What does a Processor Do?

It carries out a series of simple instructions.

Stages:

1. Fetch the next instruction.
2. Decode it.
3. Fetch required data.
4. Execute the instruction.
5. Store any results.

Instructions stored as binary *op-codes*. Each type of processor will understand different instructions encoded in different ways.

Everything happens in fixed length stages known as *cycles*.

```
Machine Code               Assembley Language
4E56 0000        longmul   link a6,#0
48E7 F880                  movem.l d0-d4/a0,-(a7)
202E 0010                  move.l 16(a6),d0
222E 000C                  move.l 12(a6),d1
206E 0008                  move.l 8(a6),a0
3600                       move.w d0,d3
C6C1                       mulu d1,d3
4840                       swap d0
3400                       move.w d0,d2
C4C1                       mulu d1,d2
4841                       swap d1
3801                       move.w d1,d4
C8C0                       mulu d0,d4
4840                       swap d0
C0C1                       mulu d1,d0
D082                       add.l d2,d0
6400 0008                  bcc mni
D8BC 0001 0000             add.l #$00010000,d4
4281             mni       clr.l d1
3200                       move.w d0,d1
4841                       swap d1
4240                       clr.w d0
4840                       swap d0
D681                       add.l d1,d3
D980                       addx.l d0,d4
2084                       move.l d4,(a0)
2143 0004                  move.l d3,4(a0)
4CDF 011F                  movem.l (a7)+,d0-d4/a0
4E5E                       unlk a6
4E75                       rts
```

Registers

Small number of storage slots within processor — very fast.

Initially small number of special use registers. (eg Accumulator, Index Register, Program Counter, Stack Pointer).

Modern processors may have many GPR (32 GPR, 32 FPU Registers).

Some processors provide sliding window of registers for passing data between functions.

Parts of a processor:

Integer Unit Or Fixed point unit. Can perform logical operations and integer arithmetic.

Floating Point Unit Can perform arithmetic on floats and doubles.

Instruction Decode Tells units what operations to perform based on op code received.

Branch Unit Deals with jumps in sequence of execution.

Memory Management Unit Memory protection, virtual memory,...

Either on chip or as co-processor. Possible co-processors: Vector unit, Graphics accelerator, $\frac{1}{r}$ processors.

CISC Processors

Complex Instruction Set Computers.

Why CISC? In 60’s and 70’s:

- memory sizes were small,
- compilers weren’t so good,
- coding in assembly was common,
- people like complex instructions,
- packing lots into an op code saved space.

In the 80s the 68k and 80x86 born into this world, sporting 100 or so types of instruction.

Features of CISC:

- Lots of nifty instructions.
- Lots of addressing modes.
- Microcoded instructions.
- Fun to write assembly for.

Down-sides:

- compilers dislike complex instructions,
- complicated to design,
- harder to squeeze onto one chip.

Why RISC?

Reduced Instruction Set Computers.

Not so much about reducing instruction set, more about simplifying design, so that:

- everything (including cache) could go on one chip,
- things were simple enough to apply speed up tricks,
- electronics are simple enough to up the clock rate.

Today the descendants of the 80s RISC processors often have quite elaborate instructions. RISC is now often applied to processors which have used the optimisations applied to the first RISC generation.

Features of RISC processors:

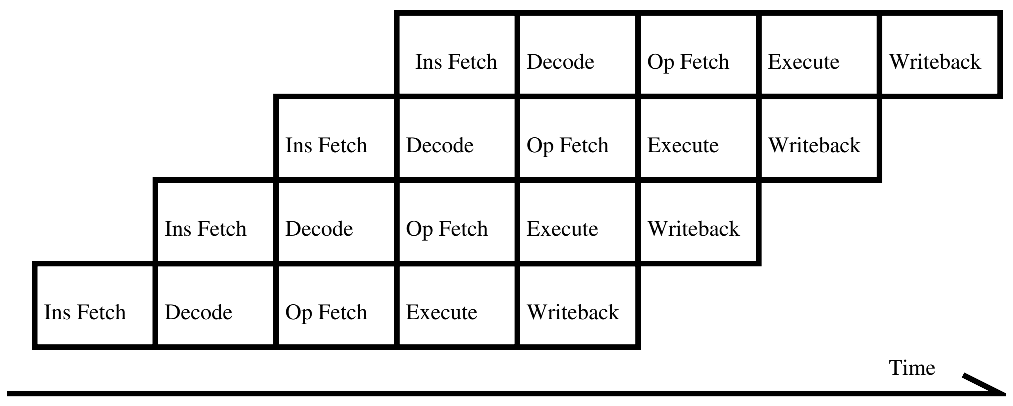
- Uniform Instruction Length,
- Load/Store architecture (simple addressing modes),
- Hardwired Instructions,
- Short cycle counts for instructions,
- Sophisticated compiler optimisations,
- Larger program sizes.

