1. MIPS Instructions 20 Points

Consider the sequence of four instructions shown below.

```
loop:  lw   $6, 0($5)
       add $7, $7, $6
       addi $5, $5, -4
       bne $5, $7, loop
```

In the space provided below, show the binary representation of each of the four instructions in this sequence. Clearly mark the instruction fields and show their decimal equivalent.

```
lw   $6, 0($5)
```

```
add  $7, $7, $6
```

```
addi $5, $5, -4
```

```
bne $5, $7, loop
```
2. **MIPS Branch and Jump Instructions**  

(a) The MIPS branch instructions are *beq* and *bne* can only change the current program counter by a limited amount. How many instructions can be reached using a branch instruction? What range of instructions can be reached relative to the program counter?

(b) The MIPS jump (*j*) instruction is also limited in its ability to change the program counter. How many instructions can be reached using a jump instruction? What range of instructions can be reached relative to the program counter?

(c) Given the limitations of the branch and jump instructions, we would like to create a pseudoinstruction that will allow a program to jump to any 32-bit address. Write a minimum-length sequence of MIPS instructions that implements this "jump anywhere" (*ja*) pseudoinstruction.

Pseudoinstruction: Assembly Language:

*ja address*
3. Procedures and Functions 20 Points

The code fragment below shows a simple C function that is called by the main program. Note that this function is a *leaf procedure* i.e., it calls no other functions or procedures.

```
C Code                                               Assembly Code
int f(int a0, int a1) { f:
    return (a0 + a1);
}
main() { main:
    int s0, s1, s2;
    s0 = 96000;
    s1 = -12;
    s2 = f(s0, s1);
}
```

(a) Assume that variables `s0`, `s1`, and `s2` and arguments `a0` and `a1` are stored in MIPS registers with the same name. In the space to the right of the C code above, fill in the MIPS assembly instructions that will implement the main function and the function `f` while storing arguments and changing the flow of control.

(b) Suppose that `f` is not a leaf procedure, but instead calls other leaf procedure `g`, which has three integer arguments. Briefly describe how your code would need to be changed. You do not need to write any code for this part – just describe what needs to be done, and list any values that would need to be stored on the stack.
4. Carry Lookahead  

In a carry-lookahead adder, each bit-slice of the adder must use inputs $a_i$ and $b_i$ and $c_i$ to produce generate function $g_i$, propagate function $p_i$, and sum output $s_i$.

A proposal has been made to replace the standard definition of the propagate function:

$$p_i = a_i + b_i$$

With a version that uses the exclusive or function:

$$p_i = a_i \oplus b_i$$

The carry function is now implemented in the carry-lookahead unit as before:

$$c_{i+1} = g_i + p_i \cdot c_i$$

(a) Will the modified propagate function work properly in the calculation of carry signals by the carry lookahead unit? Are any additional modifications needed? Why or why not?

(b) What advantage would there be in using the exclusive-or version of the propagate signal?

5. Verilog Coding  

The Verilog code shown below implements an arithmetic function on three 32-bit inputs. What is this function?

```verilog
module mystery_module ( a, b, c, y );
    input [31:0] a, b;
    output [31:0] y;

    wire w;

    assign w = (a > b) ? a : b;
    assign y = (c > w) ? c : w;
endmodule
```
6. Multiplication Hardware 15 Points

The two unsigned binary numbers shown below are to be multiplied using a standard “shift and add” sequential multiplier and a multiplier that uses Booth’s Algorithm:

\[
\begin{array}{c}
10011010 \\
\times \quad 01111010 \\
\end{array}
\]

Answer the following questions:

(a) How many bits will be needed to store the product of these two numbers?

(b) How many additions will be performed by the standard shift-and-add multiplier that multiplies these numbers?

(c) How many additions will be performed by the Booth’s Algorithm multiplier that multiplies these numbers?

(d) How many subtractions will be performed by the Booth’s Algorithm multiplier that multiplies these numbers?

(e) How many Half Adders and Full Adders would be needed to implement a combinational multiplier that can multiply these numbers?