Práctica 1. Conversión de datos: cadenas de caracteres y enteros. Repaso del uso de la pila

En esta práctica efectuaremos la conversión que hace internamente el sistema tanto para convertir cadenas de caracteres a enteros (syscall *read-int* en MIPS, función *scanf* en C o el método *nextInt()* de la clase *Scanner* en Java), como para convertir enteros a cadenas de caracteres (syscall *print-int* en MIPS, función *printf* en C o *System.out.print()* en Java). Como veremos, estas conversiones, que nos parece tan naturales cuando llamamos a las mencionadas funciones del sistema, no son tan sencillas. Por último, utilizaremos estas dos funciones para repasar el correcto uso de la pila.

1. Función atoi

En primer lugar, el alumno deberá implementar la función `atoi`, la cual recibe como entrada el puntero a una cadena de caracteres ASCII en `$a0`, cuyo contenido es un número entero expresado en notación decimal, y devuelve un entero de 32 bits en complemento a 2 en `$v0`.

Las especificaciones de la función están descritas en la dirección web
`https://www.cplusplus.com/reference/cstdlib/atoi/`

Un problema que tiene este tipo de conversión es que la cadena de entrada puede ser errónea, ya se porque el número representado en decimal no quepa en el número de bits especificado, o porque comience con caracteres que no correspondan con dígitos decimales, '+', '-' o espacios. Por ello, la función devolverá en el registro `$v1` un código de error, con los siguientes valores:

0: Todo es correcto.

1: Carácter incorrecto: el comienzo de la cadena es un carácter no válido; o es un carácter válido ('+', '-' o espacios), pero después no se encuentra un número escrito en decimal.

2: Overflow: el número no se puede representar como un entero de 32 bits en complemento a 2, es decir, está fuera del rango $[-2^{31}, 2^{31} - 1]$.

Los códigos de error distintos de 0 invalidan el resultado devuelto en el registro `$v0`.

Ejemplos:

Cadena apuntada por `$a0`	Valor devuelto en `$v0`	Valor devuelto en `$v1`
`"123"`	`123`	`0`
`"+123"`	`123`	`0`
`"  -1"`	`-1`	`0`
`"-128ABC3"`	`-128`	`0`
`"A"`	Indiferente	`1`
`"+"`	Indiferente	`1`
`"-A"`	Indiferente	`1`
`"-+"`	Indiferente	`1`
`"2000000000000"`	Indiferente	`2`
`"-5000000000000"`	Indiferente	`2`

Donde el valor de `$v0` que se muestra en la tabla corresponde a la interpretación del contenido del registro en complemento a 2, es decir, tal y como se imprimiría en pantalla cuando se invoca a la `syscall` correspondiente.

2. Función itoa

En segundo lugar, el alumno deberá implementar la función `itoa`, que convierte un número entero a una cadena de caracteres, y cuyos parámetros son los siguientes:

- `$a0`: Número a convertir.
- `$a1`: Forma de interpretar el número contenido en `$a0` según el siguiente esquema:

  0: 32 bits en complemento a 2 (tipo de datos `int` en C o Java).

  1: 32 bits en binario natural (tipo de datos `unsigned int` en C o Java).

  2: 16 bits en complemento a 2 (tipo de datos `short` en C o Java).

  3: 16 bits en binario natural (tipo de datos `unsigned short` en C o Java).

  4: 8 bits en complemento a 2 (tipo de datos `char` en C o Java).

  5: 8 bits en binario natural (tipo de datos `unsigned char` en C o Java).

  Si en este parámetro se especifica un valor diferente de los anteriores se tomará como 0.

  Para conversiones de menos de 32 bits se tomarán los dígitos de orden más bajo del registro `$a0` ignorando el resto.

- `$a2`: Dirección de la cadena donde quedará el número anterior en decimal codificado en ASCII.

  La función debe convertir el número contenido en el registro `$a0`, interpretado según se especifique en el parámetro `$a1`, en una cadena codificada en ASCII que exprese el número en decimal. La cadena debe depositarse en la dirección contenida en el registro `$a2`, y debe acabar con el terminador de cadena, es decir un 0, y debe incluir el signo cuando el número representado sea negativo. Se pueden programar funciones auxiliares si fueran necesarias.

