Homework 7

郭天魁
信息科学技术学院
1300012790

October 28, 2014

1 Homework 7

1.1 4.43

A. 这段代码将push执行前%esp−4的值，且会修改CFLAGS。

B.

```
movl REG,-4(%esp)
leal -4(%esp),%esp
```

1.2 4.45

A.

```
void bubble_p(int *data, int count) {
        int i, last;
        int *p1, *p2;
        for (last = count-1; last > 0; last--) {
                for (i = 0; i < last; i++) {
                        p1 = data + i;
                        p2 = p1 + 1;
                        if (*p2 < *p1) {
                                /* Swap adjacent elements */
                                int t = *p2;
                                *p2 = *p1;
                                *p1 = t;
                        }
                }
        }
}

int a[] = {4,3,9,1,0};

```

```
20 int main() {
21         int i;
22         bubble_p(a, 5);
23         for(i = 0; i < 4; i++)
24                 if(a[i] > a[i+1])
25                         return 1;
26         return 0;
27 }
```

B.

```
1          .pos 0
2  init:   irmovl Stack, %esp
3          irmovl Stack, %ebp
4          call Main
5          halt
6
7  bubble_p:
8          pushl   %edi
9          pushl   %esi
10         pushl   %ebx
11         mrmovl  20(%esp), %edi
12         irmovl  $1, %eax
13         rrmovl  %edi, %esi
14         subl    %eax, %esi
15         mrmovl  16(%esp), %eax
16         andl    %esi, %esi
17         jle     L1
18         rrmovl  %eax, %edi
19         rrmovl  %esi, %ebx
20         addl    %ebx, %ebx
21         addl    %ebx, %ebx
22         addl    %eax, %ebx
23         jmp     L3
24 L7:
25         mrmovl  4(%eax), %edx
26         mrmovl  (%eax), %ecx
27         pushl   %edx
28         subl    %ecx, %edx
29         popl    %edx
30         jge     L4
31         rmmovl  %ecx, 4(%eax)
32         rmmovl  %edx, (%eax)
33 L4:
34         pushl   %ebx
35         irmovl  $4, %ebx
36         addl    %ebx, %eax
```

```
37          popl    %ebx
38          pushl   %eax
39          subl    %ebx, %eax
40          popl    %eax
41          jne     L7
42  L6:
43          pushl   %eax
44          irmovl  $4, %eax
45          subl    %eax, %ebx
46          irmovl  $1, %eax
47          subl    %eax, %esi
48          popl    %eax
49          je      L1
50  L3:
51          andl    %esi, %esi
52          jle     L6
53          rrmovl  %edi, %eax
54          jmp     L7
55  L1:
56          popl    %ebx
57          popl    %esi
58          popl    %edi
59          ret
60  Main:
61          irmovl  $8, %eax
62          subl    %eax, %esp
63          irmovl  $5, %eax
64          rmmovl  %eax, 4(%esp)
65          irmovl  $a, %eax
66          rmmovl  %eax, (%esp)
67          call    bubble_p
68          irmovl  $8, %eax
69          addl    %eax, %esp
70          xorl    %eax, %eax
71          ret
72
73          .align 4
74  a:
75          .long   4
76          .long   3
77          .long   9
78          .long   1
79          .long   0
80
81          .pos 0x1000
82  Stack:
```

输出为：

```
Stopped in 235 steps at PC = 0x11.  Status 'HLT', CC Z=1 S=0 O=0

Changes to registers:
%ecx:   0x00000000      0x00000001
%esp:   0x00000000      0x00001000
%ebp:   0x00000000      0x00001000

Changes to memory:
0x00dc: 0x00000004      0x00000000
0x00e0: 0x00000003      0x00000001
0x00e4: 0x00000009      0x00000003
0x00e8: 0x00000001      0x00000004
0x00ec: 0x00000000      0x00000009
0x0fe0: 0x00000000      0x000000e0
0x0ff0: 0x00000000      0x000000d1
0x0ff4: 0x00000000      0x000000dc
0x0ff8: 0x00000000      0x00000005
0x0ffc: 0x00000000      0x00000011
```

