# Terminal Window Exercise: Feedback

- **Check range of printc character value**

  **most of you passed the character using a (32-bit) register**

  **→ without bound checking: this allows an attacker to dump any memory on the v-screen!**

  **Note that you wil execute printc in priveleged mode!**

- **Check the range of the vscreen framebuffer memory when setting pixel values! (don‘t just rely on the OS)**
  - **What happens on backspace? (buffer underflow)**

    **also: what happens if backspace is fired at left border positions?**
  - **What happens on a long string? (buffer overflow)**

- **You will likely check unsigned numbers (not signed)**

# Terminal Window Exercise: Feedback

- **Check your SP when program terminates**
  **→ Each function pushing items on stack has to remove (pop) exactly the same number of items (or a called function has to do that)**
  - **Probably not a problem here: be careful if stack can grow on user data (e.g., if you copy all characters of a string on stack)**

- **Tabulator is not just adding a fixed number of spaces**
  **→ you have to add 1 or more spaces until the cursor reaches a mod(tab_width) position (4 or 8 is easy)**

- **Decouple content from code**

```
ADRL    R0, test_string
BL      printstr
BL      pr_newline_string
BL      pr_newline_string     ; readability
```

**(The newline should be printable from the string directly)**

# Terminal Window Exercise: Feedback

**What do we have to change to have a "real" terminal window?**

**Like in a shell or dos-box window, where you can**

- **scroll through a buffer**
- **select and copy text**
- **run complex applications (e.g., editors, ASCII web browsers)**

- **Text buffer:**
  - **We would not print directly with printc or printstr into the v-screen frame buffer, but instead into a text buffer**
  - **The text buffer may be larger than what can be displayed on the screen**
  - **The buffer may include text (ASCII) and metadata (e.g., colour)**
  - **Changes in the text buffer trigger (selectively) updates in the frame buffer**
  - **May come with more functions, like set_cursor etc.**

# Virtual Lab Installation

**please get the update with timer support**

Its easiest if you load the virtual lab environment
http://www.cs.man.ac.uk/~kochd/COMP22712/kmd_setup_v2.tar.gz
directly into your Linux in a local directory.
Use wget or your browser (inside your Linux VirtualBox) for this.
Note: you can setup a shared folder, but installation may fail when using that for the installation. However, you can use a shared folder to copy data from your host machine into the VM all the way to a local directory.

Then open a shell go into your local directory containing kmd_setup_v2.tar.gz and run:

```
tar -xzf kmd_setup_v2.tar.gz

cd kmd_setup_v2_clean

./setup_vm_for_kmd.sh
```

**As usual: make a bakup (outside the VM) of all your code before the update!**

After the installation, the virtual lab environment is started.
Otherwise, you can start the environment with


# Website Update

- All video recordings up to date
- All slides up to date
- Lab manual update
- Also: first lab assignment marked

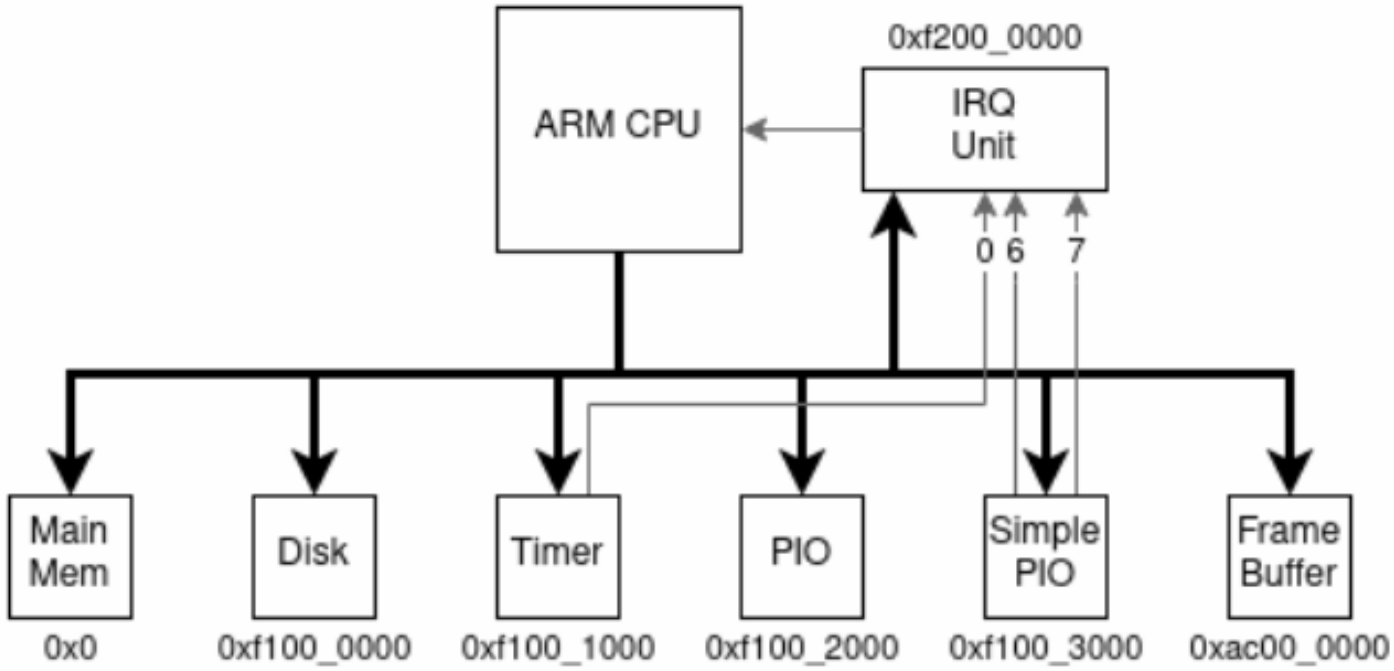
  (and feedback provided)

# Lab Manual Update



### 3.3.3 Virtual environment memory map

The virtual environment is similar to the physical arm board, but has many differences in addresses and bit maps.




**Section 3.3.3 provides the final address map used for the virtual lab**

### 3.3.4 Virtual environment I/O port map

The internal I/O region map base address `f1000000` is given in table 3.5.

| Offset | Direction | Register | Remarks |
|---|---|---|---|
| 0000 | R/W | Buffer address | Address to write disk data to |
| 0004 | R/W | Sector number | Which disk sector to read, writing to this address starts the transfer |
| 0008 | RO | Disk size | Size of disk in bytes |
| 1000 | RO | Clock cycles | Total number of emulated instructions |
| 1004 | RO | Unix seconds | The current Unix timestamp |
| 1008 | RO | Unix nanos | |
| 100c | R/W | Control register | Bit 0 = Timer enabled; Bit 1 = Use FIQ over IRQ |
| 1010 | RO | Timer | 8-bit free running, incrementing at 1 kHz (tied to clock cycles) |
| 1014 | R/W | Timer compare | Interrupt asserted when timer equals this register |
| 2000 | R/W | PIO direction | Bit 1 = Tx ready; Bit 0 = Rx ready |
| 2004 | R/W | PIO value | Connected to keypad buttons |
| 3000 | RO | Simple PIO | Connected to A, B, C, D buttons |