Marks : 16 **CSCI131**

Spring Session 2015

Due date: September 4th

Assignment 2
Assembly language programming
8 marks

Complete the associated exercises before attempting the assignment

Aim

This assignment provides some limited experience with assembly language programming.

Objectives

On completion of this assignment and its associated exercise, you will be able to:

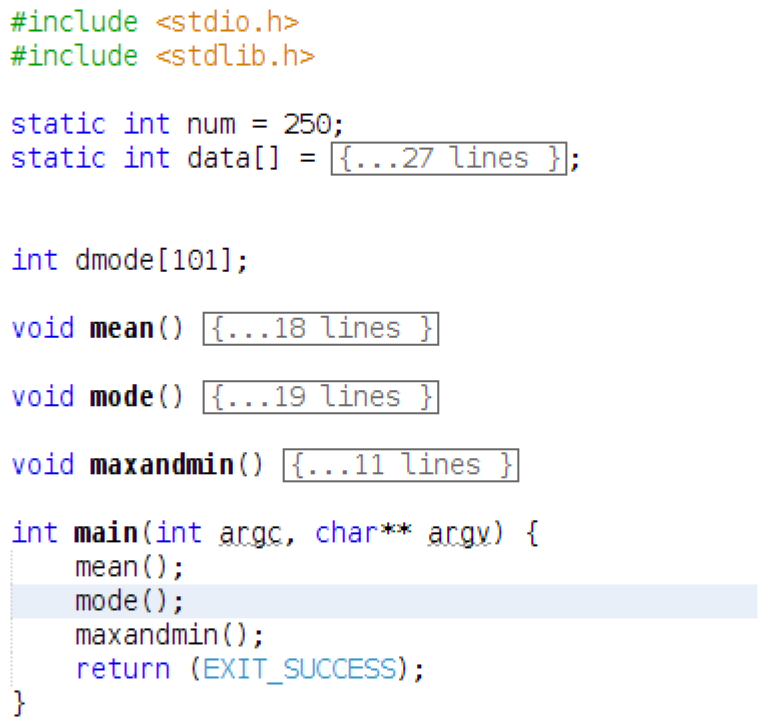
- Explain how assembly language programs are prepared using a two pass assembler, are loaded and executed
- Explain the operation of a simple assembler and the use of conventional assembly language directives for setting origins, defining variables, strings, and blocks of initialized data
- Explain how input and output can be realised using wait loops and interrupt mechanisms
- Explain "op-codes" and "addressing modes"
- Use assembly language to implement simple programming constructs such as indexed array accessing, use of pointers, simple subroutine calls, and use of a primitive "operating system" implemented with trap (svc) system call instructions.

Task

You are to implement a simple data processing application in PDP11 assembly language, making use of the supplied primitive "operating system" to handle I/O.

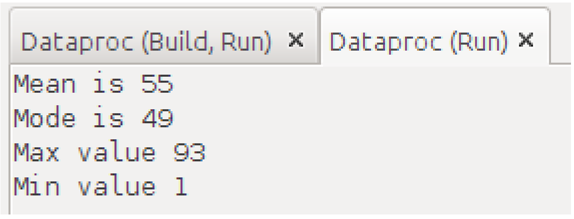
A simple program for summary statistics

The program is to produce some summary statistics for an array of data. The outline of a C version (which you should implement in full first) is as follows:



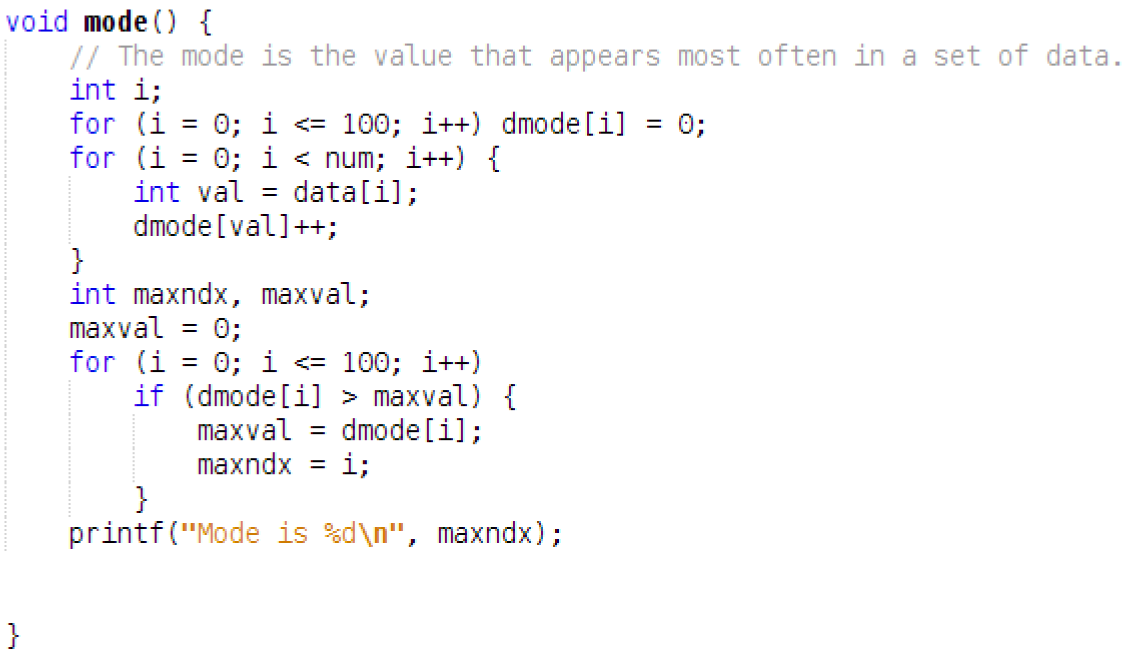
```
#include <stdio.h>
#include <stdlib.h>

static int num = 250;
static int data[] = {...27 lines };


int dmode[101];

void mean() {...18 lines }

void mode() {...19 lines }

void maxandmin() {...11 lines }

int main(int argc, char** argv) {
    mean();
    mode();
    maxandmin();
    return (EXIT_SUCCESS);
}
```

The program determines the mean, mode of the data in an array, and also identifies the largest and smallest value.



```
Mean is 55
Mode is 49
Max value 93
Min value 1
```

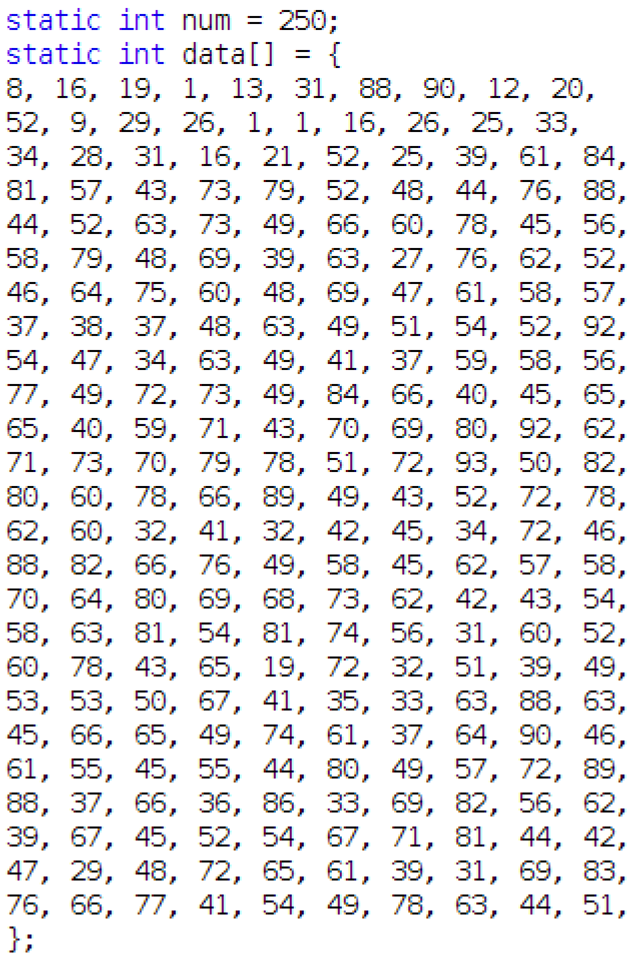
You should be able to work out the code for the functions mean() and maxandmin(). Code for the mode() function could be as follows:



```
void mode() {
    // The mode is the value that appears most often in a set of data.
    int i;
    for (i = 0; i <= 100; i++) dmode[i] = 0;
    for (i = 0; i < num; i++) {
        int val = data[i];
        dmode[val]++;
    }
    int maxndx, maxval;
    maxval = 0;
    for (i = 0; i <= 100; i++)
        if (dmode[i] > maxval) {
            maxval = dmode[i];
            maxndx = i;
        }
    printf("Mode is %d\n", maxndx);


}
```

