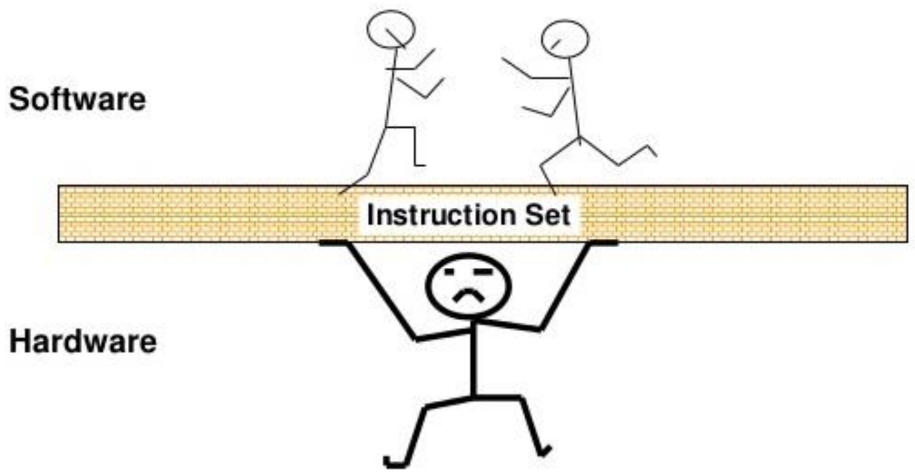
زبان و ساختار کامپیوتر

فصل سوم

معماری مجموعه دستورالعمل‌ها

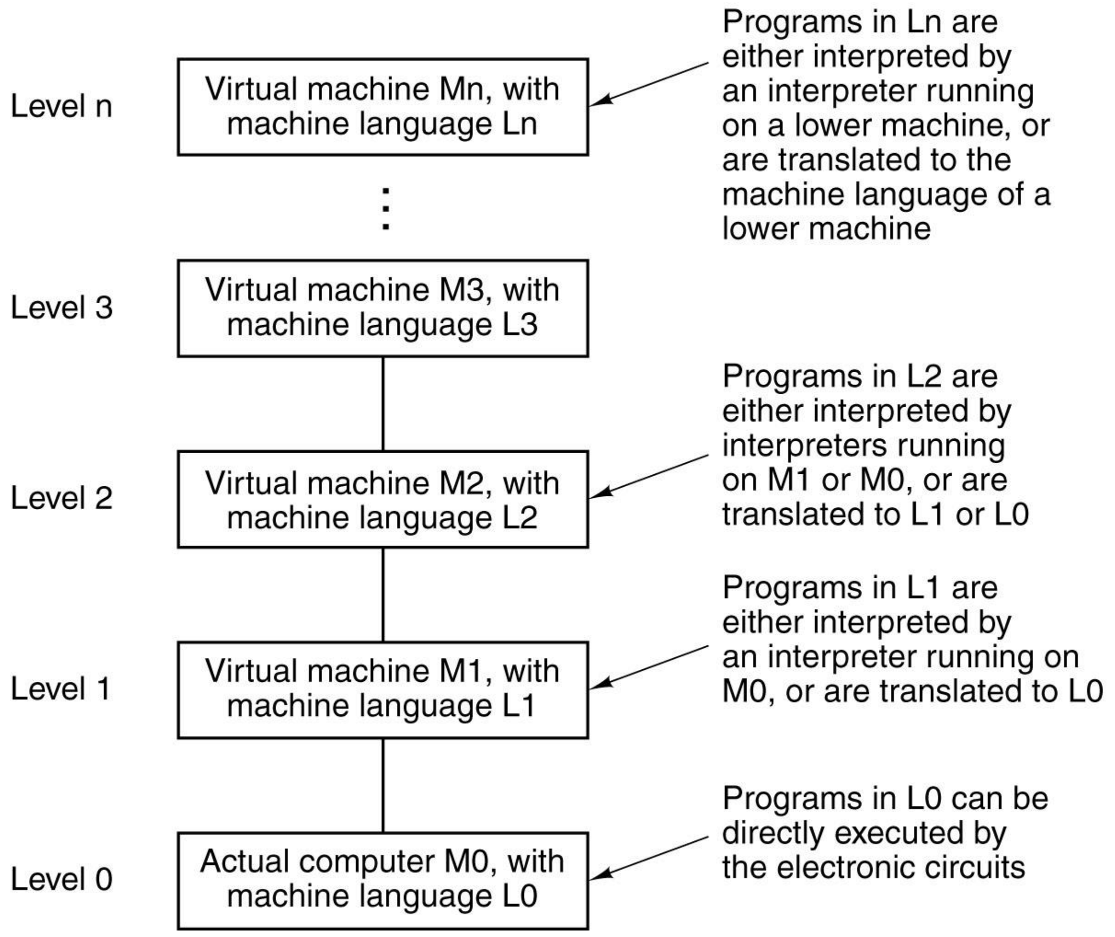


Copyright Notice

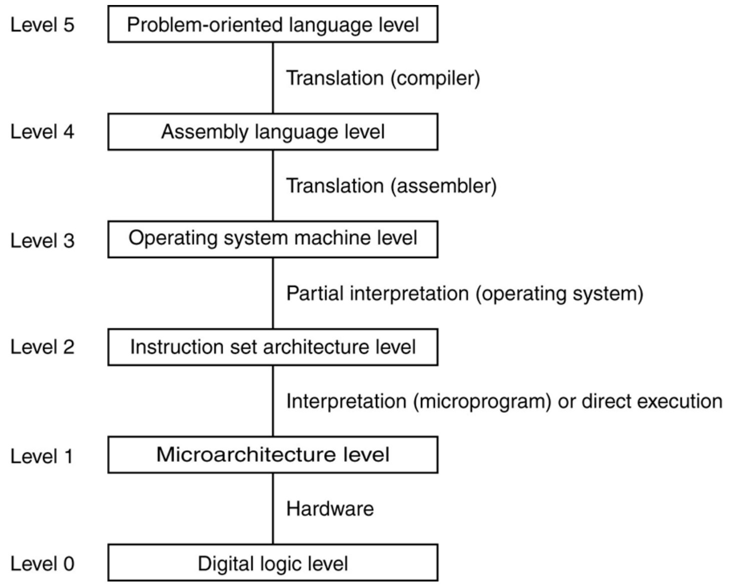
Parts (text & figures) of this lecture are adopted from:

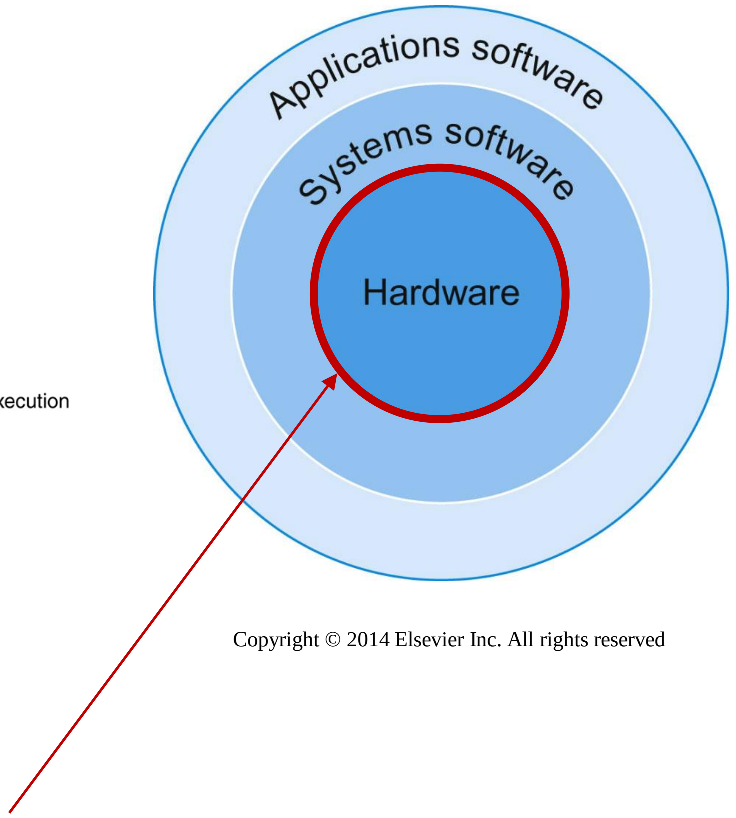
- D. Patterson & J. Hennessey, "Computer Organization & Design, The Hardware/Software Interface", 5th Ed., MK publishing, 2014
- A. Tanenbaum, "Structured Computer Organization", 5th Ed., Pearson, 2006
- "Computer System Architecture", M. Morris Mano, Pearson, 1999

A Multilevel Machine


Level n
Virtual machine Mn, with machine language Ln
Programs in Ln are either interpreted by an interpreter running on a lower machine, or are translated to the machine language of a lower machine
Level 3
Virtual machine M3, with machine language L3
Level 2
Virtual machine M2, with machine language L2
Programs in L2 are either interpreted by interpreters running on M1 or M0, or are translated to L1 or L0
Level 1
Virtual machine M1, with machine language L1
Programs in L1 are either interpreted by an interpreter running on M0, or are translated to L0
Level 0
Actual computer M0, with machine language L0
Programs in L0 can be directly executed by the electronic circuits

