Lecture 1

Introduction to Instruction Architectures & Compiler

Prof Peter YK Cheung
Imperial College London

URL: www.ee.ic.ac.uk/pcheung/teaching/EIE2-IAC/
E-mail: p.cheung@imperial.ac.uk

Intended learning outcomes

Theory

- How pipelined CPUs with caches process instructions (Autumn)
- How a compiler turns code into instructions (Autumn)
- The hardware-software interface between compiler and CPUs (A+S)

Practise

- Creating a CPU in System Verilog (Autumn)
- Creating a compiler in C++ (Spring)

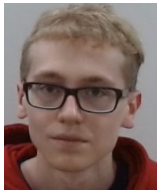
Skills

- Improved knowledge of RTL languages and tools (Autumn)
- Increased proficiency in C++ and software (Spring)

Course Instructors & TAs

Peter Cheung
(Autumn)

John Wickerson
(Spring)



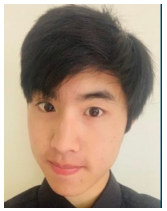
Ryan Voecks
(UTA –EIE3)

Guanxi Lu
(UTA –EIE3)

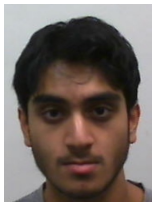
Hrishikesh Venkatesh
(UTA –EIE3)

Haocheng Fan
(UTA –EIE3)

Petr Olsan
(UTA –EIE3)



William Huynh
(UTA –EIE3)



Adam Ali
(UTA –EIE3)

Tianqi Hu
(UTA –EIE3)

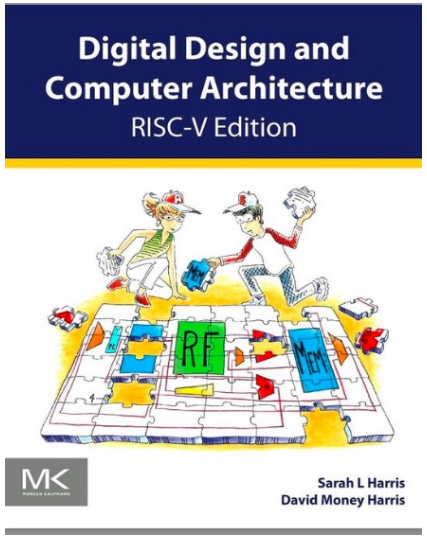
Learning approach for Instruction Architecture (Autumn Term)

- **Lectures: ~ 2 hours per week (Tuesday 4pm – 6pm)**
  - In person, cover theoretical stuff + introduction to Lab/Project
- **Reading: ~ 2-3 hours per week (untimetabled)**
  - Sections to read given in lecture
- **Supervised Lab (2 - 4 hours Thursday and/or Friday)**
  - Pair-based learning with Lab instructions in first half of term
  - Team-based project to design RISC-V processor in 2nd half of term
  - Team working in groups of 4, but individually assessed
  - Lab Sessions also serve as Tutorial Sessions – you can ask staff or UTA questions about course materials
  - Complete your partner declaration survey here:

https://forms.office.com/e/TtBK3asFr4

Autumn : Course Textbook

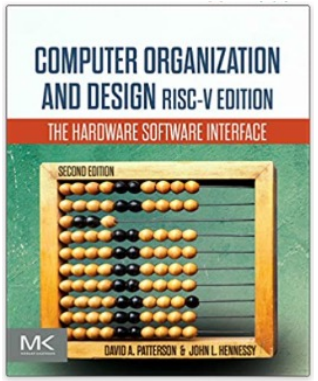
- Digital Design and Computer Architecture (RISC-V Edition) by Sarah Harris and David Harris.



https://www.vlebooks.com/product/openreader?id=IMPERIALBB&accId=8994656&isbn=9780128200650

- Computer Organization and Design RISC-V Edition, Patterson and Hennessy (~£77), electronic copy available at:

https://library-search.imperial.ac.uk/permalink/44IMP_INST/mek6kh/alma991000613172401591



Assessment – Entire module (10 ECTS)

- **The course uses three modes of assessment:**
  - **Labs (20% or 2 ECTS):**
    - Autumn (5%): Mid-term quiz on Lab experiments
    - Spring (5%): Tools for building compilers
  - **Coursework (40% or 4 ECTS):**
    - Autumn (25%): building a working RISC-V processor
    - Spring (25%): building a working C compiler
  - **Final exam (40% or 4 ECTS):**
    - Assessed knowledge of CPUs and compilers
    - Questions cover both topics of architecture and compiler

Labs and coursework for Autumn Term

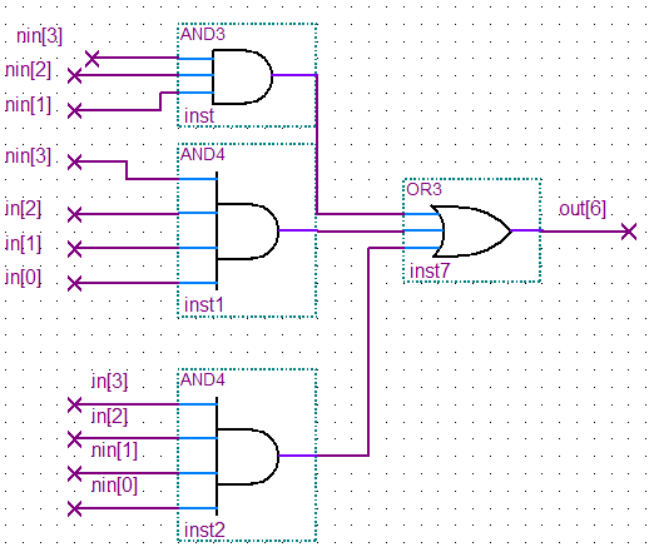
- **First half :**
  - 4 Lab Sessions to teach digital design with SystemVerilog
  - Work in pairs – you choose your own lab partner
  - Expect to keep a logbook on **git**
- **Mid-term : assessment of lab (5%)**
  - Online quiz on Lab 0 to Lab 3 – multiple choice + evidence on git
- **Second half : assessment on project (25%)**
  - Work in teams of 4 from two pairs (I choose)
  - Design a working RISC-V processor in SystemVerilog
  - Four tasks already partitioned – you allocate responsibilities
  - Assessed both as a team and individually (details later)

Overview on

Digital Hardware Design

How to describe/specify digital circuits?

Schematic diagram & gates



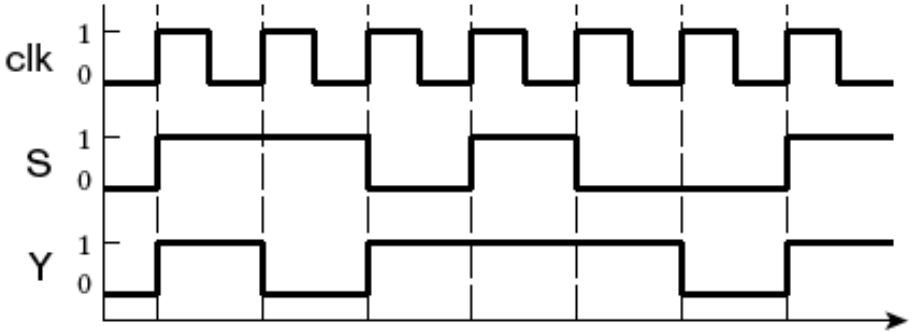
Boolean equation

out6 = /in3*/in2*/in1 + in3*in2*/in1*/in0 + /in3*in2*in1*in0

Truth table

in[3..0]	out[6:0]
0000	1000000
0001	1111001
0010	0100100
0011	0110000
0100	0011001
0101	0010010
0110	0000010
0111	1111000
1000	0000000
1001	0010000

