COPROCESADOR MATEMÁTICO (FPU) 18

18.1- INTRODUCCIÓN.

Los coprocesadores matemáticos eran circuitos integrados que se añadían, opcionalmente, a los sistemas procesadores para ampliar el número de instrucciones capaces de interpretar y ejecutar, especialmente de tipo matemático. Un coprocesador se encarga de realizar operaciones con números reales representados según algún estándar, habitualmente el IEEE 754. Los coprocesadores funcionan de dos formas diferentes:

1. Los *memory-mapped* funcionan de forma que sus operandos e instrucciones son colocadas en unas posiciones de memoria determinadas por parte del microprocesador. Cuando tiene el resultado, genera una interrupción y el procesador recoge el resultado de otra posición de memoria. Tienen la ventaja de que son independientes del microprocesador, como por ejemplo los *Weitek*. Evidentemente, se trata de coprocesdores externos. Como coprocesadores matemáticos están prácticamente en desuso; sin embargo, algún otro coprocesador funciona también de esta forma.

2. Otros, sin embargo, comparten el flujo de instrucciones con el microprocesador, de forma que cuando el microprocesador detecta una instrucción que no es suya, la envía al coprocesador, que la ejecuta y devuelve los resultados al microprocesador. Este es el caso de los coprocesadores internos como ocurre en el Pentium en el que las FPU que van incluidas dentro de los microprocesadores

Cada vez es mayor el número de programas y aplicaciones que requieren la actuación de un coprocesador matemático. Se citan algunas populares:

- Programas de simulación.
- Hojas electrónicas.
- CAD-CAM.
- Gráficos.
- Control numérico.
- Robótica.
- Visión computerizada.
- Inteligencia artificial.

18.2- ESTRUCTURA INTERNA.

El coprocesador trabaja internamente sólo en formato real, por lo que cualquier carga en los registros del coprocesador provocará que dicho valor sea convertido a coma flotante.

Sus registros están estructurados en forma de pila y se accede a ellos por el número de entrada que ocupan en la pila.

La FPU es un coprocesador que opera con unidades enteras de otros procesadores que coge sus instrucciones desde el mismo decodificador y secuenciador que la unidad de enteros compartiendo el bus del sistema con esta.. A pesar de este hecho la unidad de enteros y la FPU operan independientemente y en paralelo. (La actual microarquitectura de los procesadores Intel varía entre la variedad de familia de procesadores. Por ejemplo el procesador Pentium Pro posee dos unidades de enteros y dos FPUs mientras que el procesador Pentium tiene dos unidades de enteros y una FPU y el prcesador 486 dispone de una unidad de enteros y una FPU)

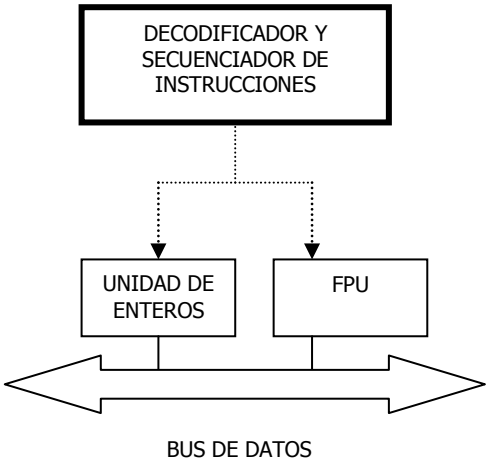


Figura 18.1 Relación entre la unidad de enteros y la FPU

Las instrucciones de ejecucción de la FPU consisten en 8 registros de datos (llamados registros de datos de la FPU) y los siguientes registros especiales:

- Registros de estado
- Registros de control
- Registros de palabra
- Registro puntero de instrucciones
- Registro puntero al último operando (puntero dato)
- Registro códigos

18.2.1 – Registros de datos

Está formada por 8 registros de 80 bits. Los registros generales R1 a R8 soportan datos con el formato normalizado de doble precisión con 80 bits. Como han de ser manejados en formato de pila, el coprocesador tiene un puntero de control de pila llamado "St". Toda interacción que tengamos que hacer con los registros del coprocesador se realiza a través del puntero de pila St, donde el ultimo valor introducido es St o St(0) y si hubiéramos rellenado todos los registros el ultimo seria St(7).

