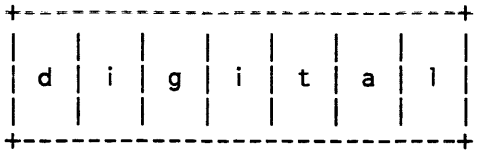


interoffice

memorandum

To: Distribution

CC: Ulf Fagerquist
Peter Hurley
Jupiter Engineering List
Jupiter Management List

Date: 20 Mar 83
From: Mike Uhler
Dept: L.S.E.G.
DTN: (8-)231-6448
Loc/Mail stop: MR01-2/E85
Net mail: UHLER at IO

Subject: Minutes of the Jupiter brainstorming meeting

Distribution:

Jim Flemming	Judy Hall	Don Hooper
John Kirchoff	Arnold Miller	Dan Murphy
John Murray	David Nixon	Pat Sullivan

1.0 Introduction

Since we are in the replanning stage for Jupiter II, it seemed appropriate to reconsider some of the design decisions that were made for Jupiter I. To do this, a meeting was held on March 3, 1983 to discuss ideas for the Jupiter II CPU.

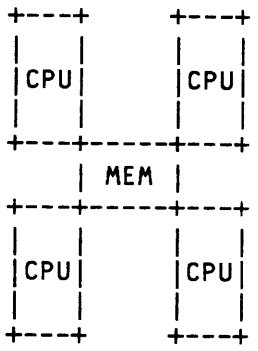
Those attending were: J. Flemming, J. Hall, D. Hooper, J. Kirchoff, A. Miller, D. Murphy, J. Murray, D. Nixon, P. Sullivan, and M. Uhler

2.0 1 Multiprocessing configurations

There was considerable discussion about the multiprocessing strategy with Jupiter. SMP configurations that exist with KL10 systems are not currently possible with Jupiter because of the tightly coupled MBOX and memory. To implement SMP, we would need to do the following:

- Design a multi-port memory.
- Add some cache consistency hardware to keep the caches on all processors in agreement.
- Design a clocking system so that all processors and the memory run synchronously.
- Design some sort of inter-processor communications capability.

Because the memory is multi-ported, it would probably have to be in a separate cabinet that could be placed between all the processor cabinets. A four processor configuration might look something like:



There were several questions raised about such a configuration. How do we connect I/O to the system? Is it done through independent ports on each CPU as we do now, or is there some sort of system interconnection that could be done?

In addition, there was concern that the current 25-bit physical memory address (32 Mwords) might not be enough to support four

processors. It seemed possible to increase the physical address to 27 bits to allow addressing of up to 128 Mwords of memory.

Such a configuration could also be used in a loosely coupled mode if the memory could be segmented into individual chunks for each processor that could not be referenced by other processors.

SMP and LCS/CFS each have their advantages and disadvantages and there was some discussion about whether we should indeed implement SMP capability in Jupiter II. The general feeling seemed to be that it would be useful if we could get memory and all processors into one cabinet, but that the usefulness decreased if that couldn't be done.

In addition, there is a possibility that the addition of SMP capability could slow down the cycle time and memory access time of the single processor configuration.

While SMP is an attractive option, TOPS-20 is committed to LCS/CFS and there seemed to be a feeling that SMP was seconday to support for LCS/CFS. In particular, there was a request for a low overhead inter-processor communications bus for the loosely coupled case. CFS could incur quite a bit of overhead going through multiple levels of protocol getting onto the CI.

3.0 External array processors

There was some discussion about the possibility of connecting an external array processor to the machine. Array processors similar to that made by Floating Point Systems have been successful on the KL10 because they could be connected directly to external memory ports. With Jupiter II, the MBOX and memory are tightly coupled and there are no external memory ports available.

The only obvious way to connect an array processor to the machine is via the IO bus to the MBOX. At a clock speed of 28ns, the maximum throughput is one word every 336 ns. It wasn't known if this was sufficient to support an array processor.

4.0 COBOL performance

There was concern about the performance of COBOL on Jupiter. It seems unlikely that we will be able to make the EXTEND instructions as they exist today run four times a KL10 without adding a large amount of special purpose hardware.

There seem to be two possibilities for improving the performance of COBOL. The first is to convert the code that now uses EXTEND to routines that use other instructions in an attempt to take advantage of the pipeline. The other possibility is to define new

instructions which don't allow the full generality of the EXTEND instructions in an effort to make them faster.

It seems that it would be possible to implement hardware that could handle the simple byte-move case often seen with MOVSLJ. It wasn't felt that this alone would help all that much without compare and convert instructions also.

One interesting possibility to speed up the translate function of some of the EXTEND instructions is to put some number of the most common translate tables into RAM to avoid a memory reference on each byte. The RAM would be loadable on microcode load and have translate tables that could be defined by each installation. The translate functions that weren't RAM resident would then run much slower.

In the absence of any firm consensus, we were left with the exercise of trying to determine how fast we could make a MOVSLJ-equivalent that takes two one-word global byte pointers, a single count, and doesn't have to store back updated byte pointers at the end.

