CS 271
Computer Architecture & Assembly Language

Lecture 8
Error detection/correction
Modularization and MASM Procedures
1/27/21, Thursday
Due Reminders

• Program #3
  • Due 1/30 11:59 PM on Canvas

• Weekly Summary Exercise 4
  • Due 1/30 11:59 PM on Canvas
Lecture Topics:

• Error-detecting and error-correcting codes
• Modularization
• MASM Procedures
Error-detecting and error-correcting codes
Simple Error Checking

- **Parity** is the total number of ‘1’ bits (including the extra parity bit) in a binary code.

- Each computer architecture is designed to use either **even parity** or **odd parity**.

- System adds a **parity bit** to make each code match the system’s parity.
Parity (error checking)

• Example parity bits for 8-bit code 11010110
  • Even parity system: 111010110 (sets parity bit to 1 to make a total of 6 1-bits)
  • Odd parity system: 011010110 (sets parity bit to 1 to keep 5 1-bits)

• Code is checked for parity error whenever it is used.

• Examples for even-parity architecture:
  • 101010101 error (5 one-bits)
  • 100101010 OK (4 one-bits)

• Examples for odd-parity architecture:
  • 101010101 OK (5 one-bits)
  • 100101010 error (4 one-bits)
Parity (error checking)

- Used for checking memory, network transmissions, etc.
  - Error detection

- Not 100% reliable
  - Works only when error is in odd number of bits
  - ... but very good because most errors are single-bit
A very short game

• For each of the following screens: \[ \begin{array}{c}
\end{array} \]
  • Write down the letter of the screen only if your birth date is on the screen
<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>3</th>
<th>5</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>11</td>
<td>13</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>19</td>
<td>21</td>
<td>23</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>27</td>
<td>29</td>
<td>31</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>6</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>11</td>
<td>14</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>19</td>
<td>22</td>
<td>23</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>27</td>
<td>30</td>
<td>31</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>----</td>
<td>-----</td>
<td>------</td>
<td>------</td>
<td>------</td>
</tr>
<tr>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>21</td>
<td>22</td>
<td>23</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>29</td>
<td>30</td>
<td>31</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>25</td>
<td>26</td>
<td>27</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>29</td>
<td>30</td>
<td>31</td>
<td></td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
</tr>
<tr>
<td>---</td>
<td>-------------</td>
<td>-------------</td>
<td>-------------</td>
<td>-------------</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td>2</td>
<td>12</td>
<td>13</td>
</tr>
<tr>
<td>3</td>
<td>11</td>
<td>11</td>
<td>13</td>
<td>21</td>
</tr>
<tr>
<td>4</td>
<td>13</td>
<td>21</td>
<td>14</td>
<td>23</td>
</tr>
<tr>
<td>5</td>
<td>15</td>
<td>22</td>
<td>15</td>
<td>23</td>
</tr>
<tr>
<td>6</td>
<td>17</td>
<td>23</td>
<td>17</td>
<td>23</td>
</tr>
<tr>
<td>7</td>
<td>19</td>
<td>23</td>
<td>19</td>
<td>23</td>
</tr>
<tr>
<td>8</td>
<td>21</td>
<td>24</td>
<td>20</td>
<td>24</td>
</tr>
<tr>
<td>9</td>
<td>23</td>
<td>25</td>
<td>22</td>
<td>25</td>
</tr>
<tr>
<td>10</td>
<td>25</td>
<td>26</td>
<td>23</td>
<td>26</td>
</tr>
<tr>
<td>11</td>
<td>27</td>
<td>27</td>
<td>24</td>
<td>26</td>
</tr>
<tr>
<td>12</td>
<td>29</td>
<td>28</td>
<td>25</td>
<td>27</td>
</tr>
<tr>
<td>13</td>
<td>30</td>
<td>28</td>
<td>26</td>
<td>28</td>
</tr>
<tr>
<td>14</td>
<td>31</td>
<td>29</td>
<td>27</td>
<td>29</td>
</tr>
<tr>
<td>15</td>
<td></td>
<td>30</td>
<td>28</td>
<td>30</td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
<td>16</td>
<td>31</td>
</tr>
</tbody>
</table>
### A

