Experiment No.6
Implement process scheduling algorithms FCFS and SJF using CPU-OS Simulator.
Date of Performance:
Date of Submission:

Aim: To study and implement process scheduling algorithms FCFS and SJF using CPU-OS Simulator.

Objective: Its main objective is to increase CPU utilization and hence the throughput of the system by keeping the CPU as busy as possible.

Theory:

A Process Scheduler schedules different processes to be assigned to the CPU based on particular scheduling algorithms.

These algorithms are either non-preemptive or preemptive. Non-preemptive algorithms are designed so that once a process enters the running state, it cannot be preempted until it completes its allotted time, whereas the preemptive scheduling is based on priority where a scheduler may preempt a low priority running process anytime when a high priority process enters into a ready state.

First Come First Serve (FCFS)

Jobs are executed on first come, first serve basis. It is a non-preemptive, pre-emptive scheduling algorithm. Easy to understand and implement. Its implementation is based on FIFO queue. Poor in performance as average wait time is high.

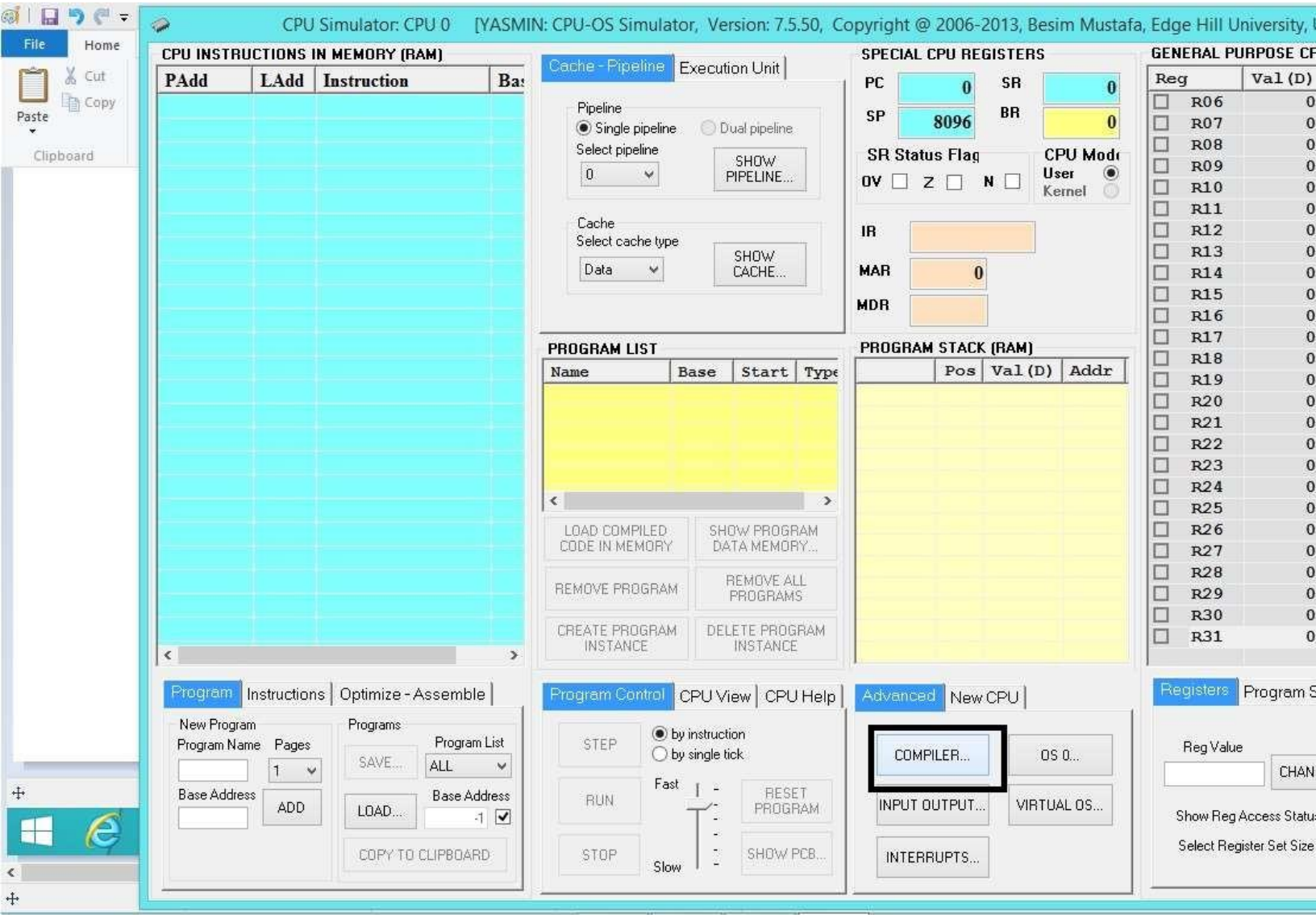
Shortest Job First (SJF)

This is also known as shortest job first, or SJF. This is a non-preemptive, pre-emptive scheduling algorithm. Best approach to minimize waiting time. Easy to implement in Batch systems where required CPU time is known in advance. Impossible to implement in interactive systems where required CPU time is not known. The processer should know in advance how much time process will take.

Installation steps of CPU-OS simulator:

- Go to http://www.teach-sim.com/. Select downloads and get the latest version of the simulator. Install the simulator using the downloaded executable file. It will install the CPU-OS Simulator on your system.
- Or you can install it from cpu-os-simulator.software.informer.com website.

- Open the compiler window by selecting the **COMPILER** button in the current window. You should now be looking at the compiler window.



