(17) and P(16) and
    P(15) and P(14) and P(13) and P(12) and P(11) and P(10) and P(9) and
    P(8) and P(7) and P(6) and P(5) and P(4) and P(3) and G(2)) or
    (P(28) and P(27) and P(26) and P(25) and P(24) and P(23) and P(22)
    and P(21) and P(20) and P(19) and P(18) and P(17) and P(16) and
    P(15) and P(14) and P(13) and P(12) and P(11) and P(10) and P(9) and
    P(8) and P(7) and P(6) and P(5) and P(4) and P(3) and P(2) and G(1))
    or
    (P(28) and P(27) and P(26) and P(25) and P(24) and P(23) and P(22)
    and P(21) and P(20) and P(19) and P(18) and P(17) and P(16) and
    P(15) and P(14) and P(13) and P(12) and P(11) and P(10) and P(9) and
    P(8) and P(7) and P(6) and P(5) and P(4) and P(3) and P(2) and P(1)
    and G(0)) or
    (P(28) and P(27) and P(26) and P(25) and P(24) and P(23) and P(22)
    and P(21) and P(20) and P(19) and P(18) and P(17) and P(16) and
    P(15) and P(14) and P(13) and P(12) and P(11) and P(10) and P(9) and
    P(8) and P(7) and P(6) and P(5) and P(4) and P(3) and P(2) and P(1)
    and P(0) and Cin);

C(30) <=    G(29) or
    (P(29) and G(28)) or
    (P(29) and P(28) and G(27)) or
    (P(29) and P(28) and P(27) and G(26)) or
    (P(29) and P(28) and P(27) and P(26) and G(25)) or
    (P(29) and P(28) and P(27) and P(26) and P(25) and G(24)) or
    (P(29) and P(28) and P(27) and P(26) and P(25) and P(24) and G(23))
    or
    (P(29) and P(28) and P(27) and P(26) and P(25) and P(24) and P(23)
    and G(22)) or
    (P(29) and P(28) and P(27) and P(26) and P(25) and P(24) and P(23)
    and P(22) and G(21)) or
    (P(29) and P(28) and P(27) and P(26) and P(25) and P(24) and P(23)
    and P(22) and P(21) and G(20)) or
    (P(29) and P(28) and P(27) and P(26) and P(25) and P(24) and P(23)
    and P(22) and P(21) and P(20) and G(19)) or
    (P(29) and P(28) and P(27) and P(26) and P(25) and P(24) and P(23)
    and P(22) and P(21) and P(20) and P(19) and G(18)) or
    (P(29) and P(28) and P(27) and P(26) and P(25) and P(24) and P(23)
    and P(22) and P(21) and P(20) and P(19) and P(18) and G(17)) or
    (P(29) and P(28) and P(27) and P(26) and P(25) and P(24) and P(23)
    and P(22) and P(21) and P(20) and P(19) and P(18) and P(17) and
    G(16)) or
    (P(29) and P(28) and P(27) and P(26) and P(25) and P(24) and P(23)
    and P(22) and P(21) and P(20) and P(19) and P(18) and P(17) and
    P(16) and G(15)) or
    (P(29) and P(28) and P(27) and P(26) and P(25) and P(24) and P(23)
    and P(22) and P(21) and P(20) and P(19) and P(18) and P(17) and
    P(16) and P(15) and G(14)) or
    (P(29) and P(28) and P(27) and P(26) and P(25) and P(24) and P(23)
    and P(22) and P(21) and P(20) and P(19) and P(18) and P(17) and
    P(16) and P(15) and P(14) and G(13)) or
    (P(29) and P(28) and P(27) and P(26) and P(25) and P(24) and P(23)
    and P(22) and P(21) and P(20) and P(19) and P(18) and P(17) and
    P(16) and P(15) and P(14) and P(13) and G(12)) or
```

```
   (P(29) and P(28) and P(27) and P(26) and P(25) and P(24) and P(23)
   and P(22) and P(21) and P(20) and P(19) and P(18) and P(17) and
   P(16) and P(15) and P(14) and P(13) and P(12) and G(11)) or
   (P(29) and P(28) and P(27) and P(26) and P(25) and P(24) and P(23)
   and P(22) and P(21) and P(20) and P(19) and P(18) and P(17) and
   P(16) and P(15) and P(14) and P(13) and P(12) and P(11) and G(10))
   or
   (P(29) and P(28) and P(27) and P(26) and P(25) and P(24) and P(23)
   and P(22) and P(21) and P(20) and P(19) and P(18) and P(17) and
   P(16) and P(15) and P(14) and P(13) and P(12) and P(11) and P(10)
   and G(9)) or
   (P(29) and P(28) and P(27) and P(26) and P(25) and P(24) and P(23)
   and P(22) and P(21) and P(20) and P(19) and P(18) and P(17) and
   P(16) and P(15) and P(14) and P(13) and P(12) and P(11) and P(10)
   and P(9) and G(8)) or
   (P(29) and P(28) and P(27) and P(26) and P(25) and P(24) and P(23)
   and P(22) and P(21) and P(20) and P(19) and P(18) and P(17) and
   P(16) and P(15) and P(14) and P(13) and P(12) and P(11) and P(10)
   and P(9) and P(8) and G(7)) or
   (P(29) and P(28) and P(27) and P(26) and P(25) and P(24) and P(23)
   and P(22) and P(21) and P(20) and P(19) and P(18) and P(17) and
   P(16) and P(15) and P(14) and P(13) and P(12) and P(11) and P(10)
   and P(9) and P(8) and P(7) and G(6)) or
   (P(29) and P(28) and P(27) and P(26) and P(25) and P(24) and P(23)
   and P(22) and P(21) and P(20) and P(19) and P(18) and P(17) and
   P(16) and P(15) and P(14) and P(13) and P(12) and P(11) and P(10)
   and P(9) and P(8) and P(7) and P(6) and G(5)) or
   (P(29) and P(28) and P(27) and P(26) and P(25) and P(24) and P(23)
   and P(22) and P(21) and P(20) and P(19) and P(18) and P(17) and
   P(16) and P(15) and P(14) and P(13) and P(12) and P(11) and P(10)
   and P(9) and P(8) and P(7) and P(6) and P(5) and G(4)) or
   (P(29) and P(28) and P(27) and P(26) and P(25) and P(24) and P(23)
   and P(22) and P(21) and P(20) and P(19) and P(18) and P(17) and
   P(16) and P(15) and P(14) and P(13) and P(12) and P(11) and P(10)
   and P(9) and P(8) and P(7) and P(6) and P(5) and P(4) and G(3)) or
   (P(29) and P(28) and P(27) and P(26) and P(25) and P(24) and P(23)
   and P(22) and P(21) and P(20) and P(19) and P(18) and P(17) and
   P(16) and P(15) and P(14) and P(13) and P(12) and P(11) and P(10)
   and P(9) and P(8) and P(7) and P(6) and P(5) and P(4) and P(3) and
   G(2)) or
   (P(29) and P(28) and P(27) and P(26) and P(25) and P(24) and P(23)
   and P(22) and P(21) and P(20) and P(19) and P(18) and P(17) and
   P(16) and P(15) and P(14) and P(13) and P(12) and P(11) and P(10)
   and P(9) and P(8) and P(7) and P(6) and P(5) and P(4) and P(3) and
   P(2) and G(1)) or
   (P(29) and P(28) and P(27) and P(26) and P(25) and P(24) and P(23)
   and P(22) and P(21) and P(20) and P(19) and P(18) and P(17) and
   P(16) and P(15) and P(14) and P(13) and P(12) and P(11) and P(10)
   and P(9) and P(8) and P(7) and P(6) and P(5) and P(4) and P(3) and
   P(2) and P(1) and G(0)) or
   (P(29) and P(28) and P(27) and P(26) and P(25) and P(24) and P(23)
   and P(22) and P(21) and P(20) and P(19) and P(18) and P(17) and
   P(16) and P(15) and P(14) and P(13) and P(12) and P(11) and P(10)
   and P(9) and P(8) and P(7) and P(6) and P(5) and P(4) and P(3) and
   P(2) and P(1) and P(0) and Cin);

C(31) <=    G(30) or
```

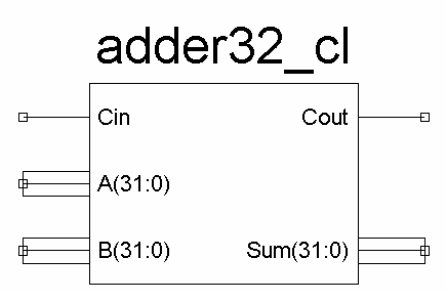
```
(P(30) and G(29)) or
(P(30) and P(29) and G(28)) or
(P(30) and P(29) and P(28) and G(27)) or
(P(30) and P(29) and P(28) and P(27) and G(26)) or
(P(30) and P(29) and P(28) and P(27) and P(26) and G(25)) or
(P(30) and P(29) and P(28) and P(27) and P(26) and P(25) and G(24))
or
(P(30) and P(29) and P(28) and P(27) and P(26) and P(25) and P(24)
and G(23)) or
(P(30) and P(29) and P(28) and P(27) and P(26) and P(25) and P(24)
and P(23) and G(22)) or
(P(30) and P(29) and P(28) and P(27) and P(26) and P(25) and P(24)
and P(23) and P(22) and G(21)) or
(P(30) and P(29) and P(28) and P(27) and P(26) and P(25) and P(24)
and P(23) and P(22) and P(21) and G(20)) or
(P(30) and P(29) and P(28) and P(27) and P(26) and P(25) and P(24)
and P(23) and P(22) and P(21) and P(20) and G(19)) or
(P(30) and P(29) and P(28) and P(27) and P(26) and P(25) and P(24)
and P(23) and P(22) and P(21) and P(20) and P(19) and G(18)) or
(P(30) and P(29) and P(28) and P(27) and P(26) and P(25) and P(24)
and P(23) and P(22) and P(21) and P(20) and P(19) and P(18) and
G(17)) or
(P(30) and P(29) and P(28) and P(27) and P(26) and P(25) and P(24)
and P(23) and P(22) and P(21) and P(20) and P(19) and P(18) and
P(17) and G(16)) or
(P(30) and P(29) and P(28) and P(27) and P(26) and P(25) and P(24)
and P(23) and P(22) and P(21) and P(20) and P(19) and P(18) and
P(17) and P(16) and G(15)) or
(P(30) and P(29) and P(28) and P(27) and P(26) and P(25) and P(24)
and P(23) and P(22) and P(21) and P(20) and P(19) and P(18) and
P(17) and P(16) and P(15) and G(14)) or
(P(30) and P(29) and P(28) and P(27) and P(26) and P(25) and P(24)
and P(23) and P(22) and P(21) and P(20) and P(19) and P(18) and
P(17) and P(16) and P(15) and P(14) and G(13)) or
(P(30) and P(29) and P(28) and P(27) and P(26) and P(25) and P(24)
and P(23) and P(22) and P(21) and P(20) and P(19) and P(18) and
P(17) and P(16) and P(15) and P(14) and P(13) and G(12)) or
(P(30) and P(29) and P(28) and P(27) and P(26) and P(25) and P(24)
and P(23) and P(22) and P(21) and P(20) and P(19) and P(18) and
P(17) and P(16) and P(15) and P(14) and P(13) and P(12) and G(11))
or
(P(30) and P(29) and P(28) and P(27) and P(26) and P(25) and P(24)
and P(23) and P(22) and P(21) and P(20) and P(19) and P(18) and
P(17) and P(16) and P(15) and P(14) and P(13) and P(12) and P(11)
and G(10)) or
(P(30) and P(29) and P(28) and P(27) and P(26) and P(25) and P(24)
and P(23) and P(22) and P(21) and P(20) and P(19) and P(18) and
P(17) and P(16) and P(15) and P(14) and P(13) and P(12) and P(11)
and P(10) and G(9)) or
(P(30) and P(29) and P(28) and P(27) and P(26) and P(25) and P(24)
and P(23) and P(22) and P(21) and P(20) and P(19) and P(18) and
P(17) and P(16) and P(15) and P(14) and P(13) and P(12) and P(11)
and P(10) and P(9) and G(8)) or
(P(30) and P(29) and P(28) and P(27) and P(26) and P(25) and P(24)
and P(23) and P(22) and P(21) and P(20) and P(19) and P(18) and
P(17) and P(16) and P(15) and P(14) and P(13) and P(12) and P(11)
and P(10) and P(9) and P(8) and G(7)) or
```

```
    (P(30) and P(29) and P(28) and P(27) and P(26) and P(25) and P(24)
    and P(23) and P(22) and P(21) and P(20) and P(19) and P(18) and
    P(17) and P(16) and P(15) and P(14) and P(13) and P(12) and P(11)
    and P(10) and P(9) and P(8) and P(7) and G(6)) or
    (P(30) and P(29) and P(28) and P(27) and P(26) and P(25) and P(24)
    and P(23) and P(22) and P(21) and P(20) and P(19) and P(18) and
    P(17) and P(16) and P(15) and P(14) and P(13) and P(12) and P(11)
    and P(10) and P(9) and P(8) and P(7) and P(6) and G(5)) or
    (P(30) and P(29) and P(28) and P(27) and P(26) and P(25) and P(24)
    and P(23) and P(22) and P(21) and P(20) and P(19) and P(18) and
    P(17) and P(16) and P(15) and P(14) and P(13) and P(12) and P(11)
    and P(10) and P(9) and P(8) and P(7) and P(6) and P(5) and G(4)) or
    (P(30) and P(29) and P(28) and P(27) and P(26) and P(25) and P(24)
    and P(23) and P(22) and P(21) and P(20) and P(19) and P(18) and
    P(17) and P(16) and P(15) and P(14) and P(13) and P(12) and P(11)
    and P(10) and P(9) and P(8) and P(7) and P(6) and P(5) and P(4) and
    G(3)) or
    (P(30) and P(29) and P(28) and P(27) and P(26) and P(25) and P(24)
    and P(23) and P(22) and P(21) and P(20) and P(19) and P(18) and
    P(17) and P(16) and P(15) and P(14) and P(13) and P(12) and P(11)
    and P(10) and P(9) and P(8) and P(7) and P(6) and P(5) and P(4) and
    P(3) and G(2)) or
    (P(30) and P(29) and P(28) and P(27) and P(26) and P(25) and P(24)
    and P(23) and P(22) and P(21) and P(20) and P(19) and P(18) and
    P(17) and P(16) and P(15) and P(14) and P(13) and P(12) and P(11)
    and P(10) and P(9) and P(8) and P(7) and P(6) and P(5) and P(4) and
    P(3) and P(2) and G(1)) or
    (P(30) and P(29) and P(28) and P(27) and P(26) and P(25) and P(24)
    and P(23) and P(22) and P(21) and P(20) and P(19) and P(18) and
    P(17) and P(16) and P(15) and P(14) and P(13) and P(12) and P(11)
    and P(10) and P(9) and P(8) and P(7) and P(6) and P(5) and P(4) and
    P(3) and P(2) and P(1) and G(0)) or
    (P(30) and P(29) and P(28) and P(27) and P(26) and P(25) and P(24)
    and P(23) and P(22) and P(21) and P(20) and P(19) and P(18) and
    P(17) and P(16) and P(15) and P(14) and P(13) and P(12) and P(11)
    and P(10) and P(9) and P(8) and P(7) and P(6) and P(5) and P(4) and
    P(3) and P(2) and P(1) and P(0) and Cin);

Cout <=     G(31) or
    (P(31) and G(31)) or
    (P(31) and P(30) and G(29)) or
    (P(31) and P(30) and P(29) and G(28)) or
    (P(31) and P(30) and P(29) and P(28) and G(27)) or
    (P(31) and P(30) and P(29) and P(28) and P(27) and G(26)) or
    (P(31) and P(30) and P(29) and P(28) and P(27) and P(26) and G(25))
    or
    (P(31) and P(30) and P(29) and P(28) and P(27) and P(26) and P(25)
    and G(24)) or
    (P(31) and P(30) and P(29) and P(28) and P(27) and P(26) and P(25)
    and P(24) and G(23)) or
    (P(31) and P(30) and P(29) and P(28) and P(27) and P(26) and P(25)
    and P(24) and P(23) and G(22)) or
    (P(31) and P(30) and P(29) and P(28) and P(27) and P(26) and P(25)
    and P(24) and P(23) and P(22) and G(21)) or
    (P(31) and P(30) and P(29) and P(28) and P(27) and P(26) and P(25)
    and P(24) and P(23) and P(22) and P(21) and G(20)) or
```

(P(31) and P(30) and P(29) and P(28) and P(27) and P(26) and P(25) and P(24) and P(23) and P(22) and P(21) and P(20) and G(19)) or
(P(31) and P(30) and P(29) and P(28) and P(27) and P(26) and P(25) and P(24) and P(23) and P(22) and P(21) and P(20) and P(19) and G(18)) or
(P(31) and P(30) and P(29) and P(28) and P(27) and P(26) and P(25) and P(24) and P(23) and P(22) and P(21) and P(20) and P(19) and P(18) and G(17)) or
(P(31) and P(30) and P(29) and P(28) and P(27) and P(26) and P(25) and P(24) and P(23) and P(22) and P(21) and P(20) and P(19) and P(18) and P(17) and G(16)) or
(P(31) and P(30) and P(29) and P(28) and P(27) and P(26) and P(25) and P(24) and P(23) and P(22) and P(21) and P(20) and P(19) and P(18) and P(17) and P(16) and G(15)) or
(P(31) and P(30) and P(29) and P(28) and P(27) and P(26) and P(25) and P(24) and P(23) and P(22) and P(21) and P(20) and P(19) and P(18) and P(17) and P(16) and P(15) and G(14)) or
(P(31) and P(30) and P(29) and P(28) and P(27) and P(26) and P(25) and P(24) and P(23) and P(22) and P(21) and P(20) and P(19) and P(18) and P(17) and P(16) and P(15) and P(14) and G(13)) or
(P(31) and P(30) and P(29) and P(28) and P(27) and P(26) and P(25) and P(24) and P(23) and P(22) and P(21) and P(20) and P(19) and P(18) and P(17) and P(16) and P(15) and P(14) and P(13) and G(12)) or
(P(31) and P(30) and P(29) and P(28) and P(27) and P(26) and P(25) and P(24) and P(23) and P(22) and P(21) and P(20) and P(19) and P(18) and P(17) and P(16) and P(15) and P(14) and P(13) and P(12) and G(11)) or
(P(31) and P(30) and P(29) and P(28) and P(27) and P(26) and P(25) and P(24) and P(23) and P(22) and P(21) and P(20) and P(19) and P(18) and P(17) and P(16) and P(15) and P(14) and P(13) and P(12) and P(11) and G(10)) or
(P(31) and P(30) and P(29) and P(28) and P(27) and P(26) and P(25) and P(24) and P(23) and P(22) and P(21) and P(20) and P(19) and P(18) and P(17) and P(16) and P(15) and P(14) and P(13) and P(12) and P(11) and P(10) and G(9)) or
(P(31) and P(30) and P(29) and P(28) and P(27) and P(26) and P(25) and P(24) and P(23) and P(22) and P(21) and P(20) and P(19) and P(18) and P(17) and P(16) and P(15) and P(14) and P(13) and P(12) and P(11) and P(10) and P(9) and G(8)) or
(P(31) and P(30) and P(29) and P(28) and P(27) and P(26) and P(25) and P(24) and P(23) and P(22) and P(21) and P(20) and P(19) and P(18) and P(17) and P(16) and P(15) and P(14) and P(13) and P(12) and P(11) and P(10) and P(9) and P(8) and G(7)) or
(P(31) and P(30) and P(29) and P(28) and P(27) and P(26) and P(25) and P(24) and P(23) and P(22) and P(21) and P(20) and P(19) and P(18) and P(17) and P(16) and P(15) and P(14) and P(13) and P(12) and P(11) and P(10) and P(9) and P(8) and P(7) and G(6)) or
(P(31) and P(30) and P(29) and P(28) and P(27) and P(26) and P(25) and P(24) and P(23) and P(22) and P(21) and P(20) and P(19) and P(18) and P(17) and P(16) and P(15) and P(14) and P(13) and P(12) and P(11) and P(10) and P(9) and P(8) and P(7) and P(6) and G(5)) or
(P(31) and P(30) and P(29) and P(28) and P(27) and P(26) and P(25) and P(24) and P(23) and P(22) and P(21) and P(20) and P(19) and P(18) and P(17) and P(16) and P(15) and P(14) and P(13) and P(12) and P(11) and P(10) and P(9) and P(8) and P(7) and P(6) and P(5) and G(4)) or

```
   (P(31) and P(30) and P(29) and P(28) and P(27) and P(26) and P(25)
   and P(24) and P(23) and P(22) and P(21) and P(20) and P(19) and
   P(18) and P(17) and P(16) and P(15) and P(14) and P(13) and P(12)
   and P(11) and P(10) and P(9) and P(8) and P(7) and P(6) and P(5) and
   P(4) and G(3)) or
   (P(31) and P(30) and P(29) and P(28) and P(27) and P(26) and P(25)
   and P(24) and P(23) and P(22) and P(21) and P(20) and P(19) and
   P(18) and P(17) and P(16) and P(15) and P(14) and P(13) and P(12)
   and P(11) and P(10) and P(9) and P(8) and P(7) and P(6) and P(5) and
   P(4) and P(3) and G(2)) or
   (P(31) and P(30) and P(29) and P(28) and P(27) and P(26) and P(25)
   and P(24) and P(23) and P(22) and P(21) and P(20) and P(19) and
   P(18) and P(17) and P(16) and P(15) and P(14) and P(13) and P(12)
   and P(11) and P(10) and P(9) and P(8) and P(7) and P(6) and P(5) and
   P(4) and P(3) and P(2) and G(1)) or
   (P(31) and P(30) and P(29) and P(28) and P(27) and P(26) and P(25)
   and P(24) and P(23) and P(22) and P(21) and P(20) and P(19) and
   P(18) and P(17) and P(16) and P(15) and P(14) and P(13) and P(12)
   and P(11) and P(10) and P(9) and P(8) and P(7) and P(6) and P(5) and
   P(4) and P(3) and P(2) and P(1) and G(0)) or
   (P(31) and P(30) and P(29) and P(28) and P(27) and P(26) and P(25)
   and P(24) and P(23) and P(22) and P(21) and P(20) and P(19) and
   P(18) and P(17) and P(16) and P(15) and P(14) and P(13) and P(12)
   and P(11) and P(10) and P(9) and P(8) and P(7) and P(6) and P(5) and
   P(4) and P(3) and P(2) and P(1) and P(0) and Cin);

end Behavioral;
```

Simbol :



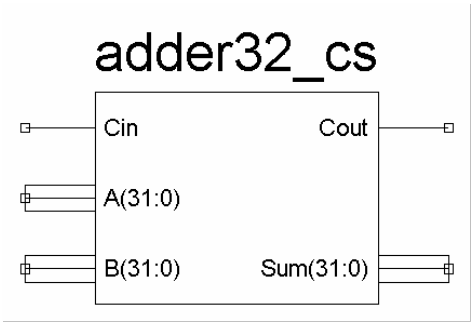
Carry Selector Adder 32 bit

```
Nama File   :     adder32_cs.vhd
Hirarki     :     # adder32_cs
                  |
                  +-> # full_adder
                      |
                      +-> # half_adder
Fungsi      :     Penjumlahan 32 bit
Program     :

