Input/Output: Polling and Interrupts

Outline

- I/O Background
- Polling
- Interrupts

Anatomy: 5 components of any Computer

Computer

Processor (active)

Control ("brain")

Datapath ("brawn")

Memory (passive) (where programs, data live when running)

Devices

Input

Output

Keyboard, Mouse Disk (where programs, data live when not running)

Display, Printer
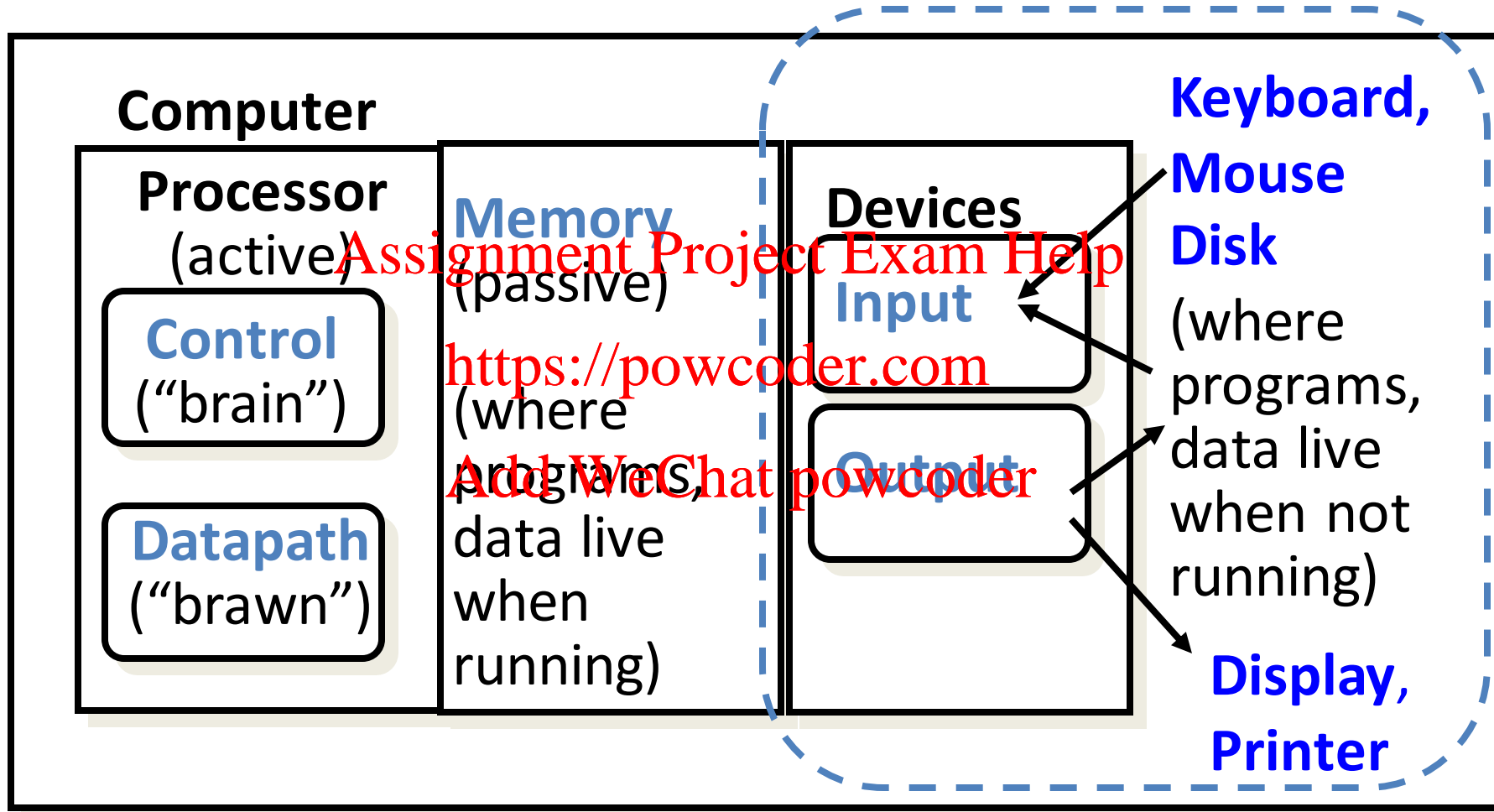
Motivation for Input/Output

