PROTECTED MODE AND THE CLARION DOS EXTENDER

BOB COX

PRODUCT DEVELOPER
CLARION SOFTWARE

Any problems with 386MAX or NetRoom

XBCAST.EXE + Minimize SmartDrive

— Max " "

or other Windows aware

Computer Memory And How It Works

* TRANSISTORS, FLIP-FLOPS AND BITS
  Transistors are used to make devices called flip-flops. Flip-flops come in different flavors, one of which is "dynamic", and is used on most PCs. Flip-flops are used as toggle switches (on/off) called "Bits" in computer memory devices known as Dynamic Random Access Memory

* BINARY NUMBERS
  By grouping bits together, we can represent numbers which inturn represent images, sounds, letters, etc. Using bits, the largest representable number is:

  => 2^bits (2 to the power of the number of bits used).

* CLOCK CYCLES
  All the bits in computer memory receive an electronic signal at precise intervals called the "clock cycle". This determines the rate at which the values in the bits can be changed.

* MEMORY AND BYTES
  Groups of 8 bits are called "Bytes". Computer memory consists of a long stream of bytes which may be referred to by a number called an "Address".

* THE CPU
  The Central Processing Unit is a chip which accepts "Instructions" and then directs, other devices according to the instructions .

* INSTRUCTIONS
  Instructions, sometimes called "opcodes" (operation codes), are numbers which the CPU uses to determine its next action.

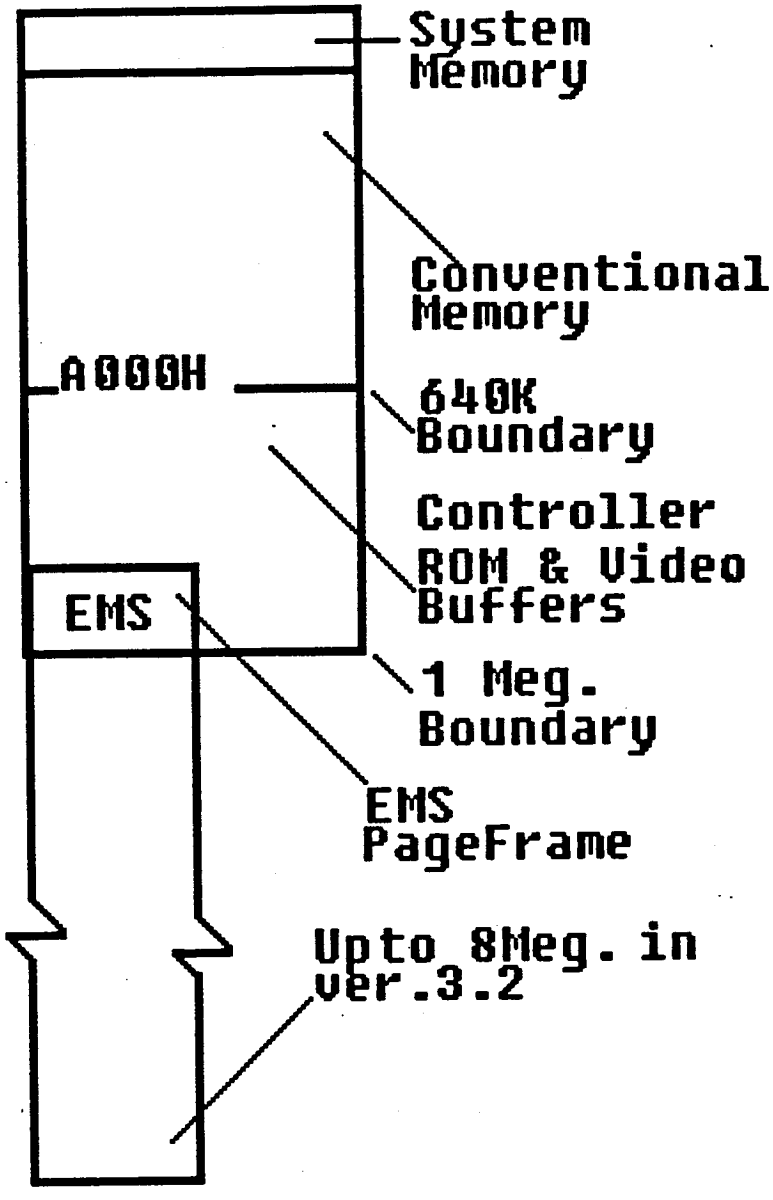
* REGISTERS
  Registers are a fast kind of memory inside the CPU. Some of the registers in the CPU can be controled by using CPU instructions.