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

```
entity adder32_cs is
    Port ( A, B : in std_logic_vector(31 downto 0);
           Cin : in std_logic;
           Sum : out std_logic_vector(31 downto 0);
           Cout : out std_logic);
end adder32_cs;

architecture Behavioral of adder32_cs is

      component full_adder
            port( A, B, Cin : in std_logic;
                  Sum, Cout : out std_logic);
      end component;

      signal C : std_logic_vector(16 downto 1);
      signal Ca : std_logic_vector(32 downto 17);
      signal Cb : std_logic_vector(32 downto 17);
      signal Sa : std_logic_vector(31 downto 16);
      signal Sb : std_logic_vector(31 downto 16);

begin

FA1 :  full_adder port map (A(0), B(0), Cin, Sum(0), C(1));
FA2 :  full_adder port map (A(1), B(1), C(1), Sum(1), C(2));
FA3 :  full_adder port map (A(2), B(2), C(2), Sum(2), C(3));
FA4 :  full_adder port map (A(3), B(3), C(3), Sum(3), C(4));
FA5 :  full_adder port map (A(4), B(4), C(4), Sum(4), C(5));
FA6 :  full_adder port map (A(5), B(5), C(5), Sum(5), C(6));
FA7 :  full_adder port map (A(6), B(6), C(6), Sum(6), C(7));
FA8 :  full_adder port map (A(7), B(7), C(7), Sum(7), C(8));
FA9 :  full_adder port map (A(8), B(8), C(8), Sum(8), C(9));
FA10 : full_adder port map (A(9), B(9), C(9), Sum(9), C(10));
FA11 : full_adder port map (A(10), B(10), C(10), Sum(10), C(11));
FA12 : full_adder port map (A(11), B(11), C(11), Sum(11), C(12));
FA13 : full_adder port map (A(12), B(12), C(12), Sum(12), C(13));
FA14 : full_adder port map (A(13), B(13), C(13), Sum(13), C(14));
FA15 : full_adder port map (A(14), B(14), C(14), Sum(14), C(15));
FA16 : full_adder port map (A(15), B(15), C(15), Sum(15), C(16));

FA17a : full_adder port map (A(16), B(16), '0', Sa(16), Ca(17));
FA18a : full_adder port map (A(17), B(17), Ca(17), Sa(17), Ca(18));
FA19a : full_adder port map (A(18), B(18), Ca(18), Sa(18), Ca(19));
FA20a : full_adder port map (A(19), B(19), Ca(19), Sa(19), Ca(20));
FA21a : full_adder port map (A(20), B(20), Ca(20), Sa(20), Ca(21));
FA22a : full_adder port map (A(21), B(21), Ca(21), Sa(21), Ca(22));
FA23a : full_adder port map (A(22), B(22), Ca(22), Sa(22), Ca(23));
FA24a : full_adder port map (A(23), B(23), Ca(23), Sa(23), Ca(24));
FA25a : full_adder port map (A(24), B(24), Ca(24), Sa(24), Ca(25));
FA26a : full_adder port map (A(25), B(25), Ca(25), Sa(25), Ca(26));
FA27a : full_adder port map (A(26), B(26), Ca(26), Sa(26), Ca(27));
FA28a : full_adder port map (A(27), B(27), Ca(27), Sa(27), Ca(28));
FA29a : full_adder port map (A(28), B(28), Ca(28), Sa(28), Ca(29));
FA30a : full_adder port map (A(29), B(29), Ca(29), Sa(29), Ca(30));
FA31a : full_adder port map (A(30), B(30), Ca(30), Sa(30), Ca(31));
FA32a : full_adder port map (A(31), B(31), Ca(31), Sa(31), Ca(32));
```

```
FA17b : full_adder port map (A(16), B(16), '1', Sb(16), Cb(17));
FA18b : full_adder port map (A(17), B(17), Cb(17), Sb(17), Cb(18));
FA19b : full_adder port map (A(18), B(18), Cb(18), Sb(18), Cb(19));
FA20b : full_adder port map (A(19), B(19), Cb(19), Sb(19), Cb(20));
FA21b : full_adder port map (A(20), B(20), Cb(20), Sb(20), Cb(21));
FA22b : full_adder port map (A(21), B(21), Cb(21), Sb(21), Cb(22));
FA23b : full_adder port map (A(22), B(22), Cb(22), Sb(22), Cb(23));
FA24b : full_adder port map (A(23), B(23), Cb(23), Sb(23), Cb(24));
FA25b : full_adder port map (A(24), B(24), Cb(24), Sb(24), Cb(25));
FA26b : full_adder port map (A(25), B(25), Cb(25), Sb(25), Cb(26));
FA27b : full_adder port map (A(26), B(26), Cb(26), Sb(26), Cb(27));
FA28b : full_adder port map (A(27), B(27), Cb(27), Sb(27), Cb(28));
FA29b : full_adder port map (A(28), B(28), Cb(28), Sb(28), Cb(29));
FA30b : full_adder port map (A(29), B(29), Cb(29), Sb(29), Cb(30));
FA31b : full_adder port map (A(30), B(30), Cb(30), Sb(30), Cb(31));
FA32b : full_adder port map (A(31), B(31), Cb(31), Sb(31), Cb(32));

   process (C(16), Sa(31 downto 16), Sb(31 downto 16), Ca(32), Cb(32))
   begin
      if (C(16) = '0') then
         Sum(31 downto 16) <= Sa(31 downto 16);
         Cout <= Ca(32);
      else
         Sum(31 downto 16) <= Sb(31 downto 16);
         Cout <= Cb(32);
      end if;
   end process;

end Behavioral;
```

Simbol :

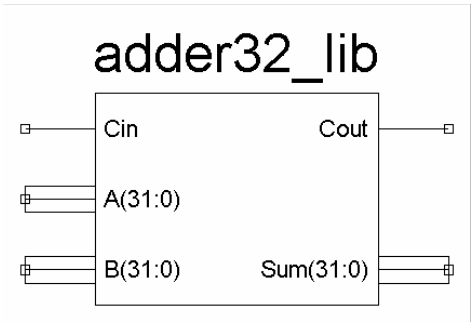


Adder 32 bit menggunakan Library

```
Nama File   :     adder32_lib.vhd
Hirarki     :     # adder32_lib
Fungsi      :     Penjumlahan 32 bit
Program     :

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity adder32_lib is
```

```
    Port ( A, B : in std_logic_vector(31 downto 0);
           Cin : in std_logic;
           Sum : out std_logic_vector(31 downto 0);
           Cout : out std_logic);
end adder32_lib;

architecture Behavioral of adder32_lib is

      signal Stmp : std_logic_vector(32 downto 0);

begin

      Stmp(32 downto 0) <= ('0' & A(31 downto 0)) +
                           ('0' & B(31 downto 0)) + Cin;

      Cout <= Stmp(32);
      Sum(31 downto 0) <= Stmp(31 downto 0);

end Behavioral;
```

Simbol :



Adder dan subtractor 32 bit menggunakan Ripple Carry Adder, Carry Lookahead Adder, Carry Selector Adder, dan Library

Nama File : addsub32_rc.vhd / addsub32_cl.vhd / addsub32_cs.vhd / addsub32_lib_par.vhd

Hirarki :

```
# addsub32
|
+-> # adder32
```

Fungsi : Penjumlahan, pengurangan dan melewatkan 32 bit

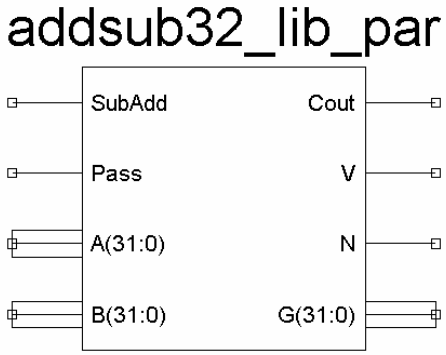
Pass	SubAdd	Operation
0	0	Adder
0	1	Subtractor
1	0	Pass
1	1	Pass

Program :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

```
entity addsub32_lib_par is
    Port ( A, B : in std_logic_vector(31 downto 0);
           SubAdd, Pass : in std_logic;
           G : out std_logic_vector(31 downto 0);
           Cout, V, N : out std_logic);
end addsub32_lib_par;

architecture Behavioral of addsub32_lib_par is

      signal Btmp, Gtmp : std_logic_vector (31 downto 0);
      signal CoutT : std_logic;

      component adder32_lib
            Port( A, B : in std_logic_vector(31 downto 0);
                  Cin : in std_logic;
                  Sum : out std_logic_vector(31 downto 0);
                  Cout : out std_logic);
      end component;

begin

      adder : adder32_lib port map (A => A, B => Btmp, Cin => SubAdd,
              Sum => Gtmp, Cout => CoutT);

      add_sub : process (SubAdd, Pass, B, SubAdd) begin
            if (Pass = '0') then
                  for I in 0 to 31 loop
                        Btmp(I) <= B(I) XOR SubAdd;
                  end loop;
            else Btmp <= X"00000000";
            end if;
      end process;

      G <= Gtmp;
      Cout <= CoutT;
      N <= Gtmp(31);
      V <= CoutT XOR A(31) XOR Btmp(31) XOR Gtmp(31);

end Behavioral;
```

Simbol :



Adder dan Subtractor 32 bit dimana Adder dan Subtractor menggunakan Library

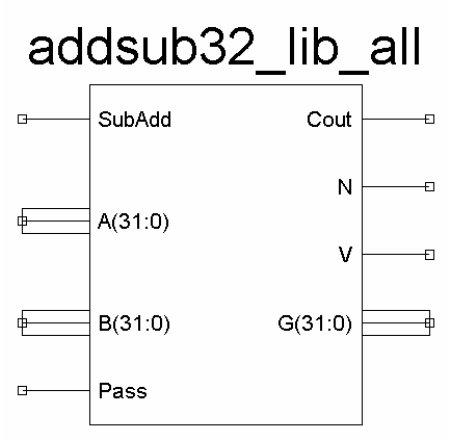
Nama File : addsub32_lib_all.vhd
Hirarki : # addsub32_lib_all
Fungsi : Penjumlahan, pengurangan dan melewatkan 32 bit

Pass	SubAdd	Operation
0	0	Adder
0	1	Subtractor
1	0	Pass
1	1	Pass

Program :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity addsub32_lib_all is
    Port ( A, B : in std_logic_vector(31 downto 0);
           SubAdd, Pass : in std_logic;
           G : out std_logic_vector(31 downto 0);
           Cout, N, V : out std_logic);
end addsub32_lib_all;

architecture Behavioral of addsub32_lib_all is

      signal Gtmp : std_logic_vector(32 downto 0);
      signal Btmp : std_logic_vector(31 downto 0);

begin

      process (SubAdd, Pass, A, B, Btmp) begin
            if(Pass = '1') then
                  Btmp <= X"00000000";
            else
                  Btmp <= B;
            end if;

            if (SubAdd = '0') then
                  Gtmp(32 downto 0) <= ('0' & A(31 downto 0)) +
                                       ('0' & Btmp(31 downto 0));
            else
                  Gtmp(32 downto 0) <= ('0' & A(31 downto 0)) -
                                       ('0' & Btmp(31 downto 0));
            end if;
      end process;

      G(31 downto 0) <= Gtmp(31 downto 0);
      Cout <= Gtmp(32);
      N <= Gtmp(31);
      V <= Gtmp(32) XOR A(31) XOR Btmp(31) XOR Gtmp(31);

end Behavioral;
```

Simbol :



Logic Unit 32 bit

Nama File : LogicUnit32.vhd
Hirarki : # LogicUnit32
Fungsi : Logika AND, OR, XOR, dan NOR

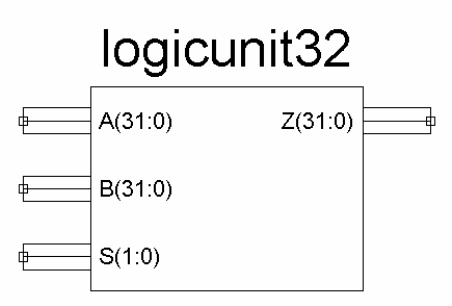
S	Operation
00	AND
01	OR
10	XOR
11	NOR

Program :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity LogicUnit32 is
    Port ( A, B : in std_logic_vector(31 downto 0);
           S : in std_logic_vector(1 downto 0);
           Z : out std_logic_vector(31 downto 0));
end LogicUnit32;

architecture Behavioral of LogicUnit32 is

begin

   process (S, A, B) begin
      case S is
         when "00"   => Z(31 downto 0) <= A(31 downto 0) AND
                                          B(31 downto 0);
         when "01"   => Z(31 downto 0) <= A(31 downto 0) OR
                                          B(31 downto 0);
         when "10"   => Z(31 downto 0) <= A(31 downto 0) XOR
                                          B(31 downto 0);
         when others => Z(31 downto 0) <= A(31 downto 0) NOR
                                          B(31 downto 0);
```

```
      end case;
   end process;

end Behavioral;
```

Simbol :



Aritmetic Logic Unit 32 bit

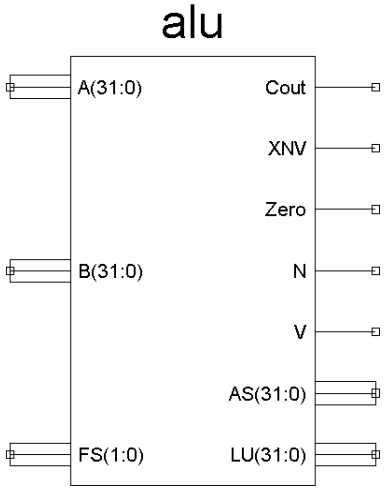
```
Nama File   :     ALU.vhd
Hirarki     :     # ALU
                  |
                  +-> # addsub32_lib_par
                  |   |
                  |   +-> # adder32_lib_par
                  |
                  +-> # LogicUnit32
Fungsi      :     Aritmatika (adder dan subtractor) & Logika (AND, OR,
                  XOR, dan NOR)

                  Output      FS          Operation
                  ------------------------------------------------
                    AS        00          Adder       \
                    AS        01          Subtractor  | --> Arithmetic
                    As        10          Pass        |
                    As        11          Pass (X)    /

                    LU        00          AND         \
                    LU        01          OR          | --> Logic
                    LU        10          XOR         |
                    LU        11          NOR         /
```

Program :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity ALU is
    Port ( A, B : in std_logic_vector(31 downto 0);
           FS : in std_logic_vector(1 downto 0);
           AS, LU : out std_logic_vector(31 downto 0);
           Cout, XNV, Zero, N, V : out std_logic);
end ALU;
```

```
architecture Behavioral of ALU is

      component addsub32_lib_par
          Port ( A, B : in std_logic_vector(31 downto 0);
                 SubAdd, Pass : in std_logic;
                 G : out std_logic_vector(31 downto 0);
                 Cout, V, N : out std_logic);
      end component;

      component LogicUnit32
          Port ( A, B : in std_logic_vector(31 downto 0);
               S : in std_logic_vector(1 downto 0);
               Z : out std_logic_vector(31 downto 0));
      end component;

      signal AStmp : std_logic_vector(31 downto 0);
      signal Ntmp, Vtmp : std_logic;

begin

   add_sub : addsub32_lib_par port map (A=>A, B=>B, SubAdd=>FS(0),
             Pass=>FS(1), G=>AStmp, Cout=>Cout, V=>Vtmp, N=>Ntmp);
   logic_unit : LogicUnit32 port map (A=>A, B=>B, S=>FS(1 downto 0),
                Z=>LU);
   N <= Ntmp;
   V <= Vtmp;
   XNV <= Ntmp XOR Vtmp;
   AS <= AStmp;
   Zero <= NOT( AStmp(31) OR AStmp(30) OR AStmp(29) OR AStmp(28) OR
                AStmp(27) OR AStmp(26) OR AStmp(25) OR AStmp(24) OR
                AStmp(23) OR AStmp(22) OR AStmp(21) OR AStmp(20) OR
                AStmp(19) OR AStmp(18) OR AStmp(17) OR AStmp(16) OR
                AStmp(15) OR AStmp(14) OR AStmp(13) OR AStmp(12) OR
                AStmp(11) OR AStmp(10) OR AStmp(9)  OR AStmp(8)  OR
                AStmp(7)  OR AStmp(6)  OR AStmp(5)  OR AStmp(4)  OR
                AStmp(3)  OR AStmp(2)  OR AStmp(1)  OR AStmp(0) );

end Behavioral;
```

Simbol :



Shifter 32 bit

```
Nama File   :     shifter32.vhd
Hirarki     :     # shifter32
Fungsi      :     shift logical left/right, shift aritmetic right

                    AS    LR    Operation
                  ----------------------------------------
                    0     0     Logical Shift Right
                    0     1     Logical Shift Left
                    1     0     Arithmetic Shift Right
                    1     1     Arithmetic Shift Left (X)
Program     :
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity shifter_32 is
    Port ( A : in std_logic_vector(31 downto 0);
           SH : in std_logic_vector(4 downto 0);
           LR, AS : in std_logic;
           Z : out std_logic_vector(31 downto 0));
end shifter_32;

architecture Behavioral of shifter_32 is

begin

   process(SH, LR, AS, A) begin
      if (AS = '0' and LR = '0') then

         case (SH) is
            when "00001" => Z <= '0' & A(31 downto 1);
            when "00010" => Z <= "00" & A(31 downto 2);
            when "00011" => Z <= "000" & A(31 downto 3);
            when "00100" => Z <= "0000" & A(31 downto 4);
            when "00101" => Z <= "00000" & A(31 downto 5);
            when "00110" => Z <= "000000" & A(31 downto 6);
            when "00111" => Z <= "0000000" & A(31 downto 7);
            when "01000" => Z <= "00000000" & A(31 downto 8);
            when "01001" => Z <= "000000000" & A(31 downto 9);
            when "01010" => Z <= "0000000000" & A(31 downto 10);
            when "01011" => Z <= "00000000000" & A(31 downto 11);
            when "01100" => Z <= "000000000000" & A(31 downto 12);
            when "01101" => Z <= "0000000000000" & A(31 downto 13);
            when "01110" => Z <= "00000000000000" & A(31 downto 14);
            when "01111" => Z <= "000000000000000" & A(31 downto 15);
            when "10000" => Z <= "0000000000000000" & A(31 downto 16);
            when "10001" => Z <= "00000000000000000" & A(31 downto 17);
            when "10010" => Z <= "000000000000000000" &
                                 A(31 downto 18);
            when "10011" => Z <= "0000000000000000000" &
                                 A(31 downto 19);
            when "10100" => Z <= "00000000000000000000" &
                                 A(31 downto 20);
```

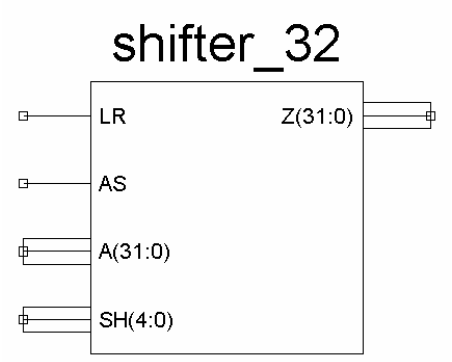
```
         when "10101" => Z <= "00000000000000000000" &
                              A(31 downto 21);
         when "10110" => Z <= "00000000000000000000" &
                              A(31 downto 22);
         when "10111" => Z <= "00000000000000000000" &
                              A(31 downto 23);
         when "11000" => Z <= "00000000000000000000" &
                              A(31 downto 24);
         when "11001" => Z <= "00000000000000000000" &
                              A(31 downto 25);
         when "11010" => Z <= "00000000000000000000" &
                              A(31 downto 26);
         when "11011" => Z <= "00000000000000000000" &
                              A(31 downto 27);
         when "11100" => Z <= "00000000000000000000" &
                              A(31 downto 28);
         when "11101" => Z <= "00000000000000000000" &
                              A(31 downto 29);
         when "11110" => Z <= "00000000000000000000" &
                              A(31 downto 30);
         when "11111" => Z <= "00000000000000000000" &
                              A(31);
         when others  => Z <= A(31 downto 0);
      end case;

   elsif (AS = '0' and LR = '1') then
      case (SH) is
         when "00001" => Z <= A(30 downto 0) & '0';
         when "00010" => Z <= A(29 downto 0) & "00";
         when "00011" => Z <= A(28 downto 0) & "000";
         when "00100" => Z <= A(27 downto 0) & "0000";
         when "00101" => Z <= A(26 downto 0) & "00000";
         when "00110" => Z <= A(25 downto 0) & "000000";
         when "00111" => Z <= A(24 downto 0) & "0000000";
         when "01000" => Z <= A(23 downto 0) & "00000000";
         when "01001" => Z <= A(22 downto 0) & "000000000";
         when "01010" => Z <= A(21 downto 0) & "0000000000";
         when "01011" => Z <= A(20 downto 0) & "00000000000";
         when "01100" => Z <= A(19 downto 0) & "000000000000";
         when "01101" => Z <= A(18 downto 0) & "0000000000000";
         when "01110" => Z <= A(17 downto 0) & "00000000000000";
         when "01111" => Z <= A(16 downto 0) & "000000000000000";
         when "10000" => Z <= A(15 downto 0) & "0000000000000000";
         when "10001" => Z <= A(14 downto 0) & "00000000000000000";
         when "10010" => Z <= A(13 downto 0) & "000000000000000000";
         when "10011" => Z <= A(12 downto 0) &
                              "0000000000000000000";
         when "10100" => Z <= A(11 downto 0) &
                              "00000000000000000000";
         when "10101" => Z <= A(10 downto 0) &
                              "00000000000000000000";
         when "10110" => Z <= A(9 downto 0)  &
                              "00000000000000000000";
         when "10111" => Z <= A(8 downto 0)  &
                              "00000000000000000000";
         when "11000" => Z <= A(7 downto 0)  &
                              "00000000000000000000";
```

```
        when "11001" => Z <= A(6 downto 0)  &
                             "00000000000000000000000000";
        when "11010" => Z <= A(5 downto 0)  &
                             "00000000000000000000000000";
        when "11011" => Z <= A(4 downto 0)  &
                             "00000000000000000000000000";
        when "11100" => Z <= A(3 downto 0)  &
                             "00000000000000000000000000";
        when "11101" => Z <= A(2 downto 0)  &
                             "00000000000000000000000000";
        when "11110" => Z <= A(1 downto 0)  &
                             "00000000000000000000000000";
        when "11111" => Z <= A(0)           &
                             "00000000000000000000000000";
        when others  => Z <= A(31 downto 0);
     end case;

  else  -- AS = '1' and LR = '0'
     case (SH) is
        when "00001" => Z <= A(31) & A(31 downto 1);
        when "00010" => Z <= A(31) & A(31) & A(31 downto 2);
        when "00011" => Z <= A(31) & A(31) & A(31) &
              A(31 downto 3);
        when "00100" => Z <= A(31) & A(31) & A(31) & A(31) &
              A(31 downto 4);
        when "00101" => Z <= A(31) & A(31) & A(31) & A(31) & A(31)
              & A(31 downto 5);
        when "00110" => Z <= A(31) & A(31) & A(31) & A(31) & A(31)
              & A(31) & A(31 downto 6);
        when "00111" => Z <= A(31) & A(31) & A(31) & A(31) & A(31)
              & A(31) & A(31) & A(31 downto 7);
        when "01000" => Z <= A(31) & A(31) & A(31) & A(31) & A(31)
              & A(31) & A(31) & A(31) & A(31 downto 8);
        when "01001" => Z <= A(31) & A(31) & A(31) & A(31) & A(31)
              & A(31) & A(31) & A(31) & A(31) & A(31 downto 9);
        when "01010" => Z <= A(31) & A(31) & A(31) & A(31) & A(31)
              & A(31) & A(31) & A(31) & A(31) & A(31) &
              A(31 downto 10);
        when "01011" => Z <= A(31) & A(31) & A(31) & A(31) & A(31)
              & A(31) & A(31) & A(31) & A(31) & A(31) & A(31) &
              A(31 downto 11);
        when "01100" => Z <= A(31) & A(31) & A(31) & A(31) & A(31)
              & A(31) & A(31) & A(31) & A(31) & A(31) & A(31) &
              A(31) & A(31 downto 12);
        when "01101" => Z <= A(31) & A(31) & A(31) & A(31) & A(31)
              & A(31) & A(31) & A(31) & A(31) & A(31) & A(31) &
              A(31) & A(31) & A(31 downto 13);
        when "01110" => Z <= A(31) & A(31) & A(31) & A(31) & A(31)
              & A(31) & A(31) & A(31) & A(31) & A(31) & A(31) &
              A(31) & A(31) & A(31) & A(31 downto 14);
        when "01111" => Z <= A(31) & A(31) & A(31) & A(31) & A(31)
              & A(31) & A(31) & A(31) & A(31) & A(31) & A(31) &
              A(31) & A(31) & A(31) & A(31) & A(31 downto 15);
        when "10000" => Z <= A(31) & A(31) & A(31) & A(31) & A(31)
              & A(31) & A(31) & A(31) & A(31) & A(31) & A(31) &
              A(31) & A(31) & A(31) & A(31) & A(31) &
              A(31 downto 16);
```

