Ch. 1 - From Zero to One

1.2 The Art of Managing Complexity

Abstraction

- levels of abstraction for an electronic computing system
  - application software
  - operating systems
  - microarchitecture
  - logic
  - digital circuits
  - analog circuits
  - devices
  - physics
- hiding details when they are not important
- systems can be viewed from many different levels of abstraction
- electronic devices
  - terminals - connection points in electronic systems
- analog circuits
- digital circuits
- microarchitecture - links the logic and architecture levels of abstraction
- architecture - level of abstraction that describes a computer from the programmer's perspective
- operating system

Discipline

- the act of intentionally restricting your design choices so that you can work more productively at a higher level of abstraction
- using interchangeable parts is an application of discipline
- digital discipline sets context
  - allows for combination of components

The Three -Y's

- hierarchy
  - dividing a system into modules, then further subdividing each of these modules until the pieces are easy to understand

- modularity
  - states that the modules have well-defined functions and interfaces, so that they connect together easily without unanticipated side effects
- regularity
  - seeks uniformity among the modules
  - common modules are reused many times, reducing the number of distinct modules that must be designed

1.3 The Digital Abstraction

- Digital systems
  - represent information with discrete-valued variables
    - variables with a finite number of distinct values
- Babbage's machine (1870's) used 0 - 9 with 25 rows of gears, or 25 digit precision
- amount of information D in a discrete valued variable with N distinct states is measured in units of bits as D = log2N bits
- bit - Binary digIT
- boolean logic - George Boole (mid 1800's)
- digital abstraction allows designers to not be concerned with physical medium of digital signal generation

1.4 Number Systems

Decimal Numbers

- 0 - 9 (base 10)
- right to left column weights of 10^column (0 indexed)
- n-digit number represents one of 10^n possible numbers

Binary Numbers

- one of two values (0 or 1, true or false, etc)
- base 2
- an n-bit binary number represents one of 2^n possibilities

Hexadecimal Numbers

- group of four bits represents one of 2^4 = 16 possibilities
- base 16
- 0-9 A-F

Bytes, Nibbles, and All That Jazz

- byte - group of 8 bits
- nibble - group of four bits
- words - data chunk handled by microprocessor, dependent on architecture
- least significant bit - bit in 1's column
- most significant bit - bit at opposite end of lsb
- least and most significant bytes in words
- kilo - 2^10 bytes
- mega - 2^20
- giga - 2^30
- remember powers of 2

Binary addition

- like decimal addition with carries of 1 from column to column
- digital systems usually operate on a fixed number of digits
- overflow - for addition, if the result is too big to fit in the available digits

Signed Binary Numbers

Sign/Magnitude

- msb is sign bit and remainder is magnitude
- ordinary binary addition does not work for sign/magnitude numbers
- an n-bit sign/magnitude number spans the range –2^n–1 + 1, 2^n–1 – 1
- both +0 and –0 exist

Two's Complement Numbers

- identical to unsigned binary numbers except that the most significant bit position has a weight of –2^n–1 instead of 2^n–1
- single 0 representation and addition work properly for both positive and negative numbers
- 0 -> all 0's
- greatest positive value -> 0 and all 1's
- lowest negative value -> 1 and all 0's
- -1 -> all 1's
- to reverse sign (take two's complement)
  - invert all bits
  - add 1
- to subtract - take two's complement of second operand and then add

- 0 is considered positive because it's sign bit is 0
- n-bit two's complement numbers represent one of 2^n possible values
- overflow may occur
    - overflow occurs if the two numbers being added have the same sign bit and the result has the opposite sign bit

Comparison of Number Systems

- three most common are unsigned, two's complement, and sign/magnitude
- ranges
    - unsigned - 0, 2^n - 1
    - sign/magnitude - -2^(n-1) + 1, 2^(n-1) - 1
    - two's complement
- two's complement numbers are convenient because they represent both positive and negative integers and because ordinary addition works for all numbers
- unless stated otherwise, assume that all signed binary numbers use two's complement representation

1.5 Logic Gates

- simple digital circuits that take one or more binary inputs and produce a binary output
- relationship between input and output can be described with a truth table or boolean equation
- single input gates
    - NOT
    - buffer
- two input gates
    - AND
        - 00->0
        - 01->0
        - 10->0
        - 11->1
    - OR
        - 00->0
        - 01->1
        - 10->1
        - 11->1
    - XOR
        - 00->0
        - 01->1
        - 10->1

        - 11->0
    - NAND
        - 001
        - 011
        - 101
        - 110
    - NOR
        - 001
        - 010
        - 100
        - 110
    - XNOR
        - 001
        - 010
        - 100
        - 111

Multiple Input Gates

- n-input AND gate - true when all inputs are true
- n-input OR gate - true when at least 1 input is true
- 3-input NOR gate - output true only if 1 of the inputs is true

1.6 Beneath the Digital Abstraction

- a digital system uses discrete-valued variables but the variables are represented by continuous physical quantities

Supply Voltage

- lowest voltage in the system is 0 V, also called ground or GND
- highest voltage in the system comes from the power supply and is usually called VDD
- in 1970's and 1980's technology, VDD was generally 5V
- as chips have progressed to smaller transistors, VDD has dropped to 3.3 V, 2.5 V, 1.8 V, 1.5 V, 1.2 V
    - to save power and avoid overloading the transistors

Logic Levels

- used to map continuous variables onto a discrete binary variable
- first gate is driver -> second gate is receiver

- The driver produces a LOW (0) output in the range of 0 to VOL or a HIGH(1) output in the range of VOH to VDD
- if the receiver gets an input in the range of 0 to VIL, it will consider the input to be LOW
- if the receiver gets an input in the range of VIH to VDD, it will consider the input to be HIGH
- if the receiver's input should fall in the forbidden zone (VIL -> VIH) gate behavior is unpredictable
- output and input high and low logic levels

Noise Margins

- the amount of noise that could be added to a worst-case output such that the signal can still be interpreted as a valid input

DC Transfer Characteristics

- DC transfer characteristics of a gate describe the output voltage as a function of the input voltage when the input is changed slowly enough that the output can keep up
- called transfer characteristics because they describe the relationship between input and output voltages
- a reasonable place to choose the logic levels is where the slope of the transfer characteristic dV(Y) / dV(A) is –1 (two points are called the unity gain points)

Static Discipline

- requires that, given logically valid inputs, every circuit element will produce logically valid outputs
- trade freedom of using arbitrary analog circuit elements in return for the simplicity and robustness of digital circuits
- logic families - grouping of gates such that all gates in a logic family obey the static discipline when used with other gates in the family
- four major families
  - Transistor-Transistor Logic (TTL)
  - Complementary Metal- Oxide-Semiconductor Logic (CMOS)
  - Low Voltage TTL Logic (LVTTL)
  - Low Voltage CMOS Logic (LVCMOS)

1.7 CMOS Transistors

- transistors - electrically controlled switches that turn ON or OFF when a voltage or current is applied to a control terminal
- two main types of transistors
  - bipolar junction transistors

    - metal-oxide-semiconductor field effect transistors (MOSFETs or MOS transistors)
- 1958 - Jack Kilby at Texas Instruments built the first integrated circuit containing two transistors
- 1959 - Robert Noyce at Fairchild Semiconductor patented a method of interconnecting multiple transistors on a single silicon chip

Semiconductors

- MOS transistors are built from silicon
    - silicon (Si) has four electrons in its valence shell and forms bonds with four adjacent atoms, resulting in a crystalline lattice
    - becomes a better conductor when small amounts of impurities, called dopant atoms, are carefully added
- n-type dopant - the electron carries a negative charge
- hole
- p-type dopant
- semiconductor - name given to class of conductors when conductivity changes over many orders of magnitude depending on the concentration of dopants

Diodes

- diode - a junction between an p-type and n-type silicon
- cathode - n-type region
- anode - p-type region
- forward biased - voltage on anode is higher
- reverse biased - voltage on cathode is higher

Capacitors

- capacitor - consists of two conductors separated by an insulator
    - when a voltage V is applied to one of the conductors, the conductor accumulates electric charge Q and the other conductor accumulates the opposite charge –Q
- capacitance C of the capacitor is the ratio of charge to voltage: $C = Q/V$
    - proportional to the size of the conductors and inversely proportional to the distance between them

nMOS and pMOS Transistors

- MOSFET is a sandwich of several layers of conducting and insulating materials, built on thin flat wafers of silicon
    - created by sequence of steps in which dopants are implanted into the silicon, thin films of silicon dioxide and silicon are grown, and metal is deposited

  - between each step, the wafer is patterned so that the materials appear only where they are desired
- once complete, the wafer is cut into rectangles called chips or dice that contain thousands, millions, or even billions of transistors
- chip is then placed in a plastic or ceramic package with metal pins to connect it to a circuit board
- n-type transistors, called nMOS, have regions of n-type dopants adjacent to the gate called the source and the drain and are built on a p-type semiconductor substrate
- pMOS transistors are just the opposite, consisting of p-type source and drain regions in an n-type substrate
- MOSFET behaves as a voltage-controlled switch in which the gate voltage creates an electric field that turns ON or OFF a connection between the source and drain
- name field effect transistor comes from this principle of operation
- description of nMOS and pMOS functionality here

CMOS NOT Gate

Other CMOS Logic Gates

Transmission Gates

- nMOS transistors are good at passing 0 and pMOS transistors are good at passing 1
  - parallel combination of the two passes both values well
- such a circuit is called a transmission gate

Pseudo-nMOS Logic

- transistors in series are slower than transistors in parallel
- resistors in series have more resistance than resistors in parallel
- pMOS transistors are slower than nMOS transistors because holes cannot move around the silicon lattice as fast as electrons
- parallel nMOS transistors are fast
- series pMOS transistors are slow especially when many are in series
- pseudo-nMOS logic replaces the slow stack of pMOS transistors with a single weak pMOS transistor that is always ON
  - often called a weak pull-up
  - can be used to build fast NOR gates with many inputs
  - useful for certain memory and logic arrays
  - disadvantage is that a short circuit exists between VDD and GND
    - when the output is LOW or the weak pMOS and nMOS transistors are both ON
    - the short circuit draws continuous power
    - pseudo-nMOS logic must be used sparingly

1.8 Power Consumption

- amount of energy used per unit time
- very important
- digital systems draw both dynamic and static power
- dynamic power is the power used to charge capacitance as signals change between 0 and 1
- static power is the power used even when signals do not change and the system is idle

Ch.2 - Combination Logic Design

2.1 Introduction

- digital circuit - a network that processes discrete-valued variables
  - black box with
    - one or more discrete-valued input terminals
    - one or more discrete-valued output terminals
    - a functional specification describing the relationship between inputs and outputs
    - a timing specification describing the delay between inputs changing and outputs responding
  - composed of nodes and elements
- element - a circuit with inputs, outputs, and a specification
- node - a wire, whose voltage conveys a discrete-valued variable
  - input
  - output
  - internal
- combinational or sequential
- combinational circuit - outputs depend only on the current values of the inputs
  - combines the current input values to compute the output
- sequential circuit - outputs depend on both current and previous values of the inputs
  - depends on the input sequence
- a combinational circuit is memoryless
- a sequential circuit has memory
- combination composition rules
  - every circuit element is itself combinational
  - every node of the circuit is either designated as an input to the circuit or connects to exactly one output terminal of a circuit element
  - the circuit contains no cyclic paths
    - every path through the circuit visits each circuit node at most once

- certain circuits that disobey these rules are still combinational, so long as the outputs depend only on the current values of the inputs
    - making this determination is difficult
- the functional specification of a combinational circuit is usually expressed as a truth table or a Boolean equation

2.2 Boolean Equations

- boolean algebra is used to simplify boolean equations
- axioms
    - A1 - `B = 0 if B != 1`
    - A2 - `complement of 0 = 1`
    - A3 - `0 • 0 = 0`
    - A4 - `1 • 1 = 1`
    - A5 - `0 • 1 = 1 • 0 = 0`
- duals
    - A1' - `B = 1 if B != 0`
    - A2' - `complement of 1 = 0`
    - A3' - `1 + 1 = 1`
    - A4' - `0 + 0 = 0`
    - A5' - `1 + 0 = 0 + 1 = 1`
- names
    - 1 - binary field
    - 2 - not
    - 3 - and/or
    - 4 - and/or
    - 5 - and/or
- duality - if the symbols 0 and 1 and the operators • (AND) and + (OR) are interchanged the statement will still be correct

Axioms

Theorems of One Variable

- identity
    - B AND 1 = B
    - dual - B OR 0 = B
- null element theorem
    - B AND 0 = 0
    - dual - B OR 1 = 1
- 0 is null for AND

- 1 is null for OR
- idempotency
  - B AND B = B
  - dual - B OR B = B
- involution (complementing a variable twice results in original variable
  - complement of complement of B = B
- complements
  - B AND complement of B = 0
  - dual - B OR complement of B = 1

Theorems of Several Variables

- commutativity
  - T6 - B • C = C • B
  - T6' - B + C = C + B
- associativity
  - T7 - (B • C) • D = B • (C • D)
  - T7' - (B + C) + D = B + (C + D)
- distributivity
  - T8 - (B • C) + (B • D) = B • (C + D)
  - T8' - (B + C) • (B + D) = B + (C • D)
- covering
  - T9 - B • (B + C) = B
  - T9' - B + (B • C) = B
- combining
  - T10 - (B • C) + (B • C) = B
  - T10' - (B + C) • (B + C) = B
- consensus
  - T11 - (B • C) + (B • D) + (C • D) = B • C + B • D
  - T11' - (B + C) • (B + D) • (C + D) = (B + C) • (B + D)
- De Morgan's Theorem
  - T12 - conjugate of B0 • B1 • B2... = B0 + B1 + B2...
  - T12' - conjugate of B0 + B1 + B1... = B0 • B1 • B2...

- De Morgan's Theorem
    - complement of the product of all the terms is equal to the sum of the complement of each term
    - complement of the sum of all the terms is equal to the product of the complement of each term
- rules for bubble pushing
    - pushing bubbles backward (from the output) or forward (from the inputs) changes the body of the gate from AND to OR or vice versa
    - pushing a bubble from the output back to the inputs puts bubbles on all gate inputs
    - pushing bubbles on all gate inputs forward toward the output puts a bubble on the output

The Truth Behind it All

- proof for theorems with a finite number of variables
    - show that the theorem holds for all possible values of these variables
    - this method is called perfect induction
    - can be done with a truth table

Simplifying Equations

- an equation in sum-of-products form is minimized if it uses the fewest possible implicants
    - if there are several equations with the same number of implicants, the minimal one is the one with the fewest literals
- an implicant is called a prime implicant if it cannot be combined with any other implicants in the equation to form a new implicant with fewer literals
    - the implicants in a minimal equation must all be prime implicants
    - otherwise, they could be combined to reduce the number of literals
- expanding an implicant is sometimes useful in minimizing equation

2.4 From Logic to Gates

- schematic - diagram of a digital circuit showing the elements and the wires that connect them together
- programmable logic array (PLA) - inverters, AND gates, and OR gates are arrayed in a systematic fashion

2.5 Multilevel Combinational Logic

- two-level logic - logic in sum-of-products form that consists of literals connected to a level of AND gates connected to a level of OR gates

Hardware Reduction

- some logic functions require an enormous amount of hardware when built using two-level logic
    - XOR function of multiple variables
    - three-input XOR using the two-level techniques we have studied so far
        - can be built out of a cascade of two-input XORs
    - eight-input XOR would require 128 eight-input AND gates and one 128-input OR gate for a two-level sum-of-products implementation
    - better option is to use a tree of two-input XOR gates
- selecting the best multilevel implementation of a specific logic function is not a simple process
    - fewest gates
    - fastest
    - shortest design time
    - least cost
    - least power consumption

Bubble Pushing

- CMOS circuits prefer NANDs and NORs over ANDs and ORs
- reading the equation by inspection from a multilevel circuit with NANDs and NORs can be difficult
- bubble pushing is a helpful to redraw circuits so that the bubbles cancel out and the function can be more easily determined
- guidelines
    - begin at the output of the circuit and work toward the inputs
    - push any bubbles on the final output back toward the inputs so that you can read an equation in terms of the output (for example, Y) instead of the complement of the output
    - working backward, draw each gate in a form so that bubbles cancel
        - if the current gate has an input bubble, draw the preceding gate with an output bubble
        - if the current gate does not have an input bubble, draw the preceding gate without an output bubble
- bubbles in series cancel

X'S and Z'S, Oh My

- real circuits can also have illegal and floating values, represented symbolically by X and Z
- symbol X indicates that the circuit node has an unknown or illegal value
  - commonly happens if it is being driven to both 0 and 1 at the same time
- contention when a node is driven both HIGH and LOW is considered to be an error and must be avoided
- actual voltage on a node with contention may be somewhere between 0 and VDD, depending on the relative strengths of the gates driving HIGH and LOW
  - often, but not always, in the forbidden zone
- contention also can cause large amounts of power to flow between the fighting gates, resulting in the circuit getting hot and possibly damaged
- X values are also sometimes used by circuit simulators to indicate an uninitialized value
- designers also use the symbol X to indicate "don't care" values in truth tables
  - do not confuse
  - X indicates that the value of the variable in the truth table is unimportant (can be either 0 or 1) when it appears in a truth table
  - X indicates that the circuit node has an unknown or illegal value when it appears in a circuit

Floating Value: Z

- Z indicates that a node is being driven neither HIGH nor LOW
  - node is said to be floating, high impedance, or high Z
  - typical misconception is that a floating or undriven node is the same as a logic 0
  - floating node might be 0, might be 1, or might be at some voltage in between, depending on the history of the system
- does not always mean there is an error in the circuit, so long as some other circuit element does drive the node to a valid logic level when the value of the node is relevant to circuit operation
- common way to produce a floating node is to forget to connect a voltage to a circuit input, or to assume that an unconnected input is the same as an input with the value of 0
- tristate buffer has three possible output states
  - HIGH (1)
  - LOW (0)
  - floating (Z)
  - input A, output Y, and enable E
  - when the enable is TRUE acts as a simple buffer, transferring the input value to the output
  - when the enable is FALSE, the output is allowed to float (Z)

    - commonly used on busses that connect multiple chips
        - microprocessor
        - video controller
        - Ethernet controller
    - all need to communicate with the memory system in a personal computer
        - each chip can connect to a shared memory bus using tristate buffers
        - only one chip at a time is allowed to assert its enable signal to drive a value onto the bus
        - other chips must produce floating outputs so that they do not cause contention with the chip talking to the memory
        - any chip can read the information from the shared bus at any time
    - modern computers have higher speeds possible with point-to-point links
        - chips are connected to each other directly rather than over a shared bus

Karnaugh Maps

- Karnaugh maps (K-maps)
    - graphical method for simplifying Boolean equations
    - invented in 1953 by Maurice Karnaugh a telecommunications engineer at Bell Labs
    - work well for problems with up to four variables
    - give insight into manipulating Boolean equations
- logic minimization involves combining terms
    - two terms containing an implicant P and the true and complementary forms of some variable A are combined to eliminate A
- K-maps use grid to show elimination

Circular Thinking

Logic Minimization with K-Maps

- rules for finding a minimized equation from a K-map
    - use the fewest circles necessary to cover all the 1's
    - all the squares in each circle must contain 1's
    - each circle must span a rectangular block that is a power of 2 (i.e., 1, 2, or 4) squares in each direction
    - each circle should be as large as possible
    - a circle may wrap around the edges of the K-map
    - a 1 in a K-map may be circled multiple times if doing so allows fewer circles to be used

Don't Cares

- "don't care" entries for truth table inputs reduce the number of rows in the table when some vari-ables do not affect the output
    - indicated by the symbol X, which means that the entry can be either 0 or 1
- also appear in truth table outputs where the output value is unimportant or the corresponding input combination can never happen
    - can be treated as either 0's or 1's at the designer's discretion
- X's allow for even more logic minimization in K-maps
    - can be circled if they help cover the 1's with fewer or larger circles, but they do not have to be circled if they are not helpful

The Big Picture

- Boolean algebra and Karnaugh maps are two methods of logic simplification
- goal is to find a low-cost method of implementing a particular logic function
- computer programs called logic synthesizers produce simplified circuits from a description of the logic function
- for large problems, logic synthesizers are much more efficient than humans
- for small problems, a human with a bit of experience can find a good solution by inspection
- never used a Karnaugh map in real life to solve a practical problem
- insight gained from the principles underlying Karnaugh maps is valuable
    - also on job interviews

Combinational Building Blocks

- combinational logic is often grouped into larger building blocks to build more complex systems
- three building blocks
    - full adders
    - priority circuits
    - seven-segment display decoders
- two more commonly used building blocks
    - multiplexers
    - decoders

Multiplexers

- most commonly used combinational circuits
- choose an output from among several possible inputs based on the value of a select signal
- sometimes affectionately called a mux

2:1 Multiplexer

- multiplexer chooses between the two data inputs based on the select
  - input S is also called a control signal because it controls what the multiplexer does
- 2:1 multiplexer can be built from sum-of-products logic
  - Boolean equation for the multiplexer may be derived with a Karnaugh map or read off by inspection
- multiplexers can be built from tristate buffers
  - tristate enables are arranged such that, at all times, exactly one tristate buffer is active

Wider Multiplexers

- 4:1 multiplexer has four data inputs and one output
  - two select signals are needed to choose among the four data inputs
- can be built using sum-of-products logic, tristates, or multiple 2:1 multiplexers
- product terms enabling the tristates can be formed using AND gates and inverters
- can also be formed using a decoder
- wider multiplexers, such as 8:1 and 16:1 multiplexers, can be built by expanding the methods
- N:1 multiplexer needs log2N select lines

Multiplexer Logic

- multiplexers can be used as lookup tables to perform logic functions
- 4:1 multiplexer used to implement a two-input AND gate
  - inputs, A and B, serve as select lines
  - multiplexer data inputs are connected to 0 or 1 according to the corresponding row of the truth table
- a 2N-input multiplexer can be programmed to perform any N-input logic function by applying 0's and 1's to the appropriate data inputs
  - by changing the data inputs, the multiplexer can be reprogrammed to perform a different function
- can cut the multiplexer size in half using only a 2N–1-input multiplexer to perform any N-input logic function
  - provide one of the literals, as well as 0's and 1's, to the multiplexer data inputs

Decoders

- N inputs and 2N outputs
  - asserts exactly one of its outputs depending on the input combination
- 2:4 decoder
  - when A1:0 = 00, Y0 is 1

    - when A1:0 = 01, Y1 is 1
    - etc
    - outputs are called one-hot, because exactly one is "hot" (HIGH) at a given time

Decoder Logic

- decoders can be combined with OR gates to build logic functions
    - two-input XNOR function using a 2:4 decoder and a single OR gate
    - each output of a decoder represents a single minterm, the function is built as the OR of all the minterms in the function
- when using decoders to build logic, it is easiest to express functions as a truth table or in canonical sum-of-products form
- N-input function with M 1's in the truth table can be built with an N:2N decoder and an M-input OR gate attached to all of the minterms containing 1's in the truth table
    - used for Read Only Memories (ROMs)

Timing

- one of the most challenging issues in circuit design is timing
    - speed of circuit
- delay between an input change and the subsequent output change
- timing diagram
    - portrays the transient response of the buffer circuit when an input changes
    - transition from LOW to HIGH is called the rising edge
    - transition from HIGH to LOW (not shown in the figure) is called the falling edge
- measure delay from the 50% point of the input signal, A, to the 50% point of the output signal, Y
- 50% point is the point at which the signal is half-way (50%) between its LOW and HIGH values as it transitions

Propagation and Contamination Delay

- combinational logic is characterized by its propagation delay and contamination delay.

