

18-447

Computer Architecture

Lecture 2: Fundamental Concepts and ISA

Prof. Onur Mutlu

Carnegie Mellon University

Spring 2012, 1/23/2012

Reminder: Homeworks for Next Two Weeks

- Homework 0
 - Due today, right before lecture
- Homework 1
 - Due next Monday (Jan 30), 11:59pm, on Blackboard
 - MIPS warmup, ISA concepts, basic performance evaluation

Reminder: Lab Assignment 1

- A functional C-level simulator for a subset of the MIPS ISA
- Due Friday Feb 3, at the end of Friday lab

- Start early, you will have a lot to learn
- Homework 1 and Lab 1 are synergistic
 - Homework questions are meant to help you in the Lab

A Note on Hardware vs. Software

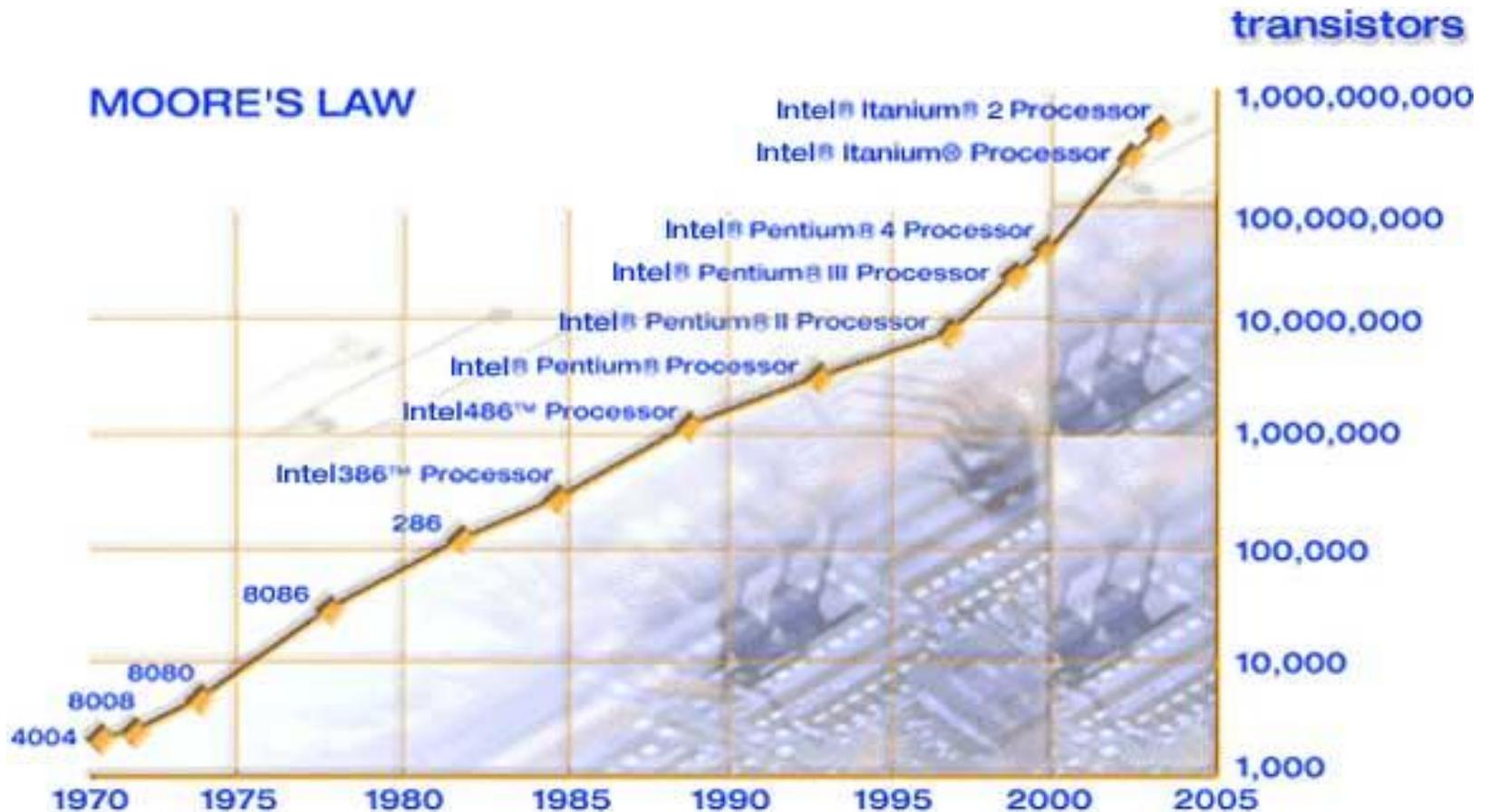
- This course is classified under “Computer Hardware”
- However, you will be much more capable if you master both hardware and software (and the interface between them)
 - Can develop better software if you understand the underlying hardware
 - Can design better hardware if you understand what software it will execute
 - Can design a better computing system if you understand both
- This course covers the HW/SW interface and microarchitecture
 - We will focus on tradeoffs and how they affect software