Hierarchical Levels (Reminder)


Level 5
Problem-oriented language level
Translation (compiler)
Level 4
Assembly language level
Translation (assembler)
Level 3
Operating system machine level
Partial interpretation (operating system)
Level 2
Instruction set architecture level
Interpretation (microprogram) or direct execution
Level 1
Microarchitecture level
Hardware
Level 0
Digital logic level



Tanenbaum, Structured Computer Organization, 5th Edition

Instruction Set Architecture (ISA)

Instruction Set Architecture (ISA)

- How the machine appears to a machine language programmer
- What a compiler outputs
  - ignoring operating-system calls & symbolic assembly language
- Specifies:
  - Memory Model
  - Registers
  - Available data types
  - Available instructions

ISA Exclusion

- Issues not part of ISA (not visible to the compiler):
  - it is pipelined or not
  - it is superscalar or not
  - cache memory is used or not
  - …

ISA Exclusion

- Issues not part of ISA (not visible to the compiler):
  - it is pipelined or not
  - it is superscalar or not
  - cache memory is used or not
  - …
- However some of these properties do affect performance & is better to be visible to the compiler writer!

ISA Aspects

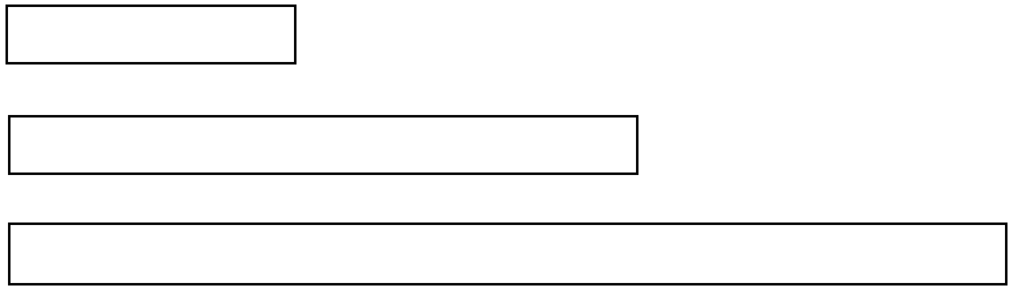
- How many instructions needed?
- What functionalities required?
  - Load/Store
  - Control (such as compare & jump)
  - Arithmetic & logical operations
- How are the instructions applied?

Key ISA Decisions

- Instruction length?
- How many registers?
- Where operands reside?
- Which instructions can access memory?
- Instruction format?
- Operand format?
  - How many? How big?

Instruction Length

Variable:



x86 – Instructions vary from 1 to 17 Bytes long

VAX – from 1 to 54 Bytes

Fixed:

MIPS, PowerPC:

all instruction are 4 Bytes long

Variable-length Instructions

- Require multi-step fetch and decode
- Allow for a more flexible and compact instruction set
- CISC processors like x86 & VAX
  - (Complex Instruction Set Computing)

Fixed-length Instructions

- Allow easy fetch and decode
- Simplify pipelining and parallelism
- RISC processors like MIPS & PowerPC
  - (Reduced Instruction Set Computing)

MIPS TECHNOLOGIES

How Many Registers?

- All computers have a set of registers (register file)
- What is a Register?
  - A storage element within a processor
  - Memory to hold values that will be used soon
  - Keeps temporary results
  - Access time much faster than memory
  - Reduces # of accesses to memory

Small # of Registers

- Fewer bits required to specify which one
- Less hardware
- Faster access
  - Shorter wires
  - Faster index decoding (fewer logic levels)
- Faster context switch
  - When all registers need saving

