ARM Assembler

Addressing Modes

⟨op1⟩: Data Addressing Mode

Used when processing data

- Moving data from one register to another:
  DESTINATION ⟵ SOURCE
  DESTINATION must be a register
  SOURCE can be any ⟨*op1*⟩ value

```
MOV   r0, r1          R0 ← R1
```

- Performing an operation on data:
  DESTINATION ⟵ SOURCE_1 OPERATION SOURCE_2
  DESTINATION and SOURCE_1 must be registers
  SOURCE_2 can be any ⟨*op1*⟩ value

```
ADD   r0, r1, r2      R0 ← R0 + R2
SUB   r0, r1, #1      R0 ← R0 − 1
AND   r0, r0, r1      R0 ← R0 ∧ R1
```

Data Processing

- Four possible values for ⟨*op1*⟩:

$\#nnn$	Immediate
Rn	Register
Rn, ⟨*shift*⟩ $\#nnn$	Scaled Immediate
Rn, ⟨*shift*⟩ Rs	Scaled Register

- Possible values of ⟨*shift*⟩:

LSL	Logical Shift Left
LSR	Logical Shift Right
ASR	Arithmetic Shift Right
ROR	Rotate Right
RRX	Rotate Right eXtended

Immediate

- Also known as *literal* addressing
- Constant value incorporated into instruction:

```
MOV  r0, #12
```

$$\langle op1 \rangle \leftarrow \text{IR(value)}$$
$$\text{ALU} \leftarrow \langle op1 \rangle$$
$$\text{R}0 \leftarrow \text{ALU}$$

- Range of valid immediate values:

  0 through to 255 or 0x00 through to 0xFF
- The immediate value can be rotated right:

```
MOV  r0, #0x12, 8
```

$$\langle op1 \rangle \leftarrow \text{IR(value)} \ggg \text{IR(shift)}$$
$$\text{ALU} \leftarrow \langle op1 \rangle$$
$$\text{R}0 \leftarrow \text{ALU}$$

Will move 0x12000000 into register R0

use 16 to move 0x00120000

use 24 to move 0x00001200

Register

- Data is held in a register,
  R0 – R12, SP, LR, *or* PC, but not CPSR

```
MOV  r0, r1
```

$$\langle op1 \rangle \leftarrow \mathrm{R}1$$
$$\mathrm{ALU} \leftarrow \langle op1 \rangle$$
$$\mathrm{R}0 \leftarrow \mathrm{ALU}$$

- Accessing CPSR is a Privileged Instruction

- Register for current mode is used unless a mode is given and in a privileged mode:

```
MOV  r0, r14_usr
```

Scaled Values

- Data is held in a register
- Value is scaled according to a shift type:

LSL	Logical Shift Left	($\ll$)
LSR	Logical Shift Right	($\gg$)
ASR	Arithmetic Shift Right	($+\gg$)
ROR	Rotate Right	($\ggg$)
RRX	Rotate Right eXtended	(C $\ggg$)

- Shift value give in two ways:
  - $\Rightarrow$ **Immediate**: By a specified amount:

    `... r1, LSL #8` $\quad \langle op1 \rangle \leftarrow \mathrm{R}1 \ll \mathrm{IR(shift)}$

  - $\Rightarrow$ **Register**: Shift value stored in a register:

    `... r1, LSL r2` $\quad \langle op1 \rangle \leftarrow \mathrm{R}1 \ll \mathrm{R}2$

Shift Types (1/2)

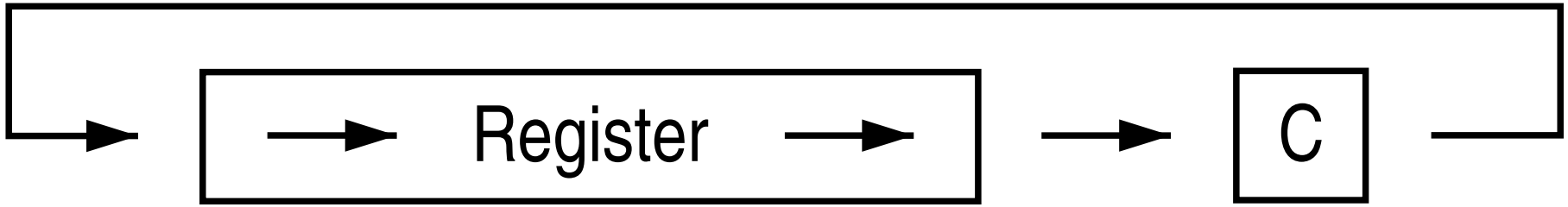
- **LSL**: Logical Shift Left ($\ll$)
  Signed or Unsigned multiply by 2^n

