Chapter 3 Arithmetic and Logic Instructions

- Unsigned numbers
  - Zero or positive numbers
  - 0 – 0xFFFFFFFF (0 to 2^{32} -1) for 32-bit operand
- Affecting flags in ARM instructions
  - We need to explicitly request it by "S suffix"

Instruction (Flags unchanged)		Instruction (Flags updated)	
ADD	Add	**ADDS**	Add and set flags
ADC	Add with carry	**ADCS**	Add with carry and set flags
SUB	SUBS	**SUBS**	Subtract and set flags
SBC	Subtract with carry	**SBCS**	Subtract with carry and set flags
MUL	Multiply	**MULS**	Multiply and set flags
UMULL	Multiply long	**UMULLS**	Multiply Long and set flags
RSB	Reverse subtract	**RSBS**	Reverse subtract and set flags
RSC	Reverse subtract with carry	**RSCS**	Reverse subtract with carry and set flags
Note: The above instruction affect all the N, Z, C, and V flag bits of CPSR (current program status register) but the N and V flags are for signed data and are discussed in Chapter 5.			

1

ADD and ADDS (1)

```
1 .global _start
2 _start:
3
4     LDR r2, =0xFFFFFFF5
5     MOV r3, #0x0B
6     ADDS r1, r2, r3 //r1 = r2 + r3
7
8     LDR r4, =0xFFFFFFFF
9     ADDS r5, r3, r4   //r5 = r3+r4
```

The **first** ADDS in line 6

Z=____

C=____

The **second** ADDS in line 9

Z=____

C=____

2

ADD and ADDS (2)

```
1 .global _start
2 _start:
3 //Example3-2.s
4     LDR r2, =0xFFFFFFF1
5     MOV r3, #0x0F
6     ADDS r3, r3, r2 //r3 = r2 + r3
7
8     ADD r3, r3, #0x7  //r3 = r3+ 7
9     MOV r1, r3
```

The **first** ADDS in line 6

Z=____ C=__

The ADD in line 8

Z =___ and C =__

3

Increment / Decrement (increase/decrease by 1)

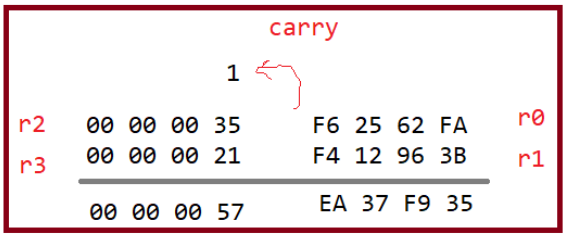
- There is no "increment" instruction in ARM
- We use, instead, add 1
- EX) `ADD r4, r4, #1  // r4 = r4 +1`

- There is no "decrement" instruction in ARM
- We use, instead, subtract 1
- EX) `SUB r4, r4, #1  // r4 = r4 +1`

4

ADC (Add with Carry) - 1

- Format: ADC rd, rn, op2 // rd = rn + op2 + Carry
- **Addition of Multi-Word data**
  - Big numbers added could reach 8 byte wide or more
  - Registers hold only 4 bytes
  - Breakdown 8-byte data, break into two (2) 4-byte words
  - Example: addition of 0x35 F6 25 62 FA (5-byte data) and 0x 21 F4 12 96 3B (5-byte data)



5

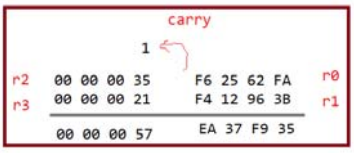
ADC (Add with Carry) - 2

- **Addition of Multi-Word data**
  - Example: addition of 0x35 F6 25 62 FA (5-byte data) and 0x 21 F4 12 96 3B (5-byte data)

Editor (Ctrl-E)

Compile and Load (F5) Language: ARMv7 Pro

```
1  .global _start
2  _start:
3  //Example3-4.s
4  //Addition of multi-word data
5  // A = 0x 35 F6 25 62 FA  (5-byte data)
6  // B = 0x 21 F4 12 96 3B  (5-byte data)
7
8      LDR r0, =0xF62562FA
9      LDR r1, =0xF412963B
10     MOV r2, #0x35
11     MOV r3, #0x21
12     ADDS r5, r1,r0
13     ADC r6, r2,r3
14
```



6

SUB (Subtraction of unsigned numbers)

