

CMPE-250 Assembly and Embedded Programming

Laboratory Exercise Five

Polled Serial I/O

By submitting this report, I attest that its contents are wholly my individual writing about this exercise and that they reflect the submitted code. I further acknowledge that permitted collaboration for this exercise consists only of discussions of concepts with course staff and fellow students. Other than code provided by the instructor for this exercise, all code was developed by me.

Shubhang Mehrotra
Performed 25th February 2021.
Submitted 4th March 2021.

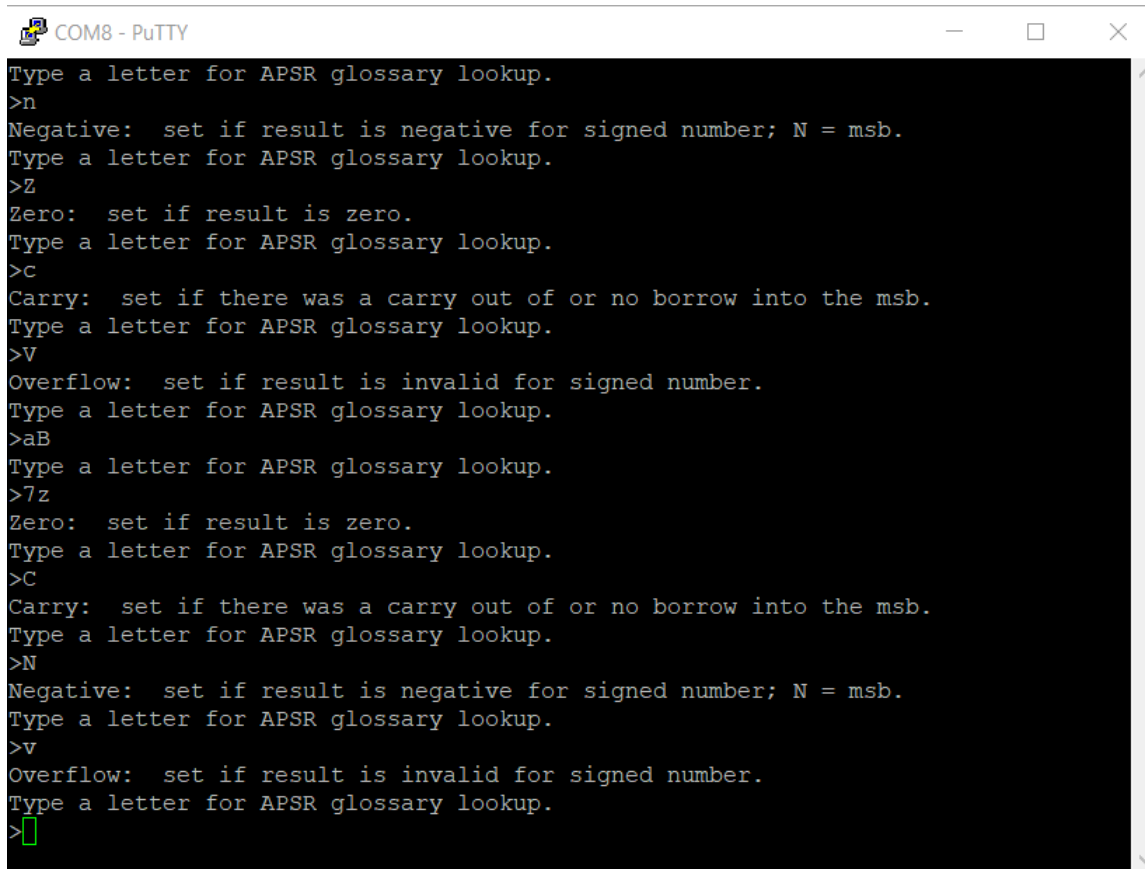
Lab Section 1
Instructor: Muhammad Shaaban
TA: Anthony Bacchetta
Aidan Trabuco
Sam Myers
Payton Burak

Lecture Section 1
Lecture Instructor: Dr. Roy Melton

Image

The activity introduced the performer to KL05Z Freedom Board and the PuTTY environment to interact with the board. This enabled the performer to test out the UART Polling in an intuitive manner. The program initialized connection with the Board via USB COM Port (COM8 in this case,) at 9600 Baud rate using PuTTY and its “Serial” connectivity functionality.

