makes underflow very unlikely. Denormals are likely to arise only when an application generates a great many intermediates, so many that they cannot be held on the register stack or in temporary real memory variables. If storage limitations force the use of short or long reals for intermediates, and small values are produced, underflow may occur, and if masked, may generate denormals.

Accessing a denormal may produce an exception as shown in table S-24. (The denormalized exception signals that a denormal has been fetched.) Denormals may have reduced significance due to lost low-order bits, and an option of the proposed IEEE standard precludes operations on non-normalized operands. This option may be implemented in the form of an exception handler that responds to unmasked denormalized exceptions. Most users will mask this exception so that computation may proceed; any loss of accuracy will be analyzed by the user when the final result is delivered.

As table S-24 shows, the division and remainder operations do not accept denormal divisors and raise the invalid operation exception. Recall, also, that the transcendental instructions require normalized operands and do *not* check for exceptions. In all other cases, the NDP converts denormals to unnormals, and the unnormal arithmetic rules then apply.

Unnormals

An unnormal is the "descendent" of a denormal and therefore of a masked underflow response. An unnormal may exist only in the temporary real format; it may have any exponent that a normal may have, but it is distinguished from a normal by the integer bit of its significand, which is always 0. An unnormal in a register is tagged valid.

Unnormals allow arithmetic to continue following an underflow while still retaining their identity as numbers which may have reduced significance. That is, unnormal operands generate unnormal results, so long as their unnormality has a significant effect on the result. Unnormals are thus prevented from "masquerading" as normals, numbers which have full significance. On the other hand, if an unnormal has an insignificant effect on a calculation with a normal, the result will be normal. For example, adding a small unnormal to a large normal yields a normal result. The converse situation yields an unnormal.

Table S-25 shows how the instruction set deals with unnormal operands. Note that the unnormal may be the original operand or a temporary created by the 8087 from a denormal.

Table S-24. Exceptions Due to Denormal Operands

Operation	Exception	Masked Response
FLD (short/long real)	D	Load as equivalent unnormal
arithmetic (except following)	D	Convert (in a work area) denormal to equivalent unnormal and proceed
Compare and test	D	Convert (in a work area) denormal to equivalent unnormal and proceed
Division or FPREM with denormal divisor	I	Return real *indefinite*

Table S-25. Unnormal Operands and Results

Operation	Result
Addition/subtraction	Normalization of operand with larger absolute value determines normalization of result.
Multiplication	If either operand is unnormal, result is unnormal.
Division (unnormal dividend only)	Result is unnormal.
FPREM (unnormal dividend only)	Result is normalized.
Division/FPREM (unnormal divisor)	Signal invalid operation.
Compare/FTST	Normalize as much as possible before making comparison.
FRNDINT	Normalize as much as possible before rounding.
FSQRT	Signal invalid operation.
FST, FSTP (short/long real destination)	If value is above destination's underflow boundary, then signal invalid operation; else signal underflow.
FSTP (temporary real destination)	Store as usual.
FIST, FISTP, FBSTP	Signal invalid operation.
FLD	Load as usual.
FXCH	Exchange as usual.
Transcendental instructions	Undefined; operands must be normal and are not checked.

Zeros and Pseudo-Zeros

As discussed in section S.3, the real and packed decimal data types support signed zeros, while the binary integers represent a single zero, signed positive. The signed zeros behave, however, as though they are a single unsigned quantity. If necessary, the FXAM instruction may be used to determine a zero's sign.

The zeros discussed above are called true zeros; if one of them is loaded or generated in a register, the register is tagged zero. Table S-26 lists the results of instructions executed with zero operands and also shows how a true zero may be created from nonzero operands. (Nonzero operands are denoted "X" or "Y" in the table.)

Only the temporary real format may contain a special class of values called pseudo-zeros. A pseudo-zero is an unnormal whose significand is all zeros, but whose (biased) exponent is nonzero (true zeros have a zero exponent). Neither is a pseudo-zero's exponent all ones, since this encoding is reserved for infinities and NANs. A pseudo-zero result will be produced if two unnormals, containing a total of more than 64 leading zero bits in their significands, are multiplied together. This is a remote possibility in most applications, but it can happen.

Table S-26. Zero Operands and Results

Operation/Operands	Result	Operation/Operands	Result
FLD, FBLD [1]		Division	
+0	+0	±0 ÷ ±0	Invalid operation
−0	−0	±X ÷ ±0	Zerodivide
FILD [2]		+0 ÷ +X, −0 ÷ −X	+0
+0	+0	+0 ÷ −X, −0 ÷ +X	−0
FST, FSTP		−X ÷ −Y, +X ÷ +Y	+0, underflow [8]
+0	+0	−X ÷ +Y, +X ÷ −Y	−0, underflow [8]
−0	−0		
+X [3]	+0	FPREM	
−X [3]	−0	±0 rem ±0	Invalid operation
FBSTP		±X rem ±0	Invalid operation
+0	+0	+0 rem +X, +0 rem −X	+0
−0	−0	−0 rem +X, −0 rem −X	−0
FIST, FISTP		+X rem +Y, +X rem −Y	+0 [9]
+0	+0	−X rem −Y, −X rem +Y	−0 [9]
−0	+0		
+X [4]	+0	FSQRT	
−X [4]	+0	−0	−0
		+0	+0
Addition			
+0 plus +0	+0	Compare	
−0 plus −0	−0	±0 : +X	A < B
+0 plus −0, −0 plus +0	*0 [5]	±0 : ±0	A = B
−X plus +X, +X plus −X	*0 [5]	±0 : −X	A > B
±0 plus ±X, ±X plus ±0	†X [6]		
		FTST	
Subtraction		±0	Zero
+0 minus −0	+0	FCHS	
−0 minus +0	−0	+0	−0
+0 minus +0, −0 minus −0	*0 [5]	−0	+0
+X minus +X, −X minus −X	*0 [5]	FABS	
±0 minus ±X, ±X minus ±0	†X [6]	±0	+0
		F2XM1	
Multiplication		+0	+0
+0 • +0, −0 • −0	+0	−0	−0
+0 • −0, −0 • +0	−0	FRNDINT	
+0 • +X, +X • +0	+0	+0	+0
+0 • −X, −X • +0	−0	−0	−0
−0 • +X, +X • −0	−0	FXTRACT	
−0 • −X, −X • −0	+0	+0	Both +0
+X • +Y, −X • −Y	+0, underflow [7]	−0	Both −0
+X • −Y, −X • +Y	−0, underflow [7]		

Notes:

[1] Arithmetic and compare operations with real memory operands interpret the memory operand signs in the same way.

[2] Arithmetic and compare operations with binary integers interpret the integer sign in the same manner.

[3] Severe underflows in storing to short or long real may generate zeros.

[4] Small values ($|X| < 1$) stored into integers may round to zero.

[5] Sign is determined by rounding mode:
* = + for nearest, up or chop
* = − for down

[6] † = sign of X.

(7) Very small values of X and Y may yield zeros, after rounding of true result. NDP signals underflow to warn that zero has been yielded by nonzero operands.

(8) Very small X and very large Y may yield zero, after rounding of true result. NDP signals underflow to warn that zero has been yielded from nonzero operands.

(9) When Y divides into X exactly.

Pseudo-zero operands behave like unnormals, except in the following cases where they produce the same results as true zeros:

- compare and test instructions
- FRNDINT (round to integer)
- division, where the dividend is either a true zero or a pseudo-zero (the divisor is a pseudo-zero).

In addition and subtraction of a pseudo-zero and a true zero or another pseudo-zero, the pseudo-zero(s) behave like unnormals, except for the determination of the result's sign. The sign is determined as shown in table S-26 for two true zero operands.

Infinities

The real formats support signed representations of infinities. These values are encoded with a biased exponent of all ones and a significand of $1_{\Delta}00...00$; if the infinity is in a register, it is tagged special. The significand distinguishes infinities from NANs, including real *indefinite*.

A programmer may code an infinity, or it may be created by the NDP as its masked response to an overflow or a zerodivide exception. Note that when rounding is up or down, the masked response may create the largest valid value representable in the destination rather than infinity. See table S-33 for details. As operands, infinities behave somewhat differently depending on how the infinity control field in the control word is set (see table S-27). When the projective model of infinity is selected, the infinities behave as a single unsigned representation; because of this, infinity cannot be compared with any value except infinity. In affine mode, the signs of the infinities are observed, and comparisons are possible.

Table S-27. Infinity Operands and Results

Operation	Projective Result	Affine Result
Addition		
$+\infty$ plus $+\infty$	Invalid operation	$+\infty$
$-\infty$ plus $-\infty$	Invalid operation	$-\infty$
$+\infty$ plus $-\infty$	Invalid operation	Invalid operation
$-\infty$ plus $+\infty$	Invalid operation	Invalid operation
$\pm\infty$ plus $\pm$X	$*\infty$	$*\infty$
$\pm$X plus $\pm\infty$	$*\infty$	$*\infty$
Subtraction		
$+\infty$ minus $-\infty$	Invalid operation	$+\infty$
$-\infty$ minus $+\infty$	Invalid operation	$-\infty$
$+\infty$ minus $+\infty$	Invalid operation	Invalid operation
$-\infty$ minus $-\infty$	Invalid operation	Invalid operation
$\pm\infty$ minus $\pm$X	$*\infty$	$*\infty$
$\pm$X minus $\pm\infty$	$\dagger\infty$	$\dagger\infty$
Multiplication		
$\pm\infty \bullet \pm\infty$	$\oplus\infty$	$\oplus\infty$
$\pm\infty \bullet \pm$Y	$\oplus\infty$	$\oplus\infty$
$\pm 0 \bullet \pm\infty$, $\pm\infty * \pm 0$	Invalid operation	Invalid operation

Table S-27. Infinity Operands and Results (Cont'd.)

Operation	Projective Result	Affine Result
Division		
$\pm\infty \div \pm\infty$	Invalid operation	Invalid operation
$\pm\infty \div \pm X$	$\oplus\infty$	$\oplus\infty$
$\pm X \div \pm\infty$	$\oplus 0$	$\oplus 0$
FSQRT		
$-\infty$	Invalid operation	Invalid operation
$+\infty$	Invalid operaton	$+\infty$
FPREM		
$\pm\infty$ rem $\pm\infty$	Invalid operation	Invalid operation
$\pm\infty$rem $\pm X$	Invalid operation	Invalid operation
$\pm Y$ rem $\pm\infty$	*Y	*Y
± 0 rem $\pm\infty$	*0	*0
FRNDINT		
$\pm\infty$	$*\infty$	$*\infty$
FSCALE		
$\pm\infty$ scaled by $\pm\infty$	Invalid operation	Invalid operation
$\pm\infty$ scaled by $\pm X$	$*\infty$	$*\infty$
± 0 scaled by $\pm\infty$	*0	*0
$\pm Y$ scaled by $\pm\infty$	Invalid operation	Invalid operation
FXTRACT		
$\pm\infty$	Invalid operation	Invalid operation
Compare		
$\pm\infty : \pm\infty$	A = B	$-\infty < +\infty$
$\pm\infty : \pm Y$	A ? B (and) invalid operation	$-\infty < Y < +\infty$
$\pm\infty : \pm 0$	A ? B (and) invalid operation	$-\infty < 0 < +\infty$
FTST		
$\pm\infty$	A ? B (and) invalid operation	$*\infty$

Notes: X = zero or nonzero operand

Y = nonzero operand

* = sign of original operand

† = sign is complement of original operand's sign

⊕= sign is "exclusive or" original operand signs (+ if operands had same sign, − if operands had different signs)

NANs

A NAN (Not-A-Number) is a member of a class of special values that exist in the real formats only. A NAN has an exponent of 11...11B, may have either sign, and may have any significand except $1_{\Delta}00...00$B, which is assigned to the infinities. A NAN in a register is tagged special.

The 8087 will generate the special NAN, real *indefinite*, as its masked response to an invalid operation exception. This NAN is signed negative; its significand is encoded $1_{\Delta}100...00$. All other NANs represent programmer-created values.

Whenever the NDP uses an operand that is a NAN, it signals invalid operation. Its masked response to this exception is to return the NAN as the operation's result. If both operands of an instruction are NANs, the result is the NAN with the larger absolute value. In this way, a NAN that enters a computation propagates through the computation and will eventually be delivered as

the final result. Note, however, that the transcendental instructions do not check their operands, and a NAN will produce an undefined result.

By unmasking the invalid operation exception, the programmer can use NANs to trap to the exception handler. The generality of this approach and the large number of NAN values that are available, provide the sophisticated programmer with a tool that can be applied to a variety of special situations.

For example, a compiler could use NANs to references to uninitialized (real) array elements. The compiler could pre-initialize each array element with a NAN whose significand contained the index (relative position) of the element. If an application program attempted to access an element that it had not initialized, it would use the NAN placed there by the compiler. If the invalid operation exception were unmasked, an interrupt would occur, and the exception handler would be invoked. The exception handler could determine which element had been accessed, since the operand address field of the exception pointers would point to the NAN, and the NAN would contain the index number of the array element.

NANs could also be used to speed up debugging. In its early testing phase a program often contains multiple errors. An exception handler could be written to save diagnostic information in memory whenever it was invoked. After storing the diagnsotic data, it could supply a NAN as the result of the erroneous instruction, and that NAN could point to its associated diagnostic area in memory. The program would then continue, creating a different NAN for each error. When the program ended, the NAN results could be used to access the diagnostic data saved at the time the errors occurred. Many errors could thus be diagnosed and corrected in one test run.

Data Type Encodings

Tables S-28 through S-31 summarize how various types of values are encoded in the seven NDP data types. In all tables, the less significant bits are to the right and are stored in the lowest memory addresses. The sign bit is always the left-most bit of the highest-addressed byte.

Notice that in every format one encoding is interpreted as representing the special value *indefinite*. The 8087 produces this encoding as its response to a masked invalid operation exception. In the case of the reals, *indefinite* can be loaded and stored like any NAN and it always retains its special identity; programmers are advised not to use this encoding for any other purpose. Packed decimal *indefinite* may be stored by the NDP in a FBSTP instruction; attempting to use this encoding in a FBLD instruction, however, will have an undefined result. In the binary integers, the same encoding may represent either *indefinite* or the largest negative number supported by the format (-2^{15}, -2^{31} or -2^{63}). The 8087 will store this encoding as its masked response to an invalid operation, or when the value in a source register represents, or rounds to, the largest negative integer representable by the destination. In situations where its origin may be ambiguous, the invalid operation exception flag can be examined to see if the value was produced by an exception response. When this encoding is loaded, or used by an integer arithmetic or compare operation, it is always interpreted as a negative number; thus *indefinite* cannot be loaded from a packed decimal or binary integer.

Table S-28. Binary Integer Encodings

	Class	Sign	Magnitude
Positives	(Largest)	0	11...11
	•	•	•
	•	•	•
	•	•	•
	(Smallest)	0	00...01
	Zero	0	00...00
Negatives	(Smallest)	1	11...11
	•	•	•
	•	•	•
	•	•	•
	(Largest/*Indefinite**)	1	00...00

Word: ←15 bits→
Short: ←31 bits→
Long: ←63 bits→

* If this encoding is used as a source operand (as in an integer load or integer arithmetic instruction), the 8087 interprets it as the largest negative number representable in the format: -2^{15}, -2^{31}, or -2^{63}. The 8087 will deliver this encoding to an integer destination in two cases:

1) if the result is the largest negative number,

2) as the response to a masked invalid operation exception, in which case it represents the special value *integer indefinite*.

Exception Handling Details

Table S-32 lists every exception condition that the NDP detects and describes the processor's response when the relevant exception mask is set. The unmasked responses are described in table S-6. Note that if an unmasked overflow or underflow occurs in an FST or FSTP instruction, no result if stored, and the stack and memory are left as they existed *before* the instruction was executed. This gives an exception handler the opportunity to examine the offending operand on the stack top.

When rounding is directed (the RC field of the control word is set to "up" or "down"), the 8087 handles a masked overflow differently than it does for the "nearest" or "chop" rounding modes. Table S-33 shows the NDP's masked response when the true result is too large to be represented in it's destination real format. For a normalized result, the essence of this response is to deliver ∞ or the largest valid number representable in the destination format, as dictated by the rounding mode and the sign of the true result. Thus, when RC=down, a positive overflow is rounded down to the largest positive number. Conversely, when RC=up, a negative overflow is rounded up to the largest negative number. A properly signed ∞ is returned for a positive overflow with RC=up, or a negative overflow with RC=down. For an unnormalized result, the action is similar except that the the unnormal character of the result is preserved if the sign and rounding mode do not indicate that ∞ should be delivered.

In all masked overflow responses for directed rounding, the overflow flag is *not* set, but the precision exception *is* raised to signal that the exact true result has not been returned.

