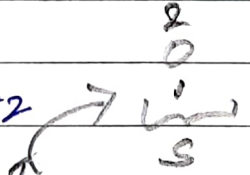
LECTURE 9 - BINARY ENCODING

SUBEQS R0, R1, #2



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10
0	0	0	0	0	0	1	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0

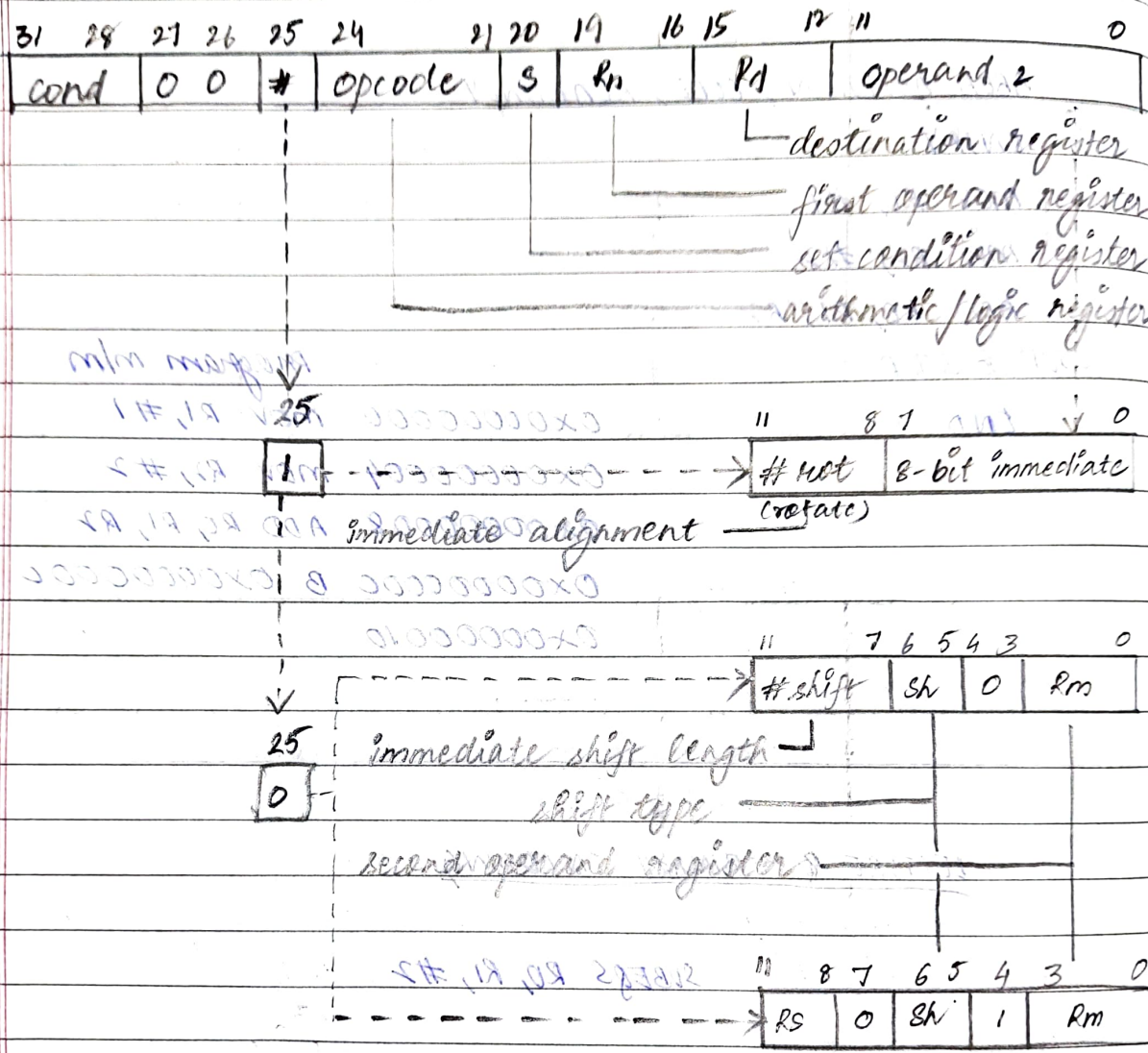
4 Bit Cond. code | Res. | Im. Op. | 4 Bit opCode | Op. 1 Reg. | Dest. Reg.

