Gebze Technical University
Department of Computer Engineering
CSE 312 – Operating Systems
Semester Project Report

Hüseyin Emre Sekanlı
ID: 200104004049

1. Introduction

In this project I designed and implemented a minimal cooperative operating system (OS) for the GTU-C312 CPU and built a Python simulator to run it. My goals were to create a simple instruction set, enforce user- and kernel-mode memory protection, and demonstrate round-robin scheduling among three example threads: a sort routine, a linear search, and a loop that prints a constant.

2. CPU Simulator

I wrote the simulator in Python, reserving eleven thousand memory words and defining addresses 0–3 as special registers (program counter, stack pointer, syscall result, and a global instruction counter). The loader reads a text file with a Data Section and an Instruction Section, stripping comments marked by "#." In the main loop I repeatedly fetch one instruction at the program counter, execute it, increment the PC when appropriate, and bump the global instruction count.

To enforce protection, the simulator begins in kernel mode and only allows user-mode code to access addresses 1000 and above. Any attempt to read or write below 1000 in user mode halts the offending thread. When a USER instruction is executed in kernel mode, I record the thread's identifier, the current global instruction count as its start time, zero out its per-thread instruction count, and switch into user mode at the specified PC and stack pointer.

3. Thread Table Layout and Bookkeeping

I expanded each thread's table entry to six words in memory, storing:
• the thread ID;
• the start time (the global instruction count when the thread first ran);
• the number of instructions the thread has executed so far;
• its state (0 = ready, 1 = running, 2 = blocked, 3= terminated); #block not working on my project
• its saved program counter;
• and its saved stack pointer.

Whenever a user-mode instruction completes, I increment both the global and that thread's instruction counters. This lets me profile how many instructions each thread used during the simulation.

4. System Calls and Scheduling

I implemented three system calls, where threads would normally work on USER space, these calls would work on kernel:
• PRN prints a memory word to standard output and consumes an extra 100 ticks;
• YIELD saves the current thread's PC and SP back into its table entry, marks it ready, and then picks the next ready thread in round-robin order, restoring its context and resuming execution in user mode;
• HLT in user mode marks the thread terminated and likewise switches to the next ready thread or halts the CPU if none remain.

Thread selection is managed by scanning the table entries in ascending ID order, wrapping around from the last thread back to thread 0. This ensures fair time slices in a classic round-robin fashion.

5. OS Design in GTU-C312 Assembly

My assembly-level OS code begins in kernel mode at PC 0, where it initializes the stack pointer and sets each thread's state, PC, and SP fields in the thread table. A small dispatcher stub at PC 10–16 checks each thread's state and jumps to a launch block for any thread marked ready. In each launch block I set the state to running, execute a USER instruction to switch modes, and upon return use a conditional jump to either keep the thread running or mark it terminated before returning to the dispatcher.

In the Data Section, addresses 0–3 hold the special registers:
[0] = PC
[1] = SP
[2] = SYSCALL_RES
[3] = INSTR_COUNT
These track the program counter, stack pointer, syscall result, and global instruction count, respectively.

I maintain a thread table (addresses 21–86) where each entry stores:
tid | start_time | used_instrs | state | pc | sp

I have 11 threads ; an OS , 3 working and 7 non-working. The dispatcher scans TIDs 1 through 10 in ascending order, launching any thread whose state = READY.
• If a thread yields, it is marked READY and control returns to the dispatcher.
• If a thread terminates, it is marked TERMINATED and skipped on subsequent scans.
Threads resume execution from their saved PC/SP when re-dispatched, ensuring true round-robin scheduling.

The three user threads themselves reside at PCs 1000–1023, 2000–2032, and 3000–3010:

• Thread 1 reads three elements, performs pairwise comparisons and swaps using the **SUBI (subtract immediate)** instruction and conditional jumps, yields after printing the first element, then prints the remaining two and halts.

• Thread 2 loads an array and a key, loops with indirect addressing to search for the key, prints the index or –1, and then halts.

• Thread 3 loops a fixed number of times, printing “42” each iteration before halting.

6. Simulation Results

When I run the combined OS and simulator on the sample program, the output and scheduling order are as follows:

1. The sort thread yields after its first comparison (saving its PC),
2. The search thread yields before printing the result,
3. The loop thread prints “42” three times and yields,
4. The sort thread resumes at its saved PC, prints the remaining sorted values, and terminates,
5. The search thread prints the block number the searched key is in, and terminates
6. The loop thread prints 42 one more time and terminates.
7. The simulator halts with no threads left.

