Lab 3 – Single Cycle Processor

Overview

The scope of this lab is to explore the implementation of a single cycle processor in Verilog. Through these implementations, I expect to learn more about how processors work and the function of each sub-component of a processor.

Modules implemented

Diagram with all Modules labelled:

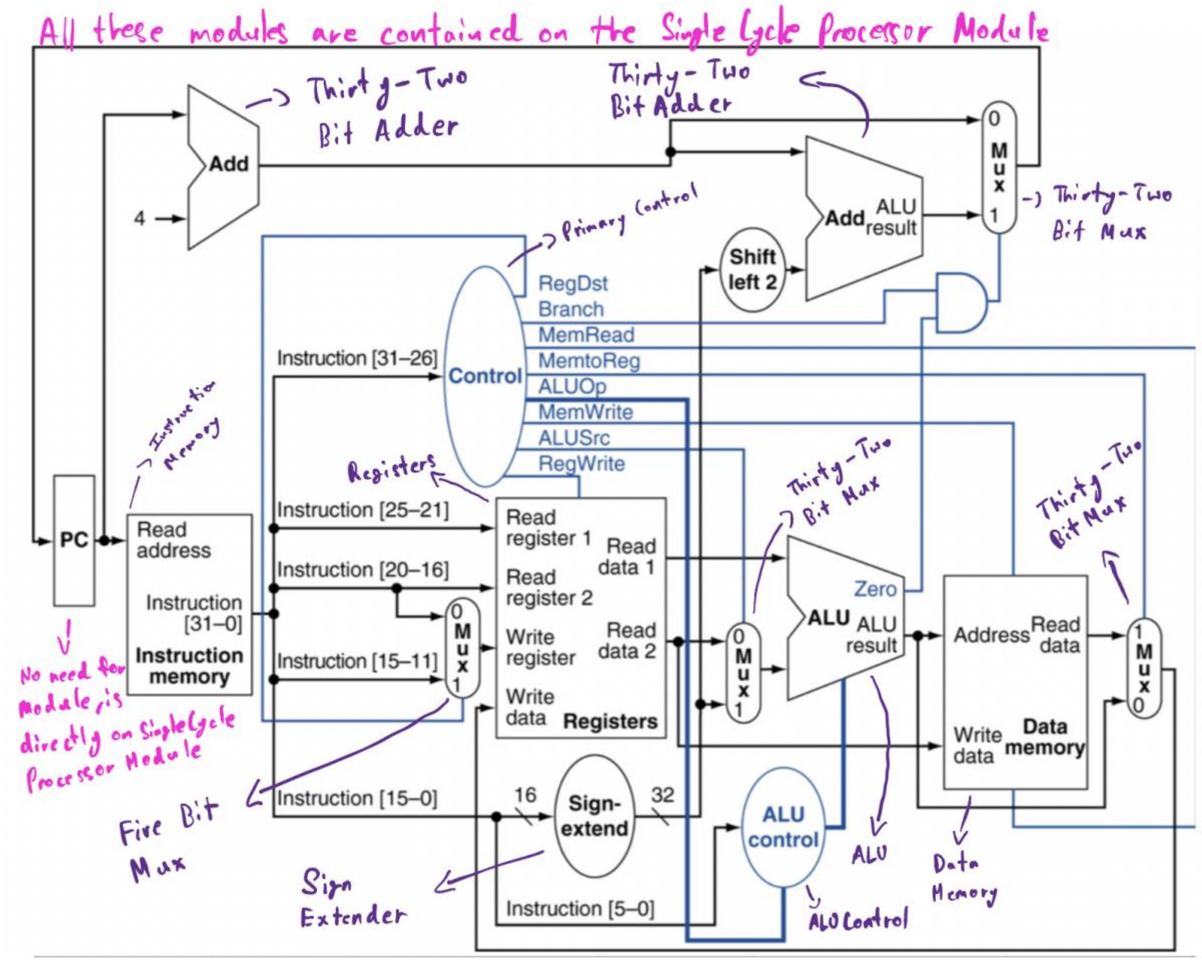


Figure 1: Diagram of Module Names

Primary Control (primarycontrol.v)

The primary control takes in the first 6 bits of the instruction from the instructionmemory and then sends controlling signals to various parts of the processor. There are four different combinations of inputs that my primary control accepts and then accordingly assigns values to the 8 different outputs. These outputs can be seen in Figure 2 for verification that they are as intended.

