**Software Engineering Department**

**Computer Organisation and Programming Course   
final assignment**

**Pocket Calculator application**

**Written by:**

**Racheli Dekel 200864957**

**&**

**Roni Naor 316350479**

**15 August 2021**

**Lecturer: Dr Yigal Hoffner**

TABLE OF CONTENTS

TABLE OF CONTENTS 1

1 Pocket Calculator application design 2

1.1 Extra Work Carried Out (extra 25% items) 2

1.2 Major Design/Implementation Decisions 2

1.3 The High-level Algorithms 3

2 The User Guide 5

3 Program Listing (optional here OR in separate document) 6

# Pocket Calculator application design

This calculator calculates arithmetic actions between two numbers (signed integers).

The user is requested to enter first number->operation (- + / \*)->second number->’=’.

Operations are:

* for subtraction

+ for addition

/ for division

\* for multiplication

The result will be printed to the screen after the user entered ‘=’.

After every calculation, the user is asked to enter Q\q for quitting the program or C\c to continue.

## Extra Work Carried Out

1) Over and underflow: the program checks if the number that was entered is in the range of -32768<=number<=+32767, and if not, the program will print an ERROR.

2) Fast multiplication- the while loop is running only 8 (short int) times and not as many times as the multiplier.

3) Sophisticated input: the input is not separated by enter or space, the calculation is represented in one line.

## Major Design/Implementation Decisions

The following major design decisions were made during the design and implementation phases:

* Number is positive unless a ‘-‘sign was entered ahead of the number and then it will be considered us negative number.
* OP (operator) is a global variable.
* Arithmetic functions and checking overflow and underflow through stack.

## The High-level Algorithms

**Main()**Char op;  
int Num1,Num2;

Printf (*array\_directions)* <”input numbers, operator and ‘=’...”> ;

Printf (*Array\_choice)* <”Press Q/q to quit C/c to continue”> ;

***WLoop***, Printf (*nextStep\_decision)* <”Q/q or C/c?”> ;

Select=Getc();

IF (select==’Q’ || select==’q’)

Then goto ***END***;

FI

IF (select==’C’ || select==’c’)

Then break;

ELSE print *frsterr* <”Invalid selection”> ;

Goto ***WLoop***;

FI

FlagNum=0;

Num1= **inputNumber**();

Num2= **inputNumber**();

IF (OP == ‘+‘)

THEN ADDFUNC(Num1,Num2);

Printf (resultADD);

Goto ***WLoop***;

ELSE IF ( OP == ‘-’)

THEN SUBFUNC(Num1,Num2);

Printf (resultSUB);

Goto ***WLoop***;

ELSE IF ( OP == ‘\*’)

THEN MULTFUNC(Num1,Num2);

Printf (resultMULT);

Goto ***WLoop***;

ELSE IF ( OP == ‘/’)

THEN DIVFUNC(Num1,Num2);

Printf (resultDIV);

Goto ***WLoop***;

***END***;

**//gets from user: number, operator, second number and ‘=’  
Int inputNumber ()**

Int Counterdig=NUM=FLAG=0;

Char chr;

Chr=getc();

***WHILEL***, IF (chr==’+’)

THEN op = ’+’;

Break;

FI

IF (chr==’-’)

THEN IF (Counterdig==0)

THEN FLAG++;

Chr = getc();

Goto ***WHILEL***;

ELSE op = ’-‘;

Break;

FI

IF (chr==’\*’)

THEN op = ‘\*’;

Break;

FI

IF (chr==’/’)

THEN op = ‘/’;

Break;

FI

IF (chr==’=’)

THEN IF (FLAG==1)

THEN NUM= -NUM;

Break;

FI

ELSE IF (chr>=0 && chr<=9)

THEN NUM = NUM \* 10 + chr - 30H;

ELSE printf(“ERROR! <> is not a number”);

Chr=getc();

Goto ***WHILEL***;

FI

Return NUM;

**//fast multiplication  
Void MULTFUNC(NUM1,NUM2)**

Int FLAGMULT1= FLAGMULT= DIGCOUNTMULT= RESULTMULT=0;

Int BREAKCONDMULT=8;

MULTIPLIER=NUM2;

IF(MULTIPLIER<0)

THEN FLAGMULT++;

MULTIPLIER=- MULTIPLIER;

FI

MULTIPLICAND= NUM1;

IF (MULTIPLICAND<0)

THEN FLAGMULT1++;

