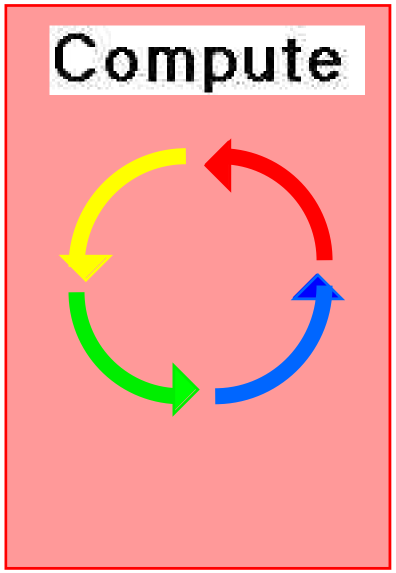
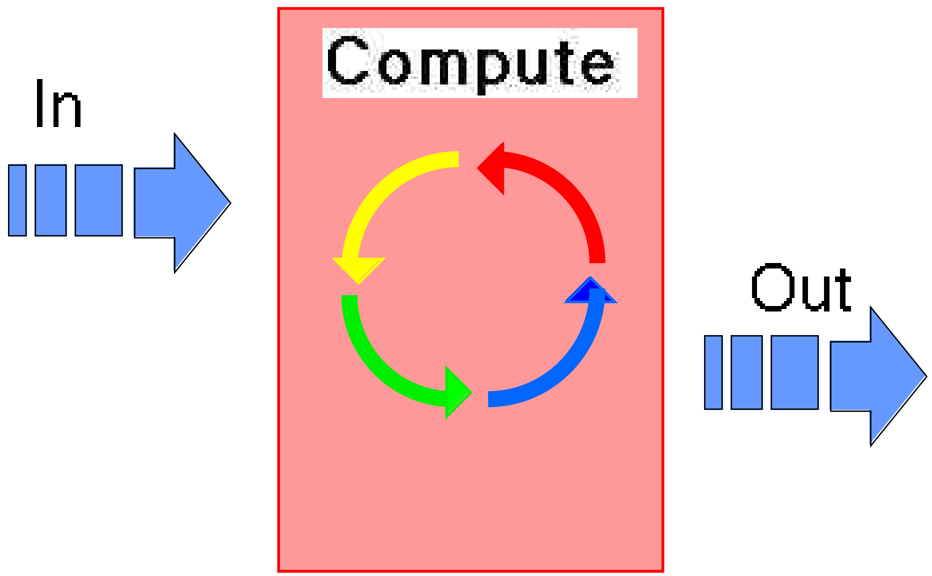
EG2069: Part 1

Introduction to Computers



Gorry Fairhurst

Dept of Engineering
University of Aberdeen

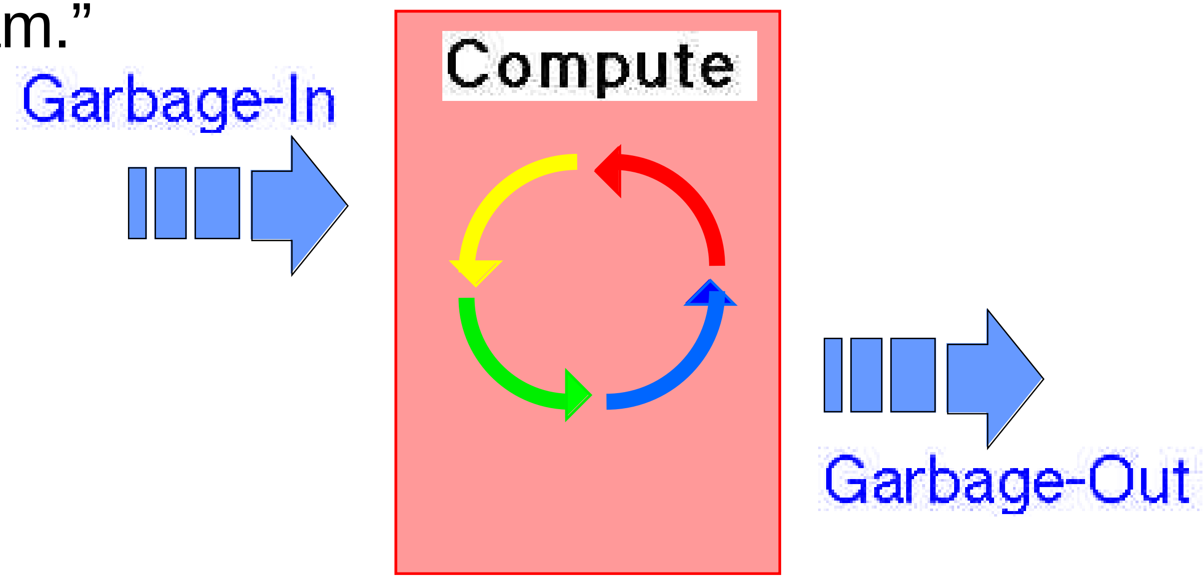


What is a computer?

Gorry Fairhurst

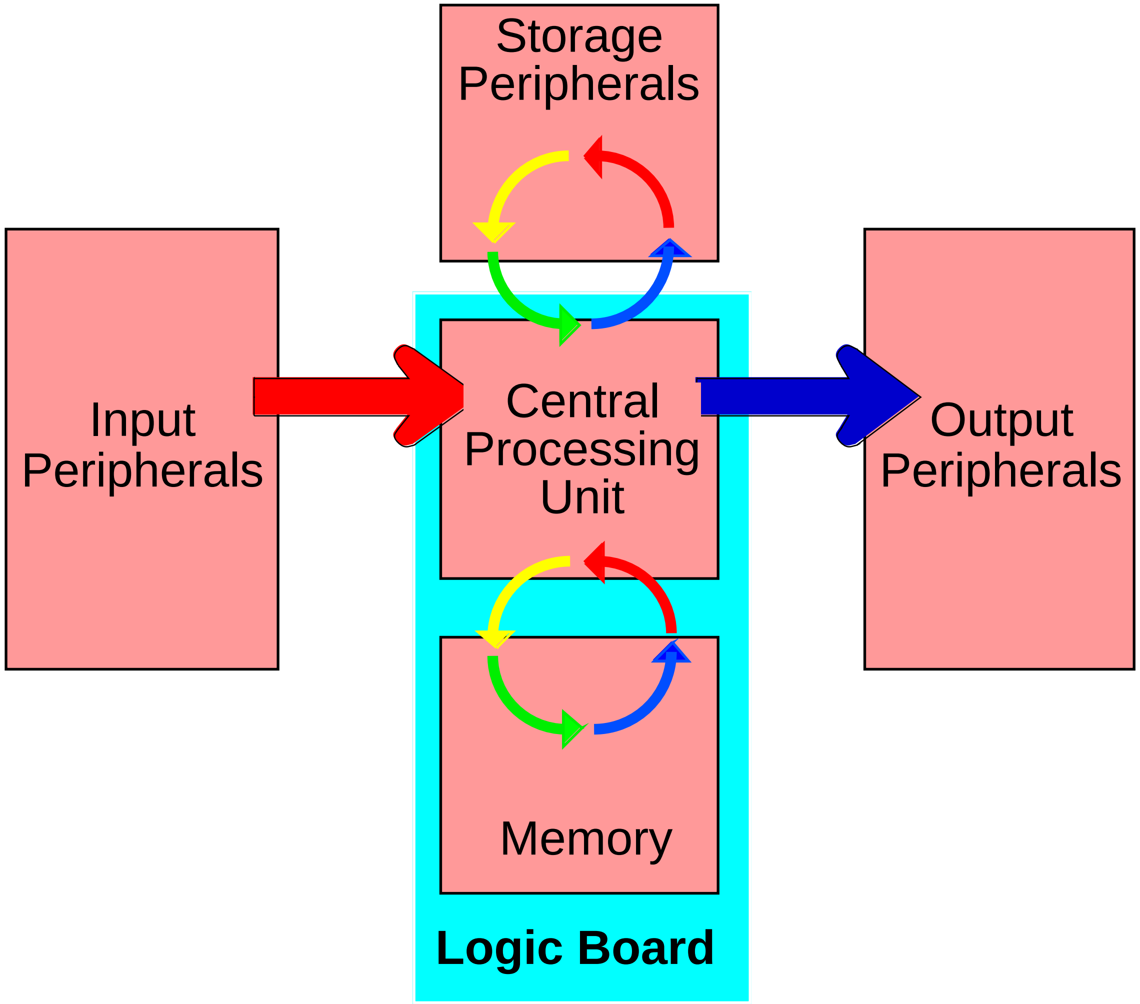
No fixed definition...

"A computer is a machine which can accept data, process the data and supply the results. The term is used for any computing device that operates according to a stored program."



The computer is only useful with a **valid program** and **correct data**

Gorry Fairhurst

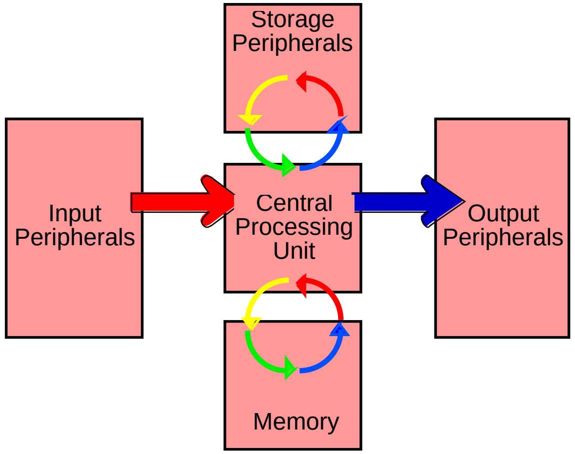


Peripheral Devices

Gorry Fairhurst

Input Devices

Scanner
Optical Character Recognition
Mouse
Keyboard
Microphone
Bar Code Reader
CD /CD-R
DVD/DVD-ROM
EPROM
Modem



Storage Devices

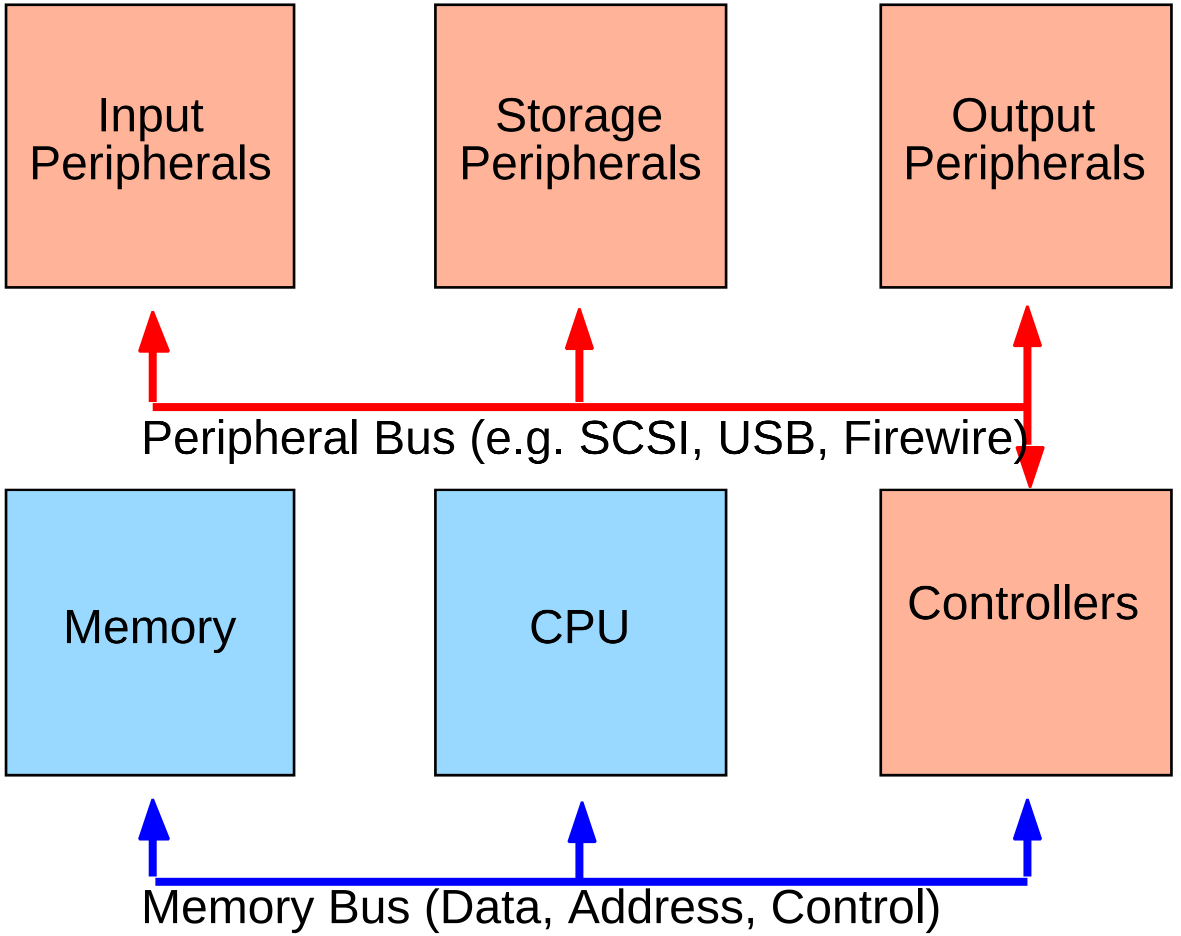
Magnetic Tape
Magnetic Disks
Magneto-Optical Discs
CD-RW
DVD-RAM
Flash Card