- propagation delay tpd is the maximum time from when an input changes until the output or outputs reach their final value

- contamination delay tcd is the minimum time from when an input changes until any output starts to change its value

- underlying causes of delay in circuits include the time required to charge the capacitance in a circuit and the speed of light

- tpd and tcd may be different for many reasons
  - different rising and falling delays
  - multiple inputs and outputs, some of which are faster than others
  - circuits slowing down when hot and speeding up when cold
- manufacturers normally supply data sheets specifying these delays for each gate
- propagation and contamination delays are also determined by the path a signal takes from input to output
- critical path limits the speed at which the circuit operates
- propagation delay of a combinational circuit is the sum of the propagation delays through each element on the critical path
- contamination delay is the sum of the contamination delays through each element on the short path
- equations here

Glitches

- possible that a single input transition can cause multiple output transitions
  - glitches or hazards
- usually don't cause problems
  - important to realize that they exist and recognize them when looking at timing diagrams
- glitches are not a problem when waiting for the propagation delay to elapse before we depend on the output because the output eventually settles to the right answer
- glitch can occur when a change in a single variable crosses the boundary between two prime implicants in a K-map
  - can eliminate the glitch by adding redundant implicants to the K-map to cover these boundaries
  - comes at the cost of extra hardware
  - simultaneous transitions on multiple inputs can also cause glitches
  - glitches cannot be fixed by adding hardware
  - vast majority of interesting systems have simultaneous (or near-simultaneous) transitions on multiple inputs, glitches are a fact of life in most circuits
- point of discussing glitches is not to eliminate them but to be aware that they exist
  - especially important when looking at timing diagrams on a simulator or oscilloscope

Ch. 3 - Sequential Logic Design

3.2 Latches and Flip-Flops

- bistable element - element with two stable states; fundamental building block of memory
- connecting pair of inverters in loop leads to simple bistable element
  - additional third state with both outputs halfway between
  - called a metastable state
- an element with N stable states conveys log2N bits of information
  - bistable element stores one bit
  - state of the cross-coupled inverters is contained in one binary state variable, Q
  - value of Q tells everything about the past that is necessary to explain the future behavior
    - if Q = 0, it will remain 0 forever = if Q = 1, it will remain 1 forever
  - circuit has another node, Q hat
  - Q hat does not contain any additional information because if Q hat is known, Q is also known

SR Latch

- composed of two cross-coupled NOR gates; one of the simplest sequential circuits
  - state can be controlled through two inputs, S and R, which can set and reset the output Q
- steps through input possibilities here

D Latch

- SR latch behaves strangely when both S and R are simultaneously asserted
  - conflates what and when
- D latch solves these problems
  - data input, D, controls what the next state should be
  - clock input, CLK, controls when the state should change clock controls when data flows through the latch
  - when CLK = 1, the latch is transparent
    - data at D flows through to Q as if the latch were just a buffer
  - when CLK = 0, the latch is opaque
    - blocks the new data from flowing through to Q
    - Q retains the old value
- D latch is sometimes called a transparent latch or a level-sensitive latch
- D latch updates its state continuously while CLK = 1

D Flip-Flop

- can be built from two back-to-back D latches controlled by complementary clocks
  - master and slave
- IMPORTANT!!!! D flip-flop copies D to Q on the rising edge of the clock, and remembers its state at all other times
  - rising edge of the clock is often called the clock edge
  - D input specifies what the new state will be
  - clock edge indicates when the state should be updated
- also known as
  - master-slave flip-flop
  - edge-triggered flip-flop
  - positive edge-triggered flip-flop

Register

- N-bit register is a bank of N flip-flops that share a common CLK input
  - all bits of the register are updated at the same time
- key building block of most sequential circuits

Enabled Flip-Flop

- adds another input called EN or ENABLE to deter- mine whether data is loaded on the clock edge
  - when EN is TRUE, enabled flip-flop behaves like an ordinary D flip-flop
  - when EN is FALSE, enabled flip-flop ignores the clock and retains its state

Resettable Flip-Flop

- adds another input called RESET
  - when RESET is FALSE, resettable flip-flop behaves like an ordinary D flip-flop
  - when RESET is TRUE, the resettable flip-flop ignores D and resets the output to 0
- useful when need to force a known state (i.e., 0) into all the flip-flops in a system when turned on
- synchronously or asynchronously resettable
  - synchronously resettable flip-flops reset themselves only on the rising edge of CLK
  - asynchronously resettable flip-flops reset themselves as soon as RESET becomes TRUE, independent of CLK

Transistor-Level Latch and Flip-Flop Designs

Putting it all Together

- latches and flip-flops are fundamental building blocks of sequential circuits
    - D latch is level-sensitive
    - D flip-flop is edge-triggered
    - D latch is transparent when CLK = 1, allowing the input D to flow through to the output Q
    - D flip-flop copies D to Q on the rising edge of CLK
    - at all other times latches and flip-flops retain their old state
    - a register is a bank of several D flip-flops that share a common CLK signal

3.3. Synchronous Logic Design

- sequential circuits include all circuits that are not combinational
    - output cannot be determined simply by looking at the current input
- many complex formulations

Some Problematic Circuits

Synchronous Sequential Circuits

- cyclic paths - outputs are fed directly back to inputs
    - none in combinational logic
    - cyclic paths can lead to undesirable races or unstable behavior
- break cycles by inserting registers
    - contain state of system at clock edge
    - synchronized to clock
- sequential circuit has a finite set of discrete states {S0, S1,..., Sk−1}
- synchronous sequential circuit has a clock input
    - rising edges indicate a sequence of times at which state transitions occur
- current state and next state distinguish the state of the system at the present from the state to which it will enter on the next clock edge
- rules of synchronous sequential circuit composition
    - a circuit is a synchronous sequential circuit if it consists of interconnected circuit elements such that
        - every circuit element is either a register or a combinational circuit
        - at least one circuit element is a register
        - all registers receive the same clock signal
        - every cyclic path contains at least one register

- asynchronous otherwise
- flip-flop is the simplest synchronous sequential circuit
- Two other common types of synchronous sequential circuits
    - finite state machines
    - pipelines

Synchronous and Asynchronous Circuits

- async more general but sync easier to design and use
    - virtually all digital systems are essentially synchronous
- may need async circuits for communication between systems with different clocks

3.4 Finite State Machines

- a circuit with k registers that can be in one of a finite number (2k) of unique states
    - has M inputs, N outputs, and k bits of state
    - receives a clock and, optionally, a reset signal
    - consists of two blocks of combinational logic
        - next state logic
        - output logic
        - a register that stores the state
    - on each clock edge FSM advances to the next state which was computed based on the current state and inputs
- two general classes (characterized by functional specifications)
    - Moore machines
        - outputs depend only on the current state of the machine
    - Mealy machines
        - outputs depend on both the current state and the current inputs
- provide a systematic way to design synchronous sequential circuits given a functional specification

FSM Design Example

- start with state transition diagram
    - possible states of system
    - transitions between states
- generate state transition table from transition diagram
    - indicates for each state and input what the next state should be
    - must be assigned binary encodings to build a real circuit
- generate output table
    - indicates what the output should be for each state

State Encodings

- how to determine the encoding that produces the circuit with the fewest logic gates or the shortest propagation delay?
  - no simple answer
  - must try all possibilities which is infeasible when number of states is large
- binary encoding vs. one-hot encoding
  - one-hot encoding - separate bit of state is used for each state

Moore and Mealy Machines

- Mealy machines are similar to Moore machines
  - the outputs can depend on inputs as well as the current state
  - in state transition diagrams for Mealy machines the outputs are labeled on the arcs instead of in the circles
  - block of combinational logic that computes the outputs uses the current state and inputs

Factoring State Machines

- designing complex FSMs is often easier if they can be broken down into multiple interacting simpler state machines such that the output of some machines is the input of others

Deriving an FSM from a Schematic

- examine circuit, stating inputs, outputs, and state bits
- write next state and output equations
- create next state and output tables
- reduce the next state table to eliminate unreachable states
- assign each valid state bit combination a name
- rewrite next state and output tables with state names
- draw state transition diagram
- state in words what the FSM does
  - be careful to succinctly describe the overall purpose and function of the FSM
  - do not simply restate each transition of the state transition diagram

FSM Review

- powerful way to systematically design sequential circuits from a written specification
- use the following procedure to design an FSM:
  - identify the inputs and outputs
  - sketch a state transition diagram

    - for a Moore machine
        - write a state transition table
        - write an output table
    - for a Mealy machine
        - write a combined state transition and output table
    - select state encodings
        - selection affects the hardware design
    - write boolean equations for the next state and output logic
    - sketch the circuit schematic

3.5 Timing of Sequential Logic

- flip-flop copies the input D to the output Q on the rising edge of the clock
- sampling D on the clock edge
- if D is stable at either 0 or 1 when the clock rises behavior is clearly defined
- what happens if D is changing at the same time the clock rises?
- sequential element has an aperture time around the clock edge
    - input must be stable for the flip-flop to produce a well-defined output
- aperture of a sequential element is defined
    - setup time - before clock edge
    - hold time - before and after the clock edge
- dynamic discipline limited to using signals that change outside the aperture time
- can think of time in discrete units called clock cycles
- a signal may glitch and oscillate wildly for some bounded amount of time
    - concerned only with its final value at the end of the clock cycle after it has settled to a stable value
    - can simply write A[n], the value of signal A at the end of the nth clock cycle, where n is an integer
    - (rather than A(t), the value of A at some instant t, where t is any real number)
- clock period has to be long enough for all signals to settle
    - sets a limit on the speed of the system
    - clock skew - in real systems the clock does not reach all flip-flops at precisely the same time
    - this variation in time increases the necessary clock period
- may be impossible to satisfy the dynamic discipline

The Dynamic Discipline

- states that the inputs of a synchronous sequential circuit must be stable during the setup and hold aperture time around the clock edge
    - guarantees that the flip-flops sample signals while they are not changing

    - can treat signals as discrete in time as well as in logic levels because only concerned with the final values of the inputs at the time they are sampled

Setup Time Constraint

Hold Time Constraint

Putting it all Together

- sequential circuits have setup and hold time constraints that dictate the maximum and minimum delays of the combinational logic between flip-flops
- modern flip-flops are usually designed so that the minimum delay through the combinational logic is 0
    - flip-flops can be placed back-to-back
- maximum delay constraint limits the number of consecutive gates on the critical path of a high-speed circuit
    - a high clock frequency means a short clock period

Clock Skew

- variation in clock edges (time at which clock reaches registers, etc)
    - wires from the clock source to different registers may be of different lengths
    - noise
    - clock gating
    - use of gated and ungated clocks
- effectively increases both the setup time and the hold time
- adds to the sequencing overhead, reducing the time available for useful work in the combinational logic
- increases the required minimum delay through the combinational logic
- designers must not permit too much clock skew
    - sometimes flip-flops are intentionally designed to be particularly slow to prevent hold time problems even when the clock skew is substantial

Metastability

- not always possible to guarantee that the input to a sequential circuit is stable during the aperture time especially when the input arrives from the external world

Metastable State

- when a flip-flop samples an input that is changing during its aperture the output Q may momentarily take on a voltage between 0 and VDD that is in the forbidden zone
    - eventually the flip-flop will resolve the output to a stable state of either 0 or 1
    - the resolution time required to reach the stable state is unbounded

- every bistable device has a metastable state between the two stable states

Resolution Time

- if the input to a bistable device such as a flip-flop changes during the aperture time the output may take on a metastable value for some time before resolving to a stable 0 or 1
- amount of time required to resolve is unbounded
    - for any finite time, t, the probability that the flip-flop is still metastable is nonzero
    - probability drops off exponentially as t increases
- wait long enough can expect with exceedingly high probability that the flip-flop will reach a valid logic level

Synchronizers

- goal of a digital system designer should be to ensure that, given asynchronous inputs, the probability of encountering a metastable voltage is sufficiently small
    - to guarantee good logic levels all asynchronous inputs should be passed through synchronizers
- a device that receives an asynchronous input D and a clock CLK
    - produces an output Q within a bounded amount of time
    - output has a valid logic level with extremely high probability
    - if D is stable during the aperture Q should take on the same value as D
    - if D changes during the aperture Q may take on either a HIGH or LOW value but must not be metastable

Derivation of Resolution Time

3.6 Parallelism

- speed of a system is characterized by the latency and throughput of information moving through it
- token - group of inputs that are processed to produce a group of outputs
- latency (of a system) - time required for one token to pass through the system from start to end
- throughput - number of tokens that can be produced per unit time
- parallelism - imrpovement of throughput by processing several tokens at the same time
- spatial parallelism
    - multiple copies of the hardware are provided so that multiple tasks can be done at the same time
- temporal parallelism
    - task is broken into stages like an assembly line
    - multiple tasks can be spread across the stages

    - each task must pass through all stages
    - a different task will be in each stage at any given time so multiple tasks can overlap
- temporal parallelism is commonly called pipelining
- pipelining (temporal parallelism) is particularly attractive because it speeds up a circuit without duplicating the hardware
    - registers are placed between blocks of combinational logic to divide the logic into shorter stages that can run with a faster clock
    - registers prevent a token in one pipeline stage from catching up with and corrupting the token in the next stage

Ch. 4 - Hardware Description Languages

- NOTE: see code examples with explanations from repo and book
- 1990s designers became more productive by working at a higher level of abstraction by specifying just the logical function and allowing a computer-aided design (CAD) tool to produce the optimized gates
    - specifications are generally given in a hardware description language (HDL)
    - two leading hardware description languages are SystemVerilog and VHDL

Modules

- block of hardware with inputs and outputs
    - behavioral models describe what a module does
    - structural models describe how a module is built from simpler pieces

SystemVerilog

```
module sillyfunction(input logic a, b, c, output logic y);
    assign y = ~a & ~b & ~c | a & ~b & ~c | a & ~b & c;
endmodule
```

- begins with the module name and a listing of the inputs and outputs
- assign statement describes combinational logic
- ~ indicates NOT, & indicates AND, and I indicates OR.\
- logic signals such as the inputs and outputs are Boolean variables (0 or 1)
    - (may also have floating and undefined values)
- logic type was introduced in SystemVerilog
    - supersedes the reg type (source of confusion in Verilog)
- logic should be used everywhere except on signals with multiple drivers
- signals with multiple drivers are called nets

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity sillyfunction is
    port(a, b, c: in STD_LOGIC; y: out STD_LOGIC);
end;

architecture synth of sillyfunction is
begin
    y <= (not a and not b and not c) or (a and not b and not c) or (a and not b an
end;
```

- three parts
  - library use clause
  - entity declaration
  - architecture body
- entity declaration lists the module name and its inputs and outputs
- architecture body defines what the module does
- signals (inputs and outputs) must have a type declaration
- digital signals should be declared to be STD_LOGIC type
- STD_LOGIC signals can have a value of '0' or '1', as well as floating and undefined values
- STD_LOGIC type is defined in the IEEE.STD_LOGIC_1164 library
- lacks a good default order of operations between AND and OR
  - boolean equations should be parenthesized

Language Origins

- VHDL is more verbose and cumbersome
- both languages are fully capable of describing any hardware system
- both have quirks
- use is the one that is already being used at site or the one that customers demand
- most CAD tools allow mixture

Simulation and Synthesis

- logic synthesis transforms HDL code into a netlist describing the hardware
- logic synthesizer might perform optimizations to reduce the amount of hardware required
- netlist may be a text file or drawn as a schematic to help visualize the circuit
- circuit descriptions in HDL resemble code in a programming language
  - remember that the code is intended to represent hardware
  - not all commands can be synthesized into hardware
  - do not think of as a program

    - think of as a description of hardware

4.2 Combinational Logic

- synchronous sequential circuits consist of combinational logic and registers
- outputs of combinational logic depend only on the current inputs

Bitwise Operators

- act on single-bit signals or on multi-bit busses

Logic Gates

Comments and White Space

- not necessary but useful for comprehension
- be consistent with naming conventions

Reduction Operators

- imply a multiple-input gate acting on a single bus
- eight-input AND gate
    - analogous reduction operators exist for OR, XOR, NAND, NOR, and XNOR gates
- multiple-input XOR performs parity, returning TRUE if an odd number of inputs are TRUE

Eight-input AND

Conditional Assignment

- conditional assignments select the output from among alternatives based on an input called the condition

Multiplexer

Internal Variables

- neither inputs nor outputs but are used only internal to the module
    - similar to local variables in programming language
- assignment statements take place concurrently

Full Adder

Precedence

- precedence tables here

Numbers

- can be specified in binary, octal, decimal, or hexadecimal
- size, i.e., the number of bits, may optionally be given, and leading zeros are inserted to reach this size
- underscores in numbers are ignored and can be helpful in breaking long numbers into more readable chunks

Z's and X's

- use z to indicate a floating value
    - useful for describing a tristate buffer whose output floats when the enable is 0
- use x to indicate an invalid logic level
    - if a bus is simultaneously driven to 0 and 1 by two enabled tristate buffers (or other gates), the result is x, indicating contention
- at the start of simulation state nodes such as flip-flop outputs are initialized to an unknown state (x in SystemVerilog and u in VHDL)
    - helpful to track errors caused by forgetting to reset a flip-flop before its output is used

Tristate Buffer

Truth Tables with Undefined and Floating Inputs

Bit Swizzling

- collective of group of operations that operate on a subset of a bus or to concatenate signals to form busses

Bit Swizzling

Delays

- HDL statements may be associated with delays specified in arbitrary units
    - helpful during simulation to predict how fast a circuit will work
    - for debugging purposes to understand cause and effect

Logic Gates with Delays

4.3 Structural Modeling

- describing a module in terms of how it is composed of simpler modules
    - behavioral modeling - describing a module in terms of the relationships between inputs and outputs

- instance - copy of same module
    - distinguished by different names

Structural Model fo 4:1 multiplexer

Structural Model fo 2:1 multiplexer

- complex systems are designed hierarchically
    - overall system is described structurally by instantiating its major components
    - each component is described structurally from its building blocks recursively until the pieces are simple enough to describe behaviorally
    - good style to avoid or minimize mixing structural and behavioral descriptions within a single module

Accessing Parts of Busses

4.4 Sequential Logic

- HDL synthesizers recognize certain idioms and turn them into specific sequential circuits

Registers

- vast majority of modern commercial systems are built with registers using positive edge-triggered D flip-flops

Register

Resettable Register

- when simulation begins or power is first applied to a circuit the output of a flop or register is unknown
    - indicated with x in SystemVerilog and u in VHDL
- good practice to use resettable registers so that on powerup you can put your system in a known state
    - reset may be either asynchronous or synchronous
    - asynchronous reset occurs immediately
    - synchronous reset clears the output only on the next rising edge of the clock

Resettable Register

Enabled Registers

- enabled registers respond to the clock only when the enable is asserted

Resettable Enabled Register

Multiple Registers

- a single always/process statement can be used to describe multiple pieces of hardware

Synchronizer

Latches

- D latch is transparent when the clock is HIGH, allowing data to flow from input to output
    - latch becomes opaque when the clock is LOW, retaining its old state
- not all synthesis tools support latches well
    - use edge-triggered flip-flops instead unless tool does support latches and you have a good
    - ensure HDL does not imply any unintended latches
        - many synthesis tools warn you when a latch is created

4.5 More Combinational Logic

D Latch

Inverter Using always/process

- always/process statements are used to describe sequential circuits because they remember the old state when no new state is prescribed
    - always/process statements can also be used to describe combinational logic behaviorally if the sensitivity list is written to respond to changes in all of the inputs and the body prescribes the output value for every possible input combination
- HDLs support blocking and nonblocking assignments in an always/ process statement
    - group of blocking assignments are evaluated in the order in which they appear in the code
    - group of nonblocking assignments are evaluated concurrently
        - all of the statements are evaluated before any of the signals on the left hand sides are updated

Full Adder Using always/process

Case Statements

- TODO: more here

3:8 Decoder

If Statements

Priority Circuit

Truth Tables with Don't Cares

Priority Circuit using don't cares

Blocking and Nonblocking Assignments

Blocking and Nonblocking Assignments Guidelines

Combinational Logic

Full Adder Using Nonblocking Assignments

Sequential Logic

Bad Synchronizer with Blocking Assignments

4.6 Finite State Machines

- consists of a state register and two blocks of combinational logic to compute the next state and the output given the current state and the input
- HDL descriptions of state machines are correspondingly divided into three parts to model the state register, the next state logic, and the output logic

Divide-by-3 Finite State Machine

Pattern Recognizer Moore FSM

Pattern Recognizer Mealy FSM

4.7 Data Types

- TODO: more here

4.8 Parameterized Modules

- TODO: more here

4.9 Testbenches

- TODO: more here

Ch.5 - Digital Building Blocks

5.2 Arithmetic Circuits

- building blocks of computers
  - addition
  - subtraction
  - comparisons
  - shifts
  - multiplication
  - division

Half Adder

- 1-bit half adder
  - two inputs, A and B,
  - two outputs, S and Cout
  - S is the sum of A and B
  - if A and B are both 1, S is 2
  - 2 cannot be represented with a single binary digit
    - indicated with a carry out Cout in the next column
  - can be built from an XOR gate and an AND gate.
- multi-bit adder
  - Cout is added or carried in to the next most significant bit
- half adder lacks a Cin input to accept Cout of the previous column
- full adder solves this problem

Full Adder

- accepts the carry in Cin

```
Cin A B | Cout S
0   0 0 | 0    0
0   0 1 | 0    1
0   1 0 | 0    1
0   1 1 | 1    0
1   0 0 | 0    1
```

```
1   0 1 | 1    0
1   1 0 | 1    0
1   1 1 | 1    1
```

Carry Propagate Adder

- N-bit adder sums two N-bit inputs, A and B, and a carry in Cin to produce an N-bit result S and a carry out Cout
    - commonly called a carry propagate adder (CPA) because the carry out of one bit propagates into the next bit
    - three common CPA implementations
        - ripple-carry adders
        - carry-lookahead adders
        - prefix adders

Ripple-Carry Adder

- simplest way to build an N-bit carry propagate adder
    - chain together N full adders
    - Cout of one stage acts as the Cin of the next stage
    - full adder module is reused many times to form a larger system
    - disadvantage of being slow when N is large
    - delay of the adder grows directly with the number of bits

Carry-Lookahead Adder

- solves problem of ripple-carry adder slowness by dividing the adder into blocks and providing circuitry to quickly determine the carry out of a block as soon as the carry in is known
    - looks ahead across the blocks rather than waiting to ripple through all the full adders inside a block
- use generate (G) and propagate (P) signals that describe how a column or block determines the carry out
    - ith column of an adder is said to generate a carry if it produces a carry out independent of the carry in
    - ith column of an adder is guaranteed to generate a carry Ci if Ai and Bi are both 1
    - Gi, the generate signal for column i, is calculated as Gi = AiBi
    - column is said to propagate a carry if it produces a carry out whenever there is a carry in

Prefix adders

- extend the generate and propagate logic of the carry-lookahead adder to perform addition faster
    - first compute G and P for pairs of columns
    - then for blocks of 4
    - then for blocks of 8
    - then 16 etc until the generate signal for every column is known
    - sums are computed from these generate signals
- strategy of a prefix adder is to compute the carry in Ci−1 for each column i as quickly as possible then compute the sum

Putting It All Together

- half adder
- full adder
- three types of carry propagate adders
    - ripple-carry
    - carry-lookahead
    - prefix adders
- faster adders require more hardware and therefore are more expensive and power-hungry
- trade-offs must be considered when choosing an appropriate adder for a design
- HDLs provide the + operation to specify a CPA
- modern synthesis tools select among many possible implementations
    - choose the cheapest (smallest) design that meets the speed requirements

Adder

Subtraction

- flip the sign of the second number then add
    - for two's complement number - invert the bits and add 1

Subtractor

Comparators

- determines whether two binary numbers are equal or if one is greater or less than the other
    - takes as input two N-bit binary numbers A and B
- equality comparator
    - produces a single output indicating whether A is equal to B (A == B)
    - simpler

    - 4-bit equality comparator
        - checks to determine whether the corresponding bits in each column of A and B are equal using XNOR gates
        - numbers are equal if all of the columns are equal
- magnitude comparator
    - produces one or more outputs indicating the relative values of A and B
    - usually done by computing A – B and looking at the sign (most significant bit) of the result
        - if result is negative (i.e., the sign bit is 1), then A is less than B
        - otherwise A is greater than or equal to B

Comparators

ALU

- combines a variety of mathematical and logical operations into a single unit
    - addition
    - subtraction
    - magnitude comparison
    - AND
    - OR
- forms the heart of most computer system

```
000 A AND B
001 A OR B
010 A+B
011 not used
100 A AND conjugate B
101 A OR conjugate B
110 A−B
111 SLT
```

- set if less than (SLT) operation is used for magnitude comparison
    - when A<B, Y=1
    - otherwise, Y=0
    - (e.g. Y is set to 1 if A is less than B)
- some produce extra outputs (flags) that indicate information about the ALU output
    - an overflow flag indicates that the result of the adder overflowed
    - a zero flag indicates that the ALU output is 0

Shifters and Rotators

- move bits and multiply or divide by powers of 2
    - shifter shifts a binary number left or right by a specified number of positions
        - logical shifter
            - shifts the number to the left (LSL) or right (LSR) and fills empty spots with 0's
        - arithmetic shifter
            - same as a logical shifter, but on right shifts fills the most significant bits with a copy of the old most significant bit (msb)
            - useful for multiplying and dividing signed numbers
    - rotator rotates number in circle such that empty spots are filled with bits shifted off the other end
- an N-bit shifter can be built from N N:1 multiplexers
- left shift is a special case of multiplication
    - shift by N bits multiplies the number by 2^N
- arithmetic right shift is a special case of division
    - shift by N bits divides the number by 2^N

Multiplication

- partial products are formed by multiplying a single digit of the multiplier with the entire multiplicand
    - shifted partial products are summed to form the result
- an NxN multiplier multiplies two N-bit numbers and produces a 2N-bit result
    - partial products in binary multiplication are either the multiplicand or all 0's
- multiplication of 1-bit binary numbers is equivalent to the AND operation
    - AND gates are used to form the partial products

Multiplier

Division

- can be performed using the following algorithm for N-bit unsigned numbers in the range [0, 2N−1]

```
R'=0
for i = N−1 to 0
    R={R' << 1, Ai}
    D=R−B
    if D < 0 then Qi = 0, R' = R    // R < B
    else Qi = 1, R' = D             // R ≥ B
R=R'
```

- partial remainder R is initialized to 0

- most significant bit of the dividend A then becomes the least significant bit of R
- divisor B is repeatedly subtracted from this partial remainder to determine whether it fits
- if difference D is negative (i.e., the sign bit of D is 1), then the quotient bit Qi is 0 and the difference is discarded
- otherwise, Qi is 1, and the partial remainder is updated to be the difference
- partial remainder is then doubled (left-shifted by one column)
- the next most significant bit of A becomes the least significant bit of R
- repeat
- result satisfies A/B = Q + R/B

Further Reading

- Digital Arithmetic - Ercegovac and Lang
- CMOS VLSI Design - Weste and Harris

5.3 Number Systems

- fixed-point numbers are analogous to decimals
  - some of the bits represent the integer part
  - remainder represent the fraction
- floating-point numbers are analogous to scientific notation with a mantissa and an exponent

Fixed-Point Number Systems

- implied binary point between the integer and fraction bits
  - analogous to the decimal point between the integer and fraction digits of an ordinary decimal number
- signed fixed-point numbers can use either two's complement or sign/magnitude notation
  - in sign/magnitude form the most significant bit is used to indicate the sign
  - in two's complement is formed by inverting the bits of the absolute value and adding a 1 to the least significant (rightmost) bit
- just a collection of bits
  - no way of knowing the existence of the binary point except through agreement for interpretation

Floating-Point Number Systems

- analogous to scientific notation
- circumvent the limitation of having a constant number of integer and fractional bits
  - permits representation of very large and very small numbers
  - sign
  - mantissa (M)

    - base (B)
    - exponent (E)
- 32 bits are used to represent 1 sign bit, 8 exponent bits, and 23 mantissa bits
- binary floating-point
    - first bit of the mantissa (to the left of the binary point) is always 1
        - doesn't need to be stored
        - called the implicit leading one
        - only the fraction bits are stored

Special Cases: 0, ±∞, and NaN

- IEEE floating-point standard has special cases
    - zero
    - infinity
    - illegal results For example, representing the
- zero is problematic in floating-point notation because of the implicit leading one
- NaN is used for numbers that don't exist

```
Number      Sign        Exponent        Fraction
0           X           00000000        00000000000000000000000
∞           0           11111111        00000000000000000000000
−∞          1           11111111        00000000000000000000000
NaN         X           11111111        Non−zero
```

Single- and Double-Precision Formats

- single-precision - 32 bits
- double-precision (doubles) - 64 bits

