ISA Binary Encoding (5A)

Young Won Lim
9/14/19

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

This document was produced by using LibreOffice.

Young Won Lim
9/14/19

Based on

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

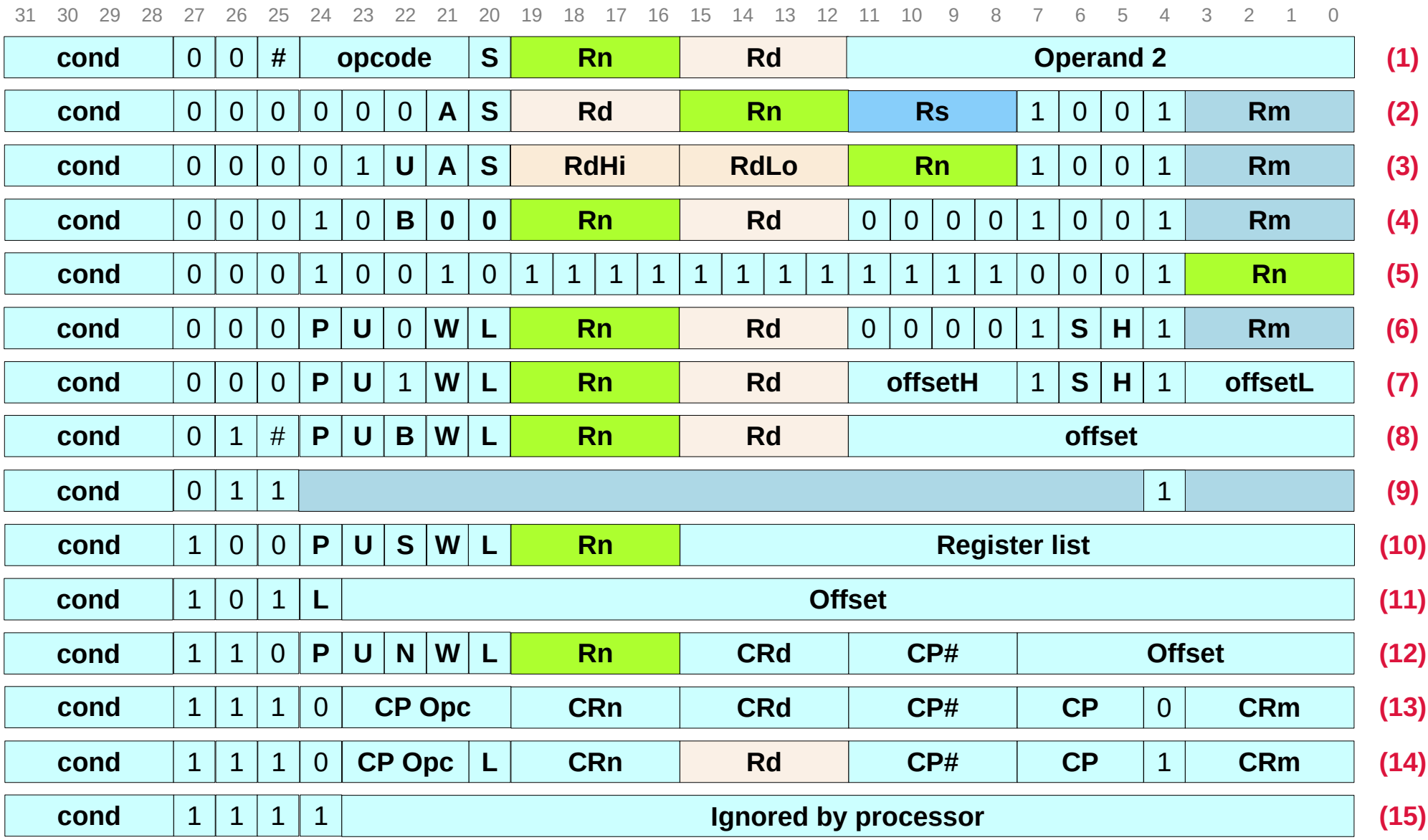
Rn	1st Operand Reg / Base Reg
Rm	2nd Operand Reg / Operand Reg / Offset Reg / Source Reg
Rd	Left most Reg : Source / Destination Reg
S	Set Condition Codes / Signed / Restore PSR and force user bit
P	Pre/Post Index
U	Up/Down
B	Unsigned Byte/Word
H	Half-word Address
L	Link / Load/Store
W	Write-back (auto-index)
Opcode	4-bit op codes
Sh	Shift type
R	CPSR/SPSR

Rn	1st Operand Reg / Base Reg
Rm	2nd Operand Reg / Operand Reg / Offset Reg / Source Reg
Rd	Source / Destination Reg
S	Set Condition Codes / Signed / Restore PSR and force user bit
B	Unsigned Byte/Word
H	Half-word Address
L	Link / Load/Store
T	Selects the user view in the non-usermodes

<cond>	Conditions for conditional execution
<shift>	Shift type and the shift amount (except RRX)
CPSR	Current Program Status Register
SPSR	Saved Program Status Register
RdHi	The most significant 32-bits
RdLo	The least significant 32-bits

<CP#>	Coprocessor number
<Cop1>	Coprocessor operation 1
<Cop2>	Coprocessor operation 2
CRd	Coprocessor Rd
CRn	Coprocessor Rn
CRm	Coprocessor Rm

All listings (1)



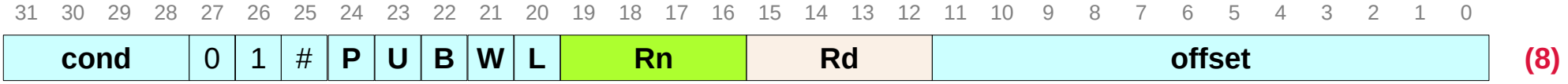
All listings (2)

(1)	Data Processing / PSR Transfer	Rd := Rn <op> operand2 (shifted)
(2)	Multiply	Rd := Rm * Rs + Rn
(3)	Multiply Long	RdHi : RdLo := Rm * Rs
(4)	Single Data Swap	Rd := [Rn]; [Rn]:= Rm
(5)	Branch and Exchange	PC := Rn; (Rn[0]=1 Thumb, else ARM)
(6)	Halfword Data Transfer: register offset	Rd :=: [Rn, Rm] / [Rn], Rm
(7)	Halfword Data Transfer: immediate offset	Rd :=: [Rn, Offset] / [Rn], Offset
(8)	Single Data Transfer	Rd :=: [Rn, Offset] / [Rn], Offset
(9)	Undefined	
(10)	Block Data Transfer	[Rn, i] :=: {Register List}
(11)	Branch	PC := Offset
(12)	Coprocessor Data Transfer	CRd :=: [Rn,Offset] / [Rn], Offset
(13)	Coprocessor Data Operation	CRd :=: CRn <CP Opc, CP> CRm
(14)	Coprocessor Register Transfer	Rd :=: CRn <CP Opc, CP> CRm
(15)	Software Interrupt	

