CSCI131 Assignment 2

Mohamed Asjad Athick (4970512)

3 September 2015

Report

Table of Contents

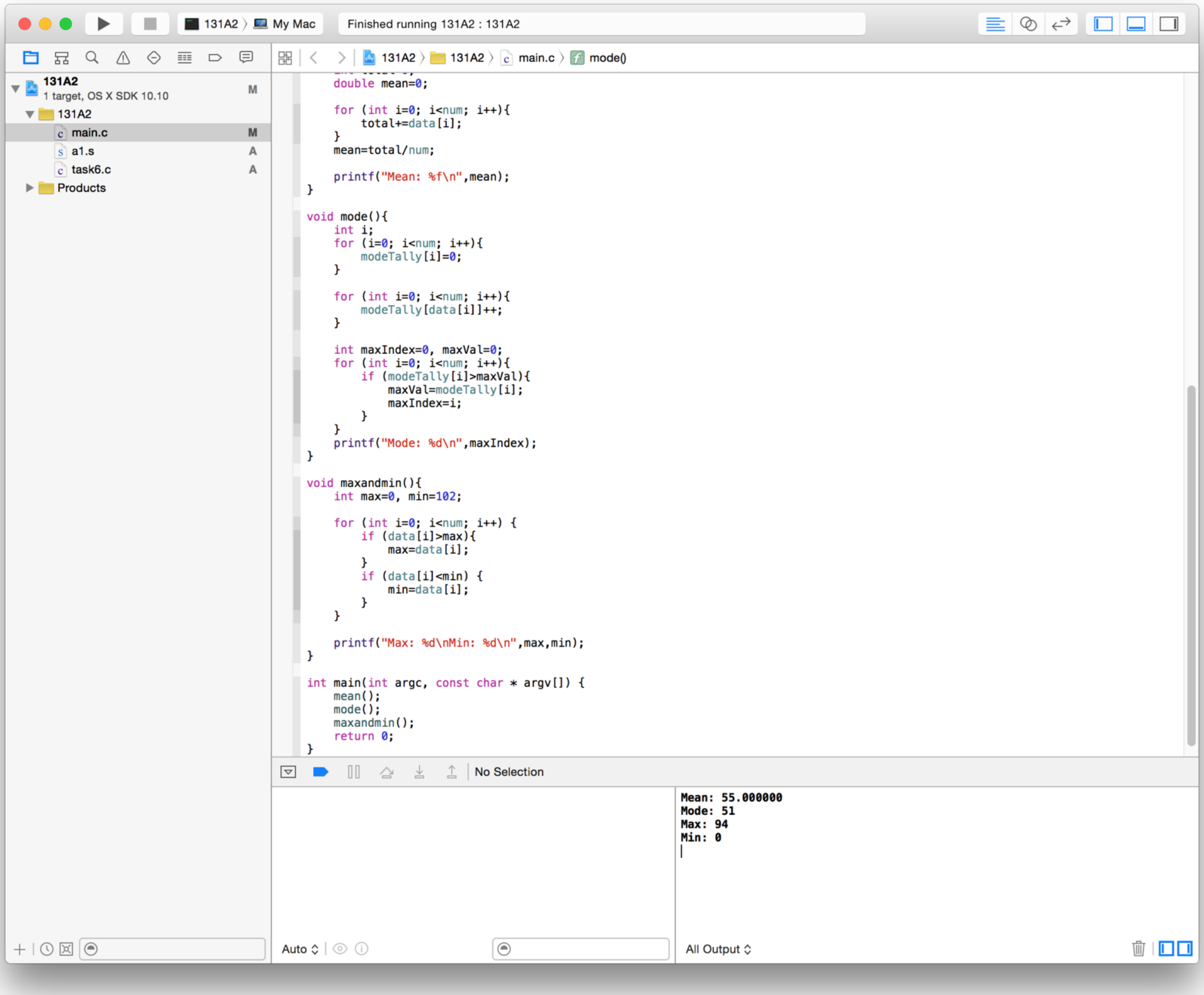
Introduction

A program to calculate the Mean, Mode, Max and Min of an array of integers was written in C, and Assembly for the PDP-11 architecture.

