Project2: Optimizing the Performance of a Pipelined Processor

CS 359, COMPUTER ARCHITECTURE, Yanyan Shen, Spring 2017

1 Preknowledge

I have read the CSAPP book and concluded several basic points which are essential to solving the problems in Part A, B, and C.

- There are 8 registers in the Y86 system

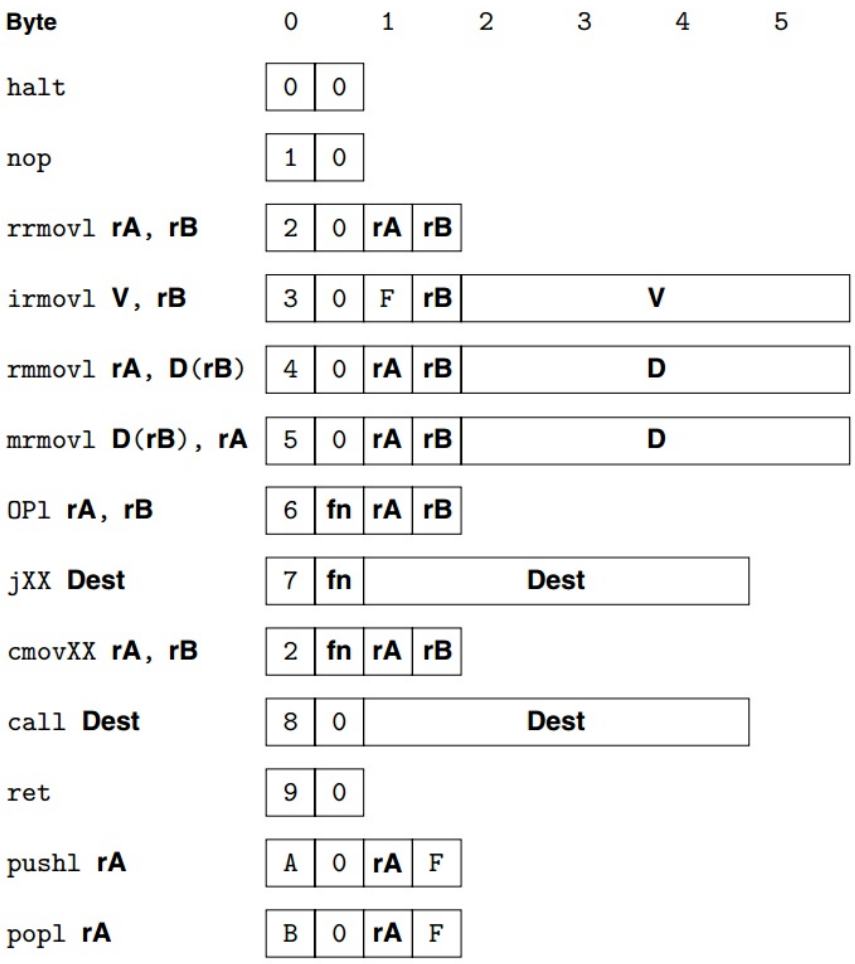
  $$\%eax, \%ecx, \%edx, \%ebx$$

  $$\%esi, \%edi, \%esp, \%ebp$$

  Each of these registers stores a word. Among then register $\%esp$ is used as a stack pointer by the push, pop, call, and return instructions.

- There are three single-bit condition codes, ZF, SF, and OF, storing information about the effect of the most recent arithmetic or logical instruction.

- The Y86 instruction set is largely a subset of the IA32 instruction set but it still have some differences. The picture below shows the Y86 instruction set.



- In the Y86 system, the stack starts at a certain address and grows toward lower addresses, which prevents space conflict.

2 Part A

2.1 Description

- The program **sum.ys** was used to sum linked list elements iteratively. We are supposed to add the sum of a list from head to tail using the Y86 codeing rules.
- The program **rsum.ys** is similar to the **sum.ys**, except it sums linked list elements recursively . We are supposed to add the sum of a list from head to tail using the Y86 codeing rules.
- The program **copy.ys** is used for two purposes. First it copies a block of words from one part of memory to another area of memory. Second it computes the checksum (Xor) of all the words copied.

