Modifications to UNIX† to Allow Four Mega Bytes of Main Memory on a 11/40 Class Processor

Clement T. Cole

Computer Research Group
Computer Research Laboratory
Tektronix Laboratories

Sterling J. Huxley

Computer Resource Department
Technology Group

ABSTRACT

This paper describes the modifications to UNIX Version 7, to efficiently use the 4 Megabyte extented addressing capabilities of a 11/40 class processor after hardware modification. Discussed are the reasons why such changes are desirable, what parts of the machine are effected and a few comparisons of the results.

June 20, 1981

† UNIX is a Trademark of Bell Laboratories.

Modifications to UNIX† to Allow Four Mega Bytes of Main Memory on a 11/40 Class Processor

Clement T. Cole

Computer Research Group
Computer Research Laboratory
Tektronix Laboratories

Sterling J. Huxley

Computer Resource Department
Technology Group

Introduction

We will be describing the changes that were made to a standard UNIX system to allow the addition of extra main memory beyond that DEC normally allows. We will first be discussing the reasons for such radical departure from a normal system. Next we will discuss the changes made, and the effects that these non-standard hardware and software changes have to a user program. Last of all we will look at some performance statistics and see how they relate to the earlier justification for this work being started.

History

In 1972, Ritchie and Thompson released to the world the Fifth Edition of the UNIX Timesharing System[1]. This system was designed to run with minimal hardware. In 1978, the Seventh Edition of UNIX was released with the caveat that it must run on larger PDP/11 systems. The term "larger" was intended to mean DEC PDP 11's with a hardware kludge to allow a seventeenth address bit. This kludge is known as the separate instruction and data bit[1]. This bit separates instruction and data fetchs to main memory on the Unibus[2]. The term "minimal" included the 11/40 and 11/35 processors that do not support the separate I/D feature. In those 6 years and two releases of UNIX what happened?

For a first order view, UNIX was improved, at the expense of size. These improvements were assorted, from simple bug fixes, to modernize an original design decision that became inappropriate. The fundamental change was to correct for a hardware problem

† UNIX is a Trademark of Bell Laboratories.

1] Another term for this feature is: separate I and D

that became acute over those same 6 years: disk technology improved drastically, forcing the earlier file block[2] to physical disk block mapping to be unusable.

The Seeking Problem

When UNIX Version 6 was released, UNIX was forced into viewing each large, DEC RP03 style, disk as many smaller pseudo disks. This feature was introduced because the fundamental data size on a PDP 11 is a 16 bit integer. An RP03 has more than 65536 physical blocks per disk. Later DEC introduced the RP06 disks that contains about 150 Megabyte of storage. Technology was bound to improve again, making the pseudo disk solution a little unbearable. Clearly, there are times when a user would wish to address every physical block on the disk.

Generic UNIX does supports a system call to allow a user program to address any byte in a file. The *seek* system call takes three parameters[3]. The descriptor of the file to seek on, the amount to seek, and a third parameter that specifies the type of seek, i.e. seek relative from the beginning, end or current file position. The UNIX operating system supports a data abstraction that allows a user program to view any device as a sequential file of bytes. Using the seek system call and this data abstraction, we can write programs to manipulate any byte on any disk[3]. Unfortunately, as we discussed before, the operating system had difficulty building the complete abstraction for large disks. These principles beget Version 7's new seek system call.

In Version 7, the seek system call was removed and replaced with a lseek (long seek) system call[4]. The difference between the two calls is that the parameter that specifies a seek count, was changed from a short integer (16 bits), to a long integer (32 bits).

The change from a short to a long had major ramifications. Not only did it nearly double the size of many of the already large tables that the operating system needed to keep in main memory, but in the operating system size was increased in tiny increments by arithmetics. What used to be a add instruction, now needs to be an inline expansion for 32 bit arithmetic. Not that these changes are terrible, but many of these expansions have the tendency to slowly add to the size of the kernel. See the appendix

---

2] A block is the fundamental unit of storage on a disk drive. It contains 512 bytes of 8 bits each.

3] An example of a program that would like to manipulate an entire disk at a time is a program that operates on entire file systems, like dump(1m), restor(1m), and icheck(1m).

for a demonstration of the code expansion problem.

