B2

APPENDIX

ARM and Thumb Instruction Encodings

Andrew Sloss,
ARM; *Dominic Symes,* ARM;
Chris Wright,
Ultimodule Inc.

This appendix gives tables for the instruction set encodings of the 32-bit ARM and 16-bit Thumb instruction sets. We also describe the fields of the processor status registers *cpsr* and *spsr*.

B2.1 ARM Instruction Set Encodings

Table B2.1 summarizes the bit encodings for the 32-bit ARM instruction set architecture ARMv6. This table is useful if you need to decode an ARM instruction by hand. We've expanded the table to aid quick manual decode. Any bitmaps not listed are either unpredictable or undefined for ARMv6.

To use Table B2.1 efficiently, follow this decoding procedure:

- Look at the leading hex digit of the instruction, bits 28 to 31. If this has a value `0xF`, then jump to the end of Table B2.1. Otherwise, the top hex digit represents a condition *cond*. Decode *cond* using Table B2.2.

TABLE B2.1 ARM instruction decode table.

Instruction classes (indexed by *op*)	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
AND | EOR | SUB | RSB | ADD | ADC | SBC | RSC	*cond*				0	0	0	0	*op*			*S*	*Rn*				*Rd*				*shift_size*				*shift*		0	*Rm*			
AND | EOR | SUB | RSB | ADD | ADC | SBC | RSC	*cond*				0	0	0	0	*op*			*S*	*Rn*				*Rd*				*Rs*				0	*shift*		1	*Rm*			
MUL	*cond*				0	0	0	0	0	0	0	*S*	*Rd*				0	0	0	0	*Rs*				1	0	0	1	*Rm*			
MLA	*cond*				0	0	0	0	0	0	1	*S*	*Rd*				*Rn*				*Rs*				1	0	0	1	*Rm*			
UMAAL	*cond*				0	0	0	0	0	1	0	0	*RdHi*				*RdLo*				*Rs*				1	0	0	1	*Rm*			
UMULL | UMLAL | SMULL | SMLAL	*cond*				0	0	0	0	1	*op*		*S*	*RdHi*				*RdLo*				*Rs*				1	0	0	1	*Rm*			
STRH | LDRH *post*	*cond*				0	0	0	0	*U*	0	0	*op*	*Rn*				*Rd*				0	0	0	0	1	0	1	1	*Rm*			
STRH | LDRH *post*	*cond*				0	0	0	0	*U*	1	0	*op*	*Rn*				*Rd*				*immed* [7:4]				1	0	1	1	*immed* [3:0]			
LDRD | STRD | LDRSB | LDRSH *post*	*cond*				0	0	0	0	*U*	0	0	*op*	*Rn*				*Rd*				0	0	0	0	1	1	*op*	1	*Rm*			
