Roll No: 1803067

Lab Performance Test 3

Lab Task Q1

Question:

Q1. Write an Assembly Program for expression:

G = 60;
T = 70;
F = G + 40 - T;

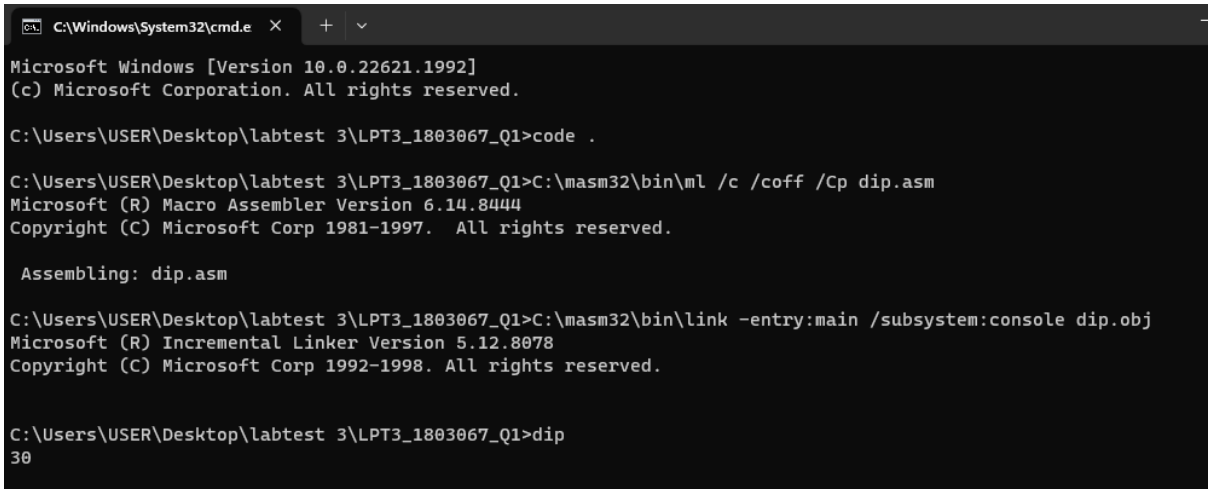
if(T > 0)
{
print(F)
};

Solution (Bold your own written code):

```
;###############   1803067 ######################

;start -1
.686
.model flat, c
include C:\masm32\include\msvcrt.inc
includelib C:\masm32\lib\msvcrt.lib

.stack 100h
printf PROTO arg1:Ptr Byte, printlist:VARARG
scanf PROTO arg2:Ptr Byte, inputlist:VARARG

.data
output_integer_msg_format byte "%d", 0Ah, 0
output_string_msg_format byte "%s", 0Ah, 0
input_integer_format byte "%d",0

G sdword ?
T sdword ?
F sdword ?
```

```
.code

main proc
    mov eax ,60
    mov G , eax

    mov eax ,70
    mov T , eax

    mov eax , G
    add eax , 40
    sub eax , T

    mov F , eax

    mov eax , T
    mov ebx , 0

    cmp eax , ebx
    JLE exit_if
    INVOKE printf, ADDR output_integer_msg_format, F
exit_if:

    ret
main endp
end
```

Output (Screen/SnapShot):



Lab Task Q2(a and b)

Question:

Q2. Consider following code snippets:

```
LET a as INT =  SCAN();
LET c as INT = SCAN() - a + 2;
PRINT(c);
PRINT(10);
```

(a) Generate Intermediate Code Generation from the given code snippet.
(b) Generate Code Generation from the given code snippet.

