Lab 9 Generating MIPS CODE
Yuhao Lan

Levels:
a) Can only read and write variables and numbers 40% off
b) A program which calls no functions and does no array arithmetic at least 30% off
c) Can do a-b and "if" and "while" 20% off
d) Can do a-c and also arrays 10%
e) Can do a-d and handle function calls
f) Add strings to "write", Extra Credit 5%... You MUST do a-e do get the extra credit

I have finished a) b) c) and part of e). So I think I will get 85% of the whole scores.

TEST ONE:

```
void main (int b)
{
    read b;
    if (b > 10)
      write 1;
    else
      write 0;
}
```

the output:

```
2
3  main:
4  subu $sp,$sp,4 #substract the stack pointer for main to get space for global variables
5  addi $gp,$sp,0 #copy the space to global pointer for global variables
6  subu $sp, $sp,16 #substract the stack pointer for main
7  subu $t2,$sp,16 #get the space of function
8  sw $ra,($t2) #Store the return address
9  sw $sp,4($t2) #Store the old stack pointer
10 addu $sp,$t2,0 #copy the ~~~~~
11
12 li $v0,5 #read a value
13 syscall #print the value
14 sw $v0,8($sp) # read store the local value on the stack
15 lw $t2,8($sp) #expr left hand is local
16 sw $t2,12($sp) #expr left hand store in sp
17 li $t3,10
18 lw $t2,12($sp) #lw the final local left hand
19 sgt $t2,$t2,$t3
20 sw $t2,12($sp)
21 lw $t0,12($sp) #expr is expr
22 beq $t0,0,LABEL1 #if the expression is false go to label1
23 li $a0,1
24 li $v0,1
25 syscall
26 j LABEL2 #go to label2
27
28 LABEL1 : #label1 is here
29
30 li $a0,0
31 li $v0,1
32 syscall
33
34 LABEL2 : #label2 is here
35
36 li $v0,0 # return zero
37 lw $ra ($sp) # get the original return address
38 lw $sp 4($sp) # get the old stack pointer
39 jr $ra #jump to the next instruction
```

MIPS OUTPUT:

```
King:spim yuhaolan$ vi test.asm
King:spim yuhaolan$ ./spim
Loaded: ./exceptions.s
(spim) load "test.asm"
(spim) run
11
1(spim) run
8
0(spim)
```

TEST TWO:

```
int x;

int main(int b)
{  int y;
   int sum;
   sum=0;
   x=1;
   read y;
   while (x <= y)
     {
     sum= sum + x;
     write x;
     x= x+1;
     }
   write sum;

}
```

the output:

```
1
2
3  main:
4  subu $sp,$sp,4 #substract the stack pointer for main to get space for global variables
5  addi $gp,$sp,0 #copy the space to global pointer for global variables
6  subu $sp, $sp,32 #substract the stack pointer for main
7  subu $t2,$sp,32 #get the space of function
8  sw $ra,($t2) #Store the return address
9  sw $sp,4($t2) #Store the old stack pointer
10 addu $sp,$t2,0 #copy the ~~~~~
11
12 li $t2,0 #load the number on the right to assign
13 sw $t2,16($sp) #store the number into the stack
14 li $t2,1 #load the number on the right to assign
15 sw $t2,0($gp) #store the number into the global pointer
16 li $v0,5 #read a value
17 syscall #print the value
18 sw $v0,12($sp) # read store the local value on the stack
19 LABEL3:
20
21 lw $t2,0($gp) #expr left hand is global
22 sw $t2,20($gp) #expr left hand store in gp
23 lw $t3,12($sp) #expr on right hand is local
24 lw $t2,20($gp) #lw the final global left hand
25 sle $t2,$t2,$t3
26 sw $t2,20($sp)
27 lw $t0,20($sp)
28 beq $t0,0,LABEL4