Why Study Computer Architecture?

What is Computer Architecture?

- The science and art of designing, selecting, and interconnecting hardware components and designing the hardware/software interface to create a computing system that meets functional, performance, energy consumption, cost, and other specific goals.
- We will soon distinguish between the terms *architecture*, and *microarchitecture*.

Moore's Law



Moore, “Cramming more components onto integrated circuits,”
Electronics Magazine, 1965. Component counts double every year

What Do We Use These Transistors for?

Why Study Computer Architecture?

- **Enable better systems:** make computers faster, cheaper, smaller, more reliable, ...
 - By exploiting advances and changes in underlying technology/circuits
- **Enable new applications**
 - Life-like 3D visualization 20 years ago?
 - Virtual reality?
 - Personal genomics?
- **Enable better solutions** to problems
 - Software innovation is built into trends and changes in computer architecture
 - > 50% performance improvement per year has enabled
- **Understand why computers work the way they do**

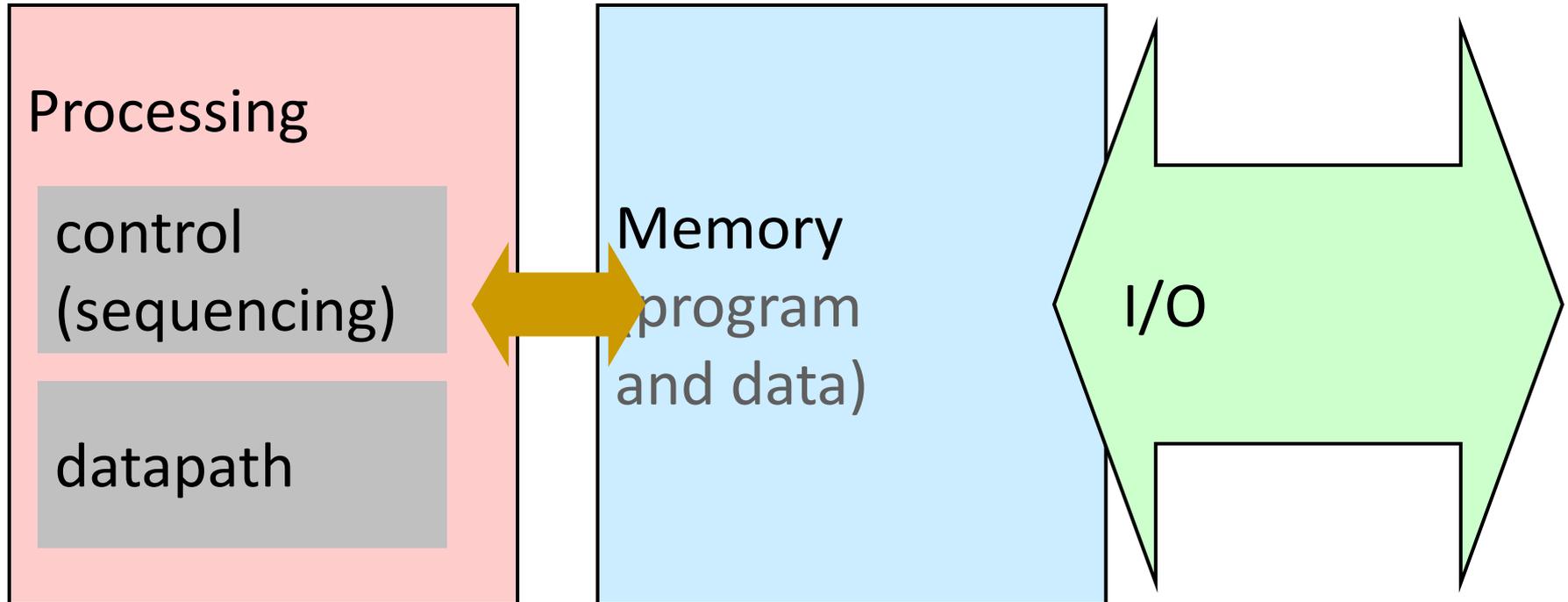
Computer Architecture Today

- Today is a very exciting time to study computer architecture
- Industry is in a large paradigm shift (to multi-core)
- Many problems motivating and caused by the shift
 - Power/energy constraints
 - Complexity of design → multi-core
 - Technology scaling → new technologies
 - Memory wall/gap
 - Reliability wall/issues
 - Programmability wall/problem
- You can revolutionize the way computers are built, if you understand both the hardware and the software (and change each accordingly)
 - Book: Kuhn, “[The Structure of Scientific Revolutions](#)” (1962)

Fundamental Concepts

What is A Computer?