The data for the array are to be created using a supplied “datagen” program.

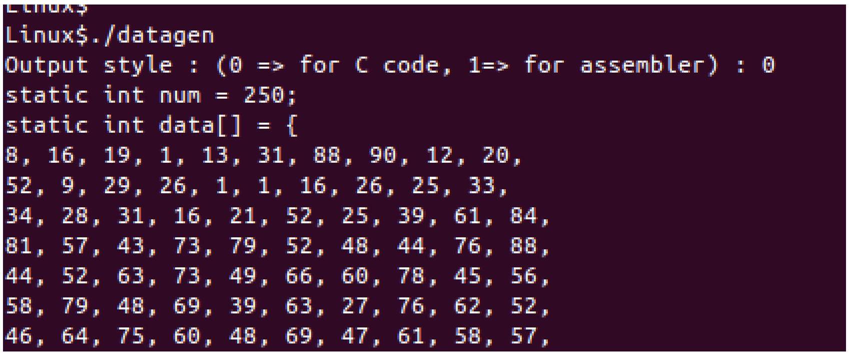
Example data – your values will differ:



```
static int num = 250;
static int data[] = {
8, 16, 19, 1, 13, 31, 88, 90, 12, 20,
52, 9, 29, 26, 1, 1, 16, 26, 25, 33,
34, 28, 31, 16, 21, 52, 25, 39, 61, 84,
81, 57, 43, 73, 79, 52, 48, 44, 76, 88,
44, 52, 63, 73, 49, 66, 60, 78, 45, 56,
58, 79, 48, 69, 39, 63, 27, 76, 62, 52,
46, 64, 75, 60, 48, 69, 47, 61, 58, 57,
37, 38, 37, 48, 63, 49, 51, 54, 52, 92,
54, 47, 34, 63, 49, 41, 37, 59, 58, 56,
77, 49, 72, 73, 49, 84, 66, 40, 45, 65,
65, 40, 59, 71, 43, 70, 69, 80, 92, 62,
71, 73, 70, 79, 78, 51, 72, 93, 50, 82,
80, 60, 78, 66, 89, 49, 43, 52, 72, 78,
62, 60, 32, 41, 32, 42, 45, 34, 72, 46,
88, 82, 66, 76, 49, 58, 45, 62, 57, 58,
70, 64, 80, 69, 68, 73, 62, 42, 43, 54,
58, 63, 81, 54, 81, 74, 56, 31, 60, 52,
60, 78, 43, 65, 19, 72, 32, 51, 39, 49,
53, 53, 50, 67, 41, 35, 33, 63, 88, 63,
45, 66, 65, 49, 74, 61, 37, 64, 90, 46,
61, 55, 45, 55, 44, 80, 49, 57, 72, 89,
88, 37, 66, 36, 86, 33, 69, 82, 56, 62,
39, 67, 45, 52, 54, 67, 71, 81, 44, 42,
47, 29, 48, 72, 65, 61, 39, 31, 69, 83,
76, 66, 77, 41, 54, 49, 78, 63, 44, 51,
};
```

