To: Distribution

From: Robert S. Coren

Date: 09/21/81

Subject: Use of Extended Memory for FNP Buffers

This MTB describes a proposed mechanism for using extended memory in the Datanet 66XX FNP (18X) for data buffers. A certain amount of familiarity with the workings of the Multics FNP is assumed, although a brief summary of the extended memory feature is included for those who need it.

STATEMENT OF THE PROBLEM

Limits on the amount of available buffer space pose a serious problem for the Multics FNP software. The fact that executable code and buffers used for I/O to and from the various channels have to share the directly-addressable 32K 18-bit words of basic memory restricts not only the number of channels that can be run successfully on a given FNP, but also the number of different types of channels that can be run, since each additional line type may require an additional module of control tables. Thus as channels and channel types are added and available space drops, throughput for all channels decreases, the notorious "bell/quit" problem manifests itself, and with some configurations the FNP will not boot at all, or will crash very shortly after bootload.

Starting in MR8, we have made very limited use of the extended memory feature of the FNP (i.e., the ability to use addresses higher than 32K). The extended memory is currently used exclusively for data bases that are allocated at initialization time for the duration of the FNP bootload, as explained below under "Summary of the FNP Paging Mechanism." The effect of this is that the available extended memory is largely wasted. In fact, although additional memory for an FNP is available in 32K increments up to 256K 18-bit words, Multics supports no more than 64K, and doesn't make effective use of more than about 48K.

SUMMARY OF PROPOSED SOLUTION

I propose to modify the FNP software to allow I/O buffers to be allocated anywhere in available memory, with preference given to high (i.e., extended) memory. This will free a substantial amount of space in low memory for additional code and control tables if needed, and greatly reduce the probability that buffer space itself will become uncomfortably tight. (Space allocated temporarily for purposes other than I/O buffers will continue to be allocated in low memory.) This will make it possible to make effective use of whatever memory is available, and accordingly we can remove the arbitrary 64K limit, thereby making it feasible to sell an FNP with as much extended memory as a customer wants/needs, up to a full complement of 256K.

The proposed implementation will be described in more detail following a brief description of the FNP paging mechanism.

SUMMARY OF PAGING MECHANISM IN THE FNP

The architecture of the Datanet 66 series of machines allows 15 bits for an address that can be referenced through a machine instruction (e.g., as an indirect word, or held in an index register). This restricts the directly addressable memory to 32K, i.e., the highest referenceable address is 077777 octal. However, if extended memory is enabled, a "paging" mechanism is used to reference addresses up to 256K (777777) by dividing the directly addressable 32K into 256-word "windows" onto arbitrary 256-word blocks.

The way this works is that, if the paging hardware is enabled, a page table of 128 1-word entries (at a fixed location in low memory) is used to map 15-bit addresses into 18-bit addresses. Each page table entry describes a 256-word region of 15-bit addresses, where the region described by each entry is determined by its position in the table; such an entry includes, among other things, an "active" bit and the address of a 256-word block in extended memory. If an entry's active bit is on, then all references to addresses within that entry's region are translated to addresses within the block in extended memory to which the entry points.

Perhaps this discussion can be clarified by a description of how the current FNP software makes use of the paging mechanism. The only page table entry ever made active is the last one, i.e., the one describing addresses 77400 through 77777. Suppose the software needs to reference data at extended address 123463. The page table entry is set to point to the block starting at 123400, and its active bit is set. An instruction reference to 77463 will now in fact access the contents of 123463, and similarly any reference in the range 77400-77777 will access the corresponding address in the range 123400-123777.

A range of addresses such as 77400-77777 in the example above is referred to as a "window" or "paging window"; the range of addresses from 123400 to 123477 is called a "page"; the 15-bit address used in the instruction (77463 in this case) is called a "virtual" address; and the 18-bit resultant address (123463) is called a "real" or "absolute" address.

In the current implementation, the primary use for extended memory is for terminal information blocks (TIBs) and software communication regions (SFCMs), of which one of each is allocated for each communications channel at FNP initialization time. They are allocated starting at the base of the second 32K (i.e., starting at 100000), and in such a way as to ensure that the TIB and SFCM for any channel are in the same page. When work is to be started on behalf of a channel, either its TIB or its SFCM is found first, depending on what module is involved; in either case, the page table word for the paging window is set to point to the page containing both. Since the TIB and the SFCM each contains the address of the other in virtual form -- i.e., the address at the appropriate offset in the window -- it is possible, given the address of one, to load a usable address for the other directly into an index register.

