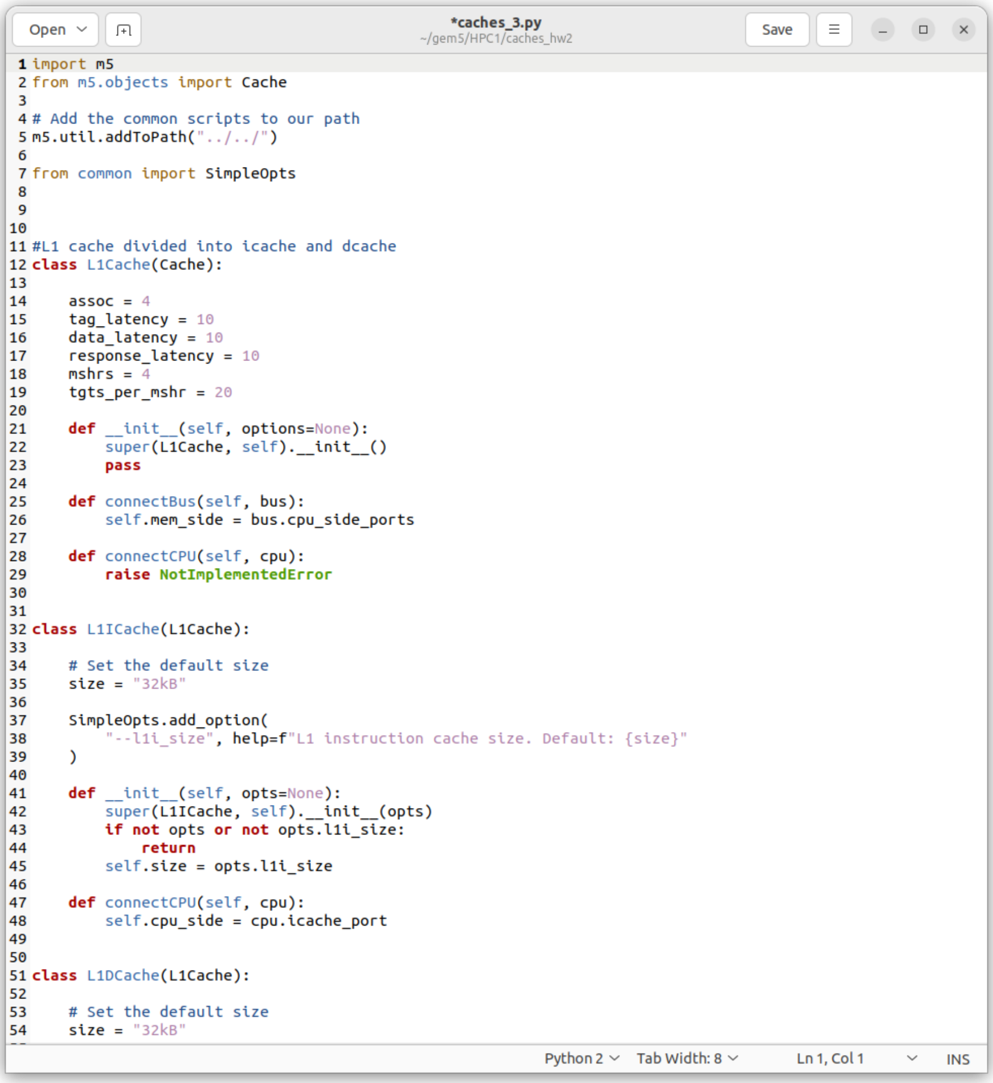
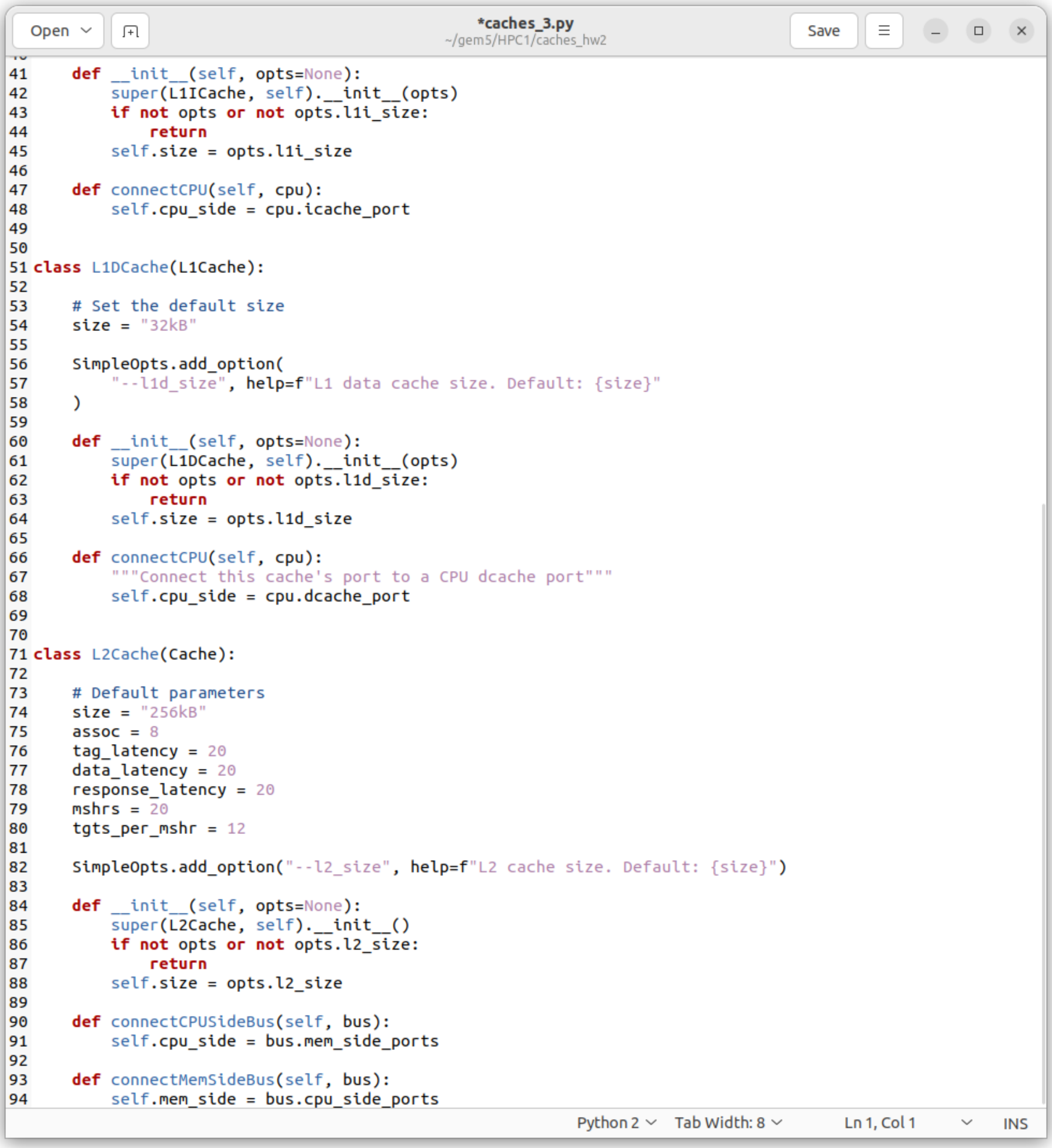
Lab Assignment-1   Part 2

