ISA Assembler Format (4A)

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

This document was produced by using LibreOffice.

Young Won Lim
9/5/19

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>

<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 (CLS)

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)

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>

Branch Conditions

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 give 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 low or same

Data Processing Instructions (1)

Data Processing Instructions

<op> {<cond>} {S} Rd, Rn, #<32-bit immediate>
<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, ASI, ASR, ROR, RRX)
Except RRX, the shift amount
    By a 5-bit immediate (<shift>)
    Bu a register (Rs)

Data Processing Instructions (2)

Data Processing Instructions

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

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

Data Processing Instructions

Opcode	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	RSUB	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	See on Rn AND Op2

Conditional Flags

N=1 if the result is negative

Z=1 if the result is zero

C=1 the carry out of the ALU when the operation is arithmetic
(ADD, ADC, SUB, SBC, RSB, RSC, CMP, CMN), or
the carry out of the shifter (C is preserved when no shift)

V=1 if overflow is occurred during arithmetic operations
only when an arithmetic operation has operands
that are viewed as 2's complement signed value
(V is preserved when non-arithmetic operations)

Condition Code Suffixes <cond>

Suffix	Flags	Meaning
EQ	Z = 1	Equal
NE	Z = 0	Not equal
CS or HS	C = 1	Higher or same, unsigned
CC or LO	C = 0	Lower, unsigned
MI	N = 1	Negative
PL	N = 0	Positive or zero
VS	V = 1	Overflow
VC	V = 0	No overflow
HI	C = 1 and Z = 0	Higher, unsigned
LS	C = 0 or Z = 1	Lower or same, unsigned
GE	N = V	Greater than or equal, signed
LT	N != V	Less than, signed
GT	Z = 0 and N = V	Greater than, signed
LE	Z = 1 and N != V	Less than or equal, signed
AL	any value	Always. This is the default when no suffix is specified.

https://community.arm.com/processors/b/blog/posts/condition-codes-1-condition-flags-and-codes

Comparison Instruction

cmp	Works like **subs**, but does not store the result.
cmn	Works like **adds**, but does not store the result.
tst	Works like **ands**, but does not store the result.
teq	Works like **eors**, but does not store the result.

https://community.arm.com/processors/b/blog/posts/condition-codes-1-condition-flags-and-codes

Data Transfer Instructions – single word and unsigned byte

The pre-indexed form of t,he 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

Data Transfer Instructions – single word and unsigned byte

The pre-indexed form of t,he instruction

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

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

Data Transfer Instructions – single word and unsigned byte

The post-indexed form

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

```
LDR | STR          Rd, [Rn], <offset>
LDR | STR T        Rd, [Rn], <offset>
LDR | STR B        Rd, [Rn], <offset>
LDR | STR BT       Rd, [Rn], <offset>
LDR | STR   <cond> Rd, [Rn], <offset>
LDR | STR T <cond> Rd, [Rn], <offset>
LDR | STR BT<cond> Rd, [Rn], <offset>
LDR | STR BT<cond> Rd, [Rn], <offset>
```

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 byte

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. #+/–<12-bit immediate>

= 2. #+/–Rm {, shift}

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

Data Transfer Instructions – half-word and signed byte

The pre-indexed form of the instruction

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

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

The post-indexed form

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

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

Multiple data transfer instructions

The normal form

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

<add mode> specifies one of the addressing mode
<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

Multiple data transfer instructions – the normal form

The normal form

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

<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		(I,D)x(B,A) | (F,E)x(A,D)	Rn,	Registers
LDM | STM		(I,D)x(B,A) | (F,E)x(A,D)	Rn**!**,	Registers
LDM | STM	<cond>	(I,D)x(B,A) | (F,E)x(A,D)	Rn,	Registers
LDM | STM	<cond>	(I,D)x(B,A) | (F,E)x(A,D)	Rn**!**,	Registers