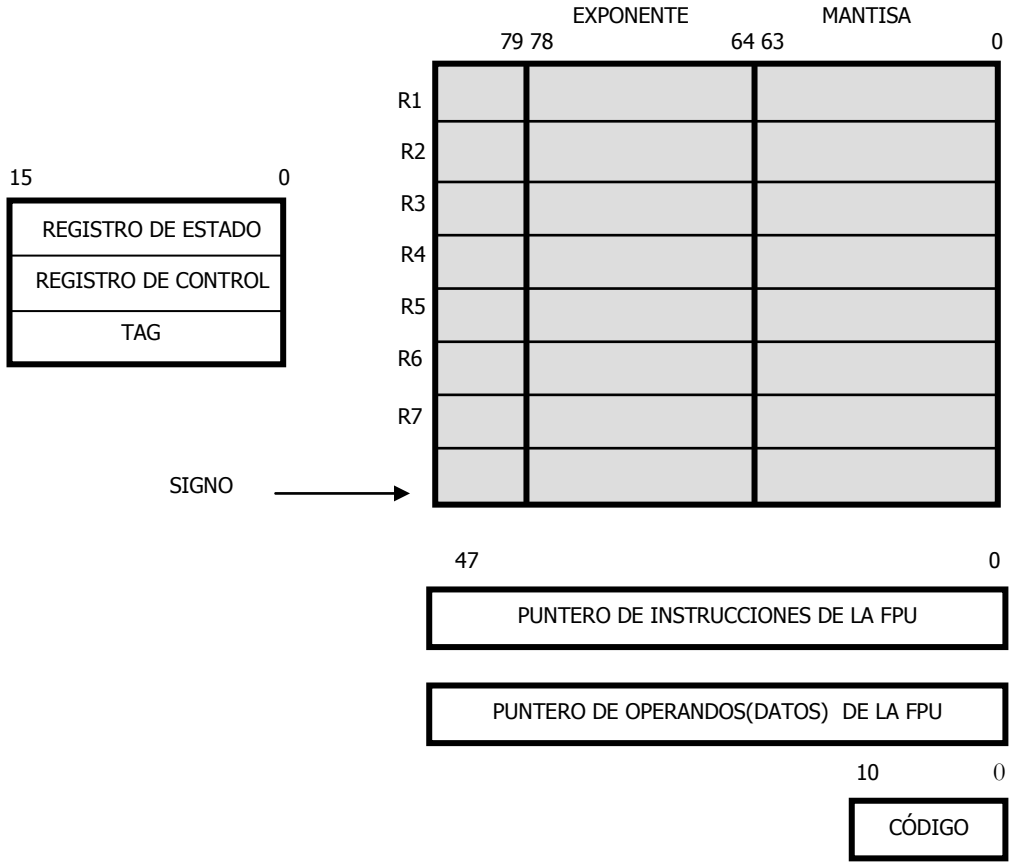


Figura 18.2 Registros del coprocesador

Podemos observar cómo dichos registros están divididos en tres secciones: signo, exponente de 15 bits y mantisa de 64 bits.

Todas las operaciones del coprocesador se realizan usando los ocho registros generales, que están organizados en forma de pila, no pudiéndose acceder a ellos de forma arbitraria.

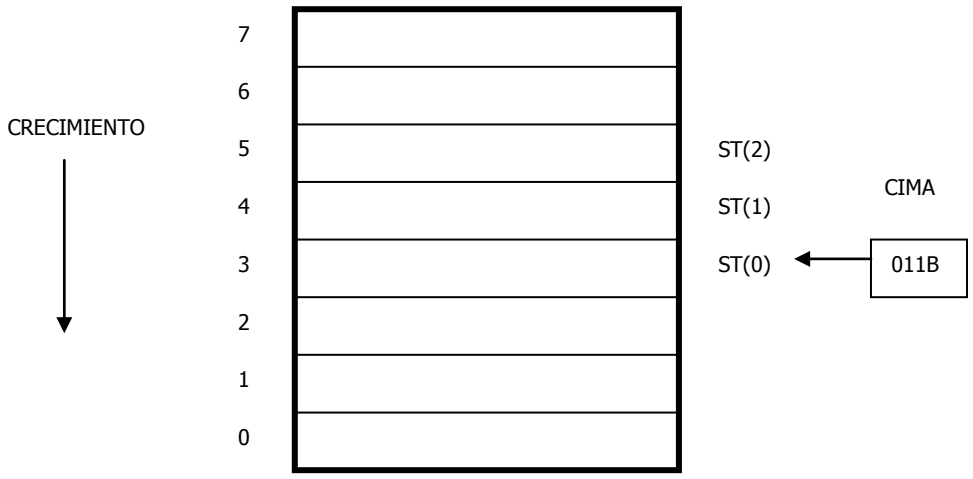


Figura 18.3 Registro de pila de datos de la FPU

Todas las direcciones de los registros de datos están relativas al registro de la cima de la pila. El número de registro de la cima de la pila activo es almacenado en el campo CIMA en la palabra de estado de la CPU. Las operaciones de carga decrementan la cima en unidad en unidad y cargan un valor en la nueva cima de la pila, cargan operaciones, cargan el valor del registro cima actual en memoria y posteriormente incrementan la cima una a una.. (Para la FPU una operación de carga es equivalente a un apilamiento, lo que se conoce como "push" y una operación de almacenamiento equivlae a una extracción o "pop").

Si una operación de carga es realizada cuando la cima está a 0, se crea una devolución del registro y el nuevo valor de la cima es puesto a 7. La excepción de desbordamiento de pila de la unidad de coma flotante, indica cuando es posible que ocurra esta devolución y borran el valor para que sea reescrito. (La excepción que indica desbordamiento de pila es la #IS).

Muchas instrucciones en coma flotante tienen varios modos de direccionamiento para permitir al programador operar implícitamente en la cima de la pila, u operar explícitamente en un registro específico relativo a la cima. Los ensambladores llevan a cabo este modo de direccionamiento, usando la expresión ST(0) o simplemente ST, para representar la cima de la pila actual ST(i) para especificar el registro "i"avo desde la cima en la cima ($0 \leq i \leq 7$). Por ejemplo si la cima contiene 011B (registro 3 en la cima de la pila), la siguiente instrucción debería añadir el contenido de dos registros en la pila (registros 3 y 5).

Como ocurre en los registros de propósito general de en los procesadores de unidades de enteros, el contenido de los registros de datos de la FPU no se ve afectado por procedimientos de llamadas, en otras palabras, los valores se mantienen a través de procedimientos limitantes. Una llamada la pueden usar los registros de datos de la FPU para pasar parámetros entre procedimientos. Los procedimientos de llamada pueden hacer referencia a los parámetros pasados a través de los registros pila usados en el actual puntero de la pila (TOP) y en las nomenclaturas ST(0) y ST(i). Suele ser habitual para un procedimiento de llamada dejar un valor de retorno o resultado en el registro ST(0) cuando se devuelve la ejecución de una llamada o programa.

El coprocesador soporta los siguientes tipos de datos:

1. Enteros de 16, 32 y 64 bits.
2. BCD empaquetados de 80 bits.
3. Números en coma flotante de 32, 64 y 80 bits.

18.2.2 – Registros de estado

Estos 16 bits de registro de estado indican la situación actual de al FPU. Los flags incluidos son los siguientes: flag-acupado o "busy-flag", puntero de la cima de la pila (TOP), flags de condición de código, de estado de error y de excepción. La FPU activa estos flags para mostrar el resultado de sus operaciones.
El contenido de los registros de estado de la FPU (referidos al estado de palabra)pueden ser almacenados en memoria usando la instrucciones FTSW/ FSTSW , FSTENV/FNSTENV, Y FSAVE/FNSAVE. Pueden ser almacenados también en el registro AX de la unidad de enteros usando las instrucciones FSTSW/FNSTSW

En la figura18.4 se muestra la distribución de los 16 bits de Palabra de Estado del coprocesador matemático.

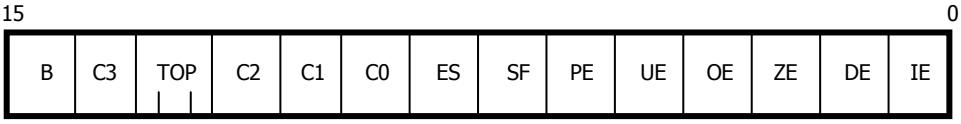


Figura 18.4. Distribución de los bits en la Palabra de Estado del coprocesador matemático.