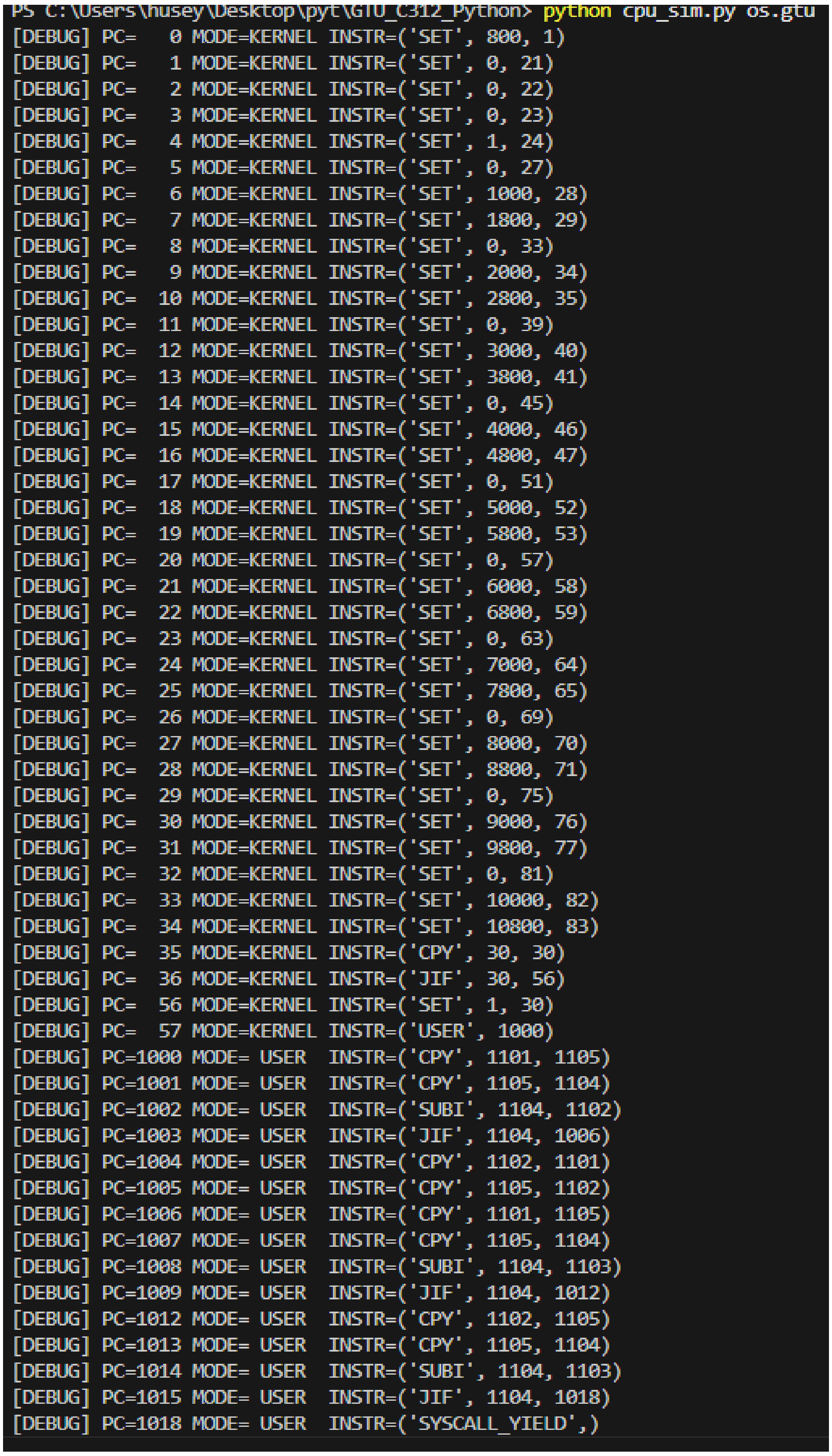
Basically SYSCALL YIELD yields the cpu for the next thread, and waits for the next time its queue comes. (round robin scheduling)

The global instruction counter and each thread’s per-thread counter confirm that each thread received exactly one time slice before yielding or halting.

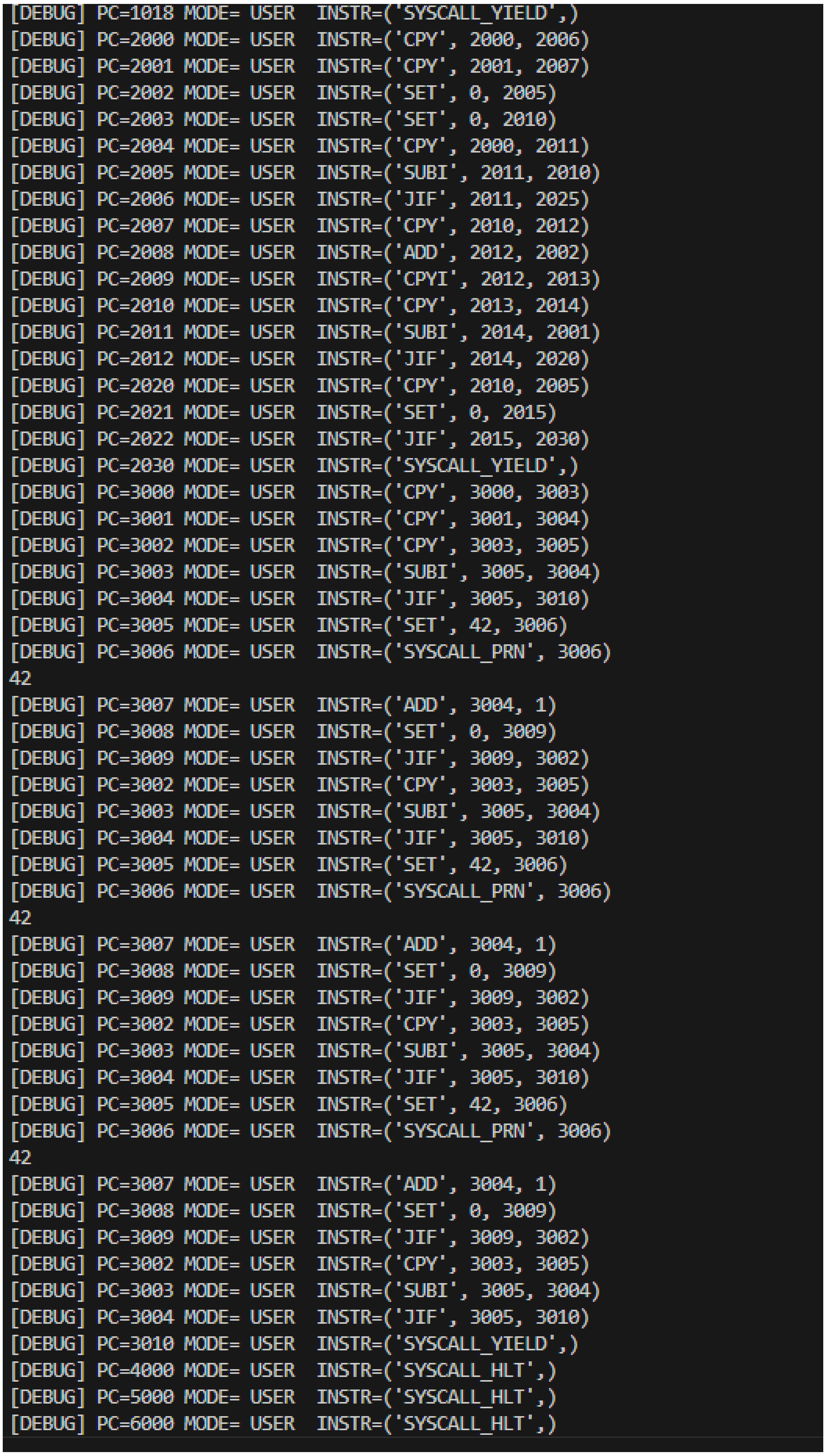
7. Conclusion

Through this project I gained hands-on experience with low-level CPU design, memory protection, and cooperative multitasking. I learned how to represent an OS and its threads in a unified instruction file, how to manage a thread-table data structure, and how to implement context switches in both a high-level language (Python) and a toy assembly language. My implementation meets the project requirements for scheduling correctness, profiling, and protection enforcement.

