The ARM Instruction Set

Peng-Sheng Chen

Fall, 2017

Outline

- Introduction
- Exceptions
- Conditional execution
- Branch, branch with link and eXchange
- Software Interrupt (SWI)
- Data processing instructions
- Multiply instructions
- Data transfer instructions
- Coprocessor instructions
- Others

Outline

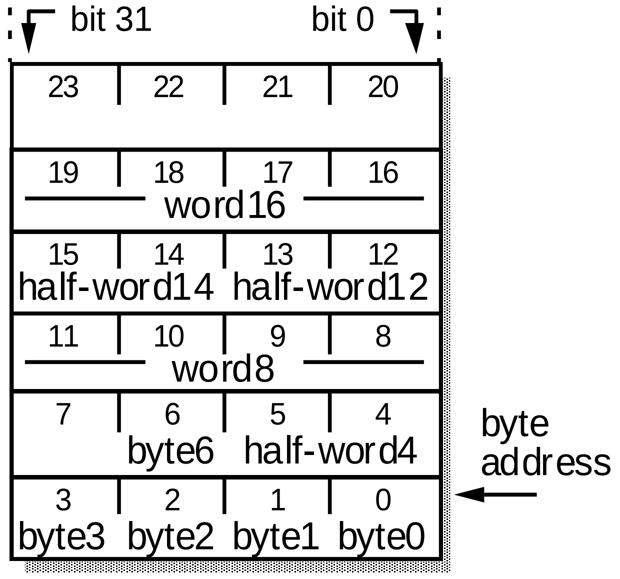
- Introduction
- Exceptions
- Conditional execution
- Branch, branch with link and eXchange
- Software Interrupt (SWI)
- Data processing instructions
- Multiply instructions
- Data transfer instructions
- Coprocessor instructions
- Others

Introduction (1)

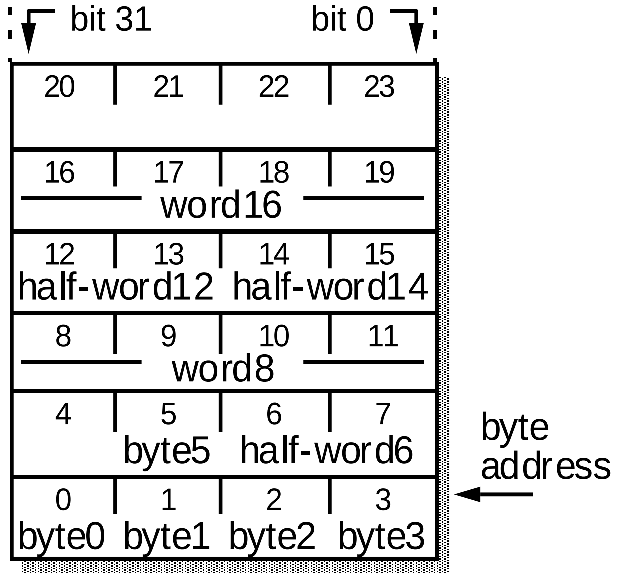
- **ARM processors support six data types**
  - 8-bit signed and unsigned bytes
  - 16-bit signed and unsigned half-words
  - 32-bit signed and unsigned words
- **All ARM operations are on 32-bits operands** (except Thumb instruction set)
  - The shorter data types are only supported by data transfer function

Introduction (2)

- **Memory organization**
  - Little-endian
  - Big-endian
- **Default is little-endian**



(a) Little-endian memory organization



(b) Big-endian memory organization

ARM Operating Modes and Register Usage

CPSR[4:0]	Mode	Use	Registers
10000	User	Normal user code	user
10001	FIQ	Processing fast interrupts	_fiq
10010	IRQ	Processing standard interrupts	_irq
10011	SVC	Processing software interrupts (SWIs)	_svc
10111	Abort	Processing memory faults	_abt
11011	Undef	Handling undefined instruction traps	_und
11111	System	Running privileged operating system tasks	user

- Each privileged mode has a SPSR, Saved Program Status Register
  - Save the state of the CPSR when privileged mode is entered

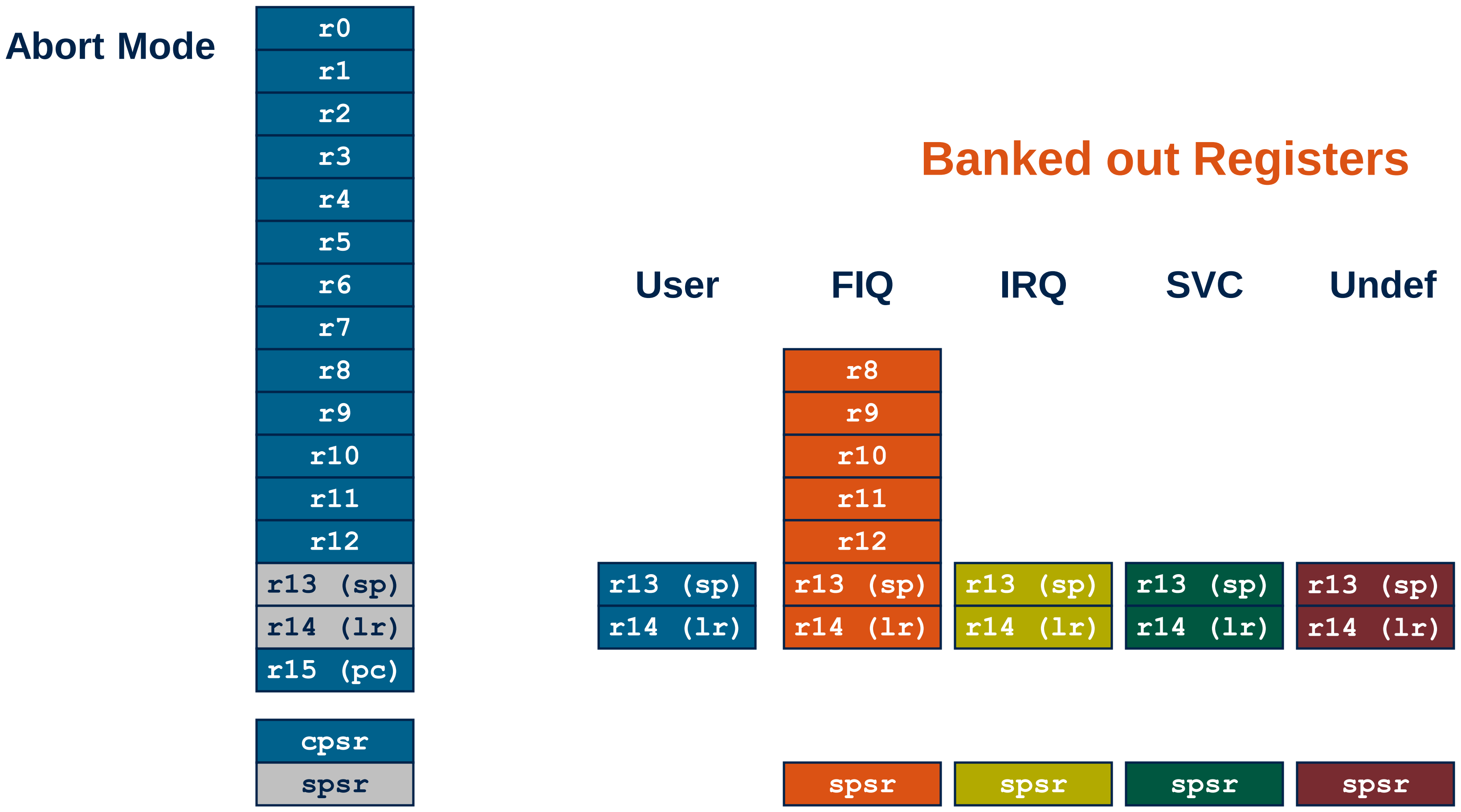
ARM

The ARM Register Set

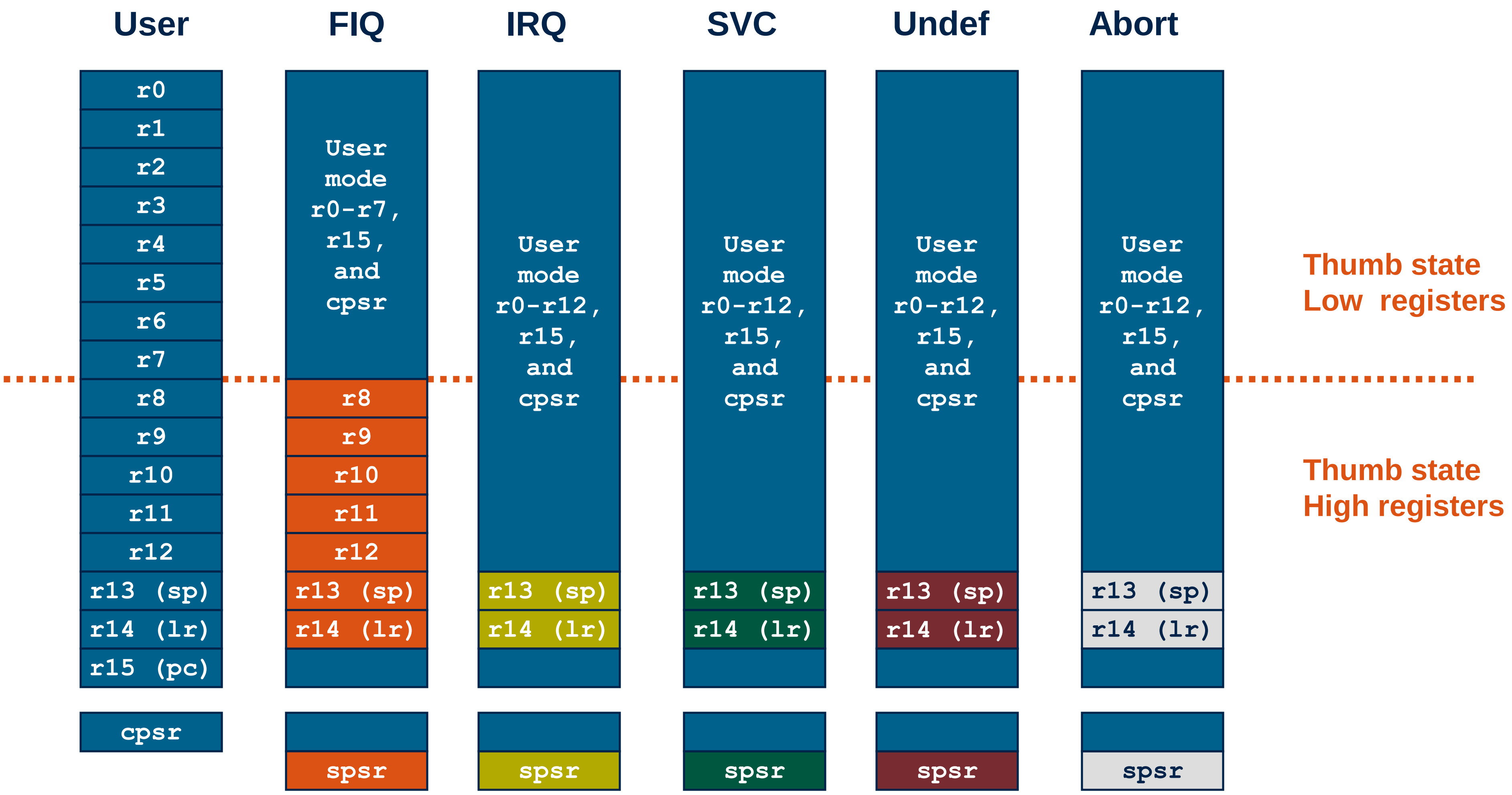
Reference from Intel higher education program (http://www.intel.com)

Current Visible Registers

Banked out Registers



Register Organization Summary



User	FIQ	IRQ	SVC	Undef	Abort
r0	User mode r0-r7, r15, and cpsr	User mode r0-r12, r15, and cpsr	User mode r0-r12, r15, and cpsr	User mode r0-r12, r15, and cpsr	User mode r0-r12, r15, and cpsr
r1					
r2					
r3					
r4					
r5					
r6					
r7					
r8	r8				
r9	r9				
r10	r10				
r11	r11				
r12	r12				
r13 (sp)	r13 (sp)	r13 (sp)	r13 (sp)	r13 (sp)	r13 (sp)
r14 (lr)	r14 (lr)	r14 (lr)	r14 (lr)	r14 (lr)	r14 (lr)
r15 (pc)					
cpsr					
	spsr	spsr	spsr	spsr	spsr

Note: System mode uses the User mode register set

Reference from Intel higher education program (http://www.intel.com)

Outline

- Introduction
- Exceptions
- Conditional execution
- Branch, branch with link and eXchange
- Software Interrupt (SWI)
- Data processing instructions
- Multiply instructions
- Data transfer instructions
- Coprocessor instructions
- Others

Exceptions

- Exceptions are usually used to handle unexpected events which arise during the execution of a program

Exception Groups

- **Direct effect of executing an instruction**
  - SWI
  - Undefined instructions
  - Prefetch aborts (memory fault occurring during fetch)
- **A side-effect of an instruction**
  - Data abort (a memory fault during a load or store data access)
- **Exceptions generated externally**
  - Reset
  - IRQ
  - FIQ

Exception Entry

- Change to the corresponding mode
- Save the address of the instruction following the exception instruction in **r14 of the new mode**
- Save the old value of CPSR in the **SPSR of the new mode**
- Disable IRQ
- If the exception is a FIQ, disables further FIQ
- Force PC to execute at the relevant vector address

Exception Vector Addresses

Exception	Mode	Vector address
Reset	SVC	0x00000000
Undefined instruction	UND	0x00000004
Software interrupt (SWI)	SVC	0x00000008
Prefetch abort (instruction fetch memory fault)	Abort	0x0000000C
Data abort (data access memory fault)	Abort	0x00000010
IRQ (normal interrupt)	IRQ	0x00000018
FIQ (fast interrupt)	FIQ	0x0000001C

Exception Return

- Any modified user registers must be restored
- Restore CPSR
- Resume PC in the correct instruction stream

Some Problem in Exception Return

- "**Restore CPSR**" and "**Resume PC**" cannot be carried out **independently**
- **Restore CPSR first**
  - The r14 holding the return address is no longer accessible
- **Resume PC first**
  - The exception handler loses control of the instruction stream and cannot cause "Restore CPSR" to take place

Solution I

- **The return address is in r14**
- **The "S" modifier after the opcode signifies the special form of the instruction**
  - To return from a SWI or undefined instruction trap
    - **MOVS pc, r14**
  - To return from an IRQ, FIQ or prefetch abort
    - **SUBS pc, r14, #4**
    - Return one instruction early in order to execute the instruction that was usurped for the exception entry
  - To return from a data abort to retry the data access
    - **SUBS pc, r14, #8**

Solution II

- **The return address has been saved onto a stack**
  - **LDMFD	r13!, {r0-r3,pc}^**
  - "**^**" indicates that this is a special form of the instruction

Exception Priorities

- Reset
- Data abort
- FIQ
- IRQ
- Prefetch abort
- SWI, undefined instruction

Highest priority

$\downarrow$

Address Exceptions

Exception	Mode	Vector address
Reset	SVC	0x00000000
Undefined instruction	UND	0x00000004
Software interrupt (SWI)	SVC	0x00000008
Prefetch abort (instruction fetch memory fault)	Abort	0x0000000C
Data abort (data access memory fault)	Abort	0x00000010
IRQ (normal interrupt)	IRQ	0x00000018
FIQ (fast interrupt)	FIQ	0x0000001C

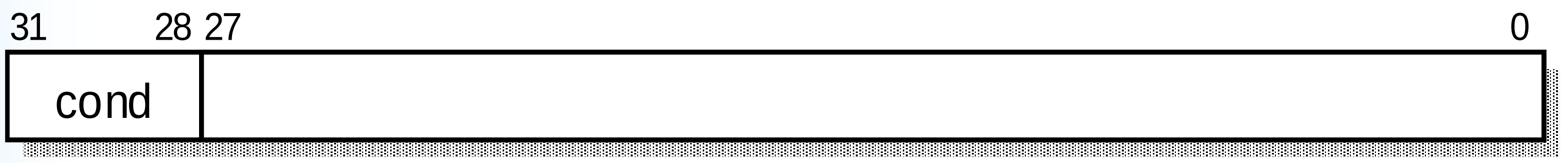
- **Where is 0x00000014?**
- **This location was used on earlier ARM which operated within a 26-bit address space**
- **Trap when ARM operated outside 26-bit address space**

Outline

- Introduction
- Exceptions
- Conditional execution
- Branch, branch with link and eXchange
- Software Interrupt (SWI)
- Data processing instructions
- Multiply instructions
- Data transfer instructions
- Coprocessor instructions
- Others

The ARM Condition Code Field

- Every instruction is conditionally executed
- Each of the 16 values of the condition field causes the instruction to be executed or skipped according to **the values of the N, Z, C and V flags in the CPSR**



N: Negative   Z: Zero   C: Carry   V: oVerflow

ARM Condition Codes

Opcode [31:28]	Mnemonic extension	Interpretation	Status flag state for execution
0000	EQ	Equal / equals zero	Z set
0001	NE	Not equal	Z clear
0010	CS/HS	Carry set / unsigned higher or same	C set
0011	CC/LO	Carry clear / unsigned lower	C clear
0100	MI	Minus / negative	N set
0101	PL	Plus / positive or zero	N clear
0110	VS	Overflow	V set
0111	VC	No overflow	V clear
1000	HI	Unsigned higher	C set and Z clear
1001	LS	Unsigned lower or same	C clear or Z set
1010	GE	Signed greater than or equal	N equals V
1011	LT	Signed less than	N is not equal to V
1100	GT	Signed greater than	Z clear and N equals V
1101	LE	Signed less than or equal	Z set or N is not equal to V
1110	AL	Always	any
1111	NV	Never (do not use!)	none

Condition Field

- In ARM state, all instructions are conditionally executed according to the CPSR condition codes and the instruction's condition field
- Fifteen different conditions may be used
- **"Always" condition**
  - Default condition
  - May be omitted
- **"Never" condition**
  - The sixteen (1111) is reserved, and must not be used
  - May use this area for other purposes in the future

Alternative Mnemonics

- **Use “CS”, if you**
  - Add two unsigned integers
  - Test whether there was a carry-out from the addition
- **Use “HS”, if you**
  - Compare two unsigned integers
  - Test whether the first was higher or the same as the second

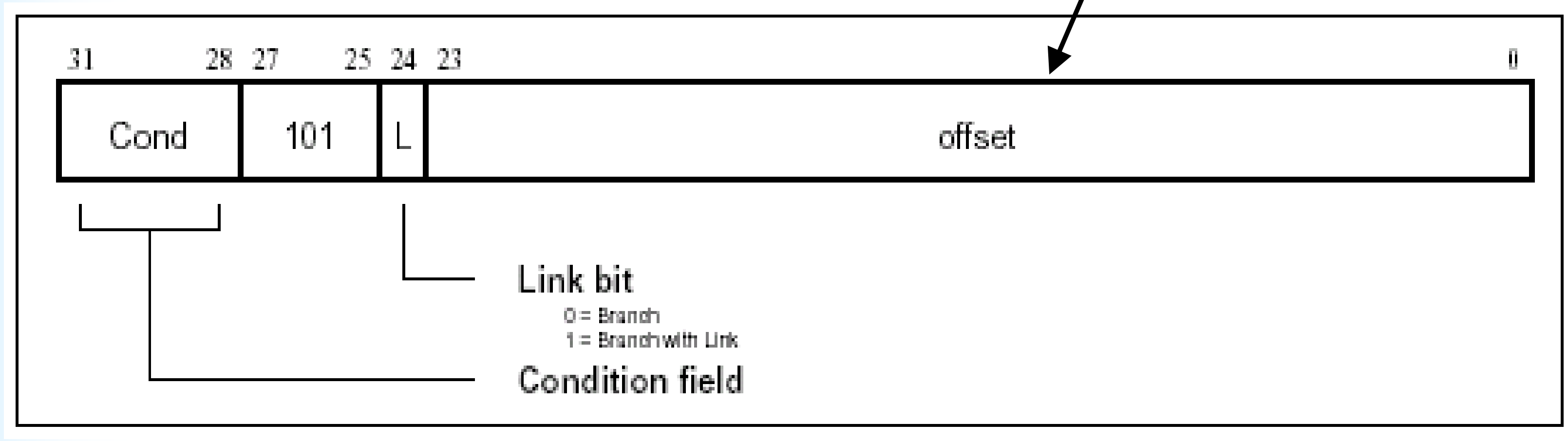
Outline

- Introduction
- Exceptions
- Conditional execution
- Branch, branch with link and eXchange
- Software Interrupt (SWI)
- Data processing instructions
- Multiply instructions
- Data transfer instructions
- Coprocessor instructions
- Others

Branch and Branch with Link

When the processor executes a branch instruction, the offset is added to the PC, and the machine begins fetching instructions from this new address.