9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	1	0

4 Bit Imm. ROR | 8-bit operand 2

PTO

* DATA PROCESS :



SUBEQS R0, R1, #2

* CONDITION CODES :

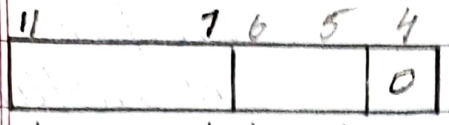
Code	Suffix	Flags	meaning
0000	EQ	Z set	equal
0001	NE	Z clear	not equal
0010	CS	C set	unsigned higher or same
0011	CC	C clear	unsigned lower
0100	MI	N set	negative

0101	PL	N clear	positive or zero
0110	VS	V set	overflow
0111	VC	V clear	no overflow
1000	HI	C set and Z clear	unsigned higher
1001	LS	C clear or Z set	unsigned lower or same
1010	GE	N equals V	greater or equal
1011	LT	N not equal to V	less than
1100	GT	Z clear AND (N equal to V)	greater than
1101	LE	Z set OR (N not equal to V)	less than or equal
1110	AL	(ignored)	always

+ OPCODES :

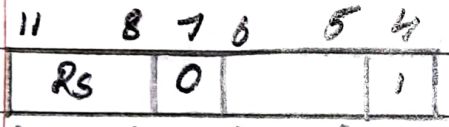
Assembler Mnemonic	OpCode	Action
AND	0000	operand1 AND operand2
EOR	0001	operand1 EOR operand2
SUB	0010	operand1 - operand2
RSB	0011	operand2 - operand1
ADD	0100	operand1 + operand2
ADC	0101	operand1 + operand2 + carry
SBC	0110	operand1 - operand2 + carry - 1
RSC	0111	operand2 - operand1 + carry - 1
TST	1000	as AND, but result is not written
TEQ	1001	as EOR, but result is not written
CMP	1010	as SUB, but result is not written
CMN	1011	as ADD, but result is not written
ORR	1100	operand1 OR operand2
MOV	1101	operand2 (operand1 is ignored)
BIC	1110	operand1 AND NOT operand2 (Bit clear)
MVN	1111	NOT operand2 (operand1 is ignored)

* SHIFTS:



Shift Type
00 = logical left
01 = logical right
10 = arithmetic right
11 = rotate right

Shift amount
5 bit unsigned integer



Shift Type
00 = logical left
01 = logical right
10 = arithmetic right
11 = rotate right

Shift register
Shift amount specified in bottom byte of Rs

LECTURE 10 : ARM INSTRUCTION SET - DATA TRANSFER - I

① Load / store instructions

② Used to move Word (32-bit), signed and unsigned Half Word (16-bit) and Byte to and from registers

③ Can be used to load PC [Program Counter]
(if target address is beyond branch instruction range)

PTO

Load - Memory to Register
Store - Register to Memory

LDR	Load Word	STR	Store Word
LDRH	Load Half Word	STRH	Store Half Word
LDRSH	Load Signed Half Word	STRSH	Store Signed Half Word
LDRB	Load Byte	STRB	Store Byte
LDRSB	Load Signed Byte	STRSB	Store Signed Byte

* SINGLE REGISTER LOAD :
(BASE PLUS OFFSET PRE-INDEXED WITH IMMEDIATE OFFSET)

```
LDR R0, = 0x40000000
LDR R1, [R0]
LDR R2, [R0, #4]
```

Base Plus Offset
Pre-Indexed Addr. Mode
with Imm. Offset

Indexing is done before load operation

	Data M/M		Data
20 → 0x40000000	0x10101010	R0	0x40000000
0x40000004	0x20202020	R1	0x10101010
0x40000008	0x00000003	R2	0x20202020

* SINGLE REGISTER LOAD : ☞ Increment First, Load Next
(BASE PLUS OFFSET PRE-INDEXED WITH IMMEDIATE OFFSET AND WRITE BACK)

```
LDR R0, = 0x40000000
LDR R1, [R0]
LDR R2, [R0, #4]!
```

Pre-indexed Addr. Mode
with Imm. Offset
& Write Back
(Auto-indexing)

