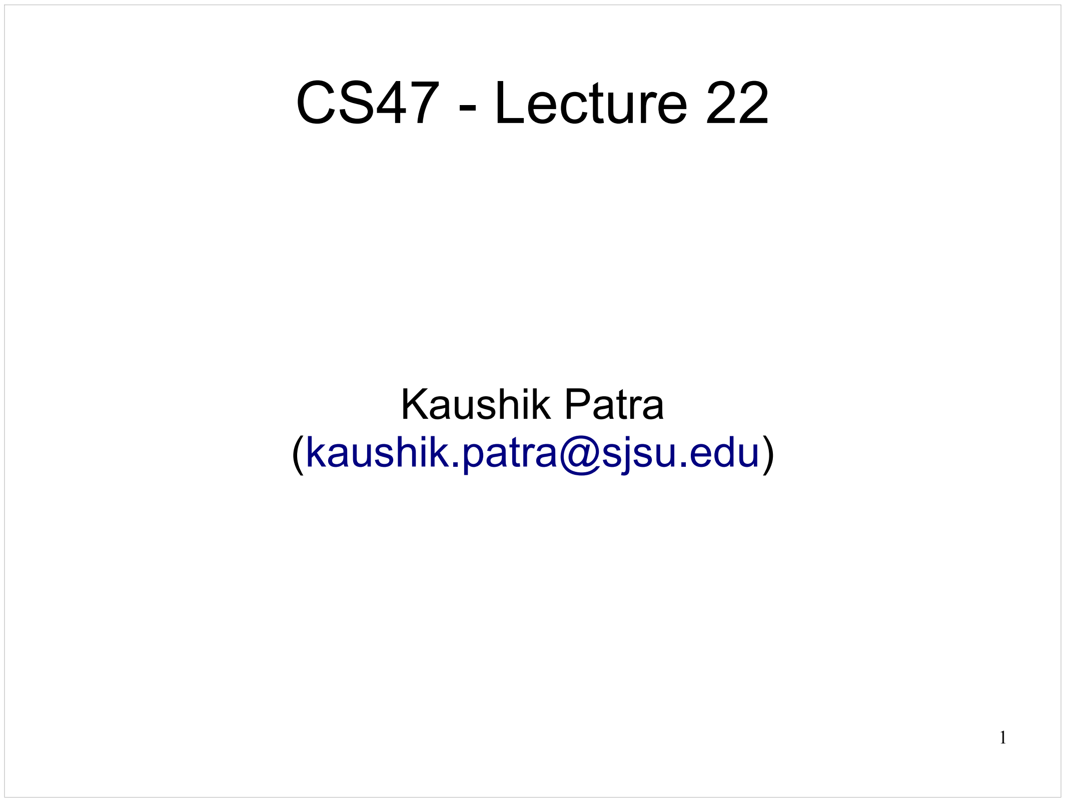


CS47 - Lecture 22

Kaushik Patra
(kaushik.patra@sjsu.edu)

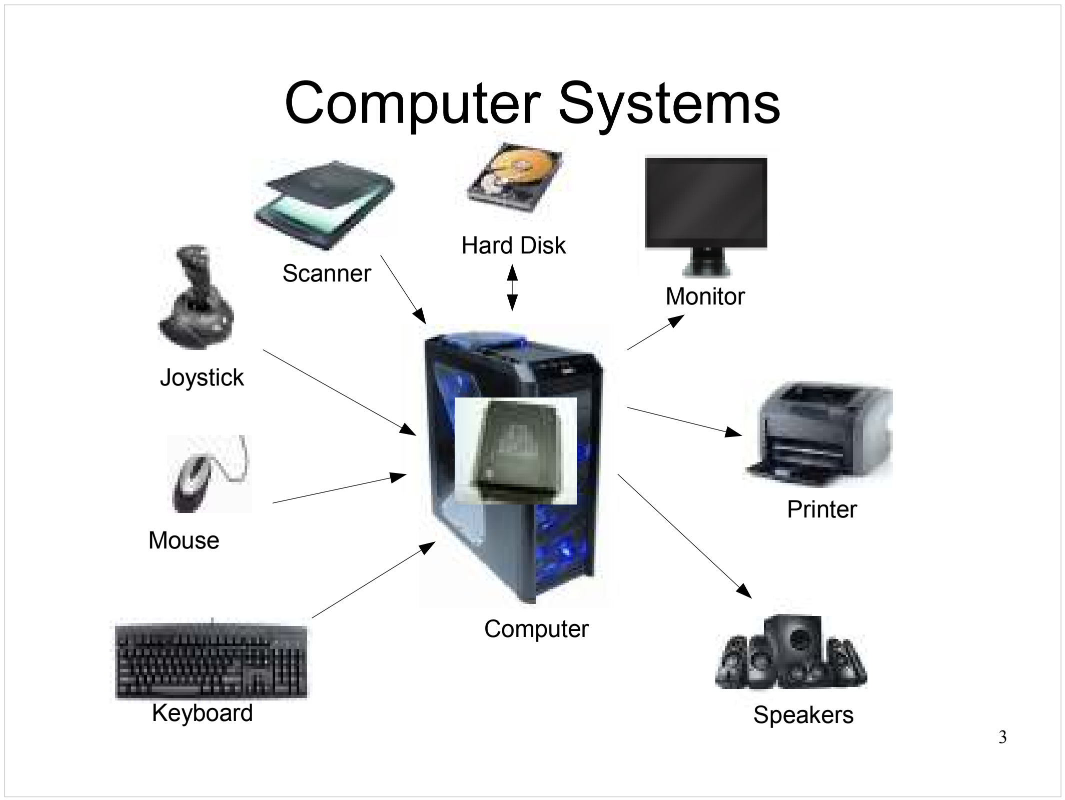
1

- Exception & Interrupt
- MIPS Support for Interrupt

Reference Books:

1) *Appendix A.7 of 'Computer Organization & Design by Hennessy, Patterson*

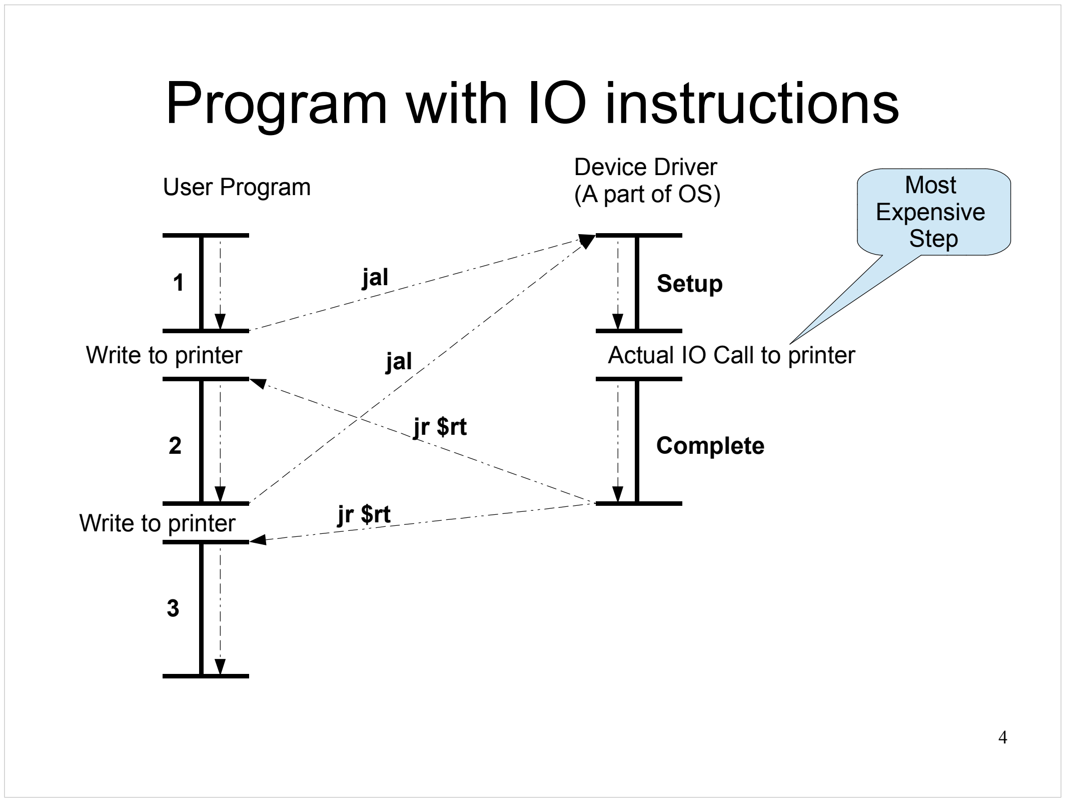
Interrupt and Exception ...



Computer Systems

- A computer itself is not very useful unless external devices for data input and output is attached. Commonly used input devices are keyboard, mouse, joystick, scanner, etc. These input devices captures information from human to process. On the other hand commonly used output devices are monitor, printer, speaker systems, etc. There output devices shows human perceptible result of processed input information. There are another types of devices which acts as both input and output device. For example hard disk drive (or any other read/write storage media) which can be used to store information for future use. Computer can intake information from these devices and write back processed information for future reference. All these devices are called peripheral devices.

- The main component of a computer box is the processor which execute instruction faster than any input / output / IO devices. Usually processor is faster than peripheral devices by order or thousand or even more.

