Lecture 14: Let's put together a Manual Processor

Customer Specification

A mathematician has asked us to design a simple digital device that works rather like a pocket calculator. She is interested in calculations involving pairs of numbers. For data inputs A and B, the device should be capable of performing 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

All these we could do using the arithmetic circuits that we designed in the previous lecture, however we are also asked to provide shift facilities allowing calculations such as:

R = A $\rightarrow$ 1
R = A $\leftarrow$ 1

The arrow indicates a shift to the left or right. The binary number on the left of the arrow is shifted by the number of binary digits indicated on the right of the arrow. Just to be awkward, the mathematician also wants an accumulate mode in which the different operations can be applied to the most recent result, perhaps using a third number. For example: R = A+B+C could be computed using:

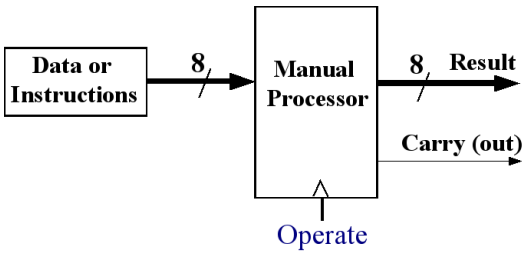
Step 1: ACC = (A PLUS B)
Step 2: R = (ACC PLUS C)

Similarly it would be possible to calculate:

(A PLUS B) / 2 = (A PLUS B) $\rightarrow$ 1

Design Specification

The simplest type of design which meets the customer specification is called an *externally programmed digital computer* or *manual processor*. (It is a conceptual device, so don't go to PC World and expect to buy one!) We will design this device over the course of the next two lectures. Externally programmed digital computers have two kinds of input: one for data and the other for instructions that tell the computer what operation(s) to perform. Conceptually we can think of the data and instruction inputs as being separate, however in practice we will use just one input, which can be used for either instructions of data.



Before we can design the detailed workings of the manual processor, we need to define the external interface. We will choose to use an eight bit architecture so: the input data are 8 bits wide and the result is 8 bits wide with an additional 1-bit carry out (which is needed for the PLUS operation). The 8/ in the figure above indicates the presence of eight parallel lines.

Design by Monolithic State Machine

The manual processor is a sequential system. The shared data/instruction lines mean that the instruction and data inputs must to be read and stored internally in turn according to the clock input and further clock cycles may be needed for processing. For example, the steps required to compute the average of two numbers A and B might be:

External (manual) operation	On the clock pulse
1. Set the input bits to the first operand (A)	Load the input bits into register A
2. Set the input bits to the second operand (B)	Load the input bits into register B
3. Set the input bits to be the instruction A plus B	Load the input bits into the instruction register IR
4. -	The result is calculated
5. -	Load the result back into register A
6. -	Set the input bits to be the instruction A $\rightarrow$ 1
7. -	Load the result into a result register
8. Collect the result from the output bits	-

We see that the manual processor is a sequential digital circuit, and we could try to design it as a large Moore machine. The machine state would consist of 8-bits of data in register A, 8-bits of data in register B and 8 instruction bits in IR and perhaps an extra carry bit. This means that there will be in excess of 24 state flip-flops. To design the state sequencing logic we will have to design twenty-four, 24-input 1-output combinatorial circuits! Not a productive use of time - it looks like we will need to take the functional approach.

Design by Control Unit and Data Paths

We can build an externally programmed digital computer using a far simpler state machine, by dividing the design into the Data Paths and the Control Unit (or controller). In addition, this method allows us to use our arithmetic circuits, which we will build into an Arithmetic-Logic Unit (ALU). The Data Paths consist of:

The I/O Ports
The arithmetic-logic unit ALU and shifters
Data registers
Connections between the above controlled by multiplexers.

The Control Unit determines what sequence of actions occurs in the processor at any time. This is achieved by decoding the instruction bits. The control unit is implemented as a state machine and interacts with the data paths in three ways:

1. *Multiplexer Select Inputs.* The outputs of the control unit provide the select inputs of the multiplexers. By setting different values for the select inputs, the control unit can determine the directions of data flow around the data paths.