```
when "10001" => Z <= A(31) & A(31) & A(31) & A(31) & A(31)
      & A(31) & A(31) & A(31) & A(31) & A(31) & A(31) &
      A(31) & A(31) & A(31) & A(31) & A(31) & A(31) &
      A(31 downto 17);
when "10010" => Z <= A(31) & A(31) & A(31) & A(31) & A(31)
      & A(31) & A(31) & A(31) & A(31) & A(31) & A(31) &
      A(31) & A(31) & A(31) & A(31) & A(31) & A(31) & A(31)
      & A(31 downto 18);
when "10011" => Z <= A(31) & A(31) & A(31) & A(31) & A(31)
      & A(31) & A(31) & A(31) & A(31) & A(31) & A(31) &
      A(31) & A(31) & A(31) & A(31) & A(31) & A(31) & A(31)
      & A(31) & A(31 downto 19);
when "10100" => Z <= A(31) & A(31) & A(31) & A(31) & A(31)
      & A(31) & A(31) & A(31) & A(31) & A(31) & A(31) &
      A(31) & A(31) & A(31) & A(31) & A(31) & A(31) & A(31)
      & A(31) & A(31) & A(31 downto 20);
when "10101" => Z <= A(31) & A(31) & A(31) & A(31) & A(31)
      & A(31) & A(31) & A(31) & A(31) & A(31) & A(31) &
      A(31) & A(31) & A(31) & A(31) & A(31) & A(31) & A(31)
      & A(31) & A(31) & A(31) & A(31 downto 21);
when "10110" => Z <= A(31) & A(31) & A(31) & A(31) & A(31)
      & A(31) & A(31) & A(31) & A(31) & A(31) & A(31) &
      A(31) & A(31) & A(31) & A(31) & A(31) & A(31) & A(31)
      & A(31) & A(31) & A(31) & A(31) & A(31 downto 22);
when "10111" => Z <= A(31) & A(31) & A(31) & A(31) & A(31)
      & A(31) & A(31) & A(31) & A(31) & A(31) & A(31) &
      A(31) & A(31) & A(31) & A(31) & A(31) & A(31) & A(31)
      & A(31) & A(31) & A(31) & A(31) & A(31) &
      A(31 downto 23);
when "11000" => Z <= A(31) & A(31) & A(31) & A(31) & A(31)
      & A(31) & A(31) & A(31) & A(31) & A(31) & A(31) &
      A(31) & A(31) & A(31) & A(31) & A(31) & A(31) & A(31)
      & A(31) & A(31) & A(31) & A(31) & A(31) & A(31) &
      A(31 downto 24);
when "11001" => Z <= A(31) & A(31) & A(31) & A(31) & A(31)
      & A(31) & A(31) & A(31) & A(31) & A(31) & A(31) &
      A(31) & A(31) & A(31) & A(31) & A(31) & A(31) & A(31)
      & A(31) & A(31) & A(31) & A(31) & A(31) & A(31) &
      A(31) & A(31 downto 25);
when "11010" => Z <= A(31) & A(31) & A(31) & A(31) & A(31)
      & A(31) & A(31) & A(31) & A(31) & A(31) & A(31) &
      A(31) & A(31) & A(31) & A(31) & A(31) & A(31) & A(31)
      & A(31) & A(31) & A(31) & A(31) & A(31) & A(31) &
      A(31) & A(31) & A(31 downto 26);
when "11011" => Z <= A(31) & A(31) & A(31) & A(31) & A(31)
      & A(31) & A(31) & A(31) & A(31) & A(31) & A(31) &
      A(31) & A(31) & A(31) & A(31) & A(31) & A(31) & A(31)
      & A(31) & A(31) & A(31) & A(31) & A(31) & A(31) &
      A(31) & A(31) & A(31) & A(31 downto 27);
when "11100" => Z <= A(31) & A(31) & A(31) & A(31) & A(31)
      & A(31) & A(31) & A(31) & A(31) & A(31) & A(31) &
      A(31) & A(31) & A(31) & A(31) & A(31) & A(31) & A(31)
      & A(31) & A(31) & A(31) & A(31) & A(31) & A(31) &
      A(31) & A(31) & A(31) & A(31) & A(31 downto 28);
when "11101" => Z <= A(31) & A(31) & A(31) & A(31) & A(31)
      & A(31) & A(31) & A(31) & A(31) & A(31) & A(31) &
      A(31) & A(31) & A(31) & A(31) & A(31) & A(31) & A(31)
```

```
                & A(31) & A(31) & A(31) & A(31) & A(31) & A(31) &
                A(31) & A(31) & A(31) & A(31) & A(31) &
                A(31 downto 29);
            when "11110" => Z <= A(31) & A(31) & A(31) & A(31) & A(31)
                & A(31) & A(31) & A(31) & A(31) & A(31) & A(31) &
                A(31) & A(31) & A(31) & A(31) & A(31) & A(31) & A(31)
                & A(31) & A(31) & A(31) & A(31) & A(31) & A(31) &
                A(31) & A(31) & A(31) & A(31) & A(31) & A(31) &
                A(31 downto 30);
            when "11111" => Z <= A(31) & A(31) & A(31) & A(31) & A(31)
                & A(31) & A(31) & A(31) & A(31) & A(31) & A(31) &
                A(31) & A(31) & A(31) & A(31) & A(31) & A(31) & A(31)
                & A(31) & A(31) & A(31) & A(31) & A(31) & A(31) &
                A(31) & A(31) & A(31) & A(31) & A(31) & A(31) & A(31)
                & A(31);
            when others  => Z <= A(31 downto 0);
         end case;

      end if;

   end process;

end Behavioral;
```

Simbol :



Barrel Shifter 32 bit dengan 1 bit Selector Multiplexer

Nama File : BarrelShift32_1
Hirarki : # BarrelShift32_1
Fungsi : shift logical left/right, shift aritmetic right

AS	LR	Operation
0	0	Logical Shift Right
0	1	Logical Shift Left
1	0	Arithmetic Shift Right
1	1	Arithmetic Shift Left (X)

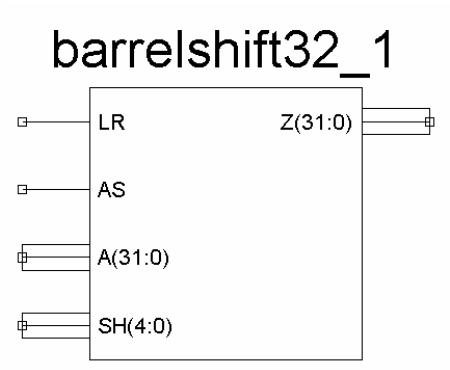
Program :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

```
entity BarrelShift32_1 is
    Port ( A : in std_logic_vector(31 downto 0);
           SH : in std_logic_vector(4 downto 0);
         LR, AS : in std_logic;
           Z : out std_logic_vector(31 downto 0));
end BarrelShift32_1;

architecture Behavioral of BarrelShift32_1 is

      signal SH_2C : std_logic_vector(5 downto 0);
      signal Z1 : std_logic_vector(62 downto 0);
      signal Z2 : std_logic_vector(46 downto 0);
      signal Z3 : std_logic_vector(38 downto 0);
      signal Z4 : std_logic_vector(34 downto 0);
      signal Z5 : std_logic_vector(32 downto 0);

      signal AS_32 : std_logic_vector(31 downto 0);

begin

   Arithmetic_Shift : process(AS, A(31)) begin
      for i in 31 downto 0 loop
         AS_32(i) <= AS AND A(31);
      end loop;
   end process;

   Left_or_Right : process(LR, SH) begin
      if (LR = '0') then
         SH_2C <= '0'&SH;
      else
         SH_2C <= ('1'&(not SH)) + '1';
      end if;
   end process;

   barel_shift : process (SH_2C, A, Z1, Z2, Z3, Z4, Z5, AS_32) begin
      case SH_2C(5) is
         when '0'    => Z1(62 downto 0) <= AS_32(30 downto 0) &
                        A(31 downto 0);
         when others => Z1(62 downto 0) <= A(30 downto 0) &
                        AS_32(31 downto 0);
      end case;

      case SH_2C(4) is
         when '0' =>    Z2(46 downto 0) <= Z1(46 downto 0);
         when others => Z2(46 downto 0) <= Z1(62 downto 16);
      end case;

      case SH_2C(3) is
         when '0' =>    Z3(38 downto 0) <= Z2(38 downto 0);
         when others => Z3(38 downto 0) <= Z2(46 downto 8);
      end case;

      case SH_2C(2) is
         when '0' =>    Z4(34 downto 0) <= Z3(34 downto 0);
         when others => Z4(34 downto 0) <= Z3(38 downto 4);
      end case;
```

```
      case SH_2C(1) is
         when '0' =>    Z5(32 downto 0) <= Z4(32 downto 0);
         when others => Z5(32 downto 0) <= Z4(34 downto 2);
      end case;

      case SH_2C(0) is
         when '0' =>    Z(31 downto 0) <= Z5(31 downto 0);
         when others => Z(31 downto 0) <= Z5(32 downto 1);
      end case;

   end process;

end Behavioral;
```

Simbol :



Barrel Shifter 32 bit dengan 2 bit Selector Multiplexer

Nama File : BarrelShift32_2
Hirarki : # BarrelShift32_2
Fungsi : shift logical left/right, shift aritmetic right

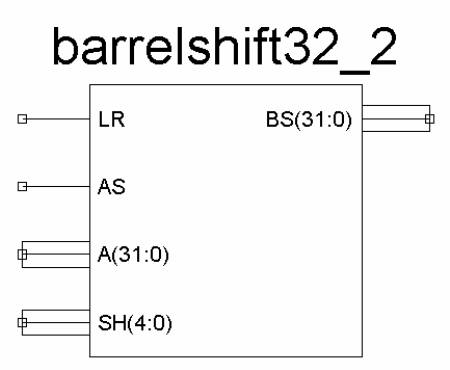
AS	LR	Operation
0	0	Logical Shift Right
0	1	Logical Shift Left
1	0	Arithmetic Shift Right
1	1	Arithmetic Shift Left (X)

Program :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity BarrelShift32_2 is
    Port ( A : in std_logic_vector(31 downto 0);
           SH : in std_logic_vector(4 downto 0);
         LR, AS : in std_logic;
           BS : out std_logic_vector(31 downto 0));
end BarrelShift32_2;

architecture Behavioral of BarrelShift32_2 is
```

```
	signal SH_2C : std_logic_vector(5 downto 0);
	signal Z1 : std_logic_vector(46 downto 0);
	signal Z2 : std_logic_vector(34 downto 0);

	signal AS_32 : std_logic_vector(31 downto 0);

begin

	Arithmetic_Shift : process(AS, A) begin
		for i in 31 downto 0 loop
			AS_32(i) <= AS AND A(31);
		end loop;
	end process;

	Left_or_Right : process(LR, SH) begin
		if (LR = '0') then
			SH_2C <= '0'&SH;
		else
			SH_2C <= ('1'&(not SH)) + '1';
		end if;
	end process;

	barel_shift : process (SH_2C, A, Z1, Z2, AS, AS_32) begin
		case SH_2C(5 downto 4) is
			when "00" =>   Z1(46 downto 0) <= AS_32(14 downto 0)
			               & A(31 downto 0);
			when "01" =>   Z1(46 downto 0) <= AS_32(30 downto 0)
			               & A(31 downto 16);
			when "10" =>   Z1(46 downto 0) <= A(14 downto 0) &
			               AS_32(31 downto 0);
			when others => Z1(46 downto 0) <= A(30 downto 0) &
			               AS_32(31 downto 16);
		end case;

		case SH_2C(3 downto 2) is
			when "00" =>   Z2(34 downto 0) <= Z1(34 downto 0);
			when "01" =>   Z2(34 downto 0) <= Z1(38 downto 4);
			when "10" =>   Z2(34 downto 0) <= Z1(42 downto 8);
			when others => Z2(34 downto 0) <= Z1(46 downto 12);
		end case;

		case SH_2C(1 downto 0) is
			when "00" =>   BS(31 downto 0) <= Z2(31 downto 0);
			when "01" =>   BS(31 downto 0) <= Z2(32 downto 1);
			when "10" =>   BS(31 downto 0) <= Z2(33 downto 2);
			when others => BS(31 downto 0) <= Z2(34 downto 3);
		end case;

	end process;

end Behavioral;
```

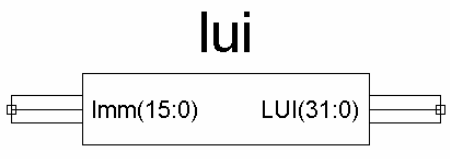
Simbol :



Load Upper Immediate

```
Nama File   :     LUI.vhd
Hirarki     :     # LUI
Fungsi      :     Memindahkan 16 bit immediate ke 16 bit MSB
Program     :

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity LUI is
    Port (  Imm : in std_logic_vector(15 downto 0);
            LUI : out std_logic_vector(31 downto 0));
end LUI;

architecture Behavioral of LUI is

begin

      LUI <= Imm & X"0000";

end Behavioral;
```

Simbol :



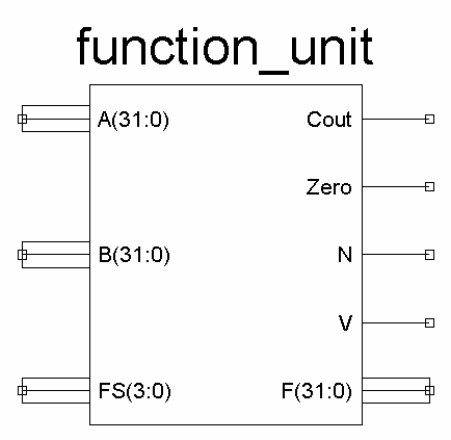
Function Unit

```
Nama File   :     Function_Unit.vhd
```

```
Hirarki     :     # Function_Unit
                  |
                  +-> # ALU
                  |   |
                  |   +-> # addsub32_lib_par
                  |   |   |
                  |   |   +-> # adder32_lib
                  |   |
                  |   +-> # LogicUnit32
                  |
                  +-> # BarrelShift32_2
                  |
                  +-> # LUI
Fungsi      :     Aritmatika, logika, shifter, set, dan LUI

                  FS          Operation
                  ---------------------------------
                  00 00       Adder                   \
                  00 01       Subtractor              | --> Arithmetic
                  00 10       Pass                    |
                  00 11       Pass (X)                /

                  01 00       AND                     \
                  01 01       OR                      | --> Logic
                  01 10       XOR                     |
                  01 11       NOR                     /

                  10 00       Logical Shift Right     \
                  10 01       Logical Shift Left      | --> Shifter
                  10 10       Arithmetic Shift Right  |
                  10 11       (X)                     /

                  11 0X       XNV                     > --> SET
                  11 1X       Load Upper Immediate    > --> Load

Program     :

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity Function_Unit is
      Port ( A, B : in std_logic_vector(31 downto 0);
             FS : in std_logic_vector(3 downto 0);
             F : out std_logic_vector(31 downto 0);
             Cout, Zero, N, V : out std_logic);
end Function_Unit;

architecture Behavioral of Function_Unit is

      component ALU
          Port ( A, B : in std_logic_vector(31 downto 0);
               FS : in std_logic_vector(1 downto 0);
                 AS, LU : out std_logic_vector(31 downto 0);
               Cout, XNV, Zero, N, V : out std_logic);
```

```
	end component;

	component BarrelShift32_2
	    Port ( A : in std_logic_vector(31 downto 0);
	           SH : in std_logic_vector(4 downto 0);
	         LR, AS : in std_logic;
	           BS : out std_logic_vector(31 downto 0));
	end component;

	component LUI
	    Port ( Imm : in std_logic_vector(15 downto 0);
	         LUI : out std_logic_vector(31 downto 0));
	end component;

	signal AStmp, LUtmp, BStmp, LUItmp, LXtmp : std_logic_vector(31
downto 0);
	signal MF : std_logic_vector(1 downto 0);
	signal XNV : std_logic;

begin

	A_L_U : ALU port map (A=>A, B=>B, FS=>FS(1 downto 0), AS=>AStmp,
	        LU=>LUtmp, Cout=>Cout, XNV=>XNV, Zero=>Zero, N=>N, V=>V);
	Barrel_Shift : BarrelShift32_2 port map (A=>A, SH=>B(4 downto 0),
	               LR=>FS(0), AS=>FS(1), BS=>BStmp);
	LUI : LUI port map (Imm=>B(15 downto 0), LUI=>LUItmp);

	MUX_LX : process(FS, LUItmp, XNV)
	begin
	      if(FS(1) = '0') then
	            LXtmp <= XNV & X"0000000" & B"000";
	      else
	            LXtmp <= LUItmp;
	      end if;
	end process MUX_LX;

	MF <= FS(3 downto 2);

	MUX_F : process(MF, AStmp, LUtmp, BStmp, LXtmp)
	begin
	      case (MF) is
	            when "01" => F <= LUtmp;
	            when "10" => F <= BStmp;
	            when "11" => F <= LXtmp;
	            when others => F <= AStmp;
	      end case;
	end process MUX_F;

end Behavioral;
```

Simbol :



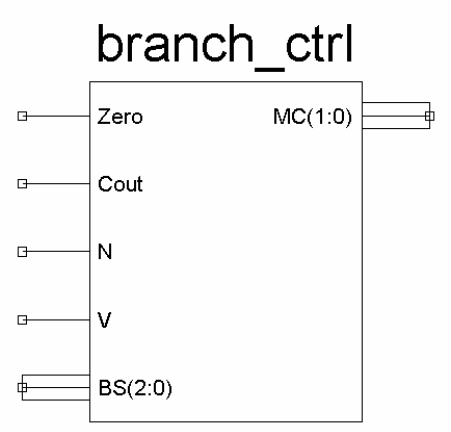
Branch Control

Nama File : branch_ctrl.vhd
Hirarki : # branch_ctrl
Fungsi : Pengaturan untuk menentukan alamat instruksi selanjutnya
Program :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Branch_Ctrl is
      port( BS : in std_logic_vector(2 downto 0);
            Zero, Cout, N, V : in std_logic;
            MC : out std_logic_vector(1 downto 0)
      );
end Branch_Ctrl;

architecture Behavioral of Branch_Ctrl is

begin

   Branch_Ctrl_Process : process(BS, Zero, Cout, N, V) begin

      -- Branch Higher
      if   ((BS = "000") AND ((Cout = '1') AND (Zero = '0')))
            then MC <= "01";
      elsif((BS = "000") AND ((Cout = '0') OR  (Zero = '1')))
            then MC <= "00";

      -- Branch Higher Equal
      elsif((BS = "001") AND (Cout = '1')) then MC <= "01";
      elsif((BS = "001") AND (Cout = '0')) then MC <= "00";

      -- Branch Equal
      elsif((BS = "010") AND (Zero = '1')) then MC <= "01";
      elsif((BS = "010") AND (Zero = '0')) then MC <= "00";
```

```
      -- Branch Greater
      elsif((BS = "011") AND (((N XOR V) = '0') AND (Zero = '0')))
            then MC <= "01";
      elsif((BS = "011") AND (((N XOR V) = '1') OR  (Zero = '1')))
            then MC <= "00";

      -- Branch Greater Equal
      elsif((BS = "100") AND (( N XOR V) = '0')) then MC <= "01";
      elsif((BS = "100") AND (( N XOR V) = '1')) then MC <= "00";

      -- Jump
      elsif( BS = "101") then MC <= "10";

      -- Jump Register
      elsif( BS = "110") then MC <= "11";

      -- Next Instruction
      else MC <= "00";

      end if;

   end process Branch_Ctrl_Process;

end Behavioral;
```

Simbol :

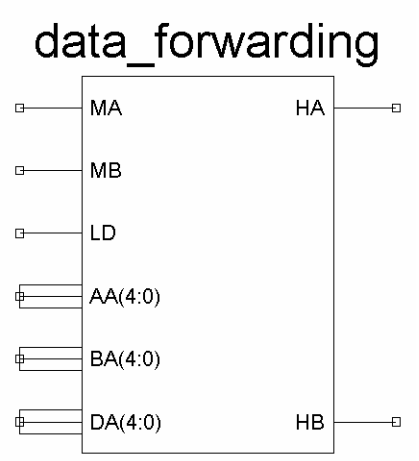


Data Forwarding

```
Nama File   :     Data_Forwarding.vhd
Hirarki     :     # Data_Forwarding
Fungsi      :     Memeriksa terjadinya Data Depedency
Program     :

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Data_Forwarding is
      port( MA, MB, LD : in std_logic;
            AA, BA, DA : in std_logic_vector(4 downto 0);
```

```
            HA, HB : out std_logic
      );
end Data_Forwarding;

architecture Behavioral of Data_Forwarding is
begin

      DF_Sel : process(LD, DA, AA, BA, MA, MB) begin
            if((AA = DA) AND (MA = '0') AND (LD = '1') AND
              (DA /= "00000")) then
                  HA <= '1';
            else
                  HA <= '0';
            end if;

            if((BA = DA) AND (MB = '0') AND (LD = '1') AND
              (DA /= "00000")) then
                  HB <= '1';
            else
                  HB <= '0';
            end if;
      end process DF_Sel;

end Behavioral;
```

Simbol :



Instruction Decoder

```
Nama File   :     ID.vhd
Hirarki     :     # ID
Fungsi      :     Pengkodean dari Opcode menjadi Control Word
Program     :
```

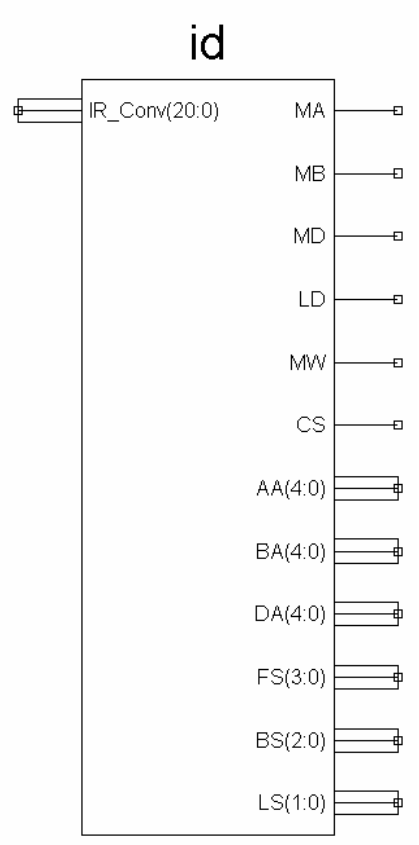
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity ID is
      Port( IR_Conv : in std_logic_vector(20 downto 0);
```

```
            AA, BA, DA : out std_logic_vector(4 downto 0);
            FS : out std_logic_vector(3 downto 0);
            BS : out std_logic_vector(2 downto 0);
            LS : out std_logic_vector(1 downto 0);
            MA, MB, MD, LD, MW, CS : out std_logic);
end ID;

architecture Behavioral of ID is

      signal OPCODE : std_logic_vector(5 downto 0);
      signal DEC_OPCODE : std_logic_vector(14 downto 0);

begin

   OPCODE <= IR_Conv(20 downto 15);
   DA <= IR_Conv(14 downto 10);
   AA <= IR_Conv(9 downto 5);
   BA <= IR_Conv(4 downto 0);

   process (OPCODE) begin
      -- ADD   RD, RA, RB     Addition
      if    (OPCODE = "000000") then
            DEC_OPCODE <= B"1_0_111_0_00_0000_0_0_0";
      -- ADI   RD, RA, Im     Add Immediate Sign
      elsif (OPCODE = "000001") then
            DEC_OPCODE <= B"1_0_111_0_00_0000_1_0_1";
      -- ADIU  RD, RA, Im     Add Immediate Unsign
      elsif (OPCODE = "000010") then
            DEC_OPCODE <= B"1_0_111_0_00_0000_1_0_0";
      -- SUB   RD, RA, RB     Subtract
      elsif (OPCODE = "000011") then
            DEC_OPCODE <= B"1_0_111_0_00_0001_0_0_0";
      -- SUI   RD, RA, Im     Sub Immediate Sign
      elsif (OPCODE = "000100") then
            DEC_OPCODE <= B"1_0_111_0_00_0001_1_0_1";
      -- SUIU  RD, RA, Im     Sub Immediate Unsign
      elsif (OPCODE = "000101") then
            DEC_OPCODE <= B"1_0_111_0_00_0001_1_0_0";

      -- AND   RD, RA, RB     AND
      elsif (OPCODE = "000110") then
            DEC_OPCODE <= B"1_0_111_0_00_0100_0_0_0";
      -- ANDI  RD, RA, Im     AND Immdiate
      elsif (OPCODE = "000111") then
            DEC_OPCODE <= B"1_0_111_0_00_0100_1_0_0";
      -- OR    RD, RA, RB     OR
      elsif (OPCODE = "001000") then
            DEC_OPCODE <= B"1_0_111_0_00_0101_0_0_0";
      -- ORI   RD, RA, Im     OR Immdiate
      elsif (OPCODE = "001001") then
            DEC_OPCODE <= B"1_0_111_0_00_0101_1_0_0";
      -- XOR   RD, RA, RB     XOR
      elsif (OPCODE = "001010") then
            DEC_OPCODE <= B"1_0_111_0_00_0110_0_0_0";
      -- XORI  RD, RA, Im     XOR Immediate
      elsif (OPCODE = "001011") then
            DEC_OPCODE <= B"1_0_111_0_00_0110_1_0_0";
```

```
-- NOR   RD, RA, RB     NOR
elsif (OPCODE = "001100") then
      DEC_OPCODE <= B"1_0_111_0_00_0111_0_0_0";
-- NORI  RD, RA, Im     NOR Immediate
elsif (OPCODE = "001101") then
      DEC_OPCODE <= B"1_0_111_0_00_0111_1_0_0";

-- SLR   RD, RA, Im     Shift Logical Right
elsif (OPCODE = "001110") then
      DEC_OPCODE <= B"1_0_111_0_00_1000_1_0_0";
-- SLRV  RD, RA, RB     Shift Logical Right Variable
elsif (OPCODE = "001111") then
      DEC_OPCODE <= B"1_0_111_0_00_1000_0_0_0";
-- SLL   RD, RA, Im     Shift Logical Left
elsif (OPCODE = "010000") then
      DEC_OPCODE <= B"1_0_111_0_00_1001_1_0_0";
-- SLLV  RD, RA, RB     Shift Logical Left Variable
elsif (OPCODE = "010001") then
      DEC_OPCODE <= B"1_0_111_0_00_1001_0_0_0";
-- SAR   RD, RA, Im     Shift Arithmetic Right
elsif (OPCODE = "010010") then
      DEC_OPCODE <= B"1_0_111_0_00_1010_1_0_0";
-- SARV  RD, RA, RB     Shift Arithmetic Right Variable
elsif (OPCODE = "010011") then
      DEC_OPCODE <= B"1_0_111_0_00_1010_0_0_0";

-- LUI   RD, Im         Load Upper Immediate
elsif (OPCODE = "010100") then
      DEC_OPCODE <= B"1_0_111_0_00_1111_1_0_0";
-- LA    RD             Load Address
elsif (OPCODE = "010101") then
      DEC_OPCODE <= B"1_0_111_0_00_0010_0_1_0";

-- SB    RA, RB         Store Byte
elsif (OPCODE = "010110") then
      DEC_OPCODE <= B"0_0_111_1_00_0010_0_0_0";
-- SH    RA, RB         Store Half Word
elsif (OPCODE = "010111") then
      DEC_OPCODE <= B"0_0_111_1_01_0010_0_0_0";
-- SW    RA, RB         Store Word
elsif (OPCODE = "011000") then
      DEC_OPCODE <= B"0_0_111_1_10_0010_0_0_0";
-- LB    RD, RA         Load Byte
elsif (OPCODE = "011001") then
      DEC_OPCODE <= B"1_1_111_0_00_0010_0_0_0";
-- LH    RD, RA         Load Half Word
elsif (OPCODE = "011010") then
      DEC_OPCODE <= B"1_1_111_0_01_0010_0_0_0";
-- LW    RD, RA         Load Word
elsif (OPCODE = "011011") then
      DEC_OPCODE <= B"1_1_111_0_10_0010_0_0_0";

-- SLT   RD, RA, RB     Set if Less Then
elsif (OPCODE = "011100") then
      DEC_OPCODE <= B"1_0_111_0_00_1101_0_0_0";
-- SLTI  RD, RA, Im     Set if Less Then Immediate
elsif (OPCODE = "011101") then
```

```
            DEC_OPCODE <= B"1_0_111_0_00_1101_1_0_1";

      -- DI                   Disable Interrupt
      elsif (OPCODE = "011110") then
            DEC_OPCODE <= B"0_0_111_0_00_0010_0_0_0";
      -- EI                   Enable Interrupt
      elsif (OPCODE = "011111") then
            DEC_OPCODE <= B"0_0_111_0_00_0010_0_0_0";

      -- BE   RA, RB, Im      Branch on Equal
      elsif (OPCODE = "100000") then
            DEC_OPCODE <= B"0_0_010_0_00_0001_0_0_0";
      -- BH   RA, RB, Im      Branch on Higer Then
      elsif (OPCODE = "100001") then
            DEC_OPCODE <= B"0_0_000_0_00_0001_0_0_0";
      -- BHE  RA, RB, Im      Branch on Higer Then Equal
      elsif (OPCODE = "100010") then
            DEC_OPCODE <= B"0_0_001_0_00_0001_0_0_0";
      -- BG   RA, RB, Im      Branch on Greater Then
      elsif (OPCODE = "100011") then
            DEC_OPCODE <= B"0_0_011_0_00_0001_0_0_0";
      -- BGE  RA, RB, Im      Branch on Greater Then Equal
      elsif (OPCODE = "100100") then
            DEC_OPCODE <= B"0_0_100_0_00_0001_0_0_0";
      -- JMP  Target          Jump
      elsif (OPCODE = "100101") then
            DEC_OPCODE <= B"0_0_101_0_00_0010_0_0_0";
      -- JL   Terget          Jump and Link
      elsif (OPCODE = "100110") then
            DEC_OPCODE <= B"1_0_101_0_00_0010_0_1_0";
      -- JR   RB              Jump Register
      elsif (OPCODE = "100111") then
            DEC_OPCODE <= B"0_0_110_0_00_0010_0_0_0";
      -- JRL  RB              Jump Register and Link
      elsif (OPCODE = "101000") then
            DEC_OPCODE <= B"1_0_110_0_00_0010_0_1_0";

      else  DEC_OPCODE <= B"0_0_000_0_00_0000_0_0_0";
      end if;
   end process;

   LD <= DEC_OPCODE(14);
   MD <= DEC_OPCODE(13);
   BS <= DEC_OPCODE(12 downto 10);
   MW <= DEC_OPCODE(9);
   LS <= DEC_OPCODE(8 downto 7);
   FS <= DEC_OPCODE(6 downto 3);
   MB <= DEC_OPCODE(2);
   MA <= DEC_OPCODE(1);
   CS <= DEC_OPCODE(0);

end Behavioral;
```

