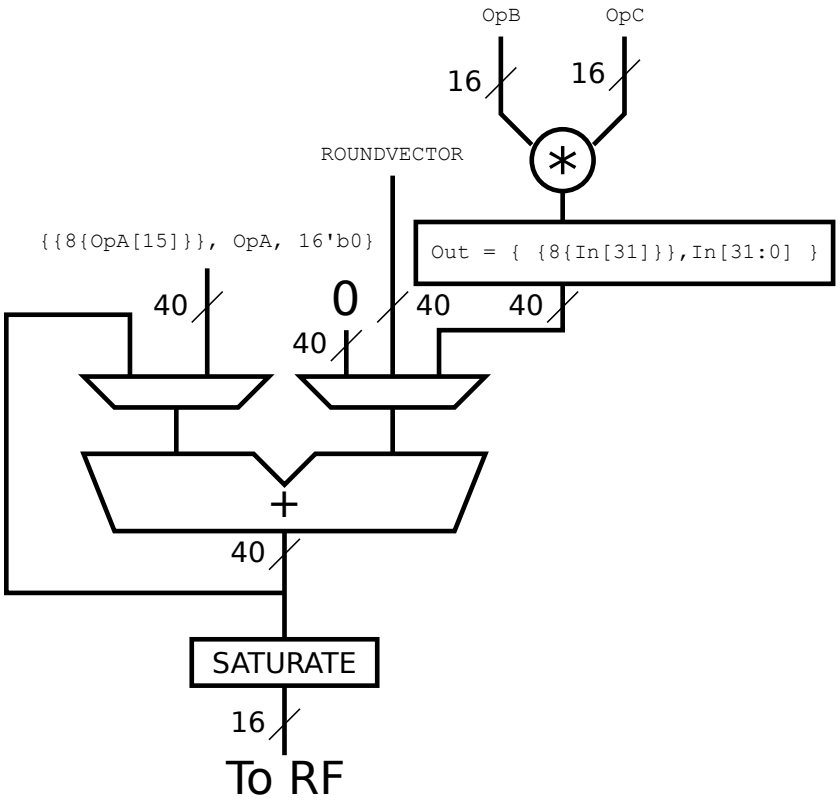
Examination
Design of Embedded DSP Processors, TSEA26

Date	2011-01-12
Room	TER4
Time	14:00-18:00
Course code	TSEA26
Exam code	TEN 1
Course name	Design of Embedded DSP Processors
Department	ISY, Department of EE
Number of questions	5
Number of pages (including this page)	8
Responsible teacher	Andreas Ehliar
Phone number during the exam time	013-178405
Visiting the exam room	Around 15 and 17
Course administrator	Ylva Jernling, 013-282648, ylva@isy.liu.se
Permitted equipment	None, besides an English dictionary
Grading	**Points** **Swedish grade** 41-50 5 31-40 4 21-30 3 0-20 U
Important information:	• Answers can be given in English or Swedish. Don't write any answers on the exam sheet. • **When designing a hardware unit you should attempt to minimize the amount of hardware used.** (Unless otherwise noted in the exercise.) A correct but not justified answer may not give full points on the question. Your number of points may also depend on how easy it is for us to understand and verify your answer. • The width of data buses and registers must be specified unless otherwise noted. Likewise, the alignment must be specified in all concatenations of signals or buses. When using a box such as *"SATURATE"* or *"ROUND"* in your schematic, you must (unless otherwise noted) describe the content of this box! (E.g. with RTL code). • All numbers are in two's complement format unless otherwise noted.

Question 1: General knowledge (3p)

a) Explain, using at most two sentences, the major problem with this MAC unit. (1p)



b) When creating the architecture of a DSP processor, it is important to know the area cost of various common components. Of the following four components, which one is most area expensive and which one is least area expensive? (All components are optimized for speed.) (1p)

- 16 bit multiplier
- 16 bit wide single port memory with an address space of 16 bits
- 16 bit 2-to-1 multiplexer
- 16 bit adder

c) Describe the most important corner case when implementing the following operation: `ABS(A[4:0])`. (A is a two's complement number.) (1p)

Question 2: Arithmetic Unit (10p)

Draw a **schematic** and a **control table** for an arithmetic unit capable of the following operations:

- OP0: No operation
- OP1: RESULT = SAT(A + B)
- OP3: RESULT = SAT(A - B)
- OP4: RESULT = SAT(ABS(A))
- OP5: RESULT = SAT(ABS(A+B))
- OP6: RESULT = SAT(ABS(A-B))
- OP7: Set S flag to 0. Other flags should remain unchanged.

