iAPX 286 LOADALL Instruction

by Bill Rash

Test Instruction Purpose

The iAPX 286 microprocessor (part number 80286) has an undocumented instruction used by Intel test programs to allow direct access to internal registers for fast initialization. The instruction is called LOADALL. Each 80286 is tested with the LOADALL instruction. LOADALL is guaranteed to work on each 80286.

LOADALL allows explicit control of the descriptor cache register associated with each segment register independent of the segment register value and descriptor tables. LOADALL can be used to extend either real mode or protected mode. The protected mode 80286 can be extended to emulate iAPX 86 real mode programs with LOADALL. In real mode, LOADALL can provide addressability to the other 15 Mbytes of the 286 physical address space.

The operation of LOADALL is closely tied to the internal hardware of the 80286. The iAPX 386 will not have the same internal hardware. LOADALL will not work on an iAPX 386. The iAPX 386 has an alternative means of emulating iAPX 86 real mode programs.

LOADALL Description

All CPU registers (including LDTR, TR, GDTR, IDTR, and MSW) are loaded from memory by this instruction. The normally hidden descriptor cache registers for the ES, DS, SS, CS, TR, and LDT registers are also loaded. LOADALL may be executed in either real address mode or protected mode (CPL must be 0). Any attempt to execute LOADALL at any privilege level other than 0 in protected mode causes exception 13 with an error code of 0.

LOADALL allows direct control over the base, limit, and access rights associated with each segment register. These values are kept in the descriptor cache registers which are normally hidden from programs. In protected mode, LOADALL can set the selector, base address, limit, and access rights for a segment register without a descriptor table entry corresponding to the program visible selector value. The normal protected mode protection rules can also be changed. In real address mode, the physical address, limit, and access rights for a paragraph id can also be changed from the normal real mode definition.

The standard protected mode segment register loading checks (for privilege and access rights) are not performed by LOADALL on the values loaded into the descriptor caches. Using LOADALL in iAPX 86 real mode also does not involve any checks. Once loaded, the 80286 hardware will perform physical address calculation within the segment, offset checks against the limit, and access rights checks for all memory accesses using that segment register in either operating mode.

To retain protected mode system integrity, the policies used to define descriptor table contents must also be applied to the dynamicly created descriptors loaded into the descriptor cache registers with LOADALL. Once defined, the 80286 segment access hardware will limit segment usage to the physical memory region defined.

The LOADALL instruction is encoded in two consecutive bytes as 00001111 00000101, with 00001111 at the lowest memory address. LOADALL executes in 195 clocks and performs 51 bus cycles.

LOADALL cannot switch the 80286 from protected mode to real mode. Once in protected mode, the MSW value loaded by LOADALL must have a one in bit position 0. The RESET input is the only way to reenter real mode.

LOADALL reads a 102 byte area of physical memory starting at physical memory location 000800H (2048). The entire execution state of the 80286 (consisting of 24 registers) is defined upon completion of this instruction. The descriptor cache registers for the ES, DS, SS, CS, TR, and LDT are directly loaded from this area. The instruction requires 195 clocks with no wait states.

LOADALL Memory Area Format

Physical Memory Address in Hexadecimal	Associated CPU Register
800-805	None
806-807	MSW
808-815	None
816-817	TR
818-819	Flag word
81A-81B	IP
81C-81D	LDT
81E-81F	DS
820-821	SS
822-823	CS
824-825	ES
826-827	DI
828-829	SI
82A-82B	BP
82C-82D	SP
82E-82F	BX
830-831	DX
832-833	CX
834-835	AX
836-83B	ES descriptor cache
83C-841	CS descriptor cache
842-847	SS descriptor cache
848-84D	DS descriptor cache
84E-853	GDTR
854-859	LDT descriptor cache
85A-85F	IDTR
860-865	TSS descriptor cache

No checks are made between the program visible selector values and the associated descriptor table entry. LOADALL does not perform any descriptor table accesses. No checks are made regarding the type or access rights defined by the descriptor. Any new descriptors defined by this instruction will be automatically used by subsequent processor extension memory references.

Any subsequent segment register load instruction will reload the associated descriptor cache register in the normal manner according to the operating mode of the CPU. In real mode, the low 4 bits and high 4 bits of the base address are set to zero. The paragraph id is inserted into bits 19-4 of the base address. The segment limit is reset to FFFFH and access rights is changed to a writable segment. In protected mode, the base address, limit, and access rights are loaded from the descriptor.

