AMERICAN UNIVERSITY OF BEIRUT

EECE321 Project

Testing Assembly Codes and Comparing to C++ Values

Ahmad Abbas
Tamara Fakih
Gabriel Manessa

May 2024

Contents

1 Testing Classical Gram-Schmidt

1.1 Figures for Q Matrix in Assembly

Address	+0	+1	+2	+3
0x100000A8	69	41	B2	BE
0x100000A4	3A	CD	13	3F
0x100000A0	05	FB	96	BE
0x1000009C	C1	D9	4C	3F
0x10000098	0F	B1	05	3F
0x10000094	00	00	00	00
0x10000090	08	FB	16	3E
0x1000008C	BD	D9	CC	BE
0x10000088	6A	41	32	3F
0x10000084	3A	CD	13	3F

0x100000C0	C8	B9	3C	BF
0x100000BC	D4	16	36	BD
0x100000B8	69	41	B2	BE
0x100000B4	3A	CD	13	3F
0x100000B0	07	FB	16	3F
0x100000AC	9E	9C	E3	3E
0x100000A8	69	41	B2	BE

Figure 1: Q Matrix in Assembly - Part 1 and Part 2

1.2 Figures for R Matrix in Assembly

Address	+0	+1	+2	+3
0x100000F0	06	54	9F	BE
0x100000EC	74	E0	82	3F
0x100000E8	00	00	00	00
0x100000E4	00	00	00	00
0x100000E0	31	67	16	40
0x100000DC	53	1D	A7	3F
0x100000D8	F1	19	F5	3F
0x100000D4	00	00	00	00
0x100000D0	D7	B3	5D	40
0x100000CC	66	90	8A	40
0x100000C8	3A	CD	93	3F
0x100000C4	D7	B3	DD	3F

0x10000100	28	99	CF	3F
0x100000FC	00	00	00	00
0x100000F8	00	00	00	00
0x100000F4	00	00	00	00
0x100000F0	06	54	9F	BE

Figure 2: R Matrix in Assembly - Part 1 and Part 2

1.3 C++ Code Results

```
Classical Gram-Schmidt Q Matrix:

[0.57735, 0.696311, -0.400099, 0.147442]
[0, 0.522233, 0.800198, -0.294884]
[0.57735, -0.348155, 0.444554, 0.589768]
[0.57735, -0.348155, -0.0444554, -0.73721]

Classical Gram-Schmidt R Matrix:

[1.73205, 1.1547, 4.33013, 3.4641]
[0, 1.91485, 1.30558, 2.35005]
[0, 0, 1.02247, -0.311188]
[0, 0, 0, 1.62186]
```

Figure 3: C++ Code Results

1.4 Q and R Matrices from C++ Code

$$Q = \begin{bmatrix} 0.57735 & 0.696311 & -0.400099 & 0.147442 \\ 0 & 0.522233 & 0.800198 & -0.294884 \\ 0.57735 & -0.348155 & 0.444554 & 0.589768 \\ 0.57735 & -0.348155 & -0.044554 & -0.73721 \end{bmatrix}$$

$$R = \begin{bmatrix} 1.73205 & 1.1547 & 4.33013 & 3.4641 \\ 0 & 1.91485 & 1.30558 & 2.35005 \\ 0 & 0 & 1.02247 & -0.311188 \\ 0 & 0 & 0 & 1.62186 \end{bmatrix}$$

2 Testing Modified Gram-Schmidt

2.1 Figures for Q Matrix in Assembly

Address	+0	+1	+2	+3
0x100000A8	69	41	B2	BE
0x100000A4	3A	CD	13	3F
0x100000A0	06	FB	96	BE
0x1000009C	C0	D9	4C	3F
0x10000098	0F	B1	05	3F
0x10000094	00	00	00	00
0x10000090	0B	FB	16	3E
0x1000008C	BA	D9	CC	BE
0x10000088	6A	41	32	3F
0x10000084	3A	CD	13	3F

0x100000C0	C7	B9	3C	BF
0x100000BC	BD	16	36	BD
0x100000B8	69	41	B2	BE
0x100000B4	3A	CD	13	3F
0x100000B0	06	FB	16	3F
0x100000AC	A1	9C	E3	3E