29 lw $t2,16($sp) #expr left hand is local
30 sw $t2,24($sp) #expr left hand store in sp
31 lw $t3,0($gp) #expr on right hand is global
32 lw $t2,24($sp) #lw the final local left hand
33 add $t2,$t2,$t3
34 sw $t2,24($sp)
35 lw $t2,24($sp) #load the expression on the left local
36 sw $t2,16($sp) #sw the expression on the left local value in assign
37 lw $a0,0($gp) #lw the global pointer value to print
38 li $v0,1
39 syscall
40 lw $t2,0($gp) #expr left hand is global
41 sw $t2,28($gp) #expr left hand store in gp
42 li $t3,1
43 lw $t2,28($gp) #lw the final global left hand
44 add $t2,$t2,$t3
45 sw $t2,28($sp)
46 lw $t2,28($sp) #load the expression on the left local
47 sw $t2,0($gp) #sw the expression on the left global value in assign
48 j LABEL3
49
50 LABEL4 :
51
52 lw $a0,16($sp) #lw the local pointer value to print
53 li $v0,1
54 syscall
55 li $v0,0 # return zero
56 lw $ra ($sp) # get the original return address
57 lw $sp 4($sp) # get the old stack pointer
58 jr $ra #jump to the next instruction
```

MIPS OUTPUT:

```
King:spim yuhaolan$ ./spim
Loaded: ./exceptions.s
(spim) load "test1.asm"
(spim) run
8
1234567836(spim) run
```

TEST THREE:

```
int y;
void main(void)
{    int x;
     write 5;
     read x;
     write x;
     read y;
     write x+y;
}
```

The MIPS CODE:

```
1
2
3  main:
4  subu $sp,$sp,4 #substract the stack pointer for main to get space for global variables
5  addi $gp,$sp,0 #copy the space to global pointer for global variables
6  subu $sp, $sp,16 #substract the stack pointer for main
7  subu $t2,$sp,16 #get the space of function
8  sw $ra,($t2) #Store the return address
9  sw $sp,4($t2) #Store the old stack pointer
10 addu $sp,$t2,0 #copy the ~~~~~
11
12 li $a0,5
13 li $v0,1
14 syscall
15 li $v0,5 #read a value
16 syscall #print the value
17 sw $v0,8($sp) # read store the local value on the stack
18 lw $a0,8($sp) #lw the local pointer value to print
19 li $v0,1
20 syscall
21 li $v0,5 #read a value
22 syscall #print the value
23 sw $v0,0($gp) # read store the global value on the stack
24 lw $t2,8($sp) #expr left hand is local
25 sw $t2,12($sp) #expr left hand store in sp
26 lw $t3,0($gp) #expr on right hand is global
27 lw $t2,12($sp) #lw the final local left hand
28 add $t2,$t2,$t3
29 sw $t2,12($sp)
30 lw $a0,12($sp)
31 li $v0,1
32 syscall
33 lw $t2,8($sp) #expr left hand is local
34 sw $t2,12($sp) #expr left hand store in sp
35 lw $t3,0($gp) #expr on right hand is global
36 lw $t2,12($sp) #lw the final local left hand
37 add $t2,$t2,$t3
38 sw $t2,12($sp)
39 li $v0,0 # return zero
40 lw $ra ($sp) # get the original return address
41 lw $sp 4($sp) # get the old stack pointer
42 jr $ra #jump to the next instruction
```

MIPS OUTPUT:

```
Loaded: ./exceptions.s
(spim) load "test2.asm"
(spim) run
59
98
17(spim)
```

EMIT.C:

```
#ifndef SYMTABLE
#define SYMTABLE
#include "symbol.h"
#endif


#ifndef AST
#define AST
#include "AST.h"
#endif

void AssignFormals(ASTnode *p, ASTnode *q,int foffset,FILE* fp)
{
   if(p==NULL && q==NULL)
      {
        return ;
      } //If both are null then it means its a VOID and a 1 is returned.

