GTU-C312 CPU and Operating System Simulator

This project simulates a CPU based on the Von Neumann architecture and a non-preemptive, multi-threading operating system that runs on this CPU, using a round-robin algorithm. The system is developed in C and can execute its own assembly language.

Project Report Contents

Overall Architecture

The project is fundamentally based on the **Von Neumann architecture**. In this architecture, both program instructions and data are stored in the same memory space. Adhering to this principle, the simulator consists of three main modules:

1. **Memory:** The central unit that holds the entire system state, program codes, and data. It is simulated as a single large array.
2. **Parser:** Reads `.asm` files written in the project-specific assembly language, understands the commands and data within these files, and places them into memory in the appropriate format.
3. **CPU (Central Processing Unit):** The unit that sequentially fetches, decodes, and executes instructions from memory. It controls the flow of the program.

The general workflow is as follows:

1. The **Parser** reads the `.asm` file provided by the user.

2. It parses the commands and data from the file and writes them to the main memory managed by the **Memory** module.
3. The **CPU** begins executing instructions, starting from the address where the Operating System (OS) code is located.
4. The Operating System runs the scheduler to switch between other threads, managing the entire system.

Module Descriptions

Memory Module and Virtual Memory Structure

The memory is the most fundamental component of the project, modeled as a single integer array of `24000` cells (`int memory[24000]`). However, this physical memory acts as a **virtual memory** layer for the threads.

Virtual Memory Abstraction:

Each thread (including the Operating System) does not see the entire memory, but only a 2000-cell block allocated to it. Within this block, addresses start from 0. For example, when a thread wants to access address 50 in its data area, the CPU takes this address 50 and adds it to the thread's physical base address in memory to find the real address. This structure makes each thread feel as if it is working in its own isolated memory space and simplifies memory management.

- Functions like `check_data_address()` and `check_instruction_address()` prevent a thread from accessing outside its allocated virtual memory boundaries. In case of any violation, the system is terminated with an error message.

Memory Map:

A 2000-cell block is allocated for each thread (including the OS).

- **Data Section:** Relative addresses `0` - `999` (1000 cells).
- **Instruction Section:** Relative addresses `1000` - `1999` (1000 cells).

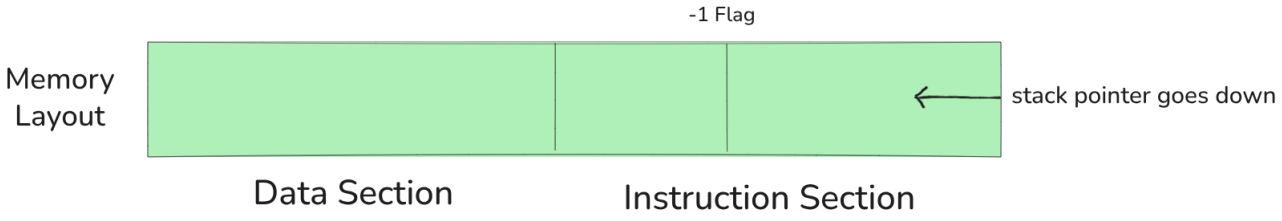
Physical Memory Layout:

- **Registers:** Physical addresses `0-19`. Critical CPU registers like the Program Counter (PC) and Stack Pointer (SP) are stored here.
- **Operating System (Thread 0):**
  - **Data:** Physical addresses `20-999`.
  - **Instructions:** Physical addresses `1000-1999`.
- **User Threads (Threads 1-10):** They are arranged in `2000`-cell blocks after the OS. For example, Thread 1 uses `2000-3999`, Thread 2 uses `4000-5999`, and so on.

Stack Architecture:

The stack for each thread starts from the end of its own instruction section and grows backwards (towards lower addresses).

- **Stack Overflow Protection:** When the parser finishes writing a thread's instructions to memory, it writes a `-1` flag in the memory cell immediately after the last instruction. When stack-writing instructions like `PUSH` or `CALL` encounter this `-1` flag, it is interpreted as a "stack overflow," and the system stops with an error.



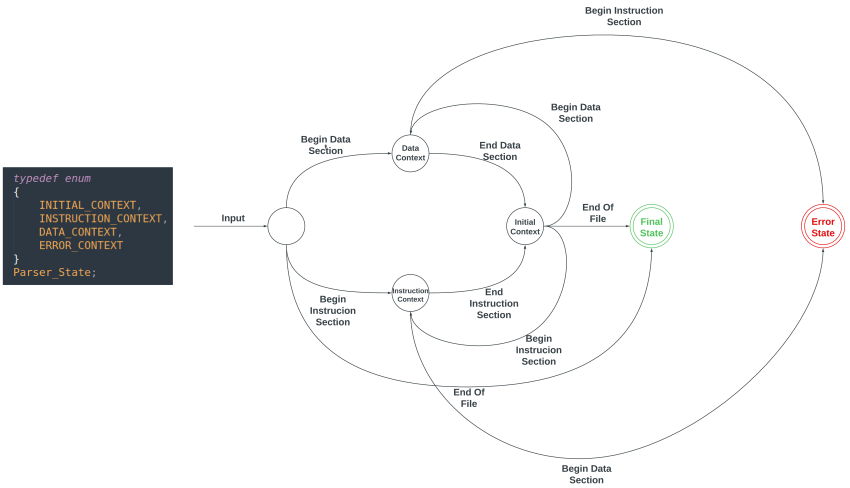
My purpose of using this design is as follows: since I am using virtual memory, I give the user the freedom to use all memory cells between 0 and the memory size defined for him. Therefore, it did not seem appropriate for me as a place to place the stack. Since the number of instructions is likely to be a little smaller, I found it more appropriate to use that part. I thought this would be the best architecture in a memory-constrained environment.

Parser Module

This module, located in `parser.c`, is responsible for reading `.asm` assembly files and loading them into memory.

- It operates with a DFA (Deterministic Finite Automaton) logic, transitioning between labels like `Begin Data Section`, `End Data Section`, `Begin Instruction Section`, and `End Instruction Section`.
- It can process multiple thread definitions (OS, Thread 1, Thread 2...) within the same file.
- It parses lines in the format `<address> <value>` in the data section and `<index> <MNEMONIC> [operand1] [operand2]` in the instruction section.
- It writes this parsed information to the correct physical addresses according to the respective thread's virtual memory map.

DFA structure on which parser is based




```c
while (fgets(line_buffer, ASM_LINE_BUFFER_SIZE, file) != NULL)
{
    line_number++;

    char * clean_line = trim_whitespace(line_buffer);
    if (strlen(clean_line) == 0 || clean_line[0] == '#') {
        continue;
    }

    switch (state)
    {
        case INITIAL_CONTEXT:...

        case INSTRUCTION_CONTEXT:...

        case DATA_CONTEXT:...

        case ERROR_CONTEXT:...
    }
}
```

CPU (Central Processing Unit) Module

The CPU, implemented in `cpu.c`, is the heart of the simulation.

- **Instruction Cycle:** It runs in an infinite loop within the `cpu_execute_instruction` function. In each cycle:
    1. **Fetch:** It reads the instruction and its operands from the address pointed to by the Program Counter (PC) register.

