

COMSM1302

Overview of Computer Architecture

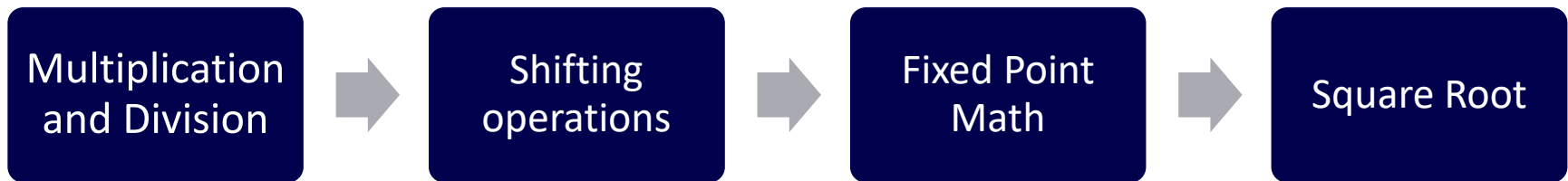
Lecture 13

Advanced Math Operations

In the previous lecture

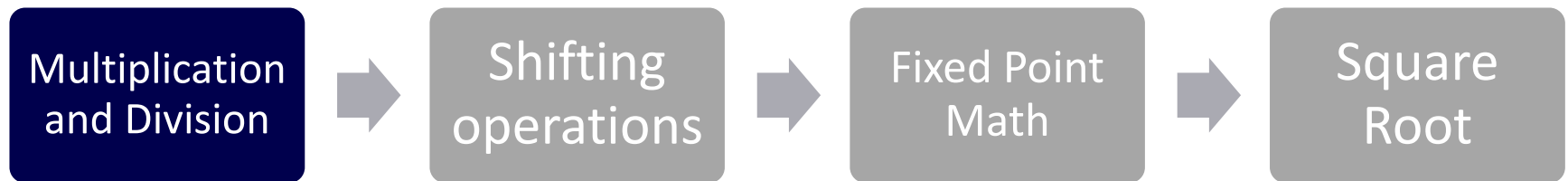
- General introduction to ARM architecture.
- Conditional code flags and conditional execution.
- Data processing instructions
 - Arithmetic and logical operations.
 - Comparisons (no results - just set condition codes)
 - Data movement between registers.
- Branching instructions.

In this lecture



- At the end of this lecture:
 - Solve problems that require multiplication and division.
 - Use shifting operations to do efficient calculations.
 - Use fixed point math to do accurate calculations.

Advance Math Operations



🔥 Multiplication Instructions

- Multiply

- MUL{cond}{S} Rd, Rm, Rs

- $Rd = Rm * Rs$

32 32 \swarrow $32 * 32$

Usually, we need a 64 bits register to store the multiplication of two 32 bits value. Rd only stores the 32 LSB.