可以看出内存中的值已被排序。

1.3 4.46

```
        .pos 0
init:
        irmovl Stack, %esp
        irmovl Stack, %ebp
        call Main
        halt

bubble_p:
        pushl   %edi
        pushl   %esi
        pushl   %ebx
        mrmovl  20(%esp), %edi
        irmovl  $1, %eax
        rrmovl  %edi, %esi
        subl    %eax, %esi
        mrmovl  16(%esp), %eax
        andl    %esi, %esi
        jle     L1
        rrmovl  %eax, %edi
        rrmovl  %esi, %ebx
        addl    %ebx, %ebx
```

```
22          addl    %ebx, %ebx
23          addl    %eax, %ebx
24          jmp     L3
25  L7:
26          mrmovl  4(%eax), %edx
27          mrmovl  (%eax), %ecx
28          pushl   %ebx
29          pushl   %edx
30          subl    %ecx, %edx
31          popl    %edx
32          cmovl   %ecx, %ebx
33          cmovl   %edx, %ecx
34          cmovl   %ebx, %edx
35          rmmovl  %edx, 4(%eax)
36          rmmovl  %ecx, (%eax)
37          irmovl  $4, %ebx
38          addl    %ebx, %eax
39          popl    %ebx
40          pushl   %eax
41          subl    %ebx, %eax
42          popl    %eax
43          jne     L7
44  L6:
45          pushl   %eax
46          irmovl  $4, %eax
47          subl    %eax, %ebx
48          irmovl  $1, %eax
49          subl    %eax, %esi
50          popl    %eax
51          je      L1
52  L3:
53          andl    %esi, %esi
54          jle     L6
55          rrmovl  %edi, %eax
56          jmp     L7
57  L1:
58          popl    %ebx
59          popl    %esi
60          popl    %edi
61          ret
62  Main:
63          irmovl  $8, %eax
64          subl    %eax, %esp
65          irmovl  $5, %eax
66          rmmovl  %eax, 4(%esp)
67          irmovl  $a, %eax
```

```
68          rmmovl  %eax, (%esp)
69          call    bubble_p
70          irmovl  $8, %eax
71          addl    %eax, %esp
72          xorl    %eax, %eax
73          ret
74
75          .align 4
76  a:
77          .long   4
78          .long   3
79          .long   9
80          .long   1
81          .long   0
82
83          .pos 0x1000
84  Stack:
```