2. **Decode:** It converts the read numeric opcode into the `Opcode` enum structure to understand which instruction to execute.
3. **Execute:** It calls the corresponding instruction's function (`exec_set`, `exec_add`, etc.) to execute the command.

- **Virtual Addressing:** The CPU always works with relative (virtual) addresses when executing instructions. For example, a `SET 50 123` command writes the value `123` to the `50`th cell of the currently active thread's data section. To do this, the CPU calculates the physical address by adding `50` to the base address stored in the `cpu->curr_data_base_for_active_entity` variable.
- **CPU Modes (`KERNEL` and `USER`):** The CPU can operate in two modes. It is in `KERNEL` mode when executing operating system code and in `USER` mode when executing user threads. This provides a foundation for adding authorization and protection mechanisms in the future.

```
typedef struct
{
    Memory * mem;
    CPU_mode mode; /* USER - KERNEL */
    bool is_halted; /* is halted with HLT instruction */
    int curr_thread_id;
    long int curr_data_base_for_active_entity;
    long int curr_instruction_base_for_active_entity;

    /* Pending print values for delayed SYSCALL PRN */
    long int pending_print_values[MAX_PROGRAM_ENTITIES];
    bool has_pending_print[MAX_PROGRAM_ENTITIES];

    /* Debug level for logging */
    int debug_level;
}
CPU;
```

Operating System Structure and Thread Management

The simulator has an operating system whose code resides in `os.asm` and is the first thread to run when the system boots. The OS is responsible for managing other threads.

Thread Structure and States

The state and context of each thread are stored in a **Thread Table** in memory, starting at address `220`. For each thread, there is an 8-cell record in this table:

- Thread ID
- Thread State
- Program Counter (PC)
- Stack Pointer (SP)

- Data Base Address
- Instruction Base Address
- Instruction Count
- Wakeup Instruction Count

A thread can be in one of four different states during its lifecycle:

1. **READY:** Ready to run, waiting for the CPU to be allocated to it.
2. **RUNNING:** The thread that is currently executing on the CPU.
3. **BLOCKED:** Temporarily stopped due to an I/O operation or waiting. For example, a thread that calls `SYSCALL PRN` enters this state for 100 instruction cycles.
4. **TERMINATED:** The thread has completed its execution or has been terminated due to an error.

Scheduler and Round-Robin Algorithm

The scheduler (starting with the `scheduler` label in `os.asm`) is the OS component that decides which thread will run next.

- **Round-Robin:** It uses a simple round-robin algorithm. It iterates through the entire thread table, starting from the thread after the currently running one.
- It selects the first thread it encounters in the `READY` state as the next thread to run.
- If it cannot find a `READY` user thread to run, it always selects the **`idle` thread** (Thread 1) to prevent the system from being idle. The `idle` thread, as soon as it runs, returns control to the OS with a `SYSCALL YIELD` command, causing the scheduler to search for another thread again. This creates an efficient "busy-waiting" loop.

Context Switch Mechanism

Communication between the CPU and the OS is handled by a special protocol, which forms the basis of the context switch.

1. **Request:** After the OS scheduler selects a thread to run, it copies all of the thread's information from the Thread Table (PC, SP, base addresses, etc.) to a special area in memory called the **Mailbox** (addresses `600-607`).
2. Then, it writes the signal `-999` to the `17`th register (`REG_CONTEXT_SWITCH_SIGNAL`).
3. **Detection:** The CPU checks the `17`th register at the beginning of each instruction cycle.
4. If the value is `-999`, it understands that there is a context switch request.
5. **Execution:** The CPU stops its current task, reads the new thread's information from the Mailbox, and updates its own internal registers and state (PC, SP, `curr_thread_id`, base addresses).
6. Finally, it clears the signal by writing `999` (`CTX_SWITCH_DONE`) to the `17`th register and starts executing the new thread's code from where it left off.

System Calls (Syscalls)

User threads use the `SYSCALL` command to request services from the OS.

- When a thread executes a `SYSCALL` command, the CPU automatically switches to `KERNEL` mode and directs the PC to the OS's **Syscall Handler** address (`OS_SYSCALL_HANDLER_ADDR`).
- The OS performs the relevant action based on which syscall was requested (`PRN`, `HLT`, `YIELD`).
    - **`SYSCALL PRN addr`**: Requests to print the value at address `addr`. The OS sets this thread's state to `BLOCKED` and sets its wakeup counter to `100`. The CPU decrements this counter with each instruction cycle. When the counter reaches zero, the thread becomes `READY` again.
    - **`SYSCALL HLT`**: Indicates that the thread wants to terminate its execution. The OS updates the thread's state to `TERMINATED`.
    - **`SYSCALL YIELD`**: The thread voluntarily wants to yield the CPU to another thread. The OS updates the thread's state to `READY` and runs the scheduler.
- After the operation is complete, the OS scheduler selects a new thread, and a context switch is performed to that thread.

Usage

Compilation

The project comes with a `Makefile`. To compile, use the following commands in the project's root directory in your terminal.

To compile all tests and the simulator:

```
make
```

To compile only the simulator:

```
make simulator
```

To clean the files created after compilation:

```
make clean
```

Execution

The simulator is created under the `bin/` directory after compilation. It can take command-line arguments to run.

Basic Usage:

```
./bin/simulator <assembly_file.asm>
```

Example:

```
./bin/simulator programs/os.asm
```

Advanced Options:

- `-D <debug_mode>`: Used to get different levels of debugging information during the simulation.
  - `0`: Default mode. Dumps the non-zero parts of memory only after the simulation is complete.
  - `1`: Dumps the memory after each instruction is executed.
  - `2`: Dumps the memory after each instruction and waits for the user to press `ENTER` to proceed to the next instruction.
  - `3`: Dumps the Thread Table only when a context switch or syscall occurs, showing the thread states.
- `-max <max_instructions>`: Sets the maximum number of instructions the simulator will run. Used to prevent infinite loops. The default value is `200000`.

