Examination
Design of Embedded DSP Processors, TSEA26

Date	2010-08-27
Room	
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	Dake Liu
Phone number during the exam time	013-281256
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.

Question 1: General knowledge (5p)

a) You are asked to increase the performance of a 16-tap FIR filter which is running on a pipelined single issue RISC processor with 32 registers. Discuss what is most important to add to this RISC processor: A MAC instruction or a zero overhead loop instruction. (2p)

b) You are asked to increase the performance of an 8-point FFT which is running on a pipelined single issue RISC processor with 32 registers. Discuss what is most important to add to this RISC processor: A bit-reversed addressing mode or a butterfly instruction. (2p)

c) A 5-tap FIR filter is implemented using fractional multiplication in the MAC unit. The filter coefficients are 0.25, 0.75, -0.5, 0.25, and 0.75. It is also known that the absolute value of all input samples are always less than 0.75. Assuming that we want the output of the FIR filter in fractional format, how many guard bits do we need in the accumulator? (1p)

Question 2: Arithmetic units (10p)

Draw a schematic for an arithmetic unit capable of the following operations:

- OP0: `RESULT = A+B`
- OP1: `RESULT = A-B`
- OP2: `RESULT = (A+B+1)/2` (average with rounding)
- OP3: `RESULT = ABS(B)`
- OP4: `RESULT = MAX(A,B)`
- OP5: `RESULT = ABS(B-A)`

Constraints:

- Overflow must be handled for OP2, OP3 and OP5. It is up to you if you want to handle overflow in the remaining cases.
- `RESULT`, `A`, and `B` are 8 bits wide

Tasks:

a) Draw a hardware schematic of your arithmetic unit. You should minimize the amount of hardware, especially adders. You should also annotate the bit width of all signals except control signals to muxes. (7p)

b) Draw a control table for your ALU (3p)

Question 3: MAC (10p)

Design a MAC unit capable of the following operations:

- OP0: No operation
- OP1: `ACR = 0`
- OP2: `ACR = A * B` (Fractional multiplication (signed))
- OP3: `ACR = A * B + ACR` (Fractional multiplication (signed))
- OP4: `ACR = 1.25 * ACR` (Scaling)
- OP5: Load `ACR` with a fractional value from a register
- OP6: `ACR = SATURATE(ROUND(ACR))`
- OP7: `RF = ACR[7:0]`
- OP8: `RF = ACR[15:8]`
- OP9: `RF = SIGNEXTEND(ACR[19:16])`

Constraints:

- `A` and `B` are 8 bits, registers are 8 bits
- `ACR` is 20 bits (including 4 guard bits).
- **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.**
- Rounding is performed in such a way that OP8 can be used to read out the saturated and rounded result.

Tasks:

a) Draw a hardware schematic for your MAC unit. You must annotate the bit width of all signals except mux control signals. (7p)

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

Question 4: Understanding the pipeline (15p)

A schematic of a processor is shown on the next page. The datapath is 16 bits wide and there are 16 registers, `r0` to `r15`. `OpB` will always be set to the contents of a register whereas `OpA` can be set to either a register or the 16 least significant bits of the instruction word depending on whether bit 16 of the instruction word is 0 or 1, as shown in the figure. The program memory contains 4096 (2^{12}) instructions. The following instructions are already implemented:

Instruction	Explanation	Instruction	Explanation
`nop`	No operation	`set OpW, immediate`	`OpW = immediate`
`add OpW,OpA,OpB`	`OpW = OpA + OpB`	`load OpW, OpA`	`OpW = DM[OpA]`
`sub OpW,OpA,OpB`	`OpW = OpA - OpB`	`store OpA, OpB`	`DM[OpA] = OpB`

There are also 16 program flow control instructions implemented in the decoder, tentatively named `PFCOP0` ... `PFCOP15`. When a jump instruction is encountered, the decoder will set the signal `PFC_OP[4]` to 1. Additionally, `PFC_OP[3:0]` is set to 0 if `PFCOP0` was decoded, 1 if `PFCOP1` was decoded, and so on. If a jump instruction is not decoded, the value of `PFC_OP[3:0]` is undefined. Each `PFCOP` instruction may (but don't have to) use `OpA` and `OpB`. The 12 least significant bits (`PFC_DATA`) in the instruction word can be used as an absolute address, pc relative address, loop counter setting, etc.

