*Course No. ELEC 5200/6200*

*Computer Architecture and Design*

Project: Part-4

Date: April 7, 2017

**

*Prepared by*

*Md. Mahabubul Alam*

*Banner ID: 903937005*

*Graduate Student, Electrical Engineering*

*Auburn University*

*Benjamin Cyr*

*Banner ID: 903466318*

*Senior, Computer Engineering*

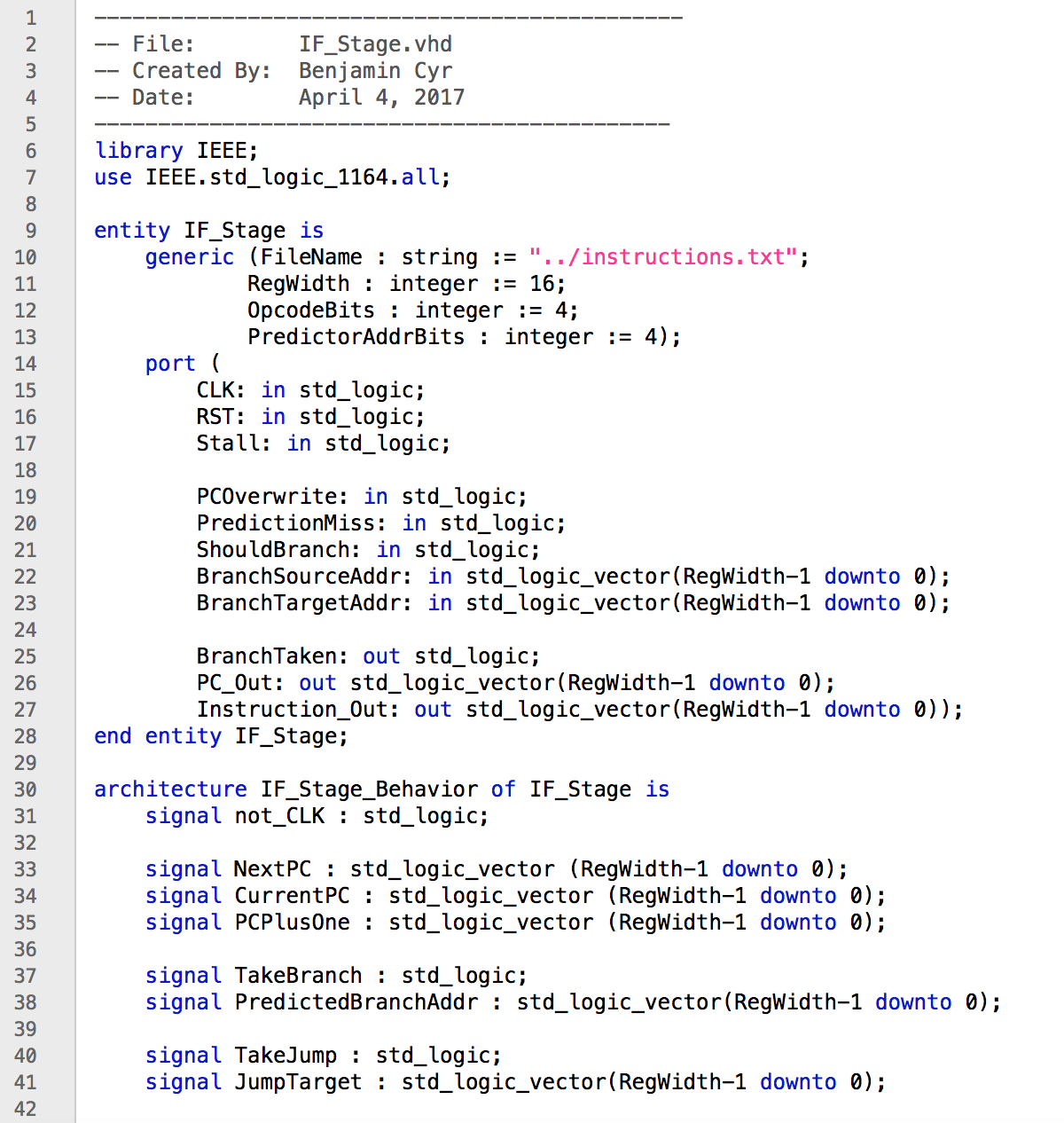
*Auburn University*

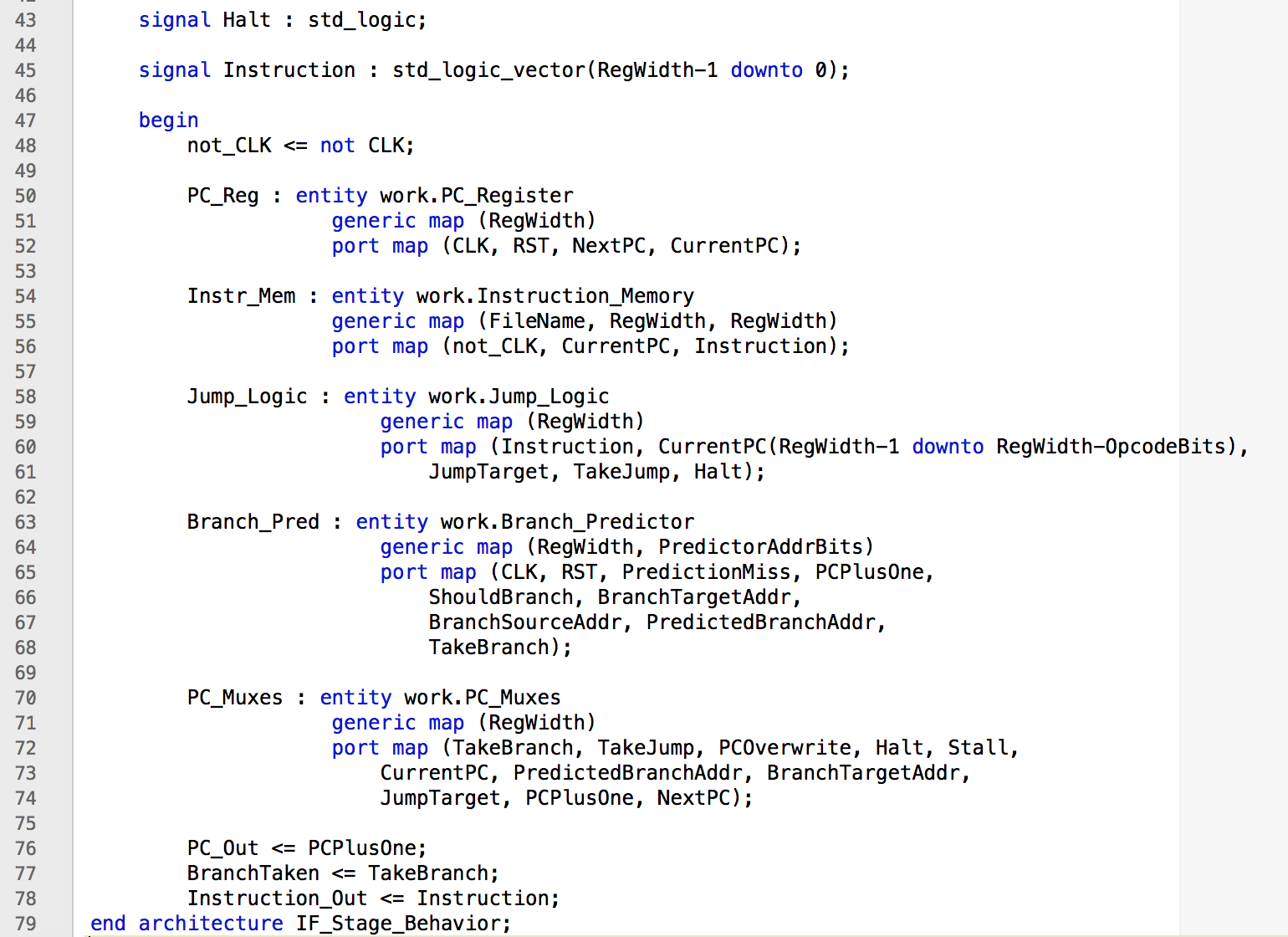
***Project Requirements:***

Verify this top-level component as described in the block diagram and register transfers in Part 2. Write a testbench to verify the correct execution of each instruction in your ISA (except the instructions which need data memory access, like “lw” and “sw”). Show in the simulation where you verified each required register transfer for the CPU. (If some register transfers are common to multiple instructions, you do not need to show them separately for every instruction – but it might be a good idea to do so anyway.) After that, write a short program using your instruction set (containing all kinds of instructions except “lw” and “sw”), hand compile the chosen test program into binary code and then use this code to verify if your datapath gives you the expected results.

