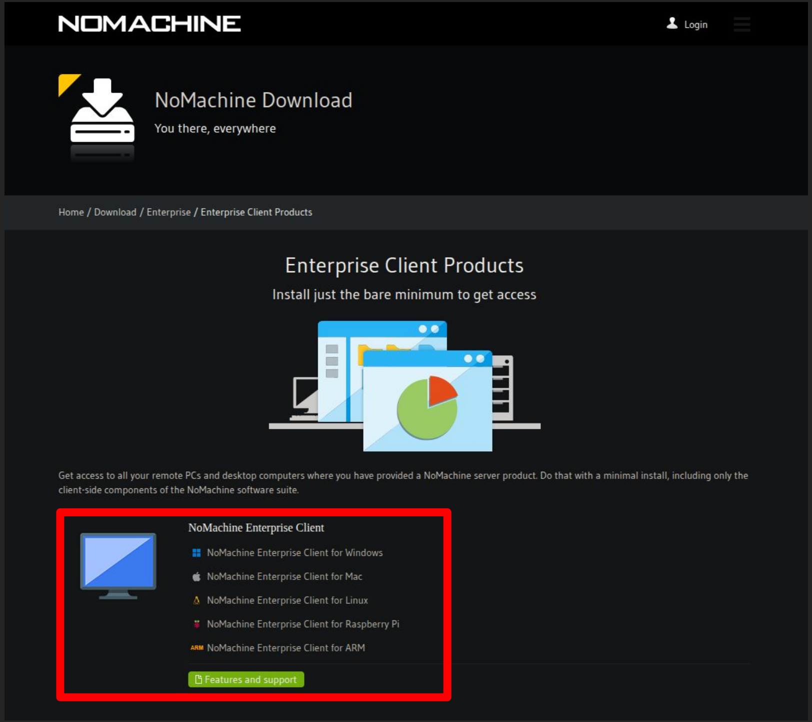
Required Software

Download **NoMachine Enterprise Client** on your laptop



Design Your Own CPU

Session 2 – The single-cycle CPU

Session 1 – Intro to SystemVerilog

Session 2 – The single-cycle CPU

Session 3 – Optimisation and bare-metal programming

About Me

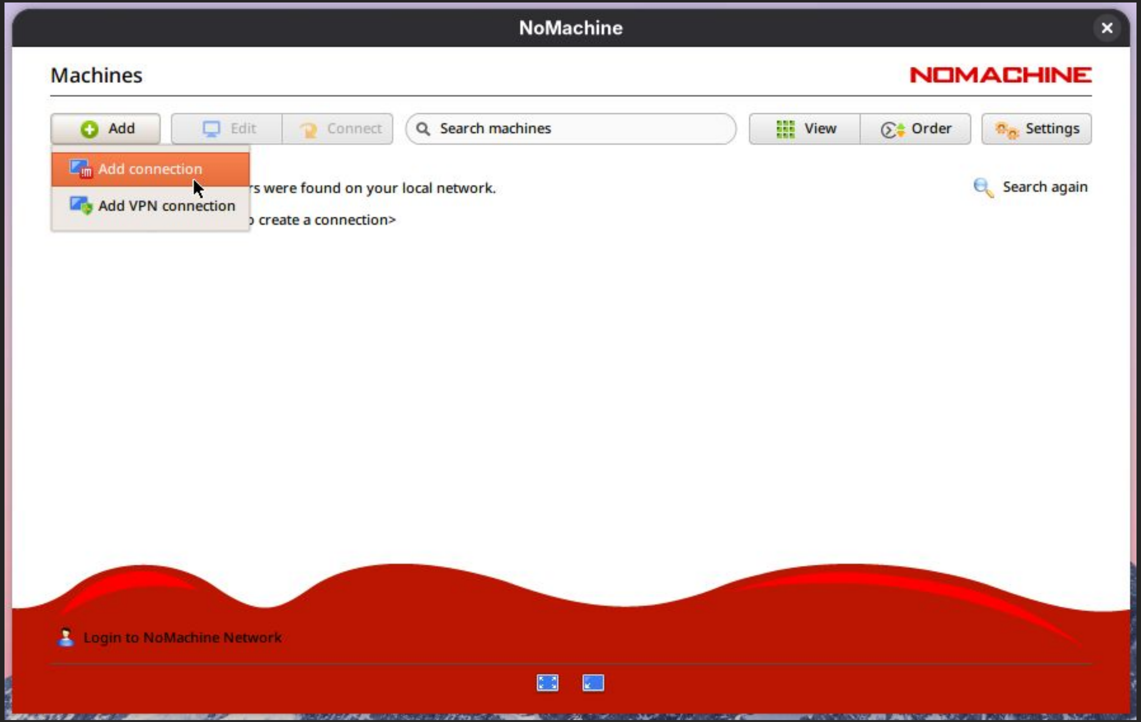
- Part IV Electronic Engineering with Computer Systems
- IP: *A Universal Superscalar Processor Design for MIPS and RISC-V*
- GDP: *Verify a RISC-V Core in 10 Weeks*
- R&D Intern at Codasip (Fabless RISC-V IP firm)

