## 数据块传输指令(LDM, STM)

数据块传输指令用来装载或者存储当前可见寄存器堆的子集。该类指令支持所有的堆栈操作模式，维护满栈或者空栈（可以在存储器中扩大或者缩小），而且该类指令对于有效的保护或者恢复现场，以及主存移动大块的数据给予了很好的支持

**指令格式**

**LDM/STM**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 1 | 0 | 0 | P | U | S | W | L | Rn | | | | | 10bit\_high\_RegistList | | | | | | | | | | 0 | 0 | H | 6bit\_low\_RegistList | | | | | |

### 寄存器列表

该指令可能引起当前寄存器体中任何寄存器的传输（非用户模式的程序可以对用户模式的寄存器进行传输）。在指令编码中，寄存器列表占16位，由高10位寄存器列表和低6位寄存器列表拼接而成，每一位对应一个寄存器。UniCore32微处理器有32个通用寄存器，块传输指令一次只能传递R0-R15（H = 0）或R16-R31（H = 1）中的寄存器。

当H = 0时，如果第0位为1，将引起R0被传输；如果为0，R0不会被传输；同样，第1位控制R1，依此类推。当H = 1时，如果第0位为1，将引起R16被传输；如果为0，R16不会被传输；如果第15位为1，将引起R31（PC）被传输。

可以指定寄存器堆的任何子集或者全集，唯一的约束是寄存器列表不能为空。

如果需要将R31寄存器存储到存储器中，实际保存到存储器中的R31的内容为STM指令的地址加8。

注意，汇编器不支持基址回写，并且基址在寄存器列表中的指令形式。

### 寻址模式

传输的地址由基址寄存器（Rn）的内容，P位（先/后）以及U位（加/减）来决定。寄存器按照从低到高的顺序传输，所以R31（如果在寄存器列表中）总是在最后传输。最低的寄存器对应存储器的低地址。下面我们举例说明，假设需要传输寄存器R1，R5和R7，基址寄存器Rn＝0x1000，并且W＝1（基址需要写回）。图 3‑6：后加寻址。图 3‑7：先加寻址。图 3‑8：后减寻址。图 3‑9：先减寻址。在各图中，示意了寄存器传输的顺序，地址的计算以及指令执行结束以后Rn的数值。

在所有的情况下，如果W＝0（不需要将改变的基址寄存器写回到寄存器堆），那么Rn将保持0x1000不变，除非Rn出现在LDM指令的寄存器列表中，这样Rn将会被装入的数据修改。

注意，如果在特权模式下需要对用户模式下的地址空间进行块数据的传输，使用隐式的方法来表示T位。即在特权模式下， 如果P＝0，W＝1时，系统访问的是用户模式下的地址空间。例如指令：LDM &Rn++, {R1，R2，…}。

### 地址对齐

地址应该为字对齐的，但是非字对齐的地址并不影响指令的执行。然而，地址的最低2位将出现在A[1:0]，并且存储器系统会做出不同的解释。

R1

R5

R1

R7

R5

R1

Rn

0x100c

0x1000

0x0FF4

0x100c

0x1000

0x0FF4

0x100c

0x1000

0x0FF4

0x100c

0x1000

0x0FF4

1

2

3

4

Rn

图 3‑7 后加寻址

R1

R5

R1

R7

R5

R1

Rn

0x100c

0x1000

0x0FF4

0x100c

0x1000

0x0FF4

0x100c

0x1000

0x0FF4

0x100c

0x1000

0x0FF4

1

2

3

4

Rn

图 3‑7 先加寻址

Rn

0x100c

0x1000

0x0FF4

0x100c

0x1000

0x0FF4

0x100c

0x1000

0x0FF4

0x100c

0x1000

0x0FF4

1

2

3

4

Rn

R7

R5

R1

R1

R5

R1

图 3‑8 后减寻址

Rn

0x100c

0x1000

0x0FF4

0x100c

0x1000

0x0FF4

0x100c

0x1000

0x0FF4

0x100c

0x1000

0x0FF4

1

2

3

4

Rn

R7

R5

R1

R1

R5

R1

图 3‑9 先减寻址

### S位的应用

当在LDM/STM指令中设置S位时，它的含义依赖于指令的类型以及R31是否在寄存器列表中。只有在特权模式下执行指令时，才可以设置S位。

LDM指令，R31在寄存器列表中，设置S位（改变模式）

如果指令为LDM指令，那么当R31被装入数据时，将SMSR\_<mode>传输到CMSR中。本指令不进行用户寄存器体的传输，而是进行当前模式下的寄存器传输。

STM指令，R31在寄存器列表中，设置S位（用户寄存器体传输）

需要进行传输的寄存器来自用户模式的寄存器体，而不是当前模式下的寄存器体。当进行进程切换时，该类指令可以用来保存用户状态。当采用这种机制时，不允许写回基址寄存器。

R31不在寄存器列表中，设置S位（用户寄存器体传输）

对于LDM和STM指令，寄存器列表中的寄存器为用户模式下的寄存器而不是当前处理器模式下的寄存器体。当进行进程切换时，该类指令可以用来保存用户状态。当采用这种机制时，不允许写回基址寄存器。

