

**Number Systems**

Decimal to Binary conversion

INTEGERS:

- Repeatedly divide by two and note down the remainders. Writing the remainders down (starting with the *last* division and working *up*) gives the binary result, e.g.

$2 / 243 = 121$  remainder 1  
 $2 / 121 = 60$  remainder 1  
 $2 / 60 = 30$  remainder 0  
 $2 / 30 = 15$  remainder 0  
 $2 / 15 = 7$  remainder 1  
 $2 / 7 = 3$  remainder 1  
 $2 / 3 = 1$  remainder 1  
 $2 / 1 = 0$  remainder 1.

Hence,  $(243)_{10} = (11110011)_2$

FRACTIONS:

- Repeatedly multiply by two (each time taking of the integer part, if any) and note down the integer part of the result. Write from the *top down*, e.g.,

$0.125 * 2 = 0.250$   
 $0.250 * 2 = 0.5$   
 $0.5 * 2 = 1.0$   
 $0 * 2 = 0$  – equals zero, so ignore this one

Hence,  $(0.125)_{10} = (001)_2$

Binary to decimal conversion

- Take sum of the products of each digit and it's column heading, e.g.,

<b>8</b>	<b>4</b>	<b>2</b>	<b>1</b>	.	1/2	1/4	1/8
1	1	0	1	.	1	0	1

$8*1 + 4*1 + 2*0 + 1*1 = 8 + 4 + 0 + 1 = 13$   
 $1/2 * 1 + 1/4 * 0 + 1/8 * 1 = 1/2 + 1/8 = 5/8 = 0.625$

Hence,  $(1101)_2 = (13.625)_{10}$

Hexadecimal to Binary

- Translate each digit into a four digit binary sequence.

Hex	Bin	Hex	Bin
0	0000	A	1010
1	0001	B	1011
2	0010	C	1100
3	0011	D	1101
4	0100	E	1110
5	0101	F	1111
6	0110		
7	0111		
8	1000		
9	1001		

Hence  $D3EF_H = (1101001111101111)_2$



Revision Notes

E.g., to represent 0110.01 (6.25). First express in Standard Index Form, i.e.  $1.1001 \times 2^2$ . Take off the integer part – this is implied – to leave **1001**. This is the mantissa (ultimately must be 23 bits long). The real exponent is 2. Hence, is biased exponent is  $2 + 127 = 129 = \mathbf{1000001}$ . Sign is positive, thus **0**. So, 6.25 in IEEE 754 Floating Point Representation is:

**010000001**100100000000000000000000

- 0 is defined by a mantissa of 0 and an exponent of 0
- + or – infinity is defined by a mantissa of 0 and exponent of 255
- Addition and Subtraction:
  - Align the significands
  - Add or subtract the significands (*including* the integer part)
  - Normalise result (i.e., shift significand and decrement or increment the exponent.

E.g.,  $6.75 - 6.25 = 0.5$

6.75 = 01000000110110000000000000000000

6.25 = 01000000110010000000000000000000

Significands (including integer part) = 1.1011 and 1.1001

1.1011

-1.1001

0.0010

Normalise to 1.0000 (shift three to the left) and therefore decrement exponent by 3.

So, result is 01111110100000000000000000000000

Check: Sign = 0  
 Real exponent = 01111110 = 126.  $126 - 127 = -1$   
 Mantissa = 1  
 Therefore =  $(1)_2 \times 2^{-1} = (0.1)_2 = (0.5)_{10}$  which is correct.

~

'von Neumann' Machine Concept & The 'Little Man' Computer

- Introduces the following concepts, now followed by every modern computer system:
  - Stored program
  - Main Memory contains both program *and* data
  - Arithmetic Logic Unit operates on binary data
  - Control Unit interprets and acts on instructions from the memory (the program)  
 Instruction cycle consists of a FETCH and an EXECUTE
- The 'Little Man' Computer is a model to help us understand the basic workings of a von Neumann computer. It consists of 100 mailboxes (the main memory), an input and output basket, a calculator (the ALU) and a Little Man (the CPU) with a counter (Instruction Register).
- Program consists of opcodes (from a given instruction set) and data, e.g. 2xx means subtract the contents of mailbox xx from the number currently in the calculator.
- Example program (using Instruction Set from notes):