LDRD | STRD | LDRSB | LDRSH *post*	*cond*				0	0	0	0	*U*	1	0	*op*	*Rn*				*Rd*				*immed* [7:4]				1	1	*op*	1	*immed* [3:0]			
MRS Rd, cpsr | MRS Rd, spsr	*cond*				0	0	0	1	0	*op*	0	0	1	1	1	1	*Rd*				0	0	0	0	0	0	0	0	0	0	0	0
MSR cpsr, Rm | MSR spsr, Rm	*cond*				0	0	0	1	0	*op*	1	0	*f*	*s*	*x*	*c*	1	1	1	1	0	0	0	0	0	0	0	0	*Rm*			
BXJ	*cond*				0	0	0	1	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	0	*Rm*			
SMLAxy	*cond*				0	0	0	1	0	0	0	0	*Rd*				*Rn*				*Rs*				1	*y*	*x*	0	*Rm*			
SMLAWy	*cond*				0	0	0	1	0	0	1	0	*Rd*				*Rn*				*Rs*				1	*y*	0	0	*Rm*			
SMULWy	*cond*				0	0	0	1	0	0	1	0	*Rd*				0	0	0	0	*Rs*				1	*y*	1	0	*Rm*			
SMLALxy	*cond*				0	0	0	1	0	1	0	0	*RdHi*				*RdLo*				*Rs*				1	*y*	*x*	0	*Rm*			
SMULxy	*cond*				0	0	0	1	0	1	1	0	*Rd*				0	0	0	0	*Rs*				1	*y*	*x*	0	*Rm*			
TST | TEQ | CMP | CMN	*cond*				0	0	0	1	0	*op*		1	*Rn*				0	0	0	0	*shift_size*				*shift*		0	*Rm*			
ORR | BIC	*cond*				0	0	0	1	1	*op*	0	*S*	*Rn*				*Rd*				*shift_size*				*shift*		0	*Rm*			
MOV | MVN	*cond*				0	0	0	1	1	*op*	1	*S*	0	0	0	0	*Rd*				*shift_size*				*shift*		0	*Rm*			
BX | BLX	*cond*				0	0	0	1	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	0	0	*op*	1	*Rm*			
CLZ	*cond*				0	0	0	1	0	1	1	0	1	1	1	1	*Rd*				1	1	1	1	0	0	0	1	*Rm*			
QADD | QSUB | QDADD | QDSUB	*cond*				0	0	0	1	0	*op*		0	*Rn*				*Rd*				0	0	0	0	0	1	0	1	*Rm*			
BKPT	1	1	1	0	0	0	0	1	0	0	1	0	*immed*[15:4]												0	1	1	1	*immed* [3:0]			
TST | TEQ | CMP | CMN	*cond*				0	0	0	1	0	*op*		1	*Rn*				0	0	0	0	*Rs*				0	*shift*		1	*Rm*			
ORR | BIC	*cond*				0	0	0	1	1	*op*	0	*S*	*Rn*				*Rd*				*Rs*				0	*shift*		1	*Rm*			
MOV | MVN	*cond*				0	0	0	1	1	*op*	1	*S*	0	0	0	0	*Rd*				*Rs*				0	*shift*		1	*Rm*			
SWP | SWPB	*cond*				0	0	0	1	0	*op*	0	0	*Rn*				*Rd*				0	0	0	0	1	0	0	1	*Rm*			
STREX	*cond*				0	0	0	1	1	0	0	0	*Rn*				*Rd*				1	1	1	1	1	0	0	1	*Rm*			
LDREX	*cond*				0	0	0	1	1	0	0	1	*Rn*				*Rd*				1	1	1	1	1	0	0	1	1	1	1	1