Figure 4: Q Matrix in Assembly - Part 1 and Part 2

2.2 Figures for R Matrix in Assembly

Address	+0	+1	+2	+3
0x100000F0	04	54	9F	BE
0x100000EC	74	E0	82	3F
0x100000E8	00	00	00	00
0x100000E4	00	00	00	00
0x100000E0	31	67	16	40
0x100000DC	53	1D	A7	3F
0x100000D8	F1	19	F5	3F
0x100000D4	00	00	00	00
0x100000D0	D7	B3	5D	40
0x100000CC	66	90	8A	40
0x100000C8	3A	CD	93	3F
0x100000C4	D7	B3	DD	3F

0x10000100	28	99	CF	3F
0x100000FC	00	00	00	00
0x100000F8	00	00	00	00
0x100000F4	00	00	00	00
0x100000F0	04	54	9F	BE

Figure 5: R Matrix in Assembly - Part 1 and Part 2

2.3 C++ Code Results

```
Modified Gram-Schmidt Q Matrix:

[0.57735, 0.696311, -0.400099, 0.147442]
[0, 0.522233, 0.800198, -0.294884]
[0.57735, -0.348155, 0.444554, 0.589768]
[0.57735, -0.348155, -0.0444554, -0.73721]

Modified Gram-Schmidt R Matrix:

[1.73205, 1.1547, 4.33013, 3.4641]
[0, 1.91485, 1.30558, 2.35005]
[0, 0, 1.02247, -0.311188]
[0, 0, 0, 1.62186]
```

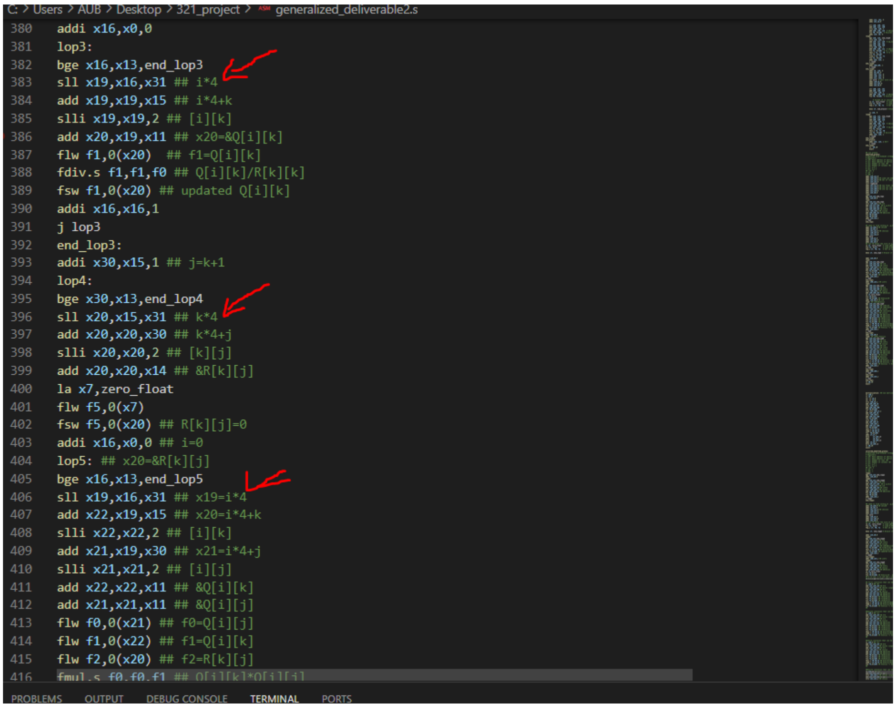
Figure 6: C++ Code Results

2.4 Q and R Matrices from C++ Code

$$Q = \begin{bmatrix} 0.57735 & 0.696311 & -0.400099 & 0.147442 \\ 0 & 0.522233 & 0.800198 & -0.294884 \\ 0.57735 & -0.348155 & 0.444554 & 0.589768 \\ 0.57735 & -0.348155 & -0.044554 & -0.73721 \end{bmatrix}$$

$$R = \begin{bmatrix} 1.73205 & 1.1547 & 4.33013 & 3.4641 \\ 0 & 1.91485 & 1.30558 & 2.35005 \\ 0 & 0 & 1.02247 & -0.311188 \\ 0 & 0 & 0 & 1.62186 \end{bmatrix}$$

