
ELEC2041
Microprocessors and Interfacing

Lecture 9: C/Assembler Logical and Shift - I

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

March 2005

Saeid Nooshabadi

saeid@unsw.edu.au

ELEC2041 lec9-logical-I.1

Saeid Nooshabadi

Overview

◦ **Bitwise Logical Operations**

- OR
- AND
- XOR

◦ **Shift Operations**

- Shift Left
- Shift Right
- Rotate
- Field Extraction

ELEC2041 lec9-logical-I.2

Saeid Nooshabadi

Review: Assembly Variables: Registers

- **Assembly Language uses registers as operands for data processing instructions**

Register No.	Register Name	
r0	→ a1	All register are identical in hardware, except for PC
r1	→ a2	
r2	→ a3	
r3	→ a4	
r4	→ v1	
r5	→ v2	PC is the program Counter; It always contains the address the instruction being fetched from memory
r6	→ v3	
r7	→ v4	
r8	→ v5	By Software convention we use different registers for different things
r9	→ v6	
r10	→ v7	
r11	→ ip	C Function Variables: v1 – v7 Scratch Variables: a1 – a4
r12	→ sp	
r13	→ lr	
r14	→ pc	
r15	→	

ELEC2041 lec9-logical-I.3

Saeid Nooshabadi

Review: ARM Instructions So far

add

sub

mov

ELEC2041 lec9-logical-I.4

Saeid Nooshabadi

Bitwise Operations (#1/2)

- Up until now, we've done arithmetic (add, sub, rsb) and data movement mov.
- All of these instructions view contents of register as a single quantity (such as a signed or unsigned integer)
- **New Perspective:** View contents of register as 32 bits rather than as a single 32-bit number

Bitwise Operations (#2/2)

- Since registers are composed of 32 bits, we may want to access individual bits rather than the whole.
- Introduce two new classes of instructions/Operations:
 - Logical Instructions
 - Shift Operators

Logical Operations

- Operations on less than full words
 - Fields of bits or individual bits
- Think of word as **32 bits** vs. 2's comp. integers or unsigned integers
- Need to extract bits from a word, or insert bits into a word
- Extracting via Shift instructions
 - C operators: << (shift left), >> (shift right)
- Inserting and inverting via And/OR/EOR instructions
 - C operators: & (bitwise AND), | (bitwise OR), ^ (bitwise EOR)

Logical Operators

- Operator Names:
 - and, bic, orr, eor:
- Operands:
 - Destination : Register
 - Operand #1: Register
 - Operand #2: Register, or Shifted registers, or an immediate

Example: and a1, v1, v2
 and a1, v1, v2, lsl #5
 and a1, v1, v2, lsl v3
 and a1, v1, #0x40
- ARM Logical Operators are all **bitwise**, meaning that bit 0 of the output is produced by the respective bit 0's of the inputs, bit 1 by the bit 1's, etc.

Logical AND Operator (#1/3)

° 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

Logical AND Operator (#2/3)

° The second bitstring in the example is called a **mask**. It is used to isolate the rightmost 8 bits of the first bitstring by masking out the rest of the string (e.g. setting it to all 0s).

° Thus, the and operator can be used to set certain portions of a bitstring to 0s, while leaving the rest alone.

• In particular, if the first bitstring in the above example were in a1, then the following instruction would mask the rightmost 8 bits a and clears leftmost 24 bits :

```
and    a1,a1,#0xFF
```

Logical AND Operator (#3/3)

° AND: bit-by-bit operation leaves a 1 in the result only if both bits of the operands are 1. For example, if registers v1 and v2 are

0000 0000 0000 0000 0000 1101 0000 0000_{two}

0000 0000 0000 0000 0011 1100 0000 0000_{two}

° After executing ARM instruction

```
and a1,v1, v2 ; a1 ← v1 & v2
```

° Value of register a1

0000 0000 0000 0000 0000 1100 0000 0000_{two}

Logical BIC (AND NOT) Operator (#1/3)

° 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

Logical BIC (AND NOT) Operator (#2/3)