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>00001</td>
<td>00011</td>
<td>00101</td>
<td>00111</td>
<td>01001</td>
<td>01011</td>
<td>01101</td>
<td>01111</td>
</tr>
<tr>
<td>01001</td>
<td>01011</td>
<td>01101</td>
<td>01111</td>
<td>10001</td>
<td>10011</td>
<td>10101</td>
<td>10111</td>
</tr>
<tr>
<td>10001</td>
<td>10011</td>
<td>10101</td>
<td>10111</td>
<td>11001</td>
<td>11011</td>
<td>11101</td>
<td>11111</td>
</tr>
<tr>
<td>11001</td>
<td>11011</td>
<td>11101</td>
<td>11111</td>
<td>11111</td>
<td>11111</td>
<td>11111</td>
<td>11111</td>
</tr>
</tbody>
</table>

### B

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>00010</td>
<td>00011</td>
<td>00110</td>
<td>00111</td>
<td>01010</td>
<td>01011</td>
<td>01110</td>
<td>01111</td>
</tr>
<tr>
<td>01010</td>
<td>01011</td>
<td>01110</td>
<td>01111</td>
<td>10010</td>
<td>10011</td>
<td>10110</td>
<td>10111</td>
</tr>
<tr>
<td>10010</td>
<td>10011</td>
<td>10110</td>
<td>10111</td>
<td>11010</td>
<td>11011</td>
<td>11110</td>
<td>11111</td>
</tr>
<tr>
<td>11010</td>
<td>11011</td>
<td>11110</td>
<td>11111</td>
<td>11111</td>
<td>11111</td>
<td>11111</td>
<td>11111</td>
</tr>
</tbody>
</table>

### C

<table>
<thead>
<tr>
<th></th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>00100</td>
<td>00101</td>
<td>00110</td>
<td>00111</td>
<td></td>
</tr>
<tr>
<td>01100</td>
<td>01101</td>
<td>01110</td>
<td>01111</td>
<td></td>
</tr>
<tr>
<td>10100</td>
<td>10101</td>
<td>10110</td>
<td>10111</td>
<td></td>
</tr>
<tr>
<td>11100</td>
<td>11101</td>
<td>11110</td>
<td>11111</td>
<td></td>
</tr>
</tbody>
</table>

### D

<table>
<thead>
<tr>
<th></th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>01000</td>
<td>01001</td>
<td>01010</td>
<td>01011</td>
<td></td>
</tr>
<tr>
<td>01100</td>
<td>01101</td>
<td>01110</td>
<td>01111</td>
<td></td>
</tr>
<tr>
<td>10100</td>
<td>10101</td>
<td>10110</td>
<td>10111</td>
<td></td>
</tr>
<tr>
<td>11100</td>
<td>11101</td>
<td>11110</td>
<td>11111</td>
<td></td>
</tr>
</tbody>
</table>

### E

\[
E = 2^3 + 2^1 + 2^2 + 2^4 \\
= 1 + 2 + 4 + 16
\]
Error-correcting: Hamming Codes

- n-bit code word \( (n = m + r) \)
  - m data bits
  - r check bits (to check parity)
- There are \( 2^n \) possible code words
- Only \( 2^m \) code words are valid

• Parity is the sum of one check bit and its selected data bits
  • May be even or odd
  • Used for detecting and correcting errors in memory, network transmissions, etc.
    • ECC memory, etc.
Parity check for single-bit errors

- Number of parity bits depends on word size
  - Number of required parity bits (r) is $\log_2 m + 1$
  
  \[
  = \log_2 8 + 1 = 3 + 1 = 4
  \]

- Guarantees **Hamming distance** of 2
  - i.e., to change one valid code to another valid code, **at least 2 bits must be changed**
  - If a valid code gets only one bit changed, the resulting code is invalid

- There are many invalid codes
  - Invalid codes indicate errors
Arranging the Parity Bits

- For 8 data bits, how many parity bits should be added?
- Number the bits left → right, 1→n
  - Note: different from usual numbering
- Bits numbered with powers of 2 are parity bits; others are data bits.

<table>
<thead>
<tr>
<th>p</th>
<th>p</th>
<th>d</th>
<th>p</th>
<th>d</th>
<th>d</th>
<th>d</th>
<th>p</th>
<th>d</th>
<th>d</th>
<th>d</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- \(2^0\) - 1 bit
- \(2^1\) - 1 bit
- \(2^2\) - 2 bits
- \(2^3\) - 4 bits
Hamming code Example (p1)

