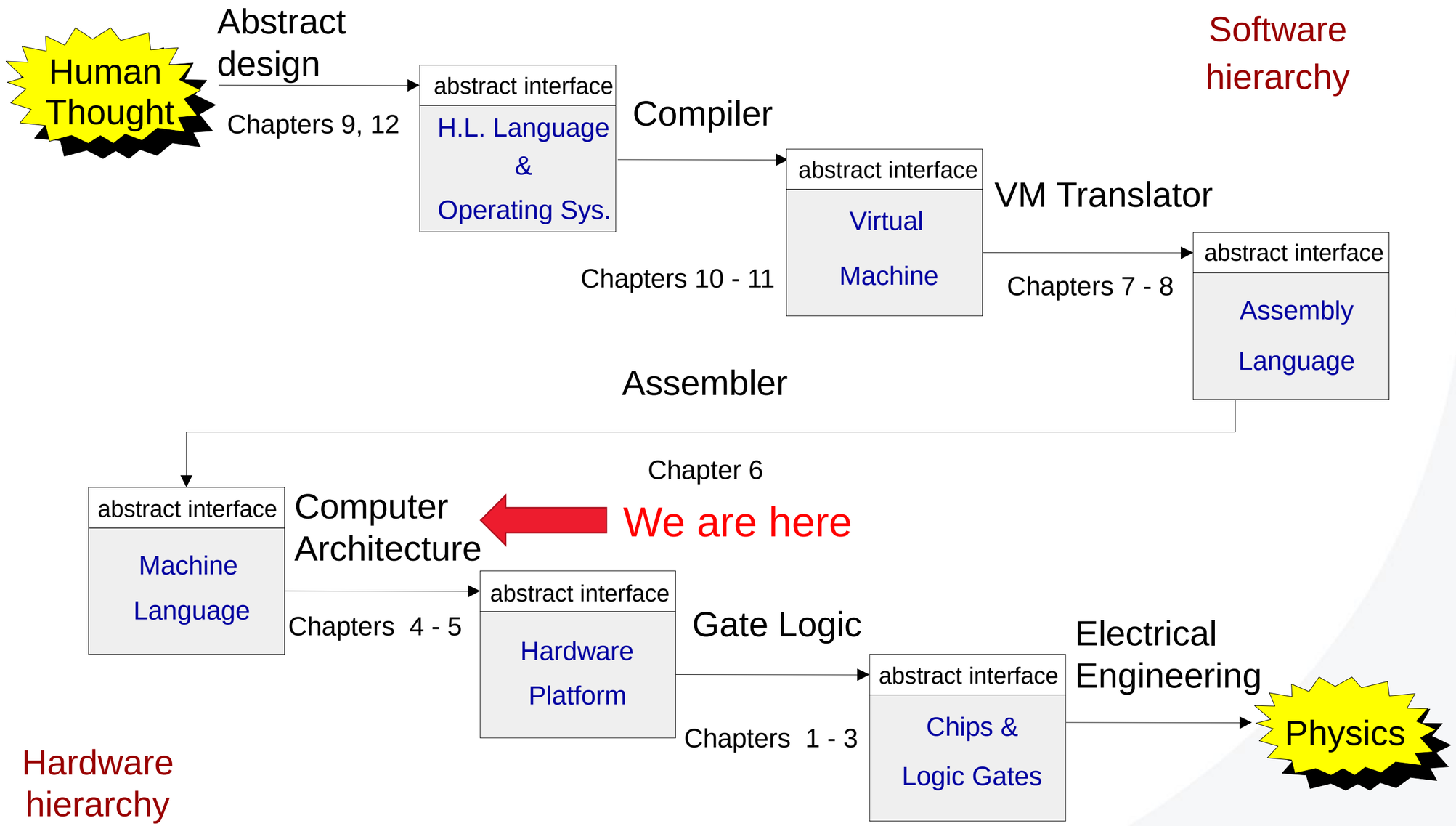
THE UNIVERSITY of ADELAIDE

Computer Systems

Lecture 05: Computer Architecture

Memory & Arithmetic Logic Unit

Our Journey



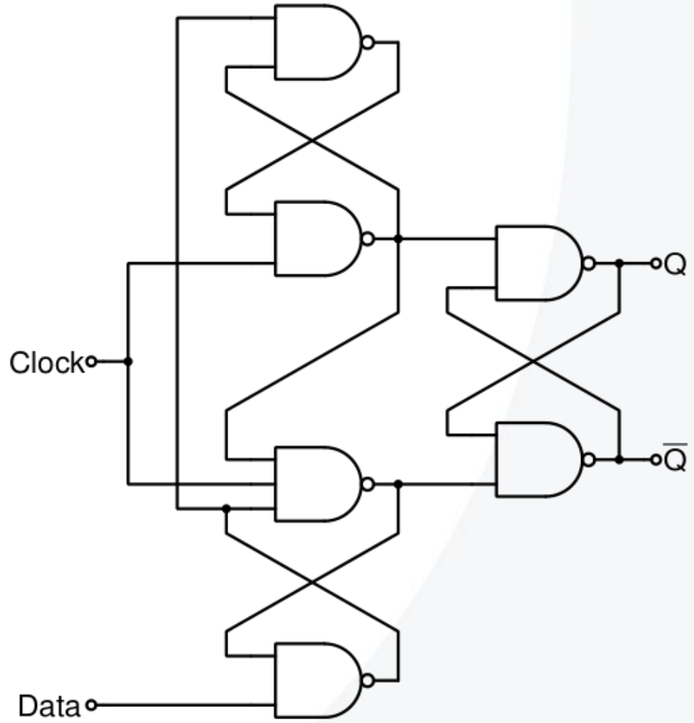
(Abstraction–implementation paradigm)

THE UNIVERSITY of ADELAIDE

Review: Data Flip Flop (DFF)

- Holds the input value for 1 clock cycle
- Can be used to build a Register, then RAM

out(t) = in(t-1)

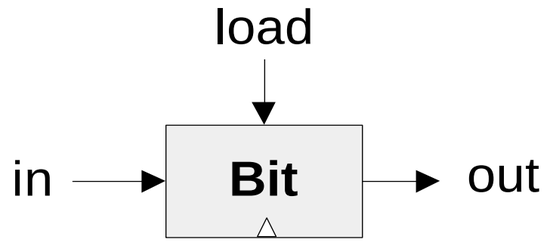


Review: 1-bit register

We want to be able to:

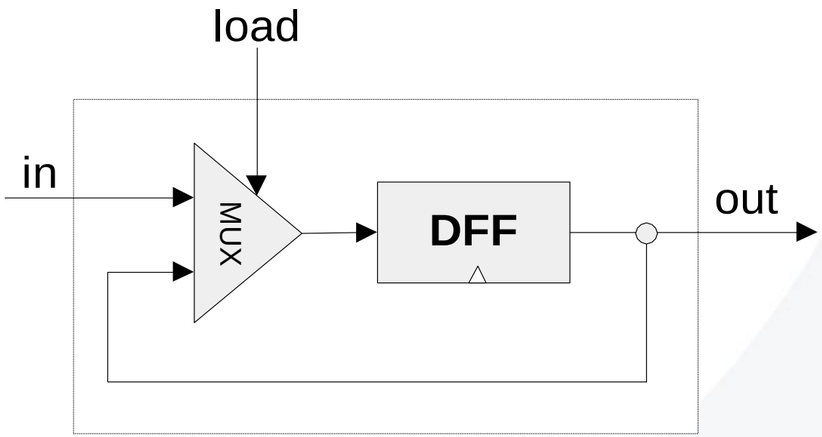
Change the state of the bit

Retain that state until we want to change it.



