

MSc Course in Information and Communications Technologies

Session 1: Inside the Computer

Keith Cheverst
Computing Department

<http://courses.landmarc.net/401>

Overview

- Today we're going to talk about...
 - How computers work
 - Key concepts
 - Computer arithmetic (binary)
 - Computer logic
 - How programs/applications map onto electronic switches...

I will go very fast, so I expect you to ask questions if you have them!

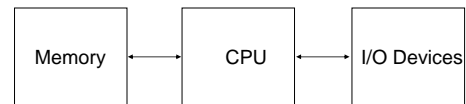
Concepts and Terminology

- Key concepts in computing
 - Program or Algorithm
 - Memory
 - Processor
 - I/O device



Basic Computer Architecture

- The components of a computer are connected together via one or more buses.



The Software-Hardware Hierarchy

Applications Software (e.g. packages)

System Software (e.g. operating systems)

Computer Hardware
(e.g. CPU, memory, I/O devices)

Implications for performance...

Why Study Binary

- In a digital computer all information is represented using the binary number system.
- Reason: because it is easy to build electronic circuits which operate on the basis of two states, either on or off.

Number Systems

- Labelling the columns.

Hundreds	Tens	Units
7	1	1

Number Systems

- Labelling the columns.

Hundreds	Tens	Units
7	1	1
10^2	10^1	10^0
7	1	1

Number Systems (1)

- Labelling the columns.

Hundreds	Tens	Units
7	1	1
10^2	10^1	10^0
7	1	1

$$711 = 7 * 100 + 1 * 10 + 1 * 1$$

Number Systems (2)

- Arithmetic

Hundreds	Tens	Units
7	1	1
	9	9 +
<hr/>		
<hr/>		

Number Systems (3)

- Arithmetic

Hundreds	Tens	Units
7	1	1
	9	9 +
<hr/>		
8	1	0

Number Systems (4)

- Range

Hundreds	Tens	Units
	9	9

In general: Range is $10^N - 1$

The Basics of Binary

- Let's consider an 8 digit binary number
- Label the columns starting at the right in increasing powers of 2

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128's	64's	32's	16's	8's	4's	2's	1's

#1

The Basics of Binary

- Let's consider an 8 digit binary number
- Label the columns starting at the right in increasing powers of 2

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128s	64s	32s	16s	8s	4s	2s	1s
1	0	1	0	1	0	1	1

= 128 + 32 + 8 + 2 + 1 = 171

Some Binary Bits and Pieces

- Terminology
 - Each Binary digit is called a bit.
 - Eight binary digits are called a byte.
 - The left most bit is called the most significant bit (or MSB).
 - The right most bit is called the least significant bit (or LSB).
- Range
 - Normally, range is $0 \dots 2^N - 1$

#2

Binary Arithmetic

- Rules for binary arithmetic are the same as for decimal arithmetic.

00101010	= 42
00011111	= 31
<hr/>	

#3

Binary Arithmetic

- Rules for binary arithmetic are the same as for decimal arithmetic.

00101010	= 42
00011111	= 31
<hr/>	
01001001	= 73

Converting 2 binary...

Subtraction

- Convert 135 to base 2

Subtraction

- Convert 135 to base 2

		MSB	
135	we can 1	↓	subtract 128 (r7)
7	we cant 0		subtract 64
7	we cant 0		subtract 32
7	we cant 0		subtract 16
7	we cant 0		subtract 8
7	we can 1		subtract 4 (r3)
3	we can 1		subtract 2 (r1)
1	we can 1	LSB	subtract 1 (r0)

Look and Guess (subtraction)

- Convert 135 to base 2

Look and Guess (subtraction)

- Convert 135 to base 2

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128's	64's	32's	16's	8's	4's	2's	1's
1	0	0	0	0	1	1	1

Representation of Characters

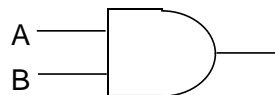
- Characters such as 'A' must also be represented.
- Standard representation (ASCII code).

01000001	A
01000011	B

- ◆ Also have to represent numbers as characters, e.g. '7' but note these aren't the same as 7.

