# ISA Assembler Format (4B)

## Data Transfer Instructions

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

This document was produced by using LibreOffice.

Young Won Lim
2/5/20

# Based on

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

# Data Transfer Instructions

## Single Word and Unsigned Byte

**The pre-indexed form of the instruction**

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

**The post-indexed form**

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

**A useful PC-relative form (assembler does all the work)**

LDR | STR {<cond>} Rd, LABEL

## Half Word and Signed Data

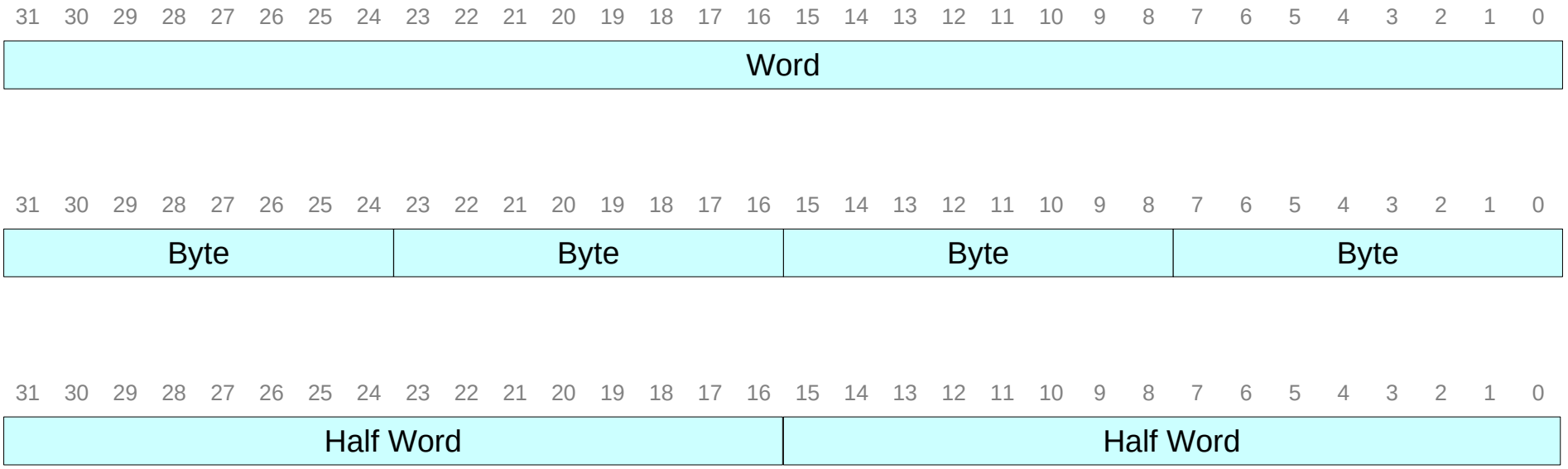
**The pre-indexed form of the instruction**

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

**The post-indexed form**

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

# Word, Byte, Half Word



# Data Transfer Mnemonics

little endian case

| | | | |
|---|---|---|---|
| Unsigned | Byte | Transfer | **B** |
| Unsigned | Half Word | Transfer | **H** |
| Signed | Byte | Transfer | **SB** |
| Signed | Half Word | Transfer | **SH** |

# Data Transfer Instructions

| | | |
|---|---|---|
| **LDR** | **STR** | Single Word Transfers |
| **LDR B** | **STR B** | Unsigned Byte Transfers |
| **LDR H** | **STR H** | Unsigned Half Word Transfers |
| **LDR SB** | ~~**STR SB**~~ | Signed Byte Transfer |
| **LDR SH** | ~~**STR SH**~~ | Signed Half Word Transfer |

# Data Transfer Instructions

LDR | STR {<cond>} {**B**} Rd, [Rn, <**offset1**>] {!}
LDR | STR {<cond>} {**B**} {**T**} Rd, [Rn], <**offset1**>
LDR | STR {<cond>} Rd, LABEL

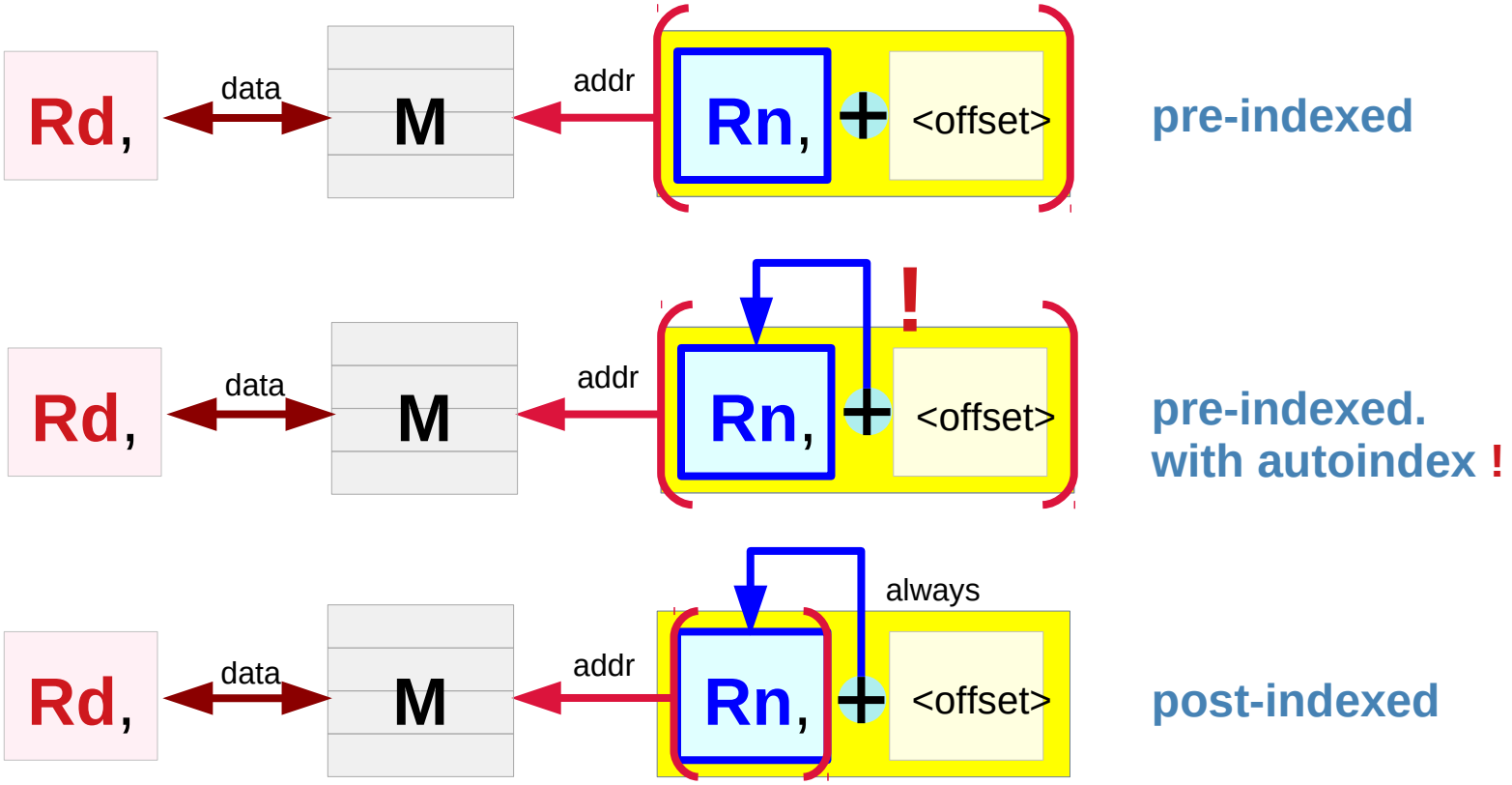
<**offset1**> = 1. #+/–<**12**-bit immediate>
= 2. +/–Rm {, <**shift**>}

LDR | STR {<cond>} **H** | **SH** | **SB** Rd, [Rn, <**offset2**>] {!}
LDR | STR {<cond>} **H** | **SH** | **SB** Rd, [Rn], <**offset2**>

<**offset2**> = 1. #+/–<**8**-bit immediate>
= 2. +/–Rm …. no shifted operand

# Pre-indexing and Post-indexing

| | |
|---|---|
| Rd, [Rn, <offset>] | pre-indexed |
| Rd, [Rn, <offset>] ! | pre-indexed with autoindex |
| Rd, [Rn], <offset> | post-indexed |



# Data Transfer Instructions – single word and unsigned byte

## Single Word and Unsigned Byte

| | |
|---|---|
| LDR \| STR {<cond>} {**B**} Rd, [Rn, <**offset**>] {!} | pre-indexed |
| LDR \| STR {<cond>} {**B**} {**T**} Rd, [Rn], <**offset**> | post-indexed |
| LDR \| STR {<cond>} Rd, LABEL | PC-relative |

**B** selects unsigned byte transfer, the default is **word**

<**offset**>= 1. #+/–**<12-bit immediate>**
= 2. +/–**Rm {, <shift>}**

**!** selects write-back (auto-indexing) in the pre-indexed form

**T** selects the user view of the memory translation and protection system
should only be used in non-user mode

<shift> general shift operation but you cannot specify
the shift amount by a register.