Pointer value is updated at the base

	Data M/M
R0 → 0x40000000	0x10101010
0x40000004	0x20202020
0x40000008	0x00000003

	Data
R0	0x40000004
R1	0x10101010
R2	0x20202020

+ SINGLE REGISTER LOAD: → Load First, Increment Next
(BASE PLUS OFFSET POST-INDEXED WITH IMMEDIATE OFFSET)

```
LDR R0, = 0x40000000
LDR R1, [R0]
LDR R2, [R0], #4
```

← Post-indexed Addr. Mode With Imm. offset

	Data M/M
R0 → 0x40000000	0x10101010
0x40000004	0x20202020
0x40000008	0x00000003

	Data
R0	0x40000004
R1	0x10101010
R2	0x10101010

+ SINGLE REGISTER LOAD AND STORE:

```
LDR R0, = 0x40000000
LDR R1, [R0]
LDR R2, [R0], #4
STR R2, [R0, #4]!
```

← Post-indexed Addr. Mode
← Pre-indexed Addr. Mode with write back

	Data M/M
R0 → 0x40000000	0x10101010
0x40000004	0x20202020
0x40000008	0x10101010

	Data
R0	0x40000008
R1	0x10101010
R2	0x10101010

LECTURE 11: ARM INSTRUCTION SET - DATA TRANSFER - II

* SINGLE REGISTER LOAD (DECREMENT):

```
LDR R0, = 0x40000004
LDR R1, [R0]
LDR R2, [R0, #-4]!
```

Pre-indexed Addn Mode with Write Back (auto-indexing)

	Data M/M
0x40000000	0x10101010
0x40000004	0x20202020
0x40000008	0x00000003

R0 →

	Data
R0	0x40000000
R1	0x20202020
R2	0x10101010

* SINGLE REGISTER LOAD:

(BASE PLUS OFFSET PRE-INDEXED WITH REGISTER OFFSET AND WRITE BACK)

```
LDR R0, = 0x40000000
LDR R1, [R0]
mov R2, #4
LDR R3, [R0, R2]!
```

Pre-indexed Reg. offset with Write Back (auto-indexing)

[done while programming]

	Data M/M
0x40000000	0x10101010
0x40000004	0x20202020
0x40000008	0x00000003

R0 →

	Data
R0	0x40000004
R1	0x10101010
R2	0x00000004
R3	0x20202020

PTO →

* SINGLE REGISTER LOAD :

(BASE PLUS OFFSET PRE-INDEXED WITH SCALED REGISTER OFFSET AND WRITE BACK)

```
LDR R0, = 0x40000000
LDR R1, [R0]
LDR R3, [R0, R2, LSL #2]!
```

Pre-indexed
Scaled Reg. offset
with Write Back

MOV R2, #1
[done while programming]

	Data M/M
R0 → 0x40000000	0x10101010
0x40000004	0x20202020
0x40000008	0x00000002

	Data
R0	0x40000004
R1	0x10101010
R2	0x00000001
R3	0x20202020

* SINGLE REGISTER LOAD HALFWORD :

(BASE PLUS OFFSET PRE-INDEXED WITH IMMEDIATE OFFSET)

```
LDR R0, = 0x40000000
LDRH R1, [R0]
LDRH R2, [R0, #4]
```

Base Plus offset
Pre-indexed Addr. mode
with Imm. offset

	Data M/M
R0 → 0x40000000	0x10101010
0x40000004	0x20202020
0x40000008	0x00000003

	Data
R0	0x40000000
R1	0x00001010
R2	0x00002020

PTO →

* SINGLE REGISTER LOAD BYTE :
(BASE PLUS OFFSET PRE-INDEXED WITH IMMEDIATE OFFSET)

```
LDR R0, =0x40000000
LDRB R1, [R0]
LDRB R2, [R0, #4]
```

Base Plus Offset
Pre-indexed Addr. Mode
with Imm. Offset

	Data M/M
0x40000000	0x10101010
0x40000004	0x20202020
0x40000008	0x00000003

R0 → 0x40000000

	Data
R0	0x40000000
R1	0x00000010
R2	0x00000020

* SINGLE REGISTER LOAD SIGNED HALFWORD :
(BASE PLUS OFFSET PRE-INDEXED WITH IMMEDIATE OFFSET)

```
LDR R0, =0x40000000
LDRSH R1, [R0]
LDRSH R2, [R0, #4]
```