```
Format  Total Bits  Sign Bits   Exponent Bits   Fraction Bits
single  32          1           8               23
double  64          1           11              52
```

Rounding

- arithmetic results that fall outside of the available precision must round to a neighboring number
    - round down
    - round up
    - round toward zero
    - round to nearest
    - default rounding mode - round to nearest

- round to nearest mode
    - if two numbers are equally near, the one with a 0 in the least significant position of the fraction is chosen
    - overflows are rounded up to ±∞ and underflows are rounded down to 0
- number overflows when its magnitude is too large to be represented
- number underflows when it is too tiny to be represented

Floating-Point Addition

1. extract exponent and fraction bits
2. prepend leading 1 to form the mantissa
3. compare exponents
4. shift smaller mantissa if necessary
5. add mantissas
6. normalize mantissa and adjust exponent if necessary
7. round result
8. assemble exponent and fraction back into floating-point number

5.4 Sequential Building Blocks

Counters

- N-bit binary counter is a sequential arithmetic circuit with clock and reset inputs and an N-bit output Q
    - reset initializes the output to 0
    - counter then advances through all 2N possible outputs in binary order incrementing on the rising edge of the clock
    - can be composed of an adder and a resettable register
    - on each cycle the counter adds 1 to the value stored in the register

Counter

Shift Registers

- shift register
    - clock
    - serial input Sin
    - serial output Sout
    - N parallel outputs QN−1:0
    - on each rising edge of the clock a new bit is shifted in from Sin and all the subsequent contents are shifted forward
    - last bit in the shift register is available at Sout

- can be viewed as serial-to-parallel converters
    - input is provided serially (one bit at a time) at Sin
    - After N cycles the past N inputs are available in parallel at Q.
- can be constructed from N flip-flops connected in series
- can be modified to perform both serial-to-parallel and parallel-to-serial operations by adding a parallel input DN−1:0, and a control signal Load

Scan Chains

- shift registers are often used to test sequential circuits using a technique called scan chains
- testing combinational circuits is relatively straightforward
- testing sequential circuits is more difficult
    - circuits have state
- starting from a known initial condition
    - large number of cycles of test vectors may be needed to put the circuit into a desired state

Shift Register with Parallel Load

5.5. Memory Arrays

- digital systems also require memories to store the data used and generated by such circuits
- registers built from flip-flops are a kind of memory that stores small amounts of data
- memory arrays can efficiently store large amounts of data

Overview

- memory is organized as a two-dimensional array of memory cells
- memory reads or writes the contents of one of the rows of the array
- row is specified by an address
- value read or written is called data
- array with N-bit addresses and M-bit data has 2N rows and M columns
- each row of data is called a word
- array contains 2N M-bit words
- depth of an array is the number of rows
- width is the number of columns (word size)
- size of an array is given as depth × width

Bit Cells

- memory arrays are built as an array of bit cells
    - each stores 1 bit of data

    - each bit cell is connected to a wordline and a bitline
    - memory asserts a single wordline that activates the bit cells in that row for each combination of address bits
    - when the wordline is HIGH, the stored bit transfers to or from the bitline
    - otherwise the bitline is disconnected from the bit cell
- circuitry to store the bit varies with memory type.
- to read a bit cell
    - bitline is initially left floating (Z)
    - wordline is turned ON
        - allows the stored value to drive the bitline to 0 or 1
- to write a bit cell
    - bitline is strongly driven to the desired value
    - wordline is turned ON
        - connectsthe bitline to the stored bit
    - strongly driven bitline overpowers the contents of the bit cell
        - write the desired value into the stored bit

Organization

- practical memories are very large
- behavior of larger arrays can be extrapolated from smaller arrays In this example, the array stores the data from Figure 5.39(b).
- a memory read
    - a wordline is asserted
    - corresponding row of bit cells drives the bitlines HIGH or LOW
- a memory write
    - bitlines are driven HIGH or LOW first
    - a wordline is asserted
        - allows the bitline values to be stored in that row of bit cells

Memory Ports

- all memories have one or more ports
- each gives read and/or write access to one memory address
- multiported memories can access several addresses simultaneously

Memory Types

- memory arrays are specified by their size (depth x width) and the number and type of ports
- all memory arrays store data as an array of bit cells
    - differ in how they store bits
- classified based on how they store bits in the bit cell

- broadest classification
  - random access memory (RAM) vs. read only memory (ROM)
- RAM
  - volatile - loses its data when the power is turned off
- ROM
  - nonvolatile - retains its data indefinitely, even without a power source
- RAM is called random access memory because any data word is accessed with the same delay as any other
- sequential access memory
  - tape recorder
    - accesses nearby data more quickly than faraway data
- ROM is called read only memory because it could only be read but not written
- ROMs are randomly accessed too
- most modern ROMs can be written as well as read
- important distinction to remember is that RAMs are volatile and ROMs are nonvolatile
- two major types of RAMs
  - dynamic RAM (DRAM)
    - stores data as a charge on a capacitor
  - static RAM (SRAM)
    - stores data using a pair of cross-coupled inverters
- many flavors of ROMs that vary by how they are written and erased

Dynamic Random Access Memory (DRAM)

- stores a bit as the presence or absence of charge on a capacitor
  - nMOS transistor behaves as a switch that either connects or disconnects the capacitor from the bitline
  - when wordline is asserted
    - nMOS transistor turns ON and the stored bit value transfers to or from the bitline.
  - when the capacitor is charged to VDD the stored bit is 1
  - when it is discharged to GND the stored bit is 0
  - capacitor node is dynamic because it is not actively driven HIGH or LOW by a transistor tied to VDD or GND
- on read
  - data values are transferred from the capacitor to the bitline
  - destroys the bit value stored on the capacitor
    - data word must be restored (rewritten) after each read
- on write
  - data values are transferred from the bitline to the capacitor
- even when DRAM is not read the contents must be refreshed (read and rewritten) every few milliseconds because the charge on the capacitor gradually leaks away

StaticRandomAccessMemory(SRAM)

- static because stored bits do not need to be refreshed
  - data bit is stored on cross-coupled inverters
  - each cell has two outputs - bitline and bitline
    - when the wordline is asserted both nMOS transistors turn on and data values are transferred to or from the bitlines
  - if noise degrades the value of the stored bit the cross-coupled inverters restore the value (unlike DRAM)

Area and Delay

- flip-flops, SRAMs, and DRAMs are all volatile memories
  - each has different area and delay characteristics

Memory Type	Transistors per Bit Cell	Latency
flip-flop	~20	fast
SRAM	6	medium
DRAM	1	slow

- data bit stored in a flip-flop is available immediately at its output
  - take at least 20 transistors to build
  - more transistors a device has the more area, power, and cost it requires
- DRAM latency is longer than that of SRAM because its bitline is not actively driven by a transistor
  - must wait for charge to move (relatively) slowly from the capacitor to the bitline
  - also fundamentally has lower throughput than SRAM because it must refresh data periodically and after a read
  - technologies such as synchronous DRAM (SDRAM) and double data rate (DDR) SDRAM have been developed to overcome this problem
  - SDRAM uses a clock to pipeline memory accesses
  - DDR SDRAM, sometimes called simply DDR, uses both the rising and falling edges of the clock to access data doubling the throughput for a given clock speed
  - DDR was first standardized in 2000 and ran at 100 to 200 MHz
  - DDR2, DDR3, and DDR4, increased the clock speeds, with speeds in 2012 being over 1 GHz
- latency and throughput also depend on memory size
  - larger tend to be slower than smaller
- best memory type for a particular design depends on the speed, cost, and power constraints

Register Files

- register file - group of registers
    - usually built as a small, multiported SRAM array
        - more compact than an array of flip-flops

Read Only Memory

- stores a bit as the presence or absence of a transistor
- to read the cell
    - bitline is weakly pulled HIGH
    - wordline is turned ON
    - if the transistor is present, it pulls the bitline LOW
    - if it is absent, the bitline remains HIGH
    - ROM bit cell is a combinational circuit and has no state to "forget" if power is turned off
- conceptually ROMs can be built using two-level logic with a group of AND gates followed by a group of OR gates
    - AND gates produce all possible minterms and hence form a decoder
    - ROM can perform any two-level logic function
- in practice ROMs are built from transistors instead of logic gates to reduce their size and cost
- contents of the ROM bit cell during manufacturing by the presence or absence of a transistor in each bit cell
- programmable ROM (PROM) places a transistor in every bit cell but provides a way to connect or disconnect the transistor to ground
- fuse-programmable ROM
    - user programs the ROM by applying a high voltage to selectively blow fuses
    - if the fuse is present the transistor is connected to GND and the cell holds a 0
    - if the fuse is destroyed the transistor is disconnected from ground and the cell holds a 1
    - also called a one-time programmable ROM because the fuse cannot be repaired once it is blown
- reprogrammable ROMs provide a reversible mechanism for connecting or disconnecting the transistor to GND
- erasable PROMs (EPROMs) replace the nMOS transistor and fuse with a floating-gate transistor
    - floating gate is not physically attached to any other wires
    - when suitable high voltages are applied electrons tunnel through an insulator onto the floating gate, turning on the transistor and connecting the bitline to the wordline (decoder output)
    - when the EPROM is exposed to intense ultraviolet (UV) light for about half an hour the electrons are knocked off the floating gate turning the transistor off

    - programming and erasing
- electrically erasable PROMs (EEPROMs) and flash memory use similar principles but include circuitry on the chip for erasing as well as programming
    - EEPROM bit cells are individually erasable
    - flash memory erases larger blocks of bits and is cheaper because fewer erasing circuits are needed
- flash has become an extremely popular way to store large amounts of data in portable battery-powered systems such as cameras and music players
- modern ROMs are not really read only
    - can be programmed (written) as well
- difference between RAM and ROM is that ROMs take a longer time to write but are nonvolatile

Logic Using Memory Arrays

- used primarily for data storage
- can also perform combinational logic functions
- a 2N-word x M-bit memory can perform any combinational function of N inputs and M outputs
- memory arrays used to perform logic are called lookup tables (LUTs)
- by using memory to perform logic the user can look up the output value for a given input combination (address)
    - each address corresponds to a row in the truth table
    - each data bit corresponds to an output value

Memory HDL

RAM

ROM

5.6 Logic Arrays

- gates can be organized into regular arrays
- if the connections are made programmable these logic arrays can be configured to perform any function without the user having to connect wires in specific ways
- logic arrays are mass produced in large quantities
    - inexpensive
- software tools allow users to map logic designs onto these arrays
- most are also reconfigurable
    - valuable during development and also useful in the field
    - system can be upgraded by simply downloading the new configuration

Programmable Logic Array

- implement two-level combinational logic in sum-of-products (SOP) form
- built from an AND array followed by an OR array
- inputs (in true and complementary form) drive an AND array
  - produces implicants which in turn are ORed together to form the outputs
- ROMs can be viewed as a special case of PLAs
  - decoder behaves as an AND plane that produces all 2^M minterms
  - ROM array behaves as an OR plane that produces the outputs
  - if the function does not depend on all 2^M minterms a PLA is likely to be smaller than a ROM
- simple programmable logic devices (SPLDs) are enhanced PLAs that add registers and various other features to the basic AND/OR planes
  - SPLDs and PLAs have largely been displaced by FPGAs which are more flexible and efficient for building large systems

Field Programmable Gate Array

- an array of reconfigurable gates
- user can implement designs on the FPGA using either an HDL or a schematic
- more powerful and more flexible than PLAs
  - can implement both combinational and sequential logic
  - can implement multi-level logic functions (PLAs can only implement two-level logic)
  - built-in multipliers
  - high-speed I/Os
  - data converters including analog-to-digital converters
  - large RAM arrays
  - processors
- built as an array of configurable logic elements (LEs)
  - referred to as configurable logic blocks (CLBs)
  - each LE can be configured to perform combinational or sequential functions
- LEs are surrounded by input/output elements (IOEs) for interfacing with the outside world
  - IOEs connect LE inputs and outputs to pins on the chip package
  - LEs can connect to other LEs and IOEs through programmable routing channels
- Altera Corp.
- Xilinx, Inc.
- designer configures an FPGA by first creating a schematic or HDL description of the design
  - design is synthesized onto the FPGA
  - synthesis tool determines how the LUTs, multiplexers, and routing channels should be configured to perform the specified functions

    - configuration information is downloaded to the FPGA
    - FPGA may download its SRAM contents from a computer in the laboratory or from an EEPROM chip when the system is turned on
    - some manufacturers include an EEPROM directly on the FPGA or use one-time programmable fuses to configure the FPGA

Array Implementations

- ROMs and PLAs commonly use pseudo-nMOS or dynamic circuits instead of conventional logic gates to minimize their size and cost
- PLAs can be built using pseudo-nMOS circuits

Ch. 6 - Architecture

6.1 Introduction

- architecture is the programmer's view of a computer
    - defined by the instruction set (language) and operand locations (registers and memory)
- different architectures exist
    - x86
    - MIPS
    - SPARC
    - PowerPC
- to understanding any computer architecture learn its language
    - instructions = words
    - instruction set = vocabulary
    - instructions indicate both the operation to perform and the operands to use
    - operands may come from memory, from registers, or from the instruction itself
    - instructions are encoded as binary numbers in a format called machine language
- microprocessors are digital systems that read and execute machine language instructions
    - represent the instructions in a symbolic format called assembly language
- instruction sets of different architectures like different dialects than different languages
    - almost all architectures define basic instructions that operate on memory or registers
- architecture does not define the underlying hardware implementation
    - many different hardware implementations of a single architecture exist
    - Intel and Advanced Micro Devices (AMD) both sell various microprocessors belonging to the same x86 architecture
- microarchitecture - the specific arrangement of registers, memories, ALUs, and other building blocks to form a microprocessor
    - many different microarchitectures exist for a single architecture

- use MIPS architecture
    - first developed by John Hennessy and colleagues at Stanford in the 1980s
    - used by
        - Silicon Graphics
        - Nintendo
        - Cisco

6.2 Assembly Language

- human-readable representation of the computer's native language
- instruction specifies operation to perform and the operands on which to operate

Instructions

- mnemonic - indicates what operation to perform
- source operands - operands (data sources) on which operation is performed
- destination operand - operand (data location) to which the result of the operation is written
- MIPS instruction set
    - number of instructions is kept small so that the hardware required to decode the instruction and its operands can be simple, small, and fast
    - elaborate operations (less common) are performed using sequences of multiple simple instructions
- MIPS is a reduced instruction set computer (RISC) architecture
- complex instruction set computers (CISC)
    - architectures with many complex instructions
    - Intel x86 architecture
- the cost of implementing complex instructions in a CISC architecture is added hardware and overhead that slows down simple instructions
- RISC architecture minimizes the hardware complexity and the necessary instruction encoding by keeping the set of distinct instructions small

Operands: Registers, Memory, and Constants

- instruction operates on operands
- computers operate on 1's and 0's - not variable names
- instructions need a physical location from which to retrieve the binary data
- operands can be stored in registers or memory, or they may be constants stored in the instruction itself
- computers use various locations to hold operands in order to optimize for speed and data capacity

- operands stored as constants or in registers are accessed quickly but they hold only a small amount of data
    - additional data must be accessed from memory (large but slow)
- MIPS - 32-bit architecture
    - extended to 64 bits in commercial products (which may be standard now)

Registers

- instructions need to access operands quickly so that they can run fast
- operands stored in memory take a long time to retrieve
- most architectures specify a small number of registers that hold commonly used operands
- MIPS architecture uses 32 registers
    - register set or register file
    - fewer registers = faster access
- small register file is typically built from a small SRAM array
    - uses a small decoder and bitlines connected to relatively few memory cells
    - has a shorter critical path than a large memory does
- MIPS generally stores variables in 18 of the 32 registers
    - $s0–$s7
    - $t0–$t9
- register names beginning with $s are called saved registers
    - store standard local variables
    - special purpose when they are used with function calls
- register names beginning with $t are called temporary registers
    - used for storing temporary variables

The Register Set

```
Name        Number      Use
$0          0           the constant value 0
$at         1           assembler temporary
$v0–$v1     2–3         function return value
$a0–$a3     4–7         function arguments
$t0–$t7     8–15        temporary variables
$s0–$s7     16–23       saved variables
$t8–$t9     24–25       temporary variables
$k0–$k1     26–27       operating system temporaries
$gp         38          global pointer
$sp         29          stack pointer
$fp         30          frame pointer
$ra         31          function return address
```

Memory

- data can also be stored in memory

- memory has many data locations but accessing it takes a longer
- register file = small and fast
- memory = large and slow
- commonly used variables are kept in registers
- MIPS architecture uses 32-bit memory addresses and 32-bit data words (may be outdated)
  - byte-addressable memory - each byte in memory has a unique address
- by convention memory is drawn with low memory addresses toward the bottom and high memory addresses toward the top
- MIPS uses the load word instruction (lw) to read a data word from memory into a register
- lw instruction specifies the effective address in memory as the sum of a base address and an offset
  - base address is a register
    - written in parentheses in the instruction
  - offset is a constant
    - written before the parentheses
- store word instruction (sw) to write a data word from a register into memory
- any register can be used to supply the base address
- MIPS memory model is byte-addressable, not word-addressable
  - each byte has a unique address
  - 32-bit word consists of four 8-bit bytes so each word address is a multiple of 4
  - both the 32-bit word address and the data value are given in hexadecimal
- also provides the lb and sb instructions that load and store single bytes in memory rather than words
  - similar to lw and sw
- byte-addressable memories are organized big-endian or little-endian
  - both - most significant byte (MSB) is on the left and the least significant byte (LSB) is on the right
  - big-endian machines - bytes are numbered starting with 0 at the big (most significant) end
  - little-endian machines - bytes are numbered starting with 0 at the little (least significant) end
  - word addresses are the same in both formats and refer to the same four bytes
    - only the addresses of bytes within a word differ
- IBM PowerPC (old Macs) uses big-endian
- Intel x86 architecture (found in PCs and new Macs) uses little-endian
- some MIPS processors are little-endian and some are big-endian
- choice of endianness is completely arbitrary
  - causes problems when sharing data between computers
- word addresses for lw and sw must be word aligned for MIPS
  - the address must be divisible by 4

Constants/Immediates

- constants/immediates - values are immediately available from the instruction and do not require a register or memory access Add immediate,
- addi - add immediate
    - adds the immediate specified in the instruction to a value in a register
- immediate specified in an instruction is a 16-bit two's complement number in the range [–32,768, 32,767]
- subtraction is equivalent to adding a negative number
    - no subi instruction in the MIPS architecture
- MIPS instruction set makes the compromise of supporting three instruction formats
    - three register operands
    - two register operands and a 16-bit immediate
    - 26-bit immediate

6.3 Machine Language

- program written in assembly language is translated from mnemonics to a representation using only 1's and 0's
- MIPS uses 32-bit instructions
    - encodes all instructions as words that can be stored in memory.
    - some instructions may not require all 32 bits of encoding
    - variable-length instructions would add too much complexity
- MIPS makes the compromise of defining three instruction formats
    - R-type - three registers
    - I-type - two register operands and a 16-bit immediate
    - J-type - (jump instructions) 26-bit immediate

R-Type Instructions

- short for register-type
- three registers as operands
    - two as sources
    - and one as a destination

```
op      rs      rt      shamt   funct
6 bits  5 bits  5 bits  5 bits  6 bits
```

- 32-bit instruction has six fields
    - op (also called opcode or operation code)
        - all R-type instructions have an opcode of 0

    - rs - source operand
    - rt - source operand
    - rd - destination operand
    - shamt
        - used only in shift operations
        - binary value stored indicates the amount to shift
        - 0 for all other R-type instructions
    - funct (also called the function)
        - specifies the operation to be performed
    - operation performed encoded in op and funct

I-Type Instructions

- short for immediate-type
- two register operands and one immediate operand

```
op      rs      rt      imm
6 bits  5 bits  5 bits  16 bits
```

- 32-bit instruction has four fields
    - op
        - specifies opcode which determines operation
    - rs
        - always source operand
    - rt
        - source operand for most
        - destination operand for some
        - listed first in assembly when used as destination
        - always second register field in machine language instruction
    - imm
        - holds the 16-bit immediate operand
        - negative immediate values are represented using 16-bit two's complement notation
        - 16-bit immediate field but used in 32-bit operations
            - upper half should be all 0's for positive immediates
            - upper half should be all 1's for negative immediates
            - called sign extension
                - N-bit two's complement number is sign-extended to an M-bit number (M > N) by copying the sign bit (most significant bit) of the N-bit number into all of the upper bits of the M-bit number
                - does not change its value

- most MIPS instructions sign-extend the immediate
    - an exception to this rule is that logical operations (andi, ori, xori) place 0's in the upper half (called zero extension)

J-Type Instructions

- short for jump-type
- used only with jump instructions

