* *PC Unit*:

**Its Function**: To store memory location value

**Pin/Pout**: 32 in , 32 out

**Implementation**: Just one memory cell inside the PC that will change based off of

Internal logic.

**Testing**: Feed in a 32 bit value and see what the program counter’s output pins sho

**Hours:** 2 hours hopefully, plus testing

* *ALU Unit*:

**Its Function**: Based off of instruction set, it performs a certain operation

**Pin/Pout**: 32 in , 32 in , 64 out?

**Implementation**: Have bunch of function blocks (i.e. adder, subtracter, divider, multiplier,…) attached and multiplexed to the control signal

**Testing**: Testing the individual pieces by reading the output pins, then integration testing ith the MUX.

**Hours:** Easily 20.

* *Adder*

**Its Function**: Using the XOR to implement a 64 bit adder. Maybe additional input for 2’s complement to subtract. Maybe an output for carry flag to go off globally. Another input for the carry in

**Pin/Pout**: 32 in , 32 in , 1 pin for 2’s complement, 1 pin for carry in, 1 pin for carry out (I might be complicating things a bit. Please let me know).

**Implementation**: Using the architecture described already. Have 2 32 bit blocks. Encapsulate them with the carry in/out functionality as well as subtraction functionality

**Testing**: Simple tests to pass in to see if adder/subtracter is in fact working

**Hours:** testbench 1-2 hours, adder/subtractor – 4-5 hours

* *Subtractor – see adder*
* Divider

**Its Function**: Divides 2 32 bit numbers. We could go into fractions, but no.

**Pin/Pout**: 32 in, 32 in (whole numbers only), 32 out (most significant 32 bits are already zero -> because easy division testing)

**Implementation**: Need to look up implementation.

**Testing**: Simple tests to pass in to see if output pins coming out are correct

**Hours:** testbench 2 hours (or maybe I can copy and paste tester code from another testbench written for other stuff) , divider – 5 hours, research implementation – 1hr

* Multiplier –

**Its Function**: Multiply 2 32 bit numbers. Output is the result.

**Pin/Pout**: 32 in, 32 in (whole numbers only), 32 out (most significant 32 bits are already zero -> because easy multiplication testing/design – is that okay?)

**Implementation**: Have the implementation with shifting and addition. Do we need to implement a shifter?

**Testing**: Try to copy test bench of other units

**Hours:** testbench 2 hours (again, try to copy code) , multiplier – 4 hours

* Others – Should be relatively simple (but I could seriously be wrong here) – 0-8 hours

**Testing**: Try to copy test bench of other units

* The 3 Memory Modules:
  + Instruction Memory:

**Its Function**: To simply store instructions and possibly give back the address of an instruction?

**Pin/Pout**: Read Address = 10 bits for 2^10 instructions. Instruction out = 32 bits because Opcodes are 32 bits. Do we need a way to return the 10 bit address for a given instruction (jump statement/branching statements might require this) as output. Maybe this happens at compile time and the optimizer basically puts the address of the branch into the Instruction memory.

**Implementation**: - We should talk! But it should be multiplexing of the read address. How do we load up the instructions initially?

**Testing**: Same way as testing data memory but also, we might want to test the other things that I am considering.

**Hours:** testbench 3 hours, data memory– 4-5 hours

* + General Purpose Registers:

**Its Function**: For storage of temporary data or quick access data

**Pin/Pout**: Read Reg1 = 5 bits for 2^5 or 32 addresses, Read Reg2 = 5 bits, Write Reg = 1 bit, Write Data = 32 bits, Read Data 2 = 32 bits, Read data 1 = 32 bits

**Implementation**:

**Testing**: Same way as testing data memory

**Hours:** testbench 1 hours, data memory– 2 hours

* + Data Memory: SRAM

**Its Function**: This is effectively the SRAM, the thing we use when we are doing the .data directive or .space directive

**Pin/Pout**: Address – 10 bits for , Write Data -32 bits, Read data – 32 bits

**Implementation**: The address is used to multiplex. I might need to contact someone I know who knows more about this. 2^10 or 1024 by 32 bits for about 32000 or 4000 byte data.

**Testing**: Writing into the memory, and then next cycle, reading into the memory

**Hours:** testbench 2 hours, data memory– 4 hours

* Add : can we just use a simpler form of the adder used in the ALU
* Control:
  + This is the sequential logic, and I will use a Finite State Machine
  + Need to do more research !
  + I have to deal with R-Type, J-Type, and I-Type Instructions -=> How would I implement I-Type instructions
* Other Units:
  + Mux 0 to 1 : 3 hrs – testing and implementation – what should this do?
  + Sign Extend : 32 in, 64 out – sign extend the 32 by taking 2’s compliment and then if MSB is 1, then
  + Shift left 2: 3 hrs – testing and function