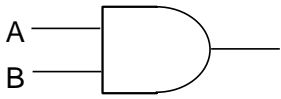
Logic Gates

- AND



Logic Gates

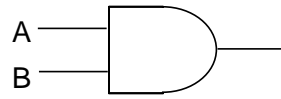
- AND



Input		Output
A	B	
F	F	
F	T	
T	F	
T	T	

Logic Gates

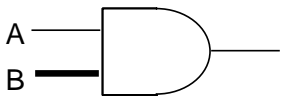
- AND



Input		Output
A	B	
F	F	F
F	T	
T	F	
T	T	

Logic Gates

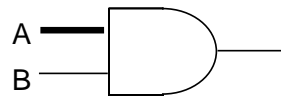
- AND



Input		Output
A	B	
F	F	F
F	T	F
T	F	
T	T	

Logic Gates

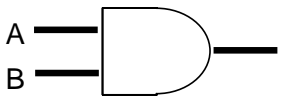
- AND



Input		Output
A	B	
F	F	F
F	T	F
T	F	F
T	T	

Logic Gates

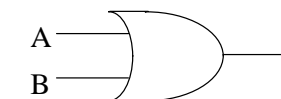
- AND



Input		Output
A	B	
F	F	F
F	T	F
T	F	F
T	T	T

Logic Gates

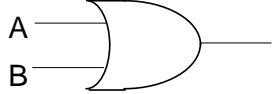
- OR



Input		Output
A	B	
F	F	
F	T	
T	F	
T	T	

Logic Gates

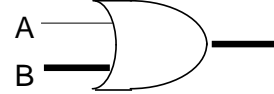
- OR



Input		Output
A	B	
F	F	F
F	T	
T	F	
T	T	

Logic Gates

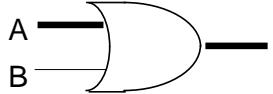
- OR



Input		Output
A	B	
F	F	F
F	T	T
T	F	
T	T	

Logic Gates

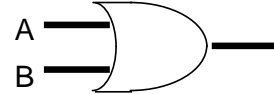
- OR



Input		Output
A	B	
F	F	F
F	T	T
T	F	T
T	T	

Logic Gates

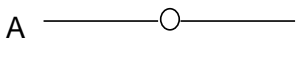
- OR



Input		Output
A	B	
F	F	F
T	F	T
F	T	T
T	T	T

Logic Gates

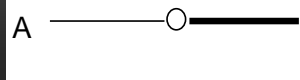
- NOT



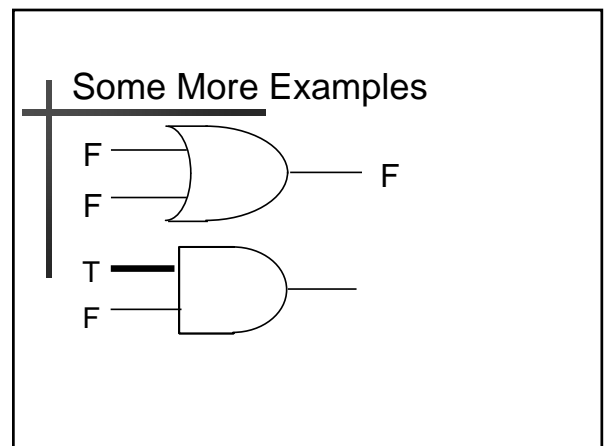
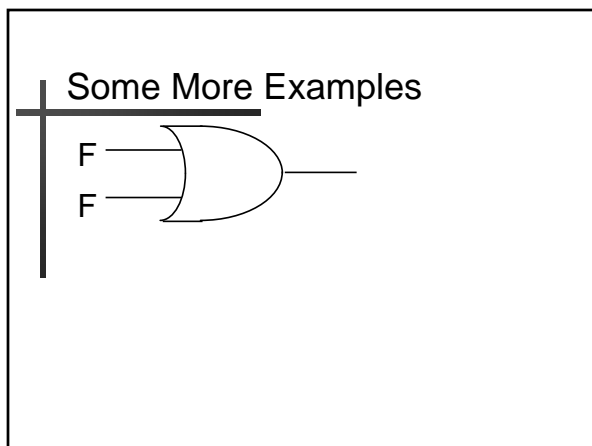
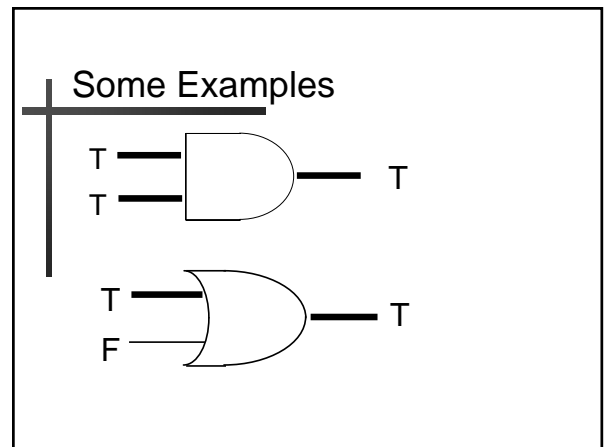
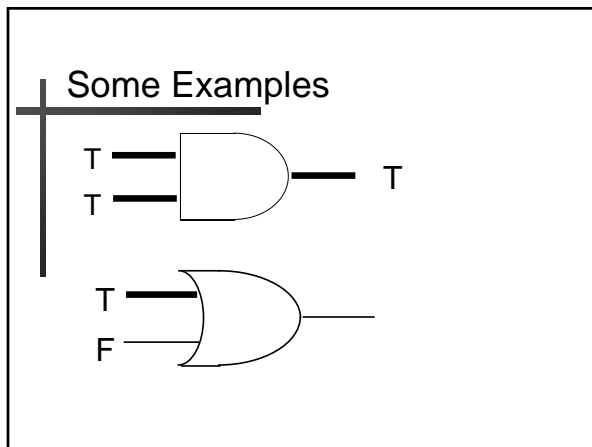
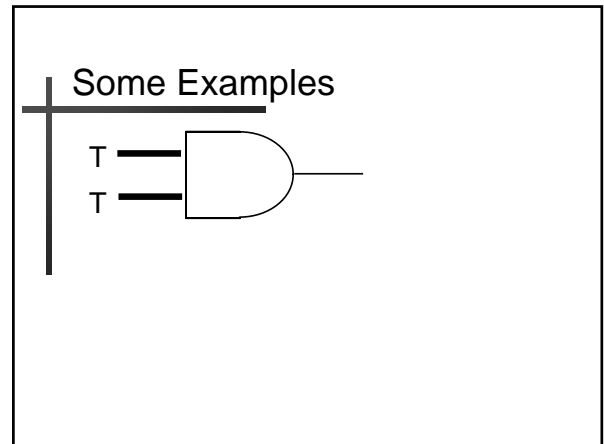
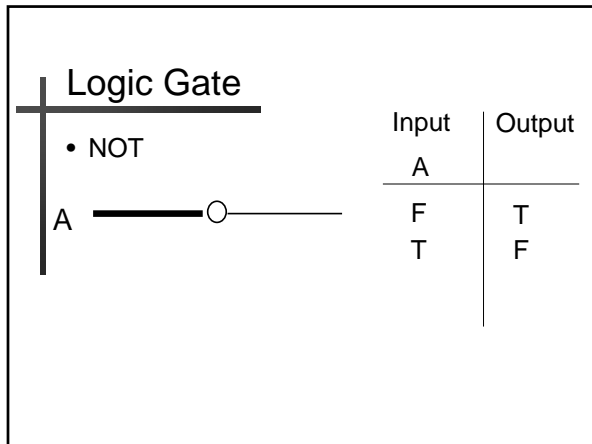
Input	Output
A	
F	
T	

