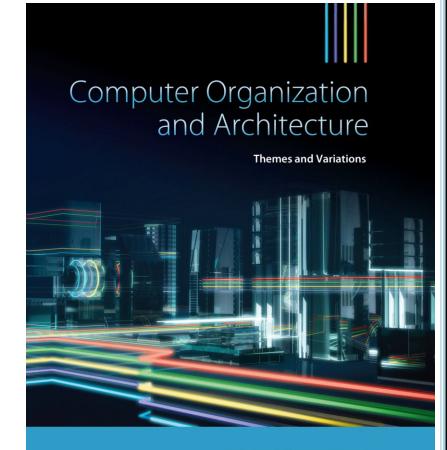
## Part 7

### CHAPTER 3

# Architecture and Organization



Alan Clements

1

These slides are being provided with permission from the copyright for CS2208 use only. The slides must not be reproduced or provided to anyone outside of the class.

All download copies of the slides are for personal use only. Students must destroy these copies within 30 days after receipt of final course evaluations.



### **Instruction Encoding** An Insight into the ARM's Architecture

- ☐ The branch instruction (Figure 3.41) has
  - an 8-bit op-code
  - a 24-bit <u>signed</u> program-counter relative <u>offset</u> (<u>word</u> address offset).
- □ Converting the 24-bit <u>word</u> offset to real <u>byte</u> address:

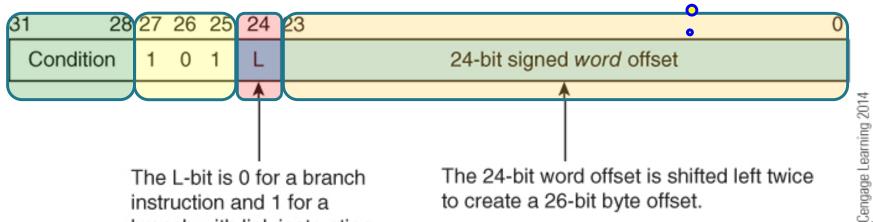
branch with link instruction.

- shift left twice the 24-bit *word* offset to convert the *word-offset* address to a **byte-offset** address, Do not forget the pipelining effect
- **sign-extended** to 32 bits,
- added it to the current value of the program counter • (the result is in the range  $PC \pm 32$  MBytes).

Basically, it is the number of instructions away from the current location (after considering the pipelining effect.

#### **FIGURE 3.41**

Encoding ARM's branch and branch-with-link instructions



89

l R. El-Sakka

**Execute on condition** 

Not equal (i.e., not zero)

Unsigned higher or same

Unsigned lower or same

Equal (i.e., zero)

Unsigned lower

Positive or zero

Unsigned higher

Greater or equal

Negative

Overflow

Less than

Greater than

Less than or equal

Always (default)

Never (reserved)

No overflow

ARM's Conditional Execution and Branch Control Mnemonics

**Branch on Flag Status** 

Z set

C set

N set

V set

Z clear

C clear

N clear

V clear

C set and Z clear

N set and V set, or

N set and V clear, or

N clear and V set

N clear and V set

Z clear, and either N set and

Z set, or N set and V clear, or

V set, or N clear and V clear

N clear and V clear

C clear or Z set

#### **Instruction Encoding**

#### ARM Instruction: : Loop B Loop

Condition = 1110 (always - unconditional)

$$L = 0 \text{ (Not } BL)$$

#### Loop B Loop

• • • •

PC location

Branch backward · · · ·

instructions

PC location + pipelining

#### Word-offset = -2

	_	_	_		_		_	_	_	_				_		_	_	_	_	_	_		•	_	_	_	_	_	_	_	_	0
Ī	1	1	1	0	1	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0

FIGURE 3.41

TABLE 3.2

Mnemonic

EQ

NE

CS

CC

MI

PL

VS

VC

HI

LS

GE

LT

GT

LE

AL

NV

Encoding

0000

0001

0010

0011

0100

0101

0110

0111

1000

1001

1010

1011

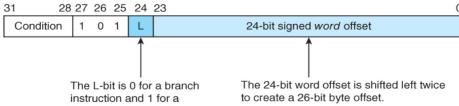
1100

1101

1110

1111

Encoding ARM's branch and branch-with-link instructions



vvoid-onset – -2

= 2\_1111 1111 1111 1111 1110

**OxEAFFFFE** 

The L-bit is 0 for a branch instruction and 1 for a society to create a 26-bit byte offset.

The 24-bit word offset is shifted left twice instruction and 1 for a society to create a 26-bit byte offset.

