Optimizacion codigo para matrizes

- Ver la matriz como un vector y recorrerlo, recortas un bulce (-30%).
- Desenrollar Codigo, instruciones de salto consumen mucho tiempo(-30%).
- Las instrucciones multimedia, pueden mover 128 bytes de golpe, simplifican el desenrollar. [..]

Instrucciones Multimedia (SIMD)

Sirven para hacer operaciones con una cantidad grande de bytes(128).

[tabla de instrucciones]

Structs

Ensamblador no soporta los datos estrucurados, los programadores deben mantenerlos manualmente.

Alineaminento de Datos (linux 32 bits)

Para asegurar que los sistemas de cache i paginación funcionan correctamente y para poder hacer acesos a memoria por longword o quadword, los datos deben alinearse. El compilador es el que inserta "espacios en blanco".

Por tanto, "una direccion debe ser múltiplo de k".

- char alineado a 1-byte (@ cualquiera)
- short alineado a 2-bytes (@ ...0)
- int alineado a 4-bytes (@ ...00)
- puntero(4 bytes), double(8 bytes), Long double(12 bytes) alineado a 4 bytes

Las estructuras, debe cumplir la restriccion del elemento maximo(maxima alineacion) que contien sus campos, tambien su @ inicio lo tiene que cumplir. El orden de los campos de un struct pueden hacer que este ocupe mas.

GESTIO DE SUBRUTINAS

En C-linux 32 bits, los parametros se pasan por la pila, **de derecha a izquierda**.

- Vectores y matrices siempre se pasan por &.
- Los structs se pasan por valor siempre.
- Char(1byte) ocupan 4 bytes y Short(2bytes) ocupan 4 bytes.
- Las variables locales se alinean en la pila como si fuera un struct. El tamaño de variables locales debe ser multiplo de 4.
- **%ebp, %esp se salvan siempre implicitamente.**
- **%ebx, %esi, %edi se han de salvar si son modificados por la rutina**.
- **%eax, %ecx, %edx se pueden modificar dentro de la subrutina, el llamador debe salvarlos.**
- **Los resultados se devuelven por %eax.**
- La pila siempre debe estar alineado a 4.

Bloque de activacion - Ejemplo de Subrutina

1. Paso de Parametros y llamada

   Se colocan en la pila de derecha(primeros) a izquierda, al hacer push el %esp se desplaza 4 hacia arriba(%esp - 4). Los registros que utilizamos (%eax, %ecx o %edx) deben ser salvados antes de llamar. Por ultimo, se invoca la subrutina, al hacer call %esp = %esp - 4 y se guarda la @ret.

2. Enlace dinámico y * al bloque de activacion

   Push de %ebp a la pila y %ebp se coloca donde %esp.

3. Variables locales y salvar estado del llamador

   Agrandamos la pila, subiendo el %esp para hacer caber las variables locales, en el orden que se declaran. Los registros que necesitaremos utilizar (%ebx, %esi o %edi) los guardamos en la pila; los otros pueden ser modificados sin problema.

4. Cuerpo de la subrutina y retorno

   Se ejecuta el cuerpo de la subrutina, teniendo en cuenta el bloque de activacion. El resultado a retornar siempre debe colocarse en %eax.

5. Restaurar estado de registros y eliminacion de variables locales.

   Pop de los registros que hayan sido utilizados (%ebx, %esi o %edi), para las variables locales, bajamos el %esp (%esp + x). Por ultimo restauramos el %ebp y hacemos ret.

6. Recoger/usar el resultado

   Para la rutina padre, el resultado estara en %eax;

Gestion de Registros en Subrutinas

- Registros %eax, %ecx i %edx (registros no seguros)

  Se pueden modificar dentro de una subrutina --> El padre debe guardarlos al invocar.

- Registros %ebx, %esi, %edi (registros Seguros)

  Si una subrutina quiere modificarlos, debe guardarlos --> El padre no debe guardarlos al invocar.

- Registros %ebp, %esp

  Registros para la gestion de la pila.