Problema 18.

Considerad el siguiente código escrito en C, suponiendo que las constantes N y M han sido declaradas previamente en un `#define`.

```
int mat 1[M][N];
int mat2[N][M]

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

Para esta subrutina el compilador genera el siguiente código en ensamblador:

```
SumaElemento:
      pushl %ebp
      movl %esp, %ebp

      movl 8(%ebp), %eax
      movl 12(%ebp), %ecx
      sall $2, %ecx
      leal (,%eax,8), %edx
      subl %eax, %edx
      leal (%eax, %eax, 4), %eax
      movl mat2(%ecx, %eax, 4), %eax
      addl mat1(%ecx, %edx, 4), %eax

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

a) ¿Cuánto valen las constantes `M` y `N`?

b) ¿Cuántas instrucciones estáticas tiene el código?

c) ¿Cuántas instrucciones dinámicas tiene el código?

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

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?

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?

Problema 19.

Dado el siguiente código escrito en C:

```
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 quedarian almacenadas en memoria las estructuras sx y s2, indicando claramente los deplazamientos 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) Traducid la siguiente sentencia a ensamblador del x86, suponiendo que está dentro de la función examen:

```
return(*x+aux.i3);
```

d) Traducid la siguiente sentencia a ensamblador del x86, suponiendo que está dentro de la función examen:

```
aux.i1 = F(&(*p1).tabla[j], y);
```

e) Traducid la siguiente sentencia a ensamblador del x86, suponiendo que está dentro de la función examen:

```
i = j * y;
```

f) Traducid la siguiente sentencia a ensamblador del x86, suponiendo que está dentro de la función examen:

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

g) Traducid la siguiente sentencia a ensamblador del x86, suponiendo que está dentro de la función examen:

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

h) Traducid la siguiente sentencia a ensamblador del x86, suponiendo que está dentro de la función examen:

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

i) Traducid la siguiente sentencia a ensamblador del x86, suponiendo que está dentro de la función examen:

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