Timing Diagram



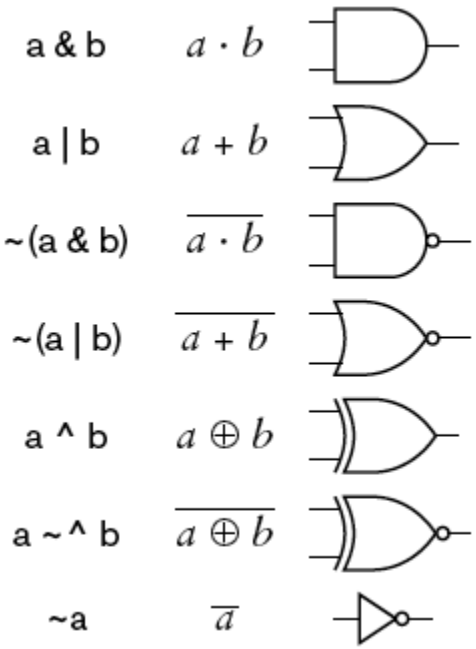
Hardware Description Language (HDL)

3-to-1 MUX

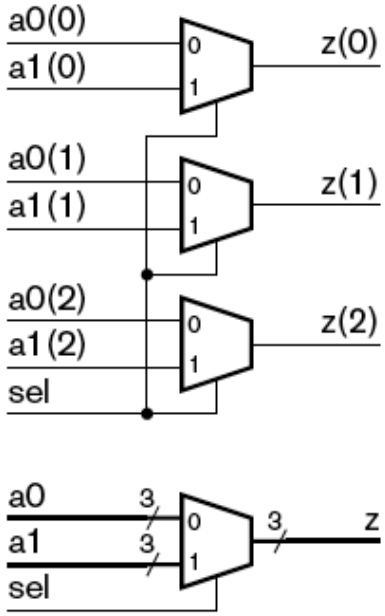
```
module mux32three(i0,i1,i2,sel,out);
input [31:0] i0,i1,i2;
input [1:0] sel;
output [31:0] out;
reg [31:0] out;

always @ (i0 or i1 or i2 or sel)
begin
  case (sel)
    2'b00: out = i0;
    2'b01: out = i1;
    2'b10: out = i2;
    default: out = 32'bx;
  endcase
end
endmodule
```

Basic digital building blocks

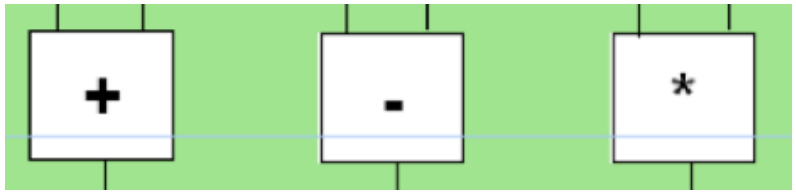
Primitive Logic Gates



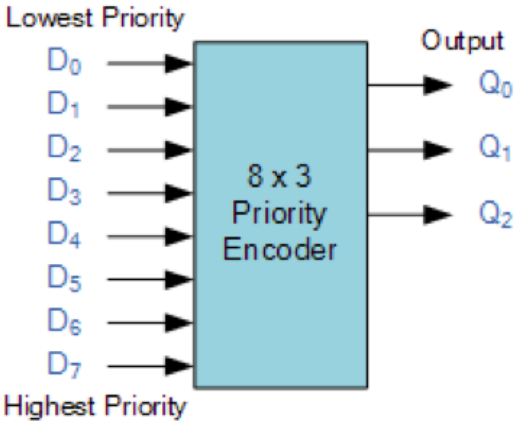
Multiplexers



Arithmetic circuits

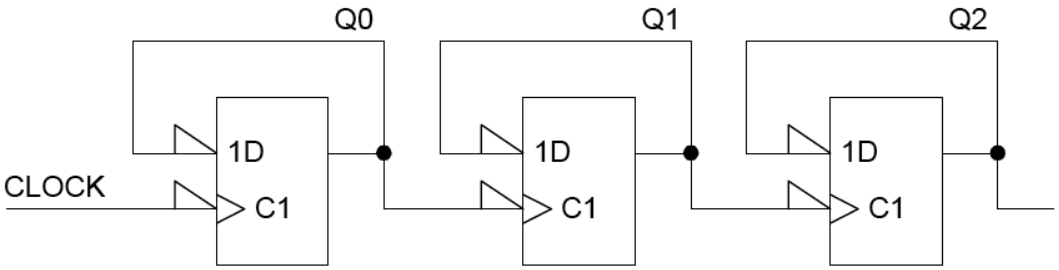


Encoders

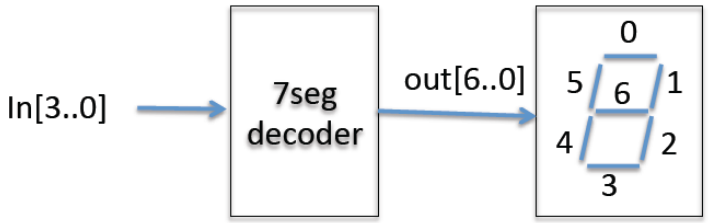


Inputs								Outputs		
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Q_2	Q_1	Q_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	x	0	0	1
0	0	0	0	0	1	x	x	0	1	0
0	0	0	0	1	x	x	x	0	1	1
0	0	0	1	x	x	x	x	1	0	0
0	0	1	x	x	x	x	x	1	0	1
0	1	x	x	x	x	x	x	1	1	0
1	x	x	x	x	x	x	x	1	1	1

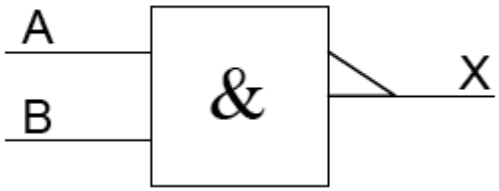
Flipflops and Registers

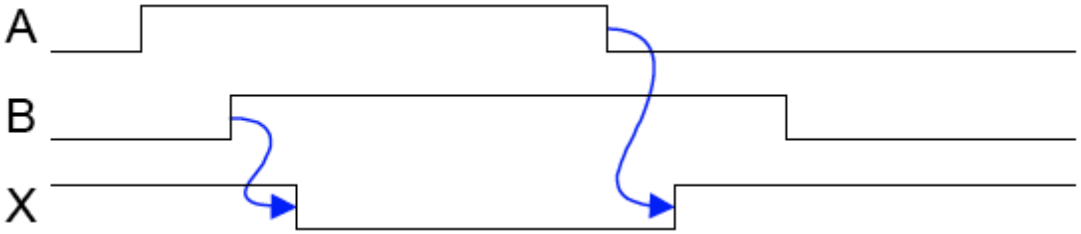


Decoders



Cause & Effect





Input B going high causes X to go low

Input A going low causes X to go high

Propagation Delay:

The time delay between a cause (an input changing) and its effect (an output changing), assuming output load capacitance of 30pF.

