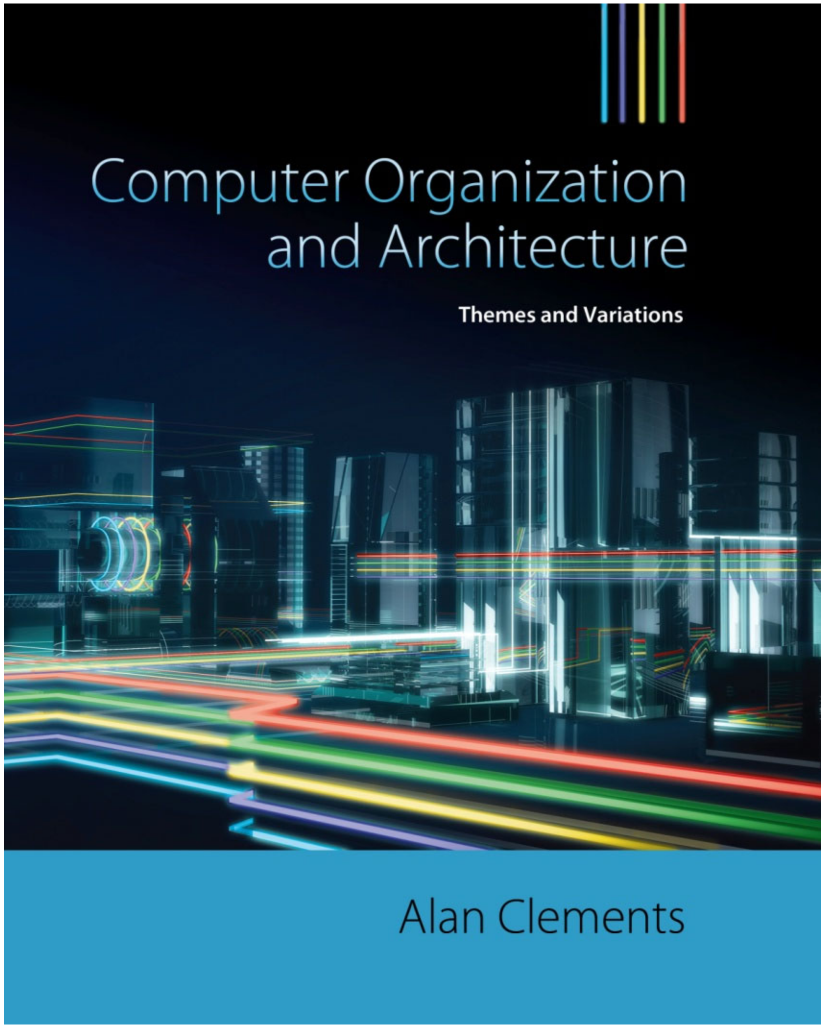
Part 7

CHAPTER 3

Architecture and Organization



CENGAGE Learning

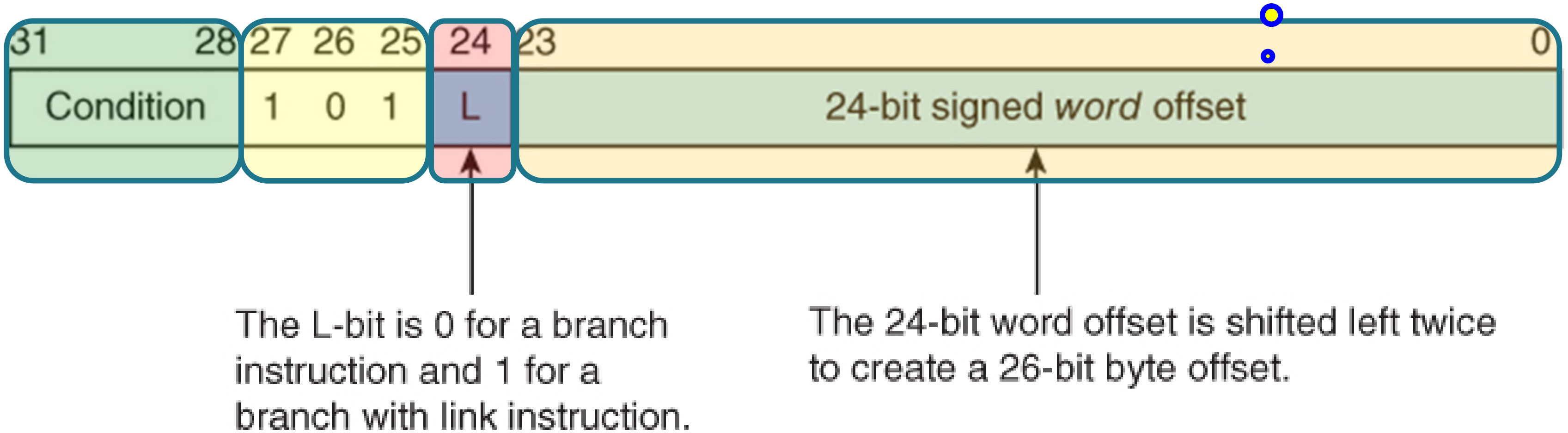
Instruction Encoding
An Insight into the ARM's Architecture

- The branch instruction (Figure 3.41) has
  - an 8-bit op-code
  - a 24-bit ***signed*** program-counter relative ***offset*** (***word*** *address offset*).
- Converting the 24-bit ***word*** offset to real ***byte*** address:
  - shift left twice the 24-bit ***word*** offset to convert the ***word-offset*** address to a ***byte-offset*** address,
  - ***sign-extended*** to 32 bits,
  - added it to the current value of the program counter
    (***the result is in the range PC ± 32 MBytes***).

Do not forget the pipelining effect

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



Instruction Encoding

ARM Instruction: : **Loop B Loop**

Condition = 1110 (always – unconditional)

L = 0 (Not **BL**)

Loop B Loop

. . . .

. . . .

PC location

PC location + pipelining

Branch backward 2 instructions

Word-offset = -2

= 2_1111 1111 1111 1111 1111 1110

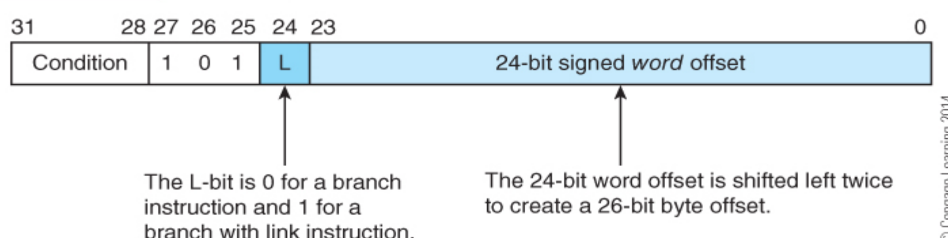
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
1	1	1	0	1	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0

0xEAFFFFFE

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)

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



Instruction Decoding

Machine Language Instruction: **0x1AFFFFFD**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	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

Branch backward 3 instructions.