- Multiply Accumulate

- Does addition for free

- MLA{cond}{S} Rd, Rm, Rs, Rn

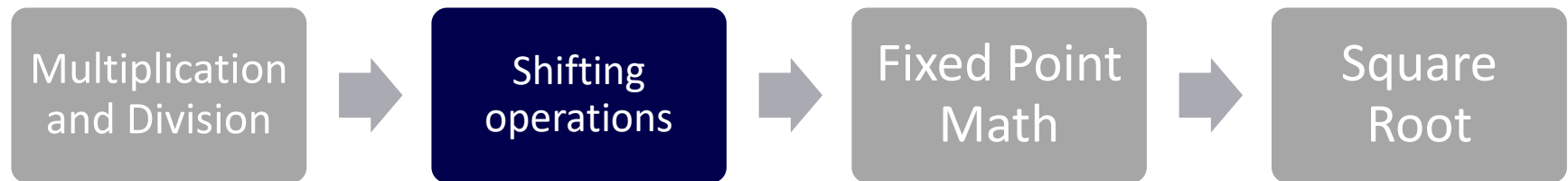
- $Rd = (Rm * Rs) + Rn$

Division Instructions

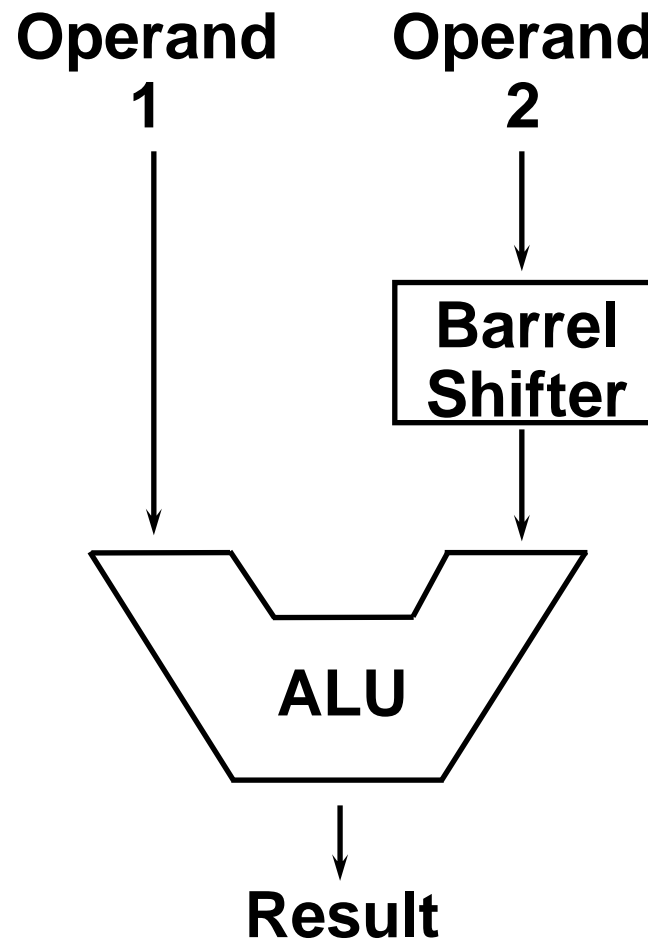
- **Unsigned integer** division
 - UDIV{cond} Rd, Rm, Rs ; Rd = Rm / Rs

$9 / 2 = 4$
- **Signed integer** division
 - SDIV{cond} Rd, Rm, Rs ; Rd = Rm / Rs

Advance Math Operations



The Barrel Shifter



very quick, no need for a clock phase, can be done during any data processing instruction.

🔥 Barrel Shifter – Logical Left Shift

- Shifts left by the specified amount (multiplies by powers of two) e.g.
 - LSL{cond}{S} Rd, Rm, Rs
 - LSL{cond}{S} Rd, Rm, #sh
 - MOV r1, r0, LSL #1
 - LSL r1, r0, #1

Logical Shift Left (LSL)

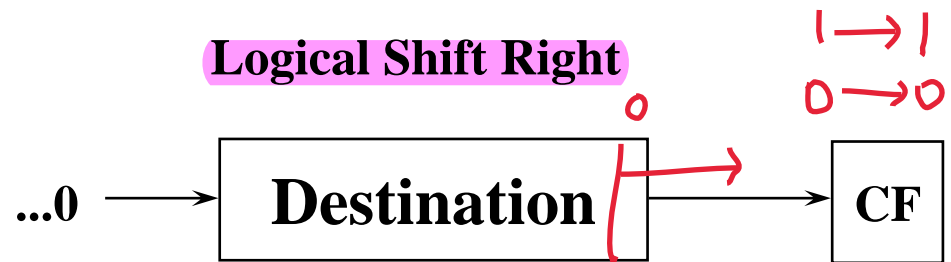


🔥 Barrel Shifter – Logical Right Shift

- Shifts right by the specified amount (divides by powers of two) e.g.

$\div 2$

- LSR{cond}{S} Rd, Rm, Rs
- LSR{cond}{S} Rd, Rm, #sh
- MOV r1, r0, LSR #1
- LSR r1, r0, #1



🔥 Barrel Shifter – Arithmetic Right Shift

- Shifts right (divides by powers of two) and preserves the sign bit, for 2's complement operations.

