Computer Architecture HW #5
Address calculation and instruction count

Name: Xu Haoran
Student ID: 72344187

1. Show the matrix base address (of A, B and C) in hexadecimal.
Address of A: 0x563844694040
Address of B: 0x5638446a78c0
Address of C: 0x5638446bb140

2. Report the size of a double floating point number.
Size of a double floating point number: 8 bytes

3. Report the size of the matrix, in bytes.
Matrix size: 100 x 100
Total memory used: 80,000 Bytes
Matrix size: 1000 x 1000
Total memory used: 8,000,000 Bytes
Matrix size: 10000 x 10000
Total memory used: 800,000,000 Bytes

4. For each of A, B and C, report the address of:
array[0][0]
array[0][1]
array[0][size-1]
array[1][0]
array[size-1][0]
array[size-1][1]
array[size-1][size-1]

```
--------SIZE = 100--------
A[0][0] = 0x563844694040
A[0][1] = 0x563844694048
A[0][SIZE-1] = 0x563844694358
A[1][0] = 0x563844694360
A[SIZE-1][0] = 0x5638446a75a0
A[SIZE-1][1] = 0x5638446a75a8
A[SIZE-1][SIZE-1] = 0x5638446a78b8

B[0][0] = 0x5638446a78c0
B[0][1] = 0x5638446a78c8
B[0][SIZE-1] = 0x5638446a7bd8
B[1][0] = 0x5638446a7be0
B[SIZE-1][0] = 0x5638446bae20
B[SIZE-1][1] = 0x5638446bae28
B[SIZE-1][SIZE-1] = 0x5638446bb138

C[0][0] = 0x5638446bb140
```

```
C[0][1] = 0x5638446bb148
C[0][SIZE-1] = 0x5638446bb458
C[1][0] = 0x5638446bb460
C[SIZE-1][0] = 0x5638446ce6a0
C[SIZE-1][1] = 0x5638446ce6a8
C[SIZE-1][SIZE-1] = 0x5638446ce9b8

--------SIZE = 1000--------
A[0][0] = 0x55b5939fe040
A[0][1] = 0x55b5939fe048
A[0][SIZE-1] = 0x55b5939fff78
A[1][0] = 0x55b5939fff80
A[SIZE-1][0] = 0x55b59419d300
A[SIZE-1][1] = 0x55b59419d308
A[SIZE-1][SIZE-1] = 0x55b59419f238

B[0][0] = 0x55b59419f240
B[0][1] = 0x55b59419f248
B[0][SIZE-1] = 0x55b5941a1178
B[1][0] = 0x55b5941a1180
B[SIZE-1][0] = 0x55b59493e500
B[SIZE-1][1] = 0x55b59493e508
B[SIZE-1][SIZE-1] = 0x55b594940438

C[0][0] = 0x55b594940440
C[0][1] = 0x55b594940448
C[0][SIZE-1] = 0x55b594942378
C[1][0] = 0x55b594942380
C[SIZE-1][0] = 0x55b5950df700
C[SIZE-1][1] = 0x55b5950df708
C[SIZE-1][SIZE-1] = 0x55b5950e1638

--------SIZE = 10000--------
A[0][0] = 0x55a435d45040
A[0][1] = 0x55a435d45048
A[0][SIZE-1] = 0x55a435d588b8
A[1][0] = 0x55a435d588c0
A[SIZE-1][0] = 0x55a465821fc0
A[SIZE-1][1] = 0x55a465821fc8
A[SIZE-1][SIZE-1] = 0x55a465835838

B[0][0] = 0x55a465835840
B[0][1] = 0x55a465835848
B[0][SIZE-1] = 0x55a4658490b8
B[1][0] = 0x55a4658490c0
B[SIZE-1][0] = 0x55a4953127c0
B[SIZE-1][1] = 0x55a4953127c8
B[SIZE-1][SIZE-1] = 0x55a495326038

C[0][0] = 0x55a495326040
C[0][1] = 0x55a495326048
C[0][SIZE-1] = 0x55a4953398b8
```

C[1][0] = 0x55a4953398c0
C[SIZE-1][0] = 0x55a4c4e02fc0
C[SIZE-1][1] = 0x55a4c4e02fc8
C[SIZE-1][SIZE-1] = 0x55a4c4e16838

5. Give a formula for calculating the address of element i,j of the matrix.
&array[i][j] = base_address + ((i * SIZE) + j) * sizeof(double)

6. Is this row major or column major form of storing a matrix?
This is row-major. The elements of matrix are stored in memory row by row.

