CS2261 Media Device Architecture

* Quiz Monday
* No PowerPoint for this recitation

Mode 0

* For Modes 3 and 4, we used BG02
* For Mode 0, we can use any/multiple backgrounds from BG0-BG03
  + Typically we use Background 0 because it’s the front-most background
* However, it’s necessary to set up each background to how you want it to look
  + Backgrounds connect tiles
* For our lab, we enabled Background 0 in the goToStart() function instead of the initialize() function
* In order to form the background, we have to tell the program where the tiles and maps are stored
  + They are located in Video Buffer like before
  + But, they are overlapping
  + Four blocks of memory for character blocks (CHARBLOCKS) which is where your tiles go
  + 32 blocks of memory for screen blocks (SCREENBLOCKS) where the maps are stored
    - Videobuffer: Mode0. 64KB big

|  |  |
| --- | --- |
| 1. Character Block 0 | 0. Screen blocks |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
| 1. CB1 |  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
| 1. CB2 |  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
| 1. CB3 |  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

* If the data for tiles and maps overlap, one will be deleted since they can’t coexist in the same spot
* The solution it to begin storing the map from screen block 31 (the end) and work our way up, and store tiles starting at character block 0 and work our way down
  + Videobuffer has 64 KB of memory.
* To set up BG0, write code to tell the register for background 0 where the info for tiles and maps is stored. Also enable Mode 0 and BG0:
  + REG\_DISPCTL = MODE0 | BG0\_ENABLE;
  + REG\_BG0CNT = BG\_8BPP | BG\_CHARBLOCK(0) | BG\_SCREENBLOCK(31);
* USENTI note: when saving images from USENTI as bmps, save with 8(256 colors) option in the little box that pops up, NEVER as 4(16 colors) option, no matter whether you are using 4bpp or 8bpp tiles
* Use DMA (the DMANow function in application for labs and HWs) to put the tiles and map into VB
  + DMA options: increment the source and the destination
* USENTI gives you a length for the map, palette, and the tiles, but this length is in bytes and we transfer data at 16 bits at a time
  + Halve the length it gives you for DMA to move the data properly
  + Example from lab:
  + DMANow(3, lab07Pal, PALETTE, lab07PalLen/2);
    - lab07PalLen/2 wull always be 256, so you can just put 256 if you want
  + DMANow(3, lab07Tiles, &CHARBLOCK[0], lab07TilesLen/2);
  + DMANow(3, bgMap, &SCREENBLOCK[31], lab07MapLen/2);
* Size in mode 0 is important
  + Videobuffer has a total of 64 KB space, which is 65,526 bytes
    - If you go over, you’ll be writing over other stuff
  + One screenblock is 1024 shorts big, which is 2048 bytes
  + One tile’s size varies depending on if it’s 8bpp or 4bpp
    - 8bpp: 8p \* 8p = 64p = 64 bytes
    - 4bpp: 8p \* 8p = 32 bytes because its 4 bits per pixel
      * So 4bpp tiles are smaller
  + How many distinct tiles can we have in a map?
    - 210 = 1024
      * The 10 comes from the bits 0-9 that hold tile index
    - 8bpp: 1024 (max number of tiles) \* 64 = 64KB
    - HOWEVER if you have this max number of 8bpp tiles, you will take up the entire space in videobuffer and there will be no room for the maps
    - Use 4bpp: 1024 \* 32 = 32KB (half VB)
      * Never go past CB1
* The screen is 20 tiles by 30 tiles
* To move around on the screen, use offset
  + Vertical and horizontal offset
  + HOFF and VOFF
    - // Background Offset Registers
    - #define REG\_BG0HOFF (\*(volatile unsigned short \*)0x04000010)
    - #define REG\_BG0VOFF (\*(volatile unsigned short \*)0x04000012)
  + These are write-only memory though, so make variables that directly correspond to the registers, and use those to read where you are and to write to memory and move around
    - int hOff;
    - int vOff;
  + if (BUTTON\_HELD(BUTTON\_UP)) {
  + vOff--;
  + }
  + if (BUTTON\_HELD(BUTTON\_DOWN)) {
  + vOff++;
  + }
  + if (BUTTON\_HELD(BUTTON\_RIGHT)) {
  + hOff++;
  + }
  + if (BUTTON\_HELD(BUTTON\_LEFT)) {
  + hOff--;
  + }
  + // Update the Background 0 Offset Registers with those variables
  + REG\_BG0VOFF = vOff;
  + REG\_BG0HOFF = hOff;