JUEGO DE INSTRUCCIONES DE LA FAMILIA x86

INSTRUCCIONES DEL NIVEL DE APLICACIÓN

RESUMEN DE LAS INSTRUCCIONES DE LA FPU

Notación empleada en la descripción de las instrucciones de la FPU

- Para representar de una forma genérica los diferentes operandos que aceptan las distintas instrucciones de la FPU se utilizarán los siguientes símbolos:

ST(i):	Uno de los registros ST(0) a ST(7) de la pila de registros de la FPU. **ST(0) es la cima de la pila de registros de la FPU.**
m32real:	Operando de memoria que referencia un dato real simple (32 bits).
m64real:	Operando de memoria que referencia un dato real doble (64 bits).
m80real:	Operando de memoria que referencia un dato real extendido (80 bits).
m16int:	Operando de memoria que referencia un dato entero palabra (16 bits).
m32int:	Operando de memoria que referencia un dato entero corto (32 bits).
m64int:	Operando de memoria que referencia un dato entero largo (64 bits).
m80bcd:	Operando de memoria que referencia un dato decimal BCD empaquetado de 80 bits.
m2byte:	Operando de memoria que referencia un área de memoria de 2 bytes para cargar/almacenar la palabra de control o la palabra de estado de la FPU.
m14byte:	Operando de memoria que referencia un área de memoria 14 bytes para cargar/almacenar el entorno de la FPU.
m94byte:	Operando de memoria que referencia un área de memoria de 94 bytes para cargar/almacenar el estado de la FPU.

- Se considera como FPU de partida el 80387. Las instrucciones que sólo están disponibles a partir del procesador Pentium Pro se marcan con (Pentium Pro ->).

- Las operaciones simbólicas PUSH y POP se refieren aquí a la pila de registros de la FPU y no deben confundirse con las instrucciones de pila de la unidad de enteros. La operación simbólica POP sin operando retira el dato que está en la cima de la pila de registros de la FPU sin almacenarlo en ningún sitio.

- Internamente, todos las datos se almacenan y operan en el formato real extendido. Las instrucciones de la FPU que acceden a la memoria realizan automáticamente la conversión a este formato desde o hacia el formato de los datos en memoria.

- No se han incluido las instrucciones de punto flotante SIMD del Pentium III y Pentium 4.

Instrucciones de transferencia de datos reales

Instrucción		Descripción	Operación
FLD	m32real	Convertir un dato real simple almacenado en memoria al formato real extendido e introducirlo en la pila de la FPU.	PUSH RealExtendido([m32real])
FLD	m64real	Convertir un dato real doble almacenado en memoria al formato real extendido e introducirlo en la pila de la FPU.	PUSH RealExtendido([m64real])
FLD	m80real	Introducir en la pila de la FPU un real extendido almacenado en memoria.	PUSH [m80real]
FLD	ST(i)	Introducir en la pila de la FPU el valor en el registro ST(i).	PUSH ST(i)
FST	m32real	Almacenar en memoria el valor de ST(0) en formato real simple.	[m32real] <- RealSimple(ST(0))
FST	m64real	Almacenar en memoria el valor de ST(0) en formato real doble.	[m64real] <- RealDoble(ST(0))
FST	ST(i)	Copiar el valor de ST(0) en ST(i).	ST(i) <- ST(0)
FSTP	m32real	Almacenar en memoria el valor de ST(0) en formato real simple y después extraerlo de la pila de la FPU.	[m32real] <- RealSimple(ST(0)) ; POP
FSTP	m64real	Almacenar en memoria el valor de ST(0) en formato real doble y después extraerlo de la pila de la FPU.	[m64real] <- RealDoble(ST(0)) ; POP
FSTP	m80real	Almacenar en memoria el valor de ST(0) en formato real extendido y después extraerlo de la pila de la FPU.	[m80real] <- ST(0) ; POP
FSTP	ST(i)	Copiar el valor de ST(0) en ST(i) y después extraerlo de la pila de la FPU.	ST(i) <- ST(0) ; POP
FXCH	ST(i)	Intercambiar los valores de los registros ST(0) y ST(i).	ST(0) <-> ST(i)
FXCH		Intercambiar los valores de los registros ST(0) y ST(1).	ST(0) <-> ST(1)

Instrucciones de transferencia condicional de datos reales (Pentium Pro->)

