Lecture-2 (Instruction Set Architecture)
CS422-Spring 2018

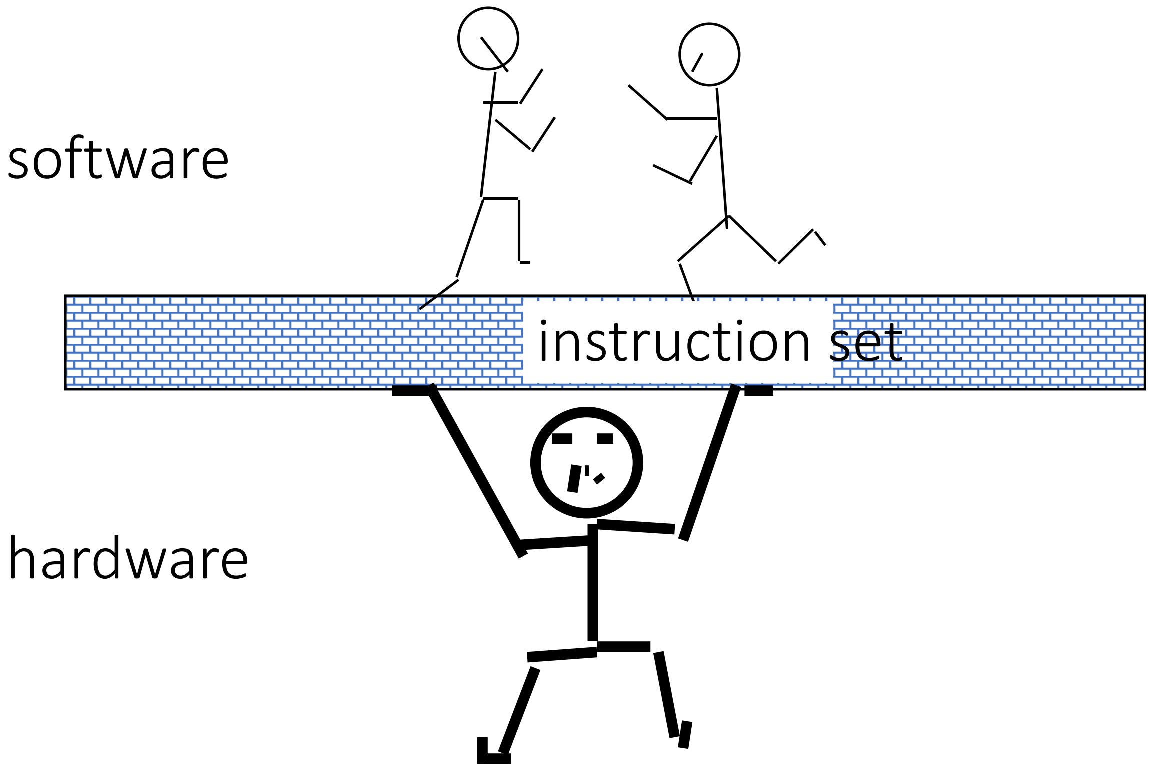
Biswa@CSE-IITK

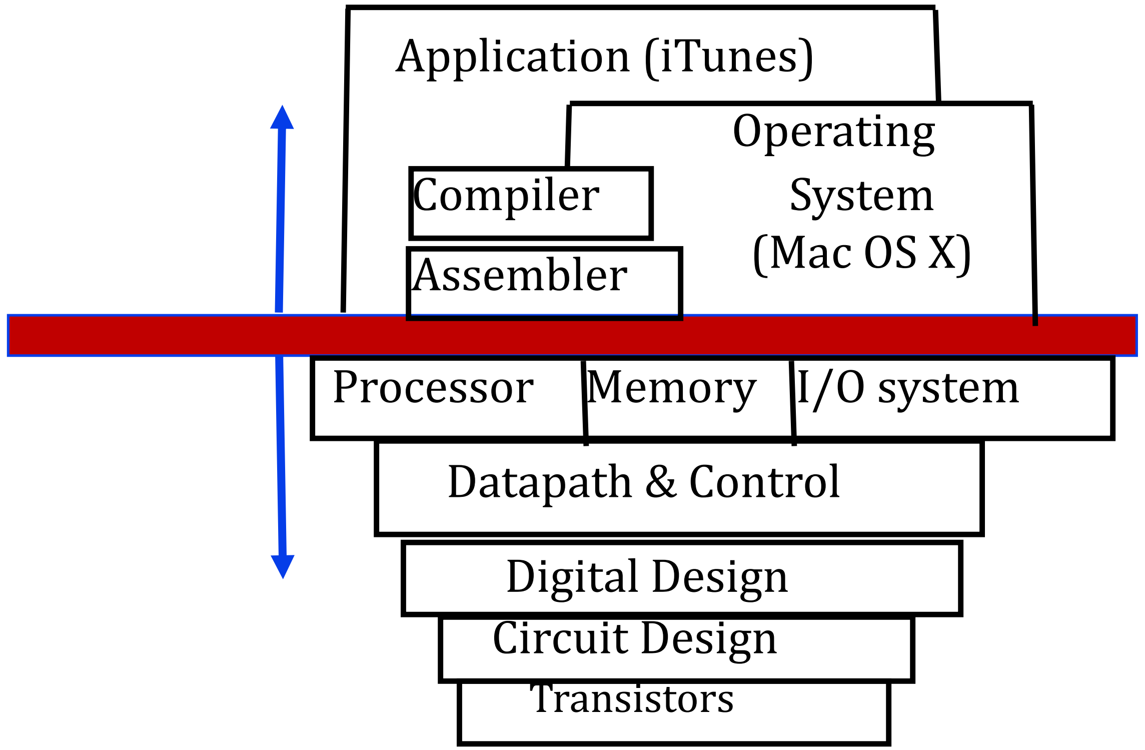
ISA

… 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