The report details the code in both versions with evidence for correct operation

Evidence for Correct Operation

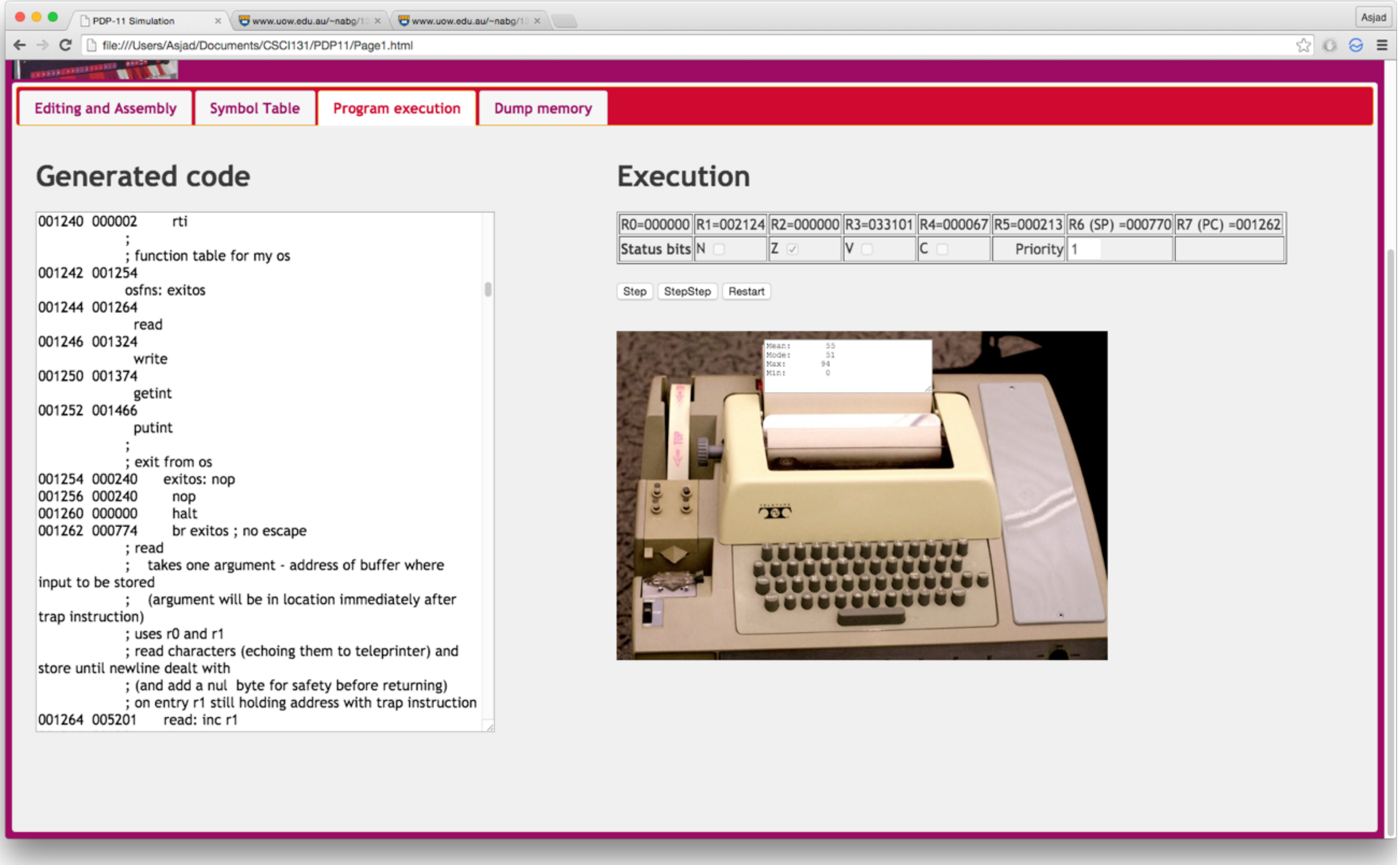
C



```c
    double mean=0;

    for (int i=0; i<num; i++){
        total+=data[i];
    }
    mean=total/num;

    printf("Mean: %f\n",mean);
}

void mode(){
    int i;
    for (i=0; i<num; i++){
        modeTally[i]=0;
    }

    for (int i=0; i<num; i++){
        modeTally[data[i]]++;
    }

    int maxIndex=0, maxVal=0;
    for (int i=0; i<num; i++){
        if (modeTally[i]>maxVal){
            maxVal=modeTally[i];
            maxIndex=i;
        }
    }
    printf("Mode: %d\n",maxIndex);
}

void maxandmin(){
    int max=0, min=102;

    for (int i=0; i<num; i++) {
        if (data[i]>max){
            max=data[i];
        }
        if (data[i]<min) {
            min=data[i];
        }
    }

    printf("Max: %d\nMin: %d\n",max,min);
}

int main(int argc, const char * argv[]) {
    mean();
    mode();
    maxandmin();
    return 0;
}
```

```
Mean: 55.000000
Mode: 51
Max: 94
Min: 0
```