Instrucción		Descripción	Operación
FCMOVB	ST(0), ST(i)	Copiar ST(i) en ST(0) si menor.	ST(0) <- ST(i) si menor (CF = 1)
FCMOVE	ST(0), ST(i)	Copiar ST(i) en ST(0) si igual.	ST(0) <- ST(i) si igual (ZF = 1)
FCMOVBE	ST(0), ST(i)	Copiar ST(i) en ST(0) si menor o igual.	ST(0) <- ST(i) si menor o igual (CF = 1 o ZF = 1)
FCMOVU	ST(0), ST(i)	Copiar ST(i) en ST(0) si no-ordenados.	ST(0) <- ST(i) si no-ordenados (PF = 1)
FCMOVNB	ST(0), ST(i)	Copiar ST(i) en ST(0) si no menor.	ST(0) <- ST(i) si no menor (CF = 0)
FCMOVNE	ST(0), ST(i)	Copiar ST(i) en ST(0) si no igual.	ST(0) <- ST(i) si no igual (ZF = 0)
FCMOVNBE	ST(0), ST(i)	Copiar ST(i) en ST(0) si no menor o igual.	ST(0) <- ST(i) si no menor o igual (CF = 0 y ZF = 1)
FCMOVNU	ST(0), ST(i)	Copiar ST(i) en ST(0) si no no-ordenados.	ST(0) <- ST(i) si no no-ordenados (PF = 0)

Instrucciones de transferencia de datos enteros

Instrucción	Descripción	Operación
FILD m16int	Convertir un dato entero palabra almacenado en memoria al formato real extendido e introducirlo en la pila de la FPU.	PUSH RealExtendido([m16int])
FILD m32int	Convertir un dato entero corto almacenado en memoria al formato real extendido e introducirlo en la pila de la FPU.	PUSH RealExtendido([m32int])
FILD m64int	Convertir un dato entero largo almacenado en memoria al formato real extendido e introducirlo en la pila de la FPU.	PUSH RealExtendido([m64int])
FIST m16int	Almacenar en memoria el valor de ST(0) en formato entero palabra.	[m16int] <- EnteroPalabra(ST(0))
FIST m32int	Almacenar en memoria el valor de ST(0) en formato entero corto.	[m32int] <- EnteroCorto(ST(0))
FISTP m16int	Almacenar en memoria el valor de ST(0) en formato entero palabra y después extraerlo de la pila de la FPU.	[m16int] <- EnteroPalabra(ST(0)) ; POP
FISTP m32int	Almacenar en memoria el valor de ST(0) en formato entero corto y después extraerlo de la pila de la FPU.	[m32int] <- EnteroCorto(ST(0)) ; POP
FISTP m64int	Almacenar en memoria el valor de ST(0) en formato entero largo y después extraerlo de la pila de la FPU.	[m64int] <- EnteroLargo(ST(0)) ; POP

Instrucciones de transferencia de datos en BCD empaquetado

Instrucción	Descripción	Operación
FBLD m80bcd	Convertir un BCD empaquetado almacenado en memoria al formato real extendido e introducirlo en la pila de la FPU.	PUSH RealExtendido([m80bcd])
FBSTP m80bcd	Almacenar en memoria el valor de ST(0) en formato entero BCD empaquetado y después extraerlo de la pila de la FPU.	[m80bcd] <- EnteroBCD(ST(0)) ; POP

Instrucciones de carga de constantes

Instrucción	Descripción	Operación
FLD1	Introducir 1.0 en la pila de la FPU.	PUSH 1.0
FLDL2E	Introducir el logaritmo en base 2 del número e en la pila de la FPU.	PUSH $\log_2 e$
FLDL2T	Introducir el logaritmo en base 2 de 10 en la pila de la FPU.	PUSH $\log_2 10$
FLDLG2	Introducir el logaritmo en base 10 de 2 en la pila de la FPU.	PUSH $\log_{10} 2$
FLDLN2	Introducir el logaritmo natural de 2 en la pila de la FPU.	PUSH $\log_e 2$
FLDPI	Introducir el número pi en la pila de la FPU.	PUSH pi
FLDZ	Introducir +0.0 en la pila de la FPU.	PUSH +0.0

Instrucciones de suma

Instrucción		Descripción	Operación
FADD	m32real	Sumar a ST(0) un real simple almacenado en memoria.	ST(0) <- ST(0) + RealExtendido([m32real])
FADD	m64real	Sumar a ST(0) un real doble almacenado en memoria.	ST(0) <- ST(0) + RealExtendido([m64real])
FADD	ST(i),ST(0)	Sumar a ST(i) el valor de ST(0).	ST(i) <- ST(i) + ST(0)
FADD	ST(0),ST(i)	Sumar a ST(0) el valor de ST(i).	ST(0) <- ST(0) + ST(i)
FADDP	ST(i),ST(0)	Sumar a ST(i) el valor de ST(0) y después extraer un dato de la pila (el que está en ST(0)).	ST(i) <- ST(i) + ST(0) ; POP
FADDP / FADD		Sumar a ST(1) el valor de ST(0) y después extraer un dato la pila (el que está en ST(0)). El resultado queda en ST(0).	ST(1) <- ST(1) + ST(0) ; POP
FIADD	m16int	Sumar a ST(0) un entero palabra almacenado en memoria.	ST(0) <- ST(0) + RealExtendido([m16int])
FIADD	m32int	Sumar a ST(0) un entero corto almacenado en memoria.	ST(0) <- ST(0) + RealExtendido([m32int])