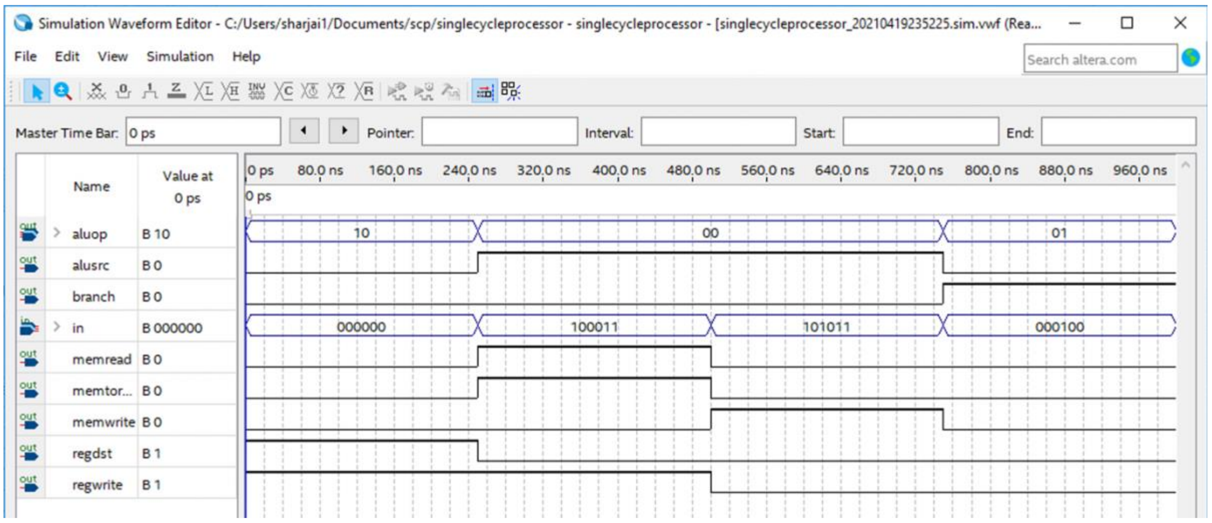

Simulation Waveform Editor - C:/Users/sharjai1/Documents/scp/singlecycleprocessor - singlecycleprocessor - [singlecycleprocessor_20210419235225.sim.vwf (Rea...
File Edit View Simulation Help
Master Time Bar: 0 ps
Pointer:
Interval:
Start:
End:
Name
Value at 0 ps
0 ps 80.0 ns 160.0 ns 240.0 ns 320.0 ns 400.0 ns 480.0 ns 560.0 ns 640.0 ns 720.0 ns 800.0 ns 880.0 ns 960.0 ns
aluop B 10 10 00 01
alusrc B 0
branch B 0
in B 000000 000000 100011 101011 000100
memread B 0
memtor... B 0
memwrite B 0
regdst B 1
regwrite B 1

Figure 2: primarycontrol.vwf showing the correct outputs for Primary Control module

Registers (registers.v)

The registers module has 32 registers of 32 bits each. The module can read two registers at once and output their values to read data 1 and read data 2 outputs. It can also write to the register address in Write Register input with data in Write Data when RegWrite in from the primary control is asserted high. Figure 3 shows the correct operation of Registers with examples of the reset line working, the “regAdd1” and “regAdd2” retrieving items from registers and then outputting to “out1” and “out2” respectively, and “regWrite” from the primary control and “wdata”, “wreg” working to write to two different registers.

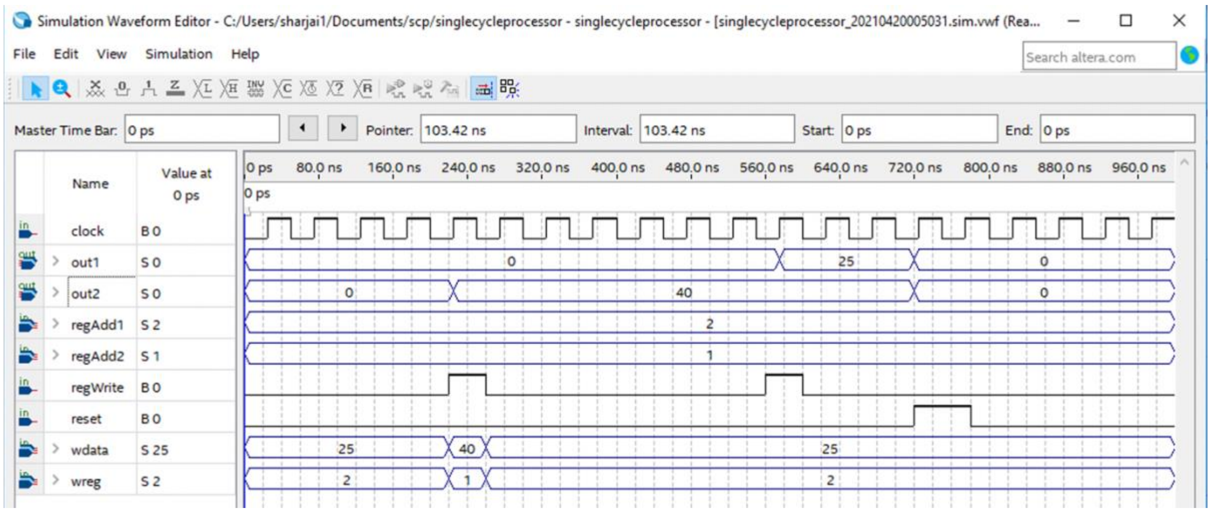

Simulation Waveform Editor - C:/Users/sharjai1/Documents/scp/singlecycleprocessor - singlecycleprocessor - [singlecycleprocessor_20210420005031.sim.vwf (Rea...
File Edit View Simulation Help
Master Time Bar: 0 ps
Pointer: 103.42 ns
Interval: 103.42 ns
Start: 0 ps
End: 0 ps
Name
Value at 0 ps
0 ps 80.0 ns 160.0 ns 240.0 ns 320.0 ns 400.0 ns 480.0 ns 560.0 ns 640.0 ns 720.0 ns 800.0 ns 880.0 ns 960.0 ns
clock B 0
out1 S 0 0 25 0
out2 S 0 0 40 0
regAdd1 S 2 2
regAdd2 S 1 1
regWrite B 0
reset B 0
wdata S 25 25 40 25
wreg S 2 2 1 2

Figure 3: registers.vwf showing the correct outputs for registers module

Instruction Memory (instructionmemory.v)

The instruction memory has 32 memory locations of 32 bits each. It takes an input from program counter which corresponds to each memory location, shifts this input to the right by 2 so that the locations are correctly referenced, and then outputs the data stored in the referenced location. The correct operation is shown in Figure 4. The VWF shows binary 0100 which is decimal 4th address in the memory location. This shifted to the right by 2, thus index 1 in the memory location is accessed. Which is correctly shown as the output.