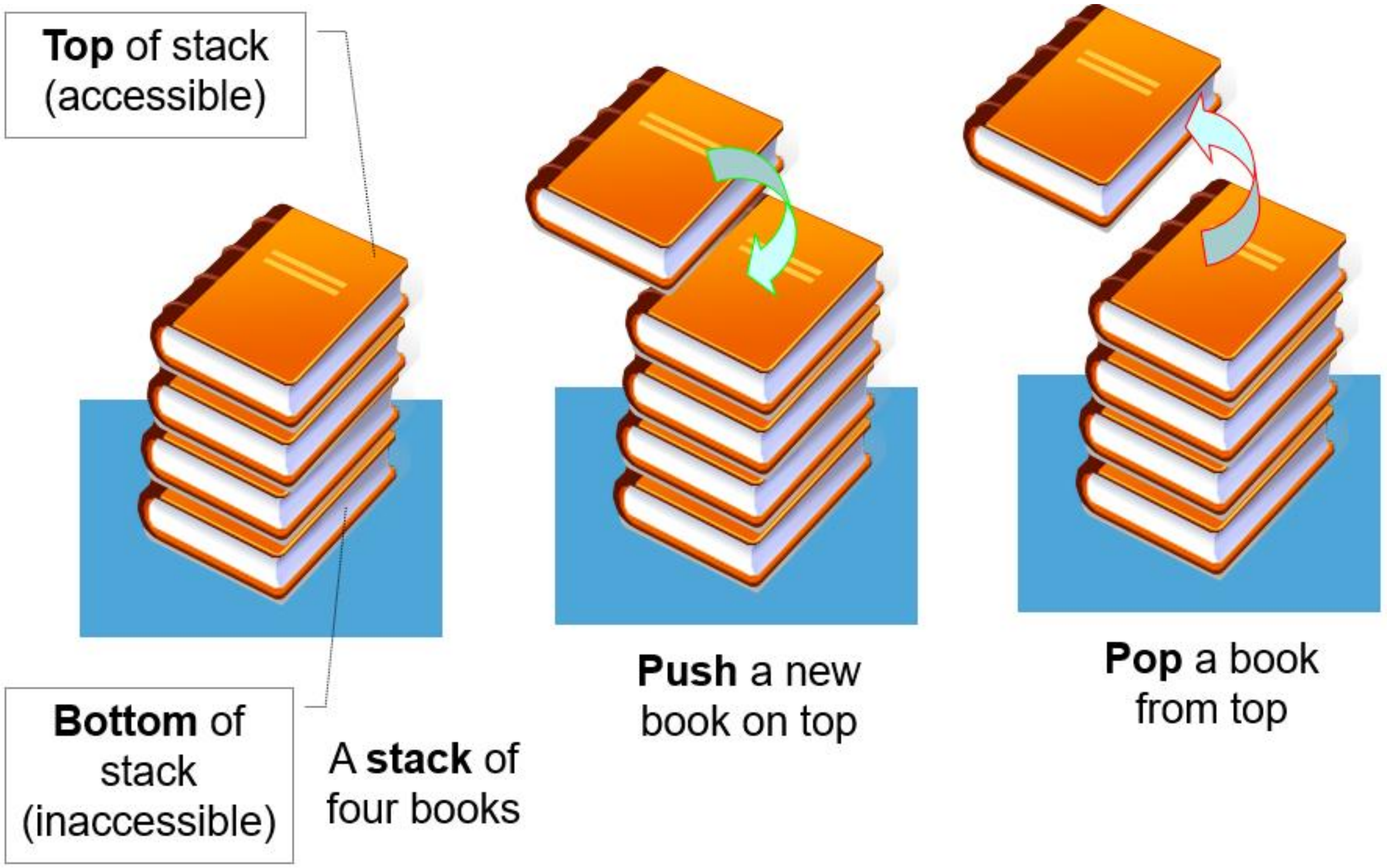
Larger # of Registers

- Fewer memory loads and stores
  - Less CPU/memory data transfers
- Easier to do several operations at once