Reasons for Attending These Workshops

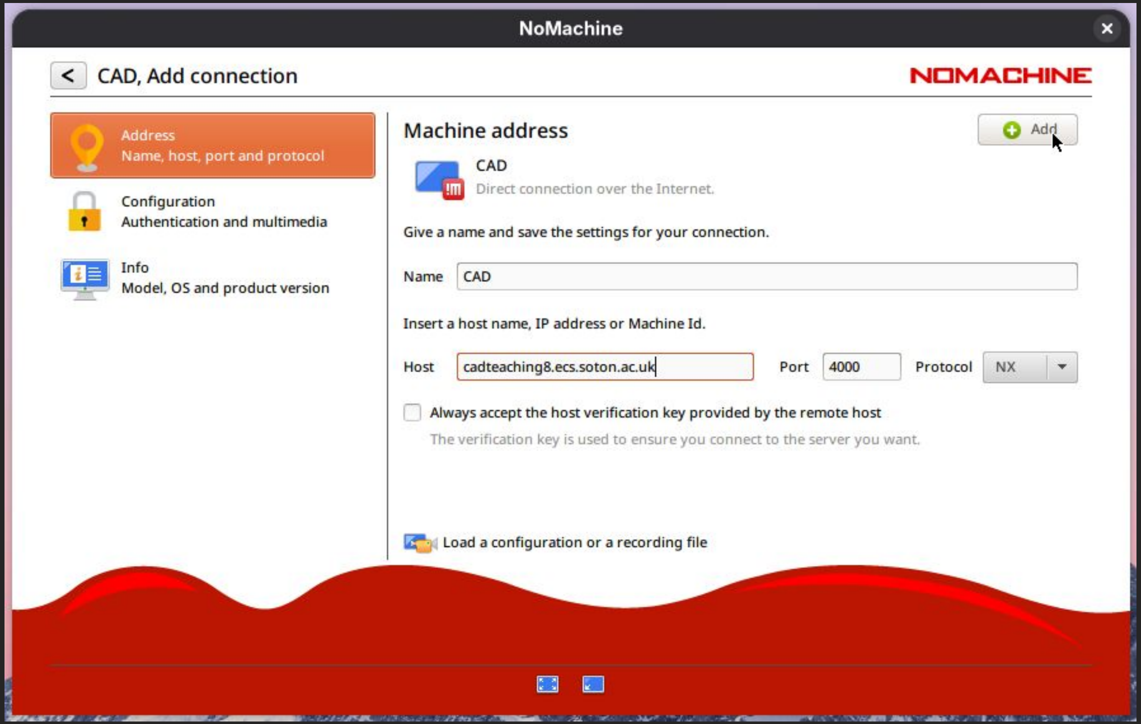
- You’re interested in doing a digital design focused IP or GDP
- You want an internship in digital design
- You want to learn or have a refresher on SystemVerilog
- You want to know how digital systems are designed
- For fun