Format: SUB rd, rn, op2 //rd = rn – op2

- Actual work of ARM: rn + 2's complement of op2
- Because CPU has adder circuitry

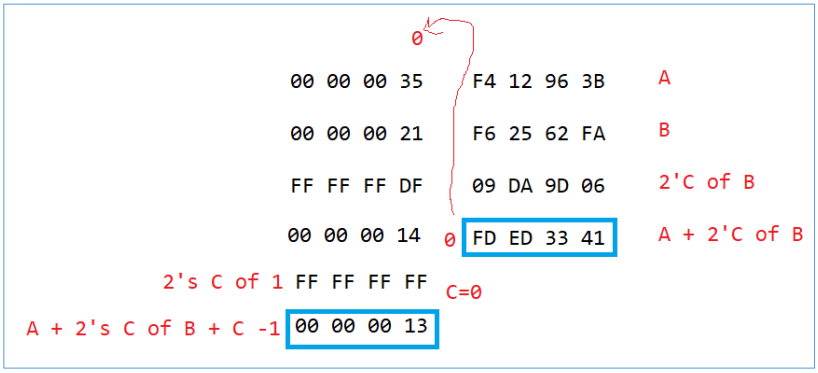
Example: (a) 4F – 39 (b) 4F - 05

```
1  .text
2  .global _start
3  _start:
4  //Example3-5.s
5  //Subtraction of unsigned
6  // SUB rd, rn, op2  ; rd
7
8      MOV r2, #0x4F
9      MOV r3, #0x39
10     SUBS r4, r2, r3  //r4
11     //     00 00 00 4F
12     //     FF FF FF C7
13     // 1   00 00 00 16
14
15     SUBS r5, r2,#0x05
16     //     00 00 00 4F
17     //     FF FF FF FB  +
18     // 1   00 00 00 4A
19
20 _stop:
21     B _stop
22 .end
```

7

SBC (Subtract with Carry)

- For subtraction of multi-word numbers
- Format
  - SBC rd, rn, op2 //**rd = rn-op2 + C - 1**
- Example: A = 0x 35 F4 12 96 3B B = 0x 21 F6 25 62 FA
  - A - B



8

SBC (Subtract with Carry)

- For subtraction of multi-word numbers
- Format
  - SBC rd, rn, op2
    //rd = rn-op2 + C - 1
- Example: A = 0x 35 F4 12 96 3B
  B = 0x 21 F6 25 62 FA
  - A - B

Editor (Ctrl-E)

Compile and Load (F5) Language: ARMv7 Example3-4.s [chan

```
1  .text
2  .global _start
3  _start:
4  //Example3-8.s
5  //Subtraction of multi-word numbers
6  // SBC rd, rn, op2  ; rd = rn - op2 + C
7  // A = 0x 35 F4 12 96 3B
8  // B = 0x 21 F6 25 62 FA
9      LDR r0,=0xF62562FA
10     LDR r1, =0xF412963B
11     MOV r2, #0x21
12     MOV r3, #0x35
13     SUBS r5, r1, r0  //r5 = r1 + 2's C of r1
14     //    F4 12 96 3B
15     //    09 DA 9D 06
16     // 0  FD ED 33 41   C=0 no Carry
17
18 //  SUBS r6, r3,r2
19
20     SBC r6, r3,r2    //r6 = r3 + 2's C of r2 + C - 1
21     //    00 00 00 35
22     //    FF FF FF DF
23     //    FF FF FF FF
24     //  1 00 00 01 13   C=1
25
26 _stop:
27     B _stop
28 .end
29
```

Multiplication (of unsigned numbers)

MUL --- regular multiply --- result is less than 32-bit
UMULL – long multiply – result is greater than 32-bit

Instruction	Source 1	Source 2	Destination	Result
MUL	Rn	Op2	Rd (32 bits)	Rd=Rn×Op2
UMULL	Rn	Op2	RdLo, RdHi (64 bits)	RdLo:RdHi=Rn×Op2

Note 1: *Using MUL for word × word multiplication preserves only the lower 32 bit result in Rd and the rest are dropped. If the result is greater than 0xFFFFFFFF, then we must use UMULL (unsigned Multiply Long) instruction.*

Note 2: *In some CPUs the C flag is used to indicate the result is greater than 32-bit but this is not the case with ARM MUL instruction.*

MUL (multiply)
Format: MUL rd, rn, op2 // rd = rn * op2
UMULL (Long multiply)
Format: UMULL rdLo, rdHi, Rn, op2 //rdHi:rdLo = rn*op2

