Uso del Coprocesador Matemático

El coprocesador o unidad de control de punto flotante es un circuito integrado especializado o parte del procesador principal, que se caracteriza por tener un conjunto de instrucciones orientadas a la resolución de cálculos matemáticos complejos y con alta precisión (funciones básicas, trigonométricas, logarítmicas, etc.)

El coprocesador puede manejar valores en distintas precisiones, ***operando internamente con 80 bits***:

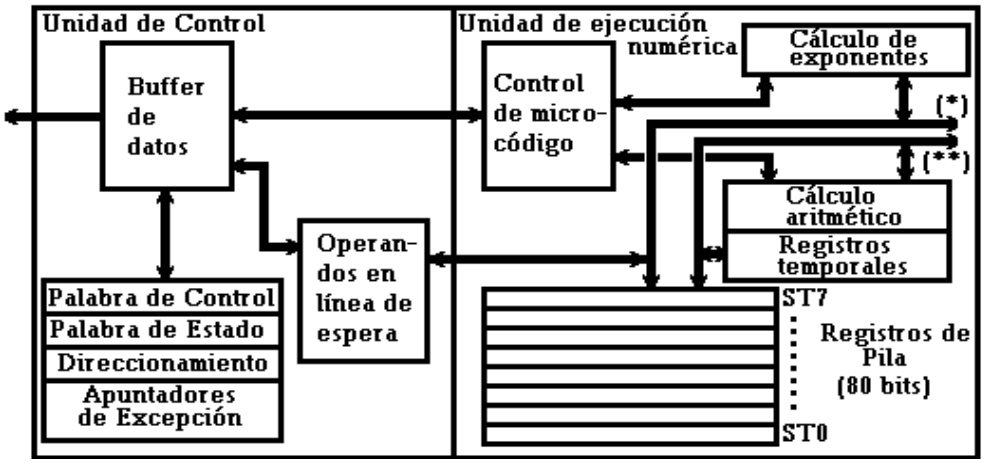
Words, 16 bits = 2 bytes, (DD) definen enteros con signo (-32768 a +32767).

Dwords, 32 bits = 4 bytes, (DD) definen enteros con signo (-2.147.483.648 a +2.147.483.647), y valores de punto flotante con precisión simple.

Qwords, 64 bits = 8 bytes, (DQ) definen valores en el formato de punto flotante de doble precisión.

80 bits = 10 bytes, (DT) definen cantidades en el formato de punto flotante con precisión plena ("full precision")

(Existen algunas instrucciones del coprocesador que reconocen el formato BCD en 10 bytes)



El coprocesador matemático posee básicamente una unidad de control y una unidad de ejecución que posee 8 registros de punto flotante que pueden almacenar 80 bits y están organizados como una pila LIFO.
Cada registro se nombra como ST0, ST1, ST2,…….,ST7 .
ST0 se refiere siempre al valor en el tope de la pila y todos aquellos valores nuevos se añaden al tope.

Al cargar el primer registro (ST0) con un dato se produce automáticamente un desplazamiento de los datos contenidos en los otros registros. Así, si se escribe en ST0 un dato que se encuentra en memoria (instrucción FLD → load), entonces el dato que está en ST0 se transfiere a ST1 dejando disponible ST0, el dato que está en ST1 se transfiere a ST2, el de ST2 a ST3, etc., y se pierde el dato en ST7.

El coprocesador y el procesador se comunican directamente haciendo uso de las instrucciones propias del primero. Para distinguir dichas instrucciones, en general se les coloca una F delante de cada una de ellas.

Las instrucciones para la carga desde memoria al tope de la pila y viceversa, son:

FLD n	Carga un nº de punto flotante (n) de la memoria a la pila del coprocesador (Puede ser un DD, DQ o un DT)
FILD n	Ídem FLD, pero con un entero (n) convirtiéndolo a flotante
FST *dest*	copia de la pila ST(0) a la memoria (*dest*). El *dest*, puede ser precisión simple o doble
FIST *dest*	Ídem FST, para enteros.
FSTP *dest*	Ídem FST, pero además lo quita de la pila
FISTP *dest*	Ídem FIST, pero lo quita de la pila, para enteros

Para las comparaciones de números en punto flotante, el coprocesador utiliza en la palabra de estado, entre otras, cuatro banderas: c0, c1, c2 y c3 (bits 8 9 10 y 14) que mantienen información que corresponde a la indicación del código de condición para las instrucciones de comparación.

