We acknowledge and pay our respects to the Kaurna people,
the traditional custodians whose ancestral lands we gather on.

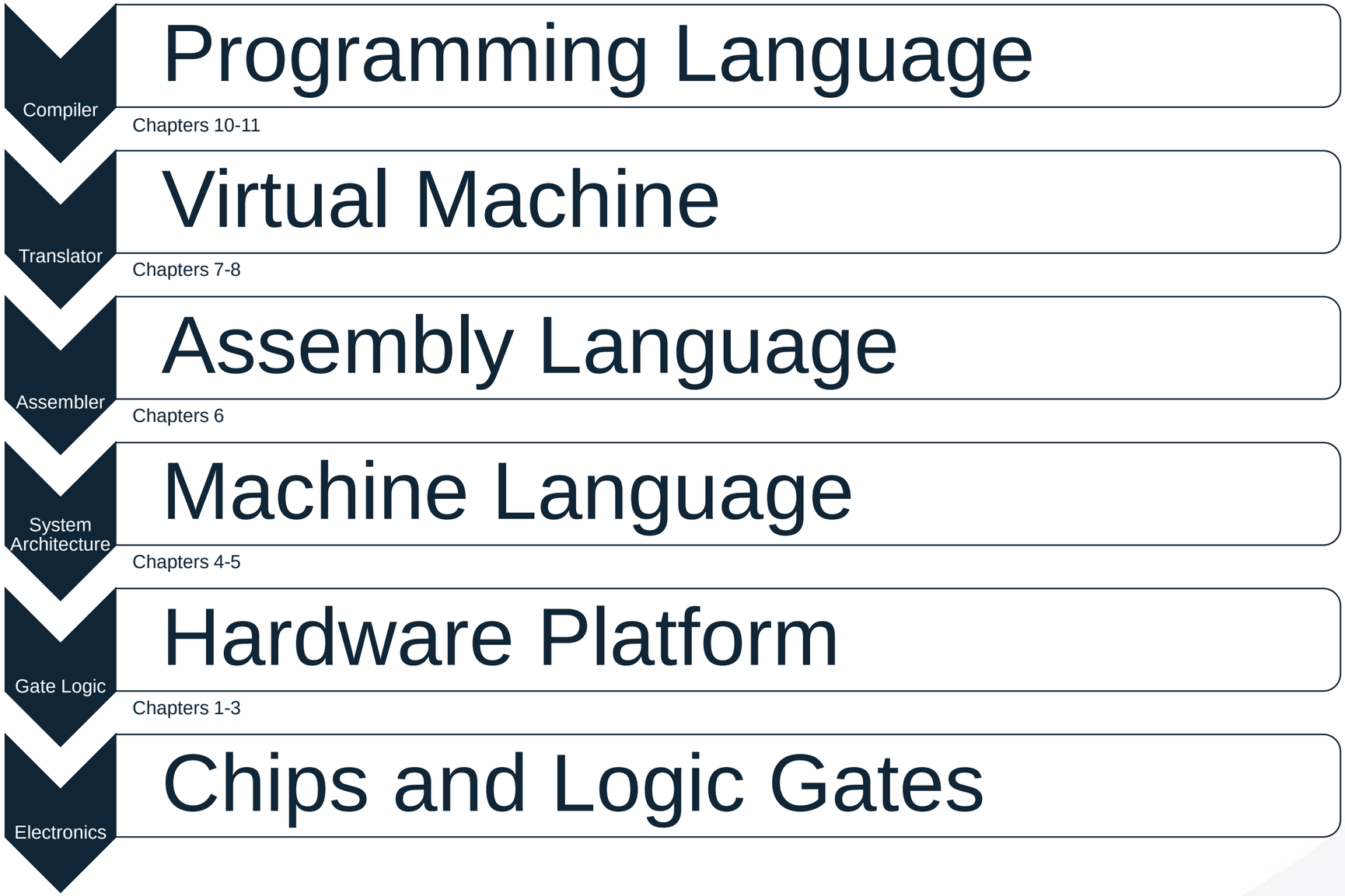
We acknowledge the deep feelings of attachment and relationship of the Kaurna people to country and we respect and value their past, present and ongoing connection to the land and cultural beliefs.

