```c
#include "genmc.h"
#include "gentac.h"
#include <stdlib.h>
#include <stdio.h>
#include "C.tab.h"
#include "regstack.h"
#include "mc_env.h"
#include "string.h"

#define INSN_BUF 64

extern TOKEN *lookup_loc(TOKEN*, FRME*);
extern TOKEN *assign_to_var(TOKEN*, FRME*,TOKEN*);
extern void declare_var(TOKEN*, FRME*);
extern void declare_fnc(TOKEN*, CLSURE*, FRME*);
extern CLSURE *find_fnc(TOKEN* , FRME* );
extern TOKEN* use_temp_reg(FRME *);

int call_stack;

TAC* find_endproc(TAC* i){
  int nested_depth = 0;
  while(i != NULL){
    if(i->op == tac_innerproc){
      nested_depth++;
    }
    if(i->op == tac_endproc){
      if(nested_depth == 0){
        return i;
      }
      else{
        nested_depth--;
      }
    }
    i = i->next;
  }
  return i;
}

MC* find_lst(MC* mc){
  while(mc->next != NULL){
    mc = mc->next;
  }
  return mc;
}

int count_locals(TAC* i){
  int n = 0;
  while(i != NULL && i->op != tac_endproc){
    if(i->op == tac_store){
      n++;
    }
    i = i->next;
  }
  return n;
}

TOKEN * new_dst(FRME *e){
    for(int i=0; i<MAXREGS; i++){
        if(!reg_in_use(i,e)){
```

```
 61             TOKEN* dst = (TOKEN*)malloc(sizeof(TOKEN));
 62             if(dst==NULL){printf("fatal: failed to generate
    destination\n");exit(1);}
 63             dst->type=IDENTIFIER;
 64             dst->lexeme = (char*)calloc(1,2);
 65             sprintf(dst->lexeme,"t%i",i);
 66             dst->value = i;
 67             return dst;
 68         }
 69     }
 70 }
 71
 72 MC* new_minus(TAC* tac){
 73   MC *mc = malloc(sizeof(MC));
 74   mc->insn = malloc(sizeof(INSN_BUF));
 75   sprintf(mc->insn,"sub
    $%s,$%s,$%s",tac->stac.dst->lexeme,tac->stac.src1->lexeme,tac->stac.src2->lexe
    me);
 76   return mc;
 77 }
 78 MC* new_div(TAC* tac){
 79   MC *mc = malloc(sizeof(MC));
 80   mc->insn = malloc(sizeof(INSN_BUF));
 81   sprintf(mc->insn,"div
    $%s,$%s",tac->stac.src1->lexeme,tac->stac.src2->lexeme);
 82   mc->next = malloc(sizeof(MC));
 83   mc->next->insn = malloc(sizeof(INSN_BUF));
 84   sprintf(mc->next->insn,"mflo $%s",tac->stac.dst->lexeme);
 85   return mc;
 86 }
 87 MC* new_mod(TAC* tac){
 88   MC *mc = malloc(sizeof(MC));
 89   mc->insn = malloc(sizeof(INSN_BUF));
 90   sprintf(mc->insn,"div
    $%s,$%s",tac->stac.src1->lexeme,tac->stac.src2->lexeme);
 91   mc->next = malloc(sizeof(MC));
 92   mc->next->insn = malloc(sizeof(INSN_BUF));
 93   sprintf(mc->next->insn,"mfhi $%s",tac->stac.dst->lexeme);
 94   return mc;
 95 }
 96 MC* new_mult(TAC* tac){
 97   MC *mc = malloc(sizeof(MC));
 98   mc->insn = malloc(sizeof(INSN_BUF));
 99   sprintf(mc->insn,"mult
    $%s,$%s",tac->stac.src1->lexeme,tac->stac.src2->lexeme);
100   mc->next = malloc(sizeof(MC));
101   mc->next->insn = malloc(sizeof(INSN_BUF));
102   sprintf(mc->next->insn,"mflo $%s",tac->stac.dst->lexeme);
103   return mc;
104 }
105 MC* new_plus(TAC* tac){
106   MC *mc = malloc(sizeof(MC));
107   mc->insn = malloc(sizeof(INSN_BUF));
108   sprintf(mc->insn,"add
    $%s,$%s,$%s",tac->stac.dst->lexeme,tac->stac.src1->lexeme,tac->stac.src2->lexe
    me);
109   return mc;
110 }
111
112 MC* init_mc(){
```

```
113     MC *mc = malloc(sizeof(MC));
114     mc->insn = malloc(sizeof(INSN_BUF));