Solution (Bold your own written code):

lexer.l

```
%option noyywrap

%{
    #define INT_TYPE 1
    #include <stdio.h>
    #include <stdlib.h>
    #include <string.h>
    #include "parser.tab.h"

    int lineno = 1; // initialize to 1
    void yyerror();
```

```
%}

alpha     [a-zA-Z]
digit     [0-9]
alnum     {alpha}|{digit}
print     [ -~]

ID        {alpha}{alnum}*
ICONST    [0-9]{digit}*
%%

"//".*        { }

"INT"       {yylval.int_val=INT_TYPE; return INT; }
"LET"      {return LET;}
"as"       {return AS;}
"if"       {return IF;}
"else"     {return ELSE;}
"while"    { return WHILE; }

"+"       { return ADDOP; }
"-"       { return SUBOP; }
"*"       { return MULOP; }
"/"       { return DIVOP; }
"=="      { return EQUOP; }
">"       { return GT; }
"<"       { return LT; }

"("       { return LPAREN; }
")"       { return RPAREN; }
"{"       { return LBRACE; }
"}"       { return RBRACE; }
";"       { return SEMI; }
"="       { return ASSIGN; }
```

```
"PRINT"       { return PRINT; }
"SCAN"       { return SCAN; }

{ID}        {strcpy(yylval.str_val, yytext); return ID;}
{ICONST}    {yylval.int_val=atoi(yytext); return ICONST;}

"\n"        { lineno += 1; }
[ \t\r\f]+

.       { yyerror("Unrecognized character"); }
```

parser.y

```
%{
    #include <stdio.h>
    #include <stdlib.h>
    #include <string.h>
    #include "symtab.c"
    #include "codeGen.c"
    void yyerror();
    extern int lineno;
    extern int yylex();

%}

%union
{
    char str_val[100];
    int int_val;
}

%token PRINT SCAN LET AS
%token ADDOP SUBOP MULOP DIVOP EQUOP LT GT
```

```
%token LPAREN RPAREN LBRACE RBRACE SEMI ASSIGN ELSE
%token<str_val> ID
%token<int_val> ICONST
%token<int_val> INT
%token<int_val> IF
%token<int_val> WHILE

%left LT GT /*LT GT has lowest precedence*/
%left ADDOP
%left MULOP /*MULOP has lowest precedence*/

//%type<int_val> T

%start program

%%
program: {gen_code(START, -1); insert("buffer",INT_TYPE); } code
{gen_code(HALT, -1);}
code: code st |;
st : dec SEMI | print SEMI ;
dec : LET ID AS INT ASSIGN exp
    { insert( $2,$4) ;

    int address=idcheck($2) ;
    if(address!=-1)
    {
        gen_code(STORE,address);
    }
    else{
        yyerror ();
    }

     };
exp : scan {
```

```
    int address=idcheck("buffer") ;
    if(address!=-1)
    {
        gen_code(SCAN_INT_VALUE,address);
        gen_code(LD_VAR,address);
    }
    else{
        yyerror ();
    }

}
    | exp ADDOP exp { gen_code(ADD,-1);}
    | exp SUBOP exp { gen_code(SUB,-1);}
    | T;
print : PRINT LPAREN ID RPAREN
{
    int address=idcheck($3) ;
    if(address!=-1)
    {
        gen_code(PRINT_INT_VALUE,address);
    }
    else{
        yyerror ();
    }

}|PRINT LPAREN ICONST RPAREN
{
    gen_code(LD_INT ,$3);
    int address=idcheck("buffer") ;
    if(address!=-1)
    {
        gen_code(STORE,address);
```

```
        gen_code(PRINT_INT_VALUE,address);

    }
    else{
        yyerror ();
    }

}

;
scan : SCAN LPAREN RPAREN ;
T : ID {
    int address=idcheck($1) ;
    if(address!=-1)
    {
        gen_code(LD_VAR,address);
    }
    else{
        yyerror ();
    }

}| ICONST {
    gen_code( LD_INT ,$1);
};
%%

void yyerror ()
{
    printf("Syntax error at line %d\n", lineno);
    exit(1);
}

int main (int argc, char *argv[])
{
```

```
    yyparse();
    printf("Parsing finished!\n");

    printf("============= INTERMEDIATE CODE===============\n");
    print_code();

    printf("============= ASM CODE===============\n");
    print_assembly();

    return 0;
}
```