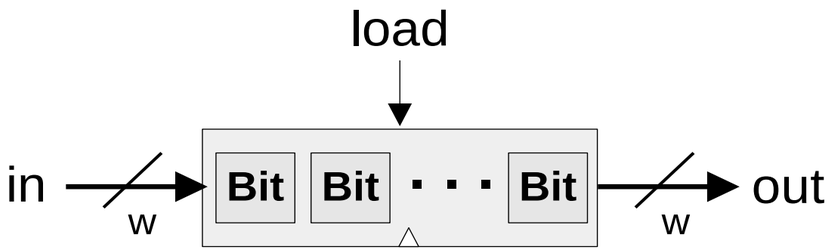
if load(t-1) then out(t)=in(t-1)
else out(t)=out(t-1)

Implementation



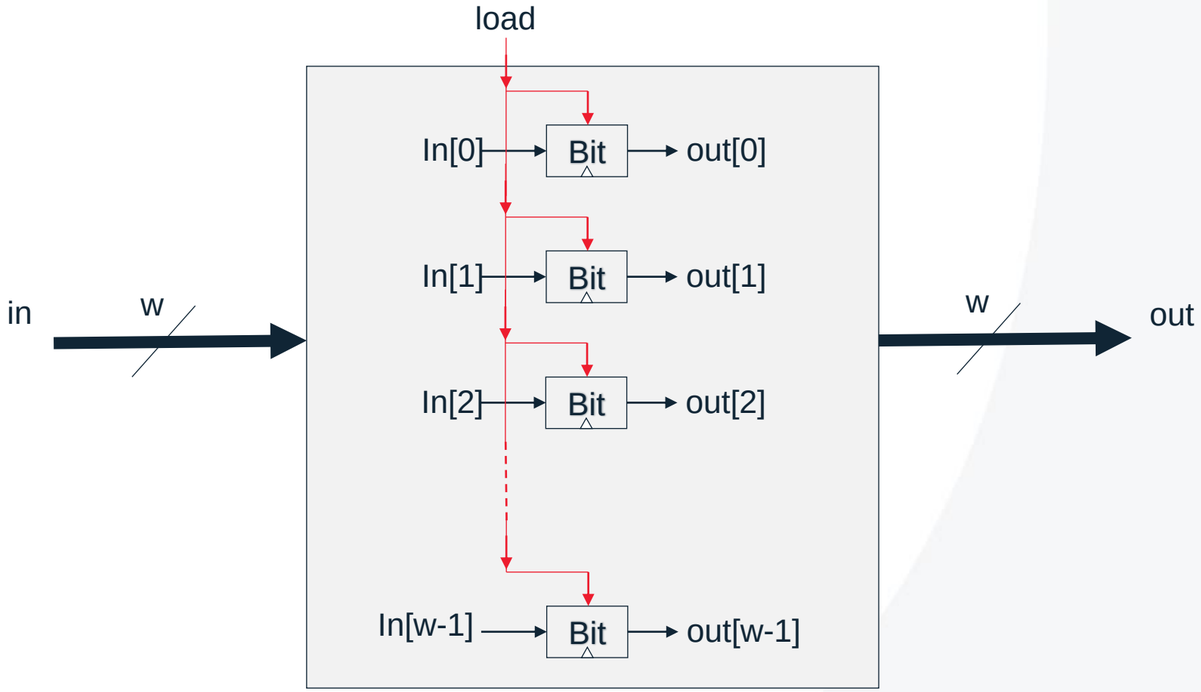
Review: Multi-bit registers

We know we can work with more than one bit.



if load(t-1) then out(t)=in(t-1)
else out(t)=out(t-1)

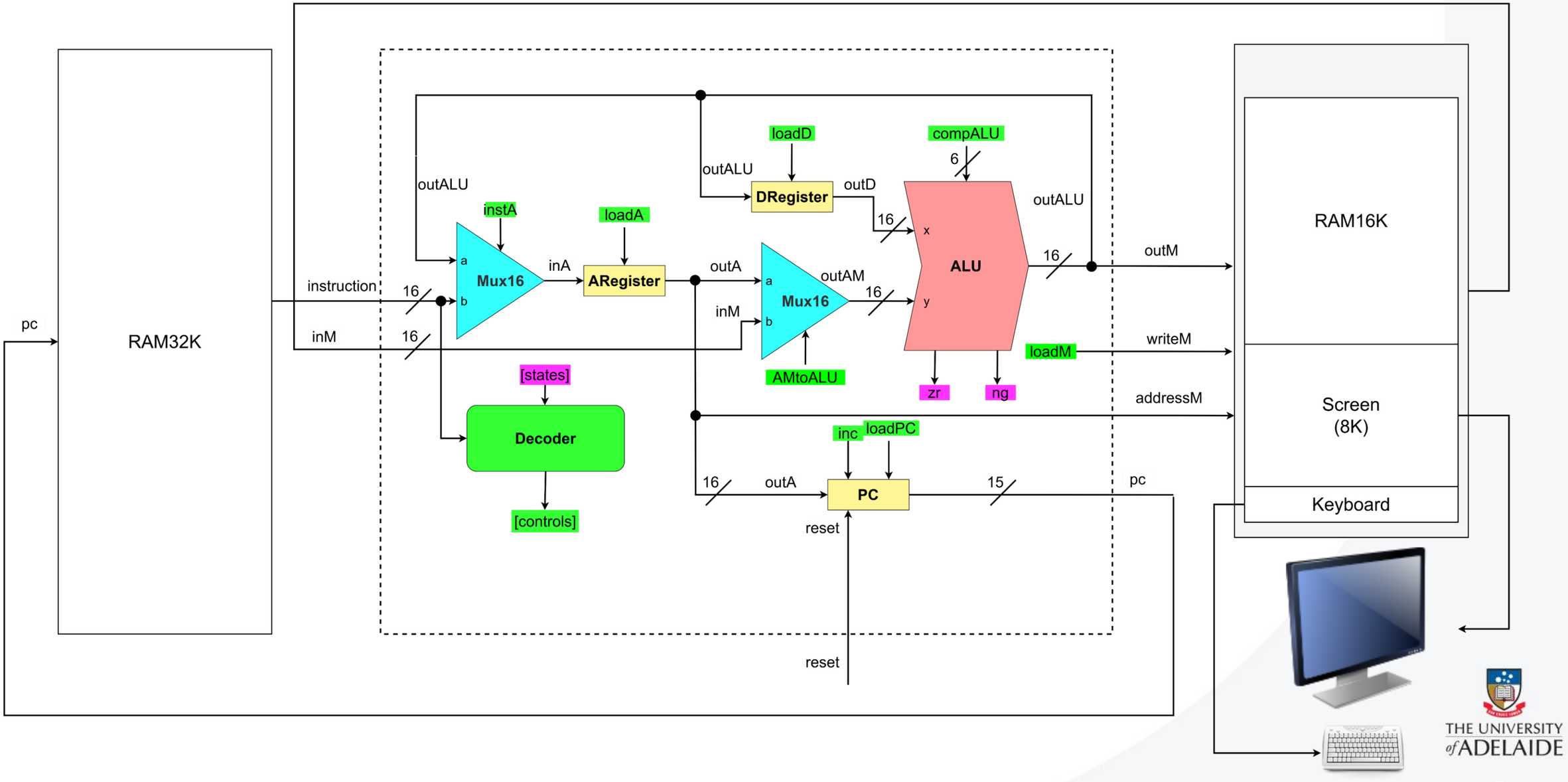
w-bit register



Computer Architecture

The Hack Computer

No worries for now



The Hack computer

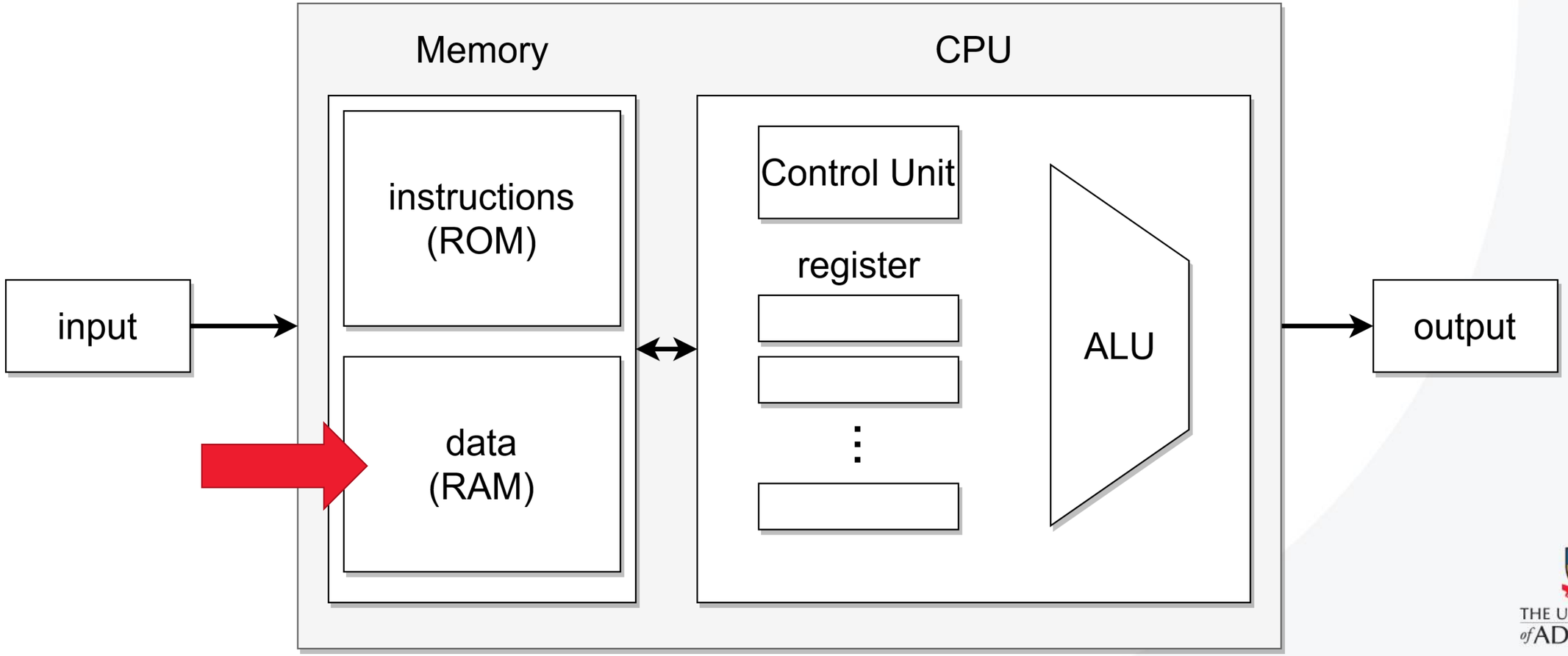
No worries for now

A 16-bit machine consisting of the following elements:

- Data memory: **`RAM`** – an addressable sequence of registers
- Instruction memory: **`ROM`** – an addressable sequence of registers
- Registers: **`D`**, **`A`**, **`M`**, where **`M`** stands for **`RAM[A]`**
- Processing: **`ALU`**, capable of computing various functions
- Program counter: **`PC`**, holding an address to the instruction memory
- Control: The **`ROM`** is loaded with a sequence of 16-bit instructions, one per memory location, beginning at address 0. Fetch-execute cycle: later
- Instruction set: Two instructions types: A-instruction, C-instruction.

THE UNIVERSITY of ADELAIDE

Von Neumann Architecture



THE UNIVERSITY of ADELAIDE

Random Access Memory (RAM)

We've seen what a multi-bit register looks like.

But first…

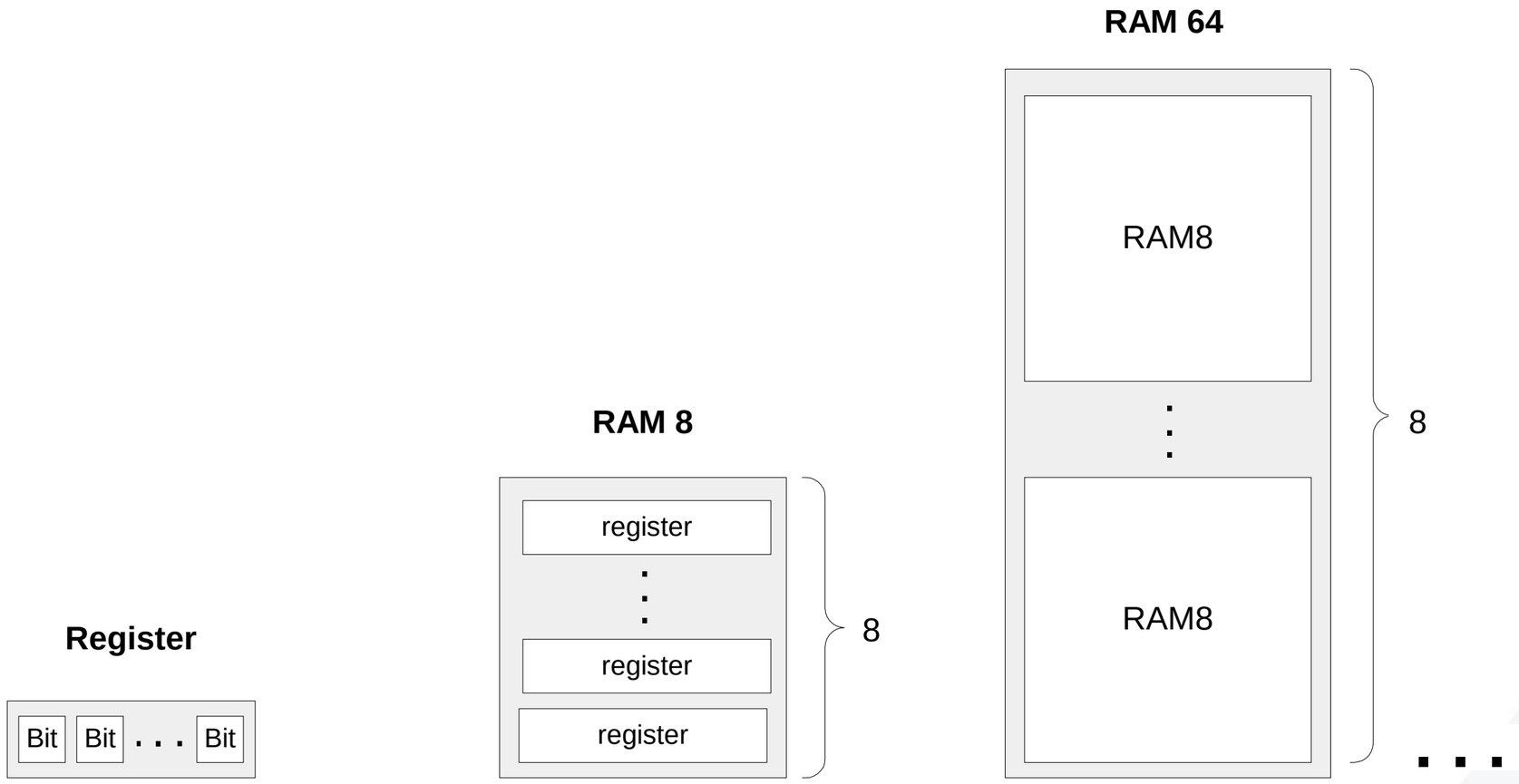
Terminology check! What are the following:

- Bit: one logic '0' or '1', the smallest unit of data that a computer can process and store.
- Byte: 8 bits in a row.
- Word: fix-sized unit depends on the hardware architecture (register size).

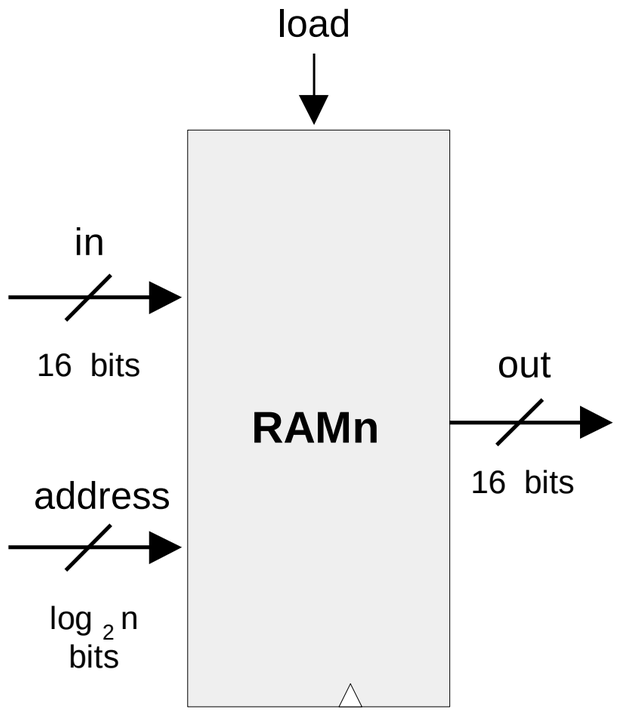
What do each of these look like in terms of abstract diagrams?

THE UNIVERSITY of ADELAIDE

From Little Things Big Things Grow



RAM interface

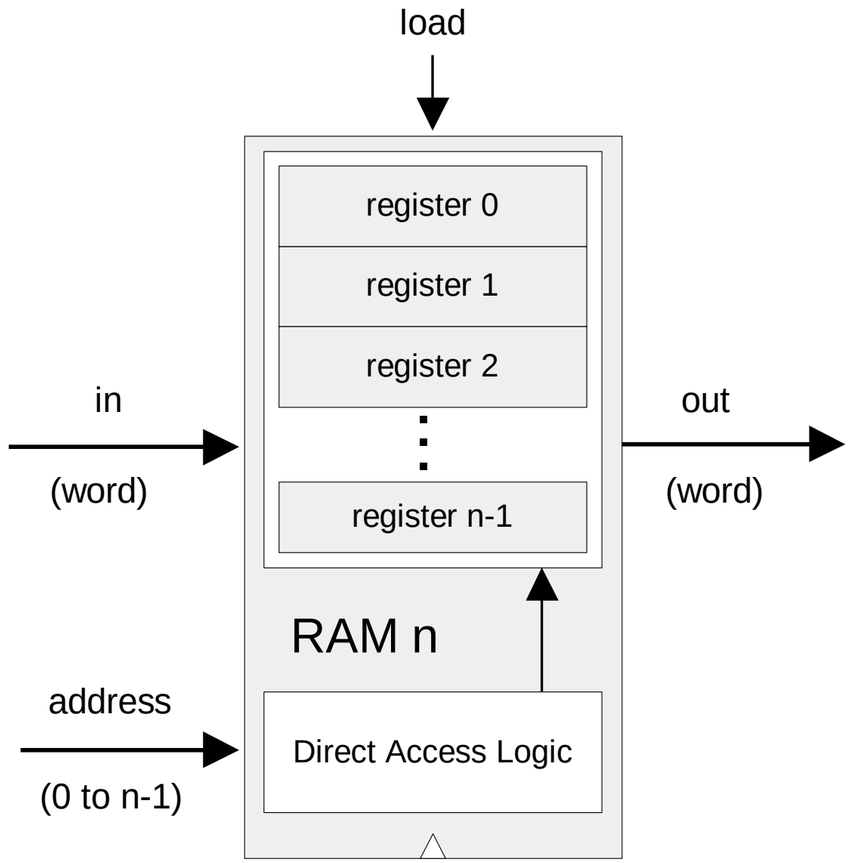



```
Chip name:   RAMn  // n and k are listed below
Inputs:      in[16], address[k], load
Outputs:     out[16]
Function:    out(t)=RAM[address(t)](t)
             If load(t-1) then
                RAM[address(t-1)](t)=in(t-1)
Comment:     "=" is a 16-bit operation.
```

