Examination
Design of Embedded DSP Processors, TSEA26

Date	2010-10-23
Room	R34, R35, R41, R42, R44, U6
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)	7
Responsible teacher	Andreas Ehliar
Visiting the exam room	Around 15.00 and 17.00
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. • Your number of points will depend on how easy it is for us to understand and verify your answer. A correct but not justified answer may not give full points on the question. • 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 specified.

1 Question 1: Program Counter (10p)

Design a PC module capable of the following operations:

- **OP1:** PC++
- **OP2:** PC = PC + signextend(Immediate)
- **OP3:** PC = PC
- **OP4:** PC = RF
- **OP5:** Push(PC); PC = PC + signextend(Immediate)
- **OP6:** Push(PC); PC = RF
- **OP7:** Pop(PC)
- **OP8:** if(flag) PC = PC + signextend(Immediate) else PC++
- **OP8:** if(!flag) PC = PC + signextend(Immediate) else PC++

Inputs to your PC module:

- Immediate[9:0]: Jump target offset from the instruction word
- RF[15:0]: Input from register file
- Flag: 1 bit signal from ALU
- All control signals that you specify in the control table
- Reset signal
- (And, of course, a clock signal)

Outputs from your PC module:

- PC (16 bits wide) (which is then sent to the program memory)

Other constraints:

- After the processor is reseted, the processor should start executing at address 0
- You must also have a hardware stack in your PC module capable of holding two entries.

Also, note that for this exercise you don't have to worry about the rest of the pipeline. That is, you can assume that the instruction decoder has already dealt with delay slots, setting the flag in the ALU, etc.

a) Draw a schematic of your PC module **(7p)**
b) Draw a control table for your PC module **(3p)**

2 Question 2: Address Generator (10p)

Create an AGU capable of the following addressing modes. The AGU should have one address register (AR). The address space is 16 bits. The value from the register file is 16 bits and marked as RF in the following explanation. Immediates are 10 bits. ADDR[15:0] is the address output to the memory.

The following addressing modes have been requested by the software engineers:

- **OP0:** NOP
- **OP1:** ADDR = RF + signextend(Immediate)
- **OP2:** ADDR = AR + RF
- **OP3:** AR = RF
- **OP4:** ADDR = AR + signextend(Immediate)
- **OP5:** ADDR = AR + signextend(Immediate*2)
- **OP6:** ADDR = AR + signextend(Immediate*4)
- **OP7:** ADDR = AR + Immediate; AR += Immediate
- **OP8:** ADDR = AR + Immediate * 2; AR += Immediate * 2
- **OP9:** ADDR = AR + Immediate * 4; AR += Immediate * 4
- **OP10:** Bit-reversed mode (see below)

Regarding OP10: The software engineers believe that it is crucial that you include some sort of bit-reversed addressing mode, but they don't know how to handle this efficiently in hardware.

The bit-reversed addressing mode will only be needed for a buffer that is 1024 words large. You may also implement this bit-reversed addressing mode in whatever way you want to, as long as you can use it to iterate over the entire buffer in 1024 clock cycles.

In addition, you are also allowed to place some conditions on the programmer (that is, he may have to place his bit-reversed buffer at a special address in the memory).

a) Draw a schematic for your AGU
b) Draw a control table for your AGU

3 Question 3: MAC-unit (15p)

Design a MAC unit capable of supporting the following two functions:

```
// Input data in r0-r5 are in 16-bit fractional format
// Output data in r6 and r7 should be in 16-bit fractional format
//
// Intermediate calculations are performed using a sufficient number
// of bits, so that SAT() will be able to detect an overflow.
function butterfly_part()

    r6  = SAT(ROUND(r0 * r2 - r1 * r3 - r4))
    r7  = SAT(ROUND(r0 * r3 + r1 * r2 - r5))

endfunction


// * DM0 and DM1 are 16 bit wide and contains signed integers in this
//   example.
// * sumofproducts and sumofdiff are signed integers that are "wide
//   enough" (no overflow should occur during the repeat loop)

function filter(buffer)
    AR0 = r0 // Use the instruction set AR0,r0
    AR1 = r1 // Use the instruction set AR1,r1

    sumofproducts = 0
    sumofdiff = 0

    repeat(30)
        sumofproducts += DM0[AR0] * DM1[AR1]
        sumofdiff += abs(DM0[AR0++] - DM1[AR1++])
    endrepeat

    if(sumofproducts > 0x7fffffff) then
        sumofproducts = 0x7fffffff
    else if(sumofproducts < -0x80000000) then
        sumofproducts = -0x80000000
    endif

    // Read out the 32-bit result to general purpose registers
    r2 = sumofproducts[31:16]
    r3 = sumofproducts[15:0]
    r4 = sumofdiff[31:16]
    r5 = sumofdiff[15:0]

endfunction
```

