Akash Bhowmick – ab6858

(Answers marked in red)

NYU Tandon School of Engineering

Fall 2020, ECE 6913 – Section B, INET

Homework Assignment 2

Course Assistant: Guanhong Liu, gl1937@nyu.edu

Instructor: Azeez Bhavnagarwala, email: ajb20@nyu.edu

[released Saturday September 12th 2020] [due* **Saturday September 19th 2020, *before* 11:55 PM**]

You *are allowed* to discuss HW assignments only with other colleagues taking the class. You are *not allowed* to share your solutions with other colleagues in the class. Please feel free to reach out to the Instructor during office hours or by appointment if you need any help with the HW.

Please enter your responses in this Word document after you download it from NYU Classes. *Please use the NYU Classes portal to upload your completed HW. Please do not upload images of handwritten sheets or PDFs of scanned sheets of handwritten solutions. Please be sure to type-in your solutions into Word or Google Docs and upload machine readable documents only.*

- *In RISCV, only load and store instructions access memory locations*
- *These instructions must follow a 'format' to access memory*
- *Assume a 32-bit machine in all problems unless asked to assume otherwise*

Problem 1:

Assume address in memory of '`A[0]`', '`B[0]`' and '`C[0]`') are stored in Registers `x27, x30, x31`. Assume values of variables `f, g, h, i,` and `j` are assigned to registers `x5, x6, x7, x28, x29` respectively

Write down RISC V Instruction(s) to

(a) Load Register `x5` *with content* of `A[10]`

```
lw x5, 40(x27)
```

(b) Store contents of Register `x5` into `A[17]`

```
sw x5, 68(x27)
```

(c) add 2 operands: one in `x5` - a register, the other in in Register x6. Assume result of operation to be stored in register `x7`

```
add x7, x5, x6
```

(d) copy contents at one memory location to another: `C[g] = A[i+j+31]`

```
add x8, x28, 29
addi x8, x8, 31
slli x8, x8,2
lw x9, x8(x27)
slli x6, x6, 2
sw x9, x6(x31)
```

(e) implement in RISC V these line of code in C:

(i) `f = g - A[B[9]]`

```
lw x8, 36(x30)// in x8 load B[9]
slli x8, x8, 2 // offset adjustment
lw x9, x8(x27) // in x9 load A[B[9]]
sub x9, x6, x9 // subtract g – A[B[9]]
addi x5, x9, 0x0 // put the result from the subtraction into
f
```

(ii) `f = g - A[C[8] + B[4]]`

```
lw x8, 16(x30)// load B[4]
lw x9, 32(x31) // load C[8]
add x10, x9, x8 // add C[8] and B[4]
lw x9, x10(x27) // load A[C[8] + B[4]]
sub x9, x6, x9 // subtract contents of x9 from g
addi x5, x9, 0x0 // put the result from the subtraction into
f
```

(iii) `A[i] = B[2i+1], C[i] = B[2i]`

```
add x8, x28, x28 // 2i
addi x8, x8, 0x01 // 2i +1
lw x9, x8(x30) // load B[2i +1]
slli x10, x28, 2 // multiply by 4 to get the address offset
sw x9, x10(x27) // store the result from B[2i+1] into A[i]
```

```
add x8, x28, x28 // 2i
lw x9, x8(x30) // load B[2i]
slli x10, x28, 2 // multiply by 4 to get the address offset
sw x9, x10(x31) // store the result from B[2i] into C[i]
```

(iv) `A[i] = 4B[i-1] + 4C[i+1]`

```
sub x8, x28, 1
slli x10, x8, 2
lw x10, x10(x30)
slli x10,x10, 2
add x9, x28, 1
slli x9, x9, 2
lw x11, x11(x31)
slli x11, x11, 2
add x10, x10, x11
slli x11, x28, 2
sw x10, x11(x27)
```

(v) `f = g - A[C[4] + B[12]]`

```
lw x8, 16(x30)// load B[12]
lw x9, 32(x31) // load C[4]
add x10, x9, x8 // add C[4] and B[12]
lw x9, x10(x27) // load A[C[4] + B[12]]
sub x9, x6, x9 // subtract contents of x9 from g
addi x5, x9, 0x0 // put the result from the subtraction into
f
```

Problem 2:

Assume the following register contents:

`x5 = 0x00000000AAAAAAAA, x6 = 0x1234567812345678`

a. For the register values shown above, what is the value of `x7` for the following sequence of instructions?

srli x7, x5, 16 – instruction will shift contents of x5 right by 16 and final contents will be stored in x7

x7 = x5 / 2^16 = 0000 0000 0000 AAAA

addi x7, x7, -128 – add immediate x7 <= x7 – 128(0x80)

x7 = 0000 0000 0000 AA2A

srai x7, x7, 2 – shift right arithmetic immediate (similar to logical shift except we don't make change the sign)