3 Generalized Using Logbase2



```
C: > Users > AUB > Desktop > 321_project > generalized_deliverable2.s
addi x16,x0,0
lop3:
bge x16,x13,end_lop3
sll x19,x16,x31 ## i*4
add x19,x19,x15 ## i*4+k
slli x19,x19,2 ## [i][k]
add x20,x19,x11 ## x20=&Q[i][k]
flw f1,0(x20)  ## f1=Q[i][k]
fdiv.s f1,f1,f0 ## Q[i][k]/R[k][k]
fsw f1,0(x20) ## updated Q[i][k]
addi x16,x16,1
j lop3
end_lop3:
addi x30,x15,1 ## j=k+1
lop4:
bge x30,x13,end_lop4
sll x20,x15,x31 ## k*4
add x20,x20,x30 ## k*4+j
slli x20,x20,2 ## [k][j]
add x20,x20,x14 ## &R[k][j]
la x7,zero_float
flw f5,0(x7)
fsw f5,0(x20) ## R[k][j]=0
addi x16,x0,0 ## i=0
lop5: ## x20=&R[k][j]
bge x16,x13,end_lop5
sll x19,x16,x31 ## x19=i*4
add x22,x19,x15 ## x20=i*4+k
slli x22,x22,2 ## [i][k]
add x21,x19,x30 ## x21=i*4+j
slli x21,x21,2 ## [i][j]
add x22,x22,x11 ## &Q[i][k]
add x21,x21,x11 ## &Q[i][j]
flw f0,0(x21) ## f0=Q[i][j]
flw f1,0(x22) ## f1=Q[i][k]
flw f2,0(x20) ## f2=R[k][j]
fmul.s f0,f0,f1 ## Q[i][k]*Q[i][j]
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS
```

Figure 7: Generalized Logbase2 - Figure 1

```
logbase2: ## final value in x11, input in x14
addi sp,sp,-12
sw x10,0(sp)
sw x6,4(sp)
sw x1,8(sp)
li x10,1 ## inital , maybe its 2 so logbase2(2)=1
li x6,2 ## start with 2

d1:
beq x6,x14,doned1
slli x6,x6,1
addi x10,x10,1
j d1
doned1:
addi x11,x10,0
lw x10,0(sp)
lw x6,4(sp)
lw x1,8(sp)
jalr x0,0(x1)
```

Figure 8: Generalized Logbase2 - Figure 2

4 Unrolling

4.1 Unrolling One Loop

```
lop5: ## x20=&R[k][j]
bge x16,x13,end_lop5
slli x19,x16,2 ## x19=i*4
add x22,x19,x15 ## x20=i*4+k
slli x22,x22,2 ## [i][k]
add x21,x19,x30 ## x21=i*4+j
slli x21,x21,2 ## [i][j]
add x22,x22,x11 ## &Q[i][k]
add x21,x21,x11 ## &Q[i][j]
flw f0,0(x21) ## f0=Q[i][j]
flw f1,0(x22) ## f1=Q[i][k]
flw f2,0(x20) ## f2=R[k][j]
fmul.s f0,f0,f1 ## Q[i][k]*Q[i][j]
fadd.s f2,f0,f2 ## R[k][j]+Q[i][k]*Q[i][j]
fsw f2,0(x20) ## R[k][j] += Q[i][k] * Q[i][j]
addi x16,x16,1
j lop5
end_lop5:
```

