

Group Members: Nachiket Kelkar & Puneet Bansal

Date: 7th October 2018

Project 1: Interactive Memory Manipulation

Principles of Embedded Software

Effort required						
Milestone	Task	Best	Likely	Worst	Estimated effort	Date
	Develop and Document the project plan	1.5	1.5	2	1.61	18/09/2018
Submit the project plan						
	Setting up the build environment	0.25	0.25	1	0.42	
	Setup the project GIT repository	0.25	0.25	0.5	0.31	
	Initial makefile, main() and framework	5	5	6	5.22	
	Basic command line prompt and parse	2	2	3	2.22	
	Build basic help function	1.5	1.5	3	1.83	21/09/2018
Command Prompt with the help functionality						
	Allocate Memory	2	2	4	2.44	
	Free Memory	1	2	3	2.00	
	Display Memory	2	2	4	2.44	
	Write Memory	2	2	4	2.44	
	Invert Momory bits	1	1.5	2	1.50	24/09/2018
Memory Operations						
	Measuring execution time	2	2.5	3	2.50	
	Add bound checks and warnings to functions	2	2.5	3	2.50	
	Develop psudo-random number generation algorithm	1	1.5	2	1.50	26/09/2018
Timed Operations and Bound Checks						
	Write pattern	3	4	6	4.22	
	Verify Pattern	3	4	6	4.22	29/09/2018
Write and Verify with bound check and time measurements						
	Makefile of the entire project	2	3.5	6	3.72	
	Rigorous testing of each functionality on command line	1	1	5	1.89	
	Automated tesing development	6	6	6	6.00	03/10/2018
Feature Complete						
Final report		8	9	10	9.00	
Total Hours		46.5	54	79.5	58	10/10/2018

Principles of Embedded Software

Project 1: Interactive Memory Manipulation

SYNTAX DOCUMENTATION

In the program we have implemented an interactive interface, wherein the user only types the operation/command they need to perform and the program asks for additional information as and when it is required.

Several commands that are implemented for this command line utility are:

- | | |
|-----------------------|--|
| help | - displays the supported commands with the description of each command. |
| allocate | - allocates memory to be used by user in chunks of 32-bit block. Number of such 32-bit blocks are specified by the user. |
| free | - free's the allocated block of memory if any. |
| write | - writes a 32-bit value provided by the user on the address specified. The address can be specified by writing the entire address or giving an offset from the starting address. |
| disp | - displays the content of the number of memory blocks provided by the user. |
| invert | - inverts the contents of the memory on the specified address given by the user. |
| write_pattern | - write the 32-bit pattern to the several address locations given by the user by specifying address or offset and number of blocks to write based on the seed value provided. |
| verify_pattern | - verifies the content of the memory if it contains correct values generated by the write_pattern function based on the seed value provided by the user. |
| ext | - it is the exit command which exit's the command line utility. |

1. Command 'help'

The syntax of help command is 'help'

The help command lists all actions that are available to be taken on the data and description of each function in short for a quick start of command line utility.

help command in action



```
Welcome to the command line
Type Help to see the commands
>>help
1)allocate      - allocate 32-bit blocks of memory
2)free         - free the block of allocated memory
3)disp         - display the 32-bit blocks of memory
4)write        - write the 32-bit word to the memory location
5)invert       - invert the contents of the memory in memory location
6)write_pattern - write a random pattern to the blocks of memory
7)verify_pattern - verify the contents of memory written by write_pattern
8)ext          - exit the utility
```

2. Command 'allocate'

The syntax of allocate command is 'allocate'

When user invokes allocate command and enters the number of 32-bit memory blocks to be allocated then it allocates the requested number of blocks if the contiguous memory location of the specified size is available.

