high level algorithm

main(){

int num=0;

int num1=0;

(\*func\_ptr) operator1\_ptr=&addition;

(\*func\_ptr) operator2\_ptr;

while(cc != ‘q’){

if(cc>2f && cc<3a){ //if it’s a digit

num=multy\_by\_10(num) +(cc – 30);

flag =0;

}

else if(cc>29 && cc<30){ //if its operator

switch (cc) {

case 2a:

operator2\_ptr = &multiplication();

break;

case 2b:

operator2\_ptr = &addition();

break;

case 2c:

printf(“invalid input”);

num=0; //initialise variables

num1=0; //

operator1\_ptr=&addition(); //

break;

case 2d:

operator2\_ptr = &subraction();

break;

case 2e:

printf(“invalid input”);

num=0; //initialise variables

num1=0; //

operator1\_ptr=&addition(); //

break;

case 2f:

operator2\_ptr = &division();

}

if(flag==1) //if user pressed an operator twice

operator1\_ptr=operator2\_ptr;

else{

num2=num;

num1= operator1\_ptr(num1,num2);

operator1\_ptr= operator2\_ptr;

}

flag=1;

num=0;

}

Else if(cc == ‘=’){

num2=num;

if(flag==1) //if user pressed “=” right after operator

printf(operator1\_ptr(num1,num1));

else

printf(operator1\_ptr(num1,num));

num=0; //initialise variables

num1=0; //

operator1\_ptr=addition; //

}

Else {

printf(“invalid input”);

num=0;

num1=0;

operator1\_ptr=&addition();

}

}

Printf(“good bye”);

}

int multiplication(int num\_1, int num\_2){

int temp=num1;

num1=0;

while(num2 != 0){

num1=num1+temp;

num- - ;

}

return num1;

}

int addition(int num1, int num2){

return num1+num2;

}

int subtraction(int num1,int num2){

return num1-num2

}

(int,int)division (int num1 ,int num2){

If(num2 == 0)

Printf(“math error”);

return ;

If(num1<0 ){

num1=-num1;

return –division(num1,num2);

}

int counter=0;

while (num1>0){

num1=num1-num2;

if(num1>= 0)

counter ++;

}

If(num1 == 0) //it means that the remainer is 0

return (counter,num1);

return (counter,num1+num2); //else – the remainer is num1 + num2

}

printf(String[ ]) {

S\_ptr = &String

WHILE (\*S\_ptr != ’\0’)

Do

output(char);

S\_ptr ++;

END

}

transformations:

main(){

start, num=0;

num1=0;

num2;

operator1\_ptr=&addition();

operator2\_ptr;

while, IF(cc-q==0)

THEN printf(“good bye”)

ELSE IF(cc-2F > 0)

THEN IF(cc-3A < 0)

THEN num=multy\_by\_10(num)+(cc-30);

flag =0;

FI

ELSE IF(cc -29 > 0)

THEN DO IF(cc -2A == 0)

THEN operator2\_ptr = & multiplication ();

FI

IF(cc-2B == 0)

THEN operator2\_ptr = &addition();

FI

IF(cc-2C == 0)

THEN printf(“invalid input”)

Go to start;

FI

IF(cc-2D ==0)

THEN operator2\_ptr =&subtraction()’

FI

IF(cc-2E == 0)

THEN printf(“invalid input”);

Go to start;

FI

IF(cc-2F == 0)

THEN operator2\_ptr = &division;

FI

IF(flag === 1)

THEN operator1\_ptr=operator2;

ELSE num2=num;

num1= operator1(num1,num2);

operator1= operator2;

num=0;

flag=1;

FI

OD

ELSE IF (cc-3D == 0)

THEN DO num2=num;

IF (flag-1 == 0)

THEN printf(operator1\_ptr(num1,num1));

ELSE printf(operator1\_ptr(num1,num));

FI

num=0;

num1=0;

operator1\_ptr=&addition();

OD

ELSE DO printf(“invalid input”);

num=0;

num1=0;

operator1\_ptr=&addition();

OD

FI

FI

FI

FI

Printf(“good bye”);

Print\_string(String[ ]) {

S\_ptr = &String;

IF (\*S\_ptr – 00 == 00)

THEN END

ELSE DO

output(char);

S\_ptr ++

}

print\_num(number) {

IF (number <0)

