* 1. What does the code hint about the kind of instruction set? (e.g. Accumulator, Register Memory, Memory Memory, Register Register) Please justify your answer.  
       
     Ans.   
       
     1. Register-Memory โดยพิจารณาจากการ instructions `mov DWORD PTR [rbp-4], edi`  
     2. Register-Register โดยพิจารณาจากการ instructions `mov rbp, rsp`
  2. Can you tell whether the architecture is either Restricted Alignment or Unrestricted Alignment? Please explain how you come up with your answer.  
       
       
     จาก assembly code นี้ จะเห็นว่าตัวแปรต่างๆ จะถูกจองในตำแหน่งที่เป็นพหุคูณของ 4 นั่นคือเป็น Restricted Alignment

max1(int, int):

push rbp

mov rbp, rsp

mov DWORD PTR [rbp-4], edi

mov DWORD PTR [rbp-8], esi

mov eax, DWORD PTR [rbp-4]

cmp eax, DWORD PTR [rbp-8]

jle .L2

mov eax, DWORD PTR [rbp-4]

jmp .L4

* 1. Create a new function (e.g. testMax) to call max1. Generate new assembly code. What does the result suggest regarding the register saving (caller save vs. callee save)? Please provide your analysis.  
       
       
       
     จะเห็นว่า esi, edi จะทำการเก็บ argument แล้วนำไปใช้ต่อใน max1 ทำให้มันเป็น caller save  
     แต่ rbp นั้นเป็น callee save เพราะถูกใช้ในเป็น frame pointer  
     ส่วน eax ใช้สำหรับ return value

testMax():

push rbp

mov rbp, rsp

mov esi, 4

mov edi, 3

call max1(int, int)

mov eax, 0

pop rbp

ret

* 1. How do the arguments be passed and the return value returned from a function? Please explain the code.  
       
       
       
       
       
     Ans. Arguments จะถูก pass ผ่าน register เช่น edi, esi, edx, ecx และ function จะทำการ return ผ่าน register eax

testMax():

push rbp

mov rbp, rsp

mov esi, 4

mov edi, 3

call max1(int, int)

mov eax, 0

pop rbp

ret

* 1. Find the part of code (snippet) that does comparison and conditional branch. Explain how it works.  
       
       
       
     จะทำการเปรียบเทียบระหว่างค่าบน eax และค่าบน [rbp-8]  
     หาก eax น้อยกว่าหรือเท่ากับ [rbp-8] จะทำการย้ายไปทำงานที่ .L2  
     แต่ถ้าไม่ ก็จะทำงานต่อไปจนถึง `jmp .L4`  
     แล้วไปทำที่ .L4 ต่อ

cmp eax, DWORD PTR [rbp-8]

jle .L2

mov eax, DWORD PTR [rbp-4]

jmp .L4

.L2:

mov eax, DWORD PTR [rbp-8]

.L4:

pop rbp

ret

* 1. If max.c is compiled with optimization turned on (using “gcc -O2 -S max.c”), what are the differences that you may observe from the result (as compared to that without optimization). Please provide your analysis.  
       
       
       
       
       
     หลังจากทำการ optimize แล้วจะได้ว่า function max1 และ max2 นั้นมี assembly เหมือนกันเลย

max1(int, int):

push rbp

mov rbp, rsp

mov DWORD PTR [rbp-4], edi

mov DWORD PTR [rbp-8], esi

mov eax, DWORD PTR [rbp-4]

cmp eax, DWORD PTR [rbp-8]

jle .L2

mov eax, DWORD PTR [rbp-4]

jmp .L4

.L2:

mov eax, DWORD PTR [rbp-8]

.L4:

pop rbp

ret

max2(int, int):

push rbp

mov rbp, rsp

mov DWORD PTR [rbp-20], edi

mov DWORD PTR [rbp-24], esi

mov eax, DWORD PTR [rbp-20]

cmp eax, DWORD PTR [rbp-24]

setg al

movzx eax, al

mov DWORD PTR [rbp-8], eax

cmp DWORD PTR [rbp-8], 0

je .L6

mov eax, DWORD PTR [rbp-20]

mov DWORD PTR [rbp-4], eax

jmp .L7

.L6:

mov eax, DWORD PTR [rbp-24]

mov DWORD PTR [rbp-4], eax

.L7:

mov eax, DWORD PTR [rbp-4]

pop rbp

ret

max1(int, int):

cmp edi, esi

mov eax, esi

cmovge eax, edi

ret

max2(int, int):

cmp edi, esi

mov eax, esi

cmovge eax, edi

ret

* 1. Please estimate the CPU Time required by the max1 function (using the equation CPI=ICxCPIxTc). If possible, create a main function to call max1 and use the time command to measure the performance. Compare the measure to your estimation. What do you think are the factors that cause the difference? Please provide your analysis. (You may find references online regarding the CPI of each instruction.)  
       