All listings (3)

(1)	Data Processing / PSR Transfer	I, Opcode, S, Rn, Rd, Operand2
(2)	Multiply	A, S, Rd, Rn, Rs, Rm
(3)	Multiply Long	U, A, S, RdHi, RdLo, Rn, Rm
(4)	Single Data Swap	B, Rn, Rd, Rm
(5)	Branch and Exchange	Rm
(6)	Halfword Data Transfer: register offset	P,U, W, L, Rn, Rd, S, H, Rm
(7)	Halfword Data Transfer: immediate offset	P,U, W, L, Rn, Rd, Offset, S, H, Offset
(8)	Single Data Transfer	P, U, B, W, L, Rn, Rd, Offset
(9)	Undefined	
(10)	Block Data Transfer	P, U, S, W, L, Rn, Register List
(11)	Branch	L, Offset
(12)	Coprocessor Data Transfer	P, U, N, W, L, Rn , CRd, CP#, Offset
(13)	Coprocessor Data Operation	CP Opc, CRn, CRd, CP#, CP, CRm
(14)	Coprocessor Register Transfer	CP Opc, L, CRn, Rd, CP# CP, CRm
(15)	Software Interrupt	Software Interrupt

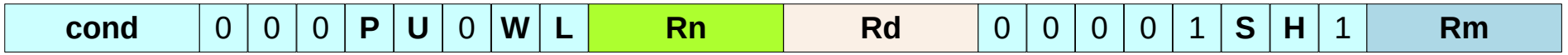
Data Transfer Instructions



(8)

Single Data Transfer

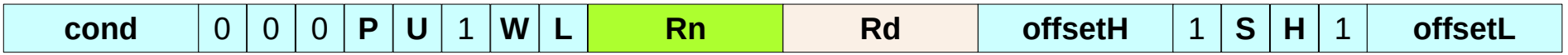
Rd :=: [Rn, Offset] / [Rn], Offset



(6)

Halfword Data Transfer: register offset

Rd :=: [Rn, Rm] / [Rn], Rm



(7)

Halfword Data Transfer: immediate offset

Rd :=: [Rn, Offset] / [Rn], Offset

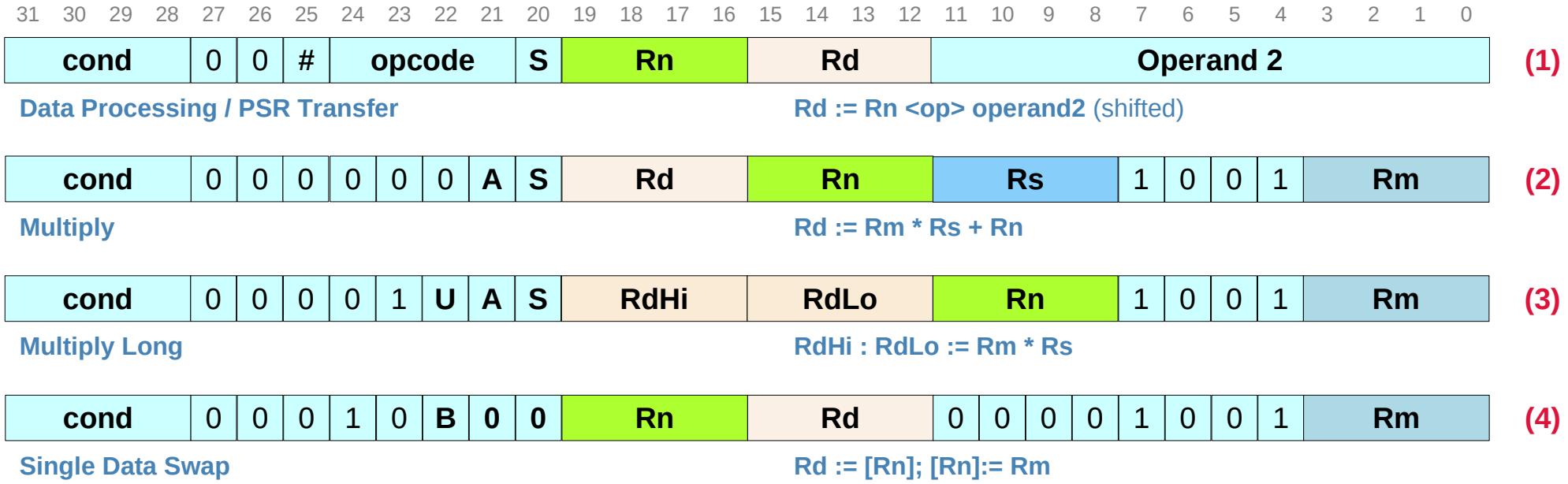
(10)

Block Data Transfer

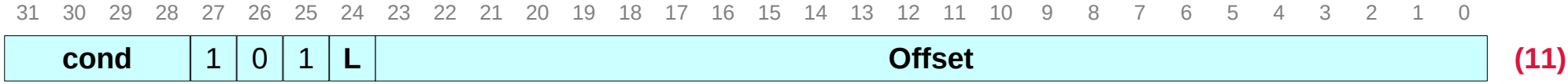
[Rn, i] :=: {Register List}

P Pre/Post Index
U Up/Down
B Unsigned Byte/Word
S Restore PSR and force user bit
W Write-back (auto-index)
L Link / Load/Store

Data Processing Instructions



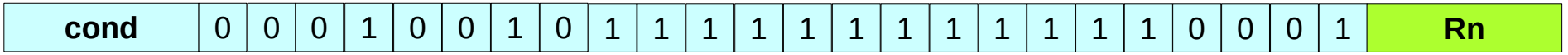
Branch and Branch Exchange



(11)

Branch

PC := Offset

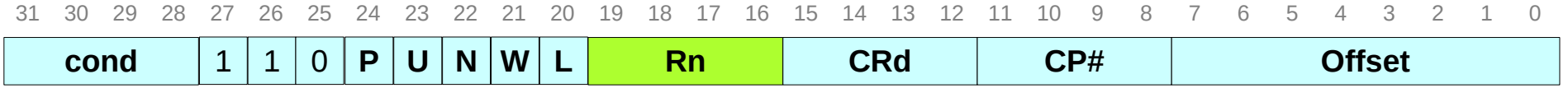


(5)

Branch and Exchange

PC := Rn; (Rn[0]=1 Thumb, else ARM)

Coprocessor Instructions



(12)

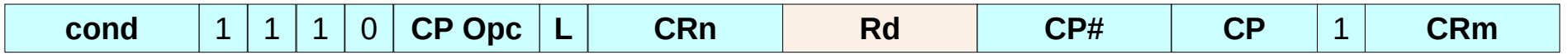
Coprocessor Data Transfer

CRd :=: [Rn,Offset] / [Rn], Offset

(13)