Instruction Set Architecture (ISA)





ISA

Syntax: ADD $8 $9 $10 Semantics: $8 = $9 + $10

Bitfield:

opcode	**rs**	**rt**	**rd**	**shamt**	**funct**

"R-Format"

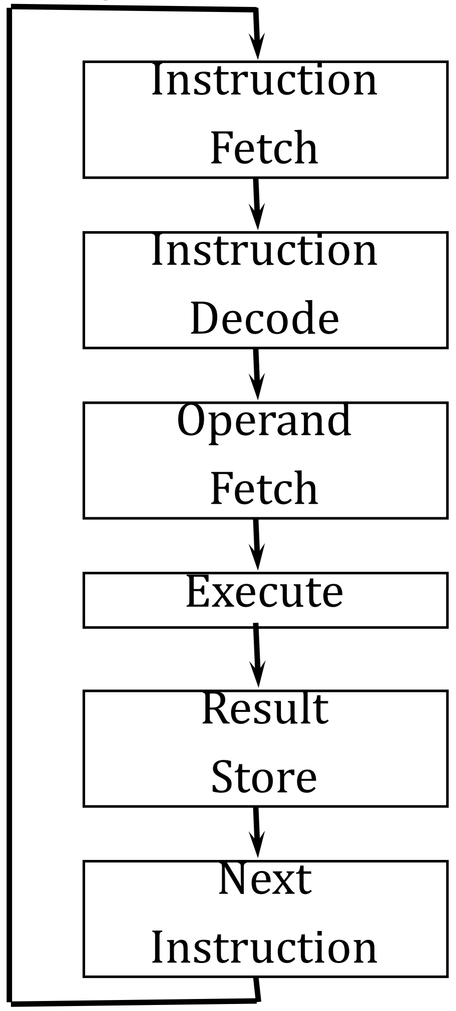
Binary: 000000 01001 01010 01000 00000 100000

