

---

**ELEC2041**  
**Microprocessors and Interfacing**

**Lecture 10: C/Assembler Logical and Shift – II  
& Multiplication**

<http://webct.edtec.unsw.edu.au/>

**March 2006**

**Saeid Nooshabadi**  
**saeid@unsw.edu.au**

---

**Overview**

- **Shift Operations**
  - Field Insertion
- **Multiplication Operations**
  - Multiplication
  - Long Multiplication
  - Multiplication and accumulation
  - Signed and unsigned multiplications

---

**Review: ARM Instructions So far**

add  
sub  
mov  
and  
bic  
orr  
eor

**Data Processing Instructions with  
shift and rotate**

lsl, lsr, asr, ror

---

**Review: Masking via Logical AND**

- **AND:** Note that anding a bit with 0 produces a 0 at the output while anding a bit with 1 produces the original bit.
- This can be used to create a **mask**.

- Example:

1011 0110 1010 0100 0011 1101 1001 1010  
**Mask:** 0000 0000 0000 0000 0000 0000 **1111 1111**

- The result of anding these two is:

0000 0000 0000 0000 0000 0000 **1001 1010**

## Review: Masking via Logical BIC

◦ **BIC (AND NOT):**Note that bicing a bit with 1 produces a 0 at the output while bicing a bit with 0 produces the original bit.

◦ This can be used to create a **mask**.

• Example:

1011 0110 1010 0100 0011 1101 1001 1010

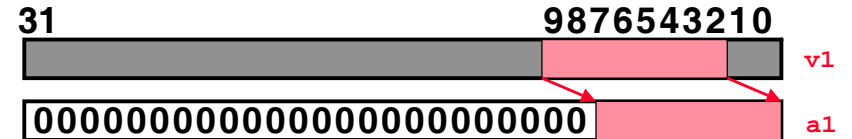
**Mask:** 0000 0000 0000 0000 0000 0000 1111 1111

• The result of bicing these two is:

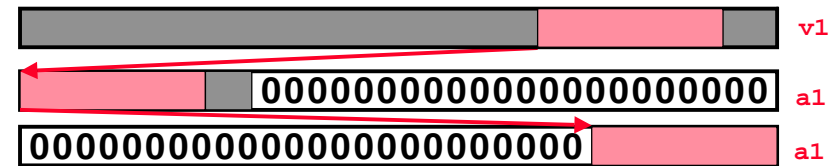
1011 0110 1010 0100 0011 1101 0000 0000

## Extracting a field of bits (#1/2)

◦ Extract bit field from bit 9 (left bit no.) to bit 2 (size=8 bits) of register v1, place in rightmost part of register a1

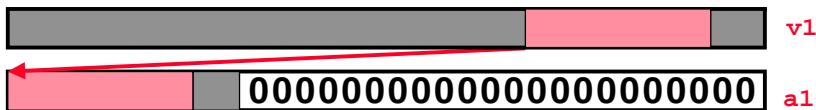


◦ Shift field as far left as possible (9 → 31) and then as far right as possible (31 → 7)

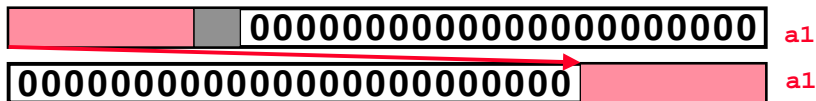


## Extracting a field of bits (#2/2)

`mov a1, v1, lsl #22 ; 8 bits to left end (31-9)`

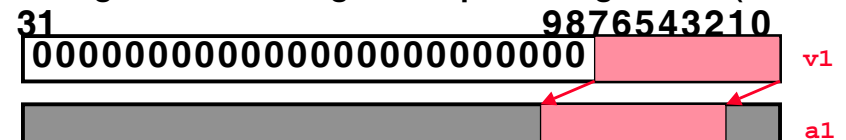


`mov a1, a1, lsr #24 ; 8 bits to right end (7-0)`

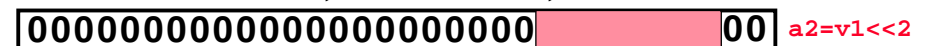


## Inserting a field of bits

◦ Insert bit field into bit 9 (left bit no.) to bit 2 (size=8 bits) of register a1 from rightmost part of register v1 (rest is 0)



◦ Shift left field 2 bits, Mask out field, OR in field



```

◦ mov a2, v1, lsl #2 ; field left 2
  bic a1, a1, #0x3FC ; mask out 9-2
                                ; 0x03FC = 0011 1111 1100
  orr a1, a1, a2      ; OR in field

```

◦ **bic** stands for 'bit clear, where '1' in the second operand clears the corresponding bit in the first

