Quiz # 3
Dec. 5, 2013 (with solutions)

1. Implementing XOR with AND gates

Recall that the Exclusive-OR (XOR) function is 1 if an odd number of the inputs are 1, and 0 otherwise. Suppose that we have to build an circuit that computes XOR with AND gates, OR gates, and inverters.

(a) Draw a circuit to compute the XOR of 7 variables with the fewest possible number of AND and OR gates, and as many inverters as you want.

Solution
Any solution with 15 AND/OR gates gets full credit. For instance:
\[ \text{XOR}(\text{XOR}(x_1, x_2, x_3), \text{XOR}(x_4, x_5, x_6), x_7) \]

(b) Draw a circuit to compute the XOR of 26 variables with the fewest possible number of AND and OR gates, and as many inverters as you want.

Solution
Any solution with 63 AND/OR gates gets full credit. For instance:
\[ \text{XOR}(\text{XOR}(x_1, x_2, x_3), \text{XOR}(x_4, x_5, x_6), \text{XOR}(x_7, x_8, x_9), \text{XOR}(x_{10}, x_{11}, x_{12}), \text{XOR}(x_{13}, x_{14}, x_{15}), \text{XOR}(x_{16}, x_{17}, x_{18}), \text{XOR}(x_{19}, x_{20}, x_{21}), \text{XOR}(x_{22}, x_{23}, x_{24}), \text{XOR}(x_{25}, x_{26})) \]

(You can have XOR “boxes” in your drawings. You don’t have to show the contents of each box, but show the contents of each box of a given size.)

Here \( \text{XOR}(A, B) = \text{OR}(\text{AND}(A, B'), \text{AND}(A', B)) \),
and \( \text{XOR}(A, B, C) = \text{OR}(\text{AND}(A, B', C'), \text{AND}(A', B, C'), \text{AND}(A', B', C), \text{AND}(A, B, C)) \).
2. Batcher Sorting Network

A comparator is a device which sorts two numbers $x$ and $y$, as shown in Figure 1.

![Comparator Diagram](image)

Figure 1: a comparator.

The Batcher sorting network was presented in class. It is constructed with a block called the Merge network. Given two sorted input sequences $x_0, x_1, \ldots, x_{n/2-1}$ and $x'_0, x'_1, \ldots, x'_{n/2-1}$, the Merge[n] network produces a sorted output sequence $y_0, y_1, \ldots, y_{n-1}$.

The recursive construction of the Merge[8] network is shown in Figure 2. The recursive construction of the Batcher[8] network, based on the Merge[8] network, is shown in Figure 3. The Merge[2] and Batcher[2] networks both consist of a single balancer.

![Merge[8] Diagram](image)

![Batcher[8] Diagram](image)

(a) What is the depth of the Merge[8] network?
(b) What is the depth of the Batcher[8] network?
(c) What is the depth of the Merge[64] network?
(d) What is the depth of the Batcher[64] network?

(The depth of a network is the number of stages, i.e., the number of comparators that one traverses from an input to an output.)

Solution

(a) 3
(b) 6
(c) 6
(d) 21

Figure 3: the Batcher[8] network.
3. A Bit of Horticulture – Traversing Trees

Consider the following data structure:

```c
struct node {
    int x;
    struct node *left;
    struct node *right;
};
```

The tedious code to setup a tree is shown at the end. There is also a sketch of the corresponding tree.

(a) What does the following function print out?

```c
void dfs(struct node *p) {
    if (p->left != NULL) {
        dfs(p->left);
    }
    printf("%d ", p->x);
    if (p->right != NULL) {
        dfs(p->right);
    }
}
```

```c
int main(int argc, char **argv) {
    struct node *p = setup_tree();
    dfs(p);
}
```
Solution
4 2 8 5 9 1 6 3 12 10 7 11
(b) What does the following function print out?

```c
void dfs1(struct node *p) {
    printf("%d\n", p->x);
    if (p->left != NULL) {
        dfs2(p->right);
    }
    if (p->right != NULL) {
        dfs2(p->left);
    }
}

void dfs2(struct node *p) {
    if (p->left != NULL) {
        dfs1(p->left);
    }
    if (p->right != NULL) {
        dfs1(p->right);
    }
    printf("%d\n", p->x);
}

int main(int argc, char **argv) {
    struct node *p = setup_tree();
    dfs1(p);
}
```