MULTIPLICAND=- MULTIPLICAND;

WHILE (DIGCOUNTMULT- BREAKCONDMULT<0)

DO

IF (MULTIPLIER%2!=0)

THEN RESULTMULT= RESULTMULT+ MULTIPLICAND;

MULTIPLICAND= MULTIPLICAND%10;

DIGCOUNTMULT++;

OD

IF (FLAGMULT!=0)

THEN RESULTMULT=- RESULTMULT;

Printf(RESULTMULT);

Return;

# The User Guide

1. The user must input valid numbers, or an Error will be printed.
2. The program requests the user to enter first number, operator, second number and sign ‘=’ for result.
3. After every action, the user can quit the program by typing q or Q, or he can continue by typing c or C.
4. Underflow and Overflow: user cannot enter numbers not in the range of:   
   -32768<=number<=+32767, or an error will be printed.
5. No need CR (enter) between every typing.
6. User cannot divide a number by zero, if he will- an error will be printed.
7. If the user entered invalid digit, he does not need to enter everything from the start.

We keep what he entered until the error, and we print it to the screen.

“Instruction message”

“c/C to continue or  
q/Q to quit”

q\Q

c\C

“Result of calculation”

"appropriate ERROR”

An error occurred in the input?

YES

NO

Wanted calculation

# Program Listing

**//Racheli Dekel & Roni Naor**

Main, LDA Array\_directions

BSA print

LDA Array\_choice

BSA print

choice, LDA nextStep\_decision

BSA print

InptLoop, SKI // Repeat

BUN InptLoop // Until (FGI == 1) i.e. i/p available at KB

INP //

EchoM5, SKO // Repeat

BUN EchoM5 // Until (FGO == 0) i.e. i/p available at KB

OUT

STA select

ADD MinusQ

SZA

BUN nextt1

BUN EndMain

nextt1, LDA select

ADD MinusSq

SZA

BUN nextt2

BUN EndMain

nextt2, LDA select

ADD MinusC

SZA

BUN nextt3

BUN Mstart

nextt3, LDA select

ADD MinusSc

SZA

BUN nextt4

BUN Mstart

nextt4, LDA frsterr

BSA print

BUN choice

Mstart, CLA

STA flagNum

LDA operation\_message

BSA print

CLA

STA Optemp

BSA InputNumber

STA num1

LDA flag

SZA

BUN check2

BUN check1

check1, LDA num1

INC

SPA

BUN end1

bun cont1

check2, LDA num1

ADD minusOne

SNA

BUN end2

BUN cont1

cont1, LDA num1

BSA PUSH

ContinueM, ISZ flagNum

BSA InputNumber

STA num2

LDA flag

SZA

BUN check4

BUN check3

check3, LDA num2

INC

SPA

BUN end1

BUN cont2

check4, LDA num2

ADD minusOne

SNA

BUN end2

BUN cont2

cont2, LDA num2

BSA PUSH

LDA Op

ADD checkPlus // if (op==+)

SZA

BUN Next1

BSA ADDFUNC

BUN choice // then goto ADDFUNC

Next1, LDA Op

ADD checkMinus // else

SZA // if (op==-)

BUN Next2 // then goto SUBFUNC

BSA SUBFUNC

BUN choice

Next2, LDA Op // else

ADD checkMuL // if (op==\*)

SZA // then goto MULTFUNC

BUN Next3

BSA MULTFUNC

BUN choice

Next3, BSA DIVFUNC // else goto DIVFUNC

CLA

BUN choice

end1, LDA prompt\_overflow

BSA print

CLA

BUN Mstart

end2, LDA prompt\_underrflow

BSA print

CLA

BUN Mstart

EndMain, HLT

**//Gets number as input from user, prints error if invalid, then gets operator as input and returns the number.**

**// The OP is global  
//sends back from main and gets the second number from user- if invalid, prints error**

**InputNumber**, HEX 0

CLA

STA Flag // Flag =0;

STA Counterdig

STA num // num = 0;

Digit\_InputLoop, SKI // Repeat

BUN Digit\_InputLoop // Until (FGI == 1) i.e. i/p available at KB

INP //

STA chr

LDA Optemp

SZA

BUN notzero1

LDA chr //

ADD checkPlus // While ( (chr = inputchar()) != CR)

SZA

BUN ContinueCheck1

LDA chr

STA OP

BUN End\_loopL //

ContinueCheck1, LDA chr // DO

ADD checkMinus //

SZA // IF (cc == Sign\_asciiL)

