

# **ECE 252 / CPS 220**

## **Advanced Computer Architecture I**

### **Lecture 2**

### **History, Instruction Set Architectures**

Benjamin Lee  
Electrical and Computer Engineering  
Duke University

[www.duke.edu/~bcl15](http://www.duke.edu/~bcl15)  
[www.duke.edu/~bcl15/class/class\\_ece252fall11.html](http://www.duke.edu/~bcl15/class/class_ece252fall11.html)



# ECE 252 Administria

## 8 September – Homework #1 Due

Assignment on web page. Teams of 2-3.

Submit hard copy in class. Email code to TA's

## 13 September – Class Discussion

Roughly one reading per class. Do not wait until the day before!

1. Hill et al. "Classic machines: Technology, implementation, and economics"
2. Moore. "Cramming more components onto integrated circuits"
3. Radin. "The 801 minicomputer"
4. Patterson et al. "The case for the reduced instruction set computer"
5. Colwell et al. "Instruction sets and beyond: Computers, complexity, controversy"



# A Bit of History

## Historical Narrative

- Helps understand why ideas arose
- Helps illustrate the design process

## Technology Trends

- Future technologies may be as constrained as older ones

## Learning from History

- 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

Charles Babbage (1791-1871)

- Lucasian Professor of Mathematics
- Cambridge University, 1827-1839

## Contributions

- Difference Engine 1823
- Analytic Engine 1833



## Approach

- mathematical tables (astronomy), nautical tables (navy)
- any continuous function can be approximated by polynomial
- mechanical gears and simple calculators



# Difference Engine

## Weierstrass

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

Example: All you need is an adder

$$f(n) = n^2 + n + 41$$

$$d1(n) = f(n) - f(n-1) = 2n$$

$$d2(n) = d1(n) - d1(n-1) = 2$$

$$f(n) = f(n-1) + d1(n) = f(n-1) + (d1(n-1)+2)$$

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



# Harvard Mark I

## History

- Howard Aiken – Professor of Physics, Harvard University
- Built in 1944 in IBM Endicott Laboratories

## Technology

- Mechanical with some electro-magnetically controlled relays, gears
- 750,000 components weigh 5 tons
- Synchronizing clock beats 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



# ENIAC

## History

- Electronic Numerical Integrator and Computer (ENIAC), 1943-45
- Eckert and Mauchly, University of Pennsylvania

## Technology

- First electronic, operational, general-purpose analytic calculator
- 30 tons, 72 square meters, 200KW

## Performance

- Reads 120 cards per minute
- 200 microsecond add, 6 millisecond division
- 1000x faster than Mark I
- Not very reliable!

## Application

- World War II ballistic calculations



# EDVAC

## History

- Eckert, Mauchly, von Neumann, et al., 1944
- ENIAC instructions execute independently of calculated results
- Human intervention required to change instruction flow
- EDVAC addresses this problem

## Technology

- First stored program computer
- Program manipulated as data

## Instruction Sequencing

- Manual control: calculators
- External automatic control: Harvard Mark I (paper tape)
- Internal automatic control: EDVAC (read-write memory)
- Stored program computer: same storage for data and instructions





# Technology and Reliability

## Early Technology

- ENIAC: 18,000 vacuum tubes, 20 10-digit numbers
- EDVAC: 4,000 vacuum tubes, 2,000 word storage

## BINAC

- Two processors that checked each other for reliability
- Did not work because processors never agreed

## Reliability Challenges

- Vacuum tubes, Williams tubes
- Charge storage on tube surface

## Technology Solution

- Magnetic-core memory, 1954
- J. Forrester, MIT
- MIT Whirlwind, MTBF 20min, considered reliable



# Computing in the 1950s

Hardware was expensive

## Storage and Memory Challenges

- Stores were small (1000 words), no resident system software
- Memory access time was 10-50x slower than processor cycle
- Instruction execution time dominated by memory access time

## Computation Challenges

- Emphasize design of complex control circuits to execute an instruction
- Neglect time required to decode instruction
- Programmer's view of machine inseparable from HW implementation



# Early Instruction Sets

## Single Accumulator

- Carry-over from calculators, typically less than 2-dozen instructions
- Single operand (AC)

