Department of Computing
Course DOC 112 -- Hardware

Lecture 14: Let's Put it Together! - A Manual Processor

Customer Specification

A mathematician has asked us to design a simple digital device that works similarly to a pocket calculator.

The mathematician is interested in calculations involving pairs of numbers. For data inputs A and B, the device should be capable of the following operations :

R = A PLUS B
R = A MINUS B
R = B MINUS A
R = A XOR B
R = A OR B
R = A AND B

or any of the above operations followed by a binary shift left or right e.g.

R = (A PLUS B) >> 1
R = (A XOR B) << 1

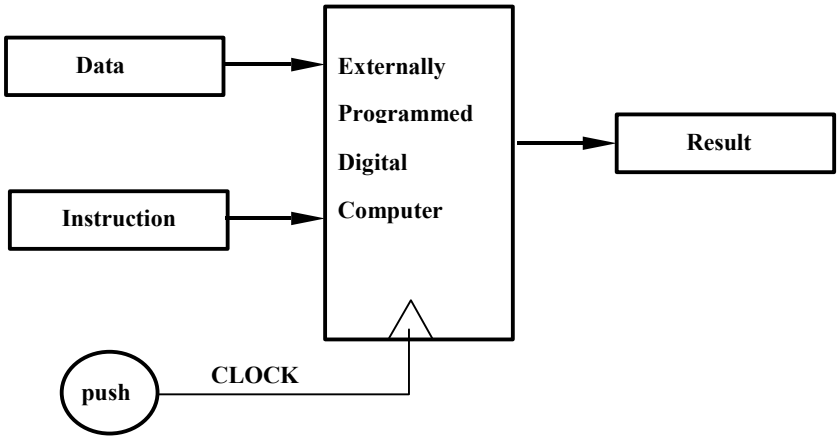
This allows the mathematician to compute averages as (A PLUS B) / 2 = (A PLUS B) >> 1

Just to be awkward, the mathematician also wants an "accumulate" mode in which the different operations can be applied to the most recent result with one new input number e.g. R = A+B+C

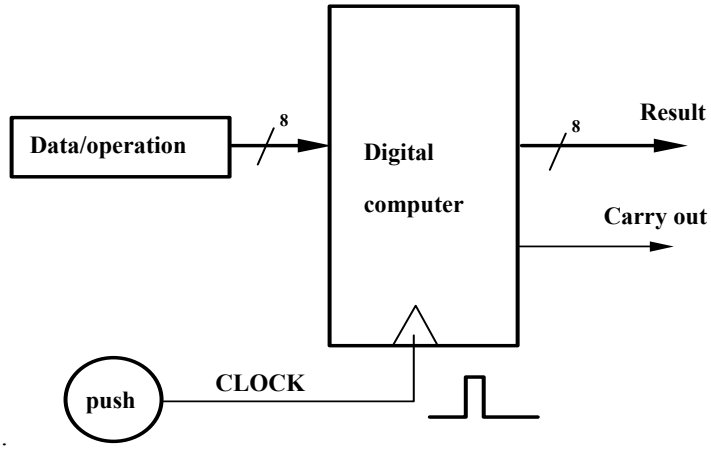
Step 1 ACC = (A PLUS B)
Step 2 R = (ACC PLUS C)

Design Specification

The simplest type of design which meets the customer specification is called an "**externally programmed**" digital computer. We will design this device over the course of the next two lectures. Externally programmed digital computers have two kinds of signal input; one for digital data, the other for "instruction" bits which tell the computer what operation(s) to perform :



Before we can design the workings of the externally programmed digital computer, we need to define the **external interface** :
Data is 8 bits wide
Data and **Instruction** will share one 8-bit set of input lines
Result is 8 bit wide with an additional 1-bit **Carry out** (which is needed for the PLUS operation)



Design by Monolithic State Machine

The Externally Programmed Digital Computer is a sequential system. The shared data/instruction inputs mean that the instruction and data inputs A, B must to be stored internally across different clock cycles as first the instruction then two data inputs are loaded.