Mailbox Address	Contents	Description
00	901	Copy an input to the calculator
01	399	Place the value of the calculator in mailbox 99
02	901	Copy an input to the calculator
03	199	Add the number in the calculator to the number in mailbox 99
04	902	Output the number in the calculator
05	000	Halt
99		Data

**Computer Systems Organisation**

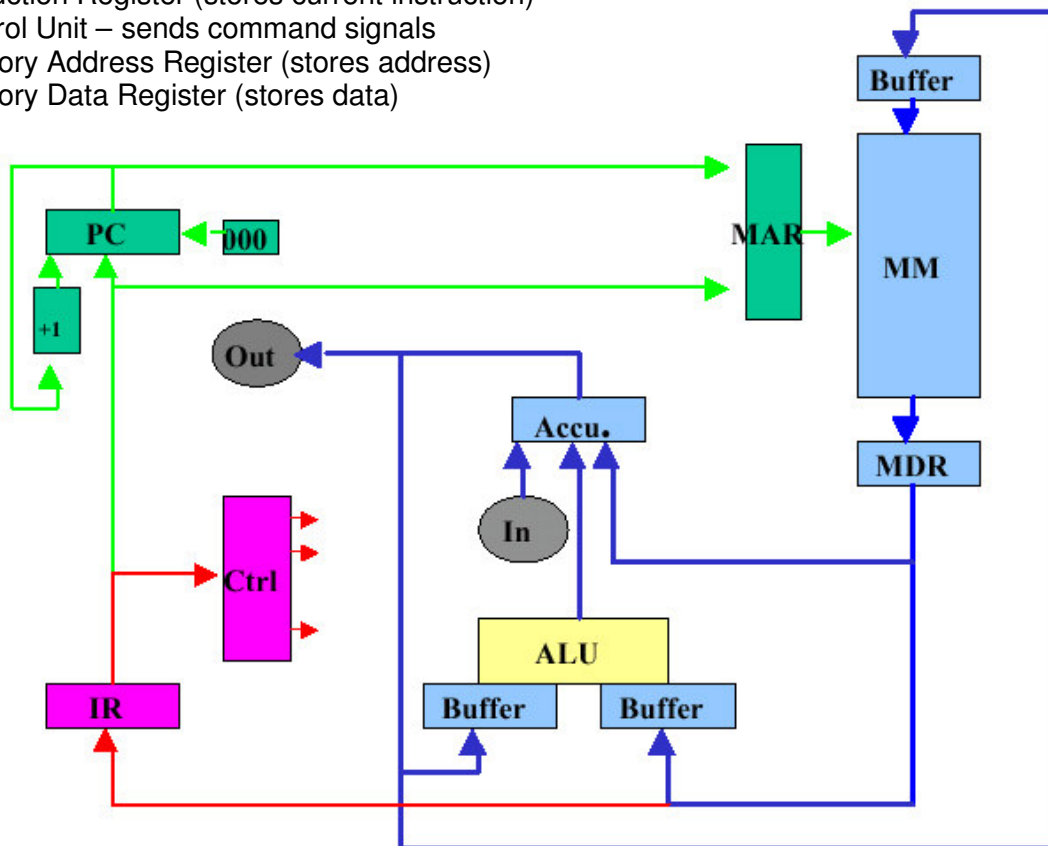
Bus Interconnection

- A bus is a communication path that that connects two or more devices.
- It is a shared transmission medium – a signal transmitted by one device is available for reception by all other devices; only one device can transmit at a time.
- Data Bus – carries data
- Address Bus – carries addresses
- Control Bus – carries control signals – command and timing, e.g. Memory R/W, I/O R/W, Bus request, Bus grant, etc.