- **TOP**: Este campo de 3 bits de la Palabra de Estado muestra el primer registro. Realiza labores similares al puntero de pila ESP. Estos bits pueden, mediante instrucciones en coma flotante tales como FLD y FSTP (éstas son similares a las instrucciones de tipo entero PUSH y POP), decrementar el TOP en 1 y colocar un valor en su respectivo registro o incrementar el TOP en 1 y retirar el registro pertinente. La pila se incrementa de manera descendente para registrar números menores. Implícitamente, la mayor parte de instrucciones direccionan el registro cima de la pila, es decir, el registro con el número almacenado en el campo TOP del registro de estado. También podremos especificar explícitamente un registro con diversas instrucciones en coma flotante. Fíjese que el registro especificado explícitamente no es absoluto sino relativo a TOP.

- **B y ES:** Siempre tienen el mismo valor e informa del estado de error. Si ES = 1 se produce una excepción no mascarable cuyo motivo lo indican los bits SP e IE. Bajo ningún concepto el bit B produce ningún dato con respecto al estado de la unidad numérica y en consecuencia al pin BUSY.

- **SF:** Diferencia entre las operaciones inválidas causadas por el desbordamiento de la pila y por otras causas. Si SF está activado el bit C1 diferencia entre un overflow (C1=1) y un underflow (C1=0). Para una interpretación del código C3-C0 de una comparación u operaciones similares adjuntamos la tabla 7.1.

- Los cuatro **flags de condición de código** (desde C0 a C3) indican el resultado de operaciones aritméticas y de comparación en coma flotante. Estos flags se usan principalmente para almacenamiento de información usada en excepciones. El flag de condición de estado C1 es usado para gran variedad de funciones. Cuando, los bits IE y SF de la FPU están activados, indicando desbordamiento de la excepción overflow o underflow (#IS), el flag C1 lo distingue de la siguiente manera: si esta a 1 overflow , sino underflow. Cuando el flag PE de palabra de estado está activado indica que se ha producido un resultado inexacto (redondeo), el flag C1 es puesto a 1 si los últimos redondeos han sido descendentes. La instrucción FXAM activa C1 para examinar el valor del signo.

El bit de condición de código C2 es usado por las instrucciones FPREM y FPREM1 para indicar una reducción correctamente (o resto parcial). Cundo se completa una reducción completamente, los flags de condición de estado C0, C3 y C1 son activados haciendo referencia a los tres bits menos significativos del cociente (Q2, Q1 y Q0 respectivamente).
Las instrucciones FPTAN; FSIN,FCOS y FSINCOS ponen el flag C2 a 1 para indicar que el operando fuente está fuera del rango permitido ($\pm 2^{63}$)

INSTRUCCIÓN	C0	C3	C2	C1
FCOM, FCOMP, FCOMPP, FICOM, FICOMP, FTST, FUCOM, FUCOMP, FUCOMPP	Resultados de comparaciones		Operandos no Comparables	0 ó #IS
FCOMI, FCOMIP, FUCOMI, FUCOMIP	No definidas			#IS
FXAM	Clase de operando			Signo
FPREM, FPREM1	Q2	Q1	0 = reducción completa 1 = reducción incompleta	Q0 ó #IS
F2XM1, FADD, FADDP, FBSTP, FCMOVcc, FIADD, FDIV, FDIVP, FDIVR, FDIVRP, FIDIV, FIDIVR, FIMUL, FIST, FISTP, FISUB, FISUBR,FMUL, FMULP, FPATAN, FRNDINT, FSCALE, FST, FSTP, FSUB, FSUBP, FSUBR, FSUBRP,FSQRT, FYL2X, FYL2XP1	No definido			Roundup ó #IS
FCOS, FSIN, FSINCOS, FPTAN	No definido		1 = operando fuente fuera de rango	Roundup ó #IS (No definido si C2=1)
FABS, FBLD, FCHS, FDECSTP, FILD, FINCSTP, FLD, Constantes de Carga, FSTP (ext. real), FXCH, FXTRACT	No definido			0 ó #IS
FLDENV, FRSTOR	Cada bit se carga de la memoria			
FFREE, FLDCW, FCLEX/FNCLEX, FNOP, FSTCW/FNSTCW, FSTENV/FNSTENV, FSTSW/FNSTSW,	No definido			
FINIT/FNINIT, FSAVE/FNSAVE	0	0	0	0

Tabla 18.1. Códigos en coma flotante para el Pentium.

- **PE:** Precisión.
- **UE:** Underflow.
- **OE:** Overflow.
- **ZE:** División por cero.
- **DE:** Operando desnormalizado.
- **IE:** Operación inválida.

} 0 = No se produce excepción
1 = Condición de excepción generada

Estos 6 flags indican que una o más excepciones en coma flotante han sido detectadas desde que los bits fueron limpiados por última vez.
Estos flags de excepcion se pueden considerar un poco pegajosos, es decir, permanecen activados hasta que son explícitamente limpiados. Para ello se utilizan las siguintes instrucciones:

- FCLEX/FNCLEX (excepciones de limpieza)
- FINIT/FNINIT (reinilializan la FPU)

- FSAVE/FNSAVE
- FRSTOR o FLDENV (para sobreescribir los flags)

18.2.3 – Saltos y movimientos condicionales en la condición de código de la FPU

La arquitectura de la FPU de Intel (comenzó con el procesador Pentium PRO) posee dos mecanismos para saltar y efectuar movimientos condicionales de acuerdo con las comparaciones de dos valores en coma flotante. Estos mecanismos se refieren a "nuevo mecanismo" y "viejo mecanismo".

El mecanismo viejo está disponible a priori en la FPU de los procesadores Pentium Pro. Este mecanismo usa instrucciones de comparación en coma flotante (FCOM, FCOMP, FCOMPP, FTST, FUCOMPP, FICOM Y FICOMP) para comparar dos valores en coma flotante y activar los flag de condición de código (C0, C3) de acuerdo con los resultados. Los contenidos de la condición de código son después copiados en los flags de estado del registro EFLAGS mediante dos pasos como se puede ver en la figura de la siguiente página.

