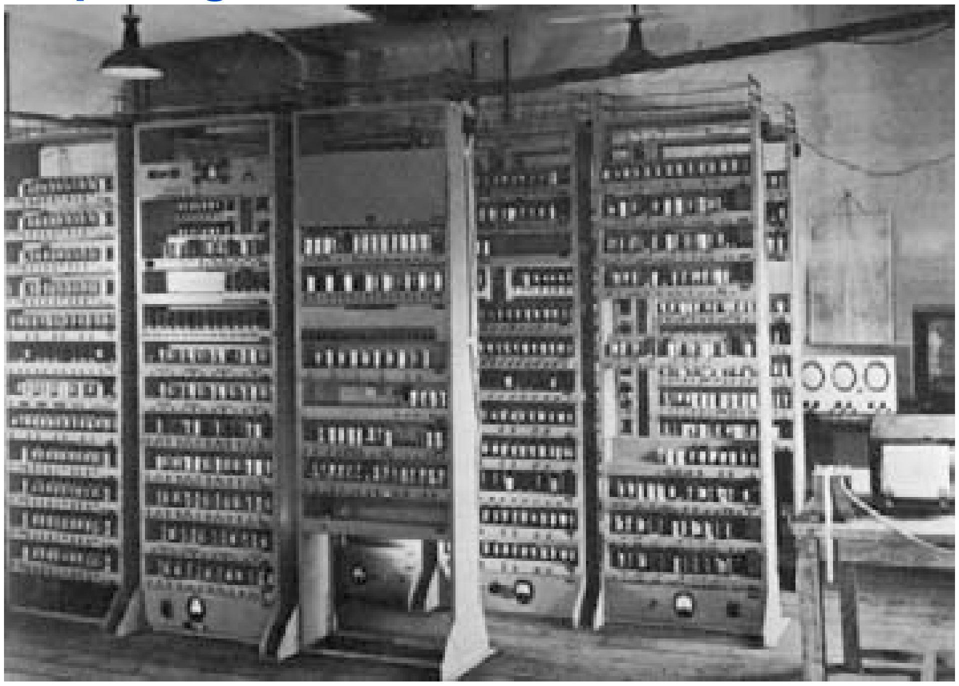
CS 152 Computer Architecture and Engineering

Lecture 1 - Introduction

Krste Asanovic
Electrical Engineering and Computer Sciences
University of California at Berkeley

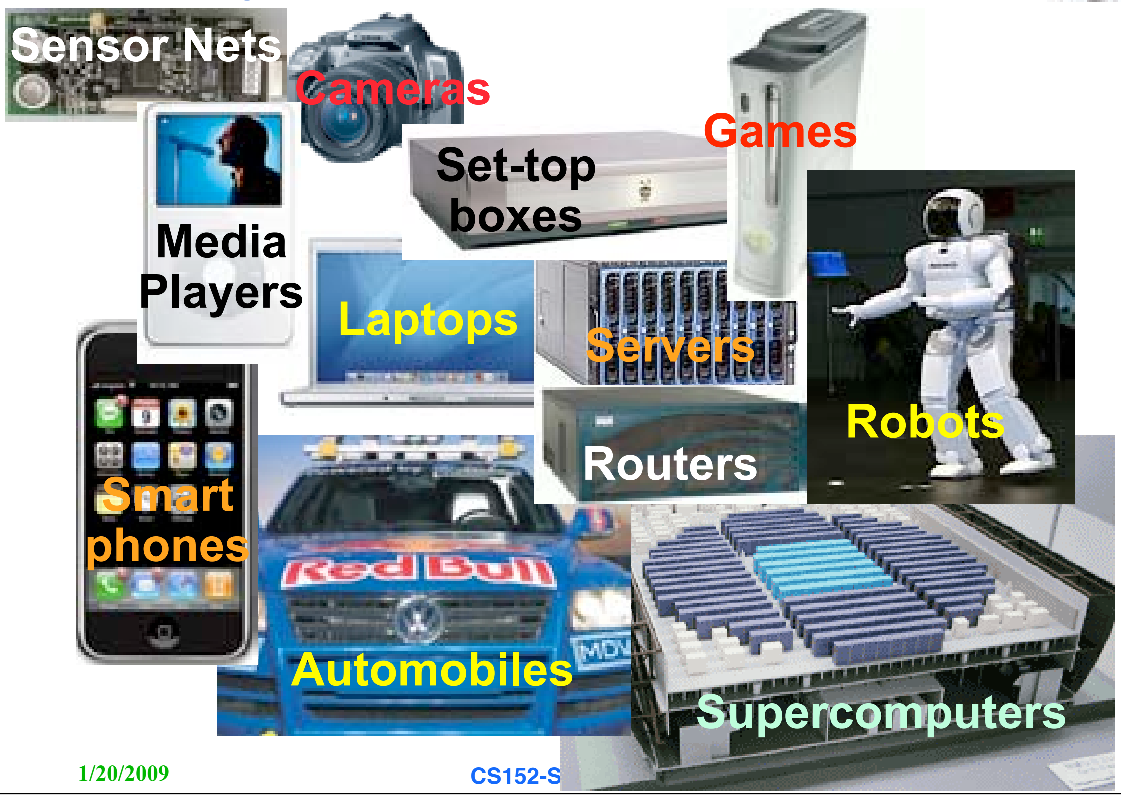
`http://www.eecs.berkeley.edu/~krste`
`http://inst.eecs.berkeley.edu/~cs152`

