Program Layout & Execution: Factorial Recursion

Team Members:

- Ganesh (014631747)
- Tushar (018282446)
- Jeevan (017739397)
- Sindhura (018169840)

GitHub Repository: https://github.com/ganeshsjsu/softcpu

---

1. How to Run the Factorial Program

Follow these steps to download, compile, and execute the factorial recursion program.

Step 1: Download & Build

```
# Clone the repository
git clone https://github.com/ganeshsjsu/softcpu.git
cd software-cpu

# Build the emulator and assembler
make
```

Step 2: Run the Factorial (Assembly Version)

This is the main project deliverable using our custom SoftCPU-16.

```
# 1. Assemble the source code
./softcpu assemble programs/factorial.asm -o build/factorial.bin

# 2. Run the program (Output: 120)
./softcpu run build/factorial.bin

# 3. (Optional) Run with trace to visualize recursion
./softcpu run build/factorial.bin --trace
```

Step 3: Run the C Prototype

This is the C proof-of-concept created to demonstrate the logic.

```
gcc recursion_demo.c -o recursion_demo
./recursion_demo
```

---

2. Program Layout & Execution Analysis

Memory Layout

The SoftCPU-16 architecture utilizes a 64KB address space (`0x0000` to `0xFFFF`).

- **Code Segment (`0x0000`)**: The instructions for the factorial program are loaded starting at `0x0000`.
- **Stack Segment (`0xFFFF`)**: The stack is initialized at the top of memory (`0xFFFF`) and grows **downwards**.
  - `R7` serves as the Stack Pointer (SP).
  - Recursion pushes data downwards (decreasing address).
- **IO Segment (`0xFF00`)**: Address `0xFF00` is memory-mapped to the console output. We write to this address to print the final decimal result.

Function Calls & Recursion Logic

The functionality is implemented using standard Stack Frame conventions:

1. **Function Call (`CALL`)**:
   - The `CALL factorial` instruction pushes the **Return Address** onto the stack and jumps to the function label.
   - This allows the CPU to know where to return after the function completes.
2. **Recursive Step (`PUSH`)**:
   - As `factorial(n)` calls `factorial(n-1)`, we must preserve the current value of `n`.
   - `PUSH R0`: Saves the current `n` to the stack.
   - A new stack frame is created for each call (e.g., for 5, then 4, then 3...), deepening the stack.
3. **Unwinding (`RET` & `POP`)**:
   - Once the base case (`n=1`) is reached, the function returns.
   - `POP R1`: The saved values are popped off the stack in reverse order.
   - The calculation `result = result * popped_n` is performed as the stack unwinds back to the main function.

---

3. Contributions

- **Ganesh**: Implemented the core Emulator and `factorial.asm` logic. Wrote the report.
- **Tushar**: Designed the Instruction Set Architecture (ISA) and documentation.
- **Jeevan**: Implemented the Assembler and binary file generation.
- **Sindhura**: Created the test cases and verified the factorial recursion trace.