Example: 74AC00: Advanced CMOS 2-input NAND gate

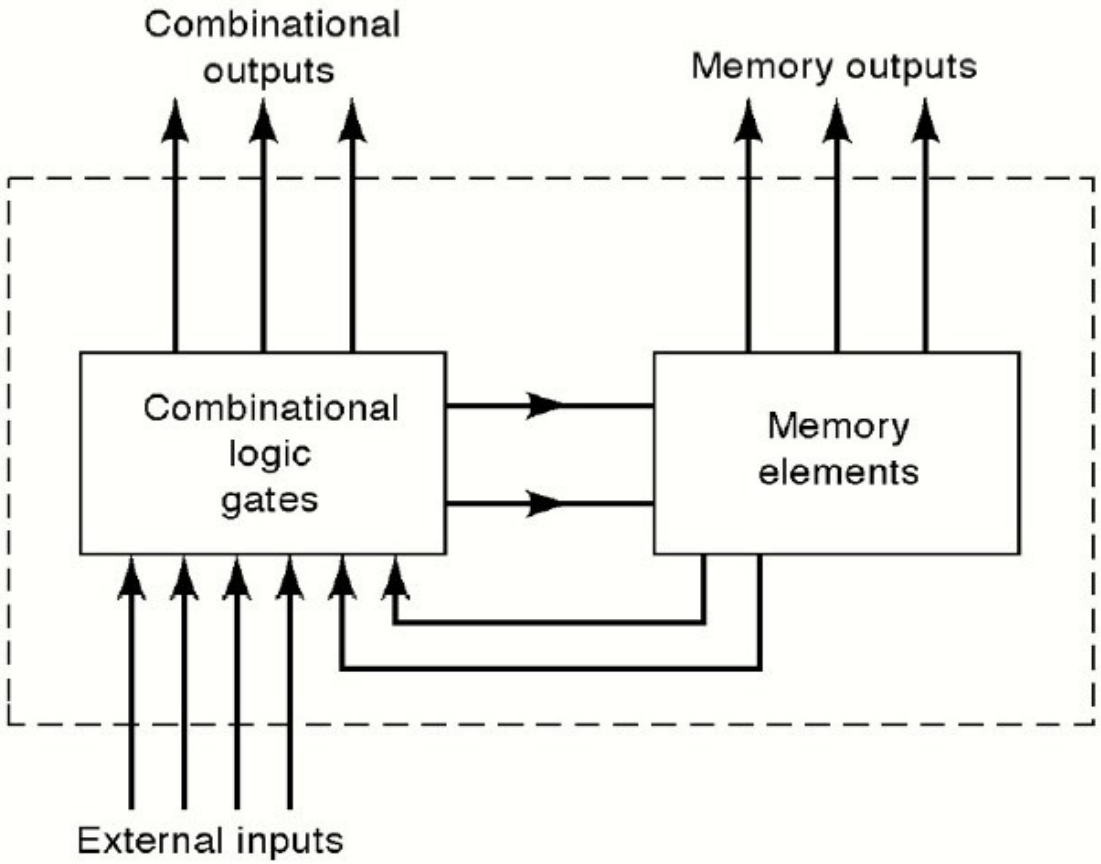
	min	typ	max	
A↑ to X↓ (t_{PHL})	1.5	4.5	6.5	ns
A↓ to X↑ (t_{PLH})	1.5	6.0	8.0	ns

t_{PHL} and t_{PLH} refer to the direction that the output changes: high-to-low or low-to-high.

Combinatorial and sequential logic

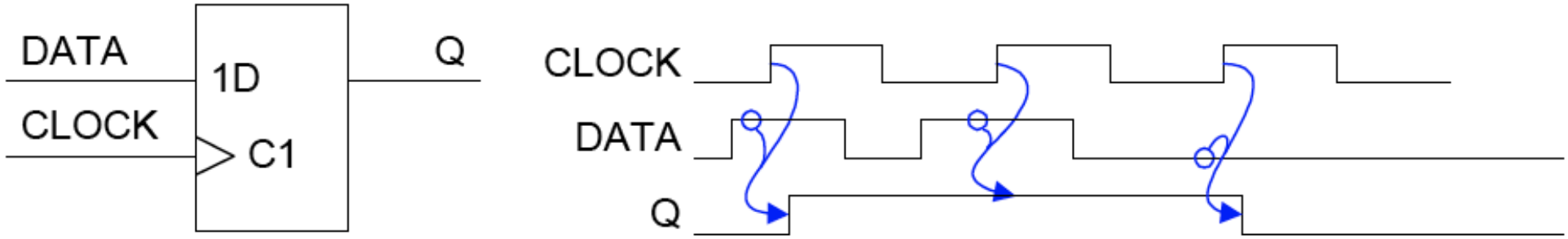
Imperial College London

General digital system diagram



DECA Part 1 9.3 Autumn 2019

D-Flipflop (1)



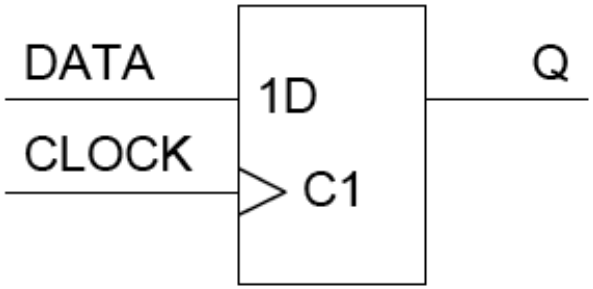
Notation:

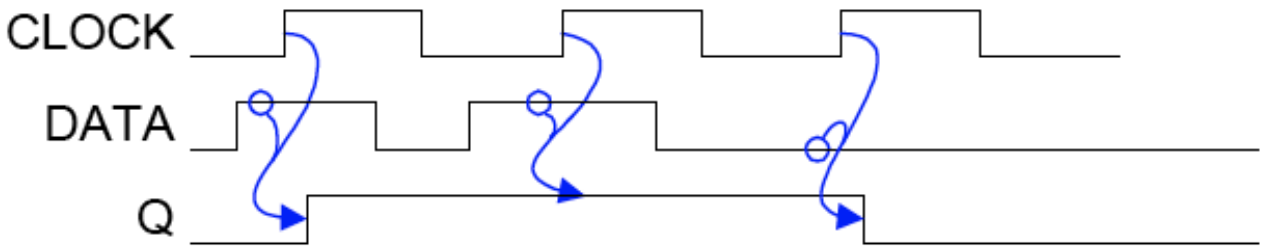
>	input effect happens on the rising edge
C1	C ⇒ Clock input, 1 ⇒ This input *is* input number 1.
1D	D ⇒ Data input, 1 ⇒ This input *is controlled by* input number 1.

The meaning of a number depends on its position:

A number *after* a letter is used to identify a particular input.
A number *before* a letter means that this input is controlled by one of the other inputs.

D-Flipflop (2)



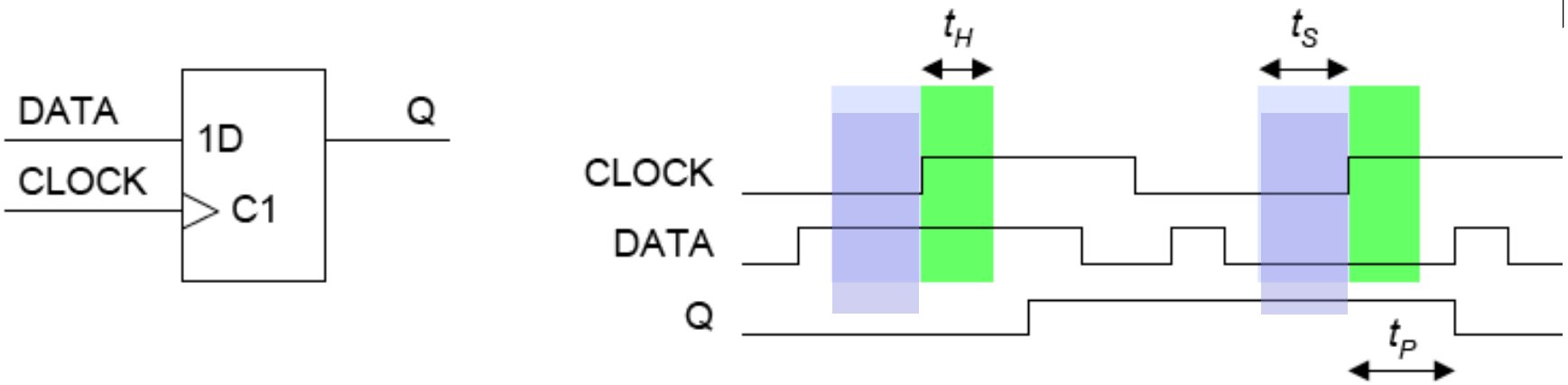


Cause and Effect:

- CLOCK↑ causes Q to change after a short delay. This is the *only* time Q ever changes.
- The value of D *just before* CLOCK↑ is the new Q.
- Propagation delay CLOCK↑ to Q is typically 1 ns.
- Propagation delay DATA to Q **does not make sense** since DATA changing does not cause Q to change.