Principles of Embedded Software

Project 1: Interactive Memory Manipulation

allocate command in action ↓

```
>>allocate  
  
State the number of 32-bits block of memory to be allocated:-  
>>9  
  
****MEMORY ALLOCATED****
```

3. Command 'free'

The syntax of free command is 'free'

The free command is used in order to free memory which has been assigned to the user. This command can be used if the memory is already allocated for the use of user. No input is required from the user after invoking the command.

free command in action ↓

```
Type the function you would like to perform or type help to see the menu  
>>free  
The 0x5584d152da80 address and following addresses are free
```

4. Command 'write'

The syntax of write command is 'write'

When the user invokes the write function the function returns all the available address user can use to write the content. Available addresses are address that are allocated by the user using allocate command. It gives the option to enter the address manually or provide an offset from the start of allocated address. Once address is entered the user is asked to provide the 32-bit value in hex format to be written to the memory location selected and writes the corresponding data to the memory location.

write command in action ↓

```
Enter the function you like to perform or type help to see the menu  
>>write  
  
The available addresses on which you can write are  
  
0x5584d152da80  
0x5584d152da84  
0x5584d152da88  
0x5584d152da8c  
0x5584d152da90  
0x5584d152da94  
0x5584d152da98  
0x5584d152da9c  
0x5584d152daa0  
  
Do you wish to:  
(1)Type the address on which you want to write  
(2)Give an offset from 0x5584d152da80  
>>2  
  
Enter the offset from 0x5584d152da80  
>>0  
  
Enter the 32 bit hex value you want to write  
>>0xe1e1e1e1  
  
**Value Written**
```

Principles of Embedded Software

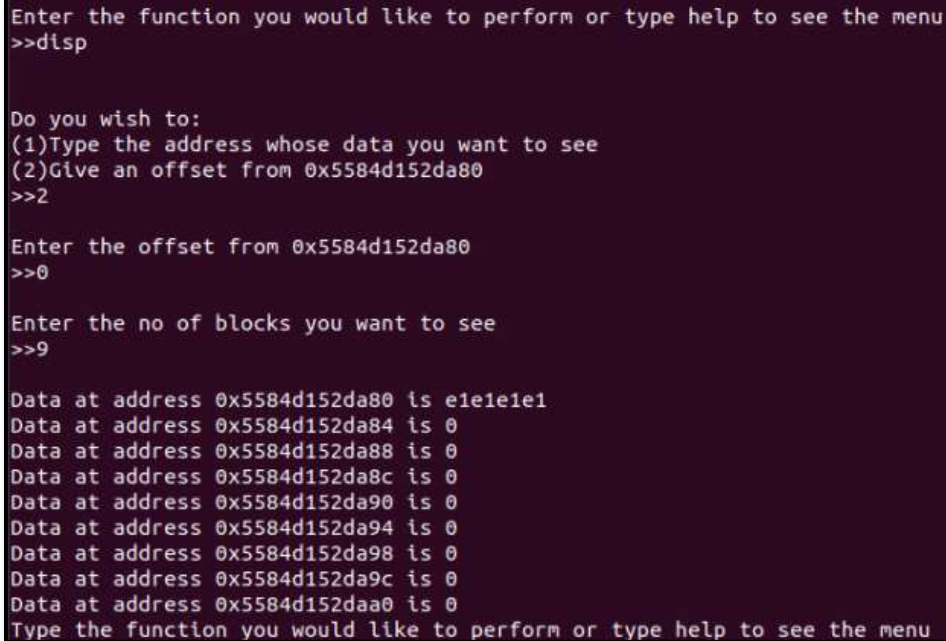
Project 1: Interactive Memory Manipulation

5. Command 'disp'

The syntax of display command is 'disp'

The disp function displays the contents of the memory location. When the user invokes the disp function then user is asked to enter the address manually or provide an offset from the start of allocated address. This address becomes the starting address to be displayed. Then user is asked to enter the number of blocks of memory to be displayed after the address. The command displays the memory location and its respective contents on the command line.