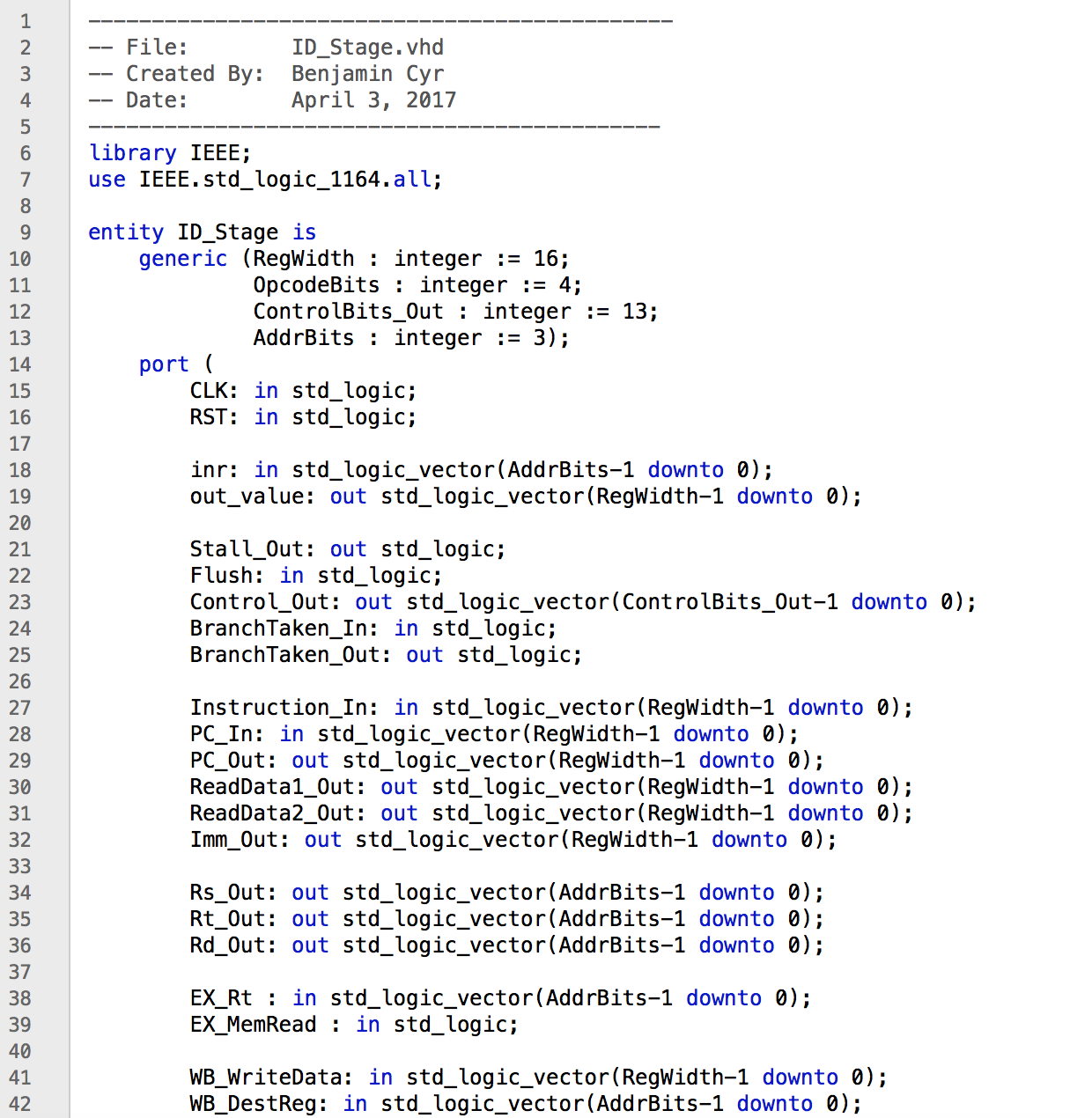
***Source Code:***

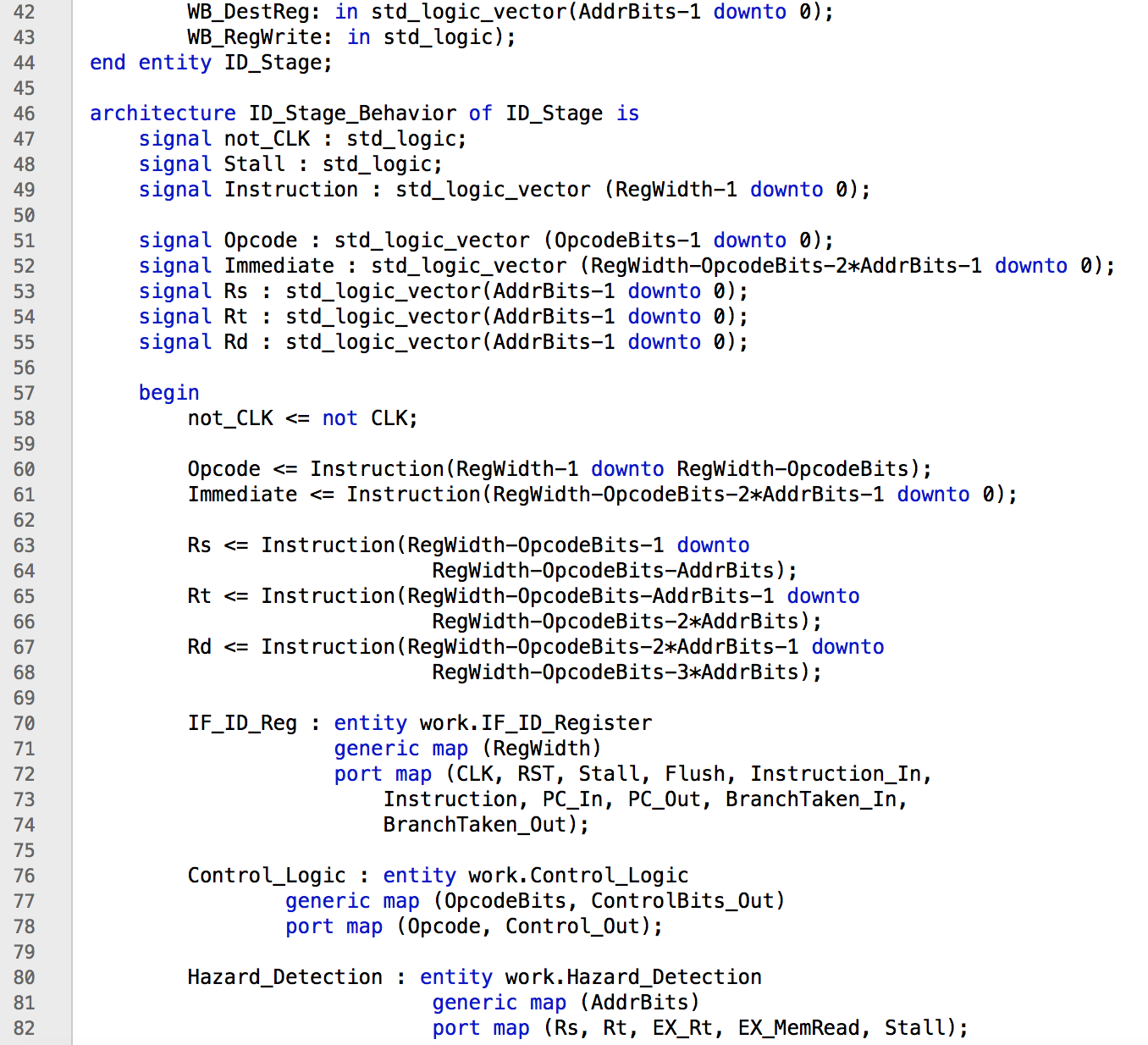
IF\_Stage.vhd: Consists of all the parts of the Instruction Fetch Stage.

