Embedded Systems

# Ch 12B ARM Assembly Language

Byung Kook Kim
Dept of EECS
Korea Advanced Institute of Science and Technology

# Overview

- *6. Exceptions*
- *7. Conditional Execution*
- *8. Branch Instructions*
- *9. Software Interrupt (SWI)*
- *10. Data Processing Instructions*
- *11. Special Data Transfer Instructions*
- *12. Coprocessor Instructions*
- *13. Miscellaneous Instructions*

- References
  - Steve Furber, “ARM System-on-chip architecture”, Second Edition, Addison Wesley, 2000.

# 6. Exceptions

- **ARM exceptions**
  - 1. Exceptions generated as the direct effect of executing an instruction
    - Software interrupts,
    - Undefined instructions (coprocessor instructions where the requested coprocessor is absent),
    - Prefetch aborts (invalid instruction due to memory fault)
  - 2. Exceptions generated as a side-effect of an instruction
    - Data aborts (memory fault during load/store)
  - 3. Exceptions generated externally, unrelated to the instruction flow
    - Reset
    - IRQ
    - FIQ

# Exceptions (II)

- **Exception entry**
  - Changes to the operating mode corresponding to the particular exception
  - Saves the address of the instruction following the exception entry instruction in r14 of the new mode
  - Saves the old value of the CPSR in the SPSR of the new mode
  - Disables IRQs by setting bit 7 of the CPSR and, if the exception is a fast interrupt, disables further fast interrupts by setting bit 6 of the CPSR
  - Forces the PC to begin executing at the relevant vector address.

| **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 |

# Exceptions (III)

- **Exception return**
  - Any modified user registers must be restored from the handler's stack
  - The CPSR must be restored from the appropriate SPSR
  - The PC must be changed back to the relevant instruction address in the user instruction stream

  - When the return address is in r14
    - 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 ; One instruction early`
    - To return from a data abort to retry the data access:
      - `SUBS pc, r14, #8 ; Two instruction early`

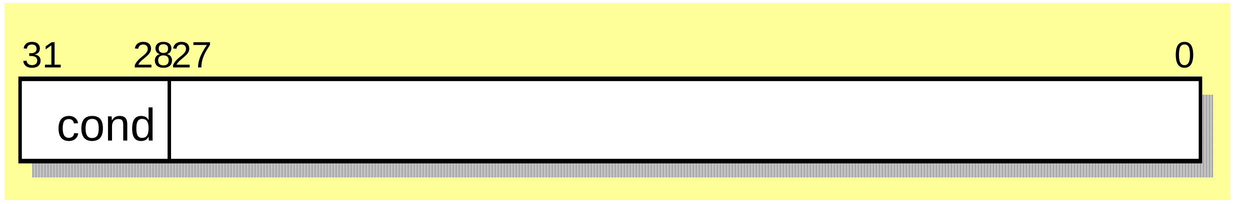
  - Restore user registers
    - `LDMFD r13!, {r0-r3,pc}^ ; PC, CPSR restored simultaneously.`

# Exceptions (IV)

- **Exception priorities**
  - Order in which the exceptions are handled
    - 1. Reset (highest priority)
    - 2. Data abort
    - 3. FIQ
    - 4. IRQ
    - 5. Prefetch abort
    - 6. SWI, undefined instruction.
- Address exceptions ; An old exception
  - Vector address 0x00000014
  - Earlier ARM processors with 26-bit address space
  - Load/store outside the address apace.

## 7. Conditional Execution

- **Condition field**
  - Top four bits of all ARM instructions



  - Instructions to be executed or skipped according to the values of N, Z, C, and V flags in the CSPR
  - Do not use the NV (never) condition

# Conditional Execution (II)

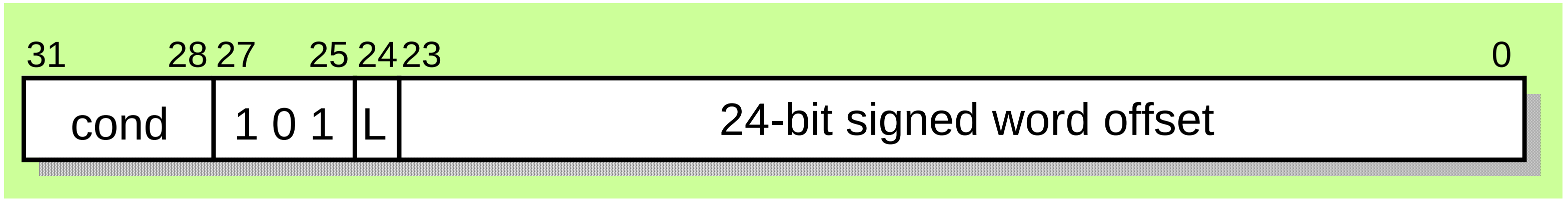
- **ARM condition codes**

| 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 |

# 8. Branch Instructions

## 8A. Branch and Branch with Link (B, BL)

- **Branch & branch with link**
  - Standard way to cause a switch in the sequence of instruction execution
  - Binary encoding:



  - Branch range: +-32 Mbytes
    - PC + (Offset << 2) + 8
  - Assembler format
    - B{L}{<cond>} <target adress>
  - BL: Move address of the next instruction into the link register (r14).

## Branch and Branch with Link (II)

- **Branch examples**
  - An unconditional jump

    ```
            B       LABEL              ; Unconditional jump
            ...
    LABEL   ...                        ; To here
    ```

