Input/Output Mechanics

Examples of Input/Output (I/O) Devices

Standard Interfaces
1. User output
   - Console or Prompt (just text), Video Display, printer, speakers
2. User input
   - Keyboard, mouse, trackball, game controller, scanner, microphone, touch screens, camera (still and video), game controllers
3. Storage
   - Disk drives, CD & DVD drives, flash-based storage, tape drive

More...
Communication
- Network (wired, wireless, optical, infrared), modem
Sensor inputs
- Temperature, vibration, motion, acceleration, GPS
- Barcode scanner, magnetic strip reader, RFID reader
Control outputs
- Motors, actuators

How to Perform I/O?

Require specialized knowledge and protection
- Knowledge of I/O device work
  - Programmers don’t want to know this!
  - Unless you programmer who is writing device drivers
- Protection for shared I/O resources (e.g. Hard-Disk) want process isolation

Solution: service routines or system calls
- Low-level, privileged operations performed by operating system
- Used by almost all applications and hence made a routine
- Example: keyboard & display polling routine for LC-3
- Example: getc, puts in C

LC-3 I/O

I/O is done via the TRAP instruction

<table>
<thead>
<tr>
<th>Code</th>
<th>Equivalent</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OUT</td>
<td>TRAP x21</td>
<td>Write one character (in R0[7:0]) to console.</td>
</tr>
<tr>
<td>GETC</td>
<td>TRAP x20</td>
<td>Read one character from keyboard. Character stored in R0[7:0].</td>
</tr>
<tr>
<td>IN</td>
<td>TRAP x23</td>
<td>Read (and echo) one character from keyboard. Character stored in R0[7:0].</td>
</tr>
<tr>
<td>PUTS</td>
<td>TRAP x22</td>
<td>Write null-terminated string to console. Starting address of string is in R0.</td>
</tr>
</tbody>
</table>
EXAMPLE

:Prompt the user to enter a number, quits when pressed 3, prints DONE before terminating

```
.ORIG x3000
LD R1, VAL3
NOT R1, R1
ADD R1, R1, #1
LEA R0, ENTER ;R0 contains address of first char of string
PUTS ; displays the string
LOOP:
    IN ; read and echo (R0 contains the char)
    AND R3, R3, #0
    ADD R3, R1, #0
    BRnp LOOP
    LEA R0, ENDMSG ; R0 = starting address of string
    PUTS
    HALT
ENTER:    .STRINGZ "Enter a number from 1 to 9:
ENDMSG: .STRINGZ 
VAL3: .FILL x0033
.END
```

System Call

1. User program invokes system call i.e. calls the part of the system code (O.S code) that deals with I/O
   - E.g. TRAP x20 (GETC) in LC3

2. Operating system code performs operation
   - TRAP routine follows calling conventions (callee-save) just like regular user routine

3. Returns control to user program

LC3 TRAP Instruction

```
<table>
<thead>
<tr>
<th>Trap vector</th>
<th>Location in Trap Vector Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1 1 1 0 0 0 0</td>
<td>trapvect8</td>
</tr>
</tbody>
</table>
```

LC3 Trap Vector Table

**Trap Vector Table**

- Used to associate code with trap number
- The table contains the address of where the service routine is located
- The table is stored in memory (x0000 through x00FF)

```
<table>
<thead>
<tr>
<th>Vector Table (Memory location)</th>
<th>Routine location</th>
<th>Routine</th>
</tr>
</thead>
<tbody>
<tr>
<td>x21</td>
<td>x0430</td>
<td>output a character to the monitor</td>
</tr>
<tr>
<td>x23</td>
<td>x04A0</td>
<td>input a character from the keyboard</td>
</tr>
<tr>
<td>x25</td>
<td>xFD70</td>
<td>halt the program (HALT)</td>
</tr>
</tbody>
</table>
```

Note: The service routines are located at location at x0200 - x2FFF in memory except the HALT routine
TRAP Mechanism Operation

1. Lookup starting address
2. Transfer to service routine
3. Return

I/O Basics

Control/Status Registers
- CPU tells device what to do — write to control register
- CPU checks whether task is done — read status register