Output Devices

Printer
Plotter
Punched Paper Tape
CD-R
DVD-ROM
EPROM
Modem

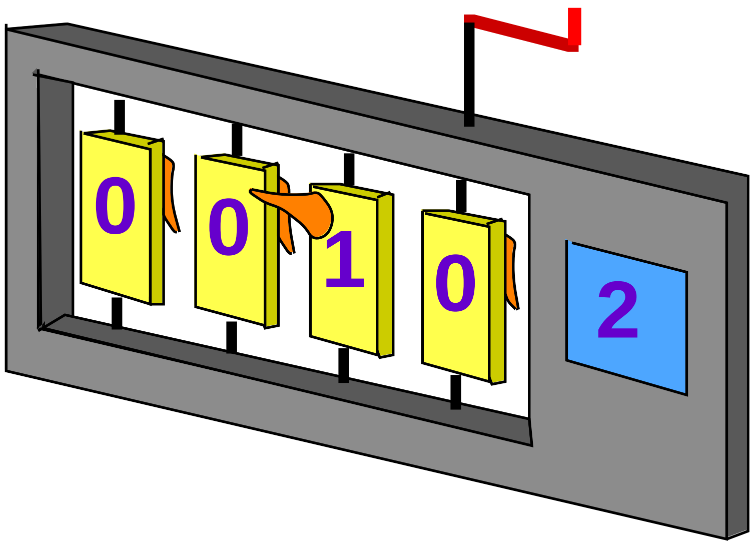
Computer Busses

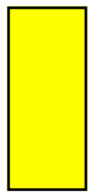
Gorry Fairhurst



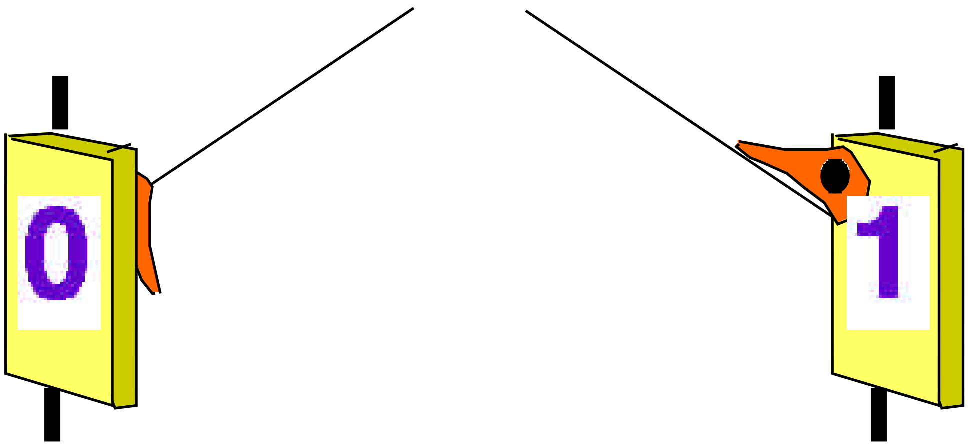
Gorry Fairhurst

Binary Numbers





One "BIT"
or BInary digiT



A bit can take only one of two values:

It is always either 0 or 1

Nybbles, Bytes and Words

Gorry Fairhurst

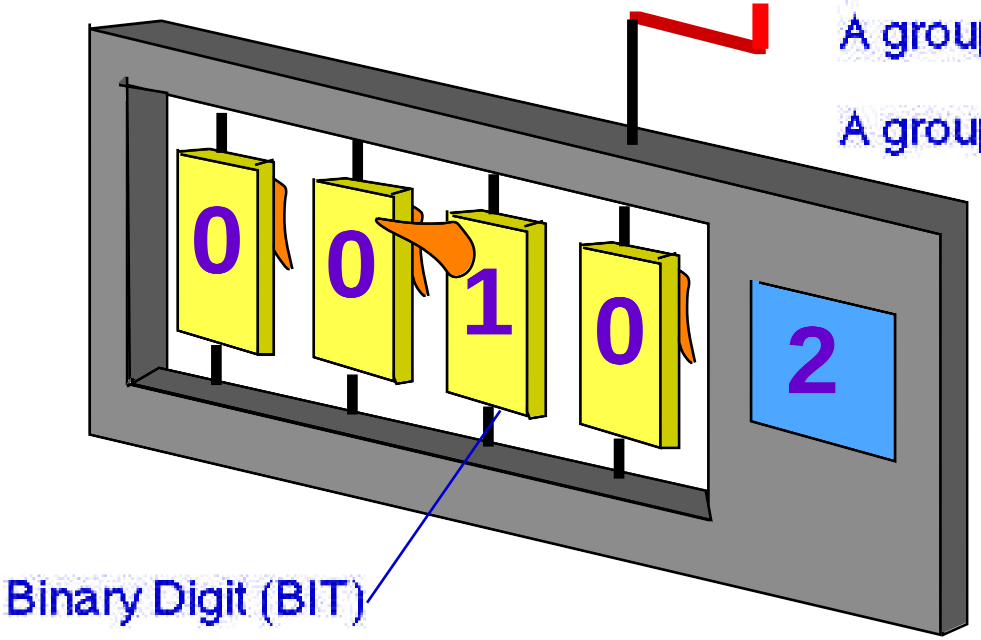
A single bit is not very useful

Bits are grouped together to form groups

A group of 4 bits = Nibble

A group of 8 bits = Byte

A group of bytes = Word



Binary to Decimal

Gorry Fairhurst

Convert by adding weights of digits

e.g. consider the binary number 1010

1010 $= 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$

$= 8 + 2$

$= 10.$

The same process as in decimal

e.g. $305 = 3 \times 10^2 + 0 \times 10^1 + 5 \times 10^0$

N.B. 100 in decimal = one hundred

100 in binary = four

Dec.	Binary
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111

Decimal to Binary

Gorry Fairhurst

Use repeated division by 2,
and record the remainders

e.g. convert 12 in decimal to binary

12 /2	= 6	rem 0
6/2	= 3	rem 0
3/2	= 1	rem 1
1/2	= 0	rem 1

Reading the remainders upwards:
12 is 1100 in binary

You can check by converting it back:

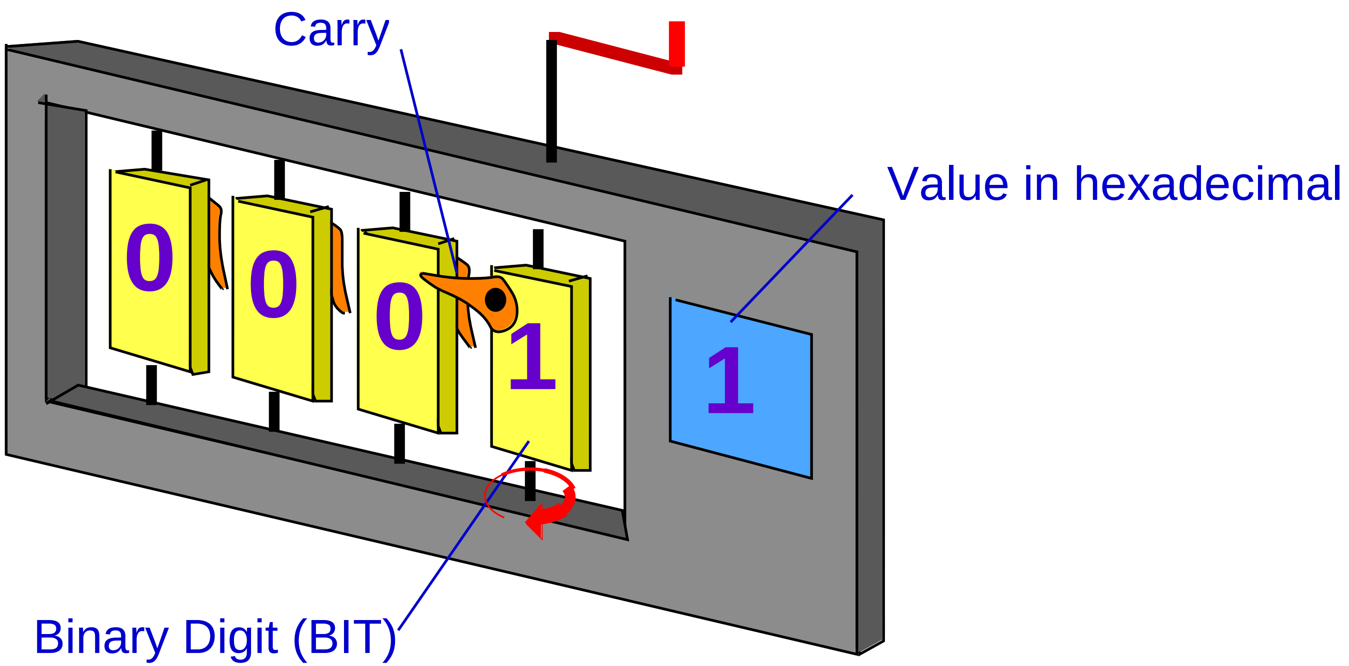
$1100 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0$
$= 8+4 = 12$

Dec.	Binary
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111

Gorry Fairhurst

Computers hold binary values in a “register”

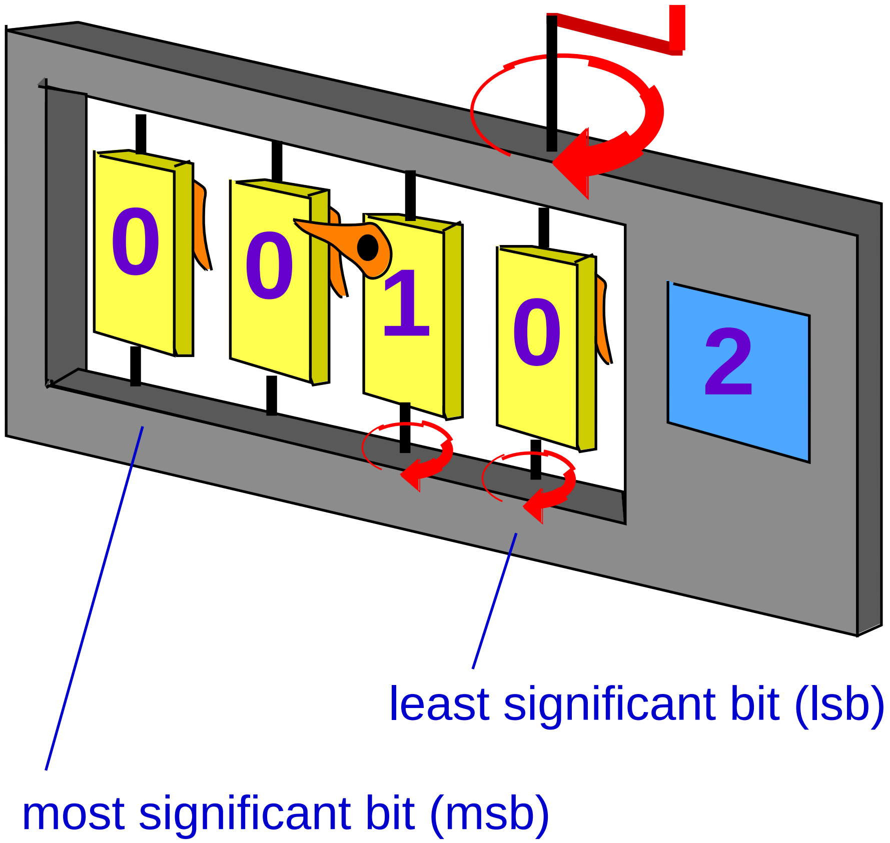
Consider the process of ***incrementing*** a register (adding one to the value stored in the register)



Incrementing the Model Register

Gorry Fairhurst

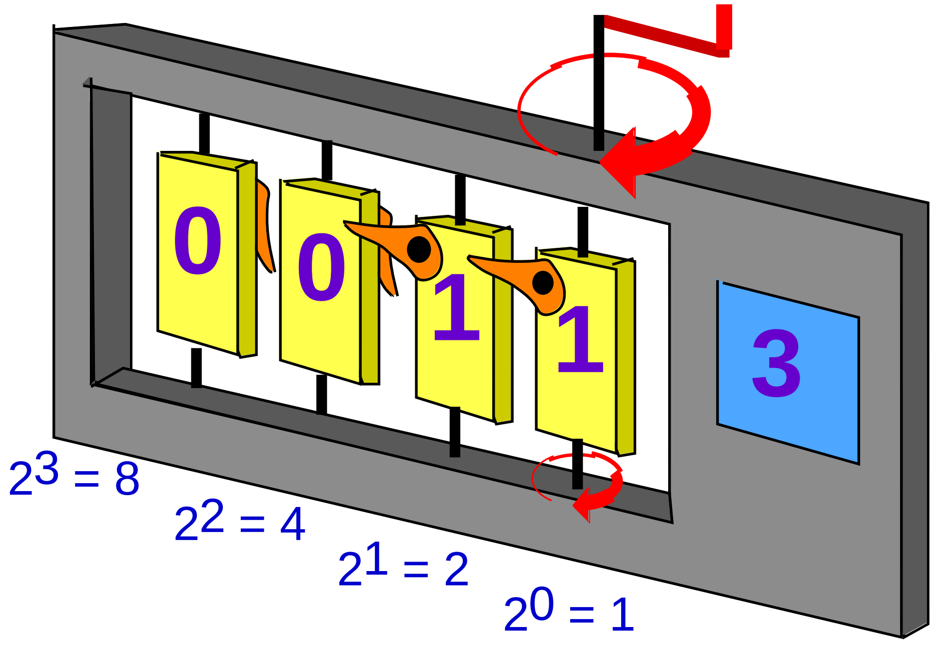
register ++



Incrementing the Model Register

Gorry Fairhurst

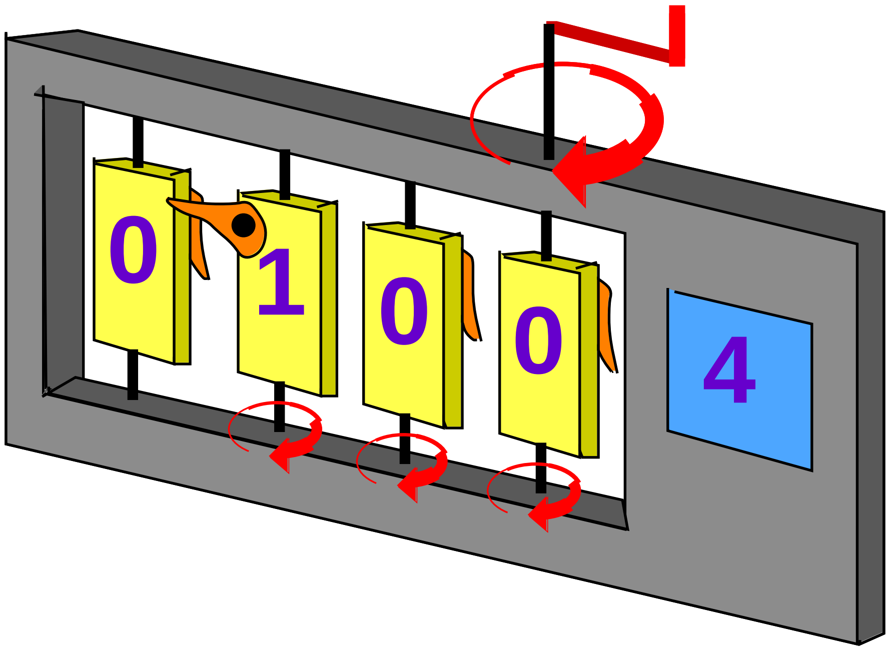
register ++



Incrementing the Model Register

Gorry Fairhurst

register ++



To add n, turn handle “n” times.

N.B.Real registers don’t use handles!!!
They use logic gates - but they do generate carries between digits

Binary Addition

Gorry Fairhurst

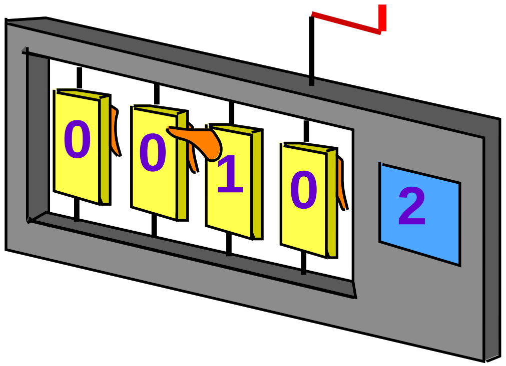
Adding single digits

```
    0        0        1        1
+   0    +   1    +   0    +   1
-----    -----    -----    -----
    0        1        1    1   0
```

Carry

Adding binary numbers

```
     010          110          101
+    101     +    101     +    011
--------     --------     --------
     111         1011         1000
--------     --------     --------
                 1             111
```



In general a group of n bits may represent a set of 2^n values

i.e. digits $\{0,1,2, \ldots, (2^n-1)\}$

For 4 bits, n=4 therefore 2^4 or 16 values

Digits 0..15 {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15}

It's not convienent to use two symbols for one digit!!!

