Processor

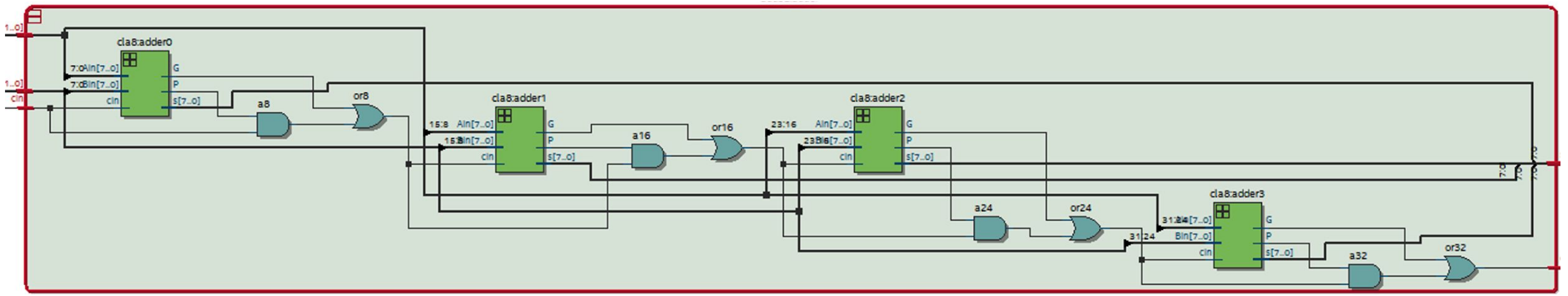
Clayton Shafer

Objective

- Purpose - Create a pipelined MIPS(ish) processor to be used for the final project
    - The processor uses the ISA provided in the project checkpoint document
- Design considerations:
    - Correctness
    - Speed (both clock speed and cycles per instruction)

ALU

- Addition: 2 level hierarchical carry-lookahead adder
- Subtraction: uses same adder, invert subtracted term, carry in a one
- And/or: each bit passed into a 2 input logic gate
- Shift: Uses a 32 bit barrel shifter, made up of smaller shifters



Register File

- Basic register made up of 32 D flip flops
- Register file made up of 32 registers
- Two read ports, read register chosen with a 5-32 decoder and a series of tri state buffers (due to delay associated with 32-1 mux)
- One write port
- Clocked on negative edge, to allow time for values read from registers to reach the pipeline registers

Pipeline

- Pipeline is made up of 5 stages:
  - Fetch - Instruction fetched from instruction memory
  - Decode - Register values read from register file
  - Execute - ALU operations
  - Memory - Data saved and loaded from memory
  - Writeback - Data saved to registers
- Full bypassing implemented to resolve data hazards
- Special purpose register placed in between each of the pipeline stages, in order to save the values after each step of the instruction is carried out

Math Instructions, Memory Instructions

- Math instructions mostly implemented using the ALU
  - Mult/Div implemented separately
- Values read during decode, sent to ALU, then saved during writeback
- Immediate type instructions implemented by using a mux to input the immediate to the ALU
- Instruction memory implemented using a Quartus syncram ROM component
- Data memory implemented using a Quartus syncram RAM component
- Output from ALU sent to address input for sw/lw, second register value sent to data input for sw

Branch, Jump Instructions

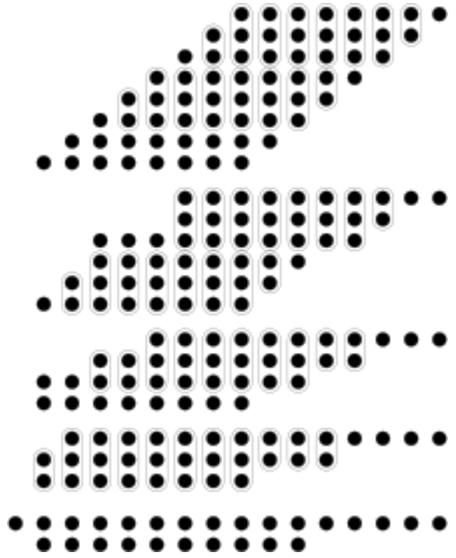
- Due to time constraints, naive approach was taken to implement jump and branch instructions
- Branch control bits are found in the execute stage, as part of the ALU
- Jump involves a one cycle penalty, as the jump is resolved in the decode stage
- Branch involves a two cycle penalty, as the branch is resolved in the execute stage

Exceptions

- Exception control signals originate from the ALU and the mult/div units
- Value to be written to register 30 was found using a 8:3 encoder, and then muxed into the writeback data port
- Branch exception is implemented in the same way as the other branch instructions
- Setx is resolved at the writeback stage by setting the data port equal to the extended target value

