XEROX

SYSTEM DEVELOPMENT DIVISION
Communications Systems
August 13, 1977

To: Metcalfe

From: Crowther

Subject: D0 Simulator status.

The D0 simulator and its associated ddt now exist as mesa programs on <Crowther>dzeroSimulator.image.

The simulator has been tested on two sections of microcode provided by Don Charnley - the first is a routine to do unsigned multiply, and the second a fragment of the emulator. The latter was successfully run on a tiny mesa program which fitted in a single quadword. The mesa program computed a series of triangular numbers, exercising both main memory access and the hardware stack features. The D0 mechanism for accessing more than a single quadword is yet to be specified.

At the present time there is no microcode assembler for the D0. Thus the load and dump commands probably don't work (although the load will load the output of a dump), and extensive testing of the simulator is awful work. (It took hours to get Charnley's programs correctly assembled.) In these circumstances one must expect a rash of minor bugs still to be found.

The ddt is a crude version of standard PDP-10 ddt. It is meant to be self documenting, but probably succeeds only for people with some experience with other ddt's, or those who have at least read the DEC ddt manual.

The simulator tries to mimic the actual D0 design at the hardware register level, and is thus at least partially documented by the D0 manual itself. The help commands for the ddt provide further information about the simulator and its control.

The simulator and the ddt are really two quite separate entities. The only communication between the two is a pair of routines called ReadSimulator and WriteSimulator, which deal with a 32 bit address and a 32 bit contents, plus a number of signals which come up from the simulator when the user tries to do illegal or non-implemented things.

The simulator has a register for each D0 hardware register (more or less), and keeps two copies of each register, one for the start of a 70 ns simulated cycle and one for the output of that cycle. At each simulated clock tick the start registers are loaded from the output registers of the preceding cycle. Unlike the registers, there is no attempt to simulate the D0 gates. Whenever a gating function is required the simulator computes it in the most convenient fashion.

All of the simulated hardware registers plus many of the gating functions are accessible to the ddt to aid in microcode debugging and general understanding of what is going on.

The simulator does not simulate I/O, memory faults or the hardware to cope with then, and the maintenance panel or other start up hardware. It does simulate the task mechanism, although there are probably errors remaining in that section since there is nothing to test it with. It does not simulate those sections of the mesa-help hardware

which have not yet been finalized, although it does simulate some functions (nextInst and NextOp) not described in the D0 manual.

The simulator may be run in single cycle or single instruction mode, as well as the normal continuous mode. It may also be run in a non-overlapped mode, which is exactly like the regular mode except that the next instruction is not started until the current one is finished. This mode is particularly helpful for microcode debugging - otherwise it is rather confusing to know where you are when you stop.

I have done no timing measurements. We had guessed that the simulator would run 1000 times slower than real time, and nothing so far has altered that estimate. The only "measurement" that exists is that executing 1000 instructions in non-overlap mode causes a noticeable pause - perhaps a second - before control returns.

I would especially like to thank Chuck Thacker, Brian Rosen, and Don Charnley, who patiently answered a lot of dumb questions while I came to understand how a D0 works. I sincerely hope the D0 I have simulated is at least in the spirit of the one they tried to describe to me, and perhaps also true in most of the detail.

There is a great deal of detail left to describe, of the sort "operation so and so was inplemented under the assumption xxx because at the time several hardware implementations were being considered, and xxx seemed like the eventual winner." These details are important but dull, and will be described in another memo. There will inevitably be wrong choices here - a penalty of simulating a machine while the final stages of design are still going on. For example, there are 100 codes which describe the cycle/masker operation. For the hardware this amounts to patterns in a prom, and is to first order irrelevant. There will surely be second order reasons to pick a particular pattern, but that decision has yet to be made. The simulator has chosen a particular mapping because it had to have some mapping or it would not function. Some day when the correct mapping is chosen the simulator will need to catch up.