ESTRUCTURA DE COMPUTADORES
Grado en Ingeniería Informática

Sesión de laboratorio número 2

LLAMADAS AL SISTEMA

Objetivos

- Consolidar los conocimientos sobre la codificación ASCII
- Hacer inventario de instrucciones de bifurcación del MIPS.
- Utilizar las instrucciones de bifurcación para implementar bucles y condicionales en ensamblador
- Entender los tres materiales con que se hace un programa en ensamblador (instrucciones, datos y llamadas al sistema).
- Conocer y hacer uso de las llamadas al sistema por medio de la instrucción máquina `syscall`.

Material

- El simulador *pcspim-ES*.
- Archivos fuente en ensamblador *forever.s*, *ascii-console.s, ascii-for, echo.s*.

Bibliografía

- D.A. Patterson y J. L. Hennessy, *Estructura y diseño de computadores*, Reverté, capítulo 2, 2011.

Introducción teórica

El código ASCII

El estándar Unicode actual, capaz de codificar textos de multitud de lenguas desde 1991, tiene un precedente en el estándar americano ASCII (*American Standard Code for Information Interchange*) definido en 1963. El estándar ASCII evolucionó posteriormente para ajustarse mejor a las necesidades de almacenamiento y comunicaciones digitales cambiantes.

Entre las características de las primeras versiones de ASCII conviene destacar:

- Codificaba los caracteres en 7 bits, para poder añadir un bit adicional de paridad que completara los 8 bits que se almacenaban o se transmitían.
- De los 128 códigos posibles, reservaba los 32 primeros (del 0 al 31) y el último (127) para control, sin representar ningún carácter gráfico.
- Los 95 códigos restantes representaban letras, números y signos de puntuación, propios del alfabeto anglosajón. No consideraba, por tanto, la Ç, la Ñ ni las vocales acentuadas.

- Siguiendo el orden alfabético, las letras mayúsculas tienen códigos consecutivos. Igualmente pasa con las minúsculas. Así, ascii('B') = ascii('A') + 1; ascii('d') = ascii('a') + 3
- Los dígitos '0' al '9' también tienen códigos consecutivos. Por lo tanto, ascii('7') = ascii('0') + 7.

	_0	_1	_2	_3	_4	_5	_6	_7	_8	_9	_A	_B	_C	_D	_E	_F
0_	*NUL* 00 0	*SOH* 01 1	*STX* 02 2	*ETX* 03 3	*EOT* 04 4	*ENQ* 05 5	*ACK* 06 6	*BEL* 07 7	*BS* 08 8	*HT* 09 9	*LF* 0A 10	*VT* 0B 11	*FF* 0C 12	*CR* 0D 13	*SO* 0E 14	*SI* 0F 15
1_	*DLE* 10 16	*DC1* 11 17	*DC2* 12 18	*DC3* 13 19	*DC4* 14 20	*NAK* 15 21	*SYN* 16 22	*ETB* 17 23	*CAN* 18 24	*EM* 19 25	*SUB* 1A 26	*ESC* 1B 27	*FS* 1C 28	*GS* 1D 29	*RS* 1E 30	*US* 1F 31
2_	**(SP)** 20 32	**!** 21 33	**"** 22 34	**#** 23 35	**$** 24 36	**%** 25 37	**&** 26 38	**'** 27 39	**(** 28 40	**)** 29 41	*** 2A 42	**+** 2B 43	**,** 2C 44	**-** 2D 45	**.** 2E 46	**/** 2F 47
3_	**0** 30 48	**1** 31 49	**2** 32 50	**3** 33 51	**4** 34 52	**5** 35 53	**6** 36 54	**7** 37 55	**8** 38 56	**9** 39 57	**:** 3A 58	**;** 3B 59	**<** 3C 60	**=** 3D 61	**>** 3E 62	**?** 3F 63
4_	**@** 40 64	**A** 41 65	**B** 42 66	**C** 43 67	**D** 44 68	**E** 45 69	**F** 46 70	**G** 47 71	**H** 48 72	**I** 49 73	**J** 4A 74	**K** 4B 75	**L** 4C 76	**M** 4D 77	**N** 4E 78	**O** 4F 79
5_	**P** 50 80	**Q** 51 81	**R** 52 82	**S** 53 83	**T** 54 84	**U** 55 85	**V** 56 86	**W** 57 87	**X** 58 88	**Y** 59 89	**Z** 5A 90	**[** 5B 91	**** 5C 92	**]** 5D 93	**^** 5E 94	**_** 5F 95
6_	**`** 60 96	**a** 61 97	**b** 62 98	**c** 63 99	**d** 64 100	**e** 65 101	**f** 66 102	**g** 67 103	**h** 68 104	**i** 69 105	**j** 6A 106	**k** 6B 107	**l** 6C 108	**m** 6D 109	**n** 6E 110	**o** 6F 111
7_	**p** 70 112	**q** 71 113	**r** 72 114	**s** 73 115	**t** 74 116	**u** 75 117	**v** 76 118	**w** 77 119	**x** 78 120	**y** 79 121	**z** 7A 122	**{** 7B 123	**|** 7C 124	**}** 7D 125	**~** 7E 126	*DEL* 7F 127