- We will cover all three components



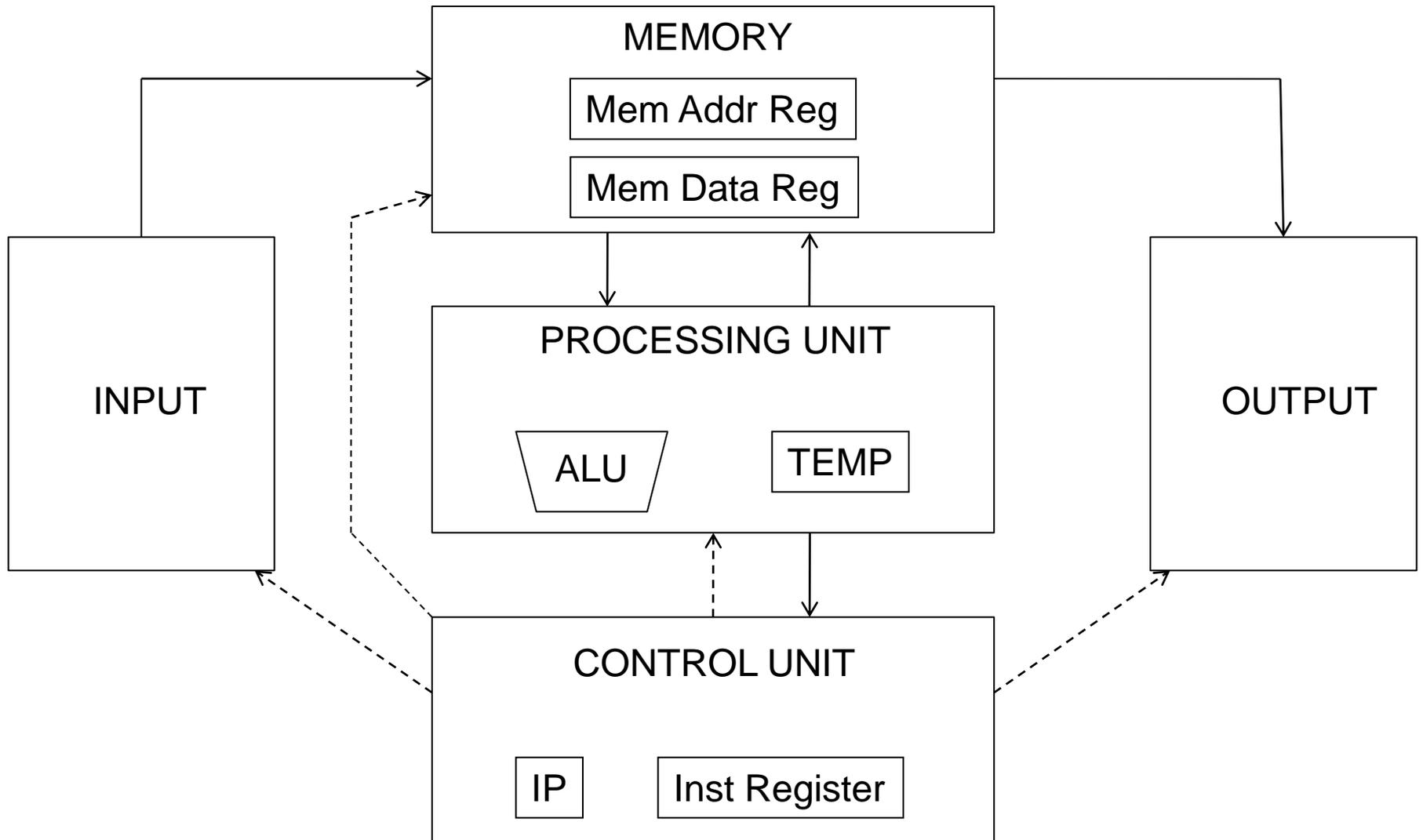
The Von Neumann Model/Architecture

- Also called *stored program computer* (instructions in memory). Two key properties:
- Stored program
 - Instructions stored in a linear memory array
 - Memory is unified between instructions and data
 - The interpretation of a stored value depends on the control signals
When is a value interpreted as an instruction?
- Sequential instruction processing
 - One instruction processed (fetched, executed, and completed) at a time
 - Program counter (instruction pointer) identifies the current instr.
 - Program counter is advanced sequentially except for control transfer instructions

The Von Neumann Model/Architecture

- Recommended reading
 - Burks, Goldstein, von Neumann, “[Preliminary discussion of the logical design of an electronic computing instrument](#),” 1946.
 - Patt and Patel book, Chapter 4, “The von Neumann Model”
- Stored program