Computing Devices Then…

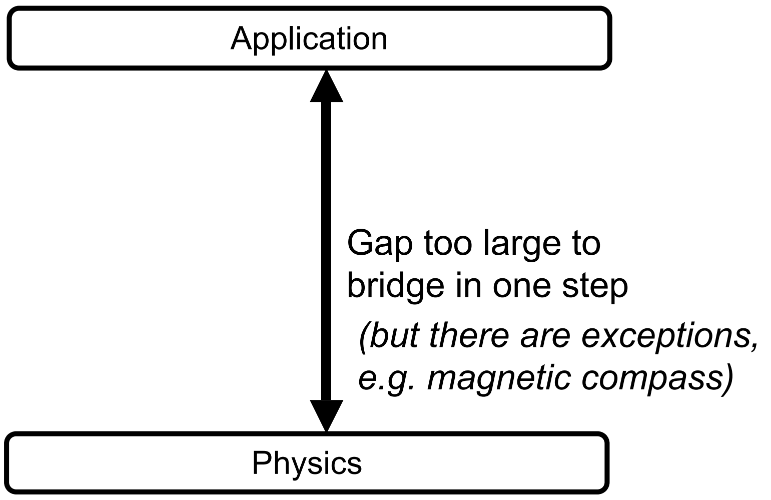


EDSAC, University of Cambridge, UK, 1949

Computing Devices Now

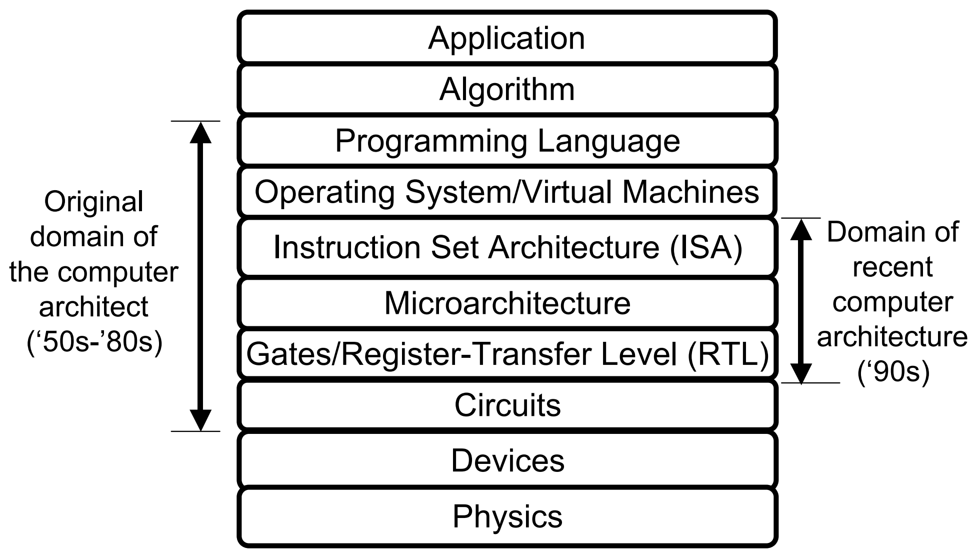


What is Computer Architecture?

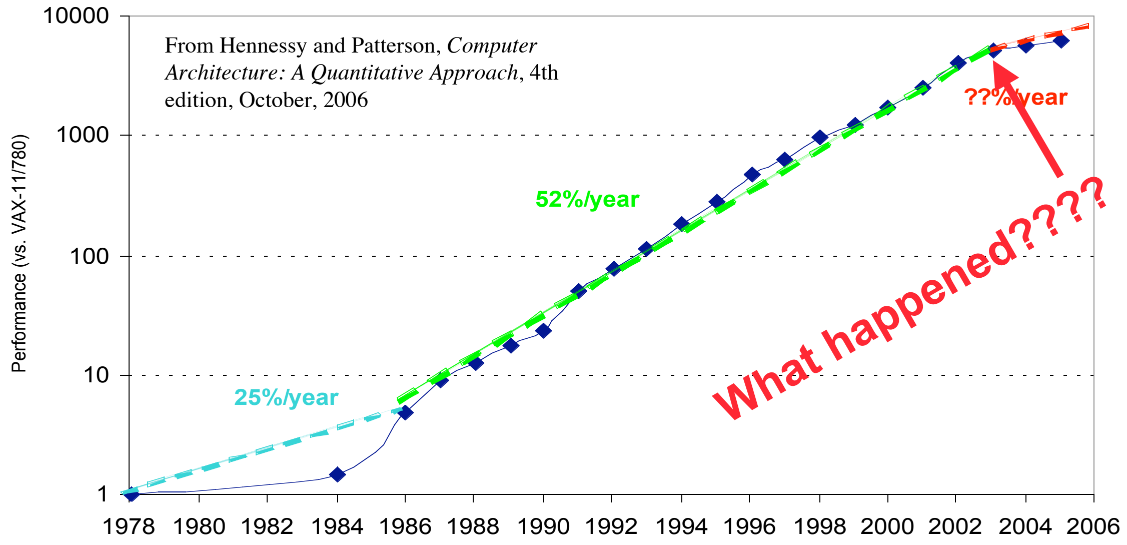

Application
Gap too large to bridge in one step
(but there are exceptions, e.g. magnetic compass)
Physics

In its broadest definition, computer architecture is the *design of the abstraction layers* that allow us to implement information processing applications efficiently using available manufacturing technologies.

Abstraction Layers in Modern Systems


Application
Algorithm
Programming Language
Operating System/Virtual Machines
Instruction Set Architecture (ISA)
Microarchitecture
Gates/Register-Transfer Level (RTL)
Circuits
Devices
Physics
Original domain of the computer architect ('50s-'80s)
Domain of recent computer architecture ('90s)

Uniprocessor Performance



- VAX : 25%/year 1978 to 1986
- RISC + x86: 52%/year 1986 to 2002
- RISC + x86: ??%/year 2002 to present

The End of the Uniprocessor Era

Single biggest change in the history of computing systems

Conventional Wisdom in Computer Architecture

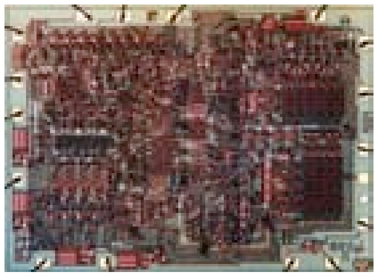
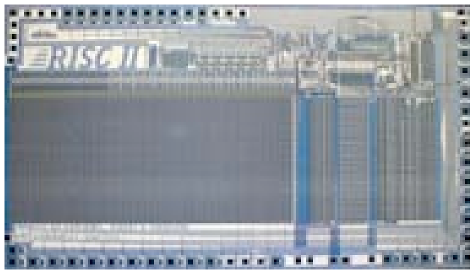
- Old Conventional Wisdom: Power is free, Transistors expensive
- New Conventional Wisdom: "Power wall" Power expensive, Transistors free (Can put more on chip than can afford to turn on)
- Old CW: Sufficient increasing Instruction-Level Parallelism via compilers, innovation (pipelining, superscalar, out-of-order, speculation, VLIW, …)
- New CW: "ILP wall" law of diminishing returns on more HW for ILP
- Old CW: Multiplies are slow, Memory access is fast
- New CW: "Memory wall" Memory slow, multiplies fast (200 clock cycles to DRAM memory, 4 clocks for multiply)
- Old CW: Uniprocessor performance 2X / 1.5 yrs
- New CW: Power Wall + ILP Wall + Memory Wall = Brick Wall
  - Uniprocessor performance now 2X / 5(?) yrs

  ⇒ Sea change in chip design: multiple "cores" (2X processors per chip / ~ 2 years)

  » More, simpler processors are more power efficient

Sea Change in Chip Design

- Intel 4004 (1971): 4-bit processor, 2312 transistors, 0.4 MHz, 10 micron PMOS, 11 mm^2 chip
- RISC II (1983): 32-bit, 5 stage pipeline, 40,760 transistors, 3 MHz, 3 micron NMOS, 60 mm^2 chip
- 125 mm^2 chip, 0.065 micron CMOS = **2312 RISC II+FPU+Icache+Dcache**
  - RISC II shrinks to ~ 0.02 mm^2 at 65 nm
  - Caches via DRAM or 1 transistor SRAM?