Instrucciones de resta

Instrucción		Descripción	Operación
FSUB	m32real	Restar a ST(0) un real simple almacenado en memoria.	ST(0) <- ST(0) - RealExtendido([m32real])
FSUB	m64real	Restar a ST(0) un real doble almacenado en memoria.	ST(0) <- ST(0) - RealExtendido([m64real])
FSUB	ST(i),ST(0)	Restar a ST(i) el valor de ST(0).	ST(i) <- ST(i) - ST(0)
FSUB	ST(0),ST(i)	Restar a ST(0) el valor de ST(i).	ST(0) <- ST(0) - ST(i)
FSUBP	ST(i),ST(0)	Restar a ST(i) el valor de ST(0) y después extraer un dato de la pila (el que está en ST(0)).	ST(i) <- ST(i) - ST(0) ; POP
FSUBP / FSUB		Restar a ST(1) el valor de ST(0) y después extraer un dato de la pila (el que está en ST(0)). El resultado queda en ST(0).	ST(1) <- ST(1) - ST(0) ; POP
FISUB	m16int	Restar a ST(0) un entero palabra almacenado en memoria.	ST(0) <- ST(0) - RealExtendido([m16int])
FISUB	m32int	Restar a ST(0) un entero corto almacenado en memoria.	ST(0) <- ST(0) - RealExtendido([m32int])

Instrucciones de resta al revés

Instrucción		Descripción	Operación
FSUBR	m32real	Restar a un real simple almacenado en memoria el valor de ST(0) y dejar el resultado en ST(0).	ST(0) <- RealExtendido([m32real]) - ST(0)
FSUBR	m64real	Restar a un real doble almacenado en memoria el valor de ST(0) y dejar el resultado en ST(0).	ST(0) <- RealExtendido([m64real]) - ST(0)
FSUBR	ST(i),ST(0)	Restar al valor de ST(0) el valor de ST(i) y dejar el resultado en ST(i).	ST(i) <- ST(0) - ST(i)
FSUBR	ST(0),ST(i)	Restar al valor de ST(i) el valor de ST(0) y dejar el resultado en ST(0).	ST(0) <- ST(i) - ST(0)
FSUBRP	ST(i),ST(0)	Restar al valor de ST(0) el de ST(i) y dejar resultado en ST(i). Después extraer un dato de la pila (el que está en ST(0)).	ST(i) <- ST(0) - ST(i) ; POP
FSUBRP / FSUBR		Restar al valor de ST(0) el de ST(1) y dejar res. en ST(1). Después extraer un dato de la pila. El result. queda en ST(0).	ST(1) <- ST(0) - ST(1) ; POP
FISUBR	m16int	Restar al valor de un entero palabra en memoria el valor de ST(0) y dejar el resultado en ST(0).	ST(0) <- RealExtendido([m16int]) - ST(0)
FISUBR	m32int	Restar al valor de un entero corto en memoria el valor de ST(0) y dejar el resultado en ST(0).	ST(0) <- RealExtendido([m32int]) - ST(0)

Instrucciones de multiplicación

Instrucción		Descripción	Operación
FMUL	m32real	Multiplicar ST(0) por un real simple almacenado en memoria.	ST(0) <- ST(0)*RealExtendido([m32real])
FMUL	m64real	Multiplicar ST(0) por un real doble almacenado en memoria.	ST(0) <- ST(0)*RealExtendido([m64real])
FMUL	ST(i),ST(0)	Multiplicar ST(i) por el valor de ST(0).	ST(i) <- ST(i)*ST(0)
FMUL	ST(0),ST(i)	Multiplicar ST(0) por el valor de ST(i).	ST(0) <- ST(0)*ST(i)
FMULP	ST(i),ST(0)	Multiplicar ST(i) por el valor de ST(0) y después extraer un dato de la pila (el que está en ST(0)).	ST(i) <- ST(i)*ST(0) ; POP
FMULP / FMUL		Multiplicar ST(1) por el valor de ST(0) y extraer un dato de la pila (el que está en ST(0)). El resultado queda en ST(0).	ST(1) <- ST(1)*ST(0) ; POP
FIMUL	m16int	Multiplicar ST(0) por un entero palabra almacenado en memoria.	ST(0) <- ST(0)*RealExtendido([m16int])
FIMUL	m32int	Multiplicar ST(0) por un entero corto almacenado en memoria.	ST(0) <- ST(0)*RealExtendido([m32int])