The descriptor cache entries are in the following format:

bytes 0-2 — 24-bit physical base address of the segment. The bytes are stored in ascending order with the least significant byte at lowest memory address.

byte 3 — Access rights byte is in the format of the access byte in a descriptor. The only difference is that the present bit becomes a valid bit. If zero, the descriptor is considered invalid and any memory reference using the descriptor will cause exception 13 with an error code of zero. Loading a descriptor cache register with an invalid descriptor does not cause an immediate exception. Any attempted use of the descriptor to reference memory causes the exception. Such an exception is restartable and the saved machine state appears as if the instruction had not been attempted. The value loaded by LOADALL can be read without any exceptions. The DPL fields of the SS and CS descriptor caches determine the CPL. The DPL fields of the DS and ES descriptor caches should be 3. The CS descriptor may be loaded with a writable data segment descriptor.

bytes 4-5 — 16-bit limit of the segment. The word is stored in two bytes in normal word format. The interpretation of this field is determined by the type of segment identified by byte 3. Grow-down data segments are a special case of how to interprete the limit field. The data sheet describes how this field works.

The GDTR and IDTR are in the following format:

bytes 0-2 — 24-bit physical base address of segment. The bytes are stored in ascending order with the least significant byte at the lowest memory address.

byte 3 — Should be zeroes

bytes 4-5 — 16-bit limit of the segment. The word is stored as two bytes in normal word format.

To allow a program to execute properly after executing LOADALL, the following is required of the descriptor cache register contents:

1. The stack segment is a writeable, valid data segment.

2. The code segment can be of three types: execute only, read/execute only, or read/write/execute. To be execute only, use an execute only code segment access rights byte value. To be execute/read only use a execute/read code segment access rights byte value. To be read/write/execute use a writable, expand-up data segment access rights byte value.

For proper protected mode operation, the following is required:

3. The DPL field of the CS descriptor cache access rights byte must equal the DPL field of the SS descriptor cache access rights byte. These DPL fields are the CPL of the processor.

4. The DPL fields of the ES and DS descriptors should be 3 to prevent their being zeroed by RET or IRET instructions.

Executing Real Mode Programs in Protected Mode

An iAPX 86/88 program using real mode addressing can be executed in protected mode with full protection between it and other programs. All segment register semantics of iAPX 86 real mode can be emulated. The address space of the real mode program can also be limited to less than 1 megabyte and be relocated anywhere in the 16 Megabyte physical address space. The following sections describe several aspects of this emulation.

Address space relocation and control

iAPX 86 real mode emulation requires any segment register load instruction cause a protection exception. An error code with bits 1-0 being zero and bits 15-2 being non-zero identify a segment register load exception. Such exceptions are restartable. All instructions that do not load a segment register run at full iAPX 286 speed and with full access checks.

The exception handler must interprete the segment register load instruction to place the iAPX 86 paragraph id and the associated protected descriptor into the LOADALL memory area. LOADALL then loads the segment register with the value used by the interrupted instruction, points the descriptor cache entry at the protected physical memory region, and restores the other segment registers.

Most segment load instructions will cause exception 13 if all LDT and GDT entries are marked with a privilege level less than the CPL of the emulated program. The CPL of the emulated program is defined by the DPL fields of the CS and SS descriptor caches.

Segment register loads using a selector value of 0000H to 0003H do not cause an exception on loading the segment register. Instead, any memory reference using the segment register will cause exception 13 with an error code of 0. No memory reference will occur. This case can be identified by checking whether DS or ES contain a value of 0-3. These exceptions are also restartable.

Limits can be enforced on the size of the emulated iAPX 86 address space. An iAPX 86 paragraph id that is outside the defined memory area can be loaded, but the segment register can be marked invalid for memory addressing. LOADALL can be used to load the iAPX 86 paragraph id into the segment register, but the descriptor cache entry is marked invalid. The paragraph id can still be read without causing a protection exception.

If a selector value is loaded whose segment overruns the end of the defined physical memory area, the limit field can be set less then 65535 to prevent accesses outside the defined memory area with that segment register.