Adding cache module to our processor description

Creating caches.py file- We can add all the required parameters of cache

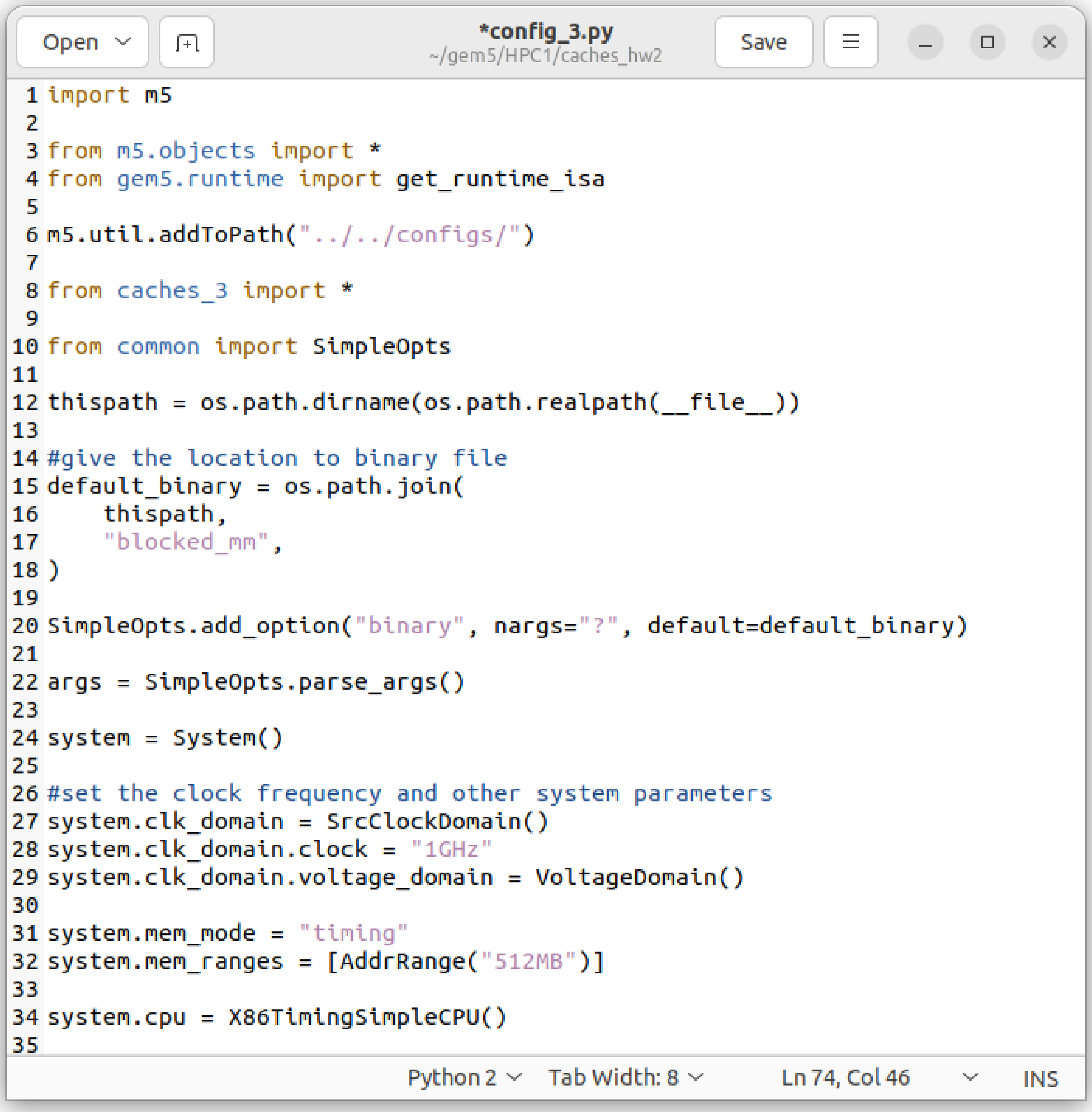


```python
import m5
from m5.objects import Cache

# Add the common scripts to our path
m5.util.addToPath("../../")

from common import SimpleOpts


#L1 cache divided into icache and dcache
class L1Cache(Cache):

    assoc = 4
    tag_latency = 10
    data_latency = 10
    response_latency = 10
    mshrs = 4
    tgts_per_mshr = 20

    def __init__(self, options=None):
        super(L1Cache, self).__init__()
        pass

    def connectBus(self, bus):
        self.mem_side = bus.cpu_side_ports

    def connectCPU(self, cpu):
        raise NotImplementedError


class L1ICache(L1Cache):

    # Set the default size
    size = "32kB"

    SimpleOpts.add_option(
        "--l1i_size", help=f"L1 instruction cache size. Default: {size}"
    )

    def __init__(self, opts=None):
        super(L1ICache, self).__init__(opts)
        if not opts or not opts.l1i_size:
            return
        self.size = opts.l1i_size

    def connectCPU(self, cpu):
        self.cpu_side = cpu.icache_port


class L1DCache(L1Cache):

    # Set the default size
    size = "32kB"
```