Setup

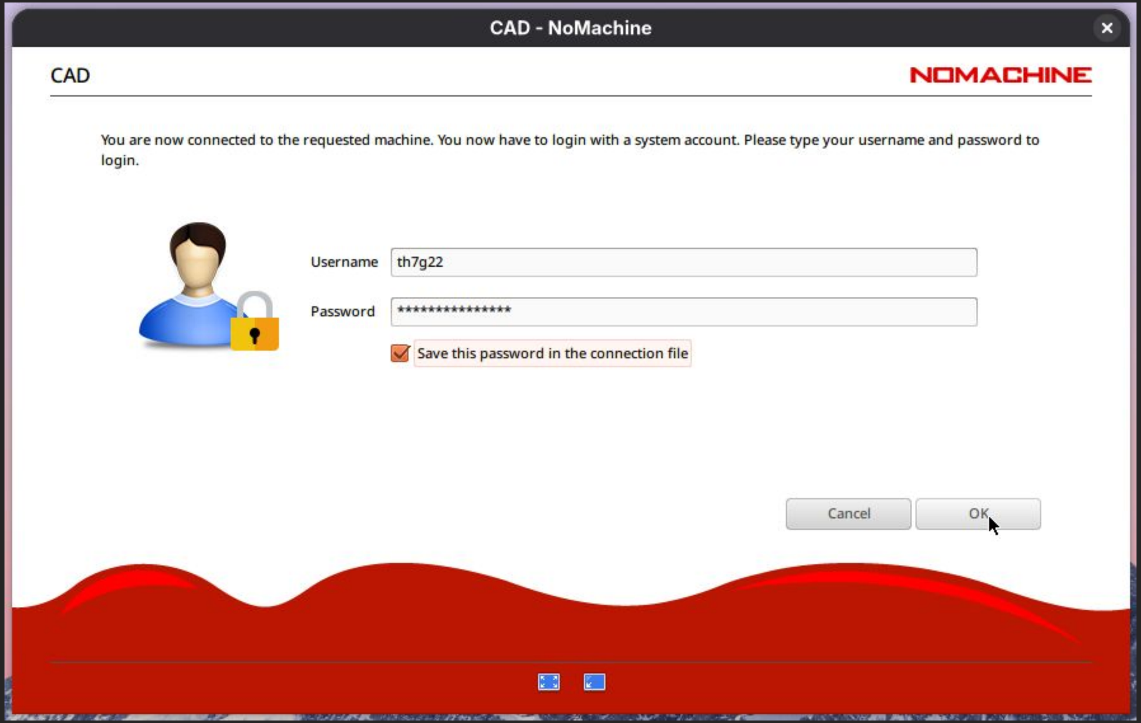
Setup (1)


NoMachine
Machines
NOMACHINE
Add
Edit
Connect
Search machines
View
Order
Settings
Add connection
Add VPN connection
rs were found on your local network.
create a connection>
Search again
Login to NoMachine Network

