Error-Correction Codes

CS6025 Data Encoding

Yizong Cheng

4-7-15

Block Codes

- Cyclic redundancy check (CRC) appends a checksum to data so that the codeword is a multiple of $G(x)$, the generator.
- Addition of codewords forms other codewords.
- All codewords are multiples of $G(x)$ and thus form a cyclic group.
- When the codeword size is fixed, we may transmit a block of them along with a parity word.
- For instance, we have codewords A, B, C, D, E, F and the Parity word $P = A + B + C + D + E + F$.
- (n,k) block codes have n bits in a block with k of them for data.

Syndrome

- Suppose we received a block.
- We can compute the syndrome as
- $S = A + B + C + D + E + F – P$.
- If $S = 0$, no error is indicated.
- Otherwise if we also know which codeword is wrong (based on its own CRC?), we can subtract the syndrome S from that received word and recover the correct codeword.
- $S = A + B + C + D + dD + E + F – P$ is not zero and $D + dD$ is wrong
- $D = (D + dD) – S$ corrects the error.

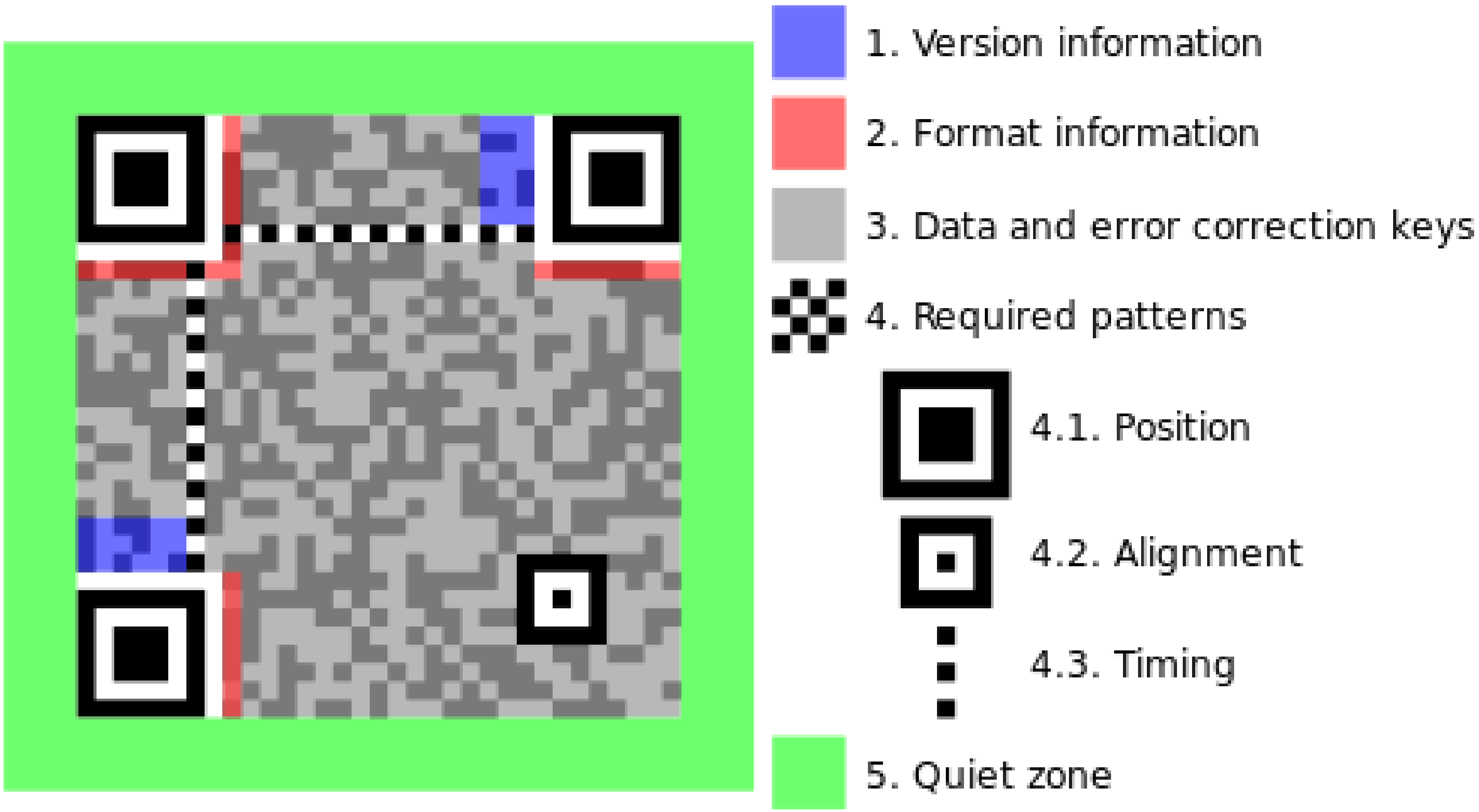
Double Parity Code

- $P = A + B + C + D + E + F$
- $Q = 6A + 5B + 4C + 3D + 2E + F$
- $S1 = A + B + C + D + E + F - P$
- $S2 = 6A + 5B + 4C + 3D + 2E + F - Q$
- Assume one of the words may contain error.
- If $S1 = 0$ but $S2 != 0$, then Q contains an error.
- If $S2 = 0$ but $S1 != 0$, then P contains an error.
- If $S1 = S2$, F is wrong; if $2S1 = S2$, E is wrong; if $3S1 = S2$, D is wrong;
- if $4S1 = S2$, C is wrong; if $5S1 = S2$, B is wrong; if $6S1 = S2$, A is wrong.