2.2 Solution

sum.ys

```
1                       irmovl Stack,%esp
2                       rrmovl %esp,%ebp
3                       irmovl ele1, %edx
4                       pushl %edx
5                       call sum_list
6                       halt
7
8                       .align 4
9                       ele1:
10                      .long 0x00a
11                      .long ele2
12                      ele2:
13                      .long 0x0b0
14                      .long ele3
15                      ele3:
16                      .long 0xc00
17                      .long 0
18
19                      sum_list:
20                      pushl %ebp
21                      xorl %eax,%eax
22                      rrmovl %esp,%ebp
23                      mrmovl 8(%ebp),%edx
24                      andl %edx,%edx
25                      je L4
26                      L5:
27                      mrmovl (%edx),%esi
28                      addl %esi,%eax
29                      mrmovl 4(%edx),%edx
30                      andl %edx,%edx
31                      jne L5
32                      L4:
33                      rrmovl %ebp,%esp
34                      popl %ebp
```

```
35                  ret
36
37                  .pos 0x100
38                  Stack:
```

rsum.ys

```
1                   .pos 0
2                   init:
3                   irmovl Stack, %esp
4                   irmovl Stack, %ebp
5                   jmp Main
6
7                   .align 4
8                   ele1:
9                   .long 0x00a
10                  .long ele2
11                  ele2:
12                  .long 0x0b0
13                  .long ele3
14                  ele3:
15                  .long 0xc00
16                  .long 0
17
18                  Main:
19                  irmovl ele1, %edx
20                  pushl %edx
21                  call rsum_list
22                  halt
23
24                  rsum_list:
25                  pushl %ebp
26                  rrmovl %esp,%ebp
27                  mrmovl 0x8(%ebp),%edx
28                  xorl %eax,%eax
29                  pushl %ebx
30                  andl %edx,%edx
31                  je End
32                  irmovl $0xc,%esi
33                  subl %esi,%esp
34                  mrmovl (%edx),%ebx
35                  mrmovl 0x4(%edx),%ecx
36                  pushl %ecx
37                  call rsum_list
38                  addl %ebx,%eax
39
40                  End:
41                  mrmovl 0xfffffffc(%ebp),%ebx
42                  rrmovl %ebp, %esp
43                  popl %ebp
44                  ret
```

```
45
46                  .pos    0x100
47                  Stack:
```

copy.ys

```
1                   .pos 0
2                  init:
3                  irmovl Stack, %esp
4                  irmovl Stack, %ebp
5                  jmp Main
6
7                  .align 4
8                  src:
9                  .long 0x00a
10                 .long 0x0b0
11                 .long 0xc00
12                 # Destination block
13                 dest:
14                 .long 0x111
15                 .long 0x222
16                 .long 0x333
17
18                 Main:
19                 irmovl $3,%eax
20                 pushl %eax
21                 irmovl dest,%edx
22                 pushl %edx
23                 irmovl src,%ecx
24                 pushl %ecx
25                 call Copy
26                 halt
27
28                 Copy:
29                 pushl %ebp
30                 rrmovl %esp,%ebp
31                 mrmovl 8(%ebp),%ecx
32                 mrmovl 12(%ebp),%ebx
33                 mrmovl 16(%ebp),%edx
34                 irmovl $0,%eax
35                 andl %edx,%edx
36                 je End
37
38                 Loop:
39                 mrmovl (%ecx),%esi
40                 rmmovl %esi,(%ebx)
41                 xorl %esi,%eax
42                 irmovl $4,%edi
43                 addl %edi,%ecx
44                 addl %edi,%ebx
45                 irmovl $-1,%edi
```

```
46                  addl %edi,%edx
47                  jne Loop
48
49                  End:
50                  popl %ebp
51                  ret
52
53                  .pos 0x100
54                  Stack:
```

2.3 Analysis

2.3.1 sum.ys

- For the program **sum.ys**, the line 1 $\sim$ 19 and line 37 $\sim$ 38 are the preparatory work and do not need further explanation. For the main part, which is the sum.ys, first we store the $\%ebp$ part to stack, then we set $\%eax$ to zero as the inital sum value, $\%edx$ as the inital list position, then if the list has reached its end, we jump to state L5, which restore the value. If not, we goto the LOOP L5 part.

- In the L5 part, first $\%eax+ = \%esi$ to add to the sum, then $(\%edx) + 4$ to goto the list's next elements. After doing that, again we judge whether the list has reached its end and the condition is exactly the same.

- Based the outcome of this program, $\%eax$ stores the overall sum value, which is $0xcba$, $\%ebp$ get popped and get its inital value, which is the $0x100$, $\%esi$ stores the last elements of the list, which is $\%0xc00$, which proves that our program is correct.