MSB Signed extension ←

Base Plus offset
Pre-indexed Addr. Mode
with Imm. offset

	Data M/M
0x40000000	0x1010FF10
0x40000004	0x20202020
0x40000008	0x00000003

R0 → 0x40000000

	Data
R0	0x40000000
R1	0xFFFFFF10
R2	0x00002020

PTO →

* SINGLE REGISTER LOAD SIGNED BYTE :
(BASE PLUS OFFSET PRE-INDEXED WITH IMMEDIATE OFFSET)

LDR R0, = 0x40000000

MSB Signed Extension ← LDRSB R1, [R0]

LDRSB R2, [R0, #4] → Base Plus Offset Pre-indexed addr. Mode with Imm. offset

	Data m/m
R0 → 0x40000000	0x101010FF
0x40000004	0x20202020
0x40000008	0x00000003

	Data
R0	0x40000000
R1	0xFFFFFFFF
R2	0x00000020

* EXAMPLE :

R0 = 0x00000000

R1 = 0x 00090000

mem 32 [0x00009000] = 0x01010101

mem 32 [0x00009004] = 0x02020202

LDR R0, [R1, #4] !

Pre-indexing with writeback :

R0 = 0x02020202

R1 = 0x00009004

LDR R0, [R1, #4]

Pre-Indexing :

R0 = 0x02020202

R1 = 0x00009000

LDR R0, [R1], #4

Post-indexing :

R0 = 0x01010101

R1 = 0x 0009004

* LOAD / STORE ADDRESSING MODES :

```
LDR     R0 , = 0x40000000       ; R0 = 0x40000000
LDR     R1 , [R0]               ; R1 = mem32[0x40000000]
LDR     R1 , [R0, #4]           ; R1 = mem32[0x40000000 + 4]
LDR     R1 , [R0, #-4]          ; R1 = mem32[0x40000000 - 4]
LDR     R1 , [R0, R2]           ; R1 = mem32[0x40000000 + R2]
LDR     R1 , [R0, -R2]          ; R1 = mem32[0x40000000 - R2]
LDR     R1 , [R0, R2, LSL #2]   ; R1 = mem32[0x40000000 + R2 << 2]
LDR     R1 , [R0, -R2, LSL #2]  ; R1 = mem32[0x40000000 - R2 << 2]
LDRH    R1 , [R0]               ; R1 = mem16[0x40000000]
LDRB    R1 , [R0]               ; R1 = mem8[0x40000000]
LDRSH   R1 , [R0]               ; R1 = Sign Ext. mem16[0x40000000]
LDRSB   R1 , [R0]               ; R1 = Sign Ext. mem8[0x40000000]
STR     R1 , [R0]               ; mem32[0x40000000] = R1
```