```
op          addr
6 bits      26 bits
```

- op
    - 6-bit opcode for operation to perform (some jump)
- addr
    - single 26-bit address operand to specify an address

Interpreting Machine Language Code

- decipher the fields of each 32-bit instruction word
- all formats start with a 6-bit opcode field
    - start with opcode
        - if 0 - R-type
        - otherwise I-type or J-type

The Power of the Stored Program

- machine language
    - series of 32-bit numbers representing instructions
    - instructions can be stored in memor
- called the stored program concept
    - key reason why computers are so powerful
    - running a different program does not require large amounts of time and effort to reconfigure or rewire hardware
    - only requires writing the new program to memory
    - general purpose computing
- instructions in a stored program are retrieved, or fetched, from memory and executed by the processor
- MIPS programs
    - instructions are normally stored starting at address 0x00400000
    - instruction addresses advance by 4 bytes, not 1
    - processor fetches the instructions from memory sequentially to run or execute the stored program

    - fetched instructions decoded and executed by hardware
    - address of current instruction stored in 32-bit register called the program counter (PC)
        - separate from the 32 registers
        - operating system sets the PC to address 0x00400000
        - processor reads the instruction at that memory address and executes the instruction
        - processor increments PC by 4 to 0x00400004, fetches and executes that instruction
        - repeat
- architectural state of microprocessor holds state of a program
- for MIPS the architectural state consists of the register file and PC
    - if OS saves state at some point in the program it can interrupt the program, do something else then restore the state and continue program properly

6.4 Programming

Arithmetic/Logical Instructions

Logical Instructions

- include and, or, xor, and nor (R-type)
- operate bit-by-bit on two source registers and write the result to the destination register The figure shows the values stored in the destination register , rd, after the instruc- tion executes.
- and instruction is useful for masking bits (i.e., forcing unwanted bits to 0)
    - any subset of register bits can be masked
- or instruction is useful for combining bits from two registers
- MIPS does not provide a NOT instruction
    - NOR instruction can substitute
        - A NOR $0 = NOT A
- logical operations can also operate on immediates
    - I-type instructions are andi, ori, and xori
    - nori is not provided
        - easily implemented using other instructions

Shift Instructions

- shift the value in a register left or right by up to 31 bits

    - multiply or divide by powers of two

- sll (shift left logical), srl (shift right logical), and sra (shift right arithmetic)

- left shifts always fill the least significant bits with 0's

- right shifts can be either logical (0's shift into the most significant bits) or arithmetic (the sign bit shifts into the most significant bits)

- shifting a value left by N is equivalent to multiplying it by 2^N

- arithmetically shifting a value right by N is equivalent to dividing it by 2^N

- MIPS also has variable-shift instructions

  - sllv (shift left logical vari- able), srlv (shift right logical variable), and srav (shift right arithmetic variable)
  - of the form sllv rd, rt, rs
    - order of rt and rs is reversed from most R-type instructions
  - rt ($s1) holds the value to be shifted
  - five least significant bits of rs ($s2) give the amount to shift
  - shifted result is placed in rd
  - shamt field is ignored and should be all 0's

- TODO: more here

Generating Constants

- addi instruction is helpful for assigning 16-bit constants
- use a load upper immediate instruction (lui) followed by an or immediate (ori) instruction to assign 32-bit constants
  - lui loads a 16-bit immediate into the upper half of a register and sets the lower half to 0
  - ori merges a 16-bit immediate into the lower half

Multiplication and Division Instructions

- multiplying two 32-bit numbers produces a 64-bit product
- ividing two 32-bit numbers produces a 32-bit quotient and a 32-bit remainder
- MIPS architecture has two special-purpose registers, hi and lo, which are used to hold the results of multiplication and division
  - mult $s0, $s1 multiplies the values in $s0 and $s1
    - 32 most significant bits of the product are placed in hi and the 32 least significant bits are placed in lo
    - div $s0, $s1 computes $s0/$s1
    - quotient is placed in lo and the remainder is placed in hi
- MIPS provides another multiply instruction that produces a 32-bit result in a general purpose register
  - mul $s1, $s2, $s3 multiplies the values in $s2 and $s3 and places the 32-bit result in $s1

Branching

- computer performs different tasks depending on the input to make decisions
  - if/else statements
  - switch/case statements
  - while loops
  - for loops
  - conditionally execute code depending on some test
- program counter increments by 4 after each instruction sequentially execute instructions
- branch instructions modify the program counter to skip over sections of code or to repeat previous code
- conditional branch instructions perform a test and branch only if the test is TRUE
- unconditional branch instructions, called jumps, always branch

Conditional Branches

- MIPS instruction set has two conditional branch instructions
  - branch if equal (beq)
  - branch if not equal (bne)
- beq branches when the values in two registers are equal, and bne branches when they are not equal
  - NOTE: branches are written as beq rs, rt, imm, where rs is the first source register
    - reversed from most I-type instructions
- assembly code uses labels to indicate instruction locations in the program
  - when the assembly code is translated into machine code, these labels are translated into instruction
  - MIPS assembly labels are followed by a colon (:) and cannot use reserved words, such as instruction mnemonics
  - most programmers indent their instructions but not the labels, to help make labels stand out

Jump

- unconditionally branch, or jump, using the three types of jump instructions
  - jump (j)
    - jumps directly to the instruction at the specified label
  - jump and link (jal)
    - is similar to j but is used by functions to save a return address
  - jump register (jr)
    - jumps to the address held in a register

- after the j target instruction code unconditionally continues executing the add instruction at the label target
    - all of the instructions between the jump and the label are skipped.

Conditional Statements

- if, if/else, and switch/case statements
- each conditionally execute a block of code consisting of one or more statements

If Statements

- executes a block of code, the if block, only when a condition is met
- assembly code for the if statement tests the opposite condition of the one in the high-level code
- bne instruction branches (skips the if block) when not condition
- otherwise, i == j, the branch is not taken, and the if block is executed as desired

If/Else Statements

- execute one of two blocks of code depending on a condition
    - when the condition in the if statement is met, the if block is executed
    - otherwise, the else block is executed
- if/else assembly code tests the opposite condition of the one in the high-level code
- assembly code tests for the opposite condition
    - opposite condition is TRUE, bne skips the if block and executes the else block
    - otherwise, the if block executes and finishes with a jump instruction (j) to jump past the else block

Switch/Case Statements

- execute one of several blocks of code depending on the conditions
    - if no conditions are met, the default block is executed
- equivalent to a series of nested if/else statements

Getting Loopy

- repeatedly execute a block of code depending on a condition
- for loops and while loops are common loop constructs used by high-level languages

While Loops

- repeatedly execute a block of code until a condition is not met
    - assembly code for while loops tests the opposite condition of the one given in the high-level code
    - if that opposite condition is TRUE, the while loop is finished

For Loops

- repeatedly execute a block of code until a condition is not met
    - add support for a loop variable, which typically keeps track of the number of loop executions
    - initialization code executes before the for loop begins
    - condition is tested at the beginning of each loop
    - if the condition is not met, the loop exits
    - loop operation executes at the end of each loop

Magnitude Comparison

- previously used beq and bne to perform equality or inequality comparisons and branches
- MIPS provides the set less than instruction, slt, for magnitude comparison
    - sets rd to 1 when rs < rt. Otherwise, rd is 0

Arrays

- useful for accessing large amounts of similar data
- organized as sequential data addresses in memory
- each element is identified by a number called its index
- number of elements in the array is called the size of the array

Array Indexing

- first step in accessing an array element is to load the base address of the array into a register
- base address points to the start of the array
- offset can be used to access subsequent elements of the array

Bytes and Characters

- numbers in the range [–128, 127] can be stored in a single byte rather than an entire word
- fewer than 256 characters on an English language keyboard, English characters are often represented by bytes
- C language uses the type char to represent a byte or character
- early computers lacked a standard mapping between bytes and English characters, so exchanging text between computers was difficult
- in 1963, the American Standards Association published the American Standard Code for Information Interchange (ASCII), which assigns each text character a unique byte value
- ASCII values are given in hexadecimal
- lower-case and upper-case letters differ by 0x20 (32)

- MIPS provides load byte and store byte instructions to manipulate bytes or characters of data
    - load byte unsigned (lbu)
    - load byte (lb)
    - store byte (sb)
- load byte unsigned (lbu) zero-extends the byte, and load byte (lb) sign-extends the byte to fill the entire 32-bit register
- store byte (sb) stores the least significant byte of the 32-bit register into the specified byte address in memory
- lbu loads the byte at memory address 2 into the least significant byte of $s1 and fills the remaining register bits with 0
- lb loads the sign-extended byte at memory address 2 into $s2
- sb stores the least significant byte of $s3 into memory byte 3
    - it replaces 0xF7 with 0x9B
    - more significant bytes of $s3 are ignored
- series of characters is called a string
    - variable length
    - programming languages must provide a way to determine the length or end of the string
    - null character (0x00) signifies the end of a string in C

Function Calls

- aka procedures
- reuse frequently accessed code and to make a program more modular and readable
- inputs, called arguments, and an output, called the return value
- functions should calculate the return value and cause no other unintended side effects
- when one function calls another, the calling function, the caller, and the called function, the callee, must agree on where to put the arguments and the return value
- MIPS - the caller conventionally places up to four arguments in registers $a0–$a3 before making the function call, and the callee places the return value in registers $v0–$v1 before finishing
    - convention allows both functions to know where to find the arguments and return value, even if the caller and callee were written by different people
- callee must not interfere with the function of the caller
    - means that the callee must know where to return to after it completes and it must not trample on any registers or memory needed by the caller
    - caller stores the return address in $ra at the same time it jumps to the callee using the jump and link instruction (jal)
    - callee must not overwrite any architectural state or memory that the caller is depending on

    - callee must leave the saved registers, \$s0–\$s7, \$ra, and the stack, a portion of memory used for temporary variables, unmodified

Function Calls and Returns

- MIPS uses the jump and link instruction (jal) to call a function and the jump register instruction (jr) to return from a function
- jump and link (jal) and jump register (jr \$ra) are the two essential instructions needed for a function call
- jal performs two operations: it stores the address of the next instruction (the instruction after jal) in the return address register (\$ra), and it jumps to the target instruction

Input Arguments and Return Values

- by MIPS convention, functions use \$a0–\$a3 for input arguments and \$v0–\$v1 for the return value
- according to MIPS convention, the calling function, main, places the function arguments from left to right into the input registers, \$a0–\$a3
- called function stores the return value in the return register, \$v0
- a function that returns a 64-bit value, such as a double-precision floating point number, uses both return registers, \$v0 and \$v1
- when a function with more than four arguments is called, the additional input arguments are placed on the stack, which we discuss next

The Stack

- memory that is used to save local variables within a function
- expands (uses more memory) as the processor needs more scratch space and contracts (uses less memory) when the processor no longer needs the variables stored there
- last-in-first-out (LIFO) queue
    - last item pushed onto the stack is the first one that can be pulled (popped) off
- each function may allocate stack space to store local variables but must deallocate it before returning
- top of the stack is the most recently allocated space
- MIPS stack grows down in memory
- stack expands to lower memory addresses when a program needs more scratch space
- stack pointer, \$sp, is a special MIPS register that points to the top of the stack
- points to (gives the address of) data

- stack pointer ($sp) starts at a high memory address and decrements to expand as needed
- one of the important uses of the stack is to save and restore registers that are used by a function
    - function should not modify any registers besides the one containing the return value $v0
- function saves registers on the stack before it modifies them, then restores them from the stack before it returns
    i. makes space on the stack to store the values of one or more registers
    ii. stores the values of the registers on the stack
    iii. executes the function using the registers
    iv. restores the original values of the registers from the stack
    v. deallocates space on the stack
- stack space that a function allocates for itself is called its stack frame
- each function should access only its own stack frame, not the frames belonging to other functions

Preserved Registers

- if the calling function does not use those registers, the effort to save and restore them is wasted
- to avoid this waste, MIPS divides registers into preserved and nonpreserved categories
- preserved registers include $s0–$s7 (hence their name, saved)
- nonpreserved registers include $t0–$t9 (hence their name, temporary)
- function must save and restore any of the preserved registers that it wishes to use, but it can change the nonpreserved registers freely
- when one function calls another, the former is the caller and the latter is the callee
- callee must save and restore any preserved registers that it wishes to use
- callee may change any of the nonpreserved registers. Hence, if the caller is holding active data in a nonpreserved register, the caller needs to save that nonpreserved register before making the function call and then needs to restore it afterward
- preserved registers are also called callee-save, and nonpreserved registers are called caller-save
- $s0–$s7 are generally used to hold local variables within a function, so they must be saved
- $ra must also be saved, so that the function knows where to return

- $t0–$t9 are used to hold temporary results before they are assigned to local variables
    - calculations typically complete before a function call is made, so they are not preserved, and it is rare that the caller needs to save them
- $a0–$a3 are often overwritten in the process of calling a function
    - must be saved by the caller if the caller depends on any of its own arguments after a called function returns
- $v0–$v1 certainly should not be preserved, because the callee returns its result in these registers
- stack above the stack pointer is automatically preserved as long as the callee does not write to memory addresses above $sp
    - does not modify the stack frame of any other functions
    - stack pointer itself is preserved, because the callee deallocates its stack frame before returning by adding back the same amount that it subtracted from $sp at the beginning of the function

Recursive Function Calls

- function that does not call others is called a leaf function
- function that does call others is called a nonleaf function
- nonleaf functions are somewhat more complicated because they may need to save nonpreserved registers on the stack before they call another function, and then restore those registers afterward
    - caller saves any non-preserved registers ($t0–$t9 and $a0–$a3) that are needed after the call
    - callee saves any of the preserved registers ($s0–$s7 and $ra) that it intends to modify
- recursive function
    - nonleaf function that calls itself

Additional Arguments and Local Variables

- functions may have more than four input arguments and local variables
    - stack is used to store these temporary values
- by MIPS convention, if a function has more than four arguments, the first four are passed in the argument registers as usual
- additional arguments are passed on the stack, just above $sp
- caller must expand its stack to make room for the additional arguments
- function can also declare local variables or arrays
- ocal variables are declared within a function and can be accessed only within that function

- local variables are stored in $s0–$s7
  - if there are too many local variables, they can also be stored in the function's stack frame
  - local arrays are stored on the stack
- callee's stack frame
  - holds the function's own arguments, the return address, and any of the saved registers that the function will modify
  - holds local arrays and any excess local variables
  - the callee has more than four arguments, it finds them in the caller's stack frame
  - accessing additional input arguments is the one exception in which a function can access stack data not in its own stack frame

Addressing Modes

- MIPS uses five addressing modes
  - register-only
  - immediate
  - base
  - PC-relative
  - pseudo-direct
- first three modes (register-only, immediate, and base addressing) define modes of reading and writing operands
- last two (PC-relative and pseudo-direct addressing) define modes of writing the pro- gram counter, PC

Register-Only Addressing

- registers for all source and destination operands
- all R-type instructions use register-only addressing

Immediate Addressing

- uses the 16-bit immediate along with registers as operands
- some I-type instructions, such as add immediate (addi) and load upper immediate (lui), use immediate addressing

Base Addressing

- memory access instructions, such as load word (lw) and store word (sw), use base addressing
  - effective address of the memory operand is found by adding the base address in register rs to the sign-extended 16-bit offset found in the immediate field

PC-Relative Addressing

- conditional branch instructions use PC-relative addressing to specify the new value of the PC if the branch is taken
- signed offset in the immediate field is added to the PC to obtain the new PC
    - branch destination address is said to be relative to the current PC
- 16-bit immediate field gives the number of instructions between the BTA and the instruction after the branch instruction (the instruction at PC + 4)
- processor calculates the BTA from the instruction by sign-extending the 16-bit immediate, multiplying it by 4 (to convert words to bytes), and adding it to PC + 4

Pseudo-Direct Addressing

- address is specified in the instruction
- jump instructions, j and jal, ideally would use direct addressing to specify a 32-bit jump target address (JTA) to indicate the instruction address to execute next
- J-type instruction encoding does not have enough bits to specify a full 32-bit JTA
    - six bits of the instruction are used for the opcode, so only 26 bits are left to encode the JTA
    - two least significant bits, JTA1:0, should always be 0, because instructions are word aligned
    - next 26 bits, JTA27:2, are taken from the addr field of the instruction
    - four most significant bits, JTA31:28, are obtained from the four most significant bits of PC + 4
    - called pseudo-direct
- processor calculates the JTA from the J-type instruction by appending two 0's and prepending the four most significant bits of PC + 4 to the 26-bit address field (addr)
- because the four most significant bits of the JTA are taken from PC + 4, the jump range is limited
- all J-type instructions, j and jal, use pseudo-direct addressing
- NOTE: jump register instruction, jr, is not a J-type instruction
    - R-type instruction that jumps to the 32-bit value held in register rs

Lights, Camera, Action: Compiling, Assembling, and Loading

The Memory Map

- defines where code, data, and stack memory are located
- with 32-bit addresses, the MIPS address space spans 232 bytes = 4 gigabytes (GB)
- word addresses are divisible by 4 and range from 0 to 0xFFFFFFFC

- MIPS architecture divides the address space into four parts or segments
  - text segment
  - global data segment
  - dynamic data segment
  - reserved segments

The Text Segment

- stores the machine language program
- large enough to accommodate almost 256 MB of code
- NOTE: that the four most significant bits of the address in the text space are all 0, so the j instruction can directly jump to any address in the program

The Global Data Segment

- stores global variables that, in contrast to local variables, can be seen by all functions in a program
  - defined at start-up, before the program begins executing
  - declared outside the main function in a C program and can be accessed by any function
- global data segment is large enough to store 64 KB of global variables
- global variables are accessed using the global pointer ($gp), which is initialized to 0x100080000
  - $gp does not change during program execution unlike the stack pointer ($sp)
  - any global variable can be accessed with a 16-bit positive or negative offset from $gp
  - offset is known at assembly time, so the variables can be efficiently accessed using base addressing mode with constant offsets

The Dynamic Data Segment

- holds the stack and the heap
  - data in this segment are not known at start-up but are dynamically allocated and deallocated throughout the execution of the program
- largest segment of memory used by a program, spanning almost 2 GB of the address space
- heap stores data that is allocated by the program during runtime
- memory allocations are made by the malloc function in C
- in C++ and Java, new is used to allocate memory
- heap data can be used and discarded in any order
- heap grows upward from the bottom of the dynamic data segment
- if the stack and heap ever grow into each other, the program's data can become corrupted
  - memory allocator tries to ensure that this never happens by returning an out-of-memory error if there is insufficient space to allocate more dynamic data

The Reserved Segments

- used by the operating system and cannot directly be used by the program
- part of the reserved memory is used for interrupts and for memory-mapped I/O

Translating and Starting a Program

- steps required to translate a program from a high-level language into machine language and to start executing that program
  - the high-level code is compiled into assembly code
  - assembly code is assembled into machine code in an object file
  - linker combines the machine code with object code from libraries and other files to produce an entire executable program
  - most compilers perform all three steps of compiling, assembling, and linking in practice
  - loader loads the program into memory and starts execution

```
High–Level Code –> Compiler –> Assembly Code –> Assembler –> Object File –> Linker
```

Step 1: Compilation

- compiler translates high-level code into assembly language
- the .data and .text keywords are assembler directives that indicate where the text and data segments begin
- labels are used for global variables
- their storage location will be determined by the assembler

Step 2: Assembling

- assembler turns the assembly language code into an object file containing machine language code
- makes two passes through the assembly code
  - first pass - the assembler assigns instruction addresses and finds all the symbols, such as labels and global variable names
  - the names and addresses of the symbols are kept in a symbol table
  - symbol addresses are filled in after the first pass, when the addresses of labels are known
  - global variables are assigned storage locations in the global data segment of memory, starting at memory address 0x10000000
  - second pass through - assembler produces the machine language code
  - addresses for the global variables and labels are taken from the symbol table
  - machine language code and symbol table are stored in the object file

Step 3: Linking

- most large programs contain more than one file
- if the programmer changes only one of the files, it would be wasteful to recompile and reassemble the other files
- programs often call functions in library files
  - library files almost never change
- linker combines all of the object files into one machine language file called the executable
- relocates the data and instructions in the object files so that they are not all on top of each other
- uses the information in the symbol tables to adjust the addresses of global variables and of labels that are relocated

Step 4: Loading

- operating system loads a program by reading the text segment of the executable file from a storage device (usually the hard disk) into the text segment of memory
- operating system sets $gp to 0x10008000 (the middle of the global data segment) and $sp to 0x7FFFFFFC (the top of the dynamic data segment), then performs a jal 0x00400000 to jump to the beginning of the program

Odds and Ends

Pseudoinstructions

- if an instruction is not available in the MIPS instruction set it is probably because the same operation can be performed using one or more existing MIPS instructions
  - MIPS is a reduced instruction set computer (RISC)
  - instruction size and hardware complexity are minimized by keeping the number of instructions small
- MIPS defines pseudoinstructions that are not actually part of the instruction set but are commonly used by programmers and compilers
- when converted to machine code, pseudoinstructions are translated into one or more MIPS instructions
- load immediate pseudoinstruction (li) loads a 32-bit constant using a combination of lui and ori instructions
- no operation pseudoinstruction (nop, pronounced "no op") performs no operation
  - PC is incremented by 4 upon its execution
  - no other registers or memory values are altered
  - the machine code for the nop instruction is 0x00000000.
- some pseudoinstructions require a temporary register for intermediate calculations

Exceptions

- like an unscheduled function call that jumps to a new address
- may be caused by hardware or software
    - processor may receive notification that the user pressed a key on a keyboard
    - processor may stop what it is doing, determine which key was pressed, save it for future reference, then resume the program that was running
    - hardware exception triggered by an input/output (I/O) device such as a keyboard is often called an interrupt
    - program may encounter an error condition such as an undefined instruction
    - program then jumps to code in the operating system (OS), which may choose to terminate the offending program
- software exceptions are sometimes called traps
- other causes of exceptions
    - division by zero
    - attempts to read nonexistent memory
    - hardware malfunctions
    - debugger breakpoints
    - arithmetic overflow
- processor records the cause of an exception and the value of the PC at the time the exception occurs
    - jumps to the exception handler function
    - exception handler is code (usually in the OS) that examines the cause of the exception and responds appropriately (by reading the keyboard on a hardware interrupt, for example)
    - then returns to the program that was executing before the exception took place
    - in MIPS, the exception handler is always located at 0x80000180
    - when an exception occurs, the processor always jumps to this instruction address, regardless of the cause
- MIPS architecture uses a special-purpose register, called the Cause register, to record the cause of the exception
- different codes are used to record different exception causes
- exception handler code reads the Cause register to determine how to handle the exception
- some other architectures jump to a different exception handler for each different cause instead of using a Cause register
- MIPS uses another special-purpose register called the Exception Program Counter (EPC) to store the value of the PC at the time an exception takes place
- processor returns to the address in EPC after handling the exception
    - analogous to using $ra to store the old value of the PC during a jal instruction
- EPC and Cause registers are not part of the MIPS register file

- mfc0 (move from coprocessor 0) instruction copies these and other special-purpose registers into one of the general purpose registers
- Coprocessor 0 is called the MIPS processor control
  - handles interrupts and processor diagnostics
- syscall and break instructions cause traps to perform system calls or debugger breakpoints
- exception handler uses the EPC to look up the instruction and determine the nature of the system call or breakpoint by looking at the fields of the instruction
- summary
  - an exception causes the processor to jump to the exception handler
  - exception handler saves registers on the stack, then uses mfc0 to look at the cause and respond accordingly
  - when the handler is finished, it restores the registers from the stack, copies the return address from EPC to $k0 using mfc0, and returns using jr $k0

Signed and Unsigned Instructions

- MIPS has certain instructions that come in signed and unsigned flavors
  - addition and subtraction
  - multiplication and division
  - set less than
  - partial word loads

Addition and Subtraction

- performed identically whether the number is signed or unsigned
- interpretation of the results is different
- if two large signed numbers are added together, the result may incorrectly produce the opposite sign
- adding two huge negative numbers gives a positive result
  - called arithmetic overflow
- the C language ignores arithmetic overflows, but other languages, such as Fortran, require that the program be notified
- MIPS processor takes an exception on arithmetic overflow
  - program can decide what to do about the overflow
- signed versions are add, addi, and sub
- unsigned versions are addu, addiu, and subu
- versions are identical except that signed versions trigger an exception on overflow, whereas unsigned versions do not
- C programs technically use the unsigned versions of these instructions because they ignore the exceptions

Multiplication and Division

- behave differently for signed and unsigned numbers
- as an unsigned number, 0xFFFFFFFF represents a large number, but as a signed number it represents –1
    - 0xFFFFFFFF x 0xFFFFFFFF would equal 0xFFFFFFFE00000001 if the numbers were unsigned but 0x0000000000000001 if the numbers were signed
- mult and div treat the operands as signed numbers
- multu and divu treat the operands as unsigned numbers

Set Less Than

- set less than instructions can compare either two registers (slt) or a register and an immediate (slti)
- signed (slt and slti) and unsigned (sltu and sltiu) versions
- in a signed com- parison, 0x80000000 is less than any other number, because it is the most negative two's complement number
- in an unsigned comparison, 0x80000000 is greater than 0x7FFFFFFF but less than 0x80000001, because all numbers are positive
- NOTE: sltiu sign-extends the immediate before treating it as an unsigned number

Loads

- byte loads come in signed (lb) and unsigned (lbu) versions
- lb sign-extends the byte, and lbu zero-extends the byte to fill the entire 32-bit register
- MIPS provides signed and unsigned half-word loads (lh and lhu), which load two bytes into the lower half and sign- or zero-extend the upper half of the word

Floating-Point Instructions

- MIPS architecture defines an optional floating-point coprocessor, known as coprocessor 1
- early MIPS implementations - floating-point coprocessor was a separate chip that users could purchase if they needed fast floating-point math
- in recent MIPS implementations, the floating-point coprocessor is built in alongside the main processor
- MIPS defines thirty-two 32-bit floating-point registers, $f0–$f31
    - separate from the ordinary registers used so far
- supports both single- and double-precision IEEE floating-point arithmetic
- double-precision (64-bit) numbers are stored in pairs of 32-bit registers, so only the 16 even-numbered registers ($f0, $f2, $f4,..., $f30) are used to specify double- precision operations
- floating-point instructions all have an opcode of 17 (100012)
    - require both a funct field and a cop (coprocessor) field to indicate the type of instruction
    - MIPS defines the F-type instruction format for floating-point instructions

    - come in both single- and double-precision flavors
    - cop = 16 (100002) for sin- gle-precision instructions or 17 (100012) for double-precision instructions
    - F-type instructions have two source operands, fs and ft, and one destination, fd (like R-type)
    - precision is indicated by .s and .d in the mnemonic
- floating-point arithmetic instructions include addition (add.s, add.d), subtraction (sub.s, sub.d), multiplication (mul.s, mul.d), and division (div.s, div.d) as well as negation (neg.s, neg.d) and absolute value (abs.s, abs.d)
- floating-point branches have two parts
    - compare instruction is used to set or clear the floating-point condition flag (fpcond)
    - conditional branch checks the value of the flag
    - compare instructions include equality (c.seq.s/c.seq.d), less than (c.lt.s/c.lt.d), and less than or equal to (c.le.s/c.le.d)
    - conditional branch instructions are bc1f and bc1t that branch if fpcond is FALSE or TRUE, respectively
    - inequality, greater than or equal to, and greater than comparisons are performed with seq, lt, and le, followed by bc1f
- floating-point registers are loaded and stored from memory using lwc1 and swc1
    - move 32 bits, so two are necessary to handle a double-precision number

Real-World Perspective: x86 Architecture

- x86, also called IA-32, is a 32-bit architecture originally developed by Intel
- AMD also sells x86 compatible microprocessors
- x86 architecture has a long and convoluted history dating back to 1978, when Intel announced the 16-bit 8086 microprocessor
- IBM selected the 8086 and its cousin, the 8088, for IBM's first personal computers
- In 1985, Intel introduced the 32-bit 80386 microprocessor, which was backward compatible with the 8086, so it could run software developed for earlier PCs
- processor architectures compatible with the 80386 are called x86 processors
- Pentium, Core, and Athlon processors are well known x86 processors
- various groups at Intel and AMD over many years have shoehorned more instructions and capabilities into the antiquated architecture
    - result is far less elegant than MIPS
- software compatibility is far more important than technical elegance, so x86 has been the de facto PC standard for more than two decades
- more than 100 million x86 processors are sold every year
- huge market justifies more than $5 billion of research and development annually to continue improving the processors

- example of a Complex Instruction Set Computer (CISC) architecture
- programs for CISC architectures usually require fewer instructions
- instruction encodings were selected to be more compact, so as to save memory, when RAM was far more expensive than it is today
  - instructions are of variable length and are often less than 32 bits
  - trade-off is that complicated instructions are more difficult to decode and tend to execute more slowly

x86 Registers

- 8086 microprocessor provided eight 16-bit registers
  - could separately access the upper and lower eight bits of some of these registers
- when the 32-bit 80386 was introduced, the registers were extended to 32 bits
  - registers are called EAX, ECX, EDX, EBX, ESP, EBP, ESI, and EDI
- for backward compatibility, the bottom 16 bits and some of the bottom 8-bit portions are also usable
- eight registers are almost, but not quite, general purpose
  - certain instructions cannot use certain registers
  - other instructions always put their results in certain registers
- x86 program counter is called EIP (the extended instruction pointer)
  - advances from one instruction to the next or can be changed with branch, jump, and function call instructions

x86 Operands

- MIPS instructions always act on registers or immediates
- explicit load and store instructions are needed to move data between memory and the registers
- x86 instructions may operate on registers, immediates, or memory
  - partially compensates for the small set of registers
- MIPS instructions generally specify three operands
  - two sources and one destination
- x86 instructions specify only two operands
  - source
  - second
    - both a source and the destination
  - always overwrite one of their sources with the result
  - combinations are possible except memory to memory
- supports a much wider variety of memory addressing modes
- memory locations are specified with any combination of a base register, displacement, and a scaled index register
  - displacement can be an 8-, 16-, or 32-bit value

    - scale multiplying the index register can be 1, 2, 4, or 8
    - base + displacement mode is equivalent to the MIPS base addressing mode for loads and stores
    - scaled index provides an easy way to access arrays or structures of 2-, 4-, or 8-byte elements without having to issue a sequence of instructions to generate the address

table operand locations

table memory addressing modes

Status Flags

Meaning AH <– AH + BL AX <– AX + 0xFFFF EAX <– EAX + EDX Data Size 8-bit 16-bit 32-bit

- uses status flags (also called condition codes) to make decisions about branches and to keep track of carries and arithmetic overflow
- uses a 32-bit register, called EFLAGS, that stores the status flags
- architectural state of an x86 processor includes EFLAGS as well as the eight registers and the EIP

x86 Instructions

- larger set of instructions than MIPS
- also has instructions for floating-point arithmetic and for arithmetic on multiple short data elements packed into a longer word
- D indicates the destination (a register or memory location)
- S indicates the source (a register, memory location, or immediate).
- NOTE: some instructions always act on specific registers
    - 32 x 32-bit multiplication always takes one of the sources from EAX and always puts the 64-bit result in EDX and EAX
    - LOOP always stores the loop counter in ECX
    - PUSH, POP, CALL, and RET use the stack pointer, ESP
- conditional jumps check the flags and branch if the appropriate condition is met
    - come in many flavors
    - JZ jumps if the zero flag (ZF) is 1
    - JNZ jumps if the zero flag is 0
    - jumps usually follow an instruction, such as the compare instruction (CMP), that sets the flags

x86 Instruction Encoding

- instruction encodings are truly messy, a legacy of decades of piecemeal changes
- NOTE: for 32 bit
- x86 instructions vary from 1 to 15 bytes

- opcode may be 1, 2, or 3 bytes
- followed by four optional fields
    - Mod R/M
        - specifies an addressing mode
        - byte uses the 2-bit Mod and 3-bit R/M field to specify the addressing mode for one of the operands
        - operand can come from one of the eight registers, or from one of 24 memory addressing modes
        - due to artifacts in the encodings, the ESP and EBP registers are not available for use as the base or index register in certain addressing modes
        - Reg field specifies the register used as the other operand
        - for certain instructions that do not require a second operand, the Reg field is used to specify three more bits of the opcode
    - SIB
        - specifies the scale, index, and base registers in certain addressing modes
        - specifies the index register and the scale (1, 2, 4, or 8) in addressing modes using a scaled index register
        - if both a base and index are used, the SIB byte also specifies the base register
    - Displacement
        - indicates a 1-, 2-, or 4-byte displacement in certain addressing modes
    - Immediate
        - 1-, 2-, or 4-byte constant for instructions using an immediate as the source operand
    - instruction can be preceded by up to four optional byte-long prefixes that modify its behavior
- MIPS fully specifies the instruction in the opcode and funct fields of the instruction
- x86 uses a variable number of bits to specify different instructions
    - uses fewer bits to specify more common instructions, decreasing the average length of the instructions
    - ome instructions even have multiple opcodes
    - add AL, imm8 performs an 8-bit add of an immediate to AL
        - is represented with the 1-byte opcode, 0x04, followed by a 1-byte immediate
    - A register (AL, AX, or EAX) is called the accumulator
    - add D, imm8 performs an 8-bit add of an immediate to an arbitrary destination, D (memory or a register)
        - represented with the 1-byte opcode 0x80 followed by one or more bytes specifying D, followed by a 1-byte immediate
- many instructions have shortened encodings when the destination is the accumulator

Other x86 Peculiarities

- 80286 introduced segmentation to divide memory into segments of up to 64 KB in length

- when the OS enables segmentation, addresses are computed relative to the beginning of the segment
- processor checks for addresses that go beyond the end of the segment and indicates an error, thus preventing programs from accessing memory outside their own segment
- segmentation proved to be a hassle for programmers and is not used in modern versions of the Windows operating system
- x86 contains string instructions that act on entire strings of bytes or words
- operations include moving, comparing, or scanning for a specific value
  - these instructions are usually slower than performing the equivalent operation with a series of simpler instructions, so they are best avoided
- 0x66 prefix is used to choose between 16- and 32-bit operand sizes
  - other prefixes include ones used to lock the bus (to control access to shared variables in a multiprocessor system), to predict whether a branch will be taken or not, and to repeat the instruction during a string move

The Big Picture

- x86 tends to have shorter programs, because a complex instruction is equivalent to a series of simple MIPS instructions and because the instructions are encoded to minimize memory use
- has too few registers, and the instructions are difficult to decode
- x86 is firmly entrenched as the dominant computer architecture for PCs

Ch. 7 - Microarchitecture

7.1 Introduction

- connection between logic and architecture
  - specific arrangement of registers, ALUs, finite state machines (FSMs), memories, and other logic building blocks needed to implement an architecture
- particular architecture (MIPS) may have many different microarchitectures
  - different trade-offs of performance, cost, and complexity
  - all run the same programs
  - internal designs vary widely

Architectural State and Instruction Set

- computer architecture is defined by its instruction set and architectural state
- architectural state for MIPS consists
  - program counter
  - 32 registers

- any MIPS microarchitecture must contain all of this state
- based on the current architectural state the processor executes a particular instruction with a particular set of data to produce a new architectural state
- some microarchitectures contain additional nonarchitectural state to either simplify the logic or improve performance
- consider only a subset of the MIPS instruction set for simplicity
    - R-type arithmetic/logic instructions
        - add, sub, and, or, slt
    - memory instructions
        - lw, sw
    - branches
        - beq
    - extended to handle addi and j
    - sufficient to write many interesting programs

Design Process

- two interacting parts
    - datapath
        - operates on words of data
        - contains structures such as memories, registers, ALUs, and multiplexers
    - control unit
        - receives the current instruction from the datapath and tells the datapath how to execute instruction
        - produces multiplexer select, register enable, and memory write signals to control the operation of the datapath
- start with hardware containing the state elements
    - include the memories and the architectural state (the program counter and registers)
- add blocks of combinational logic between the state elements to compute the new state based on the current state
- instruction is read from part of memory
    - load and store instructions then read or write data from another part of memory
- partition the overall memory into two smaller memories
    - instructions
    - data
- four state elements
    - the program counter
    - register file
    - instruction memory
    - data memory
- data busses

- address busses
- control signals
- program counter
  - ordinary 32-bit register
  - output, PC, points to the current instruction
  - input, PC', indicates the address of the next instruction
- instruction memory
  - single read port
  - takes a 32-bit instruction address input, A
  - reads the 32-bit data (i.e., instruction) from that address onto the read data output, RD
- 32-element x 32-bit register file
  - two read ports and one write port
  - read ports take 5-bit address inputs, A1 and A2, each specifying one of 25 = 32 registers as source operands
  - read the 32-bit register values onto read data outputs RD1 and RD2, respectively
  - write port takes
    - 5-bit address input, A3
    - 32-bit write data input, WD
    - write enable input, WE3
    - a clock
  - if write enable is 1 the register file writes the data into the specified register on the rising edge of the clock
- data memory
  - single read/write port
  - if the write enable, WE, is 1 it writes data WD into address A on the rising edge of the clock
  - if the write enable is 0, it reads address A onto RD
- instruction memory, register file, and data memory are all read combinationally
  - if the address changes the new data appears at RD after some propagation delay
  - no clock is involved
  - written only on the rising edge of the clock
- state of the system is changed only at the clock edge
- address, data, and write enable must setup sometime before the clock edge and must remain stable until a hold time after the clock edge
- state elements are synchronous sequential circuits because they change their state only on the rising edge of the clock
- microprocessor is built of clocked state elements and combinational logic
  - also synchronous sequential circuit
- processor can be viewed as a
  - giant finite state machine

    - a collection of simpler interacting state machines

MIPS Microarchitectures

- three microarchitectures viewed for the MIPS processor architecture
    - differ in the way that the state elements are connected together and in the amount of nonarchitectural state
    - single-cycle
        - executes an entire instruction in one cycle
        - does not require any non-architectural state
        - cycle time is limited by the slowest instruction
    - multicycle
        - executes instructions in a series of shorter cycles
        - simpler instructions execute in fewer cycles than complicated ones
        - reduces the hardware cost by reusing expensive hardware blocks such as adders and memories
            - adder may be used on several different cycles for several purposes while carrying out a single instruction
            - accomplishes this by adding several nonarchitectural registers to hold intermediate results
        - executes only one instruction at a time
        - each instruction takes multiple clock cycles
    - pipelined
        - applies pipelining to the single-cycle microarchitecture
        - can execute several instructions simultaneously
        - improves the throughput significantly
        - must add logic to handle dependencies between simultaneously executing instructions
        - requires nonarchitectural pipeline registers
        - added logic and registers are worthwhile
            - all commercial high-performance processors use pipelining today

7.2 Performance Analysis

- only gimmick-free way to measure performance is by measuring the execution time of a program of interest to application domain in question
    - the computer that executes the program fastest has the highest performance
- next best choice is to measure the total execution time of a collection of programs that are similar
- called benchmarks
    - execution times of these programs are commonly published

- execution time of a program, measured in seconds, is given by

