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

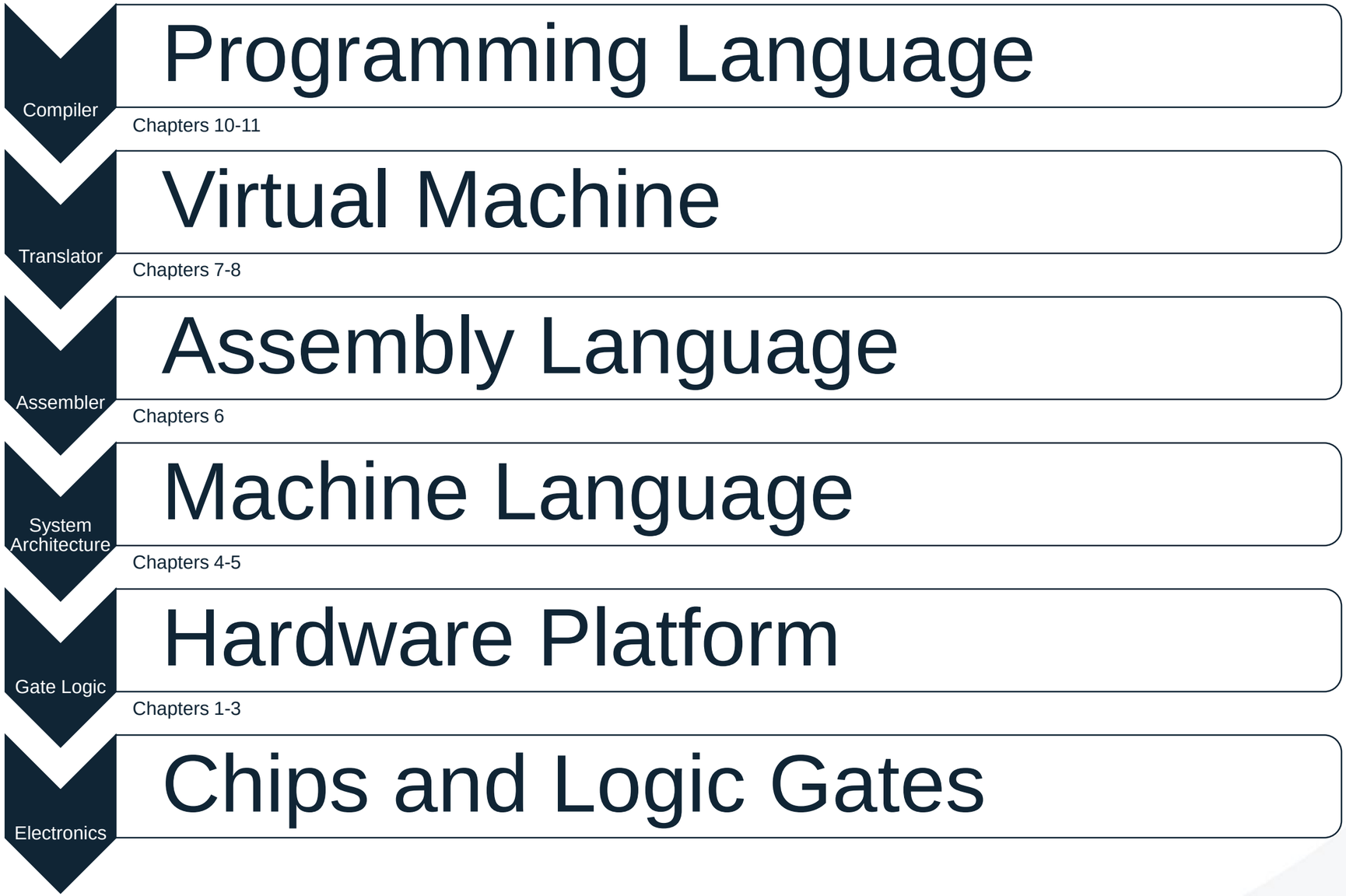
We acknowledge the deep feelings of attachment and relationship of the Kaurna people to country and we respect and value their past, present and ongoing connection to the land and cultural beliefs.

THE UNIVERSITY of ADELAIDE

Computer Systems

Lecture 11: Signals and Caching

Review: The whole system



THE UNIVERSITY of ADELAIDE

Signals & Caching

THE UNIVERSITY of ADELAIDE

The Hack Computer

The HACK machine is very simple

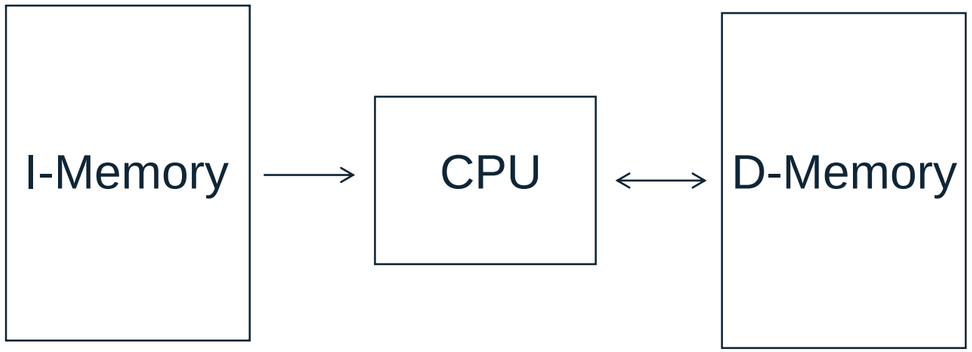
it has 64KB of separate read only instruction memory

