**Overview**

The code implements a simple operating system (OS) simulator that processes jobs from an input file and writes the output to another file. The OS supports basic operations such as reading from and writing to memory, loading data into registers, storing data from registers into memory, comparing data, and conditional branching based on comparisons.

**Input Details:**

**$AMJ 0001 0003 0001** Job ID: 0001, Total Time limit: 0003, Total line limit: 0001

**GD 10 PD 10 H** Instructions to be executed

**$DTA** Start of data card

**Hello World** Data

**$END 0001** End Job 0001

**Key Components**

* **Memory (M):** 100 words, each word containing 4 bytes.
* **Instruction Register (IR)**: Holds the current instruction being executed.
* **Instruction Counter (IC):** Points to the next instruction to be executed.
* **General Purpose Register (R):** Used for temporary storage of data.
* **Toggle Register (C):** Used for conditional operations.
* **System Interrupt (SI):** Indicates the type of operation to be performed.
* **Buffer:** Temporarily holds data read from the input file.

**Functions**

1. **INIT():** Initializes memory, registers, and toggle register.

2. **MOS():** Master Mode Supervisor to handle system interrupts (SI).

3. **read():** Reads data from the input file to memory.

4. **write():** Writes data from memory to the output file.

5. **terminate():** Marks the end of a job.

6. **load():** Loads jobs from the input file into memory.

7. **Execute():** Executes the instructions loaded into memory.

8. **clearBuffer():** Clears the buffer.

9. **printMemory():** Prints the content of memory.

**Sequence of Execution**

**1. Initialization:**

* An instance of the `OS` class is created.
* Input (`input.txt`) and output (`output.txt`) files are opened.
* The program checks if the input file exists. If it doesn't, it prints "Failure" and exits. If it does, it prints "Success" and calls the `load()` function.

**2. Loading Jobs:**

* The `load()` function reads the input file line by line.
* It checks for control cards (`$AMJ`, `$DTA`, `$END`):
* **$AMJ:** Start of a new job. Calls `INIT()` to initialize the OS.
* **$DTA:** Start of data cards. Sets `IC` to 0 and calls `Execute()`.
* **$END:** End of the current job. Resets the memory pointer (`ptr`) to 0.
* If it's an instruction card, it loads instructions into memory starting from `M[0]`.

**3. Execution:**

* The `Execute()` function fetches instructions from memory using the instruction counter (`IC`) and loads them into the instruction register (`IR`).
* It increments `IC` to point to the next instruction.
* Based on the instruction in `IR`, it performs the corresponding operation:  
  **GD:** Read data from input file to memory. Sets `SI` to 1 and calls `MOS()`.  
  **PD:** Write data from memory to output file. Sets `SI` to 2 and calls `MOS()`.  
  **H:** Halt. Sets `SI` to 3 and calls `MOS()`.  
  **LR:** Load data from memory to general purpose register (`R`).  
  **SR:** Store data from general purpose register (`R`) to memory.  
  **CR:** Compare data in general purpose register (`R`) with memory.  
  **BT:** Branch if toggle register (`C`) is true.

**4. Handling System Interrupts:**

* The `MOS()` function handles different system interrupts:  
  **SI = 1:** Calls `read()` to read data from the input file to memory.  
  **SI = 2:** Calls `write()` to write data from memory to the output file.  
  **SI = 3:** Calls `terminate()` to mark the end of the current job.

**5.** **Read and Write Operations:**

* The `read()` function reads a line of data from the input file and stores it in memory starting from the address specified in the instruction.
* The `write()` function writes a block of data from memory to the output file starting from the address specified in the instruction.

**Example Execution**

Let's go through an example job from `input.txt`:

**Job 1**

$AMJ000100030001

GD10PD10H

$DTA

Hello World!

$END0001

* **$AMJ000100030001:** Start of a new job. Initializes the OS.
* **GD10PD10H:** Instructions for the job:
  + **GD10:** Read data into memory starting at `M[10]`.
  + **PD10:** Write data from memory starting at `M[10]` to the output file.
  + **H:** Halt.
* **$DTA:** Indicates the start of data.
* **Hello World!:** The actual data to be read into memory.
* **$END0001:** End of the job.

**Execution:**

1. **INIT()** is called to initialize the OS.

2. **GD10** reads "Hello World!" into memory starting at `M[10]`.

3. **PD10** writes the data from `M[10]` to the output file.

4. **H** halts the execution of the job.

5. **$END0001** marks the end of the job and resets the memory pointer.