```
Execution Time = (# instructions)(cycles/instruction)(seconds/cycle)
```

- number of instructions in a program depends on the processor architecture
    - some architectures have complicated instructions that do more work per instruction
        - complicated instructions are often slower to execute in hardware
    - number of instructions also depends on the programmer
- number of cycles per instruction, often called CPI, is the number of clock cycles required to execute an average instruction
    - reciprocal of the throughput (instructions per cycle, or IPC)
    - different microarchitectures have different CPIs
- number of seconds per cycle is the clock period, Tc
    - determined by the critical path through the logic on the processor
    - different microarchitectures have different clock periods
    - logic and circuit designs also significantly affect the clock period
- challenge of the microarchitect is to choose the design that minimizes the execution time while satisfying constraints cost and/or power consumption
- determining the best choice requires careful analysis because microarchitectural decisions affect both CPI and Tc and are influenced by logic and circuit designs
- many other factors that affect overall computer performance
    - hard disk
    - memory
    - graphics system
    - network connection

7.3 Single-Cycle Processor

- first design
    - MIPS microarchitecture that executes instructions in a single cycle
- start by constructing the datapath by connecting state elements with combinational logic that can execute the various instructions
- control signals determine which specific instruction is carried out by the datapath at any given time
- controller contains combinational logic that generates the appropriate control signals based on the current instruction

Single-Cycle Datapath

- development of a single-cycle datapath adding one piece at a time to the state elements

- program counter (PC) register contains the address of the instruction to execute
    - read this instruction from instruction memory
    - PC is simply connected to the address input of the instruction memory
    - instruction memory reads out, or fetches, the 32-bit instruction, labeled Instr
- processor's actions depend on the specific instruction that was fetched
- first determine datapath connections for the lw instruction then generalize the datapath to handle the other instructions
- lw instruction
    - read the source register containing the base address
        - specified in the rs field of the instruction, Instr25:21
        - these bits of the instruction are connected to the address input of one of the register file read ports, A1
    - register file reads the register value onto RD1
    - requires an offset
        - stored in the immediate field of the instruction, Instr15:0
        - 16-bit immediate might be either positive or negative, it must be sign-extended to 32 bits
        - 32-bit sign-extended value is called SignImm
        - (sign extension simply copies the sign bit (most significant bit) of a short input into all of the upper bits of the longer output)
        - SignImm15:0 = Instr15:0 and SignImm31:16 = Instr15
    - processor must add the base address to the offset to find the address to read from memory
    - requires an ALU to perform this addition
        - receives two operands, SrcA and SrcB
        - SrcA comes from the register file, and SrcB comes from the sign-extended immediate
        - 3-bit ALUControl signal specifies the operation
        - generates a 32-bit ALUResult and a Zero flag, that indicates whether ALUResult == 0
        - ALUControl signal should be set to 010 to add the base address and offset for lw
        - ALUResult is sent to the data memory as the address for the load instruction
        - data is read from the data memory onto the ReadData bus, then written back to the destination register in the register file at the end of the cycle
        - port 3 of the register file is the write port
    - destination register for the lw instruction is specified in the rt field, Instr20:16
        - connected to the port 3 address input, A3, of the register file
    - ReadData bus is connected to the port 3 write data input, WD3, of the register file
    - control signal called RegWrite is connected to the port 3 write enable input, WE3, and is asserted during a lw instruction so that the data value is written into the register file

    - write takes place on the rising edge of the clock at the end of the cycle
    - processor must compute the address of the next instruction, PC', while the instruction is being executed
    - next instruction is at PC + 4 (32 bits = 4 bytes)
        - use another adder to increment the PC by 4
    - new address is written into the program counter on the next rising edge of the clock
    - completes the datapath for the lw instruction
- extend the datapath to also handle the sw instruction
    - reads a base address from port 1 of the register file and sign-extends an immediate (like lw)
    - ALU adds the base address to the immediate to find the memory address
        - all of these functions are already supported by the datapath described above
    - also reads a second register from the register file and writes it to the data memory
        - register is specified in the rt field, Instr20:16
        - bits of the instruction are connected to the second register file read port, A2
        - register value is read onto the RD2 port
        - connected to the write data port of the data memory
        - write enable port of the data memory, WE, is controlled by MemWrite
        - MemWrite = 1 to write the data to memory
        - ALUControl = 010 to add the base address and offset
        - RegWrite = 0, because nothing should be written to the register file
        - NOTE: data is still read from the address given to the data memory but that this ReadData is ignored because RegWrite = 0
- extend the datapath to handle the R-type instructions add, sub, and, or, and slt
    - read two registers from the register file, perform some ALU operation on them, and write the result back to a third register file
    - differ only in the specific ALU operation
    - can all be handled with the same hardware, using different ALUControl signals
    - register file reads two registers
    - ALU performs an operation on these two registers
    - ALU always received its SrcB operand from the sign-extended immediate (SignImm)
    - add a multiplexer to choose SrcB from either the register file RD2 port or SignImm
    - multiplexer is controlled by a new signal, ALUSrc
    - ALUSrc is
        - 0 for R-type instructions to choose SrcB from the register file
        - 1 for lw and sw to choose SignImm
    - useful to enhance the datapath's capabilities by adding a multiplexer to choose inputs from several possibilities

    - add another multiplexer to choose between ReadData and ALUResult because R-type instructions write the ALUResult to the register file
        - output Result
        - controlled by another new signal, MemtoReg
            - 0 for R-type instructions to choose Result from the ALUResult
            - 1 for lw to choose ReadData
            - don't care about the value for sw because sw does not write to the register file
    - add a third multiplexer to choose WriteReg from the appropriate field of the instruction because the register is specified by the rd field, Instr15:11, for R-type instructions
        - multiplexer is controlled by RegDst
            - 1 for R-type instructions to choose WriteReg from the rd field, Instr15:11
            - 0 for lw to choose the rt field, Instr20:16
            - don't care about the value of RegDst for sw because sw does not write to the register file
- extend the datapath to handle beq
    - beq compares two registers
        - if equal it takes the branch by adding the branch offset to the program counter
            - offset is a positive or negative number, stored in the imm field of the instruction, Instr15:0
            - offset indicates the number of instructions to branch past
            - immediate must be sign-extended and multiplied by 4 to get the new program counter value: PC'=PC+4+SignImm x 4
    - next PC value for a taken branch, PCBranch, is computed by shifting SignImm left by 2 bits, then adding it to PCPlus4
        - left shift by 2 is an easy way to multiply by 4, because a shift by a constant amount involves just wires
    - two registers are compared by computing SrcA–SrcB using the ALU
        - if ALUResult is 0, as indicated by the Zero flag from the ALU, the registers are equal
    - add a multiplexer to choose PC' from either PCPlus4 or PCBranch
        - PCBranch is selected if the instruction is a branch and the Zero flag is asserted
        - Branch is 1 for beq and 0 for other instructions
    - ALUControl = 110 so the ALU performs a subtraction
    - ALUSrc=0 to choose SrcB from the register file
    - RegWrite and MemWrite are 0, because a branch does not write to the register file or memory
    - don't care about the values of RegDst and MemtoReg because the register file is not written

Single-Cycle Control

- computes the control signals based on the opcode and funct fields of the instruction, Instr31:26 and Instr5:0
- most of the control information comes from the opcode
    - R-type instructions also use the funct field to determine the ALU operation
- simplify design by factoring the control unit into two blocks of combinational logic
- main decoder computes most of the outputs from the opcode
    - also determines a 2-bit ALUOp signal
- ALU decoder uses this ALUOp signal in conjunction with the funct field to compute ALUControl

```
ALUOp encoding
ALUOp   Meaning
00      add
01      subtract
10      look at funct field
11      n/a
```

- truth table can use don't care's X1 and 1X instead of 01 and 10 to simplify the logic because ALUOp is never 11
- ALU should add or subtract when ALUOp is 00 or 01
- decoder examines the funct field to determine the ALUControl when ALUOp is 10
- for the implemented R-type instructions
    - first two bits of the funct field are always 10
    - may ignore them to simplify the decoder
- control signals for each instruction were described as we built the datapath

```
ALU decoder truth table
fill in
```

- all R-type instructions use the same main decoder values
    - differ only in the ALU decoder output
- for instructions that do not write to the register file (e.g., sw and beq), the RegDst and MemtoReg control signals are don't cares (X)
    - address and data to the register write port do not matter because RegWrite is not asserted
    - logic for the decoder can be designed using your favorite techniques for combinational logic design
- TODO: more here

7.6 HDL Representation

Single-Cycle Processor

Single-Cycle MIPS Processor

Controller

Main Decoder

ALU Decoder

Datapath

Generic Building Blocks

Register File

Adder

Left Shift

Sign Extension

Resettable Flip-Flop

2:1 Multiplexer

Testbench

MIPS Testbench

MIPS Top-Level Module

MIPS Data Memory

MIPS Instruction Memory

7.7 Exceptions

- unexpected (exceptional) changes in the control flow of a program
- two types considered
  - undefined instructions
  - arithmetic overflow

- when an exception takes place the processor copies the PC to the EPC register and stores a code in the Cause register indicating the source of the exception
    - causes include 0x28 for undefined instructions and 0x30 for overflow
- processor jumps to the exception handler at memory address 0x80000180
- exception handler is code that responds to the exception (part of the operating system)
- exception registers are part of Coprocessor 0, a portion of the MIPS processor that is used for system functions
    - Coprocessor 0 defines up to 32 special-purpose registers, including Cause and EPC
- exception handler may use the mfc0 (move from coprocessor 0) instruction to copy these special-purpose registers into a general-purpose register in the register file
    - Cause register is Coprocessor 0 register 13, and EPC is register 14
- must add EPC and Cause registers to the datapath and extend the PCSrc multiplexer to accept the exception handler address to handle exceptions
    - two new registers have write enables, EPCWrite and CauseWrite, to store the PC and exception cause when an exception takes place
    - cause is generated by a multiplexer that selects the appropriate code for the exception
- ALU must also generate an overflow signal
- must add a way to select the Coprocessor 0 registers and write them to the register file to support the mfc0 instruction
- mfc0 instruction specifies the Coprocessor 0 register by Instr15:11
- add another input to the MemtoReg multiplexer to select the value from Coprocessor 0
- controller receives the overflow flag from the ALU
- generates three new control signals
    - one to write the EPC
    - second to write the Cause register
    - third to select the Cause
- also includes two new states to support the two exceptions and another state to handle mfc0
- if the controller receives an undefined instruction (one that it does not know how to handle), it proceeds to S12, saves the PC in EPC, writes 0x28 to the Cause register, and jumps to the exception handler
- if the controller detects arithmetic overflow on an add or sub instruction, it proceeds to S13, saves the PC in EPC, writes 0x30 in the Cause register, and jumps to the exception handler
- NOTE: when an exception occurs, the instruction is discarded and the register file is not written
- when an mfc0 instruction is decoded, the processor goes to S14 and writes the appropriate Coprocessor 0 register to the main register file

7.8 Advanced Microarchitecture

- high-performance microprocessors use a wide variety of techniques to run programs faster

- time required to run a program is proportional to the period of the clock and to the number of clock cycles per instruction (CPI)
  - to increase performance need to speed up the clock and/or reduce the CPI
- many existing speedup techniques
  - focus on the concepts
  - see Computer Architecture - Hennessy & Patterson for details
- advances in CMOS manufacturing reduce transistor dimensions by 30% in each direction every 2 to 3 years
  - doubles the number of transistors that can fit on a chip
- smaller transistors are faster and generally consume less power.
- even if the microarchitecture does not change, the clock frequency can increase because all the gates are faster
- smaller transistors enable placing more transistors on a chip
- microarchitects use the additional transistors to build more complicated processors or to put more processors on a chip
- power consumption increases with the number of transistors and the speed at which they operate
  - now an essential concern
- designers have must juggle trade-offs among speed, power, and cost for chips with billions of transistors in some of the most complex systems that humans have ever built

Deep Pipelines

Aside from advances in manufacturing, the

- easiest way to speed up the clock is to chop the pipeline into more stages
  - each stage can run faster because it contains less logic
- consider a classic five-stage pipeline (10 to 20 stages are now commonly used, more?)
- maximum number of pipeline stages is limited by
  - pipeline hazards
  - sequencing overhead
  - cost
- longer pipelines introduce more dependencies
  - solved by
    - forwarding
    - stalls, which increase the CPI
- pipeline registers between each stage have sequencing overhead from their setup time and clk-to-Q delay (as well as clock skew)
- sequencing overhead makes adding more pipeline stages give diminishing returns
- adding more stages increases the cost because of the extra pipeline registers and hardware required to handle hazards

Branch Prediction

- ideal pipelined processor would have a CPI of 1
- branch misprediction penalty is a major reason for increased CPI
- pipelines get deeper = branches are resolved later in the pipeline
    - branch misprediction penalty gets larger
    - all the instructions issued after the mispredicted branch must be flushed
- most pipelined processors use a branch predictor to guess whether the branch should be taken
- some branches occur when a program reaches the end of a loop (e.g., a for or while statement) and branches back to repeat the loop
- loops tend to be executed many times, so these backward branches are usually taken
- static branch prediction
    - simplest form of branch prediction checks the direction of the branch and predicts that backward branches should be taken
    - called static because it does not depend on the history of the program
- forward branches are difficult to predict without knowing more about the specific program
    - most processors use dynamic branch predictors, which use the history of program execution to guess whether a branch should be taken
    - maintain a table of the last several hundred (or thousand) branch instructions that the processor has executed
    - table, sometimes called a branch target buffer, includes the destination of the branch and a history of whether the branch was taken
- one-bit dynamic branch predictor
    - remembers whether the branch was taken the last time and predicts that it will do the same thing the next time
    - while the loop is repeating, it remembers that the beq was not taken last time and predicts that it should not be taken next time
    - correct prediction until the last branch of the loop, when the branch does get taken
    - if the loop is run again, the branch predictor remembers that the last branch was taken
        - incorrectly predicts that the branch should be taken when the loop is first run again
    - mispredicts the first and last branches of a loop
- 2-bit dynamic branch predictor
    - solves 1-bit problem by having four states
        - strongly taken
        - weakly taken
        - weakly not taken
        - strongly not taken
    - when the loop is repeating, it enters the "strongly not taken" state and predicts that the branch should not be taken next time

    - correct until the last branch of the loop, which is taken and moves the predictor to the "weakly not taken" state
    - when the loop is first run again, the branch predictor correctly predicts that the branch should not be taken and reenters the "strongly not taken" state
    - mispredicts only the last branch of a loop
- branch predictors may be used to track even more history of the program to increase the accuracy of predictions
- good branch predictors achieve better than 90% accuracy on typical programs
- branch predictor operates in the Fetch stage of the pipeline so that it can determine which instruction to execute on the next cycle
    - when it predicts that the branch should be taken, the processor fetches the next instruction from the branch destination stored in the branch target buffer
    - by keeping track of both branch and jump destinations in the branch target buffer, the processor can also avoid flushing the pipeline during jump instructions

Superscalar Processor

- multiple copies of the datapath hardware to execute multiple instructions simultaneously
- two-way superscalar processor
    - fetches and executes two instructions per cycle
- datapath fetches two instructions at a time from the instruction memory
- has a six-ported register file to read four source operands and write two results back in each cycle
- also contains two ALUs and a two-ported data memory to execute the two instructions at the same time
- designers commonly refer to the reciprocal of the CPI as the instructions per cycle, or IPC
- executing many instructions simultaneously is difficult because of dependencies
- parallelism comes in temporal and spatial forms
    - pipelining is a case of temporal parallelism
    - multiple execution units is a case of spatial parallelism
    - superscalar processors exploit both forms of parallelism to squeeze out performance far exceeding that of our single-cycle and multicycle processors
- commercial processors may be three-, four-, or even six-way super-scalar
    - must handle control hazards such as branches as well as data hazards
    - real programs have many dependencies, so wide superscalar processors rarely fully utilize all of the execution units
    - large number of execution units and complex forwarding networks consume vast amounts of circuitry and power

Out-of-Order Processor

- looks ahead across many instructions to issue, or begin executing, independent instructions as rapidly as possible to cope with the problem of dependencies
- instructions can be issued in a different order than that written by the programmer as long as dependencies are honored so that the program produces the intended result
- two-way superscalar out-of-order processor
  - processor can issue up to two instructions per cycle from anywhere in the program, as long as dependencies are
- dependence of add on lw by way of $t0 is a read after write (RAW) hazard
  - add must not read $t0 until after lw has written it
  - type of dependency we are accustomed to handling in the pipelined processor
  - inherently limits the speed at which the program can run, even if infinitely many execution units are available
- dependence of sw on or by way of $t3 and of and on sub by way of $t0 are RAW dependencies
- dependence between sub and add by way of $t0 is called a write after read (WAR) hazard or an antidependence
  - sub must not write $t0 before add reads $t0, so that add receives the correct value according to the original order of the program
  - WAR hazards could not occur in the simple MIPS pipeline, but they may happen in an out-of-order processor if the dependent instruction (in this case, sub) is moved too early
  - a WAR hazard is not essential to the operation of the program
    - merely an artifact of the programmer's choice to use the same register for two unrelated instructions
    - MIPS architecture only has 32 registers, so sometimes the programmer is forced to reuse a register and introduce a hazard just because all the other registers are in use
- a third type of hazard is called write after write (WAW) or an output dependence
  - occurs if an instruction attempts to write a register after a subsequent instruction has already written it
  - results in the wrong value being written to the register
  - WAW hazards are not essential either
    - artifacts caused by the programmer's using the same register for two unrelated instructions
  - squashing the add - program could eliminate a WAW hazard by discarding the result of the add instead of writing it to $t0
- out-of-order processors use a table to keep track of instructions waiting to issue
  - table, sometimes called a scoreboard, contains information about the dependencies
  - size of the table determines how many instructions can be considered for issue

    - on each cycle, the processor examines the table and issues as many instructions as it can, limited by the dependencies and by the number of execution units (e.g., ALUs, memory ports) that are available
- instruction level parallelism (ILP) is the number of instructions that can be executed simultaneously for a particular program and microarchitecture
    - theoretical studies have shown that the ILP can be quite large for out-of-order microarchitectures with perfect branch predictors and enormous numbers of execution units
    - practical processors seldom achieve an ILP greater than 2 or 3, even with six-way superscalar datapaths with out-of-order execution

Register Renaming

- out-of-order processors use a technique called register renaming to eliminate WAR hazards
    - adds some nonarchitectural renaming registers to the processor
    - a MIPS processor might add 20 renaming registers, called $r0-$r19
    - programmer cannot use these registers directly, because they are not part of the architecture
    - processor is free to use them to eliminate hazards

Single Instruction Multiple Data

- (SIMD) a single instruction acts on multiple pieces of data in parallel
- common application of SIMD is to perform many short arithmetic operations at once, especially for graphics processing
    - also called packed arithmetic
- performing packed arithmetic requires modifying the ALU to eliminate carries between the smaller data elements
- short data elements often appear in graphics processing
    - when the components from four adjacent pixels are packed into a 32-bit word, the processing can be performed four times faster
- SIMD instructions are even more helpful for 64-bit architectures, which can pack eight 8-bit elements, four 16-bit elements, or two 32-bit elements into a single 64-bit word
- also used for floating-point computations
    - four 32-bit single-precision floating-point values can be packed into a single 128-bit word

Multithreading

- adding more execution units to a superscalar or out-of-order processor gives diminishing returns
- memory is much slower than the processor
    - most loads and stores access a smaller and faster memory, called a cache

    - when the instructions or data are not available in the cache, the processor may stall for 100 or more cycles while retrieving the information from the main memory
- multithreading is a technique that helps keep a processor with many execution units busy even if the ILP of a program is low or the program is stalled waiting for memory
- computers can run multiple processes simultaneously
    - each process consists of one or more threads that also run simultaneously
    - degree to which a process can be split into multiple threads that can run simultaneously defines its level of thread level parallelism (TLP)
- in a conventional processor, the threads only give the illusion of running simultaneously
    - threads actually take turns being executed on the processor under control of the OS
    - when one thread's turn ends, the OS saves its architectural state, loads the architectural state of the next thread, and starts executing that next thread
    - called context switching
    - as long as the processor switches through all the threads fast enough, the user perceives all of the threads as running at the same time
- a multithreaded processor contains more than one copy of its architectural state, so that more than one thread can be active at a time
    - if we extended a MIPS processor to have four program counters and 128 registers, four threads could be available at one time
    - if one thread stalls while waiting for data from main memory, the processor could context switch to another thread without any delay, because the program counter and registers are already available
    - if one thread lacks sufficient parallelism to keep all the execution units busy, another thread could issue instructions to the idle units
- multithreading does not improve the performance of an individual thread, because it does not increase the ILP
    - it does improve the overall throughput of the processor, because multiple threads can use processor resources that would have been idle when executing a single thread
    - relatively inexpensive to implement, because it replicates only the PC and register file, not the execution units and memories

Homogeneous Multiprocessors

- multiprocessor system consists of multiple processors and a method for communication between the processors
- homogeneous multiprocessing (symmetric multiprocessing (SMP))
    - two or more identical processors share a single main memory
    - may be separate chips or multiple cores on the same chip
- modern processors have enormous numbers of transistors available
    - using them to increase the pipeline depth or to add more execution units to a superscalar processor gives little performance benefit and is wasteful of power

- around 2005, computer architects made a major shift to building multiple copies of the processor on the same chip
  - copies are called cores
- can be used to run more threads simultaneously or to run a particular thread faster
  - threads are divided up among the processors
  - running a particular thread faster is much more challenging
    - programmer must divide the existing thread into multiple threads to execute on each processor
    - difficult when the processors need to communicate with each other
- one of the major challenges for computer designers and programmers is to effectively use large numbers of processor cores
- other forms of multiprocessing
  - heterogeneous multiprocessing
    - (asymmetric multiprocessors) use separate specialized microprocessors for separate tasks
  - clusters
    - each processor has its own local memory system
    - can also refer to a group of PCs connected together on the network running software to jointly solve a large problem

Heterogeneous Multiprocessors

- advantages of homogeneous multiprocessors
  - relatively simple to design because the processor can be designed once and then replicated multiple times to increase performance
  - programming for and executing code on a homogeneous multiprocessor is relatively straightforward because any program can run on any processor in the system and achieve approximately the same performance
- continuing to add more and more cores is not guaranteed to provide continued performance improvement
- as of 2012, consumer applications employed only 2–3 threads on average at any given time
  - enough to keep dual- and quad-core systems busy
  - programs need start incorporating significantly more parallelism
  - continuing to add more cores beyond this point will provide diminishing benefits
- general purpose processors are generally not the most power efficient option for performing a given operation because they are designed to provide good average performance they
- energy inefficiency is especially important in highly power-constrained portable environments
- heterogeneous multiprocessors aim to address these issues by incorporating different types of cores and/or specialized hardware in a single system
  - each application uses those resources that provide the best performance, or power-performance ratio

- heterogeneous systems can take a number of forms
    - can incorporate cores with different microarchitectures that have different power, performance, and area tradeoffs
        - could include both simple single-issue in-order cores and more complex superscalar out-of-order cores
        - cores may all use the same ISA, which allows an application to run on any of the cores, or may employ different ISAs, which can allow further tailoring of a core to a given task
- other heterogeneous systems can include a mix of traditional cores and specialized hardware
    - floating-point coprocessors are an early example of this
    - early microprocessors did not have space for floating-point hardware on the main chip
    - today's microprocessors include one or more floating-point units on chip and are now beginning to include other types of specialized hardware
        - AMD and Intel both have processors that incorporate a graphics processing unit (GPU) or FPGA and one or more traditional x86 cores on the same chip
- heterogeneous systems
    - add complexity, both in terms of designing the different heterogeneous elements and in terms of the additional programming effort to decide when and how to make use of the varying resources
- homogeneous multiprocessors are good for situations, like large data centers, that have lots of thread level parallelism available
- heterogeneous systems are good for cases that have more varying workloads and limited parallelism

7.9 Real-World Perspective: x86 Microarchitecture

- x86 processors evolved through progressively faster and more complicated microarchitectures
- Intel invented the first single-chip microprocessor, the 4-bit 4004, in 1971 as a flexible controller for a line of calculators
    - contained 2300 transistors manufactured on a 12-mm2 sliver of silicon in a process with a 10-µm feature size and operated at 750 kHz
- 4004 inspired the 8-bit 8008, then the 8080, which eventually evolved into the 16-bit 8086 in 1978 and the 80286 in 1982
- in 1985, Intel introduced the 80386, which extended the 8086 architecture to 32 bits and defined the x86 architecture
- 80386 is a multicycle processor with a 32-bit datapath
    - some of the control signals are generated using a microcode PLA that steps through the various states of the control FSM
    - memory management unit in the upper right controls access to the external memory

- the 80486 dramatically improved performance using pipelining
  - added an on-chip floating-point unit
    - previous Intel processors either sent floating-point instructions to a separate coprocessor or emulated them in software
  - too fast for external memory to keep up, so it incorporated an 8-KB cache onto the chip to hold the most commonly used instructions and data Chapter 8 describes caches in more detail and revisits the cache systems on Intel x86 processors.
- Pentium processor is a superscalar processor capable of executing two instructions simultaneously
  - Intel switched to the name Pentium instead of 80586 because AMD was becoming a serious competitor selling interchangeable 80486 chips, and part numbers cannot be trademarked
  - uses separate instruction and data caches
  - uses a branch predictor to reduce the performance penalty for branches
- Pentium Pro, Pentium II, and Pentium III processors all share a common out-of-order microarchitecture, code-named P6
  - complex x86 instructions are broken down into one or more micro-ops similar to MIPS instructions
  - micro-ops are then executed on a fast out-of-order execution core with an 11-stage pipeline
  - The 32-bit datapath is called the Integer Execution Unit (IEU)
  - floating-point datapath is called the Floating Point Unit (FPU)
  - processor also has a SIMD unit to perform packed operations on short integer and floating-point data
  - a larger portion of the chip is dedicated to issuing instructions out-of-order than to actually executing the instructions
  - the instruction and data caches grew to 16 KB each
  - Pentium III also has a larger but slower 256-KB second-level cache on the same chip
- by the late 1990s, processors were marketed largely on clock speed
- Pentium 4 is another out-of-order processor with a very deep pipeline to achieve extremely high clock frequencies
  - started with 20 stages, and later versions adopted 31 stages to achieve frequencies greater than 3 GHz
  - the chip packs in 42 to 178 million transistors (depending on the cache size), so even the major execution units
  - decoding three x86 instructions per cycle is impossible at such high clock frequencies because the instruction encodings are so complex and irregular
    - the processor predecodes the instructions into simpler micro-ops, then stores the micro-ops in a memory called a trace cache
  - later versions of the Pentium 4 also perform multithreading to increase the throughput of multiple threads

    - reliance on deep pipelines and high clock speed led to extremely high power consumption, sometimes more than 100 W
    - unacceptable in laptops and makes cooling of desktops expensive
- Intel discovered that the older P6 architecture could achieve comparable performance at much lower clock speed and power
- Pentium M uses an enhanced version of the P6 out-of-order microarchitecture with 32-KB instruction and data caches and a 1- to 2-MB second-level cache
- Core Duo is a multicore processor based on two Pentium M cores connected to a shared 2-MB second-level cache
- in 2009 Intel introduced a new microarchitecture code-named Nehalem that streamlines the initial Core microarchitecture
    - includes the Core i3, i5, and i7 series
    - extend the instruction set to 64 bits
    - offer from 2 to 6 cores
    - 3–15 MB of third-level cache
    - built-in memory controller
    - some models include built-in graphics processors, also called graphics accelerators
    - some support "Turbo- Boost" to improve the performance of single-threaded code by turning off the unused cores and raising the voltage and clock frequency of the boosted core
    - some offer "hyperthreading" Intel's term for 2-way multi-threading that doubles the number of cores from the user's perspective

Ch. 8 - Memory and I/O Systems

- computer's ability to solve problems requires memory system and the input/output (I/O) devices
    - monitors
    - keyboards
    - printers
- permits manipulation and observation of results of its computations
- performance depends on the memory system as well as the processor microarchitecture
- early processors were relatively slow, so memory was able to keep up
- processor speed has increased at a faster rate than memory speeds
    - DRAM memories are currently 10 to 100 times slower than processors
- increasing gap between processor and DRAM memory speeds demands increasingly ingenious memory systems to try to approximate a memory that is as fast as the processor
- processor communicates with the memory system over a memory interface
- simple memory interface used in our multicycle MIPS processor
    - processor sends an address over the Address bus to the memory system

    - for a read, MemWrite is 0 and the memory returns the data on the ReadData bus
    - for a write, MemWrite is 1 and the processor sends data to memory on the WriteData bus
- memory systems use a hierarchy of storage to quickly access the most commonly used data while still having the capacity to store large amounts of data
- computer memories are primarily built from dynamic RAM (DRAM) and static RAM (SRAM)
    - ideally fast, large, and cheap
    - in practice, a single memory only has two of these three attributes
    - systems can approximate the ideal by combining a fast small cheap memory and a slow large cheap memory
    - fast memory stores the most commonly used data and instructions, so on average the memory system appears fast
    - large memory stores the remainder of the data and instructions, so the overall capacity is large
    - combination of two cheap memories is much less expensive than a single large fast memory
    - principles extend to using an entire hierarchy of memories of increasing capacity and decreasing speed
- computer memory is generally built from DRAM chips
- in 2012, a typical PC had a main memory consisting of 4 to 8 GB of DRAM, and DRAM cost about $10 per gigabyte (GB)
    - prices have declined at about 25% per year for the last three decades
    - memory capacity has grown at the same rate, so the total cost of the memory in a PC has remained roughly constant
    - DRAM speed has improved by only about 7% per year, whereas processor performance has improved at a rate of 25 to 50% per year
    - DRAM access time is one to two orders of magnitude longer than the processor cycle time (tens of nanoseconds, compared to less than one nanosecond)
- to counteract this trend, computers store the most commonly used instructions and data in a faster but smaller memory, called a cache
    - cache is usually built out of SRAM on the same chip as the processor
    - cache speed is comparable to the processor speed, because SRAM is inherently faster than DRAM, and because the on-chip memory eliminates lengthy delays caused by traveling to and from a separate chip
    - in 2012, on-chip SRAM costs were on the order of $10,000/GB, but the cache is relatively small (kilobytes to several megabytes), so the overall cost is low
    - can store both instructions and data
    - if the processor requests data that is available in the cache, it is returned quickly
        - called a cache hit
    - otherwise, the processor retrieves the data from main memory (DRAM)
        - called a cache miss

    - if the cache hits most of the time, then the processor seldom has to wait for the slow main memory, and the average access time is low
- third level in the memory hierarchy is the hard drive
    - computer systems use the hard drive to store data that does not fit in main memory
    - in 2012, a hard disk drive (HDD), built using magnetic storage, cost less than $0.10/GB and had an access time of about 10 ms
    - hard disk costs have decreased at 60%/year but access times scarcely improved
    - solid state drives (SSDs), built using flash memory technology, are an increasingly common alternative to HDDs
    - SSDs have been used by niche markets for over two decades, and they were introduced into the mainstream market in 2007
    - SSDs overcome some of the mechanical failures of HDDs, but they cost ten times as much at $1/GB (old)
    - hard drive provides an illusion of more capacity than actually exists in the main memory
        - called virtual memory
    - main memory, also called physical memory, holds a subset of the virtual memory
    - main memory can be viewed as a cache for the most commonly used data from the hard drive
- process
    - processor first seeks data in a small but fast cache that is usually located on the same chip
    - if the data is not available in the cache, the processor then looks in main memory
    - if the data is not there either, the processor fetches the data from virtual memory on the large but slow hard disk

8.2 Memory System Performance Analysis

- need quantitative ways to measure the performance of memory systems to evaluate the cost-benefit trade-offs of various alternatives
- memory system performance metrics are miss rate or hit rate and average memory access time

