Il Coprocessore Matematico Intel 8087

Introduzione

Molti conoscono il microprocessore 8086 capostipite dei microprocessori che fanno girare i nostri PC, molti meno quelli che conoscono il coprocessore matematico ugualmente presente, che provvede allo svolgimento delle operazioni matematiche più complicate: l'aritmetica fra numeri reali, le operazioni logaritmiche, esponenziali e trigonometriche.

E' giunta l'ora di colmare questa lacuna con un breve corso sul capostipite dei coprocessori dei PC: l'Intel 8087.

Un po' di Storia

I primi PC (anni '80) comprendevano il solo microprocessore(8086, 80286, 80386) ed avevano uno zoccolo libero sulla motherboard dove era possibile inserire un integrato supplementare che era il "coprocessore matematico". Questo integrato, che all'epoca costava più del microprocessore, aggiunto sulla motherboard consentiva di svolgere in modo molto veloce le operazioni matematiche più complesse (funzioni trigonometriche, logaritmiche, ecc.). Veniva acquistato solo da chi usava il PC per svolgere lavori complessi: progettazione architettonica 3D, CAD, ecc. . Dopo un po' di anni, con la progressiva disponibilità di spazio sui chip, a partire dall'Intel 80486, fu inserito sullo stesso chip del microprocessore.

Chi impara la programmazione assembler del PC, spesso si limita alla programmazione del microprocessore base: l'Intel 8086, evitando le notevoli complicazioni aggiunte nella programmazione asm a 32 bit (dal '386 in poi). Mentre questo ampliamento dell'assembler non è sempre necessario, è un peccato evitare di studiare il funzionamento del coprocessore matematico, almeno nella versione base 8087, in quanto le operazioni fra numeri reali, trigonometriche e trascendentali, sono molto comuni.

Il microprocessore 8086, ed i suoi coetanei, sono in grado di effettuare le 4 operazioni aritmetiche (nello z80 solo somma e sottrazione, le altre due indirettamente mediante apposite routine) su numeri interi con o senza segno e su numeri BCD. Mancano all'appello i numeri reali e le più complesse operazioni logaritmiche, esponenziali e trigonometriche. Ad occuparsi di ciò è un apposito dispositivo detto Coprocessore Matematico o FPU (Floating Point Unit).
Il fatto che l'unità che si occupa di queste operazioni sia distinta dalla CPU è legato a ragioni storiche. Mentre per fare somme e sottrazioni fra interi bastano circuiti molto semplici (per fare la somma fra due numeri di N bit bastano 2*N porte logiche), già per effettuare moltiplicazioni e divisioni fra interi la cosa si complica abbastanza. Passando poi alle operazioni logaritimiche, esponenziali e trigonometriche la complessità dei circuiti cresce enormemente.

Agli inizi della microelettronica bisognava realizzare sistemi molto semplici in quanto su ogni chip si potevano realizzare poche porte logiche.
Ci si limitò quindi a CPU essenziali in tutto. Si realizzavano dei chip aggiuntivi che si potevano montare o meno sulla motherboard e che integravano il set delle operazioni della CPU con queste operazioni più complesse: le FPU. I Computer che non disponevano di FPU potevano comunque realizzare anche queste ultime operazioni facendo però uso di subroutine lunghe e complesse che rallentavano molto la velocità del computer (alcuni programmi venivano prodotti nelle due versioni per computer con o senza FPU).
All'inizio la complessità (intesa come numero di porte logiche contenute in un chip) delle FPU superava ampiamente quella delle CPU.
Col tempo, all'aumentare del numero delle porte logiche realizzabili su un singolo chip, dapprima aumentarono le funzioni delle CPU, successivamente si realizzarono sullo stesso chip CPU e FPU.

Nei Personal Computer (IBM Compatibili) man mano che l'INTEL realizzava una nuova CPU realizzava anche la corrispondente FPU: CPU 8086 e FPU 8087; CPU 80286 e FPU 80287; CPU 80386 e FPU 80387. La CPU 80486 venne realizzata in due versioni: 80486SX che non conteneva la FPU e 80486DX che la conteneva già all'interno.
Dopo quel microprocessore, tutti i successivi la dispongono già al loro interno.

Sebbene ormai non esistano più in circolazione Personal Computer privi di Coprocessore Matematico, per mantenere la compatibilità col software precedente, l'organizzazione interna e' rimasta sostanzialmente la stessa per cui logicamente continuiamo a studiare e programmare il Coprocessore come un'entita' a parte.
In questo corso tralasceremo la parte hardware e vedremo solo l'organizzazione ed il funzionamento interno.

Architettura Interna

L'8087 è suddiviso internamente in due unità: la CU (Control Unit = unità di Controllo) e la NEU (Numeric Execution Unit= unità di esecuzione numerica). La prima si occupa principalmente di leggere e scrivere i dati in memoria e di effettuare il fetch dei codici operativi. La seconda svolge propriamente i calcoli.
La CU è organizzata in modo molto simile alla BIU(Bus Unit Interface) della CPU. In particolare dispone della stessa Coda di Prefetch lunga 6 byte (nell'8086 e 8087).
Due segnali hardware (QS1,QS0) escono dalla CPU ed entrano nella FPU informando costantemente quest'ultima di quanti byte sono presenti nella coda di prefetch del primo ed imponendo che anche l'altra si riempia e svuoti allo stesso modo.
Altri tre segnali (S2,S1,S0) portano alla FPU l'informazione di che tipo di ciclo di memoria si sta effettuando (101= ciclo di lettura in memoria, 110= ciclo di scrittura in memoria).
Altri segnali (S6,S5,S4,S3) informano la FPU di quando si sta effettuando un ciclo di Fetch.
Con tutte queste informazioni la CU è quindi in grado di:

- leggere i codici operativi di tutte le istruzioni che vengono lette dalla CPU
- inserirli nella propria Coda di Prefetch e di mantenerla in ogni momento identica a quella della CPU.

