Juan Carlos Ruiz García 26510953-A Grupo C2

Problem 3.1

Operand	Value
%eax	0x100
0x104	0xAB
$0x108	0x108
(%eax)	0xFF
4(%eax)	0xAB
9(%eax,%edx)	(0x10C) → 0x11
260(%ecx,%edx)	(0x108) → 0x13 — {260 + 1 + 3 = 264$_{10}$ ⇒ 108$_{16}$}
0xFC(,%ecx,4)	(0x100) → 0xFF — {0xFC + 0x1 * 4}
(%eax,%edx,4)	(0x10C) → 0x11 — {0x100 + 0x3 * 4}

Problem 3.2

```
movl  %eax, (%esp)
movw  (%eax), %dx
movb  $0xFF, %bl
movb  (%esp,%edx,4), %dh
pushl $0xFF
movw  %dx, (%eax)
popl  %edi
```

Debemos ver en el caso de la instrucción mov, hacia donde se mueve el dato y el tamaño de dicho destino. Así sabremos si usar movw, movl ...

Problem 3.3

1 movb $0xF, (%bl) → No se puede usar %bl como dirección de un registro.

2 movl %ax, (%esp) → movl es para 32 bits y %ax es de 16 bits

3 movw (%eax), 4(%esp) → No pueden ser dos referencias a memoria tanto el destino como el origen.

4 movb %ah, %sh → El registro %sh no existe.

5 movl %eax, $0x123 → No se puede usar una constante como destino

6 movl %eax, %dx → El destino es un registro de 16 bits y el origen uno de 32 bits.

7 movb %si, 8(%ebp) → %si es un registro de 32 bits y movb una instrucción para 8 bits.

Problem 3.4

src_t	dest_t	Instrucción
int	int	movl %eax, (%edx)
char	int	movsbl %al, (%edx)
char	unsigned	movsbl %al, (%edx)
unsigned char	int	movzbl %al, (%edx)
int	char	movb %al, (%edx)
unsigned	unsigned char	movb %al, (%edx)
unsigned	int	movl %ecx, (%edx)

Problem 3.5

```
void decode1(int *xp, int *yp, int *zp) {
    int aux1, aux2, aux3;
    aux1 = *yp; aux2 = *zp; aux3 = *xp;
    *yp = aux1;
    *zp = aux2;
    *xp = aux3;
}
```

Problem 3.7

Instruction	Destination	Value
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

Problem 3.8

```
1  movl   8(%ebp), %eax
2  sall   $2, %eax
3  movl   12(%ebp), %ecx
4  subl   %ecx, %eax
```

Problem 3.9

```
int arith (int x,
           int y,
           int z)
{
    int t1 = x^y;
    int t2 = t1 >> 3;
    int t3 = ~t2;
    int t4 = t3 - z;
    return t4;
}
```

Problem 3.30

(A.) %eax toma el valor de la dirección de memoria del tope de la pila

(B.) No es necesaria la instrucción ret ya que tras la llamada a next:, se continuará por donde debía.

(C.) Es la única forma en IA32 de obtener el valor del contador de programa en un registro.

Problem 3.47

src_t	dest_t	Instruction	S	D
long	long	movq	%rdi	%rax
int	long	movslq	%edi	%rax
char	long	movsbq	%dil	%rax
unsigned int	unsigned long	movl	%edi	%eax
unsigned char	unsigned long	movzbl	%dil	%eax
long	int	movl	%edi	%eax
unsigned long	unsigned	movl	%edi	%eax

Problem 3.6

Instruction	Result
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	9x+7
leal 0xA(, %ecx, 4), %edx	4y+10
leal 9(%eax, %ecx, 2), %edx	x+2y+9

Problem 3.10

(A) El efecto el realizar una operación XOR sobre el mismo número es la obtención del valor 0. Mostraré un ejemplo.

$x = 4$; $(4_{10} = 100_{2})$

$x = 0$; →

$$\begin{array}{r} 100 \\ 100 \\ \hline 000_{2} = 0_{10} \end{array}$$

(B.) movl $0, %edx

(C.) Con xorl se requieren de tan solo 2 bytes para realizar la operación, mientras que con movl se requieren de 5 bytes.