Figure 9: Before Unrolling Loop

```
## First iteration that can be done in independent core
li x19,0 ## [0]
add x22,x19,x15 ## 0+k
slli x22,x22,2 ## [0][k] single precision
add x21,x19,x30 ## 0+j
slli x21,x21,2 ## [0][j]
add x22,x22,x11 ## &Q[0][k]
add x21,x21,x11 ## &Q[0][j]
flw f0,0(x22) ## f0=Q[0][k]
flw f1,0(x21) ## f1=Q[0][j]
flw f2,0(x20) ## f2=R[k][j]
fmul.s f0,f0,f1 ## Q[0][k]*Q[0][j]
fadd.s f2,f0,f2 ## R[k][j]+Q[0][k]*Q[0][j]
fsw f2,0(x20) ## R[k][j] += Q[0][k] * Q[0][j]
##

##Second iteration that can be done in independent core
li x19,4 ## [1]
add x22,x19,x15 ## 4+k
slli x22,x22,2 ## [1][k] single precision
add x21,x19,x30 ## 1+j
slli x21,x21,2 ## [1][j]
add x22,x22,x11 ## &Q[1][k]
add x21,x21,x11 ## &Q[1][j]
flw f0,0(x22) ## f0=Q[1][k]
flw f1,0(x21) ## f1=Q[1][j]
flw f2,0(x20) ## f2=R[k][j]
fmul.s f0,f0,f1 ## Q[1][k]*Q[1][j]
fadd.s f2,f0,f2 ## R[k][j]+Q[1][k]*Q[1][j]
fsw f2,0(x20) ## R[k][j] += Q[1][k] * Q[1][j]
##
```

Figure 10: After Unrolling Loop

4.2 Unrolling Multiple Loops

```
lopp6: ## x20=& R[k][j]
bge x16,x13,end_lopp6
slli x22,x16,2 ## i*4
add x22,x22,x30 ## i*4+j
slli x22,x22,2 ## [i][j]
slli x21,x16,2 ## i*4
add x21,x21,x15 ## i*4+k
slli x21,x21,2 ## [i][k]
add x22,x22,x11 ## &Q[i][j]
add x21,x21,x11 ## &Q[i][k]
flw f0,0(x21) ## Q[i][k]
flw f1,0(x22)## Q[i][j]
flw f2,0(x20) ## R[k][j]
fmul.s f2,f0,f2 ## R[k][j] * Q[i][k]
fsub.s f1,f1,f2 ## Q[i][j] - R[k][j] * Q[i][k]
fsw f1,0(x22) ## Q[i][j] -= R[k][j] * Q[i][k]
addi x16,x16,1
j lopp6
```

Figure 11: Before Unrolling second Loop - Figure 3

```
## First iteration that could be done on independent cores
##, once R[k][j] is finalized above
li x22,0 ## i=0
add x22,x22,x30 ## i*0+j
slli x22,x22,2 ## [0][j]
li x21,0
add x21,x21,x15 ## i*0+k
slli x21,x21,2 ## [0][k]
add x22,x22,x11 ## &Q[0][j]
add x21,x21,x11 ## & Q[0][k]
flw f0,0(x21) ## Q[0][k]
flw f1,0(x22)## Q[0][j]
flw f2,0(x20) ## R[k][j]
fmul.s f2,f0,f2 ## R[k][j] * Q[0][k]
fsub.s f1,f1,f2 ## Q[0][j] - R[k][j] * Q[0][k]
fsw f1,0(x22) ## Q[0][j] -= R[k][j] * Q[0][k]
##

## Second iteration that could be done on independent cores
##, once R[k][j] is finalized above
li x22,4 ## i=1
add x22,x22,x30 ## 4+j
slli x22,x22,2 ## [1][j]
li x21,4
add x21,x21,x15 ## 4+k
slli x21,x21,2 ## [1][k]
add x22,x22,x11 ## &Q[1][j]
add x21,x21,x11 ## & Q[1][k]
flw f0,0(x21) ## Q[1][k]
flw f1,0(x22)## Q[1][j]
flw f2,0(x20) ## R[k][j]
fmul.s f2,f0,f2 ## R[k][j] * Q[1][k]
fsub.s f1,f1,f2 ## Q[1][j] - R[k][j] * Q[1][k]
fsw f1,0(x22) ## Q[1][j] -= R[k][j] * Q[1][k]
##
```