The output for this job in `output.txt` will be:

**Hello World!**

**Job 2**

$AMJ0003000120004

GD20LR20SR33LR21SR32LR22SR31

LR23SR30PD30H$DTA

a\_\_\_b\_\_\_c\_\_\_d\_\_\_$END0002

**Execution:**

**$AMJ0003000120004:** Start of a new job.

Call INIT() to initialize the OS.

ptr = 0.

**GD20LR20SR33LR21SR32LR22SR31LR23SR30PD30H:** Load instructions into memory:

M[0] = 'GD20'

M[1] = 'LR20'

M[2] = 'SR33'

M[3] = 'LR21'

M[4] = 'SR32'

M[5] = 'LR22'

M[6] = 'SR31'

M[7] = 'LR23'

M[8] = 'SR30'

M[9] = 'PD30'

M[10] = 'H'

ptr now points to the next available memory location.

**$DTA:** Indicates the start of data.

Set IC = 0 and call Execute().

**Execution:**

IR = '**GD20**':

Set SI = 1 and call MOS().

MOS(): Calls read().

read(): Reads "a\_\_\_b\_\_\_c\_\_\_d\_\_\_" into memory starting at M[20]:

M[20] = 'a\_\_\_'

M[21] = 'b\_\_\_'

M[22] = 'c\_\_\_'

M[23] = 'd\_\_\_'

IR = '**LR20**':

Load M[20] into register R.

R = 'a\_\_\_'

IR = '**SR33**':

Store R into M[33].

M[33] = 'a\_\_\_'

IR = '**LR21**':

Load M[21] into register R.

R = 'b\_\_\_'

IR = '**SR32**':

Store R into M[32].

M[32] = 'b\_\_\_'

IR = '**LR22**':

Load M[22] into register R.

R = 'c\_\_\_'

IR = '**SR31**':

Store R into M[31].

M[31] = 'c\_\_\_'

IR = '**LR23**':

Load M[23] into register R.

R = 'd\_\_\_'

IR = '**SR30**':

Store R into M[30].

M[30] = 'd\_\_\_'

IR = '**PD30**':

Set SI = 2 and call MOS().

MOS(): Calls write().

write(): Writes data from M[30] to the output file.

IR = '**H**':

Set SI = 3 and call MOS().

MOS(): Calls terminate().

**$END0002**: End of the job. Reset ptr = 0.

**Output:**

d\_\_\_c\_\_\_b\_\_\_a\_\_\_

**Job 3**

$AMJ0002000120004

GD20GD30GD40GD50PD20PD30LR20CR30BT11

PD40PD50H$DTA

VIT

VIIT

NOT

SAME

$END0002

**Control Cards and Instructions:**

* **$AMJ0002000120004**: Start of a new job. The OS initializes using INIT().
* **GD20GD30GD40GD50PD20PD30LR20CR30BT11PD40PD50H:** Instructions to execute.
* **$DTA:** Indicates the start of data.
* **Data:**VIT  
  VIIT  
  NOT  
  SAME
* **$END0002:** End of the job.

**Execution:**

* **INIT():** Initializes the OS.
* **GD20:** Reads "VIT" into memory starting at M[20].
* **GD30:** Reads "VIIT" into memory starting at M[30].
* **GD40:** Reads "NOT" into memory starting at M[40].
* **GD50:** Reads "SAME" into memory starting at M[50].
* **PD20:** Writes data from memory starting at M[20] to the output file.
* **PD30:** Writes data from memory starting at M[30] to the output file.
* **LR20:** Loads data from M[20] into the general-purpose register R.
* **CR30:** Compares data in R with data at M[30] (comparison fails, C remains false).
* **BT11:** Branches to instruction at address 11 if C is true (does not branch since C is false).
* **PD40:** Writes data from memory starting at M[40] to the output file.
* **PD50:** Writes data from memory starting at M[50] to the output file.
* **H:** Halts the execution of the job.

The output for this job in `output.txt` will be:

**VIT**

**VIIT**

**NOT**

**SAME**

**Job 4**

$AMJ0005000170003

GD40GD20GD30PD40LR40CR43BT09PD20PD30

SR40LR41CR42BT14PD20PD30H$DTA

1\_\_\_0\_\_\_0\_\_\_1\_\_\_

NOT

PALINDROME

$END0005

**Control Cards and Instructions:**

* **$AMJ0005000170003:** Start of a new job. The OS initializes using INIT().
* **GD40GD20GD30PD40LR40CR43BT09PD20PD30SR40LR41CR42BT14PD20PD30H:** Instructions to execute.
* **$DTA:** Indicates the start of data.
* **Data:**1\_\_\_0\_\_\_0\_\_\_1\_\_\_  
  NOT  
  PALINDROME
* **$END0005:** End of the job.