# Data Transfer Instructions – single word and unsigned byte

**The pre-indexed form of the instruction**

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

| | |
|---|---|
| LDR \| STR | Rd, [Rn, +-Rm] {!} |
| LDR \| STR | Rd, [Rn, +-Rm, <sh>, amount] {!} |
| LDR \| STR | Rd, [Rn, #+-<12-bit immediate>] {!} |
| LDR \| STR B | Rd, [Rn, +-Rm]] {!} |
| LDR \| STR B | Rd, [Rn, +-Rm, <sh>, amount] {!} |
| LDR \| STR B | Rd, [Rn, #+-<12-bit immediate>] {!} |
| LDR \| STR <cond> | Rd, [Rn, +-Rm] {!} |
| LDR \| STR <cond> | Rd, [Rn, +-Rm, <sh>, amount] {!} |
| LDR \| STR <cond> | Rd, [Rn, #+-<12-bit immediate>] {!} |
| LDR \| STR <cond> B | Rd, [Rn, +-Rm]] {!} |
| LDR \| STR <cond> B | Rd, [Rn, +-Rm, <sh>, amount] {!} |
| LDR \| STR <cond> B | Rd, [Rn, #+-<12-bit immediate>] {!} |

# Data Transfer Instructions – single word and unsigned byte

**The post-indexed form**

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

| | |
|---|---|
| LDR \| STR | Rd, [Rn], +-Rm |
| LDR \| STR | Rd, [Rn], +-Rm, <sh>, amount |
| LDR \| STR | Rd, [Rn], #+-<12-bit immediate> |
| LDR \| STR {B}{T} | Rd, [Rn], +-Rm |
| LDR \| STR {B}{T} | Rd, [Rn], +-Rm, <sh>, amount |
| LDR \| STR {B}{T} | Rd, [Rn], #+-<12-bit immediate> |
| LDR \| STR <cond> | Rd, [Rn], +-Rm |
| LDR \| STR <cond> | Rd, [Rn], +-Rm, <sh>, amount |
| LDR \| STR <cond> | Rd, [Rn], #+-<12-bit immediate> |
| LDR \| STR <cond> {B}{T} | Rd, [Rn], +-Rm |
| LDR \| STR <cond> {B}{T} | Rd, [Rn], +-Rm, <sh>, amount |
| LDR \| STR <cond> {B}{T} | Rd, [Rn], #+-<12-bit immediate> |

# Data Transfer Instructions – single word and unsigned byte

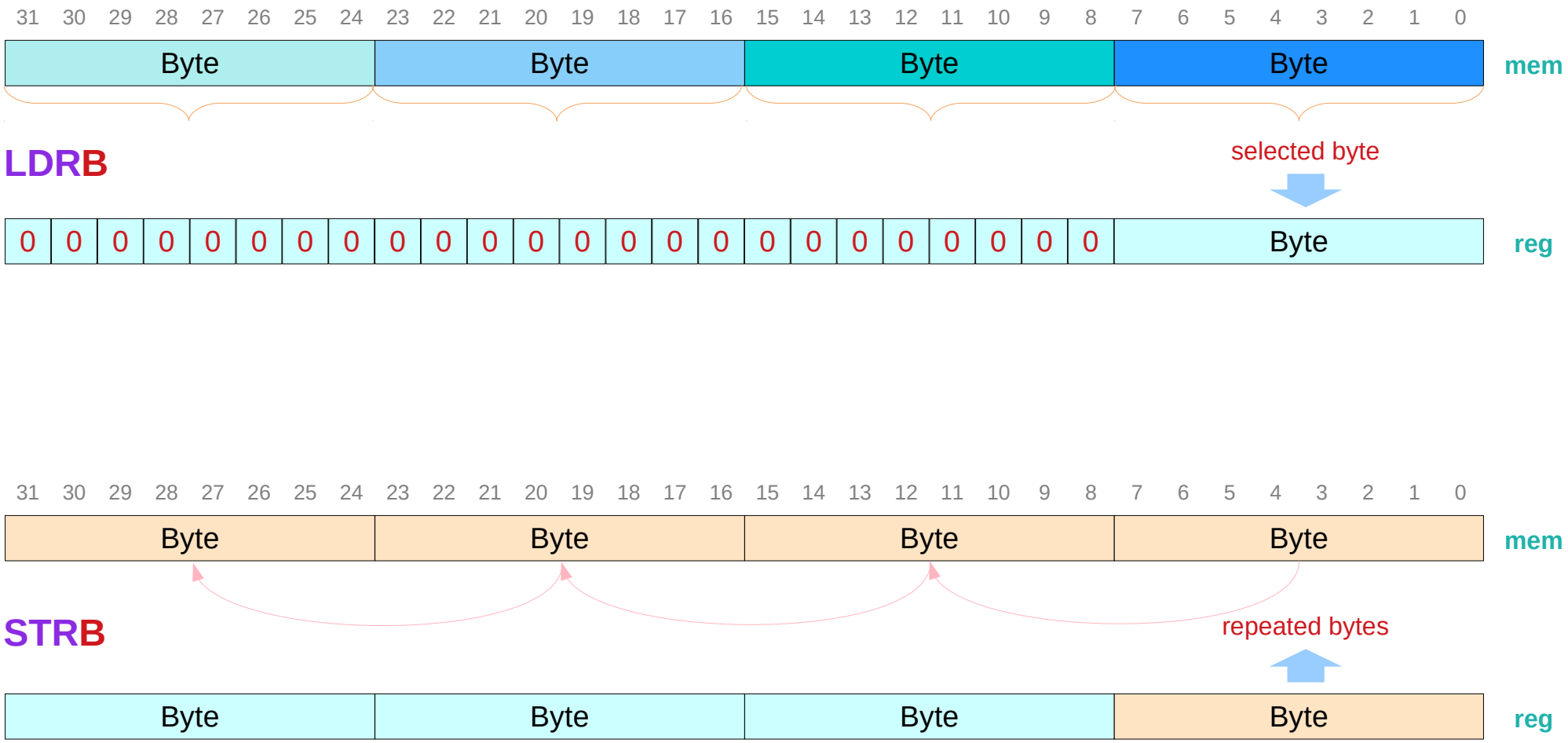
**A useful PC-relative form (assembler does all the work)**