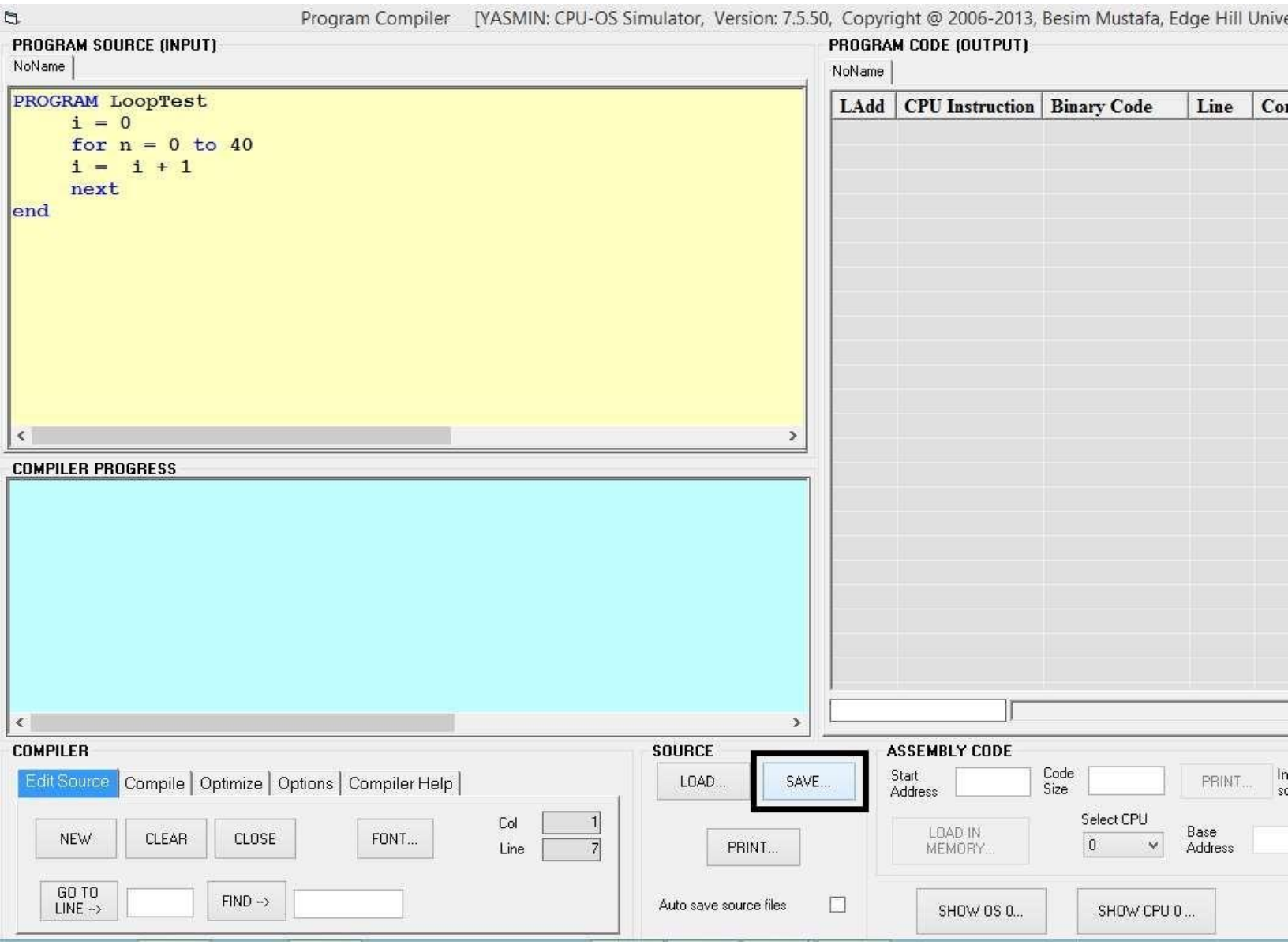
- In the compiler window, enter the following source code in the compiler's source editor window (under PROGRAM SOURCE frame title):

```
PROGRAM LoopTest
        i = 0
        for n = 0 to 40
        i = i + 1
```

