**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)**

**{**

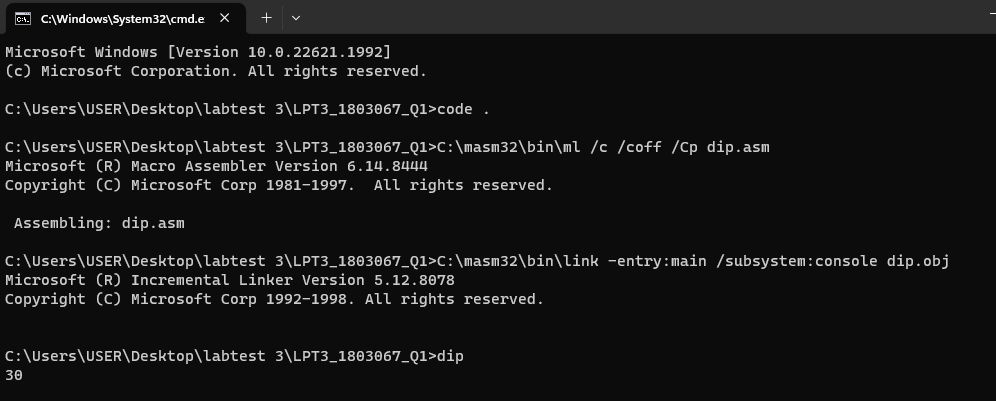
**print(F)**

**};**

**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;  } |

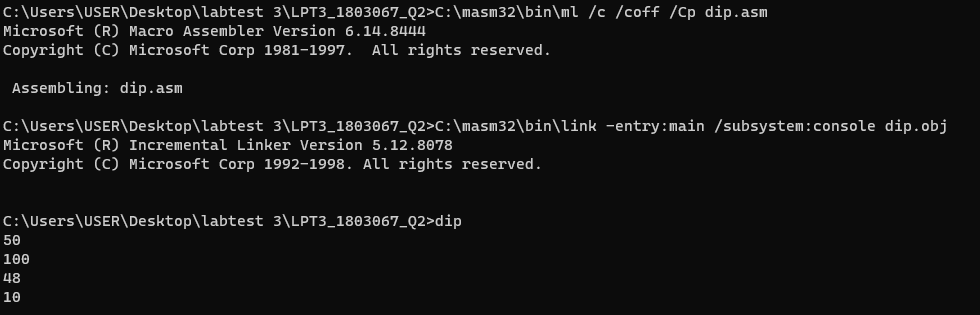
**Codegen.c**

|  |
| --- |
| #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:**

****