Codegen.c

```
#include "codeGen.h"

int gen_label()
{
    return code_offset;
}

void gen_code(enum code_ops op, int arg)
{
    code[code_offset].op = op;
    code[code_offset].arg = arg;

    code_offset++;
}

void print_code()
{
    int i = 0;
```

```
    for(i=0; i<code_offset; i++)
    {
        printf("%3d: %-15s  %4d\n", i, op_name[code[i].op], code[i].arg);
    }
}

void print_assembly()
{
    int i = 0;
    int j = 0;

    int stack_variable_counter = 0;

    for(i=0; i<code_offset; i++)
    {
        printf("\n;%s %d\n", op_name[code[i].op], code[i].arg);

        if(code[i].op == LD_INT || code[i].op == LD_VAR)
            stack_variable_counter++;

        if(code[i].op == ADD)
            stack_variable_counter--;

        switch(code[i].op)
        {
            case START:
                            printf(".686\n");
                            printf(".model flat, c\n");
                            printf("include
C:\\masm32\\include\\msvcrt.inc\n");
                            printf("includelib
C:\\masm32\\lib\\msvcrt.lib\n");
                            printf("\n");
                            printf(".stack 100h\n");
```

```
                        printf("printf PROTO arg1:Ptr Byte,
printlist:VARARG\n");
                        printf("scanf PROTO arg2:Ptr Byte,
inputlist:VARARG\n");
                        printf("\n");
                        printf(".data\n");
                        printf("output_integer_msg_format byte \"\%
%d\", 0Ah, 0\n");
                        printf("output_string_msg_format byte \"\%
%s\", 0Ah, 0\n");
                        printf("input_integer_format byte \"\%
%d\",0\n");
                        printf("\n");
                        printf("number sdword ?\n");
                        printf("\n");
                        printf(".code\n");
                        printf("\n");
                        printf("main proc\n");
                        printf("\tpush ebp\n");
                        printf("\tmov ebp, esp\n");
                        printf("\tsub ebp, 100\n");
                        printf("\tmov ebx, ebp\n");
                        printf("\tadd ebx, 4\n");
                        break;

            case HALT:
                        printf("\tadd ebp, 100\n");
                        printf("\tmov esp, ebp\n");
                        printf("\tpop ebp\n");
                        printf("\tret\n");
                        printf("main endp\n");
                        printf("end\n");
                        break;
```

```
            case STORE:
                            printf("\tmov eax, [ebx-4]\n");
                            printf("\tmov dword ptr [ebp-%d], eax\n",
4*code[i].arg);
                            break;

            case SCAN_INT_VALUE:
                            printf("\tpush eax\n");
                            printf("\tpush ebx\n");
                            printf("\tpush ecx\n");
                            printf("\tpush edx\n");
                            for(j=address-1; j>=0; j--)
                                printf("\tpush [ebp-%d]\n", 4*j);
                            for(j=1; j<=stack_variable_counter; j++)
                                printf("\tpush [ebp+%d]\n", 4*j);
                            printf("\tpush ebp\n");

                            printf("\tINVOKE scanf, ADDR
input_integer_format, ADDR number\n");

                            printf("\tpop ebp\n");
                            for(j=stack_variable_counter; j>=1; j--)
                                printf("\tpop [ebp+%d]\n", 4*j);
                            for(j=0; j<=address-1; j++)
                                printf("\tpop [ebp-%d]\n", 4*j);

                            printf("\tmov eax, number\n");
                            printf("\tmov dword ptr [ebp-%d], eax\n",
4*code[i].arg);

                            printf("\tpop edx\n");
                            printf("\tpop ecx\n");
                            printf("\tpop ebx\n");
                            printf("\tpop eax\n");
```

```
                        break;


            case PRINT_INT_VALUE:
                        printf("\tpush eax\n");
                        printf("\tpush ebx\n");
                        printf("\tpush ecx\n");
                        printf("\tpush edx\n");
                        for(j=address-1; j>=0; j--)
                            printf("\tpush [ebp-%d]\n", 4*j);
                        for(j=1; j<=stack_variable_counter; j++)
                            printf("\tpush [ebp+%d]\n", 4*j);