115     mc->insn = ".globl main";
116     mc->next = malloc(sizeof(MC));
117     mc->next->insn = malloc(sizeof(INSN_BUF));
118     mc->next->insn = ".text";
119     return mc;
120 }
121
122 MC* make_syscall(int code){
123   MC* mc;
124   mc = malloc(sizeof(MC));
125   mc->insn = malloc(sizeof(INSN_BUF));
126   sprintf(mc->insn,"li $v0 %d",code);
127
128   MC* last = find_lst(mc);
129   last->next =  malloc(sizeof(MC));
130   last->next->insn = malloc(sizeof(INSN_BUF));
131   last->next->insn = "syscall";
132   return mc;
133 }
134
135 MC* new_smpl_ld(FRME* e, TOKEN* src, TOKEN* dst){
136   MC *mc = malloc(sizeof(MC));
137     mc->insn = malloc(sizeof(INSN_BUF));
138     if(src->type == CONSTANT){
139         sprintf(mc->insn, "li $%s,%d",dst->lexeme,src->value);
140     }
141     else if(src->type == IDENTIFIER){
142         TOKEN *loc = lookup_loc(src,e);
143         sprintf(mc->insn, "move $%s,$%s",dst->lexeme,src->lexeme);
144     }
145     return mc;
146 }
147
148 TOKEN* lookup_from_memory(TOKEN* name, FRME* e, AR* ar){
149     MC *mc = malloc(sizeof(MC));
150     mc->insn = malloc(sizeof(INSN_BUF));
151     mc->insn = "# Looking up token from memory";
152     MC* last = find_lst(mc);
153     int j = 0;
154     TOKEN* t;
155    while(e != NULL){
156     BNDING *bindings = e->bindings;
157     int i = 1;
158     while(bindings != NULL){
159         if(bindings->name == name){
160           t = new_token(IDENTIFIER);
161           t->lexeme = malloc(sizeof(INSN_BUF));
162           sprintf(t->lexeme,"%d($sp)",call_stack+ar->size-e->stack_pos-8-4*i);
163           return t;
164         }
165         if(bindings->type == IDENTIFIER){
166           i++;
167         }
168         bindings = bindings->next;
169     }
170     j+= e->size;
171     e = e->next;
172   }
```

```
173   return t;
174 }
175
176 MC* new_ld(FRME *e, TAC* tac, AR* curr){
177     MC *mc = malloc(sizeof(MC));
178     mc->insn = malloc(sizeof(INSN_BUF));
179     if(tac->ld.src1->type == CONSTANT){
180         sprintf(mc->insn, "li
    $%s,%d",tac->ld.dst->lexeme,tac->ld.src1->value);
181     }
182     else if(tac->ld.src1->type == IDENTIFIER){
183         TOKEN *loc = lookup_loc(tac->ld.src1,e);
184         if(loc == NULL){
185           loc = lookup_from_memory(tac->ld.src1,e,curr);
186           sprintf(mc->insn, "lw $%s, %s",tac->ld.dst->lexeme,loc->lexeme);
187         }
188         else{
189           sprintf(mc->insn, "move $%s,$%s",tac->ld.dst->lexeme,loc->lexeme);
190         }
191     }
192     return mc;
193 }
194
195  MC* new_str(TAC* tac, FRME* e){
196   MC *mc = malloc(sizeof(MC));
197   MC *last;
198   mc->insn = malloc(sizeof(INSN_BUF));
199   TOKEN* t = lookup_loc(tac->ld.dst,e);
200   if(t == NULL){
201     declare_var(tac->ld.dst,e);
202     assign_to_var(tac->ld.dst,e,tac->ld.src1);
203   }
204   else if(t->value != tac->ld.src1->value) {
205     assign_to_var(tac->ld.dst,e,tac->ld.src1);
206   }
207   return mc;
208 }
209
210 MC* new_ift(TAC* tac, FRME* e){
211   TOKEN* dst1 = lookup_loc(tac->ift.op1,e);
212   MC* mc = NULL;
213   if(dst1 == NULL){
214     dst1 = new_dst(e);
215     declare_var(tac->ift.op1,e);
216     assign_to_var(tac->ift.op1,e,dst1);