Range: +/- 32 Mbytes



Assembler syntax: B {L} {cond} <expression>

Example

An unconditional jump

```
       B     LABEL    ; unconditional jump
       …
LABEL: …              ; to here
```

To execute a loop ten times

```
      MOV   r0, #10  ; initialize loop counter
LOOP: …
      SUBS  r0, #1   ; decrement counter setting CCs
      BNE   LOOP     ; if counter <> 0 repeat loop
      …              ; else drop through
```

To call a subroutine

```
      BL    SUB      ; branch and link to subroutine
      …              ; return to here
      …
SUB:  …              ; subroutine entry point
      MOV   PC, r14  ; return
```

Branch Conditions

Branch	Interpretation	Normal uses
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 gave 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 lower or same

Branch and Exchange Instruction Binary Encoding

A mechanism for switching the processor to execute Thumb instructions

(1) `BX|BLX Rm`

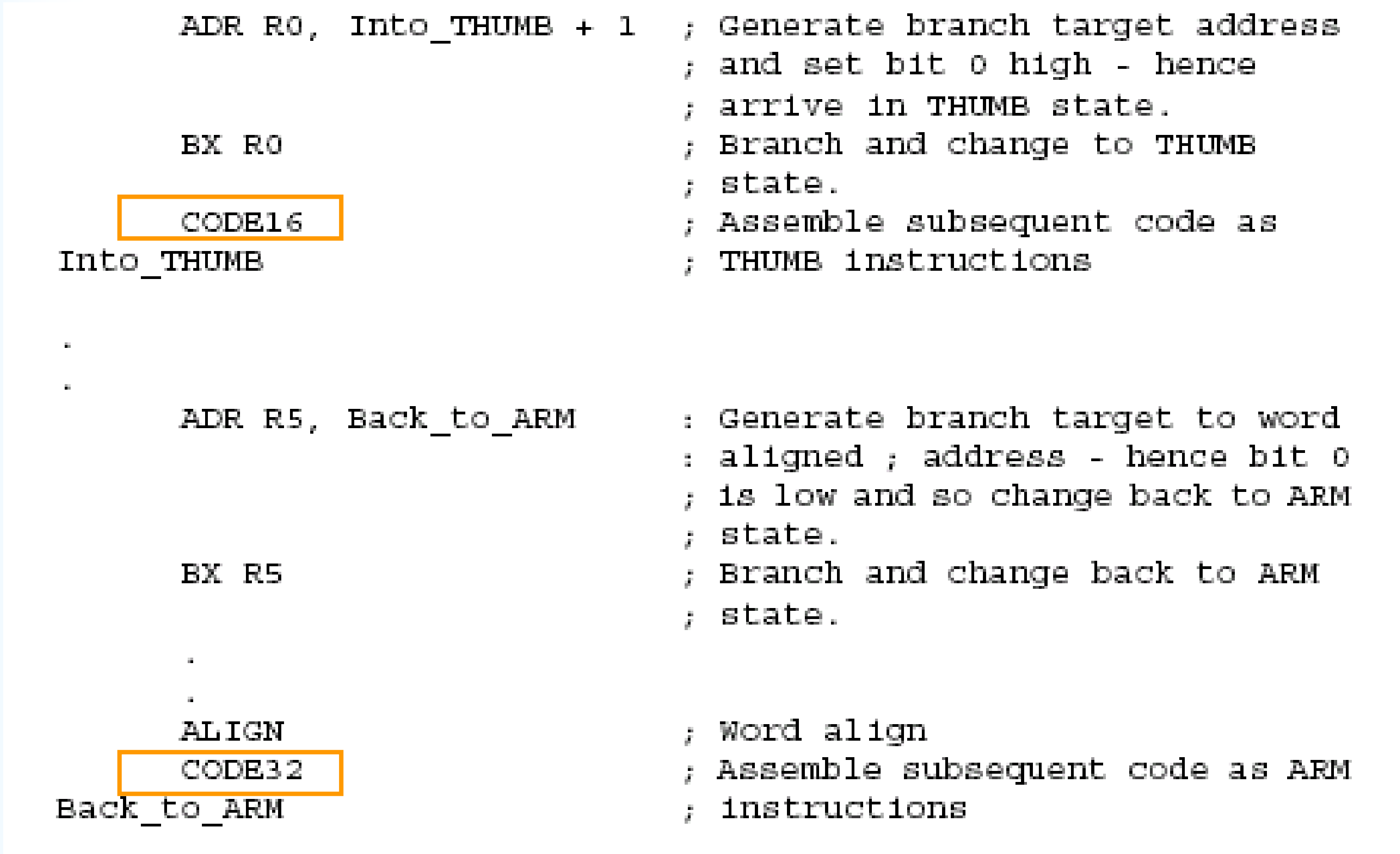
31 28	27 6	5	4	3 0
cond	0 0 0 1 0 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0	L	1	Rm

(2) `BLX label`

Available on ARM v5T

31 28	27 25	24	23 0
1 1 1 1	1 0 1	H	24-bit signed word offset

Branch and Exchange - Example



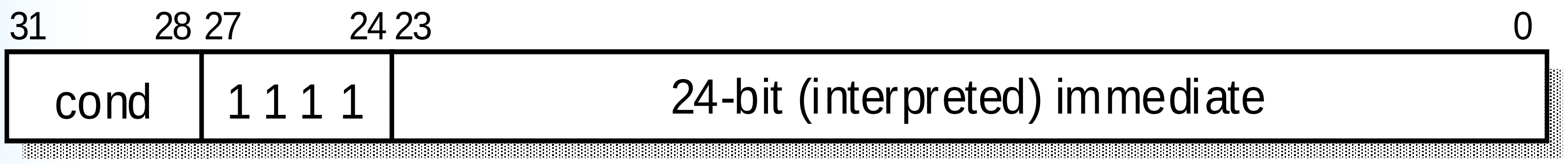
```
        ADR R0, Into_THUMB + 1      ; Generate branch target address
                                    ; and set bit 0 high - hence
                                    ; arrive in THUMB state.
        BX R0                       ; Branch and change to THUMB
                                    ; state.
        CODE16                      ; Assemble subsequent code as
  Into_THUMB                        ; THUMB instructions

  .
  .
        ADR R5, Back_to_ARM         : Generate branch target to word
                                    : aligned ; address - hence bit 0
                                    ; is low and so change back to ARM
                                    ; state.
        BX R5                       ; Branch and change back to ARM
                                    ; state.
        .
        .
        ALIGN                       ; Word align
        CODE32                      ; Assemble subsequent code as ARM
  Back_to_ARM                       ; instructions
```

- 指示Assembler把下面的指令翻譯成16-bit Thumb instruction or 32-bit ARM instruction
- GNU GAS: “.code 16” or “.code 32”

Outline

- Introduction
- Exceptions
- Conditional execution
- Branch, branch with link and eXchange
- Software Interrupt (SWI)
- Data processing instructions
- Multiply instructions
- Data transfer instructions
- Coprocessor instructions
- Others

Software Interrupt (SWI)



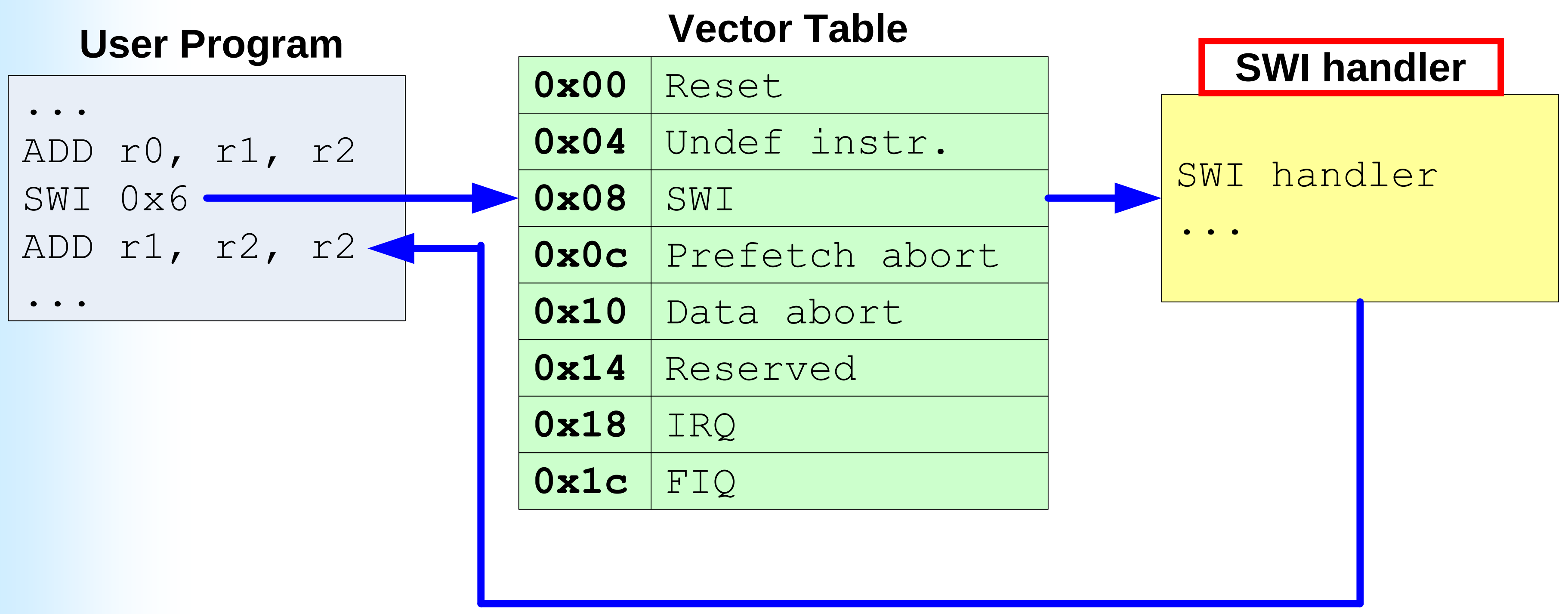
Assembler syntax: SWI {cond} <24-bit immediate>

SWI number: 表示一個特定的功能

Processor Actions for SWI (1)

- Enter supervisor mode
- Save the address of the instruction in **r14_svc**
- Save the **CPSR** in **SPSR_svc**
- Disable IRQs
- Set the **PC** to 0x8

Processor Actions for SWI (2)



A SWI Handler

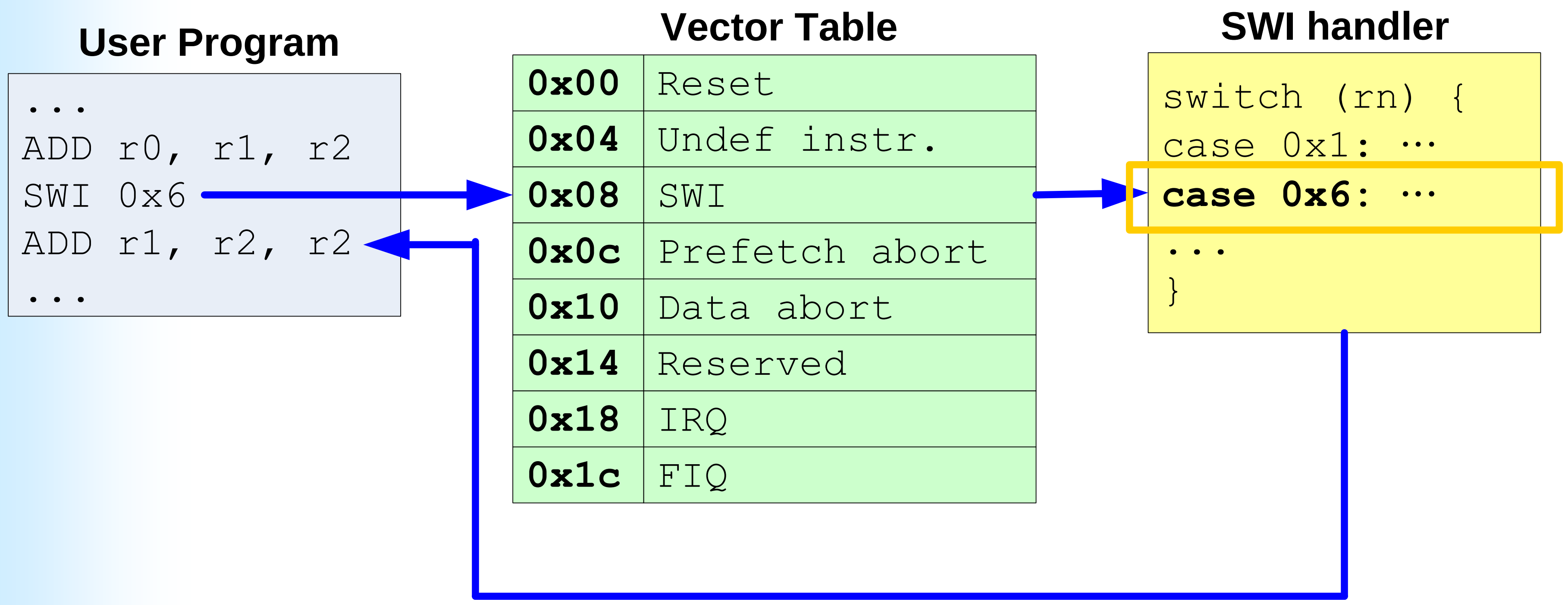
```
SWI_handler
    ;
    ; save register r0-r12 and r14
    ;
    STMF sp!, {r0-r12, r14}

    ; read the SWI instruction
    LDR  r10, [r14, #-4]

    ; mask off top 8 bits
    BIC  r10, r10, #0xff000000

    ; r10 contains the SWI number
    ; get the address of service routine from r10
    BL   service_routine

    ; return from SWI handler
    LDMFD sp!, {r0-r12, pc}^
```

A SWI Handler

```
SWI_handler
    ;
    ; save register r0-r12 and r14
    ;
    STMF sp!, {r0-r12, r14}

    ; read the SWI instruction
    LDR  r10, [r14, #-4]

    ; mask off top 8 bits
    BIC  r10, r10, #0xff000000

    ; r10 contains the SWI number
    ; get the address of service routine from r10
    BL   service_routine

    ; return from SWI handler
    LDMFD sp!, {r0-r12, pc}^
```

```
switch (r10) {
   case 1: call sub1
   case 2: call sub2
   case 3: call sub3
   …
}
```

Processor Actions for SWI (3)



User Program

```
...
ADD r0, r1, r2
SWI 0x6
ADD r1, r2, r2
...
```

Vector Table

0x00	Reset
0x04	Undef instr.
0x08	SWI
0x0c	Prefetch abort
0x10	Data abort
0x14	Reserved
0x18	IRQ
0x1c	FIQ

SWI handler

```
switch (rn) {
case 0x1: …
case 0x6: …
...
}
```

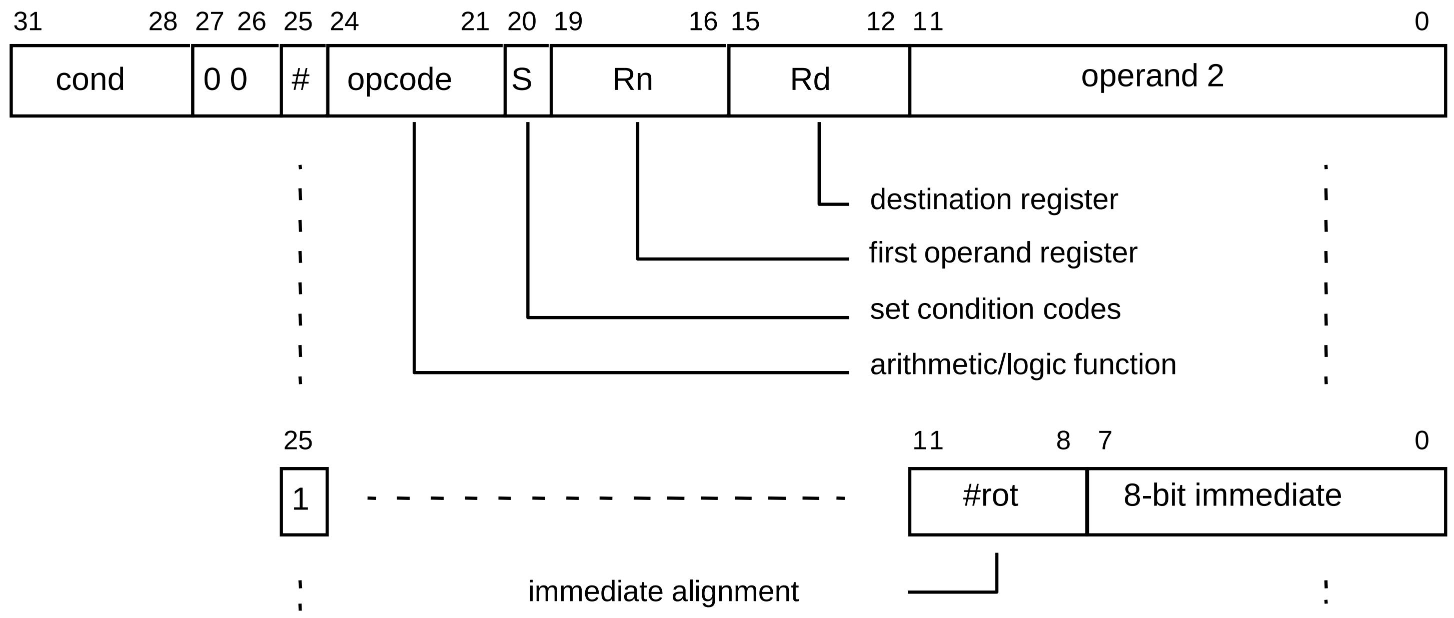
Types of SWIs in ARM Angel (axd or armsd)