      else
      {

//Switch case is used to be able to assign if value passed is either a
number,identifier, expression or a function call
         switch(p->right->type)
         {

          case NUMBER:
                printf("a");
                fprintf(fp, "li $t2 %d#For a scalar\n",p->right->value);
                break;
          case IDENT :
               printf("a1");

                emitASTprint(p->right,fp);
```

```
                fprintf(fp, "lw $t2 %d($sp)#Assign the word in memory to t2
\n",p->right->symbol->offset*4);
                break;
          case EXPR:
                printf("a2");

                emitASTprint(p->right,fp);
                fprintf(fp, "lw $t2 %d($sp)#Assign the word in memory to t2(for
expr)\n",p->right->symbol->offset*4);
                break;
          case CALL:
                printf("a3");

                emitASTprint(p->right,fp);
                fprintf(fp, "addu $t2 $v0 0# Assign the return value to t2\n");
                break;
           }


       }

    fprintf(fp, "subu $t3 $sp %d #Change the stack pointer and assign it to
t3\n",foffset*4);
    fprintf(fp, "sw $t2 %d($t3)#Store the t2 value in the
stack\n",q->symbol->offset*4);
    AssignFormals(p->right,q->left,foffset,fp); //Make a recursive call to assign all
actuals to formals
}




//void emit(FILE* fp,char* s,char* command,char* comment)
void emit_header(FILE *fp, int offset, int maxoffset)
```

```
{
  // update the gp and sp so that we can call main with global variable
        printf("##!!@@@the offset in main is %d!!!!\n",offset);
        fprintf(fp,"subu $sp,$sp,%d #substract the stack pointer for main to get
space for global variables\n",(offset+1)*4);//!!!!
        fprintf(fp,"addi $gp,$sp,0 #copy the space to global pointer for global
variables\n");
        fprintf(fp,"subu $sp, $sp,%d #substract the stack pointer for
main\n",maxoffset*4);//!!!!!
}

void emit_func_start(ASTnode *p, FILE* fp)
{
 char s[100];
 sprintf(s,"%s:\n",p->name);
 fprintf(fp,"\n");
 fprintf(fp,"\n");
 fprintf(fp,"%s",s);

 if(strcmp(p->name, "main")==0)
 {
   emit_header(fp, p->symbol->offset, p->value);
  }
//!! move the sp to the function     sub   sp  p->value
  fprintf(fp,"subu $t2,$sp,%d #get the space of function\n",p->value*4);
  fprintf(fp,"sw $ra,($t2) #Store the return address\n");
  fprintf(fp,"sw $sp,4($t2) #Store the old stack pointer\n");

  fprintf(fp,"addu $sp,$t2,0 #copy the ~~~~~\n");

  fprintf(fp,"\n");

}

void emit_func_return(ASTnode *p, FILE *fp)
{
  char s[100];
```

```
  if(p == NULL || p->right == NULL)
  {
   fprintf(fp,"li $v0,0 # return zero \n");
  }
  else
  {
     switch(p->right->type)
     {
       case NUMBER:
           fprintf(fp,"li $v0,%d\n", p->right->value);
         break;

       case IDENT:
         emitASTprint(p->right, fp);
        fprintf(fp,"lw $v0,%d($sp)\n", (p->right->symbol->offset)*4);

        //fprintf(fp,"lw $v0,($t2) #load $t2 into $a0 for write\n");
         //fprintf(fp, "add $v0,$t0, $zero #return with $v1\n");
         break;

       case EXPR:
         emitASTprint(p->right,fp);
         fprintf(fp,"lw $v0,%d($sp)\n", (p->right->symbol->offset)*4);
         //fprintf(fp,"add $v0,$t0,$zero  #return with $v1\n");
        break;
     }
  }

  fprintf(fp,"lw $ra ($sp) # get the original return address\n");
  fprintf(fp,"lw $sp 4($sp) # get the old stack pointer\n");
  fprintf(fp,"jr $ra #jump to the next instruction\n");
}


//lw $ra ($sp)
```

```
//##############################################//#######################
#######################//#################################################
//##############################################//#######################
#######################//#################################################
//##############################################//#######################
#######################//#################################################
```

```
void  emitASTprint(ASTnode *p, FILE *fp)// what is level
{
  int i;
              int ii;
               char * CreateTemp()
            {
                char hold[100];
                char *label;
                sprintf(hold,"LABEL%d",GTEMP++);
                label=strdup(hold);
                return (label);
            }

   //FILE * fp;
   //fp = fopen ("MIPS.txt", "w+");
  if (p == NULL ) return;
  else
   {
```

```
   switch (p->type)
  {
        //declaration
        case VARDEC :
           printf("Variable ");
           switch(p->operator)
           {
              case INTDEC : printf("INT");break;
              case VOIDDEC : printf("VOID");break;

           }
          printf(" %s",p->name);
               if (p->value > 0)
               {
                 printf("[%d]",p->value);
               }
               printf("\n");
               break;

//*************************************FUNCTION*****************
*******************************
//*************************************FUNCTION*****************
*******************************
//*************************************FUNCTION*****************
*******************************



       case FUNCTIONDEC :
       /*
  subu $sp, $sp, NUM  # number of bytes we need for the global data segment
     addi $gp, $sp, 0 # update the global pointer
     subu $sp, $sp, NUM1 # the number of bytes needed for main to run

  NUM is the global offset of all entities declared at the global level.
