

Where are we?

We have explored a single clock cycle processor design
[datapath and control]

We have introduced pipelining
and established its performance
benefits and operational
challenges

We have explored pipelined
processor implementation
[datapath and control]

Exceptions [Interrupts!]



Exceptions and Interrupts

- “Unexpected” events requiring change in flow of control
 - Different ISAs use the terms differently
- ***Exception (traps)***
 - Arises within the CPU
 - e.g., undefined opcode, overflow, divide by zero, address error, syscall ...
- ***Interrupt***
 - From an external I/O controller (memory, I/O device)
- Handling exceptions always reduces the performance

More on Exceptions

- **Arithmetic overflow** occurs when the absolute value of a result is too large for the format.
- **Arithmetic underflow** occurs when the absolute value of a result is too small for the format (e.g., floating point)
- The **divide by 0** trap is self-explanatory.
- **Illegal instruction** may happen if program counter or the memory was corrupted
- **Bus error** for inaccessible resources
- **Segmentation fault** is an attempt to access memory in an illegal way.

More on Interrupts


- Handling I/O with busy waiting or polling has challenging drawbacks
- Interrupts are a hardware-based solution for handling IO
 - IO devices indicate the need of service by raising a ***flag*** (changing a bit value)
 - Added hardware checks the flags → CPU continues processing instructions
 - On interrupt detection, it is up to software to actually perform the I/O transaction

Exception/Interrupt Handlers

- A special OS program that is executed in the event of exception or interrupt
- Also known as interrupt service routines (ISRs)
- Typical actions
 - Fatal problem → the running program may be terminated (e.g., segmentation or bus errors) and the offending instruction should be reported for debugging
 - Recoverable → continue the execution of the current program after handling the interrupt (e.g., IO operation)

Handling Exceptions/Interrupts

- Hardware saves current PC in a special register
- **CPU jumps to an exception handler**, which
 - Saves the current processor state
 - Take appropriate internal action for an exception, or Deal with the external source of the interrupt
 - Restores the previous processor state
 - Resumes user program execution, if possible
- Exception handler resides in kernel memory, inside the OS
 - Kernel has extra privileges and protection over user code
 - However this is not simulated in SPIM. . .



| |
|----------------------|
| add \$t0, \$t1, \$t0 |
| sub \$t2, \$t1, \$t0 |
| sub \$t0 \$s0, \$a0 |
| sll \$t0, \$t0, 2 |
| . |
| . |
| . |
| add \$t0, \$k1, \$k0 |
| sub \$t0, \$k0, \$k1 |
| eret |

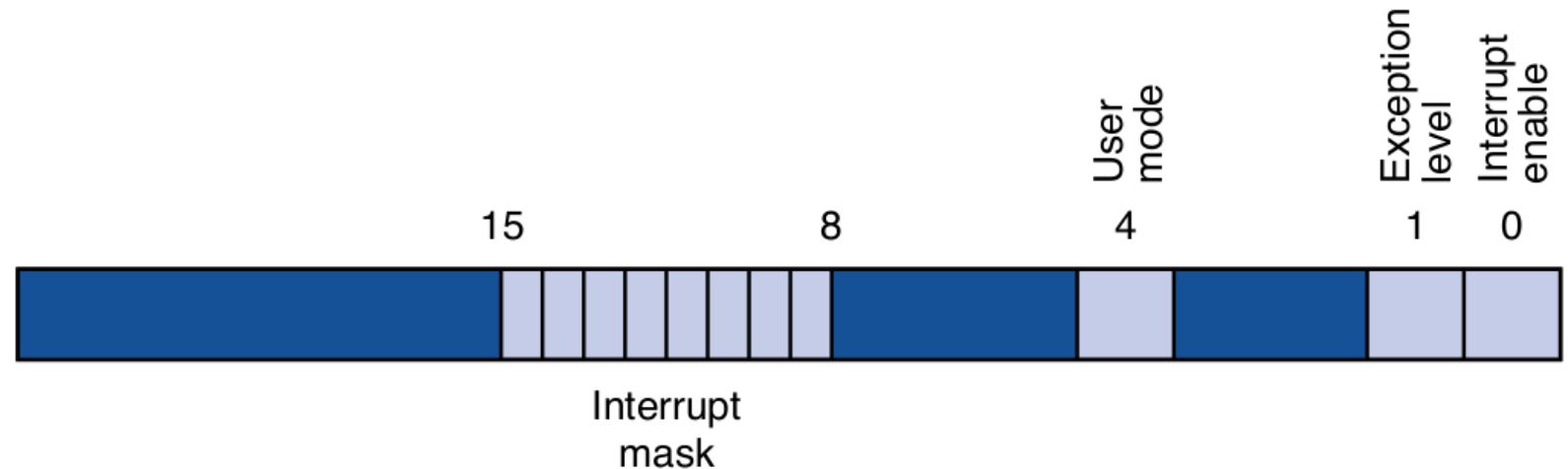
Handling Exceptions **in MIPS**

- MIPS uses ***coprocessor 0*** for exception/interrupt handling
- The following registers are simulated in MARS