Solutions to Address Space Limitations

The Version 7 UNIX Kernel (adding both instruction and data) would not fit into the 64k byte limit that a PDP 11/40 class processor imposes, and still be able to have a several runable processes in the kernel process tables. To be truthful, large UNIX Version 6 systems had much difficulty fitting into the 11/40's address constraints. Many different groups created ingenious schemes to win back kernel space. The most notable were the Calgary Buffer Modifications that moved the system I/O buffers out of the normal kernel address space and made the system change its memory management registers to address the data in those buffers. For small 11/40 installations these modifications were not completely necessary. However, if these modifications were not installed, UNIX would have to be "shoe horned" into fitting the address space available from the DEC 16 bit virtual addresses. When Version 7 came, these modifications became necessary for even the smallest 11's to be able to run more than five or six processes (not users) at the same time. Soon, another interesting method was suggested by people at UC Berkeley[5]. They suggest that the kernel be "overlayed." This overlay scheme is not a classical disk overlay, but for efficiency, an in memory overlay. Meaning that the whole kernel would be kept in main memory at one time, and the memory management registers changed as needed.

The "in-memory" overlay method, when it is used with the Calgary buffer modifications, seems to be among the best methods of gaining back kernel address space to date. Unfortunately, these modifications have one unhappy side effect. They trade off kernel size at the expense of available user memory. A PDP-11/40 class processor's memory management unit can only address 18 bits (256 K bytes) of main memory. It is easy for a UNIX kernel to become greater than 1/2 of that size and in some cases be closer to 3/4's of our 256 K byte limit. Remember that the reason for letting the size of the kernel become large in the first place was to allow more that a few processes to try to run at the same time. We now have enough kernel buffers, but not enough main memory; and unhappy Catch 22. To this end, the software people asked the hardware people for a little help.

Able Computer Corp. produced exactly what was needed. They have a new memory management board called: **ENABLE**. This board works in conjunction with the DEC memory management unit to produce 22 bit's worth of address on a machine that previously had 18 bit's worth. Making the theoretical size, of the

address space of this new hybrid 11/40 class processor[4], 4 megabytes.

The ENABLE board plugs into a new backplane and creates a new 22 bit Modified Unibus[6]. It works *without* other modifications to the hardware.

With the use of this new hardware UNIX now can address the 22 bit's of address space needed. By overlaying the UNIX kernel, and for that matter, any user program, it is capable of growing extremely large. The changes to the UNIX kernel to support the ENABLE board have *no* affect on user programs.

The Able ENABLE Board

When a program runs on a unmapped PDP-11, the processor will produce a 16 bit address. With mapping turned on, this 16 bit address is referred to as a **virtual** address. To produce a 22 bit **physical** address there are two relocations that a virtual address must go through. The DEC relocation hardware will produce an 18 bit address. The ENABLE relocation will transform the DEC 18 bit address into a 22 bit physical address.

1. DEC relocation

A 16 bit DEC virtual address (VA), produced by the PDP 11 cpu, builds a 18 bit DEC physical address (PA) as follows:

DEC VA bits 13-15 (three bits) are used to index one of 8 DEC Page Address Registers or PAR's. The lower 12 bits of the indexed DEC PAR are added to the DEC VA bits 6-12 (seven bits) to produce a 12 bit field which goes into bits 6-17 of the DEC PA. Bits 0-5 of the DEC VA are concatenated with the result to produce DEC PA bits 0-5.

The 18 bit DEC PA is now gated to a normal 18 bit Unibus.

2. ENABLE Relocation

The ENABLE relocation process is analogous to the DEC relocation.

The ENABLE PAR's are indexed by bits 13-17 of the DEC PA[5].

4] The name of the macro that we have used in code found in the appendix for this new hybrid is FORTYZ. We will have the standalone system set the global variable "cputype" to 1 greater than the normal. e.g. 41 for an 11/40 class processor and 46 for an 11/45 class processor.