For example, the actions to computer the average of two numbers A and B would be :

1.	**Set the input bits to represent the instruction "add then >>1" and <u>push</u>**	A clock pulse is generated and the instruction for "add then >>1" is stored internally.
2.	**Set the input bits to represent data A and <u>push</u>**	A clock pulse is generated and the bits of **A** are stored internally.
3.	**Set the input bits to represent data B and <u>push</u>**	A clock pulse is generated and the bits of **B** are stored internally.
4.	**<u>push</u>**	Operations are executed and "clocked" into the output register

The intermediate result must also be stored internally across different clock cycles in accumulate mode.

We know how to design sequential systems using state machines. So let's design our Externally Programmed Digital Computer as one large Moore Machine.

The Moore Machine "state" will consist of 8-bits data **A,** 8-bits data **B** and 8 bits **IR** (the instruction) and some extra bits to encode the correct sequence of operations. This means that there will be in excess of 24 state flip-flops. To design the state transition logic (F) we will have to design in excess of twenty-four, 24-input 1-output combinatorial circuits !

Not a productive use of time.

Design by Control Unit and Datapath

We can build an Externally Programmed Digital Computer using a far simpler state machine, by dividing the design into **Datapath** and **Control Unit**. In addition, this method allows us to re-use our arithmetic circuits from Lecture 13.

The **Datapath** consists of :

1 I/O Ports
2 Arithmetic-Logic Unit ALU and Binary Shifters
3 Data Registers
4 Connections between the above. Multiplexers route data between I/O ports, computation blocks and registers.

The **Control Unit** determines what actions occur in the processor at a given time. This is achieved by 'decoding' the opcode bits of the instruction. For the Externally Programmed Digital Computer, the control unit includes a sequential system which is implemented as a state machine. Control of computation in the Datapath is achieved in three ways:

Multiplexer Select Inputs. The outputs of the Control Unit drive the select inputs of the multiplexers in the datapath. By setting different values for the select inputs of the multiplexers, the Control Unit can control the direction of dataflow around the datapath.

ALU and Shifter Select Inputs. The Control Unit drives the select inputs of the ALU and shifter. By setting different control inputs to the ALU, the Control Unit can control which ALU operation is executed.

Register Clock Inputs. The Control Unit drives the clock inputs of the data registers. By enabling/disabling clocking of different registers at different times, the Control Unit can control which registers store the results of different computations.

Datapath of the Externally Programmed Digital Computer

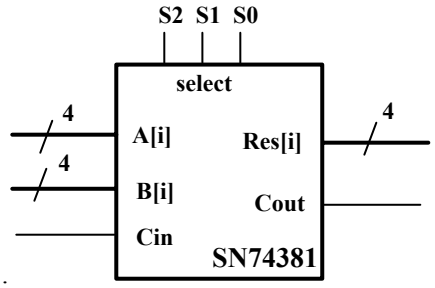
1 I/O Ports

The I/O Ports of the datapath are the 8-bit **DataIn** input, 8-bit **DataOut** output and 1-bit **CarryOut** output.

