RISC V Assembly Programs

Dr. Girish H
Professor
Department of ECE
Cambridge Institute of Technology
&
Kavinesh
Research Staff
CCCIR
Cambridge Institute of Technology

Discrete Fourier Transform

DFT of an N-point sequence xn, n = 0, 1, 2, . . . , N − 1 is defined as

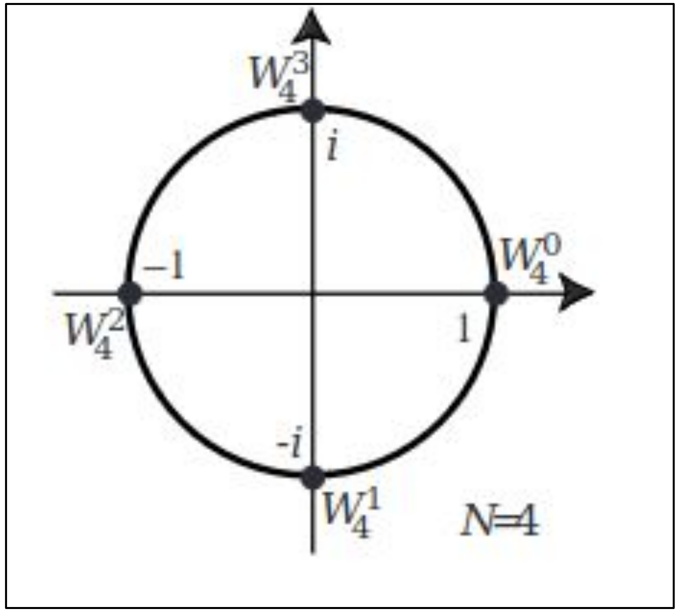
$$X_k = \sum_{n=0}^{N-1} x_n \, e^{-j\frac{2\pi k}{N}n} \qquad k = 0, 1, 2, \cdots, N-1$$

- An N-point sequence yields an N-point transform
- Xk can be expressed as an inner product:

$$X_k = \begin{bmatrix} 1 & e^{-j\frac{2\pi k}{N}} & e^{-j\frac{2\pi k}{N}2} & \ldots & e^{-j\frac{2\pi k}{N}(N-1)} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{N-1} \end{bmatrix}$$

Relationship between exponential forms and twiddle factors (W) for Periodicity = N

Sr. No.	Exponential form	Symbolic form
01	$e^{-j2\pi n/N} = e^{-j2\pi(n+N)/N}$	$W_N^{n} = W_N^{n+N}$
02	$e^{-j2\pi(n+N/2)/N} = - e^{-j2\pi n/N}$	$W_N^{n+N/2} = - W_N^{n}$
03	$e^{-j2\pi k} = e^{-j2\pi Nk/N} = 1$	$W_N^{N+K} = 1$
04	$e^{-j2(2\pi/N)} = e^{-j2\pi/(N/2)}$	$W_N^{2} = W_{N/2}$



DFT Calculation

- The forward DFT, frequency domain output in the range $0<k<N-1$ is given by:

$$X(k) := \sum_{n=0}^{n-1} \left[x(n) \left(W_N \right)^{nk} \right]$$

- While the Inverse DFT, tim ... in the range $0<k<N-1$ is denoted by,

$$x(n) := \frac{1}{N} \left[\sum_{k=0}^{n-1} \left[X(k) \left(W_N \right)^{-nk} \right] \right]$$

- Requires **N^2** complex multiplies and **N(N-1)** complex additions

Faster DFT computation

- Take advantage of the symmetry and periodicity of the complex exponential

symmetry: $$W_N^{k[N-n]} = W_N^{-kn} = (W_N^{kn})^*$$

Periodicity: $$W_N^{kn} = W_N^{k[n+N]} = W_N^{[k+N]n}$$

- Note that two length N/2 DFTs take less computation than one length N DFT:
- Algorithms that exploit computational savings are collectively called
  Fast Fourier Transforms

Fast Fourier Transform

FFT is an algorithm to convert a time domain signal to DFT efficiently.

