

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 deberíais repasar 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:

$$@Matriz[fila][columna] = @Matriz[fila-1][columna] + \langle columnas \rangle \times \langle tamaño.en.bytes.de.un.elemento \rangle$$

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[i][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 siguiente 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;
}
```

Traduce el interior de la rutina a ensamblador y ponlo dentro del código Practica3CompletarA.s. Ejecútalo con el programa Practica3MainA.c y, cuando funcione, calcula 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 siguiente 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+=salto)
            res+=Matriz[i][j];
    // Fin del codigo a introducir

    return res;
}
```

Traduce el interior de la rutina a ensamblador y ponlo dentro del código Practica3CompletarB.s. Ejecútalo con el programa Practica3MainB.c y, cuando funcione, calcula 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.

Nombre: Liang Chen

Grupo: 12

Nombre: _____

Hoja de respuesta al Estudio Previo

1. for (i=0,suma=0;i<N;i++)
 suma+=Matriz[i][i];

La traducción a código ensamblador del anterior código C es:

```
xorl %ecx,%ecx #i = 0
xorl %eax,%eax #suma = 0
leal Matriz,%edx #edx = @Matriz
for:  cmpl $10,%ecx # i - 10
      jge ffor
      addl (%edx,%ecx,4),%ebx#ebx=@M+40i
      addl (%ebx,%ecx,4),%eax #Suma = ...
      incl %ecx #++i
      jmp for
ffor:
```

2. Realizando acceso secuencial la traducción es:

```
xorl %ecx,%ecx #i = 0
xorl %eax,%eax #suma = 0
leal Matriz,%edx #edx = @Matriz
for:  cmpl $440,%ecx # i - 440
      jge ffor
      addl (%edx,%ecx),%ebx#ebx=@M+i
      addl (%ebx,%ecx),%eax #Suma = ...
      addl $44, %ecx #i+44
      jmp for
ffor:
```

La versión aleatoria ejecuta:

63 instrucciones.

La secuencial ejecuta: **63**

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++;
    }
```

La traducción a código ensamblador del anterior código C es:

```
xorl %ecx,%ecx # i = 0
xorl %edx,%edx # j = 0
movl $1, ResFila
leal Matriz, %eax # eax = @Matriz
leal ResFila, %edi #edi = @ResFila
for: cmpl $10, %ecx #i -10

    jge ffor

while: cmpl $0, (%eax,%edx,4) #@M+j*4

    je fwhile

if:  cmpl $100, (%eax,%edx,4)

    jge fiif

    movl (%eax,%edx,4), %esi #esi = @M+4j =M[i][j]

    addl %esi,(%edi,%ecx,4)

fiif: incl %edx # ++j

    jmp while

fwhile: xorl %edx,%edx #j = 0

    incl %ecx #++i

    movl $1, (%edi,%ecx,4) #ResFila[i] = 1

    addl $40,%eax #M[i][j] = @Matriu 10x10 = 40 bytes per fila

    jmp for

ffor:
```

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 instrucciones en ciclos y con un CPI de: .
2. Entregad en el Racó de la asignatura el fichero `Practica3CompletarB.s`. El programa completo ejecuta instrucciones en ciclos y con un CPI de: .
3. Las optimizaciones de código que se podrían aplicar a los dos códigos realizados son:

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).