**Execution:**

* **INIT():** Initializes the OS.
* **GD40:** Reads "1\_\_\_0\_\_\_0\_\_\_1\_\_\_" into memory starting at M[40].
* **GD20:** Reads "NOT" into memory starting at M[20].
* **GD30:** Reads "PALINDROME" into memory starting at M[30].
* **PD40:** Writes data from memory starting at M[40] to the output file.
* **LR40:** Loads data from M[40] into the general-purpose register R.
* **CR43:** Compares data in R with data at M[43] (comparison fails, C remains false).
* **BT09:** Branches to instruction at address 9 if C is true (does not branch since C is false).
* **PD20:** Writes data from memory starting at M[20] to the output file.
* **PD30:** Writes data from memory starting at M[30] to the output file.
* **SR40:** Stores data from R into M[40].
* **LR41:** Loads data from M[41] into the general-purpose register R.
* **CR42:** Compares data in R with data at M[42] (comparison fails, C remains false).
* **BT14:** Branches to instruction at address 14 if C is true (does not branch since C is false).
* **PD20:** Writes data from memory starting at M[20] to the output file.
* **PD30:** Writes data from memory starting at M[30] to the output file.
* **H:** Halts the execution of the job.

The output for this job in `output.txt` will be:

**1\_\_\_0\_\_\_0\_\_\_1\_\_\_**

**NOT**

**PALINDROME**

**Job 5**

$AMJ0003000120004

GD20LR20SR35PD30SR43SR47PD40SR51SR55

SR59PD50H

$DTA

\*

$END0002

**Control Cards and Instructions:**

* **$AMJ0003000120004:** Start of a new job. The OS initializes using INIT().
* **GD20LR20SR35PD30SR43SR47PD40SR51SR55SR59PD50H:** Instructions to execute.
* **$DTA:** Indicates the start of data.
* **Data:** \*
* **$END0002:** End of the job.

**Execution:**

* **INIT():** Initializes the OS.
* **GD20:** Reads "\*" into memory starting at M[20].
* **LR20:** Loads data from M[20] into the general-purpose register R.
* **SR35:** Stores data from R into M[35].
* **PD30:** Writes data from memory starting at M[30] to the output file (note that M[30] might be empty).
* **SR43:** Stores data from R into M[43].
* **SR47:** Stores data from R into M[47].
* **PD40:** Writes data from memory starting at M[40] to the output file (note that M[40] might be empty).
* **SR51:** Stores data from R into M[51].
* **SR55:** Stores data from R into M[55].
* **SR59:** Stores data from R into M[59].
* **PD50:** Writes data from memory starting at M[50] to the output file (note that M[50] might be empty).
* **H:** Halts the execution of the job.

The output for this job in `output.txt` will be:

**\***

**\* \***

**\* \* \***

**Job 6**

$AMJ0003000120004

GD20PD20LR20SR21PD20SR22PD20SR23PD20

H

$DTA

\*

$END0002

**Control Cards and Instructions:**

* **$AMJ0003000120004:** Start of a new job. The OS initializes using INIT().
* **GD20PD20LR20SR21PD20SR22PD20SR23PD20H:** Instructions to execute.
* **$DTA:** Indicates the start of data.
* **Data:** \*
* **$END0002:** End of the job.

**Execution:**

* **INIT():** Initializes the OS.
* **GD20:** Reads "\*" into memory starting at M[20].
* **PD20:** Writes data from memory starting at M[20] to the output file.
* **LR20:** Loads data from M[20] into the general-purpose register R.
* **SR21:** Stores data from R into M[21].
* **PD20:** Writes data from memory starting at M[20] to the output file.
* **SR22:** Stores data from R into M[22].
* **PD20:** Writes data from memory starting at M[20] to the output file.
* **SR23:** Stores data from R into M[23].
* **PD20:** Writes data from memory starting at M[20] to the output file.
* **H:** Halts the execution of the job.

The output for this job in `output.txt` will be:

**\***

**\* \***

**\* \* \***

**\* \* \* \***

**Conclusion**

This OS simulator reads jobs from an input file, processes the instructions, and writes the output to another file. It demonstrates basic OS functionalities such as reading, writing, loading, storing, comparing, and conditional branching, which are executed based on the instructions provided in the input file.