Question 1 (2 points)

Convert the binary value 110000001111111111101110 into

a) hexadecimal (base 16)

b) octal (base 8)

Question 2 (4 points)

Assuming a five-bit word length, convert the binary value 11100 to decimal, supposing the representation is

a) unsigned

b) sign-magnitude

c) ones complement

d) two's complement

Question 3 (3 points)

Decode the following binary numbers as MIPS instructions and give the equivalent MIPS assembly language (MAL) statements. Show memory addresses, if any, in hexadecimal.

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x40</td>
<td>0000110010111100000000000100100</td>
</tr>
<tr>
<td>0x44</td>
<td>0100011000000100010000110000010</td>
</tr>
<tr>
<td>0x48</td>
<td>0000010011000011111111111111111</td>
</tr>
</tbody>
</table>

Question 4 (2 points)

In the MIPS procedure-calling convention, there exist a compromise between a pure callee-saved and a pure caller-saved convention. That is, some registers are callee-saved ($s$ registers) and some are caller-saved ($t$ registers).
In one English sentence, explain why the MIPS designers chose this mixed strategy rather than either pure callee-saved or pure caller-saved.

**Question 5 (4 points)**

Consider this C struct definition:

```c
struct foo {
    int a[3];
    char b[4];
    int *c;
} baz;
```

Suppose that register $16$ contains the address of `baz`.

Here is a fragment of C code:

```c
baz.c = baz.a;
baz.c[1] = baz.b[2];
```

Below is a buggy translation into MIPS assembly language. Find and fix all the bugs:

```assembly
lw  $8, 0($16)
sw  $8, 28($16)

lw  $8, 20($16)

lw  $9, 28($16)
sw  $8, 4($9)
```
**Question 6 (8 points)**

a) Translate the following C procedure, which recursively computes the number of times a given character occurs in a given string, to MAL. Use the convention in which arguments are passed in registers. Use as little stack space as possible.

```c
int count(char ch, char *str) {
    if (*str == '\0') return 0;
    if (*str == ch) return count(ch, str+1)+1;
    return count(ch, str+1);
}
```

b) Now translate to MAL the following iterative computation of the same function. Again, use the convention in which arguments are passed in registers. Use as little stack space as possible.

```c
int count (char ch, char *str) {
    int result=0;
    while (*str != '\0') {
        if (*str == ch) result++;
        str++;
    }
    return result;
}
```

**Question 7 (3 points)**

Using only two-input NOR gates (which return the value NOT(A OR B)), implement the CARRY output of a half-adder.
(Hint: What is A NOR A?)