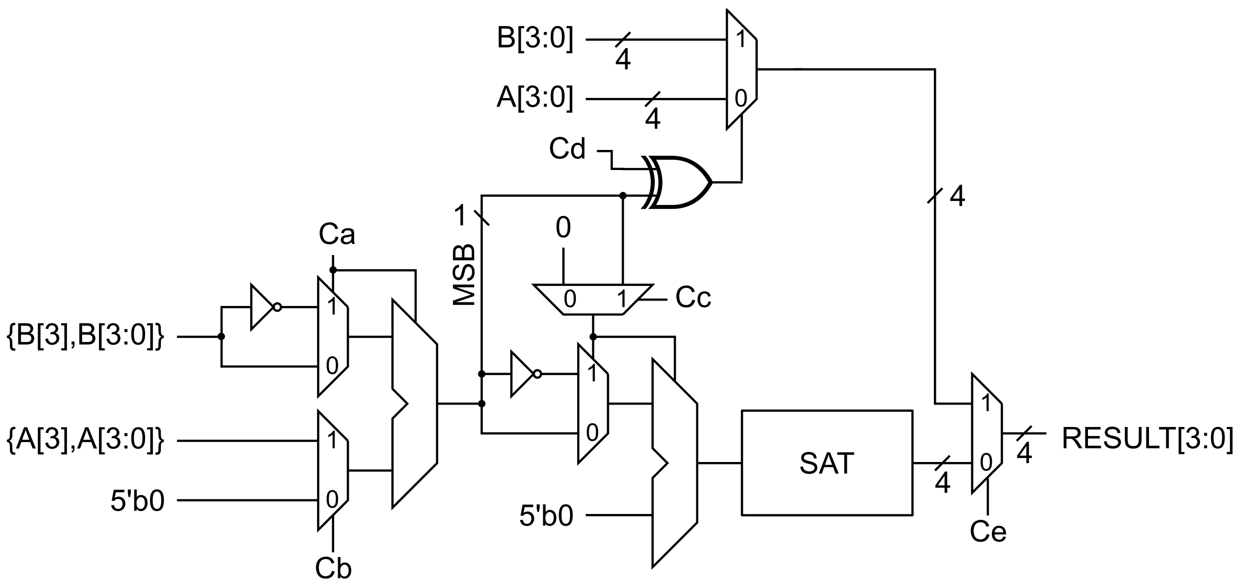
Solutions for TSEA26 exam on 2009-10-22 (v1.1)

Andreas Ehliar

October 23, 2012

Question 1

Note: All wires are 5 bits wide unless otherwise noted. All mux control signals are 1 bit wide.



Control table

Instruction	Ca	Cb	Cc	Cd	Ce
ABS(A−B)	1	1	1	x	0
MAX(A,B)	1	1	x	0	1
MIN(A,B)	1	1	x	1	1

SAT box

```
case(in[4:3])
    2'b00: out[3:0] = in[3:0];
    2'b01: out[3:0] = 4'b0111;
    2'b10: out[3:0] = 4'b1000;
    2'b11: out[3:0] = in[3:0];
endcase
```