2. *ALU and Shifter Function Select Inputs.* The control unit sets the select inputs of the ALU and shifter thus determining the arithmetic operation that is carried out.

3. *Register Clock Inputs.* The control unit provides the clock inputs of the data registers, thus determining which registers are loaded at each step of a computation.

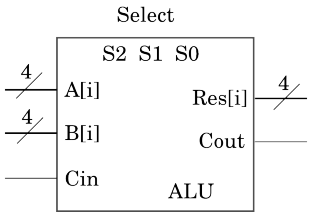
For the time being we will design a simple controller which just carries out the third function. The multiplexer select inputs and the arithmetic functions will be taken directly from the instruction register.

1 The I/O Ports

The I/O Ports are the 8-bit input, the 8-bit output and 1-bit carry lines. They are simply data highways or buses.

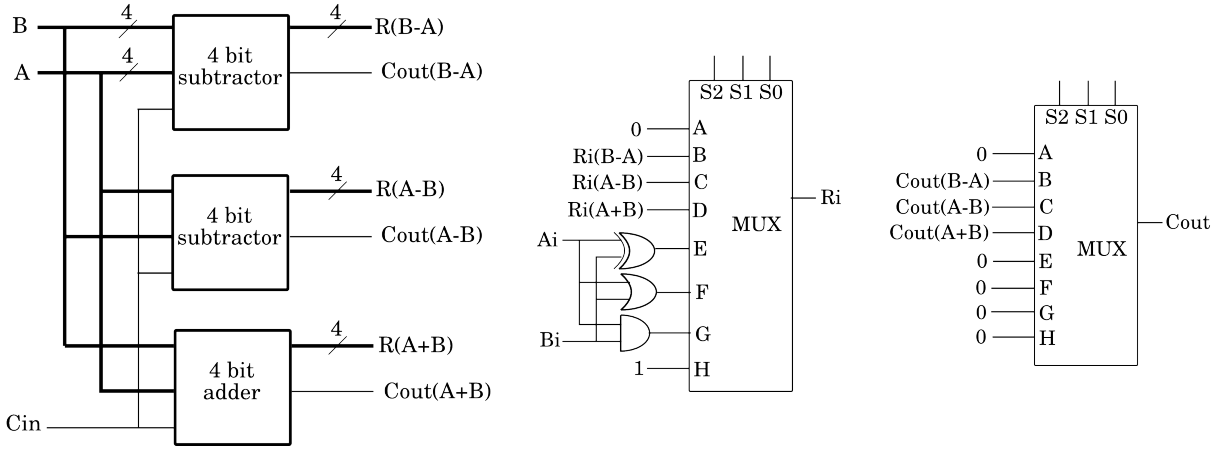
2 The Arithmetic-Logic Unit

We have covered arithmetic operations in previous lectures. Now we apply our accumulated knowledge to build the arithmetic/logic unit popularly known as the ALU. Many hardware design systems will provide ALU circuits in the form of a ready made functional block:

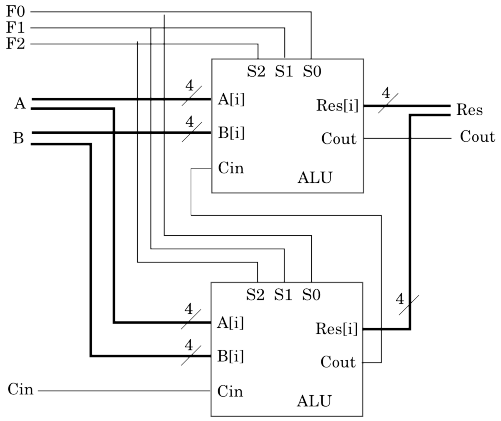


Selection Bits	000	001	010	011	100	101	110	111
Function	0	B-A	A-B	A plus B	A xor B	A+B	A.B	-1