– ASR{cond}{S} Rd, Rm, Rs

– ASR{cond}{S} Rd, Rm, #sh

– MOV r1, r0, ASR #1

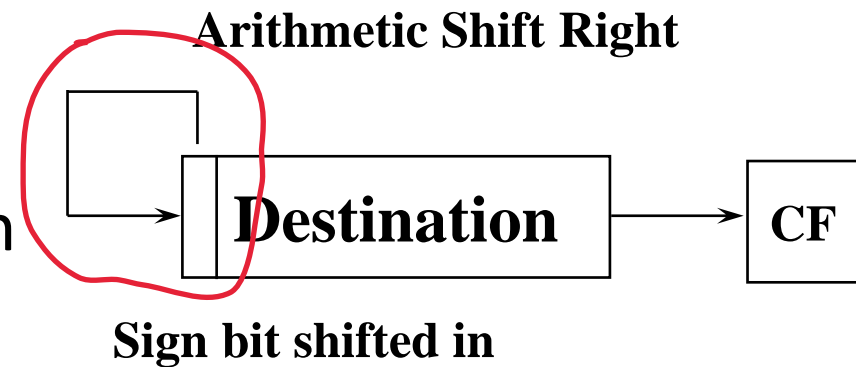
– ASR r1, r0, #1

$r_0 = -6$

$r_1 = -3$

$1010 \rightarrow 1101$

$-6 \quad -3$



Barrel Shifter – Shift Operations

- Logical Shift Left: LSL
- Logical Shift Right: LSR
- Arithmetic Shift Right: ASR

- Why do not we have Arithmetic Shift Left (ARL) ?

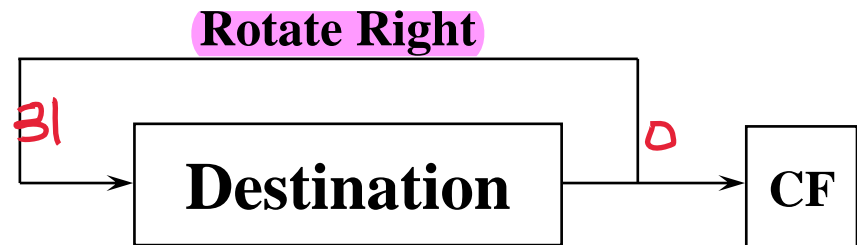
Because there will be a carryout in the left

🔥 Barrel Shifter - Rotations

Rotate Right (ROR)

- **Similar to an ASR** but the bits wrap around as they leave the LSB and appear as the MSB.
 - `ROR{cond}{S} Rd, Rm, Rs`
 - `ROR{cond}{S} Rd, Rm, #sh`
 - `MOV r1, r0, ROR #1`
 - `ROR r1, r0, #1`

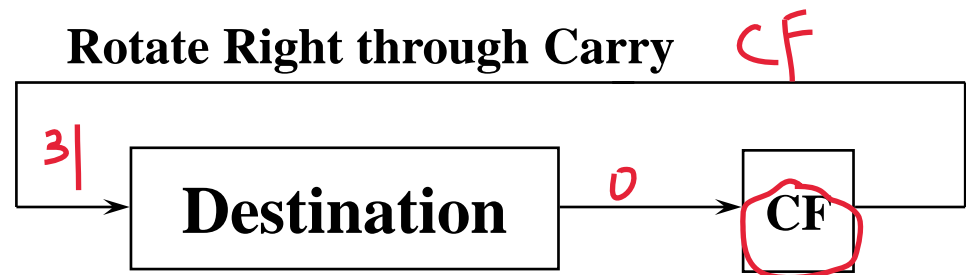
No ROS, because it can be achieved by ROR



🔥 Barrel Shifter - Rotations

Rotate Right Extended (RRX) by one bit.

- This operation uses the CPSR C flag as a 33rd bit.
 - `RRX{cond}{S} Rd, Rm`
 - `MOV r1, r0, RRX`
 - `RRX r1, r0`



Using a Shifted Register

- Multiplications by a constant equal to a $((\text{power of } 2) \pm 1)$ can be done in one cycle.