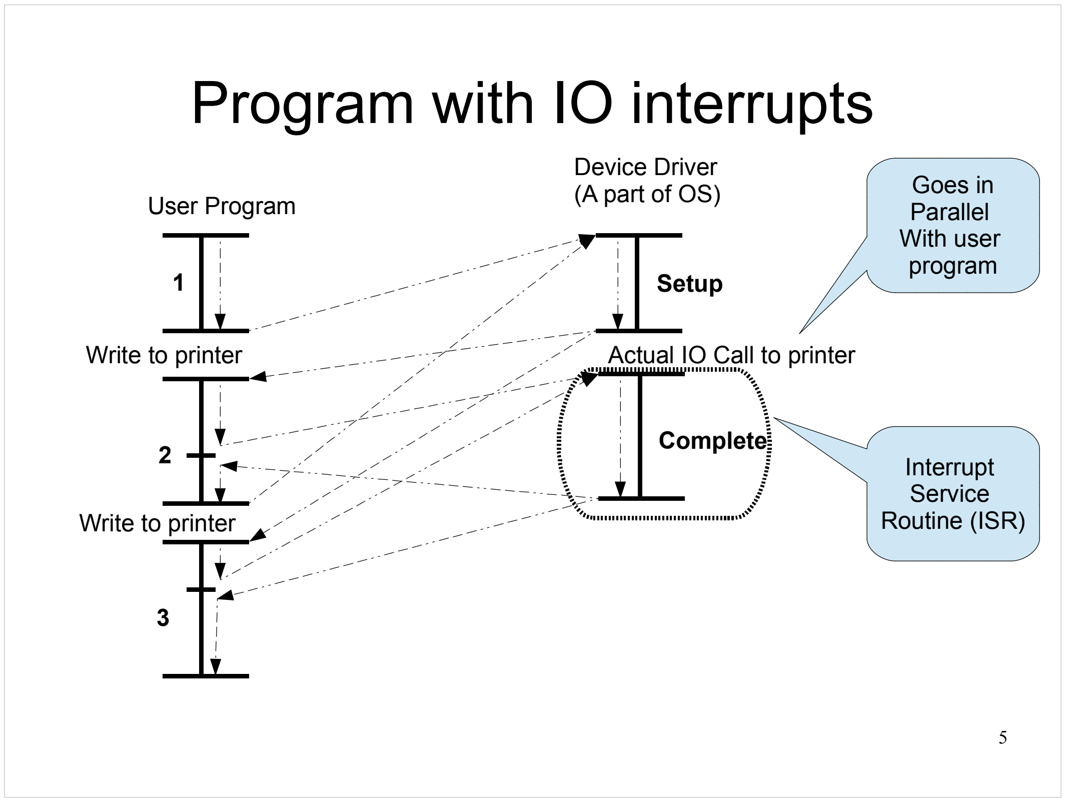


Program with IO instructions

- To interface to external devices and transfer data to them, a specialized program is needed (sometime called IO program). This program sets up external devices ready to receive / transmit data to 'n fro processor. Once the setup is done, data transfer command is send to devices and start of data transfer. This data transfer / action completion step is the most time consuming because it is usually involves elctro-mechanical process. This time consumption is usually in order of thousand times or more compare to actual program running on processor. Once the device is done data transferring, IO program has to perform some completion / wrapping up steps for the data to be accessible to processor.

- All the different IO programs for different IO devices are integrated to operating system in form of device drivers. Each device driver acts as separate procedures in overall system execution environment. Once user program request for a data transfer (using IO APIs) control switches to specific device driver procedure. Device driver procedure performs device setup and calls the device operation. Without any sophisticated mechanism, processor waits till the entire IO operation is done. Once IO operation is done , control is returned to device driver. After device driver finalize the operation result, control is back to user program.

- A user program will be stalled significantly, if it is IO intensive. However, once the request is made to IO device, theoretically the rest of the user program can continue (in many cases) in parallel while IO device is busy in transferring data.