Simbol :

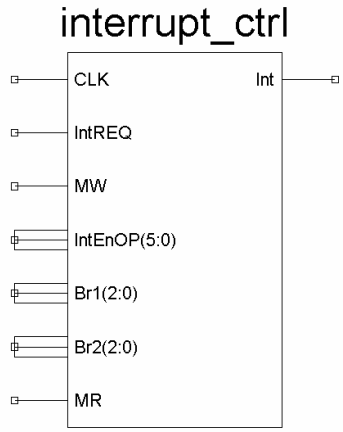


Interrupt Control

```
Nama File   :     interrupt_ctrl.vhd
Hirarki     :     # interrupt_ctrl
Fungsi      :     Mengatur proses terjadinya interrupt
Program     :

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Interrupt_Ctrl is
      port( CLK : in std_logic;
            IntREQ, MW, MR : in std_logic;
            IntEnOP : in std_logic_vector(5 downto 0);
            Br1, Br2 : in std_logic_vector(2 downto 0);
            Int : out std_logic);
end Interrupt_Ctrl;

architecture Behavioral of Interrupt_Ctrl is

      signal Int_Sel, E_Int_Next, E_Int_Curr, BG1, BG2 : std_logic;

begin

   Int_Sel_DEC : process(IntEnOP, Br1, Br2) begin
```

```
      if(((IntEnOP = "011110") OR (IntEnOP = "011111")) AND
        (Br1 = "111") AND (Br2 = "111")) then
         Int_Sel <= '1';
      else
         Int_Sel <= '0';
      end if;
   end process;

   MUX_E_Int : process(Int_Sel, E_Int_Curr, IntEnOP) begin
      if(Int_Sel = '0') then
         E_Int_Next <= E_Int_Curr;
      else
         E_Int_Next <= IntEnOP(0);
      end if;
   end process;

   E_Int_Step : process begin
      wait until CLK 'EVENT AND CLK = '0';
         E_Int_Curr <= E_Int_Next;
   end process;

   Branch_Decoder : process(Br1, Br2) begin
      if(Br1 /= "111") then
         BG1 <= '0';
      else
         BG1 <= '1';
      end if;

      if(Br2 /= "111") then
         BG2 <= '0';
      else
         BG2 <= '1';
      end if;
   end process;

   Int <= IntREQ AND E_Int_Curr AND (NOT IntEnOP(5)) AND BG1 AND
          BG2 AND (NOT MW) AND (NOT MR) AND (NOT Int_Sel);

end Behavioral;
```

Simbol :

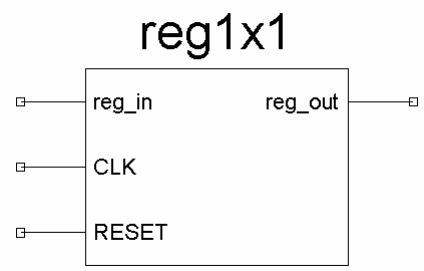


Register 1, 3, 4, 5, 26, 32 bit

```
Nama File   :     reg1x1.vhd
Hirarki     :     # reg1x1
Fungsi      :     Menyimpan proses pada Pipeline
Program     :
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity reg1x1 is
    Port ( reg_in : in std_logic;
           reg_out : out std_logic;
           CLK : in std_logic;
           RESET : in std_logic);
end reg1x1;

architecture Behavioral of reg1x1 is

begin

      process(CLK, RESET, reg_in) begin
            if(RESET = '1') then
                  reg_out <= '0';
            elsif(CLK'EVENT and CLK = '1') then
                  reg_out <= reg_in;
            end if;
      end process;

end Behavioral;
```

Simbol :



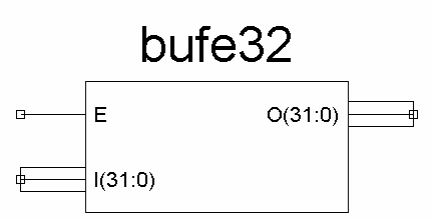
Three State Buffer 32 bit

```
Nama File   :     BUFE32.vhd
Hirarki     :     # BUFE32
Fungsi      :     Buffer bus data
Program     :
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

```
entity BUFE32 is
      port( E : in std_logic;
            I : in std_logic_vector(31 downto 0);
            O : out std_logic_vector(31 downto 0)
      );
end BUFE32;

architecture Behavioral of BUFE32 is

      component BUFE
            port( E : in std_logic;
                  I : in std_logic;
                  O : out std_logic
            );
      end component;

begin

      bit00 : BUFE port map (E=>E, I=>I(0),  O=>O(0));
      bit01 : BUFE port map (E=>E, I=>I(1),  O=>O(1));
      bit02 : BUFE port map (E=>E, I=>I(2),  O=>O(2));
      bit03 : BUFE port map (E=>E, I=>I(3),  O=>O(3));
      bit04 : BUFE port map (E=>E, I=>I(4),  O=>O(4));
      bit05 : BUFE port map (E=>E, I=>I(5),  O=>O(5));
      bit06 : BUFE port map (E=>E, I=>I(6),  O=>O(6));
      bit07 : BUFE port map (E=>E, I=>I(7),  O=>O(7));
      bit08 : BUFE port map (E=>E, I=>I(8),  O=>O(8));
      bit09 : BUFE port map (E=>E, I=>I(9),  O=>O(9));
      bit10 : BUFE port map (E=>E, I=>I(10), O=>O(10));
      bit11 : BUFE port map (E=>E, I=>I(11), O=>O(11));
      bit12 : BUFE port map (E=>E, I=>I(12), O=>O(12));
      bit13 : BUFE port map (E=>E, I=>I(13), O=>O(13));
      bit14 : BUFE port map (E=>E, I=>I(14), O=>O(14));
      bit15 : BUFE port map (E=>E, I=>I(15), O=>O(15));
      bit16 : BUFE port map (E=>E, I=>I(16), O=>O(16));
      bit17 : BUFE port map (E=>E, I=>I(17), O=>O(17));
      bit18 : BUFE port map (E=>E, I=>I(18), O=>O(18));
      bit19 : BUFE port map (E=>E, I=>I(19), O=>O(19));
      bit20 : BUFE port map (E=>E, I=>I(20), O=>O(20));
      bit21 : BUFE port map (E=>E, I=>I(21), O=>O(21));
      bit22 : BUFE port map (E=>E, I=>I(22), O=>O(22));
      bit23 : BUFE port map (E=>E, I=>I(23), O=>O(23));
      bit24 : BUFE port map (E=>E, I=>I(24), O=>O(24));
      bit25 : BUFE port map (E=>E, I=>I(25), O=>O(25));
      bit26 : BUFE port map (E=>E, I=>I(26), O=>O(26));
      bit27 : BUFE port map (E=>E, I=>I(27), O=>O(27));
      bit28 : BUFE port map (E=>E, I=>I(28), O=>O(28));
      bit29 : BUFE port map (E=>E, I=>I(29), O=>O(29));
      bit30 : BUFE port map (E=>E, I=>I(30), O=>O(30));
      bit31 : BUFE port map (E=>E, I=>I(31), O=>O(31));

end Behavioral;
```

Simbol :



Three State Buffer 3 bit

```
Nama File   :     BUFE3.vhd
Hirarki     :     # BUFE3
Fungsi      :     Buffer
Program     :

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity BUFE3 is
      port( E : in std_logic;
            I : in std_logic_vector(2 downto 0);
            O : out std_logic_vector(2 downto 0)
      );
end BUFE3;

architecture Behavioral of BUFE3 is

      component BUFE
            port( E : in std_logic;
                  I : in std_logic;
                  O : out std_logic
            );
      end component;

begin

      bit00 : BUFE port map (E=>E, I=>I(0),  O=>O(0));
      bit01 : BUFE port map (E=>E, I=>I(1),  O=>O(1));
      bit02 : BUFE port map (E=>E, I=>I(2),  O=>O(2));

end Behavioral;
```

Simbol :

RISC

```
Nama File   :     risc.vhd
Hirarki     :     # RISC
                  |
                  +-> # Function_Unit
                  |   |
                  |   +-> # ALU
                  |   |   |
                  |   |   +-> # addsub32_lib_par
                  |   |   |   |
                  |   |   |   +-> # adder32_lib
                  |   |   |
                  |   |   +-> # LogicUnit32
                  |   |
                  |   +-> # BarrelShift32_2
                  |   |
                  |   +-> # LUI
                  |
                  +-> # register_file_gab_DF
                  |   |
                  |   +-> # const_unit
                  |   |
                  |   +-> # register_file_DPRAM
                  |
                  +-> # branch_ctrl
                  |
                  +-> # Data_Forwarding
                  |
                  +-> # ID
                  |
                  +-> # reg1x1
                  |
                  +-> # reg1x2
                  |
                  +-> # reg1x3
                  |
                  +-> # reg1x4
                  |
                  +-> # reg1x5
                  |
                  +-> # reg1x26
                  |
                  +-> # reg1x32
                  |
                  +-> # BUFE32
Program     :
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity RISC is
    Port ( Inst : in std_logic_vector(31 downto 0);
           Inst_Add : out std_logic_vector(31 downto 0);
```

```
            BSel : out std_logic_vector(3 downto 0);
            Data : inout std_logic_vector(31 downto 0);
            Data_Add : out std_logic_vector(31 downto 0);

            IntREQ : in std_logic;
            IntACK : out std_logic;
            IntADD : in std_logic_vector(2 downto 0);

            CLK : in std_logic;
            RESET : in std_logic;

            MW : inout std_logic;
            MR : inout std_logic
          );
end RISC;

architecture Behavioral of RISC is

      component ID
            Port( IR_Conv : in std_logic_vector(20 downto 0);
                  AA, BA, DA : out std_logic_vector(4 downto 0);
                  FS : out std_logic_vector(3 downto 0);
                  BS : out std_logic_vector(2 downto 0);
                  LS : out std_logic_vector(1 downto 0);
                  MA, MB, MD, LD, MW, CS : out std_logic);
      end component;

      component Register_File_Gab_DF
            Port( D_data, D_data_DF, PC_1 : in
                        std_logic_vector(31 downto 0);
                  Imm : in std_logic_vector(15 downto 0);
                  AA, BA, DA : in std_logic_vector(4 downto 0);
                  MA, MB, HA, HB, CS, LD, CLK : in std_logic;
                  Bus_A, Bus_B : out std_logic_vector(31 downto 0));
      end component;

      component Function_Unit
            Port( A, B : in std_logic_vector(31 downto 0);
                  FS : in std_logic_vector(3 downto 0);
                  F : out std_logic_vector(31 downto 0);
                  Cout, Zero, N, V : out std_logic);
      end component;

      component Branch_Ctrl
            port( BS : in std_logic_vector(2 downto 0);
                  Zero, Cout, N, V : in std_logic;
                  MC : out std_logic_vector(1 downto 0));
      end component;

      component Data_Forwarding
            port( MA, MB, LD : in std_logic;
                  AA, BA, DA : in std_logic_vector(4 downto 0);
                  HA, HB : out std_logic);
      end component;

      component Interrupt_Ctrl
            port( CLK : in std_logic;
```

```
            IntREQ, MW, MR : in std_logic;
            IntEnOP : in std_logic_vector(5 downto 0);
            Br1, Br2 : in std_logic_vector(2 downto 0);
            Int : out std_logic);
end component;

component reg1x32
      Port( reg_in : in std_logic_vector(31 downto 0);
            reg_out : out std_logic_vector(31 downto 0);
            CLK : in std_logic;
            RESET : in std_logic);
end component;

component reg1x26
      Port( reg_in : in std_logic_vector(25 downto 0);
            reg_out : out std_logic_vector(25 downto 0);
            CLK : in std_logic;
            RESET : in std_logic);
end component;

component reg1x5
      Port( reg_in : in std_logic_vector(4 downto 0);
            reg_out : out std_logic_vector(4 downto 0);
            CLK : in std_logic;
            RESET : in std_logic);
end component;

component reg1x4
      Port( reg_in : in std_logic_vector(3 downto 0);
            reg_out : out std_logic_vector(3 downto 0);
            CLK : in std_logic;
            RESET : in std_logic);
end component;

component reg1x3
      Port( reg_in : in std_logic_vector(2 downto 0);
            reg_out : out std_logic_vector(2 downto 0);
            CLK : in std_logic;
            RESET : in std_logic);
end component;

component reg1x2
      Port( reg_in : in std_logic_vector(1 downto 0);
            reg_out : out std_logic_vector(1 downto 0);
            CLK : in std_logic;
            RESET : in std_logic);
end component;

component reg1x1
      Port( reg_in : in std_logic;
            reg_out : out std_logic;
            CLK : in std_logic;
            RESET : in std_logic);
end component;

component BUFE32
      port( E : in std_logic;
```

```
            I : in std_logic_vector(31 downto 0);
            O : out std_logic_vector(31 downto 0)
      );
end component;

--Data Dependency
signal Bus_D_DF : std_logic_vector(31 downto 0);
signal HA, HB : std_logic;

--Branch Hazard
signal BP, LD_In_1_BH, MW_In_BH : std_logic;
signal BS_In_BH : std_logic_vector(2 downto 0);
signal IR_In_BH : std_logic_vector(31 downto 0);

--Branch & Jump
signal MC : std_logic_vector(1 downto 0);
signal JA : std_logic_vector(25 downto 0);
signal PC_In, JA_add, JRA : std_logic_vector(31 downto 0);

--ID
signal AA, BA : std_logic_vector(4 downto 0);
signal MA, MB, CS : std_logic;

--IF
signal PC : std_logic_vector(31 downto 0);
signal PC_1_In, PC_Inc, IR_In : std_logic_vector(31 downto 0);

--DO
signal PC_1, IR, PC_2_In, Bus_A_In, Bus_B_In :
       std_logic_vector(31 downto 0);
signal DA_In_1 : std_logic_vector(4 downto 0);
signal FS_In : std_logic_vector(3 downto 0);
signal BS_In : std_logic_vector(2 downto 0);
signal LS_In : std_logic_vector(1 downto 0);
signal MD_In_1, LD_In_1, MW_In : std_logic;

--EX
signal Bus_A, Bus_B, F_In, PC_2, BrA :
       std_logic_vector(31 downto 0);
signal FS : std_logic_vector(3 downto 0);
signal BS : std_logic_vector(2 downto 0);
signal DA_Out_1, DA_In_2 : std_logic_vector(4 downto 0);
signal Zero, Cout, N, V, MD_Out_1, MD_In_2, LD_Out_1, LD_In_2 :
       std_logic;

--Memory Control In/Out
signal DataTmpOut, DataTmpIn : std_logic_vector(31 downto 0);
signal LS : std_logic_vector(1 downto 0);

--WB
signal F, Bus_D, DataReg : std_logic_vector(31 downto 0);
signal DA : std_logic_vector(4 downto 0);
signal MD, LD : std_logic;

--Interrupt
signal IntADD_1, PC_In_1, PC_1_In_1 :
       std_logic_vector(31 downto 0);
```

```
	signal Int : std_logic;

begin

	--Data Dependency dengan Data Forwarding

	Data_Forwarding_Ctrl : Data_forwarding port map (MA=>MA, MB=>MB,
	    AA=>AA, BA=>BA, LD=>LD_Out_1, DA=>DA_Out_1, HA=>HA, HB=>HB);

	MUX_D_DF_Sel : process(MD_Out_1, F_In, Data) begin
		if (MD_Out_1 = '0') then
			Bus_D_DF <= F_In;
		else
			Bus_D_DF <= Data;
		end if;
	end process MUX_D_DF_Sel;

	--Branch Hazard

	BP <= MC(0) OR MC(1);

	--Branch & Jump

	Branch_Ctrl_PM : Branch_Ctrl port map (BS=>BS, Zero=>Zero,
	                 Cout=>Cout, N=>N, V=>V, MC=>MC);

	MUX_C : process(MC, PC_Inc, BrA, JRA, JA_add) begin
		if    (MC = "00") then PC_In_1 <= PC_Inc;
		elsif (MC = "01") then PC_In_1 <= BrA;
		elsif (MC = "10") then PC_In_1 <= JA_add;
		else                   PC_In_1 <= JRA;
		end if;
	end process;

	--Interrupt

	IntADD_1 <= X"0000000" & IntADD & '0';

	MUX_I_PC : process(Int, IntADD_1, PC_In_1) begin
		if(Int = '1') then
			PC_In <= IntADD_1;
		else
			PC_In <= PC_In_1;
		end if;
	end process;

	MUX_I_PC_1 : process(Int, PC, PC_1_In_1) begin
		if(Int = '1') then
			PC_1_In <= PC;
		else
			PC_1_In <= PC_1_In_1;
		end if;
	end process;

	MUX_IR : process(BP, Int, IR_In_BH) begin
		if(Int = '0') then
			if(BP = '1') then
```

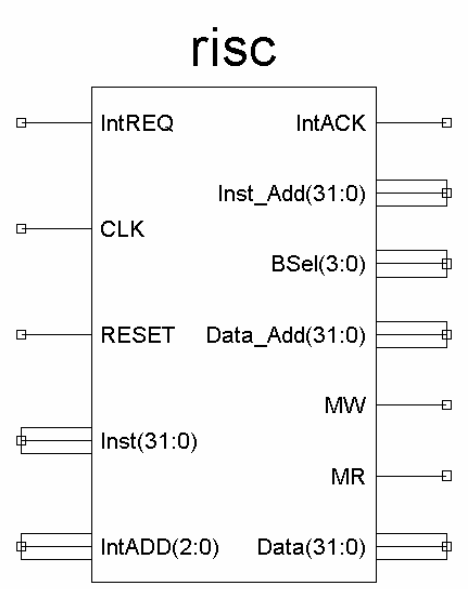
```
                IR_In <= X"00000000";
            else
                IR_In <= IR_In_BH;
            end if;
      else
            IR_In <= X"57C00000";
      end if;
end process;

--Interrupt Control

Interrupt_Control : Interrupt_Ctrl port map (CLK=>CLK,
                    IntREQ=>IntREQ, MW=>MW, MR=>MR,
                    IntEnOP=>Inst(31 downto 26), Br1=>BS_In,
                    Br2=>BS, Int=>Int);

IntACK <= Int;

--IF

Inst_Add <= PC;
PC_Inc <= PC + "1";
PC_1_In_1 <= PC_Inc;
IR_In_BH <= Inst;

--DO

Instruction_Decoder : ID port map (IR_Conv=>IR(31 downto 11),
                      LD=>LD_In_1_BH, DA=>DA_In_1, MD=>MD_In_1,
                      BS=>BS_In_BH, MW=>MW_In_BH, LS=>LS_In,
                      FS=>FS_In, MA=>MA, MB=>MB, AA=>AA, BA=>BA,
                      CS=>CS);

Branch_Prediction : process(BP, LD_In_1_BH, BS_In_BH, MW_In_BH)
begin
      if(BP = '1') then
            LD_In_1 <= '0';
            BS_In <= "111";
            MW_In <= '0';
      else
            LD_In_1 <= LD_In_1_BH;
            BS_In <= BS_In_BH;
            MW_In <= MW_In_BH;
      end if;
end process;

Register_File : Register_File_Gab_DF port map (D_data=>Bus_D,
      D_data_DF=>Bus_D_DF , PC_1=>PC_1, Imm=>IR(15 downto 0),
      AA=>AA, BA=>BA, DA=>DA, MA=>MA, MB=>MB, HA=>HA, HB=>HB,
      CS=>CS, LD=>LD, CLK=>CLK, Bus_A=>Bus_A_In, Bus_B=>Bus_B_In
      );

PC_2_In <= PC_1;

--EX

FunctionUnit : Function_Unit port map (A=>Bus_A, B=>Bus_B,
```

```
      FS=>FS, F=>F_In, Cout=>Cout, Zero=>Zero, N=>N, V=>V);

JRA <= Bus_B;

BrA <= (PC_2) + (JA(25) & JA(25) & JA(25) & JA(25) & JA(25) &
       JA(25) & JA(25) & JA(25) & JA(25) & JA(25) & JA(25) &
       JA(25) & JA(25) & JA(25) & JA(25) & JA(25) &
       JA(25 downto 21) & JA(10 downto 0));

JA_add <= (PC_2) + (JA(25) & JA(25) & JA(25) & JA(25) & JA(25) &
          JA(25) & JA(25 downto 0));

LD_In_2 <= LD_Out_1;
DA_In_2 <= DA_Out_1;
MD_In_2 <= MD_Out_1;

TS_Buffer : BUFE32 port map (E=>MW, I=>DataTmpOut, O=>Data);

Data_Add(31 downto 0) <= Bus_A(31 downto 0);

MR <= MD_In_2;

--Memory Control Out (Instruksi Store)