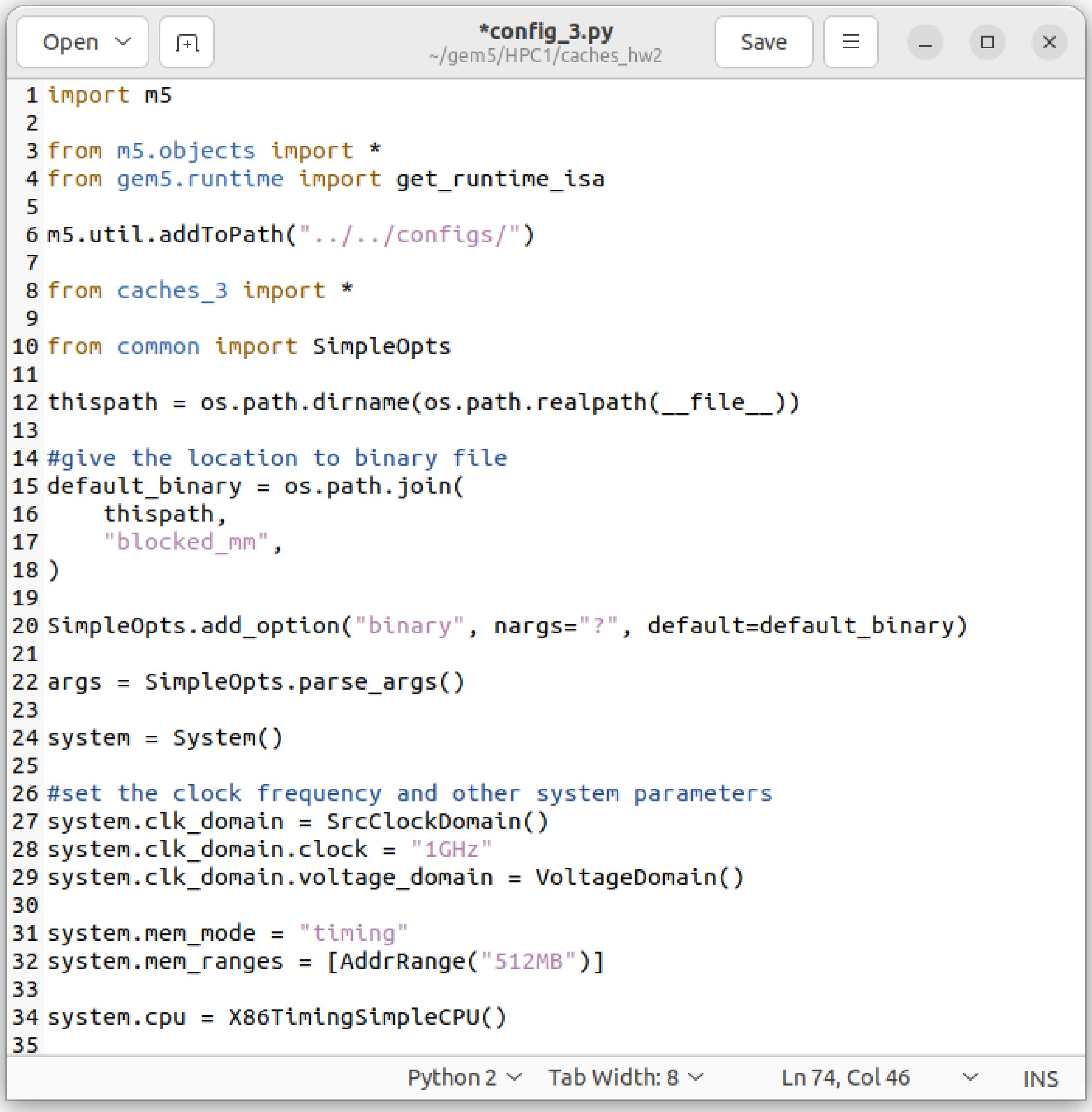


```
    def __init__(self, opts=None):
        super(L1ICache, self).__init__(opts)
        if not opts or not opts.l1i_size:
            return
        self.size = opts.l1i_size

    def connectCPU(self, cpu):
        self.cpu_side = cpu.icache_port


class L1DCache(L1Cache):

    # Set the default size
    size = "32kB"

    SimpleOpts.add_option(
        "--l1d_size", help=f"L1 data cache size. Default: {size}"
    )

    def __init__(self, opts=None):
        super(L1DCache, self).__init__(opts)
        if not opts or not opts.l1d_size:
            return
        self.size = opts.l1d_size

    def connectCPU(self, cpu):
        """Connect this cache's port to a CPU dcache port"""
        self.cpu_side = cpu.dcache_port


class L2Cache(Cache):

    # Default parameters
    size = "256kB"
    assoc = 8
    tag_latency = 20
    data_latency = 20
    response_latency = 20
    mshrs = 20
    tgts_per_mshr = 12

    SimpleOpts.add_option("--l2_size", help=f"L2 cache size. Default: {size}")

    def __init__(self, opts=None):
        super(L2Cache, self).__init__()
        if not opts or not opts.l2_size:
            return
        self.size = opts.l2_size

    def connectCPUSideBus(self, bus):
        self.cpu_side = bus.mem_side_ports

    def connectMemSideBus(self, bus):
        self.mem_side = bus.cpu_side_ports
```