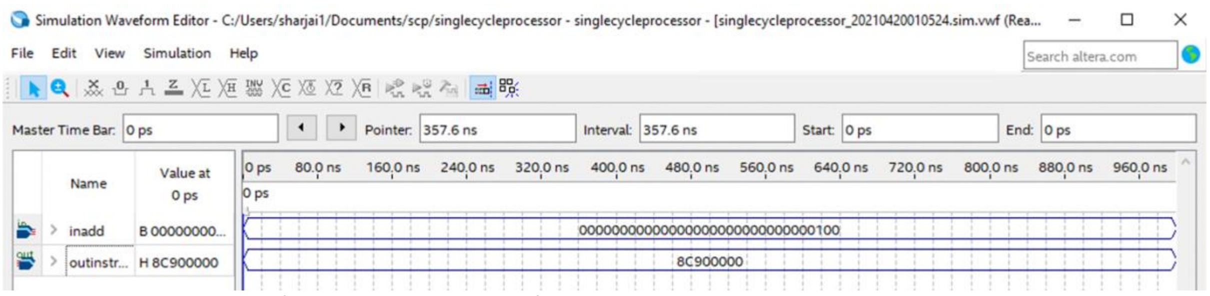

Simulation Waveform Editor - C:/Users/sharjai1/Documents/scp/singlecycleprocessor - singlecycleprocessor - [singlecycleprocessor_20210420010524.sim.vwf (Rea...
File Edit View Simulation Help
Master Time Bar: 0 ps
Pointer: 357.6 ns
Interval: 357.6 ns
Start: 0 ps
End: 0 ps
Name
Value at 0 ps
inadd B 00000000...
outinstr... H 8C900000
00000000000000000000000000000100
8C900000

Figure 4: instructionmemory.vwf showing the correct output for the instruction memory module

Data Memory (datamemory.v)

The data memory has 32 memory locations of 32 bits each. It is capable of writing and reading from this memory. Reading is always taking place, whereas writing only occurs when the memwrite input from the primary control is asserted high. Figure 5 shows the correct functioning of the data memory module. The VWF shows the accessing of memory location "4" (index 1) and the output is in "outdata" output. The writing functionality is shown in the third cycle where "memwrite" is asserted high and write data, "wdata" is 50. The reset functionality is shown near the end of the VWF where the reset line is high and the value in memory location 8 is reset to 300. output"m" can be ignored, it was only used for debugging and is used to show that the output of the mips code is stored mipmarin the correct location and has the correct value.

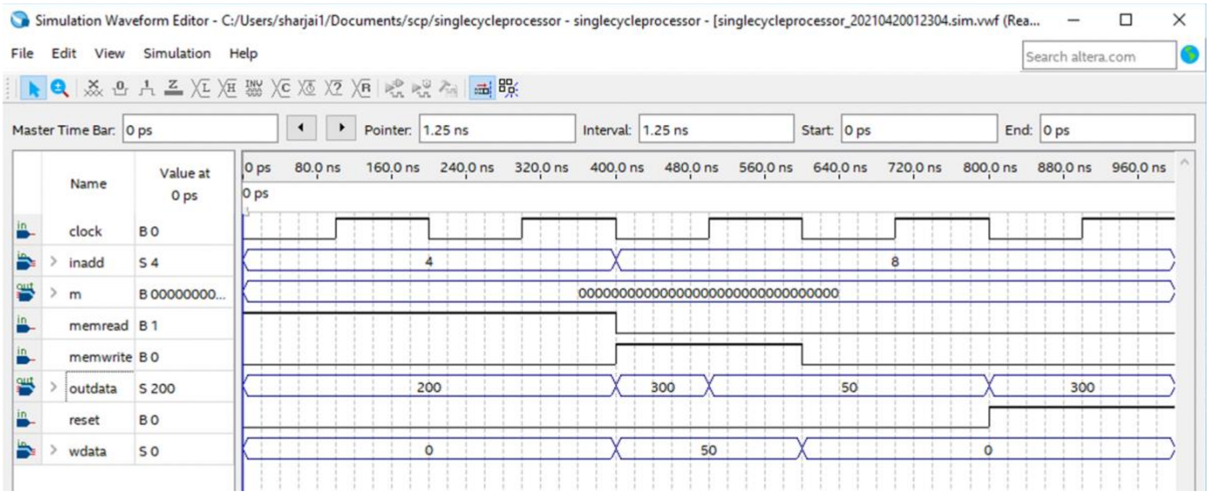
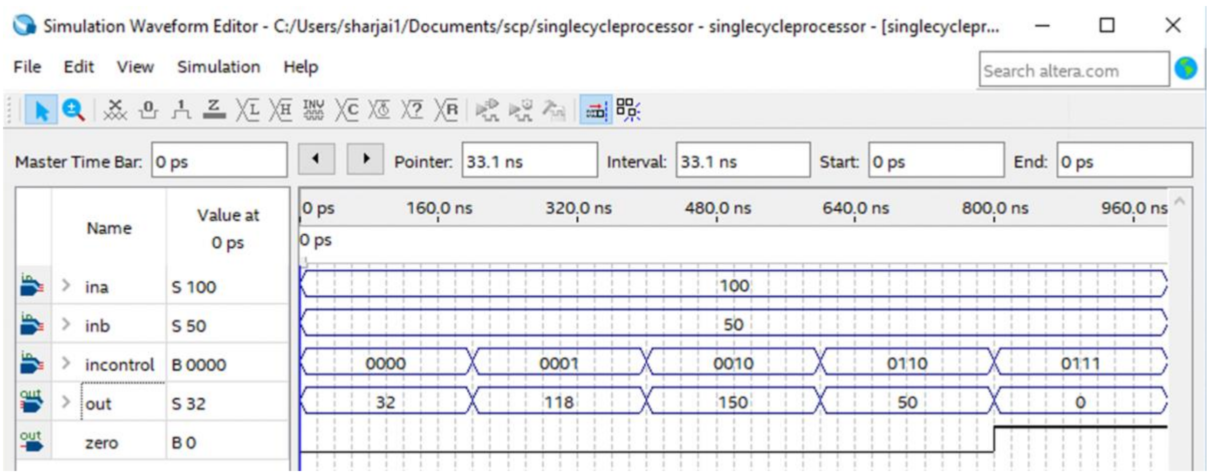


Figure 5: datamemory.vwf showing the correct output for the data memory module

ALU (alu.v)

The ALU module is responsible for doing arithmetic on two sets of 32 bit inputs. It performs 5 different kinds of operations based on the 4 bit input from the ALU Control. Figure 6 shows the correct output for the module. The five different values correspond to instructions for "AND", "OR", "Addition", "Subtraction", and "Set Less Than" Respectively. The correct output is shown in the "out" output field. The correct functioning of zero is shown for the last case where "out" is zero.