Example Advanced Usage:

```
./bin/simulator programs/os.asm -D 3 -max 50000
```

Sample Outputs

Now here I will give an example where 3 user threads and OS threads are running (I would like to remind you that thread 1 is always the idle thread):

```
ahmet@ahmet-Inspiron-14-5401:~/DERSLER/4_SINIF/spring/Operating Systems/Operating-System-Simulation$ make simulator
gcc  -g -Wall -Wextra -std=c11 -Iinclude -c simulation/simulator.c -o obj/simulator.o
Compiled: simulation/simulator.c -> obj/simulator.o
gcc  -g -Wall -Wextra -std=c11 -Iinclude -c src/cpu.c -o obj/cpu.o
Compiled: src/cpu.c -> obj/cpu.o
gcc  -g -Wall -Wextra -std=c11 -Iinclude -c src/memory.c -o obj/memory.o
Compiled: src/memory.c -> obj/memory.o
gcc  -g -Wall -Wextra -std=c11 -Iinclude -c src/parser/parser.c -o obj/parser.o
Compiled: src/parser/parser.c -> obj/parser.o
gcc  -g -Wall -Wextra -std=c11 -Iinclude obj/simulator.o obj/cpu.o obj/memory.o obj/parser.o -o bin/simulator
Simulator built successfully!
ahmet@ahmet-Inspiron-14-5401:~/DERSLER/4_SINIF/spring/Operating Systems/Operating-System-Simulation$
```

```
ahmet@ahmet-Inspiron-14-5401:~/DERSLER/4_SINIF/spring/Operating Systems/Operating-System-Simulation$ ./bin/simulator programs/os.asm -D 3
GTU-C312 CPU Simulator
======================

Configuration:
- Program file: programs/os.asm
- Debug mode: 3
- Max instructions: 20000000

Memory initialized (24000 bytes)
Loading program from: programs/os.asm
Program loaded successfully!


=== THREAD TABLE DUMP ===
Thread Table Base Address: 220
Format: [ID] [State] [PC] [SP] [DataBase] [InstrBase] [IC] [WakeupCount]
States: 0=READY, 1=RUNNING, 2=BLOCKED, 3=TERMINATED

Thread 0: [0] [1:RUNNING] [1000] [1999] [20] [1000] [0] [0]
Thread 1: [1] [0:READY] [3000] [3999] [2000] [3000] [0] [0]
Thread 2: [2] [0:READY] [5000] [5999] [4000] [5000] [0] [0]
Thread 3: [3] [0:READY] [7000] [7999] [6000] [7000] [0] [0]
Thread 4: [4] [0:READY] [9000] [9999] [8000] [9000] [0] [0]
Thread 5: [5] [3:TERMINATED] [11000] [11999] [10000] [11000] [0] [0]
Thread 6: [6] [3:TERMINATED] [13000] [13999] [12000] [13000] [0] [0]
Thread 7: [7] [3:TERMINATED] [15000] [15999] [14000] [15000] [0] [0]
Thread 8: [8] [3:TERMINATED] [17000] [17999] [16000] [17000] [0] [0]
Thread 9: [9] [3:TERMINATED] [19000] [19999] [18000] [19000] [0] [0]
Thread 10: [10] [3:TERMINATED] [21000] [21999] [20000] [21000] [0] [0]
=== END THREAD TABLE ===

CPU initialized in KERNEL mode
Starting execution...
```

```
Thread 3: [3] [0:READY] [7000] [7999] [6000] [7000] [0] [0]
Thread 4: [4] [0:READY] [9000] [9999] [8000] [9000] [0] [0]
Thread 5: [5] [3:TERMINATED] [11000] [11999] [10000] [11000] [0] [0]
Thread 6: [6] [3:TERMINATED] [13000] [13999] [12000] [13000] [0] [0]
Thread 7: [7] [3:TERMINATED] [15000] [15999] [14000] [15000] [0] [0]
Thread 8: [8] [3:TERMINATED] [17000] [17999] [16000] [17000] [0] [0]
Thread 9: [9] [3:TERMINATED] [19000] [19999] [18000] [19000] [0] [0]
Thread 10: [10] [3:TERMINATED] [21000] [21999] [20000] [21000] [0] [0]
=== END THREAD TABLE ===

CPU: Context Switch Request DETECTED (Signal: -999) by OS (presumably).

[INSTRUCTION 455] CONTEXT SWITCH DETECTED: Thread 0 -> Thread 2
CPU: Switching to Thread ID: 2, State: READY, PC: 5000, SP: 5999, DB: 4000, IB: 5000, IC: 0, WC: 0

[INSTRUCTION 456] CONTEXT SWITCH DETECTED: Thread 0 -> Thread 2
Executed 500 instructions... (Thread: 2)

[INSTRUCTION 561] SYSCALL DETECTED at PC=5096, Thread=2

[INSTRUCTION 561] SYSCALL DETECTED at PC=5096, Thread=2

=== THREAD TABLE DUMP ===
Thread Table Base Address: 220
Format: [ID] [State] [PC] [SP] [DataBase] [InstrBase] [IC] [WakeupCount]
States: 0=READY, 1=RUNNING, 2=BLOCKED, 3=TERMINATED

Thread 0: [0] [0:READY] [1000] [1999] [20] [1000] [0] [0]
Thread 1: [1] [0:READY] [3003] [3999] [2000] [3000] [0] [0]
Thread 2: [2] [1:RUNNING] [5000] [5999] [4000] [5000] [0] [0]
Thread 3: [3] [0:READY] [7000] [7999] [6000] [7000] [0] [0]
Thread 4: [4] [0:READY] [9000] [9999] [8000] [9000] [0] [0]
Thread 5: [5] [3:TERMINATED] [11000] [11999] [10000] [11000] [0] [0]
Thread 6: [6] [3:TERMINATED] [13000] [13999] [12000] [13000] [0] [0]
Thread 7: [7] [3:TERMINATED] [15000] [15999] [14000] [15000] [0] [0]
Thread 8: [8] [3:TERMINATED] [17000] [17999] [16000] [17000] [0] [0]
Thread 9: [9] [3:TERMINATED] [19000] [19999] [18000] [19000] [0] [0]
Thread 10: [10] [3:TERMINATED] [21000] [21999] [20000] [21000] [0] [0]
=== END THREAD TABLE ===


[INSTRUCTION 562] THREAD TABLE AFTER SYSCALL:

=== THREAD TABLE DUMP ===
Thread Table Base Address: 220
Format: [ID] [State] [PC] [SP] [DataBase] [InstrBase] [IC] [WakeupCount]
States: 0=READY, 1=RUNNING, 2=BLOCKED, 3=TERMINATED

Thread 0: [0] [0:READY] [1000] [1999] [20] [1000] [0] [0]
Thread 1: [1] [0:READY] [3003] [3999] [2000] [3000] [0] [0]
[INSTRUCTION 1016] THREAD TABLE AFTER SYSCALL:

=== THREAD TABLE DUMP ===
Thread Table Base Address: 220
Format: [ID] [State] [PC] [SP] [DataBase] [InstrBase] [IC] [WakeupCount]
States: 0=READY, 1=RUNNING, 2=BLOCKED, 3=TERMINATED

Thread 0: [0] [0:READY] [1000] [1999] [20] [1000] [0] [0]
Thread 1: [1] [0:READY] [3003] [3999] [2000] [3000] [0] [0]
Thread 2: [2] [3:TERMINATED] [5099] [5999] [4000] [5000] [1000] [0]
Thread 3: [3] [1:RUNNING] [7000] [7999] [6000] [7000] [0] [0]
Thread 4: [4] [0:READY] [9000] [9999] [8000] [9000] [0] [0]
Thread 5: [5] [3:TERMINATED] [11000] [11999] [10000] [11000] [0] [0]
Thread 6: [6] [3:TERMINATED] [13000] [13999] [12000] [13000] [0] [0]
Thread 7: [7] [3:TERMINATED] [15000] [15999] [14000] [15000] [0] [0]
Thread 8: [8] [3:TERMINATED] [17000] [17999] [16000] [17000] [0] [0]
Thread 9: [9] [3:TERMINATED] [19000] [19999] [18000] [19000] [0] [0]