BUN ContinueCheck2

LDA Counterdig

Sza

Bun cont

Bun yesmin

cont, LDA chr

STA OP

BUN End\_loopL

ContinueCheck2, LDA chr

ADD checkMuL

SZA

BUN ContinueCheck3

LDA chr

STA OP

BUN End\_loopL

ContinueCheck3, LDA chr

ADD checkDiV

SZA

BUN ContinueL

LDA chr

STA OP

BUN End\_loopL

notzero1, LDA chr

ADD checkEqual2

SZA

BUN ContinueCheck4

BUN End\_loopL

ContinueCheck4, LDA chr

ADD checkMinus

SZA

BUN ContinueL

Yesmin, ISZ Flag // THEN Flag++;

LDA Sign\_asciiL

EchoL1, SKO

BUN EchoL1

OUT

BUN Digit\_InputLoop

ContinueL, LDA chr //

EchoL, SKO // Repeat

BUN EchoL // Until (FGO == 0) i.e. i/p available at KB

OUT //

Numcheck, LDA chr

ADD MAsciimax

SNA // IF (chr > Asciimax)

BUN Error // THEN printf("error");

BUN Numcheck2

Numcheck2, LDA chr

ADD MAsciimin // IF(chr < MAsciimin)

SPA //

BUN Error

BUN ContinueL1 // THEN printf("error");

Error, LDA chr

STA Errorptrnum I

LDA Error\_digit\_print

BSA print

LDA operation\_message //>>

BSA PRINT

LDA flagNum

SZA

BUN secondNum

Bun firstNum

secondNum, LDA NUM1

BSA PutSignedInt

LDA op

Bsa PUTC

LDA FLAG  
 SPA

BUN firstNum   
LDA Sign\_asciiL  
BSA PUTC

firstNum, LDA Counterdig

SPA

BUN StartOver

counterPOS, LDA NUM

BSA PutSignedInt

BUN StartOver

ContinueL1, isz Counterdig

LDA num //

CLE

CIL //

CIL //

CIL //

ADD num //

ADD num //

ADD chr //

ADD Minus30 //

STA num // ELSE num = num \* 10 + chr - 30H;

// FI;

StartOver, BUN Digit\_InputLoop // OD

End\_loopL, LDA chr

STA Optemp

EchoL2, SKO // Repeat

BUN EchoL2 // Until (FGO == 0) i.e. i/p available at KB

OUT //

LDA Flag // IF (Flag ==1)

SZA //

BUN TurnMinus //

BUN ContinueL2 //

TurnMinus, LDA num // THEN num = num \* -1;

CMA //

INC //

STA num //

// FI;

ContinueL2, LDA num //

BUN InputNumber I // return (Num);

**//MainData**

flagNum, hex 0

Counterdig, hex 0

minusOne, HEX -1

num1, DEC 0

num2, DEC 0

select, DEC 0

MinusQ, HEX FFAF //-Q

MinusSq, HEX FF8F //-q

MinusC, HEX FFBD //-C

MinusSc, HEX FF9D //-c

**//inputNumData**

op, HEX 0

Optemp, DEC 0

Num, DEC 0

chr, DEC 0

Minus30, HEX -30

Sign\_asciiL, HEX 2D //-

MAsciimax, HEX FFC6 //Uplimit>9

MAsciimin, HEX FFD1 //Downlimit<0

checkPlus, HEX FFD5 //-+

checkMinus, HEX FFD3 //--

checkMuL, HEX FFD6 //-\*

checkDiV, HEX FFD1 //-/

checkEqual2, HEX FFC3 //-=

Errorptrnum, HEX 758

Flag, DEC 0

**//function that prints every char in array until terminator:0**

print, HEX 0

STA LPTR

LpLoop, LDA LPTR I

SZA

BUN Output\_Loop

BUN LpEndLoop

Output\_Loop, SKO

BUN Output\_Loop

OUT

ISZ LPTR

BUN LpLoop

LpEndLoop, BUN print I

//printData

LPTR, HEX 0

**//function that prints signed int number**

PutSignedInt, HEX 0

STA Number

LDA Power

STA PowerPtr

LDA LLZero

STA flagint

LDA Number

AND Mask //

SZA // IF ((Number && Mask) # 0)

BUN MinusL //

BUN Continue //

MinusL, LDA Sign\_ascii // THEN

BSA Puts2 // output ("-");

