CS5230 ASSIGNMENT 4

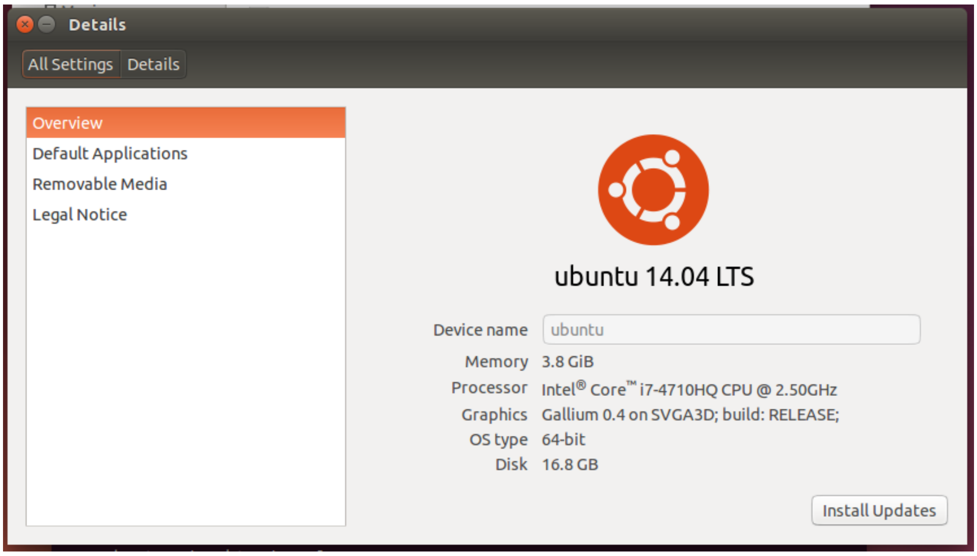
Name: Yang Mo   Matric No: A0091836X

1. Submission info:

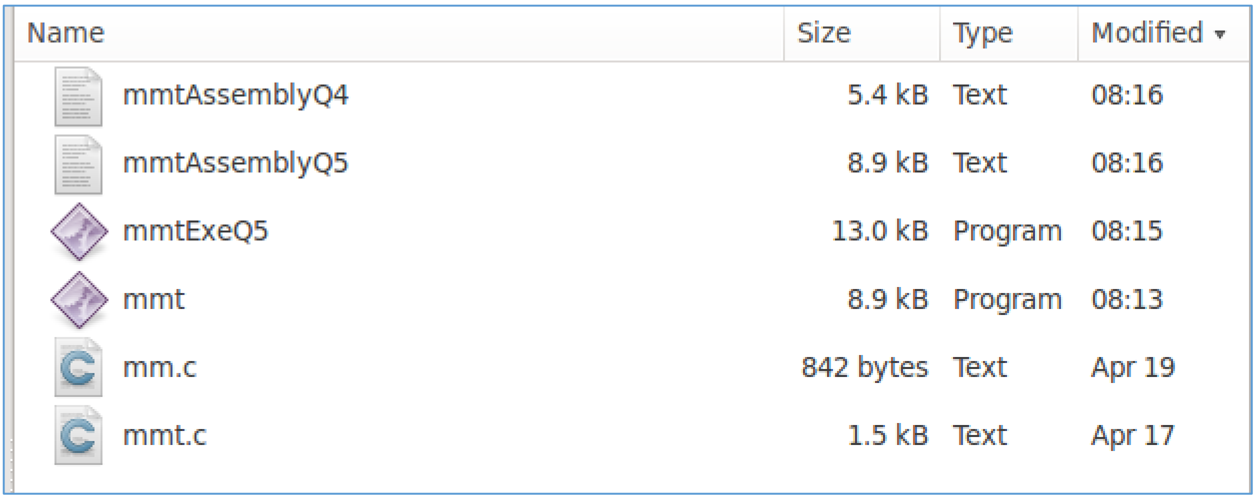
[Yang Mo][A0091836X]assignment-4.zip

2. System Specification

This program assignment is tested and run with an Ubuntu system running on and windows 10 laptop using a virtual machine software. Here is the screenshot of the system:



3. Submitted files (in src folder):



Name	Size	Type	Modified
mmtAssemblyQ4	5.4 kB	Text	08:16
mmtAssemblyQ5	8.9 kB	Text	08:16
mmtExeQ5	13.0 kB	Program	08:15
mmt	8.9 kB	Program	08:13
mm.c	842 bytes	Text	Apr 19
mmt.c	1.5 kB	Text	Apr 17