Coprocessor Data Operation

CRd :=: CRn <CP Opc, CP> CRm

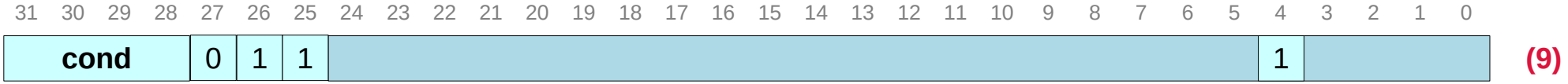


(14)

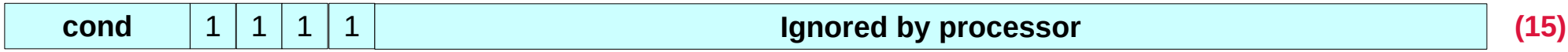
Coprocessor Register Transfer

Rd :=: CRn <CP Opc, CP> CRm

Miscellaneous Instructions



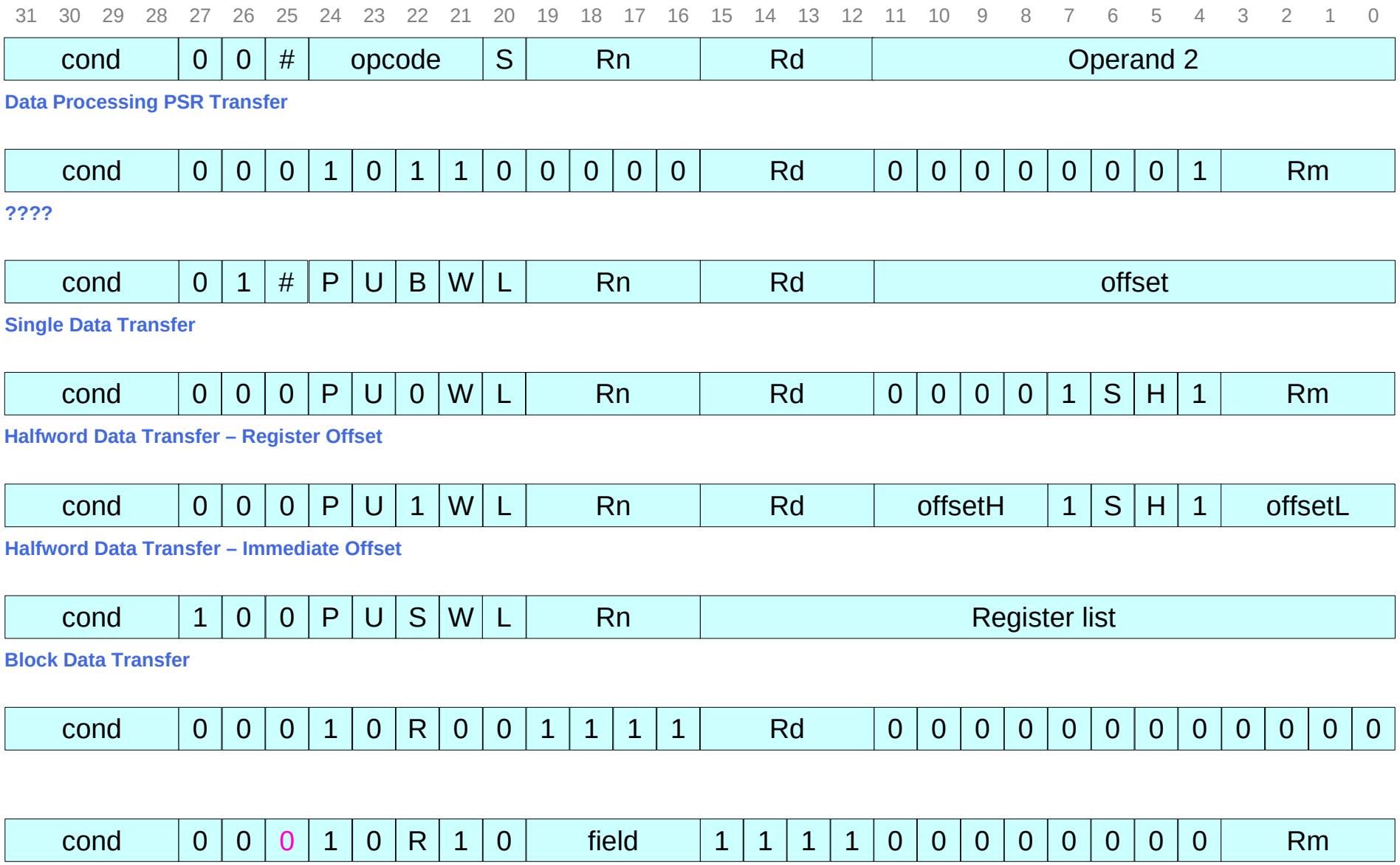
Undefined



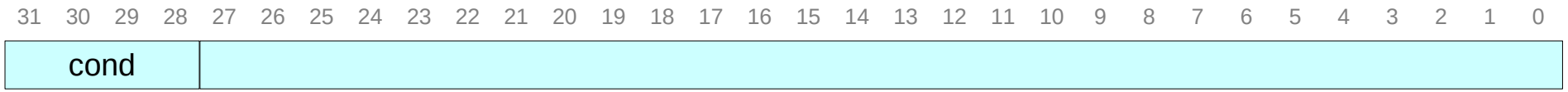
Software Interrupt

Rn	1st Operand Reg / Base Reg
Rd	Source/Destination Reg
Rm	2nd Operand Reg / Source Reg / Offset Reg / Operand Reg

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
cond				0	0	#	opcode				S	Rn				Rd				Operand 2											
cond				0	0	0	1	0	1	1	0	0	0	0	0	Rd				0	0	0	0	0	0	0	1	Rm			
cond				0	1	#	P	U	B	W	L	Rn				Rd				offset											
cond				0	0	0	P	U	#	W	L	Rn				Rd				offsetH				1	S	H	1	offsetL			
cond				1	0	0	P	U	S	W	L	Rn				Register list															
cond				0	0	0	1	0	B	0	0	Rn				Rd				0	0	0	0	1	0	0	1	Rm			
cond				0	0	0	1	0	R	0	0	1	1	1	1	Rd				0	0	0	0	0	0	0	0	0	0	0	0
cond				0	0	0	1	0	R	1	0	field				1	1	1	1	0	0	0	0	0	0	0	0	Rm			


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
cond 0 0 # opcode S Rn Rd Operand 2
Data Processing PSR Transfer
cond 0 0 0 1 0 1 1 0 0 0 0 0 Rd 0 0 0 0 0 0 0 1 Rm
????
cond 0 1 # P U B W L Rn Rd offset
Single Data Transfer
cond 0 0 0 P U 0 W L Rn Rd 0 0 0 0 1 S H 1 Rm
Halfword Data Transfer – Register Offset
cond 0 0 0 P U 1 W L Rn Rd offsetH 1 S H 1 offsetL
Halfword Data Transfer – Immediate Offset
cond 1 0 0 P U S W L Rn Register list
Block Data Transfer
cond 0 0 0 1 0 R 0 0 1 1 1 1 Rd 0 0 0 0 0 0 0 0 0 0 0 0
cond 0 0 0 1 0 R 1 0 field 1 1 1 1 0 0 0 0 0 0 0 0 Rm

