COMPUTER ARCHITECTURE & ASSEMBLY LANGUAGE
14:332:331
Rutgers University, Spring 2021, Prof Maria Striki

HOMEWORK 2B, 68 pts = 8 + 16 + 10 + 16+18 Issue: 03-17-20 Due: Wed, 03-31-21, 9.00pm

Your Name: Ashwin Anand , John F. Crespo , Jacques Tege (We all contributed to each problem)

Problem 1 (8 points)

You have the following high level C code:

```
long long int midterm_pr1(long long int i, long long int j, int A) {  while
(A[23*i] == 11*j) && (i > 0)) {
 long long int f;
 i = i – 17;
           j = j + 17;
 f = 2 *j;
 }
 return f;
  }
```

Arguments: i, j, A (or else A[0])stored in x10, x11, x12.
Result: returned in x13

For your temporary computations use any registers you want but make a distinction whether they are saved or temp, using the stack if needed, where needed. Also, make sure your callee returns to the main function. You may use Labels such as Loop1, Loop2, … Exit1, Exit2….

NOTE: please do not forget to map each data type to the correct number of bytes: check every single variable about how many bytes it occupies in memory: 1, 2, 4, 8, 16 bytes? Then think what you need to do to go from one element to the next and to the next.

The mapping of Data Types to Byte Lengths is recorded below:

Type Name 32-bit size CPU 64-bit size CPU
short 2 bytes 2 bytes
int 4 bytes 4 bytes
long int 4 bytes 8 bytes
long long int 8 bytes 8 bytes

Questions:

Q1: (10 pts) Write the corresponding RISC-V code for the above high level C that implements the above C function as a callee within your main function (main is the caller). **Q2: (4 pts)** How many RISC-V instructions are needed to implement the above code? What is the total number of RISC-V instructions executed to complete the loop in the best case scenario (maximum possible iterations) and in the worst case scenario (minimum possible iterations)? The only information you have is that variable *i* is initialized to 100 (decimal).

Problem #1 i = x10 ; j = x11 ; A = x12
result : returned in x13

Assume //jalr x1, midterm_pr1

need $23 \cdot 4 = 92i$ because every long data type uses 4-bytes.

```
midterm_pr1:
      addi  sp, sp, -16
      sd    x8, 8(sp)
      sd    x20, 0(sp)

Loop  slli  x5, x10, 6      // x5 = 64i
      slli  x6, x10, 4      // x6 = 16i
      add   x6, x10, x6     // x6 = 80i
      slli  x5, x10, 3      // x5 = 8i
      add   x6, x5, x6      // x6 = 88i
      slli  x5, x10, 2      // x5 = 4i
      add   x6, x5, x6      // x6 = 88i + 4i = 92i
      add   x6, x6, x12     // x6 = A[23·i] = A[F-
      slli  x7, x11, 3      // x7 = 8j
      slli  x8, x11, 1      // x8 = 2·j
      add   x7, x7, x8      // x7 = 10j
      slli  x8, x11, 0      // x8 = j
      add   x8, x7, x8      // x8 = 11j
      lw    x29, 0(x6)      // x29 = A[23·i]
      bne   x8, x29, Exit
      bge   x0, x10, Exit
      addi  x10, x10, -17   // x10 = i - 17
      addi  x11, x11, 17    // x11 = j + 17
      slli  x20, x11, 1     // x20 = f = 2·j
      beq   x0, x0, Loop

Exit  addi  x14, x20, x0
      ld    x20, 0(sp)
      ld    x8, 8(sp)
      addi  sp, sp, 16
      jalr  x0, 0(x1)
```

Solution:
There are 28 RISC-V instructions to be executed to implement the code seen in the picture above. The best case scenario would be when none of the conditions are met so the program jumps to the exit label which will execute 23 RISC--V instructions. The worst case scenario would be when all of the conditions given i = 100 and the code will be executed recursively 5 times till it hits the exit label, the total count will be 20(5)+3+5 = 108 RISC-V instructions.

