COMP.2030 Lab 9 – Key

1. Suppose that MIPS $t0 and $t1 registers are mapped to %rax and %rbx x86-64 registers, respectively. But we only use 32 bits of x86 registers. Convert MIPS instruction on the left to corresponding x86-64 instructions. For labels, use the same labels.

lw $t1, X($zero) movq X, %rbx

li $t0, 1 movq $1, %rax

lw $t1, X($t0) movq X(%rax), %rbx

move $t0, $t1 movq %rbx, %rax

sw $t0, ($t1) movq (%rbx). %rax

addi $t0, $t0, 1 incq $rax

1. that MIPS $t0, $t1 and $t2 registers are mapped to %rax, %rbx and %rcx in x86-64 registers, respectively. But we only use 32 bits of x86 registers. Convert MIPS instruction on the left to corresponding x86-64 instructions. For labels, use the same labels.

lw $t2, X($zero)

li $t0, 1

rept:

bge $t0, $t2, exit

sll $t0, $t0, 2

lw $t1, X($t0)

bgt $t2, $t1, next

move $t2, $t1

sw $t0, ($t1)

next:

sra $t0, $t0, 2

addi $t0, $t0, 1

movq X, %rcx

movq $1, %rax

rept:

cmpq %rcx, %rax

jge exit (%rax>=%rcx)

shlq $2, %rax

movq X(%rax), %rbx

cmpq %rbx, %rcx

jgt next (%rax>%rcx)

movq %rbx, %rcx

movq %rax, (%rbx)

next:

sarq $2, %rax

incq $t0

addq $3, %rax

1. A function fun\_a has the following overall structure:

long fun\_a(unsigned long x){

# *x in %rdi*

fun\_a:

movl $0, %eax

jmp .L5

.L6:

xorq %rdi, %rax

shrq %rdi # *shift right 1*

.L5:

testq %rdi, %rdi

jne .L6

andl $1, %eax

ret

long val = 0;

while ( x ){

val ^= x;

x = x>>1;

}

val &= 0x1;

return val;

}

The gcc C compiler generates the assembly code on the right. Reverse engineer the operation of this code and fill in the missing parts of the C code using the assembly-code version.

1. The gcc C compiler generates the assembly code on the right. Reverse engineer the operation of this code and fill in the missing parts of the C code using the assembly-code version. Fill in spaces

long loop(long x, long n){

# x in %rdi, n in %esi

Loop:

movl %esi, %ecx

movl $1, %edx

movl $0, %eax

jmp .L2

.L3:

movq %rdi, %r8

andq %rdx, %r8

orq %r8, %rax

salq %cl, %rdx

.L2:

testq %rdx, %rdx

jne .L3

ret

long result = 0;

long mask;

for (mask = 1;

mask != 0;

mask = mask << n){

result |= (x&mask);

}

return result;

}