Inputs to this module:

- DM0_result[15:0], DM1_result[15:0]
- OpA[15:0], OpB[15:0]: Operands from the register file
- Control signals (created by you in your control table)
- (And, of course, a clock signal)

Outputs from this module:

- TO_RF[15:0] - This is sent to the register file writeback port

Constraints:

- The register file, DM0, and DM1 are 16 bits wide.
- It is up to you to decide how many accumulators you will need and how large they should be.
- The other parts of the processor have enough features to support these assembly programs (that is, the PC has a repeat instruction, the AGU have all relevant addressing modes, etc)
- The function butterfly_part() should be executed in at most 15 clock cycles (excluding the return instruction)
- The function filter() should be executed in at most 80 clock cycles (excluding the return instruction)

a) Select an instruction set for your MAC unit and write assembly programs for both functions. You should also decide how many accumulator registers you will need and how wide such a register should be. **(6p)**
b) Draw a schematic of your MAC unit (including a control table) **(9p)**

4 Question 4: Custom instructions (10p)

You have been tasked with accelerating a piece of code which contains a lot of **complex valued** multiplications, additions, and subtractions. A complex value is stored in a normal register by putting the imaginary part into the MSB part of the register and the real part in the LSB part of the register.

Your task is to make sure that the following piece of code should be able to execute in **at most 7 clock cycles**. All operations in this code are done on complex valued data!

```
o0 = i0 * c0 - i1
o1 = i0 + i1
o2 = i2 * c1 - i3
o3 = i2 + i3
```

Constraints:

- You need to be able to execute the excerpt listed above in at most 7 clock cycles.
- You can assume that all operands will be present in the register file. You can also assume that all results should be written back to the register file.
- You don't have to worry about overflows. (The code is written in such a way that overflows cannot occur.)

Inputs:

- **OpA[31:16]:** Imaginary part of the first operand from the register file
- **OpA[15:0]:** Real part of the first operand from the register file
- **OpB[31:16]:** Imaginary part of the second operand from the register file
- **OpB[15:0]:** Real part of the second operand from the register file
- Control signals (decided by you in the control table)

Outputs:

- TO_RF[31:16]: Imaginary part of the result
- TO_RF[15:0]: Real part of the result

a) Select a suitable instruction set for your module **(2p)**
b) Draw a schematic of your accelerator. **(6p)**
c) Draw a control table for your accelerator **(2p)**

5 Question 5: General knowledge (5p)

a) The following is a list of components commonly present in a DSP processor. One very important component is missing, which one? **(1p)**

- Arithmetic Logic Unit
- Program counter
- Register File
- Address Generator Unit
- Instruction decoder
- Data memory 0 (also includes memory mapped I/O)
- Data memory 1
- Multiply-Accumulate Unit

b) Draw a pipeline diagram of a DSP processor where you include all components from the previous task (including the missing component). Note: You should not draw the contents of the various parts, (e.g. just drawing a box and writing "PC" inside is enough for the program counter module, etc) **(1p)**

c) A four-tap FIR filter has the following coefficients: 0.4, 1.4, -1.1, and -0.1. Samples are in the range of $[-2, 2)$. How many guard bits are required if the result of the FIR filter should be stored in fractional format and we want to be able to detect an overflow in all cases? **(1p)**

d) In a pipelined processor you can have data, control, and structural hazards. Explain what a control hazard is. **(2p)**