- **Processor is the new transistor?**

Déjà vu all over again?

- Multiprocessors imminent in 1970s, '80s, '90s, …
- "… today's processors … are nearing an impasse as technologies approach the speed of light.."

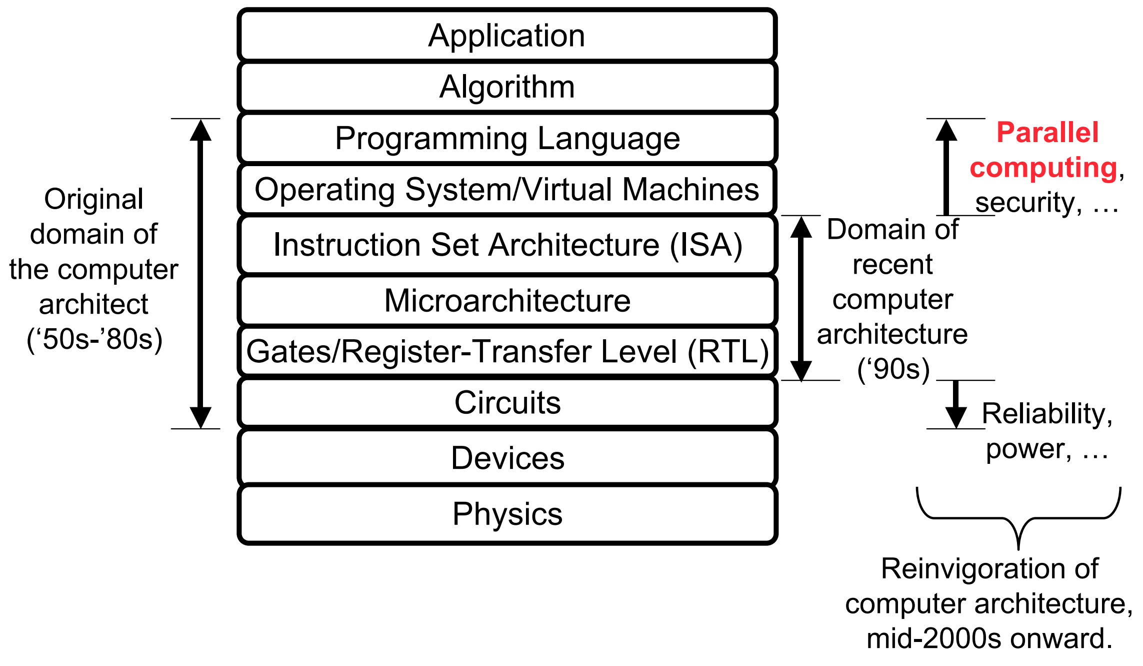
  David Mitchell, *The Transputer: The Time Is Now* (1989)
- Transputer was premature
  ⇒ Custom multiprocessors tried to beat uniprocessors
  ⇒ Procrastination rewarded: 2X seq. perf. / 1.5 years
- "We are dedicating all of our future product development to multicore designs. … This is a sea change in computing"

  Paul Otellini, President, Intel (2004)
- Difference is all microprocessor companies have switched to multiprocessors (AMD, Intel, IBM, Sun; all new Apples 2+ CPUs)
  ⇒ Procrastination penalized: 2X sequential perf. / 5 yrs
  ⇒ Biggest programming challenge: from 1 to 2 CPUs

Problems with Sea Change

- Algorithms, Programming Languages, Compilers, Operating Systems, Architectures, Libraries, … not ready to supply Thread-Level Parallelism or Data-Level Parallelism for 1000 CPUs / chip,
- Architectures not ready for 1000 CPUs / chip
  - Unlike Instruction-Level Parallelism, cannot be solved by computer architects and compiler writers alone, but also cannot be solved without participation of architects
- Need a reworking of all the abstraction layers in the computing system stack

Abstraction Layers in Modern Systems

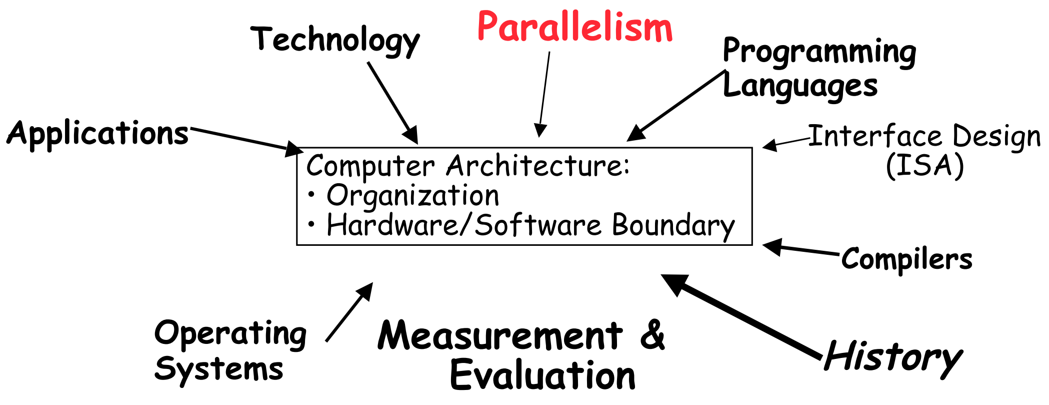

Application
Algorithm
Programming Language
Operating System/Virtual Machines
Instruction Set Architecture (ISA)
Microarchitecture
Gates/Register-Transfer Level (RTL)
Circuits
Devices
Physics
Original domain of the computer architect ('50s-'80s)
Domain of recent computer architecture ('90s)
Parallel computing, security, …
Reliability, power, …
Reinvigoration of computer architecture, mid-2000s onward.