LDR | STR {<cond>} Rd, LABEL

LDR | STR Rd, LABEL
LDR | STR <cond> Rd, LABEL

# Unsigned Byte Transfer B

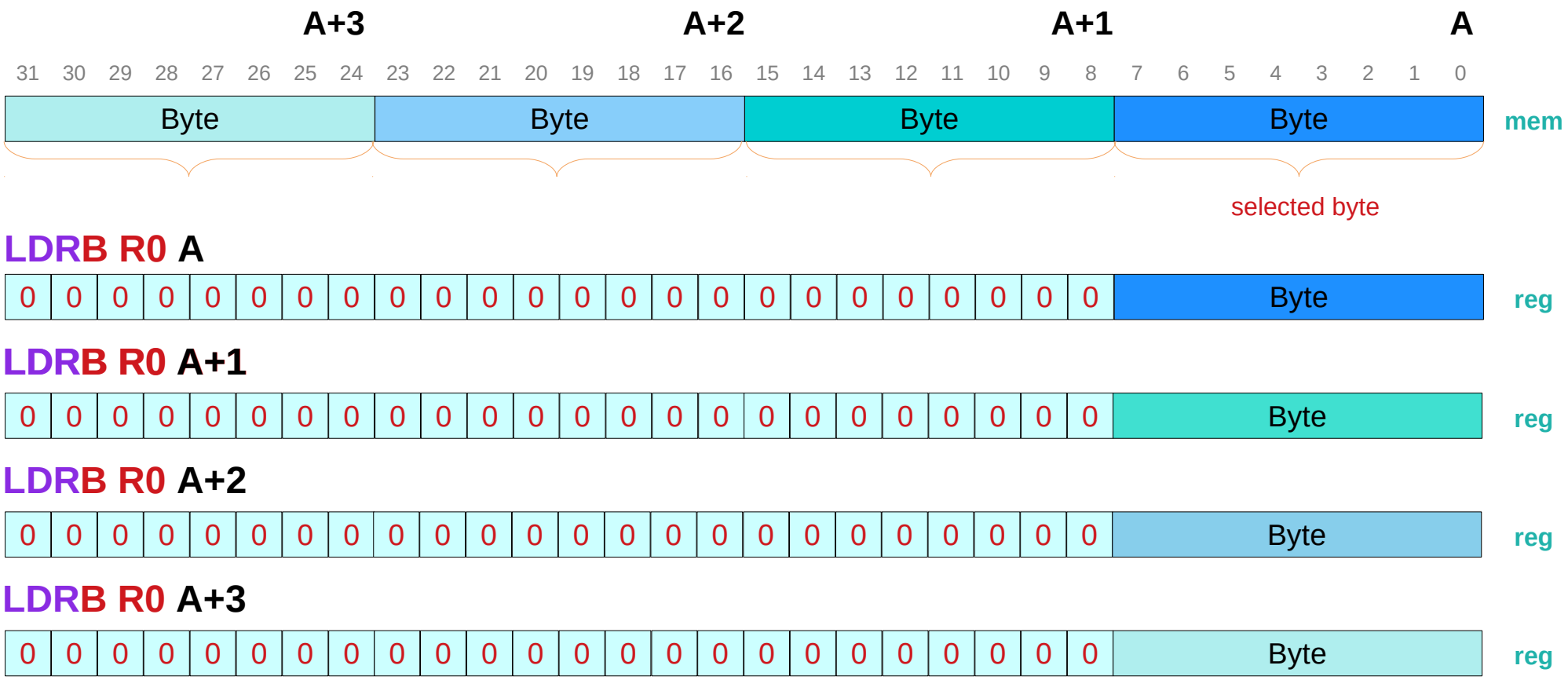
little endian case



# Unsigned Byte Load

**B**

**little endian case**



# T suffix

**The post-indexed form**

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

| | | | |
|---|---|---|---|
| LDR T | Rd, [Rn], <offset> | STR T | Rd, [Rn], <offset> |
| LDR BT | Rd, [Rn], <offset> | STR BT | Rd, [Rn], <offset> |

if T is present the **W** bit will be **set** in a **post-indexed** instruction,
forcing **non-privileged (user) mode** for the transfer cycle.

T is not allowed when a **pre-indexed** addressing mode
is specified or implied.

T selects the user view of the memory translation and protection system
should only be used in non-user mode

# T suffix applications

accesses memory as if in the **user-mode**,
applies the **permission check** based on the code being "user".

useful in a **kernel**

where a user-space process passes a pointer to the kernel,
and you want to ensure that the **user process**, not the kernel,
had right **permissions** to read the data.

used in a privileged code, such as an **exception handler**,

to test whether an **access** is possible in thread mode.

For example, if a **user mode access** were **aborted**,
the **exception handler** may try to correct the problem
by changing the memory protection settings.
The LDRT could then be used to test
whether the access was now possible.

http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.faqs/ka14336.html

# Data Transfer Instructions – half-word and signed data

## Half Word and Signed Data

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

**H | SH | SB** selects the data type (H: half-word, S:signed)

<**offset**>  = 1. **#+/–<8-bit immediate>**
= 2. **+/–Rm**  …. **no shifted operand**

! selects write-back (auto-indexing) in the pre-indexed form

# Data Transfer Instructions – half-word and signed data

**The pre-indexed form of the instruction**

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

| | |
|---|---|
| LDR \| STR H | Rd, [Rn, #+-Rm] {!} |
| LDR \| STR H | Rd, [Rn, #+-<8-bit immediate>] {!} |
| LDR \| STR SH | Rd, [Rn, #+-Rm] {!} |
| LDR \| STR SH | Rd, [Rn, #+-<8-bit immediate>] {!} |
| LDR \| STR SB | Rd, [Rn, #+-Rm] {!} |
| LDR \| STR SB | Rd, [Rn, #+-<8-bit immediate>] {!} |
| LDR \| STR <cond> H | Rd, [Rn, #+-Rm] {!} |
| LDR \| STR <cond> H | Rd, [Rn, #+-<8-bit immediate>] {!} |
| LDR \| STR <cond> SH | Rd, [Rn, #+-Rm] {!} |
| LDR \| STR <cond> SH | Rd, [Rn, #+-<8-bit immediate>] {!} |
| LDR \| STR <cond> SB | Rd, [Rn, #+-Rm] {!} |
| LDR \| STR <cond> SB | Rd, [Rn, #+-<8-bit immediate>] {!} |