La NEU, si occupa dello svolgimento delle operazioni. E' un complesso sistema sequenziale che per eseguire molte operazioni deve ripetere un gruppo di operazioni elementari per un gran numero di volte. Il calcolo delle funzioni trigonometriche si ottiene mediante l'esecuzione di un ciclo di operazioni che viene troncato quando l'approssimazione raggiunta è pari a quanto voluto (più avanti vedremo che la durata di molte operazioni può variare molto proprio in relazione al numero di cicli necessari per raggiungere l'approssimazione voluta).

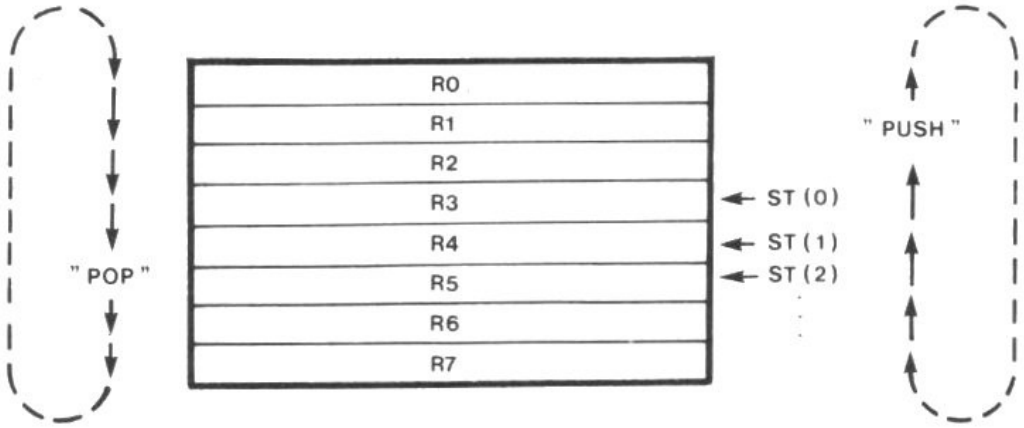
I tempi di esecuzione sono conseguentemente molto più lunghi di quelli delle normali operazioni svolte dalla CPU.
Fra i vari segnali di collegamento alla CPU ve n'è anche uno READY, che serve a segnalare alla CPU che la FPU non ha ancora completato l'esecuzione dell'istruzione corrente, per cui la CPU deve attendere che finisca.

Registri Interni

Registri Dati

L'8087 ha 8 registri dati organizzati in uno Stack Circolare. I dati vengono inseriti con una istruzione Push e prelevati con una Pop. Siccome questo Stack ha dimensione limitata, la FPU

rileva se cerchiamo di inserire più dati dei posti disponibili (es. 9 push consecutive) oppure se ne vogliamo prelevare più di quanti sono presenti. In questi casi la FPU segnala l'errore (rispettivamente stack overflow e stack underflow).



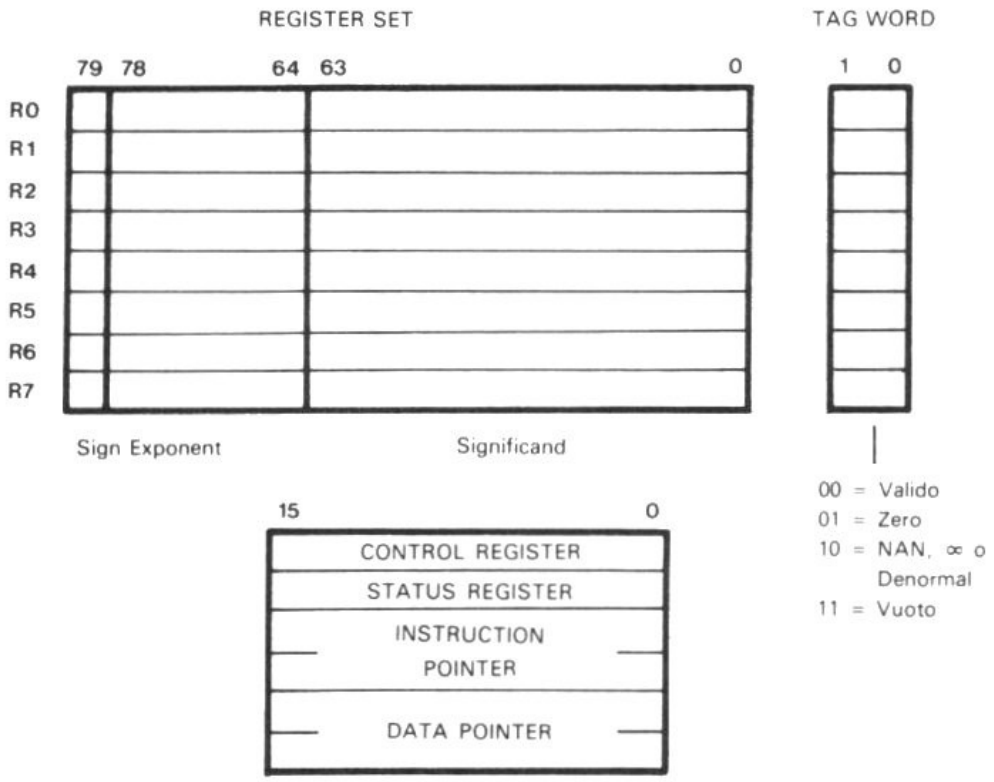
Push/Pop dello stack

La gestione di uno stack circolare si fa con due puntatori: uno che indica il primo posto occupato, l'altro che segnala il primo libero. Il puntatore che indica il primo occupato (quello che uscirebbe se facessimo una Pop) prende il nome di ST (Stack Top), è grande 3 bit (siccome i registri sono 8, bastano 3 bit per rappresentare l'indirizzo di uno di questi) ed è leggibile dal programmatore.
Ognuno degli 8 registri dati è grande ben 80 bit e può contenere numeri interi, reali e BCD.
Per ognuno dei registri Dati ve n'è associato un altro di 2 bit detto TAG WORD. In questi "mini_registri" sono scritte, automaticamente dalla FPU, informazioni relative al dato associato:

TAG WORD - convenzioni	
00	Dato Valido
01	Zero
10	NAN, infinito, Denormalizzato
11	Vuoto

Al reset, prima che venga caricato un dato in un registro, esso è marcato Vuoto (vale a dire nella Tag Word associata sarà scritto 11). Stessa cosa dopo che viene fatta una Pop. Dopo una Push verrà marcato 00. Il codice 10 viene riservato a numeri particolari che vedremo successivamente.