Figure 12: After Unrolling second Loop - Figure 4

5 QR Extension

```
solveforx: ## given y is in x8, H is in x24
## QR DECOMPOSE H
## where modified takes A in x10,
## and result it in x11
## and x14
addi x10,x24,0 ## A---> H
call modified_granny ## now x11--> Q, x14-> R
addi x10,x23,0 ## result in Q_t
addi x26,x11,0 ## input Q in x26
call transposeMatrix ## our result in x10
## auto auto tghyrt ma3a x23 , same address :)
## our y is in x8, our Q_t is in x10
addi x11,x25,0 ## now x11 holds new addres for y_prime
addi x24,x8,0 ## now x24--> holds y
addi x27,x10,0
call multiplyMatrixVector
## y_prime is in x11, R--> x14
addi x18,x11,0 ## y_prime --> x18
addi x10,x28,0
call backwardSubstitution  ## our result in x10
li a0,10
ecall
```

Figure 13: QR Extension - Assembly Code for Function

Address	+0	+1	+2	+3
0x10000180	BD	16	36	BD
0x1000017C	A1	9C	E3	3E
0x10000178	C0	D9	4C	3F
0x10000174	BA	D9	CC	BE
0x10000170	69	41	B2	BE
0x1000016C	69	41	B2	BE
0x10000168	0F	B1	05	3F
0x10000164	6A	41	32	3F
0x10000160	3A	CD	13	3F
0x1000015C	3A	CD	13	3F
0x10000158	00	00	00	00
0x10000154	3A	CD	13	3F

0x10000190	C7	B9	3C	BF
0x1000018C	06	FB	16	3F
0x10000188	06	FB	96	BE
0x10000184	0B	FB	16	3E
0x10000180	BD	16	36	BD

Figure 14: QR Extension - Transposing Q Matrix

0x10000140	A2	1B	84	BF
0x1000013C	8E	3E	33	40
0x10000138	C6	D1	5E	3F
0x10000134	30	3A	CB	40

Figure 15: QR Extension - Result of MultiplyMatrix Vector (y')

Address	+0	+1	+2	+3
0x10000150	B6	E8	22	BF
0x1000014C	C0	E8	22	40
0x10000148	13	00	00	BF
0x10000144	00	A3	8B	BF

Figure 16: QR Extension - Result of Backward Substitution (x)

```
Transposed Q Matrix:
[0.57735, 0, 0.57735, 0.57735]
[0.696311, 0.522233, -0.348155, -0.348155]
[-0.400099, 0.800198, 0.444554, -0.0444554]
[0.147442, -0.294884, 0.589768, -0.73721]

Result of Q_T * y (y_prime):
[6.35085, 0.870388, 2.80069, -1.03209]

Result of backward substitution (x):
[-1.09091, -0.5, 2.54545, -0.636364]
```

Figure 17: QR Extension - C++ Code Results

5.1 Transposed Q Matrix from C++ Code

$$Q^T = \begin{bmatrix} 0.57735 & 0 & -0.400099 & 0.147442 \\ 0.57735 & 0.522233 & 0.800198 & -0.294884 \\ 0.57735 & -0.348155 & 0.444554 & 0.589768 \\ 0.57735 & -0.348155 & -0.044554 & -0.73721 \end{bmatrix}$$

5.2 y' Vector from C++ Code

$$y' = \begin{bmatrix} 6.35085 \\ 0.870388 \\ 2.80069 \\ -1.03209 \end{bmatrix}$$

5.3 x Vector from C++ Code

$$x = \begin{bmatrix} -1.09091 \\ -0.5 \\ 2.54545 \\ -0.636364 \end{bmatrix}$$

6 Caching

```
loop2:
    bge x16, x13, end_loop2
    slli x19, x16, 2
    add x19, x17, x19  # &q[i]
    slli x20, x16, 2 #
    add x20, x20, x15
    slli x20, x20, 2
    add x20, x20, x10  # &A[i][k]
    flw f1, 0(x20)
    fsw f1, 0(x19)  # q[i] = A[i][k]
    addi x16, x16, 1
    j loop2
end_loop2:
```

