EGE593-01: Functional Verification of Hardware Systems

**LC3 Microcontroller**

**(Fetch and Decode modules)**

|  |  |
| --- | --- |
| Group Members | **Department** |
| **Adam Secovine** | **CE** |
| **Ivan Vivar** | **CE** |

**Course Professor: Rajeev Narayanan**

**Date: 11/7/14**

Table of Contents

[Objective 3](#_Toc406488794)

[Functional Coverage 4](#_Toc406488795)

[Fetch Bugs 5](#_Toc406488796)

[Decode Bugs 7](#_Toc406488797)

[Appendix A 11](#_Toc406488798)

[Fetch 12](#_Toc406488799)

[Fetch Block Test Cases 13](#_Toc406488800)

[Case 1: Reset 13](#_Toc406488801)

[Case 2: Update PC 13](#_Toc406488802)

[Case 3: Hold PC 14](#_Toc406488803)

[Case 4: Enable Instruction Memory Read Signal 14](#_Toc406488804)

[Decode Block 15](#_Toc406488805)

[Decode Block Test Cases 16](#_Toc406488806)

[Case 1: Reset 16](#_Toc406488807)

[Case 2: npc\_in & npc\_out 16](#_Toc406488808)

[Case 3: Instr\_dout & IR 17](#_Toc406488809)

[Case 4: E\_Control - ADD 17](#_Toc406488810)

[Case 5: E\_Control -ADDi 18](#_Toc406488811)

[Case 6: E\_Control – AND 18](#_Toc406488812)

[Case 7: E\_Control – ANDi 19](#_Toc406488813)

[Case 8: E\_Control - NOT 19](#_Toc406488814)

[Case 9: E\_Control - LEA 20](#_Toc406488815)

[Case 10: W\_Control - ADD 20](#_Toc406488816)

[Case 11: W\_Control - ADDi 21](#_Toc406488817)

[Case 12: W\_Control - AND 21](#_Toc406488818)

[Case 13: W\_Control - ANDi 22](#_Toc406488819)

[Case 14: W\_Control – NOT 22](#_Toc406488820)

[Case 15: W\_Control - LEA 23](#_Toc406488821)

[Case 16: Enable\_Decode 23](#_Toc406488822)

[Appendix B 24](#_Toc406488823)

# Objective

Familiarize ourselves with the functionality and the specifications of the unpipelined LC-3 microcontroller and to conduct a functional verification on a portion of the microprocessor design. The LC-3 microcontroller consists of a Fetch, Decode, Execute and Write Back block. Our portion of focus on the LC-3 is the Fetch and Decode blocks. Although the LC-3 has a large instruction set we will only be focusing on the following six: ADD, ADDi, AND, ANDi, NOT, LEA. These instructions are relevant to the functioning of the Decode block, while the Fetch block is primarily concerned with the proper incrementing for the program counter. In order to properly test the functionality of these blocks, we first developed a detailed test plan; then, we implemented this test plan using SystemVerilog.

Procedure

As mentioned, as a first step in the verification of the Fetch and Decode modules of the LC-3, we developed a detailed test plan, in which we outline the different case to be tested. These cases were dictated by the specification provided. The full test plan of the two blocks mentioned can be seen in Appendix A. Given the limited nature of these blocks, the plan itself is primarily linear, with only minor use of random values.

Beyond the test plan, we were also interested in achieving maximum functional coverage for both of these blocks. To accomplish this, we created two cover groups, each with bins specific to the inputs to the block being tested. These cover groups can be found in the blocks’ respective test benches.

We were also concerned with properly recording the bugs found in each module, according to the particular test cases. For this, a package, containing two classes (one for each block being tested), was written. Each class references a struct which contains the inputs and outputs specific to the block being tested. As each error is encountered within the tasks of each testbench, the values of all I/Os, at the time of the error, are recorded into a new struct, and that struct is appended onto a queue of error structs.

The complete code for each testbench, the ‘testing.sv’ package, and the top modules associated with each block to be tested can be seen in Appendix B.

Results

This section includes the results of the testbenches created for the Fetch and Decode modules of the LC3 microcontroller and lists the bugs found within the respective module. The tests’ functional coverage is also listed.

## Functional Coverage

Given the limited nature of the blocks to be tested, functional coverage of 100% was expected, and achieved. Figure 1 and Figure 2 show these results, as well as the particular bins created for each covergroup.

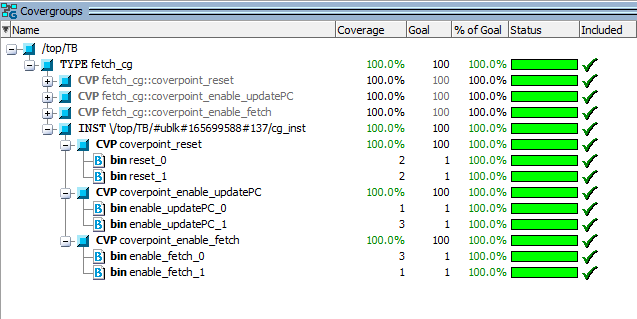


Figure 1: Functional Coverage for Fetch Block

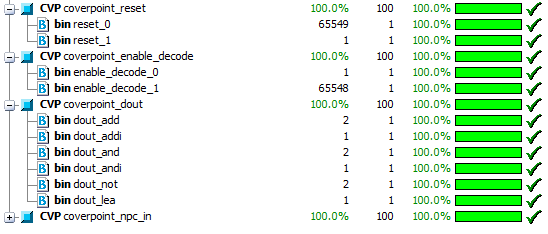


Figure 2: Functional Coverage for Decode Block

As the *npc\_in* signal (input to the Decode block) is 16-bits long, individual bins were not created, with auto-bins allowed. The overall coverpoint was measured to be 100%, along with all of the other coverpoints measured for both blocks.

## Fetch Bugs

Below, the scoreboard report of the Fetch block can be seen (Figure 4).

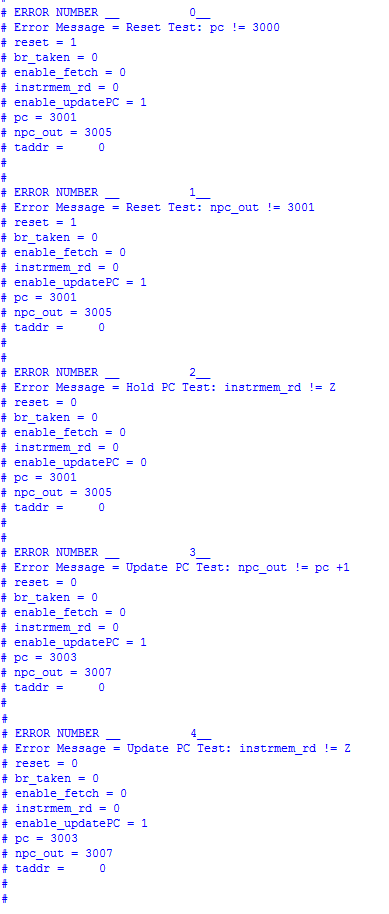


Figure 3: Fetch Scoreboard

An explanation of the bugs found can be seen below, in Table1 through Table 4.