LOAD	x	$AC \leftarrow M[x]$
STORE	x	$M[x] \leftarrow (AC)$
ADD	x	$AC \leftarrow (AC) + M[x]$
SUB	x	
SHIFT LEFT		$AC \leftarrow 2 \times (AC)$
SHIFT RIGHT		
JUMP	x	$PC \leftarrow x$
JGE	x	if $(AC) \geq 0$ then $PC \leftarrow x$
LOAD ADR	x	$AC \leftarrow \text{Extract address field } (M[x])$
STORE ADR	x	



# Single Accumulator Machine

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

LOOP	LOAD	N	# AC $\leftarrow$ M[N]
	JGE	DONE	# if(AC>0), PC $\leftarrow$ DONE
	ADD	ONE	# AC $\leftarrow$ AC + 1
	STORE	N	# M[N] $\leftarrow$ AC
F1	LOAD	A	# AC $\leftarrow$ M[A]
F2	ADD	B	# AC $\leftarrow$ (AC) + M[B]
F3	STORE	C	# M[C] $\leftarrow$ (AC)
	JUMP	LOOP	
DONE	HLT		

How to modify the addresses A, B and C ?



# Self-Modifying Code

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

LOOP	LOAD	N	# AC $\leftarrow$ M[N]
	JGE	DONE	# if (AC $\geq$ 0), PC $\leftarrow$ DONE
	ADD	ONE	# AC $\leftarrow$ AC + M[ONE]
	STORE	N	# M[N] $\leftarrow$ AC
F1	LOAD	A	# AC $\leftarrow$ M[A]
F2	ADD	B	# AC $\leftarrow$ AC + M[B]
F3	STORE	C	# M[C] $\leftarrow$ (AC)
	LOAD ADR	F1	# AC $\leftarrow$ address field (M[F1])
	ADD	ONE	# AC $\leftarrow$ AC + M[ONE]
	STORE ADR	F1	# changes address of A
	LOAD ADR	F2	
	ADD	ONE	
	STORE ADR	F2	# changes address of B
	LOAD ADR	F3	
	ADD	ONE	
	STORE ADR	F3	# changes address of C
	JUMP	LOOP	
DONE	HLT		

Each iteration requires:

	total	book-keeping
Inst fetch	17	14
Stores	5	4



# Index Registers

Specialized registers to simplify address calculations

- T. Kilburn, Manchester University, 1950s
- Instead of single AC register, use AC and IX registers

## Modify Existing Instructions

- Load x, IX       $AC \leftarrow M[x + (IX)]$
- Add x, IX       $AC \leftarrow (AC) + M[x + (IX)]$

## Add New Instructions

- Jzi x, IX      if  $(IX)=0$ , then  $PC \leftarrow x$ , else  $(IX) \leftarrow (IX)+1$
- Loadi x, IX       $IX \leftarrow M[x]$  (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	# load n into IX
LOOP	JZi	DONE, IX	# if(IX=0), DONE
	LOAD	LASTA, IX	# $AC \leftarrow M[LASTA + (IX)]$
	ADD	LASTB, IX	# note: LASTA is address
	STORE	LASTC, IX	# of last element in A
	JUMP	LOOP	
DONE	HALT		

- Longer instructions (1-2 bits), index registers with ALU circuitry
- Does not require self-modifying code, modify IX instead
- Improved program efficiency (operations per iteration)

	total	book-keeping
Inst fetch	5	2
Stores	1	0



# Operations on Index Registers

Increment index register by k

$AC \leftarrow (IX)$  *new instruction*

$AC \leftarrow (AC) + k$

$IX \leftarrow (AC)$  *new instruction*

also the AC must be saved and restored

Instead, manipulate index register directly

INCi k, IX

$IX \leftarrow (IX) + k$

STOREi x, IX

$M[x] \leftarrow (IX)$  (extended to fit a word)

IX begins to resemble AC

- Several index registers, accumulators
- Motivates general-purpose registers (e.g., MIPS ISA R0-R31)





# Evolution of Addressing Modes

## 1. Single accumulator, absolute address

Load x                       $AC \leftarrow M[x]$

## 2. Single accumulator, index registers

Load x, IX                 $AC \leftarrow M[x + (IX)]$

## 3. Single accumulator, indirection

Load (x)                  $AC \leftarrow M[M[x]]$

## 4. Multiple accumulators, index registers, indirection

Load Ri, IX, (x)         $R_i \leftarrow M[M[x] + (IX)]$

## 5. Indirect through registers

Load Ri, (Rj)             $R_i \leftarrow M[M[(R_j)]]$

## 6. The Works

Load Ri, Rj, (Rk)         $R_j = \text{index}, R_k = \text{base address}$   
 $R_i \leftarrow M[R_j + (R_k)]$



