Logic And Shift Instructions

ARM Logic Instructions

C/C++/Java operator

•	AND	dst = src1 AND src2	src1 & src2
•	EOR	dst = src1 EOR src2	src1 ^ src2
•	ORR	dst = src1 OR src2	src1 | src2
•	MVN	dst = NOT src2	~src2
•	ORN	dst = src1 NOR src2	~(src1 | src2)
•	BIC	dst = src1 AND NOT src2	(src1 & ~src2)

- dst: register
- src1: register
- src2: register OR constant

Examples

```
ORR    R0, R1, R2         ; R0 = R1 | R2
AND    R0, R0, #0x0F      ; R0 = R0 & 0x0F
EORS   R1, R3, R0         ; R1 = R3 ^ R0 + set condition code flags
```

Logic And Shift Instructions

AND

- dst = src1 & src2
- each bit in dst is the AND of the corresponding bits in src1 and src2 (see truth table)
- can be used to clear selected bits

src1	src2	AND
0	0	0
0	1	0
1	0	0
1	1	1

truth table

```
MOV     R0, #0xAA
AND     R0, R0, #0x0F
```

	0xAA	1010 1010
&	0x0F	0000 1111
	0x0A	0000 1010

clears most significant bits

- if src2 used as a mask, clears bit if corresponding bit in mask is 0

Logic And Shift Instructions

OR

- dst = src1 | src2
- each bit in dst is the OR of the corresponding bits in src1 and src2 (see truth table)
- can be used to set selected bits

src1	src2	OR
0	0	0
0	1	1
1	0	1
1	1	1

truth table

```
MOV     R0, #0x0A
ORR     R0, R0, #0xF0
```

	0x0A	0000 1010
|	0xF0	1111 0000
	0xFA	1111 1010

sets most significant bits

- if src2 used as a mask, sets bit if corresponding bit in mask is 1

Logic And Shift Instructions

EOR / XOR

src1	src2	EOR
0	0	0
0	1	1
1	0	1
1	1	0

truth table

- dst = src1 ^ src2
- each bit in dst is the EOR of the corresponding bits in src1 and src2 (see truth table)
- can be used to invert selected bits

```
MOV     R0, #0x0A
EOR     R0, R0, #0x0F
```

	0x0A	0000 1010
^	0x0F	0000 1111
	0x05	0000 0101

inverts least significant bits

- if src2 used as a mask, inverts bit if corresponding bit in mask is 1

Logic And Shift Instructions

MVN (Move NOT)

- dst = ~src2
- each bit in dst is the inverse (~ or NOT) of the corresponding bit in src2 (see truth table)
- can be used to invert ALL bits

src2	NOT
0	1
1	0

truth table

```
MOV     R0, #0x0A
MVN     R0, R0
```

~	0x0000000A	... 0000 1010
	0xFFFFFFF5	... 1111 0101

inverts ALL bits

Logic And Shift Instructions

ORN (NOR)

- dst = ~(src1 | src2)
- each bit in dst is the NOR of the corresponding bits in src1 and src2 (see truth table)

src1	src2	NOR
0	0	1
0	1	0
1	0	0
1	1	0

truth table

```
MOV     R0, #0x0A
ORN     R0, R0, #0x0F
```

	0x0000000A	… 0000 1010
NOR	0x0000000F	… 0000 1111
	0xFFFFFFF0	… 1111 0000

Logic And Shift Instructions

BIC (bit clear)

- dst = src1 & ~src2
- each bit in dst is set to src1 & ~src2 using the corresponding bits in src1 and src2 (see truth table)
- can be used to clear selected bits

src1	src2	BIC
0	0	0
0	1	0
1	0	1
1	1	0

truth table

```
MOV     R0, #0xAA
BIC     R0, R0, #0xF0
```

	0xAA	1010	1010
BIC	0xF0	1111	0000
	0x0A	0000	1010

clear selected bits

- if src2 used as a mask, clears bit if corresponding bit in mask is 1

How to Clear Bits

- write ARM instructions to clear bits 3 and 4 of R1 (LS bit is bit 0)

```
LDR     R1, =0x12345678     ; load test value
LDR     R2, =0xFFFFFFE7     ; AND mask to clear bits
AND     R1, R1, R2          ; R1 = 0x12345660
```