Assembly



Generated code

```
001240  000002       rti
                  ;
                  ; function table for my os
001242  001254
                  osfns: exitos
001244  001264
                    read
001246  001324
                    write
001250  001374
                    getint
001252  001466
                    putint
                  ;
                  ; exit from os
001254  000240     exitos: nop
001256  000240       nop
001260  000000       halt
001262  000774       br exitos ; no escape
                  ; read
                  ;    takes one argument - address of buffer where
input to be stored
                  ;    (argument will be in location immediately after
trap instruction)
                  ; uses r0 and r1
                  ; read characters (echoing them to teleprinter) and
store until newline dealt with
                  ; (and add a nul  byte for safety before returning)
                  ; on entry r1 still holding address with trap instruction
001264  005201     read: inc r1
```

Execution

R0=000000	R1=002124	R2=000000	R3=033101	R4=000067	R5=000213	R6 (SP) =000770	R7 (PC) =001262
Status bits	N ☐	Z ☑	V ☐	C ☐	Priority	1	

```
Mean:        55
Mode:        51
Max:        94
Min:         0
```

Code Explanation

C code

/Users/Asjad/NetBeansProjects/CppApplication_2/main.c

```c
//
//  main.c
//  131A2
//
//  Created by Asjad Athick on 31/08/2015.
//  Copyright (c) 2015 Asjad Athick. All rights reserved.
//

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

static int num = 250;
static int data[] = {
    5, 3, 12, 72, 22, 30, 2, 11, 32, 17,
    1, 8, 70, 7, 1, 30, 39, 43, 0, 58,
    9, 10, 13, 6, 15, 51, 63, 57, 79, 85,
    76, 89, 55, 81, 51, 34, 78, 68, 72, 59,
    75, 77, 66, 59, 61, 47, 63, 51, 43, 60,
    66, 70, 72, 46, 83, 39, 66, 40, 73, 83,
    90, 76, 66, 46, 41, 50, 48, 58, 50, 42,
    68, 43, 48, 59, 67, 66, 46, 66, 37, 64,
    79, 42, 45, 78, 67, 64, 59, 37, 80, 80,
    89, 73, 76, 34, 51, 37, 49, 43, 31, 41,
    79, 84, 51, 72, 39, 34, 58, 59, 82, 38,
    51, 31, 57, 53, 65, 63, 86, 84, 43, 58,
    42, 47, 70, 58, 59, 44, 42, 51, 45, 55,
    55, 65, 68, 68, 72, 54, 83, 72, 60, 53,
    73, 94, 68, 74, 38, 77, 73, 68, 28, 21,
    69, 58, 51, 62, 66, 70, 28, 50, 40, 53,
    61, 58, 23, 81, 56, 48, 68, 79, 80, 53,
    55, 54, 52, 64, 64, 59, 27, 88, 60, 67,
    66, 52, 63, 73, 36, 49, 65, 45, 46, 53,
    42, 33, 85, 64, 79, 71, 62, 45, 76, 40,
    76, 61, 34, 52, 58, 57, 67, 84, 77, 25,
    49, 40, 65, 62, 56, 47, 40, 43, 69, 72,
    72, 57, 70, 39, 57, 51, 58, 31, 91, 77,