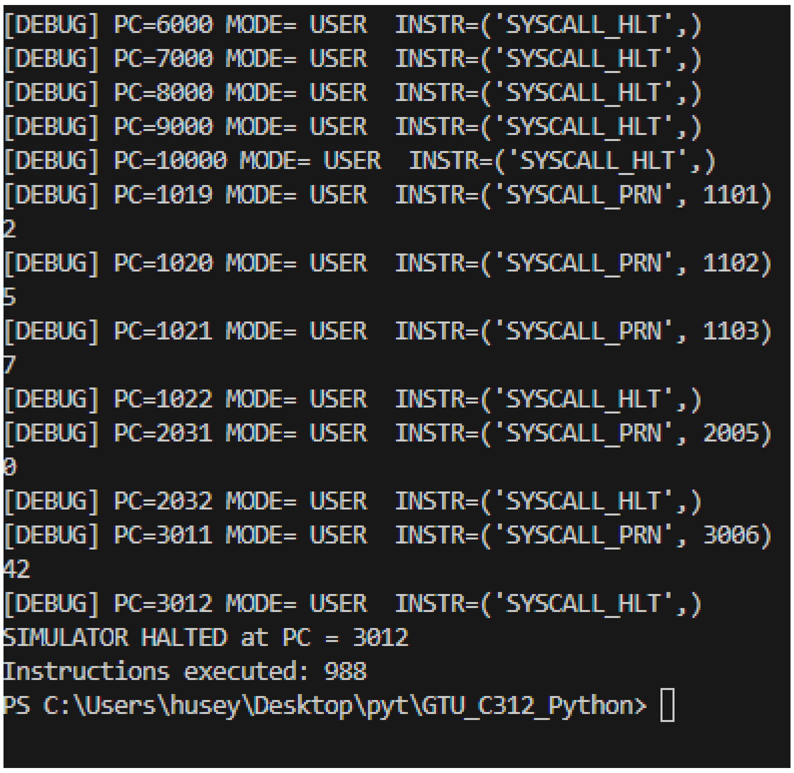
SS :



```
PS C:\Users\husey\Desktop\pyt\GTU_C312_Python> python cpu_sim.py os.gtu
[DEBUG] PC=   0 MODE=KERNEL INSTR=('SET', 800, 1)
[DEBUG] PC=   1 MODE=KERNEL INSTR=('SET', 0, 21)
[DEBUG] PC=   2 MODE=KERNEL INSTR=('SET', 0, 22)
[DEBUG] PC=   3 MODE=KERNEL INSTR=('SET', 0, 23)
[DEBUG] PC=   4 MODE=KERNEL INSTR=('SET', 1, 24)
[DEBUG] PC=   5 MODE=KERNEL INSTR=('SET', 0, 27)
[DEBUG] PC=   6 MODE=KERNEL INSTR=('SET', 1000, 28)
[DEBUG] PC=   7 MODE=KERNEL INSTR=('SET', 1800, 29)
[DEBUG] PC=   8 MODE=KERNEL INSTR=('SET', 0, 33)
[DEBUG] PC=   9 MODE=KERNEL INSTR=('SET', 2000, 34)
[DEBUG] PC=  10 MODE=KERNEL INSTR=('SET', 2800, 35)
[DEBUG] PC=  11 MODE=KERNEL INSTR=('SET', 0, 39)
[DEBUG] PC=  12 MODE=KERNEL INSTR=('SET', 3000, 40)
[DEBUG] PC=  13 MODE=KERNEL INSTR=('SET', 3800, 41)
[DEBUG] PC=  14 MODE=KERNEL INSTR=('SET', 0, 45)
[DEBUG] PC=  15 MODE=KERNEL INSTR=('SET', 4000, 46)
[DEBUG] PC=  16 MODE=KERNEL INSTR=('SET', 4800, 47)
[DEBUG] PC=  17 MODE=KERNEL INSTR=('SET', 0, 51)
[DEBUG] PC=  18 MODE=KERNEL INSTR=('SET', 5000, 52)
[DEBUG] PC=  19 MODE=KERNEL INSTR=('SET', 5800, 53)
[DEBUG] PC=  20 MODE=KERNEL INSTR=('SET', 0, 57)
[DEBUG] PC=  21 MODE=KERNEL INSTR=('SET', 6000, 58)
[DEBUG] PC=  22 MODE=KERNEL INSTR=('SET', 6800, 59)
[DEBUG] PC=  23 MODE=KERNEL INSTR=('SET', 0, 63)
[DEBUG] PC=  24 MODE=KERNEL INSTR=('SET', 7000, 64)
[DEBUG] PC=  25 MODE=KERNEL INSTR=('SET', 7800, 65)
[DEBUG] PC=  26 MODE=KERNEL INSTR=('SET', 0, 69)
[DEBUG] PC=  27 MODE=KERNEL INSTR=('SET', 8000, 70)
[DEBUG] PC=  28 MODE=KERNEL INSTR=('SET', 8800, 71)
[DEBUG] PC=  29 MODE=KERNEL INSTR=('SET', 0, 75)
[DEBUG] PC=  30 MODE=KERNEL INSTR=('SET', 9000, 76)
[DEBUG] PC=  31 MODE=KERNEL INSTR=('SET', 9800, 77)
[DEBUG] PC=  32 MODE=KERNEL INSTR=('SET', 0, 81)
[DEBUG] PC=  33 MODE=KERNEL INSTR=('SET', 10000, 82)
[DEBUG] PC=  34 MODE=KERNEL INSTR=('SET', 10800, 83)
[DEBUG] PC=  35 MODE=KERNEL INSTR=('CPY', 30, 30)
[DEBUG] PC=  36 MODE=KERNEL INSTR=('JIF', 30, 56)
[DEBUG] PC=  56 MODE=KERNEL INSTR=('SET', 1, 30)
[DEBUG] PC=  57 MODE=KERNEL INSTR=('USER', 1000)
[DEBUG] PC=1000 MODE= USER  INSTR=('CPY', 1101, 1105)
[DEBUG] PC=1001 MODE= USER  INSTR=('CPY', 1105, 1104)
[DEBUG] PC=1002 MODE= USER  INSTR=('SUBI', 1104, 1102)
[DEBUG] PC=1003 MODE= USER  INSTR=('JIF', 1104, 1006)
[DEBUG] PC=1004 MODE= USER  INSTR=('CPY', 1102, 1101)
[DEBUG] PC=1005 MODE= USER  INSTR=('CPY', 1105, 1102)
[DEBUG] PC=1006 MODE= USER  INSTR=('CPY', 1101, 1105)
[DEBUG] PC=1007 MODE= USER  INSTR=('CPY', 1105, 1104)
[DEBUG] PC=1008 MODE= USER  INSTR=('SUBI', 1104, 1103)
[DEBUG] PC=1009 MODE= USER  INSTR=('JIF', 1104, 1012)
[DEBUG] PC=1012 MODE= USER  INSTR=('CPY', 1102, 1105)
[DEBUG] PC=1013 MODE= USER  INSTR=('CPY', 1105, 1104)
[DEBUG] PC=1014 MODE= USER  INSTR=('SUBI', 1104, 1103)
[DEBUG] PC=1015 MODE= USER  INSTR=('JIF', 1104, 1018)
[DEBUG] PC=1018 MODE= USER  INSTR=('SYSCALL_YIELD',)
```



```
[DEBUG] PC=1018 MODE= USER  INSTR=('SYSCALL_YIELD',)
[DEBUG] PC=2000 MODE= USER  INSTR=('CPY', 2000, 2006)
[DEBUG] PC=2001 MODE= USER  INSTR=('CPY', 2001, 2007)
[DEBUG] PC=2002 MODE= USER  INSTR=('SET', 0, 2005)
[DEBUG] PC=2003 MODE= USER  INSTR=('SET', 0, 2010)
[DEBUG] PC=2004 MODE= USER  INSTR=('CPY', 2000, 2011)
