Lenguaje Maquina x86

Instrucciones referencian varios operandos(inmediatos, registros, memoria).

Instrucciones aritmeticas pueden leer/escribir(solo 1 puede estar en memoria).

Las referencias de Memoria se calcula como: MEM[A+Rb+Ri*s] -->A(Rb, Ri, s). Sirven para navegar por tipos de datos.

Las instrucciones tienen longuitud variable.

Hay pocos registros.

Vision del Programador

- Registros: Espacio lineal de 2^32^ posiciones de 1 byte.

  Los registros que hay disponibles son: %eax, %ebx, %ecx, %edx, %esi, %edi;

  Todos estos registros son de 32 bits. Pero el mismo registro(de %eax a %edx) se puede acceder como registro de 16 bits unico (%ax, %bx, %cx) o como dos de 8 bits(%ah, %al, ...).

  Tambien hay algunos registros especiales:

  %esp, %ebp y su version de 16 bits, estan destinados para trabajo en subrutinas.

  %eip esta destinado al program counter.

  %eflags esta destinado a la palabra de estado.

- EEFlags(Codigo Condicion)

  Informacion sobre comportamiento de las instrucion(bit carry, bit de Z, etc..)

- Memoria:

  Codigo Objecto, Datos Programa, Datos SO. Direccionable a nivel de byte.

  Stack del proceso: Se usa para las llamadas a funciones, guardar variables, etc..

  Modos de direccionamiento:

  - Inmediato: $19 (Byte, Word o LongWord)
  - Registros: %eax, %ebx, %ah, ...
    - Memoria: D(rB, rI, s) -> M[rB+rI * s + D] //@BASE + (Index * Escala) + Offset

- Tipos de datos Basicos:

  Hay que destacar que los tipos de datos estan diseñados para byte(int), word(int), longword(int) o quadword(float).

  Se utiliza Little Endian!

RANGOS:	Byte(8 bits)	Word(16)	LongWord(32)
Naturales	255	65.535	4.294.967.215
Enteros(Ca2)	-128 a 127	-32.768 a 32.767	-2.147.483.648 a 2.147.483.647
Reales(IEEE 754)			2.23 * 10^-38^ a 1.79 * 10^38^

- [Repasar IEEE 754]

Operaciones Primitivas: Aritmeticologicas, Transferencia de datos y Saltos.

Codificación Instrucciones(17/69)

OpCode: 1-2 Bytes

Modo: 1 byte

SIB:

- Scale(2 bits): 0, 1, 2, 3 son los possibles valores.
- Index(3 bits)
- Base(3 bits)

  Desplazamiento: 0, 1,2 o 4 Bytes.

  Inmediato: 0, 1, 2 o 4 bytes.

Instrucciones

INSTRU	Desc	Complementos	
MOVx op1, op2	op2 <- op1	**L**ongWord, **W**ord, **B**yte	Movimiento
MOVSxy op1, op2	op2 <-Extsign(op1)	BaW, BaL, WaL	Movimiento
MOVZSxy op1, op2	op2 <-Extzero(op1)	BaW, BaL, WaL	Movimiento
PUSHL op1	%esp <= %esp - 4, M[%esp] <= op1		Empilar
POPL op1	op1 <= M[%esp], %esp <= %esp + 4		Desempilar
LEAL op1, op2	op2 <= &op1	op1: @M	Aritmetica
ADDx op1, op2	op2 <= op2+op1	L, W, B	Aritmetica
SUBx op1, op2	op2 <= op2-op1	L, W, B	Aritmetica
ADCx op1, op2	op2 <= op2 + op1 + CF	L, W, B	Aritmetica
SBBx op1, op2	op2 <= op2 - op1 - CF	L, W, B	Aritmetica
INCx op1	op1 += 1	L, W, B	Aritmetica
DECx	op1 -= 1	L, W, B	Aritmetica
NEGx	op1 = - op1	L, W, B	Aritmetica
IMUL op1, op2	op2 <= op2 * op1	op2: %	Aritmetica
IMUL inm, op1, op2	op2 <= op1 * inm	inm: $,	Multiplicacion
IMULL op1	%ext <= op1 * %eax	op1: @M o %	Multiplicacion Enteros
MULL op1	%ext <= op1 * %eax	op1: @M o %	Multiplicacion Naturales
CLTD	%ext : ExtSign(%eax)		Extension Signo
IDIVL op1	%eax <= %ext / op1, %edx <= %ext % op1	op1: @M o %	Division Enteros

INSTRU	Desc	Complementos	
DIVL op1	%eax <= %ext / op1, %edx <= %ext % op1	op1: @M o %	Division Naturales
ANDx op1,op2	op2 <= op1 & op1	L, W, B	Lógicas
ORx op1, op2	op2 <= op2 | op1	L, W, B	Lógicas
XORx op1, op2	op2 <= op2 ^ op1	L, W, B	Lógicas
NOTx op1	op1 <= ! op1	L, W, B; k: $	Lógicas
SALx k, op1	op1 <= op1 << k	L, W, B; k: $	Lógicas
SHLx k, op1	op1 <= op1 << k	L, W, B; k: $	Lógicas
SARx k, op1	op1 <= op1 >> k	L, W, B; k: $	Lógicas
SHRx k, op1	op1 <= op1 >> k	L, W, B; k: $	Lógicas
CMPx op1, op2	op2 - op1	L, W, B, mod. flags	Lógicas
TESTx op1, op2	op2&op1	L, W, B i mod. flags	op1 == op2 ?
JMP etiq	%eip <= @etiq	&etiq	Salta Incondicional
JMP op	%eip <= op	op es @	Salta Incondicional
Jcc etiq	%eip <= etiq (if)	cc: E, NE, G, GE, L, LE	Salta condicional Enteros
Jcc etiq	%eip <= etiq (if)	cc: A, AE, B, BE	Salta condicional Naturales
Jcc etiq	%eip <= etiq (if)	cc: Z, NZ, C, NC, O	Salta condicional flags
CALL etiq	%esp <= %esp - 4; M[%esp]<=EIP; %eip<= &etiq	Guarda @ret i PC = & etiq	Llamar f(x)
CALL op	%esp <= %esp - 4; M[%esp]<=EIP; %eip<= &etiq	Guarda @ret i PC = & op	Llamar f(x)
RET	%eip<= M[%esp]; %esp <=%esp+4		Retorna

%ext = %edx : %eax

Tabla de Flags	
JE	Jump equal
JNE	Jump not equal
JS	Jump Negative
JNS	Jump not Negative
JG	Jump Greater (signed)
JGE	Jump Greater or equal (signed)
JL	Jump Less (signed)
JLE	Jump Less or equal (signed)
JA	Jump Greater (not signed)
JAE	Jump Greater or equal (not signed)
JB	Jump Less (not signed)
JBE	Jump Less or equal (not signed)

Traduccion de Sentencias C a Ensamblador

- IF-THEN-ELSE
- DO-WHILE
- WHILE
- FOR
- SWITCH(vector de punteros a los case)

Tipos de datos Estructurados

- Vectores: "Seq de datos iguales almacenados en memoria".

  v[i] -> @v + i * sizeof(tipo).

- Matrices: Vectores almacenado por filas.

  A i, j : @A + (i* NumCols + j) * sizeof(tipo)

- Matrices 3d: Se almacenan tmb en posiciones consecutivas.

  3d i, j, k: @3d + (i * Numero_cara2 * Numero_cara3 + j * Numero_cara3 + k) * sizeof(tipo)

A la hora de acceder a una matriz de 50 por 80, se ejecutan 32267 instruciones, hace falta mejorarlo. Seguira en la classe 02.