- **LSR**: Logical Shift Right ($\gg$)
  Unsigned divide by 2^n

- **ASR**: Arithmetic Shift Right ($+\gg$)
  Signed divide by 2^n

Shift Types (2/2)

- **ROR**: Rotate Right (⋙)



- **RRX**: Rotate Right Extended (C ⋙)
  Can only move one bit, no shift value allowed
  Used for multi-word rotates

⟨op2⟩: Memory Addressing Mode

Used when accessing memory

- Reading (Loading) data from memory:
  DESTINATION ⟵ M(SOURCE)
  DESTINATION must be a register
  SOURCE is any ⟨*op2*⟩ value

  `LDR r1, [r12]` R1 ← M(R12)

- Writing (Storing) data into memory
  M(DESTINATION) ⟵ SOURCE
  SOURCE must be a register
  DESTINATION is any ⟨*op2*⟩ value

  `STR r1, [r12]` M(R12) ← R1

  Store is the *only* ARM instruction to place the SOURCE before the DESTINATION

Memory Addressing (Syntax)

- Offset Addressing

`[R`n`, #`⟨*value*⟩`]`	Offset Immediate
`[R`n`, R`m`]`	Offset Register
`[R`n`, R`m`,` ⟨*shift*⟩ `#`⟨*value*⟩`]`	Offset scaled

- Pre-Index Addressing

`[R`n`, #`⟨*value*⟩`]!`	Pre-Index Immediate
`[R`n`, R`m`]!`	Pre-Index Register
`[R`n`, R`m`,` ⟨*shift*⟩ `#`⟨*value*⟩`]!`	Pre-Index scaled

- Post-Index Addressing

`[R`n`],` `#`⟨*value*⟩	Post-Index Immediate
`[R`n`],` `R`m	Post-Index Register
`[R`n`],` `R`m`,` ⟨*shift*⟩ `#`⟨*value*⟩	Post-Index scaled

Memory Addressing (RTL)

- Offset Addressing: `LDR R0, [R1, R2]`

  $\langle op2 \rangle \leftarrow R1 + R2$
  $MBR \leftarrow M(\langle op2 \rangle)$
  $R0 \leftarrow MBR$

- Pre-Index Addressing: `LDR R0, [R1, R2]!`

  $\langle op2 \rangle \leftarrow R1 + R2$
  $R1 \leftarrow \langle op2 \rangle$
  $MBR \leftarrow M(\langle op2 \rangle)$
  $R0 \leftarrow MBR$

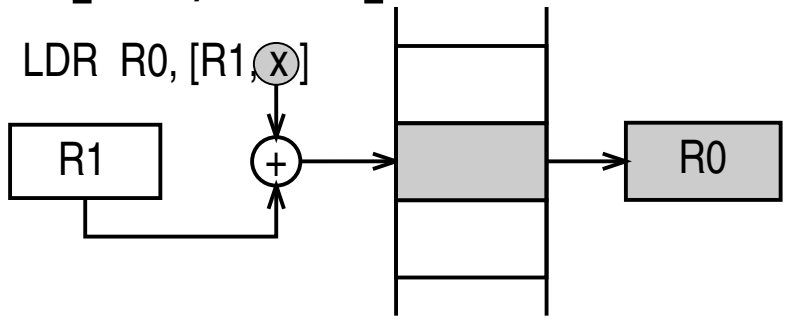
- Post-Index Addressing: `LDR R0, [R1], R2`

  $\langle op2 \rangle \leftarrow R1$
  $R1 \leftarrow R1 + R2$
  $MBR \leftarrow M(\langle op2 \rangle)$
  $R0 \leftarrow MBR$

Memory Addressing (RTL)

- Offset Addressing: `LDR R0, [R1, R2]`

  $\langle op2 \rangle \leftarrow R1 + R2$
  $MBR \leftarrow M(\langle op2 \rangle)$
  $R0 \leftarrow MBR$



