コンパイラ実験

2班 伏見遼平 小泉実加

方針

- 速度は気にせず、ちゃんと動くことを優先
- ただし、同じ用途の部分は関数にまとめて、あとで内部実装を最適化しやすいようにした
  - ex.）cogen_mov（mem -> mem の移動を避けながらいいかんじに movl する）

tokenizer

- strdup を活用
  - こんな関数があるなんて知らなかった
  - よく自分で作ってたけど…

parser

- 特に工夫なし

envの実装

- scan_syntree_program
  - 再帰的に、すべての部分式 expr の expr->info に syntree_info を書き込む
  - 今いるスコープブロックに対応する env（infoのリストを持っている）に、expr->info を追加
  - 変数が登場したら、env に問い合わせて、すでに定義されていればそのアドレスを使う

env.h

- `env_t scan_syntree_program(program_t prog);`
- `env_t scan_syntree_fun_def(fun_def_t fun_def, env_t env_global);`
- `env_t scan_syntree_decls(var_decl_list_t decls, env_t env);`
- `env_t scan_syntree_params(var_decl_list_t decls, env_t env);`
- `env_t scan_syntree_stmt(stmt_t s, env_t p_env);`
- `env_t scan_syntree_expr(expr_t e, env_t env);`

code generator

- ブール代数の扱いが大変だった
  - 型とキャストを導入しないならば、%eflags を部分式間で受け渡さず、即 int 0/1 に変換した方がよかった
  - 逆に早めにキャストを導入しておけば楽だった (%eflags を int に変換してあるレジスタに入れる関数など..)

codegen.h

- `void cogen_program(FILE *fp, program_t ds);`
- `void cogen_fun_def(FILE *fp, fun_def_t f);`
- `void cogen_stmt(FILE *fp, stmt_t s);`
- `void cogen_expr(FILE *fp, expr_t e);`

- 補助関数群
  - `char* cogen_addr(syntree_info_t info);`
  - `char* cogen_pr_reg(reg_t reg);`
  - `char* cogen_mov(FILE* fp, expr_t right, expr_t left);`
  - `char* cogen_mov_reg(FILE *fp, expr_t e, reg_t reg);`
  - `char* get_label();`

詰まったところ

- 0(%esp)と(%esp)は違う!?!?

結果

- basic_tests のテストは全て通過!

結果

- basic_tests のテストは全て通過!（嘘）
  - break と continue は動かないのでその部分のコードは消した

結果

- int_app_tests には（幸いにも）break と continue が含まれていなかったので、動いた

実行時間の比較（秒）

	gcc	gcc -O3	cogen
pi.c	2.318	0.793	4.141
fib.c	0.756	0.341	1.400
prime	-	-	NG

反省

- 大きく詰まったことはなかった
  - 時間外労働が少なかったのが原因では???
- 2人の分担がうまくできなかった
  - “env.h”インターフェイスの共有不足で、mergeするのに1日くらい使った

今後の課題

- prime.c が NG になった理由を調査
- continue, break を実装する
- ブール代数をよりスマートに実装する

高速化

- レジスタをできるだけ使う
  - scan_syntree_program がうまく作れればよい（最初の部分式への割当がイイカンジにできていればよい）
  - 手で行うと難しいし、上の部分木で利用しているレジスタとかぶると困る
  - 部分木と env が「いま使えるレジスタ」の情報を管理し、scan_syntree 中にその情報を使う
  - ex. 上の部分木で eax, ebx を使っているなら、下では ecx, edx, esi, edi が利用可能

高速化

- code generator の前にプリプロセッサで冗長な命令をのぞく
  - x = 1 + 2; //-> 3 に置き換え可能
  - 使われていない変数の検出など