### 使用R31作为基址寄存器

对于LDM和STM指令，不能使用R31作为基址寄存器。

### 寄存器列表中包含基址寄存器

总是覆盖回写的基址内容。

当执行的指令为LSM指令，如果需要基址写回，并且基址出现在寄存器列表中，会产生以下的情况：

在STM指令中，如果基址为寄存器列表中第一个或第二个寄存器，那么存储的数据为基址的原值。

例如：

STM &Rn++, {Rn, ….} 或 STM &Rn, {Rm, Rn, …}。

在STM指令中，如果基址为寄存器列表中第三个以及以后的寄存器，那么存储的数据为基址的新值。

例如：

STM &Rn++, { Rl, Rm, … , Rn, …}。

在LDM指令中，如果基址为寄存器列表中的第一个寄存器，那么基址寄存器中的数据为的新的基址。

注意，UniCore32的汇编器不支持基址回写，并且基址在寄存器列表中的指令形式。

例如：

LDM &Rn++, {Rn, …}

在LDM指令中，如果基址为寄存器列表中的第二个以及以后的寄存器，那么基址寄存器中的数据为从存储器中装入的数据。

例如：

LDM &Rn++, {Rm, Rn, …} 或者LDM &Rn++,{Rm, … , Rn, …}

### 数据失效

存储器管理系统可能不接受一些有效的地址，存储管理部件可以通过置DABORT信号有效表示地址出现问题。这种情况可能出现在多寄存器传输指令中寄存器列表的任意位置，如果UniCore32应用在一个虚拟存储系统中，那么必须能够恢复这种状况。

STM指令中的失效

如果在STM指令的执行过程中发生失效，UniCore32立即结束指令的执行，随之进入数据失效例外。存储管理部件负责阻止对存储器错误的写操作。对于处理器内部状态的唯一改变是基址的回写。如果基址寄存器已经被改变，处理器会自动恢复基址寄存器的值。

LDM指令中的失效

如果UniCore32在LDM指令的执行过程中发现了数据失效，那么它改变指令的操作以确保可以正确的恢复处理器的状态。

1. 当失效发生时，写寄存器的操作停止。失效后，不会再装入寄存器数据了，但是先前的寄存器可能已经被改变了。PC是寄存器列表中最后一个，所以不会被改变。
2. 如果基址需要写回，那么当发生数据失效时，处理器自动地恢复基址寄存器。这样确保了如果基址寄存器在寄存器列表中，并且在失效发生之前改寄存器被改变的情况下可以正确的恢复。

### 例外

数据失效。

### 汇编语法

<LDRM|STRM> [++|--][&]Rn[++|--], <Rlist,{&c}>

Rn 一个有效的寄存器号或者表达式

<Rlist> 寄存器列表 (例如， {R0,R2-R7,R10})。

[&] 表示基址寄存器是否需要写回

{&c}表示是否设置S位，即是否与PC寄存器一起装入CMSR，或者在特权模式下传输用户模式下的寄存器。

### 示例

LDM &SP++, {R0,R1,R2} ；3个寄存器出栈

STM R0++, {R16-R31} ；保存所有的高端寄存器

LDMFD &SP++, {R31} ；R31 <- (SP)，不改变CMSR

LDMFD &SP++,{R31,&c} ；R31 <- (SP), 恢复CMSR，只能发生在特权模式

STM --R13,{R0-R14,&c} ；用户模式的寄存器入栈，只能发生在特权模式

；这些指令在子程序入口处可以用来保存状态

；并在返回时恢复状态

STM &SP--,{R16-R19,R30} ；R16-R19入栈，R30用来返回，嵌套调用，可能会更改R30

LDM ++&SP,{R16-R19,R31} ；恢复寄存器并且返回

注：如果以上指令是在特权态下使用，那么凡是使得P＝0，W＝1的指令访问的都是用户态下的地址空间。

### 操作描述

LDRM [++|--][&]Rn[++|--], <Rlist>

Address = start\_address

If (H = 0)

For i = 0 to 15

If registe\_list[i] = 1 then

Ri = Memory(address, 4)

address = address + 4

Else //H = 1

For i = 16 to 30

If register\_list[i-16] = 1 then

Ri = Memory(address, 4)

Address = address +4

If register\_list[15] = 1 then

Valued = Memory[address ,4]

Pc = value and 0xFFFFFFFC

Address = address +4

LDRM [++|--][&]Rn[++|--], <Rlist\_without\_pc,&c>

Address = start\_address

If (H = 0)

For i = 0 to 15

If register\_list[i] = 1

Ri\_user = Memory[address, 4]

Address = address +4

Else //H = 1

For i = 16 to 30

If register\_list[i-16] = 1

Ri\_user = Memory[address, 4]

Address = address + 4

LDRM [++|--][&]Rn[++|--], <Rlist\_with\_pc,&c>

Address = start\_address

For I = 16 to 30

If register\_list[i-16] = 1 then

Ri = Memory[address ,4]

address = address +4

CMSR = SMSR

Value = Memory[address ,4]

Pc = value and FFFFFFFC

Address = address +4

STRM [++|--][&]Rn[++|--], <Rlist>

address = start\_address

if H = 0 then

for i = 0 to 15

if register\_list[i] = 1 then

Memory[address ,4] = Ri

Address = address + 4

Else // H = 1

For I = 16 to 31

If register\_list[i-16] = 1 then

Memory[address ,4] = Ri

Address = address + 4

STRM [++|--][&]Rn[++|--], <Rlist,&c>

address = start\_address

if H = 0 then

for i = 0 to 15

if register\_list[i] = 1 then

Memory[address ,4] = Ri\_usr

Address = address + 4

Else // H = 1

For I = 16 to 31

If register\_list[i-16] = 1 then

Memory[address ,4] = Ri\_usr

Address = address + 4