- mm.c : code for question 1
- mmt.c : code for question 2
- mmt : executable which was run for question 3
- mmtAssemblyQ4 : code for question 4
- mmtAssemblyQ5 : code for question 5
- mmtExeQ5 : executable vectorised program for question 5

4. Answers

4.1 Question 1

- corresponding code : mm.c

```c
#include <stdio.h>
#include <time.h>

#define SIZE 1000
float matrixA[SIZE][SIZE];
float matrixB[SIZE][SIZE];
float matrixC[SIZE][SIZE];

main() {
        time_t start, end;
        double diff;

        printf("Starting Initilizing Matrix..\n");
        int a, b;
        for(a = 0; a<SIZE; a++){
            for(b = 0; b<SIZE; b++){
                matrixA[a][b] = 1.0;
                matrixB[a][b] = 1.0;
            }
        }
```

Initialize all elements to be 1.0

```c
        printf("Starting Matrix Multiplication...\n");
        time(&start);

        int i, j, curIndex;
        for(i = 0; i < SIZE; i ++){
           for(j = 0; j < SIZE; j ++){
                float tempSum = 0;
                for(curIndex = 0; curIndex < SIZE; curIndex ++){
                    tempSum += matrixA[i][curIndex] * matrixB[curIndex][j];
                }
                matrixC[i][j] = tempSum;
           }
        }
```

Actual Multiplication happens here

```c
        printf("Finished Matrix Multiplication...\n");

        time(&end);
        diff = difftime(end, start);

        printf("Time Spent: %f\n", diff);

}
```

Compute the time spent in seconds

4.2 Question 2

- corresponding code : mmt.c

```c
#include <stdio.h>
#include <sys/time.h>

#define SIZE 1000

float matrixA[SIZE][SIZE];
float matrixB[SIZE][SIZE];
float matrixC[SIZE][SIZE];

float timeDiffenrenceMili(struct timeval t0, struct timeval t1)
{
    return (t1.tv_sec - t0.tv_sec) * 1000.0f + (t1.tv_usec - t0.tv_usec) / 1000.0f;
}
void doMultiplication(int tileSize){

        int i, j, m, n, curIndex;
        for(i = 0; i < SIZE; i += tileSize){
           for(j = 0; j < SIZE; j += tileSize){

               for(m = i; m < (i+tileSize) && m < SIZE; m++){
                  for(n = j; n < (i+tileSize) && n < SIZE; n++){
                      for(curIndex = 0; curIndex < SIZE; curIndex ++){
                           matrixC[m][n] += matrixA[m][curIndex] * matrixB[curIndex][n];
                      }
                  }
               }

           }
        }

}
```

```c
void MultiplyMatrix(int tileSize){

        struct timeval start, end;
        float diff;

        int a, b;
        for(a = 0; a<SIZE; a++){
            for(b = 0; b<SIZE; b++){
                matrixA[a][b] = 1.0;
                matrixB[a][b] = 1.0;
            }
        }
```

Initialize all elements to be 1.0

```c
        gettimeofday(&start, 0);

        doMultiplication(tileSize);
```

Actual tiled multiplication happens here

```c
        gettimeofday(&end, 0);
        diff = timeDiffenrenceMili(start, end);

        printf("(Tile Size, Time Spent): (%d,%f)\n", tileSize, diff);

}

main() {
        int rangeStart, rangeEnd, stepSize;
        printf("Enter Tile Size Range Start: ");
        scanf("%d", &rangeStart);

        printf("Enter Tile Size Range End: ");
        scanf("%d", &rangeEnd);

        printf("Enter Step Size: ");
        scanf("%d", &stepSize);
```

This will get the tile range and step size. After which it will multiply

```c
        int a;
        for(a = rangeStart; a <= rangeEnd; a += stepSize){
                MultiplyMatrix(a);
        }
}
```

Then it will multiply the matrices using each tile size in the range and output the time spent

