Digital Logic and Computer Architecture – CS322M

Satyajit Das

Assistant Professor, Dept CSE

Co-Founder – Revin Tech

satyajit.das@iitg.ac.in

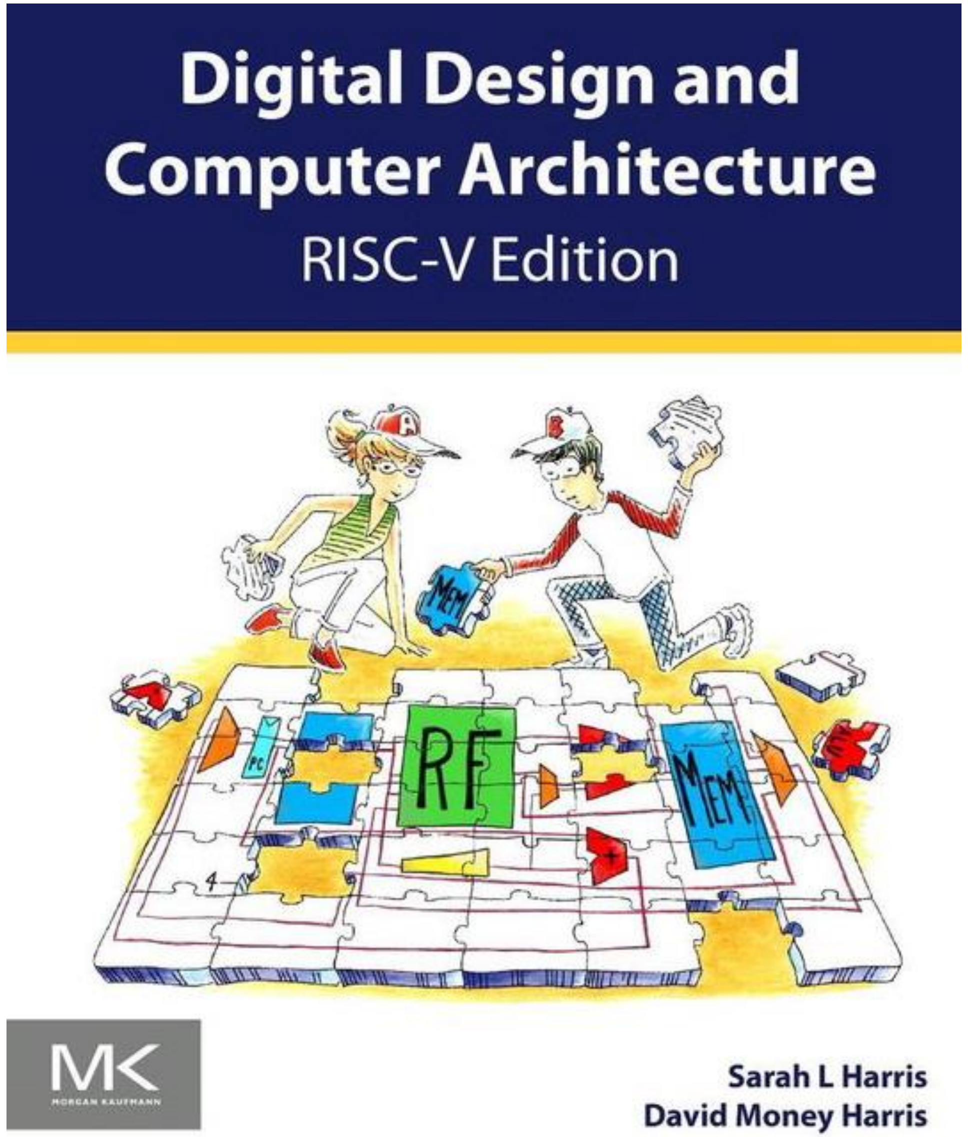
In This Lecture

- Motivation for the lecture series
  - Why should you care about
    Digital Design and Computer Architectures
- The Art of Managing Complexity
  - How can extremely complex systems be designed?
- Organization of the Course
  - Information on the lectures
  - Laboratory Exercises
  - Exam
  - How to get help when you are stuck

Course Logistics (3-0-0-6)

- Lectures in 5104
  - Class lectures + Take home lab practices
- On the class site
  - Presentations used in the class
  - Textbook

- Course TAs
  - Sagarika Das (256101018)
  - Ritwik Tiwari (254101048)


Digital Design and Computer Architecture
RISC-V Edition
Sarah L Harris
David Money Harris

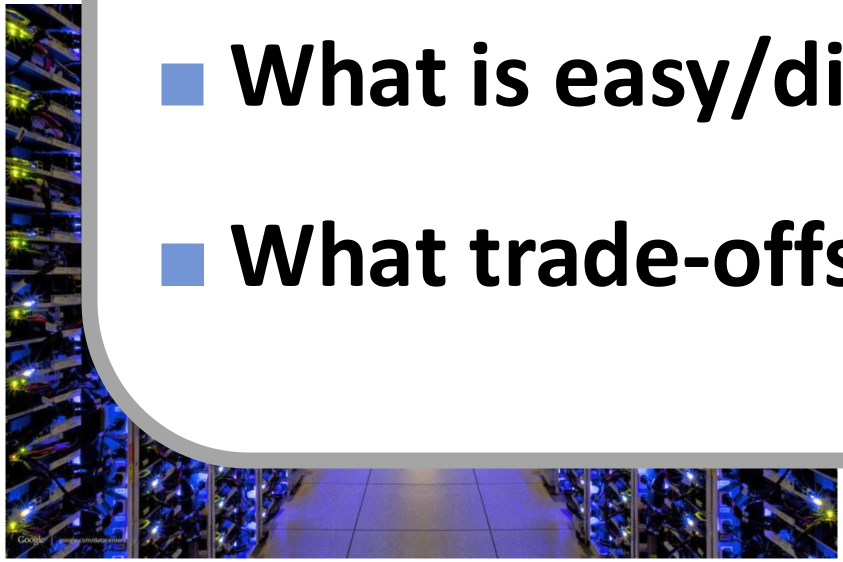
Course Logistics (3-0-0-6)

