

ISA Addressing Mode (2A)

Copyright (c) 2014 - 2019 Young W. Lim.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

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

This document was produced by using LibreOffice.

Based on

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

Registers

MRS {<cond>}	Rd , CPSR
MRS {<cond>}	Rd , SPSR
<op> {<cond>} {S}	Rd , Rn , #<32-bit immediate>
LDR STR {<cond>} {B}	Rd , [Rn , <offset>] {!}
LDR STR {<cond>} {B} {T}	Rd , [Rn], <offset>
LDR STR {<cond>} {B}	Rd , LABEL
LDR STR {<cond>} H SH SB	Rd , [Rn , <offset>] {!}
LDR STR {<cond>} H SH SB	Rd , [Rn], <offset>
LDM STM {<cond>} <add mode>	Rn {!}, <registers>
<op> {<cond>} {S}	Rd , Rn , Rm , {<shift>}
MSR {<cond>}	CPSR_<field>, Rm
MSR {<cond>}	SPSR_<field>, Rm
MUL {<cond>} {S}	Rd , Rm , Rs
MLA {<cond>} {S}	Rd , Rm , Rs , Rn
<mul> {<cond>} {S}	RdHi , RdLo , Rm , Rs
SWP {<cond>} {B}	Rd , Rm , [Rn]
CLZ {<cond>}	Rd , Rm

Registers in Data Processing Instructions

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

Rd ← **Rn** 1st Operand + Immediate
Rd ← **Rn** **Rm** 1st Operand + 2nd Operand

Registers in Data Transfer Instructions

Rn 1st Operand Reg / Base Reg

Rm 2nd Operand Reg / Operand Reg / Offset Reg / Source Reg

Rd Source / Destination Reg

Rd ← **[Rn]** **Load** dst reg from base reg

Rd → **[Rn]** **Store** src reg to base reg

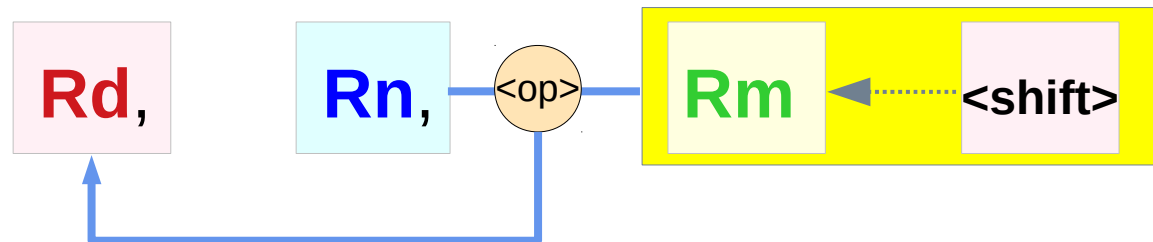
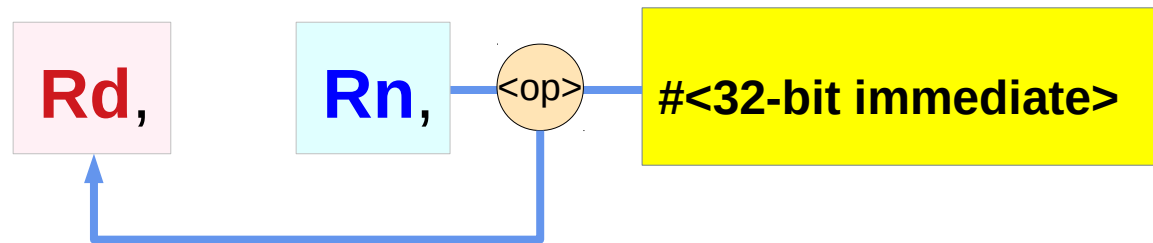
[Rn] ← **Reg list** **Store Multiple** list of regs to base reg