- I/O is how humans interact with computers

- I/O gives computers long-term memory

- I/O lets computers do amazing things

I/O Device Examples and Speeds

- Kilobytes transferred per second from mouse to display… million to one range of rates!

Device	Behaviour	Partner	Data Rate KB/s
Keyboard	Input	Human	0.01
Mouse	Input	Human	0.02
Voice output	Output	Human	5.00
Floppy disk	Storage	Machine	50.00
Laser Printer	Output	Human	100.00
Magnetic Disk	Storage	Machine	10,000.00
Network-LAN	Input/Output	Machine	10,000.00
Graphics Display	Output	Human	30,000.00

What do we need to make I/O work?

- A way to connect many types of devices to the Proc-Mem

- A way to control these devices, respond to them, and transfer data

- A way to present them to user programs so they are useful

Files   *APIs*

Operating System

Proc

Mem

PCI Bus

SCSI Bus

cmd reg.

data reg.
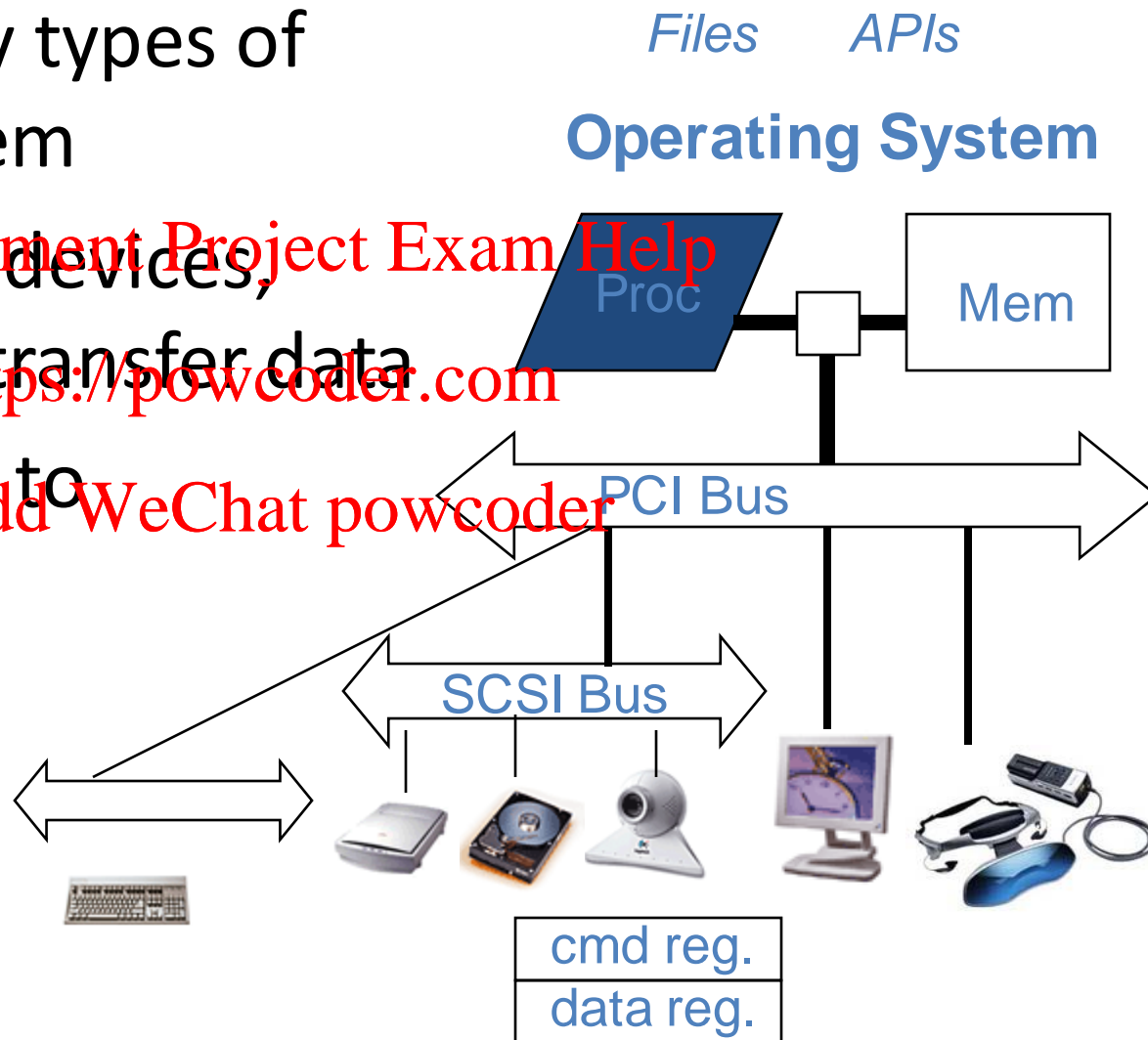
Instruction Set Architecture for I/O