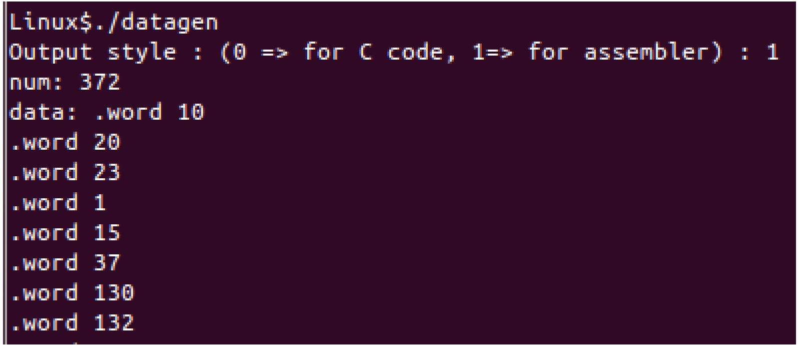
An executable version of the datagen program for the Ubuntu computers in the lab can be downloaded from http://www.uow.edu.au/~nabg/131/Resources/datagen (after download, change the permission to make the file executable). This program will generate a personalized set of data values; it will generate the same set of values each time you run the program, and can output the data in a form for pasting into C code or into your assembly language program.

Output for C:



```
Linux$
Linux$./datagen
Output style : (0 => for C code, 1=> for assembler) : 0
static int num = 250;
static int data[] = {
8, 16, 19, 1, 13, 31, 88, 90, 12, 20,
52, 9, 29, 26, 1, 1, 16, 26, 25, 33,
34, 28, 31, 16, 21, 52, 25, 39, 61, 84,
81, 57, 43, 73, 79, 52, 48, 44, 76, 88,
44, 52, 63, 73, 49, 66, 60, 78, 45, 56,
58, 79, 48, 69, 39, 63, 27, 76, 62, 52,
46, 64, 75, 60, 48, 69, 47, 61, 58, 57,
```

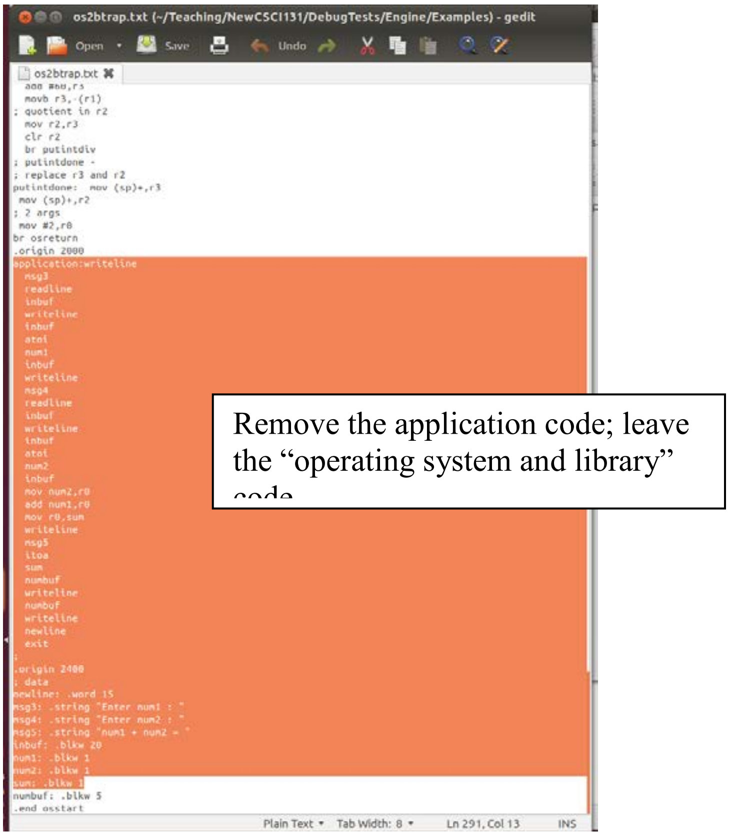
Output for assembly language (same values, just reformatted and printed in octal):



```
Linux$./datagen
Output style : (0 => for C code, 1=> for assembler) : 1
num: 372
data: .word 10
.word 20
.word 23
.word 1
.word 15
.word 37
.word 130
.word 132
```