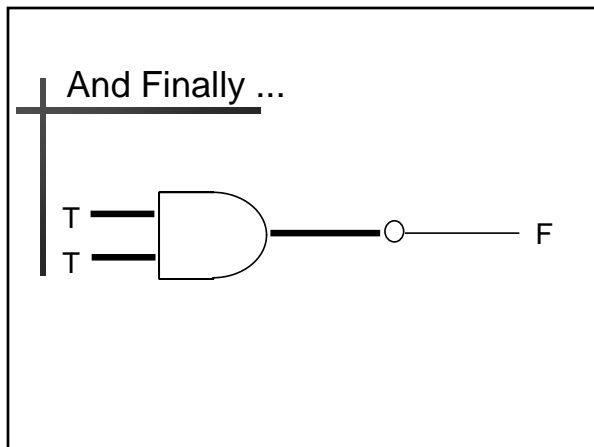
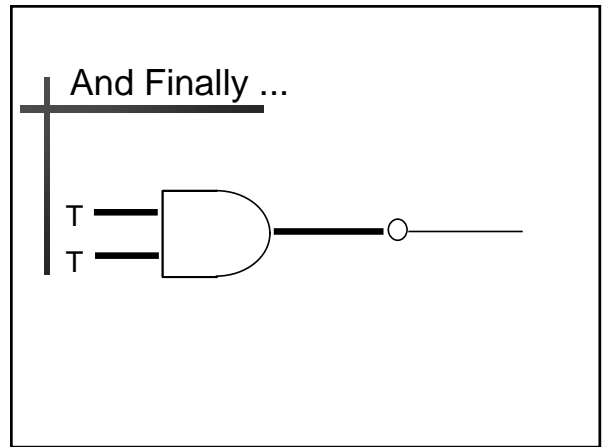
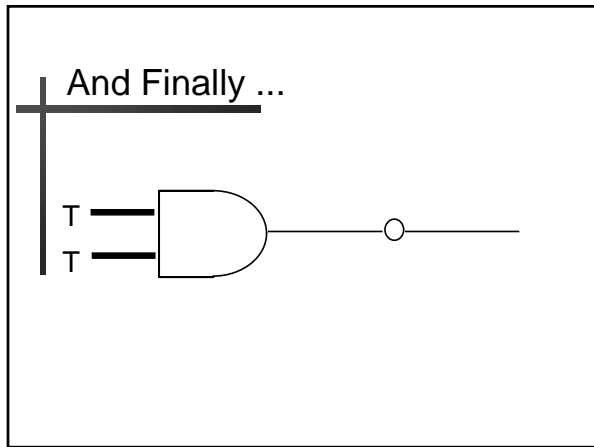
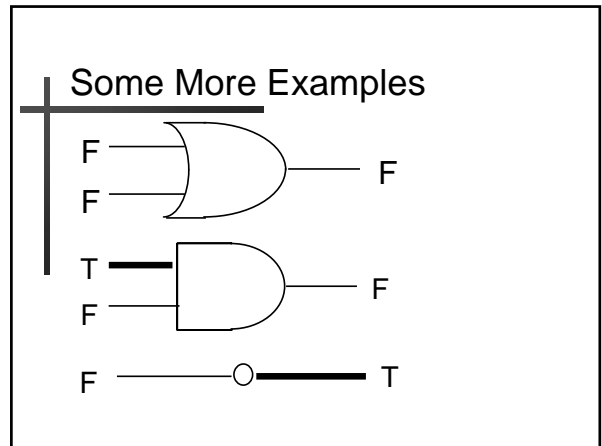
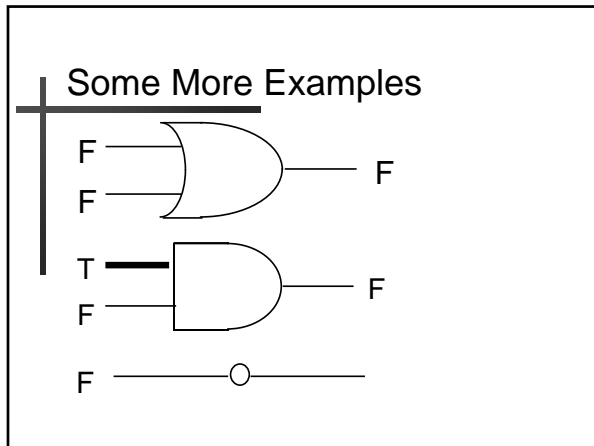
Logic Gates

- NOT



Input	Output
A	
F	T
T	





Using Logic Gates

- Logic gates can be used to store and manipulate binary information if we replace T and F with 1 and 0.

Input		Output
A	B	
0	0	0
0	1	0
1	0	0
1	1	1

Diagram: An AND gate with inputs labeled 'A' and 'B' and one output line.