217     mc = new_smpl_ld(e,tac->ift.op1,dst1);
218   }
219   TOKEN* dst2 = lookup_loc(tac->ift.op2,e);
220   if(dst2 == NULL){
221     dst2 = new_dst(e);
222     declare_var(tac->ift.op2,e);
223     assign_to_var(tac->ift.op2,e,dst2);
224     if(mc != NULL){
225       mc->next = new_smpl_ld(e,tac->ift.op2,dst2);
226     }
227     else {mc = new_smpl_ld(e,tac->ift.op2,dst2);}
228   }
229
230   MC* last = find_lst(mc);
231   if(last != NULL){
```

```
232     last->next = malloc(sizeof(MC));
233     last->next->insn = malloc(sizeof(INSN_BUF));
234     last = last->next;
235   }
236   else{
237     last = malloc(sizeof(MC));
238     last->insn = malloc(sizeof(INSN_BUF));
239   }
240
241   switch(tac->ift.code){
242     case '>':
243       sprintf(last->insn,"ble $%s $%s
    %s",dst1->lexeme,dst2->lexeme,tac->ift.lbl->lexeme);
244       break;
245     case '<':
246       sprintf(last->insn,"bge $%s $%s
    %s",dst1->lexeme,dst2->lexeme,tac->ift.lbl->lexeme);
247       break;
248     case EQ_OP:
249       sprintf(last->insn,"bne $%s $%s
    %s",dst1->lexeme,dst2->lexeme,tac->ift.lbl->lexeme);
250       break;
251     case NE_OP:
252       sprintf(last->insn,"beq $%s $%s
    %s",dst1->lexeme,dst2->lexeme,tac->ift.lbl->lexeme);
253       break;
254     case LE_OP:
255       sprintf(last->insn,"bgt $%s $%s
    %s",dst1->lexeme,dst2->lexeme,tac->ift.lbl->lexeme);
256       break;
257     case GE_OP:
258       sprintf(last->insn,"blt $%s $%s
    %s",dst1->lexeme,dst2->lexeme,tac->ift.lbl->lexeme);
259   }
260   delete_constants(e);
261   return mc;
262 }
263
264 MC* new_gtl(TAC* i){
265   MC* mc = malloc(sizeof(MC));
266   mc->insn = malloc(sizeof(INSN_BUF));
267   sprintf(mc->insn,"j %s",i->gtl.lbl->lexeme);
268   return mc;
269 }
270
271 MC* new_lbli(TAC* i){
272   MC* mc = malloc(sizeof(MC));
273   mc->insn = malloc(sizeof(INSN_BUF));
274   sprintf(mc->insn,"%s:",i->lbl.name->lexeme);
275   return mc;
276 }
277
278 MC* save_frame(AR* ar, FRME *e){
279   MC *mc = malloc(sizeof(MC));
280   mc->insn = malloc(sizeof(INSN_BUF));
281   mc->insn = "# Saving frame";
282   MC* last = find_lst(mc);
283   int i = 0;
284   while(e != NULL && ar->arity != 0){
285     BNDING *bindings = e->bindings;
```

```
286     int i = 1;
287     while(bindings != NULL){
288         last->next = malloc(sizeof(MC));
289         last->next->insn = malloc(sizeof(INSN_BUF));
290         if(bindings->type == IDENTIFIER){
291           sprintf(last->next->insn,"sw $%s
    %d($sp)",bindings->loc->lexeme,8+4*i);
292            i++;
293         }
294 
295         bindings = bindings->next;
296         last = find_lst(last);
297     }
298     break;
299   }
300   last->next = malloc(sizeof(MC));
301   last->next->insn = malloc(sizeof(INSN_BUF));
302   last->next->insn = "# End of saving frame";
303   return mc;
304 }
305 
306 MC* gen_frame(AR* ar){
307 
308 
309   MC *mc = malloc(sizeof(MC));
310   mc->insn = malloc(sizeof(INSN_BUF));
311   mc->insn = "# Creating new frame";
312   MC* last = find_lst(mc);
313 
314   //allocate stack space for new frame
315   last->next = malloc(sizeof(MC));
316   last->next->insn = malloc(sizeof(INSN_BUF));
