# Contents




**Contributors:**

- Daniel Fitz (Sanchez)

## 0.1 Assessment

- Online Quizzes (10% = Best 10 $\times$ 1%)
  Due Mondays at 8am
- Mid-semester exam (10% or 20%)
  Saturday (centrally scheduled - sometime week 5 to 7)
  Multiple-choice, open-book
- Prac Exam (Pass/Fail)
  Held during Monday/Wednesday Learning Lab sessions in week 6
  You must pass in order to pass the course
- Project (20%)
  Develop a microcontroller program
- Final Exam (50% or 60%)
  Short answer, problem solving, open-book

# Chapter 1

# Lecture Notes

## 1.1 Bits, Bytes and Binary

### 1.1.1 Structured Computer Organization

**Level 5:** Problem-oriented language level
**Level 4:** Assembly language level
**Level 3:** Operating system machine level
**Level 2:** Instruction set architecture level
**Level 1:** Microarchitecture level
**Level 0:** Digital Logic level

### 1.1.2 Unsigned Number in Binary

Each bit position has a value $\rightarrow$ $2^n$ (starting at zero). Add all values of the positions together and that's unsigned value.

### 1.1.3 Converting Decimal to Binary

- Method 1
  rewrite $n$ as sum of powers of 2 (by repeatedly subtracting largest power of 2 not greater than $n$)
  Assemble binary number from 1's in bit positions corresponding to those powers of 2, 0's elsewhere
- Method 2
  Divide $n$ by 2
  Remainder of division (0 or 1) is next bit
  Repeat with $n$ = quotient

**Note 1: Example**

Convert 53 to binary

$$\frac{53}{2} = 26 \text{ rem } 1 \Rightarrow 1$$
$$\frac{26}{2} = 13 \text{ rem } 0 \Rightarrow 0$$
$$\frac{13}{2} = 6 \text{ rem } 1 \Rightarrow 1$$
$$\frac{6}{2} = 3 \text{ rem } 0 \Rightarrow 0$$
$$\frac{3}{2} = 1 \text{ rem } 1 \Rightarrow 1$$
$$\frac{1}{2} = 1 \text{ rem } 1 \Rightarrow 1$$

$\therefore 53 \equiv 0b110101$

### 1.1.4 Least and Most Significant Bits

**Most Significant Bit (MSB):** Bit that's worth the most, the left-most bit
**Least Significant Bit (LSB):** Bit that's worth the least, the right-most bit

**Note 2: Radices**

- **Radix:** number system base
- A radix-k number system
  $k$ different symbols to represent digits 0 to $k-1$
  Value of each digit is (from the right) $k^0, k^1, k^2, k^3, \ldots$
- Often convenient to deal with
  **Octal** (radix-8) - Symbols: 0, 1, 2, 3, 4, 5, 6, 7
  *One octal digit corresponds to 3 bits*
  **Hexadecimal** (radix-16) - Symbols: 0, 1, 2, 3, 4, 5, 6, 7, 7, 8, 9, A, B, C, D, E, F
  *One hexadecimal digit corresponds to 4 bits (useful)*

**Note 3: Radix Identification**

- Hexadecimal
  - Leading 0x (C, Atmel AVR)
  - Trailing h (Some assembly languages)
  - Leading $ (Atmel AVR Assembly)
- Octal
  - Leading 0 (C, Atmel AVR)
  - Trailing q (Some assembly languages)
  - Leading @ (Some assembly languages)
- Binary
  - Leading 0b (Atmel AVR Assembly, Some C)
  - Trailing b (Some assembly languages)
  - Leading % (some assembly languages)

### 1.1.5 Conversions

Easiest to convert from most formats to binary then to the desired format.

#### Octal

**From Binary:** Group bits into series of 3 and then convert to decimal (0b010 = 02)

**To Binary:** Convert each octal number to binary and append

#### Hex

**From Binary:** Group bits into series of 4 and then convert to hex with overflow being apart of the alphabet (0b1100 = 0xC)

**To Binary:** Convert each hex number to binary and append

#### Decimal

**From Binary:** Add together the powers of two at each position $n$ ($0b1010 = 2^3 + 2^1 = 10$)

**To Binary:** Starting with LSB, divide by 2 with the remainder being bit value at position. ($9 = 9/2 = 4rem1, 4/2 = 2rem0, 2/2 = 1rem0, 1/2 = 0rem1. \therefore 9 = 0b1001$)

### 1.1.6 Negative Numbers

#### Signed Magnitude

Leftmost bit is the sign bit, true is negative and false is positive

#### One's Complement

Leftmost bit = sign-bit (as per signed magnitude), true is negative and false is positive. If negative all bits are inverted

#### Two's Complement

MSB signifies if negative, true is negative and false is positive. To negate invert all bits and add decimal 1.