7. Compile your C program to assembly ("-S" option on compile).
gcc -S matrix-multiplication.c -o matrix-mul.s

8. Edit the assembly program, adding comments. Ignore everything except the matrix multiply portion:
a. Identify the loop controls. How many nested loops are there?
There are 3 nested loops.
loop control instructions:

```
; loop control
    ; int i = 0
    movl    $0, -12(%rbp)                   ; i = 0
    jmp     .L7                             ; jump to outer loop

.L12: ; loop of i
    ; int j = 0
    movl    $0, -8(%rbp)                    ; j = 0
    jmp     .L8                             ; jump to middle loop

.L11: ; loop of j
    ; int k = 0
    movl    $0, -4(%rbp)                    ; k = 0
    jmp     .L9                             ; jump to inner loop

.L10: ; loop of k
    ; load 3 matrices to xmm registers
    ...

; loop condition check
.L9: ;inner loop
    cmpl    $99, -4(%rbp)                   ; if (k <= 99)
    jle     .L10                            ;    repeat inner loop (L10)

    ; j++
    addl    $1, -8(%rbp)
.L8: ;middle loop
```

```
    cmpl    $99, -8(%rbp)                  ; if (j <= 99)
    jle     .L11                           ;    repeat middle loop

    ; i++
    addl    $1, -12(%rbp)
.L7: ;outer loop
    cmpl    $99, -12(%rbp)                 ; if (i <= 99)
    jle     .L12                           ;    repeat outer loop
```

b. Identify the instructions that calculate the element addresses for A, B, and C matrix elements.

```
.L10: ; loop of k
    ; load 3 matrices to xmm registers
    ; ----------- load C[i][j] to xmm1 -----------
    ; xmm register: 128-bit for floating point operations
    ; purpose: load C[i][j] to xmm1 for addition (address calculation: addr = &C +
((i * SIZE) + j) * 8)
    ; we use SIZE = 100 as the example
    movl    -8(%rbp), %eax                 ; eax = j
    movslq  %eax, %rcx                     ; rcx = (long)j
    movl    -12(%rbp), %eax                ; eax = i
    movslq  %eax, %rdx                     ; rdx = (long)i

    movq    %rdx, %rax                     ; rax = i
    salq    $2, %rax
   ; rax = i * 4 sal:Left shift destination by 2 bits (*4)
    addq    %rdx, %rax                     ; rax = i * 4 + i = i*5

   ; 0(,%rax,4) => 0 + rax * 4 -> i * 5 * 4 = i * 20
   ; optimization to avoid real multiply instructions like imul or shl
    leaq    0(,%rax,4), %rdx
   ; rdx = i * 20 leaq: Load effective address of source into destination
   addq    %rdx, %rax                     ; rax = i * 25 (20 + 5)
   salq    $2, %rax                       ; rax = i * 100 (25 * 4) shift left
    addq    %rcx, %rax                     ; rax = i * 100 + j (rcx = j)

    ; use relative addressing to get the address of C[0][0] (global variable)
    leaq    C(%rip), %rax                  ; rax = &C[0][0]
    movsd   (%rdx,%rax), %xmm1             ; xmm1 = C[i][j] (rdx,rax)

    ; ----------- load A[i][k] to xmm2 -----------
    ; same as above, load A[i][k] to xmm2 for multiplication
    movl    -4(%rbp), %eax                 ; eax = k
    movslq  %eax, %rcx                     ; rcx = (long)k
    movl    -12(%rbp), %eax                ; eax = i
    movslq  %eax, %rdx                     ; rdx = (long)i
    movq    %rdx, %rax
    salq    $2, %rax                       ; rax = i * 4
    addq    %rdx, %rax                     ; rax = i * 5
    leaq    0(,%rax,4), %rdx            ; rdx = i * 20
    addq    %rdx, %rax                  ; rax = i * 25
    salq    $2, %rax                    ; rax = i * 100
    addq    %rcx, %rax                  ; rax = i * 100 + k
    leaq    0(,%rax,8), %rdx            ; rdx = offset of A[i][k]
    leaq    A(%rip), %rax               ; rax = &A[0][0]
    movsd   (%rdx,%rax), %xmm2          ; xmm2 = A[i][k]

    ; ----------- load B[k][j] to xmm0 -----------
    ; same as above, load B[k][j] to xmm0
    movl    -8(%rbp), %eax              ; eax = j
    movslq  %eax, %rcx                  ; rcx = (long)j
    movl    -4(%rbp), %eax              ; eax = k
    movslq  %eax, %rdx                  ; rdx = (long)k
    movq    %rdx, %rax
    salq    $2, %rax                    ; rax = k * 4
    addq    %rdx, %rax                  ; rax = k * 5
    leaq    0(,%rax,4), %rdx            ; rdx = k * 20
    addq    %rdx, %rax                  ; rax = k * 25
    salq    $2, %rax                    ; rax = k * 100
    addq    %rcx, %rax                  ; rax = k * 100 + j
    leaq    0(,%rax,8), %rdx            ; rdx = offset of B[k][j]
    leaq    B(%rip), %rax               ; rax = &B[0][0]
    movsd   (%rdx,%rax), %xmm0          ; xmm0 = B[k][j]
```