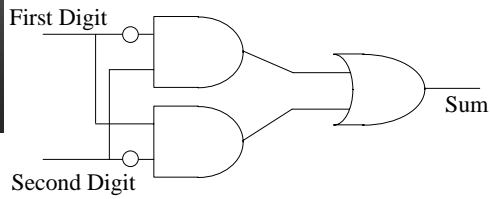
Addition Using Logic Gates

00101010 = 42
00011111 = 31

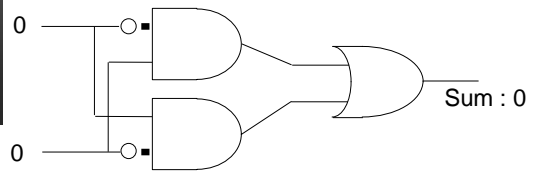
Designing the Circuit – the Sum

00101010 = 42
00011111 = 31

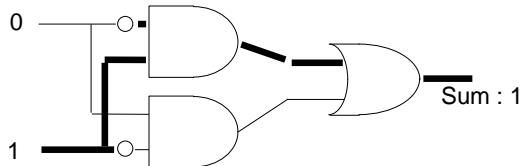
Calculating the Sum



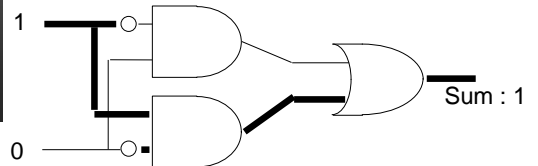
Calculating the Sum



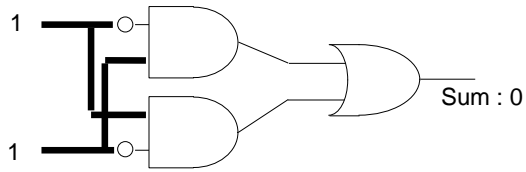
Calculating the Sum



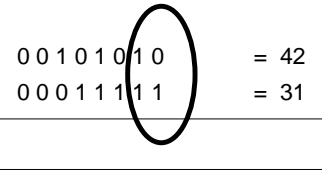
Calculating the Sum



Calculating the Sum

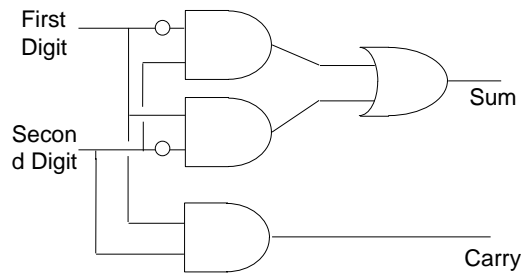


Designing the Circuit – the Carry

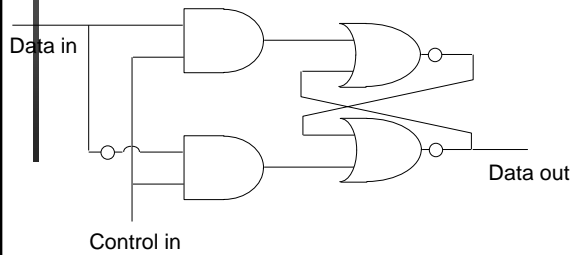


Calculating the Carry

A Half Adder

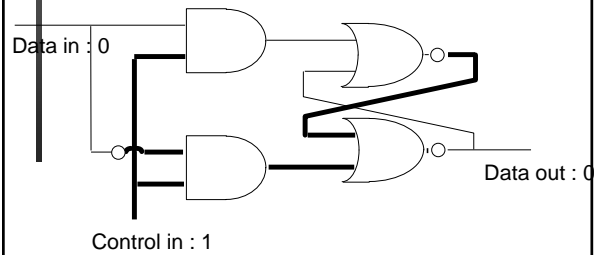


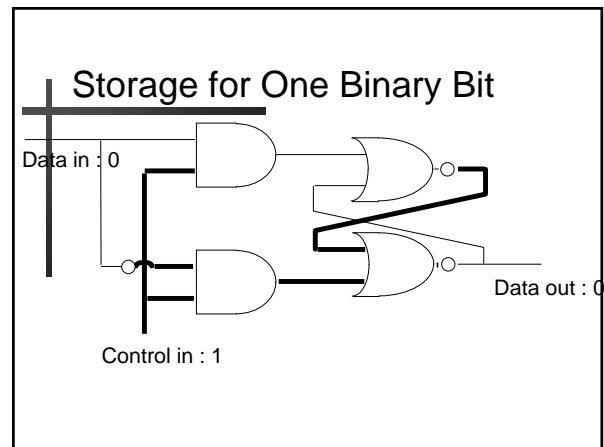
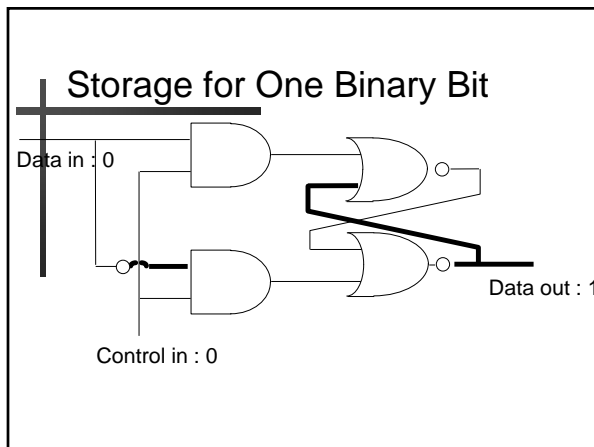
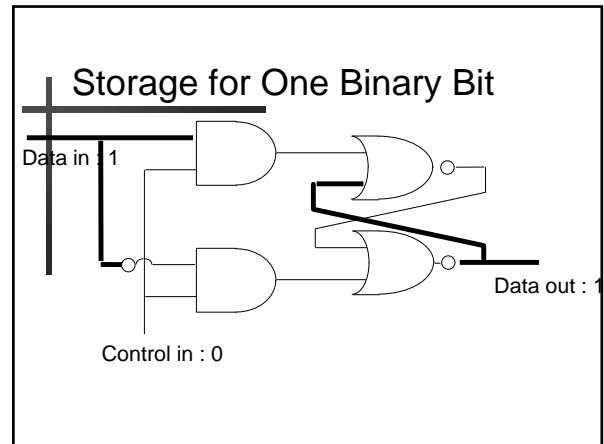
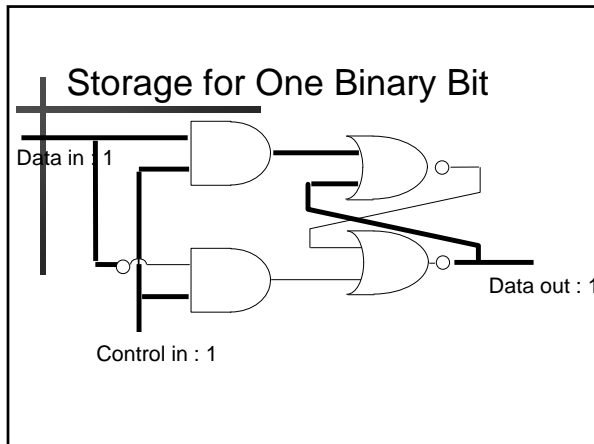
Storage for One Binary Bit



- This type of circuit is called a flip-flop.

