Computer Architecture and Operating Systems

First year, Computer Science

Prof. dr. Henri Luchian

Lect. dr. Vlad Rădulescu

- examination
  - two written tests - course
    - one for each half of the semester
  - one practical test - laboratory
    - assembly language
- in order to be accepted to the written tests
  - be present at the laboratory classes
    - any student may be absent at most twice during each half of the semester

Contents: First Half of the Semester

I. Introduction

II. Combinational circuits and Boole functions

III. Sequential circuits and automata

IV. Internal representations

V. Computer architecture and organization

I. Introduction

I.1. Evolution

How Do We Define Computing?

- what operations can be performed?
- evolution
  - abacus: addition
  - toothed wheels (Leibniz, Pascal): addition, multiplication
  - Babbage: external instructions, branch computing
  - von Neumann: memorized program; execution as a sequence of instructions; memory hierarchies
  - parallel computing, quantum computing, etc.

Universal Computing Machines

- a universal computing machine can behave like any particular computing machine
  - so it can solve any problem that a particular computing machine can solve
- example - the computer
  - depending on the program it executes, it solves problems like: matrix computation, graphic design, desktop publishing, etc.

Short History (1)

- positional writing of numbers
  - Indians, Arabs
- Boole algebra
  - George Boole, 1854
- the incompleteness theorem
  - Kurt Gödel, 1935
- the link between Boole algebra and circuits
  - Claude Shannon, 1938

Short History (2)

- The "Neumannian" computer
  - John von Neumann, 1946
- the transistor
  - Shockley, Brittain, Bardeen, 1947
- integrated circuits

I.2. Empirical Laws

Empirical Laws

- in any field of science, the laws depend (one way or another) on experiments or on real world observations
- reproductibility leads to the idea of empirical laws: they are true in most cases, according to observations

Empirical Laws in Computer Science

- the "90:10" law (Donald Knuth)
  - 90% of the execution time of a program is used for 10% of the instructions
- Amdahl's law
  - highest efficiency in improving a system (either concrete or abstract) is achieved when the most intensively used subsystem is optimized
- locality laws - spatial, temporal

Amdahl's Law (1)

- consider a system (hardware or software) and one of its components
- that component works a certain percentage f_a of the system's total work time
- and it is improved, such that if gets to work a times faster than before
- how much times faster does the system as a whole become?

Amdahl's Law (2)

$$A(a, f_a) = \frac{1}{(1 - f_a) + \frac{f_a}{a}}$$

- highest increase of the overall speed
  - better improvement of the component (a)
  - improving the components with the highest weight in the system's work time (f_a)
    - i.e., the most intensely used ones

Temporal Locality

- if a memory location is accessed at a certain moment, it will very likely be accessed again in the near future
- examples
  - variables are used repeatedly in a program
  - program loops - instructions are repeated

Spatial Locality

- if a memory location is accessed at a certain moment, the neighboring locations will very likely be accessed again in the near future
- examples
  - walking through arrays
  - the execution of instruction sequences - their codes are stored at consecutive memory addresses

Physical Order and Logical Order

- instructions to be executed are stored in memory in the physical order
- they are read from memory and executed
  - the rule: in the same order they are memorized
  - exception: jumping over several instructions
- the result is the logical order of instructions
  - may differ from one run to another
  - an instruction can be executed 0, 1, 2, … times

II. Combinational Circuits and Boole Functions

Analogical Signal vs. Digital Signal

- analogical signal - continuous
  - if it can have values *a* and *b*, then it can also have any value in [*a*,*b*]
- digital signal - discrete
  - can only have a few distinct levels (values)
  - computer - 2-level digital signal (0 and 1)
  - there are also other computing systems apart from PCs

Types of Circuits

- combinational circuits
  - the output values depend only on the input values
  - the same input values always yield the same output values
- sequential circuits
  - beside the inputs, the output values also depend on the state of the circuit
  - which evolves through time

Truth tables

- how to describe how a combinational circuit works?
- apply each possible combination of input values
- and observe the output values for each such combination
- together, these relationships (input-output) make a truth table

Circuits and Boole Functions

- each truth table has a corresponding Boole function
  - so each combinational circuit has a corresponding Boole function

inputs			outputs		
I_1	**...**	**I_n**	**O_1**	**...**	**O_m**
0	0...0	0	?	?...?	?
0	0...0	1	?	?...?	?
...	...	...	...	...	...
1	1...1	1	?	?...?	?