Instrucciones de división

Instrucción		Descripción	Operación
FDIV	m32real	Dividir ST(0) por un real simple almacenado en memoria.	ST(0) <- ST(0)/RealExtendido([m32real])
FDIV	m64real	Dividir ST(0) por un real doble almacenado en memoria.	ST(0) <- ST(0)/RealExtendido([m64real])
FDIV	ST(i),ST(0)	Dividir ST(i) por el valor de ST(0).	ST(i) <- ST(i)/ST(0)
FDIV	ST(0),ST(i)	Dividir ST(0) por el valor de ST(i).	ST(0) <- ST(0)/ST(i)
FDIVP	ST(i),ST(0)	Dividir ST(i) por el valor de ST(0) y después extraer un dato de la pila (el que está en ST(0)).	ST(i) <- ST(i)/ST(0) ; POP
FDIVP / FDIV		Dividir ST(1) por el valor de ST(0) y después extraer un dato de la pila (el que está en ST(0)). El resultado queda en ST(0).	ST(1) <- ST(1)/ST(0) ; POP
FIDIV	m16int	Dividir ST(0) por un entero palabra almacenado en memoria.	ST(0) <- ST(0)/RealExtendido([m16int])
FIDIV	m32int	Dividir ST(0) por un entero corto almacenado en memoria.	ST(0) <- ST(0)/RealExtendido([m32int])

Instrucciones de división al revés

Instrucción		Descripción	Operación
FDIVR	m32real	Dividir un real simple almacenado en memoria por el valor de ST(0) y dejar el resultado en ST(0).	ST(0) <- RealExtendido([m32real])/ST(0)
FDIVR	m64real	Dividir un real doble almacenado en memoria por el valor de ST(0) y dejar el resultado en ST(0).	ST(0) <- RealExtendido([m64real])/ST(0)
FDIVR	ST(i),ST(0)	Dividir el valor de ST(0) por el valor de ST(i) y dejar el resultado en ST(i).	ST(i) <- ST(0)/ST(i)
FDIVR	ST(0),ST(i)	Dividir el valor de ST(i) por el valor de ST(0) y dejar el resultado en ST(0).	ST(0) <- ST(i)/ST(0)
FDIVRP	ST(i),ST(0)	Dividir el valor de ST(0) por el de ST(i) y dejar resultado en ST(i). Después extraer un dato de la pila (el que está en ST(0)).	ST(i) <- ST(0)/ST(i) ; POP
FDIVRP / FDIVR		Dividir el valor de ST(0) por ST(1) y dejar resultado en ST(1). Después extraer un dato de la pila. El result. queda en ST(0).	ST(1) <- ST(0)/ST(1) ; POP
FIDIVR	m16int	Dividir el valor de un entero palabra en memoria por el valor de ST(0) y dejar el resultado en ST(0).	ST(0) <- RealExtendido([m16int])/ST(0)
FIDIVR	m32int	Dividir el valor de un entero corto en memoria por el valor de ST(0) y dejar el resultado en ST(0).	ST(0) <- RealExtendido([m32int])/ST(0)