    74, 44, 70, 91, 70, 35, 67, 51, 72, 55,
    59, 68, 92, 74, 65, 63, 23, 61, 74, 51,
};
```

```

int modeTally[101];

void mean(){
    int total=0;
    double mean=0;

    for (int i=0; i<num; i++){
        total+=data[i];
    }
    mean=total/num;

    printf("Total: %d\nMean: %f\n",total,mean);
}

void mode(){
    int i;
    for (i=0; i<num; i++){
        modeTally[i]=0;
    }

    for (int i=0; i<num; i++){
        modeTally[data[i]]++;
    }

    int maxIndex=0, maxVal=0;
    for (int i=0; i<num; i++){
        if (modeTally[i]>maxVal){
            maxVal=modeTally[i];
            maxIndex=i;
        }
    }

    printf("Max val: %d\nMode: %d\n",maxVal,maxIndex);

}

void maxandmin(){
    int max=0, min=102;

    for (int i=0; i<num; i++) {
        if (data[i]>max){
            max=data[i];
        }
```

```c
        if (data[i]<min) {
            min=data[i];
        }
    }

    printf("Max: %d\nMin: %d\n",max,min);
}

int main(int argc, const char * argv[]) {
    mean();
    mode();
    maxandmin();
    return 0;
}

```

Explanation of Assembly functions

(all the assembly code is attached at the end of the report)

Mean

```
mean:
  clr r0
  clr r1
  mov #len,r2
meanloop: add data(r0),r1
  add #2,r0
  sob r2,meanloop
  mov r1,r3
  mov r1,r5
  clr r4
  div num,r4
  mov r4,meanv
  return
```

Explanation:

- Register 0 is cleared
- Register 1 is cleared
- #len is moved into r2 (Len holds the length of the array = 250) (r2 is the counter)
- Label defined ‘meanloop’
- Data(ro) is added to r1 (r1 acts as the accumulator)
- ro is incremented by 2, to point to the next word in the array
- SOB checks the value of r2. If not 0, branches back to ‘meanloop’
- Value of r1 moved to r3 (represents total)
- Value of r1 moved to r5 to use the div function (r4,r5 used)
- Div performed on r4
- Value of r4 is moved into meanv
- Mean is calculated, returns to application

Mode

```
mode:
  clr r0
  clr r1
  clr r2
  clr r3
  clr r4
  clr r5
  mov #len,r2
modeloop: mov data(r0),r4
  asl r4
  inc dmode(r4)
  add #2,r0
  sob r2,modeloop
  ;run through again and find max index
  modemaxandmin: mov #mminval,maxv
  mov #mmaxval,minv
  mov #len,r2
  mov #dmode,r0
mdmaxandmin: mov #145,r2
mdmaxandminloop: cmp @r0,maxv
  blt mdmaxandminnotlarge
  mov @r0,maxv
  mov r0,r4
  mov #dmode,r5
  sub r5,r4
  mov r4,r5
  clr r4
  div #2,r4
  mov r4,mdmndx
mdmaxandminnotlarge: cmp @r0,minv
  bgt mdmaxandminnotsmall
  mov @r0,minv
mdmaxandminnotsmall: add #2,r0
  sob r2,mdmaxandminloop
  mov maxv,modev
  return
```