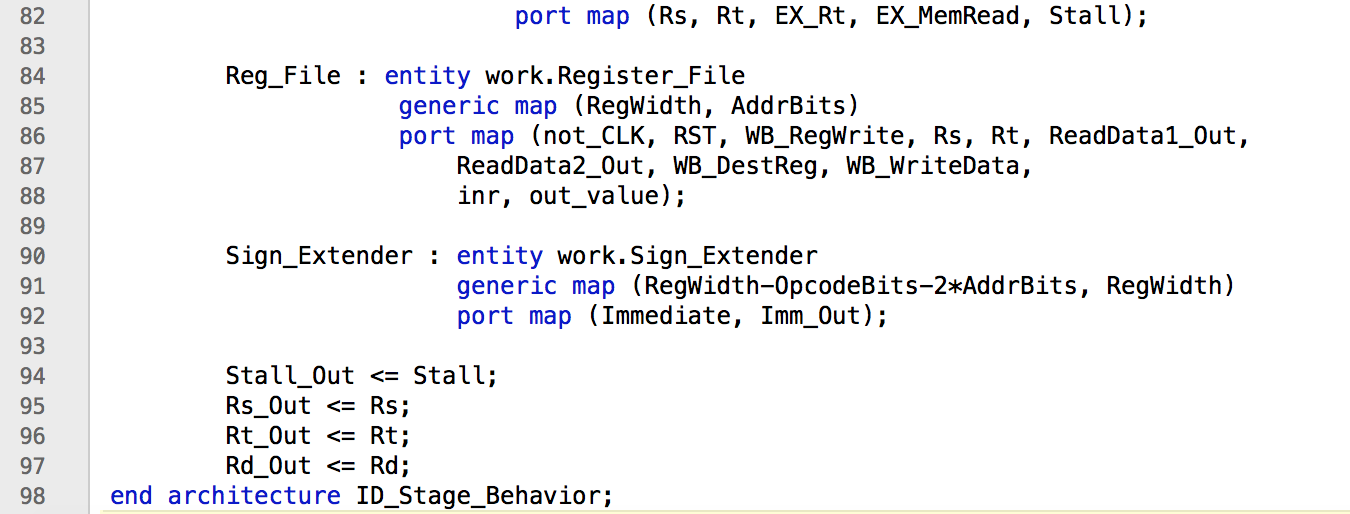


******

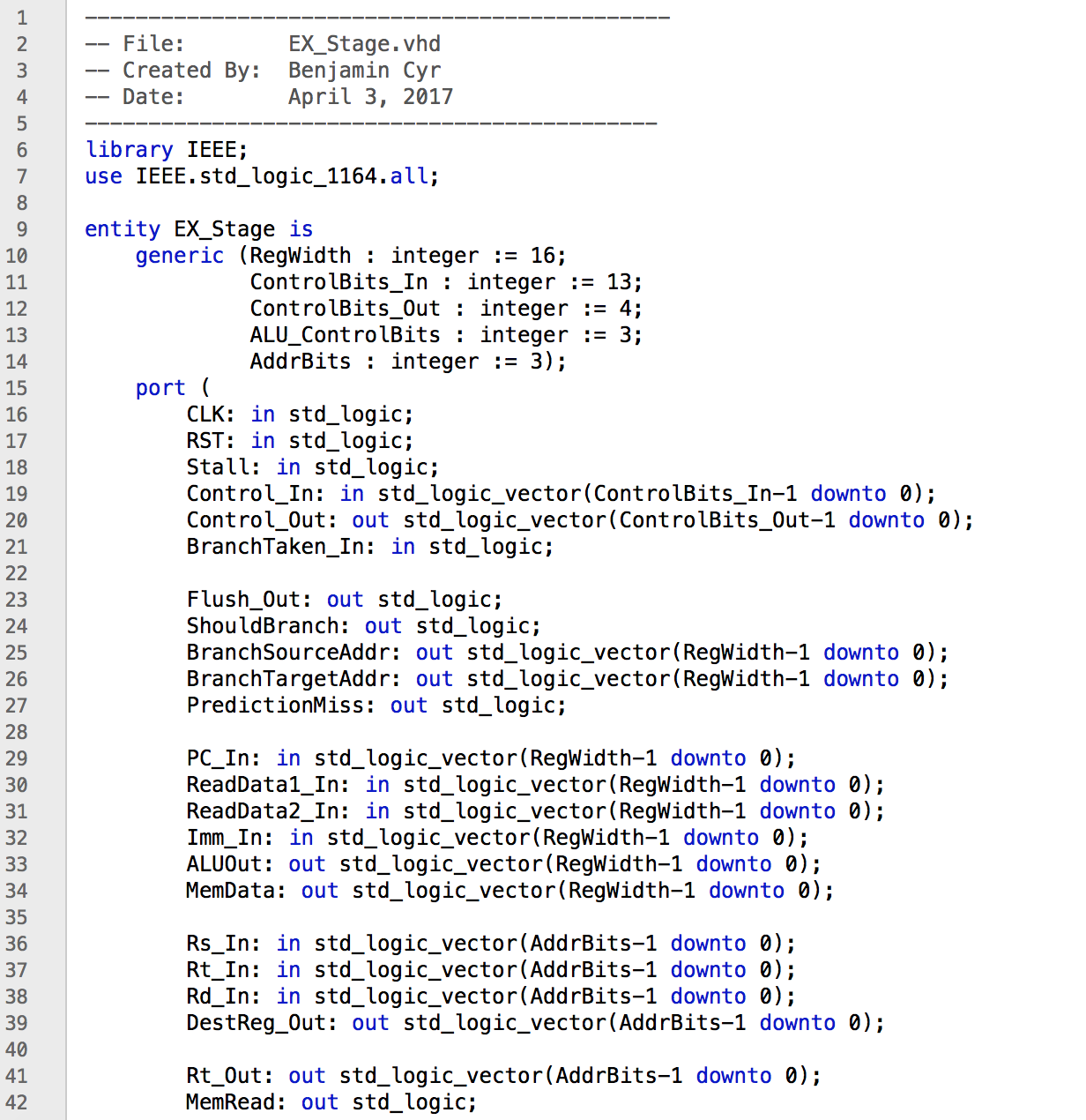
ID\_Stage.vhd: Consists of all the parts of the Instruction Decode Stage.