Simulation Waveform Editor - C:/Users/sharjai1/Documents/scp/singlecycleprocessor - singlecycleprocessor - [singlecyclepr...
File Edit View Simulation Help
Search altera.com

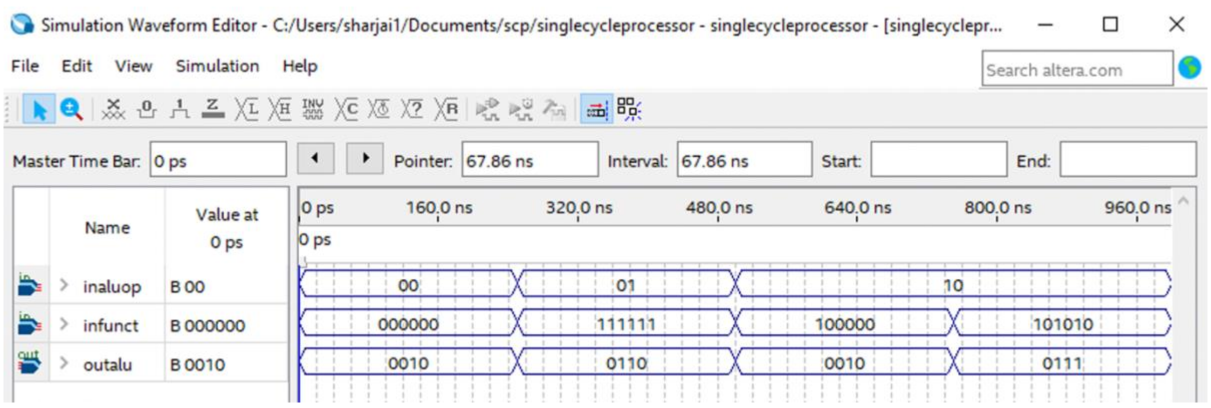
```
Master Time Bar: 0 ps    Pointer: 33.1 ns    Interval: 33.1 ns    Start: 0 ps    End: 0 ps

Name        Value at 0 ps   0 ps    160.0 ns    320.0 ns    480.0 ns    640.0 ns    800.0 ns    960.0 ns
ina         S 100           100
inb         S 50            50
incontrol   B 0000          0000        0001        0010        0110        0111
out         S 32            32          118         150         50          0
zero        B 0
```

Figure 6: alu.vwf showing the correct output for the ALU Module

ALU Control (alucontrol.v)

The ALU Control module is responsible for selecting the type of calculation that the ALU does. It takes in an ALU op code from the primary control as well as a funct value if it is doing a r-type arithmetic calculation. Figure 7 shows the correct “outalu” output selection for what goes to the ALU based on different combinations of “inaluop” and “infunct”.


Simulation Waveform Editor - C:/Users/sharjai1/Documents/scp/singlecycleprocessor - singlecycleprocessor - [singlecyclepr...
File Edit View Simulation Help
Search altera.com

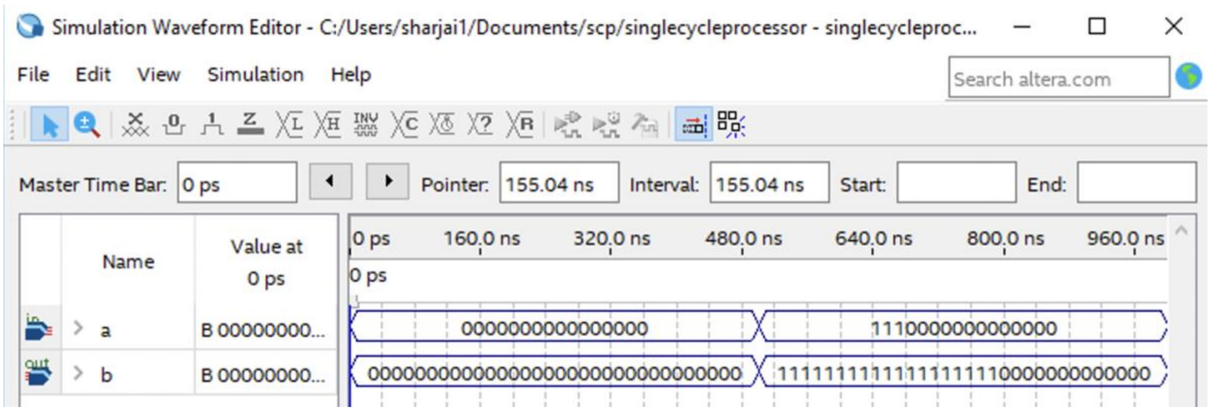
```
Master Time Bar: 0 ps    Pointer: 67.86 ns    Interval: 67.86 ns    Start:    End:

Name      Value at 0 ps   0 ps    160.0 ns    320.0 ns    480.0 ns    640.0 ns    800.0 ns    960.0 ns
inaluop   B 00            00          01          10
infunct   B 000000        000000      111111      100000      101010
outalu    B 0010          0010        0110        0010        0111
```

Figure 7: alucontrol.vwf showing the correct output for the ALU Control Module

Sign Extender (signextender.v)

The sign extender module is responsible for extending the most significant bit of a 16 bit input to the more significant 16 bits of a 32 bit output. Figure 8 shows the correct output, “b”, for two different combinations of “a”.