Explanation:

- Registers r0-r5 cleared
- Value of len moved to r2 (counter, val=250)
- Define label 'modeloop'
- Mode data(ro) to r4 (r0 is the index of the array)
- ASL shifts data in r4 to left (multiply by 2)
- INC increments the data in dmode(r4) (increasing the frequency of the number)
- ADD #2,r0 adds 2 to the value of r0, to point at the next element
- SOB checks the value of r2, and branches back to 'modeloop' if the value is not 0
- Define label 'modemaxmin' (this part of the program finds the max value in dmode)
- minv, maxv initialized to default values

- #len moved to r2 (counter)
- r0 points to dmode
- Set r2 to 145 (length of array in octal, counter)
- value pointed to by r0 is compared to maxv
- If less than, branch to ‘mdmaxandminnotlarge’
- value pointed to by r0 is moved to maxv
- Value of r0 moved to r4
- Address of dmode moved to r5
- Subtract r5 from r4 (to calculate the index of the array)
- Move the value of r4 to r5 (to divide)
- r4 is cleared
- divide value of r4 by 2 (r5 is where the data is at)
- Move the value of r4 to mdmndx (mode max index)
- Define label ‘mdmaxandminnotlarge’
- Compare the value pointed to by r0 with minv
- Branch to ‘mdmaxandminnotsmall’ if greater than
- Move what’s pointed to by r0 to minv
- Define lavel mdmaxandminnotsmall
- Add 2 to value of r0 (increment pointer to next data)
- SOB checks value of r2 and branches back to mdmaxandminloop if not 0
- Return to the calling application

Max and Min

```
maxandmin: mov #mminval,maxv
  mov #mmaxval,minv
  mov #len,r2
  mov #data,r0
maxandminloop: cmp @r0,maxv
  blt maxandminnotlarge
  mov @r0,maxv
maxandminnotlarge: cmp @r0,minv
  bgt maxandminnotsmall
  mov @r0,minv
maxandminnotsmall: add #2,r0
  sob r2,maxandminloop
  return
```

Explanation:

- Define label ‘maxandmin’
- Initialize maxv,minv with default values
- Set r2 to len (length of array, = 250 decimal)
- Set r0 to point to data array
- Define label ‘maxandminloop’
- Compare value pointed to by r0 to maxv
- If less than, branch to ‘maxandminnotlarge’
- Move the value pointed to by r0 to maxv
- Define label ‘maxandminnotlarge’
- Compare value pointed to by r0 to minv
- If greater than, branch to ‘maxandminnotsmall’
- Move value pointed to by r0 to minv
- Define label ‘maxandminnotsmall’
- Add 2 to r0 (to point to next element of array)
- SOB branches to maxandminloop if the value of r2 is not 0
- Returns to calling function