THEN number = *-* number;

output(‘-‘) // minus sign

FI

WHILE (Count ≥ 0)

DO

digit =0;

WHILE (number – 10^Count > 0)

DO

number = number – 10^Count; //10^Count will be in array

digit++;

OD

END

Put\_char(digit);

Count—

OD

END;

Put\_char(last digit);

}

put\_char(char){

out(char);

}

}

Addition(int num1,int num2){

Result = num1 + num2 (in 2’s complement)

IF ((num1)msb == (num2)msb)

THEN IF ((num1)msb != (Result)msb)

THEN printf(“Math error”) //over/under flow

FI

FI

}

subtraction(int num1,int num2){

num2=-num2;

return addition(num1,num2);

}

Multiplication(int num1, int num2){

IF(num1 == 0)

THEN return 0;

ELSE IF(num2 == 0)

THEN return 0;

ELSE Result = 0;

FOR (16 binary digits in multiplier)

DO

IF (digit == 1)

THEN result = result + multiplicand;

FI

Shift(multiplicand) 1 place to left; // multiply by 2

OD;

END

FI

FI

}

(int,int) division (int num1 ,int num2){

int remainer=0;

int counter=0;

flag2=0;

IF(num2 == 0)

THEN printf("math error");

Go to start.

ELSE IF(num1<0 )

THEN num1= -num1;

Flag=1;

FI

WHILE (num1>0)

DO num1=num1-num2;

IF (num1>=0)

THEN counter ++;

FI

OD

IF(num1==0)

THEN remainer=0;

ELSE remainer= num1+num2;

FI

FI

IF (flag==1)

THEN Counter= -counter;

FI

Return(counter,remainer);

}

The assembly code

Start, CLA // main(){

STA num // int num=0;

STA num1 // int num1=0;

LDA addition\_ptr //

STA operator1\_ptr // (\*func\_ptr) operator1\_ptr=&addition;

LDA initialize\_stack //

STA ptr //

Main\_while,           SKI //

BUN Main\_while //

INP // while(cc != ‘q’){

Echo1, SKO //

BUN Echo1 //

OUT //

STA cc //

ADD Minus\_q //

SZA // if(cc>2f && cc<3a){

BUN check\_if\_digit //

LDA good\_bye\_ptr //

BSA Print\_string //

BUN end\_program //

check\_if\_digit, LDA cc //

ADD Minus\_2F //

SPA //

BUN check\_if\_operator //

LDA cc //

ADD Minus\_3A //

SNA //

BUN check\_if\_operator //

Its\_a\_digit, LDA num //

BSA multy\_by\_10 //

ADD cc //

ADD Minus\_30 //

STA num // num=multy\_by\_10(num) +(cc – 30);

ADD check\_over\_flow //

SPA // if(num-32767>0)

BUN Number\_ok //

BUN MathError // printf(“math error”);

Number\_ok, CLA //

STA flag // flag =0;

BUN Main\_while     // }

check\_if\_operator, LDA cc // else

ADD Minus\_29 //

SPA // if(cc>29 && cc<30){

BUN check\_if\_equal //

LDA cc //

ADD Minus\_30 //

SNA //

BUN check\_if\_equal //

LDA cc //

ADD Minus\_2A //

SZA // switch (cc) {

BUN next1 // case ‘\*’:

LDA multiplication\_ptr //

STA operator2\_ptr // operator2\_ptr = &multiplication();

BUN check\_flag // break;

next1, LDA cc //

ADD Minus\_2B //

SZA // case ‘+’:

BUN next2 //

LDA addition\_ptr //

STA operator2\_ptr // operator2\_ptr = &addition();

BUN check\_flag // break;

next2, LDA cc //

ADD Minus\_2C //

SZA // case ‘,’ :

BUN next3 // printf(“invalid input”);

BUN invalid // go to start;

next3, LDA cc //

ADD Minus\_2D //

SZA // case ‘-‘ :

BUN next4 //

LDA subtraction\_ptr //

STA operator2\_ptr // operator2\_ptr = &subraction();

BUN check\_flag // break;

next4, LDA cc //

ADD Minus\_2E //

SZA // case ‘.’ :

BUN next5 // printf(“invalid input”);

BUN invalid // go to start;

next5, LDA cc //

ADD Minus\_2F //

SZA // case ‘/’ :

