PRACTICA 6

Procesador RISC: utilizando la E/S

Objetivos: Familiarizarse con el desarrollo de programas para procesadores con sets reducidos de instrucciones (RISC). Resolver problemas y verificarlos a través de simulaciones. Dominar el uso del puerto de E/S provisto en el WinMIPS64.

Puerto de E/S mapeado en memoria de datos – Resumen

CONTROL y DATA son direcciones de memoria fijas para acceder al puerto de E/S. Aplicando distintos comandos a través de CONTROL, es posible producir salidas o ingresar datos a través de la dirección DATA. Las direcciones de memoria de CONTROL y DATA son 0x10000 y 0x10008 respectivamente. Como el set de instrucciones del procesador MIPS64 no cuenta con instrucciones que permitan cargar un valor inmediato de más de 16 bits (como es el caso de las direcciones mencionadas), resulta conveniente definirlas en la memoria de datos, así:

```
CONTROL: .word32 0x10000
   DATA: .word32 0x10008
```

De esa manera, resulta sencillo cargar esas dos direcciones en registros mediante:

```
lwu  $s0, CONTROL($zero)  ; Carga el valor 0x10000 en $s0
lwu  $s1, DATA($zero)     ; Carga el valor 0x10008 en $s1
```

- Si se escribe en DATA un número entero y se escribe un 1 en CONTROL, se interpretará el valor escrito en DATA como un entero sin signo y se lo imprimirá en la pantalla alfanumérica de la terminal.
- Si se escribe en DATA un número entero y se escribe un 2 en CONTROL, se interpretará el valor escrito en DATA como un entero con signo y se lo imprimirá en la pantalla alfanumérica de la terminal.
- Si se escribe en DATA un número en punto flotante y se escribe un 3 en CONTROL, se imprimirá en la pantalla alfanumérica de la terminal el número en punto flotante.
- Si se escribe en DATA la dirección de memoria del comienzo de una cadena terminada en 0 y se escribe un 4 en CONTROL, se imprimirá la cadena en la pantalla alfanumérica de la terminal.
- Si se escribe en DATA un color expresado en RGB (usando 4 bytes para representarlo: un byte para cada componente de color e ignorando el valor del cuarto byte), en DATA+4 la coordenada Y, en DATA+5 la coordenada X y se escribe un 5 en CONTROL, se pintará con el color indicado un punto de la pantalla gráfica de la terminal, cuyas coordenadas están indicadas por X e Y. La pantalla gráfica cuenta con una resolución de 50x50 puntos, siendo (0, 0) las coordenadas del punto en la esquina inferior izquierda y (49, 49) las del punto en la esquina superior derecha.
- Si se escribe un 6 en CONTROL, se limpia la pantalla alfanumérica de la terminal.
- Si se escribe un 7 en CONTROL, se limpia la pantalla gráfica de la terminal.
- Si se escribe un 8 en CONTROL, se permitirá ingresar en la terminal un número (entero o en punto flotante) y el valor del número ingresado estará disponible para ser leído en DATA.
- Si se escribe un 9 en CONTROL, se esperará a que se presione una tecla en la terminal (la cuál no se mostrará al ser presionada) y el código ASCII de dicha tecla estará disponible para ser leído en DATA.

1) El siguiente programa produce la salida de un mensaje predefinido en la ventana Terminal del simulador WinMIPS64. Teniendo en cuenta las condiciones de control del puerto de E/S (en el resumen anterior), modifique el programa de modo que el mensaje a mostrar sea ingresado por teclado en lugar de ser un mensaje fijo.

```
        .data
texto:  .asciiz  "Hola, Mundo!"     ; El mensaje a mostrar
CONTROL: .word32  0x10000
DATA:   .word32  0x10008

        .text
        lwu    $s0, DATA($zero)     ; $s0 = dirección de DATA
        daddi  $t0, $zero, texto    ; $t0 = dirección del mensaje a mostrar
        sd     $t0, 0($s0)          ; DATA recibe el puntero al comienzo del mensaje

        lwu    $s1, CONTROL($zero)  ; $s1 = dirección de CONTROL
        daddi  $t0, $zero, 6        ; $t0 = 6 -> función 6: limpiar pantalla alfanumérica
        sd     $t0, 0($s1)          ; CONTROL recibe 6 y limpia la pantalla

        daddi  $t0, $zero, 4        ; $t0 = 4 -> función 4: salida de una cadena ASCII
        sd     $t0, 0($s1)          ; CONTROL recibe 4 y produce la salida del mensaje
        halt
```

2) Escriba un programa que utilice sucesivamente dos subrutinas: La primera, denominada `ingreso`, debe solicitar el ingreso por teclado de un número entero (de un dígito), verificando que el valor ingresado realmente sea un dígito. La segunda, denominada `muestra`, deberá mostrar en la salida estándar del simulador (ventana Terminal) el valor del número ingresado expresado en letras (es decir, si se ingresa un '4', deberá mostrar 'CUATRO'). Establezca el pasaje de parámetros entre subrutinas respetando las convenciones para el uso de los registros y minimice las detenciones del cauce (ejercicio similar al ejercicio 6 de Práctica 2).

Resolución:

```
          .data
CONTROL:  .word32 0x10000
DATA:     .word32 0x10008

sale:     .byte 0x20 ; Sale con espacio
cero:     .byte 0x30
nueve:    .byte 0x39

letras:   .asciiz "CERO"
          .asciiz "UNO"
          .asciiz "DOS"
          .asciiz "TRES"
          .asciiz "CUATRO"
          .asciiz "CINCO"
          .asciiz "SEIS"
          .asciiz "SIETE"
          .asciiz "OCHO"
          .asciiz "NUEVE"

          .code
          daddi $s0, $0, -1
salto:    jal ingreso
          beq $s0, $v0, final
          dadd $a0, $0, $v0
          jal muestra
          j salto
final:    halt

ingreso:  lwu $t2, CONTROL($0) ; $t2 = dirección de CONTROL
          lwu $t3, DATA($0)    ; $t3 = dirección de DATA

          lbu $t4, cero($0)
          lbu $t5, nueve($0)
          lbu $t7, sale($0)
          daddi $t0, $0, 9
sigue:    sd $t0, 0($t2)
          ld $t1, 0($t3)
          bne $t7, $t1, comp
          daddi $v0, $0, -1
          j fin_ing

comp:     slt $t6, $t1, $t4
          bnez $t6, sigue
          slt $t6, $t5, $t1
          bnez $t6, sigue
          daddi $v0, $t1, -0x30
fin_ing:  jr $ra

muestra:  lwu $t2, CONTROL($0) ; $t2 = dirección de CONTROL
          lwu $t3, DATA($0)    ; $t3 = dirección de DATA

          dsll $t1, $a0, 3
          daddi $t0, $t1, letras

          sd $t0, 0($t3)     ; DATA recibe el puntero al comienzo del mensaje
          daddi $t0, $0, 4   ; $t0 = 4 -> función 4: salida de una cadena ASCII
          sd $t0, 0($t2)     ; CONTROL recibe 4 y produce la salida del mensaje

          jr $ra
```

3) Escriba un programa que realice la suma de dos números enteros (de un dígito cada uno) utilizando dos subrutinas: La denominada `ingreso` del ejercicio anterior (ingreso por teclado de un dígito numérico) y otra denominada `resultado`, que muestre en la salida estándar del simulador (ventana Terminal) el resultado numérico de la suma de los dos números ingresados (ejercicio similar al ejercicio 7 de Práctica 2).

4) Escriba un programa que solicite el ingreso por teclado de una clave (sucesión de cuatro caracteres) utilizando la subrutina `char` de ingreso de un carácter. Luego, debe comparar la secuencia ingresada con una cadena almacenada en la variable `clave`. Si las dos cadenas son iguales entre si, la subrutina llamada `respuesta` mostrará el texto "Bienvenido" en la salida estándar del simulador (ventana Terminal). En cambio, si las cadenas no son iguales, la subrutina deberá mostrar "ERROR" y solicitar nuevamente el ingreso de la clave.

