ISA Assembler Format (4A)

Data Processing Instructions

Young Won Lim
2/4/20

Copyright (c) 2014 - 2020 Young W. Lim.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

Please send corrections (or suggestions) to youngwlim@hotmail.com.

This document was produced by using LibreOffice.

Based on

ARM System-on-Chip Architecture, 2nd ed, Steve Furber

Branch and Branch with Link (B, BL)

Branch and Branch with Link (B, BL)

B{L} {<cond>} <target address>

L : the branch and link

<cond> : condition codes, AL if omitted

<target address> : a label in the assembler code
The assembler will generate the offset
target address – branch instruction address + 8

```
B           <target address>
B<cond>     <target address>
BL          <target address>
BL<cond>    <target address>
```

Branch, Branch with Link and eXchange (BX, BLX)

Branch, Branch with Link and eXchange (BX, BLX)

B{L}X {<cond>} Rm

BLX <target address>

L : the branch and link

<cond> : condition codes, **AL** if omitted

<target address> : a label in the assembler code

The assembler will generate the **offset**

target address – branch instruction address + 8

BX	Rm
BLX	Rm
BLX	<target address>
BX<cond>	Rm
BLX<cond>	Rm
BLX<cond>	<target address>

Condition Code Suffixes <cond>

Suffix	Flags	Meaning
EQ	Z = 1	Equal
NE	Z = 0	Not equal
CS or **HS**	C = 1	Higher or same, unsigned
CC or **LO**	C = 0	Lower, unsigned
MI	N = 1	Negative
PL	N = 0	Positive or zero
VS	V = 1	Overflow
VC	V = 0	No overflow
HI	C = 1 and Z = 0	Higher, unsigned
LS	C = 0 or Z = 1	Lower or same, unsigned
GE	N = V	Greater than or equal, signed
LT	N != V	Less than, signed
GT	Z = 0 and N = V	Greater than, signed
LE	Z = 1 and N != V	Less than or equal, signed
AL	any value	Always. This is the default when no suffix is specified.

https://community.arm.com/processors/b/blog/posts/condition-codes-1-condition-flags-and-codes

Conditional Flags

N=1 if the result is negative

Z=1 if the result is zero

C=1 the carry out of the ALU when the operation is arithmetic
(ADD, ADC, SUB, SBC, RSB, RSC, CMP, CMN), or
the carry out of the shifter (C is preserved when no shift)

V=1 if overflow is occurred during arithmetic operations
only when an arithmetic operation has operands
that are viewed as 2's complement signed value
(V is preserved when non-arithmetic operations)

Conditional Flag Setting Instructions

Data Processing Instructions

<op> {<cond>} {**S**} Rd, Rn, #<32-bit immediate>

<op> {<cond>} {**S**} Rd, Rn, Rm, {<shift>}

Multiply Instructions

MUL {<cond>} {**S**} Rd, Rm, Rs

MLA {<cond>} {**S**} Rd, Rm, Rs, Rn

<mul> {<cond>} {**S**} RdHi, RdLo, Rm, Rs

UMULL, UMLAL, SMULL, SMLAL

<mul>	**Meaning**
MUL	Multiply (32-bit)
MLA	Multiply-Accumulate (32-bit)
UMULL	Unsigned Multiply Long
UMLAL	Unsigned Multiply Acc Long
SMULL	Singed Multiply Long
SMLAL	Unsigned Multiply Acc Long

<op>	**Meaning**
AND	Logical bit-wise AND
EOR	Logical bit-wise exclusive OR
SUB	Subtract
RSUB	Reverse subtract
ADD	Add
ADC	Add with carry
SBC	Subtract with carry
RSC	Reverse subtract with carry
TST	Test

Branch Conditions

B<cond>		
B	Unconditional	Always take this branch
BAL	Always	Always take this branch
BEQ	Equal	Comparison equal or zero result
BNE	Not equal	Comparison not equal or non-zero result
BPL	Plus	Result positive or zero
BMI	Minus	Result minus or negative
BCC	Carry clear	Arithmetic operation did not give carry-out
BLO	Lower	Unsigned comparison give lower
BCS	Carry set	Arithmetic operation gave carry-out
BHS	Higher or same	Unsigned comparison gave higher or same
BVC	Overflow clear	Signed integer operation; no overflow occurred
BVS	Overflow set	Signed integer operation; overflow occurred
BGT	Greater than	Signed integer comparison gave greater than
BGE	Greater or equal	Signed integer comparison gave greater or equal
BLT	Less than	Signed integer comparison gave less than
BLE	Less or equal	Signed integer comparison gave less than or equal
BHI	Higher	Unsigned comparison gave higher
BLS	Lower or same	Unsigned comparison gave low or same

Data Processing Instructions

Data Processing Instructions

<op> {<cond>} {S} **Rd**, **Rn**, #<32-bit immediate> ... *12-bit encoded*
<op> {<cond>} {S} **Rd**, **Rn**, Rm, {<shift>}

Omitting Rn when the instruction is monadic (MOV, MVN)
Omitting Rd when the instruction only produces condition code
(CMP, CMN, TST, TEQ)
<shift> specifies
the shift type (LSL, LSR, ASL, ASR, ROR, RRX)
the shift amount (except RRX)
By a 5-bit immediate (# <#shift>)
By a register (Rs)

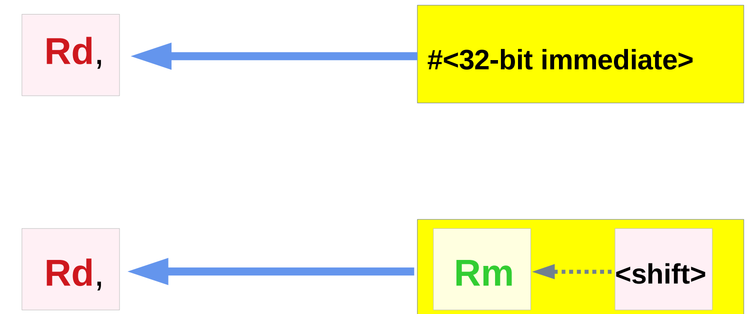
Data Processing Type 1 – no 1st operand **Rn**