disp command in action



```
Enter the function you would like to perform or type help to see the menu
>>disp

Do you wish to:
(1)Type the address whose data you want to see
(2)Give an offset from 0x5584d152da80
>>2

Enter the offset from 0x5584d152da80
>>0

Enter the no of blocks you want to see
>>9

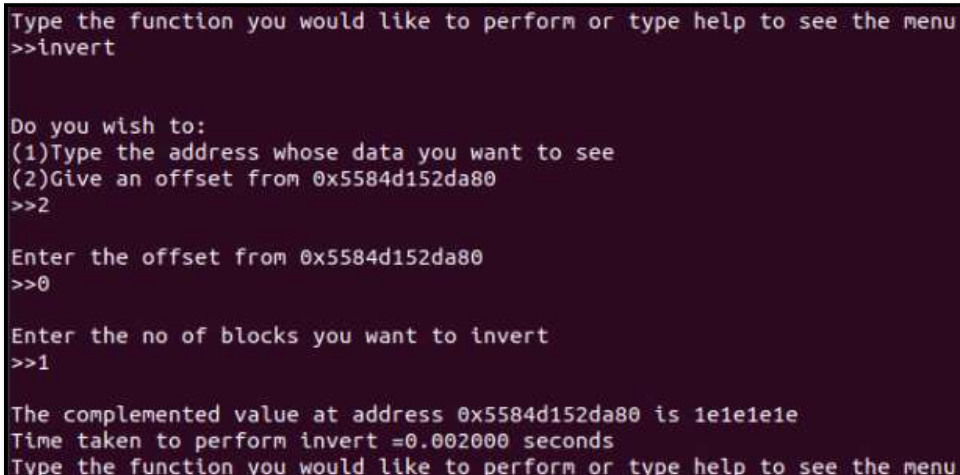
Data at address 0x5584d152da80 is e1e1e1e1
Data at address 0x5584d152da84 is 0
Data at address 0x5584d152da88 is 0
Data at address 0x5584d152da8c is 0
Data at address 0x5584d152da90 is 0
Data at address 0x5584d152da94 is 0
Data at address 0x5584d152da98 is 0
Data at address 0x5584d152da9c is 0
Data at address 0x5584d152daa0 is 0
Type the function you would like to perform or type help to see the menu
```

6. Command 'invert'

The syntax of invert command is 'invert'

When the user invokes the invert function user is asked to enter the address manually or provide an offset from the start of allocated address. The user needs to enter the number of memory blocks to invert. The contents in the specified address locations are inverted. Also, the function returns the amount of time required for block inversion.

invert command in action



```
Type the function you would like to perform or type help to see the menu
>>invert

Do you wish to:
(1)Type the address whose data you want to see
(2)Give an offset from 0x5584d152da80
>>2

Enter the offset from 0x5584d152da80
>>0

Enter the no of blocks you want to invert
>>1

The complemented value at address 0x5584d152da80 is 1e1e1e1e
Time taken to perform invert =0.002000 seconds
Type the function you would like to perform or type help to see the menu
```

Principles of Embedded Software

Project 1: Interactive Memory Manipulation

7. Command 'write_pattern'

The syntax of write pattern command is 'write_pattern'

The command displays all the available address user can write pattern to. Then the user is asked to enter the address manually or provide an offset from the start of allocated address. User needs to enter the number of 32-bit blocks to write pattern to and seed value. The seed value and address are passed onto the pseudo random number generator and it gives back the pattern to write the memory at the corresponding memory address and reports back the time required for writing the pattern.

write_pattern command in action ↓

```
Type the function you would like to perform or type help to see the menu
>>write_pattern
The available addresses on which you can write are
0x5584d152da80
0x5584d152da84
0x5584d152da88
0x5584d152da8c
0x5584d152da90
0x5584d152da94
0x5584d152da98
0x5584d152da9c
0x5584d152daa0

Do you wish to:
(1)Type the address whose data you want to see
(2)Give an offset from 0x5584d152da80
>>2

Enter the offset from 0x5584d152da80
>>1
Enter the no of 32-bit memory blocks to write pattern to
>>8
Enter the seed value to generate the psudo-random patttern
>>7
Write Pattern successful
Time taken to perform write pattern =0.030000 seconds
Type the function you would like to perform or type help to see the menu
```

The contents of the memory location after write_pattern is successful.

```
Type the function you would like to perform or type help to see the menu
>>disp

Do you wish to:
(1)Type the address whose data you want to see
(2)Give an offset from 0x5584d152da80
>>2

Enter the offset from 0x5584d152da80
>>0

Enter the no of blocks you want to see
>>9

Data at address 0x5584d152da80 is 1e1e1e1e
Data at address 0x5584d152da84 is cac8589e
Data at address 0x5584d152da88 is c6c45492
Data at address 0x5584d152da8c is c2c05096
Data at address 0x5584d152da90 is dedc4c8a
Data at address 0x5584d152da94 is dad8488e
Data at address 0x5584d152da98 is d6d44482
Data at address 0x5584d152da9c is d2d04086
Data at address 0x5584d152daa0 is eeec7cba
Type the function you would like to perform or type help to see the menu
>>
```