Question 2

a) Instruction set

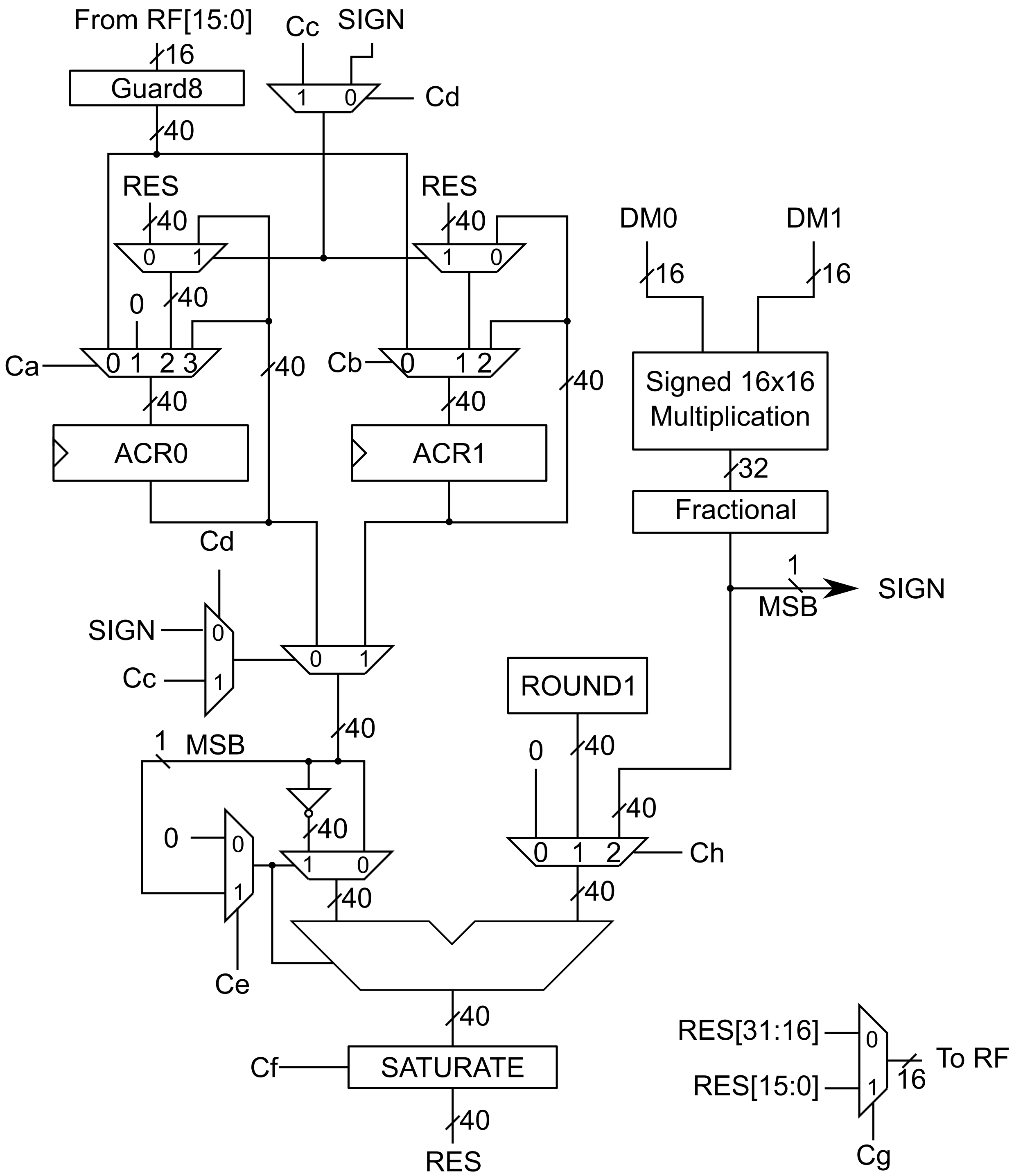
clracr0:	Clear accumulator 0
mac0:	ACR0 = ACR0 + fracmul(dm0,dm1)
sat0:	ACR0 = saturate(ACR0)
getacrh:	Rx = ACR0 (high)
getacrl:	Rx = ACR0 (low)
loadacrx:	ACRx = RF (fractional)
mac_abs:	As mac0, but select ACRx based on sign of fracmul(dm0,dm1)
abs1:	ACR1 = abs(ACR1)
getsrx:	RF = SAT(ROUND(ACRx))

b) Implementation

Note that there is no need to divide ACR0 and ACR1 into ACR0H, ACR0L, ACR1H and ACR1L here. (Although no points will be deducted if you have done this.)

Also note that we only need a 16x16 bit multiplier in this case since we know that we are only going to use fractional multiplication.

Also, we don't really need as many as 8 guard bits here since we know that we will never repeat more than 32 times, but this was not intentional from my side. (If you use less than 8 guard bits you would need to motivate this.)



	Ca	Cb	Cc	Cd	Ce	Cf	Cg	Ch		Ca	Cb	Cc	Cd	Ce	Cf	Cg	Ch
clracr0	0	2	x	x	x	x	x	x	loadacr1	3	0	x	x	x	x	x	x
mac0	2	2	0	1	0	0	x	2	mac_abs	2	1	x	0	0	0	x	2
sat0	2	2	0	1	0	1	x	0	abs1	3	1	1	1	1	0	x	0
getacrh	3	2	0	1	0	0	0	0	getsr0	3	2	0	1	0	1	0	1
getacrl	3	2	0	1	0	0	1	0	getsr1	3	2	1	1	0	1	0	1
loadacr0	0	2	x	x	x	x	x	x									