THE UNIVERSITY of ADELAIDE

Computer Systems

Lecture 03: Memory and Machine Language

Review: The whole system

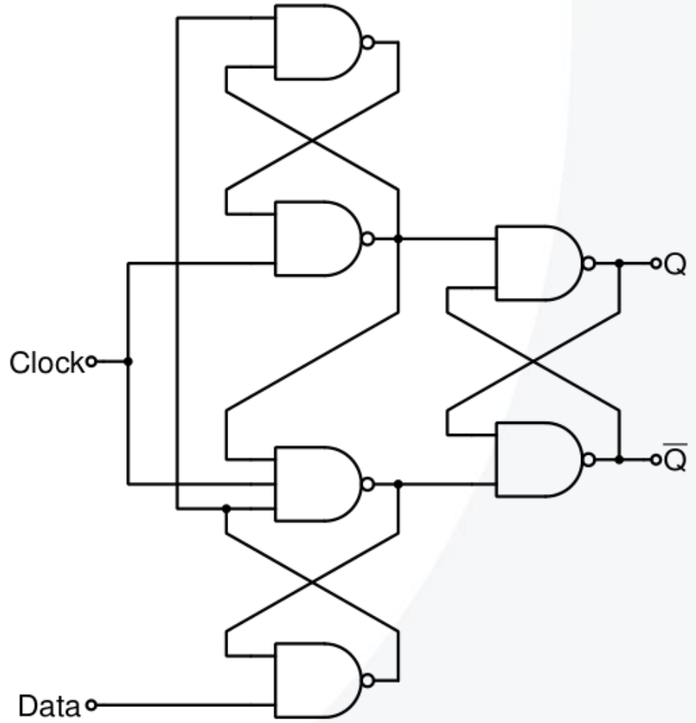


THE UNIVERSITY of ADELAIDE

Review: DFF

- Holds the input value for 1 clock tick
- Can be used to build a Register

out(t) = in(t-1)

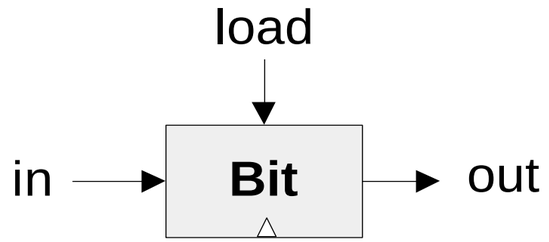


Review: 1-bit register

We want to be able to:

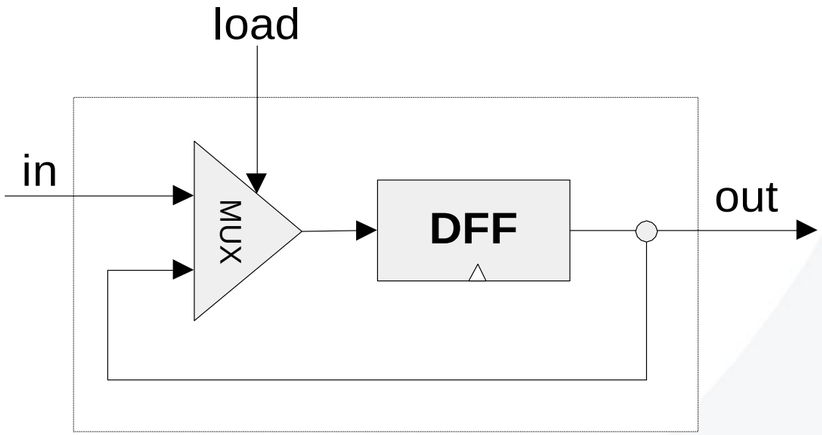
Change the state of the bit

Retain that state until we want to change it.



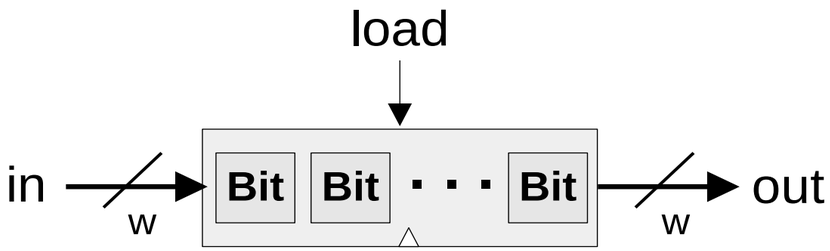
if load(t-1) then out(t)=in(t-1)
else out(t)=out(t-1)