- What must the processor do for I/O?
  - Input: reads a sequence of bytes
  - Output: writes a sequence of bytes
- Some processors have special I/O instructions
- Alternative model (used by MIPS):
  - Use loads for input, stores for output
  - Called "Memory Mapped Input/Output"
  - A portion of the address space is dedicated to communication paths to Input or Output devices (i.e., there is no memory there)
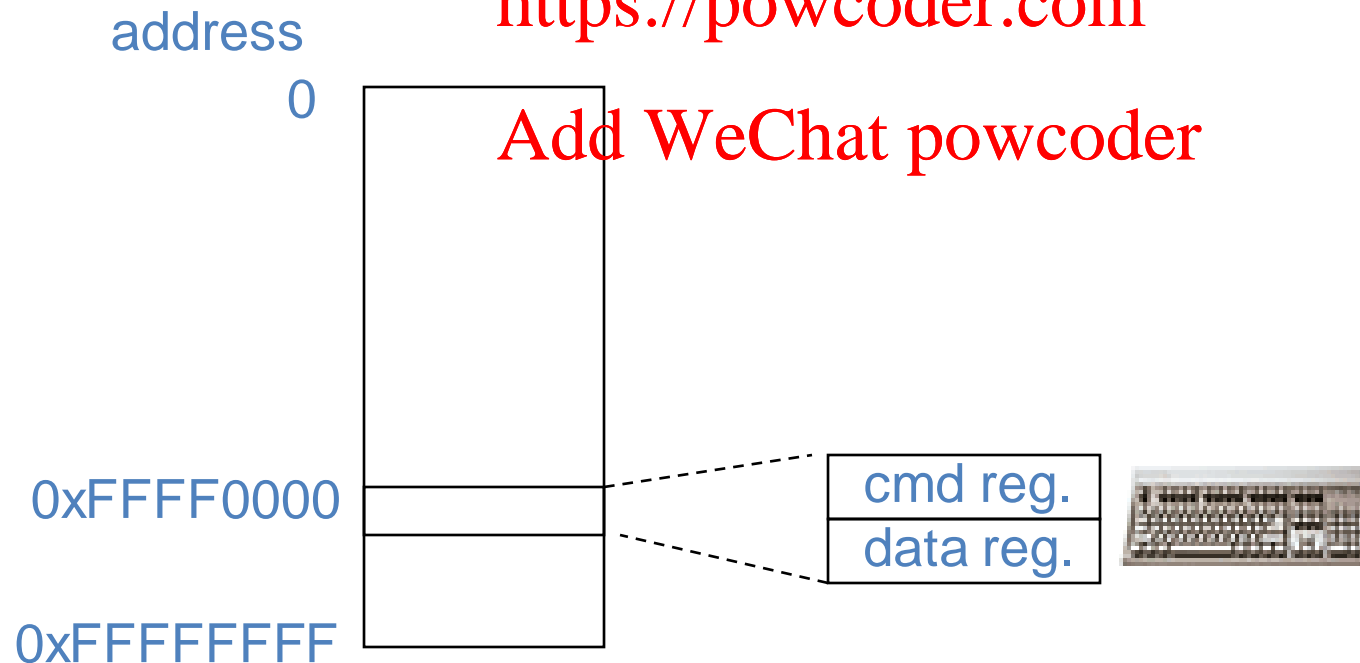
Memory Mapped I/O

