Computer Organizations and Systems
Assignment -2

Submitted By -
Sayam Kumar
S20180010158 Sec-A

Question-1. Solving Question-3 from Mid Sem 1

Code -

```
#include<stdio.h>
typedef struct oldSensorData
{
	short code;
	long start;
	char raw[3];
	double data;
}oldSensorData;

typedef struct newSensorData
{
	short code;
	short start;
	char raw[5];
	short sense;
	short ext;
	double data;
}newSensorData;
```

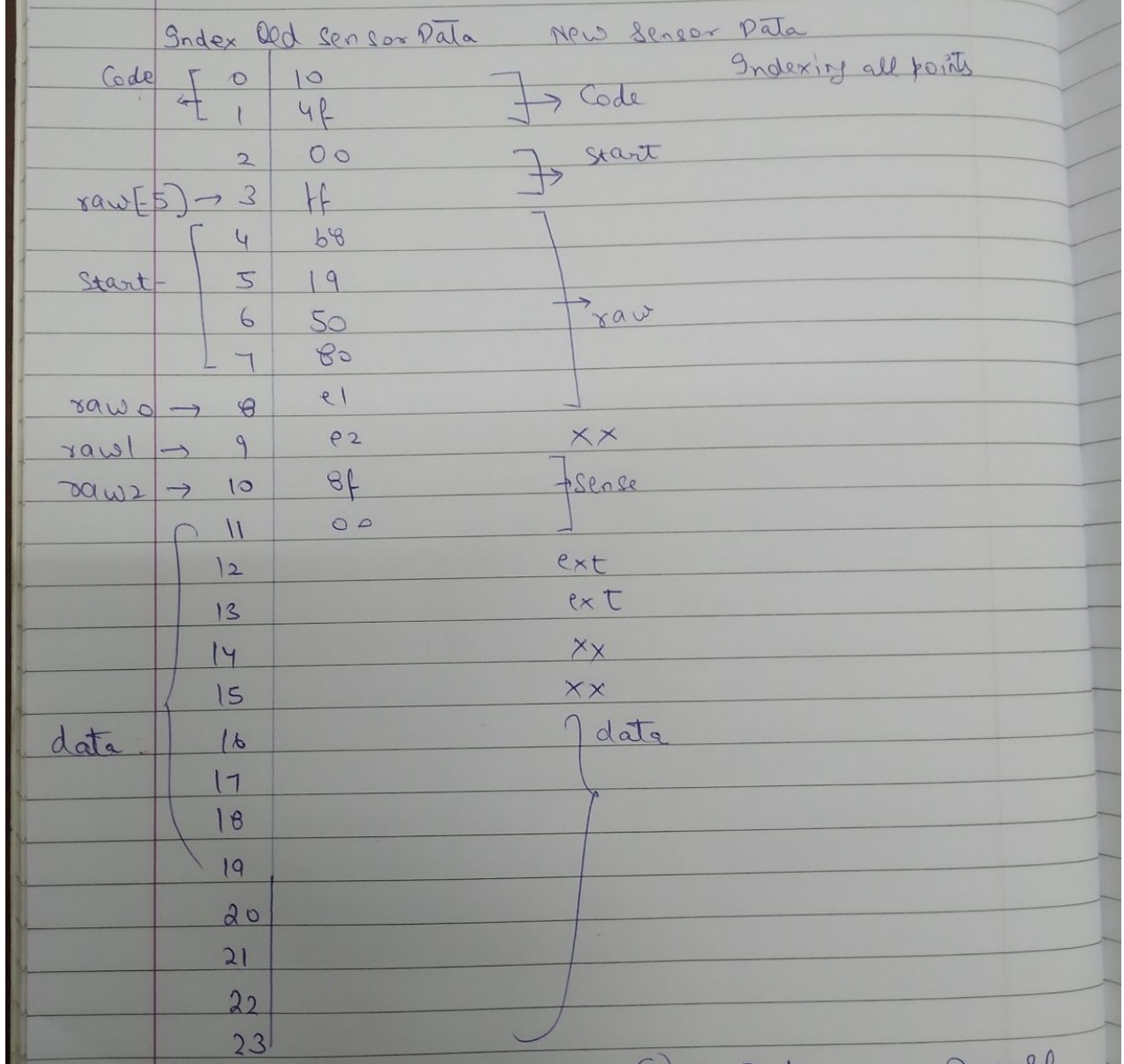
```
int main()
{
	oldSensorData *o;
	o->code = 0x104f;
	o->start = 0x80501ab8;
	o->raw[0] = 0xe1;
	o->raw[1] = 0xe2;
	o->raw[2] = 0x8f;
	o->raw[-5] = 0xff;
	o->data = 1.5;

	n->code = 0x104f;
	n->start = 0x0000;
	n->raw[0] = 0x00;
	n->raw[1] = 0x00;
	n->raw[2] = 0x00;
	n->raw[3] = 0x00;
	n->raw[4] = 0xb8;
	n->sense = 0xff50;
	n->ext = 0x0000;
	n->data = 0x00000000008fe2e1;
}
```

