University Of Portsmouth
BSc (Hons) Computer Science
First Year

Architecture and Operating Systems - Computer
M30943
September 2022 - May 2023
20 Credits

Thomas Boxall
up2108121@myport.ac.uk

Contents

Page 1

Introduction to Module

📅 26-09-22   🕐 16:00   🎓 Farzad   📍 RB LT1

Division of the Module

This module is split into two parts: computer (this part) which is worth 70% and maths (the other part) which is worth 30%. The two parts are run completely independently of each other. The only time they come together is when the final overall score is calculated. There are two separate Moodle pages (one for Computer and one for Maths)

Computer Module assessments

For the Computer section of the module, there are two assessments. One is in January 2023, which will be a Computer Based Test (covering content taught in the first teaching block). It is worth 30% of the over module score. The second is in the May/June 2023 assessment period. It will be computer based. This assessment will be worth 40% of the overall module score.
Both assessments are closed book however a formula sheet will be provided for the January assessment. Nothing is provided for the May/June assessment.
The pass mark for the entire module is 40%, this score is generated from all the computer assessments AND all the maths assessments.

Module structure

There will be a one hour lecture per week, where content is introduced to us. This will be delivered using worksheets for the first 10 weeks.
There will also a practical session each week where the cohort is split into smaller groups. These sessions will be a chance to practice the ideas introduced in the lectures. There will be more members of staff around at the practical sessions to help out.

> **More Information on Practical Sessions**
> There are practical session guidelines available in the induction slides or on Moodle.

> **Content in each Week**
> There is a teaching plan on Moodle which outlines the content covered each week as well as the weeks in which the exams will be held.

Page 2

Binary Arithmetic

📅 26-09-2022   🕐 16:15   🎓 Farzad   📍 RB LT1

Number Systems

There are a number of different number systems and different methods to convert between them.

Denary (Base 10)

Used most commonly, this is the one most people learn.

10^x	10^3	10^2	10^1	10^0
$10^x =$	1000	100	10	1
	4	2	5	1

The total of the numbers above would be calculated in the following way:
$4251 = (1000 \times 4) + (100 \times 2) + (10 \times 5) + (1 \times 1)$
Denary is also known as base 10, this means each column can have one of ten possible values (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)

Binary (Base 2)

This is base 2, this means each column can have one of two possible values (0, 1). The columns are also different. Moving from right to left, the columns double each time.

2^x	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
$2^x =$	128	64	32	16	8	4	2	1
	1	0	1	1	0	0	1	1

The largest value which can be stored in 8-bits of binary is 11111111_2 or 255_{10}.

Hexadecimal (Base 16)

Also known as Hex. Using this method, numbers up to 255 can be stored in two characters. This is used a lot in computing, especially in graphics and website development. Each column can have one of 16 values (1 2 3 4 5 6 7 8 9 A B C D E F). The letters are used to represent two-digit numbers as seen below.

Hex:	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Decimal:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

To calculate the value held in a Hex number, we calculate in a similar way to Denary and Binary as seen below.

16^x	16^3	16^2	16^1	16^0
$16^x =$	4096	256	16	1
	D	3	C	E

$D3CE = (13 \times 4096) + (3 \times 256) + (12 \times 16) + (14 \times 1) = 54222$

Converting Between Number Systems

Binary To Denary

Add together all the columns in which there is a 1. Using the example shown in the binary section, the total would be 179.

Denary To Binary

This is the reverse of binary to denary. Work from right to left seeing if the value will fit into the column, if it won't then mark down an zero and move onto the next.

Denary to Hex

The easiest way to do this is to go via Binary. Convert the number into binary, then split the binary into two nibbles. The values inputted in the previous step don't need to change. With the two nibbles of (4, 2, 1, 0), convert each of them back into denary, giving two individual digits, then convert each of those into Hex.

Binary Addition

Basic Rules

There are four basic rules to binary addition:

```
0 + 0 =  0
0 + 1 =  1
1 + 1 =  1
1 + 1 = 10
```

The last one (1+1) is a special case; strictly speaking, the answer is 0 with the 1 carried over. This is particularly useful in digital circuitry.

Binary Addition Example

Add 100 + 011

1. Draw out the binary addition columns

$$
\begin{array}{ccc}
  & 1 & 1 & 0 \\
+ & 0 & 1 & 1 \\
\hline
\end{array}
$$

2. Start with the right-most column and add the digits

$$
\begin{array}{cccc}
 & 1 & 1 & 0 \\
+ & 0 & 1 & 1 \\
\hline
 & & & 1 \\
\end{array}
$$

3. Move to the next column. Add those digits together. As this is 1+1 = 0 carry 1, we write a little 1 in the next column as a carry.

$$
\begin{array}{cccc}
 & {}^{1}1 & 1 & 0 \\
+ & 0 & 1 & 1 \\
\hline
 & & 0 & 1 \\
\end{array}
$$

4. Move to the next column and add that. Remember to add the carry (making the sum $1+1+0$). This results in 0 carry 1, so, again, we add a little 1 in the next column. It doesn't matter that there aren't any other numbers there to be added, we will make a column!

$$
\begin{array}{ccccc}
 & {}^{1} & {}^{1}1 & 1 & 0 \\
+ & & 0 & 1 & 1 \\
\hline
 & & 0 & 0 & 1 \\
\end{array}
$$

5. Add the final column

$$
\begin{array}{ccccc}
 & {}^{1} & {}^{1}1 & 1 & 0 \\
+ & & 0 & 1 & 1 \\
\hline
 & 1 & 0 & 0 & 1 \\
\end{array}
$$

This gives us out final answer of $110 + 011 = 1001$

Binary Multiplication

There are 4 basic rules for binary multiplication.

```
0 x 0 = 0
0 x 1 = 0
1 x 0 = 0
1 x 1 = 1
```

Binary Multiplication Example

Multiply 10×11

1. Draw out the multiplication grid as you would for a standard column multiplication with decimal numbers.

$$
\begin{array}{cccc}
 & & 1 & 1 \\
\times & & 1 & 0 \\
\hline
\end{array}
$$

2. Take the right-most digit of the bottom binary number, we will multiply it with each of the digits above and place their results directly underneath.
   $0 \times 1 = 0$, place this directly under the 0
   $0 \times 1 = 0$, place this to the left.

$$
\begin{array}{cccc}
 & & 1 & 1 \\
\times & & 1 & 0 \\
\hline
 & & 0 & 0 \\
\end{array}
$$

3. Next, move to the next digit on the bottom row, repeat the same process as before.

$$
\begin{array}{cccc}
 & & 1 & 1 \\
\times & & 1 & 0 \\
\hline
 & & 0 & 0 \\
 & 1 & 1 & \\
\end{array}
$$

4. We then add our two answer rows together, using the rules of binary addition.

$$
\begin{array}{cccc}
 & & 1 & 1 \\
\times & & 1 & 0 \\
\hline
 & & 0 & 0 \\
+ & 1 & 1 & \\
\hline
 & 1 & 1 & 0 \\
\end{array}
$$

This gives us the final answer of $11 \times 10 = 110$

Binary Subtraction

At this stage, there are four basic rules for binary subtraction. At a later stage, there will be negative numbers introduced when we look at signed binary so more rules will be introduced.

```
 0 - 0 = 0
 1 - 0 = 1
 1 - 1 = 0
10 - 1 = 1
```

Binary Subtraction Example

Subtract $110 - 001$

1. Draw out the subtraction columns as you would for a standard decimal column subtraction

$$\begin{array}{cccc} & 1 & 1 & 0 \\ - & 0 & 0 & 1 \\ \hline \end{array}$$

2. Start at the right hand most column $(0 - 1)$. This is something which we can't do, so we have to borrow 1 from the left hand column. To represent this, cross out the borrowed digit, replace with a 0 and in the current column, add a little 1 to the left.
   This leaves us with $10 - 1$, which we can do and know equals 1.

$$\begin{array}{cccc} 1 & {}^{0}\cancel{1} & {}^{1}0 \\ - & 0 & 0 & 1 \\ \hline & & & 1 \end{array}$$

3. Now move to the next column, and perform that operation. This is the middle column which is $0 - 0 = 0$.

$$\begin{array}{cccc} 1 & {}^{0}\cancel{1} & {}^{1}0 \\ - & 0 & 0 & 1 \\ \hline & & 0 & 1 \end{array}$$

4. Move to the next column, and perform that operation. This is the left column which is $1 - 0 = 1$

$$\begin{array}{cccc} 1 & {}^{0}\cancel{1} & {}^{1}0 \\ - & 0 & 0 & 1 \\ \hline & 1 & 0 & 1 \end{array}$$

This gives us the final answer of $110 - 001 = 101$

Binary Division

Binary division follows much the same procedure as 'bus stop' decimal division.

Binary Division Example

Divide $110 \div 10$

1. Draw out the division columns as you would for a standard decimal 'bus stop' division.

$$1 \quad 0 \,\overline{|\,1 \quad 1 \quad 0}$$

2. Start by looking for factors and find 11 is greater than 10. We then write the number of times the value goes into 11 at the top, and the value itself underneath.

$$
\begin{array}{r}
1 \\
1 \quad 0 \,\overline{|\,1 \quad 1 \quad 0} \\
1 \quad 0
\end{array}
$$

3. We then subtract to see if there is a remainder. $(11 - 10 = 01)$
The remainder is written up on the top line

$$
\begin{array}{r}
1 \quad 1 \\
1 \quad 0 \,\overline{|\,1 \quad 1 \quad 0} \\
-\ 1 \quad 0 \\
\hline
0 \quad 1
\end{array}
$$

4. We then bring down the final digit in the division (0), to where we are working.

$$
\begin{array}{r}
1 \quad 1 \\
1 \quad 0 \,\overline{|\,1 \quad 1 \quad 0} \\
1 \quad 0 \\
\hline
0 \quad 1 \quad 0
\end{array}
$$

5. Now, we look to see if our divisor can fit in again. It does fit again, so we subtract it. $(010 - 10 = 0)$ It leaves no remainder.

$$
\begin{array}{r}
1 \quad 1 \\
1 \quad 0 \,\overline{|\,1 \quad 1 \quad 0} \\
1 \quad 0 \\
\hline
0 \quad 1 \quad 0 \\
-\ 1 \quad 0 \\
\hline
0
\end{array}
$$

This gives us the answer of $110 \div 10 = 11$.

Page 3

Worksheet 1

📅 20-09-22   👁 Worksheet

Basic Exercises

1. Convert the following numbers to binary

   (a) 12 = 1100
   (b) 103 = 1100111
   (c) 97 = 1100001
   (d) 55 = 0110111
   (e) 395 = 110001011

2. Convert the following binary numbers to decimal

   (a) 1101 = 13
   (b) 101001 = 41
   (c) 110111 = 55
   (d) 1000011 = 135
   (e) 11111110 = 254

3. Convert the following decimal numbers to hexadecimal numbers

   (a) 1026

0100	0000	0010
4	0	2

   = 402

   (b) 5678

0001	0110	0010	1110
1	6	2	E

   = 162E

   (c) 9567

0010	0101	0101	1111
2	5	5	F

   = 255E

(d) 72627

0001	0001	1011	1011	0011
1	1	B	B	3

= 11BB3

(e) 115497

0001	1100	0011	0010	1001
1	C	3	2	9

= 1C329

4. Convert the following hexadecimal numbers to binary numbers

    (a) 2D = 0010 1101

    (b) F3A = 1111 0011 1010

    (c) 1BD = 0001 1011 1101

    (d) ABC = 1010 1011 1100

    (e) D3F2 = 1101 0011 1111 0010

Core Exercises

a) 11 + 11

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

= 110

b) 100 + 10

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

= 110

c) 111 + 11

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

= 1010

d) 110 + 100

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

= 1010

e)

```
      1  1  1  1
 +    1  1  0  0
  ─────────────
   1  1  0  1  1
      1  1
```

= 11011

f) 11 - 01

```
      1        1
 -    0        1
  ─────────────
      1        0
```

= 10

g) 11 - 10

```
   1        1
 - 1        0
  ──────────
   0        1
```

= 10

h) 111-100

```
   1     1     1
 - 1     0     0
  ──────────────
   0     1     1
```

= 011

i) 101 - 011

```
  ⁰1̷    ¹0    1
 -  0     1     1
  ──────────────
    0     1     0
```

= 010

j) 11 × 11

```
            1  1
 ×          1  1
  ─────────────
            1  1
 +       1  1
  ─────────────
      1  0  0  1
```

= 1001

k) 101 × 111

```
          1  0  1
 ×        1  1  1
  ───────────────
          1  0  1
       1  0  1
 +  1  0  1
  ─────────────────
   1  0  0  0  1  1
      1  1  1
```

= 100011

l) 1101 × 1010

```
            1  1  0  1
 ×          1  0  1  0
  ───────────────────
            0  0  0  0
         1  1  0  1
      0  0  0  0
 +  1  1  0  1
  ──────────────────────
   1  0  0  1  0  0  1  0
      1  1  1
```

= 10000010

m) 110 ÷ 11

```
              1   0
    1   1 │ 1   1   0
              1   1
          ─────────
              0   0   0
```

= 10

n) 110 ÷ 10

```
                  1   1
    1   0 │ 1   1   0
              1   0
          ─────────
              0   1   0
                  1   0
              ─────────
                  0   0
```

= 11

o) 1100 ÷ 100

```
                      1   1
    1   0   0 │ 1   1   0   0
                  1   0   0
              ─────────────
                  0   1   0   0
                      1   0   0
                  ─────────────
                      0   0   0
```

= 11

More Core Exercises

a) 11 + 01

```
            1   1
    +       0   1
      ───────────
      1     0   0
      ───────────
      1     1
```

= 100

b) 10 + 10

```
            1   0
    +       1   0
      ───────────
      1     0   0
      ───────────
      1
```

= 100

c) 101 + 11

```
          1   0   1
    +         1   1
      ───────────────
      1   0   0   0
      ───────────────
      1   1   1
```

= 1000

d) 111 + 110

```
          1   1   1
    +     1   1   0
      ───────────────
      1   1   0   1
      ───────────────
      1   1
```

e) 1001 + 101

```
      1   0   0   1
  +       1   0   1
  ─────────────────
      1   1   1   0
              1
```

= 1110

f) 1101 + 1011

```
      1   1   0   1
  +   1   0   1   1
  ─────────────────
  1   1   0   0   0
  1   1   1   1
```

= 11000

g) 11-1

```
  1           1
  -           1
  ───────────────
  1           0
```

= 10

h) 101 - 100

```
  1       0       1
  -   1   0       0
  ───────────────────
  0       0       1
```

= 001

i) 110 - 101

```
  1      ⁰1̸     ¹⁰0̸
  -  1    0      1
  ───────────────────
     0    0      1
```

= 001

j) 1110 - 11

```
  1      ⁰1̸     ¹1̸     ¹⁰0̸
  -              1       1
  ───────────────────────────
  1       0      1       1
```

= 1011

k) 1100 - 1001

```
  1      ⁰1̸     ¹0̸     ¹⁰0̸
  -  1    0      0       1
  ───────────────────────────
     0    0      1       1
```

= 0011

l) 11010 - 10111

```
  1      ⁰1̸     ¹0̸    ¹⁰1̸    ¹⁰0̸
  -  1    0      1      1      1
  ──────────────────────────────────
     0    0      0      1      1
```

= 00011

m) 11 × 11

```
            1   1
    ×       1   1
            1   1
    +     1   1
        1 0 0 1
            1
```

= 1001

n) 100 × 10

```
        1   0   0
    ×       1   0
        0   0   0
    + 1   0   0
      1 0 0 0
```

= 1000

o) 111 × 101

```
          1   1   1
    ×     1   0   1
          1   1   1
        0   0   0
    +   1   1   1
      1 0 0 0 1 1
          1   1
```

= 100011

p) 1000 × 110

```
        1   1   0   1
    ×       1   1   0
        0   0   0   0
      1   0   0   1
    + 1   0   0   1
      1 1 0 1 1 0
```

= 110110

q) 1101 × 1101

```
            1   1   0   1
    ×       1   1   0   1
            1   1   0   1
          0   0   0   0
        1   1   0   1
    +   1   1   0   1
      1 0 1 0 1 0 0 1
        1 1 1 1 1
```

= 10101001

r) 1110 × 1101

```
            1   1   1   0
    ×       1   1   0   1
            1   1   1   0
          0   0   0   0
        1   1   1   0
    +   1   1   1   0
      1 0 1 1 0 1 1 0
        1 1 1 1
```

=10110110

s) $100 \div 10$

```
            1   0
    1   0 | 1   0   0
            1   0
            ___
            0   0   0
```

= 10

t) $1001 \div 11$

```
                1   1
    1   1 | ⁰1̶   ¹0̶   ¹⁰0̶   1
                1   1
                ___
                0   1   1
                    1   1
                    ___
                    0   0
```

= 11

u) $1100 \div 100$

```
                    1   1
    1   0   0 | 1   1   0   0
                1   0   0
                ___
                0   1   0   0
                    1   0   0
                    ___
                    0   0   0
```

=11

Page 4

Negative Numbers

📅 03-10-22   🕐 16:00   🎓 Farzad   📍 RB LT1

In computers, subtraction is not possible. We must convert the calculation to be an addition. For example 5−3 is not possible, so it becomes 5+(−3). This means we need to be able to represent negative numbers in binary; there are three methods we can use to do this.