Table S-29. Packed Decimal Encodings

Class		Sign		Magnitude																							
				digit				digit				digit				digit				. . .		digit					
Positives	(Largest)	0	0000000	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	. . .	1	0	0	1
		•	•														•								
		•	•														•								
		•	•														•								
	(Smallest)	0	0000000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	. . .	0	0	0	1
	Zero	0	0000000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	. . .	0	0	0	0
Negatives	Zero	1	0000000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	. . .	0	0	0	0
	(Smallest)	1	0000000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	. . .	0	0	0	1
		•	•														•								
		•	•														•								
		•	•														•								
	(Largest)	1	0000000	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	. . .	1	0	0	1
*Indefinite**		1	1111111	1	1	1	1	1	1	1	1	U	U	U	U	U	U	U	U	. . .	U	U	U	U
		← 1 byte →		← 9 bytes →																							

* The *packed decimal indefinite* encoding is stored by FBSTP in response to a masked invalid operation exception. Attempting to load this value via FBLD produces an undefined result.
Note: "UUUU" means bit values are undefined and may contain any value.

Table S-30. Real and Long Real Encodings

Class			Sign	Biased Exponent	Significand* Δff...ff
Positives		NANs	0 • • • 0	11...11 • • • 11...11	11...11 • • • 00...01
		∞	0	11...11	00...00
	Reals	Normals	0 • • • 0	11...10 • • • 00...01	11...11 • • • 00...00
		Denormals	0 • • • 0	00...00 • • • 00...00	11...11 • • • 00...01
		Zero	0	00...00	00...00

Table S-30. Real and Long Real Encodings (Cont'd.)

Class			Sign	Biased Exponent	Significand* $_{\Delta}$ff...ff
Negatives	Reals	Zero	1	00...00	00...00
		Denormals	1 • • • 1	00...00 • • • 00...00	00...01 • • • 11...11
		Normals	1 • • • 1	00...01 • • • 11...10	00...00 • • • 11...11
	∞		1	11...11	00...00
	NANs		1 • • •	11...11 • • •	00...01 • • •
		Indefinite	1	11...11	10...00
			• • • 1	• • • 11...11	• • • 11...11
			Short:	← 8 bits →	← 23 bits →
			Long:	← 11 bits →	← 52 bits →

* Integer bit is implied and not stored.

Table S-31. Temporary Real Encodings

Class		Sign	Biased Exponent	Significand I_{Δ}ff..ff
Positives	NANs	0 • • • 0	11...11 • • • 11...11	111...11 • • • 100...01
	∞	0	11...11	100...00

Table S-31. Temporary Real Encodings (Cont'd.)

Class			Sign	Biased Exponent	Significand $I_{\Delta}ff...ff$
Positives	Reals		0	11...10	Normals
			•	•	111...11
			•	•	•
			•	•	•
			•	•	•
			•	•	•
			•	•	100...00
			•	•	
			•	•	Unnormals
			•	•	
			•	•	011...11
			•	•	•
			•	•	•
			•	•	•
			0	00...01	000...00
					Denormals
			0	00...00	011...11
			•	•	•
			•	•	•
			•	•	•
			0	00...00	000...01
		Zero	0	00...00	000...00
Negatives		Zero	1	00...00	000...00
					Denormals
			1	00...00	000...01
			•	•	•
			•	•	•
			•	•	•
			1	00...00	011...11
			1	00...01	Unnormals
			•	•	000...00
			•	•	•
			•	•	•
			•	•	•
			•	•	011...11
			•	•	
			•	•	Normals
			•	•	
			•	•	100...00
			•	•	•
			•	•	•
			•	•	•
			1	11...10	111...11
	∞		1	11...11	100...00

Table S-31. Temporary Real Encodings (Cont'd.)

Class			Sign	Biased Exponent	Significand $I_{\Delta}ff...ff$
Negatives	NANs	*Indefinite*	1	11...11	100...00
			•	•	•
			•	•	•
			•	•	•
			1	11...11	110...00
			•	•	•
			•	•	•
			•	•	•
			1	11...11	111...11
				←15 bits→	←64 bits→

Table S-32. Exception Conditions and Masked Responses

Condition	Masked Response
Invalid Operation	
Source register is tagged empty (usually due to stack underflow).	Return real *indefinite*.
Destination register is not tagged empty (usually due to stack overflow).	Return real *indefinite* (overwrite destination value).
One or both operands is a NAN.	Return NAN with larger absolute value (ignore signs).
(Compare and test operations only): one or both operands is a NAN.	Set condition codes "not comparable".
(Addition operations only): closure is affine and operands are opposite-signed infinities; or closure is projective and both operands are ∞ (signs immaterial).	Return real *indefinite*
(Subtraction operations only): closure is affine and operands are like-signed infinities; or closure is projective and both operands are ∞ (signs immaterial).	Return real *indefinite*.
(Multiplication operations only): ∞ * 0; or 0 * ∞.	Return real *indefinite*.
(Division operations only): ∞ ÷ ∞; or 0 ÷ 0; or 0 ÷ pseudo-zero; or divisor is denormal or unnormal.	Return real *indefinite*.
(FPREM instruction only): modulus (divisor) is unnormal or denormal; or dividend is ∞.	Return real *indefinite*, set condition code = "complete remainder".
(FSQRT instruction only): operand is nonzero and negative; or operand is denormal or unnormal; or closure is affine and operand is −∞; or closure is projective and operand is ∞.	Return real *indefinite*.

Exception Conditions and Masked Responses (Cont'd.)

Invalid Operation	
(Compare operations only): closure is projective and ∞ is being compared with 0 or a normal, or ∞.	Set condition code = "not comparable"
(FTST instruction only): closure is projective and operand is ∞.	Set condition code = "not comparable".
(FIST, FISTP instructions only): source register is empty, or a NAN, or denormal, or unnormal, or ∞, or exceeds representable range of destination.	Store integer *indefinite*.
(FBSTP instruction only): source register is empty, or a NAN, or denormal, or unnormal, or ∞, or exceeds 18 decimal digits.	Store packed decimal *indefinite*.
(FST, FSTP instructions only): destination is short or long real and source register is an unnormal with exponent in range.	Store real *indefinite*.
(FXCH instruction only): one or both registers is tagged empty.	Change empty register(s) to real *indefinite* and then perform exchange.
Denormalized Operand	
(FLD instruction only): source operand is denormal.	No special action; load as usual.
(Arithmetic operations only): one or both operands is denormal.	Convert (in a work area) the operand to the equivalent unnormal and proceed.
(Compare and test operations only): one or both operands is denormal *or unnormal* (other than pseudo-zero).	Convert (in a work area) any denormal to the equivalent unnormal; normalize as much as possible, and proceed with operation.
Zerodivide	
(Division operations only): divisor = 0.	Return ∞ signed with "exclusive or" of operand signs.
Overflow	
(Arithmetic operations only): rounding is nearest or chop, and exponent of true result > 16,383.	Return properly signed ∞and signal precision exception.
(FST, FSTP instructions only): rounding is nearest or chop, and exponent of true result > +127 (short real destination) or > +1023 (long real destination).	Return properly signed ∞ and signal precision exception.

Exception Conditions and Masked Responses (Cont'd.)

Underflow	
(Arithmetic operations only): exponent of true result <−16,382 (true).	Denormalize until exponent rises to −16,382 (true), round significand to 64 bits. If denormalized rounded significand = 0, then return true 0; else, return denormal (tag = special, biased exponent =0).
(FST, FSTP instructions only): destination is short real and exponent of true result <− 126 (true).	Denormalize until exponent rises to −126 (true), round significand to 24 bits, store true 0 if denormalized rounded significand = 0; else, store denormal (biased exponent = 0).
(FST, FSTP instructions only): destination is long real and exponent of true result <−1022 (true).	Denormalize until exponent rises to −1022 (true), round significand to 53 bits, store true 0 if rounded denormalized significand = 0; else, store denormal (biased exponent = 0).
Precision	
True rounding error occurs.	No special action.
Masked response to overflow exception earlier in instruction.	No special action.

Table S-33. Masked Overflow Response for Directed Rounding

True Result		Rounding Mode	Result Delivered
Normalization	Sign		
Normal	+	Up	+∞
Normal	+	Down	Largest finite positive number(1)
Normal	−	Up	Largest finite negative number(1)
Normal	−	Down	−∞
Unnormal	+	Up	+∞
Unnormal	−	Down	Largest exponent, result's significand(2)
Unnormal	+	Up	Largest exponent, result's significand(2)
Unnormal	−	Down	−∞

(1) The largest valid representable reals are encoded:
exponent: 11...10B
significand: $(1)_{\Delta}$11...10B

(2) The significand retains its identity as an unnormal; the true result is rounded as usual (effectively chopped toward 0 in this case). The exponent is encoded 11...10B.

S.10 Programming Examples

Conditional Branching

As discussed in section S.7, the comparison instructions post their results to the condition code bits of the 8087 status word. Although there are many ways to implement conditional branching following a comparison, the basic approach is as follows:

- execute the comparison,
- store the status word,
- inspect the condition code bits,
- jump on the result.

Figure S-26 is a code fragment that illustrates how two memory-resident long real numbers might be compared (similar code could be used with the FTST instruction). The numbers are called A and B, and the comparison is A to B. The comparison itself simply requires loading A onto the top of the 8087 register stack and then comparing it to B and popping the stack in the same instruction. The status word is written to memory and the code waits for completion of the store before attempting to use the result.

There are four possible orderings of A and B, and bits C3 and C0 of the condition code indicate which ordering holds. These bits are positioned in the upper byte of the status word so as to correspond to the CPU's zero and carry flags (ZF and CF), if the byte is written into the flags (see figures 2-32 and S-6). The code fragment, then, sets ZF and CF to the values of C3 and C0 and then uses the CPU conditional jumps to test the flags. Table 2-15 shows how each conditional jump instruction tests the CPU flags.

The FXAM instruction updates all four condition code bits. Figure S-27 shows how a jump table can be used to determine the characteristics of the value examined. The jump table (FXAM_TBL) is initialized to contain the 16-bit displacement of 16 labels, one for each possible condition code setting. Note that four of the table entries contain the same value, since there are four condition code settings that correspond to "empty."

The program fragment performs the FXAM and stores the status word. It then manipulates the condition code bits to finally produce a number in register BX that equals the condition code times 2. This involves zeroing the unused bits in the byte that contains the code, shifting C3 to the right so that it is adjacent to C2, and then shifting the code to multiply it by 2. The resulting value is used as an index which selects one of the displacements from FXAM_TBL (the multiplication of the condition code is required because of the 2-byte length of each value in FXAM_TBL). The unconditional JMP instruction effectively vectors through the jump table to the labelled routine that contains code (not shown in the example) to process each possible result of the FXAM instruction.

```
            .
            .
            .
A           DQ      ?
B           DQ      ?
STAT_87     DW      ?
            .
            .
            .
            FLD     A           ;LOAD A ONTO TOP OF 87 STACK
            FCOMP   B           ;COMPARE A:B, POP A
            FSTSW   STAT_87     ;STORE RESULT
            FWAIT               ;WAIT FOR STORE
```

Figure S-26. Conditional Branching for Compares

```
        ;
        ;LOAD CPU REGISTER AH WITH BYTE OF
        ;    STATUS WORD CONTAINING CONDITION CODE
        MOV     AH, BYTE PTR STAT_87+1
        ;
        ;LOAD CONDITION CODES INTO CPU FLAGS
        SAHF
        ;
        ;USE CONDITIONAL JUMPS TO DETERMINE
        ;    ORDERING OF A AND B
        JB      A_LESS_OR_UNORDERED
        ;CF (C0) = 0
        JNE     A_GREATER
A_EQUAL:
        ;CF (C0) = 0, ZF (C3) = 1
        .
        .
        .
A_GREATER:
        ;CF (C0) = 0, ZF (C3) = 0
        .
        .
        .
A_LESS_OR_UNORDERED:
        ;CF (C0) = 1, TEST ZF (C3)
        JNE     A_LESS
A_B_UNORDERED:
        ;CF (C0) = 1, ZF (C3) = 1
        .
        .
        .
A_LESS:
        ;CF (C0) = 1, ZF (C3) = 0
        .
        .
        .
```

Figure S-26. Conditional Branching for Compares (Cont'd.)

```
        .
        .
        .
FXAM_TBL            DW POS_UNNORM, POS_NAN, NEG_UNNORM,
&                       NEG_NAN, POS_NORM, POS_INFINITY,
&                       NEG_NORM, NEG_INFINITY, POS_ZERO,
&                       EMPTY, NEG_ZERO, EMPTY, POS_DENORM,
&                       EMPTY, NEG_DENORM, EMPTY
STAT_87             DW ?
```

Figure S-27. Conditional Branching for FXAM

```
        .
        .
        .
    ;EXAMINE ST, STORE RESULT, WAIT FOR COMPLETION
        FXAM
        FSTSW       STAT_87
        FWAIT
    ;CLEAR UPPER HALF OF BX, LOAD CONDITION CODE
    ;   IN LOWER HALF
        MOV         BH,0
        MOV         BL, BYTE PTR STAT_87+1
    ;COPY ORIGINAL IMAGE
        MOV         AL,BL
    ;CLEAR ALL BITS EXCEPT C2-C0
        AND         BL,00000111B
    ;CLEAR ALL BITS EXCEPT C3
        AND         AL,01000000B
    ;SHIFT C3 TWO PLACES RIGHT
        SHR         AL,1
        SHR         AL,1
    ;SHIFT C2-C0 ONE PLACE LEFT (MULTIPLY BY 2)
        SAL         BX,1
    ;DROP C3 BACK IN ADJACENT TO C2 (000XXXX0)
        OR          BL,AL
    ;JUMP TO THE ROUTINE ''ADDRESSED'' BY CONDITION CODE
        JMP         FXAM_TBL[BX]
    ;
    ;HERE ARE THE JUMP TARGETS, ONE TO HANDLE
    ;   EACH POSSIBLE RESULT OF FXAM
POS_UNNORM:
        .
        .
POS_NAN:
        .
        .
NEG_UNNORM:
        .
        .
NEG_NAN:
        .
        .
POS_NORM:
        .
        .
POS_INFINITY:
        .
        .
NEG_NORM:
        .
        .
NEG_INFINITY:
        .
```

Figure S-27. Conditional Branching for FXAM (Cont'd.)

```
                .
POS_ZERO:
                .
                .
EMPTY:
                .
                .
NEG_ZERO:
                .
                .
POS_DENORM:
                .
                .
NEG_DENORM:
                .
                .
                .
```

Figure S-27. Conditional Branching for FXAM (Cont'd.)

Exception Handlers

There are many approaches to writing exception handlers. One useful technique is to consider the exception handler interrupt procedure as consisting of "prologue," "body" and "epilogue" sections of code. (For compatibility with the 8087 emulators, this procedure should be invoked by interrupt pointer (vector) number 16.)

At the beginning of the prologue, CPU interrupts have been disabled by the CPU's normal interrupt response mechanism. The prologue performs all functions that must be protected from possible interruption by higher-priority sources. Typically this will involve saving CPU registers and transferring diagnostic information from the 8087 to memory. When the critical processing has been completed, the prologue may enable CPU interrupts to allow higher-priority interrupt handlers to preempt the exception handler.

The exception handler body examines the diagnostic information and makes a response that is necessarily application-dependent. This response may range from halting execution, to displaying a message, to attempting to repair the problem and proceed with normal execution.

The epilogue essentially reverses the actions of the prologue, restoring the CPU and the NDP so that normal execution can be resumed. The epilogue must *not* load an unmasked exception flag into the 8087 or another interrupt will be requested immediately (assuming 8087 interrupts are also loaded as unmasked).

Figures S-28 through S-30 show the ASM-86 coding of three skeleton exception handlers. They show how prologues and epilogues can be written for various situations, but only provide comments indicating where the application-dependent exception handling body should be placed.

Figures S-28 and S-29 are very similar; their only substantial difference is their choice of instructions to save and restore the 8087. The tradeoff here is between the increased diagnostic information provided by FNSAVE and the faster execution of FNSTENV. For applications that are sensitive to interrupt latency, or do not need to examine register contents, FNSTENV reduces the duration of the "critical region," during which the CPU will not recognize another interrupt request (unless it is a non-maskable interrupt).

After the exception handler body, the epilogues prepare the CPU and the NDP to resume execution from the point of interruption (i.e., the instruction following the one that generated the unmasked exception). Notice that the exception flags in the memory image that is loaded into the 8087 are cleared to zero prior to reloading (in fact, in these examples, the entire status word

image is cleared). The prologue also provides for indicating to the interrupt controller hardware (e.g., 8259A) that the interrupt has been processed. The actual processing done here is application-dependent, but might typically involve writing an "end of interrupt" command to the interrupt controller.