Principles of Embedded Software

Project 1: Interactive Memory Manipulation

8. Command 'verify_pattern'

The syntax of verify pattern command is 'verify_pattern'

The verify_pattern command is like write_pattern command but the difference is that the verify_pattern command compares the memory contents based on the seed value and reports back the address on which wrong data is present based on seed value. If all the memory locations are valid then it just states that Verification is done and no address locations are reported back.

verify_pattern in action ↓

```
Type the function you would like to perform or type help to see the menu
>>verify_pattern
The available addresses on which you can verify are

0x5584d152da80
0x5584d152da84
0x5584d152da88
0x5584d152da8c
0x5584d152da90
0x5584d152da94
0x5584d152da98
0x5584d152da9c
0x5584d152daa0

Do you wish to:
(1)Type the address whose data you want to see
(2)Give an offset from 0x5584d152da80
>>2

Enter the offset from 0x5584d152da80
>>1
Enter the no of 32-bit memory blocks to verify pattern
>>8
Enter the seed value to generate the psudo-random patttern for verification
>>7
Verification Done
Time taken to perform write pattern =0.029000 seconds
Type help to see more options
```

The memory locations with wrong data is reported back by the command as below.

```
>>verify_pattern
The available addresses on which you can verify are

0x5582f3feba80
0x5582f3feba84

Do you wish to:
(1)Type the address whose data you want to see
(2)Give an offset from 0x5582f3feba80
>>2

Enter the offset from 0x5582f3feba80
>>0
Enter the no of 32-bit memory blocks to verify pattern
>>2
Enter the seed value to generate the psudo-random patttern for verification
>>4
Wrong pattern foundWrong pattern foundVerification Done
Time taken to perform write pattern =0.006000 seconds
Wrong addresses are
5582f3feba80
5582f3feba84
```

9. Command 'ext'

The syntax of exit command is 'ext'

The ext command when invoked exits the developed command line utility.

```
Enter the function you would like to perform or type help to see the menu
>>ext

BYEEEE!!
```

Principles of Embedded Software

Project 1: Interactive Memory Manipulation

TEST CASE INPUT FILE

```
allocate
3
write
2
0
0xe1e1e1e1
disp
2
0
3
write_pattern
2
1
2
6
disp
2
0
3
invert
2
0
1
disp
2
0
3
verify_pattern
2
1
2
6
verify_pattern
2
0
3
6
freee
ext
```


