Introduction and boolean algebra

Lecture 1 — § 1.1, 1.2

Computer Science 218

Mike Feeley

What is a computer?

so you are at a party

- and David Hilbert walks up to you and says
  - 19th century Russian mathematician
  - "so, you're a computer scientist, eh?"
  - "what is a computer exactly?"

what is your answer?

- divide yourselves into groups of 4-5
- take 5 minutes come up with your "party" answer
- then we'll talk about it

Answer depends on who you are

what you know

- any sufficiently advanced technology is indistinguishable from magic
  - Arthur C. Clarke

how you think

- deconstructionist
  - start from first principles and build to a higher-level abstraction
  - "okay, so there is this thing called string theory ..."
- generalist
  - start from high-level abstraction and explain with increasing detail
  - "it lets me surf the web ..."

Our approach

goal of course is to explain what a computer is

- starting from the bottom and working up
- we'll go from gates to a simple PC

deconstructing a PC

- software
  - firmware
  - OS
  - libraries
  - applications
- hardware
  - CPU
  - memory
  - IO devices
  - interconnection network (busses)

Course outline

roughly one week per topic

- circuits
- components
- numbers
- micro operations
- system organization
- instruction sets and assembly language (software)

in the lab

- you'll build hardware stuff
  - seven labs: easy, hard, easy, hard, easy, hard, hard
- in a circuit simulator

Layering of abstraction

RTL and micro operations

- language for describing operations on multi-bit values in registers
- busses transfer data among registers

integers, floating point, and other encodings (BCD, ASCII, unicode, etc.)

- data representations and hardware operations on them

packages

- circuit modules

sequential circuit

- combinational circuits plus flip-flop (memory/state)
- finite state diagrams

combinational circuits

- boolean algebra, truth tables and logic diagrams

gates

switches

transistors (wires, resistors, and capacitors) on a wafer

RTL description of a simple CPU

interrupt cycle

Rt_0: AR ← 0
Rt_1: M[AR] ← PC, PC ← 0
Rt_2: PC ← PC+1, IEN ← 0, R ← 0, SC ← 0

fetch and decode

$R't_0$: AR ← PC, PC ← PC+1
$R't_1$: IR ← M[AR]
$R't_2$: AR ← IR(0-11)

$t_0't_1't_2'$(IEN)(FGI+FGO): R ← 1

execute register instruction

cla $t_3d_7i'b_{11}$: AC ← 0
cle $t_3d_7i'b_{10}$: E ← 0
cma $t_3d_7i'b_9$: AC ← AC´
cme $t_3d_7i'b_8$: E ← E´
cir $t_3d_7i'b_7$: AC ← shr AC, AC(15) ← E, E ← AC(0)
cil $t_3d_7i'b_6$: AC ← shl AC, AC(0) ← E, E ← AC(15)
inc $t_3d_7i'b_5$: AC ← AC+1
spa $t_3d_7i'b_4$AC(15)´: PC ← PC+1
sna $t_3d_7i'b_3$AC(15): PC ← PC+1
sza $t_3d_7i'b_2$(AC(0)+...+AC(15))´: PC ← PC+1
sze $t_3d_7i'b_1$E´: PC ← PC+1
hlt $t_3d_7i'b_0$: X ← 0
t_3d_7i': SC ← 0

