Computer Systems and Networks

ECPE 170 – Jeff Shafer – University of the Pacific

Exam Review

Exam Overview

- **Final Exam** – Thursday, December 15th
  - 8am – Regular classroom
  - Comprehensive
  - **No calculators**
  - **Exam is optional if you are happy with your 3 earlier exam scores!**
- Exam 3 grading is not yet finished
  - ETA: This afternoon(?)
  - Check your email for instructions on how to pick up

Exam Overview

- **The *majority* of final exam questions will be heavily based on prior Quizzes and Exams**
  - **If you didn't know how to do a problem before, learn now!**
- Recommended you study
  - Old exams!
  - Old quizzes! *(and solutions on Sakai)*
  - Lecture notes (particularly for later chapters)
  - Old homework assignments
- ***I don't have formal office hours next week, but I am happy to answer any questions you have – email me to setup a time!***

Sakai Gradebook

- Please make sure that all entries in the Sakai gradebook have been recorded correctly

Chapter 2 – Data Representation

Chapter 2 Topics

- Binary Numbers
  - Binary <-> Decimal
  - Both integer and fractional numbers
- Hexadecimal
- Floating-Point Numbers
  - *IEEE single and double precision formats only – I don't care about the book's simplified format*

Division Method: Decimal to Binary

Convert 789_{10} to binary

789 / 2 = 394.5	Remainder of 1
394 / 2 = 197	Remainder of 0
197 / 2 = 98.5	Remainder of 1
98 / 2 = 49	Remainder of 0
49 / 2 = 24.5	Remainder of 1
24 / 2 = 12	Remainder of 0
12 / 2 = 6	Remainder of 0
6 / 2 = 3	Remainder of 0
3 / 2 = 1.5	Remainder of 1
1 / 2 = 0.5 (stop when <1)	Remainder of 1

Divide by 2 since we're converting to binary (base 2)

Read bottom to top:

$789_{10} = 1100010101_2$

Multiplication Method: Decimal to Binary

Convert 0.8125_{10} to binary

0.8125 * 2 = 1.625	1 (whole number)
0.625 * 2 = 1.25	1
0.25 * 2 = 0.5	0 (no whole number)
0.5 * 2 = 1.0	1

Stop when you reach 0 fractional parts remaining (or you have enough binary digits)

Read top to bottom:
0.8125_{10} = $.1101_2$

Signed Integer Representation

- To represent signed integers, computer systems use the high-order bit to indicate the sign
  - `0xxxxxxxx` = Positive number
  - `1xxxxxxxx` = Negative number

- Three ways in which signed binary integers may be expressed:
  - Signed magnitude
  - One's complement
  - Two's complement

IEEE Floating-Point Representation

- The IEEE has established standards for floating-point numbers
- **IEEE-754 single precision** standard (32 bits long)
  - 8-bit exponent (with a **bias of 127**)
  - 23-bit significand
  - A “float” in C++
- **IEEE-754 double precision** standard (64 bits long)
  - 11-bit exponent (with a **bias of 1023**)
  - 52-bit significand
  - A “double” in C++

IEEE Floating-Point Representation

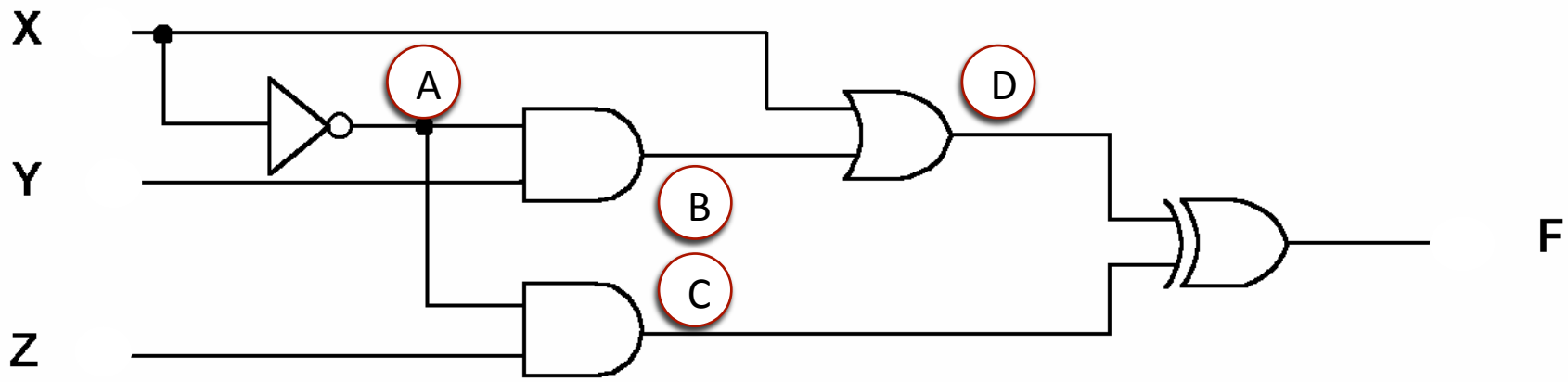
- **Key difference: Significand is normalized differently**
  - **Implied 1** to the **left** of the radix point, i.e. formatted as 1.xxx…
  - For example, $4.5 = .1001 \times 2^3$ in IEEE format is $4.5 = 1.001 \times 2^2$
  - The 1 is **implied**, which means it **is not saved in computer memory**
    - The stored significand would include only 001
    - Optimization – This saves a bit!