Resolución:

```
            .data

CONTROL:    .word32 0x10000
DATA:       .word32 0x10008

ingresar:   .asciiz "Ingrese Clave: "
bien:       .asciiz "Bienvenido.\n"
mal:        .asciiz "ERROR\n"

leida:      .byte    0,0,0,0,0
clave:      .asciiz "hola"

            .code
            daddi $sp, $0, 0x400
            daddi $a0, $0, ingresar
            jal mostrar

            daddi $a0, $0, leida
            jal leeclave

            daddi $a0, $0, leida
            jal check

            dadd $a0, $0, $v0
            jal respuesta
            halt

    mostrar: lwu $t1, CONTROL($0) ; $t1 = dirección de CONTROL
             lwu $t2, DATA($0)    ; $t2 = dirección de DATA

             sd $a0, 0($t2)     ; DATA recibe el puntero al comienzo del mensaje
             daddi $t0, $0, 4   ; $t0 = 4 -> función 4: salida de una cadena ASCII
             sd $t0, 0($t1)
             jr $ra


       char: lwu $t1, CONTROL($0) ; $t1 = dirección de CONTROL
             lwu $t2, DATA($0)    ; $t2 = dirección de DATA
             daddi $t0, $0, 9
             sd $t0, 0($t1)
             ld $v0, 0($t2)
             jr $ra


   leeclave: daddi $sp, $sp, -24
             sd $ra, 0($sp)
             sd $s0, 8($sp)
             sd $s0, 16($sp)

             daddi $s0, $0, 4
             dadd  $s1, $0, $a0
   loop_lee: jal char
             sb $v0, 0($s1)
             daddi $s1, $s1,  1
             daddi $s0, $s0, -1
             bnez $s0, loop_lee
```

```
            ld $ra, 0($sp)
            ld $s0, 8($sp)
            ld $s1, 16($sp)
            daddi $sp, $sp, 24
            jr $ra

     check: dadd $v0, $0, $0
            lwu $t0, 0($a0)
            lwu $t1, clave($0)
            bne $t0, $t1, distinta
            daddi $v0, $v0, 1
  distinta: jr $ra

 respuesta: beqz $v0, imp_error
            daddi $t0, $0, bien
            j impr
imp_error: daddi $t0, $0, mal
      impr: lwu $t1, DATA($0) ; $t1 = dirección de CONTROL
            sd $t0, 0($t1)
            daddi $t3, $0, 4
            lwu $t2, CONTROL($0)    ; $t2 = dirección de DATA
            sd $t3, 0($t2)
            jr $ra
```

5) Escriba un programa que calcule el resultado de elevar un valor en punto flotante a la potencia indicada por un exponente que es un número entero positivo. Para ello, en el programa principal se solicitará el ingreso de la base (un número en punto flotante) y del exponente (un número entero sin signo) y se deberá utilizar la subrutina `a_la_potencia` para calcular el resultado pedido (que será un valor en punto flotante). Tenga en cuenta que cualquier base elevada a la 0 da como resultado 1. Muestre el resultado numérico de la operación en la salida estándar del simulador (ventana Terminal).

Resolución:

```
        .data
CONTROL:            .word32 0x10000
DATA:               .word32 0x10008

ingreso_base:       .asciiz "Ingrese Base: "
ingreso_exponente:  .asciiz "Ingrese Exponente: "
resultado:          .asciiz "El resultado es: "

        .code
        daddi $sp, $0, 0x400    ; Inicializa el puntero al tope de la pila

        jal lee_base

        dadd $s0, $v0, $0
        jal lee_exponente

        dadd $a0, $s0, $0
        dadd $a1, $v0, $0

        jal a_la_potencia

        dadd $s0, $v0, $0

        daddi $a0, $0, resultado
        jal mostrar

        dadd $a0, $0, $s0
        jal muestra_flotante

        halt

   lee_base: daddi $sp, $sp, -8
             sd $ra, 0($sp)
             daddi $a0, $0, ingreso_base
             jal muestra_cadena

             jal lee_numero

             dadd $a0, $v0, $0
             jal muestra_flotante

             ld $ra, 0($sp)
             daddi $sp, $sp, 8
             jr $ra

    lee_exponente:  daddi $sp, $sp, -8
                    sd $ra, 0($sp)
                    daddi $a0, $0, ingreso_exponente
                    jal muestra_cadena

                    jal lee_numero

                    dadd $a0, $v0, $0
                    jal muestra_entero

                    ld $ra, 0($sp)
                    daddi $sp, $sp, 8
                    jr $ra

     a_la_potencia: daddi $t1, $0, 1
```

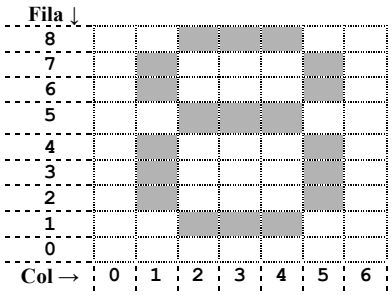
```
                    mtc1 $t1, f0
                    cvt.d.l f0, f0

                    mtc1 $a0, f1

              ciclo: beqz $a1, final
                    mul.d f0, f0, f1
                    daddi $a1, $a1, -1
                    j ciclo

               final: mfc1 $v0, f0
                    jr $ra

    muestra_entero: lwu $t2, CONTROL($0) ; $t2 = dirección de CONTROL
                    lwu $t3, DATA($0)    ; $t3 = dirección de DATA
                    sd $a0, 0($t3)      ; DATA recibe el puntero al comienzo del mensaje
                    daddi $t0, $0, 1    ; $t0 = 1 -> función 1:
                                        ; salida de un entero sin signo
                    sd $t0, 0($t2)
                    jr $ra


  muestra_flotante: lwu $t2, CONTROL($0) ; $t2 = dirección de CONTROL
                    lwu $t3, DATA($0)    ; $t3 = dirección de DATA
                    sd $a0, 0($t3)     ; DATA recibe el puntero al comienzo del mensaje
                    daddi $t0, $0, 3   ; $t0 = 3 -> función 3: salida de un flotante
                    sd $t0, 0($t2)
                    jr $ra

    muestra_cadena: lwu $t2, CONTROL($0) ; $t2 = dirección de CONTROL
                    lwu $t3, DATA($0)    ; $t3 = dirección de DATA
                    sd $a0, 0($t3)     ; DATA recibe el puntero al comienzo del mensaje
                    daddi $t0, $0, 4   ; $t0 = 4 -> función 4: salida de una cadena ASCII
                    sd $t0, 0($t2)
                    jr $ra

lee_numero:  lwu $t2, CONTROL($0) ; $t2 = dirección de CONTROL
             lwu $t3, DATA($0)    ; $t3 = dirección de DATA
             daddi $t0, $0, 8   ; $t0 = 8 -> función 8: leer entero o flotante
             sd $t0, 0($t2)
             ld $v0, 0($t3)
             jr $ra
```

6) El siguiente programa produce una salida estableciendo el color de un punto de la pantalla gráfica (en la ventana Terminal del simulador WinMIPS64). Modifique el programa de modo que las coordenadas y color del punto sean ingresados por teclado.

```
          .data
coorX:    .byte   24                  ; coordenada X de un punto
coorY:    .byte   24                  ; coordenada Y de un punto
color:    .byte   255, 0, 255, 0      ; color: máximo rojo + máximo azul => magenta
CONTROL:  .word32 0x10000
DATA:     .word32 0x10008

          .text
          lwu    $s6, CONTROL($zero)  ; $s6 = dirección de CONTROL
          lwu    $s7, DATA($zero)     ; $s7 = dirección de DATA

          daddi  $t0, $zero, 7        ; $t0 = 7 -> función 7: limpiar pantalla gráfica
          sd     $t0, 0($s6)          ; CONTROL recibe 7 y limpia la pantalla gráfica

          lbu    $s0, coorX($zero)    ; $s0 = valor de coordenada X
          sb     $s0, 5($s7)          ; DATA+5 recibe el valor de coordenada X
          lbu    $s1, coorY($zero)    ; $s1 = valor de coordenada Y
          sb     $s1, 4($s7)          ; DATA+4 recibe el valor de coordenada Y
          lwu    $s2, color($zero)    ; $s2 = valor de color a pintar
          sw     $s2, 0($s7)          ; DATA recibe el valor del color a pintar

          daddi  $t0, $zero, 5        ; $t0 = 5 -> función 5: salida gráfica
          sd     $t0, 0($s6)          ; CONTROL recibe 5 y produce el dibujo del punto
          halt
```

7) Se desea realizar la demostración de la transformación de un carácter codificado en ASCII a su visualización en una matriz de puntos con 7 columnas y 9 filas. Escriba un programa que realice tal demostración, solicitando el ingreso por teclado de un carácter para luego mostrarlo en la pantalla gráfica de la terminal.