The code for serialized polling was written and downloaded to the KL05Z board using the Keil IDE. It was then tested against the input: “nZcVaB↵7zCNv ”, the result of the run is shown in Figure 1.



```
COM8 - PuTTY
Type a letter for APSR glossary lookup.
>n
Negative: set if result is negative for signed number; N = msb.
Type a letter for APSR glossary lookup.
>Z
Zero: set if result is zero.
Type a letter for APSR glossary lookup.
>c
Carry: set if there was a carry out of or no borrow into the msb.
Type a letter for APSR glossary lookup.
>V
Overflow: set if result is invalid for signed number.
Type a letter for APSR glossary lookup.
>aB
Type a letter for APSR glossary lookup.
>7z
Zero: set if result is zero.
Type a letter for APSR glossary lookup.
>C
Carry: set if there was a carry out of or no borrow into the msb.
Type a letter for APSR glossary lookup.
>N
Negative: set if result is negative for signed number; N = msb.
Type a letter for APSR glossary lookup.
>v
Overflow: set if result is invalid for signed number.
Type a letter for APSR glossary lookup.
>
```

Figure 1. Result

Figure 1 shows a PuTTY Serial window with the results of the various inputs. The code written performed the required function only when one of the keywords– “C”, “N”, “Z” or “V”, was read. It ignored all other inputs, except the “Enter,” upon which it output a carriage return accompanied with a Line feed character, to move the cursor to the next line. It is evident from Figure 1 that the code was successful in its implementation, as it ignores all other inputs and outputs the desired strings when it receives one of the keywords.

Question: Memory Ranges

It is an essential part of assembly language to understand the specifics about your code to improve the efficiency. To know how the computer is storing and interacting with the code, one can look at the generated Listing and Map files. The Listing file shows precisely how the assembler translates the source file into machine code while the Map file contains the static symbols and their relative addresses to each other. Looking at the two can give information about the exact memory locations where the code is stored and executed at in both ROM and RAM. More information can be extracted from them depending upon how they are setup. In advance usages, they can help identify crashes and/or points of optimization in the code.

For the Lab Activity, the Listing file and the Map file were generated and were analyzed for the memory addresses of the items listed in Table 1, which lists the start addresses, end addresses and the size occupied by the item in the memory.

Table 1. Memory Ranges

Object	Memory Address Start	Memory Address End (Start + Size - 1)	Size
MyCode AREA	0x00000290	0x000003AB	0x0000011C
Executable code in MyCode AREA	0x00000290	0x00000379	0x000000EA
main program	0x00000290	0x000002E9	0x0000005A
GetChar Subroutine	0x00000356	0x00000367	0x00000012
PutChar Subroutine	0x00000368	0x0000037B	0x00000012

The Start Address for “MyCode AREA” was obtained from the Map file. Under the section marked “Memory Map of the Image,” is a table which lists the execution address of each entity in the source file. Upon inspection of the table, the start addresses, and the size can be obtained. The End address can be obtained by adding the start address and the size and subtracting 1.

The Listing file contains the offset with respect to the execution addresses and can be analyzed to obtain the addresses for the subroutines and the executable area of the code. For instance, the offset listed for the GetChar subroutine is “0x000000C6.” Adding that offset to the execution address of “0x00000290,” we obtain the start address of GetChar subroutine as “0x00000356.” Similarly, the address for PutChar Subroutine is calculated from its listed offset of “0x000000D8.” The ending addresses can be obtained as mentioned above using the size and the starting address.