Principles of Embedded Software

Project 1: Interactive Memory Manipulation

REPORT QUESTIONS

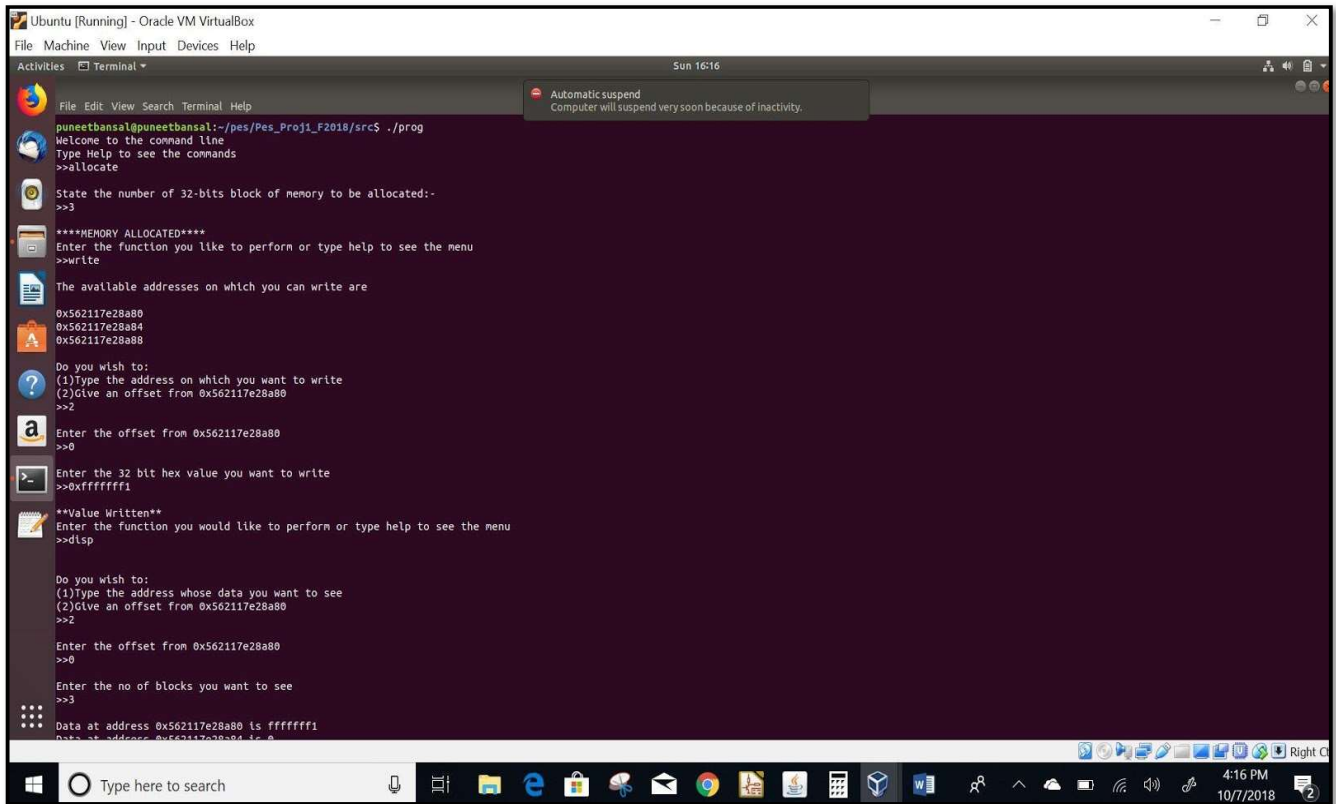
1. How well did your effort estimates match with actual effort? Provide examples of both inaccurate and accurate estimates and discuss any contributing factors to the success or failings of your effort estimates.
 - ➔ In our project the effort estimates were a mix of overestimation, underestimation and correct guesses. We overestimated the time for help, free, exit and measuring execution time. This was because of lack of clarity of the function the commands would perform.
 - ➔ The allocate command, write command and main function required more time than estimated since time was needed to refresh the concepts of malloc and function pointers. Apart from this time exceeded our estimates for pattern generator function since it needed to be corrected overtime. With time we realized the actual use of writing different contents to different memory locations which called for rework.
 - ➔ The most accurate estimate was made for setup of build environment, display, invert, write pattern and verify pattern, as the logic for implementation of these function was similar to other functions.
 - ➔ The most common contributing factor for failure of accurate estimates was lack of knowledge of C concepts used in this program like makefile, malloc and pointers. As a result of this we had to spend significant amount of time going through the concepts before we could actually start writing the code.
2. Were there any tasks that you forgot to include in your project plan?
 - ➔ The things missing from project plan was the bug fixing part. Most of the bugs are identified during testing of the code. The project plan did not include the time required to fix the bugs once they were identified after unit testing and integration testing of different functionalities.
3. What is the largest block of memory you can allocate? Discuss.
 - ➔ It will be difficult to check the maximum block of memory that can be allocated. It basically depends on the maximum contiguous memory available at that time. If the user wants to allocate x number of 32-bit blocks and there is $32 \times x$ bits of contiguous physical memory available, then the memory will be allocated, and the user will be able to write on these blocks without any segmentation fault. If $32 \times x$ bits of memory is not available, then the system would display error. However, sometimes the malloc allocates memory from the virtual memory. So, the memory would get allocated but when the user goes on to write at the address, they would receive segmentation fault. This happened because even if there is memory allocated in virtual memory, there is not sufficient contiguous space in the physical memory.
4. What happens if you try to read outside of the allocated block?
 - ➔ The system we implemented is a 32-bit system. So, it only permits the user to read complete 32 bits block of memory. In the system, there are 2 ways a user can read outside the allocated block.