- The second bitstring in the example is called a **mask**. It is used to isolate the leftmost 24 bits of the first bitstring by masking out the rest of the string (e.g. setting it to all 0s).
- Thus, the `bic` operator can be used to set certain portions of a bitstring to 0s, while leaving the rest alone.
 - In particular, if the first bitstring in the above example were in `a1`, then the following instruction would mask the leftmost 24 bits `a` and clears rightmost 8 bits:

```
bic    a1,a1,#0xFF
```

Logical BIC (AND NOT) Operator (#3/3)

- BIC: bit-by-bit operation leaves a 1 in the result only if bit of the first operand is 1 and the second operands 0. For example, if registers `v1` and `v2` are

```
0000 0000 0010 1100 0000 1101 0000 0000two
0000 0000 0000 1000 0011 1100 0000 0000two
```

- After executing ARM instruction

```
bic a1,v1, v2 ; a1 ← v1 & (not v2)
```

- Value of register `a1`

```
0000 0000 0010 0100 0000 0001 0000 0000two
```

Quiz # 2 Result

Title	N	% Correct Of:			Discrimination	Score	
		Whole Group	Upper 25%	Lower 25%		Mean	SD
ARM add instruction	146	93	100	81	0.61	93.2%	25.3
ARM register set	146	90	100	74	0.69	90.4%	29.5
ARM store instruction	146	78	100	42	0.64	78.8%	41.0
Data type	146	92	100	79	0.48	92.5%	26.5
Fetch-decode-execute cycle	146	93	100	81	0.61	93.2%	25.3
Meaning of register	146	89	100	70	0.58	89.0%	31.3
Overall Mean: 89.5%							

Computers in the News

Computers' environmental impact exposed

A new study by United Nations researchers calls for worldwide effort to reduce the environmental impact of computer equipment.

It says the high turnover of computer equipment is wasting large amounts of energy and causing unacceptable health risks.

The study says toxic substances such as flame retardants and heavy metals could pose health risks to workers making the equipment and to people living near landfill sites where used computers are dumped.

There is also concern that computer waste is often recycled in poorly managed facilities in the developing world.

The study welcomes new European rules requiring the electronics industry to take more responsibility for recycling old equipment safely but says their effectiveness is still in doubt.

It calls on governments to create greater incentives for people to upgrade computer hardware instead of replacing it.

--BBC

<http://www.abc.net.au/news/newsitems/s1061544.htm>

Computers in the News

Gagtes: Barracuda 7200.8 G00 GBytes

IIST names Seagate SATA hard drive the most significant product of the year 21/01/04

Model Number: ST3400832AS; Capacity: 400 GB; Speed: 7200 rpm;

Seek time: 8 ms avg; Latency: 4.16 ms; Interface : SATA 1.5Gb/s

- Capacity is the amount of data that the drive can store, after formatting. Most disc drive companies, including Seagate, calculate disc capacity based on the assumption that 1 megabyte = 1000 kilobytes and 1 gigabyte = 1000 megabytes.
- RPM is a measurement of how fast a hard disc's platters are spinning (in revolutions per minute). The faster the spin rate, the less time it takes for the drive to read or write a given amount of data.
- Seek time is an average of how long a drive takes to move the read/write heads to a particular track on the disc. It includes controller overhead but does not include drive latency.
- The drive interface is the "language" or protocol a drive uses to communicate with a host computer or network. The three main types of drive interfaces are ATA (IDE), SCSI, and Fibre Channel. The ATA and SCSI interfaces have evolved to include many sub-types, which may or may not be backwardly compatible.

ELEC2041 lec9-logical-1.17

Saeid Nooshabadi

Logical OR Operator (#1/2)

- OR: Similarly, note that **oring** a bit with 1 produces a 1 at the output while **oring** a bit with 0 produces the original bit.
- This can be used to force certain bits of a string to 1s.
 - For example, if a1 contains 0x12345678, then after this instruction:
`orr a1, a1, #0xFF`
 - ... a1 contains 0x123456FF (e.g. the high-order 24 bits are untouched, while the low-order 8 bits are forced to 1s).

