Práctica 1 - Introducción a la arquitectura Intel 64

Organización del Computador 2

1er Cuatrimestre 2017

1. Instrucciones básicas y modos de direccionamiento

Notas:
Se definen los tipos de datos `superlong` y `unsignedsuperlong` como:

```
typedef struct superlong_t {
        long x1;
        long x2;
} superlong;

typedef struct unsignedsuperlong_t {
        unsigned long x1;
        unsigned long x2;
} unsignedsuperlong;
```

Ejercicio 1

¿Cuál forma de almacenamiento utilizan los procesadores `Intel x86-64`: Little-endian o Big-endian?

Ejercicio 2

¿Cuáles son los registros de la arquitectura `Intel 64` visibles para el programador? ¿Qué tamaño tienen? Indique para cada uno si tienen alguna función específica.

Ejercicio 3

En una instrucción con dos operandos, ¿cuál es el fuente y cuál el destino?

Ejercicio 4

¿En qué registro se almacenan los *flags* del procesador en la arquitectura `Intel 64`?

Ejercicio 5

Determine si son correctas las siguientes instrucciones. En caso afirmativo, indicar qué modo de direccionamiento se está utilizando. En caso negativo, justificar.

a) `MOV RAX, 4`
b) `MOV 4, RAX`
c) `MOV RAX, RBX`
d) `MOV RAX, EBX`
e) `MOV RAX, RBX+2`
f) `MOV RAX, [variable]`
g) `MOV [RAX], 4`
h) `MOV [RAX], EAX`
i) `MOV [EAX], EAX`
j) `MOV [RAX],[RCX]`
k) `MOV [variable], RAX`
l) `MOV 4, [RAX]`
m) `MOV [variable_1], [variable_2]`
n) `MOV BYTE [RAX], 4`
ñ) `MOV DWORD [RAX], 4`
o) `MOV RBX, [RAX+2]`
p) `MOV RBX, [RAX+RCX]`
q) `MOV RBX, [RAX+RCX*2+3]`
r) `MOV RBX, [RAX+RCX*3+2]`
s) `MOV RBX, [[RAX]]`
t) `MOV RBX, [RAX]+2`

Ejercicio 6

¿Cuáles tamaños de operador se encuentran disponibles en la arquitectura `Intel 64`?

Ejercicio 7

¿Cuántos bytes ocupan los siguientes tipos de datos de C? Especificar en 32 y 64 bits.

a) `char`
b) `short`
c) `int`
d) `long`
e) `long long`
f) `superlong`
g) `float`
h) `double`

Realice un programa en C que muestre los valores pedidos.

Ejercicio 8

En Linux, las llamadas al sistema operativo se realizan por la interrupción 0x80. El número de *syscall* elegida se pasa en `RAX` y los primeros parámetros se pasan por registros, en orden: `RBX`, `RCX`, `RDX`, `RSI`, `RDI`. Cada *syscall* tiene un número único. Por ejemplo, para imprimir en pantalla se utiliza la *syscall* write, `RAX` = 4.

a) Utilizando dicha *syscall* escriba un programa en lenguaje ensamblador que imprima por pantalla "`Hola Mundo!`".
b) ¿Cuáles son las distintas secciones que puede tener un programa en lenguaje ensamblador? ¿Cuáles están presentes en este programa?
c) ¿Qué son las pseudoinstrucciones? ¿Cuáles son las pseudoinstrucciones del `nasm`? ¿Cuáles están presentes en este programa?

Ejercicio 9

Escriba en lenguaje ensamblador un programa que imprima por pantalla 20 veces "`Hola Mundo!`".

2. Operaciones Lógicas y Aritméticas

Ejercicio 10

Escriba un programa en lenguaje ensamblador y verifique mediante el *debugger* la diferencia en el comportamiento (*flags* y resultados) entre las siguientes instrucciones:

a) `INC`/`ADD` y `DEC`/`SUB`.
b) `NOT` y `NEG`.
c) `CMP`/`SUB` y `TEST`/`AND`.