Answers -

Page No.:
Date:
YOUVA

Mid Sem 1



(a) newData → start = 0xff00
(b) newData → raw[0] = 0x68
(c) newData → raw[1] = 0x50
(d) newData → raw[4] = 0xe1
(e) newData → Sense = 0x008f

Question-2. Solve any 6 problems from 6.9 to 6.20.

Page No.:
Date:

Q- Solve any six questions from 6.9 to 6.20

Q-1 6.9

Number of Cache Sets = S
Tag Bits = t
Set Index Bits = s
Block offset bits = b

Also
$S = 2^s$, $B = 2^b$
$t = m - (s+b)$
$C = B \times E \times S$

Part 1

(1) $m = 32$, $C = 1024$, $B = 4$, $E = 1$

Ans $S = 256$, $s = \log_2(256) = 8$
$b = \log_2(4) = 2$
$t = 32 - 2 - 8 = 22$

(2) $m = 32$, $C = 1024$, $B = 8$, $E = 4$

Ans $S = 1024/32 = 32$, $s = \log_2(32) = 5$
$b = \log_2(8) = 3$
$t = 32 - 3 - 5 = 24$

(3) $m = 32$, $C = 1024$, $B = 32$, $E = 32$

Ans $S = 1024/1024 = 1$, $s = \log_2(1) = 0$
$b = \log_2(32) = 5$
$t = 32 - 5 = 27$

Now, completing the table

Cache	m	C	B	E	S	t	s	b
1	32	1024	8	1	256	22	8	2
2	32	1024	4	4	32	24	5	3
3	32	1024	32	32	1	27	0	5

Page No.:
Date:

Q-2 6.12

① Memory is byte addressable
② Memory accesses are to 1-byte words
③ Addresses are 13 bits wide
④ E = 2, B = 4, S = 8

CO = The cache block offset
CI = The cache set index
CT = The cache tag

Ans B = 4, $b = \log_2(4) = 2$

S = 8, $s = \log_2(8) = 3$ $\quad t = 13 - 2 - 3 = 8$

CT	CT	CT	CT	CT	CT	CT	CT	CI	CI	CI	CO	CO
12	11	10	9	8	7	6	5	4	3	2	1	0

Q-3 6.13

Address = 0x0053

Representation in hex = 0x0000 1101 010 | 0 0 1 1

tag | set | block offset

A) Address format

0	1	1	0	1	0	1	0	1	0	0	1	1
12	11	10	9	8	7	6	5	4	3	2	1	0

set: bits 4–2 | Block: bits 1–0 | Tag: bits 12–5

B) Memory Reference:

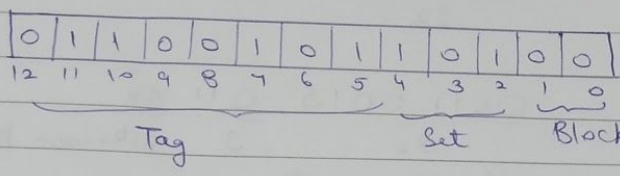
Parameter	Value
Cache Block Offset (CO)	0x3
Cache set Index (CI)	0x4
Cache Tag (CT)	0x6A
Cache hit? (Y/N)	N
Cache Byte Returned	____

Q-4 6.14

Address = 0x 0CB4

Representation in hex = 0x 0000 1100 1011 0100
tag set ↳ block

A) Address format



B) Memory Reference -

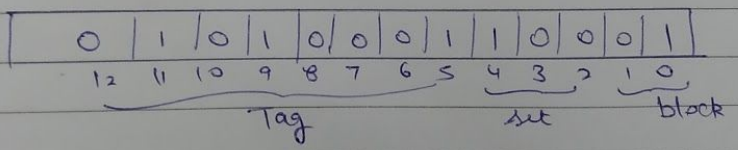
Parameter	Value
Cache Block Offset	0x0
Cache Set Index (CI)	0x5
Cache Tag (CT)	0x65
Cache Hit ? (Y/N)	N
Byte Returned	—

Q-5 6.15

Address = 0x 0A31 Repr = 0x 0000 1010 0011 0001
tag set ↳ block

A)

Address format



B) Memory Reference	Value
Cache Block Offset (CO)	0x1
Cache Set Index (CI)	0x4
Cache Tag (CT)	0x51
Cache Hit ? (Y/N)	N
Cache Byte Returned	—

S20180010158

Page No.:
Date:

Q-6 6.16 Hitting Set 3

Set 3 contains only one valid tag 0x32.

Representation of 32.