> Allows addition without requiring conversion

#### Excess $2^{m-1}$

e.g. for 8 bits, excess-128. Add 128 to the original bit and convert to binary

## 1.2 Logic Gates

**NOT Gate:** Inverts the signal (i.e. input is true, output is false)

**AND Gate:** Output is true only if **all** inputs are true

**NAND Gate:** Opposite of AND, always true unless all inputs are true

**OR Gate:** Output is true when at **least one** input is true

**XOR Gate:** Output is true if only one input is true

**Note 4: XOR Multiple Inputs**

For more than 2 inputs, XOR is true if there is an odd number of inputs true. Also referred to as the "odd function"

### 1.2.1 Logic Functions

- Logic functions can be expressed as expressions involving:
  - variables (literals), e.g. A B X
  - functions, e.g. $+. \oplus \overline{A}$
- Rules about how this works called **Boolean algebra**
- Variables and functions can only take on values **0** or **1**

**Convenctions**

**Inversion:** $\overline{A}$ (overline of A)
**AND:** dot(.) or implied by adjacency. $AB = A.B$
**OR:** plus sign. $OR(A, B, C) = A + B + C$
**XOR:** $OR(A, B) = A \oplus B = \overline{A}B + A\overline{B}$
**NAND:** $\overline{ABC}$
**NOR:** $\overline{A + B}$

**Representations of Logic Functions**

There are four representations of logic functions (assume function of $n$ inputs)

- **Truth Table**
  Lists output for all $2^n$ combinations of inputs
- **Boolean Function** (or equation)
  Describes the conditions under which the function output is true
- **Logic Diagram**
  Combination of logic symbols joined by wires
- **Timing Diagram**

### 1.2.2 Logic Function Implementation

Any logic function can be implemented as the OR of AND combinations of the inputs. Called **sum of products**.

Table 1.1: Boolean Identities

| Name | AND Form | OR Form |
|---|---|---|
| Identity Law | $1A = A$ | $0 + A = A$ |
| Null Law | $0A = 0$ | $1 + A = 1$ |
| Idempotent Law | $AA = A$ | $A + A = A$ |
| Commutative Law | $AB = BA$ | $A + B = B + A$ |
| Associative Law | $(AB)C = A(BC)$ | $(A + B) + C = A + (B + C)$ |
| Distributive Law | $A + BC = (A + B)(A + C)$ | $A(B + C) = AB + AC$ |
| Absorption Law | $A(A + B) = A$ | $A + AB = AB$ |
| De Morgan's Law | $\overline{AB} = \overline{A} + \overline{B}$ | $\overline{A + B} = \overline{AB}$ |

## 1.3 Binary Arithmetic

### 1.3.1 Equivalent Circuits

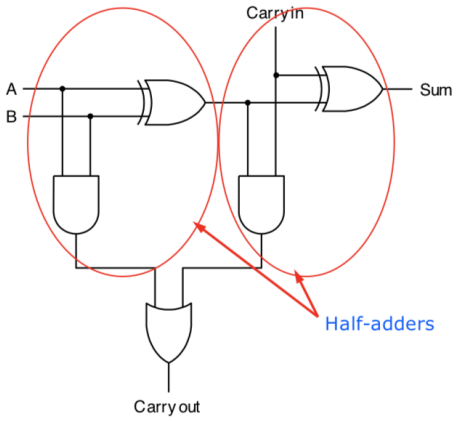
All circuits can be constructed from NAND and NOR gates

### 1.3.2 Overflow

Overflow with two's complement addition:

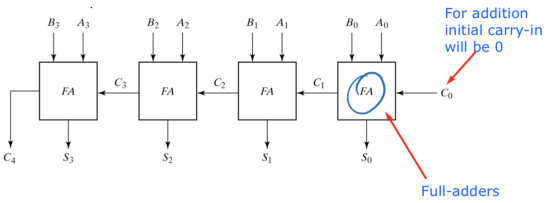
- Carry into sign-bit is different to the carry out of the sign-bit
- Equivalently, overflow occurs if
  Two negatives added together give a positive, or
  Two positives added together give a negative

### 1.3.3 Full Adder




### 1.3.4 Binary Adder

Can cascade full adders to make binary adder. This is a **ripple-carry adder**.