The only other use currently made of extended memory is for the trace buffer, which is allocated at the upper end of the second 32K (i.e., so that it ends at 177777, the highest address available in 64K). The trace routine changes the contents of the page table entry while updating the trace buffer, and restores it when finished. If metering is not enabled, two TIB/SFCM pairs fit in a page, so a maximum of 12K can be occupied by TIBs and SFCMs. With metering enabled, each TIB/SFCM pair occupies more than half a page, so up to 24K may be reserved for TIBs and SFCMs. Thus even on a fully-configured FNP with meters, the minimum space available for the trace buffer is 8K. This is enough for most purposes; trace entries vary in length from 2 to 6 words.

High Memory in I/O Operations

If appropriate flags are set, absolute addresses can be placed in the ICWs used to transfer data to and from HSLA subchannels, and likewise in the DCWs used by the DIA. This means that I/O can be done to/from extended memory without involving the paging mechanism. At present, this feature is only used for storing status from HSLA subchannels (this status is stored in the SFCM), but it will clearly be useful if extended memory is used for buffers.

Setting the Page Table Entry

The utility routine setptw is used to translate an absolute address to a virtual one, setting the page table entry to point to the appropriate page as it does so. In addition, the sections of code in the dispatcher that save and restore registers also save and restore the contents of the page table entry.

DETAILS OF THE PROPOSED IMPLEMENTATION

Additional Paging Window

The range of addresses 77000-77377 will become a second paging window reserved for buffer addresses. This means that one additional page table entry will be used. The window at 77400 will continue to be used for TIBs and SFCMs. In general, only one buffer address will be available at a time; to switch between buffers, it will be necessary to modify the buffer page table entry. This also implies that a buffer cannot be allowed to span a page boundary (see below).

Space Management

The space allocation and freeing routines will be extended to allow for the use of high memory. An additional allocation entry will be provided that is called when allocation in high memory is desired. (If the allocation cannot be made in high memory, low memory will be used if possible.) A separate free chain will be maintained in high memory. This chain will treat each page as a separate free block, i.e., free blocks that are adjacent but in different pages will not be combined.

Since all the buffer sizes used are multiples of 32 words (with one exception -- see below), and the maximum buffer size is 256 words, it will be fairly simple to enforce the restriction that a buffer cannot span a page boundary. Currently, every buffer is allocated at a 0 mod 32 address; this restriction will be extended to require buffers to be allocated as follows:

size (words)	boundary
32	32
64	64
96 or 128	128
>128	256

Small Input Buffers

Input on asynchronous channels (when not in blk_xfer mode) is currently put in 8-word buffers, of which two are allocated for every channel in receive mode (which in general means all dialed-up asynchronous channels all the time). In the new scheme, these buffers will be increased to 16 words (which will help to alleviate some problems arising from input bursts) and allocated permanently in extended memory, rather than being taken from the buffer pool, as explained below under "buffer preallocation."

Echo Buffers

Echo buffers will be allocated in extended memory, but some care will have to be taken, since an echo buffer contains input and output pointers into itself for use in updating. These pointers will be virtual for easy manipulation; the output pointer will be converted to absolute when echoing is initiated.

Continued Use of Low Memory

Dynamic allocations for uses other than data buffers will continue to be made in low memory. These include CCTs, TIB extensions, delay tables, scheduler queues, DIA request queues, echo negotiation break tables, etc. These represent comparatively little of our use of allocated memory, and since one or more of them often has to be used in conjunction with a data buffer, putting them in extended memory would necessitate either one or more additional paging windows or the constant juggling of pages. The CCT, in fact, is referenced by the hardware through a field that is insufficiently wide to hold an address above 32K.

Address Conversion

The setptw routine mentioned above will have to be extended to be able to set the buffer paging window entry as well as the TIB/SFCM entry. Currently there is no subroutine to convert a virtual address to absolute form; in the few cases where this is needed, the conversion is done inline. However, with buffer addresses in high memory it is expected to become a much more common operation, and a subroutine will be provided to "absolutize" a virtual address.

Buffer Preallocation

A scheme has been developed by some programmers at FSO, and is currently under test at CISL, whereby a pool of buffers is allocated in advance for use by high-speed synchronous channels, thereby allowing them to switch ICWs at interrupt time without incurring the expense of buffer allocation (the pool is refreshed at more leisure later). This greatly alleviates one of MCS's more severe timing problems, whereby failure to switch ICWs fast enough could cause data to be lost. This feature will be incorporated into the use of extended memory, where it will tend to offset the increased expense of buffer allocation in general (see below).