Thread 10: [10] [3:TERMINATED] [21000] [21999] [20000] [21000] [0] [0]
=== END THREAD TABLE ===


[INSTRUCTION 1016] THREAD TABLE AFTER SYSCALL:

[INSTRUCTION 1016] CONTEXT SWITCH DETECTED: Thread 3 -> Thread 0
Executed 1100 instructions... (Thread: 0)
Executed 1200 instructions... (Thread: 0)
Executed 1300 instructions... (Thread: 0)
Executed 1400 instructions... (Thread: 0)
Thread 3 unblocked (wakeup reached 0)

=== THREAD TABLE DUMP ===
Thread Table Base Address: 220
Format: [ID] [State] [PC] [SP] [DataBase] [InstrBase] [IC] [WakeupCount]
States: 0=READY, 1=RUNNING, 2=BLOCKED, 3=TERMINATED

Thread 0: [0] [0:READY] [1000] [1999] [20] [1000] [0] [0]
Thread 1: [1] [0:READY] [3003] [3999] [2000] [3000] [0] [0]
Thread 2: [2] [3:TERMINATED] [5099] [5999] [4000] [5000] [1000] [0]
Thread 3: [3] [0:READY] [7060] [7999] [6000] [7000] [3003] [0]
Thread 4: [4] [1:RUNNING] [9000] [9999] [8000] [9000] [0] [0]
Thread 5: [5] [3:TERMINATED] [11000] [11999] [10000] [11000] [0] [0]
Thread 6: [6] [3:TERMINATED] [13000] [13999] [12000] [13000] [0] [0]
Thread 7: [7] [3:TERMINATED] [15000] [15999] [14000] [15000] [0] [0]
Thread 8: [8] [3:TERMINATED] [17000] [17999] [16000] [17000] [0] [0]
Thread 9: [9] [3:TERMINATED] [19000] [19999] [18000] [19000] [0] [0]
Thread 10: [10] [3:TERMINATED] [21000] [21999] [20000] [21000] [0] [0]
=== END THREAD TABLE ===

CPU: Context Switch Request DETECTED (Signal: -999) by OS (presumably).
```

```
[INSTRUCTION 2347] THREAD TABLE AFTER SYSCALL:

[INSTRUCTION 2347] CONTEXT SWITCH DETECTED: Thread 1 -> Thread 0
Thread 4 unblocked (wakeup reached 0)
Executed 2400 instructions... (Thread: 0)
Executed 2500 instructions... (Thread: 0)
Executed 2600 instructions... (Thread: 0)
Executed 2700 instructions... (Thread: 0)

=== THREAD TABLE DUMP ===
Thread Table Base Address: 220
Format: [ID] [State] [PC] [SP] [DataBase] [InstrBase] [IC] [WakeupCount]
States: 0=READY, 1=RUNNING, 2=BLOCKED, 3=TERMINATED

Thread 0: [0] [0:READY] [1000] [1999] [20] [1000] [0] [0]
Thread 1: [1] [0:READY] [3003] [3999] [2000] [3000] [0] [0]
Thread 2: [2] [3:TERMINATED] [5099] [5999] [4000] [5000] [1000] [0]
Thread 3: [3] [1:RUNNING] [7060] [7999] [6000] [7000] [3003] [0]
Thread 4: [4] [0:READY] [9030] [9999] [8000] [9000] [5099] [0]
Thread 5: [5] [3:TERMINATED] [11000] [11999] [10000] [11000] [0] [0]
Thread 6: [6] [3:TERMINATED] [13000] [13999] [12000] [13000] [0] [0]
Thread 7: [7] [3:TERMINATED] [15000] [15999] [14000] [15000] [0] [0]
Thread 8: [8] [3:TERMINATED] [17000] [17999] [16000] [17000] [0] [0]
Thread 9: [9] [3:TERMINATED] [19000] [19999] [18000] [19000] [0] [0]
Thread 10: [10] [3:TERMINATED] [21000] [21999] [20000] [21000] [0] [0]
=== END THREAD TABLE ===

CPU: Context Switch Request DETECTED (Signal: -999) by OS (presumably).

[INSTRUCTION 2738] CONTEXT SWITCH DETECTED: Thread 0 -> Thread 3
CPU: Switching to Thread ID: 3, State: READY, PC: 7060, SP: 7999, DB: 6000, IB: 7000, IC: 3003, WC: 0

Requested SYSCALL PRN output (Thread 3): 5

[INSTRUCTION 2738] SYSCALL DETECTED at PC=7060, Thread=3

=== THREAD TABLE DUMP ===
Thread Table Base Address: 220
Format: [ID] [State] [PC] [SP] [DataBase] [InstrBase] [IC] [WakeupCount]
States: 0=READY, 1=RUNNING, 2=BLOCKED, 3=TERMINATED

Thread 0: [0] [0:READY] [1000] [1999] [20] [1000] [0] [0]
Thread 1: [1] [0:READY] [3003] [3999] [2000] [3000] [0] [0]
Thread 2: [2] [3:TERMINATED] [5099] [5999] [4000] [5000] [1000] [0]
Thread 3: [3] [1:RUNNING] [7060] [7999] [6000] [7000] [3003] [0]
Thread 4: [4] [0:READY] [9030] [9999] [8000] [9000] [5099] [0]
Thread 5: [5] [3:TERMINATED] [11000] [11999] [10000] [11000] [0] [0]
Thread 6: [6] [3:TERMINATED] [13000] [13999] [12000] [13000] [0] [0]
```

```
Executed 3400 instructions... (Thread: 0)
Executed 3500 instructions... (Thread: 0)
Executed 3600 instructions... (Thread: 0)
Executed 3700 instructions... (Thread: 0)
Executed 3800 instructions... (Thread: 0)
Executed 3900 instructions... (Thread: 0)

=== ALL USER THREADS COMPLETED ===
All user threads have terminated. Only idle thread (Thread 1) is running.
Idle thread will continue indefinitely. Stopping simulation here.
Instructions executed: 3955

=== EXECUTION SUMMARY ===
Total instructions executed: 3955
Context switches: 8
System calls: 4
CPU halted: No
SUCCESS: All user threads completed successfully!
Only idle thread remains active (as expected).

=== FINAL THREAD TABLE ===

=== THREAD TABLE DUMP ===
Thread Table Base Address: 220
Format: [ID] [State] [PC] [SP] [DataBase] [InstrBase] [IC] [WakeupCount]
States: 0=READY, 1=RUNNING, 2=BLOCKED, 3=TERMINATED

Thread 0: [0] [1:RUNNING] [1000] [1999] [20] [1000] [0] [0]
Thread 1: [1] [0:READY] [3003] [3999] [2000] [3000] [0] [0]
Thread 2: [2] [3:TERMINATED] [5099] [5999] [4000] [5000] [1000] [0]
Thread 3: [3] [3:TERMINATED] [7063] [7999] [6000] [7000] [3003] [0]
Thread 4: [4] [3:TERMINATED] [9030] [9999] [8000] [9000] [5099] [0]
Thread 5: [5] [3:TERMINATED] [11000] [11999] [10000] [11000] [0] [0]
Thread 6: [6] [3:TERMINATED] [13000] [13999] [12000] [13000] [0] [0]
Thread 7: [7] [3:TERMINATED] [15000] [15999] [14000] [15000] [0] [0]
Thread 8: [8] [3:TERMINATED] [17000] [17999] [16000] [17000] [0] [0]
Thread 9: [9] [3:TERMINATED] [19000] [19999] [18000] [19000] [0] [0]
Thread 10: [10] [3:TERMINATED] [21000] [21999] [20000] [21000] [0] [0]
=== END THREAD TABLE ===


=== FINAL CPU STATE ===
CPU Registers for Thread 0 (Mode: KERNEL):
REG_PC: 1393
REG_SP: 1997
REG_INSTR_COUNT: 3955

Simulation completed successfully!
ahmet@ahmet-Inspiron-14-5401:~/DERSLER/4_SINIF/spring/Operating Systems/Operating-System-Simulation$
```

