The 68000

Lesson 4 – Other Instructions

The Stack

Lesson Planning

- Other Operations
  - Shifts
  - Logical Operations
    - AND
    - OR
    - EOR
- The Stack

Shifts

- When shifting a base 10 number, you are multiplying or dividing by 10.
- We do the same with ASM
- Shifting %00001000 (= 8) to the left once would give %00010000 (= 16 = 8 x 2)
- Shifting %00010000 to the right would give %00001000

Logical Shifts – Syntax (1st case)

- LSd.X #<value>, Dn
  - d may be R or L depending which way we are shifting
  - X may be B, L or W
  - n may be any number from 0 to 7
  - Value bust be between 1 and 8
- Example: LSL.B #2, D1   will shift the lower byte of D1 twice to the left

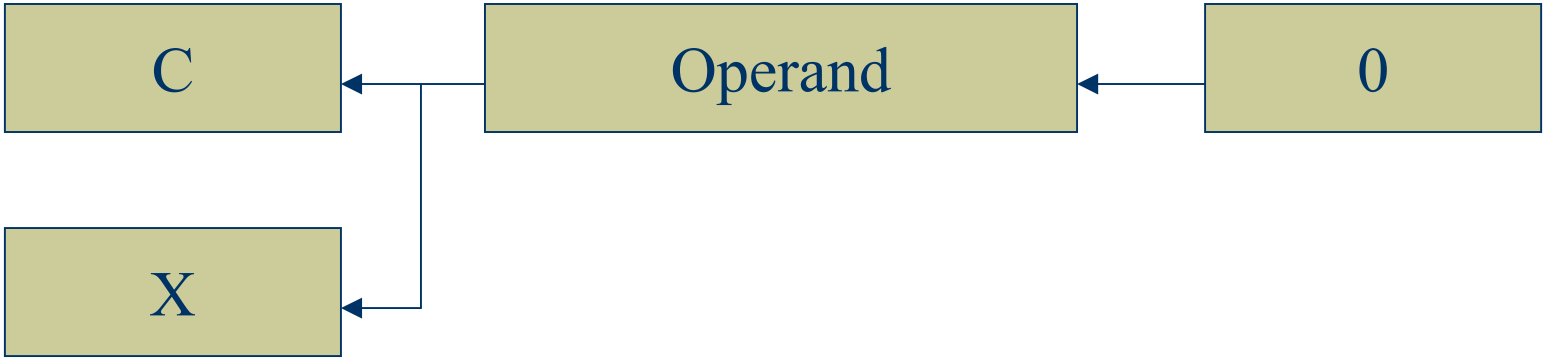
Logical Shifts – Syntax (2nd case)

- LSd.X Dx, Dy
  - d = L or R
  - X = B, L or W
  - x and y are 2 different values between 0 and 7
  - The data in Dx has to be at most 32
- Example: LSL.B D0, D1 will shift the lower byte of D1 to the left the number of times stored in D0

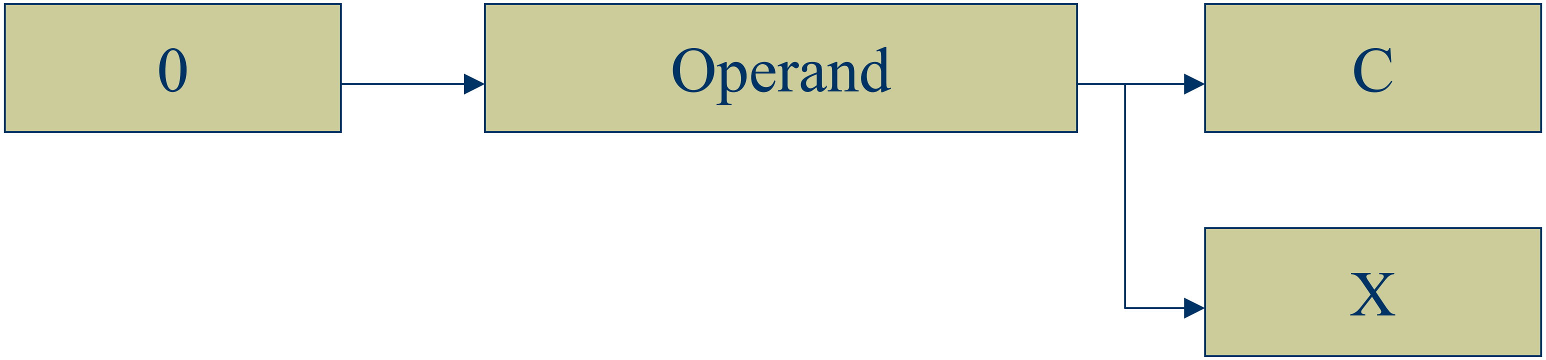
Logical Shifts – Syntax (3rd case)

- LSd <address>
  - d = L or R
  - This can only handle words
  - The shifting is only 1 bit
- Example LSL $12345678 will shift left once the value contained in the address $12345678

LSL – How does it work?



LSR – How does it work?