[Rn] → **Reg list** **Load Multiple** list of regs from base reg

Registers in data processing

<op> {<cond>} {S}
 <op> {<cond>} {S}

Rd, Rn, #<32-bit immediate>
Rd, Rn, Rm, {<shift>}



a shifted operand

ADD r3, r2, r1,
 ADD r5, r5, r3,
 MOV r0, r0,

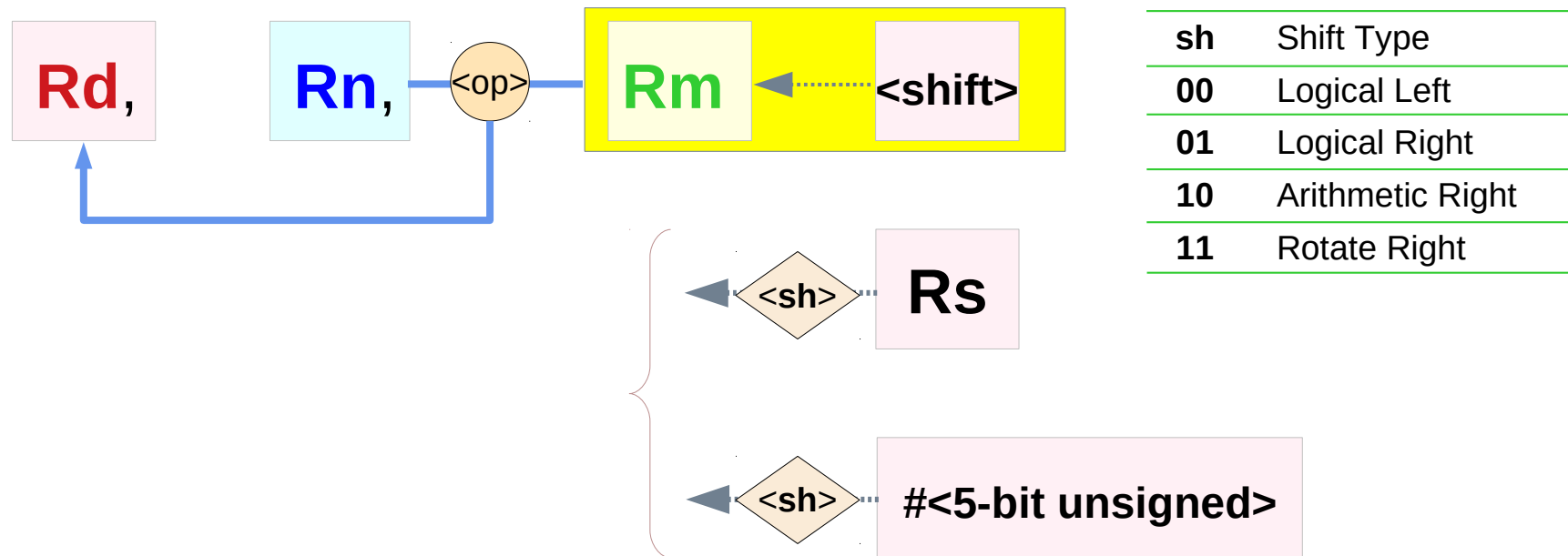
LSL #3
LSL r2
LSR #2

; r3 := r2 + r1 * 2³
 ; r5 := r5 + r3 * 2^{r2}
 ; r0 := r0 / 2²

Registers in data processing with a shifted operand

$\langle \text{op} \rangle \{ \langle \text{cond} \rangle \} \{ S \}$

$Rd, Rn, Rm, \{ \langle \text{shift} \rangle \}$

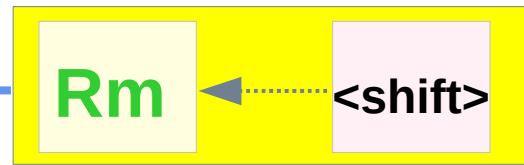