MCO_sel : process(Bus_A, LS) begin
   if   (Bus_A(1) & Bus_A(0) & LS = "0000") then BSel <= "0001";
   elsif(Bus_A(1) & Bus_A(0) & LS = "0001") then BSel <= "0011";
   elsif(Bus_A(1) & Bus_A(0) & LS = "0010") then BSel <= "1111";
   elsif(Bus_A(1) & Bus_A(0) & LS = "0100") then BSel <= "0010";
   elsif(Bus_A(1) & Bus_A(0) & LS = "1000") then BSel <= "0100";
   elsif(Bus_A(1) & Bus_A(0) & LS = "1001") then BSel <= "1100";
   elsif(Bus_A(1) & Bus_A(0) & LS = "1100") then BSel <= "1000";
   else                                          BSel <= "0000";
   end if;
end process;

MCO_MUX : process(Bus_A, Bus_B) begin
      if   ((Bus_A(1) = '1') AND (Bus_A(0) = '0')) then
            DataTmpOut(31 downto 24) <= Bus_B(15 downto 8);
      elsif((Bus_A(1) = '1') AND (Bus_A(0) = '1')) then
            DataTmpOut(31 downto 24) <= Bus_B(7 downto 0);
      else  DataTmpOut(31 downto 24) <= Bus_B(31 downto 24);
      end if;

      if(Bus_A(1) = '0') then
            DataTmpOut(23 downto 16) <= Bus_B(23 downto 16);
      else  DataTmpOut(23 downto 16) <= Bus_B(7 downto 0);
      end if;

      if(Bus_A(0) = '0') then
            DataTmpOut(15 downto 8) <= Bus_B(15 downto 8);
      else  DataTmpOut(15 downto 8) <= Bus_B(7 downto 0);
      end if;
end process;

DataTmpOut(7 downto 0) <= Bus_B(7 downto 0);
```

```
--Memory Control In (Instruksi Load)

MCI : process(Bus_A, LS, Data) begin
      if   (Bus_A(1) & Bus_A(0) & LS = "0000") then
            DataTmpIn <= X"000000" & Data(7 downto 0);
      elsif(Bus_A(1) & Bus_A(0) & LS = "0001") then
            DataTmpIn <= X"0000" & Data(15 downto 0);
      elsif(Bus_A(1) & Bus_A(0) & LS = "0100") then
            DataTmpIn <= X"000000" & Data(15 downto 8);
      elsif(Bus_A(1) & Bus_A(0) & LS = "1000") then
            DataTmpIn <= X"000000" & Data(23 downto 16);
      elsif(Bus_A(1) & Bus_A(0) & LS = "1001") then
            DataTmpIn <= X"0000" & Data(31 downto 16);
      elsif(Bus_A(1) & Bus_A(0) & LS = "1100") then
            DataTmpIn <= X"000000" & Data(31 downto 24);
      else  DataTmpIn <= Data(31 downto 0);
      end if;
end process;

--WB

MUX_D : process(MD, F, DataReg) begin
      if (MD = '0') then Bus_D <= F;
      else               Bus_D <= DataReg;
      end if;
end process;

--Register Pipeline

Reg_PC : reg1x32 port map (reg_in=>PC_In, reg_out=>PC, CLK=>CLK,
         RESET=>RESET);
Reg_PC_1 : reg1x32 port map (reg_in=>PC_1_In, reg_out=>PC_1,
           CLK=>CLK, RESET=>RESET);
Reg_PC_2 : reg1x32 port map (reg_in=>PC_2_In, reg_out=>PC_2,
           CLK=>CLK, RESET=>RESET);
Reg_IR : reg1x32 port map (reg_in=>IR_In, reg_out=>IR, CLK=>CLK,
         RESET=>RESET);

Reg_Bus_A : reg1x32 port map (reg_in=>Bus_A_In, reg_out=>Bus_A,
            CLK=>CLK, RESET=>RESET);
Reg_Bus_B : reg1x32 port map (reg_in=>Bus_B_In, reg_out=>Bus_B,
            CLK=>CLK, RESET=>RESET);

Reg_F : reg1x32 port map (reg_in=>F_In, reg_out=>F, CLK=>CLK,
        RESET=>RESET);
Reg_Data : reg1x32 port map (reg_in=>DataTmpIn, reg_out=>DataReg,
           CLK=>CLK, RESET=>RESET);

Reg_JA : reg1x26 port map (reg_in=>IR(25 downto 0), reg_out=>JA,
         CLK=>CLK, RESET=>RESET);

Reg_LD_1 : reg1x1 port map (reg_in=>LD_In_1, reg_out=>LD_Out_1,
           CLK=>CLK, RESET=>RESET);
Reg_DA_1 : reg1x5 port map (reg_in=>DA_In_1, reg_out=>DA_Out_1,
           CLK=>CLK, RESET=>RESET);
Reg_MD_1 : reg1x1 port map (reg_in=>MD_In_1, reg_out=>MD_Out_1,
           CLK=>CLK, RESET=>RESET);
```

```
        Reg_BS : reg1x3 port map (reg_in=>BS_In, reg_out=>BS, CLK=>CLK,
                 RESET=>RESET);
        Reg_MW : reg1x1 port map (reg_in=>MW_In, reg_out=>MW, CLK=>CLK,
                 RESET=>RESET);
        Reg_LS : reg1x2 port map (reg_in=>LS_In, reg_out=>LS, CLK=>CLK,
                 RESET=>RESET);
        Reg_FS : reg1x4 port map (reg_in=>FS_In, reg_out=>FS, CLK=>CLK,
                 RESET=>RESET);

        Reg_LD_2 : reg1x1 port map (reg_in=>LD_In_2, reg_out=>LD,
                   CLK=>CLK, RESET=>RESET);
        Reg_DA_2 : reg1x5 port map (reg_in=>DA_In_2, reg_out=>DA,
                   CLK=>CLK, RESET=>RESET);
        Reg_MD_2 : reg1x1 port map (reg_in=>MD_In_2, reg_out=>MD,
                   CLK=>CLK, RESET=>RESET);

end Behavioral;
```

Simbol :



RISC Tanpa Pipeline

```
Nama File   :     risc_unpipeline.vhd
Hirarki     :     # RISC_unpipeline
                  |
                  +-> # Function_Unit
                  |   |
                  |   +-> # ALU
                  |   |   |
                  |   |   +-> # addsub32_lib_par
                  |   |   |   |
                  |   |   |   +-> # adder32_lib
                  |   |   |
                  |   |   +-> # LogicUnit32
                  |   |
                  |   +-> # BarrelShift32_2
                  |   |
                  |   +-> # LUI
                  |
                  +-> # register_file_gab
                  |   |
                  |   +-> # const_unit
                  |   |
                  |   +-> # register_file_DPRAM
                  |
                  +-> # branch_ctrl
                  |
                  +-> # Data_Forwarding
                  |
                  +-> # ID
                  |
                  +-> # reg1x32
                  |
                  +-> # BUFE32
Program     :
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity RISC is
    Port ( Inst : in std_logic_vector(31 downto 0);
           Inst_Add : out std_logic_vector(31 downto 0);

           BSel : out std_logic_vector(3 downto 0);
           Data : inout std_logic_vector(31 downto 0);
           Data_Add : out std_logic_vector(31 downto 0);

           IntREQ : in std_logic;
           IntACK : out std_logic;
           IntADD : in std_logic_vector(2 downto 0);

           CLK : in std_logic;
           RESET : in std_logic;

           MW : inout std_logic;
```

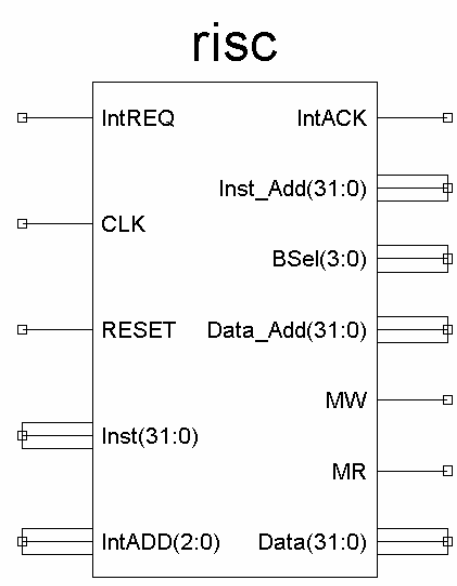
```
           MR : inout std_logic);
end RISC;

architecture Behavioral of RISC is

      component ID
            Port (IR_Conv : in std_logic_vector(20 downto 0);
                  AA, BA, DA : out std_logic_vector(4 downto 0);
                  FS : out std_logic_vector(3 downto 0);
                  BS : out std_logic_vector(2 downto 0);
                  LS : out std_logic_vector(1 downto 0);
                  MA, MB, MD, LD, MW, CS : out std_logic);
      end component;

      component Register_File_Gab
            Port (D_data, PC_1 : in std_logic_vector(31 downto 0);
                  Imm : in std_logic_vector(15 downto 0);
                  AA, BA, DA : in std_logic_vector(4 downto 0);
                  MA, MB, CS, LD, CLK : in std_logic;
                  Bus_A, Bus_B : out std_logic_vector(31 downto 0));
      end component;

      component Branch_Ctrl
            port (BS : in std_logic_vector(2 downto 0);
                  Zero, Cout, N, V : in std_logic;
                  MC : out std_logic_vector(1 downto 0));
      end component;

      component Function_Unit
            Port (A, B : in std_logic_vector(31 downto 0);
                  FS : in std_logic_vector(3 downto 0);
                  F : out std_logic_vector(31 downto 0);
                  Cout, Zero, N, V : out std_logic);
      end component;

      component Interrupt_Ctrl
            port( CLK : in std_logic;
                  IntREQ, MW, MR : in std_logic;
                  IntEnOP : in std_logic_vector(5 downto 0);
                  Br1, Br2 : in std_logic_vector(2 downto 0);
                  Int : out std_logic);
      end component;

      component reg1x32
            Port (reg_in : in std_logic_vector(31 downto 0);
                  reg_out : out std_logic_vector(31 downto 0);
                  CLK : in std_logic;
                  RESET : in std_logic);
      end component;

      component BUFE32
            port( E : in std_logic;
                  I : in std_logic_vector(31 downto 0);
                  O : out std_logic_vector(31 downto 0)
            );
      end component;
```

```
	--Branch & Jump
	signal MC : std_logic_vector(1 downto 0);
	signal JA : std_logic_vector(25 downto 0);
	signal PC_In, JA_add, JRA : std_logic_vector(31 downto 0);

	--ID
	signal AA, BA : std_logic_vector(4 downto 0);
	signal MA, MB, CS : std_logic;

	--IF
	signal PC : std_logic_vector(31 downto 0);
	signal PC_Inc : std_logic_vector(31 downto 0);

	--DO
	signal IR : std_logic_vector(31 downto 0);

	--EX
	signal Bus_A, Bus_B, PC_2, BrA : std_logic_vector(31 downto 0);
	signal FS : std_logic_vector(3 downto 0);
	signal BS : std_logic_vector(2 downto 0);
	signal Zero, Cout, N, V : std_logic;

	--Memory Control In/Out
	signal DataTmpOut : std_logic_vector(31 downto 0);
	signal LS : std_logic_vector(1 downto 0);

	--WB
	signal F, Bus_D, DataReg : std_logic_vector(31 downto 0);
	signal DA : std_logic_vector(4 downto 0);
	signal MD, LD : std_logic;

	--Interrupt
	signal IntADD_1,PC_In_1,PC_1_In_1:std_logic_vector(31 downto 0);
	signal Int : std_logic;

begin

	--Branch & Jump

	Branch_Ctrl_PM : Branch_Ctrl port map (BS=>BS, Zero=>Zero,
	                 Cout=>Cout, N=>N, V=>V, MC=>MC);

	MUX_C : process(MC, PC_Inc, BrA, JRA, JA_add) begin
	      if    (MC = "00") then PC_In_1 <= PC_Inc;
	      elsif (MC = "01") then PC_In_1 <= BrA;
	      elsif (MC = "10") then PC_In_1 <= JA_add;
	      else                   PC_In_1 <= JRA;
	      end if;
	end process;

	--Interrupt

	IntADD_1 <= X"0000000" & IntADD & '0';

	MUX_I_PC : process(Int, IntADD_1, PC_In_1) begin
	      if(Int = '1') then
	            PC_In <= IntADD_1;
```

```
      else
            PC_In <= PC_In_1;
      end if;
end process;

MUX_I_PC_1 : process(Int, PC, PC_1_In_1) begin
      if(Int = '1') then
            PC_2 <= PC;
      else
            PC_2 <= PC_1_In_1;
      end if;
end process;

MUX_IR : process(Int, Inst) begin
      if(Int = '0') then
            IR <= Inst;

      else
            IR <= X"57C00000";
      end if;
end process;

--Interrupt Control

Interrupt_Control : Interrupt_Ctrl port map (CLK=>CLK,
                    IntREQ=>IntREQ, MW=>MW, MR=>MR,
                    IntEnOP=>Inst(31 downto 26), Br1=>BS,
                    Br2=>BS, Int=>Int);

IntACK <= Int;

--IF

Inst_Add <= PC;
PC_Inc <= PC + "1";
PC_1_In_1 <= PC_Inc;

--DO

Instruction_Decoder : ID port map (IR_Conv=>IR(31 downto 11),
            LD=>LD, DA=>DA, MD=>MD, BS=>BS, MW=>MW, LS=>LS,
            FS=>FS, MA=>MA, MB=>MB, AA=>AA, BA=>BA, CS=>CS);

Register_File : Register_File_Gab port map (D_data=>Bus_D,
            PC_1=>PC_2, Imm=>IR(15 downto 0), AA=>AA, BA=>BA,
            DA=>DA, MA=>MA, MB=>MB, CS=>CS, LD=>LD, CLK=>CLK,
            Bus_A=>Bus_A, Bus_B=>Bus_B);

--EX

FunctionUnit : Function_Unit port map (A=>Bus_A, B=>Bus_B,
               FS=>FS, F=>F, Cout=>Cout, Zero=>Zero, N=>N, V=>V);

JRA <= Bus_B;

BrA <= (PC_2) + (JA(25) & JA(25) & JA(25) & JA(25) & JA(25) &
        JA(25) & JA(25) & JA(25) & JA(25) & JA(25) & JA(25) &
```

```
        JA(25) & JA(25) & JA(25) & JA(25) & JA(25) &
        JA(25 downto 21) & JA(10 downto 0));

JA_add <= (PC_2) + (JA(25) & JA(25) & JA(25) & JA(25) & JA(25) &
                    JA(25) & JA(25 downto 0));

TS_Buffer : BUFE32 port map (E=>MW, I=>DataTmpOut, O=>Data);

Data_Add(31 downto 0) <= Bus_A(31 downto 0);


--Memory Control Out (Instruksi Store)

MCO_sel : process(Bus_A, LS) begin
if(Bus_A(1) & Bus_A(0) & LS = "0000")       then BSel <= "0001";
   elsif(Bus_A(1) & Bus_A(0) & LS = "0001") then BSel <= "0011";
   elsif(Bus_A(1) & Bus_A(0) & LS = "0010") then BSel <= "1111";
   elsif(Bus_A(1) & Bus_A(0) & LS = "0100") then BSel <= "0010";
   elsif(Bus_A(1) & Bus_A(0) & LS = "1000") then BSel <= "0100";
   elsif(Bus_A(1) & Bus_A(0) & LS = "1001") then BSel <= "1100";
   elsif(Bus_A(1) & Bus_A(0) & LS = "1100") then BSel <= "1000";
   else                                          BSel <= "0000";
end if;
end process;

MCO_MUX : process(Bus_A, Bus_B) begin
      if   ((Bus_A(1) = '1') AND (Bus_A(0) = '0')) then
             DataTmpOut(31 downto 24) <= Bus_B(15 downto 8);
      elsif((Bus_A(1) = '1') AND (Bus_A(0) = '1')) then
             DataTmpOut(31 downto 24) <= Bus_B(7 downto 0);
      else   DataTmpOut(31 downto 24) <= Bus_B(31 downto 24);
      end if;

      if(Bus_A(1) = '0') then
            DataTmpOut(23 downto 16) <= Bus_B(23 downto 16);
      else  DataTmpOut(23 downto 16) <= Bus_B(7 downto 0);
      end if;

      if(Bus_A(0) = '0') then
            DataTmpOut(15 downto 8) <= Bus_B(15 downto 8);
      else  DataTmpOut(15 downto 8) <= Bus_B(7 downto 0);
      end if;
end process;

DataTmpOut(7 downto 0) <= Bus_B(7 downto 0);

--Memory Control In (Instruksi Load)

MCI : process(Bus_A, LS, Data) begin
      if   (Bus_A(1) & Bus_A(0) & LS = "0000") then
            DataReg <= X"000000" & Data(7 downto 0);
      elsif(Bus_A(1) & Bus_A(0) & LS = "0001") then ]
            DataReg <= X"0000" & Data(15 downto 0);
      elsif(Bus_A(1) & Bus_A(0) & LS = "0100") then
            DataReg <= X"000000" & Data(15 downto 8);
      elsif(Bus_A(1) & Bus_A(0) & LS = "1000") then
            DataReg <= X"000000" & Data(23 downto 16);
```

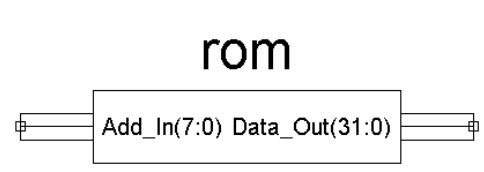
```
            elsif(Bus_A(1) & Bus_A(0) & LS = "1001") then
                  DataReg <= X"0000" & Data(31 downto 16);
            elsif(Bus_A(1) & Bus_A(0) & LS = "1100") then
                  DataReg <= X"000000" & Data(31 downto 24);
            else  DataReg <= Data(31 downto 0);
            end if;
      end process;


      --WB

      MUX_D : process(MD, F, DataReg) begin
            if (MD = '0') then Bus_D <= F;
            else               Bus_D <= DataReg;
            end if;
      end process;

      --Register Pipeline

      Reg_PC : reg1x32 port map (reg_in=>PC_In, reg_out=>PC, CLK=>CLK,
               RESET=>RESET);

      JA <= IR(25 downto 0);

end Behavioral;

Simbol      :
```



Listing Komponen uC

ROM (Read Only Memory)

```
Nama File   :     rom.vhd
Hirarki     :     # rom
Fungsi      :     Memori tempat penyimpanan program
Program     :

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity ROM is
    Port ( Add_In : in std_logic_vector(7 downto 0);
           Data_Out : out std_logic_vector(31 downto 0));
end ROM;

architecture Behavioral of ROM is

begin

   counter : process(Add_In) begin
      case Add_In is
         when "00000000" =>
            Data_Out <= B"00001000001000000001001000110100";
         when "00000001" =>
            Data_Out <= B"00000100100000001001101010111100";
         when "00000010" =>
            Data_Out <= B"01000000010000010000000000010000";
         when "00000011" =>
            Data_Out <= B"00010100101001001001101010101100";
         when "00000100" =>
            Data_Out <= B"00001000111000001101111011110000";
         when others     =>
            Data_Out <= B"00000000000000000000000000000000";
      end case;
   end process;

end Behavioral;

Simbol      :
```



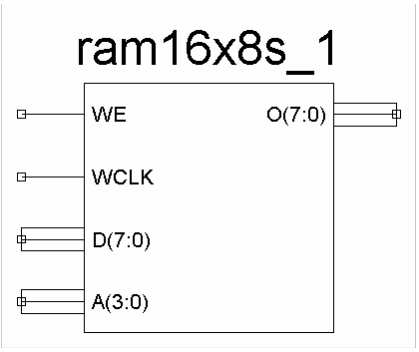
RAM16x8S_1 (16-deep by 8-wide static synchronous RAM with negative-edge clock)

```
Nama File   :     RAM16x8S_1.vhd
Hirarki     :     # RAM16x8S_1
Fungsi      :     Memori tempat penyimpanan data
```

Program :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity RAM16x8S_1 is
      port( WE, WCLK : in std_logic;
            D : in std_logic_vector(7 downto 0);
            A : in std_logic_vector(3 downto 0);
            O : out std_logic_vector(7 downto 0)
      );
end RAM16x8S_1;

architecture Behavioral of RAM16x8S_1 is

      component RAM16x1S_1
            port( WE : in std_logic;
                  D : in std_logic;
                  A3 : in std_logic;
                  A2 : in std_logic;
                  A1 : in std_logic;
                  A0 : in std_logic;
                  WCLK : in std_logic;
                  O : out std_logic
            );
      end component;

begin

      bit1 : RAM16x1S_1 port map (WE=>WE, D=>D(0), A3=>A(3), A2=>A(2),
             A1=>A(1), A0=>A(0), WCLK=>WCLK, O=>O(0));
      bit2 : RAM16x1S_1 port map (WE=>WE, D=>D(1), A3=>A(3), A2=>A(2),
             A1=>A(1), A0=>A(0), WCLK=>WCLK, O=>O(1));
      bit3 : RAM16x1S_1 port map (WE=>WE, D=>D(2), A3=>A(3), A2=>A(2),
             A1=>A(1), A0=>A(0), WCLK=>WCLK, O=>O(2));
      bit4 : RAM16x1S_1 port map (WE=>WE, D=>D(3), A3=>A(3), A2=>A(2),
             A1=>A(1), A0=>A(0), WCLK=>WCLK, O=>O(3));
      bit5 : RAM16x1S_1 port map (WE=>WE, D=>D(4), A3=>A(3), A2=>A(2),
             A1=>A(1), A0=>A(0), WCLK=>WCLK, O=>O(4));
      bit6 : RAM16x1S_1 port map (WE=>WE, D=>D(5), A3=>A(3), A2=>A(2),
             A1=>A(1), A0=>A(0), WCLK=>WCLK, O=>O(5));
      bit7 : RAM16x1S_1 port map (WE=>WE, D=>D(6), A3=>A(3), A2=>A(2),
             A1=>A(1), A0=>A(0), WCLK=>WCLK, O=>O(6));
      bit8 : RAM16x1S_1 port map (WE=>WE, D=>D(7), A3=>A(3), A2=>A(2),
             A1=>A(1), A0=>A(0), WCLK=>WCLK, O=>O(7));

end Behavioral;
```

Simbol :



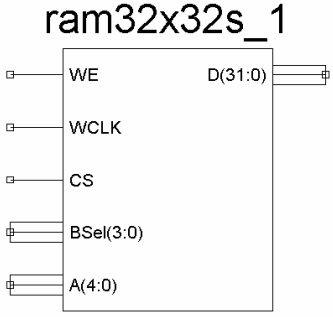
RAM32x32S_1 (32-deep by 32-wide static synchronous RAM with negative-edge clock)

Nama File : RAM32x32S_1.vhd
Hirarki : # RAM32x32S_1
|
+-> # RAM16x8S_1
Fungsi : Memori tempat penyimpanan data
Program :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity RAM32x32S_1 is
      port( WE, WCLK, CS : in std_logic;
            BSel : in std_logic_vector(3 downto 0); -- BSel=Byte Select
            A : in std_logic_vector(4 downto 0);
            D : inout std_logic_vector(31 downto 0)
      );
end RAM32x32S_1;

architecture Behavioral of RAM32x32S_1 is

      component RAM16x8S_1
            port( WE, WCLK : in std_logic;
                  D : in std_logic_vector(7 downto 0);
                  A : in std_logic_vector(3 downto 0);
                  O : out std_logic_vector(7 downto 0)
            );
      end component;

      component BUFE32
            port( E : in std_logic;
                  I : in std_logic_vector(31 downto 0);
                  O : out std_logic_vector(31 downto 0)
            );
      end component;

      signal WE1_0, WE1_1, WE1_2, WE1_3, WE2_0, WE2_1, WE2_2, WE2_3 :
std_logic;
      signal WE_tmp : std_logic;
```

```
      signal Dout, Dout1, Dout2 : std_logic_vector(31 downto 0);

begin

      WE1_0 <= WE AND (NOT CS) AND (NOT A(4)) AND BSel(0);
      WE1_1 <= WE AND (NOT CS) AND (NOT A(4)) AND BSel(1);
      WE1_2 <= WE AND (NOT CS) AND (NOT A(4)) AND BSel(2);
      WE1_3 <= WE AND (NOT CS) AND (NOT A(4)) AND BSel(3);
      WE2_0 <= WE AND (NOT CS) AND A(4) AND BSel(0);
      WE2_1 <= WE AND (NOT CS) AND A(4) AND BSel(1);
      WE2_2 <= WE AND (NOT CS) AND A(4) AND BSel(2);
      WE2_3 <= WE AND (NOT CS) AND A(4) AND BSel(3);

