Instrucciones de Pentium

Diseño de Compiladores I

Registros del procesador

Registros de 32 bits:

- EAX
- EBX
- ECX
- EDX

En una instrucción, se pueden usar 32, 16 u 8 bits.

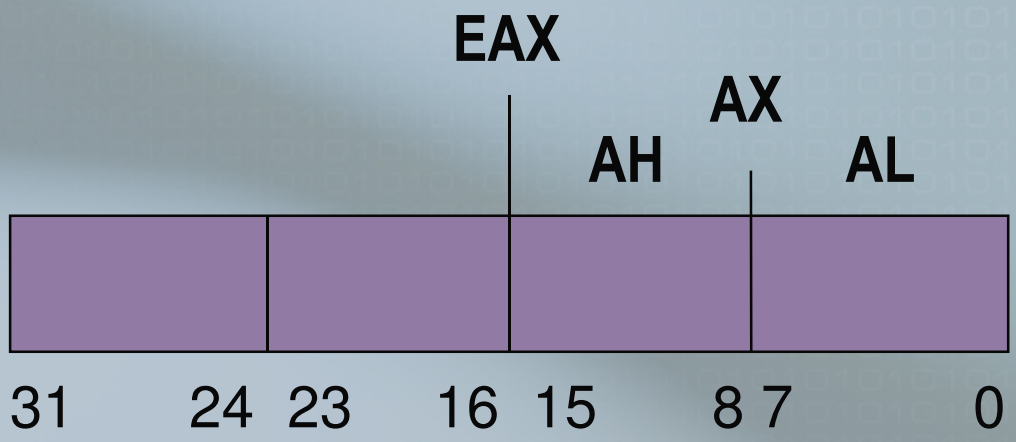
Registros del procesador

- Registros de 16 bits:
  - AX (16 bits menos significativos de EAX)
  - BX (16 bits menos significativos de EBX)
  - CX (16 bits menos significativos de ECX)
  - DX (16 bits menos significativos de EDX)

Registros del procesador

- Registros de 8 bits:
  - AH (8 bits más significativos de AX)
  - AL (8 bits menos significativos de AX)
  - BH (8 bits más significativos de BX)
  - BL (8 bits menos significativos de BX)
  - CH (8 bits más significativos de CX)
  - CL (8 bits menos significativos de CX)
  - DH (8 bits más significativos de DX)
  - DL (8 bits menos significativos de DX)

Registros del Procesador



Registros de segmentos

- **CS** (de codigo) – indica el segmento donde están las instrucciones
- **DS** (de datos) – indica el segmento donde están las variables
- **SS** (de stack) – indica el segmento donde está el stack
- **ES** (de strings o general) – para segmentos definidos por usuario

Instrucciones de transferencia de datos (no afectan flags)

- **MOV** *dest,src*

  Copia el contenido del operando fuente *(src)* en el destino *(dest)*.
  Operación: *dest* ← *src*
  Las posibilidades son:

- **MOV** *reg,{reg|mem|inmed}*
- **MOV** *mem,{reg|inmed}*
- **MOV** *{reg16|mem16},{CS|DS|ES|SS}*
- **MOV** *{DS|ES|SS},{reg16|mem16}*

Instrucciones de transferencia de datos (no afectan flags)

- **PUSH** *src*

  Pone el valor en el tope del stack.
  Operación: *SP* ← *SP - 2*, *[SP+1:SP]* ← *src*
  donde src = {reg16|mem16|reg32|mem32|CS|DS|ES|SS}.

- **POP** *dest*

  Retira el valor del tope del stack poniéndolo en el lugar indicado.
  Operación: *dest* ← *[SP+1:SP]*, *SP* ← *SP + 2*
  donde dest = {reg16|mem16|reg32|mem32|DS|ES|SS}.