Por ejemplo, para las instrucciones de comparación, la condición es anunciada a través de los bits C3 y C0. (si son 00, entonces ST0 es mayor que el operando; si son 01, ST0 es menor que el operando; cuando resulta 10, entonces ST0 es igual al operando y si son 11, los operandos son incomparables.)

Estos bits no pueden ser accedidos directamente desde la CPU, de manera que las instrucciones de salto consultan el registro *FLAGS*, pero no los registros de estado del coprocesador. Para resolverlo se deben utilizar nuevas instrucciones para transferir los bits de la palabra de estado del coprocesador a los correspondientes bits del registro *FLAGS*:

FSTSW dest	Almacena la palabra de estado del coprocesador en memoria o en dest. (generalmente registro AX)
SAHF	Almacena el registro AH en el registro *FLAGS*
LAHF	Carga el registro AH con los bits del registro *FLAGS*

Ejemplo

Si deseo efectuar **if (x >y)**

```
        fld     y     ;  ST0 = x
        fcomp   x     ; compara ST0 con y
        fstsw   ax          ; mueve los bits C a FLAGS
        sahf
        jna else_part       ; si x no es mayor que y, vaya a else_part

then_part:
        ; .... código para el “then”
        jmp     end_if
else_part:
        ;código para parte “else”
end_if:
```

Ejemplo de suma de entero y real

```
.MODEL LARGE
.386
.STACK 200h

.DATA

Numero1     dd      25              ; número en formato entero
Numero2     dd      1.25            ; número en formato real

Resul       dd      ?               ; Variable para el resultado
Desca       dd      ?               ; Variable para corrección de pila

.CODE


            fild    Numero1         ; carga en el copro la variable 1 indicando que es entero
            fld     Numero2         ; carga el número real

            fadd                    ;  Se hace la suma St(0)= St(0) + St(1)

            fstp    Resul           ; Descarga el resultado de la suma.
                                    ;St(0) fue quitado y ahora St(1) pasa a ser St(0)

            fstp    Desca           ; Descarga el otro valor, dejando la pila vacía *(1)

            mov     ax,4c00h
            int     21h
End     Start
```

(En las divisiones siempre se divide por ST(0), es decir , el divisor es ST(0))

La pila hay que dejarla siempre como estaba al principio de la ejecución de la rutina. Por dicho motivo se realiza la instrucción *(1).

También se puede usar, en vez de fadd, faddp que realiza automáticamente la suma entre St(0) y St(1) y elimina el operando sobrante de manera automática.
Cabe aclarar que las instrucciones con “p” al final SÓLO trabajan con las posiciones St(0) y St(1) de la pila.

Ejemplo, ahora usando faddp:

```
CODE:
            fild    Numero1         ; Carga en el copro la variable 1 indicando que es entero
            fld     Numero2         ; Carga el número real

            faddp                   ; Se hace la suma St(0)= St(0) + St(1) y se quita St(1)

            fstp    Resul;          ;Descarga el resultado de la suma
                                    ;St(0) fue quitado y ahora la pila queda vacía
            mov     ax,4c00h;
            int     21h
```

También se puede usar una función, ffree para liberar la pila

Start:

```
fild      Numero1          ; Carga en el copro la variable 1 indicando que es entero
fld       Numero2          ; Carga el número real

fadd                       ;Se hace la suma St(0)= St(0) + St(1)

fstp      resul            ;Descarga el resultado de la suma
                           ;St(0) fue quitado y ahora St(1) pasa a ser St(0)

ffree                      ;Se libera la pila
mov       ax,4c00h;
int       21h

mov       eax,4c00h
int       21h
```

Pero sólo sirve para liberar la pila cuando sólo tiene un elemento cargado.

