TOSH CHAUHAN S20180010017

```
typedef struct {
    short code
    long start;
    char raw [3];
    double data;
} oldSensor Data ;
```

Code	
start	
[redacted]	
raw[0] row(1] row(3]	
data	
data	

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

Code	start
raw	
raw	sense
ext	
data	
data	

The data stored in the memory would look like

(a) new Data → Start = 0xff00

(b) newData → raw[0] = 0x b8

(c) newData → raw[2] = 0x 50

(d) newData → raw[4] = 0xe1

(e) newData → sense = 0x008f

4f	10	00	ff
b8	1a	50	80
e1	e2	8f	00

float { ~~float~~[1.5]
double

feal }
(1.5)

Ashutosh Chauhan.

Q.2

(i) 6.9

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

(ii) 6.11 (S, E, B, m) ⇒ (512, 1, 32, 32)

↳

9 bit	18bit	5bit
S	t	B

since 4096×4 → 16384 memory locations will be used the cache will map to the same set for each value. Since there is only One line per set (E = 1) at most only one line/array block will be filled in the cache.

Q.2 (iii) 6.12

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

(iv) 6.13 (A) Address.

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

B. Memory Reference

Cache Block offset (CO) → 0x 3

Cache set Index → 0x 6

Cache tag → 0x 6A

Cache hit → No

Cache byte Returned → –

ASHUTOSH CHAUHAN

Q.2.
(iv)
6.14 (A.)

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

(B.) Memory Reference

Cache Block Offset → 0x 0

Cache set Index → 0x 5

Cache Tag → 0X 65

Cache hit → No

Cache byte Returned → —

Q.2.
(vi)
6.15 (A.)

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

(B).

Memory Reference:

