SESSION: HARDWARE

CHAIRMAN: Howard Wactlar

CM 181

AFFILIATION: Computer Science Department
Carnegie-Mellon University
5000 Forbes Avenue
Pittsburgh, Pennsylvania 15213

Telephone: 412-621-2600

THE DESIGN OF SUPER FOONLY

Speaker: David Pool, Stanford Artificial Intelligence Lab.

Abstract: Super Foonly is a Stanford designed processor with a PDP-10 instruction set but that will perform most instruction sequences an order of magnitude faster than a KI10. Most of its performance derives from its 2K cache and its capabilities are extended by a paging box.

OVERVIEW

Super Foonly was generated at the Stanford AI lab as the result of the familiar problem of running out of compute power. Our PDP-10 is connected to 30 display terminals, 3330 disk drives and special purpose hardware like TV displays and so it usually becomes overburdened in the early afternoons. When we first noticed the problem a couple of years ago we discussed it with DEC but decided that the then quoted price performance improvement of the KI10 was unimpressive. After some of us had managed to learn some details about large computer design form IBM personnel, we decided that we could build it ourselves. After a couple of months of preliminary work, two of us proposed to design and construct a plug in replacement of the KA10 which would use the memories already in house, the same peripherals, and require no change to user programs but that would provide a paging box and effect a program execution speedup of a factor of 10 over the KA10.

An average of 3 1/2 of us have been working on it for 2 years now although the original intention was that it would require just one year. We are currently in the final design stages and are busy laying out PC boards and wire lists. In early '73 we will be ordering parts, expect to have it constructed by June and debugged about 6 months later.

The machine will work with standard PDP-10 memories but Foonly operates four of them in parallel. Most of our memories are AMPEX supplied, double linked, 74 bits (currently being operated in 1/2 word mode).

The project started with simulations to which we fed real programs chosen from those running at the lab. The simulations predicted execution of 3 instructions per microsecond compared to measurements on our KA10 of slightly less than 1 every microsecond. The speedup is the main attribute but we introduced a few other features into the design.

MICROCODE

The machine is controlled by microcode implemented such that it is possible for some users to write modifications to it, perhaps to define new instructions. There are 512 words of writeable storage for the microcode of which 200 are permanently allocated for standard instruction interpretation. A user might also desire to place the inner loops of some long processes in the microstore which may give up to a factor of 2 additional performance as a result of eliminating instruction fetching and preparing. The decision to use microcode was influenced by Gordon Bell.

NEW ADDRESSING MODES

Other features users will see are two additional modes for indexing by the program counter and extended addressing (22 bits). In the former any indexing on register 17 will result in indexing by the program counter instead. In extended addressing mode the effective address is computed by adding the right hand 22 bits of an index register to the right hand 18 bits of an instruction or for indirect addressing the right 22 bits of the indirect word are used plus indexing if any (the indirect and indexing fields of the indirect word are shifted left). The program counter is also extended. All such memory references are in the virtual space and will be mapped by the paging box into real memory.

CONSOLE COMPUTER

The console will be replaced by a plug with which you connect Foonly to another computer known as the console computer. The latter will have a display and be capable of examining all the interior flip-flops of Foonly that would normally be connected to console lights (over 3,000 of them). It can also push the start, stop, deposit and examine switches. The main applications will be in debugging and maintenance but also for Foonly startup as its microcode memories are initially blank. We intend to make our current PDP-10 the console for Foonly. We expect to prepare simulators for the console machine that can single step Foonly through a program while running its own simulator over the same program to detect the first point of divergence. The console computer idea was suggested by John McCarthy.

DESIGN AUTOMATION

To aid in the design we generated a set of design automation programs to aid the designer in an interactive mode at a graphic display console. The programs help to lay out the circuit boards (hard copy is produced on a plotter) and then automatically prepare the wire lists.

SPONSORSHIP

Super Foonly is being sponsored by the Advanced Research Projects Agency of the Department of Defense not as a research project but simply as a solution to the problem of getting more computer power in the most convenient way. We feel the design has a lot more low cleverness in it than other computer design

but we don't claim to be pioneering any new concepts in computer organization. We would very much like to see DEC take over the construction, service and support of Super Foonly's but users must influence them to do so.

