Ejemplo sobre uso de memoria

Arquitectura del Computador - LCC - FCEIA-UNR

Agosto 2021

En este ejemplo se pretende mostrar algunas cuestiones y precauciones al momento de trabajar con datos en memoria. También es útil para familiarizarse con GDB.

Dado el siguiente código en Assembler X86-64 en el archivo `memory_ejemplo.s`:

```
.data

a: .long 0x11223344
b: .long 0x55667788

.text
.global main

main:
    movq a, %rax    #<----- Línea 1
    retq
```

Compilar utilizando la opción `-g` para poder *debuggear* utilizando GDB:

```
$ gcc -g memory_ejemplo.s
```

Luego ejecutar utilizando GDB:

```
gdb ./a.out
```

En primer lugar, es interesante ver en qué parte de la memoria están los datos. Para ello, una vez que hayamos corrido el programa dentro de GDB, podemos ver la dirección de la etiqueta `a` de la siguiente manera:

```
(gdb) info address a
```

El resultado es:

```
Symbol "a" is at 0x600870 in a file compiled without debugging.
```

También podríamos haber imprimido un puntero a la etiqueta `a`:

```
(gdb) print &a
$1 = (<data variable, no debug info> *) 0x600870
```

Ahora, es interesante verificar que efectivamente en esa dirección está el dato en cuestión:

```
(gdb) x/1xb &a
0x600870: 0x44
```

Aquí hemos usado el comando `x`, el cual deriva de "examine" y, además, como opciones hemos indicado que nos muestre un byte en formato hexadecimal (`1xb`). Entonces vemos que en la dirección `0x600870` hay almacenado un `0x44`. Ahora veamos qué hay almacenado en la siguiente dirección de memoria:

```
(gdb) x/1xb 0x600871
0x600871: 0x33
```

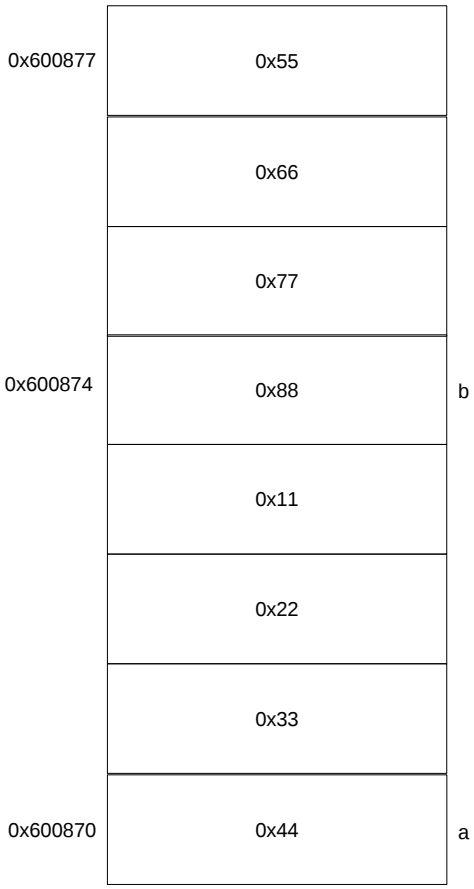
Esto es coherente si recordamos que en las arquitecturas Intel los datos mayores a un byte son almacenados en memoria utilizando el formato *little-endian*. Entonces, en la dirección etiquetada con `a` se encuentra almacenado el byte menos significativo y los siguientes bytes están almacenados hacia direcciones mayores.

Análogamente, podemos ver como está almacenado el dato a partir de la etiqueta `b`:

```
(gdb) x/1xw &b
0x600884: 0x55667788
```

donde la `w` es por *word* (4 bytes).

En la siguiente figura podemos ver una representación del esquema de memoria:



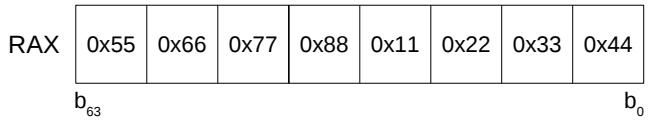
Ahora supongamos que queremos cargar algunos de estos datos en un registro. Por ejemplo, cargar el dato almacenado a partir de la etiqueta `a` al registro `rax` tal como se realiza en la línea 1 del código del ejemplo:

```
movq a, %rax
```

Veamos entonces el contenido del registro `rax`, una vez ejecutada la línea anterior:

```
(gdb) i r rax
rax            0x5566778811223344 6153737367135073092
```

El resultado se puede ver en el siguiente esquema, donde se observan los 8 bytes del registro `rax`:



¿Cómo se interpreta este resultado? La instrucción `movq a, %rax` cargó 8 bytes (debido al sufijo `q`) a partir de la dirección de memoria etiquetada con `a` (`0x600870`). En esta dirección comienza un dato tipo `long` (4 bytes). Entonces, cargó los 4 bytes almacenados a partir de la dirección etiquetada con `b` (`0x11223344`) más los 4 bytes a partir de la dirección de memoria etiquetada como `b` (`0x55667788`).

Como conclusión, este ejemplo muestra varias cuestiones:

- La memoria se va ocupando como bloques consecutivos de acuerdo a la directiva utilizada (por ejemplo, `.long` asigna un bloque de 4 bytes).
- Las etiquetas son formas de referenciar la dirección de memoria dónde comienza el bloque:

```
movq $a, %rax   #----> rax=0x600870
```

- Al momento de acceder a los datos almacenados en memoria hay que tener cuidado con el sufijo utilizado en la instrucción `mov`.