DATE: ☐☐☐☐☐☐

Multiple - Register Transfer

LDMIA — Load multiple stack Registers

STM — Store multiple Register

LDMFD — load multiple full descending
r13 ! r0 - r5 , pop from a full descending stack.

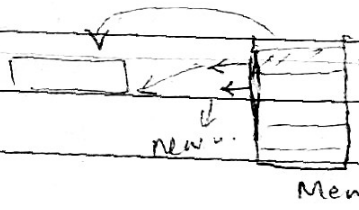
STMFD — Store multiple full descending
STMFD r13! , {r0 - r5} ; Push into full descending stack

LDMFA — Load multiple full Ascending

STMFA — Store multiple full ascending.

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

LDM - Load multiple register ;



{Rd} * N ← mem 32 [Start-address
+4*N]

STM - Save multiple register ;

{Rd} * N → mem 32
{Start-Address + 4*N]

BA = 0x14 to R0.

Rn (!) → Write back operation

BA = 0x18

Base address

DATE: 28 04 2020

Block Data Transfer ldm ①

List of registers in { }.

* Load - store multiple instructions, can transfer multiple registers between memory and the processor in a single instruction

* Multiple-register transfer instructions are more efficient from single-register transfers for moving Blocks of Data. (more than 4-bytes of memory)

* ARM implementation do not usually interrupt-instructions while they are executing.

Read or

Text Area

Data area.

5x4 = 20 bytes.

labels ref to memory.

R0 = Int A address.

DATE: ☐☐☐☐☐☐

Example for LDMIA

Here LDMIA r0!, {r1 - r3}

Load Multiple Instructions from memory to the register

① Here data will be initially available in memory location.

② This data will be copied from the memory location to the registers specified in the instruction

③ Since this is 32 bit operation, each data size would be of 4 bytes.

DDefines words each with 4bytes alignment

Memory locati → X DCD 0X 1111 11 11

2byte 1byte

Y DCD 0X 22 22 22 22

Z DCD 0X 33 33 33 33

DATE: ☐☐ ☐☐ ☐☐☐

Example for LDMIA

Here LMDIA r0!, {r1 - r3}

Load Multiple Instructions from memory to the register

① Here data will be initially available in memory location.

② This data will be copied from the memory location to the registers specified in the instruction

③ Since this is 32 bit operation, each data size would be of 4 bytes.

(Defines words each with 4byte alignment.

memory locati → x → DCD 0X 11 11 11 11

2byte 1byte

y DCD 0X 22 22 22 22

Z DCD 0X 33 33 33 33

DATE: □□/□□/□□□□

Here r0 → base address (r0 will be pointing to the base address)

'!' → write back operation

After execution of the instruction

Say if r0 is pointing to a memory location X

ie r0 = X

X → is at mem 32 (0X80018)

after the transfer of each 32 bit-size data the memory location of r0 will be incremented by 4bytes.

In this example

After transfer is 3 × 4 byte.

= 12 byte.

The memory location r0 will be pointing to is

= 0X 80018 + 12

and the register contents will be

r3 = 0X1111 1111

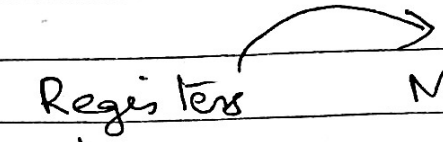
r2 = 0X2222 2222

r3 = 0X 33 33 33 33

DATE: □□ □□ □□□□

Example 2: STMIB and LDMA

STMIB → Store multiple registers to memory location



32 bit-size mem 32

→ Before ~~After~~ each transfer the memory location will be incremented by 4 bytes 4 bytes of data

Syntax: ~~XO~~

STMIB r0! {r1 - r3}

DATE: □□ □□ □□□□

Stack operation

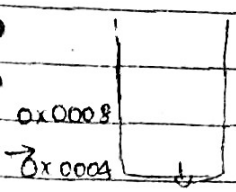
→ Pop operation uses a multiple load instruction

→ PUSH operation uses multiple store instructions

→ SP points to empty location

→ E → Empty stack (to check if stack is empty)

F → Full stack → sp points to last-used or full



Stack would be in Ascending 'A' → stack grows towards higher order memory

Decending 'D' → stack decrements, and moves towards lower order memory

to check if stack is full
- STMFD → Store multiple registers full decending
- STMFA → Store multiple registers full Ascending

to check empty
- STMEA → Store multiple registers empty Ascending
- STMED → Store multiple registers empty Decending

DATE: ☐☐/☐☐/☐☐☐☐

Example:

Before | STMFD SP! {r1, r4}

r1 = 0x 00000002

r4 = 0x 00000003

SP =

After execution?

Similarly

STMFD r13! {r2-r9} → Saves registers onto a stack

LDMFD r13! {r2-r9} → restore from stack

DATE: ☐☐☐☐☐☐

Example:

fore | STMFD SP! {r1, r4}

r1 = 0x 00000002

r4 = 0x 00000003

SP =

ter execution?

Similarly

STMFD r13! {r2-r9} → Saves registers onto the stack

LDMFD r13! {r2-r9} → restore from stack

DATE: ☐☐☐☐☐☐

SWAP Instruction : Swap a byte between memory and a register.

① It is a special load store instruction

② It swaps the contents of memory with contents of registers.

③ It reads and writes in the same bus operation

④ It prevents other instructions from reading or writing to that location until it completes (Atomic operation)

Syntax:

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

                 [Rn] → Rm ,

                 Rm → Rn
```

* temp = mem8[Rn]

mem8[Rn] = Rm

Rd = temp

DATE: ☐☐☐☐☐☐☐☐

Write an ARM program to execute the below instruction

r0 = 0x00000000

r1 = 0x1111 2222

r2 = 0x0000 9000

SWP r0, r1, [r2]

After execution ?