2.3.2 rsum.ys

- For the program **rsum.ys**, the line 1 $\sim$ 17 and line 46 $\sim$ 47 are the preparatory work and do not need further explanation. For the main part, first we let $\%edx$ stores the elel, which is the beginning, then we save $\%edx$ and call $rsum\_list$.

- For the $rsum\_list$ part(line24 $\sim$ 44),which is the recursive version of the sum_list, first we use $\%edx$ to get starting adddress, $\%eax$ to get the inital result which is 0, then we save $\%ebx$ and compare whether $\%edx$ is zero or not. If so, goto end, else, goto the next part, which is the recursive part.

- For the recursive part, the mian idea is to push $\%edx \rightarrow next$ as the first parameter and then call $rsum\_list$ by recursion.

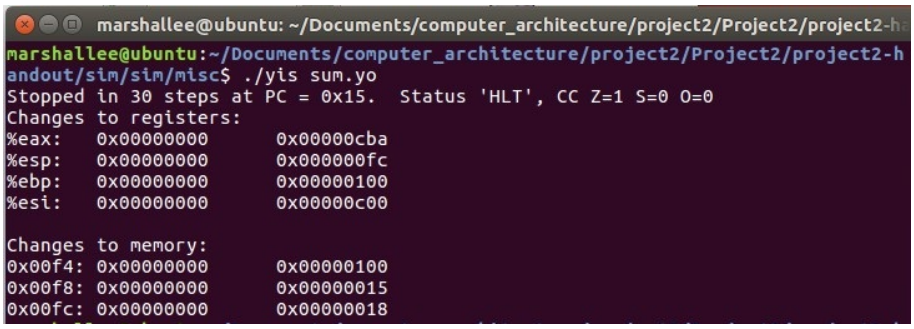
- Based the outcome of this program, $\%ebp$ pop out and its inital value is $0x100$, $\%eax$ stores the overall sum value $0xcba$, which proves that our program is correct.

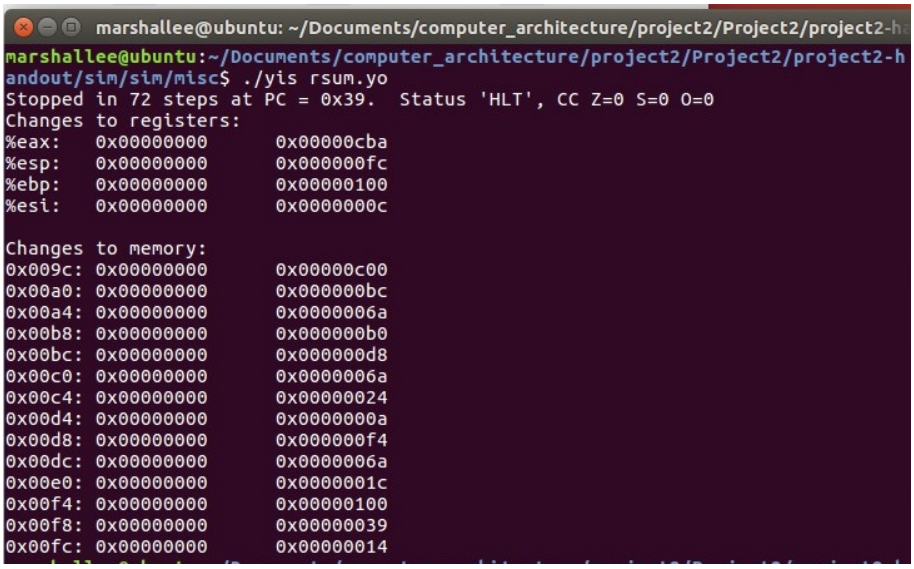
2.3.3 copy.ys

- For the program **copy.ys**, the line 1 $\sim$ 17 and line 53 $\sim$ 54 are the preparatory work and do not need further explanation. For the main part(line 18 $\sim$ 26), first we change $\%eax$ to 3, $\%edx$ to the $dest$, $\%ecx$ to the src and store the value to the stack, then we call Copy function and halt.

- For the Copy function(line $28 \sim 47$), first we let $\%ebp$ store $\%esp$, $\%ecx$ store src, $\%ebx$ store $dest$, $\%edx$ store the length, $\%eax$ stores the inital value of the result which is 0, then we compare whether the copy function has reached its end, is so, goto the End part, else, goto the Loop part.
- For the Loop part, first we get the address of the scr, then we begin the copy process, while we add the src and $dest$ by 4 to move to the next address, in the meantime we let $len - 1$ to serve as flag to decide when to stop.
- Based the outcome of this program, the address $0x0020$ value is changed from $0x111$ to $0xa$, the address $0x0024$ value is changed from $0x222$ to $0xb0$, the address $0x0028$ value is changed from $0x333$ to $0xc00$, which proves that our program is correct.