Editing config.py file to add cache to our system

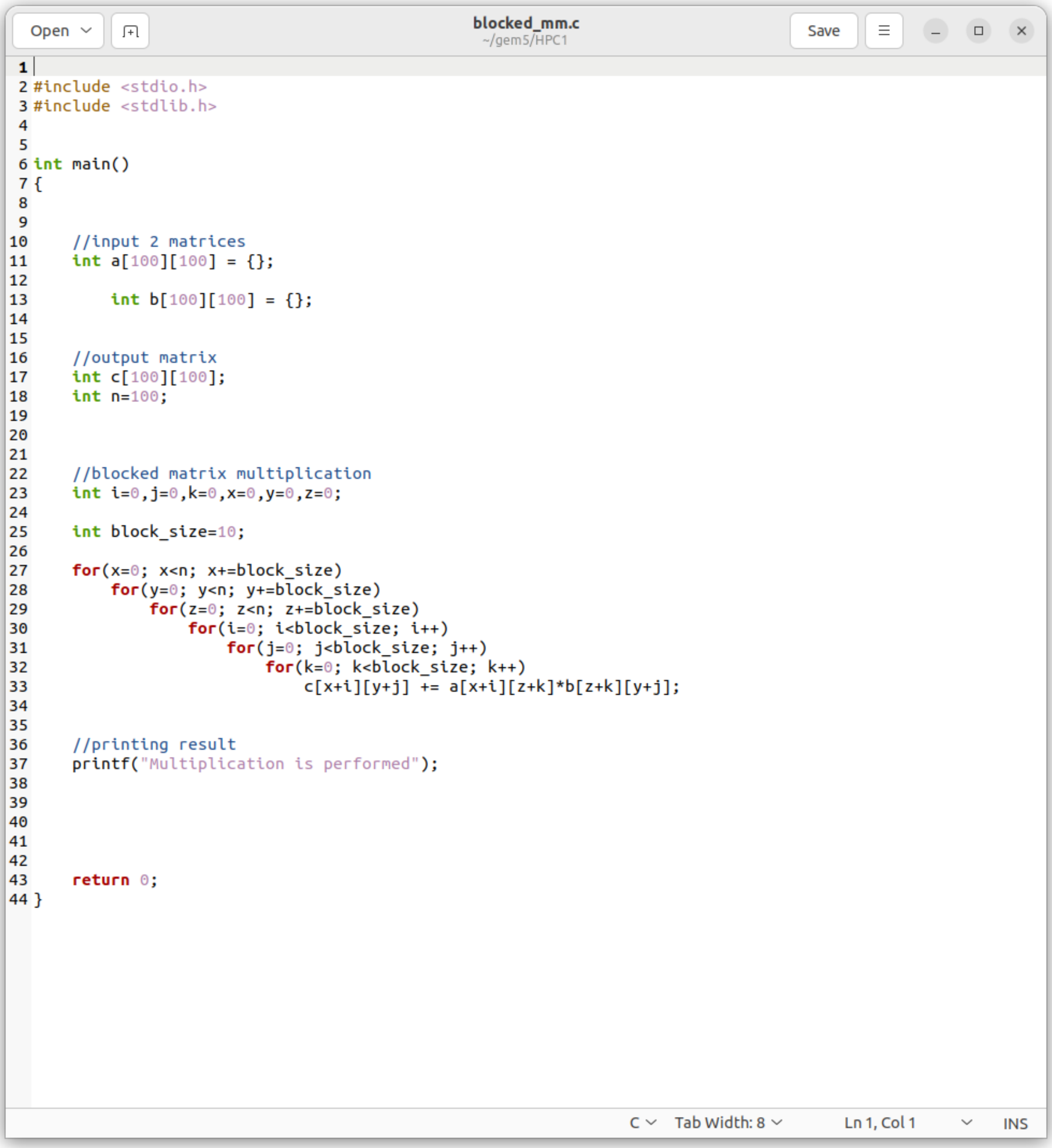


```
import m5

from m5.objects import *
from gem5.runtime import get_runtime_isa

m5.util.addToPath("../../configs/")

from caches_3 import *

from common import SimpleOpts

thispath = os.path.dirname(os.path.realpath(__file__))

#give the location to binary file
default_binary = os.path.join(
    thispath,
    "blocked_mm",
)

SimpleOpts.add_option("binary", nargs="?", default=default_binary)

args = SimpleOpts.parse_args()

system = System()

#set the clock frequency and other system parameters
system.clk_domain = SrcClockDomain()
system.clk_domain.clock = "1GHz"
system.clk_domain.voltage_domain = VoltageDomain()

system.mem_mode = "timing"
system.mem_ranges = [AddrRange("512MB")]

system.cpu = X86TimingSimpleCPU()