# Data Transfer Instructions – half-word and signed data

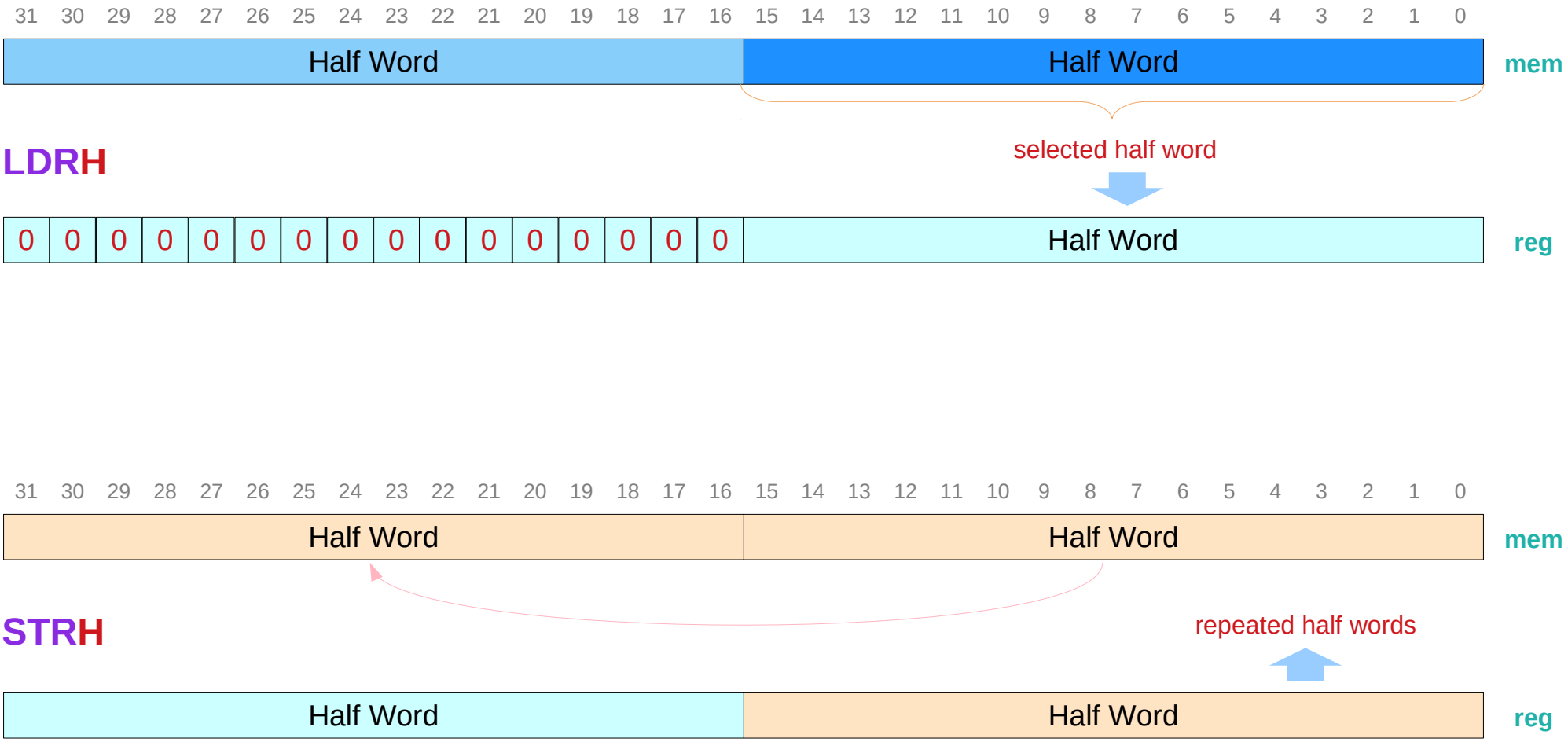
**The post-indexed form**

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

| | |
|---|---|
| LDR \| STR H | Rd, [Rn], #+-Rm |
| LDR \| STR H | Rd, [Rn], #+-<8-bit immediate> |
| LDR \| STR SH | Rd, [Rn], #+-Rm |
| LDR \| STR SH | Rd, [Rn], #+-<8-bit immediate> |
| LDR \| STR SB | Rd, [Rn], #+-Rm |
| LDR \| STR SB | Rd, [Rn], #+-<8-bit immediate> |
| LDR \| STR <cond> H | Rd, [Rn], #+-Rm |
| LDR \| STR <cond> H | Rd, [Rn], #+-<8-bit immediate> |
| LDR \| STR <cond> SH | Rd, [Rn], #+-Rm |
| LDR \| STR <cond> SH | Rd, [Rn], #+-<8-bit immediate> |
| LDR \| STR <cond> SB | Rd, [Rn], #+-Rm |
| LDR \| STR <cond> SB | Rd, [Rn], #+-<8-bit immediate> |

