The following is the rationale behind our choices for the processor:

October 6, 2015

1. 16 registers were selected because we do not wish to deal with a constrained amount of registers and we do not wish to be extremely cautious when using our registers. Also it reduces to 4 bits for addressing therefore only utilizing a fourth of the instruction per register. (Also, hex code is OP)
2. 5 unaddressable registers: PC, SP, ~~RA~~ (10/12), INT(interrupt) , and CV(compare value). these registers should not be touched except for the instructions explicitly used for this.
3. 16 instructions were selected to have an optimal amount of instructions while still only using a fourth of the instruction space. 8 instructions would be much too little while 32 was way too many types of instructions. (Also, hex code is OP)
4. Two different instruction types were selected for simplicity. All instructions use the R-type instruction except when we need to load an immediate value. By maximizing the use of R-type, we have a nice standard to implement in hardware.
5. We have decided to make a M.U.S.H. architecture. Our architecture utilizes aspects of Load-Store and stack architecture. This allows us to use registers for speed, but also store information easily on the stack.

October 9, 2015

6. Jim decided to start all of the instructions at the address 0xF000 so we can imply an F for any instruction address
7. Jim decided to put globals from 0x0000 to 0x0FFF so that whenever an address in memory would need to be fetched, that address could be expressed in 12 bits with implied leading zeros. This allows us to fetch without shifting and oring the last 4 bits of the address

October 12, 2015

8. We decided to sacrifice temporary at index 13 so that we could index $ra. When asked if he would be diggy down with this, Brian replied, “I would be diggy down”. This will save a couple instructions.
9. Because the change, jal and ret are both unneeded instructions. They will now be depreciated until possible replacement in the future (maybe replace with a load upper immediate, NSA WARNING MODE, or a find GCD)
10. We made an RTL that was based upon the multicycle RTL given out in class; however, it goes more into detail for specific instructions and shows how we will construct the datapath.
11. Due to the changes above, the Euclids algorithm is changed, too.

October 13,2015

1. Nah, we need jal. There’s no way to access PC otherwise, and we need to get the value of PC into RA for calls.

October 14,2015

1. Changed jump instruction. Instead of getting the value from RD we now get it from RS, because then we don't need to add the functionality of getting a third value from the register file.
2. Group met at the classroom during class hour to work on Milestone 3. Started with DataPath.
3. We decided we're not pipelining it.
4. We changed Store Word to store the memory from RT to RS. Same thinking process of the jump instruction. Also we use the same wiring from Load Word.

October 20,2015

1. Continued working on the DataPath, on push and pop instructions
2. Decided that there should be wires out of both ALUOut and SP to wd because of the ordering of "Access Memory / Update SP" for push and pop. Other option would be to have only ALUOut, and in that case when we're pushing a word into the the stack ALUOut first needs to be SP+0 and then SP-2. We didn' t want to have to use the ALUOut 2 times so we went with the first option. Although now it looks single-cyclish. Should ask Sid. Will ask Sid.
3. Got rid of ALUOut = Reg[IR[3-0]] on the second step of the rtl. We couldn't figure out why it was there in the first place
4. Since everytime PC gets updated, the first byte is 0xF, the PC register will only update the 12 least significant bits, and 15-12 will always stay high. That way we don't need an extra component to overwrite these bits.
5. Updated RTL and instructions descriptions, such as sub (didn't include CMP mechanic).
6. We look through each component individually and figure out what are input and output. Then design how to test it.
7. We uses $SP in our architecture and it becomes an register in the datapath. It is a separate thing which is different than MIPS.

October 23,2015

1. After discussing it with Sid, decided that SP has a separate Adder (like PC). Updated the DataPath, Control System and RTL to match the changes. Need to update the Diagram that includes the DataPath and the Control System to include SP Source and PushPop
2. Brian made a majority of the hardware components in Xilinx utilizing Verilog, he has yet to implement the tests to verify the hardware works….. but progress is being made.
3. Tests are made.

October 25,2015

1. added a mux to select -2 and 2 to the stack pointer
2. The group met and created control system diagram (later to be coded in verilog by Brian).

October 25,2015

1. made a truth table for the state machine
2. updated list of control flags to match datapath and control design
3. decided that combinational logic on branch will be part of datapath-- not control system
4. changed lw and sw in RTL

5. Changed implementation of the stack pointer this can be seen in new datapath diagram\
6. Jim still needs to use mad mspaint skillz to hack the NSA (finish datapath image)
7. Changed the FSM diagram. Don't need CMP to go into the Control System, it can just go to a circuit which outputs to the "enable" of the PC Register. For that to happen we needed to add another control flag (branch). So the logic for the enable input of the register is EN=PCWrite + (CMPisZero.branch) //CMPisZero is a NOR of all CMP bits.

October 26,2015

1. Yay! jim used his mad 1337 mspaint h4x to put control signals back on that datapath
2. No combinational units means no truth table.#Rekt
3. We decided reset state is "0000"

October 30,2015

1. Most tests are done and most components are fixed
2. Register file and write decode is unfinished because Songyu do not fully understand what they are for. So someone else will fix them, I hope.
3. When Songyu tried to commit codes to svn, it said "no space left". So Songyu guesses we need to recreate the whole project by following lab 7 instruction. It's lots of work!

November 2,2015

1. OH GAWD WE ALMOST HAD AN SVN TRAGEDY
2. *shower thought* why did they not teach us how to version control formally
3. Why not use GIT?"!?!?!?!?!?!?
4. made some sweet plans for testing
5. Scratch #1, Shit went crazy. ALU (praise be unto him) is angry with us. Our sinful ways have angered the ALU. Brian, was an asshat and never committed so today we tried making stuff work. He cried a single manly tear for causing his teammates troubles and pain. It should be noted that the tear he cried was bench pressing 250 while it slid down his cheek.
6. Jim will purge the sodom below that is the svn and e-mail the solutions
7. Brian still doesn't update the journal as much as he should.
8. Brian is finishing off the components tests and will integration testing soon.
9. sodemn has been purged

November 3,2015

1. We discovered that the memory in xilinx addresses memory as 16-bit blocks. So we changed our datapath to add/sub "1" to PC and SP (instead of "2").
2. We cannot test Stack in integration test plan because it requires the whole data path to test . So we combine it to the datapath test.
3. All the example test are removed because they are in the Xilinx Project now. Please check them in that folder.
4. No RTL,control unit design changed.
5. datapath is changed slightly