it has 48KB + 2B of separate read / write data memory

the entire machine runs off a **single clock signal**

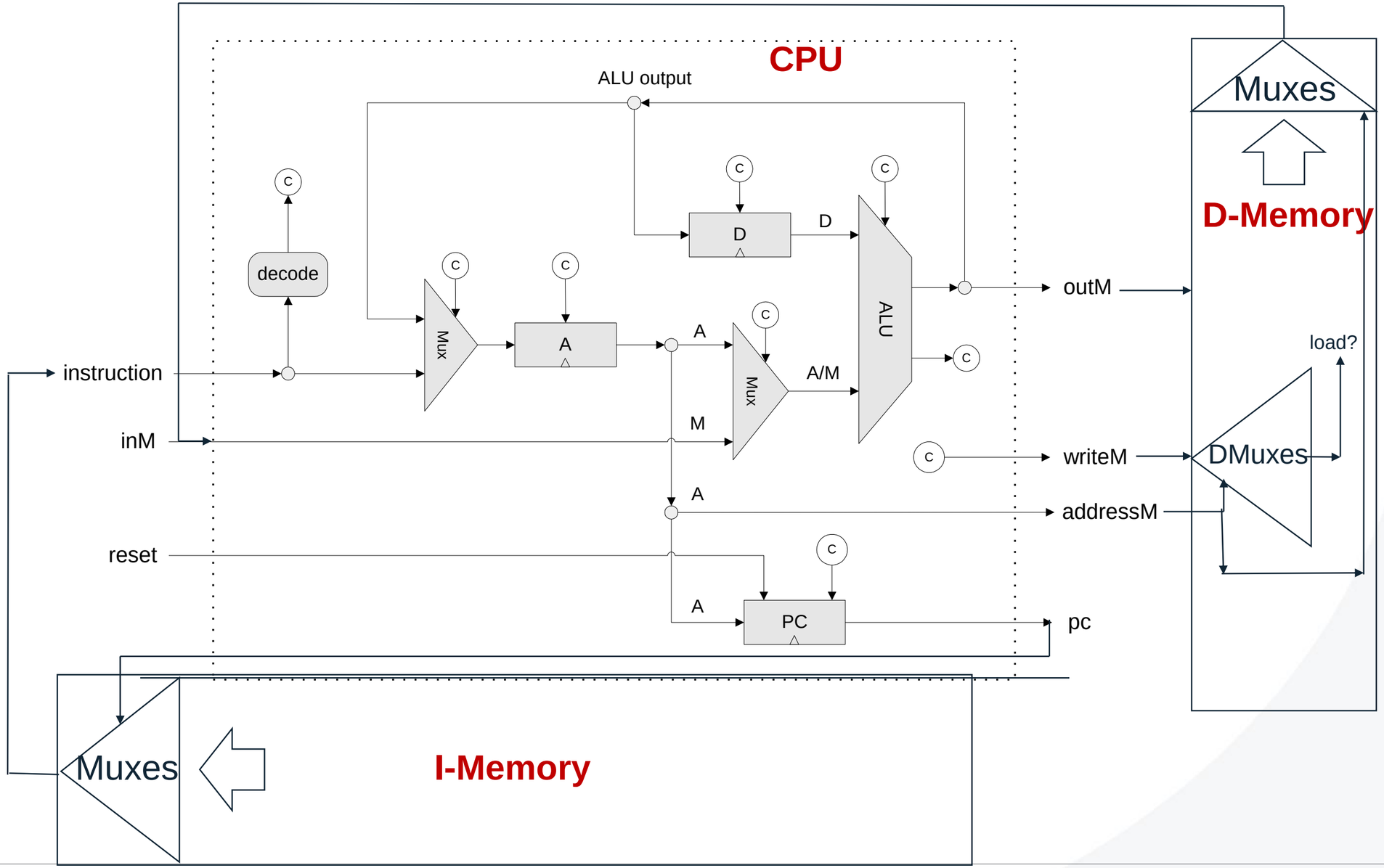
all instructions take exactly the same amount of time to execute