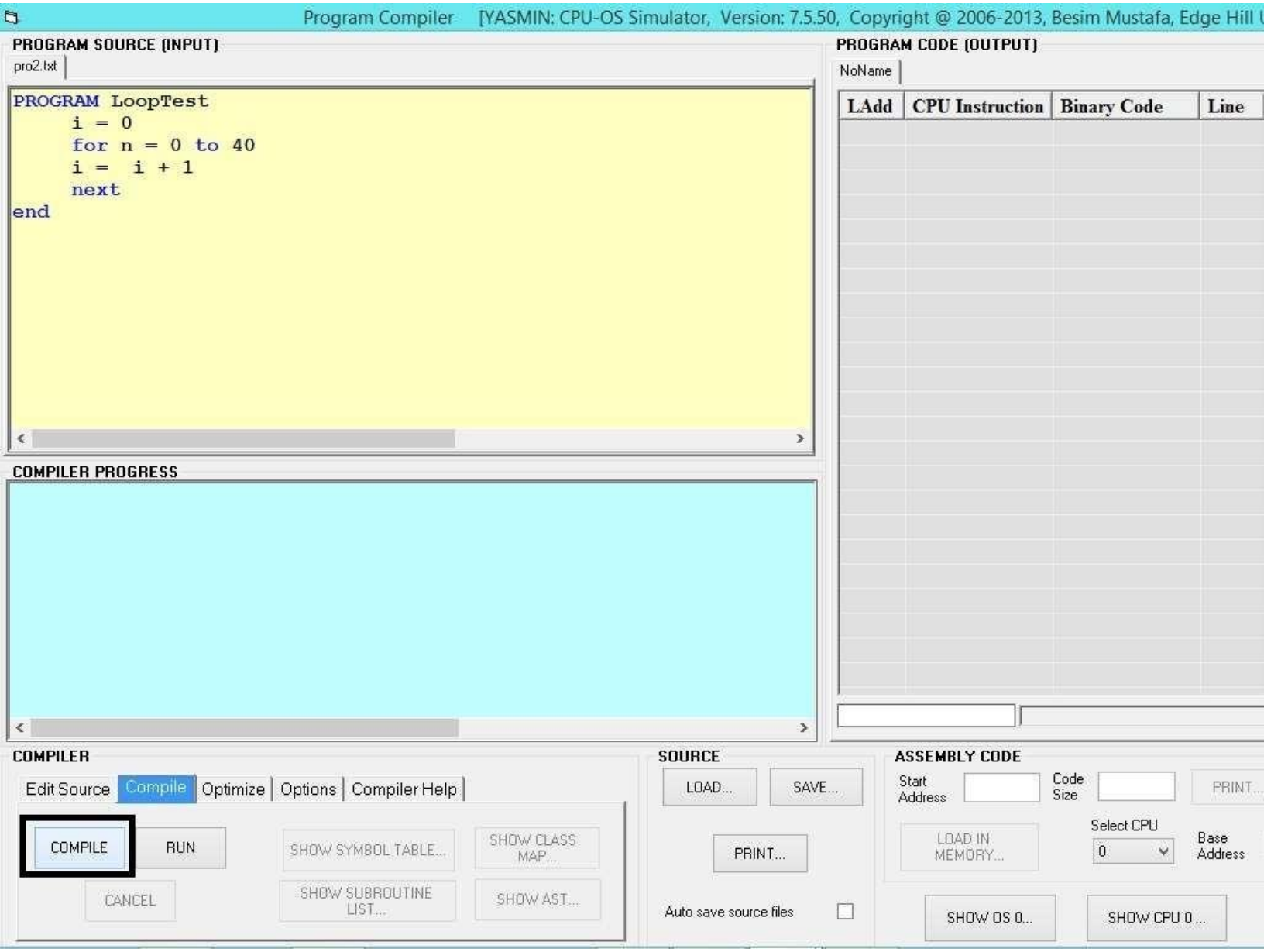
next

end

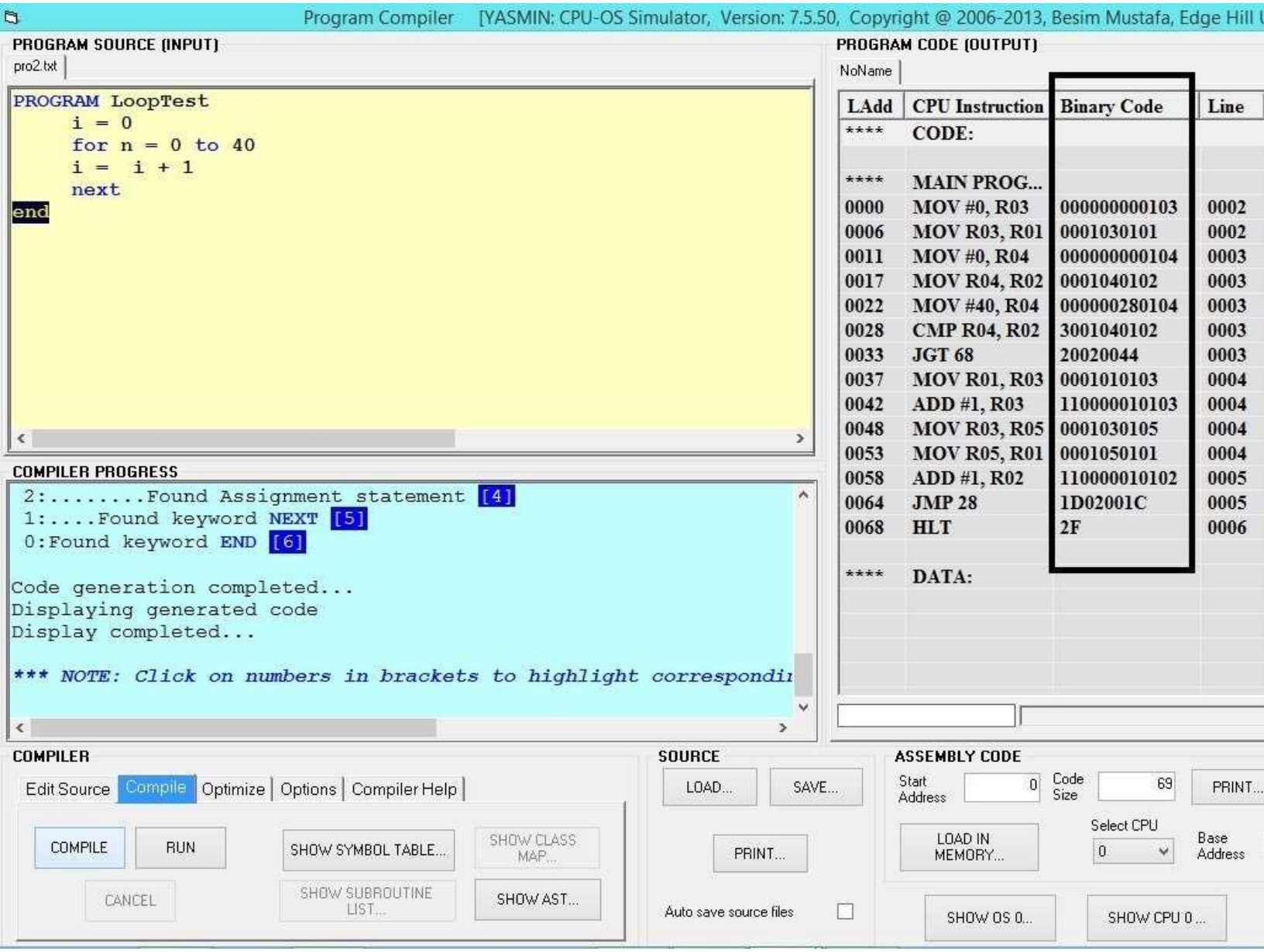
- Save your source code.



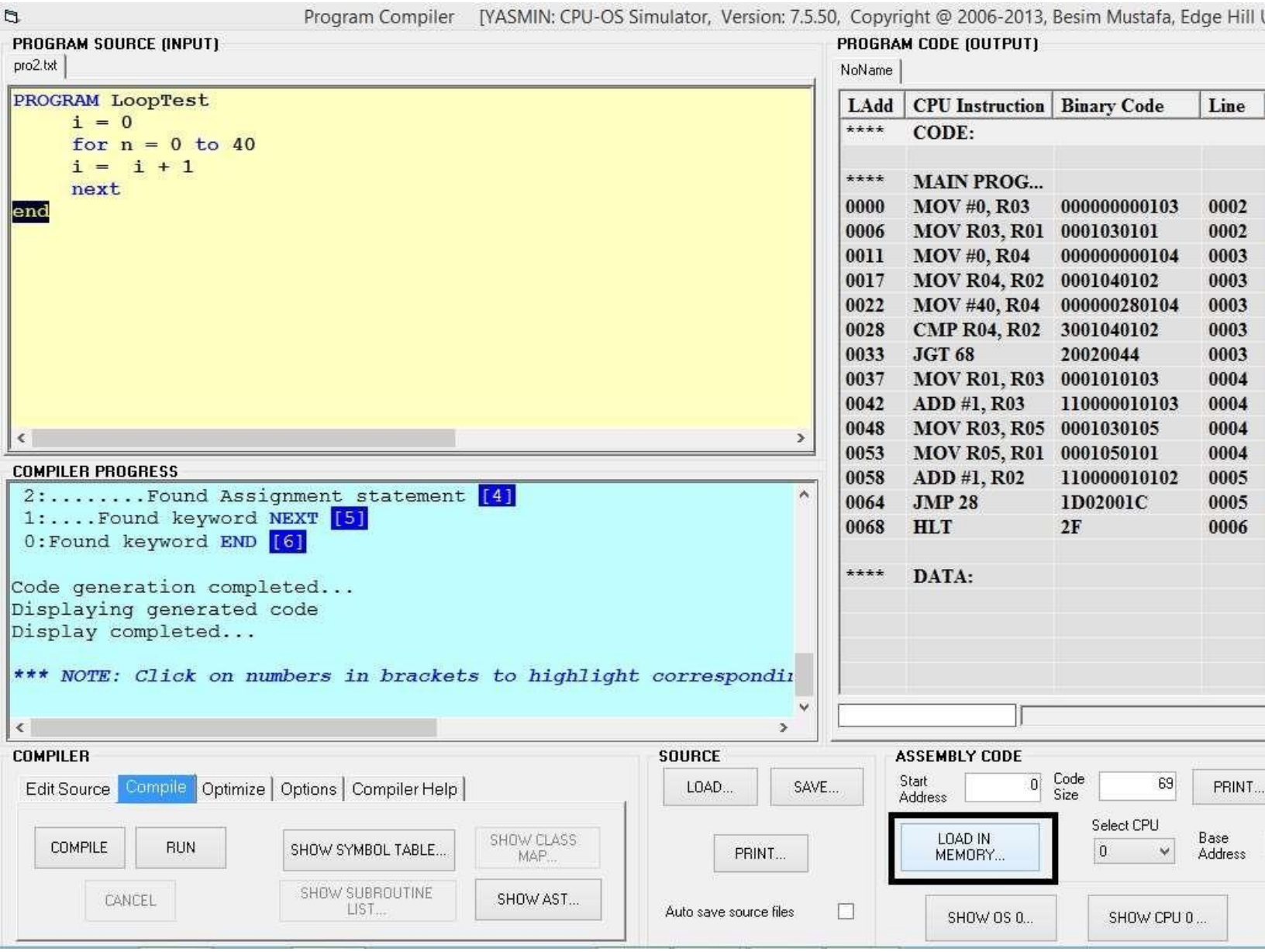
- Click on the **COMPILE** button. You should see the code created on the right in **PROGRAM CODE** view.