Instrucciones de transferencia de datos (no afectan flags)

- **LAHF**

  Copia en el registro AH la imagen de los ocho bits menos significativos del registro de indicadores.
  Operación: *AH ← SF:ZF:X:AF:X:PF:X:CF*

- **SAHF**

  Almacena en los ocho bits menos significativos del registro de indicadores el valor del registro AH.
  Operación: *SF:ZF:X:AF:X:PF:X:CF ← AH*

Instrucciones de transferencia de datos (no afectan flags)

- **PUSHF**

  Almacena los flags en la pila.
  Operación: *SP* ← *SP - 2*, *[SP+1:SP]* ← *Flags*

- **POPF**

  Pone en los flags el valor que hay en la pila.
  Operación: *Flags* ← *[SP+1:SP]*, *SP* ← *SP + 2*

Flags

- CF - Carry flag.

  Se setea si una operación arimética genera carry. Indica overflow para aritmética de enteros sin signo.
- PF - Parity flag.

  Se setea si el byte menos significativo del resultado contiene un número par de bits 1.
- ZF - Zero flag.

  Se setea si el resultado es cero.
- SF - Sign flag.

  Se setea igual al bit más significativo del resultado, que es el bit de signo de un entero con signo (0 indica un valor positivo y 1 indica un valor negativo).
- OF - Overflow flag.

  Se setea si un entero es demasiado grande para números positivos o demasiado pequeño para números negativos (excluyendo el bit de signo) para entrar en el operando destino. Indica una condición de overflow para aritmética de enteros con signo.

Instrucciones aritméticas
(Afectan los flags AF, CF, OF, PF, SF, ZF)

- **ADD** *dest,src*

  Operación: *dest* ← *dest + src*.

- **ADC** *dest,src*

  Operación: *dest* ← *dest + src + CF*.

donde *dest* = *{reg|mem}* y *src* = *{reg|mem|inmed}* no pudiendo ambos operandos estar en memoria.

Instrucciones aritméticas
(Afectan los flags AF, CF, OF, PF, SF, ZF)

- **SUB** *dest,src*

  Operación: *dest* ← *dest - src*.

- **SBB** *dest,src*

  Operación: *dest* ← *dest - src - CF*.

  donde *dest* = *{reg|mem}* y *src* = *{reg|mem|inmed}* no pudiendo ambos operandos estar en memoria.

Instrucciones aritméticas
(Afectan los flags AF, CF, OF, PF, SF, ZF)

- **CMP** *dest,src*

  Operación: *dest - src (sólo afecta flags)*.

  donde *dest* = *{reg|mem}* y *src* = *{reg|mem|inmed}* no pudiendo ambos operandos estar en memoria.

Instrucciones aritméticas (Afectan los flags AF, CF, OF, PF, SF, ZF)

MUL *dest, src*

donde *dest* = *{reg}* y *src* = *{reg|mem}*

- **Operandos de 8 bits -** *dest* debe ser AL

  Realiza una multiplicación con operandos no signados de 8 por 8 bits. El resultado tendrá 16 bits.

  Operación: *AX ← AL * {reg8|mem8}*. CF=OF=0 si AH = 0, CF=OF=1 en caso contrario. AF, PF, SF, ZF quedan indefinidos.

- **Operandos de 16 bits -** *dest* debe ser AX

  Realiza una multiplicación con operandos no signados de 16 por 16 bits. El resultado tendrá 32 bits.

  Operación: *DX:AX ← AX * {reg16|mem16}.* CF=OF=0 si DX = 0, CF=OF=1 en caso contrario. AF, PF, SF, ZF quedan indefinidos.

- **Operandos de 32 bits -** *dest* debe ser EAX

  Realiza una multiplicación con operandos no signados de 32 por 32 bits. El resultado tendrá 64 bits.

  Operación: *EDX:EAX ← EAX * {reg32|mem32}*. CF=OF=0 si EDX = 0, CF=OF=1 en caso contrario. AF, PF, SF, ZF quedan indefinidos.

Instrucciones aritméticas
(Afectan los flags AF, CF, OF, PF, SF, ZF)

IMUL *dest, src*

donde *dest* = *{reg}* y *src* = *{reg|mem|inmed}*

- **Operandos de 8 bits -** *dest* debe ser AL

  Realiza una multiplicación con operandos con signo de 8 por 8 bits. El resultado tendrá 16 bits.

