Lecture 14

Let's put together a Manual Processor

The processor

Inside every computer there is at least one processor which can take an instruction, some operands and produce a result.

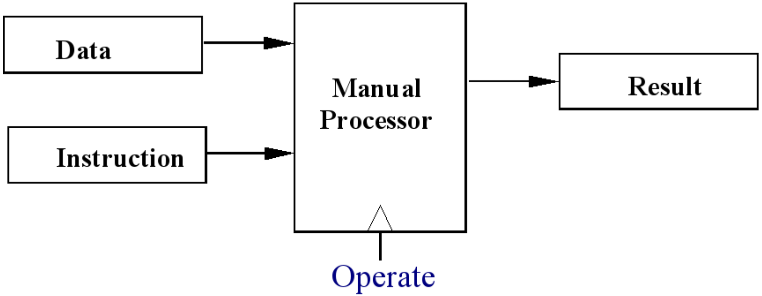
Processors can be operated in different ways for example as:

- A central processor unit (CPU)
- A peripheral of another computer
  - Array Processor
  - Graphics Processor (GPU)
- An manually programmed processor
  - Stand alone calculator

We will now design an manual (or externally programmed) processor.

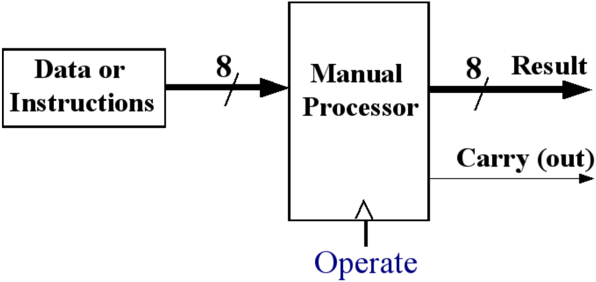
The Block Diagram of a Processor

Both the data and the instruction will be binary numbers. The processing will be a sequence of one or more steps controlled by a clock. The result will be a binary number.



An 8 bit processor architecture

We will base our design on the von Neumann architecture (1945) which subdivided the components of a processor into arithmetic units and registers, and had a common input stream for data and instructions. We will carry out processing on 8 bit bytes (similar to the processors of the 1970's).



The Actions of a Simple Processor

As an example we will find the average of two numbers.

$$\text{Result} = (A+B)/2$$

Because of our choice of architecture all numbers must be represented in 8 bits.

Thus the sum A+B must be less than 256.

We will see later on how to extend the processing to cope with larger numbers.

The Actions of a Simple Processor

The following steps are carried out:

1. Set up the processor to receive the first number from the input stream and save it in a register (A)
2. Set up the processor to receive the second number from the input stream and save it in a register (B)
3. Set the processor to add register A to register B. and then shift the result right by one bit.
4. Set up the processor to load the results on the output register (Res).

The processor is a sequential digital circuit

Designing a Processor

From the example we see that we need a number of different components to make our processor:

Registers:

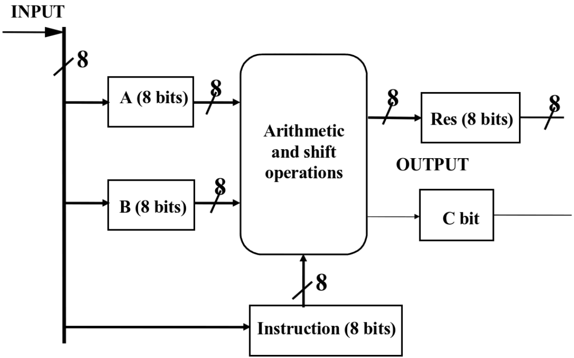
- Registers to store the input data (A) & (B)
- A register to store the result (Res)
- A one bit register to store the carry (C)(if any)
- A register to store the instruction (IR)

Arithmetic Circuits:

- An 8 bit adder
- An 8 bit shifter

The data path diagram

The example also suggests that the registers and arithmetic units must be connected in a specific way. A simple data path diagram might be:



The Data path Diagram

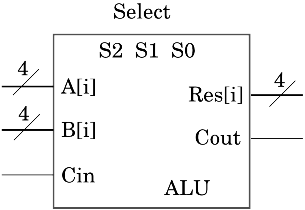
Note the following about the data path diagram:

- There is no information about when the data transfers occur. The diagram shows only the possible paths where data can be transferred.
- The arithmetic and shift operations are done by combinational circuits without more registers.
- The function of the arithmetic circuits are controlled by the bits in the instruction register.
- The processor is unable to execute further operations on the results.

The Arithmetic-Logic Unit (ALU)

We now design one important component of the central processor unit - the ALU - which carries out arithmetic or logic operations on its two inputs A and B.