[DEBUG] PC=2005 MODE= USER  INSTR=('SUBI', 2011, 2010)
[DEBUG] PC=2006 MODE= USER  INSTR=('JIF', 2011, 2025)
[DEBUG] PC=2007 MODE= USER  INSTR=('CPY', 2010, 2012)
[DEBUG] PC=2008 MODE= USER  INSTR=('ADD', 2012, 2002)
[DEBUG] PC=2009 MODE= USER  INSTR=('CPYI', 2012, 2013)
[DEBUG] PC=2010 MODE= USER  INSTR=('CPY', 2013, 2014)
[DEBUG] PC=2011 MODE= USER  INSTR=('SUBI', 2014, 2001)
[DEBUG] PC=2012 MODE= USER  INSTR=('JIF', 2014, 2020)
[DEBUG] PC=2020 MODE= USER  INSTR=('CPY', 2010, 2005)
[DEBUG] PC=2021 MODE= USER  INSTR=('SET', 0, 2015)
[DEBUG] PC=2022 MODE= USER  INSTR=('JIF', 2015, 2030)
[DEBUG] PC=2030 MODE= USER  INSTR=('SYSCALL_YIELD',)
[DEBUG] PC=3000 MODE= USER  INSTR=('CPY', 3000, 3003)
[DEBUG] PC=3001 MODE= USER  INSTR=('CPY', 3001, 3004)
[DEBUG] PC=3002 MODE= USER  INSTR=('CPY', 3003, 3005)
[DEBUG] PC=3003 MODE= USER  INSTR=('SUBI', 3005, 3004)
[DEBUG] PC=3004 MODE= USER  INSTR=('JIF', 3005, 3010)
[DEBUG] PC=3005 MODE= USER  INSTR=('SET', 42, 3006)
[DEBUG] PC=3006 MODE= USER  INSTR=('SYSCALL_PRN', 3006)
42
[DEBUG] PC=3007 MODE= USER  INSTR=('ADD', 3004, 1)
[DEBUG] PC=3008 MODE= USER  INSTR=('SET', 0, 3009)
[DEBUG] PC=3009 MODE= USER  INSTR=('JIF', 3009, 3002)
[DEBUG] PC=3002 MODE= USER  INSTR=('CPY', 3003, 3005)
[DEBUG] PC=3003 MODE= USER  INSTR=('SUBI', 3005, 3004)
[DEBUG] PC=3004 MODE= USER  INSTR=('JIF', 3005, 3010)
[DEBUG] PC=3005 MODE= USER  INSTR=('SET', 42, 3006)
[DEBUG] PC=3006 MODE= USER  INSTR=('SYSCALL_PRN', 3006)
42
[DEBUG] PC=3007 MODE= USER  INSTR=('ADD', 3004, 1)
[DEBUG] PC=3008 MODE= USER  INSTR=('SET', 0, 3009)
[DEBUG] PC=3009 MODE= USER  INSTR=('JIF', 3009, 3002)
[DEBUG] PC=3002 MODE= USER  INSTR=('CPY', 3003, 3005)
[DEBUG] PC=3003 MODE= USER  INSTR=('SUBI', 3005, 3004)
[DEBUG] PC=3004 MODE= USER  INSTR=('JIF', 3005, 3010)
[DEBUG] PC=3005 MODE= USER  INSTR=('SET', 42, 3006)
[DEBUG] PC=3006 MODE= USER  INSTR=('SYSCALL_PRN', 3006)
42
[DEBUG] PC=3007 MODE= USER  INSTR=('ADD', 3004, 1)
[DEBUG] PC=3008 MODE= USER  INSTR=('SET', 0, 3009)
[DEBUG] PC=3009 MODE= USER  INSTR=('JIF', 3009, 3002)
[DEBUG] PC=3002 MODE= USER  INSTR=('CPY', 3003, 3005)
[DEBUG] PC=3003 MODE= USER  INSTR=('SUBI', 3005, 3004)
[DEBUG] PC=3004 MODE= USER  INSTR=('JIF', 3005, 3010)
[DEBUG] PC=3010 MODE= USER  INSTR=('SYSCALL_YIELD',)
[DEBUG] PC=4000 MODE= USER  INSTR=('SYSCALL_HLT',)
[DEBUG] PC=5000 MODE= USER  INSTR=('SYSCALL_HLT',)
[DEBUG] PC=6000 MODE= USER  INSTR=('SYSCALL_HLT',)
```



```
[DEBUG] PC=6000 MODE= USER  INSTR=('SYSCALL_HLT',)
[DEBUG] PC=7000 MODE= USER  INSTR=('SYSCALL_HLT',)
[DEBUG] PC=8000 MODE= USER  INSTR=('SYSCALL_HLT',)
[DEBUG] PC=9000 MODE= USER  INSTR=('SYSCALL_HLT',)
[DEBUG] PC=10000 MODE= USER  INSTR=('SYSCALL_HLT',)
[DEBUG] PC=1019 MODE= USER  INSTR=('SYSCALL_PRN', 1101)
2
[DEBUG] PC=1020 MODE= USER  INSTR=('SYSCALL_PRN', 1102)
5
[DEBUG] PC=1021 MODE= USER  INSTR=('SYSCALL_PRN', 1103)
7
[DEBUG] PC=1022 MODE= USER  INSTR=('SYSCALL_HLT',)
[DEBUG] PC=2031 MODE= USER  INSTR=('SYSCALL_PRN', 2005)
0
[DEBUG] PC=2032 MODE= USER  INSTR=('SYSCALL_HLT',)
[DEBUG] PC=3011 MODE= USER  INSTR=('SYSCALL_PRN', 3006)
42
[DEBUG] PC=3012 MODE= USER  INSTR=('SYSCALL_HLT',)
SIMULATOR HALTED at PC = 3012
Instructions executed: 988
PS C:\Users\husey\Desktop\pyt\GTU_C312_Python>
```