Chapter 3 – Digital Logic

Chapter 3 Topics

- Boolean Algebra
  - Gates: AND, OR, NOT, XOR, NAND, NOR
- Combinatory (combinational) circuits
  - Draw a circuit from a Boolean equation
  - Derive a truth table from a circuit (or equation)
- Sequential circuits
  - SR, JK, D flip-flops and their truth tables
- Karnaugh Maps? *Not on the final…*
- Simplification with identities? *Not on the final…*

Combinational Circuit



x	y	z	A	B	C	D	F(x,y,z)
0	0	0	1	0	0	0	0
0	0	1	1	0	1	0	1
0	1	0	1	1	0	1	1
0	1	1	1	1	1	1	0
1	0	0	0	0	0	1	1
1	0	1	0	0	0	1	1
1	1	0	0	0	0	1	1
1	1	1	0	0	0	1	1

Chapter 4 – MARIE

Chapter 4 Topics

- Simple computer organization (generic)
  - Memory organization – building a main memory out of many smaller RAM chips
  - **What's the difference between byte-addressable and word-addressable?**
- MARIE organization and ISA
- MARIE assembly programming
  - *Write a subroutine that does XYZ...*
  - *Trace a program – what is the final value in the ACC?*
  - *You be the assembler…*

Example – Memory Organization

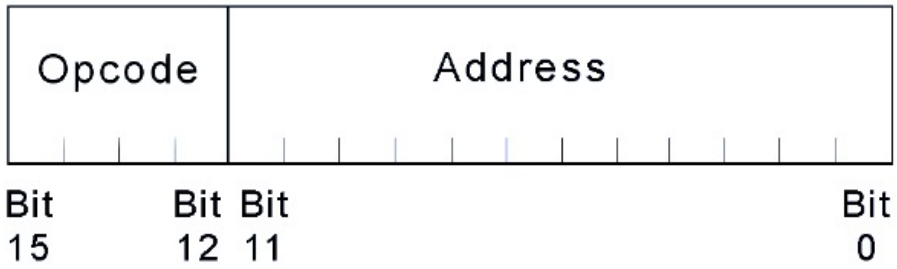
- Example: Suppose we build a 8M x 32 word-addressable main memory using 512K x 8 RAM chips.
- **How many RAM chips are necessary?**
  - 8M/512K * 32/8 = 16 * 4 = 64
- **How many RAM chips are there per word?**
  - 32/8 = 4 chips per word
- **How many address bits are needed per RAM chip?**
  - 512K addresses = 2^{10+9} = 19 address bits
- **How many banks will there be?**
  - 8M/512K = 16 banks

Example – Memory Organization

- Example: Suppose we build a 8M x 32 word-addressable main memory using 512K x 8 RAM chips.
- **How many address bits are needed for all memory?**
  - Word addressable: 8M addresses = $8*2^{20} = 2^{20+3}$ = 23 address bits
  - Byte-addressable, 8M addresses * 4 bytes per word = 2^{20+3+2} = 25 address bits
- **If high-order interleaving is used, where would address 247193_{16} be located?**
  - **Bank 4** (247193_{16} = 010 0100 0111 0001 1001 0011_2)
- **If low-order interleaving is used, where would address 247193_{16} be located?**
  - **Bank 3** (247193_{16} = 010 0100 0111 0001 1001 0011_2)

MARIE Instructions

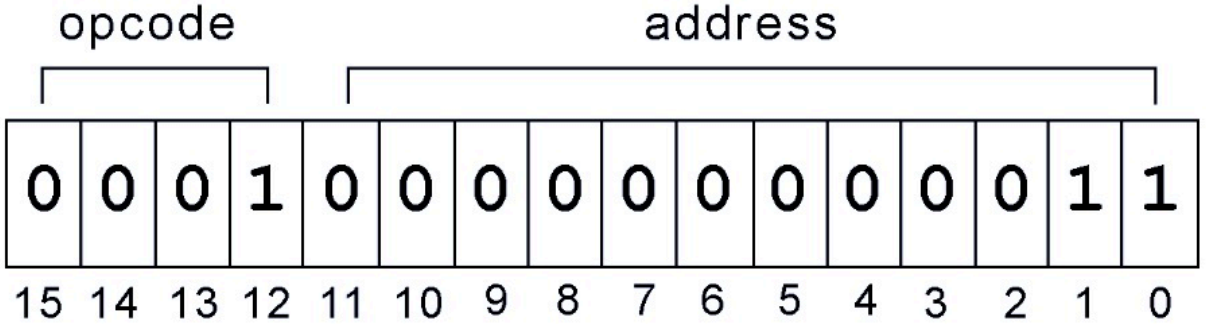
- **How does MARIE format instructions in computer memory?**



- Two fields
  - **Opcode** (4 bits) – Operation code
  - **Address** (12 bits) – Address to operate to/from

MARIE Instruction Example - LOAD

- Bit pattern for a `LOAD` instruction
  - Could be saved in memory or IR (if executing right now)



- **Decode this instruction**
  - Opcode is 1 – LOAD instruction
  - Address is 3 – Data will be loaded from here
- **What is the Hex representation of this instruction?**

Recap – MARIE Instructions (Full)

Binary	Hex	Instruction	Meaning
0001	1	**LOAD X**	Load contents of address X into AC
0010	2	**STORE X**	Store contents of AC at address X
0011	3	**ADD X**	Add contents of address X to AC
0100	4	**SUBT X**	Subtract contents of address X from AC
0101	5	**INPUT**	Input value from keyboard into AC
0110	6	**OUTPUT**	Output value in AC to display
0111	7	**HALT**	Terminate program
1000	8	**SKIPCOND**	Skip next instruction on condition based on AC value
1001	9	**JUMP X**	Load value of X into PC
1010	A	**CLEAR**	Set AC to 0
1011	B	**ADDI X**	Add contents of address Mem[X] to AC
1100	C	**JUMPI X**	Load contents of address Mem[X] into PC
1101	D	**LOADI X**	Load contents of address Mem[X] into AC
1110	E	**STOREI X**	Store contents of AC at address Mem[X]

See table 4.7 in book!