We will design a 4-bit unit that can be used as a building block to construct ALUs of any precision.



The select lines (S2, S1, S0) determine the function of A and B that appears on the output.

The ALU functions

The selection bits determine which out of 8 possible functions is used.

Selection	Function
000	0
001	B-A
010	A-B
011	A plus B
100	A XOR B
101	A OR B
110	A AND B
111	-1

When Cin = 1 three operations change:

011: Res = A plus B plus 1

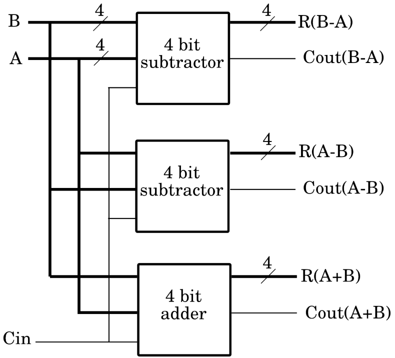
010: Res = A - B - 1

001: Res = B - A - 1

Designing the ALU

An arithmetic logic unit is a simple combinatorial circuit that can be built from components that we have already designed.

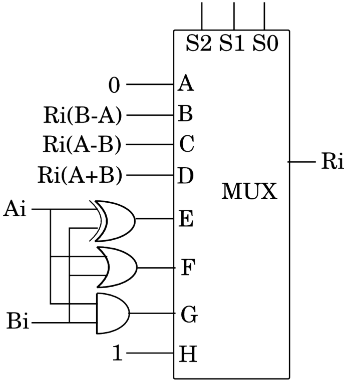
To provide the arithmetic functions we use adders and subtractors.



Designing the ALU

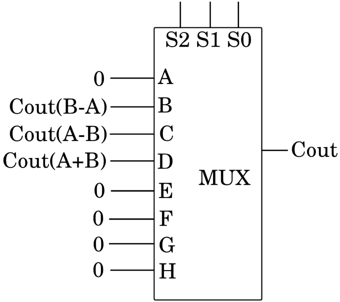
The logic functions are provided by simple gates.

We need a multiplexer for each bit of the ALU to make the function selection.



Designing the ALU

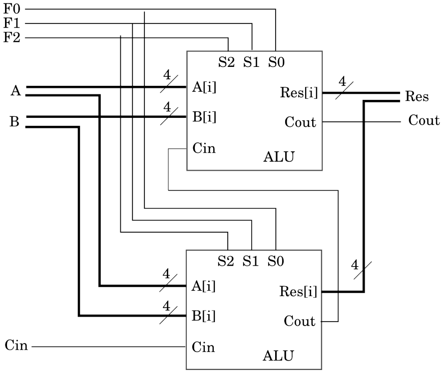
To finish the job we need one more multiplexer to provide the carry out.



Extending the ALU to 8 bits

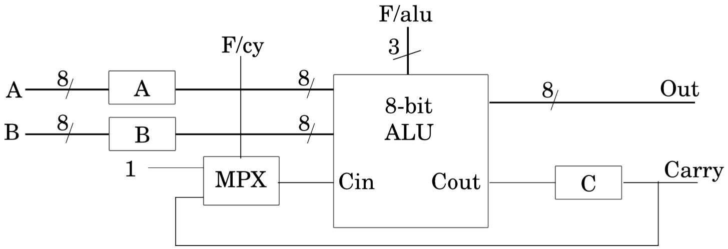
We can now follow the functional approach and extend our ALU to 8 bits

Similarly we can scale it up to 32 bits or larger as the need arises.



Organising the Carry

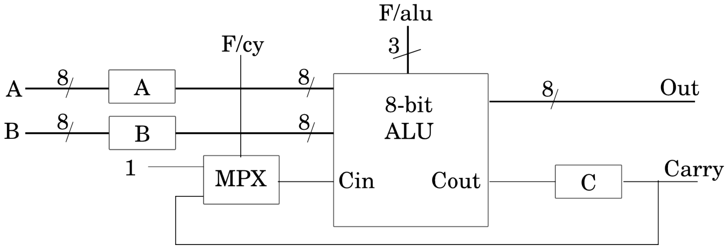
We can use a 2-to-1 multiplexer to allow two different carry input bits. The Carry in may be set to 1 or to the previous Carry out.



The carry may be set to zero by setting the register C to zero. Thus the full range of arithmetic operations can be used.

Problem

Why set the carry this way? Wouldn't it be simpler to connect the multiplexer inputs to 1 & 0 rather than using register C?

