Examination
Design of Embedded DSP Processors, TSEA26

Date	2010-01-14
Room	TER1 and TER2
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
Phone number during the exam time	013-288956
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. 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.

Question 1: General knowledge (5p)

a) A certain FIR filter has the coefficients of 0.25, -0.75, 3, and 0.5. The samples and the result shoule be in fractional format. How many guard bits do you need in the accumulator to make sure that you will always detect an overflow? (1p)

b) What is a delay slot? Explain why delay slots can increase the performance of a DSP processor. (2p)

c) In a real time system, can you trust the result of static profiling? What about dynamic profiling? (2p)

Question 2: ALU (8p)

Design an ALU capable of the following operations:

- OP0: `RESULT = A+B+Carry_in`
- OP1: `RESULT = A-B`
- OP2: `RESULT = ABS(B-A)`
- OP3: `RESULT = LEFTSHIFT(A,B[2:0])` (Shift A left by B[2:0] times)
- OP4: `RESULT = RIGHTSHIFT(A,B[2:0])` (Shift A right by B[2:0] times (not arithmetic shift))

Constraints::

- Overflow must be handled for `ABS(B-A)`. It is up to you if you want to handle overflow in any other case.
- You should minimize the amount of hardware such as adders and shifters.
- `A` is 8 bits wide, `B` is 8 bits wide, `Carry_in` is 1 bit wide. `RESULT` is 8 bits wide.

Tasks:

a) Draw a hardware schematic of your ALU. You should also annotate all signals except control signals to muxes with the bit width (6p)

b) Draw a control table for your ALU (2p)

Question 3: MAC (14p)

Design a MAC unit capable of the following operations:

- OP0: No operation
- OP1: `ACCx = A * B` (Fractional multiplication (signed))
- OP2: `ACCx = A * B + ACCy` (Fractional multiplication (signed))
- OP3: `ACCx = 4.0 * ACCy` (Scaling)
- OP4: `ACCx = 1.5 * ACCy` (Scaling)
- OP5: Load `ACCx` with a fractional value from a register
- OP6: `ACCx = SATURATE(ROUND(ACCy))`
- OP7: `ACCx = ACCy + ACCz`
- OP8: `ACCx = ACCy - ACCz`
- OP9: `RF = ACCy[7:0]`
- OP10: `RF = ACCy[15:8]`
- OP11: `RF = SIGNEXTEND(ACCy[21:16])`

Constraints::

- `A` and `B` are 8 bits, registers are 8 bits
- `ACC0`, `ACC1`, and `ACC2` are 22 bits (including 6 guard bits).
- `x`, `y`, and `z` are 2 bits wide and are sent from the instruction decoder to select the appropriate accumulator register. For example, where the description above says `ACCx`, this means that either `ACC0`, `ACC1`, or `ACC2` is used here, depending on the value of `x`. If, for example, `ACCy` is not present in that operation, the content of `y` is undefined.
- Only one multiplier may be used. You should select as small a multiplier as necessary. You also need to annotate whether it is signed or unsigned.

Tasks:

a) Draw a hardware schematic for your MAC unit. You must annotate the bit width of all signals except mux control signals. (11p)

b) Draw a control table for your MAC unit where you include all operations defined above. (3p)

Question 4: Address Generation Unit (AGU) (12p)

Profiling has shown that the following two functions are important for a certain application. Your task is to design an AGU that is capable of supporting them under the constraints given below.

```
function FIR_FILTER(samplesptr, bottom, top, coeffptr)
    repeat(128)
        ACC = ACC + dm0[samplesptr] * dm1[coeffptr]

        samplesptr = samplesptr + 1
        if samplesptr == top
        then
            samplesptr = bottom
        endif
        coeffptr = coeffptr + 1
    endrepeat
endfunction

function STORE_VAL(addr, value)
    parameter = dm0[59]
    dm0[addr] = value
    dm0[addr+2] = parameter
    dm0[addr+4] = parameter
    dm0[addr+8] = parameter
    dm0[addr+16] = parameter
    dm0[addr+32] = parameter
endfunction
```

Constraints::

- `STORE_VAL()` must execute in less than 12 clock cycles. `FIR_FILTER()` must execute in less than 140 clock cycles.
- The processor is a single scalar pipelined DSP processor which issues one instruction each clock cycle. (Like Senior.) You don't need to worry about pipeline penalties for any kind of jump in this exercise however.
- Function parameters are passed in general purpose registers
- You can assume that all other parts of the processor can handle the clock cycle requirements listed above. E.g. the MAC unit is connected to each memory, etc. This exercise is only about designing the AGU.

- The general purpose register file has two read ports and one write port. The memories are single ported.

Tasks:

a) Select the addressing modes you will need to fulfill the requirements listed above and write pseudo assembler code for the two functions. (3p)

b) Draw a hardware schematic of your AGU. You should minimize the amount of hardware in your AGU by for example making use of the fact that dm0 and dm1 will not require the same addressing modes. You don't need to annotate any bit widths in the AGU schematic. (6p)

c) Draw a control table for all of your addressing modes. Also, if your AGU includes any register, include instructions to set such registers in your control table. (3p)

Question 5: Special instructions (10p)

