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

```
            1   1
    x       1   0
    _____
```

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.

```
            1   1
    x       1   0
    _____
            0   0
```

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

```
            1   1
    x       1   0
    _____
            0   0
        1   1
```

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

```
            1   1
    x       1   0
    _____
            0   0
    +   1   1
    _____
        1   1   0
```

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.

$$
\begin{array}{cccc}
 & 1 & {}^{0}\!\!\not{1} & {}^{1}0 \\
- & 0 & 0 & 1 \\
\hline
 & & & 1 \\
\end{array}
$$

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

$$
\begin{array}{cccc}
 & 1 & {}^{0}\!\!\not{1} & {}^{1}0 \\
- & 0 & 0 & 1 \\
\hline
 & & 0 & 1 \\
\end{array}
$$

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

$$
\begin{array}{cccc}
 & 1 & {}^{0}\!\!\not{1} & {}^{1}0 \\
- & 0 & 0 & 1 \\
\hline
 & 1 & 0 & 1 \\
\end{array}
$$

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.

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

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.

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

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

$$\begin{array}{c|ccc} & & 1 & 1 \\ \hline 1 \ \ 0 & 1 & 1 & 0 \\ - & 1 & 0 & \\ \hline & 0 & 1 & \end{array}$$

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

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

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.

$$\begin{array}{c|ccc} & & 1 & 1 \\ \hline 1 \ \ 0 & 1 & 1 & 0 \\ & 1 & 0 & \\ \hline & 0 & 1 & 0 \\ - & & 1 & 0 \\ \hline & & & 0 \end{array}$$

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

Page 3

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 4

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.
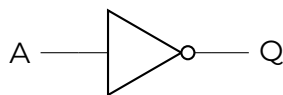
$$Q = \overline{A}$$

$$Q = A'$$

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.
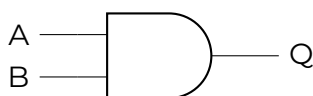
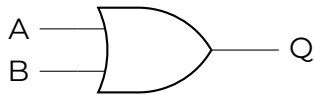$$Q = A \bigwedge B$$

$$Q = A \cdot B$$

$$Q = AB$$

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.

$$Q = A \bigvee B$$

$$Q = A + B$$

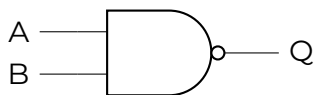| B	A	Q
0	0	0
0	1	1
1	0	1
1	1	1

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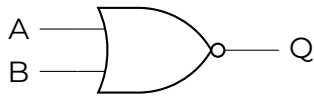
$$Q = \overline{A + B} = (A + B)'$$

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

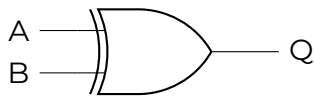| B	A	Q
0	0	1
0	1	0
1	0	0
1	1	0

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.

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

$$Q = (AB' + A'B)'$$

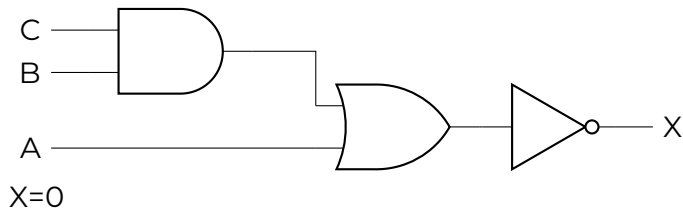| B	A	Q
0	0	1
0	1	0
1	0	0
1	1	1

Truth table for the XNOR gate.

Practice Questions

Question 1

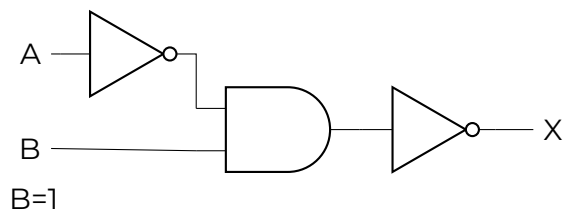Find the value of X where the inputs have the following values.

