

## Design

### -ALU

The ALU described in the first lab was used as the ALU in our processor.

### -IFU

The instruction fetch unit was made following the design from lecture eight. The IFU takes in a clock signal, initialization control signal, 16-bit immediate, and two signals to check if the branch should be taken. The IFU outputs a 32-bit address that the instruction memory uses to get the current instruction. A 30-bit PC register that writes on each clock cycle was made, as well as an n-bit extender. The n-bit extender was chosen so that it could be used for the 30-bit immediate extension in the IFU and the immediate extension for i-type instructions as well as loads and stores. A 30-bit pc was used because the bottom bits of the address out are 00 in all cases. The upper 30 bits of the address out are PC+4 when the two branch signals (zero and branch in our implementation) are high, which is what should happen when a branch is not taken. When the branch is taken, the zero and branch signals are high, and the upper 30 bits of the target address are calculated by sign extending the immediate to 30 bits and, and adding it to PC+4. The main issues faced with the IFU was initializing the value in the PC. At first, we implemented the IFU as a closed loop, only taking in a clock signal, and there was no way to set a starting value. This was fixed by adding an initial value constant and an init signal, that is high for the first cycle and then low at all other times. This init signal is used as the select to a mux placed before the PC, which selects either the initial value or the new value of PC.

### -Register File

The register file takes in a clock; registers a, b and w; Bus W; and write enable. It outputs Busses A and B. Internally it maintains 31 32-bit registers. It determines the write enable bit for each internal register by decoding RW and ANDing the write enable input bitwise with the resulting vector. Each internal register is then fed the correct write enable bit from the vector. The outputs for bus A and bus B are determined by using 32 32 to 1 muxes with bits from each register as input and RA and RB as select signals. Since the zero register always contains zero in MIPS register zero is just statically allocated a value of zero. The main problem for this component was making sure that the zero register couldn't be written to without complicated logic. This was solved by removing the zero 'register' and replacing it with a constant signal.

### -Datapath

The datapath component takes in all necessary control signals as input and outputs the current instruction. The datapath contains muxes, a register file, an extender, an ALU, an IFU, instruction memory and data memory. These are hooked together in almost exactly the same way as shown on slide 15 of lecture 8. The key differences are the lack of a jump signal and the addition of a few muxes. The first additional mux chooses RA to be Rs or Rt based on whether or not the current instruction is a shift. The next mux chooses the input of the extender to be either the last 16 bits of the instruction if it is not a shift or bits 10 – 6 if it is a shift. The next mux chooses the input of the ALU to be either the output of the ALUsrc mux for any instruction but shift or the output of the extender if it is a shift. This was needed because shift is considered to be an r-type instruction but needed input from the immediate field. The next mux chooses either the output of the previously described mux or 0 as the

input for the ALU based on whether the instruction is BGTZ since the bus b input to the ALU needs to be zero in this case. The final additional mux picks the zero signal input to the IFU to be either the ALU zero signal, NOT ALU zero, or NOT ALU zero AND NOT ALU msb depending on whether the instruction is beq, bne or bgtz, respectively. The main issue encountered in the datapath component was finding out that certain instructions needed different inputs than what we originally thought (looking at you SLL). This was solved through the use of the muxes described above.

#### **-Control (See end of document for table)**

The control was divided into two components, the ALU control and the main control. Both were implemented using a PLA, because it would be simple to find the necessary output signals and would not require complicated logic. A PLA inverter and six-to-one AND gates were made to implement the PLA. The inverter works by taking in the six bits of data, as well as six bits that correspond to whether that bit of data needs to be inverted. If  $inv(i)$  is 1, then  $data(i)$  will be inverted before it is put into the AND gate. One note about this is that they are currently separate components (the PLA inverter, and the AND6to1), however, it would have been simpler to combine these two into one component for the PLA, rather than having one of each for each minterm, but I realized it too late. The current implementation still functions, properly but is not the most efficient in terms of the number of lines of code required. For the ALU control, the inputs are the opcode and the function code and the output is the necessary function for the ALU. Espresso was used to minimize the logic of the PLA's. The R-type instructions looked at the function codes and found the output based on these, and the non R-type instructions looked at the opcode. Muxes were used to select either the output from the R-type PLA, or the non R-type PLA, with the select bit being all the opcode bits OR'd with each other, and the result inverted. This is because the opcode for R-type instructions is 000000, so the select will only be 1, when the instruction is R-type. The main issues we encountered with the ALU control, was typos in the .pla file used by Espresso. For the SLT and SLTU, we initially had the output as the subtraction ALU opcode, instead of the SLT or SLTU opcode. This generated the wrong PLA and had to be corrected.

