

INTRODUCTION

00100
00200
00300
00400
00500
00600
00700
00800
00900
01000
01100
01200
01300
01400
01500
01600
01700

I shall attempt to explain what Super Foonly is all about to those having only a knowledge of the DECsystem-10 (alias the PDP-10). This description may well include some DEC-centered bias, since its author's salary is being paid by the aforementioned computer manufacturer.

Super Foonly is touted by its designers (primarily Messrs. John Holloway, Ed McGuire, Ted Panofsky, Phil Petit, Dave Poole and Fred Wright [alphabetically]) as a replacement for the KA10 which runs ten times as fast. It is being designed at the Stanford Artificial Intelligence Laboratory, funded by the Advanced Research Projects Agency of the DOD. As of this time (25 May 72) the design is not yet complete. The designers estimate 2 to 3 months more design effort, but I would double that estimate without much fear of being too short.

00100

GENERAL DESIGN APPROACH

00150

00200

00300

00400

00500

00600

00700

00800

00900

01000

01100

01200

01300

01400

01500

01600

01700

01800

01900

02000

02100

02200

02300

02400

02500

02600

02700

02800

02900

03000

03100

03200

03300

03400

03500

03600

03700

03800

In order to achieve the desired speed-up factor, several new approaches were taken in the design. The primary departure from traditional DEC designs is the inclusion of a 2048 word high speed "cache" memory in the processor. Stanford simulations indicate that in excess of 95% of all memory references will be found in the cache. For those unfamiliar with cache memories, I will explain them in more detail later.

Super Foonly consists of several "boxes", all physically located within two fairly standard DEC 30" cabinets. The boxes include the "I" box which prepares instructions by doing the effective address calculation and doing the decoding necessary to determine the operand requirements and fetching those operands; the "E" box which does the execution of the computations; the "M" box which holds the cache and memory mapping hardware; the "I/O" connection box which filters all I/O memory operations; and the "IOT" box which performs I/O bus operations.

The machine operates in a pipe-line fashion, with all boxes operating concurrently, generally performing their various functions on a sequence of instructions. Through this means, it is expected that simple instructions can be done in one or two machine cycles of 100 ns., for an average instruction time of, say, 200 ns. compared to the >2 microseconds of the KA10.

The circuitry used is Schottky TTL, generally known as "74S" series. Most of the working registers of the machine are built as "latches", consisting of 2-2-3-4 and/or-invert gates and inverters. This scheme provides the fastest speed possible, while losing some of the convenience of D-type flip-flops. Most of the boxes use large scratchpad memories, consisting partially of 74200 256 bit chips, and partially of AMS 1503 128 bit chips where higher speed is desired.

Circuits will be mounted on quad-height, double width boards. In the prototype, control logic boards will be wire wrapped, while data path boards will be printed circuit. There is a possibility that some boards will have both printed circuit and wire wrap.

00120
00220
00320
00420
00520
00620
00720
00820
00920
01020
01120
01220
01320
01420
01520
01620
01720
01820
01920
02020
02120
02220
02320
02420
02520
02620
02720
02820
02920
03020
03120
03220
03320
03420
03520
03620
03720

I BOX DESCRIPTION

The I box prepares instructions for execution. It contains the Program Counter (PC) and the accumulators. There is only one block of AC's (like the KA10), however, there are 3 copies of them. There is also an Advanced PC which supplies the addresses for the two possible prefetched instructions.

In operation, the I box has the current instruction in a register called the IIR. The effective address is calculated using one copy of the AC's which always supplies the XR, if needed, using an adder reserved for this purpose. Another copy of the AC's decodes the "AC" field, while the third copy supplies AC+1 or c(E) as required by the instruction. The instruction part is sent to the I box control memory, which supplies what DEC calls the fetch and store cycle switches, as well as a starting address in the E box control memory.

Should the instruction require memory data, a request is issued to the M box for the word required. In any case the data is presented to the E box for its use as soon as it can take it. The I box is usually freed to go about its business for the next instruction as soon as the data is passed to the E box. This frequently means that the I box will have the next instruction prepared in 1 or 2 cycles, which overlaps with the typically 1 cycle taken by the E box to do the execution.