Target ....

BNE Target — PC location

....

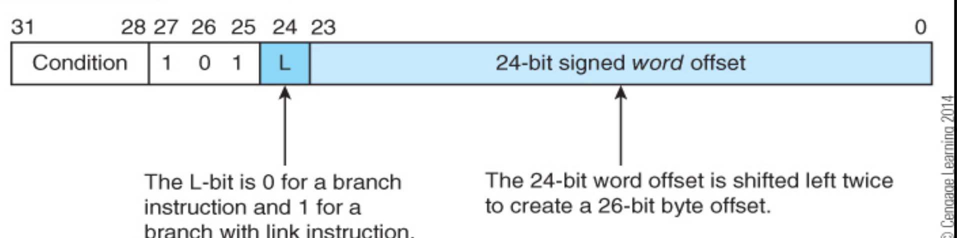
.... — PC location + pipelining

Branch backward 3 instructions

TABLE 3.2 ARM's Conditional Execution and Branch Control Mnemonics

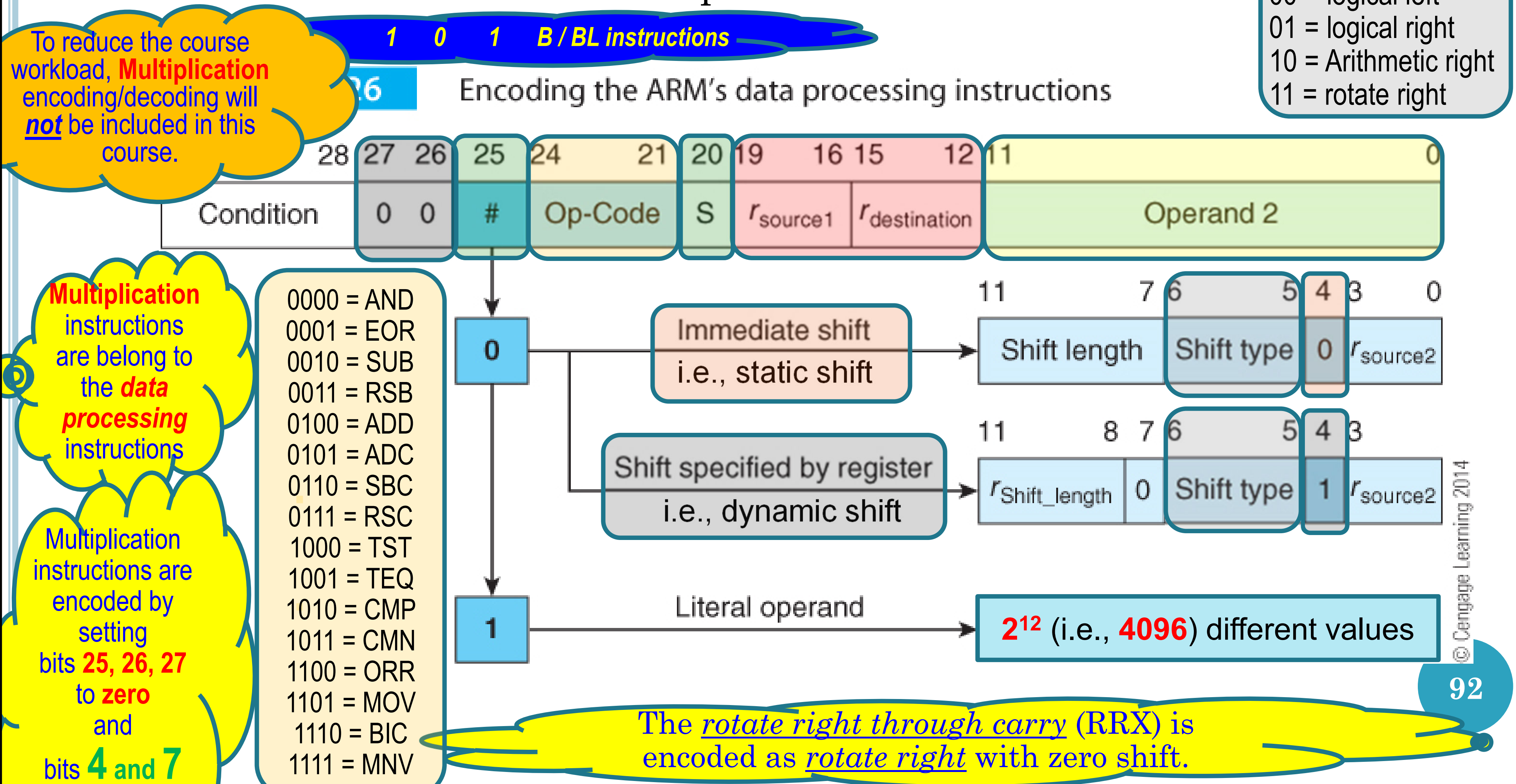
Encoding	Mnemonic	Branch on Flag Status	Execute on condition
0000	EQ	Z set	Equal (i.e., zero)
0001	NE	Z clear	Not equal (i.e., not zero)
0010	CS	C set	Unsigned higher or same
0011	CC	C clear	Unsigned lower
0100	MI	N set	Negative
0101	PL	N clear	Positive or zero
0110	VS	V set	Overflow
0111	VC	V clear	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)

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



Instruction Encoding
An Insight into the ARM's Architecture

- Figure 3.26 illustrates the structure of the ARM's ***data processing*** instructions and demonstrates how bit 25 is used to control the nature of the second source operand.



