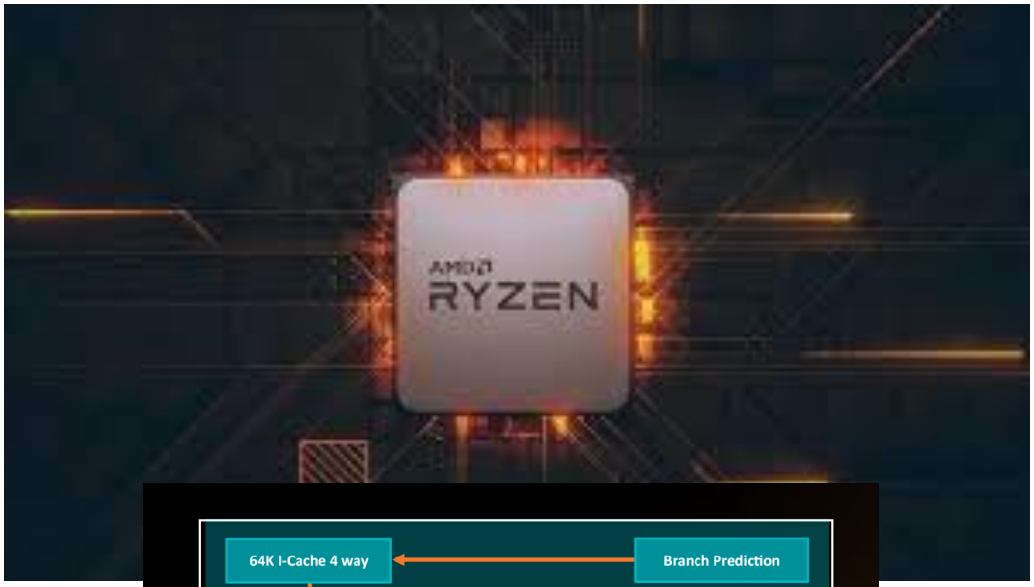
CS/ECE 552

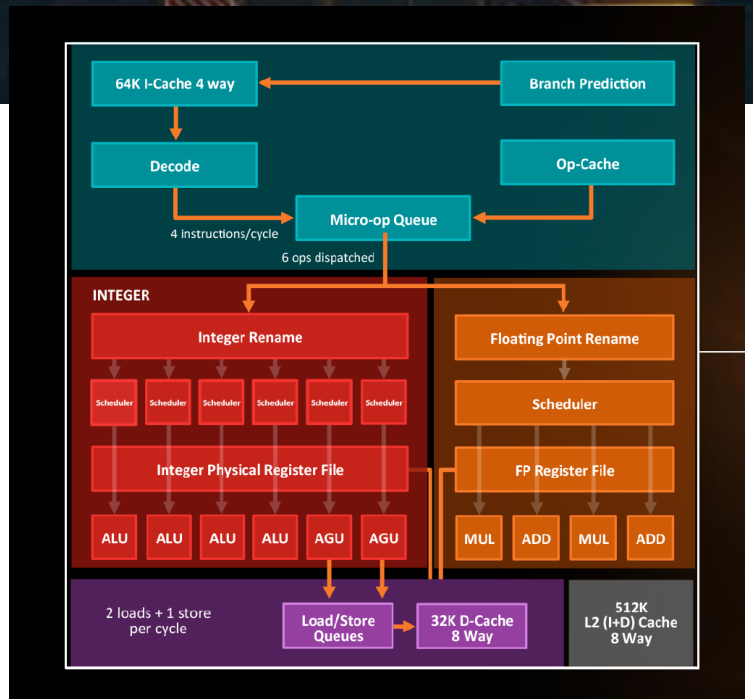
Course Introduction

Prof. Karu Sankaralingam
(based on slides by Profs. Hu, Lipasti, and San Miguel)
University of Wisconsin – Madison
Computer Sciences Department

Fall 2020

What is this course about?





