CT Lab: Data Transfer Commands

1 Introduction

In this Lab you will write your first assembly program. The program reads data from the DIP-switches (of the target hardware) and stores them in a table. The table elements can be selected through the DIP-switches and are displayed on the 7-segment display.

2 Learning Objectives

Using assembly language you are able

- to input and output data on the CT Board
- to allocate memory for tables
- to access tables with byte elements as well as tables with half word elements

3 Task 1 – Input and Output of Table Values

Write an assembly program: Upon every press of button T0 the program shall read a 4 bit input index (S11 to S8) and an 8 bit input value (S7 to S0) from the DIP-switches. The input value shall be stored in a table in RAM at the position provided by the input index and displayed on LED7 to LED0 (See Figure 1). In addition the input index shall be displayed on LED11 to LED8 for debugging

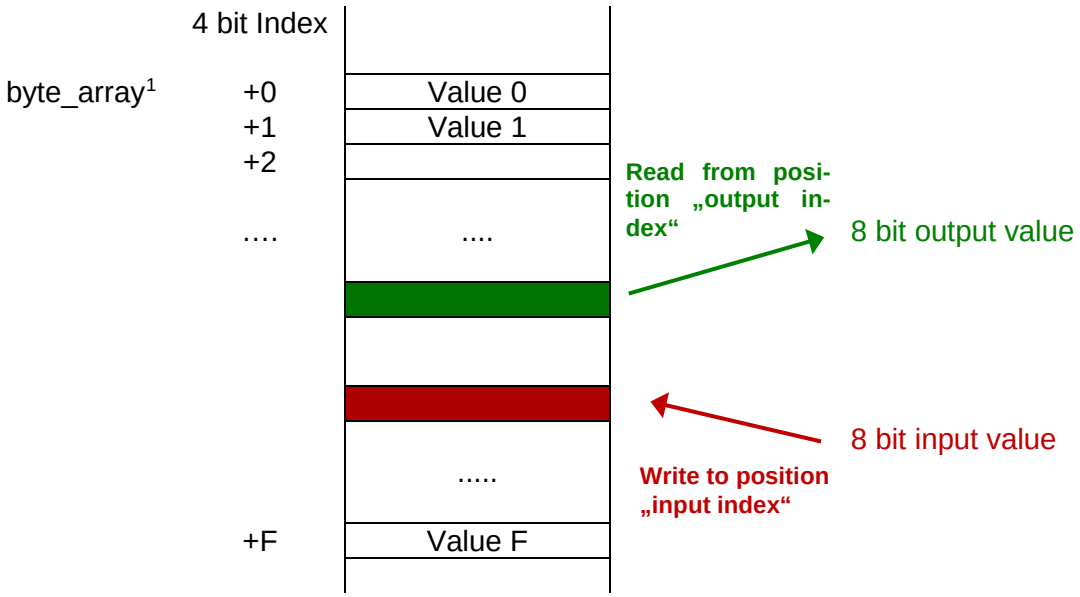

Figure 1: Writing and reading values to/from a table in memory

[1] The start address (i.e. the memory address of the element with index 0) has the name (label) **`byte_array`**.

Additionally a table value shall be queried on each turn. The desired output index can be selected through the DIP-switches S27 to S24 and shall be displayed on LED27 to LED24. The corresponding table value shall be displayed on LED23 to LED16. Figure 2 shows how the different values shall be read from or written to the target hardware.

