- Remember to register your attendance using the UoB Check-In app. Either

1. download, install, and use the native app[a] available for Android and iOS, or
2. directly use the web-based app available at

https://check-in.bristol.ac.uk

noting the latter is also linked to via the `Attendance` menu item on the left-hand side of the Blackboard-based unit web-site.

- The hardware *and* software resources located in the MVB Linux lab(s). (e.g., MVB-1.15 or MVB-2.11) are managed by the Faculty IT Support Team, a subset of IT Services. If you encounter a problem (e.g., a workstation that fails to boot, an error when you try to use some software, or you just cannot log into your account), they can help: you can contact them, to report then resolve said problem, via

https://www.bristol.ac.uk/it-support

- The lab. worksheet is written *assuming* you work in the lab. using UoB-managed and thus *supported* equipment. If you need or prefer to use your own equipment, however, various *unsupported*[b] alternatives available: for example, *you* could 1) manually install any software dependencies yourself, *or* 2) use the unit-specific Vagrant[c] box by following instructions at

https://cs-uob.github.io/COMS10015/vm

- The questions are roughly classified as either C (for core questions, that *should* be attempted within the lab. slot), A (for additional questions, that *could* be attempted within the lab. slot), or R (for revision questions). Keep in mind that we only *expect* you to attempt the C-class questions: the other classes are provided *purely* for your benefit and/or interest, so there is no problem with nor penalty for totally ignoring them.
- There is an associated set of solutions is available, at least for the C-class questions. These solutions are there for you to learn from (e.g., to provide an explanation or hint, or illustrate a range of different solutions and/or trade-offs), rather than (purely) to judge your solution against; they often present *a* solution vs. *the* solution, meaning there might be many valid approaches to and solutions for a question.
- Keep in mind that various mechanisms exist to get support with and/or feedback on your work; these include both in-person (e.g., the lab. slot itself) *and* online (e.g., the unit forum, accessible via the unit web-site) instances.

[a] https://www.bristol.ac.uk/students/support/it/software-and-online-resources/registering-attendance
[b] The implication here is that such alternatives are provided in a best-effort attempt to help you: they are experimental, and so *no* guarantees about nor support for their use will be offered.
[c] https://www.vagrantup.com

COMS10015 lab. worksheet #10

During the period of time aligned with this lab. worksheet, there is an active (or open) coursework assignment for the unit. You could address this fact by dividing your time between them. However, our (strong) suggestion is to view the former as of secondary importance (or optional, basically), and instead focus on the latter: since it is credit bearing, the coursework assignment should be viewed as of primary importance. Put another way, focus exclusively on completing the latter before you invest any time at all in the former.

Before you start work, download (and, if need be, unarchive[a]) the file

`https://assets.phoo.org/COMS10015_2025_TB-4/csdsp/sheet/lab-10_q.tar.gz`

somewhere secure[b] in your file system; from here on, we assume `${ARCHIVE}` denotes a path to the resulting, unarchived content. The archive content is intended to act as a starting point for your work, and will be referred to in what follows.

[a]For example, you could 1) use `tar`, e.g., by issuing the command `tar xvfz lab-10_q.tar.gz` in a terminal window, 2) use `ark` directly: use the `Activities` desktop menu item, search for and execute `ark`, use the `Archive`→`Open` menu item to open `lab-10_q.tar.gz`, then extract the contents via the `Extract` button, or 3) use `ark` indirectly: use the `Activities` desktop menu item, search for and execute `dolphin`, right-click on `lab-10_q.tar.gz`, select `Open with`, select `ark`, then extract the contents via the `Extract` button.

[b]For example, the `Private` sub-directory within your home directory (which, by default, cannot be read by another user).

§1. C-class, or core questions

▷ **Q1[C].** In the lecture slot(s), we studied the design of an r-register counter machine (which is a specific type of register machine). For completeness, the design presented is reproduced in Figure 1; the associated instruction set, assuming $r = 4$, is reproduced in Figure 2. The goal of *this* question is to explore concepts around counter and register machines, using a practical, hands-on approach which complements the more theoretically focused lecture slot(s).