Assembler Operation

- Example program
  - HEX and DEC directives to specify radix of constants
- Assembler **Pass #1**
  - Create symbol table
  - Create partially-assembled instructions

Address	Instruction
100	Load X
101	Add Y
102	Store Z
103	Halt
104 X,	DEC 35
105 Y,	DEC -23
106 Z,	HEX 0000

Symbol Table: ***Name, Address***

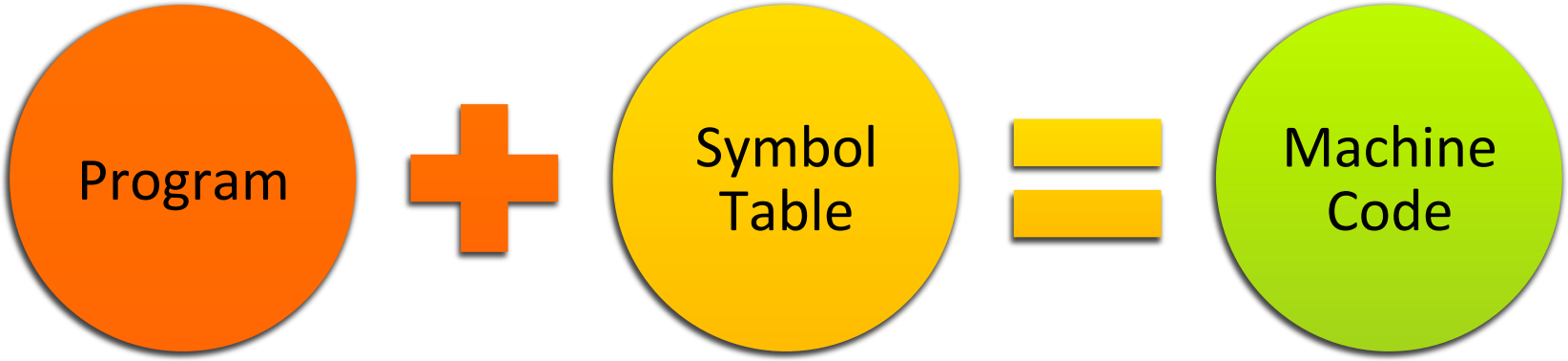
X	104
Y	105
Z	106

Partially-Assembled Program:

1	X
3	Y
2	Z
7000	

Assembler Operation

- Assembler **Pass #2**
  - Fill in details from symbol table



Assembler Operation

Program:

Address	Instruction
100	Load X
101	Add Y
102	Store Z
103	Halt
104 X,	DEC 35
105 Y,	DEC -23
106 Z,	HEX 0000

Symbol Table:

X	104
Y	105
Z	106

Machine Code:

1 1 0 4
3 1 0 5
2 1 0 6
7 0 0 0
0 0 2 3
F F E 9
0 0 0 0

Chapter 5 – Instruction Sets

Chapter 5 Topics

- Endianness
- Stack, accumulator, and register-based machines
- Infix versus postfix notation
- Expanding opcodes? *Not on the final…*
- Addressing Modes
  - Immediate, direct, indirect, indexed
- Pipelines and pipeline hazards

Endianness

- **What is a little endian computer system?**
  - Little-endian: lower bytes come first (stored in lower memory addresses)
  - Ex: Intel x86/x86-64
- **What is a big endian computer system?**
  - Higher bytes come first
  - Ex: IBM PowerPC

Gulliver's Travels

Endianness

- 32-bit number 0x456789A1 starting at address 0x10
  - **How is this saved in memory on a big endian system? On a little endian system?**

Address	Big-Endian	Little-Endian
0x10	45	A1
0x11	67	89
0x12	89	67
0x13	A1	45

One byte (8 bits) per location!

Related Problem

Addr	Value
0x10	45
0x11	67
0x12	89
0x13	A1

- **If the data starting at address 10 is interpreted on a little-endian system as an IEEE 754 single-precision value, what is the decimal value?**
- Read off number in correct order (0xA1896745) and convert to binary:
  - 1010 0001 1000 1001 0110 0111 0100 0101
- Interpret:
  - Sign: 1 (negative)
  - Exp: 01000011 (67 -127 = -60)
  - Significand: **1.**00010010110011101000101
- Result: **$-1.00010010110011101000101 \times 2^{-60}$**

Stack vs Accumulator vs Register

- **Stack architecture** (0-address machine)
  - Instructions and operands are implicitly taken from the stack
  - Stack cannot be accessed randomly
- **Accumulator architecture** (1-address machine)
  - One operand of a binary operation is implicitly in the accumulator
  - One operand is in memory, creating lots of bus traffic
- **General purpose register (GPR) architecture** (2 or 3 address machine)
  - Registers can be used instead of memory
  - Faster than accumulator architecture
  - Efficient implementation for compilers
  - Results in longer instructions

Infix Expression and ISA

- Let's see how to evaluate an infix expression using different instruction formats
- With a three-address ISA, (e.g.,mainframes), the infix expression

  Z = X × Y + W × U

  might look like this
  - ```
    MULT R1,X,Y
    MULT R2,W,U
    ADD  Z,R1,R2
    ```
    ```

Infix Expression and ISA

- In a two-address ISA, (e.g., Intel, Motorola), the infix expression

  `Z = X × Y + W × U`

  might look like this

  - ```
    LOAD R1,X
    MULT R1,Y
    LOAD R2,W
    MULT R2,U
    ADD  R1,R2
    STORE Z,R1
    ```

Note: Two-address ISAs usually require one operand to be a register
    ```

Infix Expression and ISA

- In a one-address ISA, like MARIE, the infix expression `Z = X × Y + W × U` looks like this:

```
LOAD X
MULT Y
STORE TEMP
LOAD W
MULT U
ADD TEMP
STORE Z
```

Notice that as the instructions get shorter, the program gets longer…

Tradeoff – Hopefully these small instructions are faster than the large instructions!