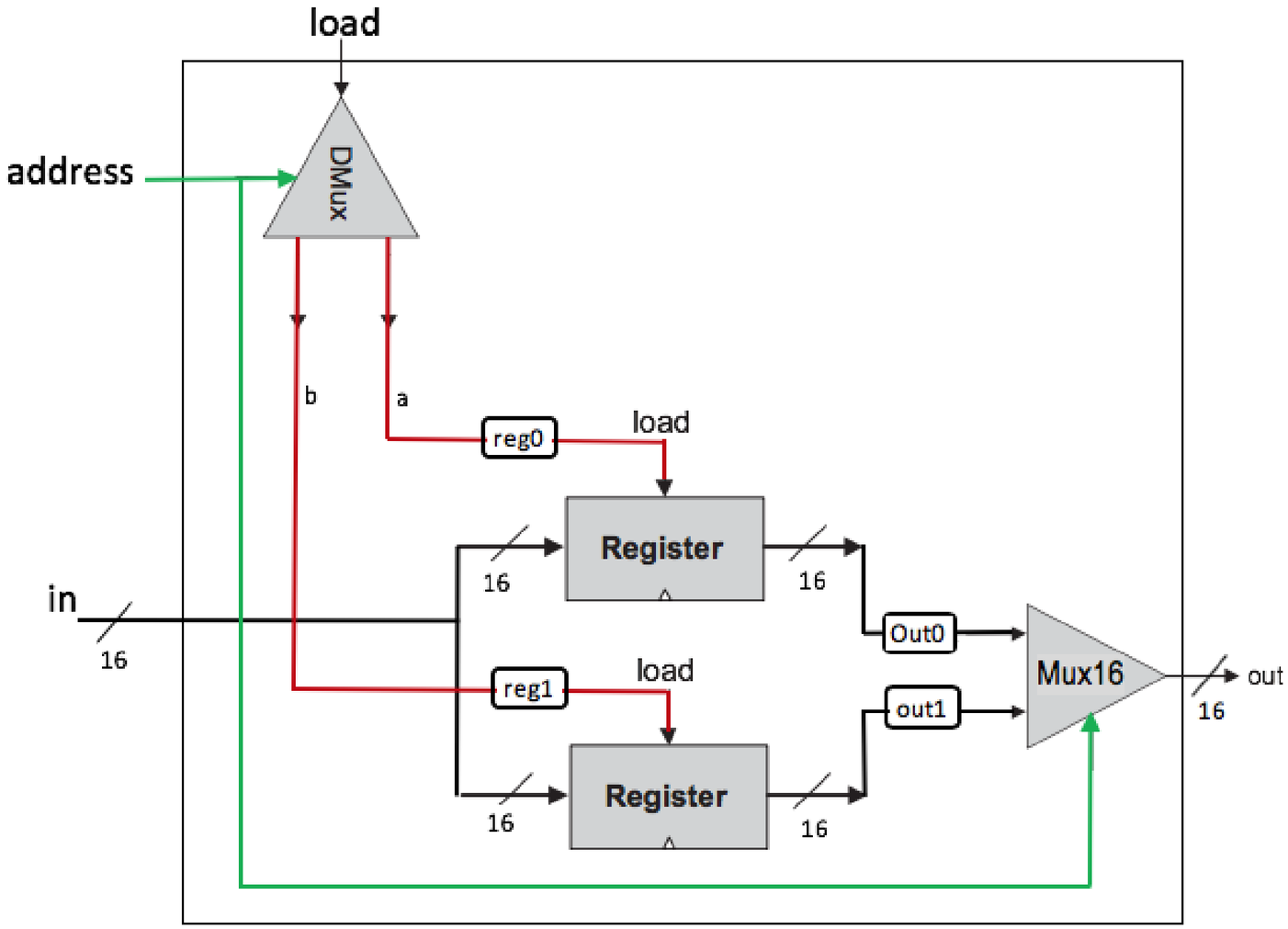
The specific RAM chips needed for the Hack platform are:

Chip name	n	K
RAM8	8	3
RAM64	64	6
RAM512	512	9
RAM4K	4096	12
RAM16K	16384	14

RAM – How do we access the right data?

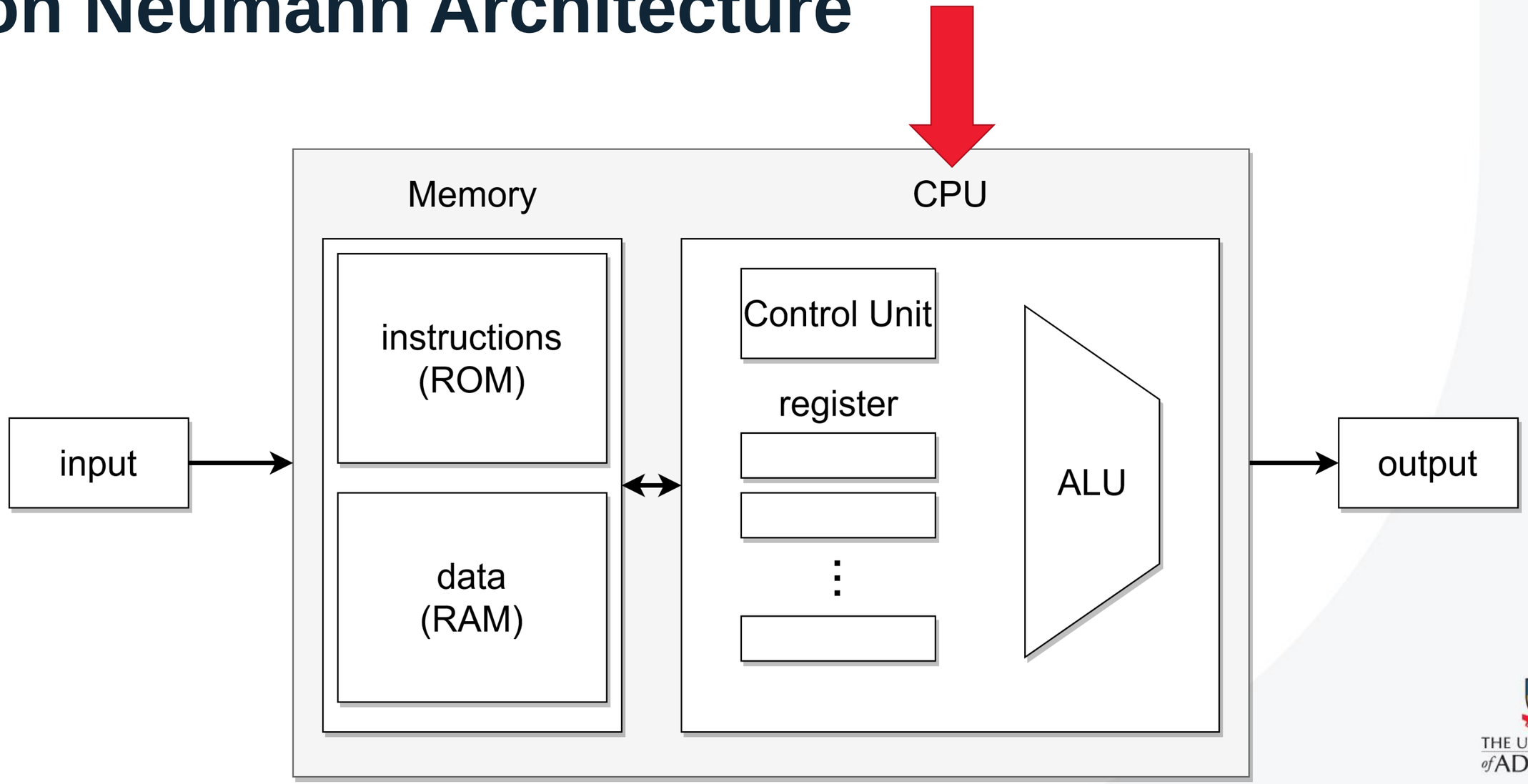


How Address Works



Why does the DMux distribute the signal, load, rather than the new value, in?