Problem 2: (16 pts)
Assume machine encoding to be: **a) Big Endian, b) Little Endian**.
REMARK 1: Assume that memory contexts/data are constantly **RECYCLED** every 8 memory single words, hence Mem[32] = Mem[0], Mem[36] = Mem[4], Mem[40] = Mem[8], … Memory addresses start strictly from 0 and grow ONLY POSITIVE. If a negative address is calculated then the Operating System throws an exception and the program is TERMINATED. **Also, the low address of memory is from THE LEFT SIDE (check scheme below).**

Q1 (10 pts): Study the code below. Show the **contents of memory** entries that have changed as well as the value stored in **x1, x2, x5, x6** after running this piece of code. Show the values in HEX. **x9** contains number 0x00011 (hexadecimal) and **x4** contains number 0x000014 (hex). These represent byte addresses within the same double word memory tuple. What is the lowest byte address of the double word (memory tuple) that contains these byte numbers?
x10 contains the **lowest byte address** of the **next memory double word**. What number is that? **x20** contains number (in hex): 0×9A26A009DD2EBB84. (the illustration below does not contain this number. You must place it there at the start of your handling the code).

Q2 (6 pts) Start by assuming the machine is **either Little or Big Endian**. If you find that one type of Endianness or both lead at any point to wrong memory access and program termination you must document in which instruction this happens and why. If one or both types lead to termination you are allowed to make a change in one instruction (not data) of the code that may produce the least significant impact (e.g., change offset or substitute the instruction with another) so that the program runs correctly. Report your changes. You should complete this exercise for both types of Endianess this time around.

.

sd x20, 0(x10) //**write this to memory** --- it is not placed in the scheme below… lb x11, 13(x9) // offset is in decimal number, careful: check how lb works addi x2, x11, 10 // Line A

```
li x3 <- 0 x 8F 47 6C B5 89 A7 38 2E // Line B
srai x1, x3, 4
ld x5, 12(x2) // Line C
and x6, x5, x1
sh x6, -5(x2) // Line D
```

Mem Low Order Content
Address (decimal)

Mem Low Order Address (decimal)	Content
28	17FD25EC
24 20	223101BA

16 12 8 4 0	18926163
	7E1565A9
	4701BAC6
	00011110
	01BAC789
	0100FACE

Solution:
The lowest memory address to hold the bytes stored in x9 and x4 is address 16 up to 24 (the memory tuple enclosed in these bounds), thus x10 would contain 24 as the next lowest byte address.

2.1)

Big endian on this run but Little Endian experiences problems at line 5 after starting under that method. The better execution was included

x9 = 0x11 = 17

x10 = 0x14 = 20

x20 = 0x9A26A009DD2EBB84

sd x20, 0(x10) //store doubleword x20 into x10 offset 0

lb x11, 13(x9) //load data at memory address stored in x9 plus the offset of 13-> x11 = 0x25

addi x2, x11, 10 //x2 = x11 + 10 // x2 = 0x25 + 10 = 0x2F

li x3, 0x8F476CB589A7382E //x3 = 0x8F476CB589A7382E

srai x1, x3, 4 // Shift contents of x3 over to the right by 4 and sign extend the new bits. Store in x1. x1 = 0xF8F476CB589A7382

ld x5, 12(x2) //load the doubleword located at memory address given in x2 plus the offset of 12 (47 + 12 = 59) x5 = 0x09DD2EBB840100FA

and x6, x5, x1

1111100011110100011101101100101101011000100110100111001110000010
Vs
0000100111011101001011101011101110000100000000010000000011111010
Who will survive the AND? Store result in x6

AND result
0000100011010100001001101000101100000000000000000000000010000010
//X6 = 0xB814740148000082
X6 = 0x08D4268B00000082

sh x6, -5(x2) // put the rightmost halfword bits of register x6 into memory at the location stored in x2 - the offset 5

X1 = 0xF8F476CB589A7382
X2 = 0x2F
X5 = 0x09DD2EBB840100FA
X6 = 0x08D4268B00000082

2.2)

When operating in little endian format, you may need to change the lb instruction to lbu in order to avoid a negative memory address error later on. Originally, the register x11 would be negative due to the sign extension in lb instruction which is not good for the program. Switching to lbu maintains the integrity of the program with very minimal changes.