So we normally use letters for digits greater than 9

Hence: {0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F}

Converting Hexadecimal to Binary

Gorry Fairhurst

Numbers represented by digits {0..F} use base 16, or ***hexadecimal***

Each hexadecimal digit may be represented by 4 bit.

To convert a hexadecimal number to binary convert each digit:
0x01FF = 0000 0001 1111 1111

Similarly:
1111 1111 0000 0000 =0xF0

N.B. To recognise hex numbers we usually write “0x” before them!

Dec.	Hex.	Binary
0	0x0	0000
1	0x1	0001
2	0x2	0010
3	0x3	0011
4	0x4	0100
5	0x5	0101
6	0x6	0110
7	0x7	0111
8	0x8	1000
9	0x9	1001
10	0xA	1010
11	0xB	1011
12	0xC	1100
13	0xD	1101
14	0xE	1110
15	0xF	1111

Converting Hexadecimal to Decimal

Gorry Fairhurst

Convert Hex to decimal by adding weights of digits

$0x1C7 = 1x16^2 + Cx16^1 + 7x16^0$

$= 1x256 + 12x16 + 7$

$= 455.$

Convert Decimal to Hexadecimal by repeated division by 16.
e.g. convert 456 to hex

456 /16	= 28	rem 8	(0x8)
28/16	= 1	rem 12	(0xC)
1/16	= 0	rem 1	(0x1)

Reading the remainders upwards:
456 is 0x1C8 in hexadecimal

Dec.	Hex.
0	0x0
1	0x1
2	0x2
3	0x3
4	0x4
5	0x5
6	0x6
7	0x7
8	0x8
9	0x9
10	0xA
11	0xB
12	0xC
13	0xD
14	0xE
15	0xF

Gorry Fairhurst

Decimal to Hexadecimal

Converting 53241 decimal to hexadecimal:

53241	÷16 =	3327	R 9	(0x9) msb
3327	÷16=	207	R 15_{10}	(0xF)
207	÷16=	12	R 15_{10}	(0xF) lsb
12	÷16=	0	R 12_{10}	(0xC)

53241 = 0x00CFF9

Convention that positive numbers start with 0x0

Hexadecimal to Decimal

Value of digit

Position of digit

Converting 0x00CFF9 to decimal:

$= (9 \times 16^3) + (15 \times 16^2) (15 \times 16^1) + (12 \times 16^0)$

= 53241

Hexadecimal Addition

Gorry Fairhurst

```
  20        0x14          0001 0100
 +5        +0x05         +0000 0101
 ___       _____         __________
 =25       =0x19         =0001 1001
```

a b c	S	C
0 0 0	0	0
0 0 1	1	0
0 1 0	1	0
0 1 1	0	1
1 0 0	1	0
1 0 1	0	1
1 1 0	0	1
1 1 1	1	1

3 bit binary adder

0 + 1 = 1

0 + 0 = 0

1+1 = 0, c

0 + 0 + c = 1

0 + 1 = 1

N.B.
Sum = 1 if there are an odd number of 1's
Carry = 1 if there are two or more 1's

Binary

2 values per digit {0,1}

e.g. 10100 = $1x2^4+0x2^3+1x2^2+0x2^1+0x2^0$

Decimal

10 values per digit {0,1,2,3,4,5,6,7,8,9}

e.g. 20 = $2x10^1+0x10^0$

Hexadecimal

16 values per digit {0,1,2,3,4,5,6,7,8,9,A,B,C,D,E.F}

e.g.14 = $1x16^1+4x16^0$

Hexadecimal Signed Numbers

Gorry Fairhurst

1's Complement (bit-wise inversion)

int x
x = ~x

int's are
normally 4 r1
(or 8 nibbles)

Examples using a 32 bit register
(8 hexadecimal digits)

e.g.

20 = 0x00000014

msb = 0 for positive number
msb = 1 for negative number.

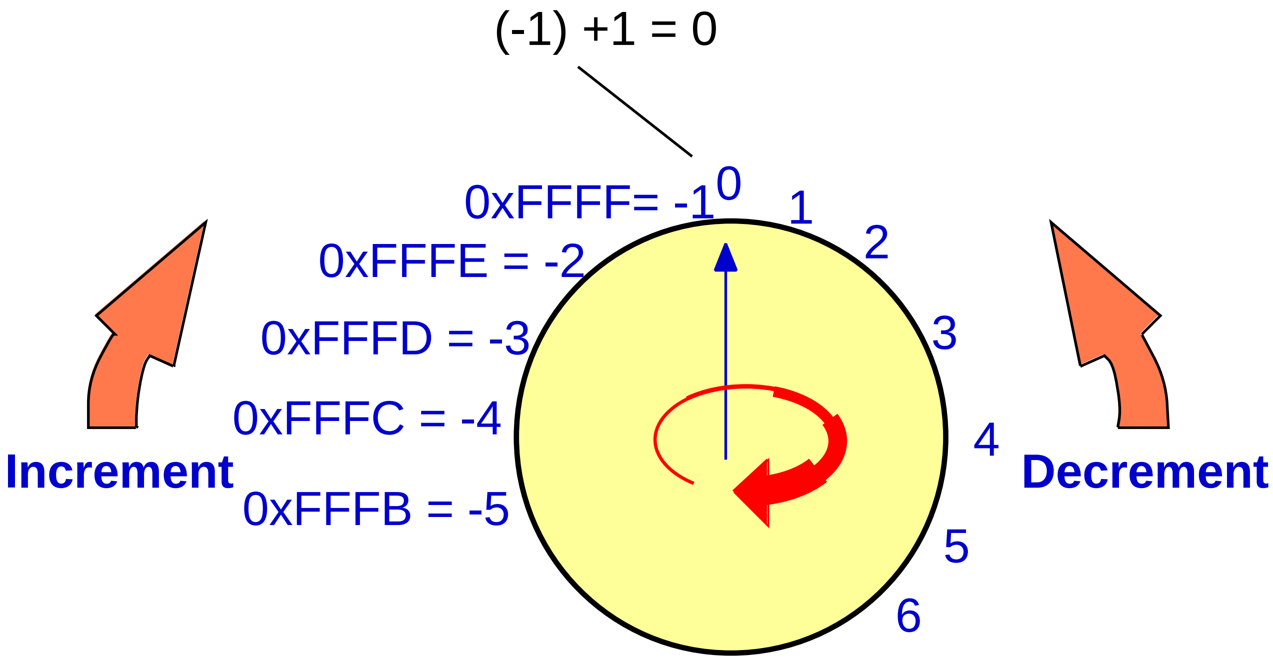
-20 = 0xFFFFFFEB

0x means the number is in hexadecimal

Note that the *size* of variable determines how many digits!

Hexadecimal Signed Numbers

Gorry Fairhurst



Subtraction

Gorry Fairhurst

Subtraction is **difficult**!

Easier to **negate** a value in 2’s complement and then **add**

20	0x14	0001 0100	0001 0100
-5	- 0x05	- 0000 0101	+1111 1011
=15	=0x0F	=0000 1111	= 0000 11 11

-5 as a byte

A carry is generated and ignored at the msb

Hexadecimal Signed Numbers

Gorry Fairhurst

2's Complement (true negation)

int x
x = (~x)+1

Examples using a 32 bit register
(8 hexadecimal digits)

e.g.

20 = 0x00000014

Sufficient to add 1 or 2 zeros before the first non-zero digit.

-20 = 0xFFFFFFEC

More care is needed to get the size correct for negative numbers

Signed and Unsigned Numbers

Sign bit

1111 1101

The binary value “1111 1101” has the msb set,
it may therefore be interpreted as either:

The *unsigned char* 0x00FD (+253)

or

The *signed char* 2’s complement number 0xFD (-3)

N.B.
In C the size of the type “char” is one byte
It is important to know the ***type*** *of the number*
to determine the value when the msb is set to 1.

Size of Variables

Gorry Fairhurst

char (8 bits)

0111 1101

short int (16 bits)

0000 0000	0111 1101

int (32 bits)

0000 0000	0000 0000	0000 0000	0111 1101

N.B. The assembler (or compiler) must determine the size of each variable to use the correct instruction

Type Conversion

Gorry Fairhurst

int = *signed* char (125)

0111 1101

0000 0000	0000 0000	0000 0000	0111 1101

int = *signed* char (-3)

1111 1101

1111 1111	1111 1111	1111 1111	1111 1101

int = *unsigned* char (253)

1111 1101

0000 0000	0000 0000	0000 0000	1111 1101

*N.B. For signed values, the sign must be **extended***

Multiplication by 2

Gorry Fairhurst

Multiplication by 2 implies adding a 0 to a binary number

e.g. consider the binary number 1010 (10 in decimal) x 2

1010 x 2 = 10100

$= 1 \times 2^3 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0$

= 20 (decimal)

This is a ***shift*** operation, each digit is ***shifted left***.

The same process as in decimal!

In the C programming language we write

a shift right n places as <<n, meaning multiply by 2^n

Hence 0x2<<1 = 0x4, 0x1<<2 =0x8.

Use ***long multiplication*** to multiply by other values

Division by 2

Gorry Fairhurst

Division by 2 implies deleting a digit from a binary number

e.g. consider the binary number 1010 (10 in decimal) / 2
1010 / 2 = 101
$= 1x2^2+0x2^1+1x2^0$
= 5 (decimal)

This is a ***shift*** operation, each digit is ***shifted right***.

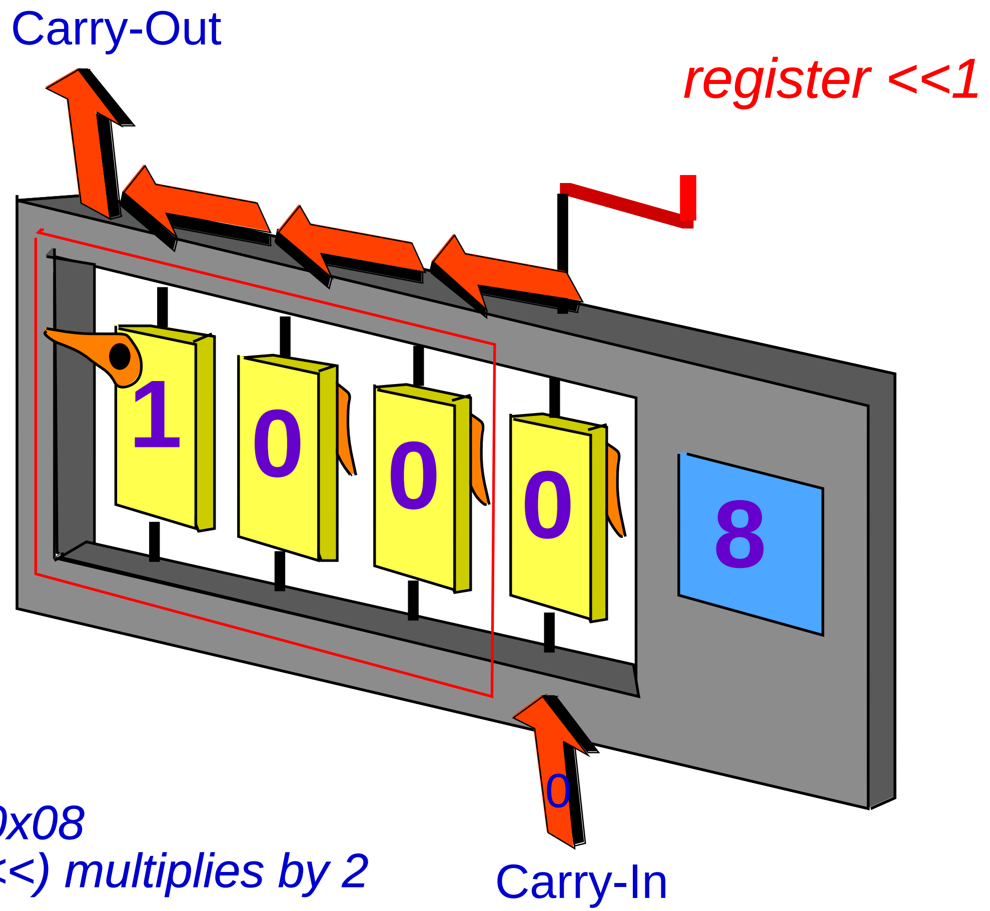
The same process as in decimal!

In the C programming language we write
a shift right n places as >>n, meaning divide by 2^n
Hence 0x2>>1 = 0x1, 0xF>>2 =0x3.