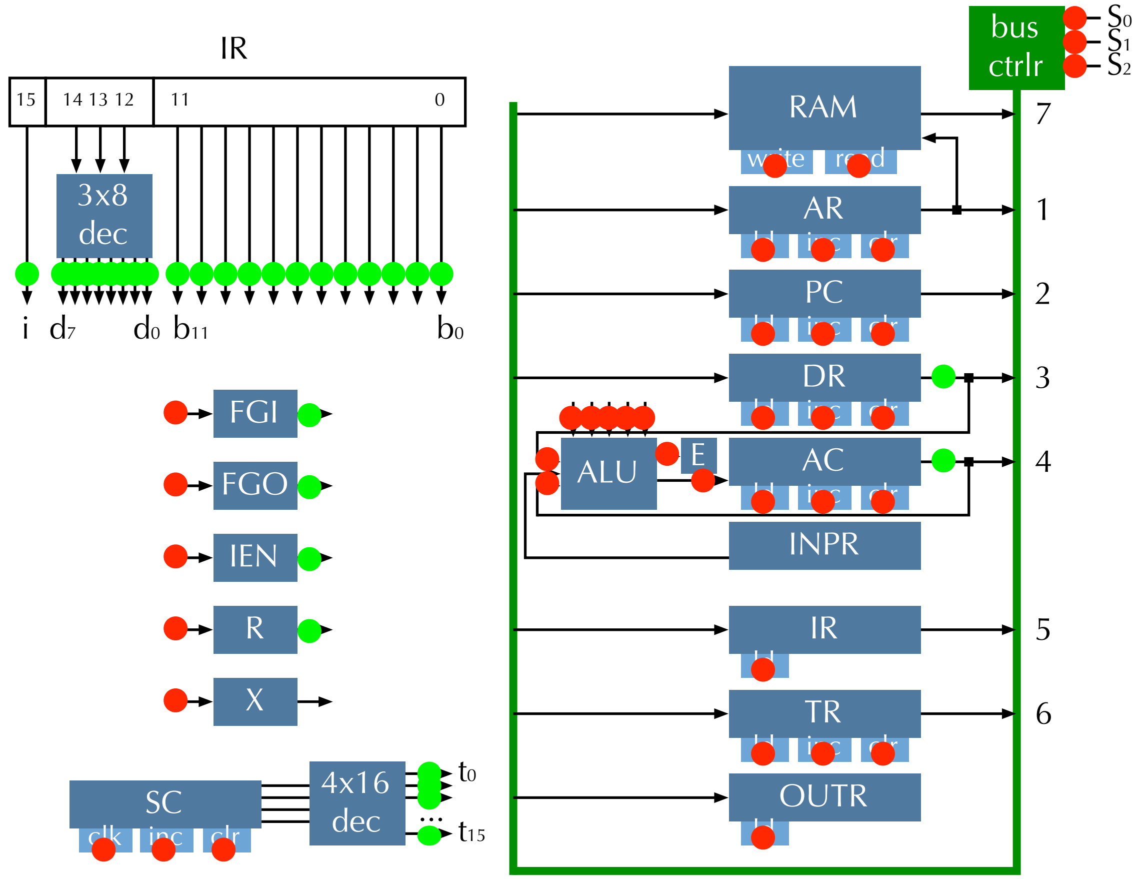
execute memory instruction

$t_3d_7'i$: AR ← M[AR]
and t_4d_0: DR ← M[AR]
t_5d_0: AC ← AC∧DR, SC ← 0
add t_4d_1: DR ← M[AR]
t_5d_1: AC ← AC+DR, E ← C_{out}, SC ← 0
lda t_4d_2: DR ← M[AR]
t_td_2: AC ← DR, SC ← 0
sta t_4d_3: M[AR] ← AC, SC ← 0
bun t_4d_4: PC ← AR, SC ← 0
bsa t_4d_5: M[AR] ← PC, AR ← AR+1
t_5d_5: PC ← AR, SC ← 0
isz t_4d_6: DR ← M[AR]
t_5d_6: DR ← DR+1
t_6d_6: M[AR] ← DR, SC ← 0
t_6d_6(DR(0)+...+DR(15))´: PC ← PC+1

execute I/O instruction

inp $t_3d_7b_{11}$: AC(0-7) ← INPR, FGI ← 0
out $t_3d_7b_{10}$: OUTR ← AC(0-7), FGO ← 0,
ski $t_3d_7b_9$FGI: PC ← PC+1
sko $t_3d_7b_8$FGO: PC ← PC+1
ion $t_3d_7b_7$: IEN ← 1
iof $t_3d_7b_6$: IEN ← 0
t_3d_7: SC ← 0

Register and bus diagram of CPU



Historical deconstruction

read more at mathworld.wolfram.com

Hilbert's 23 problems for 20th century (Paris 1900)

- is there a proof of Cantor's Continuum Hypothesis? (1)
  - no - proved undecidable in two parts: 1938 by Kurt Gödel and 1963 by Paul Cohen
- can axioms of logic be proven to be consistent? (2)
  - no - proved undecidable in 1931 by Kurt Gödel's incompleteness theorems
- is there a universal algorithm for solving Diophantine equations? (10)
  - integer solutions; e.g., Fermat's last theorem, which was solved in 1993 by Andrew Wiles
  - no - proved undecidable in 1970 by Yuri Matiyasevich

