Laboratorio Sesión 03: Introducción al ensamblador de la arquitectura x86: Estructuras de control y matrices

Objetivo

El objetivo de esta sesión es introducir la programación en ensamblador para la arquitectura x86. En concreto se trabajarán aspectos como la programación de estructuras de control (condicionales e iterativas) y el acceso a elementos estructurados (vectores y matrices).

Conocimientos Previos

Para realizar esta práctica deberíais repasar las traducciones directas de C a ensamblador del x86 de las estructuras de control que habéis visto en la clase de teoría. Además deberías respasar los modos de direccionamiento del x86.

Acceso a un vector en ensamblador

Para acceder a un elemento `i` de un vector `Vector` mediante un acceso aleatorio, la posición de memoria a la que debéis acceder es:

$$@Vector + i \times \langle tamaño\_en\_bytes\_de\_un\_elemento \rangle$$

En cambio, si queréis hacer un acceso secuencial a un elemento `i` a partir del anterior deberéis tener en cuenta:

$$@Vector[i] = @Vector[i-1] + \langle tamaño\_en\_bytes\_de\_un\_elemento \rangle$$

Acceso a una matriz en ensamblador

Si lo que queréis es acceder a un elemento en la posición `fila`, `columna` de una matriz `Matriz` mediante un acceso aleatorio, la posición de memoria a la que debéis acceder es:

$$@Matriz + (fila \times \langle columnas \rangle + columna) \times \langle tamaño\_en\_bytes\_de\_un\_elemento \rangle$$

Para realizar accesos secuenciales, dependerá de la dirección (y el sentido) del acceso. Los dos accesos secuenciales más comunes con matrices son por filas:

$$@Matriz[fila][columna] = @Matriz[fila][columna-1] + \langle tamaño\_en\_bytes\_de\_un\_elemento \rangle$$

O por columnas:

$$\begin{aligned} @Matriz[fila][columna] \quad = \quad & @Matriz[fila-1][columna] + \\ & \langle columnas \rangle \times \langle tamaño\_en\_bytes\_de\_un\_elemento \rangle \end{aligned}$$

Estudio Previo

1. Traduce a ensamblador el siguiente bucle:

```
#define N 10
int Matriz[N][N],i,suma;

for (i=0,suma=0;i<N;i++)
  suma+=Matriz[3][i];
```

2. Realiza el mismo bucle en acceso secuencial. Calcula cuántas instrucciones se ejecutan en cada versión.

3. Traduce a ensamblador el siguiente código:

```
#define N 10
#define M 100
int Matriz[N][N],i,j,ResFila[N];

for (i=0,j=0,ResFila[0]=1;i<N;i++,j=0,ResFila[i]=1)
  while(Matriz[i][j]!=0) {
    if (Matriz[i][j]==M)
      ResFila[i]*=Matriz[i][j];
    j++;
  }
```

Trabajo a realizar durante la Práctica

1. Dada una rutina que tiene el siguente código en alto nivel:

```
int OperaVec(int Vector[], int elementos) {
// La @ de Vector esta en la @ 8[ebp] y el
//   valor de la variable elementos en la @ 12[ebp]
int i;  // i esta en la @ -8[ebp]
int res; // res esta en la @ -4[ebp]

res=Vector[0];
// Código que has de introducir
for (i=1;i<elementos;i++)
  if (Vector[i]<res)
    res=Vector[i];
// Fin del código a introducir

return res;
}
```

@Vector + i*4

```
        movl $1, %eax           # i = 1
for:
        cmpl 12(%ebp), %eax
        jge end                 # i >= elementos
if:
        movl 8(%ebp), %ecx      # @Vector -> %ecx
        movl (%ecx, %eax, 4), %ecx      # i * 40 + @Vector -> %ecx || Vector[i] -> %ecx
        cmpl -4(%ebp), %ecx
        jge end_for             # Si Vector[i] >= res salta
        movl %ecx, -4(%ebp)     # res = Vector[i]
end_for:
        incl %eax                       # i++
        jmp for
end:
        movl %eax, -8(%ebp)
        movl -4(%ebp), %eax
```

Traduce el interior de la rutina a ensamblador y ponlo dentro del código `Practica3CompletarA.s`. Ejecútalo con el programa `Practica3MainA.c` y, cuando funcione, cálcula cuántos ciclos tarda, cuántas instrucciones ejecuta y cuál es el CPI resultante. Entregad en el Racó de la asignatura el fichero `Practica3CompletarA.s`.

2. Dada una rutina que tiene el siguente código en alto nivel:

```
#define N 3

int OperaMat(int Matriz[N][N], int salto) {
// La @ de Matriz esta en la @ 8[ebp] y el
//   valor de la variable salto en la @ 12[ebp]
int j; // j esta en la @ -12[ebp]
int i;  // i esta en la @ -8[ebp]
int res; // res esta en la @ -4[ebp]

// Codigo que has de introducir
res=0;
for (i=0; i <3; i+=salto)
  for (j=0; j <3; j++)
    res-=Matriz[i][i]+j;
// Fin del codigo a introducir

return res;
}
```