Use ***Booth's algorithm*** to perform long division

Model Left Shift Register

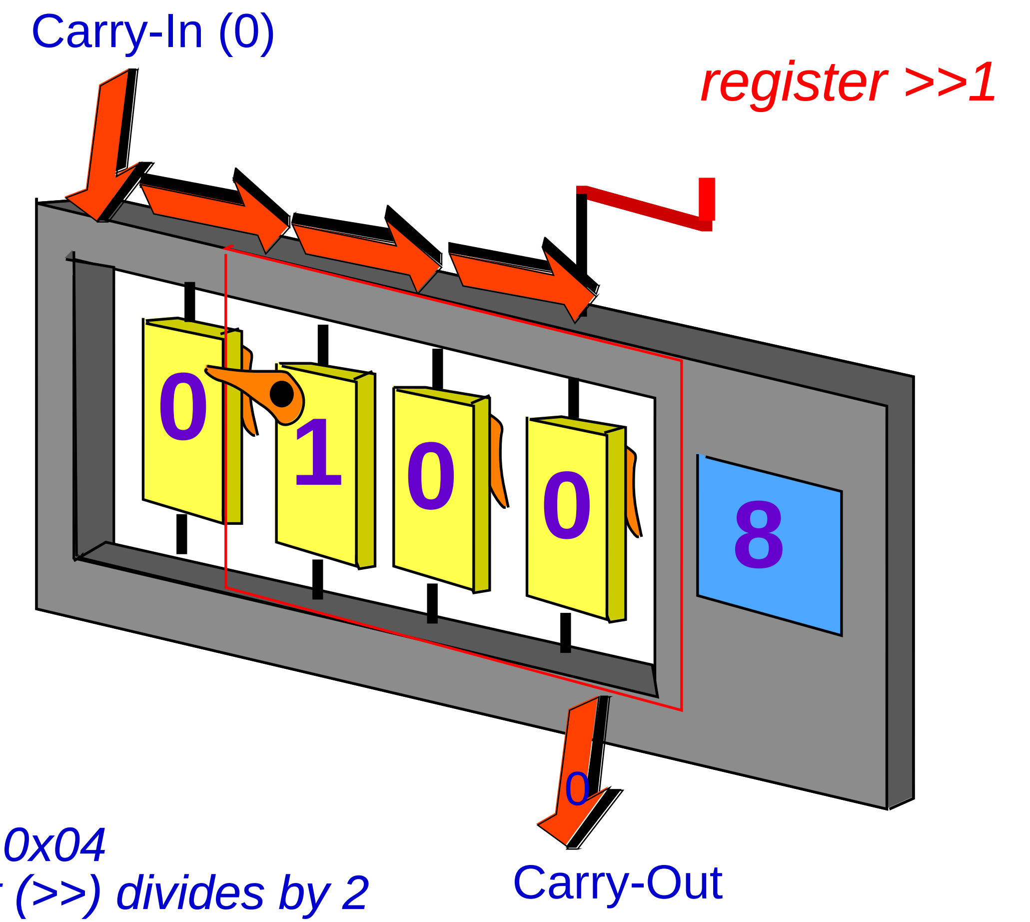
Gorry Fairhurst



N.B.
0x04<<1 = 0x08
A shift left (<<) multiplies by 2

Model Right Shift Register

Gorry Fairhurst



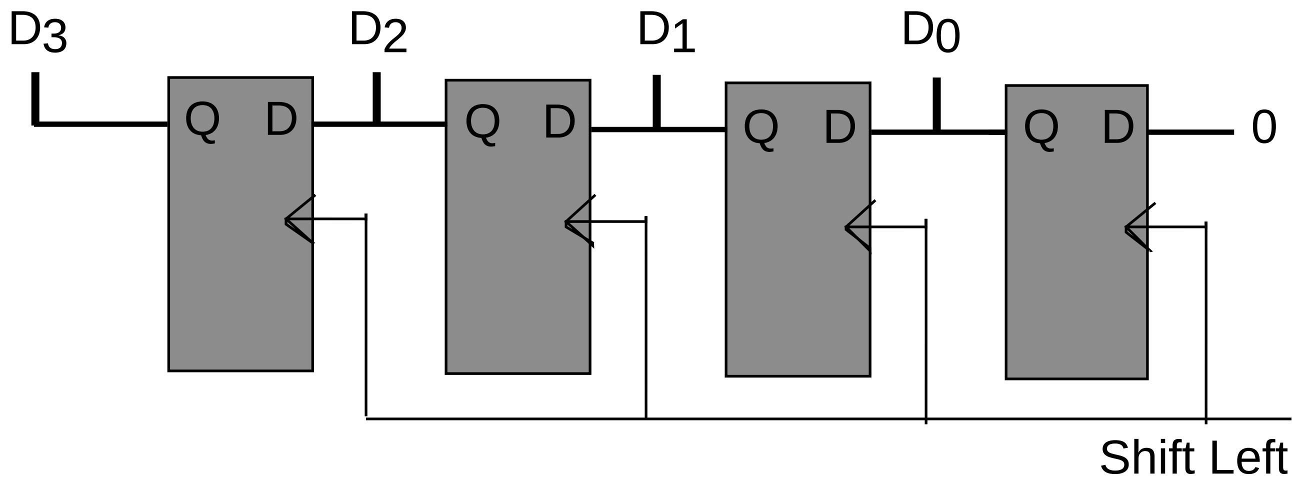
N.B.
0x08>>1 = 0x04
A shift right (>>) divides by 2

Gorry Fairhurst

Shifting is actually implemented by a shift register

The basic operation is the same:

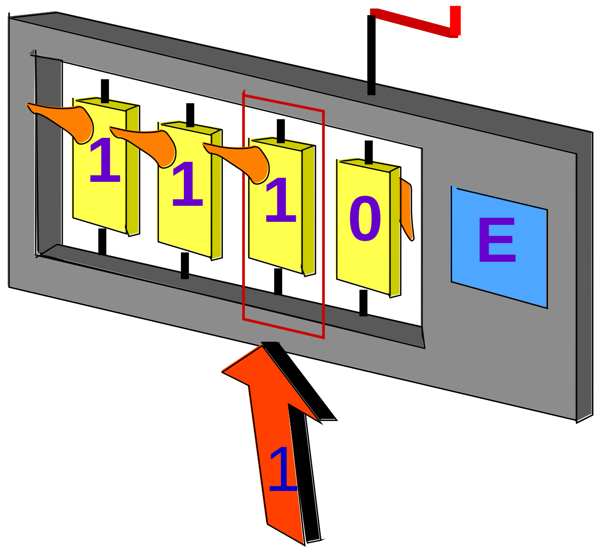
- Output of each bit feeds the input of the next
- The last bit generates a "carry"



Bit-Wise Logical Operators in the Model Register

Gorry Fairhurst

OR (preset)



	1100
OR	0010
=	1110

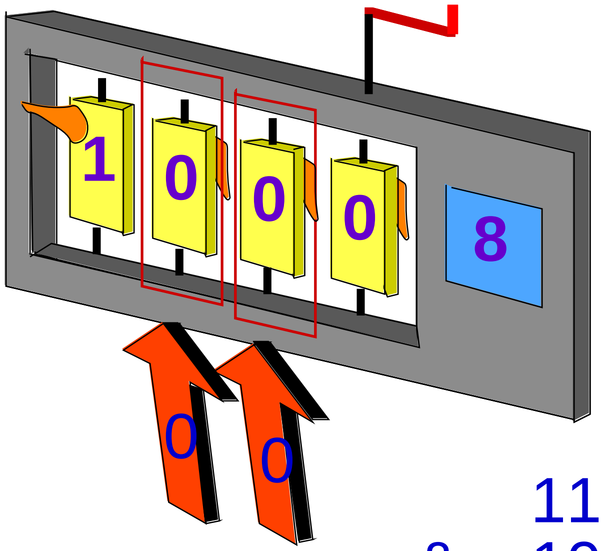
N.B. A OR 0 = A

A OR -1 = -1

Bit-Wise Logical Operators in the Model Register

Gorry Fairhurst

AND (clear)



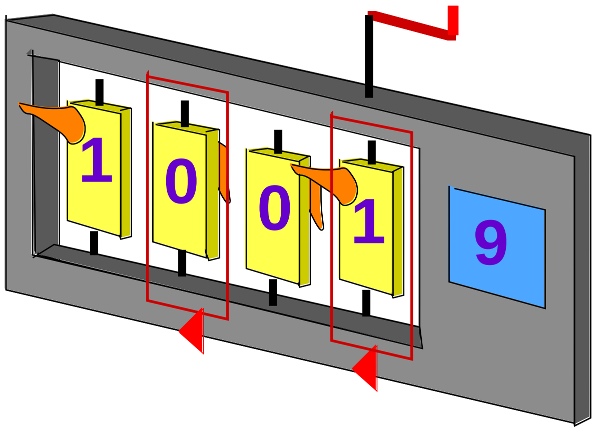
1100
& 1001
= 1000

N.B. A AND 0 = 0
A AND -1 = A

Bit-Wise Logical Operators in the Model Register

Gorry Fairhurst

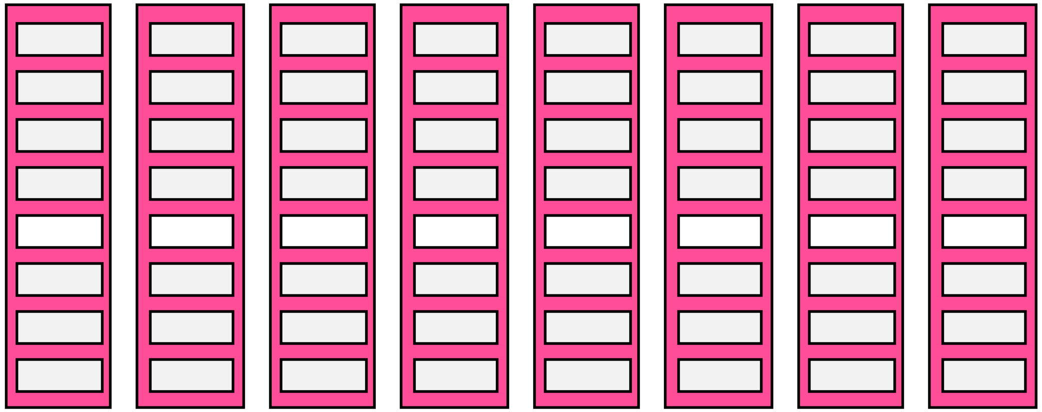
XOR (invert)



1100
XOR
0101
=
1001

N.B. A XOR -1 = ~A

Gorry Fairhurst

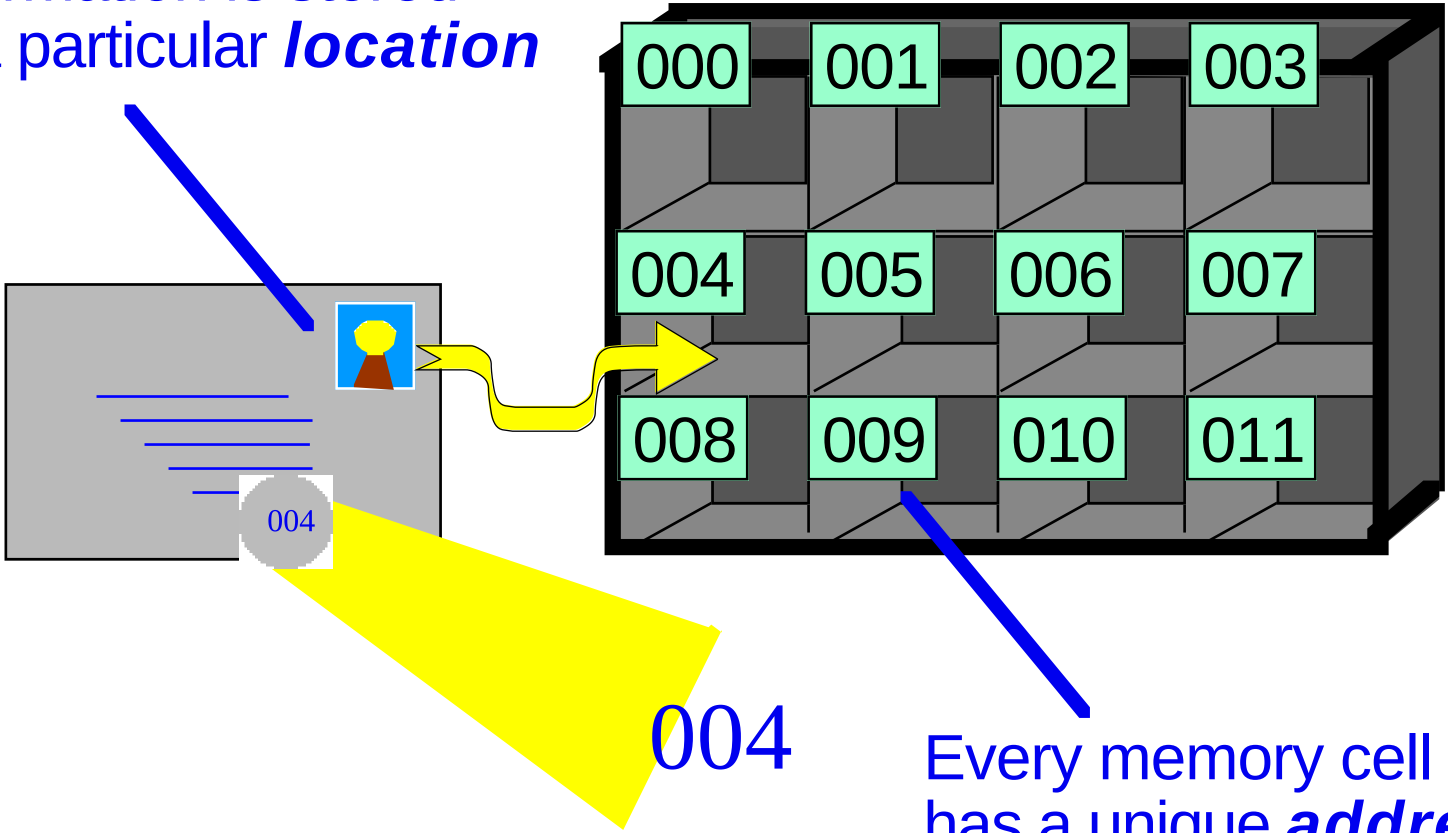


Caches & Memory

Storing Information in Memory

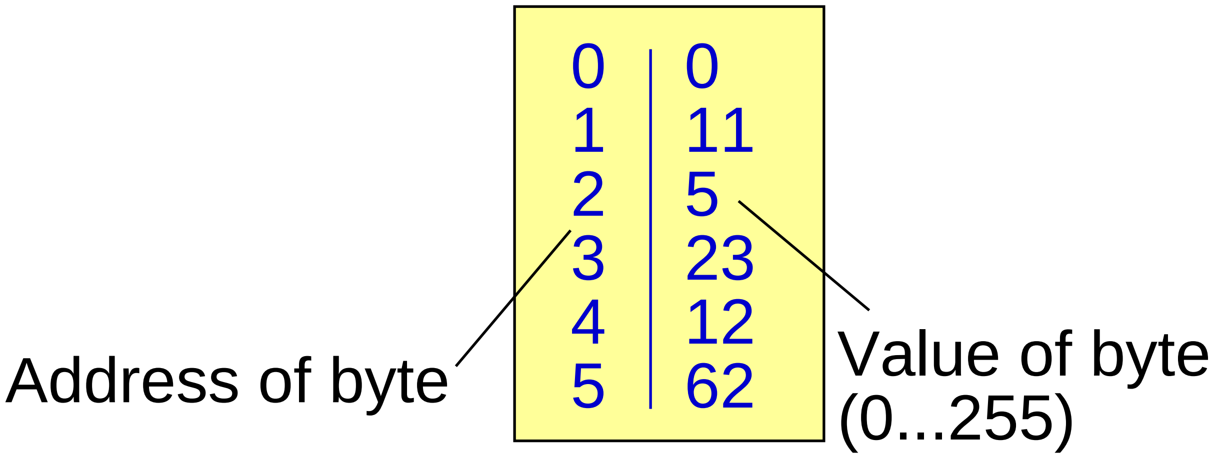
Gorry Fairhurst

Each piece of information is stored in a particular ***location***



Every memory cell has a unique ***address***

Gorry Fairhurst

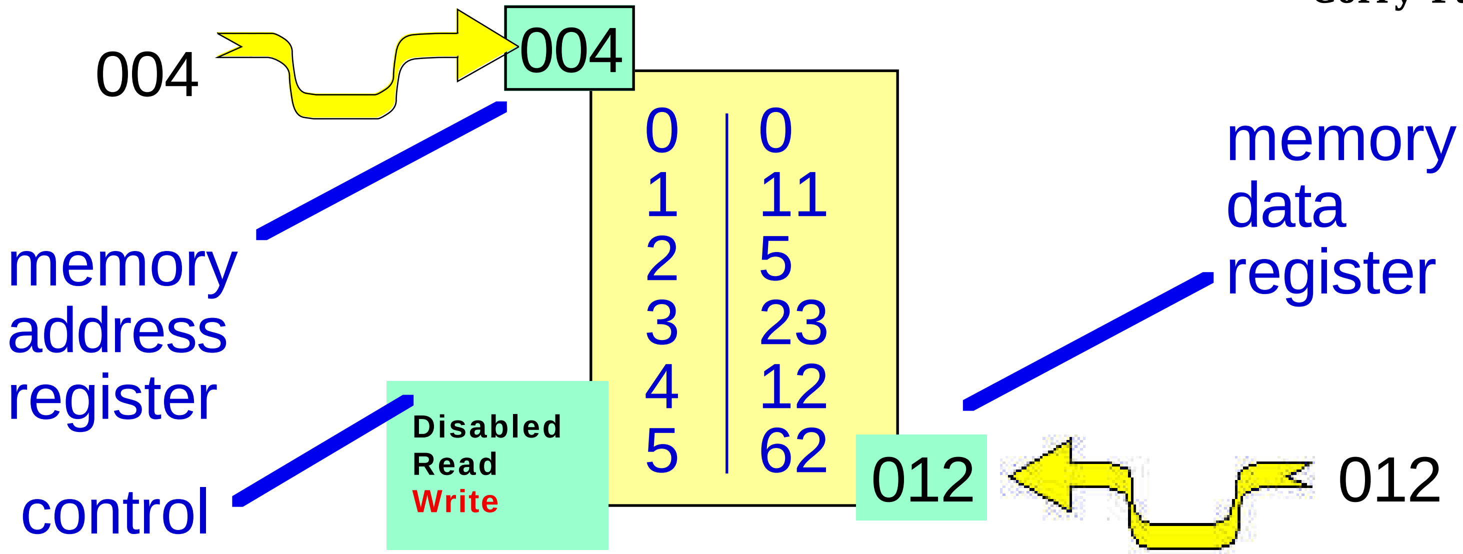


Memory is normally thought of as linear list
(in computing we call this an ***array***).