LDA Number //

CMA //

INC //

STA Number // Number = Number \* -1;

// FI;

Continue, CLA //

STA Count // Count = 0;

Out\_Loop, LDA Count //

ADD Minusfour //

SNA // While (Count < 4)

BUN End\_LoopL2 //

CLA // DO

STA Digit // Digit = 0;

LDA PowerPtr I //

STA Divisor // Divisor = \* PowerPtr;

Dividing, LDA Divisor //

CMA //

INC //

ADD Number //

SNA // While (Number - Divisor >= 0)

BUN Continue2 //

BUN Zerocheck //

Continue2, STA Number // DO Number = Number - Divisor;

ISZ Digit // Digit++;

BUN Dividing // OD

Zerocheck, LDA Flagint

SZA

BUN Endloop2

LDA Digit

SZA

BUN BeforeEloop2

BUN Zero

BeforeEloop2, ISZ Flagint

Endloop2, LDA Digit

ADD ascii\_Offset //

BSA Puts2 // Output(Digit);

Zero, ISZ PowerPtr // PowerPtr++;

ISZ Count // Count++;

BUN Out\_Loop // OD

End\_LoopL2, LDA Number //

ADD ascii\_Offset //

BSA Puts2 // Output(Number);

BUN putSignedInt I

// PutSignedIntData

Number, DEC 0

Minusfour, DEC -4

Count, DEC 0

Digit, DEC 0

Divisor, DEC 0

Power, HEX 180

PowerPtr, HEX 0

Sign\_ascii, HEX 2D // the "-" character

ascii\_Offset, HEX 30

LLCR, HEX D

Mask, HEX 8000

Flagint, DEC 0

LLZero, DEC 0

// digit to ascii representation offset

ORG 180

Power10, DEC 10000

DEC 1000

DEC 100

DEC 10

**// prints to screen what it receives**

Puts2, HEX 0

Output\_Loop2, SKO

BUN Output\_Loop2

OUT

BUN Puts2 I

**//ADD function, calculates sum between 2 numbers, and sending the result to print function**

ADDFUNC, HEX 0

BSA POP

STA ADDVAR1

BSA POP

STA ADDVAR2

CLE

ADD ADDVAR1

STA ResultADD

LDA ADDVAR1

BSA PUSH

LDA ADDVAR2

BSA PUSH

LDA ResultADD

BSA PUSH

BSA OverUnder

SZA

BUN ADDEND

LDA Print\_EqualSign

BSA print

LDA ResultADD

BSA PutSignedInt

ADDEND, BUN ADDFUNC I

**//AddFuncDATA**

ADDVAR1, DEC 0

ADDVAR2, DEC 0

ResultADD, DEC 0

**//SUB function, subtracting between 2 numbers, and sending the result to print function**

SUBFUNC, HEX 0

BSA POP

CMA

INC

STA SUBVAR1

BSA POP

STA SUBVAR2

ADD SUBVAR1

CLE

STA resultSUB

LDA SUBVAR1

BSA PUSH

LDA SUBVAR2

BSA PUSH

LDA resultSUB

BSA PUSH

BSA OverUnder

SZA

BUN SUBEND

LDA Print\_EqualSign

BSA print

LDA resultSUB

BSA PutSignedInt

SUBEND, BUN SUBFUNC I

**//SubFuncDATA**

SUBVAR1, DEC 0

SUBVAR2, DEC 0

resultSUB, DEC 0

**//checks if there's overflow or underflow, prints Over/Under correspondingly**

//returns: 1 if there is over\under flow   
// 0 no over\under flow- prints the number