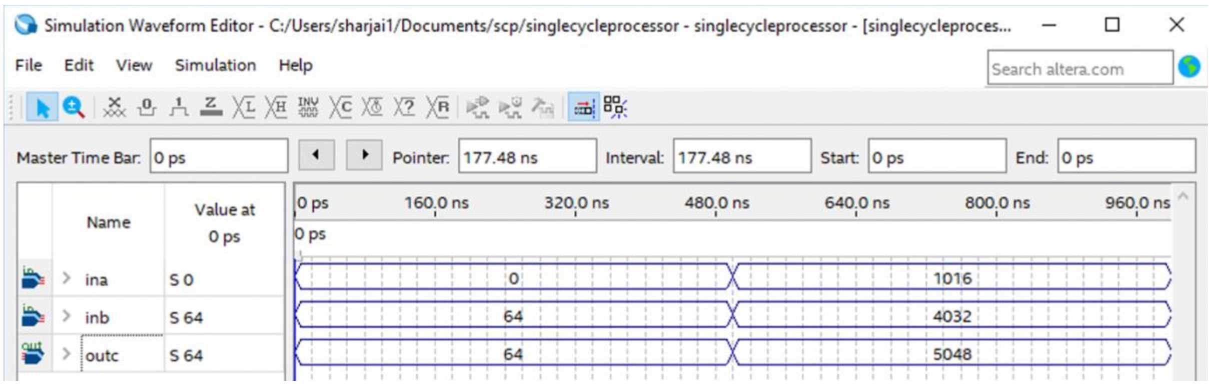

Simulation Waveform Editor - C:/Users/sharjai1/Documents/scp/singlecycleprocessor - singlecycleproc...
File Edit View Simulation Help
Search altera.com

```
Master Time Bar: 0 ps    Pointer: 155.04 ns    Interval: 155.04 ns    Start:    End:

Name   Value at 0 ps    0 ps    160.0 ns    320.0 ns    480.0 ns    640.0 ns    800.0 ns    960.0 ns
a      B 00000000...    0000000000000000                    1110000000000000
b      B 00000000...    00000000000000000000000000000000    11111111111111111110000000000000
```

Figure 8: signextender.vwf showing the correct output for the Sign Extender Module

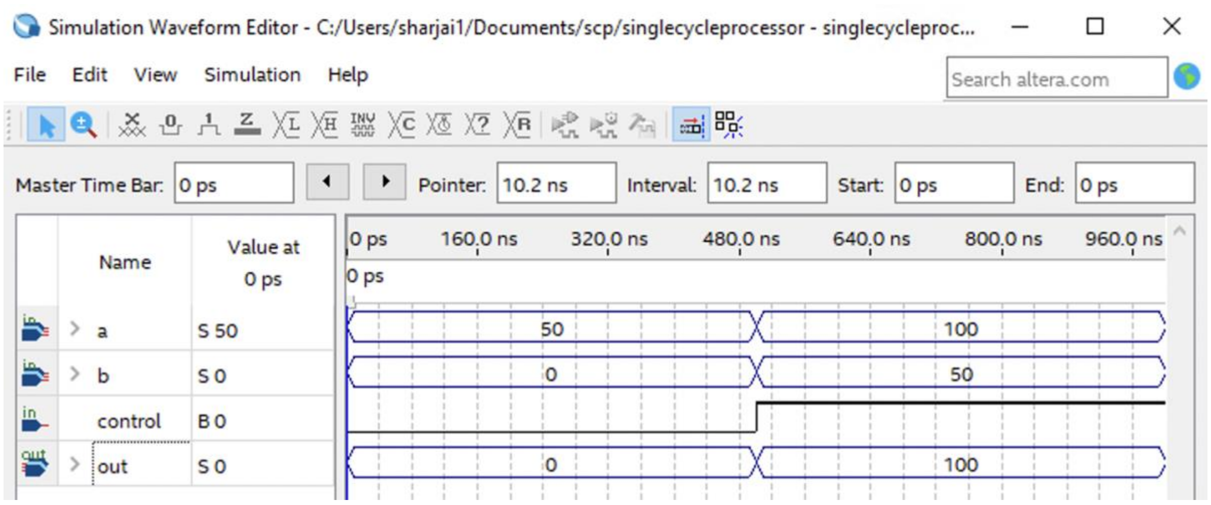
Thirty-Two Bit Adder (thirtytwobitadder.v)
The thirty two bit adder module adds two thirty two bit numbers and outputs the result. The correct output "outc" is shown in Figure 9 for two different combinations of inputs "ina" and "inb".



Name	Value at 0 ps	0 – ~480 ns	~480 – 1000 ns
ina	S 0	0	1016
inb	S 64	64	4032
outc	S 64	64	5048

Figure 9: thirtytwobitadder.vwf showing the correct output for the Thirty-Two Bit Adder

Thirty-Two Bit Mux (mux.v)
The thirty two bit mux module selects a 32 bit output based on the control input. Figure 10 shows the selection of "a" or "b" inputs and then outputting them to "out" based on the input "control".



Name	Value at 0 ps	0 – ~500 ns	~500 – 1000 ns
a	S 50	50	100
b	S 0	0	50
control	B 0	0	1
out	S 0	0	100

Figure 10: mux.vwf showing the correct output for the Thirty-Two Bit Mux Module

Five Bit Mux (fivebitmux.v)
The five bit mux module selects a 5 bit output based on the control input. Figure 11 shows the selection of "a" or "b" inputs and then outputting them to "out" based on the input "control". This is the same as the Thirty-Two Bit Module except it is for 5 bits.