* THE BUS, ADDRESSES
  The "Bus" refers to a set of wires through which numbers are passed around. Each line (wire) in the bus acts like a bit. The "Address Bus" is used by the CPU to refer to a particular byte of memory. The "Data Bus" is then used to transfer the value of the selected byte between memory and the CPU. The "Internal Bus" is used inside the CPU during execution of instructions.

* HW-INTERRUPTS
  "Hardware Interrupts" are signals that other devices send to the CPU requesting immediate attention. eg: When a key is pressed, the CPU must leave the application instructions, and process the systems keyboard instructions before returning to the application.

* PORTS
  Ports are connections that allow the CPU direct control over other connected devices.

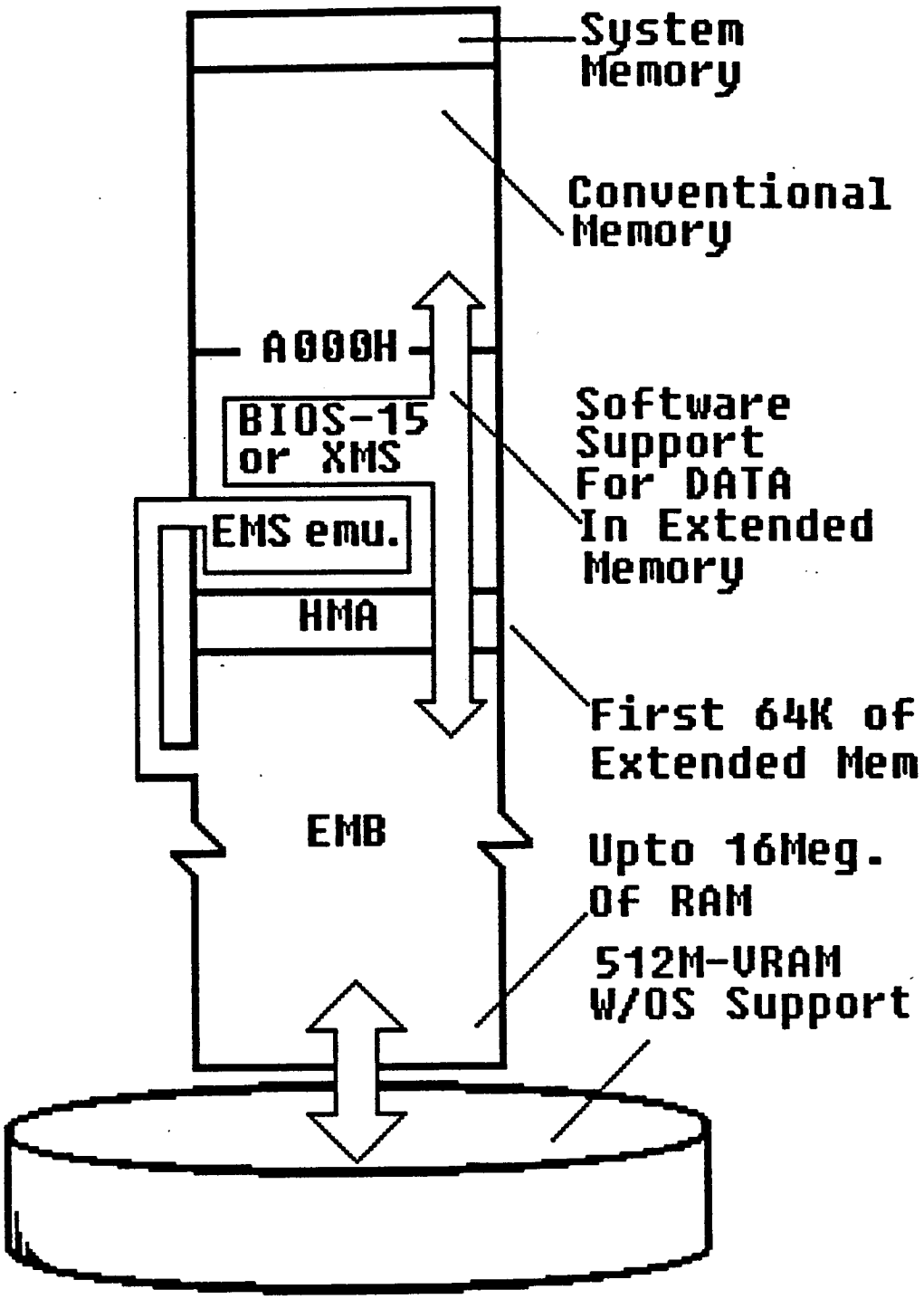
An 8088 PC Memory Map



The PC Memory Usage Map

Memory Address	IBM Defined Useage	Memory Range	386 Redefined Useage
000000-007FFF	Interrupts,BIOS Data,DOS Areas		
008000-09FFFF	Application Memory (640K)		
0A0000-0AFFFF	EGA/VGA Graphics Buffers		
0B0000-0B7FFF	Monochrome Text Buffers		
		0B0000-0B0040	Used (PSCNTL)
		0B0050-0B0F10	Used (LSL)
		0B0F20-0B1E30	Used (NE1000)
		0B1E40-0B39E0	Used (IPXODI)
		0B39F0-0B3B30	Used (PSCNTL)
		0B3B40-0B3B80	Used (LAEXIT)
		0B3B90-0B66A0	Used (CMOUSE)
		0B66B0-0B69B0	Used (PUSHDIR)
		0B69C0-0B6B30	Used (LAEXIT)
		0B6B40-0B7FE0	Available
0B8000-0B8FFF	Color Text Buffers		
0C0000-0C7FFF	EGA/VGA Video ROM		
0C8000-0CFFFF	Disk Controller & Adapter ROM		
		0C8000-0C9060	Used (ANSI)
		0C9070-0C90B0	Used (KB)
		0C90C0-0CA750	Used (PRINT)
		0CA760-0CB120	Used (KB)
		0CB130-0CB170	Available
		0CB180-0D1EC0	Used (HYPER386)