5] 5 index bits means that 32 PAR's can be accessed but only 16 can be used for an 11/40 installation. The other 16 PARs can be used by an 11/45 class processor. For the 11/40 we are using 8 for kernel and 8 for user

The contents of the indexed ENABLE PAR are added to bits 6-12 of the DEC PA to yield a 16 bit field which is placed in bits 6-21 of the ENABLE PA. Bits 0-5 of the DEC PA are again concatenated with the result to produce ENABLE PA bits 0-5. Note that is means that the original DEC VA Bits 0-5 are passed unchanged to ENABLE PA bits 0-5.

This 22 bit address is now gated on to a special 22 bit MODIFIED Unibus. This new Modified Unibus is electrically the same as the one found in an 11/44.

Setting up the Two Memory Management Units.

In order to achieve 22 bit relocation, the DEC Memory Management Unit must be effectively NOP-ed. This is done by placing data into the DEC PARs so that when the DEC 18 bit translation is operating, the 18 bit result will index into the ENABLE PARs *without* adding any other relocation. Because we know that the ENABLE board uses the upper 5 bits of the DEC 18 bit PA as an index, we should only set these bits in the DEC PARs. Contained inside a short routine in mch40.s, we will once and only once, set the contents of the DEC PARs. After that time, our new UNIX will use the ENABLE PAR's like the older versions ***had*** used the DEC PAR's.

3. Setting up DEC Page Address Registers

The DEC Kernel PAR's (DKP) should be initialized as follows:

DKP0 = 0000
DKP1 = 0200
DKP2 = 0400
DKP3 = 0600
DKP4 = 1000
DKP5 = 1200
DKP6 = 1400
DKP7 = 7600

Notice that the lower 7 bits of each number are 0. Bits 7-10 of the DEC PAR is used as an index into the ENABLE PAR's.

To explain why 7600 is placed into DKP7, we must remember that there exists a short period of time when DEC memory management is turned on and ENABLE memory management turn off. During this window, there are references made to the I/O page when we activate the ENABLE Board. Meaning, the DEC memory

---

operations. For an 11/45, we would have 8 for kernel I space, 8 for kernel D space, 8 for user I space and 8 for kernel D space.

management must be set up to handle references to the I/O page with and without the ENABLE board active. DKP7 being 7600 means that when ENABLE memory management is on all references to the I/O page go through ENABLE PAR 31 (DKP7 bits 7-11 = 31). The DEC kernel PAR's index into ENABLE PAR's 0-6 and 31. Note therefore that we are using 7 ENABLE PAR's and then skipping the next 24 ENABLE PAR's.

DEC user PAR's are initialized with 2000 and increment by 200. Therefore, DEC user PAR's index into ENABLE PAR's 8-15.

DUP0 = 2000
DUP1 = 2200
DUP2 = 2400
DUP3 = 2600
DUP4 = 3000
DUP5 = 3200
DUP6 = 3400
DUP7 = 3600

Once the DEC PAR's are initialized it is not necessary to change them[6]. Thereafter when switching user processes only the ENABLE USER PAR's will be modified. Also, A version of UNIX can be used that allows more than 64k bytes of main store to be used by a program while only manipulating the ENABLE Kernel PAR's.

A Minor Mishap with I/O Mapping

The Unibus was designed to support 18 bit DMA I/O devices[2]. Therefore on the 22 bit PDP-11's (11/70 class), an I/O map, called Unibus Map, must also be provided to translate from 18 bit addresses produced by the DMA hardware to the 22 bit addresses that the memory system sees. It should be noted, that all DMA hardware accesses memory at different times and through different drivers than the cpu uses for an instruction or data fetch. Which means on an 11/40 and 11/45 class processor, the I/O subsystem runs unmapped, producing physical address in main memory. Able exploits the fact that both the 11/45 and 11/40 class processors do not need a Unibus Map. Fortunately, DEC did not assign any other I/O devices residing at the I/O map's location for these processors. Allowing our version of UNIX to initialize this map, on our modified machines, in the same manner

6] The only time the kernel must modify these while running normally, is when you wish to support the fuiword subroutines used by the unsupported phyio(2) system call. This is because this subroutine is using the mfpi instruction, which makes some rash assumptions about the state of the DEC memory management registers.

as unmodified UNIX does on the 11/70 class machines.

Unfortunately, it was found that the ENABLE board cannot be running in 22 bit relocation mode with I/O mapping turned off. Before enabling the ENABLE board both the I/O mapping and 22 bit relocation bits in the ENABLE SSR3 must have been set.