TABLE B2.1 ARM instruction decode table. (*Continued*.)

Instruction classes (indexed by *op*)	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
STRH | LDRH *pre*	*cond*	0 0 0 1	*U* 0 *W* *op*	*Rn*	*Rd*	0 0 0 0	1 0 1 1	*Rm*
STRH | LDRH *pre*	*cond*	0 0 0 1	*U* 1 *W* *op*	*Rn*	*Rd*	*immed* [7:4]	1 0 1 1	*immed* [3:0]
LDRD | STRD | LDRSB | LDRSH *pre*	*cond*	0 0 0 1	*U* 0 *W* *op*	*Rn*	*Rd*	0 0 0 0	1 1 *op* 1	*Rm*
LDRD | STRD | LDRSB | LDRSH *pre*	*cond*	0 0 0 1	*U* 1 *W* *op*	*Rn*	*Rd*	*immed* [7:4]	1 1 *op* 1	*immed* [3:0]
AND | EOR | SUB | RSB | ADD | ADC | SBC | RSC	*cond*	0 0 1 0	*op* *S*	*Rn*	*Rd*	*rotate*	*immed*	
MSR cpsr, #imm | MSR spsr, #imm	*cond*	0 0 1 1	0 *op* 1 0	*f s x c*	1 1 1 1	*rotate*	*immed*	
TST | TEQ | CMP | CMN	*cond*	0 0 1 1	0 *op* 1	*Rn*	0 0 0 0	*rotate*	*immed*	
ORR | BIC	*cond*	0 0 1 1	1 *op* 0 *S*	*Rn*	*Rd*	*rotate*	*immed*	
MOV | MVN	*cond*	0 0 1 1	1 *op* 1 *S*	0 0 0 0	*Rd*	*rotate*	*immed*	
STR | LDR | STRB | LDRB *post*	*cond*	0 1 0 0	*U* *op* *T* *op*	*Rn*	*Rd*	*immed12*		
STR | LDR | STRB | LDRB *pre*	*cond*	0 1 0 1	*U* *op* *W* *op*	*Rn*	*Rd*	*immed12*		
STR | LDR | STRB | LDRB *post*	*cond*	0 1 1 0	*U* *op* *T* *op*	*Rn*	*Rd*	*shift_size*	*shift* 0	*Rm*
{ |S|Q|SH| |U|UQ|UH}ADD16	*cond*	0 1 1 0	0 *op*	*Rn*	*Rd*	1 1 1 1	0 0 0 1	*Rm*
{ |S|Q|SH| |U|UQ|UH}ADDSUBX	*cond*	0 1 1 0	0 *op*	*Rn*	*Rd*	1 1 1 1	0 0 1 1	*Rm*
{ |S|Q|SH| |U|UQ|UH}SUBADDX	*cond*	0 1 1 0	0 *op*	*Rn*	*Rd*	1 1 1 1	0 1 0 1	*Rm*
{ |S|Q|SH| |U|UQ|UH}SUB16	*cond*	0 1 1 0	0 *op*	*Rn*	*Rd*	1 1 1 1	0 1 1 1	*Rm*
{ |S|Q|SH| |U|UQ|UH}ADD8	*cond*	0 1 1 0	0 *op*	*Rn*	*Rd*	1 1 1 1	1 0 0 1	*Rm*
{ |S|Q|SH| |U|UQ|UH}SUB8	*cond*	0 1 1 0	0 *op*	*Rn*	*Rd*	1 1 1 1	1 1 1 1	*Rm*
PKHBT | PKHTB	*cond*	0 1 1 0	1 0 0 0	*Rn*	*Rd*	*shift_size*	*op* 0 1	*Rm*
{S|U}SAT	*cond*	0 1 1 0	1 *op* 1	*immed5*	*Rd*	*shift_size*	*sh* 0 1	*Rm*
{S|U}SAT16	*cond*	0 1 1 0	1 *op* 1 0	*immed4*	*Rd*	1 1 1 1	0 0 1 1	*Rm*
SEL	*cond*	0 1 1 0	1 0 0 0	*Rn*	*Rd*	1 1 1 1	1 0 1 1	*Rm*
REV | REV16 | | REVSH	*cond*	0 1 1 0	1 *op* 1 1	1 1 1 1	*Rd*	1 1 1 1	*op* 0 1 1	*Rm*
{S|U}XTAB16	*cond*	0 1 1 0	1 *op* 0 0	*Rn!*=1111	*Rd*	*rot* 0 0	0 1 1 1	*Rm*
{S|U}XTB16	*cond*	0 1 1 0	1 *op* 0 0	1 1 1 1	*Rd*	*rot* 0 0	0 1 1 1	*Rm*
{S|U}XTAB	*cond*	0 1 1 0	1 *op* 1 0	*Rn!*=1111	*Rd*	*rot* 0 0	0 1 1 1	*Rm*
{S|U}XTB	*cond*	0 1 1 0	1 *op* 1 0	1 1 1 1	*Rd*	*rot* 0 0	0 1 1 1	*Rm*
{S|U}XTAH	*cond*	0 1 1 0	1 *op* 1 1	*Rn!*=1111	*Rd*	*rot* 0 0	0 1 1 1	*Rm*
{S|U}XTH	*cond*	0 1 1 0	1 *op* 1 1	1 1 1 1	*Rd*	*rot* 0 0	0 1 1 1	*Rm*
STR | LDR | STRB | LDRB *pre*	*cond*	0 1 1 1	*U* *op* *W* *op*	*Rn*	*Rd*	*shift_size*	*shift* 0	*Rm*
SMLAD | SMLSD	*cond*	0 1 1 1	0 0 0 0	*Rd*	*Rn!*=1111	*Rs*	0 *op X* 1	*Rm*
SMUAD | SMUSD	*cond*	0 1 1 1	0 0 0 0	*Rd*	1 1 1 1	*Rs*	0 *op X* 1	*Rm*
SMLALD | SMLSLD	*cond*	0 1 1 1	0 1 0 0	*RdHi*	*RdLo*	*Rs*	0 *op X* 1	*Rm*