0D0000-0D7FFF	Reserved		
		0D1ED0-0DFAA0	Used (NETX)
0D8000-0DFFFF	Reserved		
		0DFAB0-0DFFE0	Available
0E0000-0E7FFF	Cassette ROM		
0E8000-0EFFFF	Cassette BASIC ROM		
		0E0000-0EFFFF	EMS Page Frame
0F0000-0F7FFF	Reserved		
		0F4000-0F6160	Used (ANARKEY)
		0F6170-0F7FF0	Available
0F8000-0FFFFF	System ROM BIOS		
100000-10FFFF	High Memory Area (HMA)		MS-DOS v5.0+
110000-??????	Application Memory		Cache,V86-Mach.
:	:		
(Extended Memory Blocks)			
:	:		
:16M on 286 or 386sx	:		
	:4G on 386 or 486		

An 80286 PC Memory Map



* REAL MODE.

In real mode addresses are computed as the segment register shifted left by 4 bits and then added to the value in the offset register yielding a 20 (16+4) bit address and an effective address range of 1M.

```
Segment (16bits)              = FFE9
Offset  (16bits)              =  016F

Absolute Address (20 bits)    = FFFFF (Last addressable byte)
```

* 16 BIT (286) PROTECTED MODE.

Segment registers are each still 16 bits long, but are used as an index into a LOCAL, and GLOBAL DESCRIPTOR TABLE (LDT & GDT). The tables contain Segment Descriptors each of which is 8 bytes long and contains the following fields:

```
Length              (16 bits)
Base Address        (24 bits)
Present bit
Privilege Level
System Seg. or Memory Seg. flag
Read/Write/Execute access rights.
```

Because of 16 bit index registers, the largest possible Descriptor Table is 64K (2^16), but the 24 bit Base Address can represent any physical address.

* MAXIMUM NUMBER OF SEGMENT DESCRIPTORS PER TABLE

=> 64K table / 8 byte descriptors = 8192 descriptors/table

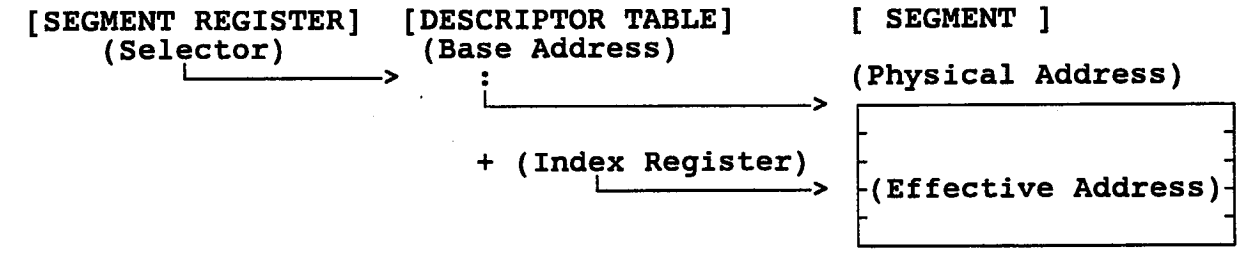
* TOTAL PHYSICAL ADDRESS SPACE

=> 2^24 == 16M.

