CS 271Computer Architecture and Assembly Language

Self-Check for Lecture#8

Solutions

1.
   a. Show the 16-bit representation of 2437(decimal).

      2437 (decimal) = 100110000101 binary = 0000100110000101(16-bit)

   b. Convert the 16-bit representation of part (a)to the corresponding odd-parity Hamming code. Add the appropriate number of parity bits.

      - Since the representation is 16-bit, we need $\log_2 16 + 1 = 5$ additional bits, i.e. the Hamming code will have 21 bits.
      - Number the places left to right starting with 1, and put the digits of the 16-bit number into the places, skipping bits with numbers that are powers of 2:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
		0		0	0	0		1	0	0	1	1	0	0		0	0	1	0	1

- Bit #1 determines parity for bits #1,3,5,7,9,11,13,15,17,19,21
  - bits # 9,13,19, and 21 are set to 1, so bit #1 must be **1** to make odd-parity.
- Bit #2 determines parity for bits #2,3,6,7,10,11,14,15,18,19
  - bit #19 is set to 1, so bit #2 must be **0**.
- Bit #4 determines parity for bits #4,5,6,7,12,13,14,15,20,21
  - bits # 12,13, and 21 are set to 1, so bit #4 must be **0**.
- Bit #8 determines parity for bits #8,9,10,11,12,13,14,15
  - bits # 9,12, and 13 are set to 1, so bit #8 must be **0**.
- Bit #16 determines parity for bits #16,17,18,19,20,21
  - bits # 19 and 21 are set to 1, so bit #16 must be **1**.

Answer:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	**0**	0	**0**	0	0	0	**0**	1	0	0	1	1	0	0	**1**	0	0	1	0	1

2. Given the 21-bit even-parity Hamming code 100001100011100110101.

   a. Which bit is incorrect?

      - Number the bits as shown in the first problem, enter the Hamming code, and mark the parity bits:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	**0**	0	**0**	0	1	1	**0**	0	0	1	1	1	0	0	**1**	1	0	1	0	1

- Bit #1 determines parity for bits #1,3,5,7,9,11,13,15,17,19,21
  - bits # 1,7,11,13,17,19, and 21 are set to 1. That's seven 1-bits, so there is an even-parity error in one of the bits in *ErrorSet1* = {1,3,5,7,9,11,13,15,17,19, 21}.
- Bit #2 determines parity for bits #2,3,6,7,10,11,14,15,18,19
  - bits #6,7,11, and 19 are set to 1. That's four 1-bits, so the error is <u>not</u> in the bits in {2,3,6,7,10,11,14,15,18, 19}.
- Bit #4 determines parity for bits #4,5,6,7,12,13,14,15,20,21
  - bits # 6,7, 12,13, and 21 are set to 1. That's five 1-bits, so there is an even-parity error in one of the bits in *ErrorSet4* = {4,5,6,7,12,13,14,15,20, 21}.
- Bit #8 determines parity for bits #8,9,10,11,12,13,14,15
  - bits # 11,12, and 13 are set to 1. That's three 1-bits, so there is an even-parity error in one of the bits in *ErrorSet8* = {8,9,10,11,12,13,14, 15}.
- Bit #16 determines parity for bits #16,17,18,19,20,21
  - bits # 16,17,19 and 21 are set to 1. That's four 1-bits, so the error is <u>not</u> in bits #16,17,18,19,20, or 21.

Answer:

- The only number in the intersection of ErrorSet1, ErrorSet4, and ErrorSet8 is 13, so the error is in bit #13.

- Another way to find it is to add the parity place numbers that have errors: 1 + 4 + 8 = 13.

b. After the error is corrected, what decimal number is represented by the Hamming code of part (a)?

- The corrected code is

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	0	0	0	0	1	1	0	0	0	1	1	**0**	0	0	1	1	0	1	0	1

- Extract the data bits … 001100110 0 010101, and convert to decimal:

Answer: **13077**

3. Note: This is NOT a programming assignment (but you might enjoy programming it anyway).

I need a program to calculate the odds of winning a lottery. The user enters the range of possible numbers and the number of picks required. For example, the user might enter 42 for the range, with 5 picks on one ticket. This will involve calculating the number of combinations of r items taken from a set of n items (i.e., $_nC_r$).

The program should display the odds of winning with one ticket.
For example: The odds of winning with 5 picks from 42 lottery numbers: 1 in 850668

a. How would you modularize this problem?

One way to break the problem into its logical components:

1)display a title screen
2)get the user's numbers
3)calculate the odds
 a. calculate nCr
  i. calculate factorials
4)display result

b. Show a hierarchy chart of your modularization.