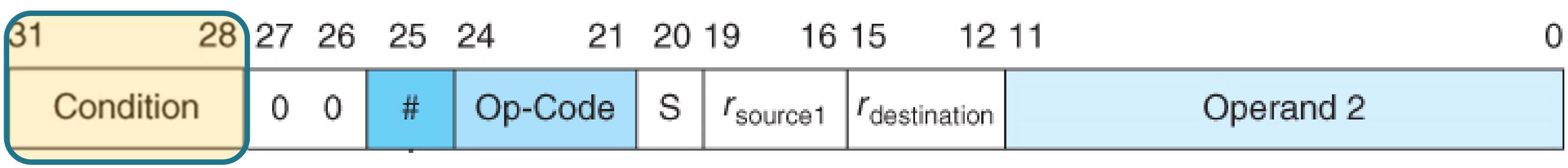
ARM's Flow Control Instructions (Conditional Execution)

- One of **ARM**'s most unusual features is that each instruction can be *conditionally executed*
  - 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*).
- for example,

```
ADDEQ  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



ARM’s Flow Control Instructions (Conditional Execution)

- 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

```
ADDCC r1,r2,r3,LSL r4    ;IF C=0 THEN [r1]<-[r2] + [r3]×2^[r4]
```

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

```
      ADDEQS   r1,r2,r3
Or
      ADDSEQ   r1,r2,r3
```

Both instructions are identical

FIGURE 3.26 Encoding the ARM’s data processing instructions

31–28	27	26	25	24–21	20	19–16	15–12	11–0
Condition	0	0	#	Op-Code	S	$r_{source1}$	$r_{destination}$	Operand 2

ARM's Flow Control Instructions (Conditional Execution)

- 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
  - In this case the subtraction is squashed if the comparison is false

ARM's Flow Control Instructions (Conditional Execution)

- 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  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++.

ARM's Flow Control Instructions (Conditional Execution)

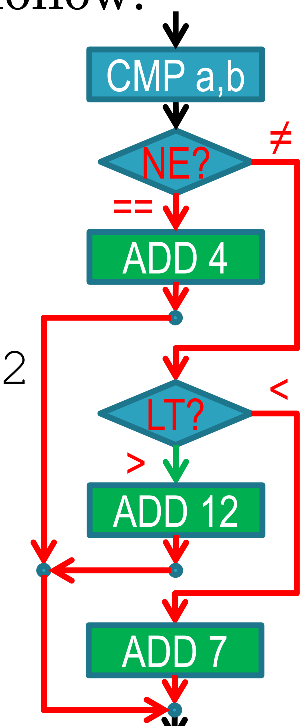
- You can also handle some testing with multiple conditions.
- Consider:
  if (a == b) e = e + 4;
  if (a < b) e = e + 7;
  if (a > b) e = e + 12;

We can use conditional execution to implement this as

