ARM Load/Store Instructions

- The ARM is a Load/Store Architecture:
  - Only load and store instructions can access memory
  - Does not support memory to memory data processing operations.
  - Must move data values into registers before using them.

ARM Load/Store Instructions

- ARM has three sets of instructions which interact with main memory. These are:
  - Single register data transfer (LDR/STR)
  - Block data transfer (LDM/STM)
  - Single Data Swap (SWP)

ARM Load/Store Instructions

- The basic load and store instructions are:

LDR	STR	Word
LDRB	STRB	Byte
LDRH	STRH	Halfword
LDRSB		Signed byte load
LDRSH		Signed halfword load

ARM Load/Store Instructions

- Memory system must support all access sizes
- Syntax:
  - LDR{<cond>}{<size>} Rd, <address>
  - STR{<cond>}{<size>} Rd, <address>

  e.g.

```
LDR R0, [R1]
STR R0, [R1]
LDREQB  R0, [R1]
```

Data Transfer: Memory to Register (load)

- To transfer a word of data, we need to specify two things:
  - Register: r0-r15
  - Memory address: more difficult
    - Think of memory as a single one-dimensional array, so we can address it simply by supplying a pointer to a memory address.
    - There are times when we will want to offset from this pointer.

ARM Addressing Modes

There are basically two types of addressing modes available in ARM

- Pre-indexex addressing: the address generated is used immediately
- Post-indexex addressing: the address generated later replaces the base register

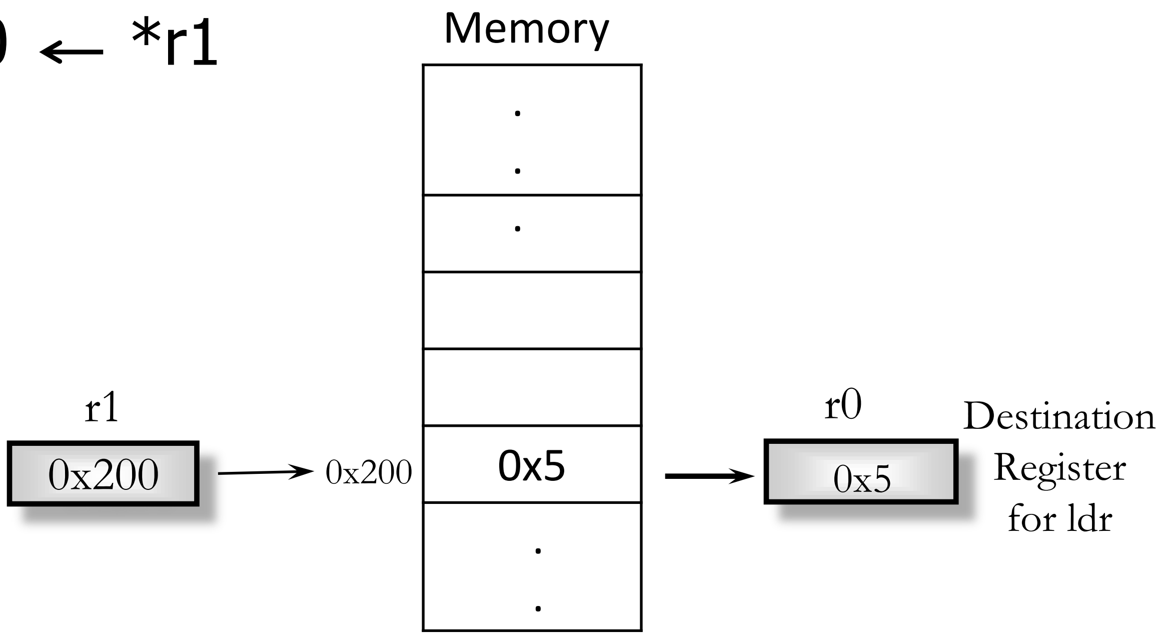
ARM Addressing Modes

[Rn] *Register*

Address accessed is value found in Rn.

