EE 443
Lab #3: Assembly Language Programming III; MIPS Floating-Point Instructions. MIPS Keyboard Controller

Assigned 2/14/2023
Due 2/21/2022 (2:00 pm)

Introduction.

The goal of this laboratory is to understand the process of working with memory-mapped peripherals and floating-point coprocessor.

ASSIGNMENT

Objective

Write a program that calculates the exact area of a circle (expressed in square inches) for a **radius** that a user enters **as an integer number** (in mm). Your program should also ask the user to enter the value for **π** as a **double-precision floating point number**. You can use a system call for this. If the user enters a value of 0 for the radius, your program should exit, otherwise, it should ask the user to enter another set of inputs. Because the program is very simple, pay special attention to user interface – display prompts before accepting inputs and messages after getting the data and before displaying results. Show the units for all data entered and calculated.

To practice peripheral programming and communication, for this assignment you are supposed to **write your own "myread_int"** code. Your program should let a user keep entering digits (0-9) until ENTER is pressed (*hint: under Windows, ENTER sends two characters – CR (0x10) followed by LF (0x0a); CR will be ignored in MARS. Under Linux, ENTER sends LF (0x0a) only*). You can assume that the user will "behave," i.e. you don't have to worry about what will happen if the user enters something other than digits. Of course, feel free to provide additional functionality.

Communicating with a keyboard controller

In MIPS memory mapped I/O address space, the two registers associated with the keyboard controller are called **receiver control** and **receiver data**. They are accessed by a MIPS program using the physical addresses 0xFFFF0000 and 0xFFFF0004, respectively.

The **ready bit** of the receiver control register is *the least significant bit* and it is set to **one** when a key is pressed on the keyboard. At the same time, the ASCII code of that key is latched into the *least significant 8 bits* of the **receiver data** register.

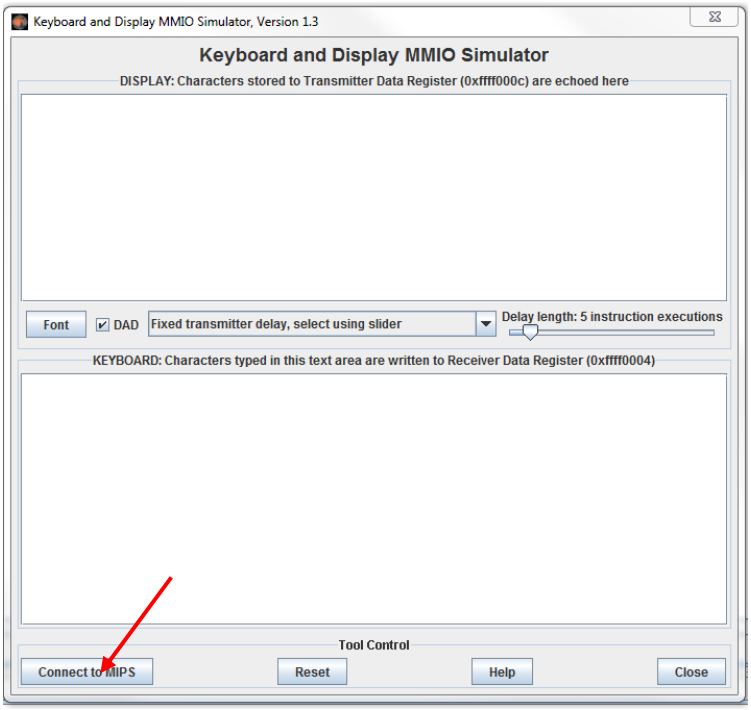
The keyboard controller automatically clears the ready bit as soon as it detects that the receiver data register is read.

Floating-point register file usage convention (you are not required to use function calls for this assignment)

`$f0 - $f3`	Function-returned values
`$f4 - $f11`	Temporary values
`$f12 - $f15`	Arguments passed into a function
`$f16 - $f19`	More temporary values
`$f20 - $f31`	Saved values

NOTES:

- For your other I/O needs, feel free to use system calls (for example, to enter a double-precision floating point value).
- Use double-precision floating-point wherever appropriate.
- In order to be able to use keyboard controller simulator in MARS, you have to open the tool called "Keyboard and Display Simulator" and connect it to your MIPS program (see figure). The tool "Floating Point Representation" can help you understand the single-precision floating point format, but unfortunately, it doesn't work with double precision formats.



Requirements

1. Write an assembly language program "LAB3.asm" to carry out the objective of Assignment.
2. The program should be commented
3. **Upload your programs to Blackboard**, before the due date and time.
4. Please demonstrate the program to the TA (or instructor) in the lab, and be prepared to answer questions about the programming environment and your programs

General suggestion:

Start small, and then expand your program. Test procedures separately.

Report and Programs Requirements

1. Keep it simple.
2. Turn in the commented code for program "LAB3.asm".
3. Follow the template for lab reports (available on Blackboard). Write a short but complete narrative account of the program.
4. Please put your name and the date you wrote the program in a comment line at the beginning of the program.

5. The programs will be graded for correctness, clarity, and effectiveness.

Answer the following questions and include them in your report. Make sure you understand what is happening on stack and in registers, because this is a common question on all exams.