  NUM1 is the # of bytes main needs for its runtime stack

       */
```

```
            emit_func_start(p,fp);//p->s1  is about param   p->right is about
compoundstmt


            emitASTprint(p->right,fp);//print the compound statement

            emit_func_return(NULL,fp);




             break;
//*************************************END-
FUNCTION************************************************
//*************************************END-
FUNCTION************************************************
//*************************************END-
FUNCTION************************************************


          case BLOCK :


             printf("BLOCK STATEMENT\n");
              emitASTprint(p->right,fp);
             break;

          case PARAMEXPR :

           printf("PARAMETER ");
             switch(p->operator)
              {
                 case INTDEC : printf("INT");break;
                 case VOIDDEC : printf("VOID");break;

              }
```

```
          printf(" %s\n",p->name);
            if (p->value > 0)
               printf("[%d]",p->value);
            break;

          //assignment statement
         case ASSIGN :
         //WRONG!!!
            printf(" ASSIGNMENT STATMENT!!!!!!\n");
            emitASTprint(p->right,fp);//VAR==IDENT
           // printf("I AM HERE %d !!!!!!\n",++ii);
            //emitASTprint(p->s1,fp);//
           // printf("I AM HERE %d !!!!!!\n",++ii);
           //assignmentstmt : var '=' expressionstmt : p->right is var    p->s1 is expr

            switch(p->s1->right->type)
             {
               case NUMBER :
               // printf("I AM HERE ~~~~num %d !!!!!!\n",++ii);
                fprintf(fp,"li $t2,%d #load the number on the right to
assign\n",p->s1->right->value);

                if(p->right->symbol->level == 0)
                {
                  fprintf(fp,"sw $t2,%d($gp) #store the number into the global
pointer\n",p->right->symbol->offset*4);
                }
                else
                {

                  fprintf(fp,"sw $t2,%d($sp) #store the number into the
stack\n",p->right->symbol->offset*4);
               }
                break;

               case IDENT :
                //printf("I AM HERE ~~~~ident %d !!!!!!\n",++ii);
```

```
            if(p->right->symbol->level == 0)//check the VAR
            {
              fprintf(fp,"sw $t2,%d($gp) #sw the global value on the left in
assign\n",p->right->symbol->offset*4);
            }
            else
            {
              fprintf(fp,"sw $t2,%d($sp) #sw the local value on the left in
assign\n",p->right->symbol->offset*4);
            }

            if (p->s1->right->symbol->level == 0)//check the expr
            {
              fprintf(fp,"lw $t2,%d($gp) #lw the global value on the right in
assign\n",p->s1->right->symbol->offset*4);

            }
            else
            {
              fprintf(fp,"lw $t2,%d($sp) #lw the local value on the right in
assign\n",p->s1->right->symbol->offset*4);

            }


            break;

           case EXPR :
            // printf("I AM HERE ~~~~~expr %d !!!!!!\n",++ii);

              emitASTprint(p->s1,fp);
              fprintf(fp,"lw $t2,%d($sp) #load the expression on the left
local\n",p->s1->right->symbol->offset*4);


              if(p->right->symbol->level == 0)
```

```
                {

                  fprintf(fp,"sw $t2,%d($gp) #sw the expression on the left global
value in assign \n",p->right->symbol->offset*4);

                }
                else
                {

                  fprintf(fp,"sw $t2,%d($sp) #sw the expression on the left local value
in assign \n",p->right->symbol->offset*4);
                }

                break;

             }
            break;

        case IDENT :
            printf("IDENTIFIER !!!!%s\n",p->name);
             //check the right is null or not
            if (p->right != NULL)
            {
               printf("     Array Reference [\n");
                emitASTprint(p->right,fp);
               printf("      ] end array\n");
            }
            break;

        case EXPR :
//######################EMIT FOR EXPR
############################################################
//######################EMIT FOR EXPR
############################################################
//######################EMIT FOR EXPR
############################################################
```

```
//######################EMIT FOR EXPR
############################################################
//######################EMIT FOR EXPR
############################################################
//WRONG!!!!