And because segments may each be up to 64K bytes long and may be managed on external storage by the OS as virtual memory. We have:

* VIRTUAL ADDRESS SPACE

=> 8192 descriptors * 2 tables * 64K/seg. = 1G



* BIOS-15/"VDISK METHOD"

  The "VDISK Method" of accessing the EMB was used by programs before any standard was established for multiple requestors of EMB. Compounding the awkward scheme, 286 machines required special handling from the BIOS after halting the CPU to switch back to real mode.

* LIM-EMS V3.2

  Lotus/Intel/MicroSoft EXPANDED Memory Specification (EMS) v3.2 was introduced to provide a protocol for programs accessing bank switched memory on the PC. It defines a 64K area of memory called the PageFrame, to which a third party memory adapter can be mapped. The PageFrame is used as a window into a larger pool of memory on the card. Up to 8M could be supported on the adapter. The Spec. was later revised for v.4.0 to include support for up to 32M and virtual memory.

* WINDOWS-286 (V2.1)

  Windows 286 still used only EMS, but on a 286 even in real mode, it is possible to generate 21 bit addresses by enabling the chips A20 line. This extra bit gives addressability to 1M+64K. The additional 64K is called the HIGH MEMORY AREA (HMA)

* XMS

  The Lotus/AST/MicroSoft Extended Memory Specification (XMS) defines a protocol for access to the EMB and the HMA. It also provides an allocation management interface and a hardware independent method of managing the HMA. Even with XMS real mode programs can only use the EMB for data storage, and there is no virtual support, only the physical memory can be used.

* VCPI

  The Virtual Control Program Interface (VCPI) defines a memory management protocol for programs running in protected mode. That is; VCPI is a DOS extender support platform. Multiple programs can now use the EMB for both CODE and DATA under DOS!