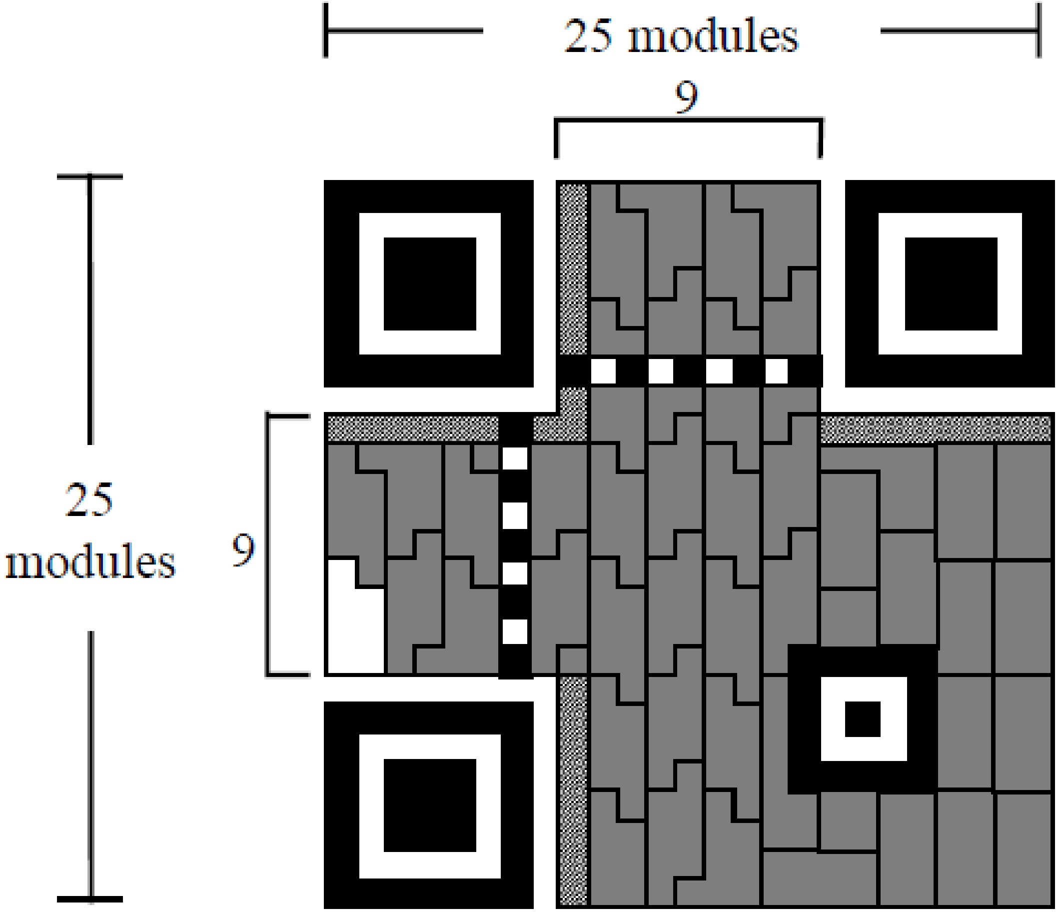
Hamming Codes

- minimum Hamming distance between codewords of 3 provides single-error correctability (move to the nearest codeword), and double-error detectability.
- (7,4) Hamming code: data bits A, B, C, D and parity bits P = B + C + D, Q = A + C + D, and R = A + B + D. (addition in Z_2)
- Check matrix M = [0111100, 1011010, 1101001]
- Transmit ABCDPQR as a 7-bit vector and the receiver receives V and computes the syndrome S = MV.
- If S is not zero and matches a column of M, then the bit corresponding to the column is in error and it to be flipped.

QR Code


1. Version information
2. Format information
3. Data and error correction keys
4. Required patterns
4.1. Position
4.2. Alignment
4.3. Timing
5. Quiet zone

Version 2 25x25 QR Code Layout



Version 2

Format Information

- Format information is a 15-bit string with the first five bits the message and the remaining 10 bits error correcting bits.
- The generator polynomial over GF(2) is $G(x) = x^{10} + x^8 + x^5 + x^4 + x^2 + x + 1$.
- The error correcting bits are computed so that the codeword is a multiple of $G(x)$.
- This is a block code (15, 5).
- It is then XORed with the Mask Pattern 101010000010010.

The 32 Codewords

000000000000000
000010100110111
000101001101110
000111101011001
001000111101011
001010011011100
001101110000101
001111010110010
010001111010110
010011011100001
010100110111000
010110010001111
011001000111101
011011100001010
011100001010011
011110101100100

100001010011011
100011110101100
100100011110101
100110111000010
101001101110000
101011001000111
101100100011110
101110000101001
110000101001101
110010001111010
110101100100011
110111000010100
111000010100110
111010110010001
111101011001000
111111111111111

Codewords are Multiples of G(x)

0	000000000000000	21	100001010011011
1	000010100110111	20	100011110101100
2	000101001101110	23	100100011110101
3	000111101011001	22	100110111000010
5	001000111101011	16	101001101110000
4	001010011011100	17	101011001000111
7	001101110000101	18	101100100011110
6	001111010110010	19	101110000101001
10	010001111010110	31	110000101001101
11	010011011100001	30	110010001111010
8	010100110111000	29	110101100100011
9	010110010001111	28	110111000010100
15	011001000111101	26	111000010100110
14	011011100001010	27	111010110010001
13	011100001010011	24	111101011001000
12	011110101100100	25	111111111111111

Linear Block Code

- If u and v are codewords, then u + v is also a codeword.
  - addition is XOR here.
- Codewords are linear combinations of a basis of five codewords.