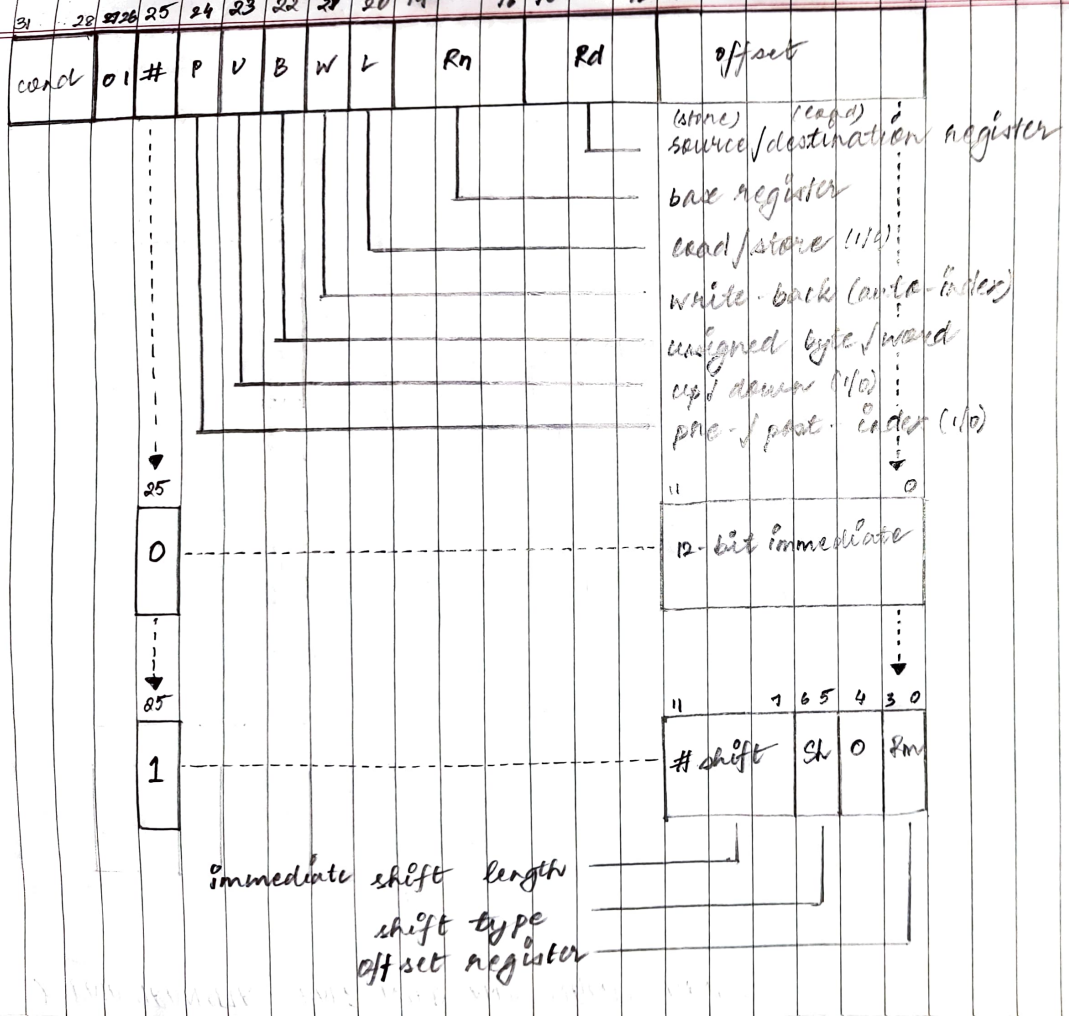
LECTURE 12 - DATA TRANSFER - BINARY ENCODING

* DATA TRANSFER INSTRUCTIONS :

① Load/store instructions
② used to move signed and unsigned word, Half Word and Byte to and from registers
③ can be used to load PC (if target address is beyond branch instruction range)

LDR	Load Word	STR	Store Word
LDRH	Load Half Word	STRH	store Half Word
LDRSH	Load Signed Half Word	STRSH	store Signed Half Word
LDRB	Load Byte	STRB	store Byte
LDRSB	Load Signed Byte	STRSB	store signed Byte

DATA TRANSFER : SINGLE WORD & UNSIGNED BYTE :



