第四章 快速傅里叶变换

Fast Forurier Transform

第四章　快速傅里叶变换

Fast Forurier Transform

4.2　基于时间抽取的基-2-FFT快速算法

基于时间抽取的基-2-FFT快速算法的编程方法

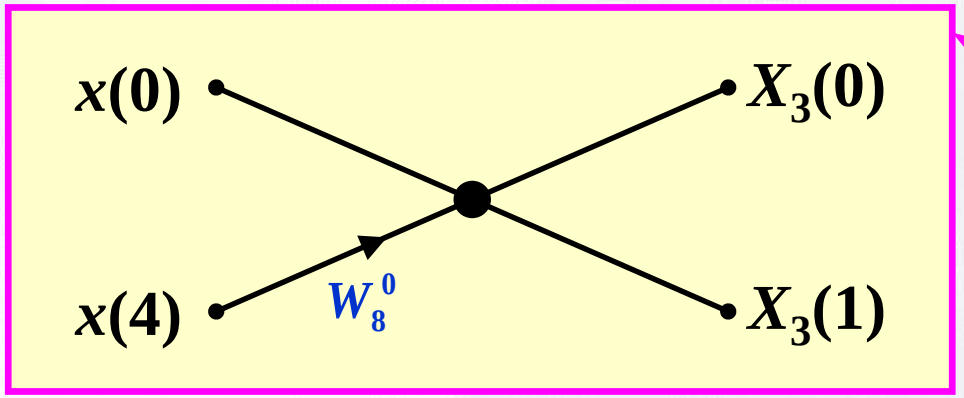
华东理工大学信息科学与工程学院　万永菁

一、原位运算（亦称同址计算） *In place computation*

FFT的每级计算都是由N个复数数据(输入)两两构成一个蝶型运算而得到另外N个复数数据(输出)。

当数据输入到存储器以后，每一组运算的结果，仍然存放在这同一组存储器中直到最后输出。

例：将$x(0)$放在单元$A(0)$中，将$x(4)$放在单元$A(1)$中，W_8^0放在一个暂存器中。



将$x(0) + W_8^0 x(4)$
→ 送回$A(0)$单元

将$x(0) - W_8^0 x(4)$
→ 送回$A(1)$单元

二、旋转因子的变化规律

如上所述，N点DIT—FFT运算流图中，每级都有$N/2$个蝶形。每个蝶形都要乘以因子W_N^P，称其为旋转因子，p称为旋转因子的指数。

观察FFT运算流图发现，第L级共有2^{L-1}个不同的旋转因子。$N=2^3=8$时的各级旋转因子表示如下：

$L=1$时，$W_N^p=W_{N/4}^J$，　$N/4=2^1=2^L$，　$J=0$

$L=2$时，　$W_N^p=W_{N/2}^J$，　$N/2=2^2=2^L$，　$J=0$，1

$L=3$时，　$W_N^p=W_N^J$，　$N=2^3=2^L$，　$J=0$，1，2，3

对$N=2^M$的一般情况，第L级的旋转因子为：

$$\because W_N^P = W_{2^L}^J \quad J = 0,1,\ldots,2^{L-1}-1$$

$$2^L = 2^M \cdot 2^{L-M} = N \cdot 2^{L-M}$$

$$\therefore W_N^P = W_{N\cdot 2^{L-M}}^J = W_N^{J\cdot 2^{M-L}} \quad J = 0,1,\ldots,2^{L-1}-1$$