Assembly Code

Key for areas of code:
BLUE: Operating System Code
RED: Main Application
PURPLE: Functions for Mean, Mode, Max and Min
BLACK: Data area, Origin 3000 holds all variables for program

```
; demo of simplified interrupt OS
;
;  system calls
;     exit = os executes halt instruction!
;     read = os will read a line from keyboard
;        returning when a newline character read
;     write = os will write a line to teletype
;        returning when line all written (nul character at end)
;    atoi, and itoa - integer<->string conversions
;
; define the operating system calls as trap instructions
exit=104400
readline=104401
writeline=104402
atoi=104403
itoa=104404
;
; data for the trap instruction
trapaddr=34 ; "interrupt entry point" - start address of request
handler
trapsw=36 ; location for status word
opsyssw=40 ; value of status word - priority 1
;
; data for the teleprinter
ttyaddr=64 ; interrupt entry point for tty - set to address of
handler routine
ttysw=66 ; holds status word to load when start handling tty event
tpsw=200 ; value to put in status word - priority 4
tps=177564 ; control register for console output
tpb=177566 ; data register for console output
;
; data for the keyboard
kbdaddr=60 ; interrupt entry point for kbd - set to address of
handler routine
kbdsw=62 ; holds status word to load when start handling kbd event
kbsw=200 ; value to put in status word
kbdc=177560 ; control register for console input
kbdb=177562 ; data register for console input
;
.origin 1000
osstart: mov #os,@#trapaddr
  mov #opsyssw, @#trapsw
  mov #tput,@#ttyaddr
  mov #tpsw,@#ttysw
  mov #kget,@#kbdaddr
  mov #kbsw,@#kbdsw
;
```

```
; need to enable interrupts from keyboard and teletype
; set 'enable' and 'done' in tty
; enable only in kbd
 mov #300,@#tps
 mov #100,@#kbdc
;
; hopefully all is ready
; start the application
  jmp application
; -----------------
; handle keyboard interrupt
kget:movb @#kbdb,ch
   movb ch, @ibufptr
  inc ibufptr
   cmpb #15,ch
   beq ilinedone
   rti
; ilinedone - add the nul byte, set flag saying input ready
ilinedone: clrb @ibufptr
  inc kbdoneflag
  rti
;
; os variables
ibufptr:.blkw 1
kbdoneflag:.blkw 1
ch:.blkw 1
; ------------
; handle teleprinter interrupt
tput:tstb @obufptr
  beq msgdone
; There is another character to go
  movb @obufptr,@#tpb
  inc obufptr
  rti
msgdone:inc printdoneflag
  rti
;
; os variables
obufptr: .blkw 1
printdoneflag: .blkw 1
; --------------------------------------------
; my micro operating system
; I will be using r0 and r1 (and maybe other registers) so save
these
os:mov r0,-(sp)
  mov r1,-(sp)
; find out which request - pick up return address as saved in stack
  mov 4(sp),r1
; program counter has been incremented - take off 2
  dec r1
  dec r1
; r1 should hold the address of the trap instruction
  mov (r1),r0
; r0 now holds the actual trap instruction that was executed
; bottom 8 bits contain request id - (though typically far fewer
; than 255 calls defined)
; clear the top byte
```

```
  bic #177400,r0
; convert index to byte offset
  clc ; just in case its set!
  rol r0
  jmp @osfns(r0)
;
; handle return from os call
; when reach here r0 should hold number of arguments used
; by last os call; need to adjust return address that is on stack
osreturn:clc
  rol r0
  add r0,4(sp)
; and put back registers
  mov (sp)+,r1
  mov (sp)+,r0
  rti
;
; function table for my os
osfns: exitos
  read
  write
  getint
  putint
;
; exit from os
exitos: nop
  nop
  halt