UNIX would like to set up the I/O mapping registers *after* the kernel is loaded and running in 22 bit mode. Since both I/O mapping and 22 bit relocation bits must be set at the same time the I/O map is enabled before the map is initialized! UNIX is now running with an undefined I/O map.

These two bits should be independent of each other. The system could then boot and run in 22 bit mode with the I/O map turned off. The initialization code for the I/O map could be left in machdep.c instead of having to move it into the assembler assist that sets up the memory management unit.

Some Results

The authors ran a simple bench mark on four UNIX configurations. These are: an unload 11/70 with 2 megabytes of main memory and 2 RP06s; an unload 11/44 with 1/4 megabytes of main memory and 1 RM02; an 11/34 with 1/4 megabytes of main memory and 2 RL01s; and the same 11/34 with 1/2 a megabytes of main memory and the new ENABLE unit installed. We understand that you must mellow these statistics due to the fact that were running a UCB kernel on the 11/70 and 11/44. We were running a DEC kernel on the 11/34[7]. The actual bench mark was to recompile the entire UNIX kernel from zero. What was typed into UNIX:

```
$ rm dev/*.o sys/*.o
$ time sh -x makeit
```

The file "makeit" contained the lines:

7] The plan was to show you the differences after we had brought up the UCB overlay kernel on the 34. Unfortunately, we were using borrowed memory from Mostek Corporation for the extra 1/4 mega on the 34. We had to give the memory back before we received the overlay code from UCB for the 11/34. We also were using a borrowed 11/34 from Tektronix Computer Automation Support. We had to give this machine back after one week. Once we get some system time again we will running the correct bench mark, and republish these results.

```
cd dev
make
cd ../sys
make
cd ../conf
make unix
```

Some Numbers

CPU Type	Real	User	System
11/70	13:06.0	5:02.9	3:12.3
11/44	23:35:0	7:42:0	6:38:7
11/34	58:23:1	17:12:3	13:08:2
11/34 with ENABLE	42:19:2	14:41:8	10:29:2

The 11/34 should come closer to the 11/44 performance once the UCB changes have been added. Note that the extra memory help the 11/34 because it did not have to swap to the extremely slow RL01s. When the Calgary mods are added the RL's will look much better, because this kernel will be allowed to have 50 or 60 buffers.

For a matter of reference, we used the DEC V7 kernel for the base 11/34 installation. This was because we felt that the people at DEC would have a better idea of how to tune the an unmodified 34 system than we might. It also gave us a reference base. To obtain their code write to:

The DEC UNIX Group
DEC, Continental Blvd
MK1-1/D29
Merrimack, NH 03054

I believe there is a charge for this tape, but check with DEC first.

In the case of the 11/44 and 11/70 we are running a Modified UCB Kernel, from the San Fransico, prelease 2BSD system. We have added a new TTY driver. Hopefully this will come in sync with the UCB one. To obtain this tape, write to:

Bob Kridle
EECS Dept.
Cory Hall
UC Berkeley
Berkeley, CA 94720

Acknowledgments

The authors would like to acknowledge some groups for helping with this effort. Armando Stettner at DEC for the 11/34 system; Mark Horton and Bob Kridle at UCB for help the 11/70 system; Larry Brown and his crew at Tek's Computer Automation Support for the use of an 11/34 and helping us when our hardware died; Mary Driscoll at Mostek Corp, for lending us the 22 Bit memory; and Ken O'Mohundro and Norm Kiefer at Able Corp for the use of a pre-release ENABLE board.

References

1.) Ritchie, D. and Thompson, K., the UNIX Timesharing system., Western Electric Corp.

2.) DEC Processor handbook, Digital Equipment Corp.

3.) UNIX Programmer's Manual, Sixth Edition, Western Electric Corp.

4.) UNIX Programmer's Manual, Seventh Edition, Volume 1 & 2, Western Electric Corp.