- Consider a data sequence $x = [x(0), x(1), \ldots, x(N-1)]$ and its DFT:

$$X(k) = \sum_{n=0}^{N-1} x(n) w_N^{kn}, \quad k = 0, \ldots, N-1$$

- We can always break the summation into two summations: one on even indices (n=0,2,4,…) and one on odd indices (n=1,3,5,…), as

$$X(k) = \sum_{n\ even} x(n) w_N^{kn} + \sum_{n\ odd} x(n) w_N^{kn}, \quad k = 0, \ldots, N-1$$

Radix-2: DIT or DIF

- Radix-2 is the first FFT algorithm.
- It was proposed by Cooley and Tukey in 1965.
- Though it is not the efficient algorithm, it lays foundation for time-efficient DFT calculations.
- The next slide shows the saving in time required for calculations with radix-2.
- The algorithms appear either in
  (a) Decimation In Time (DIT), or,
  (b) Decimation In Frequency (DIF).
- DIT and DIF, both yield same complexity and results.
- They are complementary.

Let us assume that the total number of points N is even, ie N/2 is an integer. Then we can write the DFT as

$$X(k) = \sum_{n\,even} x(n) w_N^{kn} + \sum_{n\,odd} x(n) w_N^{kn}$$

$$= \sum_{m=0}^{\frac{N}{2}-1} x(2m) w_N^{k(2m)} + \sum_{m=0}^{\frac{N}{2}-1} x(2m+1) w_N^{k(2m+1)}$$

$$= \sum_{m=0}^{\frac{N}{2}-1} x(2m) \big(\underbrace{w_N^{\,2}}_{w_{N/2}}\big)^{km} + w_N^{\,k} \sum_{m=0}^{\frac{N}{2}-1} x(2m+1) \big(\underbrace{w_N^{\,2}}_{w_{N/2}}\big)^{km}$$

$$since \quad w_N^{\,2} = \left(e^{-j2\pi/N}\right)^2 = e^{-j2\pi/(N/2)} = w_{N/2}$$

The two summations are two distinct DFT's, as we can see below

$$X(k) = \sum_{m=0}^{\frac{N}{2}-1} x(2m) w_{N/2}^{km} + w_N^{k} \sum_{m=0}^{\frac{N}{2}-1} x(2m+1) w_{N/2}^{km}$$

$$\boxed{\text{N-point DFT}} = \boxed{\text{N/2-point DFT}} + w_N^{k} \boxed{\text{N/2-point DFT}}$$

$$X_N(k) = X^{e}_{N/2}(k) + w_N^{k} X^{o}_{N/2}(k)$$

for k=0,…N-1, *where*

$$X^{e}_{N/2} = DFT\left[x^{even}\right],\ x^{even} = \left[x(0), x(2), \ldots, x(N-2)\right];$$

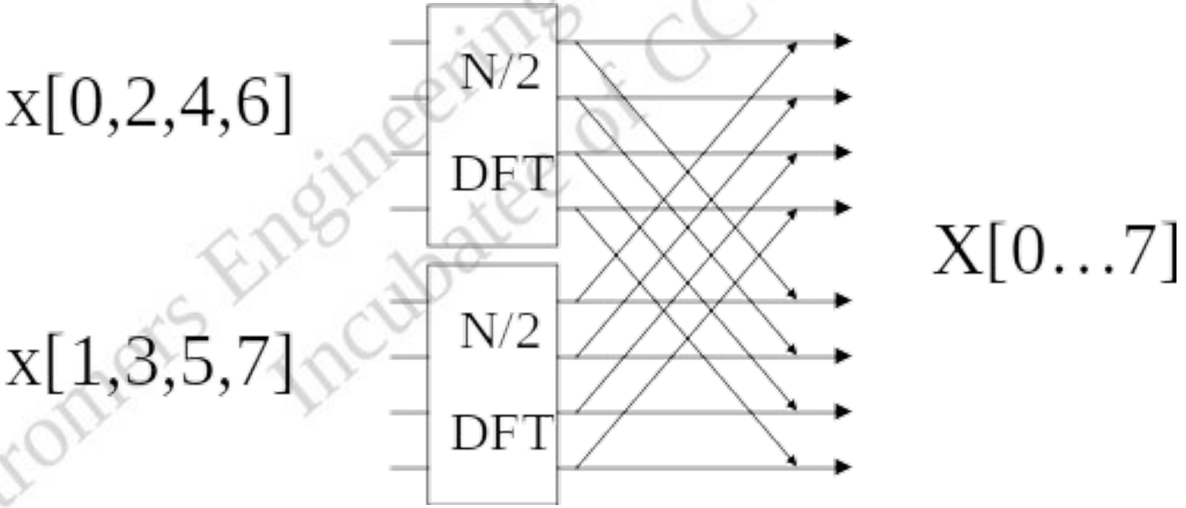
$$X^{o}_{N/2} = DFT\left[x^{odd}\right],\ x^{odd} = \left[x(1), x(3), \ldots, x(N-1)\right].$$