Cache Block offset → 0x0

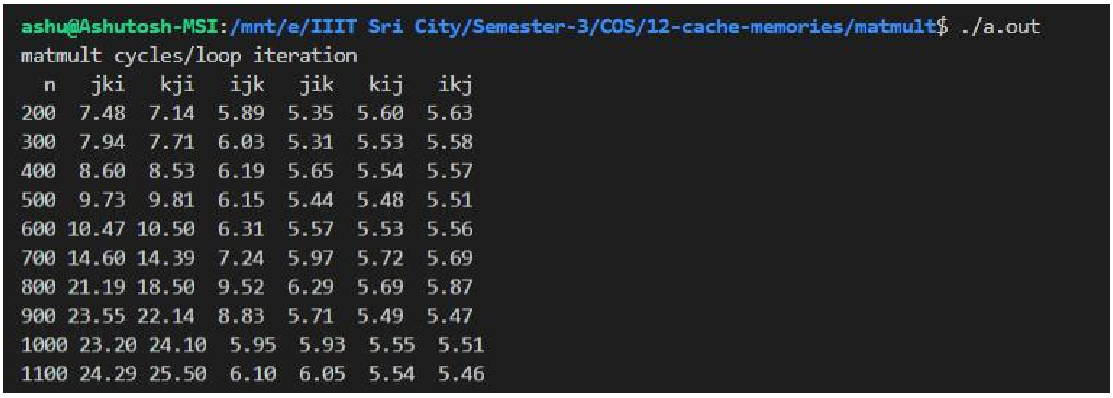
Cache set Index → 0x4

Cache Tag → 0x51

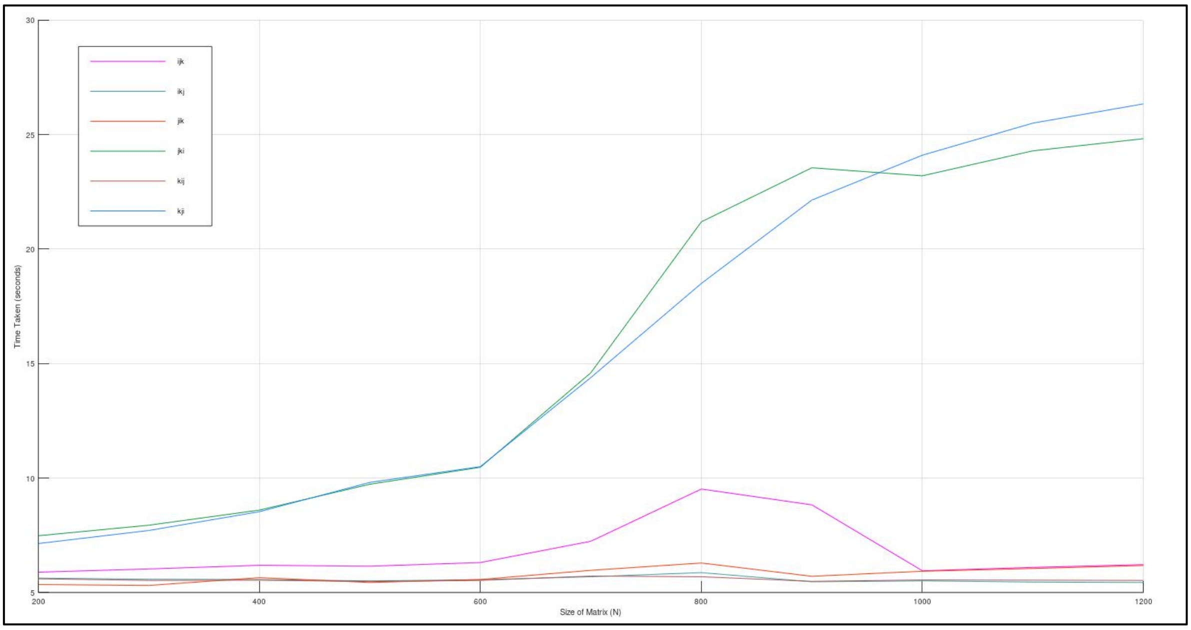
Cache hit → No

Cache Byte Returned → —

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



```
ashu@Ashutosh-MSI:/mnt/e/IIIT Sri City/Semester-3/COS/12-cache-memories/matmult$ ./a.out
matmult cycles/loop iteration
  n   jki   kji   ijk   jik   kij   ikj
200  7.48  7.14  5.89  5.35  5.60  5.63
300  7.94  7.71  6.03  5.31  5.53  5.58
400  8.60  8.53  6.19  5.65  5.54  5.57
500  9.73  9.81  6.15  5.44  5.48  5.51
600 10.47 10.50  6.31  5.57  5.53  5.56
700 14.60 14.39  7.24  5.97  5.72  5.69
800 21.19 18.50  9.52  6.29  5.69  5.87
900 23.55 22.14  8.83  5.71  5.49  5.47
1000 23.20 24.10  5.95  5.93  5.55  5.51
1100 24.29 25.50  6.10  6.05  5.54  5.46
```



On plotting the graph, we can verify that the loop in in the order *jki* , *kji* are more optimal than other.

---

4. Describe the 2019 state of the art Intel processor:

INTEL® CORE™ X-SERIES PROCESSOR FAMILY

Intel released the new X-Series Processor in its 10-Gen Core processors family.
This year Intel has released 4 Processor in X-Series as below:

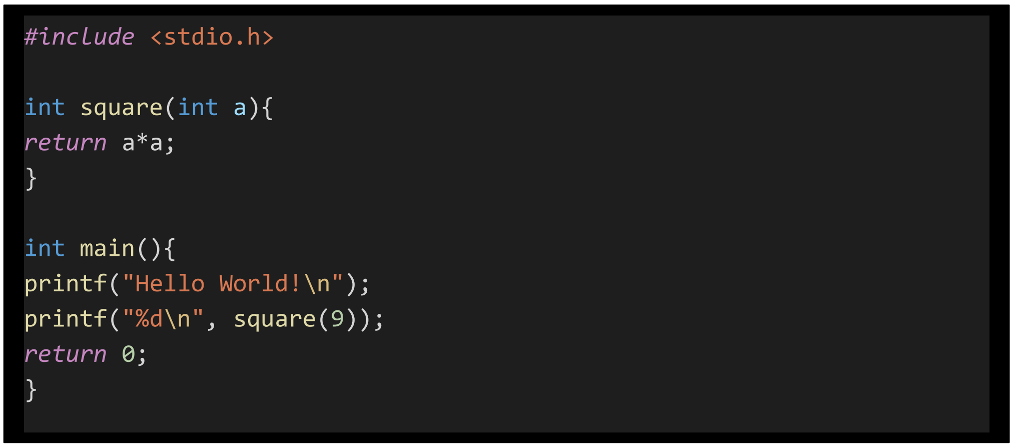
- Intel® Core™ i9-10980XE
- Intel® Core™ i9-10940X
- Intel® Core™ i9-10920X
- Intel® Core™ i9-10900X