Tabla 1. Código ASCII de 7 bits. Las 33 celdas sombreadas corresponden a caracteres de control no imprimibles. (**SP**) denota el espacio entre palabras.

Posteriormente, ASCII se extendió a 8 bits y se estandarizó como ISO/IEC_8859. Los 128 nuevos códigos incluyen 32 caracteres de control; los 96 restantes representan letras y signos de puntuación. El nuevo estándar definió diferentes partes o variantes regionales, y por eso en Europa Occidental se utiliza la parte IEC_8859-1, también llamada *latin1*. Vea en el apéndice la codificación completa de esta parte. Actualmente está integrada en el estándar Unicode.

Las llamadas al sistema en *pcspim-ES*

Los computadores disponen de un sistema operativo que ofrece un catálogo de *system calls* o *system functions*. Con ellas, los procesos pueden acceder de manera segura y eficiente a los recursos compartidos del computador: el procesador, la memoria principal y los periféricos. En el bloque de la asignatura referente a la entrada/salida, estudiaremos algunos detalles de implementación.

El simulador dispone de dos periféricos de texto: el teclado y la consola. Ambos utilizan el estándar ISO/IEC 8859-1 para codificar los caracteres. El teclado, además de los códigos

alfanuméricos, genera códigos de control combinando la tecla *ctrl* con las teclas alfabéticas. El simulador interpreta directamente las teclas de cursor y el *ctrl-C* y por eso los programas simulados no pueden leerlas.

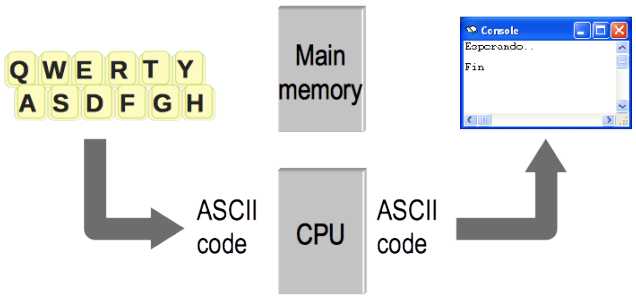


Figura 1. PCSPIM modela un computador con un teclado que produce códigos ASCII de 8 bits y una consola que los consume.

En un MIPS, las funciones de sistema se invocan mediante la instrucción **`syscall`**. Cada función se distingue por un índice único que la identifica, acepta una serie de argumentos (dependiendo del índice) y devuelve un posible resultado.

El catálogo de funciones del sistema simulado en PCSpim se encuentra en el apéndice de este enunciado. En esta práctica debemos trabajar sólo con las cinco funciones referidas en la tabla siguiente. Note que el índice que las identifica se coloca en el registro **`$v0`**, que algunas llamadas toman el parámetro contenido en el registro **`$a0`** y que las llamadas que devuelven un resultado lo hacen en **`$v0`**:

Nombre	$v0	Descripción	Argumentos	Resultado
print_int	*1*	Imprime el valor de un entero	`$a0` = entero a imprimir	—
read_int	*5*	Lee el valor de un entero	—	`$v0` = entero leído
exit	*10*	Acaba el proceso	—	—
print_char	*11*	Imprime un carácter	`$a0` = carácter a imprimir	—
read_char	*12*	Lee un carácter	—	`$v0` = carácter leído

Tabla 2. Funciones del sistema que deben utilizarse en esta práctica.

A continuación, se ilustra con un ejemplo el mecanismo de llamada. El código lee un número entero desde el teclado y lo copia en la dirección de memoria etiquetada con el nombre **`valor`**:

```
li $v0, 5          # Índice de la llamada read_int
syscall            # Llamada al sistema read_int
sw $v0, valor      # Copia el número entero en memoria
```

Algunos aspectos de las funciones de entrada/salida

Las funciones *print_char* y *read_char* no realizan ningún cambio de formato. Esto es, *print_char* imprime en la consola el código que recibe en **`$a0`** y *read_char* devuelve en **`$v0`** el código generado por el teclado. En cambio, *print_int* transforma el entero que recibe en **`$a0`** en la cadena correspondiente de caracteres (codificados en ASCII) legible por los humanos. Igualmente, *read_int* procesa una cadena de caracteres tecleada por un humano y calcula el valor entero que devuelve en **`$v0`**.