Sign and Magnitude

In this method, the Most Significant Bit (MSB) is replaced to show the sign rather than a number. A 0 represents a positive number and a 1 represents a negative number. The other bits behave the same.
Converting to and from sign and magnitude binary and decimal is the same as unsigned binary.

	+/−	64	32	16	8	4	2	1
27	0	0	0	1	1	0	1	1
-27	1	0	0	1	1	0	1	1
+13	0	0	0	0	1	1	0	1
-34	1	0	1	0	0	0	1	0

1's Complement

To convert to 1's complement, first you need to convert to unsigned binary. You then invert the bits so that 0s become 1s and 1s become 0s.
When doing a 1s complement addition, its important that any overflow bits are carried around to the least significant bit and added on there.

> **1's Complement subtraction example**
>
> Perform the calculation 10-6 = 1010 - 0110.
> First, convert the second value to 1s complement = 1010 + 1001. Then draw out the addition grid and perform the addition
>
> | | 1 | 0 | 1 | 0 |
> |---|---|---|---|---|
> | + | 1 | 0 | 0 | 1 |
> | | 0 | 0 | 1 | 1 |
> | 1 | | | | |

As we have an overflowing carry, we have to add this to the least significant bit of the answer.

	1	0	1	0
+	1	0	0	1
	0	0	1	1
+				1
	0	1	0	0
		1	1	

And here we have our final answer, 4.

2's Complement

To convert to decimal to 2's complement binary, first convert to unsigned binary. Then work from right to left, inverting the bits so that 0 becomes 1 and 1 becomes 0. However, don't flip any bits to the right of or including the first 1. All bits to the left of should be flipped.

2's Complement subtraction example

Perform the calculation 6-1 = 110-001.
First, convert the second value to 2's complement = 111. Then draw out the addition grids and perform the addition.

	1	1	0
+	1	1	1
	1	0	1
1	1		

We have an overflow carry, we discard this. This gives us our final answer of 5.

Page 5

Worksheet 2/1

📅 04-10-22   👁 Worksheet

Core Exercises

Find the negative numbers using the sign and method for the following.

(a) -7 = 10000111

(b) -12 = 10001100

(c) -15 = 1001111

(d) -46 = 10101110

(e) -57 = 10111001

(f) -112 = 11110000

(g) -149 = 100010010101

(h) -179 = 100010110011

(i) -216 = 100011011000

(j) -406 = 100110010110

(k) -1001 = 11111101001

(l) -1645 = 111001101101

Find 1's Complement for the following.

(a) 01 = 10

(b) 10 = 01

(c) 111 = 000

(d) 011 = 100

(e) 100 = 011

(f) 101 = 010

(g) 0011 = 1100

(h) 1001 = 0110

(i) 10111 = 01000

Find 2's Complement for the following.

(a) 01

```
      1   0
  +       1
 ─────────────
      1   1
```

= 11

(b) 10

```
      0   1
  +       1
 ─────────────
      1   0
      1
```

= 10

(c) 111

```
    0   0   0
  +         1
 ───────────────
    0   0   1
```

= 001

(d) 011

```
    1   0   0
  +         1
 ───────────────
    1   0   1
```

= 101

(e) 100

```
    0   1   1
  +         1
 ───────────────
    1   0   0
    1   1   0
```

= 100

(f) 101

```
    0   1   0
  +         1
 ───────────────
    0   1   1
```

= 011

(g) 0011

```
    1   1   0   0
  +             1
 ───────────────────
    1   1   0   1
```

= 1101

(h) 1001

```
    1   1   1   0
  +             1
 ───────────────────
    0   1   1   1
```

= 0111

(i) 10111

```
    0   1   0   0   0
  +                 1
 ───────────────────────
    0   1   0   0   1
```

= 01001

Carry out the following subtractions using (i) 1's complement and then (ii) 2's complement.
1's complement shown on the left and 2's complement shown on the right.

  (a) 11-01

$$
\begin{array}{c@{}c@{}c@{}c}
^1 & 1 & 1 \\
+ & 1 & 0 \\
\hline
 & 0 & 1 \\
+ & & 1 \\
\hline
 & 1 & 0 \\
\hline
 & 1 \\
\end{array}
\qquad\qquad
\begin{array}{c@{}c@{}c}
^11 & 1 \\
+ & 1 & 1 \\
\hline
\cancel{1} & 1 & 0 \\
\end{array}
$$

= 10 (right: = 10)

= 10

  (b) 11-10

$$
\begin{array}{c@{}c@{}c@{}c}
^1 & {}^11 & 1 \\
+ & 0 & 1 \\
\hline
 & 0 & 0 \\
+ & & 1 \\
\hline
 & 0 & 1 \\
\end{array}
\qquad\qquad
\begin{array}{c@{}c@{}c}
^1 & 1 & 1 \\
+ & 1 & 0 \\
\hline
\cancel{1} & 0 & 1 \\
\end{array}
$$

= 01 (right: = 01)

= 01

  (c) 101-011

$$
\begin{array}{c@{}c@{}c@{}c}
 & 1 & 0 & 1 \\
+ & 1 & 0 & 0 \\
\hline
 & 0 & 0 & 1 \\
+ & & & 1 \\
\hline
 & 0 & 1 & 0 \\
\end{array}
\qquad\qquad
\begin{array}{c@{}c@{}c@{}c}
^1 & 1 & {}^10 & 1 \\
+ & 1 & 0 & 1 \\
\hline
\cancel{1} & 0 & 1 & 0 \\
\end{array}
$$

= 010 (right: = 010)

= 010

  (d) 101-100

$$^{1}\ ^{1}1\ ^{1}0\ 1$$
$$+\ \ 0\ \ 1\ \ 1$$
―――――
$$0\ \ 0\ \ 0$$
$$+\ \ \ \ \ \ \ \ 1$$
―――――
$$0\ \ 0\ \ 1$$

$$^{1}\ \ 1\ \ 0\ \ 1$$
$$+\ \ 1\ \ 0\ \ 0$$
―――――
$$\not{1}\ \ 0\ \ 0\ \ 1$$

= 001

= 001

(e) 110-101

$$^{1}\ ^{1}1\ \ 1\ \ 0$$
$$+\ \ 0\ \ 1\ \ 0$$
―――――
$$0\ \ 0\ \ 0$$
$$+\ \ \ \ \ \ \ \ 1$$
―――――
$$0\ \ 0\ \ 1$$

$$^{1}\ ^{1}1\ \ 1\ \ 0$$
$$+\ \ 0\ \ 1\ \ 1$$
―――――
$$\not{1}\ \ 0\ \ 0\ \ 1$$

= 001

= 001

(f) 1110 - 0011

$$^{1}\ ^{1}1\ \ 1\ \ 1\ \ 0$$
$$+\ \ 1\ \ 1\ \ 0\ \ 0$$
―――――
$$1\ \ 0\ \ 1\ \ 0$$
$$+\ \ \ \ \ \ \ \ \ \ 1$$
―――――
$$1\ \ 0\ \ 1\ \ 1$$

$$^{1}\ ^{1}1\ \ 1\ \ 1\ \ 0$$
$$+\ \ 1\ \ 1\ \ 0\ \ 1$$
―――――
$$\not{1}\ \ 1\ \ 0\ \ 1\ \ 1$$

= 1011

= 1011

(g) 1100 - 1001

^{1} $^{1}1$ 1 0 0
+ 0 1 1 0
_____
0 0 1 0
+ 1
_____
0 0 1 1

^{1} $^{1}1$ 1 0 0
+ 0 1 1 1
_____
$\not{1}$ 0 0 1 1

= 0011

= 0011

(h) 11110 - 10011

^{1} $^{1}1$ $^{1}1$ 1 1 0
+ 0 1 1 0 0
_____
0 1 0 1 0
+ 1
_____
0 1 0 1 1

^{1} $^{1}1$ $^{1}1$ 1 1 0
+ 0 1 1 0 1
_____
$\not{1}$ 0 1 0 1 1

= 01011

= 01011

(i) 10111 - 10110

^{1} $^{1}1$ $^{1}0$ $^{1}1$ $^{1}1$ 1
+ 0 1 0 0 1
_____
0 0 0 0 0
+ 1
_____
0 0 0 0 1

^{1} $^{1}1$ $^{1}0$ 1 1 1
+ 0 1 0 1 0
_____
$\not{1}$ 0 0 0 0 1

= 00001

= 00001

Page 6

Worksheet 2/2

📅 05-10-22   👁 Worksheet

1. Convert the decimal numbers to signed and unsigned 8-bit binary numbers.

Decimal	Unsigned Binary	Signed		
		Sign-Magnitude	1's Complement	2's Complement
11	00001011	00001011	00001011	00001011
-11	——	10001011	11110100	11110101
29	00011101	00011101	00011101	00011101
-29	——	10011101	11100010	11100011
114	01110010	01110010	01110010	01110010
-114	——	11110010	10001101	10001110
111	01101111	01101111	01101111	01101111
-111	——	11101111	10010000	10010001

2. Convert the binary numbers using the required methods.

Binary Numbers	Unsigned	Signed		
		Sign-Magnitude	1's Complement	2's Complement
00001101	13	13	13	13
10001001	137	-9	-118	-119
11000100	196	-68	-59	-60
01000100	68	68	68	68
11111111	255	-127	0	-1
01111111	127	127	127	127
10100101	165	-37	-90	-91
11001001	201	-73	-54	-55

Page 7

Digital Gates

📅 10-10-22   🕐 16:00   🎓 Farzad   📍 RB LT1

Digital gates are used as the building block of digital systems. They manipulate inputs to provide outputs.

Throughout this lecture, its important to remember that a signal of 1 represents on (or high voltage) and a signal of 0 represents off (or low voltage).

Digital gates can be represented by a circuit symbol. Their inputs and outputs can be mapped onto a Truth Table and they have symbols which can be used in expressions to describe the circuit.

NOT Gaate

This can also be called an inverter.

As the name inverter suggests, the NOT gate inverts the bits. This means a 0 inputted, will output a 1 and a 1 inputted will output a 0.

NOT gate can only have one input.
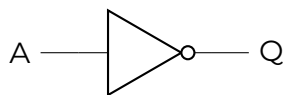
$$Q = \overline{A}$$

$$Q = A'$$

A	Q
0	1
1	0

Truth table for the NOT gate.

AND Gate

This gate compares two or more inputs. If all its inputs are 1, then it outputs 1; otherwise it outputs 0.

The rules of binary multiplication are the same of the AND gate.
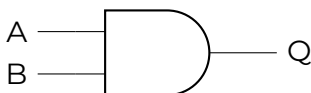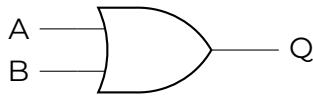
$$Q = A \bigwedge B$$

$$Q = A \cdot B$$

$$Q = AB$$

B	A	Q
0	0	0
0	1	0
1	0	0
1	1	1

Truth table for the AND gate.

OR Gate

This gate compares two or more inputs. If one or more input is 1, then it outputs 1; otherwise it outputs 0.

$$Q = A \bigvee B$$

$$Q = A + B$$

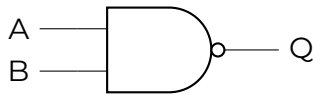| B	A	Q
0	0	0
0	1	1
1	0	1
1	1	1

Truth table for the OR gate.

NAND Gate

Not AND
This gate compares two or more inputs. It outputs 1 where at least one of the inputs are not 1 and 0 where all the inputs are 1.
Through combinations of this gate, all the other logic gates can be made, due to this, it is called a Universal Gate.

$$Q = \overline{AB} = (AB)'$$

$$Q = \overline{A \bigwedge B} = (A \bigwedge B)'$$

$$Q = \overline{A \cdot B} = (A \cdot B)'$$

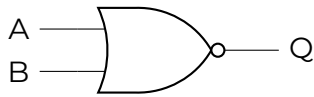| B	A	Q
0	0	1
0	1	1
1	0	1
1	1	0

Truth table for the NAND gate.

NOR Gate

Not OR
This gate compares two or more inputs. Where all the inputs are 0, it outputs 1; otherwise it outputs 0.
Through combinations of this gate, all other logic gates can be constructed, due to this it can be called a Universal Gate.

$$Q = \overline{A + B} = (A + B)'$$

$$Q = \overline{A \bigvee B} = (A \bigvee B)'$$

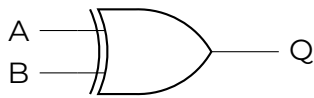| B	A	Q
0	0	1
0	1	0
1	0	0
1	1	0

Truth table for the NOR gate.

XOR Gate

eXclusive OR
This gate compares two or more inputs. For a 2-input XOR gate, where the two inputs are different, it outputs 1; otherwise it outputs 0.
This gate is used for adder circuits.

$$Q = A \oplus B$$
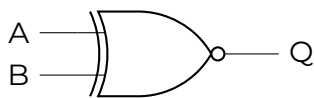
$$Q = AB' + A'B$$

$$Q = A \cdot \overline{B} + \overline{A} \cdot B$$

B	A	Q
0	0	0
0	1	1
1	0	1
1	1	0

Truth table for the XOR gate.

XNOR Gate

eXclusive Not OR
This gate compares two or more inputs and where the inputs are the same, it outputs 1 and where the inputs are different, it outputs 0.

$$Q = \overline{A \oplus B}$$

$$Q = \left(AB' + A'B\right)'$$

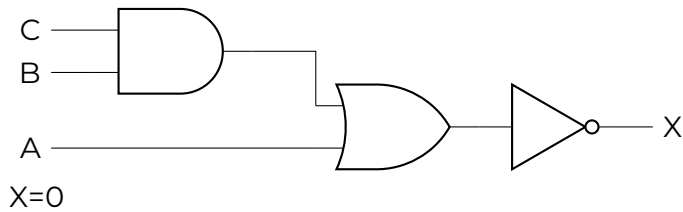| B	A	Q
0	0	1
0	1	0
1	0	0
1	1	1

Truth table for the XNOR gate.

Practice Questions

Question 1

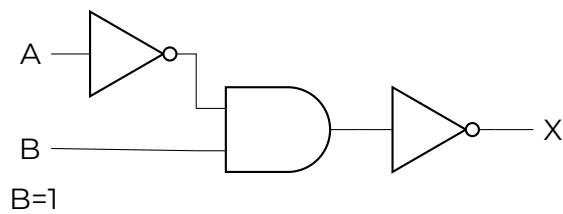Find the value of X where the inputs have the following values.

$$A = 0 \ \ B = 1 \ \ C = 1$$

X=0

Question 2

Find the value of B where the logic system is as follows and has the following values.
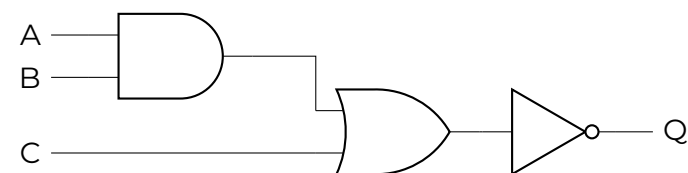
$$X = 0 \ \ A = 0$$

B=1

Question 3

Convert the following boolean logic expression to a circuit diagram.

$$\overline{A \cdot B + C}$$

Page 8

Worksheet 03
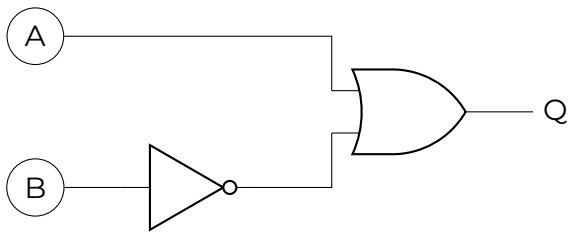
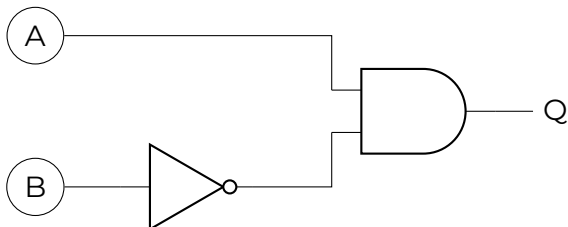📅 15-10-22   👁 Worksheet