BUN invalid //

LDA division\_ptr //

STA operator2\_ptr // operator2\_ptr = &division();

check\_flag, LDA flag // }

ADD Minus\_one //

SZA // if(flag!=1){

BUN flag\_is\_0 //

BUN next6 //

flag\_is\_0, LDA num //

STA num2 // num2=num;

LDA num1 //

BSA PUSH //

LDA num2 //

BSA PUSH //

BSA operator1\_ptr I //

STA num1 // num1= operator1\_ptr(num1,num2);

next6, LDA operator2\_ptr // operator1\_ptr= operator2\_ptr;

// }

STA operator1\_ptr // else{ operator1\_ptr=operator2\_ptr;

CLA //

STA num // num=0;

LDA one //

STA flag // flag=1;

BUN Main\_while // }

check\_if\_equal, LDA cc // Else

ADD Minus\_3D //

SZA // if(cc == ‘=’){

BUN invalid //

LDA num //

STA num2 // num2=num;

LDA flag //

ADD Minus\_one //

SZA // if(flag==1)

BUN regular\_result //

LDA num1 //

BSA PUSH //

LDA num1 //

BSA PUSH //

BSA operator1\_ptr I // printf(operator1\_ptr(num1,num1));

BSA Print\_num //

BUN Start //

regular\_result, LDA num1 // else

BSA PUSH //

LDA num //

BSA PUSH //

BSA operator1\_ptr I // printf(operator1\_ptr(num1,num));

BSA Print\_num // go to start;

BUN Start // }

Invalid, LDA invalid\_input\_ptr // Else printf(“invalid input”);

BSA Print\_string // go to start;

BUN Start // }

end\_program, HLT //}

//main data //

cc, HEX 00 //input character

num, DEC 0 //

num1, DEC 0 //

num2, DEC 0 //

flag, DEC 0 //to indicate if pressed operator before

operator1\_ptr, HEX 00 //function pointer1

operator2\_ptr, HEX 00 // function pointer1

one, DEC 1 //

Minus\_30, HEX -30 //ascii offset

Minus\_one, DEC -1 //

Minus\_q, HEX -71 //” q “

Minus\_2A, HEX -2A //” \* “

Minus\_2B, HEX -2B //” + “

Minus\_2C, HEX -2C //” , “

Minus\_2D, HEX -2D //” – “

Minus\_2E, HEX -2E //” . “

Minus\_2F, HEX -2F //” / “

Minus\_3A, HEX -3A //numbers upper limit

Minus\_29, HEX -29 //operators lower limit

Minus\_3D, HEX -3D //” = “

initialize\_stack, HEX 200 //

check\_over\_flow, DEC -32767 //max input

//multy by 10 subroutine

multy\_by\_10, HEX 0 //AUX\_int Multy(int num){

STA X //

CLE //

CIL //

CIL //

CIL //

ADD X //

ADD X // X = X\*10;

BUN multy\_by\_10 I // return X;

//sub data

X, DEC 0 //}

// Stack operations Push & Pop

POP, HEX 0 // POP() {

LDA Ptr // Ptr--;

ADD Minus\_one //

STA Ptr //

LDA Ptr I // AC = [Ptr];

BUN POP I // return

// //}

PUSH, HEX 0 // PUSH(AC) {

STA Ptr I // [Ptr] = AC;

ISZ Ptr // Ptr++;

BUN PUSH I // return

// Stack data //}

Ptr, HEX 200 //

ORG 200 //

Stack, DEC 0 // initialize 2 locations

DEC 0 //

DEC 0 //

//

// print string Subroutine

Print\_string, HEX 0 // print\_string(String[ ]) {

STA S\_ptr // S\_ptr = &String

while\_loop, LDA S\_ptr I // WHILE (\*S\_ptr != ’\0’)

SZA //

BUN Out\_loop // DO

BUN end\_loop1 //

Out\_loop, SKO //

BUN Out\_loop //

OUT // output(char);

ISZ S\_ptr // S\_ptr ++;

BUN while\_loop // END;

end\_loop1, BUN Print\_string I // }

// Sub data

S\_ptr, HEX 0

//

// print number Subroutine

Print\_num, HEX 0 // print\_num(number) {

STA number //

LDA Four //

STA Count //

LDA Three\_hundred //

STA power10ptr //

LDA number //

SNA // IF (number <0)