                        printf("\tpush ebp\n");

                        printf("\tmov eax, [ebp-%d]\n", 4*code[i].arg);
                        printf("\tINVOKE printf, ADDR
output_integer_msg_format, eax\n");

                        printf("\tpop ebp\n");
                        for(j=stack_variable_counter; j>=1; j--)
                            printf("\tpop [ebp+%d]\n", 4*j);
                        for(j=0; j<=address-1; j++)
                            printf("\tpop [ebp-%d]\n", 4*j);
                        printf("\tpop edx\n");
                        printf("\tpop ecx\n");
                        printf("\tpop ebx\n");
                        printf("\tpop eax\n");
                        break;

            case LD_VAR:
                        printf("\tmov eax, [ebp-%d]\n", 4*code[i].arg);
                        printf("\tmov dword ptr [ebx], eax\n");
                        printf("\tadd ebx, 4\n");
                        printf("\n");
```

```
                    break;

            case LD_INT:
                    printf("\tmov eax, %d\n", code[i].arg);
                    printf("\tmov dword ptr [ebx], eax\n");
                    printf("\tadd ebx, 4\n");
                    printf("\n");
                    break;

            case ADD:
                    printf("\tsub ebx, 4\n");
                    printf("\tmov eax, [ebx]\n");
                    printf("\tsub ebx, 4\n");
                    printf("\tmov edx, [ebx]\n");
                    printf("\tadd eax, edx\n");
                    printf("\tmov dword ptr [ebx], eax\n");
                    printf("\tadd ebx, 4\n");
                    printf("\n");
                    break;
            case DIV:
                    printf("\tsub ebx, 4\n");
                    printf("\tmov eax, [ebx]\n");
                    printf("\tsub ebx, 4\n");
                    printf("\tmov edx, [ebx]\n");
                    printf("\tpush ebx\n");

                    printf("\tmov ebx, eax\n");
                    printf("\tmov eax, edx\n");
                    printf("\tmov edx, 0\n");
                    printf("\tidiv ebx\n");
                    printf("\tpop ebx\n");
                    printf("\tmov dword ptr [ebx], eax\n");
                    printf("\tadd ebx, 4\n");
                    printf("\n");
```

```
                    break;
            case SUB:
                    printf("\tsub ebx, 4\n");
                    printf("\tmov eax, [ebx]\n");
                    printf("\tsub ebx, 4\n");
                    printf("\tmov edx, [ebx]\n");
                    printf("\tsub edx, eax\n");
                    printf("\tmov eax, edx\n");
                    printf("\tmov dword ptr [ebx], eax\n");
                    printf("\tadd ebx, 4\n");
                    printf("\n");
                    break;
            case MUL:
                    printf("\tsub ebx, 4\n");
                    printf("\tmov eax, [ebx]\n");
                    printf("\tsub ebx, 4\n");
                    printf("\tmov edx, [ebx]\n");
                    printf("\tIMUL eax, edx\n");
                    printf("\tmov dword ptr [ebx], eax\n");
                    printf("\tadd ebx, 4\n");
                    printf("\n");
                    break;
            case GT_OP:
                    printf("\tsub ebx, 4\n");
                    printf("\tmov eax, [ebx]\n");
                    printf("\tsub ebx, 4\n");
                    printf("\tmov edx, [ebx]\n");
                    printf("\tcmp edx, eax\n");

                    {
                        char relop_start_label[50]="LS";
                        char relop_end_label[50]="LE";
                        char number[10];
                        itoa(code[i].arg, number, 10);
```

```
                                strcat(relop_end_label, number);
                                strcat(relop_start_label, number);

                                printf("\tjg %s\n", relop_start_label);
                                printf("\tmov dword ptr [ebx], 0\n");
                                printf("\tjmp %s\n", relop_end_label);
                                printf("\t%s: mov dword ptr [ebx], 1\n",
relop_start_label);
                                printf("\t%s: add ebx, 4\n\n",
relop_end_label);
                            }
                            printf("\n");
                            break;
            case LT_OP:
                            printf("\tsub ebx, 4\n");
                            printf("\tmov eax, [ebx]\n");
                            printf("\tsub ebx, 4\n");