- Example: $r0 = r1 * 5$
 $= r1 + (r1 * 4)$

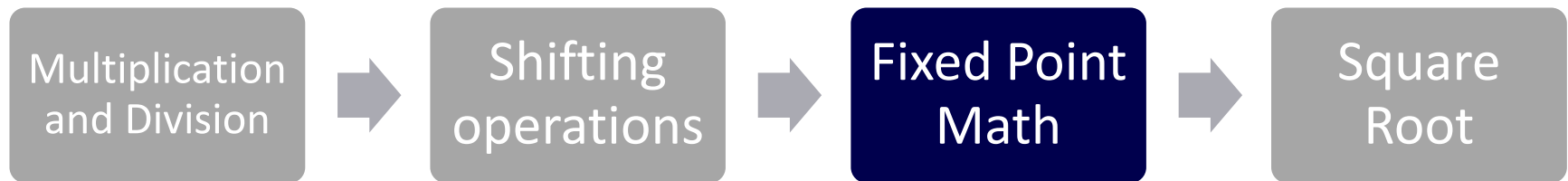
ADD r0, r1, r1, LSL #2

🔥 Using a Shifted Register -2

- Example: $r2 = r3 * 105$
 $= r3 * 15 * 7$
 $= r3 * (16 - 1) * (8 - 1)$

```
RSB r2, r3, r3, LSL #4    ; r2 = r3 * 15
RSB r2, r2, r2, LSL #3    ; r2 = r2 * 7
```


Advance Math Operations



Fixed Point Math

- We will use first 8 bits for the fraction and the last 24 bits for the integer part. 24 8
- What is r0 value that represent the value 1.5

r0 →

$$\frac{0}{23} \frac{1}{1} \frac{0}{7}$$

Fixed point

Choose the location of the point carefully, considering

- What **range** do you need?
 - from <smallest number> to <largest number>
- What **precision** do you need?
 - What is the required distance between successive numbers?

<u>2³</u>	2 ²	2 ¹	2 ⁰	2 ⁻¹	2 ⁻²	2 ⁻³	<u>2⁻⁴</u>	
8	4	2	1	0.5	0.25	0.125	0.0625	Base 10
0	0	0	0	<u>1</u>	0	<u>1</u>	<u>1</u>	0.6875
<u>1</u>	0	0	0	<u>1</u>	0	0	0	<u>8.5</u>



🔥 Multiplication with Fixed Point Math

- Multiplication of two fixed point registers cause **lose in integer** precision.

– @24.8 * @24.8 = @16.16 The fraction part bits will grow to

if we want to formalize the result to 16 bits

24.8, lsr #8(right, not left !!!!!!!!!!!)

$$2 \times 1.5 = 3$$

$$0010 \underbrace{0}_8 \times 00011000 \underbrace{0}_4 \rightarrow \underbrace{0}_{12} 0011 \underbrace{0}_{16}$$

When calculating the mul, do not calculate integer and fraction seperately

$$\begin{array}{r} 0010 \underbrace{0} \\ \times 00011 \underbrace{0} \\ \hline 0011 \underbrace{0} \end{array}$$

🔥 Division with Fixed Point Math

- Division of two fixed point registers cause **loss in fractional precision.**

$$- @16.16 / @24.8 = @16.8$$

$$3/2 = 1.5$$

$$0x300 \div 0x200$$

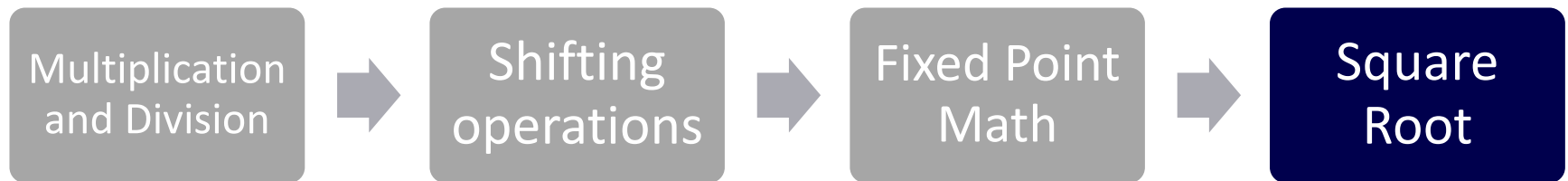


$$0x30000 \div 0x200$$



$$0x180$$

Advance Math Operations



Square Root Algorithm -1/3

- Newton-Raphson's Method (Numerical).
- $y_{n+1} = \left(y_n + \frac{w}{y_n} \right) * 1/2$

n	w	y_n	$\frac{w}{y_n}$	$\left(y_n + \frac{w}{y_n} \right)$	$\left(y_n + \frac{w}{y_n} \right) * 1/2$
0	4	4	1	5	2.5
1	4				
2	4				

