ESTRUCTURA DE COMPUTADORS
Grau en Enginyeria Informàtica

Sessió de laboratori número 2

CRIDES AL SISTEMA

Objectius

- Consolidar els coneixements sobre la codificació ASCII
- Fer inventari de les instruccions de bifurcació del MIPS.
- Aplicar les instruccions de bifurcació per a implementar bucles i condicionals en assemblador
- Entendre els tres materials amb què es fa un programa en assemblador (instruccions, dades i crides al sistema).
- Conèixer i fer ús de les crides al sistema mitjançant la instrucció màquina `syscall`.

Material

- El simulador *pcspim-ES*.
- Arxius font en assemblador *forever.s*, *ascii-console.s, ascii-for, echo.s,*

Bibliografia

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

Introducció teòrica

El codi ASCII

L'estàndard Unicode actual, que pot codificar texts de multitud de llengües des de 1991, té un precedent en l'estàndard americà ASCII (*American Standard Code for Information Interchange*) definit en 1963. L'estàndard ASCII va evolucionar posteriorment per a ajustar-se millor a les necessitats canviants d'emmagatzemament i comunicacions digitals.

Entre les característiques de les primeres versions d'ASCII convé destacar:

- Codificava els caràcters en 7 bits, per a afegir que un bit addicional de paritat que completara els 8 bits que s'emmagatzemaven o se transmetien.
- Dels 128 codis possibles, reservava els 32 primers (del 0 al 31) i l'últim (127) per a control, sense representar cap caràcter gràfic.
- Els 95 codis restants representaven lletres, nombres i signes de puntuació propis de l'alfabet anglosaxó. No considerava, per tant, la Ç, la Ñ ni les vocals accentuades.
- Seguint l'ordre alfabètic, les lletres majúscules tenen codis consecutius. Igualment passa amb les minúscules. Així, ascii('B') = ascii('A') + 1; ascii('d') = ascii('a') + 3.

- Els dígits ‘0’ al ‘9’ també tenen codis consecutius. Per tant, 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

Taula 1. Codi ASCII de 7 bits. Les 33 cel·les ombrejades corresponen a caràcters de control no imprimibles. (**SP**) denota l’espai entre paraules.

Posteriorment, ASCII es va estendre a 8 bits i va estandarditzar-se com ISO/IEC_8859. Els 128 nous codis inclouen 32 caràcters de control; els 96 restants representen lletres i signes de puntuació. El nou estàndard va definir diferents parts o variants regionals, i per això en Europa Occidental fem servir la part IEC_8859-1, també nomenada *latin1*. Vegeu en l’apèndix la codificació completa d’aquesta part. Actualment està integrada en l’estàndard Unicode.

Las crides al sistema en *pcspim-ES*

Els computadors disposen d’un sistema operatiu que ofereix un catàleg de *system calls* o *system functions*. Amb elles, els processos poden accedir de manera segura i eficient als recursos compartits del computador: el processador, la memòria principal i els perifèrics. En el bloc de l’assignatura referent a l’entrada/eixida n’estudiarem alguns detalls d’implementació.

El simulador disposa de dos perifèrics de text: el teclat i la consola. Ambdós codifiquen els caràcters segons l’estàndard ISO/IEC 8859-1. El teclat, a més a més dels codis alfanumèrics, genera codis de control en combinar la tecla *ctrl* amb les tecles alfabètiques. El simulador interpreta directament les tecles de cursor i el *ctrl-C* y per això els programes simulats no les poden llegir.

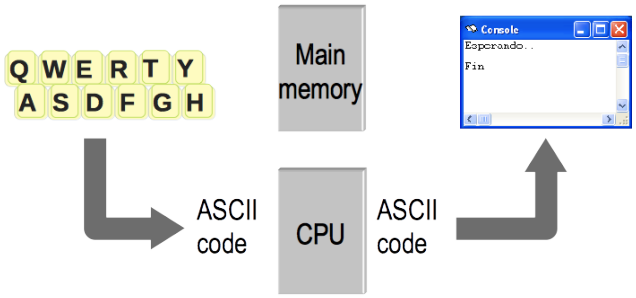


Figura 1. PCSPIM modela un computador amb un teclat que produeix codis ASCII de 8 bits i una consola que els consumeix.