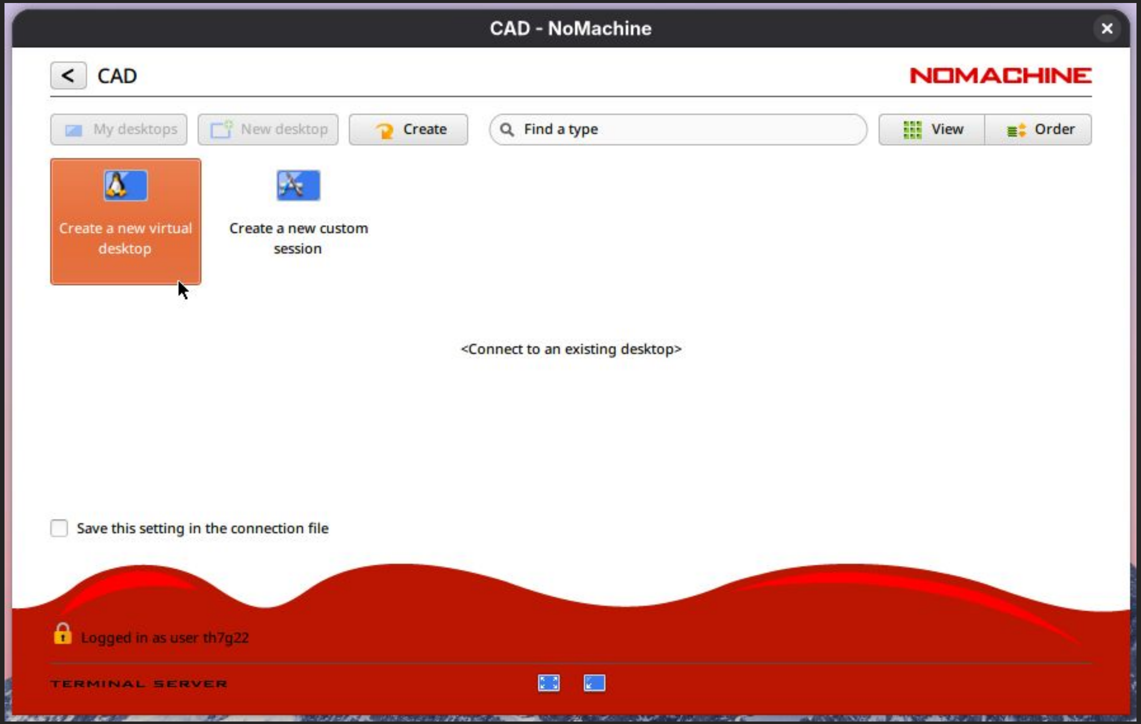
Setup (2)


NoMachine
CAD, Add connection
NOMACHINE
Address
Name, host, port and protocol
Configuration
Authentication and multimedia
Info
Model, OS and product version
Machine address
Add
CAD
Direct connection over the Internet.
Give a name and save the settings for your connection.
Name
CAD
Insert a host name, IP address or Machine Id.
Host
cadteaching8.ecs.soton.ac.uk
Port
4000
Protocol
NX
Always accept the host verification key provided by the remote host
The verification key is used to ensure you connect to the server you want.
Load a configuration or a recording file

Setup (3)


CAD - NoMachine
CAD
NOMACHINE
You are now connected to the requested machine. You now have to login with a system account. Please type your username and password to login.
Username
th7g22
Password
****
Save this password in the connection file
Cancel
OK

Setup (4)


CAD - NoMachine
CAD
NOMACHINE
My desktops
New desktop
Create
Find a type
View
Order
Create a new virtual desktop
Create a new custom session
<Connect to an existing desktop>
Save this setting in the connection file
Logged in as user th7g22
TERMINAL SERVER

Setup (5)

- **If you were here last week:**
  - **Go to the folder we were working in**
  - **`git pull`**
- *Otherwise*, in your terminal, clone this Git repository:

https://github.com/bhart17/design-your-own-cpu

- `git clone https://github.com/bhart17/design-your-own-cpu.git`
- `cd design-your-own-cpu`

Recap

Overall: Designing digital systems

- We want to model systems that take in a number of binary inputs
- They compute outputs as a function of these inputs **and** the state of the system using boolean logic
- State is updated on a clock edge