- Represent decimal 45 as 8-bit with **even** parity Hamming code.
- \( m = 8, \ r = (\log_2 8 + 1) = 4, \) so \( n = 12 \)
- \( 45 = 00101101 \) binary (8-bit)
- Fill in data bits, skipping the parity bits

\[
\begin{array}{ccccccccc}
<table>
<thead>
<tr>
<th>p</th>
<th>p</th>
<th>d</th>
<th>p</th>
<th>d</th>
<th>d</th>
<th>d</th>
<th>p</th>
<th>d</th>
<th>d</th>
<th>d</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>?</td>
<td>?</td>
<td>0</td>
<td>?</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>?</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
\end{array}
\]
Hamming code Example (p2)

- Parity bit #1 represents all place numbers having 1 in the 1’s place (i.e., all odd-numbered places)
- Even parity requires that the count of ‘1’ bits in these places (plus the parity bit) must be even.
  - There is one ‘1’ data bit in these places, so set bit #1 to 1

<table>
<thead>
<tr>
<th>p</th>
<th>p</th>
<th>d</th>
<th>p</th>
<th>d</th>
<th>d</th>
<th>d</th>
<th>p</th>
<th>d</th>
<th>d</th>
<th>d</th>
<th>d</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>0001</td>
<td>0010</td>
<td>0011</td>
<td>0100</td>
<td>0101</td>
<td>0110</td>
<td>0111</td>
<td>1000</td>
<td>1001</td>
<td>1010</td>
<td>1011</td>
<td>1100</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>?</td>
<td>0</td>
<td>?</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>?</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
Hamming code Example (p3)

- Parity bit #2 represents all place numbers having 1 in the 2’s place
- There are two ‘1’ data bits in these places, so set bit #2 to 0

<table>
<thead>
<tr>
<th>p</th>
<th>p</th>
<th>d</th>
<th>p</th>
<th>d</th>
<th>d</th>
<th>d</th>
<th>p</th>
<th>d</th>
<th>d</th>
<th>d</th>
<th>d</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>0001</td>
<td>0010</td>
<td>0011</td>
<td>0100</td>
<td>0101</td>
<td>0110</td>
<td>0111</td>
<td>1000</td>
<td>1001</td>
<td>1010</td>
<td>1011</td>
<td>1100</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>?</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>?</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>


Hamming code Example (p4)

- Parity bit #4 represents all place numbers having 1 in the 4’s place
- There are two ‘1’ data bits in these places, so set bit #4 to 0

<table>
<thead>
<tr>
<th>p</th>
<th>p</th>
<th>d</th>
<th>p</th>
<th>d</th>
<th>d</th>
<th>d</th>
<th>p</th>
<th>d</th>
<th>d</th>
<th>d</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>0001</td>
<td>0010</td>
<td>0011</td>
<td>0100</td>
<td>0101</td>
<td>0110</td>
<td>0111</td>
<td>1000</td>
<td>1001</td>
<td>1010</td>
<td>1011</td>
<td>1100</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>?</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Hamming code Example (p5)

- Parity bit #8 represents all place numbers having 1 in the 8’s place
- There are three ‘1’ data bits in these places, so set bit #8 to 1

<table>
<thead>
<tr>
<th>p</th>
<th>p</th>
<th>d</th>
<th>p</th>
<th>d</th>
<th>d</th>
<th>d</th>
<th>p</th>
<th>d</th>
<th>d</th>
<th>d</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>0001</td>
<td>0010</td>
<td>0011</td>
<td>0100</td>
<td>0101</td>
<td>0110</td>
<td>0111</td>
<td>1000</td>
<td>1001</td>
<td>1010</td>
<td>1011</td>
<td>1100</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Hamming code Example (p6)

• 45 = 00101101 (8-bit binary)
• 45 = 100001011101 (12-bit even parity Hamming Code)
Hamming Code Error Example (p0)

• 100111110111 is a 12-bit odd-parity representation. Correct its single-bit error

\[
\begin{array}{cccccccccccc}
p & p & d & p & d & d & d & p & d & d & d & d \\
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \\
1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1
\end{array}
\]

Data bits are
01110111 = 119
Hamming Code Error Example (p1)

• 100111110111 is a 12-bit odd-parity representation. Correct its single-bit error

\[
\begin{array}{cccccccccccc}
p & p & d & p & d & d & d & p & d & d & d & d \\
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \\
\hline
1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1
\end{array}
\]

• 1s parity X

Data bits are 01110111 = 119
Hamming Code Error Example (p2)

• 100111110111 is a 12-bit odd-parity representation. Correct its single-bit error

\[
\begin{array}{cccccccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \\
1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 \\
\end{array}
\]

• 1s parity X
• 2s parity X

Data bits are 01110111 = 119
Hamming Code Error Example (p3)