Overview

- CS/ECE 552: basics of modern micro-processor design
- Related Courses
  - CS/ECE 252/352 (Prerequisites) – gates, logic, memory, organization
  - CS/ECE 252/354 (Prerequisites) – high-level language down to machine language interface or instruction set architecture (ISA)
  - ECE 551 – Verilog design of low level logic blocks
  - ECE 555 – Transistor level IC design
  - CS/ECE 752/757 – advanced topics of architecture, multi-core, parallel processing
  - CS 758 – advanced topics in architecture (recently: programming heterogeneous/parallel systems)
  - ME/CS/ECE 759 – GPU programming
  - Compiler (CS 536), Operating systems (CS 537) – system software

Coverage

THE UNIVERSITY of WISCONSIN MADISON

- Performance
- Instruction set architecture (ISA): MIPS
- Basic data path implementation of ISA
- Pipelined data path (in great detail)
- Cache and Virtual memory
- Arithmetic algorithms: multiplication, division
- I/O
- Advanced topics:
  - Superscalar, multicore, security, GPUs

Lecture Format

- Before the lecture
  - Try to do the readings from the book
- During the lecture
  - Video lecture, slides and recording will be made available
  - Two in-class Quizzes
- After the lecture
  - Will work on homework problems. Homework will be due at start of class

Expected Course Outcomes

THE UNIVERSITY of WISCONSIN MADISON

- Students will be able to:
  - use standard performance metrics to compare performance of different digital systems
  - compare and contrast different adder, multiplier, and divider implementations.
  - design a pipelined data path for a RISC (reduced instruction set computer) instruction set and be familiar with concepts of data dependence, pipelined hazards and out of order execution.
  - design basic data and control cache subsystems and understand basic memory organization
  - design a pipelined RISC micro-processor system with data cache using computer aided design tool and validate the correctness of the design using logic simulation.

CAD Tools

- Will use Verilog in class
- Install Verilog simulator
  - ModelSim
- Use remote desktop for simulation etc
- Can use Mentor local laptop install
  - Student license for ModelSim (usually good for 6 months): https://www.mentor.com/company/higher_ed/modelsim-student-edition
  - CAE/CSL lab machines already have ModelSim installed
  - **Next Thursday**: tutorial on setting ModelSim setup

Relationship to ECE 551

- ECE 551
  - Focuses on function block level digital design using Verilog.
  - Comprehensive coverage of Verilog
  - More details on synthesis
- CS/ECE 552
  - Focuses on core level (processor level) digital system design
  - Architecture issues: Pipelined data path, cache and virtual memory
  - Verilog language usage will be restricted to a basic subset
  - Verilog design practices and tutorials will be covered primarily during TA-led discussion sessions.
  - Students will be assumed to have basic knowledge of Verilog and be able to start by designing simple digital modules in class.

Administrative Details

- Using Piazza for discussion + announcements
  - https://piazza.com/wisc/fall2020/fa20ece552001/home
  - Post questions here
- Course website:
  http://pages.cs.wisc.edu/~karu/courses/cs552/fall2020/wiki/
  - Most important page is the **Course Calendar** page
  - Scores available on Canvas
  - Upload assignments to Canvas
- Lots more details in syllabus – please read

Staff & Office Hours

- Instructor: Karu Sankaralingam (CS 6367)
  - Office hours 9:15am to 10am on Tuesdays
  - Or by appointment
  - Email: karu@cs.wisc.edu
- TA: Guanzhou Hu
  - Office hours: TBD
  - Email: guanzhou.hu@wisc.edu

Textbook

THE UNIVERSITY of WISCONSIN MADISON

- David A. Patterson and John L. Hennessy, Computer Organization and Design: The Hardware Software Interface, Morgan Kaufmann Publishers, 5th Edition. ISBN: 9780124077263