Table 1: Fetch Reset Test Bug 1

|  |  |
| --- | --- |
| Error in: | Reset Test |
| Specification (Expected Value) | pc = 0x3000 |
| Test Bench (Received Value) | pc = 0x3001 |
| Error description: | On reset, the value of pc should be 0x3000, but is instead read as 0x3001 |

Table 2: Fetch Reset Test Bug 2

|  |  |
| --- | --- |
| Error in: | Reset Test |
| Specification (Expected Value) | npc = 0x3001 |
| Test Bench (Received Value) | npc = 0x3005 |
| Error description: | While the npc is expected to be pc + 1, it is consistently seen to be pc + 4 |

Table 3: Fetch Hold PC Test Bug

|  |  |
| --- | --- |
| Error in: | Hold PC Test |
| Specification (Expected Value) | instrmem\_rd = Z |
| Test Bench (Received Value) | instrmem\_rd = 0 |
| Error description: | The instrmem\_rd line is expected to be high-impedance (Z) when the enable\_updatePC line is held low (0), but is instead putting out a value of zero. |

Table 4: Fetch Update PC Test Bug

|  |  |
| --- | --- |
| Error in: | Update PC Test |
| Specification (Expected Value) | pc = pc + 1 |
| Test Bench (Received Value) | pc = pc + 2 |
| Error description: | Given the enable\_updatePC high, pc should be incremented by 1, but is seen to be incremented by 2. |

## Decode Bugs

Below, the scoreboard report of the Decode block can be seen (Figure 4).

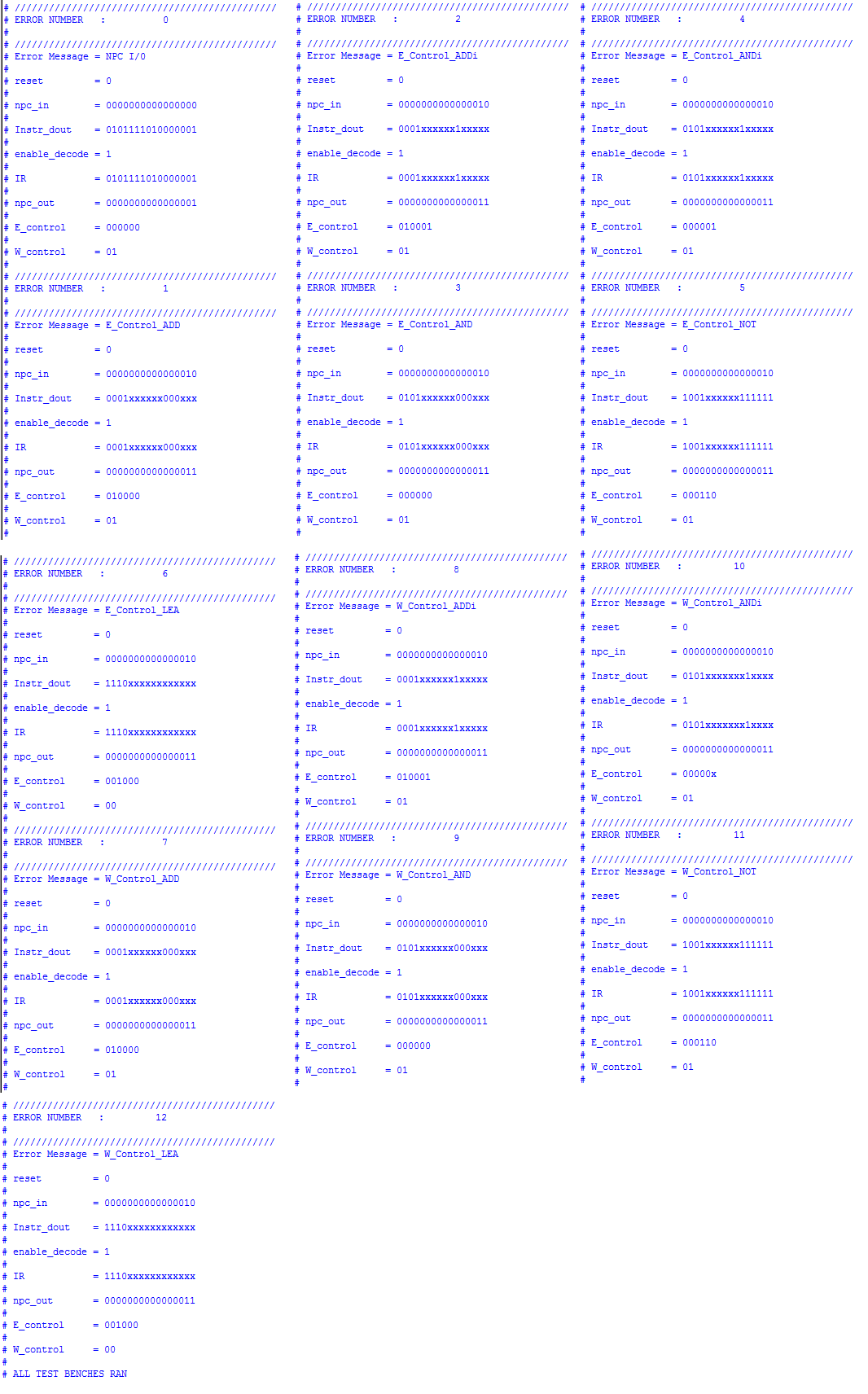


Figure 4: Decode Scoreboard

An explanation of the bugs found can be seen below, in Table 5 through Table 13.

Table 5: Decode NPC I/O Test Bug

|  |  |
| --- | --- |
| Error in: | npc\_in & npc\_out |
| Specification (Expected Value) | npc\_in = npc\_out |
| Test Bench (Received Value) | npc\_out = npc\_in + 1 |
| Error description: | npc\_in gets added a one somewhere, incrementing itself and npc\_out. |

Table 6: Decode W\_Control Test Bug 1

|  |  |
| --- | --- |
| Error in: | W\_Control – ALU Operations (ADD,ADDi,AND,ANDi,NOT) |
| Specification (Expected Value) | W\_Control = 00 |
| Test Bench (Received Value) | W\_Control = 01 |
| Error description: | W\_Control is ‘1’ instead of ‘0’ |

Table 7: Decode W\_Control Test Bug 2

|  |  |
| --- | --- |
| Error in: | W\_Control – LEA |
| Specification (Expected Value) | W\_Control = 10 |
| Test Bench (Received Value) | W\_Control = 00 |
| Error description: | W\_Control is ‘0’ instead of ‘2’ |

E\_Control is the concatenation (from MSB to LSB) of the alu\_control (2 bits), pcselect1 (2bits), pcselect2 (1 bit) and op2select (1 bit) signal. See figure below for visual description.