TABLE B2.1 ARM instruction decode table. (*Continued.*)

Instruction classes (indexed by *op*)	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
SMMLA | | | SMMLS	*cond*	0 1 1 1	0 1 0 1	*Rd*	*Rn* !=1111	*Rs*	*op* *R* 1	*Rm*
SMMUL	*cond*	0 1 1 1	0 1 0 1	*Rd*	1 1 1 1	*Rs*	0 0 *R* 1	*Rm*
USADA8	*cond*	0 1 1 1	1 0 0 0	*Rd*	*Rn* != 1111	*Rs*	0 0 0 1	*Rm*
USAD8	*cond*	0 1 1 1	1 0 0 0	*Rd*	1 1 1 1	*Rs*	0 0 0 1	*Rm*
Undefined and expected to stay so	*cond*	0 1 1 1	1 1 1 1	*x*			1 1 1 1	*x*
STMDA | LDMDA | STMIA | LDMIA	*cond*	1 0 0 0	*op* ^ *W* *op*	*Rn*	*register_list*			
STMDB | LDMDB | STMIB | LDMIB	*cond*	1 0 0 1	*op* ^ *W* *op*	*Rn*	*register_list*			
B to instruction_address+8+4*offset	*cond*	1 0 1 0	signed 24-bit branch offset					
BL to instruction_address+8+4*offset	*cond*	1 0 1 1	signed 24-bit branch offset					
MCRR | MRRC	*cond*	1 1 0 0	0 1 0 *op*	*Rn*	*Rd*	*copro*	*op1*	*Cm*
STC{L} | LDC{L} *unindexed*	*cond*	1 1 0 0	1 *L* 0 *op*	*Rn*	*Cd*	*copro*	*option*	
STC{L} | LDC{L} *post*	*cond*	1 1 0 0	*U* *L* 1 *op*	*Rn*	*Cd*	*copro*	*immed8*	
STC{L} | LDC{L} *pre*	*cond*	1 1 0 1	*U* *L* *W* *op*	*Rn*	*Cd*	*copro*	*immed8*	
CDP	*cond*	1 1 1 0	*op1*	*Cn*	*Cd*	*copro*	*op2* 0	*Cm*
MCR | MRC	*cond*	1 1 1 0	*op1* *op*	*Cn*	*Rd*	*copro*	*op2* 1	*Cm*
SWI	*cond*	1 1 1 1	immed24					
CPS | | CPSIE | CPSID	1 1 1 1	0 0 0 1	0 0 0 0	*op* *M* 0	0 0 0 0	0 0 0 *a*	*i* *f* 0	*mode*
SETEND LE | SETEND BE	1 1 1 1	0 0 0 1	0 0 0 0	0 0 0 1	0 0 0 0	0 0 *op* 0	0 0 0 0	0 0 0 0
PLD pre	1 1 1 1	0 1 0 1	*U* 1 0 1	*Rn*	1 1 1 1	*immed12*		
PLD pre	1 1 1 1	0 1 1 1	*U* 1 0 1	*Rn*	1 1 1 1	*shift_size*	*shift* 0	*Rm*
RFEDA | RFEIA | RFEDB | RFEIB	1 1 1 1	1 0 0 *op*	*op* 0 *W* 1	*Rn*	0 0 0 0	1 0 1 0	0 0 0 0	0 0 0 0
SRSDA | SRSIA | SRSDB | SRSIB	1 1 1 1	1 0 0 *op*	*op* 1 *W* 0	1 1 0 1	0 0 0 0	0 1 0 1	0 0 0	*mode*
BLX instruction+8+4*offset+2*a	1 1 1 1	1 0 1 *a*	signed 24-bit branch offset					
MCRR2 | MRRC2	1 1 1 1	1 1 0 0	0 1 0 *op*	*Rn*	*Rd*	*copro*	*op1*	*Cm*
STC2{L} | LDC2{L} *unindexed*	1 1 1 1	1 1 0 0	1 *L* 0 *op*	*Rn*	*Cd*	*copro*	*option*	
STC2{L} | LDC2{L} *post*	1 1 1 1	1 1 0 0	*U* *L* 1 *op*	*Rn*	*Cd*	*copro*	*immed8*	
STC2{L} | LDC2{L} *pre*	1 1 1 1	1 1 0 1	*U* *L* *W* *op*	*Rn*	*Cd*	*copro*	*immed8*	
CDP2	1 1 1 1	1 1 1 0	*op1*	*Cn*	*Cd*	*copro*	*op2* 0	*Cm*
MCR2 | MRC2	1 1 1 1	1 1 1 0	*op1* *op*	*Cn*	*Cd*	*copro*	*op2* 1	*Cm*