10

MUL & UMULL

Example:

(a) 0x25 * 0x65

(b) 100,000 * 150,000

Editor (Ctrl-E)

Compile and Load (F5) Language: ARMv7 Example3-

```
1  .text
2  .global _start
3  _start:
4  //Example3-11.s
5  //MUL (regular multiply)
6  //UMULL (long multiply)
7      MOV r1, #0x25
8      MOV r2, #0x65
9      MUL r3, r1, r2  // r3 = r1 * r2
10     // 37 * 101 = 3737 = 0x00 00 0e 99
11
12     LDR r4, =100000
13     LDR r5, =150000
14     MUL r6, r4, r5  // r6 = r3* r5
15     //100000 * 150000 > 32-bit
16     //r6 has only the lower 32-bit result
17
18     UMULL r8, r7, r4, r5 //r7:r8 = r3*r5
19     //r8 has the lower 32-bit result
20     //r7 has the higher 32-bit result
21
22 _stop:
23     B _stop
24 .end
25
```

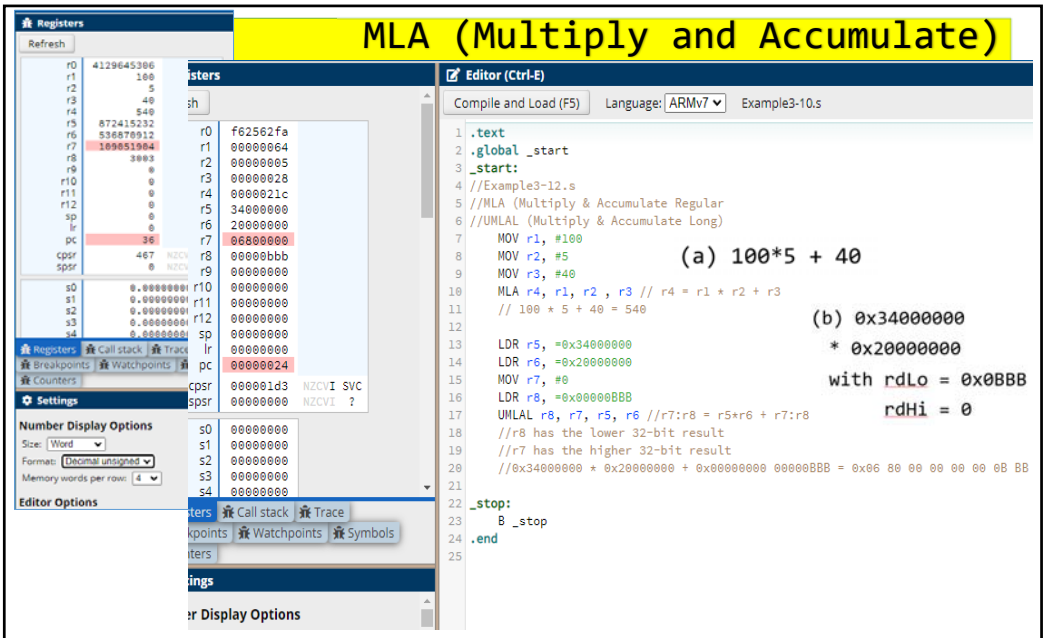
11

MLA (Multiply and Accumulate)

- Purpose: (a) Multiply 2 variables and (b) add the result to another variable – in a single instruction: MLA (regular) and UMLAL (long)
- Format: MLA rd, rm, rs, rn // rd = rm*rs + rn
- Format: UMLAL rdLo, rdHi, rn, op2 // rdHi:rdLo = rn * op2 + rdHi:rdLo
- Example
- (a) 100*5 + 40
- (b) 0x34000000
  * 0x20000000
  with rdLo = 0x0BBB
  rdHi = 0

12

MLA (Multiply and Accumulate)



Division

- Some ARM families do not have instructions for division of unsigned numbers
- Because it takes many gates to implement it
- We use SUB instructions to perform the division (in the next chapter)

Logic Instructions

Instruction (Flags Unchanged)	Action	Instruction (Flags Changed)	Hexadecimal
AND	ANDing	**ANDS**	Anding and set flags
ORR	ORRing	**ORS**	Oring and set flags
EOR	Exclusive-ORing	**EORS**	Exclusive Oring and set flags
BIC	Bit Clearing	**BICS**	Bit clearing and set flags

