Discussion 13: VM (cont), I/O

1. Consider a call to the following MIPS code (no delay slots) with the given initial page table. Assume that pages are 4KiB and that all page faults (but not protection faults) can be serviced by the OS without evicting pages. $sp is initially 0x6004, $ra is initially 0x1040, and $a0 is initially 0x1.

MIPS V.A.	Instructions
0x2004	Foo: addiu $sp, $sp, -4
0x2008	sw $ra, 0($sp)
0x200C	beq $a0, $zero, Skip
0x2010	addiu $a0, $a0, -1
0x2014	jal Foo
0x2018	Skip: lw $ra, 0($sp)
0x201C	addiu $sp, $sp, 4
0x2020	jr $ra

Initial Page Table

Valid	Dirty	A.R.	P.P.N.
0	0	None	4
1	0	Read, Exec	5
0	0	Read, Exec	1
0	0	None	1
0	0	Read, Write	12
1	0	Read, Write	3
1	0	Read, Write	2
…	…	…	…

a. Where will page faults occur in the execution of this function?

On the first instruction executed. Since 0x2004 corresponds to virtual page 2, which is not valid, a page fault will be triggered as a result of the instruction fetch. No other page faults will occur.

b. Assuming that we don't have a TLB, (or that all the TLB was flushed), what will be in the page table after this function is completely executed?

Final Page Table

Valid	Dirty	A.R.	P.P.N.
0	0	None	4
1	0	Read, Exec	5
1	0	Read, Exec	##
0	0	None	1
0	0	Read, Write	12
1	1	Read, Write	3
1	1	Read, Write	2
…	…	…	…

c. Suppose $a0 were initially 0xC00 instead of 0x1, what other exceptions can occur?

Deep into the recursion $sp would end up being 0x4FFC when executing the instruction at 0x2008, which would cause a page fault for virtual page 4. Later, $sp would be 0x3FFC, which would cause a protection fault for virtual page 3.

2. Fill this table of polling and interrupts.

Operation	Definition	Pro/Good for	Con
Polling	Forces the hardware to wait on ready bit (alternatively, if timing of device is known – the ready bit can be polled at the frequency of the device). It basically means manually checking the ready bit regularly.	PRO: -easy to write -poll handler does not have excessively high overhead -deterministic -doesn't require additional hardware Good for: -Mouse, keyboard	Infeasible on hardware with fast transfer rates that is actually rarely ready (e.g. Ethernet card receiver)
Interrupts	Hardware fires an "exception" when it becomes ready. CPU changes $PC to execute code in the interrupt handler when this occurs.	PRO: -Necessary for fast devices that are rarely ready. Good for: Fast devices - Hard drives, Network cards	-nondeterministic when interrupt occurs -interrupt handler has some overhead (saves all registers), meaning polling can actually be faster for slow, often ready devices such as mice

3. Memory Mapped I/O

Certain memory addresses correspond to registers in I/O devices and not normal memory.

0xFFFF0000 – Receiver Control:
Lowest two bits are interrupt enable bit and ready bit.

0xFFFF0004 – Receiver Data:
Received data stored at lowest byte.

0xFFFF0008 – Transmitter Control
Lowest two bits are interrupt enable bit and ready bit.

0xFFFF000C – Transmitter Data
Transmitted data stored at lowest byte.

Write MIPS code to read a byte from the receiver and immediately send it to the transmitter.

```
                lui $t0 0xffff
receive_wait: #poll on ready of receiver
                lw $t1 0($t0)
                andi $t1 $t1 1
                beq $t1 $zero receive_wait
                lb $t2 4$t0) #load data
transmit_wait: #poll on ready of transmitter
                lw $t1 8($t0)
                andi $t1 $t1 1
                beq $t1 $zero transmit_wait
                sb $t2 12($t0) #write to transmitter
```