- Sequential instruction processing

The Von-Neumann Model (of a Computer)



Aside: Dataflow Model (of a Computer)

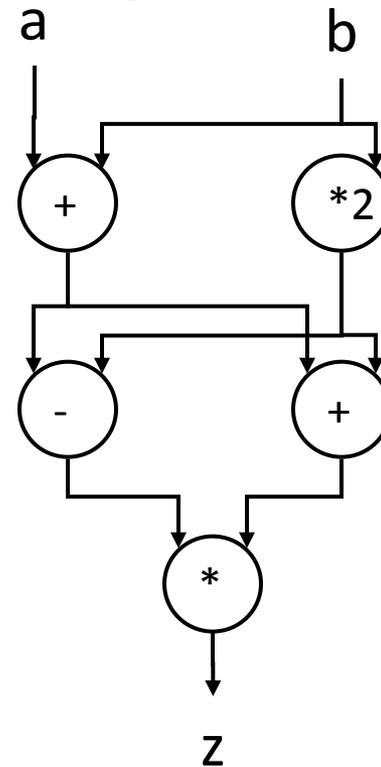
- Von Neumann model: An instruction is fetched and executed in **control flow order**
 - As specified by the **instruction pointer**
 - Sequential unless explicit control flow instruction

- Dataflow model: An instruction is fetched and executed in **data flow order**
 - i.e., when its operands are ready
 - i.e., there is **no instruction pointer**
 - Instruction ordering specified by data flow dependence
 - Each instruction specifies “who” should receive the result
 - An instruction can “fire” whenever all operands are received
 - Potentially many instructions can execute at the same time
 - Inherently more parallel

Aside: von Neumann vs Dataflow

- Consider a von Neumann program
 - What is the significance of the program order?
 - What is the significance of the storage locations?

