

Logic and Shift/Rotate Instructions

Computer Engineering 1

CT Team: A. Gieriet, J. Gruber, B. Koch, M. Loeser, M. Meli, M. Rosenthal, M. Ostertag, A. Rüst, J. Scheier

Motivation



So far, operands were bytes, half-words or words

- How can we work on a unit as small as a bit?
- How can we look at a bit or a pattern of bits within a larger unit?
- How can we modify a particular bit or a pattern of bits without affecting other bits?

Logic operations

- Help deal efficiently with elements at bit level
- Enable the implementation of algorithms where Boolean operations are needed

Like in HW, shift operations can enable

- Division and multiplication
- Test of certain bits
- Communication, ...

Motivation



06.09.2019

Instructions to process data in ALU

arithmetic
 Add

Addition, Subtraction, Increment, Decrement,

Multiplication, Division

logic

NOT, AND, OR, XOR

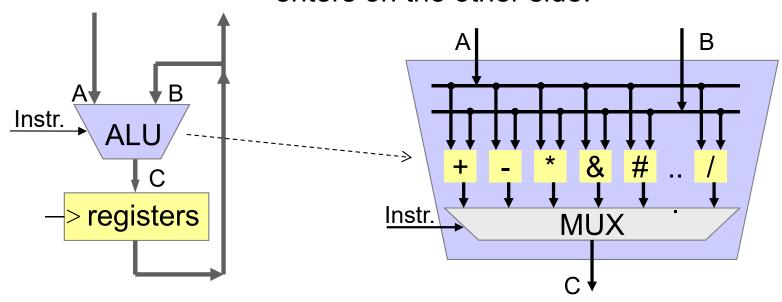
• shift

3

Left/right shift. Fill up with 0 or MSB

rotate

Cyclic left/right shift: What falls out enters on the other side.



Agenda



- Logic Instructions
 - Bit manipulations
- Shift/Rotate Instructions
- Multiplication with a Constant

Learning Objectives



At the end of this lesson you will be able

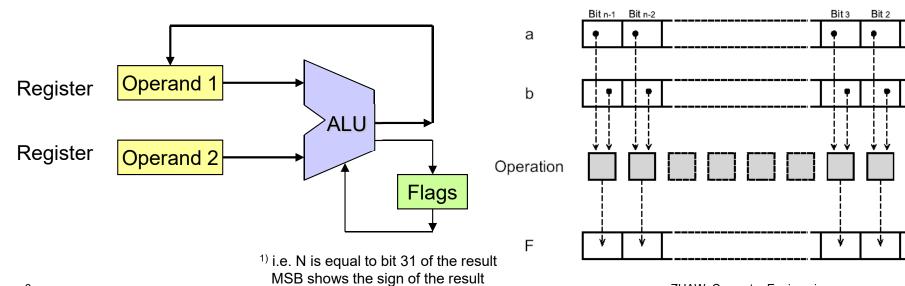
- to enumerate and to apply the ARM instructions for logic as well as for shift/rotate operations
- to determine (with the help of documents) the state of the ARM Flags (N, Z, C, V) after the execution of an instruction
- to understand and interpret ARM assembly programs with logic and shift/rotate operations
- to explain bit manipulations based on examples
- to set, clear or invert one or several bits in a bit pattern
- to implement a multiplication with a constant in assembly using shift and add instructions

Logic Instructions



Overview

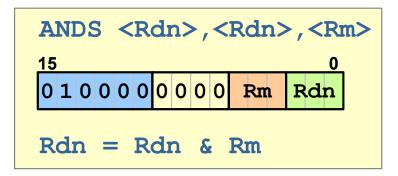
Mnemonic	Instruction	Function	C-Operator	
ANDS	Bitwise AND	Rdn & Rm	a & b	flows NI - requite (245-1)
BICS	Bit Clear	Rdn & !Rm	a & ~b	flags N = result<31> $^{1)}$ Z = 1 if result = 0
EORS	Exclusive OR	Rdn \$ Rm	a ^ b	Z = 0 otherwise
MVNS	Bitwise NOT	!Rm	~a	C and V unchanged
ORRS	Bitwise OR	Rdn # Rm	a¦b	