Otro detalle es el *eco*. La función *read_int*, además de leer del teclado, escribe en la consola los caracteres leídos, creando la ilusión de que el usuario escribe en la pantalla. La función *read_char*, en *pcspim-ES*, no genera ningún eco.

Control de flujo de ejecución en ensamblador del MIPS

Las instrucciones de salto, junto a ciertas instrucciones aritméticas, permiten construir las estructuras condicionales e iterativas.

A bajo nivel, podemos distinguir entre:

- saltos incondicionales del tipo *seguir en la dirección*, donde *dirección* señala la instrucción que se ejecutará a continuación: El MIPS dispone de la instrucción `j eti`.
- saltos condicionales, también llamados bifurcaciones, del tipo *si (condición) seguir en la dirección* donde *dirección* señala la instrucción que se ejecutaría en el caso de que la condición se cumpla. En el juego del MIPS, tenemos seis condiciones de salto; nótese que se pueden hacer tres parejas de condiciones contrarias (= y ≠, > y ≤, < y ≥).

El juego de instrucciones sólo permite las comparaciones = y ≠ entre dos registros y las comparaciones >, ≤, < y ≥ entre un registro y el cero:

`beq rs,rt,A`	`bgtz rs,A`	`bltz rs,A`
$rs = rt$	$rs > 0$	$rs < 0$
`bne rs,rt,A`	`blez rs,A`	`bgez rs,A`
$rs \neq rt$	$rs \leq 0$	$rs \geq 0$

Tabla 3. Instrucciones de bifurcación del MIPS

Este surtido de condiciones puede ampliarse con ayuda de la instrucción aritmética `slt` (*set on less than*) y las instrucciones relacionadas que se estudiarán en el tema de aritmética de enteros. Así se obtienen estas otras seis pseudoinstrucciones:

`beqz rs,A`	`bgt rs,rt,A`	`blt rs,rt,A`
$rs = 0$	$rs > rt$	$rs < rt$
`bnez rs,A`	`ble rs,rt,A`	`bge rs,rt,A`
$rs \neq 0$	$rs \leq rt$	$rs \geq rt$

Tabla 4. Pseudoinstrucciones de bifurcación del MIPS

Veamos la traducción de un par de pseudoinstrucciones de salto condicional en instrucciones máquina en la tabla siguiente:

Pseudoinstrucción	**Instrucciones máquina**
`beqz rs,A`	`beq rs,$zero,A`
`bgt rs,rt,A`	`slt $at,rt,rs` `bne $at,$zero,A`

Tabla 5. Traducción de las pseudoinstrucciones `beqz` y `bgt` en instrucciones del MIPS

Finalmente, hay una pseudoinstrucción de salto incondicional (*b*). El ensamblador la traduce en una instrucción de bifurcación en que la condición de salto se cumple siempre (por ejemplo: `b eti` se puede traducir como `beq $0,$0,eti`).

`b A`
true

Tabla 6. Pseudoinstrucción de salto incondicional del MIPS

Con estas instrucciones pueden construirse estructuras condicionales e iterativas equivalentes a las que escritas en alto nivel. Por ejemplo, si hay un bloque de instrucciones A1, A2… que sólo han de ejecutarse si el contenido de un registro `$r` es negativo, se puede utilizar una bifurcación que salte si se da la condición contraria (`$r ≥ 0`):

```
            bgez $r,L
            A1
            A2
            …
L:
```

Para iterar *n* veces un bloque de instrucciones A1, A2…, se puede utilizar un registro `$r` y escribir:

```
            li $r,n
loop:       A1
            A2
            …
            addi $r,$r,-1
            bgtz $r,loop
```

En el anexo puede consultarse un cuadro con la traducción de diversas estructuras de control de flujo.

Ejercicios de laboratorio

Configuración del simulador *pcspim-ES*

Al iniciar el simulador *pcspim-ES*, compruebe que la configuración definida en *Simulator->Settings*... coincide con la mostrada en la figura.