The New CS152

- New CS152 focuses on interaction of software and hardware
  - more architecture and less digital engineering.
- No FPGA design component
  - There is now a separate FPGA design lab class (CS194 in Fall 2008), where you can try building some of the architectural ideas we'll explore this semester (100% digital engineering)
- Much of the material you'll learn this term was previously in CS252
  - Some of the current CS61C, I first saw in CS252 nearly 20 years ago!
  - Maybe every 10 years, shift CS252->CS152->CS61C?
- Class contains labs based on various different machine designs
  - Experiment with how architectural mechanisms work in practice on real software.

CS 152 Course Focus

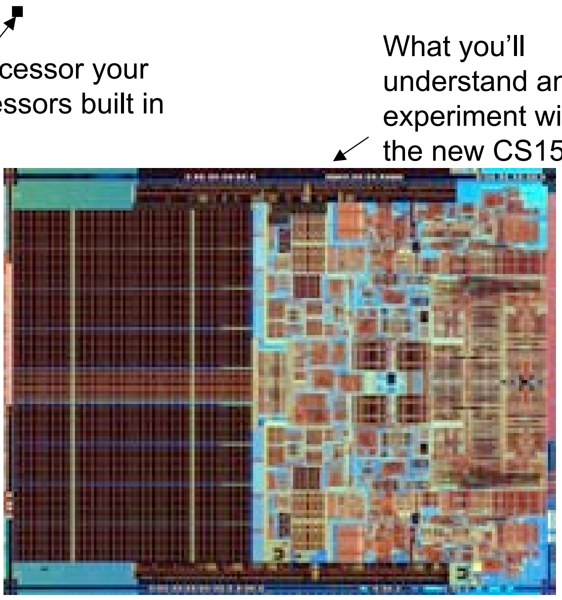
Understanding the design techniques, machine structures, technology factors, evaluation methods that will determine the form of computers in 21st Century

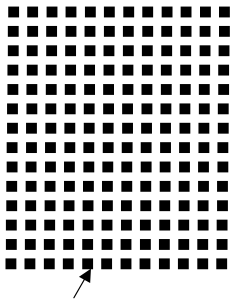


The New CS152 Executive Summary

The processor your predecessors built in CS152

What you'll understand and experiment with in the new CS152





Plus, the technology behind chip-scale multiprocessors (CMPs)

CS152 Administrivia

Instructor:	Prof. Krste Asanovic
	Office: 579 Soda Hall, **krste@eecs**
	Office Hours: M 1:30-2:30PM (email to confirm), 579 Soda
T. A.:	Scott Beamer, **sbeamer@eecs**
	Office Hours: TBD
Lectures:	Tu/Th, 5:00-6:30PM, 320 Soda (may change)
Section:	F 12-1pm, 258 Dwinelle (room may change)
Text:	*Computer Architecture: A Quantitative Approach,*
	4th Edition (Oct, 2006)
	Readings assigned from this edition, don't use earlier Eds.
Web page:	http://inst.eecs.berkeley.edu/~cs152
	Lectures available online before noon, day of lecture

CS152 Structure and Syllabus

Six modules

1. Simple machine design (ISAs, microprogramming, unpipelined machines, Iron Law, simple pipelines)
2. Memory hierarchy (DRAM, caches, optimizations)
3. Virtual memory systems, exceptions, interrupts
4. Complex pipelining (score-boarding, out-of-order issue)
5. Explicitly parallel processors (vector machines, VLIW machines, multithreaded machines)
6. Multiprocessor architectures (cache coherence, memory models, synchronization)

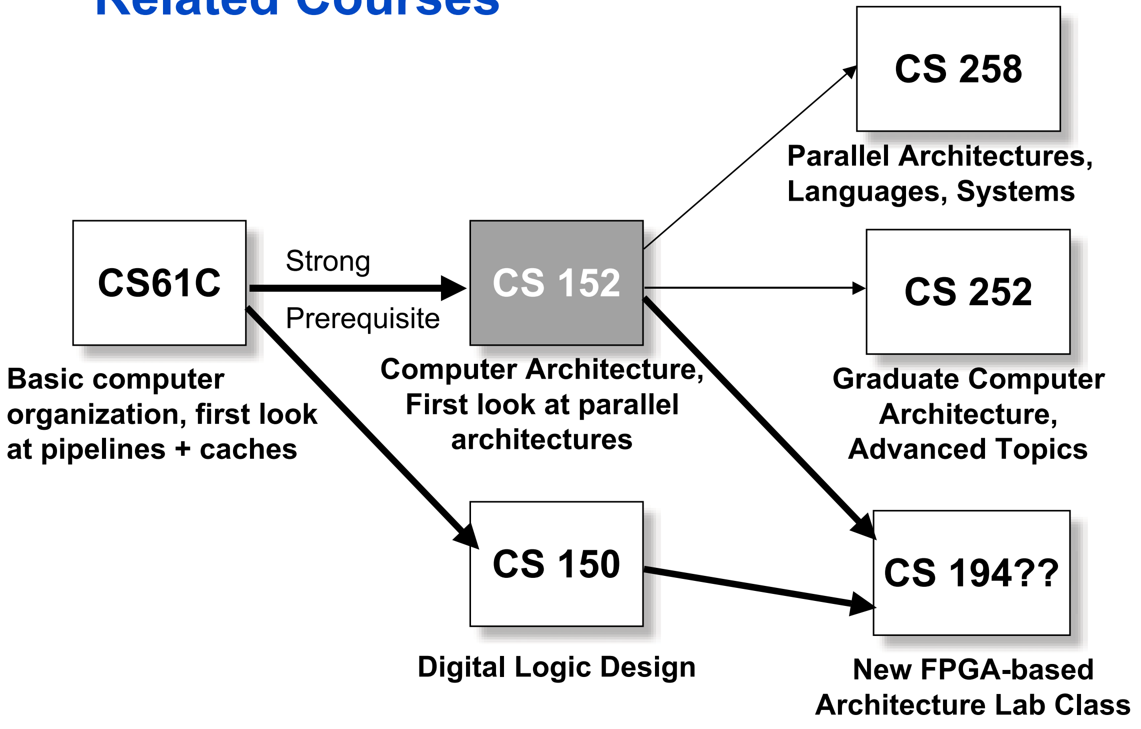
CS152 Course Components

- 20% Problem Sets (one per module)
  - Intended to help you learn the material. Feel free to discuss with other students and instructors, but must turn in your own solutions. Grading based mostly on effort, but quizzes assume that you have worked through all problems. Solutions released after PSs handed in.
- 40% Quizzes (one per module)
  - In-class, closed-book, no calculators or computers.
  - Based on lectures, problem sets, and labs
- 40% Labs (one per module)
  - Labs use advanced full system simulators (Virtutech Simics)
  - Directed plus open-ended sections to each lab