Ejemplos:

$a0	$a1	Cadena de salida apuntada por $a2
255	0	"255"
255	4	"-1"
1279	4	"-1"
1279	2	"1279"
1279	3	"1279"
-1	5	"255"
-1	0	"-1"
-1	1	"4294967295"

Donde el valor de `$a0` que se muestra en la tabla corresponde a la interpretación del contenido del registro en complemento a 2, es decir, tal y como habría que introducirlo desde el teclado cuando es solicitado por la llamada `syscall` correspondiente.

3. Repaso del uso de la pila

Por último, los alumnos deberán implementar una función llamada `pila_test` que realizará el siguiente trabajo:

1. Recibe como parámetros dos enteros de 32 bits en complemento a 2 en `$a0` y `$a1`.
2. La función transforma ambos enteros en cadena de caracteres con su representación en enteros de 16 bits en complemento a 2 mediante la función `itoa` (Nota: utilizar cadenas de caracteres declaradas de forma local para la función `pila_test`).

3. Tras esto, calcula la longitud de ambas cadenas invocando a la función `strlen`, la cual recibe como parámetro un puntero a cadena de caracteres en `$a0` y devuelve su longitud en `$v0`.
4. Finalmente, devuelve la suma de las longitudes en el registro `$v0`.

Téngase en cuenta que la función `strlen` **no debe ser implementada por los alumnos**, ya que será facilitada por la herramienta `Tablon`. Esta función, además de devolver la longitud de la cadena en `$v0`, reiniciará el contenido de todos los registros `$tX`, `$aX` y `$v1`. Por tanto, será necesario utilizar registros `$sX` para almacenar la información permanente y usar correctamente la pila para que `pila_test` funcione debidamente. También notar que, aunque la suma de la longitud de las cadenas podría realizarse de otras formas para evitar invocar funciones dentro de la propia función, no hacerlo a través de `itoa` y `strlen` conllevará la no evaluación del ejercicio.

4. Evaluación y entrega de la práctica

La práctica será evaluada con la herramienta `Tablon`, disponible en `http://tablon-aoc.infor.uva.es/`. Para ello, es necesario descargarse en cliente `http://tablon-aoc.infor.uva.es/client` y enviar el fichero `.asm` con la función a evaluar a la cola correspondiente, tal y como se explicará en los siguientes apartados. Se facilitará un usuario y contraseña a cada grupo de prácticas para acceder al `Tablon` durante la primera sesión presencial de prácticas.

Una vez enviado el código, `Tablon` realizará una serie de pruebas, testeando diferentes valores de entrada. Una vez finalizada la batería de pruebas, se podrá consultar el `Tablon` el número de pruebas superadas y el porcentaje de pruebas completado, además de otra información relevante, como el número de instrucciones ejecutadas, el número de líneas de código de la función y el número de excepciones en tiempo de ejecución. Estas métricas sirven para elaborar el ranking y desempatar dos grupos obtengan el mismo número de casos de prueba superados. Sin embargo, **la calificación de la práctica dependerá exclusivamente del número pruebas superadas**. El resto de métricas y el ranking no se tendrá en cuenta para la evaluación.

Finalmente, **para cada una de las funciones**, se debe entregar en el Campus Virtual el fichero `.asm` que obtuviera el mejor resultado en el ranking de la herramienta `Tablon`. El nombre del fichero para Campus Virtual será el número de *request* del `Tablon` con 5 cifras y la extensión `.asm`. **Se habilitará una entrega diferente en el Campus Virtual para cada función.**

Fecha límite de entrega en el Campus Virtual: 13 de octubre a las 23:55 h.

No se evaluarán las entregas del `Tablon` no depositadas dentro del plazo en el Campus Virtual.

Evaluación función atoi

Para poder evaluar la práctica, el fichero `.asm` enviado al `Tablon` debe cumplir los siguientes requisitos:

- Debe contener únicamente la función `atoi` implementada por el grupo y, si fuera necesario, las funciones que esta use.
- La función debe llamarse `atoi` y cumplir con las especificaciones detalladas en el apartado 1.