The archive includes a LogisimEvo project, and, specifically, an incomplete implementation of the design: beyond the fact that the implementation is superficially different from the design in certain, minor respects, note there is no *internal* implementation of the decoder component.

a Complete the counter machine implementation by implementing the decoder component.

b Use the completed counter machine implementation to execute the example program

$$\begin{array}{lclcl}
\mathsf{L}_0 : \textbf{if } \mathsf{R}_2 = 0 \textbf{ then goto } \mathsf{L}_5 \textbf{ else goto } \mathsf{L}_1 & \mapsto & 010100101_{(2)} & = & 0A5_{(16)} \\
\mathsf{L}_1 : \mathsf{R}_2 \leftarrow \mathsf{R}_2 - 1 \textbf{ then goto } \mathsf{L}_2 & \mapsto & 001100000_{(2)} & = & 060_{(16)} \\
\mathsf{L}_2 : \mathsf{R}_3 \leftarrow \mathsf{R}_3 + 1 \textbf{ then goto } \mathsf{L}_3 & \mapsto & 000110000_{(2)} & = & 030_{(16)} \\
\mathsf{L}_3 : \mathsf{R}_1 \leftarrow \mathsf{R}_1 + 1 \textbf{ then goto } \mathsf{L}_4 & \mapsto & 000010000_{(2)} & = & 010_{(16)} \\
\mathsf{L}_4 : \textbf{if } \mathsf{R}_0 = 0 \textbf{ then goto } \mathsf{L}_0 \textbf{ else goto } \mathsf{L}_5 & \mapsto & 010000000_{(2)} & = & 080_{(16)} \\
\mathsf{L}_5 : \textbf{if } \mathsf{R}_1 = 0 \textbf{ then goto } \mathsf{L}_9 \textbf{ else goto } \mathsf{L}_6 & \mapsto & 010011001_{(2)} & = & 099_{(16)} \\
\mathsf{L}_6 : \mathsf{R}_1 \leftarrow \mathsf{R}_1 - 1 \textbf{ then goto } \mathsf{L}_7 & \mapsto & 001010000_{(2)} & = & 050_{(16)} \\
\mathsf{L}_7 : \mathsf{R}_2 \leftarrow \mathsf{R}_2 + 1 \textbf{ then goto } \mathsf{L}_8 & \mapsto & 000100000_{(2)} & = & 020_{(16)} \\
\mathsf{L}_8 : \textbf{if } \mathsf{R}_0 = 0 \textbf{ then goto } \mathsf{L}_5 \textbf{ else goto } \mathsf{L}_9 & \mapsto & 010000101_{(2)} & = & 085_{(16)} \\
\mathsf{L}_9 : \text{halt} & \mapsto & 011000000_{(2)} & = & 0C0_{(16)}
\end{array}$$

which has been pre-loaded into the memory (i.e., ROM) component:

- Use the `Simulate`→`Reset Simulation` menu item to reset the counter machine, which sets each $\mathsf{R}_i = 0$ and $\mathsf{PC} = 0$.
- Use the poke[1] tool to set $\mathsf{R}_2 = 2$, noting that $\mathsf{PC} = 0$; you should be able to observe the initial configuration is $C_0 = (0, 0, 0, 2, 0)$.
- Use either the `Simulate`→`Ticks Enabled` menu item to *automatically* toggle the clock signals, or the `Simulate`→`Tick Once` menu item to *manually* toggle the clock signals. Doing so will force the counter machine to proceed through fetch-decode-execute cycles: you can concretely observe, e.g., each R_i and PC being updated in line with the (more abstract, "on paper") trace of computation presented in the lecture slot(s).
- Eventually the counter machine halts: you should be able to observe the final configuration is $C_{20} = (9, 0, 0, 2, 2)$.

[1]How to do so is not that obvious, but the idea is that you 1) select the poke tool, 2) click on the latch (to give it focus), then 3) type the value it should be updated to store.