Registers in stand alone shift instructions

<op> {<cond>} {S}
 <op> {<cond>} {S}

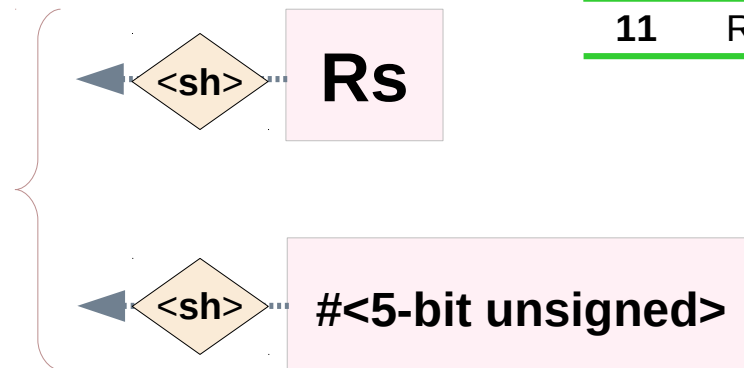
Rd, **Rm**, **Rs**
Rd, **Rm**, **#n**

Rd,



Sh	Shift Type
00	Logical Left
01	Logical Right
10	Arithmetic Right
11	Rotate Right

<op>	Shift Type
LSL	Logical Left
LSR	Logical Right
ASR	Arithmetic Right
ROR	Rotate Right



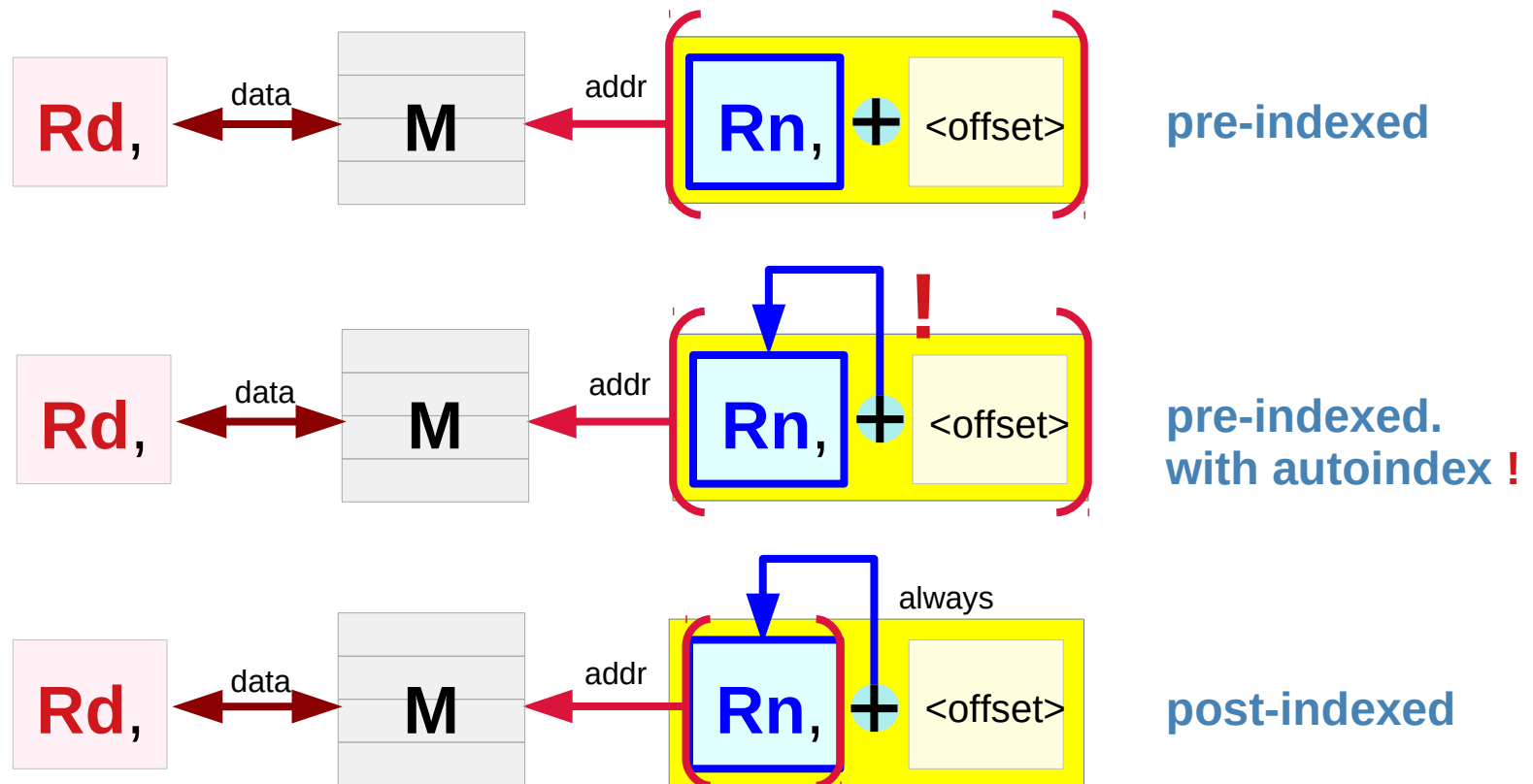
RRX {<cond>} {S}

Rd, **Rm**

Registers in data transfer

LDR|STR {<cond>} {B}
LDR|STR {<cond>} {B} {T}
LDR|STR {<cond>} {B}

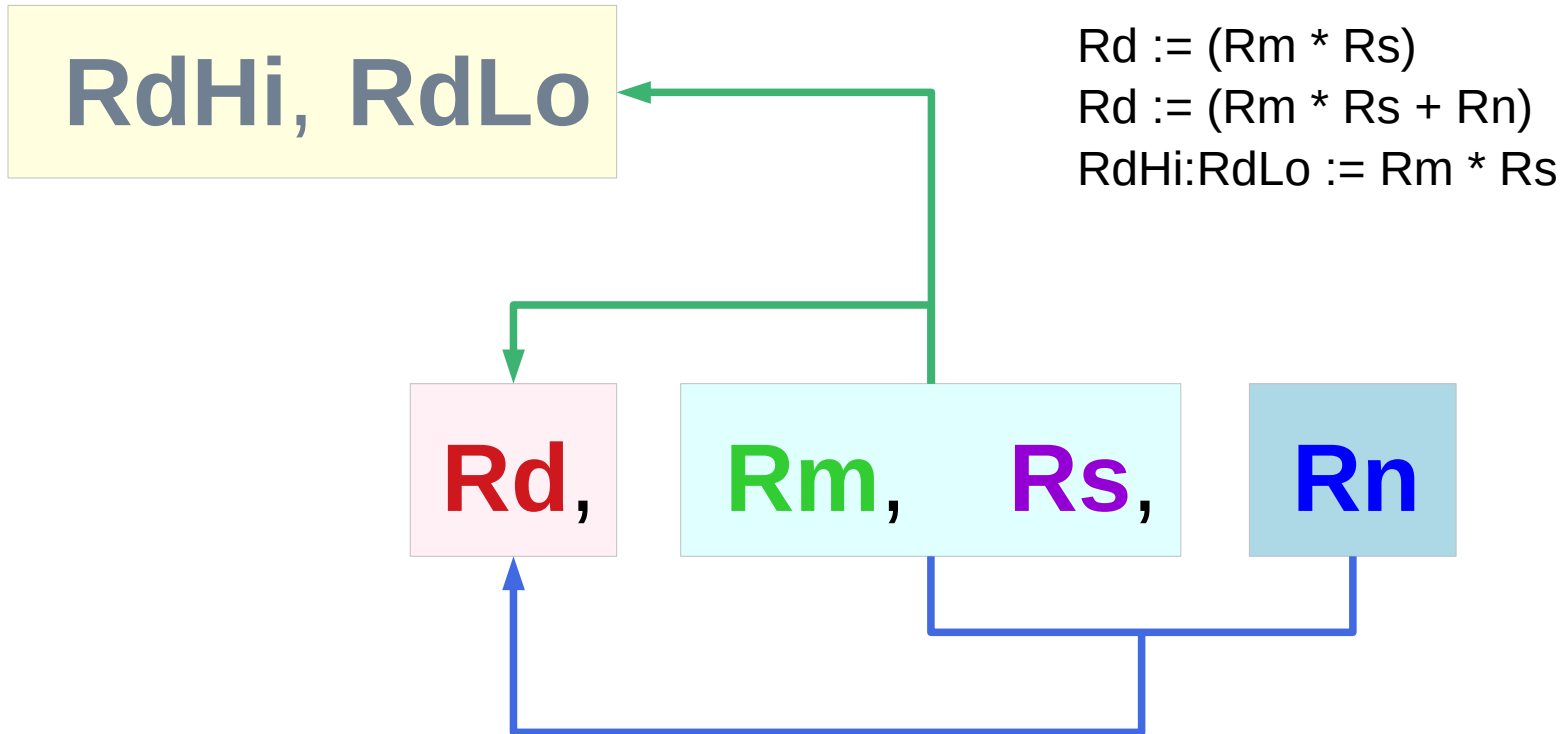
Rd, [**Rn**, <offset>] {!}
Rd, [**Rn**], <offset>
Rd, LABEL