The examples in figures S-28 and S-29 assume that the exception handler itself will not cause an unmasked exception. Where this is a possibility, the general approach shown in figure S-30 can be employed. The basic technique is to save the full 8087 state and then to load a new control word in the prologue. Note that considerable care should be taken when designing an exception handler of this type to prevent the handler from being reentered endlessly.

```
SAVE_ALL          PROC
;
;SAVE CPU REGISTERS, ALLOCATE STACK SPACE
;FOR 8087 STATE IMAGE
    PUSH      BP
        .
        .
        .
    MOV       BP,SP
    SUB       SP,94
;SAVE FULL 8087 STATE, WAIT FOR COMPLETION,
;ENABLE CPU INTERRUPTS
    FNSAVE    [BP-94]
    FWAIT
    STI
;
;APPLICATION-DEPENDENT EXCEPTION HANDLING
;CODE GOES HERE
;
;CLEAR EXCEPTION FLAGS IN STATUS WORD
;RESTORE MODIFIED STATE
;IMAGE
    MOV       BYTE PTR [BP-92], 0H
    FRSTOR    [BP-94]
;WAIT FOR RESTORE TO FINISH BEFORE RELEASING MEMORY
    FWAIT
;DE-ALLOCATE STACK SPACE, RESTORE CPU REGISTERS
    MOV       SP,BP
        .
        .
        .
    POP       BP
;
;CODE TO SEND ''END OF INTERRUPT'' COMMAND TO
;8259A GOES HERE
;
;RETURN TO INTERRUPTED CALCULATION
    IRET
SAVE_ALL          ENDP
```

Figure S-28. Full State Exception Handler

```
SAVE_ENVIRONMENT PROC
;
;SAVE CPU REGISTERS, ALLOCATE STACK SPACE
;FOR 8087 ENVIRONMENT
    PUSH    BP
        .
        .
        .
    MOV     BP,SP
    SUB     SP,14
;SAVE ENVIRONMENT, WAIT FOR COMPLETION,
;ENABLE CPU INTERRUPTS
    FNSTENV [BP-14]
    FWAIT
    STI
;
;APPLICATION EXCEPTION-HANDLING CODE GOES HERE
;
;CLEAR EXCEPTION FLAGS IN STATUS WORD
;RESTORE MODIFIED
;ENVIRONMENT IMAGE
    MOV     BYTE PTR [BP-12], 0H
    FLDENV  [BP-14]
;WAIT FOR LOAD TO FINISH BEFORE RELEASING MEMORY
    FWAIT
;DE-ALLOCATE STACK SPACE, RESTORE CPU REGISTERS
    MOV     SP,BP
        .
        .
        .
    POP     BP
;
;CODE TO SEND ''END OF INTERRUPT'' COMMAND TO
;8259A GOES HERE
;
;RETURN TO INTERRUPTED CALCULATION
    IRET
SAVE_ENVIRONMENT ENDP
```

Figure S-29. Reduced Latency Exception Handler

```
            .
            .
            .
      LOCAL_CONTROL  DW  ?  ;ASSUME INITIALIZED
            .
            .
            .
REENTRANT              PROC
;
;SAVE CPU REGISTERS, ALLOCATE STACK SPACE FOR
;8087 STATE IMAGE
    PUSH      BP
        .
        .
        .
    MOV       BP,SP
    SUB       SP,94
;SAVE STATE, LOAD NEW CONTROL WORD, WAIT
;FOR COMPLETION, ENABLE CPU INTERRUPTS
    FNSAVE    [BP-94]
    FLDCW     LOCAL_CONTROL
    FWAIT
    STI
;CODE TO SEND ''END OF INTERRUPT'' COMMAND TO
;8259A GOES HERE
        .
        .
        .
;APPLICATION EXCEPTION HANDLING CODE GOES HERE.
;AN UNMASKED EXCEPTION GENERATED HERE WILL
;CAUSE THE EXCEPTION HANDLER TO BE REENTERED.
;IF LOCAL STORAGE IS NEEDED, IT MUST BE
;ALLOCATED ON THE CPU STACK.
        .
        .
        .
;CLEAR EXCEPTION FLAGS IN STATUS WORD
;RESTORE MODIFIED STATE IMAGE
    MOV       BYTE PTR [BP-92], 0H
    FRSTOR    [BP-94]
;WAIT FOR RESTORE TO FINISH BEFORE RELEASING MEMORY
    FWAIT
;DE-ALLOCATE STACK SPACE, RESTORE CPU REGISTERS
    MOV       SP,BP
        .
        .
        .
    POP       BP
;RETURN TO POINT OF INTERRUPTION
    IRET
REENTRANT              ENDP
```

Figure S-30. Reentrant Exception Handler

Appendix A
Machine Instruction Encoding and Decoding

APPENDIX A
MACHINE INSTRUCTION ENCODING AND DECODING

8087 machine instructions assume one of five different forms as shown in table A-1. In all cases, the instructions are at least two bytes long and begin with the bit pattern 11011B, which identifies the escape class of instructions. Instructions which reference memory operands are encoded much like similar CPU instructions, since the CPU must calculate the operands effective address from the information contained in the instruction. Section 4.2 discusses this encoding scheme in more detail, and in particular, shows how each memory addressing mode is encoded.

Note that all instructions (except those coded with a "no-wait" mnemonic) are preceded by an assembler-generated CPU WAIT instruction (encoding: 10011011B). Segment override prefixes may also precede 8087 instructions in the instruction stream.

Table A-1. Instruction Encoding

	Lower-addressed Byte								Higher-addressed Byte								0, 1, or 2 bytes
(1)	1	1	0	1	1	OP-A		1	MOD		1	OP-B		R/M			DISPLACEMENT
(2)	1	1	0	1	1	FORMAT		OP-A	MOD		OP-B			R/M			DISPLACEMENT
(3)	1	1	0	1	1	R	P	OP-A	1	1	OP-B			REG			
(4)	1	1	0	1	1	0	0	1	1	1	1	OP					
(5)	1	1	0	1	1	0	1	1	1	1	1	OP					
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	

(1)Memory transfers, including applicable processor control instructions; 0, 1, or 2 displacement bytes may follow.

(2)Memory arithmetic and comparison instructions; 0, 1, or 2 displacement bytes may follow.

(3)Stack arithmetic and comparison instructions.

(4)Constant, transcendental, some arithmetic instructions.

(5)Processor control instructions that do not reference memory.

OP, OP-A, OP-B: Instruction opcode, possibly split into two fields.

MOD: Same as CPU mode field; see table 4-8.

R/M: Sames as CPU register/memory field; see table 4-10.

Table A-1. Instruction Encoding (Cont'd.)

FORMAT: Defines memory operand
- 00 = short real
- 01 = short integer
- 10 = long real
- 11 = word integer

R: 0 = return result to stack top
1 = return result to other register

P: 0 = do not pop stack
1 = pop stack after operation

REG: register stack element
- 000 = stack top
- 001 = next on stack
- 010 = third stack element, etc.

Table A-2 lists all 8087 machine instructions in binary sequence. This table may be used to "disassemble" instructions in unformatted memory dumps or instructions monitored from the data bus. Users writing exception handlers may also find this information useful to identify the offending instruction.

Table A-2. Machine Instruction Decoding Guide

1st Byte		2nd Byte	Bytes 3, 4	ASM-86 Instruction Format	
Hex	Binary				
D8	1101 1000	MOD00 0R/M	(disp-lo),(disp-hi)	FADD	short-real
D8	1101 1000	MOD00 1R/M	(disp-lo),(disp-hi)	FMUL	short-real
D8	1101 1000	MOD01 0R/M	(disp-lo),(disp-hi)	FCOM	short-real
D8	1101 1000	MOD01 1R/M	(disp-lo),(disp-hi)	FCOMP	short-real
D8	1101 1000	MOD10 0R/M	(disp-lo),(disp-hi)	FSUB	short-real
D8	1101 1000	MOD10 1R/M	(disp-lo),(disp-hi)	FSUBR	short-real
D8	1101 1000	MOD11 0R/M	(disp-lo),(disp-hi)	FDIV	short-real
D8	1101 1000	MOD11 1R/M	(disp-lo),(disp-hi)	FDIVR	short-real
D8	1101 1000	1100 0REG		FADD	ST,ST(i)
D8	1101 1000	1100 1REG		FMUL	ST,ST(i)
D8	1101 1000	1101 0REG		FCOM	ST(i)
D8	1101 1000	1101 1REG		FCOMP	ST(i)
D8	1101 1000	1110 0REG		FSUB	ST,ST(i)
D8	1101 1000	1110 1REG		FSUBR	ST,ST(i)
D8	1101 1000	1111 0REG		FDIV	ST,ST(i)
D8	1101 1000	1111 1REG		FDIVR	ST,ST(i)
D9	1101 1001	MOD00 0R/M	(disp-lo),(disp-hi)	FLD	short-real
D9	1101 1001	MOD00 1R/M		reserved	
D9	1101 1001	MOD01 0R/M	(disp-lo),(disp-hi)	FST	short-real

Table A-2. Machine Instruction Decoding Guide (Cont'd.)

1st Byte		2nd Byte	Bytes 3,4	ASM-86 Instruction Format	
Hex	Binary				
D9	1101 1001	MOD01 1R/M	(disp-lo),(disp-hi)	FSTP	short-real
D9	1101 1001	MOD10 0R/M	(disp-lo),(disp-hi)	FLDENV	14-bytes
D9	1101 1001	MOD10 1R/M	(disp-lo),(disp-hi)	FLDCW	2-bytes
D9	1101 1001	MOD11 0R/M	(disp-lo),(disp-hi)	FSTENV	14-bytes
D9	1101 1001	MOD11 1R/M	(disp-lo),(disp-hi)	FSTCW	2-bytes
D9	1101 1001	1100 0REG		FLD	ST(i)
D9	1101 1001	1100 1REG		FXCH	ST(i)
D9	1101 1001	1101 0000		FNOP	
D9	1101 1001	1101 0001		reserved	
D9	1101 1001	1101 001-		reserved	
D9	1101 1001	1101 01--		reserved	
D9	1101 1001	1101 1REG		*(1)	
D9	1101 1001	1110 0000		FCHS	
D9	1101 1001	1110 0001		FABS	
D9	1101 1001	1110 001-		reserved	
D9	1101 1001	1110 0100		FTST	
D9	1101 1001	1110 0101		FXAM	
D9	1101 1001	1110 011-		reserved	
D9	1101 1001	1110 1000		FLD1	
D9	1101 1001	1110 1001		FLDL2T	
D9	1101 1001	1110 1010		FLDL2E	
D9	1101 1001	1110 1011		FLDPI	
D9	1101 1001	1110 1100		FLDLG2	
D9	1101 1001	1110 1101		FLDLN2	
D9	1101 1001	1110 1110		FLDZ	
D9	1101 1001	1110 1111		reserved	
D9	1101 1001	1111 0000		F2XM1	
D9	1101 1001	1111 0001		FYL2X	
D9	1101 1001	1111 0010		FPTAN	
D9	1101 1001	1111 0011		FPATAN	
D9	1101 1001	1111 0100		FXTRACT	
D9	1101 1001	1111 0101		reserved	
D9	1101 1001	1111 0110		FDECSTP	
D9	1101 1001	1111 0111		FINCSTP	
D9	1101 1001	1111 1000		FPREM	
D9	1101 1001	1111 1001		FYL2XP1	
D9	1101 1001	1111 1010		FSQRT	
D9	1101 1001	1111 1011		reserved	
D9	1101 1001	1111 1100		FRNDINT	
D9	1101 1001	1111 1101		FSCALE	
D9	1101 1001	1111 111-		reserved	
DA	1101 1010	MOD00 0R/M	(disp-lo),(disp-hi)	FIADD	short-integer
DA	1101 1010	MOD00 1R/M	(disp-lo),(disp-hi)	FIMUL	short-integer
DA	1101 1010	MOD01 0R/M	(disp-lo),(disp-hi)	FICOM	short-integer
DA	1101 1010	MOD01 1R/M	(disp-lo),(disp-hi)	FICOMP	short-integer
DA	1101 1010	MOD10 0R/M	(disp-lo),(disp-hi)	FISUB	short-integer
DA	1101 1010	MOD10 1R/M	(disp-lo),(disp-hi)	FISUBR	short-integer

Table A-2. Machine Instruction Decoding Guide (Cont'd.)

1st Byte		2nd Byte		Bytes 3,4	ASM-86 Instruction Format	
Hex	Binary					
DA	1101 1010	MOD11	0R/M	(disp-lo),(disp-hi)	FIDIV	short-integer
DA	1101 1010	MOD11	1R/M	(disp-lo),(disp-hi)	FIDIVR	short-integer
DA	1101 1010	11--	----		reserved	
DB	1101 1011	MOD00	0R/M	(disp-lo),(disp-hi)	FILD	short-integer
DB	1101 1011	MOD00	1R/M	(disp-lo),(disp-hi)	reserved	
DB	1101 1011	MOD01	0R/M	(disp-lo),(disp-hi)	FIST	short-integer
DB	1101 1011	MOD01	1R/M	(disp-lo),(disp-hi)	FISTP	short-integer
DB	1101 1011	MOD10	0R/M	(disp-lo),(disp-hi)	reserved	
DB	1101 1011	MOD10	1R/M	(disp-lo),(disp-hi)	FLD	temp-real
DB	1101 1011	MOD11	0R/M	(disp-lo),(disp-hi)	reserved	
DB	1101 1011	MOD11	1R/M	(disp-lo),(disp-hi)	FSTP	temp-real
DB	1101 1011	110-	----		reserved	
DB	1101 1011	1110	0000		FENI	
DB	1101 1011	1110	0001		FDISI	
DB	1101 1011	1110	0010		FCLEX	
DB	1101 1011	1110	0011		FINIT	
DB	1101 1011	1110	01--		reserved	
DB	1101 1011	1110	1---		reserved	
DB	1101 1011	1111	----		reserved	
DC	1101 1100	MOD00	0R/M	(disp-lo),(disp-hi)	FADD	long-real
DC	1101 1100	MOD00	1R/M	(disp-lo),(disp-hi)	FMUL	long-real
DC	1101 1100	MOD01	0R/M	(disp-lo),(disp-hi)	FCOM	long-real
DC	1101 1100	MOD01	1R/M	(disp-lo),(disp-hi)	FCOMP	long-real
DC	1101 1100	MOD10	0R/M	(disp-lo),(disp-hi)	FSUB	long-real
DC	1101 1100	MOD10	1R/M	(disp-lo),(disp-hi)	FSUBR	long-real
DC	1101 1100	MOD11	0R/M	(disp-lo),(disp-hi)	FDIV	long-real
DC	1101 1100	MOD11	1R/M	(disp-lo),(disp-hi)	FDIVR	long-real
DC	1101 1100	1100	0REG		FADD	ST(i),ST
DC	1101 1100	1100	1REG		FMUL	ST(i),ST
DC	1101 1100	1101	0REG		*(2)	
DC	1101 1100	1101	1REG		*(3)	
DC	1101 1100	1110	0REG		FSUB	ST(i),ST
DC	1101 1100	1110	1REG		FSUBR	ST(i),ST
DC	1101 1100	1111	0REG		FDIV	ST(i),ST
DC	1101 1100	1111	1REG		FDIVR	ST(i),ST
DD	1101 1101	MOD00	0R/M	(disp-lo),(disp-hi)	FLD	long-real
DD	1101 1101	MOD00	1R/M		reserved	
DD	1101 1101	MOD01	0R/M	(disp-lo),(disp-hi)	FST	long-real
DD	1101 1101	MOD01	1R/M	(disp-lo),(disp-hi)	FSTP	long-real
DD	1101 1101	MOD10	0R/M	(disp-lo),(disp-hi)	FRSTOR	94-bytes
DD	1101 1101	MOD10	1R/M	(disp-lo),(disp-hi)	reserved	
DD	1101 1101	MOD11	0R/M	(disp-lo),(disp-hi)	FSAVE	94-bytes
DD	1101 1101	MOD11	1R/M	(disp-lo),(disp-hi)	FSTSW	2-bytes
DD	1101 1101	1100	0REG		FFREE	ST(i)
DD	1101 1101	1100	1REG		*(4)	
DD	1101 1101	1101	0REG		FST	ST(i)
DD	1101 1101	1101	1REG		FSTP	ST(i)

Table A-2. Machine Instruction Decoding Guide (Cont'd.)