After looking through the code, there is no place where the big endian or little endian could be violated because it does not violate the 0 - 32 bit address. The only way that would occur is if one of the bits was above 32 which would result in an error and require both the big endian and little endian to be terminated. The only error that could occur is if one of the bits was above 32 which is an error and will make both big endian and little endian to be terminated and exit the code.

Problem 3: (10 pts)

Part 1: (9 pts) Compile the assembly code for the following C code.
Part 2: (4 pts) What is the total number of RISC-V instructions needed to execute the function?

```
int f3 (int n){
 if (n>20)
 return 0;
else if (n<=1)
 return 1;
else return (4*f3(n-2)+2)
}
```

Solution:

```
f3:     addi   x6, x0, 20              x6 = 20
        addi   x7, x0, 1               x7 = 1
Rec     addi   sp, sp, -16
        sd     x1, 0(sp)
        sd     x11, 8(sp)
        bgt    x11, x6, If
        bge    x7, x11, Else If
        addi   sp, sp, -8
        sd     x8, 0(sp)
        addi   x11, x11, -2     // x11 = n-2
        jal    x1, Rec          // f3(n-2)
        ld     x8, 0(sp)        // restore stack pointer
        addi   sp, sp, 8
        ld     x1, 0(sp)
        ld     x11, 8(sp)
        add    sp, sp, 16
        slli   x8, x8, 2        // 4·f3(n-2)
        addi   x8, x8, 2        // 4·f3(n-2) +2
        jalr   x0, 0(x1)
If:     ld     x1, 0(sp)
        ld     x11, 0(sp)
        addi   sp, sp, 16
        addi   x10, x10, 3
        jalr   x0, 0(x1)
Else If: ld    x1, 0(sp)
        ld     x11, 0(sp)
        addi   sp, sp, 16
        addi   x10, x10, 1
        jalr   x0, 0(x1
```

There are 17 instructions in the recursive part the function will perform $2+\left(\frac{n}{2}\right)*17+5$ using integer rounding, the number of operations would be $\left(\frac{17}{2}\right)\cdot n+7$

the number of instructions depends on the n input

Problem 4 (16 points)

Part 1: (13 marks) Compile the RISC-V assembly code for the following C code. Assume that k and m are passed in x8 and x9 respectively. Assume that result returned in x8. This function does not have to make sense, it is a test on your knowledge of writing nested/recursive routines.

Compile the assembly code for the following C code.

```
int func (unsigned int m, unsigned int k ) {
        if (k <= 0)
        return 4;

        else if (m <=2)
        return k;

        else return 2m + 4*func(m-1,k-2) + 6*func(m,k-3);
}
```

PART 1:

```
Func:
        addi    x6, x0, 2   // x6=2
        addi    x7, x8, 0   // x7=k
Rec:
        bge   x0, x7, IF   // if  0 >=  k, go to label IF
        bge   x6, x9, ElseIF  // if  2 >= m,  go to label ElseIF
        addi  sp, sp, -24     // move sp , borrow some registers & stores values
        sd   x1, 0(sp)
        sd   x7, 8(sp)
        sd   x9, 16(sp)
        addi x9, x9, -1 // x9=  m-1
        addi x7, x7, -2 // x7 = k-2
        jal  x1, Rec  // Evaluates func(m-1, k-2)
        ld  x1, 0(sp)  // restore value
        ld  x7, 8(sp)  // restore value
        ld  x9, 16(sp)  // restore value
        addi  x29, x8, 0  // Store recursion value in x29
        addi  sp, sp, -8  // Store return add and recursion value on stack
        sd  x29, 0(sp)
        addi  x7, x7, -3  // k=k-3,  since m is already equal m
        jal  x1, Rec  // Evaluates func(m, k-3), which is stored in x8
        ld  x29, 0(sp)  // Load func(m-1,k-2)'s value
        addi  sp, sp, 8  // restore stack pointer
        ld  x1, 0(sp)  // restore value
        ld  x7, 8(sp)  // restore value
        ld  x9, 16(sp)  // restore value
        addi  sp, sp, 24
        slli x8, x8, 1   // 2 times x8
        slli  x30, x8, 2  //  x30 = 4 times x8
```

```
        add  x8, x8, x30  // x8 = 6*func(m, k-3)
        slli  x29, x29, 2    // x29 =4*func(m-1, k-2)
        add x8, x8, x29    // x8 = 6*func(m, k-3) + 4*func(m-1, k-2)
        slli  x31,  x9, 2     // x31= 2m
        addi  x8, x8 , x31   //   x8 = 6*func(m, k-3) + 4*func(m-1, k-2) + 2m
        jalr  x0, o(x1)
IF:
        addi x8, x0, 4   // x8=4
        jalr x0, 0(x1)

ElseIF:
        addi x8, x7, 0
        jalr x0, 0(x1)
```

