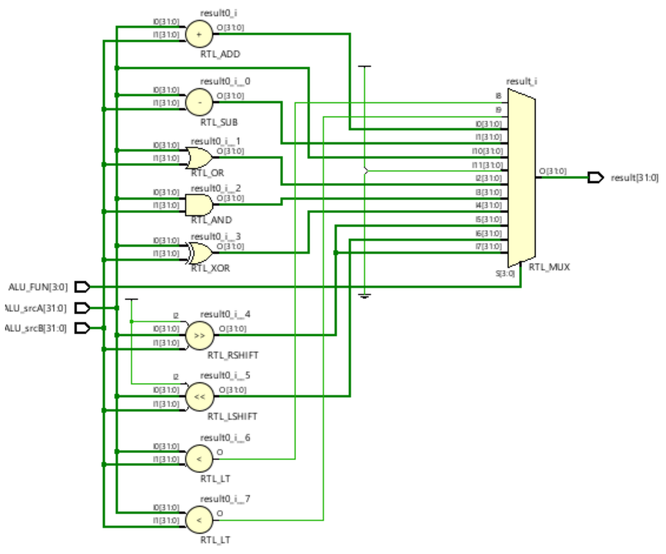
# <u>CPE 233 HW 4</u>

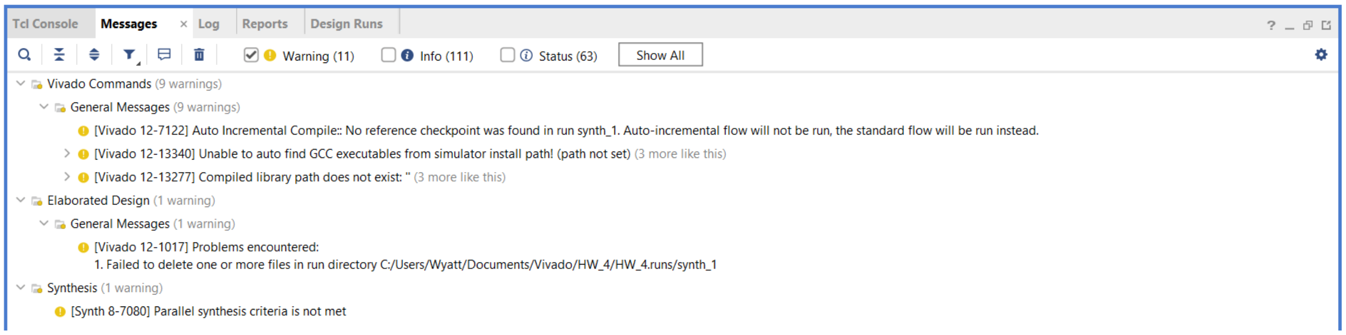
Wyatt Tack

1. Behavior Description: The ALU (short for arithmetic logic unit) is the actual operator of the OTTER MCU, in which it performs each of the mathematical, logical, and manipulations to the data stored in the REG_FILE. The design is a combinational circuit that runs each of the 11 operations in parallel and uses a large multiplexor to select which one is to be outputted as the answer. So, all operations possible are performed on the 2 inputs, but the function selector determines which one is actually outputted.