- Certain addresses are not regular memory

- Instead, these addresses correspond to registers in I/O devices

address

0

0xFFFF0000

0xFFFFFFFF

cmd reg.

data reg.

Processor-I/O Speed Mismatch

- 1 GHz microprocessor can execute 1 billion load or store instructions per second, or 4 GB/s data rate
  - I/O devices data rates range from 0.01 KB/s to 30 MB/s
- Input: device may not be ready to send data as fast as the processor loads it
  - Also, might be waiting for human to act
- Output: device not be ready to accept data as fast as processor stores it
- What to do?

Processor Checks Status before Acting

- Path to device generally has 2 registers:
  - Control Register, says it's OK to read/write (I/O ready)
  - Data Register, contains data
- Processor reads from Control Register in loop, waiting for device to set *Ready* bit in Control register (0 becomes 1) to say its OK
- Processor then loads from (input) or writes to (output) data register
  - Load from or Store into Data Register resets Ready bit (1 becomes 0) of Control Register

I/O Simulation

- MARS (and SPIM) simulate one I/O device:
  - Memory-mapped terminal (keyboard + display)
  - Read from keyboard (receiver); 2 device registers
  - Writes to screen (transmitter); 2 device registers
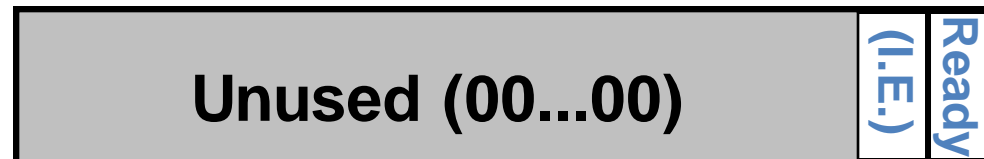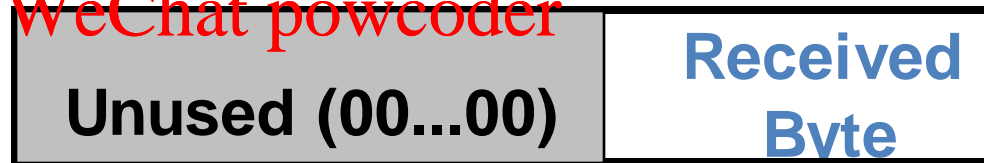
Receiver Control
`0xffff0000`

Unused (00...00)	(I.E.)	Ready

Receiver Data
`0xffff0004`

Unused (00...00)	Received Byte

Transmitter Control
`0xffff0008`

Unused (00...00)	(I.E.)	Ready

Transmitter Data
`0xffff000c`

Unused	Transmitted Byte

I/O Control and Data Registers

- Control register rightmost bit (0): Ready
  - Receiver: Ready==1 means character in Data Register has not yet been read; the 1 changes to 0 when data is read from Data Register
  - Transmitter: Ready==1 means transmitter is ready to accept a new character; 0 means the transmitter is still busy writing last char
- The I.E. (*interrupt enable*) bit we'll discuss later
- Data register rightmost byte has data
  - Receiver: last char from keyboard; other bytes in word are zero
  - Transmitter: when we write the rightmost byte, this will write the char to the display