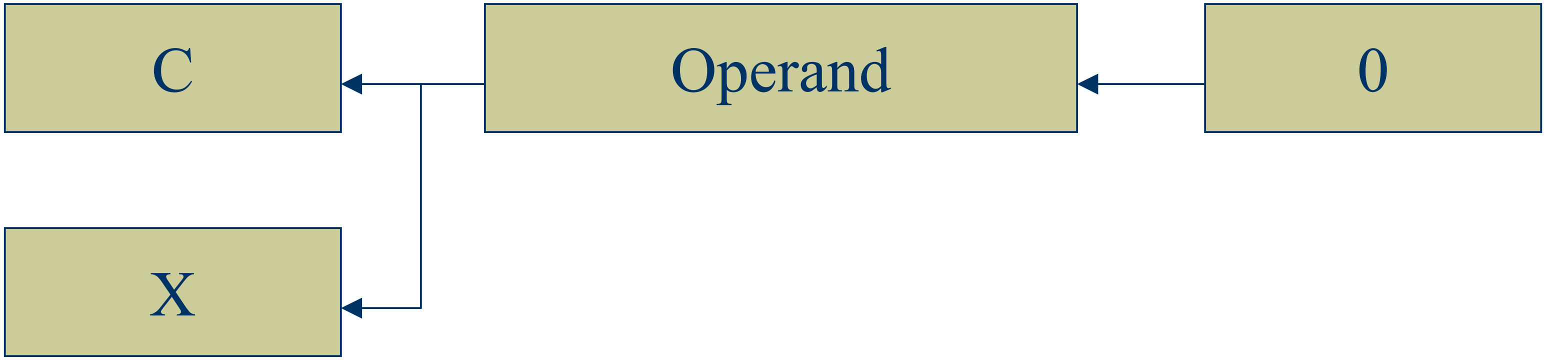


The sign strikes back

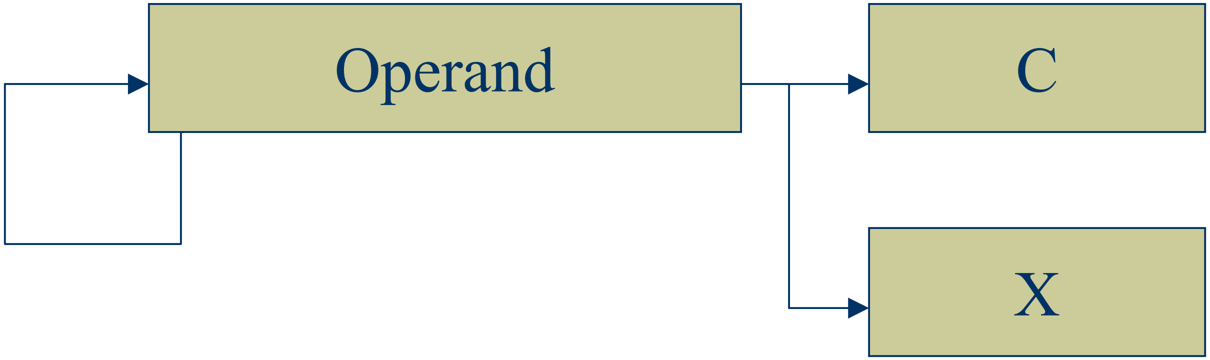
- Check this example...
- Just imagine you're shifting the byte %11111110 (= -2) to the right... It would give us %01111111 (= +127) instead of -1!!
- In those cases we shall use Arithmetic Shifts:
  - ASL and ASR
  - The syntax is the same...

ASL – How does it work?

- If the sign bit changes, the V flag will become 1, showing the overflow



ASR – How does it work?



Logical Operations - Definitions

- The or, and, eor and not instruction will perform the corresponding logical operation on each bit, one after the other
- Let’s have a look at the basic operations

Operation Tables

NOT	0	1
	1	0

OR	0	1
0	0	1
1	1	1

AND	0	1
0	0	0
1	0	1

EOR	0	1
0	0	1
1	1	0

Using AND

- AND.X Dn, <address>
- AND.X <address>, Dn
- This will do the AND operation with the 2 numbers and will store it in the destination

Using OR

- OR.X Dn, <address>
- OR.X <address>, Dn
- This will do the OR operation with the 2 numbers and will store it in the destination

Using EOR

- EOR.X Dn, <address>
- This will do the EOR operation with the 2 numbers and will store it in the destination

Immediate Operations

- You can also use ANDI, ORI and EORI, where an immediate value is given
- Mainly used for SR manipulations

The Stack

- The stack is a 16 kb area of memory (you don' t need to know where it is exactly stored)
- You have access to this memory by the stack pointer A7
- Mainly used to
  - Save variables before working with them
  - Transmit parameters to a subroutine

The Stack uses…

- It is much slower than use register
- It can be used to store or load all the registers at once
- Remember that the stack is a LIFO (Last In First Out)

Pushing data to the stack

- Imagine that A7 = $4000, D0 is $0000FFFF and you do the following instructions:

-
  ```
  move.w   D0,-(A7)
  move.l    D0,-(A7)
  sub.l       #$2,A7
  move.w   D0,(A7)
  ```

What happens...

Address	Stack	Comments
$4000	$XXXX	A7=$4000 before the instruction
$3FFE	$FFFF	A7=$3FFE after the 1st instruction
$3FFC	$FFFF	
$3FFA	$0000	A7=$3FFA after the 2nd inst.
$3FF8	$FFFF	A7=$3FF8 after the 3rd instruction

Poping data from the stack

- To get back the data we last saved, we could do:
-
  ```
  move.w   (A7)+, D0
  move.l    (A7), D1
  add.l       #$4,A7
  move.w   (A7)+, D2
  ```

What happens now...

Address	Stack	Comments
\$4000	\$XXXX	A7=\$4000 after the 4th instruction
\$3FFE	\$FFFF	A7=\$3FFE after the 3rd instruction
\$3FFC	\$FFFF	
\$3FFA	\$0000	A7=\$3FFA after the 1st inst.
\$3FF8	\$FFFF	A7=\$3FF8 before the 1st instruction

The result will be...

- We’ll have
  - D0 = $XXXXFFFF
  - D1 = $0000FFFF
  - D2 = $XXXXFFFF