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.

Byte	0	1	2	3	4	5
`halt`	0 0					
`nop`	1 0					
`rrmovl` **rA, rB**	2 0	rA rB				
`irmovl` **V, rB**	3 0	F rB	V			
`rmmovl` **rA, D(rB)**	4 0	rA rB	D			
`mrmovl` **D(rB), rA**	5 0	rA rB	D			
`OPl` **rA, rB**	6 fn	rA rB				
`jXX` **Dest**	7 fn	Dest				
`cmovXX` **rA, rB**	2 fn	rA rB				
`call` **Dest**	8 0	Dest				
`ret`	9 0					
`pushl` **rA**	A 0	rA F				
`popl` **rA**	B 0	rA F				

- 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

```
irmovl Stack,%esp
rrmovl %esp,%ebp
irmovl ele1 , %edx
pushl %edx
call sum_list
halt

.align 4
ele1:
.long 0x00a
.long ele2
ele2:
.long 0x0b0
.long ele3
ele3:
.long 0xc00
.long 0

sum_list:
pushl %ebp
xorl %eax,%eax
rrmovl %esp,%ebp
mrmovl 8(%ebp),%edx
andl %edx,%edx
je L4
L5:
mrmovl (%edx),%esi
addl %esi,%eax
mrmovl 4(%edx),%edx
andl %edx,%edx
jne L5
L4:
rrmovl %ebp,%esp
popl %ebp
```

```
        ret

        .pos 0x100
        Stack:
```

rsum.ys

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

.align 4
ele1:
.long 0x00a
.long ele2
ele2:
.long 0x0b0
.long ele3
ele3:
.long 0xc00
.long 0

Main:
irmovl ele1, %edx
pushl %edx
call rsum_list
halt

rsum_list:
pushl %ebp
rrmovl %esp,%ebp
mrmovl 0x8(%ebp),%edx
xorl %eax,%eax
pushl %ebx
andl %edx,%edx
je End
mrmovl (%edx),%ebx
mrmovl 0x4(%edx),%ecx
pushl %ecx
call rsum_list
addl %ebx,%eax