317   sprintf(last->next->insn,"addiu $sp, $sp -%d",ar->size);
318   last = find_lst(mc);
319 
320   //load return address
321   last->next = malloc(sizeof(MC));
322   last->next->insn = malloc(sizeof(INSN_BUF));
323   sprintf(last->next->insn,"sw $ra, 4($sp)");
324   last = find_lst(mc);
325 
326   //load new size into reg
327   last->next = malloc(sizeof(MC));
328   last->next->insn = malloc(sizeof(INSN_BUF));
329   sprintf(last->next->insn,"li $t1, %d",ar->size);
330   last = find_lst(mc);
331 
332    //store size on stack
333   last->next = malloc(sizeof(MC));
334   last->next->insn = malloc(sizeof(INSN_BUF));
335   last->next->insn = "sw $t1, 0($sp)";
336   last = find_lst(mc);
337 
338   last->next = malloc(sizeof(MC));
339   last->next->insn = malloc(sizeof(INSN_BUF));
340   last->next->insn = "# End of creating frame";
341   return mc;
342 }
343 
344 MC* gen_globframe(TAC* tac, FRME* e, AR* global){
```

```
345   global->sl = 0;
346   global->size = 12;
347   global->arity = 0;
348   MC *mc = malloc(sizeof(MC));
349   mc->insn = malloc(sizeof(INSN_BUF));
350   mc->insn = "#saving global frame";
351   MC* last = find_lst(mc);
352   CLSURE* f;
353   while(tac != NULL){
354       switch(tac->op){
355         case(tac_load):
356            last->next = new_ld(e,tac,global);
357            last = find_lst(last);
358            last->next = new_str(tac->next,e);
359            global->size+=4;
360            global->arity++;
361            break;
362         case(tac_proc):
363           f = malloc(sizeof(CLSURE));
364           f->env = e;
365           f->code = tac;
366           f->processed = 0;
367           declare_fnc(tac->proc.name,f,e);
368           tac = find_endproc(tac);
369       }
370       tac = tac->next;
371   }
372   e->size = global->size;
373   MC* first =malloc(sizeof(MC));
374   first->insn = malloc(sizeof(INSN_BUF));
375   first->insn = "main: ";
376   MC* r = find_lst(first);
377   r->next = gen_frame(global);
378   r = find_lst(r);
379   r->next = mc;
380   r = find_lst(r);
381   r->next = save_frame(global,e);
382   return first;
383 }
384
385 AR* calculate_frame(AR* old, TAC* tac){
386   AR* new = malloc(sizeof(AR));
387   int locals = count_locals(tac);
388   new->arity = locals + tac->proc.arity;
389   new->size = (locals*4) + (tac->proc.arity*4)+12;
390   new->sl = old->sl+1;
391   return new;
392 }
393
394 MC* restore_frame(AR* ar, FRME *e){
395   MC *mc = malloc(sizeof(MC));
396   mc->insn = malloc(sizeof(INSN_BUF));
397   mc->insn = "# Restoring frame";
398   MC* last = find_lst(mc);
399   int i = 0;
400   while(e != NULL && ar->arity != 0){
401     BNDING *bindings = e->bindings;
402     int i = 1;
403     while(bindings != NULL){
404         last->next = malloc(sizeof(MC));
```

```
405         last->next->insn = malloc(sizeof(INSN_BUF));
406         if(bindings->type == IDENTIFIER){
407           sprintf(last->next->insn,"lw $%s
    %d($sp)",bindings->loc->lexeme,8+4*i);
408         }
409         i++;
410         bindings = bindings->next;
411         last = find_lst(last);
412     }
413     break;
414   }
415   //restore return address
416   last->next = malloc(sizeof(MC));
417   last->next->insn = malloc(sizeof(INSN_BUF));