Basic Exercises

1. Translate the following boolean expressions into a digital gate circuit and construct a truth table.

a. $A + \overline{B}$

B	A	Q
0	0	1
0	1	1
1	0	0
1	1	1

b. $A \cdot \overline{B}$

B	A	Q
0	0	0
0	1	1
1	0	0
1	1	0

c. $A \cdot B \cdot C$
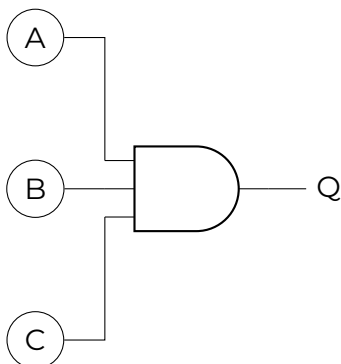
C	B	A	Q
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

d. $A + B + C$
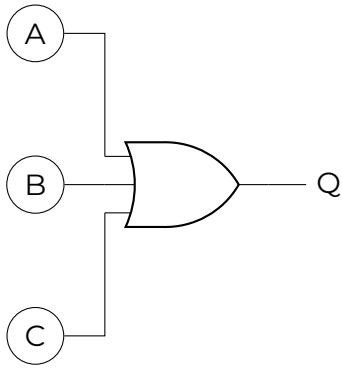
C	B	A	Q
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

e. $\overline{A} + B + C$
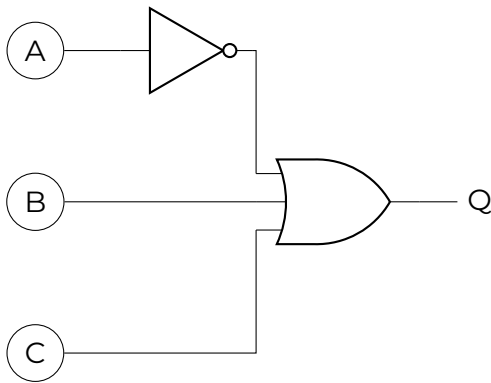
C	B	A	Q
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

f. $A \cdot B \cdot C \cdot D$
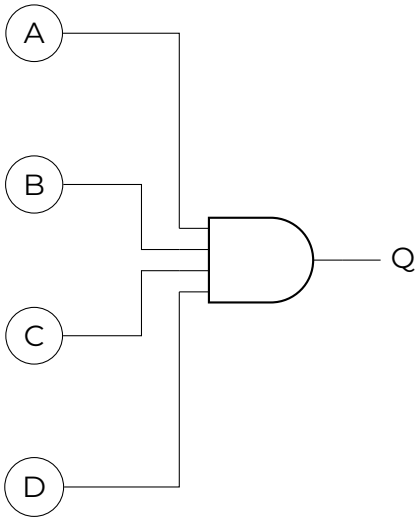
D	C	B	A	Q
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

g. $A + B + C + D$
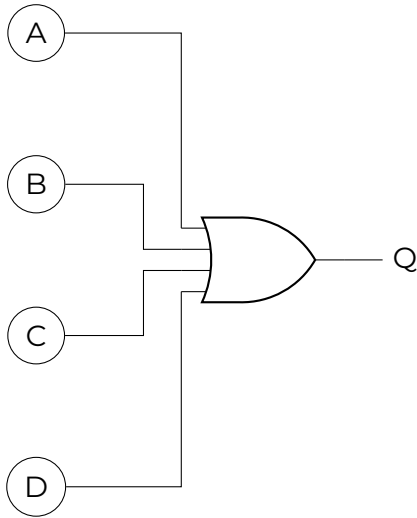
D	C	B	A	Q
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

h. $\overline{A \cdot B}$
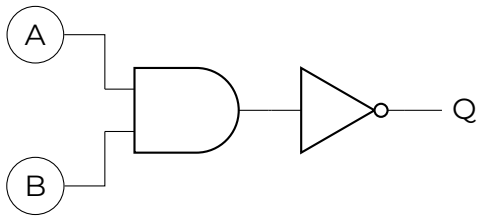
B	A	Q
0	0	1
0	1	1
1	0	1
1	1	0

i. $\overline{A + B}$
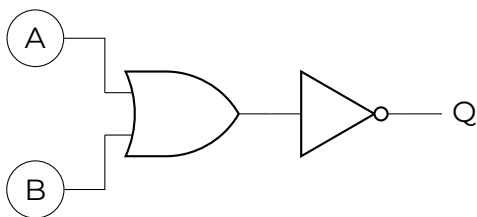
B	A	Q
0	0	1
0	1	0
1	0	0
1	1	0

j. $A \cdot B + C$
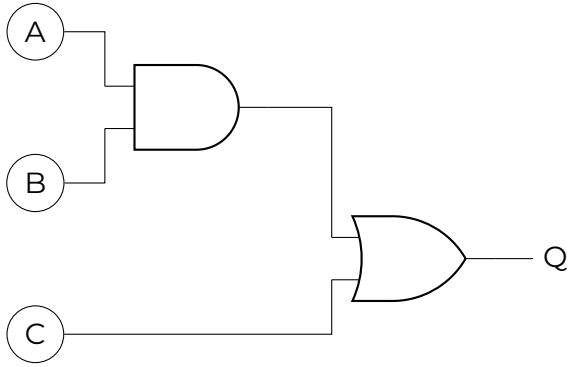
C	B	A	Q
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

k. $A \cdot B + A \cdot C$
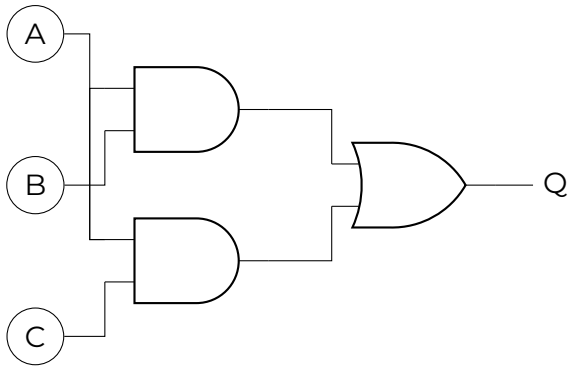
C	B	A	Q
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Page 9

Adders and Subtractors

📅 25-10-22   🕐 16:00   🎓 Farzad   📍 RB LT1

Within computers, there needs to be a way that numbers can be added together and subtracted. This is done using something called an adder subtractor. There are a number of different versions of the circuit which we need to look at first.

Half Adder

To start with, if we look at a truth table showing two inputs (B and A) and two outputs (sum and $carry$), we can show the combinations of gates which we need for a half adder.

B	A	sum	carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

From the table, we can see that the sum column represents the truth table for an XOR gate and the carry column represents the truth table for an AND gate. We can draw this as a circuit diagram.
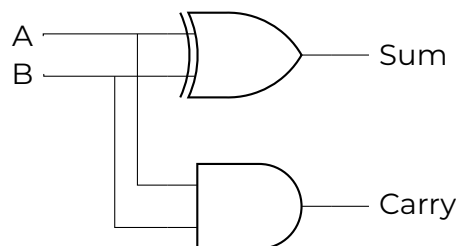
Full Adder

The circuit above is all well and good for adding one bit to another bit, but what if we are trying to do two bit addition (e.g. 11 + 01). The table below shows how we would express this in a truth table.

A	B	C_{in}	sum	C_{out}
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

This addition is a two stage. First add $A + B = S_{interim} + C_{interim}$. Then add $C_{in+S} = S_{out} + C_{interim2}$. This can be represented in the following circuit.
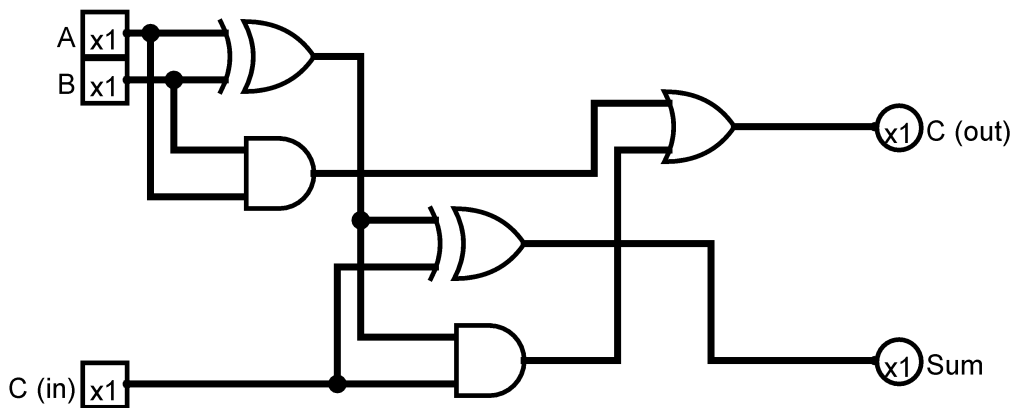
We can turn this circuit into a block for diagram simplicity.
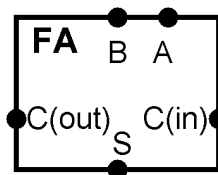
So, using 5 gates we can add 2 bits and give a carry out (this is 2 XOR, 2 AND and 1 OR). What happens if we want to add 1101 and 0110 (4 bits).
We will need a bigger adder. The general rule of thumb is one full adder for each bit we want to add.
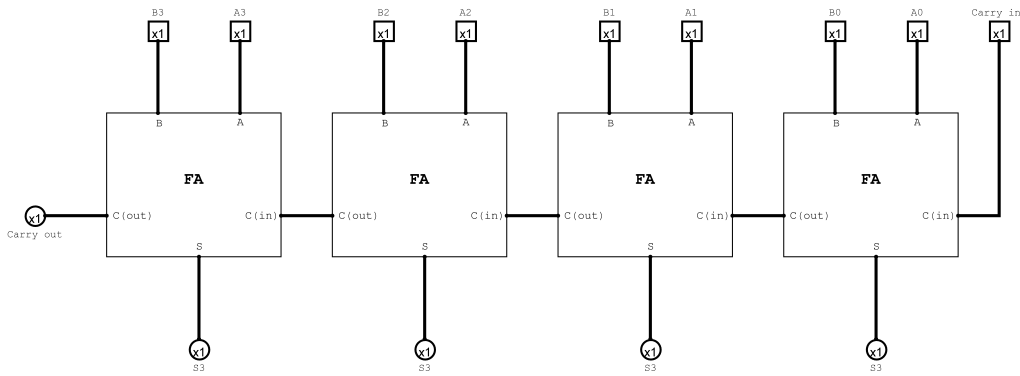
Subtracting

As we know from earlier weeks, computers do not subtract, they add a negative number. This poses a challenge as we need to convert the second number in the sum to a negative number using a two's complement number. To do this, we have to first flip the bits then add a 1 to the cary in of the least significant bit adder. This would generate the following circuit diagram.
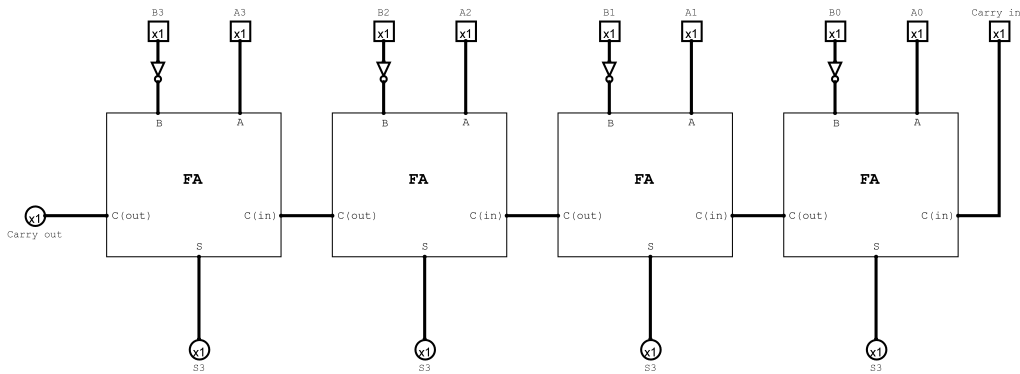
We use two's complement to avoid having positive and negative zero.
Inside a computer, it is a bad use of space to have two circuits, we instead want one 'General Purpose Circuit'. This should add and subtract.

Adder Subtractor

The subtractor bit of the circuit will always flip B. We need to find a combination of gates that when set to add, doesn't invert and when set to subtract, it inverts B. This gate is the XOR gate, this circuit diagram is shown below.
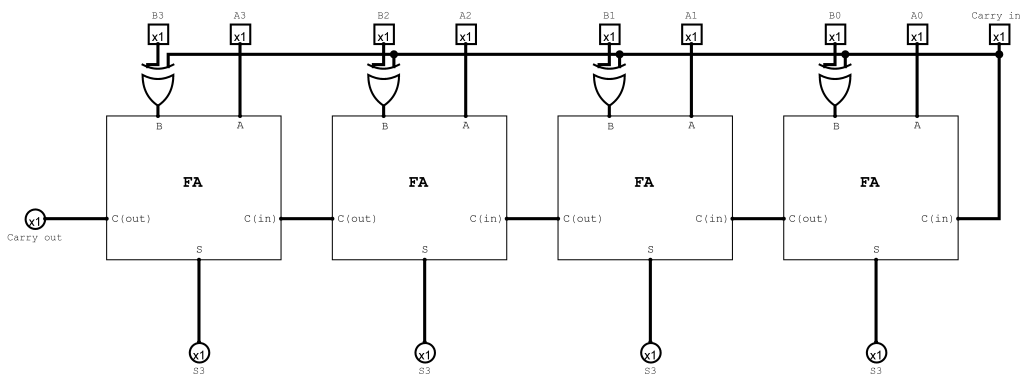
If the cary in switch is 0, B is kept the same and $C_{in} = 0$. If the carry in switch is 1, B is flipped $= \overline{B}$ and $C_{in} = 1$, which adds the 1 needed for the addition part of two's complement.

Page 10

Boolean Algebra Simplification I

📅 07-11-22   🕐 16:00   🎓 Farzad   📍 RB LT1

Now we have covered all the different gates and how they interact, we need to look at ways to make the circuits simpler. There are a number of ways this can be done, either through Boolean Algebra Simplification or through Karnaugh Maps. We'll be looking at the former in this lecture.

As a general principle of simplification, we need to get rid of the brackets to see exactly what we are dealing with then simplify (which may involve returning some things to brackets).

Laws of Boolean Algebra

Law 1: Commutative

This law states that the order of terms is interchangable without changing the overall expression permitting the gates between them do not change.

$$A + B = B + A$$

$$A \cdot B = B \cdot A$$

Law 2: Associative

This law states that where there are multiple of the same operator used brackets, and therefore gates, can be placed anywhere.

$$A + (B + C) = (A + B) + C$$
$$= A + B + C$$

$$A \cdot (B \cdot C) = (A \cdot B) \cdot C$$
$$= A \cdot B \cdot C$$

Law 3: Distributive (Associative)

This rule acts much like simplification in standard maths where a common term is 'factored' out so that it only appears once in the equation.

$$A(B + C) = A \cdot B + A \cdot C$$

Rules of Boolean Algebra

Operations with 0 and 1

Rule 1

In this rule, the 0 bit means that the output will always be A.

$$A + 0 = A$$

A	0	Q
0	0	0
1	0	1

Rule 2

In this rule, the 1 bit means that the output will always be 1.

$$A + 1 = 1$$

A	1	Q
0	1	1
1	1	1

Rule 3

In this rule, the 0 bit means that the output will always be 0.

$$A \cdot 0 = 0$$

A	0	Q
0	0	0
1	0	0

Rule 4

In this rule, the 1 bit means that the output will always be A.

$$A \cdot 1 = A$$

A	0	Q
0	0	0
1	0	1

Idempotent Rules

Rule 5

$$A + A = A$$

A	A	Q
0	0	0
1	1	1

Rule 6

$$A \cdot A = A$$

A	0	Q
0	0	0
1	1	1

Laws Of Complementarity

Rule 7

$$A + \overline{A} = 1$$

A	$\overline{A}$	Q
0	1	1
1	0	1

Rule 8

$$A \cdot \overline{A} = 0$$

A	$\overline{A}$	Q
0	1	0
1	0	0

Other Laws

Rule 9: Double Inversion

In this rule, the 0 bit means that the output will always be A.

$$\overline{\overline{A}} = A$$

$\overline{\overline{A}}$	Q
0	0
1	1

Rule 10

This rule is applicable in both directions. The bits removed can also be added back where needed. This is useful for some simplifications.

$$A + A \cdot B = A$$

$$A = A \cdot B + A$$

Derivation of Rule 10

$$A = A \qquad\qquad rule\ 10$$
$$A = A + A \cdot B$$
$$A = A + (A + A \cdot B) \cdot B \qquad\qquad rule\ 10$$
$$A = A + A \cdot B + A \cdot B \cdot B \qquad\qquad rule\ 6$$
$$A = A + A \cdot B + A \cdot B$$
$$A = A + nA \cdot B+$$
$$A = A + A \cdot B+$$

Rule 11

$$A + \overline{A} \cdot B = A + B$$

Derivation of Rule 11

$$A + \overline{A} \cdot B =$$
$$= (A + A \cdot B) + \overline{A} \cdot B \qquad\qquad rule\ 10$$
$$= A \cdot A + A \cdot B + \overline{A} \cdot B \qquad\qquad rule\ 7$$
$$= A \cdot A + A \cdot B + A \cdot \overline{A} + \overline{A} \cdot B \qquad\qquad rule\ 8$$
$$= A \cdot (A + B) + \overline{A} \cdot (A + B)$$
$$= (A + \overline{A}) \cdot (A + B) \qquad\qquad rule\ 6$$
$$= 1 \cdot (A + B)$$
$$= A + B$$

Rule 12

$$(A + B) \cdot (A + C) = A + B \cdot C$$

Derivation of Rule 12

$$(A + B) \cdot (A + C) =$$
$$= A \cdot A + A \cdot C + A \cdot B + B \cdot C \qquad\qquad rule\ 7$$
$$= A + A \cdot C + A \cdot B + B \cdot C$$
$$= A \cdot (1 + C) + A \cdot B + B \cdot C \qquad\qquad rule\ 2$$
$$= A + A \cdot B + B \cdot C$$
$$= A \cdot (1 + B) + B \cdot C \qquad\qquad rule\ 2$$
$$= A + B \cdot C$$

Page 11

Boolean Algebra Simplification II

📅 14-11-22   ⏱ 16:00   🎓 Farzad   📍 RB LT1

NEW: Drop-in's this week are there for support. They are timetabled as Drop-Ins therefore not compulsory.

