CO112 - Hardware

Prelude

The content discussed here is part of CO112 - Hardware (Computing MEng); taught by Bernhard Kainz, and Bjoern Schuller, in Imperial College London during the academic year 2018/19. The notes are written for my personal use, and have no guarantee of being correct (although I hope it is, for my own sake). This should be used in conjunction with the notes, and lecture slides. This course starts off fairly slow, especially if you have an idea of how logic gates work, and therefore the first parts won't be covered in much detail.

Lecture 1 Notes

This section will be covered in less detail, as we've gone through the majority of this in much greater depth during logic. However, we will need to change the notation we use in this course from the one used in logic, from using $\wedge$ to $\cdot$, $\vee$ to $+$, and from $\neg$ to $'$.

A	B	$A \cdot B$ (AND)	$A + B$ (OR)	A' (NOT)
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

The same distributivity laws apply, just like in **CO140**, as well as the simplification laws. In general, the laws should be the same as propositional logic, with the notation being slightly changed. Use 1 for $\top$, and 0 for $\bot$. We will also be using de Morgan's theorem on any number of variables (this can be proven by induction), such that $(V_1 + V_2 + V3 + ... + V_n)' \equiv V_1' \cdot V_2' \cdot V_3' \cdot ... \cdot V_n'$, and the same the other way around. This can be very useful later on, as we will often use NAND / NOR gates to reduce silicon area.

Lecture 2 Notes

The three operators covered in the first lecture can be represented by three logic gates; AND, OR, and NOT. The inverter (NOT), is represented by the circle at the end of the triangle. We can also create operations such as NAND, and NOR. Any of the first three gates can be built with just NAND gates, or just NOR gates. Let us represent A NAND B, with $A \uparrow B$.

- A' $A \uparrow A$
- $A \cdot B$ $(A \uparrow B)'$ $(A \uparrow B) \uparrow (A \uparrow B)$
- $A + B$ $(A' \cdot B')'$ (de Morgan's) $A' \uparrow B'$ $(A \uparrow A) \uparrow (B \uparrow B)$

We also need to introduce two new gates, which are commonly used in digital logic; XOR, and XNOR. Roughly, you can use the same rules for $\neg(A \leftrightarrow B)$, and $A \leftrightarrow B$ respectively. XOR is commonly represented by $A \oplus B$ (which is much shorter than $A \cdot B' + A' \cdot B$), and XNOR represented by $(A \oplus B)'$, instead of $A' \cdot B' + A \cdot B$. It has the following truth table;

A	B	$A \oplus B$	$(A \oplus B)'$
0	0	0	1
0	1	1	0
1	0	1	0
1	1	0	1

In general, with n inputs, we can have 2^n unique gates.

Lecture 14 Panopto

Putting together a manual processor; generally the block diagram takes in a sequence of binary numbers, one sequence for data, and another for instructions. This will result in a binary number. Our design is based on the von Neumann architecture, which divides the processor into arithmetic units and registers, with a shared stream for data, and instructions. Our model will be based on 8 bits.

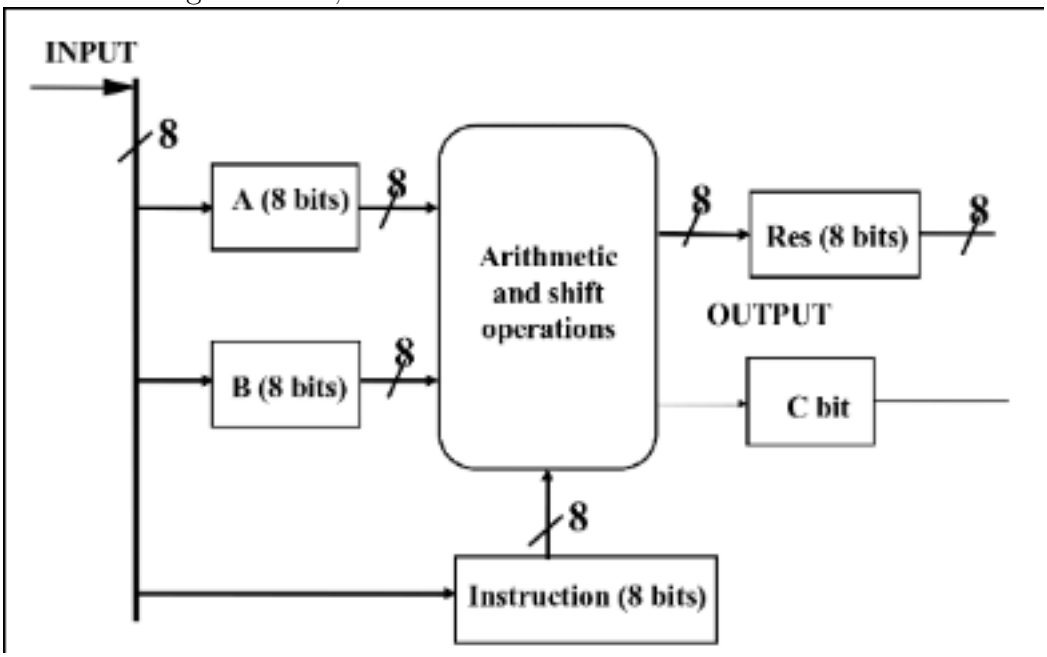
Our example action is to find the average of two numbers, A, and B, such that $R = \frac{A+B}{2}$. Since we are dealing with 8 bits, $A + B < 256$.