En un MIPS, les funcions del sistema s'invoquen mitjançant la instrucció **`syscall`**. Cada funció es distingeix per un índex únic que la identifica, pren una sèrie d'arguments (depenent de l'índex) i torna un resultat possible.

El catàleg de funcions del sistema simulat en PCSpim es troba en l'apèndix d'aquest enunciat. En aquesta pràctica només treballareu amb les cinc funcions referides en la taula següent. Noteu que el registre **`$v0`** ha de contenir l'índex que identifica la funció, que alguns casos cal desar un paràmetre en el registre **`$a0`** i que les crides que tornen un resultat ho fan en **`$v0`**:

Nom	$v0	Descripció	Arguments	Resultat
print_int	*1*	Imprimeix el valor d'un enter	`$a0` = enter a imprimir	—
read_int	*5*	Llig el valor d'un enter	—	`$v0` = enter llegit
exit	*10*	Acaba el procés	—	—
print_char	*11*	Imprimeix un caràcter	`$a0` = caràcter a imprimir	—
read_char	*12*	Llig un caràcter	—	`$v0` = caràcter llegit

Taula 2. Funcions del sistema que heu d'utilitzar en aquesta pràctica.

Il·lustrem amb un exemple el mecanisme de crida. El codi següent llegeix un nombre enter des del teclat i el copia en l'adreça de memòria etiquetada amb el nom **`valor`**:

```
li $v0, 5          # Índex de la crida read_int
syscall            # Crida al sistema read_int
sw $v0, valor      # Copia l'enter llegit a la memòria
```

Alguns aspectes de les funcions d'entrada/eixida

Les funciones *print_char* i *read_char* no fan cap canvi de format. És a dir, *print_char* imprimeix en la consola el codi que rep en **`$a0`** i *read_char* torna en **`$v0`** el codi generat pel teclat. En canvi, *print_int* transforma l'enter desat en **`$a0`** en la corresponent cadena de caràcters (codificats en ASCII) llegible pels humans. Igualment, *read_int* processa una cadena de caràcters teclejada per un humà i calcula el valor enter que torna en **`$v0`**.

Altre detall es l'*eco*. La funció *read_int*, a més a més de llegir del teclat, escriu en la consola els caràcters llegits, creant la il·lusió de que l'usuari escriu en la pantalla. La funció *read_char*, en *pcspim-ES*, no genera eco.

Control de flux d'execució en assemblador

Les instruccions de salt, junt amb certes instruccions aritmètiques, permeten construir les estructures condicionals i iteratives.

A baix nivell, podem distingir entre:

- salts incondicionals del tipus *seguir en la l'adreça*, on *adreça* senyala la instrucció que s'executarà a continuació: El MIPS disposa de la instrucció `j eti`.

- salts condicionals, també nomenats bifurcacions, del tipus *si (condició) seguir en l'adreça* on *adreça* assenyala la instrucció que s'executaria tot seguit. En el joc del MIPS, tenim sis condicions per a salts condicionals: noteu que es poden fer tres parelles de condicions contràries (= i ≠, > i ≤, < i ≥).

El joc d'instruccions només permet les comparacions = i ≠ entre dos registres i les comparacions >, ≤, < i ≥ entre un registre i el zero:

`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 ≠ rt	*rs ≤ 0*	*rs ≥ 0*

Tabla 3. Instruccions de bifurcació del MIPS

Aquest assortiment de condicions es pot ampliar amb ajuda de la instrucció aritmètica *slt (set on less than)* i les instruccions relacionades que estudiareu en el tema d'aritmètica d'enters. Així s'hi obtenen aquestes altres sis pseudoinstruccions:

`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 ≠ 0	*rs ≤ rt*	*rs ≥ rt*

Tabla 4. Pseudoinstruccions de bifurcació del MIPS

Vegeu la traducció d'un parell de pseudoinstruccions de salt en instruccions màquina en la taula següent:

Pseudoinstrucció	**Instruccions màquina**
`beqz rs,A`	`beq rs,$zero,A`
`bgt rs,rt,A`	`slt $at,rt,rs` `bne $at,$zero,A`