- Click on the button **SHOW** in **BINARY CODE** view. You should now see the **Hexadecimal Code for LOOPTEST** window

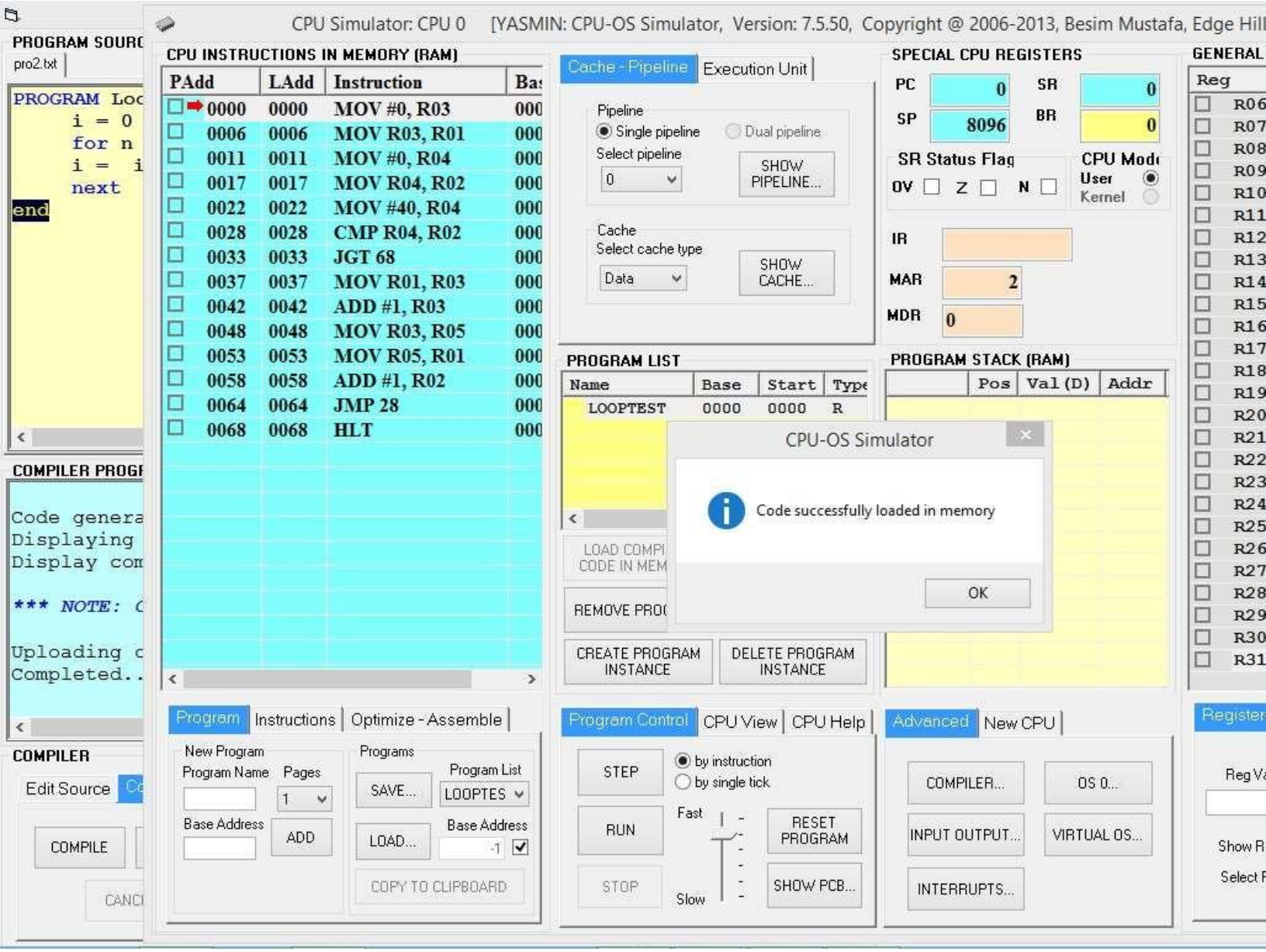

Program Compiler [YASMIN: CPU-OS Simulator, Version: 7.5.50, Copyright @ 2006-2013, Besim Mustafa, Edge Hill
PROGRAM SOURCE (INPUT)
pro2.txt
PROGRAM LoopTest
i = 0
for n = 0 to 40
i = i + 1
next
end
PROGRAM CODE (OUTPUT)
NoName
LAdd CPU Instruction Binary Code Line
**** CODE:
**** MAIN PROG...
0000 MOV #0, R03 000000000103 0002
0006 MOV R03, R01 0001030101 0002
0011 MOV #0, R04 000000000104 0003
0017 MOV R04, R02 0001040102 0003
0022 MOV #40, R04 000000280104 0003
0028 CMP R04, R02 3001040102 0003
0033 JGT 68 20020044 0003
0037 MOV R01, R03 0001010103 0004
0042 ADD #1, R03 110000010103 0004
0048 MOV R03, R05 0001030105 0004
0053 MOV R05, R01 0001050101 0004
0058 ADD #1, R02 110000010102 0005
0064 JMP 28 1D02001C 0005
0068 HLT 2F 0006
**** DATA:
COMPILER PROGRESS
2:.....Found Assignment statement [4]
1:....Found keyword NEXT [5]
0:Found keyword END [6]
Code generation completed...
Displaying generated code
Display completed...
*** NOTE: Click on numbers in brackets to highlight correspondi
COMPILER
Edit Source Compile Optimize Options Compiler Help
COMPILE RUN SHOW SYMBOL TABLE... SHOW CLASS MAP...
CANCEL SHOW SUBROUTINE LIST... SHOW AST...
SOURCE
LOAD... SAVE...
PRINT...
Auto save source files
ASSEMBLY CODE
Start Address 0 Code Size 69 PRINT...
LOAD IN MEMORY...
Select CPU 0 Base Address
SHOW OS 0... SHOW CPU 0 ...