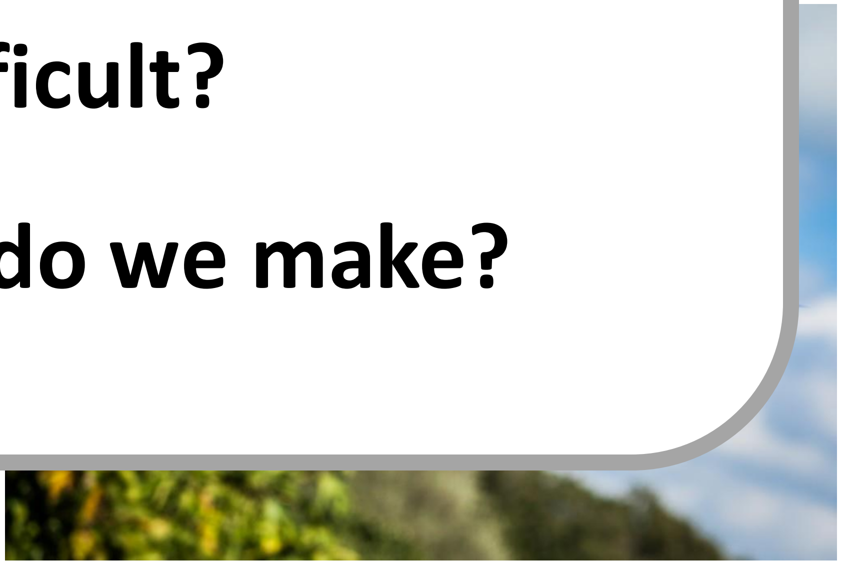
Component	**Weightage**	**Remarks**
Midterm Exam (Theory)	20%	Closed book / descriptive
End-Sem Exam (Theory)	40%	Cumulative, includes design questions
GitHub-based Lab Submissions	20%	Weekly Verilog code + outputs + commits
Quiz 1 (Theory) (**August 26**)	10%	Short, closed-book quiz
Attendance	5%	Based on class participation
Final Demo / Viva	5%	Live demonstration of a GitHub task

Course Logistics (3-0-0-6)

- Lab practices – 20%
    - 5% correctness of code
    - 10% output validation + README
    - 5% timeliness and commit history (not all code pushed last minute)

Computers are everywhere





- What goes into them?
- How are they built?
- What is easy/difficult?
- What trade-offs do we make?

The Purpose of This Course Is That You:

- Learn what’s under the hood of a computer
- Learn the principles of digital design
- Learn to systematically debug increasingly complex systems
- Design and build a microprocessor

Goal

- Be able to understand a statement such as:

  “The microarchitecture is a three-way superscalar, pipelined architecture. Three-way superscalar means that by using parallel processing techniques, the processor is able on average to decode, dispatch, and complete execution of (retire) three instructions per clock cycle. To handle this level of instruction throughput, the P6 processor family uses a decoupled, 12-stage superpipeline that supports out-of-order instruction execution.”

Taken from: http://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-manual-325462.pdf

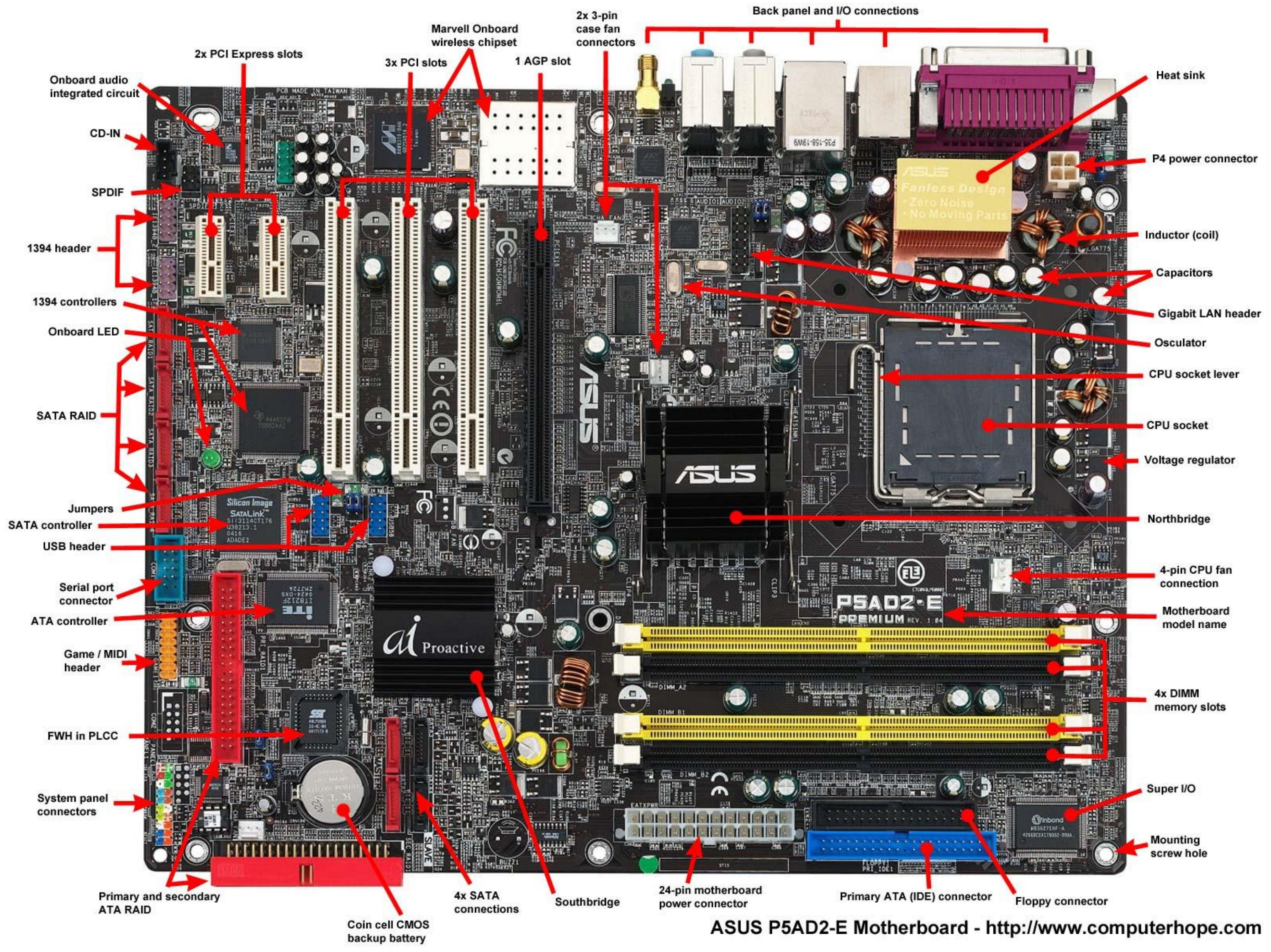
What Is This Lecture About?

