

Expressions

Expression Structure

```
typedef enum expr_kind EX_KIND;  
enum expr_kind {GlobalV, LocalV, LocalA, Cons, RegisterV,  
                RegisterA, FuncRef};
```

```
typedef struct expr_info EXPR;
```

```
struct expr_info  
{ TYPE *ex_type;  
  EX_KIND ex_kind;  
  int ex_inv;  
  EXPR *ex_next;  
}; /* expr_info */
```

Code for newexpr

```
EXPR *newexpr (TYPE *type, EX_KIND kind, int inv)
{ register EXPR *e;

  e = (EXPR *)malloc(sizeof(EXPR));

  e->ex_type = type;
  e->ex_kind = kind;
  e->ex_inv = inv;
  e->ex_next = NULL;
  return(e);
} /* newexpr */
```

Code for makeexpr

```
EXPR *makeexpr (ID *st_ptr)
{  TYPE *type;
   EX_KIND kind;
   int inv;

   type = st_ptr->id_type;
   inv = st_ptr->id_addr;
   if (st_ptr->id_level == GLOBALSCOPE)
       kind = GlobalV;
   else if (st_ptr->id_level == currscope)
       if (st_ptr->id_kind == RParameter)
           kind = LocalA;
       else
           kind = LocalV;
   else
       kind = LocalV;
   return(newexpr(type, kind, inv));
} /* makeexpr */
```

Constant Expression

```
Expression : ...
            | INTCONSTSY
              { $$ = newexpr(int_type, Cons, $1); }
            | CHARCONSTSY
              { $$ = newexpr(char_type, Cons, $1); }
            | STRINGCONSTSY
              { $$ = strexpr($1); }
            ...
            ;
```

String Expression

```
EXPR *strexpr (char *str)
{  EXPR *ex_temp;
   STR *str_temp;

   ex_temp = newexpr(str_type, Cons, sdatasize);
   sdatasize += strlen(str) + 1;
   str_temp = (STR *)malloc(sizeof(STR));
   if (str_head == NULL)
       str_head = str_end = str_temp;
   else
       str_end->str_next = str_temp;
   str_end = str_temp;
   str_temp->str_ptr = str;
   return(ex_temp);
} /* strexpr */
```

LValue

```
LValue : . . .
      | IDENTSY
        {$$ = lvalue($1);}
      ;
```

Binary Operators

```
Expression : ...
            | Expression ADDSY Expression
              { $$ = binop($1, AddOp, $3); }
            | Expression SUBSY Expression
              { $$ = binop($1, SubOp, $3); }
            | Expression MULSY Expression
              { $$ = binop($1, MulOp, $3); }
            | Expression DIVSY Expression
              { $$ = binop($1, DivOp, $3); }
            | Expression MODSY Expression
              { $$ = binop($1, ModOp, $3); }
            ;
            ...
```

Code for binop

```
EXPR *binop (EXPR *exp1, char op, EXPR *exp2)
{ int reg1, reg2, reg3;

  ... /* check that the types of the two operands are compatible with the */
  /* operation being performed. If not, issue an appropriate error */
  /* message and quit. */
  ... /* get the first expression in reg1 */
  ... /* get the second expression in reg2 */
  reg3 = getreg();
  switch (op)
  { ...
    case AddOp :
      printf("add\t%d,%d,%d\n", reg3, reg1, reg2); break;
    case SubOp :
      printf("sub\t%d,%d,%d\n", reg3, reg1, reg2); break;
    case MulOp :
      printf("mult\t%d,%d,%d\n", reg3, reg1, reg2); break;
    case DivOp :
      printf("div\t%d,%d,%d\n", reg3, reg1, reg2); break;
    case ModOp :
      printf("rem\t%d,%d,%d\n", reg3, reg1, reg2); break;
  } /* switch */
  retreg(reg1); retreg(reg2); free(exp2);
  exp1->ex_kind = RegisterV;
  exp1->ex_inv = reg3;
  return(exp1);
} /* binop */
```

Unary Minus

```
Expression : . . . .  
           | SUBSY Expression %prec UNMINUSSY  
           { $$ = unop(SubOp, $2); }  
           . . . .  
           ;
```

MIPS Code for Comparison

s? DEST, S1, S2

Boolean Operators

```
Expression : . . . .  
            | Expression ORSY Expression  
              {$$ = binop($1, OrOp, $3);}  
            | Expression ANDSY Expression  
              {$$ = binop($1, AndOp, $3);}  
            | NOTSY Expression  
              {$$ = unop(NotOp, $2);}  
            . . . .  
            ;
```