Load/store bit
0 = store to memory
1 = load from memory

Write-back bit
0 = no write-back
1 = write address into base

Byte/word bit
0 = transfer word quantity
1 = transfer byte quantity

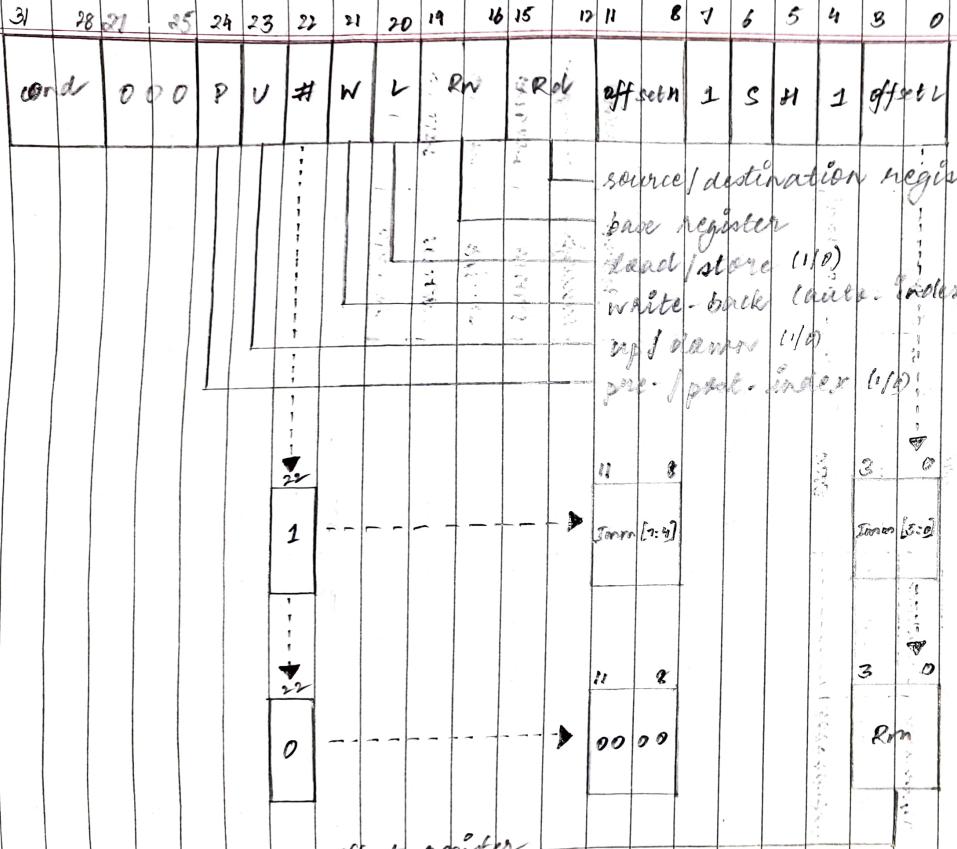
Up/Down bit
0 = down; subtract offset from base
1 = up; add offset to base

Pre/Post indexing bit
0 = post; add offset after transfer
1 = pre; add offset before transfer

Immediate offset
0 = offset is an immediate value
1 = offset is a register

Shift type
00 = logical left
01 = logical right
10 = arithmetic right
11 = rotate right

* DATA TRANSFER - HALF WORD AND SIGNED BYTE :



S H
- 00 = SWP instruction
- 01 = unsigned halfwords
- 10 = signed byte
- 11 = signed halfwords

load/store
- 0 = store to memory
- 1 = load from memory

write-back
- 0 = no write back
- 1 = write address into base

up/down
- 0 = down: subtract offset from base
- 1 = up: add offset to base

pre/post indexing
- 0 = post: add/subtract offset after transfer
- 1 = pre: add/subtract offset before transfer

LECTURE 13 - ARM INSTRUCTION SET - BLOCK DATA TRANSFER

* MULTIPLE REGISTER LOAD/STORE (BLOCK TRANSFER):

① Load Multiple instruction (LDM) - Block of data memory moved to corresponding register
② store Multiple instruction (STM) - opposite of LDM
③ LDM takes $2 + Nt$ cycles,
where N is number of registers to load
t is number of cycles required for each sequential access to memory

	Data m/m	LDM	Data
0x40000000	0x10101010	→ R0	0x10101010
0x40000004	0x20202020	→ R1	0x20202020
0x40000008	0x00000003	STM → R2	0x00000003

Syntax - <LDM|STM> {<cond>} <addressing mode> Rn{!}, <registers>

④ Addressing modes -

IA - Increment After
IB - Increment Before
DA - Decrement After
DB - Decrement Before

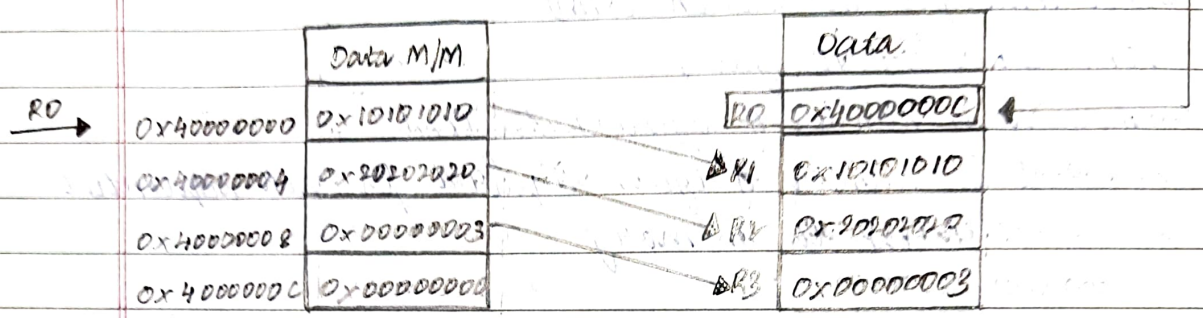
Store multiple	Load multiple
STMIA	LDMDB
STMIB	LDMDA
STMDA	LDMIB
STMDB	LDMIA