Computers

What Is This Lecture About?

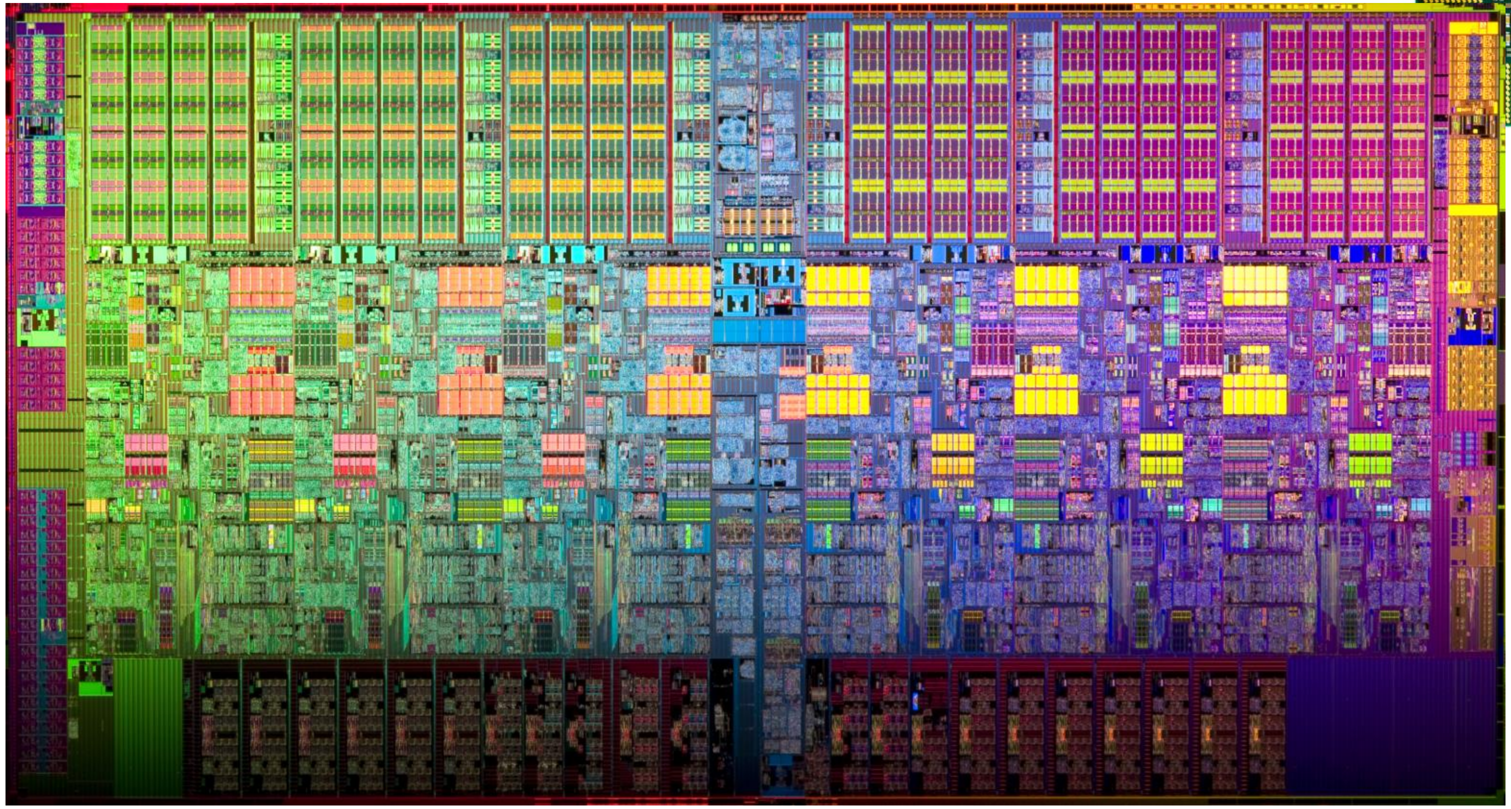
What's inside?