  - Execute loop 10 times

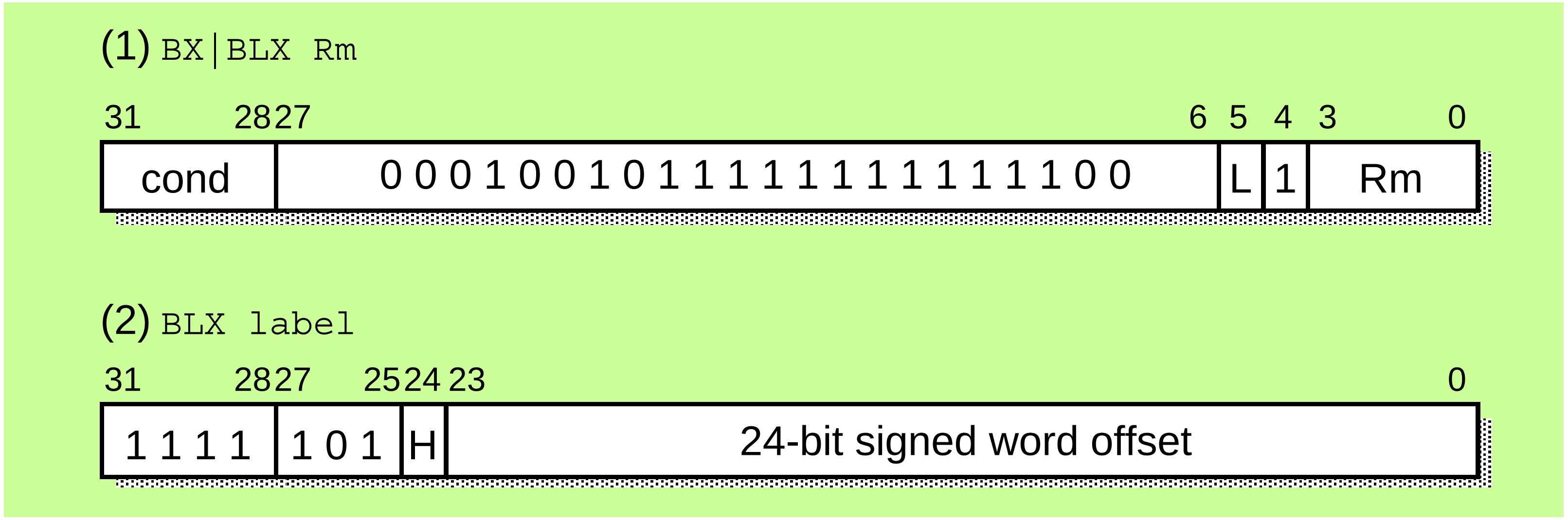
    ```
            MOV     r0, #10            ; Initialize loop counter
    LOOP    ...
            SUBS    r0, #1             ; Decrement counter setting CCs
            BNE     LOOP               ; Loop if counter <>0
            ...                        ; Else drop through
    ```

  - Call a subroutine

    ```
            BL      SUBR               ; Branch and link to SUBR
            ...                        ; Returns here
            ...
    SUBR    ...                        ; Subroutine entry point
            MOV     PC, r14            ; Return
    ```

## 8B. Branch, Branch with Link and Exchange (BX, BLX)

- **Branch, branch with link and exchange**
  - Thumb support
    - Switching the processor to execute Thumb instructions
    - Returning symmetrically to ARM and Thumb calling routines
    - On ARM processors supporting architecture v5T
  - Binary encoding



## Branch, Branch with Link and Exchange (II)

- **Description of BX instruction**
  - First format
    - Branch target is specified in a register, Rm
      - Bit[0] of Rm is copied into the T bits in the CPSR
      - Bit[31:1] are moved into the PC
    - If Rm[0] is 1, the processor switches to execute Thumb instructions.
    - If Rm[0] is 0, the processor continues executing ARM instructions. Aligned to a word boundary (Clearing Rm[1:0]).
  - Second format
    - Branch target: PC + (offset << 2) + 8
    - The H bit (bit 24) is also added into bit 1 of the resulting address
    - Range of branch: +-32 Mbytes
  - BLX
    - Moves the address of the next instruction into the link register (r14).

# Branch, Branch with Link and Exchange (III)

- **BL/BLX Assembler format**
  - 1: B{L}X{<cond>} Rm
  - 2: BLX <target address>

- Examples
  - Unconditional jump

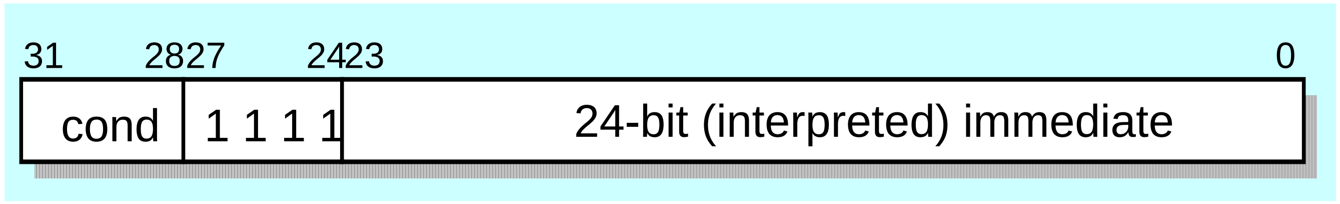
```
        BX      r0              ; Branch to address in r0
                                ; Enter Thumb state if r0[0]=1
```

  - A call to a Thumb subroutine

```
        CODE32                  ; ARM code follows
        ...
        BLX     TSUB            ; Call Thumb subroutine
        ...
        CODE16                  ; Start of Thumb code
TSUB    ...                     ; Thumb subroutine
        BX      r14             ; Return to ARM code.
```

# 9. Software Interrupt (SWI)

- Binary encoding



- Description
  - The 24-bit immediate field does not influence the operation of the instruction but may be interpreted by the system code.
  - If the condition is passed:
    - 1. Save the address of the instruction after the SWI in r14_svc
    - 2. Save the CPSR in SPSR_svc
    - 3. Enter supervisor mode and disable IRQs (but not FIQs) by setting CPSR[4:0] to 10011 and CSPR[7] to 1
    - 4. Set the PC to 0x08, and begin executing the instruction there.