5.) Haley, C. and Joy, W., revised by Jolitz, W., Running Large Text Processes on Small UNIX Systems, UC Berkeley Technical Report. Spring 1980

6.) ENABLE user's guide, Able Computer Corp. Pre-release Version, March 1981

Appendix A

Demonstration of Code Expansion - code.c

```
1       /*
2        *      a demonstration of the add problem
3        */
4
5       short   glob_short;     /* This is short in global space */
6       short   short_glob;     /* This is short in global space */
7
8       long    glob_long;      /* This is long in global space */
9       long    long_glob;      /* This is long in global space */
10
11      main ()
12      {
13
14      short   local_short;    /* This is short in local space */
15      short   short_local;    /* This is short in local space */
16
17      long    local_long;     /* This is long in local space */
18      long    long_local;     /* This is long in local space */
19
20      glob_short = 0;
21      short_glob = glob_short + 10;
22
23      local_short = 0;
24      short_local = local_short + 10;
25
26      glob_long = 0;
27      long_glob = glob_long + 10L;
28
29      local_long = 0;
30      long_local = local_long + 10L;
31
32      }
```

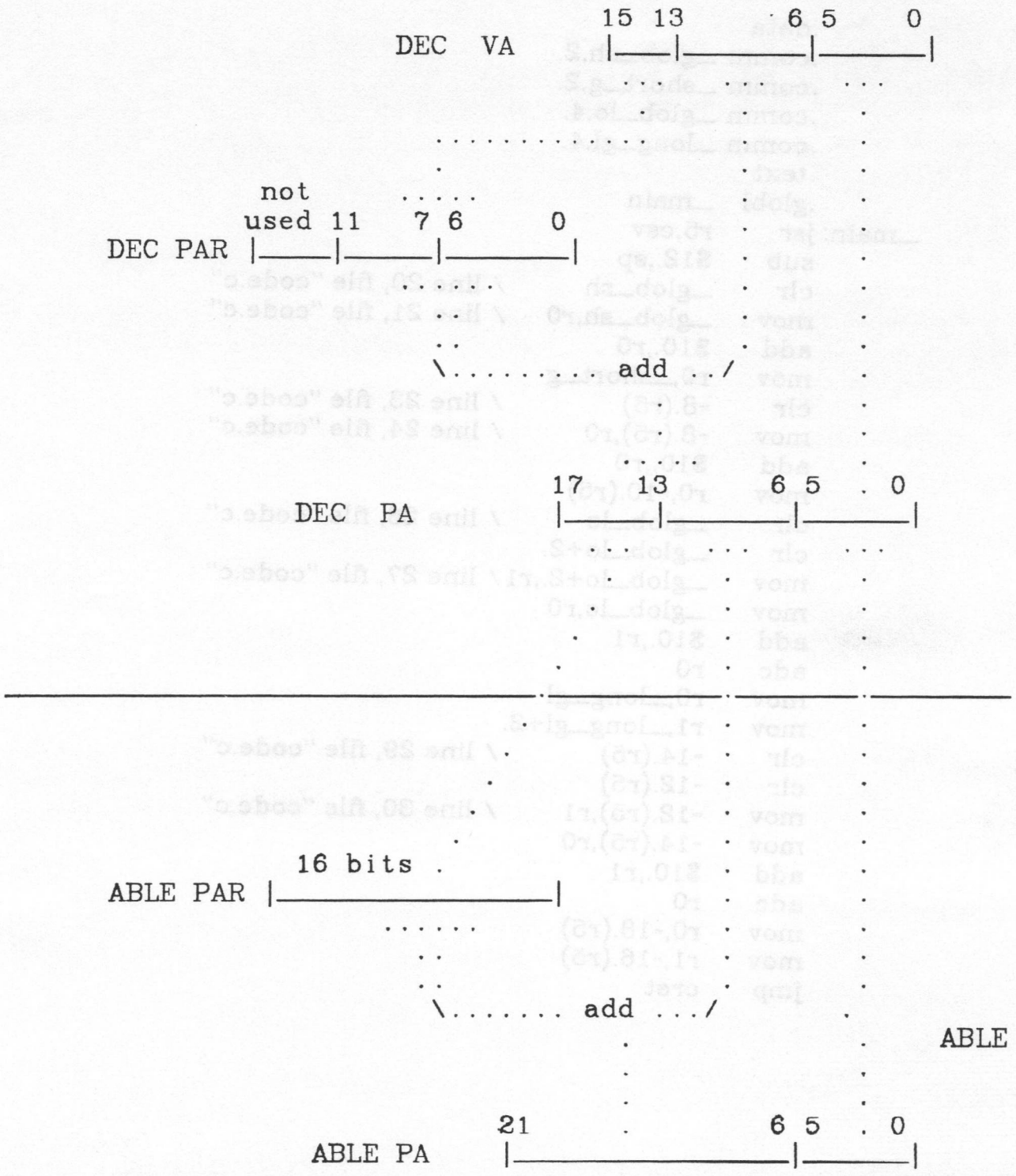
Appendix B

Demonstration of Code Expansion - code.s

```
        .data
        .comm   _glob_sh,2.
        .comm   _short_g,2.
        .comm   _glob_lo,4.
        .comm   _long_gl,4.
        .text
        .globl  _main
_main:  jsr     r5,csv
        sub     $12.,sp
        clr     _glob_sh        / line 20, file "code.c"
        mov     _glob_sh,r0     / line 21, file "code.c"
        add     $10.,r0
        mov     r0,_short_g
        clr     -8.(r5)         / line 23, file "code.c"
        mov     -8.(r5),r0      / line 24, file "code.c"
        add     $10.,r0
        mov     r0,-10.(r5)
        clr     _glob_lo        / line 26, file "code.c"
        clr     _glob_lo+2.
        mov     _glob_lo+2.,r1  / line 27, file "code.c"
        mov     _glob_lo,r0
        add     $10.,r1
        adc     r0
        mov     r0,_long_gl
        mov     r1,_long_gl+2.
