INTERNAL CORRESPONDENCE

TO: LIST
FROM: [redacted] x77026, SC4-59
SUBJECT: 80386 Architecture Change- Virtual I/O Bit Map
LIST:

DATE: 1/20/86

Summary

This memo ammends the previous virtual I/O definition by incorporating the bitmap into Protected Mode I/O as well as Virtual 86 mode I/O. The changes outlined in this memo change only the handling of I/O instructions. In particular, the PUSHF, POPF, IRET, INT n, CLI, and STI instructions are unchanged from the REV 1.8 EAS.

The flags instructions were investigated for possible virtualization of the IF (interrupt enable flag) in protected mode, as was done in Virtual 86 mode. This investigation led to the conclusion that it was not possible to extend IF "virtualization" to protected mode, due primarily to the problems with the POPF and IRET instructions.

The changes in this memo are intended to better support execution of existing "dirty" PC programs in Virtual 86 mode, and also to pass along the most critical parts of the added support to 286 and 386 protected applications to allow the OS to provide direct access to selected I/O devices to protected mode applications.

The Virtual 86 mode support provides selected access to I/O ports to programs executing in Virtual 86 mode, and also provides either direct access to the IF (Interrupt Disable Flag), or provides full virtualization of the IF flag. Direct access to IF is provided to programs executing in Virtual 86 mode if IOPL=3. All other values for IOPL provide full virtualization of IF by trapping all instructions that load or store IF in Virtual 86 mode. If an application is granted direct access to IF, a prudent system design would include a watchdog timer wired to NMI to regain control from an application that left interrupts disabled for longer than a specified time.

The protected mode support involves only the restricted access to I/O ports, but makes no changes to the instructions which load or store IF in Protected mode. This allows the OS to provide selective access to I/O ports to an application, but does not allow this along with direct access to IF. Because the 286 did not trap on POPF or IRET if CPL > IOPL, it is not possible to "virtualize" IF in protected mode. Instead, IF control can be supported through direct OS calls, or by traps on CLI and STI instructions.

1. I/O Instructions

Access to I/O devices is provided by special instructions which provide for single transfers through the EAX register, or for a string transfer to/from memory. These instructions will trap if an attempt is made to access an I/O port address that is not accessible to the processor at the current privilege level. Two mechanisms provide protection to the I/O address space: the IOPL field in the flags word, and a I/O permission bitmap in a 386 format TSS. These mechanisms only operate in protected mode and Virtual 86 mode, they do not operate in REAL mode. In REAL mode, there is no protection of the I/O space, and any I/O port can be addressed by the I/O instructions.

The IOPL field defines the minimum privilege level required for access to the I/O instructions in Protected mode. It has no effect on access to the I/O instructions in Virtual 86 mode. In Protected mode, an I/O instruction can access any I/O address if CPL <= IOPL, that is, if the current privilege level is the same, or more privileged than the level given in the IOPL field.

The 386 format TSS contains an I/O Permission bitmap, which is referenced in protected mode if CPL > IOPL, and is always referenced in Virtual 86 mode. The I/O Permission bitmap is contained within the 386 TSS segment, at an offset specified by a 16-bit offset field named *BitMapBase* in the *TaskAttribute* field in the 386 format TSS.

A program executing in Protected mode with a 286 format TSS will use only the IOPL check to determine if an I/O access is allowed, since there is no I/O Permission bitmap in the 286 format TSS! A program executing in Protected mode with a 386 format TSS will first use the IOPL test, and if it fails, will then reference the I/O Permission bitmap to see if access to the I/O port address is permitted. A program executing in Virtual 86 mode will bypass the IOPL check, and will only check the I/O Permission Bitmap to determine if access to the given I/O port is permitted.

1.1. I/O Permission Bit Map

The I/O Permission bit map can be viewed as a 0-64KBit bit string, which starts in memory at offset *BitMapBase* in the current TSS. The pointer BitMapBase is found in the upper 16 bits of the previously defined *TaskAttribute* double word field in the TSS. Each bit in the bit string corresponds to a single byte-wide I/O port (i.e. the bit for port 40 can be found at address 5, bit offset 0).

Since every port must be protected from multi-byte transfers below it, the enabling bit for *every* byte-wide port being used in a transaction must be valid for the transaction to be allowed. The algorithm accounts for any possible length and alignment combination.

Due to the use of a pointer to the base of the bit map, the bit map can be located anywhere in the TSS, or may be ignored completely (regressing to the original IOPL trap case) by pointing the BitMapBase pointer off the end of the TSS Segment. In the same manner, only a small portion of the 64K I/O space need have an associated map bit, by limiting the size of the TSS. (For example, setting the TSS Limit to {*BitMapBase* + 32} will allow bit mapping the first 256 I/O ports, while causing traps on any port greater than #255). This eliminates the commitment of 8K of memory when its not required, while allowing the fully general case if desired. If all the referenced bits are zero, the I/O will be allowed, if any are one, the I/O operation will cause a General Protection violation, with an error code of 0. Since the algorithm always reads a word of bit map information, there must always be a byte containing all ones *after* the last bit map byte which contains valid mapping information but before the tss limit.

The algorithm, examples, and a diagram describing the I/O bitmap function are shown below:

Algorithm:

```
    if RealMode then Return;
    if (ProtectedMode AND VM=0 AND CPL <= IOPL) then Return;
      else begin (* Virtual 86 mode, or Protected mode with CPL > IOPL *)
       if (TSS286) then GPFault(0);
       Ptr := (Read2(TSS+66));
       BitStringAddr := (PortNumber SHR 3) + Ptr;
       MaskShift := (PortNumber AND 7);
       nBitMask  := if (wl=BYTE) then 1
                       else if (wl=WORD) then 3
                       else if (wl=DWORD) then 15;
       mask := (nBitMask SHL MaskShift);
       CheckString := (Read2(BitStringAddr) AND mask);
       if (CheckString=0) then RETURN
       else GPFault(0)
    end;
```

The first tests determine the current execution mode of the processor, and take action appropriately. REAL mode simply returns to execute the I/O instruction. Protected mode checks IOPL first, and returns if the IOPL test passes. Virtual 86 mode (VM=1), bypasses the IOPL check to always check the bitmap.