- Assembler format
  - SWI{<cond>} <24-bit immediate>

# Software Interrupt (II)

- **Examples**
  - To output the character 'A'

```
        MOV     r0, #'A'            ; Get 'A' in r0
        SWI     SWI_WrtieC          ; and print it
```

  - A subroutine to output a text string following the call

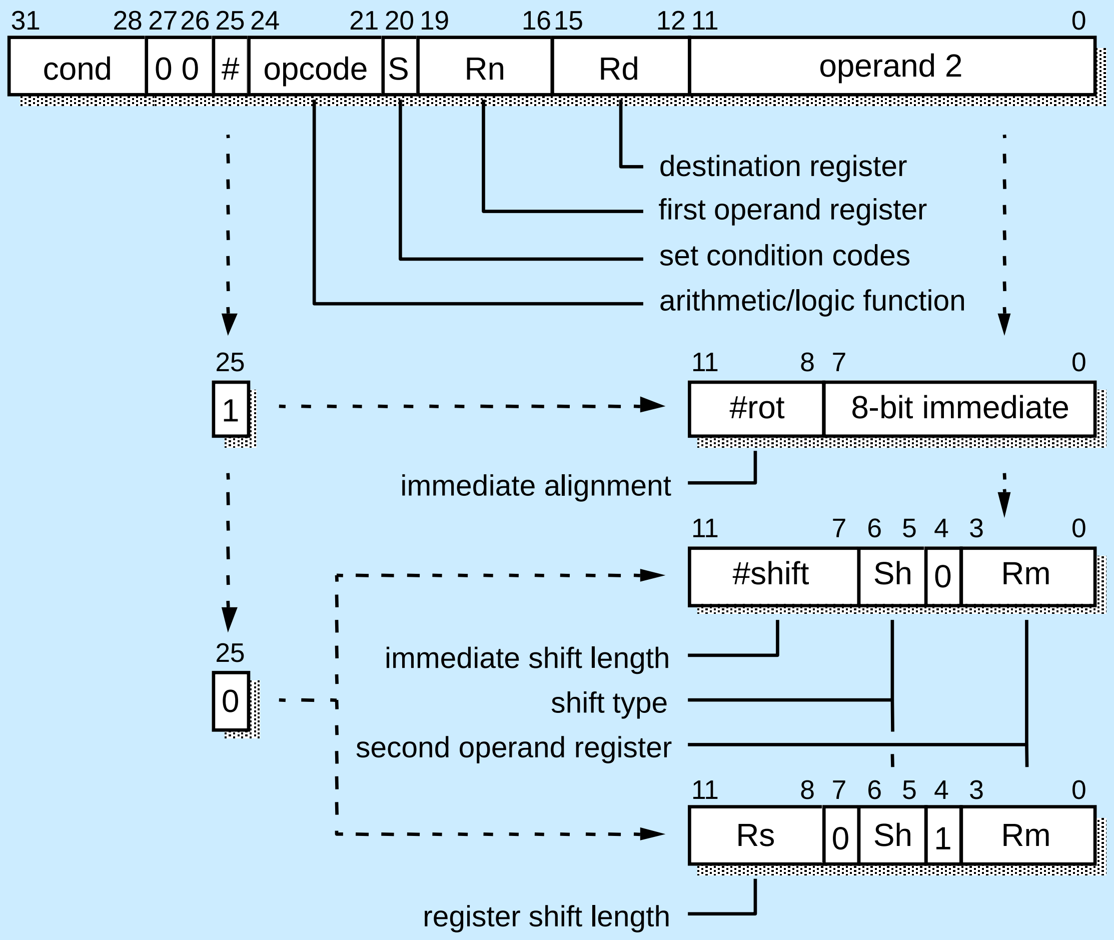
```
        BL      STROUT              ; Output follows message
        =       "Hello World", &0a, &0d, 0
        ...                         ; Return to here
        ...
STROUT  LDRB    r0, [r14], #1       ; Get character
        CMP     r0, #0              ; Check for end marker
        SWINE   SWI_WriteC          ; If not end, print
        BNE     STROUT              ; ... and loop
        ADD     r14, #3             ; align to next word
        BIC     r14, #3
        MOV     pc, r14             ; Return
```

  - Finish user program and return to monitor program

```
        SWI     SWI_Exit            ; Return to monitor.
```

# 10. Data Processing Instructions

- **Binary encoding**



## Data Processing Instructions (II)

- **ARM data processing instructions**

| **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 (III)

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

- **Examples**
  - To add r1 to r3 and place the result in r5
    - `ADD    r5, r1, r3`
  - To decrement r2 and check for zero
    - `SUBS   r2, r2, #1 ; Dec r2 and set cc`
    - `BEQ    LABEL             ; Branch if r2 zero`
    - `...                      ; Else fall through`
  - To multiply r0 by 5
    - `ADD    r0, r0, r0, LSL #2`
  - To add 64-bit integer in r0, r1 to one in r2, r3
    - `ADDS   r2, r2, r0        ; Add lower, save carry`
    - `ADC    r3, r3, r1        ; Add higher with carry.`

# 10B. Multiply Instructions

- Binary encoding

| 31 28 | 27 24 | 23 21 | 20 | 19 16 | 15 12 | 11 8 | 7 4 | 3 0 |
|---|---|---|---|---|---|---|---|---|
| cond | 0 0 0 0 | mul | S | Rd/RdHi | Rn/RdLo | Rs | 1 0 0 1 | Rm |

- Description

| **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 | RdHi:RdLo := Rm * Rs |
| 101 | UMLAL | Unsigned multiply-accumulate long | RdHi:RdLo += Rm * Rs |
| 110 | SMULL | Signed multiply long | RdHi:RdLo := Rm * Rs |
| 111 | SMLAL | Signed multiply-accumulate long | RdHi:RdLo += Rm * Rs |

  - S bit: setting of condition codes
  - 64-bit multiplies available on ARM7 (ARM7TDMI, ARM7TM, etc)

# Multiply Instructions (II)

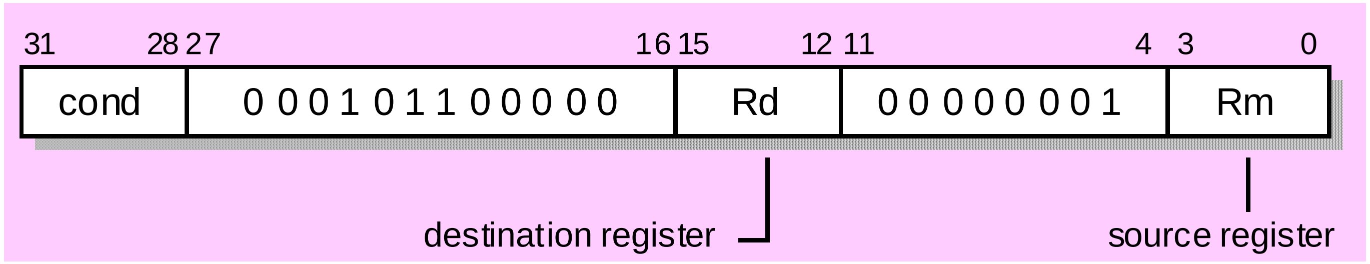
- Assembler formats
  - 32-bit product
    - MUL{<cond>}{S} Rd, Rm, Rs
    - MLA{<cond>}{S} Rd, Rm, Rs, Rn
  - 64-bit product
    - <mul>{<cond>}{S} RdHi, RdLo, Rm, Rs