Instrucciones de comparación

Instrucción		Descripción	Operación
FCOM	m32real	Comparar ST(0) con un real simple almacenado en memoria. Modificar C3, C2 y C0 según el resultado.	C3, C2, C0 <- Comparar(ST(0), RealExt([m32real]))
FCOM	m64real	Comparar ST(0) con un real doble almacenado en memoria. Modificar C3, C2 y C0 según el resultado.	C3, C2, C0 <- Comparar(ST(0), RealExt([m64real]))
FCOM	ST(i)	Comparar ST(0) con ST(i). Modificar C3, C2 y C0 según el resultado.	C3, C2, C0 <- Comparar(ST(0), ST(i))
FCOM		Comparar ST(0) con ST(1). Modificar C3, C2 y C0 según el resultado.	C3, C2, C0 <- Comparar(ST(0), ST(1))
FCOMP	m32real	Comparar ST(0) con un real simple en memoria modificando C3, C2 y C0.Después extraer un valor de la pila (el de ST(0)).	FCOM m32real ; POP
FCOMP	m64real	Comparar ST(0) con un real doble en memoria modificando C3, C2 y C0.Después extraer un valor de la pila (el de ST(0)).	FCOM m64real ; POP
FCOMP	ST(i)	Comparar ST(0) con ST(i) modificando C3, C2 y C0. Después extraer un dato de la pila (el que está en ST(0)).	FCOM ST(i) ; POP
FCOMP		Comparar ST(0) con ST(1) modificando C3, C2 y C0. Después extraer un dato de la pila (el que está en ST(0)).	FCOM ; POP
FCOMPP		Comparar ST(0) con ST(1) modificando C3, C2 y C0. Después extraer dos valores de la pila (los de ST(0) y ST(1)).	FCOM ; POP ; POP
FICOM	m16int	Comparar ST(0) con un entero palabra almacenado en memoria. Modificar C3, C2 y C0 según el resultado.	C3,C2,C0 <- Comparar(ST(0), RealExt([m16int]))
FICOM	m32int	Comparar ST(0) con un entero corto almacenado en memoria. Modificar C3, C2 y C0 según el resultado.	C3,C2,C0 <- Comparar(ST(0), RealExt([m32int]))
FICOMP	m16int	Comparar ST(0) con un entero palabra en mem. modificando C3,C2 y C0.Después extraer un valor de la pila (el de ST(0)).	FICOM m16int ; POP
FICOMP	m32int	Comparar ST(0) con un entero corto en mem. modificando. C3, C2 y C0. Después extraer un valor de la pila (el de ST(0)).	FICOM m32int ; POP
FTST		Comparar ST(0) con 0.0. Modificar C3, C2 y C0 según el resultado.	C3,C2,C0 <- Comparar(ST(0), 0.0)
FUCOM	m32real	Comparación no ordenada[1] de ST(0) con un real simple en memoria. Modificar C3, C2 y C0 según el resultado.	C3,C2,C0<- CompNoOrd(ST(0), RealExt([m32real]))
FUCOM	m64real	Comparación no ordenada[1] de ST(0) con un real doble en memoria. Modificar C3, C2 y C0 según el resultado.	C3,C2,C0<- CompNoOrd(ST(0), RealExt([m64real]))
FUCOM	ST(i)	Comparación no ordenada[1] de ST(0) con ST(i). Modificar C3, C2 y C0 según el resultado.	C3, C2, C0 <- CompNoOrd(ST(0), ST(i))
FUCOM		Comparación no ordenada[1] de ST(0) con ST(1) modificando C3,C2 y C0. Después extraer un dato de la pila (el de ST(0)).	C3, C2, C0 <- CompNoOrd(ST(0), ST(1))
FUCOMP	m32real	Comparación no ordenada[1] de ST(0) con real simple en mem. modificando C3, C2 y C0. Después extraer dato (ST(0)).	FUCOM m32real ; POP
FUCOMP	m64real	Comparación no ordenada[1] de ST(0) con real doble en mem. modificando C3, C2 y C0. Después extraer dato (ST(0)).	FUCOM m64real ; POP
FUCOMP	ST(i)	Comparación no ordenada[1] de ST(0) con ST(i) modificando C3, C2 y C0. Después extraer un valor de la pila (el de ST(0)).	FUCOM ST(i) ; POP
FUCOMP		Comparación no ordenada[1] de ST(0) con ST(1) modificando C3, C2 y C0. Después extraer un dato de la pila (ST(0)).	FUCOM ; POP
FUCOMPP		Comparación no ordenada[1] de ST(0) y ST(1) modificando C3,C2 y C0. Después extraer dos datos de pila (ST(0) y ST(1)).	FUCOM ; POP ; POP

Ver tabla de códigos de condición en la página siguiente.

[1] Una comparación no ordenada se diferencia de una comparación normal en que la primera sólo genera una excepción de operando aritmético no válido si uno o los dos operandos comparados son SNaN o están en un formato no soportado. Si uno o los dos operandos comparados son QNaN los bits de código de condición se ajustan señalando no ordenados pero no se genera ninguna excepción. Por el contrario, una instrucción de comparación del tipo FCOM genera una excepción de operación no válida cuando uno o los dos operandos comparados son NaN (QNaN o SNaN) o están en un formato no soportado.

Modificación de códigos de condición por FCOM / FCOMP / FCOMPP / FTST / FUCOM / FUCOMP / FUCOMPP

Relación de orden	C3	C2	C0
ST(0) > Operando fuente	0	0	0
ST(0) < Operando fuente	0	0	1
ST(0) = Operando fuente	1	0	0
No ordenados[2]	1	1	1

Instrucciones de comparación con resultado en FLAGS (Pentium Pro ->)

Instrucción	Descripción	Operación
FCOMI ST(0), ST(i)	Comparar ST(0) con ST(i) modificando ZF, PF y CF.	Comparar(ST(0), ST(1)) -> ZF, PF, CF
FCOMIP ST(0), ST(i)	Comparar ST(0) con ST(i) modificando ZF, PF y CF. Después extraer un dato de la pila (el que está en ST(0)).	FCOMI ST(0), ST(i) ; POP
FUCOMI ST(0), ST(i)	Comparación no ordenada[1] de ST(0) y ST(i) modificando ZF, PF y CF.	CompNoOrd(ST(0), [m32real]) -> ZF, PF, CF
FUCOMIP ST(0), ST(i)	Comparación no ordenada[1] de ST(0) y ST(i) modificando ZF, PF y CF. Después extraer un dato (el que está en ST(0)).	FUCOMI ST(0), ST(i) ; POP

