Lab: valves and lamps, one bit each

1 Objective

Continue the project of *Lab files*. Until now, the program only watched the plant. In this lab it also **acts** on it: every tank has an inlet valve, and the control panel has two lamps. All of them are driven by the outputs of a digital module, and the program keeps a copy of those outputs in a single byte, the **valve register**, with one bit per valve or lamp. At the end of this lab you will have:

- printed a byte in binary, which `printf()` cannot do in C99;
- written the functions that set, clear, toggle and test bits with named masks, without touching the other bits of the register;
- seen in practice the traps of the slides: integer promotion, a shift that goes too far and the precedence of `&`.

> Estimated time: 60–90 minutes. Keep following the Barr-C style guide, and fix every warning as if it were an error.

2 The valve register

The register is a `uint8_t`. A bit at 1 means valve open or lamp on:

Bit	7	6	5	4	3	2	1	0
Output	ALARM	RUN	–	–	INLET 3	INLET 2	INLET 1	INLET 0

- **INLET 0–3**: inlet valve of tanks 0 to 3. At the start all of them are open (0Fh): the plant is filling.
- **RUN**: lamp that changes in every cycle (on, off, on…), to show that the program is alive.
- **ALARM**: lamp that is on while at least one sensor is faulty.
- Bits 5 and 4 are not connected: they must always stay at 0.

After every reading cycle, `main()` updates the register with these rules:

Status of the tank	Inlet valve of the tank
`LOW`	Open
`HIGH`	Close
`SENSOR_FAULT`	Close, and switch the ALARM lamp on
`NORMAL`	No change

The masks are already defined in `valves.h`:

```
#define VALVES_BIT(n)       (1U << (n))

#define VALVES_ALL_INLETS   (VALVES_BIT(3U) | VALVES_BIT(2U) \
                             | VALVES_BIT(1U) | VALVES_BIT(0U))   /* 0Fh */
#define VALVES_RUN_LAMP     VALVES_BIT(6U)                       /* 40h */
#define VALVES_ALARM_LAMP   VALVES_BIT(7U)                       /* 80h */
```

> In the text of this lab, hexadecimal numbers are written with the suffix h (0Fh). In the code they are always written as `0x0FU`.

3 Requirements

- *Lab files* finished: the folder `lab_files`, which builds without warnings and prints the expected output of that lab. If your version does not work, ask your instructor for the reference version.
- The file `lab_bits_given.zip`, given by your instructor.

4 Starting point

1. Copy the whole *Lab files* project into a new folder named `lab_bits`:

```
C:\...> cd %USERPROFILE%\c_projects
C:\...> xcopy lab_files lab_bits /E /I
C:\...> cd lab_bits
C:\...\lab_bits> mingw32-make clean
C:\...\lab_bits> del answers.txt
```

2. Extract `lab_bits_given.zip` into the project folder, replacing the files that already exist. It contains:
   - `include\valves.h` and `src\valves.c`: the module you will complete. `valves.c` has the six functions with an empty body.
   - `src\main.c` (V5.0): the new main program. Before the cycles it runs `test_valves()`, which applies a sequence of operations to a register and shows the result of each one. In every cycle it calls `control_valves()`, which applies the rules of the previous section, and prints the register.
   - `include\nudam_sim.h` and `src\nudam_sim.c` (V1.1): the simulated bus now has a second module, used only in *Going further*. Everything you did in the previous labs works as before.
3. Build and run with `mingw32-make run`, and type `6`. The column *result* of the valves test is empty, and every cycle shows `valves:` followed by `open: none`: the module `valves` is still empty. Read `control_valves()` in `main.c` before you go on: find which function of `valves`

is called for each rule of the table.

5 Part 1: print a byte in binary

Write `valves_print()`:

```
void valves_print(uint8_t value);
```

It prints the 8 bits of `value`, from bit 7 to bit 0, followed by a space and the value in hexadecimal between parentheses, without a new line. For 4Bh it prints `01001011 (4Bh)`. Use the idea of the activity of the slides: a mask that starts at 80h and moves one bit to the right in every iteration, until it becomes 0.

> To print one character, use `putchar('1')`. For the hexadecimal part, `printf(" (%02Xh)", (unsigned int) value)`: `%02X` prints at least two digits, in capital letters, with a leading 0 if needed.