- **AND**: bit-wise logical AND on the operands and place the result in the destination
  - Format: AND rd, rn , op2 // rd ← rn && op2
- **ORR**: bit-wise OR
  - Format: ORR rd, rn, op2 //rd ← rn || op2
- **EOR**: bit-wise exclusive OR
  - Format: bit-wise Exclusive-OR
  - Format: EOR rd, rn, op2 //rd ← rn ^ op2
- **BIC** (bit clear): clear selected bits (held by op2) of the rn register
  - Format: BIC rd, rn, op2 //Clear bits of rn specified by op2 and place the results in rd

15

Logic Instructions – Example: Q) Write a code which performs the 6 logical instructions shown below.

1. 0x35 AND 0x0F
2. 0x97 AND 0xF0
3. 0x04 OR 0x68
4. 0x97 OR 0xF0
5. 0x54 EOR 0x78
6. 0xAA EOR 0x04
7. Clear bit 3 of 0xAA

16

MVN (Move NOT)

Format:

MVN rd, rn

//Move the complement (**NOT**) of rn to rd

Example:

```
LDR r1, =0xFFFFEEEE
MVN r2, r1   //r2 = _____________________
LDR r3, =0xAAAAAAAA
MVN r4, r3    //r4 = _____________________
LDR r5, =#0xAAAAAAAA
MVN r6,#0
              //r6 = _____________
EOR r7,r6,r5
       //r7 = r6^r5 = _____________
```

17

Chapter3 - Class Activity 1

- Find the contents of register r3 after executing the following lines of code. Submit your answers.

(1)
```
MOV r0, #0xF0
MOV r1, #0x55
BIC r3, r1, r0
```
r3=

(2)
```
LDR r1, =0x55555555
MVN r0, #0
EOR r3, r1, r0
```
r3=

(3)
```
LDR r0, =0x95999999
LDR r1, =0x94FFFFFF
ADDS r3, r1, r0
```
r3=

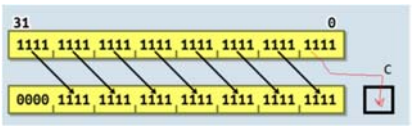
(4)
```
MOV r2, #0x01
LDR r1, #0xFFFFFFFF
ADD r3, r1, r2
ADCS r3, r3, #0
```
r3=

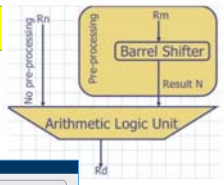
(5)
```
LDR r3, #0xFFFFFFFF
ADDS r3, r3, #1
```
r3=

18

Rotate and Barrel Shift

- Rm (operand) can be optionally shifted before fed to ALU
  - Shift amount: 8-bit immediate number (0 – 255)
  - LSR: logical shift right
  - LSL: Logical shift left
  - ROR: Rotate right
- **Logical Shift Right (LSR)**
  - **MSB** filled with 0
  - If **S** suffix is used, LSB will go to the carry flag (C)
  - Example of **LSR by 4**





Compile and Load (F5) Language: ARMv7

```
1  //Example3-16.s
2  .text
3  .global _start
4  _start:
5      MOV r0, #0x9A
6      MOVS r1, r0, LSR #3
7      //Shift r0 to right 3 times
8      //       r0 = 0000 0000 1001 1010
9      //r0 LSR #3 = 0000 0000 0001 0011
10     // then 0 to C
11     //then store the result in r1
12
13     //Another way
14     MOV r2, #0x9A
15     MOV r3, #0x03
16     MOVS r4, r2,LSR r3
17
18
19 stop:
20     B stop
21 .end
22
```

r0	0000009a
r1	00000013
r2	0000009a
r3	00000003
r4	00000013
r5	00000000
r6	00000000
r7	00000000
r8	00000000
r9	00000000
r10	00000000
r11	00000000
r12	00000000
sp	00000000
lr	00000000
pc	00000014
cpsr	000001d3
spsr	00000000

