1. (10%) At power-up or after a reset, the PIC32 will enter the main() function right away. Is this statement true? Explain your answer if you don’t think this statement is true.

No, the statement is not true. After a reset or at power-up, before entering the main(), the PIC32 will execute a short initialization code segment (automatically inserted by the MPLAB linker.) This is known as the Startup code, or the crt0 (or simply c0) code. (See P. 7 of Di Jasio.)

2. (10%) Rewrite the following C statement using a *for-loop*: `while (1) { sum += 2; }`

The while loop is equivalent to: `for ( ; 1 ; ) { sum += 2; }

3. (15%) PIC32 handles 8-bit and 16-bit integers more efficiently, both in terms of space and performance, than 32-bit integers. Is this statement true? Justify your answer.

No. 8-, 16- and 32-bit data are processed with equal efficiency. This means 32 bits (4 bytes) are allocated for all of these data; and arithmetic and logic operations for these quantities are performed using single assembly instructions in one clock cycle (see p. 62 of textbook).

4. (20%) Given a peripheral bus frequency of 36MHz and the delay equation discussed in class, will it be possible to produce a real-time delay of 1.024 seconds on PIC32 I, II or III using the constant DELAY and the C code from the “Day 2” project? Justify your answer.

Using the real-time delay equation: \( T_{\text{delay}} = \frac{1}{36 \text{ MHz}} \times 256 \times \text{DELAY} \), from the Day 2 project, where \( T_{\text{delay}} = 1.024 \text{ s} = 1024 \times 10^{-3} \text{ s} \), as required by this problem, we will obtain \( \text{DELAY} = 144000 \).

Since Timer1 of PIC32 is 16-bit wide, the new DELAY value of 144000 is too large to fit into Timer1 at the given peripheral bus frequency of 36 MHz, and a maximum prescaler value of 256. Therefore it will not be possible to produce the required 1.024 seconds of real-time delay. (See P. 32 of Di Jasio.)

5. (20%) In the lectures, we examined how *long long division* was performed, and noted that the MIPS instruction *jal* was needed. Explain what *jal* is and how it is used to support *long long division*.

The MIPS instruction *jal* stands for “jump and link”. This instruction makes a subroutine call to perform the long long division. The subroutine is part of a library named libgcc2.c. By reusing this subroutine for multiple long long division operations through *jal* calls, overall code size is minimized (p. 68 of textbook).

6. (10%) In the Morse code part of the “Day 3” project, a character array *bitmap[]* was used, where its elements take on the values of 1 or 0. How do these element values affect the LEDs’ ON/OFF in the hardware? Which PIC32MX I/O PORT was involved? Explain your answer in sufficient details.
These element values control which LED(s) are ON or OFF. For example, a value of 0 means all LEDs are OFF, and a value of 7 means all LEDs are ON, because the value is translated to a 3-bit binary value to control the lower 3 bits of PORTD, i.e. RD2, RD1 and RD0.

7. (15%) What does the following special function do: `void _on_reset(void)`? Why should you minimize the amount of code for this special function?

This special function will replace an empty place holder (i.e. stub) that the crt0 code normally calls before it starts initializing other data items. Need to minimize the code size for this special function because this function is called every time the PIC32 goes through a reset (see p. 59 of textbook).