418   last->next->insn = "lw $ra 4($sp)";
419   last = find_lst(last);
420
421   last->next = malloc(sizeof(MC));
422   last->next->insn = malloc(sizeof(INSN_BUF));
423   last->next->insn = "# End of restoring frame";
424   return mc;
425 }
426
427 FRME *extend_frme(FRME* e, TAC *ids, TOKENLIST *args){
428
429     FRME* new_frame = malloc(sizeof(FRME));
430     if(ids == NULL && args == NULL) {return new_frame;}
431     BNDING *bindings = NULL;
432     new_frame->bindings = bindings;
433     //while (ids != NULL && args != NULL) {
434        TOKENLIST* tokens = ids->proc.args;
435        TOKEN* loc;
436        while(tokens != NULL && args != NULL){
437             declare_var(tokens->name,new_frame);
438             assign_to_var(tokens->name,new_frame,new_dst(new_frame));
439             tokens=tokens->next;
440             args = args->next;
441        }
442        if(!(tokens == NULL && args == NULL)){
443            printf("error: invalid number of arguments and/or tokens,
    exiting...\n");exit(1);
444        }
445     return new_frame;
446 }
447
448
449 MC *call_func(TOKEN* name, TAC* call, FRME* e, AR* curr){
450   TOKEN* t = (TOKEN *)name;
451   CLSURE *f = find_fnc(t,e);
452   MC *mc = malloc(sizeof(MC));
453   mc->insn = malloc(sizeof(INSN_BUF));
454   if(!f->processed){
455     f->processed = 1;
456     FRME* ef;
457     if(call != NULL){
458       ef = extend_frme(e,f->code,call->call.args);
459     }
460     else{
461       ef = extend_frme(e,f->code,NULL);
462     }
```

```
463     ef->next = f->env;
464     call_stack += curr->size;
465     ef->stack_pos = call_stack;
466     mc = gen_mc0(f->code,ef,curr);
467   }
468    return mc;
469 }
470
471
472 MC* new_func_rtn(TAC* i){
473   MC *mc = malloc(sizeof(MC));
474   mc->insn = malloc(sizeof(INSN_BUF));
475   if(i->next == NULL){
476   }
477   else{
478     mc->insn = "jr $ra";
479   }
480   return mc;
481 }
482
483 MC* new_rtn(TAC* tac, FRME* e, AR* ar){
484     MC *mc = malloc(sizeof(MC));
485     mc->insn = malloc(sizeof(INSN_BUF));
486     if(tac->rtn.type == CONSTANT){
487       sprintf(mc->insn,"li $v1 %d",tac->rtn.v->value);
488     }
489     else if(tac->rtn.type == IDENTIFIER){
490       TOKEN *t = lookup_loc(tac->rtn.v,e);
491       if(t == NULL){
492         t = lookup_from_memory(tac->rtn.v,e,ar);
493       }
494       if(t == NULL){
495         sprintf(mc->insn,"move $v1 $%s",tac->rtn.v->lexeme);
496       }
497       else{
498         sprintf(mc->insn,"move $v1 $%s",t->lexeme);
499       }
500     }
501     call_stack -= e->size;
502     MC* last = find_lst(mc);
503     last->next =  malloc(sizeof(MC));
504     last->next->insn = malloc(sizeof(INSN_BUF));
505     sprintf(last->next->insn,"addiu $sp, $sp %d",ar->size);
506     last = find_lst(last);
507     last->next =  malloc(sizeof(MC));
508     last->next->insn = malloc(sizeof(INSN_BUF));
509     sprintf(last->next->insn,"jr $ra");
510     return mc;
511 }
512
513 MC* new_prc(TAC* tac, FRME* e){
514   MC *mc = malloc(sizeof(MC));
515   mc->insn = malloc(sizeof(INSN_BUF));
516   if(strcmp(tac->proc.name->lexeme,"main")==0){
517     sprintf(mc->insn,"_%s:",tac->proc.name->lexeme);
518   }
519   else{