1. La instrucción FSTSW AX mueve la palabra de estado de la FPU al registro AX.
2. La instrucción SAHF copia los 8 bits de más peso, en los cuales están incluidos los flags de condición de código, dentro de los 8 bits de menos peso de l registro EFLAGS

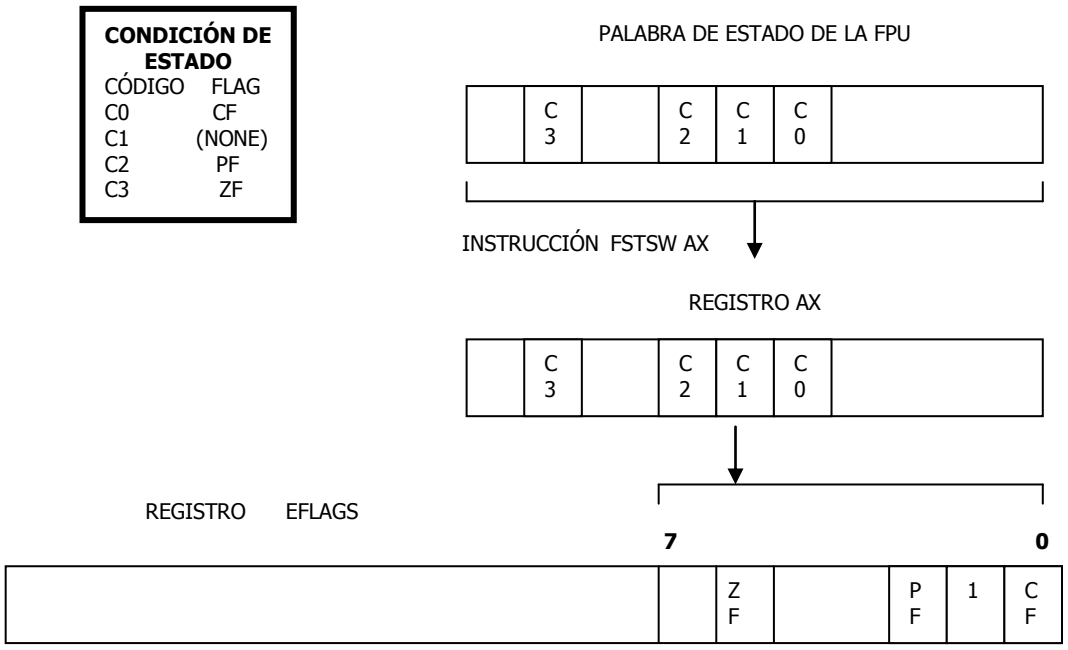


Figura 18.5 Movimiento de la condición de código al registro EFLAGS

El mecanismo nuevo está solamente disponible en el procesador Pentium Pro. Usando este mecanismo la nueva comparación en coma flotante compara y activa las instrucciones EFLAGS (FCOMI, FCOMP. FUCOMI, Y FUCOMIP) que comparan dos valores en coma flotante y activan directamente los registros EFLGS ZF, y PF. Una única instrucción sustituye tres instrucciones requeridas en el viejo mecanismo.

18.2.4 – Registros de control

Estos registros controlan la precisión de la FPU y los métodos de redondeo usados. También existen unos bits que hacen posible el enmascaramiento de excepciones. El contenido de estos registros puede ser cargado con la instrucción FLDCW y almacenado en memoria con la instrucción FSTCW/FNSTCW.
Cundo la FPU es inicializada con las instrucciones FINIT/FNINIT o FSAVE/FNSAVE, la palabra de control es puesta a 037FH, la cual enmascara todas las excepciones en coma flotante, redondea hacia el más próximo y se activa la FPU con una precisión de 64 bits.

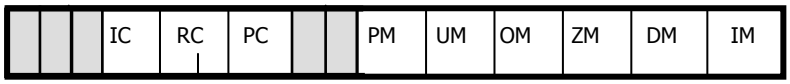


Figura 18.6. Distribución de los bits en la Palabra de Control del coprocesador matemático.

Bajo ciertas circunstancias, el Pentium genera una excepción de coprocesador. Estas excepciones pueden ser individualmente enmascaradas. Lo que es más, podemos determinar diferentes modos para precisión y redondeo. La Palabra de Control se emplea para este propósito, y su estructura la mostramos en la figura 18.4

- **IC:** No tiene significado cuando no manejan valores infinitos porque el Pentium aplica el estándar IEEE a las operaciones con coma flotante. En los terrenos de compatibilidad con el 8087 y el 80287 el bit IC está disponible pero no tiene efecto.

  El Pentium siempre maneja cantidades infinitas en el sentido de $\pm\infty$, incluso si ponemos IC a 0.

- **RC:** Los dos bits RC controlan el redondeo en el modo definido.

MODO DE REDONDEO	CAMPO RC
Redondeo al más cercano	00B
Redondeo hacia abajo	01B
Redondeo hacia arriba	10B
Redondeo a cero	11B

Tabla 18.2. Códigos en coma flotante para el Pentium.

- **PC:** El campo de control de precisión (bits 8 y 9 de la Palabra de Control de la FPU) determina la precisión (64, 53, o 24 bits) de los cálculos en coma flotante realizados por la FPU. Controla el redondeo en las instrucciones ADD, SUB, MUL, DIV y SQRT.

PRECISIÓN	CAMPO PC
Precisión simple (24 bits)	00B
Reservado	01B
Doble precision (53-Bits)	10B
Precisión extendida (64-Bits)	11B

Tabla 18.3. Tabla del campo PC en relación a la precisión.

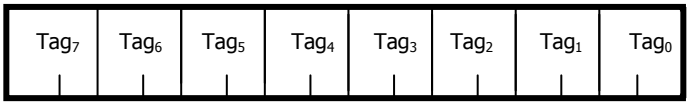
- **PM, UM, OM, ZM, DM e IM:** Controlan la generación de una excepción cada uno y su interrupción resultante. Los Pentium provocan 6 excepciones diferentes en sus operaciones con coma flotante. Podemos enmascarar las excepciones individualmente empleando los bits PM, UM, OM ZM, DM e IM. Entonces el Pentium ejecuta una rutina estándar para tratar los respectivos errores utilizando un supuesto estándar. Esto es una parte integral del chip.

