`MOV R0, R1, ROR #2` This will rotate the value of R*1* by two bits. The most significant bit of the resulting value will be the same as the least significant bit of the original value. The second most significant bit will be the same as the Carry flag. In the `S` version the Carry flag will be set to the second least significant bit of the original value. The value of R*1* is not changed by this operation.

`MOV R0, R1, ROR R2` Register R*0* will become the value of the register R*1* rotated to the right by the number of bits specified by the R*2* register. R*1* and R*2* are not altered by this operation.

Why is there no corresponding Rotate Left operation?

An Add With Carry (`ADC`, A.1 on page 127) to a zero value provides this service for a single bit. The designers of the instruction set believe that a Rotate Left by more than one bit would never be required, thus they have not provided a ROL function.

Rotate Right Extended

This is similar to a Rotate Right by one bit. The *extended* section of the fact that this function moves the value of the Carry (C) flag into the most significant bit of the value, and the least significant bit of the value into the Carry (C) flag. Thus it allows the Carry flag to be propagated though multi-word values, thereby allowing values larger than 32-bits to be used in calculations.

`MOV R0, R1 RRX` The register R*0* become the same as the value of the register R*1* rotated though the carry flag by one bit. The most significant bit of the value becomes the same as the current Carry flag, while the Carry flag will be the same as the least significant bit or R*1*. The value of R*1* will not be changed.

3.5.3 Memory Access Operands: ⟨*op2*⟩

The memory address used in the memory access instructions may also modified by the barrel shifter. This provides for more advanced access to memory which is particularly useful when dealing with more advanced data structures. It allows pre- and post-increment instructions that update memory pointers as a side effect of the instruction. This makes loops which pass though memory more efficient. We denote instructions of this type as taking one of its arguments from ⟨*op2*⟩. For a full discussion of the ⟨*op2*⟩ addressing mode we refer the reader to Chapter **??** on page **??**.

There are three main methods of specifying a memory address (⟨*op2*⟩), all of which include an offset value of some form. This offset can be specified in one of three ways:

Constant Value
: An immediate constant value can be provided. If no offset is specified an immediate constant value of zero is assumed.

Register
: The offset can be specified by another register. The value of the register is added to the address held in another register to form the final address.

Scaled
: The offset is specified by another register which can be scaled by one of the shift operators used for ⟨*op1*⟩. More specifically by the Logical Shift Left (LSL), Logical Shift Right (LSR), Arithmetic Shift Right (ASR), ROtate Right (ROR) or Rotate Right Extended (RRX) shift operators, where the number of bits to shift is specified as a constant value.