        clr     -14.(r5)        / line 29, file "code.c"
        clr     -12.(r5)
        mov     -12.(r5),r1     / line 30, file "code.c"
        mov     -14.(r5),r0
        add     $10.,r1
        adc     r0
        mov     r0,-18.(r5)
        mov     r1,-16.(r5)
        jmp     cret
```

Appendix C

Virtual to Physical Address Mapping



Appendix D

This is a annotated difference listing between the DEC sources and our sources.

In param.h

add the two definitions

```
#define SID     0       /* 0= I space only cpu */
#define FORTYZ      41      /* Able Corp ENABLE Board */
```

In seg.h

change the definition of UISA to

```
#ifdef  FORTYZ
#defineUISA    ((physadr)0163720) /* 1st user I-space reg */
#else
...         (the original UISA definition)
#endif
```

In bio.c

surround the "mapfree" with

```
#if SID || FOURTYZ
        ...
#endif
```

In tm.c

surround the "mapalloc" with

```
#if SID || FOURTYZ
        ...
#endif
```

In rl.c

surround the "mapalloc" with

```
#if SID || FOURTYZ
        ...
#endif
```

In machdep.c

re-define mmr3

```
/*** address of mmr3 to enable unibus map ***/
#ifdef  FORTYZ
#define UBMC ((physadr)0163676)
#else
#define UBMC ((physadr)0172516)
#endif
```

add the following code to initialize the unibus map

```
#if (SID || FORTYZ)
        if (((cputype == 70) || (cputype == 44))
```

```
                || (cputype == FORTYZ)) {
        for(i=0; i<62; i+=2) {
                UBMAP->r[i] = i<<12;
                UBMAP->r[i+1] = 0;
        }
/* Enable the unibus map here instead of in the boot code.
 * This is done to allow a unibus disk to be the
 * system device. (rl02, rm02/3, rk06/7)
 */
        UBMC->r[0] =| 040;
printf("UNIBUS Map enabled0);
}
#endif
```

in the clkstart routine add the comments

```
        lks = CLOCK1;
/*
** We can not do a fuiword with the ENABLE Board
**  therefore, we assume a line clock and not
**  the programmable clock. this is a hack.
**  fuiword should be fixed.
**
**      if(fuiword((caddr_t)lks) == -1) {
**              lks = CLOCK2;
**              if(fuiword((caddr_t)lks) == -1)
**                      panic("no clock");
**      }
*/
        lks->r[0] = 0115;
```