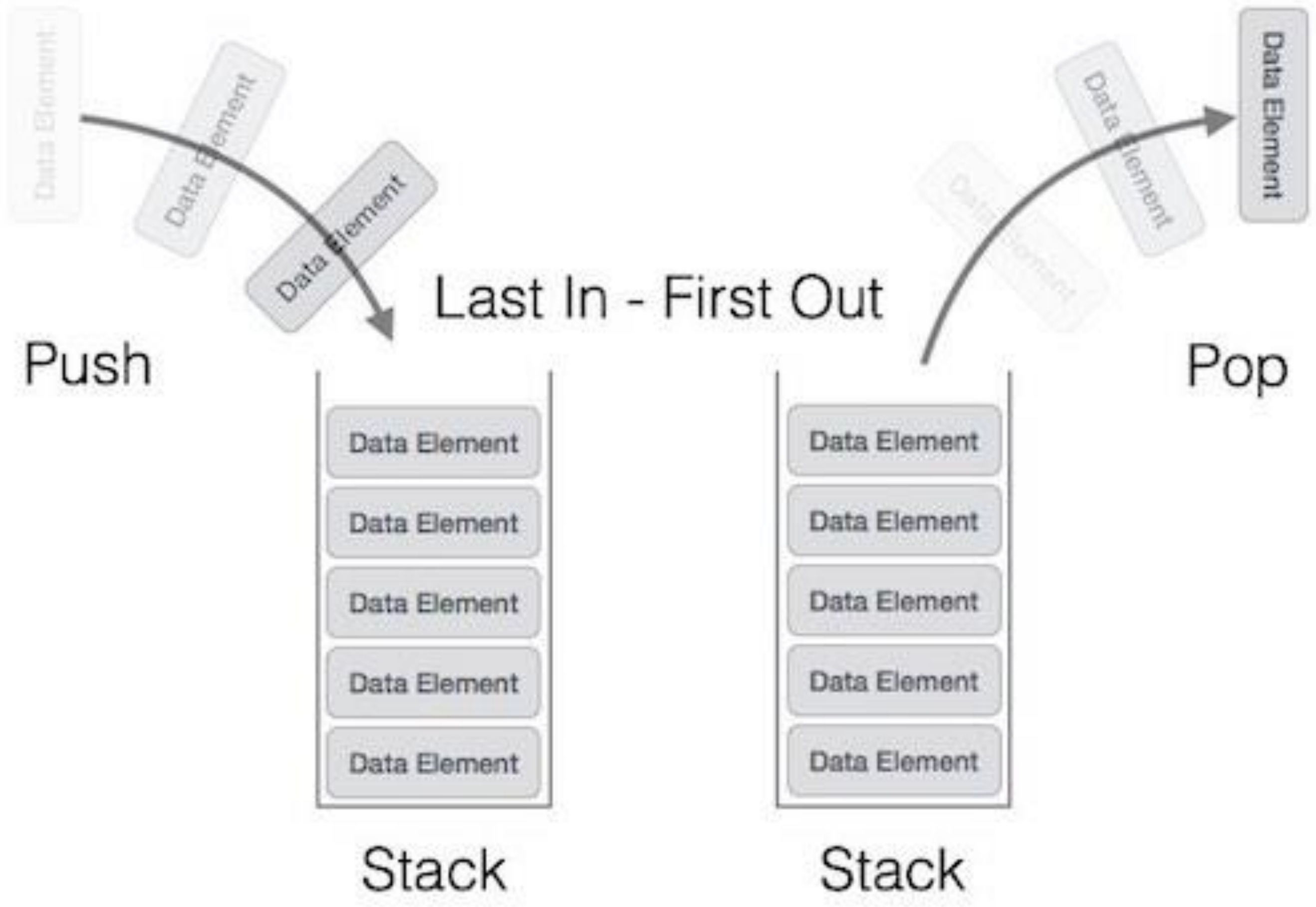
Where Operands Reside?

- Stack Machine
- Accumulator Machine
- Register-Memory Machine
- Register-Register Machine (Load-Store)

Stack Illustration


Top of stack (accessible)
Bottom of stack (inaccessible)
A **stack** of four books
Push a new book on top
Pop a book from top

Stack Representation



Stack Machine

- "Zero-operand" ISA
  - ALU operations (add, sub, ...) don't need any operands
- "Push"
  - Loads mem into 1st reg ("top of stack"),
- "Pop"
  - Does reverse
- "Add", "Sub", "Mul", and etc.
  - Combines contents of first two regs on top of stack

Example 1

Code sequence for **`C = A + B`**

Stack Accumulator Register-Memory Reg-Reg

?

Example 1-1

Code sequence for **`C = A + B`**

Stack	Accumulator	Register-Memory	Reg-Reg

```
Push A
Push B
Add
Pop  C
```

Accumulator Machine

- "1-operand" ISA
- Only 1 register called "accumulator"
- Stores intermediate arithmetic & logic results
- Instructions include:
  - "STORE" (Store AC)
  - "LOAD" (Load AC)
  - "ADD mem" (AC $\leftarrow$ AC + mem)

Example 1

Code sequence for `C = A + B`

Stack Accumulator Register-Memory Reg-Reg

?

Example 1-2

Code sequence for **`C = A + B`**

Stack	Accumulator	Register-Memory	Reg-Reg
`Push A`	`Load A`		
`Push B`	`Add B`		
`Add`	`Store C`		
`Pop C`			

Register-Memory Machine

- 2 or 3 Operands ISA
- A set of general purpose registers available
- Operands can be register or memory
- Arithmetic & logic instructions can use data in registers and/or memory
- Usually only one operand can be in memory

Example 1

Code sequence for `C = A + B`

Stack Accumulator Register-Memory Reg-Reg

?