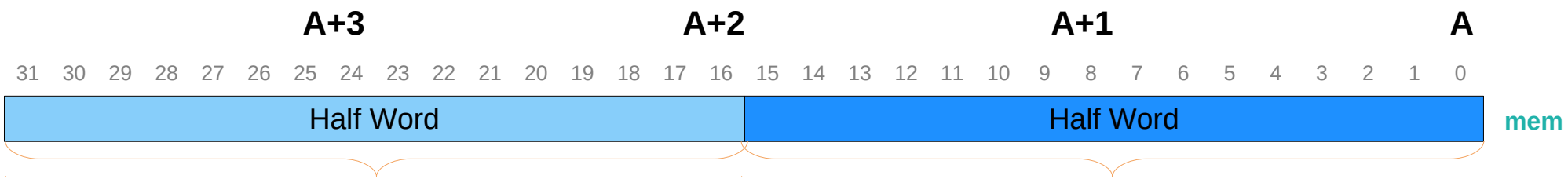
# Unsigned Half Word Transfer H

little endian case



# Unsigned Half Word LDR H

little endian case



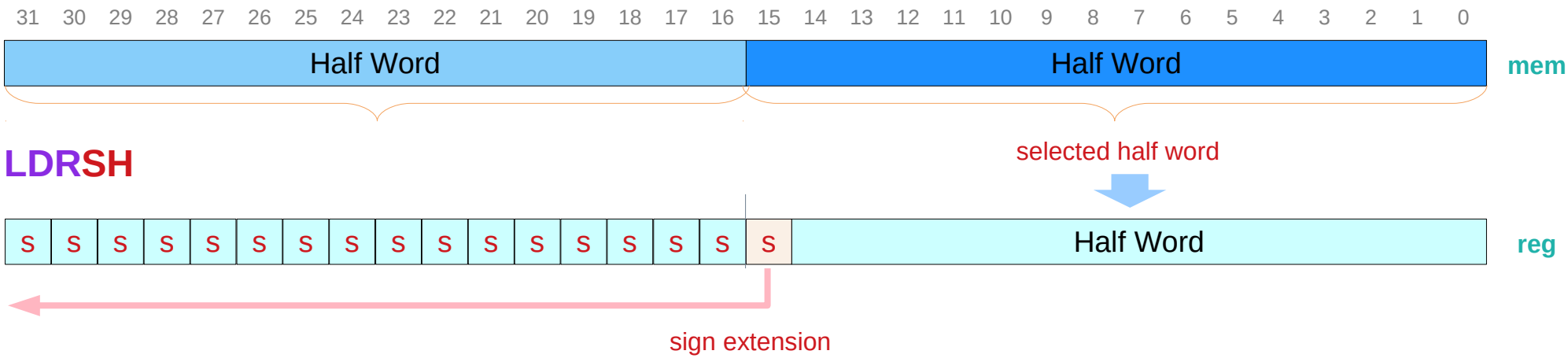

selected half word

**LDRSB R0 A**


**LDRSB R0 A+2**

# Signed Half Word Transfer SH

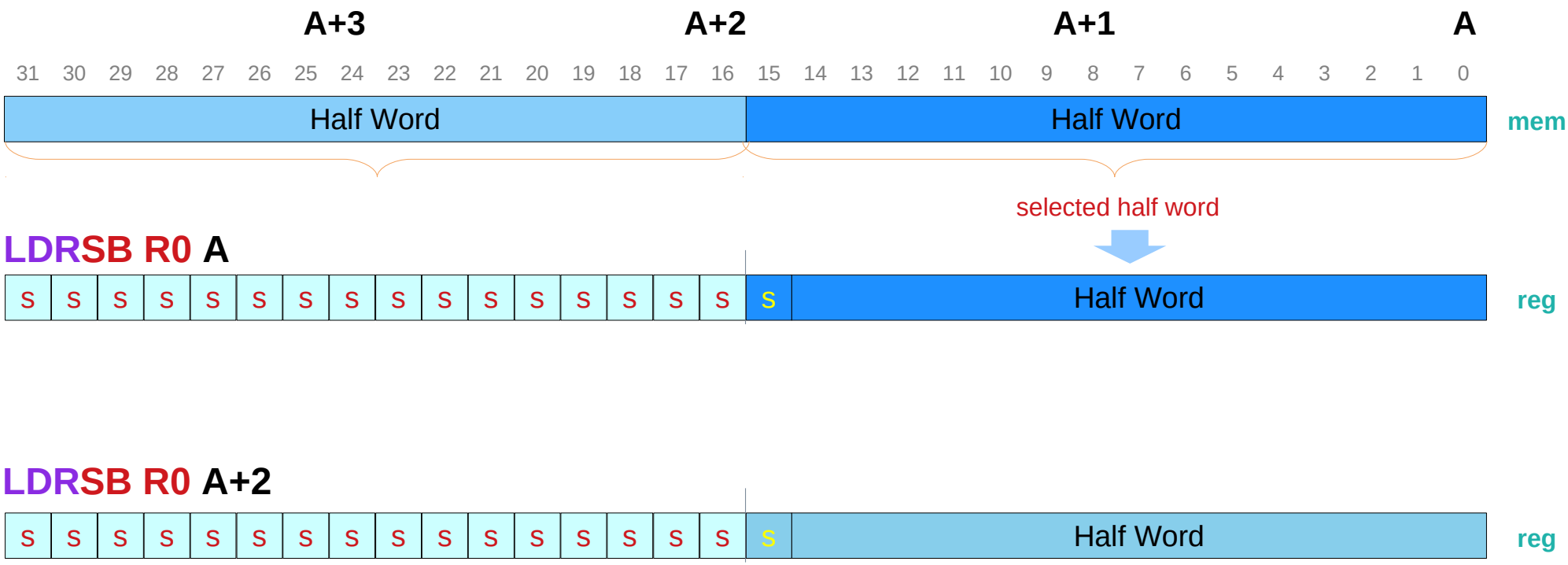
little endian case




~~STRSH~~

# Signed Half Word LDR SH

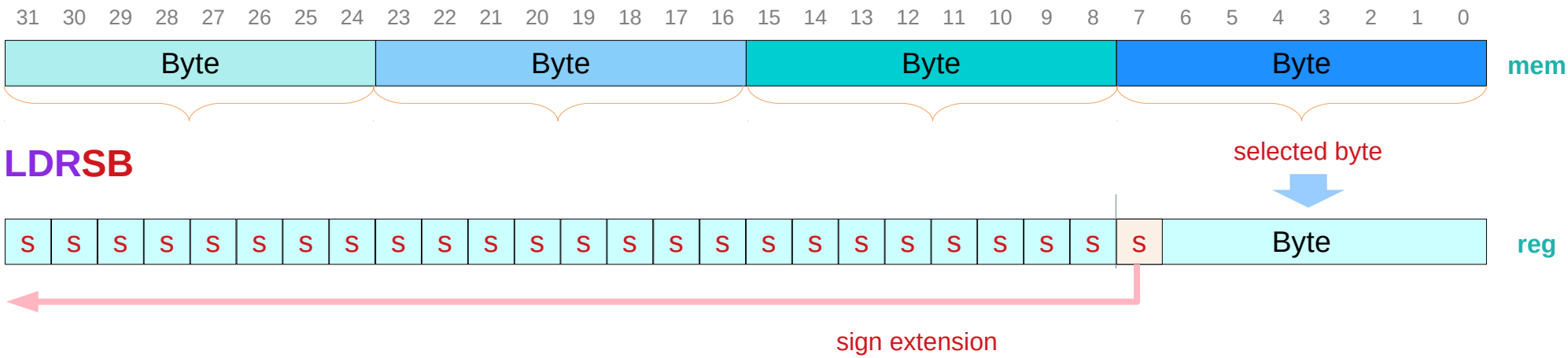
little endian case



# Signed Byte Transfer

**SB**

**little endian case**

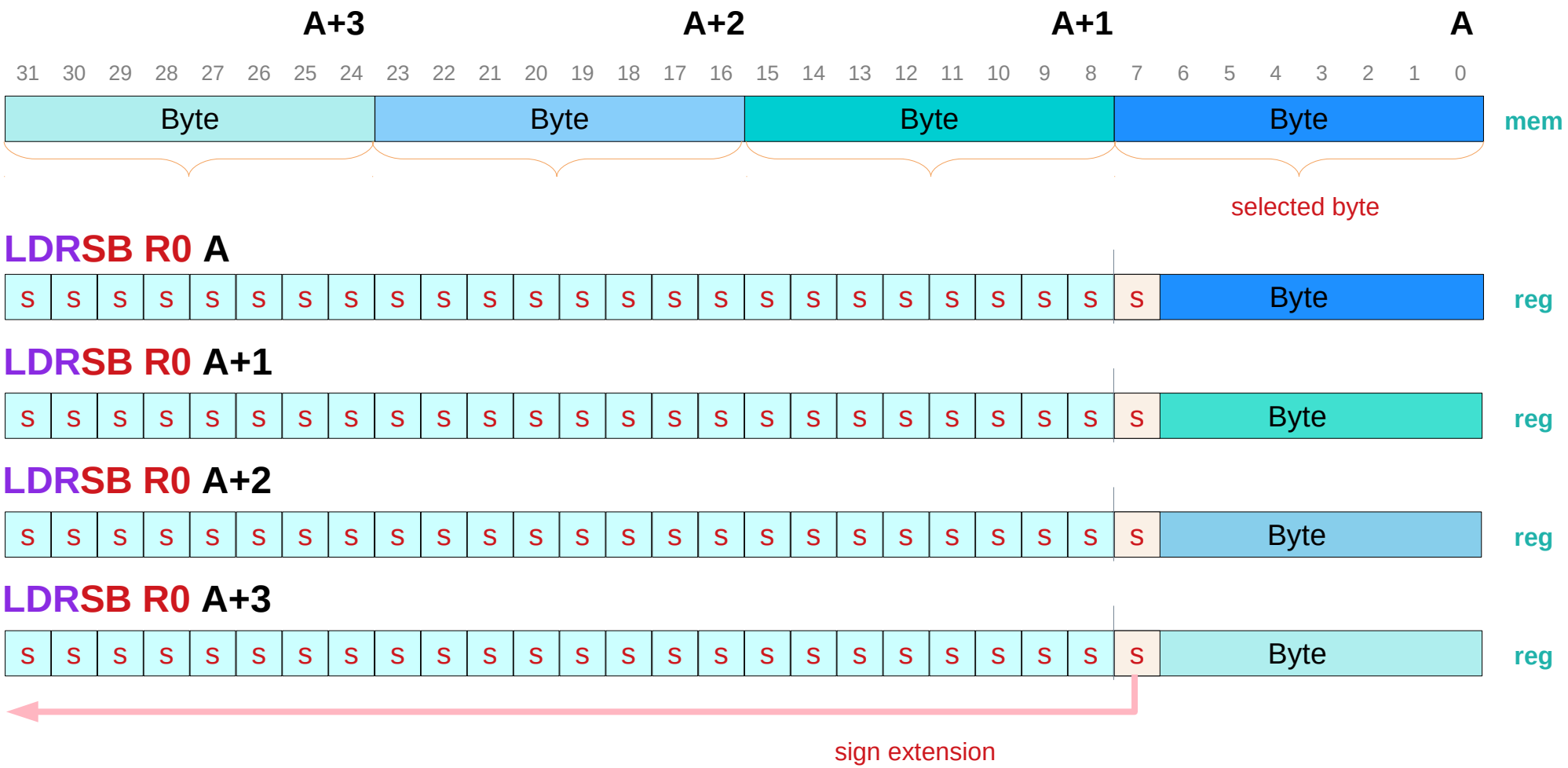



~~**STRSB**~~ (little endian case)

# Signed Byte LDR

**SB**

**little endian case**



# Data Transfer with Shifted Operand Instructions

## Single Word and Unsigned Byte

**The pre-indexed form of the instruction**

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

**The post-indexed form**

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

**A useful PC-relative form (assembler does all the work)**

LDR | STR {<cond>} Rd, LABEL ~~**shifted operand**~~

## Half Word and Signed Data

**The pre-indexed form of the instruction**

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

**The post-indexed form**

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

only **single** (word / unsigned byte) **data transfer** instructions support shifted offset

# Data Transfer Offset – <shift> operand

**Data Transfer Instructions with <shift>**

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

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

<offset>

#+/–<12-bit immediate>

**+/–Rm {, <shift>}**

**<shift type> # <#shift>**

<shift type> ~~Rs~~ … not allowed in single data transfers

| | |
|---|---|
| LSL, ASL, | Logical Shift Left, Arithmetic Shift Left |
| LSR, | Logical Shift Right |
| ASR, | Arithmetic Shift Right |
| ROR | Rotate Right |

<shift type>

| | |
|---|---|
| RRX | Rotate Right Extended |

# Unaligned Memory Access

Older ARM processors require data load and stores
to be to/from architecturally aligned addresses. This means:

| | | |
|---|---|---|
| **LDRB / STRB** | - address must be byte aligned | A, A+1, A+2, A+3 |
| **LDRH / STRH** | - address must be 2-byte aligned | A, A+2 |
| **LDR / STR** | - address must be 4-byte aligned | A |

**LDM** - address must be 4-byte aligned A

handling multiple word quantities

an unaligned load is one where
the address does not match the architectural alignment.

On older processors (ARM9 family)
an unaligned load
software synthesised
performing a series of small accesses,
combining the results.

http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.faqs/ka15414.html

# Unaligned Memory Access

The ARMv6 architecture
introduced the first hardware support for unaligned accesses.
ARM11 and Cortex-A/R processors
can deal with unaligned accesses in hardware,
removing the need for software routines.