2. Structural Design:



3. Synthesis Warnings Listing:



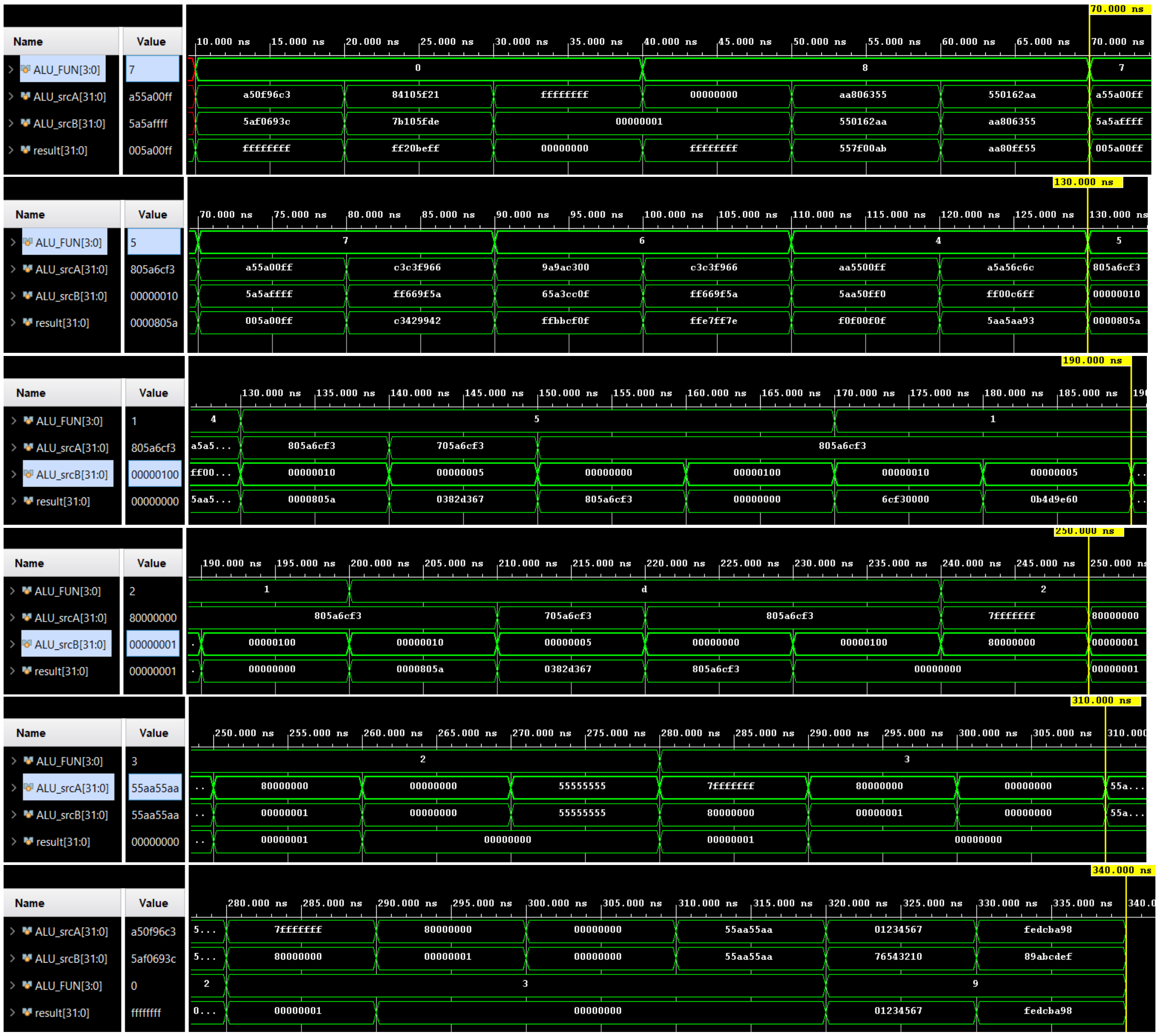
```
Vivado Commands (9 warnings)
  General Messages (9 warnings)
    [Vivado 12-7122] Auto Incremental Compile:: No reference checkpoint was found in run synth_1. Auto-incremental flow will not be run, the standard flow will be run instead.
    [Vivado 12-13340] Unable to auto find GCC executables from simulator install path! (path not set) (3 more like this)
    [Vivado 12-13277] Compiled library path does not exist: '' (3 more like this)
Elaborated Design (1 warning)
  General Messages (1 warning)
    [Vivado 12-1017] Problems encountered:
    1. Failed to delete one or more files in run directory C:/Users/Wyatt/Documents/Vivado/HW_4/HW_4.runs/synth_1
Synthesis (1 warning)
  [Synth 8-7080] Parallel synthesis criteria is not met
```