Build and run, and type `6`. The column *result* now shows `00000000 (00h)` in every row (the other functions do nothing yet), and the initial register is printed correctly:

```
Initial valves: 00001111 (0Fh)  open: none
```

The register is 0Fh, but `open: none`: the function that tests a bit is still empty.

6 Part 2: set, clear, toggle and test bits

Complete the other five functions in `src/valves.c`. Each one is very short: one operator and one mask. The point is to choose the right operator, and to change **only** the bits of the mask.

1. `valves_inlet_mask()` returns the mask of the inlet valve of a tank: a byte with only the bit of that valve at 1 (tank 2 → 04h). If the tank has no inlet valve (`tank` is not smaller than `VALVES_NUM_INLETS`), it returns 0. Use `VALVES_BIT()`.
2. `valves_set()`, `valves_clear()` and `valves_toggle()` receive the **address** of the register and change the bits of `mask`: set them to 1, set them to 0, or invert them. Use the compound operators of the slides.
3. `valves_is_set()` receives the **value** of the register and returns `true` if the bit of `mask` is 1. Compare the result of the AND with 0, never with 1, and use parentheses as in the slides:

   ```
   return (0U != (value & mask));
   ```

4. Build and run, and type `6`. The valves test must show:

```
Valves test
  operation             result
  start                 00000000 (00h)
  set inlets 0 and 2    00000101 (05h)
  set ALARM             10000101 (85h)
  toggle RUN            11000101 (C5h)
  toggle RUN            10000101 (85h)
  clear inlet 0         10000100 (84h)
  is inlet 2 open?      yes
  is inlet 0 open?      no
  is ALARM on?          yes
  mask of inlet 3       00001000 (08h)
  mask of inlet 4       00000000 (00h)
```

Then check the line `valves:` of every cycle against the expected output of the section *Checklist and submission*.

5. **What does `~` really give?** In `valves_clear()`, add temporarily this line before the operation:

```
printf("~mask = %X\n", ~mask);
```

Build, run, and keep the first two values printed for question 1. Then remove the line.

6. **A tank that does not exist.** In `valves_inlet_mask()`, remove temporarily the check of `tank`, so that the function always returns the shifted bit. Build, run, and look at the row `mask of inlet 4`. Keep it for question 2, and restore the check.
7. **Precedence.** In `valves_is_set()`, remove temporarily the inner parentheses: `return (0U != value & mask);`. Build, and keep the warning. Run and look at the three questions of the test and at the column `open:` of every cycle. Keep what you see for question 3, and restore the parentheses.

7 Part 3: questions

Answer these questions in a plain text file named `answers.txt`, in the project folder. Short answers are fine, but justify them.

1. In Part 2, step 5, which values did `~mask` print? How many bits do they have, if `mask` is a `uint8_t`? Why is the register still correct after `*p_register &= ~mask;`? With the same `mask` = 80h, would `if (~mask == 0x7FU)` be true? Why?
2. In Part 2, step 6, what did `mask of inlet 4` show? In the real plant, which output would `main()` switch with that mask? What would happen with `tank` = 40? Did gcc warn you in either case?

3. In Part 2, step 7, which warning did gcc give? How does the compiler group `0U != value & mask`? Why does the test say that inlet 0 is open after `clear inlet 0`, and why does every cycle show `open: 0`?
4. Look at the cycles of the expected output. Why does the inlet valve of tank 2 stay closed in cycles 3 to 5, when its sensor works again? Why is the valve of tank 1 open in cycle 5, and the valves of tanks 0 and 3 closed? The RUN lamp is driven with a toggle, not with a set: what would an operator conclude if the lamp stopped changing?
5. `valves_set()` receives `uint8_t *p_register`, but `valves_is_set()` receives `uint8_t value`. Why? What would happen to the register of `main()` if `valves_set()` received the register by value?
6. In Java, the register would be a `byte`. Print the value CBh (cycle 2) as a Java `byte`: what does Java show, and why? Why does `uint8_t` not have this problem? What is the purpose of the expression `(b & 0xFF)`, very common in Java code that handles bytes?