LDMIA R0!, {R1, R2, R3}
LDMIA R0!, {R1-R3}
LDMIA R0!, {R1-R3, R5}
STMIB R0!, {R1-R3}

PTO →

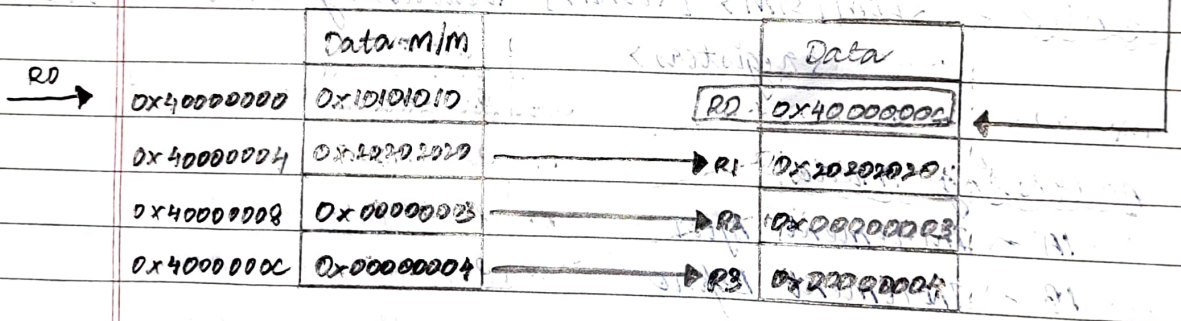
* MULTIPLE REGISTER LOAD (INCREMENT AFTER):

```
LDR R0, = 0x40000000
LDMIA R0!, {R1-R3}
```



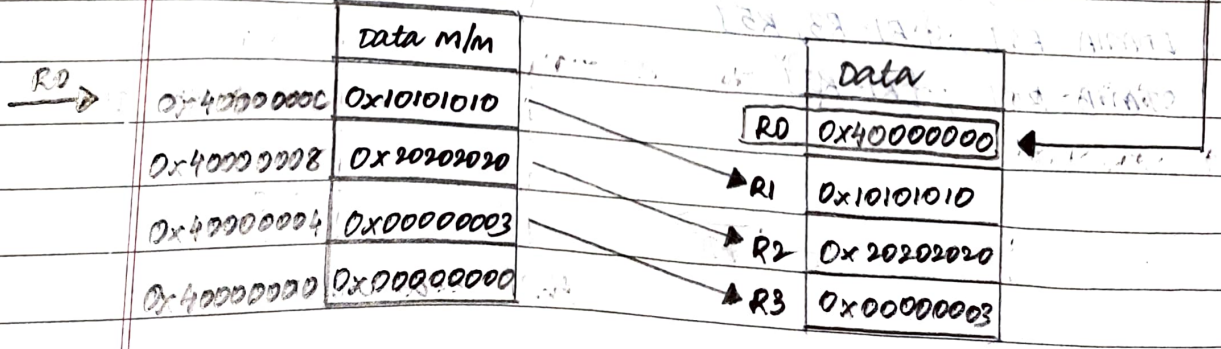
* MULTIPLE REGISTER LOAD (INCREMENT BEFORE):

```
LDR R0, = 0x40000000
LDMIB R0! {R1-R3}
```



* MULTIPLE REGISTER LOAD (DECREMENT AFTER):

```
LDR R0, = 0x4000000C
LDMDA R0! {R1-R3}
```



* MULTIPLE REGISTER LOAD (DECREMENT BEFORE):

```
LDR R0, =0x4000000C
LDMDB R0!, {R1-R3}
```

	Data M/M			Data
R0 → 0x4000000C	0x10101010		R0	0x40000000
0x40000008	0x20202020	→	R1	0x20202020
0x40000004	0x00000003	→	R2	0x00000003
0x40000000	0x00000004	→	R3	0x00000004

* MULTIPLE REGISTER STORE (DECREMENT AFTER):

