CS/ECE 552, Introduction to Computer Architecture
Spring 2012
<u>Discussion Session 11</u>

TA: Ramkumar Ravi

This document contains detailed solutions to selected problems from the final exam of spring 2006.

<u>Hamming Code</u>

PART A

- Hamming distance between **01101101** and **01010110**
  - ✓ **01101101 ^ 01010110 = 00111011**
    Count the number of 1s in the result above and that is the Hamming Distance (5)

PART B

- Minimum Hamming distance needed between a pair of valid code words to detect a single bit error
  - ✓ A code is capable of **t** error detection, iff minimum HD of the code is **t+1**
    Hence, answer is 2

PART C

- Minimum Hamming distance needed between a pair of valid code words to correct a single bit error
  - ✓ A code is capable of **t** error correction, iff minimum HD of the code is 2**t+1**
    Hence, answer is 3

PART D

- Minimum Hamming distance needed between a pair of valid code words to correct a single bit error and detect a double bit error
  - ✓ A code is capable of **d** error detection and **t** error correction, iff minimum HD of the code is **t + d + 1**
    Hence, answer is 4

PART E

	C_1	C_2	b_1	C_3	b_2	b_3	b_4	C_4
H =	1	0	1	0	1	0	1	0
	0	1	1	0	0	1	1	0
	0	0	0	1	1	1	1	0
	1	1	1	1	1	1	1	1

C1 = checks for parity across bits b1, b2 and b4
C2 = checks for parity across bits b1, b3 and b4
C3 = checks for parity across bits b2, b3 and b4

Assuming *odd parity*, given data word $b_1b_2b_3b_4$ = 0011, we find the code word

- C1 checks *odd parity* across b1, b2 and b4. In this case check *odd parity* across 0, 0 and 1. There is already off number of 1's. So **C1 should be 0** (if not add a 1 to make total number of 1's odd)
- C2 checks *odd parity* across b1, b3 and b4. In this case check *odd parity* across 0, 1 and 1. There is an even number of 1's. So **C2 should be 1** (add a 1 to make total number of 1's odd)
- C3 checks *odd parity* across b2, b3 and b4. In this case check *odd parity* across 0, 1 and 1. There is an even number of 1's. So **C3 should be 1** (add a 1 to make total number of 1's odd)
- C4 checks for parity across entire combination
  $C_1C_2b_1C_3b_2b_3b_4$ = **0101011**
  There is an even number of 1's. So **C4 should be 1** (add a 1 to make total number of 1's odd)

So code word is
$C_1C_2b_1C_3b_2b_3b_4C_4$ = **0101 0111**

PART F

- Word read from memory is $C_1C_2b_1C_3b_2b_3b_4C_4$ = **0101 0101**
- Data word is $b_1b_2b_3b_4$ = 0010
- Check bits are $C_1C_2C_3$ = 011

From the table above, calculate the original check bits for the data word 0010
C1 = check across 0, 0 and 0; So **C1 = 1**
C2 = check across 0, 1 and 0; So **C2 = 0**
C3 = check across 0, 1 and 0; So **C3 = 0**

Now Syndrome =original check bits **XOR** given check bits = 001 ^ 110 = 111
Hence bit 7 is in error

PART G

- Procedure for detecting a double error
  - ✓ Parity is OK and Syndrome is non-zero

PART H

- In the code word **0101 0101**, the parity indicated by the LSB (C_4) is *NOT OK*. With odd parity, this bit should have been a 0
- We already observed in Part F that syndrome is *non-zero* (**111**)
- Hence, a parity *NOT OK* and syndrome *NON-ZERO* indicates that there is a 1-bit error in the position indicated by the syndrome

BOOTH ENCODING

Current Bit	Bit to right	Encoding
0	0	0
0	1	1
1	0	-1
1	1	0

Note that the table above (for 1-bit Booth encoding) follows the same format as the truth table of XOR gate (for inputs 1, 0 use -1)

PART A

- Obtain Booth encoding for $(-47)_{10}$
- STEP 1: Obtain binary representation for -47 = 1101 0001

Using table, obtain 1-bit encoding starting from LSB
Current bit – LSB = 1
Bit to right – Always assume bit to right of LSB is 0
So combination is **10 -> Encoding is -1**

Move left
Current bit = 0
Bit to right = 1
So combination is **01 -> Encoding is 1**

Move left
Current bit = 0
Bit to right = 0
So combination is **00 -> Encoding is 0**

Move left
Current bit = 0
Bit to right = 0
So combination is **00 -> Encoding is 0**

Move left
Current bit = 1
Bit to right = 0
So combination is **00 -> Encoding is -1**