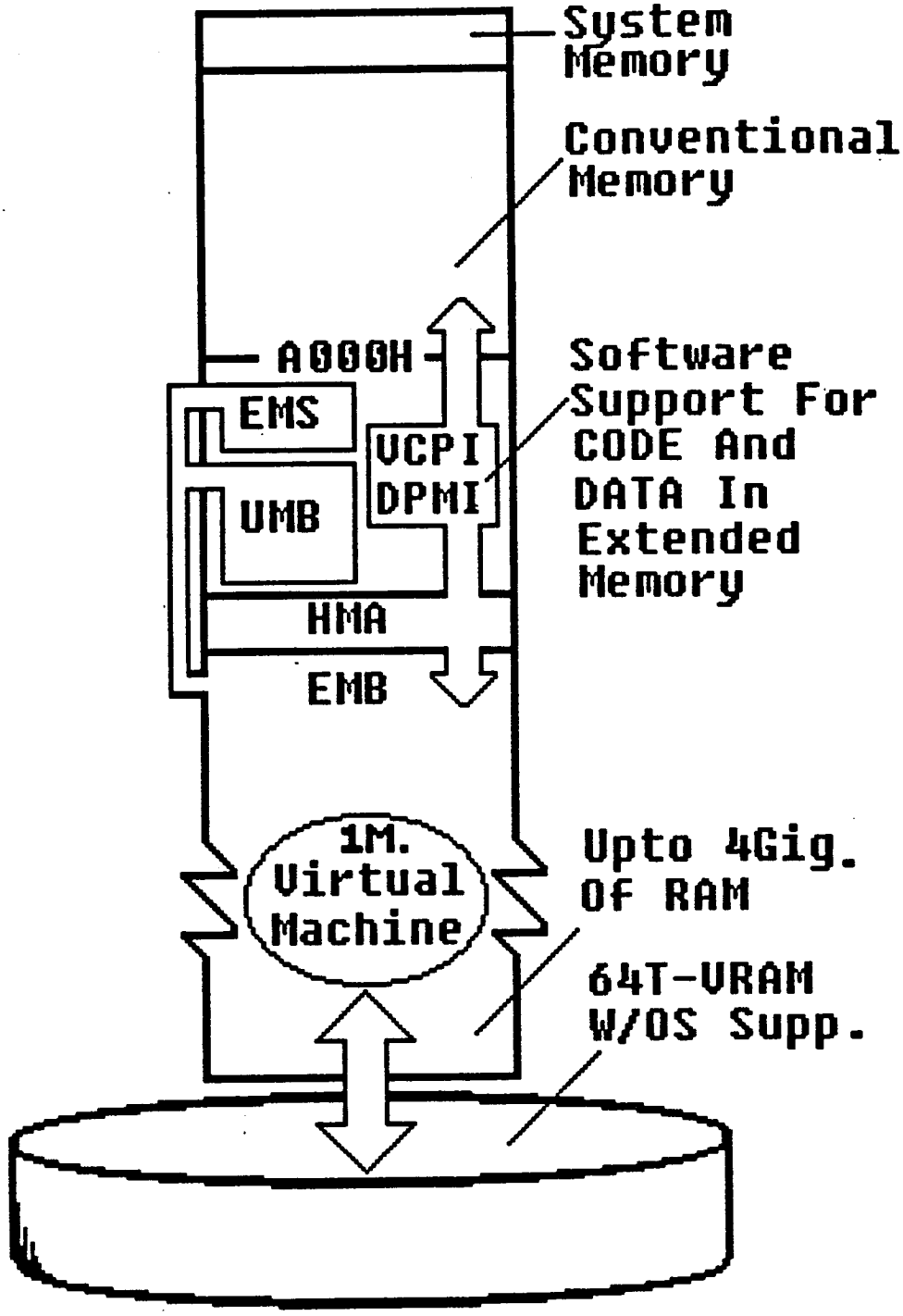
* 286DOS-EXTENDERS

  286DOS-extenders are the final link needed to complete the bridge to protected mode for DOS apps. Programs can now use huge amounts of memory while remaining compatible with DOS.

* DPMI

  The DPMI protocol, has a similar objective to that of VCPI, but the implementation is different. DPMI runs at a higher privilege level than "client" programs, and allows implementation on an 80286 processor.

A 386/486 PC Memory Map



Local Descriptor Table
GLOBAL

* 32 BIT (386) PROTECTED MODE. "THE SEGMENTED MODEL"

32 bit protected mode is very similar to 16 bit protected mode when using the Segmented memory mode of the 386. Address translation is still done through an LDT and GDT, using 8 byte selectors, and a maximum of 8,192 selectors per table maintaining 80286 compatibility.

There are however, several important extensions. A 386 segment descriptor contains the following changed or new fields:

Length	(20 bits)
Base Address	(32 bits)
Granularity	
Operand size flag	
System defined	

Also, because the index registers are 32 bits, the native segment size is 4G (2^32), the address bus is also 32 bits, so a single index register can be used to access the entire physical address space.

* PHYSICAL ADDRESS SPACE

=> 2^32 = 4G

* VIRTUAL ADDRESS SPACE

=> 2 tables * 8192 selectors * 4G segs = 64T

* THE "FLAT" MODEL

32bit addresses in the offset refer to physical memory.

* THE "PAGED" MODE

In this mode the processor uses a two level hierarchy of indices to locate 4K sized blocks of physical memory. This mode is very efficient for high performance virtual memory management.

* THE "VIRTUAL 8086 MACHINE" MODE

In V86 mode a 386, generates addresses by manipulating pointer segments and offsets exactly as it does in real mode. The result however, is a 21 bit and not a 20 bit address. This gives access to the HMA when the A20 line is active, maintaining compatibility.

More importantly, when an operating system combines this mode with the Paging mode, multiple Real mode applications can be executed in a virtualized and partially protected environment. An entire megabyte of physical memory must be reserved for use by the virtual machine, and only 8086/8088 instructions are legal in V86 mode.

* WINDOWS 386 (ENHANCED)
  The Windows Enhanced mode also allows Windows programs to execute in protected mode, but development overhead for creating Windows programs can be prohibitive.

* WINDOWS 386 (V86)
  The Virtual-86 support features of Windows v.3.0 are exciting for users who can now run many DOS programs "simultaneously". For developers however, this does not represent any advancement because the DOS programs still can not address the EMB directly but only as bank switched memory.

* OS/2 1.1
  OS/2 v1.1 was a very convincing promise of a platform to harness the power of protected mode. OS/2 had a hard sell however, because the original versions were for the 16 bit 80286, and there were upgrade, and compatibility problems, and a lack of application support.

* QEMM/386MAX
  QEMM and 386Max were among the first memory managers to use the mapping features of the 80386 to extend the "Conventional Memory" area. This was done by remapping some of the EMB memory so that some address ranges between the A0000-FFFFF could be used as normal RAM for TSRs and DOS device drivers. These regions later came to be known as the UPPER MEMORY BLOCKS (UMB).

* 386DOS-EXTENDERS
  386 DOS extenders take advantage of additional support and memory from the superior 80386 and 80486 processors. They are more of a hybrid of extenders and tend to be used more in specialized applications because of incompatibility with the 286.

* PC-DOS V.5.0
  In version 5.0 DOS included direct support for nearly all the extensions discussed above, stopping short of protected mode operation itself. In fact DOS 5.0 executes from the HMA leaving almost the entire 640K user area free, it also provides for loading device drivers and TSRs into the UMB. While XMS and HMA support are in the HIMEM.SYS driver, EMM386.SYS provides VCPI, EMS, and UMB support. These drivers herald a reign of extender programs under DOS. Finally, UMBs can be linked directly into the DOS memory chain giving existing DOS applications access to discontiguous memory in the UMB.