Tasks:

a) How many delayslots will an unconditional jump have if the `FORCE_NOP` signal is deactivated? (1p)

b) Write assembly code to calculate the sum of `r0`, `r1`, and `r2`, using a minimum amount of instructions. **Hint: The pipeline has no forwarding or register bypass!** (1p)

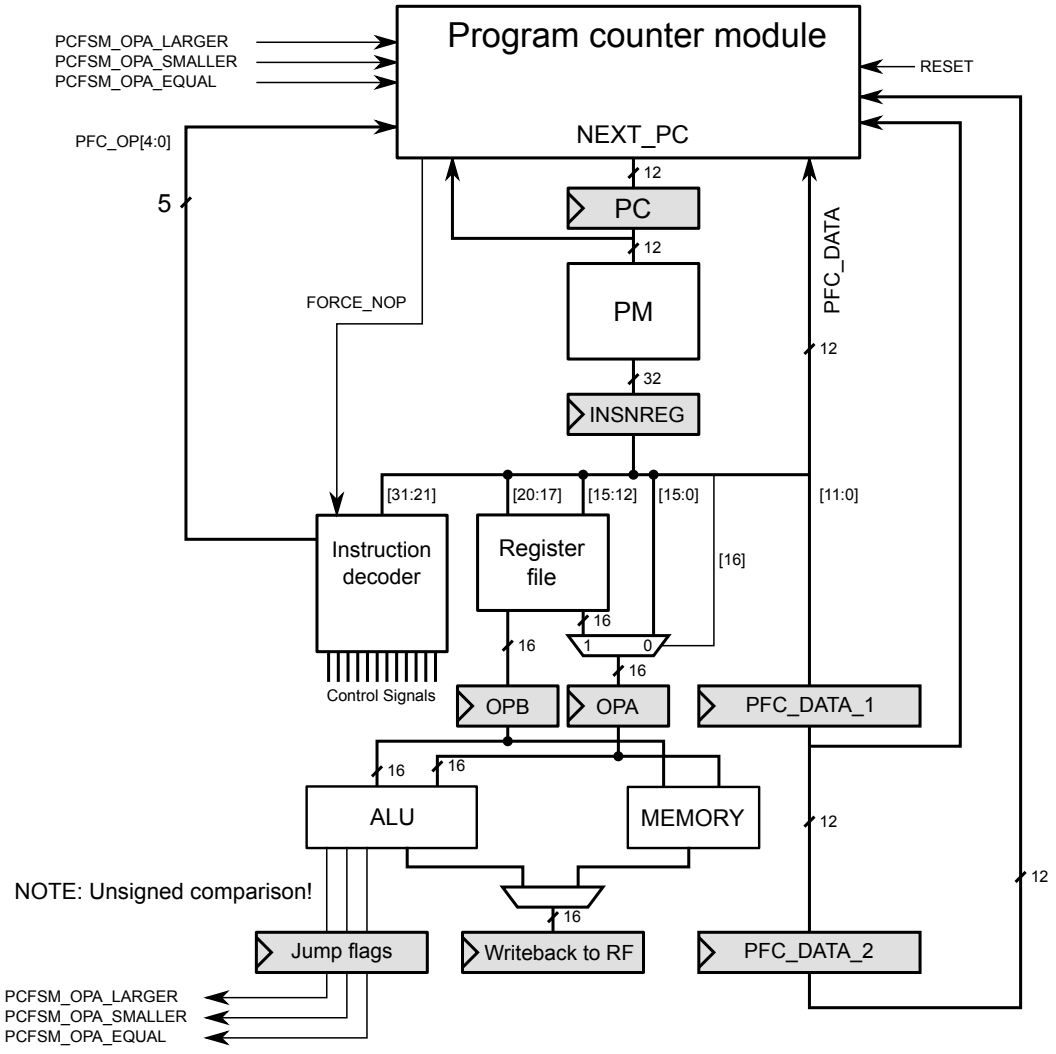
c) The following two programs need to execute in less than 200 cycles each. The size of program 1 must be less than 20 instructions. The size of program 2 must be less than 25 instructions. Decide on which program flow control instructions you will need to fulfill these constraints. You may create up to 16 program flow control instructions. (There are 4 bits in `PFC_OP[3:0]`.) You also need to describe how many delay slots each instruction has. Finally, you should write assembly code for both programs. (7p)

```
// Program 1:
// x is in r0, flag is in r1
if x >= 10
    x = x + 55
else
    x = x + 48
    if flag == 1
        x = x + 32
    end if
end if
```

```
// Program 2:
// inptr is in r0, outptr in r1
repeat 36
   x = DM[inptr]
   inptr = inptr + 1
   DM[outptr] = x
   outptr = outrptr + 1
endrepeat
```