4.3 Question 3

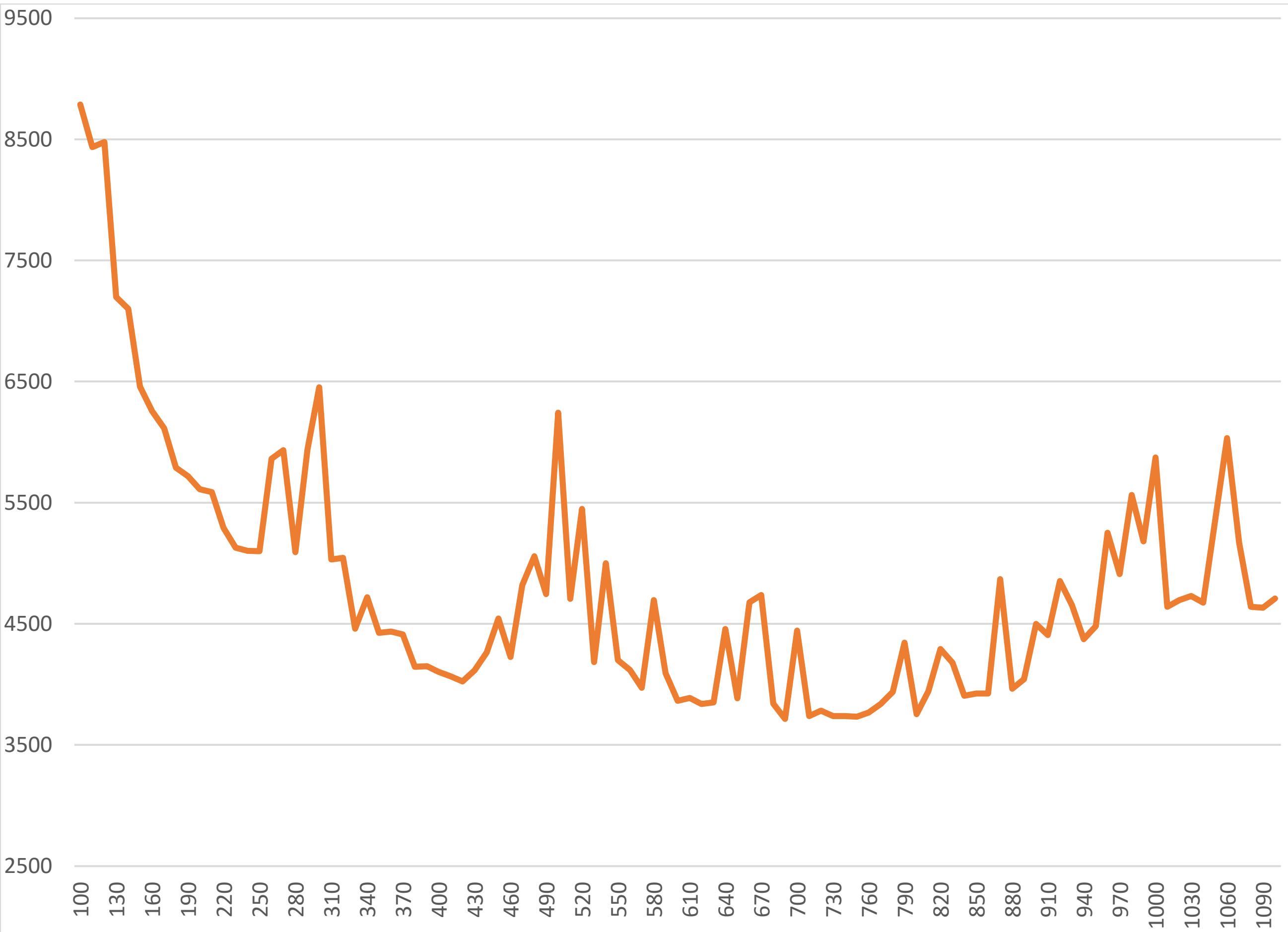
4.3.1 Execution Results

```
mo@ubuntu:~/Desktop/a4$ ./mmt
Enter Tile Size Range Start: 100
Enter Tile Size Range End: 1100
Enter Step Size: 10
(Tile Size, Time Spent): (100,9728.231445)
(Tile Size, Time Spent): (110,8226.608398)
(Tile Size, Time Spent): (120,7631.945801)
(Tile Size, Time Spent): (130,7110.616211)
(Tile Size, Time Spent): (140,6814.538086)
(Tile Size, Time Spent): (150,6976.972168)
(Tile Size, Time Spent): (160,6279.508789)
(Tile Size, Time Spent): (170,6769.264160)
(Tile Size, Time Spent): (180,6058.538086)
(Tile Size, Time Spent): (190,5629.285156)
(Tile Size, Time Spent): (200,5601.917969)
(Tile Size, Time Spent): (210,5296.043945)
(Tile Size, Time Spent): (220,5006.100098)
(Tile Size, Time Spent): (230,4896.213867)
(Tile Size, Time Spent): (240,5327.126953)
(Tile Size, Time Spent): (250,5254.357910)
(Tile Size, Time Spent): (260,4843.271973)
(Tile Size, Time Spent): (270,4583.427734)
(Tile Size, Time Spent): (280,4466.770996)
(Tile Size, Time Spent): (290,4373.857910)
(Tile Size, Time Spent): (300,4405.548828)
(Tile Size, Time Spent): (310,4412.586914)
(Tile Size, Time Spent): (320,4376.273926)
(Tile Size, Time Spent): (330,4285.148926)
(Tile Size, Time Spent): (340,4288.741211)
(Tile Size, Time Spent): (350,4239.752930)
(Tile Size, Time Spent): (360,4135.452148)
(Tile Size, Time Spent): (370,4100.017090)
(Tile Size, Time Spent): (380,4109.466797)
(Tile Size, Time Spent): (390,3977.907959)
(Tile Size, Time Spent): (400,3992.366943)
(Tile Size, Time Spent): (410,4012.839111)
(Tile Size, Time Spent): (420,3915.669922)
```

```
(Tile Size, Time Spent): (430,3949.585938)
(Tile Size, Time Spent): (440,3799.726074)
(Tile Size, Time Spent): (450,3798.812988)
(Tile Size, Time Spent): (460,3829.616943)
(Tile Size, Time Spent): (470,3940.331055)
(Tile Size, Time Spent): (480,4012.718994)
(Tile Size, Time Spent): (490,4052.767090)
(Tile Size, Time Spent): (500,4095.368896)
(Tile Size, Time Spent): (510,4046.095947)
(Tile Size, Time Spent): (520,3951.038086)
(Tile Size, Time Spent): (530,3922.936035)
(Tile Size, Time Spent): (540,3905.493896)
(Tile Size, Time Spent): (550,3869.046875)
(Tile Size, Time Spent): (560,3845.066895)
(Tile Size, Time Spent): (570,3762.794922)
(Tile Size, Time Spent): (580,3735.177979)