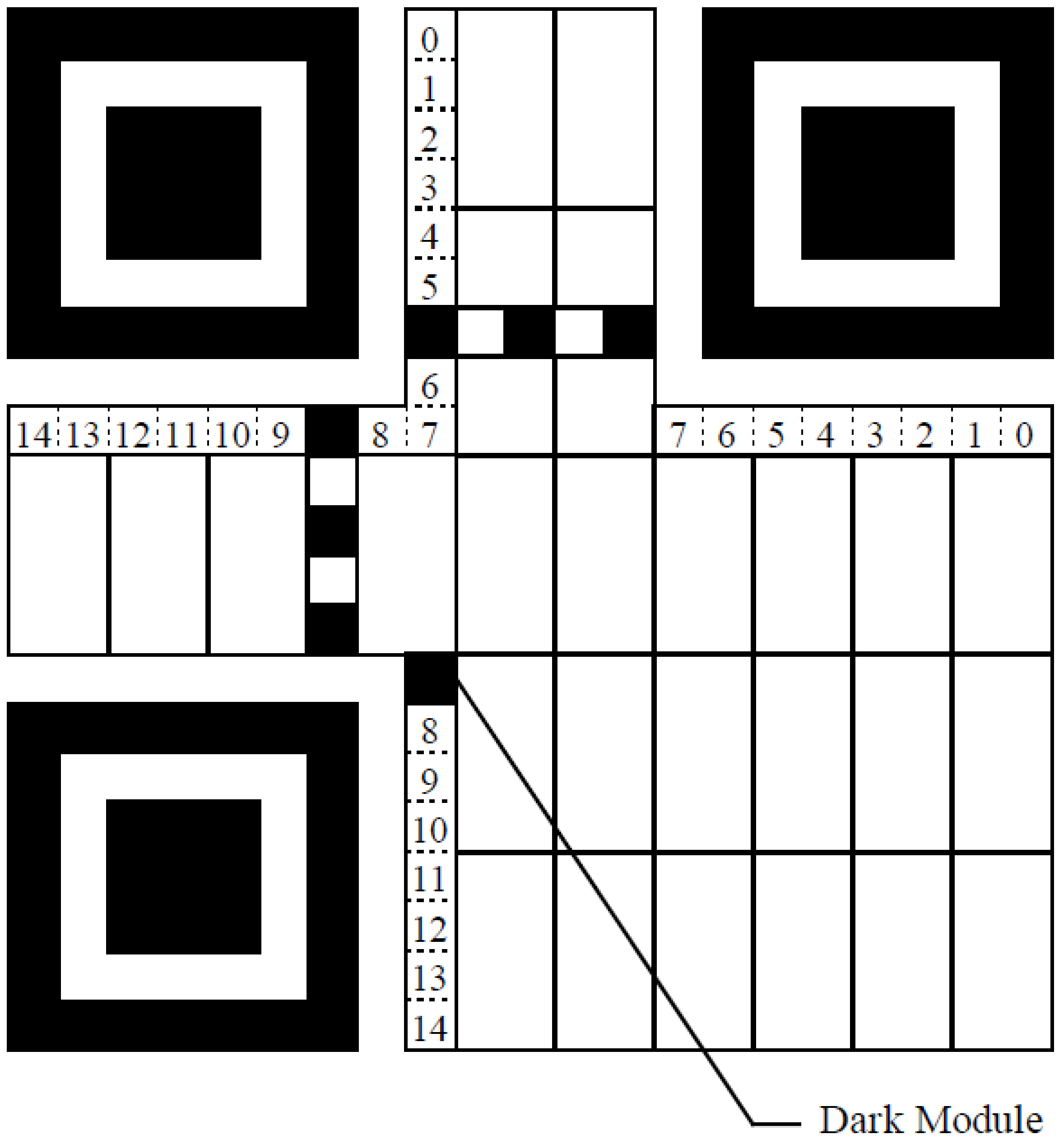
000010100110111
000101001101110
001000111101011
010001111010110
100001010011011

Cyclic Codes

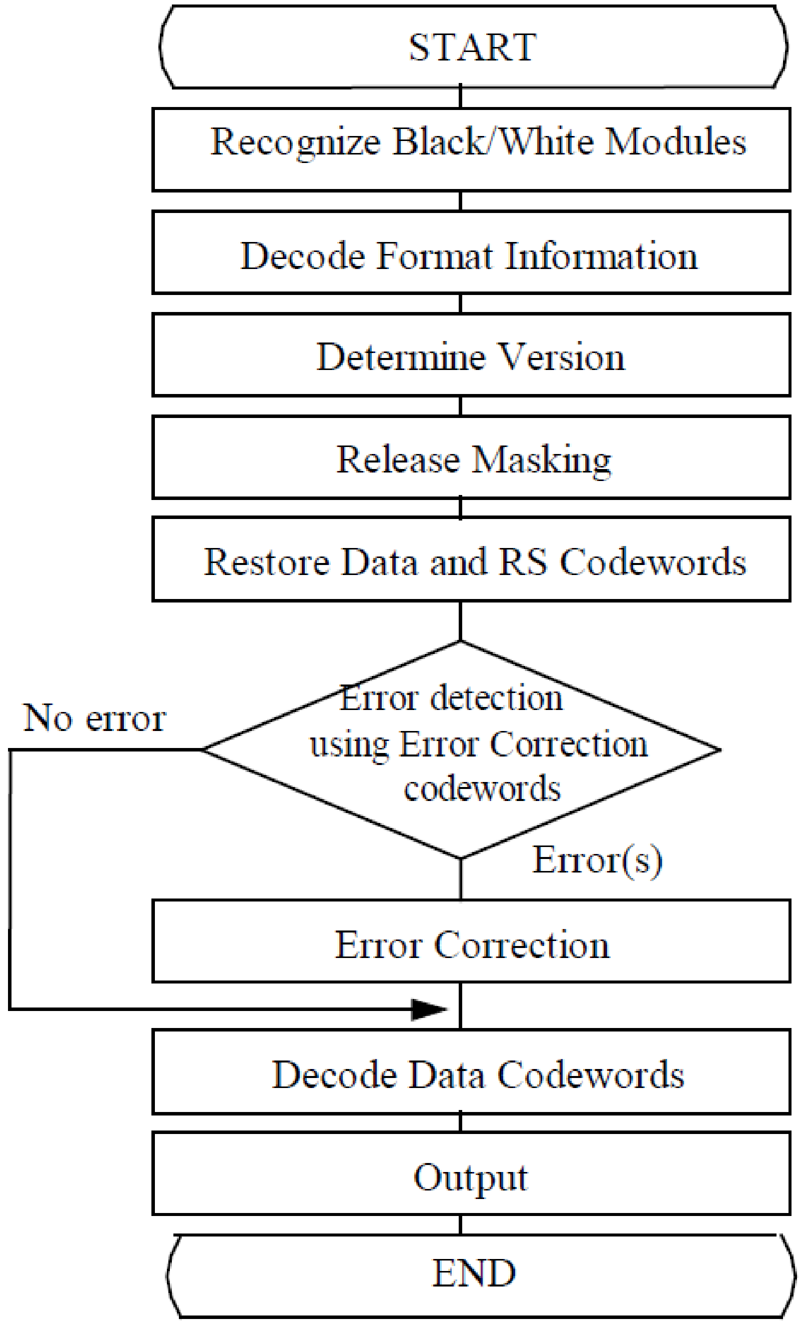
	000000000000000	111111111111111	
1	000010100110111	000111101011001	3
2	000101001101110	001111010110010	6
4	001010011011100	011110101100100	12
8	010100110111000	111101011001000	24
16	101001101110000	111010110010001	27
11	010011011100001	110101100100011	29
22	100110111000010	101011001000111	17
7	001101110000101	010110010001111	9
14	011011100001010	101100100011110	18
28	110111000010100	011001000111101	15
19	101110000101001	110010001111010	30
13	011100001010011	100100011110101	23
26	111000010100110	001000111101011	5
31	110000101001101	010001111010110	10
21	100001010011011	100011110101100	20