Nelle operazioni sui dati ci si riferisce a questi dati specificando la posizione rispetto al TOP (es. ST(0)= il Top dello Stack, cioè quello che uscirebbe dopo una Pop; ST(1) quello successivo cioè quello che uscirebbe dopo due Pop, ecc. fino ad arrivare a ST(7)).



Set di registri dell'8087

Status e Control Word

Nei microprocessori, man mano che aumenta la complessità aumenta anche il numero dei loro Flag. Qui troviamo ben due registri a 16 bit.
I Flag vengono divisi in due categorie: di Stato e di Controllo.
I Flag di Stato sono quelli che ci forniscono un risultato del processore (es. il Flag Z ci dice se il risultato è o no Zero; quello di Overflow se c'è stato Overflow). Si possono solo leggere perché non ha senso andarli a scrivere (se impostiamo il valore di Z non per questo sarà cambiato il risultato).

I Flag di Controllo sono invece quelli che ci permettono di inserire un'informazione che poi sarà utilizzata dal processore (se impostiamo il Carry a 1 o 0, possiamo far svolgere una Rotate in un modo differente). Nei microprocessori visti alcuni Flag sono a volte di Stato altre di Controllo.
Quando la complessità aumenta, i flag vengono rigidamente suddivisi nei due tipi ed è ammessa la scrittura solo per quelli di controllo.
Nel Coprocessore Matematico i flag di Stato stanno nella STATUS WORD quelli di controllo nella CONTROL WORD. Andiamoli a vedere uno ad uno.

STATUS WORD															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
B	**C3**	**T2**	**T1**	**T0**	**C2**	**C1**	**C0**	**IR**		**PE**	**UE**	**OE**	**ZE**	**DE**	**IE**
		TOP								**Exceptions**					

CONTROL WORD															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
			IC	**RC1**	**RC0**	**PC1**	**PC0**	**M**		**PM**	**UM**	**OM**	**ZM**	**DM**	**IM**
				Rounding Control		**Precision Control**				**Exceptions**					

I primi 6 bit (da 0 a 5) di Control e Status Word si riferiscono alle sei differenti Exception. Le Exception sono situazioni "anomale", vale a dire situazioni verificatesi durante i calcoli che il

programmatore può scegliere se accettare come normali oppure trattare a gusto suo con una routine apposita. Al verificarsi di queste è quindi possibile far due cose:

- generare una richiesta di interrupt alla CPU che andrà così ad eseguire una routine che il programmatore dirà alla CPU dove si trova;
- far svolgere automaticamente una sequenza predefinita di operazioni direttamente alla FPU.

I 6 bit della Control Word servono proprio a scegliere fra queste due possibilità per ognuna delle Exception: se lo metteremo ad 1, abiliteremo l'interrupt al verificarsi di quell'exception, altrimenti verrà svolta automaticamente una certa operazione. Le Exception, verranno trattate nel paragrafo successivo. I 6 Corrispondenti bit della Status Word ci informano invece se quella Exception si è verificata.

Proseguendo con i bit successivi, nella Status Word troviamo:
IR: Interrupt Request. Indica che vi è in corso una richiesta di interrupt alla CPU.
C3,C2,C1,C0: rappresentano un codice a 4 bit generato dalla FPU dopo alcune operazioni di confronto (Compare, Test, Remainder, Examine) che ci informa del risultato.
TOP: è la copia del TOP of Stack della coda circolare.
B: NEU Busy: riporta il valore del segnale Busy che indica che l'unità numerica è ancora impegnata nello svolgimento dei calcoli dell'ultima istruzione. È il segnale che viene inviato alla CPU per indicarle di attendere.

Nei bit successivi, della Control Word troviamo invece:
M: interrupt Mask. Permette di disabilitare globalmente tutte le possibili richieste di interrupt alla CPU.
PC: Precision Control: Definisce la precisione richiesta nelle operazioni su numeri reali. Viene gestita automaticamente dalla FPU.

Precision Control	
00	24 bit
01	Riservato
10	52 bit
11	64 bit

RC: Rounding Control: Definisce il modo di effettuare gli arrotondamenti (pensate alle differenze che corrono fra le funzioni del linguaggio "C" Floor, Ceil, Round).

Rounding Control	
00	Arrotonda al più vicino o pari
01	Round Down (verso – infinito)
10	Round Up (verso + infinito)
11	Chop (Arrotonda verso lo 0)

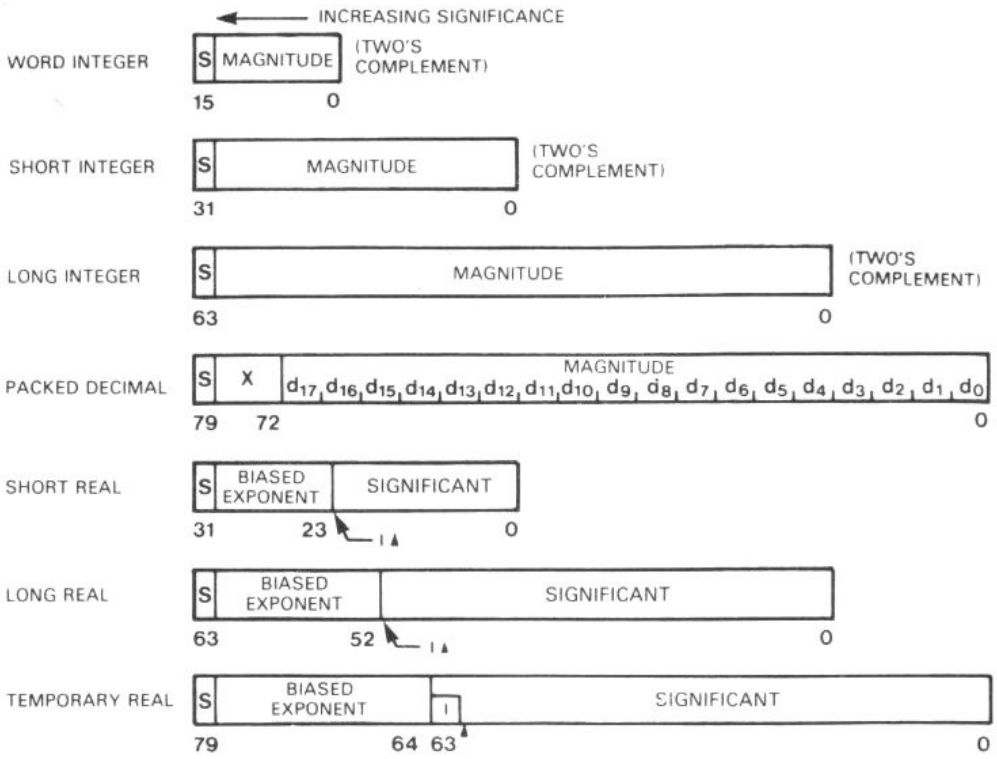
IC: Infinity Control. 0= Projective, 1= Affine. (La documentazione Intel non è esauriente al riguardo).