set 3

00110010 011xx Total 13 bits

tag 32 block

Clubbing 4 byte

0 | 0110 | 0100 | 11xx

6 4 It can be 1100 C

1101 D

1110 E

1111 F

Ans = Valid Representations to hit Set 3

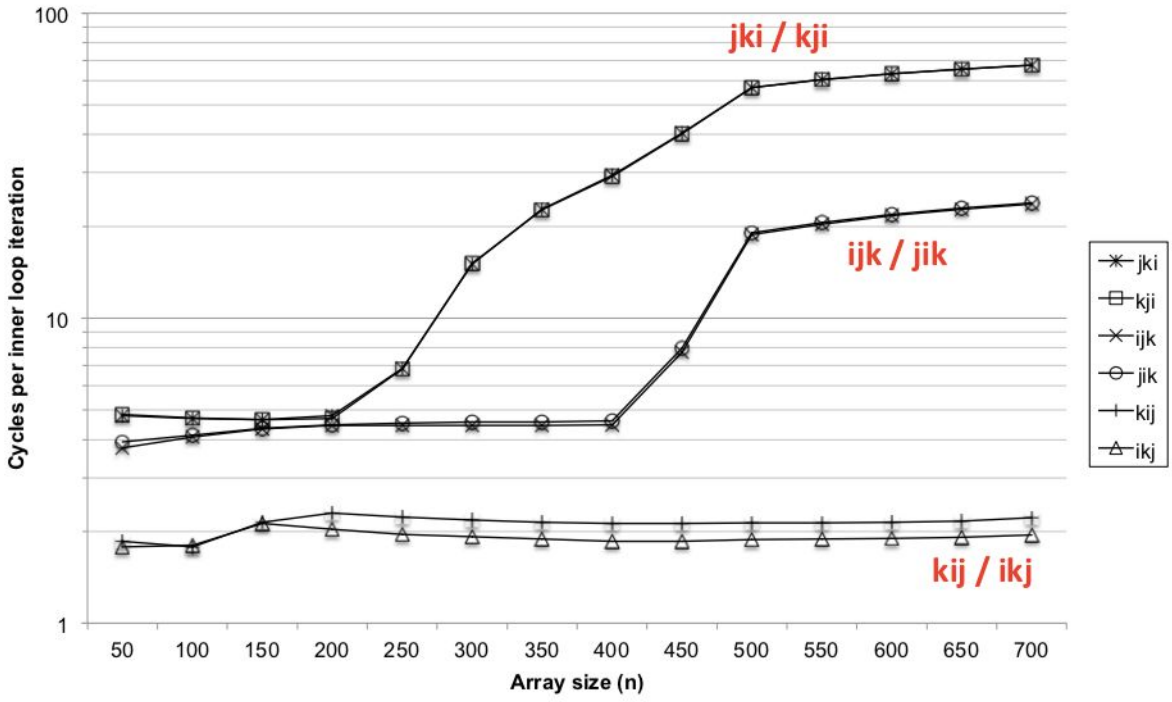
0x064C, 0x064D, 0x064E

0x064E

Question-3. Compile matrix multiplication and profile the code by changing by changing the size.

Answer - All versions of matrix multiplication code take different time to compute the product of two matrices. This is because row wise access is faster than column wise access. Moreover, there are more cache misses in column wise access because of which it computes the same product 40 times slower than row wise access.

This is evident from the graph drawn between array size and number of cycles per iteration -



The profiling of the code is done by gprof. It is the most widely used GNU profiling tool.

First adding a -pg flag in make file.

```
sayam@My_Ubuntu:~/Desktop/12-cache-memories/matmult$ cat Makefile
CC = gcc
CFLAGS = -O4 -Wall -mavx2 -pg

all: mm bmm

mm: mm.c clock.c fcycmm.c mm.h
        $(CC) $(CFLAGS) -o mm mm.c clock.c fcycmm.c

bmm: bmm.c clock.c fcycbmm.c mm.h
        $(CC) $(CFLAGS) -o bmm bmm.c clock.c fcycbmm.c

clean:
        rm -f *.o mm bmm *~
```

From 0-500 input size, the screenshots of analysis.txt file.

```
sayam@My_Ubuntu:~/Desktop/12-cache-memories/matmult$ gedit mm.c
sayam@My_Ubuntu:~/Desktop/12-cache-memories/matmult$ make clean
rm -f *.o mm bmm *~
sayam@My_Ubuntu:~/Desktop/12-cache-memories/matmult$ make
gcc -O4 -Wall -mavx2 -pg -o mm mm.c clock.c fcycmm.c
gcc -O4 -Wall -mavx2 -pg -o bmm bmm.c clock.c fcycbmm.c
sayam@My_Ubuntu:~/Desktop/12-cache-memories/matmult$ ./mm
matmult cycles/loop iteration
  n   jki   kji   ijk   jik   kij   ikj
 50  1.85  1.85  1.49  1.56  0.72  0.60
100  1.84  1.85  1.63  1.68  0.60  0.58
150  1.83  1.88  1.81  1.85  0.70  0.67
200  1.96  1.94  1.84  1.93  0.87  0.69
250  2.27  2.18  1.99  2.25  0.89  0.72
300  3.83  3.93  1.94  2.08  0.88  0.76
350  5.32  5.15  1.99  2.25  0.89  0.75
400  5.49  5.34  2.01  2.25  0.92  0.76
450  5.81  5.52  2.37  2.57  1.03  0.94
500  6.77  6.43  2.94  2.69  1.19  0.92
```

The screenshots of analysis.txt file -

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 36.71     29.89    29.89      163   183.38   183.38  jki
 31.73     55.73    25.84      139   185.88   185.88  kji
 16.07     68.81    13.08      168    77.88    77.88  ijk
  5.68     73.44     4.62      129    35.85    35.85  jik
  5.56     77.96     4.52      178    25.42    25.42  kij
  4.36     81.52     3.55      131    27.13    27.13  ikj
  0.00     81.52     0.00      908     0.00     0.00  get_counter
  0.00     81.52     0.00      908     0.00     0.00  reset
  0.00     81.52     0.00      908     0.00     0.00  start_counter