  Operación: *AX ← AL * {reg8|mem8|inmed}* realizando la multiplicación con signo. CF = OF = 0 si el resultado entra en un byte, en caso contrario valdrán 1. AF, PF, SF, ZF quedan indefinidos.

- **Operandos de 16 bits -** *dest* debe ser AX

  Realiza una multiplicación con operandos con signo de 16 por 16 bits. El resultado tendrá 32 bits.

  Operación: *DX:AX ← AX * {reg16|mem16|inmed}* realizando la multiplicación con signo. CF = OF = 0 si el resultado entra en dos bytes, en caso contrario valdrán 1. AF, PF, SF, ZF quedan indefinidos.

- **Operandos de 32 bits -** *dest* debe ser EAX

  Realiza una multiplicación con operandos con signo de 32 por 32 bits. El resultado tendrá 64 bits.

  Operación: *EDX:EAX ← EAX * {reg32|mem32|inmed}* realizando la multiplicación con signo. CF = OF = 0 si el resultado entra en cuatro bytes, en caso contrario valdrán 1. AF, PF, SF, ZF quedan indefinidos

Instrucciones aritméticas

DIV *oper*

donde *oper = {reg/mem}*

- **Operando de 8 bits no signado**
  - El dividendo debe estar en AX. El resultado tendrá 8 bits.
  - Cociente en AL. Resto en AH.
- **Operando de 16 bits no signado**
  - El dividendo debe estar en DX:AX. El resultado tendrá 16 bits.
  - Cociente en AX. Resto en DX.
- **Operando de 32 bits no signado**
  - El dividendo debe estar en EDX:EAX. El resultado tendrá 32 bits.
  - Cociente en EAX. Resto en EDX.

Instrucciones aritméticas

IDIV *oper*

donde *oper* = *{reg/mem}*

- **Operando de 8 bits con signo**
  - El dividendo debe estar en AX. El resultado tendrá 8 bits.
  - Cociente en AL. Resto en AH.
- **Operando de 16 bits con signo**
  - El dividendo debe estar en DX:AX. El resultado tendrá 16 bits.
  - Cociente en AX. Resto en DX.
- **Operando de 32 bits con signo**
  - El dividendo debe estar en EDX:EAX. El resultado tendrá 32 bits.
  - Cociente en EAX. Resto en EDX.

Instrucciones aritméticas

- **CBW**

  Extiende el signo de AL en AX. No se afectan los flags.
- **CWD**

  Extiende el signo de AX en DX:AX. No se afectan flags.
- **CWDE**

  Extiende el signo de AX en EAX. No se afectan flags.
- **CDQ**

  Extiende el signo de EAX en EDX:EAX. No se afectan flags.

Instrucciones lógicas
(Afectan los flags AF, CF, OF, PF, SF, ZF)

- **AND** *dest,src*

  Operación: *dest* ← *dest and src*.
- **TEST** *dest,src*

  Operación: *dest and src*. Sólo afecta flags.
- **OR** *dest,src*

  Operación: *dest* ← *dest or src*.
- **XOR** *dest,src*

  Operación: *dest* ← *dest xor src*.

Las cuatro instrucciones anteriores ponen CF = OF = 0, AF queda indefinido y PF, SF y ZF dependen del resultado.

Instrucciones de transferencia de control (no afectan los flags)

- **JMP** *label*

  Saltar hacia la dirección *label*.

Saltos condicionales aritméticos (usar después de CMP)

Aritmética signada (con números positivos, negativos y cero)

- **JL** *etiqueta* /**JNGE** *etiqueta*

  Saltar a *etiqueta* si es menor.

- **JLE** *etiqueta* /**JNG** *etiqueta*

  Saltar a *etiqueta* si es menor o igual.

Saltos condicionales aritméticos (usar después de CMP)

Aritmética signada (con números positivos, negativos y cero)

- **JE** *etiqueta*

  Saltar a *etiqueta* si es igual.