1st Byte		2nd Byte		Bytes 3,4	ASM-86 Instruction Format	
Hex	Binary					
DD	1101 1101	111-	----		reserved	
DE	1101 1110	MOD00	0R/M	(disp-lo),(disp-hi)	FIADD	word-integer
DE	1101 1110	MOD00	1R/M	(disp-lo),(disp-hi)	FIMUL	word-integer
DE	1101 1110	MOD01	0R/M	(disp-lo),(disp-hi)	FICOM	word-integer
DE	1101 1110	MOD01	1R/M	(disp-lo),(disp-hi)	FICOMP	word-integer
DE	1101 1110	MOD10	0R/M	(disp-lo),(disp-hi)	FISUB	word-integer
DE	1101 1110	MOD10	1R/M	(disp-lo),(disp-hi)	FISUBR	word-integer
DE	1101 1110	MOD11	0R/M	(disp-lo),(disp-hi)	FIDIV	word-integer
DE	1101 1110	MOD11	1R/M	(disp-lo),(disp-hi)	FIDIVR	word-integer
DE	1101 1110	1100	0REG		FADDP	ST(i),ST
DE	1101 1110	1100	1REG		FMULP	ST(i),ST
DE	1101 1110	1101	0---		*(5)	
DE	1101 1110	1101	1000		reserved	
DE	1101 1110	1101	1001		FCOMPP	
DE	1101 1110	1101	101-		reserved	
DE	1101 1110	1101	11--		reserved	
DE	1101 1110	1110	0REG		FSUBP	ST(i),ST
DE	1101 1110	1110	1REG		FSUBRP	ST(i),ST
DE	1101 1110	1111	0REG		FDIVP	ST(i),ST
DE	1101 1110	1111	1REG		FDIVRP	ST(i),ST
DF	1101 1111	MOD00	0R/M	(disp-lo),(disp-hi)	FILD	word-integer
DF	1101 1111	MOD00	1R/M	(disp-lo),(disp-hi)	reserved	
DF	1101 1111	MOD01	0R/M	(disp-lo),(disp-hi)	FIST	word-integer
DF	1101 1111	MOD01	1R/M	(disp-lo),(disp-hi)	FISTP	word-integer
DF	1101 1111	MOD10	0R/M	(disp-lo),(disp-hi)	FBLD	packed-decimal
DF	1101 1111	MOD10	1R/M	(disp-lo),(disp-hi)	FILD	long-integer
DF	1101 1111	MOD11	0R/M	(disp-lo),(disp-hi)	FBSTP	packed-decimal
DF	1101 1111	MOD11	1R/M	(disp-lo),(disp-hi)	FISTP	long-integer
DF	1101 1111	1100	0REG		*(6)	
DF	1101 1111	1100	1REG		*(7)	
DF	1101 1111	1101	0REG		*(8)	
DF	1101 1111	1101	1REG		*(9)	
DF	1101 1111	111-	----		reserved	

* The marked encodings are *not* generated by the language translators. If, however, the 8087 encounters one one these encodings in the instruction stream, it will execute it as follows:

(1) FSTP ST(i)

(2) FCOM ST(i)

(3) FCOMP ST(i)

(4) FXCH ST(i)

(5) FCOMP ST(i)

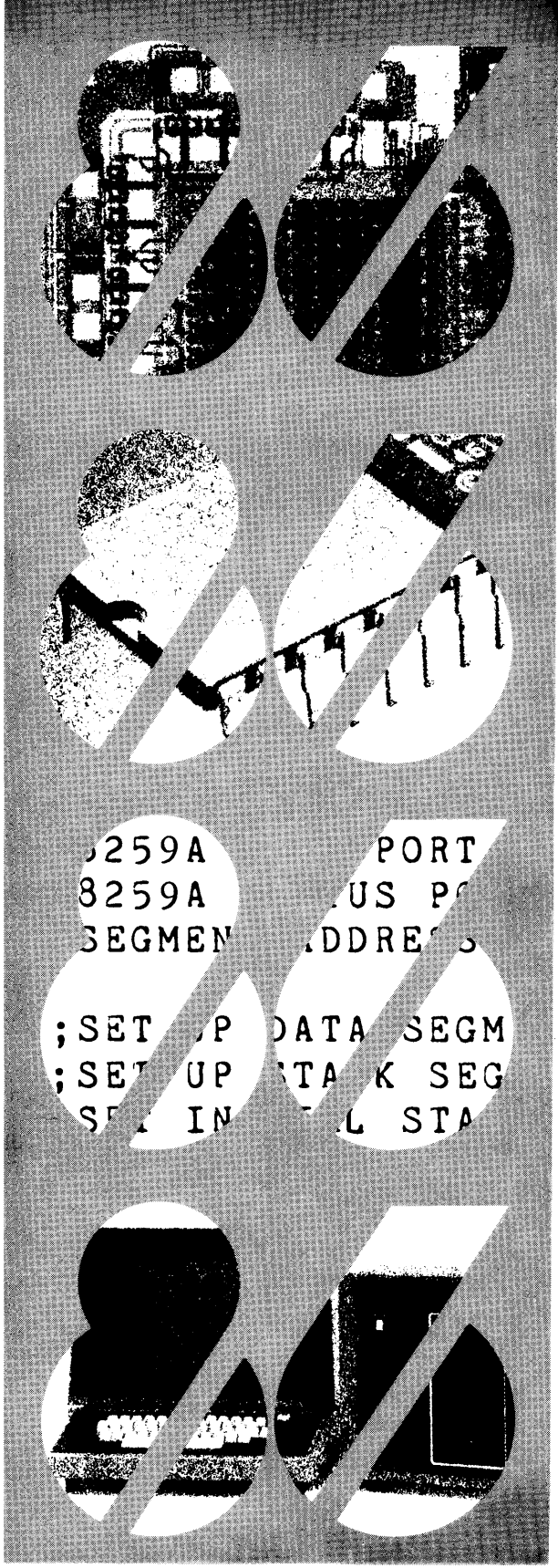
(6) FFREE ST(i) and pop stack

(7) FXCH ST(i)

(8) FSTP ST(i)

(9) FSTP ST(i)

Appendix B
Device Specification



8087
80-BIT HMOS
NUMERIC DATA PROCESSOR

- **Full Internal 80-Bit Architecture for High Performance**
- **Implements Proposed IEEE Floating Point Standard**
- **Total Numeric Support for iAPX 86/20, 88/20 Systems**
- **Expands iAPX 86/10 Datatypes to Include 32-, 64-Bit Integers, 32-, 64-, 80-Bit Floating Point, and 18-Digit BCD Operands**
- **All iAPX 86/10, 88/10 Addressing Modes Available**
- **Directly Extends iAPX 86/10, 88/10 Instruction Set to Trigonometric, Logarithmic, Exponential and Arithmetic Instructions for All Datatypes**
- **8 x 80-Bit, Individually Addressable, Numeric Register Stack**
- **Built-in Exception Handling Functions**

The Intel® 8087 is a high performance coprocessor that extends the iAPX 86/10, 88/10 architecture by providing all the required numerics support for iAPX 86/20, 88/20 systems. The 8087 is implemented in N-channel, depletion load, silicon gate technology (HMOS) and packaged in a 40-pin ceramic package. The iAPX 86/20 and 88/20 fully conform to the proposed IEEE Floating Point Standard. Using its coprocessor architecture, the 8087 adds over fifty opcodes directly into the iAPX 86/10 instruction set, making the iAPX 86/20, 88/20 a complete solution to high performance numeric processing.

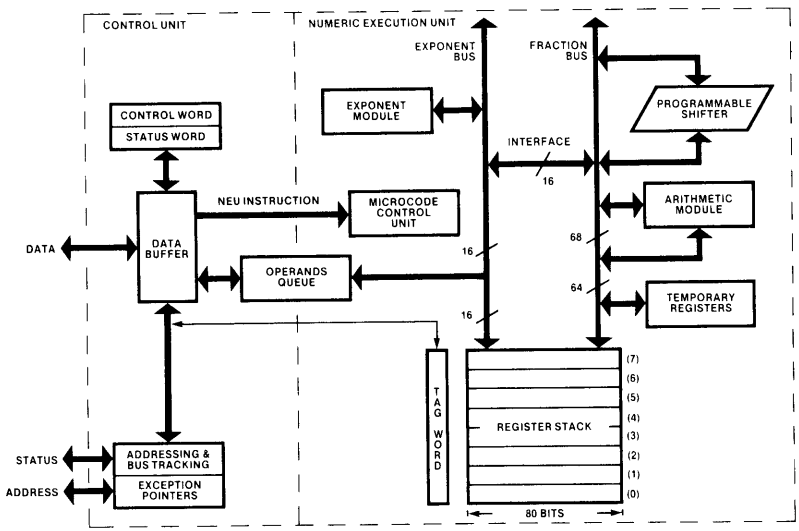


Figure 1. 8087 Block Diagram

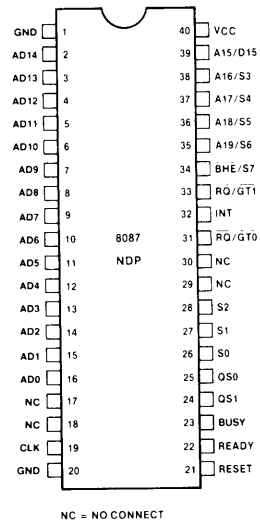


Figure 2. 8087 Pin Diagram

FUNCTIONAL DESCRIPTION

The 8087 Numeric Data Processor (NDP) is a processor extension that provides arithmetic and logical instruction support for a variety of numeric data types in iAPX 86/20, 88/20 systems. It also executes numerous built-in transcendental functions (e.g., tangent and log functions). The 8087 executes instructions as a coprocessor to a maximum mode 8086 or 8088. It effectively extends the register and instruction set of an iAPX 86/10 or 88/10 system for existing iAPX 86, 88 types and adds several new data types as well. Figure 3 presents this view of the iAPX 86/20 graphically. Essentially, the 8087 can be treated as an additional resource and an extension to the iAPX 86/10 or 88/10, providing register, datatype, control, and instruction capabilities at the hardware level that can be used as a single unified system, the iAPX 86/20, 88/20, at the programming level.

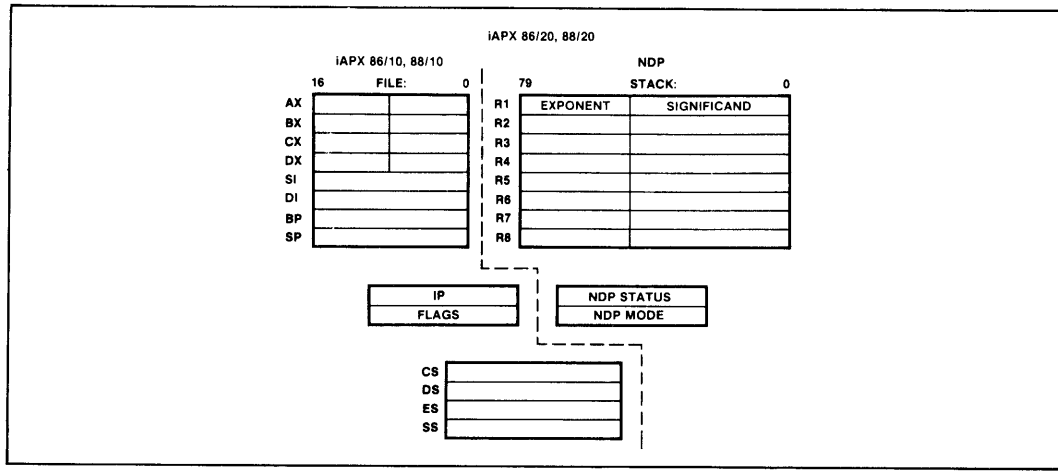


Figure 3. iAPX 86/20 Architecture

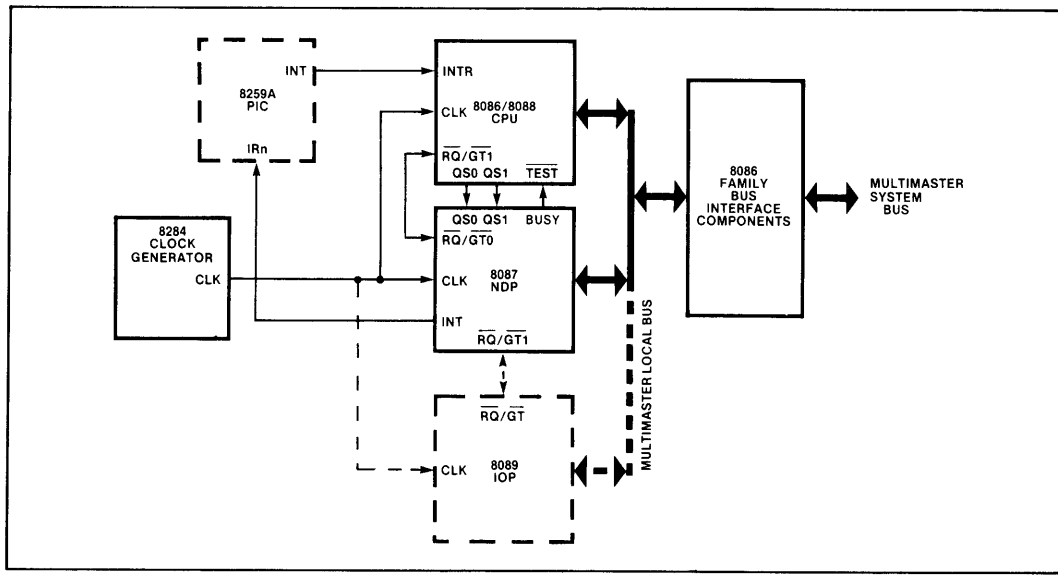


Figure 4. 8087 System Configuration

System Configuration

As a coprocessor to an 8086 or 8088, the 8087 is wired in parallel with the CPU as shown in Figure 4. The CPU's status ($\overline{S0}$-$\overline{S2}$) and queue status lines (QS0-QS1) enable the NDP to monitor and decode instructions in synchronization with the CPU and without any CPU overhead. Once started the 8087 can process in parallel with and independent of the host CPU. For resynchronization, the NDP's BUSY signal informs the CPU that the NDP is executing an instruction and the CPU WAIT instruction tests this signal to insure that the NDP is ready to execute subsequent instructions. The NDP can interrupt the CPU when it detects an error or exception. The NDP's interrupt request line is typically routed to the CPU through an 8259A Programmable Interrupt Controller. (See Figure 2 for 8087 pinout information.)

The NDP uses one of the request/grant lines of the iAPX 86, 88 architecture (typically $\overline{RQ}/\overline{GT1}$) to obtain control of the local bus for data transfers. The other request/grant line is available for general system use (for instance by an I/O processor in LOCAL mode). A bus master can also be connected to the 8087's $\overline{RQ}/\overline{GT1}$ line. In this configuration the 8087 will pass the request/grant handshake signals between the CPU and the attached master when the 8087 is not in control of the bus and will relinquish the bus to the master directly when the 8087 is in control. In this way two additional masters can be configured in an 8086/8087 system; one will share the 8086 bus with the 8087 on a first come first served basis, and the second will be guaranteed to be higher in priority than the 8087.

As Figure 4 shows, all processors utilize the same clock generator and system bus interface components (bus controller, latches, transceivers and bus arbiter).

Bus Operation

The 8087 bus structure, operation and timing are identical to all other processors in the iAPX 86, 88 series (maximum mode configuration). The address is time multiplexed with the data on the first 16/8 lines of the address/data bus. A16 through A19 are time multiplexed

Table 1. 8087 Datatypes

Data Formats	Range	Precision	Most Significant Byte: 7 07 07 07 07 07 07 07 07 07 0
Word Integer	10^4	16 Bits	I_{15} ... I_0 — Two's Complement
Short Integer	10^9	32 Bits	I_{31} ... I_0 — Two's Complement
Long Integer	10^{19}	64 Bits	I_{63} ... I_0 — Two's Complement
Packed BCD	10^{18}	18 Digits	S — D_{17} D_{16} ... D_1 D_0
Short Real	$10^{\pm 38}$	24 Bits	S E_7 E_0 F_1 ... F_{23} — F_0 Implicit
Long Real	$10^{\pm 308}$	53 Bits	S E_{10} E_0 F_1 ... F_{52} — F_0 Implicit
Temporary Real	$10^{\pm 4932}$	64 Bits	S E_{14} E_0 F_0 ... F_{63}

Integer: I

Packed BCD: $(-1)^S(D_{17}\ldots D_0)$

Real: $(-1)^S(2^{E-BIAS})(F_0 \bullet F_1\ldots)$

Bias = 127 for Short Real
1023 for Long Real
16383 for Temp Real

with four status lines S3-S6. S3, S4 and S6 are always one (high) for 8087 driven bus cycles while S5 is always zero (low). When the 8087 is monitoring CPU bus cycles (passive mode) S6 is also monitored by the 8087 to differentiate 8086/8088 activity from that of a local I/O processor or any other local bus master. (The 8086/8088 must be the only processor on the local bus to drive S6 low.) S7 is multiplexed with and has the same value as $\overline{BHE}$ for all 8087 bus cycles.

