COL216 Computer Architecture
Lab Assignment 5

This and subsequent lab assignments involve designing hardware for processors and their subsystems. The designs are to be expressed in VHDL and then simulated and synthesized for Xilinx FPGA devices.

This assignment aims at building an ALU module for the following subset of ARM instructions. Design of a shifter and a multiplier will be done in the next assignment.

Arithmetic:	<add|sub|rsb|adc|sbc|rsc> {*cond*} {s}
Logical:	<and | orr | eor | bic> {*cond*} {s}
Test:	<cmp | cmn | teq | tst> {*cond*}
Move:	<mov | mvn> {*cond*} {s}
Branch:	<b | bl> {*cond*}
Multiply:	<mul | mla> {*cond*} {s}
Load/store:	<ldr | str> {*cond*} {b | h | sb | sh }
cond:	<EQ|NE|CS|CC|MI|PL|VS|VC|HI|LS|GE|LT|GT|LE|AL>

Instructions excluded are as follows.

Co-processor:	cdp, mcr, mrc, ldc, stc
Branch and exchange:	bx
Load/Store multiple:	ldm, stm
Software interrupt:	swi
Atomic swap:	swp
PSR transfer:	mrs, msr
Multiply long:	mull, mlal

The designs will be checked automatically using a test bench that we are preparing. This requires that your design uses the following entity declaration.

```
entity ARM_alu is
        port (
                operand1 : in std_logic_vector (31 downto 0);
                operand2 : in std_logic_vector (31 downto 0);
                result : out std_logic_vector (31 downto 0);
                instruction : in std_logic_vector (31 downto 0);
                N_in, Z_in, C_in, V_in, shifter_carry : in std_logic;
                next_N, next_Z, next_C, next_V : out std_logic;
                predicate : out std_logic
        );
end;
```

The ALU needs to be designed as a purely combinational circuit. It computes "result" as a function of "operand1" and "operand2" based on the following table. Where the operands come from and where the result goes to is not of concern in this exercise. Note that "result" is the memory address for load/store instructions and branch target for branch instructions.

Instr	ins [24-21]	result
and	0 0 0 0	Op1 AND Op2
eor	0 0 0 1	Op1 EOR Op2
sub	0 0 1 0	Op1 + NOT Op2 + 1
rsb	0 0 1 1	NOT Op1 + Op2 + 1
add	0 1 0 0	Op1 + Op2
adc	0 1 0 1	Op1 + Op2 + C
sbc	0 1 1 0	Op1 + NOT Op2 + C
rsc	0 1 1 1	NOT Op1 + Op2 + C
tst	1 0 0 0	Op1 AND Op2
teq	1 0 0 1	Op1 EOR Op2
cmp	1 0 1 0	Op1 + NOT Op2 + 1
cmn	1 0 1 1	Op1 + Op2
orr	1 1 0 0	Op1 OR Op2
mov	1 1 0 1	Op2
bic	1 1 1 0	Op1 AND NOT Op2
mvn	1 1 1 1	NOT Op2

Instr	ins [23] : U bit	result
ldr	1	Op1 + Op2
ldr	0	Op1 + NOT Op2 + 1
str	1	Op1 + Op2
str	0	Op1 + NOT Op2 + 1
b		Op1 + Op2
bl		Op1 + Op2
mul		Op2
mla		Op1 + Op2

For multiply instructions, the product is computed outside the ALU and is fed into ALU as operand2, wheras operand1 is the addend for mla instruction, but ignored for mul instruction.

ALU computes "predicate" which denotes whether the condition specified by bits 28-31 of the instruction is true or not. This is done by looking at the current values of the flags N_in, Z_in, C_in, V_in, as per the table shown. Condition code "1111" indicates an undefined instruction.

cond	ins [31-28]	Flags
EQ	0 0 0 0	Z set
NE	0 0 0 1	Z clear
CS | HS	0 0 1 0	C set | higher or same
CC | LO	0 0 1 1	C clear | lower
MI	0 1 0 0	N set
PL	0 1 0 1	N clear
VS	0 1 1 0	V set
VC	0 1 1 1	V clear
HI	1 0 0 0	C set and Z clear
LS	1 0 0 1	C clear or Z set
GE	1 0 1 0	N = V
LT	1 0 1 1	N ≠ V
GT	1 1 0 0	Z clear and (N = V)
LE	1 1 0 1	Z set or (N ≠ V)
AL	1 1 1 0	ignored

It also decides the next values of the flags based on the operandsand the result values, independent of the "S" bit and the operation performed. These values are ignored if "S" bit is '0', otherwise these are used to update the flags relevant for the current instruction. The flags are outside the ALU.

Although the ALU outputs can be considered as "don't cares" for instructions that are not implemented or are undefined, these are required to be made '0' in this exercise in order to facilitate automatic checking of the designs.

Date of announcement: 17th February, 2017