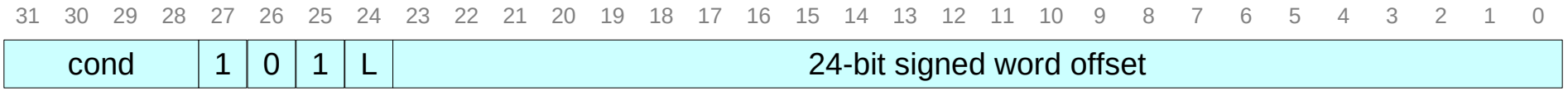
Condition Code



ARM Condition Codes

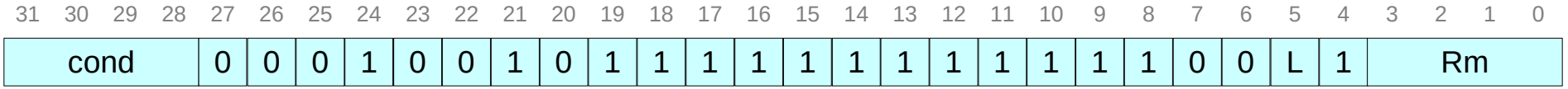
31	30	29	28			
0	0	0	0	EQ	EQual / EQuals zero	Z ← 1
0	0	0	1	NE	Not Equal	Z ← 0
0	0	1	0	CS/HS	Carry Set / unsigned High or Same	C ← 1
0	0	1	1	CC/LO	Carry Clear / unsigned Lower	C ← 0
0	1	0	0	MI	MInus / negative	N ← 1
0	1	0	1	PL	PLus / positive or zero	N ← 0
0	1	1	0	VS	oVerflow Set	V ← 1
0	1	1	1	VC	oVerflow Clear	V ← 0
1	0	0	0	HI	unsigned HIgher	C ← 1, Z ← 0
1	0	0	1	LS	unsigned Lower or Same	C ← 0, Z ← 1
1	0	1	0	GE	signed Greater than or Equal	N == V
1	0	1	1	LT	signed Less Than	N != V
1	1	0	0	GT	signed Greater Than	Z ← 0, N==V
1	1	0	1	LE	signed Less than or Equal	Z ← 1, N!=V
1	1	1	0	AL	ALways	any
1	1	1	1	NV	NeVer (do not use?)	none

Branch and Branch with Link (B, BL)

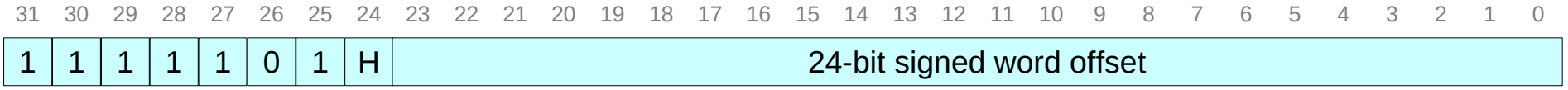


Branch and Branch with Link

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

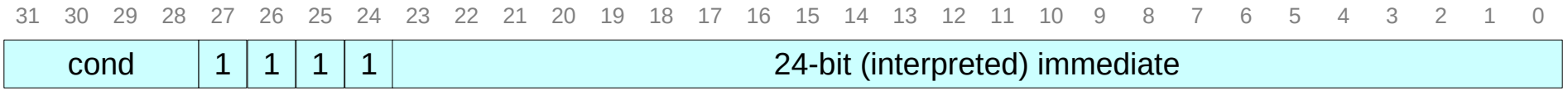


Branch with Link an eXchange



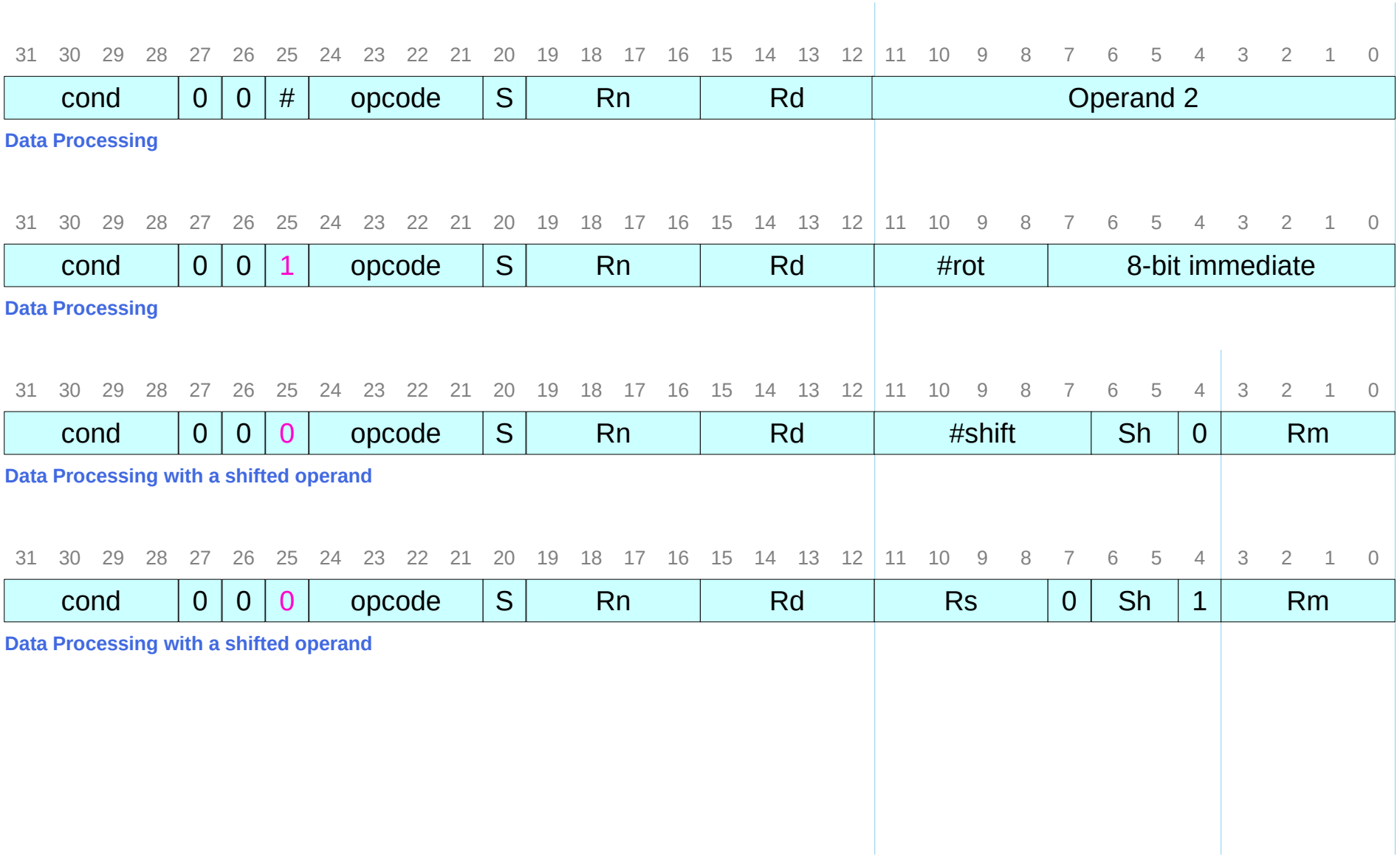
Branch with Link an eXchange

SWI (Software Interrupt)