- Now, this code needs to be loaded in memory so that the CPU can execute it. To do this,

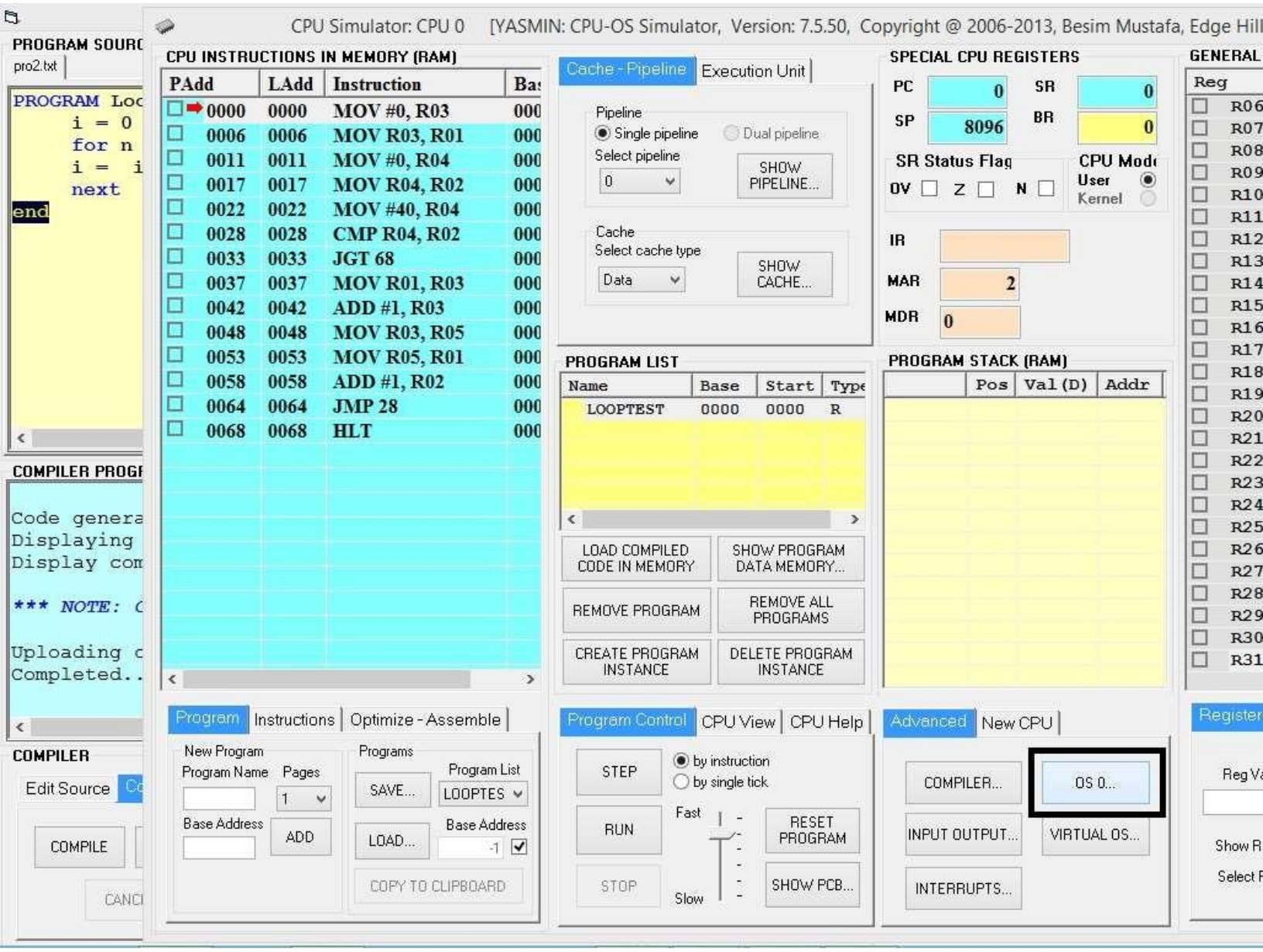


click on the **LOAD IN MEMORY** button in the current window.

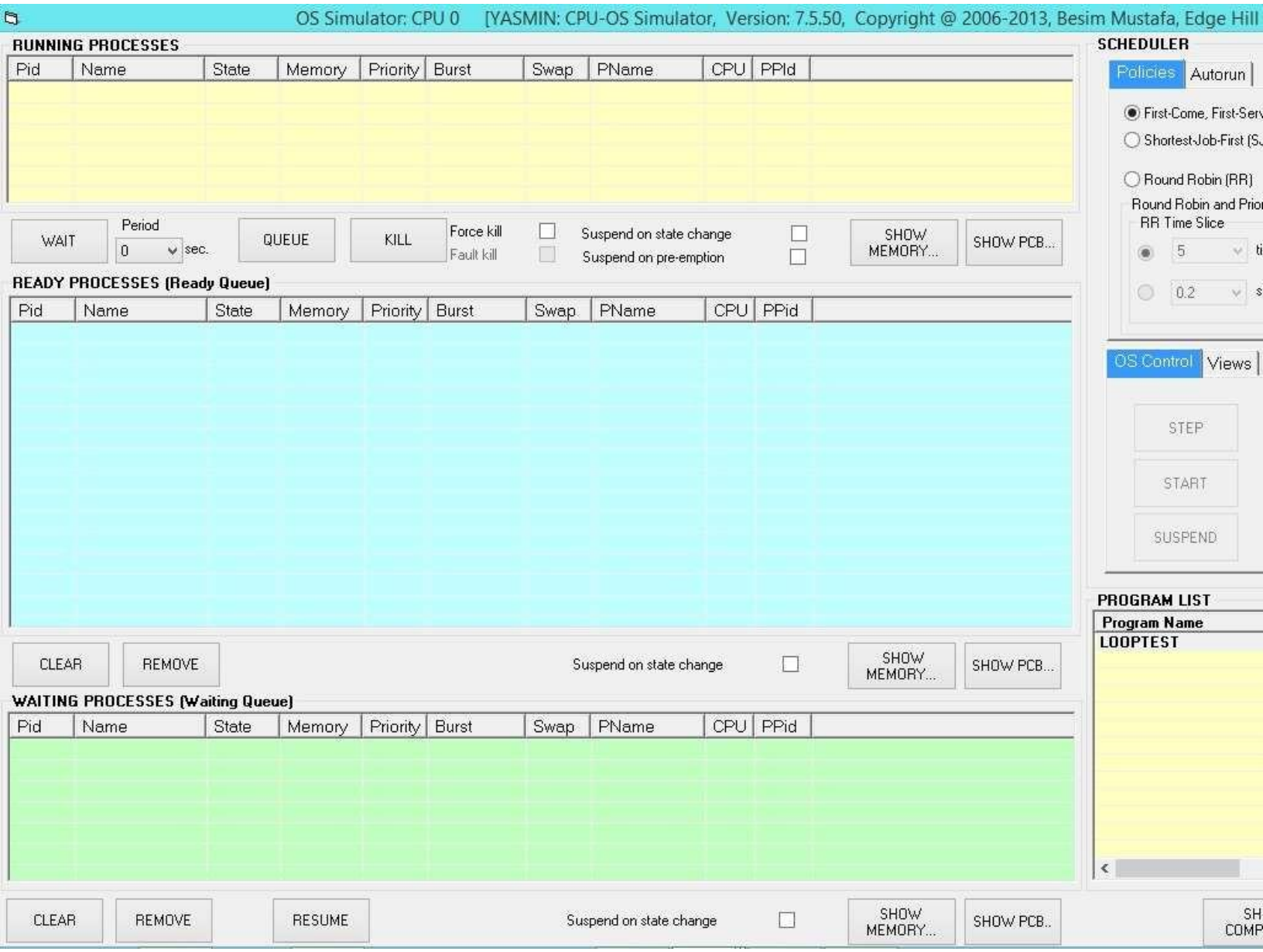
- You should now see the code loaded in memory ready to be executed. You are also back in the CPU simulator at this stage.



