CPE 233 HW 5

Wyatt Tack

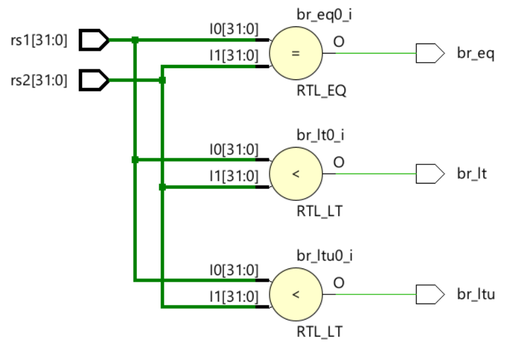
1. Behavior Description: The Branch address generator and condition generators are used to manipulate the program counter based on the different jump and branch commands available in the RISC-V assembler code. The address generator creates the new program count address based on the combination of the different immediates along with the current program count, or in case of the JALR command the register data inputted. The condition generator evaluates the two compared registers for the branch commands that would branch if the condition is met, as the condition generator gives a 1 for true and a 0 for false for the 3 needed commands: equal, less than, and less than unsigned.

   Lastly is the Memory module, which contains the stored and data that the registers read and write to, containing more referenceable memory, access to MMIO, the stack, and stores the program machine code referenced in the first address. The second address uses asynchronous read and write to manipulate the data segment, stack, and MMIO (MMIO through IO_WR and IO_IN).

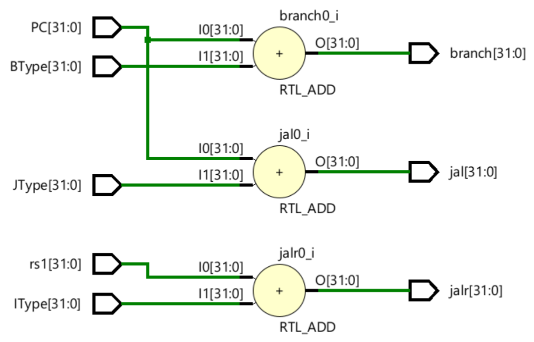
Signal Name	Size (Bits)	Purpose
MEM_CLK	1	Clock signal for synchronous actions, data stored so clock is needed
MEM_RDEN1	1	Read enable for Address 1, aka the program counter
MEM_RDEN2	1	Read enable for Adress 2, being the memory that the program executed will read from
MEM_WE2	1	Write enable for Adress 2, being the only access to alter the data stored in address 2 (Data/Stack/ MMIO)
MEM_ADDR1	14	Adress of data to output in MEM_DOUT1, connected to the program counter as input to program memory
MEM_ADDR2	32	Adress for all other data to access to either read or write from (Data/Stack/ MMIO)
MEM_DIN2	32	Data inputted to be written over address 2 data (Data/Stack/ MMIO) from write instructions
MEM_SIZE	2	Determines the size of the data to be loaded out of MEM_DOUT2, weather it be a word, byte, or half word
MEM_SIGN	1	Determines if smaller data sizes (byte/ halfword) are to be sign extended when being read to the 32-bit value MEM_DOUT2
IO_IN	32	Data stream to/from MMIO, i.e. switches and LEDs on basys board
IO_WR	1	Read/write determining bit for MMIO data, i.e. read from switches, write to LEDs
MEM_DOUT1	32	Outputs data stored at address 1, used to output machine code determined by program counter and program memory
MEM_DOUT2	32	Outputs data stored at address 2 to reg_file (Data/Stack/ MMIO)

2. Structural Design:

Branch Condition Generator:

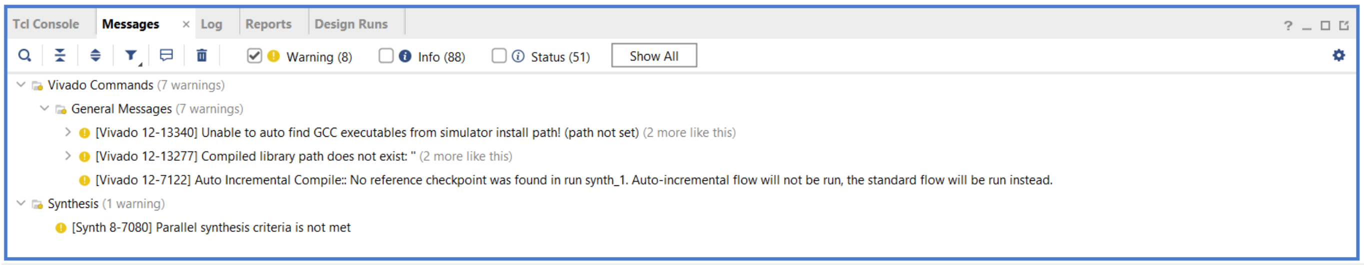


Branch Address Generator:



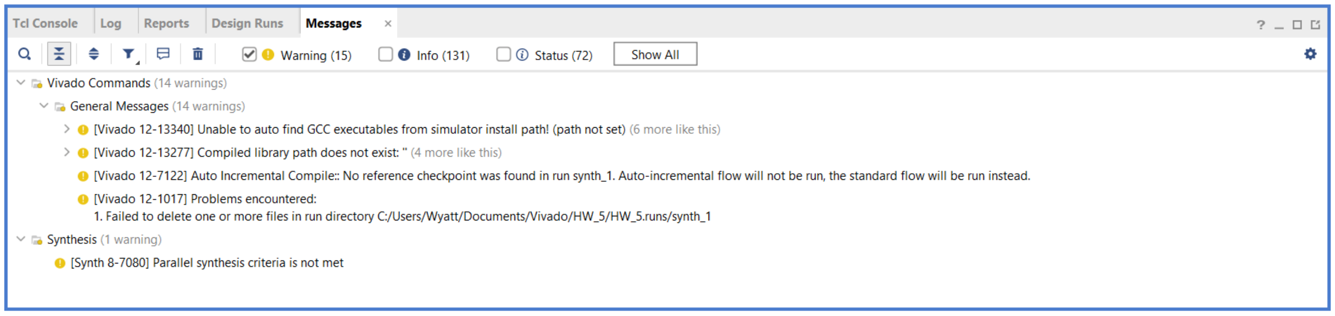
3. Synthesis Warnings Listing:

a. Branch Condition Generator:



```
Vivado Commands (7 warnings)
  General Messages (7 warnings)
    [Vivado 12-13340] Unable to auto find GCC executables from simulator install path! (path not set) (2 more like this)
    [Vivado 12-13277] Compiled library path does not exist: '' (2 more like this)
    [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.
Synthesis (1 warning)
  [Synth 8-7080] Parallel synthesis criteria is not met
```

b. Branch Address Generator



```
Vivado Commands (14 warnings)
  General Messages (14 warnings)
    [Vivado 12-13340] Unable to auto find GCC executables from simulator install path! (path not set) (6 more like this)
    [Vivado 12-13277] Compiled library path does not exist: '' (4 more like this)
    [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 12-1017] Problems encountered:
    1. Failed to delete one or more files in run directory C:/Users/Wyatt/Documents/Vivado/HW_5/HW_5.runs/synth_1
Synthesis (1 warning)
  [Synth 8-7080] Parallel synthesis criteria is not met
```

4. Verification:

a. Table for Branch Condition Generator:

Test Case Input	Test Case Output	Test Case Reasoning

rs1 = 0x00001000
rs2 = 0x10000000
	eq = 0
	lt = 1
	ltu = 1
		Tests lt and ltu

rs1 = 0x 00000100
rs2 = 0x 00000010
	eq = 0
	lt = 0
	ltu = 0
		Tests no conditions met

rs1 = 0x 02050a00
rs2 = 0x 02050a00
	eq = 1
	lt = 0
	ltu = 0
		Tests equal only

rs1 = 0x 00001000
rs2 = 0x 80010000
	eq = 0
	lt = 0
	ltu = 1
		Tests ltu only

rs1 = 0x00000000
rs2 = 0x00000000
	eq = 1
	lt = 0
	ltu = 0
		Tests eq again at 0

rs1 = 0x 40000000
rs2 = 0x 00000000
	eq = 0
	lt = 0
	ltu = 0
		Tests no conditions met

rs1 = 0x 00000000
rs2 = 0x04000000
	eq = 0
	lt = 1
	ltu = 1
		Tests lt and ltu at 0

rs1 = 0x00000000
rs2 = 0x ffffffff
	eq = 0
	lt = 0
	ltu = 1
		Tests just ltu at edge

