Carga y almacenamiento

Hay varias instrucciones que cargan datos en el tope de la pila de registro del coprocesador:

`FLD` *source*	Carga un número de punto flotante de la memoria en el tope de la pila. La *source* puede ser un número de precisión simple doble o extendida o un registro del coprocesador
`FILD` *source*	Lee un *entero* de memoria, lo convierte a punto flotante y almacena el resultado en el tope de la pila. *source* puede ser una palabra, palabra doble o una palabra cuádruple.
`FLD1`	almacena un uno en el tope de la pila.
`FLDZ`	almacena un cero en el tope de la pila.

Hay también varias instrucciones que almacenan datos de la pila en memoria. Algunas de estas instrucciones también *sacan* el número de la pila.

`FST` *dest*	Almacena el tope de la pila (`ST0`) en memoria. El *destino* puede ser un número de precisión simple o doble o un registro de coprocesador.
`FSTP` *dest*	Almacena el tope de la pila en la memoria tal como `FST`; sin embargo luego de que se almacena el número, su valor se saca de la pila. El *destino* puede ser un número de precisión simple o doble o un registro del coprocesador.
`FIST` *dest*	Almacena el valor del tope de la pila convertido a entero en memoria. El *destino* puede ser una palabra o palabra doble. La pila no sufre ningún cambio. Cómo se convierte el número de punto flotante a entero depende de algunos bits de la *palabra de control* del coprocesador. Este es un registro especial (no de punto flotante) que controla cómo trabaja el coprocesador. Por defecto, la palabra de control se inicia tal que redondea al entero más cercano cuando se convierte a entero. Sin embargo las instrucciones `FSTCW` (Store Control Word) y `FDLCW` (load control word) se pueden usar para cambiar este comportamiento.
`FISTP` *dest*	Lo mismo que `FIST`, excepto por dos cosas. El tope de la pila se saca y el *destino* también puede ser una palabra cuádruple.

Hay otras dos instrucciones que pueden mover o quitar datos de la pila en sí misma.

`FXCH ST`*n*	intercambia los valores en `ST0` y `ST`*n* en la pila (donde n es el número del registro de 1 a 7).
`FFREE ST`*n*	libera un registro en la pila, marcando el registro como no usado o vacío.

```
1  segment .bss
2  array        resq SIZE
3  sum          resq 1
4
5  segment .text
6        mov    ecx, SIZE
7        mov    esi, array
8        fldz                  ; ST0 = 0
9  lp:
10       fadd   qword [esi]    ; ST0 += *(esi)
11       add    esi, 8         ; se mueve al próximo dobule
12       loop   lp
13       fstp   qword sum      ; alamcena el resultado en sum
```

Figura 6.5: Ejemplo sumar arreglo

Suma y resta

Cada una de las instrucciones de suma calcula la suma de `ST0` y otro operando, el resultado siempre se almacena en un registro del coprocesador.

`FADD` *src*	`ST0 +=` *src*. *src* puede ser cualquier registro del coprocesador o un número de precisión simple o doble en memoria.
`FADD` *dest*`, ST0`	*dest* `+= ST0`. El *destino* puede ser cualquier registro del coprocesador
`FADDP` *dest* o `FADDP` *dest*`, ST0`	*dest* `+= ST0` entonces se saca de la pila. El *destino* puede ser cualquier registro del coprocesador.
`FIADD` *src*	`ST0 += (float)` *src*. suma un entero con `ST0`. *src* debe ser una palabra o una palabra doble en memoria.

Hay el doble de instrucciones de resta que de suma porque el orden de los operandos es importante. (¡$a + b = b + a$, pero $a - b \neq b - a$!). Por cada instrucción, hay una alterna que resta en el orden inverso. Esas instrucciones al revés todas culminan con `R` o `RP`. La Figura muestra un pequeño código que suma los elementos de un arreglo de dobles. En las líneas 10 y 13, uno debe especificar el tamaño del operando de memoria. De otra forma el ensamblador no conocería si el operando de memoria era un float (dword) o double (qword).