Problem 3.11

```
movl 8(%ebp), %eax
movl $0, %edx
divl 12(%ebp)
movl %eax, 4(%esp)
movl %edx, (%esp)
```

Problem 3.13

(A.) data_t → int
COMP → <

El sufijo 'l' y los registros usados denotan que es un tipo de dato de 32 bits y setl significa "Set byte if less" por lo que la operación debe ser '<';

(B.) data_t → short int / unsigned short int
COMP → >=

El sufijo "w" denota que ocupa 16 bits y los registros usados son de 16 bits por lo que el tipo de dato a de ser del mismo tamaño.
setge → "Set byte if greater or equal" → >=

(C.) data_t → char / unsigned char
COMP → <

El sufijo "b" indica que es de 8 bits al igual que los registros usados.
setb → "Set byte if bellow" → <

(D.) data_t → int / unsigned int / long / unsigned long
COMP → !=

El sufijo "l" indica que es de 32 bits al igual que los registros usados.
setne → "Set byte if not equal" → !=

Problem 3.14

(A.) data_t → int / unsigned int / long / unsigned long
TEST → !=

(B.) data_t → short int / unsigned short int
TEST → ==

(C.) data_t → char / unsigned char
TEST → >

(D.) data_t → short int / unsigned short int
TEST → >

Problem 3.15

(A.)
```
je 8048296
call 80482b4
```

(B.)
```
jb 8048410
movb $0x1, 0x804a010
```

(C.)
```
804837D :
804837F :
```

(D.)
```
jmp 80482a4
nop
```

(e.)

Problem 3.18

```
int test (int x, int y) {
    int val =
    if ( x < -3 ) {
        if ( y < x )
            val = x * y;
        else
            val = x + y;
    } else if ( x > 2 )
        val = x - y;
    return val;
}
```

Problem 3.19

(A.) Usando la siguiente formula se puede comprobar cuando se produce desbordamiento.

$$\frac{n!}{n} = (n-1)! \quad \Rightarrow \quad \frac{13!}{13} \neq (13-1)!$$

$$1486219500 \neq 479001600$$

El 13! produce desbordamiento en un int

(B.) Realizando la misma operación para un long long int el número que produce desbordamiento es el 20!

Problem 3.20

(A.)

Register	Variable	Initially
%eax	x	x
%ecx	y	y
%edx	n	n

(B.) C code

Test_expre → Linea 6

Body_statement → Lineas 3 a 5

Assembly code

Test_expre → Lineas 8 a 11

Body_statement → 5 a 7

(C.)

```
movl  8(%ebp), %eax   Guarda x
movl  12(%ebp), %ecx  Guarda y
movl  16(%ebp), %edx  Guarda n
.L2:  bucle
addl  %edx, %eax      x+=n
imull %edx, %ecx      y*=n
subl  $1, %edx        n--;
testl %edx, %edx      test n
jle   .L5             if(n<=0) goto .L5
cmpl  %edx, %ecx      comparar y con n
jl    .L2             if(y<n) goto .L2 (bucle)
.L5 fin
```

Problem 3.21

(A.) El registro %edx es inicializado como a+b y se va incrementando en cada iteración por a+b.

(B.)

Register	Variable	Initiallity
%ecx	a	a
%ebx	b	b
%edx	result	1
%edx	apb	a+b

(C.)

```
movl  8(%ebp), %ecx      Cargamos a
movl  12(%ebp), %ebx     Cargamos b
movl  $1, %eax           Ponemos result = 1
cmpl  %ebx, %ecx         Comparamos a con b
jge   .L11               Si >= salta a .L11
leal  (%ebx, %ecx), %edx apb = a+b
movl  $1, %eax           result = 1
.L12:  bucle
 imull  %edx, %eax       result *= apb
 addl   $1, %ecx         a++
 addl   $1, %edx         apb++
 cmpl   %ecx, %ebx       Compara a con b
 jg     .L12             Si > salta a .L12
.L11 fin
Return resultado
```

(D.)

```
int loop_while_goto(int a, int b)
{
    int result = 1;
    if(a >= b)
        goto fin;
    int apb = a+b;
  buele:
    result *= apb;
    a++; apb++;
    if(b > a)
        goto buele;
  done:
    return result;
}
```

Problem 3.22

(A.)

```
int fun_a(unsigned x) {
    int val = 0;
    while(x) {
        val ^= x;
        x >>= 1;
    }
    return val & 0x1;
}
```

(B.) Esta función se encarga de comprobar la paridad de un número. Si es par devuelve 0 y si es impar devuelve 1.

Problem 3.23

(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.) Esta función da la vuelta bit a bit de x. Es decir si obtiene el 6 en x (110) devuelve el 3 en val (011).