Examples of RISC processors today: Alpha, MIPS, SPARC, HP-PA, IBM POWER, PPC, ARM.

Various Optimisations:

- Pipelines.
- Delayed Branch.
- Super-scalar.
- Out-of-order execution.
- Speculative execution.
- Branch prediction.
- Predicated Execution.
- VLIW

Pipelines Overlap various stages of instructions.



			Ins Fetch	Decode	Op Fetch	Execute	Writeback	
		Ins Fetch	Decode	Op Fetch	Execute	Writeback		
	Ins Fetch	Decode	Op Fetch	Execute	Writeback			
Ins Fetch	Decode	Op Fetch	Execute	Writeback				

Pipeline *stalls* can be caused by unavailable data, data dependencies, slow instructions, branches.

Delayed Branch Used to avoid pipeline stalls after a branch. Next instruction is executed regardless of the direction the branch goes.

Super-scalar Multiple Fixed point, floating point and other units included allowing simultaneous execution of multiple instructions.

Out-of-order Execution Instructions my overtake one another if some are delayed, if there are no dependencies.

Speculative Execution Assume a branch goes one way and begin execution. Discard results if assumption was wrong.

Branch Prediction Remember which way this branch went before in order to make better guesses.

Predicated Instructions Avoid branches all together by having instructions which are only conditionally executed.

VLIW Very Long Instruction Word - multiple instructions rolled into one, aim to keep lots of units busy.

500MHz processor which can perform 3 instructions per cycle:
$32\text{b} * 3 * (1 + 2 + 1) * 500\text{MHz} = 192\text{Gb/s}$.

Options:

- Use very fast memory.
- Use very wide memory.
- Use *caches* of very fast memory.

As the first isn't practical mixtures of the second and third are commonly used.

Cache memory: £1000 per MB.
Normal Memory: £1 per MB.
Disk: £0.1 per MB.

Memory Hierarchy

Possible memory speeds for 500MHz processor:

	Size	Cycle Time
Registers	1kB	2ns
L1 Cache	8kB	2ns
L2 Cache	512kB	4ns
RAM	512MB	50ns
Disk	16GB	5ms

Registers internal to processor (and possibly L1). Caches made using SRAM, RAM using DRAM.

Actual rates of transfer my vary depending on latency and width.

A cache is divided into lines, which are loaded and stored from memory. A line is usually quite wide.

Each line will have certain metadata associated with it:

- The ’address tag’ of data being cached.
- Validity.
- Shared/exclusive.
- Clean/dirty.

Dirty bit used for Write Policy:

Write Through Data written down the hierarchy after each write.

Write Back Data is written out at leisure of cache.

Cache Organisation

Have to choose how memory will be mapped into cache.

Fully Associative Any line can cache data from any location in memory. Flexible, but costly and complicated.

Direct Mapped Any given location can only end up in one cache line. Simple, but subject to trashing.

Set Associative Lines grouped together into sets, each of which can store data from the same collection of locations in memory. If each set contains n lines we call it n*-way-associative*.

The cache must be cleared of old data as it fills. Policies include selection lines randomly or the least recently used line (LRU). These policies can interact badly with code.

Cache options:

- Cache size,
- n-way-associative,
- how many levels,
- line size,
- special instructions (prefetch, bypass or disable cache).

Example:

Consider how the effect of the following code on a cache of size 64 longs with lines of size 4 longs if the cache is fully associative, direct mapped or 2-way-associative.

```
int main()
{
    long a[128],b[64];

    for( i = 0; i < 128; i++ )
        a[i] = i*i;

    for( i = 0 ; i < 64; i++ )
        b[i] = a[i];
}
```

Speeding up Memory Access

Wide data bus Many memory modules in parrallel.

Interleaving Many memory modules accessed round-robin.

Addressing Tricks Conversation on bus simplified to save time. Eg: Fast Page Mode Ram (only present low order address bits), EDO RAM (overlapping of address and data conversation), SDRAM (bus is syncronus, less overhead).

Note that DRAM is subject to errors caused by ionising radiation. This leads to other modifications such as parity bits and use of ECC.

Overall effect

People often calculate effective memory access rates. Suppose 90% L1, 8% L2 and 2% Ram:

$0.9*2ns+0.08*4ns+0.02*50ns = 3.12ns$

Naturally you want code to achieve reasonable hit rates.

Moral: Access memory in as linear a fashion as possible.

Assignment 2

Write a program which calculates the dot product of two dynamically allocated vectors of variable size. You can produce the vectors randomly. Use clock(3) or getrusage(2) to time the execution of the dot product.

Plot a graph showing operations per second against the size of the vector for vectors of length 2^5 to 2^{20}. For smaller loops make sure the loop is run enough times.