Move left
Current bit = 0
Bit to right = 1
So combination is **01 -> Encoding is 1**

Move left
Current bit = 1
Bit to right = 0

So combination is **10 -> Encoding is -1**

Move left
Current bit = MSB = 1
Bit to right = 1
So combination is **11 -> Encoding is 0**

Result is 0 -1 1 -1 0 0 1 -1

PART B

- Obtain 2-bit Booth encoding by grouping the 1-bit result into pairs
- **(0 -1), (1 -1), (0 0), (1 -1)**
- In every pair refer to the first value as X and second value as X-1 and use the formula 2*X + X-1

(0 -1) = 2*0 + -1 = -1
(1 -1) = 2*1 + -1 = 1
(0 0) = 2*0 + 0 = 0
(1 -1) = 2*1 + -1 = 1

Result is -1 1 0 1

PART C

- Given 2-bit Booth encoding **0 -2 0 2 -1 2 -1 1**, find the multiplier. Use the table below

Current 2 bits (X1, X2)	Bit to right (Y1)	Math	Result
00	0	[X1 X2] = [0 0] => 0 [X2 Y1] = [0 0] => 0 2*[X1 X2]+[X2 Y1] = 0	0
00	1	[X1 X2] = [0 0] => 0 [X2 Y1] = [0 1] => 1 2*[X1 X2]+[X2 Y1] = 1	1
01	0	[X1 X2] = [0 1] => 1 [X2 Y1] = [1 0] => -1 2*[X1 X2]+[X2 Y1] = 1	1
01	1	[X1 X2] = [0 1] => 1 [X2 Y1] = [1 1] => 0 2*[X1 X2]+[X2 Y1] = 2	2
10	0	[X1 X2] = [1 0] => -1 [X2 Y1] = [0 0] => 0 2*[X1 X2]+[X2 Y1] = -2	-2
10	1	[X1 X2] = [1 0] => -1 [X2 Y1] = [0 1] => 1 2*[X1 X2]+[X2 Y1] = -1	-1
11	0	[X1 X2] = [1 1] => 0 [X2 Y1] = [1 0] => -1 2*[X1 X2]+[X2 Y1] = -1	-1
11	1	[X1 X2] = [1 1] => 0 [X2 Y1] = [1 1] => 0 2*[X1 X2]+[X2 Y1] = 0	0

Start with the LSB of the 2-bit booth encoding **1**

- ➔ Remember that the bit to the right of the LSB of the multiplier is always 0
- ➔ So with bit to right 0, what 2 current bits will give us encoding **1** ?
- ➔ From table, combination is 010
- ➔ This means **1** represents **01** in multiplier

Now, we have found the 2 LSBs of the multiplier 01

- ➔ The next two bits left to 01 has to be found
- ➔ So with bit to right as 0, what combination of current 2 bits will give us encoding **-1** ?
- ➔ From table combination is 110
- ➔ This means **-1** represents **11** in the multiplier

Now we have found the 4 LSBs of the multiplier **1101**

- ➔ The next two bits left to 11 has to be found
- ➔ So with bit to right as 1, what combination of current 2 bits will give us encoding **2** ?
- ➔ From table combination is 011
- ➔ This means **2** represents **01** in the multiplier

Our multiplier now is **01 1101**

- ➔ The next two bits left to 01 has to be found
- ➔ So with bit to right as 0, what combination of current 2 bits will give us encoding **-1** ?
- ➔ From table combination is 110
- ➔ This means **-1** represents **11** in the multiplier

Our multiplier now is **1101 1101**

- ➔ The next two bits left to 11 has to be found
- ➔ So with bit to right as 1, what combination of current 2 bits will give us encoding **2** ?
- ➔ From table combination is 011
- ➔ This means **2** represents **01** in the multiplier

Our multiplier now is **01 1101 1101**

- ➔ The next two bits left to 01 has to be found
- ➔ So with bit to right as 0, what combination of current 2 bits will give us encoding **0** ?
- ➔ From table combination is 000
- ➔ This means **0** represents **00** in the multiplier

Our multiplier now is **0001 1101 1101**

- ➔ The next two bits left to 00 has to be found
- ➔ So with bit to right as 0, what combination of current 2 bits will give us encoding **-2** ?
- ➔ From table combination is 100
- ➔ This means **-2** represents **10** in the multiplier

Our multiplier now is **10 0001 1101 1101**

- ➔ The next two bits left to 10 has to be found
- ➔ So with bit to right as 1, what combination of current 2 bits will give us encoding **0** ?
- ➔ From table combination is 111
- ➔ This means **0** represents **11** in the multiplier

Final multiplier is **1110 0001 1101 1101**