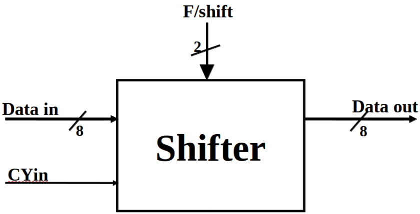


The Shifter

- Earlier in the course we discussed a specialised register, called the shift register, which could shift its contents let or right depending on a multiplexer selection.
- We could incorporate such a shift register in our design, but instead we will make use of a simpler shifter which does not store the result.
- We can achieve all the functionality we need by using multiplexers, and avoid the need for complex clocking arrangements.

A four function Shifter

Our first shifter design will have four functions determined by two selection bits and will have a data length of eight bits.



The Shifter Functions

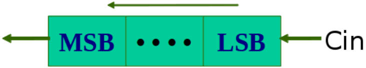
The basic shifter will perform four operations depending on its two control inputs. These are:

- 00 Hold
- 01 Arithmetic shift left (with carry in)
- 10 Arithmetic shift right
- 11 Rotate right

It is implemented simply using one multiplexer per bit.

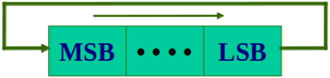
Arithmetic Shifts

Arithmetic Left Shift

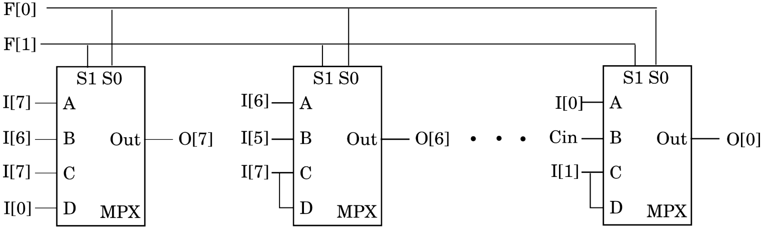


Arithmetic Right Shift

Rotate Right



The four function shifter circuit

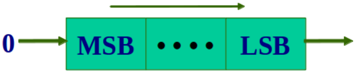


00 (A) Hold	01 (B) Arithmetic Shift Left	10 (C) Arithmetic Shift Right	11 (D) Rotate Right

Adding more functions to the Shifter

There are many other different possible shifts that programmers may want to use. One example is:

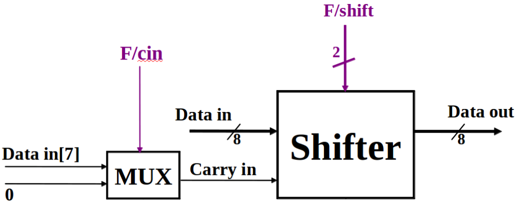
Logical Shift Right:



This could be done by using first an ALU operation (Data AND 11111110) followed by a rotate right, but that would involve more processing steps and therefore be slow.

Adding more functions to the Shifter

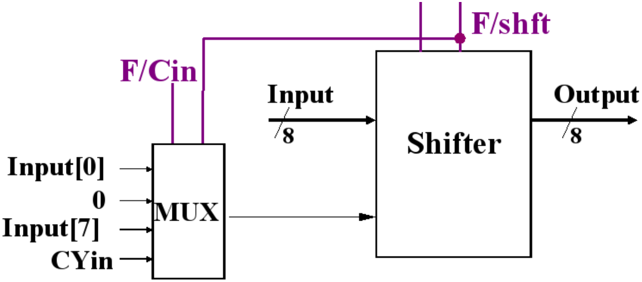
An alternative would be to add another multiplexer to switch between arithmetic right shift and logical right shift.



This Carry in connects to the C input of the most significant multiplexer in the above design. The Cin for the arithmetic left shift remains unchanged.

Adding more functions to the Shifter

More functionality can be provided by making the carry multiplexer four way.



00 -- Input[0]
01 -- 0
10 -- Input[7]
11 - Carry in

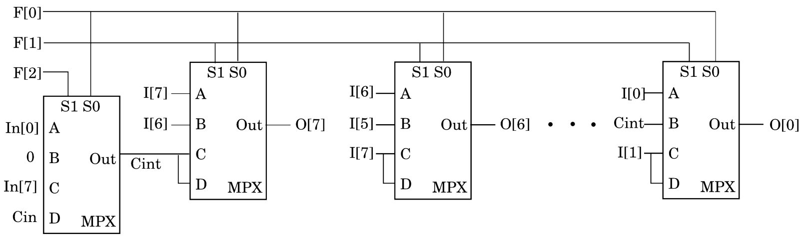
00 -- unchanged
01 -- left shift
10 -- right shift
11 -- right shift

The seven function shifter

A variety of shifts is now provided, but functions [0,0,0] and [1,0,0] are the same.

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

The circuit of the seven function shifter



Because of the way we have designed the carry we can't use this design as a building block, but it is a trivial matter to design a similar shifter of any precision.

The Data Path Diagram Again

We now can put more detail onto the data path diagram