520     sprintf(mc->insn,"%s:",tac->proc.name->lexeme);
521   }
522   return mc;
```

```
523 }
524
525 MC* load_args(TAC* tac, FRME* e){
526   MC *mc = malloc(sizeof(MC));
527   mc->insn = malloc(sizeof(INSN_BUF));
528   MC* first = mc;
529   TOKENLIST* vars = tac->proc.args;
530   TOKEN* t;
531   int i = 0;
532   while(i < tac->proc.arity && vars != NULL){
533     mc->next = malloc(sizeof(MC));
534     mc->next->insn = malloc(sizeof(INSN_BUF));
535     t = lookup_loc(vars->name,e);
536     sprintf(mc->next->insn,"move $%s $a%d",t->lexeme,i);
537     mc = find_lst(mc);
538     vars = vars->next;
539     i++;
540   }
541   return first;
542 }
543
544 MC* new_cll(TAC* tac, FRME* e, AR* ar){
545   MC* mc = malloc(sizeof(MC));
546   mc->insn = malloc(sizeof(INSN_BUF));
547   MC* last = find_lst(mc);
548   int i=0;
549   TOKENLIST* args = tac->call.args;
550   while(i< tac->call.arity && args != NULL){
551     TOKEN* t = new_token(IDENTIFIER);
552     t->lexeme = (char*)calloc(1,2);
553     sprintf(t->lexeme,"a%d",i);
554     if(tac->call.args->name->type == IDENTIFIER){
555       last->next = new_smpl_ld(e,lookup_loc(args->name,e),t);
556     }
557     else{
558        last->next = new_smpl_ld(e,args->name,t);
559     }
560     last = find_lst(last);
561     args = args->next;
562     i++;
563   }
564   last = find_lst(last);
565   last->next = malloc(sizeof(MC));
566   last->next->insn = malloc(sizeof(INSN_BUF));
567   sprintf(last->next->insn,"jal %s",tac->call.name->lexeme);
568   return mc;
569 }
570
571 MC* gen_mc0(TAC* i, FRME* e, AR* curr){
572   MC* mc, *last;
573   CLSURE* f;
574   TOKEN* name;
575   if (i==NULL || i->op == tac_endproc)
576   {delete_constants(e); mc = new_func_rtn(i); return mc;}
577
578   switch (i->op) {
579   default:
580     printf("unknown type code %d (%p) in mmc_mcg\n",i->op,i);
581
582     case tac_plus:
```

```
583       mc = new_plus(i);
584       mc->next = gen_mc0(i->next,e,curr);
585       return mc;
586     case tac_minus:
587       mc = new_minus(i);
588       mc->next = gen_mc0(i->next,e,curr);
589       return mc;
590     case tac_div:
591       mc = new_div(i);
592       last = find_lst(mc);
593       last->next = gen_mc0(i->next,e,curr);
594       return mc;
595     case tac_mod:
596       mc = new_mod(i);
597       last = find_lst(mc);
598       last->next = gen_mc0(i->next,e,curr);
599       return mc;
600     case tac_mult:
601       mc = new_mult(i);
602       last = find_lst(mc);
603       last->next = gen_mc0(i->next,e,curr);
604       return mc;
605     case tac_innerproc:
606       name = i->proc.name;
607       f = find_fnc(name,e);
608       if (f == NULL){
609         f = malloc(sizeof(CLSURE));
610         f->env = e;
611         f->code = i;
612         declare_fnc(i->proc.name,f,e);
613         i = find_endproc(i->next);
614         mc = gen_mc0(i->next,e,curr);
615         return mc;
616       }
617     case tac_proc:
618       curr = calculate_frame(curr,i);
619       e->size = curr->size;
620       mc = new_prc(i,e);
621       last = find_lst(mc);
622       last->next = gen_frame(curr);
623       last = find_lst(last);
624       last->next = load_args(i,e);
625       last = find_lst(last);