DIT Algorithm

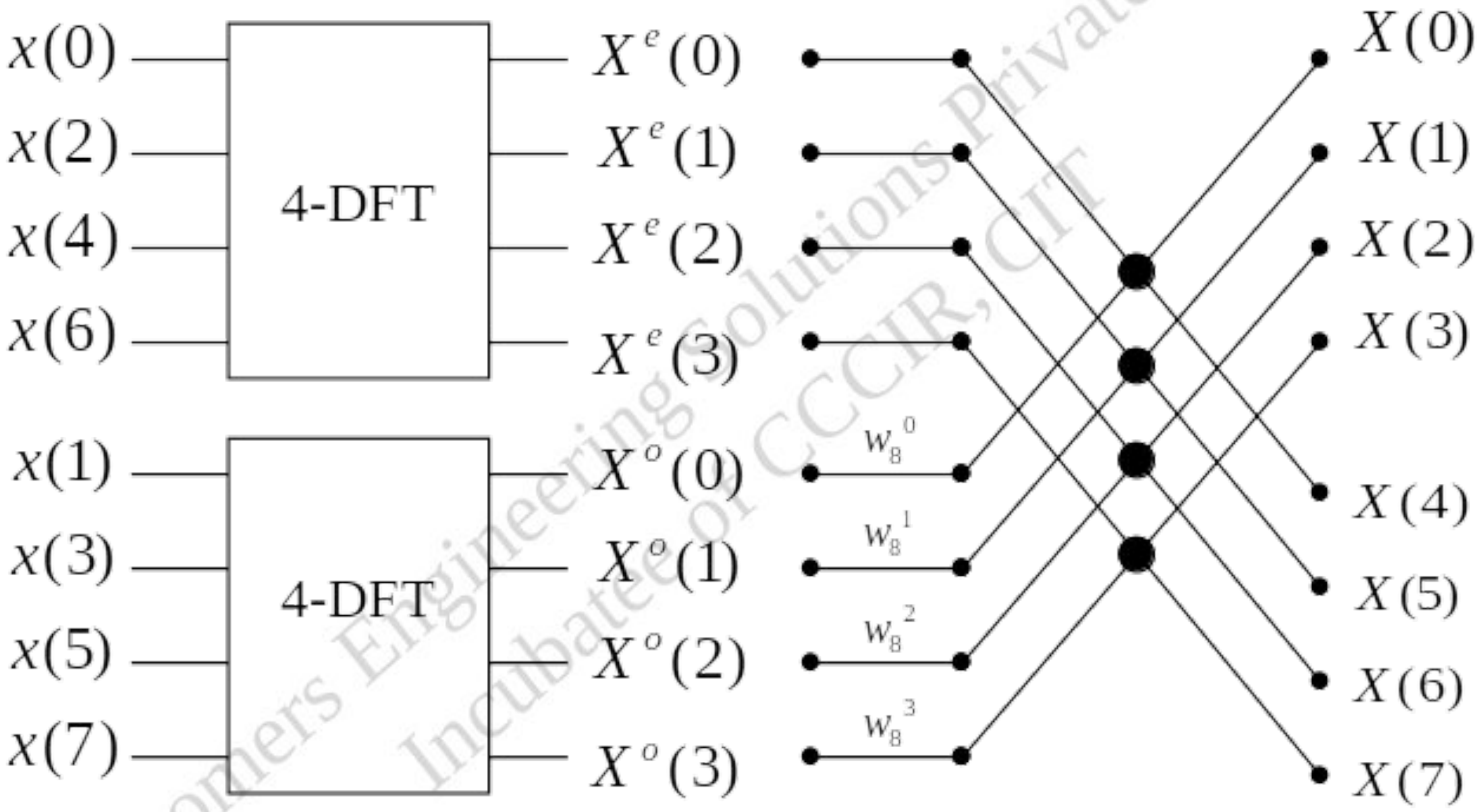
- Result is the sum of two N/2 length DFTs