$$p = J \cdot 2^{M-L}$$

4.2 基于时间抽取的基-2-FFT快速算法

设序列$x(n)$经时域奇偶分组抽选后，存入数组A中。如果蝶形运算的两个输入数据相距B个点($B=2^{L-1}$)，应用原位计算，则蝶形运算可表示成如下形式：

$$A_L(J) \Leftarrow A_{L-1}(J) + A_{L-1}(J+B)W_N^p$$

$$A_L(J+B) \Leftarrow A_{L-1}(J) - A_{L-1}(J+B)W_N^p$$

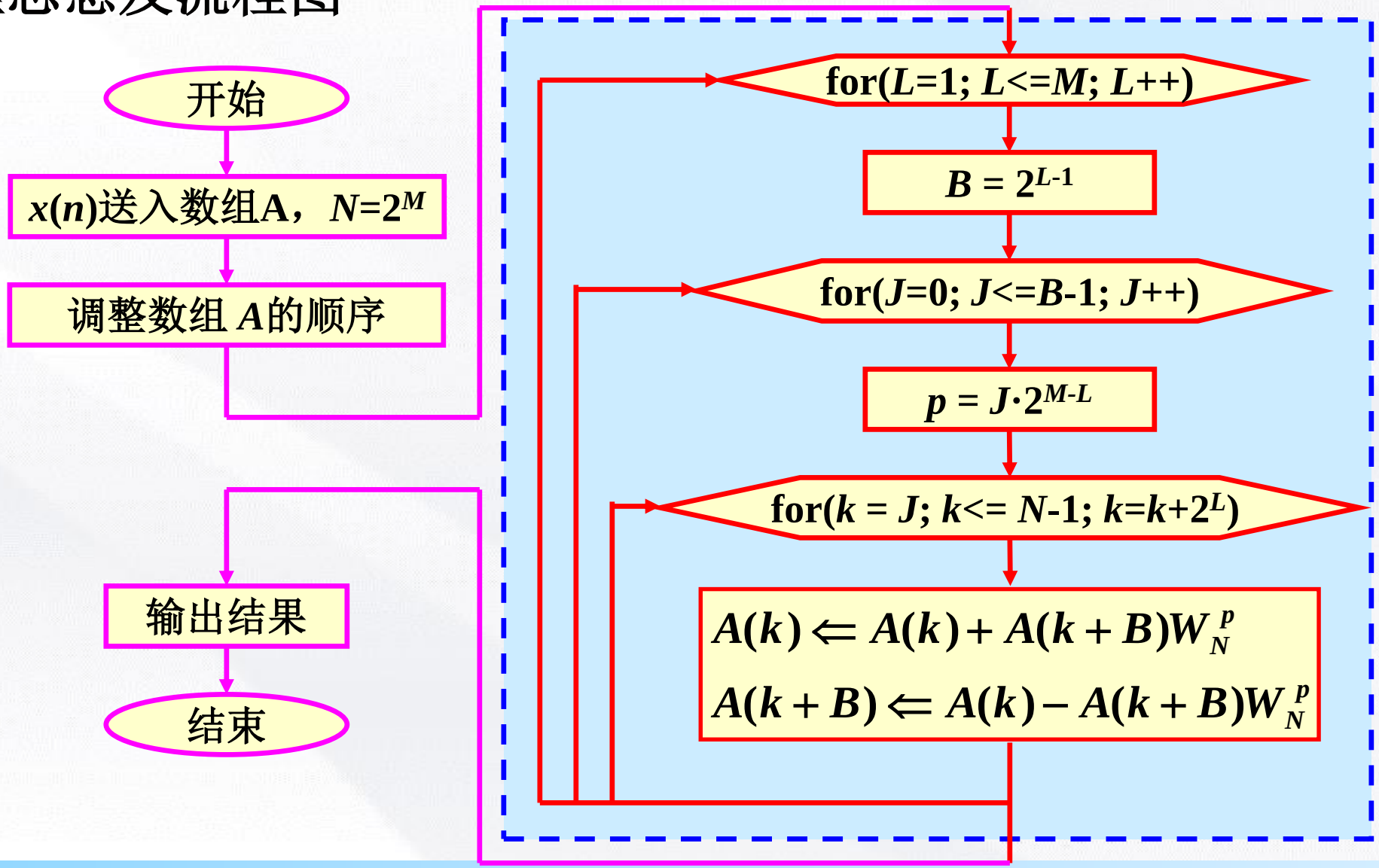
式中：$p = J \cdot 2^{M-L}$，　$J = 0,1,\ldots,2^{L-1}-1$，　$L = 1,2,\ldots,M$

下标L表示第L级运算，$A_L(J)$则表示第L级运算后数组元素$A(J)$的值。

基2-DIT-FFT算法的编程思路

- 程序的入口参数：时域信号 $x(n)$
- 输入序列的顺序：码位倒序
- 蝶形运算的级数、每一级的运算对象及旋转因子
- 程序的输出：输入信号的频谱信息 $X(k)$