18.2.5 – Flag de control de infinito

Este flag proporciona compatibilidad con el coprocesador matemático 287 de Intel; no es muy significante para versiones posteriores de la arquitectura IA-32

18.2.6 – Registros de palabra

. La FPU usa los valores de los tag para detectar desbordamientos en condiciones de overflow o underflow . Los desbordamiento de overflow ocurren cuando el puntero del campo TOP es decrementado para apuntar a un registro no vacío. Los desbordamientos de underflow se producen cuando el puntero del campo TOP es incrementado para apuntar a un registro vacío, o cuando un registro vació es referenciado como un operando fuente. Un registro no vacío viene definido como un registro que contiene valor cero (01), valor válido (00), o valor especial (10). Su estructura la mostramos en la figura 7.16. Tag_7 – Tag_0 contiene información, que identifica los contenidos de los ocho registros de datos R7-R0. El coprocesador utiliza esta información para realizar ciertas operaciones a una velocidad superior. Empleando este proceso, el Pentium puede determinar muy rápidamente ciertos valores como NAN, infinito y sin la necesidad de decodificar el valor del registro correspondiente. Mediante las instrucciones de FSTENV/FNSTENV podemos almacenar la Palabra Tag en memoria y examinarla.



00 = VÁLIDO.
01 = CERO.
10 = NAN, INFINITO, VALOR DESNORMALIZADO O FORMATO DE NÚMERO INVÁLIDO.
11 = LLENO.

Figura 18.7 Distribución de los bits en la Palabra Tag del coprocesador matemático.

18.2.7 – Punteros de instrucciones y datos

La FPU almacena instrucciones a datos (operandos) e instrucciones para las últimas instrucciones ejecutadas que no son de control. Estos punteros son almacenados en dos registros de 48 bits.

Hay que tener en cuenta que el valor de un registro puntero de dato es siempre un puntero a un operando de memoria. Si las últimas instrucciones no controladas que fueron ejecutadas no contenían un operando en memoria, el valor del registro puntero de datos es indefinido.

El contenido de los registros de los punteros de instrucciones y datos permanecen invariables cuando no se ejecuta ninguna instrucción de control. (FINIT/FNINIT, FCLEX/FNCLEX... etc). Los punteros almacenados en los registros de los punteros de instrucciones y datos contienen un offset (almacenado en los bits del 0 al 31) y un selector de segmento (almacenado en los bits del 32 al 47)

18.2.8 – Registros de código

La FPU almacena el código de las últimas instrucciones ejecutadas que no sean de control en un registro de código de 11 bits. Sólo el primer y el segundo byte de código están almacenados en el registro de código de la FPU. La figura muestra el almacenamientos de estos 2 bytes. Desde que los 5 bits de más peso del primer byte de código son los mismos para todos los códigos de unidad en coma flotante (11011B), solo los tres bits de menos peso son almacenados en este registro de código.

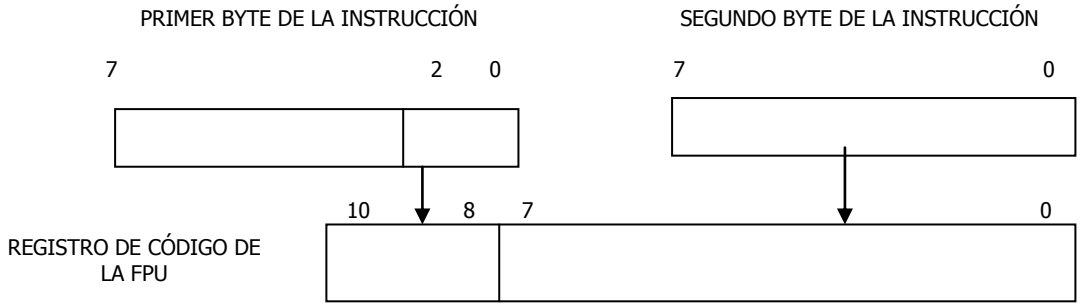


Figura 18.8 Contenido del registro código

18.3- TIPOS DE DATOS

El coprocesador puede obtener y escribir datos en memoria de los siguientes tipos.

- **Entero:** Words(16 bits), Dword(32 bits), Qwords(64 bits)
- **Real:** Words(16 bits), Dword(32 bits), Qwords(64 bits), Twords(80 bits)
- **Simple precisión en coma flotante**
- **Doble precisión en coma flotante**
- **Doble precisión extendida en coma flotante**
- **Entero con signo**
- **BCD**

A continuación se muestra una tabla de las excepciones que existen en doble precisión en coma flotante.

Excepción	Causa	Acción por defecto (si la excepción está enmascarada)
Invalid Op.	Operación sobre un Signaling NaN, formato no soportado, operación indeterminada (0*infinito, 0/0, infinito-infinito, etc.), o underflow/overflow de pila	El resultado es un Quiet NaN, un entero indefinido o un BCD indefinido.
Denormalized Op.	Al menos uno de los operandos es un Denormal, o sea, tiene el menor exponente pero una mantisa no-cero.	Continua el proceso normal
Zero Divisor	El divisor es cero mientras el dividendo es un número no cero y no infinito.	El resultado es Infinito.
Overflow	El resultado es demasiado grande para caber en el formato especificado.	El resultado es el mayor número posible o infinito.
Underflow	El resultado es no-cero pero es demasiado pequeño para caber en el formato especificado, y si está enmascarada la excepción de Underflow, la denormalización causa pérdida de precisión.	El resultado es un denormal o cero.
Resultado Inexacto (Precisión)	El resultado EXACTO no es Representable en el formato Especificado (p.ej. 1/3); el resultado es redondeado de acuerdo con el modo de redondeo.	Continua el proceso normal.

Tabla 18.4 - Tabla de excepciones

18.4- ORÍGENES DEL COPROCESADOR MATEMÁTICO EN EL PENTIUM

Esta unidad se ha rediseñado totalmente respecto a la que se usa el 486. Sin embargo, mantiene compatibilidad 100% binaria con ella. Incorpora un cauce segmentado de instrucciones de ocho etapas, que permite obtener resultados partiendo de instrucciones de coma flotante en cada ciclo de reloj. Las cuatro primeras etapas son las mismas que poseen las unidades de enteros. La quinta y la sexta, corresponden a la ejecución de las instrucciones de coma flotante. La séptima etapa se encarga de escribir el resultado en los registros adecuados y la octava realiza el informe de posibles errores que se hayan producido.

