I/O

So Far

Storage
- Registers
- Memory: global variables, stack (for function locals and linkage)

Instructions
- Arithmetic, logical, shift
- Branches, jumps, calls
- Load and store: registers ↔ memory

Software
- C → assembly (LC4) → machine language

How does this thing talk to the outside world?

Readings
P&P
- Chapter 8

Disclaimer
- Differences between LC3 and LC4 are manifest
- Read to get the general concepts and examples
- Specifics (especially code) will vary

Input/Output: Connecting to the Outside World

The “outside world”
- Person
- Another computer
- Some non-computing electro-mechanical device
- Any two of these, or all three

Some examples
- Game console (Playstation, Xbox)
- DVD player
- MP3 player (iPod)
- Cell phone
- Automated Teller Machine (ATM)
- Car’s airbag controller
- Web server
Examples of Input/Output (I/O) Devices

User output
- Display, printer, speakers

User input
- Keyboard, mouse, trackball, game controller, scanner, microphone, touch screens, camera (still and video)

Storage
- Disk drives, CD & DVD drives, flash-based storage, tape drive

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

LC4 I/O Devices

LC4 has 4 I/O devices
- Keyboard (input)
- ASCII console (output)
- 128x124 16b RGB pixel display (output)
- Timer (not really an I/O device but looks like one to software)

I/O Devices and Controllers

Most I/O devices are not purely digital themselves …
- Electro-mechanical: e.g., keyboard, mouse, disk, motor
- Analog/digital: e.g., network interface, monitor, speaker, mic

… all have digital interfaces presented by I/O Controller
- CPU (digital) talks to controller

I/O Controller Interface

I/O controller interface presented as device registers
- Control/Status: may be one register or two
- Data: may be more than one of these

For input
- CPU checks status register if input is available
- Reads input from data register (or waits if no input available)

For output
- CPU checks status register to see if can write to the device
- Writes output to data register
How are Device Register Reads/Writes Performed?

Two options

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

Memory-mapped I/O
- Assign a memory address to each device register
- Use conventional loads and stores
- Hardware intercepts loads/stores to these address
- No actual memory access performed
- LC4 (and most other platforms) do this

LC4 I/O Device Registers

Keyboard status register (KBSR): xFE00
- KBSR[15] is 1 if keyboard has new character

Keyboard data register (KBDR): xFE02
- KBDR[7:0] is last character input on keyboard

ASCII display status register (ADSR): xFE04
- ADSR[15] is 1 if console ready to display next character

ASCII display data register (ADDR): xFE06
- ADDR[7:0] is written to console

LC4 Pixel-Based Video Display

LC4 has a 128x124 pixel display
- Each pixel comprises of RGB color
- Each color is encoded in 5 bits

Entire display is memory-mapped
- One memory location per pixel
- Memory region xC000-xFDFF
- xC000-xC07F is first row, xC080-xC0FF is second row, etc.
- Memory location to set pixel color
LC4 Pixel-Based Video Display Modes

LC4 pixel-video display operates in two modes:

**Direct mode**
- Pixel changes as soon as you write location

**Double-buffered mode**
- Pixel doesn’t change right away
- Pixels change en masse when 1 written to VDCR
- Video memory (but not display) cleared when 0 written to VDCR
- This mode is faster (don’t need to erase) and looks nicer
- Available via PennSim -d option

---

How Do We Access I/O Devices In Software?
E.g. Read a character from the keyboard (equivalent to getc subroutine)

```
.new char?

// wait for keyboard input, load into R5
(GETC

// loop while KBSR[15] == 0
LC R0, OS_KBSR_ADDR
LDR R0, R0, #0
BRzp GETC

// load character from mem-mapped KBDR
LC R0, OS_KBDR_ADDR
LDR R0, R0, #0
```

---

Assembly For Keyboard Read

```
OS_KBSR_ADDR .UCONST xFE00
OS_KBDR_ADDR .UCONST xFE02

.CODE

; wait for keyboard input, load into R5
GETC

; loop while KBSR[15] == 0
LC R0, OS_KBSR_ADDR
LDR R0, R0, #0
BRzp GETC

; load character from mem-mapped KBDR
LC R0, OS_KBDR_ADDR
LDR R0, R0, #0

A few new things here, let’s dissect
```
Assembler Constants

OS_KBSR_ADDR .UCONST xFE00

. UCONST
  • Associates 16-bit constant with preceding label
  • Why not just use .FILL?
    ➢ .FILL directives show up in data memory
    ➢ .UCONST directives don’t

LC R0, OS_KBSR_ADDR ; R0 = xFE00

LC (Load Constant)
• Assembler pseudo-instr similar to LEA
• Expands into CONST, HICONST pair
• Loads value at label rather than address of label