The main control was implemented the same way as the ALU control for simplicity. The main control takes in the opcode and generates the nine control signals using a PLA. The output signals are: RegDst, RegWr, Branch, ExtOp, AluSrc, MemWr, MemtoReg, BrSel1, BrSel0. Most of the control signals are the same as in the lecture; however, we needed to add a 2-bit branch select signal, that would feed in the appropriate value to the zero input of the IFU, depending on the type of branch. BNE, BEQ, and BGTZ all have different conditions in which they should branch, so they were fed into a 3-to-1 mux with BrSel as the two select bits, and the appropriate signal was passed to the IFU. We encountered problems similar to those from the ALU control when creating the main control. The main issues were typos or writing down the wrong control signal that needed to be generated before running Espresso. However, once the skeleton of the main control was made, these errors were not too difficult to fix. Both the controls, were able to be exhaustively tested due to the small number of inputs and outputs.

#### **-Single Cell Processor**

The SSP component takes in a clock and a pcInit signal. The pc init signal writes 00400020 into the program counter. This component was very simple since we created the control and the datapath as separate components. It consists of a main control and ALU control and the datapath component. The control units take in the pieces of the current instruction they need and output the control signals which are then fed into the datapath component, which outputs the current instruction. There were no major issues in the construction of this component.

## Program Traces

We determined the correct output of each program by converting it to assembly and running through them by hand. The hand 'run' programs compared to their actual traces are shown below. The programs were modified to load the results from memory to registers for easy access.

### Unsigned Sum

```
ADD $a1 $zero $zero      a1 = 0
ADDI $a3 $zero 0x1000    a3 = 0x1000
SLL $a3 $a3 0x10000      a3 = 0x10000000
ADD $a2 $a3 $zero        a2 = a3
ADDI $a2 $a2 0x0028      a2 = 0x10000028
LW $a0 0x0000 $a3        a0 = 0000000f
ADDU $a1 $a1 $a0         a1 = 0000000f
ADDI $a3 $a3 0x0004      a3 = 0x10000004
BNE $a3 $a2 0xFFFF
SW $a1 0x0000 $a3
```

--Sums over elements 10000000 to 100000024 stores in 10000028  
 --Output is correct if 10000028 contains ffffffff

Initial few cycles of operation:

	Msgs	
/ssp_tb/clk_tb	0	
/ssp_tb/single_cycle_processor_i/datapath_i/InstrFU/PC/dout	XXXXXXXX	XXXX... 00100008 00100009 0010000A 0010000B 0010000C 0010000D
/ssp_tb/single_cycle_processor_i/datapath_i/InstructionMem/dout	XXXXXXXX	XXXX... 00002820 20071000 00E73C00 00E03020 20C60028 8CE40000
/ssp_tb/single_cycle_processor_i/datapath_i/DataMem/cs	1	
/ssp_tb/single_cycle_processor_i/datapath_i/DataMem/oe	1	
/ssp_tb/single_cycle_processor_i/datapath_i/DataMem/we	U	
/ssp_tb/single_cycle_processor_i/datapath_i/DataMem/addr	XXXXXXXX	XXXX... 00000000 00001000 10000000 10000000 10000028 10000000
/ssp_tb/single_cycle_processor_i/datapath_i/DataMem/din	XXXXXXXX	XXXX... 00000000 XXXXXXXX 00001000 00000000 10000000 XXXXXXXX
/ssp_tb/single_cycle_processor_i/datapath_i/DataMem/dout	XXXXXXXX	XXXXXXXX 0000000F 0000000F 00000000 0000000F 0000000F
/ssp_tb/single_cycle_processor_i/datapath_i/Registers/reg_out	{XXXXXXXX} {XXXXX...}	{XXXXXXXX} {XXXXXXXX} ... {XXXXXXXX} fx... {XXXXXXXX} fx... {XXXXXXXX} fx... {XXXXXXXX} fx... {XXXXXXXX} fx...
(31)	XXXXXXXX	XXXXXXXX
(30)	XXXXXXXX	XXXXXXXX
(29)	XXXXXXXX	XXXXXXXX
(28)	XXXXXXXX	XXXXXXXX
(27)	XXXXXXXX	XXXXXXXX
(26)	XXXXXXXX	XXXXXXXX
(25)	XXXXXXXX	XXXXXXXX
(24)	XXXXXXXX	XXXXXXXX
(23)	XXXXXXXX	XXXXXXXX
(22)	XXXXXXXX	XXXXXXXX
(21)	XXXXXXXX	XXXXXXXX
(20)	XXXXXXXX	XXXXXXXX
(19)	XXXXXXXX	XXXXXXXX
(18)	XXXXXXXX	XXXXXXXX
(17)	XXXXXXXX	XXXXXXXX
(16)	XXXXXXXX	XXXXXXXX
(15)	XXXXXXXX	XXXXXXXX
(14)	XXXXXXXX	XXXXXXXX
(13)	XXXXXXXX	XXXXXXXX
(12)	XXXXXXXX	XXXXXXXX
(11)	XXXXXXXX	XXXXXXXX
(10)	XXXXXXXX	XXXXXXXX
(9)	FFFFFFFF	XXXXXXXX
(8)	10000000	XXXXXXXX
(7)	10000028	XXXXXXXX 00001000 10000000
(6)	10000028	XXXXXXXX 10000000 10000028
(5)	FFFFFFFF	XXXXXXXX 00000000
(4)	C0000000	XXXXXXXX
(3)	XXXXXXXX	XXXXXXXX
(2)	XXXXXXXX	XXXXXXXX
(1)	XXXXXXXX	XXXXXXXX
(0)	00000000	00000000