NB: From here-on, the $\overline{X}$ will be changing to X' as I now know that this is the format which we will *have* to use in the exam.

DeMorgan's Theorem

Theorem 1

$$(A \cdot B)' = A' + B'$$

Theorem 2

$$(A + B)' = A' \cdot B'$$

Multi-Part DeMorgan's

Simplify $(A \cdot B + C)'$

$$
\begin{aligned}
(A \cdot B + C)' &= (A \cdot B)' \cdot C' && \text{Apply DeMorgan's to OR} \\
&= (A' + B') \cdot C' && \text{Apply DeMorgan's to bracket} \\
&= A' \cdot C' + B' \cdot C' && \text{Expand bracket}
\end{aligned}
$$

How To Simplify

1. Apply De-Morgan's Theorem where possible. This will result in as single negated terms rather than bracketed negated terms

2. Remove brackets where possible by applying distributive laws

3. Apply rules & laws to simplify

4. Factorise the expression

5. Iterate through steps 3 and 4 until the expression is simplified.

Examples

Example 1: Simplify

$$A \cdot B \cdot C' + B \cdot C' + B' \cdot A =$$
$$= B \cdot C' \cdot (A + 1) + B' \cdot A$$
$$= B \cdot C' \cdot 1 + B' \cdot A$$
$$= B \cdot C' + B' \cdot A$$

Example 2: Simplify

$$A \cdot B + A \cdot (B + C) + B \cdot (B + C) =$$
$$= A \cdot B + A \cdot B + A \cdot C + B \cdot B + B \cdot C$$
$$= A \cdot B + A \cdot C + B + B \cdot C$$
$$= A \cdot B + A \cdot C + B(1 + C)$$
$$= A \cdot B + A \cdot C + B$$
$$= B + A \cdot C$$

Example 3: Simplify

$$\left([(B \cdot D + C) \cdot (A \cdot B')] + (B' \cdot A')\right) \cdot C =$$
$$= ([(B \cdot D + C)A \cdot B'] + A' \cdot B') \cdot C$$
$$= [(A \cdot B' \cdot B \cdot D + A \cdot B' \cdot C) + A' \cdot B'] \cdot C$$
$$= A \cdot B \cdot B' \cdot C \cdot D + A \cdot B' \cdot C \cdot C + A' \cdot B' \cdot C$$
$$= A \cdot B' \cdot C + A' \cdot B' \cdot C$$
$$= B' \cdot C \cdot (A + A')$$
$$= B' \cdot C \cdot (1)$$
$$= B' \cdot C$$

Example 4: Simplify

$$A \cdot B' + A \cdot (B + C)' + B \cdot (B + C)' =$$
$$= A \cdot B' + A \cdot B' \cdot C' + B \cdot B' \cdot C'$$
$$= A \cdot B' \cdot (1 + C')$$
$$= A \cdot B' \cdot 1$$
$$= A \cdot B'$$

Page 12

Karnaugh Maps

📅 28-11-2022   🕐 16:00   🎓 Farzad   📍 RB LT1

This is the last topic we will cover before the Christmas break. There will be an exam in January, which will be a Computer Based Test. In the exam, we can use either a Karnaugh map or the rules & laws to simplify boolean algebra.

Sum Of Products Form

Sum Of Products (SOP) form is a form in which there are no brackets. A boolean algebra equation needs to be in SOP form before it can be put into a Karnaugh map.

> **Converting equation to SOP Form**
>
> Convert the following equation to SOP form.
>
> $$A(B + C) + (CD)' + (A + B)'$$
>
> We use the Distributive laws and DeMorgan's Theorem to remove the brackets. This results in
>
> $$AB + AC + C' + D' + A'B'$$

Karnaugh Maps

A Karnaugh map is a special arrangement of a truth table which can be used to simplify boolean algebra equations. It is important to recognise the order in which the terms are written in the grid.

The Karnaugh map to the right is a two input karnaugh map. This has two inputs A and B.

The Karnaugh map to the right is a three input karnaugh map. This has three inputs A, B and C.
Note the order of the inputs on the top side.

The Karnaugh map to the right is a four input karnaugh map. This has four inputs A, B, C and D.

Note the order of the inputs on the top side.

The orientation that the karnaugh map is drawn in and the order that the letters are put on the karnaugh map do not matter. What is important is that the numbers along the side only change by one digit between each cell.

It is possible to have more then 4 inputs to a Karnaugh Map however this then becomes three-dimensional. This is not covered in this module.

Simplification using Karnaugh Maps

Simplify the following expression.

$$AB + A(B + C) + B(B + C)$$

The first step is to get the equation into SOP form.

$$AB + AC + B + BC$$

Now we can take each term one by one and find where that fits into the Karnaugh Map.

We are dealing with a three input expression so we need to use a three input Karnaugh map.

Now we take AB where $A = 1$ and $B = 1$. On the Karnaugh map, we mark a 1 where this is true.

Now we take AC where $A = 1$ and $C = 1$. On the Karnaugh map, we mark a 1 where this is true.

C\BA	00	01	11	10
0			1	
1		1	1	

Now we take B where $B = 1$. On the Karnaugh map, we mark a 1 where this is true.

C\BA	00	01	11	10
0			1	1
1		1	1	1

Now we take BC where $B = 1$ and $C = 1$. On the Karnaugh map, we mark a 1 where this is true.

C\BA	00	01	11	10
0			1	1
1		1	1	1

Finally, we need to group the 1s according to the grouping rules (shown below)

C\BA	00	01	11	10
0			1	1
1		1	1	1

Finally, we pull out the groupings which gives us the answer.

$$B + AC$$

Karnaugh Map Groupings

There are a number of rules which govern the groupings of Karnaugh Map digits.

Groups must only contain 0s

	A 0	1
B 0	0	0
1	1	1

Groups may be horizontal or vertical but not diagonal

	A 0	1
B 0	0	1
1	1	1

Groups must contain 1, 2, 4, 8 (2^n) cells.

	A 0	1
B 0	1	1
1	1	1

BA C	00	01	11	10
0	1	1	1	1
1	1	1	1	1

Groups should be as large as possible

BA C	00	01	11	10
0	1	1	1	1
1			1	1

Groups can overlap. Where there is overlap, each group must have one unique 1 to that group

Groups can wrap around the table.

Groups can wrap around the entire table on multiple axis

There should be as few groups as possible, as long as this doesn't contradict any of the previous rules

These Karnaugh Map grouping rules can be summarised as:

1. No groups containing 0s are allowed

2. No diagonal grouping

3. Group sizes should only be powers of 2 (1, 2, 4, 8, 16)

4. Groups should be as large as possible

5. All the ones must be in at least one group

6. Overlapping is allowed

7. Wrap around is allowed

8. Least possible number of groups is preferable

Simplify Complex Boolean Algebra Expression using Karnaugh Maps

Simplify the following expression using Karnaugh Maps.

$$ABC'D' + ABC'D + AB'C'D' + AB'C'D + A'B'CD + A'B'CD' + A'BCD'$$

$$= D'CA' + C'A + CB'A'$$

Page 13

Exam Preperation

📅 12-12-22 🕐 17:00 🎓 Fazad 📍 PO 1.74

Information about Exam

- Exam is on 13th January 1t 9am and will last one hour

- It takes place in a number of different computer labs across university buildings

- It will cover all content taught so far

- A good revision tactic would be to go through all the worksheets and practice sheets given out so far

- There is a mock test available on Moodle

- There are about 21 questions in the exam

- A very basic calculator is permitted, however nothing scientific

- We will be given a formula sheet (containing the boolean algebra simplification rules) and paper for working out.

Page 14

Computer Structure and Function

📅 23-01-23   🕐 16:00   🎓 Farzad   📍 RB LT1

We have now finished content relating to Binary and Boolean Algebra. Until Easter (approx 5 weeks) we will be covering the CPU and how it works then until the end of the year, we will look at operating systems and how they work.

14.1   Computers as Complex Systems

A computer is a complex system. There are two key concepts to understand when referring to how a computer works, computer architecture and computer organisation.

> **Computer Architecture**
>
> This is the attributes that have a direct impact on the logical execution of a program.

> **Computer Organisation**
>
> This referrs to the operational units and their interconnections that realise the architectural operation.

The difference between these two concepts can be described with the analogy of a car: the architecture of the car will always be the same (cars always need an engine and wheels) however precise organisation (implementation) of the architecture will vary from model to model (some cars may have electric engines and some may have petrol).
This same analogy can be applied to computers: all computers require the same architecture to work however the precise organisation varies from machine to machine (historically vacuum tubes, now we use transistors).

> **Computer Structure**
>
> The way in which the components are interrelated.

> **Computer Function**
>
> The operation of each individual component as part of the structure.

14.1.1   Computer Function

re-written
2023-04-13 The computer has four main functions.

Data Processing where the computer manipulates the data for a specific function. There are a wide variety of situations this is important in however there are only a few methods/ types of data processing.

Data Storage where the computer will save data so it can be accessed at a later time/ date. Data can either be stored in the long term (through a low cost Hard Drive/ SSD) or temporarily (through higher cost RAM).

Data Movement where the computer can move data between itself and the outside world. This data will probably interface through *Input/ Output* (I/O) devices. Data movement also concerns peripherals and communication from device to device, this process is known as data communications.

Control where the computer must coordinate all the different things which are happening within it. This will encompass data processing, storage and movement. The Control unit manages the computers resources and decides the responses to instructions, including what other components do.

14.1.2   Computer Structure

Within a computer, there are a number of interrelated components.

- Peripherals

- Communication Lines

- Storage

- Processing

There are also a number of key components within a computer.

- Central Processing Unit (CPU) - controls everything in the computer

- Main Memory - stores data

- I/O - moves data between the computer and its external environment

- System Interconnection (buses) - communicate between the main memory, I/O and CPU

Central Processing Unit

The CPU has a number of components within it.

Arithmetic & Logic Unit (ALU) is an adder/ subtracter that perfoms the arithmetic and logical operations (using binary as we learnt in TB1).

Registers are very small amounts of extremely fast and very expensive memory which is internal to the CPU. It is used to speed up access to data in turn speeding up CPU performance.

Internal Bus is the internal communications line within the CPU.

Control Unit (CU) controls the operations of the CPU, which controls the operations of the computer.

14.2 History Of Computers

14.2.1 First Generation

The first generation of computers used vacuum tubes. The first computer (ENIAC, Electronic Numerical Integrator and Computer) was created during WWII out of need for automating some of the missile launching procedures at the University Of Pennsylvania. It was manually programmed using decimal (not binary) and could do 5000 additions per second. It weighed 30 tones, took up 1500 square feed and contained 18000 vacuum tubes.

14.2.2 Next Generation

The next generation of computers were designed between 1946 and 1952 by Von Neumann. This machine now incorporates the 'stored-program concept'. This computer was called an Institute of Advanced Study (IAS) Computer.
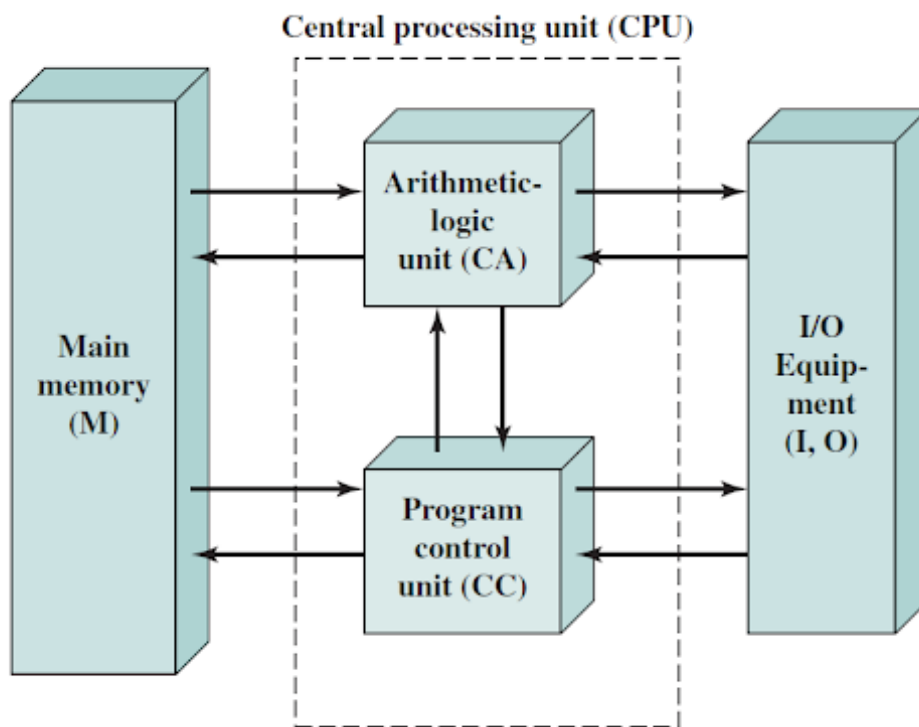
Institute of Advanced Study Computer

Memory within the IAS computer had also been re-designed since ENIAC. The memory now had 1000 locations, each 1 word (40 binary bits) long. This could either be used in Number word where the MSB acted as a sign bit or Instruction Mode where there were two instructions per word, each comprised of an 8-bit opcode (what the instruction is) and a 12 bit address (memory location of data to be used in the operation).

Page 15

WORKSHEET 10 Computer Structure and Function

📅 27-01-23   👁 Worksheet

1. **What is Computer Architecture? What is Computer Organisation? What are the main differences between them, explain them using examples?**
   Computer architecture is the attributes of the computer that have a direct impact on the logical execution of a program (for example there will need to be some form of memory and some form of addition unit). Computer organisation is the operational units and their interconnections (for example the vacuum tubes or transistors).

2. **What are the four main functions of a computer?**
   Data processing, data storage, data movement, control.

3. **Explain the functional view of a computer with the help of a flow chart.**
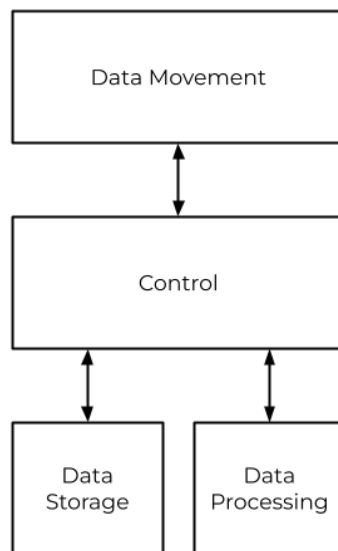
Functional view of a computer

In the above diagram, the arrows represent the movement of data throughout the blocks. The data moves from the 'movement block' where it interfaces with external peripherals. The data moves through the 'control block' where it can either go to the 'data storage' or 'data processing' blocks.

4. **Explain with a simple diagram the internal structure of the computer.** The diagram below shows the internal structure of a computer. The main memory, input and output devices, and CPU are connected together by the 'system bus'. Through this, they can share data between them.
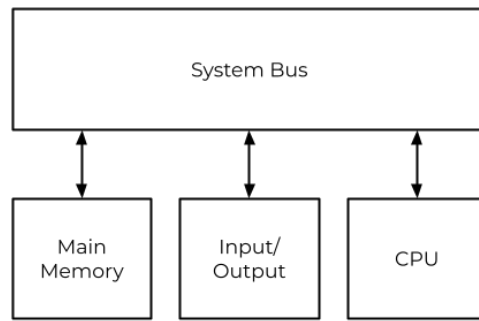
Internal structure of a computer

5. **List and define the main structural components of a processor.**
   The CPU has a number of components within it.

   The *Arithmetic & Logic Unit (ALU)* is an adder/subtractor and performs the arithmetic and logical operations which we learnt about in Teaching Block 1.