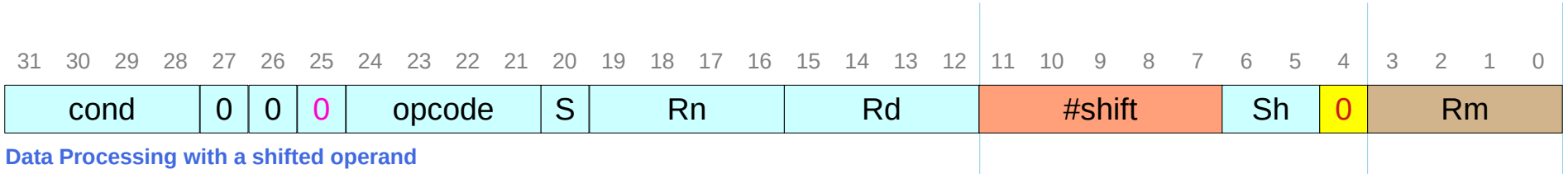


SWI

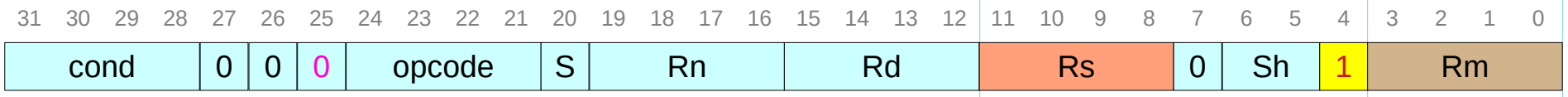
Data Processing Instructions



Data Processing Instructions – shift

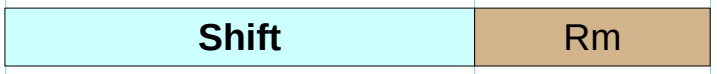


Data Processing with a shifted operand

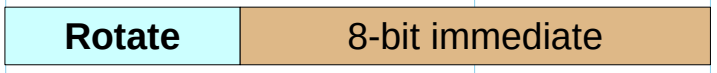


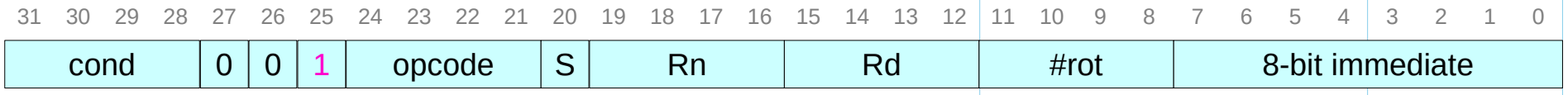
Data Processing with a shifted operand

```
ADD  r3, r2, r1, LSL #3     ; r3 := r2 + r1*2^3
ADD  r5, r5, r3, LSL r2     ; r5 := r5 + r3*2^r2
MOV r0,      r0, LSR #2     ; r0 :=      r0 / 2^2
```



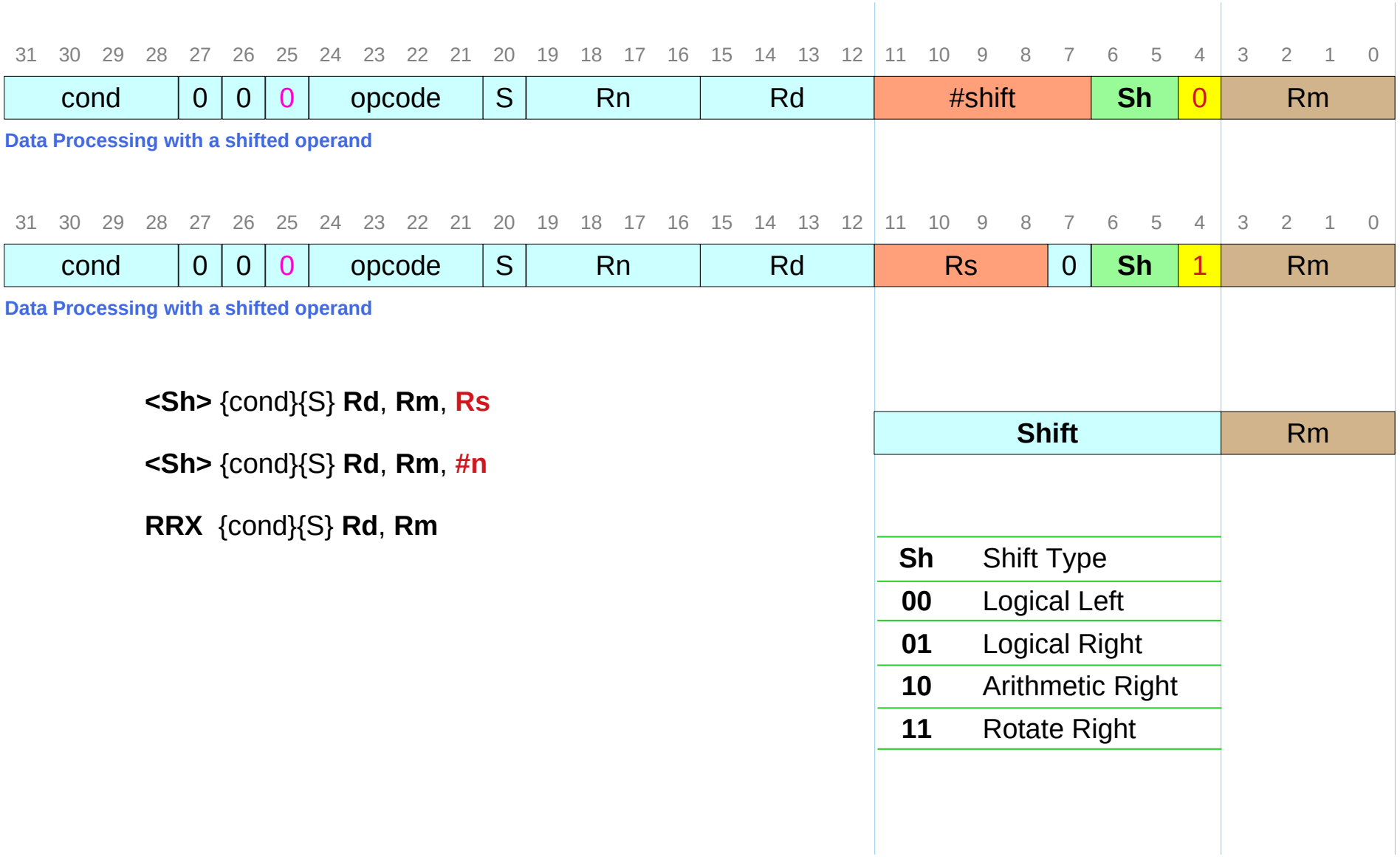

```
0xA_05 // rotate 0x05 right by 20 (=2 * 0xA)
0xB_14 // rotate 0x14 right by 22 (=2 * 0xB)
0xC_50 // rotate 0x50 right by 24 (=2 * 0xC)
```





