Framebuffer para ARCSim

M. Kurban, N. Wolovick

28 de mayo de 2001

1 Introducción

Extendemos el simulador ARCSim de la arquitectura “A RISC Computer” [1] con la incorporación de un *framebuffer*. Este buffer de marco, no es más que una zona de memoria con representación gráfica en una pantalla bidimensional de puntos discretos (*pixels*). La dirección inicial de la memoria gráfica es $Base = (10000000)_{16}$.

La única descripción que precisamos para poder utilizar este *framebuffer*, es la función que a partir de la posición de un punto y su color, genera la dirección de memoria y su contenido.

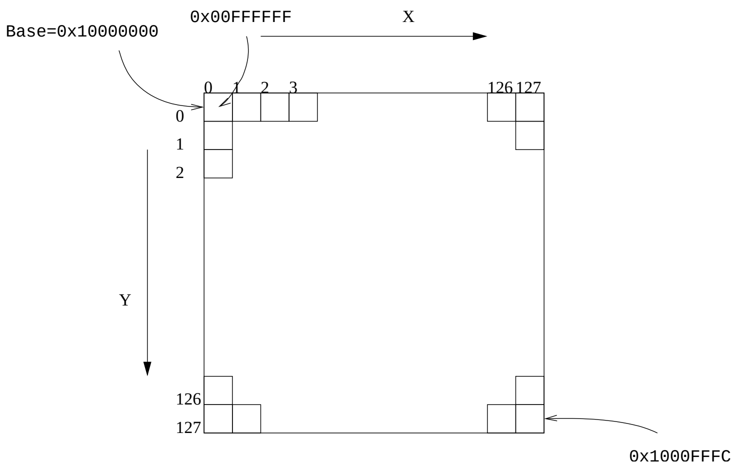
$$\begin{aligned} pixpos2fbAddr(x, y) &= Base + 4(y\ MaxX + x) \\ pixcolor2fbWord(r, g, b) &= 2^{16}r + 2^8 g + b \end{aligned}$$

donde (x, y) es la posición del punto a dibujar y (r, g, b) es la proporción de colores primarios para el punto. Todas estos valores son números naturales con las restricciones

$$0 \le x < MaxX \wedge 0 \le y < MaxY \wedge 0 \le r, g, b < 2^8$$

Básicamente cada pixel es una posición de memoria, donde la palabra apuntada por esa posición determina el color del punto. Como sólo utilizamos 3 bytes para representar el color, en cada palabra-pixel tenemos 1 byte sin utilizar.

El siguiente gráfico muestra la representación en pantalla del *framebuffer*, donde $MaxX = MaxY = 128$, es decir que nuestro display es un marco cuadrado de 128×128 elementos. Se ve también el contenido de la primera posición de memoria, que genera en pantalla un punto de color blanco.



2 Ejemplos

Veamos un programa que pone en blanco el punto de la esquina superior izquierda del marco.

ejap1.asm

```
        .begin
        .org 2048

        or      %r0, 1, %r1
        sll     %r1, 28, %r1    !%r1=0x10000000, Base
        add     %r0, 0xFF, %r3
        or      %r0, %r3, %r2
        sll     %r3, 8, %r3
        or      %r3, %r2, %r2
        sll     %r3, 8, %r3
        or      %r3, %r2, %r2   !%r2=0x00FFFFFF color

        st      %r2, %r1
        halt

        .end
```

Para dibujar puntos en cualquier lugar puede ser práctico escribir una rutina que hace los cálculos de $pixpos2fbAddr$. El siguiente programa dibuja la diagonal $x = y$ en color azul.

ejap2.asm

```
        .begin
        .org 2048

!definicion de constantes inmediatas
MaxX    .equ    128
MaxY    .equ    128
log2MX  .equ    7
log2MY  .equ    7

        sethi   0, %r1          ! %r1=x=0
        or      %r0, 0xFF, %r3  ! %r3=color=Azul

loop:   or      %r1, %r0, %r2   ! %r2=y=x
        call    plot
        add     %r1, 1, %r1     ! inc(x)
        subcc   %r1, MaxX, %r0
        bne     loop            ! while x<MaxX

        halt

plot:   or      %r0, 1, %r8
        sll     %r8, 28, %r8    ! Base
        sll     %r2, log2MX, %r9! MaxX.y
        add     %r9, %r1, %r9   ! MaxX.y+x
        sll     %r9, 2, %r9     ! 4(MaxX.y+x)
        add     %r8, %r9, %r8   ! Base+4(MaxX.y+x)
        st      %r3, %r8        ! M[Base+4(MaxX.y+x)] <- color
        jmpl    %r15+4, %r0     ! retorno

        end
```

Notamos que la rutina `plot` utiliza la convención de pasaje de parámetros por registros, que a pesar de ser restringida pues no podemos hacer llamados anidados y tampoco podemos modificar los registros del programa que llama, resulta bastante sencilla. También vemos como en `plot` utilizamos corrimientos a la derecha para multiplicar por potencias de dos.