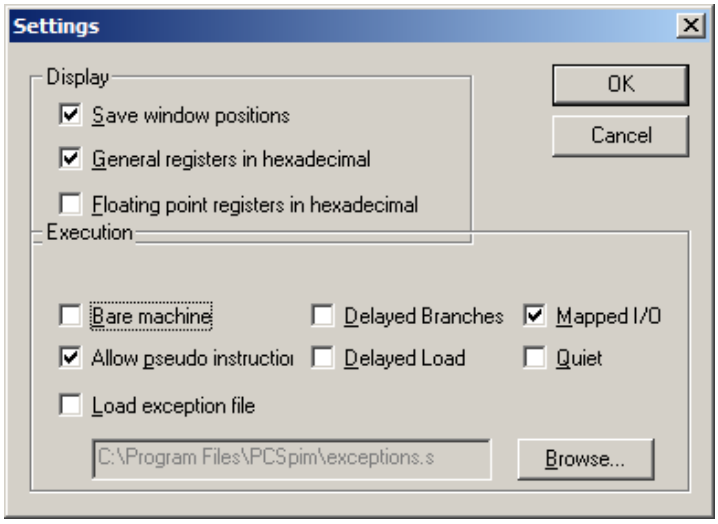


Figura 2. Configuración de *pcspim-ES* para esta práctica.

Ejercicio 1: El bucle infinito

Observe el código contenido en el fichero *forever.s* (figura 3). Note que sólo especifica el segmento de instrucciones (`.text`). Antes de simularlo, pruebe a entenderlo. El programa se limita a leer enteros de la consola y sumarlos.

- Busque en el código las cuatro llamadas al sistema utilizadas en el programa y consulte en la tabla qué hacen y cómo se utilizan. Note (1) el uso de `$v0` para seleccionar la función en todos los casos, (2) el uso de `$v0` para recoger el valor leído por *read_int* y (3) el uso de `$a0` para el argumento de *print_int()* y *print_char()*.
- Observe que hay un bucle. ¿Cuál es la primera instrucción del bucle? ¿Y la última?

```
          .globl __start
          .text 0x00400000

__start:
          li $s0, 0
loop:
          li $v0,5
          syscall
          addu $s0,$s0,$v0
          li $v0,1
          move $a0,$s0
          syscall
          li $v0,11
          li $a0,10
          syscall
          b loop

          li $v0,10
          syscall
```

```
$s0 = 0;
do {
     $v0 = read_int();
     $s0 = $s0 + $v0;
     print_int($s0);
     print_char('\n');
} forever;
exit();
```

Figura 3. Código fuente y pseudocódigo de *forever.s*, el primer ejercicio de la práctica. Observe que el bucle es infinito y que el flujo de ejecución nunca llega a las dos últimas instrucciones (`li`, `syscall`).

- ¿Qué hace cada iteración del bucle? ¿Podría explicar cada instrucción y cada pseudo-instrucción que lo compone?

Cargue ahora el programa en el simulador.

- ¿Cómo se ha traducido la línea `b loop`?
- ¿Sabe ejecutar el programa completo? Hágalo (orden *Go*, tecla F5). Mientras se ejecuta, mantenga activa la ventana *Console*. Tenga en cuenta que el programa espera entrada de números por el teclado sin escribir en la consola ningún texto que lo indique. Teclee valores numéricos con signo.

  > **Técnica experimental: el *ctrl-C***. Cuando la ventana de la consola está activa, *ctrl-C* detiene el programa en el punto en que se encuentra, igual que en la consola de Unix. Tiene el mismo efecto que *Simulator>Break*.

- Detenga la ejecución del código. Verá un mensaje que dice *"Execution paused by the user at* <dirección> *Continue execution?"*; anote la dirección y pique sobre el botón *No*. Observe ahora la ventana principal del simulador (Figura 4) y busque en ella la solución a las siguientes cuestiones:

  1. ¿Cuál es la última instrucción que se ha ejecutado? Recuerde la dirección anotada y busque la instrucción en la ventana de instrucciones.

  2. ¿Cuál será la instrucción que se iba a ejecutar en ese momento? Consulte el valor del PC actual (en la ventana del procesador) y busque la instrucción correspondiente.

  3. ¿Qué contienen los registros `$a0`, `$v0` y `$s0`? Busque el valor en la ventana del procesador. ¿Sabe cambiar la base de numeración de decimal a hexadecimal? (cuadro de configuración *Simulator>Settings>Display*).