@Matriz + (i*3 + i)*4 = @Matriz + 16i

```
        movl $0, -4(%ebp)       # res = 0
        movl $0, %eax           # i = 0
        movl 8(%ebp), %ebx      # @Matriz -> %ebx
first_for:
        cmpl $3, %eax
        jge end                 # Si i >= 3 salta
        movl $0, %ecx           # j = 0
second_for:
        cmpl $3, %ecx
        jge end_for             # Si j >= 3 salta
        imull $16, %eax, %edx   # 16 * i -> %edx
        movl (%ebx, %edx), %edx # Matriz[i][i] -> %edx
        addl %ecx, %edx         # Matriz[i][i] + j -> %edx
        subl %edx, -4(%ebp)     # res -= Matriz[i][i] + j
        incl %ecx                       # j++
        jmp second_for
end_for:
        addl 12(%ebp), %eax     # i += salto
        jmp first_for
end:
        movl %eax, -8(%ebp)
        movl %ecx, -12(%ebp)
```

Traduce el interior de la rutina a ensamblador y ponlo dentro del código `Practica3CompletarB.s`. Ejecútalo con el programa `Practica3MainB.c` y, cuando funcione, cálcula cuántos ciclos tarda, cuántas instrucciones ejecuta y cuál es el CPI resultante. Entregad en el Racó de la asignatura el fichero `Practica3CompletarB.s`.

3. Explica qué optimizaciones de código crees que se podrían aplicar a los dos códigos realizados.

Hay que ejecutar los códigos en máquinas de 32 bits

gcc Practica3MainA.c Practica3CompletarA.s -o Practica3

o

gcc -o Practica3 Practica3MainA.c Practica3CompletarA.s

1. Usar valgrind para que nos de el número de instrucciones
2. Descomentar ticks para obtener los ciclos
3. Calcular CPI = $\frac{\text{ciclos}}{\text{instrucciones}}$

Nombre:_____ Grupo:_____

Nombre:_____

Hoja de respuesta al Estudio Previo

1.
```
for (i=0,suma=0;i<N;i++)
    suma+=Matriz[3][i];
```
@Matriz + (3*10 + i)*4 = @Matriz + 120 + 4i

La traducción a código ensamblador del anterior código C es:

```
        movl $0, %eax        # i=0
        movl $0, %ecx        # suma=0
for:    cmpl 10, %eax        # Si i≥10 salta
        jge end
        addl Matriz($120, %eax, 4), %ecx   # suma += M[3][i]
        incl %eax            # i++
        jmp for
end     movl %eax, i
        movl %ecx, suma
```

Número instrucciones ejecutadas = 2 + 10*5 + 2 + 2 = 56

2. Realizando acceso secuencial la traducción es:

```
        movl $0, %eax        # i=0
        movl $0, %ecx        # suma=0
        leal Matriz($120), %edx   # @M[3][0] → %edx
for:    cmpl $10, %eax       # Si i≥10 salta
        jge for
        addl %edx, %ecx      # suma += M[3][i]
        addl $4, %edx        # acceso secuencial
        jmp for
end:    movl %eax, i
        movl %ecx, suma
```

@Matriz + 120 + 4*i → Cada 4

Número ejecuciones ejecutadas = 3 + 10*5 + 2 + 2 = 57

La versión aleatoria ejecuta: 56 instrucciones. La secuencial ejecuta: 57 instrucciones.

3.
```
for (i=0,j=0,ResFila[0]=1;i<N;i++,j=0,ResFila[i]=1)
  while(Matriz[i][j]!=0) {
    if (Matriz[i][j]==M)
      ResFila[i]*=Matriz[i][j];
    j++;
  }
```

@ResFila + 4i

@Matriz + (i*10 + j)*4 = @Matriz + 40i + 4j

La traducción a código ensamblador del anterior código C es:

```
        movl $0, %eax          # i=0
        movl $0, %ecx          # j=0
        movl $1, ResFila       # ResFila[0]=1
for:    cmpl $10, %eax         # Si i≥N salta
        jge end
        imull $40,%eax, %ebx   # 40i -> %ebx
        movl Matriz(%ebx,%ecx,4), %ebx   # Matriz[i][j] -> %ebx
while:  cmpl $0, %ebx          # Si Matriz[i][j]==0 salta
        je end_for
if:     cmpl $100, %ebx        # Si Matriz[i][j] !=M salta
        jne end_if
        imull %ebx, ResFila(,%eax,4)   # ResFila[i] *= Matriz[i][j]
end_if: incl %ecx              # j++
        jmp while
end_for: incl %eax             # i++
        movl $0, %ecx          # j=0
        movl $0, ResFila(,%eax,4)   # ResFila[i]=1
end:    movl %eax, i
        movl %ecx, j
```

Nombre: _____ Grupo: _____

Nombre: _____

Hoja de respuestas de la práctica

NOTA: Recordad que para compilar los programas en ensamblador 32 bits deberéis usar la opción de compilación de *gcc* `-m32`.

1. Entregad en el Racó de la asignatura el fichero `Practica3CompletarA.s`. El programa completo ejecuta 98 217 013 instrucciones en 24 125 774 ciclos y con un CPI de: 0'3475.
2. Entregad en el Racó de la asignatura el fichero `Practica3CompletarB.s`. El programa completo ejecuta 91 217 013 instrucciones en 36 195 985 ciclos y con un CPI de: 0'3968.
3. Las optimizaciones de código que se podrían aplicar a los dos códigos realizados son:

   En ambos programas podríamos pasar de utilizar acceso aleatorio a acceso secuencial, también podríamos "desenrollar" el bucle realizando más de una lectura y por último utilizar instrucciones SIMD.

4. Recordad entregar en el Racó de la asignatura los ficheros `Practica3CompletarA.s` y `Practica3CompletarB.s`. Debéis entregar sólo los dos ficheros fuentes, sin comprimir ni cambiarles el nombre, y sólo una versión por pareja de laboratorio (es indistinto que miembro de la pareja entregue).