1010 1010 0010 1010 >> 2 0010 1010 1010 0010 = 2AA2

x7 = 0000 0000 0000 2AA2

and x7, x7, x6 – bitwise and between x7 and x6

x7 = x7 & x6

0001 0010 0011 0100 0101 0110 0111 1000 0001 0010 0011 0100 0101 0110 0111 1000

0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0010 1010 1010 0010

x7 = 0000 0000 0000 0220

b. For the register values shown above, what is the value of x7 for the following sequence of instructions?

slli x7, x6, 4 – shift left immediate (assuming initial value of x6) => multiply by 2^4 = 16

x7 = 2345 6781 2345 6780

c. For the register values shown above, what is the value of x7 for the following sequence of instructions?

srli x7, x5, 3 – shift right logical immediate -> divide x5 by 2^3 = 8 and put contents into x7 (assume initial values of x5)

0000 0000 AAAA AAAA

1010 1010 1010 1010 1010 1010 1010 1010

0001 0101 0101 0101 0101 0101 0101 0101

x7 = 0000 0000 1555 5555

andi x7, x7, 0xFEF

only values that matter 5555 & FEF

0101 0101 0101 0101

0000 1111 1110 1111

x7 = 0000 0000 0000 0545

Problem 3:

For each RISC-V instruction below, identify the instruction format and show, wherever applicable, the value of the opcode (op), source register (rs1), source register (rs2), destination register (rd), immediate (imm), func3, func7 fields. Also provide the 8 hex char (or 32 bit) instruction for each of the instructions below

add x5, x6, x7

opcode: 0110011

rs1: 00110

rs2: 00111

rd:00101

imm:0000000

func3: 000

func7: 0000000

hexadecimal: 33/0/00

32 bit: 0000 000 00111 00110 000 00101 0110011

addi x8, x5, 512

opcode:0010011

rs1: 00101

rs2: n/a

rd: 01000

imm: 10 0000 0000

func3: 000

func7: n/a

hexadecimal: 13/0

32 bit: 0011 0000 0000 00101 000 01000 0010011

ld x3, 128(x27)

opcode: 0000011

rs1: 11011

rs2: n/a

rd: 00011

imm: 1000 0000

func3: 000

func7: n/a

hexadecimal: 03/0

32 bit: 0000 1000 0000 11011 000 00011 0000011

sd x3, 256(x28)

opcode: 0100011

rs1: 11100

rs2: 00011

rd: n/a

imm: 0001 0000 0000

func3: 011

func7: n/a

hexadecimal: 23/3

32 bit: 0001 000 11100 00011 011 00000 0100011

```
beq x5, x6 ELSE  #ELSE is the label of an instruction 16 bytes larger
                 #than the current content of PC
```

opcode: 1100011

rs1: 00101

rs2: 00110

rd: n/a

imm:16 * PC or PC << 4

func3: 000

func7: n/a

hexadecimal: 63/0

32 bit: imm[12] imm[10:5] 00110 00101 000 imm[4:1]imm[11] 1100011

add x3, x0, x0

opcode: 0110011

rs1: 00000

rs2: 00000

rd: 00011

imm:0000000

func3: 000

func7: 0000000

hexadecimal: 33/0/00

32 bit: 0000000 00000 00000 000 00011 0110011

auipc x3, FFEFA

opcode: 0010111

rs1: n/a

rs2: n/a

rd: 00011

imm:00010

func3: n/a

func7: n/a

hexadecimal: 17

32 bit: 0000 0000 0000 0001 0000 00011 001 0111

jal x3 ELSE

opcode: 1101111

rs1: n/a

rs2: n/a

rd: 00011

imm: PC * 16

func3: n/a

func7: n/a

hexadecimal: 6F

32 bit: imm[20|10:1|11|19:12] 00011 1101111

Problem 4:

(a) For the following C statement, write a minimal sequence of RISC-V assembly instructions that performs the identical operation. Assume `x5 = A`, and `x11` is the base address of C.

```
A = C[0] << 16;
```

```
slli x5 x11 16
```

(b) Find the shortest sequence of RISC-V instructions that extracts `bits 12 down to 7` from register `x3` and uses the value of this field to replace bits `28 down to 23` in register `x4` without changing the other bits of registers `x3` or `x4`. (Be sure to test your code using `x3 = 0` and `x4 = 0xffffffffffffffff`. Doing so may reveal a common oversight.)