Ejercicio 11

a) Suponer que el registro de *RFLAGS* se encuentra con `OF=1`, pero nuestro programa necesita que este *Flag*esté en 0. Escriba una secuencia corta de instrucciones que cambie el valor de `OF` al deseado.
b) Escriba una secuencia corta de instrucciones que tome el dato almacenado en el registro `DL`, ponga en 0 los 3 bits más significativos y almacene el resultado en `BL`.
c) Escriba una secuencia corta de instrucciones que tome el dato almacenado en el registro `RDI`, ponga en 1 los 5 bits menos significativos y almacene el resultado en `RSI`.
d) Escriba una secuencia corta de instrucciones que tome el dato almacenado en `AX`, ponga en 1 los 4 bits menos significativos, ponga en cero los 3 bits más significativos, invierta los bits 7, 8 y 9; y almacene el resultado en `BX`.
e) Escriba una secuencia corta de instrucciones que tome el dato almacenado en el flag de carry del registro `RFLAGS`, lo compare por igual (XOR) con el bit menos significativo del registro `BL` y almacene el resultado en el flag de carry nuevamente.

Ejercicio 12

Escriba una secuencia de instrucciones que:

a) Sume un número con signo de 8 bits almacenado en `AL` con otro también con signo de 16 bits almacenado en `BX`; y almacene el resultado en `AX`
b) Sume un número con signo de 16 bits almacenado en `AX` con otro también con signo de 32 bits almacenado en `EBX`; y almacene el resultado en `EAX`.
c) Sume un número con signo de 32 bits almacenado en `EAX` con otro también con signo de 64 bits, almacenado en memoria apuntado por `RSI`; y almacene el resultado en esa misma posición de memoria.
d) Sume un número con signo de 128 bits almacenado en `RDX:RAX` con otro también con signo de 128 bits almacenado en memoria, apuntado por `RSI`; y almacene el resultado en esa misma posición de memoria.
e) Multiplique un número con signo de 16 bits almacenado en `AX` con otro también con signo de 32 bits almacenado en `EBX` y almacene el resultado en `EDX:EAX`.
f) Divida un número con signo de 16 bits almacenado en `AX` por otro también con signo y de 16 bits, almacenado en `BX`; y almacene por un lado el cociente en `AX`, y por el otro el resto en `DX`.

Ejercicio 13

Escriba una secuencia de instrucciones que:

a) Multiplique un número de 192 bits almacenado en `RDX:RBX:RAX` por 2.
b) Multiplique un número de 192 bits almacenado en `RDX:RBX:RAX` por 2^n; donde n es un entero sin signo menor que 95, almacenado en `RCX`.
c) Divida un número de 192 bits sin signo almacenado en `RDX:RBX:RAX` por 2.
d) Divida un número de 192 bits con signo almacenado en `RDX:RBX:RAX` por 2^n; donde n es un entero sin signo menor que 95, almacenado en `RCX`.

Ejercicio 14

Suponer que recibimos en AX el estado de la máscara de interrupción de los controladores de interrupciones (PIC 1 y 2). Esta máscara indica con un 1 en el bit i que la interrupción IRQ_i está deshabilitada.

Escriba un programa en lenguaje ensamblador que imprima en pantalla cuáles son las interrupciones habilitadas. Utilice dos estrategias distintas: máscaras y *shifts*.

3. Stack e interacción C-Assembler

Ejercicio 15

Explique qué es y para qué se usa el *stack* (la pila).