$$X[k] = \underbrace{G[k]}_{\substack{\text{N/2 DFT} \\ \text{of even samples}}} + W_N^k \cdot \underbrace{H[k]}_{\substack{\text{N/2 DFT} \\ \text{of odd samples}}}$$

- Then repeat decomposition of N/2 to N/4 DFTs, etc.

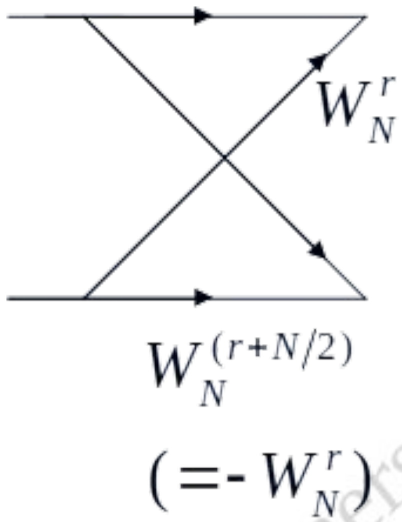


General Structure of the FFT (take, say, N=8):

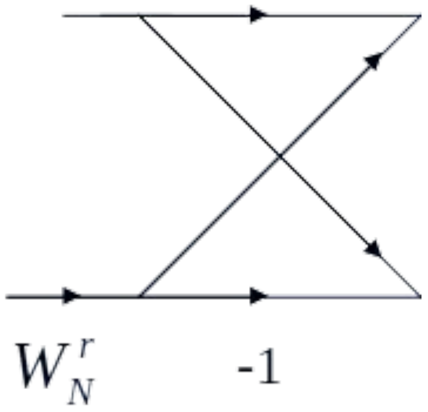


Butterfly Diagram

Cross feed of G[k] and H[k] in flow diagram is called a "butterfly", due to shape.



or simplify:



FFT C Code

```
#include <stdio.h>
#include <math.h>
#include <complex.h>

void fft(double complex *x, int N)
   {
        if (N <= 1)
 return;
// Divide sequence into even and odd parts
        double complex even[N/2], odd[N/2];
        for (int i = 0; i < N/2; i++) {
        even[i] = x[2*i];
        odd[i] = x[2*i + 1];
        }
        fft(even, N/2);
        fft(odd, N/2);
double complex t = cexp(-2.0 * M_PI * I * k / N) * odd[k];
        x[k] = even[k] + t;
        x[k + N/2] = even[k] - t;
        }
}
int main()
 {
        double complex x[] = {1, 2, 3, 4, 5, 6, 7, 8};
        int N = sizeof(x) / sizeof(x[0]);
        fft(x, N);
        printf("FFT Result:\n");
        for (int i = 0; i < N; i++) {
printf("X[%d] = %.2f + %.2fi\n", i, creal(x[i]), cimag(x[i]));
        }
        return 0;
}
```

Simulation result

Compilation command : riscv64-unknown-elf-gcc main.c -lm
Simulation command : spike pk -s a.out

```
FFT Result:
X[0] = 36.00 + 0.00i
X[1] = -4.00 + 9.66i
X[2] = -4.00 + 4.00i
X[3] = -4.00 + 1.66i
X[4] = -4.00 + 0.00i
X[5] = -4.00 + -1.66i
X[6] = -4.00 + -4.00i
X[7] = -4.00 + -9.66i
155393 cycles
```