This slide is modified from the original slide by the author **A. Clements** and used with permission. New content added and copyrighted by © **Mahmoud R. El-Sakka.** 

#### Clements

#### **Instruction Decoding**

Machine Language Instruction: **0x1AFFFFFD** 

1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	Ō	9	8	7	6	5	4	3	2	0	0
0	0	0	1	1	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1

Bits number 25 26 27 = 101

L = 0 (Not BL)Condition = 0001 (BNE)

Word offset= 0xFFFFFD = -3

BNE Target

backward
instructions

Branch backward

Target

PC location + pipelining

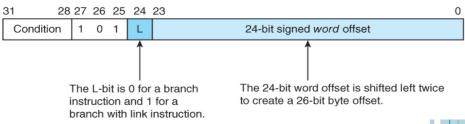
PC location

instructions

Encoding	Mnemonic	Branch on Flag Status	Execute on condition
0000	EQ	Z set	Equal (i.e., zero)
0001	NE	Z clear	Not equal (i.e., not zero)
0010	CS	C set	Unsigned higher or same
0011	CC	C clear	Unsigned lower
0100	MI	N set	Negative
0101	PL	N clear	Positive or zero
0110	VS	V set	Overflow
0111	VC	V clear	No overflow
1000	HI	C set and Z clear	Unsigned higher
1001	LS	C clear or Z set	Unsigned lower or same
1010	GE	N set and V set, or N clear and V clear	Greater or equal
1011	LT	N set and V clear, or N clear and V set	Less than
1100	GT	Z clear, and either N set and V set, or N clear and V clear	Greater than
1101	LE	Z set, or N set and V clear, or N clear and V set	Less than or equal
1110	AL		Always (default)
1111	NV		Never (reserved)

ARM's Conditional Execution and Branch Control Mnemonics

FIGURE 3.41 Encoding ARM's branch and branch-with-link instructions



This slide is modified from the original slide by the author A. Clements and used with permission. New content added and copyrighted by © Mahmoud R. El-Sakka.

# Instruction Encoding An Insight into the ARM's Architecture

□ Figure 3.26 illustrates the structure of the ARM's <u>data processing</u> instructions and demonstrates how bit 25 is used to control the nature of the second source operand.

Shift to 00 = 10

to one.

Shift type 00 = logical left 01 = logical right B / BL instructions To reduce the course 10 = Arithmetic right workload. Multiplication 6 Encoding the ARM's data processing instructions 11 = rotate right encoding/decoding will not be included in this 28 27 26 25 21 20 19 16 15 24 12 11 course. Op-Code Operand 2 Condition 0 0 r<sub>source1</sub> r<sub>destination</sub> 11 0000 = ANDinstructions Immediate shift 0001 = EOR Shift length Shift type 0 r<sub>source2</sub> are belong to 0010 = SUBi.e., static shift the data 0011 = RSBprocessing 0100 = ADD8 instructions 0101 = ADCShift specified by register 0110 = SBCShift type 1 r<sub>Shift length</sub> 0 i.e., dynamic shift 0111 = RSCMultiplication 1000 = TST instructions are 1001 = TEQ encoded by Literal operand 1010 = CMP212 (i.e., 4096) different values setting 1011 = CMN bits 25, 26, 27 1100 = ORR 92 to zero 1101 = MOV The <u>rotate right through carry</u> (RRX) is and 1110 = BIC encoded as rotate right with zero shift. bits 4 and 7 1111 = MNV

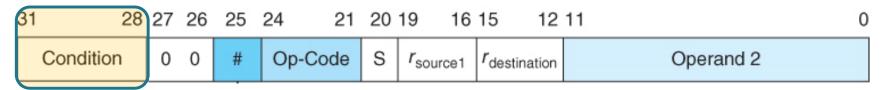
from the original slide by the author A. Clements and used with permission. New content added and copyrighted by © Mahmoud R. El-Sakka.

- ☐ One of ARM's most unusual features is that each instruction can be conditionally executed
  - o associating an instruction with a logical condition.
    - If the stated condition is true, the instruction is executed.
    - Otherwise, it is bypassed (*squashed*).
- Assembly language programmers indicate the conditional execution mode by appending the appropriate condition to a mnemonic (*mnemonic* is a text abbreviation for an operation code).
- $\Box$  for example,

ADD**EQ** 
$$r1, r2, r3$$
 ; IF Z = 1 THEN [r1] <- [r2] + [r3]

specifies that the addition is performed only if the Z-bit is set because a previous result was zero.

FIGURE 3.26 Encoding the ARM's data processing instructions



93

- ☐ There is nothing to stop you combining conditional execution and shifting because the branch and shift fields of an instruction are independent.
- ☐ You can write

ADD**CC r1**, r2, r3, LSL r4 ; IF C=0 THEN 
$$[r1] < -[r2] + [r3] \times 2^{[r4]}$$

☐ The S can be combined with the *conditional execution*, for example,

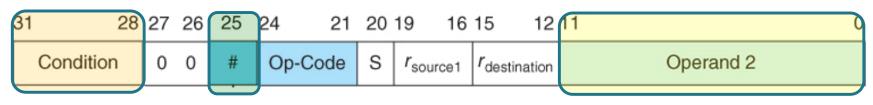
Or

ADD**EQS r1**, r2, r3

ADD**SEQ r1**, r2, r3

Both instructions are identical

FIGURE 3.26 Encoding the ARM's data processing instructions



94

- □ ARM's conditional execution mode makes it easy to implement conditional operations in a high-level language.
- □ Consider the following fragment of C code. if (P == Q) X = P - Y;
- ☐ If we assume that r1 contains P,
  r2 contains Q,
  r3 contains X, and
  r4 contains Y, then we can write

```
CMP r1,r2 ; compare P == Q

SUBEQ r3,r1,r4 ; if (P == Q) then r3 = r1 - r4
```