El carácter '8' es representado como:



Resolución:

```
      .data
CONTROL:          .word32 0x10000
DATA:       .word32 0x10008

txt_car:    .asciiz "Ingrese Caracter: "
car:        .byte 0

color:      .byte 0, 0, 0, 0

; Tabla obtenida de: https://github.com/idispatch/raster-fonts/blob/master/font-7x9.c
; Los últimos caracteres no "entran" en la memoria del simulador
; El problema esta en que cada linea nueva que ponemos con .byte se
; alinea automaticamente a 8
; Como tenemos 9 entradas por cada caracter, "perdemos" 7 bytes con esa alineación
; Configurar en arquitectura, Bus de Datos a por lo menos 12 bits

caracteres:
      ; code=0, hex=0x00
  .byte 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
  .byte 0x38, 0x44, 0xAA, 0xAA, 0x82, 0xAA, 0x94, 0x78, 0x00
  .byte 0x38, 0x7C, 0xD6, 0xD6, 0xFE, 0xBA, 0xC6, 0x7C, 0x00
```

```
.byte 0x00, 0x6C, 0xFE, 0xFE, 0xFE, 0x7C, 0x38, 0x10, 0x00
.byte 0x00, 0x10, 0x38, 0x7C, 0xFE, 0x7C, 0x38, 0x10, 0x00
.byte 0x38, 0x38, 0x10, 0xD6, 0xFE, 0xD6, 0x10, 0x7C, 0x00
.byte 0x10, 0x38, 0x7C, 0xFE, 0xFE, 0x54, 0x10, 0x38, 0x00
.byte 0x00, 0x00, 0x18, 0x3C, 0x3C, 0x18, 0x00, 0x00, 0x00
.byte 0xFE, 0xFE, 0xE6, 0xC2, 0xC2, 0xE6, 0xFE, 0xFE, 0xFE
.byte 0x00, 0x18, 0x3C, 0x66, 0x66, 0x3C, 0x18, 0x00, 0x00
.byte 0xFE, 0xE6, 0xC2, 0x98, 0x98, 0xC2, 0xE6, 0xFE, 0xFE
.byte 0x0E, 0x06, 0x0A, 0x18, 0x3C, 0x66, 0x66, 0x3C, 0x00
.byte 0x3C, 0x66, 0x66, 0x3C, 0x18, 0x3C, 0x18, 0x18, 0x00
.byte 0x00, 0x38, 0x2C, 0x20, 0x20, 0x20, 0x60, 0x60, 0x00
.byte 0x00, 0x3C, 0x24, 0x3C, 0x24, 0x24, 0x6C, 0x6C, 0x00
.byte 0x92, 0x54, 0x38, 0x28, 0xEE, 0x38, 0x54, 0x92, 0x00
   ; code=16, hex=0x10
.byte 0x00, 0x20, 0x30, 0x38, 0x3C, 0x38, 0x30, 0x20, 0x00
.byte 0x00, 0x04, 0x0C, 0x1C, 0x3C, 0x1C, 0x0C, 0x04, 0x00
.byte 0x10, 0x38, 0x7C, 0x10, 0x10, 0x7C, 0x38, 0x10, 0x00
.byte 0x6C, 0x6C, 0x6C, 0x6C, 0x6C, 0x00, 0x6C, 0x6C, 0x00
.byte 0x00, 0x3C, 0x54, 0x54, 0x3C, 0x14, 0x14, 0x14, 0x00
.byte 0x3C, 0x66, 0x60, 0x3C, 0x66, 0x66, 0x3C, 0x06, 0x66
.byte 0x00, 0x00, 0x00, 0x00, 0x00, 0x7C, 0x7C, 0x00, 0x00
.byte 0x10, 0x38, 0x7C, 0x10, 0x10, 0x7C, 0x38, 0x10, 0x7C
.byte 0x00, 0x18, 0x3C, 0x5A, 0x18, 0x18, 0x18, 0x18, 0x00
.byte 0x00, 0x18, 0x18, 0x18, 0x18, 0x5A, 0x3C, 0x18, 0x00
.byte 0x00, 0x00, 0x18, 0x0C, 0x7E, 0x0C, 0x18, 0x00, 0x00
.byte 0x00, 0x00, 0x18, 0x30, 0x7E, 0x30, 0x18, 0x00, 0x00
.byte 0x00, 0x00, 0x00, 0x00, 0x60, 0x60, 0x7C, 0x00, 0x00
.byte 0x00, 0x00, 0x24, 0x66, 0xFE, 0x66, 0x24, 0x00, 0x00
.byte 0x00, 0x00, 0x00, 0x10, 0x38, 0x7C, 0xFE, 0x00, 0x00
.byte 0x00, 0x00, 0x00, 0xFE, 0x7C, 0x38, 0x10, 0x00, 0x00

    ; code=32, hex=0x20, ascii=" "
.byte 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
    ; code=33, hex=0x21, ascii="!"
.byte 0x00, 0x18, 0x18, 0x18, 0x18, 0x00, 0x18, 0x18, 0x00
    ; code=34, hex=0x22, ascii="""
.byte 0x00, 0x6C, 0x6C, 0x44, 0x00, 0x00, 0x00, 0x00, 0x00
    ; code=35, hex=0x23, ascii="#"
.byte 0x00, 0x6C, 0x6C, 0xFE, 0x6C, 0xFE, 0x6C, 0x6C, 0x00
    ; code=36, hex=0x24, ascii="$"
.byte 0x08, 0x18, 0x3C, 0x60, 0x3C, 0x06, 0x7C, 0x18, 0x10
    ; code=37, hex=0x25, ascii="%"
.byte 0x70, 0x52, 0x76, 0x0C, 0x18, 0x3E, 0x6A, 0x0E, 0x00
    ; code=38, hex=0x26, ascii="&"
.byte 0x38, 0x6C, 0x6C, 0x38, 0x6E, 0x6C, 0x6C, 0x3E, 0x00
    ; code=39, hex=0x27, ascii="'"
.byte 0x00, 0x18, 0x18, 0x30, 0x30, 0x00, 0x00, 0x00, 0x00
    ; code=40, hex=0x28, ascii="("
.byte 0x00, 0x0C, 0x18, 0x30, 0x30, 0x30, 0x18, 0x0C, 0x00
    ; code=41, hex=0x29, ascii=")"
.byte 0x00, 0x30, 0x18, 0x0C, 0x0C, 0x0C, 0x18, 0x30, 0x00
    ; code=42, hex=0x2A, ascii="*"
.byte 0x00, 0x44, 0x6C, 0x38, 0xFE, 0x38, 0x6C, 0x44, 0x00
    ; code=43, hex=0x2B, ascii="+"
.byte 0x00, 0x00, 0x18, 0x18, 0x7E, 0x7E, 0x18, 0x18, 0x00
    ; code=44, hex=0x2C, ascii="
.byte 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x30, 0x30, 0x60
    ; code=45, hex=0x2D, ascii="-"
.byte 0x00, 0x00, 0x00, 0x00, 0x7C, 0x7C, 0x00, 0x00, 0x00
    ; code=46, hex=0x2E, ascii="."
.byte 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x30, 0x30, 0x00
    ; code=47, hex=0x2F, ascii="/"
.byte 0x00, 0x0C, 0x0C, 0x18, 0x18, 0x30, 0x30, 0x60, 0x00
    ; code=48, hex=0x30, ascii="0"
.byte 0x00, 0x3C, 0x66, 0x6E, 0x76, 0x66, 0x66, 0x3C, 0x00
.byte 0x00, 0x18, 0x78, 0x18, 0x18, 0x18, 0x18, 0x7E, 0x00
.byte 0x00, 0x3C, 0x66, 0x46, 0x1C, 0x30, 0x66, 0x7E, 0x00
.byte 0x00, 0x3C, 0x66, 0x06, 0x1C, 0x06, 0x66, 0x3C, 0x00
.byte 0x00, 0x0C, 0x1C, 0x3C, 0x6C, 0x7E, 0x0C, 0x1E, 0x00
.byte 0x00, 0x7E, 0x66, 0x60, 0x7C, 0x06, 0x66, 0x3C, 0x00
.byte 0x00, 0x1C, 0x30, 0x60, 0x7C, 0x66, 0x66, 0x3C, 0x00
.byte 0x00, 0x7E, 0x66, 0x06, 0x0C, 0x18, 0x18, 0x18, 0x00
```

```
.byte 0x00, 0x3C, 0x66, 0x66, 0x3C, 0x66, 0x66, 0x3C, 0x00
.byte 0x00, 0x3C, 0x66, 0x66, 0x3E, 0x06, 0x0C, 0x38, 0x00

    ; code=58, hex=0x3A, ascii=":"
.byte 0x00, 0x00, 0x30, 0x30, 0x00, 0x00, 0x30, 0x30, 0x00
    ; code=59, hex=0x3B, ascii=";"
.byte 0x00, 0x00, 0x30, 0x30, 0x00, 0x00, 0x30, 0x30, 0x60
    ; code=60, hex=0x3C, ascii="<"
.byte 0x00, 0x00, 0x18, 0x30, 0x60, 0x30, 0x18, 0x00, 0x00
    ; code=61, hex=0x3D, ascii="="
.byte 0x00, 0x00, 0x00, 0x7C, 0x00, 0x7C, 0x00, 0x00, 0x00
    ; code=62, hex=0x3E, ascii=">"
.byte 0x00, 0x00, 0x30, 0x18, 0x0C, 0x18, 0x30, 0x00, 0x00
    ; code=63, hex=0x3F, ascii="?"
.byte 0x00, 0x3C, 0x66, 0x06, 0x0C, 0x18, 0x00, 0x18, 0x00
    ; code=64, hex=0x40, ascii="@"
.byte 0x00, 0x3C, 0x62, 0x6E, 0x6A, 0x6C, 0x62, 0x3C, 0x00

    ; code=65, hex=0x41, ascii="A"
.byte 0x00, 0x10, 0x38, 0x28, 0x6C, 0x7C, 0xC6, 0xC6, 0x00
.byte 0x00, 0x7C, 0x66, 0x66, 0x7C, 0x66, 0x66, 0x7C, 0x00
.byte 0x00, 0x3C, 0x66, 0x60, 0x60, 0x60, 0x66, 0x3C, 0x00
.byte 0x00, 0x7C, 0x66, 0x66, 0x66, 0x66, 0x66, 0x7C, 0x00
.byte 0x00, 0x7E, 0x66, 0x60, 0x78, 0x60, 0x66, 0x7E, 0x00
.byte 0x00, 0x7E, 0x66, 0x60, 0x7C, 0x60, 0x60, 0x60, 0x00
.byte 0x00, 0x3C, 0x66, 0x60, 0x6E, 0x66, 0x66, 0x3E, 0x00
.byte 0x00, 0x66, 0x66, 0x66, 0x7E, 0x66, 0x66, 0x66, 0x00
.byte 0x00, 0x3C, 0x18, 0x18, 0x18, 0x18, 0x18, 0x3C, 0x00
.byte 0x00, 0x1E, 0x0C, 0x0C, 0x0C, 0x6C, 0x6C, 0x38, 0x00
.byte 0x00, 0x66, 0x6C, 0x6C, 0x78, 0x6C, 0x6C, 0x66, 0x00
.byte 0x00, 0x60, 0x60, 0x60, 0x60, 0x60, 0x66, 0x7E, 0x00
.byte 0x00, 0xC6, 0xC6, 0xEE, 0xFE, 0xD6, 0xD6, 0xD6, 0x00
.byte 0x00, 0x66, 0x76, 0x76, 0x7E, 0x6E, 0x66, 0x66, 0x00
.byte 0x00, 0x3C, 0x66, 0x66, 0x66, 0x66, 0x66, 0x3C, 0x00
.byte 0x00, 0x7C, 0x66, 0x66, 0x66, 0x7C, 0x60, 0x60, 0x00
.byte 0x00, 0x3C, 0x66, 0x66, 0x66, 0x76, 0x6E, 0x3C, 0x06
.byte 0x00, 0x7C, 0x66, 0x66, 0x6C, 0x78, 0x6C, 0x66, 0x00
.byte 0x00, 0x3C, 0x66, 0x60, 0x3C, 0x06, 0x66, 0x3C, 0x00
.byte 0x00, 0x7E, 0x5A, 0x18, 0x18, 0x18, 0x18, 0x3C, 0x00
.byte 0x00, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x3C, 0x00
.byte 0x00, 0xC6, 0xC6, 0x6C, 0x6C, 0x38, 0x38, 0x10, 0x00
.byte 0x00, 0xC6, 0xD6, 0xD6, 0xD6, 0x7C, 0x6C, 0x44, 0x00
.byte 0x00, 0x66, 0x66, 0x3C, 0x18, 0x3C, 0x66, 0x66, 0x00
.byte 0x00, 0x66, 0x66, 0x66, 0x3C, 0x18, 0x18, 0x3C, 0x00
.byte 0x00, 0x7E, 0x66, 0x0C, 0x18, 0x30, 0x66, 0x7E, 0x00

    ; code=91, hex=0x5B, ascii="["
