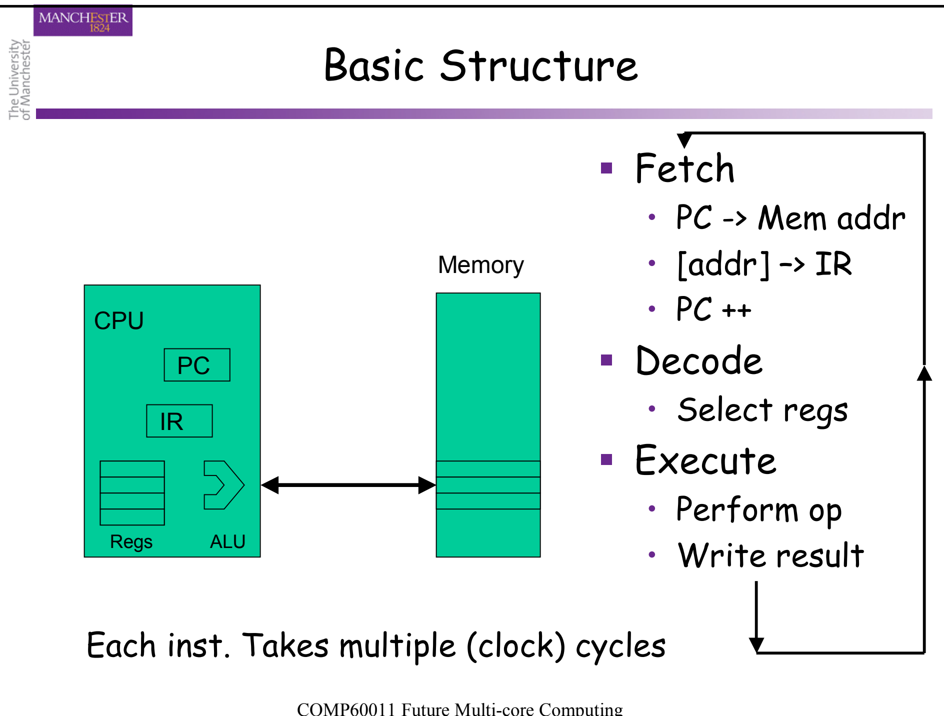
The University of Manchester
MANCHESTER 1824

A Brief Review of Processor Architecture

Why are Modern Processors so Complicated?

The University of Manchester
MANCHESTER 1824

Basic Structure

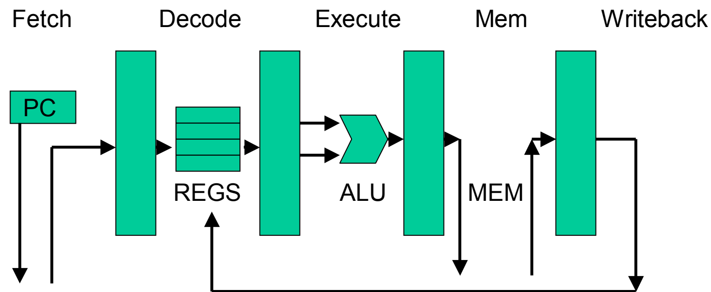


- Fetch
  - PC -> Mem addr
  - [addr] -> IR
  - PC ++
- Decode
  - Select regs
- Execute
  - Perform op
  - Write result