TABLE B2.2 Decoding table for *cond*.

Binary	Hex	*cond*
0000	0	EQ
0001	1	NE
0010	2	CS/HS
0011	3	CC/LO
0100	4	MI
0101	5	PL
0110	6	VS
0111	7	VC

Binary	Hex	*cond*
1000	8	HI
1001	9	LS
1010	A	GE
1011	B	LT
1100	C	GT
1101	D	LE
1110	E	{AL}

- Index through Table B2.1 using the second hex digit, bits 24 to 27 (shaded).
- Index using bit 4, then bit 7 or bit 23 of the instruction where these bits are shaded.
- Once you have located the correct table entry, look at the bits named *op*. Concatenate these to form a binary number that indexes the | separated instruction list on the left. For example if there are two *op* bits value 1 and 0, then the binary value 10 indicates instruction number 2 in the list (the third instruction).
- The instruction operands have the same name as in the instruction description of Appendix B1.

The table uses the following abbreviations:

- *L* is 1 if the L suffix applies for `LDC` and `STC` operations.
- *M* is 1 if `CPS` changes processor mode. *mode* is defined in Table B2.3.

TABLE B2.3 Decoding table for *mode*.

Binary	Hex	*mode*
10000	0x10	*user* mode (*_usr*)
10001	0x11	*FIQ* mode (*_fiq*)
10010	0x12	*IRQ* mode (*_irq*)
10011	0x13	*supervisor* mode (*_svc*)
10111	0x17	*abort* mode (*_abt*)
11011	0x1B	*undefined* mode (*_und*)
11111	0x1F	*system* mode

TABLE B2.4 Decoding table for *shift*, *shift_size*, and *Rs*.

shift	*shift_size*	*Rs*	Shift action
00	0 to 31	N/A	LSL #shift_size
00	N/A	*Rs*	LSL Rs
01	0	N/A	LSR #32
01	1 to 31	N/A	LSR #shift_size
01	N/A	*Rs*	LSR Rs
10	0	N/A	ASR #32
10	1 to 31	N/A	ASR #shift_size
10	N/A	*Rs*	ASR Rs
11	0	N/A	RRX
11	1 to 31	N/A	ROR #shift_size
11	N/A	*Rs*	ROR Rs
N/A	0 to 31	N/A	The *shift* value is implicit: For PKHBT it is 00. For PKHTB it is 10. For SAT it is 2* *sh*.

- *op1* and *op2* are the opcode extension fields in coprocessor instructions.
- *post* indicates a postindexed addressing mode such as `[Rn]`, `Rm` or `[Rn]`, `#immed`.
- *pre* indicates a preindexed addressing mode such as `[Rn, Rm]` or `[Rn, #immed]`.
- *register_list* is a bit field with bit *k* set if register *Rk* appears in the register list.
- *rot* is a byte rotate. The second operand is `Rm ROR (8*rot)`.
- *rotate* is a bit rotate. The second operand is `#immed ROR (2*rotate)`.
- *shift* and *sh* encode a shift type and direction. See Table B2.4.
- *U* is the up/down select for addressing modes. If *U* = 1, then we add the offset to the base address, as in `[Rn]`, #4 or `[Rn, Rm]`. If *U* = 0 then we subtract the offset from the base address, as in `[Rn, #-4]` or `[Rn],-Rm`.
- *unindexed* indicates an addressing mode of the form `[Rn],{option}`.
- *R* is 1 if the `R` (round) instruction suffix is present.
- *T* is 1 if the `T` suffix is present on load and store instructions.
- *W* is 1 if `!` (writeback) is specified in the instruction mnemonic.
- *X* is 1 if the `X` (exchange) instruction suffix is present.
- *x* and *y* are 0 for the `B` suffix, `1` for the `T` suffix.
- `^` is 1 if the `^` suffix is applied in `LDM` or `STM` instructions.