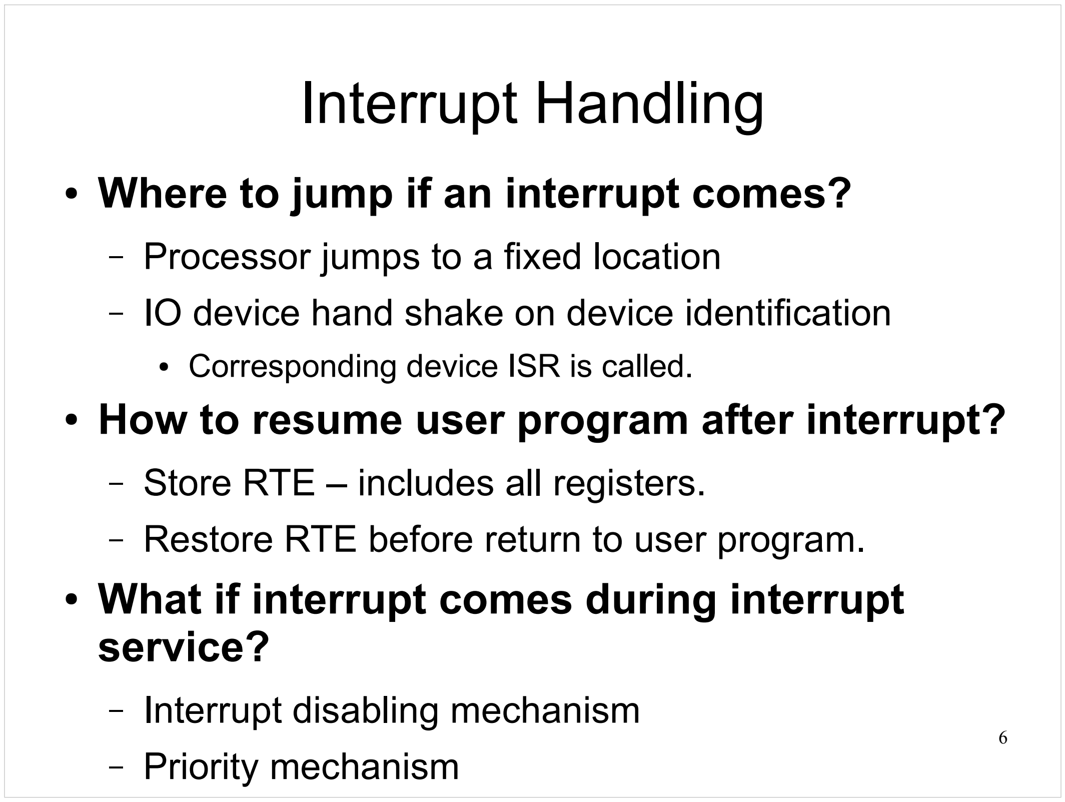
Program with IO interrupts

Device Driver
(A part of OS)
User Program
Goes in Parallel With user program
1
Setup
Write to printer
Actual IO Call to printer
2
Complete
Interrupt Service Routine (ISR)
Write to printer
3

5

- Interrupt mechanism has been developed to let processor continuing program execution in parallel while IO device is busy transferring data. In this process, the control is back to user program immediately after the IO command is issued (after the setup is done by device driver) to IO device. When IO device is done, control is immediately (after completion of the current user program instruction) back to the 'completion' part of the data driver. Once the 'completion' part is done control is back to user program at the point where the program was interrupted.

- This 'completion' part of the device driver is called 'interrupt service routine'.



Interrupt Handling

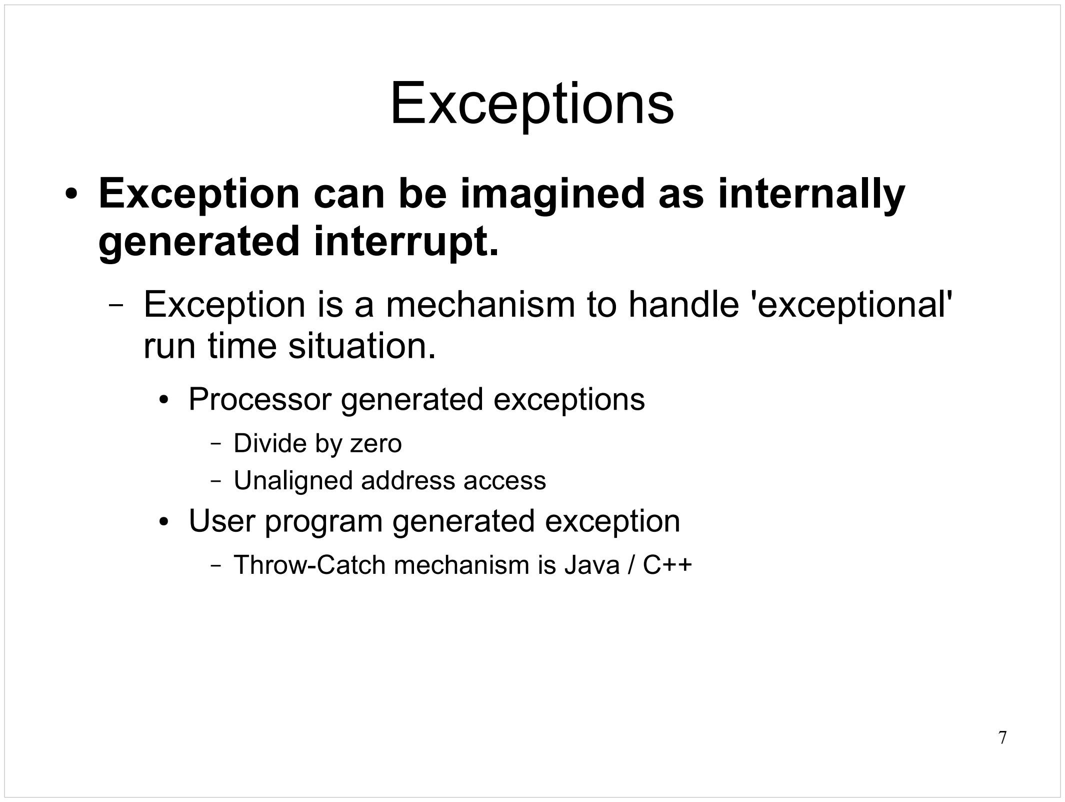
- **Where to jump if an interrupt comes?**
  - Processor jumps to a fixed location
  - IO device hand shake on device identification
    - Corresponding device ISR is called.
- **How to resume user program after interrupt?**
  - Store RTE – includes all registers.
  - Restore RTE before return to user program.
- **What if interrupt comes during interrupt service?**
  - Interrupt disabling mechanism
  - Priority mechanism

6

- Upon receiving interrupt from external devices, the current instruction (in user program) is completed and then the control is transferred to a fixed address to serve the interrupt. Devices create specific interrupt ID so that system knows which device has interrupted the execution and calls corresponding IRS (Interrupt Service Routine).

