COMP 273
Assignment #3
Due: April 16, 2013 at 23:55 on My Courses

Assembly Programming

We will not be having a 4th assignment this year, so this assignment is 1.5 assignments in length. The grade distribution will also be 1.5 the weight compared to the other assignments, so it will be graded out of 30 points instead of the regular 20 points. This assignment has three questions.

QUESTION 1: Assembler Programming

Write a program that uses a static array of integer numbers already present in the .data area and prints those numbers to the screen from cell 0 to cell n and then from cell n to cell 0.

The program will have a main function, a print function, and one global static array in the program's static data area (.data area). Your MIPS program must function and simulate the following C code:

```
int array[10] = {10, 5, 2, 20, 20, -5, 3, 19, 9, 1};

void main(void)
{
      printArray(0, 9);
      printArray(9, 0);
}

void printArray(int startIndex, int endIndex)
{
      // prints out the numbers in the order starting from startIndex
      // and ending at endIndex
}
```

You must construct this program using the SPIM or MARS simulator. Make sure to specify in a text file which simulator you used.

Note: if creating additional functions are helpful then please do so.

Note: save the source file as PrintArray.s.

QUESTION 2: Assembler Programming

Every advanced programming language needs an IO library. In this question we will create an IO library with functions: getc, putc, gets, puts, readInt. Then we will modify the program in question 1 to use your IO library. You will not actually create a library for this question.

IO LIBRARY

Implement the following assembler subroutines following the C function signatures provided here. Keep in mind that we will use MIPS convention when implementing our subroutines: registers v0, v1 are used to return values and a0, a1, a2, and a3 are used to pass parameters. If we need to pass or return extra information then we must use the run-time stack. All local variables must be instantiated on the run-time stack. All malloc or new commands must be instantiated on the heap. The run-time stack must operate as a stack and the heap must operate like a heap.

- char getc()
  Reads a single character from the MIPS keyboard buffer and returns the ASCII value.
- void gets(char *array)
  This subroutine must use the getc() subroutine. The subroutine gets() reads multiple characters from the MIPS keyboard until the user presses the enter key. Then the reading stops and the subroutine returns. Every character read in is stored at the buffer pointed to by the *array reference that was passed as an argument to the subroutine. The enter key is included as part of the input. The subroutine will artificially add the null character as the last character of the input. This would make the input a string, in the C Language sense.
- void putc(char c)
  This subroutine takes a single character passed to it and prints it to the screen. The subroutine then returns.
- void puts(char *array)
  This subroutine must use the putc subroutine. The subroutine puts() assumes that the reference "array" points to a buffer that contains only ASCII characters terminating with the null character. Using putc(), puts() outputs each character from the buffer to the screen until the null character is reached. The null character is not outputted. Then subroutine returns once it encounters the null character.
- int readInt()
  This subroutines uses gets() to read in a string it assumes is a series of ASCII character digits. The input string is stored in a local variable (using the run-time stack). Then the subroutine converts the string into an integer and returns that integer. If the string does not contain and integer number the subroutine returns 0. The definition of an integer number is: a series of ASCII digits ending with either a carriage return and null character, or just the null character. No other ASCII character can be part of this string. Once an offending character is found the subroutine simply return the integer number zero.

THE MAIN PROGRAM

We will redo the question #1's program with the following modifications: the static array will not be initialized with values at compile time, instead the program will prompt the user to input 10 numbers and will then read in 10 integer numbers placing them into the array. The remainder of the question #1 program will execute the same way as in question #1.

Note: the source file for this question is called IO.s and will contain all the IO subroutines described above and the modified main program.

QUESTION 3: Performance Issues

Assume we have a modem (a type of network device that uses the phone to connect to a network) that receives a stream (ie. data transmitted in units of some byte size, sent one at a time, having an initial byte and terminating n-bytes later) of ASCII characters from the Internet. Assume further we have a CPU that is capable of performing polling, interrupt and DMA processing of peripherals. In this question we would like to do a comparative analysis of these capabilities with the modem as our case study. We will assume that all other circuitry, OS activities, and resource demands are negligible.

The modem receives data from the Internet by units of "packet". A packet is a stream of n-bytes. So, if your packet size is 100 then the modem receives 100 bytes, one at a time, over the phone line. The modem can be placed into stream mode or buffer mode. If the modem is in buffer mode then an internal modem buffer receives the transmitted packet. There is a delay between packets of 250cs. After 250cs a new packet arrives and overwrites the contents of the buffer. When the modem is in stream mode a single 8-bit register is used to store a single byte from the packet. The delay in transmission between the bytes in a packet are 5cs. After 5cs a new byte arrives and overwrites the contents of the register.

The CPU can be used in three ways to communicate with it's peripherals. All CPUs share polling as their default method to communicate with peripherals. If a CPU has special interrupt circuitry then is can be placed into interrupt mode to communicate with a peripheral though interrupts. If a CPU has special DMA circuitry then it can be placed into DMA mode to communicate with a peripheral through the DMA. We have a single core CPU that runs at 1 GHz.

Polling works in the following way: first, it assumes that the modem is in stream mode. This means that the programmer must make sure that the assembler program they are writing to do polling will monitor the modem's single input byte register quicker than the delay between the reception of each byte. The modem delays were already discussed in the modem paragraph above. A polling program can be optimally written to process this input stream at 4cs per byte.

Interrupts work in the following way: first, it assumes that the modem is placed in buffer mode. It takes 20cs to setup the interrupt. In modem buffer mode the modem operates on its own receiving bytes from the network connection and saving them into it's internal modem buffer. Once the modem has received the entire packet or once the buffer is full the modem then activates the interrupt already setup by the programmer. The interrupt code copies the data in the modem buffer into a RAM (or cache) buffer (we will assume RAM). 100 bytes of data can be copied at 1cs per byte + 20cs of miscellaneous overhead. The interrupt needs to be re-setup for the next batch of data.

DMA works in the following way: first, it assumes that the modem is placed in buffer mode. It takes 50cs to setup the DMA. This setup need to be performed for each batch of data received. Since the modem is in buffer mode it operates on it's own until the entire packet has been received. Then it activates the DMA which copies the contents of the modem's buffer into RAM without the need of the CPU. At the end of the transfer the DMA interrupts the computer to tell it that is is done. This interrupt requires 10cs to process. Once this is done the DMA needs to be reset for the next batch of data.

Our case study: Assume our computer is receiving 5 packets. Each packet is 100 bytes long.

The comparative results:

- Calculate how long it will take, in cycles per second, to process these 5 packets in each of the CPU modes described (ie. Polling, interrupts, and DMA).
- Calculate the load on the CPU, in percentages, to process these 5 packets in each of the CPU modes described (ie. Polling, interrupts, and DMA).
- Discuss, in this case – not the general case, the merits of each of these CPU modes (ie. Polling, interrupts, and DMA).

Note: hand in a txt, doc, or pdf file named A3Q3 with the answer to this question.

WHAT TO HAND IN

Everything should be handed in electronically on My Courses.

Submit the source file and any supporting files or documentation that you think the TA would need.

HOW IT WILL BE GRADED

This assignment is worth 30 points. Everything is graded proportionally.

- Question 1 – 10 points
- Question 2 – 10 points
- Question 3 – 10 points