a lot of embedded systems may be structured like this

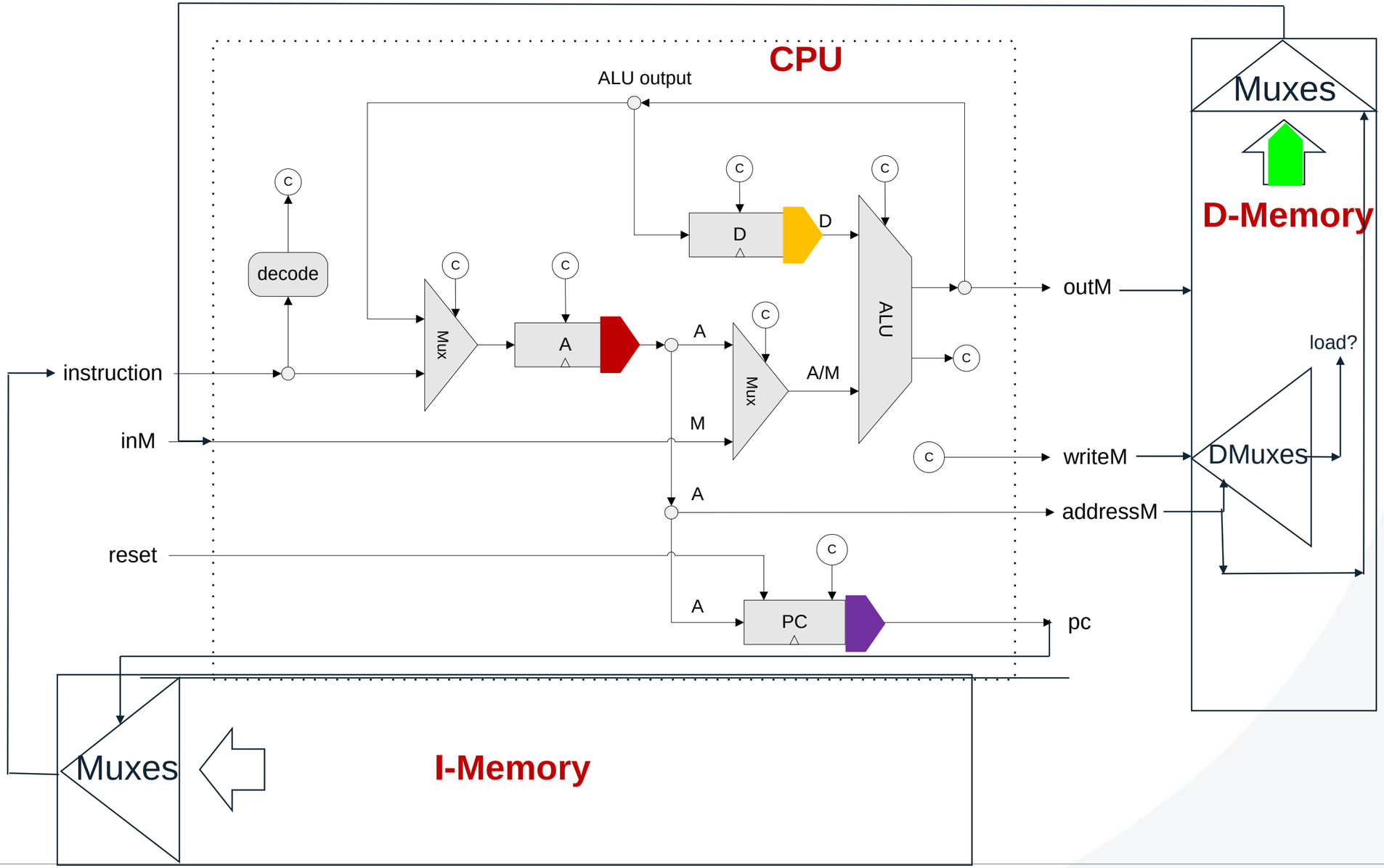


THE UNIVERSITY of ADELAIDE

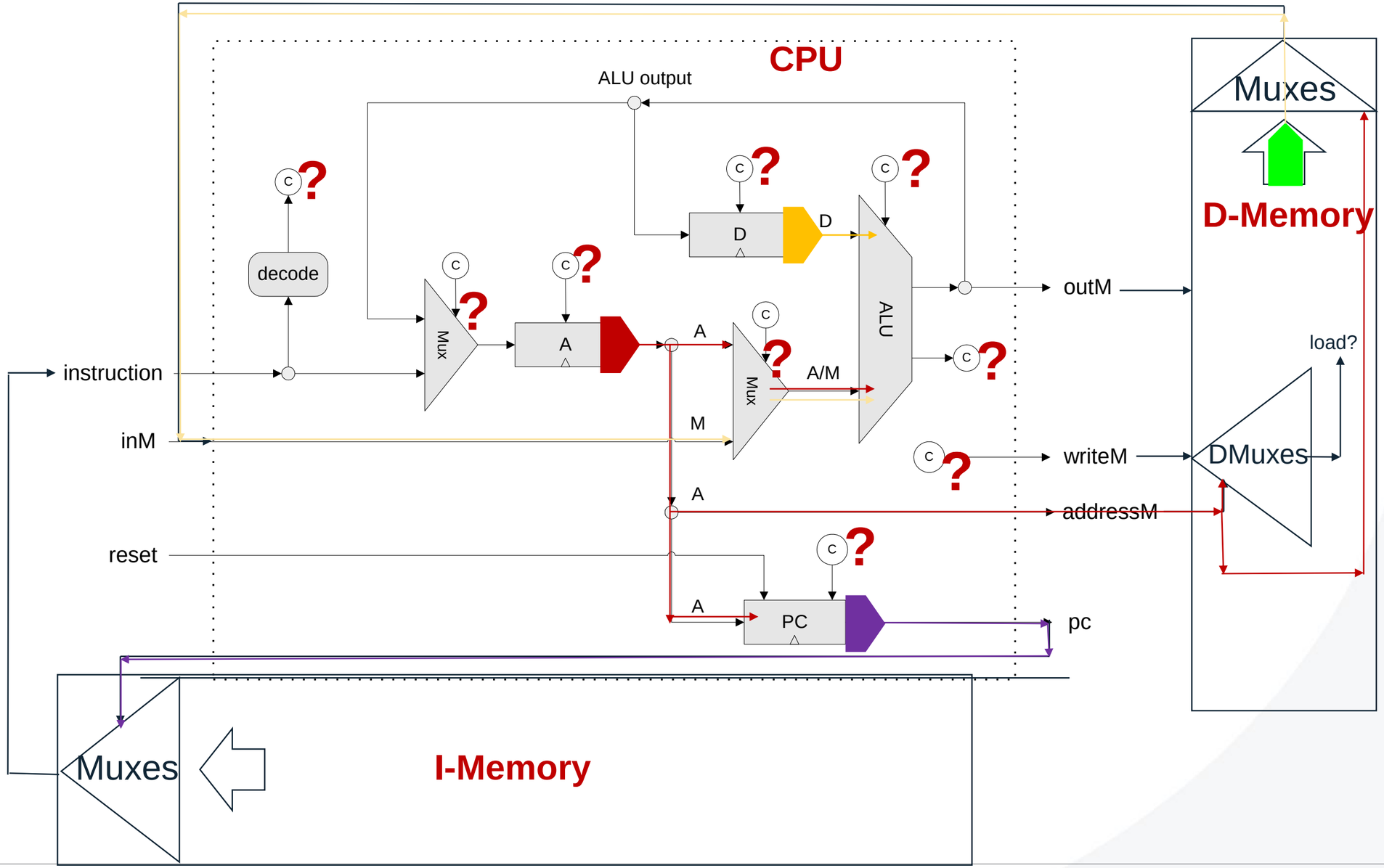
The Hack Computer



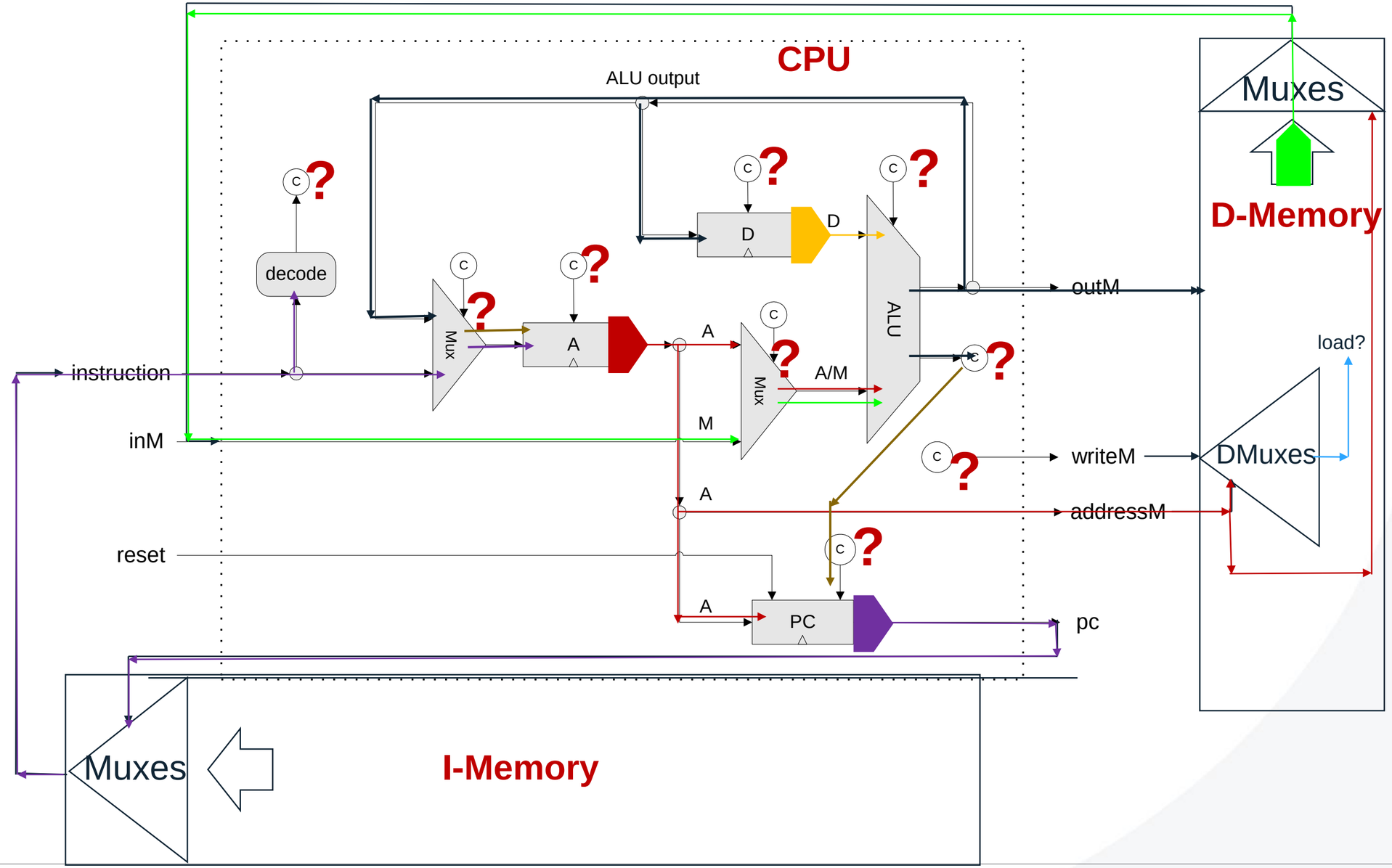
Signal Propagation – Clock Tick



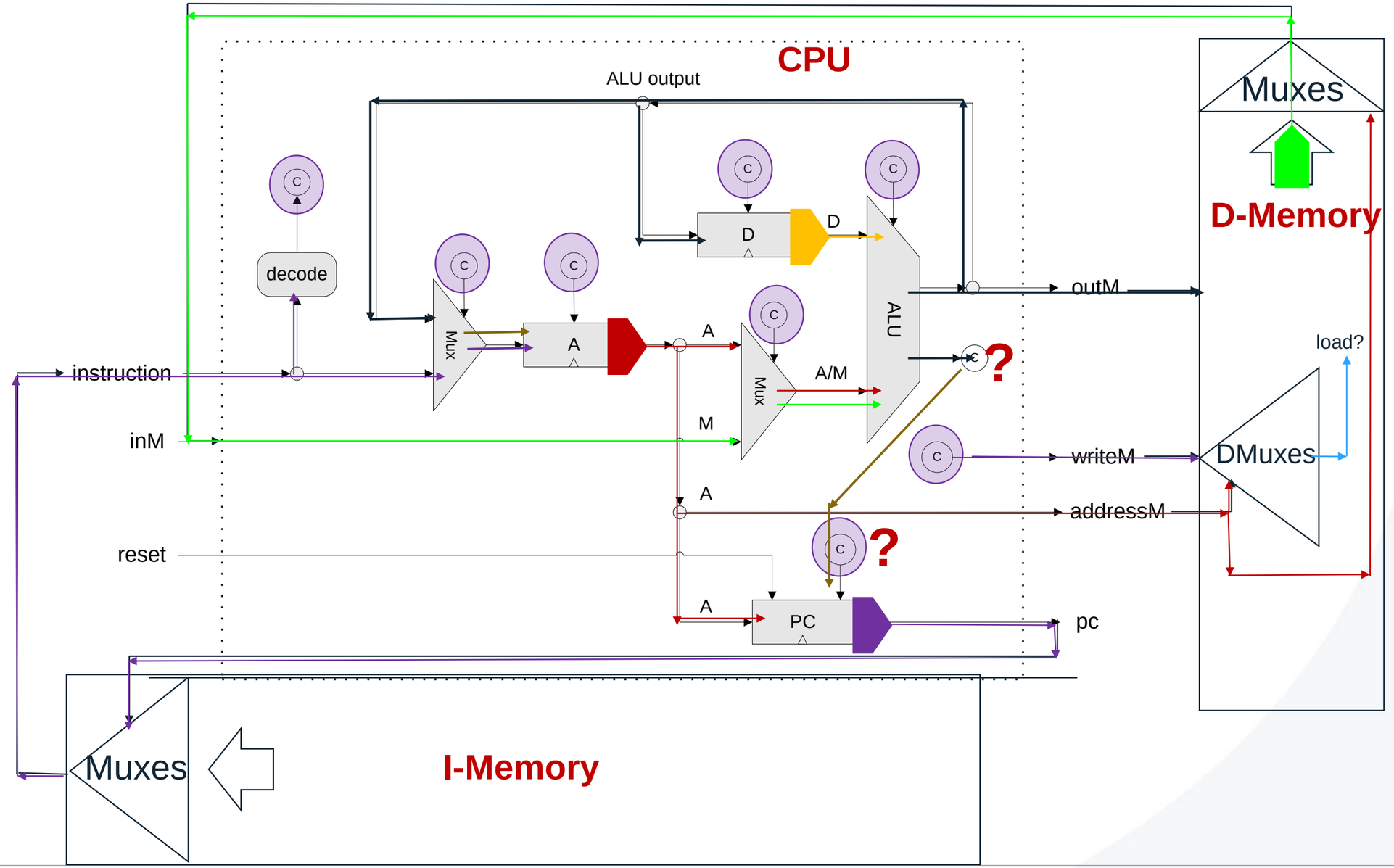
Signal Propagation – Signals Flow



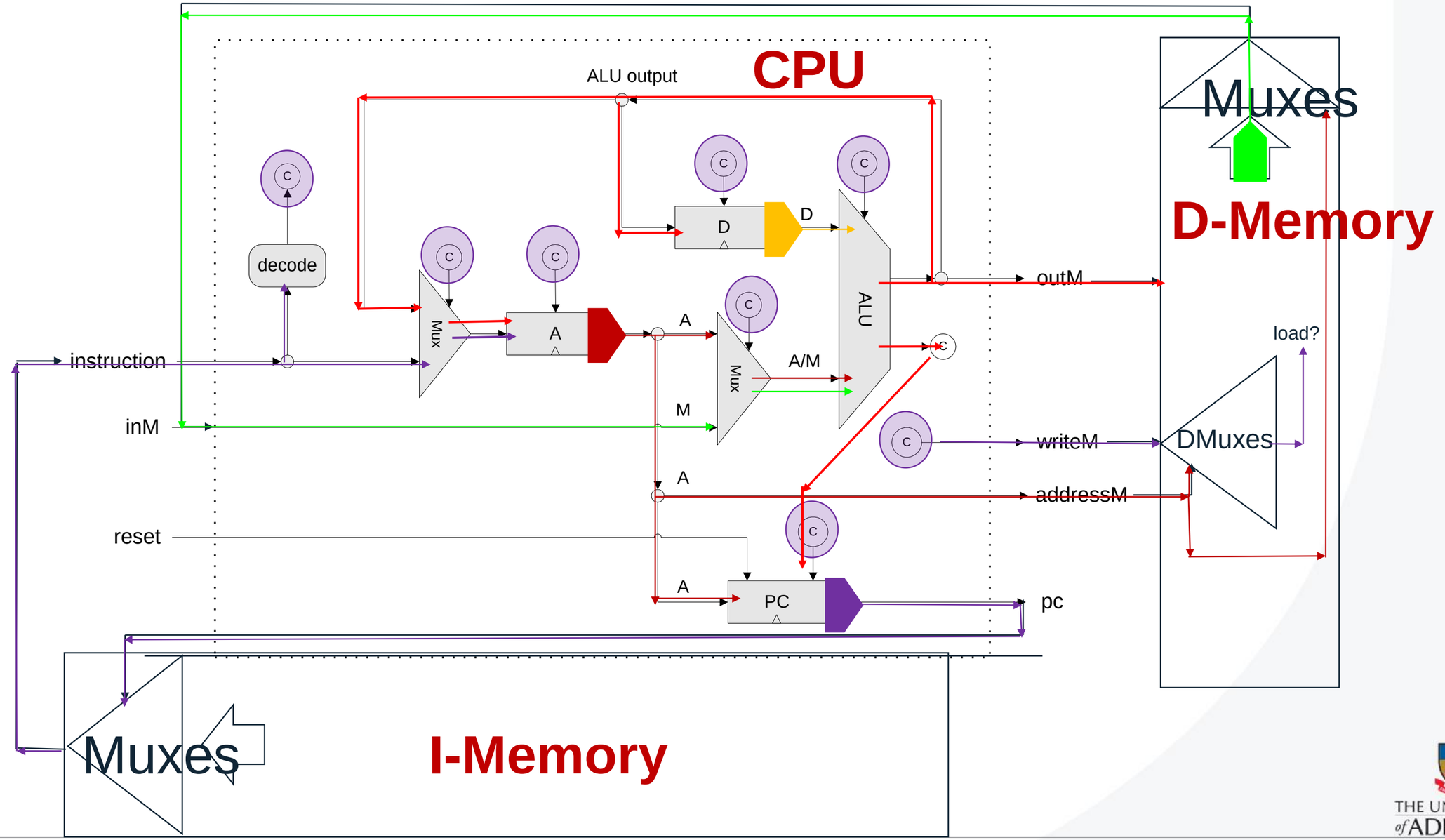
Signal Propagation – Signals Flow



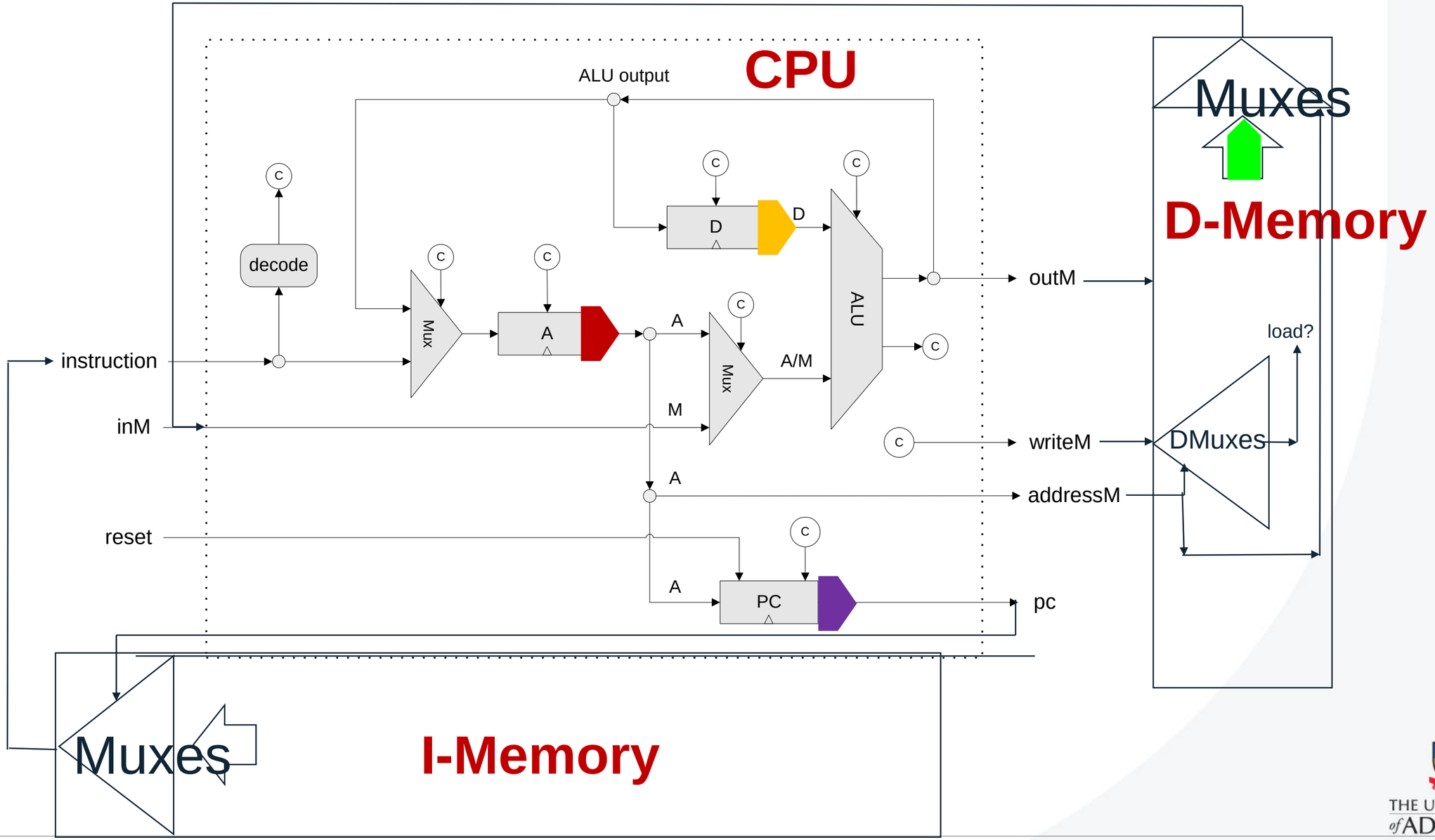
Signal Propagation – Signals Flow



Signal Propagation – Signals Flow



Signal Propagation – Clock Tick



Signal Propagation

The clock must not tick again until propagation completes

the output of every gate needs time to get to the next gate

every gate needs time to respond to its inputs

the longest path through the machine must be able to complete

The longest path is very long

the PC must send a new address to instruction memory

only then can a new instruction be used to change control signals

only then can the correct inputs reach the ALU

only then can the correct output leave the ALU

only then can a Jump be evaluated

only then is the new value of the PC known

THE UNIVERSITY of ADELAIDE

Signal Propagation

Until propagation completes bad things appear to happen

wrong data being read from memory

wrong data being written to the wrong memory address

the wrong values flowing through muxes