Levels of Abstraction for a Digital IC

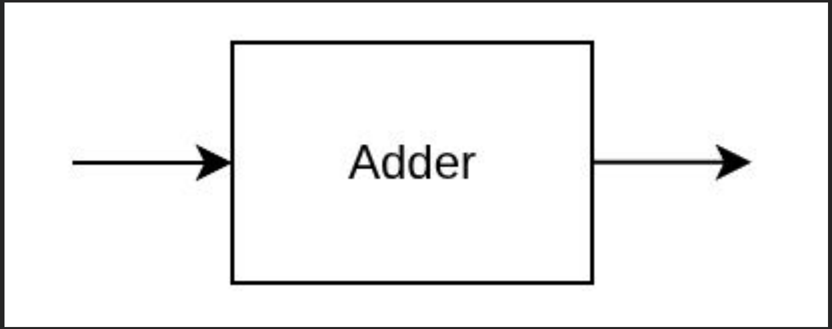
1. Specification
2. Block diagram
3. Hardware description language (such as SystemVerilog)
4. Gate-level circuit *(Generic cells)*
5. Transistor-level circuit *(Specific cells)*
    5.1. Transistor circuit diagram
    5.2. Stick diagram
    5.3. Mask diagram

Hardware Description Languages

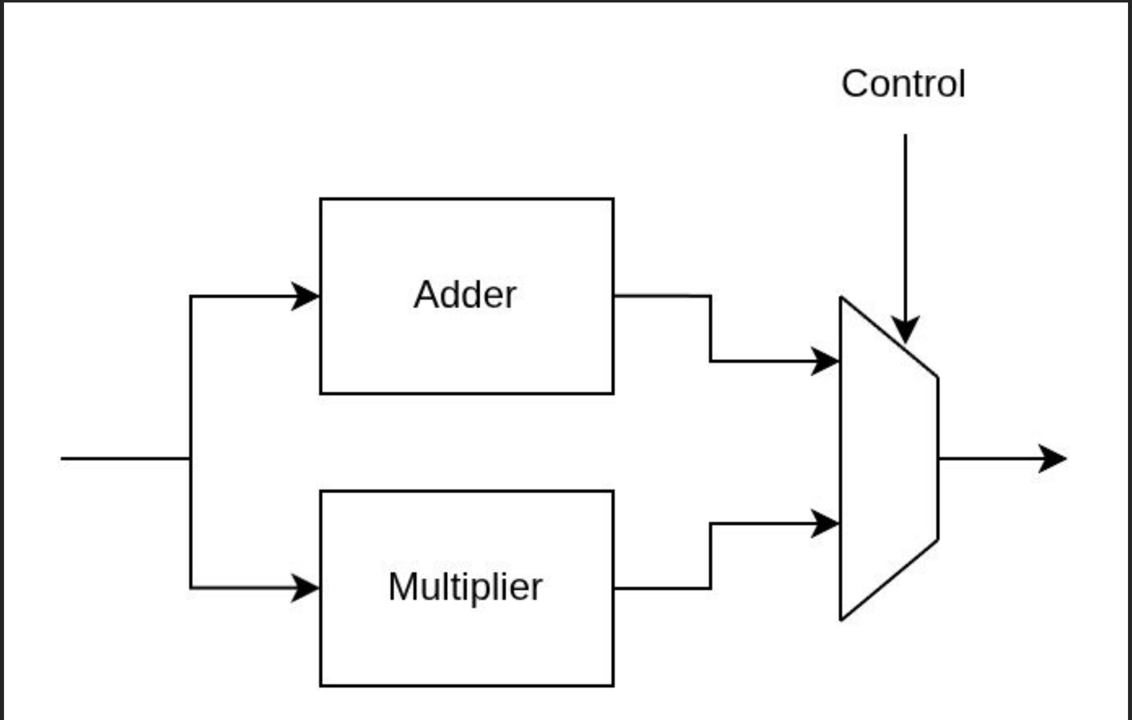
- Aiming to describe the logic between an input and output in a formal but high-level way
- Readable by humans but understandable to computers
- Historically called Register Transfer Layer – describing how data flows between *registers*
- Not programming languages!