Church-Turing thesis (1938-48)

- theoretical model for mechanical computation (decidability)
  - two equivalent models: Turing machine and lambda calculus
- doesn't describe how to build one
  - Turing machine
    - *one-way infinite paper tape of ones and zeros*
    - *in one step: read current position, move tape, write to tape, or change internal state*

von Neumann architecture (1945-46)

- model for constructing a stored-program computer
- sequence of instructions held in a store
  - fetch instruction, load data, simple operation, store, repeat

Early computers

discrete (digital) computers

- abacus [3000 BCE]
- Pascal tax collector [1642]
- Babbage's Analytical Engine [1837]
  - steam powered computer, but never built
- Babbage's Difference Engine [1847-49]
  - mechanical trig and log tables
- Enigma and code breakers
- relay computers

analog computers

- fluids
- electrical current
- slide rule

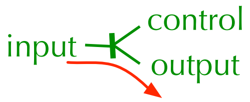
The transistor

key invention enabling electronic computers

- transistor
  - John Bardeen, Walter Brattain, and William Shockley [Bell Labs, 1947]
- integrated circuit (microchip)
  - Jack Kilby [TI 1958], Robert Noyce [Fairchild Semiconductor, 1958]

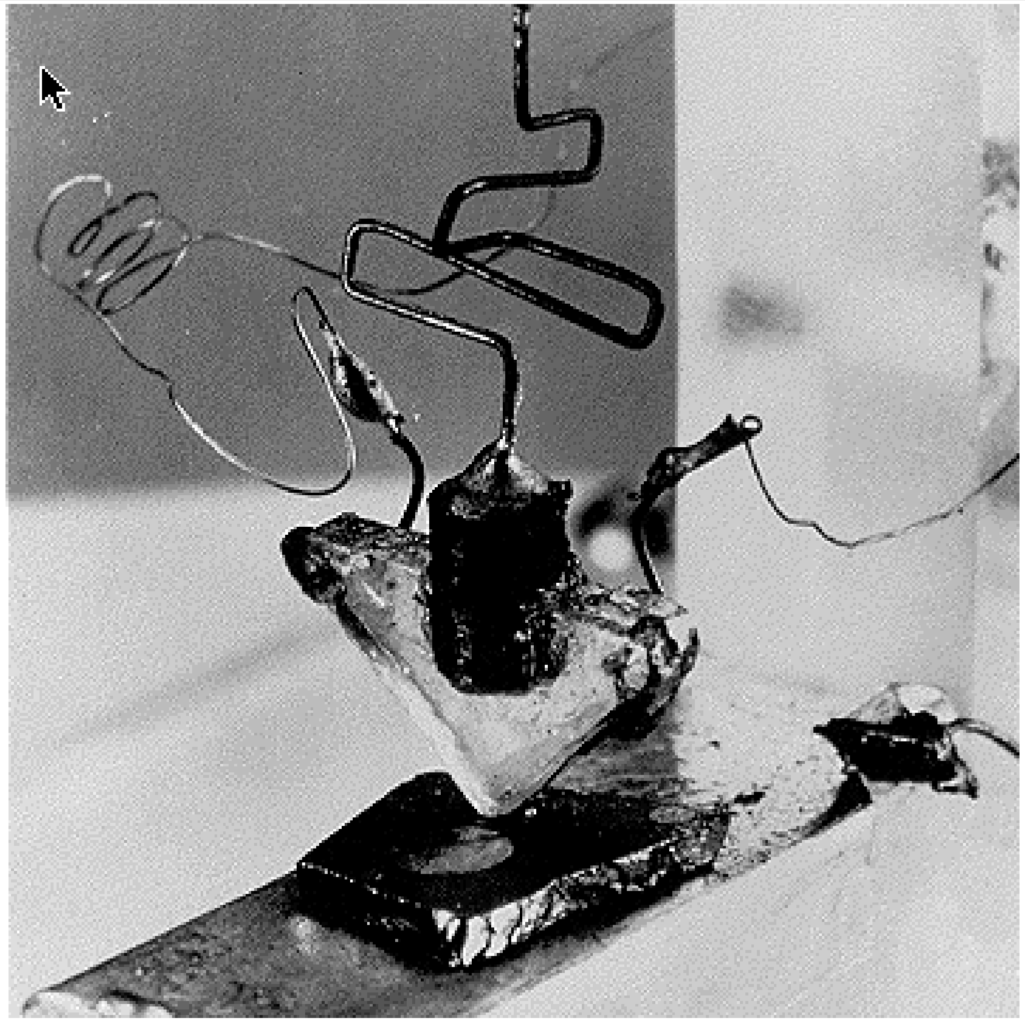
transistor as a switch

- control input opens and closes switch
  - high voltage closes switch: input flows to output
  - low voltage opens switch: output disconnected



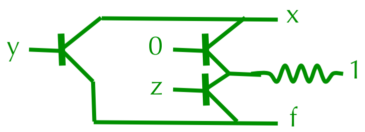
physically

- semiconductor that conducts only when charged
  - control charges the semiconductor
  - the input passes through to output iff it is conducting
- many different implantations
  - e.g., TTL, ECL, and CMOS



Using transistors to compute

simple functions are easy



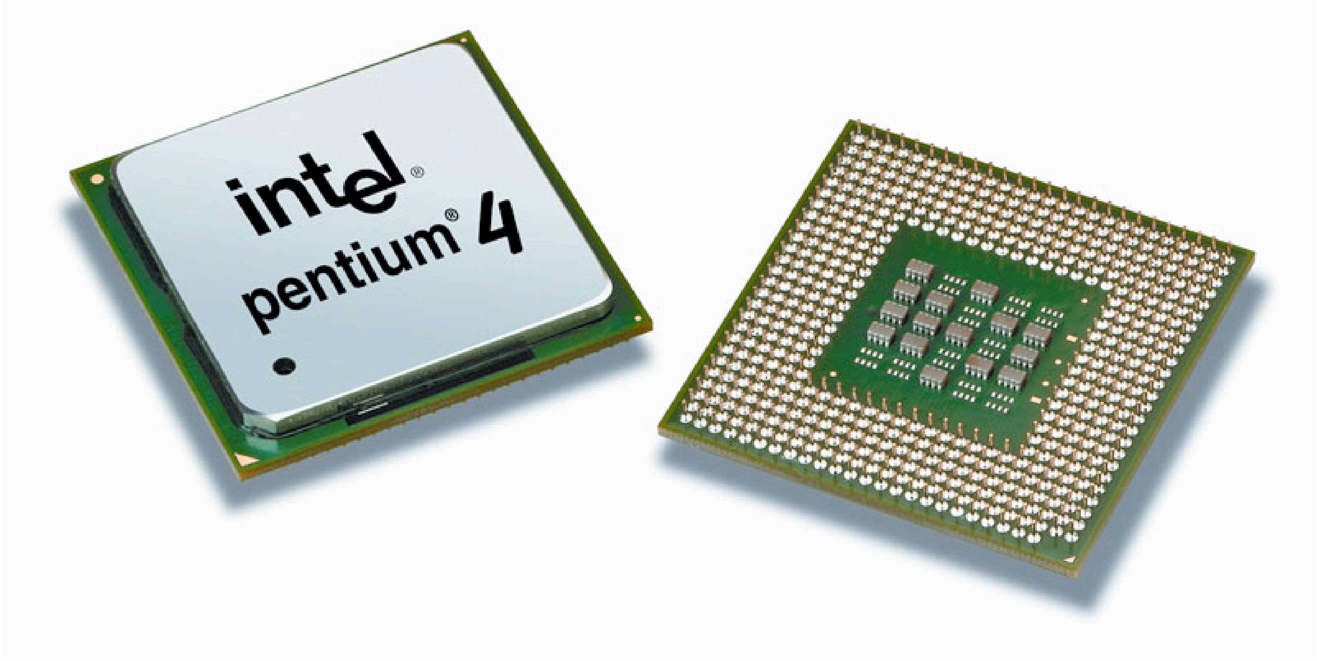
- the function f(x,y,z)
  - if x==1 then f=y else f=z
- three transistors and a resistor

