CS323 SPL Compiler - Phase 4

GuTao
GuTaoZi@GitHub

Artanisax
Artanisax@GitHub

ShadowStorm
Jayfeather233@GitHub

1 Quick Start

1.1 Project Structure

```
Phase4
├── bin
│   ├── opt   // Intermediate code optimizer
│   ├── splc  // Intermediate code generator
│   └── tcg   // Target code generator
├── include
│   ├── backend
│   │   ├── back_utility.h
│   │   ├── mips32.h
│   │   └── tac.h
│   └── frontend
│       ├── front_utility.h
│       ├── IRgen.h
│       ├── IRortho.h
│       ├── ortho.h
│       ├── treeNode.h
│       ├── type.h
│       ├── type_op.h
│       ├── uthash.h
│       └── utstack.h
├── Makefile
├── report
│   ├── Phase4.pdf
│   └── requirements.pdf
├── sample
├── src
│   ├── backend
│   │   ├── back_utility.c
│   │   ├── main.c
│   │   ├── Makefile
│   │   ├── mips32.c
│   │   ├── optimizer.c
│   │   └── tac.c
│   └── frontend
│       ├── front_utility.c
│       ├── IRgen.c
│       ├── IRortho.c
│       ├── lex.l
│       ├── Makefile
│       ├── ortho.c
│       ├── syntax.y
│       ├── treeNode.c
│       ├── type.c
│       └── type_op.c
├── test
├── test_all.sh
└── test_back.sh
```

1.2 Environment

Tool	Version
C	C99
Bison	3.0.4+
Flex	2.6.4

1.3 Build the Compiler

Under the root folder of this project, execute the following commands to build the SPL compiler as `splc`, IR optimizer as `opt`, target code generator as `tcg`, under the `/bin` folder:

- `make all`: default option, build the compiler tool chain using GCC.
- `make allt`: build the compiler tool chain using TCC.

Or you may enter the `src/frontend` and the `src/backend` folders to build frontend tools and backend tools separately:

Under `src/frontend`,

- `make splc`: build the compiler using GCC.
- `make splct`: build the compiler using Tiny C Compiler(TCC).

Under `src/backend`,

- `make opt`: build the intermediate code optimizer using GCC.
- `make optt`: build the intermediate code optimizer using TCC.
- `make tcg`: build the target code generator using GCC.
- `make tcgt`: build the target code generator using TCC.

1.4 Run the Compiler

The executable `splc` has integrated all functions and can directly compile `.spl` files into `.s` files:

```
./bin/splc filename.spl
```

By executing the command above, `filename.ir0`, `filename.ir` and `filename.s` will be generated in the same folder of `filename.spl`, corresponding to the raw IR code, the optimized IR code and the MIPS32 assembly code.

To use the target code generator separately, execute the following command:

```
./bin/tcg filename.ir
```

2 Features

Our target code generator supports translation from the intermediate code generated from SPL to MIPS32 code. Basic statements like integer alrithmetic operations and conditional/unconditional jumps, and features like memory allocation are supported.

2.1 Register Allocation

Our SPL Compiler uses 32 registers mostly according to the MIPS32 standard, with some slight modification to the usage of temporary registers and value-saving registers. We use `$t0`-`$t9` and `$s0`-`$s7` registers to store all intermediate values generated, like results of intermediate expressions and the value of variables etc.

The detailed usage of registers of SPL Compiler is shown in 表 1:

Register	Number	Description
`$zero`	0	**Constant** 0
`$at`	1	**Assembler temporary**: reserved for assembler
`$v0,$v1`	2-3	**Values**: expression evaluation or function return
`$a0,$a1,$a2,$a3`	4-7	**Arguments**: function argumetns
`$t0,$t1,$t2,$t3 $t4,$t5,$t6,$t7 $s0,$s1,$s2,$s3 $s4,$s5,$s6,$s7 $t8,$t9`	8-25	**Temporaries and Saved Values**: values may preserved across procedure calls, the caller/callee can both be responsible for saving values
`$k0,$k1`	26-27	Resesrved for OS kernel
`$gp`	28	**Global pointer**: points to the middle of a 64K static data segment
`$sp`	29	**Stack pointer**: points to thte top of the stack
`$fp`	30	**Frame pointer**
`$ra`	31	**Return address**

表 1 Register Used for SPL Compiler

For register allocation, we use **Least-Recently-Used** (LRU) strategy: whenever a register is needed for storing a new value while there is no available register, the compiler invokes `get_LRU_victim()` to check the `recent` attributes of registers, and spills the least recently used one.

For parameter passing, if a function has no more than 4 parameters, all of the parameters are saved in registers `$a0-$a3`, otherwise the parameters cannot be saved using registers will be saved on stack.

2.2 Compound Types: Struct and Array

Although this part is not required, we have implemented the memory management support for grammars of compound types like `struct` and arrays. For `DEC` statements in the intermediate code, we allocate dynamic space on `$fp`, and use pointers to access the elements of array and the members of struct. When passed as parameters, the arrays and structures are passed to the callee using their pointers pointing to addresses on `$fp`.

```
tac *emit_dec(tac *dec)
{
    /* NO NEED TO IMPLEMENT */
    Register x = fp;
    VarDesc *p = alloc_stack_space(_tac_quadruple(dec).var);
    Register y = get_register_w(_tac_quadruple(dec).var);
    _mips_iprintf("move %s, %s", _reg_name(y), _reg_name(x));
    _mips_iprintf("addi %s, %s, %d", _reg_name(x), _reg_name(x),
_tac_quadruple(dec).size);
    _mips_iprintf("sw %s, -%d($sp)", _reg_name(y), p->offset);
    return dec->next;
}
```

3 Starter Code Bugfix

There is a minor bug in the starter code:

In `emit_mul()` and `emit_div()`, the `lw` instruction should be `li`

```
    _mips_iprintf("lw %s, %d", _reg_name(y), _tac_quadruple(mul).r1->int_val);
             //   ^ should be `li`
```

Acknowledgement

Implementing a compiler from scratch has significantly enriched our comprehension of compilation principles. We acknowledge the value of this experience and are motivated to persistently refine our coding skills.

By the end of this semester, we extend our sincere appreciation to Prof. Liu and the dedicated teaching assistants for diligent efforts in meticulously reviewing every assignment and project, coupled with prompt responses to our inquiries, have greatly contributed to every phase of this project.

Thank you again for your unwavering commitment throughout this semester!