4. Verification:
   a. Table:

| Function | ALU_FUN | Arguments | | Expected Result (hex) |
|---|---|---|---|---|
| | | A | B | |
| **ADD** | 0000 | 0xA50F96C3 | 0x5AF0693C | FFFFFFFF |
| | 0000 | 0x84105F21 | 0x7B105FDE | FF20BEFF |
| | 0000 | 0xFFFFFFFF | 0x00000001 | 00000000 |
| **SUB** | 1000 | 0x00000000 | 0x00000001 | FFFFFFFF |
| | 1000 | 0xAA806355 | 0x550162AA | 557F00AB |
| | 1000 | 0x550162AA | 0xAA806355 | AA80FF55 |
| **AND** | 0111 | 0xA55A00FF | 0x5A5AFFFF | 005A00FF |
| | 0111 | 0xC3C3F966 | 0xFF669F5A | C3429942 |
| **OR** | 0110 | 0x9A9AC300 | 0x65A3CC0F | FFBBCF0F |
| | 0110 | 0xC3C3F966 | 0xFF669F5A | FFE7FF7E |
| **XOR** | 0100 | 0xAA5500FF | 0x5AA50FF0 | F0F00F0F |
| | 0100 | 0xA5A56C6C | 0xFF00C6FF | 5AA5AA93 |
| **SRL** | 0101 | 0x805A6CF3 | 0x00000010 | 0000805A |
| | 0101 | 0x705A6CF3 | 0x00000005 | 0382D367 |
| | 0101 | 0x805A6CF3 | 0x00000000 | 805A6CF3 |
| | 0101 | 0x805A6CF3 | 0x00000100 | 00000000 |
| **SLL** | 0001 | 0x805A6CF3 | 0x00000010 | 6CF30000 |
| | 0001 | 0x805A6CF3 | 0x00000005 | 0B4D9E60 |
| | 0001 | 0x805A6CF3 | 0x00000100 | 00000000 |
| **SRA** | 1101 | 0x805A6CF3 | 0x00000010 | 0000805A |
| | 1101 | 0x705A6CF3 | 0x00000005 | 0382D367 |
| | 1101 | 0x805A6CF3 | 0x00000000 | 805A6CF3 |
| | 1101 | 0x805A6CF3 | 0x00000100 | 00000000 |
| **SLT** | 0010 | 0x7FFFFFFF | 0x80000000 | 00000000 |
| | 0010 | 0x80000000 | 0x00000001 | 00000001 |
| | 0010 | 0x00000000 | 0x00000000 | 00000000 |
| | 0010 | 0x55555555 | 0x55555555 | 00000000 |

| | | | | |
|---|---|---|---|---|
| SLTU | 0011 | 0x7FFFFFFF | 0x80000000 | 00000001 |
| | 0011 | 0x80000000 | 0x00000001 | 00000000 |
| | 0011 | 0x00000000 | 0x00000000 | 00000000 |
| | 0011 | 0x55AA55AA | 0x55AA55AA | 00000000 |
| LUI COPY | 1001 | 0x01234567 | 0x76543210 | 01234567 |
| | 1001 | 0xFEDCBA98 | 0x89ABCDEF | FEDCBA98 |

Simulation:



5. System Verilog Source Code:

For ALU:

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////
// Company: Cal Poly SLO
// Engineer: Wyatt Tack
//
// Create Date: 01/24/2024
// Design Name: ALU
// Project Name: OTTER MCU
// Target Devices: Basys 3 Board
// Description: Arithmatic Logic Unit, processes functions
//              on OTTER MCU
//
//////////////////////////////////////////////////////////////////