- □ Notice how simple this operation is implemented without using a branch instruction
  - o In this case the subtraction is squashed if the comparison is false

□ Now consider a more complicated example of a C construct with a compound predicate:

```
if ((a == b) & (c == d)) e++;
```

☐ We can write

```
CMP r0,r1 ; compare a == b

CMPEQ r2,r3 ; if a == b then test c == d

ADDEQ \mathbf{r4},r4,#1 ; if a == b AND c == d THEN increment e
```

- ☐ The first line, CMP r0, r1, compares a and b.
- The next line, CMPEQ r2, r3, executes a conditional comparison only if the result of the first line was true (i.e., a == b). (*short circuit*)
- The third line, ADDEQ **r4**, r4, #1, is executed only if the previous line was true (i.e., c == d) to implement the e++.

- ☐ You can also handle some testing with multiple conditions.
- ☐ Consider: if (a == b) e = e + 4; | CMP r0, r1

We can use conditional execution to implement this as

```
;compare a == b
if (a < b) e = e + 7; ADDEQ r4, r4, #4 ; if a == b then e = e + c
if (a > b) e = e + 12; ADDLT r4, r4, #7 ; if a < b then e = e + 7
                  ADDGT r4, r4, #12; if a > b then e = e + 12
```

☐ Without the conditional execution, we can implement it as follow:

```
CMP r0,r1 ; compare a == b
         BNE NotEqual
         ADD r4, r4, \#4; if a == b then e = e + 4
Equal
         B AfterAll
NotEqual BLT LessThan
         ADD \mathbf{r4}, r4, #12; if a > b then e = e + 12
             AfterAll
LessThan ADD \mathbf{r4}, r4, #7 ; if a < b then e = e + 7
AfterAll ...
```

CMP a,b ADD 4 97 ADD 7

#### **Instruction Encoding**

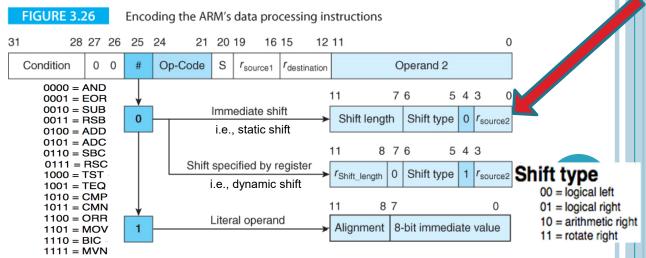
ARM Instruction:

ADD **r0**, r1, r2

Condition = 1110 (always - unconditional)

Op-Code = 0100 (i.e., $ADD$ )	TABLE 3.2	ARM's C	Conditional Execution and Branch C	ontrol Mnemonics
$C = O \left( n \circ t \land DDC \right)$	Encoding	Mnemonic	Branch on Flag Status	Execute on condition
$S = 0 \pmod{ADDS}$	0000	EQ	Z set	Equal (i.e., zero)
= 0000 (doctination or one)	0001	NE	Z clear	Not equal (i.e., not zero)
$r_{destination} = 0000 $ (destination operand)	0010	CS	C set	Unsigned higher or same
	0011	CC	C clear	Unsigned lower
$r_{source1} = 0001 $ (first operand)	0100	MI	N set	Negative
	0101	PL	N clear	Positive or zero
# = 0 (second operand not a constant)	0110	VS	V set	Overflow
` -	0111	VC	V clear	No overflow
Operand 2 (bit number $4 = 0$ )	1000	HI	C set and Z clear	Unsigned higher
	1001 1010	LS GE	C clear or Z set N set and V set, or	Unsigned lower or same
$r_{\text{source2}} = 0010$	1010	GE	N clear and V clear	Greater or equal
source2	1011	LT	N set and V clear, or	Less than
shift type = 00 (logical left)	1011	LI	N clear and V set	Loss than
Sillic type - 00 (logical lett)	1100	GT	Z clear, and either N set and	Greater than
ahift longth - 00000			V set, or N clear and V clear	
shift length = 00000	1101	LE	Z set, or N set and V clear, or	Less than or equal
<b>3 2 2</b> 2 2 2 2 <b>2 2 2 2 1</b> 1 1 1 1 <b>1 1 1</b> 1 1 1 0 0 <b>0 0 0 0</b> 0 0 0 0			N clear and V set	100
<b>0 9 8</b> 7 6 5 4 <b>3 2 1 0</b> 9 8 7 6 <b>5 4 3 2</b> 1 0 9 8 7 6 <b>5 4</b> 3 2 1 0	1110	AL		Always (default)
$ \begin{array}{ c c c c c c c c c c c c c c c c c c c$	1111	NV		Never (reserved)

0xE0810002



### **Instruction Encoding**

ARM Instruction:

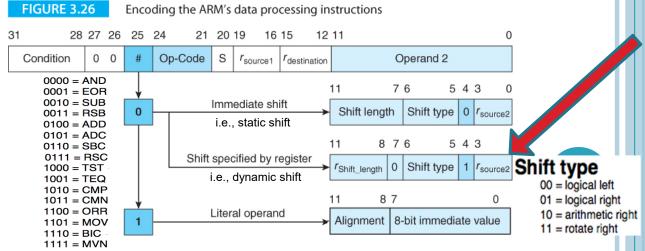
ADD ro, r

**r0**, r1, r2, LSR r3

Condition = 1110 (always - unconditional)

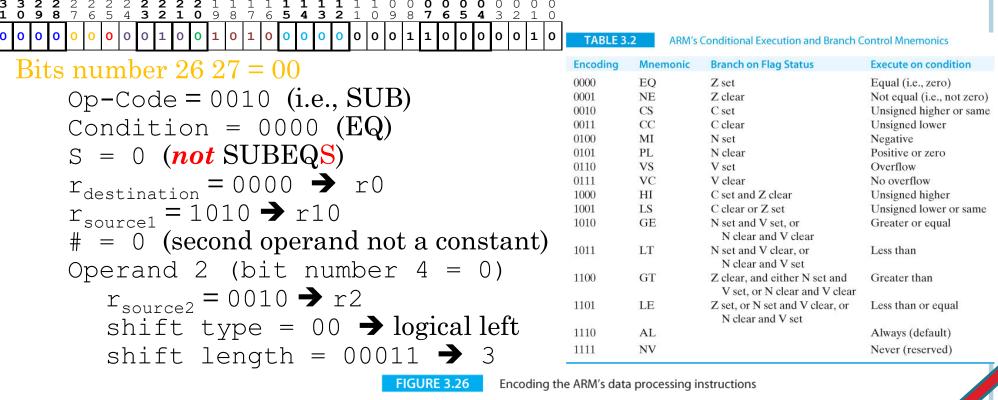
Op-Code = 0100  (i.e., $ADD$ )	TABLE 3.	ARM's	Conditional Execution and Branch C	ontrol Mnemonics
$S = 0 \pmod{ADDS}$	Encoding	Mnemonic	Branch on Flag Status	Execute on condition
$S = 0 \pmod{ADDS}$	0000	EQ	Z set	Equal (i.e., zero)
= 0.000  (doctination one rand)	0001	NE	Z clear	Not equal (i.e., not zero)
$r_{destination} = 0000 $ (destination operand)	0010	CS	C set	Unsigned higher or same
	0011	CC	C clear	Unsigned lower
$r_{source1} = 0001 $ (first operand)	0100 0101	MI PL	N set N clear	Negative Positive or zero
	0101	VS	V set	Overflow
# = 0 (second operand not a constant)	0111	VC	V clear	No overflow
One mand $\frac{1}{2}$ (bit numbers $\frac{1}{2}$ = 1)	1000	HI	C set and Z clear	Unsigned higher
Operand 2 (bit number $4 = 1$ )	1001	LS	C clear or Z set	Unsigned lower or same
<b>-</b> 0010	1010	GE	N set and V set, or	Greater or equal
$r_{\text{source}2} = 0010$	2727-2710-12		N clear and V clear	HOLE THE ROY MARKET HE
	1011	LT	N set and V clear, or	Less than
shift type = 01 (logical right)	1100	GT	N clear and V set Z clear, and either N set and	Greater than
-l-'	1100	GI	V set, or N clear and V clear	Greater than
shift length = r3 = 0011	1101	LE	Z set, or N set and V clear, or	Less than or equal
3 3 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1			N clear and V set	1
1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0	1110	AL		Always (default)
1 1 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 1 1 0 0 0 1 0	1111	NV		Never (reserved)