        /*
         if (p->s1 == NULL)
         {
           switch (p->type)
           {
             case NUMBER :
              //store the left hand side value
             //QUSTION why need sw???
                fprintf(fp,"li $t2,%d\n",p->value);
                fprintf(fp, "sw $t2,%d($sp)\n",p->symbol->offset*4);
                break;
           }

         }
         */




         //compute the left hand side

        switch(p->right->type)
        {
          case NUMBER :
            //store the left hand side value
          //QUSTION why need sw???
             printf("~~~~~~~~~~~~~~~~~~$t2,%d\n",p->right->value);
             fprintf(fp,"li $t2,%d\n",p->right->value);
             fprintf(fp, "sw $t2,%d($sp)\n",p->symbol->offset*4);
             break;
          case IDENT :
          // ID can be x or x[100]
```

```
            if (p->right->symbol->level == 0)
            {
             fprintf(fp,"lw $t2,%d($gp) #expr left hand is
global\n",p->right->symbol->offset*4);
             fprintf(fp,"sw $t2,%d($gp) #expr left hand store in
gp\n",p->symbol->offset*4);               }
            else
            {
             fprintf(fp,"lw $t2,%d($sp) #expr left hand is
local\n",p->right->symbol->offset*4);
             fprintf(fp,"sw $t2,%d($sp) #expr left hand store in
sp\n",p->symbol->offset*4);
           }
            break;
         case EXPR ://wrong QUSTION
            emitASTprint(p->right,fp);//  QUESTION should it be repeated???????
            fprintf(fp,"lw $t2,%d($sp)\n",p->right->symbol->offset*4);//QUESTION
why I need this line?????
            fprintf(fp,"sw $t2,%d($sp)\n",p->symbol->offset*4);// should it
recusivly to call
            break;


       }


       //BOTTOM UP
       //compute the right hand side
       switch(p->s1->type)
       {
         case NUMBER :
           printf("##################$t3,%d\n",p->s1->value);

          fprintf(fp,"li $t3,%d\n",p->s1->value );
          break;
         case IDENT :
          if(p->s1->symbol->level == 0)
```

```
            {
              fprintf(fp,"lw $t3,%d($gp) #expr on right hand is
global\n",p->s1->symbol->offset*4);

            }
            else
            {
              fprintf(fp,"lw $t3,%d($sp) #expr on right hand is local
\n",p->s1->symbol->offset*4);
            }
            break;
           case EXPR :
            emitASTprint(p->s1,fp);
            fprintf(fp,"lw $t3 %d($sp)\n",p->s1->symbol->offset*4);
           // fprintf(fp,"sw $t3 %d($sp)\n",p->symbol->offset*4);// QUESTON why is
p->symbol offset??? not p->right
            break;
         }
        if (p->right->symbol->level == 0)
        {
          fprintf(fp,"lw $t2,%d($gp) #lw the final global left
hand\n",p->symbol->offset*4);// should it recusivly to call

        }
        else
        {

          fprintf(fp,"lw $t2,%d($sp) #lw the final local left
hand\n",p->symbol->offset*4);// should it recusivly to call
        }

        switch(p->operator)
            {
               case PLUS :
                 fprintf(fp,"add $t2,$t2,$t3\n");
                 fprintf(fp,"sw $t2,%d($sp)\n",p->symbol->offset*4);
                 break;
```

```
                case MINUS :
                  fprintf(fp,"sub $t2,$t2,$t3\n");
                  fprintf(fp,"sw $t2,%d($sp)\n",p->symbol->offset*4);
                  break;
                //QUESTION should we do times and division?????