.byte 0x00, 0x3C, 0x30, 0x30, 0x30, 0x30, 0x30, 0x3C, 0x00
    ; code=92, hex=0x5C, ascii="\"
.byte 0x00, 0x60, 0x60, 0x30, 0x30, 0x18, 0x18, 0x0C, 0x00
    ; code=93, hex=0x5D, ascii="]"
.byte 0x00, 0x3C, 0x0C, 0x0C, 0x0C, 0x0C, 0x0C, 0x3C, 0x00
    ; code=94, hex=0x5E, ascii="^"
.byte 0x00, 0x10, 0x38, 0x6C, 0x00, 0x00, 0x00, 0x00, 0x00
    ; code=95, hex=0x5F, ascii="_"
.byte 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x7C, 0x7C
    ; code=96, hex=0x60, ascii="`"
.byte 0x00, 0x30, 0x30, 0x18, 0x18, 0x00, 0x00, 0x00, 0x00

    ; code=97, hex=0x61, ascii="a"
.byte 0x00, 0x00, 0x00, 0x3C, 0x06, 0x3E, 0x66, 0x3A, 0x00
.byte 0x00, 0x60, 0x60, 0x7C, 0x66, 0x66, 0x66, 0x5C, 0x00
.byte 0x00, 0x00, 0x00, 0x3C, 0x66, 0x60, 0x66, 0x3C, 0x00
.byte 0x00, 0x06, 0x06, 0x3E, 0x66, 0x66, 0x66, 0x3A, 0x00
.byte 0x00, 0x00, 0x00, 0x3C, 0x66, 0x7E, 0x60, 0x3C, 0x00
.byte 0x00, 0x1C, 0x36, 0x30, 0x7C, 0x30, 0x30, 0x78, 0x00
.byte 0x00, 0x00, 0x00, 0x3A, 0x66, 0x66, 0x3E, 0x06, 0x3C
.byte 0x00, 0x60, 0x60, 0x6C, 0x76, 0x66, 0x66, 0x66, 0x00
.byte 0x18, 0x18, 0x00, 0x18, 0x38, 0x18, 0x18, 0x3C, 0x00
.byte 0x0C, 0x0C, 0x00, 0x0C, 0x3C, 0x0C, 0x4C, 0x6C, 0x38
.byte 0x00, 0x60, 0x60, 0x66, 0x6C, 0x78, 0x6C, 0x66, 0x00
.byte 0x00, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x3C, 0x00
```

```
.byte 0x00, 0x00, 0x00, 0x6C, 0xFE, 0xD6, 0xD6, 0xD6, 0x00
.byte 0x00, 0x00, 0x00, 0x6C, 0x76, 0x66, 0x66, 0x66, 0x00
.byte 0x00, 0x00, 0x00, 0x3C, 0x66, 0x66, 0x66, 0x3C, 0x00
.byte 0x00, 0x00, 0x00, 0x7C, 0x66, 0x66, 0x76, 0x6C, 0x60
.byte 0x00, 0x00, 0x00, 0x3A, 0x66, 0x66, 0x6E, 0x36, 0x06
.byte 0x00, 0x00, 0x00, 0x26, 0x7E, 0x30, 0x30, 0x78, 0x00
.byte 0x00, 0x00, 0x00, 0x3C, 0x60, 0x3C, 0x06, 0x3C, 0x00
.byte 0x00, 0x10, 0x30, 0xFC, 0x30, 0x30, 0x36, 0x1C, 0x00
.byte 0x00, 0x00, 0x00, 0x66, 0x66, 0x66, 0x6E, 0x36, 0x00
.byte 0x00, 0x00, 0x00, 0xC6, 0xC6, 0x6C, 0x38, 0x10, 0x00
.byte 0x00, 0x00, 0x00, 0xD6, 0xD6, 0x7C, 0x6C, 0x44, 0x00
.byte 0x00, 0x00, 0x00, 0xC6, 0x6C, 0x38, 0x6C, 0xC6, 0x00
.byte 0x00, 0x00, 0x00, 0x66, 0x66, 0x36, 0x1C, 0x6C, 0x38
.byte 0x00, 0x00, 0x00, 0x7E, 0x06, 0x18, 0x30, 0x7E, 0x00

    ; code=123, hex=0x7B, ascii="{"
.byte 0x00, 0x1C, 0x30, 0x30, 0x60, 0x30, 0x30, 0x1C, 0x00
    ; code=124, hex=0x7C, ascii="|"
.byte 0x18, 0x18, 0x18, 0x18, 0x00, 0x18, 0x18, 0x18, 0x00
    ; code=125, hex=0x7D, ascii="}"
.byte 0x00, 0x70, 0x18, 0x18, 0x0C, 0x18, 0x18, 0x70, 0x00
    ; code=126, hex=0x7E, ascii="~"
.byte 0x00, 0x10, 0x3A, 0x6E, 0x04, 0x00, 0x00, 0x00, 0x00
    ; code=127, hex=0x7F
.byte 0x00, 0x00, 0x08, 0x1C, 0x36, 0x62, 0x7E, 0x00, 0x00
    ; code=128, hex=0x80
.byte 0x00, 0x3C, 0x66, 0x60, 0x60, 0x60, 0x66, 0x3C, 0x78
.byte 0x66, 0x66, 0x00, 0x66, 0x66, 0x66, 0x66, 0x3A, 0x00
.byte 0x0C, 0x18, 0x00, 0x3E, 0x62, 0x7E, 0x60, 0x3E, 0x00
.byte 0x1C, 0x36, 0x00, 0x3C, 0x06, 0x3E, 0x66, 0x3E, 0x00
.byte 0x36, 0x36, 0x00, 0x3C, 0x06, 0x3E, 0x66, 0x3E, 0x00
.byte 0x18, 0x0C, 0x00, 0x3C, 0x06, 0x3E, 0x66, 0x3A, 0x00
.byte 0x1C, 0x14, 0x00, 0x3C, 0x06, 0x3E, 0x66, 0x3A, 0x00
.byte 0x00, 0x00, 0x00, 0x1C, 0x36, 0x60, 0x36, 0x1C, 0x78
.byte 0x08, 0x1C, 0x00, 0x3C, 0x66, 0x7E, 0x60, 0x3E, 0x00
.byte 0x66, 0x66, 0x00, 0x3C, 0x66, 0x7E, 0x60, 0x3E, 0x00
.byte 0x18, 0x0C, 0x00, 0x3C, 0x66, 0x7E, 0x60, 0x3E, 0x00
.byte 0x66, 0x66, 0x00, 0x38, 0x18, 0x18, 0x18, 0x3C, 0x00
.byte 0x10, 0x38, 0x00, 0x38, 0x18, 0x18, 0x18, 0x3C, 0x00
.byte 0x30, 0x18, 0x00, 0x38, 0x18, 0x18, 0x18, 0x3C, 0x00
.byte 0xC6, 0x10, 0x38, 0x28, 0x6C, 0x7C, 0xC6, 0xC6, 0x00
.byte 0x38, 0x28, 0x38, 0x28, 0x6C, 0x7C, 0xC6, 0xC6, 0x00
    ; code=144, hex=0x90
.byte 0x1C, 0x30, 0x7E, 0x60, 0x7C, 0x60, 0x60, 0x7E, 0x00
.byte 0x00, 0x00, 0x00, 0x7C, 0x1A, 0x7E, 0xD8, 0x7E, 0x00
.byte 0x00, 0x1E, 0x38, 0x58, 0x5E, 0xF8, 0xD8, 0xDE, 0x00
.byte 0x10, 0x38, 0x00, 0x3C, 0x66, 0x66, 0x66, 0x3C, 0x00
.byte 0x66, 0x66, 0x00, 0x3C, 0x66, 0x66, 0x66, 0x3C, 0x00
.byte 0x18, 0x0C, 0x00, 0x3C, 0x66, 0x66, 0x66, 0x3C, 0x00
.byte 0x08, 0x1C, 0x00, 0x66, 0x66, 0x66, 0x66, 0x3A, 0x00
.byte 0x18, 0x0C, 0x00, 0x66, 0x66, 0x66, 0x66, 0x3A, 0x00
.byte 0x66, 0x66, 0x00, 0x66, 0x66, 0x36, 0x1C, 0x6C, 0x38
.byte 0x66, 0x00, 0x3C, 0x66, 0x66, 0x66, 0x66, 0x3C, 0x00
.byte 0x66, 0x00, 0x66, 0x66, 0x66, 0x66, 0x66, 0x3C, 0x00
.byte 0x08, 0x08, 0x3C, 0x60, 0x60, 0x3C, 0x10, 0x10, 0x00
.byte 0x1C, 0x36, 0x30, 0x30, 0x7C, 0x30, 0x3C, 0x66, 0x00
.byte 0x66, 0x66, 0x3C, 0x18, 0x7E, 0x18, 0x7E, 0x18, 0x00
.byte 0xE0, 0xD0, 0xD0, 0xF4, 0xCC, 0xDE, 0xCC, 0x06, 0x00
.byte 0x0E, 0x18, 0x18, 0x18, 0x7E, 0x18, 0x18, 0x18, 0x70
.byte 0x06, 0x0C, 0x00, 0x3C, 0x06, 0x3E, 0x66, 0x3A, 0x00
.byte 0x0C, 0x18, 0x00, 0x38, 0x18, 0x18, 0x18, 0x3C, 0x00
.byte 0x0C, 0x18, 0x00, 0x3C, 0x66, 0x66, 0x66, 0x3C, 0x00
.byte 0x0C, 0x18, 0x00, 0x66, 0x66, 0x66, 0x66, 0x3A, 0x00
.byte 0x76, 0xDC, 0x00, 0x6C, 0x76, 0x66, 0x66, 0x66, 0x00
.byte 0x76, 0xDC, 0x00, 0x66, 0x76, 0x7E, 0x6E, 0x66, 0x00
.byte 0x38, 0x0C, 0x3C, 0x6C, 0x34, 0x00, 0x7C, 0x00, 0x00
.byte 0x3C, 0x66, 0x66, 0x3C, 0x00, 0x7E, 0x00, 0x00, 0x00
.byte 0x00, 0x18, 0x00, 0x18, 0x30, 0x60, 0x66, 0x3C, 0x00
.byte 0x00, 0x00, 0x00, 0x3C, 0x3C, 0x30, 0x30, 0x00, 0x00
.byte 0x00, 0x00, 0x00, 0x7C, 0x7C, 0x0C, 0x0C, 0x00, 0x00
.byte 0x60, 0x60, 0x60, 0x6E, 0x1A, 0x04, 0x18, 0x1E, 0x00
.byte 0x60, 0x60, 0x60, 0x6C, 0x7C, 0x2C, 0x7C, 0x0C, 0x00
```

```
   .byte 0x00, 0x18, 0x00, 0x18, 0x18, 0x3C, 0x3C, 0x18, 0x00
   .byte 0x00, 0x00, 0x32, 0x66, 0xCC, 0x66, 0x32, 0x00, 0x00
   .byte 0x00, 0x00, 0xCC, 0x66, 0x32, 0x66, 0xCC, 0x00, 0x00
       ; code=176, hex=0xB0
   .byte 0x54, 0x00, 0xAA, 0x00, 0x54, 0x00, 0xAA, 0x00, 0x54
   .byte 0x92, 0x48, 0x24, 0x92, 0x48, 0x24, 0x92, 0x48, 0x24
   .byte 0xAA, 0x54, 0xAA, 0x54, 0xAA, 0x54, 0xAA, 0x54, 0xAA
   .byte 0x10, 0x10, 0x10, 0x10, 0x10, 0x10, 0x10, 0x10, 0x10
   .byte 0x10, 0x10, 0x10, 0x10, 0xF0, 0x10, 0x10, 0x10, 0x10
   .byte 0x10, 0x10, 0x10, 0xF0, 0x10, 0xF0, 0x10, 0x10, 0x10
   .byte 0x28, 0x28, 0x28, 0x28, 0xE8, 0x28, 0x28, 0x28, 0x28
   .byte 0x00, 0x00, 0x00, 0x00, 0xF8, 0x28, 0x28, 0x28, 0x28
   .byte 0x00, 0x00, 0x00, 0xF0, 0x10, 0xF0, 0x10, 0x10, 0x10
   .byte 0x28, 0x28, 0x28, 0xE8, 0x08, 0xE8, 0x28, 0x28, 0x28
   .byte 0x28, 0x28, 0x28, 0x28, 0x28, 0x28, 0x28, 0x28, 0x28
   .byte 0x00, 0x00, 0x00, 0xF8, 0x08, 0xE8, 0x28, 0x28, 0x28
   .byte 0x28, 0x28, 0x28, 0xE8, 0x08, 0xF8, 0x00, 0x00, 0x00
   .byte 0x28, 0x28, 0x28, 0x28, 0xF8, 0x00, 0x00, 0x00, 0x00
   .byte 0x10, 0x10, 0x10, 0xF0, 0x10, 0xF0, 0x00, 0x00, 0x00
   .byte 0x00, 0x00, 0x00, 0x00, 0xF0, 0x10, 0x10, 0x10, 0x10
       ; code=192, hex=0xC0
   .byte 0x10, 0x10, 0x10, 0x10, 0x1E, 0x00, 0x00, 0x00, 0x00
   .byte 0x10, 0x10, 0x10, 0x10, 0xFE, 0x00, 0x00, 0x00, 0x00
   .byte 0x00, 0x00, 0x00, 0x00, 0xFE, 0x10, 0x10, 0x10, 0x10
   .byte 0x10, 0x10, 0x10, 0x10, 0x1E, 0x10, 0x10, 0x10, 0x10
   .byte 0x00, 0x00, 0x00, 0x00, 0xFE, 0x00, 0x00, 0x00, 0x00
   .byte 0x10, 0x10, 0x10, 0x10, 0xFE, 0x10, 0x10, 0x10, 0x10
   .byte 0x10, 0x10, 0x10, 0x1E, 0x10, 0x1E, 0x10, 0x10, 0x10
   .byte 0x28, 0x28, 0x28, 0x28, 0x2E, 0x28, 0x28, 0x28, 0x28