Solution

1
6
7
11
12
10
3
4
5
9
8
2
Here is the (tedious) code to create the tree.

```c
struct node *setup_tree(void) {
    // create tree
    struct node *p = malloc(sizeof(struct node));
    p->left = malloc(sizeof(struct node));
    p->right = malloc(sizeof(struct node));
    p->left->left = malloc(sizeof(struct node));
    p->left->right = malloc(sizeof(struct node));
    p->right->left = malloc(sizeof(struct node));
    p->right->right = malloc(sizeof(struct node));
    p->right->right->left = malloc(sizeof(struct node));
    p->right->right->right = malloc(sizeof(struct node));
    p->right->right->left->left = malloc(sizeof(struct node));
    p->left->x = 1;
    p->left->x = 2;
    p->right->x = 3;
    p->left->left->x = 4;
    p->left->left->left = NULL;
    p->left->left->right = NULL;
    p->left->right->x = 5;
    p->right->left->x = 6;
    p->right->left->left = NULL;
    p->right->left->right = NULL;
    p->right->right->x = 7;
    p->left->right->left->x = 8;
    p->left->right->left->left = NULL;
    p->left->right->left->right = NULL;
    p->left->right->right->x = 9;
    p->left->right->right->left = NULL;
    p->left->right->right->right = NULL;
    p->right->right->left->x = 10;
    p->right->right->left->right = NULL;
    p->right->right->right->x = 11;
    p->right->right->right->left = NULL;
    p->right->right->right->right = NULL;
    p->right->right->left->left->x = 12;
    return p;
}
```
Figure 5: Tree
4. Pointers

(a) What does the following program print out?

```c
#include <stdio.h>

int main(int argc, char **argv) {

    int *p;
    int *q;
    int *r;

    int x = 1;
    int y = 2;
    int z = 3;

    p = &x;
    q = &y;
    r = &z;

    *r = x;
    x = *q;
    y = *r;

    printf("%d %d %d\n", x, y, z);

    x = 4;
    y = 5;
    z = 6;

    r = p;
    p = q;
    q = r;

    *p = 7;
    *q = 8;
    *r = 9;

    printf("%d %d %d\n", x, y, z);
}
```
Solution

2 1 1
9 7 6

(b) What does the following program print out?

```c
#include <stdio.h>

int main(int argc, char **argv) {

    int *p;
    int **q;
    int ***r;

    int x = 1;
    int y = 2;

    p = &x;
    q = &p;
    r = &q;

    *p = 3;
    **q = *p;
    ***r = **q;
    printf("%d %d %d %d %d\n", x, y, *p, **q, ***r);

    int z[5] = {2, 3, 4, 5, 6};
    z[z[z[0]]] = 7;

    int i;
    for (i = 0; i < 5; i++) {
        printf("%d ", z[i]);
    }
    printf("\n");
}
```
Solution

3 2 3 3 3
2 3 4 5 7
5. Error Correcting Codes

Suppose that Alice wants to send Bob 4 bits of information at a time, \(x_0, x_1, x_2, x_3\), over a noisy Wi-fi connection that occasionally flips bits. She decides to encode her information by adding three extra bits \(x_5, x_6, x_7\) computed as follows (here + represents exclusive OR):

\[
\begin{align*}
    x_4 &= x_1 + x_2 + x_3 \\
    x_5 &= x_0 + x_2 + x_3 \\
    x_6 &= x_0 + x_1 + x_3
\end{align*}
\]

She sends the 7 bits \(x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7\) to Bob.

Consider the following matrix, called a parity check matrix:

\[
H = \begin{bmatrix}
0 & 1 & 1 & 1 & 1 & 0 & 0 \\
1 & 0 & 1 & 1 & 0 & 1 & 0 \\
1 & 1 & 0 & 1 & 0 & 0 & 1
\end{bmatrix}.
\]

The vector of the seven 7 bits that Alice sends, \(X = [x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7]\), satisfies

\[
HX^T = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}
\]

Problems

(a) Suppose that Alice wants to send the bits 1,1,1,1. What seven bits will she transmit?

**Solution**

1,1,1,1,1,1,1.

(b) Suppose that Bob receives the seven bits 1,0,0,1,0,0,1. Which bit was flipped? What will he conclude were Alice’s original four bits?

**Solution**

Second bit. 1,1,0,1,0,0,1.