The Central Processing Unit

Essentially consists of:

- Accumulator – temporary storage
- Arithmetic and Logic Unit – performs calculations
- Main Memory – stores program and data
- Program Counter (points to the location of the next instruction)
- Instruction Register (stores current instruction)
- Control Unit – sends command signals
- Memory Address Register (stores address)
- Memory Data Register (stores data)



Sequence of an operation:

- |                |   |
|----------------|---|
| <i>FETCH</i>   | PC to MAR<br>MM to MDR                                    |
| <i>EXECUTE</i> | MDR to IR<br>IR to MAR<br>MM to MDR<br>Operation<br>PC ++ |

Revision Notes

Memory

- 128 x 8 memory means 128 locations, each holding 8 bits.
- Total amount of memory limited by the size of the address bus.
- ROM – read only
- Programmable ROM – can be written using a high current device to burn out the wiring
- Erasable ROM – can be erased (using UV light) and re-programmed.
  
- R/W memory. Two types:
  - SRAM (static) – the memory contents, once written, do not need to be addressed or manipulated to hold their values. Fast and expensive. Used mainly for CPU registers.
  - DRAM (dynamic) – contents must occasionally be ‘refreshed’ in order to rewrite them. Slower – used for main memory.
  
- Sequence of events: Select address (address decoder) => Select R/W => Write/Read data => Enable/Disable memory location for further use.

Memory allocation is shown by a memory map.

I/O Module

From an internal point of view, I/O is functionally very similar to main memory.

- Two operations – R/W
- I/O module may control more than one device
- The interface to each *external* device is referred to as a port, each with a unique address.
- Can send interrupt signals to the CPU

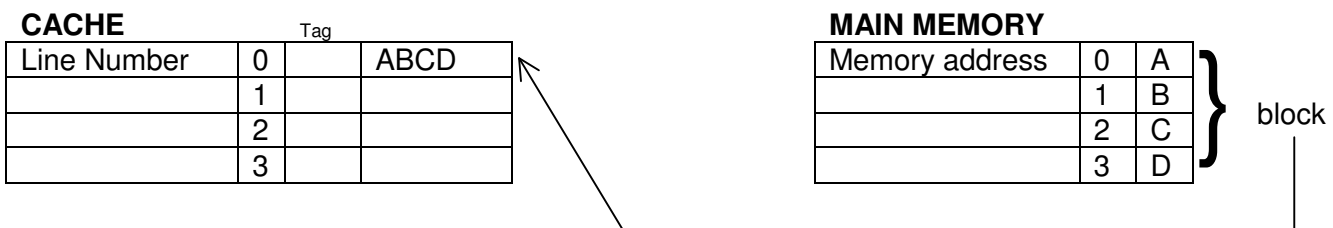
Data Exchange

- Memory to processor
- Processor to memory
- I/O to processor
- Processor to I/O
- I/O to and from memory (Direct Memory Access) – very slow, although it bypasses the processor, which can be getting on with other things

Memory Systems

Cache memory

- Small amount of fast (expensive) memory
- Sit between CPU and main memory, or may be located onboard the CPU
- Contains recently used information. Access to an item in the cache – *hit*; Access to an item not in the cache – *miss*.
- One line of cache holds one block of memory, e.g.,



## Revision Notes

- Fewer cache lines than main memory blocks
- Therefore, need to determine which memory block occupies which line.
- Also need an algorithm to map a memory block into the cache.

Three mapping techniques:

### Direct Mapping

- Each memory address contains three fields: tag, line and word.
- $2^{(\text{Total number of bits of address})} = \text{No. of address locations}$
- $2^{(\text{No. of bits in word field})} = \text{No. of words in one memory block or cache line}$   
E.g., cache line of 4 bytes.  $2^2 = 4$ , therefore 2 bit word field.
- $2^{(\text{No. of bits in line field})} = \text{No. of lines in cache}$   
E.g., 16K lines.  $2^{14} = 16K$ , therefore 14 bit line field
- Number of bits in tag field = total no. of bits – bits in line field – bits in word field
- Simple and inexpensive
- Fixed location for a given block, therefore if a program access two blocks that map to the same line repeatedly, cache misses are high.

### Associative Mapping

- A main memory block can load into any line of cache.
- Each memory address contains two fields: tag and word.
- $2^{(\text{Total number of bits of address})} = \text{No. of address locations}$
- $2^{(\text{No. of bits in word field})} = \text{No. of words in one memory block or cache line}$   
E.g., cache line of 4 bytes.  $2^2 = 4$ , therefore 2 bit word field.
- Number of bits in tag field = total no. of bits – bits in word field
- Every tag examined for a match, therefore cache searching is expensive.