      RAM16_1x8_0 : RAM16x8S_1 port map (WE=>WE1_0, WCLK=>WCLK,
         D=>D( 7 downto 0),  A=>A(3 downto 0), O=>Dout1( 7 downto 0));
      RAM16_2x8_0 : RAM16x8S_1 port map (WE=>WE2_0, WCLK=>WCLK,
         D=>D( 7 downto 0),  A=>A(3 downto 0), O=>Dout2( 7 downto 0));
      RAM16_1x8_1 : RAM16x8S_1 port map (WE=>WE1_1, WCLK=>WCLK,
         D=>D(15 downto 8),  A=>A(3 downto 0), O=>Dout1(15 downto 8));
      RAM16_2x8_1 : RAM16x8S_1 port map (WE=>WE2_1, WCLK=>WCLK,
         D=>D(15 downto 8),  A=>A(3 downto 0), O=>Dout2(15 downto 8));
      RAM16_1x8_2 : RAM16x8S_1 port map (WE=>WE1_2, WCLK=>WCLK,
         D=>D(23 downto 16), A=>A(3 downto 0), O=>Dout1(23 downto 16));
      RAM16_2x8_2 : RAM16x8S_1 port map (WE=>WE2_2, WCLK=>WCLK,
            D=>D(23 downto 16), A=>A(3 downto 0), O=>Dout2(23 downto
      16));
      RAM16_1x8_3 : RAM16x8S_1 port map (WE=>WE1_3, WCLK=>WCLK,
         D=>D(31 downto 24), A=>A(3 downto 0), O=>Dout1(31 downto 24));
      RAM16_2x8_3 : RAM16x8S_1 port map (WE=>WE2_3, WCLK=>WCLK,
         D=>D(31 downto 24), A=>A(3 downto 0), O=>Dout2(31 downto 24));

      process(A, Dout1, Dout2) begin
            if(A(4) = '0') then
                  Dout <= Dout1;
            else
                  Dout <= Dout2;
            end if;
      end process;

      WE_tmp <= (NOT WE) AND (NOT CS);
      TS_Buffer : BUFE32 port map (E=>WE_tmp, I=>Dout, O=>D);

end Behavioral;
```

Simbol :



uC (Micro Controler)

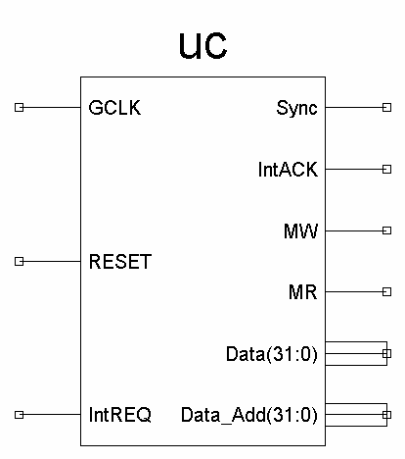
Nama File : uC.vhd
Hirarki : # uC

```
|
+-> # RISC
|   |
|   +-> # Function_Unit
|   |   |
|   |   +-> # ALU
|   |   |   |
|   |   |   +-> # addsub32_lib_par
|   |   |   |   |
|   |   |   |   +-> # adder32_lib
|   |   |   |
|   |   |   +-> # LogicUnit32
|   |   |
|   |   +-> # BarrelShift32_2
|   |   |
|   |   +-> # LUI
|   |
|   +-> # register_file_gab_DF
|   |   |
|   |   +-> # const_unit
|   |   |
|   |   +-> # register_file_DPRAM
|   |
|   +-> # branch_ctrl
|   |
|   +-> # Data_Forwarding
|   |
|   +-> # ID
|   |
|   +-> # reg1x1
|   |
|   +-> # reg1x2
|   |
|   +-> # reg1x3
|   |
|   +-> # reg1x4
|   |
|   +-> # reg1x5
|   |
|   +-> # reg1x26
|   |
|   +-> # reg1x32
|   |
|   +-> # BUFE32
|
+-> # rom
|
+-> # RAM32x32S_1
|   |
|   +-> # RAM16x8S_1
|
+-> # BUFE3
```

```
Program     :

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity uc is
    Port (  Data : inout std_logic_vector(31 downto 0);
            Data_Add : inout std_logic_vector(31 downto 0);

            GCLK, RESET : in std_logic;

            IntREQ : in std_logic;
            IntACK : inout std_logic;

            Sync : out std_logic;
            MW : inout std_logic;
            MR : inout std_logic);
end uc;

architecture Behavioral of uc is

      component RISC
          Port ( Inst : in std_logic_vector(31 downto 0);
                 Inst_Add : out std_logic_vector(31 downto 0);

                 BSel : out std_logic_vector(3 downto 0);
                 Data : inout std_logic_vector(31 downto 0);
                 Data_Add : out std_logic_vector(31 downto 0);

                 CLK, RESET, IntREQ : in std_logic;
                 IntACK : inout std_logic;
                 IntADD : in std_logic_vector(2 downto 0);

                 MW : inout std_logic;
                 MR : inout std_logic);
      end component;

      component ROM
          Port ( Add_In : in std_logic_vector(7 downto 0);
                 Data_Out : out std_logic_vector(31 downto 0));
      end component;

      component RAM32x32S_1
            port( WE, WCLK, CS : in std_logic;
                  BSel : in std_logic_vector(3 downto 0);
                  A : in std_logic_vector(4 downto 0);
                  D : inout std_logic_vector(31 downto 0)
            );
      end component;

      component BUFE3
            port( E : in std_logic;
                  I : in std_logic_vector(2 downto 0);
                  O : out std_logic_vector(2 downto 0)
            );
```

```
	end component;

	signal Inst : std_logic_vector(31 downto 0);
	signal Inst_Add : std_logic_vector(31 downto 0);

	signal Data_Add_RISC : std_logic_vector(31 downto 0);

	signal CLK : std_logic;
	signal CTR : std_logic_vector(1 downto 0);

	signal Mem_Sel, IO_Sel : std_logic;
	signal BSel : std_logic_vector(3 downto 0);

	signal IntADD : std_logic_vector(2 downto 0);

	signal MRW : std_logic;

begin

	-- Clock divide by 2
	two_bit_counter : process(GCLK, CTR, RESET) begin
		if(RESET = '1') then
			CTR <= "00";
		elsif(GCLK'EVENT AND GCLK = '1') then
			CTR <= CTR + 1;
		end if;
	end process;

	CLK <= CTR(1);
	Sync <= CTR(1);

	RISC_uP : RISC port map (IntREQ=>IntREQ, IntACK=>IntACK,
			Inst=>Inst, Inst_Add=>Inst_Add, BSel=>BSel, Data=>Data,
			Data_Add=>Data_Add_RISC, CLK=>CLK, RESET=>RESET,
			IntADD=>IntADD, MW=>MW, MR=>MR);

	MRW <= MR OR MW;
	TBuf_IntADD : BUFE3 port map (E=>MRW,
		I=>Data_Add_RISC(2 downto 0), O=>Data_Add(2 downto 0));
	Data_Add(31 downto 3) <= Data_Add_RISC(31 downto 3);
	IntADD <= Data_Add(2 downto 0);

	ROM_PM : ROM port map (Add_In=>Inst_Add(7 downto 0),
			Data_Out=>Inst);

	SRAM : RAM32x32S_1 port map (WE=>MW, WCLK=>CLK,
		CS=>Data_Add_RISC(7), BSel=>BSel,
		A=>Data_Add_RISC(6 downto 2), D=>Data);

end Behavioral;
```

Simbol :



Listing Komponen Pengontrol *Input*/*Output*

Buffer 3 dan 8

```
Nama File   :     Bufe3.vhd
Hirarki     :     # BUFE3
Fungsi      :     Buffer bus alamat
Program     :

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity BUFE3 is
      port( E : in std_logic;
            I : in std_logic_vector(2 downto 0);
            O : out std_logic_vector(2 downto 0)
      );
end BUFE3;

architecture Behavioral of BUFE3 is

      component BUFE
            port( E : in std_logic;
                  I : in std_logic;
                  O : out std_logic
            );
      end component;

begin

      bit00 : BUFE port map (E=>E, I=>I(0),  O=>O(0));
      bit01 : BUFE port map (E=>E, I=>I(1),  O=>O(1));
      bit02 : BUFE port map (E=>E, I=>I(2),  O=>O(2));

end Behavioral;

Simbol      :
```

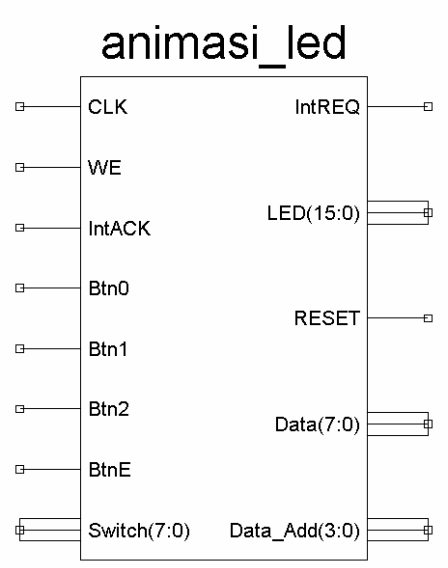
Animasi LED

```
Nama File   :     Animasi_LED.vhd
Hirarki     :     # Animasi_LED
                  |
                  +-> # BUFE3
                  |
                  +-> # BUFE8
```

```
Fungsi      :     Pengontrol input/output untuk melakukan pengujian
Program     :

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Animasi_LED is
      port( Data : inout std_logic_vector(7 downto 0);
            Data_Add : inout std_logic_vector(3 downto 0);
            CLK, WE : in std_logic;
            IntACK : in std_logic;
            RESET : inout std_logic;
            IntREQ : out std_logic;

            LED : out std_logic_vector(15 downto 0);
            Btn0, Btn1, Btn2, BtnE : in std_logic;
            Switch : in std_logic_vector(7 downto 0)
      );
end Animasi_LED;

architecture Behavioral of Animasi_LED is

      component BUFE8
            port( E : in std_logic;
                  I : in std_logic_vector(7 downto 0);
                  O : out std_logic_vector(7 downto 0)
            );
      end component;

      component BUFE3
            port( E : in std_logic;
                  I : in std_logic_vector(2 downto 0);
                  O : out std_logic_vector(2 downto 0)
            );
      end component;

      signal WE_tmp : std_logic;
      signal IntADD : std_logic_vector(2 downto 0);
      signal ADD : std_logic_vector(3 downto 0);
      signal IO_Data_Out, IO_Data_In : std_logic_vector(7 downto 0);

begin

      -- TBuf Data

      WE_tmp <= (NOT WE) AND Data_Add(3);
      TBuf_Data : BUFE8 port map (E=>WE_tmp, I=>IO_Data_Out, O=>Data);
      IO_Data_In <= Data;

      Switch_Input : process(Switch, Btn2) begin
            if (Btn2'EVENT AND Btn2 = '1') then
                  IO_Data_Out <= Switch;
            end if;
      end process;
```

```
	-- TBuf Address

	TBuf_IntADD : BUFE3 port map (E=>IntACK, I=>IntADD,
	              O=>Data_Add(2 downto 0));
	ADD <= Data_Add;

	-- LED Control
	LED_Ctrl : process(WE, ADD, CLK, IO_Data_In, RESET) begin
		if(WE = '1' AND ADD = "1000") then
			if(CLK'EVENT AND CLK = '0') then
				LED(7 downto 0) <= NOT(IO_Data_In);
			end if;
		elsif(WE = '1' AND ADD = "1001") then
			if(CLK'EVENT AND CLK = '0') then
				LED(15 downto 8) <= NOT(IO_Data_In);
			end if;
		end if;

		if(RESET = '1') then
			LED(15 downto 0) <= (others => '1');
		end if;
	end process;

	-- Intertupt Control
	Interrupt_Request : process(Btn1, Btn2) begin
		if((Btn1 = '1') OR (Btn2 = '1')) then
			IntREQ <= '1';
		else
			IntREQ <= '0';
		end if;
	end process;

	Int_Ctrl : process(Btn1, Btn2, IntACK) begin
		if((Btn1 = '1') AND (IntACK = '1')) then
			IntADD <= "001";
		elsif((Btn2 = '1') AND (IntACK = '1')) then
			IntADD <= "010";
		else
			IntADD <= "000";
		end if;
	end process;

	RESET <= Btn0 AND BtnE;

end Behavioral;
```

Simbol :


animasi_led
CLK
WE
IntACK
Btn0
Btn1
Btn2
BtnE
Switch(7:0)
IntREQ
LED(15:0)
RESET
Data(7:0)
Data_Add(3:0)

File Constrain

FPGA

Nama File : uC.ucf

```
NET "Data<0>" LOC = "P41";
NET "Data<1>" LOC = "P37";
NET "Data<2>" LOC = "P43";
NET "Data<3>" LOC = "P42";
NET "Data<4>" LOC = "P45";
NET "Data<5>" LOC = "P44";
NET "Data<6>" LOC = "P47";
NET "Data<7>" LOC = "P46";
NET "Data<8>" LOC = "P181";
NET "Data<9>" LOC = "P180";
NET "Data<10>" LOC = "P179";
NET "Data<11>" LOC = "P178";
NET "Data<12>" LOC = "P176";
NET "Data<13>" LOC = "P175";
NET "Data<14>" LOC = "P174";
NET "Data<15>" LOC = "P173";
NET "Data<16>" LOC = "P172";
NET "Data<17>" LOC = "P168";
NET "Data<18>" LOC = "P167";
NET "Data<19>" LOC = "P166";
NET "Data<20>" LOC = "P165";
NET "Data<21>" LOC = "P164";
NET "Data<22>" LOC = "P163";
NET "Data<23>" LOC = "P162";
NET "Data<24>" LOC = "P161";
NET "Data<25>" LOC = "P160";
NET "Data<26>" LOC = "P154";
NET "Data<27>" LOC = "P152";
NET "Data<28>" LOC = "P151";
NET "Data<29>" LOC = "P150";
NET "Data<30>" LOC = "P149";
NET "Data<31>" LOC = "P148";
NET "Data_Add<0>" LOC = "P59";
NET "data_add<1>" LOC = "P62";
NET "data_add<2>" LOC = "P61";
NET "data_add<3>" LOC = "P138";
NET "Data_Add<4>" LOC = "P141";
NET "Data_Add<5>" LOC = "P140";
NET "Data_Add<6>" LOC = "P139";
NET "Data_Add<7>" LOC = "P67";
NET "Data_Add<8>" LOC = "P136";
NET "Data_Add<9>" LOC = "P135";
NET "Data_Add<10>" LOC = "P134";
NET "Data_Add<11>" LOC = "P133";
NET "Data_Add<12>" LOC = "P132";
NET "Data_Add<13>" LOC = "P129";
NET "Data_Add<14>" LOC = "P127";
NET "Data_Add<15>" LOC = "P125";
NET "Data_Add<16>" LOC = "P126";
NET "Data_Add<17>" LOC = "P122";
```

```
NET "Data_Add<18>" LOC = "P123";
NET "Data_Add<19>" LOC = "P120";
NET "Data_Add<20>" LOC = "P121";
NET "Data_Add<21>" LOC = "P115";
NET "Data_Add<22>" LOC = "P119";
NET "Data_Add<23>" LOC = "P113";
NET "Data_Add<24>" LOC = "P114";
NET "Data_Add<25>" LOC = "P111";
NET "Data_Add<26>" LOC = "P112";
NET "Data_Add<27>" LOC = "P109";
NET "Data_Add<28>" LOC = "P110";
NET "Data_Add<29>" LOC = "P102";
NET "data_add<30>" LOC = "P188";
NET "data_add<31>" LOC = "P187";
NET "reset" LOC = "P63";
NET "gclk" LOC = "P80";
NET "intack" LOC = "P69";
NET "sync" LOC = "P49";
NET "MW" LOC = "P48";
NET "intreq" LOC = "P58";
```

CPLD

```
Nama File : Animasil_LED.ucf

NET "btn0" LOC = "P84";
NET "btn1" LOC = "P47";
NET "btn2" LOC = "P66";
NET "btne" LOC = "P54";
NET "clk" LOC = "P13";
NET "data<0>" LOC = "P11";
NET "data<1>" LOC = "P7";
NET "data<2>" LOC = "P6";
NET "data<3>" LOC = "P5";
NET "data<4>" LOC = "P4";
NET "data<5>" LOC = "P3";
NET "data<6>" LOC = "P2";
NET "data<7>" LOC = "P1";
NET "data_add<0>" LOC = "P83";
NET "data_add<1>" LOC = "P81";
NET "data_add<2>" LOC = "P80";
NET "data_add<3>" LOC = "P79";
NET "intreq" LOC = "P76";
NET "led<0>" LOC = "P82";
NET "led<1>" LOC = "P12";
NET "led<2>" LOC = "P14";
NET "led<3>" LOC = "P15";
NET "led<4>" LOC = "P17";
NET "led<5>" LOC = "P18";
NET "led<6>" LOC = "P19";
NET "led<7>" LOC = "P20";
NET "led<8>" LOC = "P63";
NET "led<9>" LOC = "P69";
NET "led<10>" LOC = "P67";
NET "led<11>" LOC = "P68";
```

```
NET "led<12>" LOC = "P70";
NET "led<13>" LOC = "P71";
NET "led<14>" LOC = "P72";
NET "led<15>" LOC = "P74";
NET "reset" LOC = "P77";
NET "we" LOC = "P9";
NET "intack" LOC = "P75";
NET "switch<0>" LOC = "P34";
NET "switch<1>" LOC = "P40";
NET "switch<2>" LOC = "P39";
NET "switch<3>" LOC = "P41";
NET "switch<4>" LOC = "P43";
NET "switch<5>" LOC = "P45";
NET "switch<6>" LOC = "P44";
NET "switch<7>" LOC = "P46";
```

Source Code Assembler

Nama File : asm.c

```
/*    File Name : asm0_81.c
 *    Function : Assembler
 *    Writen by : l411v
 *    Writen Date : November, 1st 2003
 *    Revision Date : July, 9th 2004
 *    Version : 0.81
 *    My Home Page : www.l411v.com
 *    Mail ME at : l411v@yahoo.com
 *    Limitation :
 *     1. Label started with 'under score' (_) and maximum 25
 *        character.
 *     2. Comment can't given.
 *     3. Write all instruction Up Case.
 *     4. Each instruction sparate by one ENTER (don't give blank
 *        line).
 *     5. TABULATION not allowed.
 *     6. Give one space after Coma (,).
 *     7. Immediate beginning with Sharp (#) and ending with 'h'
 *        character.
 *     8. Write Register with two digits (ex. R01, R09, R21).
 *     For example:
 *        _start:
 *        ADD R01, R02, R29
 *        SUB R03, R20, #AC34h
 *        JMP _start:
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

//###################################  GLOBAL VARIABLE DECLARATION  ###
/* OperandType  */
#define RD_RA_RB      0
#define RD_RA_IMM     1
#define RD_RA         2
#define RD_IMM        3
#define RD            4
#define RA_RB         5
#define RB            6
#define TARGET        7
#define RA_RB_IMM     8
#define NONE          9

/* Link and No Link instruction  */
#define NO_LINK       0
#define LINK          1

typedef unsigned short USHORT;



/* Instruction Definition  */
```

```
struct inst_def {
   char mnemonic[5];
   char opcode[7];
   USHORT operand_type;
   USHORT mnemonic_length;
   USHORT link;
   struct inst_def *next;
}*inst_head = NULL, *inst_tail, *inst_cur;

/* Label Definition  */
struct label_def {
   char label[25];  // name of label
   int label_add;   // label address
   struct label_def *next;
}*label_head = NULL, *label_tail, *label_cur;

/* Pointer to Assembly and Bits file  */
FILE *ASM_FILE, *BIT_FILE;

//##########################################  FUNCTION DECLARATION  ###
void HEXchar_to_BINchar(char HEX, char *BIN);
void regDEC_to_regBIN(char *reg_dec, char *reg_bin);
void print_program_information();
void argument_validation(int argc);
void inst_colect(char mnemonic[5], char opcode[7], \
   USHORT operand_type, USHORT mnemonic_length, USHORT link);
void instruction_definition();
void label_scanning();
void get_mnemonic(int cur_p, char *mnemonic);
int its_label(char *mnemonic, int *cur_p);
int get_inst_def(char *mnemonic);
void get_opcode(char *instruction, int *cur_p);
void cmp_label(char *label_get, int inst_cnt, char *add_hex);
void operand_decode(char *instruction, int *cur_p, int inst_cnt);
void cur_p_calibrate(int *cur_p);

//################################################  MAIN PROGRAM  ###
int  main(int argc, char *argv[]) {
   /* 'mnemonic' geted from user assembly program.  */
   char mnemonic[5];
   /* 'instruction' consist of 32 bit instruction.  */
   char instruction[32];

   /* Cursosr Pointer at user assembly file.  */
   int cur_p = 0;
   /* Counting a number of instruction. Used to know the location of \
    * a label. The value start from zero (0).  */
   int inst_cnt = 0;

   /* Line Feed  */
   char LINE_FEED[1] = {0xA};

   /* ASM_FILE : input file content of user assembly program (code).
    * BIT_FILE : output file content of 32 bits instruction. Copy \
    *            this content to 'rom.vhd' file and give the address \
    *            for each instruction.  */
```

```
   ASM_FILE = fopen(argv[1],"rb");
   BIT_FILE = fopen(argv[2],"wb");

   /* Print a information about the making of this assembler.  */
   print_program_information();

   /* Check for argument that transfered.  */
   argument_validation(argc);

   /* Definite all instruction that used.  */
   instruction_definition();

   /* Scan all label that apear in user assembly program.  */
   label_scanning();

   /* Looping to decode each instruction from user assembly \
    * program.  */
   do {
      /* Get a mnemonic (start of instruction) from user assembly \
       * program.  */
      get_mnemonic(cur_p, mnemonic);
      /* If a array of character that fetched is a label (not a \
       * mnemonic), continue to next instruction.  */
      if(its_label(mnemonic, &cur_p)) continue;
      /* Get instruction definition from mnemonic that fetched.  */
      if(!get_inst_def(mnemonic)) exit(1);
      /* Get opcode from the mnemonic.  */
      get_opcode(instruction, &cur_p);
      /* Decode all operand for each instruction that feached.  */
      operand_decode(instruction, &cur_p, inst_cnt);
      /* Write 32 instruction bits to Target FILE (BIT_FILE).  */
      fwrite(instruction, 32, 1, BIT_FILE);
      fwrite(LINE_FEED, 1, 1, BIT_FILE);   // New Line
      /* Cursor Pointer calibration to fetch next instruction.  */
      cur_p_calibrate(&cur_p);
      /* Increase the Instruction Counter.  */
      inst_cnt++;
   }while(!feof(ASM_FILE));

   /* Close FILE.  */
   fclose(ASM_FILE);
   fclose(BIT_FILE);
   printf("SUCCESS...Generated Complete !\n");
   exit(0);
}

//##########################################  FUNCTION DEFINITION  ###
void HEXchar_to_BINchar(char HEX, char *BIN) {
   switch(HEX) {
      case '0' : strcpy(BIN, "0000"); break;
      case '1' : strcpy(BIN, "0001"); break;
      case '2' : strcpy(BIN, "0010"); break;
      case '3' : strcpy(BIN, "0011"); break;
      case '4' : strcpy(BIN, "0100"); break;
      case '5' : strcpy(BIN, "0101"); break;
      case '6' : strcpy(BIN, "0110"); break;
      case '7' : strcpy(BIN, "0111"); break;
```

```
      case '8' : strcpy(BIN, "1000"); break;
      case '9' : strcpy(BIN, "1001"); break;
      case 'A' : strcpy(BIN, "1010"); break;
      case 'a' : strcpy(BIN, "1010"); break;
      case 'B' : strcpy(BIN, "1011"); break;
      case 'b' : strcpy(BIN, "1010"); break;
      case 'C' : strcpy(BIN, "1100"); break;
      case 'c' : strcpy(BIN, "1010"); break;
      case 'D' : strcpy(BIN, "1101"); break;
      case 'd' : strcpy(BIN, "1010"); break;
      case 'E' : strcpy(BIN, "1110"); break;
      case 'e' : strcpy(BIN, "1010"); break;
      case 'F' : strcpy(BIN, "1111"); break;
      case 'f' : strcpy(BIN, "1010"); break;
      default  : strcpy(BIN, "0000");
   }
}

void regDEC_to_regBIN(char *reg_dec, char *reg_bin) {
   int reg, reg_int[2], reg_mod;
   int i;
   /* 'reg_dec' allways like this: R12. Take the 12 then convert the \
    * '1' and '2' character to '1' and '2' integer (just subtract it \
    * with 30h). Put the result at 'reg_int' variable.  */
   reg_int[0] = reg_dec[1] - 0X30;
   reg_int[1] = reg_dec[2] - 0X30;
   /* Combine the 'reg_int[0]' and 'reg_int[1]' then put it at \
    * 'reg'.  */
   reg = (reg_int[0] * 10) + reg_int[1];
   /* Convert the 'reg' to biner character and put it on 'reg_bin'. \
    * Just loop four times because the maximum value of 'reg' is \
    * 31.  */
   for(i=4; i>=0; i--) {
      reg_mod = reg % 2;
      if(reg_mod == 0) reg_bin[i] = '0';
      else             reg_bin[i] = '1';
      reg /= 2;
   }
}

void print_program_information() {
   printf("*** Assembler ***\n");
   printf("Writen by     : l411v\n");
   printf("Writen Date   : November, 1st 2003\n");
   printf("Revision Date : July, 9th 2004\n");
   printf("Version       : 0.81\n");
   printf("My Home Page  : www.l411v.com\n");
   printf("Mail Me at    : l411v@yahoo.com\n...\n\n");
}

void argument_validation(int argc) {
   if(argc != 3) {
      printf("ERROR...Program Argument !\n");
      printf("Example : ASM [ASM_FILE] [BIT_FILE] <enter>\n");
      exit(1);
   }
}
```

