Code Generation Example

Tsan-sheng Hsu

tshsu@iis.sinica.edu.tw

`http://www.iis.sinica.edu.tw/~tshsu`

Warnings

- **This set of slides contain a "pseudo code" for a very simple compiler.**
  - **This is an example, not the standard solution.**
  - **Assume only integers.**
  - **Other solutions exist.**
- **This example does not pass LEX, YACC or GCC.**
- **Usage of this example is entirely to illustrate the high level ideas.**
- **Not responsible for any syntax errors.**
- **Please read LEX, YACC and GCC manuals carefully to avoid programming mistakes and conflicts.**
  - **web sites**
    - ▷ ***example: http://dinosaur.compilertools.net/***
  - **libraries**
  - . . .

Error handling (1/2)

```
char *error_message[MAXMSG]; /* tests for error messages */
void error_msg(int line_no; char * file_name;
int index; char *msg1; char *msg2, ...)
{
 switch index {
   ....
 /* some messages needed to be specially treated */
   case 3: /* insert *msg1 into error message */
       /* variable ‘‘*msg1’’ undefined */
       printf("line %d in file %s, error code = %d, %s %s\n",
       line_no,file_name,index,msg1,error_message[index]);
       ... // error_message[3] is "is undefined"
   case 4: /* insert *msg1 and *msg2 into error message */
       /* *msg1 is previously defined in file *msg2 */
       printf("line %d in file %s, error code = %d, %s %s %s\n",
       line_no,file_name,index,msg1,error_message[index],ksg2);
       ... // error_message[4] is "is previously defined in file"
```

Error handling (2/2)

```
...
default:
   if(index >= MAXMSG){
   printf("internal error, wrong error index  %d\n",index);
   }else{
     printf("line %d in file %s, error %d, %s\n",
     line_no,file_name,index,error_message[index]);
   }
   exit(1);
}
```

TEMP space management (1/4)

```
#define INIT_TEMP_SIZE 32 /* initial temp size */
char *temp_map; /* temp allocation map */
static int current_max; /* current temp space size */
static int max_temp_used; /* max number of temps ever used */
static int current_temp_used; /* current number of temps used */
void INITtemp(void)
{       int i;

        max_temp_used = current_temp_used = 0;
        current_max = INIT_TEMP_SIZE;
        if(temp_map=malloc(current_max)){
           for(i=0;i<current_max;i++)
              temp_map[i] = 0;
        }else{ /* system error */}
}

void STOPtemp(void)
{
        free(temp_map);
}
```

TEMP space management (2/4)

```
/* a very simple first fit allocation algorithm */
int newtemp(void)
{         if(current_temp_used == current_max){ // allocate more temps
             char * ttemp;
             if(ttemp=malloc(current_max*2)){ // space doubled
                for(i=0;i<current_max;i++) // copy old data
                   ttemp[i] = temp_map[i];
                free(temp_map); // return old space
                temp_map = ttemp; current_max *= 2;
                for(i=current_max/2;i<current_max;i++)// initialize
                   temp_map[i] = 0;
             }else{ /* internal system error; not enough space */
          }
          { int i=0;
            while(i < current_max && temp_map[i] > 0)  i++;
            current_temp_used++; // increase the temp counter
            if(i > max_temp_used) // max temp index ever used
               max_temp_used = i;
            temp_map[i] = 1; // ith space is allocated
            return(i);
          }
}
```

TEMP space management (3/4)

```
void freetemp(PLACE_TYPE vplace)
{       if(vplace.tag == TEMP_VAR){
          temp_map[vplace.offset] = 0; // free
          current_temp_used--;
        }
}

int return_max_temp(void)
{
        return max_temp_used;
}
```

TEMP space management (4/4)

- **Use the same bit map technique to record the set of registers used/unused.**
  - **For ease of description, use a byte to record whether a place is used.**
  - **Can use a bit vector to save space.**
- **Check for free register first, then temp space.**

Compute the size of A.R.

- $Proc \rightarrow procedure\ id\ (\ Para\_list\ )\ ;\ M_2\ Declaration\ ;\ Statements$
  - ▷ **{**
  - ▷ ***The size of local variables is*** $offset$
  - ▷ ***The size of temp space used is*** $return\_max\_temp()$
  - ▷ ***The size of the parameters is $4.size***
  - ▷ ***The sizes of control link, access link, and saved machine status are known constants.***
  - ▷ ***pop(offset); stopTEMP();***
  - ▷ **}**
- $M_2 \rightarrow \epsilon$
  - ▷ **{** ***/* enter a new scope */***
  - ▷ ***push(0,offset); INITtemp();*****}**
- $Para\_list \rightarrow Para\_list\ ,\ E$
  - ▷ **{** ***$$.size = $1.size + sizeof($3.szie)*** **}**
- $Para\_list \rightarrow E$
  - ▷ **{** ***$$.size = sizeof($1.type)*****}**

Label management and code generation

```
static int current_label; /* current label number */

void initLABELS(void)
{   current_label = 0;
}

int newLABEL(void)
{   return(current_label++);
}

void emitLABEL(int label)
{   fprintf(code_file,"LAB%d:\n",label);
}

void emitJUMP(int label)
{   fprintf(code_file,"goto LAB%d;\n",label);
}
```

Global constants and variables

```
/* tags for where variables are stored */
#define GLOBAL_VAR 1
#define LOCAL_VAR 2
#define TEMP_VAR 3
#define REG_VAR 4
#define PARA_VAR 5
#define NON_LOCAL_VAR 6
#define CONSTANT_VAR 7
...

int global_start; /* the starting address to put global variables */
int local_start;  /* the starting address to put local variables */
int temp_start;   /* the starting address to put temp variables */
...
```