Memory locations normally store a single BYTE
(each location stores a number 0..255)

Writing a Value to an Address

Gorry Fairhurst

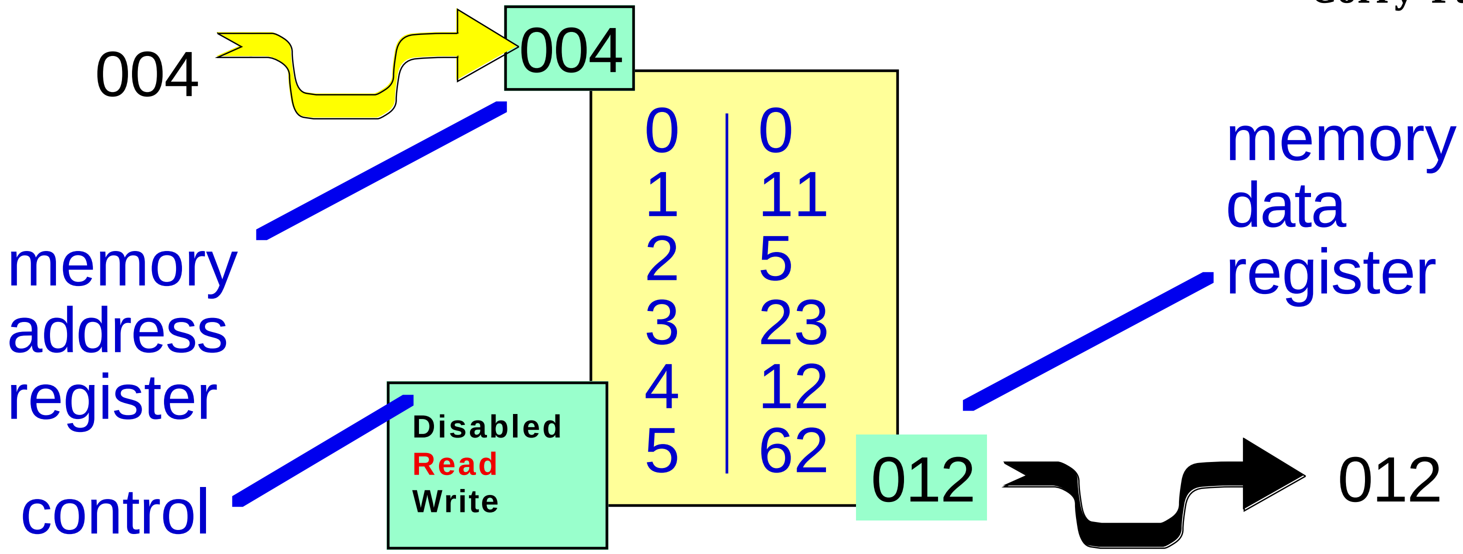


To write a value to the 4th location:

(i) Set the memory address value to 4
(ii) Set the data register to the value (e.g. 23)
(iii) Activate the WRITE control
(iv) DISABLE the memory

Reading the Value at an Address

Gorry Fairhurst



To read a value from the 4th location:

(i) Set the memory address value to 4
(ii) Set the memory to READ
(iii) The data register returns the value (e.g. 23)
(iv) DISABLE the memory

Random Access Memory (RAM)

Gorry Fairhurst

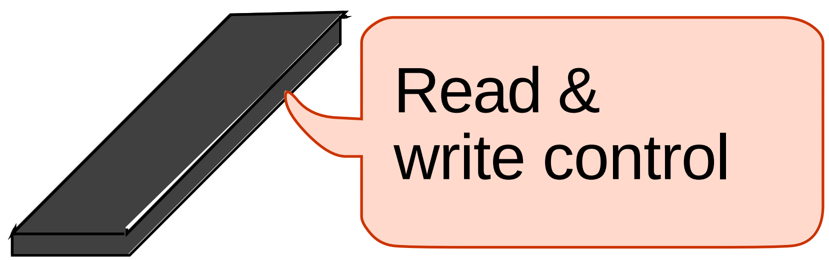
Read / Write supported

Used for storing programs and data

Looses all data when power removed (volatile)

Non-volatile alternatives:

ROM, EPROM, FLASH



Read Only Memory (ROM)

Gorry Fairhurst

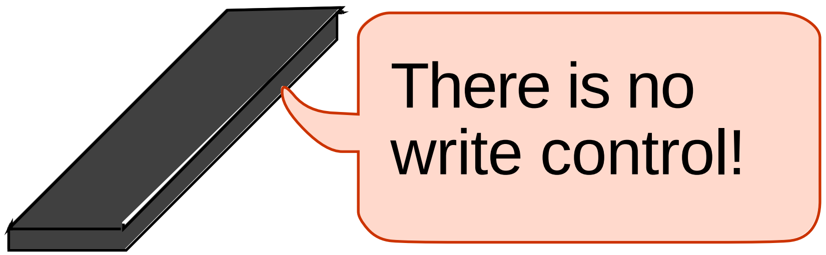
Program/Data is set at manufacture

May be mass-produced very cheaply

Can never be changed (except by replacing ROM)

Used for storing parts programs that never change
e.g. parts of operating system kernel (firmware)

For programs it is more flexible to use EPROM, FLASH



Flash Memory

Gorry Fairhurst

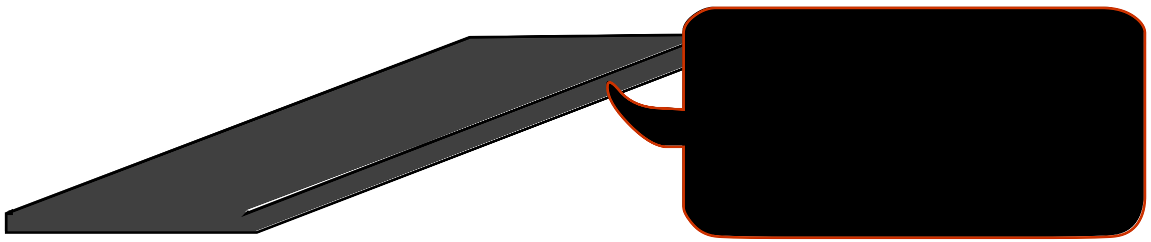
Program/Data is written by CPU

May be upgraded very easily

Used primarily for storing programs
and configuration data

Very expensive compared to ROM, EPROM

Much slower (particularly to write) than RAM



Erasable Programmable Read Only Memory

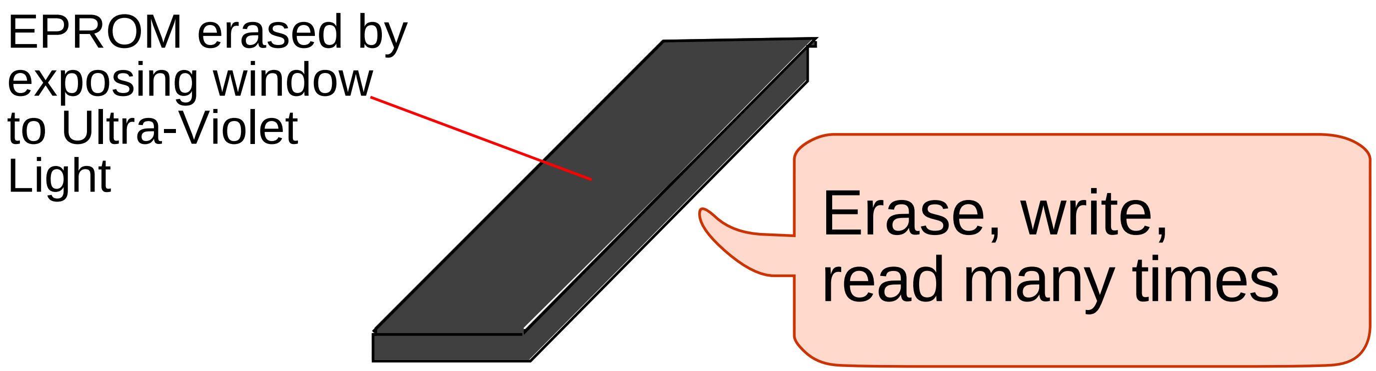
Gorry Fairhurst

Program/Data is written by an EPROM programmer

Whole chip needs to be erased
(needs to be taken out of computer)

Used primarily for storing programs

More expensive than ROM, but reusable

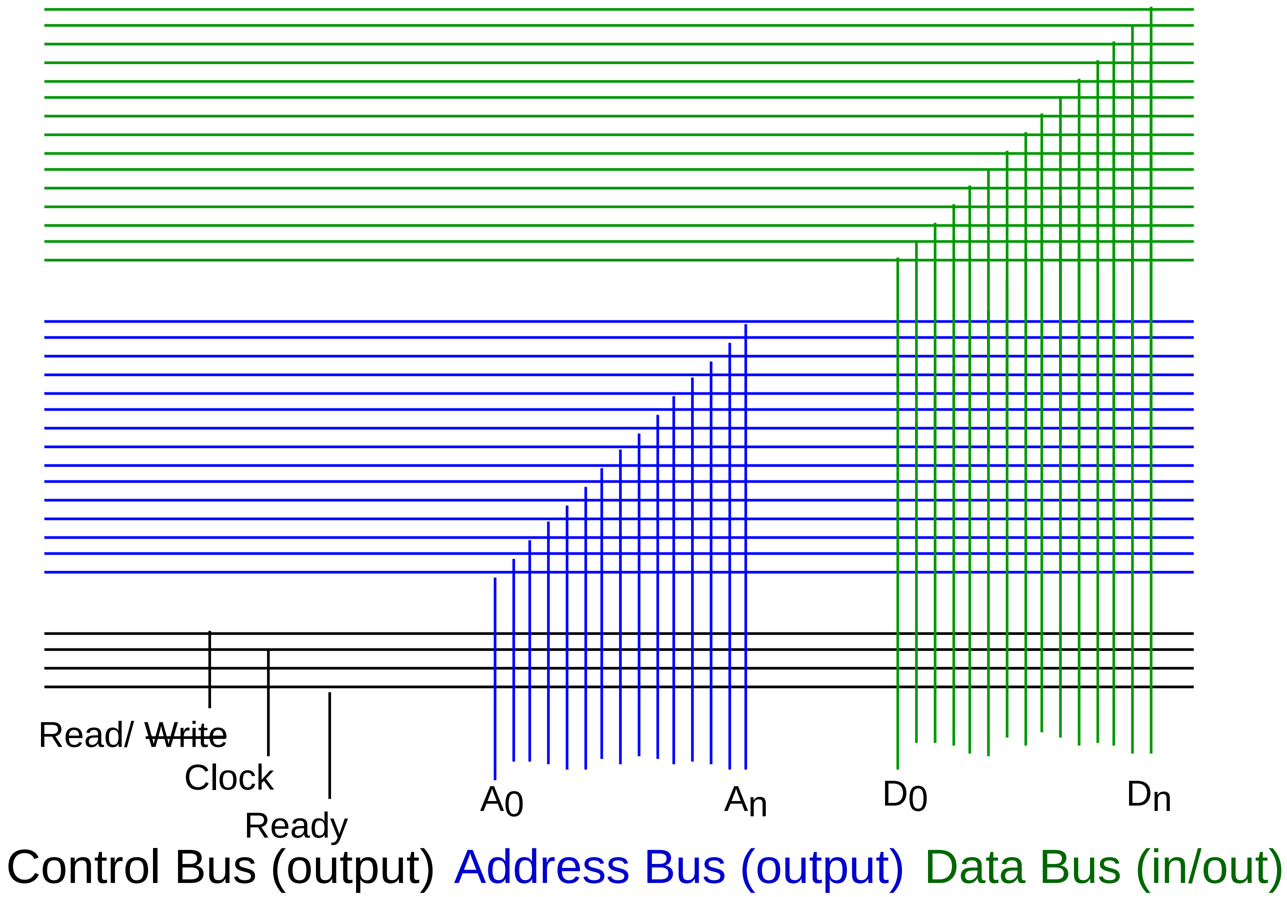


Volatile memory (looses data when no power)	
Dynamic RAM (cheap)	Static RAM (expensive)
fast	very fast
main memory	cache & I/O buffer

Non-volatile memory (keeps data when no power)		
ROM (cheap)	EPROM (cheap)	FLASH (cheap)
fast	fast	slow
programs (one use)	programs (reusable)	programs and data