A few cycles mid operation:

	Msgs	
/ssp_tb/clk_tb	0	
/ssp_tb/single_cycle_processor_i/datapath_i/InstrFU/PC/dout	XXXXXXXX	0010000E 0010000F 00100010 0010000D 0010000E 0010000F 00100010
/ssp_tb/single_cycle_processor_i/datapath_i/InstructionMem/dout	XXXXXXXX	00A42821 20E70004 14E6FFFC 8CE40000 00A42821 20E70004 14E6FFFC
/ssp_tb/single_cycle_processor_i/datapath_i/DataMem/cs	1	
/ssp_tb/single_cycle_processor_i/datapath_i/DataMem/oe	1	
/ssp_tb/single_cycle_processor_i/datapath_i/DataMem/we	U	
/ssp_tb/single_cycle_processor_i/datapath_i/DataMem/addr	XXXXXXXX	000FFFFF 10000014 FFFFFFFE 10000014 00FFFFFF 10000018 FFFFFFF0
/ssp_tb/single_cycle_processor_i/datapath_i/DataMem/din	XXXXXXXX	000F0000 10000010 10000028 000F0000 00F00000 10000014 10000028
/ssp_tb/single_cycle_processor_i/datapath_i/DataMem/dout	XXXXXXXX	XXXXXXXX 00F00000 XXXXXXXX 00F00000 XXXXXXXX 0F000000 XXXXXXXX
/ssp_tb/single_cycle_processor_i/datapath_i/Registers/reg_out	{XXXXXXXX} {XXXXX...}	{XXXXXXXX} {X...} {XXXXXXXX} {X...} {XXXXXXXX} {XXXXXXXX} {XXXXXXXX} {XXXXXXXX} {XXXXXXXX} {X...} {XXXXXXXX} {X...} {XXXXXXXX} {X...} {XXXXXXXX} {X...} {XXXXXXXX} {X...}
(31)	XXXXXXXX	XXXXXXXX
(30)	XXXXXXXX	XXXXXXXX
(29)	XXXXXXXX	XXXXXXXX
(28)	XXXXXXXX	XXXXXXXX
(27)	XXXXXXXX	XXXXXXXX
(26)	XXXXXXXX	XXXXXXXX
(25)	XXXXXXXX	XXXXXXXX
(24)	XXXXXXXX	XXXXXXXX
(23)	XXXXXXXX	XXXXXXXX
(22)	XXXXXXXX	XXXXXXXX
(21)	XXXXXXXX	XXXXXXXX
(20)	XXXXXXXX	XXXXXXXX
(19)	XXXXXXXX	XXXXXXXX
(18)	XXXXXXXX	XXXXXXXX
(17)	XXXXXXXX	XXXXXXXX
(16)	XXXXXXXX	XXXXXXXX
(15)	XXXXXXXX	XXXXXXXX
(14)	XXXXXXXX	XXXXXXXX
(13)	XXXXXXXX	XXXXXXXX
(12)	XXXXXXXX	XXXXXXXX
(11)	XXXXXXXX	XXXXXXXX
(10)	XXXXXXXX	XXXXXXXX
(9)	FFFFFFFF	XXXXXXXX
(8)	10000000	XXXXXXXX
(7)	10000028	10000010 10000014 10000018
(6)	10000028	10000028
(5)	FFFFFFFF	0000FFFF 000FFFFF 00FFFFFF
(4)	C0000000	000F0000 00F00000
(3)	XXXXXXXX	XXXXXXXX
(2)	XXXXXXXX	XXXXXXXX
(1)	XXXXXXXX	XXXXXXXX
(0)	00000000	00000000
/ssp_tb/single_cycle_processor_i/datapath_i/DataMem/mem_file	unsigned_sum.dat	unsigned sum.dat

Final few cycles of operation:

	Msgs	
/ssp_tb/clk_tb	0	
/ssp_tb/single_cycle_processor_i/datapath_i/InstrFU/PC/dout	XXXXXXXX	00100010 00100011 00100012 00100013 00100014 00100015
/ssp_tb/single_cycle_processor_i/datapath_i/InstructionMem/dout	XXXXXXXX	14E6FFFC ACE50000 20081000 00084400 8D090028 XXXXXXXX
/ssp_tb/single_cycle_processor_i/datapath_i/DataMem/cs	1	
/ssp_tb/single_cycle_processor_i/datapath_i/DataMem/oe	1	
/ssp_tb/single_cycle_processor_i/datapath_i/DataMem/we	U	
/ssp_tb/single_cycle_processor_i/datapath_i/DataMem/addr	XXXXXXXX	00000000 10000028 00001000 10000000 10000028 XXXXXXXX
/ssp_tb/single_cycle_processor_i/datapath_i/DataMem/din	XXXXXXXX	10000028 FFFFFFFF XXXXXXXX 00001000 XXXXXXXX
/ssp_tb/single_cycle_processor_i/datapath_i/DataMem/dout	XXXXXXXX	XXXXXXXX FFFFFFFF XXXXXXXX 0000000F FFFFFFFF XXXXXXXX
/ssp_tb/single_cycle_processor_i/datapath_i/Registers/reg_out	{XXXXXXXX} {XXXXX...}	{XXXXXXXX} {XXXXXXXX} {XXXXXXXX} {XXXXXXXX} {X...} {XXXXXXXX} {X...} {XXXXXXXX} {X...} {XXXXXXXX} {X...} {XXXXXXXX} {X...} {XXXXXXXX} {X...}
(31)	XXXXXXXX	XXXXXXXX
(30)	XXXXXXXX	XXXXXXXX
(29)	XXXXXXXX	XXXXXXXX
(28)	XXXXXXXX	XXXXXXXX
(27)	XXXXXXXX	XXXXXXXX
(26)	XXXXXXXX	XXXXXXXX
(25)	XXXXXXXX	XXXXXXXX
(24)	XXXXXXXX	XXXXXXXX
(23)	XXXXXXXX	XXXXXXXX
(22)	XXXXXXXX	XXXXXXXX
(21)	XXXXXXXX	XXXXXXXX
(20)	XXXXXXXX	XXXXXXXX
(19)	XXXXXXXX	XXXXXXXX
(18)	XXXXXXXX	XXXXXXXX
(17)	XXXXXXXX	XXXXXXXX
(16)	XXXXXXXX	XXXXXXXX
(15)	XXXXXXXX	XXXXXXXX
(14)	XXXXXXXX	XXXXXXXX
(13)	XXXXXXXX	XXXXXXXX
(12)	XXXXXXXX	XXXXXXXX
(11)	XXXXXXXX	XXXXXXXX
(10)	XXXXXXXX	XXXXXXXX
(9)	FFFFFFFF	XXXXXXXX
(8)	XXXXXXXX	XXXXXXXX
(7)	10000028	10000028 10000000 10000000
(6)	10000028	10000028
(5)	FFFFFFFF	FFFFFFFF
(4)	C0000000	C0000000
(3)	XXXXXXXX	XXXXXXXX
(2)	XXXXXXXX	XXXXXXXX
(1)	XXXXXXXX	XXXXXXXX
(0)	00000000	00000000
/ssp_tb/single_cycle_processor_i/datapath_i/DataMem/mem_file	unsigned_sum.dat	unsigned sum.dat

### Sort Corrected Branch

ADDI \$v0 \$zero 0x1000	v0 = 1000
SLL \$v0 \$v0 10000	v0 = 10000000
ADDI \$a0 \$v0 0x0024	a0 = 10000024
ADDI \$a1 \$v0 0x0028	a1 = 10000028
LW \$a3 0x0000 \$v0	a3 = 00000009
ADDI \$v1 \$v0 0x0004	v1 = 10000004
LW \$at 0x0000 \$v1	at = 0000000a    at = 00000008
SLT \$a2 \$a3 \$at	a2 = 1                    a2 = 0
BGTZ \$a2 0x0003	branch taken    branch not taken
SW \$at 0x0000 \$v0	10000000 <= 00000008
SW \$a3 0x0000 \$v1	10000008 <= 00000009
ADD \$a3 \$at \$zero	a3 = 00000008
ADDI \$v1 \$v1 0x0004	v1 = 10000008    v1 = 1000000c
BNE \$v1 \$a1 0xFFFF	branch taken
ADDI \$v0 \$v0 0x0004	
BNE \$v0 \$a0 0xFFFF	

--ITS BUBBLE SORT - ish

```
--Loops through 10000000 to 10000024 swapping data when it finds a larger item at a higher address.
```

```
--Completed process should have memory: 1,2,3,4,5,6,7,8,9,a
```

First few cycles of operation:

The screenshot shows the 'Messages' window in the Vivado IDE. The window is divided into three panes:

- Left Pane:** A list of messages with expand/collapse icons and a search bar. The messages are sorted by time, with the most recent at the top.
- Middle Pane:** Displays the message text, including file paths and register addresses. The messages are:
  - `/ssp_tb/clk_tb`
  - `/ssp_tb/single_cycle_processor_i/datapath_i/InstrFU/PC/dout`
  - `/ssp_tb/single_cycle_processor_i/datapath_i/InstructionMem/dout`
  - `/ssp_tb/single_cycle_processor_i/datapath_i/DataMem/cs`
  - `/ssp_tb/single_cycle_processor_i/datapath_i/DataMem/oe`
  - `/ssp_tb/single_cycle_processor_i/datapath_i/DataMem/we`
  - `/ssp_tb/single_cycle_processor_i/datapath_i/DataMem/addr`
  - `/ssp_tb/single_cycle_processor_i/datapath_i/DataMem/din`
  - `/ssp_tb/single_cycle_processor_i/datapath_i/DataMem/dout`
  - `/ssp_tb/single_cycle_processor_i/datapath_i/Registers/reg_out`
  - `/ssp_tb/single_cycle_processor_i/datapath_i/mem_file`
- Right Pane:** Shows a hex dump of the message data. The hex dump is organized into columns, with each column representing a different data source or register. The data is shown in hexadecimal format, with some values highlighted in red.

A few cycles mid operation:

	Msgs	
/ssp_tb/dk_tb	0	
/ssp_tb/single_cycle_processor_j/datapath_j/InstrFU/PC/dout	0010001A	001... 0010000E 0010000F 00100010 00100011 00100012 00100013 00100014 00100015
/ssp_tb/single_cycle_processor_j/datapath_j/InstructionMem/dout	8D090000	146... 8C610000 00E1302A 1CC00003 AC410000 AC670000 00203820 20630004 1465FFFF
/ssp_tb/single_cycle_processor_j/datapath_j/DataMem/cs	1	
/ssp_tb/single_cycle_processor_j/datapath_j/DataMem/oe	1	
/ssp_tb/single_cycle_processor_j/datapath_j/DataMem/we	0	
/ssp_tb/single_cycle_processor_j/datapath_j/DataMem/addr	10000000	FFF... 1000001C 00000000 00000000 10000000 1000001C 00000001 10000020 FFFFFFFF
/ssp_tb/single_cycle_processor_j/datapath_j/DataMem/din	XXXXXXXX	100... 00000004 00000001 00000000 00000001 00000004 00000000 1000001C 10000028
/ssp_tb/single_cycle_processor_j/datapath_j/DataMem/dout	00000001	XXX... 00000001 XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX
/ssp_tb/single_cycle_processor_j/datapath_j/Registers/reg_out	{XXXXXXXX} {XXXXX...}	{XXXXXXXX} {XXXXXXXX} {XXXXXXXX} {XXXXXXXX} {XXXXXXXX} {XXXXXXXX} {XXXXXXXX} {XXXXX...} {XXXXXXXX} {X...} {XXXXXXXX}
(31)	XXXXXXXX	XXXXXXXX
(30)	XXXXXXXX	XXXXXXXX
(29)	XXXXXXXX	XXXXXXXX
(28)	XXXXXXXX	XXXXXXXX
(27)	XXXXXXXX	XXXXXXXX
(26)	XXXXXXXX	XXXXXXXX
(25)	XXXXXXXX	XXXXXXXX
(24)	XXXXXXXX	XXXXXXXX
(23)	XXXXXXXX	XXXXXXXX
(22)	XXXXXXXX	XXXXXXXX
(21)	XXXXXXXX	XXXXXXXX
(20)	XXXXXXXX	XXXXXXXX
(19)	XXXXXXXX	XXXXXXXX
(18)	XXXXXXXX	XXXXXXXX
(17)	XXXXXXXX	XXXXXXXX
(16)	XXXXXXXX	XXXXXXXX
(15)	XXXXXXXX	XXXXXXXX
(14)	XXXXXXXX	XXXXXXXX
(13)	XXXXXXXX	XXXXXXXX
(12)	XXXXXXXX	XXXXXXXX
(11)	XXXXXXXX	XXXXXXXX
(10)	XXXXXXXX	XXXXXXXX
(9)	XXXXXXXX	XXXXXXXX
(8)	10000000	XXXXXXXX
(7)	00000009	00000004
(6)	00000000	00000000
(5)	10000028	10000028
(4)	10000024	10000024
(3)	10000028	1000001C
(2)	10000024	10000030
(1)	00000009	00000004
(0)	00000000	00000000
/ssp_tb/single_cycle_processor_j/datapath_j/DataMem/mem_file	sort_corrected_bra...	sort_corrected_branch.dat