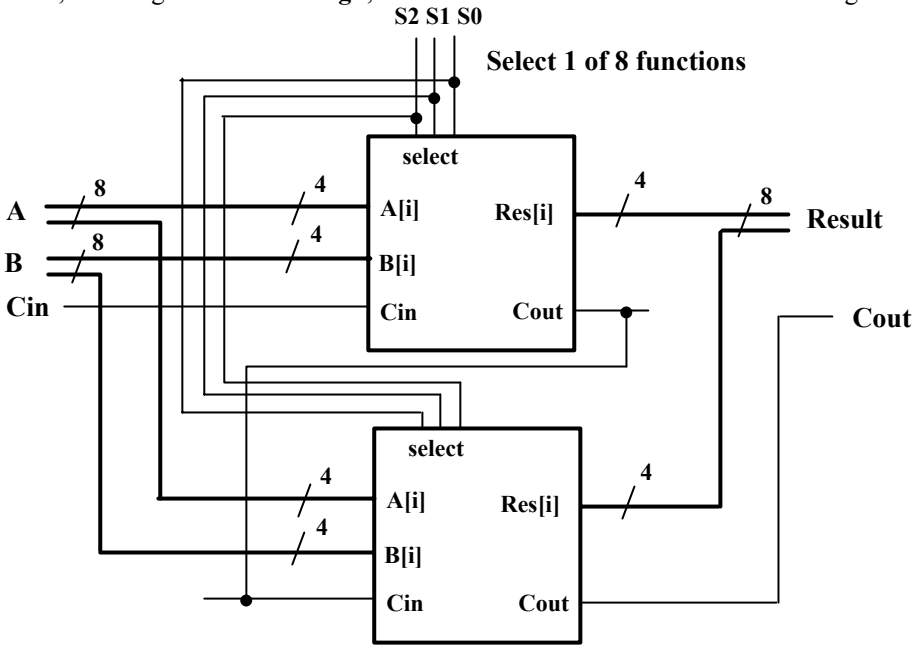
2 Arithmetic-Logic Unit ALU and Shifter

The Arithmetic-Logic Unit

We have covered both arithmetic and shifting operations in previous lectures. Now we apply our accumulated knowledge to build the arithmetic/logic unit popularly known as the **ALU**. We are in luck, we do not really have to build the arithmetic part of it. The **IC** catalogue provides us with an Arithmetic/Logic Function Generator, but, unfortunately it is only a four-bit device:



However, knowing **functional design**, we can turn two four-bit devices into one eight-bit device in a jiffy:



There are three select lines and for each bit combination we get a different function as result:

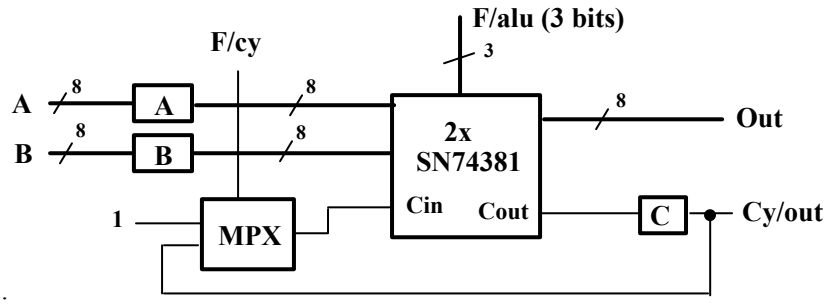
The operations that the **SN74381** device can perform are:

Selection bits:	000	001	010	011	100	101	110	111
Result:	0000	B mi A	A mi B	A pl B	A<+>B	A+B	A•B	1111

<+> = XOR

Since **A+B** looks like the logical **OR** function, we indicate arithmetical functions as **pl** (add) or **mi** (subtract) in the table.

There is this extra one bit input: **Cin**, what should be its value? A logic **0**, or a logic **1** or neither? Actually, we will be able to do a lot more with our processor if we provide some flexibility for this independent input. But how? There is a useful saying among digital designers: **If you have a problem to solve, use a multiplexer!** We can add now a **2-to-1 multiplexer** to allow two different carry bit values to input to the arithmetic unit:



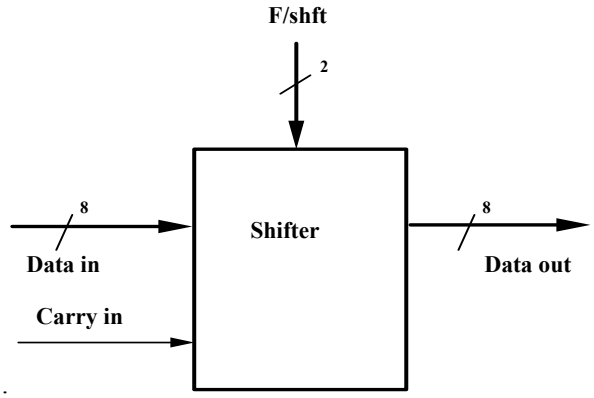
The carry bit applied to the arithmetic unit can now come for two sources. It may be a logic **1**, or the carry bit stored in the **C** register which was left there during the last operation. This arrangement looks rather arbitrary, the idea behind it is that it is not difficult to produce a zero bit in the **C** register by the selection of the appropriate ALU function. Thus, the bit can be selected either a **0** or a **1** through the multiplexer.

The three **F/alu** bits (function bits for the **ALU**) provide the selection of the arithmetic function, the one bit of **F/cy** (function bit for carry selection) controls the multiplexer and hence the carry in signal; together (four bits all) they determine the function of the arithmetic unit. For example, if the stored bit in the **C** register is **0**, **F/alu = 011**, and **F/cy = 0** (carry input of **0**), then we have a simple binary addition; however, if we have **F/alu = 011** and **F/cy = 01** (carry input of **1**), we have the function: **A plus B plus 1**.

This concludes the arithmetic part of the ALU design. We now need a binary shifter.

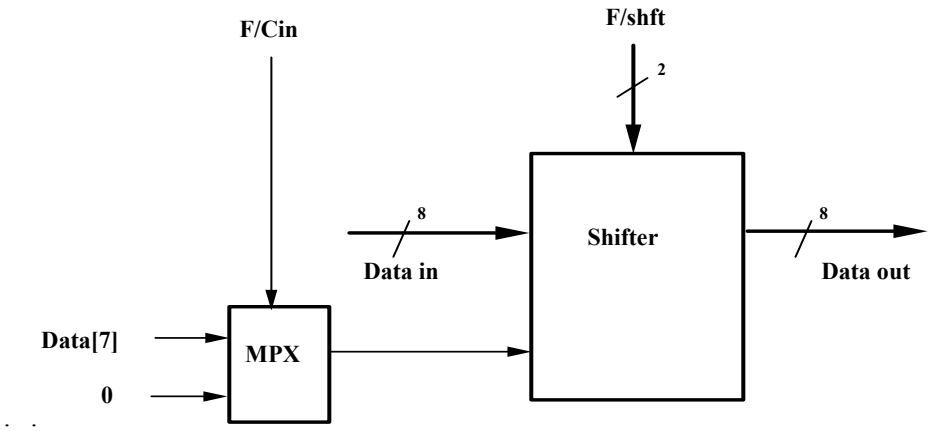
The Binary Shifter

The Binary Shifter circuit is also a combinational circuit, it has eleven bits input and eight bits output since in addition to the eight data bits, we have a **Carry in**, and two function selection bits:

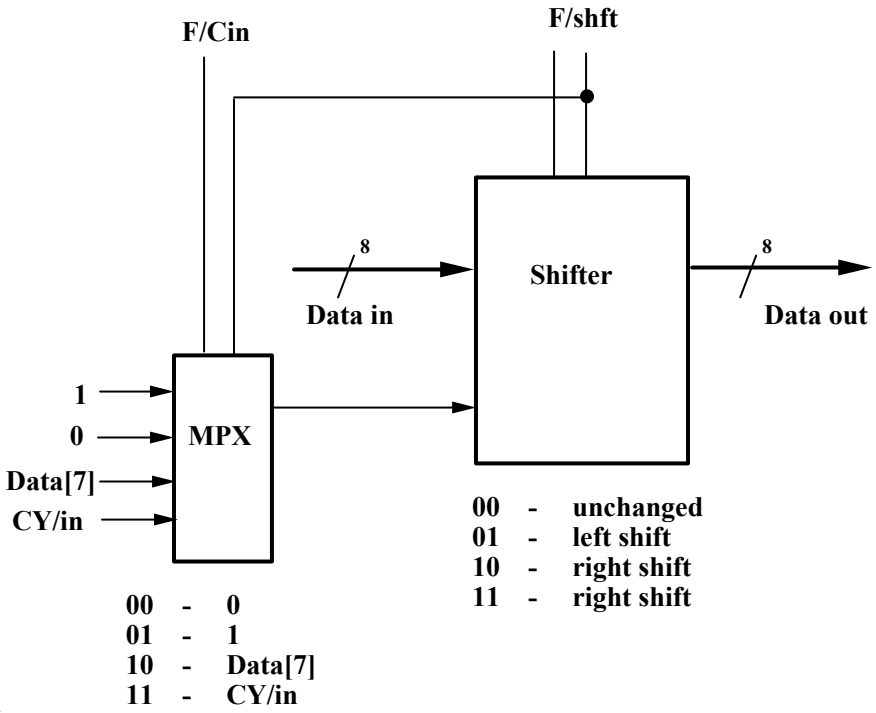


The two function selection bits can specify four different functions; there are three obvious ones: **unchanged**, **shift right**, and **shift left**. However, there are two different types of shift right functions. The **logical right shift** function shifts in a logic **0**

into the most significant bit; while the **arithmetic shift right** function leaves the most significant bit the same. We may achieve this with a multiplexer (no surprise! it is a multiplexer again!).



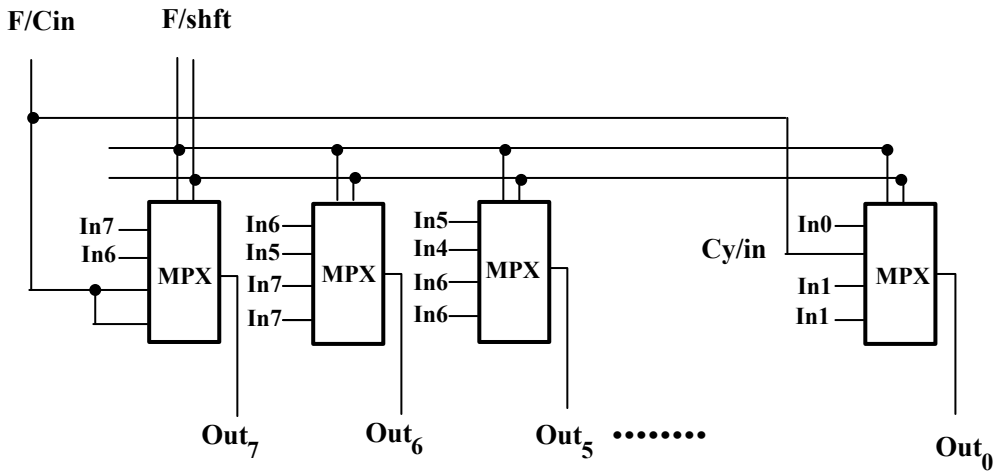
For left shift we may want **0** (for both arithmetic and logical left shift functions) or possibly an independent carry input bit. We would need a second multiplexer for this or a four-input one output multiplexer. We select the latter one and arrive at our final shifter circuit:



We have three function bits but not all eight selections provide meaningful or unique functions. We can see this in the following table:

Function select	Shift	Carry/in	Function
000	unchanged	1	unchanged
001	left shift	0	arithm/logic left shift
010	right shift	1	????????
011	right shift	0	logical right shift
100	unchanged	Data[7]	unchanged
101	left shift	CY/in	left shift with carry
110	right shift	Data[7]	arithmetic right shift
111	right shift	CY/in	right shift with carry

Again, if we wonder how to build the shift circuit, multiplexers come to our aid. Eight **4-to-1 multiplexers** are used with two selection control bits. The connection of the data input bits to the multiplexer determines the output functions. For **unchanged** we have **Out[i] = In[i]**. For **left shift** we have **Out[i] = In[i-1]**, except for the least significant bit for which **Out[0] = CY/in.** For **right shift** we have **Out[i] = In[i+1]**, except for the most significant bit for which we have **Out[7] = Cy/in.** Quite simple (??) really!