   .byte 0x28, 0x28, 0x28, 0x2E, 0x20, 0x3E, 0x00, 0x00, 0x00
   .byte 0x00, 0x00, 0x00, 0x3E, 0x20, 0x2E, 0x28, 0x28, 0x28
   .byte 0x28, 0x28, 0x28, 0xEE, 0x00, 0xFE, 0x00, 0x00, 0x00
   .byte 0x00, 0x00, 0x00, 0xFE, 0x00, 0xEE, 0x28, 0x28, 0x28
   .byte 0x28, 0x28, 0x28, 0x2E, 0x20, 0x2E, 0x28, 0x28, 0x28
   .byte 0x00, 0x00, 0x00, 0xFE, 0x00, 0xFE, 0x00, 0x00, 0x00
   .byte 0x28, 0x28, 0x28, 0xEE, 0x00, 0xEE, 0x28, 0x28, 0x28
   .byte 0x10, 0x10, 0x10, 0xFE, 0x00, 0xFE, 0x00, 0x00, 0x00
       ; code=208, hex=0xD0
;  .byte 0x28, 0x28, 0x28, 0x28, 0xF8, 0x00, 0x00, 0x00, 0x00
;  .byte 0x00, 0x00, 0x00, 0xFE, 0x00, 0xFE, 0x10, 0x10, 0x10
;  .byte 0x00, 0x00, 0x00, 0x00, 0xF8, 0x28, 0x28, 0x28, 0x28
;  .byte 0x28, 0x28, 0x28, 0x28, 0x3E, 0x00, 0x00, 0x00, 0x00
;  .byte 0x10, 0x10, 0x10, 0x1E, 0x10, 0x1E, 0x00, 0x00, 0x00
;  .byte 0x00, 0x00, 0x00, 0x1E, 0x10, 0x1E, 0x10, 0x10, 0x10
;  .byte 0x00, 0x00, 0x00, 0x00, 0x3E, 0x28, 0x28, 0x28, 0x28
;  .byte 0x28, 0x28, 0x28, 0x28, 0xE8, 0x28, 0x28, 0x28, 0x28
;  .byte 0x10, 0x10, 0x10, 0xFE, 0x10, 0xFE, 0x10, 0x10, 0x10
;  .byte 0x10, 0x10, 0x10, 0x10, 0xF0, 0x00, 0x00, 0x00, 0x00
;  .byte 0x00, 0x00, 0x00, 0x00, 0x1E, 0x10, 0x10, 0x10, 0x10
;  .byte 0xFE, 0xFE, 0xFE, 0xFE, 0xFE, 0xFE, 0xFE, 0xFE, 0xFE
;  .byte 0x00, 0x00, 0x00, 0x00, 0x00, 0xFE, 0xFE, 0xFE, 0xFE
;  .byte 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0
;  .byte 0x0E, 0x0E, 0x0E, 0x0E, 0x0E, 0x0E, 0x0E, 0x0E, 0x0E
;  .byte 0xFE, 0xFE, 0xFE, 0xFE, 0xFE, 0x00, 0x00, 0x00, 0x00
       ; code=224, hex=0xE0
;  .byte 0x00, 0x34, 0x68, 0x68, 0x68, 0x34, 0x00, 0x00, 0x00
;  .byte 0x7C, 0x66, 0x66, 0x6C, 0x66, 0x62, 0x66, 0x6C, 0x08
;  .byte 0x00, 0x7E, 0x62, 0x60, 0x60, 0x60, 0x60, 0x60, 0x00
;  .byte 0x00, 0x00, 0x6C, 0xFE, 0xF6, 0x66, 0x6C, 0x6C, 0x00
;  .byte 0x00, 0xFE, 0xC6, 0x60, 0x38, 0x30, 0x66, 0xFE, 0x00
;  .byte 0x00, 0x00, 0x00, 0x3E, 0x6C, 0x6C, 0x6C, 0x38, 0x00
;  .byte 0x00, 0x00, 0x00, 0x36, 0x36, 0x36, 0x3E, 0x62, 0x40
;  .byte 0x00, 0x00, 0x7A, 0x6A, 0x0E, 0x0C, 0x18, 0x18, 0x00
;  .byte 0x3C, 0x18, 0x3C, 0x66, 0x66, 0x3C, 0x18, 0x3C, 0x00
;  .byte 0x00, 0x3C, 0x66, 0x66, 0x7E, 0x66, 0x66, 0x3C, 0x00
;  .byte 0x00, 0x3C, 0x66, 0x66, 0x66, 0x66, 0x24, 0x66, 0x00
;  .byte 0x00, 0x3C, 0x60, 0x30, 0x18, 0x3C, 0x66, 0x3C, 0x00
;  .byte 0x00, 0x00, 0x00, 0x34, 0x4A, 0x4A, 0x4A, 0x34, 0x00
;  .byte 0x04, 0x3C, 0x66, 0x6E, 0x76, 0x66, 0x3C, 0x10, 0x20
;  .byte 0x1E, 0x30, 0x60, 0x60, 0x7E, 0x60, 0x30, 0x1E, 0x00
;  .byte 0x00, 0x00, 0x3C, 0x66, 0x66, 0x66, 0x66, 0x66, 0x00
       ; code=240, hex=0xF0"
```

```
;  .byte 0x00, 0x7C, 0x00, 0x00, 0x7C, 0x00, 0x00, 0x7C, 0x00
;  .byte 0x00, 0x18, 0x18, 0x7E, 0x18, 0x18, 0x00, 0x7E, 0x00
;  .byte 0x00, 0x30, 0x18, 0x0C, 0x18, 0x30, 0x00, 0x3C, 0x00
;  .byte 0x00, 0x0C, 0x18, 0x30, 0x18, 0x0C, 0x00, 0x3C, 0x00
;  .byte 0x0C, 0x1A, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18
;  .byte 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x58, 0x30
;  .byte 0x00, 0x18, 0x18, 0x00, 0x7E, 0x00, 0x18, 0x18, 0x00
;  .byte 0x00, 0x00, 0x1A, 0x76, 0x00, 0x1A, 0x76, 0x00, 0x00
;  .byte 0x00, 0x3C, 0x66, 0x66, 0x3C, 0x00, 0x00, 0x00, 0x00
;  .byte 0x00, 0x00, 0x18, 0x3C, 0x3C, 0x18, 0x00, 0x00, 0x00
;  .byte 0x00, 0x00, 0x00, 0x18, 0x18, 0x00, 0x00, 0x00, 0x00
;  .byte 0x0E, 0x0C, 0x0C, 0x0C, 0x0C, 0x6C, 0x3C, 0x0C, 0x00
;  .byte 0x00, 0x78, 0x6C, 0x6C, 0x6C, 0x6C, 0x00, 0x00, 0x00
;  .byte 0x00, 0x38, 0x4C, 0x18, 0x30, 0x7C, 0x00, 0x00, 0x00
;  .byte 0x00, 0x7C, 0x7C, 0x7C, 0x7C, 0x00, 0x00, 0x00, 0x00
;  .byte 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00

       .code
       daddi $sp, $0, 0x1000    ; Inicializa el puntero al tope de la pila
                                ; Con 12 bits es 0x1000, con 10 => 0x400
       jal lee_caracter
       sb $v0, car($0)
       dadd $s0, $0, $v0
       jal limpia_pantalla
       dsll  $t0, $s0, 4
       daddi $a0, $t0, caracteres
       jal imprime_caracter
       halt

lee_caracter:
       lwu    $t0, DATA(r0)       ; $t0 = dirección del registro DATA
       lwu    $t1, CONTROL($0)          ; $t1 = dirección del registro CONTROL

       daddi  $t2, $0, txt_car    ; paso a reg. la dir de inicio para salida ascii
       daddi  $t3, $0, 4          ; paso a reg. el valor para salida ascii
       sd     $t2, 0($t0)         ; seteo dir de inicio del mensaje (DATA)
       sd     $t3, 0($t1)         ; seteo salida y se escribe el mensaje (CONTROL)

       daddi  $t3, $0, 9      ; paso a reg. el valor para entrada caracter
       sd     $t3, 0($t1)     ; seteo salida y espera lectura del caracter (CONTROL)
       lbu    $v0, 0($t0)     ; guardo lo que se leyo desde teclado
       jr $ra

limpia_pantalla:
       lwu    $t1, CONTROL($0)          ; $t1 = dirección del registro CONTROL

       daddi  $t0, $0, 7          ; $t0 = 5, función 7 = limpiar salida gráfica
       sd     $t0, 0($t1)         ; CONTROL recibe 7 y limpia la pantalla grafica
       jr $ra

muestra_punto:
       lwu    $t0, DATA($0)       ; $t0 = dirección del registro DATA
       lwu    $t1, CONTROL($0)          ; $t1 = dirección del registro CONTROL

       sw     $a2, 0($t0)         ; DATA recibe el valor del color a pintar
       sb     $a1, 4($t0)         ; DATA+4 recibe el valor de coordenada Y
       sb     $a0, 5($t0)         ; DATA+5 recibe el valor de coordenada X

       daddi  $t2, $0, 5          ; $t2 = 5, función 5 = salida gráfica
       sd     $t2, 0($t1)         ; CONTROL recibe 5 y produce el dibujo del punto
       jr $ra
```

```
imprime_caracter:
       daddi $sp, $sp, -32
       sd $ra, 0($sp)
       sd $s0, 8($sp)
       sd $s1, 16($sp)
       sd $s2, 24($sp)

       dadd $s2, $0, $a0
       daddi $s0, $0, 9  ; Lineas
       daddi $s1, $0, 49 ; Posicion de la linea

sigue: dadd $a0, $0, $s1
       lbu $a1, 0($s2)
       jal dibuja_linea
       daddi $s0, $s0, -1 ; Una linea menos por dibujar
       daddi $s1, $s1, -1 ; Una linea mas abajo se dibuja
       daddi $s2, $s2, 1
       bnez $s0, sigue

       ld $ra, 0($sp)
       ld $s0, 8($sp)
       ld $s1, 16($sp)
       ld $s2, 24($sp)
       daddi $sp, $sp, 32
       jr $ra

dibuja_linea:daddi $sp, $sp, -48
             sd $ra, 0($sp)
             sd $s0, 8($sp)
             sd $s1, 16($sp)
             sd $s2, 24($sp)
             sd $s3, 32($sp)
             sd $s4, 40($sp)

             dadd $s0, $0, $a0
             dadd $s1, $0, $a1
             daddi $s2, $0, 128
             daddi $s3, $0, 7
             daddi $s4, $0, 0

continua:    and $t0, $s2, $s1
             beqz $t0, sin_punto

             dadd $a0, $0, $s4  ; X
             dadd $a1, $0, $s0  ; y
             lwu $a2, color($0)
             jal muestra_punto

 sin_punto:  daddi $s3, $s3, -1
             daddi $s4, $s4, 1
             dsrl $s2, $s2, 1
             bnez $s3, continua

             ld $ra, 0($sp)
             ld $s0, 8($sp)
             ld $s1, 16($sp)
             ld $s2, 24($sp)
             ld $s3, 32($sp)
             ld $s4, 40($sp)
             daddi $sp, $sp, 48
             jr $ra
```