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

X=0

Question 2

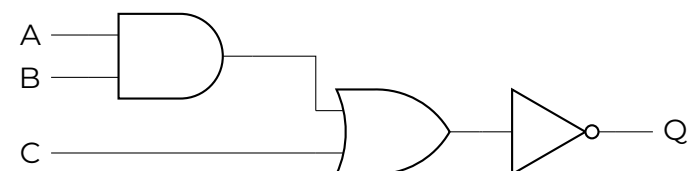Find the value of B where the logic system is as follows and has the following values.

$$X = 0 \quad A = 0$$

B=1

Question 3

Convert the following boolean logic expression to a circuit diagram.

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

Page 5

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.
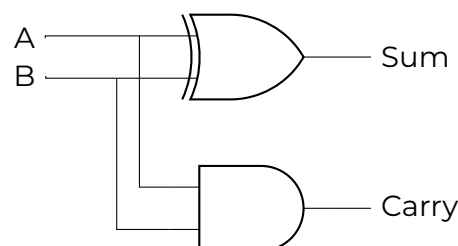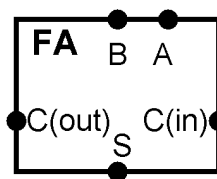

Full Adder

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.
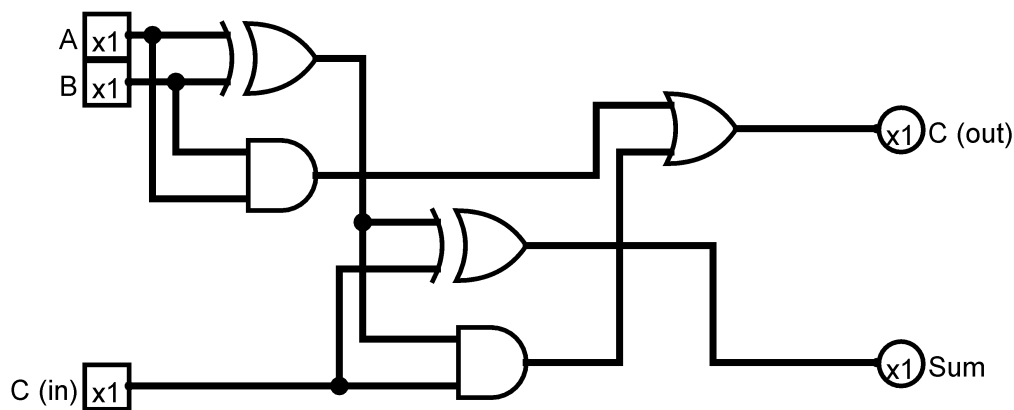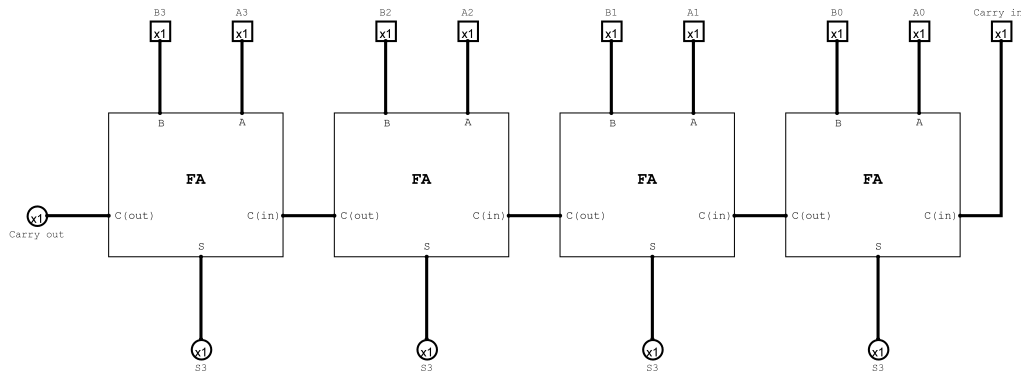

We can turn this circuit into a block for diagram simplicity.