       
       
     จากการค้นหาข้อมูล ICP ของ CPU ปัจจุบันจะอยู่ประมาณ 2.5 ดังนั้น CPI = 1/2.5 = 0.4  
     คอมพิวเตอร์ของผมมี Clock speed ที่ 4 GHz -> 2.5 \* 10-10 s  
       
     CPU Time = IC \* CPI \* Tc  
      = 4 \* 0.4 \* 2.5 \* 10-10  
      = 4 \* 10-10  
      = 0.4 ns  
       
     จากการวัด command time ได้ผลลัพธ์เป็น 0.000 s เนื่องจากเวลาในการคำนวณจริงๆ มีค่าน้อยมากๆ จนเราไม่สามารถวัดได้  
     ส่วนเวลารวมจะเป็น 0.002 s ซึ่งอาจเกิดเวลาในการ input หรือ แสดงผล

max1(int, int):

cmp edi, esi

mov eax, esi

cmovge eax, edi

ret

1. (Optimization) We will use simple fibonacci calculation as a benchmark. Please measure the execution time (using the time command) of this given program when compiling with optimization level 0 (no optimization), level 1, level 2 and level 3. (Note that some compilers do similar optimization for all level 1, level 2 and level 3. If that is the case, you will see no difference after level 1.) You may want to run each program a few times and use the average value as a result.  
     
   Ans.  
    level 0 : 10.969, 10.908, 11.066, 10.976, 10.943 เฉลี่ย 10.9724 วินาที  
    level 1 : 8.778, 8.621, 8.611, 8.671, 8.830 เฉลี่ย 8.7022 วินาที  
    level 2 : 6.241, 6.256, 6.262, 6.269, 6.281 เฉลี่ย 6.2618 วินาที  
    level 3 : 6.069, 6.049, 6.061, 6.060, 6.097 เฉลี่ย 6.0672 วินาที
2. (Analysis) As suggested by the results in Exercise 2, what kinds of optimization are used by the compiler in each level in order to make the program faster? To answer this question, use “gcc -S” to generate the assembly code for each level (e.g. “gcc -S -O2 fibo.c”) and use this result as a basis for your analysis. (Depending on your version of the compiler, the result may vary.)  
     
   แบบ level 0 แบบ level 1  
      
     
   จะเห็นว่าจาก level 0 ไป level 1 จำนวน instruction ที่เกี่ยวข้องกับ memory น้อยลงมาก ส่วนใหญ่ทำงานบน register ทั้งหมดเลย แต่ยังคงความ recursion อยู่  
     
     
     
     
     
     
     
     
     
   แบบ level 2  
     
   จะเห็นว่าการ recursion บนฟังก์ชั่น fibo หายไปแล้ว แต่ยังคงต้องมีคำสั่งเพื่อเขียน และอ่านข้อมูลจาก memory อยู่  
   แบบ level 3  
    ได้ผลลัพธ์ไม่ต่างจาก level 2 เลย

fibo(long):

push rbp

mov rbp, rsp

push rbx

sub rsp, 24

mov QWORD PTR [rbp-24], rdi

cmp QWORD PTR [rbp-24], 0

jg .L2

mov eax, 0

jmp .L3

.L2:

cmp QWORD PTR [rbp-24], 1

jne .L4

mov eax, 1

jmp .L3

.L4:

mov rax, QWORD PTR [rbp-24]

sub rax, 1

mov rdi, rax

call fibo(long)

mov rbx, rax

mov rax, QWORD PTR [rbp-24]

sub rax, 2

mov rdi, rax

call fibo(long)

add rax, rbx

.L3:

mov rbx, QWORD PTR [rbp-8]

leave

ret

fibo(long):

mov eax, 0

test rdi, rdi

jle .L6

push rbp

push rbx

sub rsp, 8

mov rbx, rdi

mov rax, rdi

cmp rdi, 1

je .L1

lea rdi, [rdi-1]

call fibo(long)

mov rbp, rax

lea rdi, [rbx-2]

call fibo(long)

add rax, rbp

.L1:

add rsp, 8

pop rbx

pop rbp

ret

.L6:

ret

fibo(long):

push r15

push r14

push r13

push r12

push rbp

push rbx

sub rsp, 168

test rdi, rdi

jle .L28

mov r12, rdi

cmp rdi, 1

je .L2

lea r15, [rdi-1]

xor r12d, r12d

.L27:

cmp r15, 1

je .L50

lea r13, [r15-1]