Example:

```
ldr r0, [r1]      @  r0 ← *r1
```



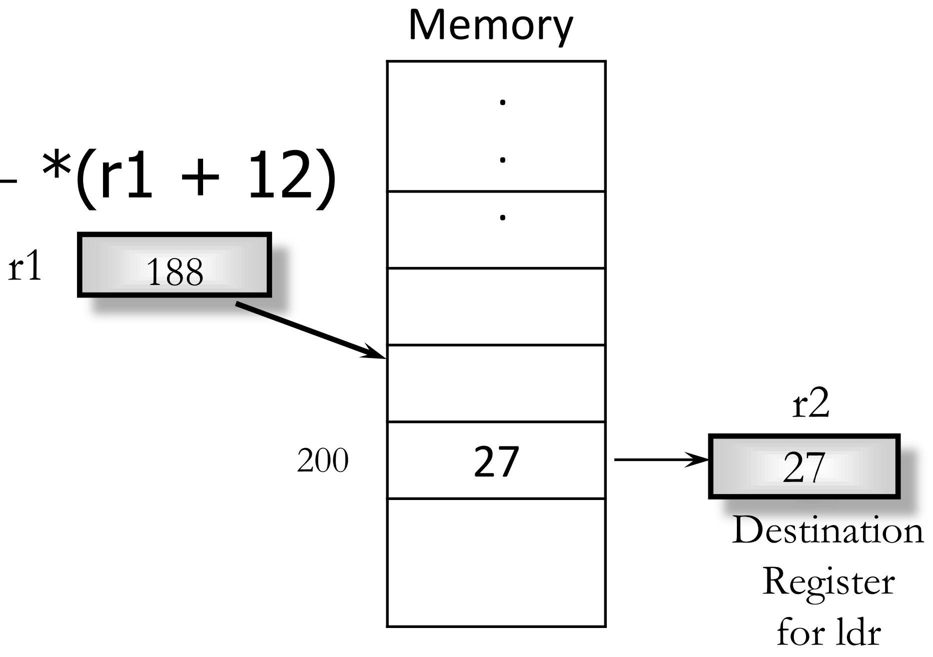
ARM Addressing Modes

[Rn, #±imm] *Immediate offset*

Address accessed is *imm* more/less than the address found in R*n*. R*n* does not change.

Example:

```
ldr r2, [r1, #12]     @  r2 ← *(r1 + 12)
```



ARM Addressing Modes

[Rn, ±Rm] *Register offset*

Address accessed is the value in R*n* ± the value in R*m*. R*n* and R*m* do not change values.

Example:

```
    ldr r2, [r0, r1]     @  r2 ← *(r0 + r1)
```

ARM Addressing Modes

[Rn, ±Rm, shift] *Scaled register offset*

Address accessed is the value in Rn ± the value in Rm shifted as specified. Rn and Rm do not change values.

Example:

```
    ldr r0, [r1, r2, lsl #2]        @  r0 ← *(r1 + r2*4)
```

ARM Addressing Modes

[R*n*, #±*imm*]!	*Immediate pre-indexed* Address accessed is as with *immediate offset* mode, but R*n*'s value updates to become the address accessed.

Example:

ldr r2, **[**r1, #12**]**! @ r1 ← r1 + 12 then r2 ← *r1

ARM Addressing Modes

[R*n*, ±R*m*]!	*Register pre-indexed* Address accessed is as with *register offset* mode, but R*n*'s value updates to become the address accessed.

Example:

```
ldr r2, [r0, r1]!        @  r0 ← r0 + r1   then r2 ← *r0
```

ARM Addressing Modes

[R*n*, ±R*m*, *shift*]!	*Scaled register pre-indexed* Address accessed is as with *scaled register offset* mode, but R*n*'s value updates to become the address accessed.

Example:

```
ldr r2, [r0, r1, lsl #2]!   @  r0 ← r0 + r1*4   then r2 ← *r0
```

ARM Addressing Modes

[R*n*], #±*imm* *Immediate post-indexed*

Address accessed is value found in R*n*, and then R*n*'s value is increased/decreased by *imm*.

Example:

```
    str r2, [r1], +4        @  *r1 ← r2   then   r1 ← r1 + 4
```

ARM Addressing Modes

[R*n*], ±R*m* *Register post-indexed*

Address accessed is value found in R*n*, and then R*n*'s value is increased/decreased by R*m*.

Example:

```
    str r0, [r1], r2      @  *r1 ← r0   then   r1 ← r1 + r2
```

ARM Addressing Modes

[R*n*], ±R*m*, *shift* *Scaled register post-indexed*

Address accessed is value found in R*n*, and then R*n*'s value is increased/decreased by R*m* shifted according to *shift*.

Example:

```
ldr r0, [r1], r2, lsl #3   @  r0 ← *r1   then   r1 ← r1 + r2*8
```

Examples of pre- and post- indexed addressing

```
str r3, [r0, r4, lsl #3]        @   pre-indexed
ldr r5, [r0, r1, lsl #3]!       @  pre indexed with writeback @ pre-indexed
ldr r0, [r1, #-8]               @  pre-indexed with negative offset
ldr r0, [r1, -r2, lsl #2]       @  negative offset shifted
ldrb r5, [r1]                   @  load byte from ea <r1>
ldrsh r5, [r3]                  @  load signed halfword from ea <r3>
ldrsb r5, [r3, #0xc1]           @  load signed byte from ea <r3+193>

str r7, [r0], #4                @  store r7 to ea<r0>, then add #24 to r0
ldr r2, [r0], r4, lsl #2        @  load r2 from ea<r0>, then add r4*4 to r0
ldrh r3, [r5], #2               @  load halfword to r3 from ea<r5>,  then
                                @  add #2 to r5

strh r2, [r5], #8               @  store halfword from r2 to ea<r5>, then
                                @  add 8 to r5
```