Square Root Algorithm -2/3

- Newton-Raphson's Method (Numerical).
- $y_{n+1} = \left(y_n + \frac{w}{y_n} \right) * 1/2$

n	w	y_n	$\frac{w}{y_n}$	$\left(y_n + \frac{w}{y_n} \right)$	$\left(y_n + \frac{w}{y_n} \right) * 1/2$
0	4	4	1	5	2.5
1	4	2.5	1.6	4.1	2.05
2	4				

Square Root Algorithm -3/3

- Newton-Raphson's Method (Numerical).
- $y_{n+1} = \left(y_n + \frac{w}{y_n} \right) * 1/2$

n	w	y_n	$\frac{w}{y_n}$	$\left(y_n + \frac{w}{y_n} \right)$	$\left(y_n + \frac{w}{y_n} \right) * 1/2$
0	4	4	1	5	2.5
1	4	2.5	1.6	4.1	2.05
2	4	2.05	1.951	4.001	2.0006

Square Root Code -1/4




n	w	y_n	$\frac{w}{y_n}$	$\left(y_n + \frac{w}{y_n}\right)$	$\left(y_n + \frac{w}{y_n}\right) * 1/2$
0	4	4	1	5	2.5
1	4	2.5	1.6	4.1	2.05
2	4	2.05	1.951	4.001	2.0006

🔥 Square Root Code -2/4

```
MOV r0, #4
MOV r1, r0
_loop:
UDIV r2, r0, r1
ADD r3, r1, r2
MOV r4, r3, lsr #1
MOV r1, r4
B _loop
```

infinite loop



n	w	y_n	$\frac{w}{y_n}$	$\left(y_n + \frac{w}{y_n}\right)$	$\left(y_n + \frac{w}{y_n}\right) * 1/2$
0	4	4	1	5	2.5
1	4	2.5	1.6	4.1	2.05
2	4	2.05	1.951	4.001	2.0006

🔥 Square Root Code -3/4

```
MOV r0, #4
MOV r1, r0
_loop:
UDIV r2, r0, r1
ADD r3, r1, r2
MOV r4, r3, lsr #1
SUB r5, r1, r4
CMP r5, 0.001
BLT _end
MOV r1, r4
B _loop
_end: b _end
```

Consider precision

When y_n - final result < 0.001 , program will end.

hash is needed and 0.001 is illegal

	r_0	r_1	r_2	r_3	r_4
n	w	y_n	$\frac{w}{y_n}$	$\left(y_n + \frac{w}{y_n}\right)$	$\left(y_n + \frac{w}{y_n}\right) * 1/2$
0	4	4	1	5	2.5
1	4	2.5	1.6	4.1	2.05
2	4	2.05	1.951	4.001	2.0006

🔥 Square Root Code -4/4



```
MOV r0,#4
MOV r1,r0
MOV r0,r0,ls1 #16 @16.16
MOV r1,r1,ls1 #8 @24.8
_loop:
UDIV r2,r0,r1 @16.8
ADD r3,r1,r2 @24.8
MOV r4,r3, lsr #1 @24.8
SUB r5,r1,r4 @24.8
CMP r5,#1 @24.8
BLT _end
MOV r1,r4 @24.8
```

```
B _loop
_end: b _end
```

n	w	y_n	$\frac{w}{y_n}$	$\left(y_n + \frac{w}{y_n}\right)$	$\left(y_n + \frac{w}{y_n}\right) * 1/2$
0	4	4	1	5	2.5
1	4	2.5	1.6	4.1	2.05
2	4	2.05	1.951	4.001	2.0006



Summary

1. Multiplication and division instructions.
2. Shifting operations.
3. Fixed point math.
4. Example: Newton and Raphson's method