### Set Associative Mapping

- Cache is divided into a number of sets each of an equal number of lines.
- A main memory block can load into any line of a given set in the cache.
- Each memory address contains two fields: tag, set and word.
- $2^{(\text{Total number of bits of address})} = \text{No. of address locations}$
- $2^{(\text{No. of bits in set field})} = \text{No. of sets} = (\text{No. of lines} / \text{No. of lines per set})$
- $2^{(\text{No. of bits in word field})} = \text{No. of words in one memory block or cache line}$
- Number of bits in tag field = total no. of bits – bits in line field – bits in word field

E.g., 64KB cache with 2 lines per set. Each lines holds 4 bytes. Main memory is 16MB

Word: 2 bits ( $4 = 2^2$ )

Set:  $64K/4 = 16K$  lines. 8K sets. Therefore = 13 bits ( $8K = 2^{13}$ )

Tag: Total = 24 ( $16M = 2^{24}$ )  $24 - 2 - 13 = 13$ . Therefore = 9 bits.

### Replacement Algorithms

- Direct Mapping – no choice.
- Associate/Set associate – four choices: Least Recently Used (LRU), First In First Out (FIFO), Least Frequently Used (LFU), Random

**Operating Systems**

**Virtual Memory**

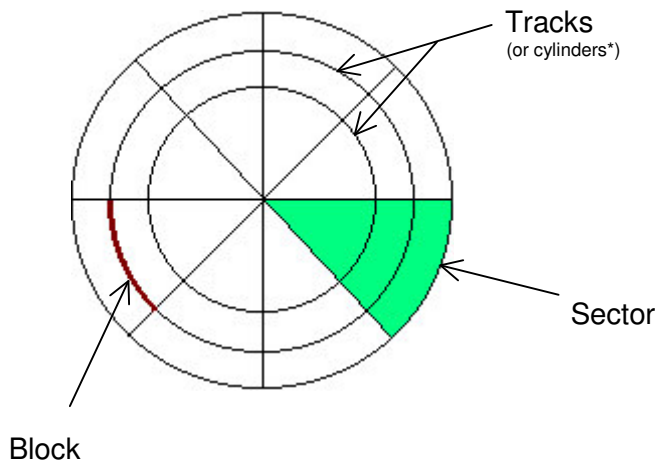
- Main memory is partitioned into fixed sized frames. Programs divided into 'pages', the same size as the frames. When the program requests a page, the corresponding frame is loaded into the main memory, hence programs can be written that are larger than the main memory.
- The frame location of each page is held in the Page Table, e.g.,

Virtual Page No.	Frame No.	Valid bit (1 if in memory)
1	4	1
2	--	0
3	7	1

- If the required page is not in memory, this is called a Page Fault. The OS must therefore swap in the required page (possibly by swapping out a page currently in memory using Replacement Algorithms as in Cache Systems, e.g. LRU, FIFO, etc.)

**External Memory**

**Magnetic Disks**



- Each sector on a single track contains one block of data (typically 512 bytes).
- All blocks contains the same amount of data, regardless of physical length. Constant speed motor – Constant *Angular Velocity*
- Seek time – time to move from one track to another.
- Latency time – once on the correct track, time to move to the correct sector.  
Maximum latency time =  $1/\text{Rotation Speed}$   
Average latency time =  $1/2 \times \text{Rotation Speed}$
- Transfer time – time to transfer one block of data =  $(1/\text{Rotation Speed}) / \text{No. of sectors}$ .
- Disks can be mutliplattered, i.e. several disks arranged on top of each other. In this case, tracks are called cylinders.

**Optical Disks**

<ul style="list-style-type: none"> <li>- Designed for maximum capacity</li> <li>- Each block same length along track</li> <li>- More bits per revolution at the outside</li> <li>- Variable speed motor – constant transfer time</li> <li>- Constant <i>Linear Velocity</i></li> </ul>	<p>One, long, spiral track. Divided into blocks. Plays from inside out.</p>
--	---