Example 1-3

Code sequence for **`C = A + B`**

Stack	Accumulator	Register-Memory	Reg-Reg
`Push A`	`Load A`	`Mov R1, A`	
`Push B`	`Add B`	`Add R1, B`	
`Add`	`Store C`	`Mov C, R1`	
`Pop C`			

Register-Register Machine

- Also called Load-Store Machine
- 2 or 3 operands ISA
- A set of general purpose registers
- Arithmetic & logical instructions can only access registers
- Access to memory only with Load & Store

Example 1

Code sequence for **`C = A + B`**

Stack Accumulator Register-Memory Reg-Reg

?

Example 1-4

Code sequence for **C = A + B**

Stack	Accumulator	Register-Memory	Reg-Reg
`Push A`	`Load A`	`Mov R1, A`	`Load R1,A`
`Push B`	`Add B`	`Add R1, B`	`Load R2,B`
`Add`	`Store C`	`Mov C, R1`	`Add R3,R1,R2`
`Pop C`			`Store C,R3`

Example 2

Register-Memory

$X = (A + B) * (C + D)$

Reg-Reg

Stack

Accumulator

?

Example 2

X = (A + B) * (C + D)

Register-Memory

```
ADD   R1, A, B
ADD   R2, C, D
MUL   X, R1, R2
```

```
MOV   R1, A
ADD   R1, B
MOV   R2, C
ADD   R2, D
MUL   R1, R2
MOV   X, R1
```

Reg-Reg

```
LOAD    R1, A
LOAD    R2, B
LOAD    R3, C
LOAD    R4, D
ADD     R1, R1, R2
ADD     R3, R3, R4
MUL     R1, R1, R3
STORE   R1
```

Stack

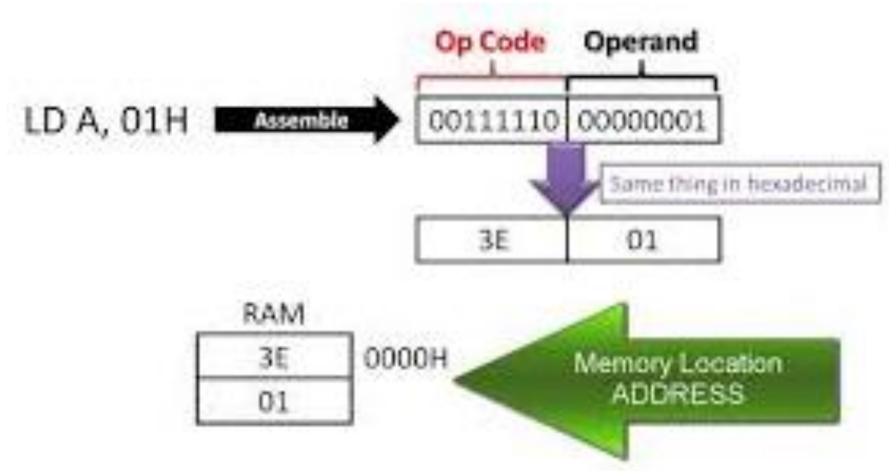
```
PUSH    A
PUSH    B
ADD
PUSH    C
PUSH    D
ADD
MUL
POP     X
```

Accumulator

```
LOAD     A
ADD      B
STORE    T
LOAD     C
ADD      D
MUL      T
STORE    X
```