                case TIMES :
                  fprintf(fp, "mult $t2,$t3\n");
                  fprintf(fp, "mflo $t2\n");
                  fprintf(fp,"sw $t2,%d($sp)\n",p->symbol->offset*4);
                  break;

                case DIVIDE :
                  fprintf(fp, "div $t2,$t3\n");
                  fprintf(fp, "mflo $t2\n");
                  fprintf(fp,"sw $t2,%d($sp)\n",p->symbol->offset*4);
                  break;
//!!WRONG
                case LESSTHANEQUAL :
                  fprintf(fp,"sle $t2,$t2,$t3\n");
                  fprintf(fp,"sw $t2,%d($sp)\n",p->symbol->offset*4);
                  break;

                case LESSTHAN :
                  fprintf(fp,"slt $t2,$t2,$t3\n");
                  fprintf(fp,"sw $t2,%d($sp)\n",p->symbol->offset*4);
                  break;

                case GREATERTHAN :
                  fprintf(fp,"sgt $t2,$t2,$t3\n");
                  fprintf(fp,"sw $t2,%d($sp)\n",p->symbol->offset*4);
                  break;

                case GREATERTHANEQUAL :
                  fprintf(fp,"sge $t2,$t2,$t3\n");
                  fprintf(fp,"sw $t2,%d($sp)\n",p->symbol->offset*4);
                  break;
```

```
            case EQUAL :
              fprintf(fp,"seq $t2,$t2,$t3\n");
              fprintf(fp,"sw $t2,%d($sp)\n",p->symbol->offset*4);
              break;

            case NOTEQUAL :
              fprintf(fp,"sne $t2,$t2,$t3\n");
              fprintf(fp,"sw $t2,%d($sp)\n",p->symbol->offset*4);


            break;


          }









//#########################################################################
######################
//#########################################################################
######################
//#########################################################################
######################
//#########################################################################
######################
//#########################END
EMIT#####################################################
//#########################################################################
######################
          printf(" EXPR");
```

```
            //emitASTprint(p->right,fp);//additiveexpression
           //relop
            //emitASTprint(p->s1,fp);//additiveexpression
           break;

        case NUMBER :
           if (p->value >0)
              printf("Number with value %d\n",p->value);
           break;

        case CALL :

        printf("I am here 1\n");



        AssignFormals(p->right,p->symbol->fparms,p->symbol->offset,fp);//Call
AssignFormals to store the actual arguments in the formals location



        printf("I am here 2\n");

        fprintf(fp,"jal %s #call the function name\n",p->name);

        printf("I am here 3\n");


           break;

        //if statement
        case IFSTMT :

//########################IF STMT
#########################################################################
####
```

```
//#######################IF STMT
#########################################################################
####
//#######################IF STMT
#########################################################################
####
//#######################IF STMT
#########################################################################
####

            printf("IF STATMENT\n");

            int ii;
            //printf("//#######################I am here %d\n",++ii);
            //hand the expression
           switch (p->right->type)
           {
                 case NUMBER :
                    fprintf(fp,"li $t0,%d #expr is number\n",p->right->value);
                    break;
                 case IDENT :
                    //!!
                    if (p->right->symbol->level == 0)
                    {

                      fprintf(fp,"lw $t0,%d($gp) #in if the expr is local
indent\n",p->right->symbol->offset*4);

                    }
                    else
                    {
                      fprintf(fp,"lw $t0,%d($sp) #in if the expr is local
indent\n",p->right->symbol->offset*4);
                    }
                    break;
                 case EXPR :
                    emitASTprint(p->right,fp);
```

```
                    fprintf(fp,"lw $t0,%d($sp) #expr is
expr\n",p->right->symbol->offset*4);
                    break;
            }

            //printf("//#######################I am here %d\n",++ii);


             char* label1=CreateTemp();
             char* label2=CreateTemp();

             fprintf(fp,"beq $t0,0,%s #if the expression is false go to
label1\n",label1);
            //printf("//#######################I am here %d\n",++ii);

            // if the expression is true we go to S1
             emitASTprint(p->s1,fp);

             if (p->s2 != NULL)
             {
               fprintf(fp,"j %s #go to label2\n",label2);
             }
            //printf("//#######################I am here %d\n",++ii);