Assembly For “Echoed” Keyboard Read

OS_KBSR_ADDR .UCONST xFE00
OS_KBDR_ADDR .UCONST xFE02
OS_ADSR_ADDR .UCONST xFE04
OS_ADDR_ADDR .UCONST xFE06

.CODE
EGETC
LC R0, OS_KBSR_ADDR ;; lop while KBSR[15]==0
LDR R0, R0, #0
BRzp EGETC
LC R0, OS_KBDR_ADDR
LDR R0, R0, #0
EGETC
LC R1, OS_ADSR_ADDR ;; loop while ADSR[15]==0
LDR R1, R1, #0
BRzp EGETC
LC R1, OS_ADDR_ADDR
STR R0, R1, #0

I/O and the Operating System (OS)

In real systems, only the operating system (OS) does I/O
• “User” programs ask OS to perform I/O on their behalf
• Three reasons for this setup

Abstraction/Standardization
• I/O device interfaces are nasty, and there are many of them
• Think of disk interfaces: S-ATA, iSCSI, IDE
• User programs shouldn’t have to deal with these interfaces
  ➢ In fact, even OS doesn’t have to deal with most of them
  ➢ Most are buried in “device drivers”

Isolation

Synchronization
• What are these last two things?

Operating Systems (OSes)

OSes virtualize the hardware for user applications
• This is a good thing

Raise the level of abstraction
• Wrap physical interfaces with nice logical ones
  ➢ Wrap disk layout in file system interface (More in CIT595)

Enforce isolation (usually with help from hardware)
• Each user program thinks it has the hardware to itself
  ➢ User programs unaware of other programs or (mostly) OS
• Makes programs much easier to write
• Makes the whole system more stable and secure
  ➢ A can’t mess with B if it doesn’t even know B exists
Aside: Virtual Machine Monitors (VMMs)

VMMs virtualize hardware for Oses
- Why is another virtualization layer important?
- Isn’t one enough?

VMMs usually provide just isolation, no abstraction
- Or very little abstraction
- OS can’t tell whether it’s running on hardware ...
- … or on top of VMM
- Several advantages to this setup

I/O, Isolation, and the OS

So what was that point about isolation?
- Three main hardware pieces: CPU, memory, I/O devices
- How does OS provide user-program-level isolation for each?

CPU isolation provided via context switching
- Time multi-plexing
- LC4 OS doesn’t do this (but it could)

Memory isolation provided via virtual memory
- Time and space multi-plexing using hardware assisted indirection
- LC4 OS doesn’t do this either (but it could)

I/O isolation provided by mediating all I/O via OS
- LC4 OS does do this - there are no other programs to isolate from

More details in CIT595

Implementing an OS: Privilege

OS isolates user programs from each other and itself
- Requires restricted access to certain parts of hardware to do this
- Restricted access should be enforced by hardware
- Acquisition of restricted access should be possible, but restricted

Restricted access mechanism is called privilege
- Hardware supports two privilege levels

“Supervisor” or “privileged” mode
- Processor can execute any code, read/write any data

“User” or “unprivileged” mode
- Processor may not execute some code, read/write some memory
  ➢ E.g., cannot read/write video memory or device registers

Privilege in LC4

Remember PSR (Processor Status Register)?
- PSR[15] is the privilege bit
- If PSR[15] = 1, current code is “privileged”, i.e., the OS

LC4 instruction and data memories split into two
- x0000-x7FFF: user segment
- x8000-xFFFF: OS segment
  ➢ Video memory (xC000-xFDFF) is in OS segment
  ➢ I/O device registers (xFE00-xFFFF) are too

If PSR[15]==0 and current program tries to …
- … execute an instruction with PC[15] == 1
- … or read/write data with address[15] == 1
- … LC4 hardware raises an exception and abort the program
LC4 OS Data Memory

Five regions
- \(x8000-x9FFF\): instructions
- \(xA000-xAFFF\): global variables
- \(xB000-xBFFF\): stack
- \(xC000-xFDFF\): video memory
- \(xFE00-xFFFF\): device registers
  - Memory-mapped
- No heap (no need, yet)

Next Time

How does user code call OS to perform I/O?
- Just a normal JSR is not "secure" enough
  - Can JSR to any 16-word aligned address
  - Can use bad parameters to crash OS or other applications
  - May be able to surreptitiously upgrade privilege

Mechanism that is used instead is called system call
- Sometimes called syscall, trap, or callgate
- Can call to a restricted set of function addresses
- Can upgrade privilege only through these channels
- Requires two (new) special instructions
  - TRAP: restricted call with privilege upgrade
  - RTI: return with privilege downgrade