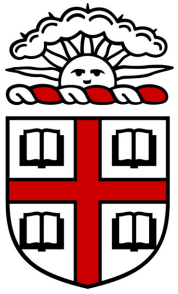
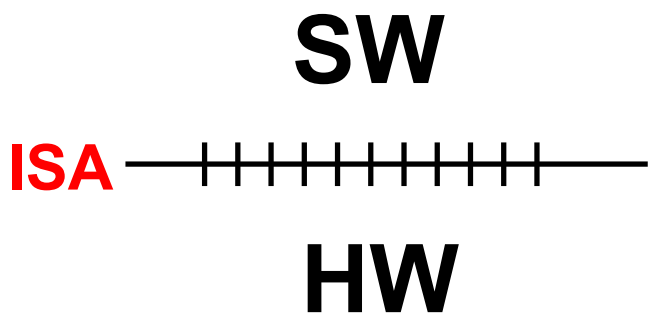
EN164: Design of Computing Systems
Lecture 08: Processor / ISA 1

Professor Sherief Reda
http://scale.engin.brown.edu
Electrical Sciences and Computer Engineering
School of Engineering
Brown University
Spring 2011



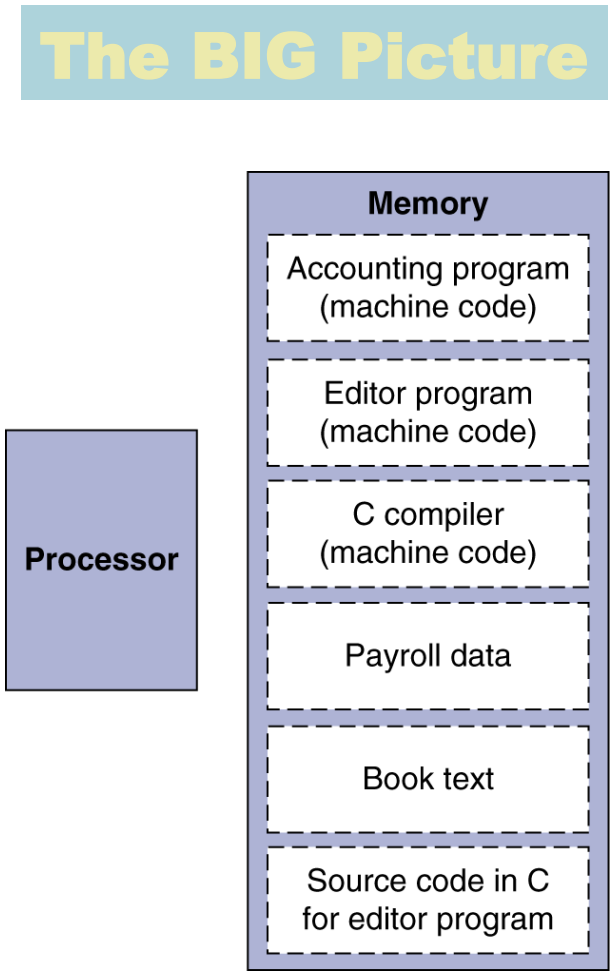
ISA is the HW/SW interface



ISA choice determines:

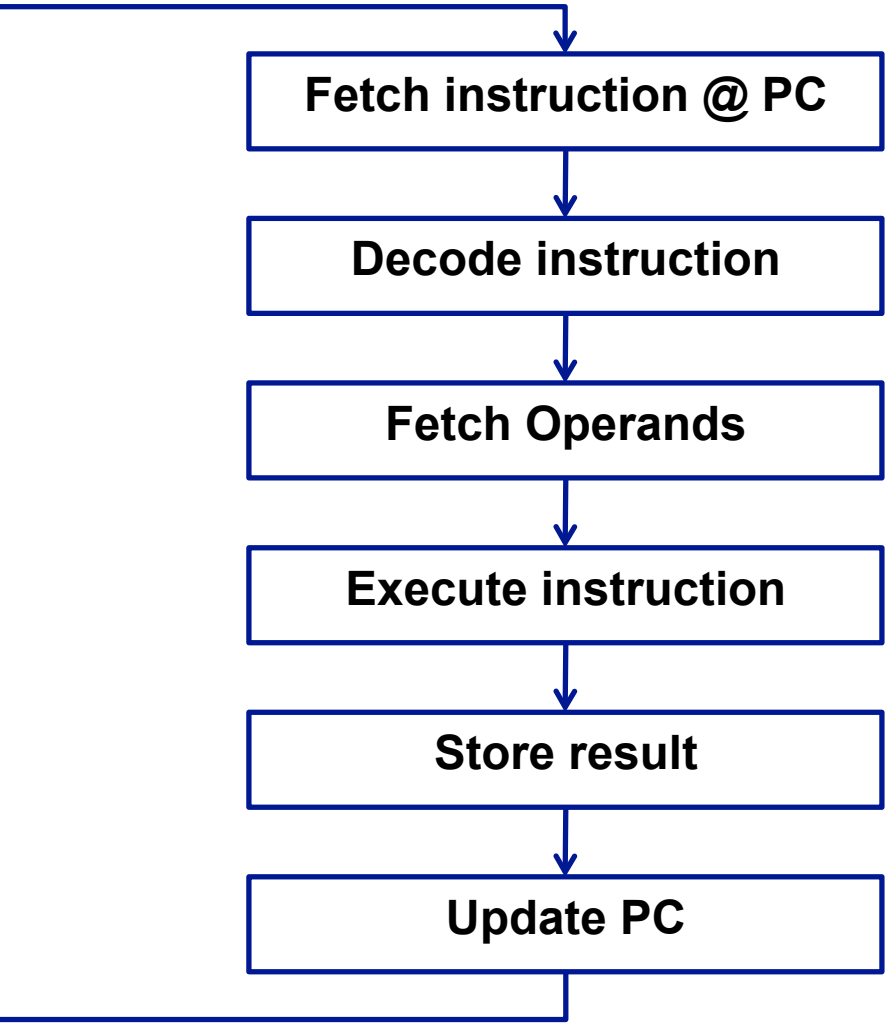
- program size (& memory size)
- complexity of hardware (CPI and f)
- execution time for different applications and domains
- power consumption
- die area (cost)

Stored program concept (von Neumann model)



- Instructions represented in binary, just like data
- Instructions and data stored in memory
- Programs can operate on programs
  - e.g., compilers, linkers, …
- Binary compatibility allows compiled programs to work on different computers
  - Standardized ISAs

Steps in execution of a program



- What is instruction format / size?
- how is it decoded?
- Where are the operands located? What are their sizes?
- What are supported operations?
- How to determine the successor instruction?

Example of an instruction

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

```
add $t0, $s1, $s2
```

assembly language

special	$s1	$s2	$t0	0	add
0	17	18	8	0	32
000000	10001	10010	01000	00000	100000

$00000010001100100100000000100000_2 = 02324020_{16}$

machine language

ISA design choices

- Number, size (fixed/variable) and format of instructions
- Operations supported (arithmetic, logical, string, floating point, jump, etc)
- Operands supported (bytes, words, signed, unsigned, floating, etc)
- Operand storage (accumulator, stack, registers, memory)
- Addressing modes

Typical operations

Data Movement	**Load (from memory)** **Store (to memory)** **memory-to-memory move** **register-to-register move** **input (from I/O device)** **output (to I/O device)** **push, pop (to/from stack)**
Arithmetic	**integer (binary + decimal) or FP** **Add, Subtract, Multiply, Divide**
Shift	**shift left/right, rotate left/right**
Logical	**not, and, or, set, clear**
Control (Jump/Branch)	**unconditional, conditional**
Subroutine Linkage	**call, return**
Interrupt	**trap, return**
Synchronization	**test & set (atomic r-m-w)**
String	**search, translate**