Setup and Hold Times

The DATA input to a flipflop or register must not change at the same time as the CLOCK.

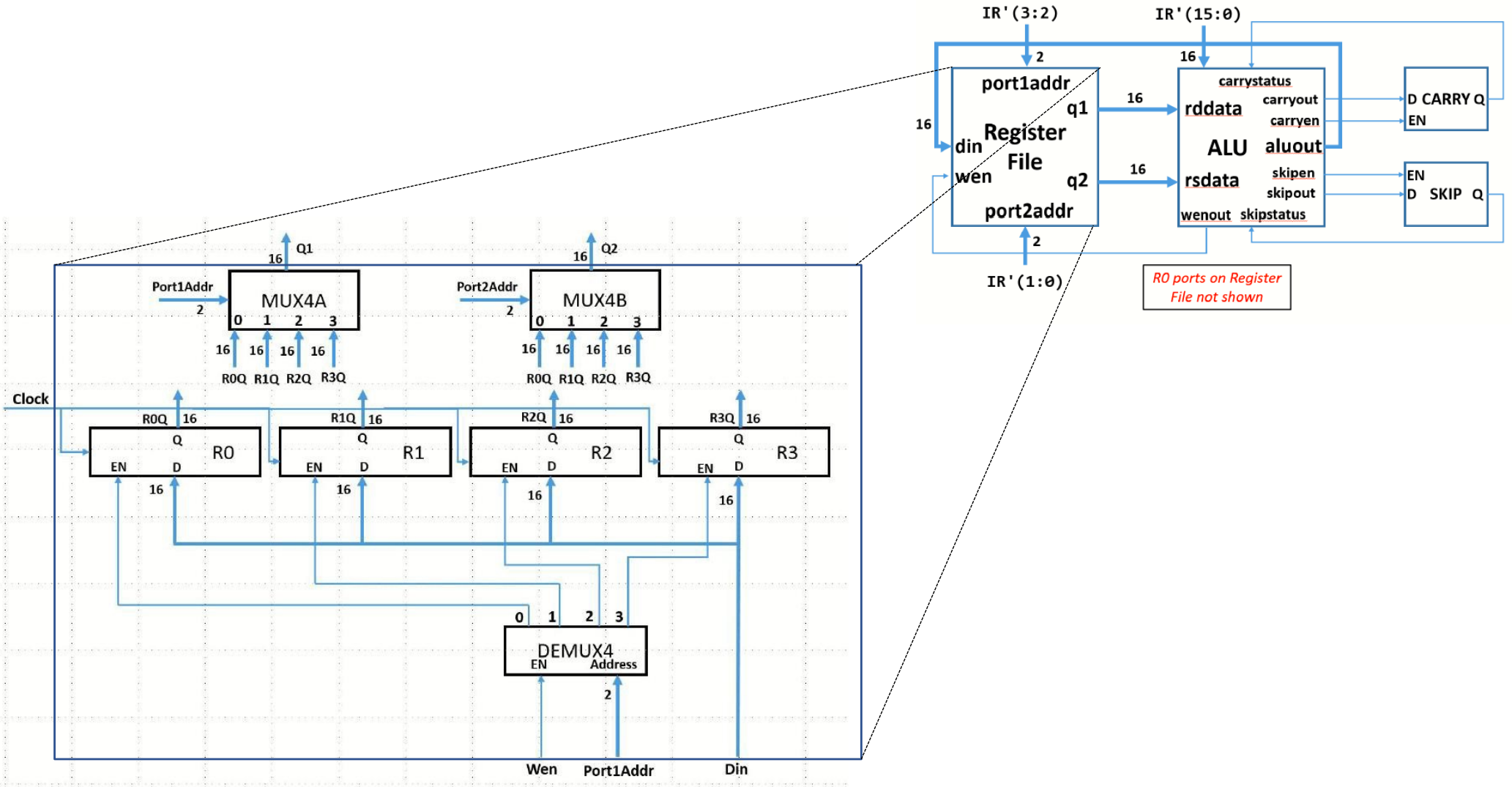


Setup Time: DATA must reach its new value at least t_S before the CLOCK↑ edge.

Hold Time: DATA must be held constant for at least t_H after the CLOCK↑ edge.

- Typical values for a register: t_S = 5 ns, t_H =3 ns (discrete logic/ I/O circuit)
  t_S = -50ps, t_H = 0.2 ns (internal LE)
- The setup and hold times define a window around each CLOCK ↑ edge within which the DATA **must not change**.
- If these requirements are not met, the Q output may oscillate for many nanoseconds before settling to a stable value.