What Is This Lecture About?



ASUS P5AD2-E Motherboard - http://www.computerhope.com

What Is This Lecture About?



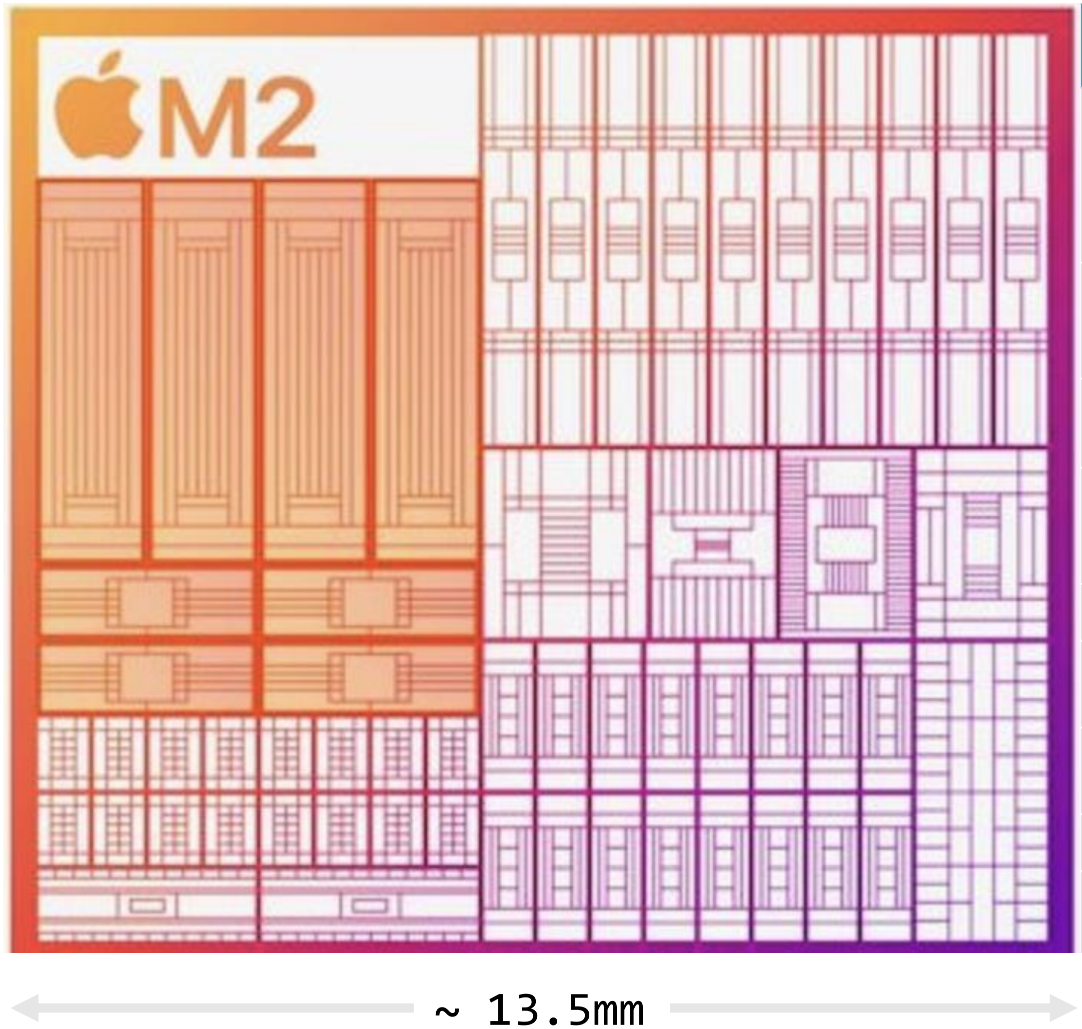
How can we build them ?

Intel Westmere: http://www.intel.com/pressroom/archive/releases/2010/20100316comp_sm.htm

The Art of Managing Complexity

- Abstraction
- Discipline
- The Three -Y’s
    - Hierarchy
    - Modularity
    - Regularity