(Tile Size, Time Spent): (590,3742.814941)
(Tile Size, Time Spent): (600,3694.687012)
(Tile Size, Time Spent): (610,3777.616943)
(Tile Size, Time Spent): (620,3661.087891)
(Tile Size, Time Spent): (630,3550.648926)
(Tile Size, Time Spent): (640,3531.445068)
(Tile Size, Time Spent): (650,3619.204102)
(Tile Size, Time Spent): (660,3676.530029)
(Tile Size, Time Spent): (670,3606.415039)
(Tile Size, Time Spent): (680,3501.253906)
(Tile Size, Time Spent): (690,3593.597900)
(Tile Size, Time Spent): (700,3488.039062)
(Tile Size, Time Spent): (710,3589.256104)
(Tile Size, Time Spent): (720,3447.155029)
(Tile Size, Time Spent): (730,3533.920898)
(Tile Size, Time Spent): (740,3571.496094)
(Tile Size, Time Spent): (750,3625.777100)
(Tile Size, Time Spent): (760,3598.330078)
(Tile Size, Time Spent): (770,3626.521973)
(Tile Size, Time Spent): (780,3627.837891)
(Tile Size, Time Spent): (790,3552.343994)
(Tile Size, Time Spent): (800,3475.333008)
(Tile Size, Time Spent): (810,3485.242920)
(Tile Size, Time Spent): (820,3651.111084)
(Tile Size, Time Spent): (830,3629.861084)
(Tile Size, Time Spent): (840,3937.705078)
(Tile Size, Time Spent): (850,4003.823975)
(Tile Size, Time Spent): (860,3690.286133)
(Tile Size, Time Spent): (870,4358.144043)
```

```
(Tile Size, Time Spent): (880,3901.748047)
(Tile Size, Time Spent): (890,3880.378906)
(Tile Size, Time Spent): (900,4085.427979)
(Tile Size, Time Spent): (910,4127.767090)
(Tile Size, Time Spent): (920,4152.009766)
(Tile Size, Time Spent): (930,4096.457031)
(Tile Size, Time Spent): (940,4119.215820)
(Tile Size, Time Spent): (950,4146.796875)
(Tile Size, Time Spent): (960,4449.383789)
(Tile Size, Time Spent): (970,4633.674805)
(Tile Size, Time Spent): (980,4267.741211)
(Tile Size, Time Spent): (990,4285.178223)
(Tile Size, Time Spent): (1000,4409.309082)
(Tile Size, Time Spent): (1010,4457.572266)
(Tile Size, Time Spent): (1020,4407.495117)
(Tile Size, Time Spent): (1030,4555.355957)
(Tile Size, Time Spent): (1040,4656.908203)
(Tile Size, Time Spent): (1050,4493.689941)
(Tile Size, Time Spent): (1060,4733.007812)
(Tile Size, Time Spent): (1070,4372.937012)
(Tile Size, Time Spent): (1080,4332.813965)
(Tile Size, Time Spent): (1090,4396.312012)
(Tile Size, Time Spent): (1100,4324.563965)
mo@ubuntu:~/Desktop/a4$
```