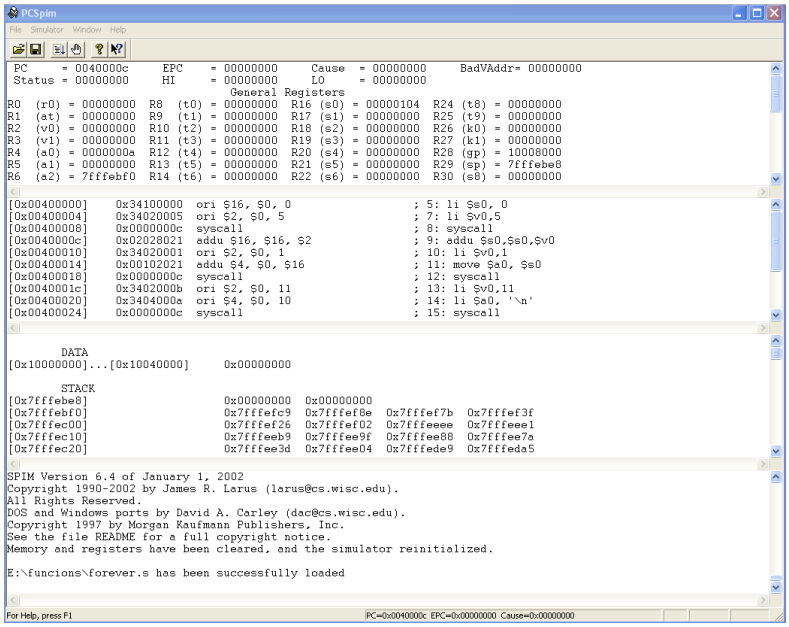


Figura 4. La interfaz del simulador muestra cuatro secciones. De arriba abajo: (1) el estado del procesador, con el contenido de sus registros más significativos, (2) la memoria de instrucciones, (3) la memoria de datos y (4) los avisos del simulador.

- Pruebe a proseguir la ejecución. Notará que a la orden *Go* (tecla F5), el simulador propone como *Starting Address* la dirección de la próxima instrucción a ejecutar; es decir, el valor actual del PC.
- Cuando tenga claro el funcionamiento del código, pase al ejercicio siguiente.

Ejercicio 2: Limitando el bucle

En este ejercicio ha de conseguir que el bucle se detenga cuando el usuario teclee un cero. La detención se consigue añadiendo al bucle un salto condicional a las instrucciones que implementan la llamada *exit()*. Parta del código de *forever* y guarde el código modificado en el archivo *"break.s"*:

```
$s0 = 0;
do {
      $v0 = read_int();
      if ($v0=0) break; ←
      $s0 = $s0 + $v0;
      print_int($s0);
      print_char('\n');
} forever;
exit();
```

Figura 5. Pseudocódigo de *break.s* (ejercicio 2), resultante de añadir la línea marcada con ← al pseudocódigo de *forever.s* (ejercicio 1).

- Ha de añadir una etiqueta al final del bucle y saltar a ella si *$v0 = 0*. Escoja un nombre apropiado para la etiqueta (**`fin`**, **`salida`**, etc). ¿Tiene claro dónde colocarla?
- Añada la instrucción de salto. ¿Tiene claro cuál?
- Compruebe con el simulador que el código es correcto y que el programa se detiene al teclear un 0.
- Cuando acaba la ejecución, ¿qué valen los registros **`$v0`** y **`$s0`**? ¿Y el PC?

Ejercicio 3: El bucle y el contador

Ha de mejorar la comunicación de *break.s* con el usuario. Tiene que contar el número de sumandos conforme avanza e imprimir la suma y el número de sumandos al final. Se trata de conseguir el siguiente diálogo por pantalla

```
$s0 = 0;
$s1 = 0
do {
     print_int($s1+1);
     print_char('>');
     $v0 = read_int();
     if ($v0=0) break;
     $s0 = $s0 + $v0;
     $s1 = $s1 + 1;
} forever;
print_char('=');
print_int($s0);
print_char('\n');
print_char('n');
print_char('=');
print_int($s1);
exit();
```

```
1>89
2>-230
3>67
4>0
=-74
n=3
```

Figura 6. Pseudocódigo y ejemplo de ejecución de *counter.s*. El programa lee enteros hasta que se teclee un 0 y mantiene en $s1 la cuenta de sumandos leídos. Hasta el final no muestra la suma acumulada.

- Parta del código de *break.s*, modifíquelo y guarde el nuevo código en *counter.s*.
- Añada las instrucciones que inicializan, incrementan e imprimen el valor de **`$s1`**.
- Saque fuera del bucle las instrucciones que imprimen el valor de **`$s0`** y complete el código final.
- Pruebe el código resultante.

Ejercicio 4. Códigos ASCII en la consola

Considere el bucle que en alto nivel se expresa como

```
for ($s0=32; $s0<127; $s0=$s0+1) {…}
```

En ensamblador y pseudocódigo se puede expresar así:

```
        li $s0,32
        li $s1,127
loop:
        .......

        addi $s0,$s0,1
        blt $s0,$s1,loop
```

```
$s0=32;
do { …
     …
     $s0 = $s0+1;
}
while ($s0<127);
```

Figura 7. El bucle *for* del ejercicio 4 utiliza un contador (**`$s0`**) que se incrementa al final de cada iteración y un registro para el valor límite (**`$s1=127`**). En este caso, la guarda es simple y se traduce en una única instrucción de salto condicional.