I/O Example

- Input: Read from keyboard into $v0

```
                lui    $t0, 0xffff      # memory address 0xffff0000
Waitloop:       lw     $t1, 0($t0)      # receiver control
                andi   $t1,$t1,0x0001   # check ready bit with mask
                beq    $t1,$zero, Waitloop
                lw     $v0, 4($t0)      # data
```

- Output: Write to display from $a0

```
                lui    $t0, 0xffff      # memory address 0xffff0000
Waitloop:       lw     $t1, 8($t0)      # transmitter control
                andi   $t1,$t1,0x0001   # check ready bit with mask
                beq    $t1,$zero, Waitloop
                sw     $a0, 12($t0)     # data
```

- Processor waiting for I/O called *Polling*

Cost of Polling?

- Assume a processor with a 1 GHz clock takes 400 clock cycles for a polling operation (call polling routine, accessing the device, and returning)

- Determine % of processor time for polling...
  - Mouse polled 30 Hz so as not to miss user movement

% Processor time to poll mouse

- Mouse Polling, Clocks/sec

  $= 30 \times 400$

  $= 12000$ clocks/sec

- % Processor for polling:

$$\frac{12 \times 10^3}{1 \times 10^9} = 0.0012\%$$

- Polling mouse little impact on processor

Cost of Polling?

- Assume a processor with a 1 GHz clock takes 400 clock cycles for a polling operation (call polling routine, accessing the device, and returning)

- Determine % of processor time for polling…
  – Floppy disk transfers data in 2-Byte units
  – Has a data rate of 50 KB/second
  – No data transfer can be missed

% Processor time to poll mouse, floppy

- Frequency of Polling Floppy

  = 50 KB/s /2B = 25K polls/sec

- Floppy Polling, Clocks/sec

  = 25K × 400 = 10,000,000 clocks/sec

- % Processor for polling:

$$\frac{10 \times 10^6}{1 \times 10^9} = 1\%$$

- OK if not too many I/O devices

Cost of Polling?

- Assume a processor with a 1 GHz clock takes 400 clock cycles for a polling operation (call polling routine, accessing the device, and returning)

- Determine % of processor time for polling…

  – Hard disk transfers data in 16-Byte chunks

  – Can transfer at 16 MB/second

  – Again, no transfer can be missed

% Processor time to hard disk

- Frequency of Polling Disk

  = 16 MB/s /16B = 1M polls/sec

- Disk Polling, Clocks/sec

  = 1M × 400 = 400,000,000 clocks/sec

- % Processor for polling:

$$\frac{400 \times 10^6}{1 \times 10^9} = 40\%$$

- Unacceptable!

What is the alternative to polling?

- Wasteful to have processor spend most of its time "spin-waiting" for I/O to be ready

- Want an unplanned procedure call that would be invoked only when I/O device is ready

- Solution: use *exception mechanism* to help I/O by *Interrupting* the program when I/O ready, return when done with data transfer

I/O Interrupt

- An I/O interrupt is like overflow exceptions except that
  - An I/O interrupt is "asynchronous"
  - More information needs to be conveyed

- An I/O interrupt is asynchronous with respect to instruction execution
  - It is not associated with any instruction, but it can happen in the middle of any given instruction
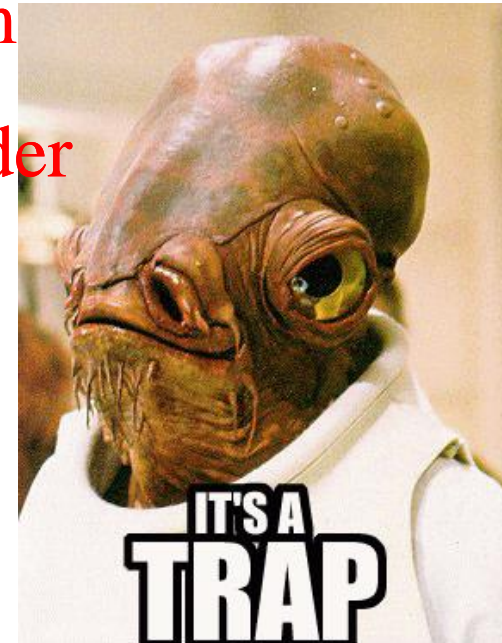 - It does not prevent any instruction from completion