BUN whileLoop //

CMA //

INC //

STA number // THEN number = *-* number

LDA Minus\_sign //

BSA Put\_char // output(‘-‘)

// FI

WhileLoop, LDA Count // WHILE (Count ≥ 0)

SPA //

BUN EndLoop //

CLA // DO

STA digit // digit=0;

Loop2, LDA power10ptr I // WHILE (number – 10^Count > 0)

ADD number //

SNA //

BUN do // DO

BUN Out\_last //

do, STA number // num= num – 10^Count;

ISZ digit // digit++;

// OD

BUN Loop2 // END

Out\_last, LDA digit //

ADD ascii\_Offset //

BSA Put\_char // Put\_char(digit);

ISZ power10ptr //

LDA count //

ADD Minus\_one // Count--;

STA Count // OD

BUN WhileLoop // END

EndLoop, LDA number //

ADD ascii\_Offset //

BSA Put\_char // Put\_char(last digit);

LDA CR //

BSA Put\_char //

BUN Print\_num I //}

//sub data

digit, DEC 0

Minus\_sign, HEX 2D // the "-" character

ascii\_Offset, HEX 30 //

number, DEC 0 //

count, DEC 4 //represent the exponent of 10

Four, DEC 4 //to initialize count

power10ptr, HEX 300 //

Three\_hundred, HEX 300 ////to initialize PTR

CR, HEX 0D // endl

//

ORG 300 //

Power10Arr, DEC -10000 // -10 with exponent 4

DEC -1000 // -10 with exponent 3

DEC -100 // -10 with exponent 2

DEC -10 // -10 with exponent 1

// Subroutine to print a char to screen

Put\_char, HEX 0 //AUX\_void put(char) {

Out, SKO

BUN Out

OUT // print(char);

BUN Put\_char I //

//addition Subroutine

addition\_ptr, HEX 330 //

ORG 330 //

Addition, HEX 0 // Addition(int num1,int num2){

BSA POP //

STA num\_1 //

AND Mask //

STA num1\_msb //

BSA POP //

STA num\_2 //

ADD num\_1 //

STA result1 // Result = num1 + num2

LDA num\_2 //

AND Mask //

CMA //

INC //

ADD num1\_msb //

SZA // IF (num1)msb == (num2)msb

BUN its\_ok //

LDA result1 //

AND Mask //

CMA //

INC //

ADD num1\_msb //

SZA // THEN IF (num1)msb != (Result)msb

BUN MathError //

its\_ok, LDA Result1 //

BUN Addition I //

MathError, LDA math\_error\_ptr // THEN Error over/under flow

BSA Print\_string // FI

BUN Start // FI

//sub data //}

num\_1, DEC 0 //

num\_2, DEC 0 //

Mask, DEC 32768 //Binary (1000000000000000) to get the msb

result1, DEC 0 //

num1\_msb, DEC 0 //

//

// subtraction Subroutine

subtraction\_ptr, HEX 400 // subtraction(int num1,int num2){

ORG 400 //

subtraction, HEX 0 //

BSA POP //

CMA //

ADD one // num2=-num2;

BSA PUSH //

BSA Addition // return addition(num1,num2);

BUN subtraction I //}

//

multiplication\_ptr, HEX 420 // Multiplication(int num1, int num2){

ORG 420 //

Multiplication, HEX 0 //

CLA //

STA Result //

STA DigitCount //

BSA POP //

STA number2 //

BSA POP //

STA number1 //

SZA // IF(num1 == 0)

BUN maybe\_second\_0 //

BUN Multiplication I // THEN return 0;

maybe\_second\_0, LDA number2 //

SZA // ELSE IF(num2 == 0)

BUN reg\_multy\_loop //

BUN Multiplication I // THEN return 0;

// ELSE Result = 0

reg\_multy\_loop, LDA DigitCount // FOR (16 binary digits in multiplier)

ADD Digits //

SZA //

BUN Body //

BUN End\_loop //

Body, LDA number2 // DO

CIR //

STA number2 //

SZE // IF (digit == 1)

BUN THEN //

BUN FI //

THEN, LDA Result // THEN result = result + multiplicand;

ADD number1 //

STA Result //

FI, LDA number1 // FI;