module ALU(
input [31:0] ALU_srcA, //Only 2 inputs, a select for the
input [31:0] ALU_srcB, //operation, and the output
input [3:0] ALU_FUN,
output logic [31:0] result
);
always_comb
begin            //combinational multiplexor of logic circuits
case (ALU_FUN)   //case statement outputs data to result
    4'b0000: result = ALU_srcA + ALU_srcB; //add
    4'b1000: result = ALU_srcA - ALU_srcB; //sub
    4'b0110: result = ALU_srcA | ALU_srcB; //or
    4'b0111: result = ALU_srcA & ALU_srcB; //and
    4'b0100: result = ALU_srcA ^ ALU_srcB; //xor
    4'b0101: result = ALU_srcA >> ALU_srcB; //SRL
    4'b0001: result = ALU_srcA << ALU_srcB; //SLL
    4'b1101: result = ALU_srcA >>> ALU_srcB; //SRA
    4'b0010: if($signed(ALU_srcA) < $signed(ALU_srcB))
                      result = 32'h00000001; //SLT
              else result = 32'h00000000;
    4'b0011: if(ALU_srcA < ALU_srcB)
                      result = 32'h00000001; //SLTU
              else result = 32'h00000000;
    4'b1001: result = ALU_srcA; //LUI-copy
    default: result = 32'hafafbcbc;
endcase
end
endmodule
```

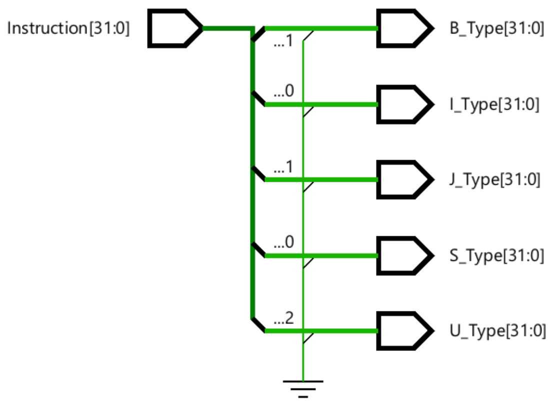
For ALU Simulation Test Bench:

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////
// Company: Cal Poly SLO
// Engineer: Wyatt Tack
//
// Create Date: 01/24/2024
// Design Name: ALU Test Bench
// Project Name: OTTER MCU
// Target Devices: Basys 3 Board
// Description: Arithmatic Logic Unit, processes functions
//                on OTTER MCU simulation source
//////////////////////////////////////////////////////////////
module ALU_TB();
logic [31:0] ALU_srcA; //same inputs and outputs as ALU,
logic [31:0] ALU_srcB; //test cases received from CPE 233
logic [3:0] ALU_FUN;   //lab manual
logic [31:0] result;
ALU UUT (.ALU_srcA(ALU_srcA), .ALU_srcB(ALU_srcB), .ALU_FUN(ALU_FUN), .result(result));
always begin
#10 ALU_FUN = 4'b0000; //add cases
    ALU_srcA = 32'hA50F96C3;
    ALU_srcB = 32'h5AF0693C;
#10 ALU_srcA = 32'h84105F21;
    ALU_srcB = 32'h7B105FDE;
#10 ALU_srcA = 32'hFFFFFFFF;
    ALU_srcB = 32'h00000001;
#10 ALU_FUN = 4'b1000; //sub cases
    ALU_srcA = 32'h00000000;
    ALU_srcB = 32'h00000001;
#10 ALU_srcA = 32'hAA806355;
    ALU_srcB = 32'h550162AA;
#10 ALU_srcA = 32'h550162AA;
    ALU_srcB = 32'hAA806355;
#10 ALU_FUN = 4'b0111; //and cases
    ALU_srcA = 32'hA55A00FF;
    ALU_srcB = 32'h5A5AFFFF;
#10 ALU_srcA = 32'hC3C3F966;
    ALU_srcB = 32'hFF669F5A;
#10 ALU_FUN = 4'b0110; //or cases
    ALU_srcA = 32'h9A9AC300;
    ALU_srcB = 32'h65A3CC0F;
#10 ALU_srcA = 32'hC3C3F966;
    ALU_srcB = 32'hFF669F5A;
#10 ALU_FUN = 4'b0100; //xor cases
    ALU_srcA = 32'hAA5500FF;
    ALU_srcB = 32'h5AA50FF0;
#10 ALU_srcA = 32'hA5A56C6C;
    ALU_srcB = 32'hFF00C6FF;
#10 ALU_FUN = 4'b0101; //srl cases
    ALU_srcA = 32'h805A6CF3;
    ALU_srcB = 32'h00000010;
#10 ALU_srcA = 32'h705A6CF3;
    ALU_srcB = 32'h00000005;
#10 ALU_srcA = 32'h805A6CF3;
    ALU_srcB = 32'h00000000;
#10 ALU_srcA = 32'h805A6CF3;
    ALU_srcB = 32'h00000100;
#10 ALU_FUN = 4'b0001; //sll cases
    ALU_srcA = 32'h805A6CF3;
    ALU_srcB = 32'h00000010;
#10 ALU_srcA = 32'h805A6CF3;
    ALU_srcB = 32'h00000005;
#10 ALU_srcA = 32'h805A6CF3;
    ALU_srcB = 32'h00000100;
#10 ALU_FUN = 4'b1101; //sra cases
    ALU_srcA = 32'h805A6CF3;
    ALU_srcB = 32'h00000010;
#10 ALU_srcA = 32'h705A6CF3;
    ALU_srcB = 32'h00000005;
#10 ALU_srcA = 32'h805A6CF3;
    ALU_srcB = 32'h00000000;
#10 ALU_srcA = 32'h805A6CF3;
    ALU_srcB = 32'h00000100;
#10 ALU_FUN = 4'b0010; //slt cases
    ALU_srcA = 32'h7FFFFFFF;
    ALU_srcB = 32'h80000000;
#10 ALU_srcA = 32'h80000000;
    ALU_srcB = 32'h00000001;
#10 ALU_srcA = 32'h00000000;
    ALU_srcB = 32'h00000000;
#10 ALU_srcA = 32'h55555555;
    ALU_srcB = 32'h55555555;
#10 ALU_FUN = 4'b0011; //sltu cases
    ALU_srcA = 32'h7FFFFFFF;
    ALU_srcB = 32'h80000000;
#10 ALU_srcA = 32'h80000000;
    ALU_srcB = 32'h00000001;
#10 ALU_srcA = 32'h00000000;
    ALU_srcB = 32'h00000000;
#10 ALU_srcA = 32'h55AA55AA;
    ALU_srcB = 32'h55AA55AA;
#10 ALU_FUN = 4'b1001; //lui-copy cases
    ALU_srcA = 32'h01234567;
    ALU_srcB = 32'h76543210;
#10 ALU_srcA = 32'hFEDCBA98;
    ALU_srcB = 32'h89ABCDEF;
end
endmodule
```