- Note: there are three "5th" editions
  - We will be using the MIPS edition

Homework

- Homework
  - ~5 homework assignments, equally weighted
  - Due at start of class
  - Will also post practice HWs on some non-Verilog concepts
- Include components for the project

Course Project

THE UNIVERSITY of WISCONSIN MADISON

- Project
  - Implement processor for WISC-SP20 ISA
  - Three main (Verilog) phases
  - Extra credit points available
  - Only individual projects
  - Demo and submit written report

Course Grading

- Grading
  - Homework 20%
  - Quizzes 10%
  - Project 70%

Course Logistics

- Web-page
- Piazza
- CSL Account
- VPN
- Remote Desktop
- ModelSim brief demo
- No Knowledge of Verilog necessary

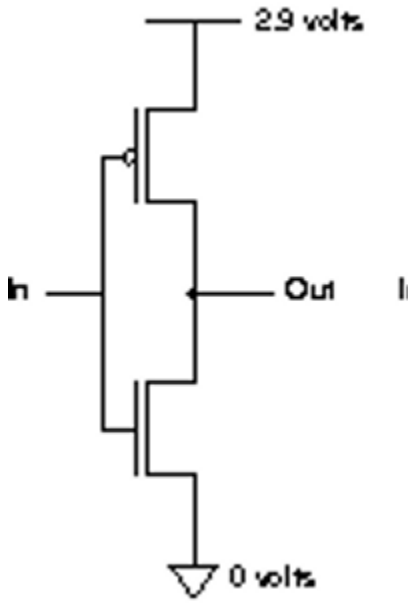
Boolean logic

$$X = \overline{A} \bullet \overline{B} \bullet C + \overline{A} \bullet B \bullet C + A \bullet \overline{B} \bullet C + A \bullet B \bullet \overline{C} + A \bullet B \bullet C$$

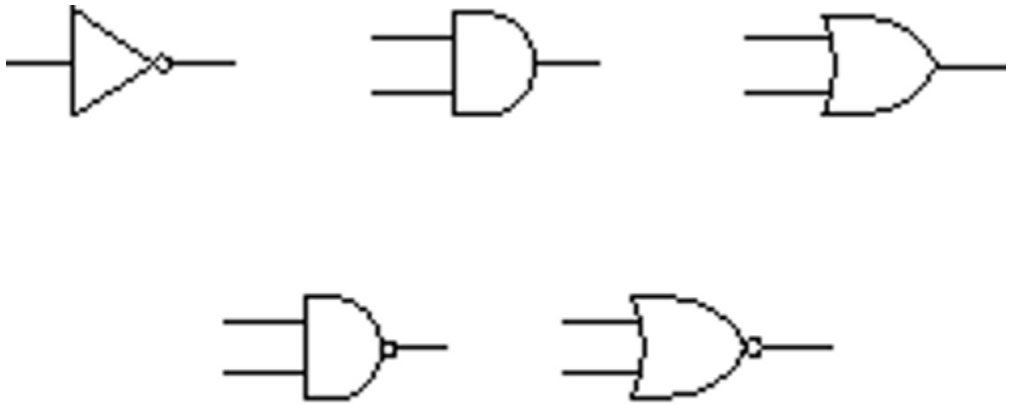
$$= \overline{A} \bullet C \bullet (\overline{B} + B) + A \bullet \overline{B} \bullet C + A \bullet B \bullet (\overline{C} + C)$$
$$= \overline{A} \bullet C \bullet (1) + A \bullet \overline{B} \bullet C + A \bullet B \bullet (1)$$
$$= \overline{A} \bullet C + A \bullet \overline{B} \bullet C + A \bullet B$$
$$= \overline{A} \bullet C + A \bullet (\overline{B} \bullet C + B)$$
$$= \overline{A} \bullet C + A \bullet (B + \overline{B}) \bullet (B + C)$$
$$= \overline{A} \bullet C + A \bullet (1) \bullet (B + C)$$
$$= \overline{A} \bullet C + A \bullet (B + C)$$
$$= \overline{A} \bullet C + A \bullet B + A \bullet C$$
$$= C \bullet (\overline{A} + A) + A \bullet B$$
$$= C \bullet (1) + A \bullet B$$
$$= A \bullet B + C$$