Data Processing Instructions – no 1st operand Rn

<op_type1> {<cond>} {S} **Rd**, #<32-bit immediate> ... *12-bit encoded*

<op_type1> {<cond>} {S} **Rd**, Rm, {<shift>}

MOV	Move
MVN	Move Negated



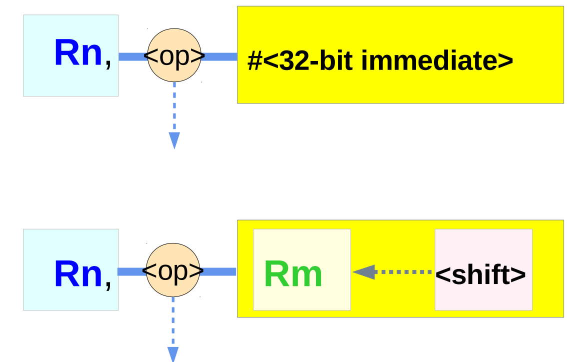
Rn : SBZ (Should Be Zero) fields

Data Processing Type 2 – no destination **Rd**

Data Processing Instructions – no destination Rd

<op_type2> {<cond>} **Rn**, #<32-bit immediate> ... *12-bit enconded*

<op_type2> {<cond>} **Rn**, Rm, {<shift>}



CMP	Compare
CMN	Compare Negated
TST	Test
TEQ	Test Equivalence

S is implicitly implied

Rd : SBZ (Should Be Zero) fields

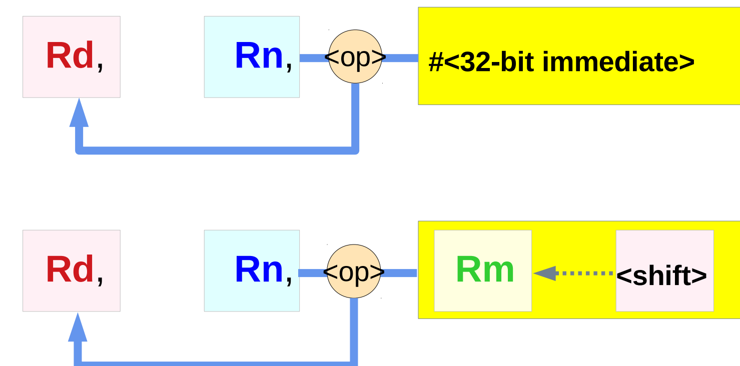
Data Processing Type 3 – Arithmetic & Logical Instructions

Data Processing Instructions

<op> {<cond>} {S} **Rd**, **Rn**, #<32-bit immediate> ... *12-bit enconded*

<op> {<cond>} {S} **Rd**, **Rn**, Rm, {<shift>}

AND	logical bit-wise AND
EOR	logical bit-wise Exclusive OR
SUB	Subtract
RSB	Reverse Subtract
ADD	Add
ADC	Add with Carry
SBC	Subtract with Carry
RSC	Reverse Subtract with Carry
ORR	logical bit-wise OR
BIC	Bit Clear



Data Processing – Format Listings

Data Processing Instructions

<op> {<cond>} {S} Rd, Rn, #<32-bit immediate> ... *12-bit enconded*

<op> {<cond>} {S} Rd, Rn, Rm, {<shift>}

<op>	Rd, Rn, #<32-bit immediate>
<op>	Rd, Rn, Rm
<op>	Rd, Rn, Rm, <shift>
<op> <cond>	Rd, Rn, #<32-bit immediate>
<op> <cond>	Rd, Rn, Rm
<op> <cond>	Rd, Rn, Rm
<op> S	Rd, Rn, #<32-bit immediate>
<op> S	Rd, Rn, Rm
<op> S	Rd, Rn, Rm, <shift>
<op> <cond> S	Rd, Rn, #<32-bit immediate>
<op> <cond> S	Rd, Rn, Rm
<op> <cond> S	Rd, Rn, Rm, <shift>

Data Processing Format

No 1st operand Rn

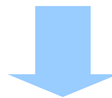
<op_type1> {<cond>} {S} **Rd**, **#<32-bit immediate>** ... *12-bit encoded*
Rm, {<shift>}

No destination Rd

<op_type2> {<cond>} **Rn**, **#<32-bit immediate>** ... *12-bit encoded*
Rm, {<shift>}

Both Rd and Rn

<op> {<cond>} {S} **Rd**, **Rn**, **#<32-bit immediate>** ... *12-bit encoded*
Rm, {<shift>}



<op2>

Operand2 **<op2>**

Operand2 **<op2>** is the flexible second operand to most instructions.

An immediate value

4-bit rotate + 8-bit immediate … *12-bit encoded*

An 8-bit number rotated right by an even number of places.

A register shifted by value

immediate shift amount : a 5-bit unsigned integer

A register shifted by register

register shift amount : the lower 8 bits of a register

http://www.davespace.co.uk/arm/introduction-to-arm/operand2.html

12-bit immediate value encoding

the 12-bit immediate value
not as a 12-bit number.
but an 4-bit rotation with a 8-bit number

[24-bit padding zeros + 8-bit number] : a full 32-bit word

the 4-bit rotation value has 2^4=16 possible settings
16 possible rotations of 8-bit number in the 32-bit word
(0, 1, 2, …, 15) * 2
(0, 2, 4, …, 30) : even number of rotations

first, the 8-bit number is zero-padded to form a 32-bit number
then rotate right the 32-bit number by 4-bit rotation * 2

https://alisdair.mcdiarmid.org/arm-immediate-value-encoding/

Immediate value encoding example

```
........................76543210   0    0000
0........................7654321   1
10........................765432   2    0001
210........................76543   3
3210........................7654   4    0010
43210........................765   5
543210........................76   6    0011
6543210........................7   7
76543210........................   8    0100
.76543210.......................   9
..76543210......................   10   0101
...76543210.....................   11
....76543210....................   12   0110
.....76543210...................   13
......76543210..................   14   0111
.......76543210.................   15
........76543210................   16   1000
.........76543210...............   17
..........76543210..............   18   1001
...........76543210.............   19
............76543210............   20   1010
.............76543210...........   21
..............76543210..........   22   1011
...............76543210.........   23
................76543210........   24   1100
.................76543210.......   25
..................76543210......   26   1101
...................76543210.....   27
....................76543210....   28   1110
.....................76543210...   29
......................76543210..   30   1111
.......................76543210.   31
```

Data Processing – Shift Operand

Data Processing Instructions with <shift>

<op>	{<cond>} {S} **Rd**, **Rn**,	**Rm**, {**<shift>**}	AND,EOR,SUB,RSB,ADD,ADC,SBC,RSC,ORR,BIC
<op_type1>	{<cond>} {S} **Rd**,	**Rm**, {**<shift>**}	MOV, MVN … special case
<op_type2>	{<cond>} **Rn**,	**Rm**, {**<shift>**}	CMP, CMN, TST,TEQ … special case

<shift>

<shift type>	# <#shift>	…. instruction-specified shift amount
<shift type>	Rs	…. register-specified shift amount
LSL, ASL,	Logical Shift Left, Arithmetic Shift Left	
LSR,	Logical Shift Right	
ASR,	Arithmetic Shift Right	
ROR	Rotate Right	