[slide from M. Martin]

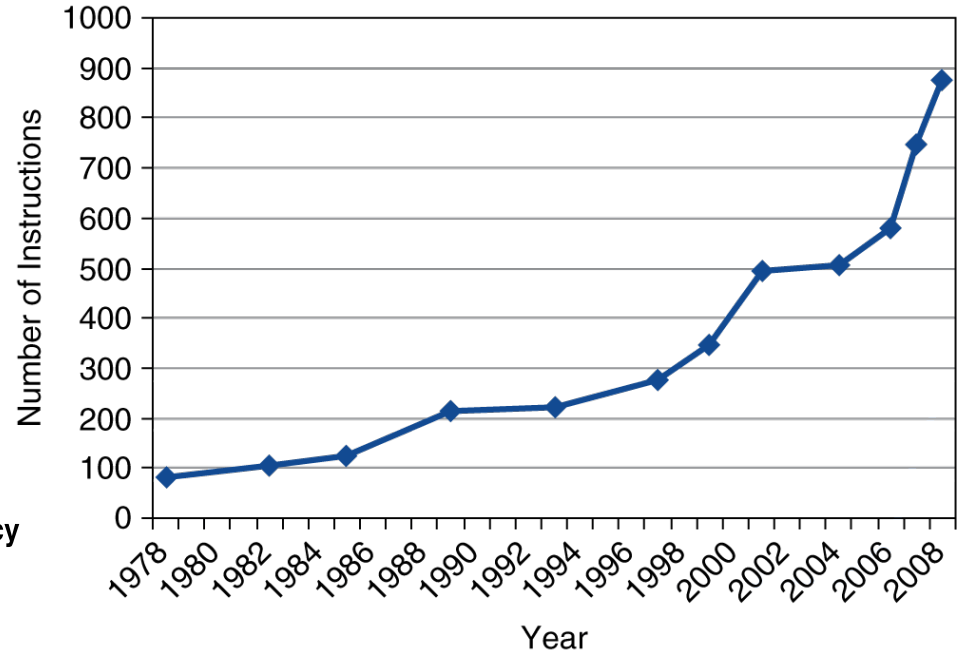
x86 ISA

- Backward compatibility ⇒ instruction set doesn't change
  - But they do accumulate more instructions