The emulated iAPX 86/88 address space can be relocated anywhere in the 16 Mbyte iAPX 286 physical address space by adding a 24-bit relocation factor to the 20-bit iAPX 86/88 physical address value associated with the iAPX 86 paragraph id.

iAPX 86/88 Interrupt Table Simulation

The LOADALL instruction allows a protected mode 80286 to provide a simulated iAPX 86/88 interrupt table to iAPX 86/88 programs. The protected mode iAPX 286 interrupt table is different from iAPX 86/88 since it must contain more information and be protected from improper use. The protected mode interrupt table can not be addressed by the same selector-offset pairs used in iAPX 86 real address mode.

The iAPX 86/88 interrupt table is simulated by having all INT instructions cause a protection exception. Setting the DPL of all IDT gate entries to less than the CPL of the emulated program will cause exception 13 for all INT instructions. The error code will indicate an IDT vector with the EXT bit cleared. External interrupts and program exceptions will continue to use the protected IDT.

The iAPX 86/88 INT instruction can be simulated by the exception 13 handler. For INT instructions, it looks into the iAPX 86 interrupt vector table for the vector associated with the interrupt vector in the error code. After simulating the machine state save, the iAPX 86/88 program is restarted at the interrupt vector address.

Interrupt handlers for external interrupts can pass control to an iAPX 86 real mode program. Each external interrupt handler for an iAPX 86 interrupt must determine if the interrupt is for a real mode program, if so then it emulates a real mode interrupt the same way as for the INT instruction.

Allowing writes into a code segment

Code segment writes are possible by using writeable data segment descriptors for the CS cache entry. Normally the code segment is write protected. If the code segment descriptor is always marked writable, then writes using the CS prefix will work correctly.

Allowing temporaries to be placed into segment registers

A temporary value which does not correspond to a valid segment causes exception 13. It is possible to place that value into the program visible segment register, but mark the descriptor cache entry invalid. The invalid descriptor lets the program reference the numeric value stored in the segment register value, (i.e. MOV AX,ES) but prevents any memory reference instruction from using the segment register to address memory (i.e. MOV AX,ES:[BX]).

This feature requires an error handler to know that exception 13 with an error code which is an invalid segment selector value indicates a potential temporary value problem. The exception handler must simulate the segment load instruction to place the error code into the appropriate segment register and use LOADALL to mark the descriptor cache entry invalid. The program may then be resumed after the segment load instruction.

Simulating I/O

All I/O instructions of the iAPX 86 program can be simulated. When the IOPL (I/O privilege level) is less than the CPL of the simulated iAPX 86 program, exception 13 will occur, with an error code of 0, on IN, OUT, STI, CLI, and LOCK instructions. The exception handler can identify these instructions and emulate their actions. The iAPX 86 program can then be restarted.

The LOCK instruction prefix causes exception 13 when CPL is greater than IOPL. For most systems, the LOCK prefix could be ignored. Restarting the program after the LOCK prefix would be acceptable. In special cases, the LOCKED instruction may need to be run with a lower CPL.

Protecting against errant iAPX 86 programs

The emulator for an iAPX 86 can protect against any random iAPX 86 program. The following steps are reccommended:

1. Store a 0 in the back link field of all task state segments to catch programs that incorrectly attempt to execute IRET with the NT flag set. Such an IRET will cause exception 13 with an error code of 0. The IRET can be emulated in the normal manner. If a task does not have a task gate pointing at it, and it is never invoked via a CALL instruction, then the back link field will never change.

2. Define the invalid opcode exception and bound exception in the IDT.

3. Test the value of a selector + 64K to see if it exceeds the last address allowed for a program. If so set the segment limit to a value limiting the segment to the defined range.

Mixing emulated real mode software with native protected mode software

A system which emulates a real mode program may also run protected mode software. If the GDT and IDT has all entries marked level 2 or less, the emulated program cannot use them if it runs at level 3. The emulated program can have a task state segment associated with it. An LDT may be present if all entries are marked level 2 or less. Normal protected mode tasks may use an LDT with entries at privilege level 3.

Interrupt handlers may use either task or interrupt/trap gates. All interrupt handlers using trap/interrupt gates must execute at privilege level 2 or less. Interrupts that use task gates may run at any privilege level.

The register save operation of the task switch or interrupt handler will work without exceptions. The iAPX 86 paragraph ids in the segment registers can be read without a protection exception. The segment registers can be reloaded with protected selectors without a protection exception. Interrupting from a emulated iAPX 86 program does not affect interrupt latency.

Returning from an interrupt requires some checks. The return from the interrupt handler must check whether an iAPX 86 real mode program had been executing. If so, the return sequence must use the LOADALL instruction to reload all the registers rather than the normal IRET instruction.