Support for unaligned accesses is limited to a sub-set of load/store instructions:

**LDRB / LDRSB / STRB**
**LDRH / LDRSH / STRH**
**LDR / STR**

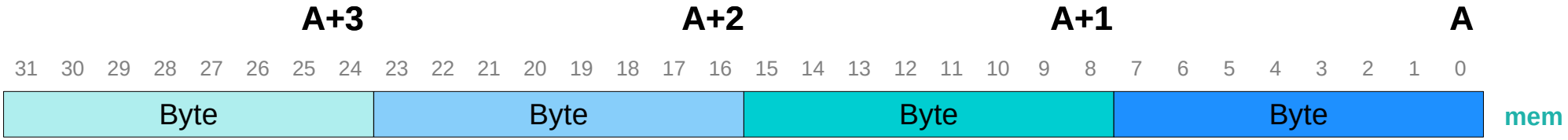
Instructions which do NOT support unaligned accesses include:

**LDM / STM**
**LDRD / STRD**

http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.faqs/ka15414.html

# Single Word LDR

little endian case




A **word load** (**LDR**) will normally use a word aligned address (**A**)

an address offset from a word boundary
will cause the data to be rotated into the register
so that the addressed byte occupies bits 0 to 7

**half-words** accessed at offsets 0 and 2 from the word boundary (**A**, **A+2**)
will be correctly loaded into bits 0 to 15 of the register.

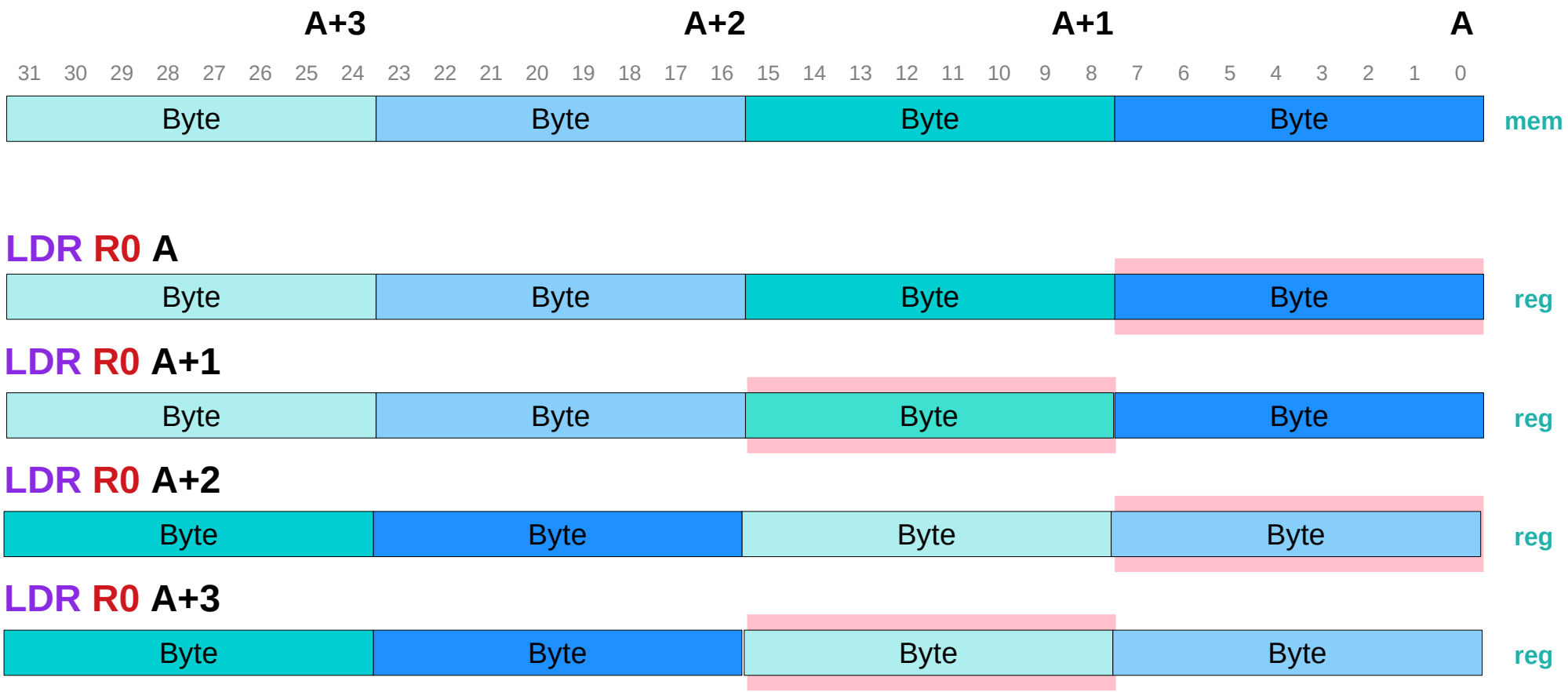
An address offset of 1 or 3 from a word boundary
will cause the data to be rotated into the register
so that the addressed byte occupies bits 15 to 8.

Two shift operations are then required to clear or to sign extend the upper 16 bits.