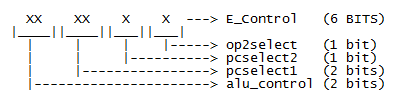


Table 8: Decode E\_Control\_ADD Test Bug

|  |  |
| --- | --- |
| Error in: | E\_Control – ADD |
| Specification (Expected Value) | E\_Control = 00XXX1 |
| Test Bench (Received Value) | E\_Control = 01XXX0 |
| Error description: | The alu\_control bits is ‘1’ instead of ‘0’  The op2select bit is ‘0’ instead of ‘1’ |

Table 9: Decode E\_Control\_ADDi Test Bug

|  |  |
| --- | --- |
| Error in: | E\_Control - ADDi |
| Specification (Expected Value) | E\_Control = 00XXX0 |
| Test Bench (Received Value) | E\_Control = 01XXX1 |
| Error description: | The alu\_control bits is ‘1’ instead of ‘0’  The op2select bit is ‘1’ instead of ‘0’ |

Table 10: Decode E\_Control\_AND Test Bug

|  |  |
| --- | --- |
| Error in: | E\_Control – AND |
| Specification (Expected Value) | E\_Control = 01XXX1 |
| Test Bench (Received Value) | E\_Control = 00XXX0 |
| Error description: | The alu\_control bits is ‘0’ instead of ‘1’  The op2select bit is ‘0’ instead of ‘1’ |

Table 11: Decode E\_Control\_ANDi Test Bug

|  |  |
| --- | --- |
| Error in: | E\_Control - ANDi |
| Specification (Expected Value) | E\_Control = 01XXX1 |
| Test Bench (Received Value) | E\_Control = 00XXX0 |
| Error description: | The alu\_control bits is ‘0’ instead of ‘1’  The op2select bit is ‘0’ instead of ‘1’ |

Table 12: Decode E\_Control\_NOT Test Bug

|  |  |
| --- | --- |
| Error in: | E\_Control - NOT |
| Specification (Expected Value) | E\_Control = 10XXXX |
| Test Bench (Received Value) | E\_Control = 00XXXX |
| Error description: | The alu\_control bits is ‘0’ instead of ‘2’ |

Table 13: Decode E\_Control\_LEA Test Bug

|  |  |
| --- | --- |
| Error in: | E\_Control - LEA |
| Specification (Expected Value) | E\_Control = XX011X |
| Test Bench (Received Value) | E\_Control = XX100X |
| Error description: | The alu\_control bits is ‘0’ instead of ‘2’ |

Conclusion

Fortunately, finding the bugs within these blocks was relatively straight forward. Creating a detailed test plan prior to writing our code greatly improved the speed with which we were able to complete the test bench, and ensured that we addressed all aspects of the specification thoroughly. By using SystemVerilog object-oriented functionality, such as packages, classes, covergroups, functions, and tasks, we were able to keep the code succinct and functionally compartmentalized. Given the ever increasing complexity and variety of today’s logic designs, the ability to properly verify the specifications of said designs is an increasingly important piece of digital hardware design, making the skills gained in this project highly valuable in today’s market.

# Appendix A

LC-3

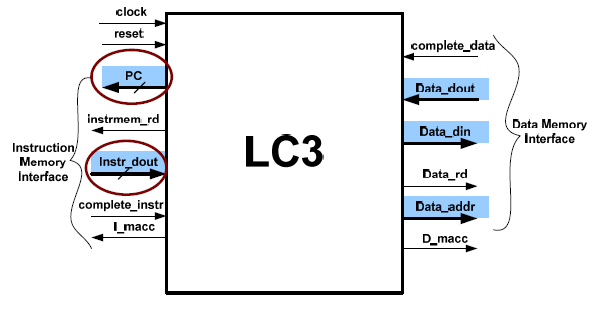


Figure 1: Top Level Block Diagram of LC-3

The input and output pins of the Design Under Test (DUT), LC-3 are listed below.

* **Inputs**
  + **clock** (1 bit)
  + **reset** (1 bit)
  + **complete\_data** (1 bit)
  + **complete\_instr** (1 bit)
  + **Instr\_dout** (16 bits): Carries the Instruction from the Instruction Memory interface into the LC-3.
  + Data\_dout (Ignore for this project)
* **Outputs**
  + **PC** (16 bits): The Program Counter contains the addres to the Instruction Memory.
  + **instrmem\_rd** (1 bit): The read enable signal for the Instruction Memory. Fetches an instruction.
  + Data\_addr (Ignored in this project)
  + **Data\_din** (16 bit):
  + **Data\_rd** (1 bit)
  + I\_macc (Ignored in this project)
  + D\_macc (Ignored in this project)

# Fetch

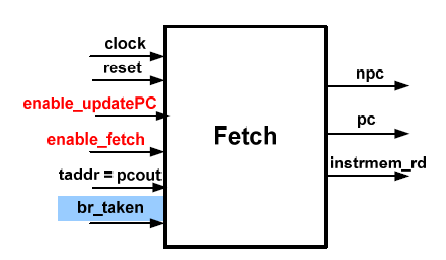


Figure 2: Top Level Block Diagram for Fetch

The inputs and output pins of the **Fetch Block** are listed below:

* **Inputs**
  + **clock** (1 bit)
  + **reset** (1 bit)
  + **enable\_updatePC** (1 bit): When 1’b1 the PC changes at the positive edge of the clock to PC+1 or taddr based on br\_taken.
  + **enable\_fetch** (1 bit): When 1’b1 the fetching (reading) of the Instruction Memory is conducted.
  + br\_taken (Ignored in this project)
  + taddr (Ignored for this project)
* **Outputs**
  + **intrmem\_rd** (1 bit): An asynchronous signal enabled when *enable\_fetch* is 1’b1 and it tells the Memory that a read rather than a write is to be conducted
  + **pc** (16 bits): The Program Counter’s current value.
  + **nps** (16 bits): Is an asynchronous signal that is equal to PC+1.

# Fetch Block Test Cases

This section introduces the list of test cases for the Fetch module which will later allow us to build test benches off of.

## Case 1: Reset

Purpose: Test of the reset functionality. Inputs set as “X” will be provided random values over multiple clock cycles.

**Inputs Signals**

* Clock: Running
* Reset: 1’b1
* enable\_updatePC: 1’bX
* enable\_fetch: 1’bX

**Expected Outputs - T1**

* pc: 16’h3000
* npc: 16’h3001
* instrmem\_rd: 1’bX

## Case 2: Update PC

Purpose: To test the normal operation of the block, without reset, with fetch disabled.

**Inputs Signals**

* Clock: Running
* Reset: 1’b0
* enable\_updatePC: 1’b1
* enable\_fetch: 1’b0

**Expected Outputs - T0**

* pc: 16’h3000
* npc: 16’h3001
* instrmem\_rd: 1’bZ

**Expected Outputs - T1**

* pc: 16’h3001
* npc: 16’h3002
* instrmem\_rd: 1’bZ

**Expected Outputs - T2**

* pc: 16’h3002
* npc: 16’h3003
* instrmem\_rd: 1’bZ

## Case 3: Hold PC

Purpose: To test whether the block will hold the same pc and npc values when enable\_updatePC is disabled.

**Inputs Signals**

* Clock: Running
* Reset: 1’b0
* enable\_updatePC: 1’b0
* enable\_fetch: 1’b0

**Expected Outputs - T0**

* pc: 16’h3000
* npc: 16’h3001
* instrmem\_rd: 1’bZ

**Expected Outputs - T1**

* pc: 16’h3000
* npc: 16’h3001
* instrmem\_rd: 1’bZ

**Expected Outputs - T2**

* pc: 16’h3000
* npc: 16’h3001
* instrmem\_rd: 1’bZ

## Case 4: Enable Instruction Memory Read Signal

Purpose: The test whether the instrmem\_rd signal will output HIGH when enabled.