Guard8 box

```
out[39:0] = { {8{in[15]}}, in[15:0], 16'b0 };
```

Fractional box

```
out[39:0] = { {7{in[31]}}, in[31:0], 1'b0};
```

ROUND1 box

```
out[39:0] = {24'b0, 1'b1, 15'b0};
```

SATURATE box

```
if (Cf) begin
    if (in[39:31] == 9'b000000000) begin
        out = in;
    end else if (in[39:31] == 9'b111111111) begin
        out = in;
    end else if (in[39] == 1) begin
        out = 40'hff80000000;
    end else begin // in[39] == 0
        out = 40'h007fffffff;
    end
end else begin
    out = in;
end
```

Question 3

a) See the course literature

b) All inputs on a large data memory in an ASIC are registered. This means that it is more important to minimize the delay of signals connected to the output of a memory than the inputs.

c) An example of a read-after-write data hazard is when an instruction is trying to read a value written by a previous instruction which is not yet available in the register file. Two examples of how to handle this is to stall the pipeline or to use register forwarding/bypass. (Then there are Write-After-Write (WAW) and Write-After-Read (WAR) hazards, but there is no need to write an essay on this question.)

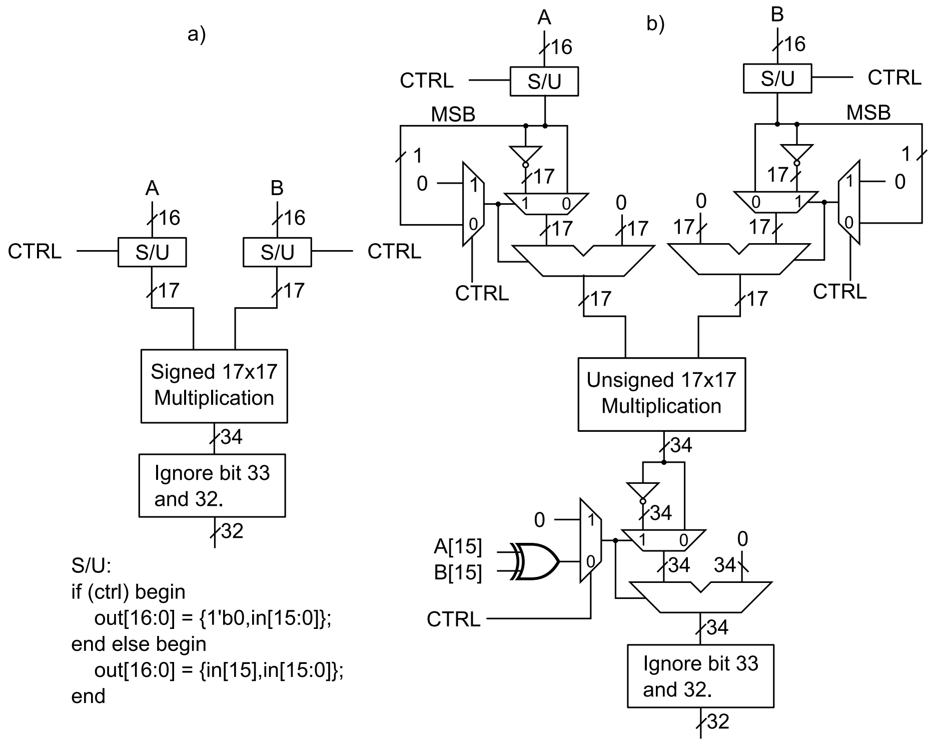
d) The worst case is $(-1 \times -1) \times 13$. To handle this case, 4 guard bits are required.

e)
```
       |11111011    (-5)
       |0000000
       |000000
      +|11011       (-5*8)
-------+--------
IGNORED|11010011    (-45)
```

f) The same hardware can be used for both fractional and integer addition without any changes.

g) When using fractional multiplication we may multiply -1 with -1 and get 1, which cannot be represented in fractional format.