Postfix Expression and ISA

- In a stack ISA, the postfix expression

  Z = X Y × W U × +

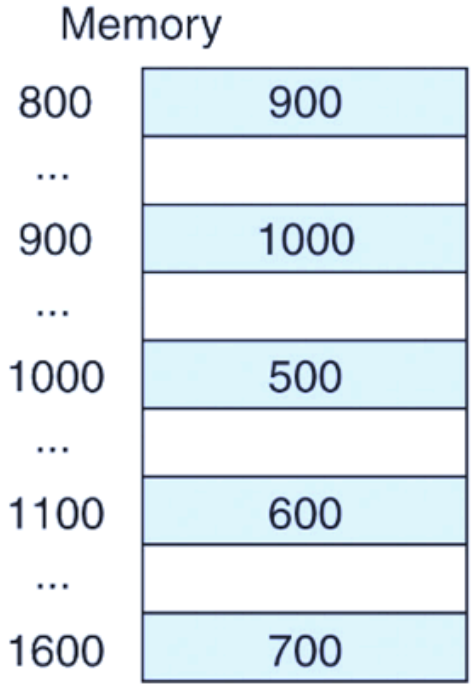
  might look like this:

  ```
  PUSH X
  PUSH Y
  MULT
  PUSH W
  PUSH U
  MULT
  ADD
  POP Z
  ```

Would this program require more execution time than the corresponding (shorter) program that we saw in the 3-address ISA?

Addressing Modes

- **Exercise: For the instruction shown, what value is loaded into the accumulator for each addressing mode?**



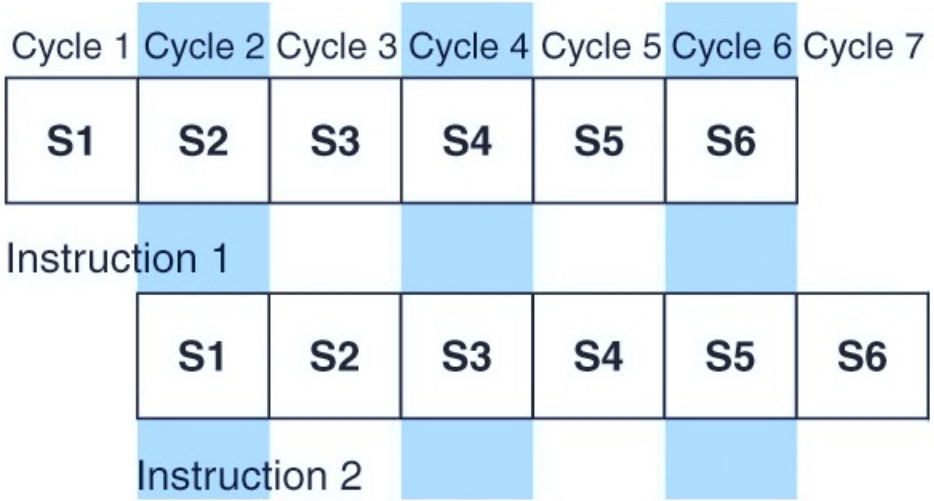
LOAD 900

Mode	Value Loaded into AC
Immediate	
Direct	
Indirect	
Indexed	

Instruction Pipelining

- For every clock cycle, one small step is carried out, and the stages are **overlapped**

S1. Fetch instruction
S2. Decode opcode
S3. Calculate effective address of operands
S4. Fetch operands
S5. Execute
S6. Store result



Pipeline Hazards

- We might not always be able to keep the pipeline full of instructions
  - **Hazards** cause pipeline conflicts and stalls
- Key hazards types
  - **Data hazards**
    - Instruction B depends on data produced by instruction A
  - **Structural hazards**
    - Instruction A and B require the same resource at the same time, e.g. shared memory
  - **Control hazards**
    - Instruction B will only execute if instruction A is true (or false) – we don't know the correct instruction to put in the pipeline now!