	0x…78	0111 1000
&	0x…E7	1110 0111
	0x…60	0110 0000

clear bits 3 and 4

- alternatively, the BIC (Bit Clear) instruction can be used with a mask of 1's in the bit positions that need to be cleared

```
LDR     R1, =0x12345678     ; load test value
LDR     R2, =0x00000018     ; AND mask to clear bits 3 and 4
BIC     R1, R1, R2          ; R1 = 0x12345660
```

- in this case, can use an immediate mask saving one instruction

```
LDR     R1, =0x12345678     ; load test value
BIC     R1, R1, #0x18       ; R1 = 0x12345660
```

How to Invert Bits

- write ARM instructions to invert bits 2 .. 5 of R1 (LS bit is bit 0)
- EOR mask = 0x3C (invert bit if corresponding bit in mask is 1)

```
LDR     R1, =0x12345678     ; load test value
LDR     R2, =0x0000003C     ; EOR mask to invert bits 2 .. 5
EOR     R1, R1, R2          ; R1 = 0x12345644
```

	0x…78	0111 1000
^	0x…3C	0011 1100
	0x…44	0100 0100

inverts bits 2, 3, 4 and 5

- in this case, can use an immediate mask saving one instruction

```
LDR     R1, =0x12345678     ; load test value
EOR     R1, R1, #0x3C       ; R1 = 0x12345644
```

LOGIC AND SHIFT INSTRUCTIONS

ARM Shift and Rotate

C/C++/Java operator

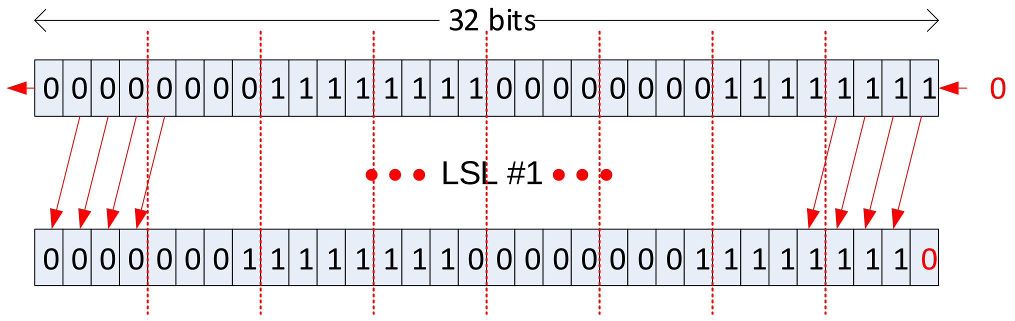
• Logical Shift Left (LSL)	a << n	// logical shift left n places
• Logical Shift Right (LSR)	a >> n	// logical shift right n places
• Arithmetic Shift Right (ASR)		
• Rotate Right (ROR)		
• Rotate Right with eXtend (RRX)		

- NB: these are NOT instructions in the same sense as ADD, SUB or ORR

Logic And Shift Instructions

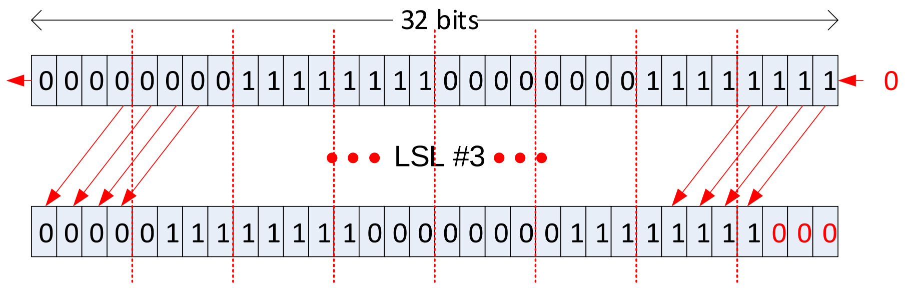
Logical Shift Left (LSL)