Definitions for Clarification

- Exception:
  - signal marking that something "out of the ordinary" has happened and needs to be handled

- Interrupt:
  - asynchronous exception
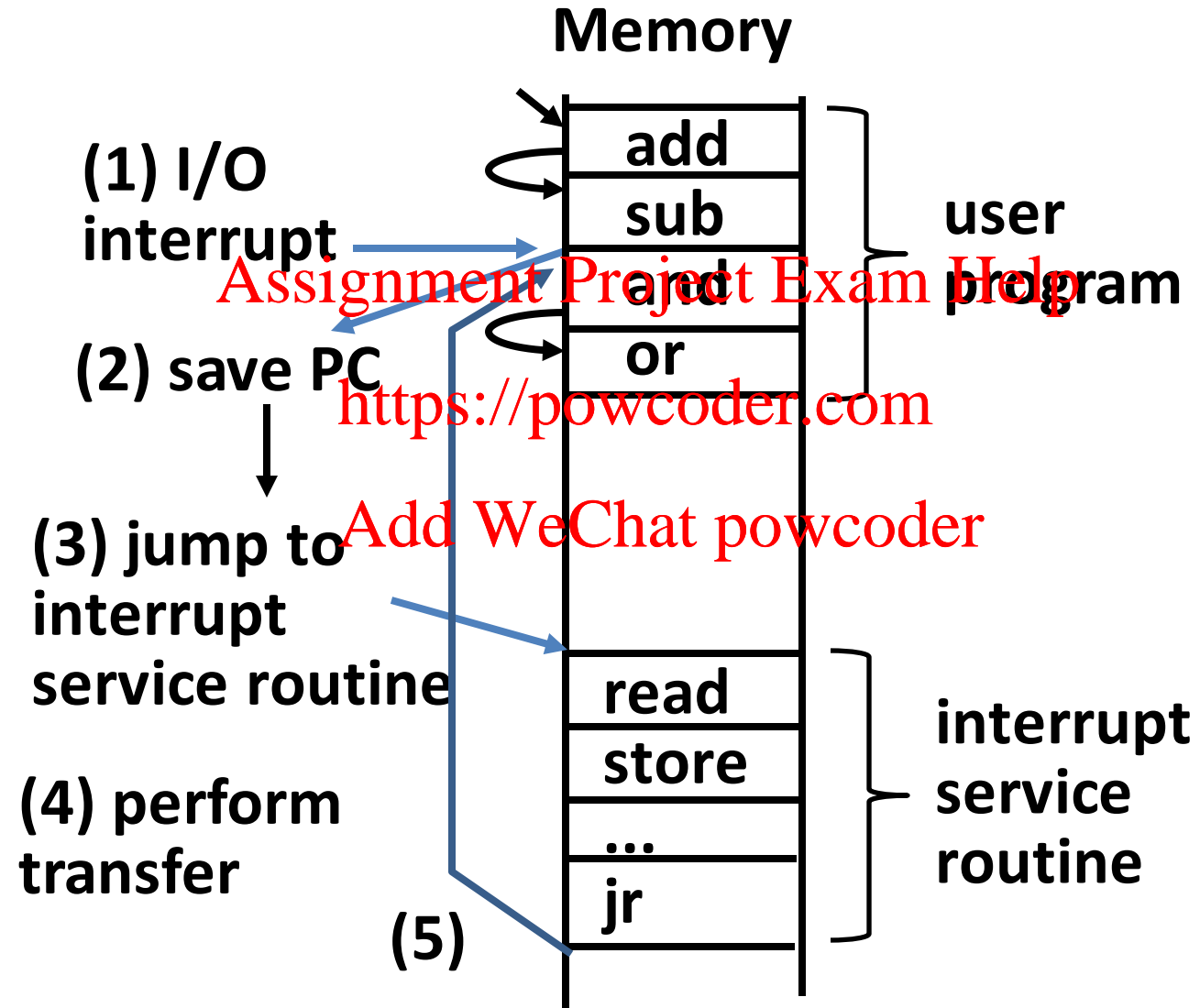
- Trap:
  - synchronous exception