0xE0810332

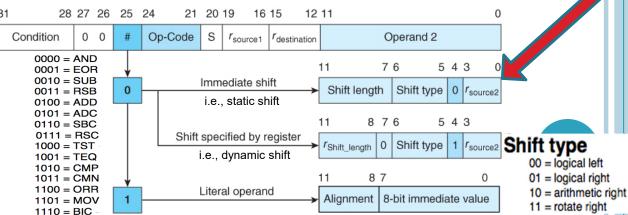


#### **Instruction Decoding**

Machine Language Instruction: 0x004A0182

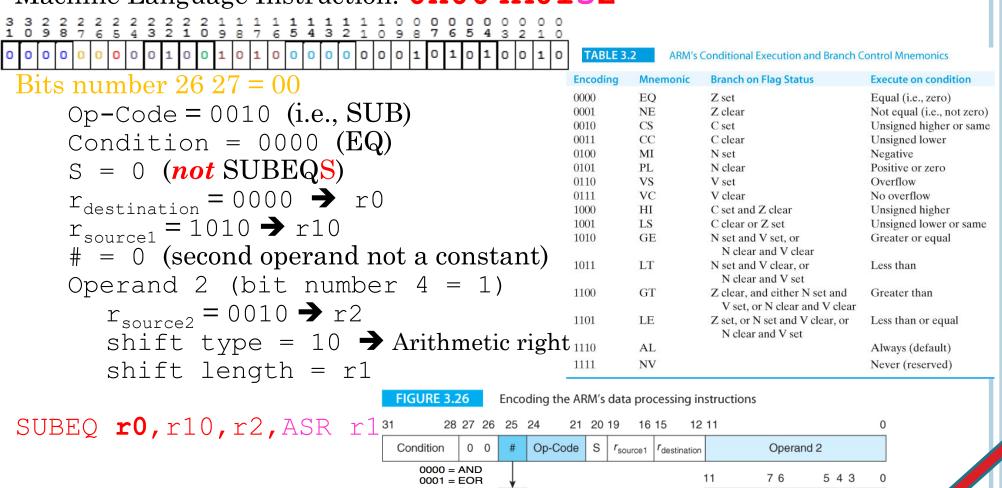


SUBEQ **r0**, r10, r2, LSL#3



#### **Instruction Decoding**

Machine Language Instruction: 0x004A0152



0010 = SUBImmediate shift Shift length | Shift type | 0 | r<sub>source2</sub> 0011 = RSBi.e., static shift 0100 = ADD0101 = ADC8 7 6 5 4 3 0110 = SBC0111 = RSCShift specified by register r<sub>Shift\_length</sub> | 0 | Shift type | 1 | r<sub>source2</sub> | Shift type 1000 = TSTi.e., dynamic shift 1001 = TEQ 00 = logical left 1010 = CMP01 = logical right 1011 = CMN 1100 = ORR Literal operand 10 = arithmetic right Alignment 8-bit immediate value 1101 = MOV 11 = rotate right 1110 = BIC

00 = logical left

01 = logical right

11 = rotate right

10 = arithmetic right

### **Instruction Encoding**

#### ARM Instruction:

CMPGT r3, r5

Condition = 1100 (GT)

Op-Code = 1010 (i.e., CMP)

S = 1 (update flags)

 $r_{destination} = 0000 (must be zeros)$ 

r<sub>source1</sub> = 0011 (first operand)

# = 0 (second operand not a constant)

Operand 2 (bit number 4 = 0)

 $r_{\text{source2}} = 0101$ 

shift type = 00 (logical left)

shift length = 00000

0 1 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 1 0 1

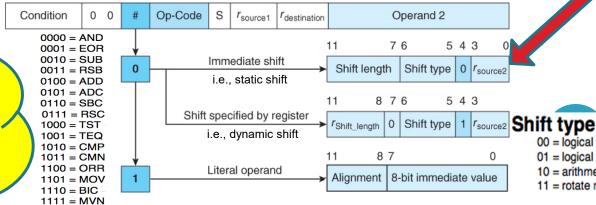
TABLE 3.2	ARM's Conditional Execution and Branch Control Mnemonics

	Encoding	Mnemonic	Branch on Flag Status	Execute on condition
	0000	EQ	Z set	Equal (i.e., zero)
	0001	NE	Z clear	Not equal (i.e., not zero
	0010	CS	C set	Unsigned higher or same
	0011	CC	C clear	Unsigned lower
	0100	MI	N set	Negative
	0101	PL	N clear	Positive or zero
	0110	VS	V set	Overflow
	0111	VC	V clear	No overflow
	1000	HI	C set and Z clear	Unsigned higher
	1001	LS	C clear or Z set	Unsigned lower or same
	1010	GE	N set and V set, or N clear and V clear	Greater or equal
	1011	LT	N set and V clear, or N clear and V set	Less than
	1100	GT	Z clear, and either N set and V set, or N clear and V clear	Greater than
)	1101	LE	Z set, or N set and V clear, or N clear and V set	Less than or equal
)	1110	AL		Always (default)
	1111	NV		Never (reserved)

#### 0xC1530005

In all test-and-compare instructions, i.e., TST, TEQ, CMP, and CMN, the destination register field MUST BE encoded as 0000

**FIGURE 3.26** Encoding the ARM's data processing instructions 21 20 19 16 15



A. Clements and used with permission. New content added and copyrighted by © Mahmoud R. El-Sakka.

### **Instruction Encoding**

ARM Instruction:

MOV PC, LR

Condition = 1110 (always - unconditional)

Op-Code = 1101 (i.e., $MOV$ )
$S = 0 \pmod{MOVS}$
$r_{destination} = 1111 (PC)$
$r_{source1} = 0000 $ (must be zeros)
# = 0 (second operand not a constant)
Operand 2 (bit number $4 = 0$ )

erand 2 (bit number $4 = 0$ )
$r_{\text{source2}} = 1110$
shift type = 00 (logical left)
shift length = 00000