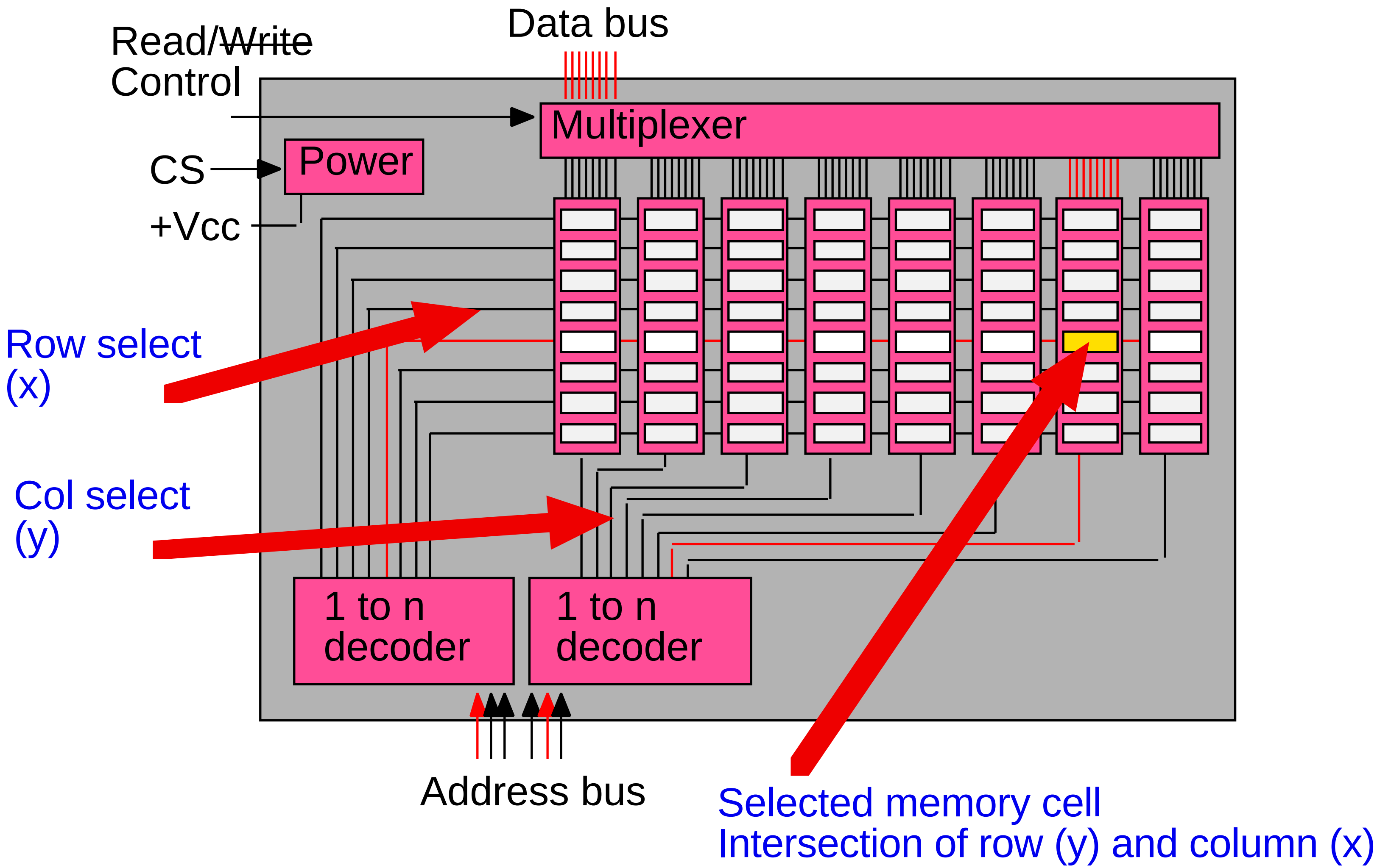
Address, Data, & Control Bus

Gorry Fairhurst



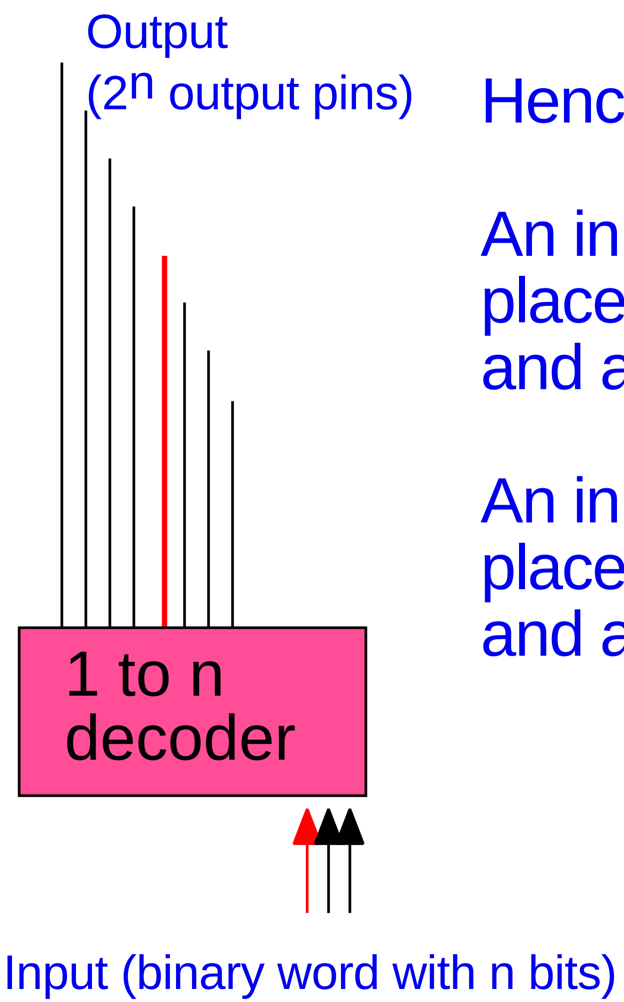
Random Access Memory (RAM)

Gorry Fairhurst



Decoder

Gorry Fairhurst



The decoder selects only one output pin

Hence a 3 bit decoder selects 1 of 8 pins

An input of 100 (4)
places a 1 at the output pin 4
and a 0 at all other output pins.

An input of 101 (5)
places a 1 at the output pin 5
and a 0 at all other output pins.

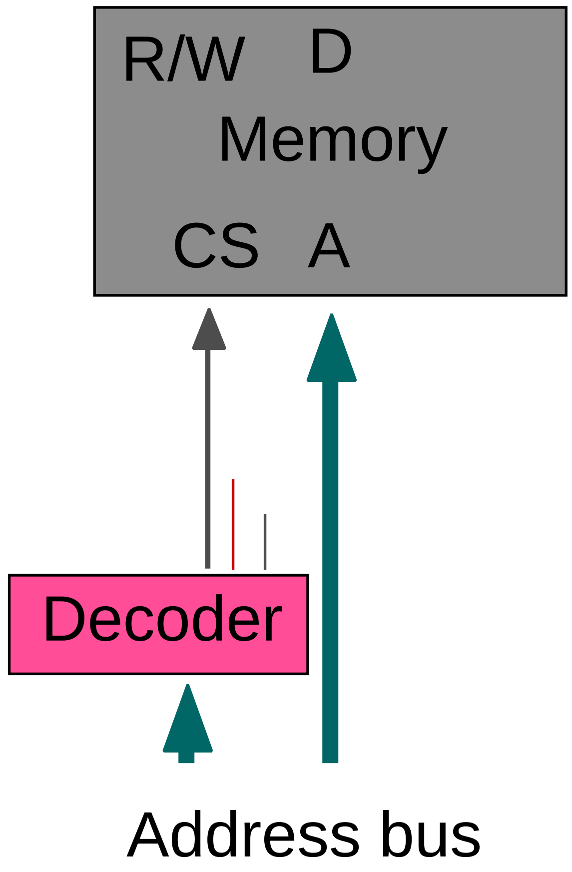
Dec.	**Binary**
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111

Control input that enables the chip
- if CS=0, ignores all other pins
- if CS=1, obeys R/W controls.

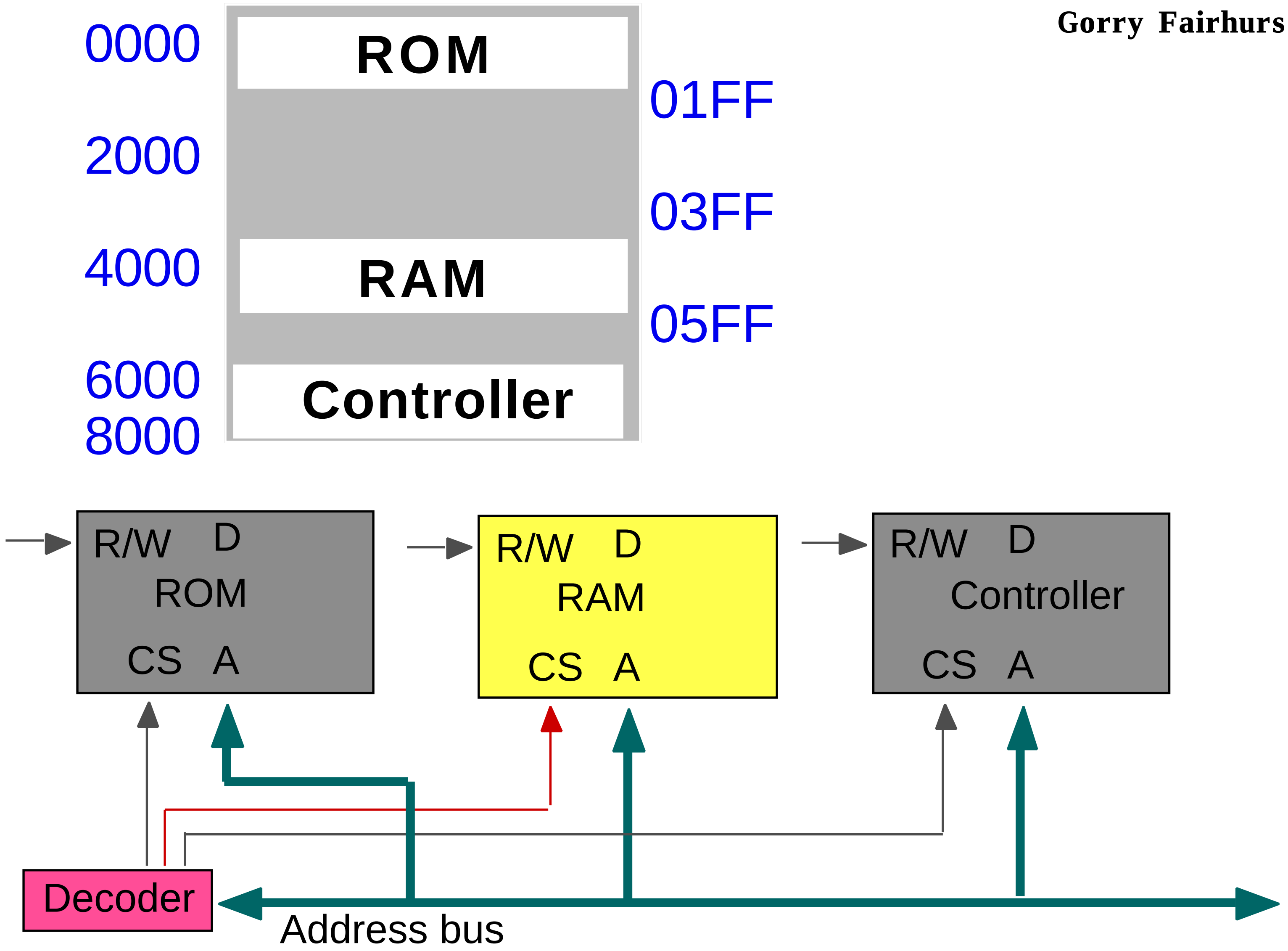
At any time, only one chip has CS=1, others must have CS=0.

CS value obtained by feeding highest bits of address bus to a decoder. Each CS is connected to an output.

The lower bits of the address bus connect to address pins of the chip.