Data Processing with a shifted operand

Data Processing Instructions – shift



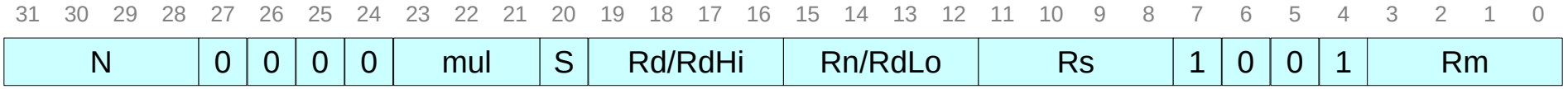
Data Processing Instructions

31	30	29	28	27	26	25	24	23	22	21	20
cond				0	0	#	0000				S
cond				0	0	#	0001				S
cond				0	0	#	0010				S
cond				0	0	#	0011				S
cond				0	0	#	0100				S
cond				0	0	#	0101				S
cond				0	0	#	0110				S
cond				0	0	#	0111				S
cond				0	0	#	1000				S
cond				0	0	#	1001				S
cond				0	0	#	1010				S
cond				0	0	#	1011				S
cond				0	0	#	1100				S
cond				0	0	#	1101				S
cond				0	0	#	1110				S
cond				0	0	#	1111				S

31	30	29	28	27	26	25	24	23	22	21	20
cond				0	0	0	opcode				S

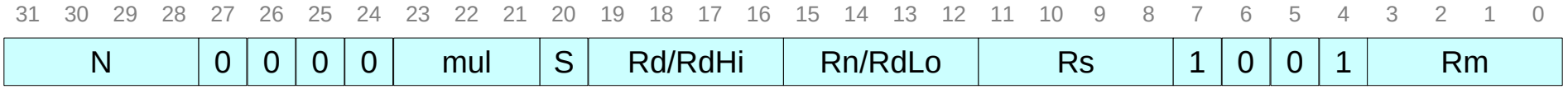
AND	Logical bit-wise AND	Rd := Rn **AND** Op2
EOR	Logical bit-wise XOR	Rd := Rn **EOR** Op2
SUB	Subtract	Rd := Rn **–** Op2
RSB	Reverse Subtract	Rd := Op2 **–** Rn
ADD	Add	Rd := Rn **+** Op2
ADC	Add with carry	Rd := Rn **+** Op2 **+** C
SBC	Subtract with carry	Rd := Rn **–** Op2 **+** C **–** 1
RSC	Reverse subtract with carry	Rd := Op2 **–** Rn **+** C **–** 1
TST	Test	Rn **AND** Op2
TEQ	Test equivalence	Rn **EOR** Op2
CMP	Compare	Rn **–** Op2
CMN	Compare negated	Rn **+** Op2
ORR	Logical bit-wise OR	Rd := Rn **OR** Op2
MOV	Move	Rd := Op2
BIC	Bit clear	Rd := Rn **AND NOT** Op2
MVN	Nive begated	Rd := **NOT** Op2

Multiply Instructions



Multiply

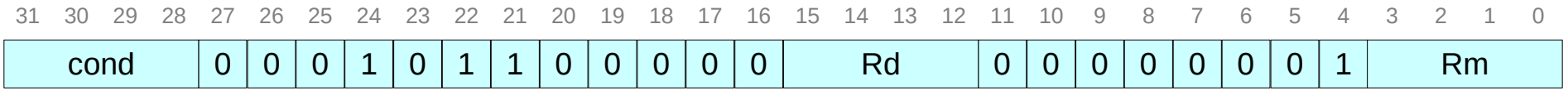
Multiply Instructions



Multiply

0	0	0	MUL	Multiply (32-bit result)	Rd := (Rm * Rs)[31:0]
0	0	1	MLA	Multiply-accumulate (32-bit result)	Rd := (Rm * Rs)[31:0]
1	0	0	UMULL	Unsigned multiply long	RdHi.RdLo := Rm * Rs
1	0	1	UMLAL	Unsigned multiply-accumulate long	RdHi.RdLo += Rm * Rs
1	1	0	SMULL	Signed multiply long	RdHi.RdLo := Rm * Rs
1	1	1	SMLAL	Signed multiply-accumulate long	RdHi.RdLo += Rm * Rs

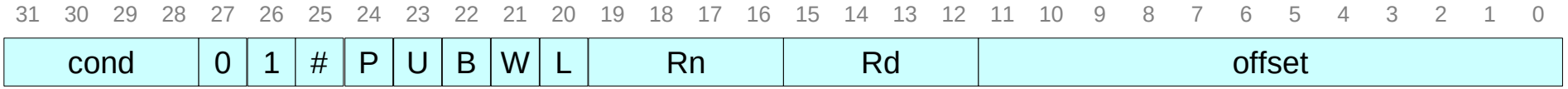
CLZ (Count leading zeros)

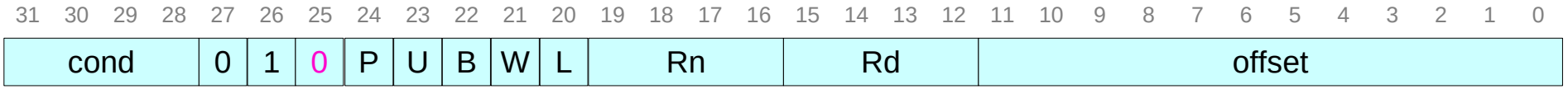


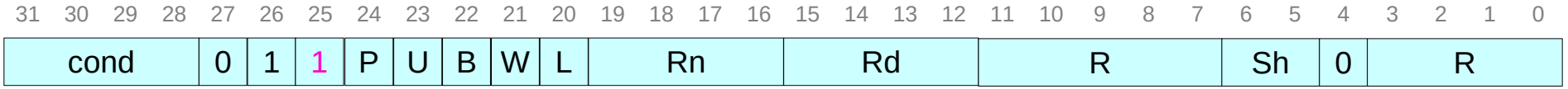
CLZ

Data Transfer Instructions

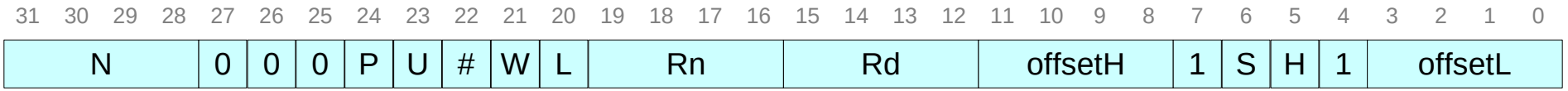
Single word and unsigned byte data transfer instructions