Para testear la función en `Tablon`, se dispone de la cola de ejecución `mars_atoi`. Se pueden enviar trabajos con el siguiente comando:

```
python ./client -u mi_usuario -x mi_password -q mars_atoi mi_fichero.asm -- ARG
```

donde ARG es la cadena que representa en decimal el número que se quiere convertir, y se obtendrá una salida similar a esta (suponiendo que $ARG = -234$ y asumiendo que el comando se lanzó correctamente):

```
Code lines: 43
Result: -234
Magic Number: 128
Instructions executed: 115
```

que nos indica el número de líneas de código de nuestro fichero (`Code lines`), el número de instrucciones ejecutadas (`Instructions executed`) y el resultado de la función (`Result`)[1]. El campo `Result` dependerá de la ejecución de nuestra función `atoi`, siendo los posibles resultados:

- *Un número entero en notación decimal*: la función `atoi` ha devuelto el código de error 0 en `$v1` y se imprime el valor devuelto en el registro `$v0`. Nótese que el valor mostrado puede ser incorrecto si `atoi` no ha sido correctamente implementada.
- *ERROR CARACTER*: la función `atoi` ha devuelto el código de error 1.
- *ERROR OVERFLOW*: la función `atoi` ha devuelto el código de error 2.
- *ERROR OTROS*: la función `atoi` ha devuelto un código de error desconocido (`$v1` $\notin \{0, 1, 2\}$).
- *ERROR CONTEXTO. Se ha modificado el registro* `$sX`: la función `atoi` ha modificado el contexto del procesador al modificar el registro permanente `$sX`.
- *Error in line Y: Runtime exception at 0xXXXXXXXX: arithmetic overflow. Processing terminated due to errors*: la función `atoi` ha provocado una excepción en el procesador, abortando la ejecución del programa. En este ejemplo, el error se debe a un overflow aritmético, pero el error variará según el programa. Nótese que `Tablon` concatena en un único fichero el programa principal con la función del grupo, por lo que el número de línea Y que provoca el error en `Tablon` será diferente del número de línea que causa el error en el fichero del grupo. Se puede obtener el número de línea que causó el error restando $Y - 110$.

La cola para evaluar la función `atoi` en `Tablon` es `lb_atoi`. Se pueden enviar trabajos con el siguiente comando:

```
python ./client -u mi_usuario -x mi_password -q lb_atoi mi_fichero.asm
```

Notar que en este caso, no es es necesario incluir argumentos adicionales al lanzar a ejecución.

La batería de pruebas evaluada, compuesta por 25 casos, está formada por las siguientes pruebas, clasificada en función del valor código de error:

- **Error 0**: `0`, `5`, `-9`, `15`, `-19`, `2222222`, `-2000000000`, `-2147483647`, `2147483647`, `12048001abcd`, `-120481230abcd`, `+12456abcd`, `␣␣␣12048abcd`, `␣␣␣-12048abcd`, `␣␣␣+12456abcd`, mas un caso secreto.
- **Error 1**: `+-`, `␣␣␣aaaa123`, `+aaaa123`, `-aaaa123`.
- **Error 2**: `-2147483649`, `2147483648`, `5000000000`, `3000000000`, mas un caso secreto.

Evaluación función itoa

Para poder evaluar la práctica, el fichero `.asm` enviado al `Tablon` debe cumplir los siguientes requisitos:

- Debe contener únicamente la función `itoa` implementada por el grupo y, si fuera necesario, las funciones que esta use.
- La función debe llamarse `itoa` y cumplir con las especificaciones detalladas en el apartado 2.

Para testear la función en `Tablon`, se dispone de la cola de ejecución `mars_itoa`. Se pueden enviar trabajos con el siguiente comando:

```
python ./client -u mi_usuario -x mi_password -q mars_itoa mi_fichero.asm -- ARG1# ARG2
```

donde $ARG1$ es el número que se quiere convertir a cadena y $ARG2$ es el tipo de datos para interpretarlo, es decir, son los parámetros `$a0` y `$a1` de la función `itoa`, y se obtendrá una salida similar a esta (suponiendo que $ARG1 = -234$ y asumiendo que el comando se lanzó correctamente):

```
Code lines: 43
Result: -234
Magic Number: 128
Instructions executed: 115
```

[1] Podemos ignorar el campo `Magic Number`. Será de utilidad en la práctica 2, pero no tendrá ningún propósito en la actual práctica

que nos indica el número de líneas de código de nuestro fichero (`Code lines`), el número de instrucciones ejecutadas (`Instructions executed`) y el resultado de la función (`Result`). El campo `Result` dependerá de la ejecución de nuestra función `itoa`, siendo los posibles resultados:

- *Un número entero en notación decimal*: la función `itoa` ha convertido el entero a cadena de caracteres y se muestra el contenido de la cadena. Nótese que el valor mostrado puede ser incorrecto si `itoa` no ha sido correctamente implementada.
- *ERROR CONTEXTO. Se ha modificado el registro* `$sX`: la función `itoa` ha modificado el contexto del procesador al modificar el registro permanente `$sX`.
- *Error in line Y: Runtime exception at 0xXXXXXXXX: arithmetic overflow. Processing terminated due to errors*: la función `itoa` ha provocado una excepción en el procesador, abortando la ejecución del programa. En este ejemplo, el error se debe a un overflow aritmético, pero el error variará según el programa. Nótese que `Tablon` concatena en un único fichero el programa principal con la función del grupo, por lo que el número de línea Y que provoca el error en `Tablon` será diferente del número de línea que causa el error en el fichero del grupo. Se puede obtener el número de línea que causó el error restando $Y - 93$.
- *XXXXXXXXXXX—— Si estas leyendo este mensaje, o bien no has modificado el string de salida, o bien la cadena no termina con el caracter fin de cadena*: Como indica el mensaje, no se ha incluido el carácter fin de cadena '$\backslash 0$' ($XXXXX$ será la cadena generada por `itoa`), o no se ha modificado la cadena pasada por parámetro en `$a2` (en este caso, $XXXXXX$ serán todo guiones).

La cola para evaluar la función `itoa` en `Tablon` es `lb_itoa`. Se pueden enviar trabajos con el siguiente comando:

```
python ./client -u mi_usuario -x mi_password -q lb_itoa mi_fichero.asm
```

Notar que en este caso, no es es necesario incluir argumentos adicionales al lanzar a ejecución.

La batería de pruebas evaluada, compuesta por 28 casos, está formada por las siguientes pruebas, clasificada en función del tipo de dato a codificar, es el siguiente:

- **Tipo 0**: `0`, `5`, `15`, `-9`, `-19`, `315123`, `-334319`, `-2147483648`, `-2147483647`
- **Tipo 1**: `-1000000000`, `-1`, `-2147483648`
- **Tipo 2**: `2147483647`, `32768`, `32767`, `-2147483648`
- **Tipo 3**: `2147483647`, `32768`, `65535`, `-2147483648`
- **Tipo 4**: `2147483647`, `128`, `127`, `-2147483648`
- **Tipo 5**: `2147483647`, `128`, `255`, `-2147483648`

Evaluación función pila_test

Para evaluar la función `pila_test`, disponemos de las siguientes colas de ejecución en la herramienta `Tablon`:

- Cola de pruebas: `pila`. Debe recibir como argumentos una cadena caracteres que será el parámetro de entrada `$a0` de `pila_test`. Se pueden obtener los siguientes resultados:
  - *La suma de las longitudes es X*: la función `pila_test` ha devuelto el resultado de la función, siendo X el resultado contenido en `$v0`. Nótese que el valor mostrado puede ser incorrecto si las funciones `pila_test` o `itoa` no ha sido correctamente implementadas. Por ejemplo, supongamos que los parámetros de entrada son `$a0`=10000 y `$a1`=131071.
    - La cadena generada por `$a0` es "10000", mientras que la cadena generada por `$a1` es "-1".
    - La longitud de la cadena generada por `$a0` es 5, mientras que la longitud de la cadena generada por `$a1` es 2.
    - Por tanto, `pila_test` debe retornar un 7 en `$v0`.
  - *ERROR CONTEXTO. Se ha modificado el registro* `$sX`: la función `pila_test` ha modificado el contexto del procesador al modificar el registro permanente `$sX`.

- *Error in line Y: Runtime exception at 0xXXXXXXXX: arithmetic overflow. Processing terminated due to errors*: la función `pila_test` ha provocado una excepción en el procesador, abortando la ejecución del programa. En este ejemplo, el error se debe a un overflow aritmético, pero el error variará según el programa. Nótese que `Tablon` concatena en un único fichero el programa principal con la función del grupo, por lo que el número de línea Y que provoca el error en `Tablon` será diferente del número de línea que causa el error en el fichero del grupo. Se puede obtener el número de línea que causó el error restando $Y - 134$.

- Cola de evaluación: `lb_pila`. No requiere argumentos de entrada. Solo ejecuta un caso de prueba, y nos permitirá comprobar si la función `pila_test` funciona correctamente.

Notar que, para la correcta evaluación por parte de `Tablon`, se debe enviar un fichero `.asm` que incluya las funciones `pila_test` e `itoa` implementadas por el grupo, y no debe incluir la función `strlen`.