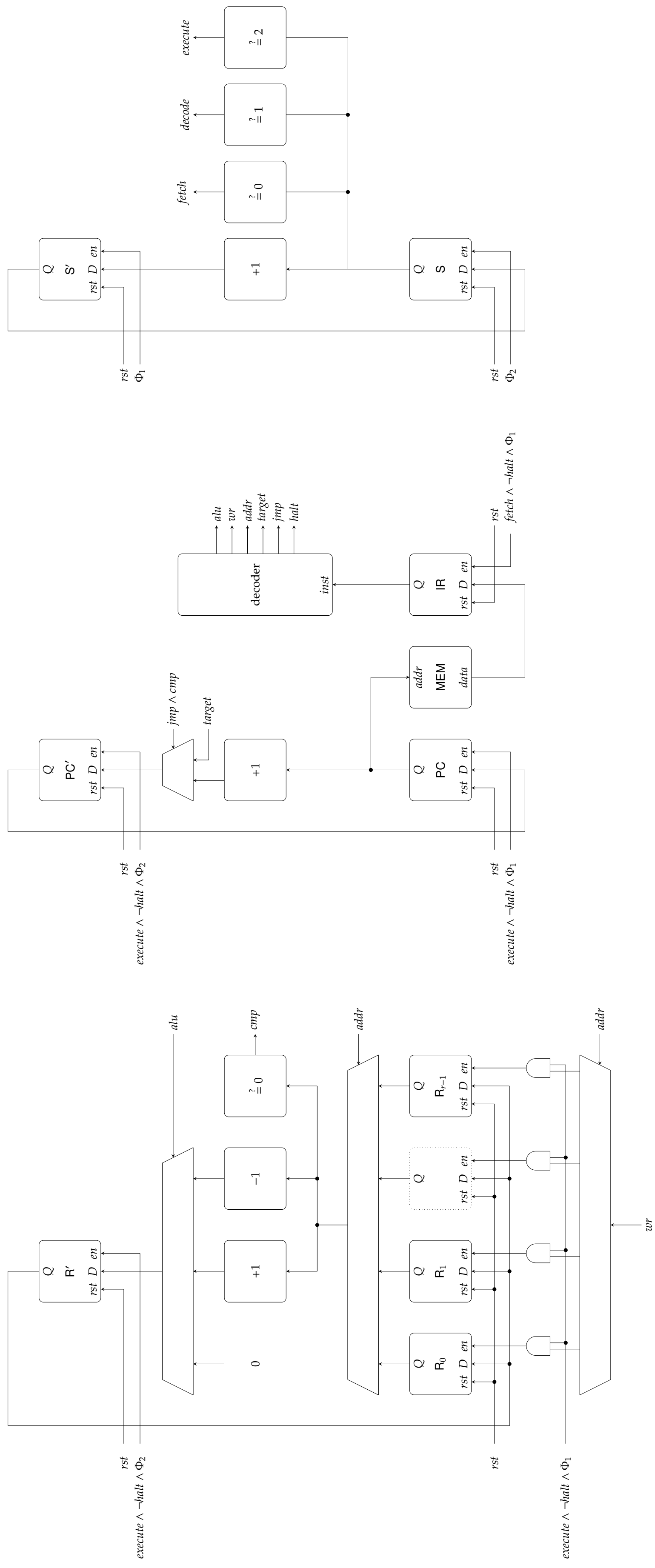

Figure 1: *A design for a r-register counter machine.*

§2. R-class, or revision questions

▷ **Q2[R].** There is a set of questions available at

`https://assets.phoo.org/COMS10015_2025_TB-4/csdsp/sheet/misc-revision_q.pdf`

Using pencil-and-paper, each asks you to solve a problem relating to Boolean algebra. There are too many for the lab. session(s) alone, but, in the longer term, the idea is simple: attempt to answer the questions, applying theory covered in the lecture(s) to do so, as a means of revising and thereby *ensuring* you understand the material.

§3. A-class, or additional questions

▷ **Q3[A].** Upgrade the counter machine implementation:

a Force execution to halt (rather than simply continuing, potentially meaning undefined or unintended behaviour) if/when decoding an instruction indicates it is invalid.

b Add support for a clear instruction, i.e., an instruction with the semantics

$$\mathsf{L}_i : \mathsf{R}_{addr} \leftarrow 0 \textbf{ then goto } \mathsf{L}_{i+1}.$$

c Add support for an unconditional branch instruction, i.e., an instruction with the semantics

$$\mathsf{L}_i : \textbf{goto } \mathsf{L}_{target}.$$

In each case, demonstrate that the upgrade functions correctly by writing and executing a short program.

▷ **Q4[A].** Write a program which adds the value in R_1 to that in R_2, storing the result in R_3.

		8 7 6	5 4	3 2 1 0
$\mathsf{L}_i : \mathsf{R}_{addr} \leftarrow \mathsf{R}_{addr} + 1$ **then goto** L_{i+1}	$\mapsto$	000	*addr*	0000
$\mathsf{L}_i : \mathsf{R}_{addr} \leftarrow \mathsf{R}_{addr} - 1$ **then goto** L_{i+1}	$\mapsto$	001	*addr*	0000
$\mathsf{L}_i :$ **if** $\mathsf{R}_{addr} = 0$ **then goto** L_{target} **else goto** L_{i+1}	$\mapsto$	010	*addr*	*target*
$\mathsf{L}_i :$ halt	$\mapsto$	011	00	0000

Figure 2: *An instruction set for a 4-register counter machine.*