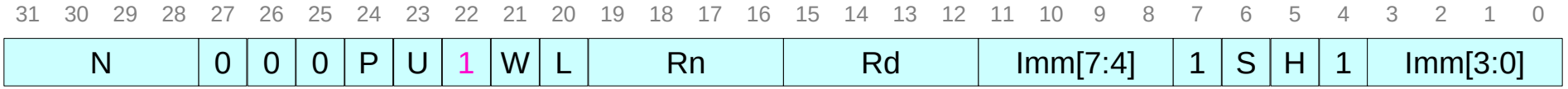


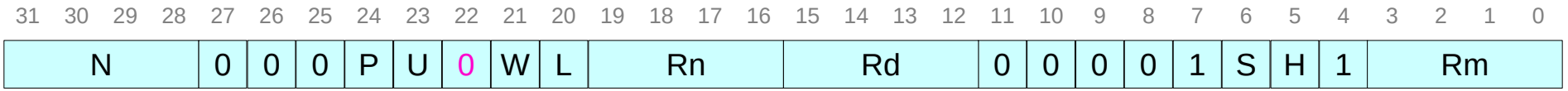




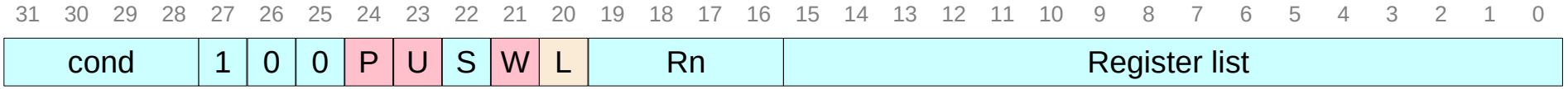
Half-word and signed byte data transfer instructions







Multiple register transfer instruction



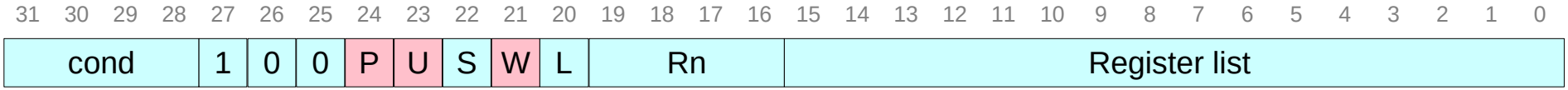
P=1 : before **U=1** : Increment **W=1** : auto index **L=1** : **LDM**
P=0 : after **U=0** : Decrement **W=0** : no auto index **L=0** : **STM**

P : Pre/Post **U : Up/Down** **W : Write-back (auto index)**

IB	**PU=11**	**DA**	**PU=00**
IA	**PU=01**	**IA**	**PU=01**
DB	**PU=10**	**DB**	**PU=10**
DA	**PU=00**	**IB**	**PU=11**

Name	Stack	Block	L	P	U
Pre-Increment Load	**LDMED**	**LDMIB**	1	1	1
Post-Increment Load	**LDMFD**	**LDMIA**	1	0	1
Pre-Decrement Load	**LDMEA**	**LDMDB**	1	1	0
Post-Decrement Load	**LDMFA**	**LDMDA**	1	0	0
Pre-Increment Store	**STMFA**	**STMIB**	0	1	1
Post-Increment Store	**STMEA**	**STMIA**	0	0	1
Pre-Decrement Store	**STMFD**	**STMDB**	0	1	0
Post-Decrement Store	**STMED**	**STMDA**	0	0	0

Multiple register transfer instruction

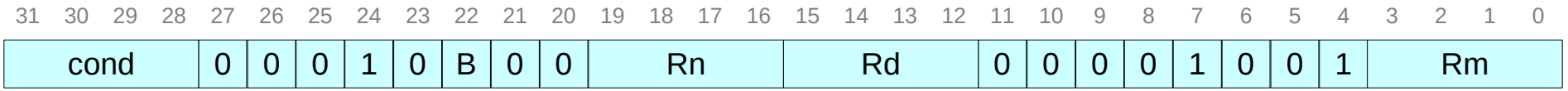



```
STMIB R8! {R0,R1,R4}
STMFA R8! {R0,R1,R4}

STMIA R8! {R0,R1,R4}
STMEA R8! {R0,R1,R4}

STMDB R8! {R0,R1,R4}
STMFD R8! {R0,R1,R4}

STMDA R8! {R0,R1,R4}
STMED R8! {R0,R1,R4}
```

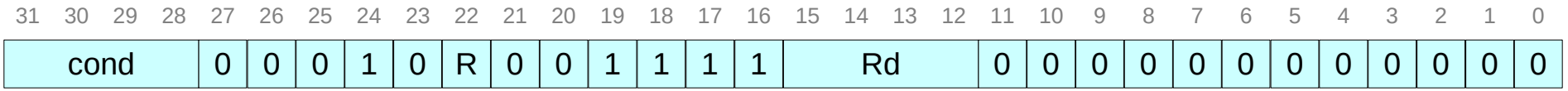
```
LDMDA R8! {R0,R1,R4}
LDMFA R8! {R0,R1,R4}

LDMDB R8! {R0,R1,R4}
LDMEA R8! {R0,R1,R4}

LDMIA R8! {R0,R1,R4}
LDMFD R8! {R0,R1,R4}

LDMIB R8! {R0,R1,R4}
LDMED R8! {R0,R1,R4}
```

```
IB   PU=11
IA   PU=01
DB   PU=10
DA   PU=00
```

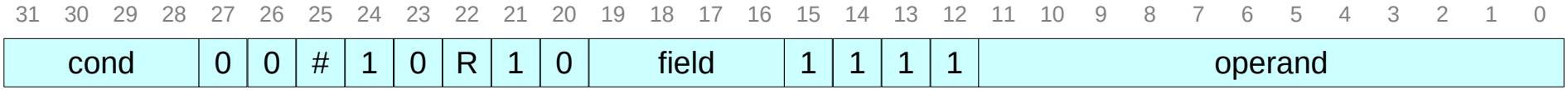
Swap memory and register instruction (SWP)