Instruction e Data Pointer

Il coprocessore non ha registri interni analoghi all'8086. è comunque in grado di indirizzare dati in memoria e conoscere l'indirizzo, oltre che il codice operativo dell'istruzione in corso. Per far ciò sfrutta la generazione indirizzi fatta dal microprocessore in un modo che verrà illustrato più avanti.

Questi indirizzi vengono memorizzati come un unico indirizzo fisico (a 20 bit) in registri detti Instruction e data pointer.
Nella figura seguente è illustrata l'organizzazione di questi registri.

Bit 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Instruction Pointer bit 15:0															
Instruction P. bit 19:16				0	Codice Operativo dell'Istruzione bit 10:0										
Data Pointer bit 15:0															
Data Pointer bit 19:16				0	0	0	0	0	0	0	0	0	0	0	0



Formato delle differenti tipologie di dati

Data Types

I tipi di dati gestiti dall'FPU sono Interi a 16,32 e 64 bit, i reali a 32, 64 e 80 bit e i numeri BCD a 18 cifre + segno.
Sui numeri interi c'è poco da dire: si tratta di normali interi con segno nei quali, quindi, il bit più significativo rappresenta il segno.
Sui numeri BCD va detto che ognuno di questi numeri occupa 80 bit. Siccome ogni cifra occupa 4 bit, in questo spazio entrerebbe un numero a 80/4=20 cifre. La scelta però è quella di memorizzare il segno che occupa 1 bit e di sprecare 7 bit del byte più significativo raggiungendo quindi un totale di 18 cifre + segno.
Sui reali c'è un po' da parlare.
Un reale decimale scritto in forma esponenziale è un numero del tipo -1234,5678 x 10^{+8}.
Chiameremo Significant o Mantissa la parte -1234,5678 ed Exponent o Esponente la parte +8 cioè l'esponente di 10.
In binario il discorso in partenza è molto simile. L'equivalente potrebbe essere –1010,1101 x 2^{+1000}.
Un numero reale al suo interno è quindi composto dalle parti Significant, Exponent e Segno del Significant. Vi sono però degli accorgimenti. Innanzitutto per evitare ambiguità nei confronti si mette la virgola sempre in una stessa posizione. Si vuole cioè evitare che due numeri come +10,11x2^{+1001} e +101,1x2^{+1000}, siccome differiscono nelle due mantisse e nei due esponenti

sembrino diversi (i numeri sono uguali, abbiamo solo spostato la virgola ed aggiustato l'esponente).
La regola seguita prevede che la virgola stia sempre dopo il primo 1 della mantissa (nell'esempio precedente il numero andrebbe perciò scritto $+1{,}011x2^{+1010}$). Bene, direte! Ma non ci si accontenta di ciò.
Siccome in binario le cifre sono solo 0 o 1, e siccome tutte le mantisse hanno un 1 prima della virgola (vi sono alcune eccezioni che però vengono gestite diversamente), chi ce lo fa fare di scriverci questo 1 prima della virgola ?
Insomma, per farla breve, nella mantissa del numero precedente viene solo scritto + ,011 risparmiandoci un bit. Questo bit risparmiato in realtà è utile perché ci permette, a parità di tutto, di avere una cifra significativa in più. L'1 prima della virgola che non viene fisicamente scritto nei registri, ma di cui nei calcoli si tiene conto, prende il nome di **Uno Implicito**.
Per quanto riguarda l'esponente c'è un'altra piccola complicazione: anziché essere un normale intero con segno è polarizzato, vale a dire al normale numero intero con segno viene sommata una costante 011111..111 pari alla dimensione dell'esponente.
Facciamo un esempio sennò le idee restano troppo confuse. Immaginiamo di avere 7 bit per la mantissa, 4 per l'esponente ed 1 per il segno della mantissa:

segno mantissa	esponente				mantissa						

Il numero binario + 1,010110101011 x 2^{+0100}, si scriverebbe:

1) segno della mantissa 0 (la regola è la solita 0 = +, 1 = -).
2) la costante di polarizzazione dell'esponente si ottiene facendo seguire un unico 0 da tanti 1 quanti posti rimangono: 0111.
3) l'esponente: +0100 in formato complemento a due è sempre 0100. A questo punto sommiamogli la costante di polarizzazione ed otteniamo : 0100+0111=1011.
4) Per la mantissa abbiamo a disposizione 7 bit. Togliendo l'1 prima della virgola la mantissa è ,010110101011. Teniamoci adesso le prime 7 cifre cioè: 0101101.
5) In conclusione il numero verrebbe scritto così:

Segno mantissa	esponente				mantissa						
0	1	0	1	1	0	1	0	1	1	0	1

Come abbiamo già detto, i formati reali gestiti sono 3:

1° **Short Real**: 32 bit totali (1 bit segno mantissa, 8 bit di esponente e 23 bit restanti + 1 implicito, di mantissa).
2° **Long Real**: 64 bit totali (1 bit segno mantissa, 12 bit di esponente e 51 bit restanti + 1 implicito, di mantissa).
3° **Temporary Real**: 80 bit totali (1 bit segno mantissa, 16 bit di esponente e 64 bit restanti di mantissa : fa eccezione in quanto **non utilizza l'1 implicito**).