19

Rotate and Barrel Shift

- **Logical Shift Right (LSR) - Example 2**

Compile and Load (F5) Language: ARMv7 untitled

```
1  //Example3-17.s
2  .text
3  .global _start
4  _start:
5      LDR r1, =#0x777
6      LDR r2, =0xA6D
7      ADDS r3, r1, r2, LSR #4
8      //Shift r2 to right 4 times
9      //       r2 = 0000 1010 0110 1101
10     //r2 LSR #4 = 0000 0000 1010 0110
11     //then add r1 and store the result in r3
12
13     //LSR is the same as divide by 2
14     MOV r4, #0x88
15     MOVS r5, r4,LSR #1
16     MOVS r6, r4,LSR #2
17     MOVS r7, r4,LSR #3
18
19     MOVS r8, r4, LSR #4
20
21 stop:
22     B stop
23 .end
24
```

Registers — Refresh

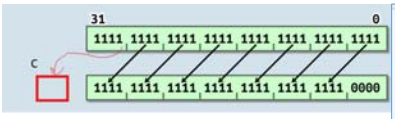
r0	0
r1	1911
r2	2669
r3	2077
r4	136
r5	68
r6	34
r7	17
r8	8
r9	0
r10	0
r11	0
r12	0
sp	0
lr	0
pc	32
cpsr	536871379 NZCVI SVC
spsr	0 NZCVI ?

r0	00000000
r1	00000777
r2	00000a6d
r3	0000081d
r4	00000088
r5	00000044
r6	00000022
r7	00000011
r8	00000008
r9	00000000
r10	00000000
r11	00000000
r12	00000000
sp	00000000
lr	00000000
pc	00000020
cpsr	200001d3 NZCVI SVC
spsr	00000000 NZCVI ?

20

Rotate and Barrel Shift

- **Logical Shift Left (LSL)**
  - **LSB** filled with 0
  - If **S** suffix is used, **MSB** will go to the carry flag (C)
  - Example of LSL by 4



r0	00000000
r1	0f000006
r2	00000600
r3	0f000006
r4	00000008
r5	00000600
r6	00000007
r7	0000000e
r8	0000001c
r9	00000038
r10	00000000
r11	00000000
r12	00000000
sp	00000000
lr	00000000
pc	00000024
cpsr	200001d3 N\C\I SVC
spsr	00000000 NZCVI ?
s0	00000000
s1	00000000

Compile and Load (F5) Language: ARMv7

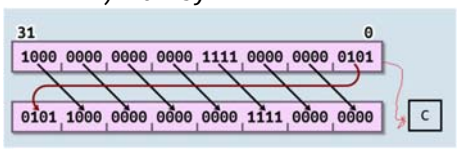
```
1  //Example3-19.s
2  .text
3  .global _start
4  _start:
5      LDR r1, =#0x0F000006
6      MOVS r2, r1, LSL #8
7      //Shift r1 to left 8 times
8      //       r1 = 0000 0000 0000 0110
9      //r1 LSL #8 = 0000 0110 0000 0000
10     //then store r1 to r2
11
12     //Another way
13     LDR r3, =#0x0F000006
14     MOV r4,#0x08
15     MOVS r5, r3,LSL r4
16
17     //LSL is the same as multiply by 2
18
19     MOV r6, #0x07
20     MOV r7, r6, LSL #1
21     MOV r8, r6, LSL #2
22     MOV r9, r6, LSL #3
23
24
25 stop:
26     B stop
27 .end
28
```

r0	0
r1	251658246
r2	1536
r3	251658246
r4	8
r5	1536
r6	7
r7	14
r8	28
r9	56
r10	0
r11	0
r12	0
sp	0
lr	0
pc	36
cpsr	536871379 N\C\I SVC
spsr	0 NZCVI ?

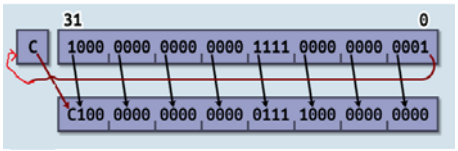
21

ROR [*rotate right*] and RRX [*rotate right extended*]

- **Simple rotation – ROR**
  - LSB to MSB
  - Ex) ROR by 4



- **No <u>Rotate Left</u> in ARM**
  - Rotate Left using ROR
  - Rotate left n bits
  - = Rotate right (32 – n) bits
- **Rotation through Carry – RRX**
  - Rotate only once




```
1  //Example3-20.s
2  .text
3  .global _start
4  _start:
5      MOV r1, #0x36
6      //r1 = 0b 0000 0000 0000 0000 0000 0011 0110
7      MOVS r2, r1, ROR #1
8      //r2 = 0b 0000 0000 0000 0000 0000 0001 1011 C=0
9      MOVS r3, r2, ROR #1
10     //r3 = 0b 1000 0000 0000 0000 0000 0000 1101 C=1
11     MOVS r4, r3, ROR #1
12     //r4 = 0b 1100 0000 0000 0000 0000 0000 0110 C=1
13
14     //Combined together
15     MOV r5, #0x36
16     MOV r6, #3
17     MOVS r7, r5, ROR r6
```