- Examples
  - Scalar product of two vectors

```
      MOV   r11, #20        ; Initialize loop counter
      MOV   r10, #0         ; Initialize total
LOOP  LDR   r0, [r8], #4    ; Get first component
      LDR   r1, [r9], #4    ; .. And second
      MLA   r10,r0, r1, r10 ; Accumulate product
      SUBS  r11, r11, #1    ; Decrement loop counter
      BNE   LOOP
```

# 10C. Count Leading Zeros

- Binary encoding



- Description
  - Set Rd to the number of the bit position of the most significant 1 in Rm.
- Assembler format
  - CLZ{<cond>} Rd, Rm
- Example

```
MOV     r0, #&100
CLZ     r1, r0              ; r1 := 8
```

- Notes
  - Useful for renormalizing.
  - Architecture v5T only.

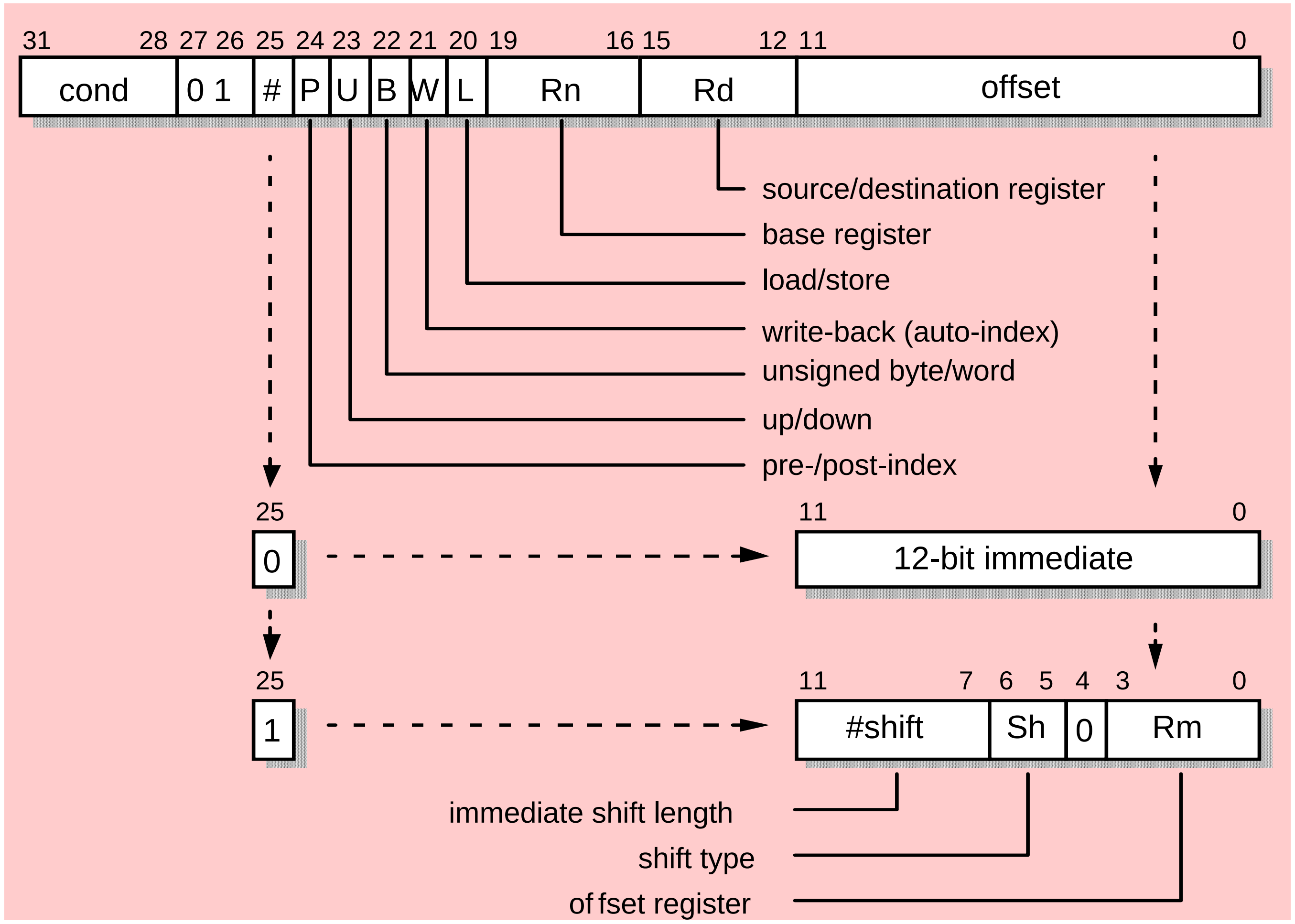
# 11. Special Data Transfer Instructions

## 11A. Single Word and Unsigned Data Byte Data Transfer Instructions

- Description
  - Construct an address starting from a base register (Rn)
  - Add (U=1) or subtract (U=0) an unsigned immediate or (possibly scaled) register offset
  - Computed address is used to load (L=1) or store (L=0)
    an unsigned byte (B=1) or word (B=0) quantity
    to or from a register (Rd), from or to memory.
  - When a byte is loaded into a register, it is zero extended to 32 bits.
  - When a byte is stored into memory, the bottom eight bits of the register are stored.
  - A pre-indexed (P=1) addressing mode uses the computed address for the load or store operation.
  - When write-back is requested (W=1), updates the base register to the computed value.
  - A post-indexed (P=0) addressing mode.