* OS/2 V.2.0
  OS/2 now uses the 32 bit registers of the 386, Paged Virtual Memory management, and hardware supported V86 machines to provide the DOS box. Furthermore, OS/2 incorporates a high performance file system, graphical user interface, complete hardware insulation, multitasking, networking and many more features. OS-2 excels in harnessing 386 capability.

Summary Of A Brief History

Year	Chip	Bus Size Address	Internal Data	Address Space Physical	Virtual	Software	Platform	Application Memory Access
72	8008	16	8	8	64K -	CPM	8008	48K
76	8080	16	8	8	64K -	:	8080	48K
76	8085	16	8	8	64K -	:	8085	48K
78	8086	20	16	16	1M -	:	8086	48K
79	8088	20	16	8	1M -	:	8088	48K
80	80186	20	16	16	1M -	xNIX	80186	400K
81	80188	20	16	8	1M -	:	80188	400K
81						PC-DOS	8088	640K
82	80286	24	16	16	16M 1G	:	8088+	640K
						&BIOS-15/"VDISK Method"	x286	640K|EMB<=Data
85						&LIM-EMS v3.2	8088+	640K|EMS<=Data*n
85	80386	32	32	32	4G 64T	:	8088+	640K|EMx
86	80386sx	24	32	16	16M 1G	:	8088+	640K|EMx
87						&Windows-286(v2.1)	x286+	640K|EMS+HMA
88	80486	32	32	32	4G 64T	:	8088+	640K|EMB
						&XMS	x286+	640K|EMB+HMA*n
						&286DOS-extenders	x286+	640K+EMB<=Code/Data
						&VCPI	x386+	640K+EMB*n
						&DPMI	x286+	640K+EMB*n
						&Windows 386 (Enhanced)	x386+	640K+EMB*n
						&Windows 386 (V86)	x386+	640K|EMB*n
						OS/2 1.1	x286+	512M*n
89						QEMM/386Max	x386+	640K|EMB+HMA+UMB
						PC-DOS +386DOS-Extender	x386+	640K+EMB
	80486sx	32	32	32	4G 64T	:		
						PC-DOS v.5.0	x386+	640K|EMB+HMA+UMB*n
91						OS/2 v.2.0	x386+	32T*n

* **SYMBOL LEGEND**
  - (:) Using the last stated operating system.
  - (&) In addition to last stated operating system.
  - (\|EM?) App. can store Data in EMS or EMB (EMx means either).
  - (+EMB) App. stores most Code and Data in EMB
  - (*n) Permits concurrent memory access by multiple programs.
  - (a<=b) Permits storage of 'b' in 'a' memory.

* **EMB**
  The "Extended Memory Blocks" is the memory on 286+ machines, whose addresses lie beyond the 1M boundary. Technically, from 110000H (1M+64K) and above. This memory can only be accessed in protected mode.

* **UMB**
  The "Upper Memory Blocks are unused sections of the reserved memory addresses between the 640K and the 1M boundary of the PC memory.

* **HMA**
  The "High Memory Area", only accessible on 286+, is the 64K section of memory immediately following the 1M boundary.

* REQUIREMENTS AND COMPATIBILITY

Runs on 80286 (w/BIOS 15H support), and all 386sx or better
Compatible with DPMI, VCPI, XMS(A20), and Hardware access.
Supports direct Memory Mapped hardware and Port access.
Works with 1-16 Megabytes of RAM
Some free disk for swapfile (size app. dependent)
OS/2 Compatible DLL/EXE formats and API calls
Supports FIXED, NONCONFORMING, and PRELOAD segment attributes

* TRANSPARENT ACCESS

Maximum physical capacity for code and data 16M
Maximum virtual capacity for code and data 512M
Full featured debugging
Exceptions produce a processor trace file (XTRACE.TXT)
Supports Dynamic Link Libraries or stand alone EXE.
BIOS functions:(10=Video,16=Keyboard,33=Mouse) in Clarion
Direct Support for all DOS (Int 21H) functions except:
(Old FCB I/O, TSRs, MS NetWork, Direct Device I/O)
Supports Huge pointers in C, C++

* DIRECT API ACCESS

Supplementary tracing functions
Out of memory recovery intercept hooks
Out of disk recovery intercept hooks
Timer tick intercept hooks
Termination handler intercept hooks
Control-C/Break intercept hooks
Print Screen intercept hooks
Support for Huge pointers in any language
Support for Segment Aliasing
Program controled loading and unloading of DLLs
Allocation for extended, conventional, and fixed segments
Calling Real mode procedures
Querry/Update segment attributes
Protected and Real mode address translation.
Direct Support for BIOS SW-Interrupt functions:

10	Video Control
11	Equipment Check
12	Base Memory Size
13	Disk Control
14	Serial Communications
16	Keyboard Control
17	Printer Control
1A	Real Time, and Clock Timer Control
20	DOS Terminate Process
33	Mouse Interface.