c.Identify the actual floating point multiply and adds (THIS IS A GOOD PLACE TO START YOUR ANALYSIS).

```
    ; ----------- arithmetics -----------
    mulsd   %xmm2, %xmm0                ; xmm0 = A[i][k] * B[k][j]
    addsd   %xmm1, %xmm0                ; xmm0 = xmm0 + C[i][j]
```

d. Identify what variables or temporary variables (without names) are kept in CPU registers.
Registers %eax, %ecx, %edx are used to fetch the values of i, j, k
%rax, %rdx, %rcx are used for calculating the byte offset of the target element from the starting address of the matrix. They store temporary values as comments below.

```
    movl    -8(%rbp), %eax              ; eax = j
    movslq  %eax, %rcx                  ; rcx = (long)j
    movl    -12(%rbp), %eax             ; eax = i
    movslq  %eax, %rdx                  ; rdx = (long)i

    movq    %rdx, %rax                  ; rax = i
    salq    $2, %rax
   ; rax = i * 4 sal:Left shift destination by 2 bits (*4)
    addq    %rdx, %rax                  ; rax = i * 4 + i = i*5

   ; 0(,%rax,4) => 0 + rax * 4 -> i * 5 * 4 = i * 20
   ; optimization to avoid real multiply instructions like imul or shl
```

```
    leaq    0(,%rax,4), %rdx
    ; rdx = i * 20 leaq: Load effective address of source into destination
   addq    %rdx, %rax                        ; rax = i * 25 (20 + 5)
   salq    $2, %rax                          ; rax = i * 100 (25 * 4) shift left
    addq    %rcx, %rax                        ; rax = i * 100 + j (rcx = j)
```

%xmm0 is used for storing final result (A[i][k]*B[k][j] + C[i][j])
%xmm1 is used for C[i][j] in addition operations
%xmm2 is used for A[i][k] in addition operations

9. Calculate the total number of instructions in the matrix multiply.
THIS IS NOT THE NUMBER OF LINES OF INSTRUCTIONS, I MEAN THE TOTAL NUMBER EXECUTED WHEN LOOPING OVER AN NxN MATRIX.