Apéndice

Set completo de instrucciones del coprocesador matemático (*2):

Instrucción	Descripción
F2XM1	0 := (2.0 ** 0) - 1.0
FABS	0 := |0|
FADD	1 := 1 + 0, pop
FBLD mem10d	push, 0 := mem10d (dato en BCD)
FBSTP mem10d	mem10d := 0, pop (dato en BCD)
FCHS	0 := -0
FCOM	comparar, 0 - 1
FCOM 0,i	comparar, 0 - i
FCOM i	comparar, 0 - i
FCOM mem4r	comparar, 0 - mem4r
FCOM mem8r	comparar, 0 - mem8r
FCOMP	comparar, 0 - 1, pop
FCOMP 0,i	comparar, 0 - i, pop
FCOMP i	comparar, 0 - i, pop
FCOMP mem4r	comparar, 0 - mem4r, pop
FCOMP mem8r	comparar, 0 - mem8r, pop
FCOMPP	comparar, 0 - 1, ambos pop
FCOS	sólo 387: push, 1/0 := coseno(ant. 0)
FDECSTP	decrementar el stack pointer
FDISI	deshabilitar interrupciones(ignora .287)
FDIV	1 := 1 / 0, pop
FDIV i	0 := 0 / i
FDIV i,0	i := i / 0
FDIV 0,i	0 := 0 / i
FDIV mem4r	0 := 0 / mem4r
FDIV mem8r	0 := 0 / mem8r
FDIVP i,0	i := i / 0, pop
FDIVR	1 := 0 / 1, pop
FDIVR i	0 := i / 0
FDIVR i,0	i := 0 / i
FDIVR 0,i	0 := i / 0
FDIVR mem4r	0 := mem4r / 0
FDIVR mem8r	0 := mem8r / 0
FDIVRP i,0	i := 0 / i, pop
FENI	habilitar interrupciones (ignora .287)
FFREE i	i vacío
FIADD mem2i	0 := 0 + mem4i
FIADD mem4i	0 := 0 + mem2i
FICOM mem2i	comparar, 0 - mem2i
FICOM mem4i	comparar, 0 - mem4i
FICOMP mem2i	comparar, 0 - mem2i, pop
FICOMP mem4i	comparar, 0 - mem4i, pop
FIDIV mem2i	0 := 0 / mem2i
FIDIV mem4i	0 := 0 / mem4i
FIDIVR mem2i	0 := mem2i / 0
FIDIVR mem4i	0 := mem4i / 0
FILD mem2i	push, 0 := mem2i
FILD mem4i	push, 0 := mem4i
FILD mem8i	push, 0 := mem8i
FIMUL mem2i	0 := 0 * mem2i
FIMUL mem4i	0 := 0 * mem4i

FINCSTP	incrementar stack pointer
FINIT	inicializar el 80x87
FIST mem2i	mem2i := 0
FIST mem4i	mem4i := 0
FISTP mem2i	mem2i := 0, pop
FISTP mem4i	mem4i := 0, pop
FISTP mem8i	mem8i := 0, pop
FISUB mem2i	0 := 0 - mem2i
FISUB mem4i	0 := 0 - mem4i
FISUBR mem2	0 := mem2i - 0
FISUBR mem4	0 := mem4i - 0
FLD i	push, 0 := old i
FLD mem10r	push, 0 := mem10r
FLD mem4r	push, 0 := mem4r
FLD mem8r	push, 0 := mem8r
FLD1	push, 0 := 1.0
FLDCW mem2i	Palabra de Control:= mem2i
FLDL2E	push, 0 := log base 2.0 de e
FLDL2T	push, 0 := log base 2.0 de 10.0
FLDLG2	push, 0 := log base 10.0 de 2.0
FLDLN2	ush, 0 := log base e de 2.0
FLDPI	push, 0 := Pi
FLDZ	push, 0 := +0.0
FMUL	1 := 1 * 0, pop
FMUL i	0 := 0 * i
FMUL i,0	i := i * 0
FMUL 0,i	0 := 0 * i
FMUL mem4r	0 := 0 * mem4r
FMUL mem8r	0 := 0 * mem8r
FMULP i,0	i := i * 0, pop
FNCLEX	borrar excepciones sin Wait
FNSTCW mem2i	mem2i := palabra de control
FNSTSW AX	AX := palabra de estado
FNSTSW mem2i	mem2i := palabra de estado
FPATAN	0 := arctan(1/0), pop
FPREM	0 := REPITE(0 - 1)
FPREM1	387 sólo: 0 := REPITE(0 - 1) IEEE compat.
FPTAN	push, 1/0 := tan(ant.0)
FRNDINT	0 := redondear(0)
FSCALE	0 := 0 * 2.0 ** 1
FSETPM	setear modo de protección
FSIN	387 sólo: push, 1/0 := seno(ant.0)
FSINCOS	387 sólo: push, 1 := seno, 0 := cos(ant.0)
FSQRT	0 := raíz cuadrada de 0
FST i	i := 0
FST mem4r	mem4r := 0
FST mem8r	mem8r := 0
FSTCW mem2i	mem2i := palabra de control
FSTP i	i := 0, pop
FSTP mem10r	mem10r := 0, pop
FSTP mem4r	mem4r := 0, pop
FSTP mem8r	mem8r := 0, pop
FSTSW AX	AX := palabra de estado
FSTSW mem2i	mem2i := palabra de estado
FSUB	1 := 1 - 0, pop
FSUB i	0 := 0 - i
FSUB i,0	i := i - 0