Von Neumann Architecture



Bringing it Together

What we understand:

- How to use logic gates to do basic arithmetic & logic operations
- How to use DFFs to build registers & RAM

How can we use this to construct a working system, or even smaller, a CPU?

What does a CPU require?

We'll need a way to

- Perform different Arithmetic and Logic operations
  - … and choose between those operations
  - … and perform those operations in some sequence
- Input and Output the data we want to work on
  - Specify location to read data in and write data out

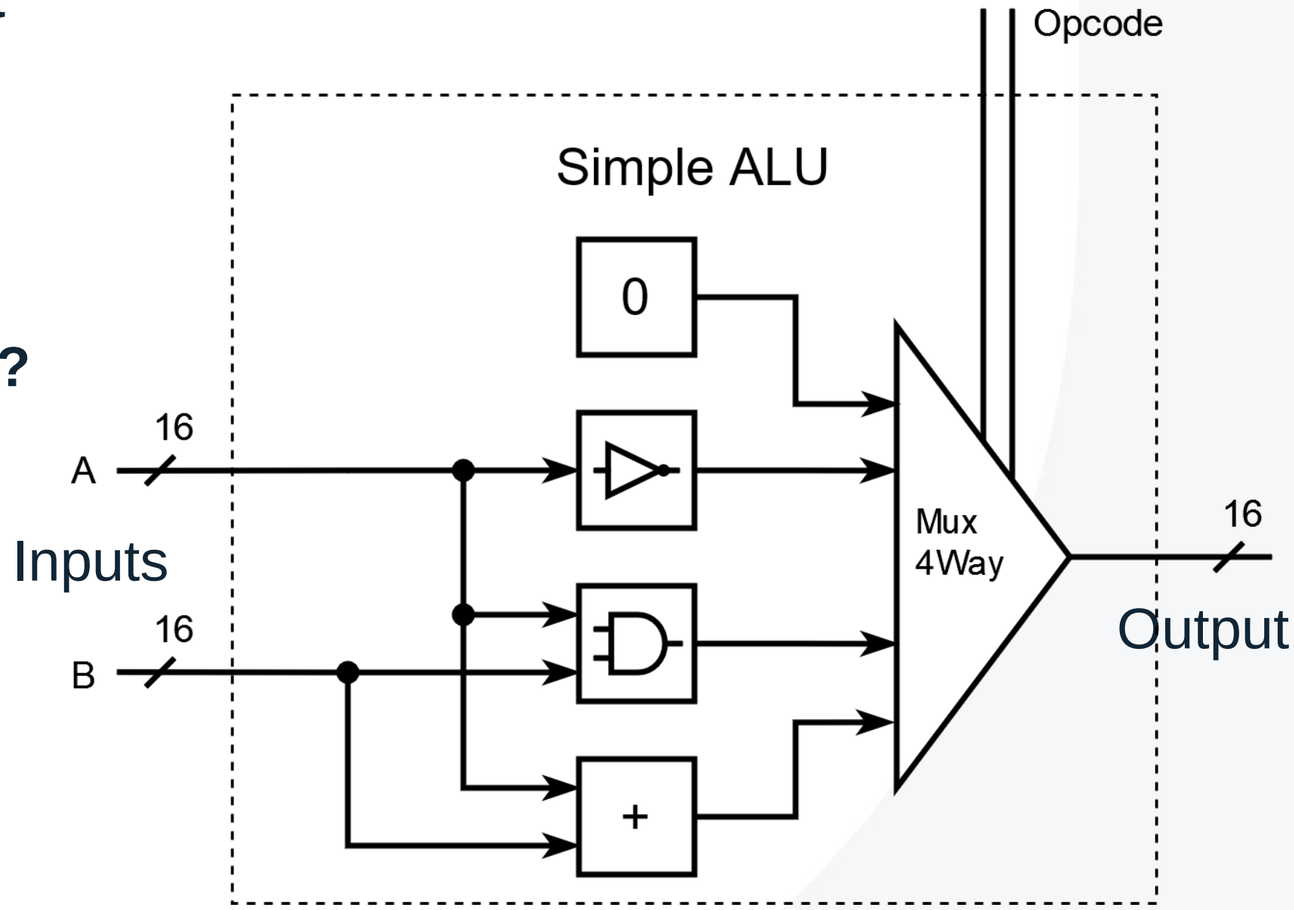
The ALU

An arithmetic logic unit (ALU) is a circuit that performs basic arithmetic and logic operations

How can we choose different operations?

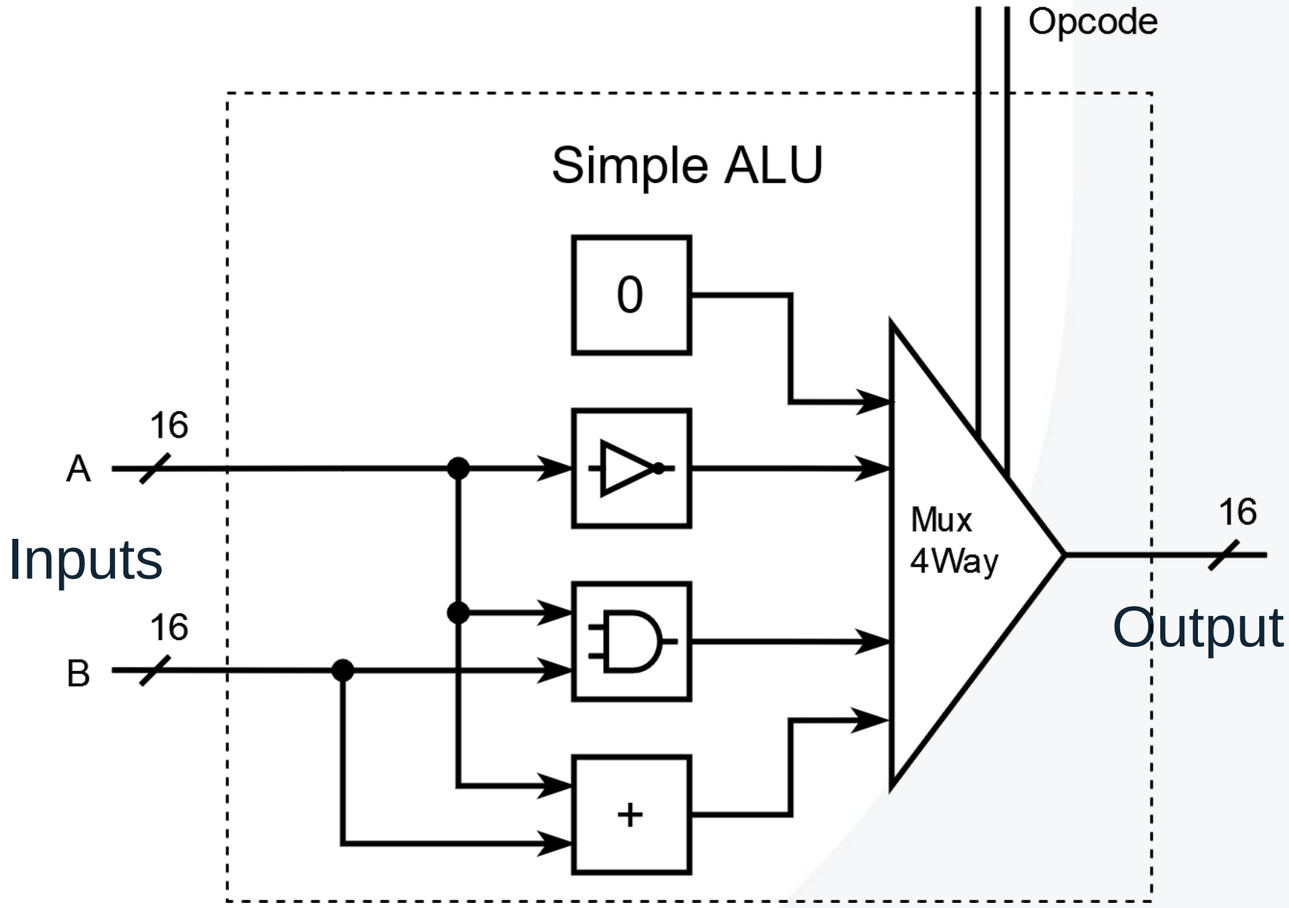
- Use Multiplexers!
- The sequence of bits that determines which operations are selected by the multiplexers is called an **Opcode**

Consider this Simple ALU →



A Simple ALU

- This ALU has 4 basic operations:
  - Output 0
  - Bitwise NOT the first input
  - AND the inputs
  - ADD the inputs
- The Opcode bits determine which operation is used:
  - 00 → 0
  - 01 → $\overline{A}$
  - 10 → $A \cdot B$
  - 11 → ADD(A, B)