- To enter the OS simulator, click on the **OS 0** button in the current window. The OS window opens.



- You should see an entry, titled **LOOPTEST**, in the **PROGRAM LIST** view. Now that this program is available to the OS simulator. To create as many instances of LOOPTEST process click on **CREATE NEW PROCESS** button as shown below.

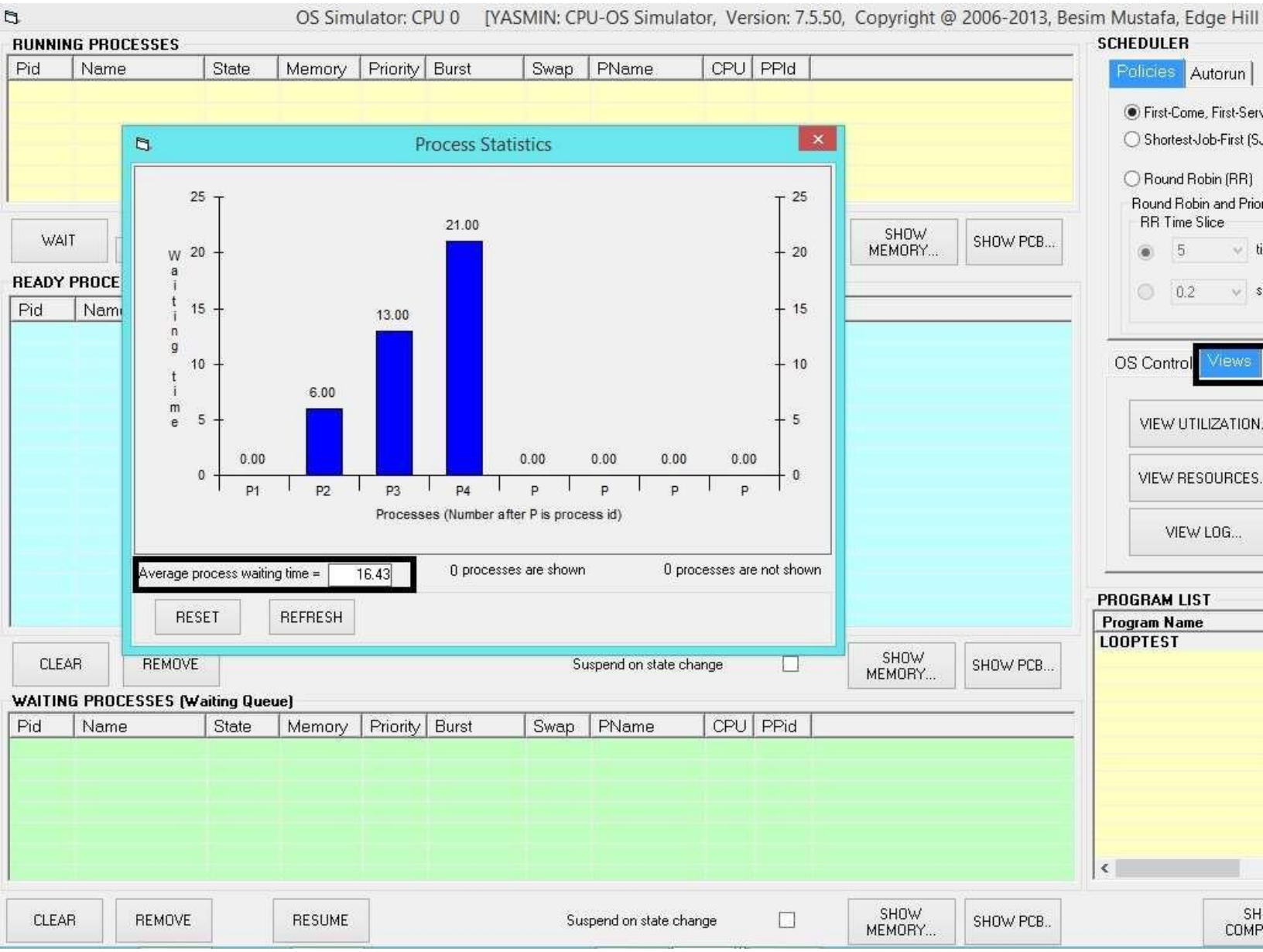


- Create 4 processes by clicking on the button 4 times. Observe the four instances of the program being queued in the ready queue which is represented by the **READY PROCESSES** view. Make sure the **First-Come-First-Served (FCFS)** option is selected in the **SCHEDULER/Policies** view. At this point the OS is inactive. To activate, first move the

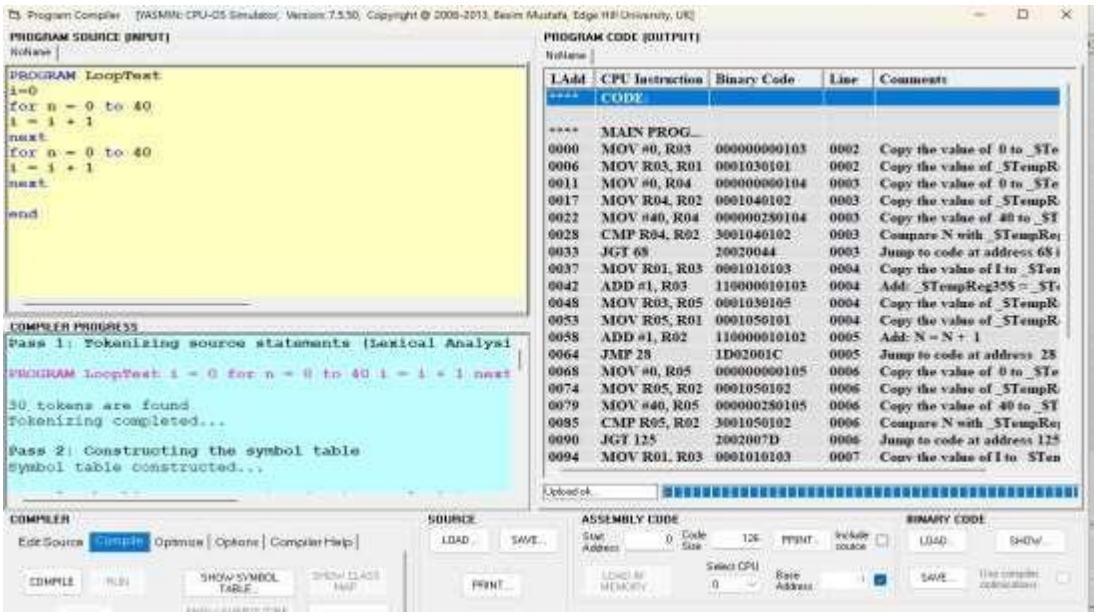
Speed slider to the fastest position, then click on the **START** button. This should start the OS simulator running the processes.