2.4 Outcome



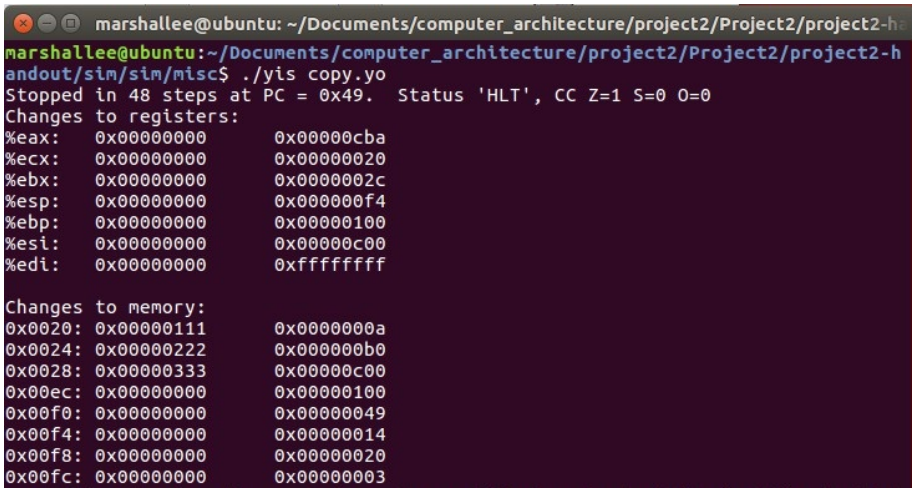


2.5 Review

3 Part B

3.1 Description

In this part, we are asked to add a new instruction *iaddl* to the SEQ processor, this instruction is meant to add a constant to a register.