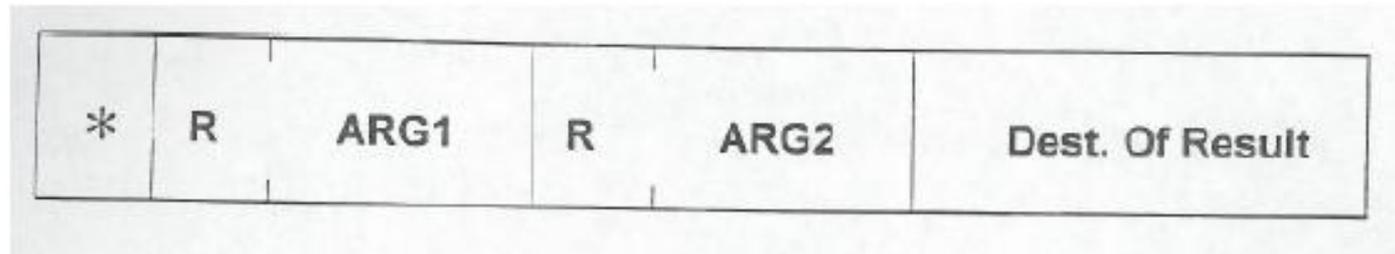
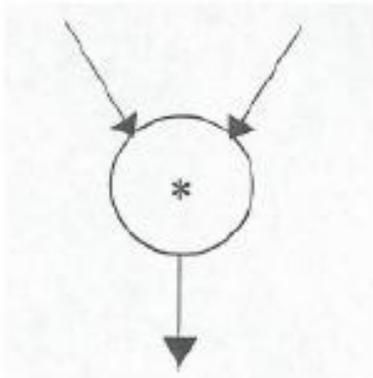
```
v <= a + b;  
w <= b * 2;  
x <= v - w  
y <= v + w  
z <= x * y
```



- Which model is more natural to you as a programmer?

Aside: More on Data Flow

- In a data flow machine, a program consists of data flow nodes
 - A data flow node fires (fetched and executed) when all its inputs are ready
 - i.e. when all inputs have tokens
- Data flow node and its ISA representation



Aside: ISA-level Tradeoff: Instruction Pointer

- Do we need an instruction pointer in the ISA?
 - Yes: Control-driven, sequential execution
 - An instruction is executed when the IP points to it
 - IP automatically changes sequentially (except control flow instructions)
 - No: Data-driven, parallel execution
 - An instruction is executed when all its operand values are available (**data flow**)

- Tradeoffs: MANY high-level ones
 - Ease of programming (for average programmers)?
 - Ease of compilation?
 - Performance: Extraction of parallelism?
 - Hardware complexity?

ISA vs. Microarchitecture Level Tradeoff

- A similar tradeoff (control vs. data-driven execution) can be made at the microarchitecture level
- ISA: Specifies how the programmer sees instructions to be executed
 - Programmer sees a sequential, control-flow execution order vs.
 - Programmer sees a data-flow execution order
- Microarchitecture: How the underlying implementation actually executes instructions
 - Microarchitecture can execute instructions in any order as long as it obeys the semantics specified by the ISA when making the instruction results visible to software
 - Programmer should see the order specified by the ISA

Let's Get Back to the Von Neumann Model

- But, if you want to learn more about dataflow...
- Dennis and Misunas, "A preliminary architecture for a basic data-flow processor," ISCA 1974.
- Gurd et al., "The Manchester prototype dataflow computer," CACM 1985.
- A later 447 lecture, 740, 742

The Von-Neumann Model

- All major *instruction set architectures* today use this model
 - x86, MIPS, SPARC, Alpha, ARM, POWER
- Underneath (at the microarchitecture level), the execution model of almost all *implementations (or, microarchitectures)* is very different
 - Pipelined instruction execution: *Intel 80486 uarch*
 - Multiple instructions at a time: *Intel Pentium uarch*
 - Out-of-order execution: *Intel Pentium Pro uarch*
 - Separate instruction and data caches
- But, what happens underneath that is not consistent with the von Neumann model is not exposed to software
 - Difference between ISA and microarchitecture

What is Computer Architecture?

- **ISA+implementation definition:** The science and art of designing, selecting, and interconnecting hardware components and designing the hardware/software interface to create a computing system that meets functional, performance, energy consumption, cost, and other specific goals.
- **Traditional (only ISA) definition:** “The term *architecture* is used here to describe the attributes of a system as seen by the programmer, i.e., the conceptual structure and functional behavior as distinct from the organization of the dataflow and controls, the logic design, and the physical implementation.” *Gene Amdahl, IBM Journal of R&D, April 1964*

