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

Homework 2A: 55 pts = 7 + 9 + 12 + 11 + 12 + 4 pts

Issued on: Wed 02-17-2021 Due on: Fri 02-26-2021 at 22.00

Your names and contributions of group members: Ashwin Anand , John F Crespo , Jacques Trege

Problem 1 (7 pts):

Part 1 (4 pts): Translate the following C code to RISC-V assembly code, using the minimum number of instructions you can think of. The values of: `w, y, i, j` are in registers: `x4, x5, x6, x7`. Register x8 holds the based address of array A.

```
for (i=0; i<=w; i++)
 for (j=0; j<y; j+=2)
 A[3*i + j] = 2*i +3*j;
```

Code in RISC V Assembly Code:

```
          x4 =w
          x5 = y
          x6 = i
          x7 = j
          add x6 , x0 , 0       // i = 0

Loop 1:
          bne  x6 , x4 , Exit  // loop goes to exit if i != to w
          addi x6 , x6 , 1      // i =  i + 1
          add  x7 , x0 , 0      // j = 0

Loop 2:
          beq   x7 , x5 , LOOP 1  //  loop goes to exit if i != to w
          addi  x7 , x7 , 2           //  i =  i + 2
          slli    x30 , x6 , 3         //  x30 = 3*i
          addi  x30 , x30 , x7      //  x30 = 3*i +j
          add  x30 , x30 , x8       // x30 = A[3*i+j]
          slli  x28 , x6 , 2          // x28 = i*2
          slli  x29 , x7 , 3          // x28 = j*3
          add x31 , x28 , x29     //x31 = x28 + x29
          sd x31 , 0(x30)           // A[3*i +j ] = 2*i + 3*j
```

```
                beq x0 , x0 , LOOP 2
EXIT
```

Part 2 (3 pts_: How many RISC-V instructions are needed to implement the above code? If w and y are initialized to 3 and 6 and elements of A are initially 0, what is the total number of RISC-V instructions executed to complete the loop?

Total Instructions: 85 is including the initialization of values and the exits

Problem 2: (9 pts)

Find the shortest sequence of RISC-V instructions that extracts bits 19 down to 8 from register `x7` and uses the value of this field to replace bits 26 down to 15 in register `x4` without changing the other bits of registers `x7` or `x4`. (Be sure to test your code using `x7 = 0` and `x4` = 0xffffffffffffffff. Doing so may reveal a common oversight.)

Solution:

```
addi   x5 , x0 , 0xFF    // create a bit mask for 19 to 8
slli   x5 , x5 , 8       // shifting masked bits
and    x28 , x7 , x5     // apply mask to x7
slli   x5 , x4 , 7       // shift the mark to cover bits 26 to 15
xori   x5,x5,-1          // not an operation
and  x4 , x4 , x5        // "zero out" positions 26 to 15 of x4
slli   x28 , x28 ,7      // move selection from x7 into positions 26 to 15
or  x4 , x4 , x28        // load bits 26 to 15 from x28
```

Problem 3: (12 pts, 3 pts each subpart)

Part 0: Suppose the PC is set to 0x20000024. What range of addresses can be reached using the RISC V a) jal instruction, b) j instruction, c) beq instruction? In other words, what is the set of possible values for the PC after the jump instruction executes?

problem 3.

Part 0:

PC is set to 0X20000024.

(a) jal instructions

the 20 bit address field in the jal instruction can encode a distance of +/- 20^{20} bytes.

20^{20} = 1048576_{dec} - 1 = 1048575 = 000FFFFF

0X20000024 +/- 0X000FFFFE

jal range => [0X1FF00024, 0X20100022]

(b) j instructions

j instructions are using a 20 bit field:

j range => [0X1FF00024, 0X20100022]

(c) beq instructions.