Hace uso de nuevos algoritmos que aceleran la ejecución de las operaciones e incluye elementos de hardware dedicados, como son : un multiplicador, un sumador y un divisor. Instrucciones de suma, multiplicación y carga de datos se ejecutan tres veces más rápido que en un 486.

La Unidad en Coma Flotante o FPU integrada en el Pentium amplía el número de, así como el repertorio de instrucciones. Los nuevos registros que proporciona el coprocesador son ocho generales de 80 bits cada uno y tres de 16 bits, siendo éstos últimos empleados en labores de control y presentación de estado.

18.5- INSTRUCCIONES

Estas son las instrucciones más comunes, se omiten algunas, por ser poco comunes en el mundo del tiempo-real. Los ciclos de reloj de las instrucciones, son los declarados por Intel para el Pentium básico, esto puede verse alterado en las CPU's MMX, PRO y Pentium II. Cualquier instrucción de cálculo (fadd, fsub, etc.) toman por defecto si no se le especifica otros operandos St(0) y St(1).

Leyenda: rm=Modo Real, vm=Modo Virtual, pm=Modo Protegido

INSTRUCCIÓN	EJEMPLO	CICLOS DE RELOJ
FABS	fabs	1
FADD [reg,reg]	fadd	3, 1
FADD memreal	fadd shortreal	3, 1
FADDP reg,ST	faddp st(6),st	3, 1
FIADD memint	fiadd int16	7, 4
FCHS	fchs	1
FCLEX	fclex	9+
FNCLEX	fnclex	9
FCOM	fcom	4, 1

FCOMP	fcomp	4, 1
FCOMPP	fcompp	4, 1
FICOM memint	ficom double	8, 4
FICOMP memint	ficomp darray[di]	8, 4
FCOS	fcos	18-124
FDECSTP	fdecstp	1
FDIV [reg,reg]	fdiv st(5),st	39
FDIV memreal	fdiv longreal	39
FDIVP reg,ST	fdivp st(6),st	39
FIDIV memint	fidiv warray[di]	42
FDIVR [reg,reg]	fdivr st(5),st	39
FDIVR memreal	fdivr longreal	39
FDIVRP reg,ST	fdivrp st(6),st	39
FIDIVR memint	fidivr warray[di]	42
FFREE ST(i)	ffree st(3)	1
FILD memint	fild quads[si]	3, 1
FINCSTP	fincstp	1
FINIT	finit	16
FNINIT	fninit	12
FIST memint	fist doubles[8]	6
FISTP memint	fistp longint	6
FLD reg	fld st(3)	1
FLD mem32real	fld longreal	1
FLD mem64real		1
FLD mem80real		3
FLD1	fld1	2

FLDZ	fldz	2
FLDPI	fldpi	5, 3
FLDL2E	fldl2e	5, 3
FLDL2T	fldl2t	5, 2
FLDLG2	fldlg2	5, 3
FLDLN2	fldln2	5, 3
FLDCW mem16	fldcw ctrlword	7
FLDENV mem	fldenv [bp+10]	37, 16-bit pm=32, 32-bit pm=33
FMUL [reg,reg]	fmul st(5),st	3, 1
FMULP reg,ST	fmulp st(6),st	3, 1
FIMUL memint	fimul warray[di]	7, 4
FNOP	fnop	1
FPATAN	fpatan	17-173
FPREM	fprem	16-64
FPREM1	fprem1	20-70
FPTAN	fptan	17-173
FRNDINT	frndint	9-20
FRSTOR mem	frstor [bp-94]	16-bit rm or vm=75; 32-bit rm or vm=95; pm=70
FSAVE mem	fsave [bp-94]	16-bit rm or vm=127+; 32-bit rm or vm=151+; pm=124+
FNSAVE mem	fnsave [bp-94]	16-bit rm or vm=127; 32-bit rm or vm=151; pm=124
FSCALE	fscale	20-31
FSIN	fsin	16-126
FSINCOS	fsincos	17-137
FSQRT	fsqrt	70
FST reg	fst st	1

FST memreal	fst longs[bx]	2
FSTP reg	fstp st(3)	1
FSTP mem32real	fstp longreal	2
FSTP mem64real		2
FSTP mem80real		3
FSTCW mem16	fstcw ctrlword	2+
FNSTCW mem16	fnstcw ctrlword	2
FSTENV mem	fstenv [bp-14]	16-bit rm or vm=50+; 32-bit rm or vm=48+; 16-bit pm=49+; 32-bit pm=50+
FNSTENV mem	fnstenv [bp-14]	16-bit rm or vm=50; 32-bit rm or vm=48; 16-bit pm=49; 32-bit pm=50
FSTSW mem16	fstsw statword	2+
FSTSW AX	fstsw ax	2+
FNSTSW mem16	fnstsw statword	2
FNSTSW AX	fnstsw ax	2
FSUB [reg,reg]	fsub st,st(2)	3, 1
FSUB memreal	fsub longreal	3, 1
FSUBP reg,ST	fsubp st(6),st	3, 1
FISUB memint	fisub double	7, 4
FSUBR [reg,reg]	fsubr st,st(2)	3, 1
FSUBR memreal	fsubr longreal	3, 1
FSUBRP reg,ST	fsubrp st(6),st	3, 1
FISUBR memint	fisubr double	7, 4
FTST	ftst	4, 1
FUCOM [reg]	fucom st(2)	4, 1
FUCOMP [reg	fucomp st(7)	4, 1

FUCOMPP	fucompp	4, 1
FWAIT	fwait	1-3
FXAM	fxam	21
FXCH [reg]	fxchg st(3)	1
FXTRACT	fxtract	13
FYL2X	fyl2x	22-111
FYL2XP1	fyl2xp1	22-103

Tabla 18.5. Instrucciones

18.6- HIOSTORIA DEL ERROR DE LA FPU DEL PENTIUM

El 30 de octubre de 1.994, el doctor Thomas R. Nicely, profesor de Matemáticas del Lynchburg College del Estado de Virginia (U.S.A.), enviaba un mensaje mediante correo electrónico a Andrew Schulman, coautor del libro Undocumented DOS y autor de Undocumented Windows (bien conocidos por los programadores de sistemas, por tratar de interrupciones y opciones diversas no documentadas de estos sistemas) como uno más de los comentarios o informaciones sobre este tipo de asuntos que Andrew solicitaba que le enviaran por este procedimiento.