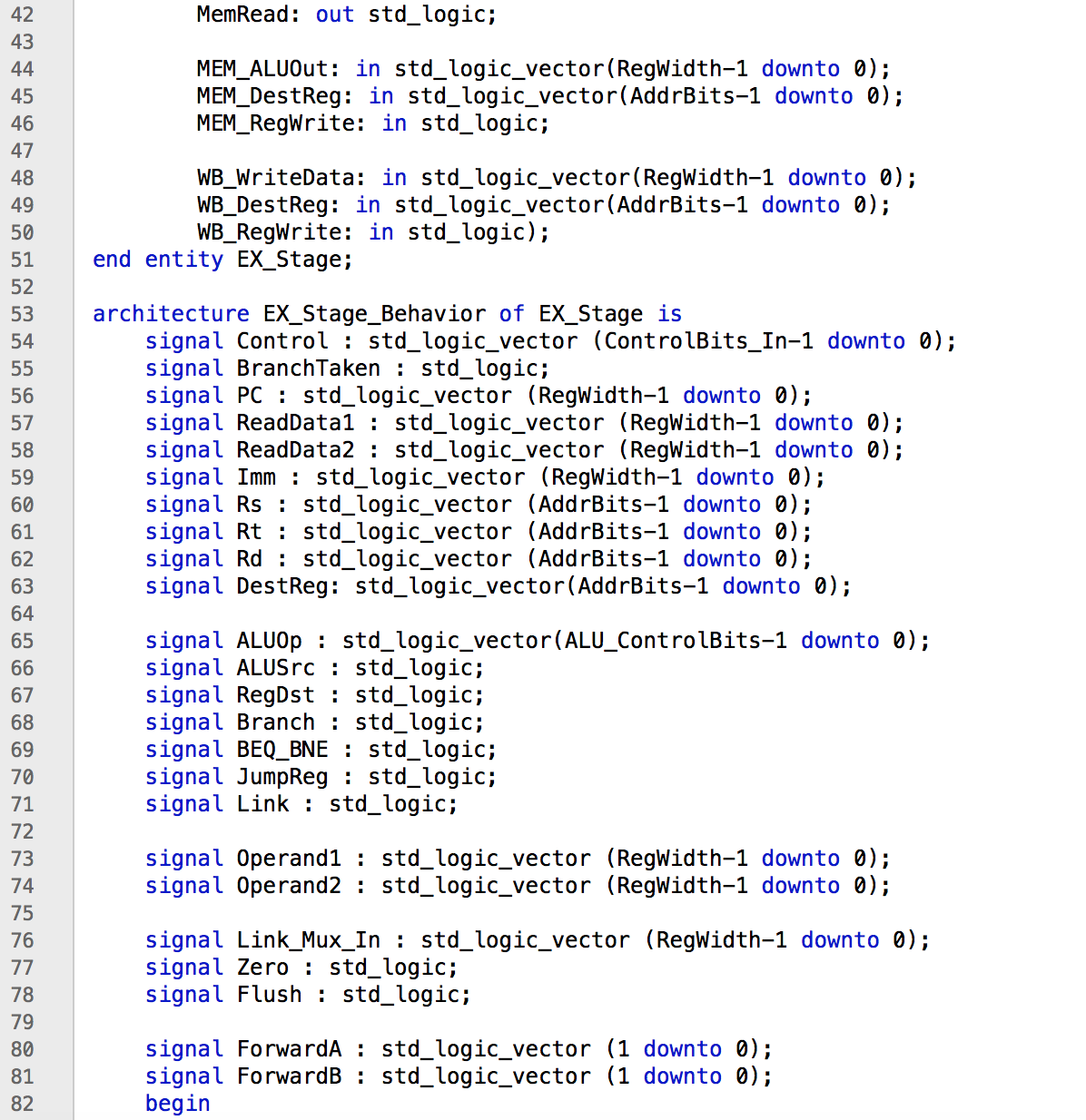
******

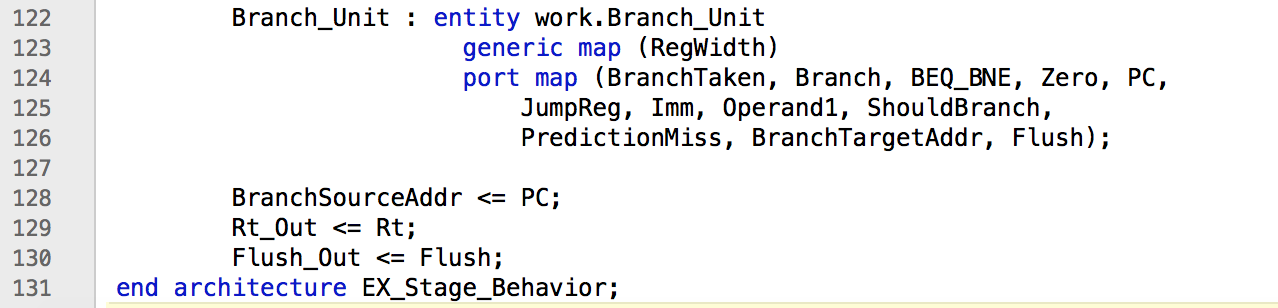
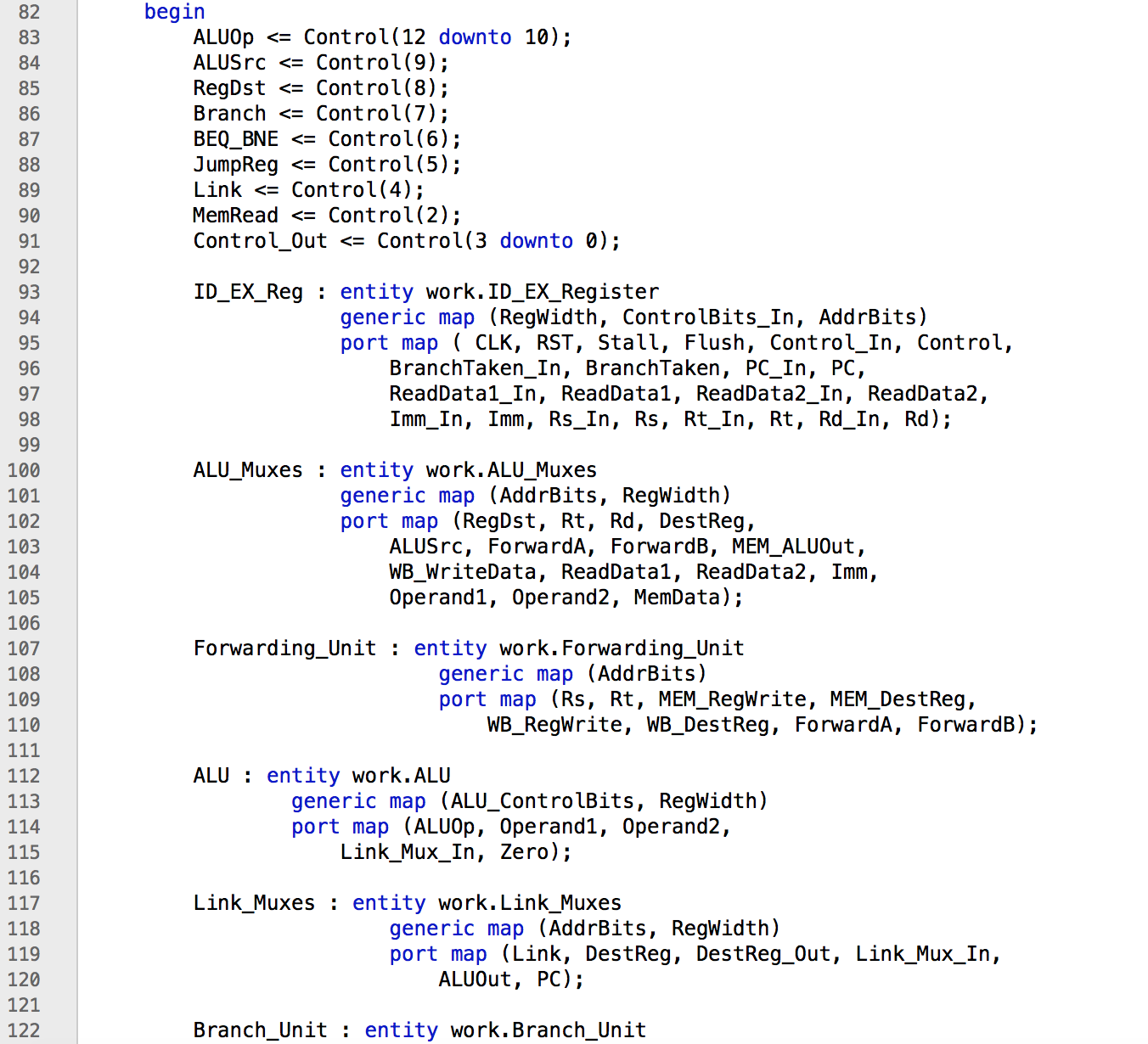
******

******

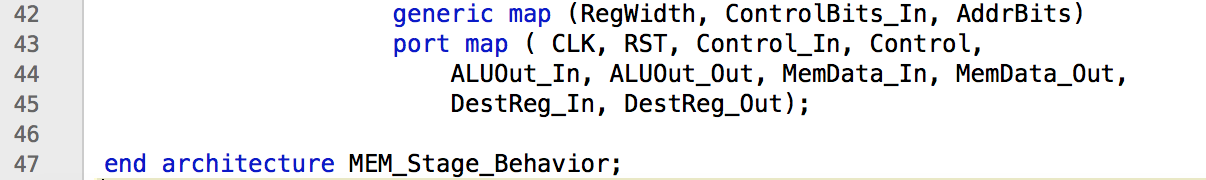
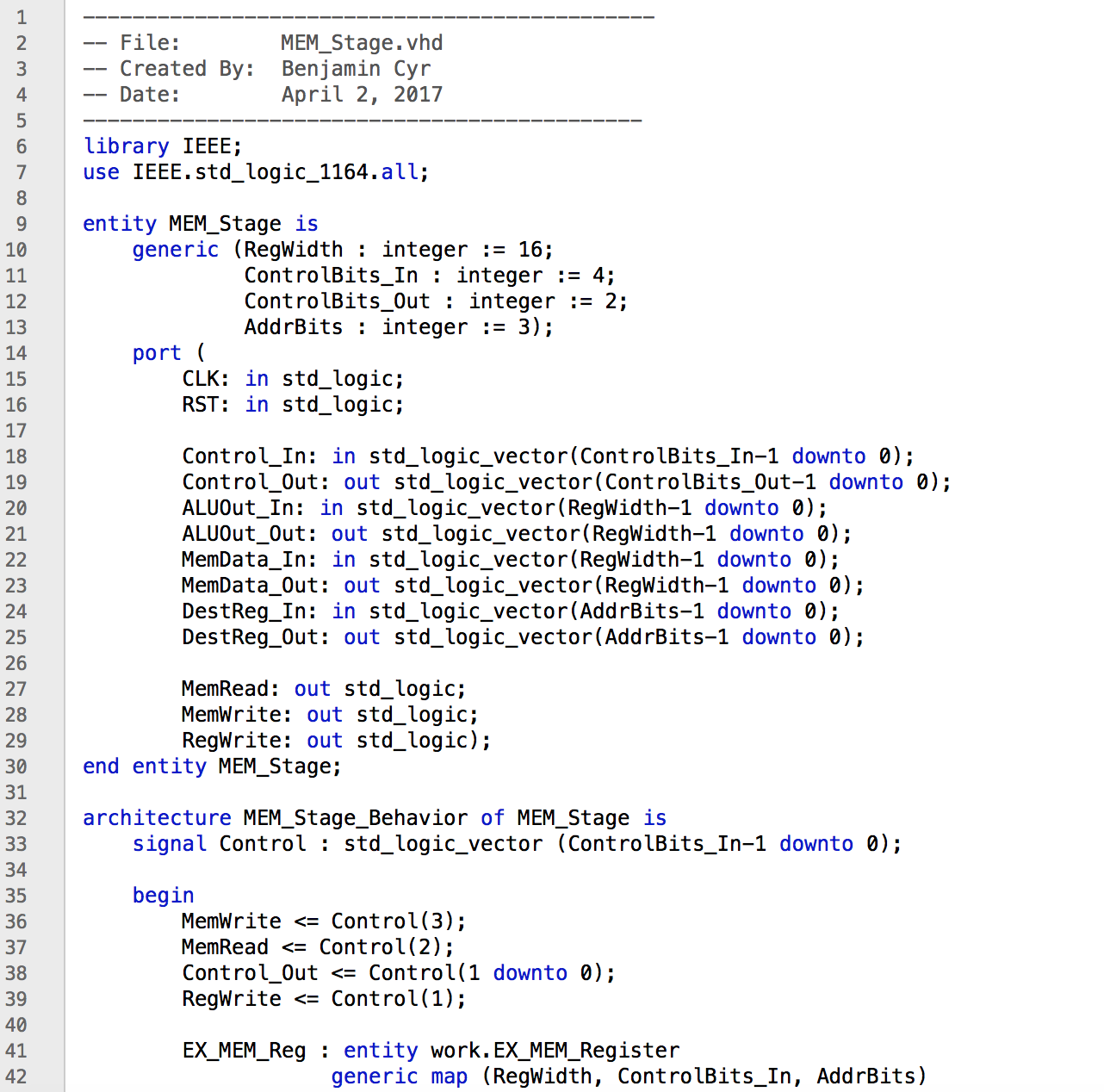
EX\_Stage.vhd: Consists of all the parts of the Execution Stage.

