Karnaugh Maps II; Bit Operations

CS 350: Computer Organization & Assembler Language Programming

Due Fri Sep 20

A. Why?

- Simplifying boolean expressions can make them more readable and require less hardware to implement.
- Karnaugh maps are a good way to find a simple logical expression for a truth function.
- Bitstring operations (`<<`, `>>`, `&`, `|`, …) let us select and manipulate bits of a bitstring and to multiply and divide integers by powers of two.

B. Outcomes

After this lecture lab, you should be able to:

- Simplify a 4-variable truth function using a Karnaugh map.
- Write code that shifts and selects bits of a bitstring.

C. Written Problems [60 points total]

For Problems 1 and 2 [20 pts each] take the given truth table and find a simplest boolean expression for the result: (a) Write out a Karnaugh map for R; (b) Draw rectangles to indicate a set of prime implicants; and (c) Translate the prime implicants into minimal DNF form.

3. [20 pts[Find a simplest boolean expression equivalent to R below: (a) Form a Karnaugh map for the expression, (b) Select prime implicants, and (c) Trans-

Problem 1

V	*X*	*Y*	*Z*	*R*
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

Problem 2

V	*X*	*Y*	*Z*	*R*
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	0
1	1	1	1	0

late the prime implicants into DNF form.

$$R = V\,X\,\neg(V\,X\,\overline{Y}\,Z) + X\overline{Z} + V\,X\,Y + \overline{Y}\,\neg(X\,Z) + V\,Y\overline{Z} + \overline{V}\,\overline{Z}$$

You can (but aren't required to) give a regular truth table for R before creating the Karnaugh map.)

D. Programming Problem: Bit Operations [40 points]

Write a C program that repeatedly reads and selects specified bits from an integer. For each round of input, read in an unsigned integer X in hexadecimal. If $X = 0$, stop the program. Otherwise read two integers (left and right endpoints) and find the bits of X specified by the two endpoints. Print out X and the resulting bitstring in decimal and hexadecimal, and go to the next round of input. If an endpoint is illegal, complain about it and skip to the next round of input.

The purpose of this problem is to practice using the bit shifting and bit manipulation operators (`<<`, `>>`, `&`, `|`, …), so use them to find the shifted bitstrings.

The exact format of your output is up to you, but you must process the inputs in the specified the sequence: X, left endpoint, right endpoint, X, left endpoint, right endpoint, … $X = 0$ causing stop. Here's an excerpt of possible sample output, with user input in bold italics. (Recall: The bits are labeled left to right.)

```
Enter X in hex (0 to stop): F0000000
Left and right endpoints: 0 3
X[0:3] = 0xF = 15

Enter X in hex (0 to stop): F0000000
Left and right endpoints: 0 8
X[0:3] = 0x1E0 = 480

Enter X in hex (0 to stop): 01234567
Left and right endpoints: 0 31
X[0:3] = 0x01234567 = 19088743

Enter X in hex (0 to stop): 01234567
Left and right endpoints: -1 43
-1 is an illegal endpoint
43 is an illegal endpoint
```


Point Breakdown

[5 pts] Looping correctly

[5 pts] Reading X and reading/verifying the endpoints.

[15 pts] Correctly calculating and printing out the selected bits.

[10 pts] Code structure (e.g., calculating the masks dynamically instead of doing things like using 32 different if tests).

[5 pts] Commenting (including your name & section in comments and output).

Miscellaneous Hints

- Left endpoint should be $\leq$ right endpoint.
- Either endpoints 0…31 are a special case or you need longer integers.
- Ignoring comments, your program could be < 50 lines long.