Major Features:

- Contains upto 18 cores (36 Threads).
- Upto 4.60 GHz Turbo Frequency.
- 24.75 Mb Intel Smart Cache.
-

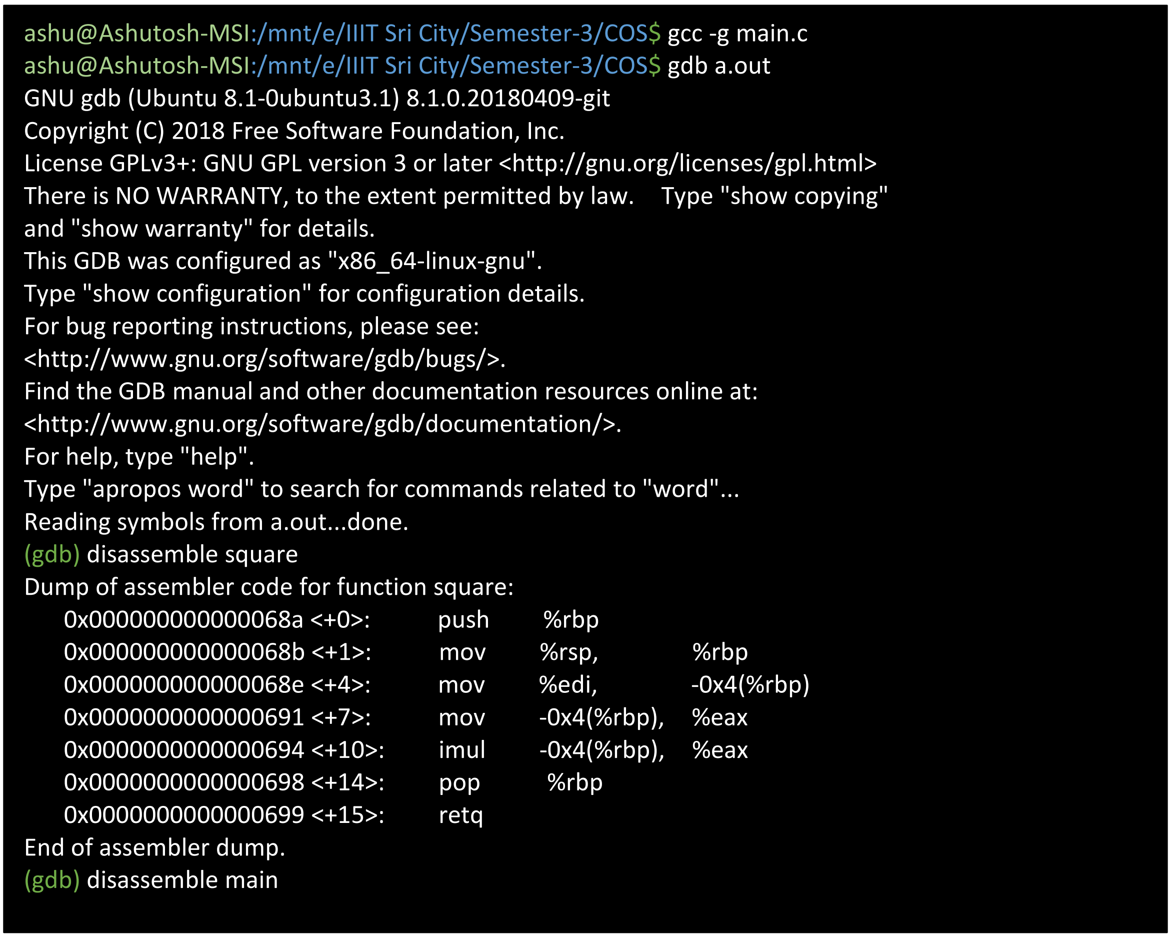
5. Using gdb, disassemble the object code.