Instruction-Level Pipelining

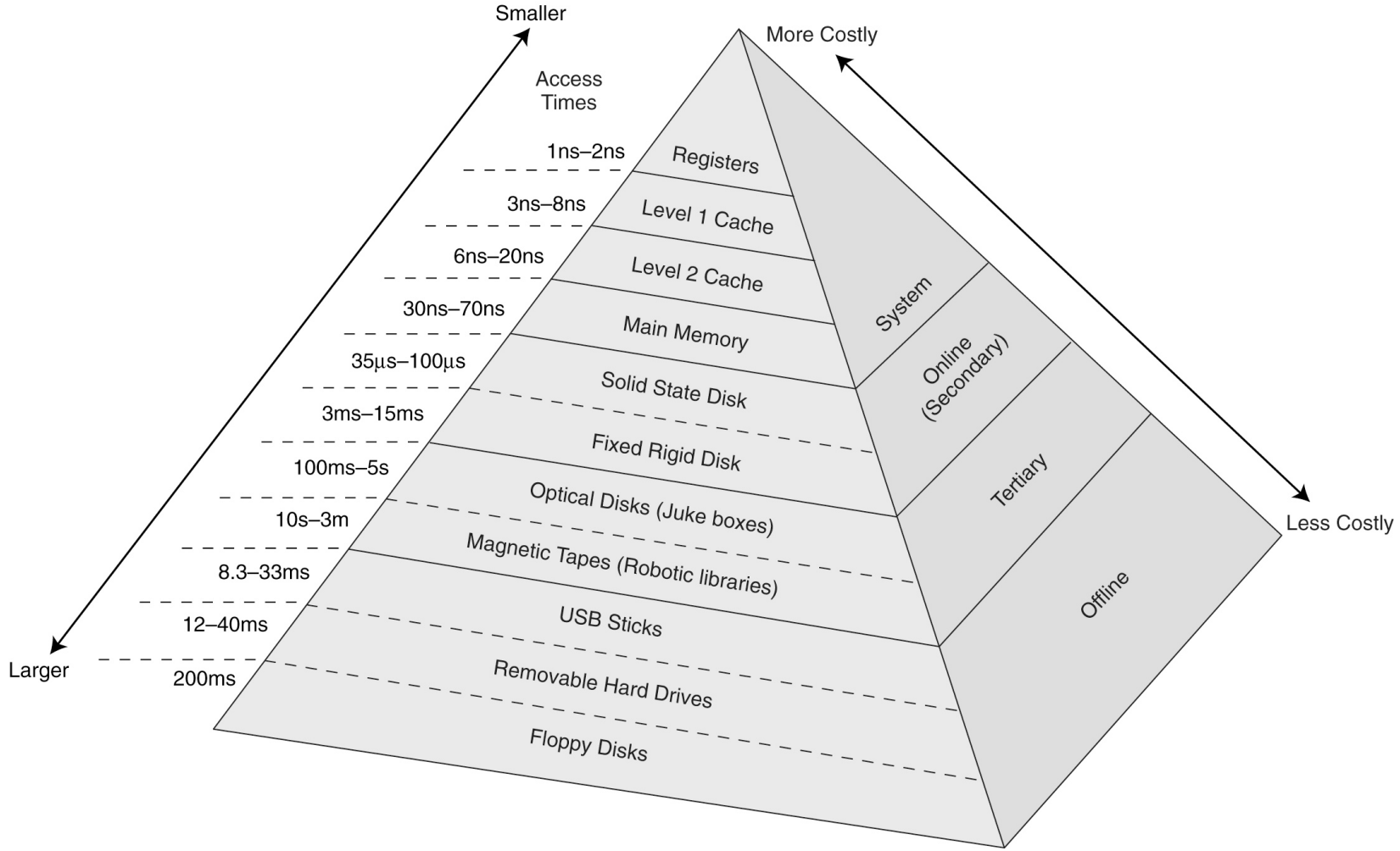
- Hazards can cause pipeline to **stall** or **flush**
  - **Stall** – pipeline is delayed for a cycle
  - **Flush** – all instructions in pipeline are deleted
- Clever hardware or clever assembly programmers (or *optimizing* compilers) can reduce the effects of these hazards
  - But not fully eliminate them…

Chapter 6 – Memory Systems

Chapter 6 Topics

- Caches
  - Direct-Map Cache
  - N-Way Set Associative Cache
  - Fully Associative Cache
- Virtual Memory and Paging

The Memory Hierarchy



Direct Mapped Cache Search

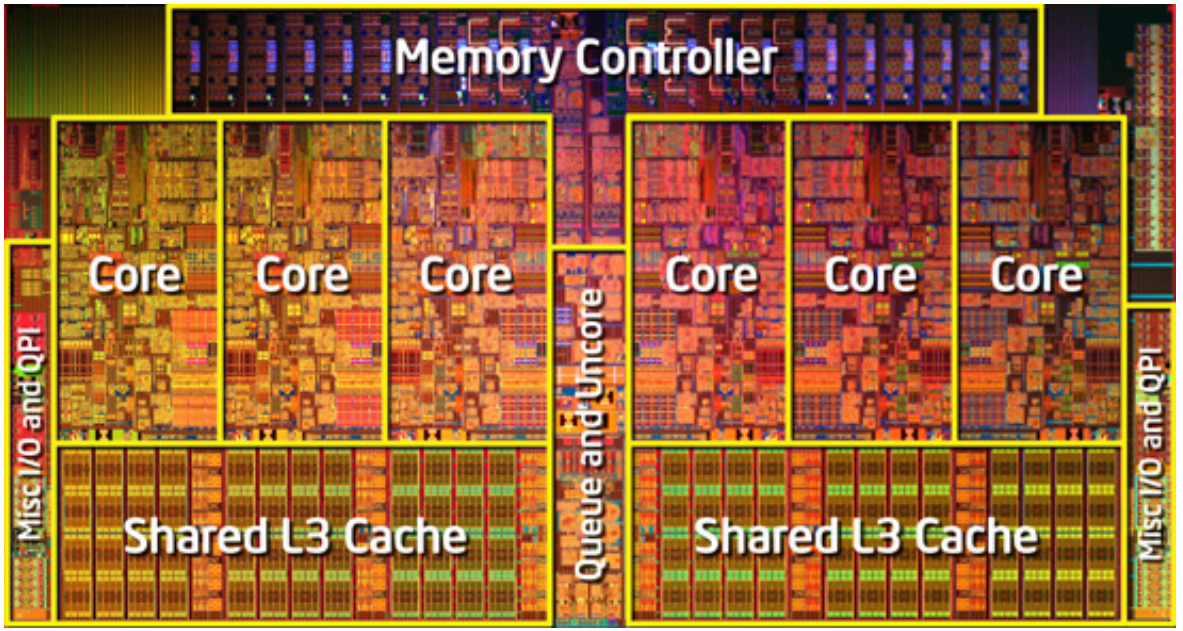
1. Take the main memory address of desired data
    1. Split into **tag**, **block**, and **offset** fields (varies by cache and block size)
2. Go to the indicated block in the cache
3. Does the tag saved in the cache match the search tag?
   Is the block marked as valid?
    1. Yes on both – we have a **cache hit**! ☺
        1. Retrieve the data (go to the byte/word indicated by offset)
    2. Otherwise, we have a **cache miss**! ☹
        1. Need to go to main memory and get the data
        2. Load in the full block from main memory into the cache

Cache Memory

- **What is a cache?**
  - Speed up memory accesses by storing recently used data closer to the CPU
  - **Closer** that main memory – on the CPU itself!
- Although cache is much smaller than main memory, its access time is a fraction of that of main memory
- Cache is **automatically** managed by the memory system
- **What two caches have we specifically learned about?**
  - Main memory cache and TLB (cache of page table)

Cache Example – Intel Core i7 980x

- High-end 6 core processor with a sophisticated multi-level cache hierarchy
- 3.5GHz, 1.17 billion transistors (!!!)