B2.2 Thumb Instruction Set Encodings

Table B2.5 summarizes the bit encodings for the 16-bit Thumb instruction set. This table is useful if you need to decode a Thumb instruction by hand. We've expanded the table to aid quick manual decode. The table contains instruction definitions up to architecture THUMBv3. Any bitmaps not listed are either unpredictable or undefined for THUMBv3.

To use the table efficiently, follow this decoding procedure:

- Index through the table using the first hex digit of the instruction, bits 12 to 15 (shaded).
- Index on any shaded bits from bits 0 to 11.
- Once you have located the correct table entry, look at the bits named *op*. Concatenate these to form a binary number that indexes the | separated instruction list on the left. For example, if there are two *op* bits value 1 and 0, then the binary value 10 indicates instruction number 2 in the list (the third instruction).

TABLE B2.5 Thumb instruction decode table.

Instruction classes (indexed by *op*)	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
`LSL | LSR`	0	0	0	0	*op*	*immed5*					*Lm*			*Ld*		
`ASR`	0	0	0	1	0	*immed5*					*Lm*			*Ld*		
`ADD | SUB`	0	0	0	1	1	0	*op*	*Lm*			*Ln*			*Ld*		
`ADD | SUB`	0	0	0	1	1	1	*op*	*immed3*			*Ln*			*Ld*		
`MOV | CMP`	0	0	1	0	*op*	*Ld/Ln*			*immed8*							
`ADD | SUB`	0	0	1	1	*op*	*Ld*			*immed8*							
`AND | EOR | LSL | LSR`	0	1	0	0	0	0	0	0	*op*		*Lm/Ls*			*Ld*		
`ASR | ADC | SBC | ROR`	0	1	0	0	0	0	0	1	*op*		*Lm/Ls*			*Ld*		
`TST | NEG | CMP | CMN`	0	1	0	0	0	0	1	0	*op*		*Lm*			*Ld/Ln*		
`ORR | MUL | BIC | MVN`	0	1	0	0	0	0	1	1	*op*		*Lm*			*Ld*		
`CPY Ld, Lm`	0	1	0	0	0	1	1	0	0	0	*Lm*			*Ld*		
`ADD | MOV Ld, Hm`	0	1	0	0	0	1	*op*	0	0	1	*Hm* & 7			*Ld*		
`ADD | MOV Hd, Lm`	0	1	0	0	0	1	*op*	0	1	0	*Lm*			*Hd* & 7		
`ADD | MOV Hd, Hm`	0	1	0	0	0	1	*op*	0	1	1	*Hm* & 7			*Hd* & 7		
`CMP`	0	1	0	0	0	1	0	1	0	1	*Hm* & 7			*Ln*		
`CMP`	0	1	0	0	0	1	0	1	1	0	*Lm*			*Hn* & 7		
`CMP`	0	1	0	0	0	1	0	1	1	1	*Hm* & 7			*Hn* & 7		
`BX | BLX`	0	1	0	0	0	1	1	1	*op*		*Rm*			0	0	0
`LDR Ld, [pc, #immed*4]`	0	1	0	0	1	*Ld*			*immed8*							
`STR | STRH | STRB | LDRSB` *pre*	0	1	0	1	0	*op*		*Lm*			*Ln*			*Ld*		
`LDR | LDRH | LDRB | LDRSH` *pre*	0	1	0	1	1	*op*		*Lm*			*Ln*			*Ld*		
`STR | LDR Ld, [Ln, #immed*4]`	0	1	1	0	*op*	*immed5*					*Ln*			*Ld*		
`STRB | LDRB Ld, [Ln, #immed]`	0	1	1	1	*op*	*immed5*					*Ln*			*Ld*		
`STRH | LDRH Ld, [Ln, #immed*2]`	1	0	0	0	*op*	*immed5*					*Ln*			*Ld*		

