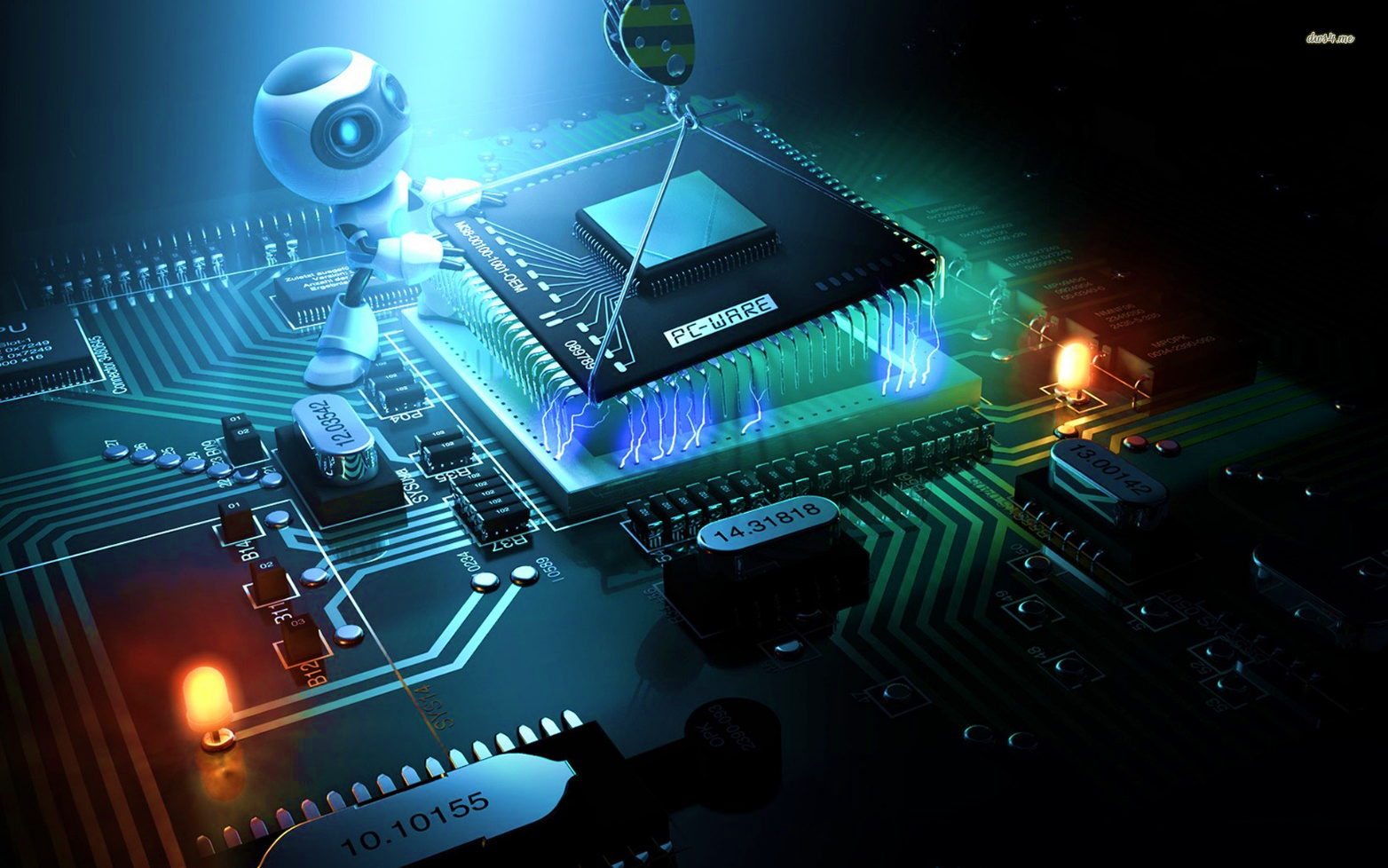
Ejercicios Semana 4-5

ESTRUCTURA DE COMPUTADORES



bRYAN mORENO PICAMÁN

Contenido

[Descripción 2](#_Toc468360893)

[3.6 Solución 3](#_Toc468360894)

[3.7 Solución 3](#_Toc468360895)

[3.8 Solución 3](#_Toc468360896)

[3.9 Solución 3](#_Toc468360897)

[3.10 Solución 3](#_Toc468360898)

[3.11 Solución 3](#_Toc468360899)

[3.12 Solución 4](#_Toc468360900)

[3.13 Solución 4](#_Toc468360901)

[3.14 Solución 4](#_Toc468360902)

[3.15 Solución 4](#_Toc468360903)

[3.16 Solución 4](#_Toc468360904)

[3.17 Solución 5](#_Toc468360905)

[3.18 Solución 5](#_Toc468360906)

[3.19 Solución 5](#_Toc468360907)

[3.20 Solución 6](#_Toc468360908)

[3.21 Solución 7](#_Toc468360909)

[3.22 Solución 8](#_Toc468360910)

[3.23 Solución 8](#_Toc468360911)

[3.24 Solución 9](#_Toc468360912)