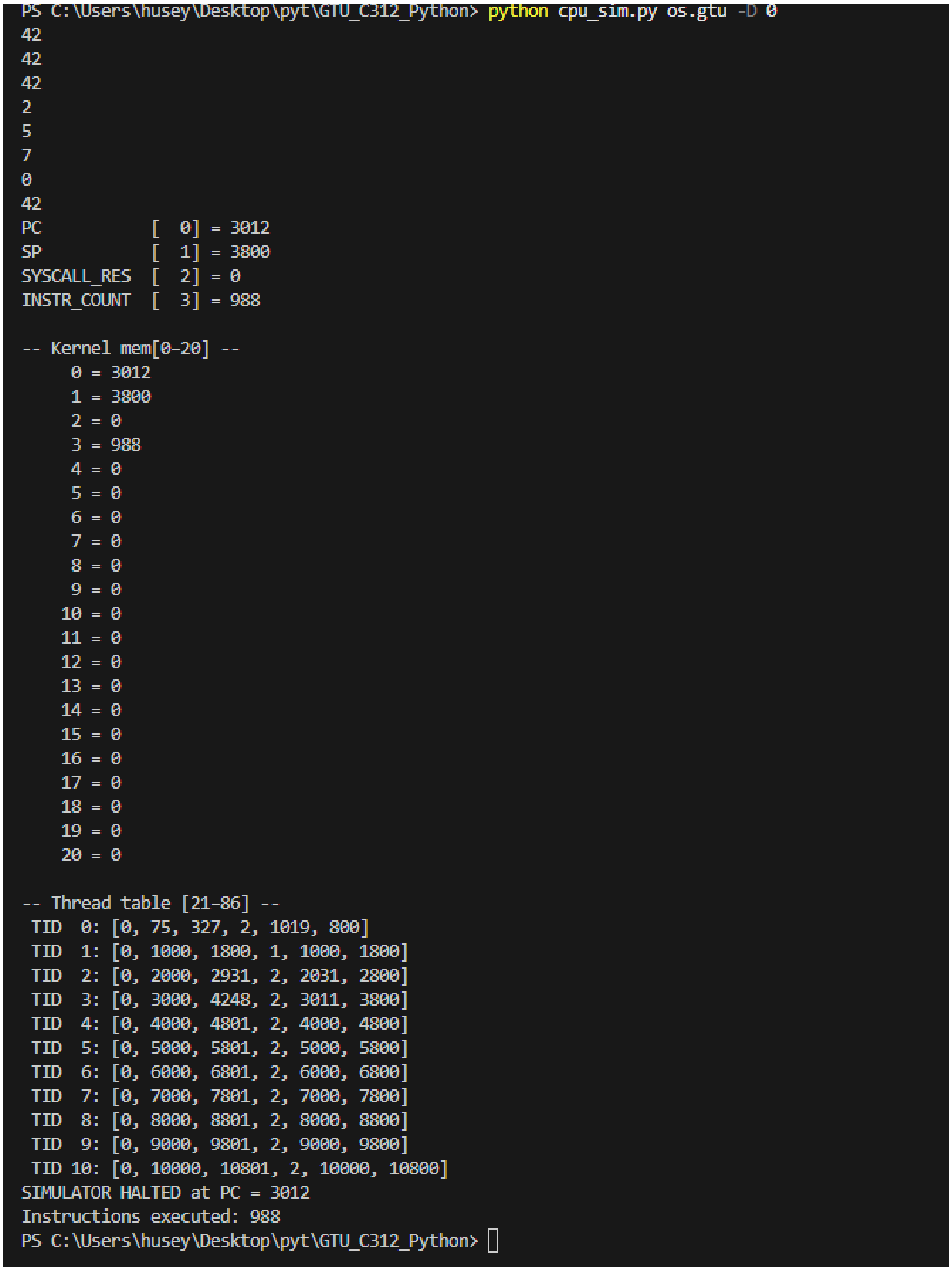
On T1, it sorts the elements 2,5,7 and prints.