Modificación de FLAGS por FCOMI / FCOMIP / FUCOMI / FUCOMIP

Relación de orden	ZF	PF	CF
ST(0) > ST(i)	0	0	0
ST(0) < ST(i)	0	0	1
ST(0) = ST(i)	1	0	0
No ordenados[2]	1	1	1

[1] Una comparación no ordenada se diferencia de una comparación normal en que la primera sólo genera una excepción de operando aritmético no válido si uno o los dos operandos comparados son SNaN o están en un formato no soportado. Si uno o los dos operandos comparados son QNaN los bits de código de condición se ajustan señalando no ordenados pero no se genera ninguna excepción. Por el contrario, una instrucción de comparación del tipo FCOM genera una excepción de operación no válida cuando uno o los dos operandos comparados son NaN (QNaN o SNaN) o están en un formato no soportado.

[2] La condición de no ordenados se produce cuando al menos uno de los operandos comparados en un NaN o un formato no soportado.

Instrucción FXAM

Instrucción	Descripción	Operación
FXAM	Examinar la clase de dato que está almacenado en ST(0) y modificar C3, C2 y C0 según el resultado.	C3, C2, C0 <- Examinar(ST(0))

Modificación de códigos de condición por FXAM

Clase de dato en ST(0)	C3	C2	C0
No soportado	0	0	0
NaN (No un número)	0	0	1
Número finito normalizado	0	1	0
Infinito	0	1	1
Cero	1	0	0
Vacío	1	0	1
Número denormalizado	1	1	0

Otras instrucciones aritméticas

Instrucción	Descripción	Operación
FABS	Sustituir el valor de ST(0) por su valor absoluto.	ST(0) <- Abs(ST(0))
FCHS	Cambiar el signo del valor en ST(0).	ST(0) <- - ST(0)
FRNDINT	Redondear el valor de ST(0) a un entero. El redondeo se realiza de acuerdo al estado del campo RC del registro SW.	ST(0) <- Redondear_a_entero(ST(0))
FSQRT	Sustituir el valor de ST(0) por su raíz cuadrada.	ST(0) <- Sqrt(ST(0))
FPREM	Calcula el resto de la división ST(0)/ST(1) y almacena el resultado en ST(0).El resto calculado por FPREM representa el valor: Resto = ST(0) - (Q*ST(1)) donde Q es el entero que se obtiene **truncando el cociente ST(0)/ST(1), es decir tomando su parte entera**. El signo del resto es el mismo que el del dividendo. El valor absoluto del resto es menor que el del divisor, excepto en el caso de que se obtenga un resto parcial. La instrucción calcula el resto a través de un proceso de resta iterativo. FPREM no puede reducir el exponente de ST(0) en más de 63 unidades. Si la instrucción tiene éxito en generar un resto menor que el divisor la operación está completa y el flag C2 de la FPU se pone a 0. En caso contrario, C2 se pone a 1 y el resultado que queda en ST(0) es un resto parcial. El exponente del resto parcial habrá disminuido al menos en 32 unidades respecto al del dividendo original. La instrucción puede volver a ejecutarse (utilizando el resto parcial como nuevo dividendo) hasta que C2 se ponga a 0.	ST(0) <- RestoParcial(ST(0)/ST(1)) C2 <- EsRestoParcial
FPREM1	Calcula el resto de la división ST(0)/ST(1) y almacena el resultado en ST(0). El cálculo se realiza conforme al estándar IEEE 754. El resto calculado por FPREM1 representa el valor: Resto = ST(0) - (Q*ST(1)) donde Q es el entero que se obtiene **redondeando el cociente ST(0)/ST(1) al entero más próximo**. El valor absoluto del resto es menor que el valor absoluto del divisor partido de 2, excepto en el caso de que se obtenga un resto parcial. La instrucción calcula el resto a través de un proceso de resta iterativo. La instrucción FPREM1 no puede reducir el exponente de ST(0) en más de 63 unidades. Si la instrucción tiene éxito en generar un resto cuyo valor absoluto es menor que el valor absoluto del divisor partido de 2, la operación está completa y el flag C2 de la FPU se pone a 0. En caso contrario, C2 se pone a 1 y el resultado que queda en ST(0) es un resto parcial. El exponente del resto parcial habrá disminuido al menos en 32 unidades respecto al del dividendo original. La instrucción puede volver a ejecutarse (utilizando el resto parcial como nuevo dividendo) hasta que C2 se ponga a 0.	ST(0) <- RestoParcial_IEEE754(ST(0)/ST(1)) C2 <- EsRestoParcial
FXTRACT	Obtiene el exponente y el significando de ST(0). Almacena el exponente en ST(0) e introduce el significando en la pila. Por tanto el exponente queda en ST(1) y el significando en ST(0).	temp <- ST(0) ; ST(0) <- Exponente(ST(0)) ; PUSH Significando(temp)