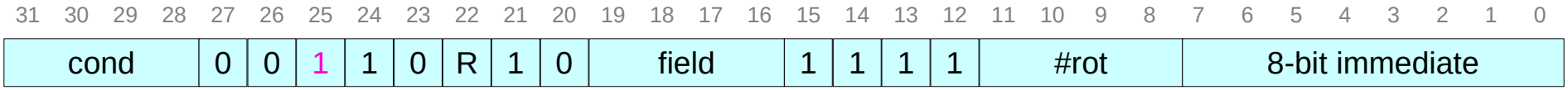


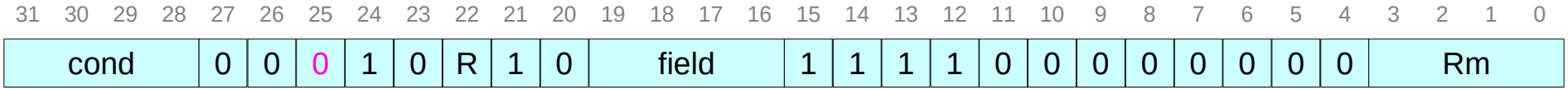
Status register to general register transfer instructions



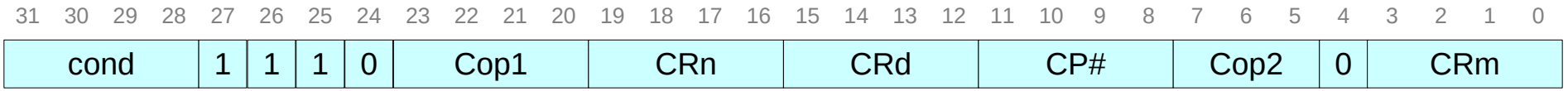
General register to status register transfer instructions



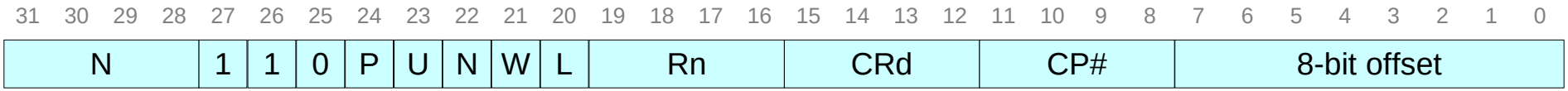




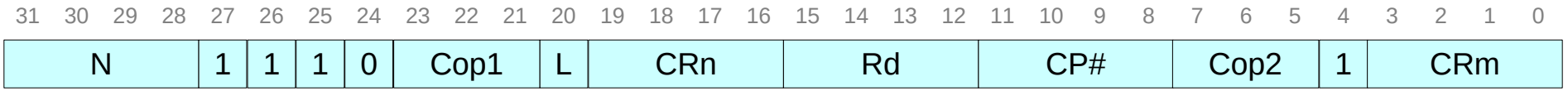
Coprocessor data operations



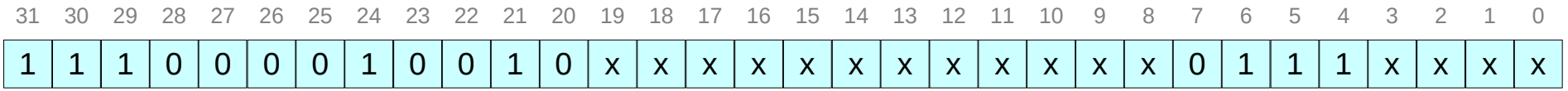
Coprocessor data transfers



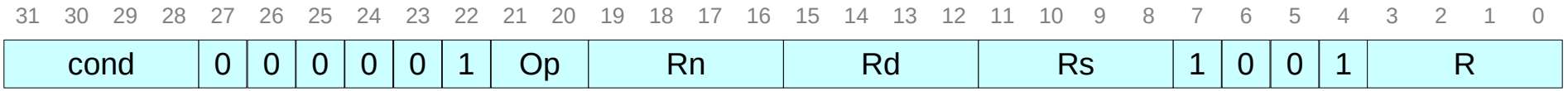
Coprocessor register transfer



Breakpoint instruction (BKPT)

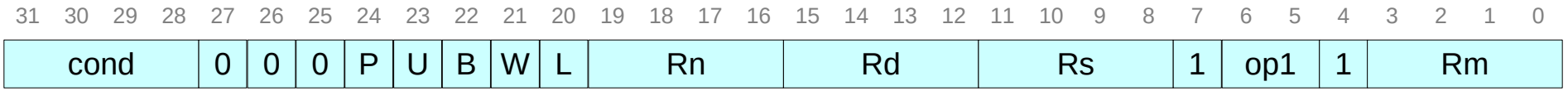


Unused arithmetic instructions

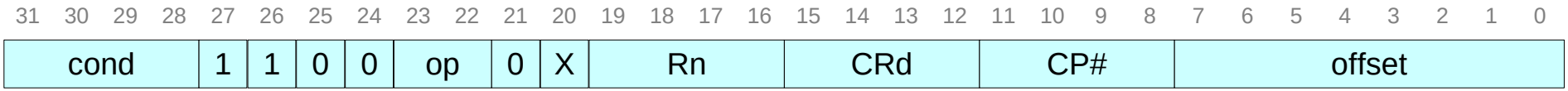


Unused control instructions

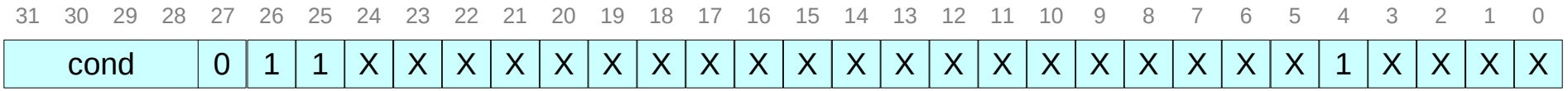
Unused load/store instructions



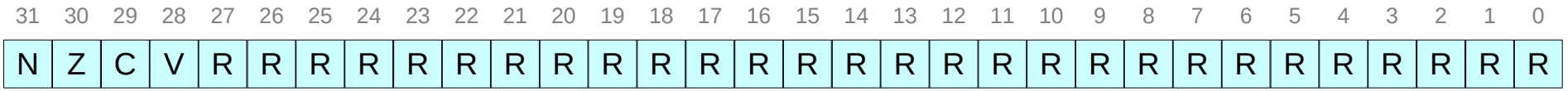
Unused coprocessor instructions



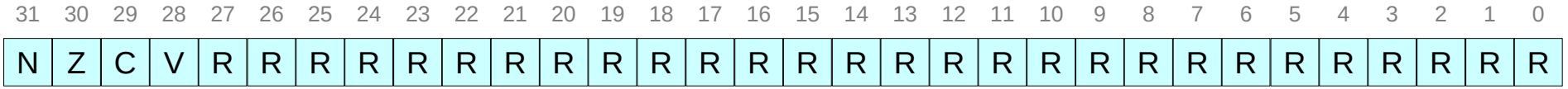
Undefined instruction space



Multiple register transfer instruction



ARM Exception Handling



References

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