626       last->next = gen_mc0(i->next,e,curr);
627       return mc;
628     case tac_load:
629       mc = new_ld(e,i,curr);
630       mc->next = gen_mc0(i->next,e,curr);
631       return mc;
632     case tac_store:
633       mc = new_str(i,e);
634       last = find_lst(mc);
635       last->next = gen_mc0(i->next,e,curr);
636       return mc;
637     case tac_if:
638       mc = new_ift(i,e);
639       last = find_lst(mc);
640       last->next = gen_mc0(i->next,e,curr);
641       return mc;
642     case tac_lbl:
```

```
643       mc = new_lbli(i);
644       last = find_lst(mc);
645       last->next = gen_mc0(i->next,e,curr);
646       return mc;
647     case tac_goto:
648       mc = new_gtl(i);
649       last = find_lst(mc);
650       last->next = gen_mc0(i->next,e,curr);
651       return mc;
652     case tac_call:
653       mc = save_frame(curr,e);
654       last = find_lst(mc);
655       last->next = new_cll(i,e,curr);
656       last = find_lst(last);
657       last->next = restore_frame(curr,e);
658       last = find_lst(last);
659       last->next = gen_mc0(i->next,e,curr);
660       last = find_lst(last);
661       last->next = call_func(i->call.name,i,e,curr);
662
663       return mc;
664     case tac_rtn:
665       mc = new_rtn(i,e,curr);
666       last = find_lst(mc);
667       last->next = gen_mc0(i->next,e,curr);
668       return mc;
669   }
670 }
671
672 MC* print_result() {
673
674     //print integer result
675     MC *mc = malloc(sizeof(MC));
676     mc->insn = malloc(sizeof(INSN_BUF));
677     mc->insn = "#print integer result";
678
679     MC* last = find_lst(mc);
680     last->next = malloc(sizeof(MC));
681     last->next->insn = malloc(sizeof(INSN_BUF));
682     last->next->insn = "move $a0 $v1";
683     last = find_lst(last);
684
685     last->next = make_syscall(PRINT_INT);
686
687     //print newline
688     last = find_lst(last);
689     last->next = malloc(sizeof(MC));
690     last->next->insn = malloc(sizeof(INSN_BUF));
691     last->next->insn = "li $a0 10";
692
693     last = find_lst(last);
694     last->next = make_syscall(PRINT_CHAR);
695
696     //exit
697     last = find_lst(last);
698     last->next = make_syscall(EXIT);
699     return mc;
700 }
701
702
```

```
703 MC* gen_mc(TAC* tac){
704   FRME* e = malloc(sizeof(FRME));
705   AR* global = malloc(sizeof(AR));
706   MC* mc = init_mc();
707   MC* last = find_lst(mc);
708   last->next = gen_globframe(tac,e,global);
709   last = find_lst(last);
710   FRME *ef = e;
711     while(ef != NULL){
712         BNDING* bindings = e->bindings;
713         while (bindings != NULL){
714             if(strcmp(bindings->name->lexeme,"main")==0){
715                 last->next = malloc(sizeof(MC));
716                 last->next->insn = malloc(sizeof(INSN_BUF));
717                 last->next->insn = "jal _main";
718                 last = find_lst(last);
719                 last->next = print_result();
720                 last = find_lst(last);
721                 last->next = call_func(bindings->name,NULL,e,global);
722                 return mc;
723             }
724             bindings = bindings->next;
725         }
726         ef = e->next;
727     }
728 }
```