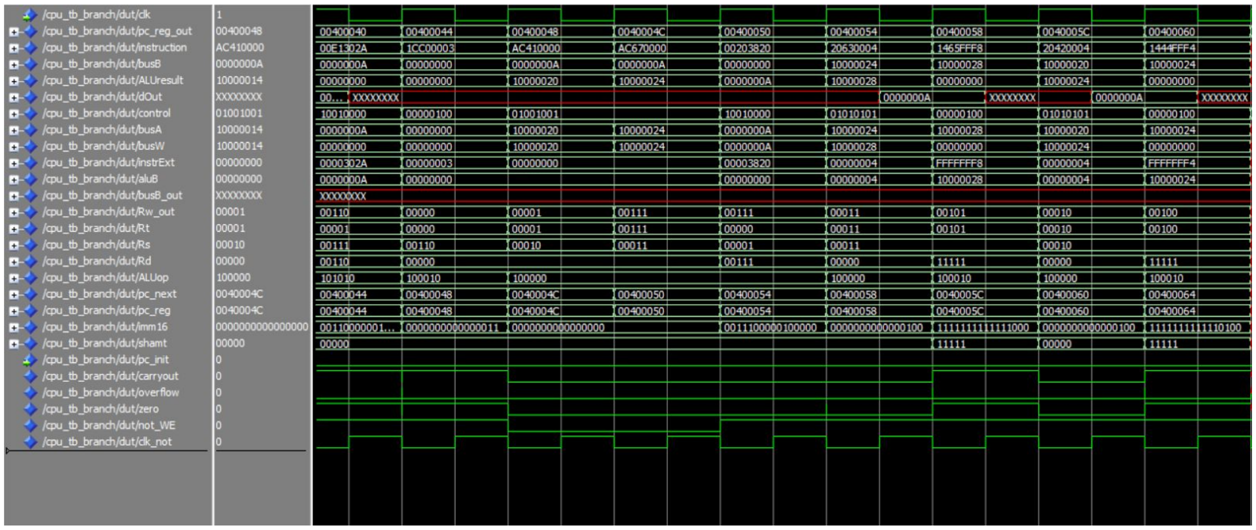
Alex Gangwish and Jason Glass
EECS 361 Group Project #1
11/21/2016