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.
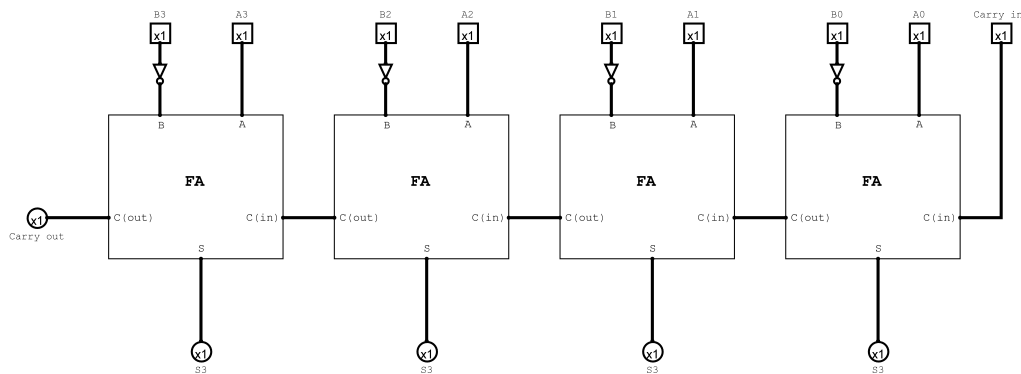
We use two's complement to avoid having positive and negative zero.
Inside a computer, it is a bad use of space to have two circuits, we instead want one 'General Purpose Circuit'. This should add and subtract.

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.
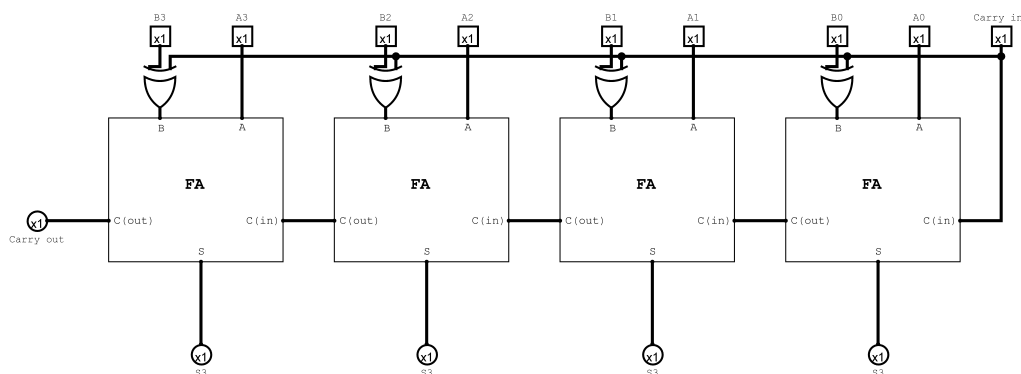
If the cary in switch is 0, B is kept the same and $C_{in} = 0$. If the carry in switch is 1, B is flipped $= \overline{B}$ and $C_{in} = 1$, which adds the 1 needed for the addition part of two's complement.

Page 6

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 7

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 8

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

Groups may be horizontal or vertical but not diagonal

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

Groups should be as large as possible

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.

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

DC\BA	00	01	11	10
00		1	1	
01	1			1
11	1			
10		1	1	

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

Page 9

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 10

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.

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.

Computer Function

There are four main functions of the computer.

- Data Processing

- Data Storage

- Data Movement

- Control

Data processing relates to the manipulation of data for a specific function.