The function `FIR_3` is responsible for 70% of the time in a hypothetical application running on the Senior processor. Your task is to evaluate the hardware cost of speeding up this function by designing a custom instruction that is able to execute `FIR_3` in one clock cycle. Additionally, it is necessary to initialize the values used by the `FIR_3` function by using the `INITFIR_3` function. However, it is expected that the `FIR_3` function will be executed around 1000 times as often as the `INITFIR_3` function. This means that the `INITFIR_3` function does not need to execute quickly.

```
function FIR_3()
    samples[2] = samples[1]
    samples[1] = samples[0]
    samples[0] = dm0[inputptr]
    inputptr = inputptr + 1

    tmp = 0
    tmp = tmp + samples[0] * coefficients[0]
    tmp = tmp + samples[1] * coefficients[1]
    tmp = tmp + samples[2] * coefficients[2]

    dm1[outputptr] = SATURATE(tmp)
    outputptr = outputptr + 1
endfunction
```

```
function INITFIR_3(val1, val2, val3, val4, val5)
    samples[0] = 0
    samples[1] = 0
    samples[2] = 0
    inputptr = val1
    outputptr = val2
    coefficients[0] = val3
    coefficients[1] = val4
    coefficients[2] = val5
endfunction
```

Constraints:

- Function parameters are passed in general purpose registers
- `samples` contains 16 bit values in signed integer format
- `coefficients` contains 16 bit values in signed integer format
- The `tmp` variable has a suitable number of guard bits.
- You don't need to pipeline this unit for maximum clock frequency.
- You should be able to issue one `FIR_3()` instruction every clock cycle.

Tasks:

Your task is to design and implement the function `FIR_3()` as a special instruction on the Senior processor. (The pipeline of the Senior processor is shown in Figure 1 on the next page.) You also need to implement support for the `INITFIR_3()` function and you will probably need to add a couple of instructions to do this.

a) What parts of the Senior processor pipeline will you need to modify (besides the instruction decoder?) (1p)

b) Draw a hardware schematic of the modified parts of the pipeline. You don't need to annotate bit widths. You don't need to annotate the contents of a *SATURATE* box. (6p)

c) Draw a control table for your hardware where you include a `NOP` instruction, the `FIR_3` instruction and the instructions necessary to implement `INITFIR_3`. You should also write pseudo assembler code for `INITFIR_3`. (3p)

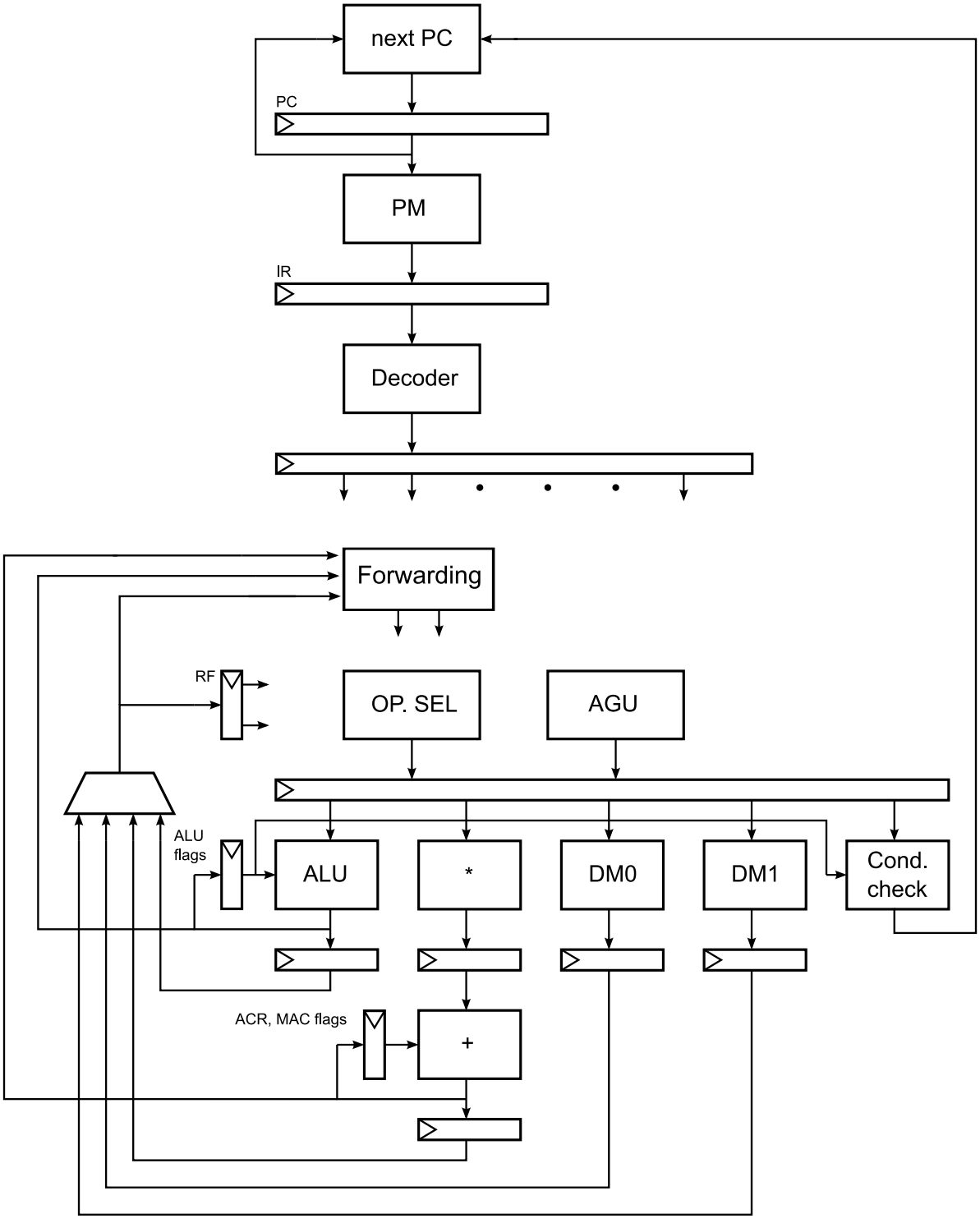

Figure 1: The Senior pipeline