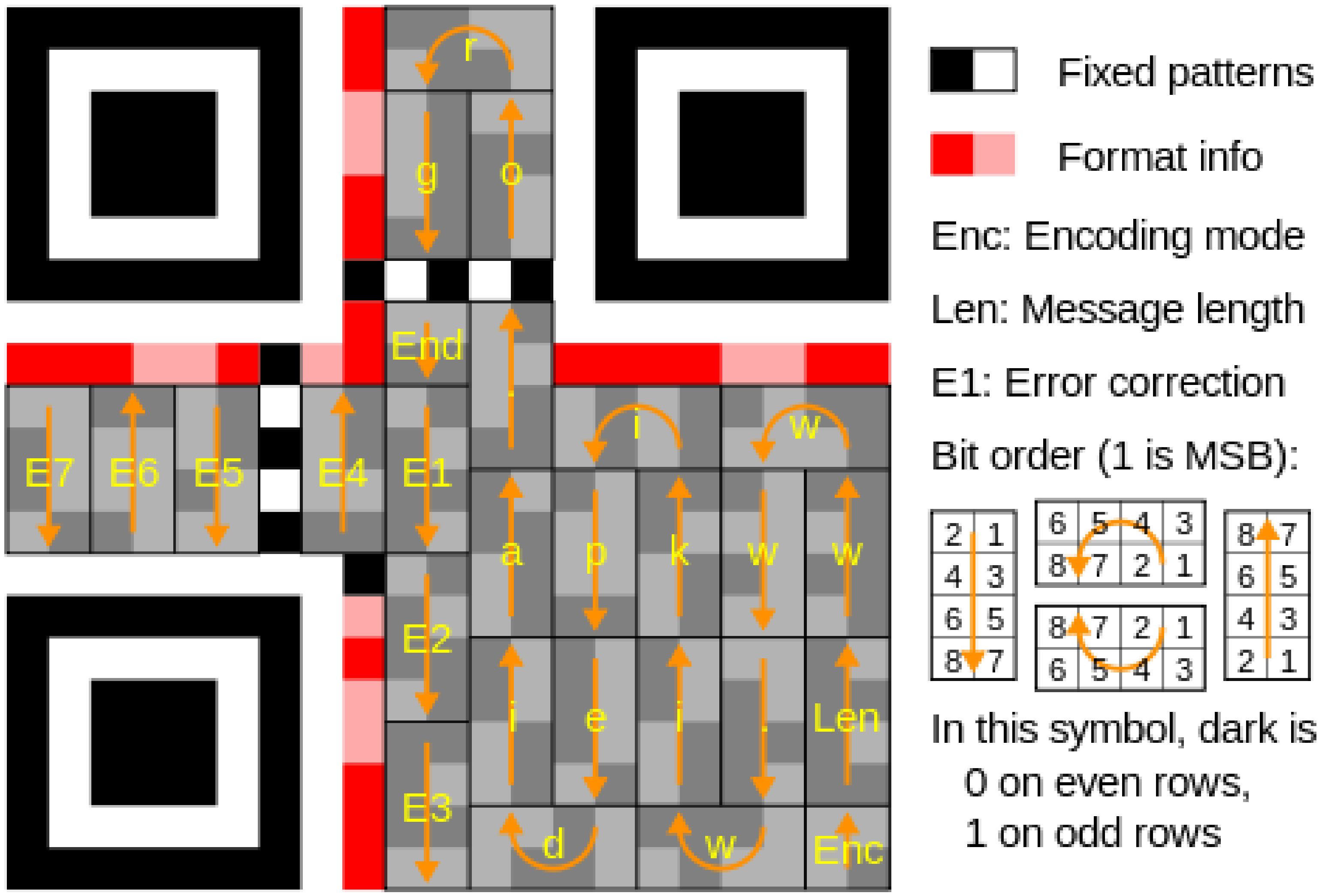
BCH (15,5) Code

- The Bose-Chaudhuri-Hocquenghem (15,5) code is used for error correction for the format information.
- 15 is a rare number that is $2^m - 1$ for ($m = 4$) that allows 7 as the minimum Hamming distance between codewords.
  - BCH codes like this are called primitive codes
- One can find a unique nearest codewords for up to three errors.
  - a triple error correcting code
- Decoding: The bit string from the 32codewords closest to the bit string read from the symbol is taken, provided the strings differ by 3 bits or less.