Q: What is the order of magnitude of Foonly's cost?
A: We expect to spend $200,000 to $250,000 for the manufacture of hardware. The design and personnel costs are not included.
Q: You are talking esssentially what looks like early KI10 prices plus the usual assortment of DEC peripherals so that for nice KI10 prices today you are talking about an order of magnitude better.
A: Hopefully so, and of course we're also talking about not having to buy all those peripherals. The idea is that you unplug your KA10 and plug in the Super Foonly, your same old peripherals will keep working.
Q: Where did the name come from?
A: Super Foonly has no acrnymic or historic significance.
Q: What class of gates are you using? What cycle times, etc.?
A: We are using TI Shotkey TTL. If we were going to start it now we would use the new brand of ECL instead. These gates have 5 nanosecond worst case delay compared to 12 nanosecond worst case delay for the KI10 gates, so the internal gates are 2 to 2 1/2 times faster. Of course the secret of the machine's speed is in the cache memory.
Q: What is the microprogram cycle time?
A: The internal machine cycle design goal is about 100 monoseconds. 125 monoseconds was used in the simulations that produced factor of 10 processing speedup estimates.

CACHE MEMORY

The secret of Super Foonly's speed is the chache or buffer memory (similar to that in IBM's new machines) which is a high speed, random access bipolar memory of modest size which sits between the main memory and the processor. We learned aout the idea from private communications before the model 85 was released but were very skeptical. John Cocke of IBM influenced us by going through a listing of our LISP interpreter and pointing out memory references that would hit the cache. We then wrote a simulator for PDP-10 programs which kept track of cache usage for a hypothetical cache like that described by Cocke and found that more that 90% of all memory references hit the cache.

Q: How big was the main memory, the programs, the cache?
A: We started with a 1K cache. We tried many programs including some 50K compiled LISP, some big interpretive LISP, some large hand-coded, and some compiled ALGOL. The final result was that for a 2K cache and for compiled ALGOL and interpreted LISP code, almost 99% of the memory cycles hit the cache.

The simulations showed that compiled programs and LISP interpreted programs were the best, compiled LISP and big hand-coded programs (like our ALGOL compiler) were the worst. The LISP compiler compiling itself and running compiled had a little over 95% cache hits. The interpretation of this is that most memory references are for instructions. LISP interpreters

spend much of their time in small loops and compiled programs spend their time doing strings of PUSH's, MOVE's, and MOVEM's. It is the hand coded tasks that leap around in clever ways following complicated control structures that suffer the lowest cache hit rate. We also varied certain cache parameters in the simulation but found that for a greater than 1K cache most results were insensitive to most parameters.

The cache simply attempts to keep around the most recently used words. We are basically using the algorithms of the IBM 370/165. Words are placed in the cache from main memory in a block of four words. Simulations showed the quadword to be much better than one word, somewhat better than 2 and preferable to 8. This gives the effect of instruction pre-fetching. Basically the cache tries to function as an associative memory. It has not 2000 separate words but 512 quadwords each with the address in real core at which the quadword nominally resides. Whenever the processor generates a memory reference to core, the cache is quickly searched to see if that location is already represented in there. If it is, no memory reference is necessary so the data is retrieved from the cache in one machine cycle of 100 nanoseconds compared to nearly a microsecond for data from core. If the desired word is not in the cache, a spot for it is found, if necessary tossing out the least recently used quadword and bringing in the desired one from memory.

We can't really afford (or know how to build) a 512 word associative memory that operates that fast so the cache really uses an approximation which is a hash coding algorithm with 4-way conflict resolution. Thus, for a real memory address there are only four possible places in the cache where it might be. The process of searching the cache then reduces to simultaneously checking these four locations which are a simple function of the address.