The output is very long because threads use a lot of loops (especially OS), so I added some important parts for understanding.

Now let's do a test with 5 user threads in the same way:

```
gcc  -g -Wall -Wextra -std=c11 -Iinclude obj/simulator.o obj/cpu.o obj/memory.o obj/parser.o -o bin/simulator
Simulator built successfully!
ahmet@ahmet-Inspiron-14-5401:~/DERSLER/4_SINIF/spring/Operating Systems/Operating-System-Simulation$ ./bin/simulator programs/os.asm -D 3
GTU-C312 CPU Simulator
======================

Configuration:
- Program file: programs/os.asm
- Debug mode: 3
- Max instructions: 20000000

Memory initialized (24000 bytes)
Loading program from: programs/os.asm
Program loaded successfully!


=== THREAD TABLE DUMP ===
Thread Table Base Address: 220
Format: [ID] [State] [PC] [SP] [DataBase] [InstrBase] [IC] [WakeupCount]
States: 0=READY, 1=RUNNING, 2=BLOCKED, 3=TERMINATED

Thread 0: [0] [1:RUNNING] [1000] [1999] [20] [1000] [0] [0]
Thread 1: [1] [0:READY] [3000] [3999] [2000] [3000] [0] [0]
Thread 2: [2] [0:READY] [5000] [5999] [4000] [5000] [0] [0]
Thread 3: [3] [0:READY] [7000] [7999] [6000] [7000] [0] [0]
Thread 4: [4] [0:READY] [9000] [9999] [8000] [9000] [0] [0]
Thread 5: [5] [0:READY] [11000] [11999] [10000] [11000] [0] [0]
Thread 6: [6] [0:READY] [13000] [13999] [12000] [13000] [0] [0]
Thread 7: [7] [3:TERMINATED] [15000] [15999] [14000] [15000] [0] [0]
Thread 8: [8] [3:TERMINATED] [17000] [17999] [16000] [17000] [0] [0]
Thread 9: [9] [3:TERMINATED] [19000] [19999] [18000] [19000] [0] [0]
Thread 10: [10] [3:TERMINATED] [21000] [21999] [20000] [21000] [0] [0]
=== END THREAD TABLE ===

CPU initialized in KERNEL mode
Starting execution...

Executed 100 instructions... (Thread: 0)

=== THREAD TABLE DUMP ===
Thread Table Base Address: 220
Format: [ID] [State] [PC] [SP] [DataBase] [InstrBase] [IC] [WakeupCount]
States: 0=READY, 1=RUNNING, 2=BLOCKED, 3=TERMINATED

Thread 0: [0] [0:READY] [1000] [1999] [20] [1000] [0] [0]
Thread 1: [1] [1:RUNNING] [3000] [3999] [2000] [3000] [0] [0]
Thread 2: [2] [0:READY] [5000] [5999] [4000] [5000] [0] [0]
Thread 3: [3] [0:READY] [7000] [7999] [6000] [7000] [0] [0]
Thread 4: [4] [0:READY] [9000] [9999] [8000] [9000] [0] [0]
```

```
Starting execution...

Executed 100 instructions... (Thread: 0)

=== THREAD TABLE DUMP ===
Thread Table Base Address: 220
Format: [ID] [State] [PC] [SP] [DataBase] [InstrBase] [IC] [WakeupCount]
States: 0=READY, 1=RUNNING, 2=BLOCKED, 3=TERMINATED

Thread 0: [0] [0:READY] [1000] [1999] [20] [1000] [0] [0]
Thread 1: [1] [1:RUNNING] [3000] [3999] [2000] [3000] [0] [0]
Thread 2: [2] [0:READY] [5000] [5999] [4000] [5000] [0] [0]
Thread 3: [3] [0:READY] [7000] [7999] [6000] [7000] [0] [0]
Thread 4: [4] [0:READY] [9000] [9999] [8000] [9000] [0] [0]
Thread 5: [5] [0:READY] [11000] [11999] [10000] [11000] [0] [0]
Thread 6: [6] [0:READY] [13000] [13999] [12000] [13000] [0] [0]
Thread 7: [7] [3:TERMINATED] [15000] [15999] [14000] [15000] [0] [0]
Thread 8: [8] [3:TERMINATED] [17000] [17999] [16000] [17000] [0] [0]
Thread 9: [9] [3:TERMINATED] [19000] [19999] [18000] [19000] [0] [0]
Thread 10: [10] [3:TERMINATED] [21000] [21999] [20000] [21000] [0] [0]
=== END THREAD TABLE ===

CPU: Context Switch Request DETECTED (Signal: -999) by OS (presumably).

[INSTRUCTION 126] CONTEXT SWITCH DETECTED: Thread 0 -> Thread 1
CPU: Switching to Thread ID: 1, State: READY, PC: 3000, SP: 3999, DB: 2000, IB: 3000, IC: 0, WC: 0

[INSTRUCTION 126] SYSCALL DETECTED at PC=3000, Thread=1

=== THREAD TABLE DUMP ===
Thread Table Base Address: 220
Format: [ID] [State] [PC] [SP] [DataBase] [InstrBase] [IC] [WakeupCount]
States: 0=READY, 1=RUNNING, 2=BLOCKED, 3=TERMINATED

Thread 0: [0] [0:READY] [1000] [1999] [20] [1000] [0] [0]
Thread 1: [1] [1:RUNNING] [3000] [3999] [2000] [3000] [0] [0]
Thread 2: [2] [0:READY] [5000] [5999] [4000] [5000] [0] [0]
Thread 3: [3] [0:READY] [7000] [7999] [6000] [7000] [0] [0]
Thread 4: [4] [0:READY] [9000] [9999] [8000] [9000] [0] [0]
Thread 5: [5] [0:READY] [11000] [11999] [10000] [11000] [0] [0]
Thread 6: [6] [0:READY] [13000] [13999] [12000] [13000] [0] [0]
Thread 7: [7] [3:TERMINATED] [15000] [15999] [14000] [15000] [0] [0]
Thread 8: [8] [3:TERMINATED] [17000] [17999] [16000] [17000] [0] [0]
Thread 9: [9] [3:TERMINATED] [19000] [19999] [18000] [19000] [0] [0]
Thread 10: [10] [3:TERMINATED] [21000] [21999] [20000] [21000] [0] [0]
=== END THREAD TABLE ===


[INSTRUCTION 127] THREAD TABLE AFTER SYSCALL:
```

```
=== THREAD TABLE DUMP ===
Thread Table Base Address: 220
Format: [ID] [State] [PC] [SP] [DataBase] [InstrBase] [IC] [WakeupCount]
States: 0=READY, 1=RUNNING, 2=BLOCKED, 3=TERMINATED

Thread 0: [0] [0:READY] [1000] [1999] [20] [1000] [0] [0]
Thread 1: [1] [0:READY] [3003] [3999] [2000] [3000] [0] [0]
Thread 2: [2] [3:TERMINATED] [5099] [5999] [4000] [5000] [1000] [0]
Thread 3: [3] [1:RUNNING] [7000] [7999] [6000] [7000] [0] [0]
Thread 4: [4] [0:READY] [9000] [9999] [8000] [9000] [0] [0]
Thread 5: [5] [0:READY] [11000] [11999] [10000] [11000] [0] [0]
Thread 6: [6] [0:READY] [13000] [13999] [12000] [13000] [0] [0]
Thread 7: [7] [3:TERMINATED] [15000] [15999] [14000] [15000] [0] [0]
Thread 8: [8] [3:TERMINATED] [17000] [17999] [16000] [17000] [0] [0]
Thread 9: [9] [3:TERMINATED] [19000] [19999] [18000] [19000] [0] [0]
Thread 10: [10] [3:TERMINATED] [21000] [21999] [20000] [21000] [0] [0]
=== END THREAD TABLE ===


[INSTRUCTION 1016] THREAD TABLE AFTER SYSCALL:

=== THREAD TABLE DUMP ===
Thread Table Base Address: 220
Format: [ID] [State] [PC] [SP] [DataBase] [InstrBase] [IC] [WakeupCount]
States: 0=READY, 1=RUNNING, 2=BLOCKED, 3=TERMINATED

Thread 0: [0] [0:READY] [1000] [1999] [20] [1000] [0] [0]
Thread 1: [1] [0:READY] [3003] [3999] [2000] [3000] [0] [0]
Thread 2: [2] [3:TERMINATED] [5099] [5999] [4000] [5000] [1000] [0]
Thread 3: [3] [1:RUNNING] [7000] [7999] [6000] [7000] [0] [0]
Thread 4: [4] [0:READY] [9000] [9999] [8000] [9000] [0] [0]
Thread 5: [5] [0:READY] [11000] [11999] [10000] [11000] [0] [0]
Thread 6: [6] [0:READY] [13000] [13999] [12000] [13000] [0] [0]
Thread 7: [7] [3:TERMINATED] [15000] [15999] [14000] [15000] [0] [0]
Thread 8: [8] [3:TERMINATED] [17000] [17999] [16000] [17000] [0] [0]
Thread 9: [9] [3:TERMINATED] [19000] [19999] [18000] [19000] [0] [0]
Thread 10: [10] [3:TERMINATED] [21000] [21999] [20000] [21000] [0] [0]
=== END THREAD TABLE ===


[INSTRUCTION 1016] THREAD TABLE AFTER SYSCALL:

[INSTRUCTION 1016] CONTEXT SWITCH DETECTED: Thread 3 -> Thread 0
Executed 1100 instructions... (Thread: 0)
Executed 1200 instructions... (Thread: 0)
Executed 1300 instructions... (Thread: 0)
Executed 1400 instructions... (Thread: 0)
Thread 3 unblocked (wakeup reached 0)
```