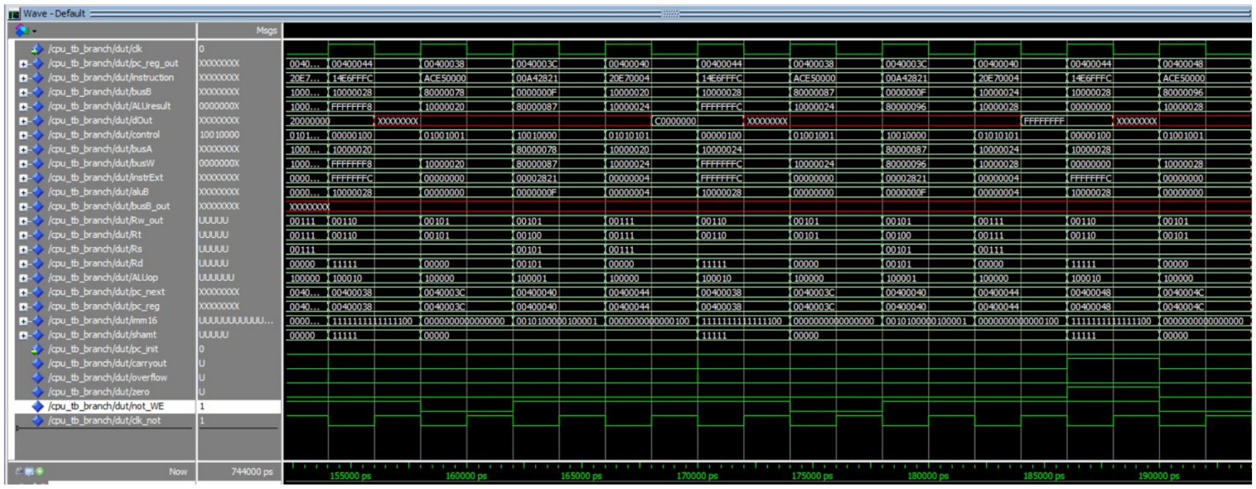
Single Cycle ALU

Sort_corrected



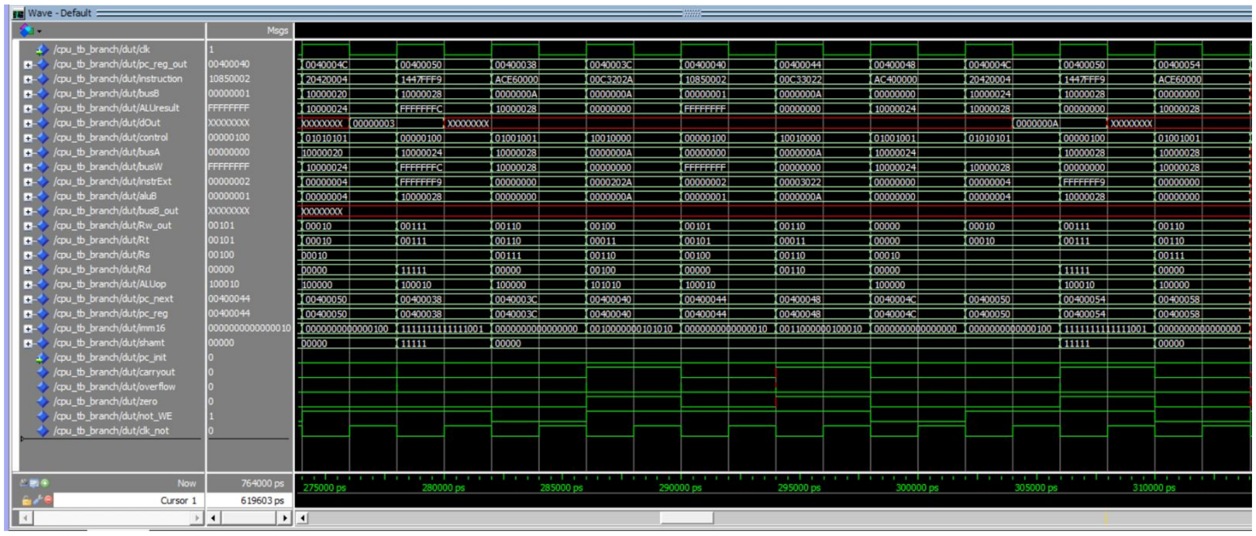
```
/cpu_tb_branch/dut/clk            1
/cpu_tb_branch/dut/pc_reg_out     00400048
/cpu_tb_branch/dut/instruction    AC410000
/cpu_tb_branch/dut/busB           0000000A
/cpu_tb_branch/dut/ALUresult      10000014
/cpu_tb_branch/dut/dOut           XXXXXXXX
/cpu_tb_branch/dut/control        01001001
/cpu_tb_branch/dut/busA           10000014
/cpu_tb_branch/dut/busW           10000014
/cpu_tb_branch/dut/instrExt       00000000
/cpu_tb_branch/dut/aluB           00000000
/cpu_tb_branch/dut/busB_out       XXXXXXXX
/cpu_tb_branch/dut/Rw_out         00001
/cpu_tb_branch/dut/Rt             00001
/cpu_tb_branch/dut/Rs             00010
/cpu_tb_branch/dut/Rd             00000
/cpu_tb_branch/dut/ALUop          100000
/cpu_tb_branch/dut/pc_next        0040004C
/cpu_tb_branch/dut/pc_reg         0040004C
/cpu_tb_branch/dut/imm16          0000000000000000
/cpu_tb_branch/dut/shamt          00000
/cpu_tb_branch/dut/pc_init        0
/cpu_tb_branch/dut/carryout       0
/cpu_tb_branch/dut/overflow       0
/cpu_tb_branch/dut/zero           0
/cpu_tb_branch/dut/not_WE         0
/cpu_tb_branch/dut/clk_not        0
```