the wrong address being written to the PC

the wrong values being written to the A and D registers

Why does the computer still work?

because the wrong values are never saved

Memory, PC, A and D do not change until the next clock tick

The Power Wall

Switching consumes power

if the outputs of chips change power is consumed

even if externally there is no change, internal changes can occur

careful routing of wires to minimise changes can save power

The overall power consumption of a processor approximates

Capacitive Load x Voltage^2 x Frequency

Power Consumption Performance Challenges

There is a practical limit to how much power can be used

Capacitive Load reduction requires new manufacturing processes

Voltage reductions may be nearing their limits

Removing the generated heat can be expensive

We want lower power consumption

THE UNIVERSITY of ADELAIDE

What Can Be Done?

Shorten the longest path

more efficient adders

- our 16-bit adder requires a signal to traverse 32 gates
- carry look ahead could reduce this to 5 gates
- a 16-bit multiplier could be implemented with a 20 gate delay

a shorter longest path may allow a shorter clock cycle / faster clock

but a faster clock means more power consumption ?

Split the processor into smaller parts linked by registers

- pipelining

Use separate clocks for different components

- processor*
- memory*
- I/O devices

Slow Memory

Assume that the RAM in our Hack computer is changed

the new memory takes **500** clock cycles to read or write anything

it is not good at remembering data but it is very cheap !!!

Assume that the RAM has a ready signal

it is cleared when addressM changes or writeM changes

it is set when the input has been saved and / or the output is correct

How do we stop the Hack processor while we wait?

we could **and** the ready signal with the load signal for A, D and PC

if these registers cannot change the Hack processor is stalled

Instructions using RAM are now 500 times slower

other instructions still complete in a single clock cycle

instruction using RAM are stalled for **500** clock cycles

but our Hack computer still works … sort of … Pong is too slow …

THE UNIVERSITY of ADELAIDE

Very Slow Memory in Real Life?

The HACK machine is very simple

it has separate instruction memory and data memory

the entire machine runs off a **single clock signal**

all instructions take exactly the same amount of time to execute

The usual Von-Neumann architecture is more complex

there is only one memory

all instructions and data live in the same memory

reading instructions interferes with reading / writing data

real memories can be very large but much slower than the CPU

- taking anywhere from 2 to more than **500** clock cycles to access

How do we make a computer fast when memory is slow?

