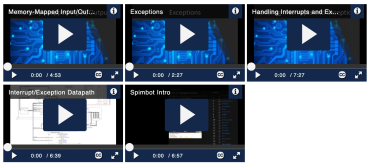
PRE13.1. 13 Text and Videos

Videos



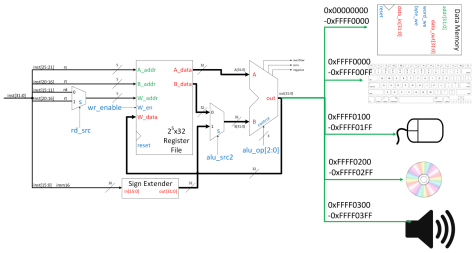
Links: Interrupts (Memory Mapped I/O), Exceptions, Handling Interrupts and Exceptions, Interrupt/Exception Datapath,SPIMbot Intro

Video credits: Geoffrey Herman, Text credits: Geoffrey Herman

The Big Picture

Memory-Mapped Input/Output (MMIO)

Input/Output devices such as keyboards, monitors, speakers, and external storage devices are not part of the processor but need a way to send/receive information to/from the processor. Memory-mapped I/O is a system where we pretend that these external devices are part of the data memory. Each device is assigned a range of memory addresses depending on how much data that device needs to send/receive. For example, in the image below the data memory only has addresses `0x00000000` to `0xFFFF0000`. Any load or store instruction to addresses not in that range would send or receive data to/from the external devices.



Examples

Read what key was pressed on a keyboard

```
lw   $t0, 0xFFFF0008($0)
```

Send a sound to the speaker

```
sw   $t0, 0xFFFF0304($0)
```

In the two examples above, we are taking advantage of a feature of the assembler. The assembler will re-assemble the 32-bit immediate into two 16-bit immediates to meet the requirements of I-type instructions.

Examples continued

Keyboard code is assembled into

```
lui  $at, 0xFFFF
lw   $t0, 0x0008($at)
```

Speaker code is assembled into

```
lui  $at, 0xFFFF
sw   $t0, 0x0304($at)
```

Interrupts

Interrupts are welcome events that need to briefly stop the processor from running your program so that other things can happen. For example, a key being pressed on a keyboard will create an interrupt. If your program requests data from an external storage device, your program will receive an interrupt some time later (external storage devices are very slow) when that data is available. Interrupts generally come from devices that are external from the central processing unit (i.e., things other than the register file, ALU, data memory, and instruction memory).

Interrupts are the way programs get the information they need want from external devices. A useful analogy is a classroom. We instructors value your input and questions. If something we said or presented is unclear, we want you to interrupt us so that we can get that valuable feedback/information. When a external device wants to interrupt, it raises an interrupt flag - `flag = 1` (kind of like raising your hand to ask a question). The program can set permission flags to indicate whether a device is allowed to interrupt (kind of like an instructor saying "any questions?" `permission = 1` or "please hold your questions for a moment" `permission = 0`).

Interrupt permission?	Interrupt flag	What happens next?
No	No	Program keeps running
No	Yes	Program keeps running
Yes	No	Program keeps running
Yes	Yes	Program is interrupted

If an interrupt happens, a special function called the interrupt handler will be automatically called by the hardware. The interrupt handler will run some code on the main processor that will collect the data being sent by the device and then send an acknowledgement flag to the device (like an instructor acknowledging a student who is raising their hand). When the device is acknowledged, it will lower its interrupt flag - `flag = 0`.

Exceptions

An exception is an error.

Exceptions are caused by a program trying to do something it is not allowed to. For example, an overflow flag of 1 during an `add` instruction would raise an arithmetic overflow exception. Similarly, an opcode that does not match a known instruction (the `except` output of the MIPS instruction decoder you implemented in Lab 5) would raise a reserved instruction exception.

Exceptions in MIPS are encoded with a 5-bit unsigned binary number. Some example exceptions (non-exhaustive).

Number	Binary	Name	Cause of Exception
0	00000	Int	No exception occurred. An hardware interrupt possible.
4	00100	AdEL	Address error exception (load or instruction fetch). Usually an unaligned address error such as trying to perform a `lw` from an address that is not divisible by 4.
5	00101	AdES	Address error exception (store). Usually an unaligned address error such as trying to perform a `sw` from an address that is not divisible by 4.
6	00110	IBE	Bus error on instruction fetch. Usually caused by trying to read an instruction from a restricted or non-existent address such as 0x00000000.
7	00111	DBE	Bus error on data load or store. Usually caused by trying to read or write to an address in the data memory that is restricted or non-existent address such as 0x00000000.
9	01001	Bp	Breakpoint exception
10	01010	RI	Reserved instruction exception. An opcode in the machine code was encountered that is not implemented by the datapath. Most likely happens when code was compiled for a newer version of the ISA and is then attempted to be run on an older processor.
12	01100	Ov	Arithmetic overflow exception. Addition or subtraction resulted in overflow for instructions that allow the overflow exception. See the footnotes on the reference guide to figure out which ones can result in an overflow exception
13	01101	Tr	It's a trap!

Cause and Status Registers

The MIPS ISA provides two special registers for identifying when interrupts will happen and when exceptions occur. **These registers are not part of the register file** and are part of what is called co-processor 0.

The Status Register

The status register keeps track of the permissions for whether an interrupt is allowed to interrupt the program. There are two types of permissions: global permissions (Interrupt Enable) and device permissions (Interrupt Mask). If Interrupt Enable is 0, no interrupts are allowed no matter what. If Interrupt Enable is 1, the interrupt mask indicates which devices may interrupt. The interrupt mask consists of 8 bits, corresponding to 8 different sources of interrupts. A bit of 0 means that a corresponding source may not interrupt. A bit of 1 means that a corresponding source my interrupt.

Interrupt Mask	Interrupt Enable
0 0 1 0 1 1 0 0	1

For example, the status register above would indicate that interrupts are enabled globally and the only three sources are allowed to interrupt.

The Cause Register

The cause register indicates the source of an interrupt or an exception (error).

The exception code is a 5-bit unsigned binary code. If the code is 0, then there is not currently an exception. If the exception code is not 0, then an error has occured. Only one error is allowed to be indicated at a time.

The cause register has 8 pending interrupt bits corresponding to the 8 bit interrupt mask bits in the status register. These bits are flags that indicate whether a source is trying to interrupt: 1 an interrupt is pending, 0 an interrupt is not pending.

Pending Interrupts	Exception Code
0 1 0 0 1 0 0 0	0 0 0 0 0

The example cause register above indicates that an exception has not occured (Exception Code == 0) and that two sources are trying to interrupt the program.

Determining whether an interrupt or an exception will happen next

```
if (exception code != 0)
   handle an exception
else if (interrupt enable == 1)
   if (interrupt mask & pending interrupts != 0)
      handle an interrupt
```

Using the two examples above, we can see that Exception Code == 0, so an exception did not occur. The Interrupt Enable == 1, so an interrupt was possible. Bitwise ANDing the Interupt Mask with the Pending Interrupts yields: 00101100 & 01001000 = 00001000. 00001000 != 00000000, so an interrupt occurs next.

An interrupt and an exception cannot occur at the same time. Because an exception is an error, it must be handled first.

Mark as read

- [ ] (a) I've read this!

Select all possible options that apply.

Save & Grade *Unlimited attempts* Save only

PRE.13

Assessment overview

Total points: 40/40

Score: 100%

Question PRE13.1

Value: 1

Total points: — /1

Auto-graded question

Previous question Next question

Personal Notes

No attached notes

Attach a file

Add text note