Depending on the iAPX 86 paragraph ids used, the IRET instruction might not cause a protection exception on returning to an emulated iAPX 86 program. The CS value of an interruptted iAPX 86 program saved on the stack or in the TSS does not correctly identify the privilege level, normally 3, of the emulated iAPX 86 real mode program. The privilege level of the interrupted program is determined by the RPL fields of the saved CS and SS selectors. If these values are the same and refer to a visible code segment, the CPU could attempt to execute the protected code segment at an incorrect address.

The interrupt handler should test whether an emulated iAPX 86 program was executing. An interrupted protected mode program can be restarted in the normal manner while an emulated program requires LOADALL.

Emulating an 8087 with the 80287

The instruction and data addresses saved in the protected mode 80287 environment area are in a different format than from the 8087. In real mode the 80287 environment is in the same format as the 8087. In protected mode the 80287 environment is changed to store 32-bit virtual pointers rather than 20-bit iAPX 86/88 physical addresses.

The 80287 can be used by both normal protected mode programs and emulated iAPX 86/88 real mode programs. The 80287 operates in either real mode or protected mode. The FSETPM instruction must be executed before starting a normal protected mode program if the 80287 was in real mode. The 80287 must be reset, via the RESET pin, to reenter real mode for an emulated iAPX 86/88 program after being used by a normal protected mode program. External hardware could reset the part to reenter real-mode. The TS bit of the MSW can be used to monitor for the first ESCAPE instruction executed in a program. The exception 7 handler can then determine what mode of operation is required in the 80287.

The 20-bit physical addresses kept by the 80287 for the instruction and data pointers will reflect the paragraph id in the program visible segment register and offset used by the ESC instruction to address memory. The descriptor cache base and limit loaded by LOADALL is used to generate physical memory addresses for data transfers.

The WAIT instructions required by the 8087 before ESC instructions can be safely executed by the 80287.

ESC instructions can cause exception 13 if the operand is outside a segment's limit. The exception 13 handler should expect such exceptions. The instruction can be restarted by simply returning to it.

If exception 9 occurs, the second or subsequent word of a numeric operand exceeded the segment's limit. This exception can <u>not</u> be restarted. The exception 9 handler must execute FNINIT before <u>any</u> other ESC or WAIT instruction. The 8086 address of the instruction and data operand will be saved in the 80287. Use FSTENV or FSAVE after FNINIT to read these values. The top of stack pointer and tag register will be changed by the FNINIT instruction.

Discrepancies from an iAPX 86/88 Using Emulation

An 80286 can not exactly emulate an 8086/88 in all possible cases. Most differences are due to the extra protection checks made in the 80286 which are not made in the 8086/88. The discrepencies listed here are minor enough that very few programs will be affected.

1. The PUSH SP instruction pushes a different value on the iAPX 286 than on the iAPX 86,88,186. The value pushed onto the stack by the 80286 is the value of SP before the push instruction executes. The value pushed onto the stack by the 8086/88/186 is the SP value after the push instruction executes.

2. Shifts and rotates on the iAPX 286 mask the count to 5 bits. The iAPX 86/88 allows all 8 bits to be used. The iAPX 186/188 also masks the shift count to 5 bits.

3. Segment wrap-around is not allowed on the 80286. Segment limit violations are not restartable in general on the 80286. Programs that rely on reading some special value when referencing non-existent memory may not be correctly run.

   Exceptions 9, 12, or 13 occur during attempts to wrap-around a segment depending on the location and type of operand involved. All exception 12 cases can be emulated and the program restarted. Exception 13 or 12 that occurs for an ESC instruction occurs before the 80286 or 80287 execute the instruction, and are therefore restartable. Exception 9 can not be restarted.

   Most simple load and store instructions that violate a segment limit are restartable. The current case that can not be restarted in general is:

   > Any floating point operand reference where the second or subsequent word exceeded a segment limit. The exception 9 handler must execute FNINIT before any other wait or ESC instruction. The internal status of the 80287 cannot be read until it is forced idle by FNINIT. The FNINIT instruction will mark all floating point data registers as empty, set top of stack to 0, and mask all errors. The numeric instruction and data addresses stored in the 80287 will correctly point at the failing instruction. If the 80286 program interrupted by the math address error is not the program that executed the failed ESC instruction, then that program can be restarted.