SWI_WriteC	SWI 0	Write a byte to the debug channel
SWI_Write0	SWI 2	Write the nullterminated string to debug channel
SWI_ReadC	SWI 4	Read a byte from the debug channel
SWI_Exit	SWI 0x11	Halt emulation this is how a program exits
SWI_EnterOS	SWI 0x16	Put the processor in supervisor mode
SWI_Clock	SWI 0x61	Return the number of centiseconds
SWI_Time	SWI 0x63	Return the number of secs since Jan. 1, 1970

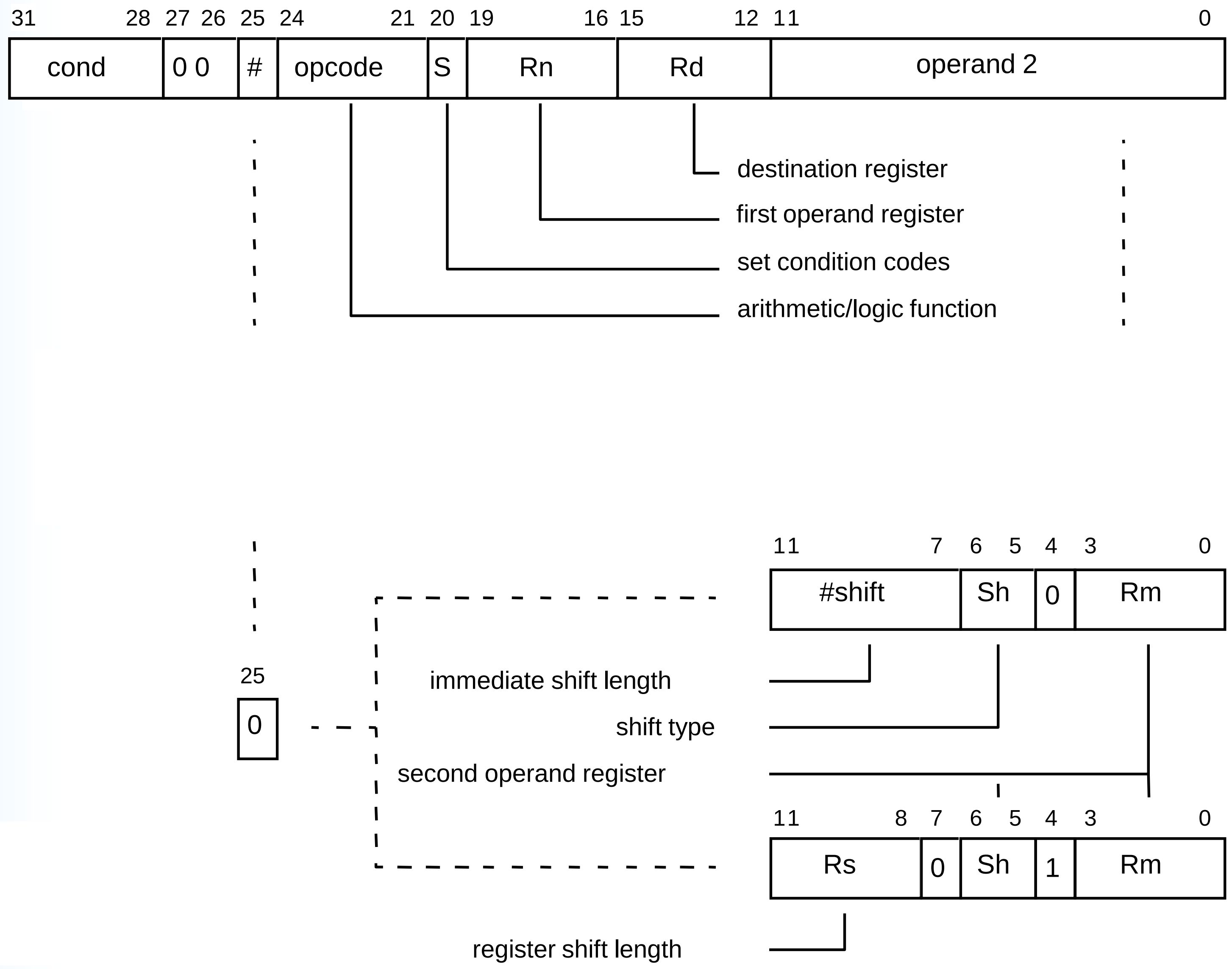
Outline

- Introduction
- Exceptions
- Conditional execution
- Branch, branch with link and eXchange
- Software Interrupt (SWI)
- Data processing instructions
- Multiply instructions
- Data transfer instructions
- Coprocessor instructions
- Others

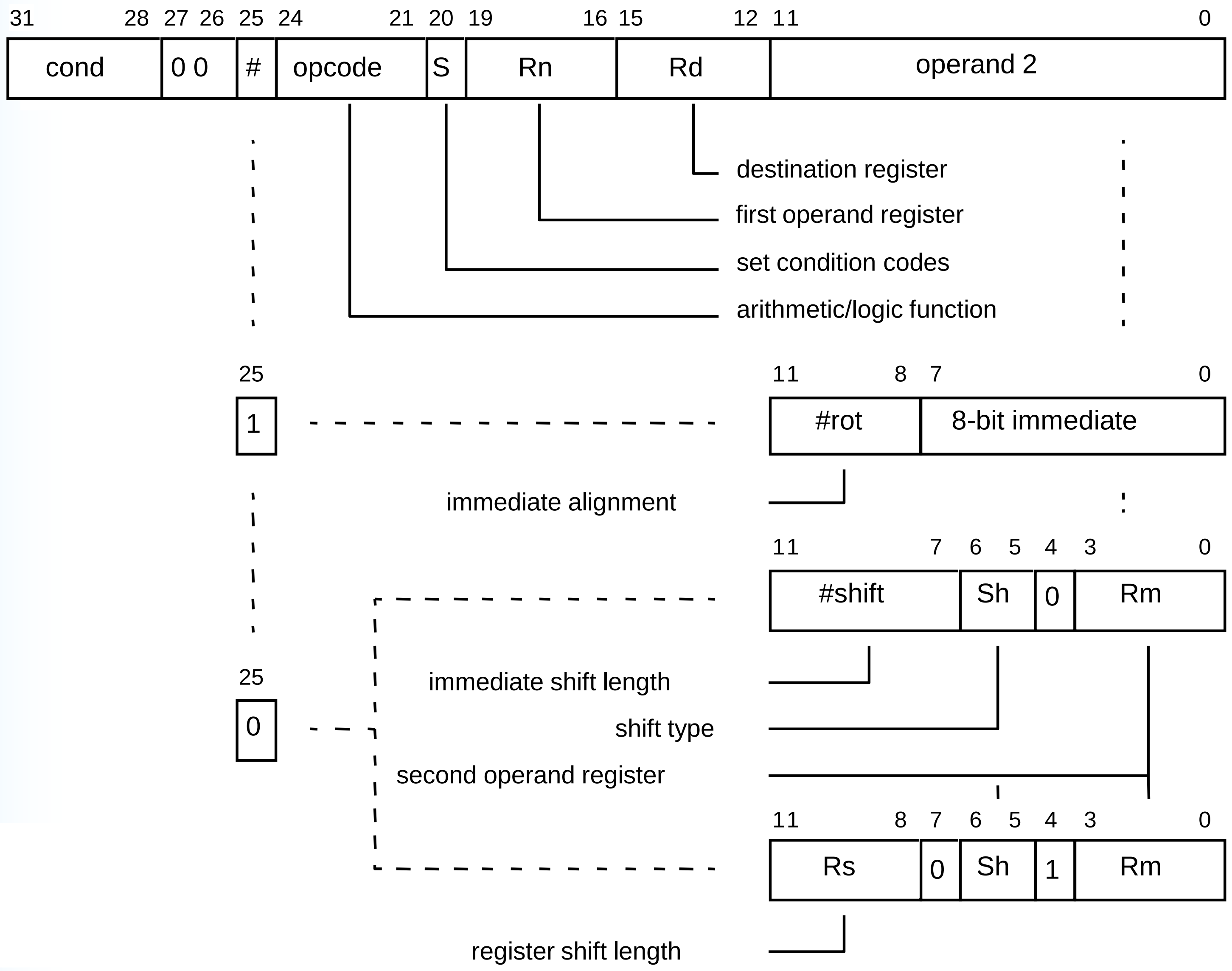
Data Processing Instruction Binary Encoding



Data Processing Instruction Binary Encoding



Data Processing Instruction Binary Encoding



Data Processing Instructions (1)

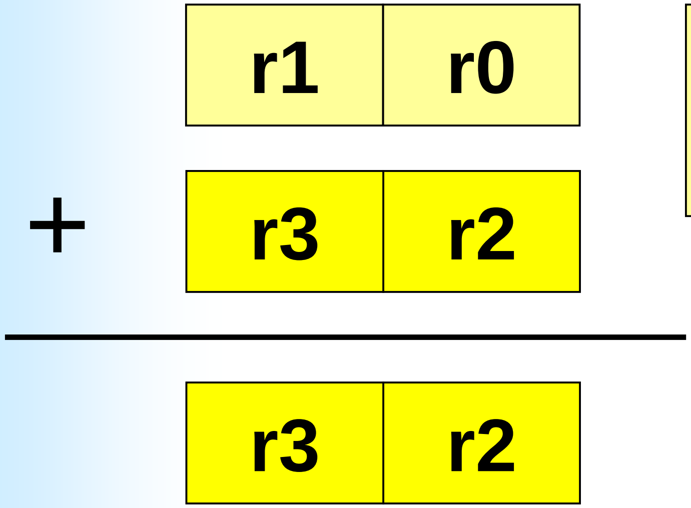
Opcode [24:21]	Mnemonic	Meaning	Effect
0000	AND	Logical bit-wise AND	Rd := Rn AND Op2
0001	EOR	Logical bit-wise exclusive OR	Rd := Rn EOR Op2
0010	SUB	Subtract	Rd := Rn - Op2
0011	RSB	Reverse subtract	Rd := Op2 - Rn
0100	ADD	Add	Rd := Rn + Op2
0101	ADC	Add with carry	Rd := Rn + Op2 + C
0110	SBC	Subtract with carry	Rd := Rn - Op2 + C - 1
0111	RSC	Reverse subtract with carry	Rd := Op2 - Rn + C - 1
1000	TST	Test	Scc on Rn AND Op2
1001	TEQ	Test equivalence	Scc on Rn EOR Op2
1010	CMP	Compare	Scc on Rn - Op2
1011	CMN	Compare negated	Scc on Rn + Op2
1100	ORR	Logical bit-wise OR	Rd := Rn OR Op2
1101	MOV	Move	Rd := Op2
1110	BIC	Bit clear	Rd := Rn AND NOT Op2
1111	MVN	Move negated	Rd := NOT Op2

Data Processing Instructions (2)

- **Assembler syntax**

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

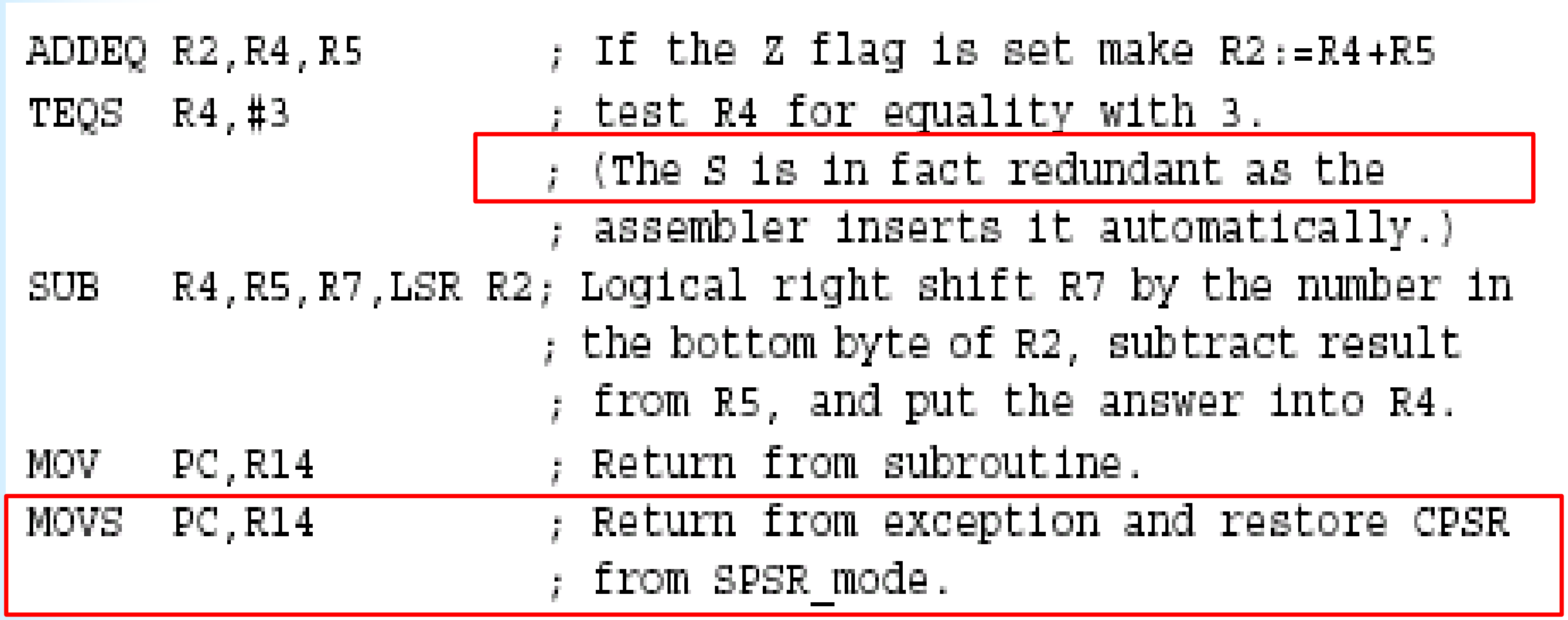
<op>{<cond>}{S} Rd, Rn, Rm, {<shift>}



```
ADDS  r2, r2, r0 ; 32-bit carry out->C
ADC   r3, r3, r1 ; C is added into
                 ; high word
```

Adding ‘S’ to the opcode, standing for ‘Set condition flags’

Examples

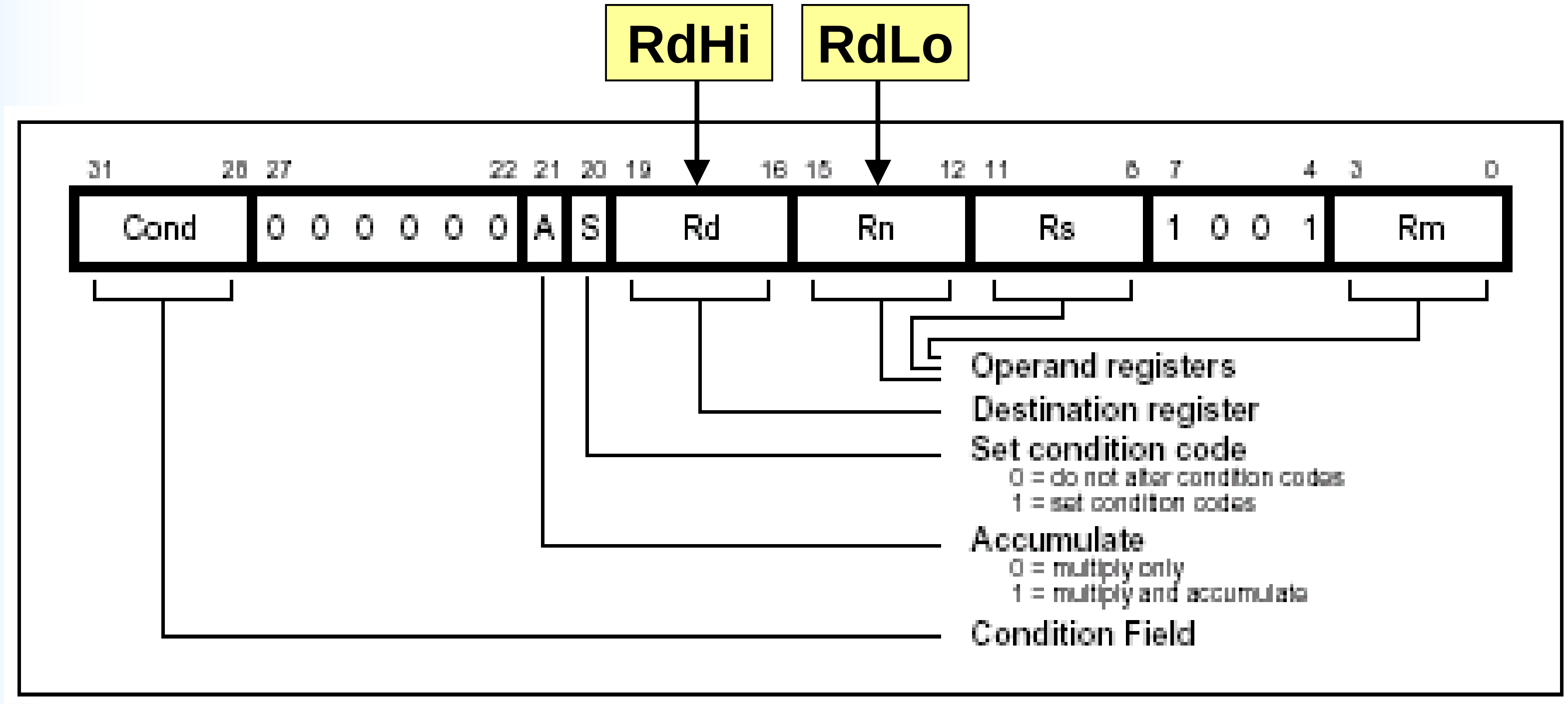


```
ADDEQ R2,R4,R5           ; If the Z flag is set make R2:=R4+R5
TEQS  R4,#3              ; test R4 for equality with 3.
                         ; (The S is in fact redundant as the
                         ; assembler inserts it automatically.)
SUB   R4,R5,R7,LSR R2;  Logical right shift R7 by the number in
                         ; the bottom byte of R2, subtract result
                         ; from R5, and put the answer into R4.
MOV   PC,R14             ; Return from subroutine.
MOVS  PC,R14             ; Return from exception and restore CPSR
                         ; from SPSR_mode.
```

Outline

- Introduction
- Exceptions
- Conditional execution
- Branch, branch with link and eXchange
- Software Interrupt (SWI)
- Data processing instructions
- Multiply instructions
- Data transfer instructions
- Coprocessor instructions
- Others

Multiply and Multiply- Accumulate



Assembler Syntax

MUL {<cond>} {S} Rd, Rm, Rs (least significant 32 bits result)