1.4 4.49

在实验材料中的适当位置添加IIADDL即可。

```
1   #/* $begin seq-all-hcl */
2
3   ####################################################################
4
5   #  HCL Description of Control for Single Cycle Y86 Processor SEQ   #
6
7   #  Copyright (C) Randal E. Bryant, David R. O'Hallaron, 2010       #
8
9   ####################################################################
10
11
12
13  ## Your task is to implement the iaddl and leave instructions
14
15  ## The file contains a declaration of the icodes
16
17  ## for iaddl (IIADDL) and leave (ILEAVE).
18
19  ## Your job is to add the rest of the logic to make it work
20
21
22
23  ####################################################################
24
25  #    C Include's.  Don't alter these                               #
26
```

```
27 ##################################################################
28
29
30
31 quote '#include <stdio.h>'
32
33 quote '#include "isa.h"'
34
35 quote '#include "sim.h"'
36
37 quote 'int sim_main(int argc, char *argv[]);'
38
39 quote 'int gen_pc(){return 0;}'
40
41 quote 'int main(int argc, char *argv[])'
42
43 quote '  {plusmode=0;return sim_main(argc,argv);}'
44
45
46
47 ##################################################################
48
49 #    Declarations.  Do not change/remove/delete any of these       #
50
51 ##################################################################
52
53
54
55 ##### Symbolic representation of Y86 Instruction Codes #############
56
57 intsig INOP     'I_NOP'
58
59 intsig IHALT    'I_HALT'
60
61 intsig IRRMOVL  'I_RRMOVL'
62
63 intsig IIRMOVL  'I_IRMOVL'
64
65 intsig IRMMOVL  'I_RMMOVL'
66
67 intsig IMRMOVL  'I_MRMOVL'
68
69 intsig IOPL     'I_ALU'
70
71 intsig IJXX     'I_JMP'
72
```

```
73 intsig ICALL    'I_CALL'
74
75 intsig IRET     'I_RET'
76
77 intsig IPUSHL   'I_PUSHL'
78
79 intsig IPOPL    'I_POPL'
80
81 # Instruction code for iaddl instruction
82
83 intsig IIADDL   'I_IADDL'
84
85 # Instruction code for leave instruction
86
87 intsig ILEAVE   'I_LEAVE'
88
89
90
91 ##### Symbolic represenations of Y86 function codes
       #####
92
93 intsig FNONE    'F_NONE'        # Default function code
94
95
96
97 ##### Symbolic representation of Y86 Registers referenced explicitly
       #####
98
99 intsig RESP     'REG_ESP'       # Stack Pointer
100
101 intsig REBP     'REG_EBP'       # Frame Pointer
102
103 intsig RNONE    'REG_NONE'      # Special value indicating "no register
       "
104
105
106
107 ##### ALU Functions referenced explicitly
       #####
108
109 intsig ALUADD   'A_ADD'         # ALU should add its arguments
110
111
112
113 ##### Possible instruction status values
       #####
```

```
114
115 intsig SAOK     'STAT_AOK'              # Normal execution
116
117 intsig SADR     'STAT_ADR'      # Invalid memory address
118
119 intsig SINS     'STAT_INS'      # Invalid instruction
120
121 intsig SHLT     'STAT_HLT'      # Halt instruction encountered
122
123
124
125 ##### Signals that can be referenced by control logic
       ####################
126
127
128
129 ##### Fetch stage inputs                ##### 
130
131 intsig pc 'pc'                          # Program counter
132
133 ##### Fetch stage computations          ##### 
134
135 intsig imem_icode 'imem_icode'          # icode field from instruction
       memory
136
137 intsig imem_ifun  'imem_ifun'           # ifun field from instruction
       memory
138
139 intsig icode      'icode'               # Instruction control code
140
141 intsig ifun       'ifun'                # Instruction function
142
143 intsig rA         'ra'                  # rA field from instruction
144
145 intsig rB         'rb'                  # rB field from instruction
146
147 intsig valC       'valc'                # Constant from instruction
148
149 intsig valP       'valp'                # Address of following
       instruction
150
151 boolsig imem_error 'imem_error'         # Error signal from instruction
        memory
152
153 boolsig instr_valid 'instr_valid'       # Is fetched instruction valid?
154
```

```
155
156
157 ##### Decode stage computations         #####
158
159 intsig valA     'vala'                  # Value from register A port
160
161 intsig valB     'valb'                  # Value from register B port
162
163
164
165 ##### Execute stage computations        #####
166
167 intsig valE     'vale'                  # Value computed by ALU
168
169 boolsig Cnd     'cond'                  # Branch test
170
171
172
173 ##### Memory stage computations         #####
174
175 intsig valM     'valm'                  # Value read from memory
176
177 boolsig dmem_error 'dmem_error'         # Error signal from data memory
178
179
180
181
182
183 ########################################################################
184
185 #    Control Signal Definitions.                                       #
186
187 ########################################################################
