Exercise Assignment Week 03: Moving Data Around (NASM)

Due Monday October 3 midnight

**Reference/Review:**

Data Section declarations

|  |  |
| --- | --- |
| DB | Define byte (8 bits) |
| DW | Define word (2 bytes – 16 bits) |
| DD | Define double (4 bytes – 32 bits) |

Most Common Modes for addressing data

|  |  |
| --- | --- |
| Register Direct addressing | Register contains the data |
| Register Indirect addressing | Register contains a memory location of where the data is |
| Immediate | Data is provided as part of the instruction |
| Memory Direct addressing | Memory address contains the data |

**Tasks:**

Create ***lastname\_firstname\_A3*** folder, and put the following exercises inside this folder. Start a **makefile**, where you can add each exercise as you work along. From inside the folder, the command “make all”, for example, will run all the exercises. (Last week you were given a template in D2L).

1. Please refer to slide #3 in this week’s PPT notes. There are 13 lines of data declarations on this slide. On a piece of paper, write out what you predict will be in each byte location (you don’t need to hand this in). Your table may look something like this (leftmost column is byte address, other columns are data):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 00 |  |  |  |  |
| 04 |  |  |  |  |
| 08 |  |  |  |  |
| 0C |  |  |  |  |
| 10 |  |  |  |  |
| 14 |  |  |  |  |
| 18 |  |  |  |  |
| 1C |  |  |  |  |
| 20 |  |  |  |  |
| 24 |  |  |  |  |
| 28 |  |  |  |  |
| 2C |  |  |  |  |
| 30 |  |  |  |  |
| 34 |  |  |  |  |
| 38 |  |  |  |  |

Now, type the data declaration into an assembly program yourself (***task1.asm***), compile and generate a “list” file. Examine the listing file. Verify your answers above.

1. Study the program “**first.asm**” provided for you. Compile, link, and run this program (together with provided files **driver.c, asm\_io.asm, asm\_io.inc**). Now, modify the program so that is also calculates some other formula/operations using the two inputs. For example, calculate (A+B)\*A, or (B^2-2A). You can choose to do any formula. It should involve at least 2 operations. State what the formula is in your comments, and also in the console output. Document as needed. Name it ***task2.asm***

NOTES:

* The program uses a .bss section for declaring data that is uninitialized
* Before and after the actual functional code, some “housekeeping” instructions are used to setup and cleanup the function call (ENTER, PUSHA, POPA, LEAVE). These instructions push and restores register values on the stack. More on the stack later. You can keep these lines of code as a standard template for assembly functions used by C.

Optional Challenge: Calculate also, the factorial of one the provided numbers. Eg. 5! = 5\*4\*3\*2\*1 Are there any limitations in your code?

1. Study the program “**array.asm**” provided for you. Compile, link, and run this program (together with provided files **driver.c, asm\_io.asm, asm\_io.inc**). Now, modify this program so that so that it uses register indirect with offset to access the table. Comment as needed. Name it ***task3.asm***

Optional Challenge: Instead of the “sum”, calculate the “average” in Exercise #3.

1. Study the program “**loop.asm**” provided for you. Compile, link, and run this program (together with provided files **driver.c, asm\_io.asm, asm\_io.inc**). Now, modify this program so that so that it will:

* Declare loop size as a constant with %assign
* Declare data initialized to be all 0’s using the “times” instruction
  + Example: *mytable: times 5 DW 0 ;allocates 5 words initialized to 0*
* In the code area:
  + Using a loop, initialize data with contents 1,2,3,4, ...
  + Using a loop, sum and print result (basically originally code)
  + Also, calculate product of all the numbers, and print result

Test your program with various loop sizes. Comment as needed. Name it ***task4.asm***

Optional: In the C driver, print out the return value of asm\_main() call. What do you see?

**Marking:**

|  |  |
| --- | --- |
|  | Max |
| Completeness | 2 |
| Correctness | 2 |
| Files are well organized | 2 |
| Code is well commented and documented | 2 |
| Proper file naming conventions | 2 |