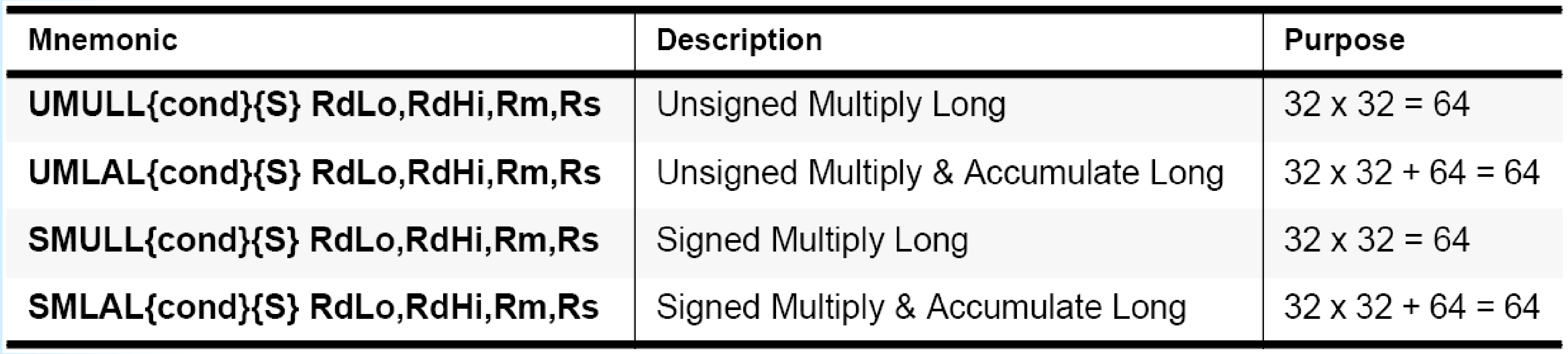
MLA {<cond>} {S} Rd, Rm, Rs, Rn

<mull>{<cond>}{S} RdLo, RdHi, Rm, Rs (full 64-bits result)

Multiply Instructions

Opcode [23:21]	Mnemonic	Meaning	Effect
000	MUL	Multiply (32-bit result)	Rd := (Rm * Rs) [31:0]
001	MLA	Multiply-accumulate (32-bit result)	Rd := (Rm * Rs + Rn) [31:0]
100	UMULL	Unsigned multiply long	
101	UMLAL	Unsigned multiply-accumulate long	
110	SMULL	Signed multiply long	
111	SMLAL	Signed multiply-accumulate long	

64-bit multiply types



Mnemonic	Description	Purpose
UMULL{cond}{S} RdLo,RdHi,Rm,Rs	Unsigned Multiply Long	32 x 32 = 64
UMLAL{cond}{S} RdLo,RdHi,Rm,Rs	Unsigned Multiply & Accumulate Long	32 x 32 + 64 = 64
SMULL{cond}{S} RdLo,RdHi,Rm,Rs	Signed Multiply Long	32 x 32 = 64
SMLAL{cond}{S} RdLo,RdHi,Rm,Rs	Signed Multiply & Accumulate Long	32 x 32 + 64 = 64

Multiply and Multiply- Accumulate (Example)

```
// multiplication (32-bit result)

MUL     r1, r2, r3        ; r1:=r2*r3
MLAEQ   r1, r2, r3, r4    ; Conditionally r1:=r2*r3+r4
                          ;
```

```
// multiplication (64-bit result)

UMULL   r1, r4, r2, r3    ; r4,r1=r2*r3
UMLAL   r1, r5, r2, r3    ; r5,r1=(r2*r3)+r5,r1
                          ;
```

Outline

- Introduction
- Exceptions
- Conditional execution
- Branch, branch with link and eXchange
- Software Interrupt (SWI)
- Data processing instructions
- Multiply instructions
- Data transfer instructions
- Coprocessor instructions
- Others

Data Transfer Instructions

- Single word and unsigned byte data transfer instructions
- Half-word and singed byte data transfer instructions
- Multiple register transfer instructions
- Swap memory and register instructions (SWP)
- Status register to general register transfer instructions
- General register to status register transfer instructions

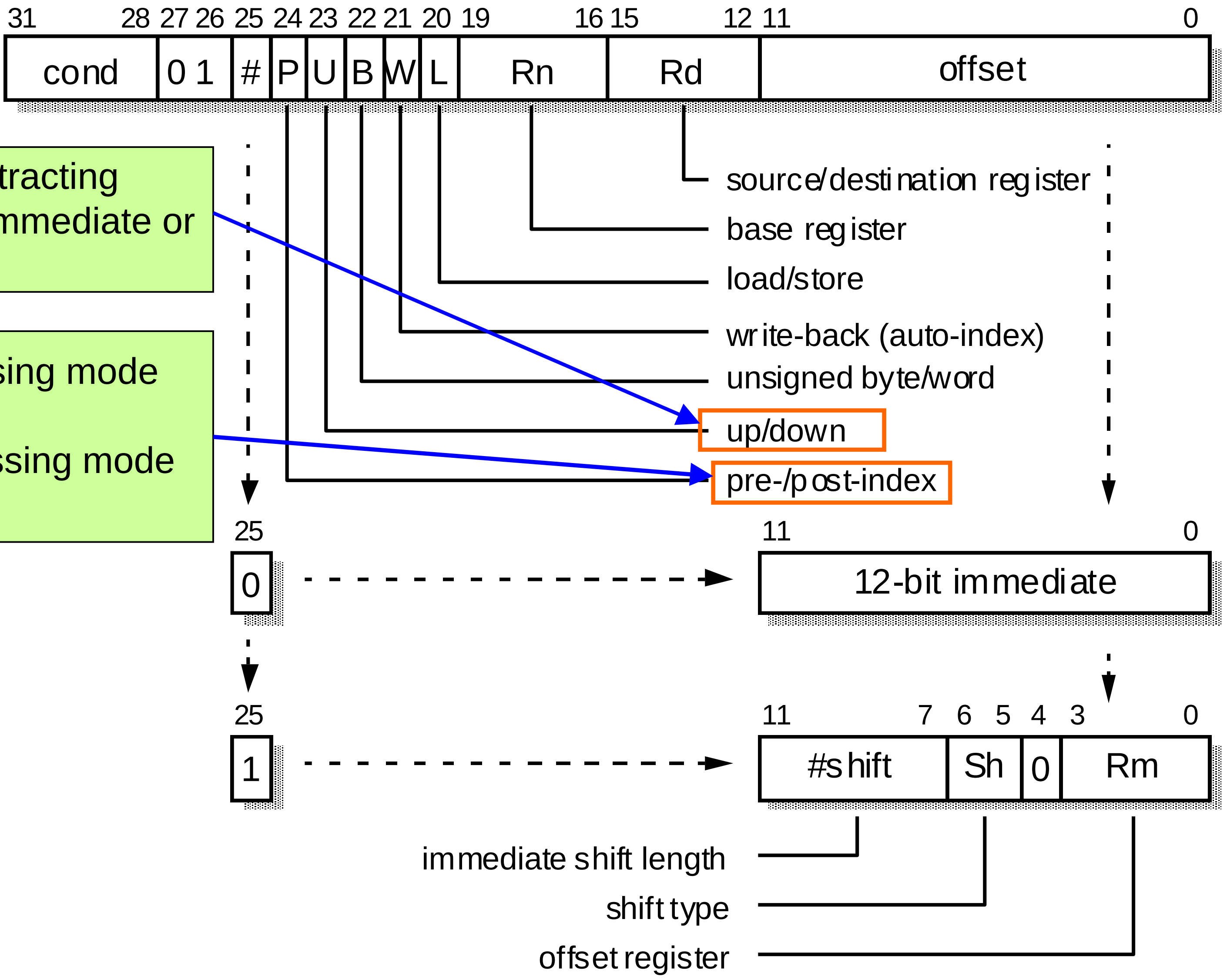
Data Transfer Instructions

- Single word and unsigned byte data transfer instructions
- Half-word and singed byte data transfer instructions
- Multiple register transfer instructions
- Swap memory and register instructions (SWP)
- Status register to general register transfer instructions
- General register to status register transfer instructions

Single word and unsigned byte data transfer instruction binary encoding

Adding (U=1) or subtracting (U=0) an unsigned immediate or register offset

Pre-indexed addressing mode (P=1)
Post-indexed addressing mode (P=0)



Assembler Syntax

Pre-indexed form

LDR|**STR**{cond}{B}   Rd, [Rn, <offset>]{!}

Post-indexed form

LDR|**STR**{cond}{B}{T}   Rd, [Rn], <offset>

- "T" selects the user view of the memory translation and protection system and should only be used in non-user modes
- If the instruction is executed when the processor is in a privileged mode, the memory system is signaled to treat the access as if the processor were in User mode

Assembler Syntax

Pre-indexed form

LDR|STR{cond}{B}   Rd, [Rn, <offset>]{!}

Post-indexed form

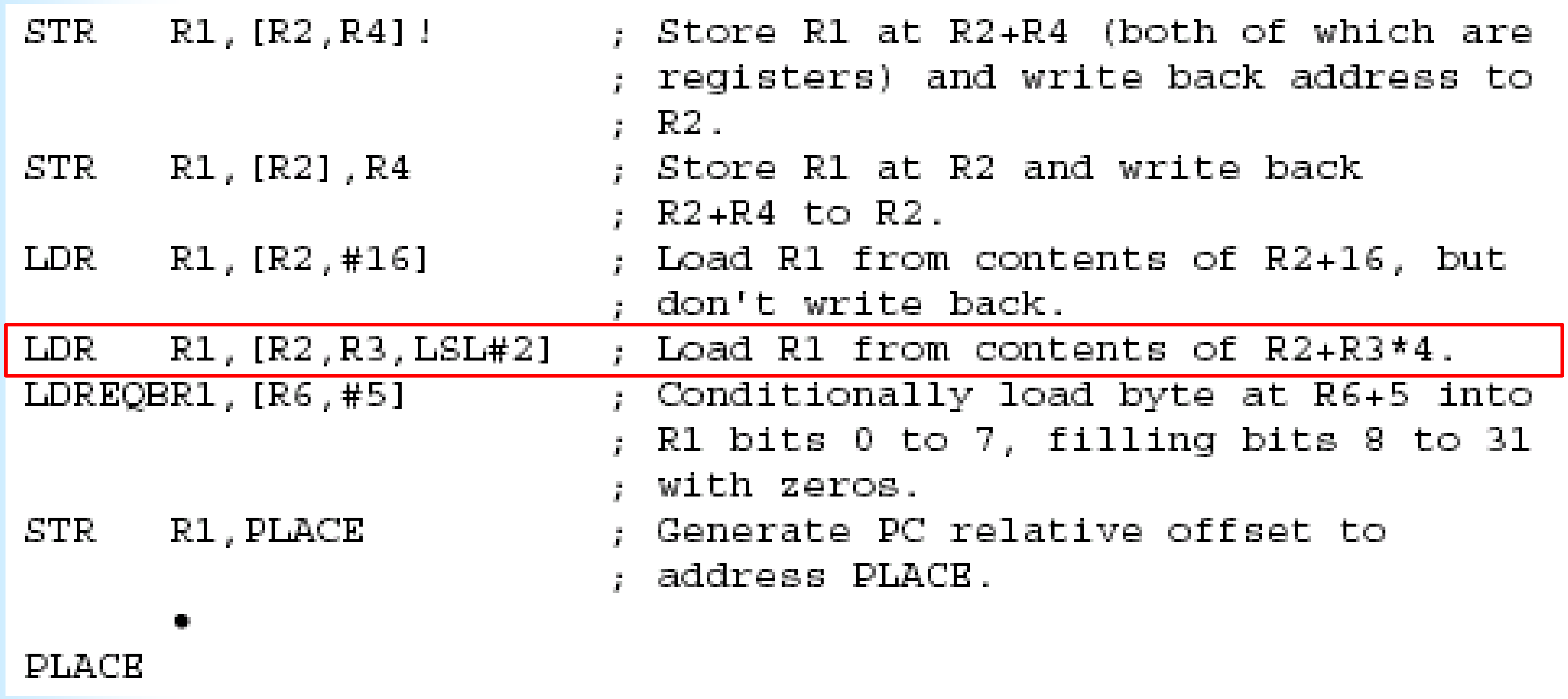
LDR|STR{cond}{B}{T} Rd, [Rn], <offset>

PC-relative form

LDR|STR{cond}{B}   Rd, LABEL

- "B" selects an unsigned byte transfer, the default is word
- <offset>
  - +/- <12-bit immediate>
  - +/- Rm {, shift}

Assembler Example

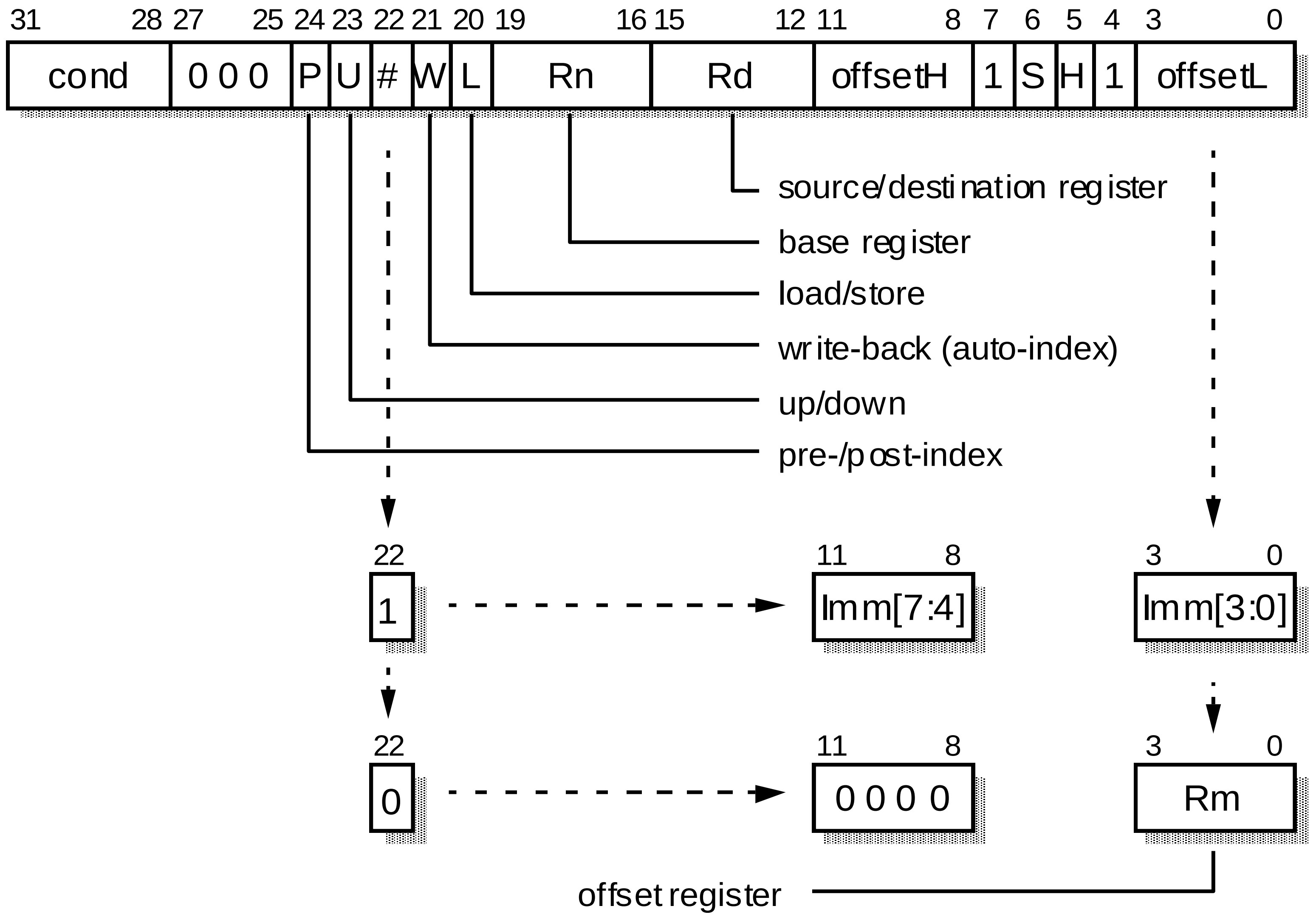


```
STR    R1,[R2,R4]!          ; Store R1 at R2+R4 (both of which are
                            ; registers) and write back address to
                            ; R2.
STR    R1,[R2],R4           ; Store R1 at R2 and write back
                            ; R2+R4 to R2.
LDR    R1,[R2,#16]          ; Load R1 from contents of R2+16, but
                            ; don't write back.
LDR    R1,[R2,R3,LSL#2]     ; Load R1 from contents of R2+R3*4.
LDREQBR1,[R6,#5]            ; Conditionally load byte at R6+5 into
                            ; R1 bits 0 to 7, filling bits 8 to 31
                            ; with zeros.
STR    R1,PLACE             ; Generate PC relative offset to
                            ; address PLACE.
       •
PLACE
```

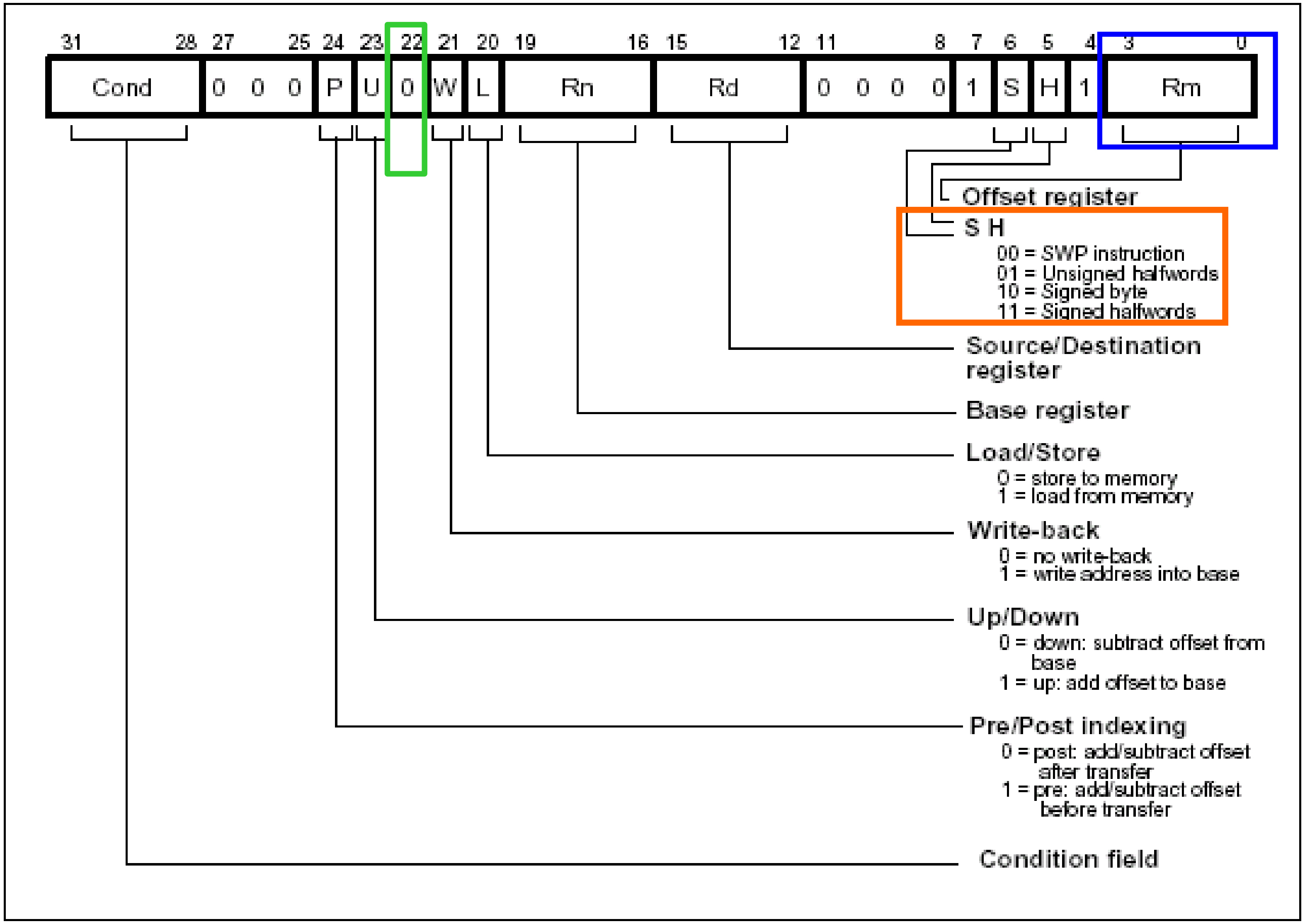
Data Transfer Instructions

- Single word and unsigned byte data transfer instructions
- Half-word and singed byte data transfer instructions
- Multiple register transfer instructions
- Swap memory and register instructions (SWP)
- Status register to general register transfer instructions
- General register to status register transfer instructions