            // if expression is false go to s2
             fprintf(fp,"\n%s : #label1 is here\n\n",label1);
             emitASTprint(p->s2,fp);

             fprintf(fp,"\n%s : #label2 is here\n\n",label2);


         break;


//#######################END
IF#################################################################
######
```

```
//######################END
IF#################################################################
######
//######################END
IF#################################################################
######

        //while loop
        case WHILEEXPR :

//######################WHILE
STMT##############################################################
#########
//######################WHILE
STMT##############################################################
#########



            printf("WHILE STATMENT!!!\n");
            char* label3=CreateTemp();//wrong why cannot be in front of printf

            fprintf(fp,"\n%s: \n\n",label3);
            switch (p->right->type)
           {
                case NUMBER :
                   fprintf(fp,"li $t0,%d\n",p->right->value);
                   break;
                case IDENT :
                   if(p->right->symbol->level == 0)
                   {
                     fprintf(fp,"lw $t0,%d($gp) #while expr is global
indent\n",p->right->symbol->offset*4);

                   }
                   else
```

```
                {
                  fprintf(fp,"lw $t0,%d($sp) #while expr is load
indent\n",p->right->symbol->offset*4);
                }
                break;
            case EXPR :
                emitASTprint(p->right,fp);
                fprintf(fp,"lw $t0,%d($sp)\n",p->right->symbol->offset*4);
                break;
        }

        char* label4=CreateTemp();

        fprintf(fp,"beq $t0,0,%s\n",label4);



         emitASTprint(p->s1,fp);
         fprintf(fp,"j %s\n",label3);

         fprintf(fp,"\n%s : \n\n",label4);
        break;
//########################END    WHILE    STMT
#########################################################################
####
//########################END    WHILE    STMT
#########################################################################
####

      //return
      case RETURNEXPR :
         printf("RETURN STATMENT\n");
          emitASTprint(p->right,fp);
          emit_func_return(p->right,fp);
          //emit fun return
         break;
      //read
```

```
        case READSTMT :
        //QUESTION!!!!!!!
              // fprintf(fp,"sub
$sp,$sp,%d\n",p->right->symbol->offset*4);//substract the sp size
               fprintf(fp,"li $v0,5 #read a value\n");
               fprintf(fp,"syscall #print the value\n");
               if(p->right->symbol->level == 0)
               {
                 fprintf(fp,"sw $v0,%d($gp) # read store the global value on the
stack\n",p->right->symbol->offset*4);

               }
               else
               {
                fprintf(fp,"sw $v0,%d($sp) # read store the local value on the
stack\n",p->right->symbol->offset*4);
               }


           printf("READ STATMENT\n");
            emitASTprint(p->right,fp);
           break;
        //write
        case WRITESTMT :

           switch(p->right->type)//check the type of write right hand
           {
              case NUMBER :
                 fprintf(fp,"li $a0,%d\n",p->right->value);
                 fprintf(fp,"li $v0,1\n");
                 fprintf(fp,"syscall\n");
                 break;
              case IDENT :
                 if (p->right->symbol->level == 0)
                 {
                   fprintf(fp,"lw $a0,%d($gp) #lw the global pointer value to
print\n",p->right->symbol->offset*4);
```

```
                fprintf(fp,"li $v0,1\n");
                fprintf(fp,"syscall\n");

              }
              else
              {
                fprintf(fp,"lw $a0,%d($sp) #lw the local pointer value to
print\n",p->right->symbol->offset*4);
                fprintf(fp,"li $v0,1\n");
                fprintf(fp,"syscall\n");
              }
              break;
            case EXPR :
              //QUESTION!!!!
               emitASTprint(p->right,fp);//QUESTION !!!!!

              fprintf(fp,"lw $a0,%d($sp)\n",p->right->symbol->offset*4);
              fprintf(fp,"li $v0,1\n");
              fprintf(fp,"syscall\n");
            break;

          }

          printf("WRITE STATMENT\n");
           emitASTprint(p->right,fp);
          break;
        case EXPRSTMT :
          printf("EXPRISSION STATMENT\n");
           emitASTprint(p->right,fp);
          break;
        case ARGLIST :
          printf("ARGLIST\n");
           emitASTprint(p->right,fp);
           emitASTprint(p->left,fp);
          break;
```

```
         default: printf("unknown type in  emitASTprint\n");


      }
        emitASTprint(p->left,fp);
    }
   //fclose(fp);
}


//!!
//!!!!


/*dummy main program so I can compile for syntax error independently
main()
{
}
 */
```