Next, the type of TSS is checked. Since 286 format TSS's have no I/O Permission Bitmap, if the current TSS has the 286 format, a GP fault is reported to be compatible with the 286 handling of I/O instructions if CPL > IOPL.

After these tests, the actual bitmap is referenced. The offset of the start of the bitmap is read from the BitMapBase field in the TSS (at offset 66 hex) and added to the ByteOffset of the relevant bits in the bitmap formed by shifting the I/O address right by 3. Two bytes are read at this byte address to ensure that the necessary bits are read for all possible I/O address alignments and lengths. The length mask is formed based on the size of the I/O reference, from 1 to 4 bits long for 1 to 4 byte I/O references. This length mask is shifted left by the I/O address MOD 8 to align the low order mask bit with the BitMap bit corresponding to the I/O address. The Length mask is ANDed with the bytes read from the BitMap to clear the irrelevant bits. If the result is zero, the I/O access is allowed. If the result is non-zero, one or more of the I/O addresses spanned by the reference is not accessible, and a GP Fault is signalled.

1.2. Special OS Considerations

1.2.1. Initializing for the Null BitMap

If a 386 OS wishes to have I/O controlled only by IOPL, and to be disallowed in Virtual 86 mode, the BitMapBase field can simply be initialized to FFFF hex. This will provide an offset greater than the TSS limit (assuming the TSS segment is < 64K!), which specifies the "null" bitmap.

1.2.2. Watchdog Timer Protection for Interrupt Disabling

If an OS provides an application in Virtual 86 mode direct access to the IF flag by setting IOPL to 3, or if it permits the application to disable interrupts through supervisor intervention, a watchdog timer can be attached to the NMI interrupt to gain control back if the application leaves interrupts diabled for too long an interval. One method for doing this is to use two timers: one is the normal system timer that interrupts the CPU periodically with the highest priority INTR interrupt level. When this timer interrupt is serviced, the watchdog timer is reloaded with a count that is the sum of the normal interval plus the maximum interrupt disabled interval. If the INTR interval timer is serviced within this latency after INTR is raised, the watchdog timer will be reset without giving an NMI. If the INTR interval timer interrupt is not serviced within the maximum latency, the watchdog timer will interrupt the processor with an NMI, so that control can be regained.

1.2.3. Virtual 86 Handling of INT n if IOPL=3

In Virtual 86 mode, the INT n instruction will cause a GP(0) fault if IOPL<3. This allows the OS to cleanly intercept interrupts from the Virtual 86 task. However, if the OS provides the Virtual 86 task direct access to IF, IOPL must be set at 3, which will diable the trapping of INT n. This function can still be provided by the OS by a careful choice of DPL for the gates in the IDT, and by extra code at the front of interrupt handlers that have DPL=3, which generally are the gates for "calling" the OS to perform a function. Any vector n that must not be accessible using the INT n instruction can be protected from application program access (including Virtual 86 mode access) by setting DPL<3 in IDT entry n. Any vector m that must be accessible via the INT m instruction to Level 3 protected mode programs must have DPL=3, which means it will also be accessible to a Virtual 86 mode program. Handlers for these interrupts may need to have code to check to see if the "calling" program was executing in Virtual 86 mode or Protected Mode, and take action appropriately. This check at each "common" software interrupt "gate" to the OS will be a bit faster than the previous method of sifting through the debris after a GP fault to determine that a software interrupt occured from Virtual 86 mode, but it will require potentially several copies of this "virtual mode" check.

Sample BitMap

```
        +3       +2       +1       +0       (Byte Offset)

      7      0 7      0 7      0 7      0   (Bit Offset)
     +--------+--------+--------+--------+
     |11110110|00001111|01001100|00000011| BitMapBase    I/O 0..31
     +--------+--------+--------+--------+
     |00100011|11001010|11111100|11111001| BitMapBase+4  I/O 32..63
     +--------+--------+--------+--------+
     |11111111|11111111|11111111|11111111| BitMapBase+8  I/O 64..95
     +--------+--------+--------+--------+
     |00000000|00000000|00000000|00000000| BitMapBase+12 I/O 96..127
     +--------+--------+--------+--------+
              ......................
     +--------+--------+--------+--------+
          TSS LIMIT ->|11111111|   ...  |
                      +--------+--------+
```

This bit map permits access to I/O ports 2..9, 12..13, 15, 20..24, 27, 33..34, 40..41, 48, 50, 52..53, 58..60, 62..63, and 96..127. . All other ports are not accessible to a Virtual 86 task, and are accessible to a Protected mode task only if CPL <= IOPL. Note that the last byte before the TSS limit must be all 1's.

Example 1:

```
        IN    EAX, 07H   ; Read Dword port 7

Offset= Byte 0, bit 7
Mask = 15 SHL 7 = 78H
BitString  =      0100110000000011 (See map)
       And 78H    0000011110000000
       Result     0000010000000000 (Not Equal to Zero)
Fault!
```

Example 2:

```
        OUT   33, AX    ; Write word port 33 (Decimal)

Offset= Byte 4, bit  1
Mask = 3 SHL 1 = 6H
BitString  =      1111110011111001 (See map)
       And 6H     0000000000000110
       Result     0000000000000000
Result = 0 I/O allowed!
```