The first three status lines, $\overline{S0}$-$\overline{S2}$, are used with an 8288 bus controller to determine the type of bus cycle being run:

$\overline{S2}$	$\overline{S1}$	$\overline{S0}$	
0	X	X	Unused
1	0	0	Unused
1	0	1	Memory Data Read
1	1	0	Memory Data Write
1	1	1	Passive (no bus cycle)

Programming Interface

Table 1 lists the seven data types the 8087 supports and presents the format for each type. Internally, the 8087 holds all numbers in the temporary real format. Load and store instructions automatically convert operands represented in memory as 16-, 32-, or 64-bit integers, 32- or 64-bit floating point numbers or 18-digit packed BCD numbers into temporary real format and vice versa.

Computations in the 8087 use the processor's register stack. These eight 80-bit registers provide the equivalent capacity of 40 16-bit registers. The 8087 register set can be accessed as a stack, with instructions operating on the top one or two stack elements, or as a fixed register set, with instructions operating on explicitly designated registers.

Table 5 lists the 8087's instructions by class. Assembly language programs are written in ASM-86, the iAPX 86, 88 assembly language. Table 2 gives the execution times of some typical numeric instructions and their equivalent time on a 5 MHz 8086.

Table 2. Execution Time for Selected 8087 Actual and Emulated Instructions

Floating Point Instruction	Approximate Execution Time (μs)	
	8087 (5 MHz Clock)	8086 Emulation
Add/Subtract Magnitude	14/18	1,600
Multiply (single precision)	19	1,600
Multiply (extended precision)	27	2,100
Divide	39	3,200
Compare	9	1,300
Load (double precision)	10	1,700
Store (double precision)	21	1,200
Square Root	36	19,600
Tangent	90	13,000
Exponentiation	100	17,100

PROCESSOR ARCHITECTURE

As shown in Figure 1, the NDP is internally divided into two processing elements, the control unit (CU) and the numeric execution unit (NEU). The NEU executes all numeric instructions, while the CU receives and decodes instructions, reads and writes memory operands and executes processor control instructions. The two elements are able to operate independently of one another, allowing the CU to maintain synchronization with the CPU while the NEU is busy processing a numeric instruction.

Control Unit

The CU keeps the 8087 operating in synchronization with its host CPU. 8087 instructions are intermixed with CPU instructions in a single instruction stream. The CPU fetches all instructions from memory; by monitoring the status signals ($\overline{S0}$-$\overline{S2}$, S6) emitted by the CPU, the NDP control unit determines when an 8086 instruction is being fetched. The CU taps the bus in parallel with the CPU and obtains that portion of the data stream.

The CU maintains an instruction queue that is identical to the queue in the host CPU. The CU automatically determines if the CPU is an 8086 or an 8088 immediately after reset (by monitoring the $\overline{BHE}$/S7 line) and matches its queue length accordingly. By monitoring the CPU's queue status lines (QS0, QS1), the CU obtains and decodes instructions from the queue in synchronization with the CPU.

After decoding the instruction, the 8086 executes all opcodes but ESCAPE (ESC), while the 8087 executes only the ESCAPE class instructions. (The first five bits of all ESCAPE instructions are identical.) The CPU does provide addressing for ESC instructions, however.

The CPU distinguishes between ESC instructions that reference memory and those that do not. If the instruction refers to a memory operand, the CPU calculates the operand's address using any one of its available addressing modes, and then performs a "dummy read" of the word at that location. (Any location within the 1M byte address space is allowed.) This is a normal read cycle except that the CPU ignores the data it receives. If the ESC instruction does not contain a memory reference (e.g., an 8087 stack operation), the CPU simply proceeds to the next instruction.

An 8087 instruction either will not reference memory, will require loading one or more operands from memory into the 8087, or will require storing one or more operands from the 8087 into memory. In the first case a non-memory reference escape is used to start 8087 operation. In the last two cases, the CU makes use of the "dummy read" cycle initiated by the CPU. The CU captures and saves the address which the CPU places on the bus. If the instruction is a load, the CU additionally captures the data word when it becomes available on the local data bus. If data required is longer than one word, the CU immediately obtains the bus from the CPU using the request/grant protocol and reads the rest of

the information in consecutive bus cycles. In a store operation, the CU captures and saves the store address as in a load, and ignores the data word that follows in the "dummy read" cycle. When the 8087 is ready to perform the store, the CU obtains the bus from the CPU and writes the operand starting at the specified address.

Numeric Execution Unit

The NEU executes all instructions that involve the register stack; these include arithmetic, logical, transcendental, constant and data transfer instructions. The data path in the NEU is 84 bits wide (68 fraction bits, 15 exponent bits and a sign bit) which allows internal operand transfers to be performed at very high speeds.

When the NEU begins executing an instruction, it activates the 8087 BUSY signal. This signal can be used in conjunction with the CPU WAIT instruction to resynchronize both processors when the NEU has completed its current instruction.

Register Set

The 8087 register set is shown in Figure 5. Each of the eight data registers in the 8087's register stack is 80 bits wide and is divided into "fields" corresponding to the NDP's temporary real data type.

At a given point in time the TOP field in the control word identifies the current top-of-stack register. A "push" operation decrements TOP by 1 and loads a value into the new top register. A "pop" operation stores the value from the current top register and then increments TOP by 1. Like 8086/8088 stacks in memory, the 8087 register stack grows "down" toward lower-addressed registers.

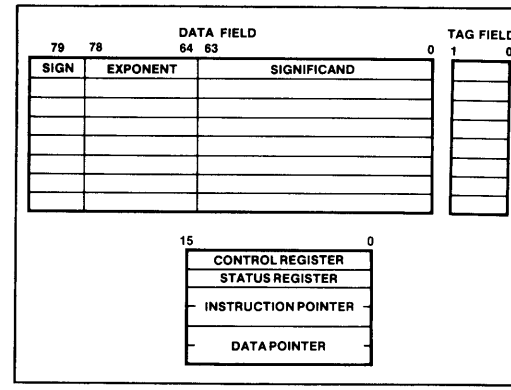


Figure 5. 8087 Register Set

Instructions may address the data registers either implicitly or explicitly. Many instructions operate on the register at the top of the stack. These instructions implicitly address the register pointed to by the TOP. Other instructions allow the programmer to explicitly specify the register which is to be used. Explicit register addressing is "top-relative."

Status Word

The status word shown in Figure 6 reflects the overall state of the 8087; it may be stored in memory and then inspected by CPU code. The status word is a 16-bit register divided into fields as shown in Figure 6. The busy bit (bit 15) indicates whether the *NEU* is executing an instruction (B = 1) or is idle (B = 0). Several instruc-

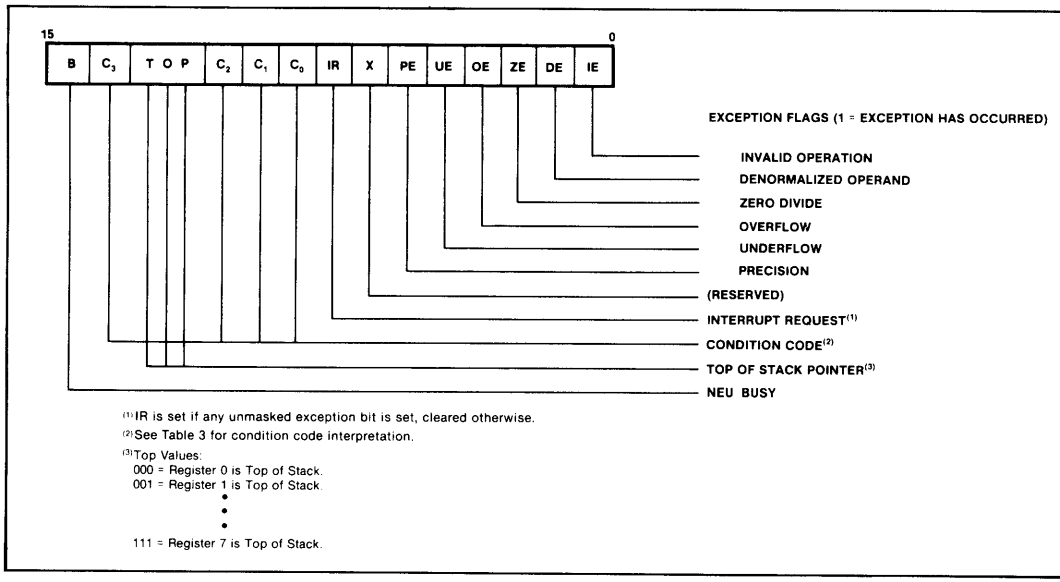


Figure 6. 8087 Status Word

tions which store and manipulate the status word are executed exclusively by the CU, and these do not set the busy bit themselves.

The four numeric condition code bits (C_0-C_3) are similar to the flags in a CPU: various instructions update these bits to reflect the outcome of NDP operations. The effect of these instructions on the condition code bits is summarized in Table 3.

Bits 14-12 of the status word point to the 8087 register that is the current top-of-stack (TOP) as described above.

Bit 7 is the interrupt request bit. This bit is set if any unmasked exception bit is set and cleared otherwise.

Bits 5-0 are set to indicate that the NEU has detected an exception while executing an instruction.

Tag Word

The tag word marks the content of each register as shown in Figure 7. The principal function of the tag word is to optimize the NDP's performance. The tag word can be used, however, to interpret the contents of 8087 registers.

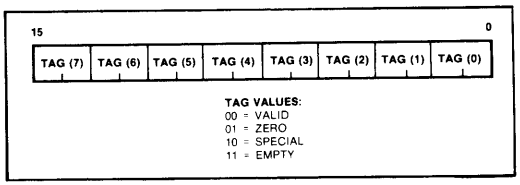


Figure 7. 8087 Tag Word

Instruction and Data Pointers

The instruction and data pointers (see Figure 8) are provided for user-written error handlers. Whenever the 8087 executes an NEU instruction, the CU saves the instruction address, the operand address (if present) and the instruction opcode. 8087 instructions can store this data into memory.

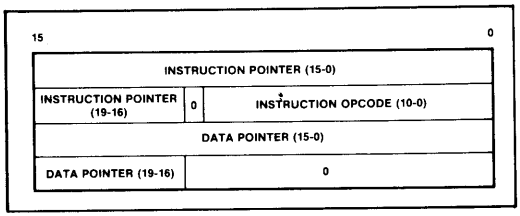


Figure 8. 8087 Instruction and Data Pointers

Table 3. Condition Code Interpretation

Instruction	C_3	C_2	C_1	C_0	Interpretation
Compare, Test	0	X	X	0	A > B
	0	X	X	1	A < B
	1	X	X	0	A = B
	1	X	X	1	A ? B (not comparable)
Remainder	U	0	U	U	Complete reduction
	U	1	U	U	Incomplete reduction
Examine	0	0	0	0	Valid, positive, unnormalized
	0	0	0	1	Invalid, positive, exponent ≠ 0
	0	0	1	0	Valid, negative, unnormalized
	0	0	1	1	Invalid, negative, exponent ≠ 0
	0	1	0	0	Valid, positive, normalized
	0	1	0	1	Infinity, positive
	0	1	1	0	Valid, negative, normalized
	0	1	1	1	Infinity, negative
	1	0	0	0	Zero, positive
	1	0	0	1	Empty
	1	0	1	0	Zero, negative
	1	0	1	1	Empty
	1	1	0	0	Invalid, positive, exponent = 0
	1	1	0	1	Empty
	1	1	1	0	Invalid, negative, exponent = 0
	1	1	1	1	Empty

X = value is not affected by instruction
U = value is undefined following instruction

Control Word

The NDP provides several processing options which are selected by loading a word from memory into the control word. Figure 9 shows the format and encoding of the fields in the control word.

The low order byte of this control word configures 8087 interrupts and exception masking. Bits 5-0 of the control word contain individual masks for each of the six exceptions that the 8087 recognizes and bit 7 contains a general mask bit for all 8087 interrupts. The high order byte of the control word configures the 8087 operating mode including precision, rounding, and infinity control. The precision control bits (bits 9-8) can be used to set the 8087 internal operating precision at less than the default of temporary real precision. This can be useful in providing compatibility with earlier generation arithmetic processors of smaller precision than the 8087. The rounding control bits (bits 11-10) provide for directed rounding and true chop as well as the unbiased round to nearest mode specified in the proposed IEEE standard. Control over closure of the number space at infinity is also provided (either affine closure, $\pm\infty$, or projective closure, ∞, is treated as unsigned, may be specified).

Exception Handling

The 8087 detects six different exception conditions that can occur during instruction execution. Any or all exceptions will cause an interrupt if unmasked and interrupts are enabled.

If interrupts are disabled the 8087 will simply suspend execution until the host clears the exception. If a specific exception class is masked and that exception occurs, however, the 8087 will post the exception in the status register and perform an on-chip default exception handling procedure, thereby allowing processing to continue. The exceptions that the 8087 detects are the following:

1. INVALID OPERATION: Stack overflow, stack underflow, indeterminate form (0/0, $\infty - \infty$, etc.) or the use of a Non-Number (NAN) as an operand. An exponent value is reserved and any bit pattern with this value in the exponent field is termed a Non-Number and causes this exception. If this exception is masked, the 8087's default response is to generate a specific NAN called INDEFINITE, or to propagate already existing NANs as the calculation result.
2. OVERFLOW: The result is too large in magnitude to fit the specified format. The 8087 will generate an encoding for infinity if this exception is masked.
3. ZERO DIVISOR: The divisor is zero while the dividend is a non-infinite, non-zero number. Again, the 8087 will generate an encoding for infinity if this exception is masked.
4. UNDERFLOW: The result is non-zero but too small in magnitude to fit in the specified format. If this exception is masked the 8087 will denormalize (shift right) the fraction until the exponent is in range. This process is called gradual underflow.
5. DENORMALIZED OPERAND: At least one of the operands or the result is denormalized; it has the smallest exponent but a non-zero significand. Normal processing continues if this exception is masked off.
6. INEXACT RESULT: If the true result is not exactly representable in the specified format, the result is rounded according to the rounding mode, and this flag is set. If this exception is masked, processing will simply continue.

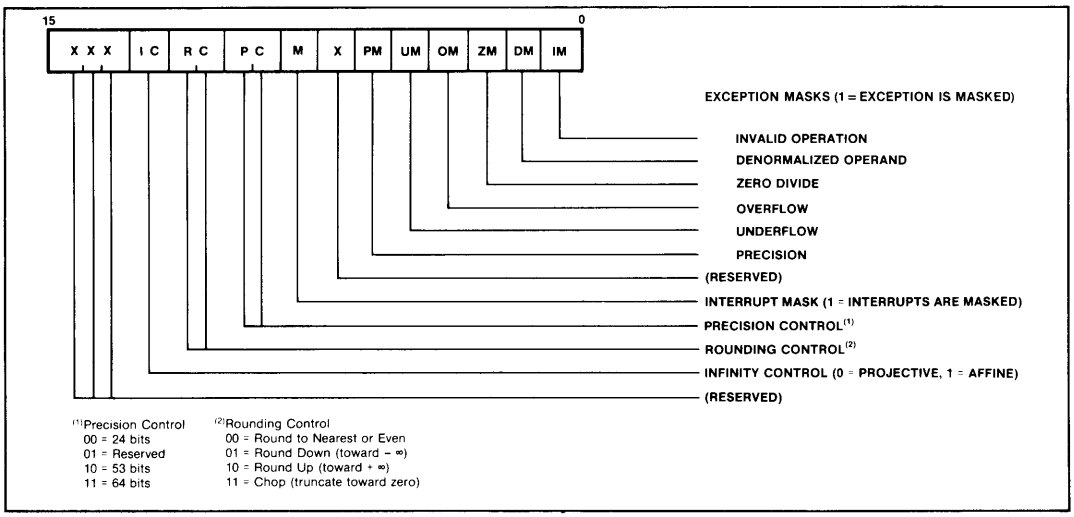


Figure 9. 8087 Control Word

Table 4. Pin Description

Pin(s)	I/O	Function
AD15-AD0	I/O	These lines constitute the time multiplexed memory address (T_1) and data (T_2, T_3, T_W, T_4) bus. A0 is analogous to $\overline{BHE}$ for the lower byte of the data bus, pins D7-D0. It is LOW during T_1 when a byte is to be transferred on the lower portion of the bus in memory operations. Eight-bit oriented devices tied to the lower half of the bus would normally use A0 to condition chip select functions. These lines are active HIGH. They are input/output lines for 8087 driven bus cycles and are inputs which the 8087 monitors when the 8086/8088 is in control of the bus.
A19/S6, A18/S5, A17/S4, A16/S3	I/O	During T_1 these are the four most significant address lines for memory operations. During memory operations, status information is available on these lines during T_2, T_3, T_W, and T_4. For 8087 controlled bus cycles, S6, S4, and S3 are reserved and currently one (HIGH), while S5 is always LOW. These lines are inputs which the 8087 monitors when the 8086/8088 is in control of the bus.
$\overline{BHE}$/S7	I/O	During T_1 the bus high enable signal ($\overline{BHE}$) should be used to enable data onto the most significant half of the data bus, pins D15-D8. Eight-bit oriented devices tied to the upper half of the bus would normally use $\overline{BHE}$ to condition chip select functions. $\overline{BHE}$ is LOW during T_1 for read and write cycles when a byte is to be transferred on the high portion of the bus. The S7 status information is available during T_2, T_3, T_W, and T_4. The signal is active LOW. S7 is an input which the 8087 monitors during 8086/8088 controlled bus cycles.
$\overline{S2}$, $\overline{S1}$, $\overline{S0}$	I/O	For 8087 driven bus cycles, these status lines are encoded as follows: $\overline{S2}$ / $\overline{S1}$ / $\overline{S0}$ 0 (LOW) X X Unused 1 (HIGH) 0 0 Unused 1 0 1 Read Memory 1 1 0 Write Memory 1 1 1 Passive Status is driven active during T_4, remains valid during T_1 and T_2, and is returned to the passive state (1, 1, 1) during T_3 or during T_W when READY is HIGH. This status is used by the 8288 Bus Controller to generate all memory access control signals. Any change in $\overline{S2}$, $\overline{S1}$, or $\overline{S0}$ during T_4 is used to indicate the beginning of a bus cycle, and the return to the passive state in T_3 or T_W is used to indicate the end of a bus cycle. These signals are monitored by the 8087 when the 8086/8088 is in control of the bus.