An example: Apple M2

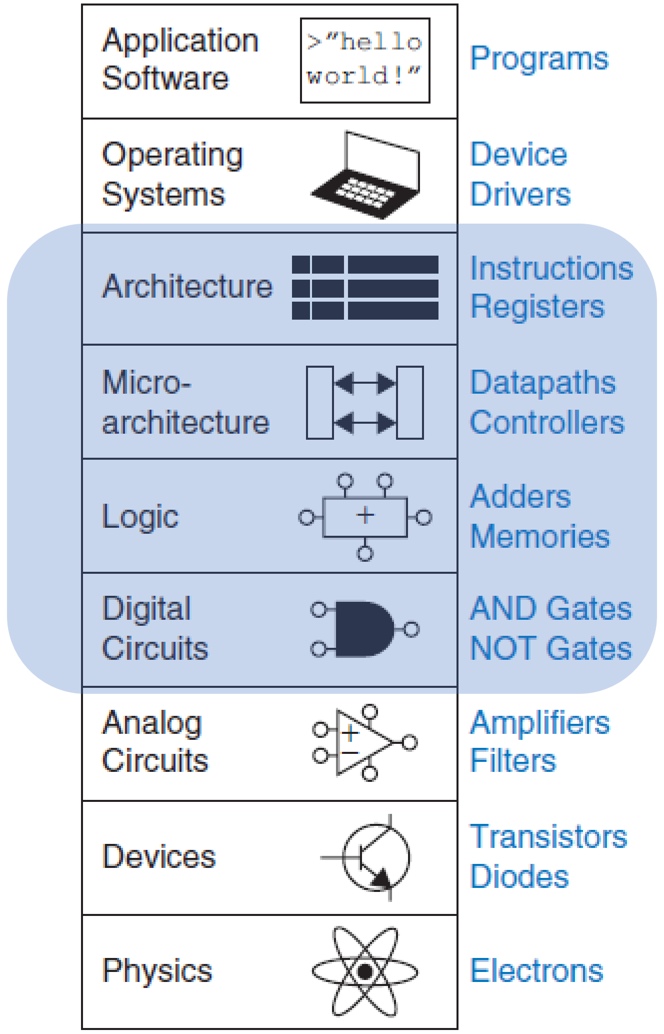


SoC	M2
CPU	4x High Performance (Avalanche?) 16MB Shared L2 4x High Efficiency (Blizzard?) 4MB Shared L2
GPU	"Next Generation" 10-Core 3.6 TFLOPS
Neural Engine	16-Core 15.8 TOPS
Memory Controller	LPDDR5-6400 8x 16-bit CH 100GB/sec Total Bandwidth (Unified)
Memory Capacity	24GB
Encode/ Decode	8K H.264, H.265, ProRes, ProRes RAW
USB	USB4/Thunderbolt 3 2x Ports
Transistors	20 Billion
Mfc. Process	"Second Generation 5nm" TSMC N5P?

https://www.anandtech.com/show/17431/apple-announces-m2-soc-apple-silicon-updated-for-2022

Abstraction

- **Hiding details when they are not important**



The Three -Y’s

- Hierarchy
  - A system is divided into modules of smaller complexity
- Modularity
  - Having well defined functions and interfaces
- Regularity
  - Encouraging uniformity, so modules can be easily re-used

From Real Problems to 1s and 0s

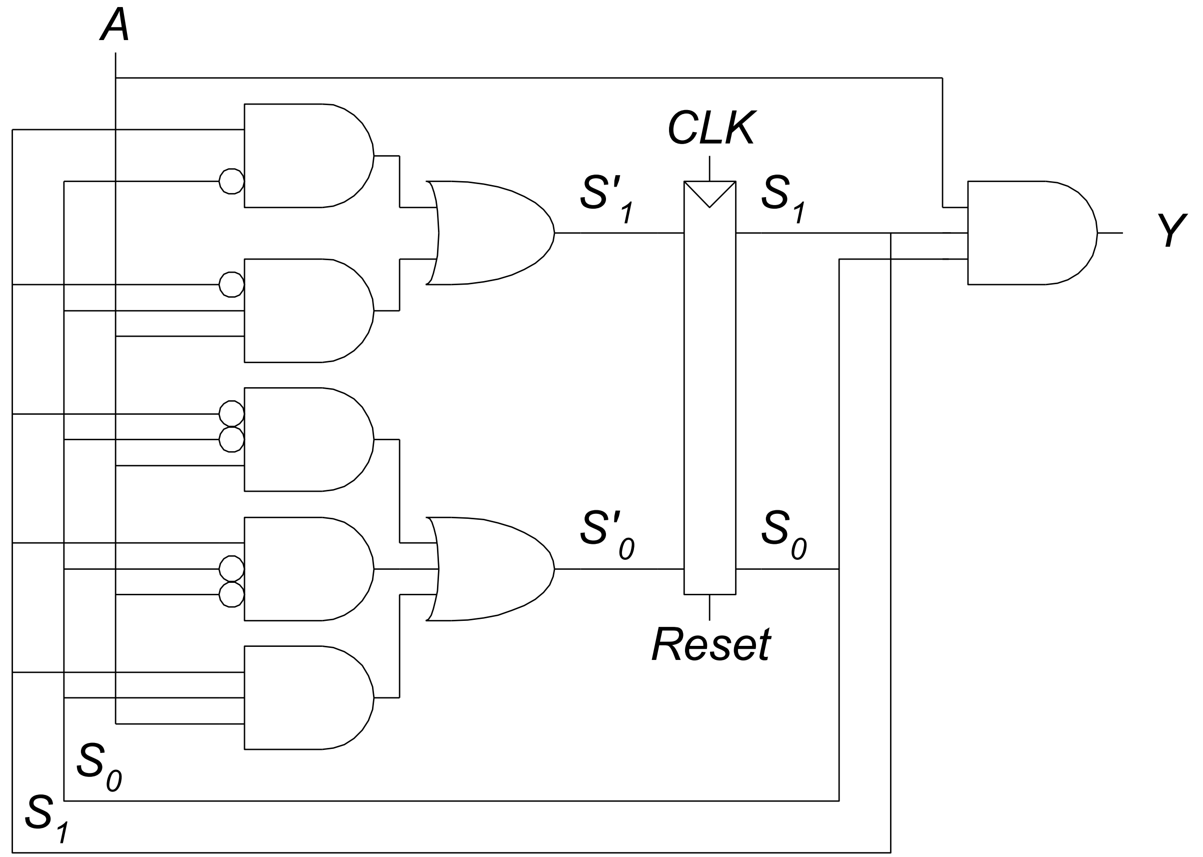
- Example Problem: You āre going to the cafeteria for lunch
  - You won't eat lunch ($\overline{\mathbf{E}}$)
  - If it is not open (**O**) or
  - If they only serve cabbage (**C**)