The first is when the user enters an invalid address. In the interface this can be done if the user enters an address that is not the starting address of a valid 32-bit block or by providing an offset value that exceeds the permissible value.

Principles of Embedded Software

Project 1: Interactive Memory Manipulation

For instance, a user allocates 3 blocks of 32 bits and writes on the first block.



```
Ubuntu [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities Terminal Sun 16:16
Automatic suspend
Computer will suspend very soon because of inactivity.

puneetbansal@puneetbansal:~/pes/Proj1_F2018/src$ ./prog
Welcome to the command line
Type Help to see the commands
>>allocate
State the number of 32-bits block of memory to be allocated:-
>>3
****MEMORY ALLOCATED****
Enter the function you like to perform or type help to see the menu
>>write
The available addresses on which you can write are
0x562117e2a80
0x562117e2a84
0x562117e2a88
Do you wish to:
(1)Type the address on which you want to write
(2)Give an offset from 0x562117e2a80
>>2
Enter the offset from 0x562117e2a80
>>0
Enter the 32 bit hex value you want to write
>>0xffffffff
**Value Written**
Enter the function you would like to perform or type help to see the menu
>>disp
Do you wish to:
(1)Type the address whose data you want to see
(2)Give an offset from 0x562117e2a80
>>2
Enter the offset from 0x562117e2a80
>>0
Enter the no of blocks you want to see
>>3
Data at address 0x562117e2a80 is ffffffff1
Data at address 0x562117e2a84 is 0
Data at address 0x562117e2a88 is 0
```

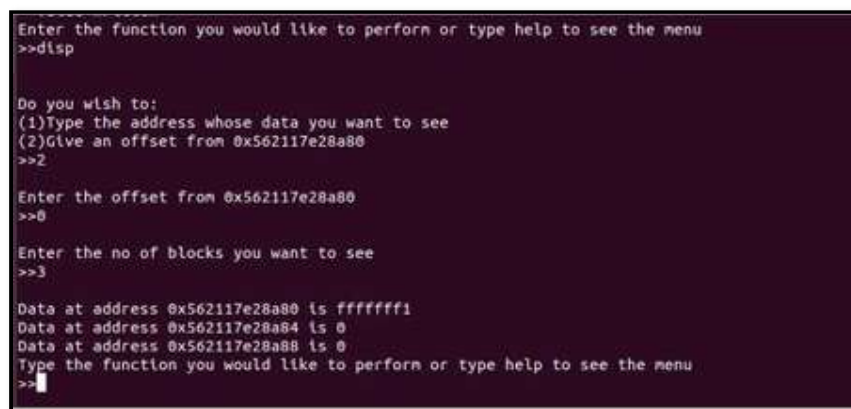
With reference to the above screenshot, the starting addresses of the 3 allocated blocks of memory are:

0x562117e2a80

0x562117e2a84

0x562117e2a88

The user writes at the address 0x562117e2a80 value of 0xffffffff. The user then enters the command for display entering offset as 0(maximum possible value=2) and the number of blocks to be 3(maximum possible value=3), both of which are within the permissible range. So, the program outputs the data at the 3 blocks of memory as seen in the screenshot below.



```
Enter the function you would like to perform or type help to see the menu
>>disp
Do you wish to:
(1)Type the address whose data you want to see
(2)Give an offset from 0x562117e2a80
>>2
Enter the offset from 0x562117e2a80
>>0
Enter the no of blocks you want to see
>>3
Data at address 0x562117e2a80 is ffffffff1
Data at address 0x562117e2a84 is 0
Data at address 0x562117e2a88 is 0
Type the function you would like to perform or type help to see the menu
>>
```

Now, if the user enters an address like 0x562117e2a70 or 0x562117e2a90 i.e. any value that is not the starting address of the 3 blocks allocated, the program displays an error message saying “memory entered is not valid”

Principles of Embedded Software

Project 1: Interactive Memory Manipulation

```
>>disp

Do you wish to:
(1)Type the address whose data you want to see
(2)Give an offset from 0x562117e28a80
>>1

Enter the address on which you want to see the contents
>>0x562117e28a70

Enter the no of blocks you want to see
>>2

Memory entered is not valid
Type disp to try again or help to see the menu
>>
```