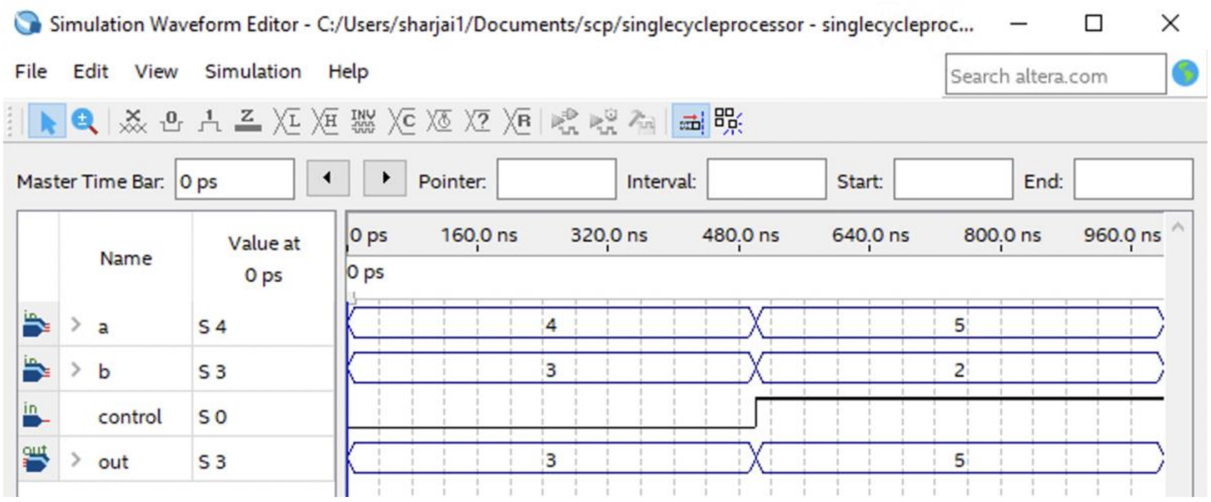

Simulation Waveform Editor - C:/Users/sharjai1/Documents/scp/singlecycleprocessor - singlecycleproc...
File Edit View Simulation Help
Search altera.com
Master Time Bar: 0 ps
Pointer:
Interval:
Start:
End:
Name
Value at 0 ps
0 ps 160.0 ns 320.0 ns 480.0 ns 640.0 ns 800.0 ns 960.0 ns
a S 4
b S 3
control S 0
out S 3
4
5
3
2
3
5

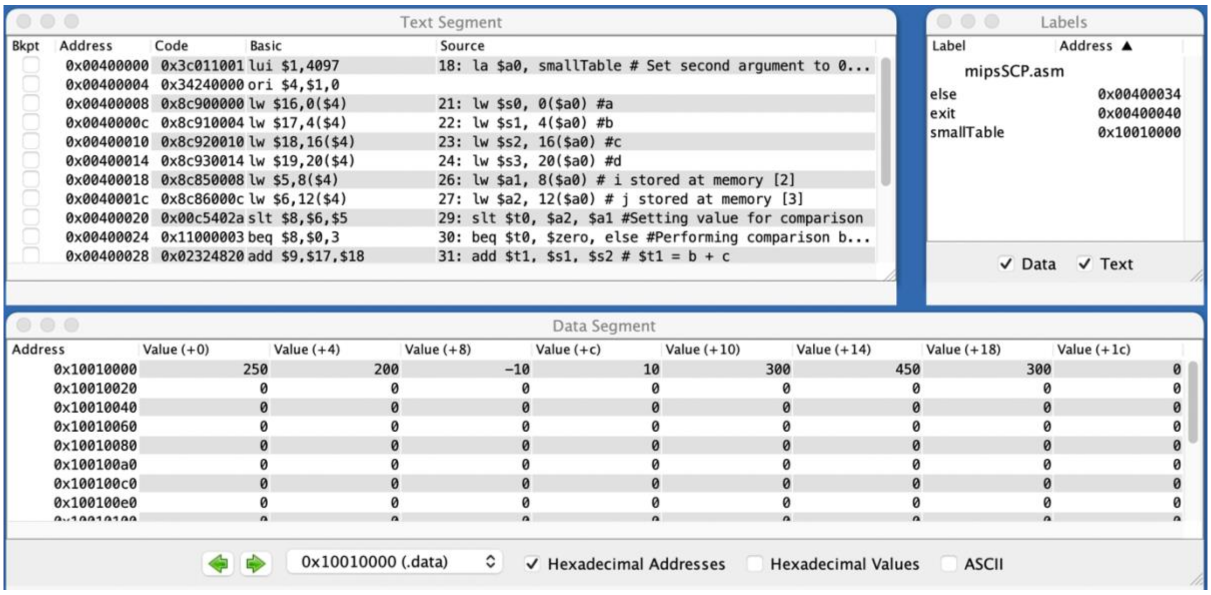
Figure 11: fivebitmux.vwf showing the correct output for the Five Bit Mux Module

Single Cycle Processor (singlecycleprocessor.v)
The functioning of the whole processor which is the Single Cycle Processor Module will be shown in the next section in Figure 13a/b.

MIPS Code Explanation

The MIPS Code starts by setting 0 as the base address to reference variables in data memory from. Then it loads a,b,c,d,i,j into registers. I loaded i and j into index 2 and 3 respectively which is location 8 and 12 in byte addressing. After that, the set less than and beq was used to replicate the i > j if statement. Based on this beq and another beq in replacement of a jump, the path followed is chosen. At the end, after all the arithmetic is done, the final value is stored in data memory at location 24, index 6. MIPS code is saved as mipsSCP.asm.

The correct execution of the MIPS code is shown in Figure 12a/b and the correct execution of the Verilog code is shown in Figure 13a/b. Figure 13b also shows the functionality of the reset line.