Tabla 5. Traducció de les pseudoinstruccions `beqz` i `bgt` en instruccions del MIPS

Amb aquestes instruccions podeu construir estructures condicionals i iteratives equivalents a les que escriviu en alt nivell. Per exemple, si hi ha un bloc d'instruccions A1, A2... que només s'ha d'executar si el contingut d'un registre $r és negatiu, podeu triar una bifurcació que salte si es dóna la condició contrària ($r ≥ 0):

Per a iterar n vegades un bloc d'instruccions A1, A2..., trieu un registre $r i escriviu:

Vegeu en l'annex un quadre amb la traducció de diverses estructures de control de flux.

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

Per a iterar n vegades un bloc d'instruccions A1, A2..., trieu un registre `$r` i escriviu:

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

En l'annex podeu consultar un quadre amb la traducció de diverses estructures de control de flux.

Exercicis de laboratori

Configuració del simulador *pcspim-ES*

En engegar el simulador *pcspim-ES*, comproveu que la configuració definida en *Simulator->Settings...* coincideix amb la mostrada en la figura.

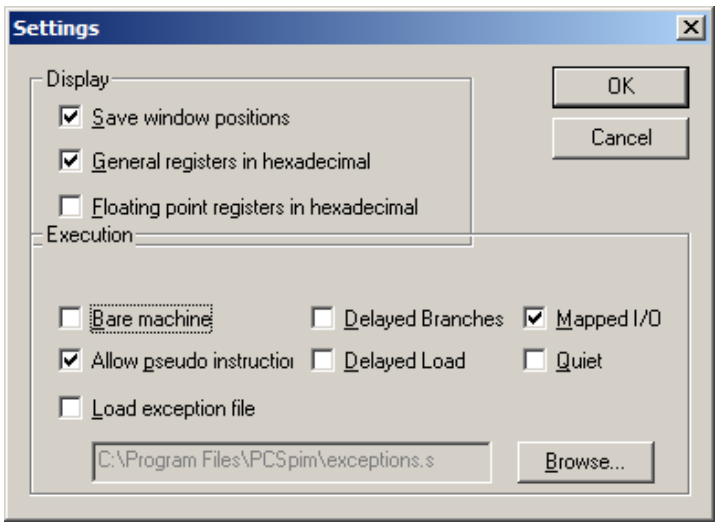


Figura 2. Configuració de *pcspim-ES* per aquesta pràctica.

Exercici 1: El bucle infinit

Observeu el codi contingut en l'arxiu *forever.s* (figura 3). Noteu que nomes especifica el segment d'instruccions (`.text`). Abans de simular-lo, proveu a entendre'l. El programa només fa que llegir enters de la consola i sumar-los.

- Busqueu en el codi les quatre crides al sistema utilitzades pel programa i consulteu en la taula què hi fan i cóm es fan servir. Noteu (1) l'ús de `$v0` per a seleccionar la funció en tots els casos, (2) l'ús de `$v0` per a arreplegar el valor llegit per *read_int* i (3) l'ús de `$a0` per a l'argument de *print_int()* i *print_char()*.
- Noteu que hi ha un bucle. Quina és la primera instrucció del bucle? I l'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. Codi font i pseudocodi de *forever.s*, el primer exercici de la pràctica. Observeu que el bucle és infinit i que el flux d'execució mai no arriba a les dues últimes instruccions (`li`, `syscall`).

- Què fa cada iteració del bucle? Podríeu explicar cada instrucció i cada pseudoinstrucció que conté?

Carregueu ara el programa en el simulador.

- Com s'ha traduït la línia `b loop`?
- Sabeu executar el programa sencer? Feu-lo (ordre *Go*, tecla F5). Mentre s'executa, manteniu activa la finestra *Console*. Teniu en compte que el programa espera entrada de números pel teclat sense escriure en la consola cap text que ho indique. Teclegeu valores numèrics amb signe.

  > **Tècnica experimental: el *ctrl-C***. Quan la ventana de la consola està activa, *ctrl-C* atura el programa en el punt en què es troba, igual que en la consola de Unix. Té el mateix efecte que *Simulator>Break*.