- **JNE** *etiqueta*

  Saltar a *etiqueta* si es distinto.

Saltos condicionales aritméticos (usar después de CMP)

Aritmética signada (con números positivos, negativos y cero)

- **JGE** *etiqueta* /**JNL** *etiqueta*

  Saltar a *etiqueta* si es mayor o igual.

- **JG** *etiqueta* /**JNLE** *etiqueta*

  Saltar a *etiqueta* si es mayor.

Saltos condicionales aritméticos (usar después de CMP)

Aritmética sin signo (con números positivos y cero)

- **JB** *etiqueta* /**JNAE** *etiqueta*

  Saltar a *etiqueta* si es menor.

- **JBE** *etiqueta* /**JNA** *etiqueta*

  Saltar a *etiqueta* si es menor o igual.

Saltos condicionales aritméticos (usar después de CMP)

Aritmética sin signo (con números positivos y cero)

- **JE** *etiqueta*

  Saltar a *etiqueta* si es igual.

- **JNE** *etiqueta*

  Saltar a *etiqueta* si es distinto.

Saltos condicionales aritméticos (usar después de CMP)

Aritmética sin signo (con números positivos y cero)

- **JAE** *etiqueta*/**JNB** *etiqueta*

  Saltar a *etiqueta* si es mayor o igual.

- **JA** *etiqueta*/**JNBE** *etiqueta*

  Saltar a *etiqueta* si es mayor.

Saltos condicionales según el valor de los indicadores:

- **JC** *label*

  Saltar si hubo arrastre/préstamo (CF = 1).

- **JNC** *label*

  Saltar si no hubo arrastre/préstamo (CF = 0).

Saltos condicionales según el valor de los indicadores:

- **JZ** *label*

  Saltar si el resultado es cero (ZF = 1).

- **JNZ** *label*

  Saltar si el resultado no es cero (ZF = 0).

Saltos condicionales según el valor de los indicadores:

- **JS** *label*

  Saltar si el signo es negativo (SF = 1).

- **JNS** *label*

  Saltar si el signo es positivo (SF = 0).

Saltos condicionales según el valor de los indicadores:

Aritmética signada (con números positivos, negativos y cero)

- **JO** *label*

  Saltar si hubo desbordamiento (OF = 1).

- **JNO** *label*

  Saltar si no hubo desbordamiento (OF = 0).

Directivas (Instrucciones para el ensamblador)

Definición de datos

Ubica memoria para un ítem de datos y opcionalmente asocia un nombre simbólico con esa dirección de memoria y/o genera el valor inicial para ese ítem.

- *[nombre]* **DB** *valor_inicial [, valor_inicial...]*
  donde *valor_inicial* puede ser una cadena o una expresión numérica cuyo resultado esté entre -255 y 255.

  Define datos de 8 bits (1 byte)

Directivas (Instrucciones para el ensamblador)

Definición de datos

- *[nombre]* **DW** *valor_inicial [, valor_inicial...]*
  Define datos de 16 bits (2 bytes)
- *[nombre]* **DD** *valor_inicial [, valor_inicial...]*
  Define datos de 32 bits (4 bytes)

Directivas (Instrucciones para el ensamblador)

Definición de datos

- *[nombre]* **DQ** *valor_inicial [, valor_inicial...]*
  define datos de 64 bits (8 bytes)
- *[nombre]* **DT** *valor_inicial [, valor_inicial...]*
  define datos de 80 bits (10 bytes)
- *[nombre]* **DS** *nro_de_bytes*
  define datos de nro. de bytes

Directivas (Instrucciones para el ensamblador)

Definición de datos

- Si se desea que no haya valor inicial, deberá utilizarse el símbolo **?**.

- Otra forma de expresar el valor inicial es:
  cuenta **DUP** *(valor_inicial [, valor_inicial...])*

  donde *cuenta* es la cantidad de veces que debe repetirse lo que está entre paréntesis.