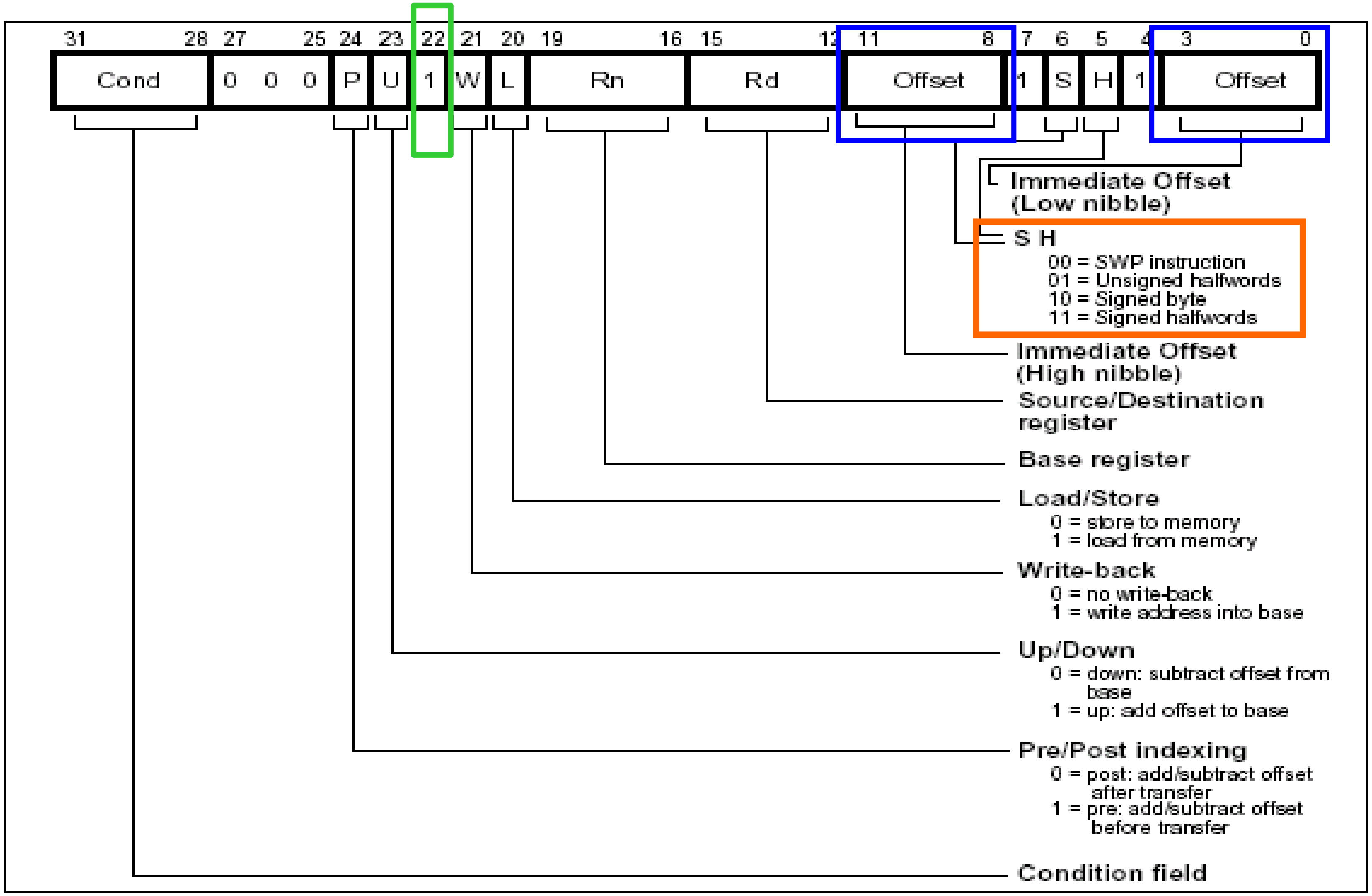
Half-word and Signed Byte Data Transfer Instruction Binary Encoding



Half-word and Signed Data Transfer with Register Offset



Half-word and Signed Data Transfer with Immediate Offset



Assembler Syntax

Pre-indexed form

LDR|**STR**{cond}<H|SH|SB>   Rd, [Rn, <offset>]{!}

Post-indexed form

LDR|**STR**{cond}<H|SH|SB>   Rd, [Rn], <offset>

- <H|SH|SB> selects the data types
- <offset>
    - +/- <8-bit immediate>
    - +/- Rm

S	H	Data type
1	0	Signed byte
0	1	Unsigned half-word
1	1	Signed half-word

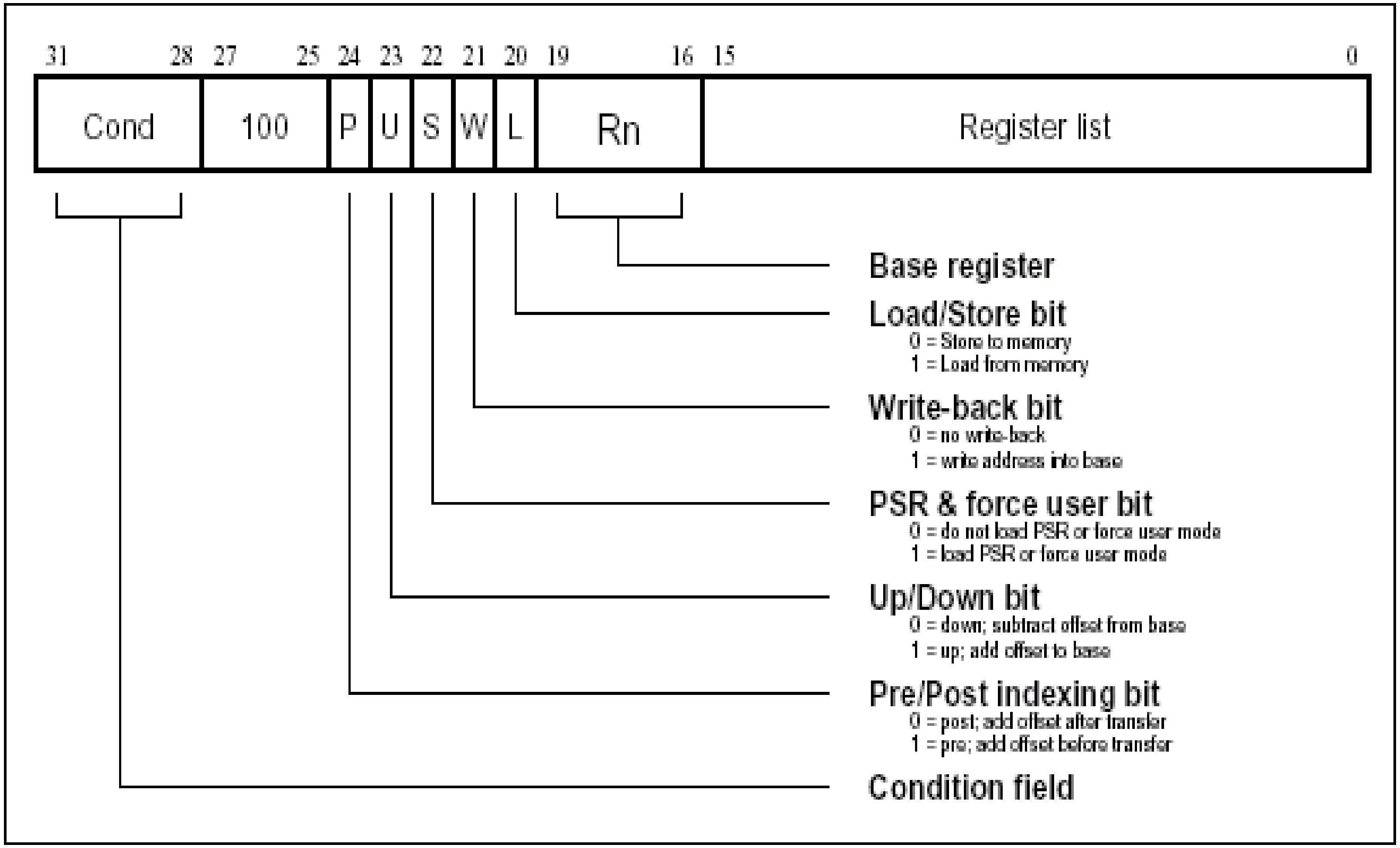
Assembler Example

```
LDRH     R1,[R2,-R3]!      ; Load R1 from the contents of the
                           ; halfword address contained in
                           ; R2-R3 (both of which are registers)
                           ; and write back address to R2
STRH     R3,[R4,#14]       ; Store the halfword in R3 at R14+14
                           ; but don't write back.
LDRSB    R8,[R2],#-223     ; Load R8 with the sign extended
                           ; contents of the byte address
                           ; contained in R2 and write back
                           ; R2-223 to R2.
LDRNESH  R11,[R0]          ; conditionally load R11 with the sign
                           ; extended contents of the halfword
                           ; address contained in R0.
```

Data Transfer Instructions

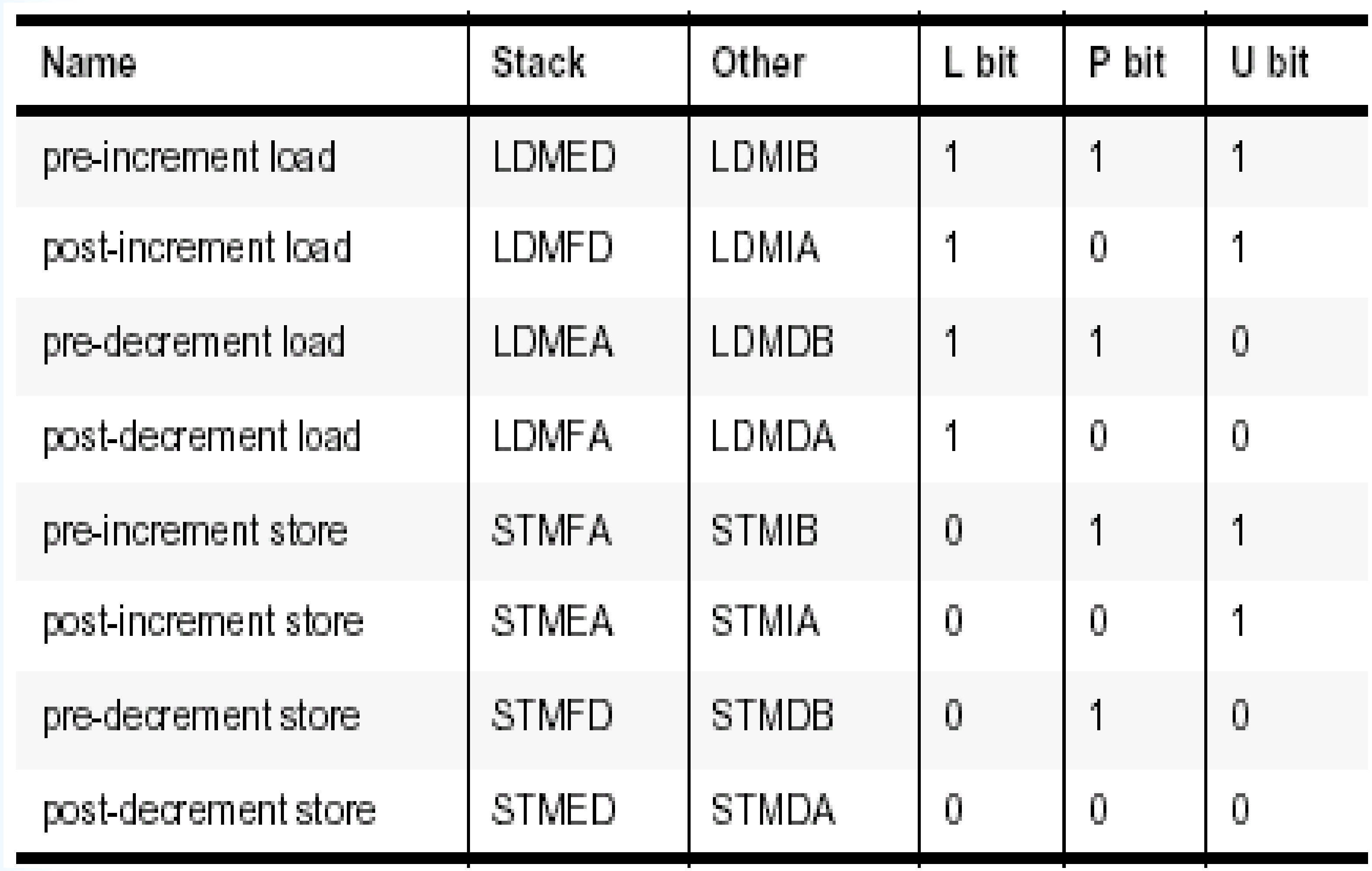
- Single word and unsigned byte data transfer instructions
- Half-word and singed byte data transfer instructions
- Multiple register transfer instructions
- Swap memory and register instructions (SWP)
- Status register to general register transfer instructions
- General register to status register transfer instructions

Multiple Register Transfer Instructions Binary Encoding



31
28
27
25
24
23
22
21
20
19
16
15
0
Cond
100
P
U
S
W
L
Rn
Register list
Base register
Load/Store bit
0 = Store to memory
1 = Load from memory
Write-back bit
0 = no write-back
1 = write address into base
PSR & force user bit
0 = do not load PSR or force user mode
1 = load PSR or force user mode
Up/Down bit
0 = down; subtract offset from base
1 = up; add offset to base
Pre/Post indexing bit
0 = post; add offset after transfer
1 = pre; add offset before transfer
Condition field

Addressing Mode Names

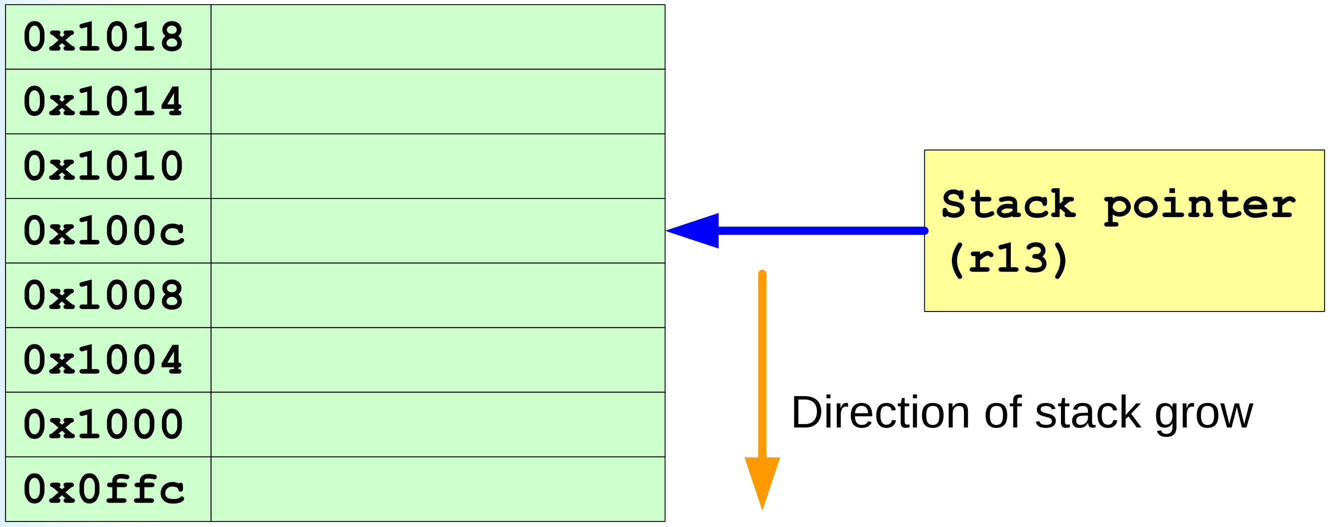


Name	Stack	Other	L bit	P bit	U bit
pre-increment load	LDMED	LDMIB	1	1	1
post-increment load	LDMFD	LDMIA	1	0	1
pre-decrement load	LDMEA	LDMDB	1	1	0
post-decrement load	LDMFA	LDMDA	1	0	0
pre-increment store	STMFA	STMIB	0	1	1
post-increment store	STMEA	STMIA	0	0	1
pre-decrement store	STMFD	STMDB	0	1	0
post-decrement store	STMED	STMDA	0	0	0