<shift type>		
RRX	Rotate Right Extended	… no shift amount

Data Processing – simulating shift instruction

Move Instructions with <shift>

MOV {<cond>} {S} **Rd**, **Rm**, **<shift>**

<shift>

<shift type> # <#shift> …. instruction-specified shift amount

<shift type> Rs …. register-specified shift amount

can simulate statndalone shift instructions of other machines

MOV R0. R0, LSR 2

MOV R0. R0, LSR Rs

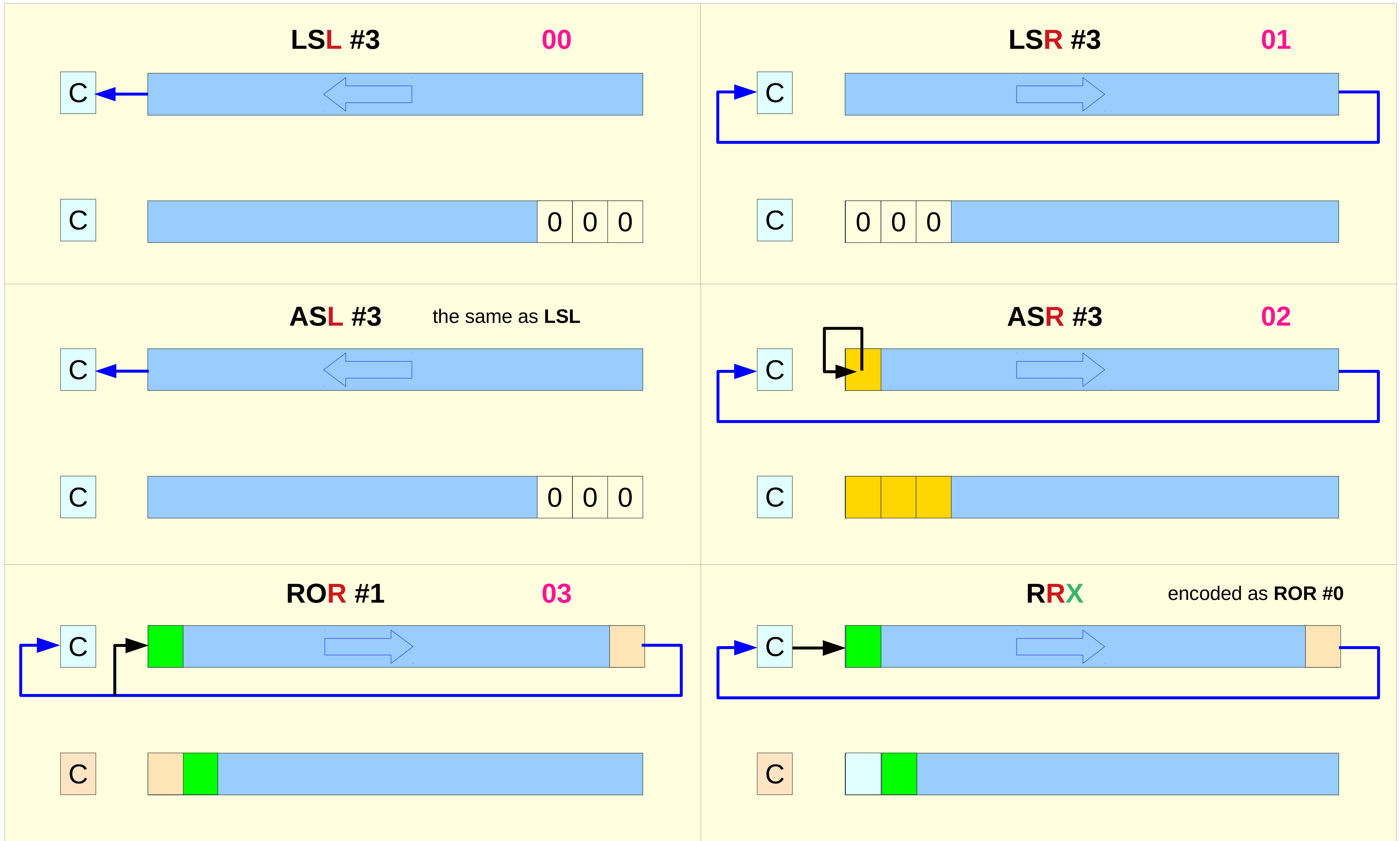
Stand Alone Shift Instruction Synonyms … syntactic sugars

op {S} {cond} **Rd, Rm, <shift>**

LSL, LSR, ASL, LSR, ASR, ROR no RRX

RealView Development Suite Ver 4.0 http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dui0204j/Cjacbgca.html

Shift Type



Shifted Operand Example

<shift> general shift operation

a. **<shiftname> Rs** .... instruction-specified shift amount

LSL Rs	(Logical Shift Left)
LSR Rs	(Logical Shift Right)
ASL Rs	(Arithmetic Shift Left)
ASR Rs	(Arithmetic Shift Right)
ROR Rs	(Rotate Right)

b. **<shiftname> #expression** .... register-specified shift amount

LSL #<sh amount>	(Logical Shift Left)
LSR #<sh amount>	(Logical Shift Right)
ASL #<sh amount>	(Arithmetic Shift Left)
ASR #<sh amount>	(Arithmetic Shift Right)
ROR #<sh amount>	(Rotate Right)

c. **RRX**

RRX	(Rotate Right with eXtend) without shift amount actually encoded as **ROR #0**

Shifted Operand

<shift> general shift operation

a. **<shiftname> Rs**
used in data processing instruction
cannot be used in data transfer instructions

b. **<shiftname> #expression**
if **#expresssion** is used, the assembler will attempt
to generate a shifted immediate **8-bit** field
to match the expression.
If this is impossible, it will give an error.

c. **RRX** (Rotate Right one bit with eXtend)

Shifted Operand Encoding

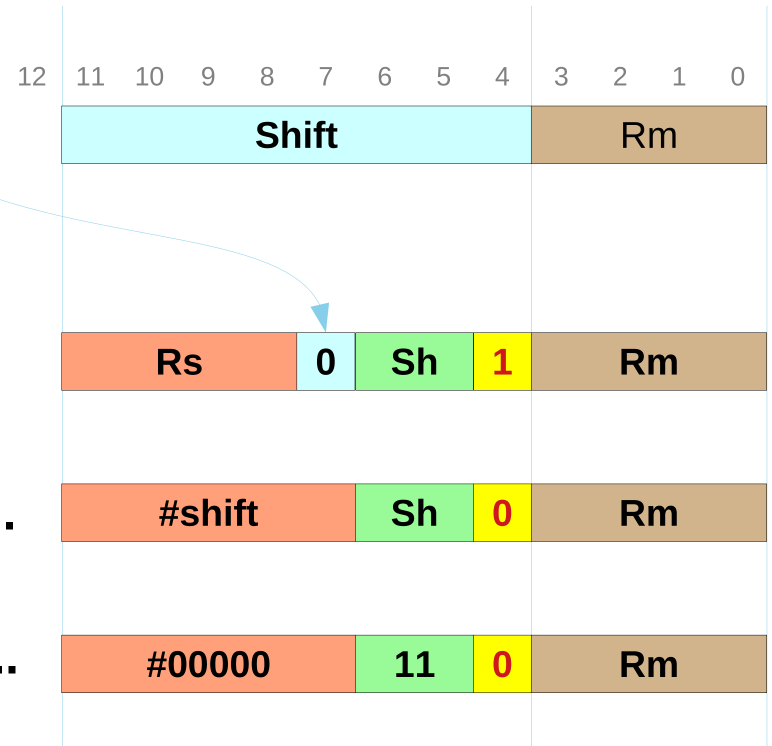
The zero in bit 7 of an instruction with a **register controlled shift** is compulsory; a one in this bit will cause the instruction to be a multiply or undefined instruction.

a. **<shiftname> Rs** ....

b. **<shiftname> #expression** ....

5-bit immediate for shift amount

c. **RRX (= ROR #0)** ....