The same error message is also displayed when a user enters an invalid offset of 3. Invalid because the maximum value of offset in this case is 2. The same can be seen in the screenshot below.

```
>>disp

Do you wish to:
(1)Type the address whose data you want to see
(2)Give an offset from 0x562117e28a80
>>2

Enter the offset from 0x562117e28a80
>>3

Enter the no of blocks you want to see
>>2

Memory entered is not valid
Type disp to try again or help to see the menu
>>
```

Now, consider the case when the user enters a valid address but an invalid block of memory to display. For instance, if the user allocated 3 blocks of memory and then asks to display 4 blocks. Then the program returns the message “The number of blocks entered is invalid” as seen in the screenshot below.”

```
>>disp

Do you wish to:
(1)Type the address whose data you want to see
(2)Give an offset from 0x557963f7ba80
>>2

Enter the offset from 0x557963f7ba80
>>0

Enter the no of blocks you want to see
>>4

The number of blocks entered is invalid
Type disp to try again or help to see the menu
>>
```

Principles of Embedded Software

Project 1: Interactive Memory Manipulation

5. What happens if you try to write outside of the allocated block?

- ➔ Let us understand this scenario, using an example. The user allocates 2 blocks of memory. The user then tries to write at 0x55f82d48ca88 which is not a valid address on which the user can write. So, the program displays the error message “The address entered is not a valid address”. This is again because we are trying to implement a 32-bit system and the user is only permitted to write at the starting addresses of the 32-bit blocks.

```
>>write
The available addresses on which you can write are
0x55f82d48ca80
0x55f82d48ca84
Do you wish to:
(1)Type the address on which you want to write
(2)Give an offset from 0x55f82d48ca80
>>1
Enter the address on which you want to write the contents
>>0x55f82d48ca88
The address entered is not a valid address
Type write to try again or help to see other options
>>
```

Again, the other problematic case would be when user enters an invalid offset. Like and offset of 3 in the case shown in the picture below

```
>>write
The available addresses on which you can write are
0x55f82d48ca80
0x55f82d48ca84
Do you wish to:
(1)Type the address on which you want to write
(2)Give an offset from 0x55f82d48ca80
>>2
Enter the offset from 0x55f82d48ca80
>>3
The address entered is not a valid address
Type write to try again or help to see other options
>>
```

6. Analyse the time it takes to invert memory for different memory block sizes. Is there a linear relationship between time and size?

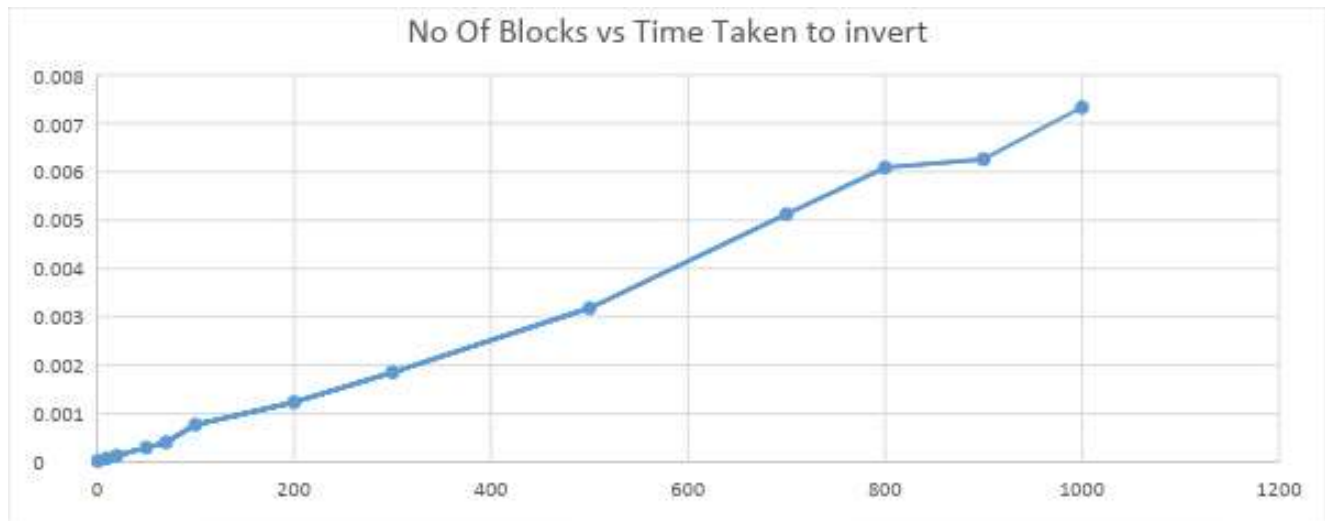
- ➔ To calculate the relationship, we tried to develop a data set of the time taken to invert compared to the number of blocks that were to be inverted.