En el mensaje, el doctor Nicely le comunicaba que mientras investigaba con el programa Matlab sobre números primos elevados con un Pentium, observó que éste le dio un resultado erróneo a una determinada operación de división con punto flotante, lo que inducía a pensar en un error en la FPU (Floating Point Unit, Unidad de punto flotante o coprocesador) del Pentium, por resultar correcta idéntica operación en los 486. Andrew envió copia del mensaje a Richard Smith, de la casa de software Phar Lap, para que pudiese comprobar el fallo en algún equipo Pentium, de los que él no disponía. Hecho esto, y a la vista de la importancia que podría tener el asunto, lo notificó públicamente la noche del 1 de noviembre en el fórum de discusión Canopus de Compuserve. Las pruebas se multiplicaron, y sencillamente desde la calculadora de Windows, resultaba fácil constatar que las matemáticas no eran exactas, al menos para el Pentium. Poco después Alex Wolfe, de la revista Electronic Engeneering Times, se interesaba por la nota que aparecía en Canopus y, comprendiendo su importancia, publicó el primer artículo escrito sobre el error, en la primera plana de la edición del 7 de noviembre de la revista. Antes, notificó lo que estaba investigando a Terje Mathisen, de Noruega, quien estaba escribiendo en USENET sobre las instrucciones del Pentium, y éste hizo público el error el 3 de noviembre en el grupo comp.sys.intel de USENET. Una vez aquí, Tim Coe, de Vitesse Semiconductor, logró describir correctamente cómo se producía el error y efectuó predicciones correctas sobre qué pares de números lo ocasionarían. El error siguió siendo discutido en este foro, y desde ahí se conoció en toda Internet, la mayor red de ordenadores que llega a grandes partes del mundo, y precursora de la no muy lejana Autopista de la Información, lo cual supone hablar de una audiencia de veinticinco millones de personas en la actualidad (en constante crecimiento). He aquí cómo la información se ha transmitido, siempre en forma de bytes y a través de los cables, hasta llegar a nuestros oídos. Pero qué es exactamente lo que falla? En términos prácticos, el error consiste en que se redondean mal los dígitos decimales desde el cuarto al decimonoveno en determinadas divisiones en coma flotante: la precisión de la respuesta es mucho menor de la esperada. Esto se produce independientemente de la precisión, simple, doble o extendida, de los números; tampoco depende de la velocidad del chip, ni de las instrucciones previas, ni del modo de redondeo. El fallo reside en la unidad de división de la FPU, y por tanto

cualquier instrucción que la utilice puede mostrar el error; no sólo la conocida FDIV, sino otras indicadas por Intel, como FPTAN, FPATAN (tangente y arcotangente en punto flotante), FYL2X, FYL2XP1 (logaritmos) y la instrucción de resto; en cambio, no fallan el seno ni el coseno, entre otros.

La causa final del error es, obviamente, humana: el olvido de unas cuantas entradas en una tabla de consulta empleada por la unidad de división. La división en punto flotante en la mayoría de los microprocesadores, como el Pentium, se realiza usando un algoritmo similar al de la caja que todos aprendimos en el colegio, y que, como aquél, produce un bit en cada ciclo de la operación. Adicionalmente, el Pentium posee un sistema de optimización que le permite deducir dos bits en cada ciclo, usando en un determinado punto de la división los restos del dividendo y del divisor como índices a una tabla de consulta de donde obtiene los siguientes dos bits. Si desafortunádamente esos restos llevan a una de las entradas que faltan en la tabla, se obtienen los dos bits incorrectos, y el resto ya se sabe... dos bits que han dado la vuelta al mundo informático. Concretamente, IBM ha indicado que las secuencias de bits peligrosas son éstas: 10001, 10100, 10111, 11010 y 11101, seguidas por una cadena de al menos 20 unos.

De lo anterior se deduce que el error sucede siempre al dividir pares de números concretos. Si dos números fallan, lo harán siempre que se repita la prueba; si no es así, nunca darán problemas. En total, considerando sólo los casos de números de simple precisión con una exactitud en el resultado inferior a la simple precisión, sólo parecen existir 1.738 pares que producen el error, y sólo 87 con una precisión de cuatro dígitos decimales. Dada la naturaleza del error, hay que añadir a estos números todos los obtenidos al multiplicar o dividir a uno o ambos números del par por una potencia de dos, y/o cambiarle el signo, ya que el error lo determinan las secuencias de bits en las mantisas, con lo que los números de igual mantisa han de ser considerados un único caso. Para la doble precisión, los casos se multiplican, y más aún si aumentamos la precisión requerida al resultado, pero al mismo tiempo la posibilidad de encontrar un error con esos números disminuye rápidamente.

18.6.1 – Mantisas, exponentes y precisiones

El Pentium, como el resto de los microprocesadores, emplea el standard IEEE 754 para representar números en punto flotante. Este tipo de almacenamiento transforma los números reales al formato signo*1.XXX*2^YYY, es decir, uno más una fracción, multiplicado por una potencia de dos, junto con el signo. El XXX se denomina mantisa; YYY es el exponente.

El standard IEEE define el número de simple precisión como aquél que tiene una mantisa de 23 bits. Esto proporciona una precisión de 24 bits (contando el primer bit) lo que equivale a aproximadamente 7 digitos decimales. Un número de doble precisión tiene una mantisa de 52 bits, unos 15 digitos decimales. La mantisa en la precisión extendida tiene 63 bits, es decir, se puede obtener una precisión de aproximadamente 18 decimales.

Para los programadores en C, una variable float es de simple precisión, mientras que un double tiene doble precisión y un long double suele tener precisión extendida (aunque no todos los compiladores soportan esta última opción).