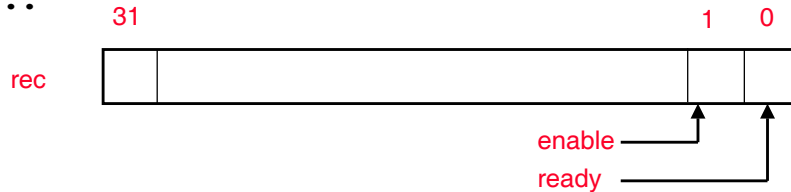
## Bit manipulation in C (#1/2)

### ◦ Convert C code to ARM ASM

### ◦ Bit Fields in C (Word as 32 bits vs int/unsigned!)

```
struct {
    unsigned int ready: 1; /* bit 0 */
    unsigned int enable: 1; /* bit 1 */
} rec;
rec.enable = 1;
rec.ready = 0;
printf("%d %d", rec.enable, rec.ready);
...

```



Brian Kernighan & Dennis Ritchie:  
The C Programming Language, 2nd Ed., PP 150

ELEC2041 lec10-logical-II&mul.9

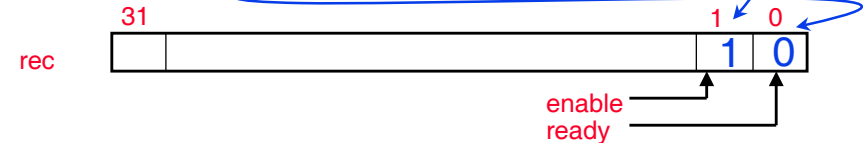
Saeid Nooshabadi

## Bit manipulation in C (#2/2)

```
struct {
    unsigned int ready: 1; /* bit 0 */
    unsigned int enable: 1; /* bit 1 */
} rec; /* v1 */
rec.enable = 1;
rec.ready = 0;
printf("%d %d", rec.enable, rec.ready);

```

```
orr v1,v1, #0x2 ;1 in bit 1
bic v1,v1 #1 ;0 in bit 0,
```



```
ldr a1, =LCO ;printf format
mov a2, v1, lsr #1 ;just bit 1
and a2, a2, 0x0001 ;mask down to 1
and a3, v1, 0x0001 ;just bit 0
bl printf ;call printf
```

ELEC2041 lec10-logical-II&mul.10

Saeid Nooshabadi

## Multiply by Power of 2 via Shift Left (#1/3)

### ◦ In decimal:

- Multiplying by 10 is same as shifting left by 1:
  - $714_{10} \times 10_{10} = 7140_{10}$
  - $56_{10} \times 10_{10} = 560_{10}$
- Multiplying by 100 is same as shifting left by 2:
  - $714_{10} \times 100_{10} = 71400_{10}$
  - $56_{10} \times 100_{10} = 5600_{10}$
- Multiplying by  $10^n$  is same as shifting left by n

ELEC2041 lec10-logical-II&mul.11

Saeid Nooshabadi

## Multiply by Power of 2 via Shift Left (#2/3)

### ◦ In binary:

- Multiplying by 2 is same as shifting left by 1:
  - $11_2 \times 10_2 = 110_2$
  - $1010_2 \times 10_2 = 10100_2$
- Multiplying by 4 is same as shifting left by 2:
  - $11_2 \times 100_2 = 1100_2$
  - $1010_2 \times 100_2 = 101000_2$
- Multiplying by  $2^n$  is same as shifting left by n

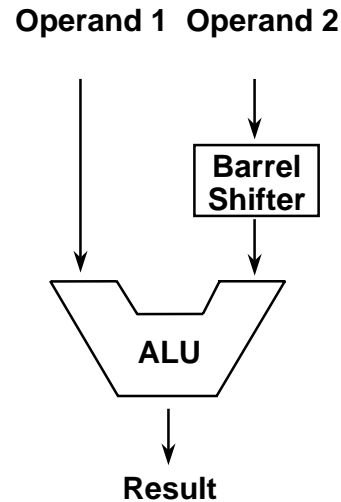
ELEC2041 lec10-logical-II&mul.12

Saeid Nooshabadi

## Multiply by Power of 2 via Shift Left (#3/3)

- Since shifting is so much faster than multiplication (you can imagine how complicated multiplication is), a good compiler usually notices when C code multiplies by a power of 2 and compiles it to a shift instruction:

```
a *= 8; (in C)
would compile to:
mov a0,a0,ls1 #3 (in ARM)
```



## Shift, Add and Subtract for Multiplication

### Add and Subtract Examples:

```
f = 5*g /* f = (4+1) x g */ (in C)
add v1,v2,v2,ls1 #2 ; v1 = v2 + v2 *4 (in ARM)
```

```
f = 105 *g /* f = (15 x 7) x g */ (in C)
/* f = (16-1) x (8-1) x g */
rsb v1,v2,v2,ls1 #4 ; v1 = -v2 + v2 *16 (in ARM)
; f = (16-1)* g
```

```
rsb v1,v1,v1,ls1 #3 ; v1 = -v1 + v1 *8 (in ARM)
; f = (8-1)* f
```