On T2, it found the element it searched for is in a[0]

On T3, above 3 prints are before yield, the last print is after yield.

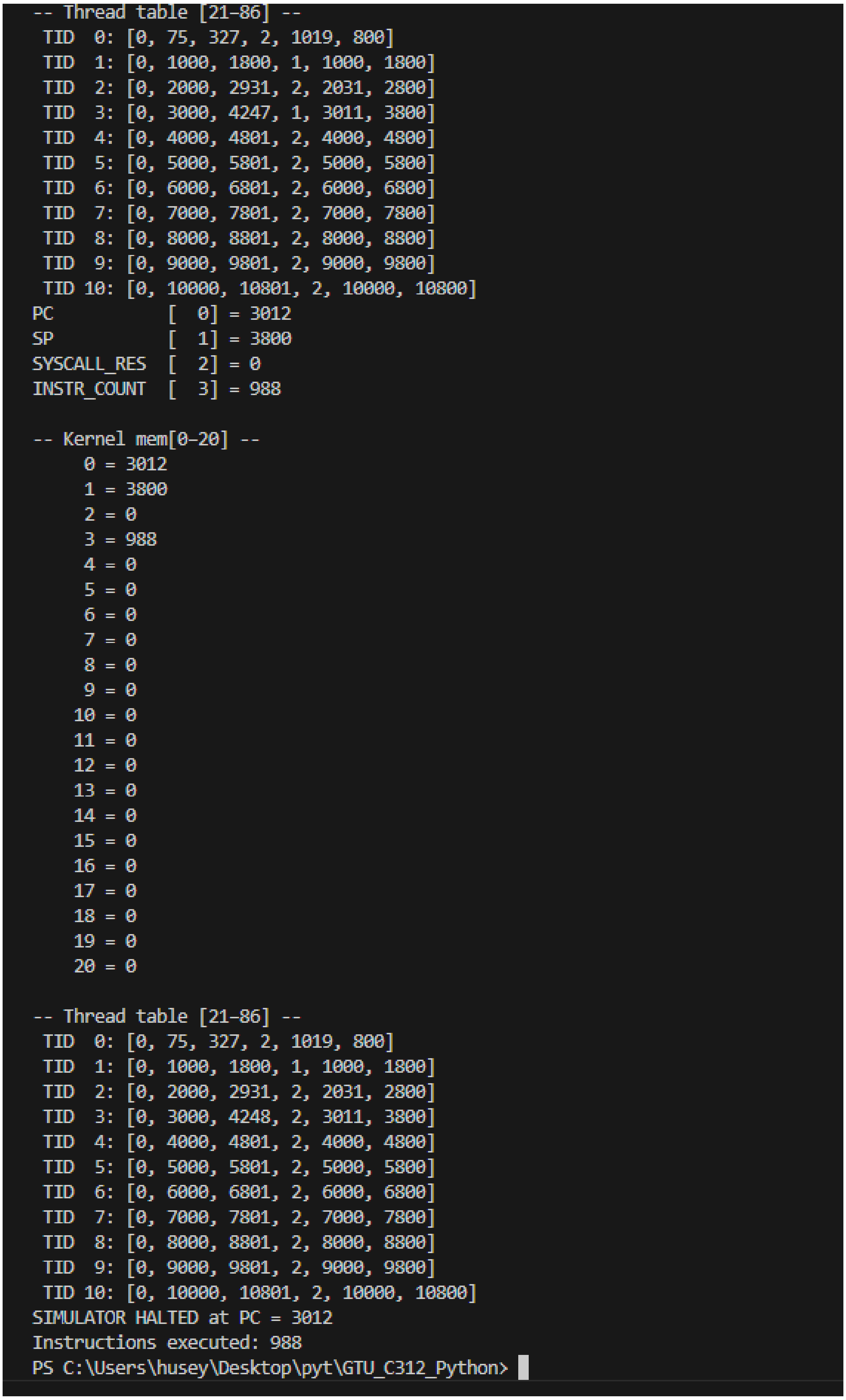
This debug is something i made along the way to follow the program properly

PDF Debugs : -D 0 :



```
PS C:\Users\husey\Desktop\pyt\GTU_C312_Python> python cpu_sim.py os.gtu -D 0
42
42
42
2
5
7
0
42
PC            [  0] = 3012
SP            [  1] = 3800
SYSCALL_RES   [  2] = 0
INSTR_COUNT   [  3] = 988

-- Kernel mem[0-20] --
     0 = 3012
     1 = 3800
     2 = 0
     3 = 988
     4 = 0
     5 = 0
     6 = 0
     7 = 0
     8 = 0
     9 = 0
    10 = 0
    11 = 0
    12 = 0
    13 = 0
    14 = 0
    15 = 0
    16 = 0
    17 = 0
    18 = 0
    19 = 0
    20 = 0

-- Thread table [21-86] --
 TID  0: [0, 75, 327, 2, 1019, 800]
 TID  1: [0, 1000, 1800, 1, 1000, 1800]
 TID  2: [0, 2000, 2931, 2, 2031, 2800]
 TID  3: [0, 3000, 4248, 2, 3011, 3800]
 TID  4: [0, 4000, 4801, 2, 4000, 4800]
 TID  5: [0, 5000, 5801, 2, 5000, 5800]
 TID  6: [0, 6000, 6801, 2, 6000, 6800]
 TID  7: [0, 7000, 7801, 2, 7000, 7800]
 TID  8: [0, 8000, 8801, 2, 8000, 8800]
 TID  9: [0, 9000, 9801, 2, 9000, 9800]
 TID 10: [0, 10000, 10801, 2, 10000, 10800]
SIMULATOR HALTED at PC = 3012
Instructions executed: 988
PS C:\Users\husey\Desktop\pyt\GTU_C312_Python>
```

-D 1:

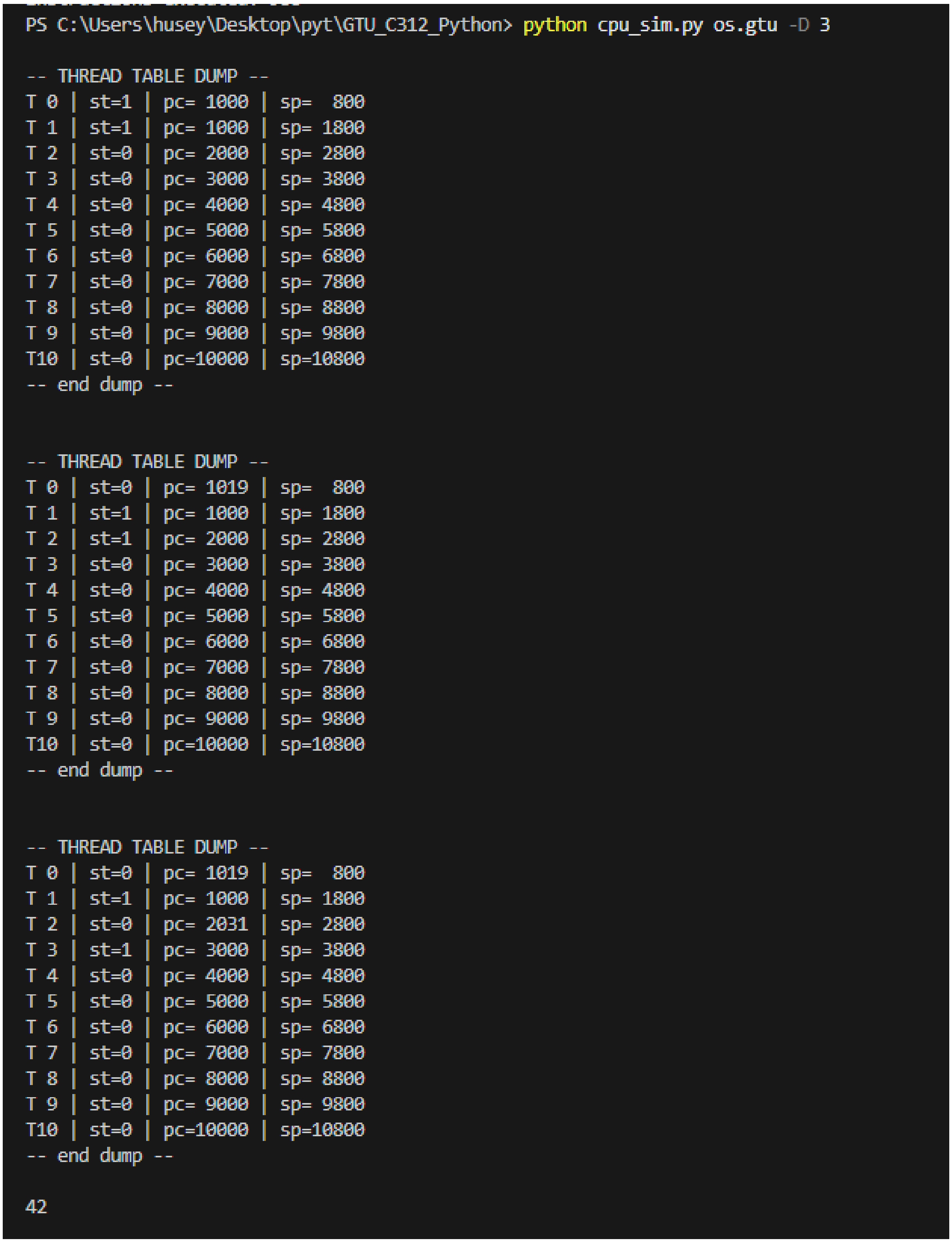


```
-- Thread table [21-86] --
 TID  0: [0, 75, 327, 2, 1019, 800]
 TID  1: [0, 1000, 1800, 1, 1000, 1800]
 TID  2: [0, 2000, 2931, 2, 2031, 2800]
 TID  3: [0, 3000, 4247, 1, 3011, 3800]
 TID  4: [0, 4000, 4801, 2, 4000, 4800]
 TID  5: [0, 5000, 5801, 2, 5000, 5800]
 TID  6: [0, 6000, 6801, 2, 6000, 6800]
 TID  7: [0, 7000, 7801, 2, 7000, 7800]
 TID  8: [0, 8000, 8801, 2, 8000, 8800]
 TID  9: [0, 9000, 9801, 2, 9000, 9800]
 TID 10: [0, 10000, 10801, 2, 10000, 10800]
PC            [  0] = 3012
SP            [  1] = 3800
SYSCALL_RES   [  2] = 0
INSTR_COUNT   [  3] = 988

-- Kernel mem[0-20] --
     0 = 3012
     1 = 3800
     2 = 0
     3 = 988
     4 = 0
     5 = 0
     6 = 0
     7 = 0
     8 = 0
     9 = 0
    10 = 0
    11 = 0
    12 = 0
    13 = 0
    14 = 0
    15 = 0
    16 = 0
    17 = 0
    18 = 0
    19 = 0
    20 = 0

-- Thread table [21-86] --
 TID  0: [0, 75, 327, 2, 1019, 800]
 TID  1: [0, 1000, 1800, 1, 1000, 1800]
 TID  2: [0, 2000, 2931, 2, 2031, 2800]
 TID  3: [0, 3000, 4248, 2, 3011, 3800]
 TID  4: [0, 4000, 4801, 2, 4000, 4800]
 TID  5: [0, 5000, 5801, 2, 5000, 5800]
 TID  6: [0, 6000, 6801, 2, 6000, 6800]
 TID  7: [0, 7000, 7801, 2, 7000, 7800]
 TID  8: [0, 8000, 8801, 2, 8000, 8800]
 TID  9: [0, 9000, 9801, 2, 9000, 9800]
 TID 10: [0, 10000, 10801, 2, 10000, 10800]
SIMULATOR HALTED at PC = 3012
Instructions executed: 988
PS C:\Users\husey\Desktop\pyt\GTU_C312_Python>
```

it is

too much to put in here, but it Works, same for D -2.