There are, of course, complications which make all of the above decidedly non-trivial. First there is the matter that the E box does not return AC results until the end of an instruction execution. Should an instruction use an AC being modified by the previous instruction, it would normally take an extra cycle to write that AC, and make it available again, so numerous comparitors exist to detect these cases, and provide direct "loop-around" paths, where possible, for the E box to use its outputs directly. Another problem involves conflicts between words being stored by the E box in memory, and either instructions or data being pre-fetched. Much hairy logic exists to detect and provide for these cases.

00100
00200
00300
00400
00500
00600
00700
00800
00900
01000
01100
01200
01300
01400
01500
01600
01700
01750
01800
01900
02000
02100
02200
02300
02400
02500
02600
02700

E BOX DESCRIPTION

The E box, as mentioned above, actually performs the calculations. It has a 512 word by 144 bit control memory which actually controls its operations, however bits of the instruction register are used to distinguish between similar instructions. Thus, although it is claimed that arbitrary new instructions can be microprogrammed, much of the logic is closely tied to existing instructions. Microprogramming will likely be painful even to a system wizard.

The E box features such goodies as a complete 72 bit shift matrix, allowing all shifts to be done in one cycle; a normalizer which can normalize floating point numbers in 1 or 2 cycles; an adder with two complete sets of ALUs, such that one set has permanently true carries and the other has permanently false carries, and the next mixer stage selects between them in 4 bit groups according to the actual carries; and a 4 bit at a time multiplication algorithm.

The main E box data path circulates thru 3 sets of latches, although there are times when the cycle is thru only two latch registers. The way this works is that while the data is being held in the first set of latches, the second set is opened to receive data. The second set is then locked (or latched up, if you prefer). Its inputs are deselected by inverting the hold signal. The first register is then unlatched, and set to look at a function of the second. After enough time for all data to arrive and circulate thru the latches has been allowed, the first latches are locked up, followed by deselection of the inputs, and the cycle repeats.

00100
00150
00200
00300
00400
00500
00600
00700
00800
00900
01000
01100
01200
01300
01400
01500
01600
01700
01800
01900
02000
02100
02200
02300
02400
02500
02600
02700
02800

M BOX DESCRIPTION

The M box contains the cache, the map and the interface to core memory. All memory requests in the system (including direct to memory devices) are filtered thru the M box, as this is the only way to avoid stale data from accumulating in the cache.

The cache consists of 4 blocks, each containing 128 quadruple data words and 128 address words for a total of 2048 data words. Locations in the 128 word dimension are binary selected from bits 27 thru 33 of the address. Words in the quad-word slice are selected by bits 34 and 35. Thus, independent of any high order address bits, 4 possible candidate words and their associated address words are selected from the cache, one from each block. These address words contain the physical page number (bits 14 thru 27) of the quad of data words. The physical page number from the map is then simultaneously compared to the 4 selected address words to determine whether any of them match, and if so, the data from that particular word is selected.

The map is reminiscent of the BBN map, in that it has hardware provisions for indirect and shared pages, and bits which tell whether a particular page has yet been referenced or written into. The map takes 14 address bits in (bits 13 - 26) and gives 13 physical address bits out, which are the physical page number mentioned above. The high order 5 virtual address bits are provided by the "extend mode" feature, which causes indirect and indexed references to use bits 13 - 35 as address bits, and bits 0 - 4 as the index and indirect fields of indirect words. This mode is under user control, and can also be entered for a single instruction by a special execute instruction.

02900
02950
03000
03100
03200
03300
03400
03500
03550
03600
03700
03800
03900
04000
04100

I/O and IOT Boxes

Words going between direct memory access channels and memory are also checked against the cache to insure that the cache copy of any given word does not get stale. The memory cycles necessary for I/O words are still taken from core thru their own memory busses. Since only one cache cycle is taken for every 4 words, even a 2 microsecond per word transfer device uses only one cache cycle out of 80. The buffers and interfaces for this purpose constitute the I/O box.

Control of the I/O bus is maintained by the IOT box which can run concurrently with the other boxes. Super Foonly has most of the KI10 I/O bus features. Requests for PI DATAI's and DATAO's are handled by the IOT box without disturbing the main instruction stream, except for the necessary cache cycles.