° Rank instruction Integer Average Percent total executed

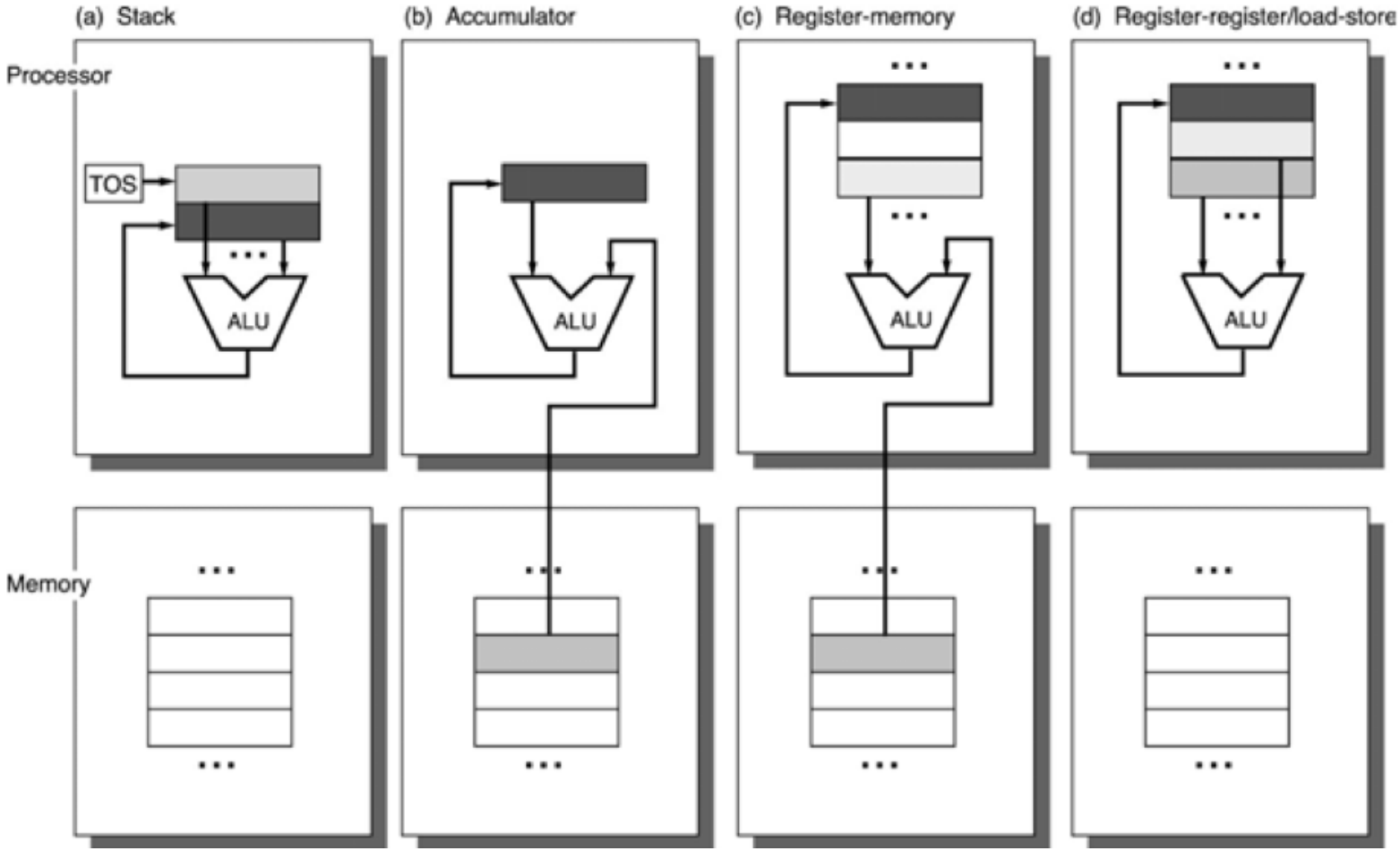
Rank	instruction	Integer Average Percent total executed
1	load	22%
2	conditional branch	20%
3	compare	16%
4	store	12%
5	add	8%
6	and	6%
7	sub	5%
8	move register-register	4%
9	call	1%
10	return	1%
	Total	96%

° Simple instructions dominate instruction frequency



x86 instruction set

Classification of ISAs



[Figure from D. Brooks -- Harvard]

These ISAs give different characteristics in terms of size of programs, number of instructions and CPI.

Examples of ISA

Instruction sequence for C = A + B for the four ISAs

Stack	Accumulator	Register (register-memory)	Register (load-store)
Push A Push B Add Pop C	Load A Add B Store C	Load R1, A Add R1, B Store C, R1	Load R1, A Load R2, B Add R3, R1, R2 Store C, R3

Some architectures (e.g. x86) support hybrid ISAs for different classes of instructions and/or for backward compatibility.

What makes a good ISA?

- Efficiency of hardware implementation
- Convenience of programming / compiling
- Matches target applications (or generality)
- Compatibility and portability

Four design principles for ISA

1. Simplicity favors regularity
2. Smaller is faster
3. Make the common case fast
4. Good design demands good compromises

*ISA design is an **art**!*

Popular ISAs

- x86 from Intel (laptops, servers)
- ARM (mobile devices)
- MIPS (embedded devices)
- Power and PowerPC from IBM (servers, old Macs)
- and many others still spoken and dead ISAs