CS152 Labs

- Each lab has directed plus open-ended assignments
  - Roughly 50/50 split of grade
- Directed portion is intended to ensure students learn main concepts behind lab
  - Each student must perform own lab and hand in their own lab report
- Open-ended assigment is to allow you to show your creativity
  - Roughly a one day "mini-project"
    - » E.g., try an architectural idea and measure potential, negative results OK (if explainable!)
  - Students can work individually or in groups of two or three
  - Group open-ended lab reports must be handed in separately
  - Students can work in different groups for different assignments

Related Courses


CS 258
Parallel Architectures, Languages, Systems
CS61C
Strong
Prerequisite
CS 152
CS 252
Basic computer organization, first look at pipelines + caches
Computer Architecture, First look at parallel architectures
Graduate Computer Architecture, Advanced Topics
CS 150
CS 194??
Digital Logic Design
New FPGA-based Architecture Lab Class

Computer Architecture: A Little History

Throughout the course we'll use a historical narrative to help understand why certain ideas arose

Why worry about old ideas?

- Helps to illustrate the design process, and explains why certain decisions were taken
- Because future technologies might be as constrained as older ones
- Those who ignore history are doomed to repeat it
  - Every mistake made in mainframe design was also made in minicomputers, then microcomputers, where next?

Charles Babbage 1791-1871

Lucasian Professor of Mathematics, Cambridge University, 1827-1839

Charles Babbage

- *Difference Engine* 1823
- *Analytic Engine* 1833
  - The forerunner of modern digital computer!

Application
- Mathematical Tables – Astronomy
- Nautical Tables – Navy

Background
- Any continuous function can be approximated by a polynomial --- *Weierstrass*

Technology
- mechanical - gears, Jacquard's loom, simple calculators

Difference Engine
A machine to compute mathematical tables

Weierstrass:
- Any continuous function can be approximated by a polynomial
- Any polynomial can be computed from *difference* tables

An example

f(n)	$= n^2 + n + 41$
d1(n)	$= f(n) - f(n-1) = 2n$
d2(n)	$= d1(n) - d1(n-1) = 2$
f(n)	$= f(n-1) + d1(n) = f(n-1) + (d1(n-1) + 2)$

all you need is an adder!

n	0	1	2	3	4
d2(n)			2	2	2
d1(n)		2	4	6	8
f(n)	41	43	47	53	61

Difference Engine

1823
- Babbage's paper is published

1834
- The paper is read by Scheutz & his son in Sweden

1842
- Babbage gives up the idea of building it; he is onto Analytic Engine!

1855
- Scheutz displays his machine at the Paris World Fare
- Can compute any 6th degree polynomial
- *Speed:* 33 to 44 32-digit numbers per minute!

Now the machine is at the Smithsonian

Analytic Engine

1833: Babbage's paper was published
- *conceived during a hiatus in the development of the difference engine*

Inspiration: *Jacquard Looms*
- looms were controlled by punched cards
  - » The set of cards with fixed punched holes dictated the pattern of weave $\Rightarrow$ *program*
  - » The same set of cards could be used with different colored threads $\Rightarrow$ *numbers*

1871: Babbage dies
- The machine remains unrealized.

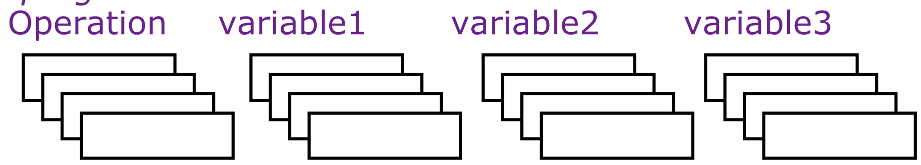
It is not clear if the analytic engine could be built even today using only mechanical technology

Analytic Engine

The first conception of a general-purpose computer

1. The *store* in which all variables to be operated upon, as well as all those quantities which have arisen from the results of the operations are placed.
2. The *mill* into which the quantities about to be operated upon are always brought.

The *program*



An operation in the *mill* required feeding two punched cards and producing a new punched card for the *store.*

An operation to alter the sequence was also provided!

The first programmer

Ada Byron *aka* "Lady Lovelace" 1815-52

Ada's tutor was Babbage himself!

Babbage’s Influence

- Babbage’s ideas had great influence later primarily because of
  - *Luigi Menabrea,* who published notes of Babbage’s lectures in Italy
  - *Lady Lovelace,* who translated Menabrea’s notes in English and thoroughly expanded them.
    “... Analytic Engine weaves *algebraic patterns*....”

- In the early twentieth century - the focus shifted to analog computers but
  - *Harvard Mark I built in 1944 is very close in spirit to the Analytic Engine.*

Harvard Mark I

- Built in 1944 in IBM Endicott laboratories
  - Howard Aiken – Professor of Physics at Harvard
  - Essentially mechanical but had some electro-magnetically controlled relays and gears
  - Weighed *5 tons* and had *750,000* components
  - A synchronizing clock that beat every *0.015* seconds (66Hz)

Performance:

0.3	seconds for addition
6	seconds for multiplication
1	minute for a sine calculation

Broke down once a week!

Linear Equation Solver
John Atanasoff, Iowa State University

1930's:
- Atanasoff built the Linear Equation Solver.
- It had 300 tubes!
- Special-purpose binary digital calculator
- Dynamic RAM (stored values on refreshed capacitors)

Application:
- Linear and Integral differential equations

Background:
- Vannevar Bush's Differential Analyzer
  --- *an analog computer*

Technology:
- Tubes and Electromechanical relays

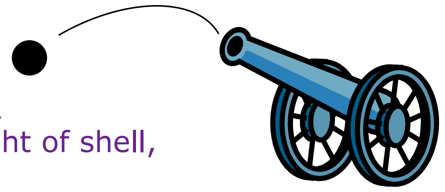
Atanasoff decided that the correct mode of computation was using electronic binary digits.

Electronic Numerical Integrator and Computer (ENIAC)

- Inspired by Atanasoff and Berry, Eckert and Mauchly designed and built ENIAC (1943-45) at the University of Pennsylvania
- The first, completely electronic, operational, general-purpose analytical calculator!
  - 30 tons, 72 square meters, 200KW
- Performance
  - Read in 120 cards per minute
  - Addition took 200 μs, Division 6 ms
  - 1000 times faster than Mark I
- Not very reliable!

WW-2 Effort

Application: Ballistic calculations



angle = f (location, tail wind, cross wind,
air density, temperature, weight of shell,
propellant charge, ...)