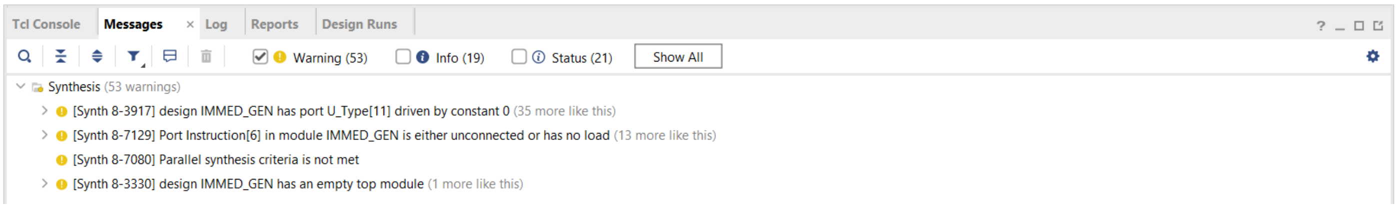
## Part 2:

1. Behavior Description: The immediate generator (IMMED_GEN) is used to take in the machine code, and for each type of code that the machine code could be produce an immediate for it. There are 5 different types of immediate, so the Immediate generator produces them all in parallel, and through the other parts of the machine code help select which of those immediate are actually meant to be used. Each different type of machine code lays out the immediate values differently in its set of binary instruction values, so concatenation is used in the IMMED_GEN to untangle the bits into the organized way they should be.

2. Structural Design:



3. Synthesis Warnings Listing:



```
Synthesis (53 warnings)
  [Synth 8-3917] design IMMED_GEN has port U_Type[11] driven by constant 0 (35 more like this)
  [Synth 8-7129] Port Instruction[6] in module IMMED_GEN is either unconnected or has no load (13 more like this)
  [Synth 8-7080] Parallel synthesis criteria is not met
  [Synth 8-3330] design IMMED_GEN has an empty top module (1 more like this)
```

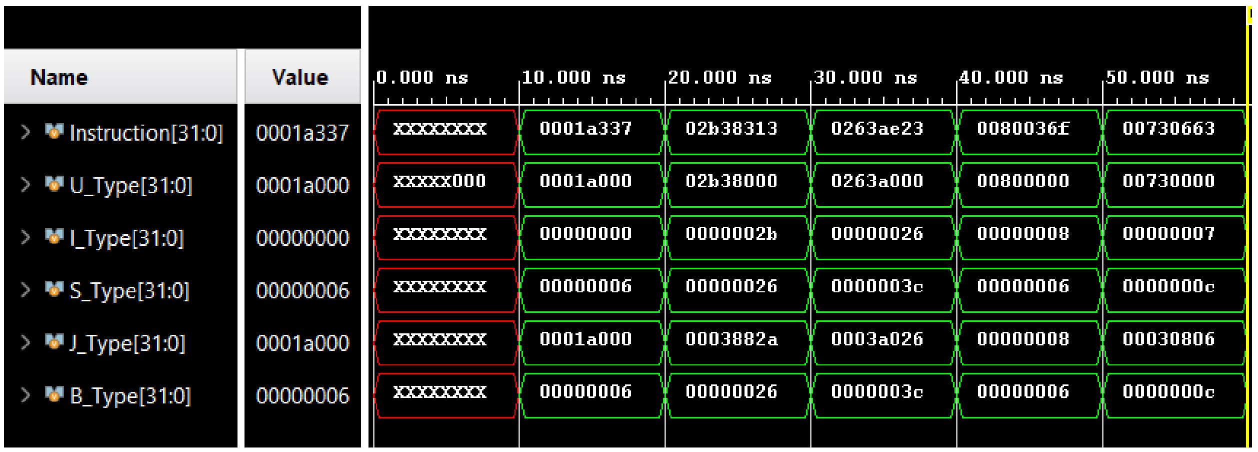
4. Verification:
   a. Assembly code used to create machine code with immediates:

```
                lui x6, 0x1a          #U-Type
                addi x6, x7, 0x2b     #I-Type
                sw x6, 0x3c(x7)       #S-Type
                jal x6, EIGHT         #J-Type
                beq x6, x7, TWELVE    #B-Type
EIGHT:          nop
                nop     #last sets of nop used to
TWELVE:         nop     #set labels for j and b
```