`FSUB` *src*	`ST0 -=` *src*. *src* puede ser cualquier registro del coprocesador o un número de presición doble o simple en memoria.
`FSUBR` *src*	`ST0 =` *src* `- ST0`. *src* puede ser cualquier registro del coprocesador o un número de presición doble o simple en memoria.
`FSUB` *dest*`, ST0`	*dest* `-= ST0`. *dest* puede ser cualquier registro del coprocesador.
`FSUBR` *dest*`, ST0`	*dest* `= ST0 -` *dest*. *dest* puede ser cualquier registro del coprocesador.
`FSUBP` *dest* o `FSUBP` *dest*`, ST0`	*dest* `-= ST0` entonces sale de la pila. *dest* puede ser cualquier registro del coprocesador.
`FSUBRP` *dest* o `FSUBRP` *dest*`, ST0`	*dest* `= ST0 -` *dest* entonces sale de la pila. *dest* puede ser cualquier registro del coprocesador.
`FISUB` *src*	`ST0 -= (float)` *src*. Resta un entero de `ST0`. *src* debe ser una palabra o palabra doble en memoria.
`FISUBR` *src*	`ST0 = (float)` *src* `- ST0`. Resta `ST0` de un entero. *src* debe ser una palabra o palabra doble en memoria.

Multiplicación y división

La instrucción de multiplicación son totalmente análogas a las instrucciones de suma.

`FMUL` *src*	`ST0 *=` *src*. *src* puede ser cualquier registro del coprocesador o un número de precisión simple o doble en memoria.
`FMUL` *dest*`, ST0`	*dest* `*= ST0`. *dest* puede ser cualquier registro del coprocesador.
`FMULP` *dest* o `FMULP` *dest*`, ST0`	*dest* `*= ST0` entonces sale de la pila. *dest* puede ser cualquier registro del coprocesador.
`FIMUL` *src*	`ST0 *= (float)` *src*. Multiplica un entero con `ST0`. *src* debe ser una palabra o palabra doble en memoria.

No es sorprendente que las instrucciones de división son análogas a las instrucciones de resta. La división por cero de un infinito.

`FDIV` *src*	`ST0 /=` *src*. *src* puede ser cualquier registro del coprocesador o un número de precisión simple o doble en memoria.
`FDIVR` *src*	`ST0 =` *src* `/ ST0`. *src* puede ser cualquier registro del coprocesador o una número de precisión simple o doble en memoria.
`FDIV` *dest*`, ST0`	*dest* `/= ST0`. *dest* puede ser cualquier registro del coprocesador.
`FDIVR` *dest*`, ST0`	*dest* `= ST0 /` *dest*. *dest* puede ser cualquier registro del coprocesador.
`FDIVP` *dest* o `FDIVP` *dest*`, ST0`	*dest* `/= ST0` entonces sale de la pila. *dest* puede ser cualquier registro del coprocesador.
`FDIVRP` *dest* o `FDIVRP` *dest*`, ST0`	*dest* `= ST0 /` *dest* entonces sale de la pila. *dest* puede ser cualquier registro del coprocesador.
`FIDIV` *src*	`ST0 /= (float)` *src*. Divide `ST0` por un entero. *src* debe ser una palabra o palabra doble en memoria.
`FIDIVR` *src*	`ST0 = (float)` *src* `/ ST0`. Divide un entero por `ST0`. *src* debe ser una palabra o palabra doble en memoria.

Comparaciones

El coprocesador también realiza comparaciones de números de punto flotante. La fórmula de instrucciones `FCOM` hacen esta operación.

`FCOM` *src*	compara `ST0` y *src*. *src* puede ser un registro del coprocesador o un float o double en memoria.
puede ser un `FCOMP` *src*	compara `ST0` y *src*, luego sale de la pila. *src* puede ser un registro del coprocesador o un float o double en memoria.
`FCOMPP`	compara `ST0` y `ST1`, entonces saca de la pila dos veces.
`FICOM` *src*	compara `ST0` y `(float)` *src*. *src* puede ser una palabra o palabra dobel en memoria.
`FICOMP` *src*	compara `ST0` y `(float)`*src*, entonces saca de la pila. *src* puede ser una palabra o palabra doble entera en memoria.
`FTST`	compara `ST0` and 0.

Estas instrucciones cambian los bits C_0, C_1, C_2 y C_3 del registro de estado del coprocesador. Desafortunadamente no es posible que la CPU acceda a estos bits directamente. Las instrucciones de salto condicional usan el registro FLAGS, no el registro de estado del coprocesador. Sin embargo es relativamente fácil transferir los bits de la palabra de estado a los bits correspondientes del registro FLGS usando algunas instrucciones nuevas.