Each inst. Takes multiple (clock) cycles

MANCHESTER 1824
The University of Manchester

Simple Pipeline

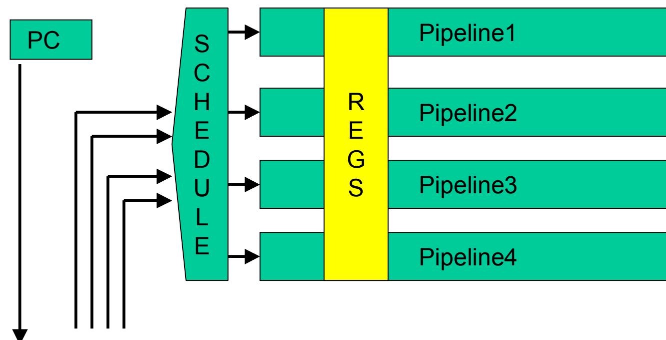


	Clock Cycle						
	1	2	3	4	5	6	7
Inst a	IF	ID	EX	MEM	WB		
Inst b		IF	ID	EX	MEM	WB	
Inst c			IF	ID	EX	MEM	WB
Inst d				IF	ID	EX	MEM
Inst e					IF	ID	EX

MANCHESTER 1824
The University of Manchester

Superscalar

- Up to 4 instructions in parallel



- 4x Speedup? Maybe 2.5
- 8 pipelines not worth it?

Superscalar Complexity

- Data dependencies
- Enough parallelism in program?
- Cache misses
- Cache contention
- Out of order execution?
- Register renaming?
- Precise exceptions
- Hardware complexity
- Clock distribution

Further Reading

- Computer Architecture: A Quantitative Approach, 4th Edition
- John L. Hennessy, David A. Patterson
- Morgan Kaufmann
- ISBN-13: 978-0123704900
- Chapters 1 & 2

MANCHESTER 1824

The University of Manchester

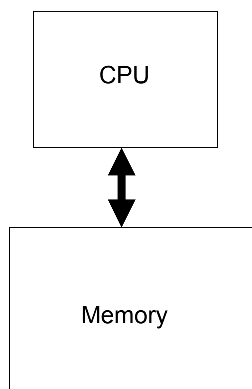
Caches

What are they?
How do they work?
What are the problems?

MANCHESTER 1824

The University of Manchester

Basic CPU Operation



- Random Access Memory (RAM)
- Indexed by address
- CPU Program Counter (PC) fetches program instructions from successive addresses (+branches)
- Instructions (may) address memory – read or write
- CPU performs operations (ADD, TEST, BRANCH etc.)

MANCHESTER 1824

The University of Manchester

Why Cache Memory?

- Modern processor speed > 1GHz
- 1 instruction / nsec (10^{-9} sec)
- Every instruction needs to be fetched from memory.
- Many instructions (1 in 3?) also access memory to read or write data.
- But RAM memory access time typically 50 nsec! (67 x too slow!)

MANCHESTER 1824

The University of Manchester

What is Cache Memory?

- Dictionary – Cache: "A secret hiding place"
- Small amount of very fast memory used as temporary store for frequently used memory locations (both instructions and data)
- Relies on fact (not always true) that, at any point in time, a program uses only a small subset (working set) of its instructions and data.

MANCHESTER 1824
The University of Manchester

Facts about memory speeds

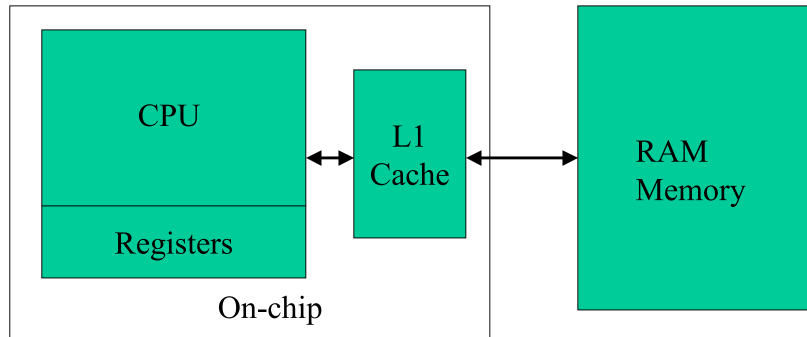
- Circuit capacitance is the thing that makes things slow (needs charging)
- Bigger things have bigger capacitance
- So large memories are slow
- Dynamic memories (storing data on capacitance) are slower than static memories (bistable circuits)