OverUnder, HEX 0

BSA POP

STA R

BSA POP

STA Y

BSA POP

STA X

CLA

STA OverflowF

STA UnderflowF

STA V

LDA X

CIL

CLA

CIL

STA MSB\_X

LDA Y

CIL

CLA

CIL

CMA

INC

ADD MSB\_X

SZA

BUN Exit

Test\_Rmsb, LDA R

CIL

CLA

CIL

STA MSB\_R

CMA

INC

ADD MSB\_X

SZA

BUN Set\_OUverflow

BUN Exit

Set\_OUverflow, LDA OUFlowFlag

STA V

LDA MSB\_R

ADD Minus\_1

SZA

BUN UnderFLow

OverFlow, LDA OUFlowFlag

STA OverflowF

BUN Printstage

UnderFlow, LDA OUFlowFlag

STA UnderflowF

Printstage, LDA OverflowF

SZA

BUN Overprint

LDA UnderflowF

SZA

BUN Underprint

BUN Exit

Overprint, LDA prompt\_overflow

BSA print

BUN Exit

Underprint, LDA prompt\_underrflow

BSA print

BUN Exit

Exit, LDA V

BUN OverUnder I

**// Over\Under flowData**

Minus\_1, DEC -1

X, DEC -32768

Y, DEC -1

R, DEC 0

MSB\_X, DEC 0 // temp storage for msb of X

MSB\_R, DEC 0 // temp storage for msb of R

OUFlowFlag, DEC 1

V, DEC 0

OverflowF, DEC 0 // overflow Flag

UnderflowF, DEC 0 // underflow Flag

//multiplication function, multiplying between 2 numbers, sends the result to a printing function

MULTFUNC, HEX 0

CLA

CLE

STA RESULTMULT //RESULTMULT=0

STA DIGCOUNTMULT //DIGCOUNTMULT=0

STA FLAGMULT //FLAGMULT=0

BSA POP

STA MULTIPLIER // MULTIPLIER=num2

CIL

SZE

BUN NEG1 // if(MULTIPLIER<0)

BUN MULTCONT1

NEG1, ISZ FLAGMULT // then FLAGMULT++

LDA MULTIPLIER

CMA

INC

STA MULTIPLIER // MULTIPLIER=- MULTIPLIER

MULTCONT1, BSA POP // fi

STA MULTIPLICAND // MULTIPLICAND=num1

CLE

CIL

SZE

BUN NEG2 // if (MULTIPLICAND<0)

BUN LOOPMULT

NEG2, ISZ FLAGMULT1 // then FLAGMULT1++

LDA FLAGMULT1

CLE // fi

LOOPMULT, LDA BREAKCONDMULT //

CMA //

INC //

ADD DIGCOUNTMULT //

SNA // while (DIGCOUNTMULT- BREAKCONDMULT<0)

BUN MULTEND // do

CLE //

LDA MULTIPLIER // shift right(CIR) MULTIPLIER

CIR //

STA MULTIPLIER //

SZE //

BUN MULTIT // if (e==1)

BUN MULTCONT2 // then

MULTIT, LDA RESULTMULT // RESULTMULT= RESULTMULT+MULTIPLICAND

ADD MULTIPLICAND //

STA RESULTMULT //

MULTCONT2, LDA MULTIPLICAND //

CIL //

STA MULTIPLICAND // shift left(cil) MULTIPLICAND

ISZ DIGCOUNTMULT // DIGCOUNTMULT++

BUN LOOPMULT // od

MULTEND, LDA FLAGMULT //

SZA // if (FLAGMULT!=0)

BUN RIS!R //

BUN RISR //

RIS!R, LDA RESULTMULT //

CMA //

INC //

STA RESULTMULT // then RESULTMULT=- RESULTMULT

//Send it back in

RISR, LDA Print\_EqualSign // else print’=’

BSA print

LDA RESULTMULT //

BSA PutSignedInt

BUN MULTFUNC I // return

**//MULTFUNC DATA**

MULTIPLIER, DEC 0

MULTIPLICAND, DEC 0

RESULTMULT, DEC 0

DIGCOUNTMULT, DEC 0

BREAKCONDMULT, DEC 8 //short int

FLAGMULT, DEC 0

FLAGMULT1, DEC 0

**// Division function, diving between 2 numbers, returns the result functions and calculates the remainder**

DIVFUNC, HEX 0

LDA ERRHEAD

STA ERRPTR // ERRPTR=0

CLA

CLE

STA FLAGDEV

STA RESULTDEV

BSA POP

STA DEVISOR

BSA POP

STA DEVIDENT

LDA DEVIDENT

CIL

SZE

BUN NEGDEV1 //if(DEVIDENT<0)

BUN DEVCONT1

NEGDEV1, LDA DEVIDENT //then

CMA

INC

STA DEVIDENT // DEVIDENT=- DEVIDENT

ISZ FLAGDEV // FLAGDEV++

DEVCONT1, CLE //else

LDA DEVISOR

SZA

BUN DEVCONT2 // if( DEVISOR==0)

ERRLOOP, LDA Divide\_error

BSA print // then

BUN DIVFUNC I // printf(“error:division by 0)