Storage for One Binary Bit





Implementation of Logic Gates

- Logic gates can be easily implemented using switches.

- ◆ A transistor is a switch

Collector

Base

Emitter

Implementation of Logic Gates

- Logic gates can be easily implemented using switches.

- ◆ A transistor is a switch

Collector

Base

Emitter

Implementation of Logic Gates

- Logic gates can be easily implemented using switches.



- ◆ A transistor is a switch

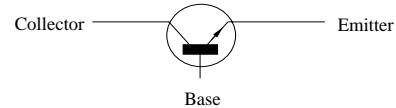


Implementation of Logic Gates

- Logic gates can be easily implemented using switches.



- ◆ A transistor is a switch

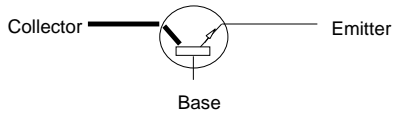


Implementation of Logic Gates

- Logic gates can be easily implemented using switches.



- ◆ A transistor is a switch

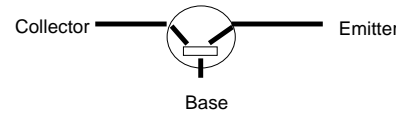


Implementation of Logic Gates

- Logic gates can be easily implemented using switches.

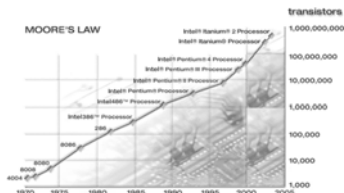


- ◆ A transistor is a switch



Implementation of Logic Gates

- Large numbers of transistors can be built onto a single chip
 - How many transistors on Pentium 4?
- i.e. large numbers of gates can be built onto a single chip.



Time for break...

- Please be back in 15 mins...

Putting it all together...

- Assembling the Components...

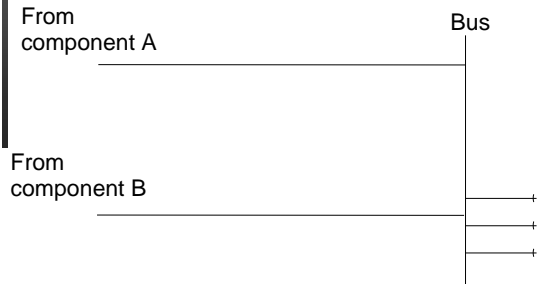
The Key Components

- Flip-flops, grouped together to form fixed size registers.
- Adders, grouped together to operate on fixed size numbers.
- A section of main memory.
- I/O devices.
- Design decision: the word size of our computer.

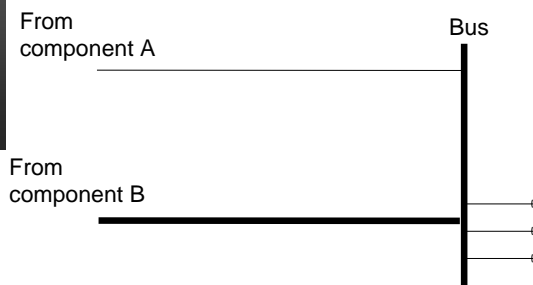
Data Buses

- Need to be able to move information between the components.
- Data Buses: a collection of wires which connect together the components.
- Width of the bus is normally equal to the word size.
- Access to the bus is controlled by placing AND and OR gates between each component and the bus.

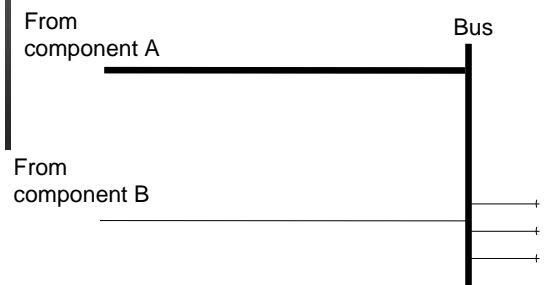
Data Buses ... contd.