Control del ensamblador

- **END** *[etiqueta]*: Debe ser la última sentencia del código fuente. La *etiqueta* indica dónde debe comenzar la ejecución del programa.

  Si el programa se compone de varios módulos, sólo el módulo que contiene la dirección de arranque del programa debe contener la directiva **END** *etiqueta*.

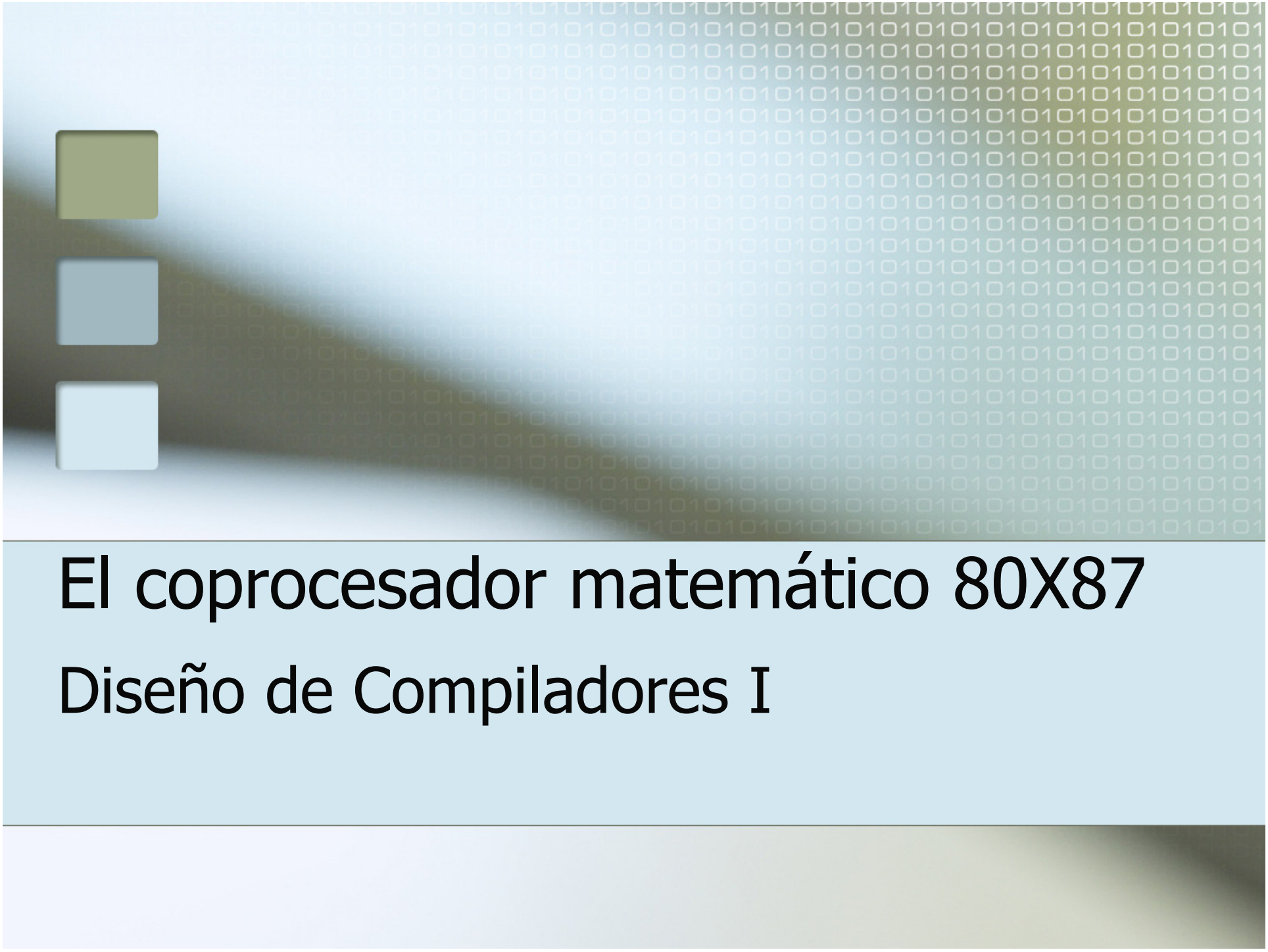
  Los demás módulos deberán terminar con la directiva **END** (sin *etiqueta*).