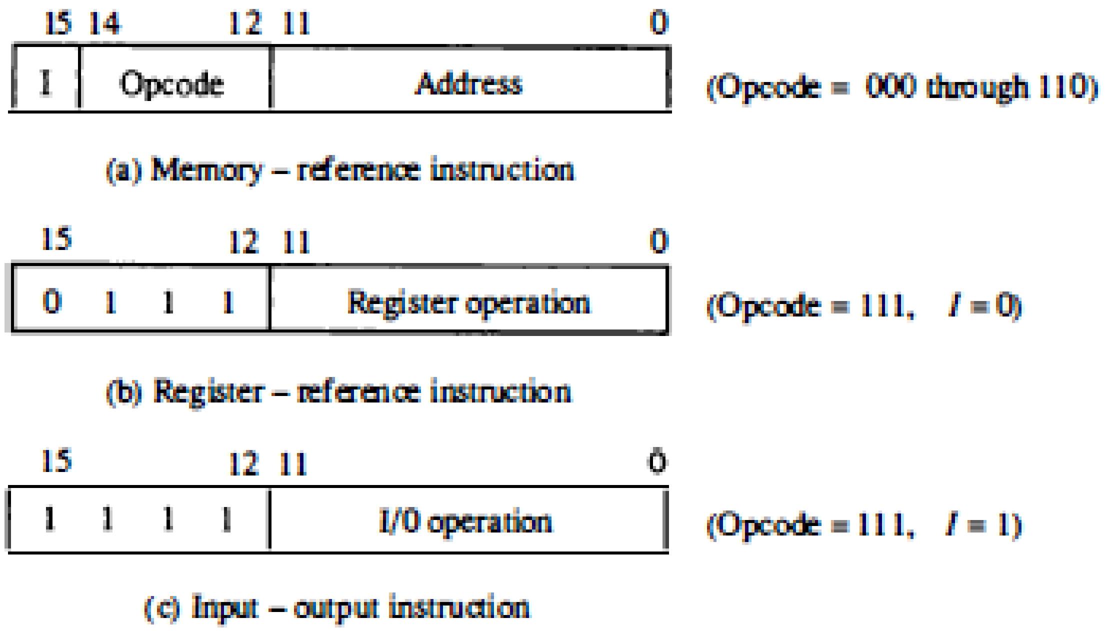
Instruction & Operand format

Which bits designate what?

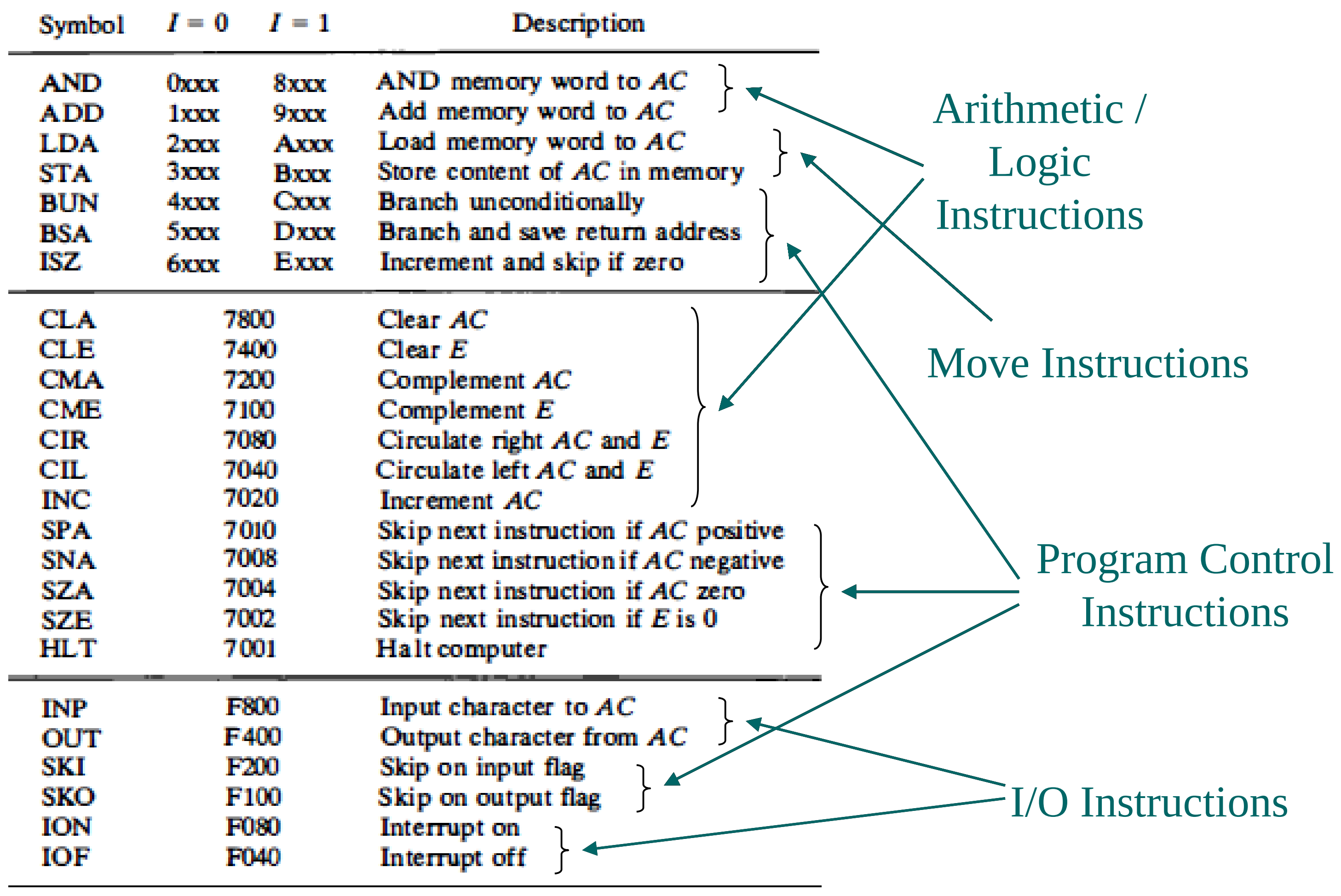


Can we have several formats in one machine?