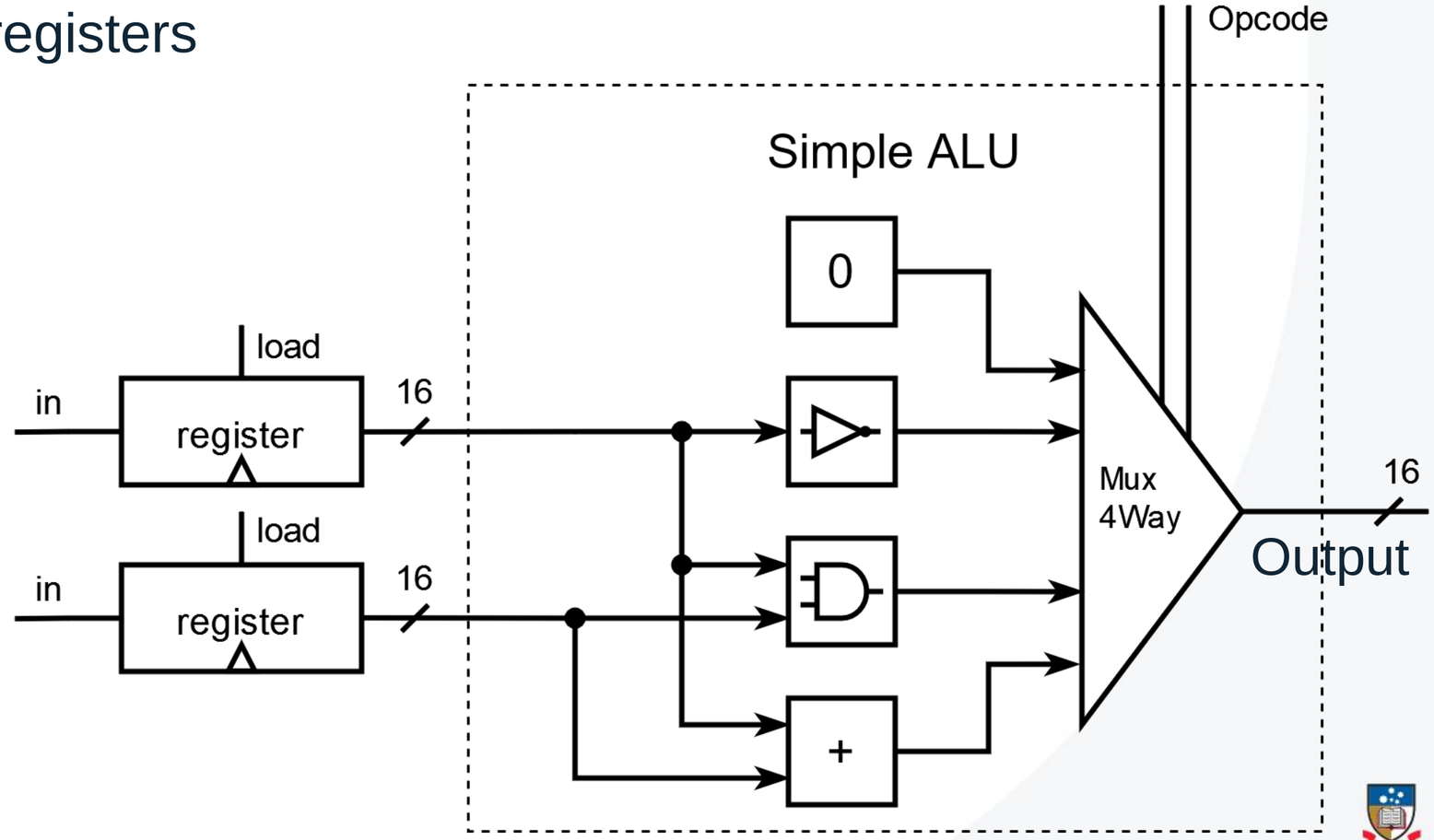


Where do the input values come from?

… with registers

Where do the input values come from?

- We can store them in registers

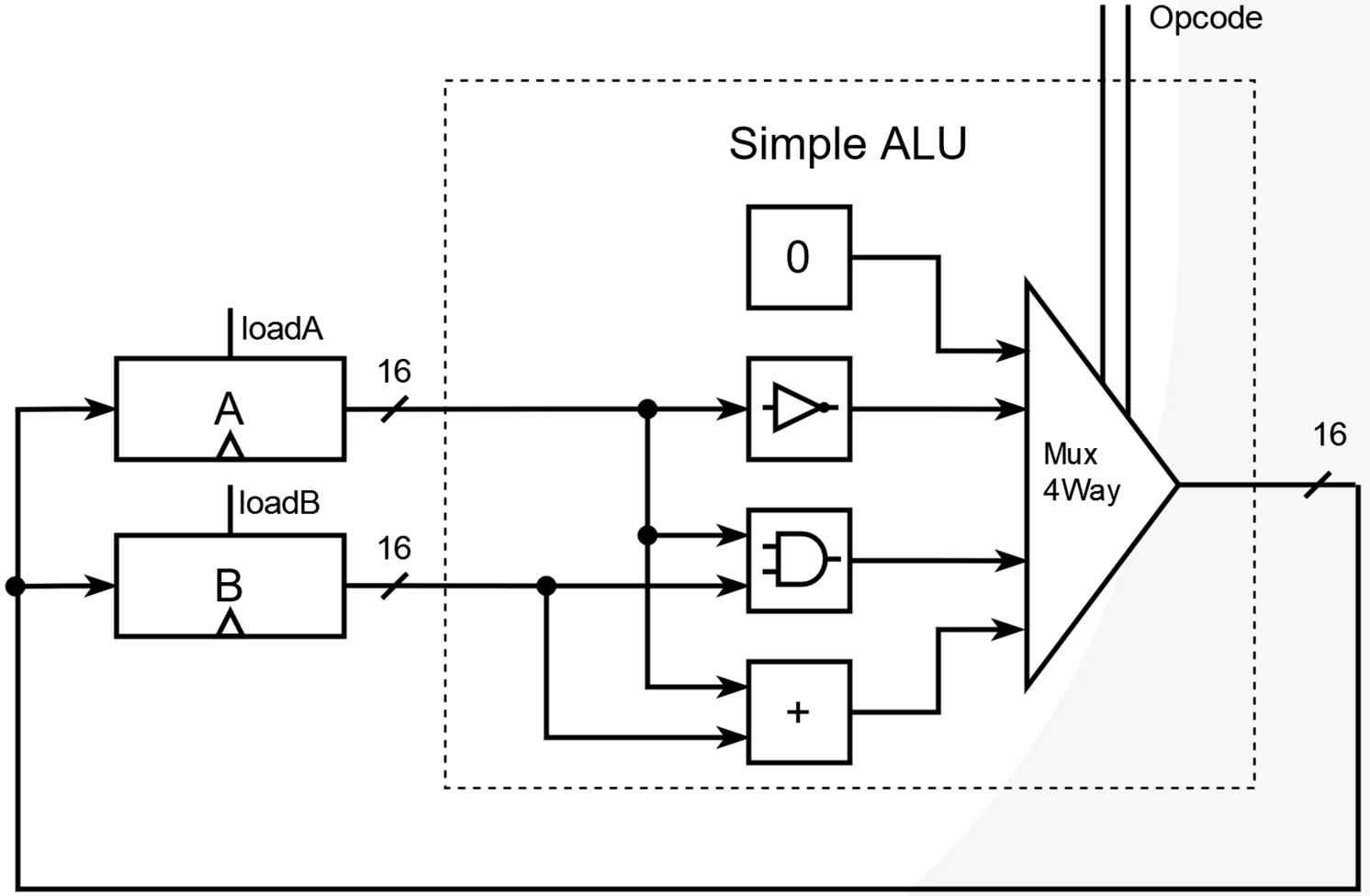


Where does the output value go to?

… to a Simple CPU

Where does the output value go to?