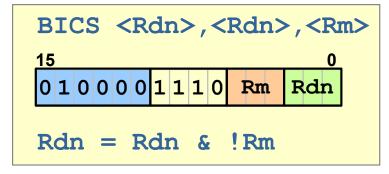


Logic Instructions



Bitwise Operations





```
EORS <Rdn>, <Rdn>, <Rm>
15 0
0 1 0 0 0 0 0 0 0 1 Rm Rdn

Rdn = Rdn $ Rm
```

```
MVNS <Rd>, <Rm>
15 0
0 1 0 0 0 0 1 1 1 1 Rm Rd

Rd = !Rm
```

- All these operations affect the flags:
 - N and Z according to result
 - C and V unchanged
- Only low registers

Logic Instructions



Example

- Update flags N and Z
- Only low registers possible!

```
ANDS <Rdn>, <Rdn>, <Rm>
15 0
0 1 0 0 0 0 0 0 0 0 Rm Rdn

Rdn = Rdn & Rm
```

```
00000002 4011
                         R1,R1,R2
                                     : R1 = R1 AND R2
                 ANDS
00000004 4011
                         R1,R2
                 ANDS
                                     ; the same (dest = R1)
00000006 4337
                         R7,R7,R6
                                       R7 = R7 OR R6
                 ORRS
                         R3,R3,R4
00000008 4063
                 EORS
                                     ; R3 = R3 EXOR R4
0000000A 4388
                         R0,R0,R1
                                     ; R0 = R0 AND NOT(R1)
                BICS
0000000C 43D1
                         R1,R2
                MVNS
                                     ; R1 = NOT(R2)
```

Bit Manipulations



Bit Manipulations (Cortex-M0)

Clear bits, e.g. clear bits 5 and 1 in register R1

```
MOVS R2,#0x22 ;00100010b
BICS R1,R1,R2
```

Set bits, e.g. set bits 6 und 3 in register R1

```
MOVS R2,#0x48 ;01001000b
ORRS R1,R1,R2
```

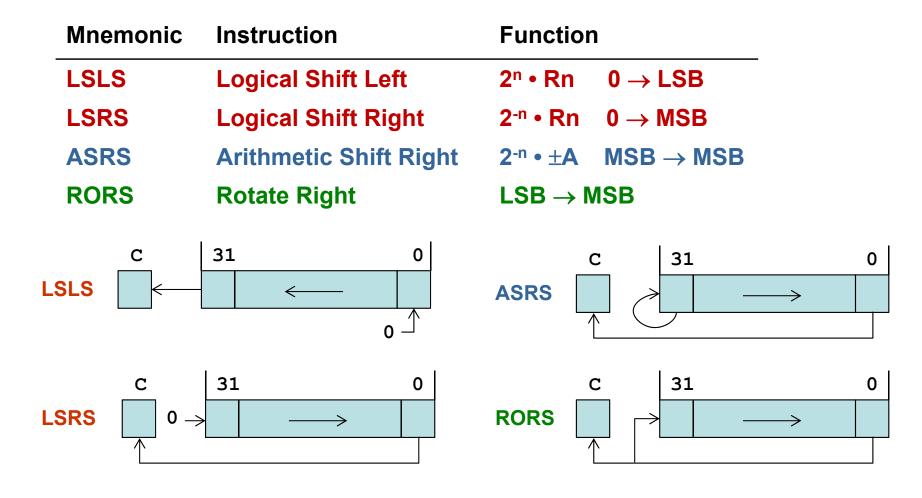
Invert bits, e.g. invert bits 4, 3 and 2 in register R1

```
MOVS R2,#0x1C ;00011100b
EORS R1,R1,R2
```

- What happens to the other bits?
- Compute the value of R1 after execution of all lines assuming that R1 contains 0xAF08 ' 24A3 at the start.
- What are the values of the flags after execution?