Design Hierarchy



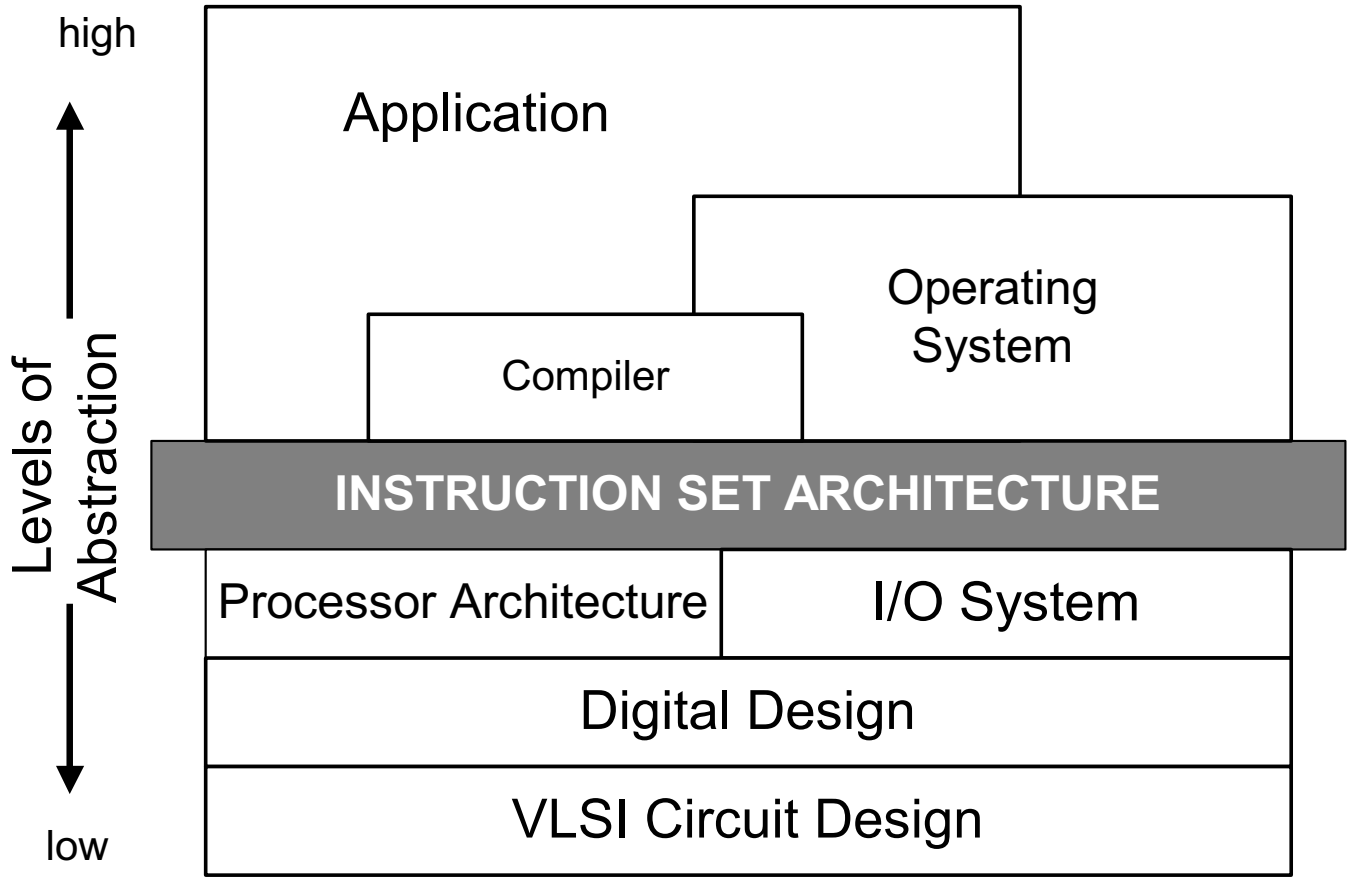
Overview on

Instruction Set Architecture (ISA)

Eight Great Ideas in Computer Architecture

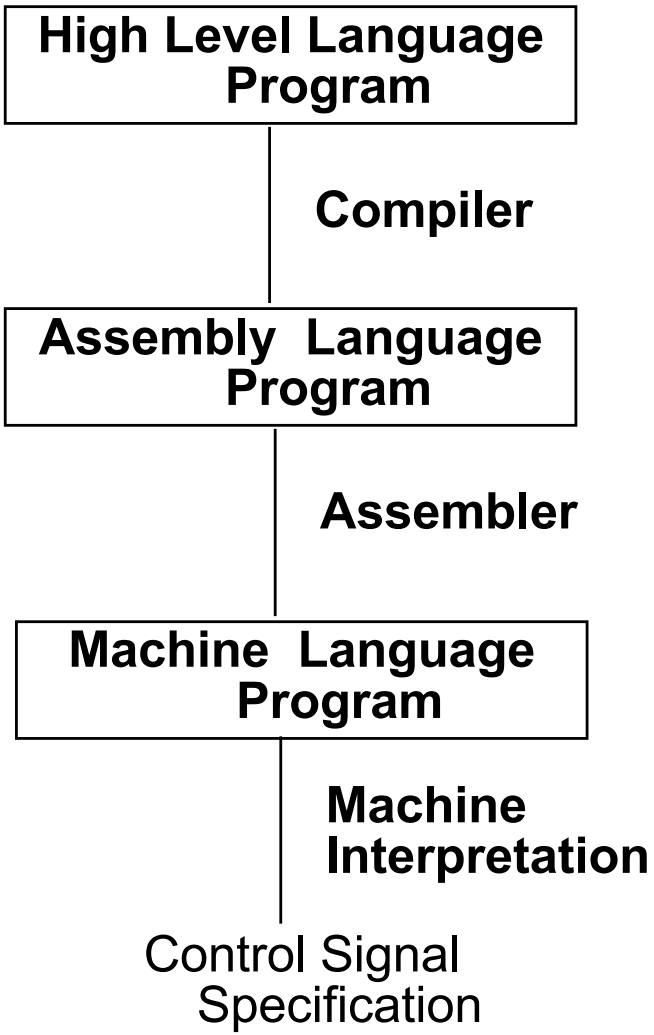
1. Design for Moore's Law
2. Use Abstraction to Simplify Design
3. Make the Common Case Fast (RISC philosophy)
4. Performance via Parallelism
5. Performance via Pipelining
6. Performance via Prediction
7. Hierachy of Memories
8. Dependability via Redundancy

What is “Computer Architecture” ?



- Key: Instruction Set Architecture (ISA)
- Different levels of abstraction

Levels of representation in computers

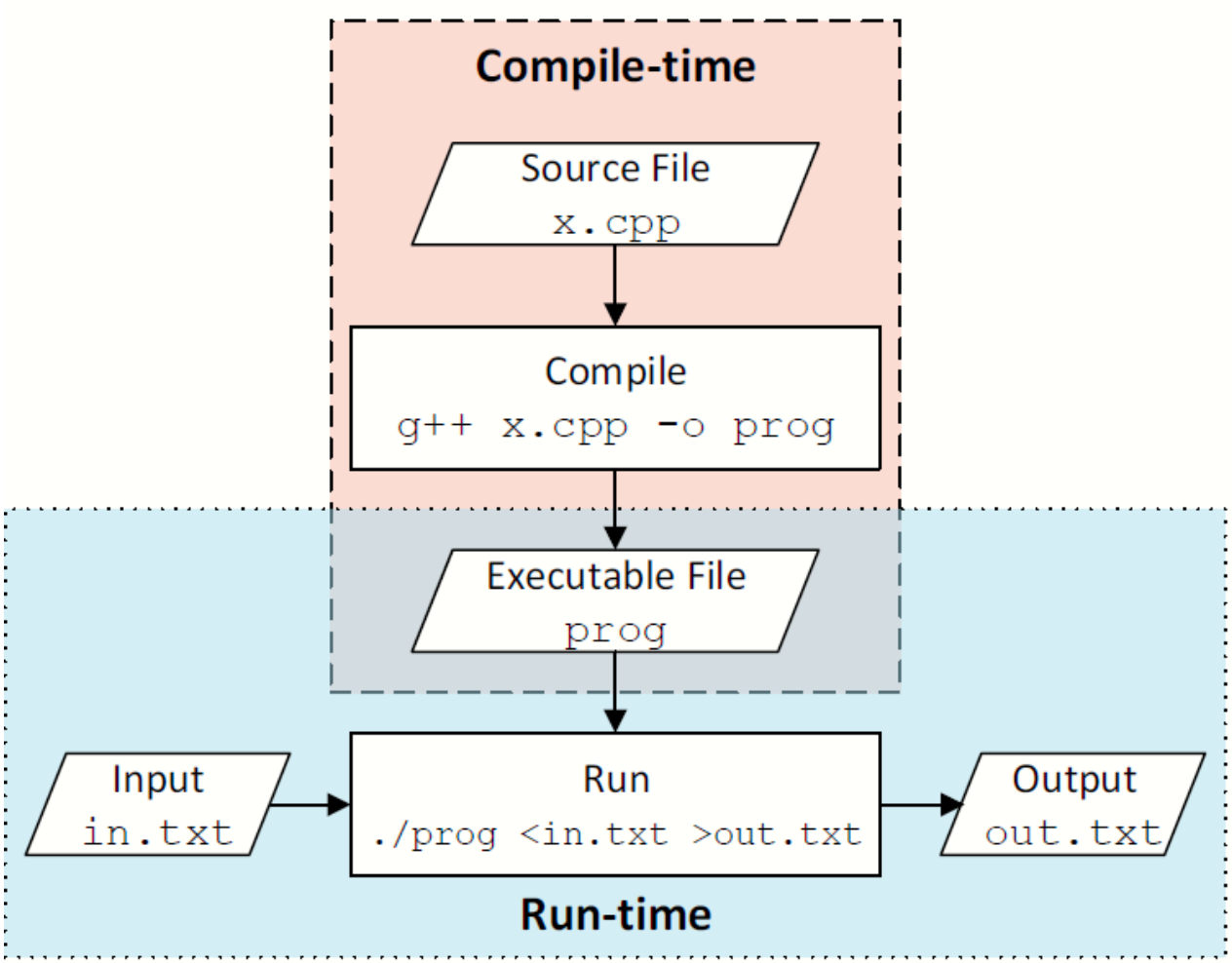



```
temp := v[k];
v[k] := v[k+1];
v[k+1] := temp;
```

```
lw   $15, 0($2)
lw   $16, 4($2)
sw   $16, 0($2)
sw   $15, 4($2)
```

```
0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111
```

