ECE 552 / CPS 550
Advanced Computer Architecture I

Lecture 2
History, Instruction Set Architectures

Benjamin Lee
Electrical and Computer Engineering
Duke University

www.duke.edu/~bcl15
www.duke.edu/~bcl15/class/class_ece252fall12.html

ECE 552 Administrivia

11 September – Homework #1 Due

Assignment on web page. Teams of 2-3.

Submit soft copies to Sakai.

Use Piazza for questions.

11 September – Class Discussion

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

Karl 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

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

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

$d2(n) = d1(n) - d1(n-1) = 2 \Rightarrow d1(n) = d1(n-1) + d2(n)$

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 ← M[x]
STORE	x	M[x] ← (AC)
ADD	x	AC ← (AC) + M[x]
SUB	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	

Single Accumulator Machine

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

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

Notice M[N] is a counter, not an index.
How to modify the addresses A, B and C ?

Self-Modifying Code

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

```
LOOP    LOAD      N       # AC ← M[N]
        JGE       DONE    # if (AC >= 0), PC ← DONE
        ADD       ONE     # AC ← AC + M[ONE]
        STORE     N       # M[N] ← AC
F1      LOAD      A       # AC ← M[A]
F2      ADD       B       # AC ← AC + M[B]
F3      STORE     C       # M[C] ← (AC)
        LOAD ADR  F1      # AC ← address field (M[F1])
        ADD       ONE     # AC ← 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 ← M[x + (IX)]
- Add x, IX AC ← (AC) + M[x + (IX)]

Add New Instructions

- 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          # load n into IX
LOOP    JZi     DONE, IX        # if(IX=0), DONE
        LOAD    LASTA, IX       # AC← 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

Option 1: Increment index register by k

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

Also, the AC must be saved and restored

Option 2: Manipulate index register directly

INCi k, IX IX ← (IX) + k
STOREi x, IX M[x] ← (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 ← M[x]