Registers in multiplication

MUL {<cond>} {S}
MLA {<cond>} {S}
<mul> {<cond>} {S}

Rd, Rm, Rs
Rd, Rm, Rs, Rn
RdHi, RdLo, Rm, Rs



Status registers

MRS {<cond>}

MRS {<cond>}

MSR {<cond>}

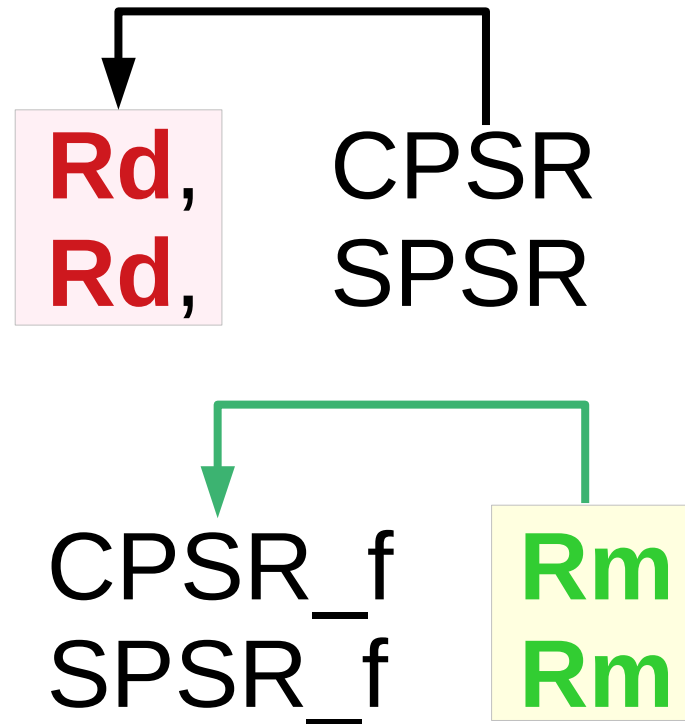
MSR {<cond>}

Rd, CPSR

Rd, SPSR

CPSR_<field>, **Rm**

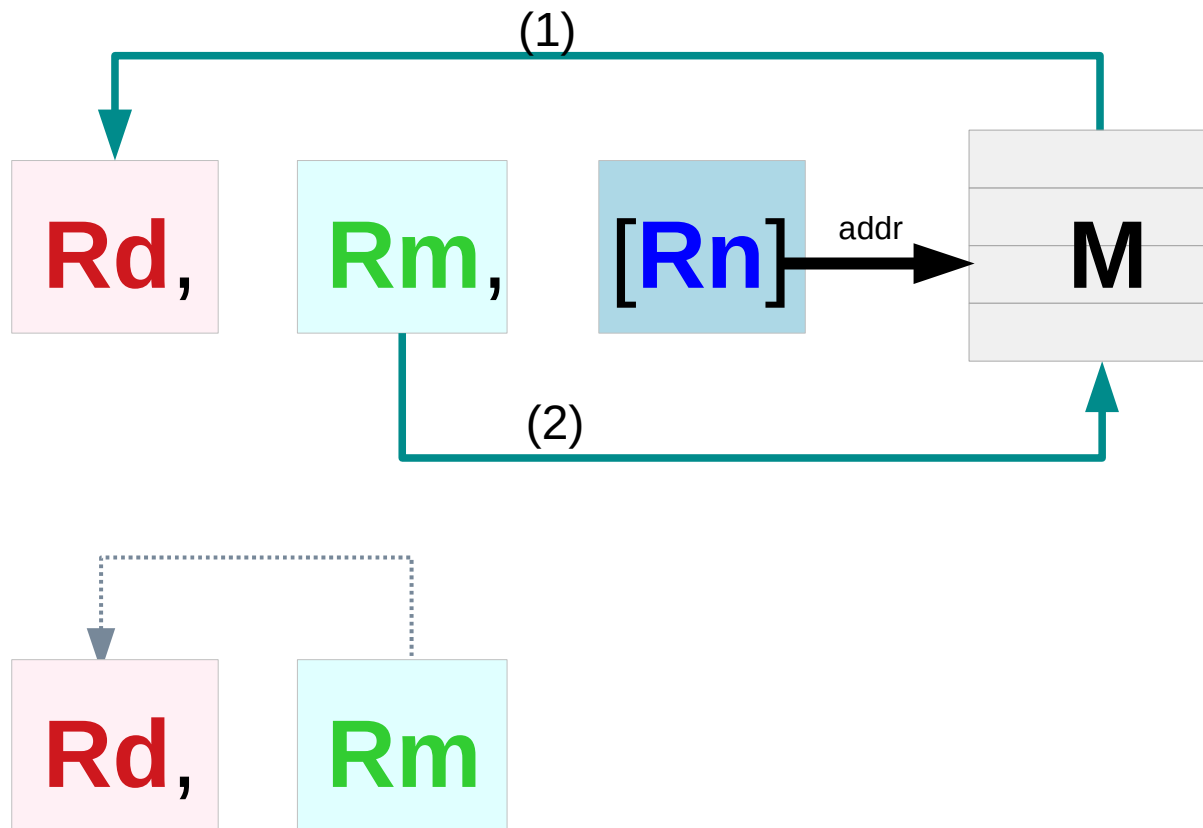
SPSR_<field>, **Rm**



Swap and clear zero registers

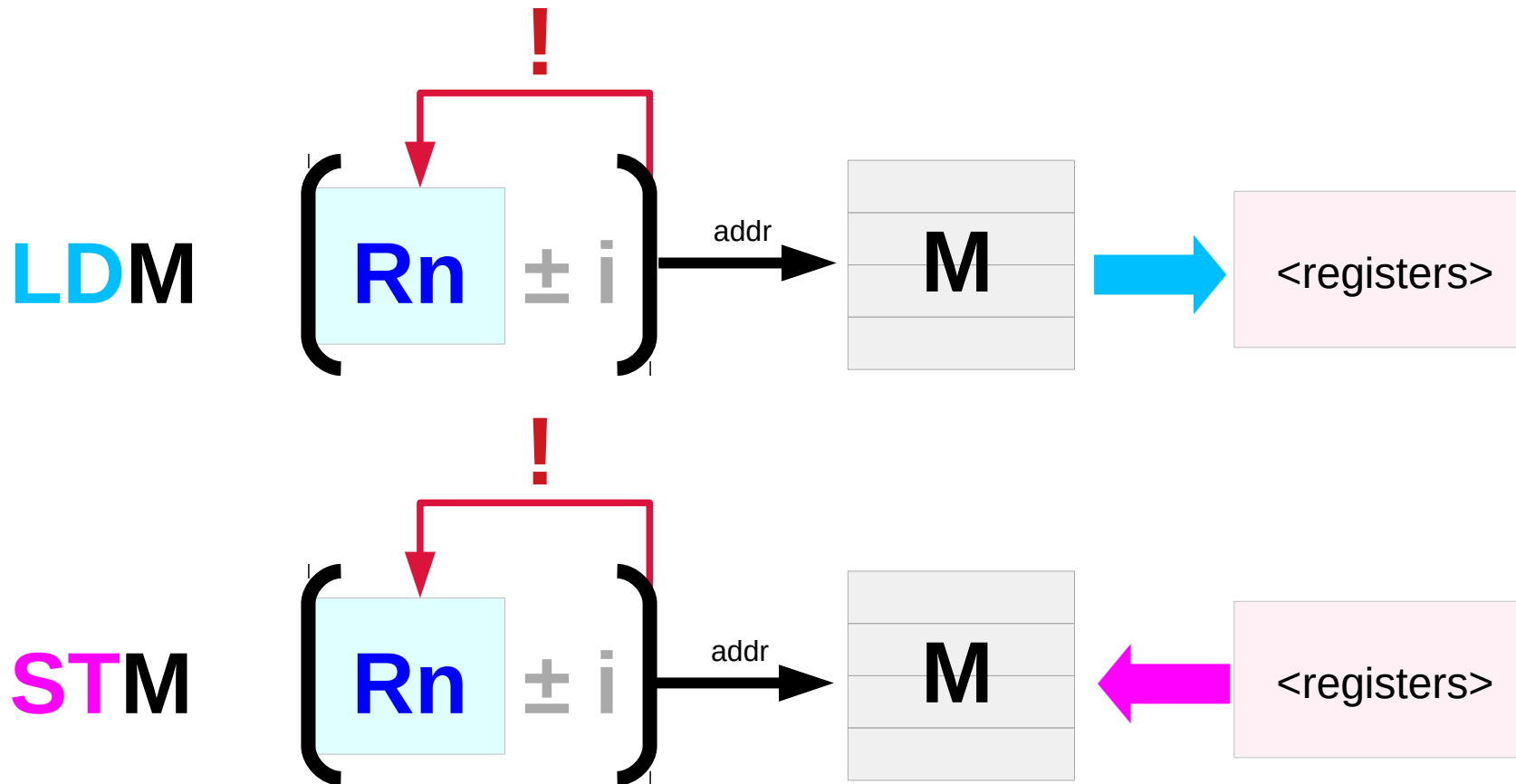
SWP {<cond>} {B}
CLZ {<cond>}

Rd, Rm, [Rn]
Rd, Rm



Registers in Multiple Data Transfers

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



General Addressing

4-address instruction

ADD d, s1, s2, next_i ; $d := s1 + s2$

3-address instruction

ADD d, s1, s2 ; $d := s1 + s2$

2-address instruction

ADD d, s1 ; $d := d + s1$

1-address instruction

ADD s1 ; $acc := acc + s1$

0-address instruction

ADD ; $top := top + next$ of the stack

General Addressing Modes

- | | |
|---|--|
| 1. immediate addressing | a binary <u>value</u> |
| 2. absolute addressing | a <u>full</u> binary <u>address</u> |
| 3. indirect addressing | <u>address</u> of a desired <u>value</u> |
| 4. register addressing | a desired <u>value</u> in a register |
| 5. register indirect addressing | <u>address</u> in a register |
| 6. base plus <u>offset</u> addressing | binary offset |
| 7. base plus <u>index</u> addressing | offset in a register (index) |
| 8. base plus <u>scaled index</u> addressing | multiplied by a constant |
| 9. stack addressing | |

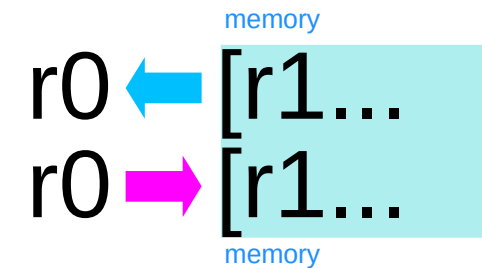
General Addressing Modes

<op> r0 ← r1 ★ r2

MOV r0 ← r1

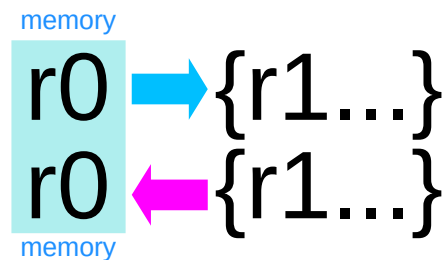
Data Processing

LDR r0 ← [r1...]
STR r0 → [r1...]



Data Transfer – Single

LDM r0 → {r1...}
STM r0 ← {r1...}



Data Transfer – Multiple

Single register load and store instructions

LDR r0, ← [r1] ; r0 := mem₃₂[r1]
STR r0, → [r1] ; mem₃₂[r1] := r0

5. register indirect addressing

Base plus offset addressing

pre-indexed

```
LDR r0, [r1, #4] ; r0 := mem32[r1 + 4]
```

pre-indexed, auto-index

```
LDR r0, [r1, #4] ! ; r0 := mem32[r1 + 4]  
; r1 := r1 + 4
```

post-indexed

```
LDR r0, [r1], #4 ; r0 := mem32[r1]  
; r1 := r1 + 4
```

6. base plus offset addressing

base : r1, offset #4

Pre-indexed Addressing

pre-indexed

```
LDR    r0, [r1, #4]    ; r0 := mem32[r1 + 4]
```

[base, offset] compute first, then transfer

pre-indexed, auto-index

```
LDR    r0, [r1, #4] ! ; r0 := mem32[r1 + 4]
```

; r1 := r1 + 4

[base, offset] compute first, then transfer
and then update base register

Post-indexed Addressing

post-indexed

```
LDR    r0, [r1], #4    ; r0 := mem32[r1]
```

```
        ; r1 := r1 + 4
```

using [base], transfer, then update base always

no !