ELEC2041 lec9-logical-1.18

Saeid Nooshabadi

Logical OR Operator (#2/2)

- OR: bit-by-bit operation leaves a 1 in the result if **either** bit of the operands is 1. For example, if registers v1 and v2

```
0000 0000 0000 0000 0000 1101 0000 0000two
0000 0000 0000 0000 0011 1100 0000 0000two
```

- After executing ARM instruction

```
ORR a1, v1, v2 ; a1 ← v1 | v2
```

- Value of register a1

```
0000 0000 0000 0000 0011 1101 0000 0000two
```

ELEC2041 lec9-logical-1.19

Saeid Nooshabadi

Logical XOR Operator (#1/2)

- EOR: **E**xoring a bit with 1 produces its complement at the output while **E**xoring a bit with 0 produces the original bit.
- This can be used to force certain bits of a string to invert.
 - For example, if a1 contains 0x12345678, then after this instruction:
`eor a1, a1, #0xFF`
 - ... a1 contains 0x12345687 (e.g. the high-order 24 bits are untouched, while the low-order 8 bits are forced to invert).

ELEC2041 lec9-logical-1.20

Saeid Nooshabadi

Logical XOR Operator (#2/2)

- EOR: bit-by-bit operation leaves a 1 in the result if if bits of the operands are different. For example, if registers `v1` and `v2`

```
0000 0000 0000 0000 0000 1101 0000 0000two
0000 0000 0000 0000 0011 1100 0000 0000two
```

- After executing ARM instruction

```
eor a1,v1,v2 ; a1 ← v1 ^ v2
```

- Value of register `a1`

```
0000 0000 0000 0000 0011 0001 0000 0000two
```

Shift Operations (#1/3)

- Move (shift) all the bits in a word to the left or right by a number of bits, filling the emptied bits with **0s**.

- Example: shift **right** by 8 bits

```
1001 0010 0011 0100 0101 0110 0111 1000
```

```
0000 0000 1001 0010 0011 0100 0101 0110
```

- Example: shift **left** by 8 bits

```
1001 0010 0011 0100 0101 0110 0111 1000
```

```
0011 0100 0101 0110 0111 1000 0000 0000
```

Shift Operations (#2/3)

- Move (shift) all the bits in a word to the right by a number of bits, filling the emptied bits with the **sign bits**.

- Example: Arithmetic shift **right** by 8 bits

```
1001 0010 0011 0100 0101 0110 0111 1000
```

```
1111 1111 1001 0010 0011 0100 0101 0110
```

```
0001 0010 0011 0100 0101 0110 0111 1000
```

```
0000 0000 0001 0010 0011 0100 0101 0110
```

Shift Operations (#3/3)

- Move (shift) all the bits in a word to the right by a number of bits, filling the emptied bits with the bits falling of the **right**.

- Example: rotate **right** by 8 bits

```
1001 0010 0011 0100 0101 0110 0111 1000
```

```
0111 1000 1001 0010 0011 0100 0101 0110
```

ARM Shift Instructions

- ARM does not have separate shift instruction.
- Shifting operations is embedded in almost all other data processing instructions.
- Pure Shift operation can be obtained via the shifted version of `mov` instruction.

Data Processing Instruction with Shift Feature

Syntax:

Example:

1 2,3,4,5 6

• where

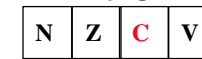
- operation `;a1 ← v1 +(v3 << 8 bits)`
- register that will receive value
- first operand (register) `add a1, v1,v3, lsl v4`
- second operand (register) `;a1 ← v1 +(v3 << v4 bits)`
- shift operation on the second operand
- shift value (immediate/register)

ELEC2041 lec9-logical-I.25

Saeid Nooshabadi

ARM Shift Variants (#1/4)