- In-order to make the interrupt service completely transparent to user program, IRS should not leave any artifact that would affect normal execution of the user program. Preserving the run-time environment is very similar to RTE saving of caller in procedure call mechanism. However, in this case, RTE includes all the register values (not only the saved register values). Moreover, stack can not be used to store the RTE in this case. Device driver usually uses OS kernel memory part to store user programs' RTE.

- To handle multiple interrupt – two mechanism has been developed. One is interrupt disabling mechanism and other is interrupt priority. The concept is that there can be multiple type and classes of interrupts – some of them are to be addressed immediately and others can be deferred till current ISR is completed. Interrupt disabling mechanism provides a way to ISR to disable part of all types of interrupt while in process. Interrupt type with highest priority can disable all other interrupts till its own process is done. Once done, it re-opens the interrupts. However, if dissabled, incoming interrupts are not lost but saved and deferred till it is re-enabled.

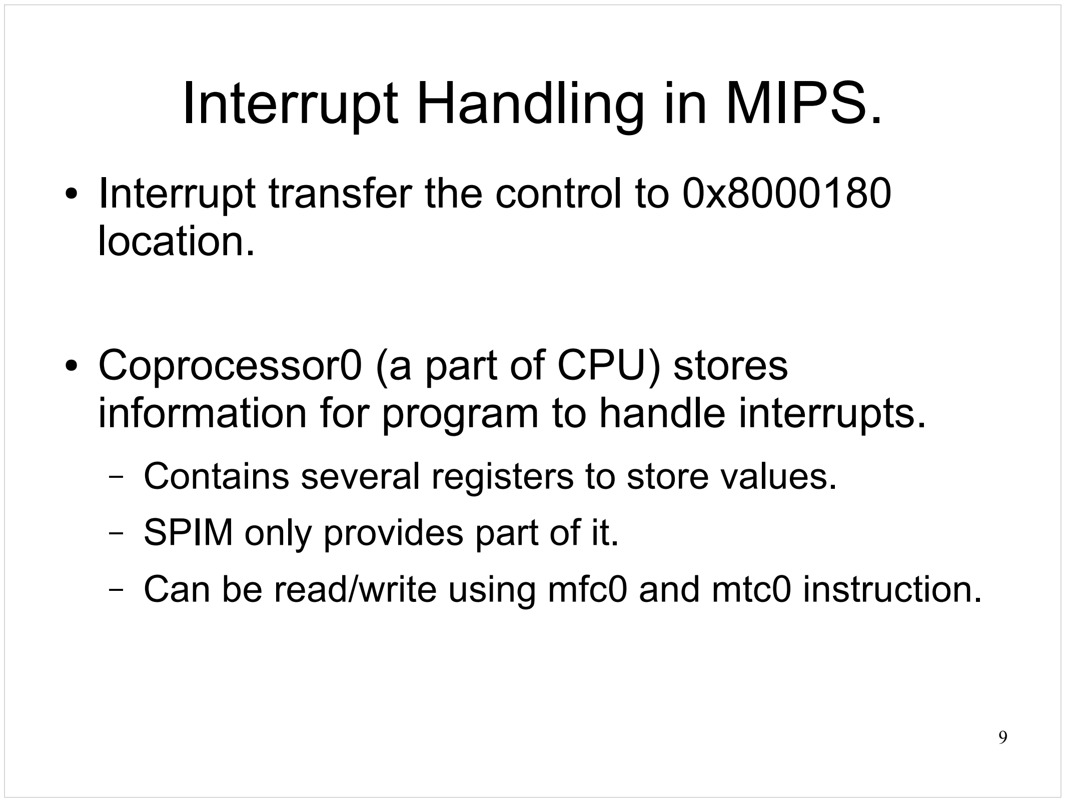


Exceptions

- **Exception can be imagined as internally generated interrupt.**
  - Exception is a mechanism to handle 'exceptional' run time situation.
    - Processor generated exceptions
      - Divide by zero
      - Unaligned address access
    - User program generated exception
      - Throw-Catch mechanism is Java / C++

7

MIPS support for interrupt & exception ...