Assembly language implementation

As in the exercise, start your assembly language version by taking the code of the example os2btrap.txt and cutting out the example application code to leave just the “operating system and library” components.

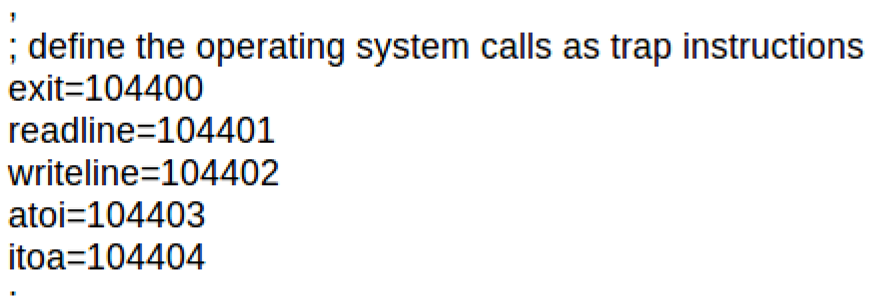


os2btrap.txt (~/Teaching/NewCSCI131/DebugTests/Engine/Examples) - gedit
Open
Save
Undo
os2btrap.txt
Plain Text
Tab Width: 8
Ln 291, Col 13
INS

```
  add #60,r3
  movb r3,-(r1)
; quotient in r2
  mov r2,r3
  clr r2
  br putintdiv
; putintdone -
; replace r3 and r2
putintdone:  mov (sp)+,r3
 mov (sp)+,r2
; 2 args
 mov #2,r0
br osreturn
.origin 2000
application:writeline
  msg3
  readline
  inbuf
  writeline
  inbuf
  atoi
  num1
  inbuf
  writeline
  msg4
  readline
  inbuf
  writeline
  inbuf
  atoi
  num2
  inbuf
  mov num2,r0
  add num1,r0
  mov r0,sum
  writeline
  msg5
  itoa
  sum
  numbuf
  writeline
  numbuf
  writeline
  newline
  exit
;
.origin 2400
; data
newline: .word 15
msg3: .string "Enter num1 : "
msg4: .string "Enter num2 : "
msg5: .string "num1 + num2 = "
inbuf: .blkw 20
num1: .blkw 1
num2: .blkw 1
sum: .blkw 1
numbuf: .blkw 5
.end osstart
```

Remove the application code; leave the “operating system and library” code

The “operating system” offers five system functions that are invoked via traps (supervisor calls):

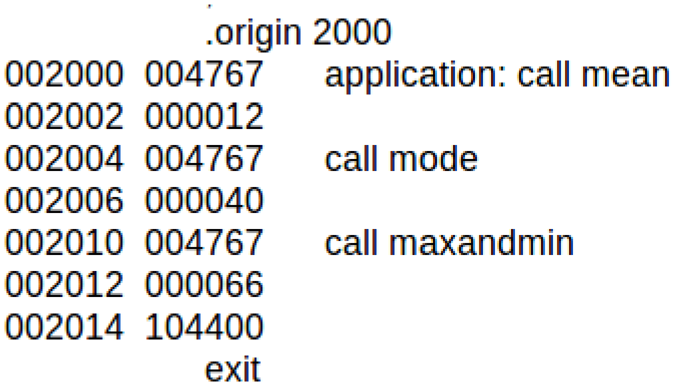


```
,
; define the operating system calls as trap instructions
exit=104400
readline=104401
writeline=104402
atoi=104403
itoa=104404
.
```

Supervisor calls:

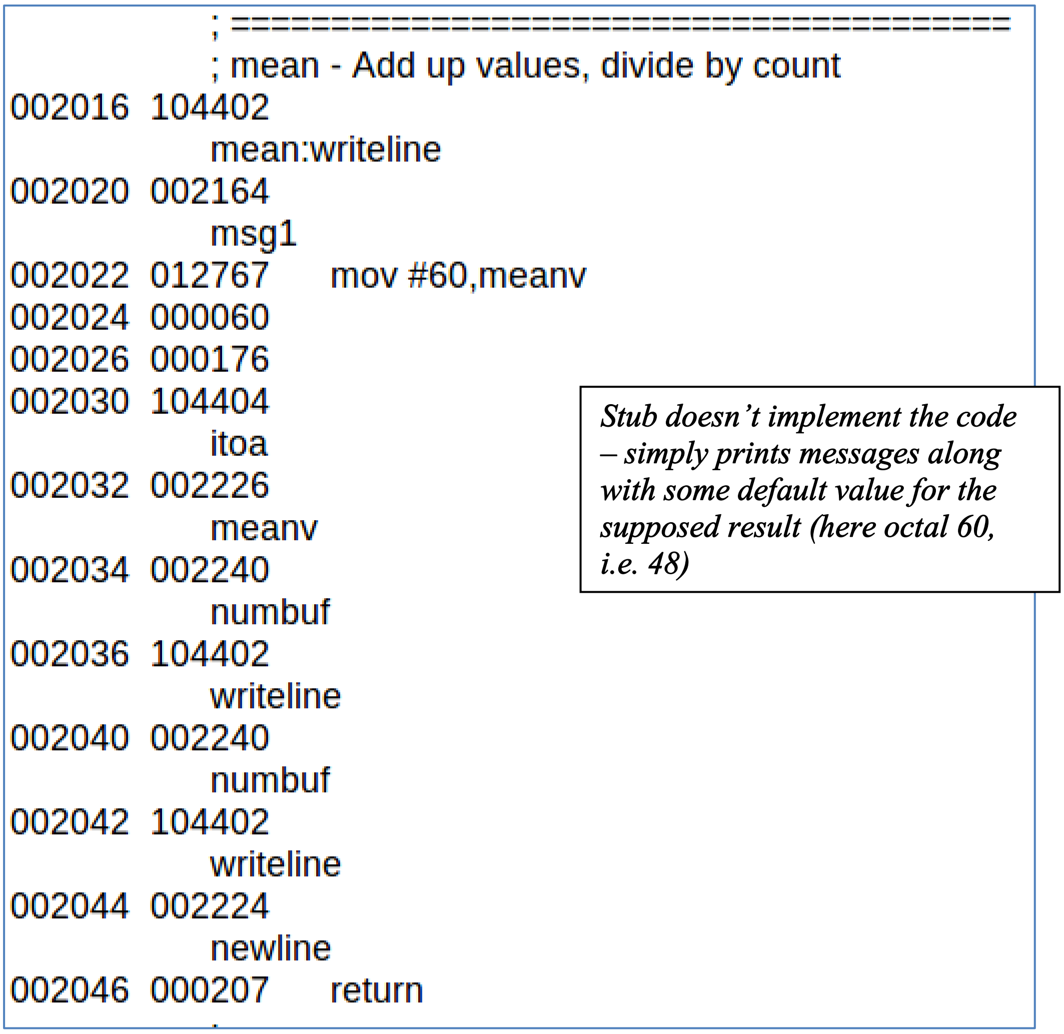
- Exit:
  Terminate execution of application program, return to OS (which then halts).
- Readline:
  This trap takes one argument in the following memory location; this is the address of a buffer (character array) where input characters will be stored. The implementation of the supervisor call reads characters from the keyboard (under interrupt control) until a newline character has been read. All characters read are store in the input buffer.
- Writeline:
  This trap takes one argument in the following memory location; this is the address of a buffer (character array) with a generated message. The implementation of the supervisor call writes the content of the array to the tty.
- Atoi:
  (Not used in this application). This trap takes one argument; this is the address of a buffer that is supposed to contain the ASCII representation of an integer. The integer value will be returned.
- Itoa:
  This trap takes two arguments; the first is the address of a variable containing an integer value, the second is the address of a buffer where an ASCII string should be generated to represent that value (in decimal).