TABLE B2.5 Thumb instruction decode table. (*Continued.*)

Instruction classes (indexed by *op*)	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
`STR | LDR Ld, [sp, #immed*4]`	1	0	0	1	*op*	*Ld*			*immed8*							
`ADD Ld, pc, #immed*4 | ADD Ld, sp, #immed*4`	1	0	1	0	*op*	*Ld*			*immed8*							
`ADD sp, #immed*4 | SUB sp, #immed*4`	1	0	1	1	0	0	0	0	*op*	*immed7*						
`SXTH | SXTB | UXTH | UXTB`	1	0	1	1	0	0	1	0	*op*		*Lm*			*Ld*		
`REV | REV16 | | REVSH`	1	0	1	1	1	0	1	0	*op*		*Lm*			*Ld*		
`PUSH | POP`	1	0	1	1	*op*	1	0	*R*	*register_list*							
`SETEND LE | SETEND BE`	1	0	1	1	0	1	1	0	0	1	0	1	*op*	0	0	0
`CPSIE | CPSID`	1	0	1	1	0	1	1	0	0	1	1	*op*	0	*a*	*i*	*f*
`BKPT immed8`	1	0	1	1	1	1	1	0	*immed8*							
`STMIA | LDMIA Ln!,{register-list}`	1	1	0	0	*op*	*Ln*			*register_list*							
`B<cond> instruction_address+4+offset*2`	1	1	0	1	*cond* < 1110				signed 8-bit offset							
Undefined and expected to remain so	1	1	0	1	1	1	1	0	*x*							
`SWI immed8`	1	1	0	1	1	1	1	1	*immed8*							
`B instruction_address+4+offset*2`	1	1	1	0	0	signed 11-bit *offset*										
`BLX ((instruction+4+(poff<<12)+offset*4) &~ 3)` This must be preceded by a branch prefix instruction.	1	1	1	0	1	unsigned 10-bit *offset*										0
This is the branch prefix instruction. It must be followed by a relative `BL` or `BLX` instruction.	1	1	1	1	0	signed 11-bit prefix offset *poff*										
`BL instruction+4+ (poff<<12)+offset*2` This must be preceded by a branch prefix instruction.	1	1	1	1	1	unsigned 11-bit *offset*										

- The instruction operands have the same name as in the instruction description of Appendix B1.

The table uses the following abbreviations:

- *register_list* is a bit field with bit *k* set if register *Rk* appears in the register list.
- *R* is 1 if *lr* is in the register list of `PUSH` or *pc* is in the register list of `POP`.

B2.3 Program Status Registers

Table B2.6 shows how to decode the 32-bit program status registers for ARMv6.

TABLE B2.6 *cpsr* and *spsr* decode table.

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
N Z C V	*Q*	*Res*	*J*	*Res*	*GE*[3:0]	*Res*	*E A*	*I F T*	*mode*

Field	Use
N	Negative flag, records bit 31 of the result of flag-setting operations.
Z	Zero flag, records if the result of a flag-setting operation is zero.
C	Carry flag, records unsigned overflow for addition, not-borrow for subtraction, and is also used by the shifting circuit. See Table B1.3.
V	Overflow flag, records signed overflows for flag-setting operations.
Q	Saturation flag. Certain operations set this flag on saturation. See for example QADD in Appendix B1 (ARMv5E and above).
J	*J* = 1 indicates Java execution (must have *T* = 0). Use the BXJ instruction to change this bit (ARMv5J and above).
Res	These bits are reserved for future expansion. Software should preserve the values in these bits.
GE[3:0]	The SIMD greater-or-equal flags. See SADD in Appendix B1 (ARMv6).
E	Controls the data endianness. See SETEND in Appendix B1 (ARMv6).
A	*A* = 1 disables imprecise data aborts (ARMv6).
I	*I* = 1 disables IRQ interrupts.
F	*F* = 1 disables FIQ interrupts.
T	*T* = 1 indicates Thumb state. T = 0 indicates ARM state. Use the BX or BLX instructions to change this bit (ARMv4T and above).
mode	The current processor mode. See Table B2.4.