**Roll No: 1803122**

**Lab Performance Test [3]**

**Lab Task Q1**

**Question:** Generate an Assembly Program for expression:

**A = 30;**

**while(A > 20)**

**{**

**A = A-1;**

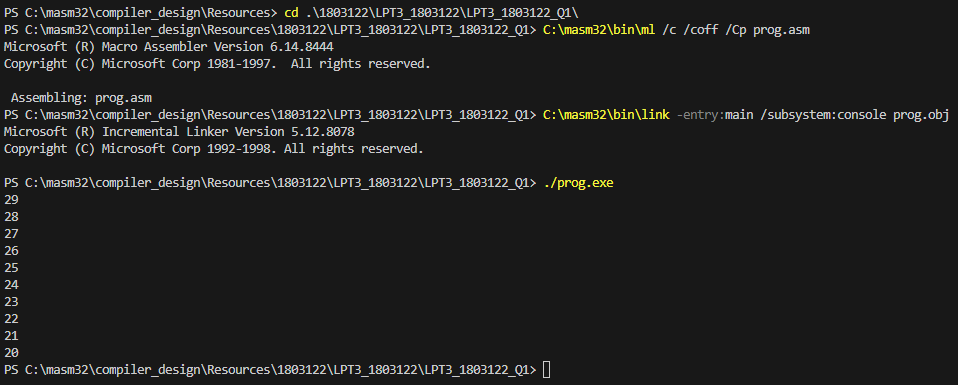
**print(A);**

**}**

**Solution (Bold your own written code):**

|  |
| --- |
| .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  input\_integer\_format byte "%d",0  output\_msg\_format byte "%s",0  output\_integer\_msg\_format byte "%d", 0Ah, 0  ;variable  A sdword 30  .code  main proc  while\_loop:      mov eax, A      cmp eax, 20      jle EXIT      sub eax, 1      mov A, eax      jmp PRINT  PRINT:      push eax      push ebx      push ecx      push edx      push ebp      INVOKE printf, ADDR output\_integer\_msg\_format, eax      pop ebp      pop edx      pop ecx      pop ebx      pop eax      jmp while\_loop    EXIT:    main endp  end |

**Output (Screen/SnapShot):**

****

**Lab Task Q2**

**Question:**

Generate Intermediate Code Generation from the given code snippet.

Generate Code Generation from the given code snippet.

**LET A as INT;**

**A = IN();**

**LET B as INT;**

**B = A + 7;**

**OUT(B);**

**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();  %}  digit     [0-9]  delim [ \t]  ws  {delim}+  ID        [AB]  ICONST    {digit}+  %%  {ws} {}  "LET"       { return LET;}  "AS"       { return AS; }  "INT"       {yylval.int\_val=INT\_TYPE; return INT; }  "("       { return LPAREN; }  ")"       { return RPAREN; }  ";"       { return SEMI; }  "="       { return ASSIGN; }  "+"       { return ADDOP; }  "OUT"       { return PRINT; }  "IN"       { 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"); } |

**Solution (Bold your own written code): 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  %token LPAREN RPAREN SEMI ASSIGN  %token<str\_val> ID  %token<int\_val> ICONST  %token<int\_val> INT  %left ADDOP  %type<int\_val> exp assignment\_print\_scan  %start program  %%  program: {gen\_code(START, -1);} code {gen\_code(HALT, -1);}  code: statements;  statements: statements statement | ;  statement:  declaration              |assignment\_print\_scan              ;  declaration: LET ID AS INT SEMI              {                  insert($2, $4);              }              ;  assignment\_print\_scan: ID ASSIGN exp SEMI                  {                      int address = idcheck($1);                      if(address != -1)                      {                          gen\_code(STORE, address);                      }                      else                          yyerror();                  }                  | PRINT LPAREN ID RPAREN SEMI                  {                      int address = idcheck($3);                      if(address != -1)                      {                          gen\_code(PRINT\_INT\_VALUE, address);                      }                      else                          yyerror();                  }                  | ID ASSIGN SCAN LPAREN RPAREN SEMI                  {                      int address = idcheck($1);                      if(address != -1)                      {                          gen\_code(SCAN\_INT\_VALUE, address);                      }                      else                          yyerror();                  }                  ;  exp: ICONST      {          gen\_code(LD\_INT, $1);      }      | ID        {              int address = idcheck($1);              if(address != -1)              {                  gen\_code(LD\_VAR, address);              }              else                  yyerror();        }      | exp ADDOP exp { gen\_code(ADD, -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;  } |