```
void inst_colect(char mnemonic[5], char opcode[7], \
      USHORT operand_type, USHORT mnemonic_length, USHORT link) {
   /* Make a new data structure.  */
   inst_cur = (struct inst_def*) malloc(sizeof(struct inst_def));
   /* Copy the information that geted from Infromation Definition.  */
   strcpy(inst_cur->mnemonic, mnemonic);
   strcpy(inst_cur->opcode, opcode);
   inst_cur->operand_type = operand_type;
   inst_cur->mnemonic_length = mnemonic_length;
   inst_cur->link = link;

   /* Put the new data structure at the end of link list.  */
   if(inst_head == NULL) {
      inst_head = inst_tail = inst_cur;
      inst_tail->next = NULL;
   }
   else {
      inst_tail->next = inst_cur;
      inst_tail = inst_cur;
      inst_tail->next = NULL;
   }
}

void instruction_definition() {
   /*-------------------------------------------------
    *                                  MnemonicLength
    *         Mnemonic  OpCode  OperandType |  LinkInst.
    *------------V--------V----------V------V-----V------  */
   inst_colect("ADD" , "000000", RD_RA_RB , 3, NO_LINK);
   inst_colect("ADI" , "000001", RD_RA_IMM, 3, NO_LINK);
   inst_colect("ADIU", "000010", RD_RA_IMM, 4, NO_LINK);
   inst_colect("SUB" , "000011", RD_RA_RB , 3, NO_LINK);
   inst_colect("SBI" , "000100", RD_RA_IMM, 3, NO_LINK);
   inst_colect("SBIU", "000101", RD_RA_IMM, 4, NO_LINK);
   inst_colect("AND" , "000110", RD_RA_RB , 3, NO_LINK);
   inst_colect("ANDI", "000111", RD_RA_IMM, 4, NO_LINK);
   inst_colect("OR"  , "001000", RD_RA_RB , 2, NO_LINK);
   inst_colect("ORI" , "001001", RD_RA_IMM, 3, NO_LINK);
   inst_colect("XOR" , "001010", RD_RA_RB , 3, NO_LINK);
   inst_colect("XORI", "001011", RD_RA_IMM, 4, NO_LINK);
   inst_colect("NOR" , "001100", RD_RA_RB , 3, NO_LINK);
   inst_colect("NORI", "001101", RD_RA_IMM, 4, NO_LINK);
   inst_colect("SLR" , "001110", RD_RA_IMM, 3, NO_LINK);
   inst_colect("SLRV", "001111", RD_RA_RB , 4, NO_LINK);
   inst_colect("SLL" , "010000", RD_RA_IMM, 3, NO_LINK);
   inst_colect("SLLV", "010001", RD_RA_RB , 4, NO_LINK);
   inst_colect("SAR" , "010010", RD_RA_IMM, 3, NO_LINK);
   inst_colect("SARV", "010011", RD_RA_RB , 4, NO_LINK);
   inst_colect("LUI" , "010100", RD_IMM   , 3, NO_LINK);
   inst_colect("LA"  , "010101", RD       , 2, NO_LINK);
   inst_colect("SB"  , "010110", RA_RB    , 2, NO_LINK);
   inst_colect("SH"  , "010111", RA_RB    , 2, NO_LINK);
   inst_colect("SW"  , "011000", RA_RB    , 2, NO_LINK);
   inst_colect("LB"  , "011001", RD_RA    , 2, NO_LINK);
   inst_colect("LH"  , "011010", RD_RA    , 2, NO_LINK);
   inst_colect("LW"  , "011011", RD_RA    , 2, NO_LINK);
   inst_colect("SLT" , "011100", RD_RA_RB , 3, NO_LINK);
```

```
   inst_colect("SLTI", "011101", RD_RA_IMM, 4, NO_LINK);
   inst_colect("DI"  , "011110", NONE     , 2, NO_LINK);
   inst_colect("EI"  , "011111", NONE     , 2, NO_LINK);
   inst_colect("BE"  , "100000", RA_RB_IMM, 2, NO_LINK);
   inst_colect("BH"  , "100001", RA_RB_IMM, 2, NO_LINK);
   inst_colect("BHE" , "100010", RA_RB_IMM, 3, NO_LINK);
   inst_colect("BG"  , "100011", RA_RB_IMM, 2, NO_LINK);
   inst_colect("BGE" , "100100", RA_RB_IMM, 3, NO_LINK);
   inst_colect("JMP" , "100101", TARGET   , 3, NO_LINK);
   inst_colect("JL"  , "100110", TARGET   , 2, LINK);
   inst_colect("JR"  , "100111", RB       , 2, NO_LINK);
   inst_colect("JRL" , "101000", RB       , 3, LINK);
}

void label_scanning() {
   int cur_p = 0;
   int inst_cnt = 0;
   char inst_char;

   /* The labael must be start with 'under score' (_).  */
   do {
      inst_char = '\0';

      /* Get the next character.  */
      fseek(ASM_FILE, cur_p, SEEK_SET);
      if(feof(ASM_FILE)) exit(0);
      cur_p++;
      inst_char = fgetc(ASM_FILE);

      /* Its a next or first insturction.  */
      if(inst_char == 0xA || cur_p == 1) {
         if(cur_p == 1) cur_p -= 1;

       /* Get the character and compare it with under score '_' that
        * mean it is a LABEL.  */
         fseek(ASM_FILE, cur_p, SEEK_SET);
         inst_char = fgetc(ASM_FILE);

         if(cur_p == 0) cur_p += 1;

         if(inst_char == '_') {
            /* Alocate memory for new structure of label definition. */
            label_cur = (struct label_def*) malloc(sizeof(struct \
            label_def));
            fseek(ASM_FILE, cur_p-1, SEEK_SET);
            fscanf(ASM_FILE, "%s", label_cur->label);
            label_cur->label_add = inst_cnt;

            /* Connect the label link list.  */
            if(label_head == NULL) {
               label_head = label_tail = label_cur;
               label_tail->next = NULL;
            }
            else {
               label_tail->next = label_cur;
               label_tail = label_cur;
               label_tail->next = NULL;
```

```
            }

            /* Pass the current label and go to next character.  */
            if(cur_p == 1) inst_cnt++;
            inst_cnt--;
            cur_p += strlen(label_cur->label);
         }
         inst_cnt++;
      }
   }while(!feof(ASM_FILE));
}

void get_mnemonic(int cur_p, char *mnemonic) {
   /* Get Mnemonic.  */
   fseek(ASM_FILE, cur_p, SEEK_SET); // Redy to fetching position
   fscanf(ASM_FILE, "%s", mnemonic); // and ... read the mnemonic
   /* Its END of FILE ???  */
   if(feof(ASM_FILE)) {
      printf("SUCCESS...Generated Complete !\n");
      exit(0);
   }
}

int its_label(char *mnemonic, int *cur_p) {
   /* Its LABEL ??? (Label started with under score ('_')).  */
   if(mnemonic[0] == '_') {
      *cur_p += strlen(mnemonic) + 1;
      return(1);
   }
   return(0);
}

int get_inst_def(char *mnemonic) {
   /* Set the Current Instruction (inst_cur) to Instruction Head \
    * (inst_head).  */
   inst_cur = inst_head;
   /* If mnemonic that fetch from user assembly program equal to \
    * mnemonic at Instruction Collection (inst_colect) then stop, \
    * else print out ERROR message.  */
   while(1) {
      if(!strcmp(inst_cur->mnemonic, mnemonic))
         return(1);
      else if(inst_cur->next == NULL) {
         printf("ERROR : Instruction \"%s\" unrecognize.\n", mnemonic);
         return(0);
      }
      else
         inst_cur = inst_cur->next;
   };
}

void get_opcode(char *instruction, int *cur_p) {
   int i;
   /* Insert Opcode to 32 bits Instruction.  */
   for(i=0; i<=5; i++) {
      instruction[i] = inst_cur->opcode[i];
   }
```

```
   /* Go to First Operand with pass the mnemonic and one space.  */
   *cur_p += (inst_cur->mnemonic_length + 1);
   fseek(ASM_FILE, *cur_p, SEEK_SET);
}

void cmp_label(char *label_get, int inst_cnt, char *add_hex) {
   int i;
   unsigned long delta_add;  // delta address
   int tmp;                  // calculation tmeporary
   USHORT cnt;               // cnt = 4 if branch and cnt = 7 if jump

   /* Compare label_get with the colection of label from HEAD to \
    * TAIL.  */
   label_cur = label_head;
   while(1) {
      if(strcmp(label_get, label_cur->label) == 0) {
       /* Label is indentify so calculate the address.  */
       if(inst_cnt < label_cur->label_add)
            delta_add = label_cur->label_add - inst_cnt - 1;
       else
          /* -1 for 28 bit is 268435455 (unsign) where maximum bits \
           * for branch and target is 16 and 26 bits.  */
          delta_add = 268435455 - (inst_cnt - label_cur->label_add);
          break;
      }
      else if(label_cur->next == NULL) {
         printf("ERROR : Unrecognize label \"%s\".\n", label_get);
         exit(1);
      }
      else
         label_cur = label_cur->next;
   }

   if(inst_cur->operand_type == RA_RB_IMM)
      cnt = 4;    // branch
   else
      cnt = 7;  // jump

   for(i=0; i<cnt; i++) {
      tmp = delta_add % 16;
      if(tmp <= 9)
         add_hex[cnt-i] = (char)tmp + '0';  // 0 - 9
      else
         add_hex[cnt-i] = ((char)tmp - 10) + 'A';  // A - F
      delta_add /= 16;
   }
}

void operand_decode(char *instruction, int *cur_p, int inst_cnt) {
   USHORT i, j;
   USHORT loop_operand;
   char reg_dec[5], reg_bin[5];           // reg_dec ( RXX,@ )
   char imm_hex[7], imm_bin[4];           // imm_hex ( #XXXXh@ )
   char target_hex[10], target_bin[4];    // target_hex ( #XXXXXXXh@ )
   char branch_bin[16];
   char label_get[25];
```

```
        // loop three time to make sure that all operand fetched
        for(loop_operand=0; loop_operand<=2; loop_operand++) {
           // Get RD (Destenition Register)
           if((inst_cur->operand_type==RD_RA_RB  && loop_operand==0) ||
              (inst_cur->operand_type==RD_RA_IMM && loop_operand==0) ||
              (inst_cur->operand_type==RD_RA     && loop_operand==0) ||
              (inst_cur->operand_type==RD_IMM    && loop_operand==0) ||
              (inst_cur->operand_type==RD        && loop_operand==0)) {

              fscanf(ASM_FILE, "%s", reg_dec);
              regDEC_to_regBIN(reg_dec, reg_bin);
              for(i=6; i<=10; i++) instruction[i] = reg_bin[i-6];
              // Go to Next Operand
              *cur_p += 5;
              fseek(ASM_FILE, *cur_p, SEEK_SET);
           }
           else if((inst_cur->operand_type==RA_RB && loop_operand==0) ||
                    (inst_cur->operand_type==RB    && loop_operand==0)) {
               strcpy(reg_bin, "00000");
               for(i=6; i<=10; i++) instruction[i] = reg_bin[i-6];
           }

           // Get RA (Source Register A)
           if((inst_cur->operand_type== RD_RA_RB && loop_operand==1) ||
              (inst_cur->operand_type==RD_RA_IMM && loop_operand==1) ||
              (inst_cur->operand_type==RD_RA     && loop_operand==1) ||
              (inst_cur->operand_type==RA_RB     && loop_operand==0) ||
              (inst_cur->operand_type==RA_RB_IMM && loop_operand==0)) {
              fscanf(ASM_FILE, "%s", reg_dec);
              regDEC_to_regBIN(reg_dec, reg_bin);

              for(i=11; i<=15; i++) instruction[i] = reg_bin[i-11];
              // Go to Next Operand
             *cur_p += 5;
              fseek(ASM_FILE, *cur_p, SEEK_SET);
           }
           else if((inst_cur->operand_type==RD_IMM && loop_operand==0) ||
                   (inst_cur->operand_type==RD     && loop_operand==0) ||
                   (inst_cur->operand_type==RB     && loop_operand==0)) {
              strcpy(reg_bin, "00000");
              for(i=11; i<=15; i++) instruction[i] = reg_bin[i-11];
           }

           // Get RB (Source Register B)
         if((inst_cur->operand_type == RD_RA_RB  && loop_operand == 2) ||
              (inst_cur->operand_type == RA_RB     && loop_operand == 1)
||
              (inst_cur->operand_type == RB        && loop_operand == 0)
||
              (inst_cur->operand_type == RA_RB_IMM && loop_operand == 1))
{

              fscanf(ASM_FILE, "%s", reg_dec);
             regDEC_to_regBIN(reg_dec, reg_bin);

             for(i=16; i<=20; i++) instruction[i] = reg_bin[i-16];
```

```
            // Go to Next Operand
            *cur_p += 5;
            fseek(ASM_FILE, *cur_p, SEEK_SET);
         }

         // Get Immediate
           if((inst_cur->operand_type == RD_RA_IMM && loop_operand == 2)
||
              (inst_cur->operand_type == RD_IMM    && loop_operand == 1))
{

              fscanf(ASM_FILE, "%s", imm_hex);

              for(i=1; i<=4; i++) {
                 HEXchar_to_BINchar(imm_hex[i], imm_bin);
                for(j=0; j<=3; j++) instruction[16+((i-1)*4)+j] =
imm_bin[j];
              }

            // Go to Next Operand
            *cur_p += 7;
            fseek(ASM_FILE, *cur_p, SEEK_SET);
         }
         else if((inst_cur->operand_type == RD_RA && loop_operand == 0)
||
                 (inst_cur->operand_type == RD    && loop_operand ==
0)) {
              for(i=16; i<=31; i++) instruction[i] = '0';
         }

         // Get Immediate for Branch Target
         if((inst_cur->operand_type == RA_RB_IMM && loop_operand == 2)) {

            fscanf(ASM_FILE, "%s", label_get);
            cmp_label(label_get, inst_cnt, imm_hex);

            //fscanf(ASM_FILE, "%s", imm_hex);
            for(i=1; i<=4; i++) {
               HEXchar_to_BINchar(imm_hex[i], imm_bin);
               for(j=0; j<=3; j++) branch_bin[((i-1)*4) + j] =
imm_bin[j];
            }

            instruction[6]  = branch_bin[0];
            instruction[7]  = branch_bin[1];
            instruction[8]  = branch_bin[2];
            instruction[9]  = branch_bin[3];
            instruction[10] = branch_bin[4];

            for(i=1; i<=11; i++) instruction[20+i] = branch_bin[4+i];

            // Go to Next Operand
            *cur_p += strlen(label_get) + 1;
            fseek(ASM_FILE, *cur_p, SEEK_SET);
         }

         // Get Target
```

```
        if((inst_cur->operand_type == TARGET && loop_operand == 0)) {

             fscanf(ASM_FILE, "%s", label_get);
           cmp_label(label_get, inst_cnt, target_hex);

           for(i=1; i<=7; i++) {
              HEXchar_to_BINchar(target_hex[i], target_bin);
              for(j=0; j<=3; j++) {
                 if(i == 1) {
                  instruction[6] = target_bin[2];
                  instruction[7] = target_bin[3];
                  j = 4;
                }
                else
                   instruction[8 + ((i-2)*4) + j] = target_bin[j];
              }
           }

           // Go to Next Operand
           *cur_p += strlen(label_get) + 1;
           fseek(ASM_FILE, *cur_p, SEEK_SET);
        }

        // Unused : Instruction[21 to 31] = '0'
        if((inst_cur->operand_type == RD_RA_RB && loop_operand == 0) ||
             (inst_cur->operand_type == RA_RB    && loop_operand == 0)
||
             (inst_cur->operand_type == RB       && loop_operand == 0))
{

           for(i=21; i<=31; i++) instruction[i] = '0';
        }

        // Operand Type 9 -> Interrupt
        if(inst_cur->operand_type == NONE && loop_operand == 0)
           for(i=6; i<=31; i++) instruction[i] = '0';

        // JL dan JRL Instruction where PC value saved in R31
        if(inst_cur->link)
           for(i=6; i<=10; i++) instruction[i] = '1';

      } // end for (operand decode)
}

void cur_p_calibrate(int *cur_p) {
   /* 'cur_p' for operand type that ended with register (RD, RA, or RB)
must \
    * be subtract by 1.  */
   switch (inst_cur->operand_type) {
      case RD_RA_RB  : *cur_p -= 1; break;
      case RD_RA     : *cur_p -= 1; break;
      case RD        : *cur_p -= 1; break;
      case RA_RB     : *cur_p -= 1; break;
      case RB        : *cur_p -= 1; break;
      default : *cur_p -= 0;
   }
```

```
}
```

Rangkuman Laporan Hasil Sintesis

Synthesize										
Komponen	**Slice**	**FF**	**4 input LUT**	**IOB**	**T-BUF**	**Timing Report**				
						Prd.	**Freq.**	**IAT**	**ORT**	**CPD**
Multiplexer										
mux2x1	1 (0%)	---	1 (0%)	4 (2%)	---	---	---	---	---	8. 063
mux2x16	9 (0%)	---	16 (0%)	49 (34%)	---	---	---	---	---	9. 548
mux2x32	18	---	32	97	---	---	---	---	---	10.

	(0%)		(0%)	(67%)						448
mux4x1	1 (0%)	---	2 (0%)	7 (4%)	---	---	---	---	---	8. 549
mux4x16	16 (0%)	---	32 (0%)	82 (56%)	---	---	---	---	---	10. 763
mux4x32	32 (1%)	---	64 (1%)	162 (112%)	---	---	---	---	---	12. 203
mux16x1	5 (0%)	---	9 (0%)	21 (14%)	---	---	---	---	---	11. 088
mux16x16	73 (3%)	---	144 (3%)	276 (191%)	---	---	---	---	---	19. 557
mux16x32	133 (5%)	---	264 (5%)	548 (380%)	---	---	---	---	---	20. 230
mux32x1	9 (0%)	---	18 (0%)	38 (26%)	---	---	---	---	---	12. 078
mux32x16	137 (5%)	---	272 (5%)	533 (370%)	---	---	---	---	---	20. 566
mux32x32	265 (11%)	---	528 (11%)	1061 (736%)	---	---	---	---	---	20. 611
Adder										
adder32_lib	16 (0%)	---	32 (0%)	98 (68%)	---	---	---	---	---	10. 402
adder32_cl	58 (2%)	---	101 (2%)	98 (68%)	---	---	---	---	---	60. 857
adder32_cs	63 (2%)	---	109 (2%)	98 (68%)	---	---	---	---	---	41. 642
adder32_rc	42 (1%)	---	73 (1%)	98 (68%)	---	---	---	---	---	63. 116
Adder dan Subtractor										
addsub32 _lib_all	54 (2%)	---	100 (2%)	101 (70%)	---	---	---	---	---	17. 782
addsub32 _lib_par	17 (0%)	---	34 (0%)	101 (70%)	---	---	---	---	---	16. 342
addsub32 _cl	81 (3%)	---	140 (2%)	101 (70%)	---	---	---	---	---	67. 985
addsub32 _cs	82 (3%)	---	143 (2%)	101 (70%)	---	---	---	---	---	45. 242
addsub32 _rc	61 (2%)	---	106 (2%)	101 (70%)	---	---	---	---	---	67. 706

Synthesize										
Komponen	**Slice**	**FF**	**4 input LUT**	**IOB**	**T-BUF**	**Timing Report**				
						Prd.	**Freq.**	**IAT**	**ORT**	**CPD**
Shifter										
barrelshift 32_1	129 (5%)	---	225 (4%)	71 (49%)	---	---	---	---	---	27. 67
barrelshift 32_2	126 (5%)	---	238 (5%)	71 (49%)	---	---	---	---	---	25. 492
shifter_32	261 (11%)	---	496 (9%)	71 (49%)	---	---	---	---	---	25. 478
Function Unit										
zero_ detector	6 (0%)	---	11 (0%)	33 (22%)	---	---	---	---	---	12. 815
logicunit32	18	---	32	98	---	---	---	---	---	10.

	(0%)		(0%)	(68%)						448
alu	42 (1%)	---	78 (1%)	135 (93%)	---	---	---	---	---	19.816
lui	0 (0%)	---	0 (0%)	48 (33%)	---	---	---	---	---	6.479
function_unit	200 (8%)	---	380 (8%)	104 (72%)	---	---	---	---	---	29.077
function_unit_input	23 (0%)	---	44 (0%)	100 (69%)	---	---	---	---	---	15.901
Register File										
dec5_32	32 (1%)	---	32 (0%)	37 (25%)	---	---	---	---	---	10.763
load32	53 (2%)	---	92 (1%)	38 (26%)	---	---	---	---	---	13.886
reg1x32 RE_1	18 (0%)	---	32 (0%)	66 (45%)	---	---	---	5.082	6.788	---
register_file_ff	1088 (46%)	992 (21%)	1145 (24%)	113 (78%)	---	---	---	12.444	9.832	20.611
register_file_dpram	230 (9%)	---	198 (4%)	112 (77%)	---	---	---	3.004	9.315	17.486
Register File Gabungan										
const_unit	1 (0%)	---	1 (0%)	49 (34%)	---	---	---	---	---	9.719
register_file_gab	265 (11%)	---	262 (5%)	163 (113%)	---	---	---	3.004	10.899	19.070
register_file_gab_df	294 (12%)	---	313 (6%)	197 (136%)	---	---	---	3.004	10.899	20.654
Control Unit										
branch_ctrl	3 (0%)	---	5 (0%)	9 (6%)	---	---	---	---	---	10.367
id	27 (1%)	---	25 (0%)	51 (35%)	---	---	---	---	---	10.764
Id_test1										
Interrupt_ctrl	4 (0%)	1 (0%)	7 (0%)	16 (11%)	---	---	---	7.620	8.372	11.402

Synthesize										
Komponen	**Slice**	**FF**	**4 input LUT**	**IOB**	**T-BUF**	**Timing Report**				
						Prd.	**Freq.**	**IAT**	**ORT**	**CPD**
Register untuk Pipeline										
reg1x1	1 (0%)	1 (0%)	---	3 (2%)	---	---	---	2.520	6.788	---
reg1x2	1 (0%)	2 (0%)	---	5 (3%)	---	---	---	2.520	6.788	---
reg1x3	2 (0%)	3 (0%)	---	7 (4%)	---	---	---	2.520	6.788	---
reg1x4	2 (0%)	2 (0%)	---	9 (6%)	---	---	---	2.520	6.788	---
reg1x5	3 (0%)	5 (0%)	---	11 (7%)	---	---	---	2.520	6.788	---

reg1x26	15 (0%)	26 (0%)	---	53 (36%)	---	---	---	2. 520	6. 788	---
reg1x32	18 (0%)	32 (0%)	---	65 (45%)	---	---	---	2. 520	6. 788	---
Buffer										
bufe32	---	---	---	65 (45%)	---	---	---	---	---	10. 587
bufe3	---	---	---	7 (4%)	---	---	---	---	---	8. 499
Prosesor RISC										
risc	770 (32%)	315 (6%)	1200 (25%)	140 (97%)	---	34. 678	28. 942	13. 707	27. 690	17. 027
SRAM										
ram16x8s _1	8 (0%)	---	8 (0%)	21 (14%)	---	---	---	2. 599	7. 731	9. 475
ram32x32s _1	92 (3%)	---	105 (2%)	43 (29%)	---	---	---	---	9. 315	14. 074
Micro Controler										
uC kosong	136 (5%)	8 (0%)	197 (4%)	69 (47%)	64 (2%)	5. 183	192. 938	---	9. 063	---
uC 1 instruksi	258 (10%)	148 (3%)	403 (8%)	69 (47%)	64 (2%)	10. 025	99. 751	---	15. 463	---
uC Test1	894 (38%)	337 (7%)	1363 (28%)	70 (48%)	64 (2%)	22. 401	44. 641	8. 928	18. 183	12. 338
uC Test2	953 (40%)	340 (7%)	1480 (31%)	70 (48%)	64 (2%)	33. 316	30. 016	10. 548	26. 529	13. 958
uC Test3	919 (39%)	338 (7%)	1415 (30%)	70 (48%)	64 (2%)	33. 316	30. 016	10. 773	26. 124	14. 183
uC Test4	955 (40%)	349 (7%)	1452 (30%)	70 (48%)	64 (2%)	23. 620	42. 337	8. 807	18. 607	11. 852
uC Test5	889 (37%)	326 (6%)	1363 (28%)	70 (48%)	64 (2%)	33. 316	30. 016	12. 087	25. 944	14. 003
uC Test6	987 (41%)	350 (7%)	1531 (32%)	70 (48%)	64 (2%)	33. 712	29. 663	10. 431	25. 962	13. 796
uC Test7	775 (32%)	297 (6%)	1183 (25%)	69 (47%)	64 (2%)	22. 065	45. 321	---	15. 508	---

Program Pengujian

- Program 1