```
miss rate = number of misses / number of total memory accesses = 1 − hit rate
hit rate = number of hits / number of total memory accesses = 1 − miss rate
```

- average memory access time (AMAT) is the average time a processor must wait for memory per load or store instruction
    - processor first looks for the data in the cache
    - if the cache misses, the processor then looks in main memory
    - if the main memory misses, the processor accesses virtual memory on the hard disk

```
AMAT = tcache + MRcache (tMM + MRmm*tVM)
```

- tcache - access time of the cache
- tMM - access time of main memory
- tVM - access time of virtual memory
- MRcache - cache miss rate
- MRMM - main memory miss rate
- Amdahl's Law - says that the effort spent on increasing the performance of a subsystem is worthwhile only if the subsystem affects a large percentage of the overall performance
    - making the memory system ten times faster will not necessarily make a computer program run ten times as fast
    - if 50% of a program's instructions are loads and stores, a tenfold memory system improvement only means a 1.82-fold improvement in program performance

Caches

- holds commonly used memory data
- the number of data words that it can hold is called the capacity, C
- designer must choose what subset of the main memory is kept in the cache
- cache is checked first for data access
- if the cache hits, the data is available immediately
- if the cache misses, the processor fetches the data from main memory and places it in the cache for future use
- cache must replace old data

What Data is Held in the Cache?

- ideal cache would anticipate all of the data needed by the processor and fetch it from main memory ahead of time so that the cache has a zero miss rate
    - impossible to predict with perfect accuracy
    - cache must guess what data will be needed based on the past pattern of memory accesses
    - cache exploits temporal and spatial locality to achieve a low miss rate
        - temporal locality means that the processor is likely to access a piece of data again soon if it has accessed that data recently
            - when the processor loads or stores data that is not in the cache, the data is copied from main memory into the cache
            - subsequent requests for that data hit in the cache

- - - spatial locality means that when the processor accesses a piece of data it is also likely to access data in nearby memory locations
            - when the cache fetches one word from memory, it may also fetch several adjacent words
            - group of words is called a cache block or cache line
            - number of words in the cache block, b, is called the block size
    - cache of capacity C contains B = C/b blocks.
- if a variable is used in a program, the same variable is likely to be used again, creating temporal locality
- if an element in an array is used, other elements in the same array are also likely to be used, creating spatial locality

How is Data Found?

- cache is organized into S sets, each of which holds one or more blocks of data
- relationship between the address of data in main memory and the location of that data in the cache is called the mapping
    - each memory address maps to exactly one set in the cache
    - some of the address bits are used to determine which cache set contains the data
    - if the set contains more than one block, the data may be kept in any of the blocks in the set
- caches are categorized based on the number of blocks in a set
    - direct mapped cache
        - each set contains exactly one block, so the cache has S = B sets
        - a particular main memory address maps to a unique block in the cache
    - N-way set associative cache
        - each set contains N blocks
        - address still maps to a unique set, with S = B/N sets
        - data from that address can go in any of the N blocks in that set
    - fully associative cache
        - has only S = 1 set
        - data can go in any of the B blocks in the set
        - another name for a B-way set associative cache

Direct Mapped Cache

- one block in each set, so it is organized into S = B sets
- hypothetical mapping
    - address in block 0 of main memory maps to set 0 of the cache
    - address in block 1 of main memory maps to set 1 of the cache, and so forth until an address in block B – 1 of main memory maps to block B – 1 of the cache

    - no more blocks of the cache, so the mapping wraps around, such that block B of main memory maps to block 0 of the cache
- each main memory address maps to exactly one set in the cache
- because many addresses map to a single set, the cache must also keep track of the address of the data actually contained in each set
    - least significant bits of the address specify which set holds the data
    - remaining most significant bits are called the tag and indicate which of the many possible addresses is held in that set
- sometimes (e.g. when the computer first starts up) the cache sets contain no data at all
    - cache uses a valid bit for each set to indicate whether the set holds meaningful data
    - if the valid bit is 0, the contents are meaningless.
- cache is constructed as an eight-entry SRAM
    - each entry, or set, contains one line consisting of 32 bits of data, 27 bits of tag, and 1 valid bit
    - cache is accessed using the 32-bit address
    - two least significant bits, the byte offset bits, are ignored for word accesses
    - next three bits, the set bits, specify the entry or set in the cache
    - load instruction reads the specified entry from the cache and checks the tag and valid bits
    - if the tag matches the most significant 27 bits of the address and the valid bit is 1, the cache hits and the data is returned to the processor
    - otherwise cache misses and the memory system must fetch the data from main memory
    - when two recently accessed addresses map to the same cache block, a conflict occurs, and the most recently accessed address evicts the previous one from the block
    - direct mapped caches have only one block in each set, so two addresses that map to the same set always cause a conflict

Multi-way Set Associative Cache

- reduces conflicts by providing N blocks in each set where data mapping to that set might be found
- each memory address still maps to a specific set, but it can map to any one of the N blocks in the set
- direct mapped cache is another name for a one-way set associative cache
- N is also called the degree of associativity of the cache
- each set contains two ways or degrees of associativity
- each way consists of a data block and the valid and tag bits
- cache reads blocks from both ways in the selected set and checks the tags and valid bits for a hit
- if a hit occurs in one of the ways, a multiplexer selects data from that way

- generally have lower miss rates than direct mapped caches of the same capacity, because they have fewer conflicts
- set associative caches are usually slower and somewhat more expensive to build because of the output multiplexer and additional comparators
- raises the question of which way to replace when both ways are full
- most commercial systems use set associative caches

Fully Associative Cache

- contains a single set with B ways, where B is the number of blocks
- memory address can map to a block in any of these ways
- another name for a B-way set associative cache with one set
- tend to have the fewest conflict misses for a given cache capacity, but they require more hardware for additional tag comparisons
- best suited to relatively small caches because of the large number of comparators

Block Size

- to exploit spatial locality, a cache uses larger blocks to hold several consecutive words
- advantage of a block size greater than one
  - when a miss occurs and the word is fetched into the cache, the adjacent words in the block are also fetched
  - subsequent accesses are more likely to hit because of spatial locality
  - a large block size means that a fixed-size cache will have fewer blocks
    - may lead to more conflicts, increasing the miss rate
    - takes more time to fetch the missing cache block after a miss, because more than one data word is fetched from main memory
  - time required to load the missing block into the cache is called the miss penalty
  - if the adjacent words in the block are not accessed later, the effort of fetching them is wasted
- principle of using larger block sizes to exploit spatial locality also applies to associative caches

Putting it All Together

- caches are organized as two-dimensional arrays
- rows are called sets
- columns are called ways
- each entry in the array consists of a data block and its associated valid and tag bits
- characterized by
  - capacity C
  - block size b (and number of blocks, B = C/b)

    - number of blocks in a set (N)

```
Organization        Number of Ways (N)      Number of Sets (S)
Direct Mapped       1                       B
Set Associative     1 < N < B               B / N
Fully Associative   B                       1
```

- each address in memory maps to only one set but can be stored in any of the ways
- Cache capacity, associativity, set size, and block size are typically powers of 2
    - makes the cache fields (tag, set, and block offset bits) subsets of the address bits
- increasing the associativity N usually reduces the miss rate caused by conflicts
    - higher associativity requires more tag comparators
- increasing the block size b takes advantage of spatial locality to reduce the miss rate
    - decreases the number of sets in a fixed sized cache and therefore could lead to more conflicts
    - also increases the miss penalty

What Data is Replaced?

- direct mapped cache
    - each address maps to a unique block and set
    - if a set is full when new data must be loaded, the block in that set is replaced with the new data
- set associative and fully associative caches
    - cache must choose which block to evict when a cache set is full
    - temporal locality suggests that the best choice is to evict the least recently used block, because it is least likely to be used again soon
    - most associative caches have a least recently used (LRU) replacement policy
- two-way set associative cache
    - use bit, U, indicates which way within a set was least recently used
    - each time one of the ways is used, U is adjusted to indicate the other way
- set associative caches with more than two ways
    - tracking the least recently used way becomes complicated
    - the ways are often divided into two groups and U indicates which group of ways was least recently used for simplification
    - upon replacement, the new block replaces a random block within the least recently used group
    - such a policy is called pseudo-LRU and is good enough in practice

Advanced Cache Design

- modern systems use multiple levels of caches to decrease memory access time

Multiple-Level Caches

- large caches are beneficial because they are more likely to hold data of interest and therefore have lower miss rates
  - tend to be slower than small ones
- modern systems often use at least two levels of caches
- first-level (L1) cache is small enough to provide a one- or two-cycle access time
- second-level (L2) cache is also built from SRAM but is larger, and therefore slower, than the L1 cache
- processor first looks for the data in the L1 cache
  - if the L1 cache misses, the processor looks in the L2 cache
  - if the L2 cache misses, the processor fetches the data from main memory
- many modern systems add even more levels of cache to the memory hierarchy, because accessing main memory is so slow

Reducing Miss Rate

- can be reduced by changing capacity, block size, and/or associativity
- understand the causes of the misses
  - compulsory
    - the first request to a cache block is called a compulsory miss, because the block must be read from memory regardless of the cache design
  - capacity
    - occur when the cache is too small to hold all concurrently used data
  - conflict
    - caused when several addresses map to the same set and evict blocks that are still needed
- changing cache parameters can affect one or more type of cache miss
  - increasing capacity can reduce conflict and capacity misses, but it does not affect compulsory misses
  - increasing block size could reduce compulsory misses (due to spatial locality) but might actually increase conflict misses (because more addresses would map to the same set and could conflict)
- best to benchmark while changing parameters
- miss rate can also be decreased by using larger block sizes that take advantage of spatial locality
  - as block size increases, the number of sets in a fixed-size cache decreases, increasing the probability of conflicts
  - for small caches, such as the 4-KB cache, increasing the block size beyond 64 bytes increases the miss rate because of conflicts
  - for larger caches, increasing the block size beyond 64 bytes does not change the miss rate

    - large block sizes might still increase execution time because of the larger miss penalty, the time required to fetch the missing cache block from main memory

Write Policy

- memory stores, or writes, follow a similar procedure as loads
- on memory store, the processor checks the cache
    - if the cache misses, the cache block is fetched from main memory into the cache, and then the appropriate word in the cache block is written
    - if the cache hits, the word is simply written to the cache block
- write-through cache
    - data written to a cache block is simultaneously written to main memory
- write-back cache
    - dirty bit (D) is associated with each cache block
    - D is 1 when the cache block has been written and 0 otherwise
    - dirty cache blocks are written back to main memory only when they are evicted from the cache
- write-through cache requires no dirty bit but usually requires more main memory writes than a write-back cache.
- modern caches are usually write-back because main memory access time is so large

The Evolution of MIPSCaches

- major trends are the introduction of multiple levels of cache, larger cache capacity, and increased associativity
- driven by the growing disparity between CPU frequency and main memory speed and the decreasing cost of transistors
- growing difference between CPU and memory speeds necessitates a lower miss rate to avoid the main memory bottleneck, and the decreasing cost of transistors allows larger cache sizes

8.4 Virtual Memory

- most modern computer systems use a hard drive made of magnetic or solid state storage as the lowest level in the memory hierarchy
- hard drive is large and cheap but slow
    - provides a much larger capacity than is possible with a cost-effective main memory (DRAM)
    - if a significant fraction of memory accesses involve the hard drive, performance is dismal

- hard disk
    - contains one or more rigid disks or platters, each of which has a read/write head on the end of a long triangular arm
    - head moves to the correct location on the disk and reads or writes data magnetically as the disk rotates
    - head takes several milliseconds to seek the correct location on the disk, which is fast from a human perspective but millions of times slower than the processor
- hard drive inexpensively gives the illusion of a very large memory while still providing the speed of faster memory for most accesses
- larger memory is called virtual memory
- smaller main memory is called physical memory
- programs can access data anywhere in virtual memory, so they must use virtual addresses that specify the location in virtual memory
- physical memory holds a subset of most recently accessed virtual memory
- physical memory acts as a cache for virtual memory
- most accesses hit in physical memory at the speed of DRAM, yet the program has the capacity of the larger virtual memory
- virtual memory divided into virtual pages, typically 4 KB in size.
- physical memory divided into physical pages of the same size
- virtual page may be located in physical memory (DRAM) or on the hard drive
- process of determining the physical address from the virtual address is called address translation
- if the processor attempts to access a virtual address that is not in physical memory, a page fault occurs, and the operating system loads the page from the hard drive into physical memory
- to avoid page faults caused by conflicts, any virtual page can map to any physical page
    - physical memory behaves as a fully associative cache for virtual memory
- in a conventional fully associative cache, every cache block has a comparator that checks the most significant address bits against a tag to determine whether the request hits in the block
- in an analogous virtual memory system, each physical page would need a comparator to check the most significant virtual address bits against a tag to determine whether the virtual page maps to that physical page
- realistic virtual memory system has so many physical pages that providing a comparator for each page would be excessively expensive
    - virtual memory system uses a page table to perform address translation
    - page table contains an entry for each virtual page, indicating its location in physical memory or that it is on the hard drive
    - each load or store instruction requires a page table access followed by a physical memory access

    - page table access translates the virtual address used by the program to a physical address
    - physical address is then used to actually read or write the data
    - page table is usually so large that it is located in physical memory
    - each load or store involves two physical memory accesses
        - page table access
        - data access
    - translation lookaside buffer (TLB) caches the most commonly used page table entries to speed address translation