Data storage has two different types, long term and short term. Long term is the Hard Drives or SSDs where data can be stored for a long period of time at relatively low costs. Short term is the RAM within a computer where data which is currently or soon-to-be needed by the CPU is stored, this is at higher cost than the long term storage. The storage of data requires movement of data from one place to another to store it.

Data Movement concerns data entering or leaving a system. This will most probably be coming from an input device (keyboard, mouse or microphone) and will most probably be going to an output device (monitor, speakers or printer). Data movement also concerns peripherals and communication from device to device.

Control is required with everything as computers must coordinate all their actions.

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 - short term memory

- I/O - input & output devices

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

Central Processing Unit

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.

History Of Computers

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.

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.
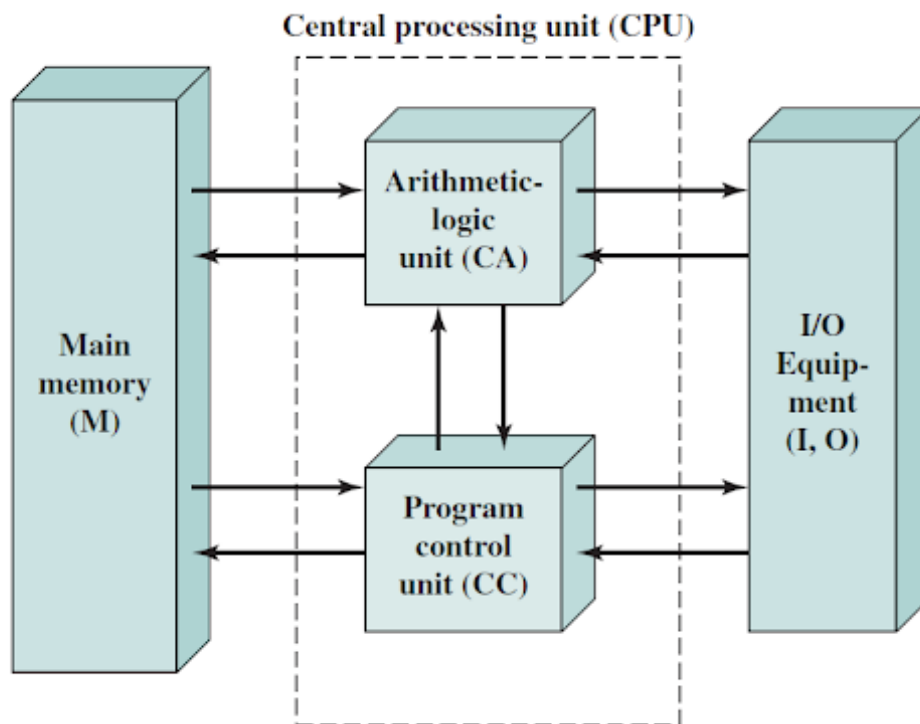
Institute of Advanced Study Computer

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 11

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.**
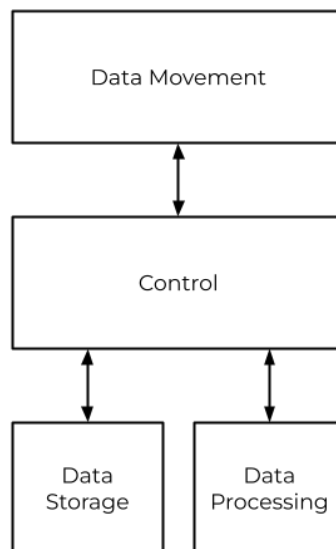
Functional view of a computer

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.
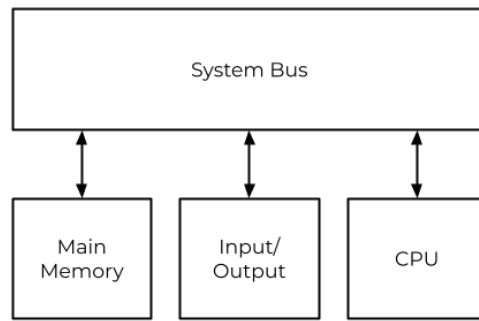
Internal structure of a computer