Implementation



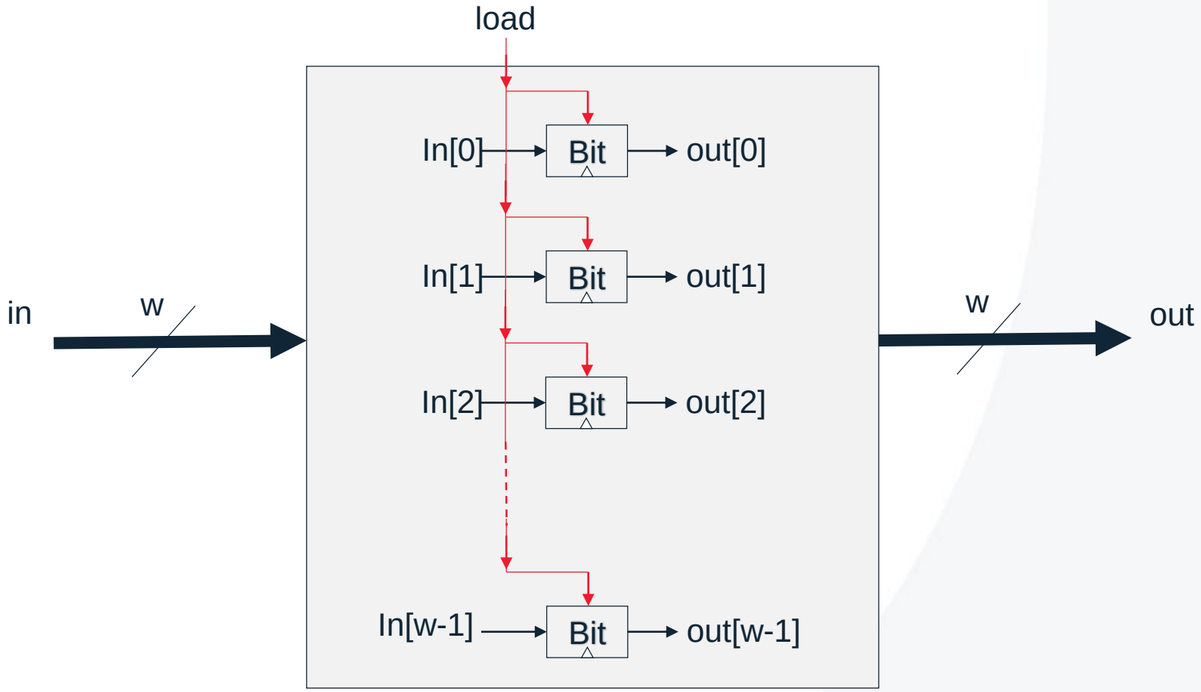
Review: Multi-bit registers

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



Memory

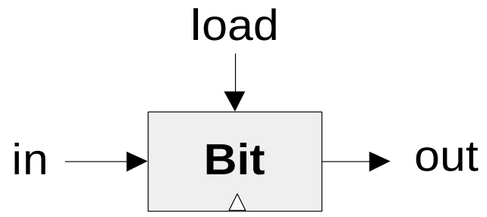
Random Access Memory

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

But first…

Terminology check! What are the following:

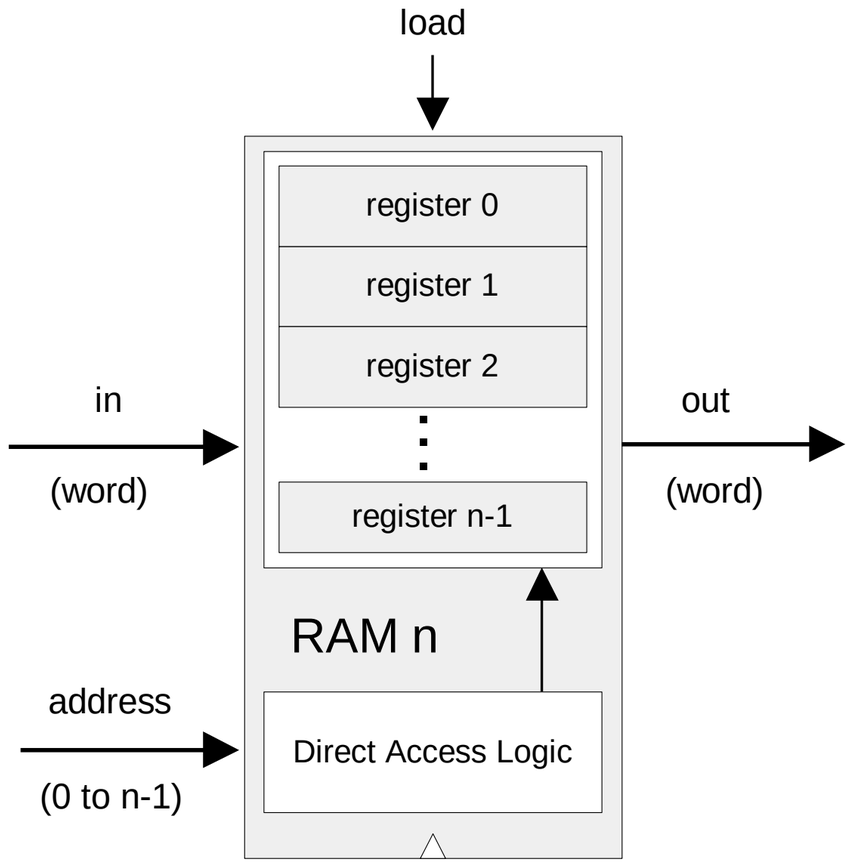
- Bit
- Byte
- Word



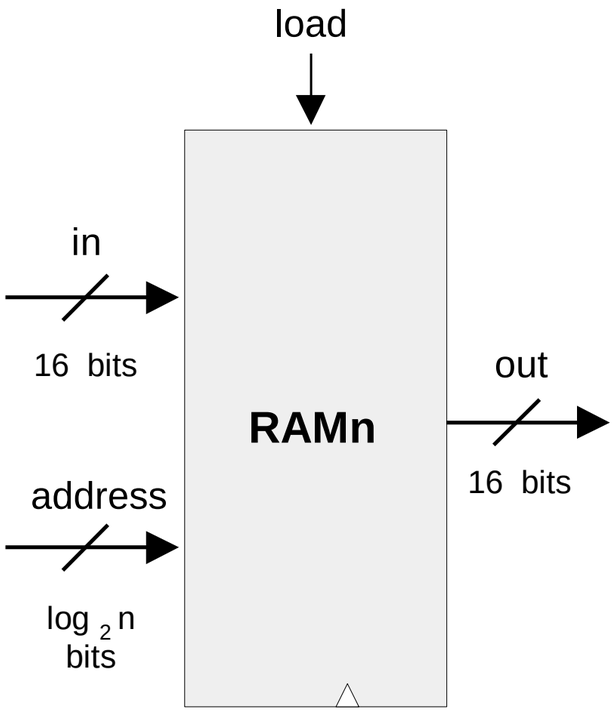
if load(t-1) then out(t)=in(t-1)
else out(t)=out(t-1)

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

RAM – How do we access the right data?



RAM interface