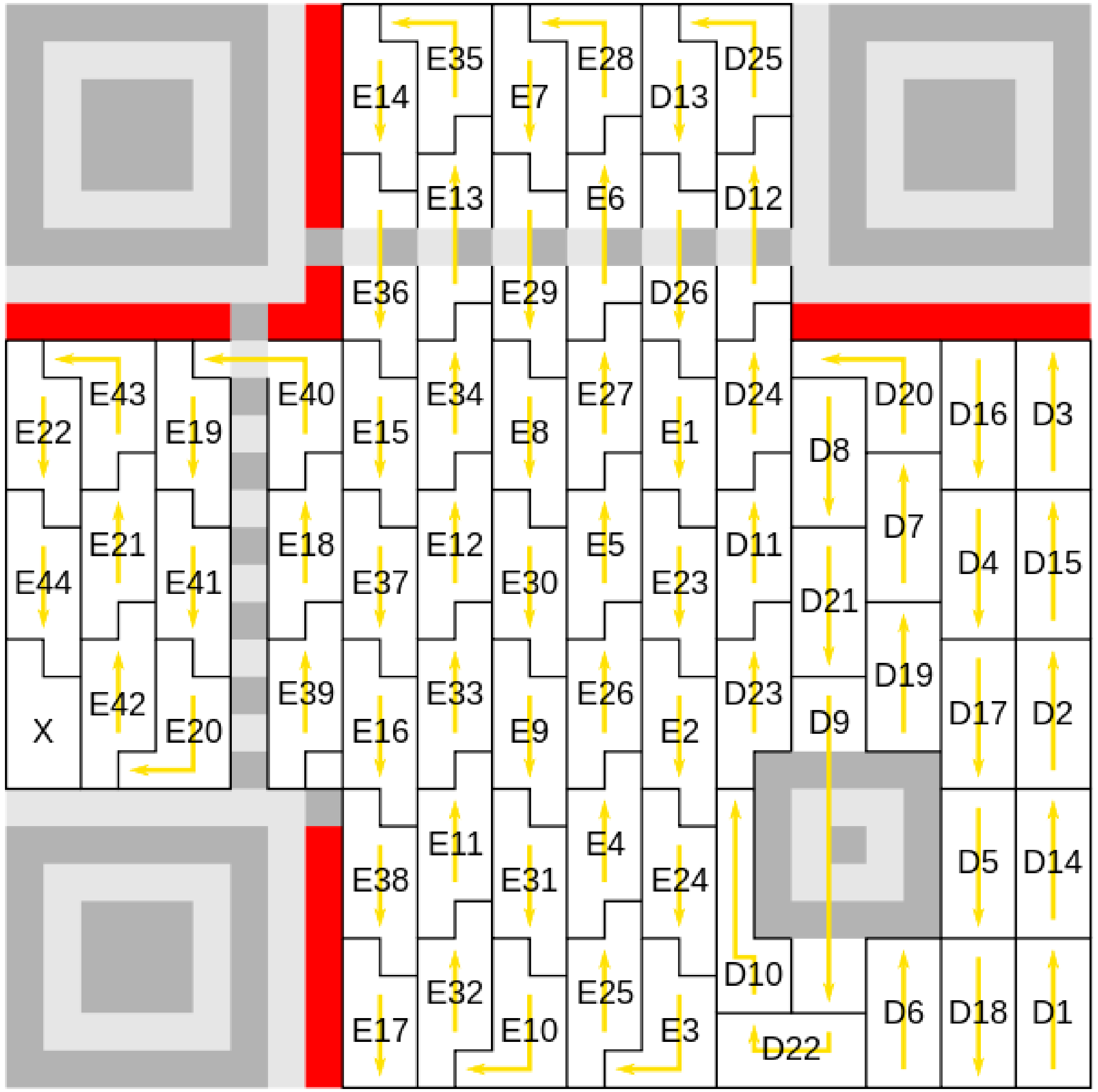


0
1
2
3
4
5
6
14 13 12 11 10 9
8 7
7 6 5 4 3 2 1 0
8
9
10
11
12
13
14
Dark Module


START
Recognize Black/White Modules
Decode Format Information
Determine Version
Release Masking
Restore Data and RS Codewords
Error detection using Error Correction codewords
No error
Error(s)
Error Correction
Decode Data Codewords
Output
END



r
g
o
End
E7
E6
E5
E4
E1
i
w
a
p
k
w
w
E2
i
e
i
Len
E3
d
w
Enc
Fixed patterns
Format info
Enc: Encoding mode
Len: Message length
E1: Error correction
Bit order (1 is MSB):
In this symbol, dark is
0 on even rows,
1 on odd rows