1. The first number is set up in input lines, and stored in register (A)
2. The same is done for the second number, and stored in register (B)
3. The arithmetic circuits are set to register (A) + (B).
4. The resulting sum is put into (A)
5. The shift circuits shift (A) one bit to the right, which is integer division by 2
6. Result is loaded into (Res).

In order to do this, we need a number of components;

- Registers
  - store data (A), and (B)
  - store result (Res)
  - one bit for carry (C)
  - store instruction (IR)
- Arithmetic circuits
  - 8-bit adder
  - 8-bit shifter

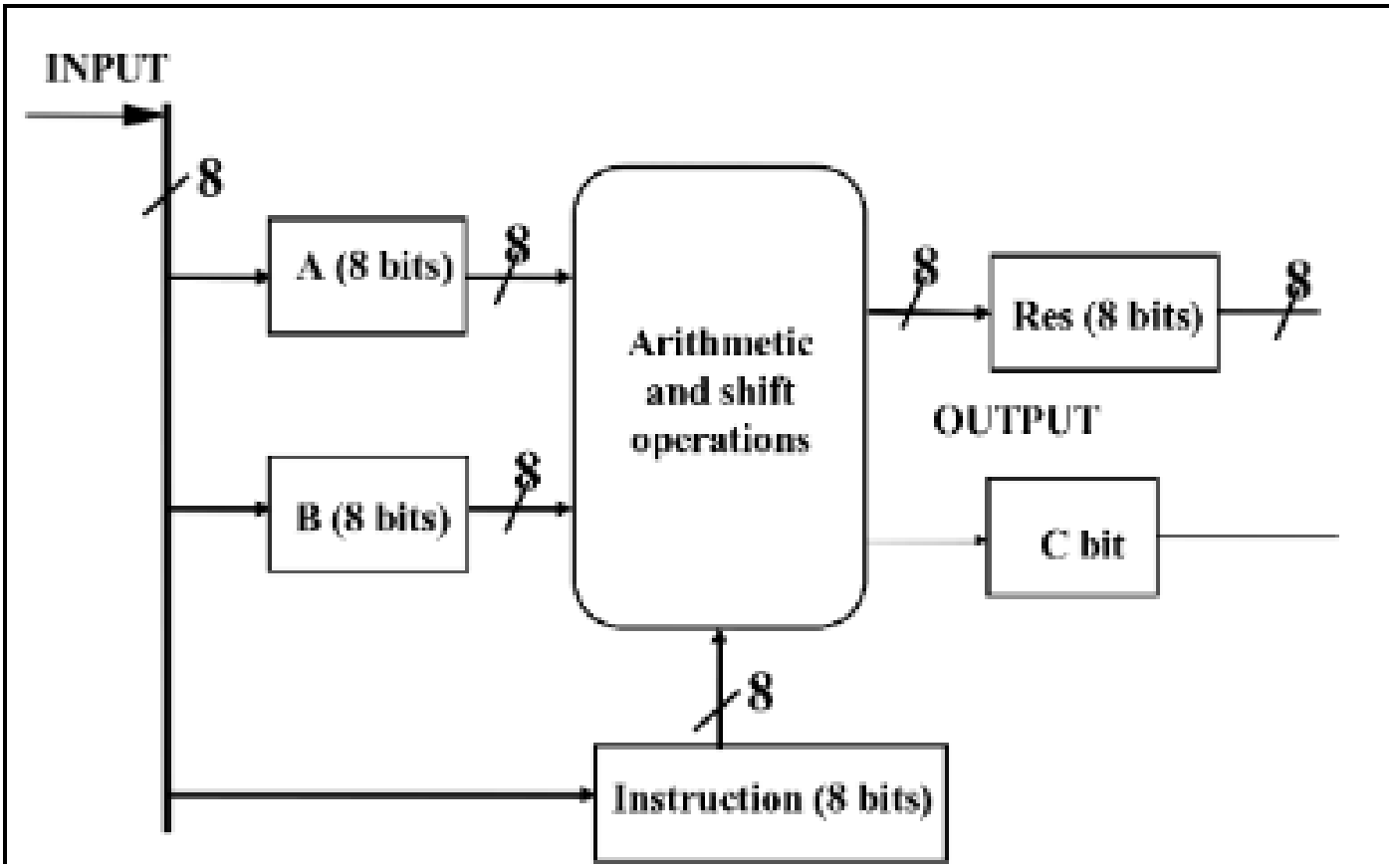
Note in the figure below, that the 8 means it is an 8-bit line.



PANOPTO 1:21:56

For an n-bit ALU, there are $n + 1$ multiplexers, as the last one handles the carry bit. The C_{in} bit for the n^{th} multiplexer is the C_{out} for the $(n-1)^{\text{th}}$ multiplexer. The C_{in} for the first multiplexer is 0.

Notes - Lecture 14


INPUT
8
A (8 bits)
8
B (8 bits)
8
Arithmetic
and shift
operations
8
Res (8 bits)
8
OUTPUT
C bit
8
Instruction (8 bits)