

Guión de la sesión cero

Jose Luis Díaz

2006

Resumen

Esta sesión cubre temas de codificación y aritmética de enteros, y a la vez es una introducción al entorno en que se desarrollarán futuras prácticas. En particular, se abordan los siguientes puntos:

- Realizar ejercicios de codificación de naturales y enteros, y operaciones aritméticas con los mismos (suma y resta sobre el papel), teniendo en cuenta a la vez cómo la limitación de bits puede producir desbordamiento.
- Observar sobre programas reales (en C++) estas limitaciones.
- Presentar al alumno el entorno de trabajo, la interfaz de comandos de Windows y el simulador de la CPU teórica.

1. Introducción a las sesiones prácticas

Charla sobre el formato de las sesiones prácticas. Tocar los siguientes aspectos:

- Habrá tres bloques de prácticas. Los bloques se dividen en sesiones. Las sesiones son de una hora, pensadas para realizar en horario de laboratorio, pero deben completarse en casa si no diera tiempo, pues a la semana siguiente se inicia sesión nueva que puede depender de la anterior.
- El trabajo a desarrollar en cada sesión se describe en un documento. El alumno debe llevarlo impreso (si no, se pone falta). También debe leer la primera página del documento con anterioridad a la realización, pues en ella se detallan los requisitos.
- El desarrollo de la práctica está muy guiado por el documento, donde se explica paso a paso todo lo que hay que hacer. El profesor apenas dará explicaciones en la pizarra, sino que resolverá dudas individuales.
- Los documentos de prácticas son a la vez exámenes de autoevaluación, puesto que contienen preguntas y casillas de respuesta. El alumno debe asegurarse de que sabe responder todo, y si no preguntar al profesor. Estos documentos no se entrega, ni se evalúan, son sólo para que el alumno pueda verificar su propio progreso.
- La única evaluación que se realizará consistirá en la entrega de un trabajo obligatorio al final del curso. El enunciado de ese trabajo se publicará en su momento, y para su ejecución será necesario poner en juego conceptos de todos los bloques prácticos.

2. Introducción al entorno

- Entrar en sesión. Señalar la carpeta en la que se encuentra el material de la asignatura. Mostrar cómo se “conecta” como unidad de red (letra Y:). Describir someramente los contenidos de esta carpeta, haciendo hincapié en la que contiene una versión del programa “para llevar”.
- Mostrar cómo abrir una interfaz de comandos (CMD), y explicar los comandos básicos para navegar por carpetas, copiar ficheros, etc. desde esta ventana. En particular, mostrar cómo acceder a la unidad Z: de usuario y cómo crear en ella una carpeta para esta práctica (que llamaremos P0), cómo mostrar los contenidos de la carpeta con DIR, etc.
- Copiar los archivos que hay en la carpeta de la asignatura a la carpeta P0 recién creada en Z:

3. Naturales y enteros en C++

3.1. Naturales

Utilizando EDIT, mostrar el código fuente de naturales-16bit.cpp. Hacer hincapié en la declaración de `a, b, c` (`short` implica el uso de 16 bits, y `unsigned` que se trata de números naturales siempre positivos).

Realizar “sobre el papel” (aunque puede ayudarse de la calculadora de Windows) los siguientes ejercicios.

- Hacer que el alumno calcule el máximo valor almacenable en estas variables.
- Pedir al alumno que codifique el número mil, en binario, usando el método de las divisiones sucesivas.
- Pedir que lo codifique en hexadecimal, también por divisiones sucesivas.
- Pasar el hexadecimal a binario, por el método de los bits de 4 en 4 para comprobar que le sale lo mismo.

Mediante la ejecución del programa, se pueden realizar los siguientes experimentos:

- Vamos a ejecutar el programa, y responder 1000 para los valores de `a` y `b`. Preguntar al alumno si el resultado cabe en el rango. Ejecutar el programa y comprobar que la respuesta sale bien. El programa muestra la respuesta en hexadecimal. Pedir que la pasen a binario por el método de los 4 bits, y que comparen el valor con el que habían calculado antes para mil. ¿Ven alguna similitud? Explicar que en binario multiplicar por dos es lo mismo que añadir un cero a la derecha, o también desplazar todos los bits una posición hacia la izquierda.
- Repetir la ejecución pero ahora dando como valores 20 000 y 30 000. Preguntar si creen que el resultado cabe, y comprobar la respuesta.
- Finalmente, repetir la ejecución dando como datos a sumar 30 000 y 40 000 ¿cabe el resultado? ¿qué saldrá? Ejecutarlo y comprobarlo. Asegurarse de que todo el mundo entiende el resultado obtenido.
- Pedir que codifiquen estos dos números en binario, y que realicen la suma sobre el papel (aritmética binaria), decodificando finalmente el resultado final. Preguntar también cuál es el bit de acarreo y cuál sería el de *overflow*.