18.6.2 – Continuando con el error

Para comprobar si un Pentium tiene el error, basta efectuar con cualquier software de calculadora u hoja de cálculo, la división de alguno de los pares de números afectados (véase ejemplo adjunto). Comparando los resultados con los reales, veremos diferencias a partir del cuarto dígito decimal. Sin embargo, es posible obtener un resultado más escandaloso al efectuar la operación x - (x/y) * y con los números afectados. Esta operación, como es fácil ver, resulta cero para cualquier x e y tomados; sin embargo, una operación efectuada en un Pentium con los x e y adecuados, devolverá un número distinto de 0. Si el resultado es erróneo, está claro que tenemos el ; si no lo es, deberemos asegurarnos de que el programa empleado utiliza las instrucciones FDIV y

similares para obtener el resultado, lo cual no es siempre seguro, aunque se sabe que lo es para la Calculadora de Windows y la hoja de cálculo Excel; por tanto el resultado de la prueba efectuada en estos programas es infalible con respecto al . Por otra parte, aunque se han desarrollado ya numerosas soluciones por software para corregirlo, no existen parches para actualizar los binarios de los programas que pudieran efectuar las operaciones prohibidas ; sólo cabe esperar que las respectivas casas de software introduzcan en el mercado una nueva versión con la leve corrección pertinente, y de momento utilizar dichos programas residentes para la corrección.
Si además de este método casero queremos información adicional a la simple verificación del error, podemos emplear el programa P87TEST.ZIP del antes mencionado Terje Mathisen, disponible en el G.U.I. Es muy de esperar que el resultado sea positivo, ya que sólo un ínfimo porcentaje de entre los primeros Pentium aparecidos carecían del error.

```
/*Este programa dará cero en cualquier procesador que no sea
Pentium, que dará 256.*/
main() double x,y,z;
x =4195835,0; y = 3145727.0;
z =x-(x/y)*y;
printf("Resultado: %f\n", z);
}
```

Intel, tras efectuar siete billones de divisiones aleatorias, indicó que el error se producía en una de cada nueve mil millones, partiendo del número de casos erróneos encontrados. El tipo de cálculos que realicemos afectará, sin duda, a la probabilidad de toparnos con un error, pero, con este dato, podríamos efectuar el siguiente razonamiento: Si ejecutamos una instrucción FDIV constantemente en un Pentium-90, al durar cada instrucción 39 ciclos (según documenta Intel) y disponer de un reloj de 90 MHz, esto es, 90 millones de ciclos, se efectuarían 2.307.692 divisiones cada segundo, y más de ocho mil millones cada hora. De esta forma, al cabo de poco más de una hora se obtendría un error. Naturalmente, estas circunstancias son imposibles. La estimación realista de Intel es la siguiente: realizando, con una hoja de cálculo, 1.000 divisiones de promedio al día, sólo obtendremos un error a los 27.000 años.

Pero sin tener en cuenta la frecuencia del error, lo importante aquí es la respuesta que Intel pueda dar a sus usuarios suspicaces. En esto Intel no puede ser alabada. Sus propios técnicos descubrieron el error ya en Junio, pero lo minimizaron, pensando que defectos los hay en todo procesador, aunque de mayor o menor importancia, y que éste no llegaría a salir a la luz, por lo que no creyó conveniente alertar a los casi dos millones de usuarios de aquel entonces (fue un grave error por su parte no tener en cuenta las leyes de Murphy). Después, a la vista de las circunstancias, trató de enmendarse, haciendo pública una descripción técnica del error y ofreciéndose a sustituir los chips erróneos: envió un fax a distribuidores de todo el mundo, cuyo contenido reproducimos en el recuadro adjunto, y que resulta cuanto menos poco serio para con los usuarios, pues indica que decide sustituir el procesador defectuoso debido a un error que sucede cada 27.000 años. Intel asegura haber reservado una partida importante de sus beneficios a lo largo del cuarto trimestre del año para la operación de sustitución, a la vez que afirma haber corregido ya el error en todos los nuevos Pentium s fabricados desde la última semana de Diciembre. A pesar de esto, los comercios (al menos en Valladolid) no están recibiendo Pentium s corregidos con regularidad, dado que a pesar de lo que se fabrique en Taiwan, lo que traiga el distribuidor a España puede ser bien diferente.

La trascendencia del error, a pesar de su difusión a través de Internet, no hubiera llegado a tanto de no haberse producido la reacción de IBM. El gigante azul sigue siendo el vendedor más importante de PCs, y por ello el hecho de que decidiera, a mediados de diciembre, retirar sus equipos con Pentium del mercado, supuso un fuerte impacto a nivel empresarial. IBM adujo que su División de Investigación, una vez al corriente del fallo, indicó que, aún siendo ciertos los datos técnicos sobre la naturaleza del error, las situaciones de riesgo podían ser mucho más numerosas que las indicadas por Intel. De hecho, según IBM, las estimaciones de Intel fallaban en sus bases, ya que una hoja de cálculo puede efectuar 5.000 divisiones cada segundo. Trabajando una media de 15 minutos diarios en una hoja de cálculo, ello supondría un error cada 24 días, o lo que es igual, a diario en una empresa con 24 ordenadores. Pero IBM no convenció al resto de fabricantes, que se han agrupado en torno a Intel, restando importancia al error y confiando en la marca. Todos

afirman haber continuado con normalidad sus ventas tras explicar la situación a sus clientes, e incluso algunas marcas dicen haber rebasado sus expectativas sobre el Pentium.

Hay realmente segundas intenciones en la decisión de IBM? El hecho innegable es que IBM está inmersa en la estrategia PowerPC. Para quienes no tengan muy claro de qué se trata, diré rápidamente que PowerPC significa Performance Optimized With Enhanced Risc-Personal Computer (está claro que este tipo de acrónimos son del agrado de todos los anglosajones), y se trata de una nueva familia de microprocesadores, no compatibles con los de Intel, que está siendo desarrollada por IBM, Apple y Motorola, y que Apple e IBM incluirán en equipos que, hasta la fecha, no son compatibles en software. Apple ya ha comercializado equipos PowerMac, además de ofrecer a todos los usuarios de los procesadores 680x0 de Motorola una fácil actualización a PowerPC. Una de sus características más importantes es que ofrecen una computación RISC (Reduced Instruction Set Computing) frente a la CISC (Complex Instruction Set Computing) de los equipos Intel hasta el 80486, siendo Pentium una solución intermedia entre ambos métodos..