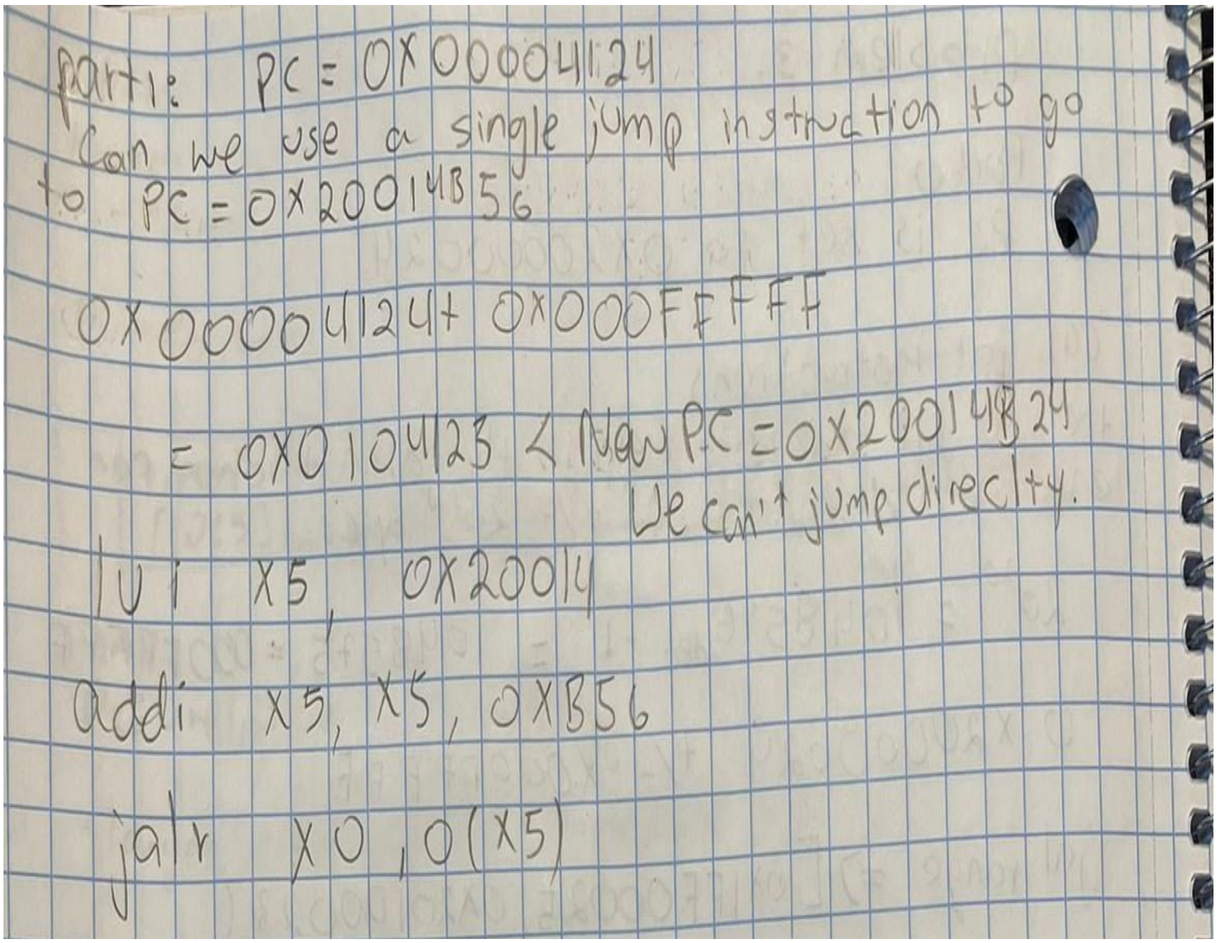
beq is using a 12 bit field.

2^{12} - 1 = 4095 = 0X00000FFF

0X20000024 +/- 0x00000FFF

beq range => [0X1FFFF024, 0X20001022]

Part 1: Suppose that the current value of PC is 0x00004124. Can we use a single jump instruction to go to PC=0x20014B56? (if yes, write the jump instruction and show the value of the immediate field in Hex. If not, use a combinations of instructions to do so and show the immediate values in Hex)


part1: PC = 0X00004124
Can we use a single jump instruction to go to PC = 0X20014B56

0X00004124 + 0X000FFFFF

= 0X0104123 < New PC = 0X20014B24

We can't jump directly.

lui X5, 0X20014

addi X5, X5, 0XB56

jalr X0, 0(X5)

Part 2: Suppose that the current value of PC is 0x00004124. Can we use a single branch instruction to go to PC=0x20014B56? (if yes, write the branch instruction and show the value of the immediate field in Hex. If not, use a combinations of instructions to do so and show the immediate values in Hex)

part 2: PC = 0X00004124

Can we use a single jump instruction to go to PC = 0X20014B56 ?

0X00004124 + 0X00000FFF

= 0X00005123 < New PC = 0X20014B24

So we can't jump directly to PC = 0X20014B56

```
lui x5, 0x20014
addi x5, x5, 0xB56
jalr x0, 0(x5)
```

Part 3: Suppose that the current value of PC is 0x1FFFFE48. Can we use a single branch instruction to go to PC=0x20000C48? (if yes, write the branch instruction and show the value of the immediate field in Hex. If not, use a combinations of instructions to do so and show the immediate values in Hex)

part 3. PC = 0X1FFFFE48

can we use a single branch instruction to goto
PC = 0X20000C48

0X1FFFE48 + 0X00000FF
= 20000E47 = 536874567.

0X20000E47 > 0X20000C48.

This means it is possible to use a single branch instruction in this case

Problem 4: (10 pts)

Part 1: (3 pts) Consider the following code:

```
lb x5, 0(x10)
sd x5, 8(x10)
```

Assume that register x10 contains address 0×10000000 and the data at address is 0×7FEEDDCCCDBC2365.