   The *Registers* are very small amounts of extremely fast memory. Register memory is very expensive. Registers are used to hold the data which is going to/ from main memory.

   The *Internal Bus* is the internal communications line within the CPU.

   The *Control Unit (CU)* is the 'heart' of the CPU. It signals to other components of the CPU to keep them in sync. The CU contains sequencing logic, CU registers and decoders, and control memory; these are beyond the scope of this course.

6. **What was the main technology used in the first generation of computers? Describe its technology.**
   The first generation of computers used vacuum tubes. Vacuum tubes work by heating up a filament which releases electrons (negatively charged). These move towards the positively charged plate (anode) which causes a current to flow.

7. **What is ENIAC and explain its working process(e.g. accumulators)?**
   ENIAC was manually programmed and worked using decimal, not binary.

8. **What is a stored-program concept?**
   The stored program concept is where the program contents is stored in main memory then during run-time it is fetched, executed and decoded.

9. **Explain with a simple diagram the von Neumann machine architecture.**
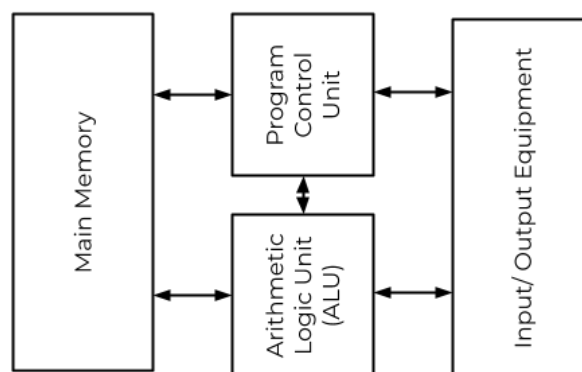
Von Neumann architecture

10. **What is the difference between ALU and control units?**
    The ALU (Arithmetic Logic Unit) is where the arithmetic and logical operations are performed whereas the Control Unit is the part of the CPU which synchronises all the other operations within the CPU.

11. **Explain IAS memory structure.**
    Memory has 1000 locations, each 1 word (40 bits) long. Could either store one number (where MSB is sign bit) or two instructions (each 20 bits long).

12. **What are the differences between Number and Instruction words in IAS computers?**
    Number words are 40 bits long (1 word) however instructions are 20 bits long (1/2 word). This means two instructions can fit into one word.

13. **Why are two different kinds of words used in von Neumann architecture?**
    We need to store instructions and numbers in the same memory.

14. **What are Opcodes and addresses in instruction words?**
    Opcodes (8-bits) are the operation which is to be performed; for example, add, subtract. Addresses (12-bits) is the address which the opcode is to be performed on.

15. **What are the advantages and disadvantages of von Neumann architecture?**
    As there is a shared instructions and data memory, you can only access one or the other at once which decreases performance. As the memory is shared, both the instructions and data have to be the same size.

Page 16

IAS Computer

📅 2023-01-30   🕐 16:00   🎓 Farzad   📍 RB LT1

16.1   IAS Computer

The IAS Computer uses the stored program concept. This is where the instructions and data required to execute the program are stored in the computer's main memory then during runtime moved to the CPU where each instruction is executed one by one. Further details on the IAS computer's structure can be found in the previous lecture. There are three key concepts of the IAS Computer

- Single read-write memory (data and instructions both stored here)

- Memory is addressable

- Execution occurs in a sequential fashion (unless explicitly modified)

16.2   Registers

Registers are very small amounts of very fast and expensive memory which is found within the CPU. There are a number of registers and each have a specific purpose.

Memory Buffer Register (MBR) contains a word to be stored in memory or sent to the I/O unit; or used to receive a word from memory or from the I/O unit.

Memory Address Register (MAR) specifies the address in memory of the word to be written from or read into the MBR.

Instruction Register (IR) contains the 8-bit opcode from the instruction currently being executed.

Program Counter (PC) contains the the address of the next instruction pair to be fetched from memory

Accumulator (AC) and Multiplier Quotient (MQ) are used to hold operands temporarily and results of ALU operations. MQ gets used if the results from the ALU is too big to fit in the AC.
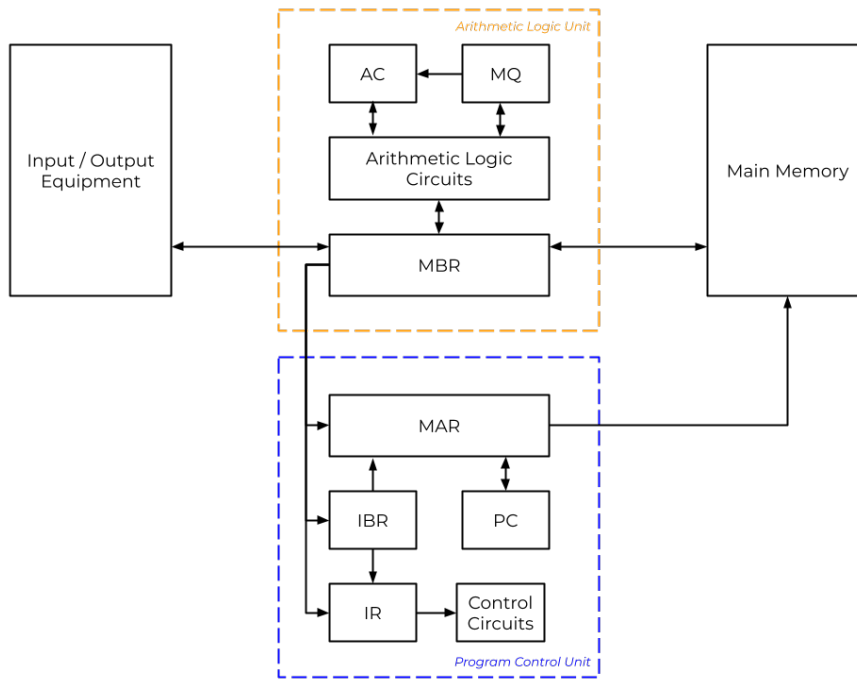
IAS Registers

16.3 Instructions

The IAS Computer had a very limited set of instructions.

Opcode	Symbolic Representation	Description

16.3.1 Data Transfer

Opcode	Symbolic Representation	Description
00001010	`LOAD MQ`	Transfer the contents of register `MQ` to the accumulator `AC`
00001001	`LOAD MQ,M(X)`	Transfer the contents of memory location `X` to `MQ`
00100001	`STOR M(X)`	Transfer contents of accumulator to memory location `X`
00000001	`LOAD M(X)`	Transfer `M(X)` to the accumulator
00000010	`LOAD-M(X)`	Transfer −`M(X)` to the accumulator
00000011	`LOAD |M(X)|`	Transfer absolute value of `M(X)` to the accumulator
00000100	`LOAD -|M(X)|`	Transfer −|M(X)| to the accumulator

16.3.2   Unconditional Branch

`00001101`	`JUMP M(X,0:19)`	Take next instruction from left half of `M(X)`
`00001110`	`JUMP M(X,20:39)`	Take next instruction from right half of `M(X)`

Conditional Branch

`00001111`	`JUMP+M(X,0:19)`	If number in the accumulator is non-negative, take next instruction from left half of `M(X)`
`00010000`	`JUMP+M(X,20:39)`	If number in the accumulator is non-negative, take the next instruction from the right half of `M(X)`

16.3.3   Arithmetic

`00000101`	`ADD M(X)`	Add `M(X)` to AC; pur result in AC
`00000111`	`ADD |M(X)|`	Add `|M(X)|` to AC; put result in AC
`00000110`	`SUB M(X)`	Subtract `M(X)` from AC; put result in AC
`00001000`	`SUB |M(X)|`	Subtract `SUB |M(X)|` from AC; put the remainder in AC
`00001011`	`MUL M(X)`	Multiply `M(X)` by MQ; put MSBs in AC, put LSBs in MQ
`00001100`	`DIV M(X)`	Divide AC by `M(X)`; put quotient in MQ and remainder in AC
`00010100`	`LSH`	Multiple AC by 2 (left shift by 1 position)
`00010101`	`RSH`	Divide AC by 2 (right shift by 1 position)

16.3.4   Address Modify

`00010010`	`STOR M(X,8:19)`	Replace left address field at `M(X)` by 12 rightmost bits of AC
`00010011`	`STOR M(X,28:39)`	Replace right address field at `M(X)` by 12 rightmost bits of AC

16.4   Programs

There are two different methods of programming computers, that we need to know about at this stage.

16.4.1   Hardwired Programs

This is like what we did in `logic.ly`. This is where the individual logic gates inputs are hand-manipulated and the connections are put together by hand to generate outputs.

16.4.2   Software Programming

This works by the code being written in a language which then gets passed through an interpreter. The interpreter will translate the high level language into a low level language which the components of the computer are able to understand and use.

Page 17

WORKSHEET 11 IAS Computer

📅 2023-02-03   👁 Worksheet

1. **What are the three key concepts of Von Neumann architecture?**
    There is a single read-write memory; the memory is addressable; and execution of instructions occurs in a sequential fashion unless explicitly modified.

2. **What is a stored-program computer?**
    A stored program computer is one where the instructions which get executed are stored within memory then loaded into the Central Processing Unit (CPU) one-by-one at run-time.

3. **What are the four main components of any general purpose computer?**
    Peripherals; communication lines; storage and processing

4. **Why are registers used in CPU?**
    Registers are used within the CPU to hold data which is being transmitted from one component of the CPU to another. They have a very small capacity and transmit data at very high rates.

5. **What is the overall function of a processor's control unit?**
    The control unit synchronises actions within the CPU and signals to components of the CPU if they need to enable/ disable a flag. This is used, for example, to signal to the ALU that the next operation it is performing is a subtraction.

6. **Explain the use of the following registers**

    (a) **Memory Buffer Register (MBR)**
        Contains the word which is being sent to/ received from main memory; or being sent to/ received from input & output devices.

    (b) **Memory Address Register (MAR)**
        Contains the address in memory of the word to be written to or read into the MBR.

    (c) **Instruction Register (IR)**
        Contains the opcode (8-bits) of the instruction currently being executed.

    (d) **Instruction Buffer Register (IBR)**
        Holds the right hand instruction while the left hand instruction from the same word is being executed.

    (e) **Program Counter (PC)**
        Holds the address in memory of the next instruction pair to be fetched.

    (f) **Accumulator (AC) & Multiplier Quotient (MQ)**
        AC holds temporary operands and results from the ALU (*Arithmetic & Logic Unit*). MQ is used if the result from the ALU is too big to fit in the AC.

7. **What are the main two phases of instruction execution**
    Fetch & Execute

8. **Explain the Fetch cycle and the registers that are used in that?**
   The program counter holds the address of the next instruction. Copy the contents of the PC to the MAR. Increment PC. Load instruction pointed to by the MAR to the MBR.

9. **Explain the Execute cycle and the registers that are used in that?**
   Fetch data from the address pointed to by the operand or perform the function stated in the opcode.

10. **What is a hardwired program?**
    A program where the individual logic gates and circuitry are manipulated to give the desired inputs.

11. **Explain the idea of programming in software**
    Programming in software works by writing the code in a high level language (for example, C, Python, etc). This code is then translated to machine code which can be processed by the hardware.

12. **What are the differences between programming in hardware and software?**
    Programming in hardware directly manipulates the electrical circuitry whereas programming in software involves additional signals before the electrical circuits can be manipulated.

13. **What do control signals do?**
    Control signals are used to signal to different components within the CPU to instruct them to perform their actions and to synchronise them.

Page 18

Fetch Execute Cycle

📅 2023-02-06   🕐 16:00   🎓 Farzad   📍 RB LT1

Computer Function

One of a computers main functions is to execute a program (this is a pre-written set of instructions). The processors execute the instructions in two stages, the first stage is to 'fetch' the instruction from memory and the second stage is to execute the instruction. Program execution is these two stages repeated until a `halt` command is executed.

Fetch & Execute Example

This example will perform the operation `2 + 3` then store the result in memory. This uses a single data register, the memory has 16-bit words of which the opcodes are 4-bits. As the Opcodes are 4 bits in length, there are 16 possible opcodes (2^4). At the start of the example, the program counter is set to 300.

> **More detailed example**
>
> The example shown below is available as a step-by-step walk though through the slides linked on Moodle.

Step 0

The data stored in memory is represented as hexadecimal as we can store one nibble with one character.

<div>

CPU Registers

PC	3	0	0	0
MAR				
MBR				
IR				
AC				

</div>

<div>

Main Memory

300	1	9	4	0
301	5	9	4	1
302	2	9	4	1
...	...	...	...	...
940	0	0	0	2
941	0	0	0	3

</div>

Step 1 - Fetch I

1. The contents of the PC is copied to the MAR

2. The contents of the memory address stored in the MAR is copied into the MBR

3. The contents of the MBR is copied into the IR

CPU Registers

PC	3	0	0	0
MAR	3	0	0	0
MBR	1	9	4	0
IR	1	9	4	0
AC				

Main Memory

300	1	9	4	0
301	5	9	4	1
302	2	9	4	1
...	...	...	...	...
940	0	0	0	3
941	0	0	0	2

Step 2 - Execute I

1. The PC is incremented

2. The opcode is analysed and the instruction is performed. In this case, the opcode is
   `1` which means `load`. The address is `940` so the contents of memory address 940 is
   copied into the AC.

CPU Registers

PC	3	0	0	1
MAR	3	0	0	0
MBR	1	9	4	0
IR	1	9	4	0
AC	0	0	0	3

Main Memory

300	1	9	4	0
301	5	9	4	1
302	2	9	4	1
...	...	...	...	...
940	0	0	0	3
941	0	0	0	2

Step 3 - Fetch II

1. The contents of the PC is copied to the MAR

2. The contents of the memory address stored in the MAR is copied into the MBR

3. The contents of the MBR is copied into the IR

CPU Registers

PC	3	0	0	1
MAR	3	0	0	1
MBR	5	9	4	1
IR	5	9	4	1
AC	0	0	0	3

Main Memory

300	1	9	4	0
301	5	9	4	1
302	2	9	4	1
...	...	...	...	...
940	0	0	0	3
941	0	0	0	2

Step 4 - Execute II

1. The PC is incremented

2. The opcode is analysed and the instruction is performed. In this case, the opcode is 5 which means `add to AC from memory`. The address is 941 so the contents of memory address 941 is added to the contents of the AC.

<table>
<tr><th colspan="5">CPU Registers</th></tr>
<tr><td>PC</td><td>3</td><td>0</td><td>0</td><td>2</td></tr>
<tr><td>MAR</td><td>3</td><td>0</td><td>0</td><td>1</td></tr>
<tr><td>MBR</td><td>5</td><td>9</td><td>4</td><td>1</td></tr>
<tr><td>IR</td><td>5</td><td>9</td><td>4</td><td>1</td></tr>
<tr><td>AC</td><td>0</td><td>0</td><td>0</td><td>5</td></tr>
</table>

<table>
<tr><th colspan="5">Main Memory</th></tr>
<tr><td>300</td><td>1</td><td>9</td><td>4</td><td>0</td></tr>
<tr><td>301</td><td>5</td><td>9</td><td>4</td><td>1</td></tr>
<tr><td>302</td><td>2</td><td>9</td><td>4</td><td>1</td></tr>
<tr><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td></tr>
<tr><td>940</td><td>0</td><td>0</td><td>0</td><td>3</td></tr>
<tr><td>941</td><td>0</td><td>0</td><td>0</td><td>2</td></tr>
</table>

Step 5 - Fetch III

1. The contents of the PC is copied to the MAR

2. The contents of the memory address stored in the MAR is copied into the MBR

3. The contents of the MBR is copied into the IR

<table>
<tr><th colspan="5">CPU Registers</th></tr>
<tr><td>PC</td><td>3</td><td>0</td><td>0</td><td>2</td></tr>
<tr><td>MAR</td><td>3</td><td>0</td><td>0</td><td>2</td></tr>
<tr><td>MBR</td><td>2</td><td>9</td><td>4</td><td>1</td></tr>
<tr><td>IR</td><td>2</td><td>9</td><td>4</td><td>1</td></tr>
<tr><td>AC</td><td>0</td><td>0</td><td>0</td><td>5</td></tr>
</table>

<table>
<tr><th colspan="5">Main Memory</th></tr>
<tr><td>300</td><td>1</td><td>9</td><td>4</td><td>0</td></tr>
<tr><td>301</td><td>5</td><td>9</td><td>4</td><td>1</td></tr>
<tr><td>302</td><td>2</td><td>9</td><td>4</td><td>1</td></tr>
<tr><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td></tr>
<tr><td>940</td><td>0</td><td>0</td><td>0</td><td>3</td></tr>
<tr><td>941</td><td>0</td><td>0</td><td>0</td><td>2</td></tr>
</table>

Step 6 - Execute III

1. The PC is incremented

2. The opcode is analysed and the instruction is performed. In this case, the opcode is 2 which means `store to AC to memory`. The address is 941 so the contents of the AC is written to memory address 941.