# Single Word and Unsigned Data Byte Data Transfer Instructions (II)

- Binary encoding



# Single Word and Unsigned Data Byte Data Transfer Instructions (III)

- Assembler format
  - Pre-indexed form
    - LDR|STR{<cond>}{B} Rd, [Rn, <offset>]{!}
    - B: unsigned byte transfer. B=0: word
    - Offset: +/-12-bit immediate or +/-Rm {, shift}
    - !: Write back (auto-indexing) in the pre-indexed form
  - Post-indexed form
    - LDR|STR{<cond>}{B}{T} Rd, [Rn], <offset>
    - T: selects the user view of the memory translation and protection system. Facility for operating system experts.
  - PC-relative form
    - LDR|STR{<cond>}{B} Rd, LABEL

## Single Word and Unsigned Data Byte Data Transfer Instructions (IV)

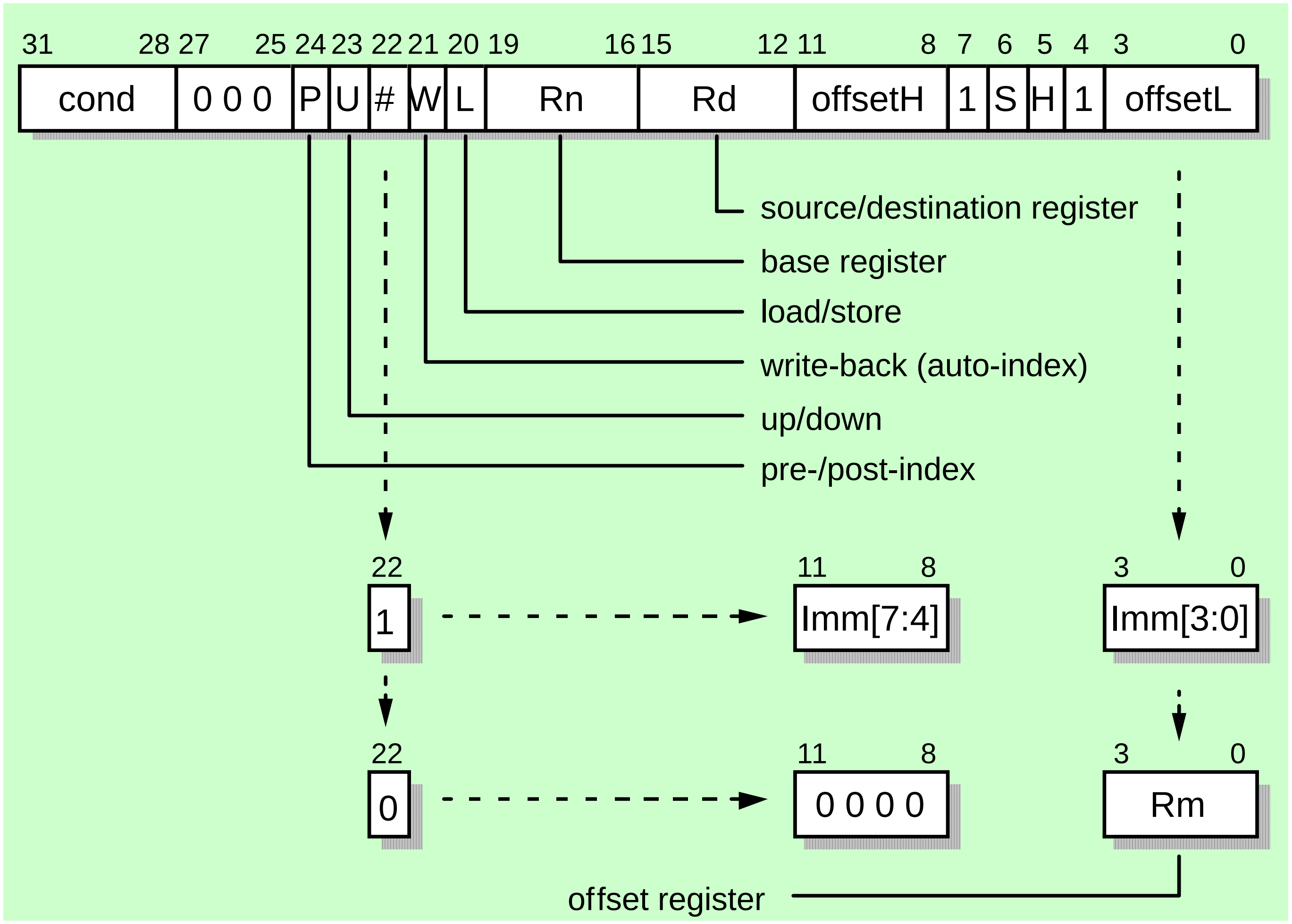
- Examples
  - To store a byte in r0 to a peripheral

```
            LDR      r1, UARTADD       ; UART address into r1
            STRB     r0, [r1]          ; Store data to UART
            ...
UARTADD &            &100000           ; Address literal
```

- Notes
  - Using the PC as the base address delivers the address of the instruction plus 8 bytes. It should not be used as the offset register, nor with any auto-indexing address mode.
  - Loading a word into the PC causes a branch. Loading a byte into a PC should be avoided.