THE UNIVERSITY of ADELAIDE

Memory Hierarchies

The usual Von-Neumann architecture is more complex

there is only one memory

all instructions and data live in the same memory

reading instructions interferes with reading / writing data

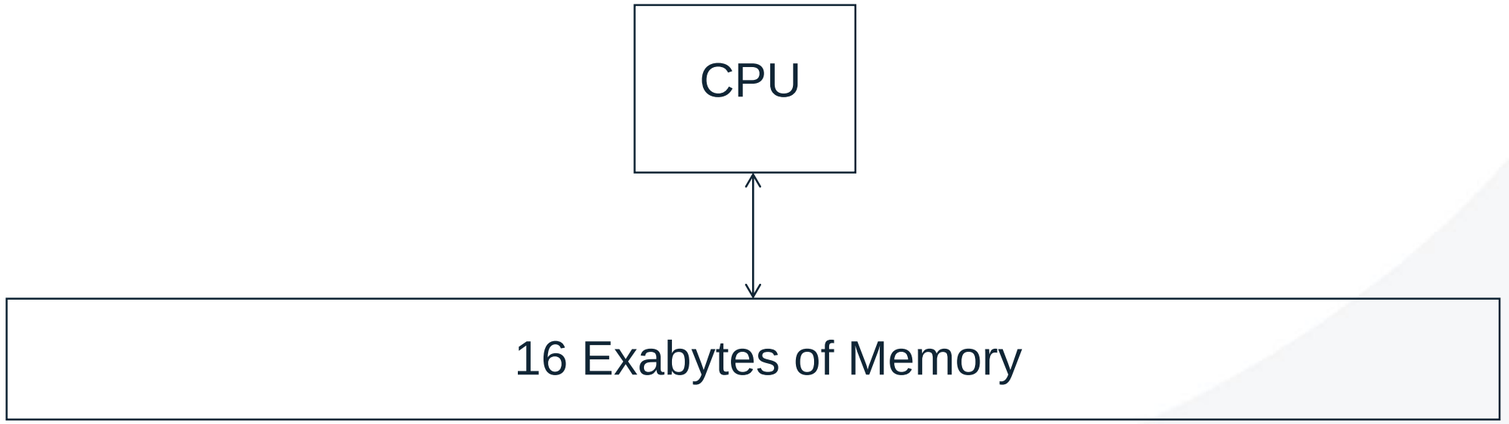
real memories can be very large but much slower than the CPU

- taking anywhere from 2 to more than 500 clock cycles to access

how do we make this work ?

- keep local copies of what is being used ?

What the programmer sees on a modern CPU:



THE UNIVERSITY of ADELAIDE

Memory Caching – Hardware Implemented

A memory cache

it may appear to behave like memory from the outside

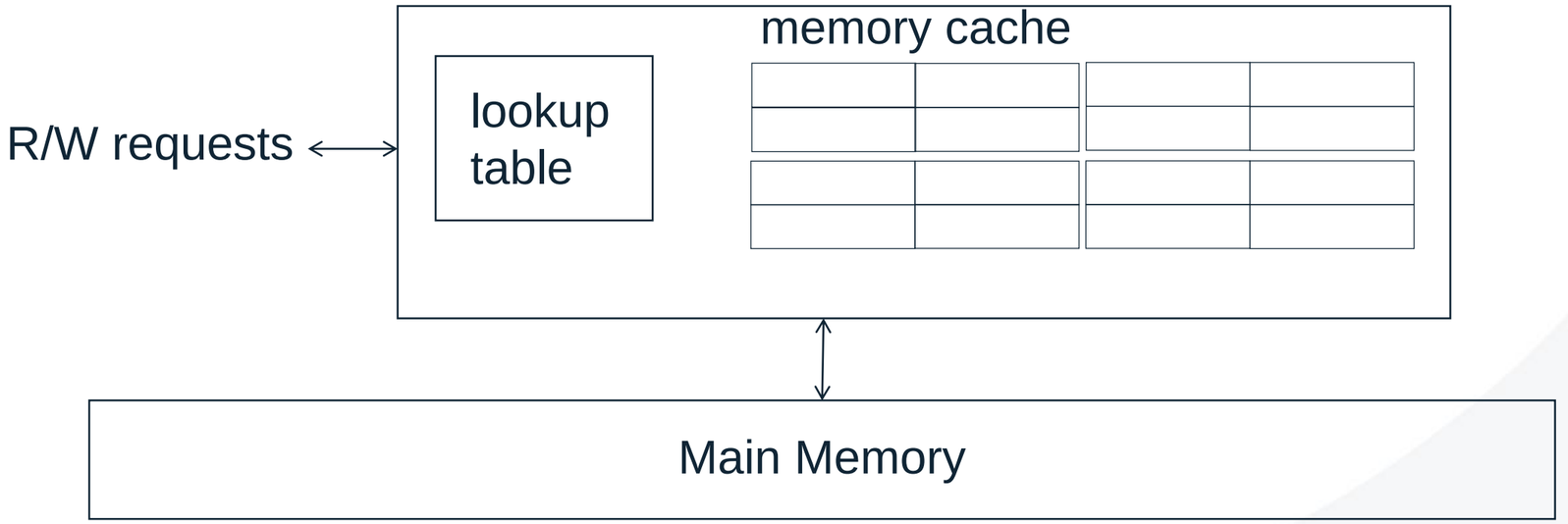
internally it keeps copies of blocks of memory that were accessed

it has a lookup table recording the addresses of the copies

access to data in the cache is very fast

access to data not in the cache must wait for the memory

access to data not in the cache may be slower than without one



Memory Caching – Hardware Implemented

Instruction Caching

when an instruction is required, copy it and others around it

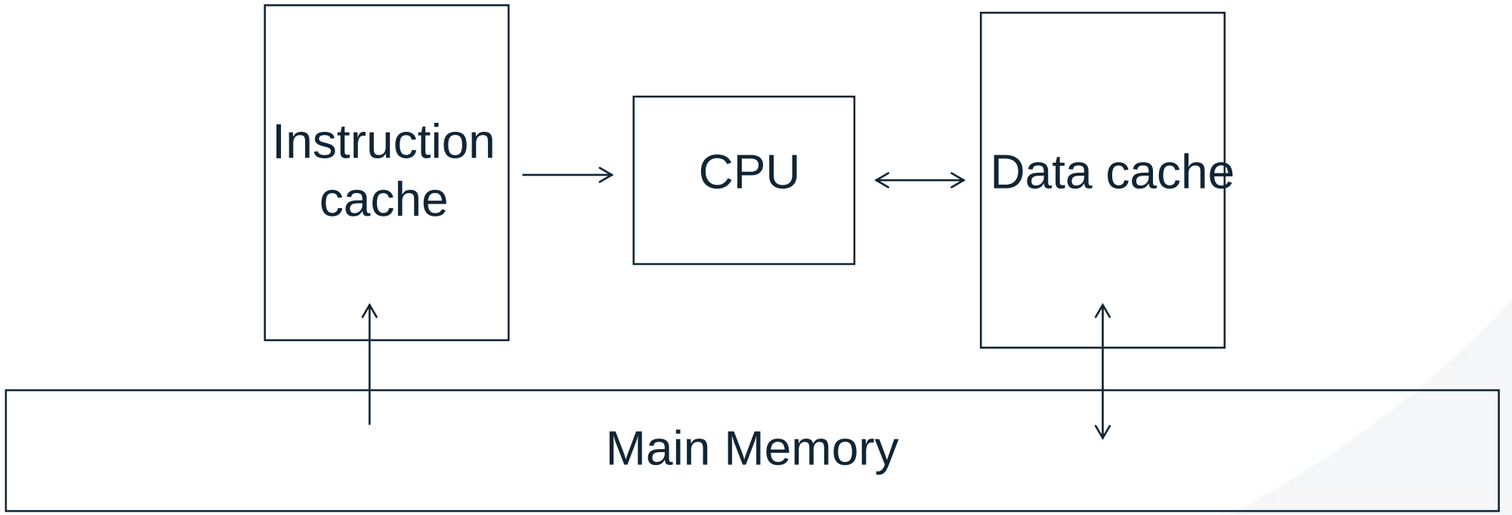
instructions already in the cache can be read very quickly