Electronic Discrete Variable Automatic Computer (EDVAC)

- ENIAC's programming system was external
  - Sequences of instructions were executed independently of the results of the calculation
  - Human intervention required to take instructions "out of order"
- Eckert, Mauchly, John von Neumann and others designed EDVAC (1944) to solve this problem
  - Solution was the *stored program computer*

    ⇒ *"program can be manipulated as data"*
- *First Draft of a report on EDVAC* was published in 1945, but just had von Neumann's signature!
  - In 1973 the court of Minneapolis attributed the honor of *inventing the computer* to John Atanasoff

Stored Program Computer

Program = A sequence of instructions

How to control instruction sequencing?

manual control	calculators
automatic control	
external (paper tape)	Harvard Mark I , 1944
	Zuse's Z1, WW2
internal	
plug board	ENIAC 1946
read-only memory	ENIAC 1948
read-write memory	EDVAC 1947 *(concept)*

- The same storage can be used to store program and data

EDSAC 1950 Maurice Wilkes

Technology Issues

ENIAC $\Rightarrow$ EDVAC

ENIAC	EDVAC
18,000 tubes	4,000 tubes
20 10-digit numbers	2000 word storage
	mercury delay lines

ENIAC had many asynchronous parallel units but only one was active at a time

BINAC : Two processors that checked each other for reliability.

Didn't work well because processors never agreed

Dominant Problem: *Reliability*

Mean time between failures (MTBF)

MIT's Whirlwind with an MTBF of 20 min. was perhaps the most reliable machine !

Reasons for unreliability:

1. Vacuum Tubes

2. Storage medium
   - acoustic delay lines
   - mercury delay lines
   - Williams tubes
   - Selections

Reliability solved by invention of Core memory by J. Forrester 1954 at MIT for Whirlwind project

Commercial Activity: 1948-52

IBM's SSEC (follow on from Harvard Mark I)

Selective Sequence Electronic Calculator

- 150 word store.
- Instructions, constraints, and tables of data were read from paper tapes.
- 66 Tape reading stations!
- Tapes could be glued together to form a loop!
- Data could be output in one phase of computation and read in the next phase of computation.

And then there was IBM 701

IBM 701 -- 30 machines were sold in 1953-54
used CRTs as main memory, 72 tubes of 32x32b each

IBM 650 -- a cheaper, drum based machine,
more than 120 were sold in 1954
and there were orders for 750 more!

Users stopped building their own machines.

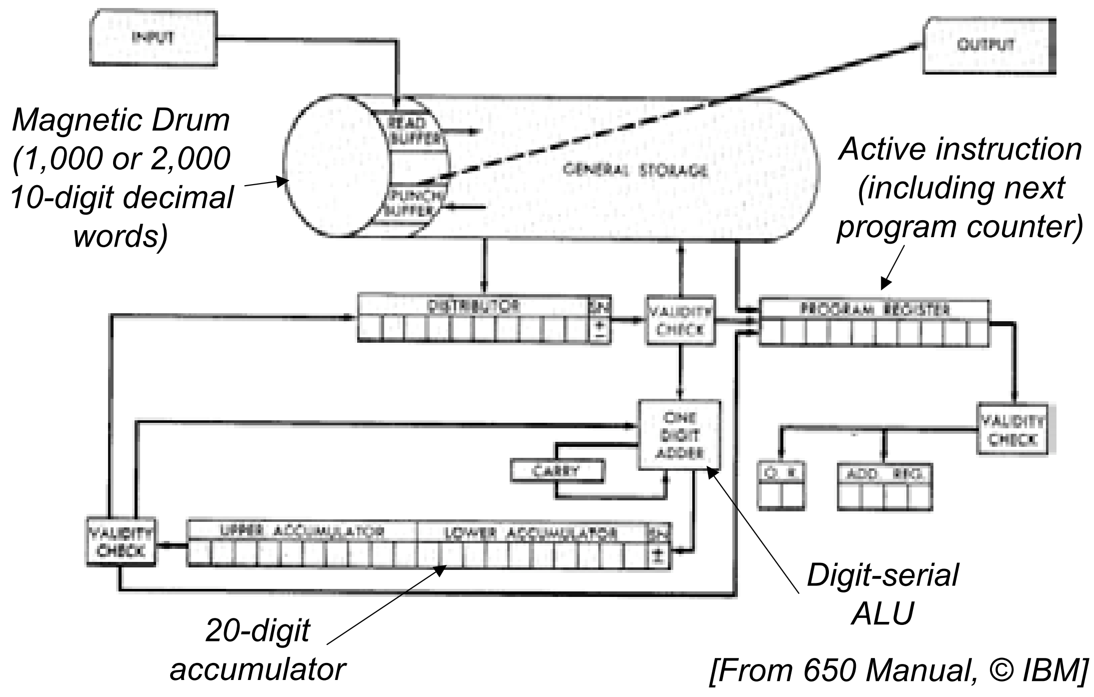
Why was IBM late getting into computer technology?

IBM was making too much money!
Even without computers, IBM revenues were doubling every 4 to 5 years in 40's and 50's.

Computers in mid 50's

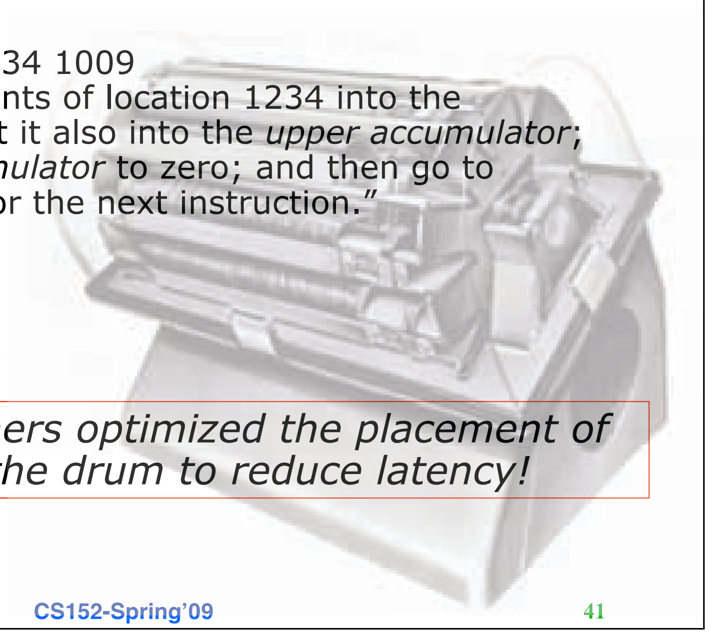
- Hardware was expensive
- Stores were small (1000 words)
  - ⇒ No resident system software!
- Memory access time was 10 to 50 times slower than the processor cycle
  - ⇒ Instruction execution time was totally dominated by the *memory reference time*.
- The *ability to design complex control circuits* to execute an instruction was the central design concern as opposed to *the speed* of decoding or an ALU operation
- Programmer's view of the machine was inseparable from the actual hardware implementation

The IBM 650 (1953-4)



[From 650 Manual, © IBM]

Programmer's view of the IBM 650



A drum machine with 44 instructions

Instruction: 60 1234 1009

- "Load the contents of location 1234 into the *distribution*; put it also into the *upper accumulator*; set *lower accumulator* to zero; and then go to location 1009 for the next instruction."