Interrupt Driven Data Transfer

Memory

(1) I/O interrupt

(2) save PC

(3) jump to interrupt service routine

(4) perform transfer

(5)

add
sub
and
or

user program

read
store
...
jr

interrupt service routine

Instruction Set Support for I/O Interrupt

- Save the PC for return

  – But where?

- Where to go when interrupt occurs?

  – MIPS defines location: 0x80000180

- Determine cause of interrupt?

  – MIPS has Cause Register, 4-bit field
    (bits 5 to 2) gives cause of exception

Instruction Set Support for I/O Interrupt

- Portion of MIPS architecture for interrupts called **coprocessor 0**

- Coprocessor 0 Instructions

  – Data transfer: lwc0, swc0
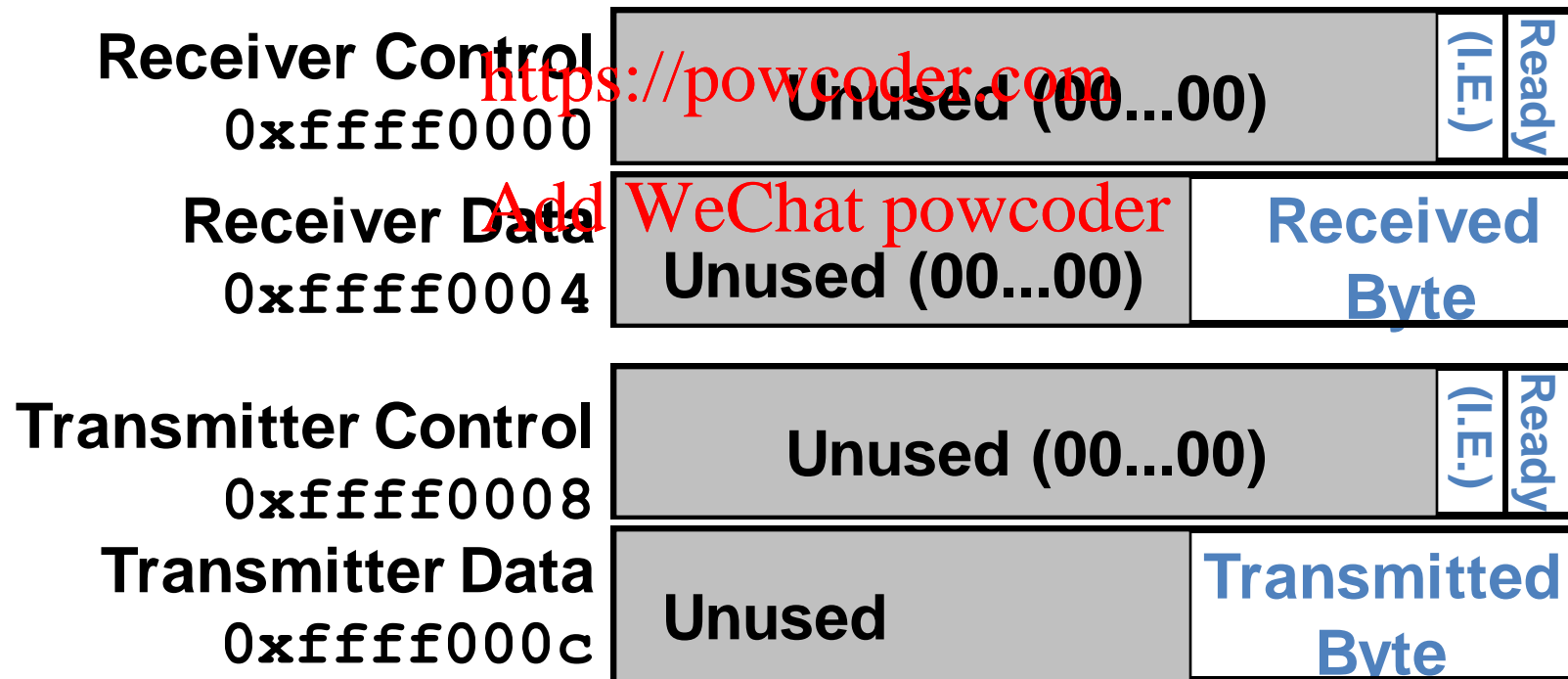
  – Move: mfc0, mtc0