II.1. Boole Functions

Algebraic Structure

- a non-void set B, containing at least two elements: a, b, $a \neq b$
- the set of binary operations $\{ +, \cdot \}$
- one unary operation $\{ \bar{} \}$
- closure: $a + b \in B$

  $a \cdot b \in B$

  $\bar{a} \in B$

Boole Functions

- B = {0,1}
- f : $B^n \rightarrow B^m$
  - function: *n* variables, *m* values
  - circuit: *n* inputs, *m* outputs
- there are $(2^m)^{2^n}$ such distinct functions
  - *n* = 1, *m* = 1: 4 unary functions of one value
  - *n* = 2, *m* = 1: 16 2-variable Boole functions of one value

Truth Tables

a	$f_0(a)$	$f_1(a)$	$f_2(a)$	$f_3(a)$
0	0	0	1	1
1	0	1	0	1
	$= 0$	$= a$	$= \bar{a}$	$= 1$

a	b	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Axioms and Theorems in Boole Algebra (1)

identity	$X + 0 = X$	$X \cdot 1 = X$
constants	$X + 1 = 1$	$X \cdot 0 = 0$
idempotence	$X + X = X$	$X \cdot X = X$
involution	$\overline{\overline{X}} = X$	
complementarity	$X + \overline{X} = 1$	$X \cdot \overline{X} = 0$
commutativity	$X + Y = Y + X$	$X \cdot Y = Y \cdot X$
associativity	$(X + Y) + Z = X + (Y + Z)$	$(X \cdot Y) \cdot Z = X \cdot (Y \cdot Z)$
distributivity	$X \cdot (Y+Z) = (X \cdot Y) + (X \cdot Z)$	$X+(Y \cdot Z)=(X+Y) \cdot (X+Z)$

Axioms and Theorems in Boole Algebra (2)

unification	$X \cdot Y + X \cdot \overline{Y} = X$	$(X + Y) \cdot (X + \overline{Y}) = X$
absorption	$X + X \cdot Y = X$ $(X + \overline{Y}) \cdot Y = X \cdot Y$	$X \cdot (X + Y) = X$ $(X \cdot \overline{Y}) + Y = X + Y$
De Morgan laws	$\overline{X + Y + \ldots} = \overline{X} \cdot \overline{Y} \cdot \ldots$	$\overline{X \cdot Y \cdot \ldots} = \overline{X} + \overline{Y} + \ldots$
generalization (duality)	$\overline{f(X_1, \ldots, X_n, 0, 1, +, \cdot)} = f(\overline{X_1}, \ldots, \overline{X_n}, 1, 0, \cdot, +)$	

The Computer - Elementary Operations

- in today's computers, elementary operations are the operations of Boole logic
  - which simulate (among others) the elementary arithmetic operations in base 2
- a combinational circuit actually implements a Boole function
  - how do we get the expression of the Boole function from the truth table?

Normal Forms

- disjunctive normal form (DNF)
  - for each row that yields value 1 on output – conjunction term ($\cdot$)
    - contains each variable: negated if the variable is 0 on that row, not negated if the variable is 1
  - these terms - connected through disjunction (+)
- conjunctive normal form (CNF): dual
- example: $F_9(x, y) = \overline{x} \cdot \overline{y} + x \cdot y = (x + \overline{y}) \cdot (\overline{x} + y)$