4. Memory address space wrap-around is not directly supported. The iAPX 86/88 allow wrap around from the top of the 1 megabyte address space into the bottom of the 1 megabyte address space (i.e. address FC00:4000 is same as 0000:0000). To emulate instructions that address memory with such wrap-around, requires the segment register limit be set to cause a protection exception for addresses beyond simulated physical address OFFFFFH, or addresses below 00000H using an expand down segment, and software emulation of the instruction to address memory at the bottom of the address space.

5. An 8086/88/186 program will require different amounts of time to execute instructions on the 80286. Most instructions will run faster on the 80286. Instructions which do not modify a segment register, and do not use a segment register with a zero in it will run faster on the 80286. Instruction that first access memory with a segment register containing a zero will run slower on the 80286. Instructions that load a segment register with a non-zero selector value will run slower on the 80286.

6. The iAPX 286 and iAPX 186 can generate the most negative number as a quotient for the IDIV instruction. The iAPX 86/88 will generate the divide error exception instead.

7. The iAPX 286 divide error return address will point at the divide instruction including prefixes. The registers will appear as if the instruction had not executed. The iAPX 86/88/186/188 return address will point after the divide instruction and the DX:AX or AH:AL registers may have been changed.

8. The numeric instruction address stored in the 80287 includes all leading prefixes before the ESC opcode. The 8087 numeric instruction address always points at the ESC opcode.

9. An iAPX 286/20 system does not require an interrupt controller for the ERROR signal. iAPX 86/20 systems use an interrupt controller to prioritize simultaneous interrupts and mask errors from the 8087 if servicing them must be delayed.

   If the same interrupt controller is provided as in the iAPX 86 system, the input used for the 8087 ERROR signal can be grounded. Instructions that control that input of the interrupt controller become NOPs. Watch out for non-specific EOI instructions inside an 8087 error handler which may affect other interrupt inputs.

   80287 errors do not normally affect an interrupt handler. As long as any program does not execute WAIT or ESC instructions, it can not be interrupted by the 80287.

   If a different interrupt system is used in the iAPX 286 system than in the iAPX 86 system, any I/O instructions to the interrupt controller may have to be emulated.

10. Numeric error interrupts use interrupt vector 16. Since an external interrupt controller may be used in iAPX 86,88,186 systems, another interrupt vector may have been used for numeric interrupts.

11. Do not perform port I/O to ports 00F8H to 00FFH. The 80287 may not operate properly if this is allowed. These I/O locations are reserved by Intel.

12. The interrupt enable bit of the flag word may not change when a POPF or IRET instruction attempts to change it. The IOPL field of the flag word controls whether IF can be changed. Subsequent PUSHF and INT instructions will save a value of IF which differs from the value in an 8086/8088 program.

13. If STI and CLI are emulated as NOPs then they will fail to change IF. Subsequent PUSHF and INT instructions will save a value of IF which differs from the value in an 8086/8088 program.

14. The flag word has two new fields: IOPL and NT. IOPL can not change except at level 0, but NT can be changed by IRET and POPF instructions. The IRET instruction attempts a task switch when NT is set. The back link field of the current TSS should have a 0 in it to cause exception 13, with an error code of 0, if the iAPX 86 program attempts an IRET after setting NT. The exception 13 handler may then simulate an iAPX 86 IRET operation.

15. The iAPX 86 address space may be limited. Programs may use some form of memory space scanner to see how much memory is available. Accesses to illegal locations are expected. The program emulator must decide what to do about illegal accesses.

16. The 80286 defines new instructions for undefined opcodes in the 8086/88. An 8086/88 program with an unknown bug in it that executes these undefined opcodes will work differently on an 80286.

17. Programs with self-modifying code may work differently on an 80286. The 80286 prefetcher can fetch more bytes ahead of the current instruction than the 8086 or 8088. A program that modifies an instruction that has already been prefetched will not see the changed instruction. Any program which jumps after modifing an instruction before executing it will correctly execute the modified instruction.

18. Regions of the emulated iAPX 86,88,186 address space can not be write protected. The XCHG, ADC, SBB, RCL, and RCR instructions are not restartable if their memory-based operand is in a write protected segment.

Extending the Address Space of Current iAPX 86 Software

Current iAPX 86 real mode programs can use the extended address space of the iAPX 286 in a limited manner. To address the extended memory, LOADALL must be used to load the descriptor cache with an base address beyond the normal 1 Mbyte address range. That segment register must not be changed by software, else the segment register will point back into the 1 Mbyte address space.