                            printf("\tmov edx, [ebx]\n");
                            printf("\tcmp edx, eax\n");

                            {
                                char relop_start_label[50]="LS";
                                char relop_end_label[50]="LE";
                                char number[10];
                                itoa(code[i].arg, number, 10);
                                strcat(relop_end_label, number);
                                strcat(relop_start_label, number);

                                printf("\tjl %s\n", relop_start_label);
                                printf("\tmov dword ptr [ebx], 0\n");
                                printf("\tjmp %s\n", relop_end_label);
                                printf("\t%s: mov dword ptr [ebx], 1\n",
relop_start_label);
```

```
                                printf("\t%s: add ebx, 4\n\n",
relop_end_label);
                            }
                            printf("\n");
                            break;
            case IF_START:
                            printf("\tmov eax, [ebx-4]\n");
                            printf("\tcmp eax, 0\n");
                            {
                                char
else_start_label[]="ELSE_START_LABEL_";
                                char number[10];
                                strcat(else_start_label,itoa(code[i].arg,
number, 10));
                                printf("\tjle %s\n", else_start_label);
                            }
                            printf("\n");
                            break;
            case ELSE_START:
                            {
                                char
else_start_label[50]="ELSE_START_LABEL_";
                                char else_end_label[50]="ELSE_END_LABEL_";
                                char number[10];
                                itoa(code[i].arg, number, 10);
                                strcat(else_end_label, number);
                                printf("\tjmp %s\n", else_end_label);
                                strcat(else_start_label, number);
                                printf("%s:\n", else_start_label);
                            }
                            printf("\n");
                            break;
            case ELSE_END:
                            {
```

```
                                char else_end_label[50]="ELSE_END_LABEL_";
                                char number[10];
                                itoa(code[i].arg, number, 10);
                                strcat(else_end_label, number);
                                printf("%s:\n", else_end_label);
                            }
                            printf("\n");
                            break;
            case WHILE_LABEL:
                            {
                                char
while_start_label[]="WHILE_START_LABEL_";
                                char number[10];
                                strcat(while_start_label,itoa(code[i].arg,
number, 10));
                                printf("%s:\n", while_start_label);
                            }
                            printf("\n");
                            break;
            case WHILE_START:
                            printf("\tmov eax, [ebx-4]\n");
                            printf("\tcmp eax, 0\n");
                            {
                                char while_end_label[]="WHILE_END_LABEL_";
                                char number[10];
                                strcat(while_end_label,itoa(code[i].arg,
number, 10));
                                printf("\tjle %s\n", while_end_label);
                            }
                            printf("\n");
                            break;
            case WHILE_END:
                            {
```

```
                                char
while_start_label[50]="WHILE_START_LABEL_";
                                char
while_end_label[50]="WHILE_END_LABEL_";
                                char number[10];
                                itoa(code[i].arg, number, 10);
                                strcat(while_start_label, number);
                                printf("\tjmp %s\n", while_start_label);
                                strcat(while_end_label, number);
                                printf("%s:\n", while_end_label);
                            }
                            printf("\n");
                            break;
                default:
                            break;
            }


        }
}
```