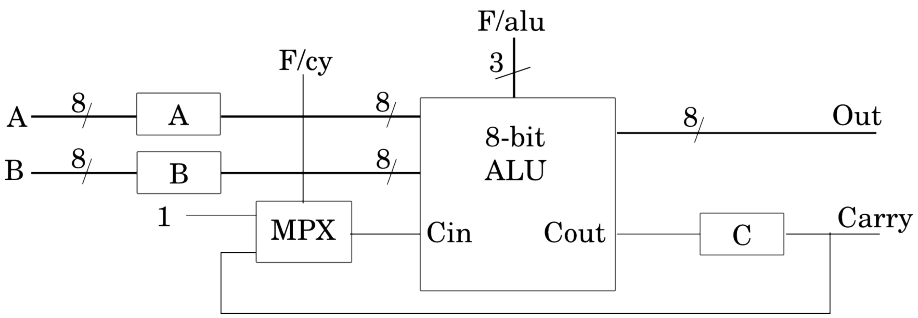
It is not too difficult to build an ALU ourselves. We will first design a four bit ALU by using four bit adders and subtractors with common inputs, a multiplexer to select each output bit and a multiplexer to select the carry.



And now, knowing functional design, we can scale up our design to eight bits (or more) without difficulty.



The outputs of two internal registers A and B will provide the inputs for the ALU. We will use A and B to store data items that are received on the input lines at different times. We will also find it convenient to store the result of carry in a register (C). How should we use the one bit Cin line of the ALU? Actually, we will be able to do a lot more with our processor if we provide some flexibility for this input. But how? There is a well known 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:

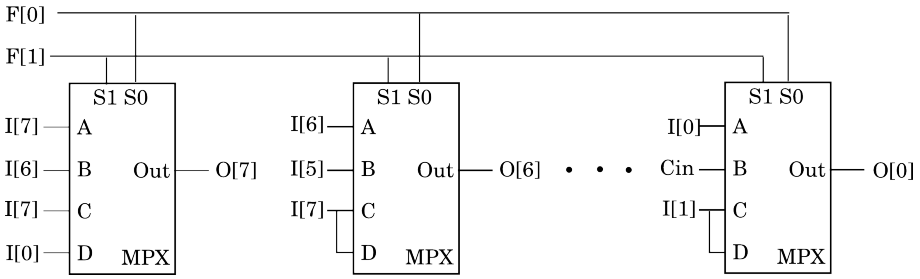


The Cin bit may be a logic 1, or the carry bit stored in the C register which was left there during the last operation. This arrangement looks rather arbitrary, the idea behind it is that it is not difficult to produce a zero bit in the C register by the selection of the appropriate ALU function. Thus, the bit can be selected either a 0 or a 1 through the multiplexer. Alternatively, if we want high precision arithmetic, say 16 bit, we can use the carry from the least significant byte result when computing the most significant byte result.

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 = 1 (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.

3. The Binary Shifter

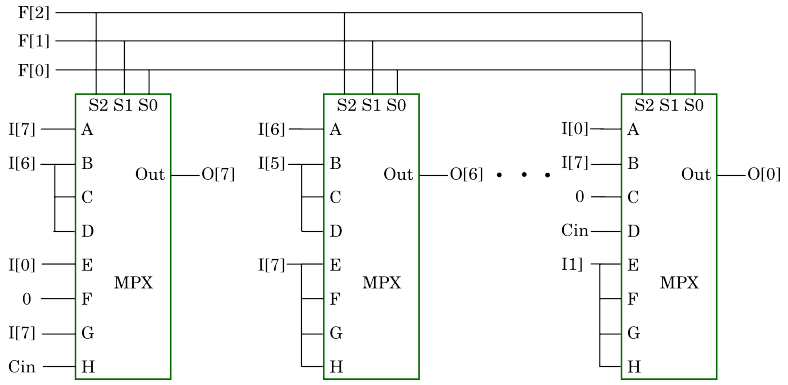
The binary shifter is different from the shift register which we designed earlier in the course. It is a simple combinational circuit with no internal storage. A basic shifter will have four modes controlled by two selection bits, eight data input lines and one carry in line. We can implement it by using - you guessed it - just multiplexers. We need one multiplexer for each data bit which simply connects a different input to the output according to the selection.



