Experiment 3: Advanced Assembly Coding

10-12.10.2022

Res. Asst. Altay Ünal
unal21@itu.edu.tr

Ra ra Rasputin
Lover of the Russian queen
There was a cat that really was gone
Ra ra Rasputin
Russia's greatest love machine
It was a shame how he carried on

Boney M

1 Introduction

In this experiment, more complex Assembly programs will be written. As we are getting more familiar with the MSP430 board and its syntax, we will try to implement more complicated algorithms and observe the results of the algorithms using the core registers inside the MSP430 board. Before the lab, it is suggested that you study the given algorithms in order to implement without any problem.

2 Russian Peasant Division

For modulus operation we will perform some steps from Russian Peasant Division which is very useful for binary systems. RPD consists of the following steps for dividend A and divisor B:

- Create variables C and D, initialize them with B and A's values respectively.
- While C is not greater than A/2, multiply it by 2.
- While B is not greater than D:
  - Subtract C from D wherever D is greater than or equal to C.
  - Divide C by 2.

- The final value in D gives you the remainder. If you subtract this number from A, this new A can be represented by summation of previously used C values. After finding the proper C values, you can find their appearance iteration in the first loop, raise them to 2nd power, and sum them.

An example implementation is given in the table below.

A	B	C	D	Explanation
151	8	8	151	–
151	8	**16**	151	Multiplying C by 2
151	8	**32**	151	–
151	8	**64**	151	–
151	8	**128**	151	Now it's greater than A/2
151	8	**64**	**23**	$151 - 128 = 23$
151	8	**32**	23	–
151	8	**16**	23	–
151	8	**8**	**7**	$23 - 16 = 7$
144	8	8	7	$151 - 7 = 144$
$\mathbf{128 + 16}$	8	8	7	$144 = 128 + 16$
				$128 = C[4], 16 = C[1]$
				$2^4 + 2^1 = 18$

Write the Assembly code that calculate modulus.

3 Hashing

Write the assembly code to hash your student IDs inside the memory. To allocate space in memory, you can use ".space" directive.

```
        .data
hash    .space 58
```

Here, 58 bytes of memory are allocated for the hash table to keep your hashed IDs. Inside the code part, you can use #hash to reach the address of the first element.

```
        mov.w    #hash, R10; First address for the hash table is
            assigned to R10
        ...
        mov.w   R5, 0(R10); Some value is copied to that address
```

Let's say that your student ID's are represented as ABCDEFGHI. Your ID is split into 3 different values, ABC, DEF and GHI, respectively. First, you need to allocate some memory other than your hash table for your split ID values so that you can enter your split ID values into the memory. Then, by using the given hash function, you need to find the hash values of your ID values obtained from the initially configured space and keep the split ID inside the hash table. The hash function is dividing by 29 and your remainder is the place of your split ID values. According to the found hash values, place your split ID values in the hash table. For example, one of your split ID values is 111 and its hash value is 24, then it must be placed into the $\#hash + 24th$ location in the table.

However, there may be collisions during the placement of your ID values. In order to solve the collisions, you are expected to use linear probing. In linear probing, if the determined location in the hash table is full, program checks for the next possible location. For example, your next split ID value is 024 but 24th location is allocated for 111 from previous example. So, 024 must be placed into the $\#hash + 25th$ location in the table.

Hint: *Enter your split ID values to the memory by hand using Memory Browser.*