CPU Registers

PC	3	0	0	3
MAR	3	0	0	2
MBR	2	9	4	1
IR	2	9	4	1
AC	0	0	0	5

Main Memory

300	1	9	4	0
301	5	9	4	1
302	2	9	4	1
...	...	...	...	...
940	0	0	0	3
941	0	0	0	5

This cycle will then continue on to the next instruction and so-on.

Page 19

Interconnections

📅 2023-02-13   🕐 16:00   🎓 Farzad   📍 RB LT1

19.1   Interconnection Structure

Computers are a network of basic modules, connected together by paths (such as the system bus). The structure of these paths depends on the exchanges taking place.

19.1.1   Memory

Within memory there are n-words, each of equal length. Each word has a unique numerical address. The operation which the memory is currently performing is controlled by read and write control signals. The location for the operation is specified by an address. Data can be written to or read from memory.

19.1.2   Input/ Output Module

The Input/ Output (I/O) Module is similar to memory from an internal point of view, in that data can be read-from and written-to it. The I/O module can control more than one external device (each port has a unique address which can be used to identify an individual device). The I/O module has internal data input and output ports, this is for data coming from/ going to other places inside the computer; it also has external data in and out ports, which are used for interfacing with an external device. The I/O module also has the ability to send interrupt signals to the CPU which signals to the CPU to stop doing what it was doing as something more important has happened (for example, printer has run out of paper) which it needs to deal with.

19.1.3   Processor

The processor reads instructions and data and writes out data after processing. It uses control signals to control the overall operations. The processor also receives interrupt signals.

19.1.4   Types of Transfer

Memory to Proecssor The processor reads an instruction or unit of data from memory

**Processor to Memory** The processor writes a unit of data to memory

I/O to Processor The processor reads data from an I/O device via an I/O module

Processor to I/O The processor sends data to the I/O device

I/O to or from Memory I/O module is allowed direct access to the memory rather than going through the CPU.

19.2   Bus Interconnection

Bus interconnections are not used much in modern systems as they are a shared transmission medium which means there is only a single route which all data must travel on. Multiple devices connect to the bus and a signal transmitted by one device is available for reception by all other devices attached to the bus. If signals overlap, they become garbled therefore only one device at a time can successfully transmit. A bus consists of multiple communication pathways (called lines); each transmits a single bit at a time which means if we want to transmit 8-bits simultaneously, we need a bus with a bus-width of 8.
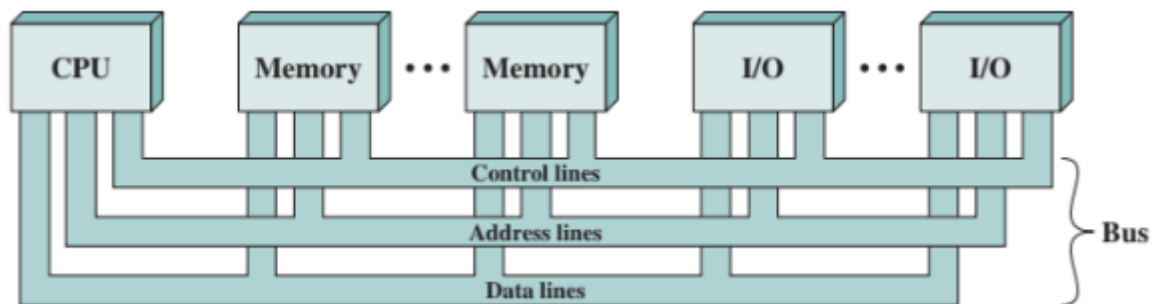
Bus interconnections

19.2.1   Data Lines

The data lines provide a path for moving data among system modules. The width of the data bus is a key factor in determining overall system performance. For example, if the data bus is 32 bits wide and the instruction to be transmitted is 64-bits then you need to use the data bus twice whereas if the data bus was 64-bits wide then you would only need to use it once.

19.2.2   Address Lines

The address bus carries the source/ destination of the data which is being transmitted on the data bus. The bus width determines the maximum possible memory capacity. Typically, the high order bits select the module on the bus and the low order bits select the memory location or I/O port.

19.2.3   Control Lines

Control lines control the access to and use of the data and address lines because the data and address lines are shared by all components and there must be a means to control their use. Control signals are transmitted on control lines, these include command signals which specify the operation to be performed and timing signals which confirm the validity of data and address information. Control lines include memory write & read, I/O write & read etc.

19.2.4   Bus Operation

If one module wishes to send data to another, it must first obtain use of the bus then it can transfer the data via the bus.
If one module wishes to request data from another module, it must first obtain the use of the bus. Then it can transfer a request to the other module over the appropriate control and address lines.

19.3 Point-To-Point Interconnection

Contemporary systems increasingly rely on point-to-point interconnections. This removes the bus system problems as well as lowering the latency, increasing the data rate and improving the scalability.

19.3.1 QuickPath Interconnect

QuickPath Interconnect (QPI) was released in 2008. It includes multiple direct connections, a layered protocol architecture and packetized data transfer.
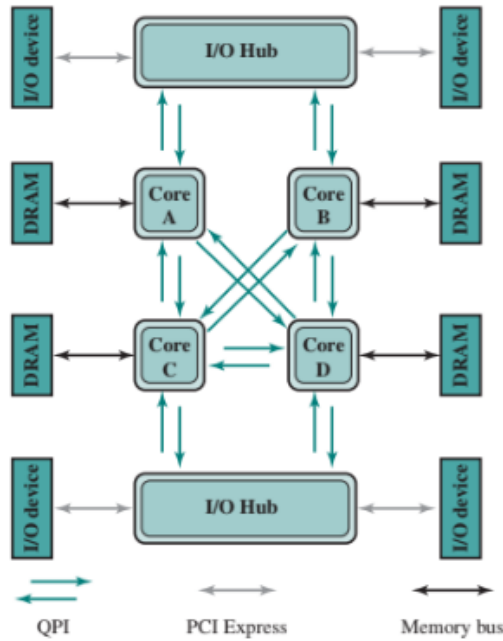
Point-To-Point Interconnection diagram showing QuickPath

Page 20

WORKSHEET 13 Interconnections

📅 2023-02-17
👁 Worksheet

1. **Why do computers need interconnection systems?**
   Computers need interconnection systems so that data and information can be transmitted between the different components which comprise the computer.

2. **What does the interconnection structure depend on?**
   The exchanges taking place.

3. **What are the types of exchanges from/to the main memory?**
   The processor can read and write data/ instructions to the main memory and the IO can read and write data to the main memory.

4. **What are the types of exchanges from/to the I/O module?**
   The I/O module can send and receive data from the main memory and send data to and receive data from the processor.

5. **What are the types of exchanges from/to the CPU?**
   The CPU can send and receive data to/ from the main memory and I/O module.

6. **What types of transfer does the interconnection structure support?**
   Bus interconnection supports transfers between the CPU, main memory and I/O module which travels through all of them. The Point-to-Point interconnection supports transfer between the same components but directly between them.

7. **What is a bus in computer systems? What are its key characteristics?**
   A bus is an interconnection line between modules. Its key characteristic is its width which determines how many bits can be transmitted in parallel.

8. **What does shared transmission medium mean?**
   Multiple data items can be transmitted down the medium at once.

9. **How do the signals overlap during transmission?**
   If two signals become overlapped then they become garbled and corrupt.

10. **What is a line in computer systems? What does it transmit?**
    A line is the name of the physical connection between components.

11. **How many lines does a bus need to transmit 2 Bytes of data?**
    16

12. **What is a system bus?**
    Collective name for the address, control and data buses.

13. **What are the different types of bus lines?**
    Address, control and data.

14. **What do the data lines do? What is a data bus?**
The data lines provide a path for moving data among system modules. The width of the data bus is a key factor in determining overall system performance. For example, if the data bus is 32 bits wide and the instruction to be transmitted is 64-bits then you need to use the data bus twice whereas if the data bus was 64-bits wide then you would only need to use it once.

15. **What do the address lines do? What is an address bus?**
The address bus carries the source/ destination of the data which is being transmitted on the data bus. The bus width determines the maximum possible memory capacity. The high order bits select the module and the low order bits select the memory location or I/O port.

16. **What do the control lines do? What is a control bus?**
Control lines control the access to and use of the data and address lines because the data and address lines are shared by all components and there must be a means to control their use. Control signals are transmitted on control lines, these include command signals which specify the operation to be performed and timing signals which confirm the validity of data and address information. Control lines include memory write & read, I/O write & read etc.

17. **How does the address bus determine the maximum possible memory capacity?**
Its width is the maximum size of the numerical value assigned to an address.

18. **How are different I/O ports addressed in computers?**
High order bits in the address lines.

19. **How does a module (memory, I/O,...) send data to another module in computers?**
If one module wishes to send data to another, it must first obtain use of the bus then it can transfer the data via the bus.

20. **How does a module (memory, I/O,...) request data from another module in computers?**
If one module wishes to request data from another module, it must first obtain the use of the bus. Then it can transfer a request to the other module over the appropriate control and address lines.

21. **What is a point to point interconnection system?**
A system where all the different components connect to each other. This means that a data transfer can go directly to where the data is intended for rather than having to go 'the long way around'.

22. **What are the main advantages of using a point to point interconnection system?**
The data being transferred can go directly to where the data is intended for rather than having to go the 'long way around'.

23. **What are the significant characteristics of QuickPath Interconnect (QPI)?**
Multiple direct connections; layered protocol architecture; packetized data transfer.

Page 21

Computer Memory Systems

📅 2023-02-20   🕐 16:00   🎓 Farzad   📍 RB LT1

Computer memory is the part of the computer where the computer stores (remembers) data and instructions (stored as binary values). Memory stores the information temporarily or permanently and provides the CPU with data and instructions. There are a number of different types of memory, each of which have a different purpose and can be found in different parts of the computer.

21.1   Memory Key Characteristics

21.1.1   Location

Memory can be *internal*, which means it is inside the core of the machine. This may include processor registers, cache memory, or main memory. Memory can also be *external*, which means it is not in the core of the machine, or even outside the machine all together. External memory can include optical disks, magnetic disks, tapes, or USB drives.

21.1.2   Capacity

The capacity of the memory concerns how much data can be stored within it. This may be measured in *words*, *bytes* or *bits*.

21.1.3   Unit of Transfer

The unit of transfer is the number of bits read out of or written into memory at a time. *Bits* are generally moved in main memory, *words* are used in registers, and *blocks* are used when moving from one memory location to the other. A *block* is multiple words together.

21.1.4   Access Method

There are a number of different access methods which can be used to access data from memory.

Sequential Access starts reading from the start of the memory unit and continues to read until it finds the location which has been requested, this is very slow and is used for tape units

Direct Access moves directly to the address which needs to be read and can read it directly, this is generally used for disk units

Random Access when used in block mode, will move the requested words as well as some from either side of the target word, this increases the efficiency and is often found in main memory and cache systems

Associative Access is a hybrid between random access and direct access, it searches through the entire memory and returns all words with a similar contents to what you requested, this is generally found in cache systems

21.1.5 Performance

Access Time is the time it takes to perform a read or write operation in random access memory and for other types of memory it is the time it takes to position the read-write mechanism at the desired location

Memory Cycle Time is the time taken, in random access memory, is the access time plus any additional time required before a second access can commence (for example, time required for transients to die out on signal lines)

Transfer Rate is the rate at which data can be transferred into or out of a memory unit.

21.1.6 Physical Type

Semiconductor memory uses different types of material to create polarised areas which can be used to represent 1s and 0s. *Magnetic* uses magnetic technology, which can be polarised to create 1s and 0s. *Optical* uses lasers to read/ write data in the form of pits and lands. *Magneto-optical* is a hybrid of magnetic and optical.

21.1.7 Physical Characteristics

Memory can either be *volatile* (when the power cuts, memory empties) or *non-volatile* (when power cuts, memory contents stays). Memory can also be *erasable* (where it can be completely emptied, for example RAM) or *non-erasable* (where it cannot be completely emptied, for example ROM).

21.2 Hierarchy of Memory

As you move down the hierarchy, the cost decreases, the capacity increases, however the access time increases which decreases the frequency of access of the memory by the processor.

Location	Memory Type
Inboard Memory	Registers
	Cache
	Main Memory
Outboard Storage	Magnetic Disk
	CD-ROM/ CD-RW
	DVD-RW/ DVD-RAM
Off-line storage	Magnetic tape

Smaller, more expensive, faster memory is generally supplemented by larger, cheaper and slower memory.

21.3 Cache Memory

The concept of cache memory is to combine fast and expensive memory with less expensive and lower speed memory. By doing this, you end up with the cache containing a copy of a portion of memory. It works by blocks of data transferring to & from main memory

and the cache memory; this transfer is slow. Words of data are then transferred between the cache memory to & from the CPU, this is much faster.

There can be multiple levels of cache memory. In the case there are three levels: the speed of transfer is fastest between the CPU and level 1 cache and is slowest between the main memory and level 3 cache. Level 2 cache is in the middle.

Page 22

WORKSHEET 14 Computer Memory Systems

📅 2023-02-24   👁 Worksheet

1. **What is computer memory?**
   The part of the computer where the computer stores data and instructions, can be stored either temporarily or permanently.

2. **Why do computers need memory?**
   To be able to store data and instructions, either temporarily or permanently.

3. **How is memory capacity usually measured?**
   In terms or words, bits or bytes (or multiples of them, eg terabytes).

4. **What is the memory unit of transfer? What are the units?**
   Bits are used when referring to main memory, words are used in registers and blocks are used when moving from one memory location to another.

5. **What does memory access mean?**
   The computer retrieving data from or sending data to main memory.

6. **What is the sequential access method in memory? Give an example.**
   The memory starts being read from the first location and all locations are examined until the desired location is found, this is used in tape units.

7. **What is the direct access method in memory? Give an example.**
   The read/write head moves directly to the address which needs to be accessed. Disk units generally use this method.

8. **What is the random access method in memory? Give an example.**
   The memory will transfer the desired word as well as some words from either side of the desired word. This is often found in main memory or cache systems.

9. **What is the associative access method in memory? Give an example.**
   Hybrid between direct and random access. It searches through the entire memory and returns all the words similar to what you have requested. Often found in cache systems.

10. **How is memory performance measured?**
    Through a number of metrics: access time, cycle time and transfer rate.

11. **What is memory access time?**
    The time it takes to access the memory.

12. **What is the memory cycle time?**
    The time it takes for one cycle of the Fetch-Decode-Execute cycle to complete

13. **What is the transfer rate in memory?**
    The maximum frequency of memory access.

14. **What are different physical types of memory?**
Semiconductor, magnetic, optical and magneto-optical.

15. **What is a volatile memory? What is non-volatile memory?**
Memory is volatile when it empties when the power is cut. Memory is non-volatile when its contents stays when the power cuts.

16. **What is ROM? Is it volatile memory? Is it erasable?**
Read Only Memory (ROM) is non-volatile and is non-erasable.

17. **What are the relationships between cost, capacity, and access time in memory design?**
As cost decreases, the capacity increases however access time increases which decreases the frequency of access of the memory.

18. **What is a good computer memory system? How it should be in terms of a combination of different types of memory?**
Smaller amounts of more expensive, fast memory (eg registers, cache) are combined with larger, cheaper slower memory (eg magnetic disk).

19. **How do you explain memory hierarchy?**

Location	Memory Type
	Registers
Inboard Memory	Cache
	Main Memory
	Magnetic Disk
Outboard Storage	CD-ROM/ CD-RW
	DVD-RW/ DVD-RAM
Off-line storage	Magnetic tape

20. **What is cache memory? Why is it important?**
Small amounts of very fast memory which is used between the CPU and registers. This allows the cache memory to contain a copy of a portion of the memory which decreases access time of the CPU getting data.

21. **How does multi-level cache organisation work?**
There are three levels of cache memory, the fastest speed of transfer is between the CPU and Level 1 cache (which is the smallest); the slowest speed of transfer is between Level 3 cache and Main Memory (L3 cache is the biggest). Level 2 cache is found in the middle of L1 and L3.

22. **How does word and block transfer work in cache memory?**
Whole blocks of data are transferred from main memory to level 3 cache. Sub blocks are transferred onto level 2. Smaller sub-parts of blocks are transferred onto level 1 where words are then transferred onto the CPU as needed.

Page 23

WORKSHEET: Guest Lecture

📅 2023-03-03   👁 Worksheet

NB: ChatGPT used for research as suggested.

1. **Operating Systems have 'entry points' - the various fine-grained, one might almost call, 'atomic' operations upon which applications and drivers may call. Look-up the different calling conventions used to access them; e.g., pascal; cdecl; fastcall etc. Why are these different ways of access used; what are the pros/cons that are being mitigated for here?**
   Calling conventions are sets of rules that determine how functions should be called and how parameters are passed between a caller and a callee in a computer program.