CPUs

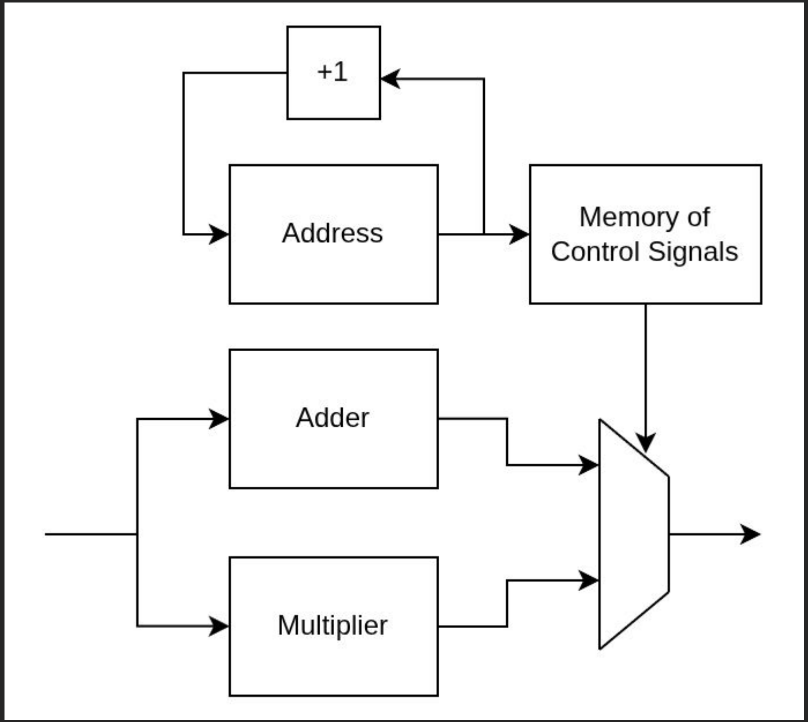
Specific Circuits



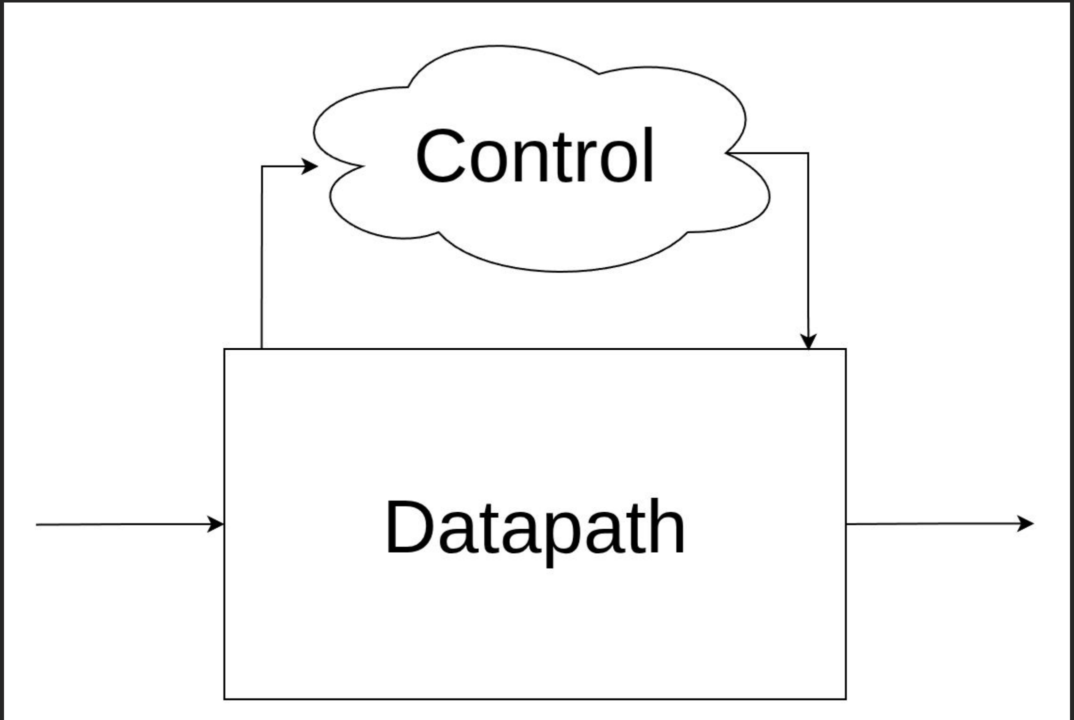
Specific Circuits with Control Signals



CPUs (1)