In addition, the ALU should also have the following flags:

- (N) Negative flag. (Set to 1 when RESULT is negative, otherwise set to 0.)
- (Z) Zero flag. (Set to 1 when RESULT is zero, otherwise set to 0.)
- (S) Saturation flag. (Set to 1 when saturation occurs, otherwise keeps it previous value.)

Notes:

- Note: The only way for the S flag to get reset to 0 is by executing OP7.
- OP0 should not change the value of any flags.
- Result, A, and B are 8 bits wide.

Question 3: Address Generator Unit (15p)

In this exercise you should create an address generator unit. It should support the addressing modes necessary to execute the following three functions under the specified clock cycle constraints.

```
// This function must execute in less than 20 clock cycles.
// Inputs: ptr is passed in register r0
function dot13(ptr)
    ptr0 = DM0[ptr+31]
    ptr1 = DM0[ptr+33]
    resultptr = DM0[ptr+35]
    tmp = 0
    for (i = 0; i < 13; i = i + 1)
        tmp += DM0[ptr0++] * DM1[ptr1++];
    endfor
    DM0[resultptr] = SAT(ROUND(tmp))
endfunction

// This function must execute in less than 140 clock cycles
// Inputs: ptr0 is passed in register r0, ptr1 in r1
function filter(ptr0, ptr1)
    tmp = 0
    for(i=0; i < 128; i = i + 1)
        tmp += DM0[ptr0+=step] * DM1[ptr1++]
        if (ptr0 > end) then
            ptr0 = start
        endif
    endfor
endfunction

// This function must execute in less than 700 clock cycles
// Inputs: ptr is passed in register r0
function interleaver(ptr)
    ptr0 = DM0[ptr+31]
    ptr1 = DM0[ptr+33]

    for(i=0; i < 256; i = i + 1)
        idx = DM0[ptr0++]
        tmp = DM1[idx]
        // You can decide whether to use DM0 or DM1 for the
        // next line
        DMx[ptr1++] = tmp
    endfor
endfunction
```

Additional information:

- A memory address is 16 bit wide, and the memories are word-addressable
- Both DM0 and DM1 are 16 bit wide single port memories
- There are 32 general purpose registers that are 16 bit wide
- An instruction has two operands, OpA and OpB. OpA will always come from a register and OpB may come either from a register or as a 16 bit immediate from the instruction word.
- You may assume that the datapath has full forwarding. (Unless you do something in your part of the design which would prohibit forwarding in some situations.)
- One instruction can be issued (in-order) each clock cycle.
- You may assume that all other paths are implemented in such a way that the functions above can be implemented. For example, there is a repeat instruction, and the MAC instruction can load both operands directly from memories.
- The functions must contain fewer than 100 instructions.
- **If you don't answer exercise b you can still get some points for exercise c!**

a) Select the addressing modes you will need to implement these three functions under the specified clock cycle constraints. (3p)

b) Draw a schematic of your AGU. In this schematic you must also include how the memories are connected to the AGU. You may use either synchronous or asynchronous memory in this part of the exercise, but you must specify which one you are using. You should also include a control table for your AGU. (7p)

(c) Write assembly code for the `filter()` and `interleaver()` function using the addressing modes supported by your AGU. (You don't need to write assembler code for the `dot13()` function, but your hardware should be capable of supporting that function anyway under the given clock cycle constraints.) **Hint:** Be careful to avoid hazards by rearranging code or inserting NOP instructions as appropriate! (4p)

d) In exercise b you were allowed to use either synchronous or asynchronous memory. However, if you were going to create a real ASIC with a large memory (e.g. DM0 or DM1), should you use synchronous or asynchronous memories? (1p)