FFT C Code

```
#include <stdio.h>
#include <math.h>
#include <complex.h>
unsigned long read_cycles(void)
{
  unsigned long cycles;
  asm volatile ("rdcycle %0" : "=r" (cycles));
  return cycles;
}
void fft(double complex *x, int N)
   {
        if (N <= 1)
 return;
// Divide sequence into even and odd parts
        double complex even[N/2], odd[N/2];
        for (int i = 0; i < N/2; i++) {
        even[i] = x[2*i];
        odd[i] = x[2*i + 1];
        }
        fft(even, N/2);
        fft(odd, N/2);
double complex t = cexp(-2.0 * M_PI * I * k / N) * odd[k];
        x[k] = even[k] + t;
        x[k + N/2] = even[k] - t;
        }
}
int main()
 {
       unsigned long start, stop;
        start = read_cycles();
        double complex x[] = {1, 2, 3, 4, 5, 6, 7, 8};
        int N = sizeof(x) / sizeof(x[0]);
        fft(x, N);
       stop = read_cycles();
        printf(" cycle :%ld\n", stop - start);
        printf("FFT Result:\n");
        for (int i = 0; i < N; i++) {
printf("X[%d] = %.2f + %.2fi\n", i, creal(x[i]), cimag(x[i]));
        }
        return 0;
}
```

Simulation result

Compilation command : riscv64-unknown-elf-gcc main.c -lm
Simulation command : spike pk a.out

cycle :9828
FFT Result:
X[0] = 36.00 + 0.00i
X[1] = -4.00 + 9.66i
X[2] = -4.00 + 4.00i
X[3] = -4.00 + 1.66i
X[4] = -4.00 + 0.00i
X[5] = -4.00 + -1.66i
X[6] = -4.00 + -4.00i
X[7] = -4.00 + -9.66i

Matrix Relations

The DFT samples defined by

$$X[k] = \sum_{n=0}^{N-1} x[n] W_N^{kn}, \quad 0 \le k \le N-1$$

can be expressed in NxN matrix as

$$\left[\mathrm{X(k)}\right] = \left[\sum_{\mathrm{n}=0}^{N-1} W_N^{nk}\right]\left[\mathrm{x(n)}\right]$$

where

$$\mathbf{X} = \left[X[0] \quad X[1] \quad \cdots\cdots \quad X[N-1]\right]^T$$

$$\mathbf{x} = \left[x[0] \quad x[1] \quad \cdots\cdots \quad x[N-1]\right]^T$$

Matrix Relation

$\sum_{k=0}^{N-1} W_N^{nk}$ can be expanded as NXN DFT matrix

$$\sum_{k=0}^{N-1} W_N^{nk} = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & W_N^{1} & W_N^{2} & \cdots & W_N^{(N-1)} \\ 1 & W_N^{2} & W_N^{4} & \cdots & W_N^{2(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & W_N^{(N-1)} & W_N^{2(N-1)} & \cdots & W_N^{(N-1)^2} \end{bmatrix}$$

DFT:

For N of length 4,range of n, k = [0 1 2 3] each.

Hence $X(n) = x(0)W_N^{n.0}+x(1)W_N^{n.1}+x(2)W_N^{n.2} + x(3)W_N^{n.3}$

		x(0)	x(1)	x(2)	x(3)
X(0)	=	W_4^{0x0}	W_4^{0x1}	W_4^{0x2}	W_4^{0x3}
X(1)	=	W_4^{1x0}	W_4^{1x1}	W_4^{1x2}	W_4^{1x3}
X(2)	=	W_4^{2x0}	W_4^{2x1}	W_4^{2x2}	W_4^{2x3}
X(3)	=	W_4^{3x0}	W_4^{3x1}	W_4^{3x2}	W_4^{3x3}

		x(0)	x(1)	x(2)	x(3)
X(0)	=	W_4^{0x0}	W_4^{0x1}	W_4^{0x2}	W_4^{0x3}
X(1)	=	W_4^{1x0}	W_4^{1x1}	W_4^{1x2}	W_4^{1x3}
X(2)	=	W_4^{2x0}	W_4^{2x1}	W_4^{2x2}	W_4^{2x3}
X(3)	=	W_4^{3x0}	W_4^{3x1}	W_4^{3x2}	W_4^{3x3}