Figure 18: Code Before Caching

```
lopcache:

    bge x16, x13, end_lopcache
    slli x19, x16, 2
    add x19, x4, x19  # &a_cache[i]
    slli x20, x16, 2 #
    add x20, x20, x15
    slli x20, x20, 2
    add x20, x20, x11  # &Q[i][k]
    flw f1, 0(x20)
    fsw f1, 0(x19)  # a_cache[i] = A[i][k]
    addi x16, x16, 1
    j lopcache
end_lopcache:
li x16, 0
lop2:
    bge x16, x13, end_lop2
    slli x19, x16, 2
    add x19, x17, x19  # &q[i]
    slli x20, x16, 2 #
    add x20, x20, x4
    flw f1, 0(x20)
    fsw f1, 0(x19)  # q[i] = a_cache[i]
    addi x16, x16, 1
    j lop2
end_lop2:
```

Figure 19: Code After Caching

7 Verifications

7.1 First Identity: $Q \times R = A$

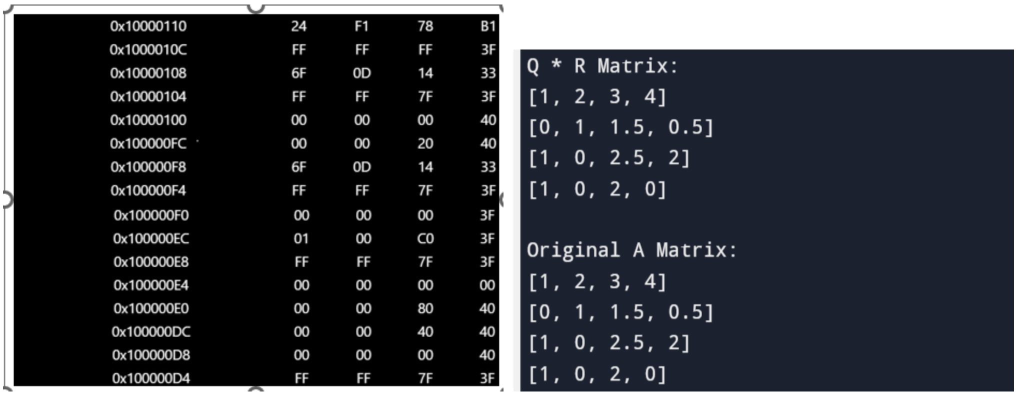


Figure 20: First Identity: $Q \times R = A$ - Assembly (left) and C++ (right)

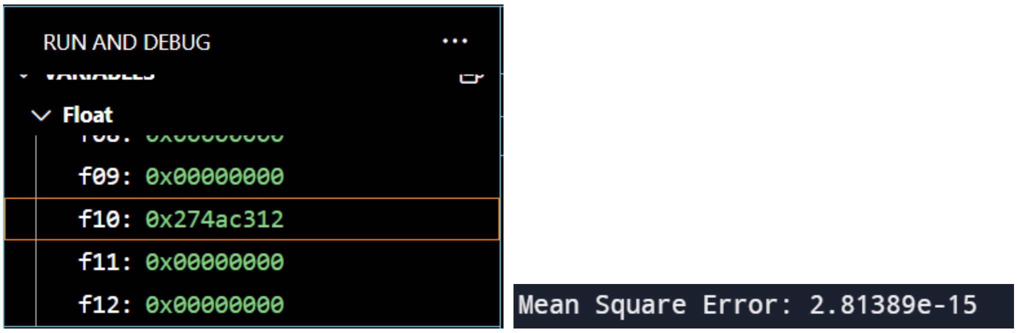


Figure 21: Mean Squared Error for $Q \times R = A$ - Assembly (left) and C++ (right)

