SPIM Homework: ENEE350
Due Date: March 13th 2007

Directions for PCSpim: (For the other versions, you will have to do some research yourself)

1.) Open up notepad and write your program. Make sure that your program begins with label "main:" (it must be written in lower case). When you're done, save it with a name of your choice making sure to give it a ".s" extension and save as type "All Files".

2.) Start up PCSpim. You will see a console window and PCspim window. Ignore the console for now. Click File then open (or just the open button) and load the file you just created.

NOTE: If you ever load a file again, PCSpim will ask you to "Clear Program and reinitialize simulator before loading", just click yes.

3.) Now it's time to run your program. You have two options:

a) **Single stepping:** Click F10 six times and you will arrive at the start of your program (it's main label). By clicking F10, you will single step through your assemble code executing one instruction on every press. This is the only way for you to see the changes in the data segment and the general purpose registers. This is the only way to debug your program.

b) **Running:** Click the GO button. From here you can stop by clicking the STOP button. You can only read registers and memory when your program is stopped. In general, running is useless unless you want your program to run continuously or it is very long.

Some notes:

PCspim gives the user a few other options. They can be found in the Simulator tab next to file. One can insert breakpoints and also change the contents of memory and the registers with the "Set Value" option. In "Settings", you can also change several things. By default, values are represented in the data segment and registers in hexadecimal. Unchecking the options, will have them shown in decimal. There are several other options and settings that you may want to explore on your own.

Assignment:

1. Return to your program for problem #2 from the first HW. Write it up in notepad and load it into PCSpim. However, in this case, you need to add some initialization code. Using whatever means necessary, give these initial values to the corresponding register: b = 26 c = 3.

Single-step through the program and note the value of the program counter (PC). Additionally, try and logically follow how the values in the registers change on each instruction.
When you have reached the end of the program (there is nothing left to execute), save the log file. Please print the log file and circle your resulting value. If it isn't correct, you are expected to fix the program by using PCspim.

Please submit the log file. Additionally, answer the following questions:
What does the program counter correspond to?
Where address does your program start at in memory? Please circle this address as well in the log file.

You can erase all the parts of the log file just before REGISTERS. They contain no relevance so this will save printer paper and ink.

2. Your assignment is to write a program to take keystrokes from the keyboard and output them to spim's console window continuously. The details for doing so (which were modified slightly from "SPIM S20: A MIPS R2000 Simulator") can be found below. This will take some reading and understanding on your part. Although this may seem hard to achieve, the program is relatively simple and completed in 8 or so lines of assembly. You will want to debug the program by single-stepping, but in order to really perform memory i/o you need to run. See the directions below for what the difference is.

Hand in the program and a screenshot of your name written to the console window.

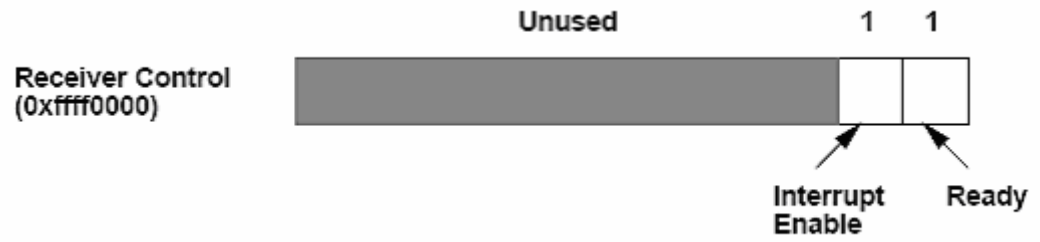
Input and Output with SPIM using the console window:

In addition to simulating the basic operation of the CPU and operating system, SPIM also simulates a memory-mapped terminal connected to the machine. When a program is running, SPIM connects its own terminal (or a separate console window in xspim) to the processor. The program can read characters that you type while the processor is running. Similarly, if SPIM executes instructions to write characters to the terminal, the characters will appear on SPIM's terminal or console window.

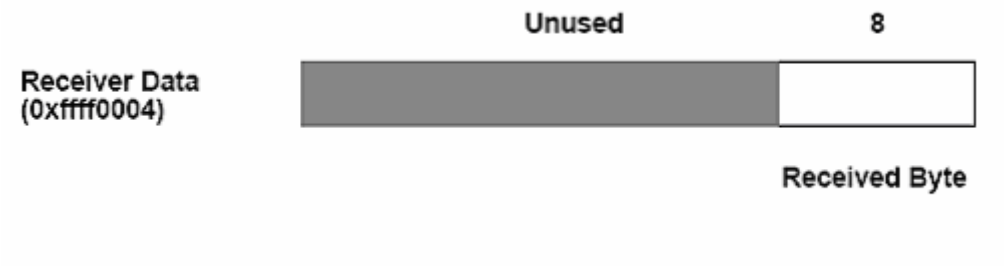
To use memory-mapped IO, spim or xspim must be started with the -mapped io ag. In PCspim, go to the simulator tab then settings. Make sure the box for "Memory Mapped I/O" has a check mark (It should by default). The terminal device consists of two independent units: a receiver and a transmitter. The receiver unit reads characters from the keyboard as they are typed. The transmitter unit writes characters to the terminal's display. The two units are completely independent. This means, for example, that characters typed at the keyboard are not automatically echoed" on the display. Instead, the

processor must get an input character from the receiver and re-transmit it to echo it.

