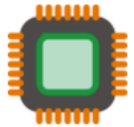


CASPER

CS305: Computer Architecture

ISA: What, Why, and How?

https://www.cse.iitb.ac.in/~biswa/courses/CS305/main.html

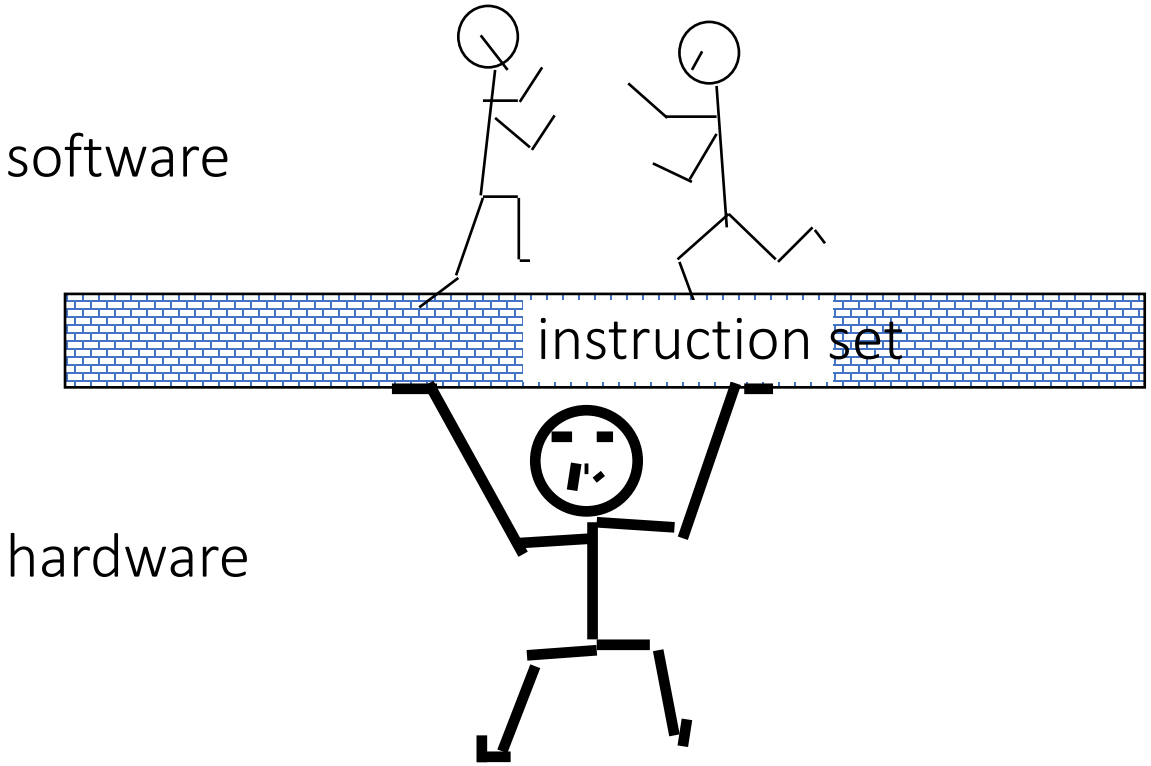
https://www.cse.iitb.ac.in/~biswa/

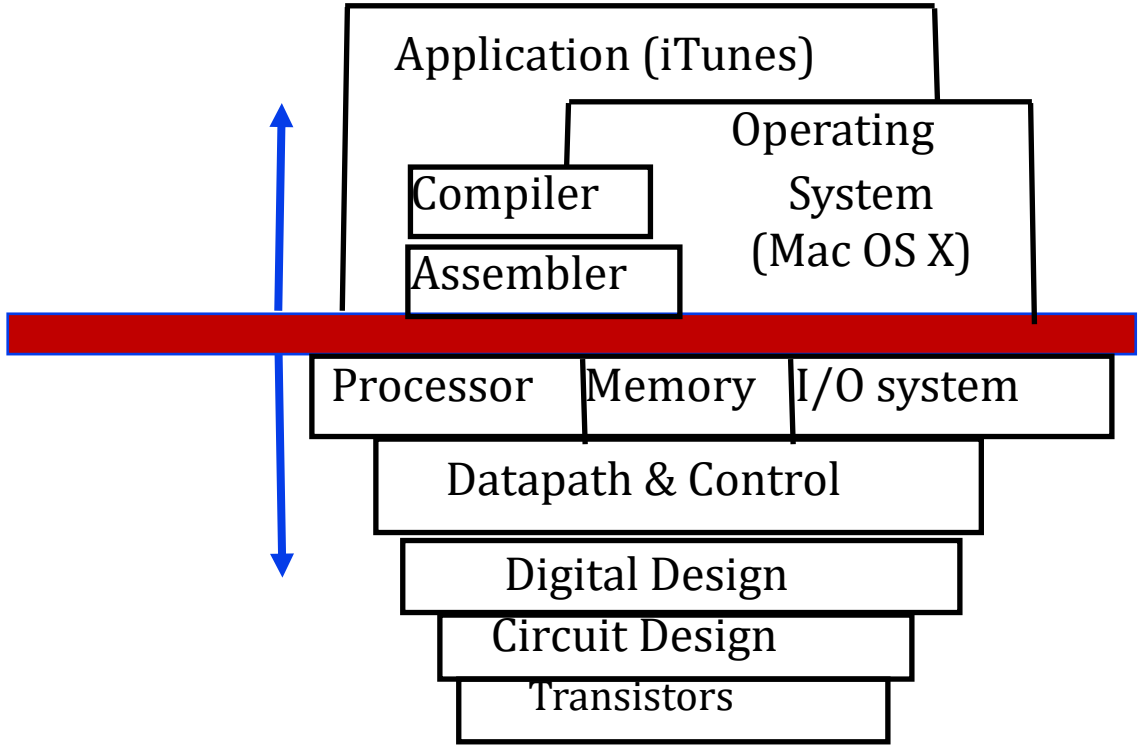
ISA

the attributes of a [computing] system as seen by the programmer, i.e. the conceptual structure and functional behavior, as distinct from the organization of the data flows and controls the logic design, and the physical implementation. –

Amdahl, Blaaw, and Brooks, 1964

ISA


software
instruction set
hardware


Application (iTunes)
Operating System (Mac OS X)
Compiler
Assembler
Processor
Memory
I/O system
Datapath & Control
Digital Design
Circuit Design
Transistors

Microarchitecture (Not exposed to us)

- Implementation of an ISA
- Programmer cannot see/access it

ISA:

add instruction

Microarch:

Implementation of an adder (ripple carry, lookahead …)

ISA: What does it provide?

Opcodes,

Addressing Modes,

Instruction Types and Formats,

Registers

Access control: user/OS

Address space,

Addressability,

Alignment

ISA: What does it provide?

Instructions:

Opcodes,

Addressing Modes,

Instruction Types and Formats,

Registers

Access control: user/OS

Memory: Address space,

Addressability,

Alignment

ISA must satisfy the needs of the software: - assembler, compiler, OS,

Microarchitecture

Rest of CS305 after ISA ☺

Caches
Memory Controllers
Branch Predictors, Prefetchers, …

Microarchitecture

Processor is in state S

Instruction

Processor moves to state SS

State

The information held in the processor at the end of an instruction to provide the processing context for the next instruction.