Fixed Patterns   Format Info

D: Data, E: Error Correction, X: Unused

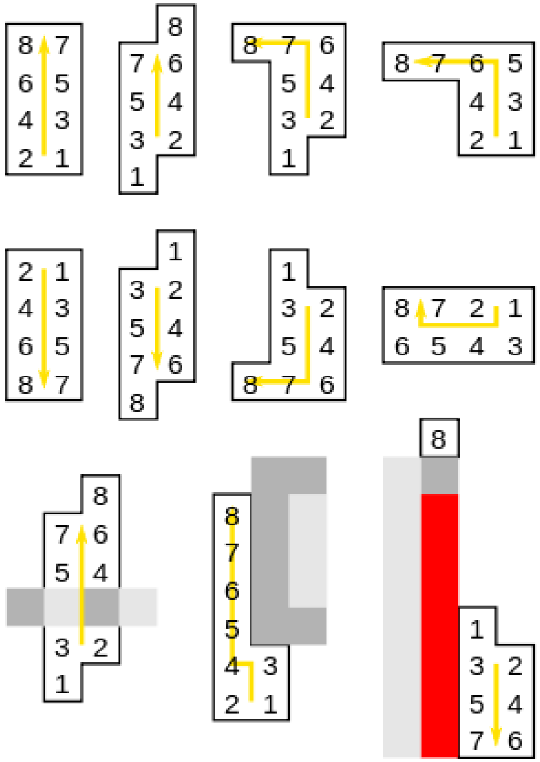
Error Correction Level H is shown

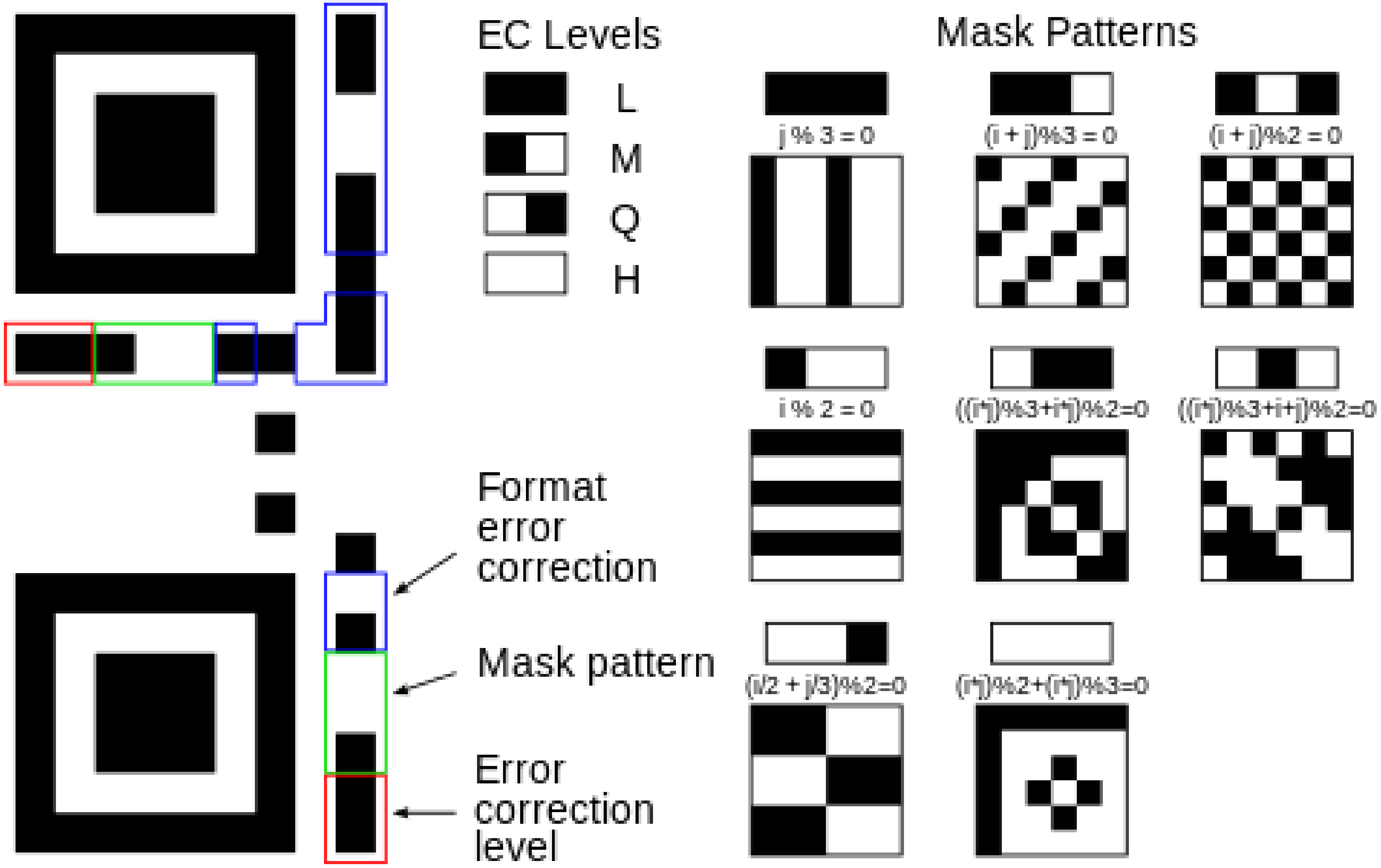
Block 1 Codewords: D1–D13, E1–E22

Block 2 Codewords: D14–D26, E23–E44

Message Data: D1–D13, D14–D26

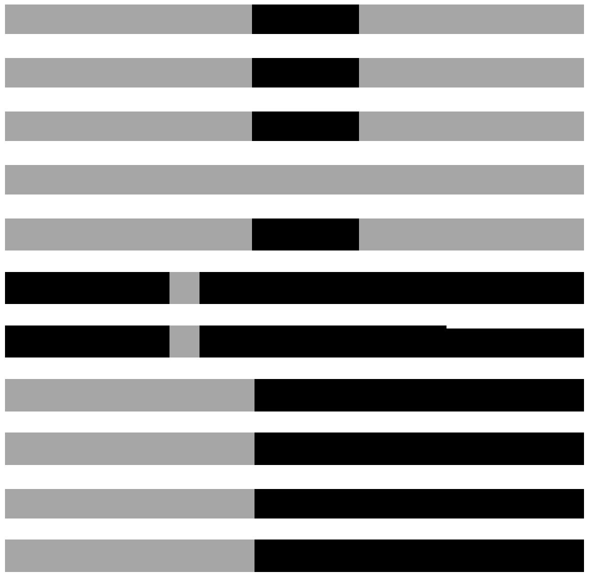
Bit order (1 is the most significant bit):





EC Levels
L
M
Q
H
Mask Patterns
j % 3 = 0
(i + j)%3 = 0
(i + j)%2 = 0
i % 2 = 0
((i*j)%3+i*j)%2=0
((i*j)%3+i+j)%2=0
(i/2 + j/3)%2=0
(i*j)%2+(i*j)%3=0
Format error correction
Mask pattern
Error correction level