Problem 3.24

(A.)

```
int sum = 0;
int i = 0;
while ( i < 10) {
    if (i & 1)
        continue;
    sum += i;
    i++;
}
```

→ Esto provocaría un bucle infinito cuando se entre en el if debido a que continue no haría el i++;

(B.)

```
int sum = 0;
int i = 0;
while (i < 10) {
    if (i & 1)
        goto incrementar;
    sum += i;
incrementar:
    i++;
}
```

Problem 3.25

(A.) $T_{MP} = 2(T_{MEM} - T_{MP})$

$$T_{MP} = 2(31 - 16) = \boxed{30}$$

(B.) La función requiere de $16 + 30 = 46$ ciclos.

Problem 3.26

(A.) `# define OP /`

(B.)
```
leal   3(%edx), %eax   value = x + 3
testl  %edx, %edx      test x
cmovns %edx, %eax      Si >= 0, value = x
sarl   $2, %eax        Return value >> 2
```

Problem 3.27

```
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;
}
```

Problem 3.31

No es una inconsistencia. Por convención, los registros %edx, %ecx, %ecx son salva-invocante. Esto quiere decir que la función los puede usar sin tener que preservar su valor.
Sin embargo los registros %ebx, %edi, %esi son salva-invocados. Es decir, la función debe ~~preservar~~ salvar su valor antes de usarlos y restaurar dicho valor antes del ret de la función.

Problem 3.32

```
int fun (int a, int d, int *p, int x)
```

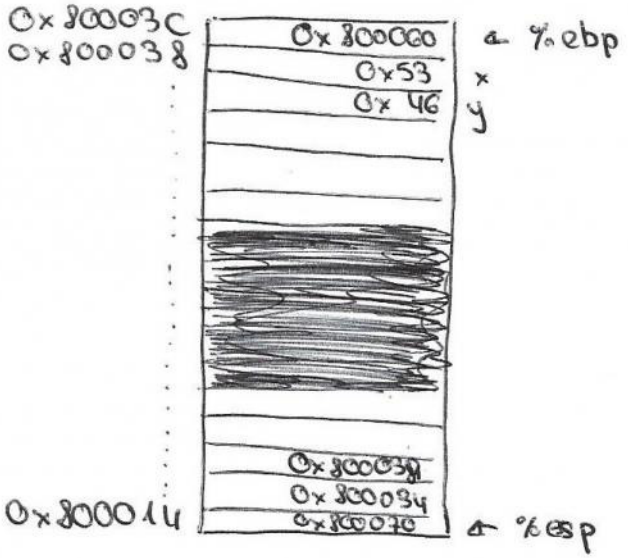
Problem 3.33

(A.) %esp = 0x800040 // línea 1
pushl %ebp // línea 2 (decremente en 4)
movl %esp, %ebp // %ebp = 0x80003C

(B.) Decrementa el puntero de pila 40, obteniendo 0x800014.

(C.) x → 0x800038
y → 0x800034

(D.)



(E.) Desde la dirección 0x800020 hasta los 0x800030 no están usados.

Problem 3.34

(A.) El registro %ebx guardará el valor de x después de la ejecución de rfun.

(B.)

```
int rfun(unsigned x) {
    if (x == 0)
        return 0;
    unsigned nx = x >> 1;
    int rv = rfun(nx);
    return (x & 0x1) + rv;
}
```

(C.) Suma el bit menos significativo de un número de forma recursiva.

Problem 3.35

Array	Element Size	Total Size	Start Ad.	Elementi
				$x_S + 2i$
S	2	14	x_S	[illegible]
T	4	12	x_T	$x_T + 4i$
U	4	24	x_U	$x_U + 4i$
V	12	96	x_V	$x_V + 12i$
W	4	16	x_W	$x_W + 4i$

Problem 3.36

Expresión	Type	Value	Assembly Code
S+1	short *	$x_S + 2$	leal 2(%edx), %eax
S[3]	short	M[x_S+6]	movw 6(%edx), %ax
&S[i]	short *	$x_S + 2i$	leal (%edx, %ecx, 2), %ecx
S[4*i+1]	short	M[x_S+8i+2]	movw 2(%edx, %ecx, 8), %cx
S+i-5	short *	$x_S + 2i - 10$	leal -10(%edx, %ecx, 2), %ecx

Problem 3.37

M = 5 N = 7

Problem 3.41

(A.)

i	c	j	d	Total	Alignment
0	4	8	12	16	4

(B.)

i	c	j	d	Total	Alignment
0	4	5	8	12	4

(C.)

w	c	Total	Alignment
0	6	10	2

(D.)

w	c	Total	Alignment
0	8	20	4

(E.)

a	p	Total	Alignment
0	32	36	4