## 1.4 Combination Logic

### 1.4.1 Combinational Circuits

Each output can be expressed as a function of $n$ input variables. Can write truth table also:

- $n$ input columns
- $m$ output columns
- $2^n$ rows (i.e. possible input combination)

**Note 5: Multiplexer (or Mux)**

- $2^n$ data inputs
- 1 output
- $n$ control (or **select**) inputs - that **select** one of the inputs to be "sent" or "steered" to the output

**Note 6: Decoder**

Converts $n$-bit input to a logic-1 on exactly one of $2^n$ outputs

### 1.4.2 Timing Diagram

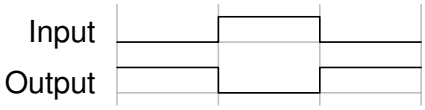



Figure 1.1: Timing Diagram of an inverter

There is a slight delay in logic timings in reality

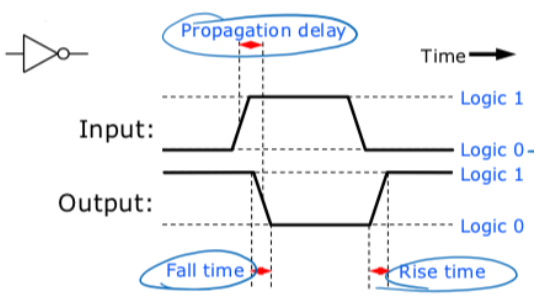



Figure 1.2: Reality of Timing

**Propagation delay:** time for change in input to affect output
**Fall time:** time taken for output to fall from 1 to 0
**Rise time:** time for output to rise from 0 to 1

## 1.5 Flip-flops

### 1.5.1 D Flip Flop

- **D** is input
- **Q** is output
- **CLK** (clock) is control input

Q copies the value of D (and remembers it) whenever CLK goes from 0 to 1 (**rising edge**).

#### Characteristic Table

**Characteristic table** defines operation of flip-flop in tabular form

Table 1.2: D Flip-Flop Characteristic Table

| D | Q(t+1) |
|---|---|
| 0 | 0 |
| 1 | 1 |

### 1.5.2 Flip-Flops Vs Latches

- The last few slides show **latches**
  These are **level-triggered** devices
- Remember we want to capture the input value at rising **edge** (a short instant)!
- Any devices based on edges are referred to as **flip-flops**
  These are **edge-triggered** devices

## 1.6 Shift Registers

### 1.6.1 Combinational vs Sequential Circuits

- **Combinational** Circuits
  - Logic gates only (no flip-flops)
  - Output is uniquely determined by the inputs
    i.e. you'll always get the same output for a given set of inputs
- **Sequential** Circuits
  - Include flip-flops
  - Output determined by current inputs and current **state** (values in the flip-flops)
  - Output can change when clock "ticks" (rising edge)

#### Sequential Circuits

- **State** is value stored in flip-flops

- Output depends on input and state
  or sometimes just the state
- Next state depends on inputs and state

#### Synchronous Sequential Circuit

- Storage elements (flip-flops) can only change at discrete instants of time

### 1.6.2 Registers

- A **register** is a group of flip-flops
  $n$-bit register consists of $n$ flip-flops capable of storing $n$ bits
- A register is a sequential circuit *without* any combinational logic

#### Shift Register

A shift register is a register which is capable of shifting its binary information in one or both directions

## 1.7 Counters

- A **counter** is a multi-bit register that goes through a determined sequence of states (values) upon the application of input pulses
- A counter which follows binary number sequence is a **binary counter**
  $n$-bit binary counter has $n$ flip-flops and can count from 0 to $2^n - 1$

**Note 7: State**

- Values stored in the flip-flops can be considered the **current state** of the circuit
- D inputs to the flip-flops are the **next state**
- D inputs are some function of the current state and inputs

#### Key Points

- Next state is a function of previous state (and possibly inputs)
- Count sequence can be binary numbers but does not have to be
  If it is, counter is a **binary counter**
- Circuits are **synchronous**
  All flip-flops have the same clock

## 1.8 State Machines

- Sequential circuits can also be called
  state machines
  finite state machines (FSMs)
- State machine has
  - Finite number of possible states
  - Only one **current state**
  - Can **transition** to other states based on inputs and current state