   - Cdecl (C calling convention): This convention is used in the C programming language and is widely used on many platforms. In this convention, the caller is responsible for cleaning up the stack after a function calll.
   - Pascal calling convention: This convention is used in Pascal and Delphi programming languages. In this convention, the callee is responsible for cleaning up the stack after a function call.
   - Fastcall calling convention: This convention is used on x86 processors and is designed to make function calls faster by passing some of the function's arguments in registers rather than on the stack.
   - Stdcall calling convention: This convention is used in the Microsoft Windows operating system and is similar to the cdecl convention, except that the callee is responsible for cleaning up the stack after a function call.

2. **What are "Systems' Programming Languages"? Why are they so named; what are their attributes, pros and cons?**
   Systems programming languages are programming languages that are designed for writing system-level software, such as operating systems, device drivers, and system utilities. These languages are typically low-level and provide direct access to system resources, such as memory, hardware, and input/output devices. They are so named because they are used to build and maintain the core systems of a computer or other computing device.

3. **Why use Assembler/Assembly language today, especially when many say that constraints on the availability of memory-usage is no longer an issue?**
   Assembler/Assembly language is still used today for a few reasons, despite the availability of modern programming languages with higher-level abstractions and more user-friendly syntax: performance, control, legacy code and debugging.

4. **Why will there never be a 128-bit operating system? I mean, NEVER. Ok, you might create an 'OS' with 128-bit wide registers, but that would only be for fun. Right?**
   It would be extremely unlikely that a 128-bit operating system will never exist. This is due to hardware limitations, diminishing returns, and complexity. It is not impossible to create a 128-bit OS, the limitations make it unlikely that it would ever be made.

5. **What's your favourite Dilbert/XKCD cartoon? You may elect to give one from the 'Far Side' if you wish.**

6. **What's your Slashdot ID number? (https://slashdot.org/) - You know 'News for Nerds, Stuff that Matters'. Your 'ID' says a lot. Really!**

7. **You're at Microsoft, interviewing folks (four scenarios) - and each is a critical hire. What do you ask? You can have two questions if you like the scenario you choose.**

   · **You're recruiting someone to work on the OS**
   · **You're recruiting someone to work on Excel**
   · **You're recruiting someone to work on UI/UX**
   · **You're recruiting a manager today (assume their background, but then expand upon it)**

   OS Developer: what experience do have developing operating systems?; how do you approach debugging complex issues in an operating systems?
   Excel Developer: what experience do you have developing applications similar to Excel?; how would you approach designing and testing an application like Excel?
   UI/UX: what design tools and methodologies do you use ot create engaging and intuitive user interfaces?; Can you walk me through your design process from initial concept to final design, how do you include stakeholders and users in the process?
   Manager: How do you approach managing a diverse team of individuals with different backgrounds an skill sets, what methods have you found to be effective in promoting collaboration and communication?; How do you balance the needs and requirements of different stakeholders?

8. **Edsger Dijkstra once said: "Computer science is no more about computers than astronomy is about telescopes." What he meant was that, in his view, Computer Science is about the 'Theory of Computation' - what a mere machine may achieve (within some stated constraints). What do you think of Dijkstra's view today? Justify.**
   Computer science encompasses more than just the physical machines that we use. It encompasses a wide range of theoretical and practical topics including algorithms, programming languages, data structures and more. The study of computer science is ultimately about understanding what can and cannot be computed.

9. **In most operating systems, one can either link one's object code to library code either statically, or dynamically (linking to LIB files, or DLLs - to use Windows' terminology). What are the dis/advantages to either model?**
   Statically linking object code to library code means that the library code is compiled and included in the final executable file. On the other hand, dynamically linking object code to library code means that the library code is loaded at runtime by the operating system, and multiple executable files can share the same library code. The advantages of static linking are: portability, stability, and performance. The advantages of dynamic linking are smaller executable size, memory usage, and flexibility.

10. **In the Hallmarks of the Portsmouth Graduate, which hallmarks are most aligned to those you think a <your computing related subject here> needs?**
    1, 2, 3, 4, 5, 7, 9, 10.

11. **If you could write a book on some aspect of Computing; perhaps software development/principles; mathematical-underpinnings, what would be a few items listed in its Contents?**
    Introduction to software development, programming fundamentals, testing and debugging, software development methodologies.

12. **Most programming languages are good at certain things, e.g., Prolog is excellent for Logic programming (goal seeking etc), whilst Scala and Functional languages may be used to write building blocks that are side-effect-free. What is your favourite language, explain why?**
Out of the Languages I've used - C# because of its power and versatility.

Unfinished.

Page 24

Operating Systems - Introduction

📅 2023-03-13   🕐 16:00   🎓 Farzad   📍 RB LT1

24.1   Development of Operating Systems

24.1.1   Late 1940s

There were no operating systems, the programmer was also the user. Programs and data were entered in binary through switching switches on the front of the machine (each switch represented one bit). Output was given through lights (each light represented one bit). The programmer did everything that an operating system does today.

24.1.2   1950s

Specialist operators were introduced. These people are not programmers. They tend to the machine, feed the programs to the machine and deliver back to the output.
Programmers use punch machines to encode their programmer as a series of holes in stiff cards. Computers can read and interpret the holes in the punch cards.
The operators acted as an human interface between the programmer and the hardware.

24.1.3   1960s

The need for a human (to load and unload punched cards, starting and stopping devices) added delay into the system.
The programmers now punched instructions onto control cards which were inserted at the appropriate places before and after the program cards and data cards
The commands were written in specially developed job control languages.

24.1.4   Late 1960s and 1970s

Computer users found that programs were growing in size and required more memory. The initial solution was to break programs up into small chunks, each fitting in the available memory.
We now have larger memories, however we need to timeshare this between different programs.
The issue of protecting one program from another became more and more important. This is a task which the OS performs.

24.1.5   1980s

Computers now have the ability to communicate with one another.

24.1.6   1990s

Previously, operating systems have been developed specifically for particular hardware platforms (for example, MS-DOS for the PC).
This move allowed generic operating systems (for example Linux) to run on any hardware.

24.2   What Is An Operating System?

The *Operating System* is a special piece of software that sits between the application and the hardware. It allows the application to interface with the hardware, so that there is only one thing interfacing with the hardware. The users interface with the applications, and at times the operating system.

Operating systems, at heart, are computer programs. They are written, tested and compiled just like any other program. Operating systems will begin running as soon as the computer has turned on, this usually happens automatically.

The operating system makes the hardware more useful and user friendly.

24.3   What Does The Operating System Do?

Operating systems provide an environment which helps other programs do productive work; helps the user to develop & run programs, by providing a convenient environment; and starts and stops applications by sharing the CPU between them.

The operating system is also responsible for managing memory, this involves keeping track of which parts of memory are in use and which are free; and providing a mechanism by which applications can ask for more memory to be allocated to them or give back memory which they no longer need.

The operating system handles inputs and outputs; differences between hardware devices (so that all applications can interface with hardware the same); and controlling input/ output and processing so that they can happen at the same time.

The operating system also provides data management, which involves managing the different physical drives and how to move data between them; protection to the CPU of overlapping processes; networking support through covering up differences between machines; and error handling & recovery which provides a way for the user to interface with errors.

24.4   Operating System Interfaces

The operating system can be interfaced with in a number of ways. This includes through a Graphical User Interface - where icons and pictures are used to control what happens in the OS or through a Command Interpreter - where text commands are entered at a prompt which the operating system interprets.

Modern operating systems will come with both a GUI and command interpreter.

The GUI, command interpreter and application programs don't directly interface with the operating system. They all interface with the system services, which provide a set of functions which request services of the operating system.

24.5   Kernel

The *kernel* is the central component (the heart) of an operating system. It interfaces between hardware components and software applications - this means it males the software interact with the hardware to get a specific task done.

The kernel decides the amount of resources (RAM, GPU, etc) to be used by every application and the order of execution of programs. The kernel has a separate memory space so that its functions are independent (this is the bit of memory, both primary and secondary, which isn't listed as usable in OS).

The kernel is the central authority which guides memory and keeps an eye on all the hardware and software data flow. The kernel responds to system calls - which are calls where processes can demand resources.

24.5.1 CPU Operation Modes

There are two modes of CPU operation, each of which have different limitations on instructions.
By default, machines run in user mode; when it wants to do something special, it has to change into kernel mode.

User Mode

The CPU can only execute a subset of its instructions - the more common ones (add, subtract, load, store, etc). When a program is executed, it doesn't have access to memory, hardware and such resources.

Kernel Mode

The CPU can execute all of its instructions, including extra privileged instructions. When a program is executed, it has direct access to memory, hardware and such resources.

24.5.2 Types of Kernel
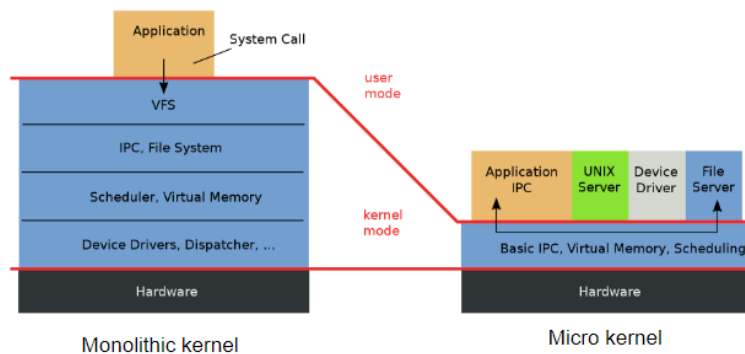
Monolithic Kernel

Both the user services and kernel services are kept in the same address space (this means the entire OS is in the kernel space). It is larger than micro kernel, less access time and fast execution, hard to extend. It has higher performance, however higher risk of a systems crash.
Linus uses the monolithic kernel.

Microkernel

The user services and kernel services are kept in separate address spaces. This means the kernel is smaller in size, there is minimum code in kernel space, greater access time with slow execution. It is easily extendible and has lower responsibility.
Windows uses a hybrid of the two types of kernel.

24.6 Types of Operating System

Different types of operating system are required for different purposes.

24.6.1 Batch Systems

These were the earliest systems developed. Data and commands to manipulate the program and data are all submitted together to the computer in the form of a *job*. There is little-to-no interaction between the users and an executing program.

These systems are used where there is no need for operator intervention once the job has started, for example in payroll processing or bank/ credit card statement generation.

24.6.2   Interactive Systems

The most common mode of using a computer is through using a keyboard, mouse and screen.
Interactive systems are a significant improvement over *batch systems* as it is now possible to intervene directly while a program is being developed or as it is running. Single user interactive systems are available, these provide multitasking and interactive computing on a single-user basis such as Windows, MS-DOS and OS/2. Also, multi user interactive systems are available, these provide interactive computing on a multi-access or multi-user basis (using different terminals) such as Unix, Linux, Windows 10, Ubuntu, Mac OS.

24.6.3   General Purpose

A given environment may want a bit of everything - for example a timesharing system may support interactive users and also include the ability to run programs in batch mode.

24.6.4   Network

To share resources such as printers and databases across a network, such as Windows NT Server.

24.6.5   Distributed

The most recent development in operating systems, this meets the requirements of a multi-user system. It consists of a group of machines acting together as one. When a user starts a program, it may run on the local machine.However if that computer is heavily loaded and the operating system knows that another machine is idle then the job may be transferred to that idle machine.

24.6.6   Specialist

Dedicated to processing large volumes of data which are maintained in an organized way such as a real-time operating system would require for example a banking system.

24.7   Design of Operating Systems

An operating system is a very large piece of software. To design, build and maintain large software systems like this requires a high-level view of how the system is structured and how its different components work together.

Page 25

WORKSHEET 15: Operating Systems (Introduction)

📅 2023-03-17   👁 Worksheet

25.1   Questions

1. **Explain where an operating system fits into a computer system**
   The operating system fits between the application and the hardware. The user interfaces with the application which then interfaces with the operating system to use resources etc.

2. **List and explain the most important functions of an operating system**

   - Provide an environment which helps other programs do productive work
   - Helps the user to develop and run programs by providing a convenient environment
   - Starts and stops applications by sharing the CPU between them
   - Managing memory - keeping track of which parts of the memory are free, which are in use and allocating memory to different applications
   - Input & output management - encapsulates inputs and outputs so that they all act the same to applications, making it easier to develop applications
   - Data management - managing the different physical drives and how data moves between them
   - Protection to the CPU - stops processes overlapping and using each others spaces
   - Networking - through covering up differences between machines
   - Error handling and recovery - provides a way for the user to interface with the errors.

3. **Explain the difference between a GUI, a command interpreter and the system service interface to an operating system.**
   GUI - little pictures which represent actions users can take in relation to the operating system. Command interpreter - user has to enter commands to interface with OS. System services - provides a set of functions which request services of the operating system.

4. **What is kernel? What is kernel mode? What is user mode?**
   Kernel is the central component of the operating system. It interfaces between hardware components and software applications (makes the software interact with the hardware to get a specific task done). Kernel mode is when the CPU can execute all of its instructions and when a program is executed it has direct access to memory, hardware and such resources. User mode is when the CPU can only execute a subset of its instructions (the most common ones) and when a program is executed it doesn't have access to memory, hardware and such resources.

5. **How does system service code get to change the CPU to run in kernel mode?**
   A special instruction is issued.

6. **Justify why the study of operating systems is relevant to a student of computing.**
   Applications we write have to interface with them and as a result it is important to have an understanding of how they work.

7. **Outline the historical development of operating systems.**

   1940s No operating systems. Programmers were the operating system.

   1950s Still no operating systems, now have specialist operators (not programmers) who load punched cards to computer and return output to programmer.

   1960s Human specialist operators are slow. Introduction of control cards (written in job control language) which get inserted at appropriate points in the program and data cards.

   Late 60s and 70s Operating systems now need to break programs into smaller chunks. Now having larger memories we need to timeshare this between different programs.

   1980s OS assists computers with communicating to one another

   1990s Move to generic operating systems (for example Linux) to run on any hardware.

8. **Distinguish the different types of operating system which have developed.**

   - Batch Systems - can only run preset jobs. There is little to no interaction between the users and an executing program.
   - Interactive systems - user interacts with it through a keyboard, mouse and screen (or equivalents). Intervening with an executing program is now possible.
   - General Purpose - does a bit of everything, including interactive users, batch mode, etc.
   - Network - to share resources such as printers and databases across a network (eg Windows NT Server)
   - Distributed - a group of machines which act together as one. Programs started by users can either run on the local machine or on another machine that is idle.
   - Specialist - dedicated to processing large volumes of data which are maintained in an organized way.

9. **Briefly describe the main modules which go to make up an operating system.**
   File Manager, IO Manager, Disk Driver, Terminal Driver, Network Manager, Network driver, Process Manager, Memory Manager.

10. **Explain the differences between Monolithic kernel and Microkernel?**
    Monolithic kernel - user services and kernel space are kept in the same address space, larger than microkernel, less time to access it, faster execution, harder to extend, higher performance, higher risk of systems crash. This is what Linux uses.
    Microkernel - user services and kernel services are kept in separate address spaces. Smaller in size, minimum code in kernel space, greater access time with slower execution, easily expandable, lower responsibility.

Page 26

Process Manager

📅 2023-03-20   🕐 16:00   🎓 Farzad   📍 RB LT1

26.1   What Is A Process?

A process is the unit of work in a computer system or a sequence of states (for example steps in the fetch-execute cycle) resulting from the action of a set of instructions on the states as they develop. A process can either be "on hold" or "running". The two commands shown below can be used in Linux systems to see a list of the current processes; in Windows, the Task Manager has a list of processes.

```
ps all
top
```

26.2   Operating System Manages Processes

The operating system maintains a Process Control Block (this is a stack data structure) for each process. It contains useful information such as the current process state, the next instruction to perform and currently allocated devices to the process. The use of Process Control Blocks enable the operating system to manage different processes effectively, by saving the current state in the process control block, switching to a different process then reloading the first process later on.

26.2.1   Process Control Block Parts

The Process Control Block has two parts.

Static Part

The static part is the resources allocated to the process and includes

- Certain amount of space in memory
- A current working directory
- Sources of input and output such as a keyboard, screen and open files
- A connection with another process over a network
- A program (sequence of instructions)

Dynamic Part

The dynamic part is a program in action and includes

- Instructions that make up a program and are actually being carried out

This is known as a 'thread of execution' or a 'thread of control' or 'thread'. A thread has access to all of the resources assigned to the task. A standard process consists of a task with a single thread.

26.3   Processor and Process

The *processor* is the agent which runs a process by executing the instructions contained in its associated program. There are far fewer *processors* than processes. This means the *processors* time has to be shared between the processes in an equal way. A process is never offered use of the *processor* while it is waiting for something (for example, keyboard input).
Processors can't have overlap of processes. It is the process manager's problem to work out the scheduling for the processor. There are different algorithms which can be used to work out the most optimum order of processes.