Cache Example – Intel Core i7 980x

- Each processor core has its own a L1 and L2 cache
  - 32kB Level 1 **(L1)** data cache
    - 8-way set associative, 64 byte block ("line") size
  - 32kB Level 1 (L1) instruction cache
    - 4-way set associative, 64-byte block size
  - 256kB Level 2 **(L2)** cache (both instruction and data)
    - 8-way set associative, 64-byte block size
- The entire chip (all 6 cores) **share** a single 12MB Level 3 **(L3)** cache
  - 16-way set associative, 64-byte block size

Cache Example – Intel Core i7 980x

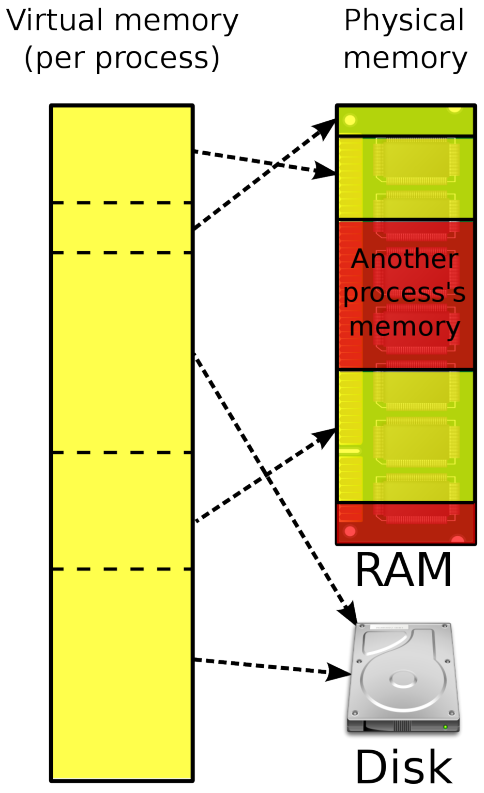
- Access time? (Measured in 3.5GHz clock cycles)
  - 4 cycles to access L1 cache
  - 9-10 cycles to access L2 cache
  - 48 cycles to access L3 cache
- Smaller caches are faster to search
  - And can also fit closer to the processor core
- Larger caches are slower to search
  - Plus we have to place them further away
- *There are also multiple levels of TLB caches too (caching page table entries…)*

Virtual Memory

Virtual Memory is a BIG LIE!

- We **lie** to your application and tell it that the system is simple:
  - Physical memory is infinite! (or at least huge)
  - You can access *all* of physical memory
  - Your program starts at *memory address zero*
  - Your memory address is *contiguous* and *in-order*
  - Your memory is *only RAM* (main memory)