b. Test Case Table:

| Test Case Input | Test Case Output | Test Case Reasoning |
|---|---|---|
| 0001a337<br>`lui x6, 0x1a` | `U-Type = 0001a000` | `Tests U-Type command LUI` |
| 02b38313<br>`addi x6, x7, 0x2b` | `I-Type = 0000002b` | `Tests I-Type command ADDI` |
| 0263ae23<br>`sw x6, 0x3c(x7)` | `S-Type = 0000003c` | `Tests S-Type command SW` |
| 0080036f<br>`jal x6, EIGHT` | `J-Type = 00000008` | `Tests J-Type command JAL` |
| 00730663<br>`beq x6, x7, TWELVE` | `B-Type = 0000000c` | `Tests B-Type command BEQ` |

c. Simulation:



```
Name               Value     0.000 ns  10.000 ns  20.000 ns  30.000 ns  40.000 ns  50.000 ns
Instruction[31:0]  0001a337  XXXXXXXX  0001a337   02b38313   0263ae23   0080036f   00730663
U_Type[31:0]       0001a000  XXXXX000  0001a000   02b38000   0263a000   00800000   00730000
I_Type[31:0]       00000000  XXXXXXXX  00000000   0000002b   00000026   00000008   00000007
S_Type[31:0]       00000006  XXXXXXXX  00000006   00000026   0000003c   00000006   0000000c
J_Type[31:0]       0001a000  XXXXXXXX  0001a000   0003882a   0003a026   00000008   00030806
B_Type[31:0]       00000006  XXXXXXXX  00000006   00000026   0000003c   00000006   0000000c
```

5. System Verilog Source Code:
   a. IMMED_GEN:

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////
// Company: Cal Poly SLO
// Engineer: Wyatt Tack
//
// Create Date: 01/24/2024
// Design Name: IMMED_GEN
// Project Name: OTTER MCU
// Target Devices: Basys 3 Board
// Description: Generates Immediate numbers needed from
//              different instruction types in assembly
//
//////////////////////////////////////////////////////////////////
module IMMED_GEN(
input [31:0] Instruction, //full machine code
output [31:0] U_Type, I_Type, S_Type, J_Type, B_Type
    ); //5 assignments of concatination for different types
assign U_Type = {Instruction[31:12], {12{1'b0}}};
assign I_Type = {{20{Instruction[31]}}, Instruction[30:20]};
assign S_Type = {{20{Instruction[31]}}, Instruction[30:25],
                    Instruction[11:7]};
assign J_Type = {{11{Instruction[31]}}, Instruction[19:12],
Instruction[20],
                    Instruction[30:21], 1'b0};
assign B_Type = {{19{Instruction[31]}}, Instruction[7],
Instruction[30:25],
                    Instruction[11:8], 1'b0};
endmodule
```

b. IMMED_GEN Simulation Source:

```
`timescale 1ns / 1ps
///////////////////////////////////////////////////////////////
// Company: Cal Poly SLO
// Engineer: Wyatt Tack
//
// Create Date: 01/24/2024
// Design Name: IMMED_GEN_TB
// Project Name: OTTER MCU
// Target Devices: Basys 3 Board
// Description: Generates Immediate numbers needed from
//              different instruction types in assembly
//              Simulation Source
///////////////////////////////////////////////////////////////
module IMMED_GEN_TB();
logic [31:0] Instruction; //full machine code
logic [31:0] U_Type, I_Type, S_Type, J_Type, B_Type;
IMMED_GEN UUT (.Instruction(Instruction), .U_Type(U_Type),
                .I_Type(I_Type), .S_Type(S_Type),
                .J_Type(J_Type), .B_Type(B_Type));
always begin
#10 Instruction = 32'h0001a337; //copied machine code from
#10 Instruction = 32'h02b38313; //RARS and the assembly
#10 Instruction = 32'h0263ae23; //code
#10 Instruction = 32'h0080036f;
#10 Instruction = 32'h00730663;

end
endmodule
```