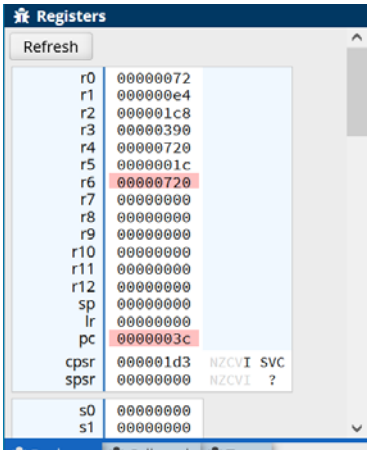
r0	00000000
r1	00000036
r2	0000001b
r3	8000000d
r4	c0000006
r5	00000036
r6	00000003
r7	c0000006
r8	0000001c
r9	00000038
r10	00000000
r11	00000000
r12	00000000
sp	00000000
lr	00000000
pc	0000001c
cpsr	a00001d3 N\C\I SVC
spsr	00000000 NZCVI ?

22

ROR [*rotate right*] and RRX [*rotate right extended*]

- **No Rotate Left in ARM**
  - **Rotate Left using ROR**
  - **Rotate left n bits**
  - **= Rotate right (32 – n) bits**

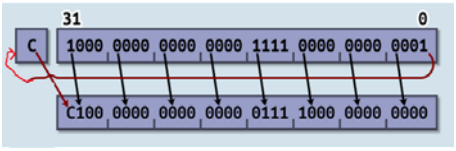
```
20      // Rotate left using ROR
21      // Rotate left n time = ROR (32-n) times
22
23      LDR r0, =0x00000072
24      MOVS r1, r0, ROR #31
25      MOVS r2, r1, ROR #31
26      MOVS r3, r2, ROR #31
27      MOVS r4, r3, ROR #31
28      //or
29      MOV r0, #0x72
30      MOV r5, #28
31      MOVS r6,r0, ROR r5
32
```



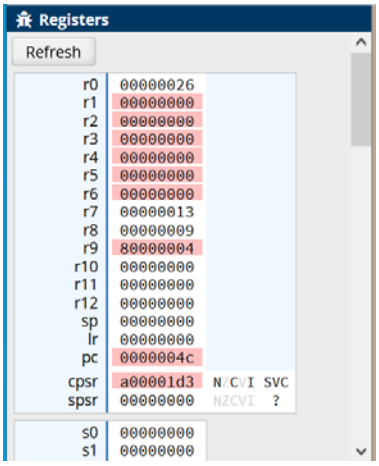
23

ROR [*rotate right*] and RRX [*rotate right extended*]

- **Rotation through Carry – RRX**
  - **Rotate only once**




```
33
34      //RRX -- Rotate right extended (rotate through C)
35      //Rotate just once
36      MOV r0, #0x26
37      // 0000 ... 0000 0000 0010 0110
38      MOVS r7, r0, RRX
39      // 0000 ... 0000 0000 0001 0011  C=0
40      MOVS r8, r7, RRX
41      // 0000 ... 0000 0000 0000 1001  C=1
42      MOVS r9, r8, RRX
43      // 1000 ... 0000 0000 0000 0100  C=1
44
```



24

Shift and Rotate Instruction (without using barrel shifter)