Type definitions

```
typedef struct place_typ {
         char tag; /* where this var is stored */
         int offset; /* store offset in an storage area for variables;
                        store the constant value for constants */
         int depth_diff; /* difference in nesting depth */
} PLACE_TYPE;

typedef struct var_typ {
         char type_tag; /* procedure, variable, int constant ... */
         PLACE_TYPE place;
         char *name;
         /* for a procedure, store the following */
         int temp_start,local_start,param_start,...;
         ...
} VAR_TYPE;
```

Usage of registers

- **Resevered registers:**
  - **Frame pointer:** FP
  - **Stack pointer:** SP

```
#define FP 1
#define SP 2

...
sprintf(``I__%1d'',FP);
...
```

- **Registers resevered for code generation**
  - I_3 **and** I_4**: for assignment**
  - I_9**: for loading** r**-addresses or** l**-addresses**

Code generation routines (1/2)

```
/* op: operator, opni: operand # i */
void emitASSIGN(char operator; PLACE_TYPE opn1,opn2,opn3)
{
     switch operator {
       case '+':
            gen_r_address(opn2,3); /* load opn2 to register I__3 */
            gen_r_address(opn3,4); /* load opn3 to register I__4 */
            fprintf(code_file,"I__3 =  I__3+ I__4;\n");
            gen_l_address(opn1,3); /* store I__3 into opn1 */
            break;
       case '-': ...
       case '*': ...
       ...
       default:
             printf("internal error in code generation, operator
             %c is undefined\n",operator); exit(1);
      }
}
```

Code generation routines (2/2)

```
void emitCOND_JUMP(...)
{
...
}
void emitIO(...)
{
...
}
void emitCALL(...)
{
...
}
...
```

Code optimization (1/2)

```
// special routines for code optimization
/* op: operator, assume all are integers */
void emitASSIGN(char operator; PLACE_TYPE opn1,opn2,opn3)
{
     switch operator {
       case '+':
            if(opn2.tag == CONSTANT_VAR){
              if(opn2.offset == 0){
               gen_r_address(opn3,3); /* load opn3 to register I__3 */
               gen_l_address(opn1,3); /* store I__3 into opn1 */
              }else if(opn2.offset == 1){
               gen_r_address(opn3,3); /* load opn3 to register I__3 */
               fprintf(code_file,"I__3 += 1;\n");
               gen_l_address(opn1,3); /* store I__3 into opn1 */
              }else{
```

Code optimization (1/2)

```
            // general constants
            gen_r_address(opn3,3); /* load opn3 to register I__3 */
            fprintf(code_file,"I__3 += %d;\n",opn2.offset);
            gen_l_address(opn1,3); /* store I__3 into opn1 */
         }else if(opn3.tag == CONSTANT_VAR){
         ...
         }else{
            gen_r_address(opn2,3); /* load opn2 to register I__3 */
            gen_r_address(opn3,4); /* load opn3 to register I__4 */
            fprintf(code_file,"I__3 = I__3+ I__4;\n");
            gen_l_address(opn1,3); /* store I__3 into opn1 */
         }
         break;

      ...
   }
}
```

Generate r-address (1/2)

```
/* load the value at ‘‘where’’ into register # result_r */
void gen_r_address(PLACE_TYPE where; int result_r)
{
 switch where.tag {
 case GLOBAL_VAR:
  fprintf(code_file,"I__%1d = AVAL__S(%d);\n",
                        result_r,global_start+where.offset);
  break;
 case LOCAL_VAR:
  /* make sure I__9 is not used for other purposes */
  fprintf(code_file,"I__9=I__%1d+%d\n",FP,local_start+where.offset);
  fprintf(code_file,"I__%1d = AVAL__S(I__9);\n",result_r);
  break;
```

Generate r-address (2/2)

```
 case TEMP_VAR:
  /* r_map[0] = 5, r_map[1] = 6, ... index of free register */
  if(where.offset <= FREE_REGISTERS){
     // these are allocated in the registers
     fprintf(code_file,"I__%1d = AVAL__S(I__%d);\n",result_r,
     r_map[where.offset]);
  }else{
     /* make sure I__9 is not used for other purposes */
     fprintf(code_file,"I__9=I__%1d+%d\n",FP,temp_start+where.offset);
     fprintf(code_file,"I__%1d = AVAL__S(I__9);\n",result_r);
  }
  break;
 case CONSTANT_VAR:
  fprintf(code_file,"I__%1d = %d;\n",result_r,where.offset); break;
 ...
 }
}
```

Generate l-address

```
/* store the value at register # result_r into the place ‘‘where’’ */
void gen_l_address(PLACE_TYPE where; int result_r)
{
 switch where.tag {
 case GLOBAL_VAR:
   fprintf(code_file,"ASSET__S(%d,I__%1d);\n",
                   global_start+where.offset,result_r);
   break;
 case LOCAL_VAR:
  /* make sure I__9 is not used for other purposes */
   fprintf(code_file,"I__9=I__%1d+%d\n",FP,local_start+where.offset);
   fprintf(code_file,"ASSET__S(I__9,I__%1d);\n",result_r);
   break;
 case TEMP_VAR:
   ...
 default: /* internal error */
 }
}
```

YACC rules

```
%union{
    int ival;
    double fval;
    VAR_TYPE vval;
}
/* define the return type of ‘‘expr’’ to be VAR_TYPE */
%type <vval> expr
...
expr : expr '+' expr
          {
             $$.place.offset = newtemp();  $$.place.tag = TEMP_VAR;
             emitASSIGN('+',$$.place,$1.place,$2.place);
             freetemp($1.place);     freetemp($2.place);  ...
          }
       | expr '-' expr   {...}
       ...
       | id {...}
...
```