Compile-time and Run-time



What is “Instruction Set Architecture (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

ISA includes:-

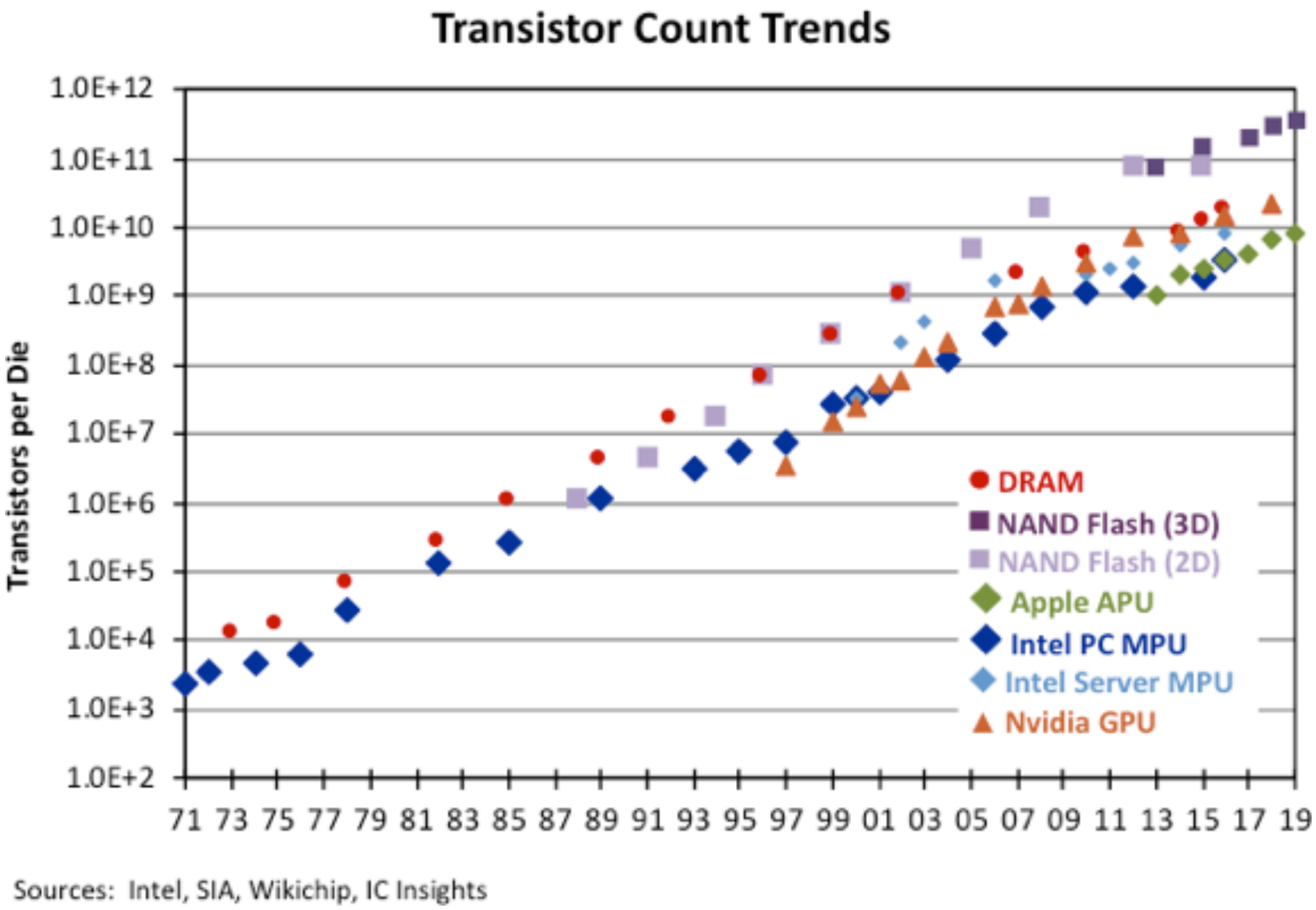
- Organization of Programmable Storage
- Data Types & Data Structures: Encodings & Representations
- Instruction Formats
- Instruction (or Operation Code) Set
- Modes of Addressing and Accessing Data Items and Instructions
- Exceptional Conditions

Instruction Set Architecture (ISA)

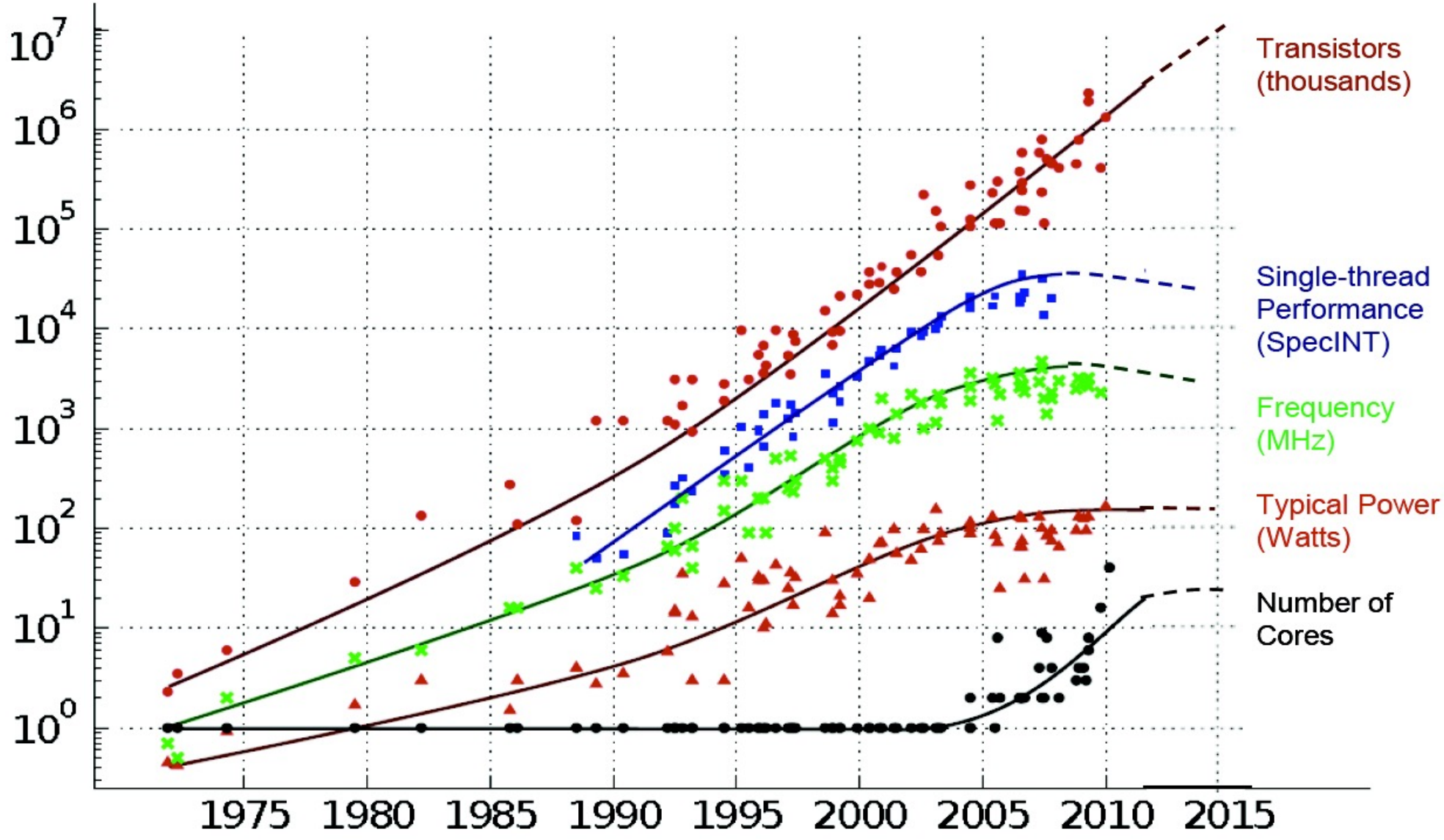
- A very important abstraction
  - interface between hardware and low-level software
  - standardizes instructions, machine language bit patterns, etc.
  - advantage: *different implementations of the same architecture*
  - disadvantage: *sometimes prevents using new innovations*