Sh	Shift Name
00	Logical Left
01	Logical Right
10	Arithmetic Right
11	Rotate Right

Instructions using a Shifted Operand

Data Processing Instructions with <shift>

<op> {<cond>} {S} **Rd**, **Rn**, Rm, {**<shift>**}	AND,EOR,SUB,RSB,ADD,ADC,SBC,RSC,ORR,BIC
<op1> {<cond>} {S} **Rd**, Rm, {**<shift>**}	MOV, MVN
<op2> {<cond>} **Rn**, Rm, {**<shift>**}	CMP, CMN, TST,TEQ

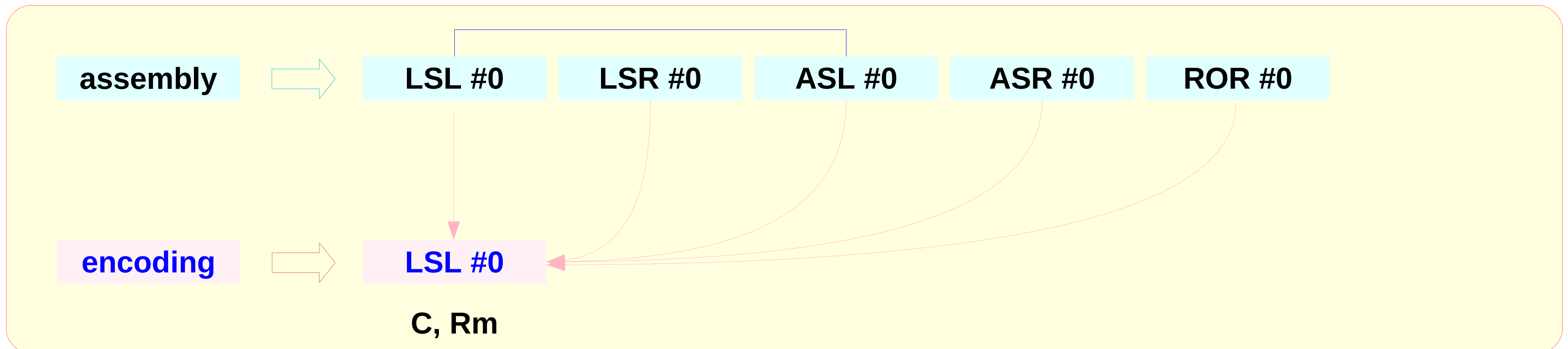
<shift type> #<#shift> … instruction specified shift amount
<shift type> Rs … register specified shift amount

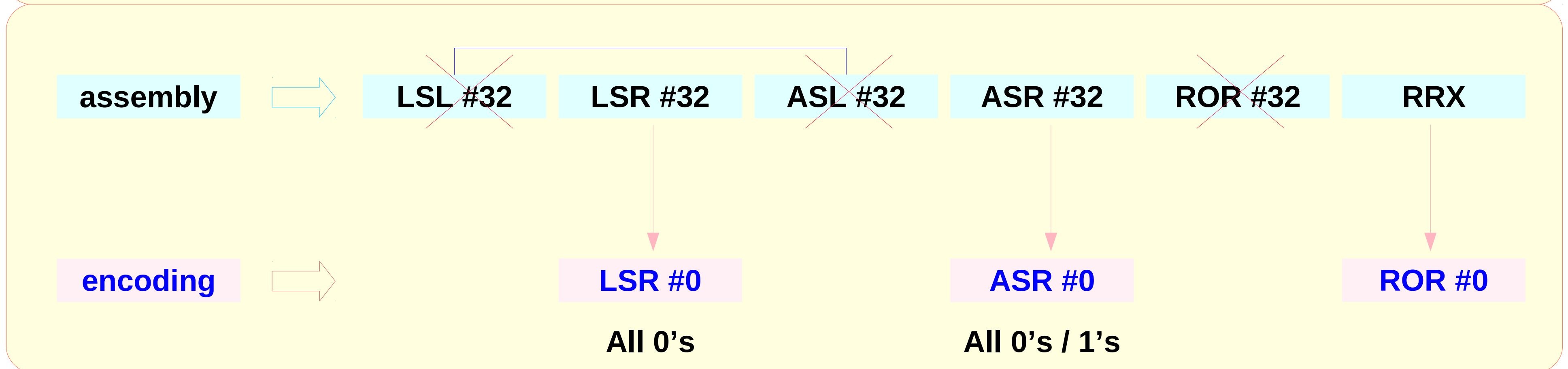
Data Transfer Instructions with <shift>

LDR | STR {<cond>} {**B**} Rd, [Rn, +/–Rm {, **<shift>**}] {!}
LDR | STR {<cond>} {**B**} {T} Rd, [Rn], +/–Rm {, **<shift>**}
LDR | STR {<cond>} **H** | **SH** | **SB** Rd, [Rn, ~~+/–Rm {, **<shift>**}~~] {!}
LDR | STR {<cond>} **H** | **SH** | **SB** Rd, [Rn], ~~+/–Rm {, **<shift>**}~~

only **single** (word / unsigned byte) **data transfer** instructions support
<shift type> #<#shift> … instruction specified shift amount
~~<shift type> Rs~~ … register specified shift amount **not** used

Instruction Specified Shift Amount Encoding

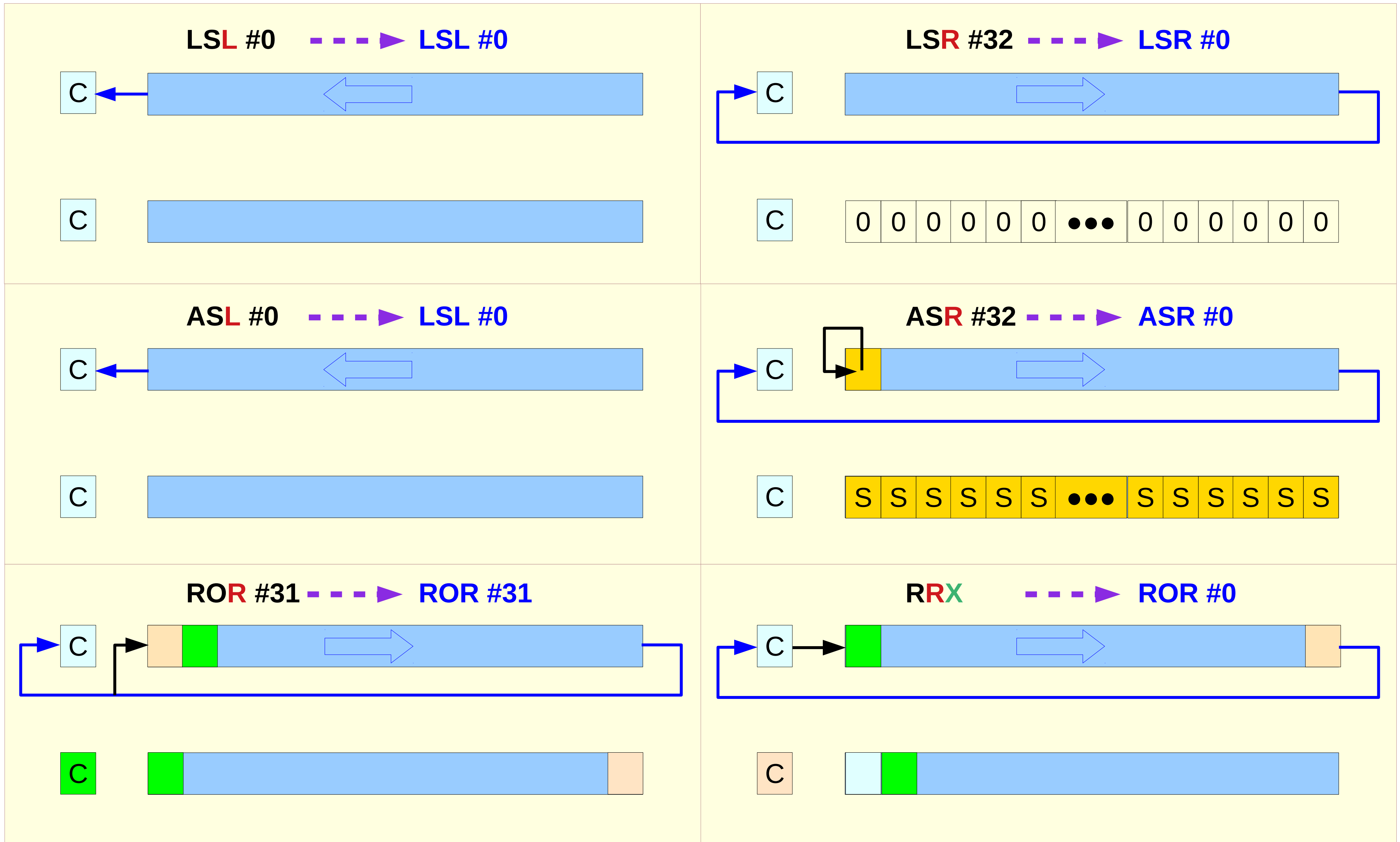




LSL #<0-31>	(Logical Shift Left)
LSR #<1-32>	(Logical Shift Right)
ASL #<0-31>	(Arithmetic Shift Left)
ASR #<1-32>	(Arithmetic Shift Right)
ROR #<1-31>	(Rotate Right)