CPUs (2)



RISC-V

- Open-source *Instruction Set Architecture*
- Defines the set of functions that our CPU needs to perform and how those instructions are encoded
- We have freedom to implement those instructions in whatever architecture we want

Assembly and Machine Code

- We can write a program in assembly language
- This is just a human readable version of the instruction that the CPU reads – a 1-to-1 mapping
- For example

```
add s1, s2, s3
```

Add the values in registers s2 and s3 and put the result in s1

RV32I Instructions

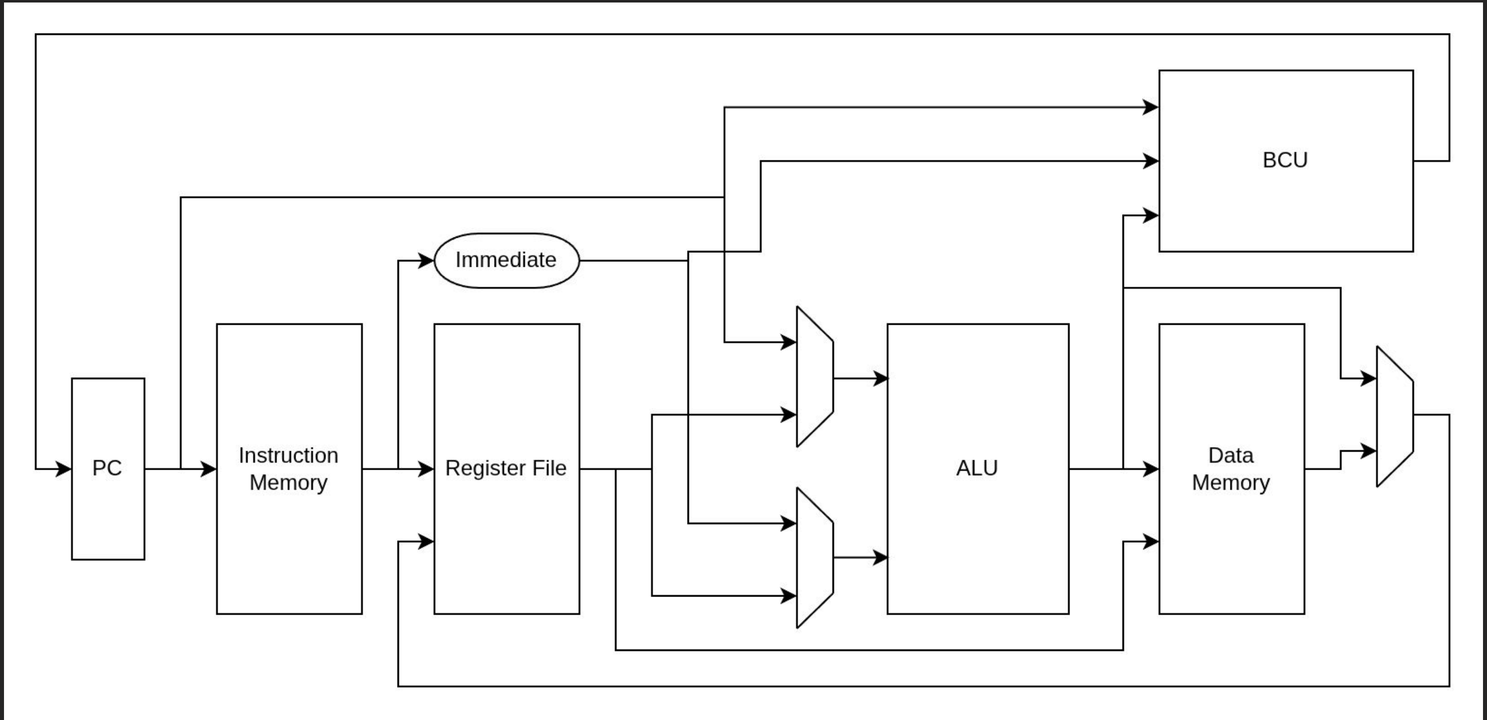
- Arithmetic Instructions
  - ADD, SUB, AND…
- Load and Store Instructions
  - LW, SW…
- Control Flow Instructions
  - JAL, BEQ…