FSUB 0,i	0 := 0 - i
FSUB mem4r	0 := 0 - mem4r
FSUB mem8r	0 := 0 - mem8r
FSUBP i,0	i := i - 0, pop
FSUBR	1 := 0 - 1, pop
FSUBR i	0 := i - 0
FSUBR i,0	i := 0 - i
FSUBR 0,i	0 := i - 0
FSUBR mem4r	0 := mem4r - 0
FSUBR mem8r	0 := mem8r - 0
FSUBRP i,0	i := 0 - i, pop
FTST	comparar 0 - 0.0
FWAIT	esperar para 87 listo (sólo 8088(86))
FXAM	C3 -- C0 := tipo de 0
FXCH	intercambio 0 y 1
FXCH 0,i	intercambio 0 y i
FXCH i	intercambio 0 y i
FXCH i,0	ntercambio 0 y i
FXTRACT	push, 1 := exponente, 0 := signifcando
FYL2X	0 := 1 * log base 2.0 de 0, pop
FYL2XP1	0 := 1 * log base 2.0 de (0+1.0), pop

***(2) Referencias**

ant.	anterior
mem4r	dirección u "offset" de memoria con un dato de 4 bytes (DobleWord, definido con la directiva "dd").
mem8r	dirección u "offset" de memoria con un dato de 8 bytes (QuadWord, definido con la directiva "dq").
mem10r	dirección u "offset" de memoria con un dato de 10 bytes, definido con la directiva "dt".
mem10d	dirección u "offset" de memoria con dato en BCD, el que será reconocido por las instrucciones FBLD y FBSTP.
mem4i, mem2i	corresponde con números enteros de 4 y 2 bytes respectivamente, con signo.
mem14 y mem94	buffers de 14 y 94 bytes que contienen el estado de la máquina 80x87.
0, 1, 2...	Posición en la pila (STn)

Existen programas assembler para operar con numeros en pto flotante y string:

Macros2.asm
Number.asm

Ambos se deben incluir en el programa assembler correspondiente en las primeras lineas con la sentencia include

```
include  macros2.asm          ;incluye macros
include  number.asm           ;incluye el asm para impresion de numeros
```

El programa number.asm utiliza ciertos procedimientos externos a el que se encuentran en la rutina

Numbers.asm

Para poder ensamblar todo, al compilarse con el TASM, se debera incluir desde el dos

tasm Compilador.asm

tlink /3 compilador.obj numbers.obj /v /s /m

Tabla de comparaciones

JE	Li = Ld
JNE	Li ≠ Ld
JB -JNAE	Li < Ld
JBE- JNA	Li ≤Ld
JA-JNBE	Li > Ld
JA-JNB	Li ≥Ld

Comparación simple	Assembler (108)

DECLARE
real var2;
real var1;
ENDDECLARE
 	
if var1 > 3
{
var1 = 3.5
 	
var2 = 4.8
}
endif
	.MODEL LARGE
	.386
	.STACK 200h
	
	MAXTEXTSIZE equ 50
	
	.DATA
	
	var2_ dd ?
	var1_ dd ?
	zv2 dd 040400000h;valor=3
	zv3 dd 040600000h;valor=3.5
	zv4 dd 04099999ah;valor=4.8
	aux db ?
	