b. **<shiftname> #expression**

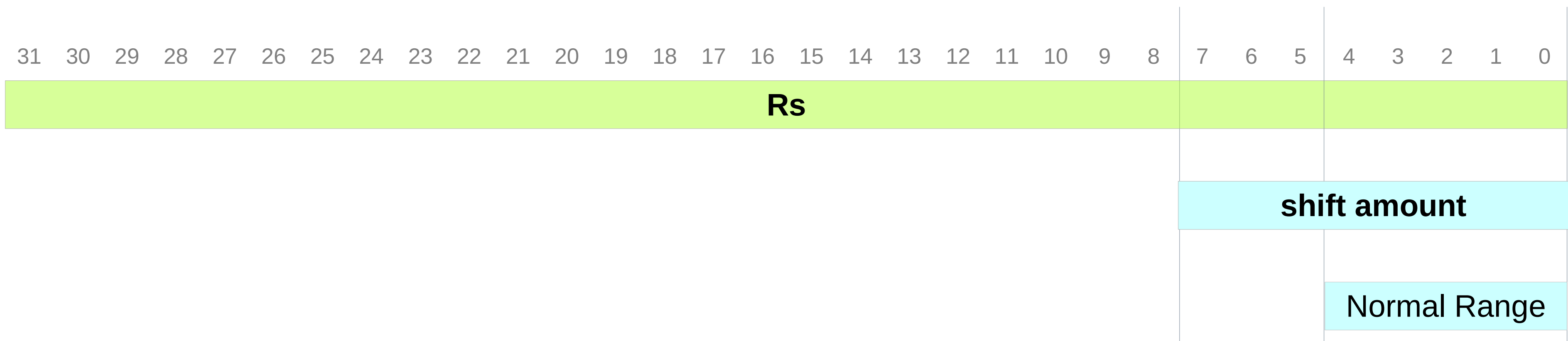
Instruction Specified Shift Amount Example



Shift Amount Register **Rs**

- **R15** cannot be **Rs**
- Only the **LSByte** of **Rs** is used as a **shift amount**.
- If the **LSByte** is zero,
  - the unchanged contents of **Rm** becomes the 2nd operand,
  - and the old CPSR **C** value becomes the shifter carry output
- The normal range is **[1, 31]**
- If the value **>= 32**, a **logical extension** of the shift

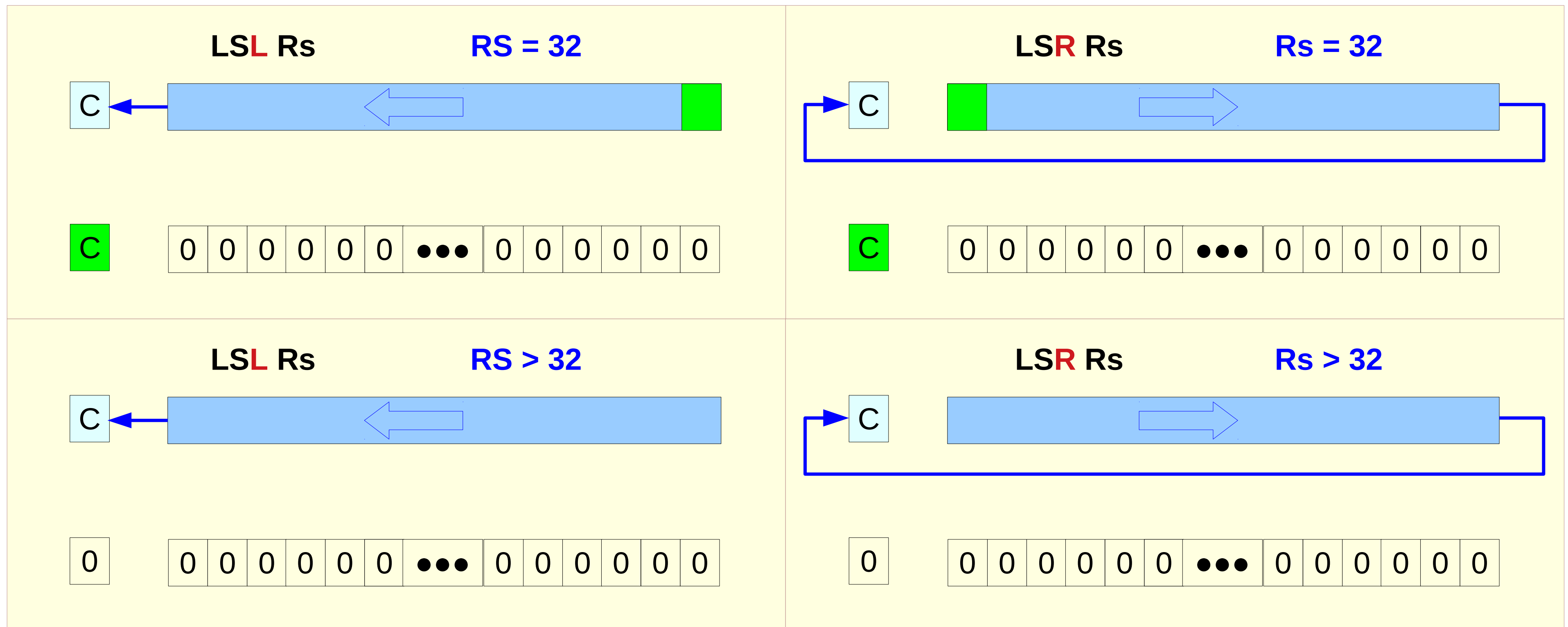
a. **<shiftname> Rs**



LSL & LSR when Rs=32 or Rs>32

LSL by **32** has result **zero**, carry out equal to **bit 0** of Rm.
LSL by **more than 32** has result **zero**, carry out **zero**.

LSR by 32 has result **zero**, carry out equal to **bit 31** of Rm.
LSR by more than 32 has result **zero**, carry out **zero**.