Multiple data transfer instructions

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>^

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

Optional suffix ^

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

op {cond} mode Rn{!}, reglist{^}

1) 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**.
   LDM {<cond>} <add mode> Rn {!}, <registers + pc>^

2) Otherwise, **data** is transferred into or out of the **User** mode registers instead of the current mode registers
   LDM | STM {<cond>} <add mode> Rn, <registers - pc>^

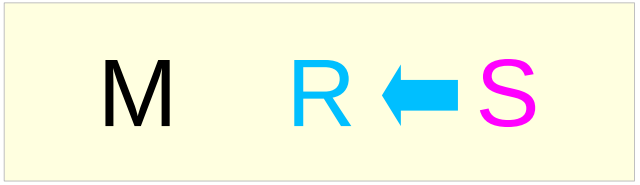
http://infocenter.arm.com/help/topic/com.arm.doc.dui0068b/DUI0068.pdf#E7.CIHCADDA

Status Register to General Register 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 Register to Status Register Transfer Instructions

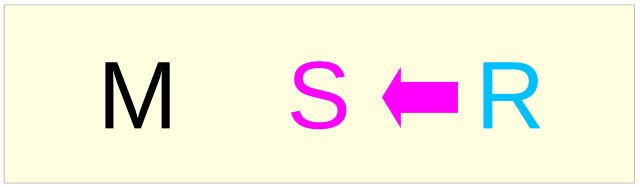
General Register to Status Register Transfer Instructions

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
```



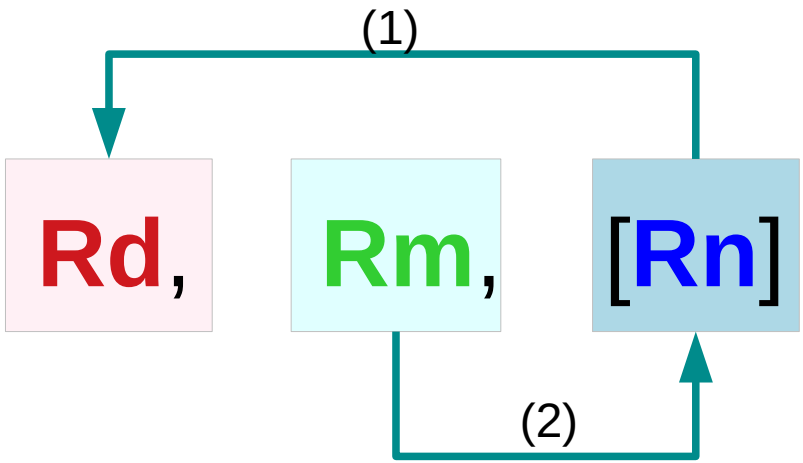
Software Interrupt (SWI)

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

Swap Memory and Register Instructions

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

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



Count Leading Zeros (CLT)

CLZ {<cond>} Rd, Rm

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
```

Multiply Instructions – 64 bit products

Producing the full 64 bits of products

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

<mul> : UMULL
UMLAL
SMULL
SMLAL

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

Multiply Instructions

Opcode [23:21]	Mnemonic	Meaning	Effect
000	MUL	Multiply (32-bit)	Rd := (Rm * Rs)
001	MLA	Multiply-Accumulate (32-bit)	Rd := (Rm * Rs + Rn)
010			
011			
100	UMULL	Unsigned Multiply Long	RdHi:RdLo := Rm * Rs
101	UMLAL	Unsigned Multiply Acc Long	RdHi:RdLo := Rm * Rs
110	SMULL	Singed Multiply Long	RdHi:RdLo := Rm * Rs
111	SMLAL	Unsigned Multiply Acc Long	RdHi:RdLo := Rm * Rs

Coprocessor Instruction

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>

Coprocessor 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>

Coprocessor Instruction

C D P
coprocessor Data Processing

L D C
Load coprocessor

S T C
Store coprocessor

M R ⬅ C
Move Reg coprocessor

M C ⬅ R
Move coprocessor Reg

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

BRK

References

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