```
CMP   r0,r1                    ;compare a == b
ADDEQ r4,r4,#4   ;if a == b then e = e +  4
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
Equal     ADD r4,r4,#4   ;if a == b then e = e + 4
          B   AfterAll
NotEqual  BLT LessThan
          ADD r4,r4,#12  ;if a > b  then e = e + 12
          B   AfterAll
LessThan  ADD r4,r4,#7   ;if a < b  then e = e + 7
AfterAll  …
```



Instruction Encoding

ARM Instruction: `ADD` **`r0`**`,r1,r2`

- `Condition = 1110` (always – unconditional)
- `Op-Code = 0100` (i.e., ADD)
- `S = 0` (not ADDS)
- $r_{destination}$ `= 0000` (destination *operand*)
- $r_{source1}$ `= 0001` (first *operand*)
- `# = 0` (second operand not a constant)
- `Operand 2 (bit number 4 = 0)`
  - $r_{source2}$ `= 0010`
  - `shift type = 00` (logical left)
  - `shift length = 00000`

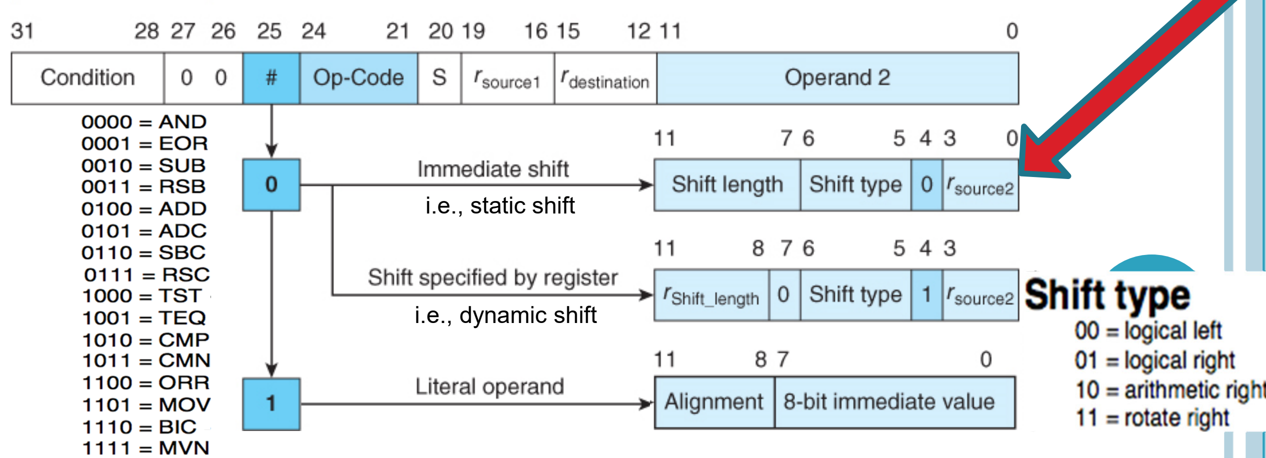
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
1	1	1	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0

0xE0810002

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)

FIGURE 3.26 Encoding the ARM's data processing instructions



Instruction Encoding

ARM Instruction: `ADD r0,r1,r2,LSR r3`

Condition = 1110 (always – unconditional)

Op-Code = 0100 (i.e., ADD)

S = 0 (not ADDS)

$r_{destination}$ = 0000 (destination *operand*)

$r_{source1}$ = 0001 (first *operand*)

= 0 (second operand not a constant)

Operand 2 (bit number 4 = 1)

- $r_{source2}$ = 0010
- shift type = 01 (logical right)
- shift length = r3 = 0011

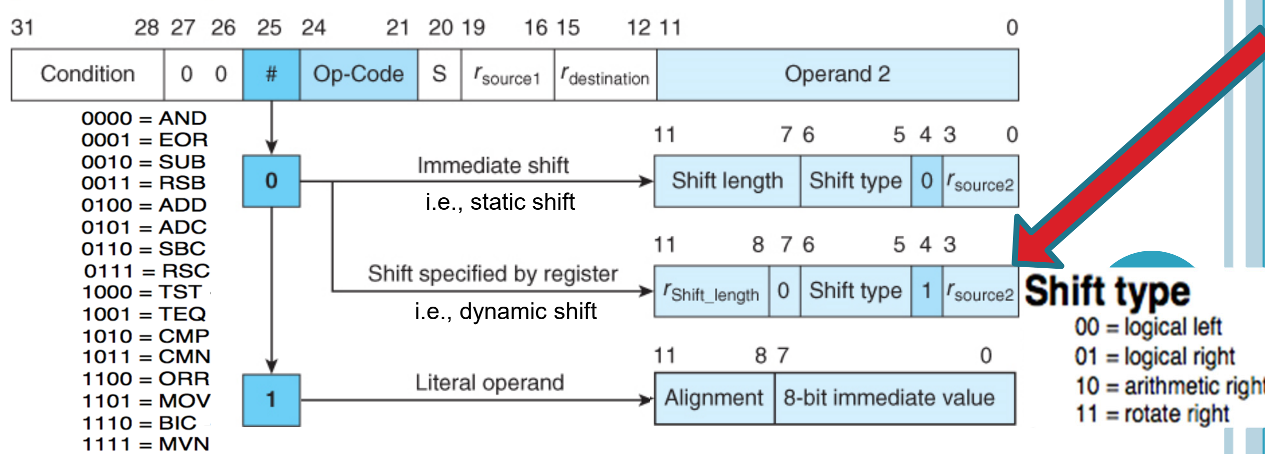
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
1	1	1	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	1	1	0	0	1	1	0	0	1	0

0xE0810332

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)

FIGURE 3.26 Encoding the ARM's data processing instructions



Instruction Decoding

Machine Language Instruction: 0x004A0182

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	1	0

Bits number 26 27 = 00

Op-Code = 0010 (i.e., SUB)

Condition = 0000 (EQ)

S = 0 (*not* SUBEQS)

$r_{destination}$ = 0000 ➔ r0

$r_{source1}$ = 1010 ➔ r10

\# = 0 (second operand not a constant)

Operand 2 (bit number 4 = 0)

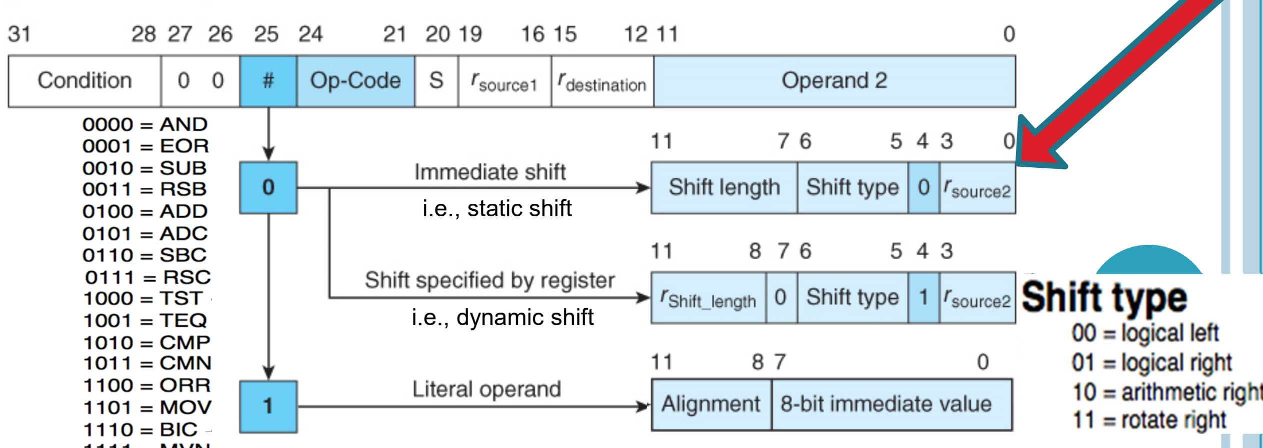
- $r_{source2}$ = 0010 ➔ r2
- shift type = 00 ➔ logical left
- shift length = 00011 ➔ 3

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

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)

FIGURE 3.26 Encoding the ARM's data processing instructions



Instruction Decoding

Machine Language Instruction: **0x004A0152**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	0	0	0	0	0	0	0	0	1	0	1	0	1	0	0	1	0

Bits number 26 27 = 00