                                                 <spontaneous>
[1]    100.0    0.00   81.52                 fcyc [1]
                29.89    0.00     163/163         jki [2]
                25.84    0.00     139/139         kji [3]
                13.08    0.00     168/168         ijk [4]
                 4.62    0.00     129/129         jik [5]
                 4.52    0.00     178/178         kij [6]
                 3.55    0.00     131/131         ikj [7]
                 0.00    0.00     908/908         reset [9]
                 0.00    0.00     908/908         start_counter [10]
                 0.00    0.00     908/908         get_counter [8]
-----------------------------------------------
                29.89    0.00     163/163         fcyc [1]
[2]     36.7   29.89    0.00     163          jki [2]
-----------------------------------------------
                25.84    0.00     139/139         fcyc [1]
[3]     31.7   25.84    0.00     139          kji [3]
-----------------------------------------------
                13.08    0.00     168/168         fcyc [1]
[4]     16.1   13.08    0.00     168          ijk [4]
-----------------------------------------------
                 4.62    0.00     129/129         fcyc [1]
[5]      5.7    4.62    0.00     129          jik [5]
-----------------------------------------------
                 4.52    0.00     178/178         fcyc [1]
[6]      5.6    4.52    0.00     178          kij [6]
-----------------------------------------------
                 3.55    0.00     131/131         fcyc [1]
[7]      4.4    3.55    0.00     131          ikj [7]
-----------------------------------------------
```

From 500-1000 input size -

```
sayam@My_Ubuntu:~/Desktop/12-cache-memories/matmult$ make clean
rm -f *.o mm bmm *~
sayam@My_Ubuntu:~/Desktop/12-cache-memories/matmult$ make
gcc -O4 -Wall -mavx2 -pg -o mm mm.c clock.c fcycmm.c
gcc -O4 -Wall -mavx2 -pg -o bmm bmm.c clock.c fcycbmm.c
sayam@My_Ubuntu:~/Desktop/12-cache-memories/matmult$ ./mm
matmult cycles/loop iteration
  n   jki   kji   ijk   jik   kij   ikj
500  6.84  6.42  3.18  2.70  1.33  1.00
550  7.75  7.25  4.42  2.82  2.88  2.12
600  8.05  7.51  4.76  2.76  1.63  1.13
650  9.05  8.61  5.50  2.91  1.85  1.22
700 10.33  9.92  5.92  2.92  1.96  1.29
750 12.73 28.27  9.41  3.14  2.18  1.43
800 14.83 18.15  6.52  3.34  2.31  2.65
850 29.72 16.88  6.48  5.28  2.73  2.17
900 20.31 20.03  7.41  3.38  2.14  1.52
950 22.50 22.87  6.19  3.45  2.15  1.52
1000 26.24 26.30  6.00  3.36  2.05  1.46
sayam@My_Ubuntu:~/Desktop/12-cache-memories/matmult$
```

The screenshots of analysis.txt file -

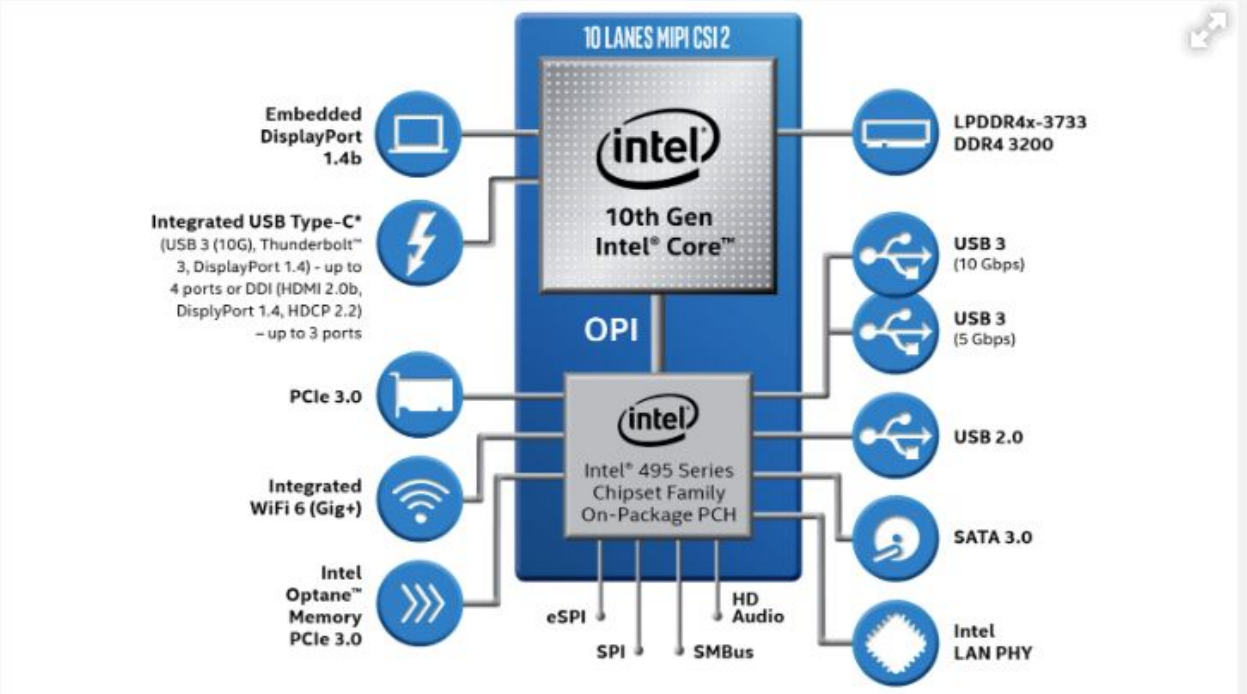
```
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 39.56    839.63   839.63      118     7.12     7.12  kji
 27.64   1426.29   586.66      173     3.39     3.39  jki
 15.73   1760.27   333.97      145     2.30     2.30  ijk
  8.56   1941.97   181.70      167     1.09     1.09  jik
  4.54   2038.43    96.46      158     0.61     0.61  kij
  4.07   2124.80    86.36      147     0.59     0.59  ikj
  0.00   2124.80     0.00      908     0.00     0.00  get_counter
  0.00   2124.80     0.00      908     0.00     0.00  reset
  0.00   2124.80     0.00      908     0.00     0.00  start_counter
```