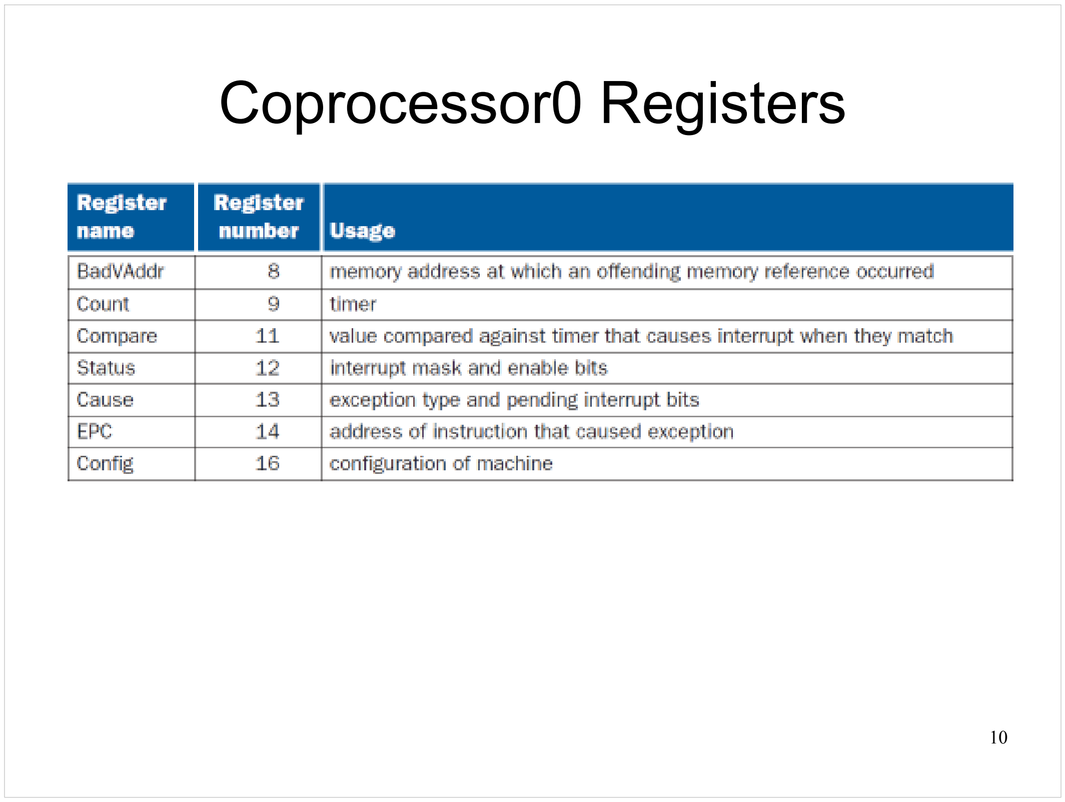


Interrupt Handling in MIPS.

- Interrupt transfer the control to 0x8000180 location.
- Coprocessor0 (a part of CPU) stores information for program to handle interrupts.
  - Contains several registers to store values.
  - SPIM only provides part of it.
  - Can be read/write using mfc0 and mtc0 instruction.

9

- Upon receiving interrupt, MIPS control goes to a fixed location of 0x80000180. This is the entry point for any interrupt service. The current instruction is completed and then the control is transferred to this location. A part of CPU, known as Coprocessor0, is implemented to store interrupt information, so that software can access them and take appropriate actions. Special instruction mfc0 and mtc0 is provided to access these special registers. SPIM provides only a subset of the complete register sets in Coprocessor0 – fundamentally because all the registers are not relevant in simulation environment.



Coprocessor0 Registers

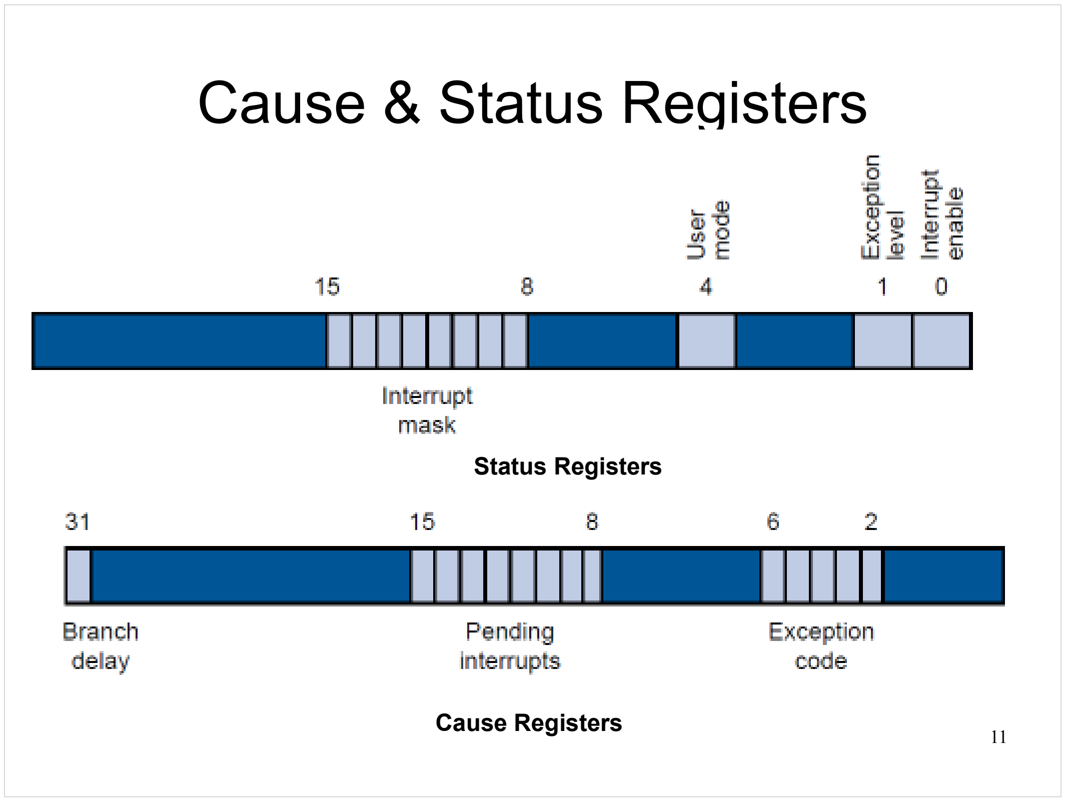
Register name	Register number	Usage
BadVAddr	8	memory address at which an offending memory reference occurred
Count	9	timer
Compare	11	value compared against timer that causes interrupt when they match
Status	12	interrupt mask and enable bits
Cause	13	exception type and pending interrupt bits
EPC	14	address of instruction that caused exception
Config	16	configuration of machine