## Shift, Add and Subtract for Division

- ARM does not have division.
- Division A/B produces a quotient and a remainder.
- It should be done via sequence of subtraction and shifting (See Experiment 3)
- For B in A/B a constant value (eg 10) simpler technique via Shift, Add and Subtract is available (Will be discussed later)

## Shift Right Arithmetic; Divide by 2???

- Shifting left by n is same as Multiplying by  $2^n$



- Shifting right by n bits would seem to be the same as dividing by  $2^n$
- Problem is signed integers
  - Zero fill is wrong for negative numbers
- Shift Right Arithmetic (asr); sign extends (replicates sign bit);
  - 1111 1111 1111 1000 = -8
  - 1111 1111 1111 1100 = -4
  - 1111 1111 1111 1110 = -2
  - 1111 1111 1111 1111 = -1

## Is asr really divide by 2?

◦ Divide +5 by 4 via asr 2; result should be 1

0000 0000 0000 0000 0000 0000 0000 0101

0000 0000 0000 0000 0000 0000 0000 0001

◦ = +1, so **does** work

◦ Divide -5 by 4 via asr 2; result should be -1

1111 1111 1111 1111 1111 1111 1111 1011

1111 1111 1111 1111 1111 1111 1111 1110

◦ = -2, not -1; Off by 1, so **doesn't always** work

◦ Rounds to  $-\infty$

## MULTIPLY (unsigned): Terms, Example

◦ Paper and pencil example (unsigned):

Multiplicand	1000	
Multiplier	1001	
	1000	+
	0000	+
	0000	+
	1000	+
Product	01001000	

• m bits x n bits = m+n bit product

• 32-bit value x 32-bit value = 64-bit value

## MULTIPLY (signed)? #(1/3)

◦ Paper and pencil example

Multiplicand	1000	-8 x
Multiplier	1001	-7 x
	1000	+
	0000	+
	0000	+
	1000	+
Product	01001000	72 ≠ 56

◦ Solution 1:

◦ 2's complement range for 4 bit number is  $-8 \leq a \leq 7 \rightarrow -56 \leq a \times b \leq 64$ .

◦ Can build a big lookup table (16 x 16 = 256) to map from binary bit patterns to multiplication results.

-8 x -8 = 01000000 ← → 64

-8 x -7 = 01001000 ← → 56

|

-2 x -2 = 11000100 ← → 04

-1 x -1 = 11100001 ← → 01

|

+7 x +7 = 00110001 ← → 49

## MULTIPLY (signed)? #(2/3)

◦ Solution 2:

Multiplicand	1001	-7 x
Multiplier	1000	-8
	00000000	+
	00000000	+
	000000	+
	11001	-
Product	00111000	56 = 56

Multiplicand	1001	-7 x
Multiplier	1000	-8
	00000000	+
	00000000	+
	000000	+
	00111	+
Product	00111000	56 = 56

## MULTIPLY (signed)? #3/3

### ◦ Solution 2 (Another example):

```

Multiplicand  1001      -7 X
Multiplier    1001      -7
-----
11111001    +
00000000    +
00000000    +
11001       -
-----
Product      00110001    49 = 49
    
```

```

Multiplicand  1001      -7 X
Multiplier    1000      -7
-----
11111001    +
00000000    +
00000000    +
00111       +
-----
Product      00110001    49 = 49
    
```

ELEC2041 lec10-logical-II&mul.21

Saeid Nooshabadi

## Multiplication Instructions

### ◦ The Basic ARM provides two multiplication instructions.

#### ◦ Multiply

• `mul Rd, Rm, Rs ; Rd = Rm * Rs`

#### ◦ Multiply Accumulate - does addition for free

• `mula Rd, Rm, Rs, Rn ; Rd = (Rm * Rs) + Rn`

$$\sum_{i=0}^N a_i \times b_i$$

#### ◦ (Lower precision multiply instructions simply throws top 32bits away)

#### ◦ Restrictions on use:

- Rd and Rm cannot be the same register
  - Can be avoided by swapping Rm and Rs around. This works because multiplication is commutative.
- Cannot use PC.

These will be picked up by the assembler if overlooked.

#### ◦ Operands can be considered signed or unsigned

- Up to user to interpret correctly.