O	*C*	*E*
0	0	0
0	1	0
1	0	1
1	1	0

- We will learn how to formulate problems in truth tables

From Logic to Circuits

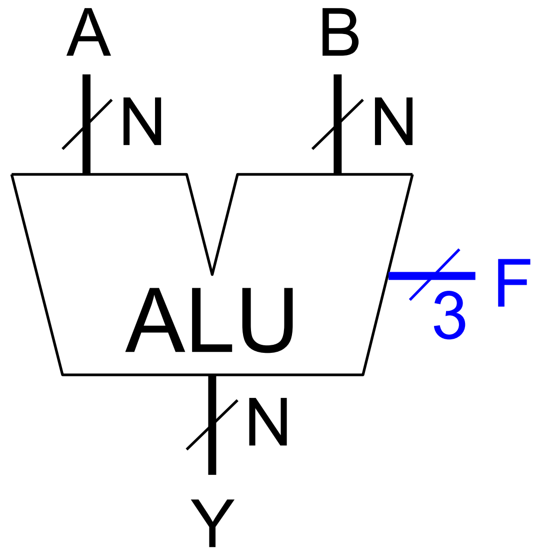
Example: Alyssa P. Hacker has a snail that crawls down a paper tape with 1's and 0's on it. The snail smiles whenever the last four digits it has crawled over are 1101.



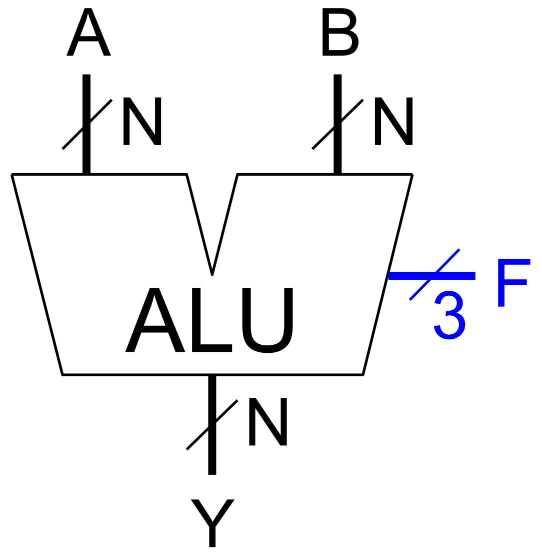
Verilog Will Help us Describe Circuits

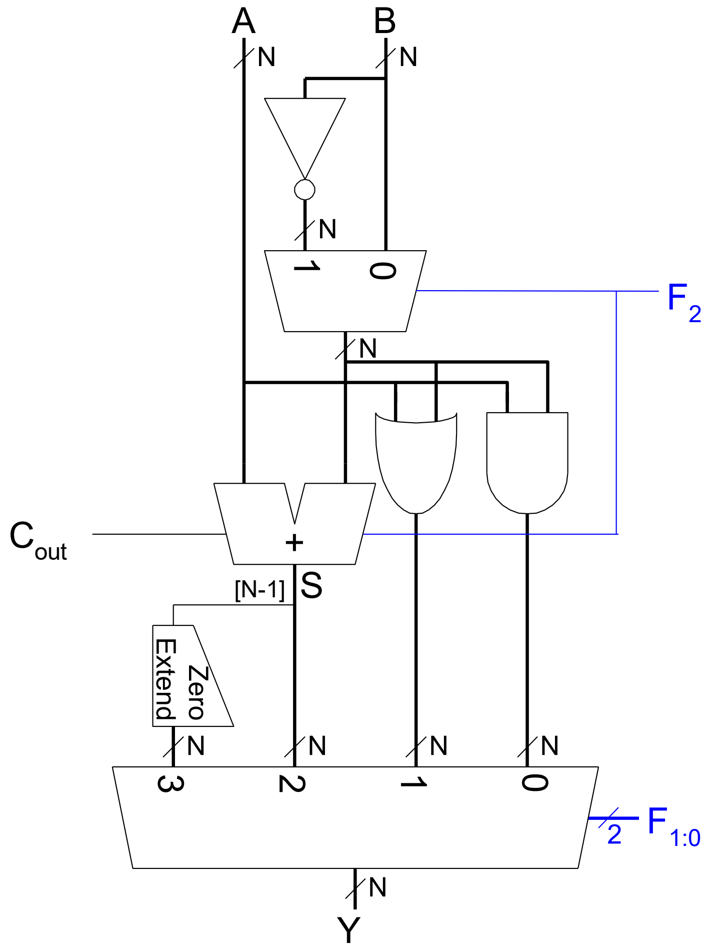
```
module divideby3FSM (input clk, input reset, output q);
   reg  [1:0] state, nextstate;

   parameter S0 = 2'b00;
   parameter S1 = 2'b01;
   parameter S2 = 2'b10;

   always @ (posedge clk, posedge reset) // state register
      if (reset) state <= S0;
      else       state <= nextstate;
   always @ (*)                          // next state logic
      case (state)
         S0:      nextstate = S1;
         S1:      nextstate = S2;
         S2:      nextstate = S0;
         default: nextstate = S0;
      endcase
   assign q = (state == S0);             // output logic
endmodule
```

How Can We Add/Multiply Binary Numbers?