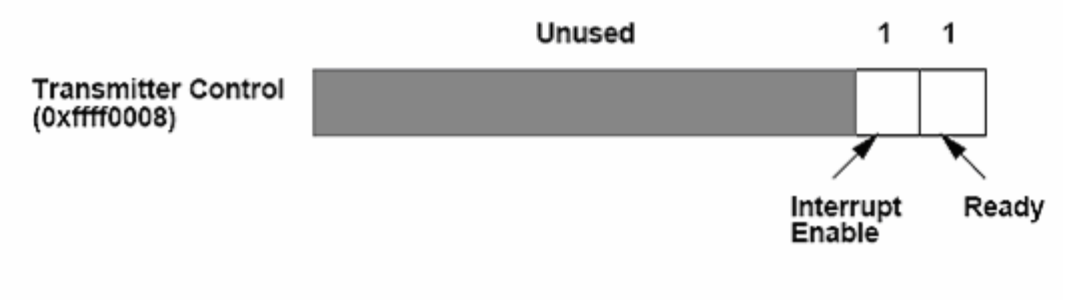
The processor accesses the terminal using four memory-mapped device registers, as shown in the figure. “Memory-mapped" means that each register appears as a special memory location. The **Receiver Control Register** is at location 0xffff0000; For this assignment, you only need to access bit 0 (called the “ready bit”): if it is one it means that a character has arrived from the keyboard but has not yet been read from the receiver data register. The ready bit is read-only: attempts to write to this bit are ignored. The ready bit changes automatically from zero to one when a character is typed at the keyboard, and it changes automatically from one to zero when the character is read from the receiver data register.



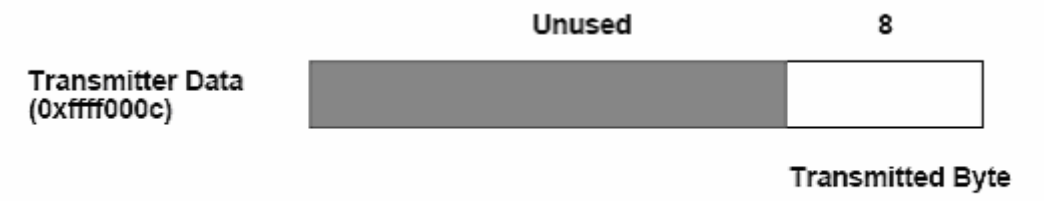
The second terminal device register is the **Receiver Data Register** (at address 0xffff0004). The low-order eight bits of this register contain the last character typed on the keyboard, and all the other bits contain zeroes. This register is read-only and only changes value when a new character is typed on the keyboard. Reading the Receiver Data Register causes the ready bit in the Receiver Control Register to be reset to zero.



The third terminal device register is the **Transmitter Control Register** (at address 0xffff0008). Only the low-order two bits of this register are used, and they behave much like the corresponding bits of the Receiver Control Register. Bit 0 is called “ready" and is read-only. If it is one it means the transmitter is ready to accept a new character for output. If it is zero it means the transmitter is still busy outputting the previous character given to it. You need not be concerned with Bit 1 for this assignment.



The final device register is the **Transmitter Data Register** (at address 0xffff000c). When it is written, the low-order eight bits are taken as an ASCII character to output to the display. When the Transmitter Data Register is written, the ready bit in the Transmitter Control Register will be reset to zero. The bit will stay zero until enough time has elapsed to transmit the character to the terminal; then the ready bit will be set back to one again. The Transmitter Data Register should only be written when the ready bit of the Transmitter Control Register is one; if the transmitter isn't ready then writes to the Transmitter Data Register are ignored (the write appears to succeed but the character will not be output).



In real computers it takes time to send characters over the serial lines that connect terminals to computers. These time lags are simulated by SPIM. For example, after the transmitter starts transmitting a character, the transmitter's ready bit will become zero for a while. SPIM measures this time in instructions executed, not in real clock time. This means that the transmitter will not become ready again until the processor has executed a certain number of instructions. If you stop the machine and look at the ready bit using SPIM, it will not change. However, if you let the machine run then the bit will eventually change back to one.