```
Analogous cache and virtual memory terms
Cache           Virtual Memory
Block           Page
Block size      Page size
Block offset    Page offset
Miss            Page fault
Tag             Virtual page number
```

Address Translation

- virtual memory uses virtual addresses to allow access to large memory
- must translate these virtual addresses to either find the address in physical memory or take a page fault and fetch the data from the hard drive
- most significant bits of the virtual or physical address specify the virtual or physical page number
- least significant bits specify the word within the page and are called the page offset
- MIPS accommodates 32-bit addresses
    - With a 2-GB=2^31-byte virtual memory, only the least significant 31 virtual address bits are used
    - 32nd bit is always 0
    - with a 128-MB = 227-byte physical memory, only the least significant 27 physical address bits are used
    - the upper 5 bits are always 0
- there are 2^31/2^12 = 2^19 virtual pages and 2^27/2^12 = 2^15 physical pages because the page size is 4 KB = 212 bytes
    - virtual and physical page numbers are 19 and 15 bits
- physical memory can only hold up to 1/16th of the virtual pages at any given time
    - remainder of virtual pages are kept on the hard drive.
- only the page number needs to be translated to obtain the physical address from the virtual address
- least significant 12 bits indicate the page offset and require no translation

- upper 19 bits of the virtual address specify the virtual page number (VPN) and are translated to a 15-bit physical page number (PPN)

The Page Table

- processor uses a page table to translate virtual addresses to physical addresses
- page table contains an entry for each virtual page
- entry contains a physical page number and a valid bit
  - if the valid bit is 1, the virtual page maps to the physical page specified in the entry
  - otherwise, the virtual page is found on the hard drive
- stored in physical memory because page table is large
- page table is indexed with the virtual page number (VPN)
- page table can be stored anywhere in physical memory, at the discretion of the OS
  - processor typically uses a dedicated register, called the page table register, to store the base address of the page table in physical memory
- to perform a load or store, the processor must first translate the virtual address to a physical address and then access the data at that physical address
- processor extracts the virtual page number from the virtual address and adds it to the page table register to find the physical address of the page table entry
  - then reads this page table entry from physical memory to obtain the physical page number
  - if the entry is valid, it merges this physical page number with the page offset to create the physical address
  - it reads or writes data at this physical address
- each load or store involves two physical memory accesses because the page table is stored in physical memory

The Translation Lookaside Buffer

- temporal and spatial locality of data accesses and the large page size mean that many consecutive loads or stores are likely to reference the same page
- processor can keep the last several page table entries in a small cache called a translation lookaside buffer (TLB)
- processor "looks aside" to find the translation in the TLB before having to access the page table in physical memory
- vast majority of accesses hit in the TLB, avoiding the time-consuming page table reads from physical memory
- organized as a fully associative cache and typically holds 16 to 512 entries
  - each entry holds a virtual page number and its corresponding physical page number
  - accessed using the virtual page number
  - if the TLB hits, it returns the corresponding physical page number
  - otherwise, the processor must read the page table in physical memory

    - designed to be small enough that it can be accessed in less than one cycle
    - typically have a hit rate of greater than 99%
    - decreases the number of memory accesses required for most load or store instructions from two to one

Memory Protection

- equally important reason to use virtual memory is to provide protection between concurrently running programs
- all concurrently running programs are simultaneously present in physical memory
    - programs should be protected from each other so that no program can crash or hijack another program
    - no program should be able to access another program's memory without permission
- virtual memory systems provide memory protection by giving each program its own virtual address space
- each program can use as much memory as it wants in that virtual address space, but only a portion of the virtual address space is in physical memory at any given time
- each program can use its entire virtual address space without having to worry about where other programs are physically located
- a program can access only those physical pages that are mapped in its page table
    - program cannot accidentally or maliciously access another program's physical pages because they are not mapped in its page table
    - in some cases multiple programs access common instructions or data
        - OS adds control bits to each page table entry to determine which programs can write to the shared physical pages

Replacement Policies

- virtual memory systems use write-back and an approximate least recently used (LRU) replacement policy
    - write-through policy would be impractical
    - store instructions would operate at the speed of the hard drive instead of the speed of the processor (milliseconds instead of nanoseconds)
- with writeback, physical page is written back to the hard drive only when it is evicted from physical memory
- writing the physical page back to the hard drive and reloading it with a different virtual page is called paging
    - hard drive in a virtual memory system is sometimes called swap space
    - processor pages out one of the least recently used physical pages when a page fault occurs, then replaces that page with the missing virtual page

- each page table entry contains two additional status bits to support these replacement policies
  - dirty bit D
    - 1 if any store instructions have changed the physical page since it was read from the hard drive
    - when a physical page is paged out, needs to be written back to the hard drive only if its dirty bit is 1
    - otherwise, hard drive already holds an exact copy of the page
  - use bit U
    - 1 if the physical page has been accessed recently
    - exact LRU replacement would be impractically complicated
    - OS approximates LRU replacement by periodically resetting all the use bits in the page table
    - when a page is accessed, its use bit is set to 1
    - on page fault, the OS finds a page with U=0 to page out of physical memory
    - does not necessarily replace the least recently used page, just one of the least recently used pages

Multilevel Page Tables

- page tables can be broken up into multiple (usually two) levels to conserve physical memory
- first-level page table is always kept in physical memory
  - indicates where small second-level page tables are stored in virtual memory
- second-level page tables each contain the actual translations for a range of virtual pages
  - if a particular range of translations is not actively used, the corresponding second-level page table can be paged out to the hard drive so it does not waste physical memory
- in a two-level page table, the virtual page number is split into two parts
  - page table number
    - indexes the first-level page table, which must reside in physical memory
    - first-level page table entry gives the base address of the second-level page table or indicates that it must be fetched from the hard drive when V is 0
  - page table offset
    - indexes the second-level page table
  - remaining 12 bits of the virtual address are the page offset for a page size of 212 = 4 KB
- two-level page table requires a fraction of the physical memory needed to store the entire page table (2 MB)
- drawback of a two-level page table is that it adds yet another memory access for translation when the TLB misses

8.5 I/O Introduction

Input/Output (I/O) systems are

- used to connect a computer with external devices called peripherals
- devices typically include keyboards, monitors, printers, and wireless networks in PCs/laptops
- includes specialized elements for embedded systems
- processor accesses an I/O device using the address and data busses in the same way that it accesses memory
- a portion of the address space is dedicated to I/O devices rather than memory
- each I/O device is assigned one or more memory addresses in this range
    - a store to the specified address sends data to the device
    - a load receives data from the device. This method of communicating with I/O devices is called memory-mapped I/O
- a load or store may access either memory or an I/O device with memory-mapped I/O
- write-enabled registers hold the values written to the I/O devices
- software that communicates with an I/O device is called a device driver
- writing a device driver requires detailed knowledge about the I/O device hardware
- addresses associated with I/O devices are often called I/O registers because they may correspond with physical registers in the I/O device

8.6 Embedded I/O Systems

- embedded systems use a processor to control interactions with the physical environment
- typically built around microcontroller units (MCUs) which combine a microprocessor with a set of easy-to-use peripherals such as general-purpose digital and analog I/O pins, serial ports, timers, etc
- microcontrollers are generally inexpensive and are designed to minimize system cost and size by integrating most of the necessary components onto a single chip
- most are smaller and lighter than a dime, consume milliwatts of power, and range in cost from a few dimes up to several dollars
- microcontrollers are classified by the size of data that they operate upon. 8-bit microcontrollers are the smallest and least expensive, while 32-bit microcontrollers provide more memory and higher performance

PIC32MX675F512H Microcontroller

- based around a 32-bit MIPS processor

- processor connects via 32-bit busses to Flash memory containing the program and SRAM containing data

- has 512 KB of Flash and 64 KB of RAM
    - other flavors with 16 to 512 KB of FLASH and 4 to 128 KB of RAM are available at various price points
- high performance peripherals such as USB and Ethernet also communicate directly with the RAM through a bus matrix
- lower performance peripherals including serial ports, timers, and A/D converters share a separate peripheral bus
- chip also contains timing generation circuitry to produce the clocks and voltage sensing circuitry to detect when the chip is powering up or about to lose power
- all addresses used by the programmer are virtual
- MIPS architecture offers a 32-bit address space to access up to 232 bytes = 4 GB of memory, but only a small fraction of that memory is actually implemented on the chip
- relevant sections from address 0xA0000000-0xBFC02FFF include the RAM, the Flash, and the special function registers (SFRs) used to communicate with peripherals
- NOTE: there is an additional 12 KB of Boot Flash that typically performs some initialization and then jumps to the main program in Flash memory
- on reset, the program counter is initialized to the start of the Boot Flash at 0xBFC00000
- pins include power and ground, clock, reset, and many I/O pins that can be used for general-purpose I/O and/or special-purpose peripherals
- PIC32 and external circuitry are mounted on a printed circuit board
    - may be an actual product (e.g. a toaster controller), or a development board facilitating easy access to the chip during testing
- LTC1117-3.3 regulator accepts an input of 4.5-12 V (e.g., from a wall transformer, battery, or external power supply) and drops it down to a steady 3.3 V required at the power supply pins
- multiple VDD and GND pins to reduce power supply noise by providing a low-impedance path
- bypass capacitors provide a reservoir of charge to keep the power supply stable in the event of sudden changes in current demand
- bypass capacitor also connects to the VCORE pin, which serves an internal 1.8 V voltage regulator
- typically draws 1–2 mA/MHz of current from the power supply

- external oscillator running up to 50 MHz can be connected to the clock pin
- microcontroller can be programmed to use an internal oscillator running at 8.00 MHz ± 2%
  - much less precise in frequency, but may be good enough
- peripheral bus clock, PBCLK, for the I/O devices (such as serial ports, A/D converter, timers) typically runs at a fraction (e.g., half) of the speed of the main system clock SYSCLK
- clocking scheme can be set by configuration bits in the MPLAB development software
- good to provide a reset button to put the chip into a known initial state of operation
  - consists of a push-button switch and a resistor connected to the reset pin, MCLR
  - reset pin is active-low, indicating that the processor resets if the pin is at 0
  - when the button is not pressed, the switch is open and the resistor pulls the reset pin to 1, allowing normal operation
  - when the button is pressed, the switch closes and pulls the reset pin down to 0, forcing the processor to reset
  - also resets itself automatically when power is turned on
- the easiest way to program the microcontroller is with a Microchip In Circuit Debugger (ICD) 3
  - known as a puck
  - allows the programmer to communicate with the PIC32 from a PC to download code and to debug the program
  - connects to a USB port on the PC and to a six-pin RJ-11 modular connector on the PIC32 development board
  - RJ-11 connector is the familiar connector used in United States telephone jacks
  - ICD3 communicates with the PIC32 over a 2-wire In-Circuit Serial Programming interface with a clock and a bidirectional data pin
  - free MPLAB Integrated Development Environment (IDE) to write your programs in assembly language or C, debug them in simulation, and download and test them on a development board by means of the ICD
- most of the pins of the PIC32 microcontroller default to function as general-purpose digital I/O pins
- same pins are shared for special-purpose I/O functions such as serial ports, analog-to-digital converter inputs, etc., that become active when the corresponding peripheral is enabled
- responsibility of the programmer to use each pin for only one purpose at any given time

General-Purpose Digital I/O

- general-purpose I/O (GPIO) pins are used to read or write digital signals
  - schematic tells the programmer the function of each pin and the hardware designer what connections to physically make
- PIC32 organizes groups of GPIOs into ports that are read and written together
  - each port may have up to 16 GPIO pins, although the PIC32 doesn't have enough pins to provide that many signals for all of its ports
  - each port is controlled by two registers
    - TRISx
      - determine whether the pin of the port is an input or an output
    - PORTx
      - registers indicate the value read from an input or driven to an output
    - x is a letter (A-G) indicating the port of interest
    - 16 least significant bits of each register correspond to the sixteen pins of the GPIO port
    - when a given bit of the TRISx register is 0, the pin is an output, and when it is 1, the pin is an input
    - good to leave unused GPIO pins as inputs (their default state) so that they are not inadvertently driven to problem values
    - each register is memory-mapped to a word in the Special Function Registers portion of virtual memory (0xBF80000-BF8FFFF)
    - programmer can access them by name rather than having to look up the address
    - can write the entire register at once or access individual bits
- each port also has corresponding SET and CLR registers that can be written with a mask indicating which bits to set or clear
- number of GPIO pins available depends on the size of the package
- logic levels are LVCMOS-compatible
- input pins expect logic levels of VIL = 0.15VDD and VIH = 0.8VDD, or 0.5 V and 2.6 V assuming VDD of 3.3V
- output pins produce VOL of 0.4 V and VOH of 2.4 V as long as the output current Iout does not exceed 7 mA

Serial I/O

If a microcontroller

- must break the message into multiple smaller transmissions if microcontroller needs to send more bits than the number of free GPIO pins
  - can send one bit
    - serial I/O
    - popular because it uses few wires and is fast enough for many applications

        - multiple standards for serial I/O have been established and the PIC32 has dedicated hardware to easily send data via these standards
    - can send several bits
        - parallel I/O
- Serial Peripheral Interface (SPI) protocol
- Universal Asynchronous Receiver/Transmitter (UART) protocol
- other common serial standards
    - Inter-Integrated Circuit (I2C)
        - 2-wire interface with a clock and a bidirectional data pin
        - used similarly to SPI
    - Universal Serial Bus (USB)
    - Ethernet. I2C (pronounced "I squared C") is a

Serial Peripheral Interface (SPI)

- easy to use and relatively fast
- physical interface consists of three pins
    - Serial Clock (SCK)
    - Serial Data Out (SDO)
    - Serial Data In (SDI)
- connects a master device to a slave device
- master produces the clock
    - initiates communication by sending a series of clock pulses on SCK
    - if it wants to send data to the slave, it puts the data on SDO, starting with the most significant bit
    - slave may simultaneously respond by putting data on the master's SDI
- PIC32 has up to four SPI ports unsurprisingly named SPI1-SPI4
    - each can operate as a master or slave
- PIC program must first configure the port
    - it can then write data to a register
    - data is transmitted serially to the slave
    - data received from the slave is collected in another register
    - when the transmission is complete, the PIC32 can read the received data
- each SPI port is associated with four 32-bit registers
    - SPIxCON
        - SPI1CON is the control register for SPI port 1
        - used to turn the SPI ON and set attributes such as the number of bits to transfer and the polarity of the clock
    - SPIxSTAT
    - SPIxBRG
    - SPIxBUF

    - all have a default value of 0 on reset
- STAT is the status register indicating whether the receive register is full
- refer to datasheet
- serial clock can be configured to toggle at half the peripheral clock rate or less
- SPI has no theoretical limit on the data rate and can readily operate at tens of MHz on a printed circuit board
    - may experience noise problems running above 1 MHz using wires on a breadboard
- BRG is the baud rate register that sets the speed of SCK relative to the peripheral clock (see formula)
- BUF is the data buffer
    - data written to BUF is transferred over the SPI port on the SDO pin, and the received data from the SDI pin can be found by reading BUF after the transfer is complete
- to prepare the SPI in master mode
    - turn it OFF by clearing bit 15 of the CON register (the ON bit) to 0
    - clear anything that might be in the receive buffer by reading the BUF register
    - set the desired baud rate by writing the BRG register
    - put the SPI in master mode by setting bit 5 of the CON register (MSTEN) to 1
    - set bit 8 of the CON register (CKE) so that SDO is centered on the rising edge of the clock
    - turn the SPI back ON by setting the ON bit of the CON register
- to send data to the slave
    - write the data to the BUF register
    - data will be transmitted serially
    - slave will simultaneously send data back to the master
    - wait until bit 11 of the STAT register (the SPIBUSY bit) becomes 0 indicating that the SPI has completed its operation
    - data received from the slave can be read from BUF
- SPI port on a PIC32 is highly configurable so that it can talk to a wide variety of serial devices
    - leads to the possibility of incorrectly configuring the port and getting garbled data transmission
    - relative timing of the clock and data signals are configured with three CON register bits called CKP, CKE, and SMP
    - by default, these bits are 0, but the timing uses CKE = 1
    - master changes SDO on the falling edge of SCK, so the slave should sample the value on the rising edge using a positive edge-triggered flip-flop
    - master expects SDI to be stable around the rising edge of SCK, so the slave should change it on the falling edge,
    - MODE32 and MODE16 bits of the CON register specify whether a 32- or 16-bit word should be sent

    - both default to 0 indicating an 8-bit transfer
- sometimes it is necessary to change the configuration bits to communicate with a device that expects different timing
- when CKP = 1, SCK is inverted. When CKE = 0, the clocks toggle half a cycle earlier relative to the data
- when SAMPLE = 1, the master samples SDI half a cycle later (and the slave should ensure it is stable at that time)
- be aware that different SPI products may use different names and polarities for these options
    - check the waveforms carefully for your device
    - can also be helpful to examine SCK, SDO, and SDI on an oscilloscope if you are having communication difficulties

---

Universal Asynchronous Receiver Transmitter (UART)

- serial I/O peripheral that communicates between two systems without sending a clock
    - systems must agree in advance about what data rate to use and must each locally generate its own clock
    - system clocks may have a small frequency error and an unknown phase relationship, the UART manages reliable asynchronous communication
- used in protocols such as RS-232 and RS-485
- computer serial ports use the RS- 232C standard, introduced in 1969 by the Electronic Industries Association
    - standard originally envisioned connecting Data Terminal Equipment (DTE) such as a mainframe computer to Data Communication Equipment (DCE) such as a modem
    - relatively slow compared to SPI, the standards have been around for so long that they remain important today
- DTE sends data to the DCE over the TX line and receives data back over the RX line
    - line idles at a logic '1' when not in use
    - each character is sent as a start bit (0), 7-8 data bits, an optional parity bit, and one or more stop bits (1's)
    - detects the falling transition from idle to start to lock on to the transmission at the appropriate time
    - seven data bits is sufficient to send an ASCII character, eight bits are normally used because they can convey an arbitrary byte of data
    - an additional bit, the parity bit, can also be sent, allowing the system to detect if a bit was corrupted during transmission
    - can be configured as even or odd
    - even parity means that the parity bit is chosen such that the total collection of data and parity has an even number of 1's
        - parity bit is the XOR of the data bits

    - receiver can then check if an even number of 1's was received and signal an error if not
    - odd parity is the reverse and is hence the XNOR of the data bits
    - common choice is 8 data bits, no parity, and 1 stop bit, making a total of 10 symbols to convey an 8-bit character of information
    - signaling rates are referred to in units of baud rather than bits/sec
    - systems must be configured for the appropriate baud rate and number of data, parity, and stop bits or the data will be garbled
    - hassle, especially for nontechnical users, which is one of the reasons that the Universal Serial Bus (USB) has replace UARTs in personal computer systems
- typical baud rates include 300, 1200, 2400, 9600, 14400, 19200, 38400, 57600, and 115200
- lower rates were used in the 1970's and 1980's for modems that sent data over the phone lines as a series of tones
- in contemporary systems, 9600 and 115200 are two of the most common baud rates
- 9600 is encountered where speed doesn't matter, and 115200 is the fastest standard rate, though still slow compared to other modern serial I/O standards
- RS-232 standard defines several additional signals
- Request to Send (RTS) and Clear to Send (CTS) signals can be used for hardware handshaking
    - can be operated in either of two modes
    - flow control mode
        - DTE clears RTS to 0 when it is ready to accept data from the DCE
    - DCE clears CTS to 0 when it is ready to receive data from the DTE
    - some datasheets use an overbar to indicate that they are active-low
    - in the older simplex mode, the DTE clears RTS to 0 when it is ready to transmit
    - DCE replies by clearing CTS when it is ready to receive the transmission
- some systems, especially those connected over a telephone line, also use Data Terminal Ready (DTR), Data Carrier Detect (DCD), Data Set Ready (DSR), and Ring Indicator (RI) to indicate when equipment is connected to the line
- original standard recommended a massive 25-pin DB-25 connector, but PCs streamlined to a male 9-pin DE-9 connector with the pinout
- cable wires normally connect straight across
- when directly connecting two DTEs, a null modem cable shown in Figure 8.42(c) may be needed to swap RX and TX and complete the handshaking
- some connectors are male and some are female
- it can take a large box of cables and a certain amount of guess-work to connect two systems over RS-232, again explaining the shift to USB
- embedded systems typically use a simplified 3- or 5-wire setup consisting of GND, TX, RX, and possibly RTS and CTS
- RS-232 represents a 0 electrically with 3 to 15 V and a 1 with –3 to –15 V (bipolar signaling)

- transceiver converts the digital logic levels of the UART to the positive and negative levels expected by RS-232 and also provides electrostatic discharge protection to protect the serial port from damage when the user plugs in a cable
- MAX3232E is a transceiver compatible with both 3.3 and 5 V digital logic
    - contains a charge pump that, in conjunction with external capacitors, generates ± 5 V outputs from a single low-voltage power supply
- PIC32 has six UARTs named U1-U6. As with SPI, the PIC program must first configure the port
    - unlike SPI, reading and writing can occur independently because either system may transmit without receiving and vice versa
    - each associated with five 32-bit registers
        - UxMODE
        - UxSTA (status)
        - UxBRG
        - UxTXREG
        - UxRXREG
    - U1MODE is the mode register for UART1
    - mode register is used to configure the UART and the STA register is used to check when data is available
    - BRG register is used to set the baud rate
    - data is transmitted or received by writing the TXREG or reading the RXREG
- MODE register defaults to 8 data bits, 1 stop bit, no parity, and no RTS/CTS flow control, so for most applications the programmer is only interested in bit 15, the ON bit, which enables the UART
- STA register contains bits to enable the transmit and receive pins and to check if the transmit and receive buffers are full
- after setting the ON bit of the UART, the programmer must also set the UTXEN and URXEN bits (bits 10 and 12) of the STA register to enable these two pins
- UTXBF (bit 9) indicates that the transmit buffer is full
- URXDA (bit 0) indicates that the receive buffer has data available
- STA register also contains bits to indicate parity and framing errors
    - a framing error occurs if the start or stop bits aren't found at the expected time
- 16-bit BRG register is used to set the baud rate to a fraction of the peripheral bus clock

formula here

- it is sometimes impossible to reach exactly the target baud rate
- as long as the frequency error is much less than 5%, the phase error between the transmitter and receiver will remain small over the duration of a 10-bit frame so data is received properly
- system will then resynchronize at the next start bit

- transmit data, wait until STA.UTXBF is clear indicating that the transmit buffer has space available, and then write the byte to TXREG
- to receive data, check STA.URXDA to see if data has arrived, and then read the byte from the RXREG
- communicating with the serial port from a C program on a PC is a bit of a hassle because serial port driver libraries are not standardized across operating systems
- programming environments such as Python, Matlab, or LabVIEW make serial communication painless

Timers

- embedded systems commonly need to measure time
- PIC32 has five 16-bit timers on board
    - timer 1 is called a Type A timer that can accept an asynchronous external clock source, such as a 32 KHz watch crystal
    - timers 2/3 and 4/5 are Type B timers
        - run synchronously off the peripheral clock and can be paired (e.g., 2 with 3) to form 32-bit timers for measuring long periods of time
- each timer is associated with three 16-bit registers
    - TxCON
    - TMRx
    - PRx
    - e.g. T1CON is the control register for Timer 1
        - CON is the control register
    - TMR contains the current time count
    - PR is the period register
- when a timer reaches the specified period, it rolls back to 0 and sets the TxIF bit in the IFS0 interrupt flag register
    - program can poll this bit to detect overflow
    - it can also generate an interrupt
- each timer acts as a 16-bit counter accumulating ticks of the internal peripheral clock (20 MHz in our example) by default
- prescaling by k:1 causes the timer to only count once every k ticks
    - useful to generate longer time intervals, especially when the peripheral clock is running fast
- another timer feature is gated time accumulation
    - timer only counts while an external pin is high
    - allows the timer to measure the duration of an external pulse
    - enabled using the CON register.

Interrupts

- timers are often used in conjunction with interrupts so that a program may go about its usual business and then
- interrupt requests occur when a hardware event occurs, such as a timer overflowing, a character being received over a UART, or certain GPIO pins toggling
  - each type of interrupt request sets a specific bit in the Interrupt Flag Status (IFS) registers
  - processor then checks the corresponding bit in the Interrupt Enable Control (IEC) registers
  - if the bit is set, the microcontroller should respond to the interrupt request by invoking an interrupt service routine (ISR)
  - ISR is a function with void arguments that handles the interrupt and clears the bit of the IFS before returning
  - PIC32 interrupt system supports single and multi-vector modes
  - single-vector mode
    - all interrupts invoke the same ISR, which must examine the CAUSE register to determine the reason for the interrupt (if multiple types of interrupts may occur) and handle it accordingly
  - multi-vector mode
    - each type of interrupt calls a different ISR
    - MVEC bit in the INTCON register determines the mode
  - MIPS interrupt system must be enabled with the ei instruction before it will accept any interrupts
- PIC32 also allows each interrupt source to have a configurable priority and subpriority
  - priority is in the range of 0–7, with 7 being highest
  - higher priority interrupt will preempt an interrupt presently being handled
  - subpriority is in the range of 0–3
  - if two events with the same priority are simultaneously pending, the one with the higher subpriority will be handled first
  - subpriority will not cause a new interrupt to preempt an interrupt of the same priority presently being serviced
  - priority and subpriority of each event is configured with the IPC registers
- each interrupt source has a vector number in the range of 0–63
- ISR function declaration is tagged by two special _ *attribute* _ directives indicating the priority level and vector number
  - compiler uses these attributes to associate the ISR with the appropriate interrupt request
  - MPLAB C Compiler For PIC32 MCUs User's Guide has more information about writing interrupt service routines

Analog I/O

- embedded systems need analog inputs and outputs to interface with the world
- use analog-to-digital-converters (ADCs) to quantize analog signals into digital values, and digital-to-analog-converters (DACs) to do the reverse
- converters are characterized by their resolution, dynamic range, sampling rate, and accuracy
- sampling rates are also listed in samples per second (sps), where 1 sps = 1 Hz
- relationship between the analog input voltage Vin(t) and the digital sample X[n] is

```
X[n] = 2^N * (Vin(t) - Vref- / Vref+ - Vref-)
n = t/fs
```

- DAC might have N = 16-bit resolution over a full-scale output range of Vref = 2.56 V