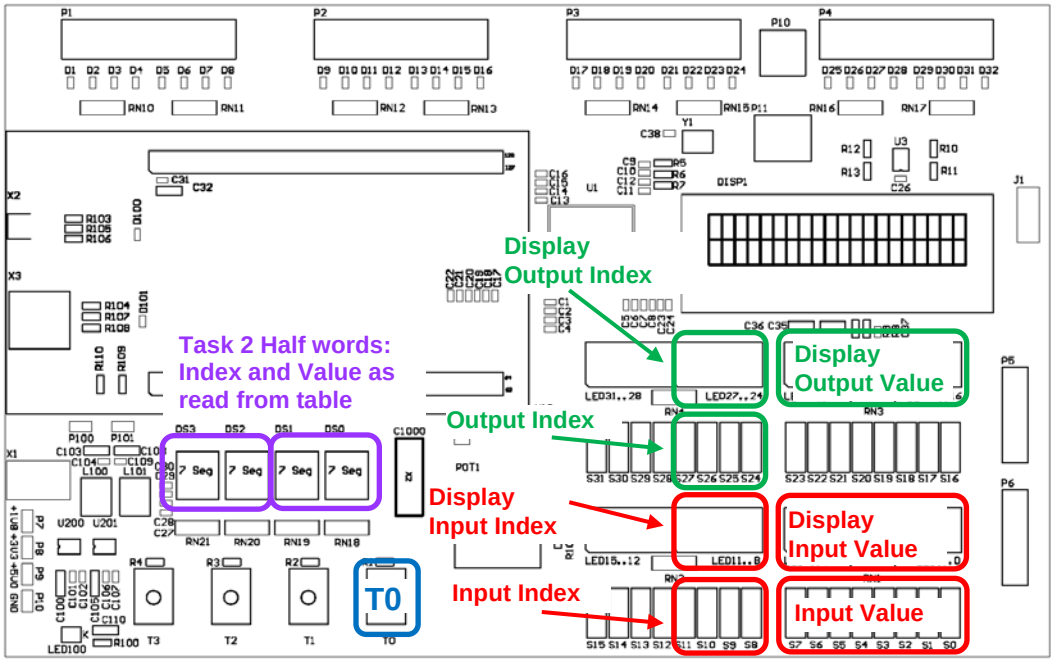

Figure 2: In- and output of table values

3.1 Given Program Frame

Use the given program frame for this lab with the assembly file *table.s*. It contains a while loop and the subprogram *waitForKey*, which gets called with the assembly command **BL**. The subprogram waits until the user presses button T0 and then continues. Expand the program step by step at the marked positions.

3.2 Read Input Index and Input Value

In this step you read the input index and the input value, both of which you need for writing into the table. Read the values for the input index from the DIP-switches S11 to S8 and the input value from S7 to S0. You can directly read a byte from the corresponding addresses. You have to mask the upper bits of the input index (i.e. clear the bits to 0) with the following instructions (**BITMASK_LOWER_NIBBLE** is already defined in the program frame).

```
LDR     R7, =BITMASK_LOWER_NIBBLE
ANDS    R1, R1, R7
```

Display the values read on LED7 to LED0 and LED11 to LED8. If a bit is '1' the corresponding LED shall be on. Check whether the masking of the input index works as intended.

3.3 Store in Table

Allocate memory in the data section for a table with 16 byte values (assembly directive **SPACE**). Be aware of the fact, that the elements are not initialized when the program starts. Store the previously read input values at the correct position in the table (input index).

Test your program with different input indexes and values by verifying the content of the table through the debugger (See Figure 3).

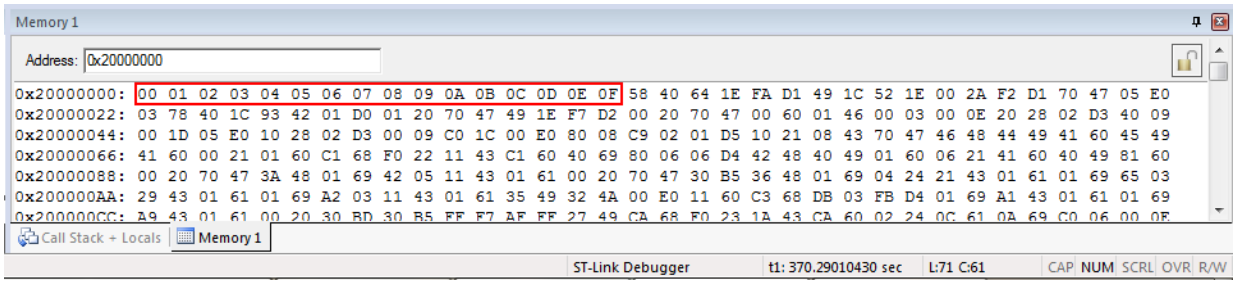

Figure 3: uVision Debugger Memory View

- The Memory view is only refreshed if you re-enter the address and press enter.

3.4 Read and Display the Output Index

Read the output index from the DIP-switches S27 to S24. You also have to mask the upper 4 bits.

Display the index on LED27 to LED24. Verify the correct behavior by trying different positions of the DIP-switches.

- For every change on the DIP-switches the button T0 has to be pressed to display the changes.

3.5 Display the Selected Table Value

Use the output index to access the table and display the corresponding value on LED23 to LED16.

Verify the correct function of your program by filling the table with defined values and reading them afterwards.

4 Task 2 – Variant with Half Word Table

Create a new version of your program that uses a table of half word elements instead of bytes. Additionally to the input value (stored in the least significant byte), the input index shall be stored in the most significant byte of the table element.

Additionally to the existing display options, use the 7-segment display to simultaneously output the contents of the selected half-word in the table: index (DS3 to DS2) and corresponding value (DS1 to DS0).

Make a copy of your existing (and working) project and expand on this copy.

- Since a half word contains two bytes, you have to multiply your table index by two. You can use the "logic shift left" instruction to achieve this.
  `LSLS R1, R1, #1`
- To display the values on the 7-segment display, use the addresses, which are described on the CT-Wiki Page 7-Segment Binary Interface. This way you can directly display the values in hexadecimal representation.

5 Task 3 – Implementation in C (optional)

Write a program in C, which stores values in a table. Define a global table of **`uint8_t`** elements. Bit masks and access addresses shall be defined using **`#define`**.

```
#define BITMASK_KEY_T0 0x01
uint8_t byteTable[16];

void main(void) {
....
}
```

6 Grading

The working programs have to be presented to the lecturer. The student has to understand the solution / source code and has to be able to explain it to the lecturer.

Task	Criteria	Weight
1	The program meets the requirements.	2/4
2	The program meets the requirements. You can explain which changes you made and why you made them.	2/4