- LSL one place (LSL #1)
- 0 shifted into LSB, MSB discarded



0x00FF00FF => 0x01FE01FE

- LSL 3 places (LSL #3)



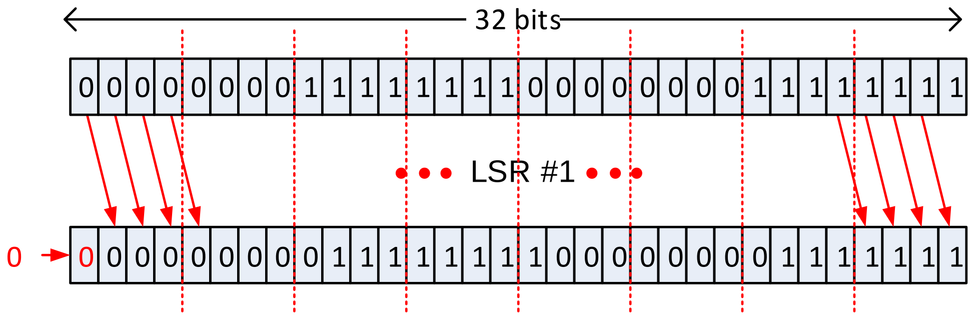
0x00FF00FF => 0x07F807F8

- can LSL 0 to 31 places

Logic And Shift Instructions

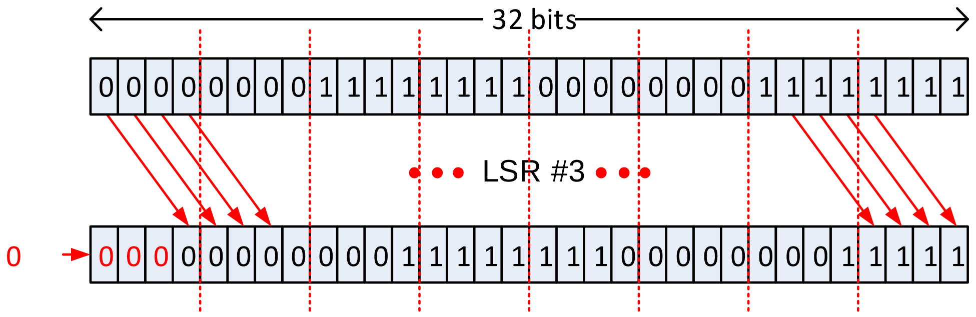
Logical Shift Right (LSR)

- LSR one place (LSR #1)
- 0 shifted into MSB, LSB discarded



0x00FF00FF => 0x007F807F

- LSR 3 places (LSR #3)

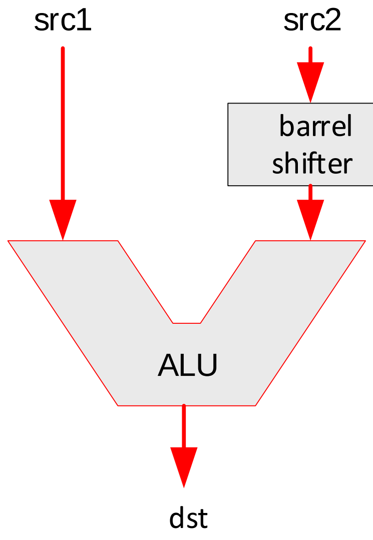


0x00FF00FF => 0x001FE01F

- can LSR 0 to 31 places

ARM shift instructions

- ARM has NO dedicated shift/rotate instructions
- instead, ALL instructions can optionally shift/rotate the src2 operand before it is used as input for the ALU operation (ADD, SUB, …)



src2 to ALU can be:

1) register with an optional shift/rotate
   - shift/rotate by constant number of places OR
   - by the number places specified in a register

2) 8 bit immediate value rotated right by an even number of places

- very ARM specific – unlike other CPUs

Shift using MOV

- ARM assembly language syntax to LSL src2 one place before MOV operation

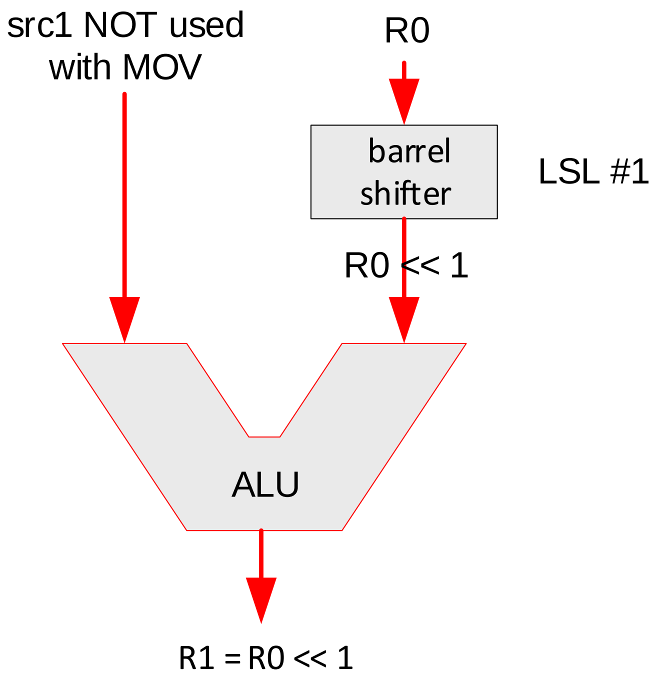
```
MOV    R1, R0, LSL #1      ; R1 = R0 << 1
```

- logical shift left 5 places

```
MOV    R1, R0, LSL #5      ; R1 = R0 << 5
```

- if NO shift specified, the default is LSL #0

```
MOV    R1, R0, LSL #0      ; R1 = R0 (NO shift)
```



Logic And Shift Instructions

Logical Shift Left

- LSL one place is the same as multiplying by 2 (if NO carry/overflow)

```
LDR     R0, =0xFF                ; R0 = 0x00FF (255)
MOV     R1, R0, LSL #1           ; R1 = 0x01FE (510)
```

- LSL n places is the same as multiplying by 2^n

```
LDR     R0, =0xFF                ; R0 = 0x00FF (255)
MOV     R1, R0, LSL #4           ; R1 = 0x0FF0 (255 x 2^4 = 255 x 16 =  4080)
```

- works for signed and unsigned integers

```
LDR     R0, =0xFFFFFFFF          ; R0 = 0xFFFFFFFF (-1)
MOV     R1, R0, LSL #2           ; R1 = 0xFFFFFFFC (-4) = R0 x 4
```

Logic And Shift Instructions

Logical Shift Right …

- LSR one place is the same as integer division by 2 (if NO carry/overflow)

```
LDR     R0, =0xFF                ; R0 = 0xFF (255)
MOV     R1, R0, LSR #1           ; R1 = 0x7F (127)
```

- LSR n places is the same as integer division by 2^n

```
LDR     R0, =0xFF                ; R0 = 0xFF (255)
MOV     R1, R0, LSR #4           ; R1 = 0x0F (255 / 2^4 = 255 / 16 = 15)
```

- works for unsigned integers

- for signed integers use arithmetic shift right (ASR) which will be covered later

Logic And Shift Instructions

Shift …

- can shift left or right by the number places specified in a register

```
MOV    R1, R0, LSL R2      ; R1 = R0 << R2
```

LSL R2 places (LS 5 bits)