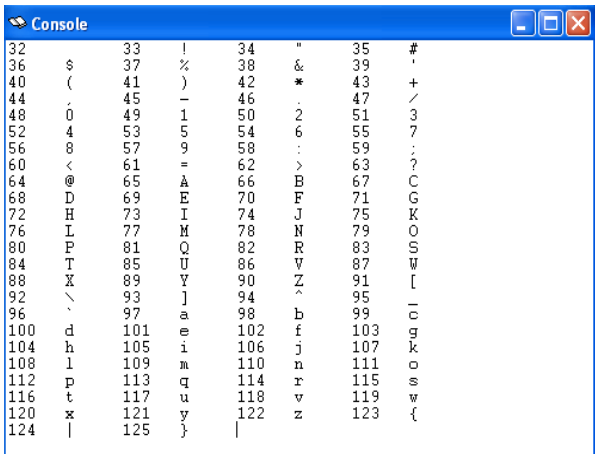
El código *ascii-console.s* imprime los caracteres gráficos del código ASCII clásico de 7 bits. En el bucle se omiten los códigos del 0 al 31 (sección C0 del estándar), el código 127 (**`DEL`**) y los caracteres extendidos del 128 al 255.

```
        li $s0,32
        li $s1,127
loop:
        li $v0,1
        move $a0,$s0
        syscall
        li $v0,11
        li $a0,9
        syscall
        li $v0,11
        move $a0,$s0
        syscall
        li $v0,11
        li $a0,10
        syscall

        addi $s0,$s0,1
        blt $s0,$s1,loop

        li $v0,10
        syscall
```

```
for ($s0=32; $s0<127; $s0=$s0+1){
      print_int($s0);
      print_char('\t'); // Tabulador
      print_char($s0);
      print_char('\n'); // Salto de línea
      }
exit();
```

Figura 8. Código, pseudocódigo y salida por pantalla de *ascii-console.s*

- Abra el archivo *ascii-console.s* con un editor de textos y observe la estructura *for* que contiene. Compruebe el funcionamiento del programa con el simulador.
- Observe el uso de los caracteres de control '\t' (código 9) y '\n' (código 10).
- ¿Cuál es el conjunto de caracteres representables por la consola? Cambie los límites del bucle para probar los códigos del 0 al 255. Ejecute el código y observe el resultado. Cuando la consola no puede imprimir un carácter, muestra ▌.
- También por probar, modifique *ascii-console.s* para que liste los caracteres en orden decreciente, del 126 al 32. Compruebe el código con el simulador.
- Modifique el código de *ascii-console.s* (y guárdelo como *ascii-console-tab.s*) para que tabule los códigos del 32 al 126 en la consola de la siguiente manera:




```
$s2=4;
for ($s0=32; $s0<127; $s0=$s0+1){
      $s2 = $s2-1;
      print_int($s0);
      print_char('\t');
      print_char($s0);
      if ($s2==0)
            $s2 = 4;
            print_char('\n');
      else
            print_char('\t');
}
exit();
```

Figura 9. Salida y pseudocódigo de *ascii-console.s*

- Compruebe el código *ascii-console-tab.s*.

Cuestiones diversas

Se trata de cuestiones de lápiz y papel, pero en algunos casos puede comprobarlas con el simulador.

1. ¿Cuál será la salida del código siguiente *(ascii-for.s)*?

```
        li $s0,'a'
        li $s1,10
loop:
        li $v0,11
        move $a0,$s0
        syscall

        addi $s0,$s0,1
        addi $s1,$s1,-1
        bgtz $s1,loop
```

   ¿Y si cambiara la línea `li $v0,11` por `li $v0,1`?
   ¿Y si cambiara la línea `addi $s0,$s0,1` por `addi $s0,$s0,-1`?
   ¿Y si cambiara la línea `addi $s1,$s1,-1` por `addi $s1,$s1,-2`?