Enter **14** for the radius and **3.14159265358979323846264338327 95** for π (you can copy and paste this value from the Windows calculator or from this assignment). Step through the program (and/or set breakpoints) to help you answer the following questions:

- What is the content of the integer register containing the radius?
- What is the content of the floating point register containing the radius (in Hex)?
  - Prove that this register contains the right value. Show the contents of all the fields and their meaning. Try not to use online FP calculators and converters. If you do ☺, make sure you understand how the IEEE754 format works, because the converters will not be available during exams.
- What is the content of the floating point register containing the pi (in Hex)?
  - What is the actual value stored in this floating point register?
  - What is the relative error compared to the value given in this assignment?

Convert Integer to double

`cvt.d.w fd, fs`

0x11	0x14	0	fs	fd	0x21
6	5	5	5	5	6

Convert the single precision floating-point number or integer in register `fs` to a double (single) precision number and put it in register `fd`.

Move from coprocessor 1

`mfc1 rt, fs`

0x11	0	rt	fs	0
6	5	5	5	11

Coprocessors have their own register sets. These instructions move values between these registers and the CPU's registers.

Move register `rd` in a coprocessor (register `fs` in the FPU) to CPU register `rt`. The floating-point unit is coprocessor 1.

Remaining MIPS-32	Name	Format	Pseudo MIPS	Name	Format
exclusive or (*rs* ⊕ *rt*)	xor	R	absolute value	abs	rd,rs
exclusive or immediate	xori	I	negate *(signed or unsigned)*	negs	rd,rs
shift right arithmetic	sra	R	rotate left	rol	rd,rs,rt
shift left logical variable	sllv	R	rotate right	ror	rd,rs,rt
shift right logical variable	srlv	R	multiply and don't check oflw *(signed or uns.)*	muls	rd,rs,rt
shift right arithmetic variable	srav	R	multiply and check oflw *(signed or uns.)*	mulos	rd,rs,rt
move to Hi	mthi	R	divide and check overflow	div	rd,rs,rt
move to Lo	mtlo	R	divide and don't check overflow	divu	rd,rs,rt
load halfword	lh	I	remainder *(signed or unsigned)*	rems	rd,rs,rt
load byte	lb	I	load immediate	li	rd,imm
load word left *(unaligned)*	lwl	I	load address	la	rd,addr
load word right *(unaligned)*	lwr	I	load double	ld	rd,addr
store word left *(unaligned)*	swl	I	store double	sd	rd,addr
store word right *(unaligned)*	swr	I	unaligned load word	ulw	rd,addr
load linked (*atomic update*)	ll	I	unaligned store word	usw	rd,addr
store cond. (*atomic update*)	sc	I	unaligned load halfword *(signed or uns.)*	ulhs	rd,addr
move if zero	movz	R	unaligned store halfword	ush	rd,addr
move if not zero	movn	R	branch	b	Label
multiply and add (S or *uns.*)	madds	R	branch on equal zero	beqz	rs,L
multiply and subtract (S or *uns.*)	msubs	I	branch on compare *(signed or unsigned)*	bxs	rs,rt,L
branch on ≥ zero and link	bgezal	I	(x = lt, le, gt, ge)		
branch on < zero and link	bltzal	I	set equal	seq	rd,rs,rt
jump and link register	jalr	R	set not equal	sne	rd,rs,rt
branch compare to zero	bxz	I	set on compare *(signed or unsigned)*	sxs	rd,rs,rt
branch compare to zero likely	bxzl	I	(x = lt, le, gt, ge)		
(x = lt, le, gt, ge)			load to floating point (*s* or *d*)	l.*f*	rd,addr
branch compare reg likely	bxl	I	store from floating point (*s* or *d*)	s.*f*	rd,addr
trap if compare reg	tx	R			
trap if compare immediate	txi	I			
(x = eq, neq, lt, le, gt, ge)					
return from exception	rfe	R			
system call	syscall	I			
break *(cause exception)*	break	I			
move from FP to integer	mfc1	R			
move to FP from integer	mtc1	R			
FP move (*s* or *d*)	mov.*f*	R			
FP move if zero (*s* or *d*)	movz.*f*	R			
FP move if not zero (*s* or *d*)	movn.*f*	R			
FP square root (*s* or *d*)	sqrt.*f*	R			
FP absolute value (*s* or *d*)	abs.*f*	R			
FP negate (*s* or *d*)	neg.*f*	R			
FP convert (*w*, *s*, or *d*)	cvt.*f*.*f*	R			
FP compare un (*s* or *d*)	c.xn.*f*	R			

FIGURE 3.25 Remaining MIPS-32 and Pseudo MIPS instruction sets. *f* means single (s) or double (d) precision floating-point instructions, and *s* means signed and unsigned (u) versions. MIPS-32 also has FP instructions for multiply and add/sub (madd.*f*/msub.*f*), ceiling (ceil.*f*), truncate (trunc.*f*), round (round.*f*), and reciprocal (recip.*f*). The underscore represents the letter to include to represent that datatype.