My notation

LDR r0, [r1, #4] ; r0 := mem₃₂[r1 + 4] pre-indexed

[r1 + 4]

LDR r0, [r1, #4] ! ; r0 := mem₃₂[r1 + 4] pre-indexed, auto-index

[r1 += 4] ; r1 := r1 + 4

LDR r0, [r1], #4 ; r0 := mem₃₂[r1] post-indexed

[r1] += 4 ; r1 := r1 + 4

Shifted register operands

ADD r3, r2, **r1, LSL #3** ; r3 := r2 + r1*2³

ADD r5, r5, **r3, LSL r2** ; r5 := r5 + r3*2^{r2}

LSL	00	Logical Shift Left
LSR	01	Logical Shift Right
ASL		Arithmetic Shift Left
ASR	10	Arithmetic Shift Right
ROR	11	Rotate Right
RRX		Rotate Right eXtended

Sh	Shift Type
00	Logical Left
01	Logical Right
10	Arithmetic Right
11	Rotate Right

Shift Types

LSL #3 **00**



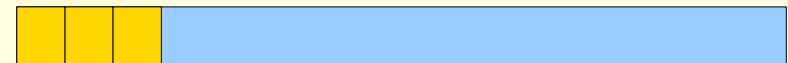
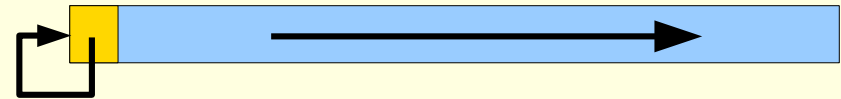
LSR #3 **01**



ASL #3 the same as LSL



ASR #3 **02**



ROR #1 **03**



RRX #3 a stand alone instruction



Stack Addressing

STMFA r8! {r0, r1, r4}

LDMFA r8! {r0, r1, r4}

STMFA r8! {r0, r1, r4}

LDMFA r8! {r0, r1, r4}

STMFD r8! {r0, r1, r4}

LDMFD r8! {r0, r1, r4}

STMED r8! {r0, r1, r4}

LDMED r8! {r0, r1, r4}

PUSH(STM) / POP(LDM)
on an **FA** type stack

PUSH(STM) / POP(LDM)
on an **EA** type stack

PUSH(STM) / POP(LDM)
on an **FD** type stack

PUSH(STM) / POP(LDM)
on an **ED** type stack

Stack Types – Semantics

Block Copy Addressing

STMIB r8! {r0, r1, r4}

Do **inc** stack top operation
before STM / LDM

LDMIB r8! {r0, r1, r4}

STMIA r8! {r0, r1, r4}

Do **inc** stack top operation
after STM / LDM

LDMIA r8! {r0, r1, r4}

STMDB r8! {r0, r1, r4}

Do **dec** stack top operation
before STM / LDM

LDMDB r8! {r0, r1, r4}

STMDA r8! {r0, r1, r4}

Do **dec** stack top operation
after STM / LDM

LMDA r8! {r0, r1, r4}

Stack Top Operations – Syntax

Ascending / Descending Stack

Full
Ascending Stack

STMFA
LDMFA

Empty
Ascending Stack

STMFA
LDMFA

Full
Descending Stack

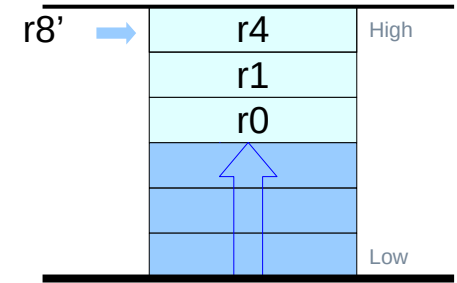
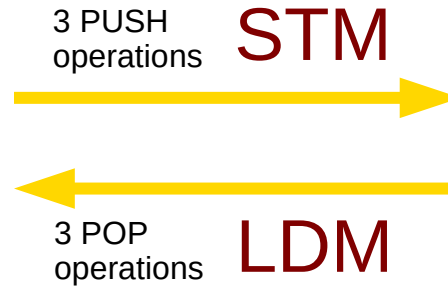
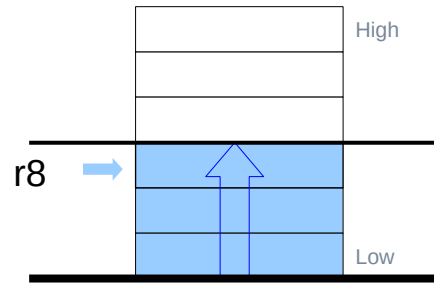
STMFD
LDMFD

Empty
Descending Stack

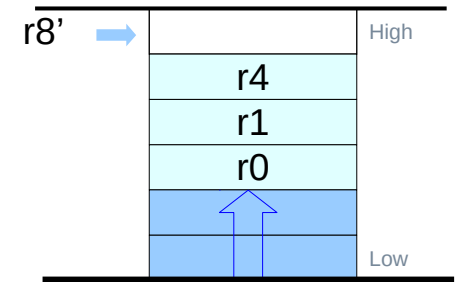
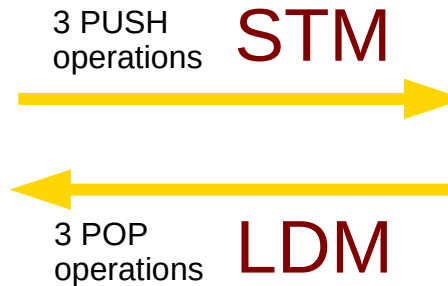
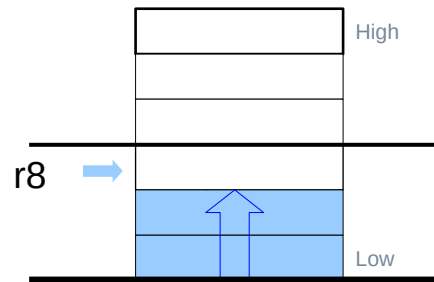
STMED
LDMED