8) El siguiente programa implementa una animación de una pelotita rebotando por la pantalla. Modifíquelo para que en lugar de una pelotita, se muestren simultáneamente varias pelotitas (cinco, por ejemplo), cada una con su posición, dirección y color particular.

```
        .data
CONTROL:        .word32 0x10000
DATA:           .word32 0x10008
color_pelota:   .word32 0x00FF0000  ; Azul
color_fondo:    .word32 0x00FFFFFF  ; Blanco

        .text
        lwu     $s6, CONTROL($zero)
        lwu     $s7, DATA($zero)

        lwu     $v0, color_pelota($zero)
        lwu     $v1, color_fondo($zero)
        daddi   $s0, $zero, 23      ; Coordenada X de la pelota
        daddi   $s1, $zero, 1       ; Coordenada Y de la pelota
        daddi   $s2, $zero, 1       ; Dirección X de la pelota
        daddi   $s3, $zero, 1       ; Dirección Y de la pelota
        daddi   $s4, $zero, 5       ; Comando para dibujar un punto
loop:   sw      $v1, 0($s7)     ; Borra la pelota
        sb      $s0, 4($s7)
        sb      $s1, 5($s7)
        sd      $s4, 0($s6)
        dadd    $s0, $s0, $s2   ; Mueve la pelota en la dirección actual
        dadd    $s1, $s1, $s3
        daddi   $t1, $zero, 48      ; Comprueba que la pelota no esté en la columna de más
        slt     $t0, $t1, $s0   ; a la derecha. Si es así, cambia la dirección en X.
        dsll    $t0, $t0, 1
        dsub    $s2, $s2, $t0
        slt     $t0, $t1, $s1   ; Comprueba que la pelota no esté en la fila de más arriba.
        dsll    $t0, $t0, 1     ; Si es así, cambia la dirección en Y.
        dsub    $s3, $s3, $t0
        slti    $t0, $s0, 1     ; Comprueba que la pelota no esté en la columna de más
        dsll    $t0, $t0, 1     ; a la izquierda. Si es así, cambia la dirección en X.
        dadd    $s2, $s2, $t0
        slti    $t0, $s1, 1     ; Comprueba que la pelota no esté en la fila de más abajo.
        dsll    $t0, $t0, 1     ; Si es así, cambia la dirección en Y.
        dadd    $s3, $s3, $t0
        sw      $v0, 0($s7)     ; Dibuja la pelota.
        sb      $s0, 4($s7)
        sb      $s1, 5($s7)
        sd      $s4, 0($s6)
        daddi   $t0, $zero, 500   ; Hace una demora para que el rebote no sea tan rápido.
demora: daddi   $t0, $t0, -1    ; Esto genera una infinidad de RAW y BTS pero...
        bnez    $t0, demora     ; ¡hay que hacer tiempo igualmente!
        j       loop
```