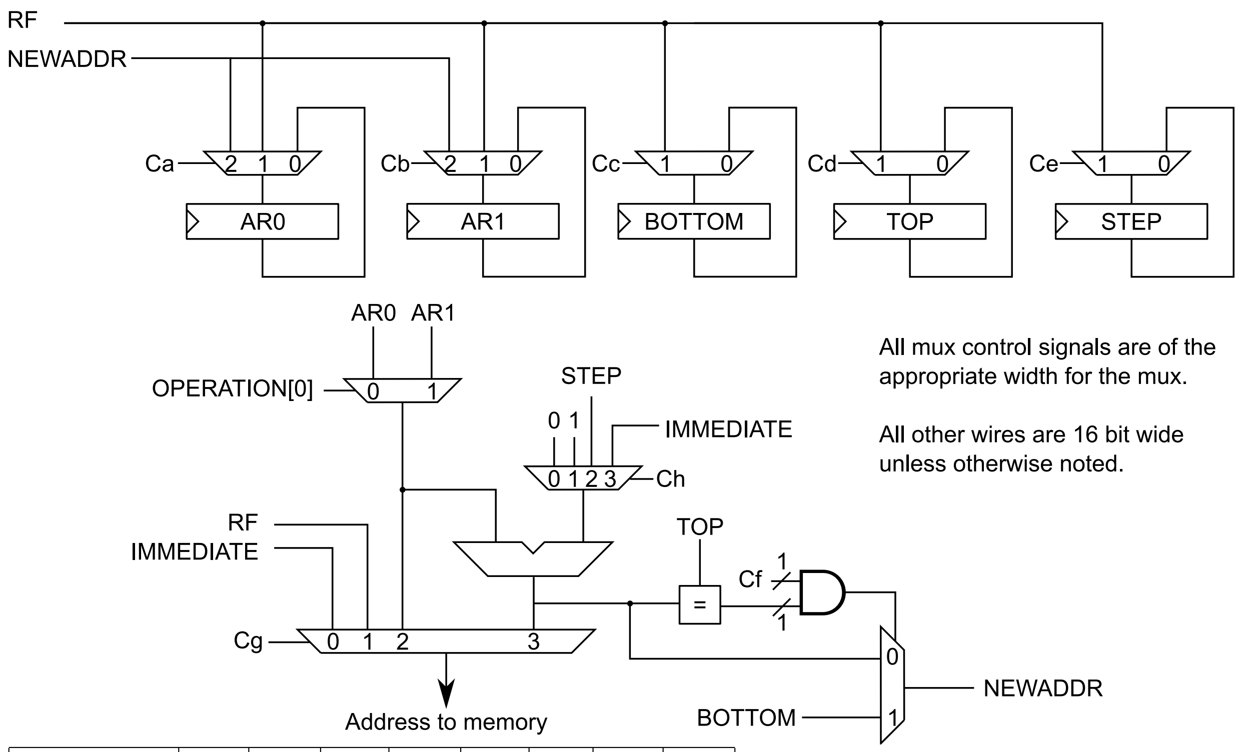
Question 4



```
S/U:
if (ctrl) begin
   out[16:0] = {1'b0,in[15:0]};
end else begin
   out[16:0] = {in[15],in[15:0]};
end
```

c) The version with the unsigned multiplier is obviously worse due to the much higher hardware cost.

Question 5


RF
NEWADDR
Ca
Cb
Cc
Cd
Ce
AR0
AR1
BOTTOM
TOP
STEP
OPERATION[0]
IMMEDIATE
Ch
Cf
Cg
=
Address to memory
All mux control signals are of the appropriate width for the mux.
All other wires are 16 bit wide unless otherwise noted.

Operation	Ca	Cb	Cc	Cd	Ce	Cf	Cg	Ch
0000	1	0	0	0	0	x	x	x
0100	0	0	0	0	0	x	0	x
0110	2	0	0	0	0	0	2	2
1000	0	0	0	0	0	0	3	3
1011	0	2	0	0	0	1	2	1

Question 6

a)

This is one possible version of pseudo assembly code for Read1bit().

```
LOAD      R0, [R15]  // CurrentAddress is in R15
LEFTSHIFT R1, 1, R14 // 1 << CurrentBit (which is in R14)
```

```
    AND       R0, R0, R1 // Memval & Bitmask

    BNE       BITWAS1
    SET       R0,1       // Set Bit = 1 in delay slot

    SET       R0,0       // Z flag was set

BITWAS1:

    ADD       R14,1,R14  // CurrentBit++
    CMP       R14,16

    BNE       NO_NEW_WORD
    NOP

    SET       R14,0      // CurrentBit = 0
    ADD       R15,1,R15  // CurrentAddress++

NO_NEW_WORD:

    RET       // Return value is in R0
```

b)

Assumptions:

- The processor has full forwarding.
- A conditional branch is predicted as not taken. If this is wrong, it costs 3 clock cycles to flush the pipeline.
- A load from memory takes 2 clock cycles to complete. The processor will stall if the program tries to use the loaded value too early.
- A return takes 3 clock cycles to complete.

Best case: The first BNE is not taken. The second BNE is not taken. The number of executed instructions are: 13. Plus 1 extra clock cycle for the load and 2 extra clock cycles for the return. The best case is 16 clock cycles.