```
                                                     <spontaneous>
[1]    100.0    0.00 2124.80                  fcyc [1]
              839.63    0.00     118/118          kji [2]
              586.66    0.00     173/173          jki [3]
              333.97    0.00     145/145          ijk [4]
              181.70    0.00     167/167          jik [5]
               96.46    0.00     158/158          kij [6]
               86.36    0.00     147/147          ikj [7]
                0.00    0.00     908/908          reset [9]
                0.00    0.00     908/908          start_counter [10]
                0.00    0.00     908/908          get_counter [8]
-----------------------------------------------
              839.63    0.00     118/118          fcyc [1]
[2]     39.5  839.63    0.00     118          kji [2]
-----------------------------------------------
              586.66    0.00     173/173          fcyc [1]
[3]     27.6  586.66    0.00     173          jki [3]
-----------------------------------------------
              333.97    0.00     145/145          fcyc [1]
[4]     15.7  333.97    0.00     145          ijk [4]
-----------------------------------------------
              181.70    0.00     167/167          fcyc [1]
[5]      8.6  181.70    0.00     167          jik [5]
-----------------------------------------------
               96.46    0.00     158/158          fcyc [1]
[6]      4.5   96.46    0.00     158          kij [6]
-----------------------------------------------
               86.36    0.00     147/147          fcyc [1]
[7]      4.1   86.36    0.00     147          ikj [7]
-----------------------------------------------
```

Conclusion - As the array size goes on increasing, the difference between the time taken by different versions also goes on increasing.

Question-4. Describe the 2019 state of the art Intel processor.
Answer - The state of the art processor is Intel 10th Gen Core U-Series and Y-Series i9. Specifications about processor is given below -
Number of cores - 18
Number of threads - 36
SATA Version - 3

Graphic Max Frequence - 1.10MHz
Max Single Core Turbo - 4.1
Cache - 8MB
Graphics EU - 64
Base Frequence - 2.3GHz
Nominal TDP - 28W



Question-5. Using gdb, disassemble the object code.
Answer - Using GNU gdb debugger, we are going to disassemble the code. Each function is converted to its corresponding assembly code by the compiler. We use -
disassemble <function_name>
to look at assembly instructions. Function calls have also been given address. It means the object file has been successfully loaded into the memory and is ready to get executed.