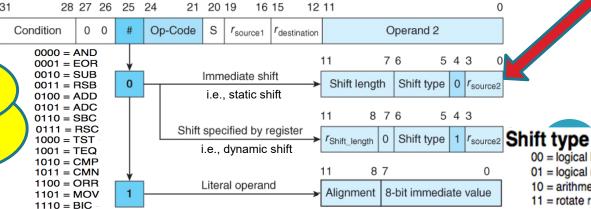
								_	_	_	_		- 1 1	. 9	<u> </u>				_	,	,	U	0									1101		LE	Z set, or N
	3																															1110		AL	N clear
1	1	1	0	0	0	0	1	1	0	1	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	0	1111		NV	
_	•						•	-				•					•		•			F	GL	IRE	3.2	26		Er	ico	dino	th:	e ARM's	data	process	ing instructions

TABLE 3.2 ARM's Conditional Execution and Branch Control Mnemor
---

	Encoding	Mnemonic	Branch on Flag Status	Execute on condition
	0000	EQ	Zset	Equal (i.e., zero)
	0001	NE	Z clear	Not equal (i.e., not zero)
	0010	CS	C set	Unsigned higher or same
	0011	CC	C clear	Unsigned lower
	0100	MI	N set	Negative
	0101	PL	N clear	Positive or zero
	0110	VS	V set	Overflow
	0111	VC	V clear	No overflow
	1000	HI	C set and Z clear	Unsigned higher
	1001	LS	C clear or Z set	Unsigned lower or same
	1010	GE	N set and V set, or N clear and V clear	Greater or equal
	1011	LT	N set and V clear, or N clear and V set	Less than
	1100	GT	Z clear, and either N set and V set, or N clear and V clear	Greater than
)	1101	LE	Z set, or N set and V clear, or N clear and V set	Less than or equal
Ó	1110	AL		Always (default)
)	1111	NV		Never (reserved)

#### 0xE1A0F00E

In all moving instructions, i.e., MOV, and MVN, the source, register field MUST BE encoded as 0000



11 = rotate right

00 = logical left

01 = logical right

10 = arithmetic right

iginal slide by the author A. Clements and used with permission. New content added and copyrighted by © Mahmoud R. El-Sakka. This slide is modified from

1111 = MVN

#### Instruction Decoding

Machine Language Instruction: **0xE1B01061** 

7 6 **5 4 3 2** 1 0 9 8 **7 6 5 4** 3 2 1 0 TABLE 3.2 ARM's Conditional Execution and Branch Control Mnemonics 0 0 0 0 00001 Encoding Mnemonic **Branch on Flag Status Execute on condition** 0000 EQ Z set Bits number 26.27 = 00Equal (i.e., zero) 0001 NE Z clear Not equal (i.e., not zero) Op-Code = 1101 (i.e., MOV) CS 0010 C set Unsigned higher or same 0011 CC C clear Unsigned lower Condition = 1110 (Always) 0100 MI N set Negative PL N clear Positive or zero 0101 S = 1 (i.e., MOVS) 0110 VS V set Overflow VC V clear No overflow 0111  $r_{destination} = 0001 \rightarrow r1$ 1000 HI C set and Z clear Unsigned higher LS 1001 C clear or Z set Unsigned lower or same GE N set and V set, or 1010 Greater or equal  $r_{source1} = 0000$  (must be zeros) N clear and V clear 1011 LT N set and V clear, or Less than # = 0 (second operand not a constant) N clear and V set 1100 GT Z clear, and either N set and Greater than Operand 2 (bit number 4 = 0) V set, or N clear and V clear 1101 LE Z set, or N set and V clear, or Less than or equal  $r_{\text{source2}} = 0001 \rightarrow r1$ N clear and V set 1110 ALAlways (default) shift type = 11 → rotate right 1111 NV Never (reserved) shift length =  $00000 \rightarrow 00$ FIGURE 3.26 Encoding the ARM's data processing instructions rotate right and 28 27 26 25 24 21 20 19 12 11 16 15 shift length =  $00 \rightarrow RRX$ Condition Op-Code Operand 2 r<sub>source1</sub> r<sub>destination</sub> 0000 = AND0001 = EOR11 7 6 0010 = SUBImmediate shift 0011 = RSBMOVS **r1**, r1, RRX Shift type 0 r<sub>source2</sub> Shift length 0100 = ADDi.e., static shift 0101 = ADC0110 = SBC8 7 6 5 4 3 0111 = RSC1000 = TSTShift specified by register Shift type 0 Shift type 1 r<sub>source2</sub> r<sub>Shift</sub> length 1001 = TEQ i.e., dynamic shift 00 = logical left 1010 = CMP01 = logical right 1011 = CMN 8 7 1100 = ORR 10 = arithmetic right 1101 = MOV Literal operand Alignment 8-bit immediate value 11 = rotate right

This slide is modified from the original slide by the author **A. Clements** and used with permission. New content added and copyrighted by © **Mahmoud R. El-Sakka.** 