4.3.2 Plot



Observations:

- Performance curve has an valley shape in the overall trend but have quite a lot fluctuations in between
- All matrices are square matrices and size is 1000 x 1000. Performance is better when the tile size is at around 420 and within the range 610 – 880.

4.4 Question 4

- Whole assembly code is **mmtAssemblyQ4** (in the code folder)

The multiplication core route is in function

```
#include <stdio.h>
#include <sys/time.h>

#define SIZE 1000

float matrixA[SIZE][SIZE];
float matrixB[SIZE][SIZE];
float matrixC[SIZE][SIZE];

float timeDiffenrenceMili(struct timeval t0, struct timeval t1)
{
    return (t1.tv_sec - t0.tv_sec) * 1000.0f + (t1.tv_usec - t0.tv_usec) / 1000.0f;
}
void doMultiplication(int tileSize){

        int i, j, m, n, curIndex;
        for(i = 0; i < SIZE; i += tileSize){
           for(j = 0; j < SIZE; j += tileSize){

               for(m = i; m < (i+tileSize) && m < SIZE; m++){
                  for(n = j; n < (i+tileSize) && n < SIZE; n++){
                      for(curIndex = 0; curIndex < SIZE; curIndex ++){
                           matrixC[m][n] += matrixA[m][curIndex] * matrixB[curIndex][n];
                      }
                  }
               }

           }
        }

}
```

So the assembly code for this function is as follows:

```
doMultiplication:
.LFB1:
        .cfi_startproc
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        movl    %edi, -36(%rbp)
        movl    $0, -20(%rbp)
        jmp     .L4
.L15:
        movl    $0, -16(%rbp)
        jmp     .L5
.L14:
        movl    -20(%rbp), %eax
        movl    %eax, -12(%rbp)
        jmp     .L6
.L13:
        movl    -16(%rbp), %eax
        movl    %eax, -8(%rbp)
        jmp     .L7
.L11:
        movl    $0, -4(%rbp)
        jmp     .L8
.L9:
        movl    -8(%rbp), %eax
        cltq
        movl    -12(%rbp), %edx
        movslq  %edx, %rdx
        imulq   $1000, %rdx, %rdx
        addq    %rdx, %rax
        movss   matrixC(,%rax,4), %xmm1
        movl    -4(%rbp), %eax
        cltq
        movl    -12(%rbp), %edx
        movslq  %edx, %rdx
        imulq   $1000, %rdx, %rdx
        addq    %rdx, %rax
        movss   matrixA(,%rax,4), %xmm2
        movl    -8(%rbp), %eax
        cltq
        movl    -4(%rbp), %edx
```

```
        movslq  %edx, %rdx
        imulq   $1000, %rdx, %rdx
        addq    %rdx, %rax
        movss   matrixB(,%rax,4), %xmm0
        mulss   %xmm2, %xmm0
        addss   %xmm1, %xmm0
        movl    -8(%rbp), %eax
        cltq
        movl    -12(%rbp), %edx
        movslq  %edx, %rdx
        imulq   $1000, %rdx, %rdx
        addq    %rdx, %rax
        movss   %xmm0, matrixC(,%rax,4)
        addl    $1, -4(%rbp)
.L8:
        cmpl    $999, -4(%rbp)
        jle     .L9
        addl    $1, -8(%rbp)
.L7:
        movl    -36(%rbp), %eax
        movl    -20(%rbp), %edx
        addl    %edx, %eax
        cmpl    -8(%rbp), %eax
        jle     .L10
        cmpl    $999, -8(%rbp)
        jle     .L11
.L10:
        addl    $1, -12(%rbp)
.L6:
        movl    -36(%rbp), %eax
        movl    -20(%rbp), %edx
        addl    %edx, %eax
        cmpl    -12(%rbp), %eax
        jle     .L12
        cmpl    $999, -12(%rbp)
        jle     .L13
.L12:
        movl    -36(%rbp), %eax
        addl    %eax, -16(%rbp)
.L5:
        cmpl    $999, -16(%rbp)
        jle     .L14
        movl    -36(%rbp), %eax
        addl    %eax, -20(%rbp)
```

```
.L4:
        cmpl    $999, -20(%rbp)
        jle     .L15
        popq    %rbp
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
.LFE1:
        .size   doMultiplication, .-doMultiplication
        .section        .rodata
        .align 8
.LC2:
        .string "(Tile Size, Time Spent): (%d,%f)\n"
        .text
        .globl  MultiplyMatrix
        .type   MultiplyMatrix, @function
```