	.CODE
	mov AX,@DATA ; inicializa el segmento de datos
	mov DS,AX
	mov es,ax ;
	fld var1_
	fld zv2
	fxch
	fcomp
	fstsw ax
	ffree st(0)
	sahf
	jbe _eti11
	fld zv3
	fstp var1_
	fld zv4
	fstp var2_
	_eti11:
	
	mov ax, 4C00h ; Indica que debe terminar la ejecución
	int 21h

Suma	assembler

DECLARE
real var2;
real var1;
ENDDECLARE
 	
var2 = var1 + 3.5
	.MODEL LARGE
	.386
	.STACK 200h
	
	MAXTEXTSIZE equ 50
	
	.DATA
	
	var2_ dd ?
	var1_ dd ?
	zv2 dd 040600000h;valor=3.5
	aux db ?
	
	.CODE
	mov AX,@DATA ; inicializa el segmento de datos
	mov DS,AX
	mov es,ax ;
	fld var1_

	fld zv2 fadd fstp var2_ mov ax, 4C00h ; Indica que debe terminar la ejecución int 21h
Resta y Division	assembler

DECLARE
real var2;
real var1;
ENDDECLARE
 	
var2 = var1 - 3.5
	.MODEL LARGE
	.386
	.STACK 200h
	
	MAXTEXTSIZE equ 50
	
	.DATA
	
	var2_ dd ?
	var1_ dd ?
	zv2 dd 040600000h;valor=3.5
	aux db ?
	
	.CODE
	mov AX,@DATA ; inicializa el segmento de datos
	mov DS,AX
	mov es,ax ;
	fld var1_
	fld zv2
	fsub
	fstp var2_
	
	mov ax, 4C00h ; Indica que debe terminar la ejecución
	int 21h

Con and y or (Polaca grupo 216)	Assembler (Grupo 108)

DECLARE
real var2;
real var1;
ENDDECLARE
 	
if 1 <> 2 AND var1 > var2
{
var1= 3
else
var1 = 5
}
endif
 	
polaca
 	
1 2 <> a b > AND SINO2 BN 3 A := IS3 BI SINO2
 	
5 b := IS3
 	
naranja etiquetas
azul saltos
	.MODEL LARGE
	.386
	.STACK 200h
	
	MAXTEXTSIZE equ 50
	
	.DATA
	
	var2_ dd ?
	var1_ dd ?
	zv2 dd 03f800000h;valor=1
	zv3 dd 040000000h;valor=2
	zv4 dd 040400000h;valor=3
	zv5 dd 040a00000h;valor=5
	aux db ?
	
	.CODE
	mov AX,@DATA ; inicializa el segmento de datos
	mov DS,AX
	mov es,ax ;
	fld zv2
	fld zv3
	fxch

	fcomp fstsw ax ffree st(0) sahf je fld var1_ fld var2_ fxch fcomp fstsw ax ffree st(0) sahf jbe fld zv4 fstp var1_ mov ax, 4C00h ; Indica que debe terminar la ejecución int 21h
OR	

DECLARE
real var2;
real var1;
ENDDECLARE
 	
if 1 <> 2 OR var1 > var2
{
var1= 3
else
var1 = 5
}
endif
	.MODEL LARGE
	.386
	.STACK 200h
	
	MAXTEXTSIZE equ 50
	
	.DATA
	
	var2_ dd ?
	var1_ dd ?
	zv2 dd 03f800000h;valor=1
	zv3 dd 040000000h;valor=2
	zv4 dd 040400000h;valor=3
	zv5 dd 040a00000h;valor=5
	aux db ?
	
	.CODE
	mov AX,@DATA ; inicializa el segmento de datos
	mov DS,AX
	mov es,ax ;
	fld zv2
	fld zv3
	fxch
	fcomp
	fstsw ax
	ffree st(0)
	sahf
	jne _eti10
	fld var1_
	fld var2_
	fxch
	fcomp
	fstsw ax
	ffree st(0)
	sahf
	jbe
	_eti10:
	fld zv4

	fstp var1_ mov ax, 4C00h ; Indica que debe terminar la ejecución int 21h