Estructura de un programa

```
.MODEL small            ; Indica el tamaño de programa
.STACK 200h             ; Inicializa Stack en dir. indicada
.DATA                   ; Indica zona de datos
   ; Aquí se definen variables y datos
.CODE                   ; Indica inicio zona de código
   ; Aquí viene el código
START:                  ; Label que indica inicio del "main"
   mov   ax,@data       ; Mueve a ds la dirección del segmento de datos
   mov   ds,ax          ; utilizando el reg. ax como intermediario
   …
   …                    ; Código del programa
END START
```

Emisión de Mensajes

Ver ejemplo: asmwhello.asm



El coprocesador matemático 80X87

Diseño de Compiladores I

Introducción

- El coprocesador aritmético 80X87 aumenta el juego de instrucciones del 80X86 mejorando su capacidad de tratamiento de números.
- Se utiliza como procesador paralelo junto al 80X86 añadiendo 8 registros de punto flotante de 80 bits así como instrucciones adicionales.

Registros

- Los ocho registros se organizan como una pila.
- Los nombres de los registros son **ST(0)**, **ST(1)**, **ST(2)**, ..., **ST(7)**. El nombre simbólico **ST** (*Stack Top*) es equivalente a **ST(0)**.
- Al poner un número en la pila, **ST(0)** contendrá el número recién ingresado, **ST(1)** será el valor anterior de **ST(0)**, **ST(2)** será el valor anterior de **ST(1)**, y así sucesivamente, con lo que se perderá el valor anterior de **ST(7)**.

Juego de instrucciones del 80X87

Instrucciones de transferencia de datos

- **FLD** *mem*

  Introduce una copia de *mem* en *ST*. La fuente debe ser un número real en punto flotante de 4, 8 ó 10 bytes. Este operando se transforma automáticamente al formato real temporal.

- **FLD** *ST(num)*

  Introduce una copia de *ST(num)* en *ST*.

- **FILD** *mem*

  Introduce una copia de *mem* en *ST*. La fuente debe ser un operando de memoria de 2, 4 u 8 bytes, que se interpreta como un número entero y se convierte al formato real temporal.

Instrucciones de transferencia de datos

- **FST** *mem*

  Copia *ST* a *mem* sin afectar el puntero de pila. El destino puede ser un operando real de 4, 8 o 10 bytes.

- **FST** *ST(num)*

  Copia *ST* al registro especificado.

- **FIST** *mem*

  Copia *ST* a *mem*. El destino debe ser un operando de 2, 4 u 8 bytes y se convierte automáticamente el número en formato temporal real a entero.

Instrucciones de transferencia de datos

- **FSTP** *mem*

  Extrae una copia de *ST* en *mem*. El destino puede ser un operando de memoria de 4, 8 ó 10 bytes, donde se carga el número en punto flotante.

- **FSTP** *ST(num)*

  Extrae *ST* hacia el registro especificado.

- **FISTP** *mem*

  Extrae una copia de *ST* en *mem*. El destino debe ser un operando de memoria de 2, 4 u 8 bytes y se convierte automáticamente el número en formato temporal real a entero.

Instrucciones de transferencia de datos

- **FXCH**

  Intercambia *ST(1)* y *ST*.

- **FXCH** *ST(num)*

  Intercambia *ST(num)* y *ST*.

Instrucciones de carga de constantes

- **FLDZ**

  Introduce el número cero en *ST*.

- **FLD1**

  Introduce el número uno en *ST*.

- **FLDPI**

  Introduce el valor de pi en *ST*.

Instrucciones aritméticas

- **FADD**

  Hace ST(1) más ST, ajusta el puntero de pila y pone el resultado en ST, por lo que ambos operandos se destruyen.