******

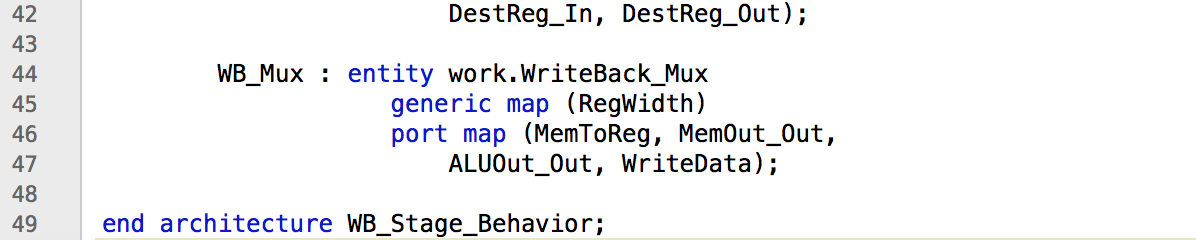
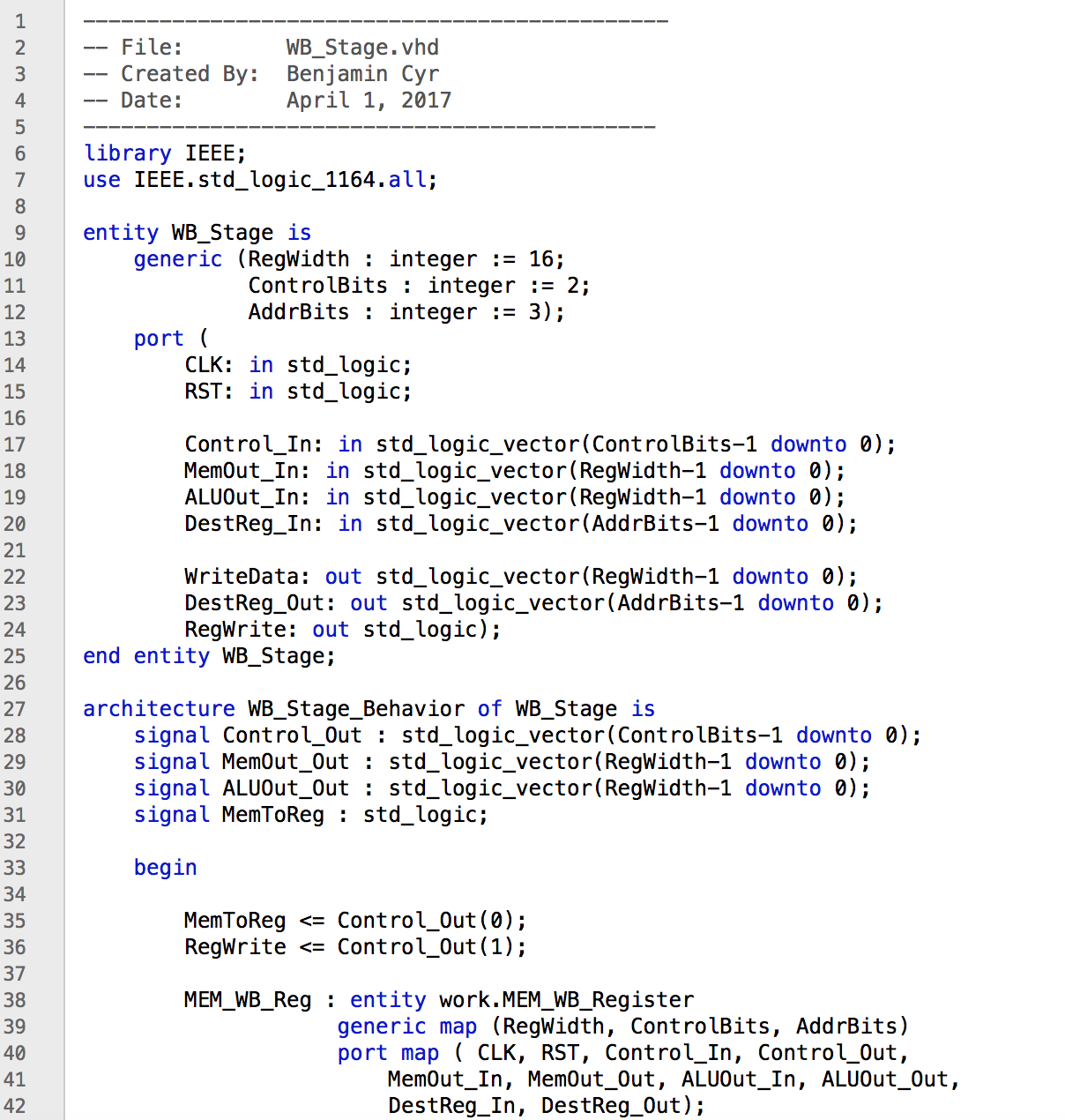
******

******

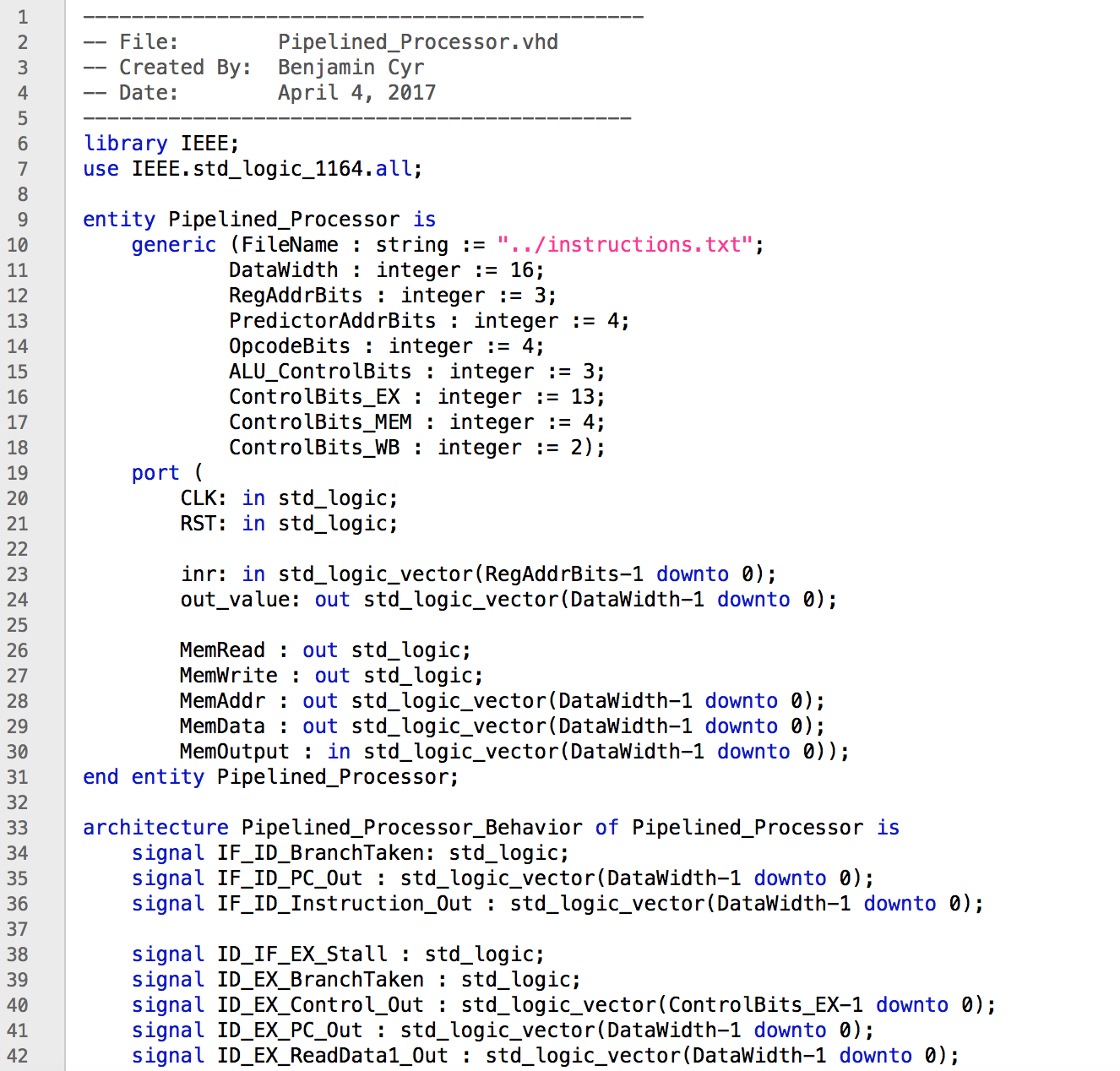
MEM\_Stage.vhd: Consists of all the parts of the Memory Stage.