7.2 Second Identity: $Q^T \times Q = I$

0x10000150	01	00	80	3F
0x1000014C	87	90	AB	32
0x10000148	43	48	02	33
0x10000144	3A	CD	13	33
0x10000140	87	90	AB	32
0x1000013C	00	00	80	3F
0x10000138	82	E3	41	33
0x10000134	7E	2D	F0	33
0x10000130	43	48	02	33
0x1000012C	82	E3	41	33
0x10000128	00	00	80	3F
0x10000124	3A	CD	13	33
0x10000120	3A	CD	13	33
0x1000011C	7E	2D	F0	33
0x10000118	3A	CD	13	33
0x10000114	FF	FF	7F	3F

```
Q_T * Q Matrix:
[1, -2.22045e-16, -4.96131e-16, -2.77556e-16]
[-2.22045e-16, 1, -1.78677e-16, -5.55112e-16]
[-4.96131e-16, -1.78677e-16, 1, 6.93889e-18]
[-2.77556e-16, -5.55112e-16, 6.93889e-18, 1]

Q_T * A Matrix:
[1.73205, 1.1547, 4.33013, 3.4641]
[-3.33067e-16, 1.91485, 1.30558, 2.35005]
[-8.88178e-16, -9.99201e-16, 1.02247, -0.311188]
[-5.55112e-16, -1.44329e-15, -1.9984e-15, 1.62186]
```

Figure 22: Second Identity: $Q^T \times Q = I$ - Assembly (left) and C++ (right)

```
RUN AND DEBUG ...
Float
  f09: 0x00000000
  f10: 0x27743c9c
  f11: 0x00000000
  f12: 0x00000000
```

```
Mean Square Error: 2.50129e-15
```

Figure 23: Mean Squared Error for $Q^T \times Q = I$ - Assembly (left) and C++ (right)

7.3 Third Identity: $Q^T \times A = R$

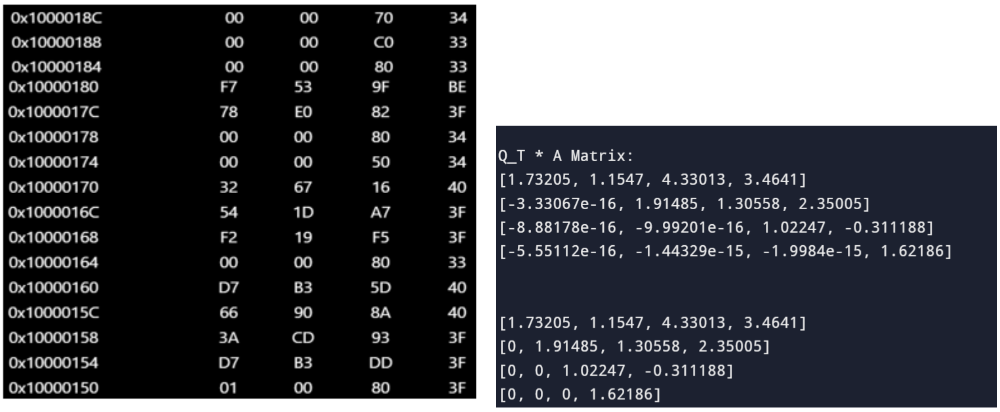


Figure 24: Third Identity: $Q^T \times A = R$ - Assembly (left) and C++ (right)

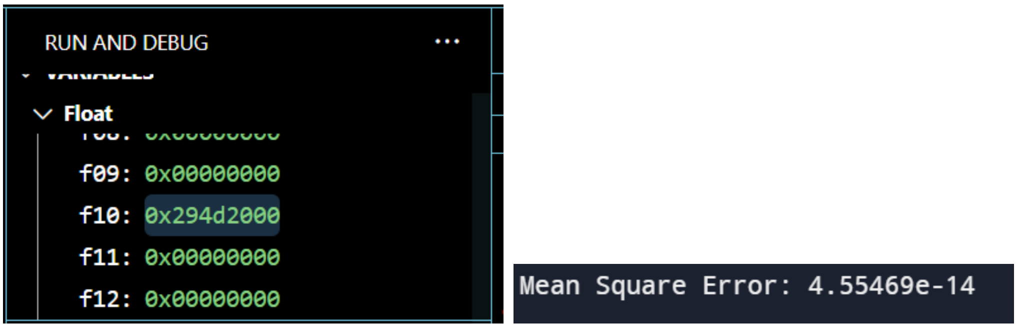


Figure 25: Mean Squared Error for $Q^T \times A = R$ - Assembly (left) and C++ (right)