The final few cycles:

	Msgs	
/ssp_tb/dk_tb	0	
/ssp_tb/single_cycle_processor_j/datapath_j/InstrFU/PC/dout	0010001A	0010001E 0010001F 00100020 00100021 00100022 00100023
/ssp_tb/single_cycle_processor_j/datapath_j/InstructionMem/dout	8D090000	8D0D0010 8D0E0014 8D0F0018 8D18001C 8D190020 XXXXXXXX
/ssp_tb/single_cycle_processor_j/datapath_j/DataMem/cs	1	
/ssp_tb/single_cycle_processor_j/datapath_j/DataMem/oe	1	
/ssp_tb/single_cycle_processor_j/datapath_j/DataMem/we	0	
/ssp_tb/single_cycle_processor_j/datapath_j/DataMem/addr	10000000	10000010 10000014 10000018 1000001C 10000020 XXXXXXXX
/ssp_tb/single_cycle_processor_j/datapath_j/DataMem/din	XXXXXXXX	XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX
/ssp_tb/single_cycle_processor_j/datapath_j/DataMem/dout	00000001	00000005 00000006 00000007 00000008 00000009 XXXXXXXX
/ssp_tb/single_cycle_processor_j/datapath_j/Registers/reg_out	{XXXXXXXX} {XXXXX...}	{XXXXXXXX} {XXXXXXXX} {X...} {XXXXXXXX} {X...} {XXXXXXXX} {X...} {XXXXXXXX} {X...} {XXXXXXXX} {XXXXX}
(31)	XXXXXXXX	XXXXXXXX
(30)	XXXXXXXX	XXXXXXXX
(29)	XXXXXXXX	XXXXXXXX
(28)	XXXXXXXX	XXXXXXXX
(27)	XXXXXXXX	XXXXXXXX
(26)	XXXXXXXX	XXXXXXXX
(25)	XXXXXXXX	XXXXXXXX
(24)	XXXXXXXX	XXXXXXXX
(23)	XXXXXXXX	XXXXXXXX
(22)	XXXXXXXX	XXXXXXXX
(21)	XXXXXXXX	XXXXXXXX
(20)	XXXXXXXX	XXXXXXXX
(19)	XXXXXXXX	XXXXXXXX
(18)	XXXXXXXX	XXXXXXXX
(17)	XXXXXXXX	XXXXXXXX
(16)	XXXXXXXX	XXXXXXXX
(15)	XXXXXXXX	XXXXXXXX
(14)	XXXXXXXX	XXXXXXXX
(13)	XXXXXXXX	XXXXXXXX
(12)	XXXXXXXX	XXXXXXXX
(11)	XXXXXXXX	XXXXXXXX
(10)	XXXXXXXX	XXXXXXXX
(9)	XXXXXXXX	XXXXXXXX
(8)	10000000	10000000
(7)	00000009	00000009
(6)	00000000	00000000
(5)	10000028	10000028
(4)	10000024	10000024
(3)	10000028	10000028
(2)	10000024	10000024
(1)	00000009	00000009
(0)	00000000	00000000
/ssp_tb/single_cycle_processor_j/datapath_j/DataMem/mem_file	sort_corrected_bra...	sort_corrected_branch.dat

The red box above contains (from bottom up) the memory locations 10000000 to 10000020. 1,2,3,4,5,6,7,8,9 checks out!

### ***Bills Branch***

ADDI \$a1 \$zero 0x0001	a1 = 0001	
ADDI \$a2 \$zero 0x0064	a2 = 0064	
ADDI \$v0 \$zero 0x1000	v0 = 1000	
SLL \$v0 \$v0 0x10000	v0 = 10000000	
ADDI \$a3 \$v0 0x0028	a3 = 10000028	
LW \$v1 0x0000 \$v0	v1 = 0000000a	v1 = 00000009
SLT \$a0 \$a2 \$v1	a0 = 0 (64 > a)	a0 = 0 (5a > 9)
BEQ \$a0 \$a1 0x0002	branch not taken	branch not taken
SUB \$a2 \$a2 \$v1	a2 = 5a	a2 = 51
SW \$zero 0x0000 \$v0	10000000 <= 0	10000004 <= 0
ADDI \$v0 \$v0 0x0004	v0 = 10000004	v0 = 100000008
BNE \$v0 \$a3 0xFFFF9	branch taken	branch taken
SW \$a2 0x0000 \$a3		

```
-- Iterates through 10000000 to 10000024 subtracting value found there from
-- a running total starting at 100, afterwards it writes 0 to the mem location
--If value is too large it skips it
--afterwards it stores the amount leftover into 10000028
-- Completed process should have memory: 0,0,0,2bc,0,0,190,0,0,0,38
```

