Bit Manipulation

CIE AS & A Level

Computer Science Revision Notes 9618

Syllabus Content
4.3 Bit Manipulation
Show understanding of and perform binary shifts
Notes and guidance

logical, arithmetic and cyclic, Left shift, right shift

Show understanding of how bit manipulation can be used to monitor / control a device
Carry out bit manipulation operations. Test and set a bit (using bit masking)

Key terms

Shift – moving the bits stored in a register a given number of places within the register; there are different types of shift.

Logical shift – bits shifted out of the register are replaced with zeros.

Arithmetic shift – the sign of the number is preserved.

Cyclic shift – no bits are lost, bits shifted out of one end of the register are introduced at the other end of the register.

Left shift – bits are shifted to the left.

Right shift – bits are shifted to the right.

Monitor – to automatically take readings from a device.

Control – to automatically take readings from a device, then use the data from those readings to adjust the device.

Mask – a number that is used with the logical operators AND, OR or XOR to identify, remove or set a single bit or group of bits in an address or register.

Binary Shifts:

A **shift** involves moving the bits stored in a register a given number of places within the register. Each bit within the register may be used for a different purpose. For example, in the IR each bit identifies a different interrupt.

There are several different types of shift.

Logical shift – bits shifted out of the register are replaced with zeros. For example, an 8-bit register containing the binary value 10101111 shifted left logically three places would become 01111000.

Arithmetic shift – the sign of the number is preserved. For example, an 8-bit register containing the binary value 10101111 shifted right arithmetically three places would become 11110101. Arithmetic shifts can be used for multiplication or division by powers of two.

Cyclic shift – no bits are lost during a shift. Bits shifted out of one end of the register are introduced at the other end of the register. For example, an 8-bit register containing the binary value 10101111 shifted left cyclically three places would become 01111101.

Left shift – bits are shifted to the left; gives the direction of shift for logical, arithmetic and cyclic shifts.

Right shift – bits are shifted to the right; gives the direction of shift for logical, arithmetic and cyclic shifts.

Table below shows the logical shifts that you are expected to use in Assembly Language:

Instruction		Explanation
Opcode	**Operand**	
LSL	n	Bits in ACC are shifted logically n places to the left. Zeros are introduced on the right-hand end
LSR	n	Bits in ACC are shifted logically n places to the right. Zeros are introduced on the left-hand end
Shifts are always performed on the ACC		

Logical Vs. Arithmetic Shift

Logical Shift and *Arithmetic Shift* are bit manipulation operations (bitwise operations).

Logical Shift

- A *Left Logical Shift* of one position moves each bit to the left by one. The vacant least significant bit (LSB) is filled with zero and the most significant bit (MSB) is discarded.
- A *Right Logical Shift* of one position moves each bit to the right by one. The least significant bit is discarded and the vacant MSB is filled with zero.
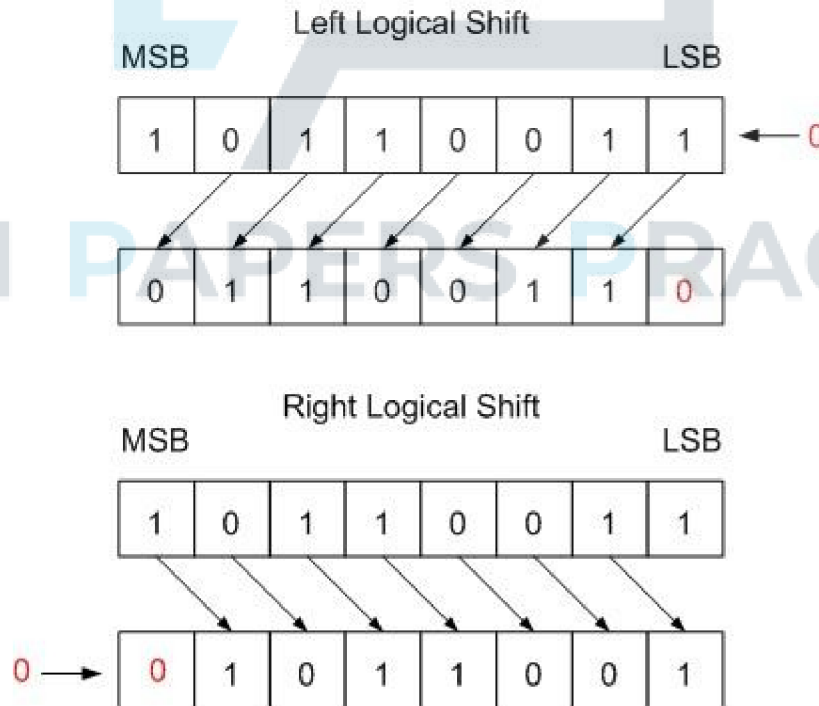
Fig. 1 Logical Shift by one bit

Arithmetic Shift

☐ A *Left Arithmetic Shift* of one position moves each bit to the left by one. The vacant least significant bit (LSB) is filled with zero and the most significant bit (MSB) is discarded. It is identical to Left Logical Shift.