  br exitos ; no escape
; read
;    takes one argument - address of buffer where input to be stored
;    (argument will be in location immediately after trap
instruction)
; uses r0 and r1
; read characters (echoing them to teleprinter) and store until
newline dealt with
; (and add a nul  byte for safety before returning)
; on entry r1 still holding address with trap instruction
read: inc r1
  inc r1
; r1 now holds address that stores address of buffer
; make it store the address of buffer
  mov (r1),ibufptr
  clr kbdoneflag
  inc @#kbdc
; now get wait in OS - interrupt handled keyboard
; will eventually set the 'line done flag'
kbwait:tst kbdoneflag
 bgt kblinedone
 wait
; returns from wait state after interrupt handled
; go back and re-check if line complete
 br kbwait
; finally - the line has been read; can return to user
kblinedone:mov #1,r0
  br osreturn
;
```

```
; write
; set up for interrupt driven output
; (initialize buffer pointer, set done flag to false etc)
write:inc r1
  inc r1
  mov (r1),obufptr
  clr printdoneflag
; send the first character
  movb @obufptr,@#tpb
  inc obufptr
; now wait in os until printdoneflag is set
wrtwait:tst printdoneflag
  bgt olinedone
; nothing to do - it's kind of wait loop
  wait
  br wrtwait
olinedone:mov #1,r0
  jmp osreturn
;
; getint
; processes all digits converting to integer
; will use r3 while doing multiplications (so save and restore)
; assumes only short integers so takes only low order part of
product
; has a local variable (valptr)
getint: mov r3,-(sp)
 inc r1
 inc r1
 mov (r1),valptr
 inc r1
 inc r1
 mov (r1),r1
 clr r3
getintl:cmpb (r1),#60
   blt getintend
   cmpb (r1),#71
   bgt getintend
; character is a decimal digit
   movb (r1)+,r0
  sub #60,r0
; r0 now holds numeric value 0-9 for next decimal digit
  mul #12,r3
 add r0,r3
 br getintl
; result in r3; put it where it should go
getintend: mov r3,@valptr
; replace r3 with saved value
  mov (sp)+,r3
; note use of 2 args
  mov #2,r0
  br osreturn
;
valptr: .word 0
; putint - this is a non-recursive version
; use r2,r3 and same local variable valptr
putint:mov r2,-(sp)
  mov r3,-(sp)
  inc r1
  inc r1
  mov (r1),valptr
  inc r1
  inc r1
  mov (r1),r1
  mov @valptr,r0
  mov #10,r2
; start by filling buffer with spaces
putintfill:movb #40,(r1)+
   sob r2,putintfill
; now generate digits
  tst r0
 bgt nonzero
; simply put 0
 movb #60,-(r1)
 br putintdone
 nonzero: clr r2
  mov r0,r3
putintdiv: tst r3
  beq putintdone
; do a division
  div #12,r2
; remainder in r3 is next value of next digit to go in buffer
  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:
  call mode
  call mean
  itoa
  meanv
  numbuf
  writeline
  msgmean
  writeline
  numbuf
  writeline
  newline
  itoa
  mdmndx
  numbuf
  writeline
  msgmode
  writeline
  numbuf
  writeline
  newline
  call maxandmin
  itoa
  maxv
  numbuf
  writeline
  msgmax
  writeline
  numbuf
  writeline
  newline
  writeline
  msgmin
  itoa
  minv
  numbuf
  writeline
  numbuf
  writeline
  newline
  exit
mean:
  clr r0
  clr r1
  mov #len,r2
meanloop: add data(r0),r1
  add #2,r0
  sob r2,meanloop
  mov r1,r3
  mov r1,r5
  clr r4
  div num,r4
  mov r4,meanv
  return
mode:
  clr r0
  clr r1
  clr r2
  clr r3
  clr r4
  clr r5
  mov #len,r2
modeloop: mov data(r0),r4
  asl r4
  inc dmode(r4)
  add #2,r0
  sob r2,modeloop
  ;run through again and find max index
  modemaxandmin: mov #mminval,maxv
  mov #mmaxval,minv
  mov #len,r2
  mov #dmode,r0
mdmaxandmin: mov #mminval,maxv