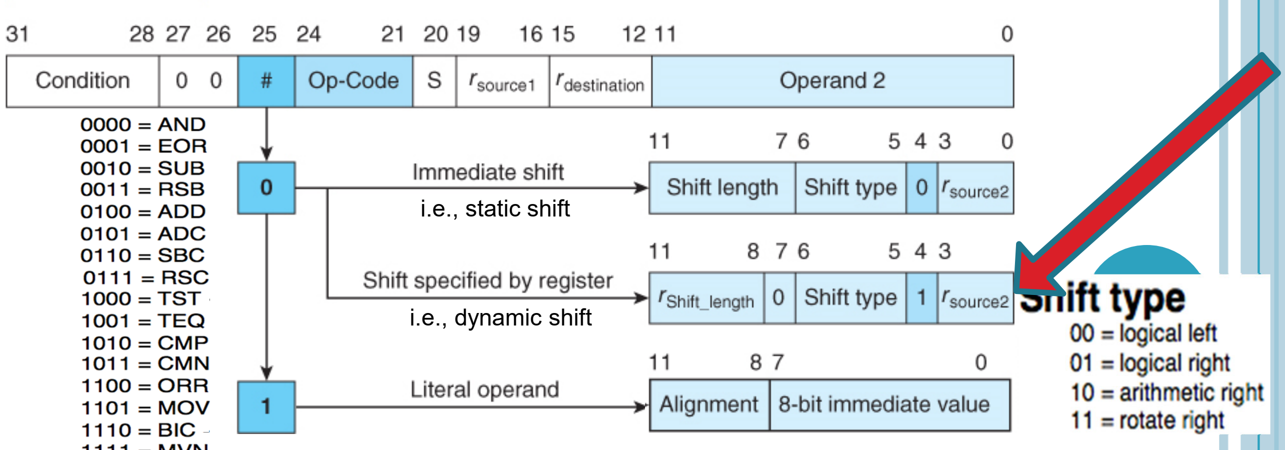
- Op-Code = 0010 (i.e., SUB)
- Condition = 0000 (EQ)
- S = 0 (*not* SUBEQS)
- $r_{destination}$ = 0000 → r0
- $r_{source1}$ = 1010 → r10
- \# = 0 (second operand not a constant)
- Operand 2 (bit number 4 = 1)
  - $r_{source2}$ = 0010 → r2
  - shift type = 10 → Arithmetic right
  - shift length = r1

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)

FIGURE 3.26 Encoding the ARM's data processing instructions

SUBEQ **r0**,r10,r2,ASR r1



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)*
- $r_{source1}$ = 0011 (first *operand*)
- \# = 0 (second operand not a constant)
- Operand 2 (bit number 4 = 0)
  - $r_{source2}$ = 0101
  - shift type = 00 (logical left)
  - shift length = 00000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
1	1	0	0	0	0	0	1	0	1	0	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1

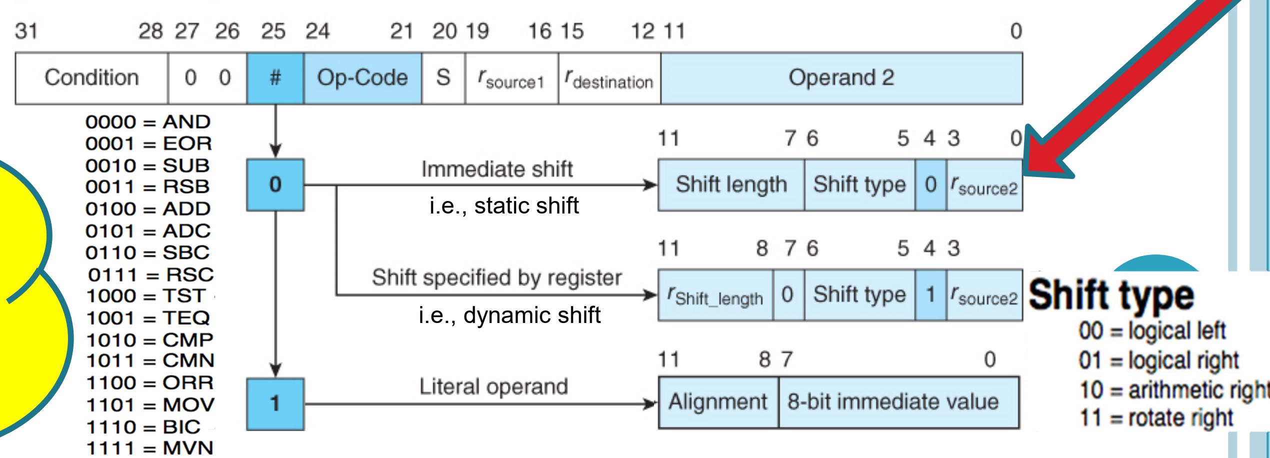
0xC1530005

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

FIGURE 3.26 Encoding the ARM's data processing instructions



Instruction Encoding

ARM Instruction: `MOV **PC**, LR`

- Condition = 1110 (always – unconditional)
- Op-Code = 1101 (i.e., MOV)
- S = 0 (not MOVS)
- $r_{destination}$ = 1111 (PC)
- *$r_{source1}$ = 0000 (must be zeros)*
- # = 0 (second operand not a constant)
- Operand 2 (bit number 4 = 0)
  - $r_{source2}$ = 1110
  - shift type = 00 (logical left)
  - shift length = 00000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
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

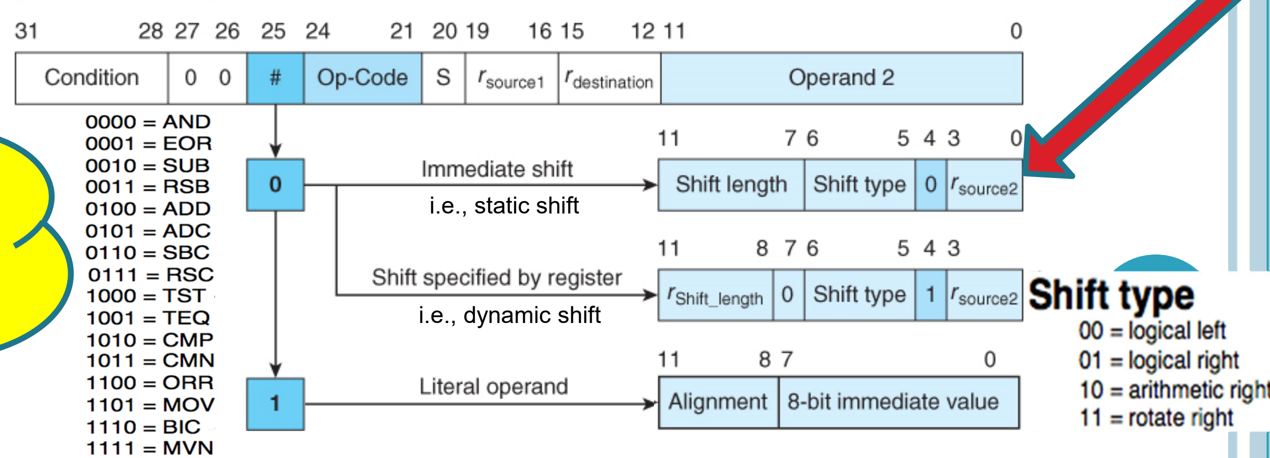
0xE1A0F00E

In all *moving instructions, i.e., MOV, and MVN, the* ***source$_1$ register*** *field* ***MUST BE encoded as 0000***

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)

FIGURE 3.26 Encoding the ARM's data processing instructions



Instruction Decoding

Machine Language Instruction: 0xE1B01061

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	0	0	0	1	1	0	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	1	1	0	0	0	0	1

Bits number 26 27 = 00

Op-Code = 1101 (i.e., MOV)

Condition = 1110 (Always)

S = 1 (i.e., MOVS)

$r_{destination}$ = 0001 ➔ r1

$r_{source1}$ = 0000 (must be zeros)

= 0 (second operand not a constant)

Operand 2 (bit number 4 = 0)

$r_{source2}$ = 0001 ➔ r1

shift type = 11 ➔ rotate right

shift length = 00000 ➔ **00**

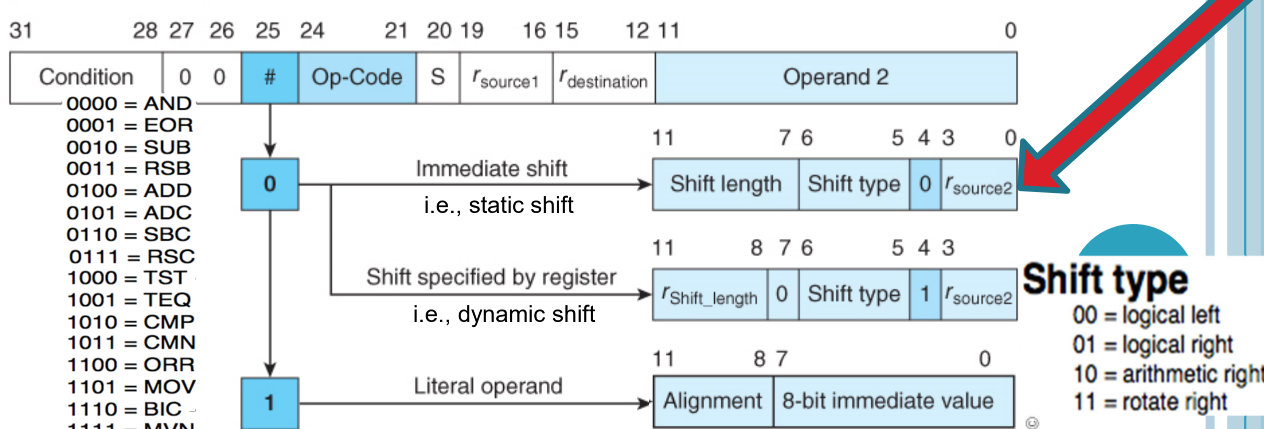
rotate right and shift length =00 ➔RRX

MOVS **r1**,r1,RRX

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)

FIGURE 3.26 Encoding the ARM's data processing instructions



Handling Literals

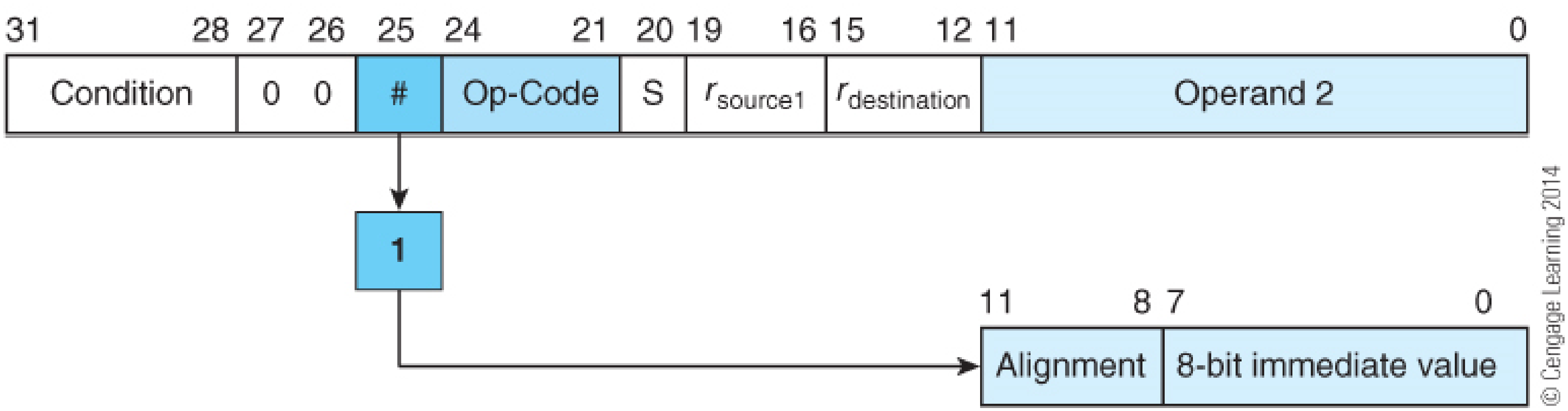
- In **ARM**, ***operand 2*** can be a literal.