También podríamos haber escrito un obscuro programa como el siguiente que resulta equivalente al anterior respecto al estado en que deja al *framebuffer*.

ejap3.asm

```
        .begin
        .org 2048

        sethi   1, %r1
        sll     %r1, 18, %r1    ! %r1=Base=0x10000000
        or      %r0, 0xFF, %r2  ! %r2=color=Azul
        or      %r0, 0x80, %r3  ! %r3=cant.puntos

loop:   st      %r2, %r1
        add     %r1, 516, %r1   ! 516=4.(128+1)
        add     %r3, -1, %r3
        andcc   %r3, %r3, %r0
        bne     loop

        halt

        .end
```

En el otro extremo podemos dar el siguiente programa que dibuja dos líneas verticales en degradé de azul. La rutina `plot` utiliza la convención de pasaje de parámetros por pila y resguarda todos los registros que modifica.

ejap4.asm

```
        .begin
        .org    2048

!definicion de constantes inmediatas
MaxX    .equ    128
MaxY    .equ    128
log2MX  .equ    7
log2MY  .equ    7

        or      %r0, MaxY, %r1
        srl     %r1, 2, %r1     ! %r1=y1=MaxY/4
        add     %r1, %r1, %r2
        add     %r2, %r1, %r2   ! %r2=y2=3MaxY/4
        or      %r0, MaxX, %r3
        add     %r3, -1, %r3    ! %r3=x=MaxX-1
        ba      endloop

loop:   add     %sp, -4, %sp
        st      %r3, %sp        ! push x
        add     %sp, -4, %sp
        st      %r2, %sp        ! push y
        add     %sp, -4, %sp
        st      %r3, %sp        ! push color
        call    plot

        add     %sp, -4, %sp
        st      %r3, %sp        ! push x
        add     %sp, -4, %sp
        st      %r1, %sp        ! push y
        add     %sp, -4, %sp
        st      %r3, %sp        ! push color
        call    plot

        add     %r3, -1, %r3    ! dec(x)
endloop:andcc   %r3, %r3, %r0
```

```
36          bpos    loop            ! while x>=0
37
38          halt
39
40  plot:   add     %sp, -4, %sp
41          st      %r15, %sp       ! push %r15
42          add     %sp, -4, %sp
43          st      %r1, %sp        ! push %r1
44          add     %sp, -4, %sp
45          st      %r2, %sp        ! push %r2
46
47          ld      %sp+16, %r1     ! y
48          sll     %r1, log2MX, %r1  ! MaxX.y
49          ld      %sp+20, %r2     ! x
50          add     %r1, %r2, %r1   ! MaxX.y+x
51          sll     %r1, 2, %r1     ! 4(MaxX.y+x)
52          or      %r0, 1, %r2
53          sll     %r2, 28, %r2    ! Base
54          add     %r1, %r2, %r1   ! Base+4(MaxX.y+x)
55          ld      %sp+12, %r2     ! color
56          st      %r2, %r1        ! M[Base+4(MaxX.y+x)] <- color
57
58          ld      %sp, %r2
59          add     %sp, 4, %sp     ! pop %r2
60          ld      %sp, %r1
61          add     %sp, 4, %sp     ! pop %r1
62          ld      %sp, %r15
63          add     %sp, 4, %sp     ! pop %r15, fin recuperacion registros
64
65          add     %sp, 12, %sp    ! saco parametros de llamada, equiv. (3*)pop %r0
66          jmpl    %r15+4, %r0     ! retorno
67
68          .end
```

Remarcamos varios puntos de este programa. En primer lugar vemos que el lazo principal comienza con un salto hacia el final de él, donde la condición se compara, esto permite reducir la cantidad de saltos que se efectúan en una construcción `while` [1, pag.159]. También vemos que en la rutina `plot` salvaguardamos todos los registros que se van a modificar a fin de ser transparente para el programa que la llame. No hacemos uso de macros, simplemente porque la versión actual de ARCAssembler no provee esta construcción. A primera vista parecería que no es necesario salvaguardar `%r15`, pero lo hacemos a para mantener una estructura que permitiría hacer llamados a rutinas dentro de `plot`. Finalmente, notamos que al salir de `plot`, la instrucción de la línea 65 limpia el marco de pila de los parámetros de entrada.

Referencias

[1] M. Murdocca and V. P. Heuring, *Principles of Computer Architecture*, Prentice-Hall, 2000.

[2] J. D. Foley and A. Van Dam, *Computer Graphics, principles and practice*, Second Edition, Addison Wesley, 1990.