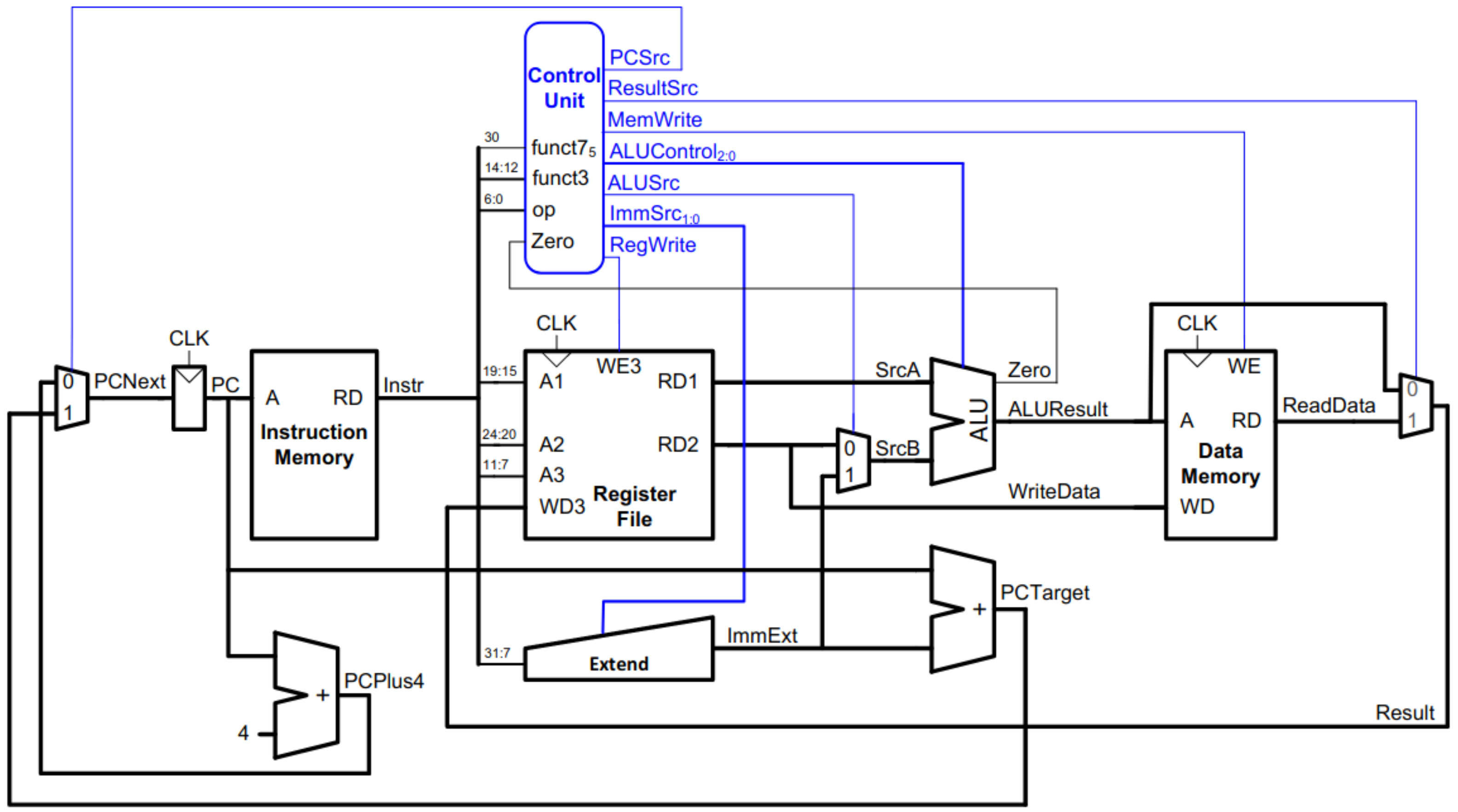


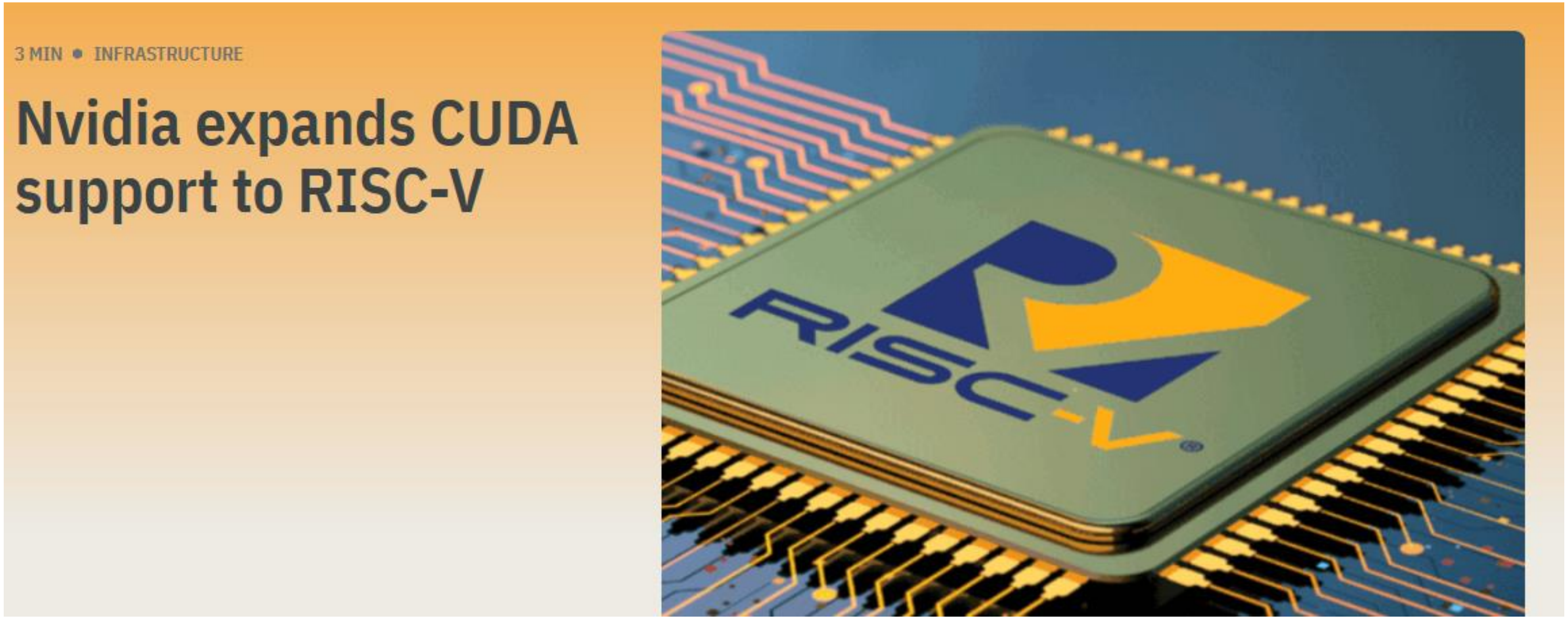
How Can We Add/Multiply Binary Numbers?