  mov #mmaxval,minv
  mov #145,r2
  mov #dmode,r0
```

```
mdmaxandminloop: cmp @r0,maxv
  blt mdmaxandminnotlarge
  mov @r0,maxv
  mov r0,r4
  mov #dmode,r5
  sub r5,r4
  mov r4,r5
  clr r4
  div #2,r4
  mov r4,mdmndx
mdmaxandminnotlarge: cmp @r0,minv
  bgt mdmaxandminnotsmall
  mov @r0,minv
mdmaxandminnotsmall: add #2,r0
  sob r2,mdmaxandminloop
  mov maxv,modev
  return
maxandmin: mov #mminval,maxv
  mov #mmaxval,minv
  mov #len,r2
  mov #data,r0
maxandminloop: cmp @r0,maxv
  blt maxandminnotlarge
  mov @r0,maxv
maxandminnotlarge: cmp @r0,minv
  bgt maxandminnotsmall
  mov @r0,minv
maxandminnotsmall: add #2,r0
  sob r2,maxandminloop
  return
.origin 3000
;constants
mmaxval=77777
mminval=100000
len=372
;String messages
newline: .word 15
msgmean: .string "Mean: "
msgmode: .string "Mode: "
msgmax: .string "Max: "
msgmin: .string "Min: "
;Data
counter: .word 0
meanv: .blkw 1
modev: .blkw 1
maxv: .blkw 1
minv: .blkw 1
mdmndx: .blkw 1
numbuf: .blkw 1
;data for mode tally
dmode: .blkw 145
;modemaxindex: .blkw 1
;modemaxval: .word 0
;actual data from datagen
num: 372
data: .word 5
.word 3
```

```
.word 14
.word 110
.word 26
.word 36
.word 2
.word 13
.word 40
.word 21
.word 1
.word 10
.word 106
.word 7
.word 1
.word 36
.word 47
.word 53
.word 0
.word 72
.word 11
.word 12
.word 15
.word 6
.word 17
.word 63
.word 77
.word 71
.word 117
.word 125
.word 114
.word 131
.word 67
.word 121
.word 63
.word 42
.word 116
.word 104
.word 110
.word 73
.word 113
.word 115
.word 102
.word 73
.word 75
.word 57
.word 77
.word 63
.word 53
.word 74
.word 102
.word 106
.word 110
.word 56
.word 123
.word 47
.word 102
.word 50
.word 111
```

```
.word 123
.word 132
.word 114
.word 102
.word 56
.word 51
.word 62
.word 60
.word 72
.word 62
.word 52
.word 104
.word 53
.word 60
.word 73
.word 103
.word 102
.word 56
.word 102
.word 45
.word 100
.word 117
.word 52
.word 55
.word 116
.word 103
.word 100
.word 73
.word 45
.word 120
.word 120
.word 131
.word 111
.word 114
.word 42
.word 63
.word 45
.word 61
.word 53
.word 37
.word 51
.word 117
.word 124
.word 63
.word 110
.word 47
.word 42
.word 72
.word 73
.word 122
.word 46
.word 63
.word 37
.word 71
.word 65
.word 101
.word 77
```

```
.word 126
.word 124
.word 53
.word 72
.word 52
.word 57
.word 106
.word 72
.word 73
.word 54
.word 52
.word 63
.word 55
.word 67
.word 67
.word 101
.word 104
.word 104
.word 110
.word 66
.word 123
.word 110
.word 74
.word 65
.word 111
.word 136
.word 104
.word 112
.word 46
.word 115
.word 111
.word 104
.word 34
.word 25
.word 105
.word 72
.word 63
.word 76
.word 102
.word 106
.word 34
.word 62
.word 50
.word 65
.word 75
.word 72
.word 27
.word 121
.word 70
.word 60
.word 104
.word 117
.word 120
.word 65
.word 67
.word 66
.word 64
```

```
.word 100
.word 100
.word 73
.word 33
.word 130
.word 74
.word 103
.word 102
.word 64
.word 77
.word 111
.word 44
.word 61
.word 101
.word 55
.word 56
.word 65
.word 52
.word 41
.word 125
.word 100
.word 117
.word 107
.word 76
.word 55
.word 114
.word 50
.word 114
.word 75
.word 42
.word 64
.word 72
.word 71
.word 103
.word 124
.word 115
.word 31
.word 61
.word 50
.word 101
.word 76
.word 70
.word 57
.word 50
.word 53
.word 105
.word 110
.word 110
.word 71
.word 106
.word 47
.word 71
.word 63
.word 72
.word 37
.word 133
.word 115
```

```
.word 112
.word 54
.word 106
.word 133
.word 106
.word 43
.word 103
.word 63
.word 110
.word 67
.word 73
.word 104
.word 134
.word 112
.word 101
.word 77
.word 27
.word 75
.word 112
.word 63
.end osstart
```