- Modern instruction set architectures:
  - ARM, Intel x86, **RISC-V**, Xtensa LX6/LX7 (used in ESP32)

Technology: Logic Density (processors)

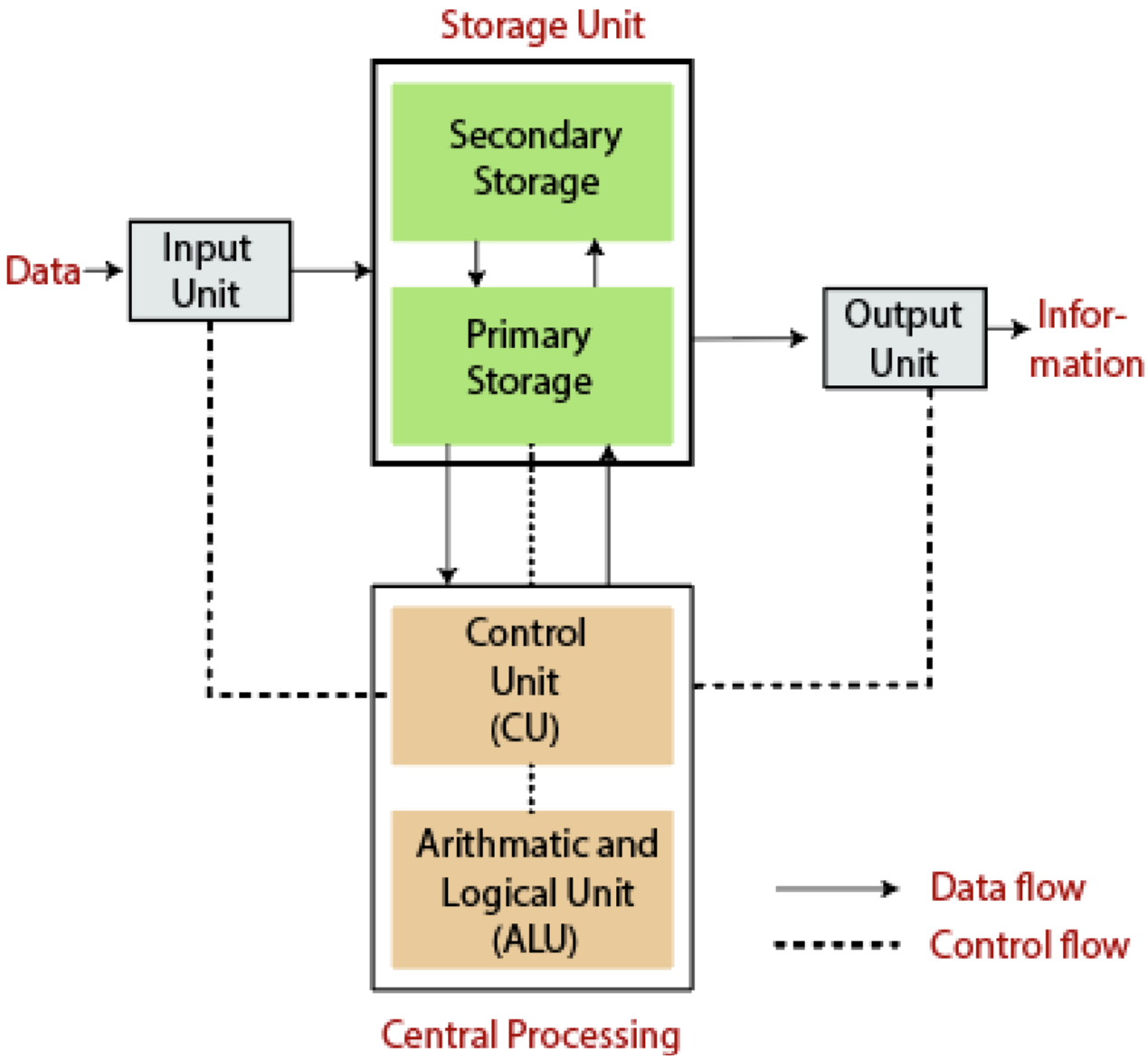


Processor Speed Improvements



Original data collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond and C. Batten