Two types of systems are examined: accessing a single large data base in a limited manner, or splitting software into normal and extended areas. The first is the easiest to implement, while the second is more general.

Access to a large data area outside the 1 Mbyte address space could be provided by a subroutine. The subroutine scans the large data structure to locate the necessary item, then copy all data between the normal address space and the extended address space.

Interrupts must be disabled while the subroutine uses segment registers that have been set by LOADALL. The reload of segment registers inside an interrupt routine would change the actual physical address from that loaded by LOADALL before the interrupt. After all accesses in the extended area are done, interrupts may be enabled.

Returning the address of an extended data structure requires passing data through a segment register. For example, the ES register could have been changed by LOADALL to point at a data area outside the bottom megabyte of physical memory. The subroutine must not reload ES while it runs. The value stored in ES is not important since it is not related to the physical address. Interrupts must not be allowed since the interrupt routine may reload ES.

A second technique uses special paragraph ids (i.e. FFFFH) to signal that a peice of software is running in extended mode. All interrupt handlers in the system must look when they return to the interrupted program to see if any of the segment registers contain FFFFH. If so then that segment register points at extended memory. LOADALL must be used to load all the registers and the segment base address used last. The LOADALL memory area should contain that value left there from the previous usage. Descriptors for the other segment registers with normal paragraph ids must be constructed before executing LOADALL.

A semaphore must be placed around software that writes into the LOADALL area such that once written into, the software can execute LOADALL without interruption.

Mixing Real Mode and Protected Mode

The 80286 can alternate between real mode and protected mode. Some programs could be executed in real mode in the bottom megabyte of memory, while others execute in protected mode in the upper 15 Mbytes of memory. An external OR gate could RESET the 80286, independent of the rest of the system, to force it to enter real mode. A short routine at the power up address could redirect the software to the correct real mode program.

After executing the real mode program, LOADALL could then quickly restart the protected mode software. LOADALL can be used as a form of task switch from real mode to a protected mode task.

One operating system could service both the real and protected mode software. Any operating system call from the real mode program would cause a switch to protected mode. The protected mode software could then construct descriptors that refer to the same physical memory addresses used by the real mode paragraph ids. After conversion, the operating system could then perform all work in protected mode.

Interrupts must be handled specially. Interrupt handlers for both real mode and protected mode must be present at all times. If an interrupt handler needs to access a data area, that data area must be addressable from both real and protected mode. The real mode interrupt table would be kept at location 000000H. The protected mode IDT could be anywhere. LOADALL will switch to the protected interrupt table.

Implementation Notes

The exception 13 handler will probably use a lookup table for the opcode byte of the instruction causing exception 13 to determine the correct action for this instruction. In general, any undefined opcode causes exception 6 and would therefore not invoke exception 13. However, some implementations may emulate some instructions. The following explains the empty entries in the opcode map to aid in determining an emulation strategy.

The following is a list of exclusions from the general rule of undefined 80286 opcodes causing exception 6.

The LOADALL instruction (opcode 0F04H) will cause exception 13 in protected mode if executed when CPL is not 0. LOADALL may be executed at any time in real address mode.

The 0F05H opcode will cause exception 13 in protected mode if executed when CPL is not 0. If 0F05H is executed in real address mode, or in protected mode when CPL=0, the 80286 stops normal execution. RESET must be used to restart the CPU in this case. The 0F05H opcode may be executed at any time in real address mode.

The opcode 82H is an alias for opcode 80H.

The 0D0H/0D1H opcode with a REG field = 6 is an alias for the SAR instruction (REG = 7).

The opcode 0D6H is a proprietary single byte instruction. It cannot cause any protection exceptions in either real mode or protected mode. No restrictions apply to its execution. It can be emulated as a NOP.

The 0F1H opcode is a prefix which performs no function. It counts like any other prefix towards the maximum instruction length. No restrictions apply to its execution.

The 0F6H/0F7H opcode with a REG field = 1 is an alias for the TEST instruction (REG=0).

Restarting string instructions which caused exception 12 (if SS override was used) or exception 13 requires updating SI, DI, and CX (if repeat was used). Which registers are updated depends on the instruction and when the exception was detected. The following rules apply:

For STOS the DI register must always be updated by the exception handler to restart the instruction. The state of the DF bit in the flag word and the operand size determines whether to use +2, +1, -1, or -2 to update DI. If a repeated STOS was used, add 2 to CX to restart the instruction. If a repeated STOS is restarted, the last write performed by STOS before the protection violation occurred may be repeated.