Can be rewritten as:

		x(0)	x(1)	x(2)	x(3)
X(0)	=	W_4^0	W_4^0	W_4^0	W_4^0
X(1)	=	W_4^0	W_4^1	W_4^2	W_4^3
X(2)	=	W_4^0	W_4^2	W_4^4	W_4^6
X(3)	=	W_4^0	W_4^3	W_4^6	W_4^9

		x(0)	x(1)	x(2)	x(3)
X(0)	=	W_4^0	W_4^0	W_4^0	W_4^0
X(1)	=	W_4^0	W_4^1	W_4^2	$-W_4^1$
X(2)	=	W_4^0	W_4^2	W_4^0	W_4^2
X(3)	=	W_4^0	$-W_4^1$	W_4^2	W_4^1

		x(0)	x(1)	x(2)	x(3)
X(0)	=	1	1	1	1
X(1)	=	1	-j	-1	j
X(2)	=	1	-1	1	-1
X(3)	=	1	j	-1	-j

FFT C Code

```
#include <stdio.h>
#include <math.h>
#include <complex.h>
#define FFT_SIZE 8
#define LOG2_FFT_SIZE 3
void matvec_mul(double complex
out[FFT_SIZE], double complex
mat[FFT_SIZE][FFT_SIZE], double
complex vec[FFT_SIZE]) {
for (int i = 0; i < FFT_SIZE; i++) {
out[i] = 0;
 for (int j = 0; j < FFT_SIZE; j++) {
      out[i] += mat[i][j] * vec[j];
       }
       }
}
```

```
void generate_dft_matrix(double complex
W[FFT_SIZE][FFT_SIZE]) {
        for (int k = 0; k < FFT_SIZE; k++) {
        for (int n = 0; n < FFT_SIZE; n++) {
      W[k][n] = cexp(-2.0 * M_PI * I * k * n / FFT_SIZE);
        }} }
int main() {
double complex x[FFT_SIZE] = {1, 2, 3, 4, 5, 6, 7, 8};
        double complex X[FFT_SIZE];
        double complex W[FFT_SIZE][FFT_SIZE];
        generate_dft_matrix(W);
        matvec_mul(X, W, x);
        printf("FFT using Matrix DFT:\n");
        for (int i = 0; i < FFT_SIZE; i++) {
        printf("X[%d] = %.2f + %.2fi\n", i, creal(X[i]),
cimag(X[i]));
        }
        return 0;
```

Simulation result

Compilation command : riscv64-unknown-elf-gcc main.c -lm
Simulation command : spike pk -s a.out

```
FFT using Matrix DFT:
X[0] = 36.00 + 0.00i
X[1] = -4.00 + 9.66i
X[2] = -4.00 + 4.00i
X[3] = -4.00 + 1.66i
X[4] = -4.00 + -0.00i
X[5] = -4.00 + -1.66i
X[6] = -4.00 + -4.00i
X[7] = -4.00 + -9.66i
176752 cycles
```

FFT C Code

```
#include <stdio.h>
#include <math.h>
#include <complex.h>
unsigned long read_cycles(void)
{
  unsigned long cycles;
  asm volatile ("rdcycle %0" : "=r" (cycles));
  return cycles;
}
void fft(double complex *x, int N)
  {
        if (N <= 1)
 return;
// Divide sequence into even and odd parts
        double complex even[N/2], odd[N/2];
        for (int i = 0; i < N/2; i++) {
        even[i] = x[2*i];
        odd[i] = x[2*i + 1];
        }
        fft(even, N/2);
        fft(odd, N/2);
double complex t = cexp(-2.0 * M_PI * I * k / N) * odd[k];
        x[k] = even[k] + t;
        x[k + N/2] = even[k] - t;
        }
}
int main()
 {
       unsigned long start, stop;
        start = read_cycles();
        double complex x[] = {1, 2, 3, 4, 5, 6, 7, 8};
        int N = sizeof(x) / sizeof(x[0]);
        fft(x, N);
       stop = read_cycles();
        printf(" cycle :%ld\n", stop - start);
        printf("FFT Result:\n");
        for (int i = 0; i < N; i++) {
printf("X[%d] = %.2f + %.2fi\n", i, creal(x[i]), cimag(x[i]));
        }
        return 0;
}
```