make the mapalloc and mapfree routines conditional

```
#if (SID || FORTYZ)

        /*
         * 11/70 routine to allocate the
         * UNIBUS map and initialize for
         * a unibus device.
         * The code here assumes that an rh on an 11/70
         * is an rh70 and contains 22 bit addressing.
         */
        int     maplock;

        mapalloc(bp)
        register struct buf *bp;
        {
                register i, a;
```

```
                        if (((cputype != 70) && (cputype != 44))
                                        && (cputype != FORTYZ))
                                return;
                        spl6();
                        while(maplock&B_BUSY) {
                                maplock |= B_WANTED;
                                sleep((caddr_t)&maplock, PSWP+1);
                        }
                        maplock |= B_BUSY;
                        spl0();
                        bp->b_flags |= B_MAP;
                        a = bp->b_xmem;
                        for(i=16; i<32; i+=2)
                                UBMAP->r[i+1] = a;
                        for(a++; i<48; i+=2)
                                UBMAP->r[i+1] = a;
                        bp->b_xmem = 1;
                }

                mapfree(bp)
                struct buf *bp;
                {

                        bp->b_flags &= ~B_MAP;
                        if(maplock&B_WANTED)
                                wakeup((caddr_t)&maplock);
                        maplock = 0;
                }
        #else
                mapalloc(bp)
                struct buf *bp;
                {
                        panic("We called mapalloc");
                }
                mapfree(bp)
                struct buf *bp;
                {
                        panic("We called mapfree");
                }
        #endif
```

In mch.s

add to the front of the file the definition

```
FORTYZ = 41.
```

after the line

```
        mov     $IO,(r0)+
```

in the user segment initialization add the following

```
        .if FORTYZ
                mov     $IO,(r0)+       / we use TWO copies
                mov     $KISA7,r0
                mov     $IO,(r0)        / turn on ENABLE's IO page
        .endif
                mov     $77406,(r1)+    / rw 4k

        .if FORTYZ
        / these are really the DEC Memory Management Registers
                mov     $2,r3
                mov     $PAR1,r0        / enable DEC User PARs
                mov     $2000,r1
                mov     $8.,r2          / do for 8 EPAR's
                br      1f
        2:      mov     $PAR2,r0        / enable DEC Kernel PARs
                clr     r1              / EPAR's increment
                mov     $7,r2

        1:      mov     r1,(r0)+        / set up the PAR
            add     $200,r1                 / increment to next page
                sob     r2,1b           /any more registers to do
                sob     r3,2b           / do the Kernel space reg's
                mov     $IO1,(r0)       / DEC KA7 sets up the IO page
        .endif
```

after the lines

```
        / get a sp and start segmentation

        mov     $_u+[usize*64.],sp
        inc     SSR0                    / start DEC Memory Manager
```

add the following

```
        .if FORTYZ
        / KLUDGE!!!!    KLUDGE!!!       KLUDGE!!!
                bis     $IO22,SSR3      / start relocation to 22 bits
                                        / AND turn IO MAP
        / KLUDGE!!!!    KLUDGE!!!       KLUDGE!!!
                bis     $BIT0,SSR4      / enable the ENABLE board
        .endif
```

at the end of the file add the following definitions

```
        .if FORTYZ
        SSR3    = 163676                / addr of ENABLE's SSR3
        SSR4    = 163674                / addr of ENABLE's SSR4
        KISA0   = 163700
        KISA6   = 163714
        KISA7   = 163776
```

```
        UISA0  = 163720
        UISA1  = 163722
        PAR1   = 177640        / the DEC user space PAR
        PAR2   = 172340        / the DEC kernel space PAR
        BIT0   = 000001
        BIT4   = 000020
               / This is sort of a crock but fixes a bug
               / in the ENABLE board.  BOTH must be on for
               / it to work.  ctc, sjh  03/20/81
               /
               / Turn on IO Mapping & 22 Bit Addressing
        IO22   = 000060
        IO     = 177600   / ENABLE IO Register (16 bits wide)
        IO1    = 7600    / DEC IO Register (12 bits wide)
        __cputype: FORTYZ
        .endif
```

and comment out the conflicting definitions as in

```
        / .if !FORTYZ
        / KISA0        = 172340
        / KISA6        = 172354
        / KISA7        = 172356
        / UISA0        = 177640
        / UISA1        = 177642
        / IO    = 7600
        / __cputype: 40.
        / .endif
```