Universidad Rey Juan Carlos

Arquitectura de Computadores/Arquitectura de Sistemas Audiovisuales II

Práctica 3: Ejecución condicional, bucles

Katia Leal Algara

NOTA: por ahora vamos a implementar todos los programas en un hilo principal de ejecución, sin realizar llamadas a subrutinas.

A. Ejecución condicional: implementación del programa `mayor.asm`

Se pide implementar un programa en el ensamblador del MIPS32 que lea dos números introducidos por el usuario y que después imprima el mayor de los números. A partir de la página A-59 del apéndice encontramos las **instrucciones de salto**. Estas instrucciones nos permite la implementación de ejecución condicional en lenguaje ensamblador, aunque no de la misma manera que un lenguaje de alto nivel. Vamos a utilizar la pseudoinstrucción `bgt` (branch on greater than) para implementar este programa. ¿En qué instrucciones básicas se descompone? Analiza dichas instrucciones y comprueba que entiendes todos los campos de la misma. ¿Cuál es el valor del *offset*? ¿es correcto?

B. Bucles: implementación del programa `multiplos.asm`

Se pide implementar un programa que lea dos números del terminal, A y B, y que a continuación imprima todos los números múltiplos de A que hay desde el número A hasta el número AxB. Se tendrán en cuenta las siguientes restricciones:

- Si B es igual a 0, el programa debe terminar
- Cuando se llega al número AxB, el programa debe terminar
- Cada múltiplo se debe imprimir en una nueva línea.

C. Leer una línea de texto de la consola: implementación del programa `palindromo.asm`

Se pide implementar un programa que lea una línea de texto de la consola y determine si la misma es un palíndromo o no. En esta primera versión, los signos de puntuación, las mayúsculas y minúsculas y los espacios en blanco no se tendrán en cuenta. Especificaciones del algoritmo:

- Lectura de una cadena de caracteres. Se reservarán 1024 bytes para leer del terminal una cadena mediante la directiva `.space`
- Se necesitarán dos punteros, uno apuntando al primer carácter de la cadena y otro que apunte al último, para recorrer la cadena en direcciones opuestas.
- Mientras que los caracteres apuntados sean iguales y el puntero de inicio sea menor al puntero del final, debemos continuar comparando caracteres. Si los caracteres son distintos, la cadena no es un palíndromo. Se debe imprimir el mensaje correspondiente y terminar.
- Si el puntero inicial es mayor o igual el puntero final, la cadena es un palíndromo. Se debe imprimir el mensaje correspondiente y terminar.

D. Convertir una cadena de texto en un número: implementación del programa `atoi_1.asm`

Se pide implementar un programa que lea una línea de texto del terminal, la convierta a un número e imprima dicho número por pantalla. En esta primera versión del algoritmo no se tendrá en cuenta el signo del número y otros caracteres con no sean dígitos. Especificaciones del algoritmo:

- Lectura de una cadena de caracteres. Se reservarán 1024 bytes para leer del terminal una cadena mediante la directiva `.space`
- Se necesitará un puntero para recorrer la cadena y una variable para generar el correspondiente número entero, num.
- Mientras que el siguiente carácter no sea '\n':
    - num = num X 10
    - num = num + (nextChar – '0')

E. Convertir una cadena de texto en un número: implementación del programa `atoi_2.asm`

Esta segunda versión de atoi sí que tendrá en cuenta el signo del número. El signo vendrá indicado por el primer carácter. Dicho carácter se tendrá en cuenta para, una vez calculado el número final, multiplicar por 1 o por -1 según corresponda.

F. Convertir una cadena de texto en un número: implementación del programa `atoi_3.asm`

Esta tercera versión de atoi copiará el comportamiento de la función de librería atoi de UNIX, de tal manera qué, cuando encuentre un carácter que no sea un dígito, detiene la conversión y devuelve el contenido calculado hasta el momento y almacenado en *num*.

G. Convertir una cadena de texto en un número: implementación del programa `atoi_4.asm`

La última versión de atoi debe detectar el desbordamiento cuando el número calculado sea mayor que el que se puede representar con 32 bits. Existen dos puntos en el programa en los que se puede producir desbordamiento:

- Cuando multiplicamos *num* por 10. Para detectar si se produce desbordamiento, después de realizar la multiplicación, debemos comprobar el contenido de los registros especiales *hi* y *lo*. Si hi es distinto de cero, entonces se ha producido desbordamiento, puesto que el número ocupa más de 32 bits. Si *hi* es igual a cero pero *lo* representa un número negativo, entonces también se produce desbordamiento, puesto que el número generado no se puede representar como un número positivo en complemento a dos. Por ejemplo, el número 3000000000 se convierte en -1294967296.
- Cuando realizamos la suma num = num + (nextChar – '0'). En este caso, si el número resultante es menor que cero, se produce desbordamiento.

H. Otros programas: implementa una nueva versión del palíndromo que ignore los espacios en blanco, mayúsculas y minúsculas y los signos de puntuación. De tal manera que reconozca como palíndromos las siguientes frases:

- La moral, claro, mal.
- La ruta natural.
- La ruta nos aportó otro paso natural.
- Le avisará Sara si va él.
- 1 2 321

Escribe un programa que pida 20 números por el terminal, que luego los ordene empleando mediante el método de la burbuja (*bublesort*) y que por último los imprima en orden creciente.