- Coprocessor 0 Registers:

Name	Number	Usage
Status	$12	Interrupt enable
Cause	$13	Exception type
EPC	$14	Return address

Interrupt Driven I/O Simulation

- I.E. is the **Interrupt Enable** bit… set it to 1 to have interrupt occur whenever Ready bit is set

Receiver Control `0xffff0000` — Unused (00…00) | Ready (I.E.)

Receiver Data `0xffff0004` — Unused (00…00) | Received Byte

Transmitter Control `0xffff0008` — Unused (00…00) | Ready (I.E.)

Transmitter Data `0xffff000c` — Unused | Transmitted Byte

Benefit of Interrupt-Driven I/O

- Find the % of 1 GHz processor consumed if the hard disk is only active 5% of the time
  - Assuming 500 clock cycle overhead for each transfer, including the interrupt
  - Also assume the hard disk can transfer at 16 MB/s in 16 byte chunks

Benefit of Interrupt-Driven I/O

- Answer:
  - Disk Interrupts/sec = 16 MB/s /16B
    = 1M interrupts/sec
  - Disk Interrupts, Clocks/sec = 1M × 500
    = 500,000,000 clocks/sec
  - % Processor needed during transfer:
    $$\frac{500 \times 10^6}{1 \times 10^9} = 50\%$$
- Disk active 5% $\Rightarrow$ 5% × 50% $\Rightarrow$ 2.5% busy

Questions Raised about Interrupts

- Which I/O device caused exception?

  – Needs to convey the identity of the device generating the interrupt

- Can avoid interrupts during the interrupt routine?

  – What if more important interrupt occurs while servicing this interrupt?

  – Allow interrupt routine to be entered again?

- Who keeps track of status of all the devices, handle errors, know where to put/supply the I/O data?

4 Responsibilities leading to OS

- The I/O system is shared by multiple programs using the processor

- Low-level control of I/O device is complex because requires managing a set of concurrent events and because requirements for correct device control are often very detailed

- I/O systems often use interrupts to communicate information about I/O operations

- Would like I/O services for all user programs under safe control

4 Functions OS must provide

- Guarantees that user's program accesses only the portions of I/O device to which user has rights (e.g., file access)

- Provides abstractions for accessing devices by supplying routines that handle low-level device operations

- Handles the exceptions generated by I/O devices (and

arithmetic exceptions generated by a program)

- Tries to provide equitable access to the shared I/O resources, as well as schedule accesses to enhance system performance

Things to Remember

- I/O gives computers their 5 senses

- I/O speed range is million to one

- Because of the speed of the processor, it must synchronize with I/O devices before using them (reading or writing)

- Polling works, but expensive

  – Processor repeatedly queries devices

- Interrupts work, but are more complex

  – Device causes an exception, causing OS to run to deal with the device

- I/O control leads to Operating Systems

Review and More Information

- Textbook 5th edition A7 and A8