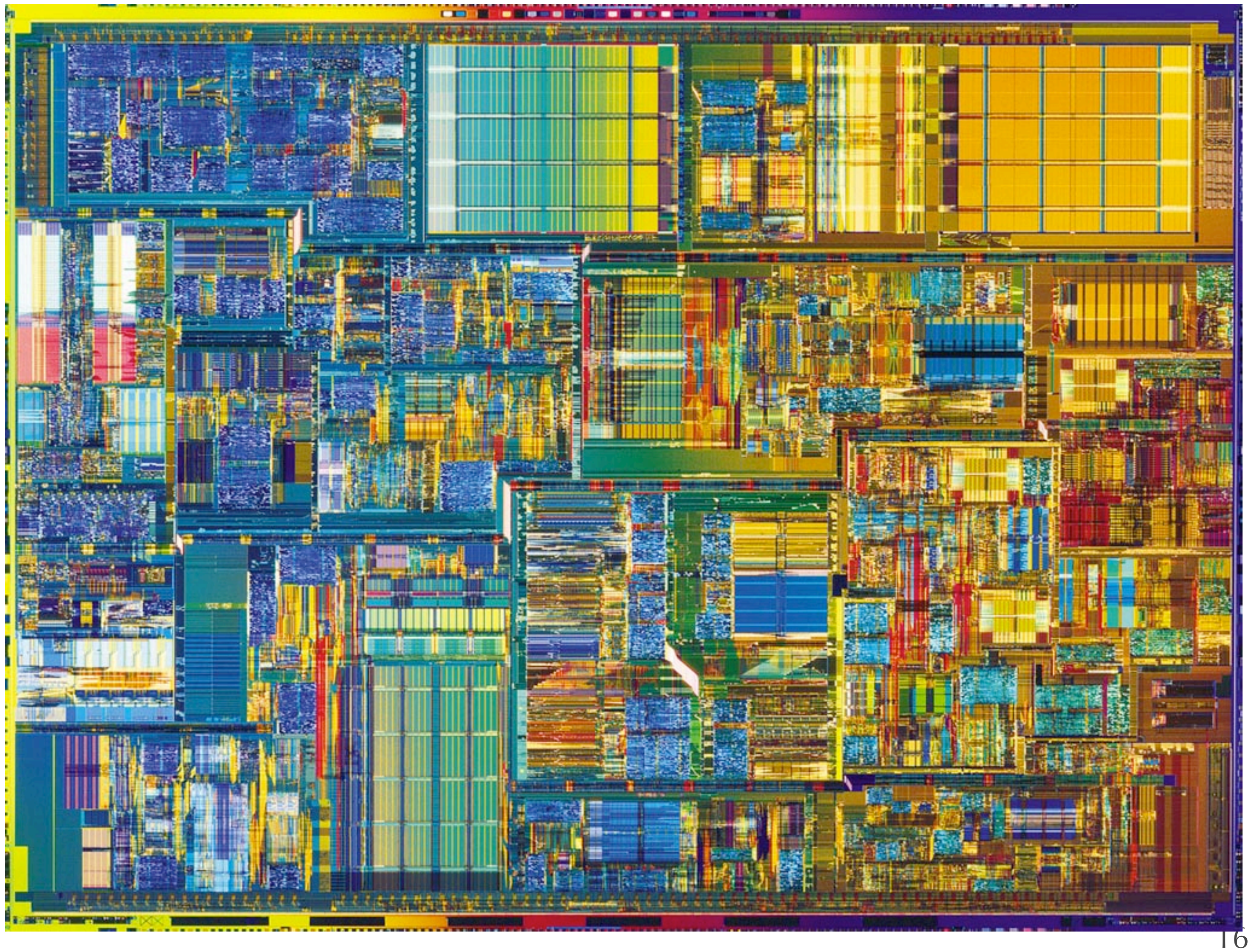
processors are made of transistors

- transistors, wires, resistors and capacitors
  - Pentium 4
    - *Willamette: 42 million transistors at 180 nm, 1,75 V, aluminum conductor*
    - *Northwood: 55 million transistors, at 130 nm, 1.5 V, copper conductor*
  - memory chip has 256 million
- on a chip
  - silicon wafer substrate
  - layers of conductor, insulator and semiconductor applied photographically
    - *P-III used 21 masks, 1 silicon layer and 6 metal (aluminum)*
  - cut from wafer, lead-stitched and encased in plastic