# Instruction Formats

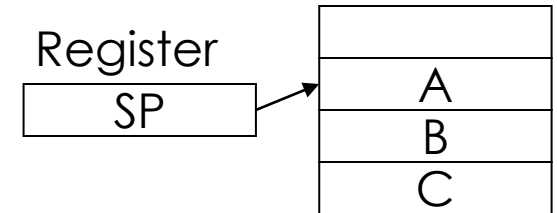
## Zero-address Formats

- Operands on a stack

add      $M[sp] \leftarrow M[sp] + M[sp-1]$

load     $M[sp] \leftarrow M[M[sp]]$

- Stack can be registers or memory
- Top of stack usually cached in registers



## One-address Formats

- Accumulator is always other implicit operand



# Instruction Formats (cont.)

## Two-address Formats

- Destination is same as one of the operand sources

$$R_i \leftarrow (R_i) + (R_j) \quad \# \text{ (Reg x Reg) to Reg}$$

$$R_i \leftarrow (R_i) + M[x] \quad \# \text{ (Reg x Mem) to Reg}$$

- x can be specified directly or via register
- x address calculation could include indexing, indirection, etc.

## Three-address Formats

- One destination and up to two operand sources

$$R_i \leftarrow (R_j) + (R_k) \quad \# \text{ (Reg x Reg) to Reg}$$

$$R_i \leftarrow (R_j) + M[x] \quad \# \text{ (Reg x Reg) to Reg}$$



# Data Formats

## Data Sizes

- Bytes, Half-words, words, double words

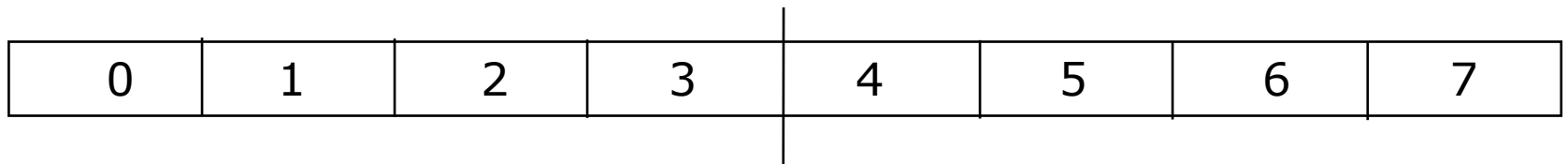
## Byte Addressing

- Location of most-, least- significant bits

LSB			MSB	Big Endian
MSB			LSB	Little Endian

## Word Alignment

- Suppose memory is organized into 32-bit words
- Word aligned addresses begin only at 0, 4, 8, ...





# Software Developments

## Numerical Libraries (up to 1955)

- floating-point operations
- transcendental functions
- matrix multiplication, equation solvers, etc.

## High-level Languages(1955-1960)

- Fortran, 1956
- assemblers, loaders, linkers, compilers

## Operating Systems (1955-1960)

- accounting programs to track usage and charges



# Compatibility

Early 1960s, IBM had 4 incompatible computers

- IBM 701, 650, 702, 1401
- Different instruction set architecture
- Different I/O system, secondary storage (magnetic taps, drums, disks)
- Different assemblers, compilers, libraries
- Different markets (e.g., business, scientific, real-time)

Motivated IBM 360



# IBM 360: Design Principles

Amdahl, Blaauw and Brooks, "Architecture of the IBM System/360" 1964

1. Support growth and successor machines
2. Connect I/O devices with general methods
3. Emphasize total performance
  - Evaluate answers per month rather than bits per microsecond
  - Emphasize programmability
4. Eliminate manual intervention
  - Machine must be capable of supervising itself
5. Reduce down time
  - Build hardware fault checking and fault location support
6. Facilitate assembly
  - Redundant I/O devices, memories for fault tolerance
7. Support flexibility
  - Some problems required floating-point words > 36bits