When the processor wants to store data (this is where we differ from IBM significantly) the same check on the cache is made. If the location is already referenced there, the new data is placed only in that cache position. IBM always causes data that is being stored by the processor to also go back to main memory in order to simplify problems of multi-processors and I/O devices. In our scheme, cache data does not get put into main memory until that location of the cache is needed for some new data to be put in. There is of course an explicit cache dump function. Our I/O devices are connected to a special adapter box that recognizes existence of the cache and resolves the problem of the data channel seeing a different copy of the word than the processor box just stored. Data channel data does not pass through the cache but a check is made to resolve any conflicts with it.

Our paging box is logically on the bus between the processor and the cache. Thus the cache is part of the memory system and addresses in it are physical memory rather that virtual addresses. This removes the problem of invalidating parts of the cache when a switch in paging mode occurs. Thus when a user transfers back and forth between his program and the system, to the cache it just looks like he is calling in subroutines in his own core image, therby eliminating inefficiencies specifically related to system calls.

The overall Foonly structure consists of the memories, the MAP (paging box), cache (M box), I/O adapter, and the processor proper which consists of 2 1/2 semi-autonomous units. The pager transforms user addresses by 512 word pages. Ignoring details of the data paths, generally data goes from the M box to the I box which is the instruction preparation unit. It decodes the instructions, calculates their effective address, fetches their operands, and tells the E box where in its microcode it should start. The data, which includes instructions and their operands then goes to the E box which just has input registers, adders, shifters, byters, bit reversers and output registers. The data comes out of the E box back to memory. The accumulators are an appendage to the E box so data comes out of and goes back into them. There is another autonomous unit called the instruction fetcher which fetches ahead 2 to 3 instructions sequentially without interpretation. Thus we have four autonomous units, the M box, the instruction fetcher, the I box and the E box, each performing its own function each machine cycle, all synchronized off the same clock. The boxes make requests to the M box which determines their priority and decides which one to service. Otherwise, there is overlap between the boxes which effects a 2 or 3 stage instruction pipeline. This flow diagram may be recognized as that of a model 85.

The main microcode control is in the E box. It has 512 144-bit words which perform most of the control. The I box has its own microcode control memory to provide information about the fetching of operands and decoding of instructions. Thus, you really can change the instruction set because there is no connection between the opcodes and what the I box does to them. It all goes through the I box control memory.

The I/O adapter interfaces I/O processors (e.g. data channels, display processors), i.e. units which in your current PDP-10 have a memory bus of their own. They will be plugged instead into the adapter box which has something like 6 simulated DEC memory buses coming out of it, so the device will not be aware that it isn't still talking directly to memory. The I/O adapter has its own bus connections to physical core, fetches quad words, stores and buffers them for each I/O device. This increases device speed and reduces memory interference. The primary function of the box, however, is to check and resolve conflicts with the cache contents on every transaction since data going to and from I/O devices does not ordinarily pass through the M box.

Q: Did your previous work convince you or will there still be some things left to tune?