DEVCONT2, CIL // else

SZE

BUN NEGDEV2 // if (DEVISOR<0)

BUN DEVLOOP

NEGDEV2, LDA DEVISOR // then

CMA

INC

STA DEVISOR // DEVISOR=- DEVISOR

LDA FLAGDEV

SZA

BUN DFIS!0 // if (FLAGDEV!=0)

BUN DFIS0

DFIS!0, CLA

STA FLAGDEV // then FLAGDEV=0

BUN DEVLOOP

DFIS0, ISZ FLAGDEV // else FLAGDEV++

DEVLOOP, LDA DEVISOR // else

CMA

INC

ADD DEVIDENT // while(DEVISOR- DEVISOR>=0)

SNA

BUN DEVCONT3

BUN DEVEND // DO

DEVCONT3, STA DEVIDENT // DEVIDENT= DEVISOR- DEVISOR

ISZ RESULTDEV // RESULTDEV++

BUN DEVLOOP // OD

DEVEND, LDA FLAGDEV // if(FLAGDEV!=0)

SZA

BUN DEVRIS!R

BUN DEVRISR

DEVRIS!R, LDA RESULTDEV //then

CMA

INC

STA RESULTDEV // RESULTDEV=- RESULTDEV

//return result

DEVRISR, LDA Print\_EqualSign //else

BSA print

LDA RESULTDEV // print DEVresult

BSA PutSignedInt

LDA Remainder

BSA print

LDA DEVIDENT

BSA PutSignedInt

BSA DIVFUNC I

//Divfunc data

ERRHEAD, HEX 600

ERRPTR, DEC 0

FLAGDEV, DEC 0

RESULTDEV, DEC 0

DEVIDENT, DEC 0

DEVISOR, DEC 0

//Stack functions

PUSH, HEX 0

STA SPtr I

ISZ SPtr

BUN Push I

POP, HEX 0

LDA SPtr

CMA

INC

CMA

STA SPtr

LDA SPtr I

BUN POP I

//Stack Data

SPtr, HEX 300

ORG 300

Stack, DEC 0

DEC 0

DEC 0

DEC 0

DEC 0

StackEnd, DEC 0

PUTC, HEX 0 // char putc(char cc)  
Output, SKO // Repeat   
 BUN Output //   
 OUT // Printer = AC  
 BUN PUTC I // return

**//Arrays of all printed strings- every string terminated by 0**

Array\_directions, HEX 400

ORG 400

HEX 49 //I

HEX 6e //n

HEX 73 //s

HEX 65 //e

HEX 72 //r

HEX 74 //t

HEX 20

HEX 66 //f

HEX 69 //i

HEX 72 //r

HEX 73 //s

HEX 74 //t

HEX 20

HEX 6e //n

HEX 75 //u

HEX 6d //m

HEX 62 //b  
 HEX 65 //e

HEX 72 //r

HEX 2d //-

HEX 3e //>

HEX 6f //o

HEX 70 //p

HEX 65 //e

HEX 72 //r

HEX 61 //a

HEX 74 //t

HEX 6f //o

HEX 72 //r

HEX 2d //-

HEX 3e //>

HEX 73 //s

HEX 65 //e

HEX 63 //c

HEX 6f //o

HEX 6e //n

HEX 64 //d

HEX 20

HEX 6e //n

HEX 75 //u

HEX 6d //m

HEX 62 //b

HEX 65 //e

HEX 72 //r

HEX 20

HEX 61 //a

HEX 6e //n

HEX 64 //d

HEX 20

HEX 27 //’

HEX 3d //=

HEX 27 //’

HEX 20

HEX 66 //f

HEX 6f //o

HEX 72 //r

HEX 20

HEX 72 //r

HEX 65 //e

HEX 73 //s

HEX 75 //u

HEX 6c //l

HEX 74 //t

HEX 0D

HEX 0

Array\_choice , HEX 450

ORG 450

HEX 50 //P

HEX 72 //r

HEX 65 //e

HEX 73 //s

HEX 73 //s

Hex 20

HEX 51 // Q

HEX 2f // /

HEX 71 // q

HEX 20

HEX 54 // t

HEX 6F // o

HEX 20

HEX 71 // q

HEX 75 // u

HEX 69 // i

HEX 74 // t

HEX 20

HEX 6F // o

HEX 72 // r

HEX 20

HEX 43 // C

HEX 2f // /

HEX 63 // c

HEX 20

HEX 74 // t

HEX 6F // o

HEX 20

HEX 63 // c

HEX 6F // o

HEX 6e // n

HEX 74 // t

HEX 69 // i

HEX 6E // n

HEX 75 // u

HEX 65 // e

HEX 20

HEX 0D

HEX 0

Prompt\_Oprtr\_Err, HEX 610

ORG 610

HEX D

HEX 45 //E

HEX 72 //R

HEX 72 //R

HEX 6F //O

HEX 72 //R

HEX 21 //!

