CS61C Input/Output

Lecture 12

February 26, 1999

Dave Patterson
(http.cs.berkeley.edu/~patterson)

www-inst.eecs.berkeley.edu/~cs61c/schedule.html

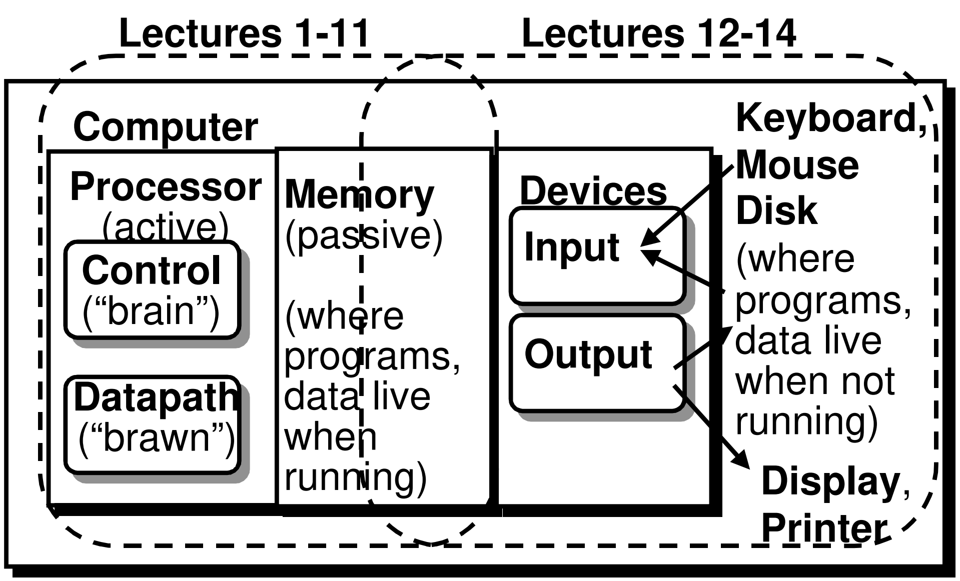
"Review.." 1/1

- **Pointer is high level language version of address**
  - **Powerful yet dangerous concept**
- **Like goto, with self-imposed discipline can achieve clarity and simplicity**
  - **Also can cause difficult to fix bugs**
- **C supports pointers, pointer arithmetic**
- **Java structure pointers have many of the same potential problems!**

Outline

- **Input/Output (I/O) Motivation and Speed**
- **Instruction set support for I/O**
- **Synchronizing Processor and I/O devices**
- **Polling to synchronize**
- **Administrivia, "Computers in the News"**
- **Example I/O interface: SPIM**
- **Weaknesses of Polling**
- **Interrupts to synchronize**
- **Conclusion**

Anatomy: 5 components of any Computer



Motivation for Input/Output

- I/O is how humans interact with computers
- I/O lets computers do amazing things:

  - Read pressure of synthetic hand and control synthetic arm and hand of fireman

  - Control propellers, fins, communicate in BOB (Breathable Observable Bubble)

  - Read bar codes of items in refrigerator
- Computer without I/O like a car without wheels; great technology, but won't get you anywhere

I/O Device Examples and Speeds

- I/O Speed: bytes transferred per second (from mouse to display: million-to-1)
- Device / Behavior / Partner / Data Rate (Kbytes/sec)

Device	Behavior	Partner	Data Rate (Kbytes/sec)
Keyboard	Input	Human	0.01
Mouse	Input	Human	0.02
Line Printer	Output	Human	1.00
Floppy disk	Storage	Machine	50.00
Laser Printer	Output	Human	100.00
Optical Disk	Storage	Machine	500.00
Magnetic Disk	Storage	Machine	10,000.00
Network-LAN	I or O	Machine	10,000.00
Graphics Display	Output	Human	30,000.00

Instruction Set Architecture for I/O

- Some machines have special input and output instructions
- Alternative model (used by MIPS):
  - Input: ~ reads a sequence of bytes
  - Output: ~ writes a sequence of bytes
- Memory also a sequence of bytes, so use loads for input, stores for output
  - Called "Memory Mapped Input/Output"
  - A portion of the address space dedicated to communication paths to Input or Output devices (no memory there)