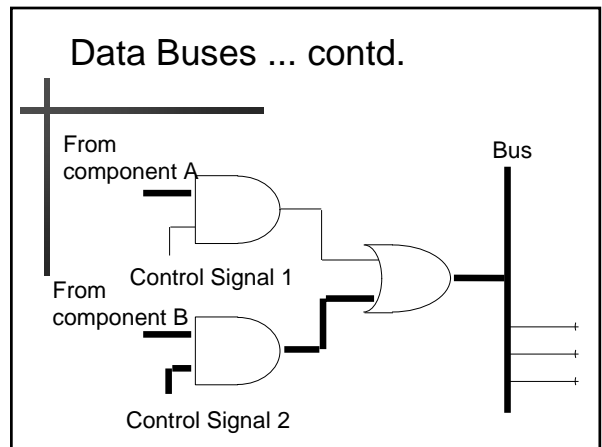
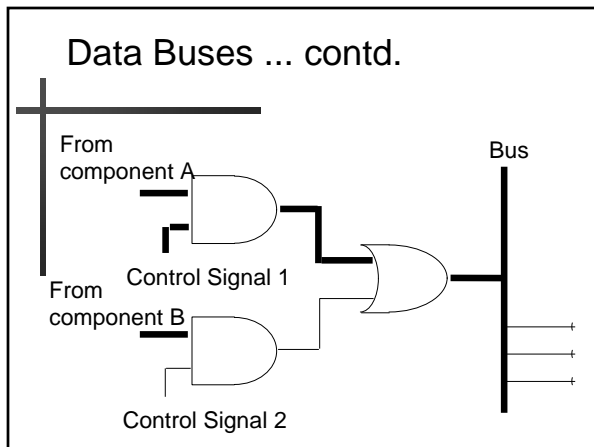
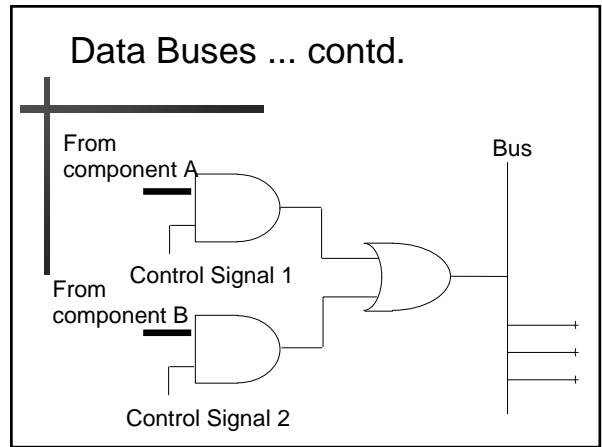
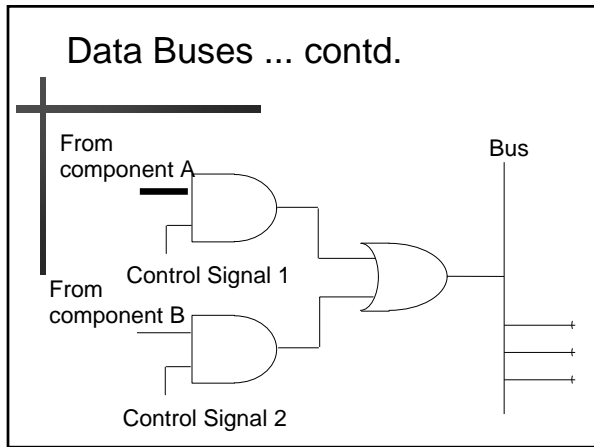
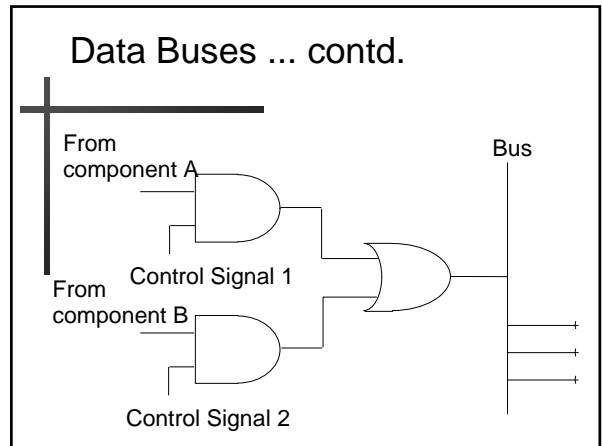
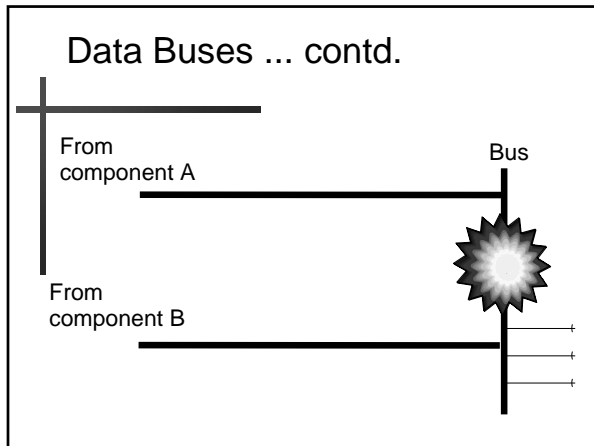


Data Buses ... contd.



Data Buses ... contd.





Control Logic

- Many of the components need control signals to regulate their activity.
- Control signals are produced by a computer component called the control logic.
- The control logic incorporates a clock which produces signals at a regular rate.
- Clock signals are used to synchronise the components.

A Complete Stored Program Computer

- Note:
 - This next bit is here to give you an appreciation/impression of how things work at a low level – **I don't expect you to be able to write your own microcode programs!**

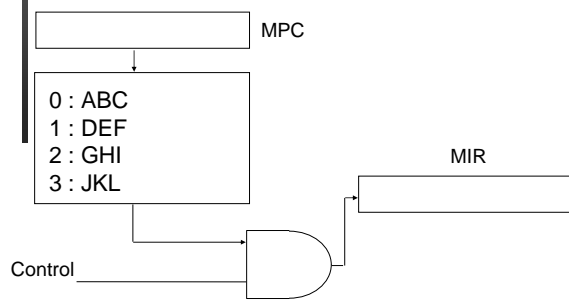
Features: Registers and Buses

- 8 Registers
 - A, B, C and D General purpose
 - MPC Microprogram counter
 - MIR Microinstruction register
 - MDR Memory data register
 - MAR Memory address register
- Data in the registers can be transferred onto the buses using control signals.

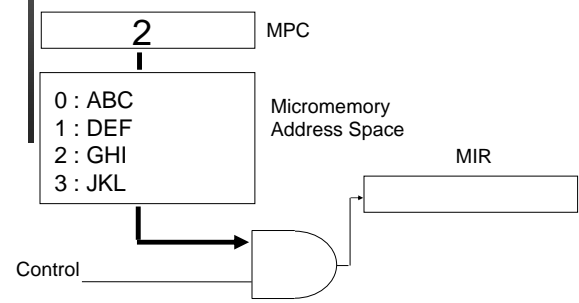
Features : Micro memory (1)

- Each cell contains 22 bits, cells addressed as 0 .. 1023.
- Contents of micro memory can be accessed by placing a valid address in the MPC register.
- The value at this address is copied into the MIR on the appropriate signal.

