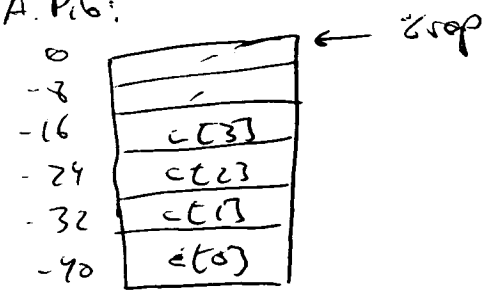
3.51)

A. Pila:



B:
```
movq    $2, -40(%rsp)
movq    $3, -32(%rsp)
movq    $5, -24(%rsp)
movq    $7, -16(%rsp)
cmpl    $3, %edi
movq    -40(%rsp, %rdi, 8), %rcx
ret
```

C: La función nunca cambia el puntero de pila. Almacena todos sus valores locales en la región más allá del puntero de pila

3.52)

A. %rbx se usa para el parámetro x

B. %rbx se debe guardar en pila, se usa push y pop para guardar y restaurar el registro

E. rfact:
```
pushq   %rbx
movq    %rdi, %rbx
movl    $1, %eax
testq   %rdi, %rdi
jle     .L11
leaq    -1(%rdi), %rdi
call    rfact
imulq   %rbx, %rcx
.L11:   popq    %rbx
ret
```

D. En lugar de decrementar y mantener el puntero de pila el código puede usar pushq y popq tanto para modificar como para restaurar

3.46) A. Suponiendo base 16'36 en 2016, 256 Tb representa
aumentar 1608×10^4 en 25 años, desde 2035

B. 16 eb. es aumentar $1054,1 \times 10^9$ más de 16,3 Gb, esto haría
tener 53 años, lo que nos da 2063

C. El aumento del presupuesto es un factor de 10 cortes/6 años.
Se cumplen los objetivos en 2029 y 2057

3.47)

src_t	dest_t	inst.	S	D	Explicación
long	long	movl	%rd	%rcx	
int	long	movslq	%edi	%rcx	ext. signo
char	long	movslq	%dil	%rcx	ext. signo
unsigned int	unsigned long	movl	%edi	%rax	ϕ c 64b
unsigned char	unsigned long	movzbq	%dil	%rcx	ϕ c 64b
unsigned char	unsigned long	movzbq	%dil	%rcx	ϕ c 64b
long	int	movslq	%edi	%eax	signo c 64b
long	int	movl	%edi	%ecx	ϕ c 64b
unsigned long	unsigned	movl	%edi	%eax	ϕ c 64b

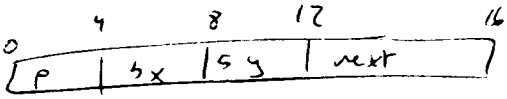
3.50) 1º) addl obtiene un entero de 32b desde la dirección del 3º arg
y le añade a la versión de 32b del 1º arg.

Luego t es el 3º arg. y x es el 1º. T es un puntero a entero y podría
ser 32 o 64b

2º) movslq extiende la suma a un entero largo. Puede inferir que t debe
ser un puntero a un entero con signo

3º) ADD suma el valor de signo extendido de la suma anterior a la
dirección indicada por el 2º reg. A partir de eso podemos
inferir que q es el 2º argumento y es puntero a un long int.

3.39) 1°)



2°) 16 bytes

3°)
```
movl  8(%ebp), %ecx
movl  8(%ecx), %edx
movl  %edx, 4(%ecx)
leal  4(%ecx), %edx
movl  %edx, (%eax)
movl  %ecx, 12(%ecx)
```

Se puede generar:

```
void sp_ini(struct prop *sp){
    sp->s.x = sp->s.y;
    sp->p = &(sp->s.x);
    sp->next = sp;
}
```

3.42) A:

Campo	a	b	c	d	e	f	g	h
tamaño	4	2	8	1	4	1	8	4
Despl.	0	4	8	16	20	24	32	40

B: Total 48 bytes de longitud, el final son 46 para satisfacer alineación de 8

C) Se pueden ordenar los elementos en orden decreciente.

```
struct S
    double c;
    long long j;
    float e;
    char *c;
    void *k;
    short b;
    char d;
    char f;
};
```

para 32b:

Campo	c	j	e	c	k	b	d	f
Tam	8	8	4	4	4	2	1	1
Desp	0	8	16	20	24	28	30	31

3.37) Código ensamblador:

```
movl  8(%ebp), %ecx
movl  12(%ebp), %edx
leal  0(,%ecx,8), %eax
subl  %ecx, %eax
addl  %edx, %eax
leal  (%edx,%edx,4), %edx
addl  %ecx, %edx
movl  mat1(,%eax,4), %eax
addl  mat2(,%edx,4), %eax
```

La referencia a la matriz mat1 está en el byte de despl 4 mientras que la matriz mat2 es compensada con el byte 4.
Mat1 tiene 7 col. y mat2 tiene 5, dando M=5 y N=7

3.38)

```
void fix_set_diag_opt(fix_mat A, int val){
    int *Abase = &A[0][0];
    int index = 0;
    do{
        Abase[index] = val;
        index += (N+1)
    }while(index != (N+1)*N);
}
```

La función index le asigna a Abase int* saltando el ancho de la matriz A. Designa una secuencia de enteros de 4 bytes que son elementos de A en orden de filas. Se introduce un index con la diagonal
El puntero debe ser escalado a 4. %eax con índice 4 es reducido por un factor 4. para N=16 para index 4 será de 4, 16(16+1) = 1,0??

3.34]

1°] El registro %ebx contiene el valor x, se puede usar para el resultado

2°]
```
int fun( unsigned x ){
    if(x == 0)
        return 0;

    unsigned ux = x >> 1;
    int r = fun(ux);

    return (x & 0x1) + r;
}
```

3°] Es recursive, calcula la suma de todos, pero el bit menos significativo se cuenta para obtener el resultado

3.35]

Array	tam.	tam. total	Dir. inicial	i
S	2	14	x s	xs + 2i
T	4	12	x r	xr + 4i
U	4	24	x u	xu + 4i
V	12	96	x v	xv + 12i
W	4	16	x w	xw + 4i

3.36]

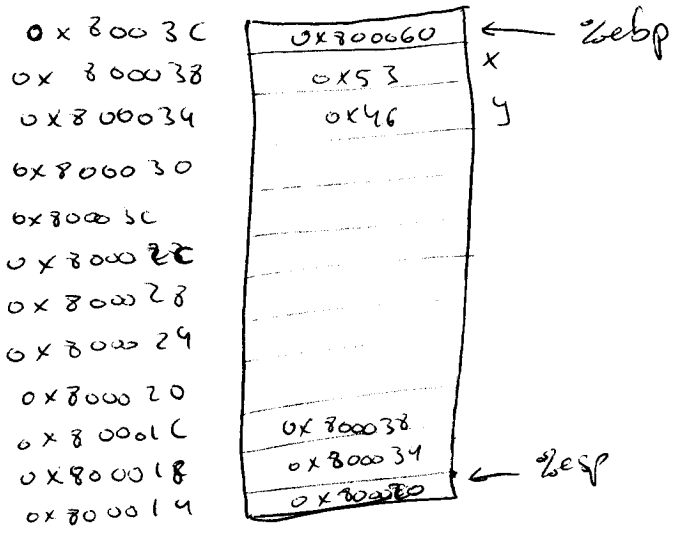
Exp.	Tipo	Val	Ensamblador
$+1	short*	xg + 2	leal 2(%edx), %eax
S[3]	short	M[xg + 6]	movswl (%edx), %eax
&S[i]	short*	xg + 2i	leal (%edx, %ecx, 2), %ecx
S[4*i+1]	short	M[xg + 8i + 2]	movw 2(%edx, %ecx, 8), %eax
&i[5]	short*	xg + 2i - 10	leal -10(%edx, %ecx, 2), %eax

3.33) 1º) %esp tiene valor 0x800040. La inst. pushl en la línea 2 disminuye el puntero de pila por 4, dando 0x80003C que es el nuevo valor de %ebp

2º) La línea 4 disminuye el puntero de pila por 40 dando 0x800014

3º) Las dos inst. leal calculan los argumentos para transmitir scanf, mientras que los movl se almacenan en pila.
Se concluye que la l.5 computa &x y la 7 &y, con valores 0x800038 y 0x800034 respectivamente

4º)



5º) El byte se dirige a través de 0x800020, 0x800035 no son utilizados

3.30) Es otro ej. de ensamblador, call no corresponde con ret. pero no es una llamada de procedimiento

- %eax se establece en la inst. de popl
- No es verdadera llamada de procedimiento ya que el control sigue el mismo orden que las inst. y la dir. de retorno de la pila
- Es la única vía para el IA32 de obtener el valor del contador del programa en un registro entero