CIL // Shift(multiplicand) 1 place to left;

STA number1 //

ISZ DigitCount // DigitCount++;

BUN reg\_multy\_loop // OD

End\_loop , LDA Result // END

BUN Multiplication I // FI

// // FI

// sub data

number1, DEC 0 //

number2, DEC 0 //

DigitCount, DEC 0 //

Digits, DEC -16 //loop size for 16 bit number

Result, DEC 0 //

Division\_ptr, HEX 500

ORG       500

Division,               HEX        0 //(int,int) division (int num1 ,int num2){

CLA //

STA remainer // int remainer=0;

STA Counter // int counter=0;

STA flag2 // int flag=0;

                                BSA        POP //

                                STA        Temp\_num2 //

                                SZA // IF(num2 == 0)

                                BUN       Continue //

                                BUN       MathError // THEN printf("math error");

Continue,           BSA        POP //

                                STA        Temp\_num1 //

                                SNA // ELSE IF(num1<0 )

                                BUN       Div\_loop //

CMA //

INC //

STA Temp\_num1 // THEN num1= -num1;

ISZ flag2 // Flag=ON;

Div\_loop , LDA Temp\_num1 // FI

SPA // WHILE (num1>0)

BUN End\_Div\_loop //

LDA Temp\_num2 // DO

CMA //

INC //

ADD Temp\_num1 //

STA Temp\_num1 // num1=num1-num2;

SNA // IF (num1>=0)

ISZ Counter // THEN counter ++;

BUN Div\_loop // FI

End\_Div\_loop, LDA Temp\_num1 // OD

SZA // IF(num1==0)

BUN find\_remainer //

CLA //

STA Remainer // THEN remainer=0;

BUN flag\_check //

find\_remainer, ADD Temp\_num2 //

STA Remainer // ELSE remainer= num1+num2;

flag\_check, LDA Flag2 // FI FI

SPA // IF (flag==1)

BUN Finish //

LDA Counter //

CMA //

INC //

STA Counter // THEN Counter= -counter;

Finish, LDA RemainerStr\_ptr // FI

BSA Print\_string //

LDA Remainer //

BSA print\_num //

LDA result\_string\_ptr //

BSA Print\_string //

LDA Counter //

BUN       Division I // Return(counter,remainer);

//sub data //}

Counter, DEC 0 //

Remainer, DEC 0 //

Temp\_num1, DEC 0 //

Temp\_num2, DEC 0 //

flag2, DEC 0 //to indicate the sign of the number

//

good\_bye\_ptr, HEX 670 //

ORG 670 //

Good\_bye, HEX 0D //”Good bye”

HEX 47 //

HEX 6F //

HEX 6F //

HEX 64 //

HEX 20 //

HEX 62 //

HEX 79 //

HEX 65 //

HEX 0D //

HEX 00 //

//

invalid\_input\_ptr, HEX 685 //

ORG 685 //

InvalidInput, HEX 0D //”Invalid input”

HEX 49 //

HEX 6E //

HEX 76 //

HEX 61 //

HEX 6C //

HEX 69 //

HEX 64 //

HEX 20 //

HEX 69 //

HEX 6E //

HEX 70 //

HEX 75 //

HEX 74 //

HEX 0D //

HEX 00 //

//

math\_error\_ptr, HEX 710 //

ORG 710 //

math\_error, HEX 0D //”Math error”

HEX 4D //

HEX 61 //

HEX 74 //

HEX 68 //

HEX 20 //

HEX 65 //

HEX 72 //

HEX 72 //

HEX 6F //

HEX 72 //

HEX 0D //

HEX 00 //

RemainerStr\_ptr, HEX 900 //

ORG 900 //

RemainerStr, HEX 52 //”Remainer :”

HEX 65 //

HEX 6D //

HEX 61 //

HEX 69 //

HEX 6E //

HEX 65 //

HEX 72 //

HEX 3A //

HEX 20 //

HEX 00 //

//

result\_string\_ptr, HEX 920 //

ORG 920 //

result\_string, HEX 20 //”Result :”

HEX 20 //

HEX 20 //

HEX 20 //

HEX 52 //

HEX 65 //

HEX 73 //

HEX 75 //

HEX 6C //

HEX 74 //

HEX 3A //

HEX 20 //

HEX 00 //

END // hallelujah!