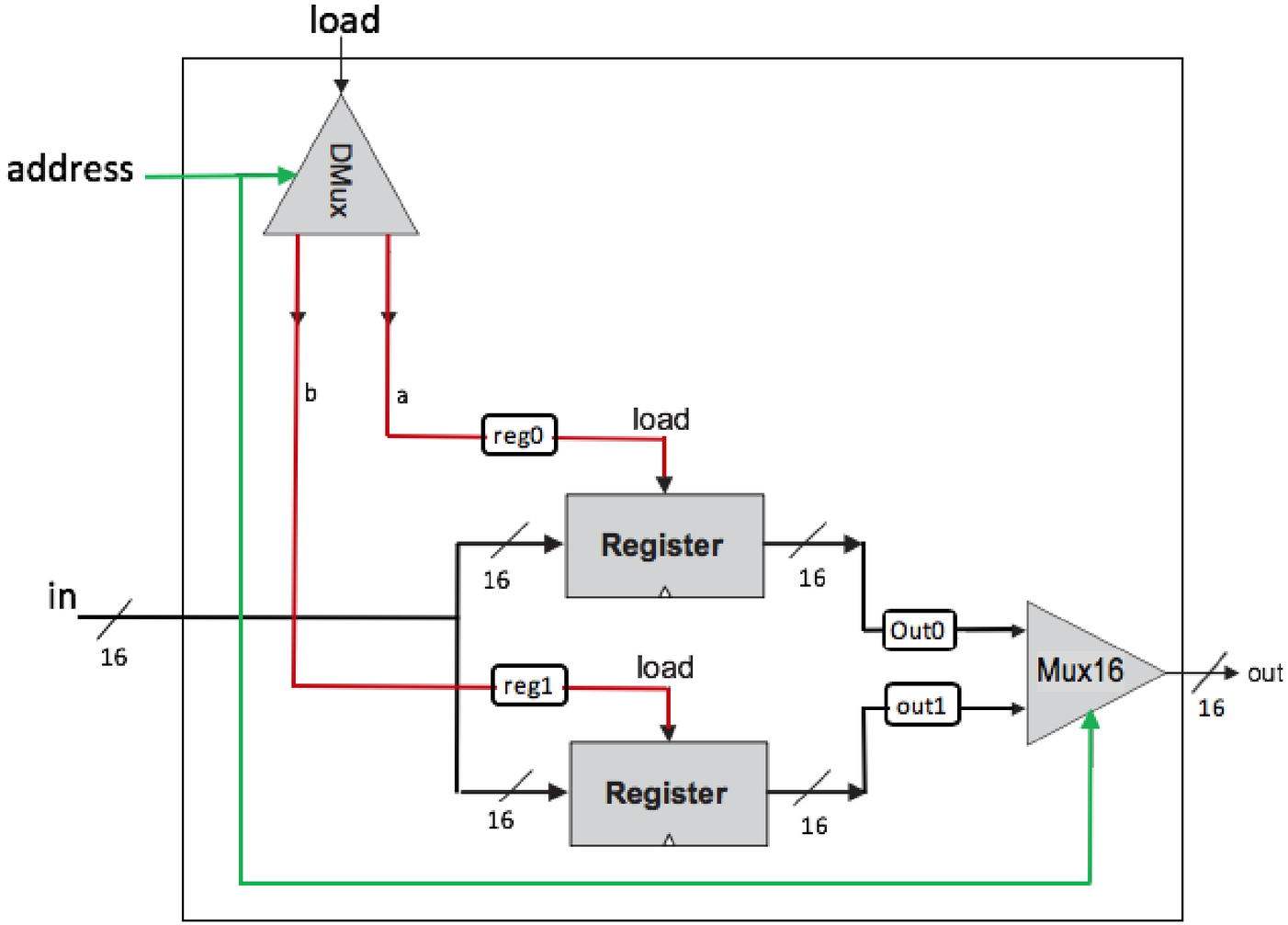



```
Chip name:   RAMn  // n and k are listed below
Inputs:      in[16], address[k], load
Outputs:     out[16]
Function:    out(t)=RAM[address(t)](t)
             If load(t-1) then
                RAM[address(t-1)](t)=in(t-1)
Comment:     "=" is a 16-bit operation.
```