- We can also use registers
- The load bits of the registers let us choose which register to write to (or neither or both).
  - 00 → do not write
  - 10 → write to register A
  - 01 → write to register B
  - 11 → write to both



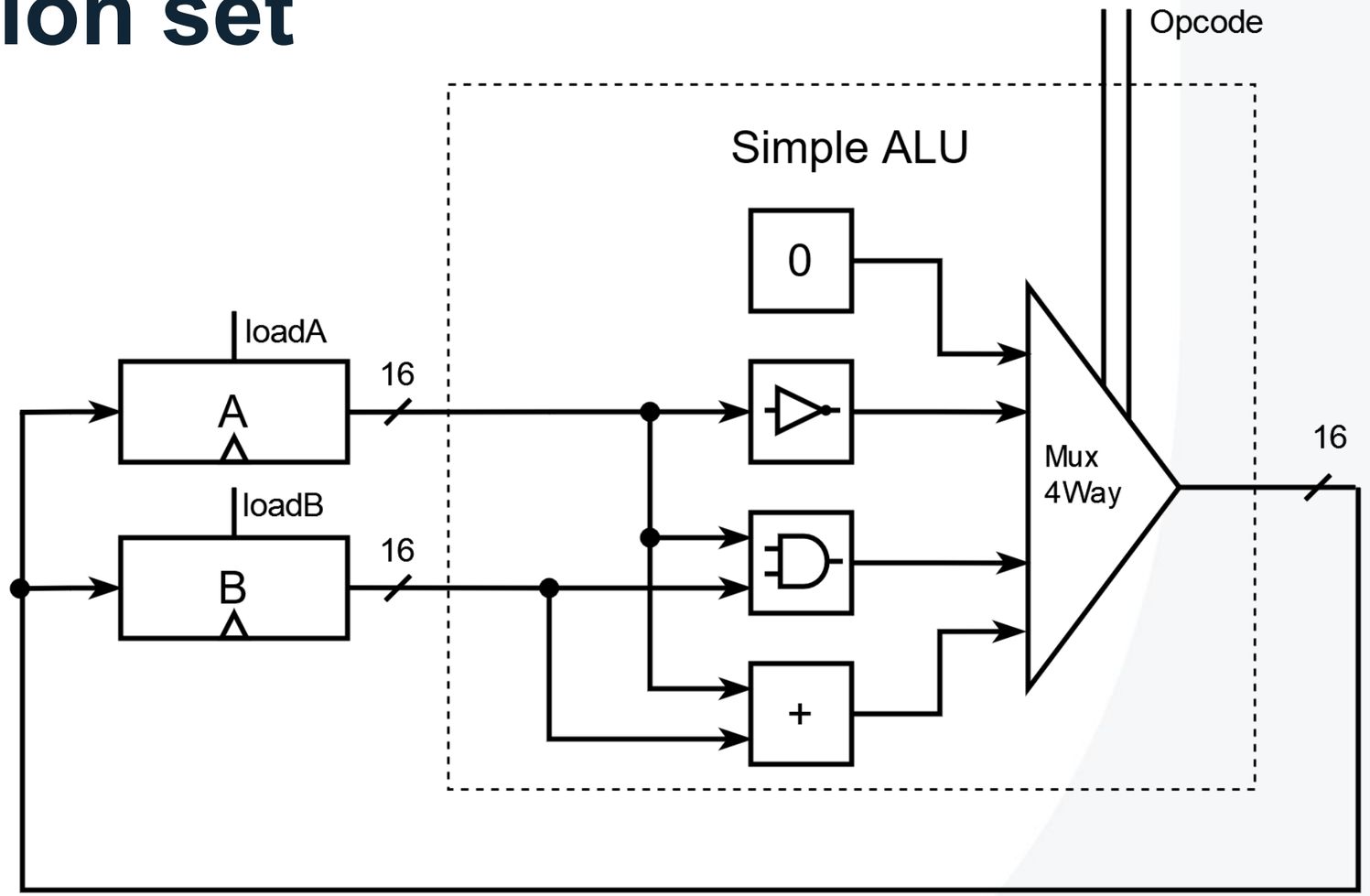
Now using sequences of 4 bits, we can perform basic computations

- We call these combinations of control bits **instructions**

… with an instruction set

Example:

Opcode		loadA	loadB
0	0	1	0
0	1	1	1
1	1	1	0
0	1	1	1
1	1	1	1
1	1	1	1



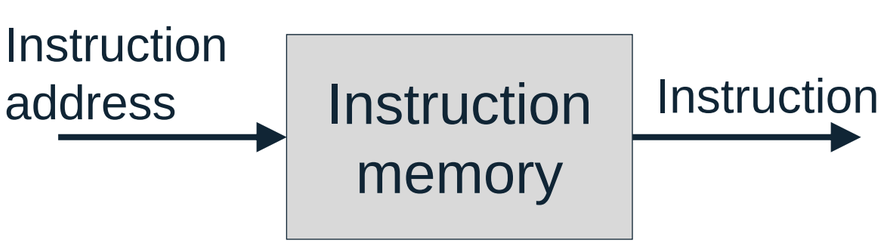
Assume output = 0 initially

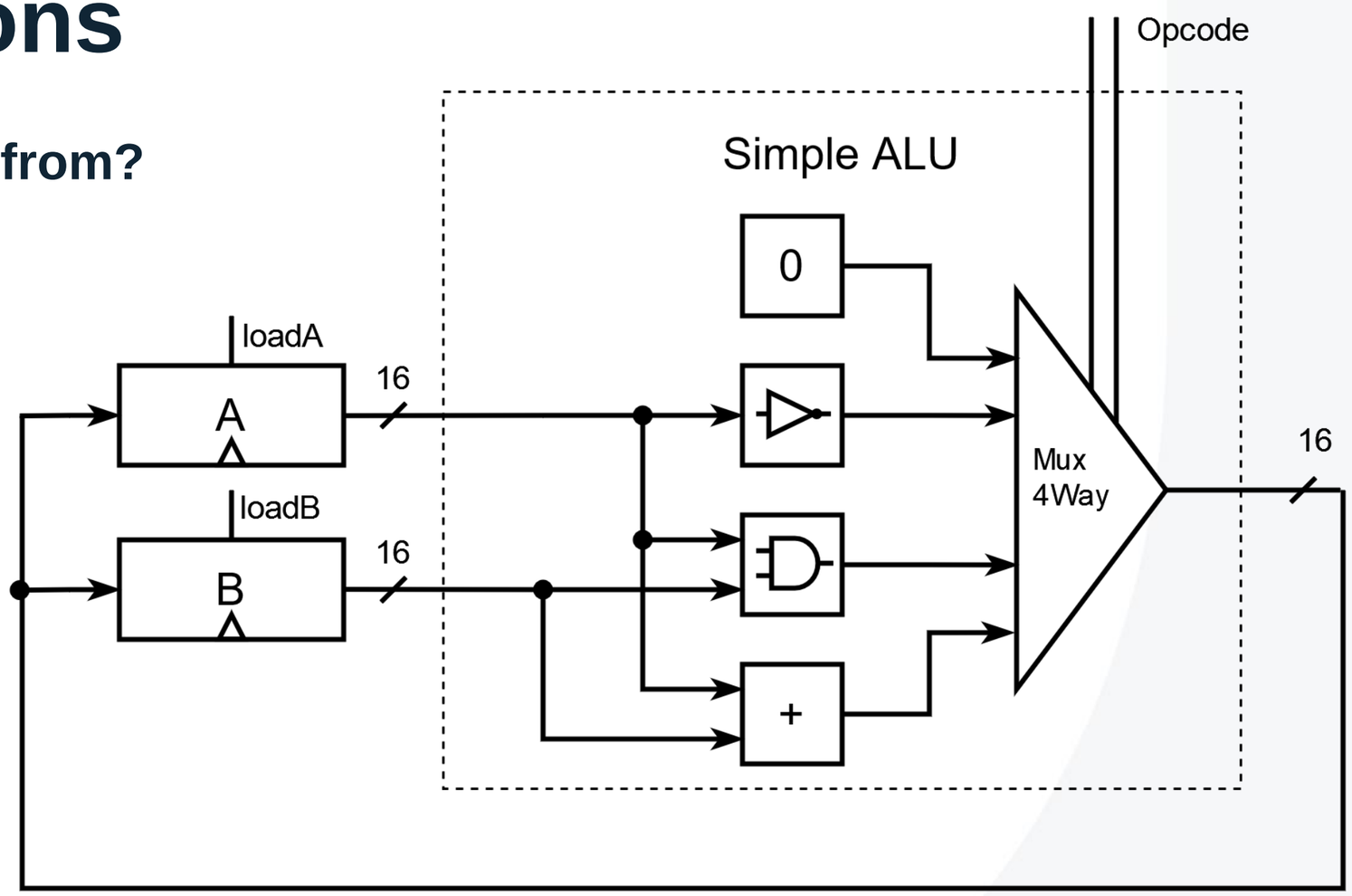
THE UNIVERSITY of ADELAIDE

Running instructions

Where do the instructions come from?

- We can store them in memory
- Feed them 1-by-1 to the CPU





How can we track which instruction is next?

We need a counter!