8 Going further (optional)

1. **Four statuses in one byte.** The status of a tank has 4 values, so it fits in 2 bits, and the status of the 4 tanks fits in one byte: tank 0 in bits 1..0, tank 1 in bits 3..2, and so on. Add two functions to the module `plant`:

```
#define PLANT_STATUS_BITS  2U
#define PLANT_STATUS_MASK  0x03U
#define PLANT_MAX_PACKED   4U

uint8_t       plant_pack_status(const tank_state_t *p_state,
                                uint8_t num_tanks);
tank_status_t plant_unpack_status(uint8_t packed, uint8_t tank);
```

   - `plant_pack_status()` inserts the status of each tank (at most `PLANT_MAX_PACKED`) in its field, with a shift of `tank * PLANT_STATUS_BITS`.
   - `plant_unpack_status()` extracts the field of one tank: shift, then AND with the mask.
   - In `main()`, after the line `valves:`, print the packed byte with `valves_print()` and the status of tank 2 extracted from it. With 6 cycles, the bytes are 55h, 55h, 75h, 51h, 51h and 92h, and the status of tank 2 is `SENSOR FAULT` only in cycle 2.

   Decode 92h by hand. Why would a fifth tank not fit? Which value of `tank_status_t` would break the packing if someone added a fifth status to the `enum`?
2. **Send the register to the module.** The simulated bus has now a digital output module (ND-

6050) at address 02 (`NUDAM_SIM_DO_ADDRESS`). The command `#AA00DD<CR>` sets its 8 outputs to the bits of the byte DD, written with two hexadecimal digits, and the module answers `><CR>`. Add to the module `valves`:

```
int valves_build_command(char *p_command, size_t command_size,
                         uint8_t address, uint8_t value);
```

It builds the command with one `snprintf()`, as `nudam_build_read()` in *Lab strings*, and returns -1 if it does not fit. In `main()`, after the line `valves:`, build the command, send it with `nudam_sim_transaction()`, check that the response is exactly `">\r"` with `strcmp()`, and print the dialogue with `nudam_sim_print()` and the outputs of the module with `nudam_sim_get_outputs()`. With 6 cycles, cycle 2 must show:

```
  valves: 11001011 (CBh)  open: 0 1 3  RUN  ALARM
  bus: #0200CB<CR>  ><CR>  outputs CBh
```

What happens if you send the command to address 03? Why does the program keep its own copy of the register, instead of reading the outputs back from the module before every change?

9 Checklist and submission

Your lab is complete if:

- ☐ `mingw32-make clean` followed by `mingw32-make` builds the project without any warning.
- ☐ `main.c`, `valves.h`, `nudam_sim`, `sensor`, `area`, `volume`, the `Makefile` and the files in `.vscode` have not been modified.
- ☐ `valves.c` uses the masks of `valves.h` and `VALVES_BIT()`: no magic numbers like `0x80`.
- ☐ `valves_set()`, `valves_clear()` and `valves_toggle()` use a compound operator and change only the bits of the mask.
- ☐ `valves_is_set()` compares the result of the AND with 0, with parentheses.
- ☐ `answers.txt` answers the six questions.
- ☐ With 6 cycles, the program prints the expected output below.