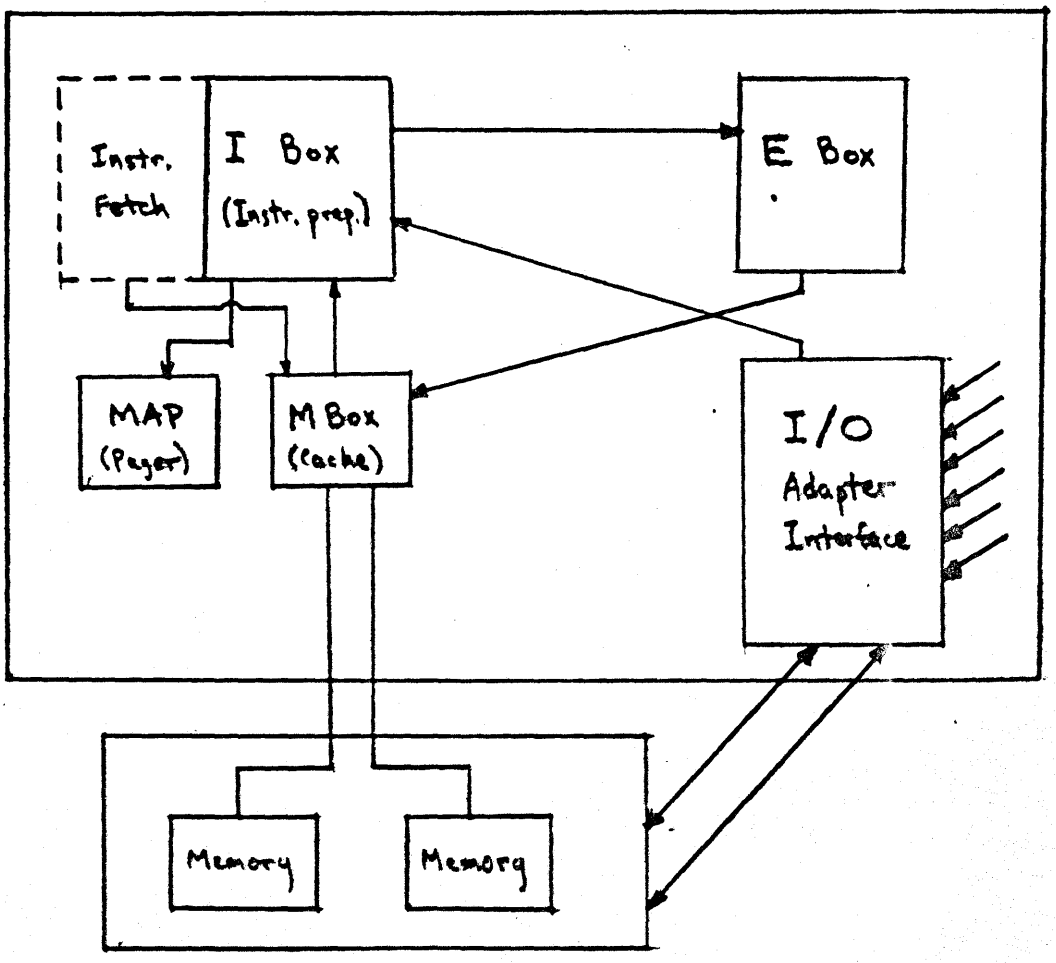
A: There is not much left to tune. Hardware constraints determined a lot of the factors: the quadword, 4-way conflict resolution, and 2K overall cache size are all hardware constraints. The way we are using the busses is essentially a hardware constraint. It would have been better to have four buses.

Q: Can you wind this discussion down in some way?

A: Buy Foonly!

Q: What is your processor's relative internal speed compared to a KA10?
A: All ordinary instructions take one cycle plus whatever is necessary to fetch data, so an add between 2 accumulators takes 1 100-nanosecond cycle and an add to memory when the data comes from the cache takes 2 100-nanosecond cycles. That's much more than a factor of 10 over the KA10, but it doesn't hit the cache all the time.
Q: Did you try your cache simulations on the monitor as well?
A: No, we have no way of simulating the execution of monitor code. You can simulate the execution of user code just by loading the simulator into the same core image as the user code but you can't do that with the monitor. We suspect the monitor may be the worst case, but most monitor code is executed at UUO level at user request and thus there is no difference in having that code in the monitor and having it in the user program. This does not apply to interrupt level operations including schedulings.
Q: Won't jobs run at different elapsed CPU speeds depending on what happens around them?
A: Yes, job speed depends upon what's in the cache.

PROCESSOR ARCHITECTURE



PRICE REDUCTIONS AND PRICING POLICY AT DEC

Speaker: Bill Kieswater, DEC, Maynard, Mass.

DEC'S POSITION RELATIVE TO SUPER FOONLY

DEC is looking at Foonly and has been working with the staff at Stanford. We are one of the vendors but the extent of our participation is at the component level, not the system design. We are also making use of some of the design aids that they have developed. You can be assured that the 10 line will take advantage of these and similar advancements by our own research groups as they prove commercially feasible in terms of the tradeoffs.

PROCESSOR PRICING

The price of the KI10 has been reduced from $380,000 to $240,000 (U.S. dollars). The KA10 price remains the same, but should just the processor be required for an upgrade to a supported dual processor system, the incremental cost is $130,000. Similarly, a second KI10 processor should you already have a KI based system is $200,000. These savings result because the software installation and support component has been removed.