End:
mrmovl  0xfffffffc(%ebp),%ebx
rrmovl  %ebp, %esp
popl %ebp
ret

.pos    0x100
```

```
Stack:
```

copy.ys

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

   .align 4
   src:
   .long 0x00a
   .long 0x0b0
   .long 0xc00
   # Destination block
   dest:
   .long 0x111
   .long 0x222
   .long 0x333

   Main:
   irmovl $3,%eax
   pushl %eax
   irmovl dest,%edx
   pushl %edx
   irmovl src,%ecx
   pushl %ecx
   call Copy
   halt

   Copy:
   pushl %ebp
   rrmovl %esp,%ebp
   mrmovl 8(%ebp),%ecx
   mrmovl 12(%ebp),%ebx
   mrmovl 16(%ebp),%edx
   irmovl $0,%eax
   andl %edx,%edx
   je End

   Loop:
   mrmovl (%ecx),%esi
   rmmovl %esi,(%ebx)
   xorl %esi,%eax
   irmovl $4,%edi
   addl %edi,%ecx
   addl %edi,%ebx
   irmovl $-1,%edi
   addl %edi,%edx
   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 L4, 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 $\sim$ 45 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 $\sim$ 42),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. It is worth noting that in line 39 the constant number $0xfffffffc$ is -4 and we use this operation to restore $\%ebx$, and in line 25, we push $\%ebp$, in line 41, we pop $\%ebp$, with these two operations we can get the old version of $\%ebp$ which is key to our recursive part.

- 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.

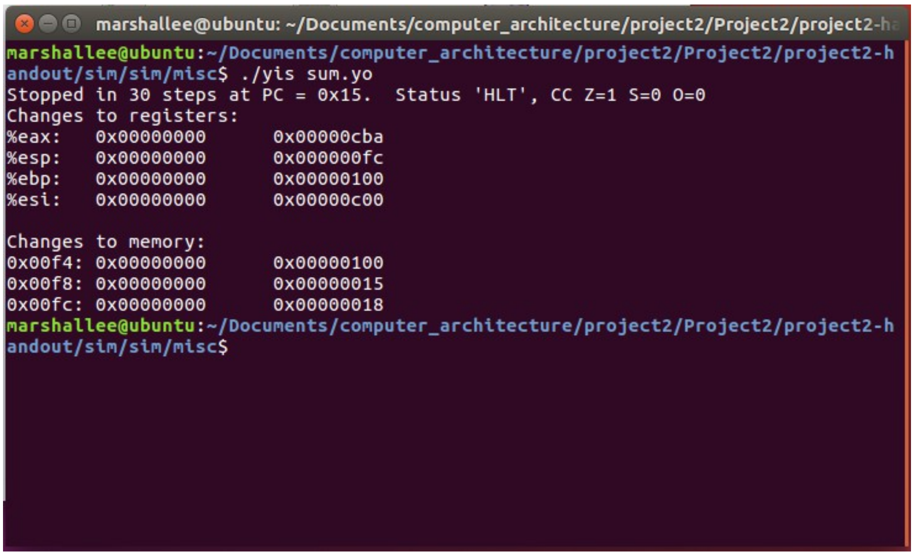
2.3.3 copy.ys

- 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.

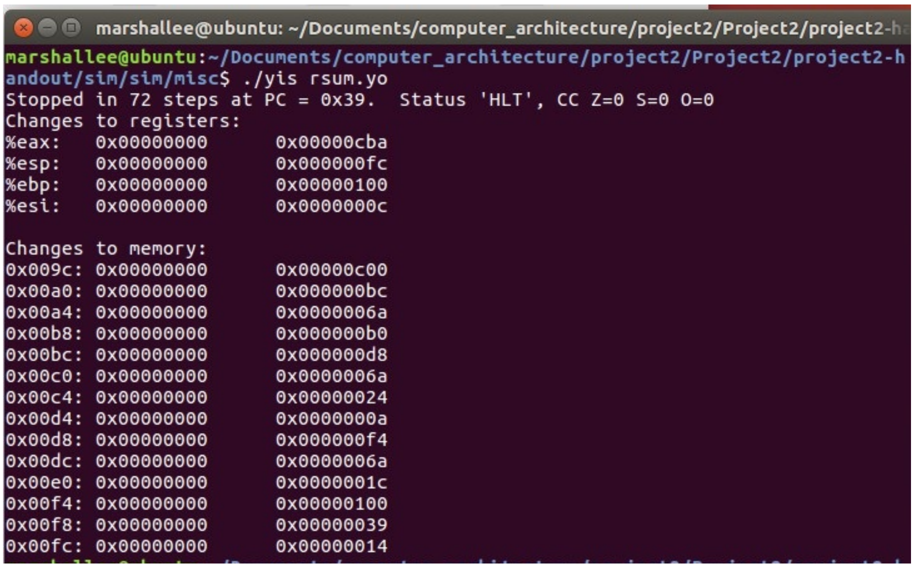
- For the Loop part, first we get the address of the *scr*, then we begin the copy process, while we add the *src* and *dest* by 4 to move to the next address, in the meantime we let $len - 1$ to serve as flag to decide when to stop.
- Based the outcome of this program, register $\%eax$ do the *xorl* instruction with each value and then add to $\%eax$, so the last value of $\%eax$ is $0xcba$, also 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



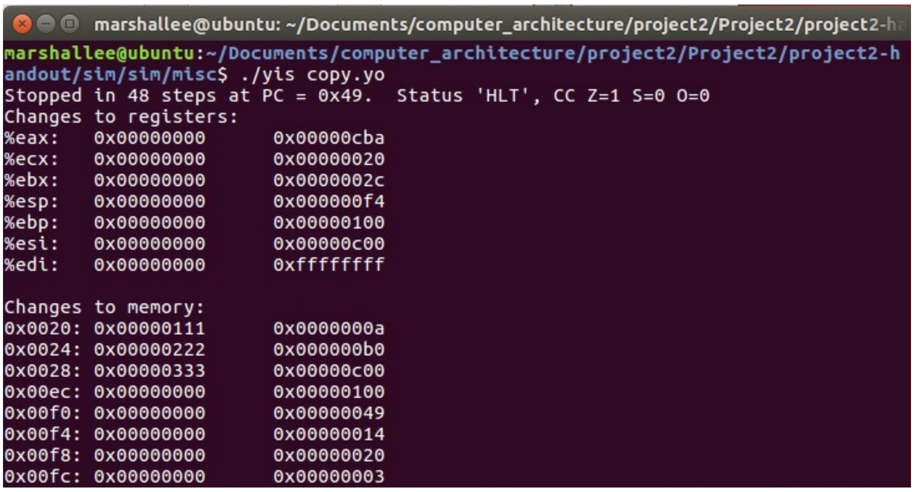
```
marshallee@ubuntu: ~/Documents/computer_architecture/project2/Project2/project2-ha
marshallee@ubuntu:~/Documents/computer_architecture/project2/Project2/project2-h
andout/sim/sim/misc$ ./yis sum.yo
Stopped in 30 steps at PC = 0x15.  Status 'HLT', CC Z=1 S=0 O=0
Changes to registers:
%eax:   0x00000000      0x00000cba
%esp:   0x00000000      0x000000fc
%ebp:   0x00000000      0x00000100
%esi:   0x00000000      0x00000c00

