EXAMEN DE PRÁCTICAS

Convocatoria ordinaria

ARQUITECTURA DE COMPUTADORES

3º, IST, ITT, IT, 4º IT-ADE y IT-AEROESPACIAL, URJC

Fuenlabrada, 13 de Diciembre de 2019

AVISO: Asegúrate que tus programas **cumplen** con los siguientes criterios. Si no se cumple alguno de ellos la **nota máxima** de tu examen será de **2 puntos**

- **Cumplimiento de especificaciones**. Se deben cumplir las especificaciones indicadas en el enunciado: nombres de funciones, nombres de archivos, funcionalidad, etc. Compruébalo antes de entregar el examen
- **Respetar el convenio**. Resuelve las preguntas **sin violar** el convenio del uso de registros (ABI del RISC-V)
- **Sin errores en tiempo de ejecución** (Runtime errors). Tus programas no deben generar excepciones al ejecutarse
- **Sin errores al ensamblar**. Los ficheros entregados NO deben dar errores al ensamblarlos. Si una función la has dejado a medio hacer, asegúrate que al menos se ensambla sin errores

Se está desarrollando una aplicación para la **evaluación de expresiones** con el RISCV, desde la línea de comandos. En esta fase inicial se están prototipando algunas de las funciones necesarias. Nuestro jefe de proyecto nos ha asignado la **implementación** de las siguientes funciones, cuya **especificación** es la siguiente:

- *int atoi1(char)*: **Convertir** un dígito (‘0’ - ‘9’) a su **número** correspondiente. La función tiene **un parámetro de entrada**: el carácter a convertir. Devuelve su valor numérico si es un **dígito correcto** (‘0’ - ‘9’) ó **-1** en caso de que sea cualquier otro dígito. Por ejemplo, atoi1(‘3’) devuelve 3, mientras que atoi(‘a’) devuelve -1

- *int evaluar(pcad):* Evaluar una expresión definida por una cadena. Como **parámetro de entrada** tiene un **puntero** a la cadena con la expresión. Devuelve el **valor** de la expresión calculada ó **-1** en caso de error (el error es debido a que la expresión es incorrecta). Las expresiones que se evalúan son **sumas de números enteros de 1 dígito**. Los números son siempre positivos. Así, esto son ejemplos de expresiones correctas: “1+1”, “0+0+5+9”, “1”,

"3+5+6+2", que darían como resultado los valores 2, 14, 1 y 16 respectivamente. Esta función comprueba si la expresión es correcta, y en caso de serlo, calcula su valor, y si no lo es devuelve -1. Ya disponemos de su **implementación en pseudocódigo python** y nuestra misión será transcribirla a lenguaje ensamblador del RISC-V, siendo lo más fieles posibles al algoritmo dado (NO la podemos implementar cómo queramos, sino que tiene ser usando exactamente este algoritmo)

```
def evaluar(cad):
  #-- Si el primer carácter es '\n' se devuelve error
  if cad[0]=="\n":
    return -1

  #-- Convertir el primer digito a numero (op1)
  op1 = atoi(cad[0])

  #-- Si la conversion falla, se devuelve error
  if op1 == -1:
    return -1

  #-- Si el siguiente carácter es "\n", la expresion
  #-- tiene solo un digito: devolvermos su valor
  if cad[1]=="\n":
    return op1;  #-- PREGUNTA 5: Nivel de maxima profundidad

  #-- Si el segundo carácter NO es '+', se devuelve error
  if cad[1]!='+':
    return -1  #-- Error

  #-- Evaluar la expresion contenida en la subcadena desde el
  #-- tercer carácter hasta el final, llamando a evaluar(recursivo)
  op2 = evaluar(cad[2:])

  #-- si op2 es -1 ha habido un error
  if op2 == -1:
    return -1

  #-- Evaluar la suma del primer digito más el valor de la subcadena
  return op1 + op2
```

- *int procesar(void):* Es una función para **interactuar con el usuario**. No tiene ningún parametro de entrada. Al invocarla, pide una cadena al usuario. Si la cadena recibida es **nula** (El usuario ha apretado ENTER sin introducir nada) retornará, devolviendo el **código -2**. En caso contrario, llamará a la *función evaluar()* pasándole la cadena introducida por el usuario. Si se produce un **error en la evaluación**, se imprimirá un mensaje de error y se devolverá el **código -1**. Por último, si todo ha ido bien, se **imprimirá el resultado** de la evaluación, y se **devolverá**. Este es un ejemplo de la salida en consola al llamar a esta función:

  ```
  Introduce expresion a calcular: 1+1
  Resultado: 2
  ```

  El valor devuelto sería 2. Este es otro ejemplo:

  ```
  Introduce expresion a calcular: hola
  Error!
  ```

  El valor devuelto en este caso sería -1

- El **programa principal** imprimirá el mensaje: "EVALUACION DE EXPRESIONES" y entrará en un bucle donde se llamará a la *función procesar()*. Se llevará la cuenta de las expresiones que se han evaluado correctamente. Si alguna da error, no se contará. El bucle termina cuando la *función procesar()* devuelve el **código -2**. Al salir del bucle, se imprimirá el valor del contador de expresiones correctas. Por ejemplo:

  ```
  Evaluaciones OK: 4
  ```

Se pide:

1. Implementar la función atoi1*()* en el fichero **atoi1.s**. (2 puntos)

2. Implementar la función evaluar() en el fichero **evaluar.s** (3 puntos)

3. Implementar la función *procesar()* en el fichero **procesar.s** (2 puntos)

4. Implementar el **programa principal** en el fichero **main.s** (2 puntos)

5. Prueba tu programa con la cadena "1+3+2+1+0" y responde: ¿Qué valor tiene el **registro SP** cuando se ejecuta la función evaluar*()* en el nivel de mayor profundidad, antes de liberar la pila y retornar? (Es el punto indicado en los comentarios del algoritmo). Responde en los comentarios dentro del fichero evaluar.s (1 punto)

Este es un **ejemplo** de la **salida en la consola** cuando se ejecuta el programa completo. Se han introducido dos expresiones correctas, una incorrecta y finalmente se ha pulsado ENTER

```
EVALUACION DE EXPRESIONES

Introduce expresion a calcular: 0+0+0+1+2+3+0
Resultado: 6

Introduce expresion a calcular: hola
Error!

Introduce expresion a calcular: 9+1+5+1+0
Resultado: 16

Introduce expresion a calcular:

Evaluaciones ok: 2
-- program is finished running --
```