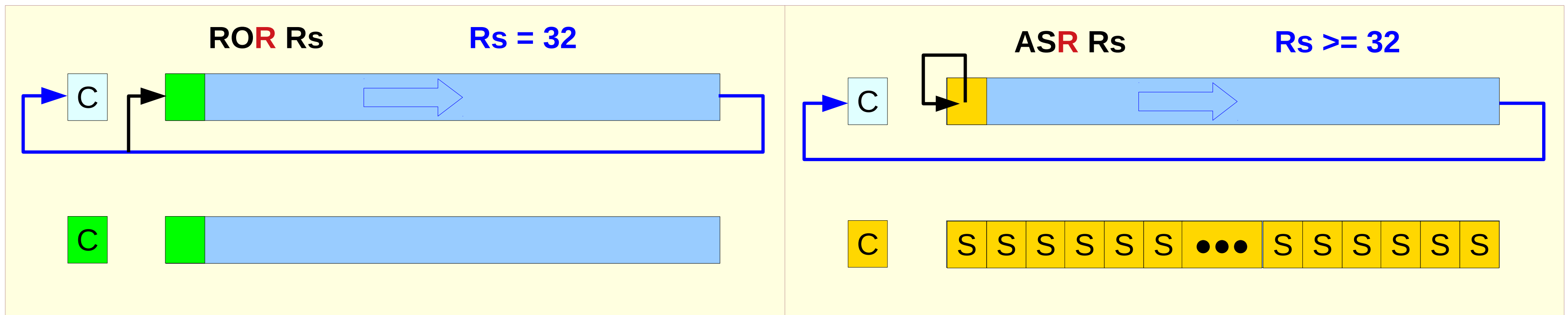


ROR & ASR when Rs=32 or Rs>32

ASR by **32 or more** has result filled with and carry out equal to **bit 31** of Rm.
ROR by **32** has result equal to **Rm**, carry out equal to **bit 31** of Rm.

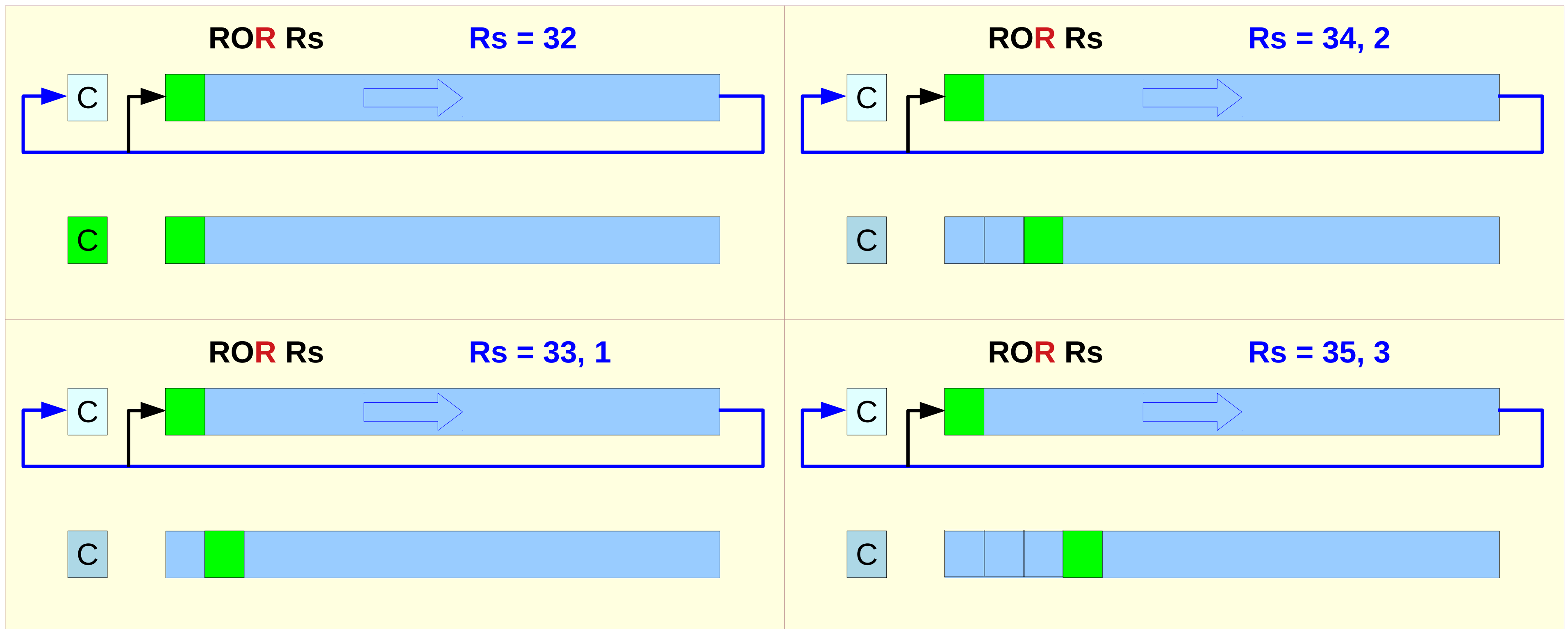
ROR by **n** where n is greater than 32
will give the same result and carry out as **ROR** by **n-32**;
therefore repeatedly subtract **32** from **n**
until the amount is in the range 1 to 32

a **logical extension** of the shift



Logical Extension of ROR

ROR by **n** where n is greater than 32
will give the same result and carry out as **ROR** by **n-32**;
therefore repeatedly subtract **32** from **n**
until the amount is in the range 1 to 32 **ROR (n *mod* 32)**



Data Processing Instructions – Opcodes

Opcode	Mnemonic	Meaning	Effect
0000	AND	Logical bit-wise AND	Rd := Rn AND Op2
0001	EOR	Logical bit-wise exclusive OR	Rd := Rn EOR Op2
0010	SUB	Subtract	Rd := Rn – Op2
0011	RSUB	Reverse subtract	Rd := Op2 – Rn
0100	ADD	Add	Rd := Rn + Op2
0101	ADC	Add with carry	Rd := Rn + Op2 + C
0110	SBC	Subtract with carry	Rd := Rn – Op2 + C –1
0111	RSC	Reverse subtract with carry	Rd := Op2 – Rn + C –1
1000	TST	Test	*set condition codes on* Rn AND Op2
1001	TEQ	Test Equivalence	*set condition codes on* Rn EOR Op2
1010	CMP	Compare	*set condition codes on* Rn – Op2
1011	CMN	Compare Negated	*set condition codes on* Rn + Op2
1100	ORR	Logical Bit-wise OR	Rd := Rn OR Op2
1101	MOV	MOV	Rd := Op2
1110	BIC	Bit Clear	Rd := Rn AND NOT Op2
1111	MVN	Move Negated	Rd := NOT Op2

Data Processing Type 1 – no 1st operand **Rn**

1101	MOV	MOV	Rd := Op2
1111	MVN	Move Negated	Rd := NOT Op2

Rn is not used

can simulate statndalone shift instructions of other machines

```
MOV R0, R0, ASL 2
```

https://community.arm.com/processors/b/blog/posts/condition-codes-1-condition-flags-and-codes

Data Processing Type 2 – no destination **Rd**

1000	TST	Test	*set condition codes on*	Rn AND Op2
1001	TEQ	Test Equivalence	*set condition codes on*	Rn EOR Op2
1010	CMP	Compare	*set condition codes on*	Rn – Op2
1011	CMN	Compare Negated	*set condition codes on*	Rn + Op2

Rd is not used

TST Works like **ANDS**, but does not store the result.
TEQ Works like **EORS**, but does not store the result.
CMP Works like **SUBS**, but does not store the result.
CMN Works like **ADDS**, but does not store the result.