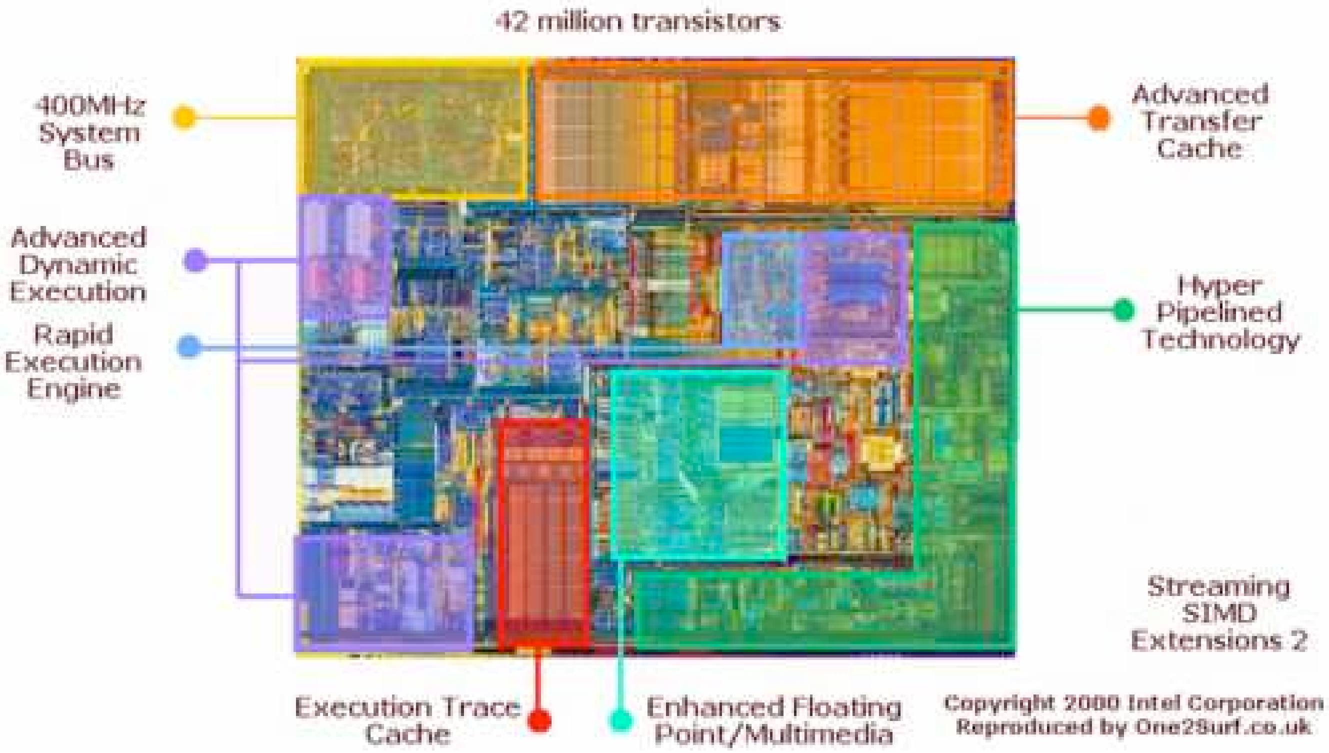
multi-million transistor functions are ... hard!

- we need an abstraction





The Intel® Pentium® 4 Processor


42 million transistors
400MHz System Bus
Advanced Transfer Cache
Advanced Dynamic Execution
Rapid Execution Engine
Hyper Pipelined Technology
Streaming SIMD Extensions 2
Execution Trace Cache
Enhanced Floating Point/Multimedia

Copyright 2000 Intel Corporation
Reproduced by One2Surf.co.uk

Boolean algebra

algebra on two-valued (binary) variables

- G. Boole [1850] and C. Shannon [1938]
- straight-forward mapping to transistor-switches
  - high voltage => 1
  - low voltage => 0

operators (lowest to highest precedence)