[3.25 Solución 9](#_Toc468360913)

[3.26 Solución 9](#_Toc468360914)

[3.27 Solución 9](#_Toc468360915)

[3.30 Solución 10](#_Toc468360916)

[3.31 Solución 10](#_Toc468360917)

[3.32 Solución 10](#_Toc468360918)

[3.33 Solución 10](#_Toc468360919)

[3.34 Solución 11](#_Toc468360920)

# Descripción

Cap.3 CS: APP (Bryant/O’Hallaron)

Probl. 3.6-3.27 pp. 212-16, 218, 222-23, 226, 229-30, 232-33, 235-36, 239-40, 243, 246

Probl. 3.28-3.34 pp. 251-52, 257-58, 262, 265-66

# 3.6 Solución

|  |  |
| --- | --- |
| Instrucción | Resultado |
| leal 6(%eax), %edx | 6+x |
| leal (%eax,%ecx), %edx | X+y |
| leal (%eax,%ecx,4), %edx | X+4y |
| leal 7(%eax,%eax,8), %edx | 7+9x |
| leal 0xA(,%ecx,4), %edx | 10+4y |
| leal 9(%eax,%ecx,2), %edx | 9+x+2y |

# 3.7 Solución

|  |  |  |
| --- | --- | --- |
| Instrucción | Destino | Valor |
| addl %ecx, (%eax) | 0x100 | 0x100 |
| subl %edx, 4(%eax) | 0x104 | 0xA8 |
| imull $16, (%eax, %edx,4) | 0x10C | 0x110 |
| incl 8(%eax) | 0x108 | 0x14 |
| decl %ecx | %ecx | 0x0 |
| subl %edx, %eax | %eax | 0xFD |

# 3.8 Solución

movl 8(%ebp), %eax  
 sall $2, %eax   
 movl 12(%ebp), %ecx  
 sarl %cl, %eax

# 3.9 Solución

int t1 = x ^y;  
 int t2 = t1 >> 3  
 int t3 = ~t2  
 int t4 = t3 – z;

# 3.10 Solución

a) x^x = 0, lo que hace es poner el registro %edx a 0. Corresponde con x = 0 en C.

b) La forma directa sería movb $0, %edx.

c) xor requiere 2 bytes, 1 para el código de operación y otro rm para indicar el registro, movl requiere 5 bytes

# 3.11 Solución

movl 8(%ebp), %eax

movl $0, %edx

divl 12(%ebp)

movl %eax, 4(%esp)

movl %edx, (%esp)

# 3.12 Solución

a) El programa hace operaciones de multiprecisión en 64bits. También hace operaciones de multiplicación de 64bits haciendo uso de aritmética de “unsigned”

# 3.13 Solución

a) El sufijo l indica operaciones de 32bits, data\_t debe ser int, < indica comparación

b) W indica operaciones de 16bits, data\_t debe ser short y la comparación >=

c) b es para obits, data\_t debe ser char y la comparación es < al igual que en l

d) Al igual que el primero indica operaciones de 32 bits, y se usa !=

# 3.14 Solución

a) data\_t = int, unsigned, pointer TEST: !=

b) data\_t = short TEST: ==

c) data\_t = char TEST: >

d )data\_t = short TEST: >

# 3.15 Solución

a) je tiene como objetivo 0x8048291+0x05

b) jb tiene como objetivo 0x8048359-25

c) Tenemos dirección de salto a 0x8048391

d) Leyendo los bytes al contrario vemos que el offset es 0xfffffffffe0, -32 en decimal, añadiéndolo a 0x80482c4 obtenemos la dirección 0x80482a4

e) El salto indirecto es denotado por la instrucción con código ff 25.

# 3.16 Solución

a)

void go\_cond (int a, int \*p){

if (p==0||a<=0){

goto done;

}

\*p += a;

done:

return;

}

b) Porque la sentencia if tiene dos condiciones a cumplir, p==0 y a <= 0.

# 3.17 Solución

a)

int godiff (int x, int y){  
 int a;  
 if ( x < y)  
 goto true;  
 a = x –y;  
 goto done;  
 true:  
 a = y – x;  
 done:  
 return a;  
}

b) La regla alternativa es más larga y complicada que la que utilizamos con lógica inversa.

# 3.18 Solución

int test(int x, int y){

int val = x ^ y;

if ( x < -3){

if (y > x)

val = x \* y ;

else

val = x + y;

}

else if (x > 2)

val = x - y;

return val;

}

# 3.19 Solución

a) El mayor entero que podemos representar es FFFF que es 65535 en decimal. El máximo valor de n es 8, porque 8! = 40320 y 9! = 362880, por lo que seria mas grande.

b) El mayor entero que podemos representar es FFFF FFFF que en decimal es 4294967295. El máximo valor de n es 12, porque 12! = 479001600 y 13! = 6227020800, por lo que se pasa.