4.5 Question 5

- Whole assembly code is **mmtAssemblyQ5** (in the code folder)
  Command to get this :
  gcc -ftree-vectorize -O3 -S -o mmtAssemblyQ5 mmt.c
- Corresponding compiled executable is **mmtExeQ5**
  Command to get this:
  gcc -ftree-vectorize -O3 -o mmtExeQ5 mmt.c

Only the core route (function doMultiplication) is shown as following:

```
doMultiplication:
.LFB25:
	.cfi_startproc
	pushq	%r15
	.cfi_def_cfa_offset 16
	.cfi_offset 15, -16
	movslq	%edi, %rax
	imulq	$4000, %rax, %r15
	salq	$2, %rax
	pushq	%r14
	.cfi_def_cfa_offset 24
	.cfi_offset 14, -24
	pushq	%r13
	.cfi_def_cfa_offset 32
	.cfi_offset 13, -32
	movl	%edi, %r13d
	pushq	%r12
	.cfi_def_cfa_offset 40
	.cfi_offset 12, -40
	xorl	%r12d, %r12d
	pushq	%rbp
	.cfi_def_cfa_offset 48
	.cfi_offset 6, -48
	pushq	%rbx
	.cfi_def_cfa_offset 56
	.cfi_offset 3, -56
	xorl	%ebx, %ebx
	movq	%rax, -8(%rsp)
.L3:
	leal	0(%r13,%rbx), %r10d
	movl	$matrixB, %r14d
	xorl	%r11d, %r11d
.L14:
	cmpl	%r10d, %ebx
	jge	.L7
	cmpl	$999, %ebx
	jg	.L7
	movq	%r12, %rcx
	movl	%ebx, %ebp
```

```
.L12:
        cmpl    %r10d, %r11d
        jge     .L9
        cmpl    $999, %r11d
        jg      .L9
        movslq  %ebp, %r9
        movq    %r14, %r8
        movl    %r11d, %esi
        imulq   $1000, %r9, %r9
        .p2align 4,,10
        .p2align 3
.L10:
        movslq  %esi, %rdi
        movq    %r8, %rdx
        leaq    (%r9,%rdi), %rax
        movss   matrixC(,%rax,4), %xmm1
        xorl    %eax, %eax
        .p2align 4,,10
        .p2align 3
.L15:
        movss   matrixA(%rcx,%rax), %xmm0
        addq    $4, %rax
        mulss   (%rdx), %xmm0
        addq    $4000, %rdx
        cmpq    $4000, %rax
        addss   %xmm0, %xmm1
        jne     .L15
        addl    $1, %esi
        addq    %r9, %rdi
        cmpl    %r10d, %esi
        movss   %xmm1, matrixC(,%rdi,4)
        je      .L9
        addq    $4, %r8
        cmpl    $1000, %esi
        jne     .L10
.L9:
        addl    $1, %ebp
        cmpl    %r10d, %ebp
        je      .L7
        addq    $4000, %rcx
        cmpl    $1000, %ebp
        jne     .L12
```

```
.L7:
        addl    %r13d, %r11d
        addq    -8(%rsp), %r14
        cmpl    $999, %r11d
        jle     .L14
        addq    %r15, %r12
        cmpl    $999, %r10d
        jg      .L2
        movl    %r10d, %ebx
        jmp     .L3
.L2:
        popq    %rbx
        .cfi_def_cfa_offset 48
        popq    %rbp
        .cfi_def_cfa_offset 40
        popq    %r12
        .cfi_def_cfa_offset 32
        popq    %r13
        .cfi_def_cfa_offset 24
        popq    %r14
        .cfi_def_cfa_offset 16
        popq    %r15
        .cfi_def_cfa_offset 8
        ret
        .cfi_endproc
.LFE25:
        .size   doMultiplication, .-doMultiplication
        .section        .rodata.str1.8,"aMS",@progbits,1
        .align 8
.LC2:
        .string "(Tile Size, Time Spent): (%d,%f)\n"
        .text
        .p2align 4,,15
        .globl  MultiplyMatrix
        .type   MultiplyMatrix, @function
```

The best way to compare is to run and see the actual time the vectorised code takes to do the same task.