THE UNIVERSITY of ADELAIDE

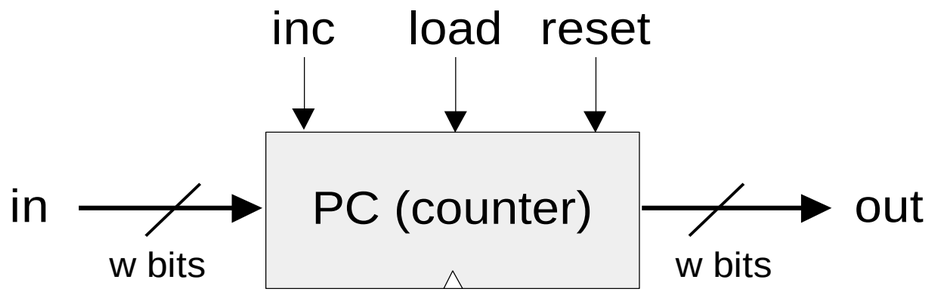
Counter

We need to develop the logic for a counter as the simplest program is run from contiguous places in memory, one after the another.

What do we need for a counter?

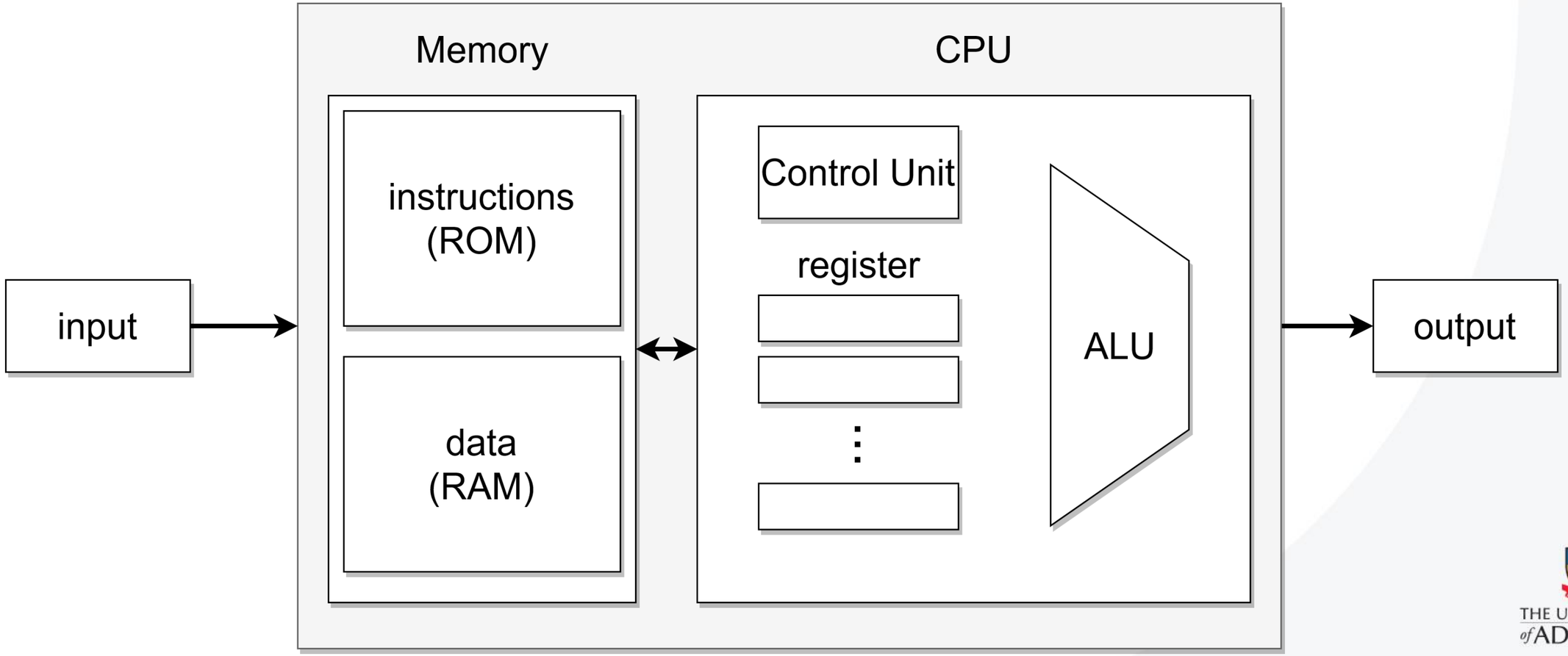
- Reset to 0
- Set counter to some value
- Increment counter in every clock cycle
- Pause if needed

Program Counter - Diagram



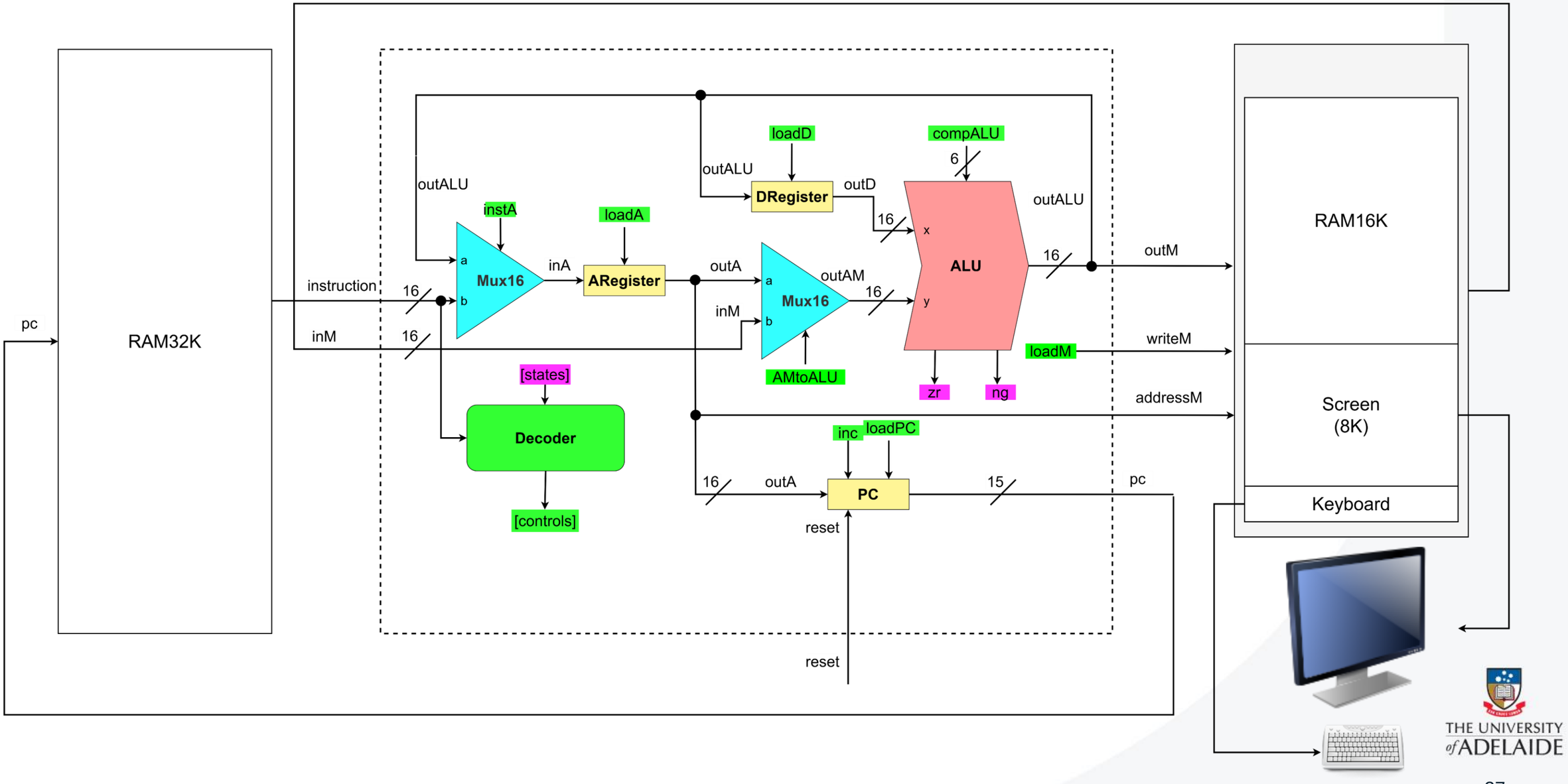
- When reset pin 1, counter resets (outputs 0 next clock tick)
- Else when load pin 1, counter set to input (outputs in next clock tick)
- Else when inc pin 1, counter increments (outputs current value + 1 next tick)
- Else counter unchanged (outputs current value next tick)

Summary


Memory
CPU
instructions
(ROM)
data
(RAM)
Control Unit
register
ALU
input
output

The Hack Computer

No worries for now



This Week

- Review Chapters 3 & 4 of the Text Book (if you haven't already).
- Assignment 2 Due this Friday (12 Aug 11:59 pm).
- Start Assignment 3 (available now).
- Practical Exam in week 4 during Workshops (come to the Workshop you enrolled).
  - Practice Exam will be available week 3 Wednesday.
- Read Chapter 5 of the Text Book before week 4.

THE UNIVERSITY of ADELAIDE