xor r14d, r14d

mov QWORD PTR [rsp+56], r12

mov QWORD PTR [rsp+64], r13

mov rbp, r13

mov r12, r14

.L26:

cmp rbp, 1

je .L49

mov QWORD PTR [rsp+72], r15

lea rcx, [rbp-1]

xor r14d, r14d

mov QWORD PTR [rsp+80], rbp

mov rbp, rcx

.L25:

cmp rbp, 1

je .L48

mov QWORD PTR [rsp+96], r14

lea rdi, [rbp-1]

xor r15d, r15d

mov r13, rbp

mov QWORD PTR [rsp+104], rcx

mov QWORD PTR [rsp+88], r12

mov r12, rdi

.L24:

cmp r12, 1

je .L47

lea r11, [r12-1]

xor r14d, r14d

mov QWORD PTR [rsp+112], r15

mov QWORD PTR [rsp+32], r14

mov rbp, r11

mov QWORD PTR [rsp+120], rdi

mov QWORD PTR [rsp+128], r11

mov QWORD PTR [rsp+136], r12

.L23:

cmp rbp, 1

je .L46

lea r12, [rbp-1]

mov QWORD PTR [rsp+152], rbp

xor r15d, r15d

mov QWORD PTR [rsp+144], r12

mov r14, r12

.L22:

cmp r14, 1

je .L45

lea rbp, [r14-1]

mov QWORD PTR [rsp+8], r13

xor ecx, ecx

mov QWORD PTR [rsp+16], rbp

mov rbx, rbp

mov QWORD PTR [rsp+24], r14

.L21:

cmp rbx, 1

je .L44

lea rbp, [rbx-1]

xor r13d, r13d

mov r14, rbp

mov rdx, rbp

mov rbp, rcx

mov rcx, rbx

mov rbx, r14

.L20:

mov r14, rbx

cmp rbx, 1

je .L43

mov QWORD PTR [rsp+40], rbx

xor r12d, r12d

mov rbx, rdx

.L16:

lea rdi, [r14-1]

mov QWORD PTR [rsp+48], rcx

call fibo(long)

mov rcx, QWORD PTR [rsp+48]

add r12, rax

sub r14, 2

je .L52

cmp r14, 1

jne .L16

mov rdx, rbx

mov rbx, QWORD PTR [rsp+40]

add r12, 1

.L18:

add r13, r12

sub rbx, 2

jne .L20

.L43:

mov rbx, rcx

lea rsi, [r13+1]

mov rcx, rbp

add rcx, rsi

sub rbx, 2

cmp rdx, 1

jne .L21

.L44:

mov r14, QWORD PTR [rsp+24]

mov rbx, rcx

mov rbp, QWORD PTR [rsp+16]

add rbx, 1

mov r13, QWORD PTR [rsp+8]

add r15, rbx

sub r14, 2

cmp rbp, 1

jne .L22

.L45:

mov rbp, QWORD PTR [rsp+152]

mov r12, QWORD PTR [rsp+144]

add r15, 1

add QWORD PTR [rsp+32], r15

sub rbp, 2

cmp r12, 1

jne .L23

.L46:

mov r14, QWORD PTR [rsp+32]

mov r15, QWORD PTR [rsp+112]

mov r12, QWORD PTR [rsp+136]

mov r11, QWORD PTR [rsp+128]

add r14, 1

mov rdi, QWORD PTR [rsp+120]

add r15, r14

sub r12, 2

cmp r11, 1

jne .L24

.L47:

mov r14, QWORD PTR [rsp+96]

mov rbp, r13

add r15, 1

mov r12, QWORD PTR [rsp+88]

mov rcx, QWORD PTR [rsp+104]

sub rbp, 2

add r14, r15

cmp rdi, 1

jne .L25

.L48:

mov rbp, QWORD PTR [rsp+80]

add r14, 1

mov r15, QWORD PTR [rsp+72]

add r12, r14

sub rbp, 2

cmp rcx, 1

jne .L26

.L49:

mov r14, r12

mov r13, QWORD PTR [rsp+64]

mov r12, QWORD PTR [rsp+56]

sub r15, 2

add r14, 1

add r12, r14

cmp r13, 1

jne .L27

.L50:

add r12, 1

.L2:

add rsp, 168

mov rax, r12

pop rbx

pop rbp

pop r12

pop r13

pop r14

pop r15

ret

.L52:

mov rdx, rbx

mov rbx, QWORD PTR [rsp+40]

jmp .L18

.L28:

xor r12d, r12d

jmp .L2

fibo(long):

push r15

push r14

push r13

push r12

push rbp

push rbx

sub rsp, 168

test rdi, rdi

jle .L28

mov r12, rdi

cmp rdi, 1

je .L2

lea r15, [rdi-1]

