CS2261 Media Device Architecture

Announcements:

* Quiz next Monday
  + Won’t be asked to code about DMA but might include some concepts
* HW3 posted on T-Square, due February 13th

Notes:

* Started off reviewing the confusion last class about pointers and the sizeof keyword
  + int \*ip;
  + sizeof(\*ip); 🡪 4, this is the size of the type of the pointer, an int, which is 4 bytes
  + sizeof(ip); 🡪 4, this is the size of the pointer. Pointers are 4 bytes
* Pop Quiz:

u16 shrtArr[5] = {0, 1, 2, 3, 4};

\*(shrtArr + 2) = 17;

What does the array contain now?

4. {0, 1, 17, 3, 4}

* + - shrtArr points to the start of the array. Adding 2 shifts where it points to over two slots to now point to the 2nd index (element “2”) and changes its value to 17
  + Is \*(shrtArr + 2) = 17; the same as shrtArr[2] = 17; ?
    - Yes, the brackets indicate the use of pointer arithmetic. shrtArr is shifted to point 2 slots over and change the value to 17
      * I say slots because the memory size for one element differs based on the type. In this case, an unsigned short

int \*Pointer;

int intArr[5] = {0, 1, 2, 3, 4};

Pointer = intArr;

Pointer++;

\*Pointer = 43;

Pointer++;

\*Pointer = -3;

Array now contains:

{0, 43, -3, 3, 4}

* When Pointer is incremented and then assigned a value, pointer arithmetic moves it to the next index of the array and assigns values there
* If the Pointer type was an unsigned short (unsigned short \*Pointer), then the value -3 would not enter the array. Unsigned means not negative, so entering a negative value wouldn’t work
  + In the slides, unsigned short is typedefed as u16

DMA

* Speed
  + A GBA is slower than a PC, with less memory, slower processor
    - We need workarounds to avoid these limitations
    - Have to keep an eye on efficiency, memory space in GBA
  + So far, what we’ve done with the rectangles, making them bigger slows them down
    - Found a bit of a solution with VBlank
  + Currently, the GBA has a lot of Central Processing Unit cycles and repeatedly accessing the CPU to write to video memory slows it down, as well as these:
    - Copying a block of memory from one place to another
      * We do this repeatedly, like copying a color value to a bunch of locations on a screen that map to a memory locations (setPixel)
    - Filling in a block of memory with a single value
    - We avoid this with DMA
  + DMA – Direct Memory Access allows us to avoid the CPU and increases efficiency.
    - Avoiding the processor and going directly to memory, allows things to move faster
    - Achieved by a circuitry that allows you to access/ write to memory without using CPU
      * Don’t have to tell the CPU what microinstructions needed to be executed to move something from one location to another
      * C is a low-level language, and during compilation, it gets even lower level. Copying from memory involves microinstructions. Avoiding this speeds things up
    - Hardware supported data copy
      * From one memory location to another avoiding CPU.
      * Up to 10x as fast as array copies
      * You set it up, the CPU is halted, data is transferred, and CPU gains back control
      * CPU doesn’t interrupt, and this is a fast process that won’t slow things down overall.
  + The idea behind this is to move something at source address into destination address some number of times represented by counter
    - You avoid the processor
    - The source and destination hold addresses, important to remember

Counter

Destination address

Source address

* DMA has three diff registers, memory storage units
  + They have addresses
  + All work together and avoid the processor
  + Source address: what your trying to move or copy
  + Destination: where you want to put it
  + Counter: how much copying you want to do
* 3 registers per channel
* Example: painting 50 contiguous pixels green
  + DMA about copying blocks of memory, has to be contiguous
  + Here we are copying the color green
  + Source Address: (address to somewhere the storage value is the color green) Have the bits for green stored at some address and put in this register
    - Tell DMA this is the address/source where you can find bits for green
  + Destination Address: Address of starting pixel to paint. Incrementor logic built around register to shift to next address
    - Address where you want to start putting stuff, in this case, the starting pixel
    - Use the offset formula to figure out where we want to put things
  + Counter: Circuitry decrements counter down from 50. When at 0, the copying stops.
    - Say we wanted to paint 5- pixels green, coy green 50 times over
    - Give counter 50, decrements after each transfer
  + “All done without repeating FETCH, DECODE”
    - Language broken down into lower level assembly language that the processor understands,
* DMA:
  + I/O device
  + Memory mapped – We can access source and destination registers using standard memory addresses
  + Source and destination registers can increment and decrement
    - Depends on problem we are trying to solve
    - E.g. reverse a picture
  + Counter register is more complicated as it can control what direction to go
* 4 DMAs
  + • 0 is for internal memory in the chip (won’t use)
  + 1 and 2 are for sound
    - we will use 3 next
* Using DMA
  + Source
    - REG\_DMAxSAD (x = 0, 1, 2, 3)
    - The location of the data that will be copied
  + Destination
    - REG\_DMAxDAD
    - Where to copy the data to
  + Amount
    - REG\_DMAxCNT (DMA control)
    - How much to copy plus options
* REG\_DMAxCNT
  + Recall they are 32 bit registers
  + Lower 16 bits contain amount to transfer
  + Upper 16 bits contain other options
    - Turn on a DMA channel
    - When to perform the DMA
    - Which direction to go
    - How much to copy at a time

Coding Example (on slides)