Pin(s)	I/O	Function
$\overline{RQ}$/$\overline{GT0}$	I/O	This request/grant pin is used by the NDP to gain control of the local bus from the CPU for operand transfers or on behalf of another bus master. It must be connected to one of the two processor request/grant pins. The request grant sequence on this pin is as follows: 1. A pulse one clock wide is passed to the CPU to indicate a local bus request by either the 8087 or the master connected to the 8087 $\overline{RQ}$/$\overline{GT1}$ pin. 2. The NDP waits for the grant pulse and when it is received will either initiate bus transfer activity in the clock cycle following the grant or pass the grant out on the $\overline{RQ}$/$\overline{GT1}$ pin in this clock if the initial request was for another bus master. 3. The 8087 will generate a release pulse to the CPU one clock cycle after the completion of the last NDP bus cycle or on receipt of the release pulse from the bus master on $\overline{RQ}$/$\overline{GT1}$.
$\overline{RQ}$/$\overline{GT1}$	I/O	This request/grant pin is used by another local bus master to force the NDP to release the local bus at the end of the processor's current bus cycle. If the NDP is not in control of the bus when the request is made the request/grant sequence is passed through the NDP on the $\overline{RQ}$/$\overline{GT0}$ pin one cycle later. Subsequent grant and release pulses are also passed through the NDP with a two and one clock delay, respectively, for resynchronization. $\overline{RQ}$/$\overline{GT1}$ has an internal pull-up resistor, and so may be left unconnected. If the NDP has control of the bus the request/grant sequence is as follows: 1. A pulse 1 CLK wide from another local bus master indicates a local bus request to the 8087 (pulse 1). 2. During the NDP's next T_4 or T_I a pulse 1 CLK wide from the 8087 to the requesting master (pulse 2) indicates that the 8087 has allowed the local bus to float and that it will enter the "RQ/GT acknowledge" state at the next CLK. The NDP's control unit is disconnected logically from the local bus during "RQ/GT acknowledge." 3. A pulse 1 CLK wide from the requesting master indicates to the 8087 (pulse 3) that the "RQ/GT" request is about to end and that the 8087 can reclaim the local bus at the next CLK. Each master-master exchange of the local bus is a sequence of 3 pulses. There must be one dead CLK cycle after each bus exchange. Pulses are active LOW.

Table 4. Pin Description (cont.)

Pin(s)	I/O	Function
QS1, QS0	I	QS1 and QS0 provide the 8087 with status to allow tracking of the CPU instruction queue. QS1 QS0 0 (LOW) 0 No Operation 0 1 First Byte of Op Code from Queue 1 (HIGH) 0 Empty the Queue 1 1 Subsequent Byte from Queue
INT	O	This line is used to indicate that an unmasked exception has occurred during numeric instruction execution when 8087 interrupts are enabled (or deadlock has been detected). This signal is typically routed to an 8259A. INT is active HIGH.
BUSY	O	This signal indicates that the 8087 NEU is executing a numeric instruction. It is connected to the CPU's $\overline{TEST}$ pin to provide CPU-NDP synchronization. In the case of an unmasked exception BUSY remains active until the exception is cleared. BUSY is active HIGH.
Ready	I	READY is the acknowledgement from the addressed memory device that it will complete the data transfer. The RDY signal from memory is synchronized by the 8284A Clock Generator to form READY. This signal is active HIGH.
Reset	I	RESET causes the processor to immediately terminate its present activity. The signal must be active HIGH for at least four clock cycles. RESET is internally synchronized.
CLK	I	The clock provides the basic timing for the processor and bus controller. It is asymmetric with a 33% duty cycle to provide optimized internal timing.
V_{CC}		V_{CC} is the +5V power supply pin.
GND		GND are the ground pins.

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias ... 0°C to 70°C
Storage Temperature ..... −65°C to + 150°C
Voltage on Any Pin with
Respect to Ground ..... −1.0 to +7V
Power Dissipation ..... 3.0 Watt

*Notice: Stresses above those listed under Absolute Maximum Ratings may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

D.C. CHARACTERISTICS T_A = 0°C to 70°C, V_{CC} = 5V ± 10%

Symbol	Parameter	Min.	Max.	Units	Test Conditions
V_{IL}	Input Low Voltage	−0.5	+0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC} + 0.5	V	
V_{OL}	Output Low Voltage		0.45	V	I_{OL} = 2.0 mA
V_{OH}	Output High Voltage	2.4		V	I_{OH} = −400 μA
I_{CC}	Power Supply Current		475	mA	T_A = 25°C
I_{LI}	Input Leakage Current		±10	μA	$0V \le V_{IN} \le V_{CC}$
I_{LO}	Output Leakage Current		±10	μA	$0.45V \le V_{OUT} \le V_{CC}$
V_{CL}	Clock Input Low Voltage	−0.5	+0.6	V	
V_{CH}	Clock Input High Voltage	3.9	V_{CC} + 1.0	V	
C_{IN}	Capacitance of Input & Output Buffers (all except I/O Buffer and CLK)		10	pF	fc = 1 MHz
C_{IO}	Capacitance of I/O Buffer (AD0-15, A_{16}–A_{19}, $\overline{BHE}$, $\overline{S2}$-$\overline{S0}$, $\overline{RQ}/\overline{GT}$) and CLK		15	pF	fc = 1MHz

A.C. CHARACTERISTICS T_A = 0°C to 70°C, V_{CC} = 5V ± 10%

Timing Requirements

Symbol	Parameter	Min.	Max.	Units	Test Conditions
TCLCL	CLK Cycle Period	200	500	ns	
TCLCH	CLK Low Time	(2/3TCLCL) − 15		ns	
TCHCL	CLK High Time	(1/3 TCLCL) + 2		ns	
TCH1CH2	CLK Rise Time		10	ns	From 1.0V to 3.5V
TCL2CL1	CLK Fall Time		10	ns	From 3.5V to 1.0V
TDVCL	Data In Setup Time	30		ns	
TCLDX	Data in Hold Time	10		ns	
TRYHCH	READY Setup Time	(2/3TCLCL) − 15		ns	
TCHRYX	READY Hold Time	30		ns	
TRYLCL	READY Inactive to CLK (See Note 3)	−8		ns	
TGVCH	$\overline{RQ}/\overline{GT}$ Setup Time	30		ns	
TCHGX	$\overline{RQ}/\overline{GT}$ Hold Time	40		ns	
TQVCL	QS0-1 Set up Time	30		ns	
TCLQX	QS0-1 Hold Time	10		ns	
TSACH	Status Active Set up Time	30		ns	
TSNCL	Status Inactive Set up Time	30		ns	

Timing Responses

Symbol	Parameter	Min.	Max.	Units	Test Conditions
TCLML	Command Active Delay (See Note 1)	10	35	ns	
TCLMH	Command Inactive Delay (See Note 1)	10	35	ns	
TRYHSH	Ready Active to Status Passive (See Note 2)		110	ns	
TCHSV	Status Active Delay	10	110	ns	
TCLSH	Status Inactive Delay	10	130	ns	
TCLAV	Address Valid Delay	10	110	ns	
TCLAX	Address Hold Time	10		ns	
TCLAZ	Address Float Delay	TCLAX	80	ns	
TSVLH	Status Valid to ALE High (See Note 1)		15	ns	
TCLLH	CLK Low to ALE Valid (See Note 1)		15	ns	
TCHLL	ALE Inactive Delay (See Note 1)		15	ns	CL = 20–100 pF for all 8087 Outputs (in addition to 8087 self-load)
TCLDV	Data Valid Delay	10	110	ns	
TCHDX	Data Hold Time	10		ns	
TCVNV	Control Active Delay (See Note 1)	5	45	ns	
TCVNX	Control Inactive Delay (See Note 1)	10	45	ns	
TCHBV	BUSY and INT Valid Delay	10	150	ns	
TCHDTL	Direction Control Active Delay (See Note 1)		50	ns	
TCHDTH	Direction Control Inactive Delay (See Note 1)		30	ns	
TCLGL	$\overline{RQ}/\overline{GT}$ Active Delay	0	85	ns	
TCLGH	$\overline{RQ}/\overline{GT}$ Inactive Delay	0	85	ns	

Notes:

1. Signal at 8284A or 8288 shown for reference only.
2. Applies only to T_3 and wait states.
3. Applies only to T_2 state (8 ns into T_3).

WAVEFORMS

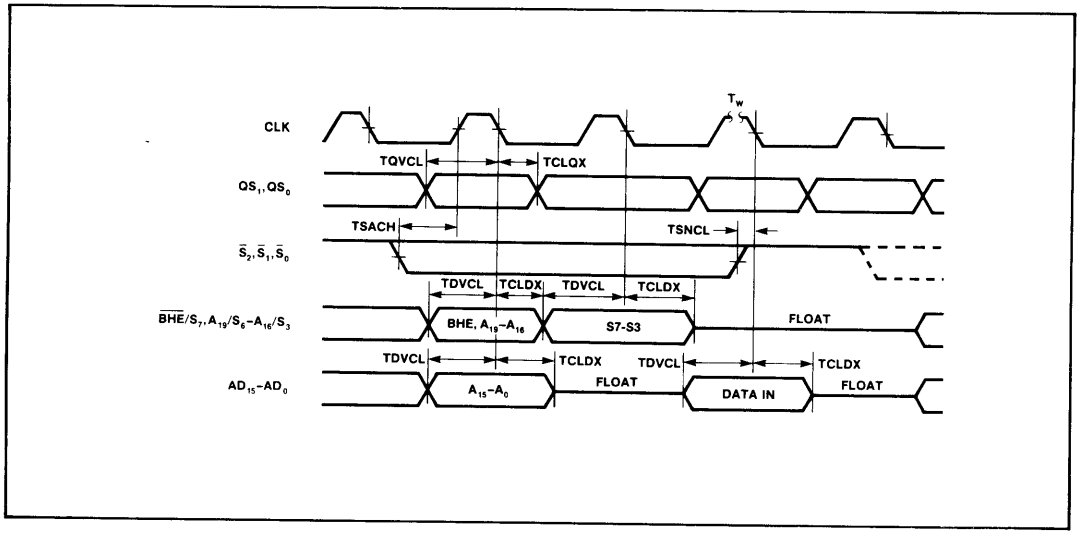
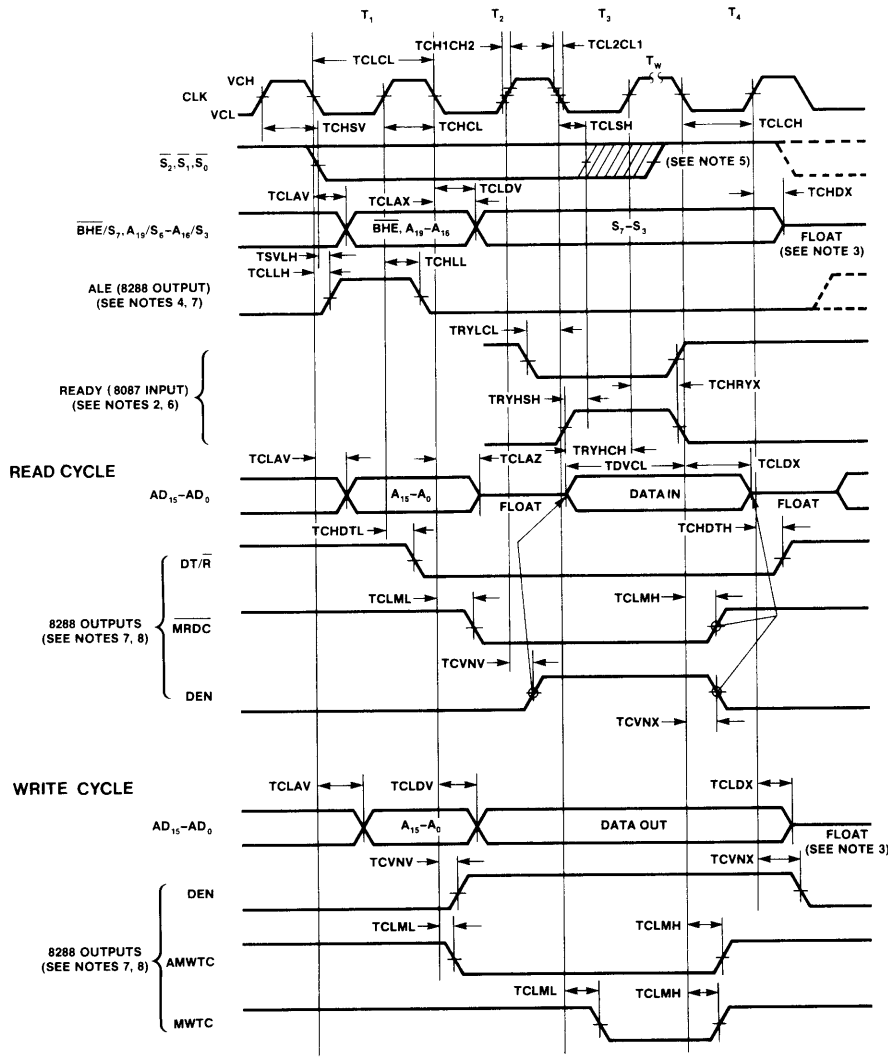


Figure 10. 8087 Bus Timing — Passive Mode



NOTES:

1. ALL SIGNALS SWITCH BETWEEN V_{OL} AND V_{OH} UNLESS OTHERWISE SPECIFIED.
2. RDY IS SAMPLED NEAR THE END OF T_2, T_3 AND T_W TO DETERMINE IF T_W MACHINE STATES ARE TO BE INSERTED.
3. THE LOCAL BUS FLOATS ONLY IF THE 8087 IS RETURNING CONTROL TO THE 8086/8088.
4. ALE RISES AT LATER OF (TSVLH, TCLLH).
5. STATUS INACTIVE IN STATE JUST PRIOR TO T_4.
6. READY SHOULD REMAIN ACTIVE UNTIL S_{0-2} BECOME INACTIVE.
7. SIGNALS AT 8284A OR 8288 ARE SHOWN FOR REFERENCE ONLY.
8. THE ISSUANCE OF 8288 COMMAND AND CONTROL SIGNALS ($\overline{MRDC}$, $\overline{MWTC}$, $\overline{AMWC}$ AND DEN) LAGS THE ACTIVE HIGH 8288 CEN.
9. ALL TIMING MEASUREMENTS ARE MADE AT 1.5V UNLESS OTHERWISE NOTED.

Figure 10. 8087 Bus Timings — Master Mode (cont.)