Data Masking for Version 1 Symbol



001

$i \bmod 2 = 0$

```
XXXXXXX   XX  XXXXXXX
X     X X   X X     X
X XXX X X   X X XXX X
X XXX X X   X X XXX X
X XXX X X XXX X XXX X
X     X X X   X     X
XXXXXXX X X X XXXXXXX

  X  XXXXXXX X XXXXX
   XXX XXXXXX X  X  X
XX XX XX  X X X   X X
 XXX    X  X X  XX  X
XXX   XX  XXXX      X
        XX  XX X   X
XXXXXXX XXXXXX XXX  X
X     X XX  X XX
X XXX X   X  X XX   X
X XXX X    XXXX
X XXX X XX   XX   XXX
X     X  XX X X X
XXXXXXX   XX  X  XX X
```

```
public class H21{

  static final int maxSize = 200;
  static final int formatLength = 15;
  String[] rawBitmap = new String[maxSize];
  int numberOfLines = 0;
  int version = 0;
  int width = 0;
  int height = 0;
  boolean[][] matrix = null;
  boolean[] format = new boolean[formatLength];
  boolean[] dataBitStream = null;
  int dataSpace = 0;

  void readRawBitmap(){
    Scanner in = new Scanner(System.in);
    while (in.hasNextLine())
      rawBitmap[numberOfLines++] = in.nextLine();
  }
```

```
void getMatrix(){
    int firstRow = 0;
    for (; firstRow < numberOfLines; firstRow++)
      if (rawBitmap[firstRow].indexOf("XXXXXXX ") >= 0) break;
    int leftPos = rawBitmap[firstRow].indexOf ("XXXXXXX ");
    int rightPos = rawBitmap[firstRow].lastIndexOf(" XXXXXXX");
    width = rightPos + 8 - leftPos;
    height = width;
    version = (width - 17) / 4;
    matrix = new boolean[height][width];
    for (int i = 0; i < height; i++)
     for (int j = 0; j < width; j++)
       matrix[i][j] =
        rawBitmap[firstRow + i].charAt(leftPos + j) == 'X';
    dataSpace = width * height - 3 * 64 - 2 * 15
      - 2 * (width - 16) - 1;
    dataBitStream = new boolean[dataSpace];
  }
```

```
void getFormatInformation(){
    for (int i = 0; i < 6; i++) format[i] = matrix[8][i];
    format[6] = matrix[8][7]; format[7] = matrix[8][8];
    format[8] = matrix[7][8];
    for (int i = 0; i < 6; i++)
     format[formatLength - 1 - i] = matrix[i][8];

    for (int i = 0; i < formatLength; i++)
     if (format[i]) System.out.print("1");
     else System.out.print("0");
    System.out.println();

  }
```

Table C.1 — Valid format information bit sequences

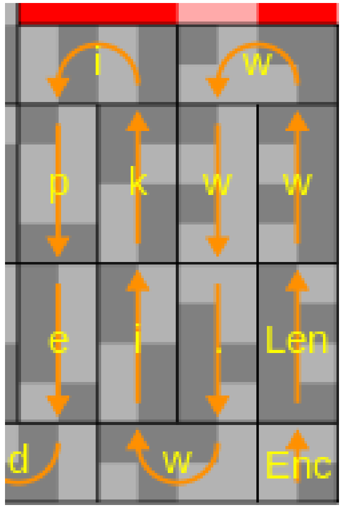
Sequence before masking		Sequence after masking (QR Code symbols)		Sequence after masking (Micro QR Code symbols)	
Data bits	Error correction bits	binary	hex	binary	hex
00000	0000000000	101010000010010	5412	100010001000101	4445
00001	0100110111	101000100100101	5125	100000101110010	4172
00010	1001101110	101111001111100	5E7C	100111000101011	4E2B
00011	1101011001	101101101001011	5B4B	100101100011100	4B1C
00100	0111101011	100010111111001	45F9	101010110101110	55AE
00101	0011011100	100000011001110	40CE	101000010011001	5099
00110	1110000101	100111110010111	4F97	101111111000000	5FC0
00111	1010110010	100101010100000	4AA0	101101011110111	5AF7
10000	1010011011	001011010001001	1689	000011011011110	06DE
10001	1110101100	001001110111110	13BE	000001111101001	03E9
10010	0011110101	001110011100111	1CE7	000110010110000	0CB0
10011	0111000010	001100111010000	19D0	000100110000111	0987
10100	1101110000	000011101100010	0762	001011100110101	1735

Demasking for mask 001 (i%2=0)

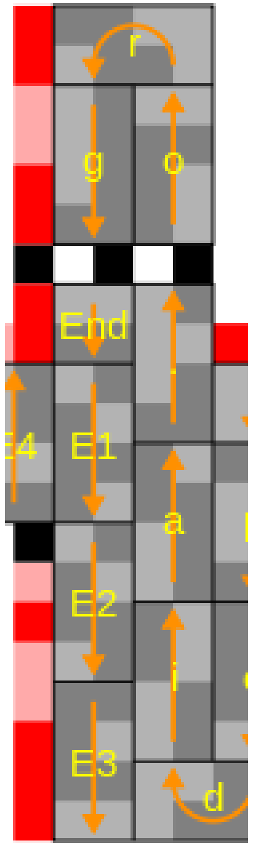
```
void demask(){
  for (int i = 0; i < 9; i++) if (i != 6)
   for (int j = 9; j < width - 8; j++)
    if (i % 2 == 0) matrix[i][j] = !matrix[i][j];
  for (int i = 9; i < height - 8; i++)
    for (int j = 0; j < width; j++) if (j != 6)
      if (i % 2 == 0) matrix[i][j] = !matrix[i][j];
  for (int i = height - 8; i < height; i++)
    for (int j = 9; j < width; j++)
      if (i % 2 == 0) matrix[i][j] = !matrix[i][j];
}
```

Getting Data Bit Stream

```
void getDataBitStream(){
  int n = 0;
  for (int i = 0; i < 4; i++){
    boolean up = (i % 2) == 0;
    int x = width - 1 - 2 * i;
    if (up)
      for (int y = height - 1; y >= 9; y--){
        dataBitStream[n++] = matrix[y][x];
        dataBitStream[n++] = matrix[y][x - 1];
      }
    else for (int y = 9; y < height; y++){
        dataBitStream[n++] = matrix[y][x];
        dataBitStream[n++] = matrix[y][x - 1];
    }
  }
```