**Inputs Signals**

* Clock: Running
* Reset: 1’bX
* enable\_updatePC: 1’bX
* enable\_fetch: 1’b1

**Expected Outputs - T0**

* pc: 16’h3000
* npc: 16’h3001
* instrmem\_rd: 1’b1

# Decode Block

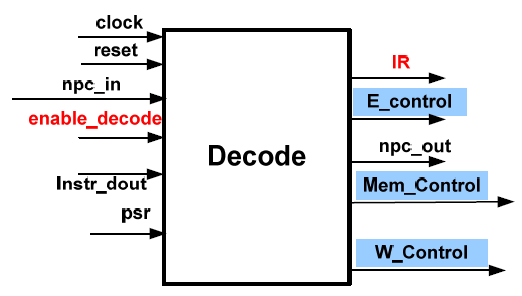


Figure 3: Top Level Block Diagram of Decode

The Input and output pins of the **Decode Block** are listed below:

* **Inputs**
  + **clock** (1 bit)
  + **reset** (1 bit)
  + **npc\_in** (16 bit): The npc value taken from the Fetch block and is passed along the pipeline to its final consuming block.
  + **Instr\_dout** (16 bit): Carries the Instruction from the Instruction Memory
  + **enable\_decode** (1 bit): When 1’b1 it allows for the operation of the decode unit in normal mode where it creates the relevant control signals at the output based on the input from the Instruction Memory. If it is 1’b0 then the Decode block stalls and does not change the output.
  + psr (Ignore for this project)
* **Outputs**
  + **IR** (16 bits): equal to *Instr\_dout*.
  + **npc\_out** (16 bits): equal to *npc\_in*.
  + **E\_control** (6 bits): This signal controls the Execute Block.
  + **W\_control** (2 bits): This signal controls the Writeback Block and decides where to write to the *RegFile*.
  + Mem\_control (Ignored for this project)

# Decode Block Test Cases

This section introduces the list of test cases for the Decode module which will later allow us to build test benches off of.

## Case 1: Reset

Purpose: Test of the reset functionality

* **Inputs**
  + clock: Running
  + reset: 1’b1
  + npc\_in: 16’bX
  + Instr\_dout: 16’bX
  + enable\_decode: 1’bX

* **Outputs** 
  + IR: 1’b0
  + npc\_out: 1’b0
  + E\_control: 1’b0
  + W\_control: 1’b0

## Case 2: npc\_in & npc\_out

Purpose: In this case we are assuming the input npc\_in and npc\_out are the same value. We are testing that the enable bit controlled by the enable\_decode signal is enabled and the npc\_out value holds the previous value of npc\_in.

* **Inputs**
  + clock: Running
  + reset: 1’b0
  + npc\_in: 16’h1010
  + Instr\_dout: 16’hXXXX
  + enable\_decode: 1’b1

* **Outputs** 
  + IR: 16’hXXXX
  + npc\_out: 16’h1010
  + E\_control: 6’bXXXXXX
  + W\_control: 1’bX

## Case 3: Instr\_dout & IR

Purpose: In this case we are assuming the input Instr\_dout and IR are the same value. We are testing that the enable bit controlled by the enable\_decode signal is disabled and the Instr\_dout holds the previous value of IR.

* **Inputs**
  + clock: Running
  + reset: 1’b0
  + npc\_in: 16’hXXXX
  + Instr\_dout: 16’h1010
  + enable\_decode: 1’b1

* **Outputs** 
  + IR: 16’h1010
  + npc\_out: 16’hXXXX
  + E\_control: 6’bXXXXXX
  + W\_control: 2’bXX

## Case 4: E\_Control - ADD

Purpose: To test that the right combination of the IR[15:12] and IR[5] bits within Instr\_dout[15:0] to produce the right E\_Control combination to produce ADD.

* **Inputs**
  + clock: Running
  + reset: 1’b0
  + npc\_in: 16’hXXXX
  + Instr\_dout: 16’b**0001**XXXXXX**0**00XXX
  + enable\_decode: 1’b1

* **Outputs** 
  + IR: 16’b**0001**XXXXXX**0**00XXX
  + npc\_out: 16’hXXXX
  + E\_control: 6’b**00**XXX**1**
  + W\_control: 2’bXX

## Case 5: E\_Control -ADDi

Purpose: To test that the right combination of the IR[15:12] and IR[5] bits within Instr\_dout[15:0] to produce the right E\_Control combination to produce ADD immediate.

* **Inputs**
  + clock: Running
  + reset: 1’b0
  + npc\_in: 16’hXXXX
  + Instr\_dout: 16’b**0001**XXXXXX**1**XXXXX
  + enable\_decode: 1’b1

* **Outputs** 
  + IR: 16’b**0001**XXXXXX**1**XXXXX
  + npc\_out: 16’hXXXX
  + E\_control: 6’b**00**XXX**0**
  + W\_control: 2’bXX

## Case 6: E\_Control – AND

Purpose: To test that the right combination of the IR[15:12] and IR[5] bits within Instr\_dout[15:0] to produce the right E\_Control combination to produce AND.

* **Inputs**
  + clock: Running
  + reset: 1’b0
  + npc\_in: 16’hXXXX
  + Instr\_dout: 16’b**0101**XXXXXX**0**00XXX
  + enable\_decode: 1’b1

* **Outputs** 
  + IR: 16’b**0101**XXXXXX**0**00XXX
  + npc\_out: 16’hXXXX
  + E\_control: 6’b**01**XXX1
  + W\_control: 2’bXX

## Case 7: E\_Control – ANDi

Purpose: To test that the right combination of the IR[15:12] and IR[5] bits within Instr\_dout[15:0] to produce the right E\_Control combination to produce AND immediate.

* **Inputs**
  + clock: Running
  + reset: 1’b0
  + npc\_in: 16’hXXXX
  + Instr\_dout: 16’b**0101**XXXXXX**1**XXXXX
  + enable\_decode: 1’b1

* **Outputs** 
  + IR: 16’b**0101**XXXXXX**1**XXXXX
  + npc\_out: 16’hXXXX
  + E\_control: 6’b**01**XXX**0**
  + W\_control: 2’bXX

## Case 8: E\_Control - NOT

Purpose: To test that the right combination of the IR[15:12] and IR[5] bits within Instr\_dout[15:0] to produce the right E\_Control combination to produce NOT immediate.

* **Inputs**
  + clock: Running
  + reset: 1’b0
  + npc\_in: 16’hXXXX
  + Instr\_dout: 16’b**1001**XXXXXX**111111**
  + enable\_decode: 1’b1

* **Outputs** 
  + IR: 16’b**1001**XXXXXX**111111**
  + npc\_out: 16’hXXXX
  + E\_control: 6’b**10**XXX**X**
  + W\_control: 2’bXX

## Case 9: E\_Control - LEA

Purpose: To test that the right combination of the IR[15:12] and IR[5] bits within Instr\_dout[15:0] to produce the right E\_Control combination to produce LEA immediate.

