ISA Assembler Format (4D)

Summary

Young Won Lim
2/5/20

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

Rn	1st Operand Reg / Base Reg
Rm	2nd Operand Reg / Operand Reg / Offset Reg / Source Reg
Rd	Left most Reg : Source / Destination Reg
S	Set Condition Codes / Signed / Restore PSR and force user bit
P	Pre/Post Index
U	Up/Down
B	Unsigned Byte/Word
H	Half-word Address
L	Link / Load/Store
W	Write-back (auto-index)
Opcode	4-bit op codes
Sh	Shift type
R	CPSR/SPSR

Rn	1st Operand Reg / Base Reg
Rm	2nd Operand Reg / Operand Reg / Offset Reg / Source Reg
Rd	Source / Destination Reg
S	Set Condition Codes / Signed / Restore PSR and force user bit
B	Unsigned Byte/Word
H	Half-word Address
L	Link / Load/Store
T	Selects the user view in the non-usermodes

<cond>	Conditions for conditional execution
<shift>	Shift type and the shift amount (except RRX)
CPSR	Current Program Status Register
SPSR	Saved Program Status Register
RdHi	The most significant 32-bits
RdLo	The least significant 32-bits

<CP#>	Coprocessor number
<Cop1>	Coprocessor operation 1
<Cop2>	Coprocessor operation 2
CRd	Coprocessor Rd
CRn	Coprocessor Rn
CRm	Coprocessor Rm

Assembler Format (1)

Branch and Branch with Link (B, BL)

B{L} {<cond>} <target address>

Branch, Branch with Link and eXchange (BX, BLX)

B{L}X {<cond>} Rm

BLX <target address>

Data Processing Instructions

<op> {<cond>} {S} Rd, Rn, #<32-bit immediate> … *12-bit encoded*

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

Single Word and Unsigned Byte Transfer Instructions

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

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

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

Half-word and Signed Byte Transfer Instructions

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

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

Multiple Register Transfer Instructions

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

Assembler Format (2)

Status Register to General Register Transfer Instructions

MRS {<cond>} Rd, CPSR | SPSR

General Register to Status Register Transfer Instructions

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

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

Multiply Instructions

MUL {<cond>} {S} Rd, Rm, Rs

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

<mul> {<cond>} {S} RdHi, RdLo, Rm, Rs

UMULL, UMLAL, SMULL, SMLAL

Software Interrupt (SWI)

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

Swap Memory and Register Instrucitons

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

Count Leading Zeros (CLZ)

CLZ {<cond>} Rd, Rm

Assembler Format (3)

Coprocessor Data Operations

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

Coprocessor Data Transfers

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

Copressor Register Transfers

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

Breakpoint Instruction (BKPT)

BKT

Unused Instruction Space

Branch and Branch with Link (B, BL)

Branch and Branch with Link (B, BL)

B{L} {<cond>} <target address>

L : the branch and link

<cond> : condition codes, AL if omitted

<target address> : a label in the assembler code

The assembler will generate the offset

target address – branch instruction address + 8

```
B          <target address>
B<cond>    <target address>
BL         <target address>
BL<cond>   <target address>
```

Branch, Branch with Link and eXchange (BX, BLX)

Branch, Branch with Link and eXchange (BX, BLX)

B{L}X {<cond>} Rm

BLX <target address>

L : the branch and link

<cond> : condition codes, **AL** if omitted

<target address> : a label in the assembler code

The assembler will generate the **offset**

target address – branch instruction address + 8

BX	Rm
BLX	Rm
BLX	<target address>
BX<cond>	Rm
BLX<cond>	Rm
BLX<cond>	<target address>

Data Processing Instructions

Data Processing Instructions

<op> {<cond>} {S} Rd, Rn, #<32-bit immediate> … *12-bit encoded*

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

Omitting Rn when the instruction is monadic (MOV, MVN)

Omitting Rd when the instruction only produces condition code

(CMP, CMN, TST, TEQ)