- Pre-Index Addressing: `LDR R0, [R1, R2]!`

  $\langle op2 \rangle \leftarrow R1 + R2$
  $R1 \leftarrow \langle op2 \rangle$
  $MBR \leftarrow M(\langle op2 \rangle)$
  $R0 \leftarrow MBR$

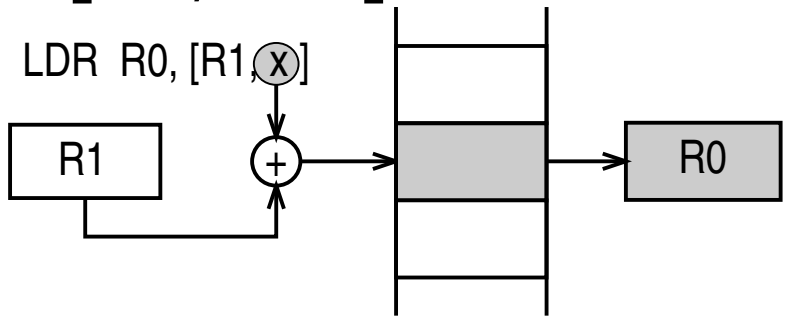
- Post-Index Addressing: `LDR R0, [R1], R2`

  $\langle op2 \rangle \leftarrow R1$
  $R1 \leftarrow R1 + R2$
  $MBR \leftarrow M(\langle op2 \rangle)$
  $R0 \leftarrow MBR$

Memory Addressing (RTL)

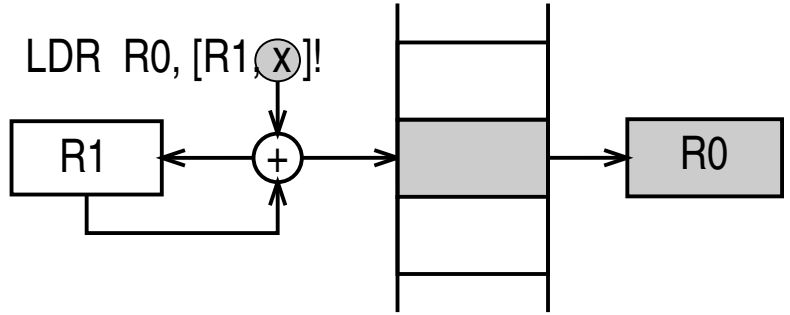
- Offset Addressing: `LDR R0, [R1, R2]`

$\langle op2 \rangle \leftarrow R1 + R2$
MBR $\leftarrow M(\langle op2 \rangle)$
R0 $\leftarrow$ MBR



- Pre-Index Addressing: `LDR R0, [R1, R2]!`

$\langle op2 \rangle \leftarrow R1 + R2$
$R1 \leftarrow \langle op2 \rangle$
MBR $\leftarrow M(\langle op2 \rangle)$
R0 $\leftarrow$ MBR



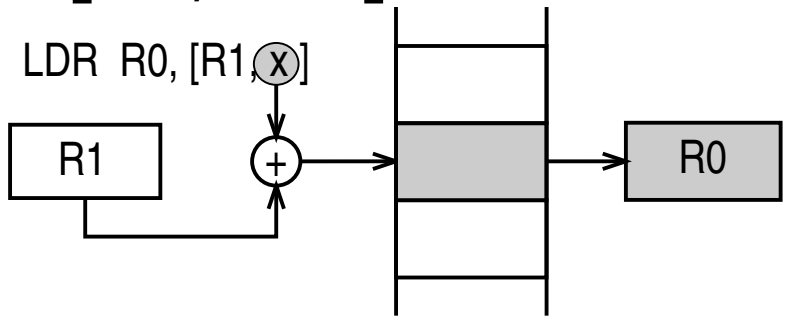
- Post-Index Addressing: `LDR R0, [R1], R2`

$\langle op2 \rangle \leftarrow R1$
$R1 \leftarrow R1 + R2$
MBR $\leftarrow M(\langle op2 \rangle)$
R0 $\leftarrow$ MBR

Memory Addressing (RTL)