Text Segment

```
Bkpt Address    Code       Basic              Source
     0x00400000 0x3c011001 lui $1,4097        18: la $a0, smallTable # Set second argument to 0...
     0x00400004 0x34240000 ori $4,$1,0
     0x00400008 0x8c900000 lw $16,0($4)       21: lw $s0, 0($a0) #a
     0x0040000c 0x8c910004 lw $17,4($4)       22: lw $s1, 4($a0) #b
     0x00400010 0x8c920010 lw $18,16($4)      23: lw $s2, 16($a0) #c
     0x00400014 0x8c930014 lw $19,20($4)      24: lw $s3, 20($a0) #d
     0x00400018 0x8c850008 lw $5,8($4)        26: lw $a1, 8($a0) # i stored at memory [2]
     0x0040001c 0x8c86000c lw $6,12($4)       27: lw $a2, 12($a0) # j stored at memory [3]
     0x00400020 0x00c5402a slt $8,$6,$5      29: slt $t0, $a2, $a1 #Setting value for comparison
     0x00400024 0x11000003 beq $8,$0,3        30: beq $t0, $zero, else #Performing comparison b...
     0x00400028 0x02324820 add $9,$17,$18     31: add $t1, $s1, $s2 # $t1 = b + c
```

Labels

Label	Address ▲
mipsSCP.asm	
else	0x00400034
exit	0x00400040
smallTable	0x10010000

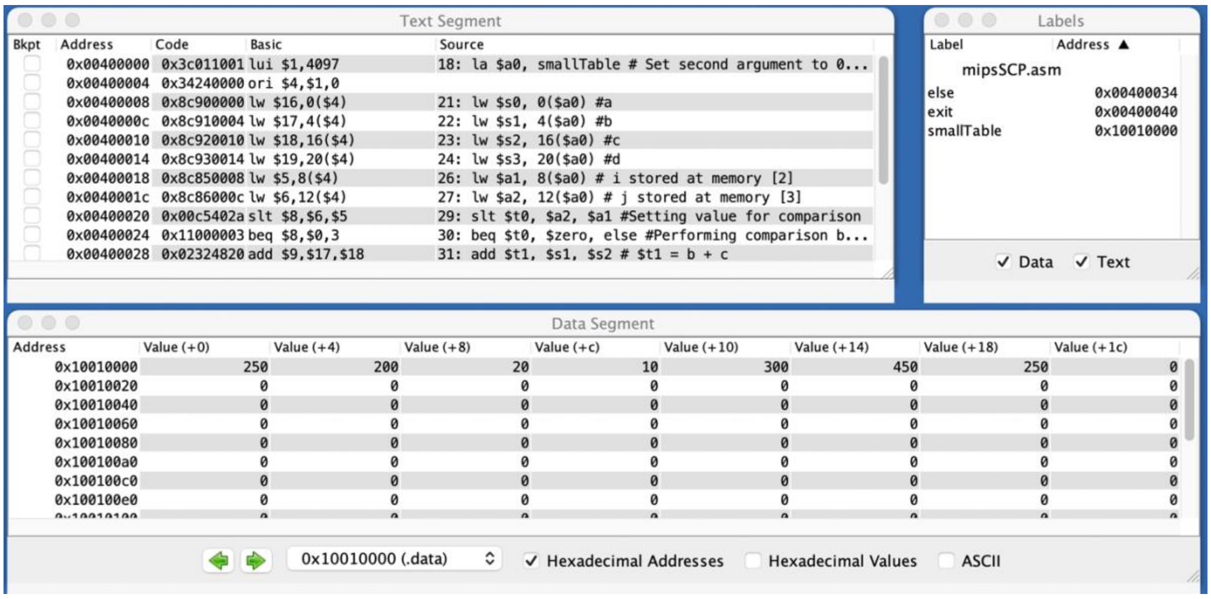
✓ Data ✓ Text

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	250	200	-10	10	300	450	300	0
0x10010020	0	0	0	0	0	0	0	0
0x10010040	0	0	0	0	0	0	0	0
0x10010060	0	0	0	0	0	0	0	0
0x10010080	0	0	0	0	0	0	0	0
0x100100a0	0	0	0	0	0	0	0	0
0x100100c0	0	0	0	0	0	0	0	0
0x100100e0	0	0	0	0	0	0	0	0

0x10010000 (.data) ✓ Hexadecimal Addresses ☐ Hexadecimal Values ☐ ASCII

Figure 12a: Correct execution of MIPS code with f stored at index 6 for i = -10, j = 10



Text Segment

```
Bkpt Address    Code       Basic              Source
     0x00400000 0x3c011001 lui $1,4097        18: la $a0, smallTable # Set second argument to 0...
     0x00400004 0x34240000 ori $4,$1,0
     0x00400008 0x8c900000 lw $16,0($4)       21: lw $s0, 0($a0) #a
     0x0040000c 0x8c910004 lw $17,4($4)       22: lw $s1, 4($a0) #b
     0x00400010 0x8c920010 lw $18,16($4)      23: lw $s2, 16($a0) #c
     0x00400014 0x8c930014 lw $19,20($4)      24: lw $s3, 20($a0) #d
     0x00400018 0x8c850008 lw $5,8($4)        26: lw $a1, 8($a0) # i stored at memory [2]
     0x0040001c 0x8c86000c lw $6,12($4)       27: lw $a2, 12($a0) # j stored at memory [3]
     0x00400020 0x00c5402a slt $8,$6,$5       29: slt $t0, $a2, $a1 #Setting value for comparison
     0x00400024 0x11000003 beq $8,$0,3        30: beq $t0, $zero, else #Performing comparison b...
     0x00400028 0x02324820 add $9,$17,$18     31: add $t1, $s1, $s2 # $t1 = b + c
```

Labels

Label	Address ▲
mipsSCP.asm	
else	0x00400034
exit	0x00400040
smallTable	0x10010000