Data Caching

when data is required, copy it and data around it

data already in the cache can be read very quickly

when data is written, keep a copy of it*



Why Instruction Caching Usually Works

Instruction Caching

instructions do not change

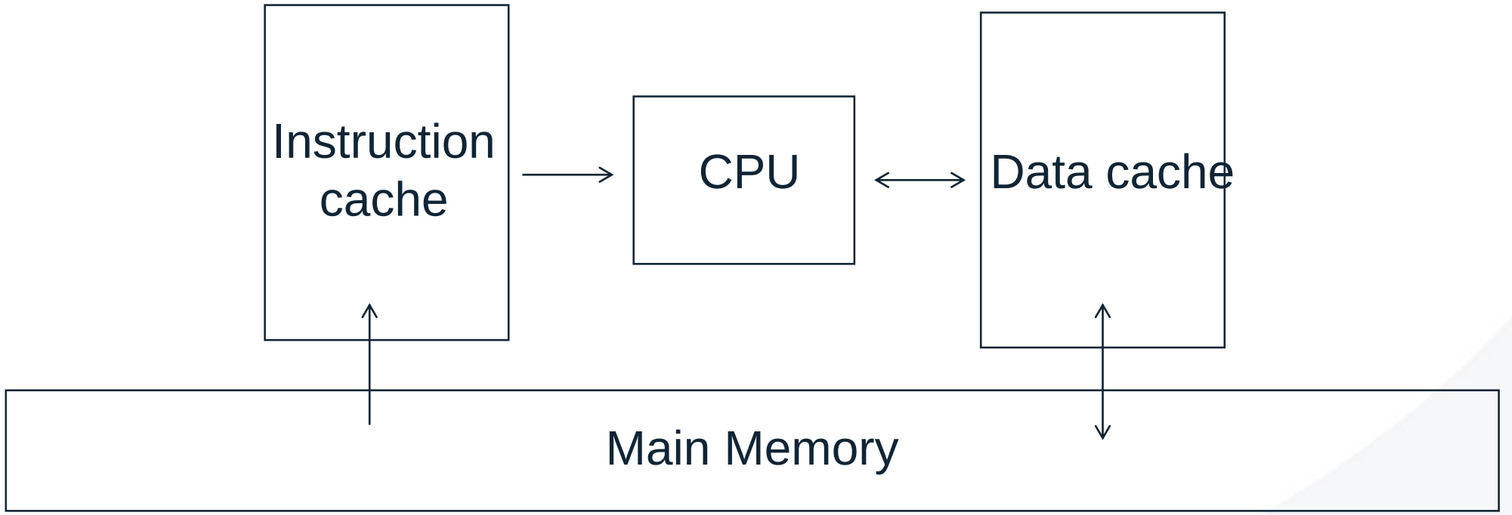
the same instructions are used over and over again

loops and frequently called functions benefit most

if the cache is full you can safely overwrite anything you like

the wrong choices can have a catastrophic impact on performance

the choices are implemented in hardware, change may be impossible



Why Data Caching Sometimes Works

Data Caching

caching data is a more complex problem

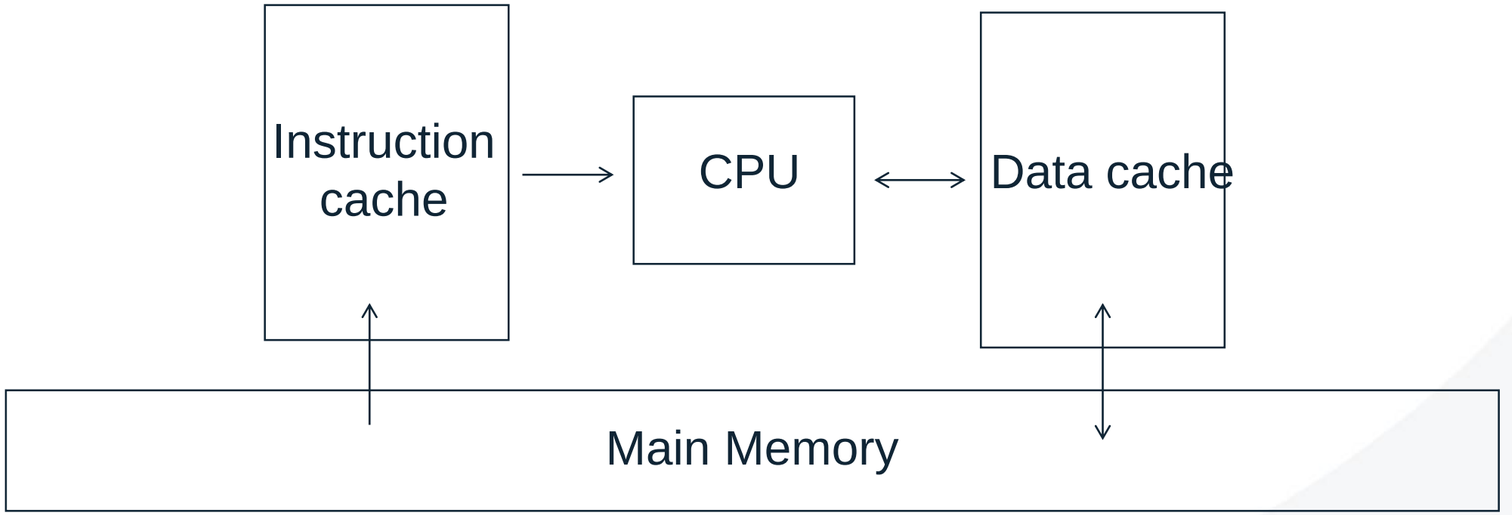
if data is frequently accessed you want it to stay in the cache

if the cache is full what do you overwrite?