Part 2 (3 marks) How many RISC-V instructions does it take to implement the C code from Part 1? If the variables *m* and *k* are initialized to 8 and 10 what is the total number of RISC-V instructions that is executed to complete the loop?

To produce this code it takes 37 instructions to execute.

If m=8, k=10, then there will be 8 recursive calls + 1 main caller. The main caller will go through the entirety of the program which is 37 instructions. Only 6 of the recursive calls will be considered leaf. The leaf calls each perform 3 instructions. While non-leaf will perform the 31 instructions inside of the Else label, and there are 4 non-leaf calls.

37 + 31*4 + 6*3= 179, there should be 179 total number of RISC-V instructions that is executed to complete the loop(recursive call)

Problem 5: (18 pts)

Part 1: (11 pts) Implement the following C code in RISC-V assembly.
Part 2: (4 pts) What is the total number of RISC-V instructions needed to execute the function?

```
      int fib (int n) {
 if (n==0)
 return 0;
          else if (n==1)
 return 1;
 else
 return fib(n-1) + fib(n-2);
 }
```

Part 3: (3 pts) For each function call above, show the contents of the stack after the function call is made. Assume the stack pointer is originally at address 0x7ffffffc, and follow the register convention of RISC-V (argument, saved, temporary, sp, RET, etc etc).

Solution

PART 1:

fib: // PC = 0x10000000

beq x10, x0, done // comparing x10's values to 0 and if equal then return 0

addi x5, x0, 1 // adding 0 and 1 to x5

beq x10, x5, done // comparing x10's values to x5's values and if equal then return 1

addi x2, x2, -16 // Allowing 2 different values of words to be put in to the stack

sd x1, 0(x2) // x1 register save return address

sd x10, 8(x2) // x10 register save return address

addi x10, x10, -1 // x10 = x10 -1

jal x1, fib // skipping in to fib label with jal link

ld x5, 8(x2) //Load value of x5 from stack //return fib(n-1)

sd x10, 8(x2) // Putting value of x10 into the stack

addi x10, x5, -2 // x10 = x5 - 2

jal x1, fib // skipping in to fib label with jal link ld x5, 8(x2) // loading value of second position into x5

add x10, x10, x5 // x10 = x10 + x5

ld x1, 0(x2) // Load 0th position on the stack to x1

addi x2, x2, 16 // x2 = x2 + 16

done: // label done to finish code up

jalr x0, 0(x1)   // returning the x0 back to the link to finish up the risc v code

PART 2:

The instruction fib(0) will be 2 instructions and fib(1) is reached with 4 instructions. A total of 17 instructions will be executed in every recursive step fib(N). The callee has to go through a cycle of callee to caller for every recursive step and then come back from caller to callee and execute the rest of the code from where recursion was performed.

The formula would be using N times so 2 + 4 + 17* N = 17*N +6

PART 3:

RET   sp = 0x7ffffffc -16x1

n

0x10000040   0x7ffffffc – 16x2

n-1

0x10000040 0x7ffffffc – 16x3

n-2

……..

0x10000040 0x7ffffffc – 16x(n-2)

n-3

0x10000040   0x7ffffffc – 16x(n-1)

n=2