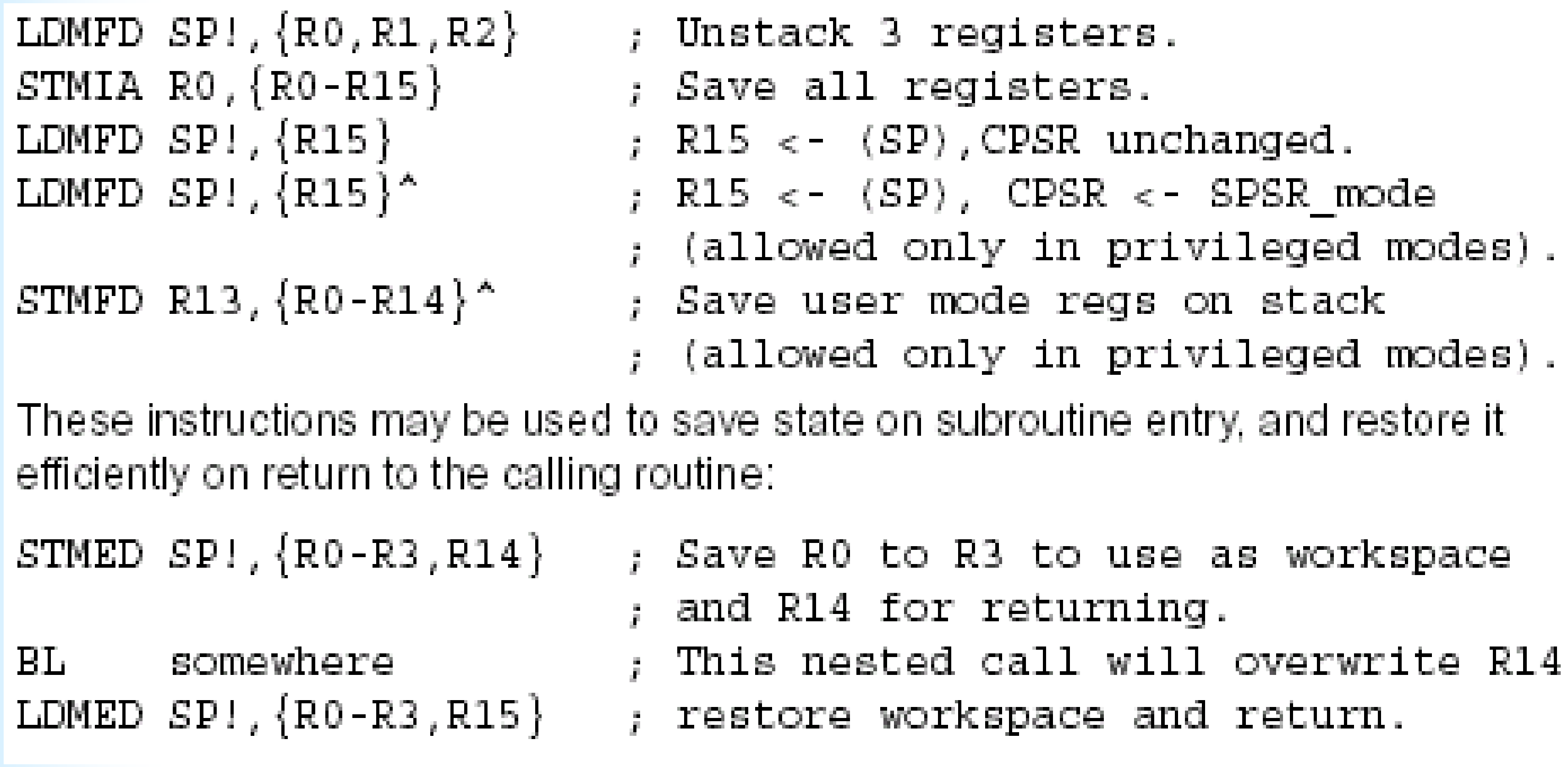
Assembler Syntax

The normal form

LDM|STM{cond}<add mode>   Rd{!}, <registers>{^}



Assembler Example



```
LDMFD SP!,{R0,R1,R2}      ; Unstack 3 registers.
STMIA R0,{R0-R15}         ; Save all registers.
LDMFD SP!,{R15}           ; R15 <- (SP),CPSR unchanged.
LDMFD SP!,{R15}^          ; R15 <- (SP), CPSR <- SPSR_mode
                          ; (allowed only in privileged modes).
STMFD R13,{R0-R14}^       ; Save user mode regs on stack
                          ; (allowed only in privileged modes).
```

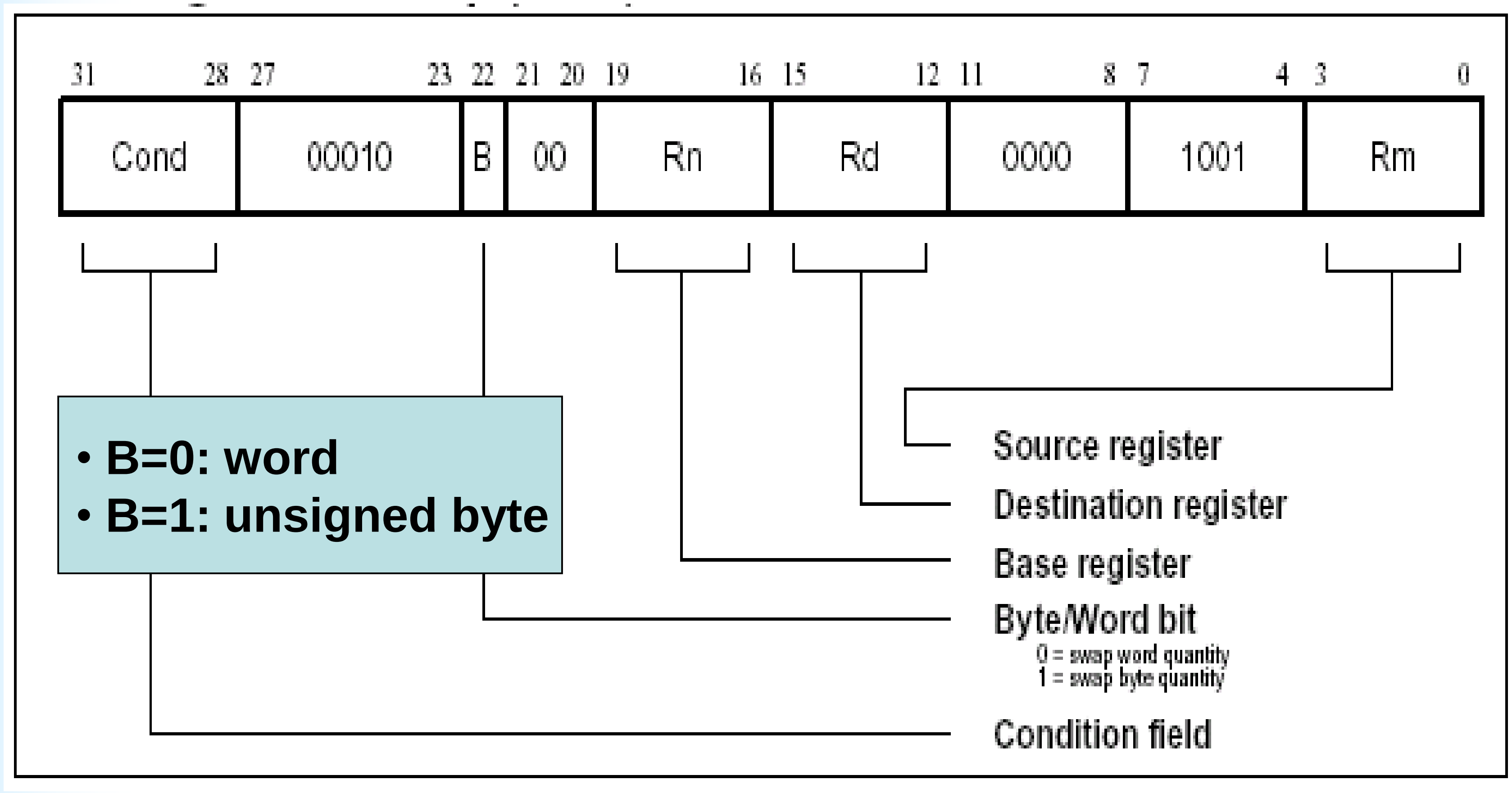
These instructions may be used to save state on subroutine entry, and restore it efficiently on return to the calling routine:

```
STMED SP!,{R0-R3,R14}     ; Save R0 to R3 to use as workspace
                          ; and R14 for returning.
BL    somewhere           ; This nested call will overwrite R14
LDMED SP!,{R0-R3,R15}     ; restore workspace and return.
```

Data Transfer Instructions

- Single word and unsigned byte data transfer instructions
- Half-word and singed byte data transfer instructions
- Multiple register transfer instructions
- Swap memory and register instructions (SWP)
- Status register to general register transfer instructions
- General register to status register transfer instructions

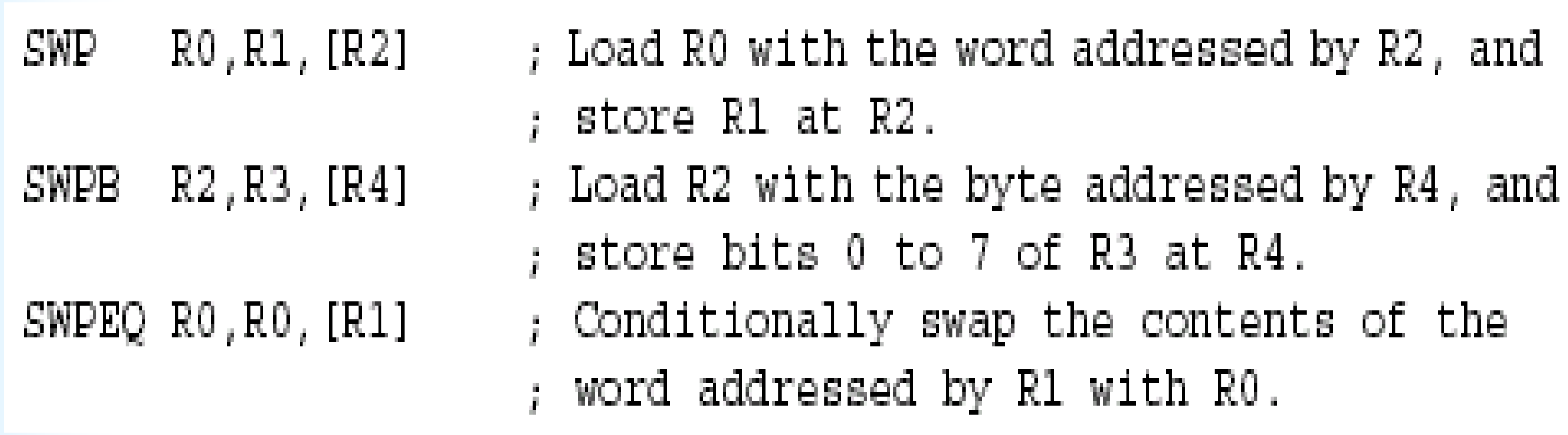
Swap Memory and Register Instruction (SWP)



- **B=0: word**
- **B=1: unsigned byte**

Assembler Syntax and Example

SWP{cond}{B}   Rd, Rm, [Rn]

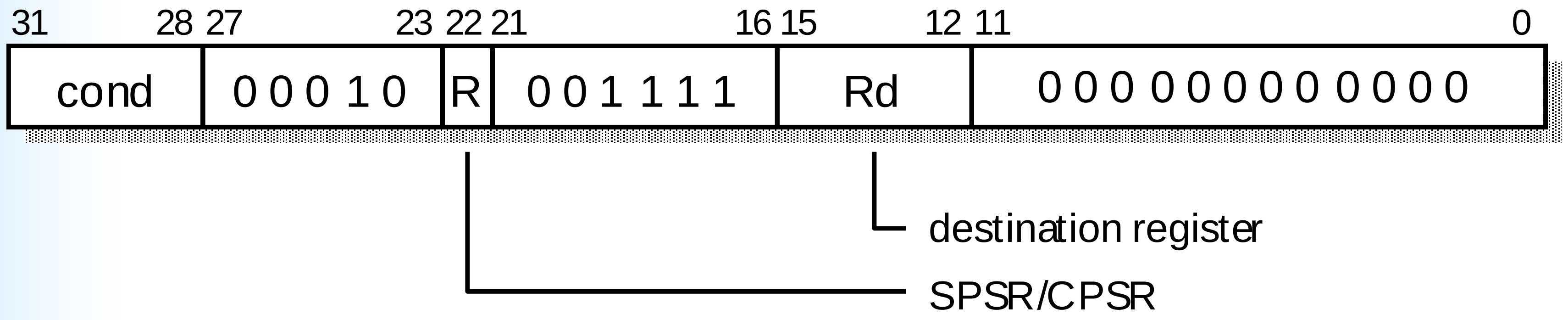


```
SWP   R0,R1,[R2]      ; Load R0 with the word addressed by R2, and
                      ; store R1 at R2.
SWPB  R2,R3,[R4]      ; Load R2 with the byte addressed by R4, and
                      ; store bits 0 to 7 of R3 at R4.
SWPEQ R0,R0,[R1]      ; Conditionally swap the contents of the
                      ; word addressed by R1 with R0.
```

Data Transfer Instructions

- Single word and unsigned byte data transfer instructions
- Half-word and singed byte data transfer instructions
- Multiple register transfer instructions
- Swap memory and register instructions (SWP)
- Status register to general register transfer instructions
- General register to status register transfer instructions

Status Register to General Register Transfer Instruction Binary Encoding



Assembler Syntax and Example

Status register to general register

MRS{cond}   Rd, CPSR|SPSR

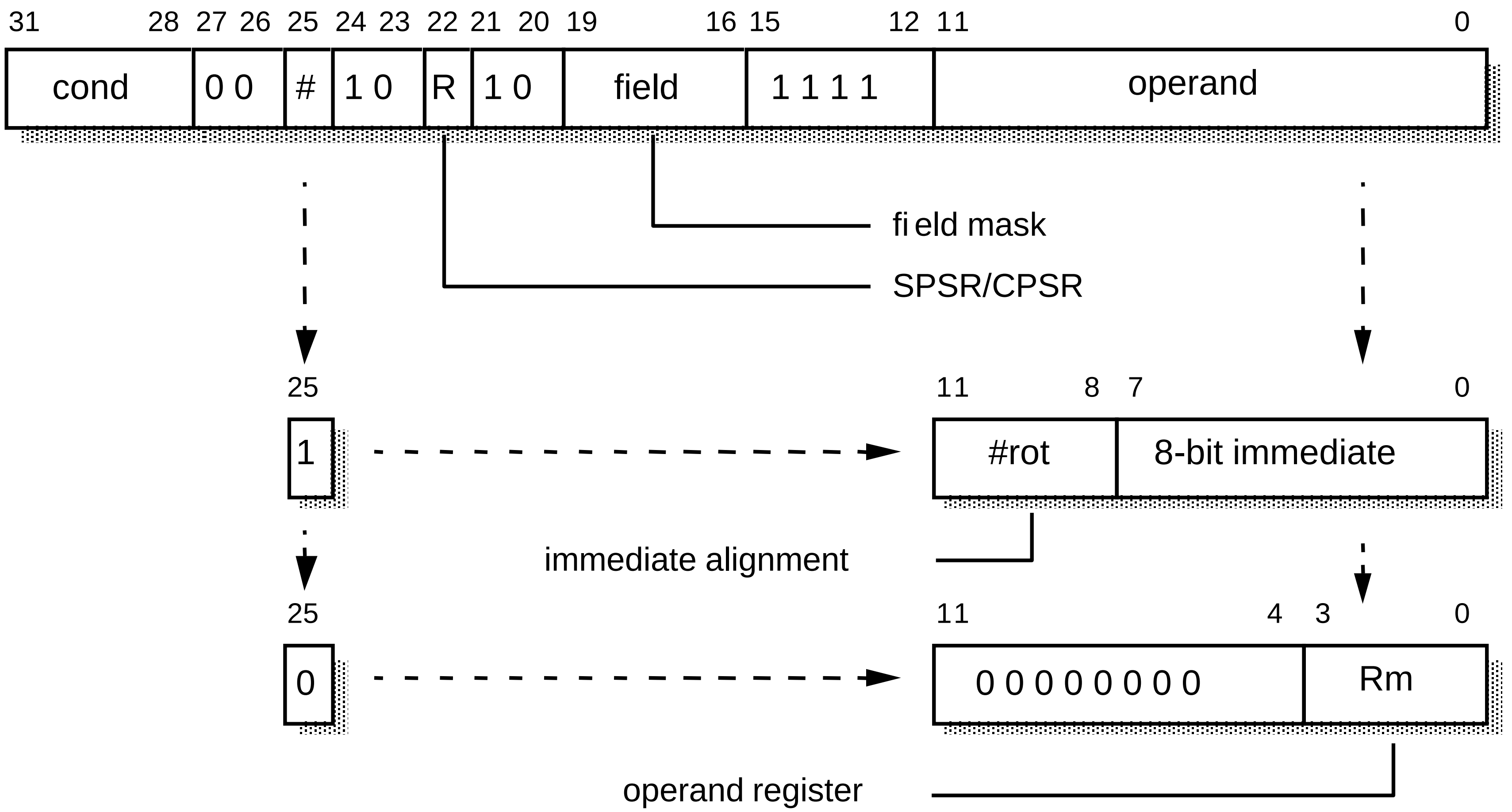
Example:

```
MRS  r0, CPSR    ; move the CPSR to r0
MRS  r3, SPSR    ; move the SPSR to r3
```

Data Transfer Instructions

- Single word and unsigned byte data transfer instructions
- Half-word and singed byte data transfer instructions
- Multiple register transfer instructions
- Swap memory and register instructions (SWP)
- Status register to general register transfer instructions
- General register to status register transfer instructions

Transfer to Status Register Instruction Binary Encoding



Assembler Syntax

General register to status register

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

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

where <field> is one of:

- **c**: the control field	PSR[7:0]
- **x**: the extension field	PSR[15:8]
- **s**: the status field	PSR[23:16]
- **f**: the flags field	PSR[31:24]

Assembler Example

```
// To set the N, Z, C and V flags

MSR     CPSR_f, #&f0000000  ; set all the flags
```

```
// switch from supervisor mode into IRQ mode

MRS     r0, CPSR            ; move the CPSR to r0
BIC     r0, r0, #&1f        ; clear the bottom 5 bits
ORR     r0, r0, #&12        ; set the bits to IRQ mode
MSR     CPSR_c, r0          ; move back to CPSR
```

Outline

- Introduction
- Exceptions
- Conditional execution
- Branch, branch with link and eXchange
- Software Interrupt (SWI)
- Data processing instructions
- Multiply instructions
- Data transfer instructions
- Coprocessor instructions
- Others

Coprocessor Architecture

- ARM supports a general-purpose extension of its instructions set through **the addition of hardware coprocessor**
- **Coprocessor architecture**
  - Up to 16 logical coprocessors
  - Each coprocessor can have up to 16 private registers (any reasonable size)
  - Using load-store architecture and some instructions to communicate with ARM registers and memory.