☐ A *Right Arithmetic Shift* of one position moves each bit to the right by one.
The least significant bit is discarded and the vacant MSB is filled with the value of the previous (now shifted one position to the right) MSB.
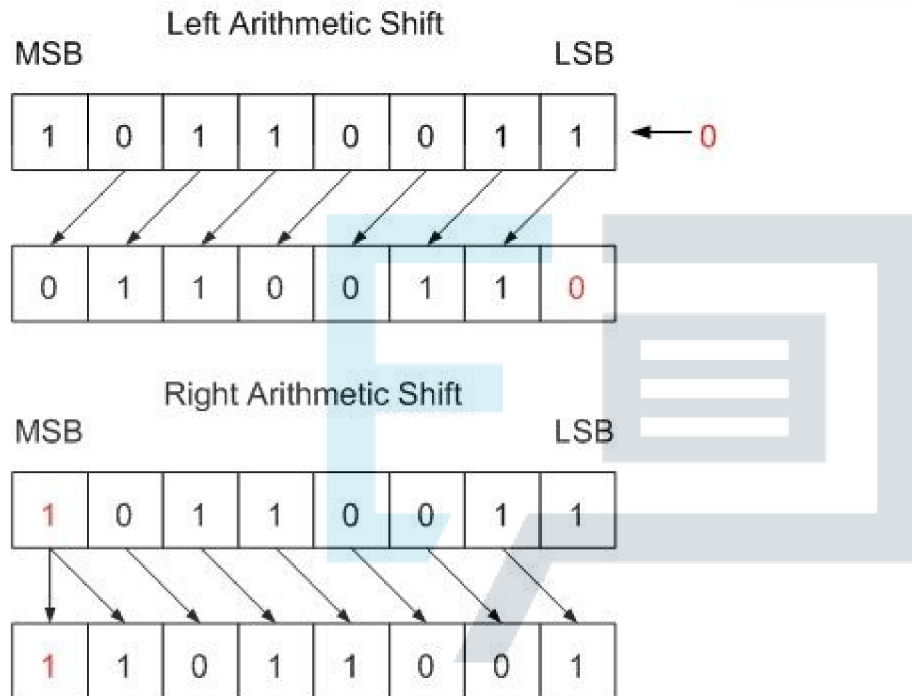

Fig. 1 Left and Right Arithmetic Shift by One Bit

Arithmetic Shift operations can be used for dividing or multiplying an integer variable.

Multiplication by left shift:

The result of a Left Shift operation is a multiplication by 2n , where n is the number of shifted bit positions.

Masking:

Masking means to keep/change/remove a desired part of information.

A bitmask is a way of accessing a particular bit. The bitmask is a number which has 0 in all bits that we don't care about, and a 1 for the bit(s) that we want to examine. By ANDing the bitmask with the original number, we can "extract" the bit(s) – if that bit was 0, then the new number will be completely zero; if the bit was 1, then the new number will be non-zero.

Consider these Masks:

	AND	OR	XOR
0	Clears the value	Retains the value	Retains the value
1	Retains the value	Sets the value	Inverts the value
Uses	Setting chosen bits to 0	Setting chosen bits to 1	Inverting chosen bits, finding differences between

Another part of the monitoring and control program would be checking whether any of the four flags were set. The machine code for running such a program could use individual bits to represent each flag. The way that flags could be set and read are illustrated by the following assembly language code fragments in which the three least significant bits (positions 0, 1 and 2) of the byte are used as flags:

```
LDD  0034     Loads a byte into the accumulator from an address
AND  #B00000000   Uses a bitwise AND operation of the contents of the accumulator with the
              operand to convert each bit to 0
STO  0034     Stores the altered byte in the original address
  :
LDD  0034
XOR  #B00000001   Uses a bitwise XOR operation of the contents of the accumulator with the
              operand to toggle the value of the bit stored in position 0. This changes the
              value of the flag it represents.
STO  0034
  :
LDD  0034
AND  #B00000010   Uses a bitwise AND operation of the contents of the accumulator with the
              operand to leave the value in position 1 unchanged but to convert every other
              bit to 0. A subsequent instruction can now compare the value of the byte with
              denary 2 to see if the flag represented by this bit position is set.
STO  0034
  :
LDD  0034
OR  #B00000100    Uses a bitwise OR operation of the contents of the accumulator with the
              operand to set the flag represented by the bit in position 2. All other bit
              positions remain unchanged.
STO  0034
```

Masking:

A mask defines which bits you want to keep, and which bits you want to clear.

Masking is the act of applying a mask to a value. This is accomplished by doing:

 Bitwise ANDing in order to extract a subset of the bits in the value

 Bitwise ORing in order to set a subset of the bits in the value

 Bitwise XORing in order to toggle a subset of the bits in the value

Below is an example of extracting a subset of the bits in the value:

Below is an example of extracting a subset of the bits in the value:

Mask: 00001111b
Value: 01010101b

Applying the mask to the value means that we want to clear the first (higher) 4 bits, and keep the last (lower) 4 bits. Thus we have extracted the lower 4 bits. The result is:

Mask: 00001111b

Value: 01010101b
Result: 00000101b

Masking is implemented using AND in above example.