Start by implementing an application that invokes stub functions that output fixed values rather than actual data:



```
               .origin 2000
002000  004767     application: call mean
002002  000012
002004  004767     call mode
002006  000040
002010  004767     call maxandmin
002012  000066
002014  104400
               exit
```

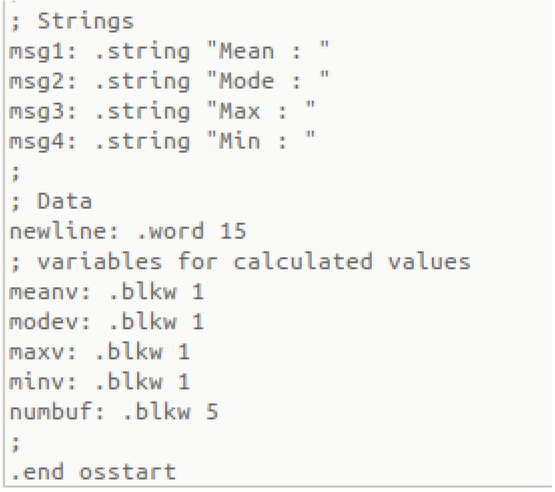
These stub functions simply print messages and output a fixed value. A stub implementation of mean() is shown below; the mode() and maxandmin() stub functions are similar:



```
                ; ========================================
                ; mean - Add up values, divide by count
002016  104402
                mean:writeline
002020  002164
                msg1
002022  012767      mov #60,meanv
002024  000060
002026  000176
002030  104404
                itoa
002032  002226
                meanv
002034  002240
                numbuf
002036  104402
                writeline
002040  002240
                numbuf
002042  104402
                writeline
002044  002224
                newline
002046  000207      return
```

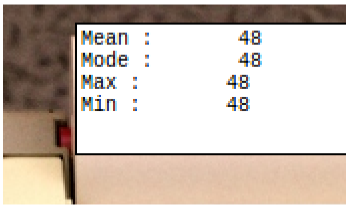
Stub doesn't implement the code – simply prints messages along with some default value for the supposed result (here octal 60, i.e. 48)

You will need message strings and some variables:



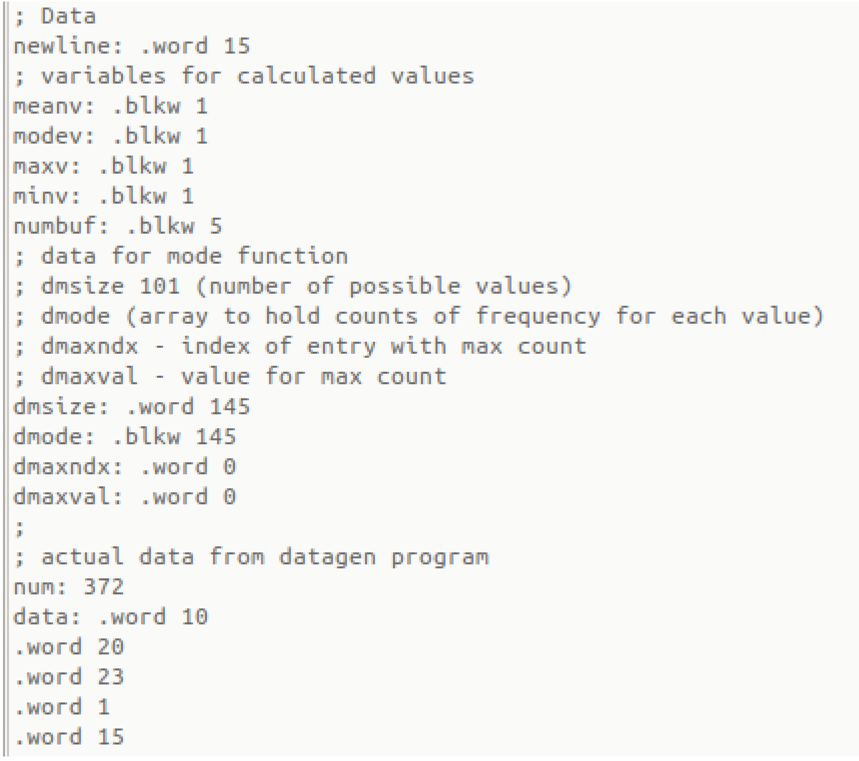
```
; Strings
msg1: .string "Mean : "
msg2: .string "Mode : "
msg3: .string "Max : "
msg4: .string "Min : "
;
; Data
newline: .word 15
; variables for calculated values
meanv: .blkw 1
modev: .blkw 1
maxv: .blkw 1
minv: .blkw 1
numbuf: .blkw 5
;
.end osstart
```