HEX 20

HEX 49 //I

HEX 6E //n

HEX 76 //v

HEX 61 //a

HEX 6C //l

HEX 69 //i

HEX 64 //d

HEX 20

HEX 69 //i

HEX 6E //n

HEX 70 //p

HEX 75 //u

HEX 74 //t

HEX D

HEX 0

Error\_digit\_print, HEX 750

ORG 750

HEX 0D

HEX 45 //E

HEX 72 //R

HEX 72 //R

HEX 6F //O

HEX 72 //R

HEX 21 //!

HEX 20

HEX -1 // Errorptrnum is in 758 address and the digit is there

HEX 20

HEX 69 //i

HEX 73 //s

HEX 20

HEX 6E //n

HEX 6F //o

HEX 74 //t

HEX 20

HEX 61 //a

HEX 20

HEX 6e //n

HEX 75 //u

HEX 6d //m

HEX 62 //b

HEX 65 //e

HEX 72 //r

HEX D

HEX 0

Print\_EqualSign, HEX 850

ORG 850

HEX 20

HEX 0

prompt\_overflow, HEX 860

ORG 860

HEX 0D

HEX 45 //E

HEX 72 //R

HEX 72 //R

HEX 6F //O

HEX 72 //R

HEX 21 //!

HEX 20

HEX 4F //O

HEX 76 //v

HEX 65 //e

HEX 72 //r

HEX 66 //f

HEX 6C //l

HEX 6F //o

HEX 77 //w

HEX D

HEX 0

prompt\_underrflow, HEX 910

ORG 910

HEX 0D

HEX 45 //E

HEX 72 //R

HEX 72 //R

HEX 6F //O

HEX 72 //R

HEX 21 //!

HEX 20

HEX 55 //U

HEX 6E //n

HEX 64 //d

HEX 65 //e

HEX 72 //r

HEX 66 //f

HEX 6C //l

HEX 6F //o

HEX 77 //w

HEX D

HEX 0

Divide\_error, HEX 940

ORG 940

HEX 0D

HEX 45 //E

HEX 72 //R

HEX 72 //R

HEX 6F //O

HEX 72 //R

HEX 21 //!

HEX 20

HEX 44 //D

HEX 69 //i

HEX 76 //v

HEX 69 //i

HEX 73 //s

HEX 69 //i

HEX 6f //o

HEX 6e //n

HEX 20

HEX 62 //b

HEX 79 //y

HEX 20

HEX 30 //0

HEX 0D

HEX 0

Remainder, HEX 960

ORG 960

HEX 0D

HEX 52 //R

HEX 65 //e

HEX 6D //m

HEX 61 //a

HEX 69 //i

HEX 6E //n

HEX 64 //d

HEX 65 //e

HEX 72 //r

HEX 3A //:

HEX 20

HEX 0

frsterr, HEX 990

ORG 990

HEX D

HEX 45 //E

HEX 72 //R

HEX 72 //R

HEX 6F //O

HEX 72 //R

HEX 21 //!

HEX 20

HEX 49 //I

HEX 6E //n

HEX 76 //v

HEX 61 //a

HEX 6C //l

HEX 69 //i

HEX 64 //d

HEX 20

HEX 73 //s

HEX 65 //e

HEX 6C //l

HEX 65 //e

HEX 63 //c

HEX 74 //t

HEX 69 //i

HEX 6F //o

HEX 6E //n

HEX 20

HEX D

HEX 0

nextStep\_decision, HEX A30

ORG A30

HEX D

HEX 51 // Q

HEX 2f // /

HEX 71 // q

HEX 20

HEX 6F // o

HEX 72 // r

HEX 20

HEX 43 // C

HEX 2f // /

HEX 63 // c

HEX 20

HEX 3f // ?

HEX 20

HEX 0

operation\_message, HEX A60

ORG A60

HEX D

HEX 3e // >

HEX 3e // >

HEX 20

HEX 0