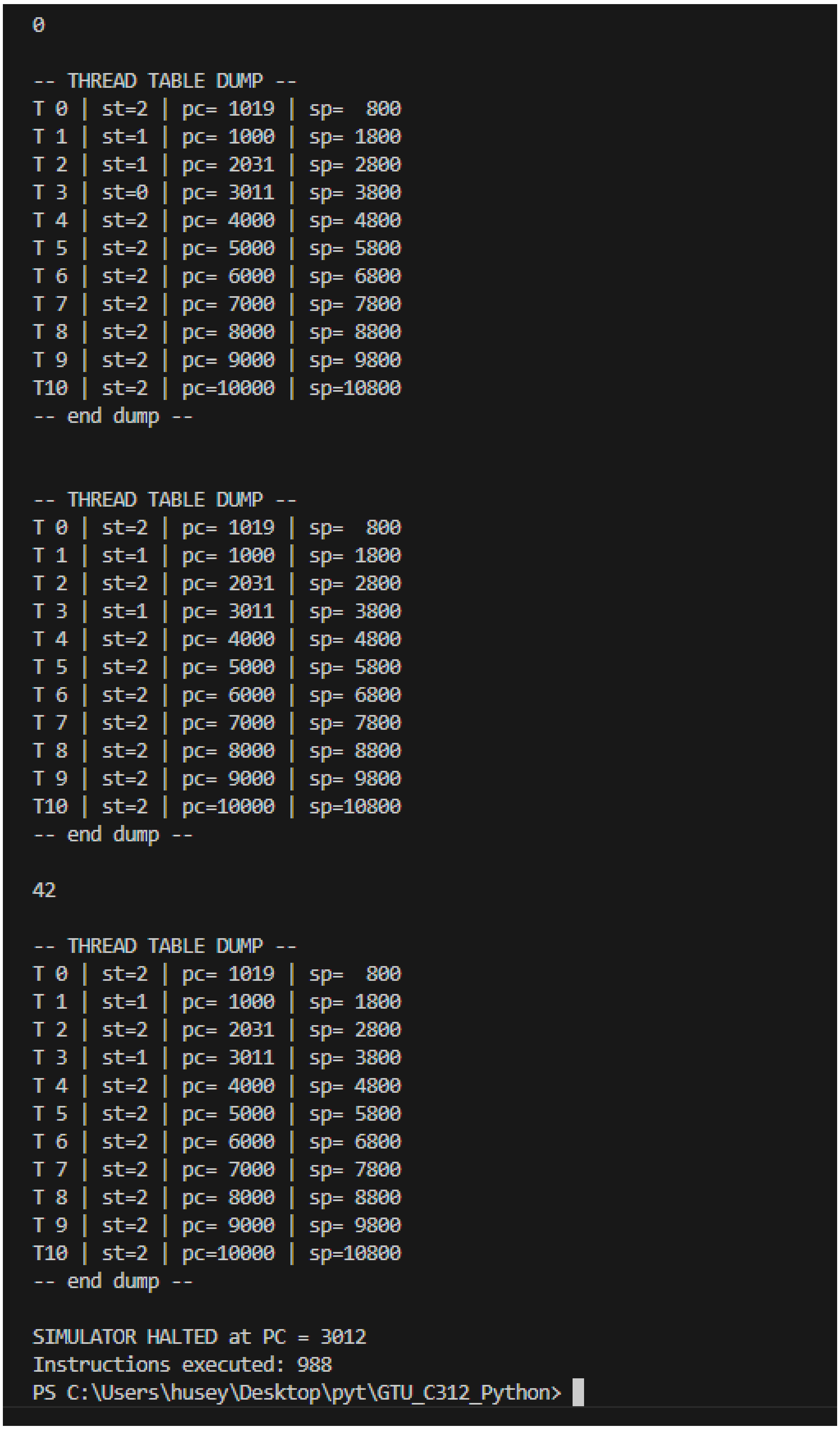
D -3 : start of it



```
PS C:\Users\husey\Desktop\pyt\GTU_C312_Python> python cpu_sim.py os.gtu -D 3

-- THREAD TABLE DUMP --
T 0 | st=1 | pc= 1000 | sp=  800
T 1 | st=1 | pc= 1000 | sp= 1800
T 2 | st=0 | pc= 2000 | sp= 2800
T 3 | st=0 | pc= 3000 | sp= 3800
T 4 | st=0 | pc= 4000 | sp= 4800
T 5 | st=0 | pc= 5000 | sp= 5800
T 6 | st=0 | pc= 6000 | sp= 6800
T 7 | st=0 | pc= 7000 | sp= 7800
T 8 | st=0 | pc= 8000 | sp= 8800
T 9 | st=0 | pc= 9000 | sp= 9800
T10 | st=0 | pc=10000 | sp=10800
-- end dump --


-- THREAD TABLE DUMP --
T 0 | st=0 | pc= 1019 | sp=  800
T 1 | st=1 | pc= 1000 | sp= 1800
T 2 | st=1 | pc= 2000 | sp= 2800
T 3 | st=0 | pc= 3000 | sp= 3800
T 4 | st=0 | pc= 4000 | sp= 4800
T 5 | st=0 | pc= 5000 | sp= 5800
T 6 | st=0 | pc= 6000 | sp= 6800
T 7 | st=0 | pc= 7000 | sp= 7800
T 8 | st=0 | pc= 8000 | sp= 8800
T 9 | st=0 | pc= 9000 | sp= 9800
T10 | st=0 | pc=10000 | sp=10800
-- end dump --


-- THREAD TABLE DUMP --
T 0 | st=0 | pc= 1019 | sp=  800
T 1 | st=1 | pc= 1000 | sp= 1800
T 2 | st=0 | pc= 2031 | sp= 2800
T 3 | st=1 | pc= 3000 | sp= 3800
T 4 | st=0 | pc= 4000 | sp= 4800
T 5 | st=0 | pc= 5000 | sp= 5800
T 6 | st=0 | pc= 6000 | sp= 6800
T 7 | st=0 | pc= 7000 | sp= 7800
T 8 | st=0 | pc= 8000 | sp= 8800
T 9 | st=0 | pc= 9000 | sp= 9800
T10 | st=0 | pc=10000 | sp=10800
-- end dump --

42
```

st 0 ready, st 1 running, st 2 terminated

-D 3 : end of it :



```
0

-- THREAD TABLE DUMP --
T 0 | st=2 | pc= 1019 | sp=  800
T 1 | st=1 | pc= 1000 | sp= 1800
T 2 | st=1 | pc= 2031 | sp= 2800
T 3 | st=0 | pc= 3011 | sp= 3800
T 4 | st=2 | pc= 4000 | sp= 4800
T 5 | st=2 | pc= 5000 | sp= 5800
T 6 | st=2 | pc= 6000 | sp= 6800
T 7 | st=2 | pc= 7000 | sp= 7800
T 8 | st=2 | pc= 8000 | sp= 8800
T 9 | st=2 | pc= 9000 | sp= 9800
T10 | st=2 | pc=10000 | sp=10800
-- end dump --


-- THREAD TABLE DUMP --
T 0 | st=2 | pc= 1019 | sp=  800
T 1 | st=1 | pc= 1000 | sp= 1800
T 2 | st=2 | pc= 2031 | sp= 2800
T 3 | st=1 | pc= 3011 | sp= 3800
T 4 | st=2 | pc= 4000 | sp= 4800
T 5 | st=2 | pc= 5000 | sp= 5800
T 6 | st=2 | pc= 6000 | sp= 6800
T 7 | st=2 | pc= 7000 | sp= 7800
T 8 | st=2 | pc= 8000 | sp= 8800
T 9 | st=2 | pc= 9000 | sp= 9800
T10 | st=2 | pc=10000 | sp=10800
-- end dump --

42

-- THREAD TABLE DUMP --
T 0 | st=2 | pc= 1019 | sp=  800
T 1 | st=1 | pc= 1000 | sp= 1800
T 2 | st=2 | pc= 2031 | sp= 2800
T 3 | st=1 | pc= 3011 | sp= 3800
T 4 | st=2 | pc= 4000 | sp= 4800
T 5 | st=2 | pc= 5000 | sp= 5800
T 6 | st=2 | pc= 6000 | sp= 6800
T 7 | st=2 | pc= 7000 | sp= 7800
T 8 | st=2 | pc= 8000 | sp= 8800
T 9 | st=2 | pc= 9000 | sp= 9800
T10 | st=2 | pc=10000 | sp=10800
-- end dump --

SIMULATOR HALTED at PC = 3012
Instructions executed: 988
PS C:\Users\husey\Desktop\pyt\GTU_C312_Python>
```

Gpt links :

https://chatgpt.com/share/68448517-b988-800b-8d60-c498f7e67870

https://chatgpt.com/share/68448552-3b00-800b-809d-36b64003cfbf

https://chatgpt.com/share/6844859c-40e4-800b-935c-156ad69b9c9e

https://chatgpt.com/share/6844862b-b5b4-800b-b29a-9068edd18eaf

https://chatgpt.com/share/6844865b-bc88-800b-9aca-5dbd65dc54ff