```



```python
import m5

from m5.objects import *
from gem5.runtime import get_runtime_isa

m5.util.addToPath("../../configs/")

from caches_3 import *

from common import SimpleOpts

thispath = os.path.dirname(os.path.realpath(__file__))

#give the location to binary file
default_binary = os.path.join(
    thispath,
    "blocked_mm",
)

SimpleOpts.add_option("binary", nargs="?", default=default_binary)

args = SimpleOpts.parse_args()

system = System()

#set the clock frequency and other system parameters
system.clk_domain = SrcClockDomain()
system.clk_domain.clock = "1GHz"
system.clk_domain.voltage_domain = VoltageDomain()

system.mem_mode = "timing"
system.mem_ranges = [AddrRange("512MB")]

system.cpu = X86TimingSimpleCPU()

```

Blocked version of matrix multiplication code-



```c

#include <stdio.h>
#include <stdlib.h>


int main()
{


	//input 2 matrices
	int a[100][100] = {};

		int b[100][100] = {};


	//output matrix
	int c[100][100];
	int n=100;



	//blocked matrix multiplication
	int i=0,j=0,k=0,x=0,y=0,z=0;

	int block_size=10;

	for(x=0; x<n; x+=block_size)
		for(y=0; y<n; y+=block_size)
			for(z=0; z<n; z+=block_size)
				for(i=0; i<block_size; i++)
					for(j=0; j<block_size; j++)
						for(k=0; k<block_size; k++)
							c[x+i][y+j] += a[x+i][z+k]*b[z+k][y+j];


	//printing result
	printf("Multiplication is performed");





	return 0;
}
```

Analysis:

For reference-

Stat1 – size of L1 instruction cache = 32KB

Size of L1 data cache = 32KB

Size of L2 cache = 256KB

Associativity = 2

tag latency, data latency, response latency =2

Stat2 – Updating associativity = 4

Stat3- Updating tag latency, data latency, response latency = 10

Stat4- size of L1 instruction cache = 128 KB

Size of L1 data cache = 128 KB

Size of L2 cache = 256KB

Associativity = 2

tag latency, data latency, response latency =2

Stat5- Updating associativity to above configuration = 4

Stat6- Updating tag latency, data latency, response latency =10

Stat7 - size of L1 instruction cache = 256 KB

Size of L1 data cache = 256 KB

Size of L2 cache = 256KB

Associativity = 2

tag latency, data latency, response latency =2

Stat8 - Updating associativity to above configuration = 4

Stat9- Updating tag latency, data latency, response latency =10

We compare few data points like total time for simulation, memory used, number of instructions, instruction rate for each block size.

