TDT4258 Lab 1

ARM Assembly Programming

👩‍🏫 Teaching Assistant: Callum Gran

Agenda

1. Lab overview
2. The main task: Palindrome Finder
3. Example inputs & wildcards
4. Recommended approach
5. CPUlator basics
6. Output requirements
7. Memory-mapped I/O
8. Common pitfalls & Debugging tips
9. Optional tasks
10. Submission & wrap-up

Lab 1 Overview

- **Goal:** Learn ARM assembly programming & I/O devices
- **Tools:** CPUlator simulator
- **Grading:** 10 points (need 27 total for exam)
- **Plagiarism rules:**
  - ✅ Discuss ideas
  - ❌ Share code

Main Task: Palindrome Finder

- Input: alphanumeric + spaces
- Case-insensitive, spaces ignored
- Wildcards:
  - `?` → matches any single character (not space)
  - `#` → matches any single character (not space)

Example Inputs

✅ Palindromes

- `level`
- `step on no pets`
- `abc?dc#a`

❌ Not palindromes

- `Palindrome`
- `First level`

Why is `abc?dc#a` a Palindrome?

Let's analyze character by character:

`a b c ? d c # a`

- First (`a`) matches last (`a`) ✅
- Second (`b`) matches `#` → `#` can be **any char except space** → choose `b` ✅
- Third (`c`) matches `c` ✅
- Fourth (`?`) matches `d` → `?` can be **any char except space** → choose `d` ✅

Recommended Approach

1. Write a **high-level solution** first (Python, C, etc.)
2. Test thoroughly
3. Translate step-by-step to ARM assembly
4. Use the provided skeleton `palinfinder.s`

Setting up CPUlator

- Go to: https://cpulator.01xz.net
- Select **ARMv7 → ARMv7 DE1-SoC**
  - Development and Education board, 1st generation, with System-on-Chip
  - Allows usage of JTAG UART and LEDs (memory-mapped I/O)
    - *You will learn more about this in Lecture 2*
- Documentation: CPUlator Docs
- Read the **sample programs** and familiarize yourself with them

CPUlator Basics

- Compile & run: **F5** → **Compile and Load**
- Step through operations with **Step Into**
- Inspect: registers, memory, IO
- End program the program (`_exit: b .`)

Output Requirements

- **LEDs**:
  - Palindrome → rightmost 5 LEDs
  - Not palindrome → leftmost 5 LEDs
- **JTAG UART**:
  - Palindrome → `"Palindrome detected"`
  - Not palindrome → `"Not a palindrome"`

Memory-Mapped I/O

Address	Device	Usage
`0xFF200000`	**LEDs (Red)**	Write value → lights up corresponding LEDs
`0xFF201000`	**JTAG UART**	Write ASCII char → sends to terminal

💡 Remember: these addresses behave like memory, but instead of RAM they **control hardware**.

Common Pitfalls

- Forgetting to normalize **case**
- Mishandling **wildcards** `?` and `#`
- Input length not checked (≥ 2 chars)
- Off-by-one errors when comparing chars
- `.ascii` vs `.asciz` (missing null terminator!)

Debugging Tips

- Step instruction by instruction
- Watch registers change
- Test multiple inputs
- Comment everything
  - If you can explain it, you know what it does!

Optional Tasks (Quick Look)

- Print numbers 0–99 via UART
- Reverse words and characters
- Two-sum problem using stack
- VGA graphics (colors & pixels)

(Not graded, but great for practice!)

Register Clobbering (ARMv7)

ARM Calling Conventions

- **Caller-saved** registers (must be saved if needed):
  - `r0–r3` (arguments, return value)
  - `r12` (scratch)
- **Callee-saved** registers (function must restore them):
  - `r4–r11` (used for local variables, preserved across calls)
- `sp` (stack pointer), `lr` (link register), `pc` (program counter) have special roles

What is "Register Clobbering"?

- Occurs when a function **overwrites a register** the caller expected to remain intact
- Example: function writes to `r4` but does **not restore** it → caller's data lost
- Assembler/CPUlator may warn:

  > *Function clobbered register(s)*

In Practice for Lab 1

- Lab 1 is forgiving: full points possible without strict conventions
- From Lab 2 onwards → **must** follow conventions for full score
- Good habit:
  - Save needed caller registers before a call
  - Restore callee-saved registers before returning

Submission Requirements

- Single file: `palinfinder.s`
- Keep input variable name unchanged
- Commented & tidy code
- Add **AI statement**
- Submit before **19th Sept 17:00**

Wrap-Up

✅ Start early, test often
✅ Comment your code
✅ Use CPUlator to debug step-by-step

📌 Ask questions on the forum (others may have the same issue)

Time for help :)

(Example of workflow from C to ARMv7 if you wish)