rs1 = 0x ffffffff
rs2 = 0x00000000
	eq = 0
	lt = 1
	ltu = 0
		Tests just lt at edge

b. Table for Branch Address Generator

Test Case Input	Test Case Output	Test Case Reasoning

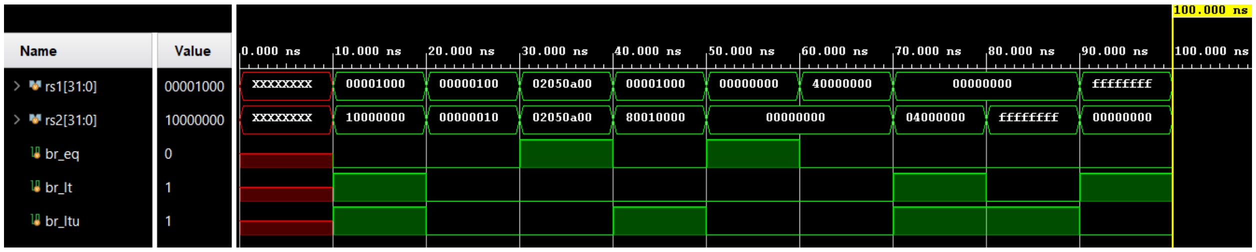
PC = 0x ffff0000
rs1 = 0x d2d2000
JType = 0x 0000a1a1
BType = 0x 0000b2b2
IType = 0x 0000c3c3
	jal = 0xffffa1a1
	branch = 0xffffb2b2
	jalr = 0x0d2dec3
		Tests standard operation

PC = 0x a1a11111
rs1 = 0x b2b22222
JType = 0x 01011111
BType = 0x 02022222
IType = 0x 03033333
	jal = 0xa2a22222
	branch = 0xa3a33333
	jalr = 0xb5b55555
		Tests standard addition again

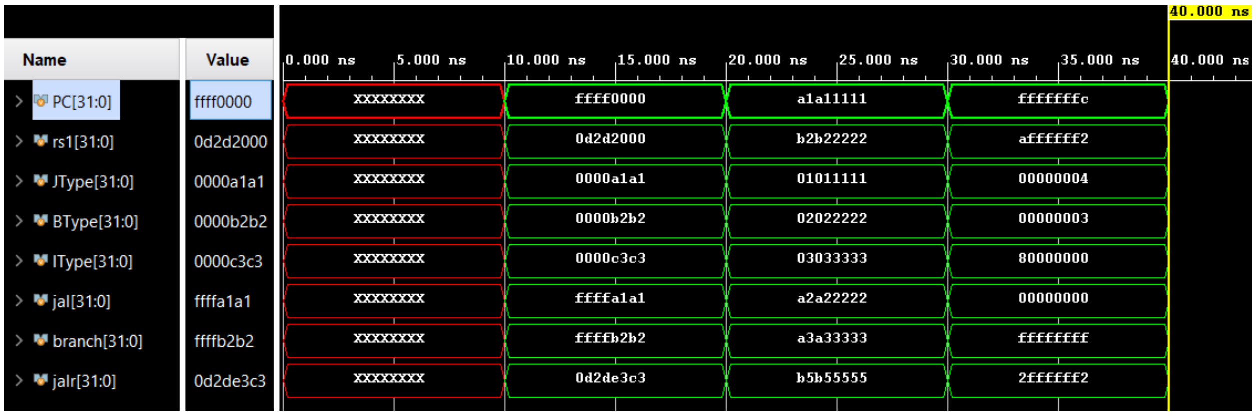
PC = 0x fffffffc
rs1 = 0x afffff2
JType = 0x 00000004
BType = 0x 00000003
IType = 0x 80000000
	jal = 0x00000000
	branch = 0xffffffff
	jalr = 0x2ffffff2
		Tests overflow and edge cases

Simulation:

- Branch Condition Generator:



- Branch Address Generator:



5. System Verilog Source Code:

For Branch Condition Generator:

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////
// Company: Cal Poly SLO
// Engineer: Wyatt Tack
//
// Create Date: 02/01/2024
// Design Name: Branch Condition Generator
// Project Name: OTTER MCU
// Target Devices: Basys 3 Board
// Description: Generates the 3 conditions between rs1 and rs2
//              being =, <, and < Signed on OTTER MCU
//
////////////////////////////////////////////////////////////
module Branch_Cond_Gen(
input [31:0] rs1, rs2,
output logic br_eq, br_lt, br_ltu
    );
always_comb //combinational block, set conditonals to binary
begin       //true/false based on evaluation:
  br_eq = (rs1 == rs2);
  br_lt = ($signed(rs1) < $signed(rs2));
  br_ltu = (rs1 < rs2);
end
endmodule
```

For Branch Condition Generator Simulation Test Bench:

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////
// Company: Cal Poly SLO
// Engineer: Wyatt Tack
//
// Create Date: 02/01/2024
// Design Name: Branch Condition Generator Simulation
// Project Name: OTTER MCU
// Target Devices: Basys 3 Board
// Description: Generates the 3 conditions between rs1 and rs2
//              being =, <, and < Signed on OTTER MCU
//              Simulation Source
//////////////////////////////////////////////////////////////
module Branch_Cond_Gen_TB();
logic [31:0] rs1, rs2;
logic br_eq, br_lt, br_ltu;
Branch_Cond_Gen UUT
(.rs1(rs1), .rs2(rs2), .br_eq(br_eq), .br_lt(br_lt), .br_ltu(br_ltu));
always begin
#10 rs1 = 32'h00001000; //lt, ltu
    rs2 = 32'h10000000;
#10 rs1 = 32'h00000100; //none
    rs2 = 32'h00000010;
#10 rs1 = 32'h02050a00; //eq
    rs2 = 32'h02050a00;
#10 rs1 = 32'h00001000; //ltu
    rs2 = 32'h80010000;
#10 rs1 = 32'h00000000; //eq
    rs2 = 32'h00000000;
#10 rs1 = 32'h40000000; //none
    rs2 = 32'h00000000;
#10 rs1 = 32'h00000000; //lt, ltu
    rs2 = 32'h04000000;
#10 rs1 = 32'h00000000; //ltu
    rs2 = 32'hffffffff;
#10 rs1 = 32'hffffffff; //lt
    rs2 = 32'h00000000;
end
endmodule
```

For Branch Address Generator:

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////
// Company: Cal Poly SLO
// Engineer: Wyatt Tack
//
// Create Date: 02/01/2024
// Design Name: Branch Address Generator
// Project Name: OTTER MCU
// Target Devices: Basys 3 Board
// Description: Generates the jumped to adress based on the
//              current PC and immediates for Jump types,
//              and RS2
//////////////////////////////////////////////////////////////
module Branch_Add_Gen(
input [31:0] PC, JType, BType, IType, rs1,
output logic [31:0] jal, branch, jalr
);
always_comb
begin                          //set next program address as what
    jal = PC + JType;          //each instruction is defined as
    branch = PC + BType;       //in assember manual
    jalr = rs1 + IType;
end
endmodule
```

For Branch Address Generator Simulation Test Bench:

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////
// Company: Cal Poly SLO
// Engineer: Wyatt Tack
//
// Create Date: 02/01/2024
// Design Name: Branch Address Generator Test Bench
// Project Name: OTTER MCU
// Target Devices: Basys 3 Board
// Description: Generates the jumped to adress based on the
//              current PC and immediates for Jump types,
//              and RS2 Simulation Source
//////////////////////////////////////////////////////////////////

module Branch_Add_Gen_TB();
logic [31:0] PC, JType, BType, IType, rs1;
logic [31:0] jal, branch, jalr;

Branch_Add_Gen UUT (.PC(PC), .JType(JType), .BType(BType),
                    .IType(IType), .rs1(rs1), .jal(jal),
                    .branch(branch), .jalr(jalr));
always
begin
#10 PC = 32'hffff0000;
    rs1 = 32'h0d2d2000;
    JType = 32'h0000a1a1; //jal = +PC
    BType = 32'h0000b2b2; //branch = +PC
    IType = 32'h0000c3c3; //jalr = +rs1
#10 PC = 32'ha1a11111;
    rs1 = 32'hb2b22222;
    JType = 32'h01011111; //jal = +PC
    BType = 32'h02022222; //branch = +PC
    IType = 32'h03033333; //jalr = +rs1
#10 PC = 32'hfffffffc;
    rs1 = 32'hafffffff2;
    JType = 32'h00000004; //jal = +PC
    BType = 32'h00000003; //branch = +PC
    IType = 32'h80000000; //jalr = +rs1
end
endmodule
```