* **Inputs**
  + clock: Running
  + reset: 1’b0
  + npc\_in: 16’hXXXX
  + Instr\_dout: 16’b**1110**XXXXXXXXXXXX
  + enable\_decode: 1’b1

* **Outputs** 
  + IR: 16’b**1110**XXXXXXXXXXXX
  + npc\_out: 16’hXXXX
  + E\_control: 6’bXX**111**X
  + W\_control: 2’bXX

## Case 10: W\_Control - ADD

Purpose: To test that the right combination of the IR[15:12] bits and MODE bit produces the right W\_Control combination to produce ADD.

* **Inputs**
  + clock: Running
  + reset: 1’b0
  + npc\_in: 16’hXXXX
  + Instr\_dout: 16’b**0001**XXXXXXX**0**00XXX
  + enable\_decode: 1’b1

* **Outputs** 
  + IR: 16’b**0001**XXXXXXX**0**00XXX
  + npc\_out: 16’hXXXX
  + E\_control: 16’hXXXX
  + W\_control: 2’b**00**

## Case 11: W\_Control - ADDi

Purpose: To test that the right combination of the IR[15:12] bits and MODE bit produces the right W\_Control combination to produce ADD immediate.

* **Inputs**
  + clock: Running
  + reset: 1’b0
  + npc\_in: 16’hXXXX
  + Instr\_dout: 16’b**0001**XXXXXX**1**XXXXX
  + enable\_decode: 1’b1

* **Outputs** 
  + IR: 16’b**0001**XXXXXX**1**XXXXX
  + npc\_out: 16’hXXXX
  + E\_control: 16’hXXXX
  + W\_control: 2’b**00**

## Case 12: W\_Control - AND

Purpose: To test that the right combination of the IR[15:12] bits and MODE bit produces the right W\_Control combination to produce AND.

* **Inputs**
  + clock: Running
  + reset: 1’b0
  + npc\_in: 16’hXXXX
  + Instr\_dout: 16’b**0101**XXXXXX000XXX
  + enable\_decode: 1’b1

* **Outputs** 
  + IR: 16’b**0101**XXXXXX000XXX
  + npc\_out: 16’hXXXX
  + E\_control: 16’hXXXX
  + W\_control: 2’b**00**

## Case 13: W\_Control - ANDi

Purpose: To test that the right combination of the IR[15:12] bits and MODE bit produces the right W\_Control combination to produce AND immediate.

* **Inputs**
  + clock: Running
  + reset: 1’b0
  + npc\_in: 16’hXXXX
  + Instr\_dout: 16’b**0101**XXXXXX**1**XXXXX
  + enable\_decode: 1’b1

* **Outputs** 
  + IR: 16’b**0101**XXXXXX**1**00XXX
  + npc\_out: 16’hXXXX
  + E\_control: 16’hXXXX
  + W\_control: 2’b**00**

## Case 14: W\_Control – NOT

Purpose: To test that the right combination of the IR[15:12] bits and MODE bit produces the right W\_Control combination to produce AND immediate.

* **Inputs**
  + clock: Running
  + reset: 1’b0
  + npc\_in: 16’hXXXX
  + Instr\_dout: 16’b**1001**XXXXXX**111111**
  + enable\_decode: 1’b1

* **Outputs** 
  + IR: 16’b**1001**XXXXXX**111111**
  + npc\_out: 16’hXXXX
  + E\_control: 16’hXXXX
  + W\_control: 2’b**00**

## Case 15: W\_Control - LEA

Purpose: To test that the right combination of the IR[15:12] bits and MODE bit produces the right W\_Control combination to produce AND immediate.

* **Inputs**
  + clock: Running
  + reset: 1’b0
  + npc\_in: 16’hXXXX
  + Instr\_dout: 16’b**1110**XXXXXXX**XXXXXX**
  + enable\_decode: 1’b1

* **Outputs** 
  + IR: 16’b**1110**XXXXXXX**XXXXXX**
  + npc\_out: 16’hXXXX
  + E\_control: 16’hXXXX
  + W\_control: 2’b**10**

## Case 16: Enable\_Decode

Purpose: The purpose of this test is to verify that the decode module does not create outputs when it is disabled. A random number will be passed to npc\_in and instr\_dout. The outputs IR and npc\_out will be checked and should not equal npc\_in and instr\_dout.

* **Inputs**
  + clock: Running
  + reset: 1’b0
  + npc\_in: 16’hXXXX
  + Instr\_dout: 16’b**1110**XXXXXXX**XXXXXX**
  + enable\_decode: 1’b1

* **Outputs** 
  + IR: 16’b**1110**XXXXXXX**XXXXXX**
  + npc\_out: 16’hXXXX
  + E\_control: 16’hXXXX
  + W\_control: 2’bXX

# Appendix B

/\*testing.sv\*/

package testing;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*STRUCT FOR FETCH ERRORS\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

typedef struct {

string er\_msg;

reg reset;

reg br\_taken;

reg enable\_fetch;

reg instrmem\_rd;

reg enable\_updatePC;

reg [15:0] taddr;

reg [15:0] pc;

reg [15:0] npc\_out;

} fetch\_struct;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*STRUCT FOR DECODE ERRORS\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

typedef struct{

string er\_msg;

reg reset;

reg [15:0] npc\_in;

reg [15:0] Instr\_dout;

reg enable\_decode;

reg [15:0] IR;

reg [15:0] npc\_out;

reg [5:0] E\_control;

reg [1:0] W\_control;

} decode\_struct;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*CLASS FOR RECORDING DECODE ERRORS\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

class decode\_error\_rec;

integer unsigned error\_count = 0;

decode\_struct ds;

decode\_struct error\_queue[$];

function void incr\_error(

string er\_msg,

input reset,

input [15:0] npc\_in,

input [15:0] Instr\_dout,

input enable\_decode,

input [15:0] IR,

input [15:0] npc\_out,

input [5:0] E\_control,

input [1:0] W\_control

);

ds = '{er\_msg,reset,npc\_in,Instr\_dout,enable\_decode,IR,npc\_out,E\_control,W\_control};

error\_queue.push\_back(ds);

error\_count = (error\_count + 1);

endfunction

function void report\_errors();

foreach(error\_queue[i]) begin

$display("//////////////////////////////////////////////");

$display("ERROR NUMBER :%d \n",i);

$display("//////////////////////////////////////////////");

$display("Error Message = %s \n",error\_queue[i].er\_msg);

$display("reset = %b \n",error\_queue[i].reset);

$display("npc\_in = %b \n",error\_queue[i].npc\_in);

$display("Instr\_dout = %b \n",error\_queue[i].Instr\_dout);

$display("enable\_decode = %b \n",error\_queue[i].enable\_decode);

$display("IR = %b \n",error\_queue[i].IR);

$display("npc\_out = %b \n",error\_queue[i].npc\_out);

$display("E\_control = %b \n",error\_queue[i].E\_control);

$display("W\_control = %b \n",error\_queue[i].W\_control);

end

endfunction

endclass //END decode\_error\_rec()