The specific RAM chips needed for the Hack platform are:

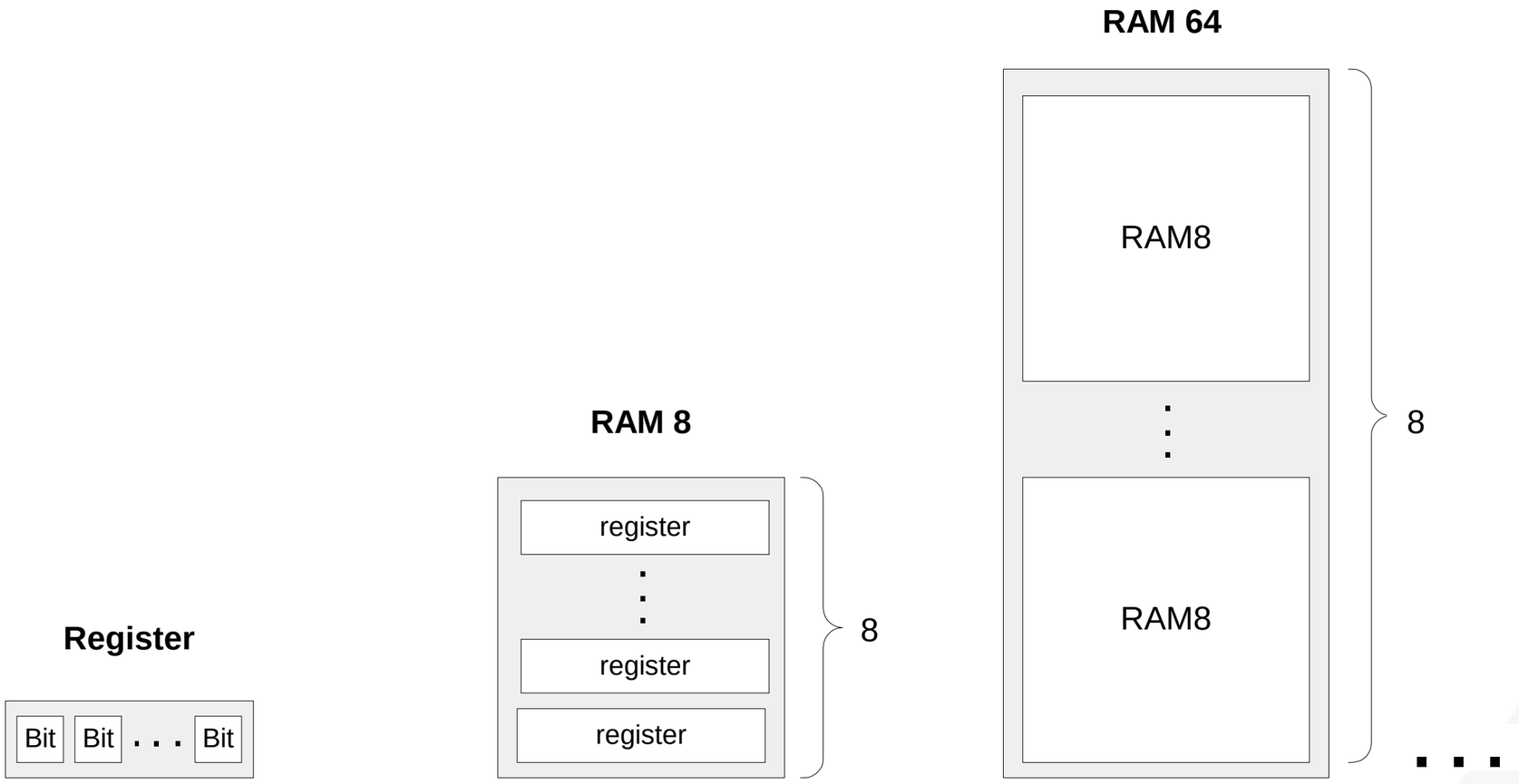
Chip name	n	K
RAM8	8	3
RAM64	64	6
RAM512	512	9
RAM4K	4096	12
RAM16K	16384	14