2. En el texto siguiente (*echo.s*), sustituya cada ***`bif`*** por una instrucción de bifurcación para que el código lea reiteradamente del teclado, haga eco de los caracteres leídos si son cifras y acabe cuando se pulsa la letra 'f'

```
__start:
      li $s0,'0'
      li $s1,'9'
      li $s2,'f'
loop:
      li $v0,12
      syscall
      bif $v0,$s2,exit
      bif $v0,$s0,loop
      bif $v0,$s1,loop
      move $a0,$v0
      li $v0,11
      syscall
      b loop
exit:
      li $v0,10
      syscall
```

3. Si se precisara una pseudoinstrucción `ca2 rt,rs` que hiciera la operación *rt = complemento_a_2(rs)*, ¿cómo se traduciría? ¿Hay alguna pseudoinstrucción estándar del MIPS equivalente a `ca2`?
4. Con ayuda del simulador, pruebe a cargar código donde aparezca la pseudoinstrucción `li $1,20` o `li $at,20`. ¿Qué dice el simulador?
5. ¿Puede explicar la diferencia entre las llamadas print_char(100) y print_integer(100)?
6. ¿Y la diferencia entre print_char('A') y print_integer('A')?
7. En la Tabla 5 tiene la traducción de dos de las seis pseudoinstrucciones de Tabla 4. ¿Cuál es la traducción de las cuatro que faltan?

Anexo

Ejemplos de control de flujo

En la tabla siguiente,

- El símbolo *bif (cond)* denota una bifurcación que salta si la condición cond se cumpe. Los símbolos *cond*, *cond1*, etc., hacen referencia a las seis condiciones simples (= y ≠, > y ≤, < y ≥) que relacionan dos valores. El asterisco indica condición contraria; por ejemplo, si *cond* = "**>**" tenemos *cond** = "≤".
- En la columna de alto nivel, los símbolos *A*, *B*, etc. indican sentencias simples o compuestas; en la columna de bajo nivel, los símbolos **A**, **B**, etc. representan los bloques de instrucciones equivalentes en ensamblador.

Condicionales.

<table>
<tr><th>Alto nivel</th><th>Ensamblador</th></tr>
<tr><td>

```
if (cond1)
      A;
else if (cond2)
      B;
else
      C;
D;
```

</td><td>

```
if:       bif (cond1*) elseif
          A
          j endif
elseif:   bif (cond2*) else
          B
          j endif
else:     C
endif:    D
```

```
if:       bif (cond1) then
          bif (cond2) elseif
          j else
then:     A
          j endif
elseif:   B
          j endif
else:     C
endif:    D
```

</td></tr>
<tr><td>

```
if (cond1 && cond2)
      A;
B;
```

</td><td>

```
if:       bif (cond1*) endif
          bif (cond2*) endif
          A
endif:    B
```

</td></tr>
<tr><td>

```
if (cond1||cond2)
      A;
B;
```

</td><td>

```
if:       bif (cond1) then
          bif (cond2*) endif
then:     A
endif:    B
```

```
if:       bif (cond1*) endif
          bif (cond2*) endif
          A
endif:    B
```

</td></tr>
</table>

Selectores

Alto nivel	Ensamblador

switch (*exp*){
case X :
  A;
  break;
case Y :
case Z :
  B;
  break;
default:
  C;
}
D;
	** bif (exp != X) caseY**
	caseX: A
	** j endSwitch**
	caseY: bif (exp != Y) default
	caseZ: bif (exp != Z) default
	** B**
	** j endSwitch**
	default: C
	endSwitch: D
	
	** bif (exp == X) caseX**
	** bif (exp == Y) caseY**
	** bif (exp == Z) caseZ**
	** j default**
	caseX: A
	** j endSwitch**
	caseY:
	caseZ: B
	** j endSwitch**
	default: C
	endSwitch: D

Iteraciones

Alto nivel	Ensamblador
while (*cond*) A; B;	**while: *bif (cond*)* endwhile** ** A** ** j while** **endwhile B**
do A; while (*cond*) B;	**do: A** ** *bif (cond)* do** ** B**

do
  A;
  if(*cond1*) continue;
  B;
  if(*cond2*) break;
  C;
while (*cond3*)
D;
	do: A
	** *bif (cond1)* while**
	** B**
	** *bif (cond2)* enddo**
	** C**
	while: *bif (cond3)* do
	enddo: D

iterar *n* veces /* *n*>0 */ A; B;	** li *$r*,*n*** **loop: A** ** addi *$r*,*$r*,-1** ** bgtz *$r*,loop** ** B**