Good programmers optimized the placement of instructions on the drum to reduce latency!

The Earliest Instruction Sets

Single Accumulator - A carry-over from the calculators.

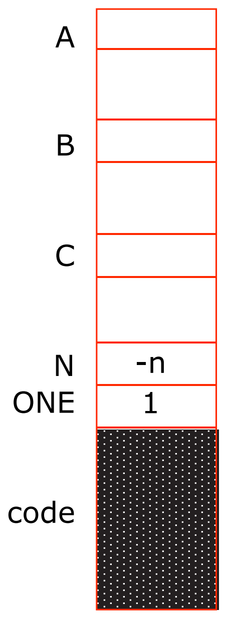
LOAD	x	AC ← M[x]
STORE	x	M[x] ← (AC)
ADD	x	AC ← (AC) + M[x]
SUB	x	
MUL	x	Involved a quotient register
DIV	x	
SHIFT LEFT		AC ← 2 × (AC)
SHIFT RIGHT		
JUMP	x	PC ← x
JGE	x	if (AC) ≥ 0 then PC ← x
LOAD ADR	x	AC ← Extract address field(M[x])
STORE ADR	x	

Typically less than 2 dozen instructions!

Programming: Single Accumulator Machine

$\boxed{C_i \leftarrow A_i + B_i, \quad 1 \leq i \leq n}$

```
LOOP   LOAD      N
       JGE       DONE
       ADD       ONE
       STORE     N
F1     LOAD      A
F2     ADD       B
F3     STORE     C
       JUMP      LOOP
DONE   HLT
```



How to modify the addresses A, B and C ?

Self-Modifying Code

```
LOOP   LOAD        N
       JGE         DONE
       ADD         ONE
       STORE       N
F1     LOAD        A
F2     ADD         B
F3     STORE       C
       LOAD ADR    F1
       ADD         ONE
       STORE ADR   F1
       LOAD ADR    F2
       ADD         ONE
       STORE ADR   F2
       LOAD ADR    F3
       ADD         ONE
       STORE ADR   F3
       JUMP        LOOP
DONE   HLT
```

modify the program for the next iteration

$\boxed{C_i \leftarrow A_i + B_i, \quad 1 \leq i \leq n}$

Each iteration involves

	total	*book-keeping*
instruction fetches	*17*	*14*
operand fetches	*10*	*8*
stores	*5*	*4*

Index Registers

Tom Kilburn, Manchester University, mid 50's

One or more specialized registers to simplify address calculation

Modify existing instructions

```
LOAD    x, IX      AC ← M[x + (IX)]
ADD     x, IX      AC ← (AC) + M[x + (IX)]
...
```

Add new instructions to manipulate *index registers*

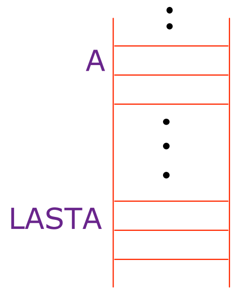
```
JZi     x, IX      if (IX)=0 then  PC ← x
                             else  IX ← (IX) + 1
LOADi   x, IX      IX ← M[x]   (truncated to fit IX)
...
```

Index registers have accumulator-like characteristics

Using Index Registers

$C_i \leftarrow A_i + B_i, \quad 1 \leq i \leq n$

```
        LOADi  -n, IX
LOOP    JZi    DONE, IX
        LOAD   LASTA, IX
        ADD    LASTB, IX
        STORE  LASTC, IX
        JUMP   LOOP
DONE    HALT
```



- *Program does not modify itself*
- *Efficiency has improved dramatically (ops / iter)*

	with index regs	without index regs
instruction fetch	5(2)	17 (14)
operand fetch	2	10 (8)
store	1	5 (4)

- *Costs:* Instructions are 1 to 2 bits longer
  Index registers with ALU-like circuitry
  Complex control

Operations on Index Registers

To increment index register by k

AC ← (IX)	*new instruction*
AC ← (AC) + k	
IX ← (AC)	*new instruction*

also the AC must be saved and restored.

It may be better to increment IX directly

INCi	k, IX	IX ← (IX) + k

More instructions to manipulate index register

STOREi	x, IX	M[x] ← (IX) (extended to fit a word)
...		

IX begins to look like an accumulator

⇒ several index registers

several accumulators

⇒ *General Purpose Registers*

Evolution of Addressing Modes

1. Single accumulator, absolute address
   LOAD x
2. Single accumulator, index registers
   LOAD x, IX
3. Indirection
   LOAD (x)
4. Multiple accumulators, index registers, indirection
   LOAD R, IX, x
   or LOAD R, IX, (x) the meaning?
   R ← M[M[x] + (IX)]
   or R ← M[M[x + (IX)]]
5. Indirect through registers
   LOAD R_I, (R_J)
6. The works
   LOAD R_I, R_J, (R_K) R_J = index, R_K = base addr

Variety of Instruction Formats

- *Two address formats:* the destination is same as one of the operand sources

(Reg × Reg) to Reg	$R_I \leftarrow (R_I) + (R_J)$
(Reg × Mem) to Reg	$R_I \leftarrow (R_I) + M[x]$

  - *x* can be specified directly or via a register
  - effective address calculation for *x* could include indexing, indirection, ...

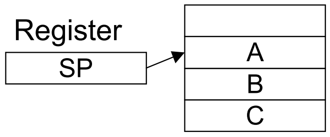
- *Three address formats:* One destination and up to two operand sources per instruction

(Reg x Reg) to Reg	$R_I \leftarrow (R_J) + (R_K)$
(Reg x Mem) to Reg	$R_I \leftarrow (R_J) + M[x]$

More Instruction Formats

- *Zero address formats:* operands on a stack

add	M[sp-1] ← M[sp] + M[sp-1]
load	M[sp] ← M[M[sp]]



  - Stack can be in registers or in memory (usually top of stack cached in registers)

- *One address formats:* Accumulator machines
  - Accumulator is always other implicit operand

Many different formats are possible!