<shift> specifies

the shift type (LSL, LSR, ASL, ASR, ROR, RRX)

the shift amount (except RRX)

By a 5-bit immediate (# <#shift>)

By a register (Rs)

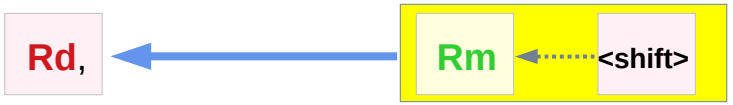
Data Processing Type 1 – no 1st operand **Rn**

Data Processing Instructions – no 1st operand Rn

<op_type1> {<cond>} {S} **Rd**, #<32-bit immediate> ... *12-bit encoded*

<op_type1> {<cond>} {S} **Rd**, Rm, {<shift>}

MOV	Move
MVN	Move Negated



Rn : SBZ (Should Be Zero) fields

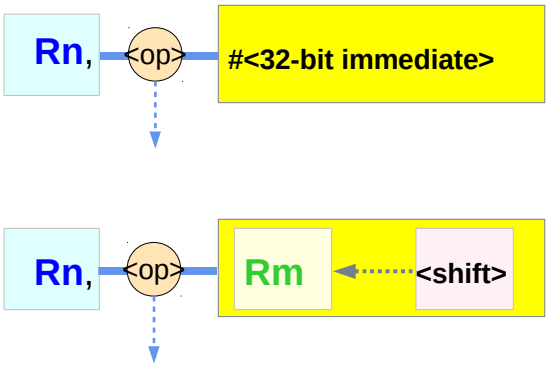
Data Processing Type 2 – no destination **Rd**

Data Processing Instructions – no destination Rd

<op_type2> {<cond>} **Rn**, #<32-bit immediate> ... *12-bit enconded*

<op_type2> {<cond>} **Rn**, Rm, {<shift>}

CMP	Compare
CMN	Compare Negated
TST	Test
TEQ	Test Equivalence



S is implicitly implied

Rd : SBZ (Should Be Zero) fields

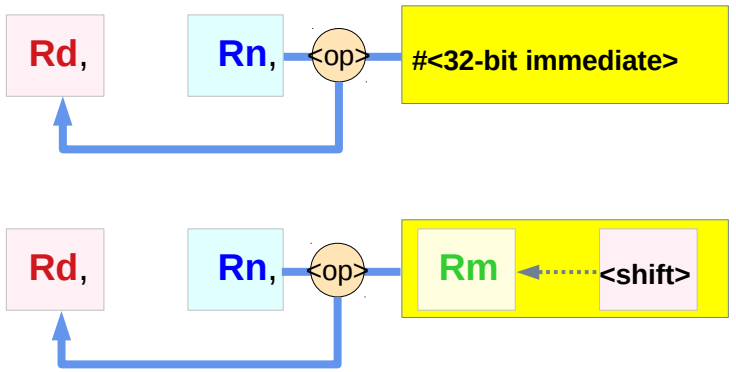
Data Processing Type 3 – Arithmetic & Logical Instructions

Data Processing Instructions

<op> {<cond>} {S} **Rd**, **Rn**, #<32-bit immediate> … *12-bit enconded*

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

AND	logical bit-wise AND
EOR	logical bit-wise Exclusive OR
SUB	Subtract
RSB	Reverse Subtract
ADD	Add
ADC	Add with Carry
SBC	Subtract with Carry
RSC	Reverse Subtract with Carry
ORR	logical bit-wise OR
BIC	Bit Clear



Data Processing – Format Listings

Data Processing Instructions

<op> {<cond>} {S} Rd, Rn, #<32-bit immediate> ... *12-bit enconded*

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

<op>	Rd, Rn, #<32-bit immediate>
<op>	Rd, Rn, Rm
<op>	Rd, Rn, Rm, <shift>
<op> <cond>	Rd, Rn, #<32-bit immediate>
<op> <cond>	Rd, Rn, Rm
<op> <cond>	Rd, Rn, Rm
<op> S	Rd, Rn, #<32-bit immediate>
<op> S	Rd, Rn, Rm
<op> S	Rd, Rn, Rm, <shift>
<op> <cond> S	Rd, Rn, #<32-bit immediate>
<op> <cond> S	Rd, Rn, Rm
<op> <cond> S	Rd, Rn, Rm, <shift>