NEW MEMORIES

DEC is now producing memories in house. For the PDP-10 the new ones are designated MF10A and MF10G. The MF10A is a 32K 1 microsecond module in a 30 inch cabinet, compatible with any existing system in the field and priced at $50,000. The MF10G is 64K, 1 microsecond, fits in a 30 inch cabinet and sells at $80,000. Expectations are that as in-house memory production increases to satisfy the minicomputer business, DEC-System 10 memory will be coming down further in price. We are also looking at in-house production of peripherals such as swapping media, file storage, magtapes other than the TU10, but not line printers.

EQUIPMENT UPGRADE PRICING

The following upgrade pricing was presented:

From	To	Price
KA10	KI10	130,000
TM10A	TM10B	8,000
RP10A	RP10C	9,000
CR10A	CR10E	12,000
LP10A	LP10C	36,000
TU20	TU40	22,000
TU30	TU40	17,000
DS10	DC75	40,000
MB10	MF10	22,000
MB10	MF10A	42,000
MB10	MF10G	72,000
MA10	ME10	20,000
MA10	MF10A	40,000
MA10	MF10G	70,000
MD10G	MF10G	45,000

MD10E	MF10A	35,000

These upgrades are offered regardless of the condition fo the device unless it has been physically modified. The field service charge for the installation and checkout is extra and is quoted on a local basis as a function of distance to the nearest field service office and so forth. I believe they have standard prices to do these upgrades.

The intention is to offer customers an upgrade from low performance devices to high performance ones as their systems expand and usuage grows. As we develop products of a higher performance nature they will be offered as upgrades to the lower performance ones. I expect to see this as a formalized policy in the near future.

[The following are summarized from the question/answer session which followed ...ed.]

DISKS

There is no upgrade from RP02's to RP03's on the same controller because the pricing probably would not be attractive enough.

DEC is committed to providing something beyond RP03's, in the 3330 category,in the future. There are a lot of contingencies as to whether it be produced in house or on the outside. The RP04 has been discussed but no formal announcement of it has been made. It may not be offered for more than a year.

MEMORY, MAGTAPES, DATA CHANNELS

There is no ME10 to MF10 upgrade probably because there have been no requests. DEC probably could offer one.

About 18 users indicated an interest in a low performance , low price but 1600 BPI magtape drive for compatibility with other vendors.

A 22 bit DF10 data channel for use with KI10's is currently being worked on as is a 22 bit MX10 memory port expander to go with it.

TENEX

DEC will supply an unsupported version of a TENEX monitor that runs on a KI10 for $15,000. The policy does not permit DEC to write any acceptance criteria for TENEX so the extent of the warranty is limited. TENEX was originally developed by Bolt, Beranek & Newman (BBN) to run on a modified KA10 (1500 wiring changes) with a BBN paging box. DEC has simulated the BBN pager in software on the KI10 and thus produced a TENEX monitor for it which seems to run at least as well as the KA10 version. No comparative performances measurements have yet been made. TENEX was designed around a limited variety of DEC furnished peripherals so it may not support all configurations in the field. There are no plans inside DEC at this time to support this monitor. The SPR's will be forwarded to BBN and they will handle them on a best effort basis.

The FORTRAN optimizing package will lease for $125 a month. It will be sent on the distribution tape.

APL

The current status of APL does not permit DEC to offer it to service bureaus as a result of a contractual agreement the owners of the language have with a service bureau. That agreement expires November, 1973. If there is enough interest DEC will pursue it. There are two versions of APL: the basic version leases for $300/month, the advanced version for $600/month. The latter includes files and other features not available in IBM versions. Each version will work on either 2741's or teletypes; the APL character set is not a requirement for its use.

UNBUNDLING

DEC is unable to predict what items will be unbundled in the future. They view unbundling as a way of recouping funding now required to maintain the product in a first class state. There is a large group who do quality assurance, software evaluation, documentation and field support that incur indirect charges to the programming effort over and above just the development programmers. As the prices of hardware goes down it reduces the margin to support such activities.

DEC supplies and separately prices APL and data base management because there were strong requirements in specific markets for them. The selection of those items to be unbundled will probably be market driven and oriented toward the more specific application areas.