* SEGMENT TRANSLATION

  Extender acts as a translator between your application and the parts of the system that can not run in protected mode, namely:

  The BIOS
  The DOS operating system.
  Any Device drivers and TSR programs loaded.

  The extender must also intercept and translate hardware interrupts which transfer control between your protected mode program and the real mode system programs. The performance penalty for switching between real and protected is exagerated when the application is DOS or BIOS intensive. Thus, a disk intensive Clarion program, for example, will be more efficient run in Real mode if possible.

* MEMORY COMPACTION

  The extender does memory compaction to increase the number of total allocations permissible in an application beyond the 8,192 limit. The Clarion extender will handle in excess of 65,000 allocations, but this also produces the possibility of memory violations escaping the hardware detection.

* STACK OVERFLOW, STACK OVERFLOW, STACK OVERFLOW, ...

  When stack overflow exceptions occur, they will probably hang, or reboot the machine because the return address of the interrupt is pushed onto the stack by the processor, causing a double exception infinitely. This condition can sometimes be avoided by using runtime stack checking in the application.

* FLUSHING MEMORY IN A RUN

  The extender will allocate blocks of extended memory, incrementally (on demand), in 1M chunks if possible. When a child program is RUN from a Clarion application the extended memory is not automatically released. This is done for speed sake, and because most child programs would not use the memory. The Clarion program can however be made to flush all unnecessary allocations before a RUN by using the command switch:

  'CLARUNFLUSH=ON'

* THE EXTENDER BROADCAST FACILITY

  Many Windows-aware programs that use the EMB can be configured to release allocated but unused, memory for Windows when it loads. For example the CONFIG.SYS command :

  DEVICE=SMARTDRV.SYS 1024 256

  SMARTDrive will use 1M of the EMB for caching when Windows is not loaded, but only 256K when Windows is loaded. This effect can also be achieved for Clarion extended programs by running the XBCAST.EXE program which ships with Clarion 3003+. XBCAST tells a Windows-aware programs to contract with the command "XBCAST +", and to expand with the command "XBCAST -".

* CLARION
  PEEK and POKE
  ADDRESS(seg,off)
  Third party libraries

* C/C++
  Unauthorized reading or writing of memory
  Uninitialized pointers
  Accessing memory mapped hardware
  Segment arithmetic

* MODULA-2,PASCAL
  Evaluation order (Pascal)
  Unauthorized reading or writing of memory
  Accessing memory mapped hardware
  Huge Pointers and segment arithmetic

* ASSEMBLY
  Unauthorized reading or writing of memory
  Use of segment registers as working registers
  Accessing memory mapped hardware
  Huge Pointers and segment arithmetic
  Interrupt handlers (run in real and protected mode)
  Record buffers in the EMB

* INTERESTING ERRORS

  - EXPANDED MEMORY MANAGER INSTALLED

    A non VCPI or DPMI compatible extended memory manager is installed and is running DOS in a virtual 8086 machine.

  - PROTECTED MODE CANNOT BE ENTERED

    This is caused when no VCPI or DPMI support is available, or the A20 line does not use a standard configuration. If this happens when you are not using a memory manager, try loading HIMEM.SYS, or some other extended memory manager.

  - OUT OF MEMORY

    There is insufficient conventional memory to satisfy a conventional memory allocation request, or insufficient memory to satisfy an extended memory allocation request when disk swapping has been disabled.

  - UNSUPPORTED DOS FUNCTION CALLED.