```
;; Nama File : Test1.asm
;; Test 1 : Test instruksi dasar, LA, dan LUI

ADIU R01, R00, #1234h   ;; R01 = 0000 1234 h
ADI R04, R00, #9ABCh    ;; R04 = FFFF 9ABC h
SLL R02, R01, #0010h    ;; R02 = 1234 0000 h
SBIU R05, R04, #9AACh   ;; R05 = FFFF 0010 h
ADIU R07, R00, #DEF0h   ;; R07 = 0000 DEF0 h
SLLV R06, R04, R05      ;; R06 = 9ABC 0000 h
ORI R03, R02, #5678h    ;; R03 = 1234 5678 h
```

```
OR R08, R06, R07        ;; R08 = 9ABC DEF0 h
ADD R09, R08, R03       ;; R09 = ACF1 3568 h
SUB R10, R08, R03       ;; R10 = 8888 8878 h
OR R11, R09, R03        ;; R11 = BEF5 7778 h
ORI R12, R10, #FF00h    ;; R12 = 8888 FF78 h
XOR R13, R10, R09       ;; R13 = 2479 BD10 h
XORI R14, R09, #F0F0h   ;; R14 = ACF1 C598 h
NOR R15, R13, R00       ;; R15 = DB86 42EF h
NORI R16, R13, #0000h   ;; R16 = DB86 42EF h
SLR R17, R01, #0002h    ;; R17 = 0000 048D h
SLRV R18, R11, R05      ;; R18 = 0000 BEF5 h
SAR R19, R15, #0011h    ;; R19 = FFFF EDC3 h
SARV R20, R15, R05      ;; R20 = FFFF DB86 h
LA R21                  ;; R21 = 0000 0015 h
SBI R22, R04, #9234h    ;; R22 = 0000 0888 h
AND R23, R03, R08       ;; R23 = 1234 5670 h
LUI R24, #9876h         ;; R24 = 9876 0000 h
ANDI R26, R20, #AAAAh   ;; R26 = 0000 8A82 h
ADIU R25, R00, #0080h   ;; R25 = 0000 0080 h
SW R25, R00
SLR R01, R01, #0008h
SB R25, R01
SLR R02, R02, #0008h
SB R25, R02
SLR R03, R03, #0008h
SB R25, R03
SLR R04, R04, #0008h
SB R25, R04
SLR R05, R05, #0008h
SB R25, R05
SLR R06, R06, #0008h
SB R25, R06
SLR R07, R07, #0008h
SB R25, R07
SLR R08, R08, #0008h
SB R25, R08
SLR R09, R09, #0008h
SB R25, R09
SLR R10, R10, #0008h
SB R25, R10
SLR R11, R11, #0008h
SB R25, R11
SLR R12, R12, #0008h
SB R25, R12
SLR R13, R13, #0008h
SB R25, R13
SLR R14, R14, #0008h
SB R25, R14
SLR R15, R15, #0008h
SB R25, R15
SLR R16, R16, #0008h
SB R25, R16
SLR R17, R17, #0008h
SB R25, R17
SLR R18, R18, #0008h
SB R25, R18
SLR R19, R19, #0008h
```

```
SB R25, R19
SLR R20, R20, #0008h
SB R25, R20
SLR R21, R21, #0008h
SB R25, R21
SLR R22, R22, #0008h
SB R25, R22
SLR R23, R23, #0008h
SB R25, R23
SLR R24, R24, #0008h
SB R25, R24
SLR R25, R25, #0008h
SB R25, R25
SLR R26, R26, #0008h
SB R25, R26
```

- **Program 2**

```
;; Nama File : Test2.asm
;; Test 2 : Test Instruksi Percabangan dan Set

LUI R01, #0FFFh          ;; R01 = 0FFF 0000 h
ORI R01, R01, #8006h     ;; R01 = 0FFF 8006 h
JR R01                   ;; JMP ke 7
ADI R08, R00, #FFFFh
JMP #000000Dh            ;; JMP ke 19
ADI R09, R00, #FFFFh
ADI R02, R00, #6C0Dh     ;; R02 = 0000 6C0D h
SLT R04, R00, R01        ;; R04 = 0000 0001 h
ADI R03, R00, #FF04h     ;; R03 = FFFF FF04 h
JRL R03                  ;; JMP ke 5
ADI R10, R00, #FFFFh
BE R00, R04, #0008h      ;; NO JMP
SLTI R05, R02, #8ACDh    ;; R05 = 0000 0000 h
SLTI R06, R02, #7ACDh    ;; R06 = 0000 0001 h
BH R05, R04, #0005h      ;; NO JMP
AND R07, R01, R03        ;; R07 = 0FFF 8004 h
BHE R04, R05, #0003h     ;; JMP ke 21
ADI R11, R00, #FFFFh
JL #FFFFFF8h             ;; JMP ke 12
ADI R12, R00, #FFFFh
ADIU R25, R00, #0080h
SB R25, R01
SLR R01, R01, #0008h
SB R25, R01
SLR R01, R01, #0008h
SB R25, R01
SLR R01, R01, #0008h
SB R25, R01
SB R25, R02
SLR R02, R02, #0008h
SB R25, R02
SLR R02, R02, #0008h
SB R25, R02
SLR R02, R02, #0008h
```

```
SB R25, R02
SB R25, R03
SLR R03, R03, #0008h
SB R25, R03
SLR R03, R03, #0008h
SB R25, R03
SLR R03, R03, #0008h
SB R25, R03
SB R25, R04
SLR R04, R04, #0008h
SB R25, R04
SLR R04, R04, #0008h
SB R25, R04
SLR R04, R04, #0008h
SB R25, R04
SB R25, R05
SLR R05, R05, #0008h
SB R25, R05
SLR R05, R05, #0008h
SB R25, R05
SLR R05, R05, #0008h
SB R25, R05
SB R25, R06
SLR R06, R06, #0008h
SB R25, R06
SLR R06, R06, #0008h
SB R25, R06
SLR R06, R06, #0008h
SB R25, R06
SB R25, R07
SLR R07, R07, #0008h
SB R25, R07
SLR R07, R07, #0008h
SB R25, R07
SLR R07, R07, #0008h
SB R25, R07
SB R25, R08
SLR R08, R08, #0008h
SB R25, R08
SLR R08, R08, #0008h
SB R25, R08
SLR R08, R08, #0008h
SB R25, R08
SB R25, R09
SLR R09, R09, #0008h
SB R25, R09
SLR R09, R09, #0008h
SB R25, R09
SLR R09, R09, #0008h
SB R25, R09
SB R25, R10
SLR R10, R10, #0008h
SB R25, R10
SLR R10, R10, #0008h
SB R25, R10
SLR R10, R10, #0008h
SB R25, R10
```

```
SB R25, R11
SLR R11, R11, #0008h
SB R25, R11
SLR R11, R11, #0008h
SB R25, R11
SLR R11, R11, #0008h
SB R25, R11
SB R25, R12
SLR R12, R12, #0008h
SB R25, R12
SLR R12, R12, #0008h
SB R25, R12
SLR R12, R12, #0008h
SB R25, R12
```

- **Program 3**

```
;; Nama File : Test3.asm
;; Test 3 : "Data Forwarding" dan "Branch Prediction"

ADIU R08, R00, #0080h   ;; Store Address of IO (LED)

ADIU R01, R00, #1111h   ;;R01 = 0000 1111
BE R01, R01, #0005h     ;;Lompat ke 0007
ADI R01, R00, #FFFFh
ADD R00, R00, R00       ;;    -NOP
ADD R00, R00, R00       ;;    -NOP
ADD R00, R00, R00       ;;    -NOP
ADD R00, R00, R00       ;;    -NOP
SB R08, R01             ;;-->Output harus 0000 1111

ADIU R02, R00, #2222h   ;;R02 = 0000 2222
BH R02, R01, #0005h     ;;Lompat ke 000F
ADI R02, R00, #FFFFh
ADD R00, R00, R00       ;;    -NOP
ADD R00, R00, R00       ;;    -NOP
ADD R00, R00, R00       ;;    -NOP
ADD R00, R00, R00       ;;    -NOP
SB R08, R02             ;;-->Output harus 0000 2222

ADIU R03, R00, #3333h   ;;R03 = 0000 3333
BHE R03, R01, #0005h    ;;Lompat ke 0017
ADI R03, R00, #FFFFh
ADD R00, R00, R00       ;;    -NOP
ADD R00, R00, R00       ;;    -NOP
ADD R00, R00, R00       ;;    -NOP
ADD R00, R00, R00       ;;    -NOP
SB R08, R03             ;;-->Output harus 0000 3333

ADI R04, R00, #8888h    ;;R04 = FFFF 8888
BG R02, R04, #0005h     ;;Lompat ke 001F
ADI R04, R00, #FFFFh
ADD R00, R00, R00       ;;    -NOP
ADD R00, R00, R00       ;;    -NOP
ADD R00, R00, R00       ;;    -NOP
```

```
ADD R00, R00, R00       ;;    -NOP
SB R08, R04             ;;-->Output harus FFFF 8888

ADI R05, R00, #9999h    ;;R05 = FFFF 9999
BGE R02, R05, #0005h    ;;Lompat ke 0027
ADI R05, R00, #FFFFh
ADD R00, R00, R00       ;;    -NOP
ADD R00, R00, R00       ;;    -NOP
ADD R00, R00, R00       ;;    -NOP
ADD R00, R00, R00       ;;    -NOP
SB R08, R05             ;;-->Output harus FFFF 9999

SLR R06, R01, #0008h    ;;R06 = 0000 0011
ADD R07, R00, R06       ;;R07 = 0000 0011
ADD R07, R07, R06       ;;R07 = 0000 0022
ADD R07, R07, R06       ;;R07 = 0000 0033
ADD R07, R07, R06       ;;R07 = 0000 0044
ADD R07, R07, R06       ;;R07 = 0000 0055
ADD R07, R07, R06       ;;R07 = 0000 0066
SB R08, R07             ;;-->Output harus 0000 0066

ADD R01, R00, R00       ;;-->Clear R01 s/d R07
ADD R02, R00, R00
ADD R03, R00, R00
ADD R04, R00, R00
ADD R05, R00, R00
ADD R06, R00, R00
ADD R07, R00, R00
JMP #FFFFFC7h           ;;-->Kembali ke 0000 >> -57d = -39h = +C7h
```

- **Program 4**

```
;* Nama File : Test4.asm       *
 * SB, SH, SW, LB, LH, LW Test *;
;; SB ---------------

ADIU R01, R00, #0080h   ;; R01 = 0000 0080
ADIU R02, R00, #0081h   ;; R02 = 0000 0081
LUI R11, #1234h
ADI R11, R11, #5678h    ;; R11 = 1234 5678
SB R00, R11
SBI R12, R11, #5673h    ;; R12 = 1234 0005
SB R12, R11
ADD R13, R12, R12       ;; R13 = 2468 000A
SB R13, R11
ADD R13, R13, R12       ;; R13 = 369C 000F
SB R13, R11
SBI R14, R13, #000Eh    ;; R14 = 369C 0001
ADD R13, R13, R14       ;; R13 = 48D0 0010
SB R13, R11

LW R15, R00             ;; R15 ->> 0000 0078 -->Hasil
SBIU R29, R12, #0001h   ;; R29 = 1234 0004
LW R16, R29             ;; R16 ->> 0000 7800 -->Hasil
ADD R28, R29, R29       ;; R28 = 2468 0008
```

```
LW R17, R28             ;; R17 ->> 0078 0000 -->Hasil
ADD R28, R28, R29       ;; R28 = 369C 000C
LW R18, R28             ;; R18 ->> 7800 0000 -->Hasil
ADD R28, R28, R29       ;; R28 = 48D0 0010
LW R19, R28             ;; R19 ->> 0000 0078 -->Hasil

SB R01, R15
SLR R15, R15, #0008h
SB R02, R15
SLR R15, R15, #0008h
SB R01, R15
SLR R15, R15, #0008h
SB R02, R15

SB R01, R16
SLR R16, R16, #0008h
SB R02, R16
SLR R16, R16, #0008h
SB R01, R16
SLR R16, R16, #0008h
SB R02, R16

SB R01, R17
SLR R17, R17, #0008h
SB R02, R17
SLR R17, R17, #0008h
SB R01, R17
SLR R17, R17, #0008h
SB R02, R17

SB R01, R18
SLR R18, R18, #0008h
SB R02, R18
SLR R18, R18, #0008h
SB R01, R18
SLR R18, R18, #0008h
SB R02, R18

SB R01, R19
SLR R19, R19, #0008h
SB R02, R19
SLR R19, R19, #0008h
SB R01, R19
SLR R19, R19, #0008h
SB R02, R19

;; SH ---------------

LUI R21, #1234h
ADI R21, R21, #5678h    ;; R21 = 1234 5678
SH R00, R21
SBI R22, R21, #5672h    ;; R22 = 1234 0006
SH R22, R21
ADI R22, R22, #0002     ;; R22 = 1234 0008
SH R22, R21

LW R25, R00             ;; R25 ->> 0000 5678 -->Hasil
```

```
SBIU R29, R22, #0004h   ;; R29 = 1234 0004
LW R26, R29             ;; R26 ->> 5678 7800 -->Hasil : Nilai 78h pada
                        ;; bit ke 8 s/d 15 merupakan nilai dari proses
                        ;; sebelumnya
ADD R28, R29, R29       ;; R28 = 2468 0008
LW R27, R28             ;; R27 ->> 0078 5678 -->Hasil : Nilai 78h pada
                        ;; bit ke 16 s/d 23 merupakan nilai dari proses
                        ;; sebelumnya

SB R01, R25
SLR R25, R25, #0008h
SB R02, R25
SLR R25, R25, #0008h
SB R01, R25
SLR R25, R25, #0008h
SB R02, R25

SB R01, R26
SLR R26, R26, #0008h
SB R02, R26
SLR R26, R26, #0008h
SB R01, R26
SLR R26, R26, #0008h
SB R02, R26

SB R01, R27
SLR R27, R27, #0008h
SB R02, R27
SLR R27, R27, #0008h
SB R01, R27
SLR R27, R27, #0008h
SB R02, R27

;; SW ---------------

LUI R21, #1234h
ADI R21, R21, #5678h    ;; R21 = 1234 5678
SW R00, R21
SBI R22, R21, #5674h    ;; R22 = 1234 0004
SW R22, R21

LW R25, R00             ;; R25 ->> 1234 5678 -->Hasil
LW R26, R22             ;; R26 ->> 1234 5678 -->Hasil

SB R01, R25
SLR R25, R25, #0008h
SB R02, R25
SLR R25, R25, #0008h
SB R01, R25
SLR R25, R25, #0008h
SB R02, R25

SB R01, R26
SLR R26, R26, #0008h
SB R02, R26
SLR R26, R26, #0008h
SB R01, R26
```

```
SLR R26, R26, #0008h
SB R02, R26

;; LB & LH & LW ---------------

LUI R20, #9876h
ADI R20, R20, #1234h
SW R00, R20             ;; R20 = 9876 1234
LB R03, R00             ;; R03 = 0000 0034 --> Hasil
ADI R25, R00, #0001h
LB R04, R25             ;; R04 = 0000 0012 --> Hasil
ADI R25, R25, #0001h
LB R05, R25             ;; R05 = 0000 0076 --> Hasil
ADI R25, R25, #0001h
LB R06, R25             ;; R06 = 0000 0098 --> Hasil
LH R07, R00             ;; R07 = 0000 1234 --> Hasil
ADI R25, R00, #0002h
LH R08, R25             ;; R08 = 0000 9876 --> Hasil

SB R01, R03
SLR R03, R03, #0008h
SB R02, R03
SLR R03, R03, #0008h
SB R01, R03
SLR R03, R03, #0008h
SB R02, R03

SB R01, R04
SLR R04, R04, #0008h
SB R02, R04
SLR R04, R04, #0008h
SB R01, R04
SLR R04, R04, #0008h
SB R02, R04

SB R01, R05
SLR R05, R05, #0008h
SB R02, R05
SLR R05, R05, #0008h
SB R01, R05
SLR R05, R05, #0008h
SB R02, R05

SB R01, R06
SLR R06, R06, #0008h
SB R02, R06
SLR R06, R06, #0008h
SB R01, R06
SLR R06, R06, #0008h
SB R02, R06

SB R01, R07
SLR R07, R07, #0008h
SB R02, R07
SLR R07, R07, #0008h
SB R01, R07
SLR R07, R07, #0008h
```

```
SB R02, R07

SB R01, R08
SLR R08, R08, #0008h
SB R02, R08
SLR R08, R08, #0008h
SB R01, R08
SLR R08, R08, #0008h
SB R02, R08
```

- **Program 5**

```
;; Nama File : Test5.asm
;; Procedure (Call/Return) and Stack (Push/Pop) Test

ADI R29, R00, #007Ch    ;; SP

ADI R11, R00, #0080h    ;; ADD1
ADI R12, R00, #0081h    ;; ADD2

ADI R01, R00, #0002h    ;; R01 = 2
ADI R02, R00, #0004h    ;; R02 = 4

;; Before Call
SW R29, R01             ;; Nilai yg tdk berubah di PUSH
SBI R29, R29, #0004h
ADI R05, R00, #0013h    ;; RX <- Target
SW R29, R31             ;; PUSH RAR
JRL R05                 ;; JR RX

;; Return Target
ADI R29, R29, #0004h    ;; Nilai yg berubah di POP
LW R02, R29
ADI R29, R29, #0004h    ;; Nilai yg tdk berubah di POP
LW R01, R29
ADI R29, R29, #0004h    ;; Nilai "return" berikutnya
LW R31, R29

SB R11, R01             ;; ADD1 = 2
SB R12, R02             ;; ADD2 = 9
JR R00
;; Procedure Proc1
ADI R01, R01, #0005h    ;; R01 += 5
ADI R02, R02, #0005h    ;; R02 += 5

;; Before Return
SW R29, R02             ;; Nilai yg berubah di PUSH
SBI R29, R29, #0004h

;; Return
JR R31                  ;; JR RAR
```

- **Program 6**

```
;*****************************************************************
;     Nama File : Test6.asm
;     Test 6 : Interrupt Test
;     Animasi LED
;
;     Jumlah LED adalah 16 buah
;     Pergantian animasi dilakukan menggunakan interrupt
*****************************************************************;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;; Interrupt Vector
Table
DI
JMP #0000004h
DI
JMP #000001Fh
DI
JMP #0000039h

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;; Main Program
;; Address : 6d = 6h
;; Deklarasi:
EI
ADIU R01, R00, #0001h   ;; R01 Yang akan di tampilkan 8-bit LSB
ADD R06, R00, R00       ;; R06 Yang akan di tampilkan 8-bit MSB
ADIU R02, R00, #8000h   ;; R02 Batas kiri
ADIU R05, R00, #0001h   ;; R05 Batas kanan
LUI R03, #0000h         ;; R03 MaxDelay
ADIU R03, R03, #7FFFh

ADIU R07, R00, #0080h   ;; R07 Alamat I/O LED 8-bit LSB
ADIU R08, R00, #0081h   ;; R08 Alamat I/O LED 8-bit MSB

;; Awal:

ADD R04, R00, R00       ;; R04 Clear Counter Tampil

;; Tampil_geser_kiri:

SB R07, R01             ;; Menampilkan LED
SLR R06, R01, #0008h
SB R08, R06
ADD R04, R04, R05       ;; Increase Counter Tampil
BE R04, R03, #0001h     ;; Lompat jk Counter = MaxDelay
JMP #FFFFFFAh           ;; Tampil_geser_kiri:
SLL R01, R01, #0001h    ;; Shift LED ke kiri
BH R01, R02, #0001h     ;; Lompat jk LED melebihi batas kiri
JMP #FFFFFF6h           ;; Awal:

;; Tampil_geser_kanan:

ADD R04, R00, R00       ;; Clear Counter Tampil
SLR R01, R01, #0001h    ;; Shift LED ke kanan
;; Loop_kanan:
SB R07, R01             ;; Menampilkan LED
SLR R06, R01, #0008h
SB R08, R06
ADD R04, R04, R05       ;; Increase Counter Tampil
```

```
BE R04, R03, #0001h      ;; Lompat jk Counter = MaxDelay
JMP #FFFFFFAh            ;; Loop_kanan:
BE R01, R05, #FFEDh      ;; Lompat jika LED melebihi batas kanan ke
Awal:
JMP #FFFFFF6h            ;; Tampil_geser_kanan:

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;; Interrupt 1
;; Address : 35d = 23h
;; Int1:
ADIU R11, R00, #0001h    ;; R11 LED Jalan
ADIU R13, R00, #8000h    ;; R13 Batas Kiri
ADIU R14, R00, #0001h    ;; R14 Flag Selesai

LUI R15, #0000h          ;; R15 MaxDelay
ADIU R15, R15, #FFFFh

ADIU R22, R00, #002Dh    ;; Alamat Loop_geser:
ADIU R23, R00, #002Eh    ;; Alamat Loop_delay:

ADIU R24, R00, #0000h    ;; R24 Yang akan di tampilkan 8-bit LSB
ADIU R25, R00, #0000h    ;; R25
ADIU R26, R00, #0000h    ;; R26 Yang akan di tampilkan 8-bit MSB

;; Loop_geser:

ADD R17, R00, R00        ;; R17 Clear Counter

;; Loop_delay:

OR R24, R25, R11         ;; Pengabungan LED Tumpuk dengan LED Jalan
SB R07, R24              ;; Menampilkan LED
SLR R26, R24, #0008h
SB R08, R26

ADD R17, R17, R14        ;; Increase Counter
BE R17, R15, #0001h      ;; Lompat jika Counter = MaxDelay
JR R23                   ;; Loop_delay:

SLL R11, R11, #0001h     ;; Shift LED Jalan ke kiri
BE R13, R11, #0001h      ;; Lompat jk LED Jalan = Batas Kiri  -->
Perubahan!!!
JR R22                   ;; Loop_geser:
ADIU R11, R00, #0001h    ;; Reset LED Jalan
OR R25, R25, R13         ;; Penumpukkan
SLR R13, R13, #0001h     ;; Shift Batas Kiri ke kanan

BE R13, R14, #0001h      ;; Lompat jk Batas Kiri = Flag Selesai
JR R22                   ;; Loop_geser:

EI                       ;; Enable Interrupt
JR R30                   ;; Interrupt Return

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;; Interrupt 2
;; Address : 62d = 3Eh
;; Int2:

LB R27, R07              ;; R27 Data dari Modul IO
```

```
LUI R15, #000Fh         ;; R15 MaxDelay
ADIU R15, R15, #FFFFh

SB R07, R27             ;; Tampilkan LED
ADD R17, R00, R00       ;; R17 Counter

;; Loop_cetak:
ADI R17, R17, #0001h    ;; Increase Counter
BE R17, R15, #0001h     ;; Lompat jk Counter = MaxDelay
JMP #FFFFFFDh           ;; Loop_cetak:

EI                      ;; Enable Interrupt
JR R30                  ;; Interrupt Return
```

- **Program 7**

```
;*****************************************************************
;     Nama File : Test7.asm
;     Test 7 : Menghitung Panjang CLOCK
*****************************************************************;

LUI R01, #00FFh         ;; R01 MaxDelay
ADIU R01, R01, #FFFFh

ADIU R02, R00, #0080h   ;; 02 Alamat I/O LED 8-bit LSB

ADIU R03, R00, #000Fh   ;; 03 Data Mulai
ADIU R04, R00, #00F0h   ;; 04 Data Berhenti

ADD R05, R00, R00       ;; R05 Clear Counter Tampil

SB R02, R03             ;; Tahap 1
ADIU R05, R05, #0001h   ;; Increase Counter R05
BE R01, R05, #0001h     ;; Lompat jk Counter = MaxDelay
JMP #FFFFFFDh           ;; Tahap 1

ADD R05, R00, R00       ;; R05 Clear Counter Tampil

SB R02, R04             ;; Tahap 2
ADIU R05, R05, #0001h   ;; Increase Counter R05
BE R01, R05, #0001h     ;; Lompat jk Counter = MaxDelay
JMP #FFFFFFDh           ;; Tahap 2

ADD R05, R00, R00       ;; R05 Clear Counter Tampil

SB R02, R03             ;; Tahap 3
ADIU R05, R05, #0001h   ;; Increase Counter R05
BE R01, R05, #0001h     ;; Lompat jk Counter = MaxDelay
JMP #FFFFFFDh           ;; Tahap 3

ADD R05, R00, R00       ;; R05 Clear Counter Tampil

SB R02, R04             ;; Tahap 4
ADIU R05, R05, #0001h   ;; Increase Counter R05
```

```
BE R01, R05, #0001h     ;; Lompat jk Counter = MaxDelay
JMP #FFFFFFDh           ;; Tahap 4

JMP #FFFFFFEBh
```