COMPUTING MACHINERY I

CPSC 355

Raheela Afzal

ICT 524

raheela.afzal1@ucalgary.ca

http://pages.cpsc.ucalgary.ca/~raheela.afzal1/cpsc355/

Flags

Flag-Setting Example

```
mov     w19, 0xffffffff
mov     w20, 0x00000001
adds    w21, w19, w20
```

The result of the operation would be **0x100000000**, but the top bit is lost because it does not fit into the 32-bit destination register and so the real result is **0x00000000**. In this case, the flags will be set as follows:

Flag	Explanation
N = 0	The result is 0, which is considered positive, and so the N (negative) bit is set to 0.
Z = 1	The result is 0, so the Z (zero) bit is set to 1.
C = 1 (unsigned overflow)	We lost some data because the result did not fit into 32 bits, so the processor indicates this by setting C (carry) to 1.
V = 0 (signed overflow)	From a two's complement signed-arithmetic viewpoint, 0xffffffff really means -1, so the operation we did was really (-1) + 1 = 0. That operation clearly does not overflow, so V (overflow) is set to 0.

https://community.arm.com/processors/b/blog/posts/condition-codes-1-condition-flags-and-codes

Bitwise Operators in C

Operators	Meaning of operators
&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR
~	Bitwise complement
<<	Shift left
>>	Shift right

Bitwise Operations

Bitwise AND

```
12 = 00001100 (In Binary)
25 = 00011001 (In Binary)

Bit Operation of 12 and 25
  00001100
& 00011001

  _________
  00001000  = 8 (In decimal)
```

Bitwise OR

```
12 = 00001100 (In Binary)
25 = 00011001 (In Binary)

Bitwise OR Operation of 12 and 25
  00001100
| 00011001

  _________
  00011101  = 29 (In decimal)
```

Bitwise XOR

```
12 = 00001100 (In Binary)
25 = 00011001 (In Binary)

Bitwise XOR Operation of 12 and 25
  00001100
| 00011001

  _________
  00010101  = 21 (In decimal)
```

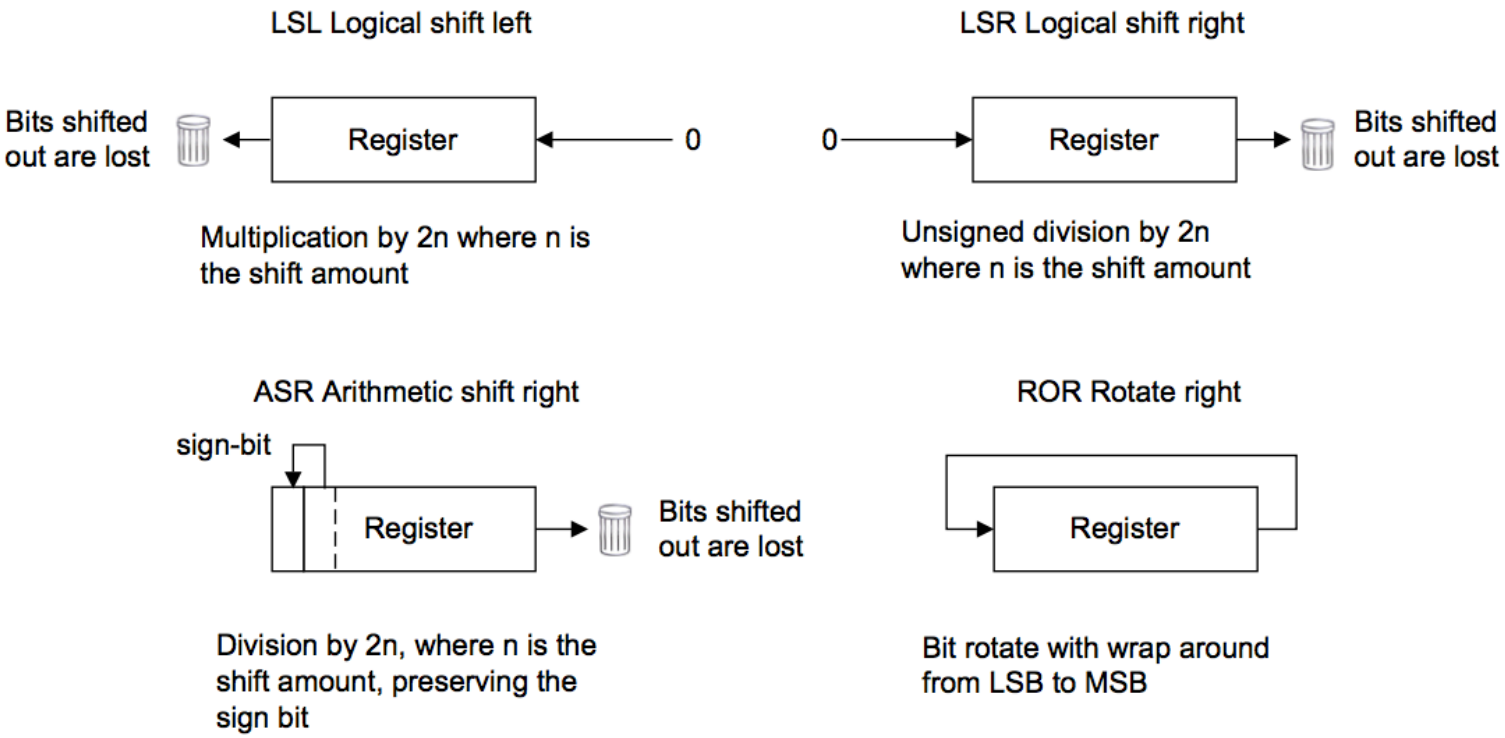
Bitwise NOT

```
35 = 00100011 (In Binary)

Bitwise complement Operation of 35
~ 00100011

  _________
  11011100  = 220 (In decimal)
```