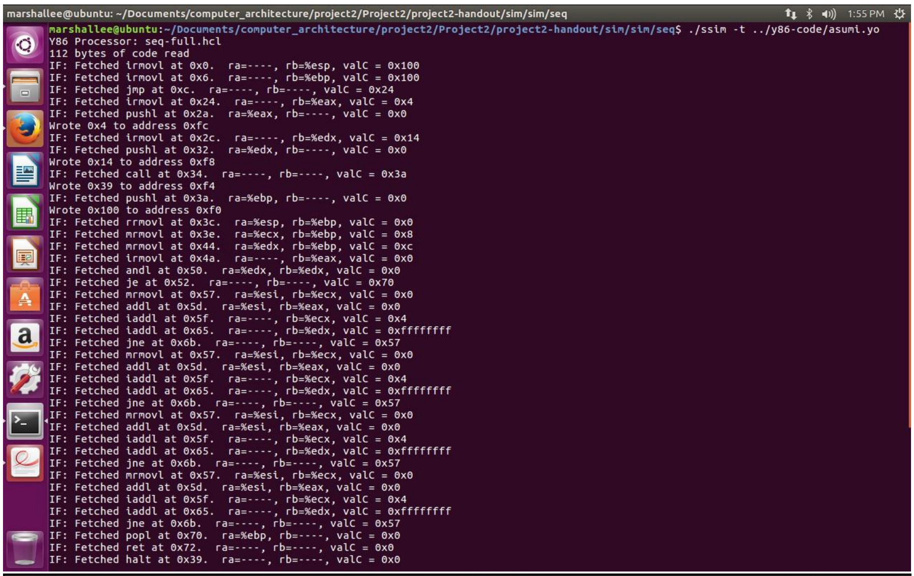
3.2 Solution

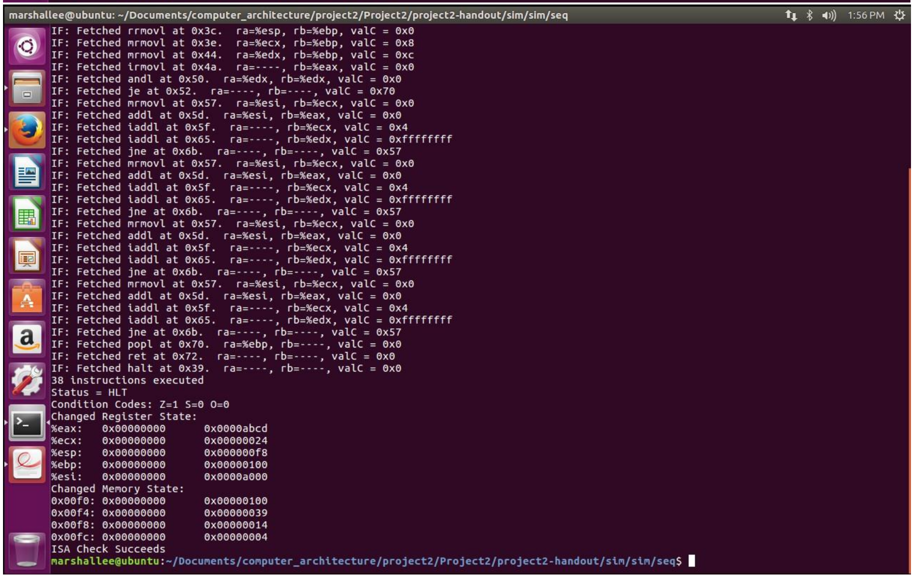
3.3 Analysis

It can be implemented by first using *irmovl* instruction to let the register contains the constant number, then we can use *addl* instruction to add the constant number to the destination register.

3.4 Outcome

3.5 Review


marshallee@ubuntu: ~/Documents/computer_architecture/project2/Project2/project2-handout/sim/sim/seq
1:55 PM
marshallee@ubuntu:~/Documents/computer_architecture/project2/Project2/project2-handout/sim/sim/seq$ ./ssim -t ../y86-code/asumi.yo
Y86 Processor: seq-full.hcl
112 bytes of code read
IF: Fetched irmovl at 0x0. ra=----, rb=%esp, valC = 0x100
IF: Fetched irmovl at 0x6. ra=----, rb=%ebp, valC = 0x100
IF: Fetched jmp at 0xc. ra=----, rb=----, valC = 0x24
IF: Fetched irmovl at 0x24. ra=----, rb=%eax, valC = 0x4
IF: Fetched pushl at 0x2a. ra=%eax, rb=----, valC = 0x0
Wrote 0x4 to address 0xfc
IF: Fetched irmovl at 0x2c. ra=----, rb=%edx, valC = 0x14
IF: Fetched pushl at 0x32. ra=%edx, rb=----, valC = 0x0
Wrote 0x14 to address 0xf8
IF: Fetched call at 0x34. ra=----, rb=----, valC = 0x3a
Wrote 0x39 to address 0xf4
IF: Fetched pushl at 0x3a. ra=%ebp, rb=----, valC = 0x0
Wrote 0x100 to address 0xf0
IF: Fetched rrmovl at 0x3c. ra=%esp, rb=%ebp, valC = 0x0
IF: Fetched mrmovl at 0x3e. ra=%ecx, rb=%ebp, valC = 0x8
IF: Fetched mrmovl at 0x44. ra=%edx, rb=%ebp, valC = 0xc
IF: Fetched irmovl at 0x4a. ra=----, rb=%eax, valC = 0x0
IF: Fetched andl at 0x50. ra=%edx, rb=%edx, valC = 0x0
IF: Fetched je at 0x52. ra=----, rb=----, valC = 0x70
IF: Fetched mrmovl at 0x57. ra=%esi, rb=%ecx, valC = 0x0
IF: Fetched addl at 0x5d. ra=%esi, rb=%eax, valC = 0x0
IF: Fetched iaddl at 0x5f. ra=----, rb=%ecx, valC = 0x4
IF: Fetched iaddl at 0x65. ra=----, rb=%edx, valC = 0xffffffff
IF: Fetched jne at 0x6b. ra=----, rb=----, valC = 0x57
IF: Fetched mrmovl at 0x57. ra=%esi, rb=%ecx, valC = 0x0
IF: Fetched addl at 0x5d. ra=%esi, rb=%eax, valC = 0x0
IF: Fetched iaddl at 0x5f. ra=----, rb=%ecx, valC = 0x4
IF: Fetched iaddl at 0x65. ra=----, rb=%edx, valC = 0xffffffff
IF: Fetched jne at 0x6b. ra=----, rb=----, valC = 0x57
IF: Fetched mrmovl at 0x57. ra=%esi, rb=%ecx, valC = 0x0
IF: Fetched addl at 0x5d. ra=%esi, rb=%eax, valC = 0x0
IF: Fetched iaddl at 0x5f. ra=----, rb=%ecx, valC = 0x4
IF: Fetched iaddl at 0x65. ra=----, rb=%edx, valC = 0xffffffff
IF: Fetched jne at 0x6b. ra=----, rb=----, valC = 0x57
IF: Fetched mrmovl at 0x57. ra=%esi, rb=%ecx, valC = 0x0
IF: Fetched addl at 0x5d. ra=%esi, rb=%eax, valC = 0x0
IF: Fetched iaddl at 0x5f. ra=----, rb=%ecx, valC = 0x4
IF: Fetched iaddl at 0x65. ra=----, rb=%edx, valC = 0xffffffff
IF: Fetched jne at 0x6b. ra=----, rb=----, valC = 0x57
IF: Fetched popl at 0x70. ra=%ebp, rb=----, valC = 0x0
IF: Fetched ret at 0x72. ra=----, rb=----, valC = 0x0
IF: Fetched halt at 0x39. ra=----, rb=----, valC = 0x0