In Hexadecimal: 012A4020

Why opcode and funct?
Why shamt?
+1 point: Piazza

ISA

Syntax: ADD $8 $9 $10 Semantics: $8 = $9 + $10



Fetch next inst from memory:012A4020

opcode	rs	rt	rd	shamt	funct

Decode fields to get : ADD $8 $9 $10

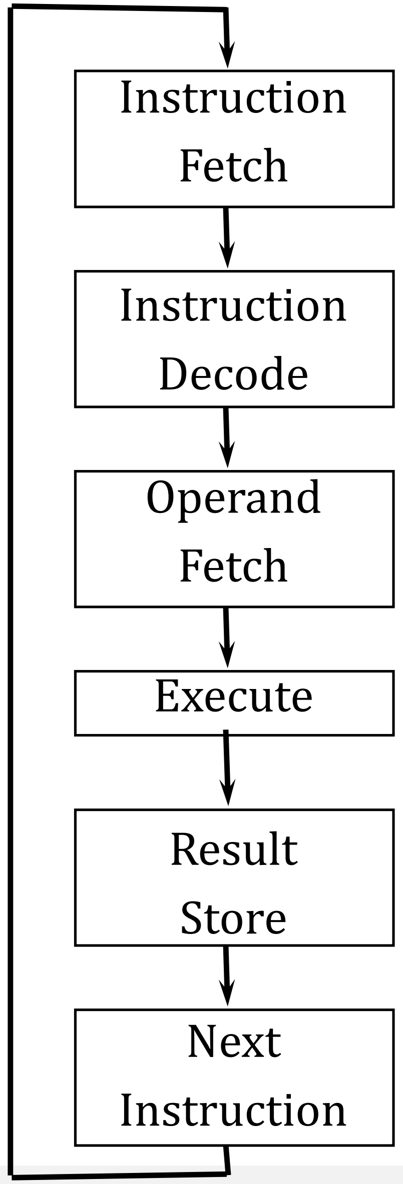
“Retrieve” register values: $9 $10

Add $9 to $10

Place this sum in $8

Prepare to fetch instruction that follows the ADD in the program.

Memory Instructions: `LW $1,32($2)`



Fetch the load inst from memory

opcode	rs	rt	offset

"I-Format"

Decode fields to get : LW $1, 32($2)

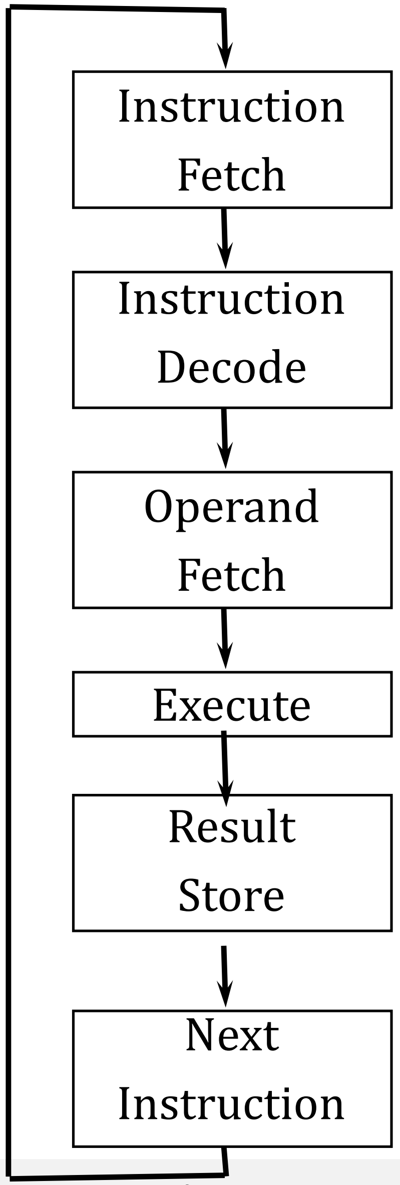
"Retrieve" register value: $2

Compute memory address: 32 + $2

Load memory address contents into: $1

Prepare to fetch instr. that follows the LW in the program.