```
ADD r0,r1,#7      ;adds 7 to r1 and puts the result in r0.
MOV 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 these12-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.

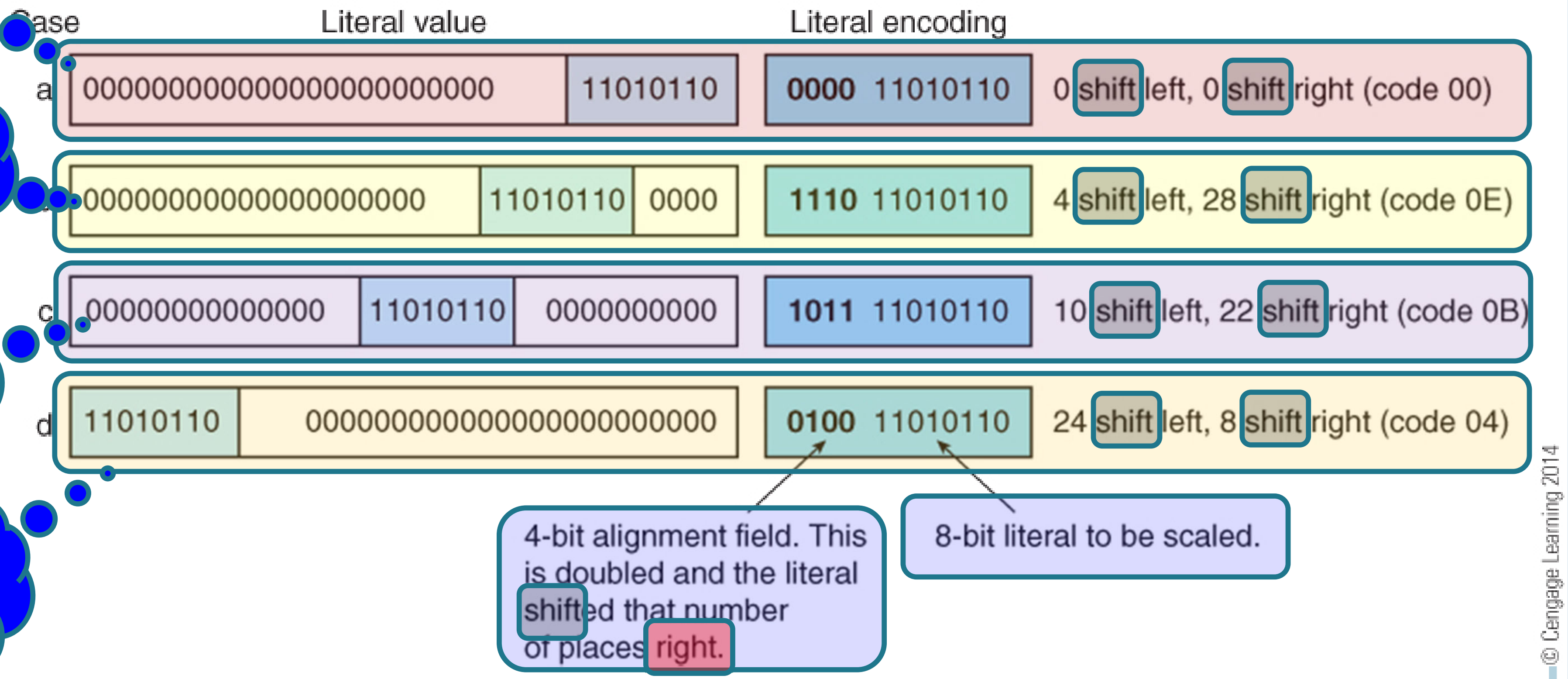
FIGURE 3.28 Diagram of ARM's literal operand encoding



Handling Literals

Replace the word "shift" in this slide by "rotate". The book is wrong!!