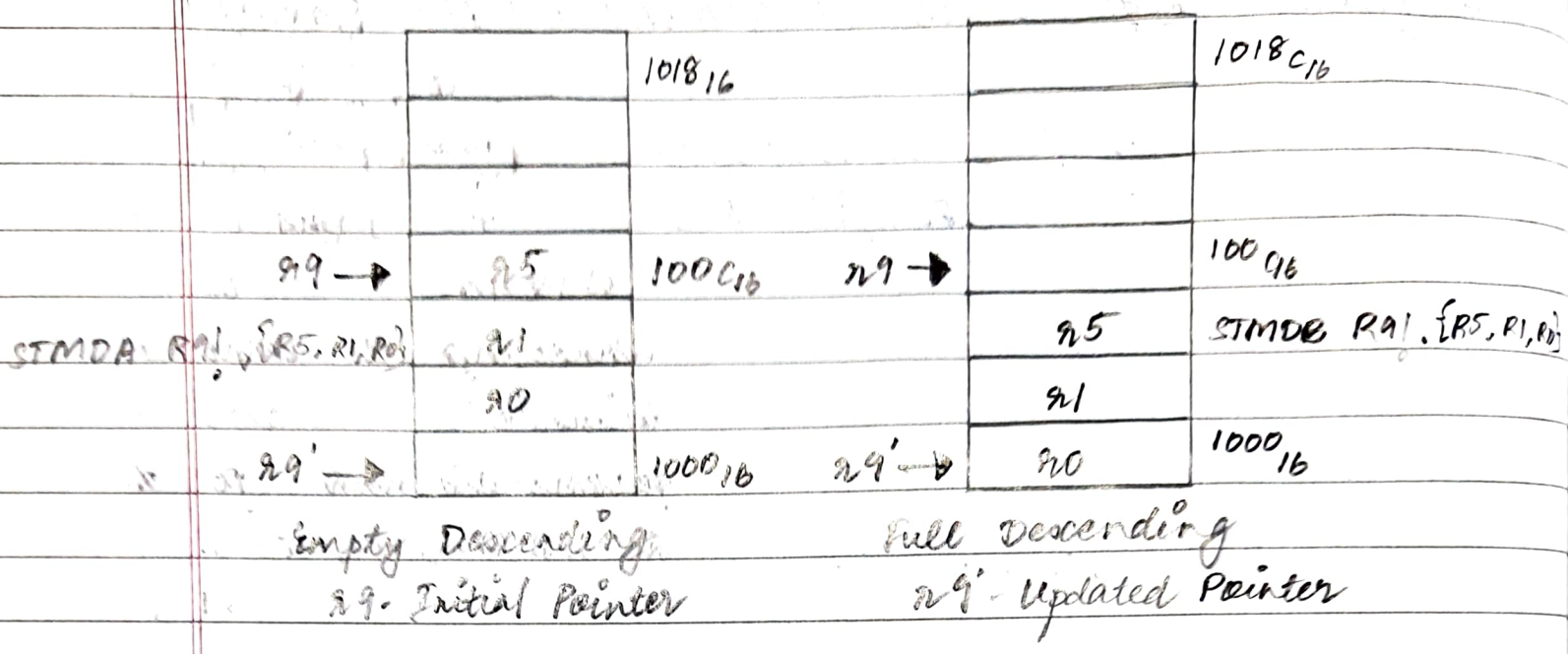
```
LDR R0, =0x4000000C
STMDA R0!, {R1-R3}
```

	Data M/M			Data
R0 → 0x4000000C	0x10101010	←	R0	0x40000000
0x40000008	0x20202020	←	R1	0x10101010
0x40000004	0x00000003	←	R2	0x20202020
0x40000000	0x00000004	←	R3	0x00000003

* TYPES OF STACK (PUSH OPERATION) :- STM
(POP OPERATION) :- LDM

R9' →		1018_{16}	R9' →	R5	$101C_{16}$
	R5			R1	
	R1			R0	
R9 →	R0	$100C_{16}$	R9 →		1000_{16}
STMIA R9!, {R0, R1, R5}					STMIA R9!, {R0, R1, R5}
		1000_{16}			1000_{16}

Empty Ascending — Full Ascending



* RELATION BETWEEN BLOCK TRANSFER AND STACK OPERATIONS :

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