Branch Instructions: `BEQ $1,$2,25`



Fetch branch inst from memory

opcode	rs	rt	offset

"I-Format"

Decode fields to get: BEQ $1, $2, 25

"Retrieve" register values: $1, $2

Compute if we take branch: $1 == $2 ?

ALWAYS prepare to fetch instr. that follows the BEQ in the program ("delayed branch"). IF we take branch, the instr. we fetch AFTER that instruction is PC + 4 + 100.

J-type during branches

The Contract: An Architect’s Contract

- To the program, it appears that instructions execute in the correct order defined by the ISA.
- As each instruction completes, the machine state (regs, mem) appears to the program to obey the ISA.
- What the machine actually does is up to the hardware designers, as long as the contract is kept.

ISA

- ISA
  - Agreed upon interface between software and hardware
    - SW/compiler assumes, HW promises
  - What the software writer needs to know to write and debug system/user programs
- Microarchitecture
  - Specific implementation of an ISA
  - Not visible to the software
- Microprocessor
  - **ISA, uarch**, circuits
  - "Architecture" = ISA + microarchitecture

ISA

- Basic element of the HW/SW interface

- Consists of
  - opcode: what the instruction does

  - operands: ?

Elements of an ISA

- Instruction processing style
  - Specifies the number of “operands” an instruction “operates” on and how it does so
  - 0, 1, 2, 3 address machines: 0-address: stack machine (push A, pop A, op)
    - 1-address: accumulator machine (ld A, st A, op A)
    - 2-address: 2-operand machine (one is both source and dest)
    - 3-address: 3-operand machine (source and dest are separate)
  - Tradeoffs? Larger operate instructions vs. more executed operations
    - Code size vs. execution time vs. on-chip memory space

ISA vs uarch

- What is part of ISA vs. Uarch?
  - Gas pedal: interface for "acceleration"
  - Internals of the engine: implement "acceleration"
- Implementation (uarch) can be various as long as it satisfies the specification (ISA)
  - Add instruction vs. Adder implementation
    - Bit serial, ripple carry, carry lookahead adders are all part of microarchitecture
  - x86 ISA has many implementations: 286, 386, 486, Pentium, Pentium Pro, Pentium 4, Core, ...
- Microarchitecture usually changes faster than ISA
  - Few ISAs (x86, ARM, SPARC, MIPS, Alpha) but many uarchs. *Why?*

ISA

- Instructions
  - Opcodes, Addressing Modes, Data Types
  - Instruction Types and Formats
  - Registers, Condition Codes
- Memory
  - Address space, Addressability, Alignment
  - Virtual memory management
- Call, Interrupt/Exception Handling, Access Control, Priority/Privilege
- I/O: memory-mapped vs. instr, Task/thread Management, Power and Thermal Management
- Multi-threading support, Multiprocessor support

intel

Intel® 64 and IA-32 Architectures
Software Developer's Manual

Volume 1:
Basic Architecture

Uarch

- Implementation of the ISA under specific design constraints and goals
- Anything done in hardware without exposure to software
  - Pipelining
  - In-order versus out-of-order instruction execution
  - Memory access scheduling policy
  - Speculative execution
  - Superscalar processing (multiple instruction issue?)
  - Clock gating
  - Caching? Levels, size, associativity, replacement policy
  - Prefetching?
  - Voltage/frequency scaling? Error correction?

ISA vs uarch

- ADD instruction's opcode
- Number of general purpose registers
- Number of ports to the register file
- Number of cycles to execute the MUL instruction
- Whether or not the machine employs pipelined instruction execution

- Remember
  - Microarchitecture: Implementation of the ISA under specific design constraints and goals

Why Have Instructions

- Alternatives
  - Special purpose hardware
  - Reconfigurable hardware (FPGAs)
  - Minimize everything as much as possible?
- Simplifies interface
  - Software knows what is available
  - Hardware knows what needs to be implemented
- Abstraction protects software and hardware
  - Software can run on new machines
  - Hardware can run old software