RAM – 2 Register Internals



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

From little things…



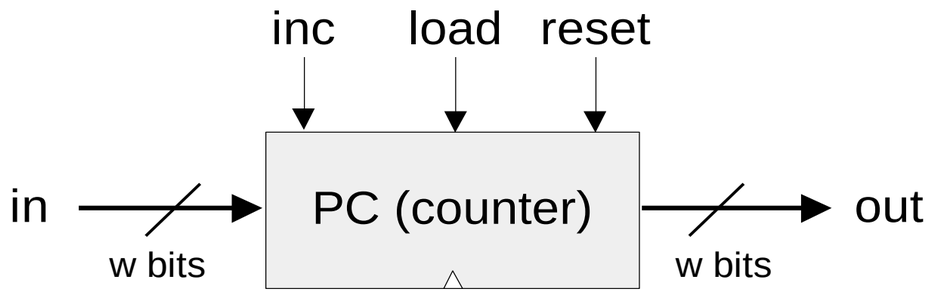
Counter

We can use the clock signal and our bit implementation to store things, including our program.

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?
- Set state to some base value
- Increment state in every clock cycle
- Stop incrementing over clock cycles
- Reset state

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)

RAM Terminology

SRAM

Static RAM, what we have described in this lecture

If it loses power, it forgets

DRAM

Dynamic RAM, built using capacitors and transistors

It forgets even if it has power

Flash RAM

Used in most solid state disks

It cannot be overwritten, whole blocks must be erased first

It forgets if changed too many times

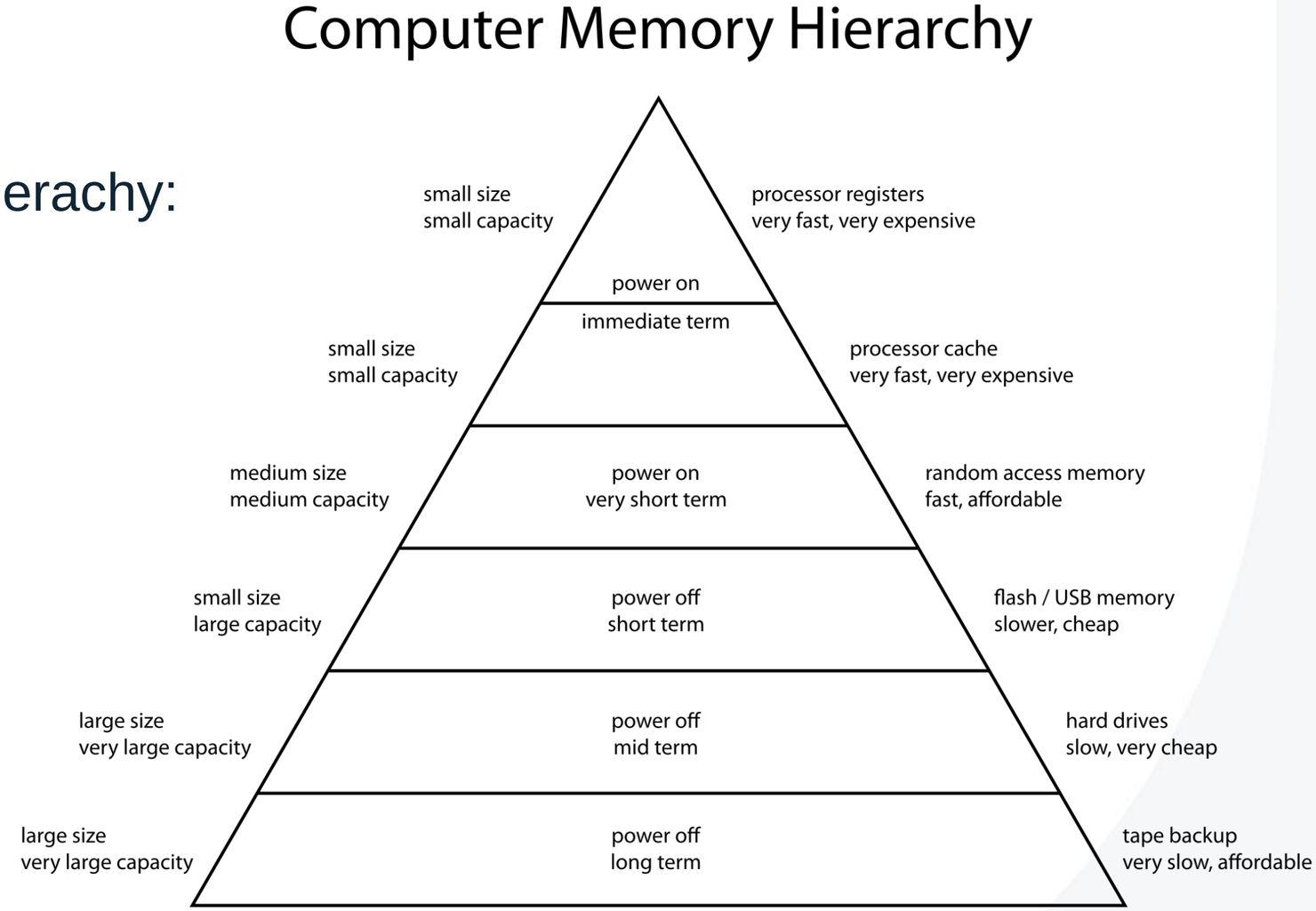
Caches

Keep local copies of data held in much slower devices

Aide in the illusion of speed

Hierarchy

Storage is organized in a hierachy:



Machine Language

THE UNIVERSITY of ADELAIDE

Early Programming

Early machines

Jacquard loom

Automata

- Physical coding on cards to trigger mechanical action

Early calculators

Babbage's Difference Engine

- Not programmable in the true sense.
- Mechanical and analogue calculator

Babbage's Analytical Engine

- Programmable for calculation
- First real 'program' for Bernoulli numbers by Ada, Countess of Lovelace. (Never actually ran but would have worked.)

THE UNIVERSITY of ADELAIDE

High Level Programming Languages

- A high-level language has to be translated to a machine-interpretable form prior to execution.
- Translation often turns the HLL into an intermediate language that is then translated to machine code.
  - Why is there an intermediate step?
- HLLs are highly readable and allow a great deal of abstract representation
  - Each statement may be translated into many machine code instructions.

Thinking about Language and Hardware

Machine language (instruction set) can be viewed as a programmer-oriented abstraction of the hardware platform

The hardware platform can be viewed as a physical means for realizing the machine language abstraction