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*CLASS FOR RECORDING FETCH ERRORS\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

class error\_rec;

integer unsigned error\_count = 0;

fetch\_struct fs;

fetch\_struct error\_queue[$];

function void incr\_error(

string er\_msg,

input reset,

input br\_taken,

input enable\_fetch,

input instrmem\_rd,

input enable\_updatePC,

input [15:0] pc,

input [15:0] npc\_out,

input [15:0] taddr

);

string error\_string = string'(er\_msg);

fs = '{er\_msg,reset,br\_taken,enable\_fetch,instrmem\_rd,enable\_updatePC,taddr,pc,npc\_out};

error\_queue.push\_back(fs);

error\_count = (error\_count + 1);

endfunction

function void report\_errors();

foreach(error\_queue[i]) begin

$display("ERROR NUMBER \_\_%d\_\_",i);

$display("Error Message = %s",error\_queue[i].er\_msg);

$display("reset = %d",error\_queue[i].reset);

$display("br\_taken = %d",error\_queue[i].br\_taken);

$display("enable\_fetch = %d",error\_queue[i].enable\_fetch);

$display("instrmem\_rd = %d",error\_queue[i].instrmem\_rd);

$display("enable\_updatePC = %d",error\_queue[i].enable\_updatePC);

$display("pc = %h",error\_queue[i].pc);

$display("npc\_out = %h",error\_queue[i].npc\_out);

$display("taddr = %d\n\n",error\_queue[i].taddr);

end

endfunction

endclass //END error\_rec()

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

endpackage

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*fetch\_test.sv\*/

`timescale 1ns/1ns

`include "C:/Users/n02549032/Desktop/Full\_Code\_Fetch\_Decode/Full\_Code\_Fetch\_Decode/testing.sv"

import testing::\*;

module fetch\_test(

output logic reset,

output logic br\_taken,

output logic enable\_fetch,

output logic enable\_updatePC,

output logic [15:0] taddr,

input bit [15:0] pc,

input bit [15:0] npc\_out,

input logic instrmem\_rd,

input bit clock

);

integer last\_pc;

//COVERGROUP \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

covergroup fetch\_cg;

coverpoint\_reset : coverpoint reset{

bins reset\_0 = {0};

bins reset\_1 = {1};

}

coverpoint\_enable\_updatePC : coverpoint enable\_updatePC{

bins enable\_updatePC\_0 = {0};

bins enable\_updatePC\_1 = {1};

}

coverpoint\_enable\_fetch : coverpoint enable\_fetch{

bins enable\_fetch\_0 = {0};

bins enable\_fetch\_1 = {1};

}

endgroup

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//RESET \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

task reset\_task(

input enable\_fetch\_t,

input enable\_updatePC\_t,

fetch\_cg cg\_inst,

error\_rec fetch\_er

);

reset <= 1'b1;

enable\_fetch <= enable\_fetch\_t;

enable\_updatePC <= enable\_updatePC\_t;

#10;

cg\_inst.sample();

// if((pc != 16'h3000)||(npc\_out != 16'h3001))begin

if(pc != 16'h3000)begin

fetch\_er.incr\_error("Reset Test: pc != 3000",reset,br\_taken,enable\_fetch,instrmem\_rd,enable\_updatePC,pc,npc\_out,16'h0000);

$display("Reset Test Failed\n");

end

if(npc\_out != 16'h3001)begin

fetch\_er.incr\_error("Reset Test: npc\_out != 3001",reset,br\_taken,enable\_fetch,instrmem\_rd,enable\_updatePC,pc,npc\_out,16'h0000);

$display("Reset Test Failed\n");

end

endtask

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//UPDATE PC \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

task update\_pc\_task(

fetch\_cg cg\_inst,

error\_rec fetch\_er

);

reset <= 1'b0;

enable\_updatePC <= 1'b1;

enable\_fetch <= 1'b0;

#10;

cg\_inst.sample();

if(npc\_out != (pc + 1))begin

fetch\_er.incr\_error("Update PC Test: npc\_out != pc +1",reset,br\_taken,enable\_fetch,instrmem\_rd,enable\_updatePC,pc,npc\_out,16'h0000);

$display("Update PC Test Failed\n");

end

if((instrmem\_rd == 1) || (instrmem\_rd == 0))begin

fetch\_er.incr\_error("Update PC Test: instrmem\_rd != Z",reset,br\_taken,enable\_fetch,instrmem\_rd,enable\_updatePC,pc,npc\_out,16'h0000);

$display("Update PC Test Failed\n");

end

endtask

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//HOLD PC \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

task hold\_pc\_task(

fetch\_cg cg\_inst,

error\_rec fetch\_er

);

last\_pc <= pc;

//$display("last\_pc = %d",last\_pc);

reset <= 1'b0;

enable\_updatePC <= 1'b0;

enable\_fetch <= 1'b0;

#10;

cg\_inst.sample();

$display("pc = %d",pc);

if(pc != last\_pc)begin

fetch\_er.incr\_error("Hold PC Test: pc != last\_pc",reset,br\_taken,enable\_fetch,instrmem\_rd,enable\_updatePC,pc,npc\_out,16'h0000);

$display("Hold PC Test Failed\n");

end

if((instrmem\_rd == 1) || (instrmem\_rd == 0))begin

fetch\_er.incr\_error("Hold PC Test: instrmem\_rd != Z",reset,br\_taken,enable\_fetch,instrmem\_rd,enable\_updatePC,pc,npc\_out,16'h0000);

$display("Hold PC Test Failed\n");

end

endtask

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//ENABLE INSTRUCTION MEMORY READ SIGNAL \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

task instr\_mem\_rd\_task(

fetch\_cg cg\_inst,

error\_rec fetch\_er,

input reset\_t,

input enable\_updatePC\_t

);

reset <= reset\_t;

enable\_updatePC <= enable\_updatePC\_t;

enable\_fetch <= 1'b1;

#10;

cg\_inst.sample();

if(instrmem\_rd != 1'b1)begin

fetch\_er.incr\_error("Enable Instr Mem Read Test: instrmem\_rd != 1",reset,br\_taken,enable\_fetch,instrmem\_rd,enable\_updatePC,pc,npc\_out,16'h0000);

$display("Instruction Memory Read Test Failed\n");

end

endtask

//INITIAL \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

initial begin

error\_rec fetch\_er;

fetch\_cg cg\_inst;

fetch\_er = new();

cg\_inst = new();

br\_taken <= 1'b0;

reset\_task($random,$random,cg\_inst,fetch\_er);

hold\_pc\_task(cg\_inst,fetch\_er);

update\_pc\_task(cg\_inst,fetch\_er);

instr\_mem\_rd\_task(cg\_inst,fetch\_er,$random,$random);

$display("The following errors were found:\n");

fetch\_er.report\_errors();

end

endmodule

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*DecodeTB.sv\*/

//`include "C:/Users/n02549032/Desktop/Fetch\_Code/testing.sv"

import testing::\*;

module DecodeTB( input bit clock,

input logic [1:0] W\_Control,

input logic [5:0] E\_Control,

input logic [15:0] IR,

input logic [15:0] npc\_out,

input logic Mem\_Control,

output logic reset,

output logic enable\_decode,

output logic [15:0] dout,

output logic [15:0] npc\_in);

covergroup decode\_covergroup;

coverpoint\_reset : coverpoint reset{

bins reset\_0 = {0};

bins reset\_1 = {1};