Bitwise Shift Instructions

Instruction	Meaning
LSL Xd, Xn, #BIMM64 **LSL Wd, Wn, #BIMM32**	Logical Shift Left Xd = Xn **<<** bimm64.
LSR Xd, Xn, #BIMM64 **LSR Wd, Wn, #BIMM32**	Logical Shift Right Xd = Xn **>>** bimm64.
ASR Xd, Xn, #BIMM64 **ASR Wd, Wn, #BIMM32**	Arithmetic Shift Right Preserves the sign bit (MSB). Xd = Xn **>>** bimm64.
ROR Xd, Xn, #BIMM64 **ROR Wd, Wn, #BIMM32**	Rotate Right Bit rotate with wrap around from LSB to MSB

Shift Operations



Note: 2n represents 2^n

http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.den0024a/ch06s02s03.html

Shift Operations - Examples

Logical Shift Right

```
212    = 11010100 (In binary)
212>>2 = 00110101 (In binary)
         [Right shift by two bits]
212>>7 = 00000001 (In binary)
212>>8 = 00000000
212>>0 = 11010100 (No Shift)
```

Logical Shift Left

```
212    = 11010100 (In binary)
212<<1 = 110101000 (In binary)
         [Left shift by one bit]
212<<0 = 11010100 (Shift by 0)
212<<4 = 110101000000 (In binary)
         =3392(In decimal)
```

```
int x = 3 ;
  int n = 2 ;
  x << n ;
printf("x = %d", x);
//prints 3
```

3 = 0000 0011
2 = 0000 0010
12 = 0000 1100

To save output of shift operation:

```
x = x << n ;
printf("x = %d", x);
```

or use the <<=; operator.

```
x <<= n ;
printf("x = %d", x);
```

This prints out 12 (if x is 3 and n is 2).

Ternary Operator in C

It can only have two conditions.

The basic syntax of using the ternary operator is this:

```
(condition) ? (if_true) : (if_false)
```

Which is basically the same as:

```
if (condition){
    if_true;
}
else{
    if_false;
}
```

```
largest = ((a > b) ? a : b);
```

expression is equivalent to

```
if (a > b){
    largest = a;
}
else{
largest = b;
}
```

In C language

Type Casting: Convert from one data type to another

(type_name)expression

```
int a=2000;
int b;
b = (long int) a; //convert from 32 bit size 'int' to 64 bit size 'long int'
```

In Assembly Language

Signed extend word: Convert from 32 bit sized register to 64 bit sized register

SXTW *Xd, Wn*

Where:

Xd

is the 64-bit name of the general-purpose destination register.

Wn

is the 32-bit name of the general-purpose source register.

Bitwise Logical Instructions

Instruction	Meaning
AND Xd, Xn, #BIMM64 **AND Wd, Wn, #BIMM32**	Xd = Xn **AND** bimm64. Wd = Wn **AND** bimm32.
ANDS Xd, Xn, #BIMM64 **ANDS Wd, Wn, #BIMM32**	Xd = Xn **AND** bimm64 or Wd = Wn **AND** bimm32, setting N & Z condition flags based on the result and clearing the C & V flags.
ORR Xd, Xn, #BIMM64 **ORR Wd, Wn, #BIMM32**	Xd = Xn **OR** bimm64. Wd = Wn **OR** bimm32.
EOR Xd, Xn, #BIMM64 **EOR Wd, Wn, #BIMM32**	Xd = Xn **EOR** bimm64 Wd = Wn **EOR** bimm32.
TST Xn, #BIMM64 **TST Wn, #BIMM32**	Bitwise test : alias for **ANDS** XZR, Xn, bimm64 or **ANDS** WZR, Wn, bimm32.