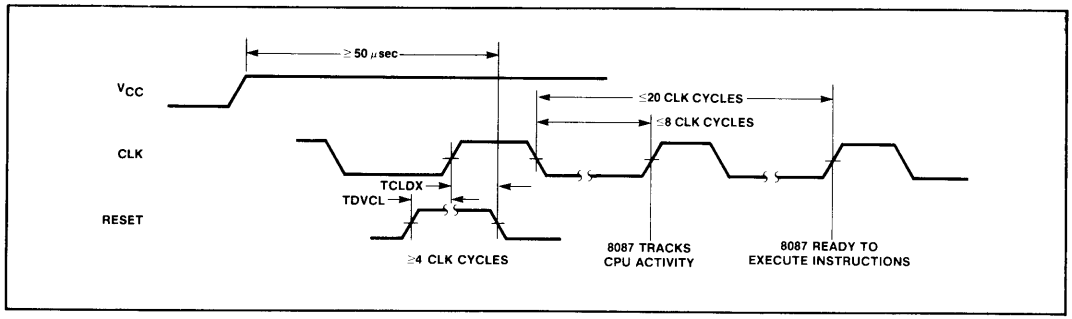


Figure 11. Reset Timing

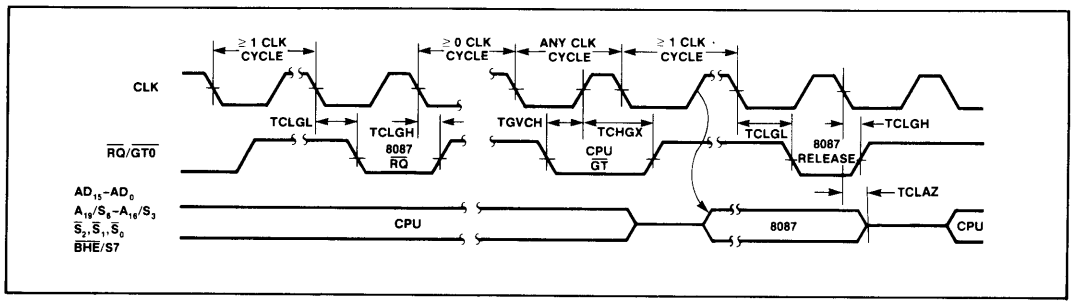
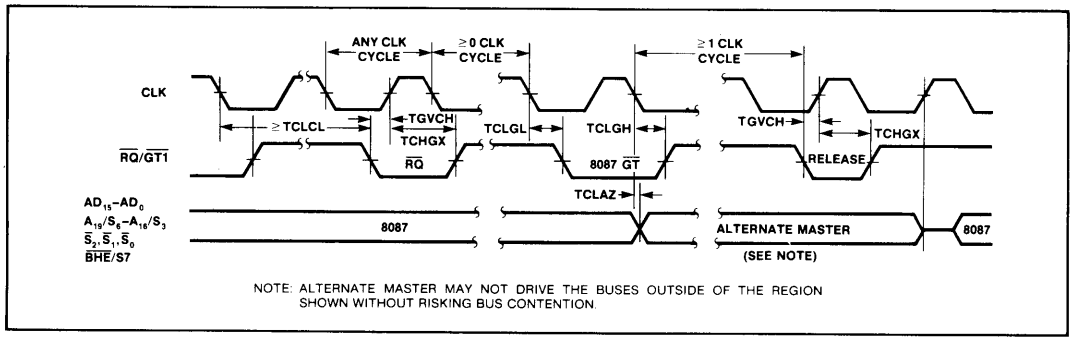


Figure 12. Request/Grant$_0$ Timing



NOTE: ALTERNATE MASTER MAY NOT DRIVE THE BUSES OUTSIDE OF THE REGION SHOWN WITHOUT RISKING BUS CONTENTION.

Figure 13. Request/Grant$_1$ Timing

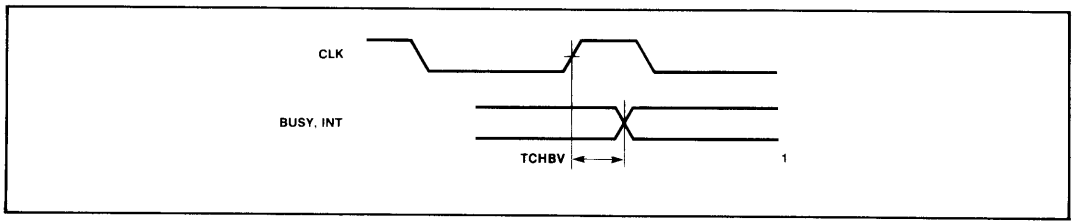


Figure 14. Busy and Interrupt Timing

Table 5. 8087 Extensions to the 8086/8088 Instruction Set

	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
Data Transfer				
FLD = LOAD				
Integer/Real Memory to ST(0)	ESCAPE MF 1	MOD 0 0 0 R/M	(DISP-LO)	(DISP-HI)
Long Integer Memory to ST(0)	ESCAPE 1 1 1	MOD 1 0 1 R/M	(DISP-LO)	(DISP-HI)
Temporary Real Memory to ST(0)	ESCAPE 0 1 1	MOD 1 0 1 R/M	(DISP-LO)	(DISP-HI)
BCD Memory to ST(0)	ESCAPE 1 1 1	MOD 1 0 0 R/M	(DISP-LO)	(DISP-HI)
ST(i) to ST(0)	ESCAPE 0 0 1	1 1 0 0 0 ST(i)		
FST = STORE				
ST(0) to Integer/Real Memory	ESCAPE MF 1	MOD 0 1 0 R/M	(DISP-LO)	(DISP-HI)
ST(0) to ST(i)	ESCAPE 1 0 1	1 1 0 1 0 ST(i)		
FSTP = STORE AND POP				
ST(0) to Integer/Real Memory	ESCAPE MF 1	MOD 0 1 1 R/M	(DISP-LO)	(DISP-HI)
ST(0) to Long Integer Memory	ESCAPE 1 1 1	MOD 1 1 1 R/M	(DISP-LO)	(DISP-HI)
ST(0) to Temporary Real Memory	ESCAPE 0 1 1	MOD 1 1 1 R/M	(DISP-LO)	(DISP-HI)
ST(0) to BCD Memory	ESCAPE 1 1 1	MOD 1 1 0 R/M	(DISP-LO)	(DISP-HI)
ST(0) to ST(i)	ESCAPE 1 0 1	1 1 0 1 1 ST(i)		
FXCH = Exchange ST(i) and ST(0)	ESCAPE 0 0 1	1 1 0 0 1 ST(i)		
Comparison				
FCOM = Compare				
Integer/Real Memory to ST(0)	ESCAPE MF 0	MOD 0 1 0 R/M	(DISP-LO)	(DISP-HI)
ST(i) to ST(0)	ESCAPE 0 0 0	1 1 0 1 0 ST(i)		
FCOMP = Compare and Pop				
Integer/Real Memory to ST(0)	ESCAPE MF 0	MOD 0 1 1 R/M	(DISP-LO)	(DISP-HI)
ST(i) to ST(0)	ESCAPE 0 0 0	1 1 0 1 1 ST(i)		
FCOMPP = Compare ST(1) to ST(0) and Pop Twice	ESCAPE 1 1 0	1 1 0 1 1 0 0 1		
FTST = Test ST(0)	ESCAPE 0 0 1	1 1 1 0 0 1 0 0		
FXAM = Examine ST(0)	ESCAPE 0 0 1	1 1 1 0 0 1 0 1		

Table 5. 8087 Extensions to the 8086/8088 Instruction Set (cont.)

	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
Arithmetic				
FADD = Addition				
Integer/Real Memory with ST(0)	ESCAPE MF 0	MOD 0 0 0 R/M	(DISP-LO)	(DISP-HI)
ST(i) and ST(0)	ESCAPE d P 0	1 1 0 0 0 ST(i)		
FSUB = Subtraction				
Integer/Real Memory with ST(0)	ESCAPE MF 0	MOD 1 0 R R/M	(DISP-LO)	(DISP-HI)
ST(i) and ST(0)	ESCAPE d P 0	1 1 1 0 R R/M		
FMUL = Multiplication				
Integer/Real Memory with ST(0)	ESCAPE MF 0	MOD 0 0 1 R/M	(DISP-LO)	(DISP-HI)
ST(i) and ST(0)	ESCAPE d P 0	1 1 0 0 1 R/M		
FDIV = Division				
Integer/Real Memory with ST(0)	ESCAPE MF 0	MOD 1 1 R R/M	(DISP-LO)	(DISP-HI)
ST(i) and ST(0)	ESCAPE d P 0	1 1 1 1 R R/M		
FSQRT = Square Root of ST(0)	ESCAPE 0 0 1	1 1 1 1 1 0 1 0		
FSCALE = Scale ST(0) by ST(1)	ESCAPE 0 0 1	1 1 1 1 1 1 0 1		
FPREM = Partial Remainder of ST(0) ÷ ST(1)	ESCAPE 0 0 1	1 1 1 1 1 0 0 0		
FRNDINT = Round ST(0) to Integer	ESCAPE 0 0 1	1 1 1 1 1 1 0 0		
FXTRACT = Extract Components of ST(0)	ESCAPE 0 0 1	1 1 1 1 0 1 0 0		
FABS = Absolute Value of ST(0)	ESCAPE 0 0 1	1 1 1 0 0 0 0 1		
FCHS = Change Sign of ST(0)	ESCAPE 0 0 1	1 1 1 0 0 0 0 0		
Transcendental				
FPTAN = Partial Tangent of ST(0)	ESCAPE 0 0 1	1 1 1 1 0 0 1 0		
FPATAN = Partial Arctangent of ST(0) ÷ ST(1)	ESCAPE 0 0 1	1 1 1 1 0 0 1 1		
F2XM1 = $2^{ST(0)}-1$	ESCAPE 0 0 1	1 1 1 1 0 0 0 0		
FYL2X = ST(1) · Log_2 [ST(0)]	ESCAPE 0 0 1	1 1 1 1 0 0 0 1		
FYL2XP1 = ST(1) · Log_2 [ST(0) + 1]	ESCAPE 0 0 1	1 1 1 1 1 0 0 1		
Constants				
FLDZ = LOAD + 0.0 into ST(0)	ESCAPE 0 0 1	1 1 1 0 1 1 1 0		
FLD1 = LOAD + 1.0 into ST(0)	ESCAPE 0 0 1	1 1 1 0 1 0 0 0		
FLDPI = LOAD π into ST(0)	ESCAPE 0 0 1	1 1 1 0 1 0 1 1		
FLDL2T = LOAD $\log_2$ 10 into ST(0)	ESCAPE 0 0 1	1 1 1 0 1 0 0 1		
FLDL2E = LOAD $\log_2$ e into ST(0)	ESCAPE 0 0 1	1 1 1 0 1 0 1 0		
FLDLG2 = LOAD $\log_{10}$ 2 into ST(0)	ESCAPE 0 0 1	1 1 1 0 1 1 0 0		
FLDLN2 = LOAD $\log_e$ 2 into ST(0)	ESCAPE 0 0 1	1 1 1 0 1 1 0 1		

Table 5. 8087 Extensions to the 8086/8088 Instruction Set (cont.)

	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
Processor Control				
FINIT = Initialize NDP	ESCAPE 0 1 1	1 1 1 0 0 0 1 1		
FENI = Enable Interrupts	ESCAPE 0 1 1	1 1 1 0 0 0 0 0		
FDISI = Disable Interrupts	ESCAPE 0 1 1	1 1 1 0 0 0 0 1		
FLDCW = Load Control Word	ESCAPE 0 0 1	MOD 1 0 1 R/M	(DISP-LO)	(DISP-HI)
FSTCW = Store Control Word	ESCAPE 0 0 1	MOD 1 1 1 R/M	(DISP-LO)	(DISP-HI)
FSTSW = Store Status Word	ESCAPE 1 0 1	MOD 1 1 1 R/M	(DISP-LO)	(DISP-HI)
FCLEX = Clear Exceptions	ESCAPE 0 1 1	1 1 1 0 0 0 1 0		
FSTENV = Store Environment	ESCAPE 0 0 1	MOD 1 1 0 R/M	(DISP-LO)	(DISP-HI)
FLDENV = Load Environment	ESCAPE 0 0 1	MOD 1 0 0 R/M	(DISP-LO)	(DISP-HI)
FSAVE = Save State	ESCAPE 1 0 1	MOD 1 1 0 R/M	(DISP-LO)	(DISP-HI)
FRSTOR = Restore State	ESCAPE 1 0 1	MOD 1 0 0 R/M	(DISP-LO)	(DISP-HI)
FINCSTP = Increment Stack Pointer	ESCAPE 0 0 1	1 1 1 1 0 1 1 1		
FDECSTP = Decrement Stack Pointer	ESCAPE 0 0 1	1 1 1 1 0 1 1 0		
FFREE = Free ST(i)	ESCAPE 1 0 1	1 1 0 0 0 ST(i)		
FNOP = No Operation	ESCAPE 0 0 1	1 1 0 1 0 0 0 0		
FWAIT = CPU Wait for NDP	1 0 0 1 1 0 1 1			

FOOTNOTES:

if mod = 00 then DISP = 0*, disp-low and disp-high are absent
if mod = 01 then DISP = disp-low sign-extended to 16-bits, disp-high is absent
if mod = 10 then DISP = disp-high; disp-low
if mod = 11 then r/m is treated as an ST(i) field

if r/m = 000 then EA = (BX) + (SI) + DISP
if r/m = 001 then EA = (BX) + (DI) + DISP
if r/m = 010 then EA = (BP) + (SI) + DISP
if r/m = 011 then EA = (BP) + (DI) + DISP
if r/m = 100 then EA = (SI) + DISP
if r/m = 101 then EA = (DI) + DISP
if r/m = 110 then EA = (BP) + DISP*
if r/m = 111 then EA = (BX) + DISP

*except if mod = 000 and r/m = 110 then EA = disp-high: disp-low.

MF = Memory Format
- 00 — 32-bit Real
- 01 — 32-bit Integer
- 10 — 64-bit Real
- 11 — 16-bit Integer

ST(0) = Current stack top
ST(i) = i^{th} register below stack top

d = Destination
- 0 — Destination is ST(0)
- 1 — Destination is ST(i)

P = Pop
- 0 — No pop
- 1 — Pop ST(0)

R = Reverse
- 0 — Destination (op) Source
- 1 — Source (op) Destination

For **FSQRT:** $-0 \leq ST(0) \leq +\infty$
For **FSCALE:** $-2^{15} \leq ST(1) < +2^{15}$ and ST(1) integer
For **F2XM1:** $0 \leq ST(0) \leq 2^{-1}$
For **FYL2X:** $0 < ST(0) < \infty$, $-\infty < ST(1) < +\infty$
For **FYL2XP1:** $0 < |ST(0)| < (2 - \sqrt{2})/2$, $-\infty < ST(1) < \infty$
For **FPTAN:** $0 \leq ST(0) < \pi/4$
For **FPATAN:** $0 \leq ST(0) < ST(1) < +\infty$

U.S. AND CANADIAN SALES OFFICES

3065 Bowers Avenue
Santa Clara, California 95051
Tel: (408) 987-8080
TWX: 910-338-0026
TELEX: 34-6372

May 1980

ALABAMA

Intel Corp.
303 Williams Avenue, S.W.
Suite 1422
Huntsville 35801
Tel: (205) 533-9353

Pen-Tech Associates, Inc.
Holiday Office Center
3322 Memorial Pkwy., S.W.
Huntsville 35801
Tel: (205) 881-9298

ARIZONA

Intel Corp.
10210 N. 25th Avenue, Suite 11
Phoenix 85021
Tel: (602) 997-9695

BFA
4426 North Saddle Bag Trail
Scottsdale 85251
Tel: (602) 994-5400

CALIFORNIA

Intel Corp.
7670 Opportunity Rd.
Suite 135
San Diego 92111
Tel: (714) 268-3563

Intel Corp.*
2000 East 4th Street
Suite 100
Santa Ana 92705
Tel: (714) 835-9642
TWX: 910-595-1114

Intel Corp.*
15335 Morrison
Suite 345
Sherman Oaks 91403
Tel: (213) 986-9510
TWX: 910-495-2045

Intel Corp.*
3375 Scott Blvd.
Santa Clara 95051
Tel: (408) 987-8086
TWX: 910-339-9279
910-338-0255

Earle Associates, Inc.
4617 Ruffner Street
Suite 202
San Diego 92111
Tel: (714) 278-5441

Mac-I
2576 Shattuck Ave.
Suite 4B
Berkeley 94704
Tel: (415) 843-7625

Mac-I
P.O. Box 1420
Cupertino 95014
Tel: (408) 257-9880

Mac-I
558 Valley Way
Calaveras Business Park
Milpitas 95035
Tel: (408) 946-8885

Mac-I
P.O. Box 8763
Fountain Valley 92708
Tel: (714) 839-3341

Mac-I
1321 Centinela Avenue
Suite 1
Santa Monica 90404
Tel: (213) 829-4797

Mac-I
20121 Ventura Blvd., Suite 240E
Woodland Hills 91364
Tel: (213) 347-5900

COLORADO

Intel Corp.*
650 S. Cherry Street
Suite 720
Denver 80222
Tel: (303) 321-8066
TWX: 910-931-2289