The Single-Cycle RISC-V Architecture





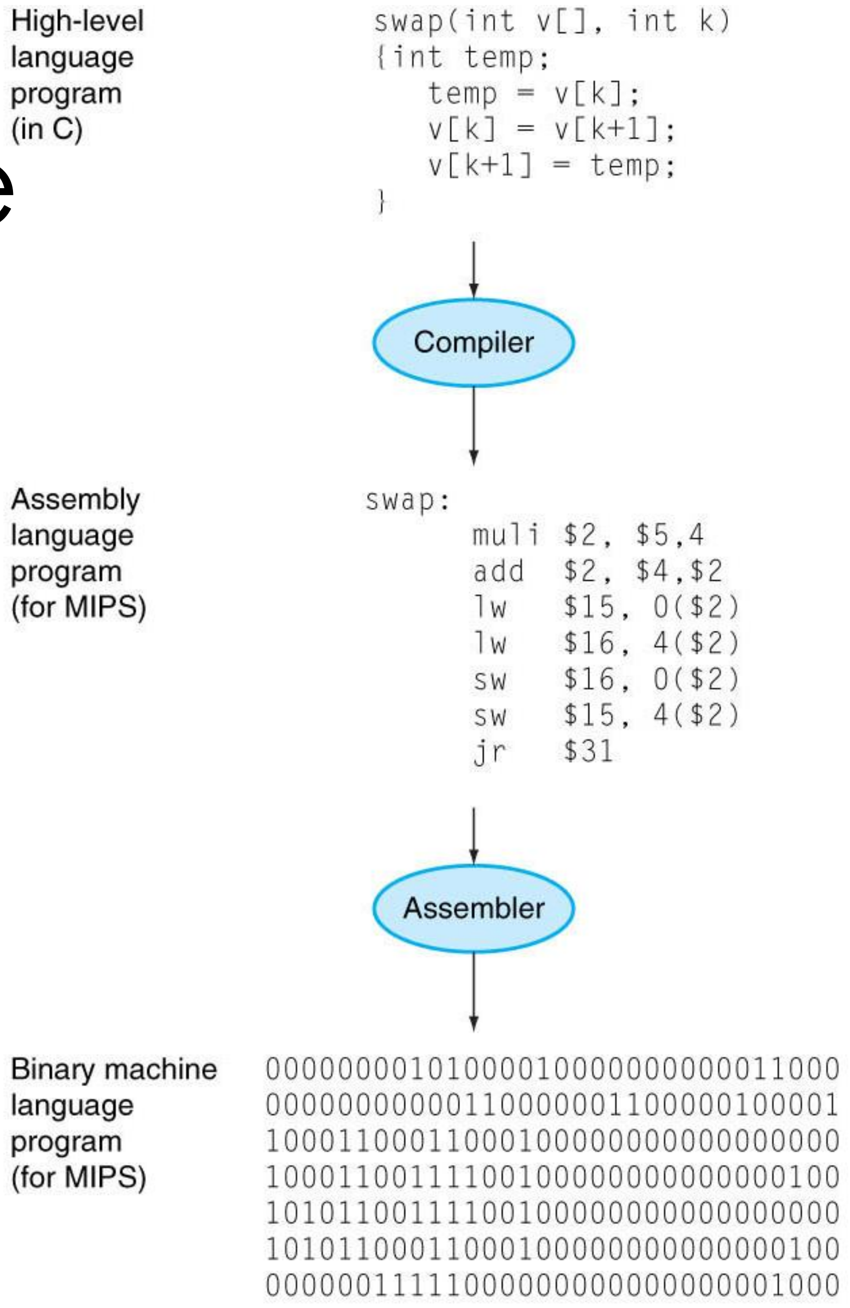
3 MIN ● INFRASTRUCTURE

Nvidia expands CUDA support to RISC-V

Source: www.techzine.eu

C to Machine Code

- Start writing program in a high-level language
- Compiler translates this into simple instructions that the processor will understand
- Assembler translates this into 1s and 0s that will control the processor


High-level language program (in C)
Compiler
Assembly language program (for MIPS)
Assembler
Binary machine language program (for MIPS)

```
swap(int v[], int k)
{int temp;
   temp = v[k];
   v[k] = v[k+1];
   v[k+1] = temp;
}
```

```
swap:
      muli $2, $5,4
      add  $2, $4,$2
      lw   $15, 0($2)
      lw   $16, 4($2)
      sw   $16, 0($2)
      sw   $15, 4($2)
      jr   $31
```

```
00000000101000010000000000011000
00000000000110000001100000100001
10001100011000100000000000000000
10001100111100100000000000000100
10101100111100100000000000000000
10101100011000100000000000000100
00000011111000000000000000001000
```