Llamadas al sistema del PCSpim

$v0	Nombre	Descripción	Argumentos	Resultado	Equivalente Java	Equivalente C
1	*print_integer*	Imprime (*) el valor de un entero	`$a0` = entero a imprimir	—	`System.out.print (int $a0)`	`printf("%d",$a0)`
2	*print_float*	Imprime (*) el valor de un *float*	`$f12` = float a imprimir	—	`System.out.print (float $f0)`	`printf("%f",$f0)`
3	*print_double*	Imprime (*) el valor de un *double*	`$f12` = double a imprimir	—	`System.out.print (double $f0)`	`printf("%Lf",$f0)`
4	*print_string*	Imprime una cadena de caracteres acabada en nul ('\0')	`$a0` = puntero a la cadena	—	`System.out.print (int $a0)`	`printf("%s",$a0)`
5	*read_integer*	Lee (*) el valor de un entero	—	`$v0` = entero leído		
6	*read_float*	Lee (*) el valor de un *float*	—	`$f0` = *float* leído		
7	*read_double*	Lee (*) el valor de un *double*	—	`$f0` = *double* leído		
8	*read_string*	Lee una cadena de caracteres (de longitud limitada) hasta encontrar un '\n' y la deja en el buffer acabada en nul ('\0')	`$a0` = puntero al buffer de entrada `$a1` = número máximo de caracteres de la cadena			
9	*sbrk*	Reservar un bloque de memoria del *heap*	`$a0` = longitud del bloque en bytes	`$v0` = dirección base del bloque de memoria		`malloc(integer n);`
10	*exit*	Fin de proceso	—	—		`exit(0);`
11	*print_character*	Imprime un carácter	`$a0` = carácter a imprimir			`putc(char c);`
12	*read_character*	Lee (**) un carácter		`$a0` = carácter leído		`getc();`

NOTAS
(*) El asterisco en *Imprime** y *Lee** indica que, además de la operación de entrada/salida, hay un cambio de representación de binario a alfanumérico o de alfanumérico a binario.
(**) En *pcspim-ES*, la función 12 lee un carácter del teclado sin producir un eco en la consola. En otras versiones del simulador sí escribe el eco

Codificación ASCII (ISO/IEC 8859-1)

Esta es la codificación utilizada por la consola y el teclado del simulador.

	_0	_1	_2	_3	_4	_5	_6	_7	_8	_9	_A	_B	_C	_D	_E	_F
	NUL	*SOH*	*STX*	*ETX*	*EOT*	*ENQ*	*ACK*	*BEL*	*BS*	*HT*	*LF*	*VT*	*FF*	*CR*	*SO*	*SI*
0_	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	DLE	*DC1*	*DC2*	*DC3*	*DC4*	*NAK*	*SYN*	*ETB*	*CAN*	*EM*	*SUB*	*ESC*	*FS*	*GS*	*RS*	*US*
1_	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
	(SP)	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
2_	20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F
	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
3_	30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F
	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
4_	40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F
	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
5_	50	51	52	53	54	55	56	57	58	59	5A	5B	5C	5D	5E	5F
	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
6_	60	61	62	63	64	65	66	67	68	69	6A	6B	6C	6D	6E	6F
	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	*DEL*
7_	70	71	72	73	74	75	76	77	78	79	7A	7B	7C	7D	7E	7F
	112	113	114	115	116	*117*	118	119	120	121	122	123	124	125	126	127
	PAD	*HOP*	*BPH*	*NBH*	*IND*	*NEL*	*SSA*	*ESA*	*HTS*	*HTJ*	*VTS*	*PLD*	*PLU*	*RI*	*SS2*	*SS3*
8_	80	81	82	83	84	85	86	87	88	89	8A	8B	8C	8D	8E	8F
	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
	DCS	*PU1*	*PU2*	*STS*	*CCH*	*MW*	*SPA*	*EPA*	*SOS*	*SGCI*	*SCI*	*CSI*	*ST*	*OSC*	*PM*	*APC*
9_	90	91	92	93	94	95	96	97	98	99	9A	9B	9C	9D	9E	9F
	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
	(NBSP)	¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	(SHY)	®	¯
A_	A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	AA	AB	AC	AD	AE	AF
	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
B_	B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	BA	BB	BC	BD	BE	BF
	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
C_	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB	CC	CD	CE	CF
	192	193	194	195	196	197	198	199	200	201	202	203	204	255	206	207
	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
D_	D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	DA	DB	DC	DD	DE	DF
	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
E_	E0	E1	E2	E3	E4	E5	E6	E7	E8	E9	EA	EB	EC	ED	EE	EF
	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ
F_	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	FA	FB	FC	FD	FE	FF
	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255
	_0	_1	_2	_3	_4	_5	_6	_7	_8	_9	_A	_B	_C	_D	_E	_F

Las celdas sombreadas corresponden a caracteres de control no imprimibles. (**SP)** denota el espacio entre palabras, (**NBSP)** significa *non-breaking space* y (**SHY)** *syllabe hyphen*.