For INS, without an IOPL violation, the DI register must always be updated by the exception handler to restart the instruction. The state of the DF bit in the flag word and the operand size determines whether to use +2, +1, -1, or -2 to update DI. If a repeated INS was used, increment CX by two to restart the instruction.

For a repeated INS with an IOPL violation, increment CX to restart the instruction, no update of DI is required. No changes are required to restart a non-repeated INS with an IOPL violation.

For SCAS the DI register must always be updated by the exception handler to restart the instruction. The state of the DF bit in the flag word and the operand size determines whether to use +2, +1, -1, or -2 to update SI. If SCAS was repeated, increment CX to restart it.

For OUTS or LODS the SI register must always be updated by the exception handler to restart the instruction. The state of the DF bit in the flag word and the operand size determines whether to use +2, +1, -1, or -2 to update SI. If OUTS or LODS was repeated, increment CX to restart it. If the exception was an IOPL violation for OUTS, and OUTS was repeated, then increment CX again to restart OUTS.

For MOVS the SI register must always be updated by the exception handler to restart the instruction. The state of the DF bit in the flag word and the operand size determines whether to use +2, +1, -1, or -2 to update SI. The DI register must also be updated if the source operand (i.e. DS:SI or seg:SI if a segment override prefix was used) did not cause the exception. After updating SI, look at the source operand address to see if exception 13 would occur. If not, then DI must also be updated the same as SI. Always increment CX to restart MOVS if it was repeated. IF DI was updated, and a repeat prefix was used, then CX must be incremented again for correct instruction restart.

For CMPS the DI register must always be updated by the exception handler to restart the instruction. The state of the DF bit in the flag word and the operand size determines whether to use +2, +1, -1, or -2 to update DI. The SI register must also be updated if the ES:DI operand did not cause the exception. After updating DI, look at ES:DI to see if exception 13 would occur. If not then SI must also be updated the same as DI. Increment CX if CMPS was repeated, and SI was updated, to restart CMPS.

Early 80286 Errata of Interest

Early versions of the 80286 have several errata items which may effect the implementation of software to emulate an 8086/8088 on a protected mode 80286 or expansion of the address space in real mode. These errata are in the A1 and B1 steppings of the 80286 and are fixed in later steppings of the 80286.

If the ES register has a null selector or ES:DI exceeds the segment limit when executing either the non-repeated MOVS or INS instructions, the saved CS:IP value seen by the exception 13 handler will point after the MOVS or INS instruction. The saved CS:IP value in later steppings will point at the failed instruction (including prefixes).

If the segment register used for the destination operand in either the POP to memory, FSTSW/FNSTSW, or FSTCW/FNSTCW instructions has a null selector in it or the segment limit is violated, the saved CS:IP value seen by the exception 13 (or 12 if SS override was used) handler will point after the POP/FSTSW/FNSTSW/FSTCW/FNSTCW instruction. The saved CS:IP value in later steppings will point at the failed instruction (including prefixes).

If the stack limit is violated by a PUSH from memory instruction, the saved CS:IP value seen by the exception 12 handler will point after the PUSH instruction. The saved CS:IP value in later steppings will point at the failed PUSH instruction (including prefixes).

If a segment limit violation or IOPL violation occurs in the repeated MOVS, INS, OUTS, CMPS, SCAS, or STOS instructions, the value of CX seen by the exception 12 or 13 handler will be the value used at the start of the instruction. The SI and DI register values will reflect the iterations used by the instruction. Later steppings of the 80286 will assure the saved value of the CX register reflects the number of iterations performed.

The LOADALL instruction may incorrectly enter protected mode. This only affects systems that use LOADALL while in real mode and want to remain in real mode. Two possible workarounds are possible: execute LOADALL using 0-wait memory for the data values or be sure bit 0 of memory location 804H is zero. HOLD requests and processor extension data transfers should be inhibited while LOADALL is running. Later steppings of the 80286 will correctly load the MSW during LOADALL with HOLD and processor extension transfers.

Bill Rash
Applications Engineer
Microprocessor Products

intel

INTEL CORPORATION
(408) 987-6573
2625 Walsh Avenue, Santa Clara, California 95051