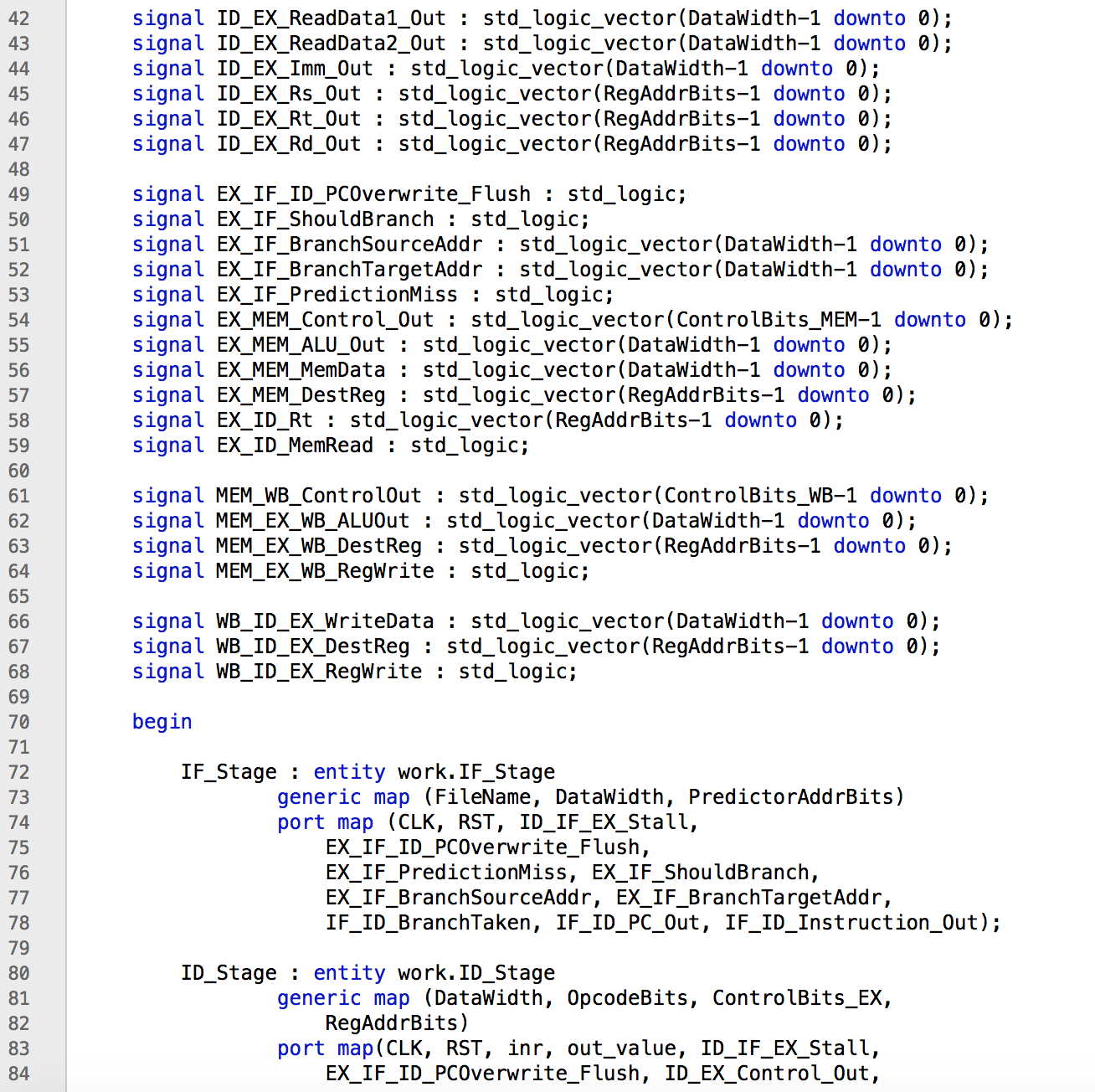
******

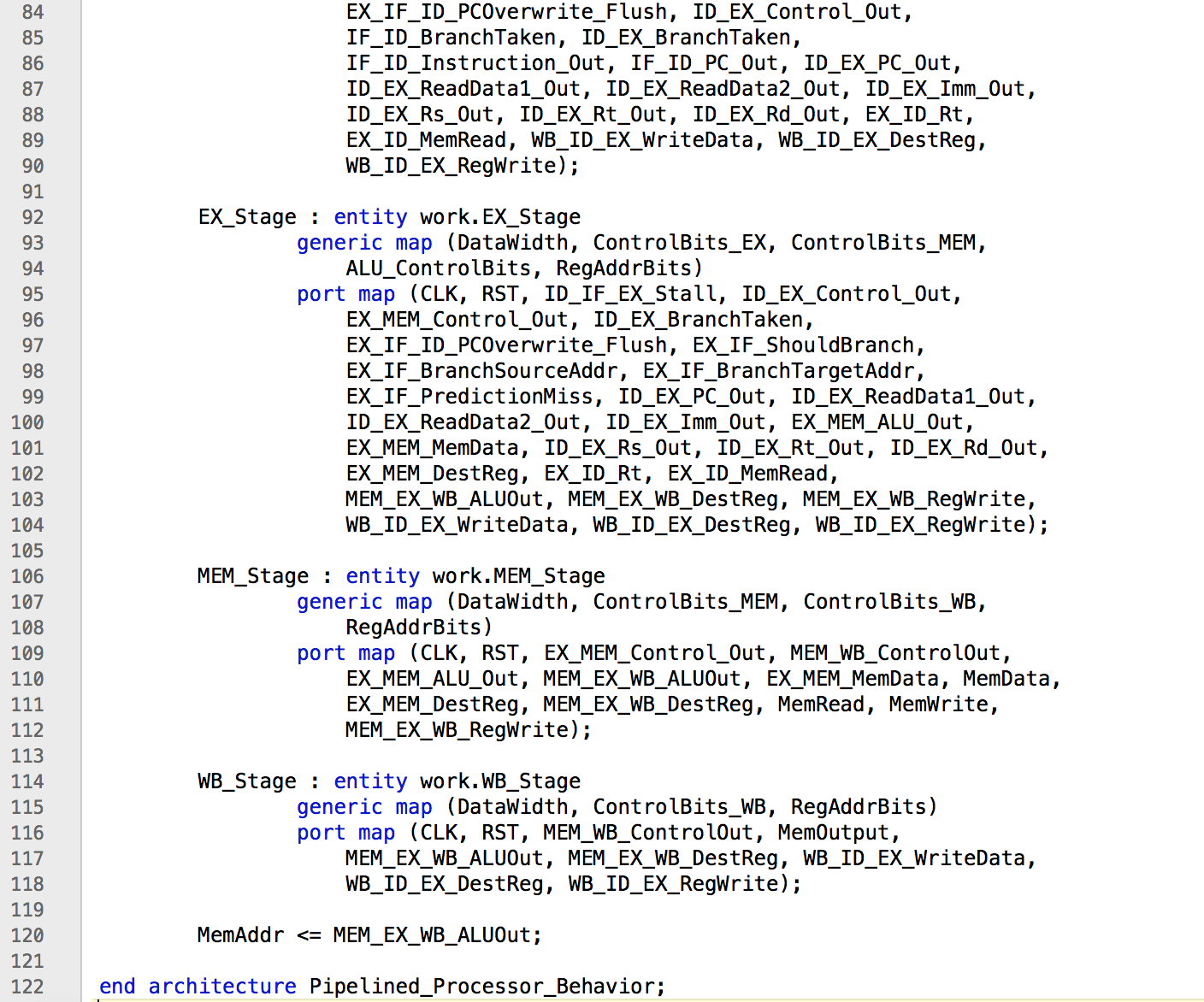
WB\_Stage.vhd: Consists of all the parts of the Writeback Stage.

******

Pipelined\_Processor.vhd: Consists of all five stages to form the processor. The only external connections are the Clock, Reset signal, inr, out\_value, and the signals required to interact with the data memory.

******

******

******

***Instruction Simulations***

Every Instruction except LW and SW was simulated individually to show that each worked correctly. In each case a small program was run using test benches, and the contents of the register file were then shown to verify the correct values were written. The results are shown below:

|  |  |  |
| --- | --- | --- |
| Time | inr | out\_value |
| 60 ns | 0 | 0000 |
| 70 ns | 1 | 0000 |
| 80 ns | 2 | ffea |
| 90 ns | 3 | 0003 |
| 100 ns | 4 | 0004 |
| 110 ns | 5 | 0000 |
| 120 ns | 6 | 0000 |
| 130 ns | 7 | 0000 |

ADDI:

ADDI $2, $0, -22

ADDI $3, $0, 3

ADDI $4, $3, 1

HALT

ADD:

|  |  |  |
| --- | --- | --- |
| Time | inr | out\_value |
| 60 ns | 0 | 0000 |
| 70 ns | 1 | 0000 |
| 80 ns | 2 | 0000 |
| 90 ns | 3 | 0000 |
| 100 ns | 4 | 0000 |
| 110 ns | 5 | 0002 |
| 120 ns | 6 | 0004 |
| 130 ns | 7 | 0006 |

ADDI $5, $0, 2

ADD $6, $5, $5

ADD $7, $6, $5

HALT

SUB:

|  |  |  |
| --- | --- | --- |
| Time | inr | out\_value |
| 60 ns | 0 | 0000 |
| 70 ns | 1 | fffb |
| 80 ns | 2 | fff6 |
| 90 ns | 3 | 0000 |
| 100 ns | 4 | 0000 |
| 110 ns | 5 | 0000 |
| 120 ns | 6 | 0000 |
| 130 ns | 7 | 0000 |