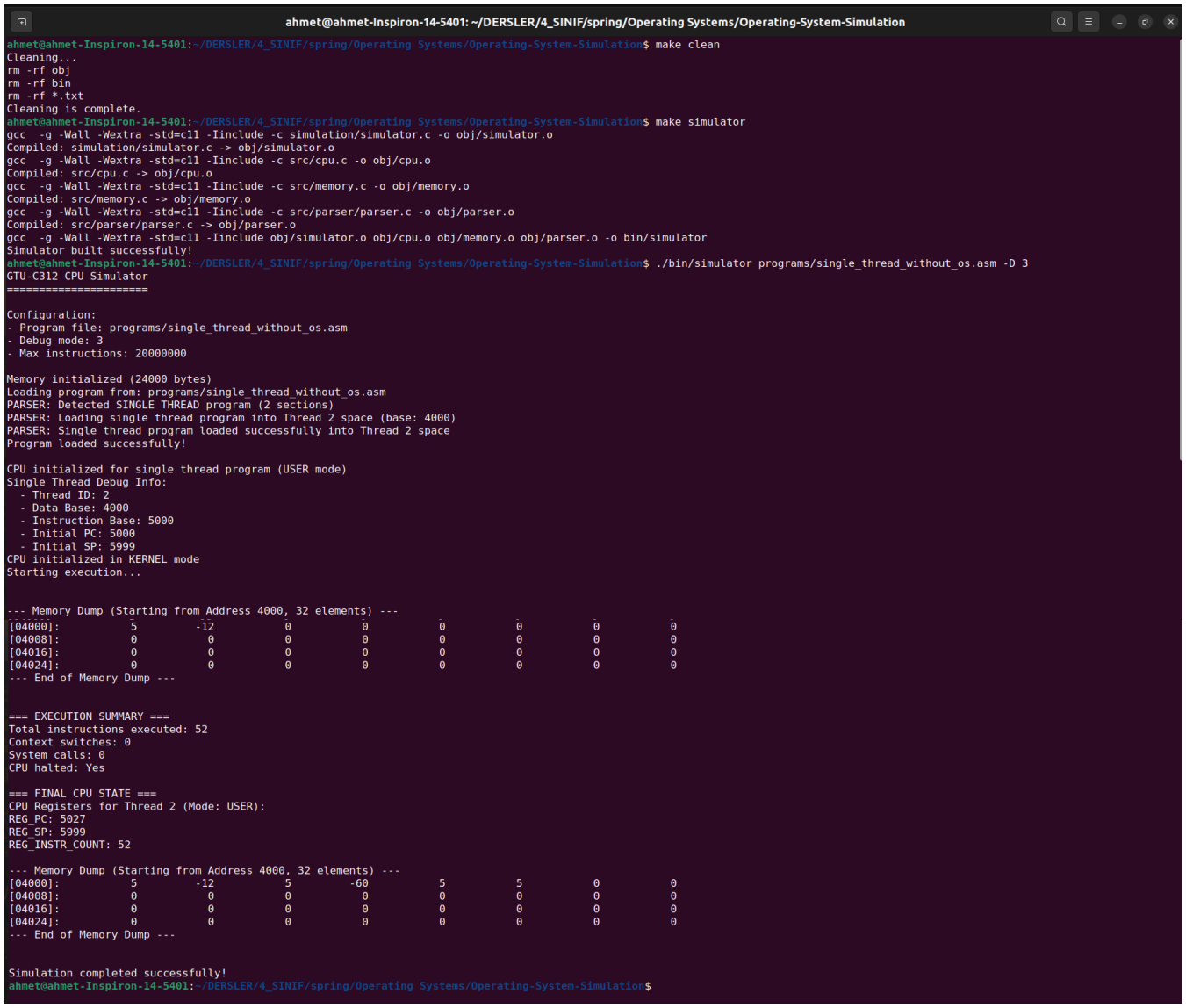
```
Thread 10: [10] [3:TERMINATED] [21000] [21999] [20000] [21000] [0] [0]
=== END THREAD TABLE ===


[INSTRUCTION 7149] THREAD TABLE AFTER SYSCALL:

=== THREAD TABLE DUMP ===
Thread Table Base Address: 220
Format: [ID] [State] [PC] [SP] [DataBase] [InstrBase] [IC] [WakeupCount]
States: 0=READY, 1=RUNNING, 2=BLOCKED, 3=TERMINATED

Thread 0: [0] [0:READY] [1000] [1999] [20] [1000] [0] [0]
Thread 1: [1] [1:RUNNING] [3003] [3999] [2000] [3000] [0] [0]
Thread 2: [2] [3:TERMINATED] [5099] [5999] [4000] [5000] [1000] [0]
Thread 3: [3] [3:TERMINATED] [7063] [7999] [6000] [7000] [3003] [0]
Thread 4: [4] [2:BLOCKED] [9036] [9999] [8000] [9000] [5099] [14]
Thread 5: [5] [3:TERMINATED] [11033] [11999] [10000] [11000] [7063] [0]
Thread 6: [6] [3:TERMINATED] [13033] [13999] [12000] [13000] [9033] [0]
Thread 7: [7] [3:TERMINATED] [15000] [15999] [14000] [15000] [0] [0]
Thread 8: [8] [3:TERMINATED] [17000] [17999] [16000] [17000] [0] [0]
Thread 9: [9] [3:TERMINATED] [19000] [19999] [18000] [19000] [0] [0]
Thread 10: [10] [3:TERMINATED] [21000] [21999] [20000] [21000] [0] [0]
=== END THREAD TABLE ===


[INSTRUCTION 7149] THREAD TABLE AFTER SYSCALL:

[INSTRUCTION 7149] CONTEXT SWITCH DETECTED: Thread 1 -> Thread 0
Thread 4 unblocked (wakeup reached 0)
Executed 7200 instructions... (Thread: 0)
Executed 7300 instructions... (Thread: 0)
Executed 7400 instructions... (Thread: 0)
Executed 7500 instructions... (Thread: 0)
Executed 7600 instructions... (Thread: 0)

=== THREAD TABLE DUMP ===
Thread Table Base Address: 220
Format: [ID] [State] [PC] [SP] [DataBase] [InstrBase] [IC] [WakeupCount]
States: 0=READY, 1=RUNNING, 2=BLOCKED, 3=TERMINATED

Thread 0: [0] [0:READY] [1000] [1999] [20] [1000] [0] [0]
Thread 1: [1] [0:READY] [3003] [3999] [2000] [3000] [0] [0]
Thread 2: [2] [3:TERMINATED] [5099] [5999] [4000] [5000] [1000] [0]
Thread 3: [3] [3:TERMINATED] [7063] [7999] [6000] [7000] [3003] [0]
Thread 4: [4] [1:RUNNING] [9036] [9999] [8000] [9000] [5099] [0]
Thread 5: [5] [3:TERMINATED] [11033] [11999] [10000] [11000] [7063] [0]
Thread 6: [6] [3:TERMINATED] [13033] [13999] [12000] [13000] [9033] [0]
Thread 7: [7] [3:TERMINATED] [15000] [15999] [14000] [15000] [0] [0]
Thread 8: [8] [3:TERMINATED] [17000] [17999] [16000] [17000] [0] [0]
```

