Question 1: Running in circles (25 min, 18 pts)

A nibble is half of a byte (4 bits). You’d like to implement LoadNibble in MAL MIPS, a function that takes one uint32_t argument n and returns the n\textsuperscript{th} nibble of memory in the lowest 4 bits of the return register (the other 28 bits should be 0). Note: The n\textsuperscript{th} nibble immediately follows the n-1\textsuperscript{th} nibble without overlapping; see box. The MIPS instruction \texttt{srlv} (“shift right variable”) might be useful here; it operates like the \texttt{shamt}-based right-shift, except that its 3\textsuperscript{rd} register argument is the variable amount to shift by.

\begin{enumerate}
\item What fraction of all the nibbles of memory can you access? \underline{1/2}
\item Implement \texttt{LoadNibble} by filling in the blanks:
\begin{verbatim}
loadNibble:       srl $t0 1
                  lbu $t0
                  and $a1 0(\_\_\_\_\_\_
                  sll $a0 2
                  srlv $v0 $a1 $a0
                  gone1:
                  andi $v0 $v0 $a0
                  gone2:
                  jr $ra
\end{verbatim}
\end{enumerate}

\item We want to rewrite \texttt{LoadNibble} to make use of a helper function \texttt{Helper} that will take two arguments. The first is an index \texttt{i} from 0-1 and the second is a byte \texttt{B}. \texttt{Helper} returns the i\textsuperscript{th} nibble in \texttt{B} placed in the lowest 4 bits of the return value (the rest 0s).

\begin{enumerate}
\item E.g., \texttt{Helper(0, 0b01100100)} $\rightarrow$ \texttt{0b0100} and \texttt{Helper(1, 0b01100100)} $\rightarrow$ \texttt{0b0110}
\item We decide we don’t need the two MIPS instructions labeled “gone1" and “gone2”. What would you replace these instructions \texttt{(and the s1l)} with to call \texttt{Helper} and implement \texttt{LoadNibble} successfully? Write the replacement below. Follow calling conventions and complete it in the fewest lines possible.
\begin{verbatim}
addiu $sp $sp -4
sw $ra 0($sp)
jal Helper       # j works too, all other lines blank (since $ra = LoadNibble’s caller)
lw $ra 0($sp)
addiu $sp $sp 4
\end{verbatim}
\end{enumerate}
Question 2: I can C clearly now, the rain is gone... (25 min, 18 pts)

A) Fill in the blank to complete this function that parses a string of octal digits (base 8) into a uint64_t. For example, calling parse_octal("71") should return the number 57. Do not use the comma operator, nested assignment, prefix/postfix operators, or function calls. You may assume that the given number "fits" into a uint64_t. (Hint: The backside of the MIPS green sheet may help.)

```c
uint64_t parse_octal(char *s) {
    uint64_t r = 0;
    while(*s) {
        r = ________________________;
        s++;
    }
    return r;
}
```

B) We have the following data packed tightly (no padding) into the struct data, and some more code below:

```c
struct {
    int16_t a;
    char b[2+(UNKNOWN_LENGTH+4)];
    int32_t c;
    int32_t d;
} data;

/* ... Some code here that fills in data.b with the longest string possible ... */

```c
char *s = data.b; /* *s is a char, so it counts by 1 byte by default if in parens */

```c
*(int16_t *)((s-1)/* or (s-2) */)
                  ________________________ ) = -1; // data.a = -1;
*(int32_t *)((s+strlen(s)+1+4)
                  ________________________ ) = -1; // data.d = -1;
```

C) Here we have a LR-tree, defined as a node with two arrays of child pointers: two left children and two right children. Each node also contains a pointer to its parent node, a unique integer ID value, and a string name field. Root nodes will have a NULL parent pointer, and leaf nodes will have arrays of NULL children pointers.

Fill in the blanks to complete this function that frees a LR-tree if called with the root of the tree. You must free ALL data associated with this LR-tree! You might not need all of the blanks, in which case use the most minimal number of blanks possible. Do not use the comma operator, nested assignment, or prefix/postfix operators.

```c
void free_lr_tree(struct lr_tree *p) {
    if (______________________________ ){
        for(size_t x = 0; x < 2; x++) {
            free_lr_tree(p->left_children[x]);
            ______________________________;
            free_lr_tree(p->right_children[x]);
            ______________________________;
        }
        free(p->name);
        ______________________________;
        free(p);
        ______________________________;
        ______________________________;
    }
}
```

```c
struct lr_tree{
    char *name;
    uint64_t ID;
    struct lr_tree *left_children[2];
    struct lr_tree *right_children[2];
    struct lr_tree *parent;
};
```