- **FADD** *mem*

  Hace ST $\leftarrow$ ST + [mem]. En *mem* deberá haber un número real en punto flotante.

- **FIADD** *mem*

  Hace ST $\leftarrow$ ST + [mem]. En *mem* deberá haber un número entero en complemento a dos.

Instrucciones aritméticas

- **FADD** *ST(num), ST*

  Realiza ST(num) ← ST(num) + ST.

- **FADD** *ST, ST(num)*

  Realiza ST ← ST + ST(num).

- **FADDP** *ST(num), ST*

  Realiza ST(num) ← ST(num) + ST y retira el valor de ST de la pila, con lo que ambos operandos se destruyen.

Instrucciones aritméticas

- **FSUB**

  Hace ST(1) menos ST, ajusta el puntero de pila y pone el resultado en ST, por lo que ambos operandos se destruyen.

- **FSUB** *mem*

  Hace ST ← ST - [mem]. En *mem* deberá haber un número real en punto flotante.

- **FISUB** *mem*

  Hace ST ← ST - [mem]. En *mem* deberá haber un número entero en complemento a dos.

Instrucciones aritméticas

- **FSUB** *ST(num), ST*

  Realiza ST(num) ← ST(num) - ST.

- **FSUB** *ST, ST(num)*

  Realiza ST ← ST - ST(num).

- **FSUBP** *ST(num), ST*

  Realiza ST(num) ← ST(num) - ST y retira el valor de ST de la pila, con lo que ambos operandos se destruyen.

Instrucciones aritméticas

- **FSUBR**

  Hace ST menos ST(1), ajusta el puntero de pila y pone el resultado en ST, por lo que ambos operandos se destruyen.

- **FSUBR** *mem*

  Hace ST ← [mem] - ST. En *mem* deberá haber un número real en punto flotante.

- **FISUBR** *mem*

  Hace ST ← [mem] - ST. En *mem* deberá haber un número entero en complemento a dos.

Instrucciones aritméticas

- **FSUBR** *ST(num), ST*

  Realiza ST(num) ← ST - ST(num).

- **FSUBR** *ST, ST(num)*

  Realiza ST ← ST(num) - ST.

- **FSUBRP** *ST(num), ST*

  Realiza ST(num) ← ST - ST(num) y retira el valor de ST de la pila, con lo que ambos operandos se destruyen.

Instrucciones aritméticas

- **FMUL**

  Multiplicar el valor de ST(1) por ST, ajusta el puntero de pila y pone el resultado en ST, por lo que ambos operandos se destruyen.

- **FMUL** *mem*

  Hace ST ← ST * [mem]. En *mem* deberá haber un número real en punto flotante.

- **FIMUL** *mem*

  Hace ST ← ST * [mem]. En *mem* deberá haber un número entero en complemento a dos.

Instrucciones aritméticas

- **FMUL** *ST(num), ST*

  Realiza ST(num) ← ST(num) * ST.

- **FMUL** *ST, ST(num)*

  Realiza ST ← ST * ST(num).

- **FMULP** *ST(num), ST*

  Realiza ST(num) ← ST(num) * ST y retira el valor de ST de la pila, con lo que ambos operandos se destruyen.

Instrucciones aritméticas

- **FDIV**

  Dividir el valor de ST(1) por ST, ajusta el puntero de pila y pone el resultado en ST, por lo que ambos operandos se destruyen.

- **FDIV** *mem*

  Hace ST ← ST / [mem]. En *mem* deberá haber un número real en punto flotante.

- **FIDIV** *mem*

  Hace ST ← ST / [mem]. En *mem* deberá haber un número entero en complemento a dos.

Instrucciones aritméticas

- **FDIV** *ST(num), ST*

  Realiza ST(num) ← ST(num) / ST.

- **FDIV** *ST, ST(num)*

  Realiza ST ← ST / ST(num).

- **FDIVP** *ST(num), ST*

  Realiza ST(num) ← ST(num) / ST y retira el valor de ST de la pila, con lo que ambos operandos se destruyen.