- Atureu l'execució del codi. Voreu un missatge que diu *"Execution paused by the user at* <adreça> *Continue execution?"*; anoteu l'adreça i piqueu sobre el botó *No*. Observeu ara la finestra principal del simulador (Figura 4) i busqueu en ella la solució a les qüestions següents:

  1. Quina és l'última instrucció que s'ha executat? Recordeu l'adreça anotada i busqueu la instrucció en la finestra d'instruccions.
  2. Quina serà la instrucció que s'anava a executar en aquest moment? Consulteu el valor del PC actual (en la finestra del processador) i busqueu la instrucció corresponent.
  3. Què contenen els registres `$a0`, `$v0` i `$s0`? Busqueu el valor en la ventana del processador. Sabeu canviar la base de numeració de decimal a hexadecimal? (quadre de configuració *Simulator>Settings>Display*).

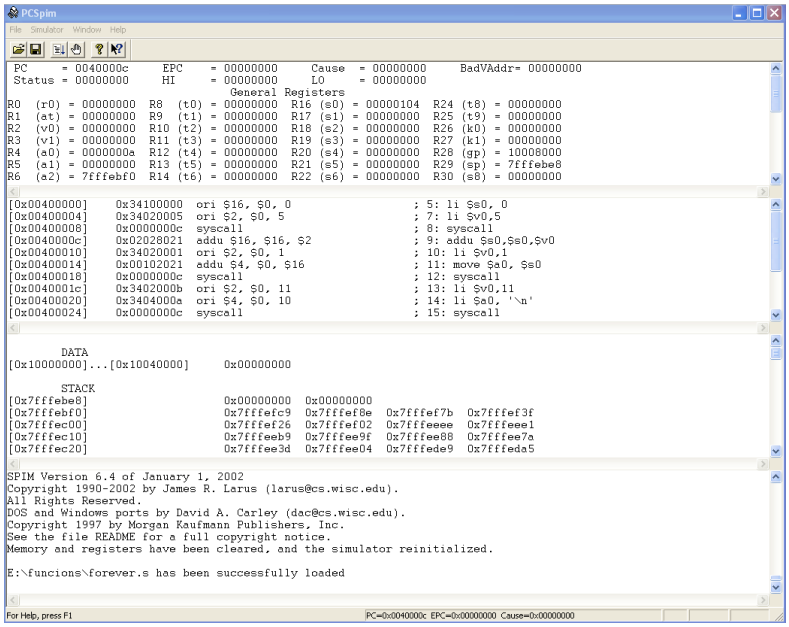


Figura 5. La interfície del simulador mostra quatre seccions. De dalt a baix: (1) l'estat del processador, amb el contingut dels seus registres més significatius, (2) la memòria d'instruccions, (3) la memòria de dades i (4) els avisos del simulador.

- Proveu a reprendre'n l'execució. Notareu que a la orden *Go* (tecla F5), el simulador proposa com *Starting Address* l'adreça de la primera instrucció a executar; és a dir, el valor actual del PC.
- Si teniu clar el funcionament del codi, passeu a l'exercici següent.

Exercici 2: Limitar el bucle

En aquest exercici heu d'aconseguir que el bucle s'ature quant l'usuari teclege un zero. La detenció s'aconsegueix afegint al bucle un salt condicional a les instruccions que implementen la crida *exit()*. Comenceu amb el codi de *forever* i guardeu el codi modificat en l'arxiu *"break.s"*:

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

Figura 6. Pseudocodi de *break.s* (exercici 2), resultant d'afegir la línia marcada amb ← al pseudocodi de *forever.s* (exercici 1).

- Heu d'afegir una etiqueta al final del bucle i saltar-hi si *$v0 = 0*. Trieu un nom adient per a l'etiqueta (**`fin`**, **`salida`**, etc.). Teniu clar on col·locar-la?
- Afegiu la instrucció de salt. Teniu clar quina?
- Comproveu amb el simulador que el codi és correcte i que el programa s'atura en teclejar un 0.
- En acabant l'execució, què valen els registres **`$v0`** i **`$s0`**? I el PC?