This stubbed out version of the application should run:



```
Mean :       48
Mode :       48
Max :       48
Min :       48
```

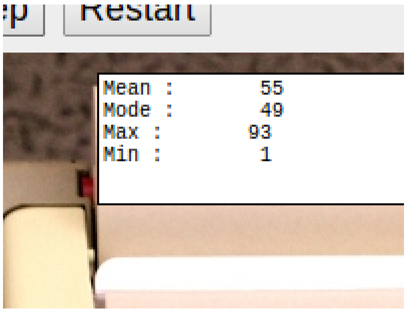
You can then add the actual data to your code:



```
; Data
newline: .word 15
; variables for calculated values
meanv: .blkw 1
modev: .blkw 1
maxv: .blkw 1
minv: .blkw 1
numbuf: .blkw 5
; data for mode function
; dmsize 101 (number of possible values)
; dmode (array to hold counts of frequency for each value)
; dmaxndx - index of entry with max count
; dmaxval - value for max count
dmsize: .word 145
dmode: .blkw 145
dmaxndx: .word 0
dmaxval: .word 0
;
; actual data from datagen program
num: 372
data: .word 10
.word 20
.word 23
.word 1
.word 15
```

Your real task is implementing the functions mean(), mode(), and maxandmin(), using simple assembly language constructs for array access and counting loops.

Your completed assembly language program should produce the same output as your C version:



```
Mean :       55
Mode :       49
Max :       93
Min :        1
```

Assignment report

You do not submit your code files!

You are marked on a report that you write. This report presents details of your assembly language program. You must include the assembly language code for the three functions (screen shots can be used). For each function, explain how you have encoded the loop constructs in assembly language and how you are accessing the array elements.

Submission

Prepare your report and convert to PDF format as the file A2.pdf.

Submission is done electronically via a program called `turnin` that runs on "banshee" – the main CS undergraduate machine. You must first transfer your A2.pdf file to your home directory on banshee (this is different from the home directory that you access on the Linux machines). You can transfer the file using a SSH file-transfer program. The Ubuntu OS allows you to open a file-browser connected to your banshee home directory – and you can simply drag your A2.pdf file across using the visual file browser.

For CSCI131, assignments are submitted electronically via the `turnin` system. For this assignment you submit your assignment via the command:

```
turnin -c csci131 -a 2 A2.pdf
```

Late submissions would be submitted as:

```
turnin -c csci131 -a 2late A2.pdf
```

The program `turnin` only works when you are logged in to the main banshee undergraduate server machine. From an Ubuntu workstation in the lab, you must open a terminal session on the local machine, and then login to banshee via ssh and run the `turnin` program.

Marking

The assignment is worth 8 marks total.

- Appearance and structure of report: 1 mark
- Evidence for correct operation (screenshots from both C and assembly language versions) : 1 mark
- Code and explanations of each of the three functions: 6 marks total