# 11B. Half-Word and Signed Byte Data Transfer Instructions

- Binary encoding



# Half-Word and Signed Byte Data Transfer Instructions (II)

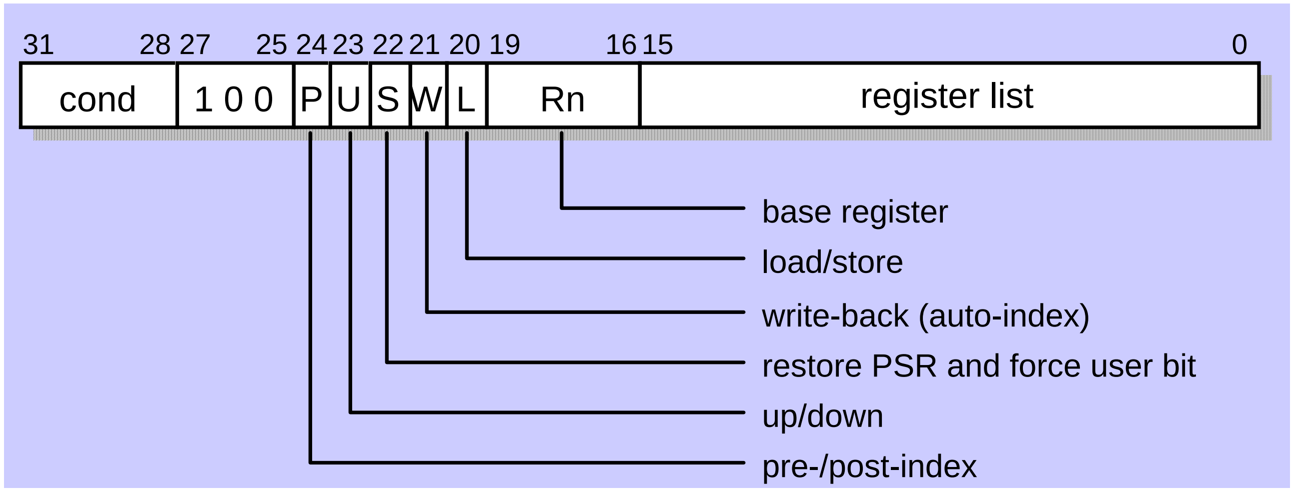
- Description
  - Very similar to the word and unsigned byte forms
  - Immediate offset is limited to 8 bits
  - Scaled register offset is no longer available

| S | H | Data type |
|---|---|---|
| 1 | 0 | Signedbyte |
| 0 | 1 | Unsignedhalf-word |
| 1 | 1 | Signedhalf-word |

- Assembler formats
  - Pre-indexed form
    - LDR|STR{<cond>}H₩SH|SB Rd, [Rn, <offset>]{!}
  - Post-indexed form
    - LDR|STR{<cond>}H|SH|SB Rd, [Rn], <offset>
    - Offset is +/-8-bit immediate or +/-Rm.

# 11C. Multiple Register Transfer Instructions

- Binary encoding



- Description
  - Register list: bit 0 is r0, bit 15 is PC.
  - The base address will be incremented (U=1) or decremented (U=0) before (P=1) or after (P=0) each word transfer.
  - Auto-indexing is supported (W=1): The base register will be increased (U=1) or decreased (U=0).
  - Used on procedure entry and return to save and restore work-space registers.

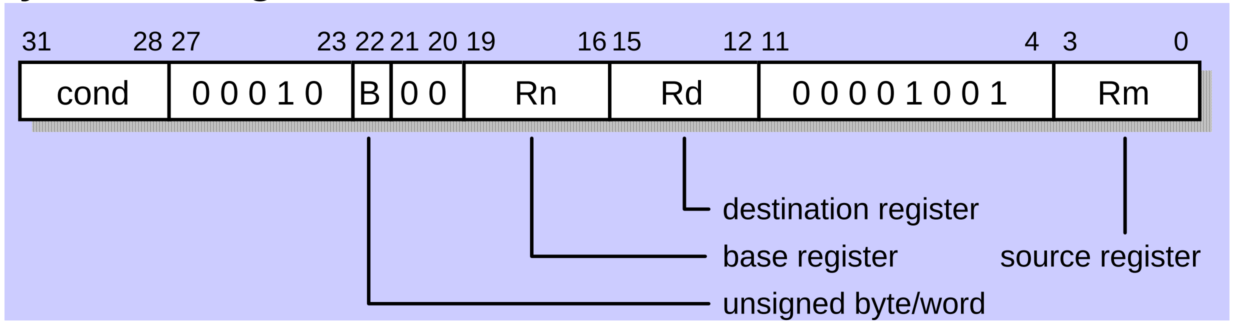
# Multiple Register Transfer Instructions (II)

- Assembler format
    - `LDM|STM{<cond>}<add mode> Rn{!}, <registers>`
  - In a non-user mode, the CPSR may be restored by
    - `LDM{<cond>}<add mode> Rn{!}, <registers + PC>^`
  - In a non-user mode, user registers may be saved/restored by
    - `LDM|STM{<cond>}<add mode> Rn, <registers - PC>^`

- Examples
  - To save three work registers and the return address upon entering a subroutine (r13=SP)
    - `STMFD   r13!, {r0-r2, r14}`
  - To restore the work registers and return
    - `LDMFD   r13!, {r0-r2, pc}.`

# 11D. Swap Memory and Register Instructions (SWP)

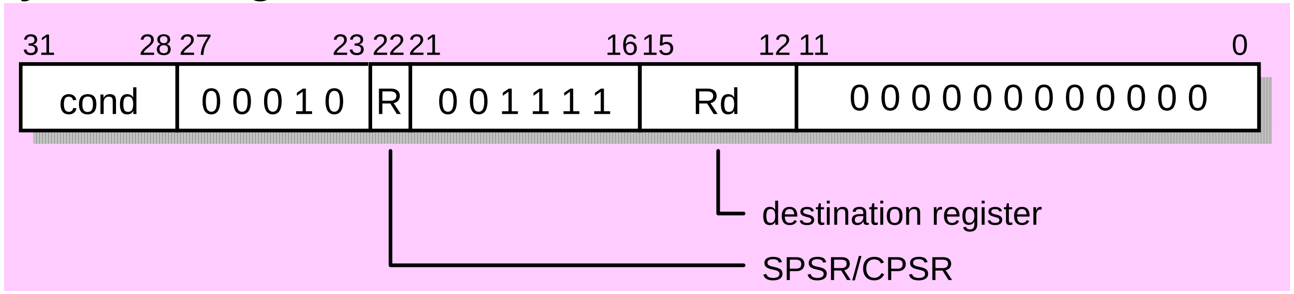
- Binary encoding



- Description
  - Loads the word (B=0) or unsigned byte (B=1) at the memory address by Rn into Rd, and stores the same data type from Rm into the same memory location.
  - Can be used as a basis of a semaphore mechanism to give mutually exclusive access to data structures that are shared between multiple processes, processors, or a processor and a DMA controller.
- Assembler format
  - SWP{<cond>}{B} Rd, Rm, [Rn].