- R2 can be a variable rather than a constant

- if **MOVS** is used instead of MOV, the last bit shifted out (left or right) is stored in the CARRY flag

```
MOV    R0, #0x55           ; R0 = 0x55                    0x55    0101 0101
MOVS   R1, R0, LSR #3      ; R1 = R0 >> 3        >> 3     0x0A    0000 1010
```

CARRY = 1

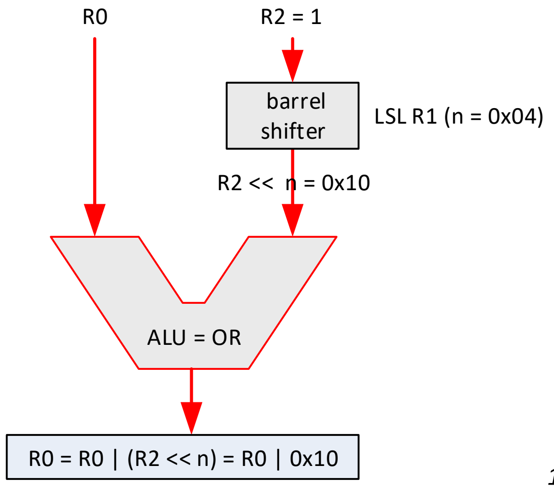
Example Shift Operations

- shifts can be followed by any operation ALU operation
- what do the following instructions do?

```
ADD     R0, R1, R1, LSL #3     ; R0 = R1 + R1 x 8 = R1 x 9
RSB     R0, R5, R5, LSL #3     ; R0 = R5 x 8 - R5 = R5 x 7
SUB     R0, R9, R8, LSR #4     ; R0 = R9 - R8/16
```

- write ARM instructions to set bit n of R0 where n is in R1 (n in range 0 .. 31)

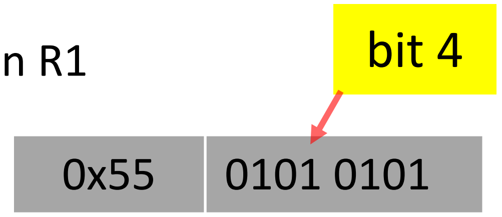
```
MOV     R1, #4                 ; R1 = 4
MOV     R2, #0x01              ; R2 = 1
ORR     R0, R0, R2, LSL R1     ; R0 = R0 | R2 << n
```



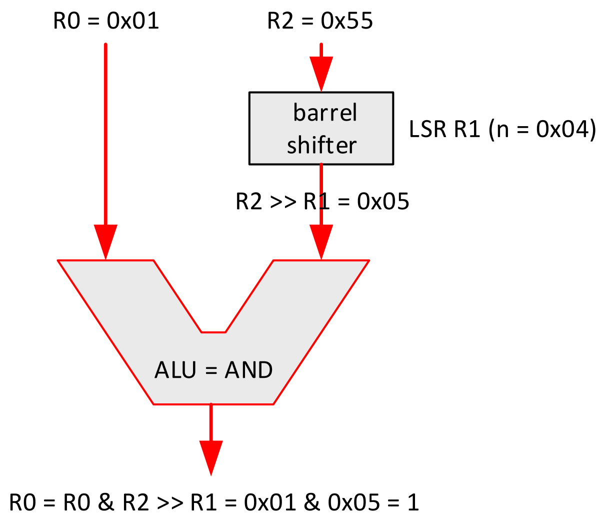
LOGIC AND SHIFT INSTRUCTIONS

Example Shift Operations …

- write ARM instructions to set R0 = nth bit of R2 where n is in R1
- example: if R2 = 0x55 and n is 4 then R0 = 1



```
MOV     R0, #1              ; R0 = 1
AND     R0, R0, R2, LSR R1  ; R0 = R0 & R2 >> R1
```



Logic And Shift Instructions

Remember

- prepare for Mid-Term Test during Study Week
- ALL students Thurs 1st Nov @ 9am in Goldsmith Hall (instead of Tutorial)