5.0 Performance measurement hooks

Given the current attempt to extract performance information from the KL10, there was some desire to add some features to Jupiter to make it easier the next time around. The KL10 meter board is a fine example of a performance measurement feature which is very useful, but which most customers probably don't know (or care) that they have.

One suggestion for Jupiter was to bring the signals that we are interested in onto the backplane and to a spare module slot (if we have one) where they would be terminated by a dummy module. Performance measurement could then be done by building a few meter boards and plugging them into the spare slot in the backplane of the machine to be measured.

In addition to the meter board, we'd also like to have a low-overhead way of gathering opcode histograms, address traces and PC traces. With current RAM densities, it doesn't seem unreasonable to put opcode histogram hardware directly into the EBOX and allow it to count at full CPU speed. Similarly, it seems possible to put a small amount of buffering into the MBOX for address and PC traces and have the MBOX microcode write the buffer into memory when it fills up.

Finally, it would be very useful to be able to count the number of micro page faults that have happened.

6.0 Translation buffer organization

Because of the problems with the KL10 translation buffer organization, there was quite a bit of interest in the Jupiter TB organization. At present, the TB is implemented as a 2K, 1-way associative, 1-word block size cache in the MBOX. Given available RAM technology, the following organizations seem possible:

2K, 1-way associative, 1-word block size (total of 2K entries)
4K, 1-way associative, 1-word block size (total of 4K entries)
1K, 2-way associative, 1-word block size (total of 2K entries)
256, 4-way associative, 1-word block size (total of 1K entries)

An additional possibility was to implement separate user and exec tables, where each table was 1K, 2-way associative, and had a 1-word block size. This organization seemed to be the best of all possible worlds, followed by the 1K, 2-way associative single table. There was some question about the ability to implement the former organization given the timing constraints.

There was some interesting discussion about the possibility of including a user process context number in the TB and with every reference from the CPU. The user process context number would be a small (8 bits was suggested) number that would be unique for each user process. The advantage is that the TB need not be flushed on a context switch. Since the user process context number isn't of infinite size, the monitor would have to cause a full TB flush when it reassigned a number.

There was a request to provide a single physical page sweep function for the TB. This function would cause the TB to be swept looking for mappings to a specified physical page. If any are found, the mappings are marked invalid. At present, the monitor must do a full TB flush when the state of a physical page changes. This function would replace the full flush. Avoiding a full TB flush could be a real win if there are multiple user process contexts contained in the TB.

In order to make any rational decisions about TB organization, we need real address traces that we can run through a cache simulator. No one knows how to generate such traces for a running system without building some special hardware to monitor a KL10. We will try to solve this problem. To provide some preliminary guidance, Dan agreed to provide data from our in-house systems on process interaction, working set size, etc.

7.0 I/O strategy

There was considerable concern about the I/O strategy for Jupiter. The primary areas of concern are:

- I/O bandwidth. Is the I/O bandwidth sufficient to handle the projected traffic for a machine that is 4-5 times a KL10?

- Dependence on corporate front-ends. Is it wise to totally depend on corporate front-ends for all I/O when the groups responsible for the front-ends may have other interests?

- Performance impact of using front-ends for I/O. Front-ends are usually touted as off-loading work from the processor. In reality, they quite often add work through the use of expensive protocols [Cf. Dan's 24-Jan-83 memo on System IO Architecture]. Will we be creating a performance bottleneck in the front-ends?

There was considerable discussion about potential solutions to some of these concerns. One popular suggestion was to build a UNIBUS adapter for the machine in addition to the existing CI/NI adapters. The advantages of this scheme include the following:

- We aren't totally dependent on corporate front-ends, especially comm front-ends, for all I/O.

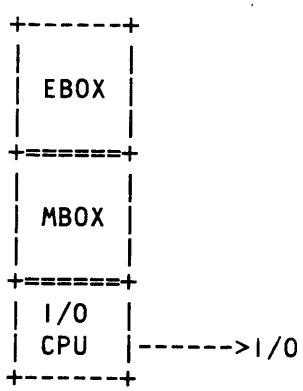
- We can immediately take advantage of any corporate advances in UNIBUS peripherals without having to wait for a front-end group to support them.

- There is potential for shortening the overall schedule since we may not have to implement assorted levels of protocol to talk to the front-ends.

In order to take full advantage of the UNIBUS adapter, it must have the capability of being a bus master so that an -11 is not required outboard of the adapter.

There was also a question of whether we should build an SI adapter to allow us to connect directly to the new corporate smart disks. Doing so would allow us an alternative to the CI/HSC connection to the disks. Arguments analogous to those listed for the UNIBUS adapter apply to the SI adapter.

One novel suggestion for a solution to the "front-ends don't off load anything" problem was the addition of a small -10 instruction set processor to handle I/O. Such a configuration might look something like:



That is, there would be a small -10 instruction set processor between the MBOX and the I/O replacing the IOBOX and possibly the ports. This processor would run normal monitor code from an on-board RAM loaded from main memory during monitor load and handle all I/O for the main CPU. Since the meeting Dan has indicated that such a scheme might offload up to 20% of the normal monitor overhead spent doing not only I/O but also scheduling, etc.

Finally, there was the question of encryption on the NI. There was consensus that we must have encryption at least up to the level that is available on the UNIBUS.