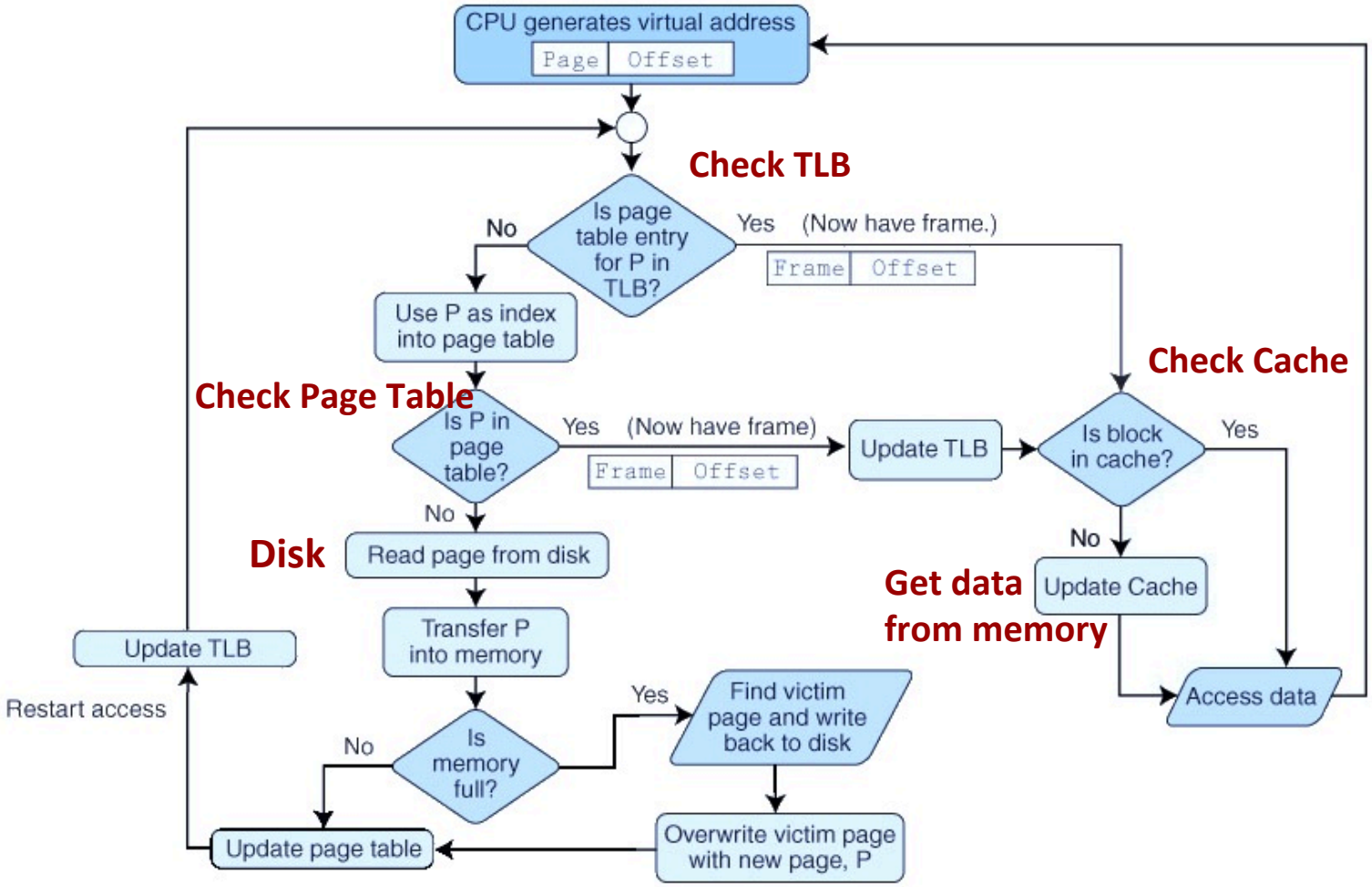
What the System Really Does



Why use Virtual Memory?

- We want to run multiple programs on the computer concurrently *(multitasking)*
  - Each program needs its own separate memory region, so physical resources must be divided
  - The amount of memory each program takes could vary dynamically over time (and the user could run a different mix of apps at once)
- We want to use multiple types of storage (main memory, disk) to increase performance and capacity
- We don't want the programmer to worry about this
  - Make the processor architect handle these details

TLB + Page Table + Main Memory + Cache



Since the third exam material (Chapters 7, 8, 9, 12) is more recent, I'm not discussing it much in the review. But don't take that to mean there won't be questions on the final!

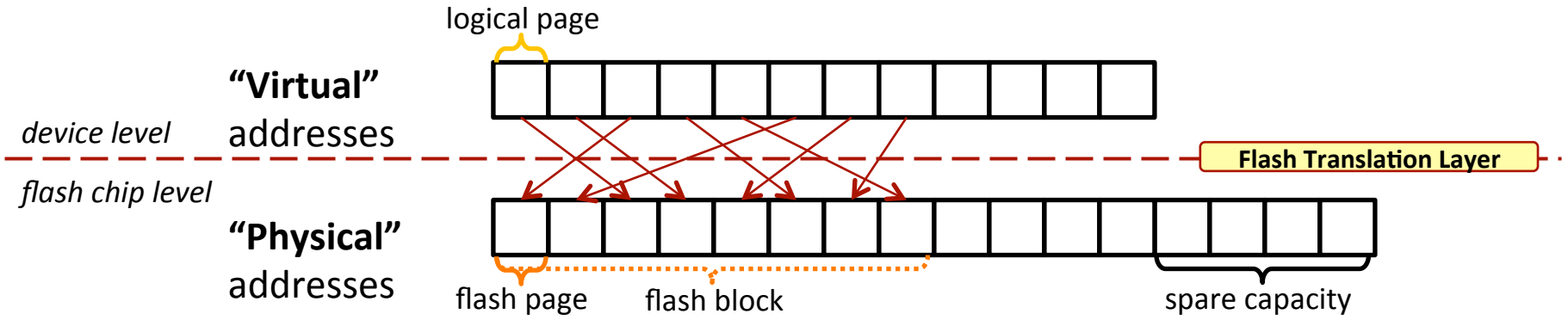
Chapter 7 – Input/Output

Chapter 7 Topics

- Hard drives
- RAID
- SSDs

Flash Translation Layer

- **Flash Translation Layer (FTL)**
  - Necessary for flash reliability and performance
  - **"Virtual" addresses** seen by the OS and computer
  - **"Physical" addresses** used by the flash memory
- Perform writes out-of-place
  - Amortize block erasures over many write operations
- Wear-leveling
  - Writing the same "virtual" address repeatedly won't write to the same physical flash location repeatedly!



Chapter 8 – Operating Systems

Chapter 8 Topics

- OS responsibilities
- Mobile Operating Systems
  - **Similarities and differences?**
- Embedded/Real-Time Operating Systems
  - **Similarities and differences?**
- Interrupts
- Compilers versus Assemblers versus Interpreters
- Databases – **ACID**!

RTOS

- Real-time operating systems (RTOS) can provide **predictable timing** for high-priority tasks (while still running a mix of low-priority tasks)
- The difference with a general-purpose OS is an RTOS provides a **guarantee** of predictable timing
  - General-purpose OS usually meets its timing goals, but how often have you experienced a hiccup (momentary stutter) while playing a video or listening to music?

Interrupts

- What devices send **interrupts**?
  - Network card
    - Data received or data has been successfully sent
  - USB controller
    - Mouse moved, key/button pressed, etc..
  - Real-time clock, high precision event timer, etc…
  - The processor itself!
    - Divide by zero, page fault, invalid opcode, etc…
    - These are usually called ***exceptions***, but they work the same way as external interrupts
- Some of these interrupts represent errors, but others are perfectly normal and commonplace…

Interrupts

- What happens when the processor sees an interrupt?
  - Stop! Save the current running process
  - Lookup the interrupt number in an **interrupt descriptor table** (which is stored in memory from 0x0000 to 0x03FF)
    - Table contains pointer to a subroutine that processes the interrupt. This is the **interrupt service routine**
  - Run the interrupt service routine

Interrupt Service Routine

- **Interrupt service routine** - The specific subroutine that is executed whenever that interrupt number occurs
  - Tend to be small and fast (so we can get back to running the previous program quickly)
  - Examples
    - Copy packet from network card to main memory?
    - Notify OS that the mouse moved to the left 2 units?
    - Notify OS key “z” was pressed on the keyboard?
    - Notify OS of page fault for memory address 0x03813?