matrix size: N*N
--------for every k loop--------
instructions (c[i][j] += A[i][k] * B[k][j]):

```
    movl    -8(%rbp), %eax
    movslq  %eax, %rcx
    movl    -12(%rbp), %eax
    movslq  %eax, %rdx
    movq    %rdx, %rax
    salq    $2, %rax
    addq    %rdx, %rax
    leaq    0(,%rax,4), %rdx
    addq    %rdx, %rax
    salq    $2, %rax
    addq    %rcx, %rax
    leaq    0(,%rax,8), %rdx
    leaq    C(%rip), %rax
    movsd   (%rdx,%rax), %xmm1

    movl    -4(%rbp), %eax
    movslq  %eax, %rcx
    movl    -12(%rbp), %eax
    movslq  %eax, %rdx
    movq    %rdx, %rax
    salq    $2, %rax
    addq    %rdx, %rax
    leaq    0(,%rax,4), %rdx
    addq    %rdx, %rax
    salq    $2, %rax
    addq    %rcx, %rax
    leaq    0(,%rax,8), %rdx
    leaq    A(%rip), %rax
    movsd   (%rdx,%rax), %xmm2

    movl    -8(%rbp), %eax
    movslq  %eax, %rcx
    movl    -4(%rbp), %eax
    movslq  %eax, %rdx
    movq    %rdx, %rax
    salq    $2, %rax
    addq    %rdx, %rax
    leaq    0(,%rax,4), %rdx
    addq    %rdx, %rax
    salq    $2, %rax
    addq    %rcx, %rax
    leaq    0(,%rax,8), %rdx
    leaq    B(%rip), %rax
    movsd   (%rdx,%rax), %xmm0

    mulsd   %xmm2, %xmm0
    addsd   %xmm1, %xmm0

    movl    -8(%rbp), %eax
    movslq  %eax, %rcx
    movl    -12(%rbp), %eax
    movslq  %eax, %rdx
    movq    %rdx, %rax
    salq    $2, %rax
    addq    %rdx, %rax
    leaq    0(,%rax,4), %rdx
    addq    %rdx, %rax
    salq    $2, %rax
    addq    %rcx, %rax
    leaq    0(,%rax,8), %rdx
    leaq    C(%rip), %rax
    movsd   %xmm0, (%rdx,%rax)

    addl    $1, -4(%rbp)
```

56 instructions for an address calculation
2 instructions for an arithmetic operation

Condition control:

```
    ...
    addl $1, -4(%rbp)  ; k++
    .L9:
    cmpl $99, -4(%rbp) ;
    jle .L10           ;
```

total instructions: 3

total instructions in an innermost loop: 56 + 2 + 3 = 61
loops: 100 * 100 * 100 (N^3)
total instructions: 61 * 100 * 100 * 100 = 61,000,000 (61 * N^3)

--------for every j loop--------

```
    addl   $1, -8(%rbp)
    cmpl   $99, -8(%rbp)
    jle    .L11
    movl   $0, -4(%rbp)
```

total extra instructions: 4
loops: 100 * 100 (N^2)
total instructions: 4 * 100 * 100 = 40,000 (4 * N^2)

--------for every i loop--------

```
    addl   $1, -12(%rbp)
    cmpl   $99, -12(%rbp)
    jle    .L12
    movl   $0, -8(%rbp)
```

total extra insttuctions: 4
loops: 100
total instructions: 4 * 100 = 400 (4 * N)

total instructions = 61,000,000 + 40,000 + 400 = 61,040,400
formula: total instructions for a N*N matrix multiplication function = 61 * N^3 + 4 * N^2 + 4 * N

Environment:
Machine Type: x86_64
Operating System: Linux 5.15.167.4-microsoft-standard-WSL2
Microsoft Windows 11 Professional 10.0.26100 Build 26100
WSL Version: 2.4.13.0
Compiler: gcc (GCC) 12.4.0
Compile command: gcc -S matrix-multiplication.c -o matrix-mul.s