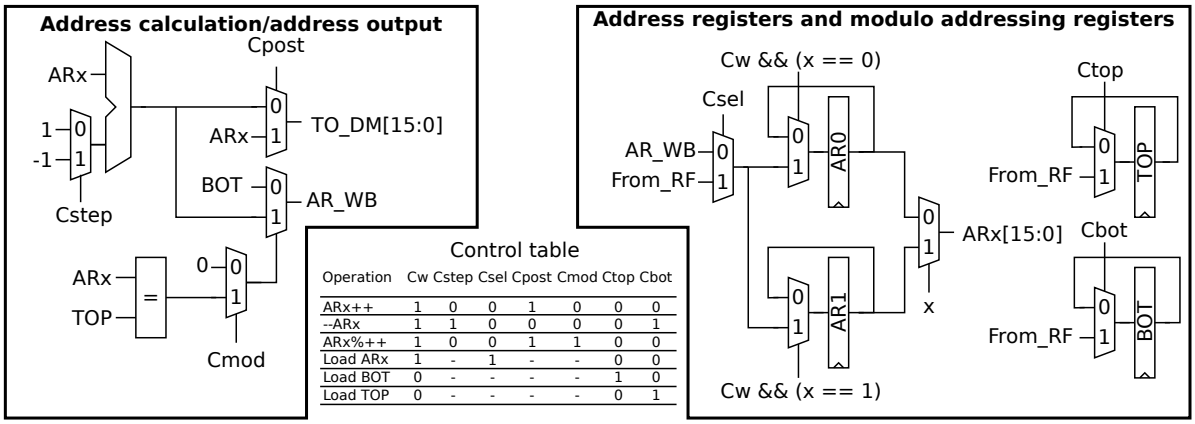
Solution proposal for the TSEA26 exam on 2012-10-26 (v1.0)

Andreas Ehliar

November 13, 2012

Solution proposal for question 1

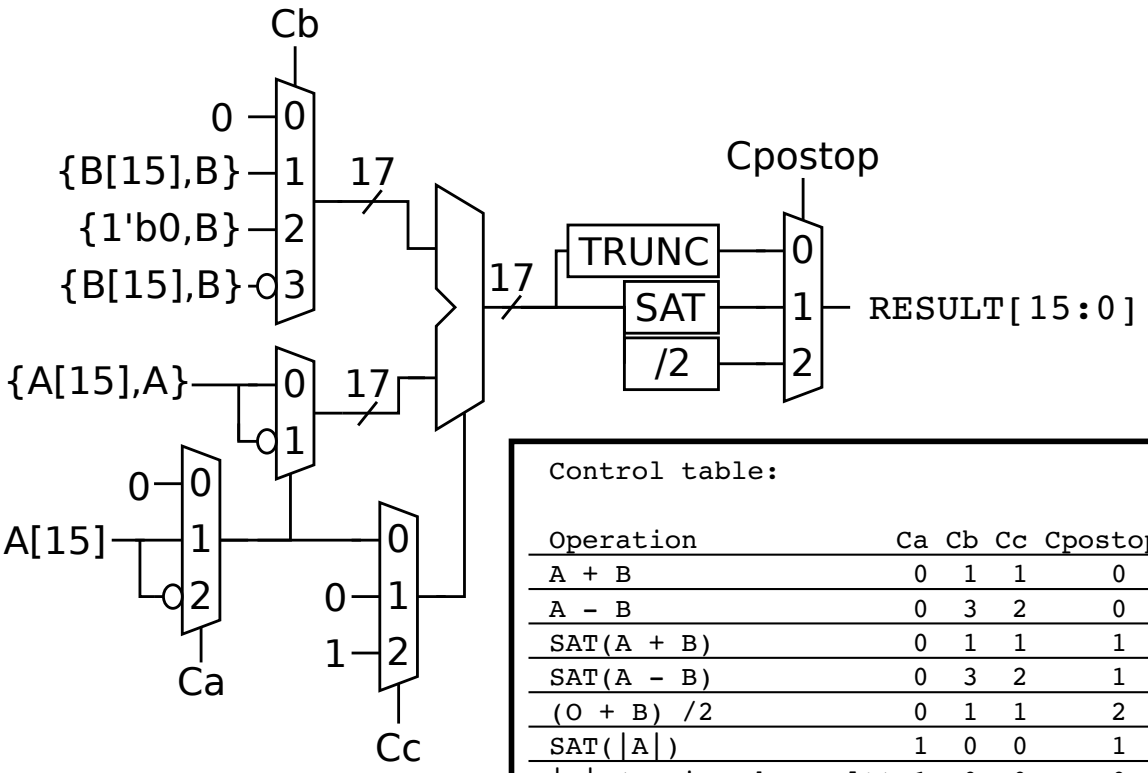


Control table

Operation	Cw	Cstep	Csel	Cpost	Cmod	Ctop	Cbot
ARx++	1	0	0	1	0	0	0
--ARx	1	1	0	0	0	0	1
ARx%++	1	0	0	1	1	0	0
Load ARx	1	-	1	-	-	0	0
Load BOT	0	-	-	-	-	1	0
Load TOP	0	-	-	-	-	0	1

NOTE: x is an input from the instruction decoder and determines whether AR0 or AR1 is used.