Overview



Note: rotate left does not exist



Opcodes (register)

- Low registers only
- Flags affected

```
flags N = result < 31 > 1

Z = 1 if result = 0

Z = 0 otherwise

C = last bit shifted out

V unchanged
```

```
ASRS <Rdn>, <Rdn>, <Rm>
15
0
0 1 0 0 0 0 0 1 0 0 Rm Rdn

Rdn = shift Rdn right
by Rm<7:0> bits,
fill with MSB<sup>2</sup>)
```

```
RORS <Rdn>, <Rdn>, <Rm>
15
0
0 1 0 0 0 0 0 1 1 1 Rm Rdn

Rdn = cyclic rotate right
by Rm<7:0> bits
```

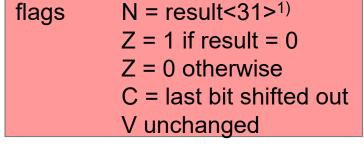
¹⁾ i.e. N is equal to bit 31 of the result / MSB shows the sign of the result

²⁾ see previous slide



■ Opcodes (immediate 0 – 31d)

low registers only

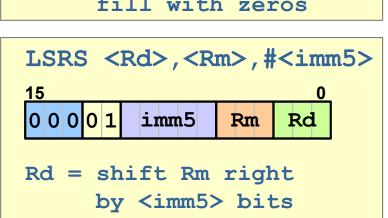


```
LSLS <Rd>, <Rm>, #<imm5>

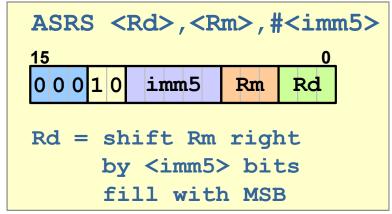
15

0
0 0 0 0 0 imm5 Rm Rd

Rd = shift Rm left
by <imm5> bits
fill with zeros
```



fill with zeros



- RORS (immediate) not supported
- LSRS/ASRS
 - <imm5> = 0 not allowed
- LSLS
 - C unaffected if <imm5> = 0



Examples

```
00000000 4904
                   LDR
                           R1,=0xCCCCCCC
00000002 2203
                           R2,#3
                   MOVS
00000004 4B04
                   LDR
                           R3,=0x66666666
00000006 4C05
                   LDR
                           R4,=0x999999999
00000008 25E3
                           R5, #0xE3
                   MOVS
0000000A 4111
                           R1,R1,R2
                                        ; register
                   ASRS
0000000C 111B
                           R3,R3,#4
                                        : immediate
                   ASRS
0000000E 41D4
                           R4,R4,R2
                                        ; register
                   RORS
0000010 00ED
                           R5,R5,#3
                                        ; immediate
                   LSLS
```

- What are the values of R1 R5 after execution?
- What are the values of the flags?



Examples Shift

Multiply with 2ⁿ

unsigned / signed

```
LSLS R0,R1,#1 ; *2
LSLS R0,R1,#2 ; *4
LSLS R0,R1,#3 ; *8
```

LSLS for signed and unsigned

Divide by 2ⁿ

unsigned

```
LSRS R0,R1,#1 ; /2
LSRS R0,R1,#2 ; /4
LSRS R0,R1,#3 ; /8
```

signed

```
ASRS R0,R1,#1 ; /2
ASRS R0,R1,#2 ; /4
ASRS R0,R1,#3 ; /8
```

LSRS and ASRS differ!



Multiplication with Constants using LSLS and ADDS

- Example: Multiplication with 13
 - Constant shown as power of 2: 13 = 8 + 4 + 1
 - R0 = 13 R1 \rightarrow R0 = (1 + 4 + 8) R1 = R1 + 4 R1 + 8 R1

```
MOVS R0, R1 ; R0 = R1 

LSLS R1, R1, #2 ; 4 • R1 

ADDS R0, R0, R1 ; R0 = R0 + 4 • R1 

LSLS R1, R1, #1 ; 2 • R1 -> 8 • R1 

ADDS R0, R0, R1 ; R0 = R0 + 8 • R1
```

Conclusion



- Logic Instructions
 - ANDS, BICS, EORS, MVNS, ORRS
- Shift/Rotate Instructions
 - LSLS, LSRS, ASRS, RORS