10

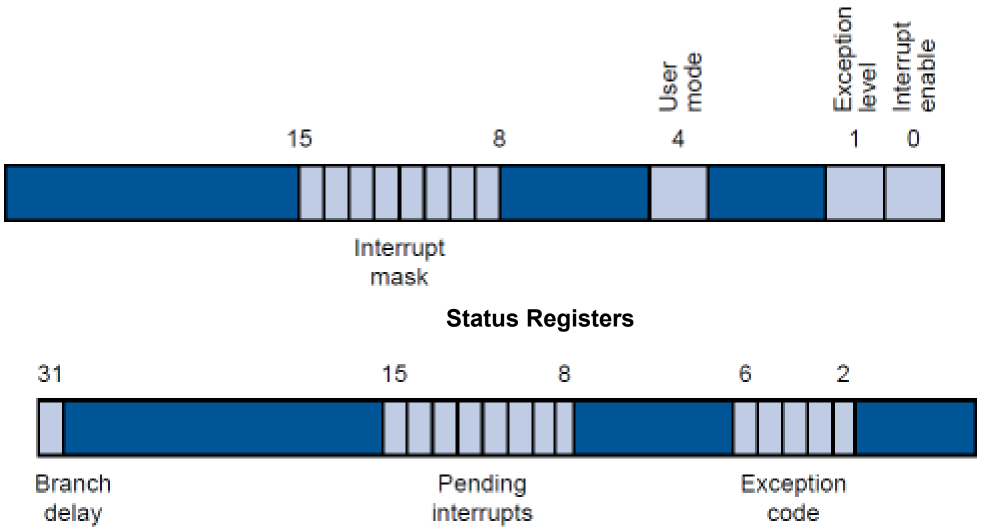
- EPC register contains the address of the user program instruction to be executed after interrupt is serviced. After the service of the interrupt, control jumps back to the address contained in EPC. However, for exceptions, EPC contains the instruction that caused the exception. Therefore, if the user program has to be resumed, it should be resumed at an address EPC+4.

- The cause register contains exception types and pending list of interrupt. Exception writes the cause as numerically encoded value (as described in slide 12).

- The status register controls the behavior of interrupt. Using this, program can enable/disable interrupt as needed.



Cause & Status Registers



Status Registers

Cause Registers

11

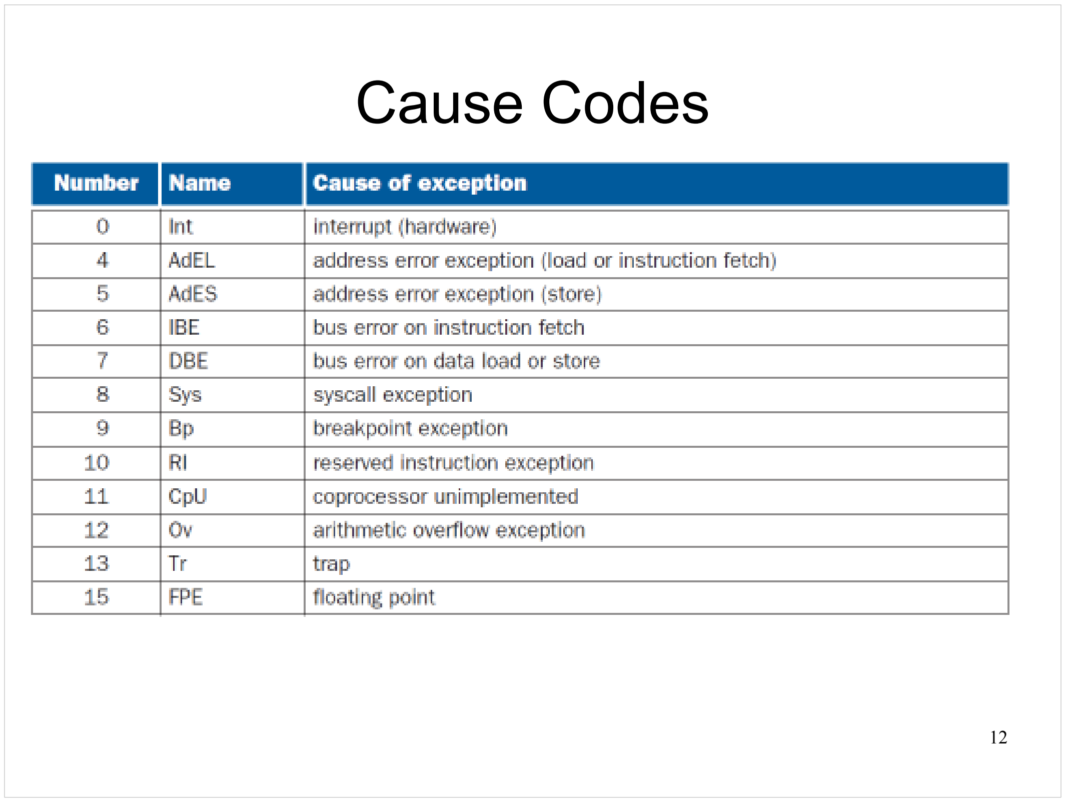
- The bits 8-15 (total 8) implements the interrupt mask for each type of six hardware interrupts and two software interrupts. A mask bit is set to 1 to allow interrupting processor at that level. This means, while doing servicing interrupt, ISR can turn off partial set of interrupts at lower level. If a interrupt mask is set to 0, it sets corresponding pending bit to 1 in the cause register (bit 8-15). If any interrupt is pending, it will call interruption of processor execution as long as corresponding interrupt mask bit is 1.