The two function selection bits can specify four different functions. The first (00:A) passes the input to the output unchanged. So for example output bit 7 is connected to input bit 7. The second function (01:B) is shift left, with a carry in at the least significant bit. This time the input bit 6 is connected to the output bit 7, and so on. The third function (10:C) is arithmetic shift right, with the top bit (bit 7) duplicated, and the last function (11:D) is rotate right which is a right shift with the bottom bit shifted into the top bit. This design can be used for a large number of purposes, but it only implements a few of the possible shifting function that we might want to use. We will probably have good reasons to use the following shifts:

F[2]	F[1]	F[0]	Shift	Carry	Function
0	0	0			unchanged
0	0	1	left		rotate left
0	1	0	left	0	arithmetic left shift
0	1	1	left	Cin	left shift with carry
1	0	0	right		rotate right
1	0	1	right	0	logical right shift
1	1	0	right	Input[7]	arithmetic right shift
1	1	1	right	Cin	shift right with carry

This more complex shifter is again implemented using one 8-1 multiplexer per bit.



4 Data Registers

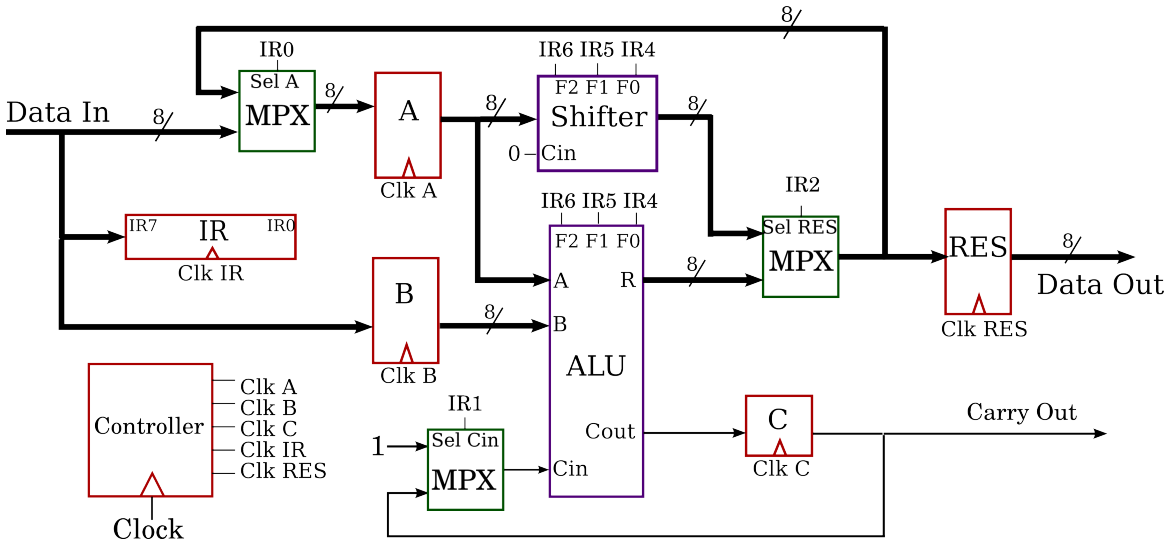
Our manual processor will contain just four data registers

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 controller.

5 The Data Path Connections and Multiplexers

The connections and multiplexers for the final data paths are shown below. Notice that on the data path diagram there is no information as to how or when data are transferred. It only tells us 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). A Control Unit (to be designed in the next lecture) will determine how and when data are transferred. There are three multiplexers in the data path diagram. Multiplexer "Sel A" connects either a new input, or an intermediate result to the input of register A. This allows computations to be done in accumulate mode. Although this is depicted as one multiplexer it is in fact eight two input multiplexers with a common select line, switching each of the eight lines forming the input to register A. We have already discussed Multiplexer "Sel Cin" which determined the ALU carry-in. Multiplexer "Sel RES" selects whether the output of the ALU or the output of the Shifter is loaded into the results register.



The details of the controller are not shown in the above diagram. It will make the processor go through a number of steps to execute a programme instruction. It is a synchronous sequential circuit that controls which register is loaded at each step of the execution. It will do this by generating the outputs ClkC, ClkR, ClkA, ClkB, ClkRES and ClkIR. We will design it next lecture.