FIGURE 3.29 ARM's literal operand encoding



Zero rotation

4 rotations left

10 rotations left

24 rotations left

The number of actual rotations is ALWAYS *even* (i.e., can not be odd at all).

Also called the 4 bits *alignment*

Handling Literals

You need to know how to *decode* and encode 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	$(32)_{10}$	0000 0000 0000 0000 0000 0000 **mnop wxyz**
1111 mnop wxyz	$(15)_{10}$	$(30)_{10}$	2	0000 0000 0000 0000 0000 00**mn opwx yz**00
1110 mnop wxyz	$(14)_{10}$	$(28)_{10}$	4	0000 0000 0000 0000 0000 **mnop wxyz** 0000
1101 mnop wxyz	$(13)_{10}$	$(26)_{10}$	6	0000 0000 0000 0000 00**mn opwx yz**00 0000
1100 mnop wxyz	$(12)_{10}$	$(24)_{10}$	8	0000 0000 0000 0000 **mnop wxyz** 0000 0000
1011 mnop wxyz	$(11)_{10}$	$(22)_{10}$	$(10)_{10}$	0000 0000 0000 00**mn opwx yz**00 0000 0000
1010 mnop wxyz	$(10)_{10}$	$(20)_{10}$	$(12)_{10}$	0000 0000 0000 **mnop wxyz** 0000 0000 0000
1001 mnop wxyz	**9**	$(18)_{10}$	$(14)_{10}$	0000 0000 00**mn opwx yz**00 0000 0000 0000
1000 mnop wxyz	**8**	$(16)_{10}$	$(16)_{10}$	0000 0000 **mnop wxyz** 0000 0000 0000 0000
0111 mnop wxyz	**7**	$(14)_{10}$	$(18)_{10}$	0000 00**mn opwx yz**00 0000 0000 0000 0000
0110 mnop wxyz	**6**	$(12)_{10}$	$(20)_{10}$	0000 **mnop wxyz** 0000 0000 0000 0000 0000
0101 mnop wxyz	**5**	$(10)_{10}$	$(22)_{10}$	00**mn opwx yz**00 0000 0000 0000 0000 0000
0100 mnop wxyz	**4**	8	$(24)_{10}$	**mnop wxyz** 0000 0000 0000 0000 0000 0000
0011 mnop wxyz	**3**	6	$(26)_{10}$	**opwx yz**00 0000 0000 0000 0000 0000 00**mn**
0010 mnop wxyz	**2**	4	$(28)_{10}$	**wxyz** 0000 0000 0000 0000 0000 0000 **mnop**
0001 mnop wxyz	**1**	2	$(30)_{10}$	**yz**00 0000 0000 0000 0000 0000 00**mn opwx**

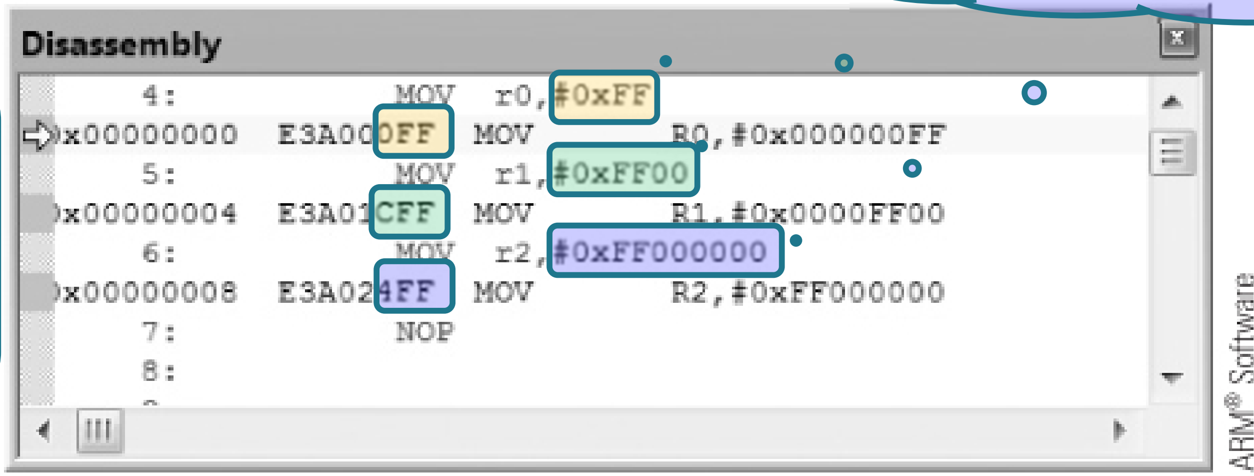
Handling Literals

8-bit immediate value =0xFF and the ZERO rotation left

8-bit immediate value =0xFF and the 8 rotations left

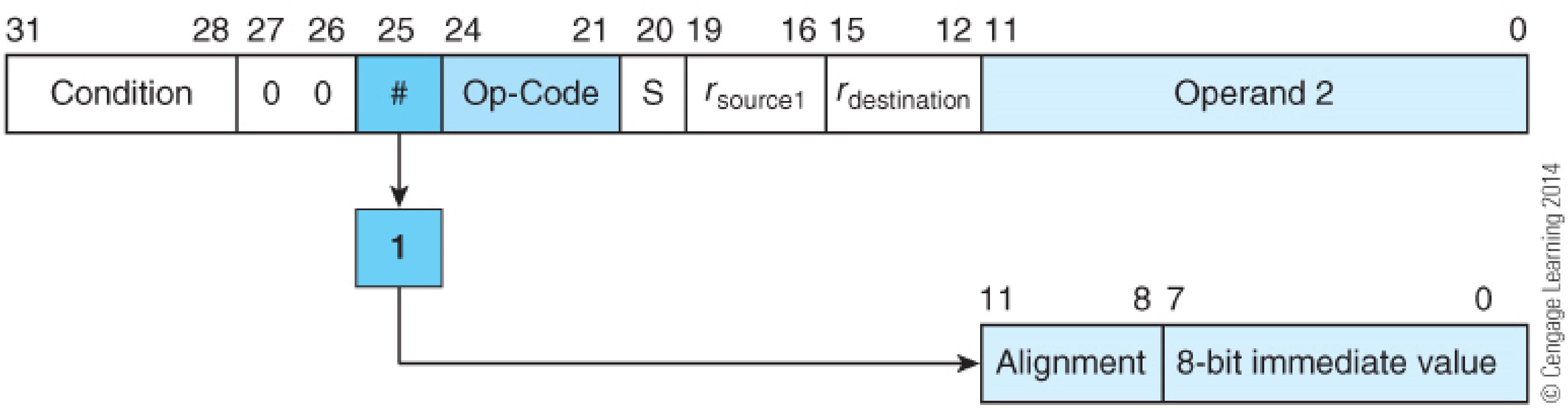
8-bit immediate value =0xFF and the 24 rotations left