Resolución:

```
            .data
CONTROL:    .word32 0x10000
DATA:       .word32 0x10008

fondo:      .word32 0x00FFFFFF  ; Blanco

pelota1:    .word32 0x00FF0000  ; Azul
            .byte 23 ; Pos X
            .byte 1 ; Pos Y
            .word 1 ; Direccion X
            .word 1 ; Direccion Y

pelota2:    .word32 0x0000FF00
            .byte 27 ; Pos X
            .byte 21 ; Pos Y
            .word -1 ; Direccion X
            .word -1 ; Direccion Y
```

```
pelota3:    .word32 0x000000FF
            .byte 1 ; Pos X
            .byte 5 ; Pos Y
            .word 1 ; Direccion X
            .word 1 ; Direccion Y


pelota4:    .word32 0x00FF00FF
            .byte 35 ; Pos X
            .byte 35 ; Pos Y
            .word -1 ; Direccion X
            .word 1 ; Direccion Y

pelota5:    .word32 0x00FFFF00
            .byte 1 ; Pos X
            .byte 35 ; Pos Y
            .word 1 ; Direccion X
            .word 1 ; Direccion Y

pelota6:    .word32 0x0000FFFF
            .byte 35 ; Pos X
            .byte 1 ; Pos Y
            .word -1 ; Direccion X
            .word 1 ; Direccion Y

            .code

            daddi $sp, $0, 0x400
loop:       daddi  $a0, $zero, pelota1
            jal    mostrar
            daddi  $a0, $zero, pelota2
            jal    mostrar
            daddi  $a0, $zero, pelota3
            jal    mostrar
            daddi  $a0, $zero, pelota4
            jal    mostrar
            daddi  $a0, $zero, pelota5
            jal    mostrar
            daddi  $a0, $zero, pelota6
            jal    mostrar

            daddi  $t0, $0, 500   ; Hace una demora para que el rebote no sea tan rápido.
demora:     daddi  $t0, $t0, -1   ; Esto genera una infinidad de RAW y BTS pero...
            bnez   $t0, demora    ; hay que hacer tiempo igualmente!
            j loop


mostrar:    daddi  $sp, $sp, -16
            sd     $s0, 0($sp)
            sd     $s1, 8($sp)
            lwu    $s0, CONTROL($0)
            lwu    $s1, DATA($0)

            lwu    $t0, fondo($zero)  ; Recupero fondo
            lwu    $t1, 0($a0)        ; Recupero color
            lbu    $t2, 8($a0)        ; Recupero Coordenada X de la pelota
            lbu    $t3, 16($a0)       ; Recupero Coordenada Y de la pelota
            ld    $t4, 24($a0)        ; Recupero Dirección X de la pelota
            ld    $t5, 32($a0)        ; Recupero Dirección Y de la pelota


            daddi  $t6, $0, 5      ; Comando para dibujar un punto

            ; Borra la pelota
            sw     $t0, 0($s1)  ; Color de Fondo
            sb     $t2, 4($s1)  ; Pos X
            sb     $t3, 5($s1)  ; Pos Y
            sd     $t6, 0($s0)  ; Dibujar

            dadd   $t2, $t2, $t4   ; Mueve la pelota en la dirección actual
            dadd   $t3, $t3, $t5
```

```
daddi  $t7, $0, 48    ; Comprueba que la pelota no está en la columna de más
slt    $t8, $t7, $t2   ; a la derecha. Si es así, cambia la dirección en X.
dsll   $t8, $t8, 1
dsub   $t4, $t4, $t8

slt    $t8, $t7, $t3   ; Comprueba que la pelota no está en la fila
                       ; de más arriba.
dsll   $t8, $t8, 1     ; Si es así, cambia la dirección en Y.
dsub   $t5, $t5, $t8

slti   $t8, $t2, 1     ; Comprueba que la pelota no está en la columna de más
dsll   $t8, $t8, 1     ; a la izquierda. Si es así, cambia la dirección en X.
dadd   $t4, $t4, $t8

slti   $t8, $t3, 1     ; Comprueba que la pelota no está en la fila de más
dsll   $t8, $t8, 1     ; abajo. Si es así, cambia la dirección en Y.
dadd   $t5, $t5, $t8

sw     $t1, 0($s1)     ; Dibuja la pelota.
sb     $t2, 4($s1)
sb     $t3, 5($s1)
sd     $t6, 0($s0)

sb    $t2, 8($a0)      ; Guardo Coordenada X de la pelota
sb    $t3, 16($a0)      ; Guardo Coordenada Y de la pelota
sd    $t4, 24($a0)      ; Guardo Dirección X de la pelota
sd    $t5, 32($a0)      ; Guardo Dirección Y de la pelota

ld     $s0, 0($sp)
ld     $s1, 8($sp)
daddi  $sp, $sp, 16
jr     $ra
```

