

Imperial College London
Department of Computing

Computer Systems (113)
Architecture (110)
Assessed Coursework 2

Deadline: Thursday 11th March 2011
Submission: Either hardcopy to the Student Administration Office (SAO) or online using **CATE**. For CATE submit a pdf file named **fpio.pdf** (all lowercase)

Answer the following 3 questions:

- 1a Suppose that the IEEE defines a new 16-bit floating point format called Half Precision that follows the same general rules as IEEE Single Precision format, except that the Exponent is 7 bits and the Significand is 8 bits

<i>Half Precision</i>	1 bit	7 bits	8 bits
<i>Format</i>	Sign S	Exponent E	Significand F

For this format determine:

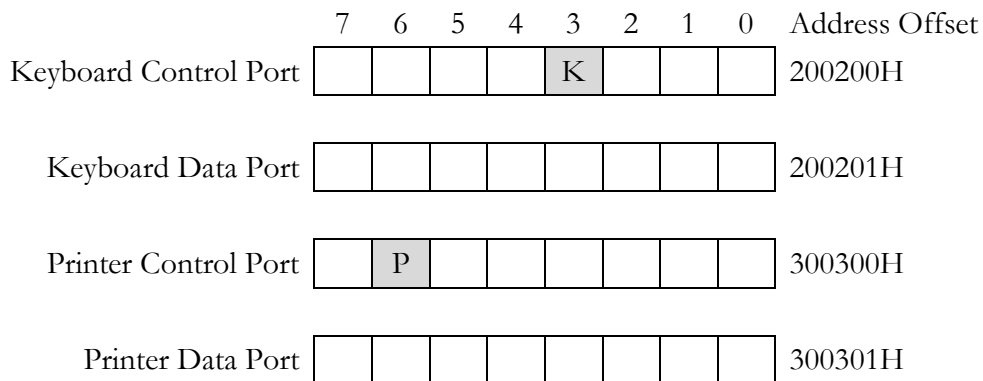
- i) the next number after 1952 that can be represented exactly.
- ii) the largest +ve normalised number that can be represented.
- iii) the smallest +ve (non-zero) normalised number that can be represented.
- iv) the largest +ve denormalised number that can be represented.
- v) the smallest +ve (non-zero) denormalised number that can be represented.

For parts (i) to (v) above, give your answer in both half precision format, and as a binary value in the form $1.bbbbbbbb \times 2^N$. For part (i) only, give your answer in decimal also.

- 1b Using binary floating-point addition, add the half precision number **43D9** HEX to **40D0** HEX. Show your working and give the result in half precision (as 16 bits) and in hexadecimal. If the result of the addition does not fit into half precision you should round the result **up** so that it does.

Question 2 is on the next page.

- 2 In this question you will be asked to use Programmed I/O to write a loop that continuously reads characters from a keyboard and outputs them to a printer. Each character that is read should be output immediately. Where possible reading and printing should proceed in parallel. The I/O ports for the keyboard and printer are defined as follows and assume for this question that they are mapped into **main memory** so can be accessed using standard instructions which take a memory operand.



For the keyboard, setting the K bit to 1 will initiate a keyboard read request. Once a character has been read into the Keyboard Data Port, the keyboard I/O controller will set the K bit to 0 to indicate completion of the transfer.

For the printer, setting the P bit to 1 will initiate a printer write request. Once the character in the Printer Data Port has been printed, the printer I/O controller will set the P bit to 0 to indicate completion of the transfer.

- i) First give a high-level language version of your programmed I/O loop.
- ii) Then give an Intel IA-32 assembly language version of your programmed I/O loop.

You can assume that no keyboard transfer is in progress at the start of your loop i.e. bit K=0.

State any additional assumptions that you make.

Question 3 is on the next page.

Q3 For this question you are asked to develop commented Intel IA-32 code that produces a beep every second given a clock device and a speaker device with the following I/O Ports, which you should again assume are mapped into **main memory**:

I/O Ports	8 bits	Address
<i>Clock Control Port</i>		FED120H
<i>Speaker Control Port</i>		FED130H

Writing the value 3 to the *Clock Control Port* causes an interrupt to occur 2 milliseconds later. You should assume that writing the value 3 overrides any 2 millisecond countdown that might be in progress from a previous write to this port.

Writing the value 6 to the *Speaker Control Port* causes a very short beep to be produced on the speaker.

Now write the following routines in IA-32 assembly language:

```

void StartBeeps()    % Generate a beep every second from now on

void StopBeeps()    % Stop generation of beeps

void ClockHandler    % Interrupt handler for Clock interrupts

int  NextBeep ()    % Return the number of milliseconds to the
                   % next beep. Assume that ints are 32-bit.

```

Once started with `StartBeeps`, a beep should be sounded every second, until a call is made to `StopBeeps`, which stops the beeps. Note: there are 1000 milliseconds in a second.

You can assume that a `StartBeeps` call is not followed later, by another `StartBeeps` call, without an intervening `StopBeeps` call.

You can also assume that your interrupt handler (`ClockHandler`) has been correctly installed in the Interrupt Descriptor Table.

State any additional assumptions that you make.