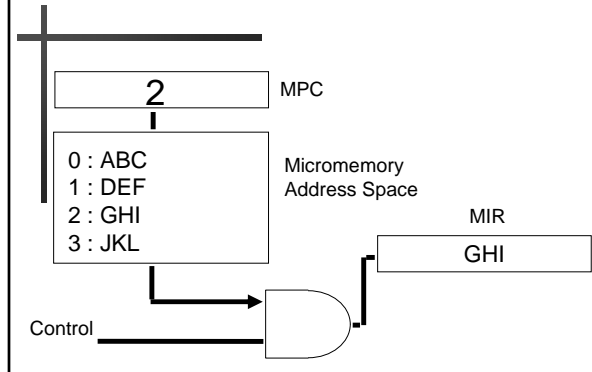
Features: Micro memory



Features: Micromemory



Features: Micromemory



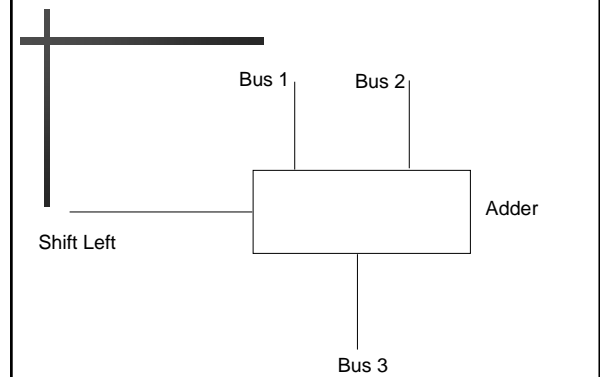
Features: Main Memory

- Each cell contains 16 bits, cells addressed as 0 .. 4095
- Contents of main memory can be accessed by placing a valid address in the MAR.
- If control signal 15 is set then the value held at this address is copied into the MDR.
- If control signal 16 is set then the value held in the MDR is copied into the memory at this address.

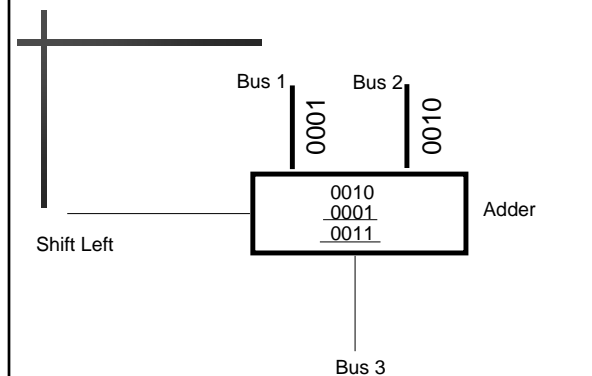
Features of Our Computer: The Adder

- The adder performs arithmetic on the values on BUS1 and BUS2 and places the result on BUS3.
- Setting control signal 7 causes the adder to perform a subtraction rather than an addition.
- If control signal 8 is set the adder shifts the result of its arithmetic (multiplies by 2).

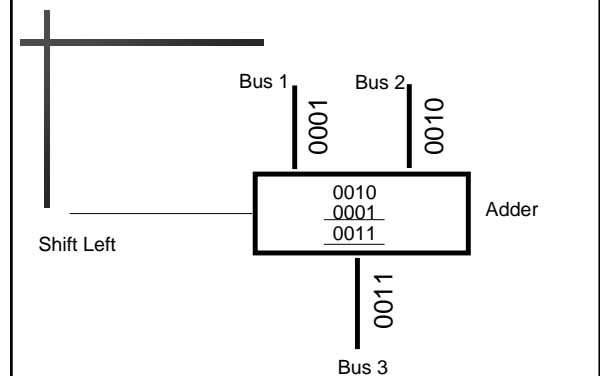
Features: The Adder

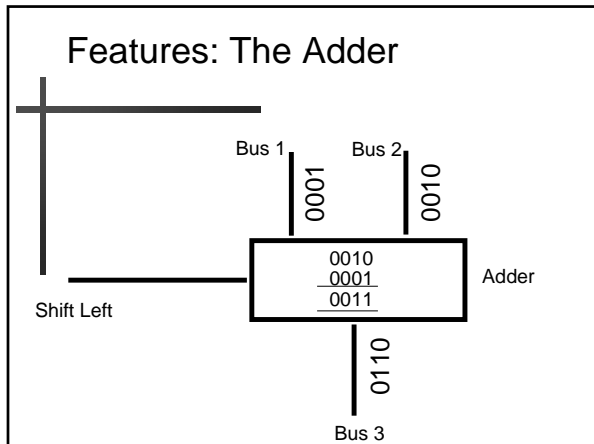


Features: The Adder



Features: The Adder





- ### Features: Test, Signals and Others
- Machine can perform two tests:
 - Test if register A contains zero.
 - Test if the MSB of A is set.
 - Control signals 5 and 17 place a 1 on BUS2.
 - Control signal 18 places the 10 most significant bits of the MIR onto BUS2.

- ### Programming Our Computer
- The behaviour of our computer depends entirely on the values of the control signals.
 - Question: what causes the signals to be turned on and off ?