Solution proposal for question 2



Control table:

Operation	Ca	Cb	Cc	Cpostop
A + B	0	1	1	0
A - B	0	3	2	0
SAT(A + B)	0	1	1	1
SAT(A - B)	0	3	2	1
(O + B) /2	0	1	1	2
SAT(|A|)	1	0	0	1
|A| (unsigned result)	1	0	0	0
SAT(unsigned(B)-|A|)	2	2	0	1

```
// SAT
always @* begin
    case(out[16:15])
    2'b00: out = in[15:0];
    2'b01: out = 16'h7fff;
    2'b10: out = 16'h8000;
    2'b11: out = in[15:0];
    endcase
end
```

```
// TRUNC
assign out[15:0] = in[15:0]
// /2
assign out[15:0] = in[16:1];
```

a) $S(|A|-|B|)$ and $S(|A-B|)$ cannot be implemented in a single clock cycle when only one adder is allowed.

b) There are many ways to implement these operations, but the most straight forward way is, as hinted in the exam, to implement a 17 bit wide temporary register that can hold intermediate results. For example:

```
SAT(|A-B|):
   TMP = A - B              // New instruction 1
   RES = SAT(ABS(TMP))      // New instruction 2
SAT(|A|-|B|):
   TMP = ABS(A)             // New instruction 3
   RES = SAT(TMP - ABS(B)) // New instruction 4
```

However, it is possible to solve the problem without actually introducing a new register as implemented in the schematic above:

```
SAT(|A-B|):  // Can actually be implemented using existing instructions:
   r0 = SAT(r1-r2)  // Assume A is in r1 and B is in r2
   r0 = SAT(ABS(r0))

SAT(|A|-|B|):
   r0 = ABS(r1)  // Note that no SAT is used. However, if we assume that r0
                 // is unsigned this will work.
   r1 = SAT(unsigned(r1)-ABS(B)) // This operation assumes that r1 is unsigned
```

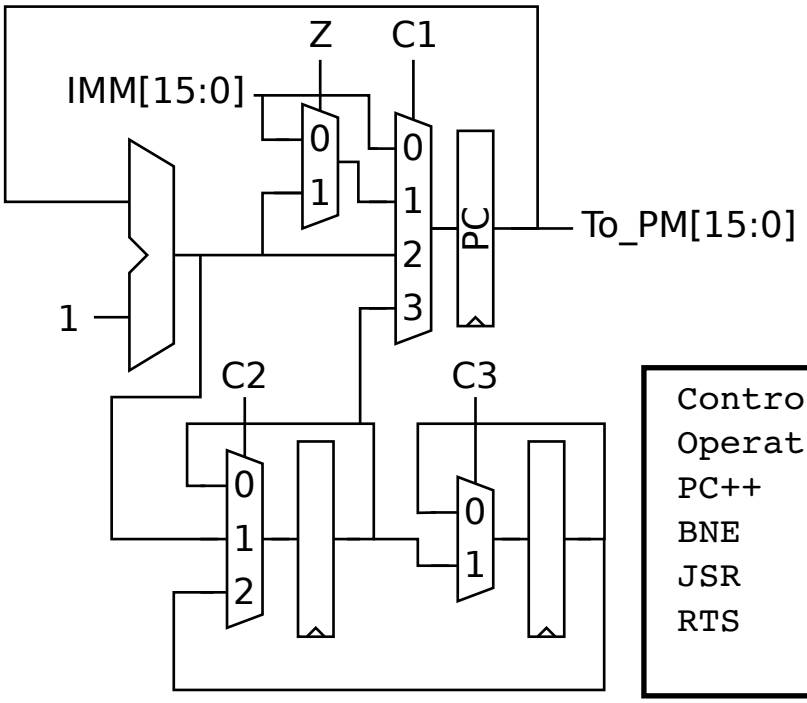
For the first variant a short motivation is in order to demonstrate that `SAT(|x|) = SAT(|SAT(x)|)`: (We assume that we saturate to a 16 bit two's complement number below.)

```
Case 1: x > 32767
   SAT(|x|) = 32767
   SAT(|SAT(x)|) = SAT(|32767|) = 32767 // Correct

Case 2: -32768 < x <= 32767
   SAT(|x|) = |x|
   SAT(|SAT(x)|) = SAT(|x|) = |x|

Case 3: x = -32768
   SAT(|x|) = SAT(|-32768|) = SAT(32768) = 32767
   SAT(|SAT(x)|) = SAT(|-32768|) = SAT(32768) = 32767

Case 4: x < -32768
   SAT(|x|) = 32767
   SAT(|SAT(x)|) = SAT(|-32768|) = SAT(32768) = 32767
```