Instrucciones para funciones trigonométricas

Instrucción	Descripción	Operación
FSIN	Calcula el seno del ángulo en radianes en ST(0) y deja el resultado en ST(0). El ángulo debe estar entre -2^{63} y $+2^{63}$ rad..	ST(0) <- sin(ST(0))
FCOS	Calcula el coseno del ángulo en radianes en ST(0) y deja el resultado en ST(0). El ángulo debe estar entre -2^{63} y $+2^{63}$ rad..	ST(0) <- cos(ST(0))
FSINCOS	Calcula el seno y el coseno del ángulo en radianes en ST(0). Sustituye ST(0) por el seno e introduce el coseno en la pila. Por tanto, el seno queda en ST(1) y el coseno en ST(0). El ángulo debe estar entre -2^{63} y $+2^{63}$ radianes.	temp <- ST(0) ; ST(0) <- sin(temp) ; PUSH cos(temp)
FPTAN	Calcula la tangente del ángulo en radianes almacenado ST(0) y deja el resultado en ST(0). Después introduce 1.0 en la pila. Por tanto, la tangente queda en ST(1) mientras que en ST(0) queda 1.0. Se introduce 1.0 en la pila para mantener la compatibilidad con el 8087 y el 80287 y para facilitar el cálculo de otras funciones trigonométricas. Por ejemplo, puede obtenerse la cotangente ejecutando FDIVRP después de FPTAN. El ángulo debe estar entre -2^{63} y $+2^{63}$ radianes.	ST(0) <- tan(ST(0)) ; PUSH 1.0
FPATAN	Calcula el arco tangente en radianes del cociente ST(1)/ST(0) y deja el resultado en ST(1). Después retira el valor en la cima de la pila. Por tanto, el resultado queda en ST(0). FPATAN devuelve el ángulo en radianes formado entre el semieje positivo X y la línea trazada desde el origen hasta el punto (ST(0), ST(1)). El ángulo depende por tanto de los signos iniciales de ST(0) y ST(1) y no del signo del cociente ST(1)/ST(0). El resultado tiene siempre el mismo signo que el valor inicial de ST(0) y está comprendido entre -pi y pi.	ST(1) <- atan(ST(1)/ST(0)) ; POP

Instrucciones para funciones logarítmicas, exponenciales y de escalado

Instrucción	Descripción	Operación
FYL2X	Calcula ST(1)*$\log_2$(ST(0)) y deja el resultado en ST(1). Después extrae el valor en la cima de la pila. Por tanto, resultado queda finalmente en ST(0). El valor inicial de ST(0) debe ser mayor que cero. La razón de la multiplicación por ST(1) es optimizar el cálculo de logaritmos en una base arbitraria: $\log_b x = (\log_b 2)*\log_2 x = (\log_2 b)^{-1}*\log_2 x$.	ST(0) <- ST(1)*$\log_2$(ST(0))
FYL2XP1	Calcula ST(1)*$\log_2$(ST(0)+1) y deja el resultado en ST(1). Después extrae el valor en la cima de la pila. Por tanto, el resultado queda finalmente en ST(0). El valor de ST(0) debe estar entre -(1-sqrt(2)/2) y (1-sqrt(2)/2). Esta instrucción proporciona una exactitud óptima cuando el valor de ST(0) está próximo a 0. El resultado puede convertirse un base de logaritmos arbitraria b utilizando en ST(1) el factor de escala $\log_b 2$.	ST(0) <- ST(1)*$\log_2$(ST(0) + 1)
F2XM1	Calcula $2^{ST(0)} - 1$ y deja el resultado en ST(0). El valor inicial de ST(0) debe estar entre -1.0 y +1.0. Para calcular una potencia de otra base puede usarse $x^y = 2^{(y*\log_2 x)}$.	ST(0) <- $2^{ST(0)} - 1$
FSCALE	Multiplica ST(0) por 2 elevado a la parte entera del valor en ST(1) y almacena el resultado en ST(0). Esta instrucción proporciona una forma rápida de multiplicar o dividir por potencias enteras de 2. También puede usarse para revertir la acción de FXTRACT mediante FSCALE seguida de FSTP ST(1).	ST(0) <- ST(0)*$2^{Int(ST(1))}$