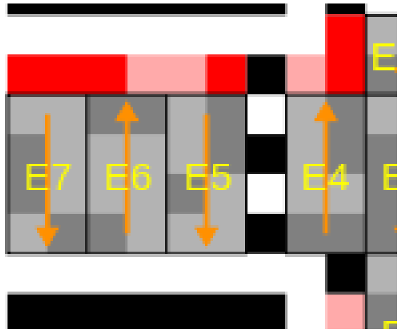


The Middle Section




```
int middleWidth = (width - 17) / 2;
   for (int i = 0; i < middleWidth; i++){
     boolean up = (i % 2) == 0;
     int x = width - 9 - 2 * i;
     if (up)
       for (int y = height - 1; y >= 0; y--){
         if (y != 6){
         dataBitStream[n++] = matrix[y][x];
         dataBitStream[n++] = matrix[y][x - 1];
         }
       }
     else for (int y = 0; y < height; y++) if (y != 6){
         dataBitStream[n++] = matrix[y][x];
         dataBitStream[n++] = matrix[y][x - 1];
     }
   }
```

The Third Section of Data Bit Stream




```
for (int i = 0; i < 4; i++){
     boolean up = (i % 2) == 0;
     int x = i == 0 ? 8 : 7 - 2 * i;
     if (up)
       for (int y = height - 9; y >= 9; y--){
         dataBitStream[n++] = matrix[y][x];
         dataBitStream[n++] = matrix[y][x - 1];
       }
     else for (int y = 9; y <= height - 9; y++){
         dataBitStream[n++] = matrix[y][x];
         dataBitStream[n++] = matrix[y][x - 1];
     }
   }
 }
```

Codeword Capacity

Table 1 — Codeword capacity of all versions of QR Code 2005

Version	No. of Modules/ side (A)	Function pattern modules (B)	Format and version information modules (C)	Data modules except (C) ($D=A^2-B-C$)	Data capacity [codewords][a] (E)	Remainder Bits
M1	11	70	15	36	5	0
M2	13	74	15	80	10	0
M3	15	78	15	132	17	0
M4	17	82	15	192	24	0
1	21	202	31	208	26	0
2	25	235	31	359	44	7
3	29	243	31	567	70	7
4	33	251	31	807	100	7
5	37	259	31	1 079	134	7

Mode Indicator and Terminator

Table 2 — Mode indicators for QR Code 2005

Mode	**QR Code symbols**	**Micro QR Code symbols**			
Version	**all**	**M1**	**M2**	**M3**	**M4**
Mode indicator length (bits)	**4**	**0**	**1**	**2**	**3**
ECI	**0111**	n/a	n/a	n/a	n/a
Numeric	**0001**	n/a	**0**	**00**	**000**
Alphanumeric	**0010**	n/a	**1**	**01**	**001**
Byte	**0100**	n/a	n/a	**10**	**010**
Kanji	**1000**	n/a	n/a	**11**	**011**
Structured Append	**0011**	n/a	n/a	n/a	n/a
FNC1	**0101** (1st position) **1001** (2nd position)	n/a	n/a	n/a	n/a
Terminator (End of Message) [a]	**0000**	**000**	**00000**	**0000000**	**000000000**

[a] The Terminator is not a mode indicator as such

- Mode indicator
- Character count indicator
- Data bit stream

Data Capacity (EC Level Based)

Table 7 — Number of symbol characters and input data capacity for QR Code 2005

Version	Error correction level	Number of data codewords	Number of data bits	Data capacity			
				Numeric	Alphanumeric	Byte	Kanji
M1	Error Detection only	3	20	5	-	-	-
M2	L	5	40	10	6	-	-
	M	4	32	8	5	-	-
M3	L	11	84	23	14	9	6
	M	9	68	18	11	7	4
M4	L	16	128	35	21	15	9
	M	14	112	30	18	13	8
	Q	10	80	21	13	9	5
1	L	19	152	41	25	17	10
	M	16	128	34	20	14	8
	Q	13	104	27	16	11	7
	H	9	72	17	10	7	4
2	L	34	272	77	47	32	20
	M	28	224	63	38	26	16
	Q	22	176	48	29	20	12
	H	16	128	34	20	14	8

Placement of Input Data

- The total number of codewords in the message shall always be equal to the total number of codewords capable of being represented in the symbol.
- The message bit stream shall be extended to fill the data capacity of the symbol corresponding to the Version and Error Correction Level, as defined in Table 8, by adding the Pad Codewords 11101100 and 00010001 alternately.

Get Message

```
int nextSymbol(int position, int bitSize){
    int result = 0;
    for (int i = 0; i < bitSize; i++){
      result <<= 1;
      if (dataBitStream[position + i]) result |= 1;
    }
    return result;
 }

 void getMessage(){
   int mode = nextSymbol(0, 4);
   int messageLength = nextSymbol(4, 8);
   for (int i = 0; i < messageLength; i++)
    // print out message
}
```

Homework 21: due 4-12-15

- Complete H21.java and run it on a version 1 symbol (test21.txt).
- Give an explanation on the format information.
- Print out the message.

Table 13 — Error correction characteristics for versions 1 to 6

Version	Total number of codewords	Error correction level	Number of error correction codewords	Number of error correction blocks	Error correction code per block [a]
1	26	L	7	1	(26,19,2) [b]
		M	10	1	(26,16,4) [b]
		Q	13	1	(26,13,6) [b]
		H	17	1	(26,9,8) [b]
2	44	L	10	1	(44,34,4) [b]
		M	16	1	(44,28,8)
		Q	22	1	(44,22,11)
		H	28	1	(44,16,14)
3	70	L	15	1	(70,55,7) [b]
		M	26	1	(70,44,13)
		Q	36	2	(35,17,9)
		H	44	2	(35,13,11)