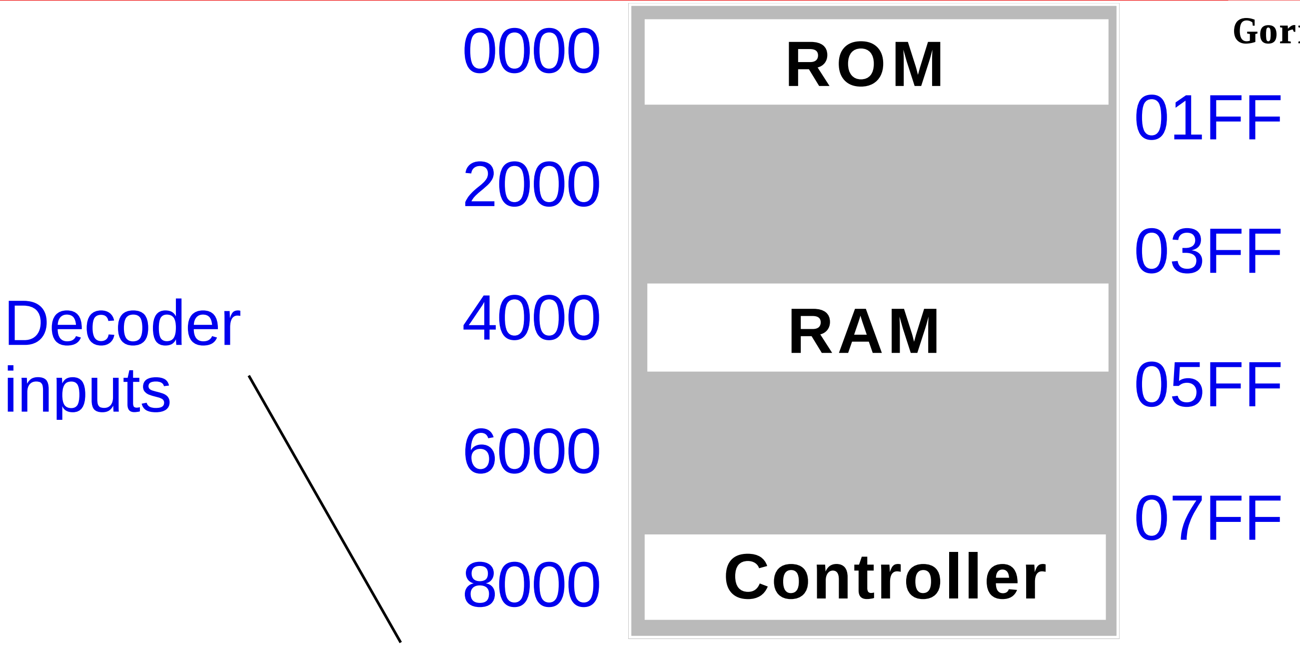


Gorry Fairhurst



Inputs to the Address Decoder

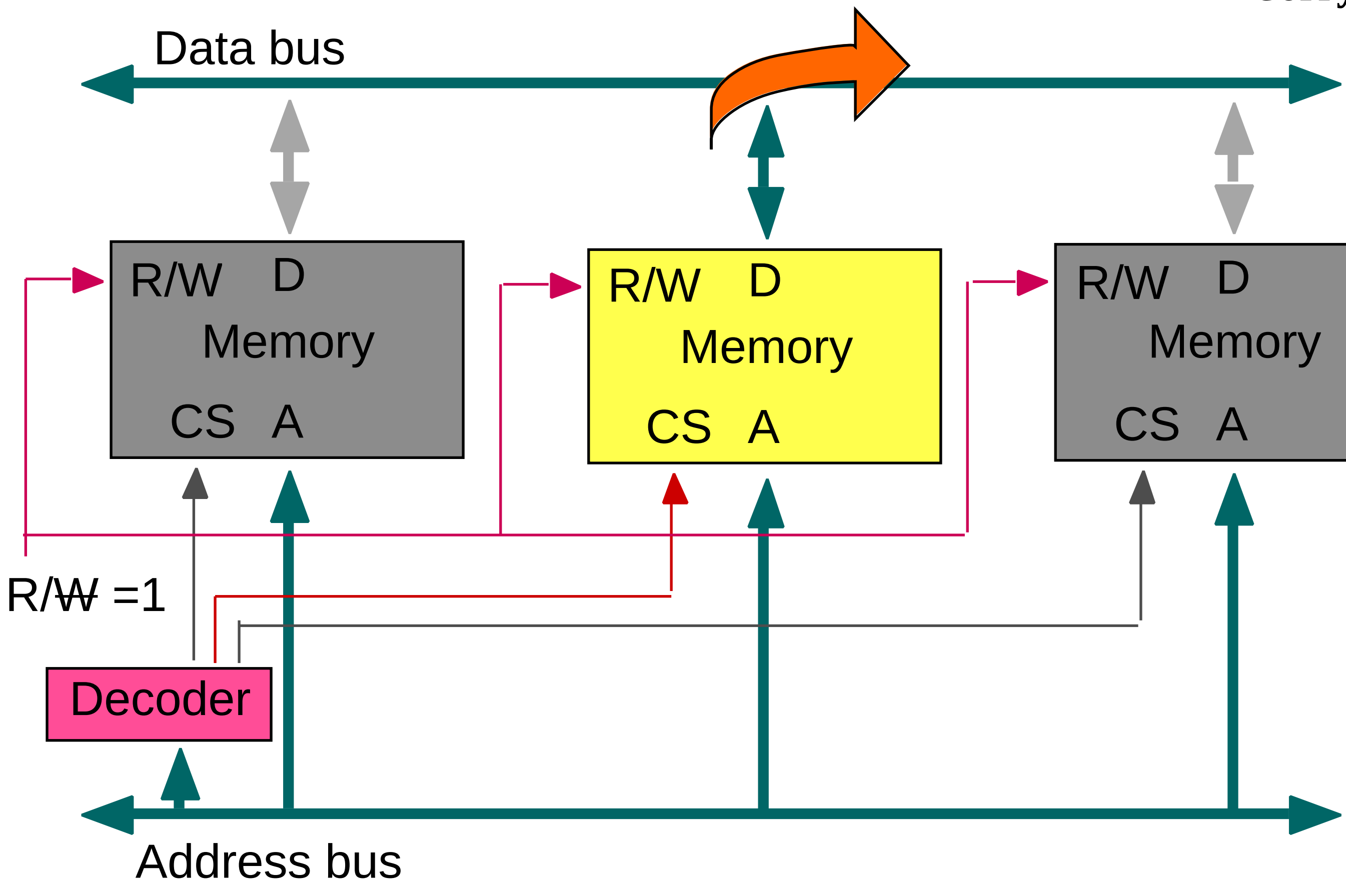
Gorry Fairhurst



ROM first	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ROM last	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1
RAM first	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RAM last	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1
Cont. first	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Cont. last	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

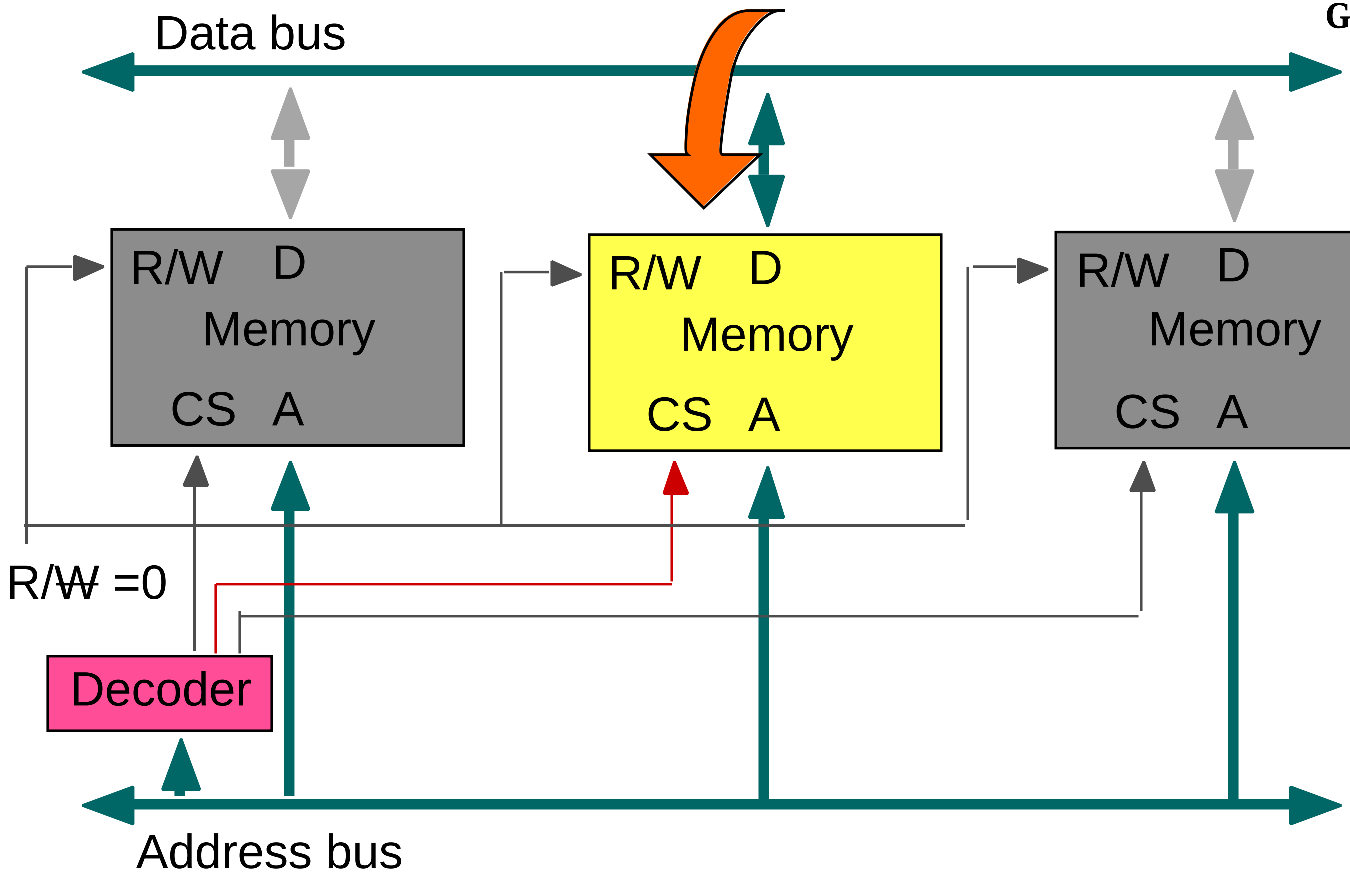
Reading a Location in Memory

Gorry Fairhurst



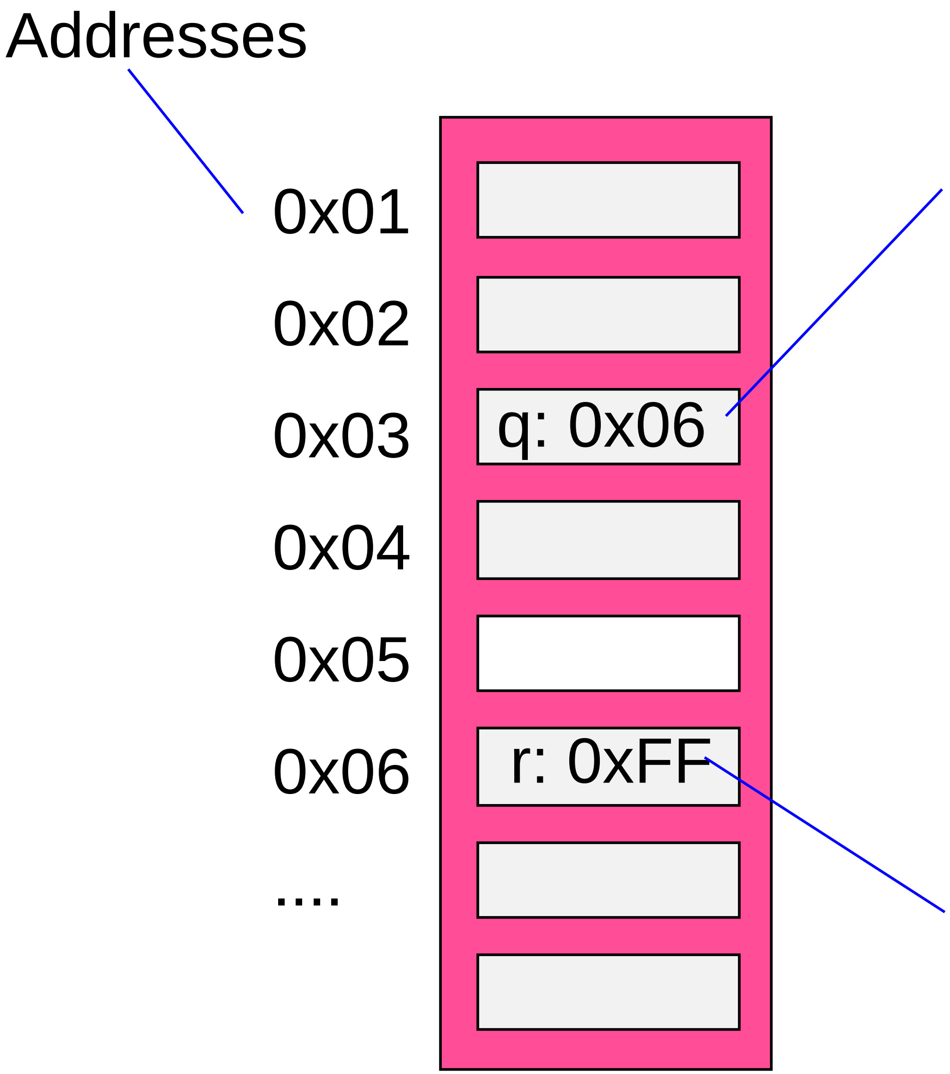
Writing a Location in Memory

Gorry Fairhurst



Addresses and Memory

Gorry Fairhurst



a variable labelled “q”

The value of q:
q == 0x06

The address of q:
&q == 0x03

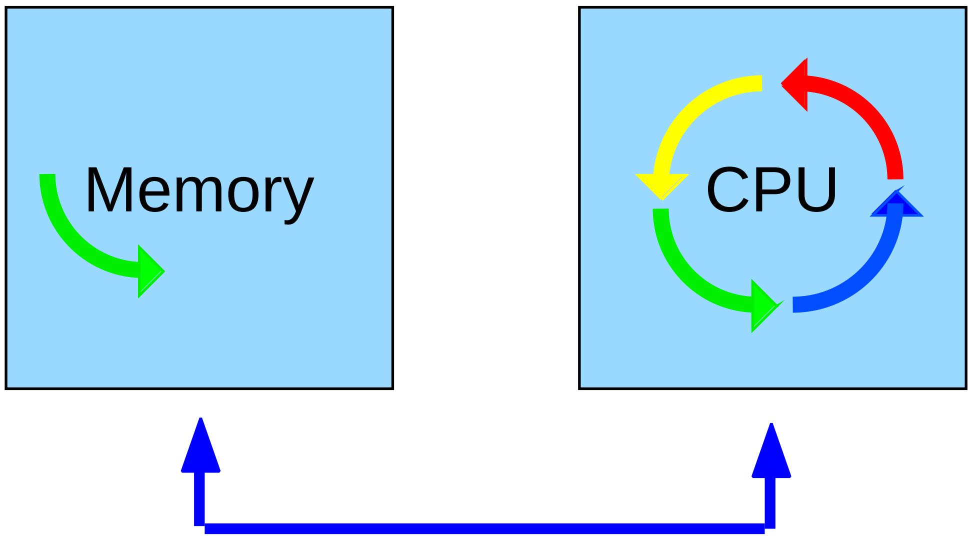
q used as a pointer:
*q == r == 0x06
(in assembler *q==(q))

a variable labelled “r”
r == 0xFF
&r == 0x06

CPUs are faster than Memory

Gorry Fairhurst

CPUs operate ***much*** faster than memory does!



Accessing memory is a severe ***bottleneck***

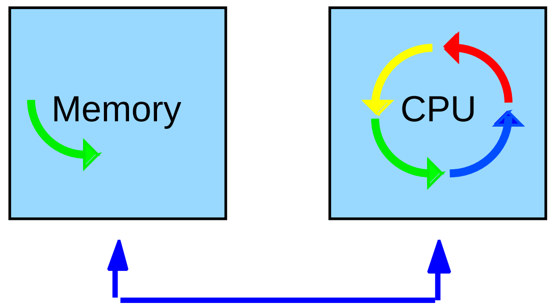
Gorry Fairhurst

Three fortunate observations:

Programs may be optimised
Using registers instead of memory to reduce **data** transfer

Programs often execute loops of instructions
The same **instructions** are often used many times

Programs usually read and write consecutive locations
Data are often stored in words, or larger groups of bytes



Gorry Fairhurst

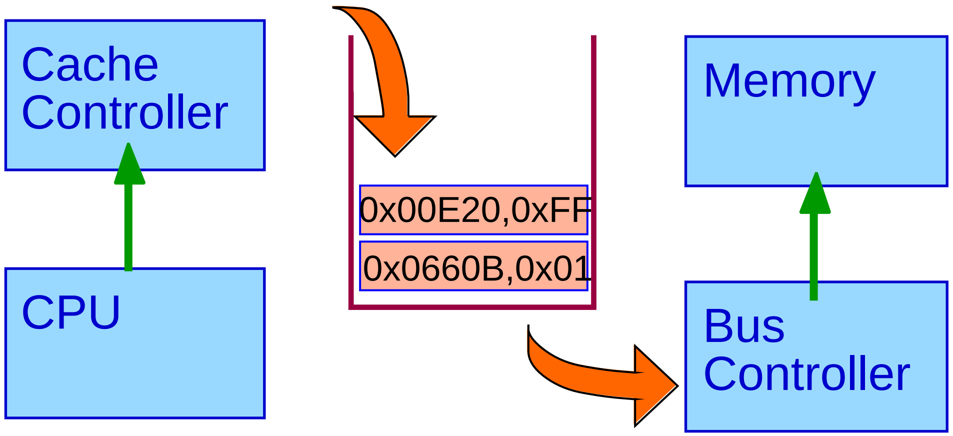
Caches can do three things to improve performance:

Recently **read** data kept in fast memory for quick re-use

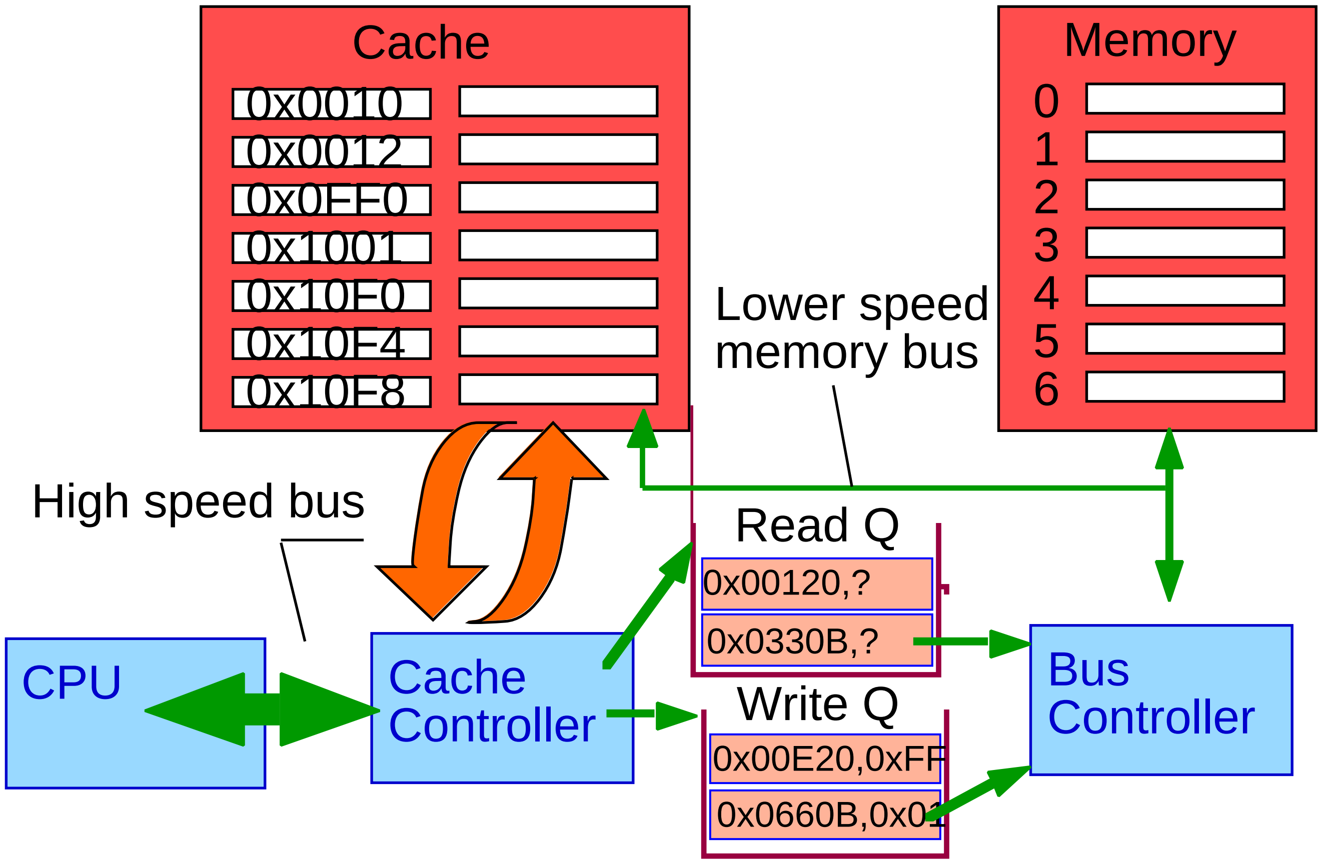
They **read** locations from memory before they are required

They defer **writing** data to memory
Allowing program to continue while memory catches up

The memory write queue

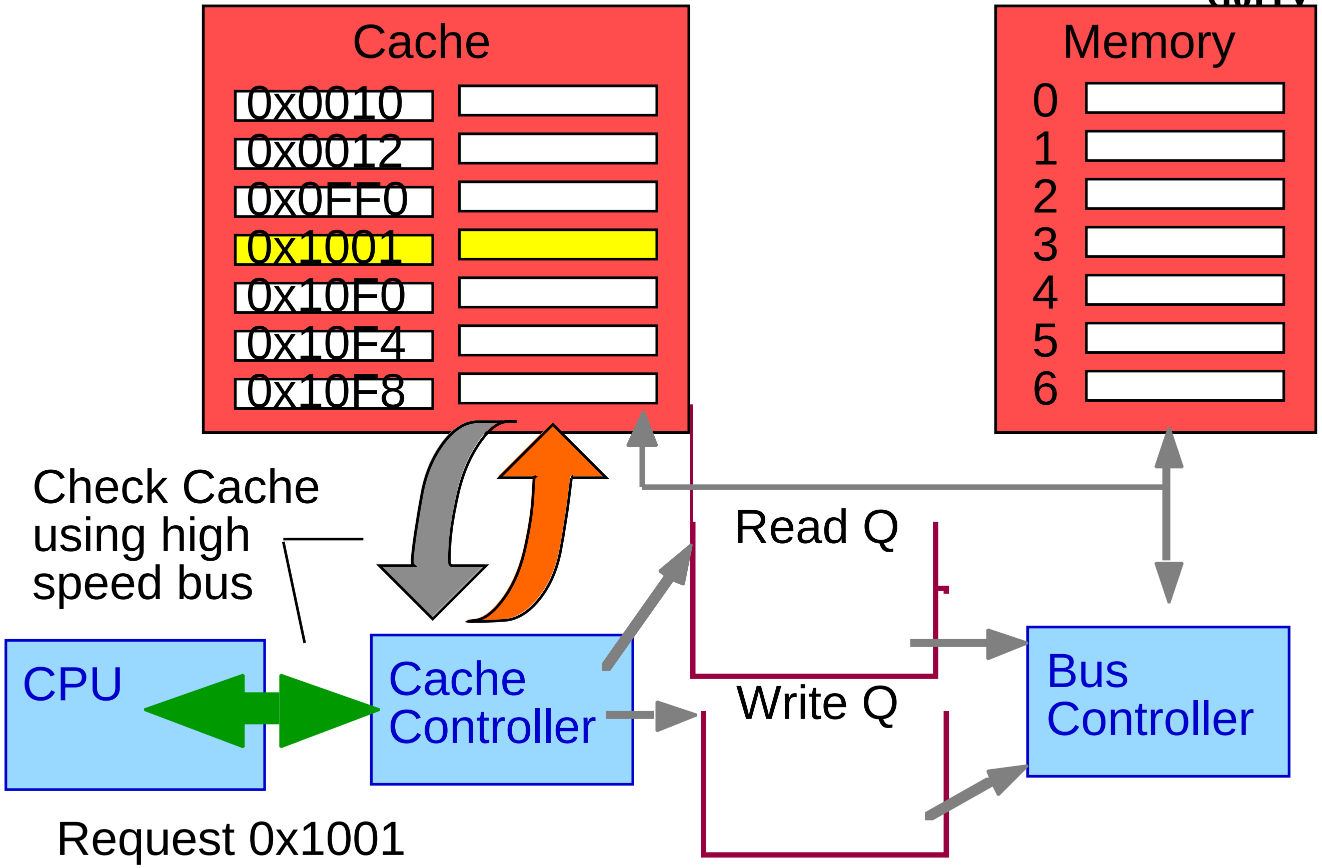


Gorry Fairhurst



Read from Memory (in Cache)

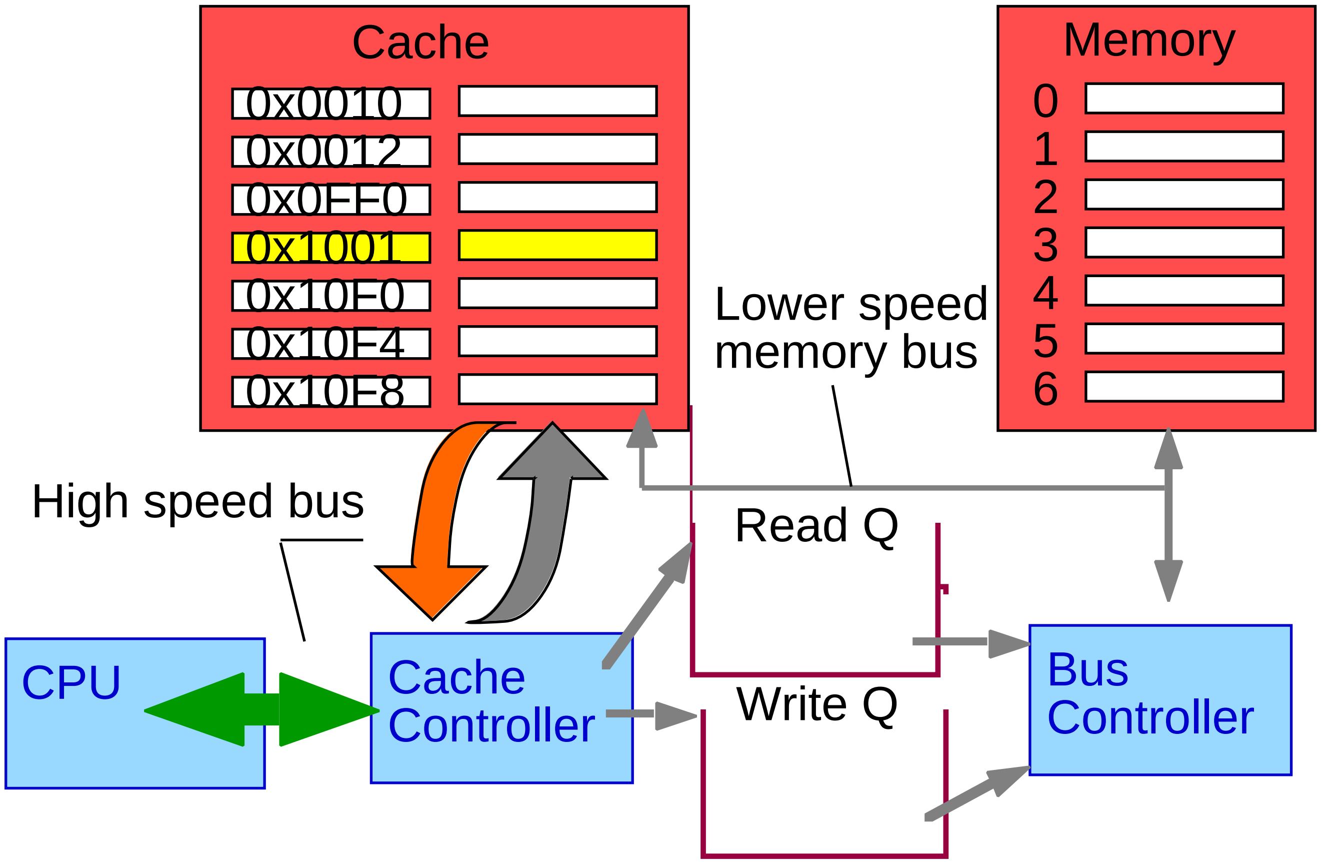
Gorry Fairhurst



Part 1: Before looking at RAM, check the locations stored in the Cache

Read from Memory (in Cache)

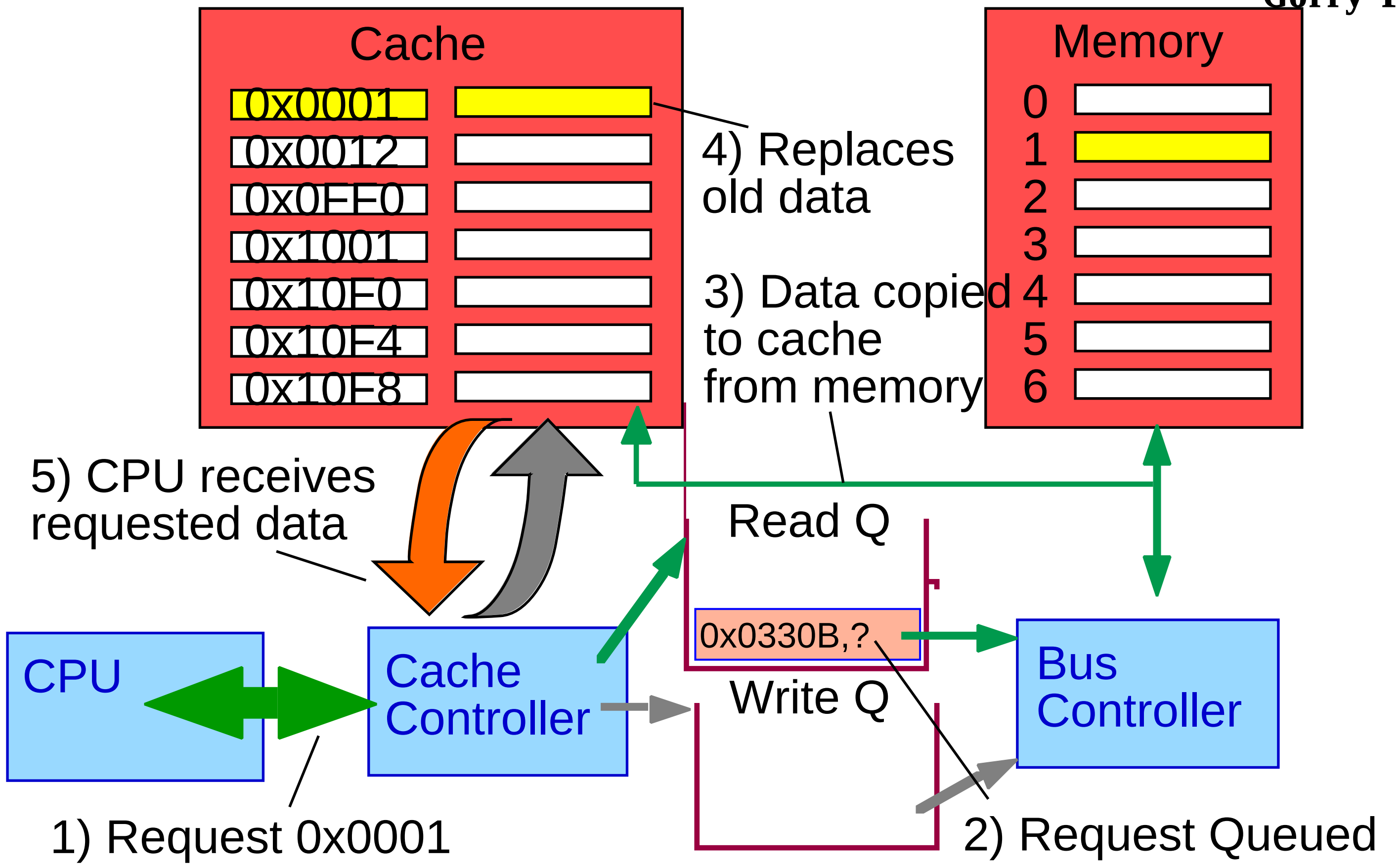
Gorry Fairhurst



Part 2: If the location is in the Cache, use the value stored in the Cache

Read from Memory (Not in Cache)

Gorry Fairhurst



Part 2: If the location is NOT in the Cache, fetch value from RAM (also store in Cache)