Memory Mapped Peripherals

- CPUs need to interact with inputs and outputs
- Integrating specific I/O into every CPU is not trivial
- Solution: access all data via the data memory interface
- Some sections of memory are not ROM or RAM – they are any generic I/O, *mapped* as a memory address

Our Design

Datapath


BCU
Immediate
PC
Instruction Memory
Register File
ALU
Data Memory

Memory Map

- ROM: 0x00000000 to 0x3FFFFFFF, size 4K
- RAM: 0x40000000 to 0x7FFFFFFF, size 4K
- *Unused: 0x80000000 to 0xBFFFFFFF*
- Virtual Console: 0xC000000 to 0xFFFFFFF
- All memory devices contain virtual copies to fill their space

Practical

1. Program Counter

- The Program Counter stores the current instruction address
- Fix it so that it correctly updates to the next PC on a clock edge

2. Register File

- The Register File stores the 32 general purpose registers
- Implement the reading and writing

3. ALU

- The Arithmetic and Logic Unit does all* of the calculations in our CPU
- An alu_op input defines what operation the ALU should do
- Implement the correct calculation based on the operation

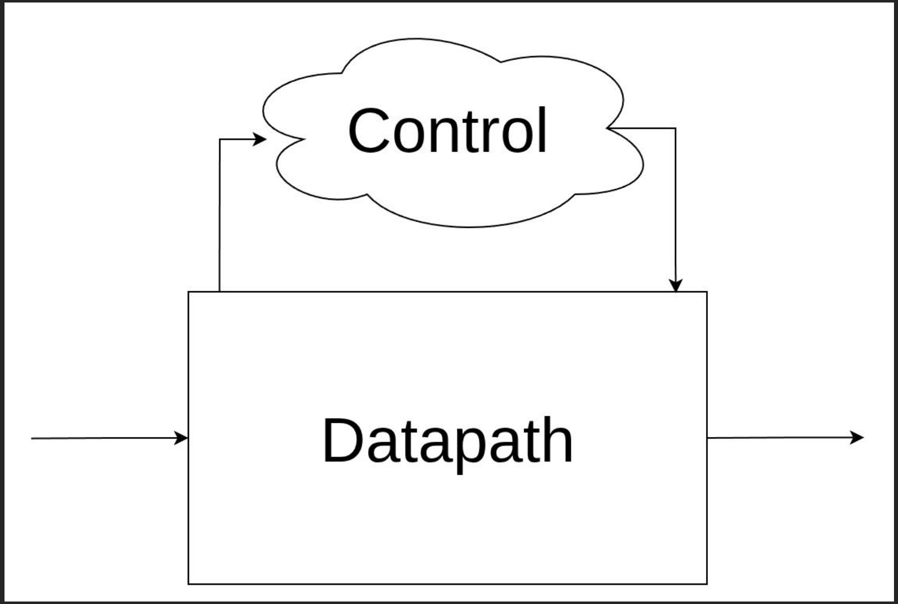
* The BCU also does branch related additions

4. BCU

- The Branch Control Unit calculates the next PC
- A branch_op input defines what criteria we have to branch or not branch
- Implement the branch logic based on the operation

5. Control

- Implement a couple of the control functions that our datapath uses
- *I've provided most of them*



Next Week

- Optimise our CPU using pipelining
- Compile C code to create your own Hello World!
- Learn about cutting edge CPU design methods

Some materials and inspiration in this presentation sourced from Iain McNally's notes (https://www.southampton.ac.uk/~bim/)