COMP 273

Assignment #3

Due: April 14, 2015 at 23:30 on myCourses

Assembly Programming

This is the last assignment in the course. It has two programming questions and one problem question. It is due on the last week of classes. The problem question is on material we will see near the end of the course.

QUESTION 1: The C Calling Convention and the OS syscall

First:
Download both MARS and SPIM. Try them out and select the assembler environment you enjoy most to use. Most students prefer MARS. The course website has links to these two assemblers.

Second:
Below is a C program that displays the values in a static array on the screen. It prints the array from front to back and then back to front. Write this program in assembler following the C calling conventions: passing parameters using the stack, local variables on the stack, saving your registers on the stack. Use the OS syscall to read and write. Your program must duplicate the below program exactly. Note that the array is global, defined in the .data area.

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

void main(void)
{
      int low, high;
      scanf("%d %d", &low, &high); // read on one line or two is okay
      printArray(low, high);
      printArray(high, low);
}

void printArray(int startIndex, int endIndex)
{
      int a; // local variable
      if (a<0 || a >= array_length) return;
      for(a = startIndex; a <= endIndex; a++)
            printf("%%d\n",array[a]);
}
```

Note: any additional functions you create to help must follow proper assembler calling techniques and local memory techniques.

QUESTION 2: Drivers Vs. Functions – MIPS Keyboard and Text Screen Drivers

Computers have many types of drivers: printer, keyboard, mouse, video, network, modem, hard disk, sound card, etc. Many of these drivers are created using assembler.

SPIM only simulates two hardware devices: the keyboard and the text-screen. These are both referred to as the console I/O devices.

In MIPS programming we have the following conventions: a MIPS subroutine can be implemented in two ways, as a driver or as a function. If a MIPS subroutine is implemented as a driver then it does not use the run-time stack but uses only the register conventions for subroutine access. In other words a0-a4 for parameter passing and v0-v1 for returning values. It only uses the run-time stack for saving the s registers. If a MIPS subroutine is implemented as a function then it follows the C function calling conventions: all parameters, local variables, and saving registers are placed on the run-time stack, and returned values are placed in vo-v1. You <u>must</u> use the above conventions when writing the answer to this question.

Write a MIPS program that asks the user to enter their first and then last name. The program then displays: "You entered: <last name>, <first name>". Where <first name> and <last name> are the words they inputted. The program then stops. Your program must adhere to the following instructions:

1. Create a <u>driver</u> called GETCHAR, char getchar(void) in MIPS, that interfaces with the keyboard's status and data registers. This driver when called returns the character in the keyboard buffer to register $V0. Do not use the OS syscall command. Access the peripheral directly. A slide in class was given to help you. Implement this program as a driver using the driver polling flowchart we discussed in class.

2. Create a <u>driver</u> called PUTCHAR, void putchar(char c) in MIPS, that interfaces with the screen's (console) status and data registers. This driver when called assumes that register $A0 contains the character that needs to be output. Only one character is output. Do not use the OS libraries. Access the peripheral directly. A slide in class was given to help you. Implement this program as a driver using the driver polling flowchart we discussed in class.

3. To test this, you will build the following <u>functions</u> in MIPS:
   a. int gets(char *buffer, int limit)
      int puts(char *buffer)

      The function GETS reads (and PUTS writes) ASCII from the keyboard into a memory space pointed to by BUFFER (from BUFFER to the screen for PUTS). It stops reading when the user either presses the enter key or when LIMIT characters is reached (stops printing when NULL is found for PUTS). The string is terminated with a '\0' (if there is space). The function also returns the number of characters it actually read (outputted for PUTS) into $v0. The function GETS must use GETCHAR and the function PUTS must use PUTCHAR.
   b. The function gets() and puts() are used to read and write the user's name, and any other string based input or output, from the main program.

<u>QUESTION 3: Performance Issues</u>

Assume we have a hard drive that can operate in both polling or interrupt mode. Assume further that the disk drive can access data in either block or byte mode. Block mode uses the disk drive's internal buffer to store 10K bytes of data. In block mode the hard drive can run on its own after receiving the address on disk and the number of bytes to read from the CPU. All of these bytes are loaded into the buffer. If the number of bytes to read is greater than the size of the buffer or if all the requested bytes have been read, an interrupt is sent to the CPU once the buffer is full or once the operation is completed. The CPU then needs to download the buffer to RAM, clear the buffer, and instructing the drive to continue reading (if needed). Byte mode simply downloads a single byte of data from the hard disk given an address on disk to the first byte in the disk drive buffer. No interrupt is sent. It is up to the CPU to know when to extract that byte from the buffer using polling. The disk drive has a status register that is set to integer 1 when the drive is busy, 0 when it is not busy and no data is in the buffer, 2 when it is not busy but a single byte is in the buffer, and 3 if it is not busy with a full buffer.

Assume that polling takes 200cs, while interrupts takes 500cs. Assume we want to copy a one meg file from disk to RAM. Assume further that all other assembler instructions take only 1cs to execute. Assume you have a 500MHz processor.

Answer the following questions:

- How many cs will it take for polling to load the file into RAM using byte mode?
- How many cs will it take for polling to load the file into RAM using block mode?
- How many cs will it take for interrupts to load the file into RAM in byte mode?
- How many cs will it take for interrupts to load the file into RAM in block mode?
- Compare the impact in percentage of processing time for the above 4 calculation
  - Where does polling loose all it time compared to where interrupts loose all its time?
- Assuming that DMA has an overhead of 1000cs and can transfer the entire file:
  - Calculate the number of cs it will take to load the file into RAM.
  - Calculate the impact on percentage of processing time.
  - Where does the DMA loose all its time?
  - How does DMA compare with polling and interrupts?
- Why do we still have these three methods of accessing peripherals?
- State a situation where polling, interrupts and DMA can be applied appropriately.

WHAT TO HAND IN

Everything should be handed in electronically to myCourses.

Submit source files for questions 1 and 2. A text file for stating which compiler you used. A PDF file for questions 3.

HOW IT WILL BE GRADED

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

- Question 1 – 7 points
- Question 2 – 7 points
- Question 3 - 6