Ascending Stack – Store / Load

Full
Ascending Stack

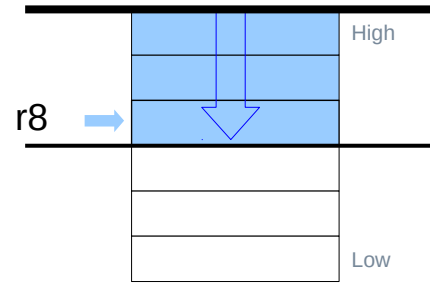


Empty
Ascending Stack



Descending Stack – Store / Load

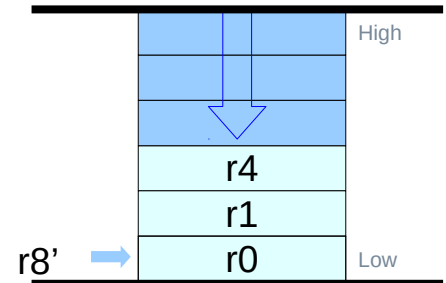
Full
Descending Stack



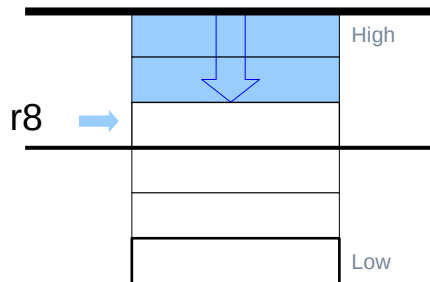
3 PUSH operations **STM**



3 POP operations **LDM**



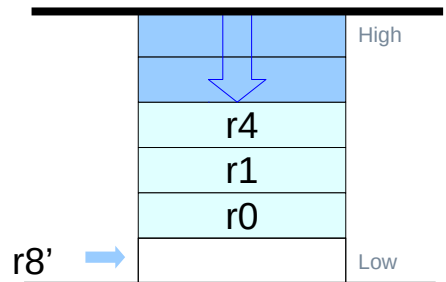
Empty
Descending Stack



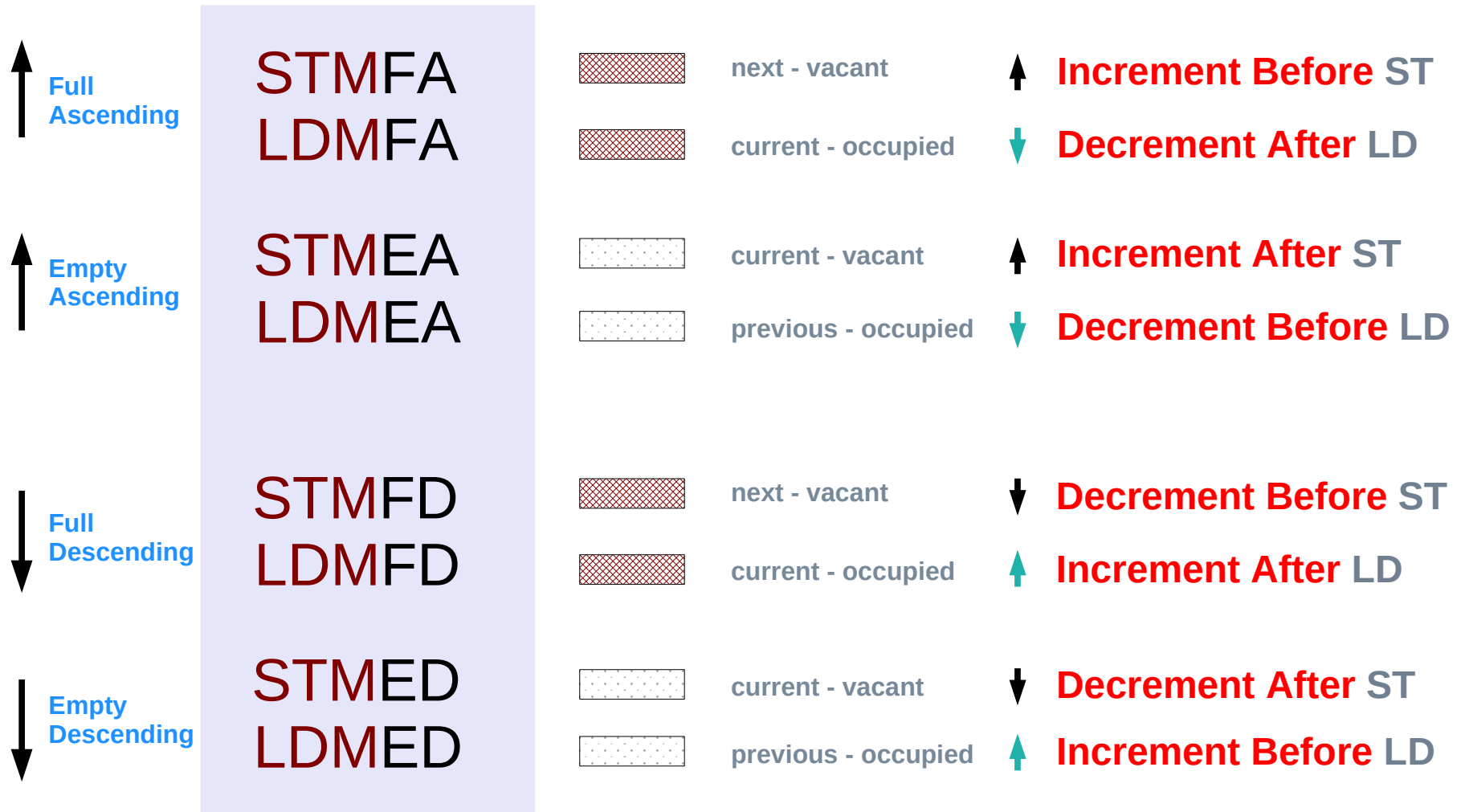
3 PUSH operations **STM**



3 POP operations **LDM**

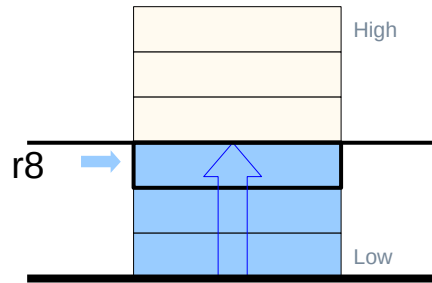


Ascending / Descending Stack & Increment / Decrement



Ascending stack – Increment / Decrement

Full
Ascending Stack



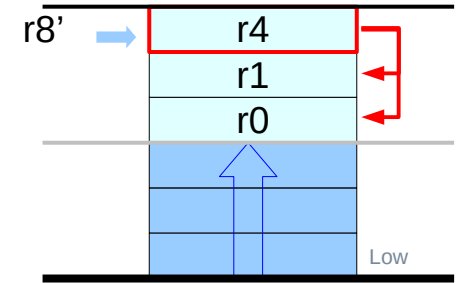
Increment Before ST

3 PUSH operations **STM**

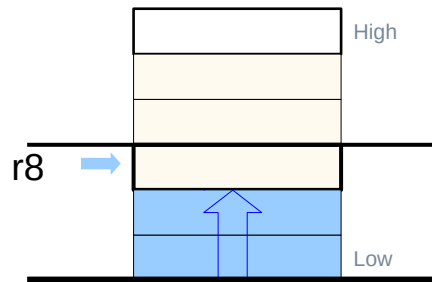


3 POP operations **LDM**

Decrement After LD



Empty
Ascending Stack



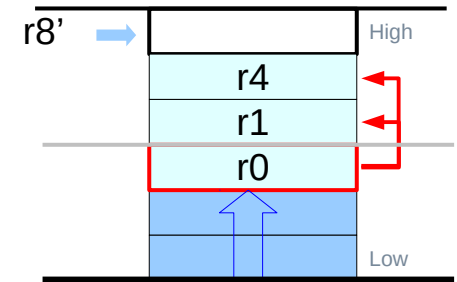
Increment After ST

3 PUSH operations **STM**



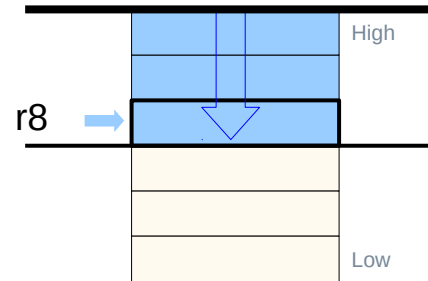
3 POP operations **LDM**

Decrement Before LD



Descending Stack – Increment / Decrement

Full
Descending Stack



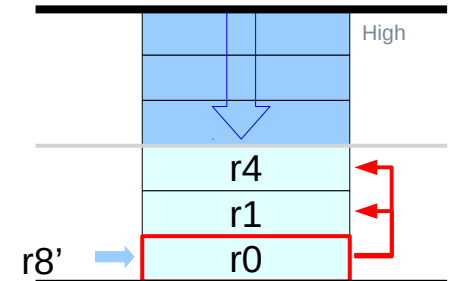
Decrement Before ST

3 PUSH operations **STM**