Matrix Multiplication without blocking-

	Time for simulation(sec)	Memory used	No. of instructions	Instruction rate	Operation rate
Stat1	0.191509	652860	53215428	991527	1162449
Stat2	0.184562	652864	53215434	982295	1151626
Stat3	0.91659	652864	53215434	949192	1112816
Stat4	0.183176	653888	53215434	953600	1117984
Stat5	0.183176	653888	53215434	966366	1132951
Stat6	0.919147	653888	53215434	822485	964267
Stat7	0.255659	653888	53215434	748264	877252
Stat8	0.183176	653888	53215434	989191	1159710
Stat9	0.914739	653888	53215434	946782	1109990

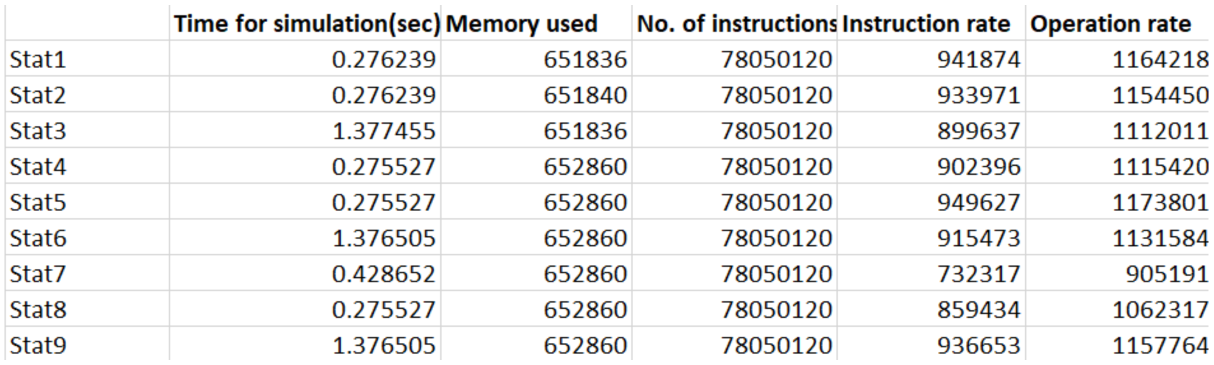
Blocked matrix multiplication with block size=2

	Time for simulation(sec)	Memory used	No. of instructions	Instruction rate	Operation rate
Stat1	0.276239	651840	78050120	958862	1185216
Stat2	0.276239	651840	78050120	938510	1160059
Stat3	1.377455	651840	78050120	932729	1152914
Stat4	0.275527	652864	78050120	911873	1127135
Stat5	0.275527	652860	78050120	952048	1176793
Stat6	1.376505	652860	78050120	912055	1127359
Stat7	0.428652	652860	78050120	750221	927322
Stat8	0.275527	652864	78050120	936552	1157639
Stat9	1.376505	652860	78050120	941390	1163619

Blocked matrix multiplication with block size=4

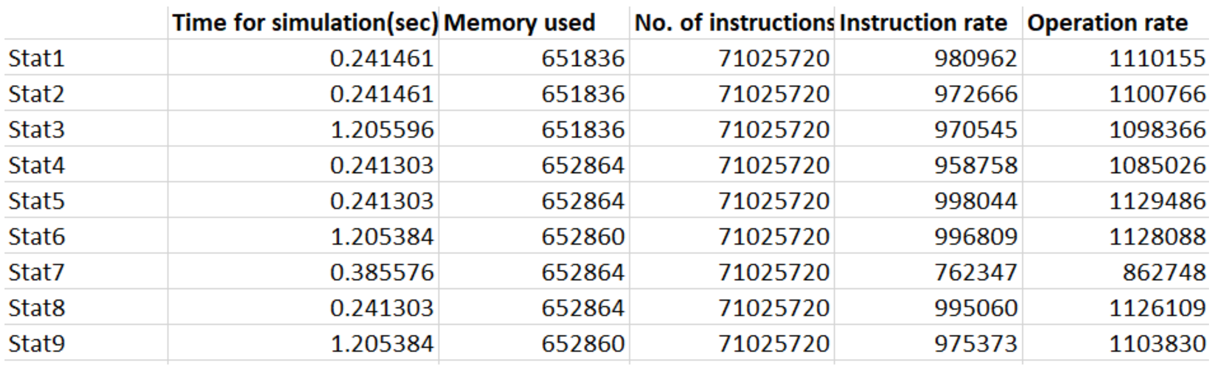
	Time for simulation(sec)	Memory used	No. of instructions	Instruction rate	Operation rate
Stat1	0.251305	651840	72999870	965085	1123081
Stat2	0.251305	651840	72999870	987896	1149626
Stat3	1.254058	651836	72999870	942115	1096350
Stat4	0.250942	652860	72999870	936907	1090290
Stat5	0.250941	652864	72999870	990046	1152127
Stat6	1.25357	652864	72999870	952973	1108985
Stat7	0.396851	652864	72999870	760212	884668
Stat8	0.250941	652860	72999870	1002227	1166303
Stat9	1.25357	652860	72999870	943010	1097391