Changes to memory:
0x00f4: 0x00000000      0x00000100
0x00f8: 0x00000000      0x00000015
0x00fc: 0x00000000      0x00000018
marshallee@ubuntu:~/Documents/computer_architecture/project2/Project2/project2-h
andout/sim/sim/misc$
```



```
marshallee@ubuntu: ~/Documents/computer_architecture/project2/Project2/project2-ha
marshallee@ubuntu:~/Documents/computer_architecture/project2/Project2/project2-h
andout/sim/sim/misc$ ./yis rsum.yo
Stopped in 72 steps at PC = 0x39.  Status 'HLT', CC Z=0 S=0 O=0
Changes to registers:
%eax:   0x00000000      0x00000cba
%esp:   0x00000000      0x000000fc
%ebp:   0x00000000      0x00000100
%esi:   0x00000000      0x0000000c

Changes to memory:
0x009c: 0x00000000      0x00000c00
0x00a0: 0x00000000      0x000000bc
0x00a4: 0x00000000      0x0000006a
0x00b8: 0x00000000      0x000000b0
0x00bc: 0x00000000      0x000000d8
0x00c0: 0x00000000      0x0000006a
0x00c4: 0x00000000      0x00000024
0x00d4: 0x00000000      0x0000000a
0x00d8: 0x00000000      0x000000f4
0x00dc: 0x00000000      0x0000006a
0x00e0: 0x00000000      0x0000001c
0x00f4: 0x00000000      0x00000100
0x00f8: 0x00000000      0x00000039
0x00fc: 0x00000000      0x00000014
```



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

Since the code in seq-full.hcl is quite long, I only list the part where we have made changes.

Fetch Stage

```
bool instr_valid = icode in
{ INOP, IHALT, IRRMOVL, IIRMOVL, IRMMOVL, IMRMOVL,
IOPL, IJXX, ICALL, IRET, IPUSHL, IPOPL, IIADDL };

bool need_regids =
icode in { IRRMOVL, IOPL, IPUSHL, IPOPL,
IIRMOVL, IRMMOVL, IMRMOVL, IIADDL };

bool need_valC =
icode in { IIRMOVL, IRMMOVL, IMRMOVL, IJXX, ICALL, IIADDL };
```

Decode Stage

```
int srcB = [
icode in { IOPL, IRMMOVL, IMRMOVL, IIADDL } : rB;
icode in { IPUSHL, IPOPL, ICALL, IRET } : RESP;
1 : RNONE;
];

int dstE = [
icode in { IRRMOVL } && Cnd : rB;
icode in { IIRMOVL, IOPL, IIADDL} : rB;
icode in { IPUSHL, IPOPL, ICALL, IRET } : RESP;
```

```
    1 : RNONE;
    ];
```

Execute Stage

```
int aluA = [
icode in { IRRMOVL, IOPL } : valA;
icode in { IIRMOVL, IRMMOVL, IMRMOVL, IIADDL } : valC;
icode in { ICALL, IPUSHL } : -4;
icode in { IRET, IPOPL } : 4;
];

int aluB = [
icode in { IRMMOVL, IMRMOVL, IOPL, ICALL,
IPUSHL, IRET, IPOPL, IIADDL } : valB;
icode in { IRRMOVL, IIRMOVL } : 0;
];