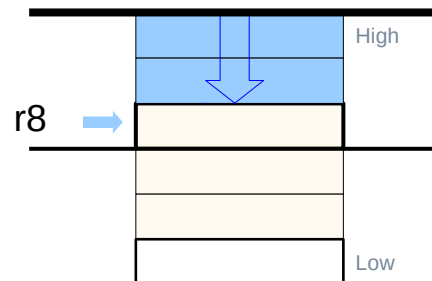


3 POP operations **LDM**

Increment After LD



Empty
Descending Stack



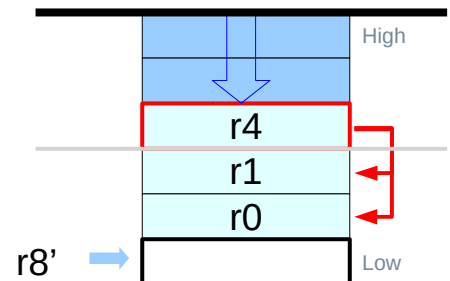
Decrement After ST

3 PUSH operations **STM**

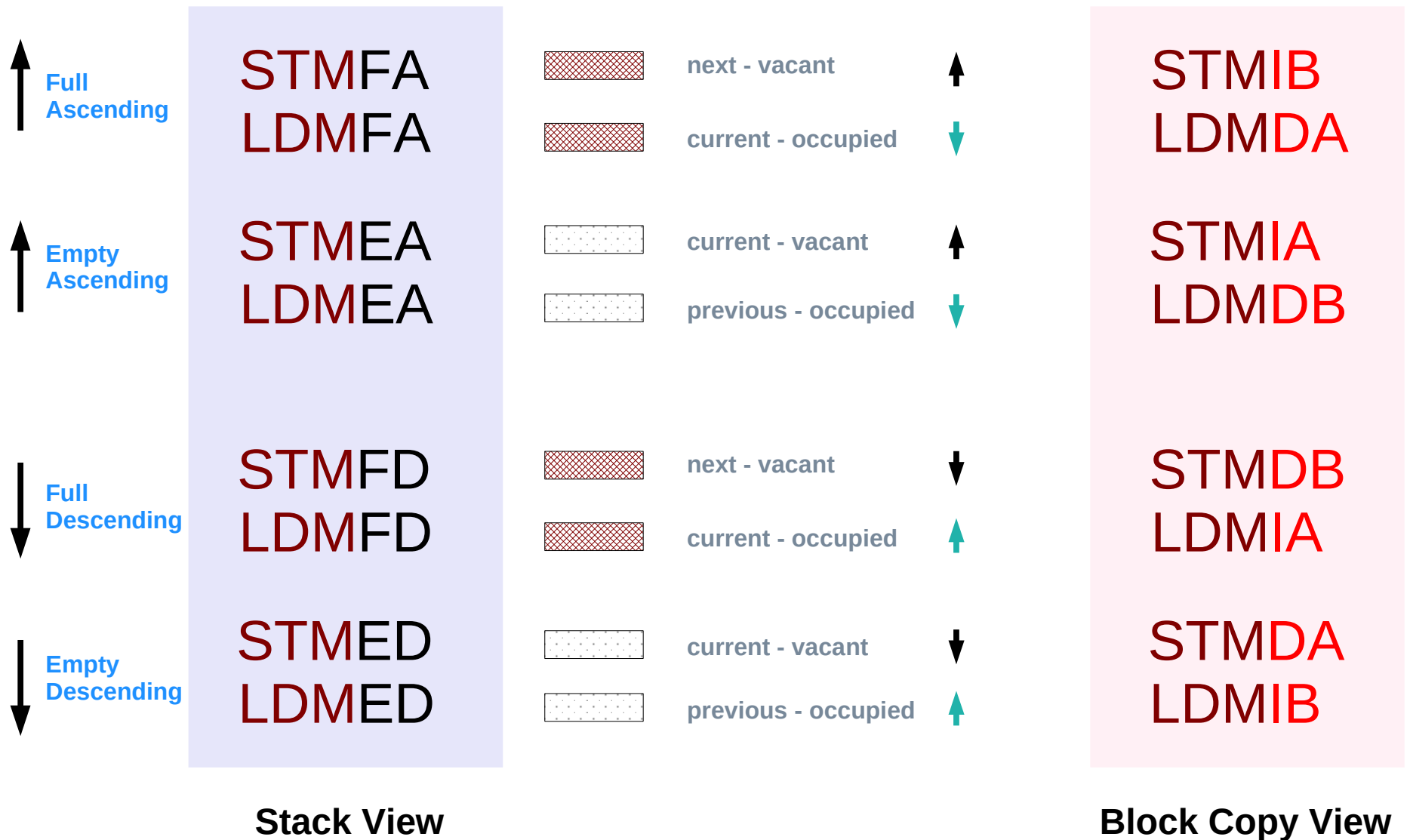


3 POP operations **LDM**

Increment Before LD

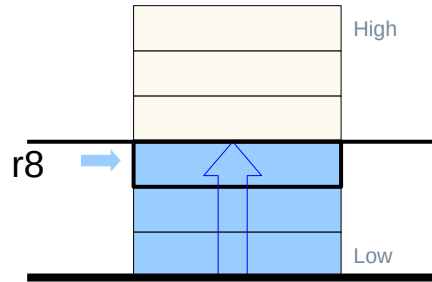


Stack and Block Copy Views



Ascending Stack – Equivalent Operations

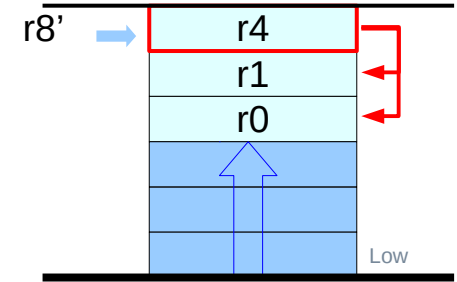
Full
Ascending Stack



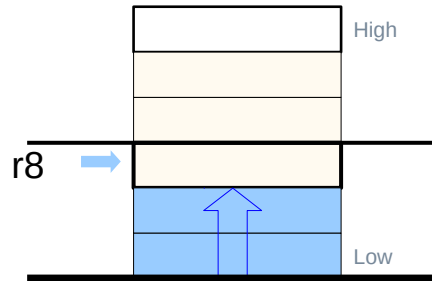
STMIB
STMFA



LDMDA
LDMFA



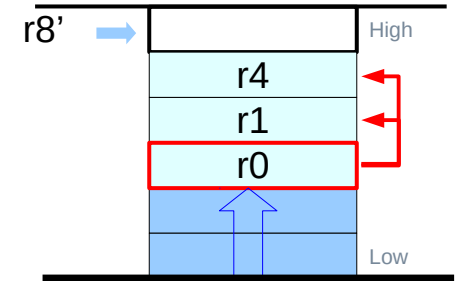
Empty
Ascending Stack



STMIA
STMFA

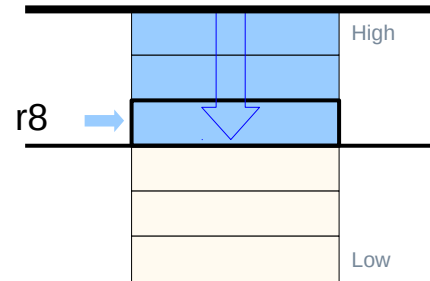


LDMDB
LDMEA



Descending Stack – Equivalent Operations

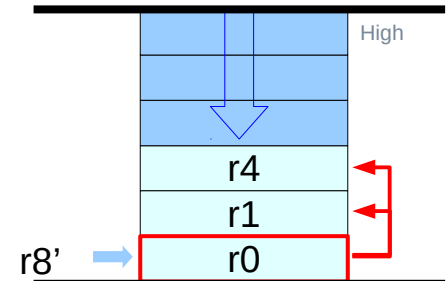
Full
Descending Stack



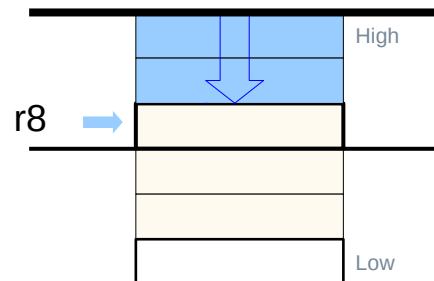
STMDB
STMFD



LDMIA
LDMFD



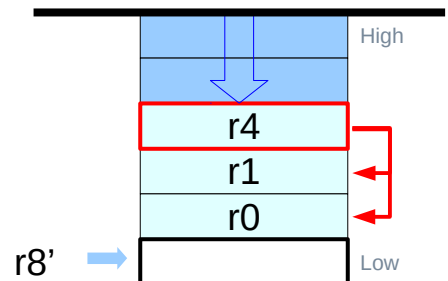
Empty
Descending Stack



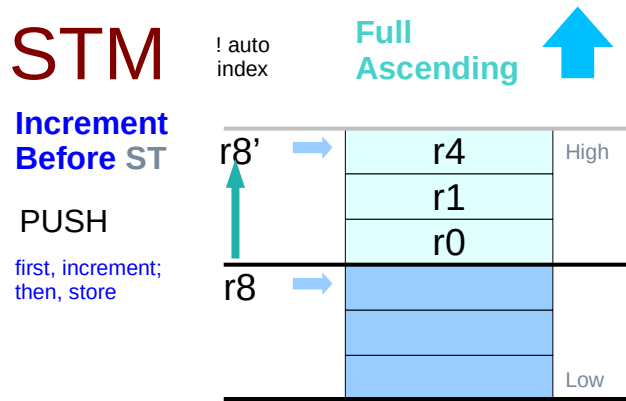
STMDA
STMED



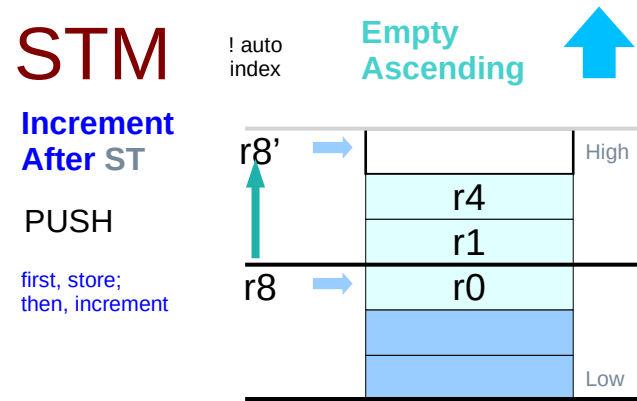
LDMIB
LDMED



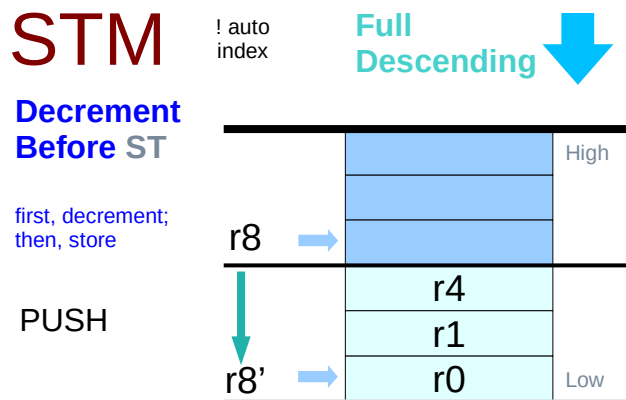
Multiple Data Transfer – STM (I,D)x(B,A)



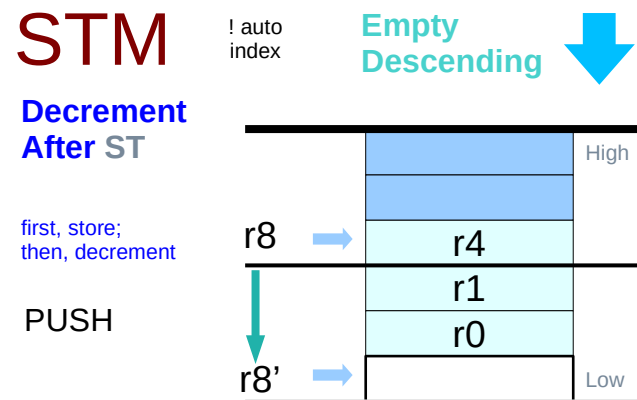
STMIB r8! {r0, r1, r4}
STMFA r8! {r0, r1, r4}



STMIA r8! {r0, r1, r4}