In Question 3, the tiled version code (mmt) is run and following time results are observed:

```
mo@ubuntu:~/Desktop/a4$ ./mmt
Enter Tile Size Range Start: 100
Enter Tile Size Range End: 1100
Enter Step Size: 10
(Tile Size, Time Spent): (100, 9728.231445)
(Tile Size, Time Spent): (110, 8226.608398)
(Tile Size, Time Spent): (120, 7631.945801)
(Tile Size, Time Spent): (130, 7110.616211)
(Tile Size, Time Spent): (140, 6814.538086)
(Tile Size, Time Spent): (150, 6976.972168)
(Tile Size, Time Spent): (160, 6279.508789)
(Tile Size, Time Spent): (170, 6769.264160)
(Tile Size, Time Spent): (180, 6058.538086)
(Tile Size, Time Spent): (190, 5629.285156)
(Tile Size, Time Spent): (200, 5601.917969)
```

In this question, the vectorised version code which is **mmtExeQ5** (assembly : **mmtAssemblyQ5).**

The time to run the same task within the same tile range is:

```
mo@ubuntu:~/Desktop/a4$ ./mmtExeQ5
Enter Tile Size Range Start: 100
Enter Tile Size Range End: 200
Enter Step Size: 10
(Tile Size, Time Spent): (100, 3401.070068)
(Tile Size, Time Spent): (110, 3519.009033)
(Tile Size, Time Spent): (120, 3087.340088)
(Tile Size, Time Spent): (130, 2782.603027)
(Tile Size, Time Spent): (140, 2699.439941)
(Tile Size, Time Spent): (150, 2612.270996)
(Tile Size, Time Spent): (160, 2963.194092)
(Tile Size, Time Spent): (170, 2837.168945)
(Tile Size, Time Spent): (180, 2665.345947)
(Tile Size, Time Spent): (190, 2563.387939)
(Tile Size, Time Spent): (200, 2541.913086)
mo@ubuntu:~/Desktop/a4$
```

It can be seen that the vectorised code takes much less time.

4.6 Question 6

(a) Write down the assembly code for your matrix multiplication routine in the instruction set of this machine – which you are free to invent.

Corresponding code in C:

```
int i, j, curIndex;
for(i = 0; i < SIZE; i ++){
    for(j = 0; j < SIZE; j ++){
        float tempSum = 0;
        for(curIndex = 0; curIndex < SIZE; curIndex ++){
            tempSum += matrixA[i][curIndex] * matrixB[curIndex][j];
        }
        matrixC[i][j] = tempSum;
    }
}
```

This is where the actual matrix multiplication happens for question 1

Assumption:

- Integer Registers: R0, R1, R2, R3, R4, R5, R6, Ra, Rb, Rc
- Floating Registers: Rz, Rs, Rt, R7, R8
- Cpt A, X, Y → this will compute the offset value with row X0 and column Y store the value in register A
- JL LABEL, A, B → this will jump the LABEL position if A is less than B
- Ra := Rb * Rc → this will multiply Rb and Rc and store the result in register Ra

Assembly code:

```
Ra := MatrixA                // address of MatrixA
Rb := MatrixB                // address of MatrixB
Rc := MatrixC                // address of MatrixC

R1 := R0 + 0                 //initialize outer counter
R4 := R0 + 100               //initialize the matrix size for later comparison
OUTER_LOOP_START:
R2 := R0 + 0                 //initialize inner counter
INNER_LOOP_START:
Rs := Rz + 0                 //Initialize register to store sum of 0
R3 := R0 + 0                 //initialize core loop counter

CORE_LOOP_START:
Cpt R5, R1, R3               //compute offset for current index for Matrix A
Cpt R6, R2, R3               //compute offset for current index for Matrix B
R5 := R5 + Ra                //compute the address of current element in Matrix A
R6 := R6 + Rb                //compute the address of current element in Matrix B
R7 := MEM[R5]                //Load the current element in Matrix A into R7
R8 := MEM[R6]                //Load the current element in Matrix B into R8
Rt := R7 * R8                //Multiplies these two elements
Rs:= Rt + Rs                 //Add the result to sum
R3 = R3 + 1                  //increase core loop counter
JL CORE_LOOP_START, R3, R4          //if core loop counter is less than 100, Jump to core loop start for core loop

Cpt R5, R1, R2               //ompute offset for current index for Matrix C
R5 = R5 + Rc                 //compute the address of current element in Matrix C
MEM[R5] := Rs                //Store the final value for the corresponding element in Matrix C into memory
R2 := R2 + 1                 //Increase the inner loop counter
JL INNER_LOOP_START, R2, R4         //if inner loop counter is less than 100, jump to inner loop start for next inner loop
R1 := R1 + 1                 //Increase the outer loop counter
JL OUTER_LOOP_START, R1, R4         //if outer loop counter is less than 100, jump to outer loop start for next outter loop
```