- Shift/Rotate without using Barrel Shifter
  - MOV r0, r2, LSL #8 //with barrel shift
  - LSL r0, r2, #8 //regular shift instruction
- ASR – Arithmetic Shift Right
  - **No** ASL !!!
- LSL – Logical shift left
- LSR – Logical shift right
- ROR – Rotate Right
- RRX – Rotate Right with Extend
- **ASR** (Arithmetic Shift Right)
  - For **signed number** shifting
  - Format: ASR rd, rm, rn
  - //shift rm by rn times
  - //and store it in rd
  - LSB bit removed
  - **Empty bit filled with MSB**
    - Keep the sign
  - **Divide by 2**

```
1  //Example3-21.s
2  .text
3  .global main
4  main:
5  //ASR --- arithmentic shift right
6  // empty bit is filled with MSB
7      LDR r0, =0xFFFFFF82
8      ASR r1, r0, #6
9      //r0 = 1111 1111 1111 1111 1111 1111 1000 0010
10     //r1 = 1111 1111 1111 1111 1111 1111 1111 1110
11
12     LDR r2, =0x2000FF18
13     MOV r3, #12
14     ASR r4,r2,r3
15     // r2 = 0010 0000 0000 0000 1111 1111 0001 1000
16     // r4 = 0000 0000 0000 0010 0000 0000 0000 1111
17
18     LDR r5, =0x0000FF18
19     MOV r6, #16
20     ASR r7, r5, r6
21     //r5= 0000 0000 0000 0000 1111 1111 0001 1000
22     //r7= 0000 0000 0000 0000 0000 0000 0000 0000
23
24 stop:
25     B stop
26 .end
27
```

r0	ffffff82	
r1	fffffffe	
r2	2000ff18	
r3	0000000c	
r4	0002000f	
r5	0000ff18	
r6	00000010	
r7	00000000	
r8	00000000	
r9	00000000	
r10	00000000	
r11	00000000	
r12	00000000	
sp	00000000	
lr	00000000	
pc	00000020	
cpsr	000001d3	NZCVI SVC
spsr	00000000	NZCVI ?
s0	00000000	
s1	00000000	

Shift and Rotate Instruction (w/o using barrel shifter)

- **LSL** (Logical Shift Left)
  - Format: LSL rd, rm, rn
  - //shift rm left by rn times
  - //and store it in rd
  - MSB removed (or goes to Carry)
    - LSLS updates Carry flag
  - **Empty bit filled with 0**
  - **Multiply by 2**

r0	00000010	
r1	00001000	
r2	00000018	
r3	0000000c	
r4	00018000	
r5	0000ff18	
r6	00000010	
r7	ff180000	
r8	00000000	
r9	00000000	
r10	00000000	
r11	00000000	
r12	00000000	
sp	00000000	
lr	00000000	
pc	00000020	
cpsr	000001d3	NZCVI SVC
spsr	00000000	NZCVI ?

Compile and Load (F5) Language: ARMv7 untitled.s [changed

```
1  //Example3-22.s
2  .text
3  .global main
4  main:
5  //LSL--- Logical Shift Left
6  // empty bit is filled with 0
7  //MSB removed (or goes to C)
8      LDR r0, =0x00000010
9      LSL r1, r0, #8
10     //r0 = 0000 0000 0000 0000 0000 0000 0001 0000
11     //r1 = 0000 0000 0000 0000 0001 0000 0000 0000
12
13     LDR r2, =0x00000018
14     MOV r3, #12
15     LSL r4,r2,r3
16     // r2 = 0010 0000 0000 0000 0000 0000 0001 1000
17     // r4 = 0000 0000 0000 0001 1000 0000 0000 0000
18
19     LDR r5, =0x0000FF18
20     MOV r6, #16
21     LSL r7, r5, r6
22     //r5= 0000 0000 0000 0000 1111 1111 0001 1000
23     //r7= 1111 1111 0001 1000 0000 0000 0000 0000
24
25 stop:
26     B stop
27 .end
28
```

Shift and Rotate Instruction (w/o using barrel shifter)

- **LSR** (Logical Shift Right)
  - Format: LSR rd, rm, rn
  - //shift rm right by rn times
  - //and store it in rd
  - LSB removed (or goes to Carry)
    - LSRS updates Carry flag
  - **Empty bit filled with 0**
  - **Divide by 2**

r0	00001000	r0	4096
r1	00000010	r1	16
r2	00018000	r2	98304
r3	0000000c	r3	12
r4	00000018	r4	24
r5	7f180000	r5	2132279296
r6	00000010	r6	16
r7	00007f18	r7	32536
r8	00000000	r8	0
r9	00000000	r9	0
r10	00000000	r10	0
r11	00000000	r11	0
r12	00000000	r12	0
sp	00000000	sp	0
lr	00000000	lr	0
pc	00000020	pc	32
cpsr	000001d3 NZCVI SVC	cpsr	467 NZ
spsr	00000000 NZCVI ?	spsr	0 NZ

```
1  //Example3-23.s
2  .text
3  .global main
4  main:
5  //LSR--- Logical Shift Right
6  //Empty bit is filled with 0
7  //LSB removed (or goes to C)
8      LDR r0, =0x00001000
9      LSR r1, r0, #8
10     //r0 = 0000 0000 0000 0000 0001 0000 0000 0000
11     //r1 = 0000 0000 0000 0000 0000 0000 0001 0000
12
13     LDR r2, =0x00018000
14     MOV r3, #12
15     LSR r4,r2,r3
16     // r2 = 0000 0000 0000 0001 1000 0000 0000 0000
17     // r4 = 0000 0000 0000 0000 0000 0000 0001 1000
18
19     LDR r5, =0x7F180000
20     MOV r6, #16
21     LSR r7, r5, r6
22     //r5= 0111 1111 0001 1000 0000 0000 0000 0000
23     //r7= 0000 0000 0000 0000 0111 1111 0001 1000
24
25 stop:
26     B stop
27 .end
28
```