Coprocessor Instructions

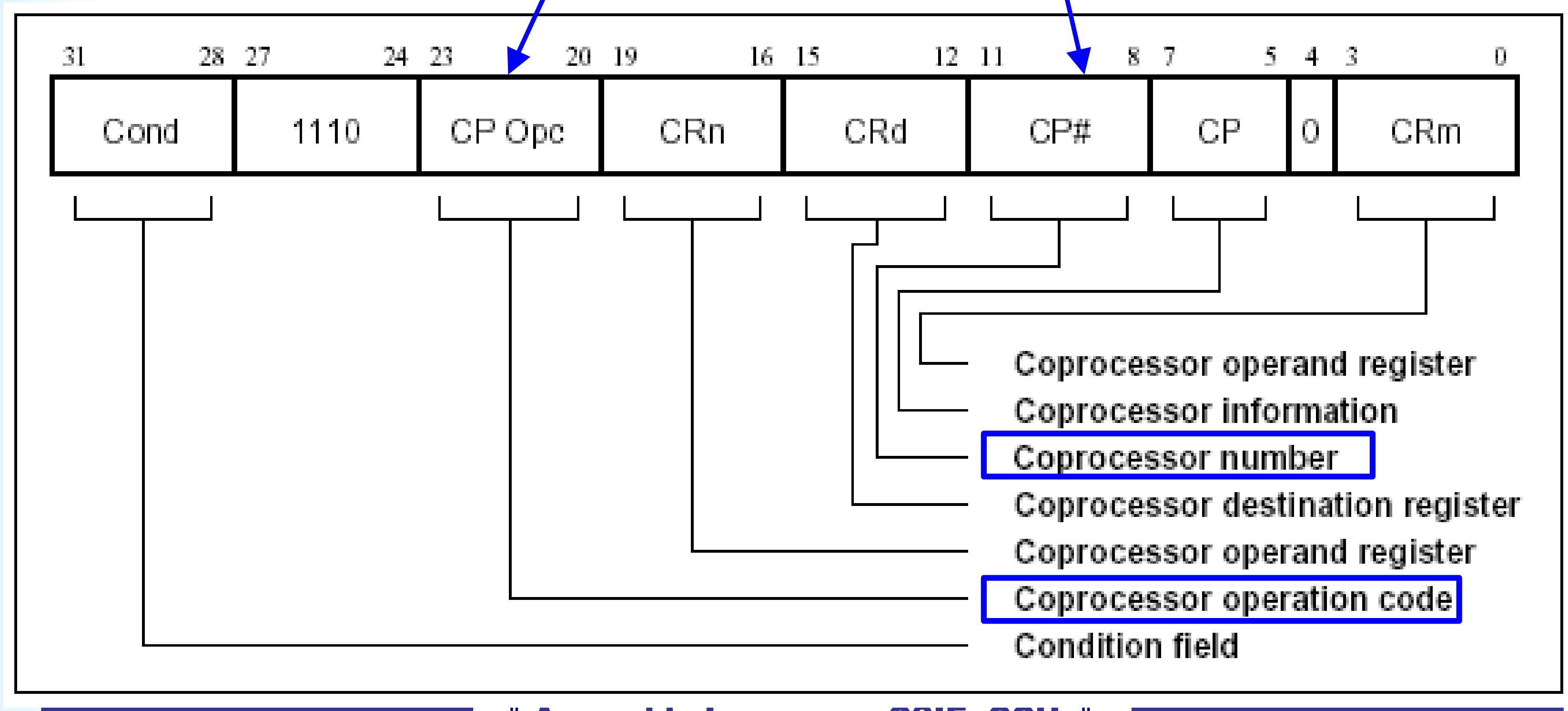
- Coprocessor registers
- Coprocessor data operations
- Coprocessor data transfers
- Coprocessor register transfers

Coprocessor Registers

- The **ARM** has sole responsibility for **control flow**
- The **coprocessor instructions** are concerned with
  - **Data processing**
  - **Data transfer**

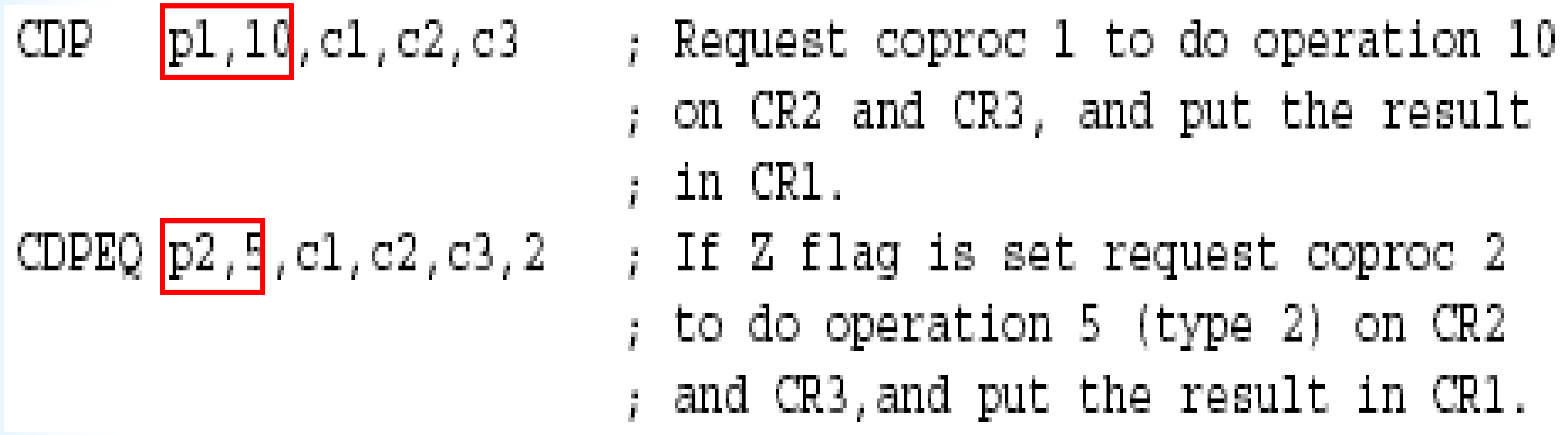
Coprocessor Data Operations

These two fields define the performed operation in the coprocessor number CP#


31
28
27
24
23
20
19
16
15
12
11
8
7
5
4
3
0
Cond
1110
CP Opc
CRn
CRd
CP#
CP
0
CRm
Coprocessor operand register
Coprocessor information
Coprocessor number
Coprocessor destination register
Coprocessor operand register
Coprocessor operation code
Condition field

Assembler Syntax and Example

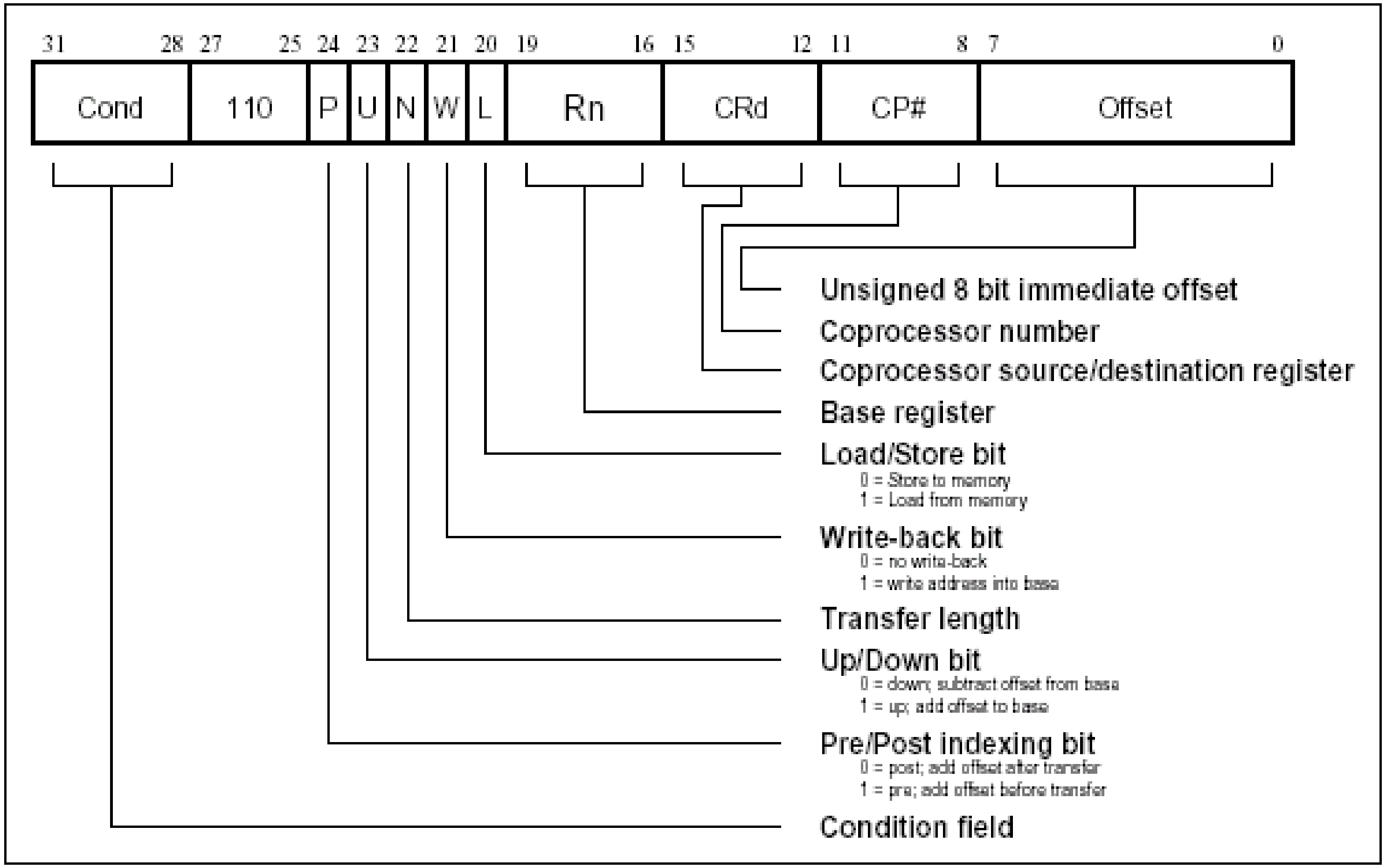
CDP{cond} <CP#>, <Cop1>, CRd, CRn, CRm {, <Cop2>}



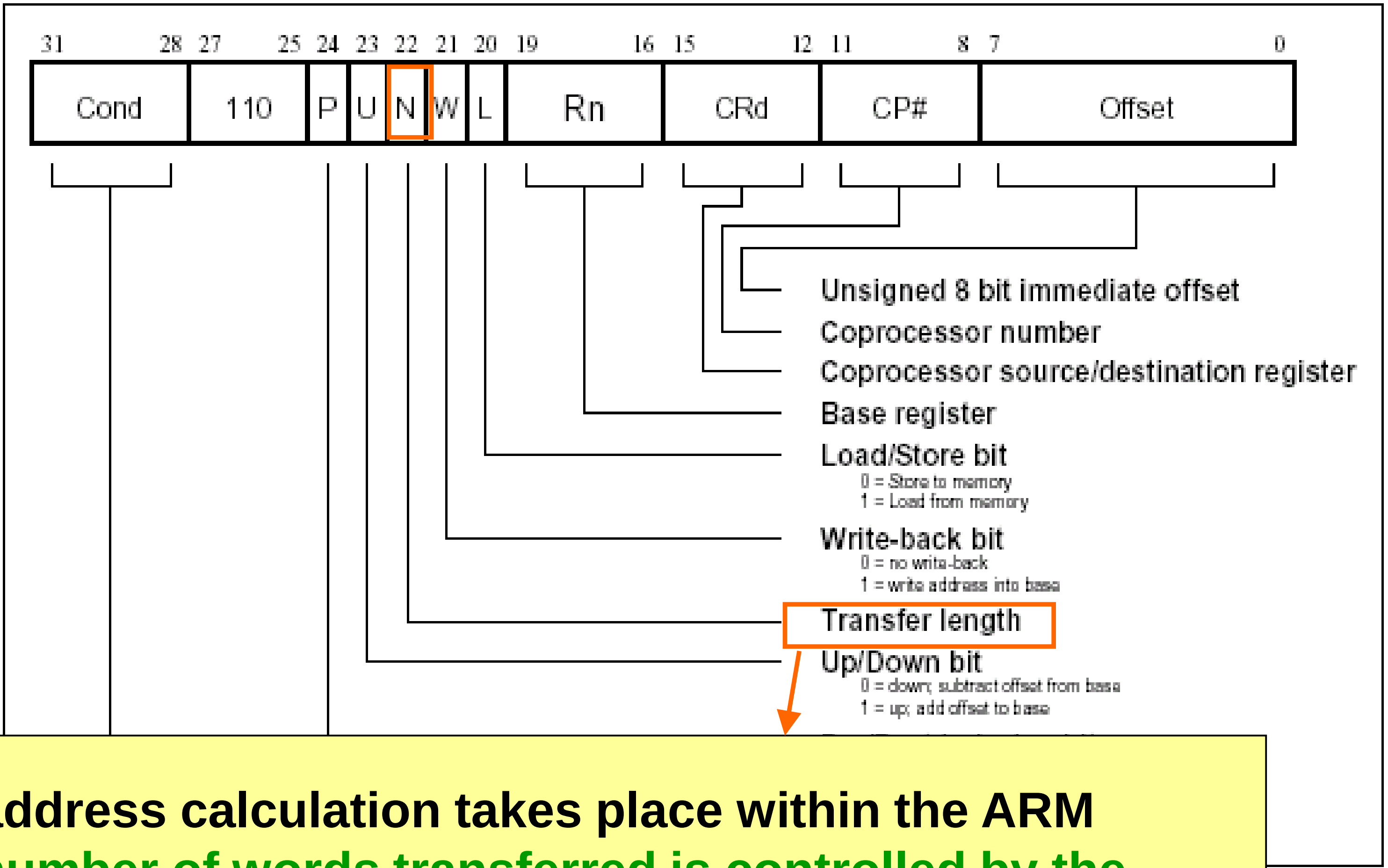
```
CDP    p1,10,c1,c2,c3        ; Request coproc 1 to do operation 10
                             ; on CR2 and CR3, and put the result
                             ; in CR1.
CDPEQ p2,5,c1,c2,c3,2        ; If Z flag is set request coproc 2
                             ; to do operation 5 (type 2) on CR2
                             ; and CR3,and put the result in CR1.
```

Coprocessor Data Transfers (1)

- Load data from memory to the coprocessor registers
- Store data from the coprocessor registers to memory



Coprocessor Data Transfers (2)



- The address calculation takes place within the ARM
- The number of words transferred is controlled by the coprocessor
- The N bit select one of two possible lengths

Assembler Syntax and Example

The pre-indexed form

LDC|STC{cond}{L} <CP#>, CRd, [Rn, <offset>]{!}

The post-indexed form

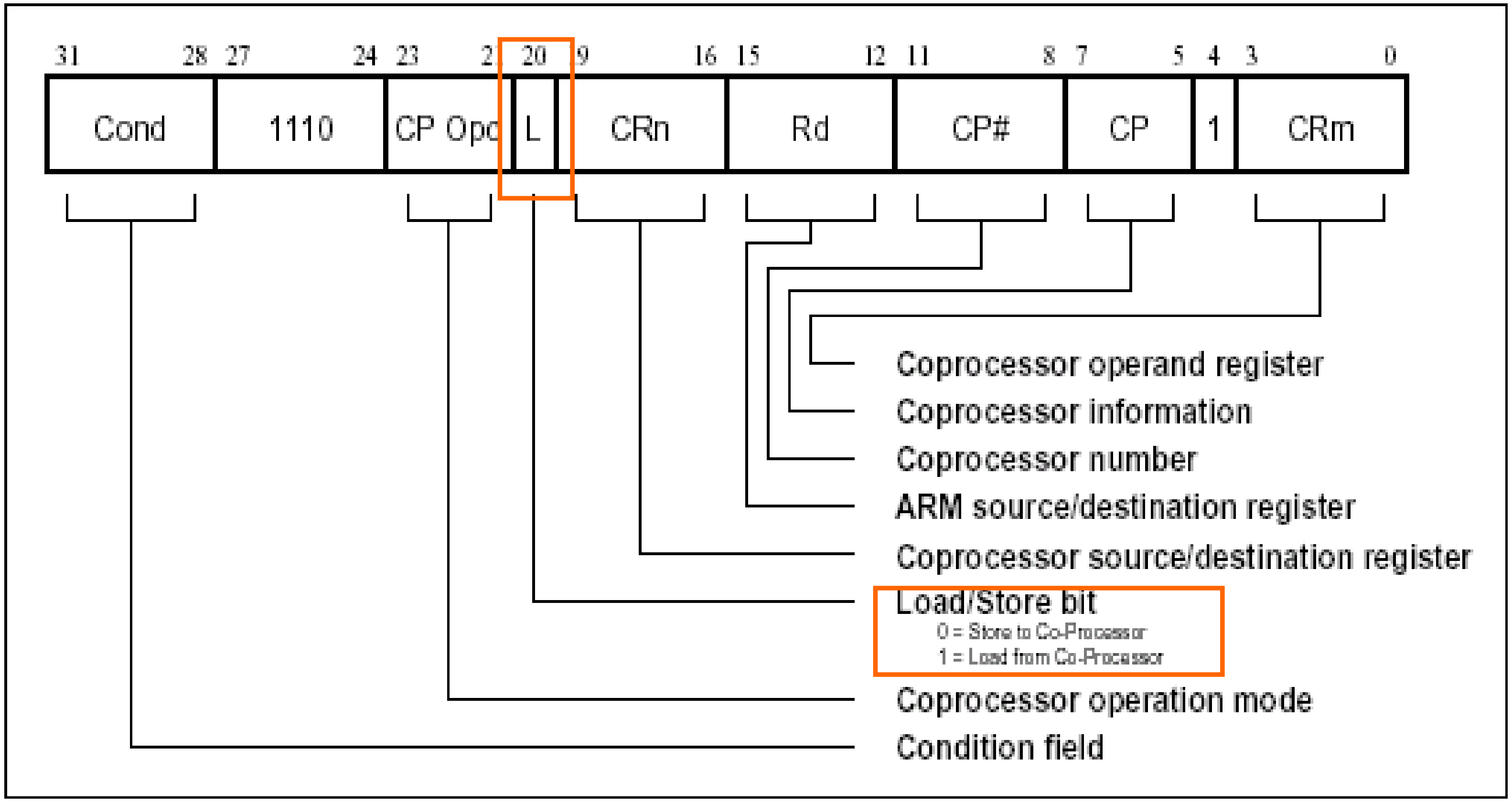
LDC|STC{cond}{L} <CP#>, CRd, [Rn], <offset>

```
LDC    p1,c2,table      ; Load c2 of coproc 1 from address
                        ; table, using a PC relative address.
STCEQL p2,c3,[R5,#24]!; Conditionally store c3 of coproc 2
                        ; into an address 24 bytes up from R5,
                        ; write this address back to R5, and use
                        ; long transfer option (probably to
                        ; store multiple words).
```

- **Limiting the maximum transfer length to 16 words will ensure that coprocessor data transfer take no longer than worst-case load and store multiple register instructions**

Coprocessor Register Transfers

• These instructions allow an integer generated in a coprocessor to be transferred directly into a ARM register or the ARM condition code flags



31 28 27 24 23 21 20 19 16 15 12 11 8 7 5 4 3 0
Cond
1110
CP Opc
L
CRn
Rd
CP#
CP
1
CRm
Coprocessor operand register
Coprocessor information
Coprocessor number
ARM source/destination register
Coprocessor source/destination register
Load/Store bit
0 = Store to Co-Processor
1 = Load from Co-Processor
Coprocessor operation mode
Condition field

Typical Examples

- The conversion from floating point (**coprocessor**) to fix point (**ARM**)
- The conversion from fix point (**ARM**) to floating point (**coprocessor**)
- A floating-point comparison which returns the result of the comparison directly to the ARM condition code flags