I also saw a few variants on these two operations that I liked. For example the following is a pretty neat implementation of $S(|A| - |B|)$:

```
What we want to do:
RESULT = SAT(NEGINV(OpA) + NEGINV(OpB) + SIGN(OpA) + SIGN(OpB))

// How we do it
RESULT = SAT(NEGINV(OpA) + NEGINV(OpB) + SIGN(OpA)) // Only needs one adder as
                                                    // SIGN(OpA) is the carry input.
RESULT = SAT(RESULT + SIGN(OpB))

NEGINV: A function that inverts all bit if the number is negative

Proof: If x and y are positive the following holds:
SAT(x+y) = SAT(SAT(x)+y)
```

Solution proposal for question 3



Control table:

Operation	C1	C2	C3
PC++	2	0	0
BNE	1	0	0
JSR	0	1	1
RTS	3	2	-

Note that we should ensure that we either push PC+1 or add one to the value that we pop to ensure that we don't run an instruction close to the JSR instruction twice.

b)

```
function1:
    set r15,#32
    move ar0,r1
    move ar1,r2
    clear acr0
    add r15,#-1
loop:
    bne loop
    mac acr0, DM0[ar0++],DM1[ar1++]  // Delay slot
    add r15,#-1                      // Delay slot

    rts
    satrnd r0,acr0 // Delay slot
```