Exercici 3: El bucle i el comptador

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 7. Pseudocodi i exemple d'execució de *counter.s*. El programa llig enters fins que s'hi teclege un 0 i manté en $s1 el compte de sumands llegits. Fins el final no mostra la suma acumulada.

- Millor partiu del codi de *break.s*, modifiqueu-lo i guardeu el nou codi en *counter.s*.
- Afegiu les instruccions que inicialitzen, incrementen e imprimeixen el valor de **`$s1`**.
- Traieu fora del bucle les instruccions que imprimeixen el valor de **`$s0`** i completeu el codi final.
- Proveu el codi resultant.

Exercici 4. Codis ASCII en la consola

Considereu el bucle que en alt nivell s'expressa com

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

En assemblador i en pseudocodi es pot expressar així:

```
        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 8. El bucle *for* de l'exercici 4 fa servir un comptador (**`$s0`**) que s'incrementa al final de cada iteració i un registre per al valor límit (**`$s1=127`**). En aquest cas, la guarda és simple i es tradueix en una única instrucció de salt condicional.

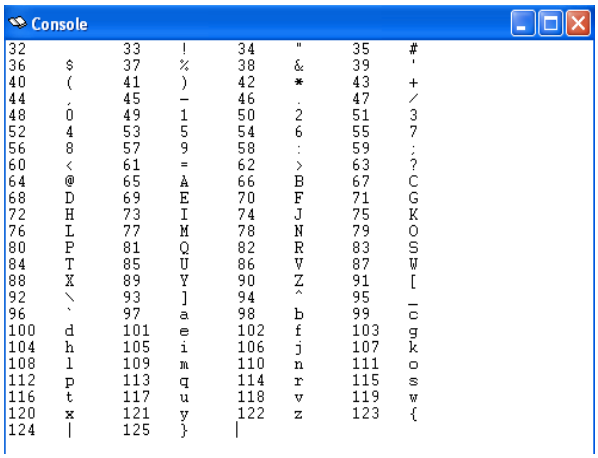
El codi *ascii-console.s* imprimeix els caràcters gràfics del codi ASCII clàssic de 7 bits. En el bucle s'ometen els codis del 0 al 31 (secció C0 de l'estàndard), el codi 127 (**`DEL`**) i els caràcters estesos del 128 al 255.

```
        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'); // Salt de línia
      }
exit();
```

Figura 9. Codi, pseudocodi i eixida per pantalla de *ascii-console.s*

- Obriu l'arxiu *ascii-console.s* amb un editor de textos i vegeu l'estructura *for* que conté. Comproveu el funcionament del programa amb el simulador.
- Observeu l'ús dels caràcters de control '\t' (codi 9) i '\n' (codi 10).
- Quin és el conjunt de caràcters representables per la consola? Canvieu els límits del bucle per a provar els caràcters del 0 al 255. Executeu el codi i observeu-ne el resultat. Quan la consola no pot imprimir un caràcter, mostra ▌.
- També per provar, modifiqueu *ascii-console.s* perquè lliste els caràcters en ordre decreixent, del 126 al 32. Comproveu el codi amb el simulador.
- Modifiqueu el codi d'*ascii-console.s* (i guardeu-lo com *ascii-console-tab.s*) perquè tabule els codis del 32 al 126 en la consola com mostra la figura:




```
$s2=4;
for ($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 10. Eixida i pseudocodi d'*ascii-console.s*

- Comproveu el codi *ascii-console-tab.s*.

Qüestions diverses

Es tracta de qüestions de llapis i paper, que en alguns casos podreu comprovar amb el simulador.

1. Quina serà l'eixida del codi *(ascii-for)* següent?

```
        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
```

   I si canviara la línia `li $v0,11` per `li $v0,1`?
   I si canviara la línia `addi $s0,$s0,1` per `addi $s0,$s0,-1`?
   I si canviara la línia `addi $s1,$s1,-1` per `addi $s1,$s1,-2`?