Machine Language

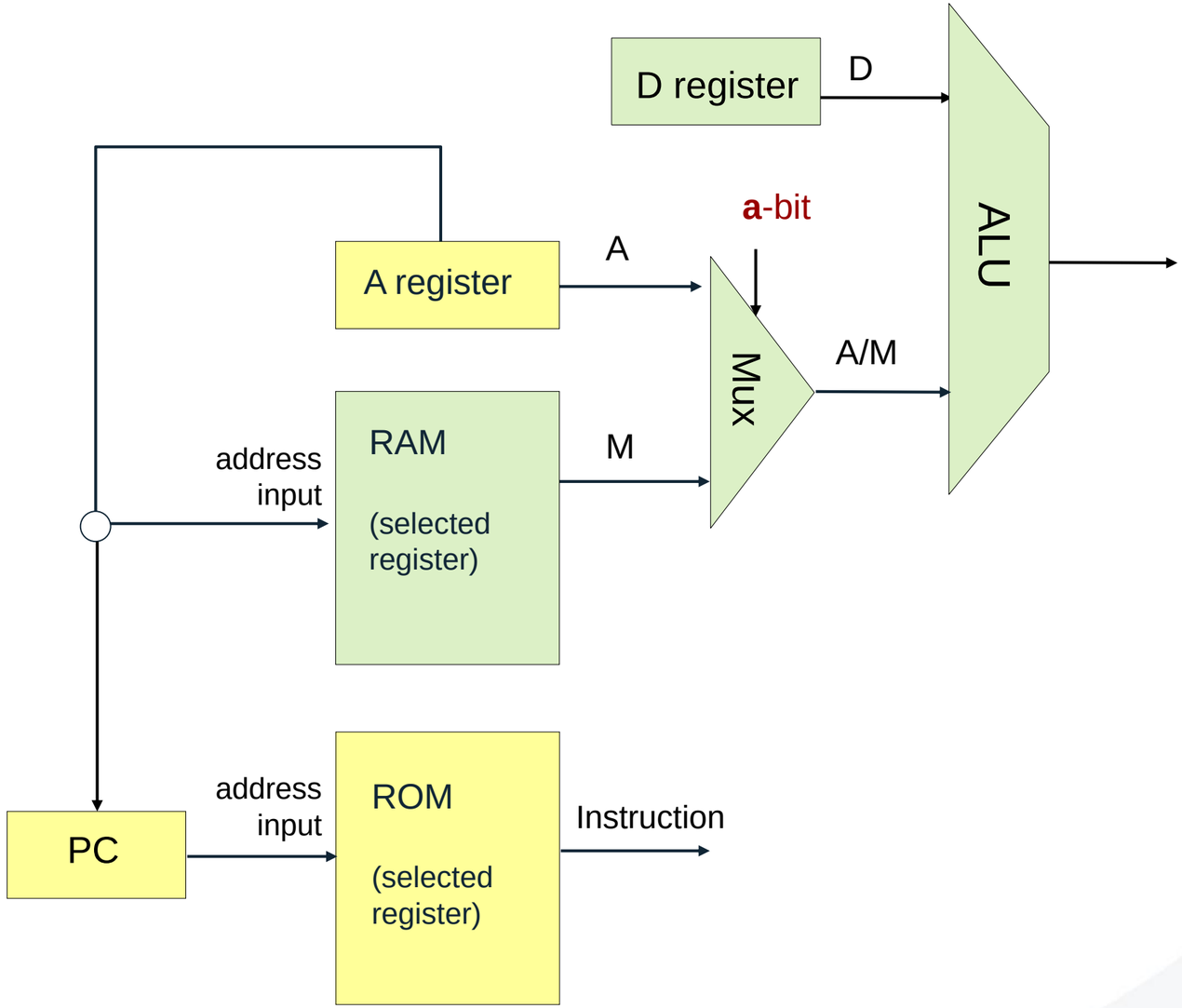
- Although the hardware is different, the idea of what a machine language does is the same from machine to machine.
  - an agreed-upon formalism for manipulating memory using a processor and a set of registers
- If you understand how one machine language works, you can understand (almost) all of them.
  - This assumes Von Neumann architectures!

How do we build it?

There are a number of key hardware components that may be required, these could include:

- RAM to store data
- ROM to store the program
- PC to identify where to find the next instruction in ROM
- A register to store the address in RAM we wish to access
- A register for storing temporary values
- An ALU to compute arithmetic and logic operations

A basic architecture



What instructions do we *need?*

- Mathematical operations
- Logical operations
- Flow of control instructions
- Conditions
- Write to memory
- Reading from memory
- Do nothing
- …

Typical machine language (a small sample)

```
// In what follows R1,R2,R3 are registers, PC is program counter,
// and addr is some value.

ADD R1,R2,R3      // R1 ← R2 + R3

ADDI R1,R2,addr   // R1 ← R2 + addr

AND R1,R1,R2      // R1 ← R1 and R2 (bit-wise)

JMP addr          // PC ← addr

JEQ R1,R2,addr    // IF R1 == R2 THEN PC ← addr ELSE PC++

LOAD R1, addr     // R1 ← RAM[addr]

STORE R1, addr    // RAM[addr] ← R1

NOP               // Do nothing

// Etc. – some 50-300 command variants
```

Typical machine language

- Machine language typically refers to the binary representation of a program.
- Assembly language is a human readable description of machine language, as on the previous slide.
- Machine language usually provides a way to embed explicit values in a program, eg the ADDI instruction.
- Machine language is usually specific to the hardware and / or assembler program that translates assembly language into binary.

Thinking about commands

CISC – Complex Instruction Set Computing

- The goal is to do as much possible in a single instruction
- Early machines were very slow and this approach help a lot!
- Instructions could be different sizes
- Instructions could read and write memory at the same time

RISC – Reduced Instruction Set Computing

- The goal is keep things simple and therefore fast
- The increased number of instructions needed to get the same result was offset by the simpler hardware being much faster

x86

Intel x86 processors are example of CISC

THE UNIVERSITY of ADELAIDE

The Hack computer

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
- 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.

Summary

- You can construct many gates from NAND – this is just one example of how gates are built up.
- By understanding arithmetic, we can combine gates to add two numbers, then combine full-adders to add larger numbers.
- DFFs allow us to store state by delaying output. This allows us to build registers.

This Week

- Review Chapters 3 & 4 of the Text Book (if you haven’t already)
- Assignment 1 Due
- Start Assignment 2
- Prac Exam Next Week during Workshops
  - Practice Exam will be available before tomorrow
- Review Chapter 5 of the Text Book before next week.