Steven Hall
6.111 Final Project
November 4, 2005

Proposal

Considering the big picture view of this project, the whole system being designed will function as a general-purpose computer, complete with a keyboard for input, a monitor for output, and a general-purpose 64-bit CPU to handle the processing in between. Additionally, it will have one or more programs assembled for it and ready to run, in order to demonstrate its capabilities. This project was inspired by the infamous Beta processor, but it will have several key differences, each of which presents a significant departure from the Beta architecture towards a unique, and perhaps superior, architecture. Of these differences, a different one shall be mentioned in each of the seven module descriptions.

The microsequencer module has a task similar to its counterpart in other processors, which is to keep track of which instruction should be executed in the next cycle. As such, it's basically a fancy counter, with the ability to take on special values when certain interrupts are signaled (divide by zero, integer overflow, stack overflow, etc.), assuming, of course, that those interrupts are not masked out by the processor's mode mask. The testing for this module will include a few interrupt tests and basic counting, since that is really the limit of this module's ability.

The memory "module" would not normally be considered a normal module (as opposed to simply a memory device), but this memory serves double duty, since the CPU's I/O capabilities are provided by memory-mapped I/O. In this case, the only real interaction between the CPU and the peripherals will occur through normal reads and writes to the locations of memory which happen to contain the video buffer, character buffer, etc., while the ps2 and xvga modules use those buffers. The test for the memory module will consist of a number of writes and reads, as well as I/O tests (also through writes and reads) with the memory mapping system.

As its name suggests, the control logic module is responsible for parsing the machine instructions and sending the appropriate control signals to the remainder of the CPU's modules in order to perform the desired task. A significant increase in complexity from its Beta counterpart, this module must also coordinate two distinct data paths, depending on whether the operands are words (64 bits) or bytes, both of which have a complete set of ALU operations for their types. The testing for this module will see a vast number of instructions sent to the module, with checks on the various control signals in response.

The next two modules are another standard in CPU architecture, the register files, although these modules also have some differences compared to most register files. In addition to the fact that there are separate sets for word and byte registers, each individual register also has an additional "signedness" bit to internally keep track of which numbers

are signed and unsigned. The effect of this should be dramatic, as the user won't have to remember signedness, eliminating a non-trivial source of error, and some operations are made possible simply because of this change (i.e. heterosigned compare). The testing here will be similar to the memory test (since this is just another form of memory), except that no I/O checks will be necessary.

Another pair of modules seen in almost every processor, the arithmetic and logic units (ALU), also has a new twist (from computer science and type theory) which should affect the cleanliness and speed of programs on this machine. These ALUs perform all the basic integer math, Boolean operations, shifts/rotations, and comparisons, although the result of these comparisons is neither a word nor a byte, but in fact one bit, representing either true or false (which makes sense, since the result of a comparison should be of Boolean type). This change should decrease register waste (due to storing just one bit of information in a whole word) while increasing type correctness (since math operations cannot be carried out with Boolean operands). The ALU testing will be fairly straightforward: supplying a variety of operations and operands while checking that the correct value (and type) is produced.

The first of two special types of modules in the processor, the stack, is represented with a word-width stack and a very unique bit-width Boolean stack. First, on the topic of the regular stack: it is a simple hardware implementation of the stack data structure designed with the goal of abstracting stack procedures away from the common memory pool (a place where they never should have been in the first place). Next, the novel Boolean stack has the job of filling in for a bit-width register file while enforcing type correctness, as the destination of all comparison results (and therefore all Boolean values in the system), the source of all values for true/false branches, and the site of all bit-width Boolean operations (in postfix notation, of course). The stack testing may be a bit tricky, namely in guaranteeing that the stack discipline is constantly maintained, so there will be a great stress test on the timing parameters for these modules.

Last (and probably the least complex), the timer module is another uncommon addition to the CPU, which serves the very simple, yet thoroughly useful, task of dividing the clock period and keeping time accurate to the millisecond in hardware (thus, without the need for system timer calls), while also permitting itself to be set (just in case). The testing for this module will be significantly easier than most of the rest of the system, since it is essentially just a counter: a few reset tests and a sanity timing check should suffice.

Finally, after all of the module-level testing draws to a (successful) close, the true test, system integration, will begin. At this point, if every module works as specified, the result should be a fully functional computer. Therefore, the testing performed at this stage will simply be to use it as intended, running programs and (ideally) writing and assembling programs on the computer itself. From this point on, all of the neat side features which were not critical for operation can be implemented.

Block Diagram