# IBM 360: General Purpose Registers

## Processor State

- 16 general-purpose, 32-bit registers
- may be used as index and base register
- register 0 has special properties
- 4 floating-point, 64-bit registers
- a program status word (PSW) with program counter (PC), condition codes, control flags

## Data Formats

- 8-bit bytes: the IBM 360 is why bytes are 8-bits long today!
- 16-bit half-words
- 32-bit words
- 64-bit double-words





# IBM 360: Initial Implementation

	<b>Model 30</b>	<b>Model 70</b>
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	1 microsecond read	Conventional circuits

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

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

- With minor modifications, ISA still survives today



# IBM z990: Forty Years Later

IBM Journal R&D, 48(3/4), May/July 2004.

64-bit virtual addressing

original S/360 was 24-bit, and S/370 was 31-bit extension

Dual core design

Dual-issue in-order superscalar

0-stage CISC pipeline

Out-of-order memory accesses

Redundant datapaths

- every instruction performed in two parallel datapaths and results compared

256KB L1 I-cache, 256KB L1 D-cache on-chip

32MB shared L2 unified cache, off-chip

512-entry L1 TLB + 4K-entry L2 TLB

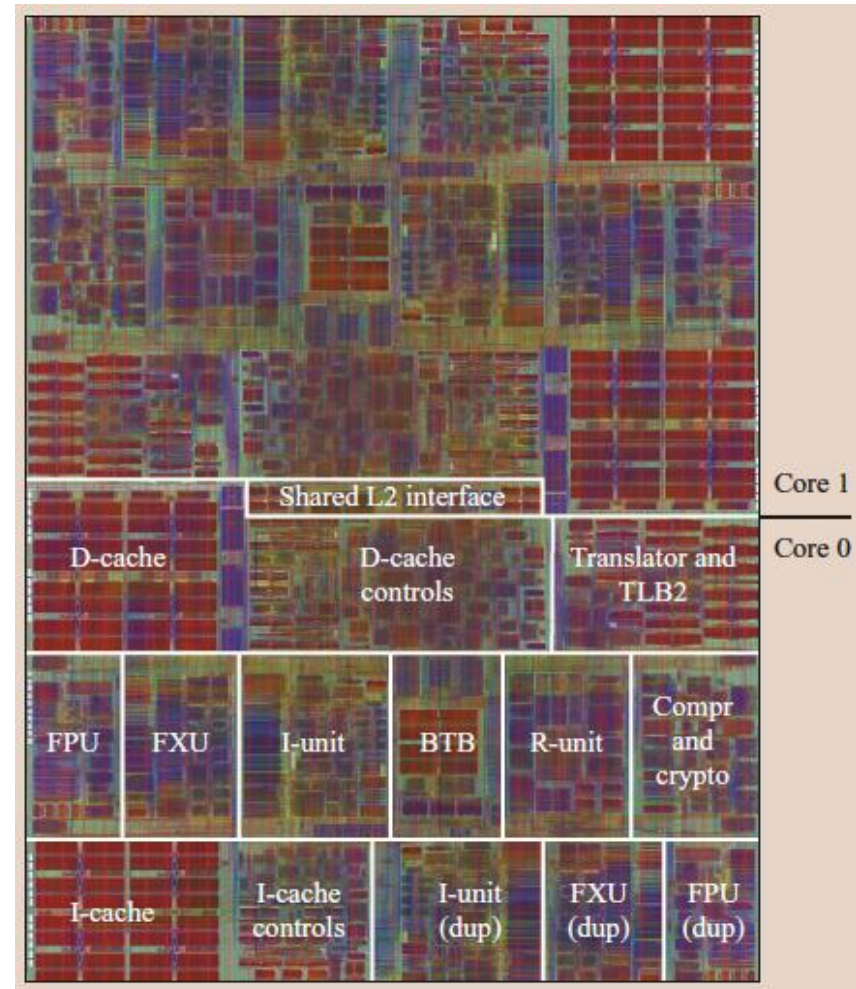
very large TLB, to support multiple virtual machines

8K-entry Branch Target Buffer

Very large buffer to support commercial workloads

Up to 64 processors (48 visible) in one machine

1.2 GHz in IBM 130nm SOI CMOS, 55W for cores





# 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)
- Alvin Lebeck (Duke)
- David Patterson (UCB)
- Daniel Sorin (Duke)