S is implied (set condition codes)

https://community.arm.com/processors/b/blog/posts/condition-codes-1-condition-flags-and-codes

Data Processing Type 3 – Arithmetic & Logical Instructions

0000	AND	Logical bit-wise AND	Rd := Rn AND Op2
0001	EOR	Logical bit-wise exclusive OR	Rd := Rn EOR Op2
0010	SUB	Subtract	Rd := Rn – Op2
0011	RSUB	Reverse subtract	Rd := Op2 – Rn
0100	ADD	Add	Rd := Rn + Op2
0101	ADC	Add with carry	Rd := Rn + Op2 + C
0110	SBC	Subtract with carry	Rd := Rn – Op2 + C –1
0111	RSC	Reverse subtract with carry	Rd := Op2 – Rn + C –1
1100	ORR	Logical Bit-wise OR	Rd := Rn OR Op2

Multiply Instructions – 32 bit products

Producing the least significant 32 bits of products

MUL {<cond>} {S} Rd, Rm, Rs
MLA {<cond>} {S} Rd, Rm, Rs, Rn

```
MUL              Rd, Rm, Rs
MUL  S           Rd, Rm, Rs
MUL     <cond>   Rd, Rm, Rs
MUL  S  <cond>   Rd, Rm, Rs

MLA              Rd, Rm, Rs, Rn
MLA  S           Rd, Rm, Rs, Rn
MLA     <cond>   Rd, Rm, Rs, Rn
MLA  S  <cond>   Rd, Rm, Rs, Rn
```

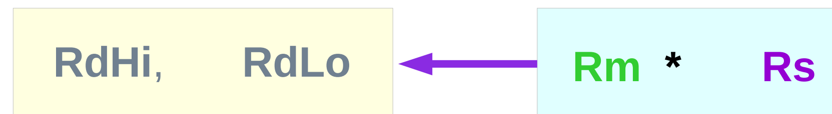
Rd := (Rm * Rs)
Rd := (Rm * Rs + Rn)

Multiply Instructions – 64 bit products

Producing the full 64 bits of products

<mul> {<cond>} {S}   RdHi, RdLo, Rm, Rs

<mul> : UMULL
UMLAL
SMULL
SMLAL



<mul>   RdHi, RdLo, Rm, Rs
<mul>S   RdHi, RdLo, Rm, Rs
<mul> <cond> RdHi, RdLo, Rm, Rs
<mul>S <cond> RdHi, RdLo, Rm, Rs

RdHi:RdLo := Rm * Rs

Multiply Instructions

Opcode [23:21]	Mnemonic	Meaning	Effect
000	MUL	Multiply (32-bit)	Rd := (Rm * Rs)
001	MLA	Multiply-Accumulate (32-bit)	Rd := (Rm * Rs + Rn)
010			
011			
100	UMULL	Unsigned Multiply Long	RdHi:RdLo := Rm * Rs
101	UMLAL	Unsigned Multiply Acc Long	RdHi:RdLo := Rm * Rs
110	SMULL	Singed Multiply Long	RdHi:RdLo := Rm * Rs
111	SMLAL	Unsigned Multiply Acc Long	RdHi:RdLo := Rm * Rs

Status Reg to General Reg Transfer Instructions

Status Register to General Register Transfer Instructions

MRS {<cond>} Rd, CPSR | SPSR

MRS	Rd, CPSR
MRS	Rd, SPSR
MRS <cond>	Rd, CPSR
MRS <cond>	Rd, SPSR

M R ⬅ S

General Reg to Status Reg Transfer Instructions

General Register to Status Register Transfer Instructions

MSR {<cond>} CPSR_f | SPSR_f, #<32-bit immediate>

MSR {<cond>} CPSR_<field> | SPSR_<field>, Rm

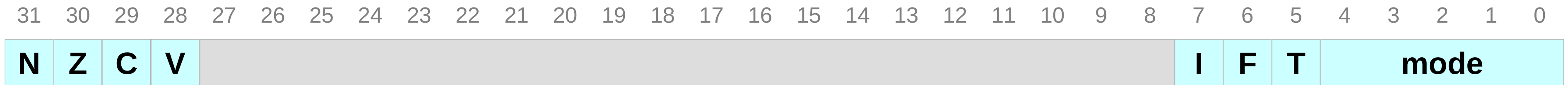
_<field> is one of

_c : the control field	PSR[7: 0]	
_x : the extension field	PSR[15: 8]	(unused on current ARMs)
_s : the status field	PSR[23:16]	(unused on current ARMs)
_f : the flag field	PSR[31:24]	

```
MSR           CPSR_f, #<32-bit immediate>
MSR           SPSR_f, #<32-bit immediate>
MSR <cond>    CPSR_f, #<32-bit immediate>
MSR <cond>    SPSR_f, #<32-bit immediate>
MSR           CPSR_<field>, Rm
MSR           SPSR_<field>, Rm
MSR <cond>    CPSR_<field>, Rm
MSR <cond>    SPSR_<field>, Rm
```

M S ⬅ R

CPSR and SPSR



Current Program Status Register (CPSR)

Saved Program Status Register (SPSR)