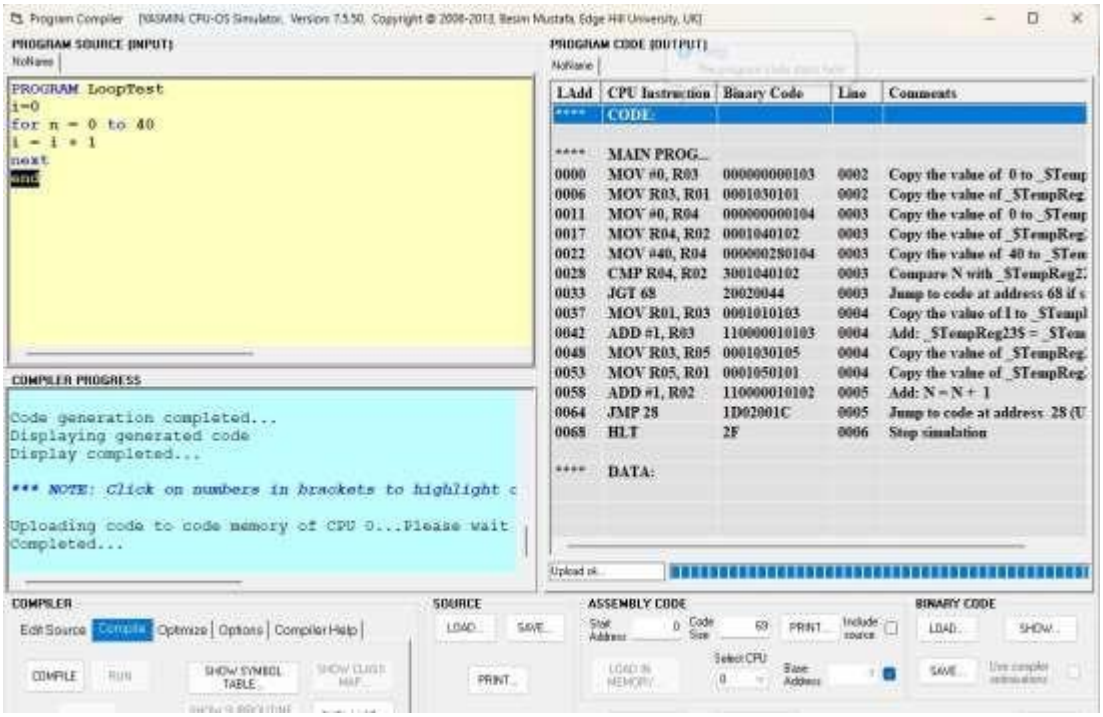
- Observe the instructions executing in the CPU simulator window. Observe the processes are running which you can see from two views that are **RUNNING PROCESSES** and **READY PROCESSES**

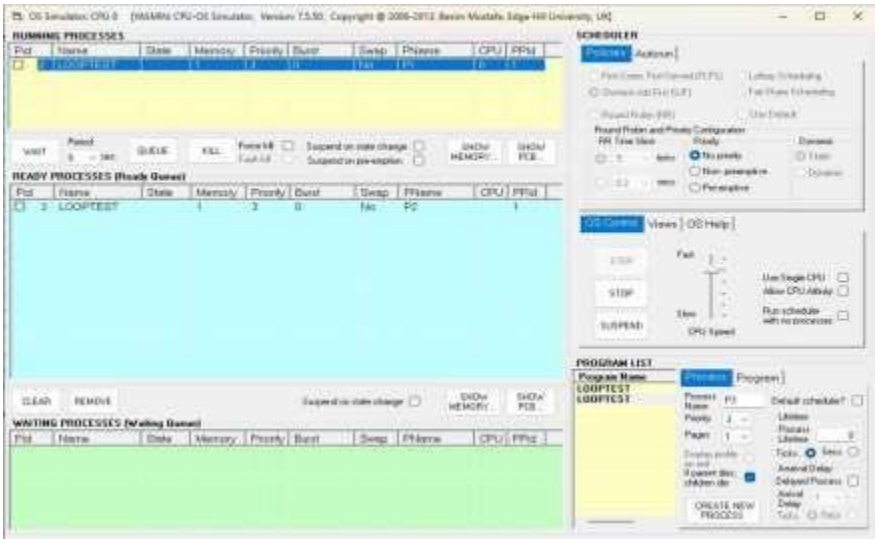
- Now, Figure out the average waiting time using the simulator.

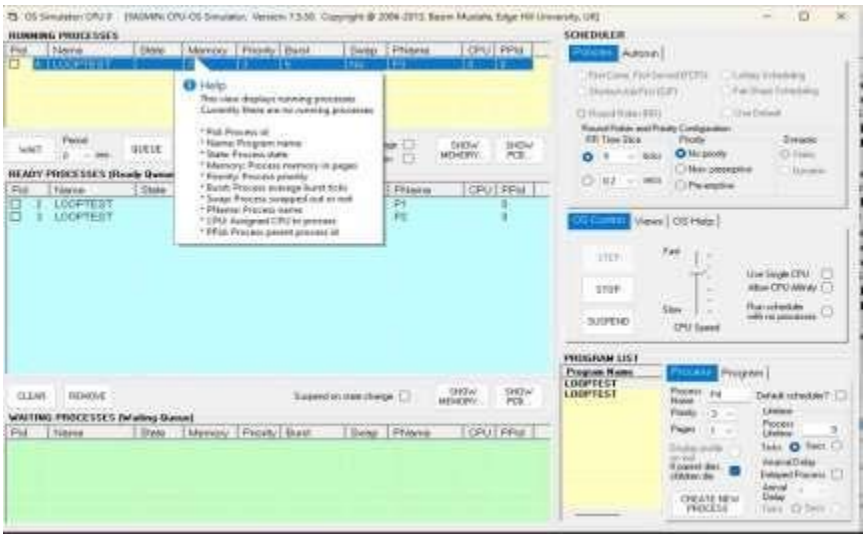

OS Simulator: CPU 0 [YASMIN: CPU-OS Simulator, Version: 7.5.50, Copyright @ 2006-2013, Besim Mustafa, Edge Hill
RUNNING PROCESSES
Pid Name State Memory Priority Burst Swap PName CPU PPId
SCHEDULER
Policies Autorun
First-Come, First-Serv
Shortest-Job-First (S
Round Robin (RR)
Round Robin and Prio
RR Time Slice
WAIT
SHOW MEMORY...
SHOW PCB...
READY PROCE
Pid Nam
OS Control Views
VIEW UTILIZATION
VIEW RESOURCES
VIEW LOG...
PROGRAM LIST
Program Name
LOOPTEST
Process Statistics
Waiting time
25
20
15
10
5
0
0.00
6.00
13.00
21.00
0.00
0.00
0.00
0.00
P1
P2
P3
P4
P
P
P
P
Processes (Number after P is process id)
Average process waiting time = 16.43
0 processes are shown
0 processes are not shown
RESET
REFRESH
CLEAR
REMOVE
Suspend on state change
SHOW MEMORY...
SHOW PCB...
WAITING PROCESSES (Waiting Queue)
Pid Name State Memory Priority Burst Swap PName CPU PPId
CLEAR
REMOVE
RESUME
Suspend on state change
SHOW MEMORY...
SHOW PCB..