Processor-I/O Speed Mismatch

- 500 MHz microprocessor can execute a 500 million load or store instructions per second, or 200,000 KB/s data rate
  - I/O devices from 0.01 KB/s to 30,000 KB/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 may 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:
  - 1 register says its OK to read/write (I/O ready), often called <u>Control Register</u>
  - 1 register to contain data, often called <u>Data Register</u>
- Processor reads from Control Register in loop, waiting for device to set Ready bit in Control reg to say its OK ($0 \Rightarrow 1$)
- Processor then loads from (input) or writes to (output) data register
  - Load from device/Store into Data Register resets Ready bit ($1 \Rightarrow 0$) of Control Register

SPIM I/O Simulation

- SPIM simulates 1 I/O device: memory-mapped terminal (keyboard + display)
  - Read from keyboard (<u>receiver</u>); 2 device regs
  - Writes to terminal (<u>transmitter</u>); 2 device regs

Register	Contents		
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	

SPIM I/O

- Control register rightmost bit (0): Ready
  - It cannot be changed by processor (like $0)
  - Receiver: Ready==1 means character in Data Register not yet been read; $1 \Rightarrow 0$ when data is read from Data Reg
  - Transmitter: Ready==1 means transmitter is ready to accept a new character; $0 \Rightarrow$ Transmitter still busy writing last char
    - (I.E. discussed later)
- Data register rightmost byte has data
  - Receiver: last char from keyboard; rest = 0
  - Transmitter: when write rightmost byte, writes char to display

I/O Example

- Input: Read from keyboard into $v0

```
          lui  $t0, 0xffff # ffff0000
Waitloop: lw   $t1, 0($t0) #control
          andi $t1,$t1,0x0001
          beq  $t1,$zero, Waitloop
          lw   $v0, 4($t0) # data
```

- Output: Write to display from $a0

```
          lui  $t0, 0xffff # ffff0000
Waitloop: lw   $t1, 8($t0) #control
          andi $t1,$t1,0x0001
          beq  $t1,$zero, Waitloop
          sw   $a0, 12($t0) # data
```

- Processor waiting for I/O called "<u>Polling</u>"

Administrivia

- Readings: Pointers: COD: 3.11, K&R Ch. 5; I/O 8.3, 8.5, A.7, A.8
- 6th homework: Due 3/3 7PM
  - Exercises 8.1, 8.5, 8.8
- 3rd Project/5th Lab: MIPS Simulator Due Wed. 3/3 7PM; deadline Thurs 8AM
- Upcoming events
  - Midterm on 3/17 5pm-8PM, 1 Pimentel
  - 4th Project due same day as midterm???

  ⇒ 4th Project due Wed. 3/10
  - 2nd online questionnaire when demo 6th lab

"Computers in the News"

- "Intel Demonstrates Performance of New Pentium Microprocessor", NY Times, 2/24/99
  - Intel has tried to position the Pentium III as a "next generation"microprocessor, but industry analysts have generally viewed the 500-MHz chip as merely an incremental advance over the company's 450-MHz Pentium II chips. [In past Intel been closing in on RISC chips.]
  - "If you are just running the same old software it doesn't do much for you," an analyst said. "It's a little faster than the Pentium II 450." Pentium III has additional instructions that offer faster processing of three-dimensional graphics for games, speech recognition processing and video compression.[software?]

Cost of Polling?

- Assume for a processor with a 500-MHz clock it 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 times/sec so as not to miss user movement
  - Floppy disk: transfers data in 2-byte units and has a data rate of 50 KB/second. No data transfer can be missed.
  - "Hard" disk: transfers data in 16-byte chunks and can transfer at 8 MB/second. Again, no transfer can be missed.

% Processor time to poll mouse, floppy

- Mouse Polling Clocks/sec = 30 * 400 = 12000 clocks/sec
- % Processor for polling: $12*10^3/500*10^6$ = 0.002%

  ⇒ Polling mouse little impact on processor
- Times Polling Floppy/sec = 50 KB/s /2B = 25K polls/sec
- Floppy Polling Clocks/sec = 25K * 400 = 10,000,000 clocks/sec
- % Processor for polling: $10*10^6/500*10^6$ = 2%

  ⇒ OK if not too many I/O devices

% Processor time to hard disk

- Times Polling Disk/sec = 8 MB/s /16B
  = 500K polls/sec
- Disk Polling Clocks/sec = 500K * 400
  = 200,000,000 clocks/sec
- % Processor for polling: $200*10^6/500*10^6$
  = 40%