bool set_cc = icode in { IOPL, IIADDL };
```

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. Since there are roughly four stages in the Y86 instruction set, we will fully discuss this part reagarding each stages.

- For the Fetch Stage, first we should add the $IIADDL$ instruction to the *instr_valid*, then we should add the $IIADDL$ instruction to the *need_regids* set, indicating that we should register to do this operation. Finally since we need a constant for add, we need to add the $IIADDL$ instruction to the *need_valc* set.

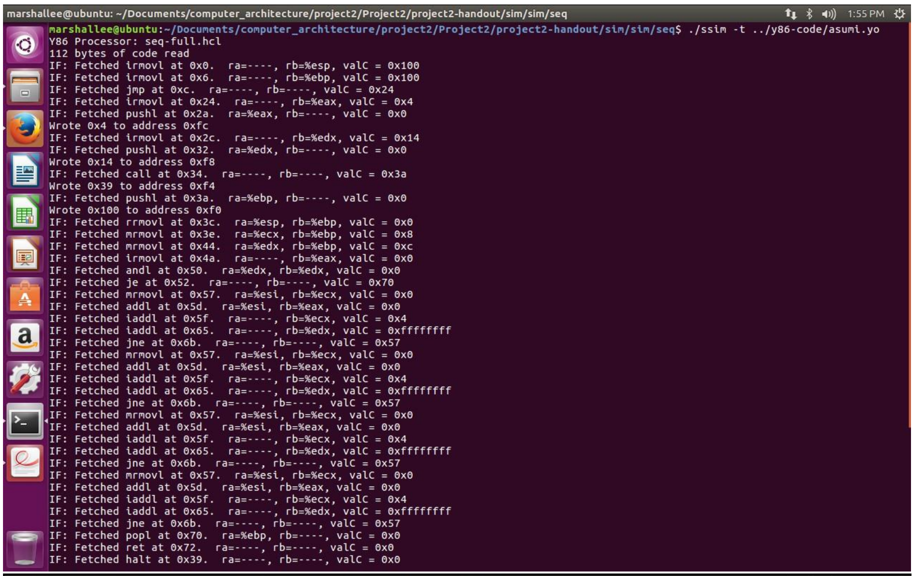
- For the Decode Stage, first we need to add the $IIADDL$ instruction to the $srcB$ set, indicating that we put the register value in this part, then we need to add the $IIADDL$ instruction to the $dstE$ set, indicating that we store the value in the destination E.

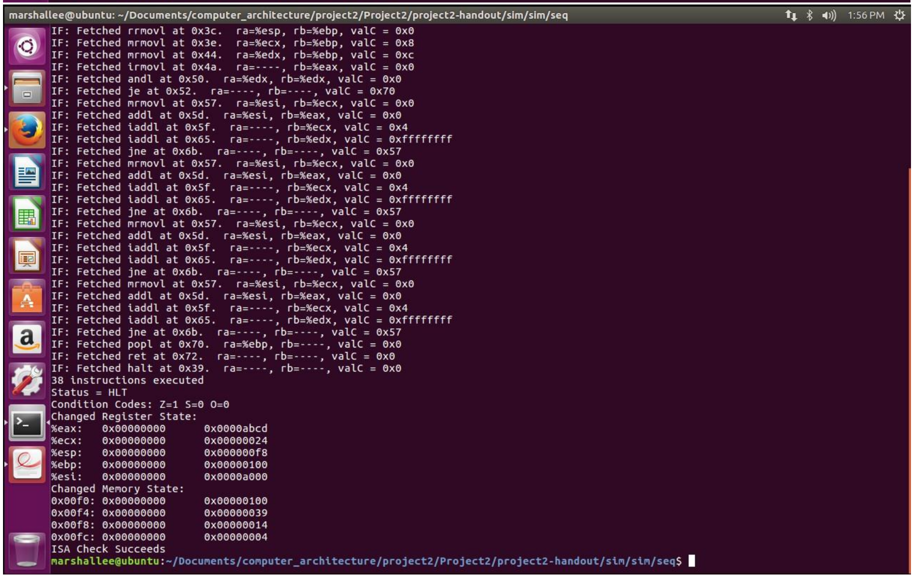
- For the Execute Stage, first we should add the $IIADDL$ instruction to the $aluA$ part, indicating that the ALU operation need the $valC$, which is the constant value, then we should add the $IIADDL$ instruction to the $aluB$ part, indicating that the ALU operation need the $valB$, which is the the destination register's value, finally we should add the $IIADDL$ instruction to the *set_cc* part, indicating that this instruction may lead the flag register to change.

- For the Memory Stage, since this instruction is to add a constant number to a register, so the Memory Stage don't have to change.

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$