$$A \cup A = A, \qquad A \cap A = A,$$
$$A \cup A' = S, \qquad A \cap A' = \phi,$$
$$A \cup B = B \cup A, \qquad A \cap B = B \cap A,$$
$$A \cup S = S, \qquad A \cap S = A,$$
$$A \cup \phi = A, \qquad A \cap \phi = \phi,$$
$$A \cup (B \cup C) = (A \cup B) \cup C, \qquad A \cap (B \cap C) = (A \cap B) \cap C,$$
$$A \cup (B \cap C) = (A \cup B) \cap (A \cup C), \qquad A \cap (B \cup C) = (A \cap B) \cup (A \cap C),$$
$$(A \cup B)' = A' \cap B', \qquad (A \cap B)' = A' \cup B',$$

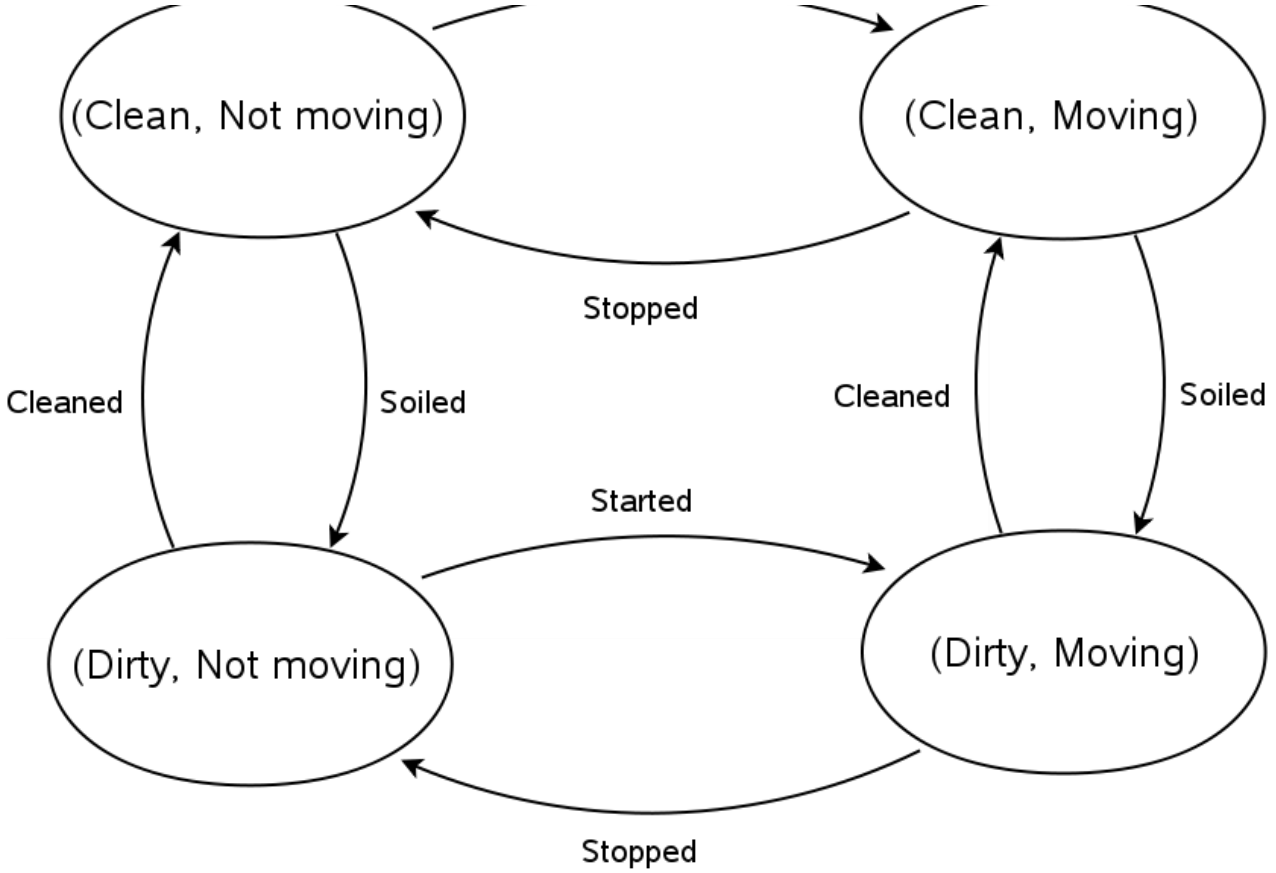
Transistors



Logic gates



State machines



Programming, c or java

```
#include<stdio.h>
main() {
        int c, first, last, middle, n, search, array[100];
        printf("Enter number of elements\n");
        scanf("%d",&n);
        printf("Enter %d integers\n", n);
        for ( c = 0 ; c < n ; c++ )