三、编程思想及流程图


开始
x(n)送入数组A，N=2^M
调整数组 A的顺序
for(L=1; L<=M; L++)
B = 2^(L-1)
for(J=0; J<=B-1; J++)
p = J·2^(M-L)
for(k = J; k<= N-1; k=k+2^L)
A(k) ⇐ A(k) + A(k + B)W_N^p
A(k + B) ⇐ A(k) − A(k + B)W_N^p
输出结果
结束

数组A中为已经倒序的x(n)

for(L=1; L<=M; L++)

$B = 2^{L-1}$

for(J=0; J<=B-1; J++)

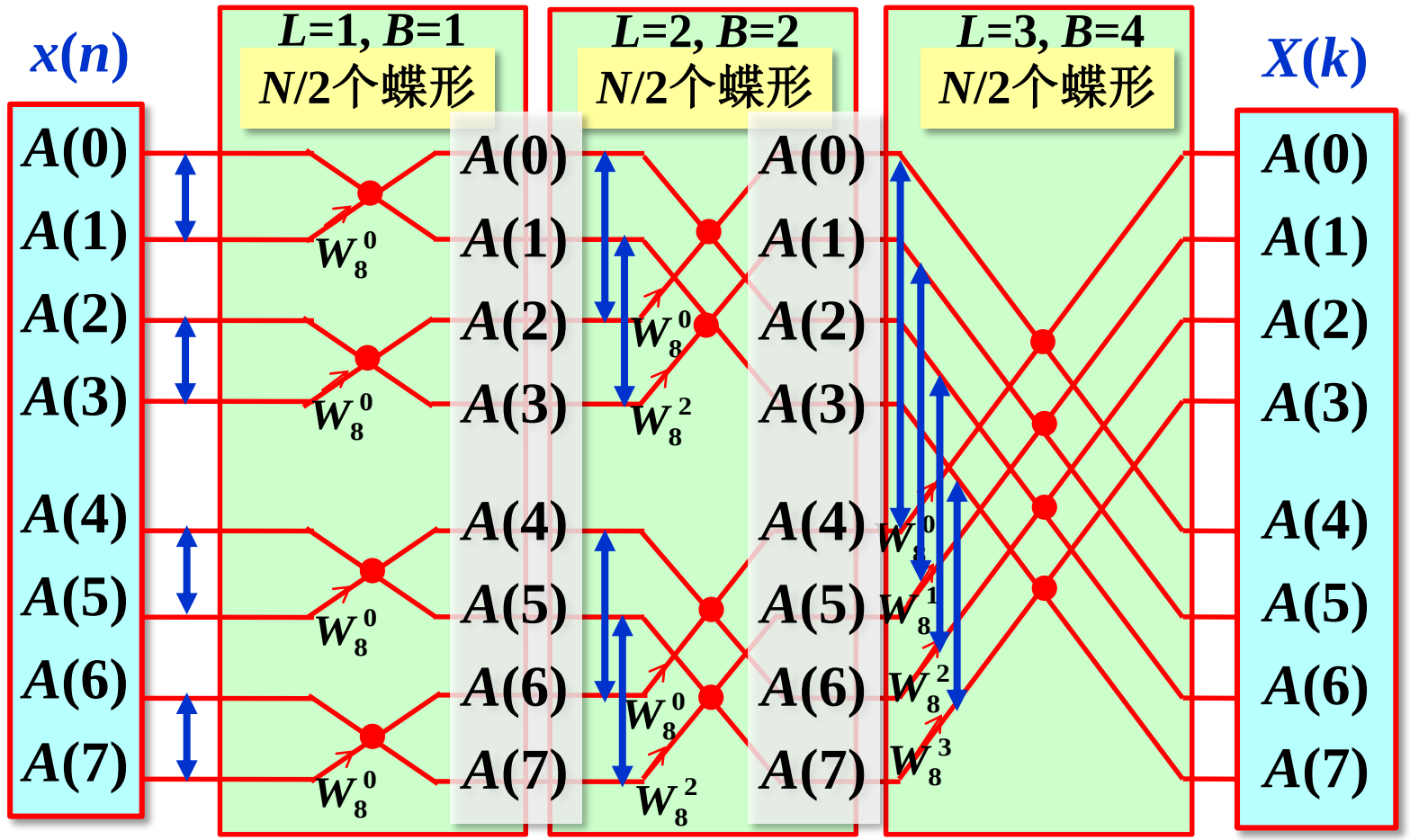
$p = J \cdot 2^{M-L}$

for(k = J; k<= N-1; k=k+2^L)