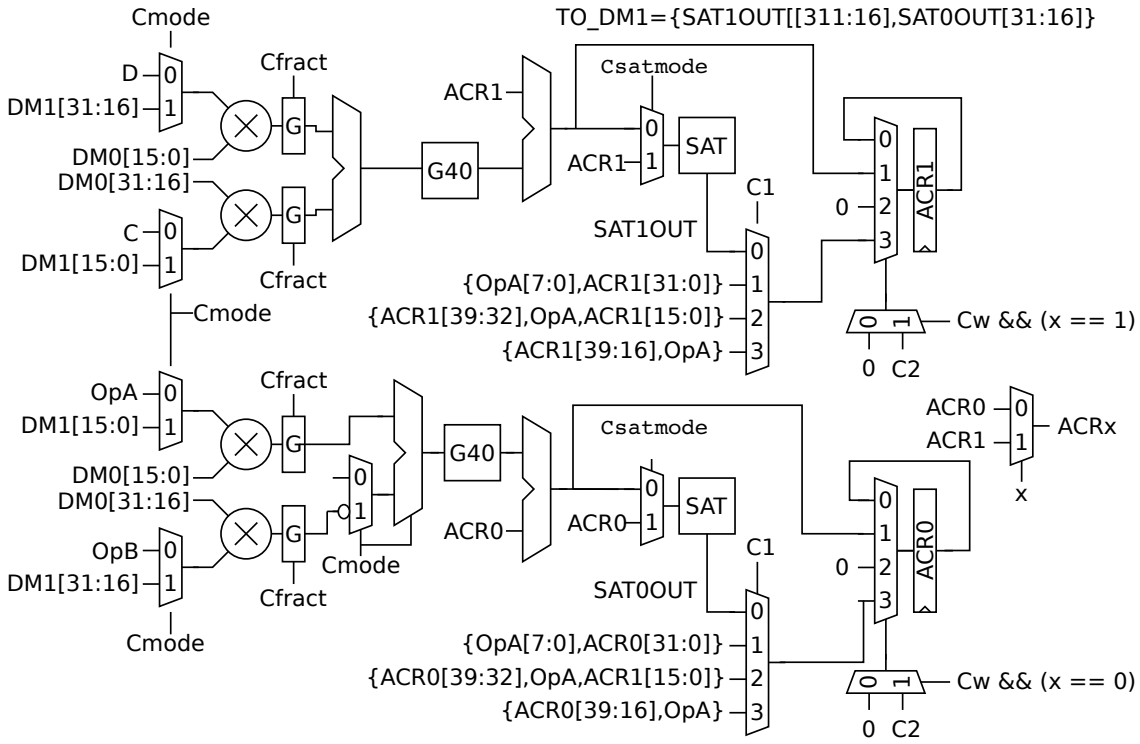
Solution proposal for question 4

To avoid any doubts about what the required solution should look like I included the constraint that four multipliers was acceptable since this exercise can actually be solved using only three multipliers[1].

```
cplx_dotproduct:
    move ar0, r0
    move ar1, r1
    clr  acr0
    clr  acr1
    repeat 30, endloop
    cplx_mac DM0[ar0++], DM1[ar1++]
endloop:
    sat  acr0
    sat  acr1
    move r0,HIGH(acr0)
    move r1,HIGH(acr1)
    ret  // Ignoring delay slots


matrix_x_vectors:
    move ar0,r4
    move ar1,r5
    move C,r2
    move D,r3
    clr  acr0
    clr  acr1
    move r2, #0x8000
    move LOW(acr0),r2
    move LOW(acr1),r2
    repeat 128, endloop2
    mat_x_vec r0,r1,DM0[ar0++], DM1[ar1++]
endloop2:
    ret // Ignoring delay slots
```

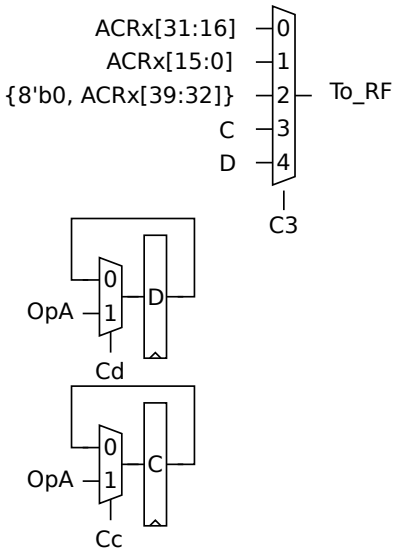
[1]This should be fairly easy for cplx_dotproduct(). However, matrix_x_vectors() is a rather more interesting challenge...

```
savestate:
    move ar0,r0 // Assume ptr points to a suitable location in DM0
    read r1,LOW(acr0)
    read r2,HIGH(acr0)
    read r3,GUARDS(acr0)
    read r4,LOW(acr1)
    read r5,HIGH(acr1)
    read r6,GUARDS(acr1)
    read r7,C
    read r8,D
    store DM0[ar0++],r1
    store DM0[ar0++],r2
       ...
    store DM0[ar0++],r8
    ret // Ignoring delay slots


restorestate:
    move ar0,r0 // Assume ptr points to a suitable location in DM0
    move r1,DM0[ar0++]
    move r2,DM0[ar0++]
       ...
    move r8,DM0[ar0++]

    set  LOW(acr0),r1
    set  HIGH(acr0),r2
    set  GUARDS(acr0),r3
    set  LOW(acr1),r4
    set  HIGH(acr1),r5
    set  GUARDS(acr1),r6
    set  C,r7
    set  D,r8
    ret // Ignoring delay slots
```




```
Control table
Operation        Cmode Cfract Csatmode C1 C2 C3 Cw Cc Cd
nop                -      -       -     -  -  -  0  0  0
clr ACRx           -      -       -     -  2  -  1  0  0
mat_x_vec          0      1       0     -  -  -  0  0  0
cplx_mult          1      0       -     -  1  -  1  0  0
sat ACRx           -      -       1     0  3  -  1  0  0

set LOW(ACRx)      -      -       -     3  3  -  1  0  0
set HIGH(ACRx)     -      -       -     2  3  -  1  0  0
set GUARD(ACRx)    -      -       -     1  3  -  1  0  0
set C              -      -       -     -  -  -  0  1  0
set D              -      -       -     -  -  -  0  0  1

read LOW(ACRx)     -      -       -     -  -  1  0  0  0
read HIGH(ACRx)    -      -       -     -  -  0  0  0  0
read GUARDS(ACRx)  -      -       -     -  -  2  0  0  0
read C             -      -       -     -  -  3  0  0  0
read D             -      -       -     -  -  4  0  0  0
```

Note: x is used to select whether ACR0 or ACR1 is used.



Solution proposal for question 5

- See the textbook.
- There are many possible answers here. One possible advantage is the reduced development time as the floating point hardware will handle all scaling typically required for fixed point computations. A possible disadvantage is of course the increased development cost.
- See the textbook.

Statistics

- Grade U: 4
- Grade 3: 8
- Grade 4: 4
- Grade 5: 4 (Best score: 45)