Unsigned_sum



```
Wave - Default
                                  Msgs
/cpu_tb_branch/dut/clk            0
/cpu_tb_branch/dut/pc_reg_out     XXXXXXXX
/cpu_tb_branch/dut/instruction    XXXXXXXX
/cpu_tb_branch/dut/busB           XXXXXXXX
/cpu_tb_branch/dut/ALUresult      0000000X
/cpu_tb_branch/dut/dOut           XXXXXXXX
/cpu_tb_branch/dut/control        10010000
/cpu_tb_branch/dut/busA           XXXXXXXX
/cpu_tb_branch/dut/busW           0000000X
/cpu_tb_branch/dut/instrExt       XXXXXXXX
/cpu_tb_branch/dut/aluB           XXXXXXXX
/cpu_tb_branch/dut/busB_out       XXXXXXXX
/cpu_tb_branch/dut/Rw_out         UUUUU
/cpu_tb_branch/dut/Rt             UUUUU
/cpu_tb_branch/dut/Rs             UUUUU
/cpu_tb_branch/dut/Rd             UUUUU
/cpu_tb_branch/dut/ALUop          UUUUUU
/cpu_tb_branch/dut/pc_next        XXXXXXXX
/cpu_tb_branch/dut/pc_reg         XXXXXXXX
/cpu_tb_branch/dut/imm16          UUUUUUUUUUU...
/cpu_tb_branch/dut/shamt          UUUUU
/cpu_tb_branch/dut/pc_init        0
/cpu_tb_branch/dut/carryout       U
/cpu_tb_branch/dut/overflow       U
/cpu_tb_branch/dut/zero           U
/cpu_tb_branch/dut/not_WE         1
/cpu_tb_branch/dut/clk_not        1
Now                               744000 ps
155000 ps   160000 ps   165000 ps   170000 ps   175000 ps   180000 ps   185000 ps   190000 ps
```

Bills_branch


```
Wave - Default
                                       Msgs
/cpu_tb_branch/dut/clk                 1
/cpu_tb_branch/dut/pc_reg_out          00400040
/cpu_tb_branch/dut/instruction         10850002
/cpu_tb_branch/dut/busB                00000001
/cpu_tb_branch/dut/ALUresult           FFFFFFFF
/cpu_tb_branch/dut/dOut                XXXXXXXX
/cpu_tb_branch/dut/control             00000100
/cpu_tb_branch/dut/busA                00000000
/cpu_tb_branch/dut/busW                FFFFFFFF
/cpu_tb_branch/dut/instrExt            00000002
/cpu_tb_branch/dut/aluB                00000001
/cpu_tb_branch/dut/busB_out            XXXXXXXX
/cpu_tb_branch/dut/Rw_out              00101
/cpu_tb_branch/dut/Rt                  00101
/cpu_tb_branch/dut/Rs                  00100
/cpu_tb_branch/dut/Rd                  00000
/cpu_tb_branch/dut/ALUop               100010
/cpu_tb_branch/dut/pc_next             00400044
/cpu_tb_branch/dut/pc_reg              00400044
/cpu_tb_branch/dut/imm16               0000000000000010
/cpu_tb_branch/dut/shamt               00000
/cpu_tb_branch/dut/pc_init             0
/cpu_tb_branch/dut/carryout            0
/cpu_tb_branch/dut/overflow            0
/cpu_tb_branch/dut/zero                0
/cpu_tb_branch/dut/not_WE              1
/cpu_tb_branch/dut/clk_not             0
                                Now    764000 ps
                           Cursor 1    619603 ps
```