A side effect of this mechanism is an increase in the size of the SFCM, so that two TIB/SFCM pairs will never fit in the same page. The amount of unused space left in a TIB/SFCM page will be 98 words if metering is not enabled; if metering is enabled, the space will be 52 words for a synchronous channel, or 74 words for an asynchronous channel. For asynchronous channels, this space will be used for the small input buffers referred to above, which will be permanently allocated at initialization time; for synchronous channels, it will be added to the buffer pool.

Executable Code

There are no plans at present to attempt to put executable code or control tables modules in extended memory. To do so would introduce some very difficult problems. In particular, in order for bind_fnp to do relocation properly, code to be put in high memory would have to be restricted to self-contained subroutines of no more than 256 words apiece, and they and their callers would have to coded very carefully; we would have to modify extensively the binder and any code that we wanted to relocate in this fashion. This was done in GRTS-II, and according to one of the implementors, "it's very easy to do it wrong." It seems unlikely at this time to be worth the time it would require.

Space Saving

It is difficult to determine accurately how much memory space is used for I/O buffers, and thus how much low-memory space is gained by putting buffers in extended memory; it varies not only according to the number, type, and speed of the channels configured, but also depending on how active the channels actually are. Using the rules of thumb given in MAM Communications for estimating required buffer space, an FNP with 96 asynchronous channels at speeds up to 1200 baud, of which half require an echo buffer, uses about 12K for I/O buffers. If 12 such channels are configured, along with 12 9600-baud synchronous channels, about 8K would be used.

IMPLICATIONS

The above proposal has several (possibly negative) implications, which are summarized below.

Efficiency

The FNP software is going to have to do more work. In particular, the space allocation and freeing subroutines will have to manage upper as well as lower memory, and modules all over the system are going to have to go to the extra expense of converting buffer addresses between virtual and absolute. Since FNP CPUs are generally not overburdened, and FNP performance (in terms of instructions executed) is rarely, if ever, a bottleneck in the system as a whole, this may not be a major concern. Whether this increased work will have a measurable effect on FNP throughput (in terms of characters per second) is unclear; further study will be necessary to determine this. The preallocation of buffers mentioned above will tend to avoid problems that might otherwise arise from local inefficiencies threatening to make critical functions (like the switching of ICWs) too slow.

The extra code involved will, of course, take up more space as well; however, the space penalty is unlikely to be significant compared to the space saved by putting the buffers in extended memory in the first place.

Continued Use of DN355

It is already true that Multics does not support the use of extended memory on the Datanet 355, mostly because the 355 does not support the direct addressing of high memory in ICWs. This is managed at present by having the binder recognize certain symbols and replacing transfer instructions with no-ops depending on the amount of memory specified in the bindfile. One result of this mechanism is that a 355 makes frequent calls to setptw, which returns without doing anything. The implementation proposed here will result in a great proliferation of such calls, along with other code that would have to be transferred around under control of the binder. In addition, the space penalty referred to above would be significant in a 32K FNP, where every word counts. It seems preferable to freeze 355 support at its present level, and add the feature described herein to the DN66XX only; furthermore, any other features to be added to FNP software would not be added to the 355, so as to avoid having to maintain and update two parallel versions of the system. Honeywell no longer offers the 355 in any case. We might have to make an exception if we make a change in ring 0 that requires a correpsonding change in the FNP, in case any of our customers still have 355s.

Trace Buffer

Sites with 64K FNPs may have become accustomed to the idea that the size of the trace buffer is virtually unlimited, and are therefore using trace buffers on the order of 20K. (This is reasonable at present, since the space thus used would otherwise be wasted.) Under the present proposal, the trace buffer would once again be competing with I/O buffers for memory. This is not a serious problem, but will require a notice in the SRB to warn sites with large trace buffers of the implications.

TASKS AND TIME ESTIMATES

The individual tasks required to implement this proposal are summarized below, along with very rough time estimates for each task.

	Weeks
Modify buffer allocation and free routines, add new entry points	3
Page table management: add entry to setptw for buffer window, add subroutine to "absolutize" an address	1
Initialization: set up extended memory free chain and buffer paging window	1
Find all places where buffer addresses are loaded or stored, to make sure address conversion is done if necessary	3
Find all places where buffers are allocated, so as to call the subroutine that allocates in extended memory	1
Miscellaneous front-end modifications: scheduler to save/restore both page table entries; metering changes; ICW and DCW setup	2
Host changes:	
metering	1
cv_cmf and bind_fnp to accept sizes above 64K	.5
Ability to dump more than 64K	.5
Testing of entire package	4
Documentation (mostly AN85)	1
Total	18

INCLUSION IN THE RELEASE

This proposal is intended for MR 10.0. If it is done, it will be inextricably intertwined with anything else that is done to the FNP code, so there would be no sensible way to remove it from the release if for some reason that appeared desirable.