ISA vs. Microarchitecture

■ ISA

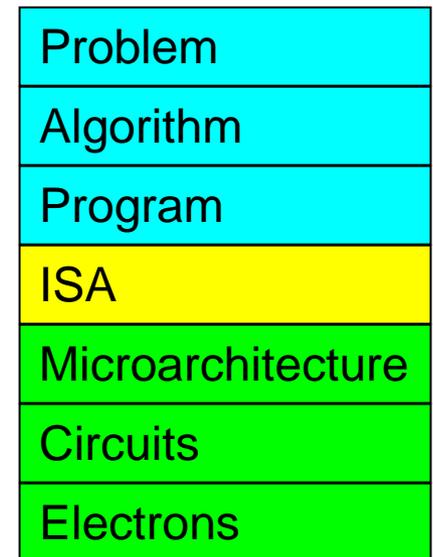
- Agreed upon interface between software and hardware
 - SW/compiler assumes, HW promises
- What the software writer needs to know to write and debug system/user programs

■ Microarchitecture

- Specific implementation of an ISA
- Not visible to the software

■ Microprocessor

- **ISA, uarch**, circuits
- “Architecture” = ISA + microarchitecture



ISA vs. Microarchitecture

- What is part of ISA vs. Uarch?
 - Gas pedal: interface for “acceleration”
 - Internals of the engine: implements “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, ...
- Microarchitecture usually changes faster than ISA
 - Few ISAs (x86, SPARC, MIPS, Alpha) but many uarchs
 - *Why?*

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

Volume 1:
Basic Architecture

Microarchitecture

- 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?

Property of 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

Design Point

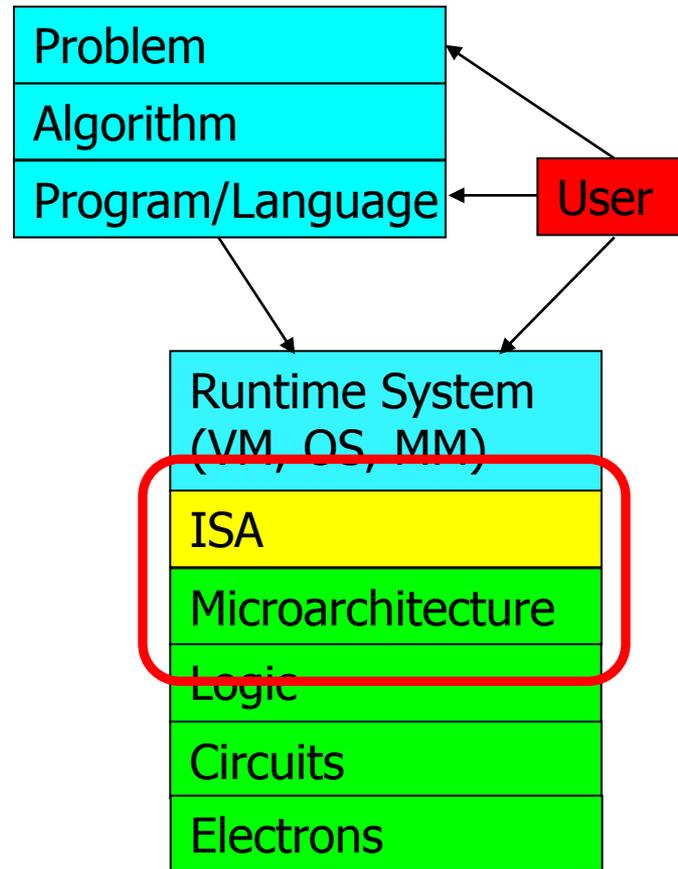
- A set of design considerations and their importance
 - **leads to tradeoffs** in both ISA and uarch
- Considerations
 - Cost
 - Performance
 - Maximum power consumption
 - Energy consumption (battery life)
 - Availability
 - Reliability and Correctness (or is it?)
 - Time to Market
- Design point determined by the “Problem” space (application space)

Problem
Algorithm
Program
ISA
Microarchitecture
Circuits
Electrons

Tradeoffs: Soul of Computer Architecture

- ISA-level tradeoffs
- Microarchitecture-level tradeoffs
- System and Task-level tradeoffs
 - How to divide the labor between hardware and software
- *Computer architecture is the science and art of making the appropriate trade-offs to meet a design point*
 - *Why art?*

Why Is It (Somewhat) Art?