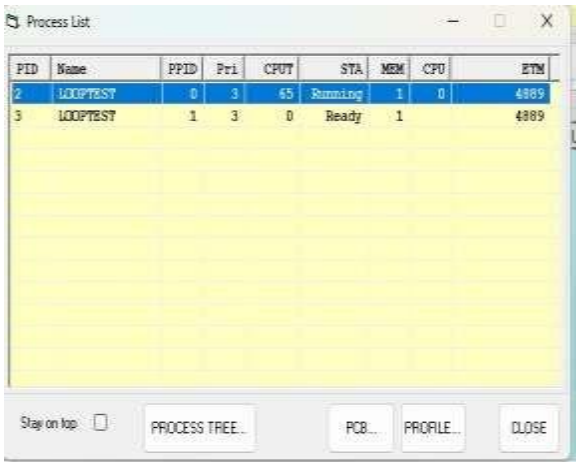
Result:

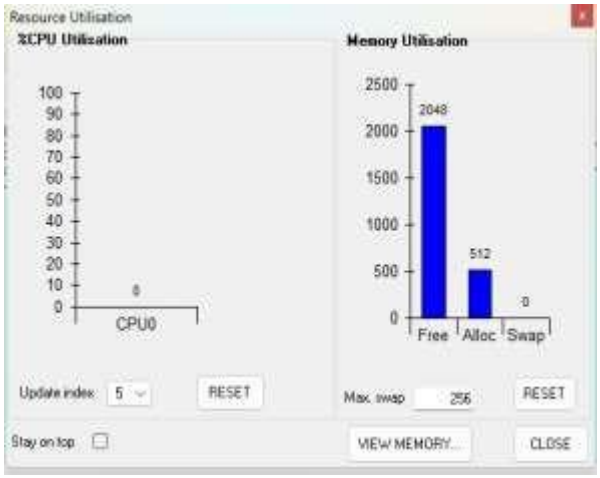

PROGRAM SOURCE (INPUT)
PROGRAM LoopTest
i=0
for n = 0 to 40
i = i + 1
next
for n = 0 to 40
i = i + 1
next
end
PROGRAM CODE (OUTPUT)
LAdd | CPU Instruction | Binary Code | Line | Comments
**** CODE:
**** MAIN PROG...
0000 MOV #0, R03 000000000103 0002 Copy the value of 0 to _STe
0006 MOV R03, R01 0001030101 0002 Copy the value of _STempR
0011 MOV #0, R04 000000000104 0003 Copy the value of 0 to _STe
0017 MOV R04, R02 0001040102 0003 Copy the value of _STempR
0022 MOV #40, R04 000000280104 0003 Copy the value of 40 to _ST
0028 CMP R04, R02 3001040102 0003 Compare N with _STempRe
0033 JGT 68 20020044 0003 Jump to code at address 68 i
0037 MOV R01, R03 0001010103 0004 Copy the value of I to _STen
0042 ADD #1, R03 110000010103 0004 Add: _STempReg35$ = _ST
0048 MOV R03, R05 0001030105 0004 Copy the value of _STempR
0053 MOV R05, R01 0001050101 0004 Copy the value of _STempR
0058 ADD #1, R02 110000010102 0005 Add: N = N + 1
0064 JMP 28 1D02001C 0005 Jump to code at address 28
0068 MOV #0, R05 000000000105 0006 Copy the value of 0 to _STe
0074 MOV R05, R02 0001050102 0006 Copy the value of _STempR
0079 MOV #40, R05 000000280105 0006 Copy the value of 40 to _ST
0085 CMP R05, R02 3001050102 0006 Compare N with _STempRe
0090 JGT 125 2002007D 0006 Jump to code at address 125
0094 MOV R01, R03 0001010103 0007 Copy the value of I to _STen
COMPILER PROGRESS
Pass 1: Tokenizing source statements (Lexical Analysi
PROGRAM LoopTest i = 0 for n = 0 to 40 i = i + 1 next
30 tokens are found
Tokenizing completed...
Pass 2: Constructing the symbol table
Symbol table constructed...
COMPILER
Edit Source | Compile | Optimize | Options | Compiler Help
COMPILE
RUN
SHOW SYMBOL TABLE
SOURCE
LOAD
SAVE
PRINT
ASSEMBLY CODE
Start Address
Code Size
PRINT
Include source
LOAD IN MEMORY
Select CPU
Base Address
BINARY CODE
LOAD
SHOW
SAVE


PROGRAM SOURCE (INPUT)
PROGRAM LoopTest
i=0
for n = 0 to 40
i = i + 1
next
end
PROGRAM CODE (OUTPUT)
LAdd | CPU Instruction | Binary Code | Line | Comments
**** CODE:
**** MAIN PROG...
0000 MOV #0, R03 000000000103 0002 Copy the value of 0 to _STemp
0006 MOV R03, R01 0001030101 0002 Copy the value of _STempReg
0011 MOV #0, R04 000000000104 0003 Copy the value of 0 to _STemp
0017 MOV R04, R02 0001040102 0003 Copy the value of _STempReg
0022 MOV #40, R04 000000280104 0003 Copy the value of 40 to _STem
0028 CMP R04, R02 3001040102 0003 Compare N with _STempReg2
0033 JGT 68 20020044 0003 Jump to code at address 68 if s
0037 MOV R01, R03 0001010103 0004 Copy the value of I to _STempl
0042 ADD #1, R03 110000010103 0004 Add: _STempReg23$ = _STem
0048 MOV R03, R05 0001030105 0004 Copy the value of _STempReg
0053 MOV R05, R01 0001050101 0004 Copy the value of _STempReg
0058 ADD #1, R02 110000010102 0005 Add: N = N + 1
0064 JMP 28 1D02001C 0005 Jump to code at address 28 (U
0068 HLT 2F 0006 Stop simulation
**** DATA:
COMPILER PROGRESS
Code generation completed...
Displaying generated code
Display completed...
*** NOTE: Click on numbers in brackets to highlight c
Uploading code to code memory of CPU 0...Please wait
Completed...
COMPILER
Edit Source | Compile | Optimize | Options | Compiler Help
COMPILE
RUN
SHOW SYMBOL TABLE
SOURCE
LOAD
SAVE
PRINT
ASSEMBLY CODE
Start Address
Code Size
PRINT
Include source
LOAD IN MEMORY
Select CPU
Base Address
BINARY CODE
LOAD
SHOW
SAVE









Conclusion: In conclusion, the CPU-OS simulator for process scheduling algorithms provides a valuable platform for studying, testing, and optimizing various scheduling strategies. It enables researchers and practitioners to analyze the performance of algorithms in a controlled environment, facilitating informed decision-making and driving innovation in operating system design.