ARM7TDMI-S Data Sheet ARM DDI 0084D Final

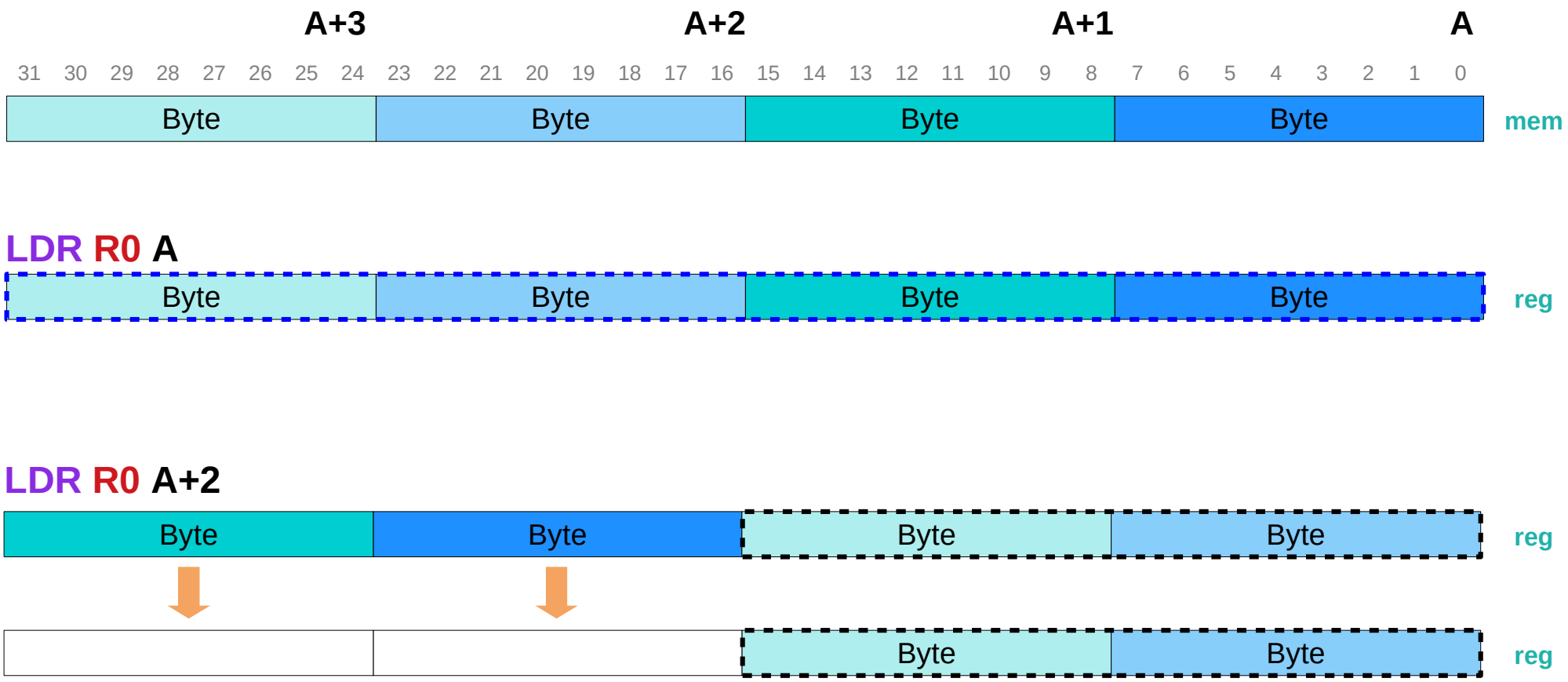
# Single Word LDR

little endian case



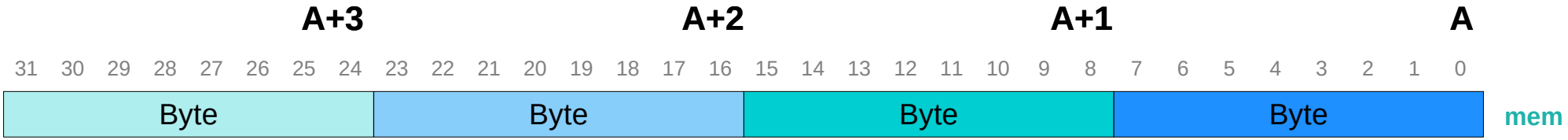
# Single Word LDR

little endian case



# Single Word LDR / STR

**little endian case**




**Loading** a **word** at a non-word-aligned address **A+1**, **A+2**, **A+3**
the loaded data is the word-aligned word
containing the addressed byte, but rotated so that
the addressed byte is in the **least significant byte**
of the destination register

**Storing** a **word** at a non-word-aligned address **A+1**, **A+2**, **A+3**
ignoring the **least significant two bits** of the address
the word is stored as though they had been **zero**

Some ARM systems may raise an **exception** under these circumstances
controlled by the **A** flag in **bit 1** of **CP15** register

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

# CP15 Register



The MMU is controlled with the System Control coprocessor registers.
from VMSAv6, several new registers, and register fields have been added:

- a TLB type register in register 0
- additional control bits to register 1
- a second translation table base register, and new control fields to register 2
- an additional fault status register to register 5
- an additional Fault Address register to register 6
- TLB invalidate by ASID support in register 8
- ASID control in register 13.

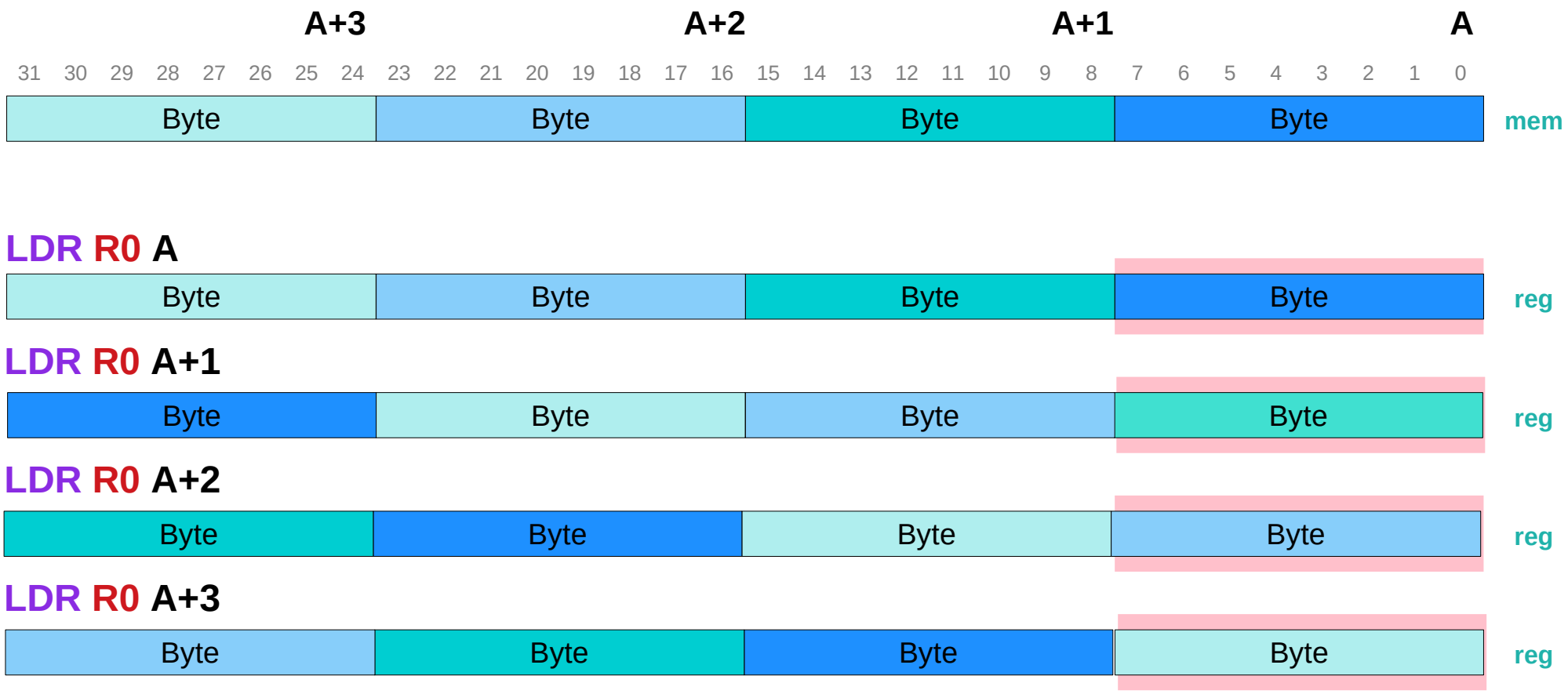
Domain support (register 3) and TLB lockdown support (register 10)
are the same as in earlier versions of the architecture.

All VMSA-related registers are accessed with instructions of the form:
**MRC p15, 0, Rd, CRn, CRm, opcode_2**
**MCR p15, 0, Rd, CRn, CRm, opcode_2**
Where CRn is the system control coprocessor register.
Unless specified otherwise, CRm and opcode_2 SBZ.

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

# Single Word LDR

little endian case



# Multiple data transfer instructions

**The normal form**

LDM | STM {<cond>} <add mode> Rn {!}, reglist

<add mode> specifies one of the addressing mode

| (I,D)x(B,A) | | | (F,E)x(A,D) | |
|---|---|---|---|---|
| **IB** | Inc Before | \| | **FA** | Full Ascending |
| **IA** | Inc After | \| | **FD** | Full Descending |
| **DB** | Dec Before | \| | **EA** | Empty Ascending |
| **DA** | Dec After | \| | **ED** | Empty Descending |

**Rn !** Updates the base register **Rn**

Increment / Decrement the base (stack top) register Rn

**reglist** examples **: {r0}, {r0, r1}, {r0, r1-r4}** **{ } necessary**

# Multiple data transfer instructions – the normal form

**The normal form**

LDM | STM {<cond>} <add mode> Rn {!}, reglist

<add mode> specifies one of the addressing mode

IB, IA, DB, DA, FA, FD, EA, ED (I,D)x(B,A) | (F,E)x(A,D)

| | | | | |
|---|---|---|---|---|
| LDM \| STM | | IB \| IA \| DB \| DA \| FA \| FD \| EA \| ED | Rn, | reglist |
| LDM \| STM | | IB \| IA \| DB \| DA \| FA \| FD \| EA \| ED | Rn!, | reglist |
| LDM \| STM | <cond> | IB \| IA \| DB \| DA \| FA \| FD \| EA \| ED | Rn, | reglist |
| LDM \| STM | <cond> | IB \| IA \| DB \| DA \| FA \| FD \| EA \| ED | Rn!, | reglist |

# Multiple data transfer instructions – non-user mode

**To restore the CPSR in a non-user mode**

LDM {<cond>} <add mode> Rn {!}, <reglist + **pc**>^

**To save / restore the use registers in a non-user mode**

LDM | STM {<cond>} <add mode> Rn, <reglist - **pc**>^

when **pc** is included in the <reglist>
… returning from an execution handler

SPSR → CPSR

when **pc** is not included in the <reglist>

load to <registers> … load to the user mode registers
store from <register> ... store from the user mode registers