- We do not (fully) know the future

We did not cover the following slides in lecture.
These are for your preparation for the next lecture.

ISA Principles and Tradeoffs

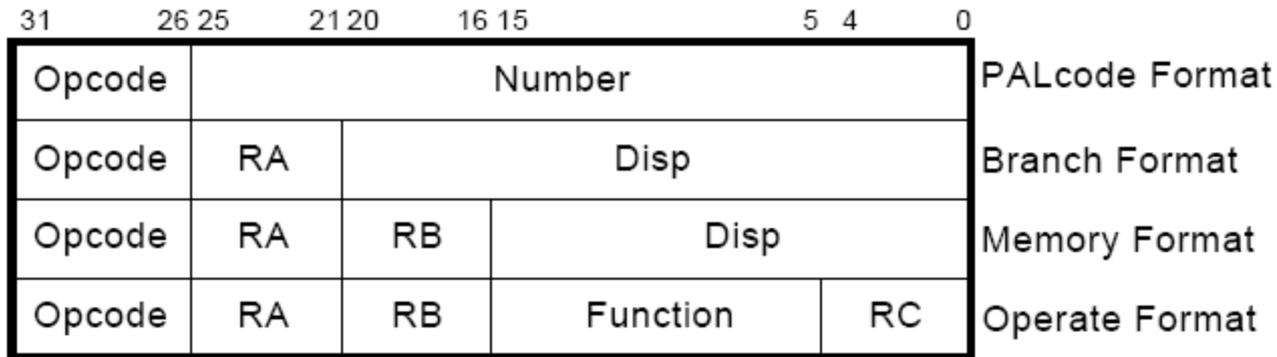
Many Different ISAs Over Decades

- x86
- PDP-x: Programmed Data Processor (PDP-11)
- VAX
- IBM 360
- CDC 6600
- SIMD ISAs: CRAY-1, Connection Machine
- VLIW ISAs: Metaflow, Cydrome, IA-64 (EPIC)
- PowerPC, POWER
- RISC ISAs: Alpha, MIPS, SPARC, ARM

- What are the fundamental differences?
 - E.g., how instructions are specified and what they do
 - E.g., how complex are the instructions

Instruction

- Basic element of the HW/SW interface
- Consists of
 - opcode: what the instruction does
 - operands: who it is to do it to
- Example from Alpha:



Set of Instructions, Encoding, and Spec

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADD ⁺	0001			DR			SR1			A	op.spec					
AND ⁺	0101			DR			SR1			A	op.spec					
BR	0000			n	z	p	PCoffset9									
JMP	1100			000			BaseR			000000						
JSR(R)	0100			A	operand.specifier											
LDB ⁺	0010			DR			BaseR			boffset6						
LDW ⁺	0110			DR			BaseR			offset6						
LEA ⁺	1110			DR			PCoffset9									
RTI	1000			000000000000												
SHF ⁺	1101			DR			SR			A	D	amount4				
STB	0011			SR			BaseR			boffset6						
STW	0111			SR			BaseR			offset6						
TRAP	1111			0000			trapvect8									
XOR ⁺	1001			DR			SR1			A	op.spec					
not used	1010															
not used	1011															

- Example from LC-3b ISA
 - http://www.ece.utexas.edu/~patt/11s.460N/handouts/new_byte.pdf

- Aside: concept of “bit steering”
 - A bit in the instruction determines the interpretation of other bits

What Are the Elements of An ISA?

■ Instruction sequencing model

- Control flow vs. data flow
- Tradeoffs?

■ 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
 - 1-address: accumulator machine
 - 2-address: 2-operand machine (one is both source and dest)
 - 3-address: 3-operand machine (source and dest are separate)
- Tradeoffs? See your homework question
 - Larger instructions vs. more executed operations
 - Code size vs. execution time

Examples

- PDP-11: A 2-address machine
 - PDP-11 ADD: 4-bit opcode, 2 6-bit operand specifiers
 - Why? Limited bits to specify an instruction
 - Disadvantage: One source operand is always clobbered with the result of the instruction
 - *How do you ensure you preserve the old value of the source?*

- X86: A 2-address machine
- Alpha: A 3-address machine
- MIPS?