Database Software

- Most databases support *transactions* to assure that the database is always in a consistent state
  - Transaction is a group of related updates bundled together
- Transactions provides the following properties:
  - **Atomicity** - All related updates occur or no updates occur
  - **Consistency** - All updates conform to defined data constraints (i.e. data types, min/max legal values, etc…)
  - **Isolation** - No transaction can interfere with another transaction
  - **Durability** - Successful updates are written to durable media as soon as possible (i.e. RAM isn't safe if the system crashes or the power fails)
- These are the **ACID** properties of transaction management

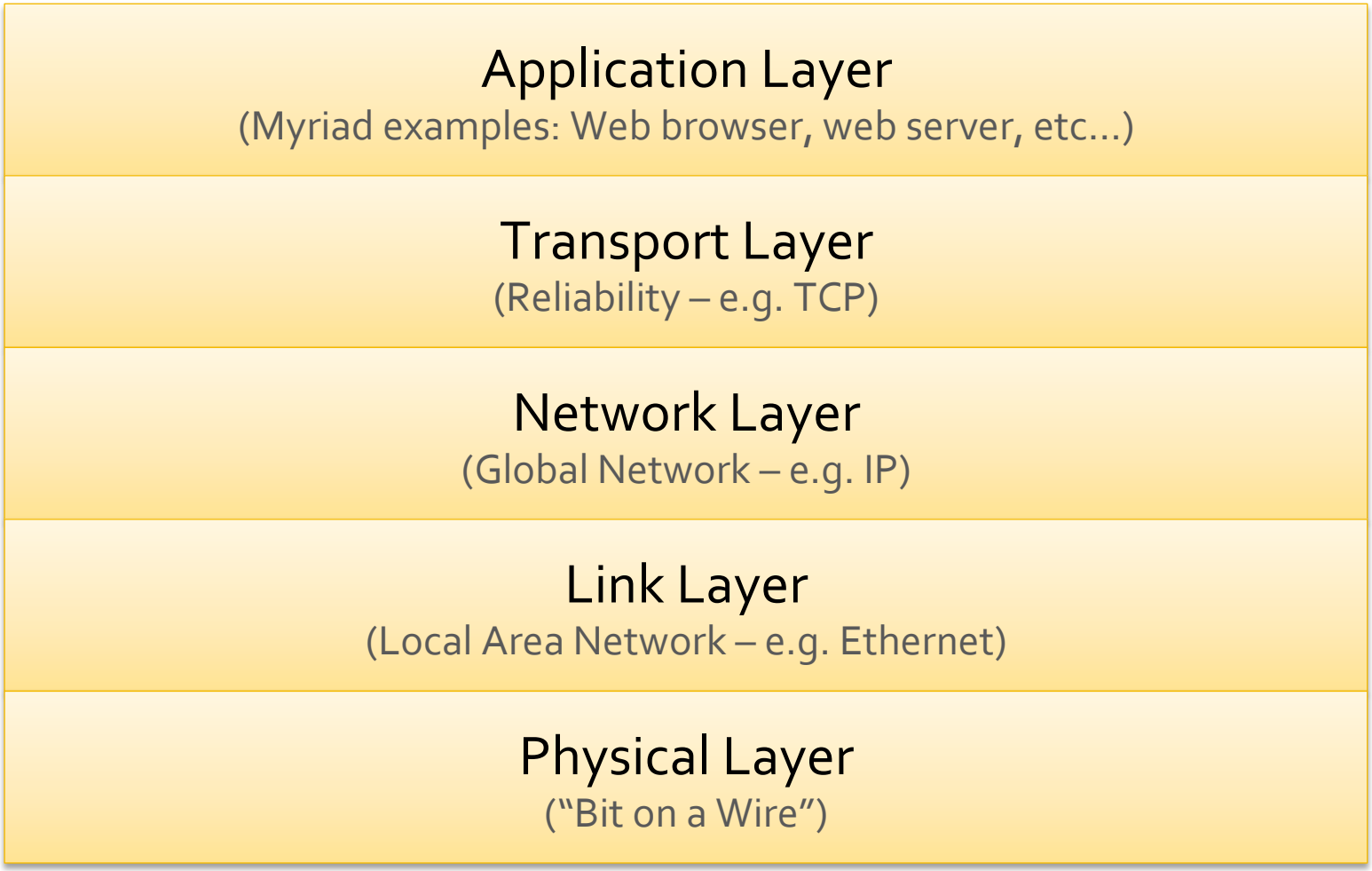
Chapter 9 – Processor Architectures

Chapter 9 Topics

- RISC versus CISC
- Instruction-level parallelism
  - **How is this different form running multiple applications on different cores of a CPU?**
  - **Why is this hard?**
  - Superscalar versus VLIW
- CPUs versus GPUs

Chapter 12 – Computer Networks

Network Model



Chapter 12 Topics

- TCP versus IP
  - **What features does IP provide?**
  - **What features does TCP provide?**
- Ethernet versus IP
  - **Where are source/destination MAC addresses used?**
  - **Where are source/destination IP addresses used?**
- Ethernet switch versus IP router
  - **What address does an Ethernet switch use to make a forwarding decision?**
  - **What address does an IP router use to make a forwarding decision?**