xor r12d, r12d

.L27:

cmp r15, 1

je .L50

lea r13, [r15-1]

xor r14d, r14d

mov QWORD PTR [rsp+56], r12

mov QWORD PTR [rsp+64], r13

mov rbp, r13

mov r12, r14

.L26:

cmp rbp, 1

je .L49

mov QWORD PTR [rsp+72], r15

lea rcx, [rbp-1]

xor r14d, r14d

mov QWORD PTR [rsp+80], rbp

mov rbp, rcx

.L25:

cmp rbp, 1

je .L48

mov QWORD PTR [rsp+96], r14

lea rdi, [rbp-1]

xor r15d, r15d

mov r13, rbp

mov QWORD PTR [rsp+104], rcx

mov QWORD PTR [rsp+88], r12

mov r12, rdi

.L24:

cmp r12, 1

je .L47

lea r11, [r12-1]

xor r14d, r14d

mov QWORD PTR [rsp+112], r15

mov QWORD PTR [rsp+32], r14

mov rbp, r11

mov QWORD PTR [rsp+120], rdi

mov QWORD PTR [rsp+128], r11

mov QWORD PTR [rsp+136], r12

.L23:

cmp rbp, 1

je .L46

lea r12, [rbp-1]

mov QWORD PTR [rsp+152], rbp

xor r15d, r15d

mov QWORD PTR [rsp+144], r12

mov r14, r12

.L22:

cmp r14, 1

je .L45

lea rbp, [r14-1]

mov QWORD PTR [rsp+8], r13

xor ecx, ecx

mov QWORD PTR [rsp+16], rbp

mov rbx, rbp

mov QWORD PTR [rsp+24], r14

.L21:

cmp rbx, 1

je .L44

lea rbp, [rbx-1]

xor r13d, r13d

mov r14, rbp

mov rdx, rbp

mov rbp, rcx

mov rcx, rbx

mov rbx, r14

.L20:

mov r14, rbx

cmp rbx, 1

je .L43

mov QWORD PTR [rsp+40], rbx

xor r12d, r12d

mov rbx, rdx

.L16:

lea rdi, [r14-1]

mov QWORD PTR [rsp+48], rcx

call fibo(long)

mov rcx, QWORD PTR [rsp+48]

add r12, rax

sub r14, 2

je .L52

cmp r14, 1

jne .L16

mov rdx, rbx

mov rbx, QWORD PTR [rsp+40]

add r12, 1

.L18:

add r13, r12

sub rbx, 2

jne .L20

.L43:

mov rbx, rcx

lea rsi, [r13+1]

mov rcx, rbp

add rcx, rsi

sub rbx, 2

cmp rdx, 1

jne .L21

.L44:

mov r14, QWORD PTR [rsp+24]

mov rbx, rcx

mov rbp, QWORD PTR [rsp+16]

add rbx, 1

mov r13, QWORD PTR [rsp+8]

add r15, rbx

sub r14, 2

cmp rbp, 1

jne .L22

.L45:

mov rbp, QWORD PTR [rsp+152]

mov r12, QWORD PTR [rsp+144]

add r15, 1

add QWORD PTR [rsp+32], r15

sub rbp, 2

cmp r12, 1

jne .L23

.L46:

mov r14, QWORD PTR [rsp+32]

mov r15, QWORD PTR [rsp+112]

mov r12, QWORD PTR [rsp+136]

mov r11, QWORD PTR [rsp+128]

add r14, 1

mov rdi, QWORD PTR [rsp+120]

add r15, r14

sub r12, 2

cmp r11, 1

jne .L24

.L47:

mov r14, QWORD PTR [rsp+96]

mov rbp, r13

add r15, 1

mov r12, QWORD PTR [rsp+88]

mov rcx, QWORD PTR [rsp+104]

sub rbp, 2

add r14, r15

cmp rdi, 1

jne .L25

.L48:

mov rbp, QWORD PTR [rsp+80]

add r14, 1

mov r15, QWORD PTR [rsp+72]

add r12, r14

sub rbp, 2

cmp rcx, 1

jne .L26

.L49:

mov r14, r12

mov r13, QWORD PTR [rsp+64]

mov r12, QWORD PTR [rsp+56]

sub r15, 2

add r14, 1

add r12, r14

cmp r13, 1

jne .L27

.L50:

add r12, 1

.L2:

add rsp, 168

mov rax, r12

pop rbx

pop rbp

pop r12

pop r13

pop r14

pop r15

ret

.L52:

mov rdx, rbx

mov rbx, QWORD PTR [rsp+40]

jmp .L18

.L28:

xor r12d, r12d

jmp .L2