Ejercicio 16

Explique qué diferencia existe entre ejecutar la instrucción `PUSH RBX` y el siguiente conjunto de instrucciones:

a)
```
SUB RSP, 8
MOV [RSP], RBX
```

b)
```
MOV [RSP], RBX
DEC RSP
DEC RSP
DEC RSP
DEC RSP
DEC RSP
DEC RSP
DEC RSP
DEC RSP
```

Ejercicio 17

Suponga un programa escrito en C (miPrograma.c), que consiste en una rutina principal que llama a una función auxiliar, con tres parámetros de tipo `long`.

a) Explique cómo se introducen los parámetros según la convención C en 32 bits y en 64 bits.
b) Explique cómo se realiza el retorno a la rutina principal.
c) ¿Cuáles son los registros que se deben preservar en la convención C para 32 y 64 bits?
d) ¿Cómo se devuelven los resultados en la convención C? (por valor y por referencia) ¿Qué queda en la pila en ambos casos?

Ejercicio 18

Muestre el contenido de la pila y los registros cuando se llama a las siguientes funciones en la convención C en 32 y 64 bits:

a) `int func_a(long a, long b);`
b) `int func_b(long a, long* b);`
c) `int* func_c(long a, long b);`
d) `void func_d(short int a, long b);`
e) `void func_e(long int a, char b, int c);`
f) `void func_f(superlong int a, char* b);`
g) `void func_g(char b, superlong int a, int c);`

Ejercicio 19

Escriba una función en lenguaje ensamblador que imprima por pantalla `''Hola Mundo!''` llamando a la función de C `printf`.

Ejercicio 20

Dado el siguiente programa en lenguaje C:

```
long global_no_ini;
long global_ini = 31416;
const long global_ini_const = 14142;

int main(int argc, char* argv[]){
    long local_no_ini;
    long local_ini = 27182;
    return 0;
}
```

a) ¿Dónde deberían estar definidas cada una de las variables y constantes del programa? (`.data`, `.rodata`, en la pila, etc).
b) Compile el programa con la opción `-S` y observe el archivo `.s` obtenido.
¿Se cumplen sus predicciones?

Ejercicio 21

Escriba una función en lenguaje ensamblador Intel 64 que:

a) Sume dos números de 128 bits con signo, cuyo prototipo sea:
`superlong suma(superlong a, superlong b);`
¿Cambia el código en lenguaje ensamblador si el prototipo de la suma es `unsignedsuperlong suma(unsignedsuperlong a, unsignedsuperlong b);` ?
b) Niegue un número de 128 bits con signo pasado como parámetro, cuyo prototipo sea: `superlong negar(superlong a);`
c) Compare dos números de 128 bits con signo y devuelva uno si el primero es mayor al segundo y cero en caso contrario, cuyo prototipo sea:
`unsigned int esMayor(superlong a, superlong b);`
d) Multiplique dos números de 128 bits sin signo, cuyo prototipo sea:
`void* producto(unsignedsuperlong a, unsignedsuperlong b);`
e) Multiplique dos números de 128 bits con signo, cuyo prototipo sea:
`void* producto(superlong a, superlong b);`
f) Divida un número de 192 bits con signo por uno de 64 bits, cuyo prototipo sea:
`superlong división(void* dividendo, long divisor);`

Nota: para los ejercicios que requieran pedir memoria puede usarse la función de C `malloc` cuyo prototipo es `void* malloc(unsigned cantBytes);` que pide `cantBytes` bytes de memoria y devuelve un puntero apuntando al espacio reservado, o 0 en caso de error.

Ejercicio 22

Escriba una función en lenguaje ensamblador que compute x^y, cuyo prototipo sea:
`superlong power(long x, unsigned long y);`.
Se puede asumir que el resultado de la operación entra en 128 bits.

Ejercicio 23

Escriba una función en lenguaje ensamblador que calcule la cantidad de factores primos de un número de 64 bits sin signo pasado como parámetro.

Se pide respetar el siguiente código C:

```
unsigned cantFactoresPrimos(unsigned long n){
    int k = 1, d = 2;
    while(d*d <= n) {
        if (n mod d == 0){
            k++;
            n = n/d;
        }
        else d++;
    }
    return k;
}
```

Ayuda: utilizar que $d_n \times d_n = (d_{n-1} \times d_{n-1}) + 2d_{n-1} + 1$, donde $d_n = d_{n-1} + 1$.