Data Registers
- CPU transfers data to/from device

Device electronics
- Performs actual operation
  - Pixels to screen, bits to/from disk, characters from keyboard

Programming Interface

How are device registers identified?
- Memory-mapped vs. special instructions

How is timing of transfer managed?
- Asynchronous vs. synchronous

Who controls transfer?
- CPU (polling) vs. device (interrupts)

Memory-Mapped vs. I/O Instructions

Instructions
- Designate opcode(s) for I/O
- Register and operation encoded in instruction

Memory-mapped
- Assign a memory address to each device register
- Use data movement instructions (LD/ST) for control and data transfer
- Hardware intercepts these address
- No actual memory access performed
Transfer Timing

Synchronous
- Data supplied at a fixed, predictable rate
- CPU reads/writes every X time units
- Infrequently used because of speed difference
  - I/O is much slower than the processor
  - E.g. A 300 MHz processor can execute an instruction every 33 nanoseconds
  - E.g. Humans don’t type as fast as a processor can read

Asynchronous
- Data rate less predictable
- CPU must synchronize with device, so that it doesn’t miss data or write too quickly
- Handles the speed mismatch

Transfer Control

Who determines when the next data transfer occurs?

Polling
- CPU keeps checking status register until new data arrives OR device ready for next data
- “Are we there yet? Are we there yet? Are we there yet?”

Interrupts
- Device sends a special signal to CPU when new data arrives OR device ready for next data
- CPU can be performing other tasks instead of polling device
- “Wake me when we get there.”

LC-3 I/O

Memory-mapped I/O

<table>
<thead>
<tr>
<th>Location</th>
<th>I/O Register</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>xFE00</td>
<td>Keyboard Status Reg (KBSR)</td>
<td>Bit [15] is 1 when keyboard has received a new character.</td>
</tr>
<tr>
<td>xFE02</td>
<td>Keyboard Data Reg (KBDR)</td>
<td>Bits [7:0] contain the last character typed on keyboard.</td>
</tr>
<tr>
<td>xFE04</td>
<td>Display Status Register (DSR)</td>
<td>Bit [15] is 1 when device ready to display another char on screen.</td>
</tr>
<tr>
<td>xFE06</td>
<td>Display Data Register (DDR)</td>
<td>Character written to bits [7:0] will be displayed on screen.</td>
</tr>
</tbody>
</table>

Asynchronous devices
- Synchronized through status registers
  - Two ways: Polling and Interrupts
  - We’ll talk first about polling, a bit on interrupts later

Input from Keyboard

When a character is typed:
- Its ASCII code is placed in bits [7:0] of KBDR (bits [15:8] are always zero)
- The “ready bit” (KBSR[15]) is set to 1
- Keyboard is disabled -- any typed characters will be ignored

When KBDR is read by processor:
- KBSR[15] is set to zero
- Keyboard is enabled
Basic Input Routine (GETC)

The interactions described can be applied to most I/O devices.
- The status register in a printer might tell you (via computer):
  - If the printer is printing, or
  - Is out of paper, or out of toner

Data/Information that is transferred can be more than just a byte (known as Block I/O).

General I/O

Role of the Operating System

How does non-privileged code perform I/O?
- Answer: it doesn't; it asks the OS to perform I/O on its behalf

How is this done?
- Making a system call into the operating system

In real systems, only the operating system (OS) does I/O
- "Normal" programs ask the OS to perform I/O on its behalf

Hardware prevents non-operating system code from
- Accessing I/O registers
- Operating system code and data
- Accessing the code and data of other programs

Why?
- Protect programs from themselves
- Protect programs from each other
- Multi-user environments

Memory Protection

Code in privileged/supervisor mode
- Can do anything
- Used exclusively by the operating system

Code in user mode
- Can't access I/O parts of memory
- Can only access some parts of memory

Division of labor
- OS - make policy choices
- Hardware - enforce the OS's policy
OS and Hardware Cooperate for Protection

Hardware support for protected memory
- Consider a n-bit protection register (MPR: memory protection register)
  - Each bit protects some region of memory

When a processor performs a load or store
- Checks the corresponding bit in MPR
- If MPR bit is not set (and not in privileged mode)
  - Trigger illegal access