marshallee@ubuntu: ~/Documents/computer_architecture/project2/Project2/project2-handout/sim/sim/seq
1:56 PM
IF: Fetched rrmovl at 0x3c. ra=%esp, rb=%ebp, valC = 0x0
IF: Fetched mrmovl at 0x3e. ra=%ecx, rb=%ebp, valC = 0x8
IF: Fetched mrmovl at 0x44. ra=%edx, rb=%ebp, valC = 0xc
IF: Fetched irmovl at 0x4a. ra=----, rb=%eax, valC = 0x0
IF: Fetched andl at 0x50. ra=%edx, rb=%edx, valC = 0x0
IF: Fetched je at 0x52. ra=----, rb=----, valC = 0x70
IF: Fetched mrmovl at 0x57. ra=%esi, rb=%ecx, valC = 0x0
IF: Fetched addl at 0x5d. ra=%esi, rb=%eax, valC = 0x0
IF: Fetched iaddl at 0x5f. ra=----, rb=%ecx, valC = 0x4
IF: Fetched iaddl at 0x65. ra=----, rb=%edx, valC = 0xffffffff
IF: Fetched jne at 0x6b. ra=----, rb=----, valC = 0x57
IF: Fetched mrmovl at 0x57. ra=%esi, rb=%ecx, valC = 0x0
IF: Fetched addl at 0x5d. ra=%esi, rb=%eax, valC = 0x0
IF: Fetched iaddl at 0x5f. ra=----, rb=%ecx, valC = 0x4
IF: Fetched iaddl at 0x65. ra=----, rb=%edx, valC = 0xffffffff
IF: Fetched jne at 0x6b. ra=----, rb=----, valC = 0x57
IF: Fetched mrmovl at 0x57. ra=%esi, rb=%ecx, valC = 0x0
IF: Fetched addl at 0x5d. ra=%esi, rb=%eax, valC = 0x0
IF: Fetched iaddl at 0x5f. ra=----, rb=%ecx, valC = 0x4
IF: Fetched iaddl at 0x65. ra=----, rb=%edx, valC = 0xffffffff
IF: Fetched jne at 0x6b. ra=----, rb=----, valC = 0x57
IF: Fetched mrmovl at 0x57. ra=%esi, rb=%ecx, valC = 0x0
IF: Fetched addl at 0x5d. ra=%esi, rb=%eax, valC = 0x0
IF: Fetched iaddl at 0x5f. ra=----, rb=%ecx, valC = 0x4
IF: Fetched iaddl at 0x65. ra=----, rb=%edx, valC = 0xffffffff
IF: Fetched jne at 0x6b. ra=----, rb=----, valC = 0x57
IF: Fetched popl at 0x70. ra=%ebp, rb=----, valC = 0x0
IF: Fetched ret at 0x72. ra=----, rb=----, valC = 0x0
IF: Fetched halt at 0x39. ra=----, rb=----, valC = 0x0
38 instructions executed
Status = HLT
Condition Codes: Z=1 S=0 O=0
Changed Register State:
%eax: 0x00000000   0x0000abcd
%ecx: 0x00000000   0x00000024
%esp: 0x00000000   0x000000f8
%ebp: 0x00000000   0x00000100
%esi: 0x00000000   0x0000a000
Changed Memory State:
0x00f0: 0x00000000   0x00000100
0x00f4: 0x00000000   0x00000039
0x00f8: 0x00000000   0x00000014
0x00fc: 0x00000000   0x00000004
ISA Check Succeeds
marshallee@ubuntu:~/Documents/computer_architecture/project2/Project2/project2-handout/sim/sim/seq$