ADDI $2, $0, 5

SUB $1, $0, $2

SUB $2, $1, $2

HALT

|  |  |  |
| --- | --- | --- |
| Time | inr | out\_value |
| 60 ns | 0 | 0000 |
| 70 ns | 1 | 0000 |
| 80 ns | 2 | 0000 |
| 90 ns | 3 | fffe |
| 100 ns | 4 | 0015 |
| 110 ns | 5 | 0014 |
| 120 ns | 6 | 0000 |
| 130 ns | 7 | 0000 |

AND:

ADDI $3, $0, 0xFFFE

ADDI $4, $0, 0x0015

AND $5, $3, $4

HALT

|  |  |  |
| --- | --- | --- |
| Time | inr | out\_value |
| 60 ns | 0 | 0000 |
| 70 ns | 1 | 0000 |
| 80 ns | 2 | 0000 |
| 90 ns | 3 | 000a |
| 100 ns | 4 | 0015 |
| 110 ns | 5 | 001f |
| 120 ns | 6 | 0000 |
| 130 ns | 7 | 0000 |

OR:

ADDI $3, $0, 0x000A

ADDI $4, $0, 0x0015

OR $5, $3, $4

HALT

|  |  |  |
| --- | --- | --- |
| Time | inr | out\_value |
| 60 ns | 0 | 0000 |
| 70 ns | 1 | ffff |
| 80 ns | 2 | 0003 |
| 90 ns | 3 | 0001 |
| 100 ns | 4 | 0000 |
| 110 ns | 5 | 0000 |
| 120 ns | 6 | 0000 |
| 130 ns | 7 | 0000 |

SLT:

ADDI $1, $0, -1

ADDI $2, $0, 3

SLT $3, $1, $2

SLT $4, $2, $1

SLT $5, $1, $1

HALT

|  |  |  |
| --- | --- | --- |
| Time | inr | out\_value |
| 60 ns | 0 | 0000 |
| 70 ns | 1 | fffa |
| 80 ns | 2 | fff4 |
| 90 ns | 3 | ffa0 |
| 100 ns | 4 | 0000 |
| 110 ns | 5 | 0000 |
| 120 ns | 6 | 0000 |
| 130 ns | 7 | 0000 |

LSI:

ADDI $1, $0, 0xFFFA

LSI $2, $1, 1

LSI $3, $2, 3

HALT

|  |  |  |
| --- | --- | --- |
| Time | inr | out\_value |
| 60 ns | 0 | 0000 |
| 70 ns | 1 | 0000 |
| 80 ns | 2 | 0000 |
| 90 ns | 3 | 0000 |
| 100 ns | 4 | ffeb |
| 110 ns | 5 | fff5 |
| 120 ns | 6 | fffe |
| 130 ns | 7 | 0000 |

RSI:

ADDI $4, $0, 0xFFEB

RSI $5, $4, 1

RSI $6, $5, 3

HALT

|  |  |  |
| --- | --- | --- |
| Time | inr | out\_value |
| 80 ns | 0 | 0000 |
| 90 ns | 1 | 0003 |
| 100 ns | 2 | 0003 |
| 110 ns | 3 | 0001 |
| 120 ns | 4 | 0000 |
| 130 ns | 5 | 0000 |
| 140 ns | 6 | 0000 |
| 150 ns | 7 | 0000 |

BNE:

ADDI $1, $0, 3

ADDI $2, $0, 3

BNE $1, $2, +1

ADDI $3, $0, 1

BNE $3, $0, +1

ADDI $4, $0, -1

HALT

|  |  |  |
| --- | --- | --- |
| Time | inr | out\_value |
| 80 ns | 0 | 0000 |
| 90 ns | 1 | 0003 |
| 100 ns | 2 | 0003 |
| 110 ns | 3 | 0000 |
| 120 ns | 4 | ffff |
| 130 ns | 5 | 0000 |
| 140 ns | 6 | 0000 |
| 150 ns | 7 | 0000 |

BEQ:

ADDI $1, $0, 3

ADDI $2, $0, 3

BEQ $1, $2, +1

ADDI $3, $0, 1

BEQ $1, $0, +1

ADDI $4, $0, -1

HALT

|  |  |  |
| --- | --- | --- |
| Time | inr | out\_value |
| 60 ns | 0 | 0000 |
| 70 ns | 1 | 0001 |
| 80 ns | 2 | 0000 |
| 90 ns | 3 | 0000 |
| 100 ns | 4 | 0000 |
| 110 ns | 5 | 0000 |
| 120 ns | 6 | 0000 |
| 130 ns | 7 | 0002 |

JL:

ADDI $1, $0, 1

JL HALT

ADDI $1, $1, 1

HALT

|  |  |  |
| --- | --- | --- |
| Time | inr | out\_value |
| 60 ns | 0 | 0000 |
| 70 ns | 1 | 0001 |
| 80 ns | 2 | 0002 |
| 90 ns | 3 | 0000 |
| 100 ns | 4 | 0000 |
| 110 ns | 5 | 0000 |
| 120 ns | 6 | 0000 |
| 130 ns | 7 | 0000 |

J:

ADDI $1, $0, 1

J +1

ADDI $1, $1, 1

ADDI $2, $1, 1

HALT

|  |  |  |
| --- | --- | --- |
| Time | inr | out\_value |
| 80 ns | 0 | 0000 |
| 90 ns | 1 | 0003 |
| 100 ns | 2 | 0004 |
| 110 ns | 3 | 0000 |
| 120 ns | 4 | 0000 |
| 130 ns | 5 | 0000 |
| 140 ns | 6 | 0000 |
| 150 ns | 7 | 0000 |

JR:

ADDI $1, $0, 3

JR $1

ADDI $1, $1, 1

ADDI $2, $1, 1

HALT

***Complete Program Simulation***

Finally, a complete program was simulated on the CPU. The program and the results are shown below:

ADDI $2, $0, 4 ; $2 = 4

ADDI $3, $0, 1 ; $3 = 1

L1: SUB $2, $2, $3 ; Loop 3 times

BNE $2, $3, L1

ADD $4, $2, $3 ; $4 = 1 + 1

OR $5, $4, $3 ; $5 = 2 | 1 = 3

AND $6, $5, $3 ; $6 = 3 & 1 = 1

L2: BEQ $5, $0, L3 ;