                scanf("%d",&array[c]);
        printf("Enter value to find\n");
        scanf("%d",&search);
        first = 0; last = n - 1; middle = (first+last)/2;
        …..
```

Assembly language

```
lw $t0, 4($gp)      # fetch N
mult $t0, $t0, $t0 # N*N
lw $t1, 4($gp)      #fetch N
ori $t2, $zero, 3  # 3
mult $t1, $t1, $t2 # 3*N
add $t2, $t0, $t1  # N*N + 3*N
sw $t2, 0($gp)      # i = ...
```

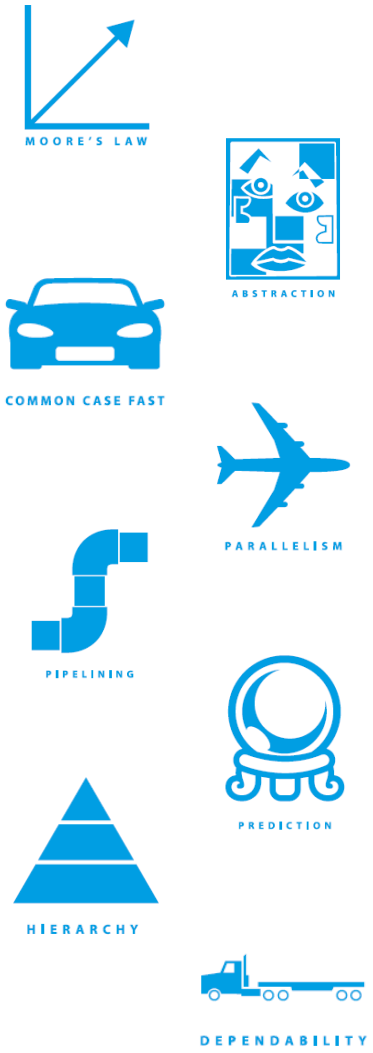
You DO NOT need to know Verilog

What You Will Learn

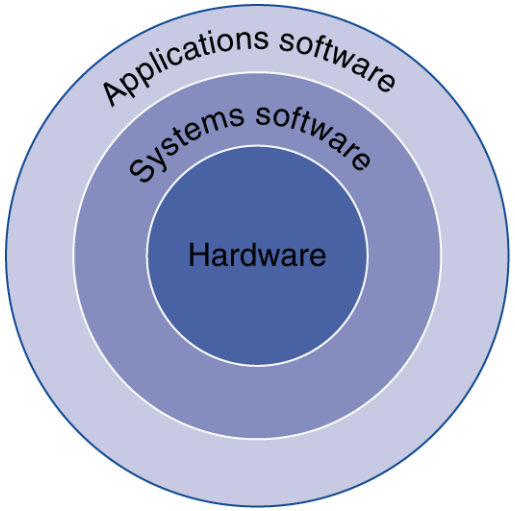
- How programs are translated into the machine language
  - And how the hardware executes them
- The hardware/software interface
- What determines program performance
  - And how it can be improved
- How hardware designers improve performance
- What is parallel processing