## 11E. Status Register to General Register Transfer Instructions

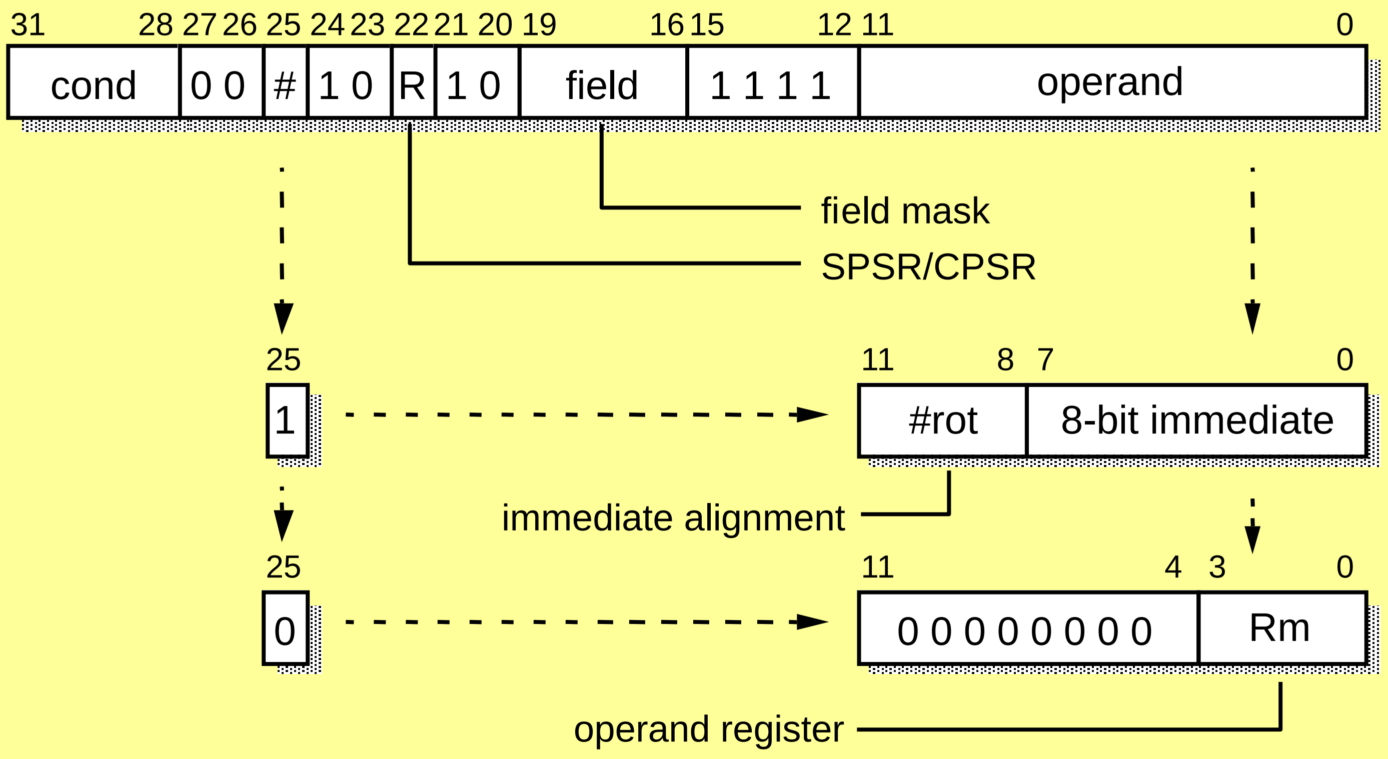
- Binary encoding



- Description
  - The CSPR (R=0) or the current SPSR (R=1) is copied into the destination register Rd.
- Assembler format
  - MRS{<cond>} Rd, CPSR|SPSR
- Examples
  - MRS r0, CPSR ; move the CPSR to r0
  - MRS r3, SPSR ; move the SPSR to r3
    - SPSR form should not be used in user mode.

# 11F. General Register to Status Register Transfer Instructions

- Binary encoding



## General Register to Status Register Transfer Instructions (II)

- Description
  - The operand (register Rm or a rotated 8-bit immediate) is moved under a field mask to the CPSR (R=0) or current mode SPSR (R=1).
  - The field mask controls the update of the four byte fields within the PSR register.
- Assembler format
  - MSR{<cond>} CPSR_f|SPSR_f, #,32-bit immediate>
  - MSR{<cond>} CPSR_<field>|SPSR_<field>, Rm
  - <field>:
    - 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 flags field – PSR[31:24]

## General Register to Status Register Transfer Instructions (III)

- Examples
  - To set the N, Z, C, and V flags:

    -
      ```
      MSR     CPSR_f, #&f0000000          ; Set all the flags
      ```
  - To set just the C flag, preserving N, Z, and V:

    -
      ```
      MRS     r0, CPSR                    ; Move the CPSR to r0
      ORR     r0, r0, #&20000000          ; Set bit 29 of r0
      MSR     CSPR_f, r0                  ; Move back to CPSR
      ```
  - To switch from supervisor mode into IRQ mode

    -
      ```
      MRS     r0, CPSR                    ; Move the CPSR to r0
      BIC     r0, r0, #$1f                ; Clear the 5 bottom bits
      ORR     r0, r0, #&12                ; Set the bits to IRQ mode
      MSR     CPSR_c, r0                  ; Save back to CPSR
      ```

# 12. Coprocessor Instructions

- Coprocessors
  - A general mechanism for extending the instruction set
    - System coprocessor – cache, memory management on ARM720
    - Floating-point ARM coprocessor
    - Application-specific coprocessor
- Coprocessor registers
  - Own private register sets
- Instruction formats
  - Coprocessor data operations
    - Completely internal to the coprocessor and cause a state change in the coprocessor registers
  - Coprocessor data transfers
    - Load or store the values in coprocessor registers from or to memory.
  - Coprocessor register transfers
    - Move values between ARM and coprocessor registers.