In the 4th thread, I made syscall calls one after the other so that we can see the output more clearly.

```
Begin Instruction Section
# Multiplication alg.
# for (i = 0; i < operand_1; i++) {
#   sum_buffer += operand_2;
# }
# loop strat:

0   CPY  0 4        # operand_1 -> 4
1   CPY  2 5        # i -> 5
2   CPY  4 100      # opeand_1: temp
3   CPY  5 101      # i: temp
4   SUBI 100 101    # N - i
5   JIF  101 9      # döngü bitti bitir: HLT
6   ADDI 3 1        # M[3] += M[1] : sum_buffer += operand_2
7   ADD  2 1        # i++, next iteration
8   JIF  99 0       # unconditionl jump
9   SYSCALL PRN 3   # printing result
10  SYSCALL PRN 1
11  SYSCALL PRN 0
12  SYSCALL YIELD
13  SYSCALL HLT
```

Single Thread Sample

Here I will show a scenario where there is no OS thread and only one user thread, since there are registers between 0-2000 and we had to create a special architecture for the OS, we start using memory starting from 2000 since there is already enough memory in single threads.



Program:

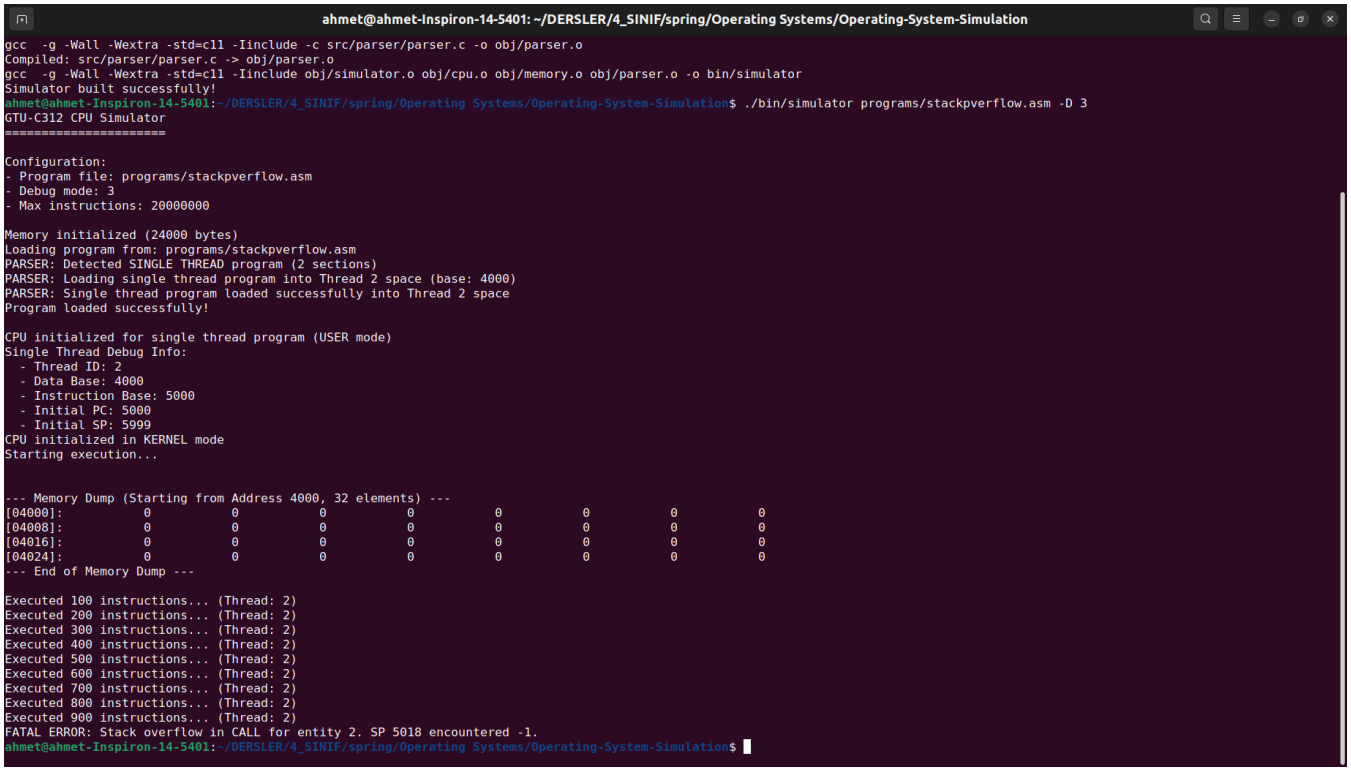
```
# programs/single_thread_without_os.asm #

# Multiplication Implementation
Begin Data Section
0   5       # mult operand 1
1   -12     # mult operand 2
2   0       # counter i for loop
3   0       # summ_buffer: for, M[3] += M[1]
99  -1      # constant -1 for unconditional jumps
End Data Section

Begin Instruction Section
# Multiplication alg.
# for (i = 0; i < operand_1; i++) {
#    sum_buffer += operand_2;
# }
# loop strat:
```

```
0   CPY  0 4        # operand_1 -> 4
1   CPY  2 5        # i -> 5
2   CPY  4 100      # opeand_1: temp
3   CPY  5 101      # i: temp
4   SUBI 100 101    # N - i
5   JIF  101 9      # döngü bitti bitir: HLT
6   ADDI 3 1        # M[3] += M[1] : sum_buffer += operand_2
7   ADD  2 1        # i++, next iteration
8   JIF  99 0       # unconditionl jump
9   HLT

End Instruction Section
```