- or
  - **x + y** = 1 if and only if either x or y is 1
- and
  - **x • y = xy** = 1 iff x and y are both 1
- complement
  - **x′** = 1 iff x is 0

example

- f = x + y´z

Truth tables

a truth table is

- a way to represent a boolean function
- useful for optimizing and sometime for designing functions

it is written as a two dimensional array with

- column for each input variable
- row for all possible input values
- column for resulting function value

for example

- f = x + y′z

x y z	y´z	F
0 0 0	0	0
0 0 1	1	1
0 1 0	0	0
0 1 1	0	0
1 0 0	0	1
1 0 1	1	1
1 1 0	0	1
1 1 1	0	1

Axioms

you know an algebra with axioms like

- identity and zero
  - x + 0 = x
  - x * 0 = 0
- commutative
  - x + y = y + x
- associative
  - x + (y + z) = (x + y) + z
- distributive
  - x (y + z) = x y + x z

axioms of boolean algebra

- same basic stuff
- plus DeMorgans theorems

Boolean algebra axioms (I)

identity	x + 0	x
	x • 1	x
zero	x + 1	1
	x • 0	0
idempotence	x + x	x
	x • x	x
complement	x + x′	1
	x • x′	0
commutative	x + y	y + x
	x • y	y • x
associative	x + (y + z)	(x + y) + z
	x • (y • z)	(x • y) • z
distributive	x • (y + z)	xy + xz
	x + (y • z)	(x + y)(x + z)

notice
every axiom has a dual
replacing +,0 with •,1

Boolean algebra axioms (II)

DeMorgan’s theorem for complementing a function

- complement variables
- change ANDs or ORs and ORs to ANDs

DeMorgan’s axioms

$(x + y)'$	$x'y'$	called a NOR
$(xy)'$	$x' + y'$	call a NAND

Boolean algebra axioms

identity	$x + 0$	=	x
	$x \bullet 1$	=	x
zero	$x + 1$	=	1
	$x \bullet 0$	=	0
idempotence	$x + x$	=	x
	$x \bullet x$	=	x
complement	$x + x'$	=	1
	$x \bullet x'$	=	0
commutative	$x + y$	=	$y + x$
	$x \bullet y$	=	$y \bullet x$
associative	$x + (y + z)$	=	$(x + y) + z$
	$x \bullet (y \bullet z)$	=	$(x \bullet y) \bullet z$
distributive	$x \bullet (y + z)$	=	$xy + xz$
	$x + (y \bullet z)$	=	$(x + y)(x + z)$
DeMorgan’s	$(x + y)'$	=	$x'y'$
	$(xy)'$	=	$x' + y'$

x and y can be either
a binary variable or
a boolean function

Simplification using axioms

simplification is

- process of transforming function by repeated application of axioms
- to yield an equivalent new function with fewer boolean operators
- useful because, as we’ll see soon, each operator requires transistors

examples ...

- $f = ab' + c'd + ab' + c'd$
- $f = abc + abc' + a'c$
- $f = ab + a(cd + cd')$

Proof using axioms

axioms can also be used to prove new theorems

- theorem is of the form *f* = *f′*
- prove by using simplification of *f* to yield *f′*
- useful because these higher-level abstractions simplify simplification

examples ...

- $(a + b)'(a' + b')' = 0$
- absorptive law: $a + ab = a$

how do we “prove” axioms?

- inspection of truth tables

Summary

goal is to explain what a computer is

- taking deconstructionist approach
- starting with transistors and working up to complete PC system

transistor

- implements an electronically controlled switch
- on integrated circuit --- lots very small ones

circuits of transistors can be described three ways

- boolean algebra
  - algebra of binary variables
- truth table
  - expressing some functions (easier than algebra)
  - optimizing functions before implementing with transistors
- logic diagram
  - next
- can convert from any one to any of the others

Administrative stuff

web page

- www.ugrad.cs.ubc.ca/cs218
- course info and lecture notes
  - contact information for TAs and me
  - handout form 10 PM day before class
  - final form of notes replace handout form on web after class
- WebCT
  - supplemental problems, answers to textbook problems, grades
  - **discussion bboard**

office hours

- MW 2-3 in CISR 339 or by appointment

approximate grading scheme

- midterm=20%, final=50%
- labs=10%
- problem sets=18%
- group participation=2%