STMFA r8! {r0, r1, r4}



STMDB r8! {r0, r1, r4}
STMFD r8! {r0, r1, r4}



STMDA r8! {r0, r1, r4}
STMED r8! {r0, r1, r4}

Multiple Data Transfer – LDM (I,D)x(B,A)

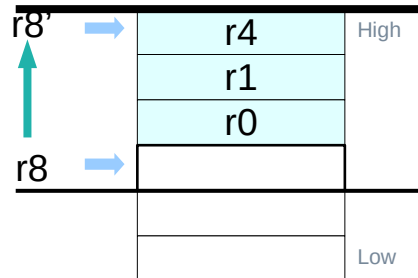
LDM

! auto index
Empty Descending ↓

Increment Before LD

POP

first, increment; then, store



LDMIB r8! {r0, r1, r4}
LDMED r8! {r0, r1, r4}

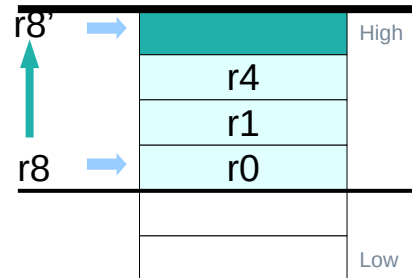
LDM

! auto index
Full Descending ↓

Increment After LD

POP

first, store; then, increment



LDMIA r8! {r0, r1, r4}
LDMFD r8! {r0, r1, r4}

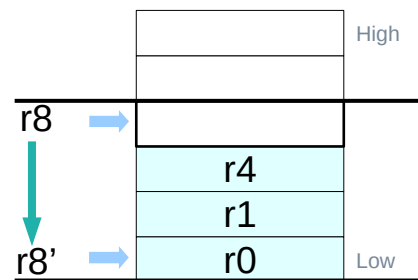
LDM

! auto index
Empty Ascending ↑

Decrement Before LD

first, decrement; then, store

POP



LDMDB r8! {r0, r1, r4}
LDMEA r8! {r0, r1, r4}

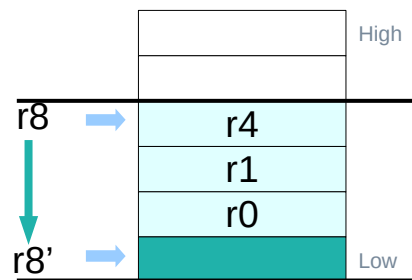
LDM

! auto index
Full Ascending ↑

Decrement After LD

first, store; then, decrement

POP



LMDA r8! {r0, r1, r4}
LDMFA r8! {r0, r1, r4}

Stack and Block Copy Views

		Full	Empty	Full	Empty
		Ascending ↑		Descending ↓	
Increment ↑	Before	STMFA STMIB			LDMED LDMIB
	After		STMEA STMIA	LDMFD LDMIA	
Decrement ↓	Before		LDMEA LDMDB	STMFD STMDB	
	After	LDMFA LMDA			STMED STMDA

References

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