Error Cases

Here I am adding some error situations that can be easily displayed with Assembly files. This program will give a stackoverflow error because there are multiple function calls and no routine that returns `RET`:

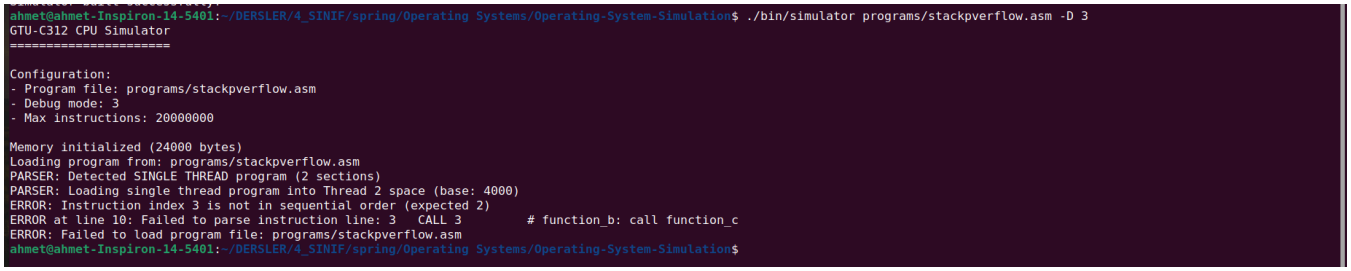
```
# Stack Overflow Demo
Begin Data Section
0   0       # dummy data
End Data Section

Begin Instruction Section
# Multiple nested function calls
0   CALL 1          # call function_a
1   CALL 2          # function_a: call function_b
2   CALL 3          # function_b: call function_c
3   CALL 4          # function_c: call function_d
4   CALL 0          # function_d: call main again (creates loop)
5   HLT             # never reached
End Instruction Section
```



Here in our memory structure, the stack grows towards the instruction section at the end of the instruction section. That's why we put a separator -1 flag in between so that we can notice it.

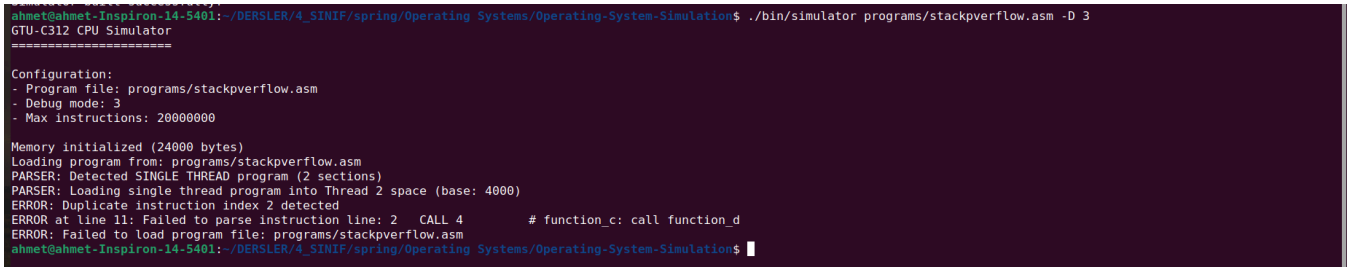
Parser errors: some syntax rules

```
# Stack Overflow Demo
Begin Data Section
0    0        # dummy data
End Data Section

Begin Instruction Section
# Multiple nested function calls
0    CALL 1          # call function_a
1    CALL 2          # function_a: call function_b
3    CALL 3          # function_b: call function_c
2    CALL 4          # function_c: call function_d
4    CALL 0          # function_d: call main again (creates loop)
5    HLT             # never reached
End Instruction Section
```



or,

```
# Stack Overflow Demo
Begin Data Section
0    0        # dummy data
End Data Section

Begin Instruction Section
# Multiple nested function calls
0    CALL 1          # call function_a
1    CALL 2          # function_a: call function_b
2    CALL 3          # function_b: call function_c
2    CALL 4          # function_c: call function_d
4    CALL 0          # function_d: call main again (creates loop)
5    HLT             # never reached
End Instruction Section
```



AI Chats:

- https://claude.ai/public/artifacts/124f8d9b-05dd-4d0d-9a20-d93e59489356
- https://claude.ai/share/55c93d72-af7d-48ad-ab72-d88cc9160707
- https://grok.com/share/bGVnYWN5_da6fac8d-825a-47ee-84f0-c361f07fec73
- https://grok.com/share/bGVnYWN5_96fc579d-5377-42a0-b511-8626e34649ec
- https://claude.ai/share/56220e0e-4a19-47b5-bb84-6005516e908c
- https://claude.ai/share/d92894fd-02c3-4473-90e6-fc06e8dd7f6c
- https://claude.ai/share/e9844582-265a-43b9-b753-c6104f043aea

I've used Gemini in many places and there is no chat sharing feature there so I won't put it here.

- Ahmet Özdemir
- 200104004062