27

Shift and Rotate Instruction (w/o using barrel shifter)

- **ROR** (Rotate Right)
  - Format: ROR rd, rm, rn
  - //Rotate rm right by rn times
  - //and store it in rd
  - LSB is moved to MSB (and goes to Carry)

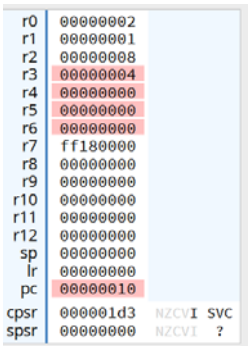
r0	00000010
r1	10000000
r2	00000018
r3	0000000c
r4	01800000
r5	0000ff18
r6	00000010
r7	ff180000
r8	00000000
r9	00000000
r10	00000000
r11	00000000
r12	00000000
sp	00000000
lr	00000000
pc	00000020
cpsr	000001d3 NZCVI SVC
spsr	00000000 NZCVI ?

```
1  //Example3-24.s
2  .text
3  .global main
4  main:
5  //ROR-- Rotate Right
6  //LSB is moved to MSB (and goes to C)
7      LDR r0, =0x00000010
8      ROR r1, r0, #8
9      //r0 = 0000 0000 0000 0000 0000 0000 0001 0000
10     //r1 = 0001 0000 0000 0000 0000 0000 0000 0000
11
12     LDR r2, =0x00000018
13     MOV r3, #12
14     ROR r4,r2,r3
15     // r2 = 0000 0000 0000 0000 0000 0000 0001 1000
16     // r4 = 0000 0001 1000 0000 0000 0000 0000 0000
17
18     LDR r5, =0x0000FF18
19     MOV r6, #16
20     ROR r7, r5, r6
21     //r5= 0000 0000 0000 0000 1111 1111 0001 1000
22     //r7= 1111 1111 0001 1000 0000 0000 0000 0000
23
24 stop:
25     B stop
26 .end
27
```

28

Shift and Rotate Instruction (w/o using barrel shifter)

- **RRX** (Rotate Right with Extend)
  - Format: RRXD rd, rm
  - //Rotate rm right by 1 bit through Carry
  - //C is shifted to MSB




```
1  //Example3-25.s
2  .text
3  .global main
4  main:
5  //RRX- Rotate Right 1 bit with Extend
6  //LSB is moved to C and C is moved to MSB
7      LDR r0, =0x00000002
8      RRX r1, r0
9      //r0 = 0000 0000 0000 0000 0000 0000 0000 0010
10     //r1 = 0000 0000 0000 0000 0000 0000 0000 0001
11
12
13     MOV r2, #0x08
14     RRXS r3, r2
15     // r2 = 0000 0000 0000 0000 0000 0000 0000 1000
16     // r4 = 0000 0001 1000 0000 0000 0000 0000 0100  C=0
17
18
19 stop:
20     B stop
21 .end
22
```

29

Chapter3 - Class Activity 2

- **Find (a)the contents of register r3 and (b) C flag, after executing the following lines of code. Submit your answers.**

(1)
```
MOV r0, #0x04
MOV r1, r0, LSR #2
MOVS r3, r0, LSR r1
```

(2)
```
LDR r1, =0xA0F2
MOV r2, #0x03
MOVS r3, r1, LSL r2
```

(3)
```
LDR r1, =0xB085
MOV r2, #3
MOVS r3, r1, LSR r2
```

(4)
```
LDR r1, =0x1234
MOV r2, #0x08
MOVS r3, r1, ROR r2
```

(5)
```
MOV r0, #0xAA
MOVS r3, r0, RRX
```

30