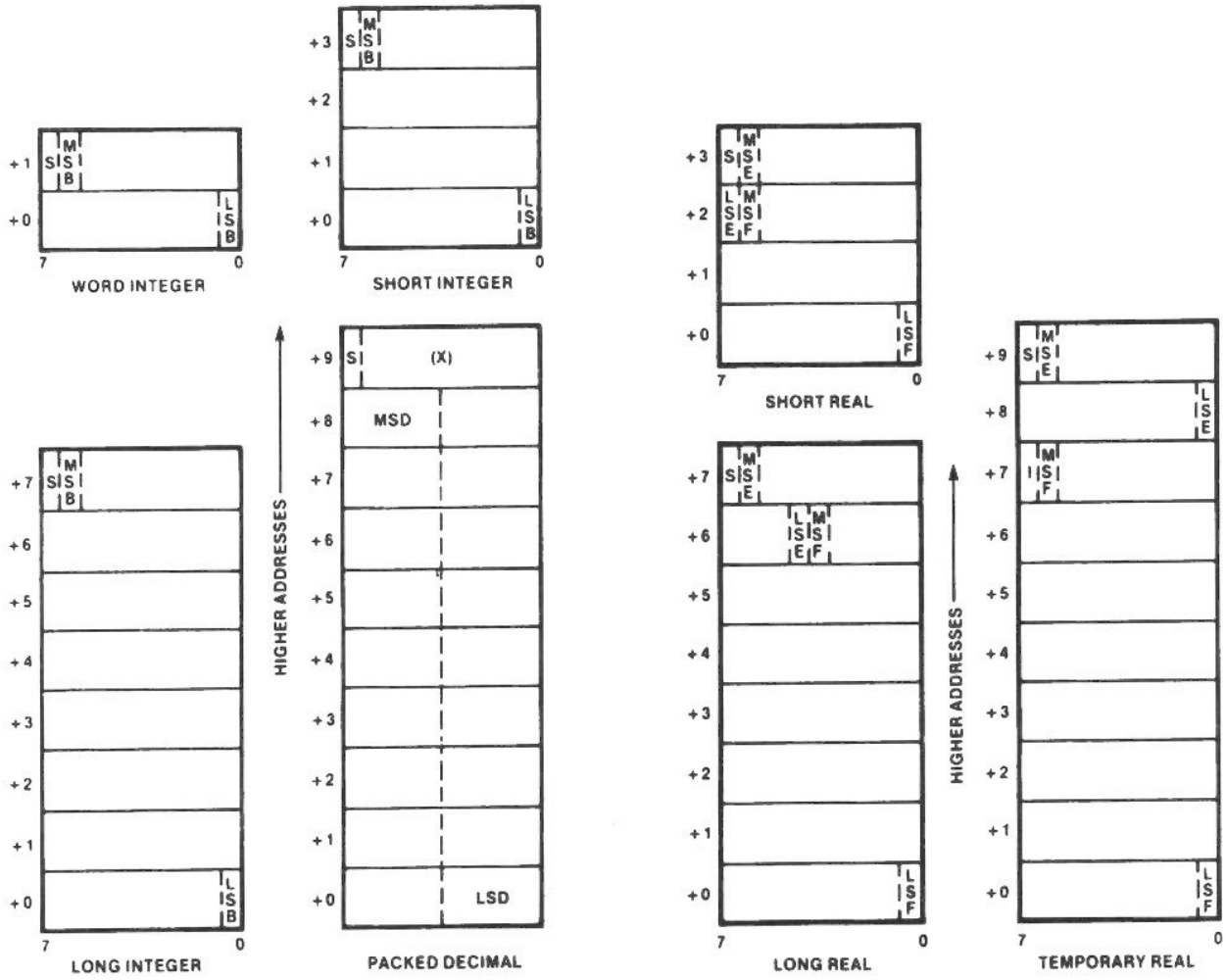
Questo modo di scrivere i numeri reali è oggetto dello standard IEEE 754 che è stato adottato da tutti i produttori di computer a partire dagli anni '80. Il formato Short Real corrisponde a quello delle variabili **float** in linguaggio C. Il Long Real al **Double**.
Il Temporary Real è poco usato nei programmi ma è il formato in cui vengono riportate nei registri interni a 80 bit tutte le variabili prima di fare i calcoli (che però hanno un'accuratezza legata alla dimensione del dato iniziale).

La rappresentazione dei numeri reali prevede anche la possibilità di memorizzare numeri particolari come + e – infinito, NAN, numeri denormalizzati, + e – zero.
Innanzitutto bisogna precisare per ognuno di questi numeri "particolari". Con + e meno infinito non si intende propriamente la stessa cosa che in matematica. + infinito è semplicemente un numero che supera il limite massimo per la dimensione del reale in uso, analogo il discorso per – infinito.
La sigla NAN sta per Not A Number e corrisponde ad entità non numeriche che vengono generate nel corso dei calcoli. All'interno dei NAN distinguiamo Indefinite: una codifica che rappresenta alcune forme indeterminate (infinito – infinito, zero per infinito o il viceversa) ed anche risultati derivanti da situazioni come "operazione su registro marcato vuoto", registro destinatario non vuoto (in genere a causa di stack overflow), estrazione di radice quadrata di un numero negativo, ecc. .
I numeri Denormalizzati sono numeri che non rispettano la normale condizione dell'uno implicito e della posizione della virgola. Sono numeri molto piccoli che vengono accettati in questa forma, con un ridotto numero di cifre significative onde evitare un peggiore arrotondamento a 0.

Codifica dei numeri reali

	Tipo		Segno	Esponente polarizzato	Significant
Positivi	NAN		0	11.....111	Da 111...111 A 000...001
	+ ∞		0	11.....111	000...000
	Reali Normali		0	Da 111...110 A 000...001	Da 111...111 A 000...000
	Reali Denormalizzati		0	000...000	Da 111...111 A 000...001
	+ Zero		0	000...000	000...000
Negativi	- Zero		1	000...000	000...000
	Reali Denormalizzati		1	000...000	Da 111...111 A 000...001
	Reali Normali		1	Da 111...110 A 000...001	Da 111...111 A 000...000
	- ∞		1	11.....111	000...000
	NAN		1	11.....111	000...001
		Indefinite	1	11.....111	100...000
			1	11.....111	111...111

Questa scelta di utilizzare alcune combinazioni di mantissa ed esponente per rappresentare i suddetti "numeri particolari" riduce seppur di poco l'intervallo che significant ed exponent possono assumere. Ad esempio un exponent di 8 bit coprirà l'intervallo +126 :-126 anziché + 127 : -128.