Summary of Control Signals

	opcode hex	function	binary opcode	function binary	RegDst	ALUSrc	MemtoReg	RegWrite	MemWrite	Branch	Jump	ExtOp	ALUop <5:0>
add	0	20	000000	100000	1	0	0	1	0	0	0	x	100000
addu	0	21	000000	100001	1	0	0	1	0	0	0	x	100001
sub	0	22	000000	100010	1	0	0	1	0	0	0	x	100010
subu	0	23	000000	100011	1	0	0	1	0	0	0	x	100011
and	0	24	000000	100100	1	0	0	1	0	0	0	x	100100
or	0	25	000000	100101	1	0	0	1	0	0	0	x	100101
slt	0	2a	000000	101010	1	0	0	1	0	0	0	x	101010
sltu	0	2b	000000	101011	1	0	0	1	0	0	0	x	101011
sll	0	0	000000	000000	1	0	0	1	0	0	0	x	000000
lw	23		100011		0	1	1	1	0	0	0	1	100000
sw	2b		101011		x	1	x	0	1	0	0	1	100000
beq	4		000100		x	0	x	0	0	1	0	x	100010
bne	5		000101		x	0	x	0	0	1	0	x	100010
bgtz	7		000111		x	0	x	0	0	1	0	x	100010
addi	8		001000		0	1	0	1	0	1	0	1	100000

Our Control Unit is built to first check if the Opcode is “000000” and thus an R-type instruction is being executed. In this case the control unit simply passes the function code out as the ALUOp as well.

If an R-type instruction has not been loaded then the control must figure out if the instruction is either a Branch, Lw, Sw or Addi. A 2-bit selection signal consisting of {opcode(3), opcode(5)} is sent into a 4-way mux. If Opcode(3) and Opcode(5) are low then the instruction is a branch, if opcode(5) is high and Opcode(3) is low then it’s a load if both high then a store. If Opcode(5) is low and opcode(3) is high it is addi. The selection signal then gets the hardwired control signal for each corresponding instruction from an 8-bit mux. The selection signal is used in the same manner to get the ALUop as well.

Design Challenges and Solutions

The following are the main issues we faced while building our processor, as well as the steps we took to resolve these issues.

DMEM Clock Signal
Perhaps the most frustrating challenge we faced had to do with our data memory unit--more specifically, upon assembling our datapath we began experiencing inconsistencies with our ability to store and read from data memory. After troubleshooting the issue, we determined that the problem was our CPU was attempting to get values from the memory unit before the address had stabilized, due to all clocked elements in our CPU being tied to the rising edge.

To solve this issue, we simply inverted the clock signal tied to data memory. This effectively caused our memory to read/write according to the falling edge of the clock, delaying its use to a half cycle later than when the registers would be utilized. We had also discussed using a second clock for the data memory that runs twice as fast as the overall CPU clock, causing data memory to be accessed twice per cycle and achieving a similar effect to our implementation. However, we decided to use our method in order to simplify clocking.

Control Design
One large problem was figuring out how to design the control unit and have it be as efficient as possible. While it was easy enough to figure out to check if the Opcode was Zero using a nor-gate , it was more difficult to create non r-type instruction control. Our solution was to find the significant bits that differed from lw, sw, bne, beq, bgtz and addi.

It was seen that only bits 5 and 3 of opcode were necessary to decode the instruction. As summarized in the control table these bits were then used to get an output from a mux which has hardwired control signals for each instruction.

Next Address Logic Issues
We had difficulties with designing the next address logic given that the NAL unit had to be able to figure out whether to branch for 3 different kinds of branches, beq, bne and bgtz. The first difficulty was discovering how to use the zero and carryout signals from the ALU to figure out the branch-true condition for each branch. For Beq zero and carryout are ‘1’, for bne zero is ‘0’ and carryout is ‘0’, and for bgtz zero is ‘0’ and carryout is ‘1’; Thus like in our control unit a 2-bit selection signal was made from [zero, carryout] to select the proper pc from a mux, although first it was necessary to determine what branch was given.

It was necessary to create an xor-and component that checked the given instruction versus the known instructions of beq, bne and bgtz to determine which branch type was given for this execution of NAL. For example BEQ is “100” so Xor that with “011” then bitwise AND and you will have ‘1’, it would be ‘0’ for any other opcode. This signal was generated for beq, bne and bgtz. Given the combination of the two layers of selection logic it was possible to create a mux-tree that outputted the proper pc_four or pc_branch given the instruction and zero/carryout signals from the ALU.