Example: Mano's Basic Computer



Mano's Basic Computer (cont)



Symbol	I = 0	I = 1	Description
AND	0xxx	8xxx	AND memory word to *AC*
ADD	1xxx	9xxx	Add memory word to *AC*
LDA	2xxx	Axxx	Load memory word to *AC*
STA	3xxx	Bxxx	Store content of *AC* in memory
BUN	4xxx	Cxxx	Branch unconditionally
BSA	5xxx	Dxxx	Branch and save return address
ISZ	6xxx	Exxx	Increment and skip if zero
CLA	7800		Clear *AC*
CLE	7400		Clear *E*
CMA	7200		Complement *AC*
CME	7100		Complement *E*
CIR	7080		Circulate right *AC* and *E*
CIL	7040		Circulate left *AC* and *E*
INC	7020		Increment *AC*
SPA	7010		Skip next instruction if *AC* positive
SNA	7008		Skip next instruction if *AC* negative
SZA	7004		Skip next instruction if *AC* zero
SZE	7002		Skip next instruction if *E* is 0
HLT	7001		Halt computer
INP	F800		Input character to *AC*
OUT	F400		Output character from *AC*
SKI	F200		Skip on input flag
SKO	F100		Skip on output flag
ION	F080		Interrupt on
IOF	F040		Interrupt off

Complex Instruction Set Architecture

- A (usually) large no of instructions (100-250)
- Some instructions that perform specialized tasks & are used infrequently
- A large variety of addressing modes (5-20)
- Variable-length instruction formats
- Instructions that manipulate operands in memory

Reduced Instruction Set Architecture

- Relatively few instructions
- Relatively few addressing modes
- More general-purpose registers
- Memory access limited to load & store instructions
- All operations done within the registers of the CPU
- Fixed-length, easily decoded instruction format
- Single-cycle instruction execution (via pipelining)
- Hardwired rather than micro-programmed control

RISC vs. CISC

- Early Trend:
  - Adding more instructions to next generation CPUs to do more complicated operations
  - VAX machine had an instruction to multiply polynomials!
- CISC Philosophy
  - Limited main memory + immature compilers
  - More dense instructions, highly encoded, variable length instructions
  - Data loading as well as calculation

RISC vs. CISC (cont.)

- RISC Philosophy
  - Keep ISA small and simple
    - Makes it easier to build faster hardware
  - Let SW do complicated operations by composing simpler instructions
- Note on "Reduced" in RISC Phrase:
  - Amount of work any single instruction accomplishes reduced
    - Single ALU operation, single memory access, …

RISC vs. CISC (cont.)

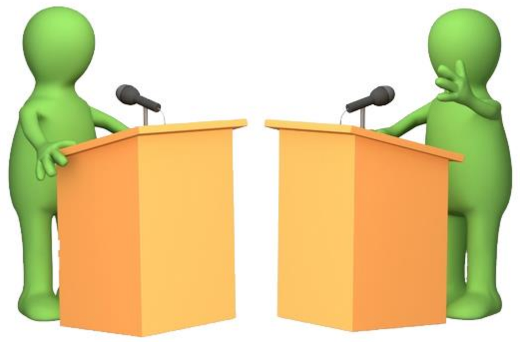
- CISC Problems
  - Performance tuning challenging
    - Complex/high-level instructions rarely used
  - Slower clock rates
  - Longer time-to-market
    - Due to prolonged design time
- CISC Features
  - Ease compiler implementation
    - HW supports all kind of addressing modes

RISC vs. CISC (cont.)

- RISC Features
  - Low complexity
    - Less error-prone HW implementation
  - Implementation advantages
    - Less transistors
    - Extra space: more registers, cache
  - Marketing
    - Reduced design time

RISC vs. CISC (cont.)

- Hybrid Solution
  - RISC core & CISC interface
  - Taking advantage of both architectures



RISC / CISC Debate

Modern (RISC) Design Principles

- Instructions should directly be executed by hw
  - no or very rare interpretation by microinstructions
- Maximize the rate at which instructions are issued
  - by means of instruction level parallelism
- Instructions should be easy to decode
  - regular, fixed length, small number of fields
- Only loads and stores should reference memory
- Plenty of registers

Outlines

- Instruction Set Architecture
- ISA Key Decisions
  - Instruction length?
  - How many registers?
  - Where operands reside?
  - Which instructions can access memory?
  - Instruction format?
  - Operand format? (How many? How big?)
- RISC vs. CISC
- Next Topic:

Addressing Modes