Instrucciones aritméticas

- **FDIVR**

  Hace ST dividido ST(1), ajusta el puntero de pila y pone el resultado en ST, por lo que ambos operandos se destruyen.

- **FDIVR** *mem*

  Hace ST ← [mem] / ST. En *mem* deberá haber un número real en punto flotante.

- **FIDIVR** *mem*

  Hace ST ← [mem] / ST. En *mem* deberá haber un número entero en complemento a dos.

Instrucciones aritméticas

- **FDIVR** *ST(num), ST*

  Realiza ST(num) ← ST / ST(num).

- **FDIVR** *ST, ST(num)*

  Realiza ST ← ST(num) / ST.

- **FDIVRP** *ST(num), ST*

  Realiza ST(num) ← ST / ST(num) y retira el valor de ST de la pila, con lo que ambos operandos se destruyen.

Instrucciones aritméticas

- **FABS**

  Pone el signo de ST a positivo (valor absoluto).

- **FCHS**

  Cambia el signo de ST.

Control del flujo del programa

- **FCOM**

  Compara ST y ST(1).

- **FCOM** *ST(num)*

  Compara ST y ST(num).

- **FCOM** *mem*

  Compara ST y mem. El operando de memoria deberá ser un número real.

- **FICOM** *mem*

  Compara ST y mem. El operando deberá ser un número entero.

Control del flujo del programa

- **FTST**

  Compara ST y cero.

- **FCOMP**

  Compara ST y ST(1) y extrae ST fuera de la pila.

- **FCOMP** *ST(num)*

  Compara ST y ST(num) y extrae ST fuera de la pila.

- **FCOMP** *mem*

  Compara ST y mem y extrae ST fuera de la pila. El operando de memoria deberá ser un número real.

Control del flujo del programa

- **FICOMP** *mem*

  Compara ST y mem y extrae ST fuera de la pila. El operando deberá ser un número entero.

- **FCOMPP**

  Compara ST y ST(1) y extrae dos elementos de la pila, perdiéndose ambos operandos.

Instrucciones de transferencia de datos de control

- **FLDCW** *mem2byte*

  Carga la palabra de control desde la memoria.

- **FSTCW** *mem2byte*

  Almacena la palabra de control en la memoria.

- **FSTSW** *mem2byte*

  Almacena la palabra de estado en la memoria.

Control del flujo del programa

- Después de hacer una comparación en el coprocesador, se debe:
  - Almacenar la palabra de estado en la memoria.
  - Copiar de la memoria al registro de indicadores del procesador.
  - Efectuar la bifurcación condicional.

Ejemplo de comparación

```
…
.DATA                          ; Indica zona de datos
    ; Aquí se definen variables y datos
.CODE                          ; Indica inicio zona de código
    ; Aquí viene el código
START:                         ; Label que indica inicio del "main"
…
FLD mem1                       ; Carga el primer operando de la comparación
FCOM mem2                      ; Compara con el segundo operando
FSTSW aux_mem_2bytes           ; Almacena la palabra de estado en la memoria.
MOV AX , aux_mem_2bytes        ; Copia el contenido en el registro AX
SAHF                           ; Almacena en los ocho bits menos
                               ; significativos del registro de indicadores
                               ; el valor del registro AH.
JB etiqueta_destino_bif        ; Bifurca (en este caso, por menor)
…
etiqueta_destino_bif:          ; Label destino de la bifurcación
…
END START                      ; Fin del "main"
```

Control del Procesador

- **F[N]INIT**

  Inicializa el coprocesador y restaura todas las condiciones iniciales en las palabras de control y de estado. Es una buena idea utilizar esta instrucción al principio y al final del programa.

- **F[N]CLEX**

  Pone a cero los indicadores de excepción y el indicador de ocupado de la palabra de estado. También limpia el indicador de pedido de interrupción del 8087.

Control del Procesador

- **FREE** ST(num)

  Marca el registro especificado como vacío.

- **FNOP**

  Copia ST a sí mismo tomando tiempo de procesamiento sin tener ningún efecto en registros o memoria.