Multiple data transfer instructions

ARM also supports multiple loads and stores:

General syntax:

\[
\text{op}<\text{address-mode} \{\text{cond}\}< <\text{rn}\>{!}, <\text{register-list}\>{^}\n\]

- \text{op} : \text{ldm}, \text{stm}
- \text{address-mode}:
  - \text{ia} – \text{Increment address after each transfer}
  - \text{ib} – \text{Increment address before each transfer.}
  - \text{da} – \text{Decrement address after each transfer}
  - \text{db} – \text{Decrement address before each transfer}
  - \text{fd} – \text{full descending stack}
  - \text{ed} – \text{empty descending stack}
  - \text{fa} – \text{full ascending stack}
  - \text{ea} – \text{empty ascending stack}.
Multiple data transfer instructions

• **cond** is an optional condition code

• **rn** is the *base register* containing the **initial memory address** for the transfer.

• **!** is an optional suffix.
  - If ! is present, the final address is written back into **rn**.
  - If the base register is in the register-list, then you must not use the writeback option.
Multiple data transfer instructions

**reg-list**

- a list of registers to be loaded or stored.
- can be a comma-separated list or an rx-ry range.
- may contain any or all of r0 - r15
- the registers are **always** loaded in order regardless to how the registers are ordered in the list.
- for both the ldm and stm instructions, reg-list must not contain the sp
- for ldm, reg-list must not contain the PC if it contains the lr
- for stm, reg-list must not contain the lr if it contains the pc
Multiple data transfer instructions

- `^` is an optional suffix. Do NOT use it in User mode or System mode.
  - forces processor to transfer the saved program status register (SPSR) into the current program status register (CPSR) at the same time, saving us an instruction
  - if `op` is LDM and `register-list` contains the pc, the CPSR is restored from the SPSR
  - otherwise, data is transferred into or out of the User mode registers instead of the current mode registers.
Multiple data transfer instructions

Example of ldmia – load, increment after

 ldmia r9, {r0-r3} @ register 9 holds the base address. “ia” says increment the base addr after each value has been loaded from memory
Multiple data transfer instructions

Example of ldmia – load, increment after

\[
\text{ldmia } \text{r9, } \{\text{r0-r3}\} \quad @ \text{ register } 9 \text{ holds the }
\]
\[
\quad @ \text{ base address}
\]

This has the same effect as four separate ldr instructions, or

\[
\text{ldr } \text{r0, [r9]}
\]
\[
\text{ldr } \text{r1, [r9, #4]}
\]
\[
\text{ldr } \text{r2, [r9, #8]}
\]
\[
\text{ldr } \text{r3, [r9, #12]}
\]

Note: at the end of the ldmia instruction, register r9 has not been changed. If you wanted to change r9, you could simply use

\[
\text{ldmia } \text{r9!, } \{\text{r0-r3, r12}\}
\]
Multiple register data transfer instructions

**ldmia – Example 2**

\[ \text{ldmia } r9, \{r0-r3, r12\} \]

- Load words addressed by r9 into r0, r1, r2, r3, and r12.
- Increment r9 after each load.

**Example 3**

\[ \text{ldmia } r9, \{r5, r3, r0-r2, r14\} \]

- Load words addressed by r9 into registers r0, r1, r2, r3, r5, and r14.
- Increment r9 after each load.
- \text{ldmib, ldmda, ldmdb} work similar to \text{ldmia}
- Stores work in an analogous manner to load instructions.
PUSH and POP

**Note:**
push is a synonym for **stmdb sp!**, reg-list
pop is a synonym for **ldmia sp!**, reg-list

**Note:**
ldmfd is a synonym for **ldmia**
stmfd is a synonym for **stmdb**
Multiple register data transfer instructions

Common usage of multiple data transfer instructions

- **Stack**
  - Function calls
  - Context switches
  - Exception handlers
Multiple register data transfer instructions

Stack

- When making nested subroutine calls, we need to store the current state of the processor.
- The multiple data transfer instructions provide a mechanism for storing state on the *runtime stack* (pointed to by the stack pointer, r13 or sp)

stack addressing:
  - stacks can *ascend* or *descend* memory
  - stacks can be *full* or *empty*
  - ARM multiple register transfers support all forms of the stack
Multiple register data transfer instructions

Stack
- Ascending stack: grows up
- Descending stack: grows down

A stack pointer (sp) holds the address of the current top of the stack

Full stack: sp is pointing to the last valid data item pushed onto the stack

Empty stack: sp is pointing to the vacant slot where the next data item will be placed
Multiple register data transfer instructions

Stack Processing

ARM support for all four forms of stacks

- **Full ascending (FA):** grows up; stack pointer points to the highest address containing a valid data item
- **Empty ascending (EA):** grows up; stack pointer points to the first empty location
- **Full descending (FD):** grows down; stack pointer points to the lowest address containing a valid data item
- **Empty descending (ED):** grows down; stack pointer points to the first empty location below the stack
Load and Store Multiples

LDMxx r10, {r0,r1,r4}
STMxx r10, {r0,r1,r4}

Base Register (Rb)
Stack -- Last in first out memory

- Multiple store / load
  - stmed
  - ldmed

### Stack example

<table>
<thead>
<tr>
<th>Address (H)</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>4000 0488</td>
<td>:</td>
</tr>
<tr>
<td>4000 0008</td>
<td>:</td>
</tr>
<tr>
<td>4000 0004</td>
<td>:</td>
</tr>
<tr>
<td>4000 0000</td>
<td>:</td>
</tr>
</tbody>
</table>
Stack push operation: stmed

Store multiple empty descending instruction

subr1:

\[
\text{stmed } r13!, \{r0-r2, r14\} \quad @ \text{push work & link registers}
\]

@ stores data on stack and decreases r13

“Empty” means Stack Pointer is pointing to an empty location
Stack pop operation: ldmed

Load multiple empty descending

ldmed    r13!, {r0-r2, r14}  @ pop work & link registers
    @ restores data to registers
    @ and increases r13

STMEDr13!, {r0-r2, r14}
when return from SUB1
New  r13'
    (r14)
    (r2)
    (r1)
    (r0)
LDMED    r13!, {r0-r2, r14}
old  r13
    (r0)
New  r13'
    (r14)
    (r2)
    (r1)
    (r0)
Stack push operation: stmfd

STMFD r13!, {r4-r7} – Push R4, R5, R6 and R7 onto the stack.
Stack pop operation: ldmed

LDMFD r13!, {r4-r7} – Pop R4,R5,R6 and R7 from the stack.