9) Escriba un programa que le permita dibujar en la pantalla gráfica de la terminal. Deberá mostrar un cursor (representado por un punto de un color particular) que pueda desplazarse por la pantalla usando las teclas 'a', 's', 'd' y 'w' para ir a la izquierda, abajo, a la derecha y arriba respectivamente. Usando la barra espaciadora se alternará entre modo desplazamiento (el cursor pasa por arriba de lo dibujado sin alterarlo) y modo dibujo (cada punto por el que el cursor pasa quedará pintado del color seleccionado). Las teclas del '1' al '8' se usarán para elegir uno entre los ocho colores disponibles para pintar.

Observaciones: Para poder implementar este programa, se necesitará almacenar en la memoria la imagen completa de la pantalla gráfica.

Si cada punto está representado por un byte, se necesitarán 50x50x1 = 2500 bytes. El simulador WinMIPS64 viene configurado para usar un bus de datos de 10 bits, por lo que la memoria disponible estará acotada a 2^{10}=1024 bytes.

Para poder almacenar la imagen, será necesario configurar el simulador para usar un bus de datos de 12 bits, ya que 2^{12}=4096 bytes, los que si resultarán suficientes. La configuración se logra yendo al menú "Configure → Architecture" y poniendo "Data Address Bus" en 12 bits en lugar de los 10 bits que trae por defecto.

Resolución:

```
          .data
.data
        CONTROL:        .word32 0x10000
        DATA:           .word32 0x10008

        color_punto:    .byte 0x20,0x40, 0x80, 0   ; Azul oscuro

        colores:        .byte 0xFF, 0xFF, 0xFF, 0 ; 1 = Blanco
                        .byte 0xFF, 0x00, 0x00, 0 ; 2 = Rojo
                        .byte 0x00, 0xFF, 0x00, 0 ; 3 - Verde
                        .byte 0x00, 0x00, 0xFF, 0 ; 4 - Azul
                        .byte 0xFF, 0xFF, 0x00, 0 ; 5 - Amarillo
                        .byte 0xFF, 0x00, 0xFF, 0 ; 6 - Violeta
                        .byte 0x80, 0x80, 0x80, 0 ; 7 - Gris
                        .byte 0x00, 0x00, 0x00, 0 ; 8 - Negro

        imagen:         .space 2500

.text
        daddi   $s0, $0, 0              ; Coordenada X del puntero.
        daddi   $s1, $0, 0              ; Coordenada Y del puntero.
        daddi   $s2, $0, 3              ; Índice del color actual.
        dadd    $s3, $0, $0             ; Dibujando? Inicialmente, no.
        daddi   $s4, $0, 50             ; Tamaño horizontal

        lwu     $s5, color_punto($0)
        lwu     $s6, CONTROL($0)
        lwu     $s7, DATA($0)

        daddi   $t1, $0, 97             ; Código ASCII de la 'a' -> izquierda
        daddi   $t2, $0, 115            ; Código ASCII de la 's' -> abajo
        daddi   $t3, $0, 100            ; Código ASCII de la 'd' -> derecha
        daddi   $t4, $0, 119            ; Código ASCII de la 'w' -> arriba
        daddi   $t7, $0, 32             ; Código ASCII de ' ' -> pintar/no pintar

        daddi   $t6, $0, 49             ; Límite de la pantalla

        daddi   $t5, $0, 5              ; Códigos de CONTROL
        daddi   $t9, $0, 9

loop:   dmul    $a0, $s1, $s4
```

```
        sw      $s5, 0($s7)          ; Dibuja el cursor en la posición actual.
        sb      $s1, 4($s7)
        sb      $s0, 5($s7)
        sd      $t5, 0($s6)

        sd      $t9, 0($s6)          ; Lee una tecla.
        lbu     $t0, 0($s7)

        dadd    $a0, $a0, $s0        ; Lee el color de la imagen sobre la que
        lbu     $a1, imagen($a0)     ; el cursor estaba parado.

        beqz    $s3, pasa
        ; Si está dibujando, pinta con el color actual.
        dadd $a1, $0, $s2

 pasa:  dsll    $a2, $a1, 3       ; Accede al color RGB a partir del índice almacenado.
                                  ; Desplazamos 3 para multiplicarlo por 8, y obtener
                                  ; el desplazamiento en la tabla de colores.
                                  ; Cada color esta alineado cada 8 bytes.
        lwu     $a3, colores($a2)

        sw      $a3, 0($s7)          ; Pinta el pixel en pantalla
        sd      $t5, 0($s6)

        sb      $a1, imagen($a0)     ; Almacena el índice del color en la imagen

        beq     $t0, $t1, mover_izquierda
        beq     $t0, $t2, mover_abajo
        beq     $t0, $t3, mover_derecha
        beq     $t0, $t4, mover_arriba
        beq     $t0, $t7, pintar_no_pintar

        slti    $v0, $t0, 49         ; Compara la tecla con el '1'.
        slti    $v1, $t0, 57         ; Compara la tecla con el '9'.

        daddi   $a1, $t0, -49

        bnez    $v0, loop
        movn    $s2, $a1, $v1

        j       loop

mover_izquierda:
        slt     $t8, $r0, $s0
        dsub    $s0, $s0, $t8
        j       loop

mover_derecha:
        slt     $t8, $s0, $t6
        dadd    $s0, $s0, $t8
        j       loop

mover_abajo:
        slt     $t8, $r0, $s1
        dsub    $s1, $s1, $t8
        j       loop

mover_arriba:
        slt     $t8, $s1, $t6
        dadd    $s1, $s1, $t8
        j       loop

pintar_no_pintar:
        xori    $s3, $s3, 1
        j       loop
```