Multiplier

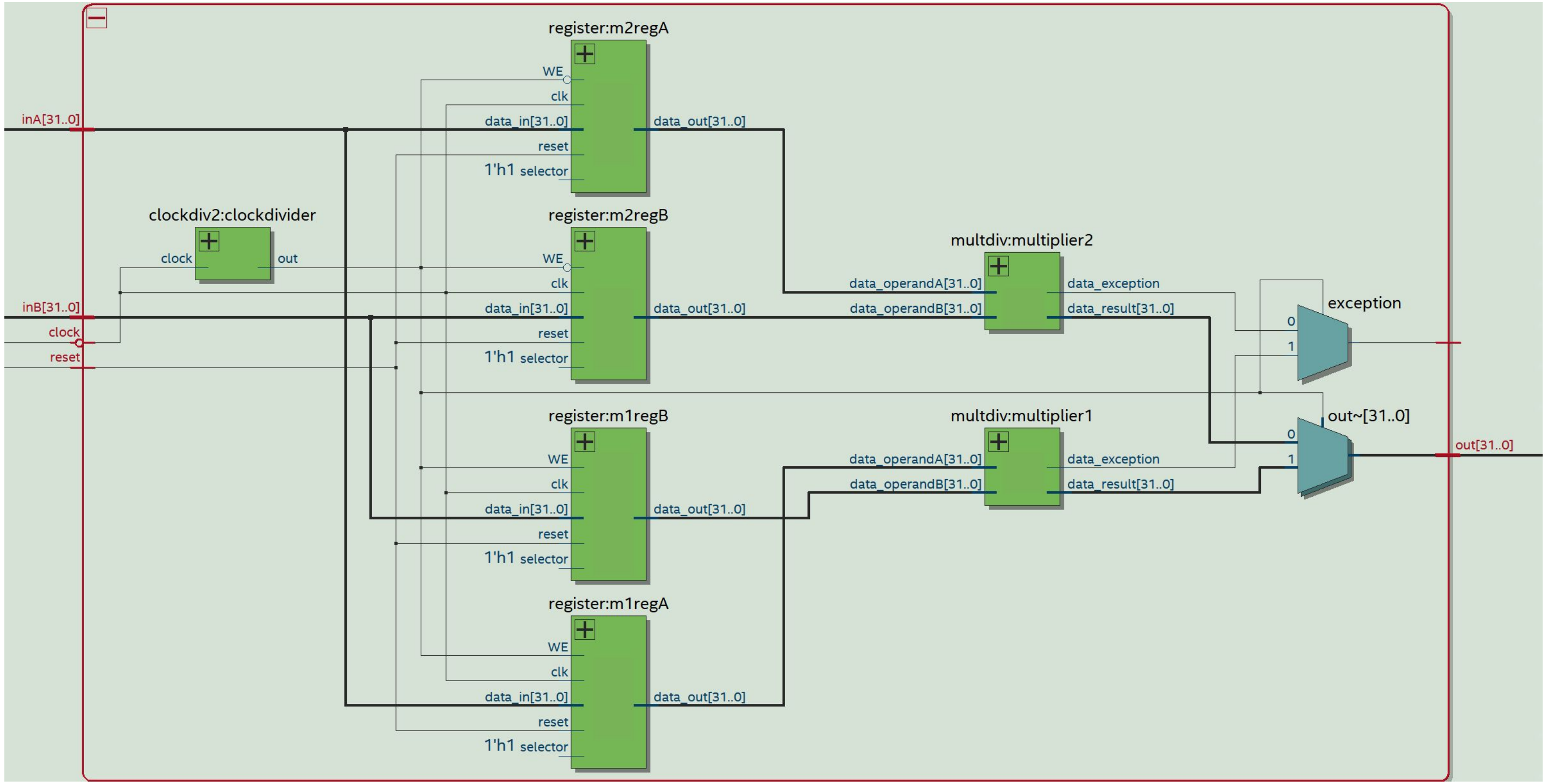
- Started by creating an 8-bit wallace tree multiplier
  - Generate partial product matrix
  - Reduce partial product using full adders and half adders
  - Add final terms using carry-lookahead adder
- Four 8 bit multipliers used to make a 16 bit multiplier, then four 16 bit multipliers used to make a 32 bit multiplier



Multiplier - Clocking

- Experimentally determined that the 32 bit multiplier takes just under 2 cycles of a 50MHz clock
- Integrated the multiplier into the processor using a modified pipeline architecture
  - Used a clock divider to alternate between two separate multiplier units
  - Multiplier result fed directly into the writeback stage - ignores memory stage
- Because instructions keep moving through the processor while multiply is in memory stage, operands must be saved in registers at the top of the multiplier pipeline

Multiplier architecture



Divider

- Created a 32 bit array divider
  - 32 rows, each consisting of a 32 bit hierarchical carry-lookahead adder
  - Nonrestoring division, meaning that each row either adds or subtracts a shifted divisor from the dividend
- Significant delay associated with the divider as signal must carry both vertically and horizontally
  - However, still comes in at 24 cycles per divide - Faster than the sequential design described in class
- Clocked using a series of D-flip flops to provide adequate delay

Pipeline Control

- Hazards often cause changes to the flow through the pipeline
- lw to ALU input - insert one nop in decode stage
- Read value after multiplication - insert one nop in decode stage
- Branch - Insert two nops to flush bad instructions
- Jump - insert one nop to flush bad instruction
- Division - two division instructions cannot happen in a row due to the nature of the division control signals. Nops are inserted between successive divides
- Division causes full stop of the pipeline until the computation is completed

Errors

- System was debugged using waveforms in the early stages, and testbenches in the later stages
- Biggest problem that is still unresolved is general poor performance on the timing simulations
    - Problem arose when implementing the exception logic, which was the last thing I did. I believe the logic that chooses which register and what value to write back is taking too long, which leads to incorrect values sometimes being saved to registers. I tried a variety of different configurations to try to fix the issue, but none were successful