1110 = BIC

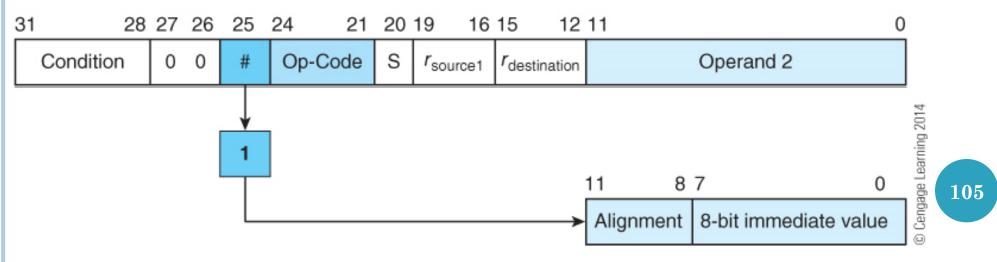
### **Handling Literals**

☐ In ARM, *operand 2* can be a literal.

```
ADD \mathbf{r0}, r1, #7 ; adds 7 to r1 and puts the result in r0. MOV \mathbf{r3}, #25 ; moves 25 into r3.
```

- □ What is the range of such literals?
  - *Operand 2* is a 12-bits field, i.e., it can encode **4096** different values
    - ARM encodes these 12-bits as a value from 0 to 255 (i.e., 8-bits) to be rotated (aligned) according to the value of the other bits (i.e., 4-bits)
- ☐ Figure 3.28 illustrate the format of ARM's instructions with a literal operand.

FIGURE 3.28 Diagram of ARM's literal operand encoding



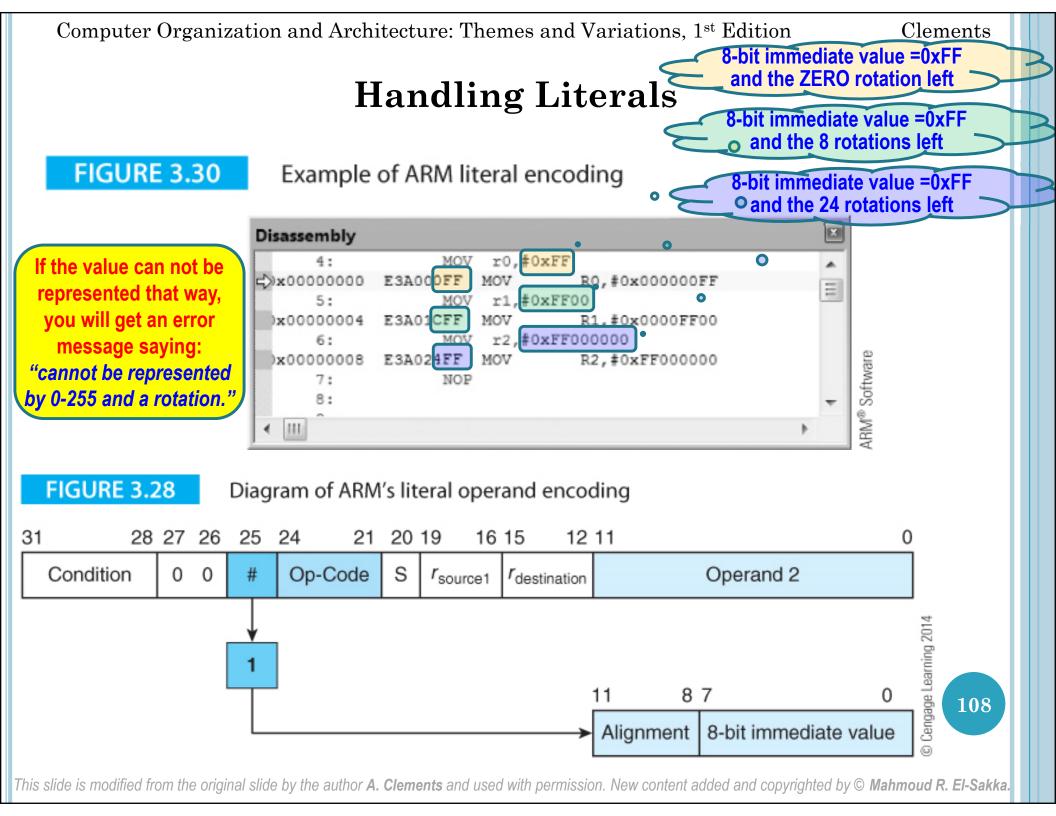
Architecture: Themes and Variations, 1st Edition