Question 4: Program Flow Control Instructions (10p)

During the design of a simple DSP processor, the following functions have been determined to be the most important:

```
function main()
    while(TRUE)
        // get_packet() returns the return value in register r0
        pkt_type = get_packet()
        if (pkt_type <= 0) then
            logerror()
        else if(pkt_type <= 9) then
            worker()
        else if(pkt_type >= 59) then
            guiworker()
        else
            logerror()
        end if
        update_outputs()
    endwhile
endfunction

function worker()
    optype = get_packet_operation() // Returns packet pointer in r0
    if (optype == 0) then
        DM0[95] = sum()
    else if (optype == 1) then
        DM0[96] = diff()
    else
        logerror()
    endif
endfunction


// get_current_length(), get_current_buffer() and
// get_previous_buffer() executes in 10 clock cycles each.
// They all return the return value in register r0.

// Must execute in at most 130 clock cycles
function sum()
    ptr = get_current_buffer() // ptr is returned in r0
    tmp = 0
    for (i=0; i < 100; i = i + 1)
        tmp = tmp + DM0[ptr + i]
    endfor
    return SAT(tmp)
endfunction
```

```
// Must execute in less than 50 + length*3 clock cycles.
function diff()
    tmp = 0
    length = get_current_length()
    ptr1 = get_current_buffer()
    ptr2 = get_previous_buffer()
    for (i=0; i < length; i = i + 1)
        tmp = tmp + ABS(DM0[ptr1 + i] - DM1[ptr2 + i])
    endfor
    return tmp
endfunction
```

Your task is to specify how the program flow control (PFC) unit should work.

Information:

- Unconditional branches have one delay slot
- Due to pipeline delays, conditional branches have two delay slots
- Register indirect branches have three delay slots
- The program flow control instructions may use the N, Z, and S flag. These flags are generated in the same way as in the arithmetic unit described earlier in this exam.
- There is no requirement to minimize the amount of PFC instructions or the size of the assembler program. (But you probably want to do that anyway to minimize the chance of any error.)
- The processor is a single scalar processor able to issue one instruction per clock cycle.
- If you make any assumptions to solve this exercise, you must document them.

a) Select the PFC instructions that you will require. For each instruction that you select you must describe exactly what the instruction does. (Hint: use pseudo code!)

b) Using these PFC instructions, implement the program shown above in assembler code. In addition to the PFC instructions that you selected in a), you may use any instruction commonly found in a single scalar DSP processor such as Senior without having to describe how it works. You don't have to implement the functions that are not shown above (e.g. logerror(), get_packet_operation(), etc).

Question 5: A number of standard designs (12p)

a) Create a unit that will add 5 guard bits to a 23 bit fixed point number (1p)

c) Create a register file which contains three registers (r0-r2). It should have two read ports and one write port. The registers should be 8 bit wide. (3p)

d) Create the loop controller part of a PFC unit. The loop controller should be able to repeat one instruction between 4 and 60 iterations. It does not have to support a loop with more than one instruction in it. (4p)

Your loop controller has the following inputs:

- init_loop: Initialize the iteration counter. Set to one when a repeat instruction is encountered in the program flow. Otherwise set to 0.
- num_iter: The number of iterations in the loop. Undefined when init_loop is 0. It is up to you to decide the width of this signal.

In addition, your loop controller should output the following signal:

- loop_done: Set to 0 when executing a loop and set to one on the last instruction of the iteration. Undefined when not running a loop.

e) Create a simple MAC unit with 5 guard bits capable of executing the following program:

```
function dot13(ptr)
    ptr0 = DM0[ptr+31]
    ptr1 = DM0[ptr+33]
    resultptr = DM0[ptr+35]
    ACC = 0
    for (i = 0; i < 13; i = i + 1)
        ACC += DM0[ptr0++] * DM1[ptr1++];
    endfor
    DM0[resultptr] = SAT(ROUND(ACC))
endfunction
```

Inputs are in 16 bit Q0.15 format. The output of the filter is in 16 bit Q2.13 format. You should draw a schematic of your MAC unit and a control table, but you don't need to write any assembler code however. You also don't need to show how your saturation unit works. (4p)