Computer Architecture

ISA + Microarchitecture

Based on lectures so far

#registers

#cycles to access a register

#Width of the register (32/64 bit)

#Instruction that uses register to access memory

#cycles to access memory

Based on lectures so far

#registers: ISA

#cycles to access a register: Microarch.

#Width of the register (32/64 bit) : ISA

#Instruction that uses register to access memory: ISA

#cycles to access memory: Microarch.

x86: It has 128/256-bit registers and one-bit too ☺

Where to place it?

- Closer to high-level language → Small semantic gap, complex instructions
  (CISC kinda? e.g. quicksort an instruction ☺)

- Closer to hardware? → Large semantic gap, simple instructions (RISC kinda?)
- Remember: Compiler+ISA defines app's instruction count

And then the Debate of RISC vs CISC

RISC: Reduced Instruction Set of Computers

Very few simple instructions

Example: MIPS

CISC: Complex Instruction Set of Computers

Lots of complicated instructions

Example: x86 kind of ☺ [x86 is CISC with RISC mysteries]

Mystery

Intel *converts* CISC ones into RISC ones, and generate microoperations.

Intelligent CISC-RISC decoder

Consumes around 2% of the chip area. Good or bad?

ISA Tradeoffs

- Simplicity
- Performance, power, cost, time to market
- RISC vs CISC
- Fixed vs Variable encoding
- Endianness
- #registers per instruction
- Code size
- Open sourced?

How Easy?

A billion-dollar idea ☺

i) requires changes to microarchitecture.
ii) requires changes to ISA.
iii) both (i) and (ii)

Think about the trade-offs ☺
Does it affect the system stack?

Thaagatchari