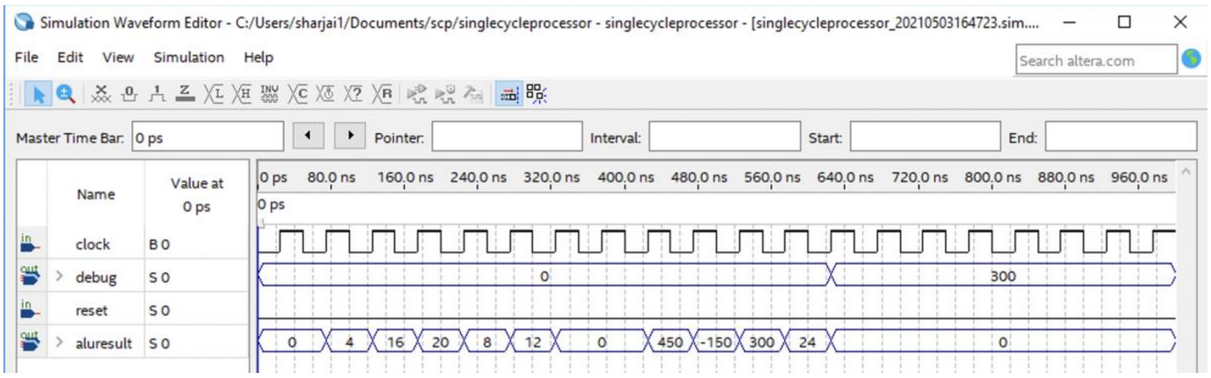
✓ Data ✓ Text

Data Segment

```
Address     Value (+0)  Value (+4)  Value (+8)  Value (+c)  Value (+10)  Value (+14)  Value (+18)  Value (+1c)
0x10010000         250         200          20          10          300          450          250            0
0x10010020           0           0           0           0            0            0            0            0
0x10010040           0           0           0           0            0            0            0            0
0x10010060           0           0           0           0            0            0            0            0
0x10010080           0           0           0           0            0            0            0            0
0x100100a0           0           0           0           0            0            0            0            0
0x100100c0           0           0           0           0            0            0            0            0
0x100100e0           0           0           0           0            0            0            0            0
```

0x10010000 (.data) ✓ Hexadecimal Addresses ☐ Hexadecimal Values ☐ ASCII

Figure 12b: Correct execution of MIPS code with f stored at index 6 for i = 20, j = 10

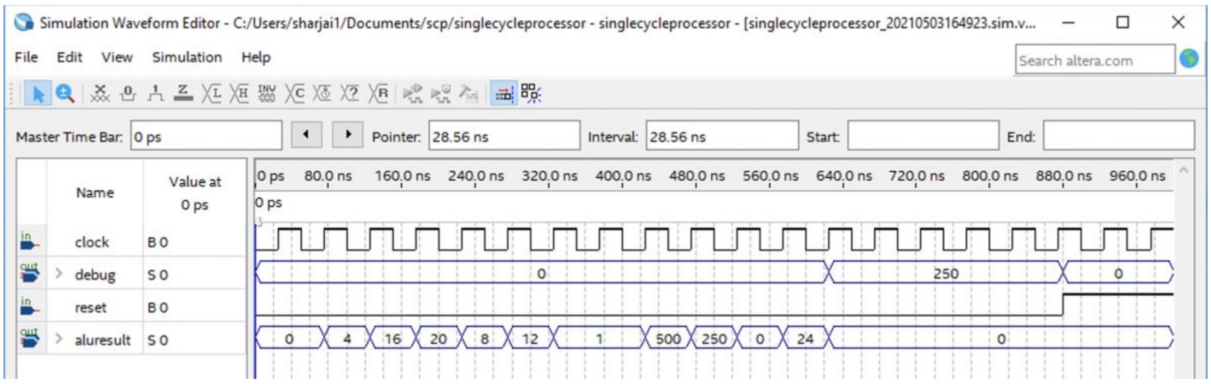


Simulation Waveform Editor - C:/Users/sharjai1/Documents/scp/singlecycleprocessor - singlecycleprocessor - [singlecycleprocessor_20210503164723.sim....

Master Time Bar: 0 ps Pointer: Interval: Start: End:

```
Name       Value at 0 ps   0 ps  80.0 ns  160.0 ns  240.0 ns  320.0 ns  400.0 ns  480.0 ns  560.0 ns  640.0 ns  720.0 ns  800.0 ns  880.0 ns  960.0 ns
clock      B 0
debug      S 0             0 | 300
reset      S 0
aluresult  S 0             0 | 4 | 16 | 20 | 8 | 12 | 0 | 450 | -150 | 300 | 24 | 0
```

Figure 13a: caseoneexecution.vwf showing the correct execution of Verilog code with f stored at index 6 for i = -10, j = 10, f is shown as debug output field. The aluresult field shows the output of the alu.



Simulation Waveform Editor - C:/Users/sharjai1/Documents/scp/singlecycleprocessor - singlecycleprocessor - [singlecycleprocessor_20210503164923.sim.v...

Master Time Bar: 0 ps Pointer: 28.56 ns Interval: 28.56 ns Start: End:

```
Name       Value at 0 ps   0 ps  80.0 ns  160.0 ns  240.0 ns  320.0 ns  400.0 ns  480.0 ns  560.0 ns  640.0 ns  720.0 ns  800.0 ns  880.0 ns  960.0 ns
clock      B 0
debug      S 0             0 | 250 | 0
reset      B 0
aluresult  S 0             0 | 4 | 16 | 20 | 8 | 12 | 1 | 500 | 250 | 0 | 24 | 0
```

Figure 13b: casetwoexecution.vwf showing the correct execution of Verilog code with f stored at index 6 for i = 20, j = 10, f is shown as debug output field. The aluresult field shows the output of the alu.