The OS must set these bits before running each program

MPR in LC3 Example

- Memory access is limited by memory protection register (MPR)
- Each MPR bit corresponds to 4K memory segment
- 1 indicates that users can access memory in this segment

Privilege

Privilege: Processor modes
- Privileged (supervisor) vs. Unprivileged (user) indicated by bit
- E.g. in LC3
  - Encoded in 15th bit of Processor Status Register (PSR)
  - PSR has other info such as Condition Codes (NZP) and Priority Level(PL)

Managing Privilege

Who sets privilege bit in PSR?
- TRAP instruction

Who clears privilege bit?
- An instruction designed to return from trap routines
- Is different from user subroutine return instruction as a user subroutine
Interrupt-Driven I/O

Timing of I/O controlled by device

- Tells processor when something interesting happens
  - Example: when character is entered on keyboard
  - Example: when monitor is ready for next character

- Processor interrupts its normal instruction processing and executes a service routine (like a TRAP)
  - Figure out what device is causing the interrupt
  - Execute routine to deal with event
  - Resume execution

- No need for processor to poll device
  - Can perform other useful work

To implement an Interrupt mechanism

1. Way for I/O device to signal CPU that it wants service
   - We already have a Ready bit indicating that
     - ready bit

2. Way to know that device has a right to request service
   - Interrupt Enable (IE) bit in device register is set by processor
     - Depending on whether processor wants to give service
   - When ready bit is set and IE bit is set, interrupt is signaled

I/O device will get service

To implement an Interrupt mechanism (contd..)

3. Whether I/O device has higher priority among multiple I/O requests AND with the current program in execution
   - Every instruction executes at a stated level of urgency
     - LC-3: 8 priority levels (PL0-PL7) with 7 being higher priority
   - Example:
     - current program runs at PL0
     - Nuclear power correction program runs at PL6
     - It’s OK for PL6 to interrupt PL0 program, but not the other way around
   - A special hardware compare priority levels of the interrupts generated, the one with higher priority gets to use the processor
   - The priority level is encoded in the PSR (Processor Status Register)

Maintaining the state of the machine

You were in the middle of adding 2 numbers in your program, but there was interrupt which needs to get serviced

- Did I complete my operation?
- After I resume, is my data in the registers that I was storing the same as I left off?

Before servicing the interrupt

- Save state of the machine, i.e. Registers, PC, mode and CCs
  - This way I can come back start executing where I left

To service interrupt:

- Load the PC with starting address of the program that is to carry out the requirements of the I/O
Interrupt Implementation using LC3 Example

Interrupt vector: INTV
- 8-bit identification for device
- Also used as index into Interrupt Vector Table, which gives starting address of Interrupt Service Routine (ISR)
  - Just like Trap Vector Table and Trap Service Routine
  - In LC3 Interrupt Vector Table (IVT): x0100 to x01FF

What if more than one device wants to interrupt?
- External logic controls which one gets to go

How Does Processor Handle an Interrupt?

After completing current instruction, If INT (interrupt) =1, then
- Don’t fetch next instruction
- Save state (PC, PSR (privilege and CCs) etc) on Supervisor stack
- Update PSR (set privilege bit i.e. PSR[15] = 1)

Index INTV into IVT to get start address of ISR (put in PC)

After service routine
- RTI (Return from Interrupt) instruction restores machine state from Supervisor stack

Different sections in memory

Example (1)

Different sections in memory

Example (2)

Push PC and PSR onto stack, set privilege bit, find
Device B (INTV=x1d) service routine address in IVT, and transfer control
Note: Have not shown other registers on supervisor stack
Example (3)

Executing AND instruction of Interrupt Routine for device B at x6202 when Device C interrupts.

Example (4)

Push PC and PSR onto stack, set privilege bit, then transfer to Device C service routine (at x6300).

Example (5)

Execute RTI at x6315; pop PSR and PC from stack.

Example (6)

Execute RTI at x6210; pop PSR and PC from stack; continue Program A as if nothing happened!
Software Interrupts

So far we are talking about hardware interrupts i.e. interrupts from I/O devices.

A software interrupt a.k.a exception is an interrupt caused when something unusual happens inside the processor.

- E.g. Divide by zero, overflow etc.