  $\Rightarrow$ Unacceptable

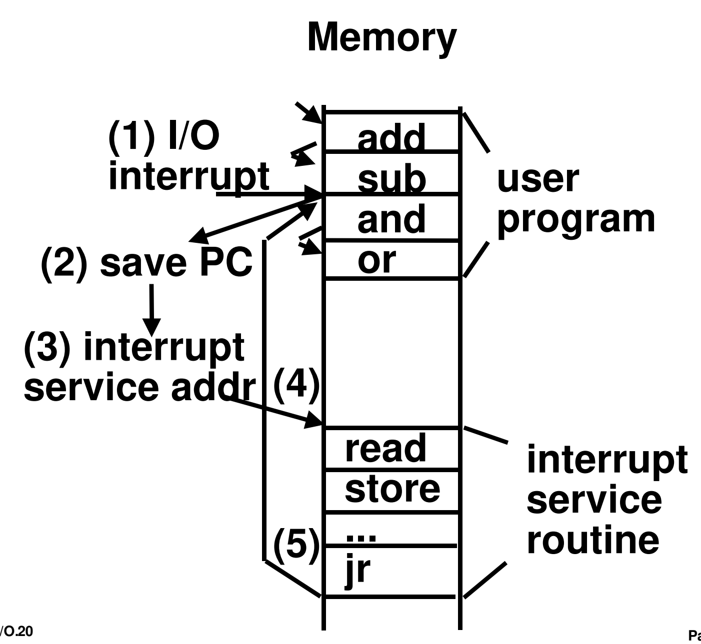
What is the alternative to polling?

- Wasteful to have processor spend most of its time “spin-waiting” for I/O to be ready
- Wish we could have an unplanned procedure call that would be invoked only when I/O device is ready
- Sounds familiar?
- Use exception mechanism for arithmetic overflow to help I/O: interrupt program when I/O ready, return when done with data transfer

I/O Interrupt

- An I/O interrupt is just like the exceptions except:
  - An I/O interrupt is “asynchronous”
  - More information needs to be conveyed
- An I/O interrupt is asynchronous with respect to instruction execution:
  - I/O interrupt is not associated with any instruction
  - I/O interrupt does not prevent any instruction from completion
    - Pick convenient point to take an interrupt

Interrupt Driven Data Transfer



Instruction Set support for I/O Interrupt

- Save the PC for return?
  - Exception Program Counter, like overflow
- Where go when interrupt occurs?
  - MIPS defines location: 0x80000080
- 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*
BadVAddr	`$8`	Bad memory address
Status	`$12`	Interrupt enable
Cause	`$13`	Exception type
EPC	`$14`	Instruction address

SPIM I/O Simulation: Interrupt Driven I/O

- I.E. stands for <u>Interrupt Enable</u>
- Set Interrupt Enable bit to 1 have interrupt occur whenever Ready bit is set

Register	Contents		
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	

Example Interrupt Routine

- Place at 0x80000080

```
.text 0c80000080
mfc0  $k0, $13 # $13 is Cause reg
mfc0  $k1, $14 # $14 is EPC reg
```

  - Why don't have to save $k0, $k1?
- Exception field is bits 5 to 2; 0 ⇒ I/O

```
andi  $k0,$k0,0x003c # select 5-2
bne   $k0,$zero, OtherException
```

- Read byte

```
sw    $ra, save0($0) # save old $31
jal   ReadandStoreByte
lw    $ra, save0($0) # restore $31
jr    $k1
```

Benefit of Interrupt-Driven I/O

- 500 clock cycle overhead for each transfer, including interrupt. Find the % of processor consumed if the hard disk is only active 5% of the time.
- Interrupt rate = polling rate
  - Disk Interrupts/sec = 8 MB/s /16B = 500K interrupts/sec
  - Disk Polling Clocks/sec = 500K * 500 = 250,000,000 clocks/sec
  - % Processor for during transfer: $250 \times 10^6 / 500 \times 10^6 = 50\%$
- 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 enterred 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

- OS guarantees that user's program accesses only the portions of I/O device to which user has rights (e.g., file access)
- OS provides abstractions for accessing devices by supplying routines that handle low-level device operations
- OS handles the interrupts generated by I/O devices, (and arithmetic exceptions generated by a program)
- OS tries to provide equitable access to the shared I/O resources, as well as schedule accesses in order to enhance system performance

"And in Conclusion.." 1/1

- I/O gives computers their 5 senses
- I/O speed range is million to one
- Processor speed means must synchronize with I/O devices before use
- Polling works, but expensive
- Interrupts works, more complex
- I/O control leads to Operating Systems
- Next: Instruction support for OS , Re-entrant Interrupt Routines