Data Formats and Memory Addresses

Data formats:

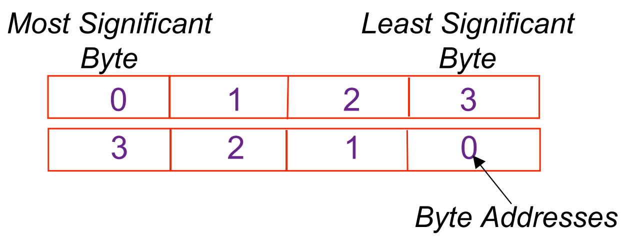
Bytes, Half words, words and double words

Some issues

- *Byte addressing*

Big Endian

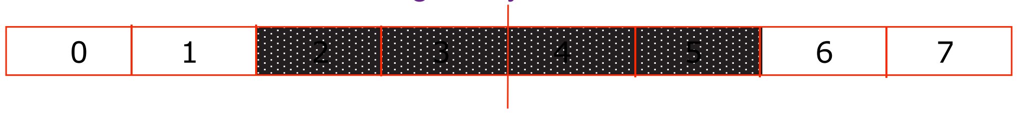
vs. Little Endian



- *Word alignment*

Suppose the memory is organized in 32-bit words.

Can a word address begin only at 0, 4, 8, ... ?



Software Developments

up to 1955 Libraries of numerical routines

- Floating point operations
- Transcendental functions
- Matrix manipulation, equation solvers, . . .

1955-60 *High level Languages* - Fortran 1956

Operating Systems -

- Assemblers, Loaders, Linkers, Compilers
- Accounting programs to keep track of usage and charges

Machines required *experienced operators*

⇒ Most users could not be expected to understand these programs, much less write them

⇒ Machines had to be sold with a lot of resident software

Compatibility Problem at IBM

By early 60's, *IBM had 4 incompatible lines of computers!*

701	→	7094
650	→	7074
702	→	7080
1401	→	7010

Each system had its own

- Instruction set
- I/O system and Secondary Storage:
  magnetic tapes, drums and disks
- assemblers, compilers, libraries,...
- market niche
  business, scientific, real time, ...

⇒ *IBM 360*

IBM 360 : Design Premises

Amdahl, Blaauw and Brooks, 1964

- The design must lend itself to *growth and successor machines*
- General method for connecting I/O devices
- Total performance - answers per month rather than bits per microsecond ⇒ *programming aids*
- Machine must be capable of *supervising itself* without manual intervention
- Built-in *hardware fault checking* and locating aids to reduce down time
- Simple to assemble systems with redundant I/O devices, memories etc. for *fault tolerance*
- Some problems required floating point words larger than 36 bits

IBM 360: *A General-Purpose Register (GPR) Machine*

- Processor State
  - 16 General-Purpose 32-bit Registers
    - » *may be used as index and base register*
    - » *Register 0 has some special properties*
  - 4 Floating Point 64-bit Registers
  - A Program Status Word (PSW)
    - » *PC, Condition codes, Control flags*
- A 32-bit machine with 24-bit addresses
  - But no instruction contains a 24-bit address!
- Data Formats
  - 8-bit bytes, 16-bit half-words, 32-bit words, 64-bit double-words

The IBM 360 is why bytes are 8-bits long today!

IBM 360: Initial Implementations

	Model 30	. . .	*Model 70*
Storage	8K - 64 KB		256K - 512 KB
Datapath	8-bit		64-bit
Circuit Delay	30 nsec/level		5 nsec/level
Local Store	Main Store		Transistor Registers
Control Store	Read only 1μsec		Conventional circuits

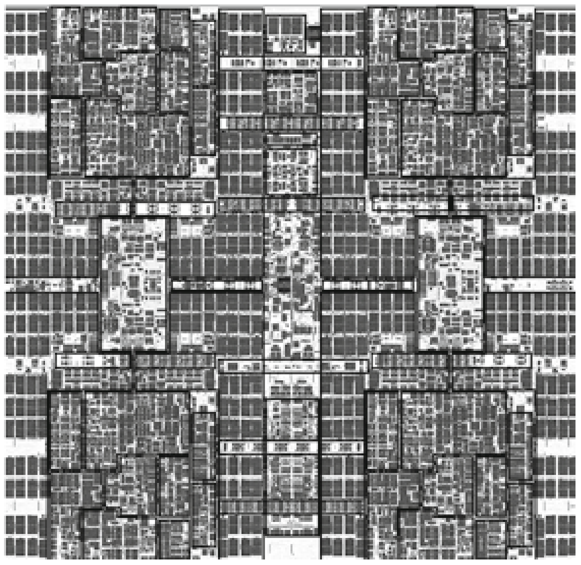
IBM 360 instruction set architecture (ISA) completely hid the underlying technological differences between various models.

Milestone: The first true ISA designed as portable hardware-software interface!

With minor modifications it still survives today!

IBM 360: 45 years later… The zSeries z10 Microprocessor

- 4.4 GHz in IBM 65nm SOI CMOS technology
- 994 million transistors in 454mm^2
- 64-bit virtual addressing
  - original S/360 was 24-bit, and S/370 was 31-bit extension
- Quad core design
- Dual-issue in-order superscalar CISC pipeline
- Out-of-order memory accesses
- Redundant datapaths
  - every instruction performed in two parallel datapaths and results compared
- 64KB L1 I-cache, 128KB L1 D-cache on-chip
- 3MB private L2 unified cache per core, on-chip
- Off-chip L3 cache of up to 48MB
- 10K-entry Branch Target Buffer
  - Very large buffer to support commercial workloads
- Hardware for decimal floating-point arithmetic
  - Important for business applications



[IEEE Micro, March 2008]

And in conclusion …

- Computer Architecture >> ISAs and RTL
- CS152 is about interaction of hardware and software, and design of appropriate abstraction layers
- Computer architecture is shaped by technology and applications
  - History provides lessons for the future
- Computer Science at the crossroads from sequential to parallel computing
  - Salvation requires innovation in many fields, including computer architecture
- Thursday is "Intro to Simics" section with Scott
- Read Chapter 1, then Appendix B for next time!

Acknowledgements

- These slides contain material developed and copyright by:
  - Arvind (MIT)
  - Krste Asanovic (MIT/UCB)
  - Joel Emer (Intel/MIT)
  - James Hoe (CMU)
  - John Kubiatowicz (UCB)
  - David Patterson (UCB)

- MIT material derived from course 6.823
- UCB material derived from course CS252