Output (Screen/SnapShot):

```
In line no 1, Inserting buffer with type INT_TYPE in symbol table.
In line no 1, Inserting a with type INT_TYPE in symbol table.
In line no 2, Inserting c with type INT_TYPE in symbol table.
Parsing finished!
============= INTERMEDIATE CODE===============
  0: start           -1
  1: scan_int_value     0
  2: ld_var           0
  3: store            1
  4: scan_int_value     0
  5: ld_var           0
  6: ld_var           1
  7: ld_int           2
  8: add             -1
  9: sub             -1
 10: store            2
 11: print_int_value    2
```

```
 12: ld_int            10
 13: store             0
 14: print_int_value   0
 15: halt              -1
============= ASM CODE==============

;start -1
.686
.model flat, c
include C:\masm32\include\msvcrt.inc
includelib C:\masm32\lib\msvcrt.lib

.stack 100h
printf PROTO arg1:Ptr Byte, printlist:VARARG
scanf PROTO arg2:Ptr Byte, inputlist:VARARG

.data
output_integer_msg_format byte "%d", 0Ah, 0
output_string_msg_format byte "%s", 0Ah, 0
input_integer_format byte "%d",0

number sdword ?

.code

main proc
        push ebp
        mov ebp, esp
        sub ebp, 100
        mov ebx, ebp
        add ebx, 4

;scan_int_value 0
        push eax
        push ebx
        push ecx
        push edx
        push [ebp-8]
        push [ebp-4]
        push [ebp-0]
        push ebp
        INVOKE scanf, ADDR input_integer_format, ADDR number
        pop ebp
        pop [ebp-0]
        pop [ebp-4]
        pop [ebp-8]
        mov eax, number
        mov dword ptr [ebp-0], eax
        pop edx
        pop ecx
```

```
        pop ebx
        pop eax

;ld_var 0
        mov eax, [ebp-0]
        mov dword ptr [ebx], eax
        add ebx, 4


;store 1
        mov eax, [ebx-4]
        mov dword ptr [ebp-4], eax

;scan_int_value 0
        push eax
        push ebx
        push ecx
        push edx
        push [ebp-8]
        push [ebp-4]
        push [ebp-0]
        push [ebp+4]
        push ebp
        INVOKE scanf, ADDR input_integer_format, ADDR number
        pop ebp
        pop [ebp+4]
        pop [ebp-0]
        pop [ebp-4]
        pop [ebp-8]
        mov eax, number
        mov dword ptr [ebp-0], eax
        pop edx
        pop ecx
        pop ebx
        pop eax

;ld_var 0
        mov eax, [ebp-0]
        mov dword ptr [ebx], eax
        add ebx, 4


;ld_var 1
        mov eax, [ebp-4]
        mov dword ptr [ebx], eax
        add ebx, 4


;ld_int 2
        mov eax, 2
```

```
        mov dword ptr [ebx], eax
        add ebx, 4


;add -1
        sub ebx, 4
        mov eax, [ebx]
        sub ebx, 4
        mov edx, [ebx]
        add eax, edx
        mov dword ptr [ebx], eax
        add ebx, 4


;sub -1
        sub ebx, 4
        mov eax, [ebx]
        sub ebx, 4
        mov edx, [ebx]
        sub edx, eax
        mov eax, edx
        mov dword ptr [ebx], eax
        add ebx, 4


;store 2
        mov eax, [ebx-4]
        mov dword ptr [ebp-8], eax

;print_int_value 2
        push eax
        push ebx
        push ecx
        push edx
        push [ebp-8]
        push [ebp-4]
        push [ebp-0]
        push [ebp+4]
        push [ebp+8]
        push [ebp+12]
        push ebp
        mov eax, [ebp-8]
        INVOKE printf, ADDR output_integer_msg_format, eax
        pop ebp
        pop [ebp+12]
        pop [ebp+8]
        pop [ebp+4]
        pop [ebp-0]
        pop [ebp-4]
        pop [ebp-8]
```

```
        pop edx
        pop ecx
        pop ebx
        pop eax

;ld_int 10
        mov eax, 10
        mov dword ptr [ebx], eax
        add ebx, 4


;store 0
        mov eax, [ebx-4]
        mov dword ptr [ebp-0], eax

;print_int_value 0
        push eax
        push ebx
        push ecx
        push edx
        push [ebp-8]
        push [ebp-4]
        push [ebp-0]
        push [ebp+4]
        push [ebp+8]
        push [ebp+12]
        push [ebp+16]
        push ebp
        mov eax, [ebp-0]
        INVOKE printf, ADDR output_integer_msg_format, eax
        pop ebp
        pop [ebp+16]
        pop [ebp+12]
        pop [ebp+8]
        pop [ebp+4]
        pop [ebp-0]
        pop [ebp-4]
        pop [ebp-8]
        pop edx
        pop ecx
        pop ebx
        pop eax

;halt -1
        add ebp, 100
        mov esp, ebp
        pop ebp
        ret
main endp
end
```

Assembly code output:

```
C:\Users\USER\Desktop\labtest 3\LPT3_1803067_Q2>C:\masm32\bin\ml /c /coff /Cp dip.asm
Microsoft (R) Macro Assembler Version 6.14.8444
Copyright (C) Microsoft Corp 1981-1997.  All rights reserved.

 Assembling: dip.asm

C:\Users\USER\Desktop\labtest 3\LPT3_1803067_Q2>C:\masm32\bin\link -entry:main /subsystem:console dip.obj
Microsoft (R) Incremental Linker Version 5.12.8078
Copyright (C) Microsoft Corp 1992-1998. All rights reserved.


C:\Users\USER\Desktop\labtest 3\LPT3_1803067_Q2>dip
50
100
48
10
```