```
Vout(t) = X[n]/2^N * Vref
```

- many microcontrollers have built-in ADCs of moderate performance
- for higher performance (e.g., 16-bit resolution or sampling rates in excess of 1 MHz), it is often necessary to use a separate ADC connected to the microcontroller
- fewer microcontrollers have built-in DACs, so separate chips may also be used
  - often simulate analog outputs using a technique called pulse-width modulation (PWM)

A/D Conversion

- PIC32 has a 10-bit ADC with a maximum speed of 1 million samples/sec (Msps)
- can connect to any of 16 analog input pins via an analog multiplexer
- analog inputs are called AN0-15 and share their pins with digital I/O port RB
- Vref+ is the analog VDD pin and Vref– is the analog GND pin
- programmer must initialize the ADC, specify which pin to sample, wait long enough to sample the voltage, start the conversion, wait until it finishes, and read the result
- highly configurable
  - automatically scan through multiple analog inputs at programmable intervals and to generate interrupts upon completion
- controlled by a host of registers
  - AD1CON1-3
  - AD1CHS
  - AD1PCFG
  - AD1CSSL
  - ADC1BUF0-F
  - AD1CON1 is the primary control register

        - has an ON bit to enable the ADC, a SAMP bit to control when sampling and conversion takes place, and a DONE bit to indicate conversion complete
    - AD1CON3 has ADCS[7:0] bits that control the speed of the A/D conversion
    - AD1CHS is the channel selection register specifying the analog input to sample
    - AD1PCFG is the pin configuration register
        - when a bit is 0, the corresponding pin acts as an ana- log input
        - when it is a 1, the pin acts as a digital input
    - ADC1BUF0 holds the 10-bit conversion result
    - other registers are not required in our simple example
- sampling time is the amount of time necessary for a new input to stabilize before its conversion can start
    - as long as the resistance of the source being sampled is less than 5 KΩ, the sampling time can be as small as 132 ns, which is only a small number of clock cycles

D/A Conversion

- PIC32 has no built-in DAC
- some DACs accept the N-bit digital input on a parallel interface with N wires, while others accept it over a serial interface such as SPI
- some DACs require both positive and negative power supply voltages, while others operate off of a single supply
- some support a flexible range of supply voltages, while others demand a specific voltage
- input logic levels should be compatible with the digital source
- ome DACs produce a voltage output proportional to the digital input, while others produce a current output
    - an operational amplifier may be needed to convert this current to a voltage in the desired range
- Analog Devices AD558 8-bit parallel DAC and the Linear Technology LTC1257 12-bit serial DAC
    - both produce voltage outputs
    - run off a single 5-15 V power supply
    - use VIH = 2.4 V such that they are compatible with 3.3 V outputs from the PIC32
    - come in DIP packages that make them easy to breadboard
    - easy to use
    - AD558
        - produces an output on a scale of 0-2.56 V
        - consumes 75 mW
        - comes in a 16-pin package
        - has a 1 µs settling time permitting an output rate of 1 Msample/sec
    - LTC1257
        - produces an output on a scale of 0-2.048V

        - consumes less than 2 mW
        - comes in an 8-pin package
        - has a 6 µs settling time
        - its SPI operates at a maximum of 1.4 MHz
- Texas Instruments is another leading ADC and DAC manufacturer

Pulse-Width Modulation

- alternative for a digital system to generate an analog output
- periodic output is pulsed high for part of the period and low for the remainder
    - duty cycle is the fraction of the period for which the pulse is high
    - average value of the output is proportional to the duty cycle For example, if the output swings between 0 and 3.3 V and has a duty cycle of 25%, the average value will be 0.25 × 3.3 = 0.825 V.
    - low-pass filtering a PWM signal eliminates the oscillation and leaves a signal with the desired average value
- PIC32 contains five output compare modules, OC1-OC5, that each, in conjunction with Timer 2 or 3, can produce PWM outputs
    - each output compare module is associated with three 32-bit registers
        - OCxCON
        - OCxR
        - OCxRS
        - CON is the control register
        - OCM bits of the CON register should be set to 1102 to activate PWM mode, and the ON bit should be enabled
        - the output compare uses Timer2 in 16-bit mode, but the OCTSEL and OC32 bits can be used to select Timer3 and/or 32-bit mode
        - in PWM mode, RS sets the duty cycle, the timer's period register PR sets the period, and OCxR can be ignored

Other Microcontroller Peripherals

- microcontrollers frequently interface with other external peripherals.
    - charactermode liquid crystal displays (LCDs)
    - VGA monitors
    - Bluetooth wireless links
    - motor control
- standard communication interfaces
    - USB
    - Ethernet

Character LCDs

- small liquid crystal display capable of showing one or a few lines of text
- commonly used in the front panels of appliances such as cash registers, laser printers, and fax machines that need to display a limited amount of information
- easy to interface with a microcontroller over parallel, RS-232, or SPI interfaces
- e.g. Crystalfontz America This section gives an example of interfacing a PIC32 microcontroller to a character LCD over an 8-bit parallel interface
  - interface is compatible with the industry-standard HD44780 LCD controller originally developed by Hitachi
  - to initialize the LCD, the PIC32 must write a sequence of instructions to the LCD
    - wait > 15000 µs after VDD is applied
    - write 0x30 to set 8-bit mode
    - wait > 4100 µs
    - write 0x30 to set 8-bit mode again
    - wait > 100 µs
    - write 0x30 to set 8-bit mode yet again
    - wait until busy flag is clear
    - write0x3Ctoset2linesand5×8dotfont
    - wait until busy flag is clear
    - write 0x08 to turn display OFF
    - wait until busy flag is clear
    - write 0x01 to clear the display
    - wait > 1530 µs
    - write 0x06 to set entry mode to increment cursor after each character
    - wait until busy flag is clear
    - write 0x0C to turn display ON with no cursor
  - to write text to the LCD, the microcontroller can send a sequence of ASCII characters
    - may also send the instructions 0x01 to clear the display or 0x02 to return to the home position in the upper left

VGA Monitor

- Video Graphics Array (VGA) monitor standard was introduced in 1987 for the IBM PS/2 computers, with a 640 × 480 pixel resolution on a cathode ray tube (CRT) and a 15-pin connector conveying color information with analog voltages
- modern LCD monitors have higher resolution but remain backward compatible with the VGA standard
- in a cathode ray tube, an electron gun scans across the screen from left to right exciting fluorescent material to display an image
- color CRTs use three different phosphors for red, green, and blue, and three electron beams

- strength of each beam determines the intensity of each color in the pixel
- at the end of each scanline, the gun must turn off for a horizontal blanking interval to return to the beginning of the next line
- after all of the scanlines are complete, the gun must turn off again for a vertical blanking interval to return to the upper left corner
- process repeats about 60–75 times per second to give the visual illusion of a steady image.
- in a 640 × 480 pixel VGA monitor refreshed at 59.94 Hz, the pixel clock operates at 25.175 MHz, so each pixel is 39.72 ns wide
- full screen can be viewed as 525 horizontal scanlines of 800 pixels each, but only 480 of the scanlines and 640 pixels per scan line actually convey the image, while the remainder are black
- scanline begins with a back porch, the blank section on the left edge of the screen
  - then contains 640 pixels, followed by a blank front porch at the right edge of the screen and a horizontal sync (hsync) pulse to rapidly move the gun back to the left edge Figure 8.51(a) shows the timing of each of these portions of the scanline, beginning with the active pixels. The
  - entire scan line is 31.778 µs long
- the vertical direction, the screen starts with a back porch at the top, followed by 480 active scan lines, followed by a front porch at the bottom and a vertical sync (vsync) pulse to return to the top to start the next frame
- new frame is drawn 60 times per second
  - NOTE: the time units are now scan lines rather than pixel clocks
- higher resolutions use a faster pixel clock, up to 388 MHz at 2048×1536 @ 85 Hz
- horizontal timing involves a front porch of 24 clocks, hsync pulse of 136 clocks, and back porch of 160 clocks
- vertical timing involves a front porch of 3 scan lines, vsync pulse of 6 lines, and back porch of 29 lines
- pixel information is conveyed with three analog voltages for red, green, and blue
- each voltage ranges from 0–0.7 V, with more positive indicating brighter
- voltages should be 0 during the front 2 and back porches
- cable can also provide an IC serial link to configure the monitor
- video signal must be generated in real time at high speed, which is difficult on a microcontroller but easy on an FPGA
- simple black and white display could be produced by driving all three color pins with either 0 or 0.7 V using a voltage divider connected to a digital output pin
- color monitor, on the other hand, uses a video DAC with three separate D/A converters to independently drive the three color pins
  - DAC receives 8 bits of R, G, and B from the FPGA
  - also receives a SYNC_b signal that is driven active low whenever HSYNC or VSYNC are asserted

    - video DAC produces three output currents to drive the red, green, and blue analog lines, which are normally 75 Ω transmission lines parallel terminated at both the video DAC and the monitor
    - RSET resistor sets the scale of the output current to achieve the full range of color
    - clock rate depends on the resolution and refresh rate
        - it may be as high as 330 MHz with a fast-grade ADV7125JSTZ330 model DAC

Bluetooth Wireless Communication

- standards now available for wireless communication
    - Wi-Fi
    - ZigBee
    - Bluetooth standards are elaborate and require sophisticated integrated circuits, but a
- growing assortment of modules abstract away the complexity and give the user a simple interface for wireless communication
    - BlueSMiRF
        - easy-to- use Bluetooth wireless interface that can be used instead of a serial cable
- Bluetooth is a wireless standard developed by Ericsson in 1994 for low-power, moderate speed, communication over distances of 5–100 meters, depending on the transmitter power level
    - commonly used to connect an earpiece to a cellphone or a keyboard to a computer
    - unlike infrared communication links, it does not require a direct line of sight between devices
- Bluetooth operates in the 2.4 GHz unlicensed industrial-scientific-medical (ISM) band
- defines 79 radio channels spaced at 1 MHz intervals starting at 2402 MHz
- hops between these channels in a pseudo-random pattern to avoid consistent interference with other devices like wireless phones operating in the same band
- Bluetooth transmitters are classified at one of three power levels, which dictate the range and power consumption
    - in the basic rate mode, it operates at 1 Mbit/sec using Gaussian frequency shift keying (FSK)
    - in ordinary FSK, each bit is conveyed by transmitting a frequency of fc ± fd, where fc is the center frequency of the channel and fd is an offset of at least 115 kHz
    - abrupt transition in frequencies between bits consumes extra bandwidth
    - Gaussian FSK, the change in frequency is smoothed to make better use of the spectrum

Motor Control

- application of microcontrollers is to drive actuators such as motors

- three general types of motors
  - DC motors
    - require a high drive current, so a powerful driver such as an H-bridge must be connected between the microcontroller and the motor
    - also require a separate shaft encoder if the user wants to know the current position of the motor
  - servo motors
    - accept a pulse-width modulated signal to specify their position over a limited range of angles
    - easy to interface, but are not as powerful and are not suited to continuous rotation
  - stepper motors
    - accept a sequence of pulses, each of which rotates the motor by a fixed angle called a step
    - more expensive and still need an H-bridge to drive the high current, but the position can be precisely controlled
- draw a substantial amount of current and may introduce glitches on the power supply that disturb digital logic
  - one way to reduce this problem is to use a different power supply or battery for the motor than for the digital logic

DC Motors

Servo Motor

Stepper Motor

8.7 PC I/O Systems

- wide variety of I/O protocols for purposes
  - memory
  - disks
  - networking
  - internal expansion cards
  - external devices
- I/O standards have evolved to offer very high performance and to make it easy for users to add devices.
  - attributes come at the expense of complexity in the I/O protocols
- motherboard contains
  - DRAM memory module slots
  - wide variety of I/O device connectors
  - power supply connector
  - voltage regulators

    - capacitors
    - DRAM modules are connected over a DDR3 interface
    - external peripherals such as keyboards or webcams can be attached over USB
    - high-performance expansion cards such as graphics cards connect over the PCI Express x16 slot
    - lower-performance cards can use PCI Express x1 or the older PCI slots
    - connects to the network using the Ethernet jack (or wi-fi)
    - hard disks are connected to a SATA port
- one of the major advances in PC I/O standards has been the development of high-speed serial links
    - most I/O was previously built around parallel links consisting of a wide data bus and a clock signal
    - the difference in delay among the wires in the bus set a limit to how fast the bus could run
    - busses connected to multiple devices suffer from transmission line problems such as reflections and different flight times to different loads
    - noise can also corrupt the data
    - oint-to-point serial links eliminate many of these problems
    - data is usually transmitted on a differential pair of wires
    - external noise that affects both wires in the pair equally is unimportant
    - transmission lines are easy to properly terminate, so reflections are small
    - no explicit clock is sent
        - clock is recovered at the receiver by watching the timing of the data transitions
    - high-speed serial link design is a specialized subject
    - good interfaces can run faster than 10 Gb/s over copper wires and even faster along optical fibers

USB

- Universal Serial Bus (USB), developed by Intel, IBM, Microsoft, and others, greatly simplified adding peripherals by standardizing the cables and software configuration process
- USB 1.0 was released in 1996
    - cable with four wires
        - 5 V
        - GND
        - differential pair of wires to carry data
    - cable is impossible to plug in backward or upside down
    - operates at up to 12 Mb/s
    - can pull up to 500 mA from the USB port

- USB 2.0 released in 2000
    - upgraded the speed to 480 Mb/s by running the differential wires much faster
    - became practical for attaching webcams and external hard disks
    - Flash memory sticks with a USB interface also replaced floppy disks as a means of transferring files between computers
- USB 3.0 released in 2008
    - further boosted the speed to 5 Gb/s
    - uses the same shape connector
    - cable has more wires that operate at very high speed
    - better suited to connecting high-performance hard disks
    - added a Battery Charging Specification that boosts the power supplied over the port to speed up charging mobile devices

PCI and PCI Express

- Peripheral Component Interconnect (PCI) bus is an expansion bus standard developed by Intel that became widespread around 1994
    - was used to add expansion cards such as extra serial or USB ports, network interfaces, sound cards, modems, disk controllers, or video cards
    - 32-bit parallel bus operates at 33 MHz, giving a bandwidth of 133 MB/s
- demand for PCI expansion cards has steadily declined
    - more standard ports such as Ethernet and SATA are now integrated into the motherboard
- contemporary motherboards often still have a small number of PCI slots, but fast devices like video cards are now connected via PCI Express (PCIe)
- PCI Express slots provide one or more lanes of high-speed serial links
    - PCIe 3.0, each lane operates at up to 8 Gb/s
- most motherboards provide an x16 slot with 16 lanes giving a total of 16 GB/s of bandwidth to data-hungry devices such as video cards

DDR3 Memory

DRAM connects to the microprocessor over a parallel bus. In 2012, the present

- standard as of 2012 is DDR3
    - third generation of double-data rate memory bus operating at 1.5 V
    - typical motherboards now come with two DDR3 channels so they can access two banks of memory modules simultaneously Figure 8.71 shows a 4 GB
- DDR3 dual inline memory module (DIMM)
- as of 2012 DIMMs typically carry 1–16 GB of DRAM
- DRAM presently operates at a clock rate of 100–266 MHz
- DDR3 operates the memory bus at four times the DRAM clock rate

- transfers data on both the rising and falling edges of the clock
- sends 8 words of data for each memory clock. At 64 bits/word, this corresponds to 6.4–17 GB/s of bandwidth
- DRAM latency remains high, with a roughly 50 ns lag from a read request until the arrival of the first word of data

Networking

- connect to the Internet over a network interface running the Transmission Control Protocol and Internet Protocol (TCP/IP)
- physical connection may be an Ethernet cable or a wireless Wi-Fi link
- Ethernet is defined by the IEEE 802.3 standard
    - developed at Xerox Palo Alto Research Center (PARC) in 1974
    - originally operated at 10 Mb/s (called 10 Mbit Ethernet
    - now is commonly found at 100 Mbit (Mb/s) and 1 Gbit (Gb/s) running on Category 5 cables containing four twisted pairs of wires
    - 10 Gbit Ethernet running on fiber optic cables is increasingly popular for servers and other high-performance computing, and 100 Gbit Ethernet is emerging
- Wi-Fi is the popular name for the IEEE 802.11 wireless network standard
    - operates in the 2.4 and 5 GHz unlicensed wireless bands, meaning that the user doesn't need a radio operator's license to transmit in these bands at low power
    - increasing performance comes from advancing modulation and signal processing, multiple antennas, and wider signal bandwidths

SATA

- in 1986 Western Digital introduced the Integrated Drive Electronics (IDE) interface
    - evolved into the AT Attachment (ATA) standard
    - uses a bulky 40 or 80-wire ribbon cable with a maximum length of 18" to send data at 16–133 MB/s
- been supplanted by Serial ATA (SATA)
    - uses high-speed serial links to run at 1.5, 3, or 6 Gb/s over a more convenient 7-conductor cable
- fastest solid-state drives in 2012 approach 500 MB/s of bandwidth, taking full advantage of SATA
- related standard is Serial Attached SCSI (SAS)
    - evolution of the parallel SCSI (Small Computer System Interface) interface
    - offers performance comparable to SATA and supports longer cables
    - common in server computers

Interfacing to a PC

- I/O standards optimized for high performance and ease of attachment but are difficult to implement in hardware
- engineers and scientists often need a way to connect a PC to external circuitry, such as sensors, actuators, microcontrollers, or FPGAs The serial connection described in Section 8.6.3.2 is sufficient for a l
- low-speed connection serial connection to a microcontroller with a UART may be sufficient
- two other options
  - data acquisition systems
  - USB links

Data Acquisition Systems

- Data Acquisition Systems (DAQs) connect a computer to the real world using multiple channels of analog and/or digital I/O
- commonly available as USB devices
- National Instruments (NI)
- high-performance DAQ prices tend to run into the thousands of dollars, mostly because the market is small and has limited competition
- example has two analog channels capable of input and output at 200 ksamples/sec with a 16 bit resolution and ±10V dynamic range
  - channels can be configured to operate as an oscilloscope and signal generator
  - has eight digital input and output lines compatible with 3.3 and 5 V systems
  - generates +5, +15, and –15 V power sup- ply outputs and includes a digital multimeter capable of measuring voltage, current, and resistance
  - can replace an entire bench of test and measurement equipment while simultaneously offering automated data logging
- some DAQs can also be controlled from C programs using the LabWindows environment, from Microsoft .NET applications using the Measurement Studio environment, or from Matlab using the Data Acquisition Toolbox

USB Links

- products contain predeveloped drivers and libraries, allowing the user to easily write a program on the PC that blasts data to and from an FPGA or microcontroller
- FTDI
- example FTDI C232HM-DDHSL USB to Multi-Protocol Synchronous Serial Engine (MPSSE)
  - provides a USB jack at one end
  - an SPI interface operating at up to 30 Mb/s at the other end
  - 3.3 V power and four general purpose I/O pins
  - cable can optionally supply 3.3 V power to the FPGA

    - three SPI pins connect to an FPGA slave device
- PC requires the D2XX dynamically linked library driver to be installed
- can then write a C program using the library to send data over the cable
- FTDI UM232H module (faster connection)
    - links a PC's USB port to an 8-bit synchronous parallel interface operating up to 40 MB/s

8.8 Real-World Perspective: x86 Memory and I/O Systems

- faster processors = elaborate memory hierarchies to keep a steady supply of data and instructions
- x86 also has an unusual programmed I/O system that differs from the more common memory- mapped I/O

x86 Cache Systems

Processor	Year	Frequency (MHz)	Level 1 Data Cache	Level 1 Instruction Cache	Level 2 Cache
80486	1989	25–100	8 KB unified		none on chip
Pentium	1993	60-300	8 KB	8 KB	256 KB - 1 MB on MCM
Pentium Pro	1995	150–200	8 KB	8 KB	256 KB–1 MB on MCM
Pentium II	1997	233-450	16 KB	16 KB	256-512 KB on cartridge
Pentium III	1999	450-1400	16 KB	16 KB	256-512 on chip
Pentium 4	2001	1400-3730	8-16 KB	12 Kop trace cache	256 KB - 2 MB on chip
Pentium M	2003	900-2130	32 KB	32 KB	2 MB shared on chip
Core Duo	2005	1500-2160	32 KB/core	32 KB/core	2 MB shared on chip
Core i7	2009	1600-3600	32 KB/core	32 KB/core	256 KB/core + 4-15 MB L3

- 80386, initially produced in 1985, operated at 16 MHz
  - lacked a cache, so it directly accessed main memory for all instructions and data
  - depending on the speed of the memory, the processor might get an immediate response, or it might have to pause for one or more cycles for the memory to react
  - these cycles are called wait states, and they increase the CPI of the processor
- microprocessor clock frequencies have increased by at least 25% per year since then
- memory latency has barely diminished
- delay from when the processor sends an address to main memory until the memory returns the data can now exceed 100 processor clock cycles
  - caches with a low miss rate are essential to good performance
- 80486 introduced a unified write-through cache to hold both instructions and data
  - Most high-performance computer systems also provided a larger second-level cache on the motherboard using commercially available SRAM chips that were substantially faster than main memory

- Pentium processor introduced separate instruction and data caches to avoid contention during simultaneous requests for data and instructions
    - caches used a write-back policy, reducing the communication with main memory
    - larger second-level cache (typically 256–512 KB) was usually offered on the motherboard
- P6 series of processors (Pentium Pro, Pentium II, and Pentium III) were designed for much higher clock frequencies
    - second-level cache on the motherboard could not keep up
    - moved closer to the processor to improve its latency and throughput
    - Pentium Pro was packaged in a multichip module (MCM) containing both the processor chip and a second-level cache chip
    - processor had separate 8-KB level 1 instruction and data caches
    - these caches were nonblocking, so that the out-of-order processor could continue executing subsequent cache accesses even if the cache missed a particular access and had to fetch data from main memory
    - second-level cache was 256 KB, 512 KB, or 1 MB in size and could operate at the same speed as the processor
    - MCM packaging proved too expensive for high-volume manufacturing
    - Pentium II was sold in a lower-cost cartridge containing the processor and the second-level cache
    - level 1 caches were doubled in size to compensate for the fact that the second-level cache operated at half the processor's speed
    - Pentium III integrated a full-speed second-level cache directly onto the same chip as the processor
    - cache on the same chip can operate at better latency and throughput, so it is substantially more effective than an off-chip cache of the same size
- Pentium 4 offered a nonblocking level 1 data cache
    - switched to a trace cache to store instructions after they had been decoded into micro-ops, avoiding the delay of redecoding each time instructions were fetched from the cache
- Pentium M design was adapted from the Pentium III
    - increased the level 1 caches to 32 KB each and featured a 1- to 2-MB level 2 cache
    - Core Duo contains two modified Pentium M processors and a shared 2-MB cache on one chip
        - shared cache is used for communication between the processors
            - one can write data to the cache
            - other can read it
- Nehalem (Core i3-i7) design adds a third level of cache shared between all of the cores on the die to facilitate sharing information between cores
    - each core has its own 64 KB L1 cache and 256 KB L2 cache, while the shared L3 cache contains 4-8+ MB

x86 Virtual Memory

- x86 processors operate in either real mode or protected mode
  - real mode is backward compatible with the original 8086
    - only uses 20 bits of address, limiting memory to 1 MB, and it does not allow virtual memory
  - protected mode was introduced with the 80286 and extended to 32-bit addresses with the 80386
    - supports virtual memory with 4-KB pages
    - also provides memory protection so that one program cannot access the pages belonging to other programs
    - buggy or malicious program cannot crash or corrupt other programs
    - all modern operating systems now use protected mode
- 32-bit address permits up to 4 GB of memory
- processors since the Pentium Pro have bumped the memory capacity to 64 GB using a technique called physical address extension
- each process uses 32-bit addresses
- virtual memory system maps these addresses onto a larger 36-bit virtual memory space
- uses different page tables for each process, so that each process can have its own address space of up to 4 GB x86 has been upgraded to x86-64 to get around the memory bottleneck
  - offers 64-bit virtual addresses and general-purpose registers
  - only 48 bits of the virtual address are used, providing a 256 terabyte (TB) virtual address space (as of 2012)
  - limit may be extended to the full 64 bits as memories expand, offering a 16 exabyte (EB) capacity
- x86 uses programmed I/O, in which special IN and OUT instructions are used to read and write I/O devices
  - x86 defines 216 I/O ports
  - IN instruction reads one, two, or four bytes from the port specified by DX into AL, AX, or EAX
  - OUT is similar, but writes the port
- connecting a peripheral device to a programmed I/O system is similar to connecting it to a memory-mapped system
- when accessing an I/O port, the processor sends the port number rather than the memory address on the 16 least significant bits of the address bus
- device reads or writes data from the data bus
- major difference is that the processor also produces an M/IO signal
  - when M/IO = 1, the processor is accessing memory
  - when it is 0, the process is accessing one of the I/O devices

  - address decoder must also look at M/IO to generate the appropriate enables for main memory and for the I/O devices
  - I/O devices can also send interrupts to the processor to indicate that they are ready to communicate

Digital System Implementation

74xx Logic

- in the 1970s and 1980s many digital systems were built from simple chips
  - handful of logic gates
  - 7404 chip contains six NOT gates
  - 7408 contains four AND gates
  - 7474 contains two flip-flops
  - chips are collectively referred to as 74xx-series logic
  - obsolete but useful for simple digital systems or class projects
  - commonly sold in 14-pin dual inline packages (DIPs)

Logic Gates

- 74xx-series chips containing basic logic gates
- sometimes called small-scale integration (SSI) chips
- built from a few transistors
- 14-pin packages typically have a notch at the top or a dot on the top left to indicate orientation
- pins are numbered starting with 1 in the upper left and going counterclockwise around the package
- need to receive power (VDD = 5 V) and ground (GND = 0 V) at pins 14 and 7
- number of logic gates on the chip is determined by the number of pins
- NOTE: pins 3 and 11 of the 7421 chip are not connected (NC) to anything
- 7474 flip-flop has the usual D, CLK, and Q terminals
  - also has a complementary output, Q
  - receives asynchronous set (also called preset, or PRE) and reset (also called clear, or CLR) signals
  - active low
    - flop sets when PRE = 0, resets when CLR = 0, and operates normally when PRE = CLR = 1

Other Functions

- 74xx series also includes more complex logic functions
    - called medium-scale integration (MSI) chips
    - larger packages to accommodate more inputs and outputs

Programmable Logic

- programmable logic
    - arrays of circuitry that can be configured to perform specific logic functions
- three forms of programmable logic so far
    - programmable read only memories (PROMs)
    - programmable logic arrays (PLAs)
    - field programmable gate arrays (FPGAs)
- configuration of these chips may be performed by blowing on-chip fuses to connect or disconnect circuit elements
    - called one-time programmable (OTP) logic
        - once a fuse is blown it cannot be restored
    - configuration may be stored in a memory that can be reprogrammed at will
    - eprogrammable logic is convenient in the laboratory because the same chip can be reused during development

PROMs

- can be used as lookup tables
    - 2N-word x M-bit PROM can be programmed to perform any combinational function of N inputs and M outputs

FPGAs

- consist of arrays of configurable logic elements (LEs) connected together with programmable wires
    - also called configurable logic blocks (CLBs)
- LEs contain small lookup tables and flip-flops
- scale gracefully to extremely large capacities, with thousands of lookup tables
- Xilinx and Altera
- lookup tables and programmable wires are flexible enough to implement any logic function
- much less efficient in speed and cost (chip area) than hard-wired versions of the same functions
- often include specialized blocks, such as memories, multipliers, and even entire microprocessors

- design is usually specified with a hardware description language (HDL)
    - some FPGA tools also support schematics
- design is then simulated
- inputs are applied and compared against expected outputs to verify that the logic is correct
- logic synthesis converts the HDL into Boolean functions
- good synthesis tools produce a schematic of the functions
- examine to ensure that the desired logic was produced
- when the synthesis results are good, the FPGA tool maps the functions onto the LEs of a specific chip
- place and route tool determines which functions go in which lookup tables and how they are wired together
- wire delay increases with length, so critical circuits should be placed close together
- timing analysis compares the timing constraints (e.g., an intended clock speed of 100MHz) against the actual circuit delays and reports any errors
- when the design is correct, a file is generated specifying the contents of all the LEs and the programming of all the wires on the FPGA
    - many FPGAs store this configuration information in static RAM that must be reloaded each time the FPGA is turned on
    - can download this information from a computer in the laboratory, or can read it from a nonvolatile ROM when power is first applied
- cost has declined at approximately 30% per year
    - becoming extremely popular

Application-Specific Integrated Circuits

- application-specific integrated circuits (ASICs)
    - chips designed for a particular purpose
    - graphics accelerators
    - network interface chips
    - cell phone chips
- designer places transistors to form logic gates and wires the gates together
- typically several times faster than an FPGA and occupies an order of magnitude less chip area (and hence cost) (hardwired for specific function)
- masks specifying where transistors and wires are located on the chip cost hundreds of thousands of dollars to produce
- if errors are discovered after the ASIC is manufactured, the designer must correct the problem, generate new masks, and wait for another batch of chips to be fabricated
    - suitable only for products that will be produced in large quantities and whose function is well defined in advance

- logic verification is especially important because correction of errors after the masks are produced is expensive
- synthesis produces a netlist consisting of logic gates and connections between the gates
  - gates in this netlist are placed
  - wires are routed between gates
  - masks are generated and used to fabricate the ASIC
- single speck of dust can ruin an ASIC, so the chips must be tested after fabrication
- fraction of manufactured chips that work is called the yield
  - typically 50 to 90%, depending on the size of the chip and the maturity of the manufacturing process

Data Sheets

Logic Families

- series of chips have been manufactured using many different technologies, called logic families
  - offer different speed, power, and logic level trade-offs
  - other chips are usually designed to be compatible with some of these logic families
- advances in bipolar circuits and process technology led to the Schottky (S) and Low-Power Schottky (LS) families
- CMOS circuits became popular as they matured in the 1980s and 1990s because they draw very little power supply or input current
  - High Speed CMOS (HC) and Advanced High Speed CMOS (AHC) families draw almost no static power
  - deliver the same current for HIGH and LOW outputs
  - conform to the "CMOS" logic levels
  - levels are incompatible with TTL circuits
  - motivates the use of High Speed TTL-compatible CMOS (HCT) and Advanced High Speed TTL-compatible CMOS (AHCT)
    - accept TTL input logic levels and generate valid CMOS output logic levels
    - slightly slower than their pure CMOS counterparts
  - all CMOS chips are sensitive to electrostatic discharge (ESD) caused by static electricity
- FPGAs often offer separate voltage supplies for the internal logic, called the core, and for the input/output (I/O) pins

- FPGAs have configurable I/Os that can operate at many different voltages, so as to be compatible with the rest of the system.

Packaging and Assembly

- integrated circuits are typically placed in packages made of plastic or ceramic
  - connect the tiny metal I/O pads of the chip to larger pins in the package for ease of connection
  - protect the chip from physical damage
  - spread the heat generated by the chip over a larger area to help with cooling
- packages are placed on a breadboard or printed circuit board and wired together to assemble the system
- generally categorized
  - through-hole
    - have pins that can be inserted through holes in a printed circuit board or into a socket
    - dual inline packages (DIPs) have two rows of pins with 0.1-inch spacing between pins
    - pin grid arrays (PGAs) support more pins in a smaller package by placing the pins under the package
  - surface mount (SMT)
    - soldered directly to the surface of a printed circuit board without using holes
    - pins are called leads
    - thin small outline package (TSOP) has two rows of closely spaced leads (typically 0.02-inch spacing)
    - plastic leaded chip carriers (PLCCs) have J-shaped leads on all four sides, with 0.05-inch spacing
    - can be soldered directly to a board or placed in special sockets
    - Quad flat packs (QFPs) accommodate a large number of pins using closely spaced legs on all four sides
    - Ball grid arrays (BGAs) eliminate the legs altogether
      - have hundreds of tiny solder balls on the underside of the package
      - carefully placed over matching pads on a printed circuit board, then heated so that the solder melts and joins the package to the underlying board

Breadboards

- breadboard is a plastic board containing rows of sockets
- all five holes in a row are connected together
- each pin of the package is placed in a hole in a separate row
- wires can be placed in adjacent holes in the same row to make connections to the pin

- often provide separate columns of connected holes running the height of the board to distribute power and ground
- easy to accidentally plug a wire in the wrong hole or have a wire fall out
- requires a great deal of care (and usually some debugging in the laboratory)
- only for prototyping not production

Printed Circuit Boards

- chip packages may be soldered to a printed circuit board (PCB)
- formed of alternating layers of conducting copper and insulating epoxy
- copper is etched to form wires called traces
- holes called vias are drilled through the board and plated with metal to connect between layers
- usually designed with computer-aided design (CAD) tools
- can etch and drill your own simple boards in the laboratory
- can send the board design to a specialized factory for inexpensive mass production
- traces are normally made of copper because of its low resistance
- traces are embedded in an insulating material, usually a green, fire resistant plastic called FR4
- typically has copper power and ground layers, called planes, between signal layers

Putting It All Together

- most modern chips with large numbers of inputs and outputs use SMT packages, especially QFPs and BGAs
- packages require a printed circuit board rather than a breadboard
- working with BGAs is challenging because they require specialized assembly equipment
- balls cannot be probed with a voltmeter or oscilloscope during debugging in the laboratory because they are hidden under the package
- designer needs to consider packaging early on to determine whether a breadboard can be used during prototyping and whether BGA parts will be required
- professional engineers rarely use breadboards when they are confident of connecting chips together correctly without experimentation

Transmission Lines

Short Termination

Mismatched Termination

When to Use Transmission Line Models

Putting it all Together

- transmission lines model the fact that signals take time to propagate down long wires because the speed of light is finite
- ideal transmission line has uniform inductance L and capacitance C per unit length and zero resistance
- transmission line is characterized by its characteristic impedance Z0 and delay td which can be derived from the inductance, capacitance, and wire length
- transmission line has significant delay and noise effects on signals whose rise/fall times are less than about 5td
- for systems with 2 ns rise/fall times, PCB traces longer than about 6 cm must be analyzed as transmission lines to accurately understand their behavior
- a digital system consisting of a gate driving a long wire attached to the input of a second gate can be modeled with a transmission line
- voltage source, source impedance ZS, and switch model the first gate switching from 0 to 1 at time 0
- driver gate cannot supply infinite current
  - modeled by ZS
  - ZS is usually small for a logic gate, but a designer may choose to add a resistor in series with the gate to raise ZS and match the impedance of the line
  - input to the second gate is modeled as ZL
  - CMOS circuits usually have little input current, so ZL may be close to infinity
  - designer may also choose to add a resistor in parallel with the second gate, between the gate input and ground, so that ZL matches the impedance of the line
- when the first gate switches, a wave of voltage is driven onto the transmission line
- source impedance and transmission line form a voltage divider
  - wave formula here
- at time td, the wave reaches the end of the line
- part is absorbed by the load impedance, and part is reflected
- reflection coefficient kr indicates the portion that is reflected
  - formulas here
- reflected wave adds to the voltage already on the line

- reaches the source at time 2td, where part is absorbed and part is again reflected
- reflections continue back and forth, and the voltage on the line eventually approaches the value that would be expected if the line were a simple equipotential wire

Economics

- economic considerations are a major factor in design decisions
- cost of a digital system can be divided
  - nonrecurring engineering costs (NRE)
    - accounts for the cost of designing the system
    - includes the salaries of the design team, computer and software costs, and the costs of producing the first working unit
    - fully loaded cost of a designer in the United States in 2012 (including salary, health insurance, retirement plan, and a computer with design tools) was roughly $200,000 per year
  - recurring costs
    - cost of each additional unit
      - components
      - manufacturing
      - marketing
      - technical support
      - shipping
- sales price must cover not only the cost of the system but also other costs such as office rental, taxes, and salaries of staff who do not directly contribute to the design (such as the janitor and the CEO)
- chips are usually purchased from a distributor rather than directly from the manufacturer (unless you are ordering tens of thousands of units)
- Digikey (www.digikey.com)
- Jameco (www.jameco.com)
- All Electronics (www.allelectronics.com)