Data Processing – Shift Operand

Data Processing Instructions with <shift>

<op>	{<cond>} {S} **Rd**, **Rn**,	**Rm**, {**<shift>**}	AND,EOR,SUB,RSB,ADD,ADC,SBC,RSC,ORR,BIC	
<op_type1>	{<cond>} {S} **Rd**,	**Rm**, {**<shift>**}	MOV, MVN	… special case
<op_type2>	{<cond>} **Rn**,	**Rm**, {**<shift>**}	CMP, CMN, TST,TEQ	… special case

<shift>

<shift type>	# <#shift> …. instruction-specified shift amount
<shift type>	Rs …. register-specified shift amount

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	… no shift amount

Data Processing – simulating shift instruction

Move Instructions with <shift>

MOV {<cond>} {S} **Rd**, **Rm**, **<shift>**

<shift>

<shift type> # <#shift> …. instruction-specified shift amount

<shift type> Rs …. register-specified shift amount

can simulate statndalone shift instructions of other machines

MOV R0. R0, LSR 2

MOV R0. R0, LSR Rs

Stand Alone Shift Instruction Synonyms … syntactic sugars

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

LSL, LSR, ASL, LSR, ASR, ROR no RRX

RealView Development Suite Ver 4.0 http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dui0204j/Cjacbgca.html

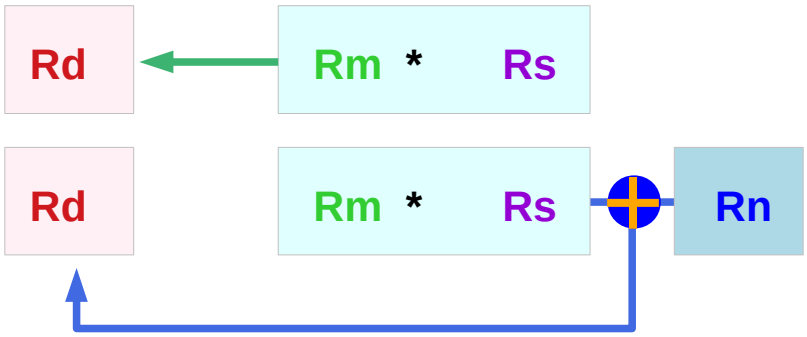
Multiply Instructions – 32 bit products

Producing the least significant 32 bits of products

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

```
MUL          Rd, Rm, Rs
MUL  S       Rd, Rm, Rs
MUL    <cond> Rd, Rm, Rs
MUL  S <cond> Rd, Rm, Rs

MLA          Rd, Rm, Rs, Rn
MLA  S       Rd, Rm, Rs, Rn
MLA    <cond> Rd, Rm, Rs, Rn
MLA  S <cond> Rd, Rm, Rs, Rn
```



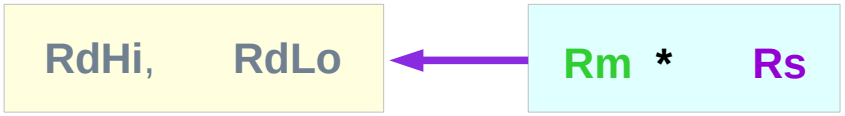
Rd := (Rm * Rs)
Rd := (Rm * Rs + Rn)

Multiply Instructions – 64 bit products

Producing the full 64 bits of products

<mul> {<cond>} {S} RdHi, RdLo, Rm, Rs

<mul> : UMULL
UMLAL
SMULL
SMLAL



RdHi:RdLo := Rm * Rs

<mul> RdHi, RdLo, Rm, Rs
<mul>S RdHi, RdLo, Rm, Rs
<mul> <cond> RdHi, RdLo, Rm, Rs
<mul>S <cond> RdHi, RdLo, Rm, Rs