$A(k) \Leftarrow A(k) + A(k+B)W_N^p$

$A(k+B) \Leftarrow A(k) - A(k+B)W_N^p$

数组A中为X(k)



L=1, B=1 :
- J=0, p=0:
  - k=0,1 k=2,3
  - k=4,5 k=6,7

L=2, B=2 :
- J=0, p=0:
  - k=0,2 k=4,6
- J=1, p=2:
  - k=1,3 k=5,7

L=3, B=4 :
- J=0, p=0:
  - k=0,4
- J=1, p=1:
  - k=1,5
- J=2, p=2:
  - k=2,6
- J=3, p=3:
  - k=3,7

四、码位倒序 *bit-reversed order*

由$N=8$蝶形图看出：原位计算时，FFT输出的$X(k)$的次序正好是顺序排列的，即$X(0)\ldots X(7)$,但输入$x(n)$都不能按自然顺序存入到存储单元中，而是按$x(0)$、$x(4)$、$x(2)$、$x(6)$、$x(1)$、$x(5)$、$x(3)$、$x(7)$的顺序存入存储单元，即为乱序输入，顺序输出。

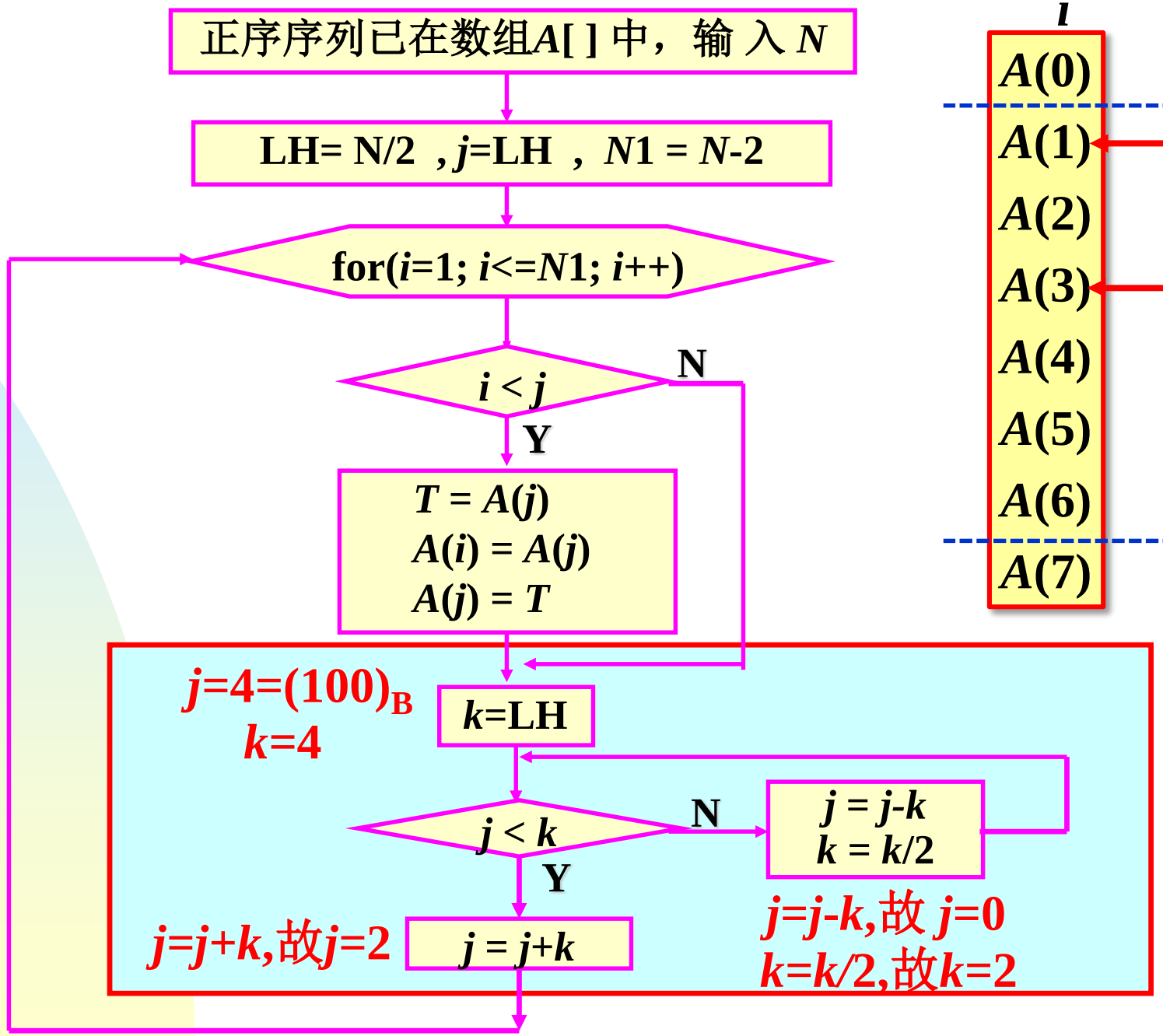
这种顺序看起来相当杂乱，然而它是有规律的。即码位倒读规则。

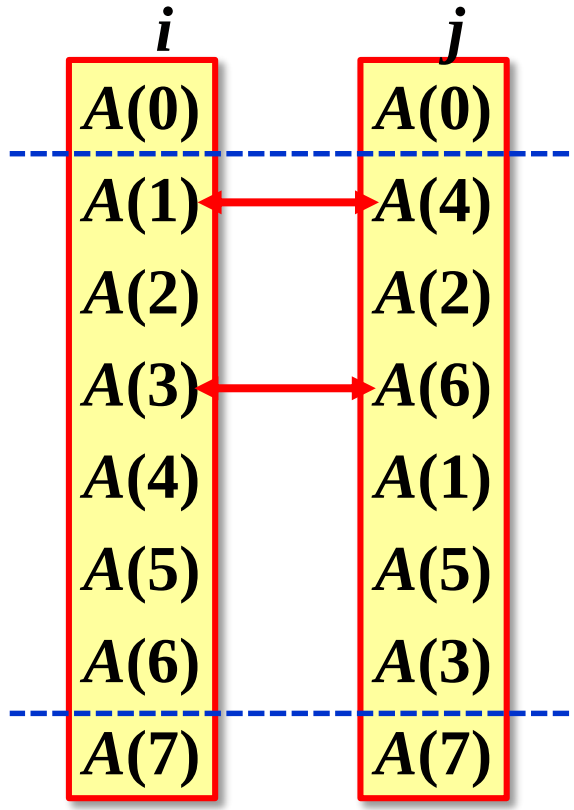
4.2 基于时间抽取的基-2-FFT快速算法