Blocked matrix multiplication with block size=8



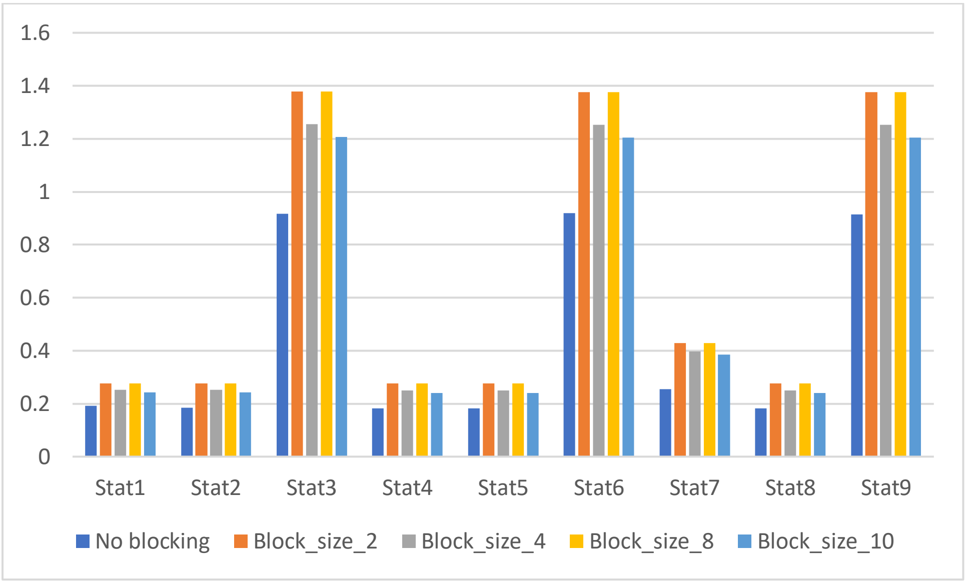
	Time for simulation(sec)	Memory used	No. of instructions	Instruction rate	Operation rate
Stat1	0.276239	651836	78050120	941874	1164218
Stat2	0.276239	651840	78050120	933971	1154450
Stat3	1.377455	651836	78050120	899637	1112011
Stat4	0.275527	652860	78050120	902396	1115420
Stat5	0.275527	652860	78050120	949627	1173801
Stat6	1.376505	652860	78050120	915473	1131584
Stat7	0.428652	652860	78050120	732317	905191
Stat8	0.275527	652860	78050120	859434	1062317
Stat9	1.376505	652860	78050120	936653	1157764

Blocked matrix multiplication with block size=10



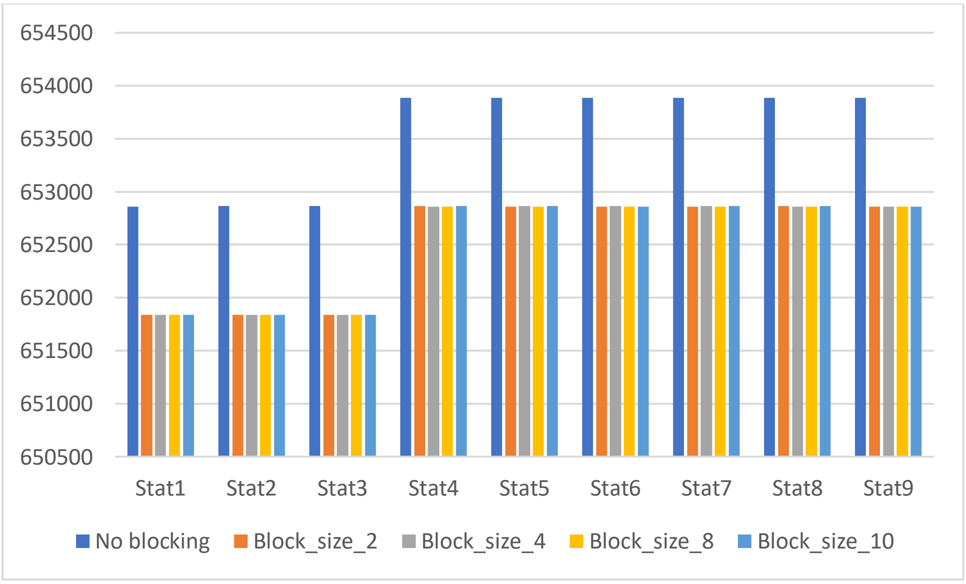
	Time for simulation(sec)	Memory used	No. of instructions	Instruction rate	Operation rate
Stat1	0.241461	651836	71025720	980962	1110155
Stat2	0.241461	651836	71025720	972666	1100766
Stat3	1.205596	651836	71025720	970545	1098366
Stat4	0.241303	652864	71025720	958758	1085026
Stat5	0.241303	652864	71025720	998044	1129486
Stat6	1.205384	652860	71025720	996809	1128088
Stat7	0.385576	652864	71025720	762347	862748
Stat8	0.241303	652864	71025720	995060	1126109
Stat9	1.205384	652860	71025720	975373	1103830