**Note 8: Types of State Machines**

**Mealy Machines:** Outputs depend on current state and inputs
**Moore Machines:** Outputs depend only on current state (flip-flop values)
Outputs can only change when state changes

### 1.8.1 State diagram

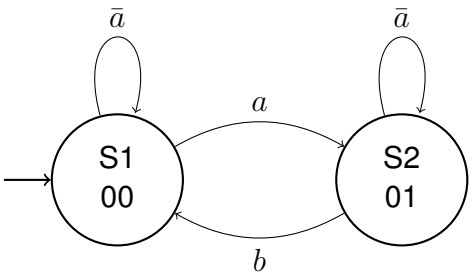



Figure 1.3: Example single input state diagram

*Note: I couldn't figure out how to add the line that is meant to go between the state label and the state number*

#### Completeness

Each possible combination of inputs should be addressed **exactly once** for each state. i.e. transition arrows from each state must encompass all possibilities (exactly once)

### 1.8.2 State tables

- State diagrams can also be represented in a state table

Table 1.3: Example State Table

| Current State | Input $U$ | Next State | Outputs | |
|---|---|---|---|---|
| | | | $Q_1$ | $Q_2$ |
| S0 | 0 | S3 | 0 | 0 |
| S0 | 1 | S1 | 0 | 0 |
| S1 | 0 | S0 | 0 | 1 |
| S1 | 1 | S2 | 0 | 1 |
| S2 | 0 | S1 | 1 | 0 |
| S2 | 1 | S3 | 1 | 0 |
| S3 | 0 | S2 | 1 | 1 |
| S3 | 1 | S1 | 1 | 1 |

Table 1.4: Two-dimensional state table

| Current State | Next State | | Outputs | |
|---|---|---|---|---|
| | $\bar{U}$ | $U$ | $Q_1$ | $Q_0$ |
| S0 | S3 | S1 | 0 | 0 |
| S1 | S0 | S2 | 0 | 1 |
| S2 | S1 | S3 | 1 | 0 |
| S3 | S2 | S0 | 1 | 1 |

### 1.8.3 State encoding

- Must encode each state into flip-flop values
- Choose
  - Number of flip-flops
  - Bit patterns that represent each state
- Ideally, choose state encoding to make combinational logic simple, for both
  - Output logic
  - Next state logic

## 1.9 ALUs and Memory

### 1.9.1 Parts of a CPU

- **Control Unit**
  - Fetches instructions from memory, makes the ALU and the registers perform the instruction
- **ALU**
  - Performs arithmetic and logical operations
- **Registers**
  - High speed memory – stores temporary results and control

### 1.9.2 Registers

Different types of registers

- **Program Counter Register**
  - Stores the memory address of the next instruction to be fetched
- **Instruction Register**
  - Contains the current instruction
- **General Purpose Registers**
  - Contains data to be operated on (e.g. data read from memory), results of operations, ...
  - Width is CPU word size
  - Sometimes called the **register file**

### 1.9.3 Buses

**Bus** = Common pathway (collection of "wires") connecting parts of a computer
Characteristics:

- Can be **internal** to CPU (e.g. ALU to registers)
- Can be **external** to CPU (e.g. CPU to memory)
- Buses have a **width** – number of bits that can be transferred together over a bus
  - May not always be the same as the word size of the computer

**Note 9: Arithmetic Logic Unit (ALU)**

Does more than adding... **Function** / **control** input dictates the operation that the ALU is to perform, e.g.

- Addition
- Increment (+1)
- Subtraction
- Bitwise AND
- Bitwise OR

Like adders, ALUs can be made up from 1-bit slices

### 1.9.4 Data Path

- Operands come from register file
- Result written to register file
- Implements routine instructions such as arithmetic, logical, shift
- Width of registers/buses is the CPU word size

## 1.10 CPU Control Unit

### 1.10.1 Control Unit

- Control signals come from the **control unit**

- Control unit must generate the control signals in the right order for a given instruction
  Instruction determined by contents of the **Instruction Register (IR)**

**Data Influencing Control**

- Need to know more than what's in the instruction register
- Control unit can't operate without knowing something about the data
- **Status Register**
  Determined by ALU operations
  Stores status of last ALU operation

**Note 10: Status Register**

Typically Includes:

**Z:** zero bit – was the last result 0?

**V:** overflow bit – did the last addition/subtraction operation overflow (assuming it was a two's complement operation)?

**C:** carry bit – was a carry out generated?

**N:** negative bit – was the result negative if considered as two's complement (i.e. what's the sign bit)?