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.
An Enabler: Moore’s Law

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

Image source: Intel
Number of transistors on an integrated circuit doubles ~ every two years

Number of transistors on an integrated circuit doubles ~ every two years

What Do We Use These Transistors for?

- Your readings for this week should give you an idea...

- Recommended

- Required for Review as part of HW 1
Recommended Reading


- Only 3 pages

- A quote:

  "With unit cost falling as the number of components per circuit rises, by 1975 economics may dictate squeezing as many as 65 000 components on a single silicon chip."

- Another quote:

  "Will it be possible to remove the heat generated by tens of thousands of components in a single silicon chip?"
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?
  - Self-driving cars?
  - Personalized genomics? Personalized medicine?

- **Enable better solutions to problems**
  - Software innovation is built on trends and changes in computer architecture
    - > 50% performance improvement per year has enabled this innovation

- Understand why computers work the way they do
Today is a very exciting time to study computer architecture.

Industry is in a large paradigm shift (to multi-core and beyond) – many different potential system designs possible.

Many difficult problems motivating and caused by the shift:
- Huge hunger for data and new data-intensive applications
- Power/energy/thermal constraints
- Complexity of design
- Difficulties in technology scaling
- Memory wall/gap
- Reliability problems
- Programmability problems
- Security and privacy issues

No clear, definitive answers to these problems.
These problems affect all parts of the computing stack – if we do not change the way we design systems.

No clear, definitive answers to these problems

Many new demands from the top (Look Up)

Fast changing demands and personalities of users (Look Up)

Many new issues at the bottom (Look Down)
Computer Architecture Today (III)

- Computing landscape is very different from 10-20 years ago
- Both UP (software and humanity trends) and DOWN (technologies and their issues), FORWARD and BACKWARD, and the resulting requirements and constraints

Every component and its interfaces, as well as entire system designs are being re-examined
Computer Architecture Today (IV)

- You can revolutionize the way computers are built, if you understand both the hardware and the software (and change each accordingly)

- You can invent new paradigms for computation, communication, and storage

- Recommended book: Thomas Kuhn, “The Structure of Scientific Revolutions” (1962)
  - Pre-paradigm science: no clear consensus in the field
  - Normal science: dominant theory used to explain/improve things (business as usual); exceptions considered anomalies
  - Revolutionary science: underlying assumptions re-examined
Computer Architecture Today (IV)

- You can revolutionize the way computers are built, if you understand both the hardware and the software (and change each accordingly)

- You can invent new paradigms for computation, communication, and storage

- Recommended book: Thomas Kuhn, “The Structure of Scientific Revolutions” (1962)
  - Pre-paradigm science: no clear consensus in the field
  - Normal science: dominant theory used to explain/improve things (business as usual); exceptions considered anomalies
  - Revolutionary science: underlying assumptions re-examined
Computer Architecture Today (IV)

- You can revolutionize the way computers are built, if you understand both the hardware and the software (and change each accordingly)

- You can invent new paradigms for computation, communication...

- Recommended book: Thomas Kuhn, "The Structure of Scientific Revolutions" (1962)
  - Pre-paradigm science: no clear consensus in the field
  - Normal science: dominant theory used to explain/improve things (business as usual); exceptions considered anomalies
  - Revolutionary science: underlying assumptions re-examined
You can revolutionize the way computers are built, if you understand both the hardware and the software (and change each accordingly).

You can invent new paradigms for computation, communication, and storage.


- Pre-paradigm science: no clear consensus in the field
- Normal science: dominant theory used to explain/improve things (business as usual); exceptions considered anomalies
- Revolutionary science: underlying assumptions re-examined
Takeaways

- It is an exciting time to be understanding and designing computing architectures

- Many challenging and exciting problems in platform design
  - That noone has tackled (or thought about) before
  - That can have huge impact on the world’s future

- Driven by huge hunger for data (Big Data), new applications, ever-greater realism, ...
  - We can easily collect more data than we can analyze/understand

- Driven by significant difficulties in keeping up with that hunger at the technology layer
  - Three walls: Energy, reliability, complexity, security, scalability
... but, first ...

- Let’s understand the fundamentals...

- You can change the world only if you understand it well enough...
  - Especially the past and present dominant paradigms
  - And, their advantages and shortcomings – tradeoffs
  - And, what remains fundamental across generations
  - And, what techniques you can use and develop to solve problems
Fundamental Concepts
What is A Computer?

- Three key components
- Computation
- Communication
- Storage (memory)
What is A Computer?

- We will cover all three components (with a focus on memory and I/O)
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
  
- 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

When is a value interpreted as an instruction?
The Von Neumann Model/Architecture

- **Recommended readings**
  - 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)
The Von Neumann Model (of a Computer)

Q: Is this the only way that a computer can operate?

A: No.

Qualified Answer: No, but it has been the dominant way
  - i.e., the dominant paradigm for computing
  - for N decades
The 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
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?

\[
\begin{align*}
v & \leftarrow a + b; \\
w & \leftarrow b \times 2; \\
x & \leftarrow v - w \\
y & \leftarrow v + w \\
z & \leftarrow x \times y
\end{align*}
\]

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

*Conditional

*Relational

*Barrier Synch
An Example Data Flow Program
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 for 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...