II.2. Logic Diagrams

The Alphabet of Logic Diagrams

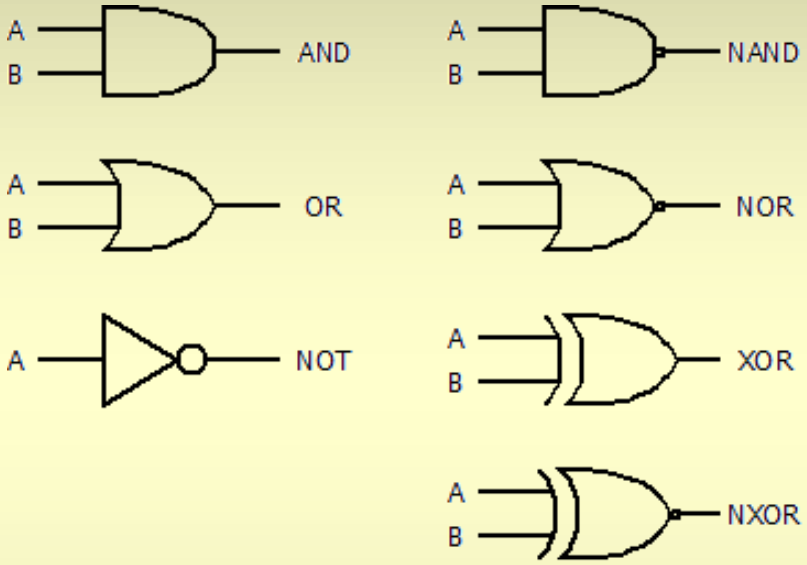
- logic gates are the implementations of some Boole functions
- so the behavior of each gate can be described by a truth table
  - corresponding to the Boole function associated with the gate
- elementary gates: AND, OR, NOT
- other gates: NAND, NOR, XOR, NXOR

The Alphabet of Logic Diagrams

A	NOT
0	1
1	0

A	B	AND	OR	NAND	NOR	XOR	NXOR
0	0	0	0	1	1	0	1
0	1	0	1	1	0	1	0
1	0	0	1	1	0	1	0
1	1	1	1	0	0	0	1

Gate Symbols



– associative binary operations can be extended operations with a finite number of operands

Minimal Set of Generators

- set of generators - set of gate types which can implement any Boole function
  - minimal set of generators - set of generators with the minimal number of gate types
- it is possible with 3 (NOT, AND, OR)
  - normal forms (disjunctive, conjunctive)
  - also possible with (NOT + AND, NOT + OR)
  - minimal - 1 (NAND, NOR)

Homework

- prove that the following sets of gate types are sets of generators
  - NOT + AND
  - NOT + OR
  - NAND
  - NOR

II.3. Circuit Implementation Through Boole Functions

Defining Boole Functions

- can be defined in several ways
  - truth table
  - expressions - variables and logical operations
  - graphic representation
  - sigma-notation (Σ)
- in the end, we need to have a Boole expression
  - which allows a gate implementation

Σ-notation(1)

- example - "majority of k inputs"
  - function value: 1 if most input variables have value 1, 0 otherwise
    - for 3 variables: $f(x_1,x_2,x_3) = \Sigma(3,5,6,7)$
- Σ-notation corresponds to the disjunctive normal form
  - each number in the brackets is a conjunction term
  - Σ denotes the disjunction of terms

Σ-notation(2)

- how many variables are necessary?
  - the lowest power of 2 that is equal to or greater than the highest number in the brackets
    - for our example: $2^2 < 7 < 2^3 \rightarrow n = 3$
- the term corresponding to a number
  - all variables, connected through conjunction
  - each variable is negated if assigned to a bit with value 0; not negated for 1
    - example: $3_{(10)} = 011_{(2)} \rightarrow \overline{x_1} \cdot x_2 \cdot x_3$

Minimization (1)

- the disjunctive normal form of the function majority of 3

$$f(A,B,C) = \overline{A} \cdot B \cdot C + A \cdot \overline{B} \cdot C + A \cdot B \cdot \overline{C} + A \cdot B \cdot C$$

  - large number of elementary gates
  - a simpler equivalent expression (**the same Boole function**) would make the circuit
    - faster
    - cheaper
    - more reliable

Minimization (2)

- how to get a simpler expression from the disjunctive normal form ?
  - equivalent rewriting
    - use the laws and axioms of Boole algebra
  - perfect induction
  - Veitch-Karnaugh method
  - Quine-McCluskey method
  - hybridization (combine the above methods)

Minimization - Algebraic Rewriting

- same example

$$f = \overline{A} \cdot B \cdot C + A \cdot \overline{B} \cdot C + A \cdot B \cdot \overline{C} + A \cdot B \cdot C$$

(idempotence)

$$= \overline{A} \cdot B \cdot C + A \cdot \overline{B} \cdot C + A \cdot B \cdot \overline{C} + A \cdot B \cdot C + A \cdot B \cdot C + A \cdot B \cdot C$$

(unification)

$$= B \cdot C + A \cdot C + A \cdot B$$

- difficult for complex expressions

Homework

- find the disjunctive normal form and study minimization through algebraic rewriting for the function "odd"
  - the function value is: 1 for an odd number of inputs with value 1; 0 otherwise