You need to know how to decode and encode literals

Clements

### Handling Literals

Encoded literal	Scale value	#of rotations right =2 × Scale value	# of rotations left =32 - 2 × Scale value	Decoded literal
0000 mnop wxyz	0	0	(32) <sub>10</sub>	0000 0000 0000 0000 0000 0000 <b>mnop wxyz</b>
1111 mnop wxyz	(15) <sub>10</sub>	(30) 10	2	0000 0000 0000 0000 0000 00 <b>mn opwx yz</b> 00
1110 mnop wxyz	(14) <sub>10</sub>	(28) 10	4	0000 0000 0000 0000 0000 <b>mnop wxyz</b> 0000
1101 mnop wxyz	(13) <sub>10</sub>	(26) 10	6	0000 0000 0000 0000 00 <b>mn opwx yz</b> 00 0000
1100 mnop wxyz	(12) <sub>10</sub>	(24) 10	8	0000 0000 0000 0000 <b>mnop wxyz</b> 0000 0000
1011 mnop wxyz	(11) <sub>10</sub>	(22) 10	(10) <sub>10</sub>	0000 0000 0000 00 <b>mn opwx yz</b> 00 0000 0000
1010 mnop wxyz	(10) <sub>10</sub>	(20) 10	(12) <sub>10</sub>	0000 0000 0000 <b>mnop wxyz</b> 0000 0000 0000
1001 mnop wxyz	9	(18) 10	(14) <sub>10</sub>	0000 0000 00 <b>mn opwx yz</b> 00 0000 0000 0000
1000 mnop wxyz	8	(16) <sub>10</sub>	(16) <sub>10</sub>	0000 0000 mnop wxyz 0000 0000 0000 0000
0111 mnop wxyz	7	(14) 10	(18) <sub>10</sub>	0000 00 <b>mn opwx yz</b> 00 0000 0000 0000 0000
0110 mnop wxyz	6	(12) 10	(20) <sub>10</sub>	0000 mnop wxyz 0000 0000 0000 0000 0000
0101 mnop wxyz	5	(10) 10	(22) <sub>10</sub>	00 <b>mn opwx yz</b> 00 0000 0000 0000 0000 0000
0100 mnop wxyz	4	8	(24) <sub>10</sub>	mnop wxyz 0000 0000 0000 0000 0000 0000
0011 mnop wxyz	3	6	(26) <sub>10</sub>	opwx yz00 0000 0000 0000 0000 0000 00mn
0010 mnop wxyz	2	4	(28) <sub>10</sub>	wxyz 0000 0000 0000 0000 0000 0000 mnop
0001 mnop wxyz	1	2	(30) <sub>10</sub>	yz00 0000 0000 0000 0000 00mn opwx



#### Handling Literals Example

If the literal value is 0x128, e.g., as in ADD R0,R1,#0x128 what is the 0-to-255 value (from 0 to 0xFF) and the align code (from 0 to 0xF)?

Convert the literal into 32-bit binary value.  $0x128 \rightarrow 0x0000\ 0000\ 0000\ 0000\ 0001\ 0010\ 1000$ 

Identify the <u>shortest</u> pattern that include all the 1's in a circular fashion  $0x128 \rightarrow 0x0000\ 0000\ 0000\ 0000\ 0001\ 0010\ 1000$ 

If the length of this pattern (*in a circular fashion*) is <u>less than or equal 8</u>, it means that you will be able to encode the number as a value from 0 to 255 and an align code.

If the length of this pattern is less than 8, augment the pattern by zeros to make it 8 bits in total (the 0-to-255 value). Make sure that the number of the other zeros to the left and to the right are even, OR the length of left and right pattern are even

 $0x128 \rightarrow 0x0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 1000$ 

the number of the <u>other zeros</u> to the left and to the right are even

The 0-to-255 value is 01 0010  $10_2 \rightarrow 0x4A$ 

A value from 0-to-255 needs to be rotated left 2 times to become 0x128, which is equivalent to 30 times rotation to the right; hence the align code is  $30 \div 2 = 15 = 0xF$  Operand 2 is 0xF4A (see slide 107)

Encoded literal	Scale value	#of rotations right =2 × Scale value	# of rotations left =32 - 2 × Scale value	Decoded literal
<b>1111</b> mnop wxyz	<b>(15)</b> <sub>10</sub>	$(30)_{10}$	2	0000 0000 0000 0000 0000 00 <b>mn opwx yz</b> 00

#### **Handling Literals Example**

If the literal value is 0x60000008, e.g., as in ADD R0,R1,#0x60000008 what is the 0-to-255 value (from 0 to 0xFF) and the align code (from 0 to 0xF)?

Convert the literal into 32-bit binary value.  $0x6000008 \Rightarrow 0x0110\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 1000$ 

Identify the <u>shortest</u> pattern that include all the 1's in a circular fashion  $0x60000008 \rightarrow 0x0110\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000$ 

If the length of this pattern (*in a circular fashion*) is <u>less than or equal 8</u>, it means that you will be able to encode the number as a value from 0 to 255 and an align code.

If the length of this pattern is less than 8, augment the pattern by zeros to make it 8 bits in total (the 0-to-255 value). Make sure that the number of the other zeros to the left and to the right are even, OR the length of left and right pattern are even

the length of left and right pattern are even

The 0-to-255 value is  $1000 \ 0110_2 \rightarrow 0x86$ 

A value from 0-to-255 needs to be rotated right 4 times to become 0x60000008,

hence the align code is  $4 \div 2 = 2 = 0 \times 2$  Operand 2 is  $0 \times 286$  (see slide 107)

110

Encoded literal	Scale value	#of rotations right =2 × Scale value	=32 - 2 × Scale value	Decoded literal
0010 mnop wxyz	<b>(2)</b> <sub>10</sub>	$(4)_{10}$	28	wx yz 0000 0000 0000 0000 0000 0000 mnop