- ### Programming Our Computer
- The behaviour of our computer depends entirely on the values of the control signals.
 - Question: what causes the signals to be turned on and off ?
 - Answer: the register MIR contains a microinstruction which specifies which lines are to be switched on.
- | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 |
| | | | | | | | | | | | | | | | | | | | | | |

- ### A Sample Micro program
- A sample micro program to multiple the contents of register C by register A.
 - Results are placed in main memory cell 1.
- ```

module multiply
{
 set MDR to 0
 repeat A times
 add C to MDR
 move 1 to MAR and tell the main memory to write
}

```

### Sketch of the Microprogram:

| Micro Memory address | Microinstruction                    |
|----------------------|-------------------------------------|
| 0                    | 0 + 0->MDR; MPC + 1 -> MPC;         |
| 1                    | MPC + TESTZERO(A) -> MPC            |
| 2                    | 5 -> MPC                            |
| 3                    | C + MDR -> MDR; MPC + 1 -> MPC;     |
| 4                    | A - 1 -> A; 0 + 1 -> MPC            |
| 5                    | 0 + 1 -> MAR; write; MPC + 1 -> MPC |

## The Microprogram in Full

|                     |   | Microinstruction |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |
|---------------------|---|------------------|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|----|----|----|----|----|
|                     |   | 1                | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 |
| Micromemory address | 0 |                  |   |   |   |   |   |   |   |   |    |    |    |    |    | 1  |    |    | 1  |    |    |    | 1  |
|                     | 1 |                  |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    | 1  |    |    | 1  |
|                     | 2 |                  |   |   |   |   |   |   | 1 |   | 1  |    |    |    |    |    |    |    |    | 1  |    |    |    |
|                     | 3 |                  |   | 1 |   |   | 1 |   |   |   |    |    |    | 1  |    |    |    |    | 1  |    |    |    | 1  |
|                     | 4 | 1                |   |   |   | 1 |   | 1 |   | 1 |    |    |    |    |    |    |    |    |    | 1  |    |    |    |
|                     | 5 |                  |   |   |   | 1 |   |   |   |   |    |    |    |    | 1  |    | 1  | 1  |    |    |    |    | 1  |

## Evaluation of Microinstructions

- Note: Each microinstruction takes one clock cycle to execute.
- Writing programs in microcode is very difficult:
  - cf code required to multiply two numbers
  - each instruction performs only a very small task.
  - instructions are closely tied to machine architecture.
  - instructions must be coded in binary.

## Providing More Support

- If we are going to write big programs we need to provide programmers with easier to use and more powerful instructions.
- Simplest languages which offer such support are called machine languages.
  - still at a very low level but much easier to use than microcode.
  - they introduce limited abstractions over the machine's hardware, e.g. the accumulator.

## An Example of a Very Simplified Machine Language...

| Op Code | Operation | Explanation     |
|---------|-----------|-----------------|
| 0001    | LOAD      | M -> ACC        |
| 0010    | STORE     | ACC -> M        |
| 0011    | ADD       | ACC + M -> ACC  |
| 0100    | SUBTRACT  | ACC - M -> ACC  |
| 0101    | MULTIPLY  | ACC * M -> ACC  |
| 0110    | DIVIDE    | ACC / M -> ACC  |
| 0111    | JUMP      | Jump to cell M  |
| 1000    | JUMPZERO  | Jump if ACC = 0 |
| 1001    | JUMPMSB   | Jump if msb = 1 |

## An Example...

| Addr | Instr (op code operator) | Meaning                         |
|------|--------------------------|---------------------------------|
| 1    | 0001 0101                | LOAD 5: Contents 5 -> ACC       |
| 2    | 0101 0101                | MULT 5: ACC * Contents 5 -> ACC |
| 3    | 0010 0101                | STORE 5: Contents of Acc -> 5   |
| 5    | 0003                     |                                 |

- Calculates  $x*x$  where  $x$  is initially in location 5.
- Compare this with its microcode program.

## Supporting Machine Language Instructions

- Early machines were built to execute machine language instructions directly:
  - this is expensive in design terms.
  - lots of duplication since many instructions have common functionality.
- By contrast, microinstructions are much simpler and machines may be built to execute microinstructions quite cheaply.

## Supporting Machine Language Instructions

- Solution ?
- Provide a means of converting from machine language instructions to microcode.
- This can be done in software; the program is called a microprogrammed interpreter.

## A Microprogrammed Interpreter

- A microprogrammed interpreter is supplied with every computer.
- The interpreter is fixed in the machine's micromemory and maps machine language instructions in main memory into microinstructions.
- The interpreter is normally the only algorithm ever microprogrammed - programmers normally only ever see machine language.

## Basic Algorithm for a Microprogrammed Interpreter

```
procedure interpreter is
begin
 while (true) do
 fetch next machine language from address in B
 add 1 to contents of B
 decode the instruction
 execute the instruction
 end while
end
```

## We don't work in machine code

- At least rarely, we develop tools to make life easier
  - Common tasks run in low level 'system software' (the operating system, e.g. Windows XP, Linux, Mac OS)
  - Programs each get a share of the central processing unit (CPU), allowing many programs to run effectively 'simultaneously'
  - We develop more abstract (English like) languages for writing programs
  - Compilers convert such languages into Machine Code
  - *What might the tradeoffs be?*

## Performance

- ◆ Many factors affect the overall performance of a computer:
  - the processor clock speed.
  - the capacity and speed of the memory.
  - the number of bits in each stored word.
  - the features provided by the instruction set.
- Performance is typically measured in either MIPS or MFLOPS - not very useful.
  - **TASK:** Try to find out how the performance of a modern PC has increased from that of a PC ten years ago...

## The Role of The Clock

- The clock period is particularly important in determining the performance of a computer.
- Clock period = time allocated for each basic internal operation to the processor.
- E.g. clock frequency of 1000 MHz (1 GHz) = period of 1 ns. If one instruction per cycle then speed = 1000 MIPS (1 GIP).
- Speeding up the clock speeds up the processor.
- But, practical factors limit the speed at which processors can execute.

## Learning Outcomes

- What should you learn from today?
  - A basic understanding of what a computer is, and,
  - Appreciation of how we get from high level code to actual computations – i.e. switches switching!
  - Factors effecting the performance...
    - Clock speed etc.
    - Note: where you put data matters (e.g. in main memory vs. secondary storage\_
      - (so we can think more widely next week about networks of computers and distribution)