Source Code:

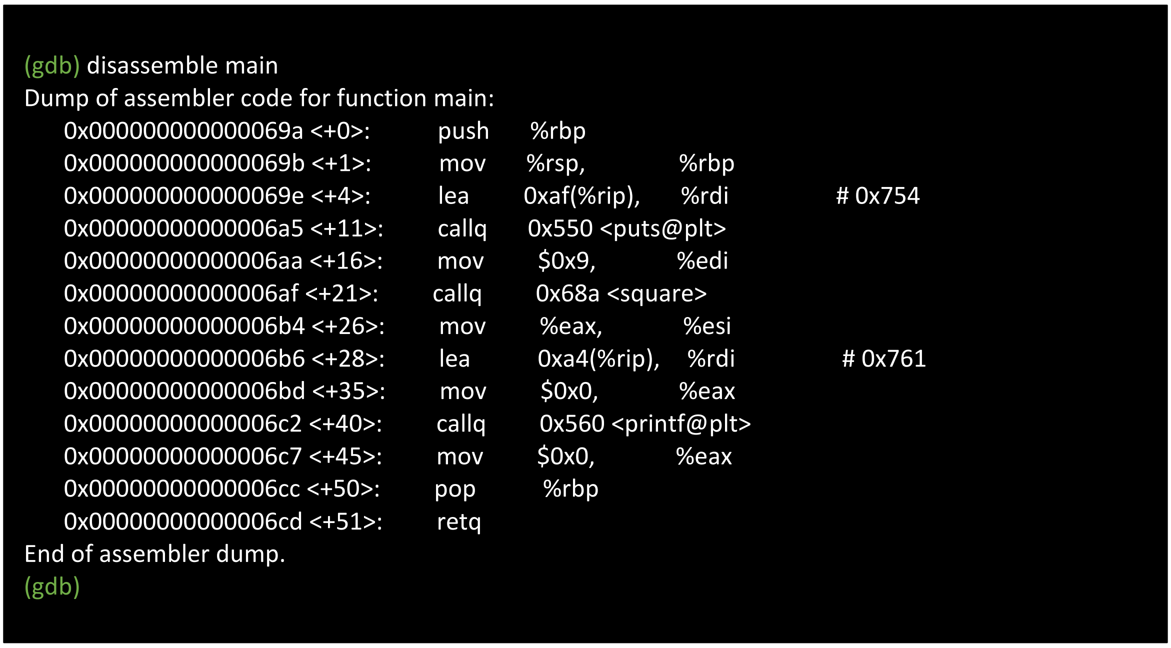


```c
#include <stdio.h>

int square(int a){
return a*a;
}

int main(){
printf("Hello World!\n");
printf("%d\n", square(9));
return 0;
}
```

GDB Output:



```
ashu@Ashutosh-MSI:/mnt/e/IIIT Sri City/Semester-3/COS$ gcc -g main.c
ashu@Ashutosh-MSI:/mnt/e/IIIT Sri City/Semester-3/COS$ gdb a.out
GNU gdb (Ubuntu 8.1-0ubuntu3.1) 8.1.0.20180409-git
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
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 a.out...done.
(gdb) disassemble square
Dump of assembler code for function square:
    0x000000000000068a <+0>:      push    %rbp
    0x000000000000068b <+1>:      mov     %rsp,         %rbp
    0x000000000000068e <+4>:      mov     %edi,         -0x4(%rbp)
    0x0000000000000691 <+7>:      mov     -0x4(%rbp),   %eax
    0x0000000000000694 <+10>:     imul    -0x4(%rbp),   %eax
    0x0000000000000698 <+14>:     pop     %rbp
    0x0000000000000699 <+15>:     retq
End of assembler dump.
(gdb) disassemble main
```



```
(gdb) disassemble main
Dump of assembler code for function main:
   0x000000000000069a <+0>:     push   %rbp
   0x000000000000069b <+1>:     mov    %rsp,         %rbp
   0x000000000000069e <+4>:     lea    0xaf(%rip),   %rdi          # 0x754
   0x00000000000006a5 <+11>:    callq  0x550 <puts@plt>
   0x00000000000006aa <+16>:    mov    $0x9,         %edi
   0x00000000000006af <+21>:    callq  0x68a <square>
   0x00000000000006b4 <+26>:    mov    %eax,         %esi
   0x00000000000006b6 <+28>:    lea    0xa4(%rip),   %rdi          # 0x761
   0x00000000000006bd <+35>:    mov    $0x0,         %eax
   0x00000000000006c2 <+40>:    callq  0x560 <printf@plt>
   0x00000000000006c7 <+45>:    mov    $0x0,         %eax
   0x00000000000006cc <+50>:    pop    %rbp
   0x00000000000006cd <+51>:    retq
End of assembler dump.
(gdb)
```