Roll No: 1903001

Lab Performance Evaluation [02]

Lab Task Q1

Question:

Q1. Write an Assembly Program for expression: *

H = 60;

U = 70;

F = H + 40 -U;

if(U > 0 && F > 0) { print(F); }

else { print(H); }

Solution (Bold your own written code):

```
;start -1
.686
.model flat, c
include E:\masm32\include\msvcrt.inc
includelib E:\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
inp_msg_format byte "%s", 0
output_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 dword ptr [ebp-0], 60 ;H
mov dword ptr [ebp-4], 70;U
mov eax, [ebp-0]
mov ebx, [ebp-4]
add eax, 40
sub eax, ebx
mov dword ptr [ebp-8], eax ; F

mov eax, [ebp-4]
cmp eax, 0
jng ELSE_
mov eax, [ebp-8]
cmp eax, 0
jng ELSE_
push [ebp-8]
push [ebp-4]
push [ebp-0]
push ebp
INVOKE printf, ADDR output_integer_msg_format, eax
pop ebp
pop [ebp-0]
pop [ebp-4]
pop [ebp-8]
jmp EXIT_

ELSE_:
push [ebp-8]
push [ebp-4]
push [ebp-0]
push ebp
mov eax, [ebp]
INVOKE printf, ADDR output_integer_msg_format, eax
pop ebp
pop [ebp-0]
pop [ebp-4]
pop [ebp-8]


EXIT_:
    ret
main endp
end
```

Output (Screen/SnapShot):

```
E:\My_Programs\CSE4102\matha_nosto\Given_LPE\LPE2\LPE2_1903001_Q1>prog
30
```

Lab Task Q2

Question:

Q2. Consider following code snippets: *

DEF X IS INT = IN();

DEF Y as INT = IN() - X + 2;

OUT(X+Y);

(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();
%}

letter [a-zA-Z]
digit [0-9]
ID ({letter})({letter}|{digit})*
ICONST {digit}+

%%
"IS" {return(IS);}
"IN" {return(SCAN);}
"INT" {return(INT);}
"OUT" {return(PRINT);}
"DEF" {return(DEF);}
"as" {return(AS);}
```

```
";" {return(SEMI);}
{ID} {strcpy(yylval.str_val, yytext); return(ID);}
"-" {return(MINUS);}
"+" {return(PLUS);}
{ICONST} {yylval.int_val=atoi(yytext); return(ICONST);}
")" {return(RP);}
"(" {return(LP);}
"=" {return(ASSIGN);}

"\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 SCAN PRINT MINUS PLUS RP LP ASSIGN IS DEF AS SEMI
%token<str_val> ID
%token<int_val> ICONST INT

%left PLUS MINUS

%type<int_val> type

%start code

%%
code: {gen_code(START, -1);}  statements  {gen_code(HALT, -1);};
statements: statements statement | ;
```

```
statement:  printf
            | assignment;

printf: PRINT LP pexp RP SEMI
    {
        char* name = "__TEMP__";
        int addr = idcheck(name);
        if(addr==-1) {
            insert(name, INT_TYPE);
            addr = idcheck(name);
        }
        gen_code(PRINT_INT_VALUE, addr);
    };
assignment: DEF ID IS type ASSIGN exp SEMI
        {
            int addr = idcheck($2);
            if(addr==-1) {
                insert($2, $4);
                addr = idcheck($2);
            }
            gen_code(STORE, addr);

        }
        | DEF ID AS type ASSIGN exp SEMI
        {
            {
            int addr = idcheck($2);
            if(addr==-1) {
                insert($2, $4);
                addr = idcheck($2);
            }
            gen_code(STORE, addr);

        }
        }
        ;

pexp: pexp PLUS T
    {
        gen_code(ADD, -1);
        char* name = "__TEMP__";
        int addr = idcheck(name);
        if(addr==-1) {insert(name, INT_TYPE);
        addr = idcheck(name);}
        gen_code(STORE, addr);
    }
    | T;
exp: exp PLUS T
```

```
    {
        gen_code(ADD, -1);
    }
    | exp MINUS T
    {
        gen_code(SUB, -1);
    }
    | T
    ;

T: ID
    {
        int addr = idcheck($1);
        if(addr!=-1) gen_code(LD_VAR, addr);
        else exit(0);
    }
   | ICONST
   {
        gen_code(LD_INT, $1);
   }
   | scanf
   ;
scanf: SCAN LP RP
    {
        char* name = "__TEMP__";
        int addr = idcheck(name);
        if(addr==-1) {
            insert(name, INT_TYPE);
            addr = idcheck(name);
        }

        gen_code(SCAN_INT_VALUE, addr);
        gen_code(LD_VAR, addr);
    };

type: INT{$$=INT_TYPE;};
%%

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;
```

Output (Screen/SnapShot):

Output.txt

In line no 1, ID __TEMP__ is not declared.
In line no 1, Inserting __TEMP__ with type INT_TYPE in symbol table.
In line no 1, ID X is not declared.
In line no 1, Inserting X with type INT_TYPE in symbol table.
In line no 2, ID Y is not declared.
In line no 2, Inserting Y 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: sub            -1
 8: ld_int          2
 9: add            -1
10: store           2
11: ld_var          1
12: ld_var          2
13: add            -1
14: store           0
15: print_int_value    0
16: halt           -1
```

============= ASM CODE===============

```
;start -1
.686
```

```
.model flat, c
include E:\masm32\include\msvcrt.inc
includelib E:\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 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


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


;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
```

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

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


;ld_var 2
        mov eax, [ebp-8]
        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


;store 0
        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
```

ScreenShot:

```
E:\My_Programs\CSE4102\matha_nosto\Given_LPE\LPE2\LPE2_1903001_Q2a>prog
12
2
4
```