Comparing SimSeconds (Number of seconds simulated)-



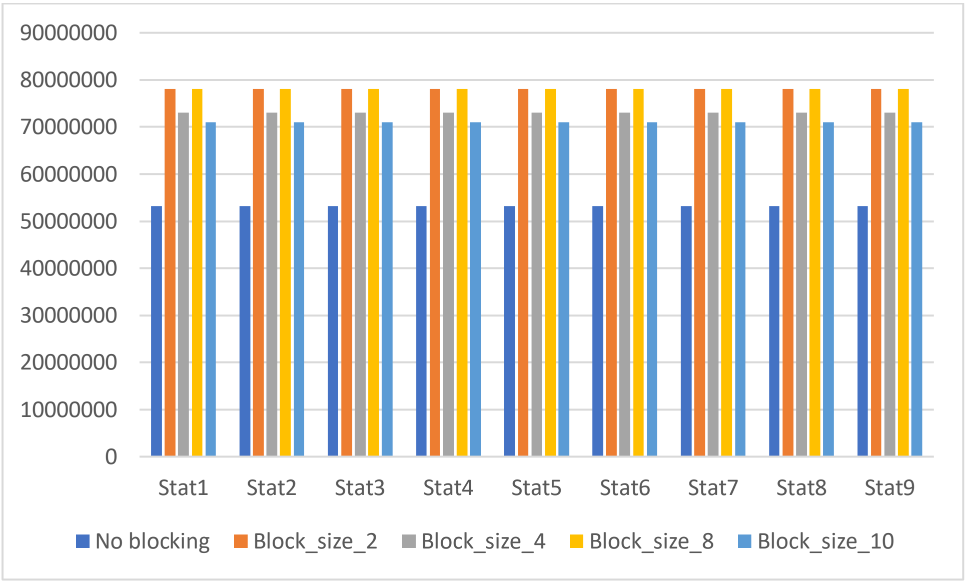
We notice that whenever we increase the latency, the total simulation time increases. This helps us conclude that latency time is responsible for deciding the speed of simulation irrespective of block size.

Comparing hostMemory (Number of Bytes of host memory used)



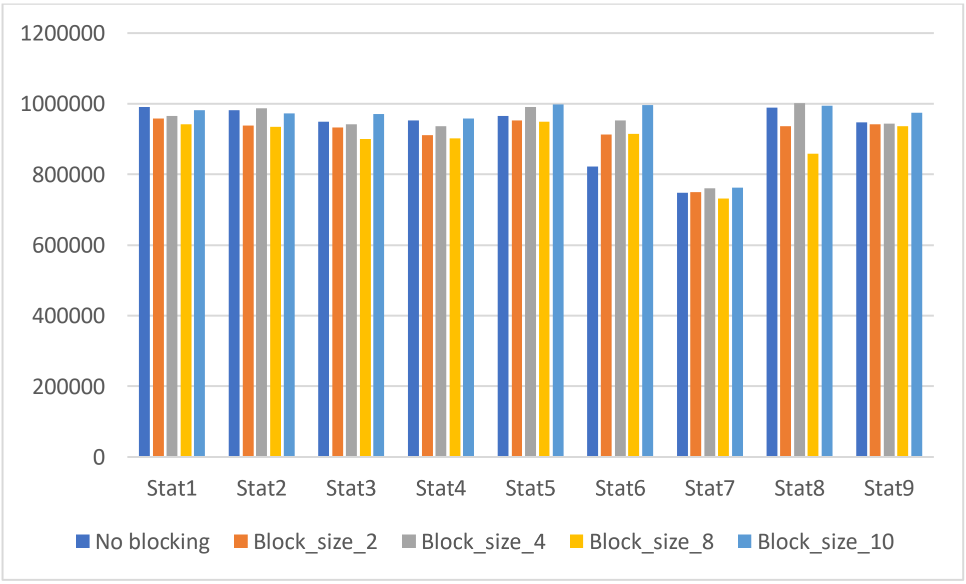
In this analysis, we can check that host memory increases as soon as L1/L2>=0.5. It also shows use of memory wherever blocking is not present.

Comparing simInsts (Number of instructions simulated)



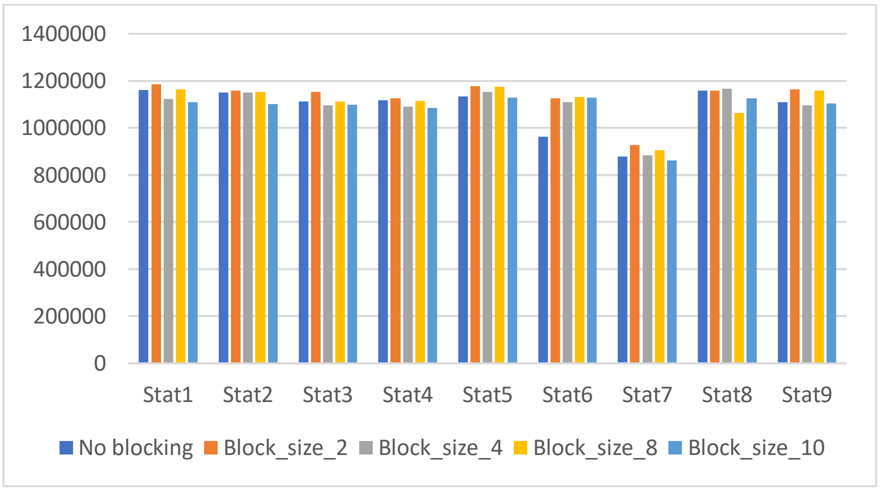
This plot displays that number of instructions depend only on block size.

Comparing hostInstRate (Simulator instruction rate (inst/s))



From previous assignment, we know that instruction simulation rate depends on CPU, and not much on cache. Although we can see some variations between block sizes and when L1/L2 size is considerably high.

Comparing hostOpRate (Simulator op (including micro-ops))



This plot is also similar to previous graph as both of these parameters largely depend on CPU performance. And we can see similar variations on changing cache parameters.