- A later lecture or course

- If you are really impatient:
  - [http://www.youtube.com/watch?v=D2uue7izU2c](http://www.youtube.com/watch?v=D2uue7izU2c)
The Von-Neumann Model

- All major *instruction set architectures* today use this model
  - x86, ARM, MIPS, SPARC, Alpha, 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 (ISA-only) 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: 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
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
  - Time to Market

- Design point determined by the “Problem” space (application space), the intended users/market
Application Space

- Dream, and they will appear...

Other examples of the application space that continue to drive the need for unique design points are the following:

1) scientific applications such as those whose computations control nuclear power plants, determine where to drill for oil, and predict the weather;
2) transaction-based applications such as those that handle ATM transfers and e-commerce business;
3) business data processing applications, such as those that handle inventory control, payrolls, IRS activity, and various personnel record keeping, whether the personnel are employees, students, or voters;
4) network applications, such as high-speed routing of Internet packets, that enable the connection of your home system to take advantage of the Internet;
5) guaranteed delivery (a.k.a. real time) applications that require the result of a computation by a certain critical deadline;
6) embedded applications, where the processor is a component of a larger system that is used to solve the (usually) dedicated application;
7) media applications such as those that decode video and audio files;
8) random software packages that desktop users would like to run on their PCs.

Each of these application areas has a very different set of characteristics. Each application area demands a different set of tradeoffs to be made in specifying the microprocessor to do the job.
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 (applications, users, market)

New demands from the top (Look Up)

New issues and capabilities at the bottom (Look Down)

New demands and personalities of users (Look Up)
Why Is It (Somewhat) Art?

And, the future is not constant (it changes)!

Changing demands at the top (Look Up and Forward)

Changing issues and capabilities at the bottom (Look Down and Forward)

- Problem
- Algorithm
- Program/Language
- Runtime System (VM, OS, MM)
- ISA
- Microarchitecture
- Logic
- Circuits
- Electrons

Changing demands and personalities of users (Look Up and Forward)
How Can We Adapt to the Future

- This is part of the task of a good computer architect

- Many options (bag of tricks)
  - Keen insight and good design
  - Good use of fundamentals and principles
    - Efficient design
    - Heterogeneity
    - Reconfigurability
    - ...
  - Good use of the underlying technology
  - ...


We Covered a Lot of This in Digital Circuits & Computer Architecture
Logic Design, Verilog, FPGAs

ISA (MIPS)

Single-cycle Microarchitectures

Multi-cycle and Microprogrammed Microarchitectures

Pipelining

Issues in Pipelining: Control & Data Dependence Handling, State Maintenance and Recovery, ...

Out-of-Order Execution

Processing Paradigms (SIMD, VLIW, Systolic, Dataflow, ...)
Covered Concurrent Execution Paradigms

- Pipelining
- Out-of-order execution
- Dataflow (at the ISA level)
- Superscalar Execution
- VLIW
- SIMD Processing (Vector and array processors, GPUs)
- Decoupled Access Execute
- Systolic Arrays
Digital Circuits Materials for Review (I)

- All Digital Circuits Lecture Videos Are Online:
  - https://www.youtube.com/playlist?list=PL5Q2soXY2Zi_QedyPWtRmFUJ2F8DdYP7I

- All Slides and Assignments Are Online:
Digital Circuits Materials for Review (II)

- Particularly useful and relevant lectures for this course

- Pipelining and Dependence Handling (Lecture 14-15)
  - https://www.youtube.com/watch?v=f522l7Q-t7g
  - https://www.youtube.com/watch?v=7XXgZIbBLIs

- Out-of-order execution (Lecture 16-17)
  - https://www.youtube.com/watch?v=0E4QTDZ2OBA
  - https://www.youtube.com/watch?v=vwLyEbIzyfI
Review Cache Lectures (from Spring 2018)

- Memory Organization and Technology (Lecture 23b)
  - https://www.youtube.com/watch?v=rvBdJ1ZLo2M

- Memory Hierarchy and Caches (Lecture 24)
  - https://www.youtube.com/watch?v=sweCA3836C0

- More Caches (Lecture 25a)
  - https://www.youtube.com/watch?v=kMUZKjaPNWo

- Virtual Memory (Lecture 25b)
  - https://www.youtube.com/watch?v=na-JL1nVTSU
This Course

- We will have more emphasis on
  - The memory system
  - Multiprocessing & multithreading
  - Reconfigurable computing
  - Customized accelerators
  - ...

- We will also likely dig deeper on some Digital Circuits concepts (as time permits)
  - ISA
  - Branch handling
  - GPUs
  - ...
Tentative Agenda (Upcoming Lectures)

- The memory hierarchy
- Caches, caches, more caches (high locality, high bandwidth)
- Main memory: DRAM
- Main memory control, scheduling, interference, management
- Memory latency tolerance and prefetching techniques
- Non-volatile memory & emerging technologies

- Multiprocessors
- Coherence and consistency
- Interconnection networks
- Multi-core issues
- Multithreading
- Acceleration, customization, heterogeneity