N Negative flag

Z Zero flag

C Carry flag

V Overflow flag

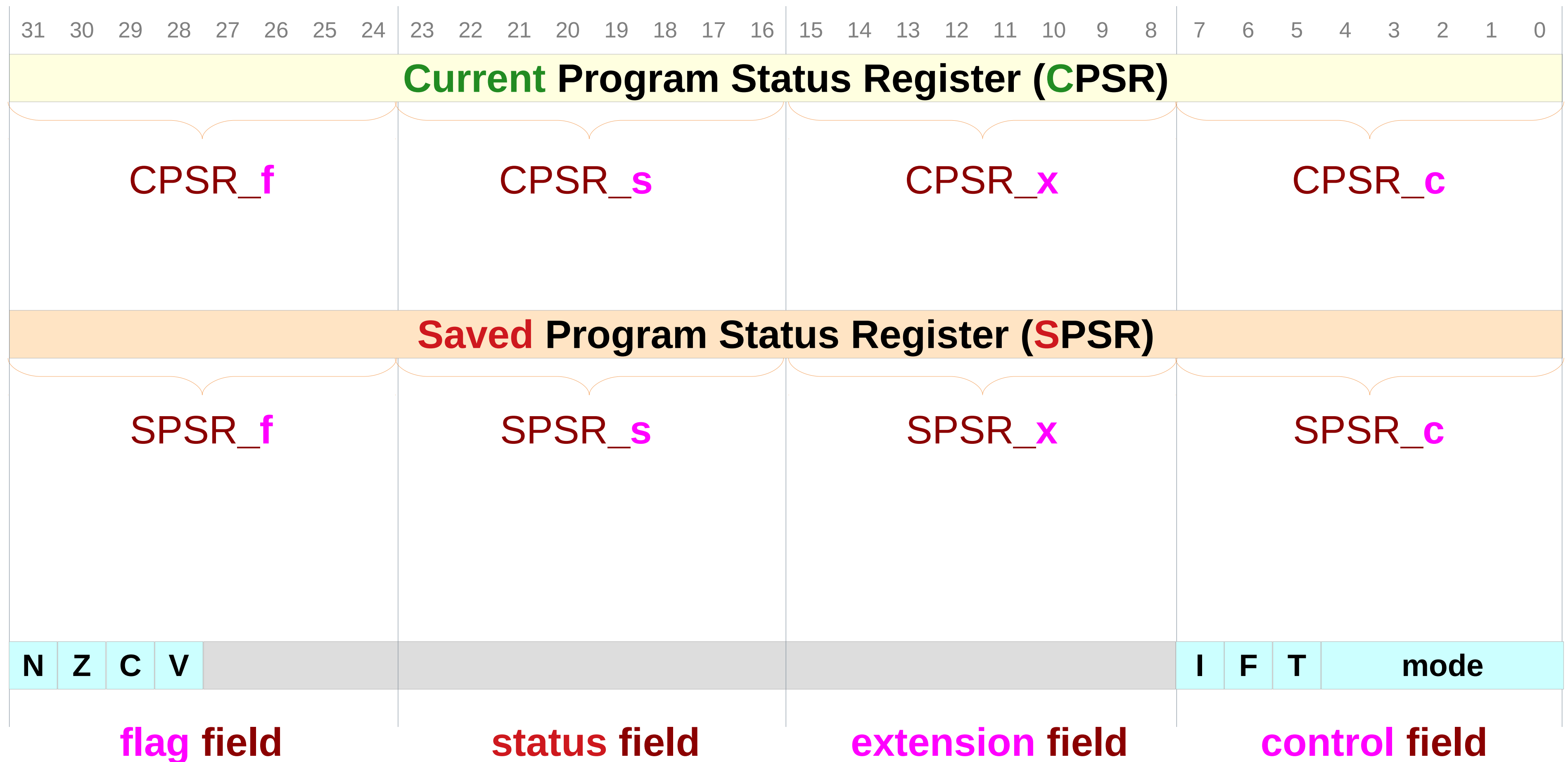
To disable Interrupt (IRQ), set **I**

To disable Fast Interrupt (FIQ), set **F**

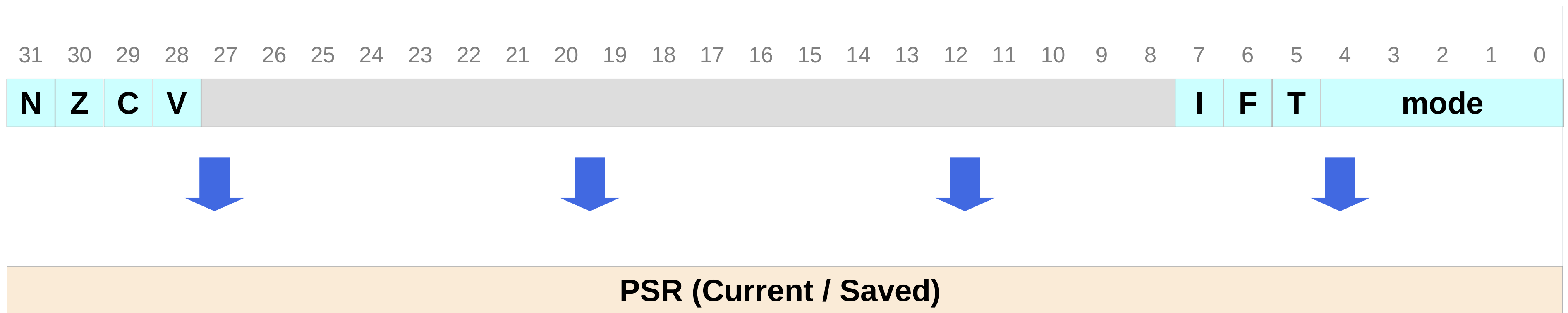
the **T** bit shows running in the Thumb state

Mode					
Usr (usr)	1	0	0	0	0
Fast Interrupt (fiq)	1	0	0	0	1
Interrupt (irq)	1	0	0	1	0
Supervisor (svc)	1	0	0	1	1
Abort (abt)	1	0	1	1	1
Undefined (und)	1	1	0	1	1
System (sys)	1	1	1	1	1

CPSR and SPSR Fields



To a General Reg From a Status Reg

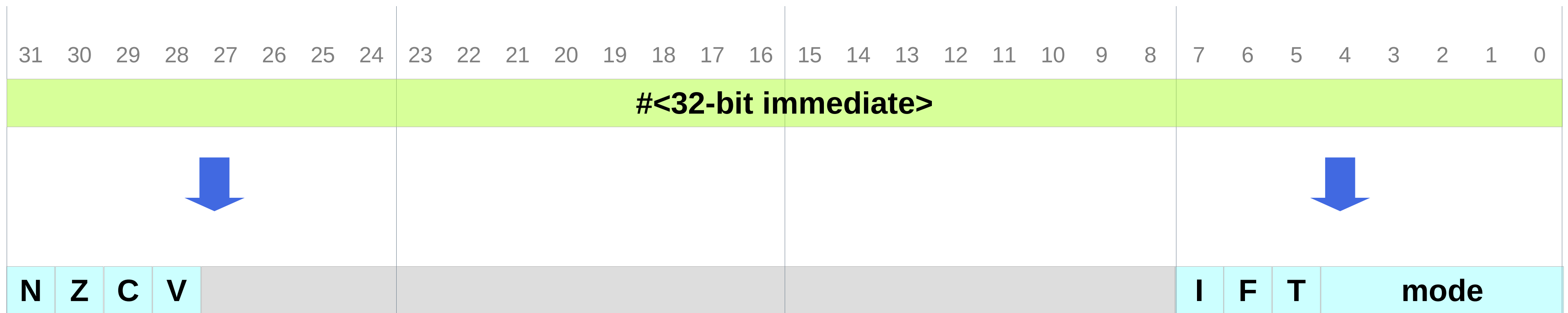


MRS Rd, CPSR
MRS Rd, SPSR

M R ⬅ S

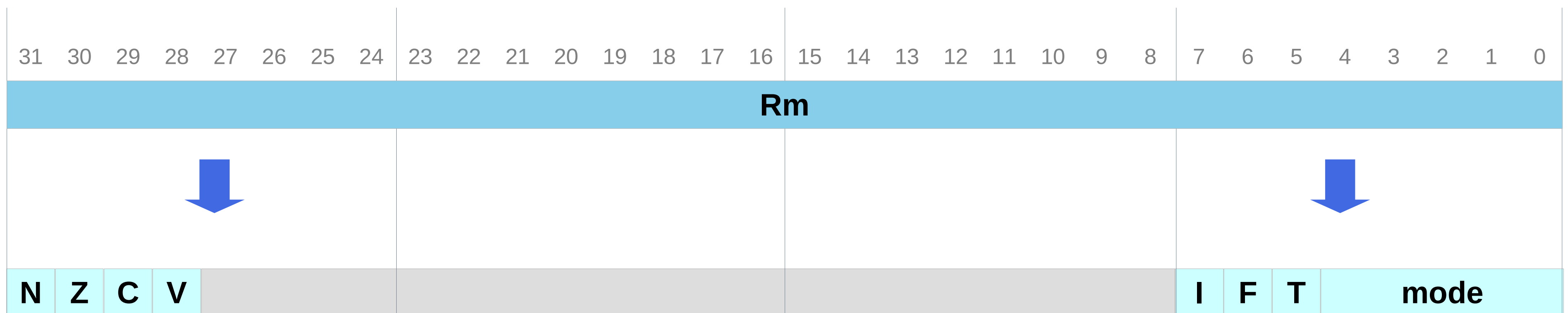
M S ⬅ R

To a Status Reg From a General Reg



MSR CPSR_f , #<32-bit immediate>
MSR SPSR_f , #<32-bit immediate>

MSR CPSR_c , #<32-bit imm>
MSR SPSR_c , #<32-bit imm>



MSR CPSR_f , **Rm**
MSR SPSR_f , **Rm**

MSR CPSR_c , **Rm**
MSR SPSR_c , **Rm**

References

[1] ftp://ftp.geoinfo.tuwien.ac.at/navratil/HaskellTutorial.pdf
[2] https://www.umiacs.umd.edu/~hal/docs/daume02yaht.pdf