3.31) Los registros %edi, %esi, %ebx son guardados
El procedimiento solo debe salvar la pila antes de saltar a sus valores y restaurarlos al regresar. Los otros 3 reg. son guardados. Estos pueden ser alterados sin afectar su comportamiento

3.32) d en p se guarda en la l.3. la sustracción de la línea 6 desde que esta se puede rebajar hasta otras determina tipos y pos. de los argumentos x y c
int fun(short c, char d, int *p, int x);
En este ejemplo se muestra como la ing. inversa es importante para identificar los puntos importantes del programa

3.28) Flujo de control de un estado switch

1º) La línea 2 agrega 2 a x para establecer rango inferior de los casos de a

2º) Líneas 3 y 4 hacen que salte al case y guarda el valor a t.
-2+6=4

3º) La entrada en la línea 3 tiene el mismo destino como la instr. de la línea 4. (=1 no se encuentra en el switch)

4º) Las entradas a las líneas 6, 7 tienen el mismo destino

(etq 2, 3) luego:
- Las etq. del case en el switch tienen valores -2, 0, 1, 2, 3, 4
- El caso con destino L6 tenía etq. 2, 3

3.29) La clave para reutilizar secuencias switch es combinar ensambladores y la tabla de saltos para solucionar los diferentes casos. Se puede ver ya que el case tiene la etq L2, otra etq repartida por la tabla de saltos .L4 para case Cay D. El código cae en la línea 4 y etq L6 que comunde con el case A y L3 al case B, esto deja L2 con el caso E

```
int switcher(int c, int b, int c){
    int ans;
    switch(c){
    case 5:
        c = b^15;

    case 0:
        ans = c+112;

        break;
    case 2:
    case 7:
        ans = (c+b)<<2;
        break
    case 4:
        ans = a;
        break;
    default: ans = b; }
```

4º) El equivalente en goto:

```
int loop_while_goto(int a, int b){
    int result = 1;
    if(a >= b)
        goto done;
    int cpb = a+b;
loop:
    result *= cpb
    a++;
    cpb++;
    if(b > c)
        goto loop;
done:
    return result;
}
```

3.26/ 1º) El operador %/ se puede ver que divide por una potencia de 2, por desp. a la derecha. Antes de contar K=2, hay que ver si el dividendo es negativo

2K-1 = 3 si el dividendo es negativo

2º)
```
leal  3(%edx), %ecx
testl %edx, %edx
cmovs %edx, %eax
sarl  $2, %eax
```

Se crea un valor temporal = x+3, en previsión de x es negativo. Con un cambio de forma condicional para x si x>=0 y se desplaza 2 para generar x/4

3.21)

Podemos ver que el registro se incrementa a c+b y es incrementado con cada iteración. Del mismo modo el valor de c es incrementado en cada iteración. Por tanto, podemos ver el valor en el registro %edx no hará siempre igual c+b.

2º) Registros:

R.	Val	Virtual
%ecx	c	c
%ebx	b	b
%eax	result	1
%edx	c+b	c+b

3º) Código:

```
movl   8(%ebp), %ecx
movl   12(%ebp), %ebx
movl   $1, %eax
cmpl   %ebx, %ecx
jge    .L1
leal   (%ebx, %ecx), %edx
movl   $1, %eax

.L2:
imull  %edx, %eax
addl   $1, %edx
addl   $1, %edx
cmpl   %ecx, %ebx
jg     .L2
.L1
```

3.20)

A) El uso de registros se determina viendo como se captan los argumentos

Uso de Registros

Reg.	Ver	Invocl
%eax	x	x'
%ecx	y	y
%edx	n	n

B) El cuerpo consiste en 3 líneas c 5 en C y 5 a 7 en ensamblador. La expresión test está en la línea 6 en C. En ensamblador se implementa en las instrucciones 8 a 11

C) El código es:

```
movl 8(%ebp), %ecx
movl 12(%ebp), %ecx
movl 16(%ebp), %edx
.L2:
addl %edx, %ecx
imull %edx, %ecx
subl $1, %edx
testl %edx, %edx
je .L5
cmpl %edx, %ecx
jl .L2
.L5:
```

3.19)

n	n!	OK?
1	1	Y
2	2	Y
3	6	Y
4	24	Y
5	120	Y
6	720	Y
7	5,040	Y
8	40320	Y
9	362880	Y
10	3,620,800	Y
11	39,916,800	Y
12	479,001,600	Y
13	1,932,053,504	Y
14	1,278,945,280	N

14 se ha desbordado ya que los números han pasado de

veces