3.2. Enteros

Ahora trabajaremos con enteros, codificados en complemento a 2. Comenzamos por estudiar el código fuente de `enteros-16bit.cpp`, haciendo hincapié en que la única diferencia con el anterior es que la palabra `unsigned` ha desaparecido. Se podría haber sustituido por `signed`, pero esto es superfluo ya que en C y C++ los enteros son `signed` por defecto, lo que significa que pueden tomar valores tanto positivos como negativos (para lo que usa C-2).

Realizar los siguientes ejercicios “sobre el papel”:

- Calcular el rango para las variables `a, b, c`
- Pedir al alumno que codifique los números -500 y 500 y que sume ambos “sobre el papel”. Preguntar por el bit de acarreo y el de *overflow*
- Usando el programa, introducir como cantidades a sumar el cero y el -500, y comprobar que el resultado (-500) tiene la codificación que habían calculado en el apartado anterior.
- Pedir que calculen qué número correspondería a esa codificación si se tratara de un número natural (`unsigned`) (se trata del 65536-500, es decir, del 65036).
- Preguntar qué creen que pasará si ejecutan de nuevo el programa `naturales-16bit` y le piden sumar el 65036 y el 500. Explicar el resultado haciendo ver que la CPU realmente sólo suma bits, y que por tanto para ella es exactamente la misma operación 65036+500 que -500+500, ya que 65036 y -500 tienen la misma representación binaria. Hacer hincapié, no obstante, en la diferente interpretación del resultado. Cuando trabajamos con enteros (`signed`), el resultado es correcto, mientras que con naturales (`unsigned`) es erróneo. Aunque la operación es la misma, los bits de acarreo y *overflow* no lo son.

Ejecutando el programa realizar los siguientes ejercicios:

- Pedir al alumno que encuentre dos números que al ser sumados darán un resultado erróneo y que prediga cuál será ese resultado.
- Hacer que lo compruebe con el programa.

3.3. Otros tamaños

Comentar que, además del modificador `short`, existe el `long`, que permite crear enteros de 32 bits. Comentar que sobre estos programas no haremos ejercicios, pues los números a manejar para producir desbordamientos son muy grandes.

Por otro lado, resulta interesante saber que para C y C++, una variable de tipo `char` es en realidad una variable que ocupa 8 bits. Habitualmente es utilizada para almacenar códigos que representan caracteres, pero nada impide en realidad almacenar en ellas enteros, con tal de que su valor “quepa” en 8 bits, e incluso realizar aritmética entre estas variables (es decir, sumar dos variables de tipo `char`).

Puesto que un “char” es tratado por C como un entero especialmente corto, no es sorprendente que se le pueda poner delante el modificador `unsigned` o `signed`, para indicar si estos enteros pequeños son en realidad naturales (siempre positivos), o enteros (que pueden tomar valores positivos o negativos, en C-2).

Esto se muestra en los programas `naturales-8bit.cpp` y `enteros-8bit.cpp`. No obstante estos programas son un poco más enrevesados de la cuenta, debido a que los *streams* `cin` y `cout` tratan una variable de tipo `char` como si contuviera un código de un carácter, y hay que engañarles un poco para lograr que la traten como si contuviera un entero de 8 bits.

El engaño consiste en leer primero el dato sobre un verdadero entero de 8 bits, y después pasar el valor leído a una variable de tipo `char`. Cuando se hace esta asignación, si el dato original tenía más de 8 bits, sólo se copiarán los bits inferiores.

Posteriormente hay que engañar también a `cout` para que muestre el valor del entero almacenado en `c`, en lugar del carácter representado por el código que hay en `c`. El truco consiste en hacer un *casting* a entero.

- Sabiendo esto, si introducimos los datos 400 y 500 en el programa **naturales-8bit** ¿qué valores recibirán en realidad las variables **a** y **b**?

La respuesta se puede encontrar en binario, tomando sólo los 8 bits inferiores, o más fácilmente restando 256 en ambos casos (este truco, en general, consistiría en restar el múltiplo de 256 más cercano)

- Más difícil, si introducimos 400 y 500 en el programa **enteros-8bit** ¿qué valores recibirán realmente las variables **a** y **b**? Comprobarlo con el programa.
- Observar el resultado de la suma en ambos casos. Especialmente, observar el resultado en hexadecimal. Es el mismo. Lo que cambia es su interpretación como natuarl o como entero con signo.

Realizar también los siguientes ejercicios:

- Calcular el rango representable para **a, b, c** en ambos programas.
- Encontrar una pareja de números que estén en el rango representable para ambos casos y que, al sumarlas como naturales dé un resultado correcto, pero como enteros dé incorrecto. (ejemplo: 100+100)
- Encontrar una pareja en el caso contrario, es decir, dos naturales que al ser sumados dan un resultado incorrecto, pero que si se interpretaran como números en C-2 darían correcto. (Ej: cualquier suma de positivo y negativo, que en naturales sería un número menor de 127 con otro mayor de 128).
- Encontrar una pareja de números que dé resultado correcto con ambos programas (cualquier pareja cuya suma sea inferior a 128).

4. Presentación de la CPU elemental

Mostrar dónde se encuentra el programa simulador de la CPU. Mostrar cómo se puede copiar para llevar a casa.

Ejecutarlo y describir someramente lo que se ve.