}

coverpoint\_enable\_decode : coverpoint enable\_decode{

bins enable\_decode\_0 = {0};

bins enable\_decode\_1 = {1};

}

coverpoint\_dout : coverpoint dout{

bins dout\_add = {16'b0001XXXXXX000XXX};

bins dout\_addi = {16'b0001XXXXXX1XXXXX};

bins dout\_and = {16'b0101XXXXXX000XXX};

bins dout\_andi = {16'b0101XXXXXX1XXXXX};

bins dout\_not = {16'b1001XXXXXX111111};

bins dout\_lea = {16'b1110XXXXXXXXXXXX};

}

coverpoint\_npc\_in : coverpoint npc\_in{

}

endgroup

initial begin

decode\_covergroup decode\_cg;

decode\_error\_rec dec\_er;

decode\_cg = new();

dec\_er = new();

RST(decode\_cg,dec\_er);

NPCI\_O(decode\_cg,dec\_er);

INSTR\_IR(decode\_cg,dec\_er);

ED\_Bit(decode\_cg,dec\_er);

EC\_ADD(decode\_cg,dec\_er);

EC\_ADDi(decode\_cg,dec\_er);

EC\_AND(decode\_cg,dec\_er);

EC\_ANDi(decode\_cg,dec\_er);

EC\_NOT(decode\_cg,dec\_er);

EC\_LEA(decode\_cg,dec\_er);

WC\_ADD(decode\_cg,dec\_er);

WC\_ADDi(decode\_cg,dec\_er);

WC\_AND(decode\_cg,dec\_er);

WC\_ANDi(decode\_cg,dec\_er);

WC\_NOT(decode\_cg,dec\_er);

WC\_LEA(decode\_cg,dec\_er);

dec\_er.report\_errors();

$display("ALL TEST BENCHES RAN");

end

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*RESET\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

task RST(decode\_covergroup decode\_cg,decode\_error\_rec dec\_er);

reset = 1'b1;

#10

npc\_in <= $random;

dout <= $random;

enable\_decode <= $random;

decode\_cg.sample();

#20

if (IR != 1'b0 || npc\_out != 1'b0 || E\_Control != 1'b0 || W\_Control != 1'b0) begin

dec\_er.incr\_error("RESET",reset,npc\_in,dout,enable\_decode,IR,npc\_out,E\_Control,W\_Control);

end

reset = 1'b0;

endtask

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*NPC I/0\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

task NPCI\_O(decode\_covergroup decode\_cg,decode\_error\_rec dec\_er);

static int i = 0;

enable\_decode = 1'b1;

#10

repeat(65535)begin

npc\_in <= i;

decode\_cg.sample();

#20

if (npc\_in != npc\_out)

begin

dec\_er.incr\_error("NPC I/0",reset,npc\_in,dout,enable\_decode,IR,npc\_out,E\_Control,W\_Control);

end

i = i + 1;

end

endtask

//\*\*\*\*\*\*\*\*\*\*\*\*Instr\_dout = dout & IR\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

task INSTR\_IR(decode\_covergroup decode\_cg,decode\_error\_rec dec\_er);

enable\_decode = 1'b1;

#10

dout <= $random;

decode\_cg.sample();

#20

if (dout != IR)

begin

dec\_er.incr\_error("Instr\_dout",reset,npc\_in,dout,enable\_decode,IR,npc\_out,E\_Control,W\_Control);

end

else

begin

$display("---------------------------------------------\n");

$display("Dout & IR is good");

$display("---------------------------------------------\n");

end

endtask

//\*\*\*\*\*\*\*\*\*\*\*\*\*Enable\_Decode\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\* enable the enable bit, pass a value for npc\_in and dout and then turn off the enable bit

pass it another value and then turn it on again.

\*/

task ED\_Bit(decode\_covergroup decode\_cg,decode\_error\_rec dec\_er);

enable\_decode <= 1'b1;

npc\_in <= 16'h0000;

dout <= 16'hFFFF;

decode\_cg.sample();

#20

enable\_decode = 1'b0;

#10

npc\_in <= 16'h0002;

dout <= 16'h0000;

decode\_cg.sample();

#20

if((dout == IR)||(npc\_in == npc\_out))begin

dec\_er.incr\_error("Enable\_Decode",reset,npc\_in,dout,enable\_decode,IR,npc\_out,E\_Control,W\_Control);

end

endtask

//\*\*\*\*\*\*\*\*\*\*\*\*\*E\_Control\_ADD\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

task EC\_ADD(decode\_covergroup decode\_cg,decode\_error\_rec dec\_er);

//repeat(5)

//begin

enable\_decode = 1'b1;

#10

dout <= 16'b0001XXXXXX000XXX;

decode\_cg.sample();

#20

if(E\_Control != 6'b00XXX1 )

begin

dec\_er.incr\_error("E\_Control\_ADD",reset,npc\_in,dout,enable\_decode,IR,npc\_out,E\_Control,W\_Control);

end

else

begin

$display("---------------------------------------------\n");

$display("E\_Control\_ADD is good");

$display("---------------------------------------------\n");

end

//end

endtask

//\*\*\*\*\*\*\*\*\*\*\*\*E\_Control\_ADDi\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

task EC\_ADDi(decode\_covergroup decode\_cg,decode\_error\_rec dec\_er);

//repeat(5)

//begin

enable\_decode = 1'b1;

#10

dout <= 16'b0001XXXXXX1XXXXX;

decode\_cg.sample();

#20

if(E\_Control != 6'b00XXX0 )

begin

dec\_er.incr\_error("E\_Control\_ADDi",reset,npc\_in,dout,enable\_decode,IR,npc\_out,E\_Control,W\_Control);

end

else

begin

$display("---------------------------------------------\n");

$display("E\_Control\_ADDi is good");

$display("---------------------------------------------\n");

end

//end

endtask

//\*\*\*\*\*\*\*\*\*\*\*\*E\_Control\_AND\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

task EC\_AND(decode\_covergroup decode\_cg,decode\_error\_rec dec\_er);

enable\_decode = 1'b1;

#10

dout <= 16'b0101XXXXXX000XXX;

decode\_cg.sample();

#20

if(E\_Control != 6'b01XXX1)

begin

dec\_er.incr\_error("E\_Control\_AND",reset,npc\_in,dout,enable\_decode,IR,npc\_out,E\_Control,W\_Control);

end

else

begin

$display("---------------------------------------------\n");

$display("E\_Control\_AND is good");

$display("---------------------------------------------\n");

end

endtask

//\*\*\*\*\*\*\*\*\*\*\*\*E\_Control\_ANDi\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

task EC\_ANDi(decode\_covergroup decode\_cg,decode\_error\_rec dec\_er);

enable\_decode = 1'b1;

#10

dout <= 16'b0101XXXXXX1XXXXX;

decode\_cg.sample();

#20

if(E\_Control != 6'b01XXX0)

begin

dec\_er.incr\_error("E\_Control\_ANDi",reset,npc\_in,dout,enable\_decode,IR,npc\_out,E\_Control,W\_Control);

end

else

begin

$display("---------------------------------------------\n");

