Stage 5: Support for shift and rotate features

ARM instruction set provides the following shift/rotate features in DP and DT instructions.

Shift/rotate in DP instructions

a) <u>With register as the source of operand2</u>

Operand2 can undergo 4 types of shift/rotate (LSL, LSR, ASR and ROR). The type of shift/rotate is specified by bits 6..5 of instruction ("00" for LSL, "01" for LSR, "10" for ASR and "11" for ROR). The amount of shift/rotate may be a contant when bit 4 of instruction = '0' or a variable when bit 4 of instruction = '1'.

a.1) Constant shift/rotate amount (e.g. add Rd, Rn, Rm, LSL #10)

An unsigned constant ranging from 0 to 31, specified in a 5 bit immediate (bits 11..7 of instruction) gives the shift/rotate amount.

a.2) Variable shift/rotate amount (e.g. add Rd, Rn, Rm, LSL Rs)

A register specified by bits 11..8 of instruction contains the shift/rotate amount.

b) <u>With immediate operand2</u>

8-bit immediate operand, zero-extended to 32 bits, can undergo ROR by an amount two times the unsigned constant specified as a 4 bit immediate (bits 11..8 of instruction).

Shift/rotate in DT instructions

a) <u>With constant address offset</u> (e.g. ldr Rd, [Rn, #100])

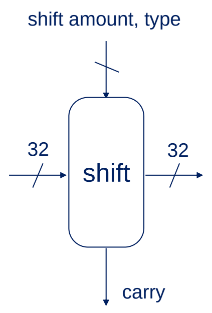
The offset is specified as a 12 bit immediate, no shift/rotate is applicable.

b) <u>With address offset provided by a register</u> (e.g. str Rd, [Rn, Rm, LSL #4])

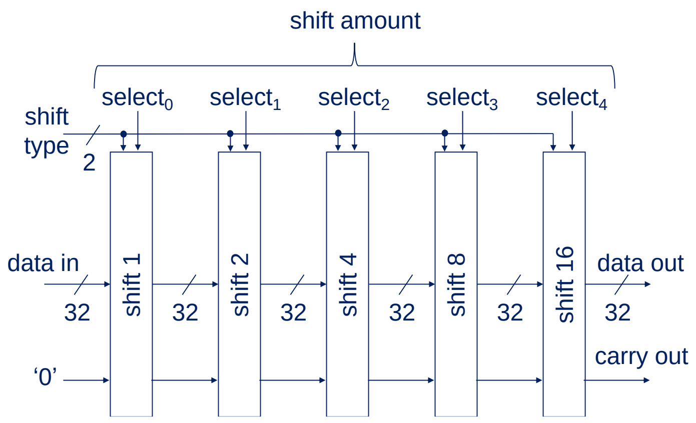
Variant (b) of DT instruction shown above was excluded in previous stages, but needs to be included now. Register and shift specification have same format as case (a) of DP instructions, with only constant shift/totate amount as in case (a.1) of DP instruction.

Design of shift/rotate unit

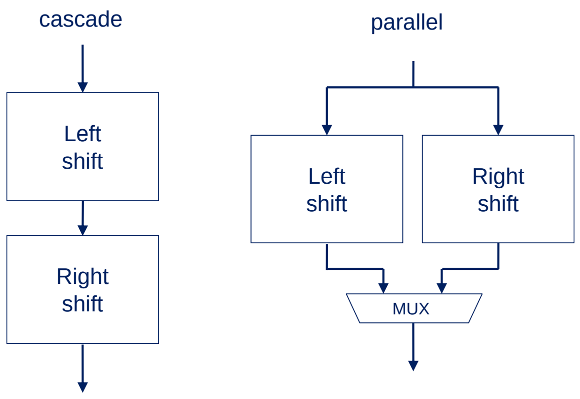
Inputs and outputs of this unit, to be designed as a separate entity-architecture pair, are shown in the adjoining figure. Selection of the input source (register or immediate), any extension required (as in case of immediate input) and source of shift amount (register or immediate) are not shown in this figure. This is a purely combinational circuit.



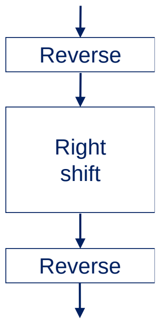
To take care of the range of shift amount from 0 to 31, we need 5 stages as shown in the figure below. Five bits of shift amount are applied as select input at different stages. Stage *i* performs shift by 2^i bits, $i = 0$ to 4.



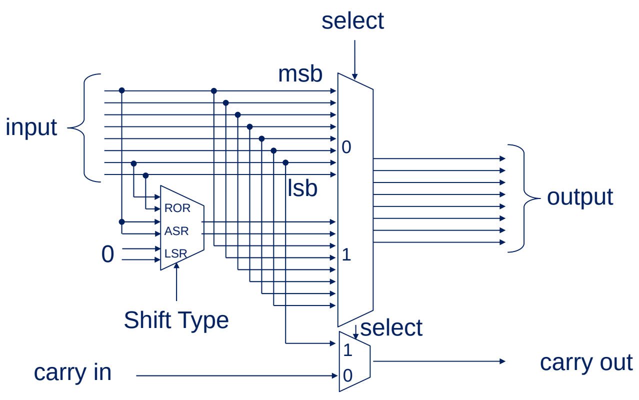
Since it is easy to integrate the 3 right shift/rotate operations (LSR, ASR and ROR), one approach to the overall design is to implement right shift/rotate operations and left shift operation (LSL) as two separate sub-units, each having its 5 stages, and combine the two together by any of the two methods shown in the adjoining figure. Here only one of the shifters should perform shift and the other should pass on data unchanged.



Another alternative is to have only a right shift/rotate unit and achieve left shift with two bit-reversal units as shown in the adjoining figure. Bit-reversal units are simply multiplexers that select either bit-reversed data (in case of left shift) or straight data (in case of right shift/rotate). Thus by doing bit-reversals and making shift type as LSR, we can perform LSL operation.



Integration of the 3 right shift/rotate operations (LSR, ASR and ROR) together into one sub-unit is illustrated for 8-bit data for stage $i = 1$ (i.e. shift amount = 2 bits) in the adjoining figure. Other stages would be similar. Note that if a stage performs a shift (select bit = '1'), the carry generated during shift is selected and if it performs no shift (select bit = '0') then the carry from the previous stage is passed on.



Integrating shift/rotate unit in overall design

In the datapath, this unit is positioned between register file and ALU. Suitable glue logic provides the necessary inputs (data, shift type and shift amount). From timing point of view, it is desirable to dedicate a clock cycle to shift/rotate operation to prevent delays of shift/rotate unit and ALU adding up. This calls for introduction of a control state in which output of shift/rotate unit is clocked into a register.

DP instructions with shift amount specified by a register (case (a.2)) require 3 registers to be read. Continuing with register file having 2 read ports and one write port, there is a need to have another control state in which the third register read is done, prior to shift/add cycle. Another case of requirement of reading 3 registers is variant (b) of DT instructions. Again, reading of register file needs to be spread over 2 cycles. Control FSM needs to be modified suitably for these changes.

It would be useful to augment the decoder component to identify different variants of operand 2 of DP instructions and offset of DT instructions and to include additional concurrent assignments in the main component to extract fields of instructions that define shift type, shift amount (immediate) and register Rs.