d) Draw a hardware schematic of the program counter module. You may use multiplexers, registers, adders, FSMs (finite state machines), and logic gates. If you use an FSM, you must include the state diagram of the FSM. (6p)

Signal	Explanation
`FORCE_NOP`	This signal makes the instruction decoder believe that it is decoding a NOP instruction.
`NEXT_PC`	The next value for the program counter.
`PC`	The current value of the program counter.
`PCFSM_OPA_LARGER`	Set to 1 if OpA is larger (unsigned) than OpB
`PCFSM_OPA_SMALLER`	Set to 1 if OpA is smaller (unsigned)than OpB
`PCFSM_OPA_EQUAL`	Set to 1 if OpA is equal to OpB
`PFC_DATA`, `PFC_DATA_1`, `PFC_DATA_2`	The twelve least significant bits in the instruction word. These signals are delayed 0, 1 or 2 clock cycles.
`PFC_OP[4]`	1 if a jump is decoded, 0 otherwise
`PFC_OP[3:0]`	The kind of jump that was decoded.
`RESET`	System reset. Will be asserted for at least 16 cycles.



Question 5: ASIP instruction selection (10p)

The following function should be implemented on the Senior processor.

```
function ROTATE_VECTOR
    A = dm0[matrixptr]
    B = dm0[matrixptr+1]
    C = dm0[matrixptr+2]
    D = dm0[matrixptr+3]
    repeat 50
        X = dm0[vectorptr++]
        Y = dm0[vectorptr++]
        ROTATEDX = A*X+B*Y
        ROTATEDY = C*X+D*Y
        dm1[resultptr++] = ROTATEDX
        dm1[resultptr++] = ROTATEDY
    endrepeat
endfunction
```

Constraints:

- `matrixptr`, `vectorptr`, and `resultptr` are available in general purpose registers when the function is called.
- `A`, `B`, `C`, `D`, `X`, `Y`, `ROTATEDX`, and `ROTATEDY` are 16 bit fractional numbers.
- You don't need to worry about saturation and rounding in this exercise.
- You may not add any ports or change the width of either DM0 or DM1.

Tasks:

Your task is to modify the pipeline of the Senior processor so that the function `ROTATE_VECTOR` can be executed in less than 130 clock cycles. You will also need to select suitable instructions to implement this function. The pipeline of the Senior processor is shown in Figure 1 on the next page. **Note: If you can't manage it in 130 clock cycles, you may still get some points if you can manage it in less than 165 clock cycles.**

a) Select a set of new instructions that will allow you to execute `ROTATE_VECTOR` in less than 130 clock cycles and translate `ROTATE_VECTOR` into assembler. (2p)

b) Draw a hardware schematic of the modified parts of the pipeline. You don't need to annotate the bit widths of any signals. You may use up to two multipliers. You may also use as many gates, multiplexers and adders as you want to (within reason). (5p)

c) Draw a control table for your hardware where you include all instructions that you selected in task a. You should also include a NOP instruction in your control table. For each of your new instructions you also need to describe any AGU or memory operation that it may perform. (3p)