- The user mode bit is set to 0 if the processor is running in kernel mode and 1 if it is running in user mode (some exceptions are disabled if kernel mode – for example accessing kernel code / data address). On SPIM it is set to 1, since SPIM processor does not implement kernel mode.

- The exception level bit is normally set to 0, but is set to 1 after an exception occurs. When this bit is 1, EPC is not updated if another exception occurs. This bit prevents an exception handler from being disturbed by another interrupt and exception.

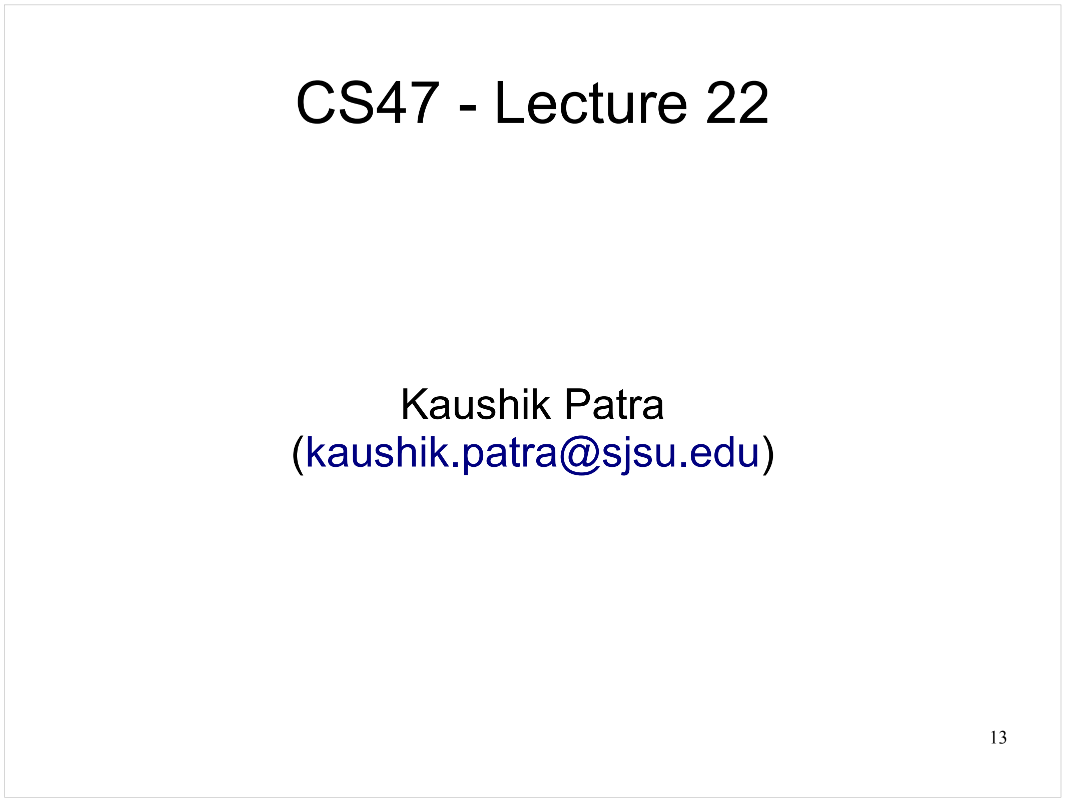
- If interrupt enable bit is set to 0, all interrupts are disabled globally.

- Cause codes are listed in next slides.



Cause Codes

Number	Name	Cause of exception
0	Int	interrupt (hardware)
4	AdEL	address error exception (load or instruction fetch)
5	AdES	address error exception (store)
6	IBE	bus error on instruction fetch
7	DBE	bus error on data load or store
8	Sys	syscall exception
9	Bp	breakpoint exception
10	RI	reserved instruction exception
11	CpU	coprocessor unimplemented
12	Ov	arithmetic overflow exception
13	Tr	trap
15	FPE	floating point



CS47 - Lecture 22

Kaushik Patra
(kaushik.patra@sjsu.edu)

13

- Exception & Interrupt
- MIPS Support for Interrupt

Reference Books:

1) *Appendix A.7 of 'Computer Organization & Design by Hennessy, Patterson*