LSI $5, $5, 4 ; $5 = 3 << 4

RSI $4, $4, 1 ; $4 = 2 >> 1

J L2

L3: ADDI $2, $0, 5 ; $2 = 5

JL FIB ; Call FIB

SLT $2, $0, $1

HALT

FIB: ADDI $1, $0, 1 ; $1 = 1

ADDI $3, $0, 1 ; $3 = 1

L4: ADDI $2, $2, -1 ; $2 -= 1

BEQ $2, $0, L5

ADD $4, $3, $1 ;$4 = $3 + $1

ADD $1, $3, $0 ;$1 = $3

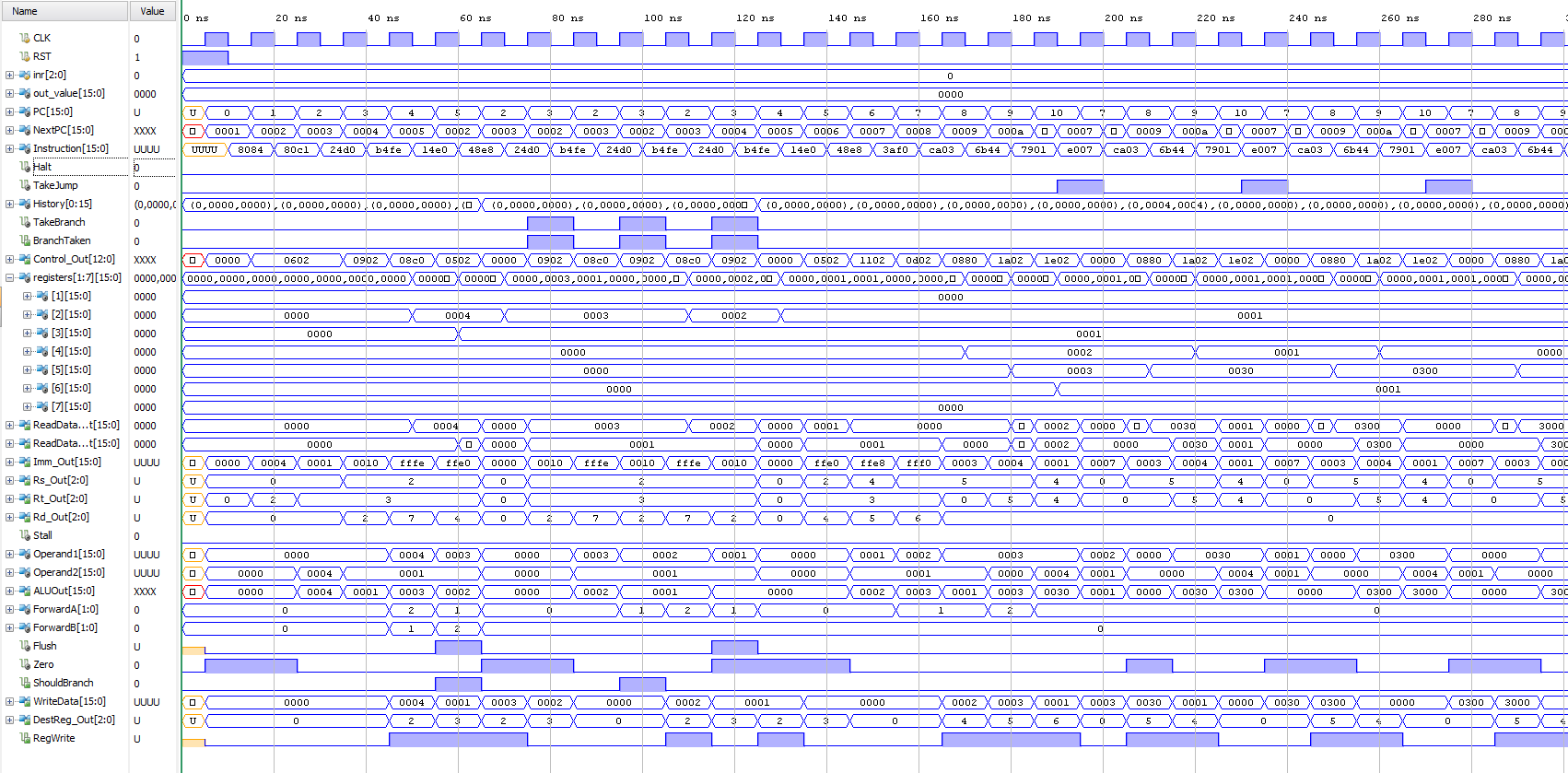
ADD $3, $4, $0 ; $3 = $4

J L4

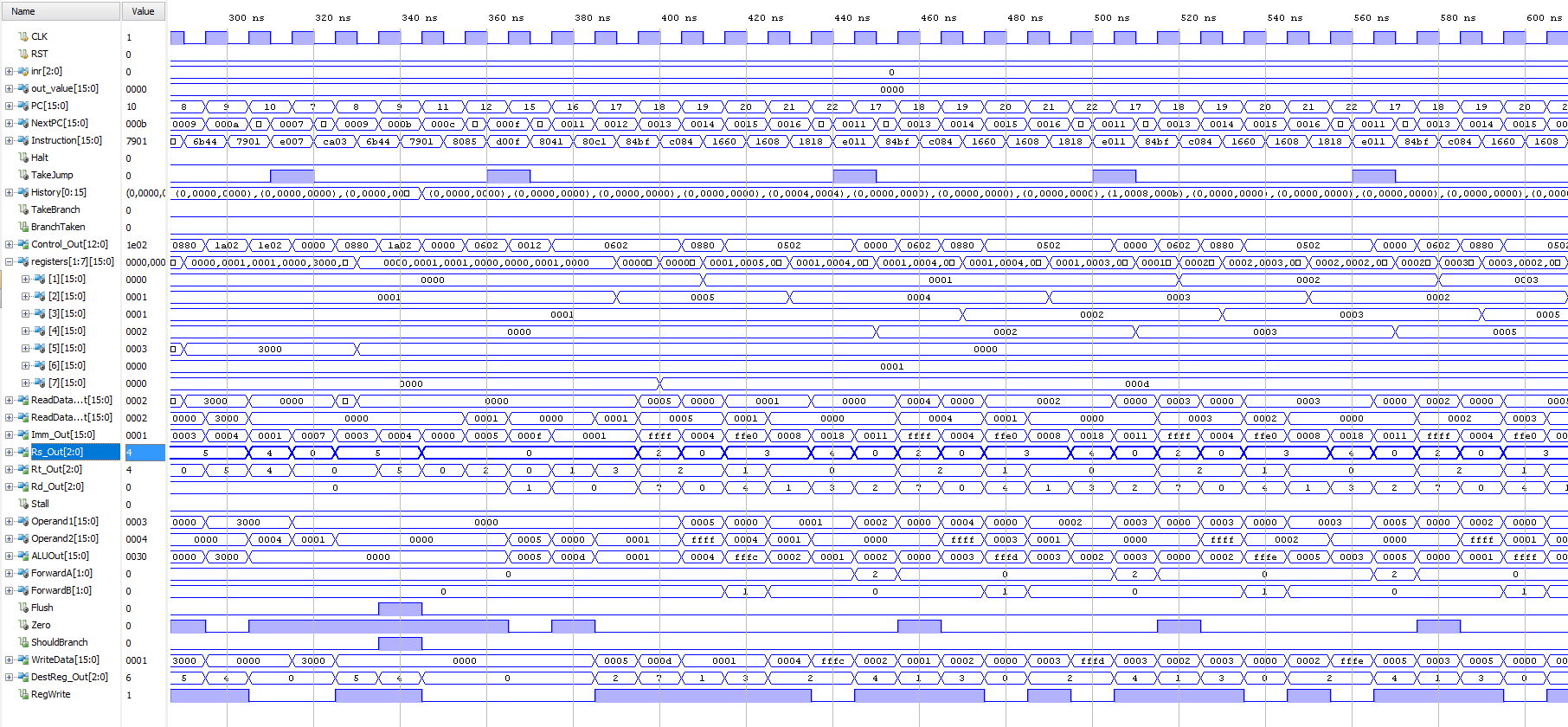
L5: ADD $1, $4, $0 ;$1 = $4

JR $7

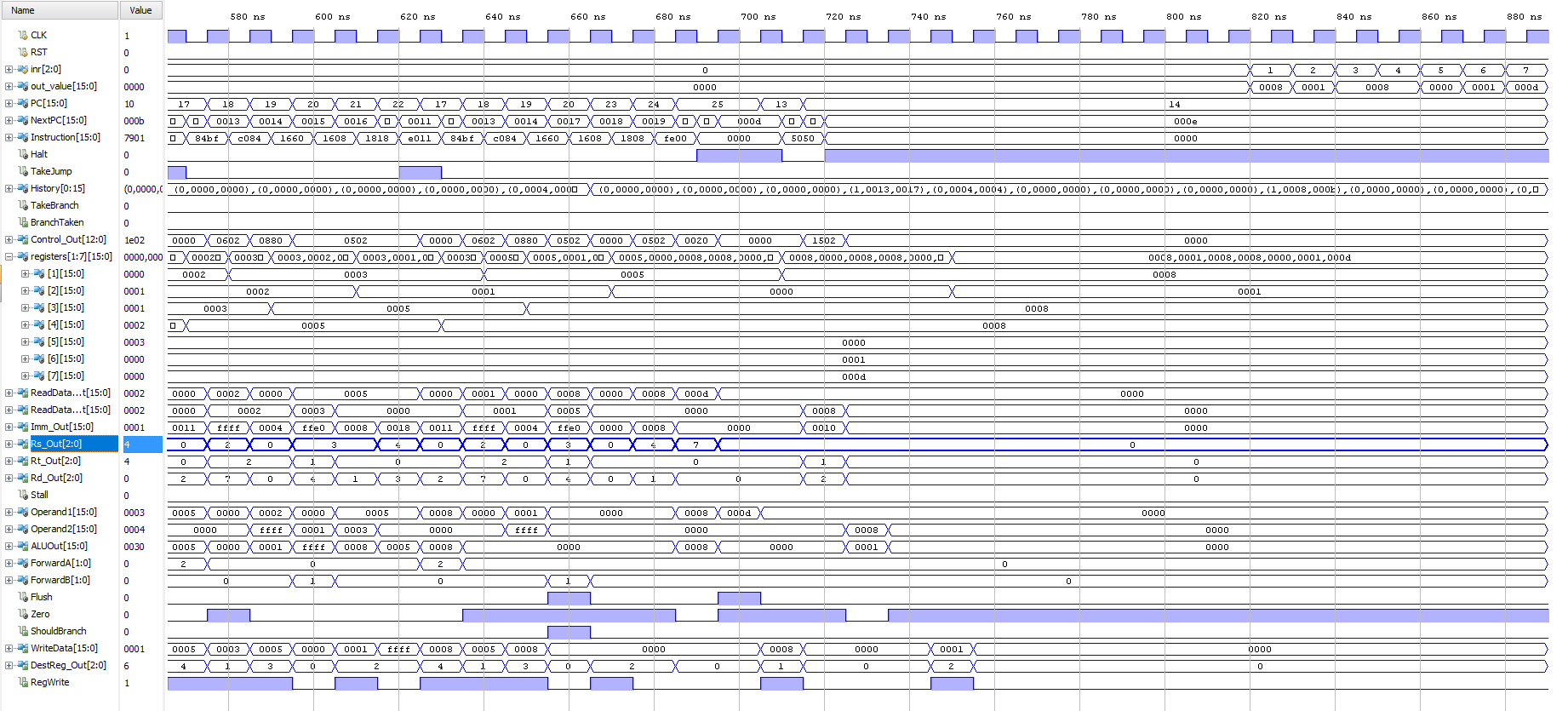
The first section shows that the Branch Predictor was functioning as expected, only flushing the pipeline twice for the L1 Loop. The registers all updated to the correct values. The Jumps for the L2 Loop were also taken and correct.

******

The second section shows that that the JL was performed correctly and $7 was updated to the correct value. The algorithm to calculate the Fibonacci number was started using register $1-$4.

******

The final section shows that the correct Fibonacci number (8) was returned from the function call. It also shows $2 was set with the SLT instruction because $0 was less than $1. Finally the program halted and kept the registers constant.

******