you cannot overwrite new data that has not been written to memory

the wrong choices can have a catastrophic impact on performance

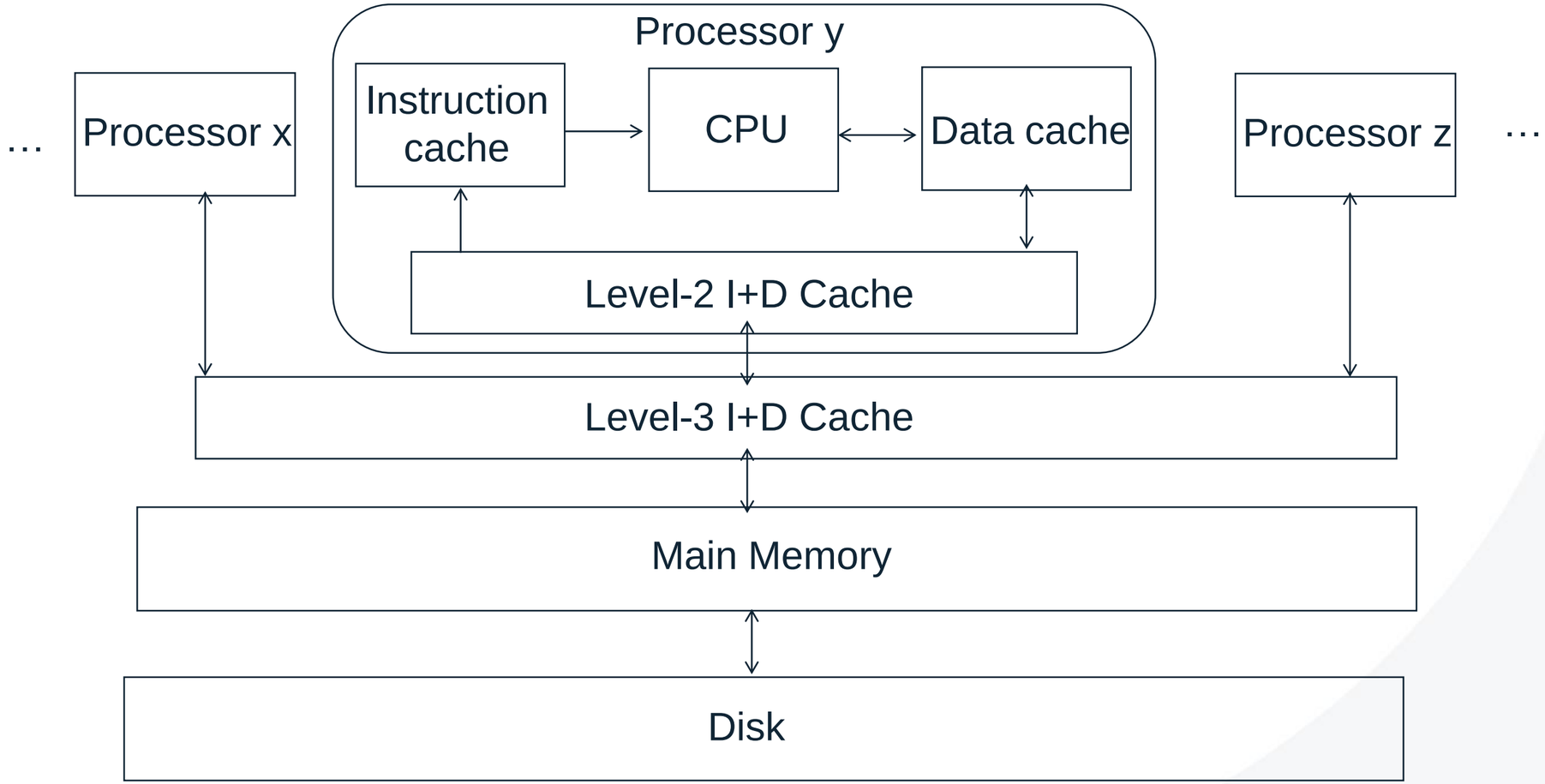
the choices are implemented in hardware, change may be impossible



The Memory Hierarchy

There can be many levels of cache

every cache has a table recording it contents



Memory Hierarchies

The caches are mostly hidden from the programmer

in some cases virtual memory can be manipulated directly

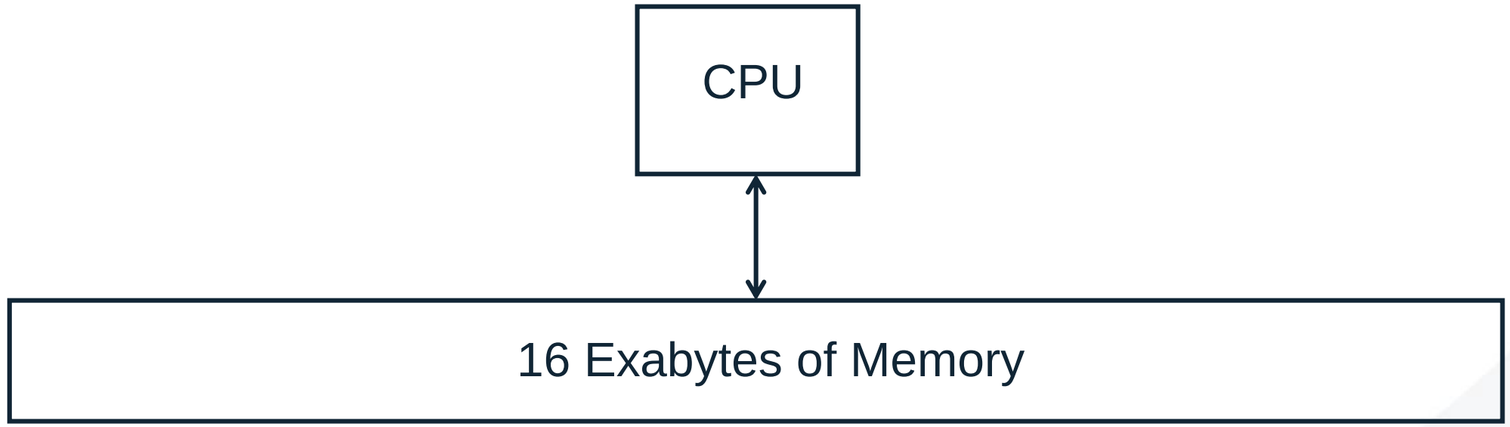
memory mapped files are a good example

virtual memory is partly implemented in software

hardware exceptions run page fault handlers to move data

caches work to hide themselves from the programmer

What the programmer sees on a modern CPU



Real World Effects

Instruction Caching

consider a simple interpreter for a virtual machine

a loop is responsible for reading the next instruction from memory

it then looks up the address of a function to simulate the instruction

it calls the function

on return it goes around the loop again

I tried to make one of these really fast

I made gcc put the simulated program counter in a physical register

I had optimized the loop reducing it from 28 to 4 instructions

What was the speed difference?

5 to 1, 7 to 1, 10 to 1, 20 to 1, 100 to 1, 500 to 1, … ?

Can you explain why?

Real World Effects

Data Caching

consider a simple loop that reads all the elements in an array

can you tell how long it will take to run?

```
// sum length elements of data in the order specified by indices
int iterate(int *data,int *indices,int length)
{
    int x ;
    for ( int i = 0 ; i < length ; i++ )
    {
        x += data[indices[i]] ;
    }
    return x ;
}
```

Can the time taken vary?

if yes, why and by how much?

if no, why not?

Real World Effects

How can you avoid or minimise these problems?

- buy a more expensive processor with a more intelligent cache
- try to keep data close together
- try to avoid unnecessary random access to data
- try to take advantage of temporal and spatial locality

Temporal Locality

- instructions and data tend to get used again in the near future
- caches that replace the least recently used content may work well

Spatial Locality

- instructions and data tend to be near something that was just used
- caches usually copy an enclosing block of instructions or data
- cache line size can vary a lot, the best size depends on the program

Summary

Caching is used just about everywhere in a computer

Instruction caching is usually very effective

- instructions do not change* during execution
- most instructions get used over and over again

Data caching can be very effective

- data changes during execution
- when data is accessed, it may well be used again soon
- when data is accessed, near by data may well be used soon

Virtual Memory (VM)

- main memory can be a cache for a much larger virtual memory
- every program can view its entire 16 exabytes of VM
- it works so long as the programs do not use too much of it
- physical resources still need to be available


THE UNIVERSITY
of ADELAIDE

This Week

- Review Chapter 11 of the Text Book (if you haven’t already)
- Prac 6 Available, due end of semester
- Prac 7 (second part of prac 6) available this week.