MIPS Peripherals and Performance
Due: December 3, 2019 at 23:55 on MyCourses

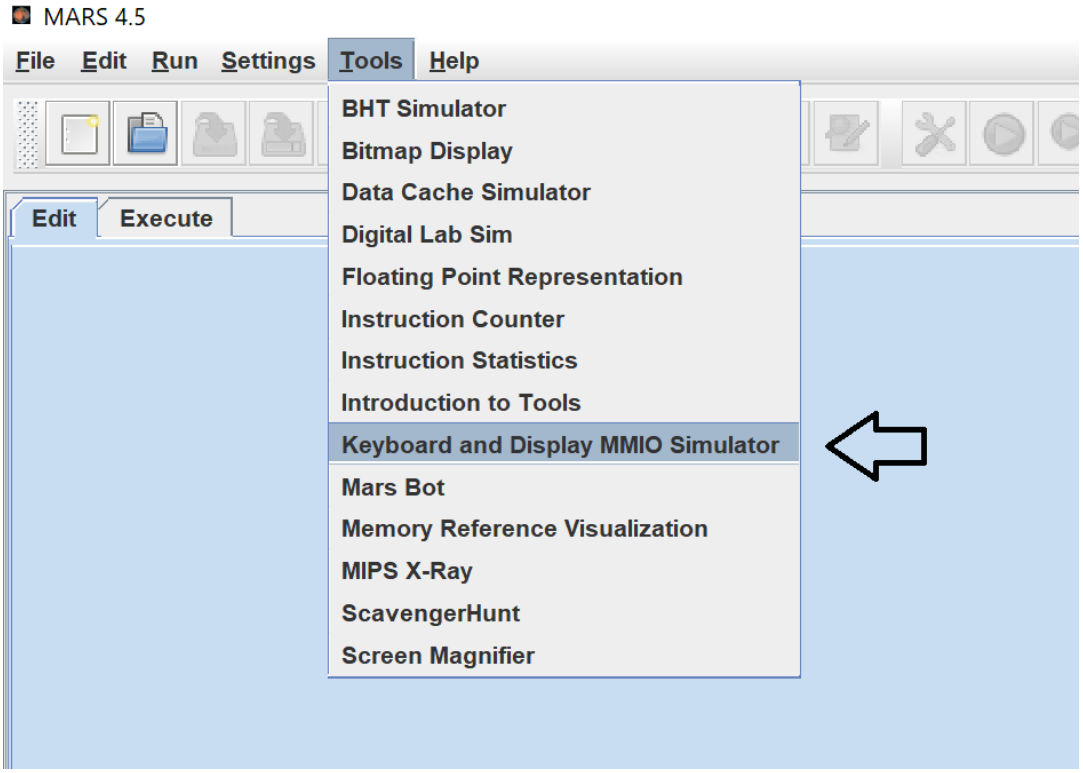
Tutorials I and J will be helpful for this assignment. The code to read and write from the MIPS keyboard and screen has been given in class. See the lecture slides.

QUESTION ONE: MIPS Keyboard and Text Screen Drivers

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

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

To access the MARS keyboard and text screen you will need to input your characters and see the results of your program from the following location:



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.

For this assignment, write the code designated as "driver" using the register-passing convention for subroutines. Write the code designated as "function" using the C-passing convention using the run-time stack.

Write a MIPS program that prompts the user to enter their first name and then prompts them to enter their 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 driver called GETCHAR, similar to `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 driver called PUTCHAR, similar to `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 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.
3. To test this, you will build the following functions 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 read (outputted for PUTS) into $v0.

      The function GETS must use GETCHAR and the function PUTS must use PUTCHAR.

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. You CANNOT uses syscall in this assignment at all for string and character operations.

QUESTION TWO: Performance

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 start address on disk and the number of bytes to read from the CPU. All these bytes are loaded into the buffer. If the number of bytes to read is greater than the size of the buffer (only the bytes that fit into the buffer are loaded) or if all the requested bytes have been read (less than the buffer size), 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 instruct 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 for one byte takes 200 ticks, while interrupts take 500 ticks. Assume we want to copy a one meg (10^6 bytes) file from disk to RAM. Assume further that all other assembler instructions, for simplicity, take only 1 tick to execute. Assume you have a 500 MHz (500 million ticks per second) processor.

Answer the following questions:

- How many ticks will it take for polling to load the file into RAM using byte mode?
- How many ticks will it take for polling to load the file into RAM using block mode?
- How many ticks will it take for interrupts to load the file into RAM in byte mode?
- How many ticks 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 lose all its time compared to where interrupts lose all its time?
- Assuming DMA has an overhead of 1000-ticks and can transfer the entire file directly to RAM without needing additional help from the CPU:
    - Calculate the number of ticks it will take to load the file into RAM.
    - Calculate the impact on percentage of processing time.
    - Where does the DMA lose 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

Hand in the following to MyCourses:

- The MARS source code for question one, called 6Q1.asm
- A PDF of the question two solutions, called 6Q2.pdf

HOW IT WILL BE GRADED

Question 1 — Total points = 20

Uses MARS	: 1 point
GETCHAR	: 2 point
PUTCHAR	: 2 points
GETS	: 2 points
PUTS	: 2 point
main()	: 2 point

Each question is graded proportionally compared with the official solution.

Question 2

2.1	: 1 point
2.2	: 1 point
2.3	: 1 point
2.4	: 1 point
2.5	: 1 point
2.6	: 3 points
2.7	: 1 point

-5% per day late.
After 2 late days, the assignment is not accepted.