188
189
190
191 ################ Fetch Stage     ###################################
192
193
194
195 # Determine instruction code
196
197 int icode = [
198
199         imem_error: INOP;
200
```

```
201         1: imem_icode;          # Default: get from instruction memory
202
203 ];
204
205
206
207 # Determine instruction function
208
209 int ifun = [
210
211         imem_error: FNONE;
212
213         1: imem_ifun;           # Default: get from instruction memory
214
215 ];
216
217
218
219 bool instr_valid = icode in
220
221         { INOP, IHALT, IRRMOVL, IIRMOVL, IRMMOVL, IMRMOVL,
222
223                 IOPL, IJXX, ICALL, IRET, IPUSHL, IPOPL, IIADDL };
224
225
226
227 # Does fetched instruction require a regid byte?
228
229 bool need_regids =
230
231         icode in { IRRMOVL, IOPL, IPUSHL, IPOPL,
232
233                         IIRMOVL, IRMMOVL, IMRMOVL, IIADDL };
234
235
236
237 # Does fetched instruction require a constant word?
238
239 bool need_valC =
240
241         icode in { IIRMOVL, IRMMOVL, IMRMOVL, IJXX, ICALL, IIADDL };
242
243
244
245 ################ Decode Stage    ###################################
246
```

```
247
248
249 ## What register should be used as the A source?
250
251 int srcA = [
252
253         icode in { IRRMOVL, IRMMOVL, IOPL, IPUSHL  } : rA;
254
255         icode in { IPOPL, IRET } : RESP;
256
257         1 : RNONE; # Don't need register
258
259 ];
260
261
262
263 ## What register should be used as the B source?
264
265 int srcB = [
266
267         icode in { IOPL, IRMMOVL, IMRMOVL, IIADDL  } : rB;
268
269         icode in { IPUSHL, IPOPL, ICALL, IRET } : RESP;
270
271         1 : RNONE;  # Don't need register
272
273 ];
274
275
276
277 ## What register should be used as the E destination?
278
279 int dstE = [
280
281         icode in { IRRMOVL } && Cnd : rB;
282
283         icode in { IIRMOVL, IOPL, IIADDL } : rB;
284
285         icode in { IPUSHL, IPOPL, ICALL, IRET } : RESP;
286
287         1 : RNONE;  # Don't write any register
288
289 ];
290
291
292
```

```
293 ## What register should be used as the M destination?
294
295 int dstM = [
296
297         icode in { IMRMOVL, IPOPL } : rA;
298
299         1 : RNONE;  # Don't write any register
300
301 ];
302
303
304
305 ################ Execute Stage   ###################################
306
307
308
309 ## Select input A to ALU
310
311 int aluA = [
312
313         icode in { IRRMOVL, IOPL } : valA;
314
315         icode in { IIRMOVL, IRMMOVL, IMRMOVL, IIADDL } : valC;
316
317         icode in { ICALL, IPUSHL } : -4;
318
319         icode in { IRET, IPOPL } : 4;
320
321         # Other instructions don't need ALU
322
323 ];
324
325
326
327 ## Select input B to ALU
328
329 int aluB = [
330
331         icode in { IRMMOVL, IMRMOVL, IOPL, ICALL,
332
333                         IPUSHL, IRET, IPOPL, IIADDL } : valB;
334
335         icode in { IRRMOVL, IIRMOVL } : 0;
336
337         # Other instructions don't need ALU
338
```

```
339 ];
340
341
342
343 ## Set the ALU function
344
345 int alufun = [
346
347         icode == IOPL : ifun;
348
349         1 : ALUADD;
350
351 ];
352
353
354
355 ## Should the condition codes be updated?
356
357 bool set_cc = icode in { IOPL, IIADDL };
358
359
360
361 ################ Memory Stage    ###################################
362
363
364
365 ## Set read control signal
366
367 bool mem_read = icode in { IMRMOVL, IPOPL, IRET };
368
369
370
371 ## Set write control signal
372
373 bool mem_write = icode in { IRMMOVL, IPUSHL, ICALL };
374
375
376
377 ## Select memory address
378
379 int mem_addr = [
380
381         icode in { IRMMOVL, IPUSHL, ICALL, IMRMOVL } : valE;
382
383         icode in { IPOPL, IRET } : valA;
384
```

```
385         # Other instructions don't need address
386
387 ];
388
389
390
391 ## Select memory input data
392
393 int mem_data = [
394
395         # Value from register
396
397         icode in { IRMMOVL, IPUSHL } : valA;
398
399         # Return PC
400
401         icode == ICALL : valP;
402
403         # Default: Don't write anything
404
405 ];
406
407
408
409 ## Determine instruction status
410
411 int Stat = [
412
413         imem_error || dmem_error : SADR;
414
415         !instr_valid: SINS;
416
417         icode == IHALT : SHLT;
418
419         1 : SAOK;
420
421 ];
422
423
424
425 ################ Program Counter Update ############################
426
427
428
429 ## What address should instruction be fetched at
430
```

```
431
432
433 int new_pc = [
434
435         # Call.  Use instruction constant
436
437         icode == ICALL : valC;
438
439         # Taken branch.  Use instruction constant
440
441         icode == IJXX && Cnd : valC;
442
443         # Completion of RET instruction.  Use value from stack
444
445         icode == IRET : valM;
446
447         # Default: Use incremented PC
448
449         1 : valP;
450
451 ];
452
453 #/* $end seq-all-hcl */
```