3 Data Registers

During the design, the choice of internal datapath registers may change as one finds out that the required operations cannot be executed with the original configuration. The datapath for the Externally Programmed Digital Computer will contain four data registers

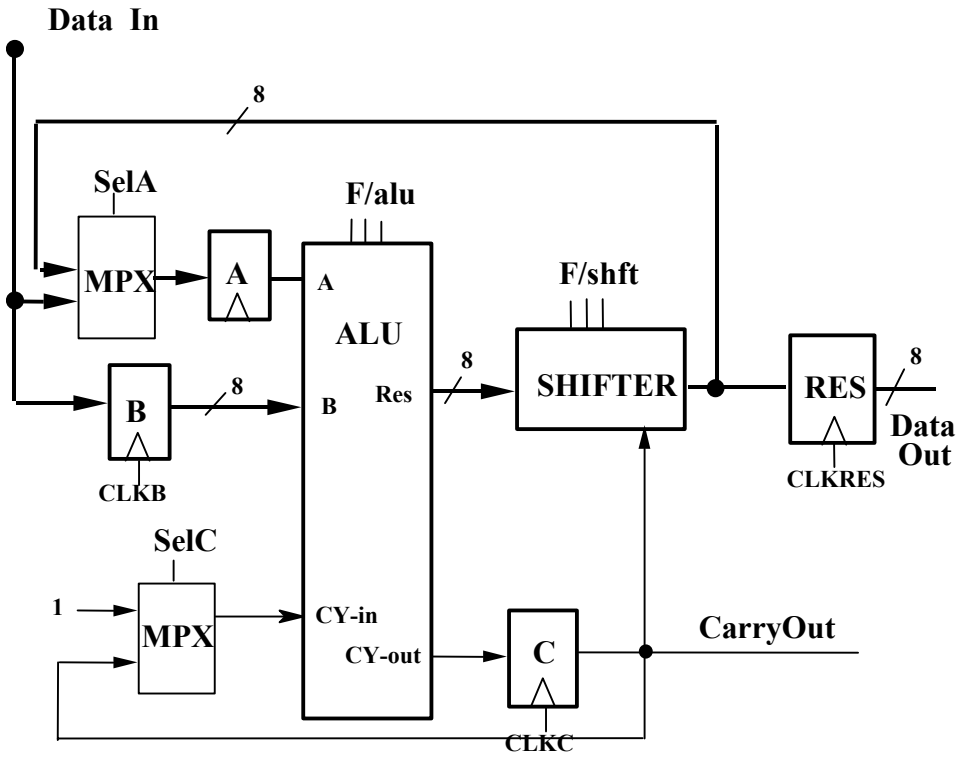
8-bit Data Register **A**	To store data input A. Also used to store intermediate result in accumulate mode.
8-bit Data Register **B**	To store data input B.
1-bit Data Register **C**	To store carry-out from the ALU
8-bit Data Register **RES**	To store the final result.

The clock inputs for each data register : CLKA, CLKB, CLKC and CLKRES are driven by the Control Unit.

4 Datapath Connections and Multiplexers

The connections and multiplexers for the final datapath are shown on the next page. Notice that on the data paths diagram there is no information how and when data are transferred; only that data can be transferred between the input and the three eight bit registers and that output is provided from the eight bit result register and the one bit carry register (a single flip-flop). The Control Unit will determine how and when data are transferred.

Multiplexer with SelA input enables either a new input, or an intermediate result to be stored in accumulate mode.
Multiplexer with SelC input enables control of ALU carry-in.



Design of the control unit is left to the next lecture. The control unit will be connected to

Data In (to collect the instruction to be decoded)
Multiplexer Select inputs :
SelA
SelC
ALU and Shifter select inputs :
F/alu
F/shft
Register Clocks :
CLKC
CLKRES, CLKA, CLKB