NUMBER OF BLOCKS	TIME TAKEN (SEC)
1	0.000020
10	0.000073
20	0.000127
50	0.000295
70	0.000405
100	0.000771

Principles of Embedded Software

Project 1: Interactive Memory Manipulation

200	0.001233
300	0.001853
500	0.003180
700	0.005122
800	0.006086
900	0.006259
1000	0.00733



In the above graph, we have the number of blocks that were inverted on the x axis. From the above graph obtained, we can see that there is almost a linear relationship between the number of blocks that are to be inverted vs the time taken to invert the blocks.

7. What are the limitations of your time measurements?

→ The time is taken by the clock itself to start and stop and since the system is digital and time is an analog entity, the digital systems can only come close to the accurate time which limits its accuracy even further.

→ We have used the `clock_t` object to store the time stamps. The use of this object limits the value that can be stored to 36min before it comes back to zero. Hence if the function of which time is measured takes more than 36 min to execute, it would not get the accurate time.

Reference → <https://bytes.com/topic/c/answers/213478-clock-function>

8. What improvements can you make to the invert function, your program or your build, to make it run faster? How much improvement did you achieve?

→ For the invert function, we tried using `!(data)` and `~(data)`. There was no performance improvement seen.

→ In the program, the performance improvement can be done by using efficient alternatives to if-else-if statements.

→ In the makefile, the further efficient implementation could be done with `CFFLAGS` and variables.

Principles of Embedded Software

Project 1: Interactive Memory Manipulation

9. What algorithm did you choose for your pattern generator? How does it generate its pattern? What are its strengths and weaknesses?
- ➔ The pseudo random number generator relies on the polynomial equation which takes seed value and generates a pattern. The pattern should be different in different blocks of memory with different seed value. To achieve this the address is taken into consideration to generate the pseudo random code so that the pattern would be unique to the address. Also, there is requirement of pattern to be predictable for same seed value and same address hence can be used to verify the contents of the memory.

ALGORITHM:

To achieve this algorithm used to generate the pattern is as follows –

1. Take seed value from user and get the address to write the pattern.
2. Generate a basic pattern using a polynomial
3. $\text{Pattern} = 123 * \text{pow}(x, 10) + 17 * \text{pow}(x, 7) + 11 * \text{pow}(x, 3) + 9 * \text{pow}(x, 2) * 3$
where x is the seed value.
4. Generate a mask 32-bit long to get the 32-bit result.
5. Mask the pattern and address
6. XOR the pattern and address and left shift address by 8 bits and repeat the process 4 times.
7. Return pattern to calling function.

Reasons for considering address and step 5 which involves left shift and XOR operation.

The pattern generated would be unique to seed value so for same seed value the pattern would be same. This is done to make the pattern predictable hence verification becomes easy and possible. Since various memory locations cannot have repeated pattern the address is considered for pattern generation.

To make pattern more diverse the address is left shifted by 8 bits and again XOR'ed with pattern.

If the address or pattern goes beyond 32-bit, the mask will mask the data and convert back to 32-bit word.

Thus, the pattern generated would be different in different block and since the logic is based on pseudo random number and address, it would be same for same seed value. Thus, the pattern can be verified in the same manner using the Pattern generator function.

Strengths of Pattern generator function –

- a. Easy polynomial equation which will always give unique value for the seed value.
- b. Easily calculated and predictable and hence verification becomes easy and possible.

Weakness of Pattern generator function –

- a. It is a time consuming process and takes a lot of cpu time which is a crucial resource for embedded system.