## 12B. Coprocessor Data Operations

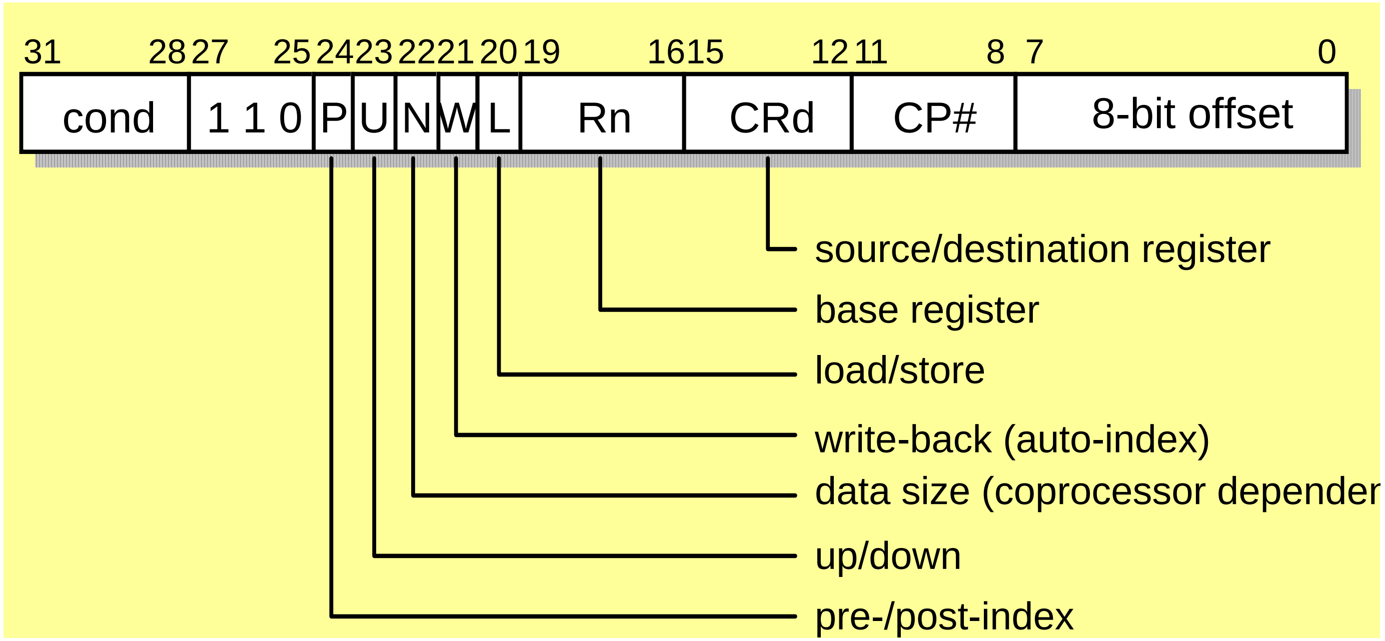
- Binary encoding

| 31 28 | 27 24 | 23 20 | 19 16 | 15 12 | 11 8 | 7 5 | 4 | 3 0 |
|---|---|---|---|---|---|---|---|---|
| cond | 1 1 1 0 | Cop1 | CRn | CRd | CP# | Cop2 | 0 | CRm |

- Description
  - Coprocessor CP# accept the instruction and perform the operation defined by the Cop1 and Cop2 fields, using CRn and CRm as source operands and packing the result in CR4
- Assembler format
  - CDP{<cond>} <CP#>, <Cop1>, CRd, CRn, CRm {, <Cop2>}
- Examples
  - CDP p2, 3, C0, C1, C2
  - CDPEQ p3, 6, C1, C5, C7, 4
- Notes
  - The interpretation of the Cop1, CRn, CRm, CRd, and Cop2 fields is coprocessor dependent.
  - The above interpretation is recommended.

# 12C. Coprocessor Data Transfers

- Binary encoding



- Description
  - The coprocessor CP# will accept the instruction.
  - The offset limited to 8 bits (12 bits for load/store)

# Coprocessor Data Transfers (II)

- Assembler format
  - Pre-indexed:
    - LDC|STC{<cond>} {L} <CP#>, CRd, [Rn, <offset>}{!}
  - Post-indexed:
    - LDC|STC{<cond>} {L} <CP#>, CRd, [Rn], <offset>

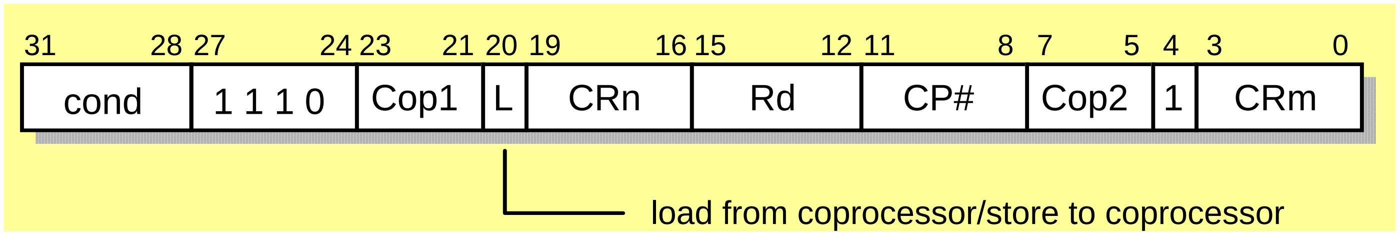
- Examples

```
LDC      p6, C0, [r1]
STCEQL   p5, C1, [r0], #4
```

- Notes
  - The interpretation of the N and CRd fields is coprocessor-dependent
  - The number of words transferred is controlled by the coprocessor.

# 12D. Coprocessor Register Transfers

- Binary encoding



- Description
  - Load: Perform an operation defined by Cop1 and Cop2 on source operands CRn and CRm and return a 32-bit integer result to the ARM which will replace it in Rd
- Assembler format
  - Move to ARM from cop
    - MRC{<cond>} <CP#>, <Cop1>, Rd, CRn, CRm{, <Cop2>}
  - Move to cop from ARM
    - MCR{<cond>} <CP#>, <Cop1>, Rd, CRn, CRm{, <Cop2>}
  - Examples
    - MCR p14, 3, r0, C1, C2
    - MCRCS p2, 4, r3, C3, C4, 6

# 13. Miscellaneous Instructions

## 13A. Breakpoint Instruction

- Binary encoding

| 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 |

- Description
  - Used for software debugging purposes (v5T only)
  - Causes the processor to take a prefetch abort when the debug hardware unit is configured appropriately.
- Assembler format
  - BRK
- Example
  - BRK ; !

# 13B. Unused Instruction Space

- 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 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 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 Instruction Space (II)

- 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 |

- Unused instruction 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 |

- Behavior of unused instructions
  - Undefined instruction trap if an attempt is made to execute an instruction in the undefined address space.