LOCK—Assert LOCK# Signal Prefix

Opcode	Instruction	Clocks	Description
F0	LOCK	1	Assert LOCK# signal for the next instruction

ises the LOCK# signal of the i486 processor to be asserted during uction that follows it. In a multiprocessor environment, this signal e that the i486 processor has exclusive use of any shared memory erted. The read-modify-write sequence typically used to implement test-and-set on the i486 processor is the BTS instruction.

The LOCK prefix functions only with the following instructions:

BTS, BTR, BTC	mem, reg/imm
XCHG	reg, mem
XCHG	mem, reg
ADD, OR, ADC, SBB, AND, SUB, XOR	mem, reg/imm
NOT, NEG, INC, DEC	mem

An undefined opcode trap will be generated if a LOCK prefix is used with any instruction not listed above.

The XCHG instruction always asserts LOCK# regardless of the presence or absence of the LOCK prefix.

The integrity of the LOCK prefix is not affected by the alignment of the memory field. Memory locking is observed for arbitrarily misaligned fields.

Flags Affected

None

Protected Mode Exceptions

#UD if the LOCK prefix is used with an instruction not listed in the "Description" section above; other exceptions can be generated by the subsequent (locked) instruction

Real Address Mode Exceptions

Interrupt 6 if the LOCK prefix is used with an instruction not listed in the "Description" section above; exceptions can still be generated by the subsequent (locked) instruction

Virtual 8086 Mode Exceptions

#UD if the LOCK prefix is used with an instruction not listed in the "Description" section above; exceptions can still be generated by the subsequent (locked) instruction

BTS—Bit Test and Set

Opcode	Instruction	Clocks	Description
0F AB	BTS *r/m16,r16*	6/13	Save bit in carry flag and set
	,r32	6/13	Save bit in carry flag and set
	,imm8	6/8	Save bit in carry flag and set
	,imm8	6/8	Save bit in carry flag and set

ightSRC];
C] ← 1;

Description

The BTS instruction saves the value of the bit indicated by the base (first operand) and the bit offset (second operand) into the CF flag and then stores 1 in the bit.

Flags Affected

The CF flag contains the value of the selected bi

Protected Mode Exceptions

#GP(0) if the result is in a nonwritable segment; #GP(0) for an illegal memory operand effective address in the CS, DS, ES, FS, or GS segments; #SS(0) for an illegal address in the SS segment; #PF(fault-code) for a page fault; #AC for unaligned memory reference if the current privilege level is 3

Real Address Mode Exceptions

Interrupt 13 if any part of the operand would lie outside of the effective address space from 0 to 0FFFFH

Virtual 8086 Mode Exceptions

Same exceptions as in Real Address Mode; #PF(fault-code) for a page fault; #AC for unaligned memory reference if the current privilege level is 3

Notes

The index of the selected bit can be given by the immediate constant in the instruction or by a value in a general register. Only an 8-bit immediate value is used in the instruction. This operand is taken modulo 32, so the range of immediate bit offsets is 0..31. This allows any bit within a register to be selected. For memory bit strings, this immediate field gives only the bit offset within a word or doubleword. Immediate bit offsets larger than 31 are supported by using the immediate bit offset field in combination with the

displacement field of the memory operand. The low-order 3 to 5 bits of the immediate bit offset are stored in the immediate bit offset field, and the high-order 27 to 29 bits are shifted and combined with the byte displacement in the addressing mode.

When accessing a bit in memory, the processor may access four bytes starting from the memory address given by:

(4 * (BitOffset DIV 32))

, or two bytes starting from the memory address given by:

(2 * (BitOffset DIV 16))

e. It may do this even when only a single byte needs to be at the given bit. You must therefore be careful to avoid referencing areas of memory close to address space holes. In particular, avoid references to memory-mapped I/O registers. Instead, use the MOV instructions to load from or store to these addresses, and use the register form of these instructions to manipulate the data.

XCHG—Exchange Register/Memory with Register

Opcode	Instruction	Clocks	Description
90+ *r*	XCHG AX,*r16*	3	Exchange word register with AX
90+ *r*	XCHG *r16*,AX	3	Exchange word register with AX
90+ *r*	XCHG EAX,*r32*	3	Exchange dword register with EAX
90+ *r*	XCHG *r32*,EAX	3	Exchange dword register with EAX
86 /*r*	XCHG *r8*,*r8*	3	Exchange byte register with EA byte
86 /*r*	XCHG *m8*,*r8*	5	Exchange byte register with EA byte
87 /*r*	XCHG *r16*,*r16*	3	Exchange word register with EA word
87 /*r*	XCHG *m16*,*r16*	5	Exchange word register with EA word
87 /*r*	XCHG *r32*,*r32*	3	Exchange dword register with EA dword
87 /*r*	XCHG *m32*,*r32*	5	Exchange dword register with EA dword

Operation

```
temp ← DEST
DEST ← SRC
SRC ← temp
```

Description

The XCHG instruction exchanges two operands. The operands can be in either order. If a memory operand is involved, the LOCK# signal is asserted for the duration of the exchange, regardless of the presence or absence of the LOCK prefix or of the value of the IOPL.

Flags Affected

None

Protected Mode Exceptions

#GP(0) if either operand is in a nonwritable segment; #GP(0) for an illegal memory operand effective address in the CS, DS, ES, FS, or GS segments; #SS(0) for an illegal address in the SS segment; #PF(fault-code) for a page fault; #AC for unaligned memory reference if the current privilege level is 3

Real Address Mode Exceptions

Interrupt 13 if any part of the operand would lie outside of the effective address space from 0 to 0FFFFH

Virtual 8086 Mode Exceptions

Same exceptions as in Real Address Mode; #PF(fault-code) for a page fault; #AC for unaligned memory reference if the current privilege level is 3