Disposizione dei dati in memoria

Vediamo adesso più concretamente un esempio che illustra come opera la denormalizzazione. Supponiamo che il risultato di un'operazione fra short real sia:

segno	esponente	1 implicito	Significant
0	-131	1	01011110001010101010111

Come detto, i calcoli internamente vengono condotti con la massima precisione di 80 bit maggiore dei 32 bit del risultato. Calcolato questo viene rilevato che l'esponente –131 è fuori dall'intervallo consentito da quel formato. Anziché arrotondarlo brutalmente a 0, si opera una serie di shift a destra con incrementi dell'esponente, fino a che l'esponente è rientrato nei limiti consentiti. La prima volta che si fa lo shift a destra come bit entrante vi sarà un 1 (l'uno implicito). Le altre volte uno 0. Ecco la serie di operazioni che verrebbe svolta:

segno	esponente	1 implicito	Significant
0	-131	1	01011110001010101010111
0	-130	0	10101111000101010101011
0	-129	0	01010111100010101010101
0	-128	0	00101011110001010101010
0	-127	0	00010101111000101010101
0	-126	0	00001010111100010101010

Il numero a questo punto ha 5 cifre significative in meno ma comunque rappresenta il risultato meglio di un semplice 0.

Exceptions

Vediamo adesso le sei exceptions associate ai primi sei bit di Control e Status Word.

- ***Invalid Operand*** (IM, IE : bit 0). Corrisponde a situazioni come stack overflow, stack underflow (troppi dati inseriti nello stack circolare o troppi prelevati. Da non confondere con overflow e underflow); operazioni indeterminate : 0/0, ∞-∞, ecc.; uso di un NAN come operando. Se l'exception è mascherata (vale a dire non è abilitato l'interrupt corrispondente) al risultato viene assegnato il valore NAN che si propaga nei risultati successivi.
- ***Denormalized Operand*** (DM, DE : bit 1). Uno o entrambi gli operandi sono numeri denormalizzati. Se l'eccezione è mascherata l'elaborazione continua normalmente.
- ***Zero Divisor*** (ZM, ZE : bit 2). In un quoziente il divisore è 0 mentre il dividendo è un numero né infinito né nullo. Se l'eccezione è mascherata, al risultato viene assegnato il valore infinito.
- ***Overflow*** (OM, OE : bit 3). Il risultato è troppo grande per il formato reale utilizzato. Se l'exception è mascherata il risultato assume la codifica di infinito.
- ***Underflow*** (UM, UE : bit 4). Il risultato non nullo, è troppo piccolo per il formato reale utilizzato. Se l'exception è mascherata viene attuato una serie di shift a destra attuando uno "shift graduale" e producendo un risultato denormalizzato.
- ***Inexact Result*** (PM, PE : bit 5). Il risultato non è rappresentabile con esattezza nel formato adottato. Il risultato viene quindi arrotondato in base alle impostazioni della Control Word e, se l'exception è mascherata, l'elaborazione continua normalmente.

Esecuzione delle Istruzioni

I codici operativi delle istruzioni sono riconoscibili sia dalla CPU che dalla FPU ma vengono eseguite solo da una delle due oppure in parte da una ed in parte dall'altra.
Per distinguerle bisogna osservare il primo byte di ogni codice operativo: quelle che richiedono l'intervento del coprocessore hanno i primi 5 bit pari a 11011.

7	6	5	4	3	2	1	0
1	1	0	1	1	x	x	x

Tale byte (più esattamente 5 bit), prende il nome di "Codice di Escape".

Nell'eseguire le istruzioni avviene una strana collaborazione fra CPU e FPU che ormai non esiste più nei microprocessori correnti. Nasce dal fatto che, non avendo al suo interno registri di segmento, puntatori ed indice, la FPU non è in grado di calcolare gli indirizzi di memoria. Per ovviare li genera la CPU e "glieli passa".

1° Nelle operazioni di lettura in memoria di dati per la FPU la CPU emette sul bus l'indirizzo del primo byte da leggere ed inizia la lettura come se il dato fosse diretto a se stessa. La FPU, che è collegata allo stesso Data ed Address Bus, si prende dall'address bus questo indirizzo e lo memorizza internamente. Si legge anche il primo byte di operando. A questo punto è in grado di proseguire la lettura degli altri byte che si trovano ad indirizzi successivi. Prende il controllo del bus, incrementa l'indirizzo precedente e legge il byte seguente. Continua così fino al completamento della lettura.

2° Nelle Operazioni di Scrittura di un dato, la CPU inizialmente esegue una lettura fittizia all'indirizzo dove dovrà essere scritto il risultato. Nessuno prende il dato, infatti quest'operazione serve solo a far prendere l'indirizzo alla FPU. Questa se deve fa le sue operazioni e quando ha terminato, utilizzando quell'indirizzo, sempre col meccanismo di incrementarlo pian piano, scrive il risultato in memoria. L'operazione di lettura fittizia iniziale prende il nome di "Dummy Read".

Set delle Istruzioni

Al fine di consentire la sincronizzazione CPU FPU nel corso dello svolgimento del programma, è necessario che la CPU attenda la FPU quando questa è impegnata nello svolgimento di un'istruzione (la FPU normalmente attende la CPU, è giusto che avvenga anche il viceversa). Per ottenere ciò, prima di ogni istruzione che riguarda il coprocessore, cominciante quindi con un codice di Escape, va messa un'istruzione Wait. Questa istruzione informa la CPU che all'esecuzione della istruzione seguente, dovrà fare attenzione al segnale BUSY per attendere che la FPU finisca. A questo riguardo va anche detto che alcuni *assemblatori* inseriscono automaticamente una Wait prima di ogni istruzione per la FPU.
Il modo d'indirizzamento per i registri dati è del tipo Top Relative, vale a dire rispetto al Top dello Stack. ST(0) sarà il primo dato che verrebbe estratto dallo Stack; ST(1) il secondo e così via. In alcuni casi l'istruzione non richiede parametri, in quanto fa riferimento ad un unico registro (in genere ST(0)).