自然顺序n	二进制码表示	码位倒读	码位倒置顺序n'
0	000	000	0
1	001	100	4
2	010	010	2
3	011	110	6
4	100	001	1
5	101	101	5
6	110	011	3
7	111	111	7

- 对于数N，在其二进制最高位加1，等于加N/2。
- 若已知某个反序号为J，为求下一个反序号，可先判J的最高位：
  - 若为0，则把该位变成1（即加$N/2$）就得到下一个反序号
  - 若为1，则将最高位变成0 (相当于减去$N/2$)，再判断次高位

倒序排列算法的流程图


正序序列已在数组A[ ]中，输入N
LH= N/2 ,j=LH ， N1 = N-2
for(i=1; i<=N1; i++)
i < j
N
Y
T = A(j)
A(i) = A(j)
A(j) = T
j=4=(100)B
k=4
k=LH
j < k
N
Y
j = j-k
k = k/2
j=j-k,故 j=0
k=k/2,故k=2
j=j+k,故j=2
j = j+k


i
j
A(0)
A(1)
A(2)
A(3)
A(4)
A(5)
A(6)
A(7)
A(0)
A(4)
A(2)
A(6)
A(1)
A(5)
A(3)
A(7)

第四章　快速傅里叶变换

Fast Forurier Transform

4.2　基于时间抽取的基-2-FFT快速算法

基于时间抽取的基-2-FFT快速算法的编程方法

华东理工大学信息科学与工程学院　万永菁