MANCHESTER 1824
The University of Manchester

Interconnection Speeds

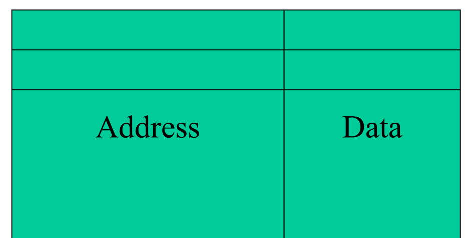
- External wires also have significant capacitance.
- Driving signals between chips needs special high power interface circuits.
- Things within a VLSI 'chip' are fast – anything 'off chip' is slow.
- Put everything on a single chip? Maybe one day! Manufacturing limitations

Basic Level 1 (L1) Cache Usage



Compiler makes best use of registers – they are the fastest.
Anything not in registers – must go (logically) to memory.
But is there a copy in cache?

Fully Associative Cache (1)



Memory Address
From CPU

Data to/from
CPU

Cache is small (so fast) (16 kbytes?)
Can only hold a few memory values

Memory stores both addresses and data

Hardware compares
Input address with
All stored addresses
(in parallel)

The University of Manchester MANCHESTER 1824

Fully Associative Cache (2)

- If address is found – this is a 'cache hit' – data is read (or written)
- If address is not found – this is a 'cache miss' – must go to main RAM memory
- But how does data get into the cache? If we have to go to main memory, should we just leave the cache as it was or do something with the new data?

The University of Manchester MANCHESTER 1824

Locality

Caches rely on locality

- Temporal Locality – things when used will be used again soon – e.g. instructions and data in loops.
- Spatial locality – things close together (adjacent addresses) in store are often used together. (e.g. instructions & arrays)

MANCHESTER 1824

The University of Manchester

What to do on a cache miss

- Temporal locality says it is a good idea to keep recently used data in the cache.
- Assume a store read for the moment, as well as using the data:
  - Put newly read value into the cache
  - But cache may be already full
  - Need to choose a location to reject (replacement policy)

MANCHESTER 1824

The University of Manchester

Cache replacement policy

- Least Recently Used (LRU) – makes sense but hard to implement (in hardware)
- Round Robin (or cyclic) - cycle round locations – least recently fetched from memory
- Random – easy to implement – not as bad as it might seem.

MANCHESTER 1824

The University of Manchester

Cache Write Strategy (1)

- Writes are slightly more complex than reads
- We always try to write to cache - address may or may not exist there
- Cache hit
  - Update value in cache
  - But what about value in RAM?
  - If we write to RAM every time it will be slow
  - Does RAM need updating every time?

MANCHESTER 1824

The University of Manchester

Cache Write Strategy (2)

- Write Through
  - Every cache write is also done to memory - slow
- Write Through with buffers
  - Write through is buffered i.e processor doesn't wait for it to finish (but multiple writes could back up)
- Copy Back
  - Write is only done to cache (mark as 'dirty')
  - RAM is updated when dirty cache entry is replaced (or cache is flushed e.g. on process switch)

The University of Manchester
MANCHESTER 1824

Cache Write Strategy (3)

- Cache miss on write
  - Write Around
    - just write to RAM
    - Subsequent read will cache it if necessary
  - Write Allocate
    - Assign cache location and write value
    - May need to reject existing entry
    - Write through back to RAM
    - Or rely on copy back later

The University of Manchester
MANCHESTER 1824

Cache Write Strategy (4)

- Fastest is Write Allocate / Copy Back – most often used
- But cache & memory are not 'coherent' (i.e. can have different values)
- Does this matter?
  - Other things accessing the same memory?
  - Autonomous I/O devices
  - Multi-processors
- Needs special handling

The University of Manchester MANCHESTER 1824

The Multi-Core Coherence Problem

- So, with multiple cores, each with their own cache, we have a serious problem
- Each may be writing to its own cache and not updating shared memory
- Not a problem if programs on the individual cores do not share variables
- But whole purpose of a shared memory programming model is to communicate via shared memory