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.**
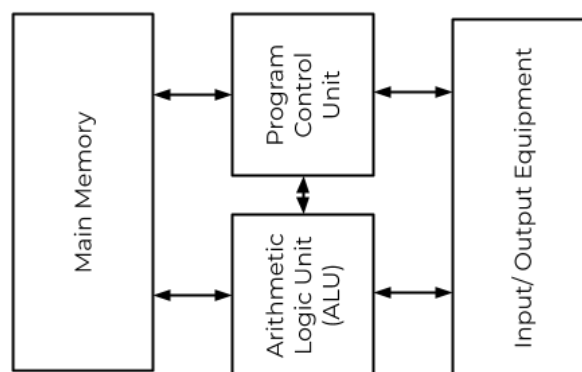
Von Neumann architecture

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 12

IAS Computer

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

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)

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 input/ output unit; or to be received from either.
**Memory Address Register (MAR)** contains the address in memory of the word to be written to or read into the MBR.
Instruction Register (IR) contains the 8-bit opcode of the instruction currently being executed.
Instruction Buffer Register (IBR) Temporarily holds the right hand instruction from a word in memory while the left hand instruction is being executed.
Program Counter (PC) holds the address of the next instruction pair to be fetched from memory.
Accumulator (AC) and Multiplier Quotient (MQ) is used to hold temporary operands and results from the ALU. The MQ will be used if the result from the ALU is too big to fit in the AC.

IAS Registers
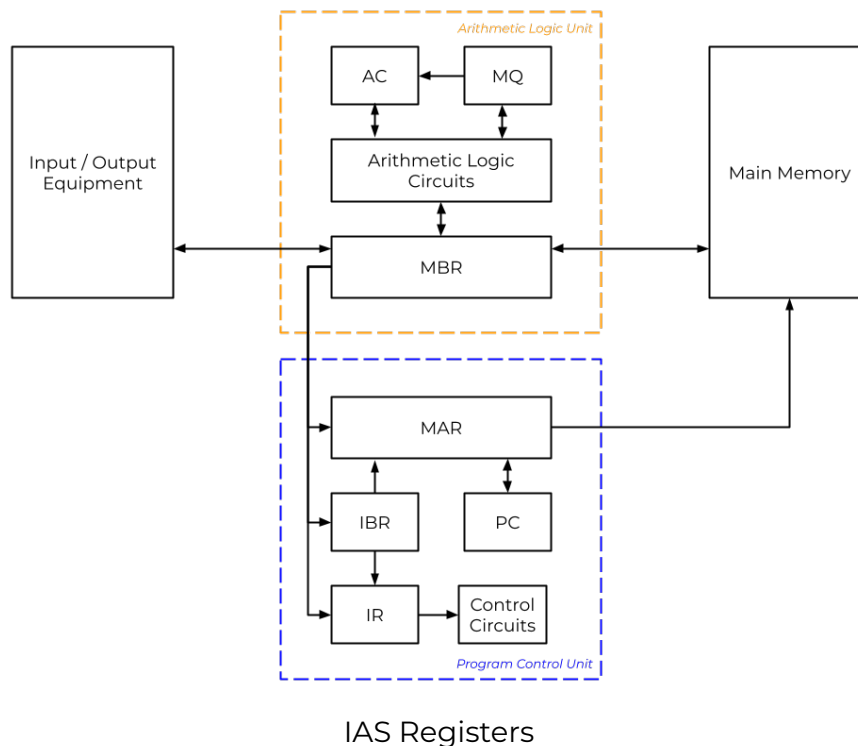
Instructions

The IAS Computer had a very limited set of instructions.

Opcode	Symbolic Representation	Description
Data Transfer		
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

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)`

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)

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 |

Programs

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

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.

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 13

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.