Le istruzioni per la FPU si possono suddividere nei seguenti gruppi:

- Trasferimento dati
- Aritmetiche
- di Comparazione
- Trascendenti
- Caricamento di Costanti
- Di Controllo

Regola generale: quando un'istruzione può operare su numeri di tipo diverso, se il suo nome inizierà per "F", agirà su reali; se inizierà per FI, su interi; infine se inizierà per FB su numeri BCD (es. FLD esegue il caricamento di un reale con push nello stack; FILD esegue il caricamento di un intero FBADD esegue il caricamento di un numero BCD).

Istruzioni di Trasferimento Dati

FLD operando sorgente	(Float Load) Legge un operando reale, lo aggiunge allo stack dopo averlo convertito in Temporary Real
FST op. destinazione	(Float Store) Il numero reale contenuto nel registro Top dello Stack ST(0) viene copiato nell'operando destinazione
FSTP op. destinazione	(Float Store and Pop) Come la precedente in più la pop del dato dallo Stack
FILD op. sorgente	Legge un operando intero e lo aggiunge allo stack
FIST op. destinazione	Il numero intero contenuto nel registro Top dello Stack ST(0) viene copiato nell'operando destinazione
FISTP op. destinazione	Come la precedente in più la pop del dato dallo Stack
FBLD op. sorgente	Legge un operando BCD a 18 cifre e lo aggiunge allo stack
FBSTP op. destinazione	Pop del dato BCD a 18 cifre contenuto nel registro Top dello Stack ST(0) e scrittura nell'operando destinazione
FXCH ST(i)	(Exchange) Scambia il registro ST(i) con ST(0)

Istruzioni Aritmetiche

Operazioni svolte:

ADD	Sorgente + Destinazione -> Destinazione
SUB	Sorgente – Destinazione -> Destinazione
SUBR	Destinazione - Sorgente -> Destinazione
MUL	Sorgente x Destinazione -> Destinazione
DIV	Sorgente : Destinazione -> Destinazione
DIVR	Destinazione : Sorgente -> Destinazione

Per queste istruzioni gli operandi si possono specificare in questi modi:

Indirizzamento	Mnemonico	Esempio	Operandi utilizzati
Stack	Fop	FADD	ST(1),ST(0)
Registro	Fop	FADD ST(2),ST(0)	ST(i),ST(0) oppure ST(0),ST(i)
Registro + Pop	FopP	FADDP ST(2),ST(0)	ST(i),ST(0)
Reale memoria	Fop	FADD alfa	ST(0), reale
Intero Memoria	FIop	FIADD beta	ST(0), intero

FADD	(Add Real) sorgente+destinazione -> destinazione
FADDP	(Add Real & Pop) sorgente+destinazione -> destinazione; Pop
FIADD	(Integer Add) sorgente+destinazione -> destinazione
FSUB	(Subtract Real) sorgente-destinazione -> destinazione
FSUBP	(Subtract Real & Pop) sorgente-destinazione -> destinazione; Pop
FISUBP	(Integer Subtract) sorgente+destinazione -> destinazione
FSUBR	(Subtract Real Reverse) destinazione-sorgente -> destinazione
FSUBRP	(Subtract Real & Pop Reverse) destinazione-sorgente -> destinazione; Pop
FISUBR	(Integer Subtract Reverse) destinazione-sorgente -> destinazione
FMUL	(Multiply Real) sorgente x destinazione -> destinazione
FMULP	(Multiply Real & Pop) sorgente x destinazione -> destinazione; Pop
FIMUL	(Integer Multiply) sorgente x destinazione -> destinazione
FDIV	(Divide Real) sorgente : destinazione -> destinazione
FDIVP	(Divide Real & Pop) sorgente : destinazione -> destinazione; Pop
FIDIV	(Integer Divide) sorgente : destinazione -> destinazione
FDIVR	(Divide Real Reverse) destinazione: sorgente -> destinazione
FDIVRP	(Divide Real & Pop Reverse) destinazione: sorgente -> destinazione; Pop
FIDIVR	(Integer Divide Reverse) destinazione: sorgente -> destinazione

Altre Istruzioni Aritmetiche

FSQRT	Esegue la radice quadrata di ST(0) e la mette al suo posto
FSCALE	Somma all'esponente del reale in ST(0) l'intero che sta in ST(1). Equivale a moltiplicare ST(0) per $2^{ST(1)}$.
FPREM	Calcola il modulo di ST(0):ST(1).
FRNDINT	Arrotonda ST(0) e lo converte in intero. Le modalità dell'arrotondamento dipendono dai bit *Rounding Control* della Control Word.
FXTRACT	Spezza ST(0)in due parti: esponente e significant. La prima va a sostituire ST(0) la seconda viene quindi aggiunta con una push.
FABS	ST(0)=Valore assoluto di ST(0)
FCHS	Cambia segno a ST(0)

//fine 3

Table 4b. Condition Code Interpretation after FPREM Instruction As a Function of Divided Value

Dividend Range	Q_2	Q_1	Q_0
Dividend < 2 * Modulus	C_3[1]	C_1[1]	Q_0
Dividend < 4 * Modulus	C_3[1]	Q_1	Q_0
Dividend ≥ 4 * Modulus	Q_2	Q_1	Q_0

NOTE:
1. Previous value of indicated bit, not affected by FPREM instruction execution.

