Program Layout & Execution:

Recursion & Call Stack on Tiny16 CPU

CMPE 220 – System Software (FA25)

Team Members

Abdul Muqtadir Mohammed

Akash Kishorbhai Devani

Faisal Barkatali Budhwani

Venkata Sai Anjana Karthikeya Nimmala Sri Naga

GitHub Repository

The full source code, recursion video, and documentation for this project are available at:

Repository URL:

https://github.com/F1804/software$_cpu_desig$n

This repository includes:

- `tiny16.cpp` — Tiny16 software CPU, assembler, and emulator.
- `factorial.c` — C program implementing recursive factorial.
- `Factorial Demo.mp4` — Recursion and call-stack explanation video.
- `Sample Drawing.png` — Memory layout and stack-frame diagram.
- `Factorial.pdf` — Final program report.

How to Download, Compile, and Run the Program

This project demonstrates recursion, memory layout, and stack behavior on both C and the Tiny16 software CPU.

1. Download the Project

Open a terminal:

```
git clone https://github.com/F1804/software_cpu_design.git
cd software_cpu_design
```

2. Compile and Run the Recursive C Program

The recursive factorial implementation is located in:

```
factorial.c
```

Compile:

```
gcc -std=c11 -O2 -o factorial factorial.c
```

Run:

```
./factorial
```

Example:

```
Enter a number: 5
Factorial of 5 = 120
```

3. Compile the Tiny16 CPU

The Tiny16 architecture is implemented in:

```
tiny16.cpp
```

Compile with:

```
g++ -std=c++17 -O2 -o tiny16 tiny16.cpp
```

4. Run Recursive Factorial on Tiny16

Tiny16 assembly implementation:

```
examples/fact.asm
```

Run directly:

```
./tiny16 run examples/fact.asm
```

This demonstrates:

- Function calls using `CALL` and `RET`
- Stack-frame creation for recursion
- Base-case detection and unwinding
- Returning values through registers/stack

5. Examine Memory Layout and Call Stack

Assemble manually:

```
./tiny16 asm examples/fact.asm -o fact.bin
```

Run with emulator + memory dump:

```
./tiny16 emu fact.bin --base 0x0000 --pc 0x0000 --dump 0x0000 0x01FF
```

This shows:

- Code and data placement
- Active stack frames for each recursive call
- Return addresses pushed to stack

6. View the Recursion Video

The file:

```
Factorial Demo.mp4
```

explains:

- How Tiny16 handles function calls
- Stack pointer movement
- Recursive expansion
- Stack unwinding back to caller

Team Member Contributions

Abdul Muqtadir Mohammed

- Analyzed Tiny16 memory layout and documented code, data, and stack organization for recursion.
- Explained fetch–decode–execute flow for recursive calls, focusing on stack pointer changes and return-address handling.
- Generated emulator memory dumps to illustrate stack-frame creation and unwinding during recursion.

Akash Kishorbhai Devani

- Organized all project deliverables—including source code, diagrams, video, and documentation—into a clear GitHub structure.
- Recorded and edited the recursion demonstration video (`Factorial Demo.mp4`), explaining call-stack behavior on Tiny16.
- Reviewed project documentation for clarity, consistency, and accuracy, ensuring the final submission met all formatting requirements.

Faisal Barkatali Budhwani

- Implemented and tested the recursive C factorial program in `factorial.c`, including input handling and recursion logic.
- Validated factorial correctness by comparing Tiny16 and C program outputs for multiple test values.
- Created explanatory documentation describing recursion flow and the relationship between C and Tiny16 implementations.

Venkata Sai Anjana Karthikeya Nimmala Sri Naga

- Implemented the Tiny16 recursive factorial algorithm in `fact.asm`, including CALL/RET logic and stack-frame management.
- Designed the Tiny16 calling convention, defining how parameters, return values, and return addresses propagate through recursion.
- Added detailed comments to visualize stack growth, base-case evaluation, and stack unwinding in the assembly program.