2. Single accumulator, index registers

   Load x, IX AC ← M[x + (IX)]

3. Single accumulator, indirection

   Load (x) AC ← M[M[x]]

4. Multiple accumulators, index registers, indirection

   Load Ri, IX, (x) Ri ← M[M[x] + (IX)]

5. Indirection through registers

   Load Ri, (Rj) Ri ← M[M[(Rj)]]

6. The Works

   Load Ri, Rj, (Rk) Rj = index, Rk = base address

   Ri ← M[Rj + (Rk)]

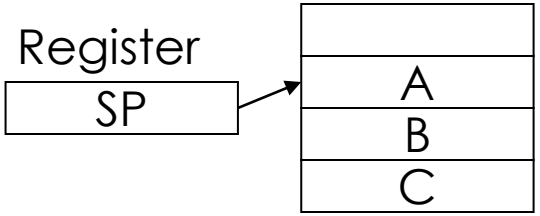
Instruction Formats

Zero-address Formats

- Instructions have zero operands
- Operands on a stack

  add M[sp] ← M[sp] + M[sp-1]

  load M[sp] ← M[M[sp]]
- Stack can be registers or memory
- Top of stack usually cached in registers



One-address Formats

- Instructions have one operand
- Accumulator is always other implicit operand

Instruction Formats (cont.)

Two-address Formats

- Destination is same as one of the operand sources

Ri ← (Ri) + (Rj)	# (Reg x Reg) to Reg
Ri ← (Ri) + M[x]	# (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

Ri ← (Rj) + (Rk)	# (Reg x Reg) to Reg
Ri ← (Rj) + M[x]	# (Reg x Reg) to Reg

Data Formats

Data Sizes

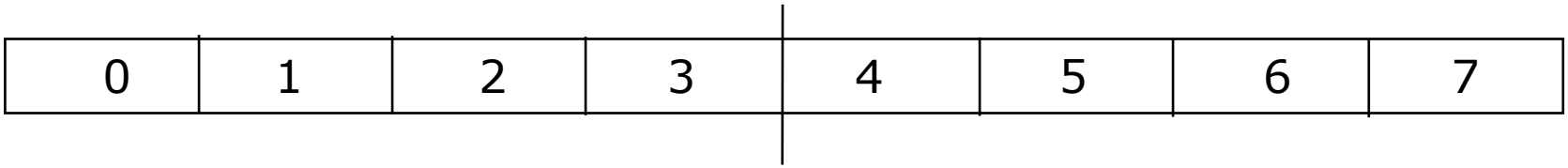
- Bytes, Half-words, words, double words

Byte Addressing

- Location of most-, least- significant bits

Word Alignment

- Suppose memory is organized into 32-bit words (e.g., 4 bytes).
- Word aligned addresses begin only at 0, 4, 8, … bytes



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)

The need for compatibility 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

	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	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 z11: 47 Years Later

IBM, HotChips 2010

5.2 GHz in IBM 45nm CMOS technology
1.4 billion transistors in 512 sq-mm

64-bit virtual addressing
- original IBM360 was 24-bit

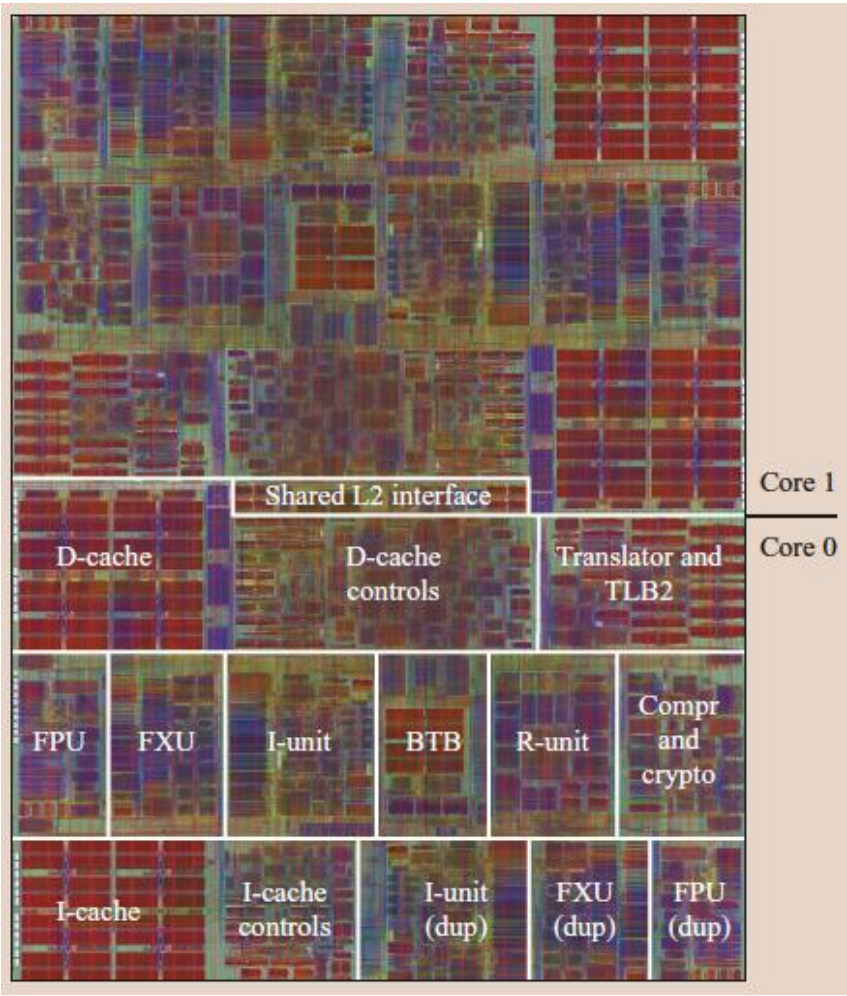
Quad-core design

Out-of-order, 3-way superscalar pipeline

Redundant datapaths
- every instruction performed in 2 parallel datapaths and results compared

L1 i-cache (64KB); L1 d-cache (128KB) d-cache
L2 cache (1.5MB), private, per-core
L3 cache (24MB), eDRAM

Scales to 96 cores in one machine



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)