Q 1: What value is stored in 0×10000008 on a big-endian machine?
Q 2: What value is stored in 0×10000008 on a little-endian machine?

Part 2: (5 pts) Write the RISC-V assembly code that creates the above 64-bit constant (0×7FEEDDCCCDBC2365) and stores that value to register x10.

Part 3: (2 pts) Provide the instruction type, assembly language instruction, and binary representation of instruction described by the following RISC-V fields: opcode=0x3, funct3=0x3, rs1=27, rd=3, imm=0x4

Solution:

Part 1 Q1) The value stored in 0×10000008 on a big-endian machine is 0x65

Part 1 Q2) The value stored in 0×10000008 on a little-endian machine is 0x7F

Part 2: RISC-V Code to Generate 64-bit constant and store in register 10.

lui x10, 0x7FEEE

addi x10, x10, 0xDCC

lui x10, 0xCDBC2

addi x10, x10, 0x365

Part 3: The instruction type is I-Type or I-Format, instruction is ld and the binary representation is as follows

00000000010011011011000110000011

Imm

Rs1

Funct3

Rd

Opcode

Problem 5: (12 pts)

Consider the following code sequence and memory state (memory contents are given in hexadecimal. Other values are in decimal).

a) Assume that the machine is **Big Endian**.

b) Assume that the machine is **Little Endian**.

Show the **contents of memory** as well as the value stored in **x5** and **x6** after running this code for both cases. Show the value in **HEX**.

```
addi   x2, x0, 14
lui    x3, 27
srli   x5, x3, 2
lb     x1, 0(x2)
and    x6, x1, x5
sw     x6, 10(x2)
```

17FD25EC	28
223101BA	24
18926163	20
7E1565A9	16
7701BAC7	12
00011110	8
01BAC789	4
0100FACE	0

Memory Address Decimal:

*** Assume that byte addressing starts from left to right, i.e., memory byte address 0 corresponds to 01, memory byte address 3 corresponds to CE, memory byte address 4 corresponds to 01, memory byte address 5 corresponds to BA.

*** If you have already solved the problem assuming the opposite direction, I will take your solution as correct if it is correct in all other analysis.

Solution:

problem 5:

```
addi  x2, x0, 14
lui   x3, 27
srli  x5, x3, 2
lb    x1, 0(x2)
and   x6, x1, x5
sw    x6, 10(x2)
```

17FD25EC	28
223101BA	24
18926163	20
731565A9	16
7701BAC7	12
0001111 0	8
01BAC789	4
0100FACE	0

(a) Assume that the machine is Big Endian.

```
addi  x2, x0, 14    // x2 = x0 + 14 = 0 + 14
lui   x3, 27        // x3 = (27) << 12 = 110,592
srli  x5, x3, 2     // x5 = x3 >> 2 = 0x00006C00
lb    x1, 0(x2)     // x1 = mem[0 + 14] = 0xFFFFFFBA
and   x6, x1, x5    // x6 = x1 and x5 = 0xFFFFFFBA
                       and 0x00006C00
sw    x6, 10(x2)    // x6 = mem[10 + x2] = mem[10 + 14]
                       = mem[24] = 0x00006C00
```

17FD25EC	28
00006C00	24
18926163	20
7E1565A9	16
7701BAC7	12
0001111 0	8
01BAC789	4
0100FACE	0

3 2 1 0

Little Endian

17FD25EC	28
006C0000	24
18926163	20
7E1565A9	16
7701BAC7	12
0001111 0	8
01BAC789	4
0100FACE	0

0 1 2 3

Big endian

Big Endian

value in register x5 = 006C00000

value in register x6 = 006C00000

Little Endian
value in register x5 = 000006C00
value in register x6 = 000006C00

Problem 6: (4 pts) Let's explore the value of labels as we discussed in class. You can work on this problem either through your lab or do some deep digging in the resources available and figure this out. For this experiment provide to beq and/or bne labels nearby the instructions executed.

1. Implement instruction: you are to implement the program of our slides presentation in slide: 100 and verify the value of the labels. What number do they show? You may load registers with the same value using li pseudo-instruction, which loads a 64-bit binary number directly to the register.

2. Explore what is the impact on your compiler's behavior if you type: beq x10, x11 4000. This is larger number supplied by the range of beq.

3. Explore what is the impact on your compiler's behavior if you type: beq x10, x11 1001. This is an odd address we feed to beq instruction. What will happen now?

Solution:

1) The numbers that the labels Exit and Loop show are 12 and -20 respectively.
2) A traditional compiler would get an error if the line beq x10, x11 4000 because 4000 is out of range for addressing.
3) When typing beq x10, x11 1001 the compiler will once again give an error because beq and its format is only compatible with even numbers. However 1001 could be doubled and then the new address would be 2002 in order to properly work.