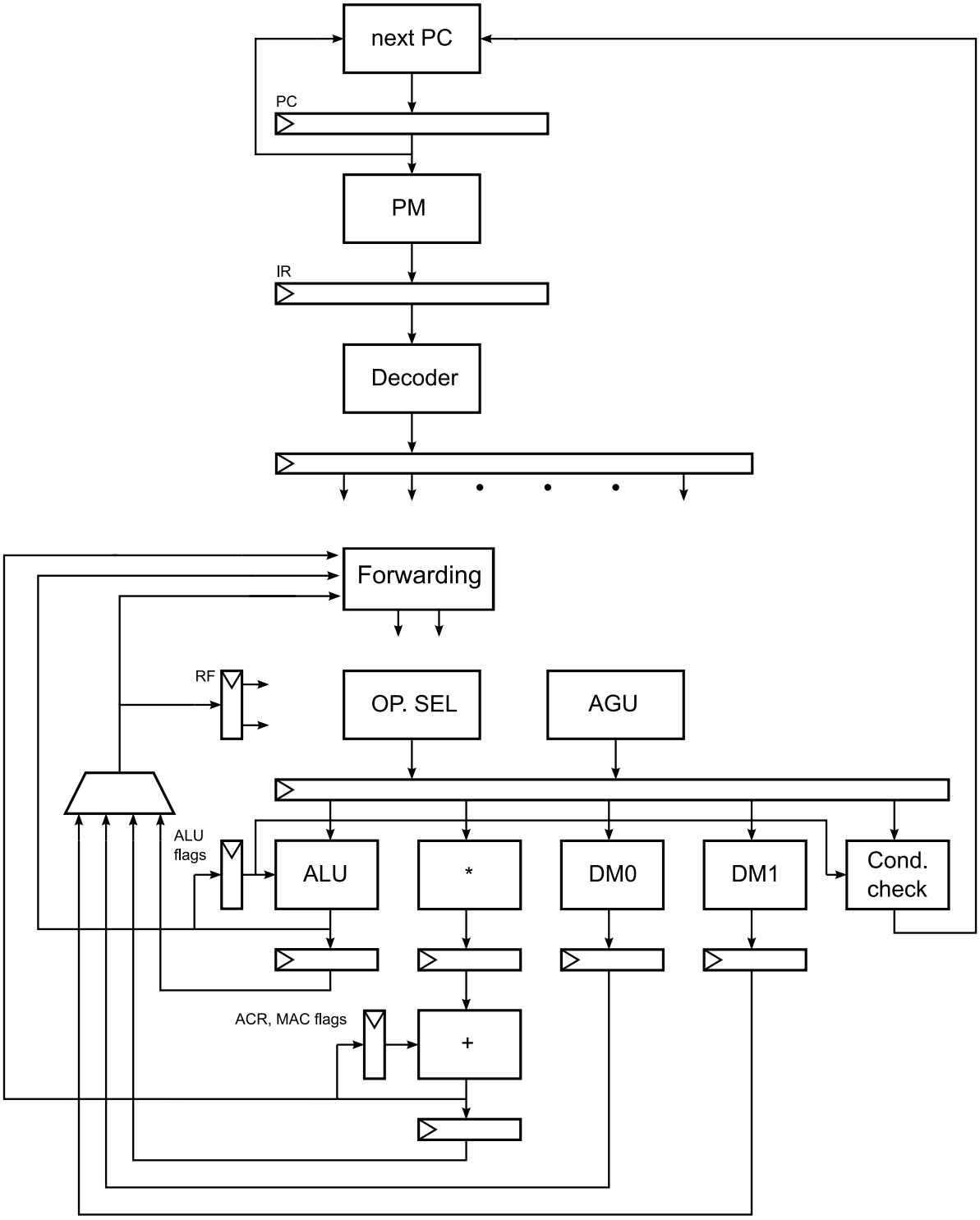


Figure 1: The Senior pipeline