FIGURE 3.30 Example of ARM literal encoding



If the value can not be represented that way, you will get an error message saying: *"cannot be represented by 0-255 and a rotation."*

FIGURE 3.28 Diagram of ARM's literal operand encoding



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 ➔ 0x0000 0000 0000 0000 0000 **0001 0010 1000**

*Identify the **shortest** pattern that include all the 1's **in a circular fashion***
0x128 ➔ 0x0000 0000 0000 0000 0000 000**1 0010 1**000

If the length of this pattern (***in a circular fashion)*** is *less than or equal 8,* 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 ➔ 0x0000 0000 0000 0000 0000 00**01 0010 10**00

*the number of the **other zeros** to the left and to the right are even*

The 0-to-255 value is **01 0010 10**$_2$ ➔ `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 ÷ 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
1111 mnop wxyz	(15)$_{10}$	(30)$_{10}$	2	0000 0000 0000 0000 0000 00mn opwx yz00

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.
0x60000008 ➔ 0x**0110** 0000 0000 0000 0000 0000 0000 **1000**

*Identify the **shortest** pattern that include all the 1's **in a circular fashion***
0x60000008 ➔ 0x0110 0000 0000 0000 0000 0000 0000 1000

If the length of this pattern (***in a circular fashion)*** is *less than or equal 8,* 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*

0x60000008 ➔ 0x0110 0000 0000 0000 0000 0000 0000 1000

the length of left and right pattern are even

The 0-to-255 value is **1000 0110**$_2$ ➔ `0x86`

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

hence the align code is 4 ÷ 2 = 2 = `0x2` ➔ Operand 2 is `0x286` ***(see slide 107)***

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

Encoded literal	Scale value	#of rotations right =2 × Scale value	# of rotations left =32 - 2 × Scale value	Decoded literal
1100 mnop wxyz	(12)	(24)	8	0000 0000 0000 0000 mnop wxyz 0000 0000

Instruction Encoding

ARM Instruction: `ORRGTS r1,r2,#0xAA00`

0x0000 0000 0000 0000 1010 1010 0000 0000

Condition = 1100 (Greater than)
Op-Code = 1100 (i.e., ORR)
S = 1 (ORRGTS)
$r_{destination}$ = 0001 (destination *operand*)
$r_{source1}$ = 0010 (first *operand*)
\# = 1 (second operand is a constant)
Operand 2 (*to be 0-255 and a rotation*)
8-bit immediate value = 0xAA
rotations left = 8
equivalent to 24 rotations right
Half of the rotations right = 12
i.e., 1100 in binary

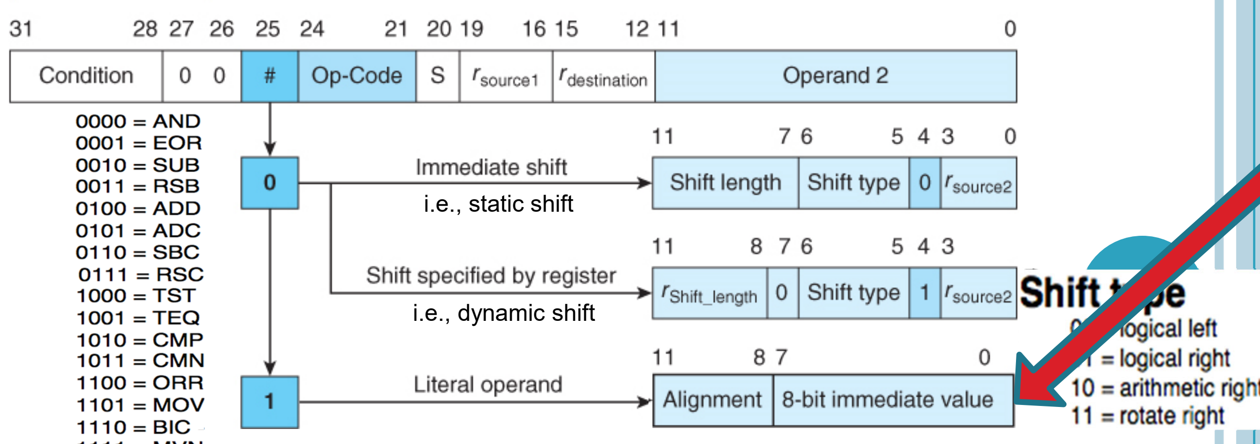
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
1	1	0	0	0	0	1	1	1	0	0	1	0	0	1	0	0	0	0	1	1	1	0	0	1	0	1	0	1	0	1	0

0xC3921CAA

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)

FIGURE 3.26 Encoding the ARM's data processing instructions



Encoded literal	Scale value	#of rotations right =2 × Scale value	# of rotations left =32 - 2 × Scale value	Decoded literal
1111 mnop wxyz	(15)	(30)	2	0000 0000 0000 0000 0000 00mn opwx yz00

Instruction Decoding

Machine Language Instruction: **0x42742F55**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
0	1	0	0	0	0	1	0	0	1	1	1	0	1	0	0	0	0	1	0	1	1	1	1	0	1	0	1	0	1	0	1

Bits number 26 27 = 00

Op-Code = 0011 (i.e., RSB)

Condition = 0100 (MI)

S = 1 (RSBMIS)

$r_{destination}$ = 0010 → r2

$r_{source1}$ = 0100 → r4

= 1 (second operand is a constant)

Operand 2 (*to be 0-255 and a rotation*)

8-bit immediate value = 0x55

Alignment = 0xF, i.e.,

Half of the rotations right=0xF

rotations right =30

rotations left = 2

0x55 = 2_0101 0101

0x55 << 2

= 2_01 0101 01**00**

= 0x154

RSBMIS r2,r4,#0x154

As a practice, try to encode this instruction by yourself

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)

FIGURE 3.26 Encoding the ARM's data processing instructions