Istruzioni di Comparazione (poco usate)

FCOM sorgente	Confronta ST(0) con l'operando sorgente. Il risultato finisce nei bit C3:C0 della Status Word
FCOMP sorgente	Come la precedente + pop finale
FCOMPP	Confronta ST(0) con ST(1) ed esegue due pop
FICOM sorgente	Converte in reale l'operando sorgente e lo confronta con ST(0)
FICOMP	Come la precedente + Pop
FTST	Confronta ST(0) con 0
FXAM	Riporta in C3:C0 un codice relativo al tipo di numero presente in ST(0)

Istruzioni Trascendenti

FPTAN	Calcola la tangente trigonometrica di ST(0) che deve essere compreso fra 0 e Π/4. Il risultato è il quoziente ST(1)/ST(0). Alla fine vi sono due operandi nello Stack al posto di uno.
FPATAN	Calcola l'arco tangente del quoziente ST(1)/ST(0). Questo rapporto deve essere minore di 1.
F2XM1	$(2^X - 1)$. Calcola $2^{ST(0)} - 1$. ST(0)dev'essere compreso fra 0 e 0,5
FYL2X	Calcola ST(1)*Log_2 ST(0)
FYL2XP1	Calcola ST(1)*Log_2 (ST(0)+1). 0<ST(0) <radice di 2/2

Istruzioni di caricamento di Costanti

Per lo svolgimento dei calcoli servono alcune costanti fondamentali. La FPU dispone di 1, 0 Pi Greco, logaritmo in base 2 di **"e"** e di 10, logaritmo in base 10 di 2 **"e"** in base **"e"** di 2. Nelle sigle T sta per Ten=10, LG e LN per logaritmo in base 10 ed e. Naturalmente queste costanti vengono inserite con la massima precisione cioè quella Temporary Real che equivale ad avere le costanti con circa 19 cifre decimali di precisione.

FLDZ	(load zero) push nello stack di 0
FLD1	(load uno) push nello stack di 1
FLDPI	(load Π) push nello stack di pi greco
FLDL2T	push nello stack di $log_2 10$
FLDL2E	push nello stack di $log_2 e$
FLDLG2	push nello stack di $log_{10} 2$
FLDLN2	push nello stack di $log_e 2$

Table 4a. Condition Code Interpretation

Instruction Type	C_3	C_2	C_1	C_0	Interpretation
Compare, Test	0	0	X	0	ST > Source or 0 (FTST)
	0	0	X	1	ST < Source or 0 (FTST)
	1	0	X	0	ST = Source or 0 (FTST)
	1	1	X	1	ST is not comparable
Remainder	Q_1	0	Q_0	Q_2	Complete reduction with three low bits of quotient (See Table 4b)
	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

NOTES:
1. ST = Top of stack
2. X = value is not affected by instruction
3. U = value is undefined following instruction
4. Q_n = Quotient bit n

Istruzioni di Controllo

FINIT	Inizializza il coprocessore in modo simile ad un reset. Azzera la coda circolare. Molto usata
FDISI	Disabilita il flag di interrupt enable (bit Control Word: M)
FENI	Abilita il flag di interrupt enable (operazione inversa della precedente)
FLDCW	Load Control Word : legge dalla memoria una word che poi inserisce nella control word
FSTCW	Store Control Word : scrive in una word della memoria la Control Word
FSTSW	Store Status Word : scrive in una word della memoria la Status Word
FCLEX	Azzera tutti i flag di exception (Control Word), di richiesta di interruzione e di busy
FSAVE	Salva a partire dall'indirizzo di memoria specificato come operando i registri dati, e quelli di environment
FRSTOR	Operazione inversa della precedente: legge le informazioni precedenti da memoria
FSTENV	Salva in memoria i registri che costituiscono l'environment
FLDENV	Legge l'environment
FINCSTP	Incrementa ST : attenzione il registro resta marcato vuoto
FDECSTP	Decrementa ST
FFREE	Marca come vuoto il registro specificato, agendo sulla corrispondente Tag Word
FNOP	Nop

Durata delle istruzioni

Senza addentrarci tanto nel calcolo della durata delle istruzioni ci basta sapere che le istruzioni più complesse durano centinaia e in qualche caso un migliaio di Tstates (FYL2X dura da 900 a 1100 Tstates; FYL2XP1 dura da 700 a 1000 Tstates; FPATAN da 250 a 800). A parte queste che costituiscono casi limite, in generale tutte sono comunque più lente di quelle 8086.

Coprocessori successivi

IL set delle istruzioni è stato ampliato specie per le funzioni trascendenti. La durata delle singole istruzioni risulta anche più che dimezzata.

Esempi:

Moltiplicazione dei due float alfa, beta. Risultato in gamma (alfa, beta e gamma sono le label che ne contrassegnano l'indirizzo iniziale in memoria).

Wait ;una wait davanti ad ogni istruzione per il coprocessore. Questa non sarebbe necessaria
Finit ; serve ad inizializzare il coprocessore. Stack vuoto
Wait
Fld alfa ; leggo alfa e push nello stack
Wait
Fld beta ; leggo alfa e push nello stack
Wait
Fmul St(1),St(0) ; gli operandi sono i due primi dati nello stack; Il risultato è il nuovo ST(0)
Wait
FstP gamma ; St(0) va in gamma. Pop di St(0). È rimasto un dato nello stack

Algoritmo per il calcolo degli esponenziali

Il ridotto numero di funzioni messo a disposizione per realizzare le funzioni trascendentali crea qualche problema per l'individuazione dell'algoritmo per il calcolo degli esponenziali. Per questo motivo vi fornisco degli elementi ulteriori per realizzarlo.

La funzione base per il calcolo degli esponenziali è F2XM1 che però ha il limite di accettare un argomento compreso fra –1 ed 1 (nell'8087 era limitato a –1/2 , +1/2)

Il problema è quindi come estenderne il campo di applicabilità ad argomenti di modulo maggiore. Ricordiamoci la beneamata funzione FSCALE che somma a ST(0) l'intero che si deve trovare in ST(1). Il suo effetto equivale a moltiplicare ST(0) per $2^{ST(1)}$.

Ricordiamoci allora di una proprietà fondamentale delle potenze : $2^{a+b} = 2^a x 2^b$.

Poniamo di dover calcolare 2^n , essendo n un numero reale qualsiasi. Poniamo n= d+i, dove d è la sua parte decimale ed i quella intera. Utilizziamo F2XM1 per calcolare 2^d e FSCALE per moltiplicare questo primo termine per 2^i , ottenendo così il risultato cercato.