The first few cycle of operation:

The screenshot displays the 'bills' branch in the 'bills' project. The left pane shows the file tree with the following structure:

- ./sssp\_tb/dk\_tb
- ./sssp\_tb/single\_cycle\_processor\_i/datapath\_i/InstrFUPC/dout
- ./sssp\_tb/single\_cycle\_processor\_i/datapath\_i/InstrMem/dout
- ./sssp\_tb/single\_cycle\_processor\_i/datapath\_i/DataMem/cs
- ./sssp\_tb/single\_cycle\_processor\_i/datapath\_i/DataMem/oe
- ./sssp\_tb/single\_cycle\_processor\_i/datapath\_i/DataMem/we
- ./sssp\_tb/single\_cycle\_processor\_i/datapath\_i/DataMem/addr
- ./sssp\_tb/single\_cycle\_processor\_i/datapath\_i/DataMem/din
- ./sssp\_tb/single\_cycle\_processor\_i/datapath\_i/DataMem/dout
- ./sssp\_tb/single\_cycle\_processor\_i/datapath\_i/Registers/reg\_out
- (31)
- (30)
- (29)
- (28)
- (27)
- (26)
- (25)
- (24)
- (23)
- (22)
- (21)
- (20)
- (19)
- (18)
- (17)
- (16)
- (15)
- (14)
- (13)
- (12)
- (11)
- (10)
- (9)
- (8)
- (7)
- (6)
- (5)
- (4)
- (3)
- (2)
- (1)
- (0)
- ./sssp\_tb/single\_cycle\_processor\_i/datapath\_i/DataMem/mem\_file

The right pane shows the 'bills\_branch.dat' file, which is a binary representation of a branch. The file content is a binary representation of a branch, with columns for 'bills\_branch.dat' and 'bills\_branch.dat'. The file is a binary representation of a branch, with columns for 'bills\_branch.dat' and 'bills\_branch.dat'.



A few cycles mid operation:

	Msgs	
/ssp_tb/dk_tb	1	
/ssp_tb/single_cycle_processor_i/datapath_i/InstrFU/PC/dout	00100012	0010000E 0010000F 00100010 00100011 00100012 00100013 0010000D
/ssp_tb/single_cycle_processor_i/datapath_i/InstructionMem/dout	20420004	00C3202A 10850002 00C33022 AC400000 20420004 1447FFF9 8C430000
/ssp_tb/single_cycle_processor_i/datapath_i/DataMem/cs	1	
/ssp_tb/single_cycle_processor_i/datapath_i/DataMem/oe	1	
/ssp_tb/single_cycle_processor_i/datapath_i/DataMem/we	0	
/ssp_tb/single_cycle_processor_i/datapath_i/DataMem/addr	10000004	00000000 FFFFFFFF 00000044 10000010 10000014 FFFFFFFE 10000014
/ssp_tb/single_cycle_processor_i/datapath_i/DataMem/din	10000000	00000005 00000001 00000005 00000009 10000010 10000028 00000005
/ssp_tb/single_cycle_processor_i/datapath_i/DataMem/dout	00000009	XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX
/ssp_tb/single_cycle_processor_i/datapath_i/Registers/reg_out	{XXXXXXXX} {XXXXX...	{XXXXXXXX} {XXXXX...} {XXXXXXXX} {XXXXXXXX} {XXXXXXXX} {XXXXXXXX} {XXXXXXXX} {XXXXXXXX}
(31)	XXXXXXXX	XXXXXXXX
(30)	XXXXXXXX	XXXXXXXX
(29)	XXXXXXXX	XXXXXXXX
(28)	XXXXXXXX	XXXXXXXX
(27)	XXXXXXXX	XXXXXXXX
(26)	XXXXXXXX	XXXXXXXX
(25)	XXXXXXXX	XXXXXXXX
(24)	XXXXXXXX	XXXXXXXX
(23)	XXXXXXXX	XXXXXXXX
(22)	XXXXXXXX	XXXXXXXX
(21)	XXXXXXXX	XXXXXXXX
(20)	XXXXXXXX	XXXXXXXX
(19)	XXXXXXXX	XXXXXXXX
(18)	XXXXXXXX	XXXXXXXX
(17)	XXXXXXXX	XXXXXXXX
(16)	XXXXXXXX	XXXXXXXX
(15)	XXXXXXXX	XXXXXXXX
(14)	XXXXXXXX	XXXXXXXX
(13)	XXXXXXXX	XXXXXXXX
(12)	XXXXXXXX	XXXXXXXX
(11)	XXXXXXXX	XXXXXXXX
(10)	XXXXXXXX	XXXXXXXX
(9)	XXXXXXXX	XXXXXXXX
(8)	XXXXXXXX	XXXXXXXX
(7)	10000028	10000028
(6)	0000005A	00000049
(5)	00000001	00000001
(4)	00000000	00000001 XXXXXXXX
(3)	0000000A	00000005
(2)	10000000	10000010 10000014
(1)	XXXXXXXX	XXXXXXXX
(0)	00000000	00000000
/ssp_tb/single_cycle_processor_i/datapath_i/DataMem/mem_file	bills_branch.dat	bills_branch.dat