• 100111110111 is a 12-bit odd-parity representation. Correct its single-bit error

\[
\begin{array}{cccccccccccc}
p & p & d & p & d & d & d & p & d & d & d & d \\
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \\
1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\
\end{array}
\]

• 1s parity X
• 2s parity X
• 4s parity V

Data bits are 01110111 = 119
Hamming Code Error Example (p4)

- 100111110111 is a 12-bit odd-parity representation. Correct its single-bit error

\[
\begin{array}{cccccccccccc}
p & p & d & p & d & d & d & p & d & d & d & d \\
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \\
1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\
\end{array}
\]

- 1s parity X
- 2s parity X
- 4s parity √
- 8s parity X

Data bits are 01110111 = 119
Hamming Code Error Example (p5)

- 100111110111 is a 12-bit odd-parity representation. Correct its single-bit error

- The only bit that is in 1s and 2s and 8s and is NOT in 4s is bit number 11. Therefore, the number should be 100111110101.

- The data bits should be 01110101 = 117

Data bits are
01110111 = 119
• Regardless of external representation, all I/O eventually is converted into electrical (binary) codes.
• Inside the computer, everything is represented by gates (open/closed)
• Since the number of gates in each group (byte, word, etc.) is finite, computers can represent numbers only within a finite range.

• Representations may be truncated; overflow/underflow can occur, and the Status Register will be set.

• Limited precision for floating-point representations.
• Inside the computer
  • Bytes, words, etc., can represent a finite number of combinations of off/on switches.
  • Each distinct combination is called a code.
  • Each code can be used to represent:
    • Number value
    • Memory address
    • Machine instruction
    • Keyboard character

• **Representation is neutral**. The operating system and the programs decide how to interpret the codes.
You should be able to show the binary/hexadecimal representations of:

- Integer values (signed / unsigned)
- Characters (tables given on tests, don’t memorize)
- Floating-point values
- Error-detecting codes (parity)
- Error-correcting codes (Hamming)

You should be able to convert representations:

- Binary $\rightarrow$ Decimal
- Decimal $\rightarrow$ Hexadecimal
- Hexadecimal $\rightarrow$ Binary
Modularization and MASM Procedures
Modular Development (motivations)

• Team environment
  • Easier to divide the work and assign tasks
• Incremental testing
  • Easier to test “section” as you develop the program
• Reliability
  • Easier to verify (procedures/parameters)
• Debugging
  • Easier to find errors
• Maintenance
  • Easier to make changes
• Re-usable code
  • Easier to use library modules (don’t re-invent)
MASM Procedures

• The “sections” of your main program are like modules (for modular development)
• Convert the “modules” into procedures
• Convert the **main** procedure into the **control module**
  • The control module calls the procedures
    • **main** should be (mostly) procedure calls
  • Any procedure may call another procedure
    • ... or call itself (recursion: more later)
• Top-Down Design (functional decomposition) involves the following:
  • Design your program before starting to code
  • Separate the program into its major tasks
  • Break large tasks into smaller ones
  • Use a hierarchical structure based on procedure calls
  • Test individual procedures separately
Creating Procedures

• A **procedure** is the assembly equivalent of a Java **method** or C/C++ **function**.

• Example procedure named **sample**:

```
sample PROC
  .
  .
  ret
sample ENDP
```

• Note that it looks a lot like **main**
  • Has **ret** instead of **exit**
CALL and RET Instructions

• The **CALL** instruction calls a procedure
  1. Pushes the **offset** of the **next instruction** in the calling procedure onto the **system stack**
     - i.e., pushes the **current address** in the **EIP** register onto the system stack (recall the **Instruction Execution Cycle**?
  2. Copies the **address** of the **called procedure** into **EIP**
  3. Executes the called procedure until **RET**

• The **RET** instruction returns control to the calling procedure
  • Pops the top of stack into **EIP**
    • i.e., execution continues in the calling procedure at the instruction after the **CALL**

• **Note:** Much more later about the system stack
Programming with Procedures

1. Write `main` to decide what variables and procedures are needed
2. Write procedure stubs
   - `test`
3. Declare variables
   - `test`
4. Implement procedures one at a time
   - Test each procedure separately
   - Recommended order of implementation:
     - Output, input, process
5. Do the housekeeping (document procedures)
   - `test`
Example program: Using Top-Down Design

- Get 2 integers (a and b) from the user
- Find the summation of integers in [a...b]
- Display the result