# 3.20 Solución

a)

|  |  |  |
| --- | --- | --- |
| Registro | Variable | Inicializado |
| %eax | x | x |
| %ecx | y | y |
| %edx | n | n |

b)

C:

test-expr = ((n > 0) && (y < n))

body-statement = líneas 3,4 y 5

EMSAMBLADOR:

test-expr =líneas 8 y 10

body-statement = líneas 5, 6 y 7

c)

Argumentos: x en %ebp+8, y en %ebp+12, n en %ebp+16

Registros: x en %eax, y en %ecx y n en %edx

movl 8(%ebp), %eax x en %eax

movl 12(%ebp), %ecx y en %ecx

movl 16(%ebp), %edx n en %edx

.L2:

addl %edx, %eax x += n

imull %edx, %ecx : y\*=n

subl $1, %edx n--

testl %edx, %edx

jle .L5

cmpl %edx, %ecx

jl .L2

.L5:

# 3.21 Solución

a) Vemos que el registro es inicializado a a+b y se incrementa en cada iteración. De la misma forma el valor de a se incrementa de forma que el valor en el registro %edx siempre va a ser igual a a+b

b)

|  |  |  |
| --- | --- | --- |
| Registro | Variable | Inicializado |
| %ecx | a | a |
| %ebx | b | b |
| %eax | result | 1 |
| %edx | a+b | a+b |

c)

a en %ebp+8, b en %ebp+12

a en %ecx, b en %ebx, result en %eax, a+b en %edx

movl 8(%ebp), %ecx a en %ecx

movl 12(%ebp), %ebx b en %ebx

movl $1, %eax result a 1

cmpl %ebx, %ecx a < b

jge .L11 If a >= b, goto L11

leal (%ebx,%ecx), %edx en %edx, a+b

.L12

imull %edx, %eax result\*(a+b)

addl $1, %ecx a++

addl $1, %edx a+b++

cmpl %ecx, %ebx

jg .L12

.L11

d)

int loop\_while\_goto(int a, int b){

int result = 1;

if (a >= b) goto done;

int suma = a+b;

loop:

result \*= suma;

a++;

suma++;

if(b > a) goto loop;

done: return result;

}

# 3.22 Solución

a)

int fun\_a (unsigned x){

int val = 0;

while (x){

val ^= x;

x >>= 1;

}

return val & 0x1;

}

b) Calcula la paridad de x.

# 3.23 Solución

a)

int fun\_b (unsigned x){

int val = 0;

int i;

for( i=0; i < 32 ; i++){

val = (val << 1) | (x & 0x1);

x >>= 1;

}

return val;

}

b) Invierte los bits de x.

# 3.24 Solución

a)

int sum=0;

int i=0

while(i<10){

if(i&1)

continue;

sum+=1;

i++;

}

b)

int sum=0;

int i=0

while(i<10){

if(i&1)

goto update;

sum+=1;

update:

i++;

}

# 3.25 Solución

a) Podemos aplicar la formula directamente Tmp=2(31-16)=30

b) Cuando precedimos mal la función requiere alrededor de 16+30=46 ciclos

# 3.26 Solución

a) El operador es /, se puede ver un ejemplo de división por potencias de 2 .

b)

x en %edx

leal 3(%edx),%eax

testl %edx,%edx

cmovns %edx,%eax

sarl $2,%eax

# 3.27 Solución

int test (int x, int y){

int val = 4\*x;

if (y > 0) {

if (x < y)

val = x - y;

else

val = x ^ y;

}

else if (y < -2)

val =x + y ;

return val;

}

# 3.30 Solución

a) El valor del tope de la pila.

b) Porque no es una llamada a procedimiento, el control sigue el orden de las instrucciones y la dirección de retorno se extrae de la pila.

c) Este fragmento sirve para obtener el valor del contador de programa en un registro.

# 3.31 Solución

Este problema hace referencia a la convención de registros, edi,esi y eba son salva invocado, el procedimiento debe guardarlos antes de alterar sus valores y restaurarlos antes de retornar. Los otros tres registros son salva invocantes, pueden alterarse sin afectar el comportamiento del invocado.

# 3.32 Solución

El prototipo de la función sería:

int fun (short c, char d, int\* p, int x);

# 3.33 Solución

a) Se ajusta a 0x80003C.

b) Se ajusta a 0x800014.

c) x está almacenada en la dirección 0x800038 y la variable y en la dirección 0x800034.

d)

|  |  |  |
| --- | --- | --- |
| 0x80003C | 0x800060 | <-%ebp |
| 0x800038 | 0x46 | X |
| 0x800034 | 0x53 | Y |
| 0x800030 |  |  |
| 0x80002C |  |  |
| 0x800028 |  |  |
| 0x800024 |  |  |
| 0x800020 |  |  |
| 0x80001C | 0x800038 |  |
| 0x800018 | 0x800034 |  |
| 0x800014 | 0x300070 | <- %esp |

e)Desde 0x800020 hasta 0x800030.

# 3.34 Solución

a) Almacena el valor de x.

b)

int rfun (unsigned x){

if (x == 0){

return 0;

}

unsigned nx =x>>1 ;

int rv = rfun(nx);

return (x & 0x1) + rv;

}

c) Calcula la suma de los bits del argumento x.