A Typical Computer System with I/O



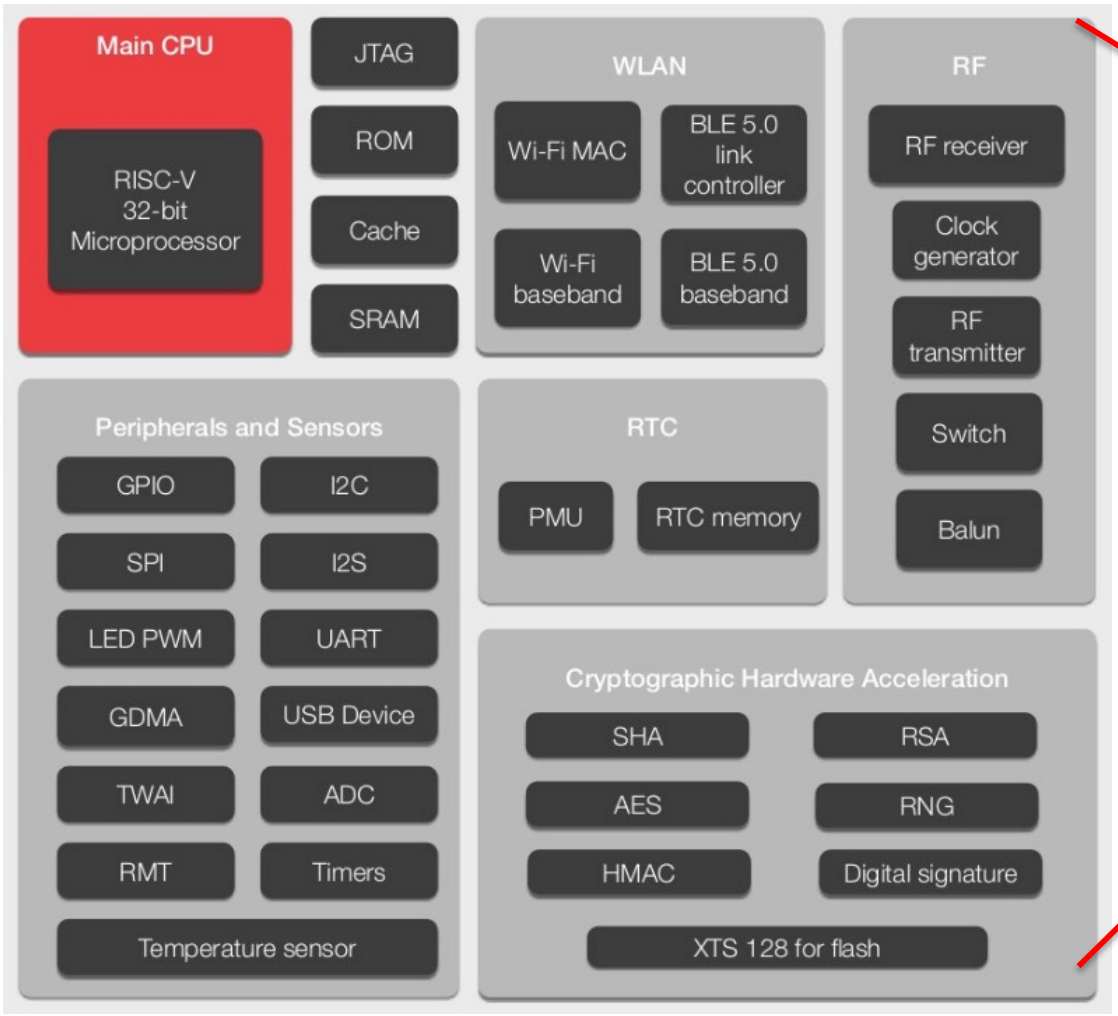
RISC-V Characteristics

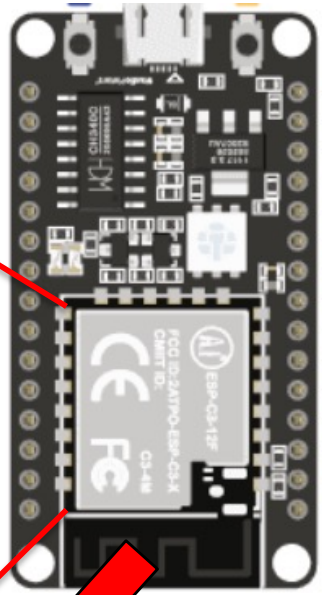
- Emphasis on simplicity and regularity
  - 32-bit instructions
- Smaller is faster
  - Small register file and fewer instructions
- Optimise for common cases
  - e.g. include support for constants
- One ISA family with different variants

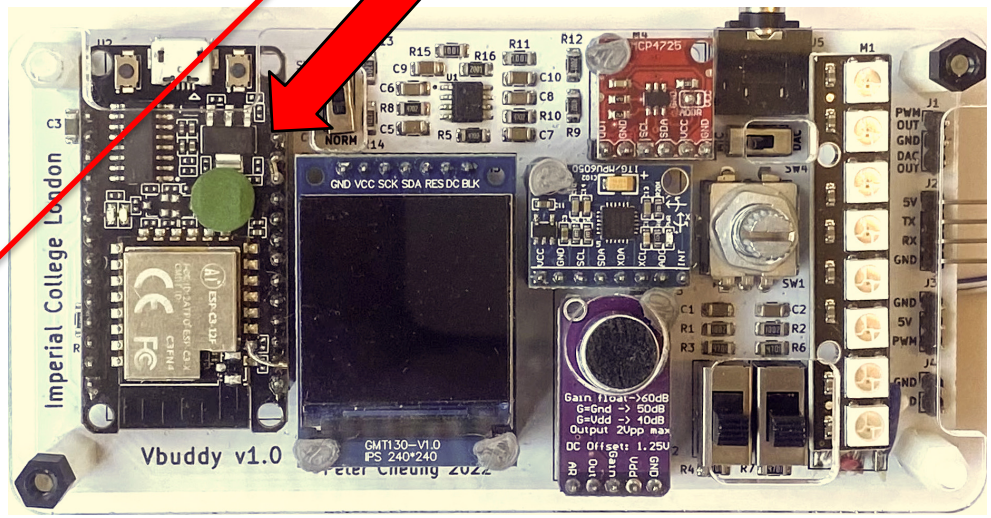
Name	Description	Version	Status	Instruction Count
RV32I	Base Integer Instruction Set - 32-bit	2.1	Frozen	49
RV32E	Base Integer Instruction Set (embedded) - 32-bit, 16 registers	1.9	Open	Same as RV32I
RV64I	Base Integer Instruction Set - 64-bit	2.0	Frozen	14
RV128I	Base Integer Instruction Set - 128-bit	1.7	Open	14

- Open-source (up to a certain extend)

ESP32-C3 Microcontroller – Chip, module, board







Lab 0 – Setting up the Development Environment

- Follow instructions at the following Github page:

https://github.com/EIE2-IAC-Labs/Lab0-devtools.git

- You need to install and learn to use the following tools:
  - **VS Code** – the Integrated Development Environment (IDE)
  - **Verilator** – compile SystemVerilog HDL into C++
  - **RISC-V GNU toolchain** – open-source tools for RISC-V including compiler, assembler, simulator etc.
  - **gtkWave** – view waveforms generator by Verilator model for debugging
  - **Git and Github** – to record your work and your design (for assessment)

- Optional but useful to learn and install:
  - **Linux commands** – only basic ones
  - **Markdown language (MD)** – used with Git, Github
  - **Obsidian** – Cross-platform open-source note taking tool
  - **Make utilities** – used to compile and manage software build
  - **Bash** – basic scripting language all EIE students should know

8 most useful Linux Commands

Command	Example	Description
1. **ls**	ls ls -alF	Lists files in current directory List in long format
2. **cd**	cd tempdir cd .. cd ~dhyatt/web-docs	Change directory to tempdir Move back one directory Move into dhyatt's web-docs directory
3. **mkdir**	mkdir graphics	Make a directory called graphics
4. **rmdir**	rmdir emptydir	Remove directory (must be empty)
5. **cp**	cp file1 web-docs cp file1 file1.bak	Copy file into directory Make backup of file1
6. **rm**	rm file1.bak rm *.tmp	Remove or delete file Remove all file
7. **mv**	mv old.html new.html	Move or rename files
8. **more**	more index.html	Look at file, one page at a time

Basic Markdown Syntax

Element	Markdown Syntax

Heading	`# H1`
	`## H2`
	`### H3`

Bold	`**bold text**`
Italic	`*italicized text*`
Blockquote	`> blockquote`

Ordered List	`1. First item`
	`2. Second item`
	`3. Third item`

Unordered List	`- First item`
	`- Second item`
	`- Third item`

Code	`` `code` ``
Horizontal Rule	`---`
Link	`[title](https://www.example.com)`
Image	`![alt text](image.jpg)`

Basic Markdown Syntax

Element	Markdown Syntax

Heading	`# H1`
	`## H2`
	`### H3`

Bold	`**bold text**`
Italic	`*italicized text*`
Blockquote	`> blockquote`

Ordered List	`1. First item`
	`2. Second item`
	`3. Third item`

Unordered List	`- First item`
	`- Second item`
	`- Third item`

Code	`` `code` ``
Horizontal Rule	`---`
Link	`[title](https://www.example.com)`
Image	`![alt text](image.jpg)`

Obsidian – Best note taking app?