(b) Perform software pipelining and show the resultant code. You may assume predicated execution. If you assume a special branch instruction in support of software pipelining, then explain clearly what the instruction does.

Assume that each instruction takes one cycle to complete

Cpt R5, R1, R3 Cpt R6, R2, R3				
R5 := R5 + Ra R6 := R6 + Rb				Start-up code
R7 := MEM[R5] R8 := MEM[R6]	Cpt R5, R1, R3 Cpt R6, R2, R3			
	R5 := R5 + Ra R6 := R6 + Rb			
Rt := R7 * R8				
	R7 := MEM[R5] R8 := MEM[R6]	Cpt R5, R1, R3 Cpt R6, R2, R3		
		R5 := R5 + Ra R6 := R6 + Rb		Pipe line instructions
Rs:= Rt + Rs R3 = R3 + 1				
	Rt := R7 * R8			
JL **CORE_LOOP_START**, R3, R4		R7 := MEM[R5] R8 := MEM[R6]	Cpt R5, R1, R3 Cpt R6, R2, R3	
			R5 := R5 + Ra R6 := R6 + Rb	
	Rs:= Rt + Rs R3 = R3 + 1			
		Rt := R7 * R8		
	JL **CORE_LOOP_START**, R3, R4		R7 := MEM[R5] R8 := MEM[R6]	Cpt R5, R1, R3 Cpt R6, R2, R3
				R5 := R5 + Ra R6 := R6 + Rb
		Rs:= Rt + Rs R3 = R3 + 1		
			Rt := R7 * R8	
		JL **CORE_LOOP_START**, R3, R4		R7 := MEM[R5] R8 := MEM[R6]
			Rs:= Rt + Rs R3 = R3 + 1	
				Rt := R7 * R8
			JL **CORE_LOOP_START**, R3, R4	
				Rs:= Rt + Rs R3 = R3 + 1
				JL **CORE_LOOP_START**, R3, R4

```
Rb := MatrixB              // address of MatrixB
Rc := MatrixC              // address of MatrixC

R1 := R0 + 0               //initialize outer counter
R4 := R0 + 100             //initialize the matrix size for later comparison
OUTER_LOOP_START:
R2 := R0 + 0               //initialize inner counter
INNER_LOOP_START:
Rs := Rz + 0               //Initialize register to store sum of 0
R3 := R0 + 0               //initialize core loop counter

R9 := R3 + 1
R10 := R9 + 1

Cpt R5, R1, R3
Cpt R6, R2, R3
R5 := R5 + Ra
R6 := R6 + Rb
R7 := MEM[R5]
R8 := MEM[R6]
Cpt R5, R1, R9
Cpt R6, R2, R9
R5 := R5 + Ra
R6 := R6 + Rb
Rt := R7 * R8
R7 := MEM[R5]
R8 := MEM[R6]
Cpt R5, R1, R10
Cpt R6, R2, R10

CORE_LOOP_START:
R5 := R5 + Ra
R6 := R6 + Rb
Rs:= Rt + Rs
R3 = R3 + 3
Rt := R7 * R8
R7 := MEM[R5]
R8 := MEM[R6]
Cpt R5, R1, R3
Cpt R6, R2, R3
JL CORE_LOOP_START, R3, R4

Cpt R5, R1, R2             //ompute offset for current index for Matrix C
R5 = R5 + Rc               //compute the address of current element in Matrix C
MEM[R5] := Rs              //Store the final value for the corresponding element in Matrix C into memory
R2 := R2 + 1               //Increase the inner loop counter
JL INNER_LOOP_START, R2, R4        //if inner loop counter is less than 100, jump to inner loop start for next inner loop
R1 := R1 + 1               //Increase the outer loop counter
JL OUTER_LOOP_START, R1, R4        //if outer loop counter is less than 100, jump to outer loop start for next outter loop
```