$display("E\_Control\_ANDi is good");

$display("---------------------------------------------\n");

end

endtask

//\*\*\*\*\*\*\*\*\*\*\*\*E\_Control\_NOT\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

task EC\_NOT(decode\_covergroup decode\_cg,decode\_error\_rec dec\_er);

//repeat(5)

//begin

enable\_decode = 1'b1;

#10

dout <= 16'b1001XXXXXX111111;

decode\_cg.sample();

#20

if(E\_Control != 6'b10XXXX)begin

dec\_er.incr\_error("E\_Control\_NOT",reset,npc\_in,dout,enable\_decode,IR,npc\_out,E\_Control,W\_Control);

end

endtask

//\*\*\*\*\*\*\*\*\*\*\*\*E\_Control\_LEA\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

task EC\_LEA(decode\_covergroup decode\_cg,decode\_error\_rec dec\_er);

enable\_decode = 1'b1;

#10

dout <= 16'b1110XXXXXXXXXXXX;

decode\_cg.sample();

#20

if(E\_Control != 6'bXX011X)

begin

dec\_er.incr\_error("E\_Control\_LEA",reset,npc\_in,dout,enable\_decode,IR,npc\_out,E\_Control,W\_Control);

end

else

begin

$display("---------------------------------------------\n");

$display("E\_Control\_LEA is good");

$display("---------------------------------------------\n");

end

endtask

///////////////////////////////////////////////////W\_CONTROL/////////////////////////////////////////////////////////////////////////////

//\*\*\*\*\*\*\*\*\*\*\*\*W\_Control\_ADD\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

task WC\_ADD(decode\_covergroup decode\_cg,decode\_error\_rec dec\_er);

enable\_decode = 1'b1;

#10

dout <= 16'b0001XXXXXX000XXX;

decode\_cg.sample();

#20

if(W\_Control != 2'b00)

begin

dec\_er.incr\_error("W\_Control\_ADD",reset,npc\_in,dout,enable\_decode,IR,npc\_out,E\_Control,W\_Control);

end

endtask

//\*\*\*\*\*\*\*\*\*\*\*\*W\_Control\_ADDi\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

task WC\_ADDi(decode\_covergroup decode\_cg,decode\_error\_rec dec\_er);

enable\_decode = 1'b1;

#10

dout <= 16'b0001XXXXXX1XXXXX;

decode\_cg.sample();

#20

if(W\_Control != 2'b00)

begin

dec\_er.incr\_error("W\_Control\_ADDi",reset,npc\_in,dout,enable\_decode,IR,npc\_out,E\_Control,W\_Control);

end

//enable\_decode <= 1'b0;

endtask

//\*\*\*\*\*\*\*\*\*\*\*\*W\_Control\_AND\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

task WC\_AND(decode\_covergroup decode\_cg,decode\_error\_rec dec\_er);

enable\_decode = 1'b1;

#10

dout <= 16'b0101XXXXXX000XXX;

#20

if(W\_Control != 2'b00)

begin

dec\_er.incr\_error("W\_Control\_AND",reset,npc\_in,dout,enable\_decode,IR,npc\_out,E\_Control,W\_Control);

end

endtask

//\*\*\*\*\*\*\*\*\*\*\*\*W\_Control\_ANDi\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

task WC\_ANDi(decode\_covergroup decode\_cg,decode\_error\_rec dec\_er);

enable\_decode = 1'b1;

#10

dout <= 16'b0101XXXXXXX1XXXX;

decode\_cg.sample();

#20

if(W\_Control != 2'b00)

begin

dec\_er.incr\_error("W\_Control\_ANDi",reset,npc\_in,dout,enable\_decode,IR,npc\_out,E\_Control,W\_Control);

end

endtask

//\*\*\*\*\*\*\*\*\*\*\*\*W\_Control\_NOT\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

task WC\_NOT(decode\_covergroup decode\_cg,decode\_error\_rec dec\_er);

enable\_decode = 1'b1;

#10

dout <= 16'b1001XXXXXX111111;

decode\_cg.sample();

#20

if(W\_Control != 2'b00)

begin

dec\_er.incr\_error("W\_Control\_NOT",reset,npc\_in,dout,enable\_decode,IR,npc\_out,E\_Control,W\_Control);

end

endtask

//\*\*\*\*\*\*\*\*\*\*\*\*W\_Control\_LEA\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

task WC\_LEA(decode\_covergroup decode\_cg,decode\_error\_rec dec\_er);

enable\_decode = 1'b1;

#10

dout <= 16'b1110XXXXXXXXXXXX;

decode\_cg.sample();

#20

if(W\_Control != 2'b10)

begin

dec\_er.incr\_error("W\_Control\_LEA",reset,npc\_in,dout,enable\_decode,IR,npc\_out,E\_Control,W\_Control);

end

endtask

endmodule

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*DecodeTop.sv\*/

module top();

bit clock;

always #5 clock = ~clock;

reg reset, enable\_decode;

reg [15:0] dout;

reg [15:0] npc\_in;

reg [1:0] W\_Control;

reg Mem\_Control;

reg [5:0] E\_Control;

reg [15:0] IR;

reg [15:0] npc\_out;

Decode DUT(.clock(clock),

.reset(reset),

.enable\_decode(enable\_decode),

.dout(dout),

.E\_Control(E\_Control),

.npc\_in(npc\_in),

.Mem\_Control(Mem\_Control),

.W\_Control(W\_Control),

.IR(IR),

.npc\_out(npc\_out));

DecodeTB TB(.clock(clock),

.reset(reset),

.enable\_decode(enable\_decode),

.dout(dout),

.E\_Control(E\_Control),

.npc\_in(npc\_in),

.Mem\_Control(Mem\_Control),

.W\_Control(W\_Control),

.IR(IR),

.npc\_out(npc\_out));

endmodule

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*top.sv\*//\*Fetch Top\*/

`timescale 1ns/1ns

import testing::\*;

module top();

bit clk;

//reg clk\_reg;

reg \_reset;

reg \_br\_taken;

reg \_enable\_fetch;

reg \_enable\_updatePC;

reg [15:0] \_taddr;

reg [15:0] \_pc;

reg [15:0] \_npc\_out;

reg \_instrmem\_rd;

Fetch DUT(

.clock(clk),

.reset(\_reset),

.enable\_updatePC(\_enable\_updatePC),

.enable\_fetch(\_enable\_fetch),

.pc(\_pc),

.npc\_out(\_npc\_out),

.instrmem\_rd(\_instrmem\_rd),

.taddr(\_taddr),

.br\_taken(\_br\_taken)

);

fetch\_test TB(

.reset(\_reset),

.br\_taken(\_br\_taken),

.enable\_fetch(\_enable\_fetch),

.enable\_updatePC(\_enable\_updatePC),

.taddr(\_taddr),

.pc(\_pc),

.npc\_out(\_npc\_out),

.instrmem\_rd(\_instrmem\_rd),

.clock(clk)

);

always begin

#5 clk <= ~clk;

//always #5 clk\_reg <= clk;

end

endmodule