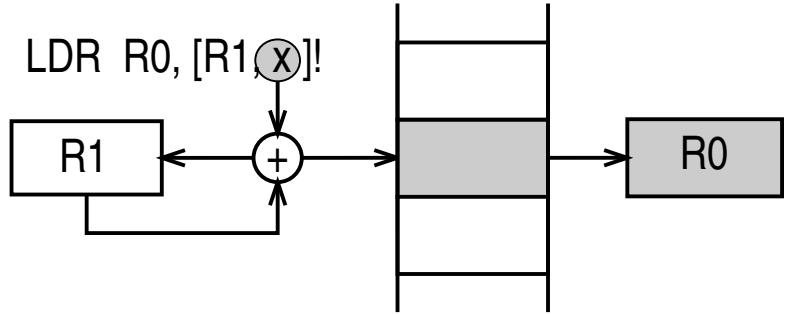
- Offset Addressing: `LDR R0, [R1, R2]`

$\langle op2 \rangle \leftarrow R1 + R2$
MBR $\leftarrow M(\langle op2 \rangle)$
R0 $\leftarrow$ MBR



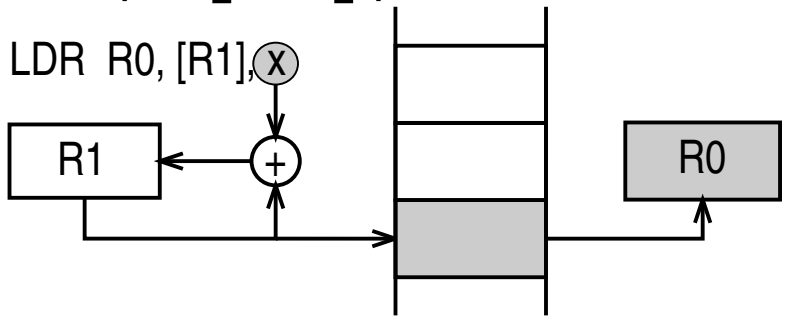
- Pre-Index Addressing: `LDR R0, [R1, R2]!`

$\langle op2 \rangle \leftarrow R1 + R2$
$R1 \leftarrow \langle op2 \rangle$
MBR $\leftarrow M(\langle op2 \rangle)$
R0 $\leftarrow$ MBR



- Post-Index Addressing: `LDR R0, [R1], R2`

$\langle op2 \rangle \leftarrow R1$
$R1 \leftarrow R1 + R2$
MBR $\leftarrow M(\langle op2 \rangle)$
R0 $\leftarrow$ MBR



Offset Addressing

[Rn, #⟨*value*⟩]	Offset Immediate
[Rn, Rm]	Offset Register
[Rn, Rm, ⟨*shift*⟩ #⟨*value*⟩]	Offset scaled

- Calculate address by adding offset to base register Rn

  Immediate: ⟨*op2*⟩ ← Rn + IR(value)

  Register: ⟨*op2*⟩ ← Rn + Rm

  Scaled: ⟨*op2*⟩ ← Rn + Rm ⟨*shift*⟩ IR(value)

- Read data from calculated memory address

  MAR ← ⟨*op2*⟩

  MBR ← M(MAR)

  ALU ← MBR

Pre-Index Addressing

[Rn, #⟨*value*⟩]!	Pre-Index Immediate
[Rn, Rm]!	Pre-Index Register
[Rn, Rm, ⟨*shift*⟩ #⟨*value*⟩]!	Pre-Index scaled

- Calculate address by adding offset to base register Rn

  Immediate: ⟨*op2*⟩ ← Rn + IR(value)

  Register: ⟨*op2*⟩ ← Rn + Rm

  Scaled: ⟨*op2*⟩ ← Rn + Rm ⟨*shift*⟩ IR(value)

- Write the address back into the base register (!)

  Rn ← ⟨*op2*⟩

- Read data from calculated memory address

  MAR ← ⟨*op2*⟩

  MBR ← M(MAR)

  ALU ← MBR

Post-Index Addressing

[Rn], #⟨*value*⟩	Post-Index Immediate
[Rn], Rm	Post-Index Register
[Rn], Rm, ⟨*shift*⟩ #⟨*value*⟩	Post-Index scaled

- Address contained in the base register Rn

  ⟨*op2*⟩ ← Rn

- Increment base register (Rn)

  Immediate: Rn ← Rn + IR(value)
  Register: Rn ← Rn + Rm
  Scaled: Rn ← Rn + Rm ⟨*shift*⟩ IR(value)

- Read data address in base register

  MAR ← ⟨*op2*⟩
  MBR ← M(MAR)
  ALU ← MBR