Instrucciones de control de la FPU

Instrucción		Descripción	Operación
FINIT		Procesa las excepciones de punto flotante no enmascaradas pendientes y a continuación inicializa la FPU cargando los registros de estado, control, tag, puntero de instrucción y puntero de datos de la FPU con sus valores por defecto.	ProcesarExcepcionesFPU;CW<-037Fh;SW<-0; TW<-FFFFh;FPU_DP<-0;FPU_IP<-0;FPU_IO<-0
FNINIT		Inicializa la FPU cargando las palabras de estado, control, etiquetas, puntero de instrucción y puntero de datos de la FPU con sus valores por defecto.	CW <- 037Fh ; SW <- 0 ;TW <- FFFFh; FPU_DP <- 0 ; FPU_IP <- 0 ; FPU_IO <- 0
FLDCW	m2byte	Carga la palabra de control de la FPU con un dato de 16 bits ubicado en memoria.	CW <- [m2byte]
FSTCW	m2byte	Procesa las excepciones de punto flotante no enmascaradas pendientes y a continuación almacena en memoria el contenido de la palabra de control de la FPU.	ProcesarExcepcionesFPU; [m2byte] <- CW
FNSTCW	m2byte	Almacena en memoria el contenido de la palabra de control de la FPU.	[m2byte] <- CW
FSTSW	m2byte	Procesa las excepciones de punto flotante no enmascaradas pendientes y a continuación almacena en memoria el contenido de la palabra de estado de la FPU.	ProcesarExcepcionesFPU; [m2byte] <- SW
FNSTSW	m2byte	Almacena en memoria el contenido de la palabra de estado de la FPU.	[m2byte] <- SW
FSTSW	AX	Procesa las excepciones de punto flotante no enmascaradas pendientes y a continuación almacena en AX el contenido de la palabra de estado de la FPU.	ProcesarExcepcionesFPU; AX <- SW
FNSTSW	AX	Almacena en AX el contenido de la palabra de estado de la FPU.	AX <- SW
FCLEX		Procesa las excepciones de punto flotante no enmascaradas pendientes y a continuación pone a cero los flags de excepción de la FPU, el flag de resumen de excepción, el flag de fallo de pila y el flag de ocupado.	ProcesarExcepcionesFPU; PE<-0; UE<-0; OE<-0; ZE<-0; DE<-0; IE<-0; ES<-0; SF<-0; B<-0
FNCLEX		Pone a cero los flags de excepción de la FPU, el flag de resumen de excepción, el flag de fallo de pila y el flag de ocupado.	PE<-0; UE<-0; OE<-0; ZE<-0; DE<-0; IE<-0; ES<-0; SF<-0; B<-0
FLDENV	m14byte	Carga el entorno de la FPU[1] desde la posición de memoria especificada.	(Entorno de la FPU)[1] <- [m14byte]
FSTENV	m14byte	Procesa las excepciones de punto flotante no enmascaradas pendientes y a continuación almacena en memoria el entorno de la FPU[1].	ProcesarExcepcionesFPU; [m14byte] <- (Entorno de la FPU)[1]
FNSTENV	m14byte	Almacena en memoria el entorno de la FPU[1].	[m14byte] <- (Entorno de la FPU)[1]
FRSTOR		Carga el estado de la FPU[2] desde la posición de memoria especificada.	(Estado de la FPU)[2] <- [m94byte]
FSAVE		Procesa las excepciones de punto flotante no enmascaradas pendientes. A continuación, almacena en memoria el estado de la FPU[2] e inciliaciza la FPU (ver FNINIT).	ProcesarExcepcionesFPU; [m94bytes] <- (Estado de la FPU)[2]; InicializarFPU
FNSAVE		Almacena en memoria el estado de la FPU[2] e inicializa la FPU (ver FNINIT).	[m94bytes] <- (Estado de la FPU)[2]; InicializarFPU
FINCSTP		Incrementa el campo TOP (puntero a la cima de la pila) de la palabra de estado. Si TOP contiene 7, pasa a 0.	Si(TOP = 7) TOP <-0; EnOtroCaso TOP <- TOP + 1
FDECSTP		Decrementa el campo TOP (puntero a la cima de la pila de la FPU) de la palabra de estado. Si TOP vale 0, pasa a 7.	Si(TOP = 0) TOP <-7; EnOtroCaso TOP <- TOP - 1
FFREE	ST(i)	Marca el registro ST(i) como vacía cargando 11b en el campo asociado a ST(i) en la palabra de etiquetas.	TAG(i) <- 11b
FWAIT / WAIT		Procesa las excepciones de punto flotante no enmascaradas pendientes.	ProcesarExcepcionesFPU
FNOP		No se realiza ninguna operación de FPU.	Ninguna

[1] El entorno de la FPU está formado por el contenido de las palabras de control, estado y etiquetas, los punteros de instrucción y datos de la FPU y el último código de operación.

[2] El estado de la FPU está formado por el entorno de la FPU[1] y el contenido de los registros de datos ST(0) a ST(7) de la FPU.