Status Reg to General Reg Transfer Instructions

Status Register to General Register Transfer Instructions

MRS {<cond>} Rd, CPSR | SPSR

```
MRS          Rd, CPSR
MRS          Rd, SPSR
MRS <cond>   Rd, CPSR
MRS <cond>   Rd, SPSR
```

General Reg to Status Reg Transfer Instructions

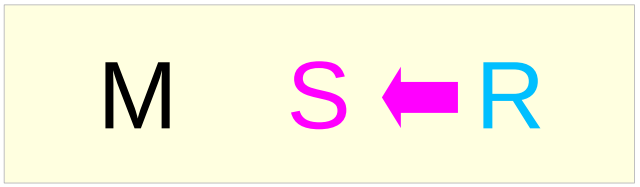
General Register to Status Register Transfer Instructions

MSR {<cond>} CPSR_f | SPSR_f, #<32-bit immediate>
MSR {<cond>} CPSR_<field> | SPSR_<field>, Rm

_<field> is one of

_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 flag field	PSR[31:24]	

```
MSR          CPSR_f, #<32-bit immediate>
MSR          SPSR_f, #<32-bit immediate>
MSR <cond>   CPSR_f, #<32-bit immediate>
MSR <cond>   SPSR_f, #<32-bit immediate>
MSR          CPSR_<field>, Rm
MSR          SPSR_<field>, Rm
MSR <cond>   CPSR_<field>, Rm
MSR <cond>   SPSR_<field>, Rm
```



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>

Data Transfer Instructions – single word and unsigned byte

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

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

Data Transfer Instructions – half-word and signed data

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>

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>

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

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 {!}, <registers + **pc**>^

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

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

Software Interrupt (SWI)

Software Interrupt (SWI)

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

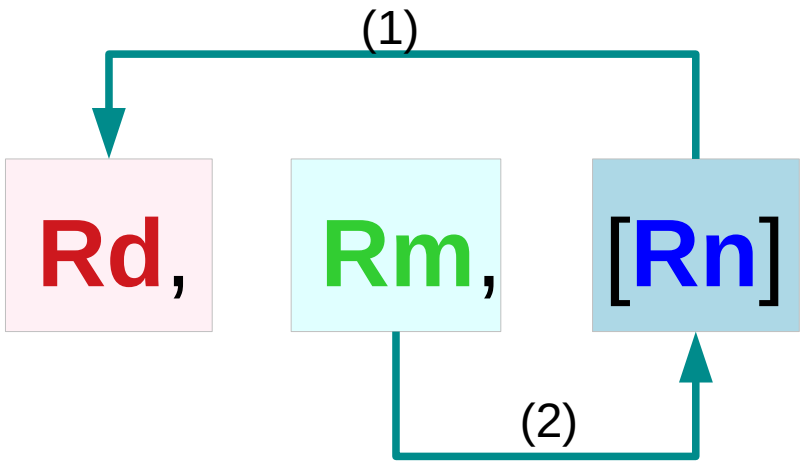
Swap Memory and Register Instructions

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

Coprocessor Instruction

Coprocessor Data Transfers

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

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

Coprocessor Data Operations

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

Coprocessor Register Transfers

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

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

Coprocessor Data Transfers – Preindex

Preindex Coprocessor Data Transfer

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

CP# : Coprocessor Number

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

Coprocessor Data Transfers – Postindex

Postindex Coprocessor Data Transfer

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

CP# : Coprocessor Number

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

Coprocessor Data Operations

Coprocessor Data Processing Operations

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

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

CP# : Coprocessor Number

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

Coprocessor Register Transfers (R←C)

Move to ARM Register from Coprocessor

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

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

CP# : Coprocessor Number

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

Coprocessor Register Transfers (C←R)

Move to Coprocessor from ARM Registers

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

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

CP# : Coprocessor Number

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

Breakpoint Instruction

Breakpoint Instruction (BKPT)

BRK

References

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