Simulation result

Compilation command : riscv64-unknown-elf-gcc main.c -lm
Simulation command : spike pk a.out

Cycle :27070
FFT using Matrix DFT:
X[0] = 36.00 + 0.00i
X[1] = -4.00 + 9.66i
X[2] = -4.00 + 4.00i
X[3] = -4.00 + 1.66i
X[4] = -4.00 + -0.00i
X[5] = -4.00 + -1.66i
X[6] = -4.00 + -4.00i
X[7] = -4.00 + -9.66i

Factorial Assembly Instructions

```
"la x12, data\n\t"
"lw x13, 0(x12)\n\t"
"addi x14, x0, 1\n\t"
"loop:\n\t"
"mul x14, x14, x13\n\t"
"addi x13, x13, -1\n\t"
"blt x0, x13, loop\n\t"
"la x15,factorial\n\t"
"sw x14,0(x15)\n\t"
```

Factorial Risc-V inline assembly

```c
#include<stdio.h>
void factorial_asm(void);
static int data = 5;
static int factorial;
int main()
{
factorial_asm();
printf("factorial is %d", factorial);
}
```

```c
void factorial_asm()
{
asm volatile(
"la x12, data\n\t"
"lw x13, 0(x12)\n\t"
"addi x14, x0, 1\n\t"
"loop:\n\t"
"mul x14, x14, x13\n\t"
"addi x13, x13, -1\n\t"
"blt x0, x13, loop\n\t"
"la x15,factorial\n\t"
"sw x14,0(x15)\n\t"
);
}
```

Palindrome RISC-V Assembly Instructions

```
"la x10, a\n\t"
"ld x11, 0(x10)\n\t"
"addi x5, x11, 0\n\t"
"addi x6, x0, 0\n\t"
"loop:\n\t"
"addi x7, x0, 10\n\t"
"mul x6, x6, x7\n\t"
"rem x8, x5, x7\n\t"
"add x6, x6, x8\n\t"
"div x5, x5, x7\n\t"
"bne x5, x0, loop\n\t"
"la x12, b\n\t"
"sw x6, 0(x12)\n\t"
```

Palindrome Risc-V inline assembly

```
#include <stdio.h>
void swap_asm(void);
static int a=121;
static int b;
int main()
{
 swap_asm();
if (a==b)
 printf("number is a palindrome %d = %d\n", a, b);
else
 printf("not a palindrome\n");
}
```

```
void swap_asm()
{
 asm volatile(
"la x10, a\n\t"
"ld x11, 0(x10)\n\t"
"addi x5, x11, 0\n\t"
"addi x6, x0, 0\n\t"
"loop:\n\t"
"addi x7, x0, 10\n\t"
"mul x6, x6, x7\n\t"
"rem x8, x5, x7\n\t"
"add x6, x6, x8\n\t"
"div x5, x5, x7\n\t"
"bne x5, x0, loop\n\t"
"la x12, b\n\t"
"sw x6, 0(x12)\n\t"
            );
}
```

Bit Reversal Risc-V inline assembly

```
#include <stdio.h>

static unsigned int a = 0b11010010;
int main()
{
printf("Before bit reversal: a = 0x%x\n", a);
    bit_reverse_asm();
printf("After bit reversal:  a = 0x%x\n", a);
    return 0;
}
void bit_reverse_asm()
{
```

```
  asm volatile(
      "la     t0, a\n\t"          // Load addr
      "lw     t1, 0(t0)\n\t"      // Load value
      "li     t2, 0\n\t"          // t2 will hold
      "li     t3, 32\n\t"         //
      "1:\n\t"
      "slli   t2, t2, 1\n\t"      // t2 <<= 1
  "andi   t4, t1, 1\n\t"      // t4 = t1 & 1
      "or     t2, t2, t4\n\t"     // t2 |= t4
      "srli   t1, t1, 1\n\t"      // t1 >>= 1
      "addi   t3, t3, -1\n\t"     // count--
      "bnez   t3, 1b\n\t"         // if count !=
     void bit_reverse_asm()
"sw     t2, 0(t0)\n\t"      // Store rev
  );
}
```

Thank you