2. En el text *(echo.s)* següent, heu de substituir cada ***bif*** por una instrucció de bifurcació perquè el codi llisca reiteradament del teclat, faça eco dels caràcters llegits només si son xifres i acabe en prémer la lletra '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 calguera una pseudoinstrucció `ca2 rt,rs` que fera l'operació *rt = complement_a_2(rs)*, com es traduiria? Hi ha alguna pseudoinstrucció estàndard del MIPS equivalent a `ca2`?
4. Amb l'ajuda del simulador, proveu a carregar codi on aparega la pseudoinstrucció `li $1,20` o `li $at,20`. Què diu el simulador?
5. Podeu explicar la diferència entre les crides print_char(100) i print_integer(100)?
6. I la diferència entre print_char('A') i print_integer('A')?
7. En la Taula 5 teniu la traducció de dues de les sis pseudoinstruccions de la Taula 4. Quina és la traducció de les quatre que falten?

Annex

Exemples de control de flux

En la taula següent,

- El símbolo *bif (cond)* denota una bifurcació que salta si la condició *cond* s'acompleix. Els símbols *cond*, *cond1*, etc., fan referència a les sis condicions simples (= y ≠, > y ≤, < y ≥) que relacionen dos valors. El asterisc indica condició contrària; per exemple, si *cond* = "*>*" tenim *cond** = "≤".
- En la columna d'alt nivell, els símbols *A*, *B*, etc. indiquen sentències simples o compostes; en la columna de baix nivell, els símbols **A**, **B**, etc. representen els blocs d'instruccions equivalents en assemblador.

Condicionals

Alt nivell	Assemblador

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

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

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

Selectors

Alt nivell	Assemblador

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

Iteracions

Alt nivell	Assemblador
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**

Crides al sistema del PCSpim

$v0	Nom	Descripció	Arguments	Resultat	Equivalent Java	Equivalent C
1	*print_integer*	Imprimeix (*) el valor d'un enter	`$a0` = enter a imprimir	—	`System.out.print (int $a0)`	`printf("%d",$a0)`
2	*print_float*	Imprimeix (*) el valor d'un *float*	`$f12` = float a imprimir	—	`System.out.print (float $f0)`	`printf("%f",$f0)`
3	*print_double*	Imprimeix (*) el valor d'un *double*	`$f12` = double a imprimir	—	`System.out.print (double $f0)`	`printf("%Lf",$f0)`
4	*print_string*	Imprimeix una cadena de caràcters acabada en nul ('\0')	`$a0` = punter a la cadena	—	`System.out.print (int $a0)`	`printf("%s",$a0)`
5	*read_integer*	Llig (*) el valor d'un entero	—	`$v0` = enter llegit		
6	*read_float*	Llig (*) el valor d'un *float*	—	`$f0` = *float* llegit		
7	*read_double*	Llig (*) el valor d'un *double*	—	`$f0` = *double* llegit		
8	*read_string*	Llig una cadena de caràcters (de longitud limitada) fins trobar un '\n' i la deixa en el buffer acabada en nul ('\0')	`$a0` = punter al buffer d'entrada `$a1` = nombre màxim de caràcters de la cadena			
9	*sbrk*	Reserva un bloc de memòria del *heap*	`$a0` = longitud del bloc en bytes	`$v0` = adreça base del bloc de memòria		`malloc(integer n);`
10	*exit*	Final de procés	—	—		`exit(0);`
11	*print_character*	Imprimeix un caràcter	`$a0` = caràcter a imprimir			`putc(char c);`
12	*read_character*	Llig (**) un caràcter		`$a0` = caràcter llegit		`getc();`

NOTES
(*) El asterisc en *Imprimeix** y *Llig** indica que, a més a més de l'operació d'entrada/eixida, hi ha un canvi de format de binari a alfanumèric o d'alfanuméric a binari.
(**) En *pcspim-ES*, la funció 12 llig un caràcter del teclat sense produir un eco en la consola. En altres versions del simulador sí escriu l'eco

Codificació ASCII (ISO/IEC 8859-1)

Aquesta és la codificació utilitzada per la consola i el teclat PC-SPIM.

	_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 cel·les ombrejades corresponen a caràcters de control no imprimibles. (**SP)** denota l'espai entre paraules, (**NBSP)** significa *non-breaking space* y (**SHY)** *syllabe hyphen*.