PROBLEMA 18

```
int mat1[M][N];
int mat2[N][M];

int SumaElemento(int i, int j){
    return mat1[i][j] + mat2[j][i];
}
```

SumaElemento:

```
pushl %ebp
movl %esp, %ebp

movl 8(%ebp), %eax   // %eax ← i
movl 12(%ebp), %ecx  // %ecx ← j
sall $2, %ecx        // %ecx ← 4j
leal (,%eax,8), %edx // %edx ← 8i
subl %eax, %edx      // %edx ← 8i - i
leal (%ecx,%eax,4), %eax // %eax ← 4j+4i
movl mat2(%ecx,%eax,4), %eax
addl mat1(%ecx,%edx,4), %eax

movl %ebp, %esp
popl %ebp
ret
```

mat2 + 4j + 4i + 16i

%eax ← 4j + (4i) 4 + mat2

%ecx ← 4j + (8i) 4 + mat1

4j + 32i + mat1

a) ¿Cuánto valen las constantes M y N?

@mat2[i][j] = @mat2 + 4(i*M + j) = @mat2 + 4iM + 4j

@mat1[i][j] = @mat1 + 4(i*N + j) = @mat1 + 4iN + 4j

$4iM = 20i \Rightarrow M = \frac{20i}{4i} = 5$

$4iN = 28i \Rightarrow N = \frac{28i}{4i} = 7$

M = 5 **N = 7**

b) ¿Cuántas instrucciones estáticas tiene el código?

13 instrucciones estáticas

c) ¿Cuántas instrucciones dinámicas tiene el código?

13 instrucciones dinámicas

d) ¿Cuántos accesos a memoria se producen al ejecutar este código?

4 accesos a memoria

e) Suponiendo que cada ciclo se ejecutan 0.8 instrucciones si éstas no acceden a la memoria de datos y 0.5 si acceden a la memoria de datos, ¿cuántos ciclos tarda en ejecutarse el programa anterior?

0.5 memoria → 1 instrucción → 2 ciclos (2 c./i.) → 2 · 9 → 18 ciclos

0.8 no memoria → 1 instrucción → 1.25 ciclos (1.25 c./i.) → 1.25 · 4 → 5 ciclos

} 23 ciclos

f) Si cambiamos la memoria del procesador de forma que los accesos a las instrucciones fuesen más rápidos y se ejecutaran 0.1 instrucciones más por ciclo ¿cuál sería la ganancia para este programa?

0.6 memoria → 1 instrucción → 1.667 ciclos (1.667 c./i.) → 15 ciclos

0.4 no memoria → 1 instrucción → 1.111 ciclos (1.111 c./i.) → 4.444 ciclos

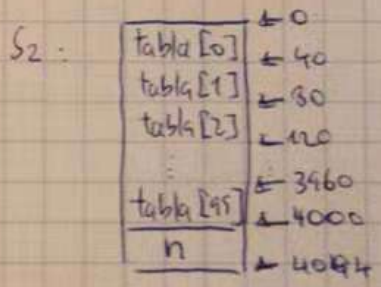
} 19.444 ciclos

$\text{Ganancia} = \frac{23}{19.444} = 1.182$ = Un 18.2% más rápido

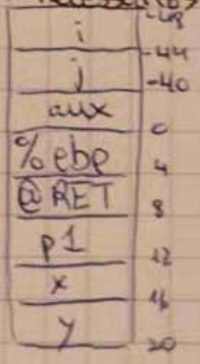
PROBLEMA 19

```
typedef struct {
  int i1;
  char c2[30];
  int i3;
} sx;

typedef struct {
  sx tabla[100];
  int n;
} s2;

int F(sx *p2, int y);

int examen (s2 *p1, int *x, int y){
  int i, j;
  sx aux;
  ...
}
```

a) Dibujad como quedarían almacenadas en memoria las estructuras sx y s2, indicando claramente los desplazamientos respecto el inicio y el tamaño de todos los campos



b) Dibujad el bloque de activación de la función examen, indicando claramente los desplazamientos relativos al EBP necesarios para acceder a los parámetros y las variables locales



c) Traducad la siguiente sentencia, suponiendo que está dentro de la función examen:

return (*x + aux.i3); →

```
movl 12(%ebp), %eax
movl (%eax), %eax
addl 4(%ebp), %eax
```

d) Traducid la siguiente sentencia, suponiendo que está dentro de la función externa:

```
aux1 = F(&(*p1).tabla[j], y);

movl 8(%ebp), %eax      // %eax = p1
movl 44(%ebp), %ecx     // %ecx = j
imull $40, %ecx
addl %ecx, %eax         // %eax = &(*p1).tabla[j]
movl 16(%ebp), %ecx
pushl %ecx
pushl %eax
call F
addl $8, %esp
movl %eax, -40(%ebp)
```

e) Traducid:

```
i = j * y;

movl 16(%ebp), %eax
movl 44(%ebp), %ecx
imull %eax, %ecx
movl %ecx, -48(%ebp)
```

f) Traducid:

```
aux.c2[i] = aux.c2[23];

movb -13(%ebp), %al
leal -40(%ebp), %ecx
addl $4, %ecx
addl -48(%ebp), %ecx
movb %al, (%ecx)
```

g) Traducid:

```
for (i = 0; (i < y) && (i < (*p1).n); i = i+5)
    (*p1).tabla[i].i1 = (*p1).tabla[i].i3 + i;

pushl %edi
movl $0, %ecx           // %ecx = i
movl 8(%ebp), %ecx      // %ecx = p1
for: cmpl 16(%ebp), %eax
jge fifor
cmpl 4000(%ecx), %eax
jge fifor
imull $40, %eax, %edx   // i*40
addl %ecx, %edx         // p1+i*40
movl %edx, %edi
movl 36(%edi), %edi     // *p1+i*40+i3
addl %eax, %edi
movl %edi, (%edx)
addl $5, %eax
jmp for

fifor: popl %esi
```

h) Traducid:

```
if ( aux.i1 != y)
    aux.i3 = i;
else
    aux.i3=0;
```

```
if:   movl 16(%ebp), %eax
      movl -40(%ebp), %ecx
      cmpl %eax, %ecx
      je  else
jmpend movl -48(%ebp), %edx
      movl %edx, -36(%ecx)
else: movl -44(%ebp), %edx
      movl %edx , -36(%ecx)
end:
```

i) Traducid:

```
i=0;
while (aux.c2[i] != ' '){
    aux.c2[i] = '#';
    i++;
}

movl $0, %eax     //%eax=i
leal -40(%ebp), %ecx
while: cmpb $' ', 4(%ecx,%eax)
    je fi
    movb $'#', 4(%ecx,%eax)
    incl %eax
    jmp while
fi:
```