Westek Data Products, Inc.
25921 Fern Gulch
P.O. Box 1355
Evergreen 80439
Tel: (303) 674-5255

Westek Data Products, Inc.
1322 Arapahoe
Boulder 80302
Tel: (303) 449-2620

Westek Data Products, Inc.
1228 W. Hinsdale Dr.
Littleton 80120
Tel: (303) 797-0482

CONNECTICUT

Intel Corp.
Peacock Alley
1 Padanaram Road, Suite 146
Danbury 06810
Tel: (203) 792-8366
TWX: 710-456-1199

FLORIDA

Intel Corp.
1001 N.W. 62nd Street, Suite 406
Ft. Lauderdale 33309
Tel: (305) 771-0600
TWX: 510-956-9407

Intel Corp.
5151 Adanson Street, Suite 203
Orlando 32804
Tel: (305) 628-2393
TWX: 810-853-9219

Pen-Tech Associates, Inc.
201 S.E. 15th Terrace, Suite K
Deerfield Beach 33441
Tel: (305) 421-4989

Pen-Tech Associates, Inc.
111 So. Maitland Ave., Suite 202
P.O. Box 1475
Maitland 32751
Tel: (305) 645-3444

GEORGIA

Pen Tech Associates, Inc.
Cherokee Center, Suite 21
627 Cherokee Street
Marietta 30060
Tel: (404) 424-1931

ILLINOIS

Intel Corp.*
2550 Golf Road, Suite 815
Rolling Meadows 60008
Tel: (312) 981-7200
TWX: 910-651-5881

Technical Representatives
1502 North Lee Street
Bloomington 61701
Tel: (309) 829-8080

INDIANA

Intel Corp.
9101 Wesleyan Road
Suite 204
Indianapolis 46268
Tel: (317) 299-0623

IOWA

Technical Representatives, Inc.
St. Andrews Building
1930 St. Andrews Drive N.E.
Cedar Rapids 52405
Tel: (319) 393-5510

KANSAS

Intel Corp.
9393 W. 110th St., Ste. 265
Overland Park 66210
Tel: (913) 642-8080

Technical Representatives, Inc.
8245 Nieman Road, Suite 100
Lenexa 66214
Tel: (913) 888-0212, 3, & 4
TWX: 910-749-6412

Technical Representatives, Inc.
360 N. Rock Road
Suite 4
Wichita 67206
Tel: (316) 681-0242

MARYLAND

Intel Corp.*
7257 Parkway Drive
Hanover 21076
Tel: (301) 796-7500
TWX: 710-862-1944

Mesa Inc.
11900 Parklawn Drive
Rockville 20852
Tel: Washington (301) 881-8430
Baltimore (301) 792-0021

MASSACHUSETTS

Intel Corp.*
27 Industrial Ave.
Chelmsford 01824
Tel: (617) 667-8126
TWX: 710-343-6333

EMC Corp.
381 Elliot Street
Newton 02164
Tel: (617) 244-4740
TWX: 922531

MICHIGAN

Intel Corp.*
26500 Northwestern Hwy.
Suite 401
Southfield 48075
Tel: (313) 353-0920
TWX: 810-244-4915

Lowry & Associates, Inc.
135 W. North Street
Suite 4
Brighton 48116
Tel: (313) 227-7067

Lowry & Associates, Inc.
3902 Costa NE
Grand Rapids 49505
Tel: (616) 363-9839

MINNESOTA

Intel Corp.
7401 Metro Blvd.
Suite 355
Edina 55435
Tel: (612) 835-6722
TWX: 910-576-2867

MISSOURI

Intel Corp.
502 Earth City Plaza
Suite 121
Earth City 63045
Tel: (314) 291-1990

Technical Representatives, Inc.
320 Brookes Drive, Suite 104
Hazelwood 63042
Tel: (314) 731-5200
TWX: 910-762-0618

NEW JERSEY

Intel Corp.*
Raritan Plaza
2nd Floor
Raritan Center
Edison 08817
Tel: (201) 225-3000
TWX: 710-480-6238

NEW MEXICO

BFA Corporation
P.O. Box 1237
Las Cruces 88001
Tel: (505) 523-0601
TWX: 910-983-0543

BFA Corporation
3705 Westerfield, N.E.
Albuquerque 87111
Tel: (505) 292-1212
TWX: 910-989-1157

NEW YORK

Intel Corp.*
350 Vanderbilt Motor Pkwy.
Suite 402
Hauppauge 11787
Tel: (516) 231-3300
TWX: 510-227-6236

Intel Corp.
80 Washington St.
Poughkeepsie 12601
Tel: (914) 473-2303
TWX: 510-248-0060

Intel Corp.*
2255 Lyell Avenue
Lower Floor East Suite
Rochester 14606
Tel: (716) 254-6120
TWX: 510-253-7391

Measurement Technology, Inc.
159 Northern Boulevard
Great Neck 11021
Tel: (516) 482-3500

T-Squared
4054 Newcourt Avenue
Syracuse 13206
Tel: (315) 463-8592
TWX: 710-541-0554

T-Squared
2 E. Main
Victor 14564
Tel: (716) 924-9101
TWX: 510-254-8542

NORTH CAROLINA

Intel Corp.
154 Huffman Mill Rd.
Burlington 27215
Tel: (919) 584-3631

Pen-Tech Associates, Inc.
1202 Eastchester Dr.
Highpoint 27260
Tel: (919) 883-9125

OHIO

Intel Corp.*
6500 Poe Avenue
Dayton 45415
Tel: (513) 890-5350
TWX: 810-450-2528

Intel Corp.*
Chagrin-Brainard Bldg., No. 210
28001 Chagrin Blvd.
Cleveland 44122
Tel: (216) 464-2736
TWX: 810-427-9298

OREGON

Intel Corp.
10700 S.W. Beaverton
Hillsdale Highway
Suite 324
Beaverton 97005
Tel: (503) 641-8086
TWX: 910-467-8741

PENNSYLVANIA

Intel Corp.*
275 Commerce Dr.
200 Office Center
Suite 300
Fort Washington 19034
Tel: (215) 542-9444
TWX: 510-661-2077

Q.E.D. Electronics
300 N. York Road
Hatboro 19040
Tel: (215) 674-9600

TEXAS

Intel Corp.*
2925 L.B.J. Freeway
Suite 175
Dallas 75234
Tel: (214) 241-9521
TWX: 910-860-5617

Intel Corp.*
6420 Richmond Ave.
Suite 280
Houston 77057
Tel: (713) 784-3400
TWX: 910-881-2490

Industrial Digital Systems Corp.
5925 Sovereign
Suite 101
Houston 77036
Tel: (713) 988-9421

Intel Corp.
313 E. Anderson Lane
Suite 314
Austin 78752
Tel: (512) 454-3628

WASHINGTON

Intel Corp.
Suite 114, Bldg. 3
1603 116th Ave. N.E.
Bellevue 98005
Tel: (206) 453-8086
TWX: 910-443-3002

WISCONSIN

Intel Corp.
150 S. Sunnyslope Rd.
Brookfield 53005
Tel: (414) 784-9060

CANADA

Intel Semiconductor Corp.*
Suite 233, Bell Mews
39 Highway 7, Bells Corners
Ottawa, Ontario K2H 8R2
Tel: (613) 829-9714
TELEX: 053-4115

Intel Semiconductor Corp.
50 Galaxy Blvd.
Unit 12
Rexdale, Ontario
M9W 4Y5
Tel: (416) 675-2105
TELEX: 06983574

Multilek, Inc.*
15 Grenfell Crescent
Ottawa, Ontario K2G 0G3
Tel: (613) 226-2365
TELEX: 053-4585

Multilek, Inc.
Toronto
Tel: (416) 245-4622

Multilek, Inc.
Montreal
Tel: (514) 481-1350

*Field Application Location

INTERNATIONAL SALES AND MARKETING OFFICES

3065 Bowers Avenue
Santa Clara, California 95051
Tel: (408) 987-8080
TWX: 910-338-0026
TELEX: 34-6372

INTERNATIONAL DISTRIBUTORS/REPRESENTATIVES

May 1980

ARGENTINA

Micro Sistemas S.A.
9 De Julio 561
Cordoba
Tel: 54-51-32-880
TELEX: 51837 BICCO

AUSTRALIA

A.J.F. Systems & Components Pty. Ltd.
310 Queen Street
Melbourne
Victoria 3000
Tel:
TELEX:

Warburton Franki
Corporate Headquarters
372 Eastern Valley Way
Chatswood, New South Wales 2067
Tel: 407-3261
TELEX: AA 21299

AUSTRIA

Bacher Elektronische Geraete GmbH
Rotenmulgasse 26
A 1120 Vienna
Tel: (0222) 83 63 96
TELEX: (01) 1532

Rekirsch Elektronik Geraete GmbH
Lichtensteinstrasse 97
A1000 Vienna
Tel: (222) 347646
TELEX: 74759

BELGIUM

Inelco Belgium S.A.
Ave. des Croix de Guerre 94
B1120 Brussels
Tel: (02) 216 01 60
TELEX: 25441

BRAZIL

Icotron S.A.
0511-Av. Mutinga 3650
6 Andar
Pirituba-Sao Paulo
Tel: 261-0211
TELEX: (011) 222 ICO BR

CHILE

DIN
Av. Vic. Mc kenna 204
Casilla 6055
Santiago
Tel: 227 564
TELEX: 3520003

CHINA

C.M. Technologies
525 University Avenue
Suite A-40
Palo Alto, CA 94301

COLOMBIA

International Computer Machines
Adpo. Aereo 19403
Bogota 1
Tel: 232-6635
TELEX: 43439

CYPRUS

Cyprus Eltrom Electronics
P.O. Box 5393
Nicosia
Tel: 21-27982

DENMARK

STL-Lyngso Komponent A/S
Ostmarken 4
DK-2860 Soborg
Tel: (01) 67 00 77
TELEX: 22990

Scandinavian Semiconductor
Supply A/S
Nannasgade 18
DK-2200 Copenhagen
Tel: (01) 83 50 90
TELEX: 19037

FINLAND

Oy Fintronic AB
Melkonkatu 24 A
SF-00210
Helsinki 21
Tel: 0-692 6022
TELEX: 124 224 Ftron SF

FRANCE

Celdis S.A.*
53, Rue Charles Frerot
F-94250 Gentiily
Tel: (1) 581 00 20
TELEX: 200 485

Feutrier
Rue des Trois Glorieuses
F-42270 St. Priest-en-Jarez
Tel: (77) 74 67 33
TELEX: 300 0 21

Metrologie*
La Tour d'Asnieres
4, Avenue Laurent Cely
92606-Asnieres
Tel: 791 44 44
TELEX: 611 448

Tekelec Airtronic*
Cite des Bruyeres
Rue Carle Vernet
F-92310 Sevres
Tel: (1) 534 75 35
TELEX: 204552

GERMANY

Electronic 2000 Vertriebs GmbH
Neumarkter Strasse 75
D-8000 Munich 80
Tel: (089) 434061
TELEX: 522561

Jermyn GmbH
Postfach 1180
D-6077 Camberg
Tel: (06434) 231
TELEX: 484426

Kontron Elektronik GmbH
Breslauerstrasse 2
8057 Eching B
D-8000 Munich
Tel: (89) 319.011
TELEX: 522122

Neye Enatechnik GmbH
Schillerstrasse 14
D-2085 Quickborn-Hamburg
Tel: (04106) 6121
TELEX: 02-13590

GREECE

American Technical Enterprises
P.O. Box 156
Athens
Tel:* 30-1-8811271
30-1-8219470

HONG KONG

Schmidt & Co.
28/F Wing on Center
Connaught Road
Hong Kong
Tel: 5-455-644
TELEX: 74766 Schmc Hx

INDIA

Micronic Devices
104/109C, Nirmal Industrial Estate
Sion (E)
Bombay 400022, India
Tel: 486-170
TELEX: 011-5947 MDEV IN

ISRAEL

Eastronics Ltd.*
11 Rozanis Street
P.O. Box 39300
Tel Aviv 61390
Tel: 475151
TELEX: 33638

ITALY

Eledra 3S S.P.A.*
Viale Elvezia, 18
I 20154 Milan
Tel: (02) 34.93.041-31.85.441
TELEX: 332332

JAPAN

Asahi Electronics Co. Ltd.
KMM Bldg. Room 407
2-14-1 Asano, Kokura
Kita-Ku, Kitokyushu City 802
Tel: (093) 511-6471
TELEX: AECKY 7126-16

Hamilton-Avnet Electronics Japan Ltd
YU and YOU Bldg. 1-4 Horidome-Cho
Nihonbashi
Tel: (03) 662-9911
TELEX: 2523774

Nippon Micro Computer Co. Ltd.
Mutsumi Bldg. 4-5-21 Kojimachi
Chiyoda-ku, Tokyo 102
Tel: (03) 230-0041

Ryoyo Electric Corp.
Konwa Bldg.
1-12-22, Tsukiji, 1-Chome
Chuo-Ku, Tokyo 104
Tel: (03) 543-7711

Tokyo Electron Ltd.
No. 1 Higashikata-Machi
Midori-Ku, Yokohama 226
Tel: (045) 471-8811
TELEX: 781-4473

KOREA

Koram Digital
Room 411 Ahil Bldg.
49-4 2-GA Hoehyun-Dong
Chung-Ku Seoul
Tel: 23-8123
TELEX: K23542 HANSINT

Leewood International, Inc.
C.P.O. Box 4046
112-25, Sokong-Dong
Chung-Ku, Seoul 100
Tel: 28-5927
CABLE: "LEEWOOD" Seoul

NETHERLANDS

Inelco Nether. Comp. Sys. BV
Turfstekerstraat 63
Aalsmeer 1431 D
Tel: (2977) 28855
TELEX: 14693

Koning & Hartman
Koperwerf 30
2544 EN Den Haag
Tel: (70) 210.101
TELEX: 31528

NEW ZEALAND

W. K. McLean Ltd.
P.O. Box 18-065
Glenn Innes, Auckland, 6
Tel: 587-037
TELEX: NZ2763 KOSFY

NORWAY

Nordisk Elektronik (Norge) A/S
Postoffice Box 122
Smedsvingen 4
1364 Hvalstad
Tel: 02 78 62 10
TELEX: 17546

PORTUGAL

Ditram
Componentes E Electronica LDA
Av. Miguel Bombarda, 133
Lisboa 1
Tel: (19) 545313
TELEX: 14347 GESPIC

SINGAPORE

General Engineers Associates
Blk 3, 1003-1008, 10th Floor
P.S.A. Multi-Storey Complex
Telok Blangah/Pasir Panjang
Singapore 5
Tel: 271-3163
TELEX: RS23987 GENERCO

SOUTH AFRICA

Electronic Building Elements
Pine Square
18th Street
Hazelwood, Pretoria 0001
Tel: 789 221
TELEX: 30181SA

SPAIN

Interface
Av. Generalisimo 51 9°
E-Madrid 16
Tel: 456 3151

ITT SESA
Miguel Angel 16
Madrid 10
Tel: (1) 4190957
TELEX: 27707/27461

SWEDEN

AB Gosta Backstrom
Box 12009
10221 Stockholm
Tel: (08) 541 080
TELEX: 10135

Nordisk Elektronik AB
Box 27301
S-10254 Stockholm
Tel: (08) 635040
TELEX: 10547

SWITZERLAND

Industrade AG
Gemsenstrasse 2
Postcheck 80 - 21190
CH-8021 Zurich
Tel: (01) 60 22 30
TELEX: 56788

TAIWAN

Taiwan Automation Co.*
3d Floor #75, Section 4
Nanking East Road
Taipei
Tel: 771-0940
TELEX: 11942 TAIAUTO

TURKEY

Turkelek Electronics
Apapurk Boulevard 169
Ankara
Tel: 189483

UNITED KINGDOM

Comway Microsystems Ltd.
Market Street
68-Bracknell, Berkshire
Tel: (344) 51654
TELEX: 847201

G.E.C. Semiconductors Ltd.
East Lane
North Wembley
Middlesex HA9 7PP
Tel: (01) 904-9303/908-4111
TELEX: 28817

Jermyn Industries
Vestry Estate
Sevenoaks, Kent
Tel: (0732) 501.44
TELEX: 95142

Rapid Recall, Ltd.
6 Soho Mills Ind. Park
Wooburn Green
Bucks, England
Tel: (6285) 24961
TELEX: 849439

Sintrom Electronics Ltd.*
Arkwright Road 2
Reading, Berkshire RG2 0LS
Tel: (0734) 85464
TELEX: 847395

VENEZUELA

Componentes y Circuitos
Electronicos TTLCA C.A.
Apartado 3223
Caracas 101
Tel: 718-100
TELEX: 21795 TELETIPOS

*Field Application Location