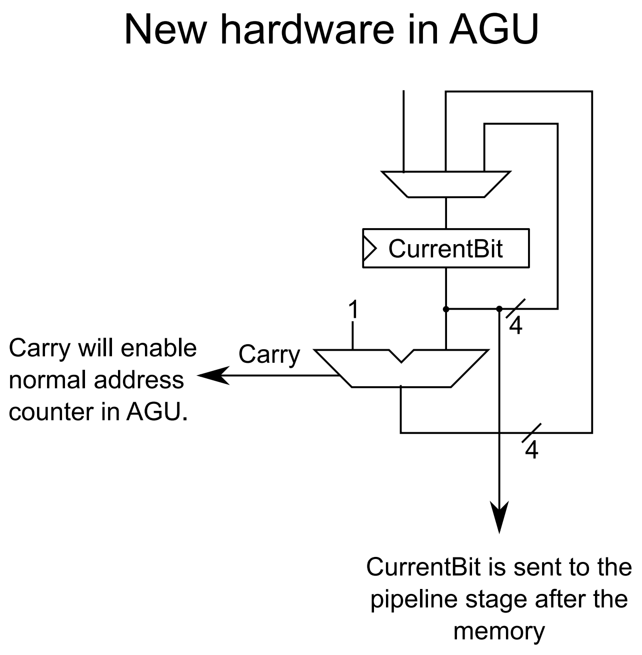
Worst case: Both BNEs are taken. The number of executed instructions are: 10. Plus 1 extra clock cycle for the load, 2 extra clock cycles for the return, and 2×3 extra clock cycles to flush the pipeline when both BNEs are wrong. The worst case is therefore 19 clock cycles.

c)

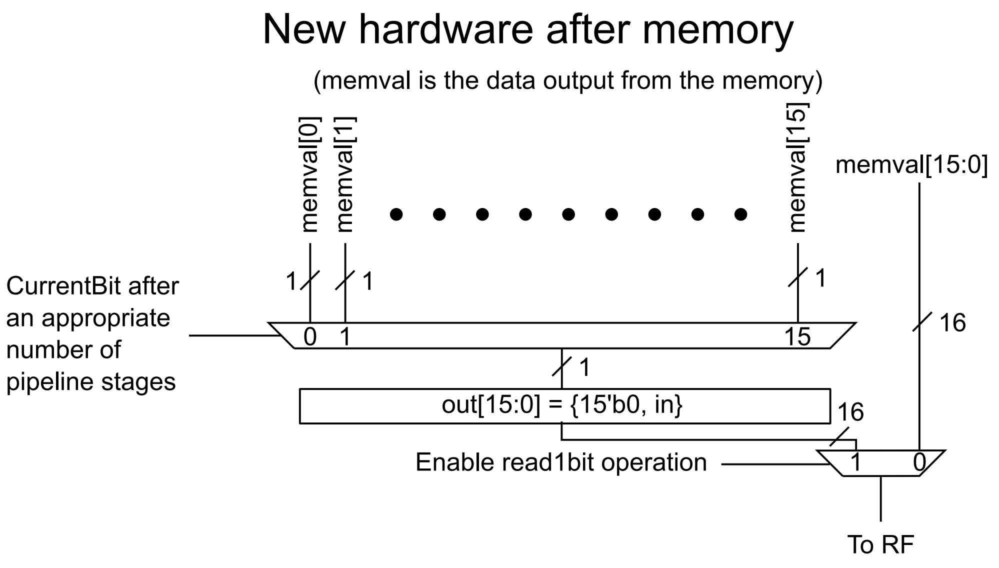
Besides the instruction decoder you would need to change the AGU and add some logic after the memory.

d)

- The AGU has to be modified to support a 4-bit counter used to index bits.
- After the memory you should insert a multiplexer to select the correct bit as indicated by the CurrentBit signal.



New hardware in AGU
CurrentBit
1
4
Carry
Carry will enable normal address counter in AGU.
4
CurrentBit is sent to the pipeline stage after the memory



New hardware after memory
(memval is the data output from the memory)
memval[0]
memval[1]
memval[15]
memval[15:0]
1
1
1
16
CurrentBit after an appropriate number of pipeline stages
0
1
15
1
out[15:0] = {15'b0, in}
16
Enable read1bit operation
1
0
To RF

Revision history

- v1.0: Initial revision
- v1.1: Fixed typo in control table for Q5 operation 1000. Cg shold be 3, not x. Thanks to Andreas Öhlin for noticing this.