# Multiple data transfer instructions

| 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 data transfer instruction mnemonics

| Pre | Inc | Load |
|---|---|---|
| Before | Inc | LDRIB |
| Empty | Descend | LDRED |

| Pre | Inc | Store |
|---|---|---|
| Before | Inc | STRIB |
| Full | Ascend | STRFA |

| Post | Inc | Load |
|---|---|---|
| After | Inc | LDRIA |
| Full | Descend | LDRFD |

| Post | Inc | Store |
|---|---|---|
| After | Inc | STRIA |
| Empty | Ascend | STREA |

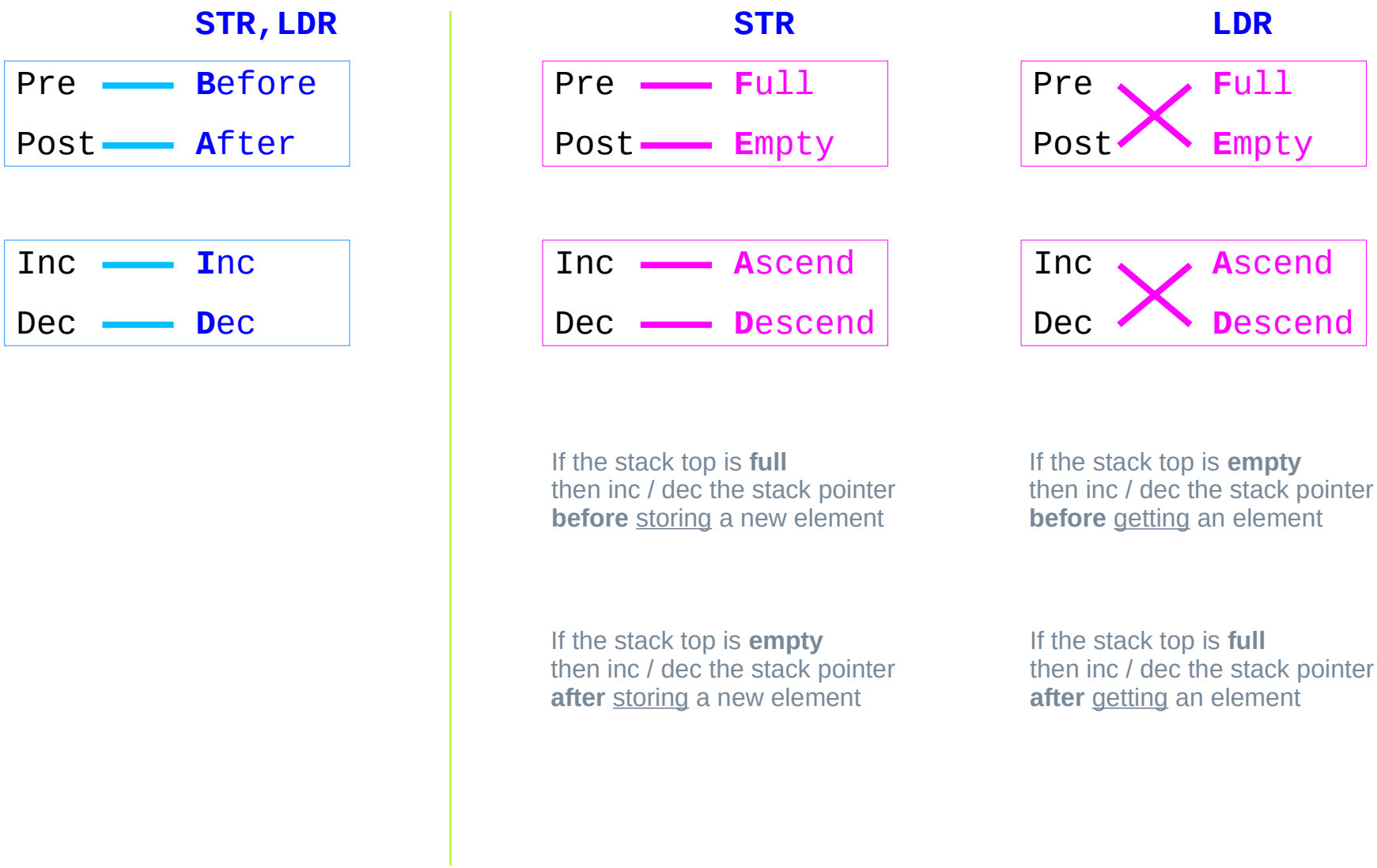
| Pre | Dec | Load |
|---|---|---|
| Before | Dec | LDRDB |
| Empty | Ascend | LDREA |

| Pre | Dec | Store |
|---|---|---|
| Before | Dec | STRDB |
| Full | Descend | STRFD |

| Post | Dec | Load |
|---|---|---|
| Before | Dec | LDRDA |
| Full | Ascend | LDRFA |

| Post | Dec | Store |
|---|---|---|
| Before | Dec | STRDA |
| Empty | Descend | STRED |

# Multiple data transfer instruction mnemonic rules



# Multiple data transfer instructions – auto indexing !

**Rn !** Updates the base register Rn

**I**ncrement / **D**ecrement the **base register Rn**

| | |
|---|---|
| **LDMIB, LDMIA**<br>**STMIB, STMIA** | **LDMDB, LDMDA**<br>**STMDB, STMDA** |

Increment / Decrement the **stack top pointer Rn**

| | | |
|---|---|---|
| **LDMFD, LDMED**<br>**STMFA, STMEA** | **LDMFA, LDMEA**<br>**STMFD, STMED** | **…. Pop**<br>**…. Push** |
| **Descending**<br>**Ascending** | **Ascending**<br>**Descending** | **…. Pop (Load)**<br>**…. Push (Store)** |

# Optional suffix ^ – recover CPSR

must not use it in **User** mode or **System** mode.

- LDM {<cond>} <add mode> Rn {!}, <registers + **pc**>^

  If op is **LDM** and <reglist> contains the **pc** (**r15**),

  in addition to the normal multiple register transfer,
  the **SPSR** is copied into the **CPSR**.

  this is for returning from **exception handlers**.

  use this only from **exception modes**.

http://infocenter.arm.com/help/topic/com.arm.doc.dui0068b/DUI0068.pdf#E7.CIHCADDA

# Optional suffix ^ – user mode register load/store

must not use it in **User** mode or **System** mode.

- LDM | STM {<cond>} <add mode> Rn, <registers - **pc**>^

  when pc is not included in <reglist>,
  data is transferred into or out of the **User** mode registers

  instead of the current non-user mode registers

  no auto write back **!** is allowed

http://infocenter.arm.com/help/topic/com.arm.doc.dui0068b/DUI0068.pdf#E7.CIHCADDA

# Software Interrupt (SWI)

**Software Interrupt (SWI)**

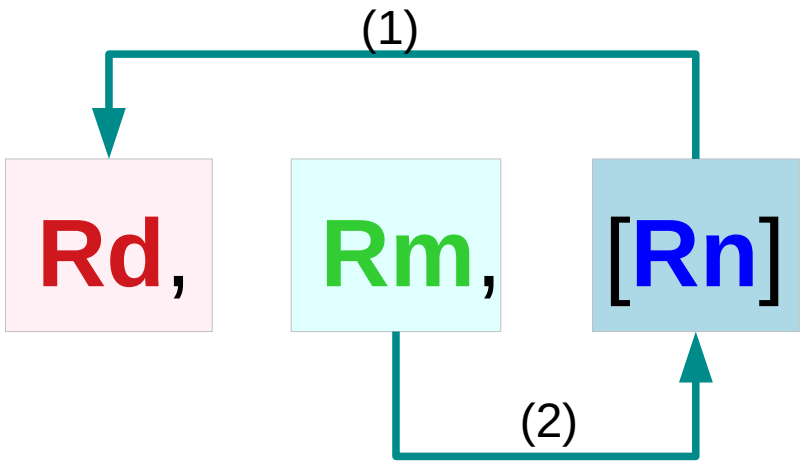
SWI {<cond>}  <24-bit immediate>

# Swap Memory and Register Instructions

**Swap Memory and Register Instrucitons**

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

Rd := [Rn], [Rn] = Rm
the swap address by the base register (Rn)



# Count Leading Zeros (CLZ)

**Count Leading Zeros (CLZ)**

CLZ {<cond>}  Rd, Rm

## References

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