26.4   Operating System Overhead

It is a frequent occurrence that a process begins running on a processor and almost immediately stops again to wait for some input to become available. The operating system has to save the whole state of the machine the moment the process stops running and do the reverse (reload the state of the machine) when a process wants to start running again. Saving and restoring state is overhead and non-productive work, and here it is becoming a large part of the overall work of the computer. Because of increasing the size of operations, and operating systems become more complex, larger amounts of data have to be saved/ restored.

26.5   Multi-Threading

The operating system overhead led to the idea of having a number of paths of execution (threads) through the program at the same time. If one thread is blocked (eg, waiting on user input) another can execute. It is not necessary to save and restore the full state of the machine for this, as it is using the same memory, files and devices - it is simply jumping to another location in the program code.
Each thread must maintain some state information of its own, for example the program counter and general-purpose registers. That is so that when it regains control, it can continue from the point it was at before it lost control.
Multi-threading enables the processing of multiple threads at one time, rather than multiple processes. Multi-threading is an important utility for many computer programs and increase performance efficiency and scalability.

26.5.1   Comparison of single-threading, multi-threading, and baking a cake

Operating System Process	Baking a Cake	Baking Several Cakes
Task (resources)	Ingredients (resources)	Ingredients (resources)
Program	Recipe	Recipe
Thread	Actual sequence of operations carried out as directed by the recipe (eg mixing, baking)	While one cake is in the oven, we may be mixing the ingredients for another. WE following the recipe at two different places at the same time. Instead of idly waiting for the first cake to be baked, we are using that time productively: this is multi-threading.
Process	Make a cake	Make several cakes

26.6   Process, Tasks and Threads

The operating system needs to keep track of processes. It uses data structures called '*Process Control Blocks*' to represent the different objects it is dealing with. A process consists of a task, which is the resources allocated, and one or more threads (or control paths) through the code. It must be possible to represent the one task and a variable number of threads. It needs a static part, and a dynamic part that can grow or shrink.

26.6.1   Tasks

The structure of the static component of a task is shown below. It is represented using a stack.

owner
unique process id
address space description
program starting address
data starting address
stack starting address
list of threads
number of threads
open file information
default directory

26.6.2   Threads

Types of Thread

User Threads are above the kernel and don't have kernel support. These are threads that application programmers use in their programs. This saves on the overhead of bothering the operating system each time control changes from one to another. The operating system gives control of the CPU to a process.
Kernel Threads are supported within the kernel of the OS itself. The OS handles switching between threads. All modern OSs support kernel level threads, allowing the kernel to perform multiple simultaneous tasks and/ or to service multiple kernel system calls simultaneously.

Thread Structure

The Kernel also maintains a thread data structure for each active thread

task which I belong to
list of threads in Task
volatile environment
state
links
thread's base priority
maximum priority
current priority

One of the more important fields in this thread structure is the volatile environment. This has fields for the information that has to be saved each time the thread looses control of the CPU. Its contents is shown below

stack pointer
memory management registration
general-purpose registers
program counter
program status register

Thread State

Each thread has a life cycle of its own, during which it may be in one of two different states `RUN` or `WAIT`. The state a thread is in at any particular time is recorded in the state field of the thread structure. The `RUN` state means the thread is ready and able to do some work; there will be many threads in this state and they will be linked together on a run queue. The `WAIT` state means that for a reason, the thread is unable to use the CPU; generally threads in this state will be waiting for some event and threads in this state are linked together on a wait queue.
When a new thread is created, it will always begin in the `RUN` state in on the run queue. Eventually it will be given a time slice on the CPU. When this transition takes place is

the responsibility of the scheduler. A thread may stop using the CPU for three reasons: *voluntarily* where it is waiting for a resource and enters the `WAIT` state; *termination* where the thread or process is terminated; or *preemption* where the OS takes the processor from a thread, this happens when a thread has used up its share of time or when the CPU is needed to handle some more urgent work. A thread leaves the `WAIT` state when the event it is waiting for occurs.

26.7   Context Switching

Context Switching is the whole procedure of reallocating a processor from one thread to another. Context switching is called when

- A thread looses control of the processor and moves into the `WAIT` stat ti wait for a resource to become available

- A timer interrupts to tell it that it has used up its time slice

It is absolutely essential that the current state of the machine is saved when a thread looses control of the CPU and that these are available for when the thread is made active again. This state, called the volatile environment, is saved in the thread structure of its own.

26.8   Thread Scheduling

The main objective of the scheduler is to see that the CPU is shared among all contending threads as fairly as possible. The scheduler tries to make the response time as short as possible. In real-time systems, the scheduler will have to be able to guarantee that response times will never exceed a certain maximum. Thread priority queues and thread multilevel priority queues are used to prioritise threads getting time on the CPU.

26.8.1   Thread Priority

In an interactive system there is no way to predict in advance how many threads will be running, or how long they will want to run for. They could be scheduled on a first-come-first-served basis, but this way some unimportant threads could monopolize the system while urgent threads wait on a queue. The solution to this is to order all of the runnable or ready threads by some priority criterion, the priority is allocated to the thread when it is created. The descriptors of all runnable threads are linked into the run queue, ordered by descending priority. The most eligible thread is at the head, then the next and so on. ALl the other threads, which are not runnable, are linked together on the wait queue.

26.8.2   Thread Multilevel Priority Queues

In this example, the system will have two priority queues: high and low priority.
When a thread is created, it is initially put on the high-priority queue. If it uses its full time slice, this implies that it is compute-bound. So when it is context switched out, it is moved to the tail of the low-priority queue. On the other hand, ot fhe thread gives up the processor to wait for an I/O event, then it can bse assumed that it is interactive. Then when the event it is waiting for occurs, it is moved from the wait queue to the high-priority queue. Being placed in the high priority queue after being context switched off the CPU expects the thread to keep the CPU for a short amount of time when it gets back on the CPU.

Page 27

WORKSHEET: Operating Systems - Process Manager

📅 2023-03-24   👁 Worksheet

1. **Distinguish between the static part of a process (the task) and the dynamic part (the thread).**
   The static part of the process is the resources allocated to the process and includes the address in memory which the process can use; the current working directory; input and output devices; connection with another process over the network and the program itself. The dynamic part of the process is used when the process executes on the CPU and contains the instruction that is actively being carried out.

2. **How can more than one thread be executed at the same time on a machine with only one CPU?**
   The CPU can have multiple cores, this makes the CPU act as though there are multiple CPUs as each core can run independent of each other.

3. **What is the motivation behind introducing multiple threads of control in one process?**
   Multiple threads means that multiple things can be executed concurrently. This then means that if one thread is waiting on something, for example a user to press a key on the keyboard, then another thread can be executing therefore the CPUs time is not wasted. Multi-threading increases performance efficiency and scalability.

4. **Outline the fields in the data structures used to represent a process, a task, and a thread within an operating system.**
   A process contains the process state, process number, program counter, registers, memory limits and a list of open files. A task contains the owner, unique process id, address space description, program starting address, data starting address, stack starting address, list of threads, number of threads, open file information and the default directory. A thread contains the task that it belongs to, the list of threads in the task, the volatile environment (stack pointer, memory management registration, general-purpose registers, program counter, program status register), state, links, thread's base priority, maximum priority, current priority.

5. **Explain the difference between calling a function and creating a new thread to execute that function.**
   When you call a nre function, it pauses execution in the thread then executes the function. Once the function is complete the main execution will continue. When a new thread is created to execute the function, both the new function and main execution will happen concurrently. This is the basis of how asynchronous and synchronous functions work.

6. **Outline the life cycle of a thread.**
   A thread is created then can have one of two states - `RUN` or `WAIT`. When a thread is in the `RUN` state then it actively executing on a core of the CPU. When a thread is in the

`WAIT` state, it is not actively executing as needs some information (eg, a keyboard input from a user). When the process which is being executed on a thread is complete, the thread will be disposed of.

7. **Explain how context switching works.**
Context Switching, the process of reallocating a processor from one thread to another thread, is called when a thread looses control fo the processor and moves into the `WAIT` state to wait for a resource to become available or a timer interrupts to tell the thread it has used up its time slice. It works by saving the contents of the volatile environment into the thread's data structure.

8. **Describe the different objectives of a scheduler in interactive, batch and real-time systems.**
Interactive systems have the ability to be used through a keyboard, mouse and screen. The scheduler will work to give each process an equal time slice so that the user has the best possible experience using it. Batch systems are setup so that once a batch job is started, they cannot be interrupted. The scheduler, once the job has begun, prioritise the completion of that job and then deal with everything else. Real-time systems prioritise the real-time processing of data over everything else. This will be reflected in the scheduler, in that it will prioritise the input and manipulation of data.

9. **Explain what is meant by priority scheduling.**
Threads can be scheduled based on their priority. There are different methods which can be used to assign a priority to the thread. Generally the thread's priority is worked out as soon as a thread is created.

10. **Explain how multilevel priority feedback queues work.**
Generally the same as priority scheduling except the priority of the thread is also taken into account. Different queues, eg high and low priority queues will be treated differently in that high priority tasks will be executed before low priority tasks.

Page 28

Process Scheduling I

📅 2023-03-27   🕐 16:00   🎓 Farzad   📍 RB LT1

28.1   CPU Scheduling

CPU Scheduling is a basic requirement of a multiprogrammed operating system. This allows the CPU to be switched between processes, in turn making the computer more efficient. In a system with a single processor, only one process can run at a time; this means the other processes must wait until the CPU becomes free, and is able to be rescheduled, to be able to run. The objective of multiprogramming is to have a process running on the CPU at all times, maximising CPU utilization.

There are different methods of determining how long the process can run on the CPU for before it is removed (this is covered a bit later).

Several processes are kept in memory at one time, some of these will be `READY` to execute, this means they are waiting for time on the CPU and some will be in the `WAIT` state, meaning they require something before they can proceed (generally input or output event).

CPU scheduling is core to the design & purpose of the operating system.

28.2   Input/ Output Cycle

The execution of a process alternates between two states: CPU execution (burst) and I/O wait. Process execution begins with a CPU burst and will end with a CPU burst with a system request to terminate execution.

28.3   CPU Scheduler

Whenever a process leaves the CPU and/ or the CPU becomes idle, the operating system has to select one of the processes in the ready queue to be executed. This process is completed by the CPU scheduler. The items (processes) in the ready queue are generally the process control blocks of the processes.
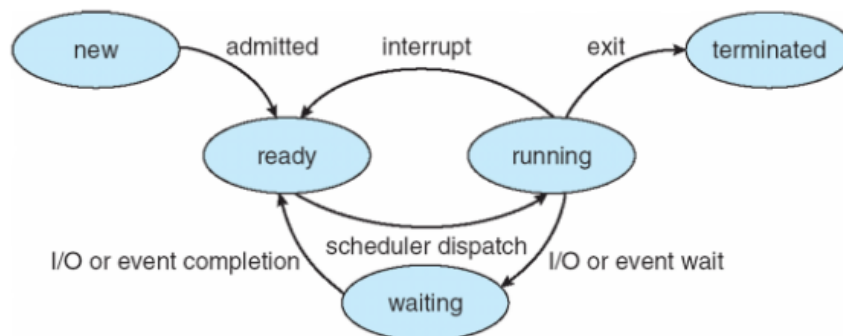
Process State Diagram

28.4   Types of Scheduling

There are two key types of scheduling. There are different algorithms which implement these different types of scheduling (these will be covered later on).

28.4.1   Non-Preemptive Scheduling

In non-preemptive scheduling, a process cannot be stopped mid execution, unless it needs something (eg input from user), in which case it will be stopped and placed on the ready queue. This means that unless the process switches to the `WAIT` state or it terminates, then there is no way to stop it. No choices are made in terms of which process to execute next, one is picked from the ready queue. This scheduling method was used by Windows version 3.x.

28.4.2   Preemptive Scheduling

In preemptive scheduling, if a process with higher priority than that executing arrives into the ready queue then the currently executing process will be stopped and the higher priority one will be given the CPU to execute on. Preemptive scheduling decisions will take place when either a process switches from the running state to the waiting state; or when a process switches from the wait state to teh ready state. There are choices in terms of scheduling. There may also be a requirement for specialist hardware - for example a timer. Windows 95 and all subsequent versions of Windows have used this type of scheduling.

28.5   Scheduler

A *scheduler* is a piece of systems software which handles process scheduling in a number of ways. Their role is to look at processes submitted to the system and decide which processes to run.

Long Term Scheduler   looks at the big picture. This works out the bits of the programmes which run in what sequence and how many processes stay in the ready queue. This happens away from the CPU.

Medium Term Scheduler   processes movement between the wait queue and ready queue.

Short Term Scheduler   decides which process from the ready queue is the next for execution and the order of execution for the other processes in the ready queue.

28.6   Dispatcher

The dispatcher is a program which gives the process control over the CPU after it has been selected by the short term scheduler. This involves three things

1. Switching Context

2. Switching to User Mode

3. Jumping to the proper location in the user program to restart that program

28.7   Scheduling Criteria

CPU Utilization   we want to keep the CPU as busy as possible

Throughput   this is the number of processes that complete their execution per time unit

Turnaround Time this is the amount of time it takes to execute a particular process

$$\text{Turnaround Time} = \text{Exit Time} - \text{Arrival Time}$$

Waiting Time this is the amount of time a process has been waiting in the ready queue.

$$\text{Waiting Time} = \text{Turnaround Time} - \text{Burst Time}$$

Response Time is the amount of time it takes from when a request was submitted until the first response is produced

28.8 Scheduling Algorithms

We will explore six scheduling algorithms over the remainder of this lecture and the next lecture.

28.8.1 First Come, First Served (FCFS)

This scheduling algorithm works by allocating the CPU to the process which arrives first. Processes are stored in a Queue which operates as First In First Out (FIFO). When a new process enters the ready queue, its process control block is linked to the tail of the queue. When the CPU is free, it is allocated to the process at the head of the queue. The newly running process is then removed from the queue.
Once the CPU has been allocated to a process, the process keeps control of the CPU until it releases the CPU either by terminating or requesting I/O (which would result in it being put in the `WAIT` queue).

> **FCFS Order Of Execution**
>
> See Week 19 slides on Moodle for an example process execution order with different length processes in different orders.

28.8.2 Shortest-Job-First Scheduling (SJF)

This scheduling algorithm works by executing the CPU to the process with the shortest execution time first. It gives the minimum average waiting time for a given set of processes; however it needs to know the length of all the processes in the ready queue.
SJF can also be preemptive and works as preemptive is described above.

Page 29

Process Scheduling II

📅 2023-04-24   🕐 16:00   🎓 Farzad   📍 RB LT1

This lecture continues from where we left off before Easter.

29.1 Scheduling Algorithms

29.1.1 Priority Scheduling

As each process arrives into the ready queue, a priority number is associated with it. This is an integer where the lower the value, the higher the priority. This algorithm can either be preemptive or non-preemptive. There is a significant issue with this algorithm: *starvation or indefinite blocking* is where a process is left waiting indefinitely, generally this will be found in low-priority processes; this can be overcome by using *aging* where the priority of a process is gradually increased at set time intervals.

Preemptive

If a higher priority process than that currently executing arrives at the CPU, then the current task will be removed from the CPU and the new, high priority task will be executed.

Non-Preemptive

If a higher priority process than that currently executing arrives at the CPU, it will have to wait for the current process to finish before it can execute.

29.1.2 Round Robin Scheduling

Round Robin (RR) scheduling transforms the Ready Queue into a circular FIFO queue. This enables each process to be given a small unit of time on the CPU (called a time quantum or time slice, usually about 10-100ms) then, wherever the process is up-to in execution, it is removed from the CPU and added to the back of the ready queue; the process from the front of the ready queue is then executed. This removes the starvation problem and if the process needs less time than the time quantum allows for then it will leave the CPU early and the next process can start. The same process will not have more than one consecutive time quantum unless it is the only process in the ready queue. The performance of the RR algorithm comes down to the length of the time quantum, in that if it is too short then lots of time will be wasted by context switching.

29.1.3 Multilevel Queue Scheduling

In this scheduling algorithm, the ready queue is split into separate queues. These may be, for example, foreground and background processes. The two different types of processes have different response-time requirements and so may have different scheduling needs. Processes are permanently assigned to one queue (generally based on some property of the process, such as memory size, process priority or process type). Each queue has its

own scheduling algorithm, for example foreground might use RR and background might use FCFS.
Scheduling is done between the between the queues. There are two methods which can be used here

Fixed Prioirity Scheduling

Here the queues are serviced in fixed priorities. For example, serve all from foreground then serve background. This leads to the high chance of process starvation.

Time Slice

In this method, each of the queues get a certain amount of CPU time within which it can schedule its processes. For example 80% of time goes to foreground and 20% of time goes to background.

29.1.4   Multilevel Feedback Queue Scheduling

This scheduling algorithm works similarly to *multilevel queue scheduling* however in this one the process can move between the various queues and aging can be implemented along the way. Multilevel feedback queue schedulers are defined by the following parameters

- The number of queues

- The scheduling algorithms for each queue

- The method used to determine when to upgrade a process

- The method used to determine when to demote a process

- The method used to determine which queue a process will enter when that process need service