    Some third party library is calling DOS with an unsupported function code. Or the program is running amuck, amuck, amuck.

```
USING QEMM386 MEMORY MANAGER                                  COMMENTS
=====CONFIG.SYS=================================================================
DEVICE = C:\QEMM\QEMM386.SYS INCLUDE=B000-B7FF RAM  ;No Stealth, Use Graphics
DEVICE = C:\QEMM\LOADHI.SYS /R:2 C:\DOS\ANSI.SYS
BUFFERS= 20
DOS    = HIGH                                       ;No UMB links
STACKS = 9,256                                      ;Do Not Use STACKS=0,0

=====AUTOEXEC.BAT===============================================================
c:\qemm\loadhi /r:2 c:\dos\print /D:PRN /Q:10
c:\qemm\loadhi /r:1 c:\cmd\lsl
c:\qemm\loadhi /r:3 c:\cmd\ne1000
c:\qemm\loadhi /r:3 c:\cmd\ipxodi A
       :

USING EMM386 MEMORY MANAGER                                   COMMENTS
=====CONFIG.SYS=================================================================
DEVICE      = C:\WINDOWS\HIMEM.SYS                  ;XMS,HMA Support
REM DEVICEHIGH = C:\WINDOWS\EMM386.EXE RAM          ;EMS Emulation+UMB Support
DEVICEHIGH = C:\WINDOWS\EMM386.EXE NOEMS            ;UMB Support Only
DEVICEHIGH = C:\WINDOWS\SMARTDRV.SYS 128 /A         ;Disk Caching
DEVICEHIGH = C:\WINDOWS\MOUSE.SYS /Y
DEVICEHIGH = C:\DOS\ANSI.SYS
FILES       = 60
BUFFERS     = 20
DOS         = HIGH,UMB                              ;With UMB links.
STACKS      = 9,256                                 ;Do Not Use STACKS=0,0

=====AUTOEXEC.BAT===============================================================
LOADHIGH C:\DOS\PRINT /D:PRN /Q:10
LOADHIGH C:\CMD\LSL
LOADHIGH C:\CMD\NE1000
LOADHIGH C:\CMD\IPXODI A
LOADHIGH C:\CMD\NETX  /PS=DEV2 /C=C:\CMD\NET.CFG
       :
```

Better for 3005

* THE DOS 'MEM' COMMAND

The DOS MEM command with the /C switch and "DOS=HIGH,UMB" set in the CONFIG.SYS file, will display free blocks in the UMB

```
Conventional Memory :
  Name                Size in Decimal       Size in Hex
-------------      ---------------------   -------------
   :                      :             :          :

Upper Memory :
  Name                Size in Decimal       Size in Hex
-------------      ---------------------   -------------
   :                      :             :          :

Total bytes available to programs (Conventional+Upper) :646368   (631.2K)
Largest executable program size :                        631632   (616.8K)
        :
```

*** EXAMPLE OF 'ADDRESS()' KEEPS DISKETTE MOTOR RUNNING**

```
    program                                         !Demonstrates ADDRESS
runval  byte(91)                                    !91 = 5sec. of timer ticks
runflag byte(0)                                     !holds flag bits for motor status
    code; i# = status('A:')                         !starts up diskette motor
    i# = 32000                                      !initialize test interval
    loop until keyboard()                           !include error processing and wait
     poke(address(40H,40H),runval)                  !hold motor count down high
     if i# = 0 then                                 !if interval elapsed, test motor
        peek(address(40H,3FH),runflag)              !test status of drive A:
        runflag = BAND(runflag,1)                   !mask all bits except bit 1
        if runflag then                             !if bit is set, motor is running
            type('Motor Running.' & @lf)            !make a note to user
        else
            type('Problem: Motor Not Running' & @lf)    !Problems ...
        end!if
        i# = 32000                                  !reset test interval counter
     else
        i# -= 1                                     !decrement test interval counter
     end!if
    end!loop
    runval = 20                                     !set motor countdown to 1 timetick
    poke(address(40H,40H),runval)                   !latch the value to BIOS memory
    type('Motor Shutting Down' & @lf)               !tell user to expect shutdown
```

*** EXAMPLE OF EXCEPTION TRACING**

```
    program
    map;mopmess;someproc
     module('extender')
        PrChar(byte),name('PrChar')                 !print 1 character
        PrStr(*cstring),raw,name('PrStr')           !print cstring
        FatalError(*cstring),raw,name('FatalError')!immediate terminate w/msg.
        PrSp,name('PrSp')                           !print 1 space
        PrSps(ushort),name('PrSps')                 !print some spaces
        PrWord(ushort),name('PrWord')               !print value of ushort
        PrByte(byte),name('PrByte')                 !print value of byte
        PrLn,name('PrLn')                           !next print on a new line
     end!extender
    end!map
myrec   string(100)
val     long(1)
    code;someproc;return                            !call procedure and exit prg.
someproc procedure;code
    shutdown(mopmess)                               !arm the shutdown trace func
    myrec = 'Some Stuff Happens Here' & @lf         !execute procedure
    poke(0,val)                                     !Naughty naughty... XTRACE.TXT
    myrec = 'This Stuff Will Never Happen' & @lf!other program code...
    shutdown()                                      !dissarm shutdown trace func
mopmess procedure
tstr    cstring(255)                                !temporary cstring
    code; tstr = 'MYREC='& myrec                    !assign output to temp cstring
    PrStr(tstr)                                     !dump info into XTRACE.TXT
```