**Solution (Bold your own written code): symtab.c**

|  |
| --- |
| #include "symtab.h"  #include <stdio.h>  list\_t \*head = NULL;  int address = 0;  extern int lineno;  void insert(char\* name, int type)  {      if(search(name)==NULL)      {          list\_t\* temp = (list\_t\*)malloc(sizeof(list\_t));          strcpy(temp->st\_name, name);          temp->st\_type = type;          temp->address = address;          address++;          printf("In line no %d, Inserting %s with type %s in symbol table.\n", lineno, name, typename[type]);          temp->next = head;          head = temp;      }      else      {          printf("In line no %d, Same variable %s is declared more than once.\n", lineno, name);      }  }  list\_t\* search(char \*name)  {      list\_t\* current = head;      while(current!=NULL)      {         if(strcmp(name, current->st\_name)!=0)          current = current->next;         else          break;      }      return current;  }  int idcheck(char\* name)  {      list\_t \*id = search(name);      if(id==NULL)      {          printf("In line no %d, ID %s is not declared.\n", lineno, name);          return -1;      }      else      {          return id->address;      }  }  int gettype(char \*name)  {      list\_t\* temp = search(name);      if(temp==NULL)      {          printf("In line no %d, ID %s is not declared.\n", lineno, name);          return UNDEF\_TYPE;      }      else          return temp->st\_type;  }  int typecheck(int type1, int type2)  {      if(type1==INT\_TYPE && type2==INT\_TYPE)          return INT\_TYPE;      else if (type1==REAL\_TYPE && type2==REAL\_TYPE)          return REAL\_TYPE;      else if (type1==CHAR\_TYPE && type2==CHAR\_TYPE)          return CHAR\_TYPE;      else      {          if (type1 > 3 || type1 < 0) type1 = 0;          if (type2 > 3 || type2 < 0) type2 = 0;          printf("In line no %d, Data type %s is not matched with Data type %s.\n", lineno, typename[type1], typename[type2]);          return UNDEF\_TYPE;      }  } |

**Solution (Bold your own written code): symtab.h**

|  |
| --- |
| #define UNDEF\_TYPE 0  #define INT\_TYPE 1  #define REAL\_TYPE 2  #define CHAR\_TYPE 3  char \*typename[] = {"UNDEF\_TYPE", "INT\_TYPE", "REAL\_TYPE", "CHAR\_TYPE"};  typedef struct list\_t  {      char st\_name[40];      int st\_type;      int address;      struct list\_t \*next;  }list\_t;  void insert(char\* name, int type);  list\_t\* search(char \*name);  int idcheck(char\* name);  int gettype(char \*name);  int typecheck(int type1, int type2); |

**Solution (Bold your own written code): codeGen.c**

|  |
| --- |
| #include "codeGen.h"  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 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;              default:                              break;          }      }  } |

**Solution (Bold your own written code): codeGen.h**

|  |
| --- |
| enum code\_ops  {      START,      HALT,      LD\_INT,      LD\_VAR,      STORE,      SCAN\_INT\_VALUE,      PRINT\_INT\_VALUE,      ADD  };  char \*op\_name[] = {"start", "halt", "ld\_int", "ld\_var", "store", "scan\_int\_value", "print\_int\_value", "add"};  struct instruction  {      enum code\_ops op;      int arg;  };  struct instruction code[999];  extern int address;  int code\_offset = 0; |

**Output (Screen/SnapShot):**