The final few cycles:

	Msgs	
/ssp_tb/dk_tb	1	
/ssp_tb/single_cycle_processor_i/datapath_i/InstrFU/PC/dout	00100013	00100012 00100013 00100014 00100015 00100016 00100017 00100018
/ssp_tb/single_cycle_processor_i/datapath_i/InstructionMem/dout	1447FFF9	20420004 1447FFF9 ACE60000 20081000 00084400 8D090028 XXXXXXXX
/ssp_tb/single_cycle_processor_i/datapath_i/DataMem/cs	1	
/ssp_tb/single_cycle_processor_i/datapath_i/DataMem/oe	1	
/ssp_tb/single_cycle_processor_i/datapath_i/DataMem/we	0	
/ssp_tb/single_cycle_processor_i/datapath_i/DataMem/addr	FFFFFFF0	10000028 00000000 10000028 00001000 10000000 10000028 XXXXXXXX
/ssp_tb/single_cycle_processor_i/datapath_i/DataMem/din	10000028	10000024 10000028 00000038 XXXXXXXX 00001000 XXXXXXXX
/ssp_tb/single_cycle_processor_i/datapath_i/DataMem/dout	XXXXXXXX	XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX
/ssp_tb/single_cycle_processor_i/datapath_i/Registers/reg_out	{XXXXXXXX} {XXXXX...	{XXXXXXXX} {XXXXX...} {XXXXXXXX} {XXXXXXXX} {XXXXXXXX} {XXXXXXXX} {XXXXXXXX} {XXXXXXXX}
(31)	XXXXXXXX	XXXXXXXX
(30)	XXXXXXXX	XXXXXXXX
(29)	XXXXXXXX	XXXXXXXX
(28)	XXXXXXXX	XXXXXXXX
(27)	XXXXXXXX	XXXXXXXX
(26)	XXXXXXXX	XXXXXXXX
(25)	XXXXXXXX	XXXXXXXX
(24)	XXXXXXXX	XXXXXXXX
(23)	XXXXXXXX	XXXXXXXX
(22)	XXXXXXXX	XXXXXXXX
(21)	XXXXXXXX	XXXXXXXX
(20)	XXXXXXXX	XXXXXXXX
(19)	XXXXXXXX	XXXXXXXX
(18)	XXXXXXXX	XXXXXXXX
(17)	XXXXXXXX	XXXXXXXX
(16)	XXXXXXXX	XXXXXXXX
(15)	XXXXXXXX	XXXXXXXX
(14)	XXXXXXXX	XXXXXXXX
(13)	XXXXXXXX	XXXXXXXX
(12)	XXXXXXXX	XXXXXXXX
(11)	XXXXXXXX	XXXXXXXX
(10)	XXXXXXXX	XXXXXXXX
(9)	XXXXXXXX	XXXXXXXX XXXXXXXX 00001000 10000000 00000038
(8)	XXXXXXXX	XXXXXXXX
(7)	10000028	10000028
(6)	0000003E	00000038
(5)	00000001	00000001
(4)	00000000	00000000
(3)	00000006	00000005
(2)	10000018	10000024 10000028
(1)	XXXXXXXX	XXXXXXXX
(0)	00000000	00000000
/ssp_tb/single_cycle_processor_i/datapath_i/DataMem/mem_file	bills_branch.dat	bills_branch.dat



The red box above contains memory location 10000028. 00000038 checks out!

## Control Signal Tables

### Main Control

Name\Type	R-Type	LW	SW	BEQ	BNE	BGTZ	ADDI
RegDst	1	0	X	X	X	X	0
RegWr	1	1	0	0	0	0	1
Branch	0	0	0	1	1	1	0
ExtOp	X	1	1	X	X	X	1
ALUSrc	0	1	1	0	0	0	1
MemWr	0	0	1	0	0	0	0
MemtoReg	0	1	X	X	X	X	0
BrSel(1)	X	X	X	0	0	1	X
BrSel(0)	X	X	X	0	1	X	X

### ALU Control

#### R-type Instructions:

Function Code	ALUop
100000	0000
100001	0000
100010	0001
100011	0001
101010	0101
101011	0110
100100	0010
100101	0100
000000	1000

#### Non-R-Type Instructions

OP Code	ALUop
100011	0000
101011	0000
000100	0001
000101	0001
000111	0001
001000	0000