Eight Great Ideas

- Design for ***Moore's Law***
- Use ***abstraction*** to simplify design
- Make the ***common case fast***
- Performance *via* ***parallelism***
- Performance *via* ***pipelining***
- Performance *via* ***prediction***
- ***Hierarchy*** of memories
- ***Dependability*** *via* redundancy



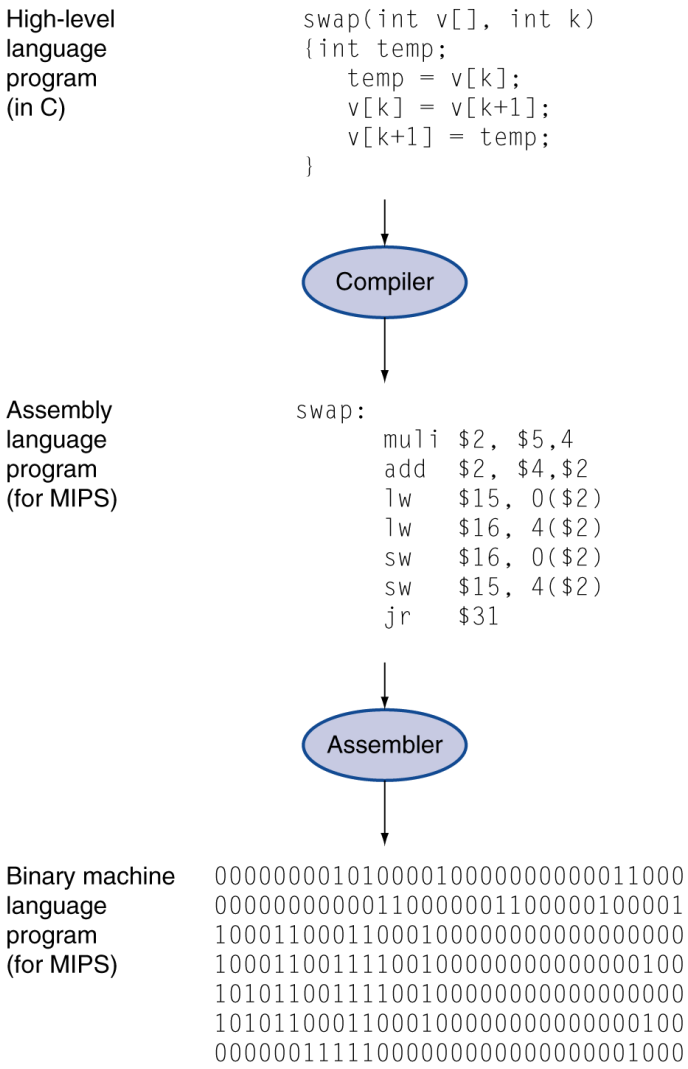
Below Your Program



- Application software
  - Written in high-level language
- System software
  - Compiler: translates HLL code to machine code
  - Operating System: service code
    - Handling input/output
    - Managing memory and storage
    - Scheduling tasks & sharing resources
- Hardware
  - Processor, memory, I/O controllers

Levels of Program Code

- High-level language
  - Level of abstraction closer to problem domain
  - Provides for productivity and portability
- Assembly language
  - Textual representation of instructions
- Hardware representation
  - Binary digits (bits)
  - Encoded instructions and data



Understanding Performance

- Algorithm
  - Determines number of operations executed
- Programming language, compiler, architecture
  - Determine number of machine instructions executed per operation
- Processor and memory system
  - Determine how fast instructions are executed
- I/O system (including OS)
  - Determines how fast I/O operations are executed

For Tuesday

- Read chapter 1.1 – 1.5