- `lsl` (logical shift left): shifts left and fills emptied bits with 0s



Destination

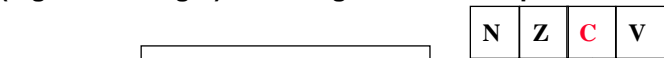
0

`mov a1, v1, lsl #8 ;a1 ← v1 << 8 bits`

`mov a1, v1, lsl v2 ;a1 ← v1 << v2 bits`

shift amount between 0 to 31

- `lsr` (logical shift right): shifts right and fills emptied bits with 0s



Destination

0

`mov a1, v1, lsr #8 ;a1 ← v1 >> 8 bits`

`mov a1, v1, lsr v2 ;a1 ← v1 >> v2 bits`

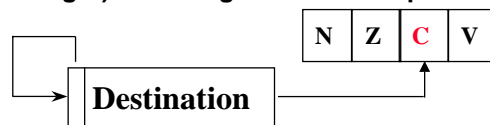
shift amount between 0 to 31

ELEC2041 lec9-logical-I.26

Saeid Nooshabadi

ARM Shift Variants (#2/4)

- `asr` (arithmetic shift right): shifts right and fills emptied bits by sign extending



Sign bit shifted in

`mov a1, v1, asr #8 ;a1 ← v1 >> 8 bits`

`;a1[31:24]=v1[31]`

`mov a1, v1, asr v2 ;a1 ← v1 >> v2 bits`

`;a1[31:24]=v1[31]`

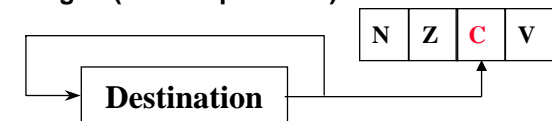
shift amount between 0 to 31

ELEC2041 lec9-logical-I.27

Saeid Nooshabadi

ARM Shift Variants (#3/4)

- `ror` (rotate right): shifts right and fills emptied bits by bits falling of the right. (bits wrap around)



Destination

`mov a1, v1, ror #8`

`;a1 ← v1 >> 8 bits`

`;a1[31:24]← v1[7:0]`

`mov a1, v1, ror v2`

`;a1 ← v1 >> v2 bits`

`;a1[31:(31-v2)] ← v1[v2:0]`

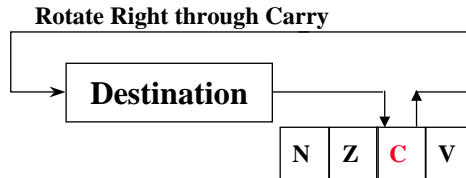
Rotate amount between 1 to 31

ELEC2041 lec9-logical-I.28

Saeid Nooshabadi

ARM Shift Variants (#4/4)

4. **rrx** (rotate right through carry): This operation uses the CPSR C flag as a 33rd bit for rotation. (wrap around through carry)



```
mov a1, v1, rrx    ;a1 ← v1 >> 1 bit
                  ;a1[31] ← CF
                  ;CF ← v[0]
```

- Only Rotation by one bit is allowed
- Encoded as `ror #0`.

Isolation with Shift Instructions (#1/2)

- ° Suppose we want to isolate byte 0 (rightmost 8 bits) of a word in `a0`. Simply use:

```
and a0, a0, #0xFF
```

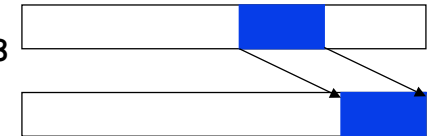


- ° Suppose we want to isolate byte 1 (bit 15 to bit 8) of a word in `a0`. We can use:

```
and a0, a0, #0xFF00
```

but then we still need to shift to the right by 8 bits...

```
mov a0, a0, lsr #8
```



Isolation with Shift Instructions (#2/2)

- ° Instead, we can also use:

```
mov a0, a0, lsl #16
mov a0, a0, lsr #24
```

0001 0010 0011 0100 **0101 0110 0111 1000**

0101 0110 **0111 1000** 0000 0000 0000 0000

0000 0000 0000 0000 0000 0000 **0101 0110**

ELEC2041 Reading Materials (Week #3)

- ° Week #3: 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:**

- and

- bic

- orr

- Eor

- **Data Processing Instructions with shift and rotate:**