The final output of the program, typing `6`, must be:

```
Plant configuration: 4 tanks
  tank  radius(m)  height(m)  low(%)  high(%)  capacity(L)
     0       0.50       2.00      20       90       1570.8
     1       0.75       3.00      15       85       5301.4
     2       0.40       1.50      25       90        754.0
     3       1.00       2.50      10       95       7854.0
  plant capacity: 15480.2 L

Valves test
  operation             result
  start                 00000000 (00h)
  set inlets 0 and 2    00000101 (05h)
  set ALARM             10000101 (85h)
  toggle RUN            11000101 (C5h)
  toggle RUN            10000101 (85h)
  clear inlet 0         10000100 (84h)
  is inlet 2 open?      yes
  is inlet 0 open?      no
  is ALARM on?          yes
  mask of inlet 3       00001000 (08h)
  mask of inlet 4       00000000 (00h)

Number of cycles (1-6): 6

Initial valves: 00001111 (0Fh)  open: 0 1 2 3

Cycle 0
  tank    raw  level(m)  fill(%)  volume(L)  status
     0   1200     0.586     29.3      460.3  NORMAL
     1   2000     1.465     48.8     2589.2  NORMAL
     2   2048     0.750     50.0      377.1  NORMAL
     3   3500     2.137     85.5     6712.8  NORMAL
  plant volume: 10139.4 L (65.5 % of capacity), 0 sensor fault(s)
  valves: 01001111 (4Fh)  open: 0 1 2 3  RUN

Cycle 1
  tank    raw  level(m)  fill(%)  volume(L)  status
     0   1800     0.879     44.0      690.5  NORMAL
     1   1500     1.099     36.6     1941.9  NORMAL
     2   2050     0.751     50.1      377.5  NORMAL
     3   3550     2.167     86.7     6808.7  NORMAL
  plant volume: 9818.5 L (63.4 % of capacity), 0 sensor fault(s)
  valves: 00001111 (0Fh)  open: 0 1 2 3

Cycle 2
  tank    raw  level(m)  fill(%)  volume(L)  status
     0   2400     1.172     58.6      920.6  NORMAL
```

```
     1   1000     0.733     24.4     1294.6  NORMAL
     2  65535         -        -          -  SENSOR FAULT
     3   3600     2.198     87.9     6904.6  NORMAL
  plant volume: 9119.8 L (58.9 % of capacity), 1 sensor fault(s)
  valves: 11001011 (CBh)  open: 0 1 3  RUN  ALARM

Cycle 3
  tank    raw  level(m)  fill(%)  volume(L)  status
     0   3000     1.465     73.3     1150.8  NORMAL
     1    600     0.440     14.7      776.8  LOW
     2   2047     0.750     50.0      376.9  NORMAL
     3   3700     2.259     90.4     7096.4  NORMAL
  plant volume: 9400.8 L (60.7 % of capacity), 0 sensor fault(s)
  valves: 00001011 (0Bh)  open: 0 1 3

Cycle 4
  tank    raw  level(m)  fill(%)  volume(L)  status
     0   3600     1.758     87.9     1380.9  NORMAL
     1    400     0.293      9.8      517.8  LOW
     2   2049     0.751     50.0      377.3  NORMAL
     3   3850     2.350     94.0     7384.1  NORMAL
  plant volume: 9660.1 L (62.4 % of capacity), 0 sensor fault(s)
  valves: 01001011 (4Bh)  open: 0 1 3  RUN

Cycle 5
  tank    raw  level(m)  fill(%)  volume(L)  status
     0   3800     1.856     92.8     1457.6  HIGH
     1    300     0.220      7.3      388.4  LOW
     2   2048     0.750     50.0      377.1  NORMAL
     3   3950     2.411     96.5     7575.9  HIGH
  plant volume: 9799.0 L (63.3 % of capacity), 0 sensor fault(s)
  valves: 00000010 (02h)  open: 1

History saved in history.csv

Volume history (L), valid readings read from history.csv
  tank  samples        min        max       mean
     0        6      460.3     1457.6     1010.1
     1        6      388.4     2589.2     1251.5
     2        5      376.9      377.5      377.2
     3        6     6712.8     7575.9     7080.4
```

Submit the folders `src` and `include` and the file `answers.txt`, as indicated by your instructor. Do not submit the `build` folder or `history.csv`: run `mingw32-make clean` first. Keep the project: the next labs start from it.

10 Troubleshooting

- `implicit declaration of function 'valves_print'`**.** `main.c` includes `valves.h`: check that the file is in the folder `include`, and that `valves.c` is in `src`.
- **The binary has 7 digits, or 9.** Your mask does not start at 80h, or the loop does not stop when the mask becomes 0.
- **The hexadecimal shows** `4b` **instead of** `4B`**, or** `F` **instead of** `0F`**.** Use `%02X`: capital `X`, width 2, filled with 0.
- `set ALARM` **also clears inlets 0 and 2.** You used `=` instead of `|=`: the register takes the value of the mask, and the other bits are lost.
- `clear inlet 0` **gives 01h or 00h.** For 01h, you wrote `&= mask` and forgot the `~`: the AND keeps the bit you wanted to clear and erases the others. For 00h, you wrote `&= !mask`: `!` is the logical NOT, which gives 0 or 1, not the bitwise complement `~`. gcc does not warn in either case.
- `is inlet 2 open?` **and** `is ALARM on?` **say** `no`**.** You compared the result of the AND with 1: for inlet 2, `value & mask` is 04h, and for the ALARM lamp 80h, never 1. Compare with 0.
- **Bits 5 and 4 appear at 1.** `valves_inlet_mask()` does not check the tank, or a function changes bits outside the mask.