```
sayam@My_Ubuntu:~/Desktop/05-machine-basics$ make clean
rm -f *~ *.64s *.64d sum64
sayam@My_Ubuntu:~/Desktop/05-machine-basics$ make
gcc -Og  -S -m64 sum.c -o sum.64s
gcc -Og -Wall -g  -m64 sum.c -o sum
objdump -d sum > sum.64d
rm -f sum
gcc -Og -Wall -g  -m64 sum.c -o sum64
gcc -Og  -S -m64 swap.c -o swap.64s
gcc -Og  -S -m64 arith.c -o arith.64s
sayam@My_Ubuntu:~/Desktop/05-machine-basics$
```

```
sayam@My_Ubuntu:~/Desktop/05-machine-basics$ gcc sum.c -o sum -ggdb
sayam@My_Ubuntu:~/Desktop/05-machine-basics$ gdb sum
GNU gdb (Ubuntu 7.11.1-0ubuntu1~16.5) 7.11.1
Copyright (C) 2016 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from sum...done.
(gdb)
```

```
(gdb) disassemble sumstore
Dump of assembler code for function sumstore:
   0x00000000004005ef <+0>:     push   %rbp
   0x00000000004005f0 <+1>:     mov    %rsp,%rbp
   0x00000000004005f3 <+4>:     sub    $0x28,%rsp
   0x00000000004005f7 <+8>:     mov    %rdi,-0x18(%rbp)
   0x00000000004005fb <+12>:    mov    %rsi,-0x20(%rbp)
   0x00000000004005ff <+16>:    mov    %rdx,-0x28(%rbp)
   0x0000000000400603 <+20>:    mov    -0x20(%rbp),%rdx
   0x0000000000400607 <+24>:    mov    -0x18(%rbp),%rax
   0x000000000040060b <+28>:    mov    %rdx,%rsi
   0x000000000040060e <+31>:    mov    %rax,%rdi
   0x0000000000400611 <+34>:    callq  0x4005d6 <plus>
   0x0000000000400616 <+39>:    mov    %rax,-0x8(%rbp)
   0x000000000040061a <+43>:    mov    -0x28(%rbp),%rax
   0x000000000040061e <+47>:    mov    -0x8(%rbp),%rdx
   0x0000000000400622 <+51>:    mov    %rdx,(%rax)
   0x0000000000400625 <+54>:    nop
   0x0000000000400626 <+55>:    leaveq
   0x0000000000400627 <+56>:    retq
End of assembler dump.
(gdb)
```


(gdb) disassemble plus
Dump of assembler code for function plus:
0x00000000004005d6 <+0>: push %rbp
0x00000000004005d7 <+1>: mov %rsp,%rbp
0x00000000004005da <+4>: mov %rdi,-0x8(%rbp)
0x00000000004005de <+8>: mov %rsi,-0x10(%rbp)
0x00000000004005e2 <+12>: mov -0x8(%rbp),%rdx
0x00000000004005e6 <+16>: mov -0x10(%rbp),%rax
0x00000000004005ea <+20>: add %rdx,%rax
0x00000000004005ed <+23>: pop %rbp
0x00000000004005ee <+24>: retq
End of assembler dump.
(gdb)

Thank You!