

COMS10015 lab. worksheet #11: an EDVAC-like computer

Although some questions have a written solution below, for others it will be more useful to experiment in a hands-on manner (e.g., using a concrete implementation). The file

https://assets.phoo.org/COMS10015_2025_TB-4/csdsp/sheet/lab-11_s.tar.gz

supports such cases.

§1. C-class, or core questions

- ▷ **S1[C].** a. The archive provided includes source code for an example emulator implementation, the design of which is fairly simple: it uses the variable **MEM** to represent the memory **MEM**, **A** to represent the accumulator **A**, **PC** to represent the program counter **PC**, and an additional variable **IR** to represent the instruction register. When the emulator is executed the state is first “reset”, e.g., by setting $PC = 0$ and initialising the memory content. Then, a do loop emulates one fetch-decode-execute cycle per iteration:

- the next instruction is loaded from **MEM** using the address in **PC**, then **PC** is incremented,
- the instruction in **IR** is “decoded” into 1) an opcode held in **opcode**, and 2) an operand held in **n**,
- the instruction is executed by applying the semantics which opcode implies; if **opcode** = 21, for example, this means we need to apply the semantics $MEM[n] = A$, i.e., store the accumulator **A** into memory at address **n**.

Note that the loop terminates when a halt instruction is executed, and that the memory content is dumped before and after the loop (in the latter case, allowing inspection of any output).

- b. As an example, imagine $n = 8$ and $m = 13$: the array **X** has eight elements, which occupy **MEM**[13] to **MEM**[20] inclusive.

If n is known before we write the program, the easiest way to compute r is to use $n - 1$ addition instructions to accumulate the result into **A**, then store it into memory, e.g., at **MEM**[12]. This is demonstrated in Figure 1. However, if n is unknown before we write the program then we need another approach: we cannot unroll the loop as above, because we do not know how many addition instructions to use. The obvious approach is to use a loop, which iterates n times over a body of instructions each i -th of which adds X_i to **A**. This is demonstrated in Figure 2, which employs self-modifying code to implement the loop.

- instructions #0 to #2 load and accumulate X_i into **A**,
- instructions #3 to #5 self-modify instruction #0 which loads X_i : the change means that in the next iteration, the instruction will load X_{i+1} ,
- instructions #6 to #8 control the loop by loading the loop bound from **MEM**[11] and comparing it with **MEM**[0]: if the result is negative, further iterations are required,
- finally instruction #9 halts the program if enough iterations have been completed.

§2. R-class, or revision questions

- ▷ **S2[R].** There is a set of solutions available at

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

§3. A-class, or additional questions

- ▷ **S3[A].** There is no associated solution for this question, because it is somewhat open-ended with respect to 1) the goal or challenge presented, and/or 2) the assumptions and decisions you make, and therefore the design space of viable solutions.
- ▷ **S4[A].** There is no associated solution for this question, because it is somewhat open-ended with respect to 1) the goal or challenge presented, and/or 2) the assumptions and decisions you make, and therefore the design space of viable solutions.

Address	Content	Meaning
0	220013	$\mapsto A \leftarrow \text{MEM}[13]$
1	300014	$\mapsto A \leftarrow A + \text{MEM}[14]$
2	300015	$\mapsto A \leftarrow A + \text{MEM}[15]$
3	300016	$\mapsto A \leftarrow A + \text{MEM}[16]$
4	300017	$\mapsto A \leftarrow A + \text{MEM}[17]$
5	300018	$\mapsto A \leftarrow A + \text{MEM}[18]$
6	300019	$\mapsto A \leftarrow A + \text{MEM}[19]$
7	300020	$\mapsto A \leftarrow A + \text{MEM}[20]$
8	210012	$\mapsto \text{MEM}[12] \leftarrow A$
9	100000	$\mapsto \text{halt}$
10		
11		
12	0	
13	X[0]	
14	X[1]	
15	X[2]	
16	X[3]	
17	X[4]	
18	X[5]	
19	X[6]	
20	X[7]	

Figure 1: Computing the sum of all integers X_i for $0 \leq i < n$ using a straight-line sequence of additions.

Address	Content	Meaning
0	220013	$\mapsto A \leftarrow \text{MEM}[13]$
1	300012	$\mapsto A \leftarrow A + \text{MEM}[12]$
2	210012	$\mapsto \text{MEM}[12] \leftarrow A$
3	220000	$\mapsto A \leftarrow \text{MEM}[0]$
4	300010	$\mapsto A \leftarrow A + \text{MEM}[10]$
5	210000	$\mapsto \text{MEM}[0] \leftarrow A$
6	220000	$\mapsto A \leftarrow \text{MEM}[0]$
7	310011	$\mapsto A \leftarrow A - \text{MEM}[11]$
8	800000	$\mapsto \text{PC} \leftarrow 0 \text{ iff. } A \neq 0$
9	100000	$\mapsto \text{halt}$
10	1	
11	220021	
12	0	
13	X[0]	
14	X[1]	
15	X[2]	
16	X[3]	
17	X[4]	
18	X[5]	
19	X[6]	
20	X[7]	

Figure 2: Computing the sum of all integers X_i for $0 \leq i < n$ using a self-modifying loop structure.