Assembler Syntax

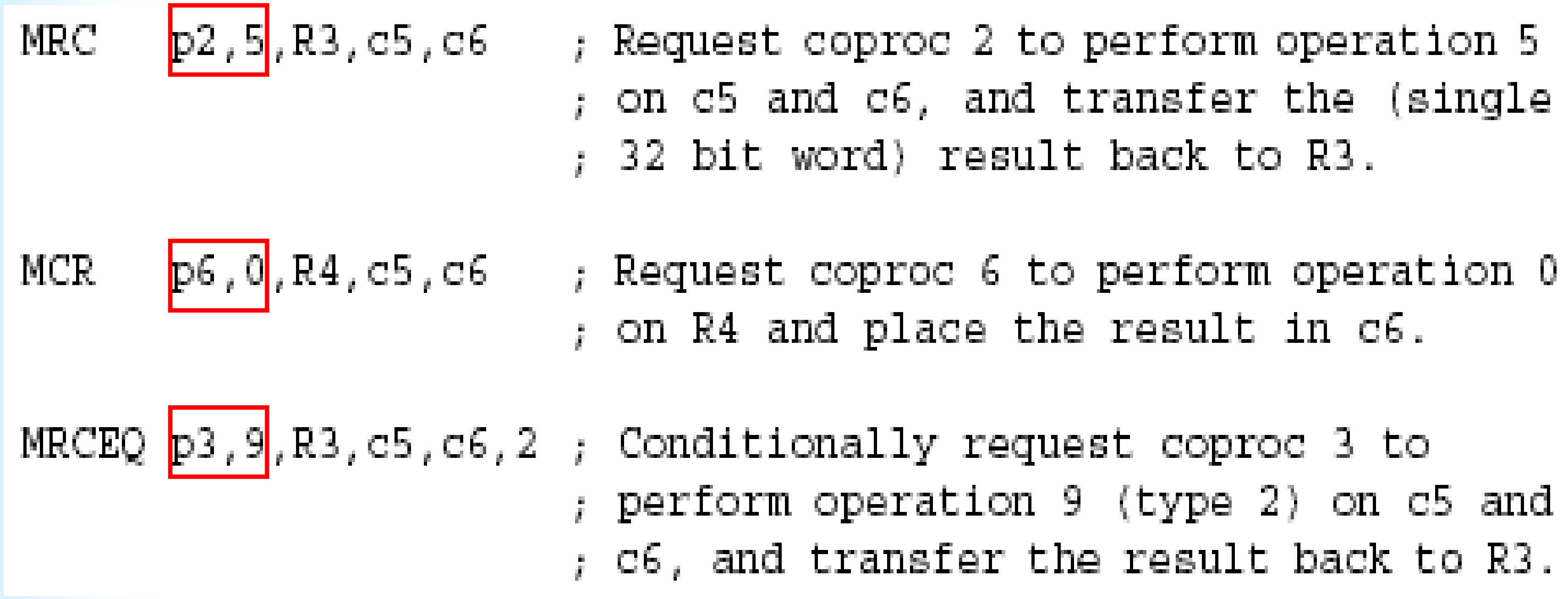
Move from coprocessor to ARM register

MRC{cond}   <CP#>, <Cop1>, Rd, CRn, CRm, {,<Cop2>}

Move from ARM register to coprocessor

MCR{cond}   <CP#>, <Cop1>, Rd, CRn, CRm, {,<Cop2>}

Assembler Example



```
MRC    p2,5,R3,c5,c6      ; Request coproc 2 to perform operation 5
                          ; on c5 and c6, and transfer the (single
                          ; 32 bit word) result back to R3.

MCR    p6,0,R4,c5,c6      ; Request coproc 6 to perform operation 0
                          ; on R4 and place the result in c6.

MRCEQ  p3,9,R3,c5,c6,2    ; Conditionally request coproc 3 to
                          ; perform operation 9 (type 2) on c5 and
                          ; c6, and transfer the result back to R3.
```

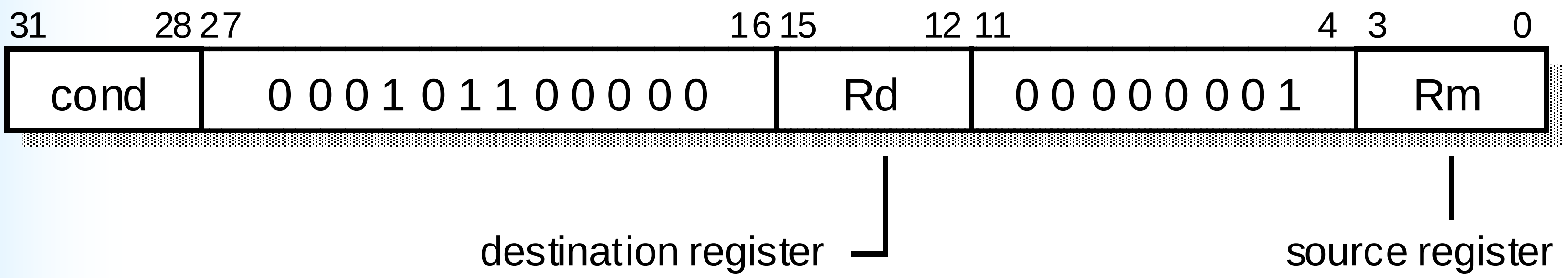
Outline

- Introduction
- Exceptions
- Conditional execution
- Branch, branch with link and eXchange
- Software Interrupt (SWI)
- Data processing instructions
- Multiply instructions
- Data transfer instructions
- Coprocessor instructions
- Others

Others

- Count leading zeros (CLZ, v5T)
- Breakpoint instruction (BKPT, v5T)
- Unused instruction space
- Memory faults
- ARM architecture variants

Count Leading Zeros (CLZ, v5T)



Assembler Syntax: CLZ{cond} Rd, Rm

Example:

```
    MOV       r0,  #&100
; r0:=0x00000000000000000000000100000000
    CLZ       r1,  r0        ; r1:=23
```

Breakpoint Instruction (BKPT, v5T)

- **Software debugging purposes**
- **They cause the processor to break from normal instruction execution and enter appropriate debugging procedures**
- **Unconditional**

31 28	27 20	19 16	15 12	11 8	7 4	3 0
1 1 1 0	0 0 0 1 0 0 1 0	x x x x	x x x x	x x x x	0 1 1 1	x x x x

Assembler syntax: BKPT

Unused Instruction Space (1)

- **The ARM processors should take the undefined instruction trap if any unused opcode is executed**
- **Old ARM version (including ARM6 and ARM7) will behave unpredictably**

Unused load/store instructions

31 28	27 25	24	23	22	21	20	19 16	15 12	11 8	7	6 5	4	3 0
cond	0 0 0	P	U	B	W	L	Rn	Rd	Rs	1	op1	1	Rm

Unused control instructions

31 28	27 23	22 21	20	19 16	15 12	11 8	7	6	4	3 0
cond	0 0 0 1 0	op1	0	Rn	Rd	Rs	op2		0	Rm
cond	0 0 0 1 0	op1	0	Rn	Rd	Rs	0	op2	1	Rm
cond	0 0 1 1 0	op1	0	Rn	Rd	#rot	8-bit immediate			

Unused Instruction Space (2)

Unused arithmetic instructions

31 28	27 22	21 20	19 16	15 12	11 8	7 4	3 0
cond	0 0 0 0 0 1	op	Rn	Rd	Rs	1 0 0 1	Rm

Unused coprocessor instructions

31 28	27 25 24	23 22	21	20	19 16	15 12	11 8	7 0
cond	1 1 0 0	op	0	X	Rn	CRd	CP#	offset

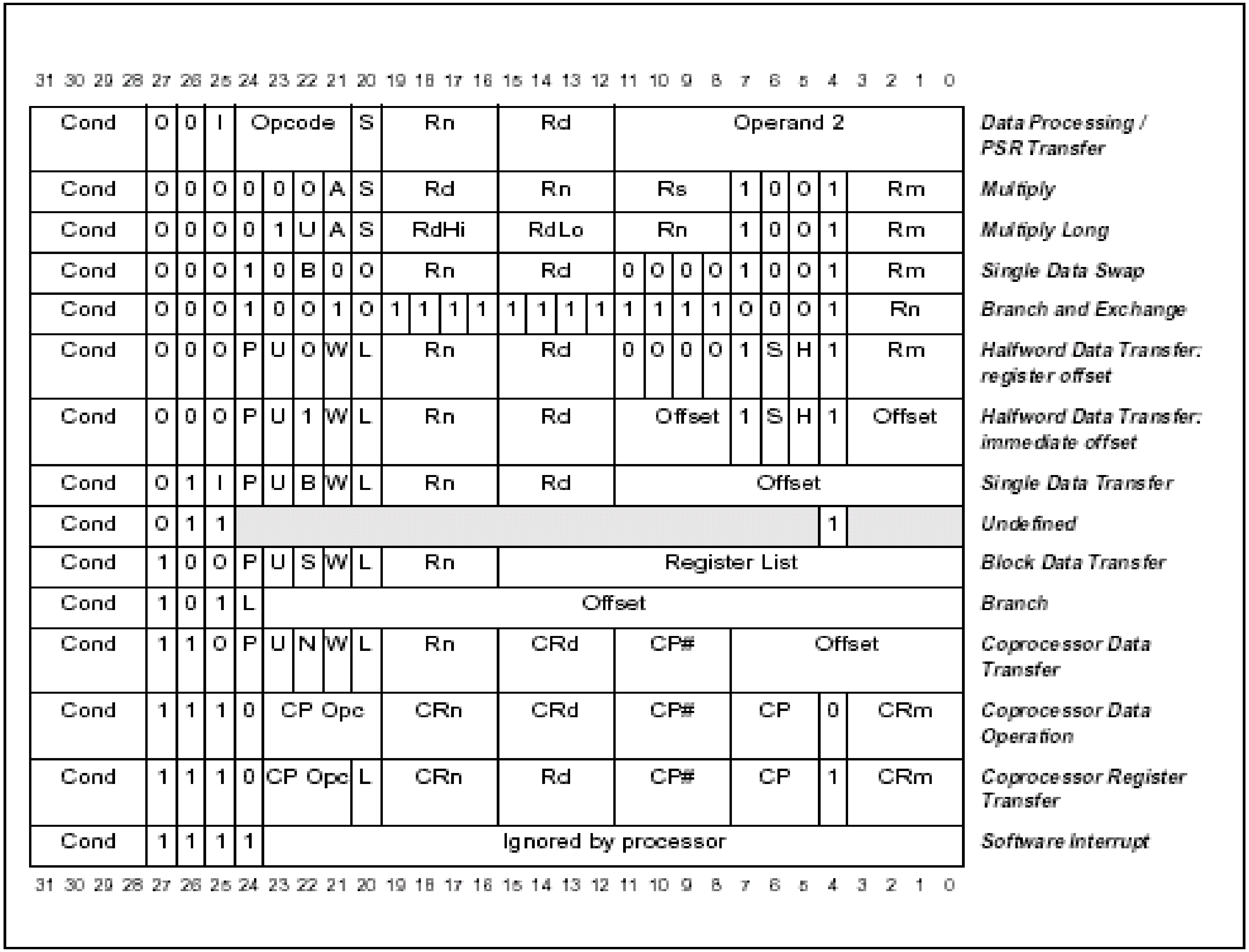
Undefined instructions space

31 28	27 25	24 5	4	3 0
cond	0 1 1	X X X X X X X X X X X X X X X X X X X X	1	X X X X

ARM Architecture Variants

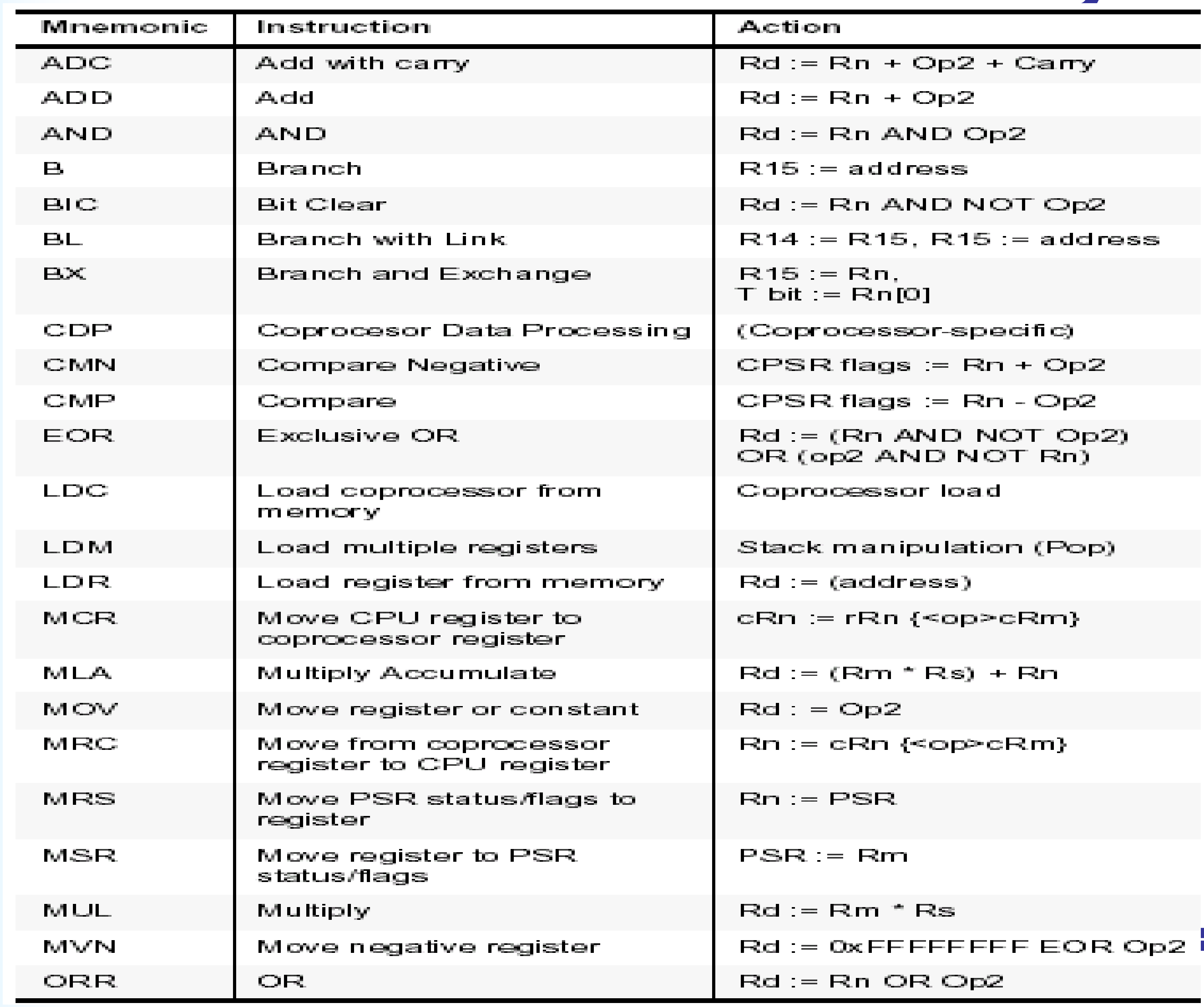
Core	Architecture
ARM1	v1
ARM2	v2
ARM2as, ARM3	v2a
ARM6, ARM600, ARM610	v3
ARM7, ARM700, ARM710	v3
ARM7TDMI, ARM710T, ARM720T, ARM740T	v4T
StrongARM, ARM8, ARM810	v4
ARM9TDMI, ARM920T, ARM940T	v4T
ARM9ES	v5TE
ARM10TDMI, ARM1020E	v5TE

Format Summary



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	I	Opcode				S	Rn				Rd				Operand 2												*Data Processing / PSR Transfer*
Cond				0	0	0	0	0	0	A	S	Rd				Rn				Rs				1	0	0	1	Rm				*Multiply*
Cond				0	0	0	0	1	U	A	S	RdHi				RdLo				Rn				1	0	0	1	Rm				*Multiply Long*
Cond				0	0	0	1	0	B	0	0	Rn				Rd				0	0	0	0	1	0	0	1	Rm				*Single Data Swap*
Cond				0	0	0	1	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	1	Rn				*Branch and Exchange*
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				Offset				1	S	H	1	Offset				*Halfword Data Transfer: immediate offset*
Cond				0	1	I	P	U	B	W	L	Rn				Rd				Offset												*Single Data Transfer*
Cond				0	1	1																					1					*Undefined*
Cond				1	0	0	P	U	S	W	L	Rn				Register List																*Block Data Transfer*
Cond				1	0	1	L	Offset																								*Branch*
Cond				1	1	0	P	U	N	W	L	Rn				CRd				CP#				Offset								*Coprocessor Data Transfer*
Cond				1	1	1	0	CP Opc				CRn				CRd				CP#				CP			0	CRm				*Coprocessor Data Operation*
Cond				1	1	1	0	CP Opc			L	CRn				Rd				CP#				CP			1	CRm				*Coprocessor Register Transfer*
Cond				1	1	1	1	Ignored by processor																								*Software Interrupt*
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	

Instruction Summary



Mnemonic	Instruction	Action
ADC	Add with carry	Rd := Rn + Op2 + Carry
ADD	Add	Rd := Rn + Op2
AND	AND	Rd := Rn AND Op2
B	Branch	R15 := address
BIC	Bit Clear	Rd := Rn AND NOT Op2
BL	Branch with Link	R14 := R15, R15 := address
BX	Branch and Exchange	R15 := Rn, T bit := Rn[0]
CDP	Coprocesor Data Processing	(Coprocessor-specific)
CMN	Compare Negative	CPSR flags := Rn + Op2
CMP	Compare	CPSR flags := Rn - Op2
EOR	Exclusive OR	Rd := (Rn AND NOT Op2) OR (op2 AND NOT Rn)
LDC	Load coprocessor from memory	Coprocessor load
LDM	Load multiple registers	Stack manipulation (Pop)
LDR	Load register from memory	Rd := (address)
MCR	Move CPU register to coprocessor register	cRn := rRn {<op>cRm}
MLA	Multiply Accumulate	Rd := (Rm * Rs) + Rn
MOV	Move register or constant	Rd : = Op2
MRC	Move from coprocessor register to CPU register	Rn := cRn {<op>cRm}
MRS	Move PSR status/flags to register	Rn := PSR
MSR	Move register to PSR status/flags	PSR := Rm
MUL	Multiply	Rd := Rm * Rs
MVN	Move negative register	Rd := 0xFFFFFFFF EOR Op2
ORR	OR	Rd := Rn OR Op2

Instruction Summary (cont'd)

Mnemonic	Instruction	Action
RSB	Reverse Subtract	Rd := Op2 - Rn
RSC	Reverse Subtract with Carry	Rd := Op2 - Rn - 1 + Carry
SBC	Subtract with Carry	Rd := Rn - Op2 - 1 + Carry
STC	Store coprocessor register to memory	address := CRn
STM	Store Multiple	Stack manipulation (Push)
STR	Store register to memory	<address> := Rd
SUB	Subtract	Rd := Rn - Op2
SWI	Software Interrupt	OS call
SWP	Swap register with memory	Rd := [Rn], [Rn] := Rm
TEQ	Test bitwise equality	CPSR flags := Rn EOR Op2
TST	Test bits	CPSR flags := Rn AND Op2

- Backup

Memory Faults

- ARM processors allow the memory system to fault on any memory access
- The memory system returns a signal to indicate the memory access has failed
- The processor will then enter an exception handler
- The system software will attempt to recover from the problem

The Common Sources of Memory Faults

- Page absent
- Page protected
- Soft memory errors
- Embedded systems
- Prefetch aborts
- Data aborts
- LDM data abort

Page Absent

- The addressed memory location has been paged out to disk
  - This will cause MMU to abort the access
  - The system software will identify the abort
  - Fetch the required page into memory from disk
  - Change the translation table in MMU
  - Retry the aborted access

Page Protected

- The addressed memory location is temporarily inaccessible
- When a page is loaded into memory, OS make it read only
  - An attempt to write the page will fault
  - Alert OS that the page has been modified and must be saved when it is swapped out to disk again

Soft Memory Errors

- A soft error has been detected in the memory
  - Such as parity check
  - The faulting process must be terminated

Embedded Systems

- HD is usually unavailable in embedded systems
  - The memory is small
  - Error detection is rarely incorporated
  - Use a library of routine stored in ROM

Prefetch Aborts

- If an instruction fetch faults, the memory raises the abort signal, and returns a meaningless word
- Recover the exception

Data Aborts

- It raises during an access to memory for a data
- Try to recover from the abort

LDM Data Abort

- The accesses may be accessed across page boundary
- This must prevent the PC from being overwritten
- The PC and the base (modified) register can be preserved