ELEC2041 lec10-logical-II&mul.22

Saeid Nooshabadi

## Multiplication Example

### ◦ Example:

- in C: `a = b * c;`
  - in ARM:
    - let b be v1; let c be v2; and let a be v3 (It may be up to 64 bits)
- ```
mul v3, v2, v1 ;a = b*c
                ; lower half of product into
                ; v3. Upper half is thrown up
```

### ◦ Note: Often, we only care about the lower half of the product.

ELEC2041 lec10-logical-II&mul.23

Saeid Nooshabadi

## Multiplication and Accumulate Example

### ◦ One example of use of `mula` is for string to number conversion: eg

Convert string="123" to value=123

```

value = 0
loop = 0
len = length of string
Rd = value
while loop <> len
c = extract(string, len - loop,1)
Rm = 10 ^ loop
Rs = ASC(c) - ASC ('0')
mula Rd, Rm, Rs, Rd
loop = loop + 1
endwhile
    
```

ELEC2041 lec10-logical-II&mul.24

Saeid Nooshabadi

## Multiply-Long and Multiply-Accumulate Long

- Instructions are
  - MULL which gives  $RdHi, RdLo := Rm * Rs$
  - MLAL which gives  $RdHi, RdLo := (Rm * Rs) + RdHi, RdLo$
- However the full 64 bit of the result now matter (lower precision multiply instructions simply throws top 32 bits away)
  - Need to specify whether operands are signed or unsigned
- Therefore syntax of new instructions are:
  - `umull RdLo, RdHi, Rm, Rs ; RdHi, RdLo := Rm * Rs`
  - `umlal RdLo, RdHi, Rm, Rs ; RdHi, RdLo := (Rm * Rs) + RdHi, RdLo`
  - `smull RdLo, RdHi, Rm, Rs ; RdHi, RdLo := Rm * Rs (Signed)`
  - `smlal RdLo, RdHi, Rm, Rs ; RdHi, RdLo := (Rm * Rs) + RdHi, RdLo (Signed)`
- Not generated by some compilers. (Needs Hand coding)

$$\sum_{i=0}^N a_i \times b_i$$

ELEC2041 lec10-logical-II&mul.25

Saeid Nooshabadi

## Division

- No Division Instruction in ARM
- Division has to be done in software through a sequence of shift/ subtract / add instruction.
  - General A/B implementation (See Experiment 3)
  - For B in A/B a constant value (eg 10) simpler technique via Shift, Add and Subtract is available (Will be discussed later)

ELEC2041 lec10-logical-II&mul.26

Saeid Nooshabadi

## Quiz

1. Specify instructions which will implement the following:
  - a)  $a1 = 16$
  - b)  $a2 = a1 * 4$
  - c)  $a1 = a2 / 16$  (r1 signed 2's comp.)
  - d)  $a2 = a3 * 7$
2. What will the following instructions do?
  - a) `add a1, a2, a2, lsl #2`
  - b) `rsb a3, a2, #0`
3. What does the following instruction sequence do?
  - `add a1, a2, a2, lsl #1`
  - `sub a1, a1, a2, lsl #4`
  - `add a1, a1, a2, lsl #7`

ELEC2041 lec10-logical-II&mul.27

Saeid Nooshabadi

## Quiz Solution (#1/2)

1. Specify instructions which will implement the following:
  - a)  $a1 = 16$  `mov a1, #16`
  - b)  $a2 = a1 * 4$  `mov a2, a1, lsl #2`
  - c)  $a1 = a2 / 16$  (r1 signed 2's comp.) `mov a1, a2, asr #4`
  - d)  $a2 = a3 * 7$  `rsb a2, a3, a3, lsl #3`  
`a2 = a3 * (8-1)`

whereas `sub a2, a3, a3, lsl #3` would give  $a3 * -7$
2. What will the following instructions do?
  - a) `add a1, a2, a2, lsl #2`  
`a1 = a2 + (a2 * 4) i.e a1 := a2 * 5`
  - b) `rsb a3, a2, #0`
    - `r2 = 0 - r1 i.e r2 := -r1`

ELEC2041 lec10-logical-II&mul.28

Saeid Nooshabadi

## Quiz Solution (#2/2)

---

### 3. What does the following instruction sequence do?

```
add a1, a2, a2, lsl #1
sub a1, a1, a2, lsl #4
add a1, a1, a2, lsl #7
```

$$a1 = a2 + (a2 * 2) = a2 * 3$$

$$a1 = a1 - (a2 * 16) = (a2 * 3) - (a2 * 16) = a2 * -13$$

$$\begin{aligned} a1 &= a1 + (a2 * 128) = (a2 * -13) + (a2 * 128) \\ &= r1 * 115 \\ \text{i.e. } a1 &= a2 * 115 \end{aligned}$$

## ELEC2041 Reading Materials (Week #4)

---

- Week #4: Steve Furber: ARM System On-Chip; 2nd Ed, Addison-Wesley, 2000, ISBN: 0-201-67519-6. We use [chapters 3 and 5](#)
- ARM Architecture Reference Manual –On CD ROM

## “And in Conclusion...”

---

### ◦ New Instructions:

```
mul
mla
umull
umlal
smull
smlal
```