| Register Name | Register # | Description |
|---------------|------------|--|
| VAddr | \$8 | offending memory reference |
| Status | \$12 | controls which interrupts are enabled |
| Cause | \$13 | exception type, and pending interrupts |
| EPC | \$14 | PC where exception/interrupt occurred |

Status Register (**MIPS \$12@C0**)

- **Interrupt enable bit:** global interrupt enable (or disable)
- **Exception level bit:** automatically set during an exception; prevents handler itself being interrupted
- **Interrupt mask** which of the 8 interrupts are allowed
 - Eight interrupt bits: 6 hardware, 2 software levels



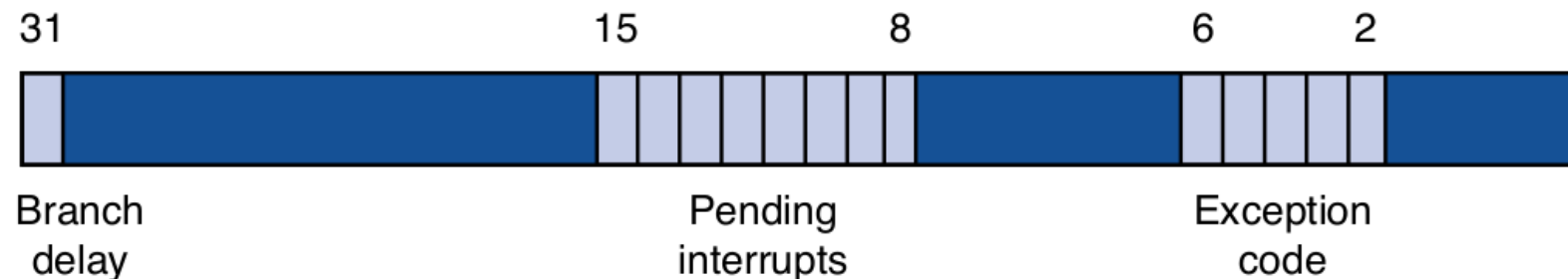
Cause Register (**MIPS \$13@C0**)

- Exception code is a 5-bit integer field

| | |
|----|--------------------------------|
| 4 | ADDRESS_EXCEPTION_LOAD |
| 5 | ADDRESS_EXCEPTION_STORE |
| 8 | SYSCALL_EXCEPTION |
| 9 | BREAKPOINT_EXCEPTION |
| 10 | RESERVED_INSTRUCTION_EXCEPTION |

| | |
|----|-------------------------------|
| 12 | ARITHMETIC_OVERFLOW_EXCEPTION |
| 13 | TRAP_EXCEPTION |
| 15 | DIVIDE_BY_ZERO_EXCEPTION |
| 16 | FLOATING_POINT_OVERFLOW |
| 17 | FLOATING_POINT_UNDERFLOW |

- Pending interrupts is bitfield of the 8 interrupts



MIPS Coprocessor 0 Instructions

- Special Register instructions (*mfc0* and *mtc0*)

move from coprocessor 0

mfc0 \$reg, \$regC0 # \$reg << \$regC0

move to coprocessor 0

mtc0 \$reg, \$regC0 # \$reg >> \$regC0

#Example: Allow all hardware interrupts

mfc0 \$t0, \$12

ori \$t0, \$t0, 0xff01

mtc0 \$t0, \$12

MIPS Interrupt/Exception Handler

- Handler **always at address 0x80000180** in kernel memory
 - Use the **.ktext 0x80000180** and **.kdata** directives
- **Must save and later restore all used registers**
 - Including \$at
 - Except \$k0 and \$k1 may be used freely
 - Should not use stack – may point to invalid memory
 - Can temporarily spill registers to .kdata
- Return control to the user program with **eret**
 - Jumps to EPC, and resets Exception level in Status
 - ISR must increment EPC by 4 to proceed to the next instruction

Summary

- Identified possible causes of interrupts and exceptions
- Analysed ISR actions
- Explore MIPS HW and ISA support for exception/interrupt handlers
- Identified guidelines for writing ISR for MIPS

READING

- Appendix A.7 and A.8