x3 = 0x 0000 0000 0000 0000 – 0000 0000 0000 0000 0000 0000 0000

------

x4 = 0x ffff ffff ffff ffff – 1111 1111 1111 1111 1111 1111 1111 1111

------

addi x8 x0 x0

addi x7 x0 0x0FC0

slli x7 x7 1

and x7 x7 x3

slli x7 x7 11

addi x8 x8 0x0FC0

slli x8 x8 12

xori x8 x8 -1

and x8 x8 x4

or x4 x7 x8

(c) Provide a minimal set of RISC-V instructions that may be used to implement the following pseudoinstruction:

```
not x5, x6 // bit-wise invert
```

```
xori x5, x6, 0x01
```

[Hint: note that there is no 'not' instruction in RISCV. However, an XOR immediate instruction could be used]

Problem 5:

Suppose the program counter (PC) is set to $0x60000000_{hex}$.

a. What range of addresses can be reached using the RISC-V *jump-and-link* (`jal`) instruction? (In other words, what is the set of possible values for the PC after the jump instruction executes?)

In the jal instruction there are 20 bits for the offset values in 2'complement => the range would be from -2^19 to 2^19 -1

Maximum address value = 0x60000000 + 2^19 – 1 = 0x 6007 FFFF

Minimum address value = 0x 6000 0000 – 2^19 = 0x 5FFF 8000

b. What range of addresses can be reached using the RISC-V *branch if equal* (`beq`) instruction? (In other words, what is the set of possible values for the PC after the branch instruction executes?)

The range of beq instruction is -128 to 127

Maximum address value = 0x60000000 + 127 = 0x 6000 007F

Minimum address value = 0x60000000 – 128 = 0x 5FFF FF20

Problem 6:

Assume that the register `x6` is initialized to the value 10. What is the final value in register `x5` assuming the `x5` is initially zero?

The final value will be 20, this is because x6 will decrease by 1 10 times, and x5's value will increase by 2 during the same time period meaning 2*10 = 20

```
LOOP:     beq x6, x0, DONE
          addi x6, x6, -1
          addi x5, x5, 2
          jal x0, LOOP
DONE:
```

a. For the loop above, write the equivalent C code. Assume that the registers `x5` and `x6` are integers `acc` and `i`, respectively.

```
acc = 0; i = 10
while (i>0){
        i = i-1
        acc = acc + 2
}

or

int main{
        int acc = 0;
        int I = 10;
loop:
        (if I == 0)
                goto done;
```

```
        else {
                I = I – 1;
                Acc = acc +2;
                Goto loop;
        }
done:
        printf("acc = %d", acc);
        return acc;
}
```

b. For the loop written in RISC-V assembly above, assume that the register `x6` is initialized to the value N. How many RISC-V instructions are executed?

For every iteration the loop will run 4 times and then the final check. This means that given that x6 has the value N the loop will run 4*N + 1 times. In the earlier example there were a total of 41 RISC-V instructions exceuted.

c. For the loop written in RISC-V assembly above, replace the instruction "`beq x6, x0, DONE`" with the instruction "`blt x6, x0, DONE`" and write the equivalent C code.

```
int main{
        int acc;
        int I;
loop:
        (if I < 0)
                goto done;
        else {
                I = I – 1;
                Acc = acc +2;
                Goto loop;
        }
done:
        printf("acc = %d", acc);
        return acc;
}
```

The difference is now we are not checking for equality but rather less than 0.

Problem 7:

a. Translate the following C code to RISC-V assembly code. Use a minimum number of instructions. Assume that the values of `a, b, i,` and `j` are in registers `x5, x6, x7,` and `x29,` respectively. Also, assume that register `x10` holds the base address of the array `D`.

```
for(i=0; i<a; i++)
     for(j=0; j<b; j++)
          D[4*j] = i + j;
```

```
X5 = a; x6 = b; x7 = I; x29 = j;
```

```
Add x7, x0, x0
Loop 1: Beq x7, x5, Done
        Addi x7, x7, 1
        Add x29, x0, x0
Loop 2: beq x29, x6, Loop1
        Addi x29, x29, 1
        Slli x30, x30, 2
        Add x30, x30, x10
        Add x31, x7, x29
       Sd x31, 0(x30)
       Beq x0, x0, Loop2
Done:
```

b. How many RISC-V instructions does it take to implement the C code from 7a. above? If the variables **a** and **b** are initialized to **10** and **1** and all elements of **D** are initially 0, what is the total number of RISC-V instructions executed to complete the loop?

Loop 2 will run b times, and each run loop 2 will contribute 7 instructions. Furthermore, loop 2 will be run a amount of times as well called from loop 1. Loop 1 is 3 instructions long and will run 10 times.

Total number of executions: (3*10) + (7*1*10) + 1 = 101

Problem 8:

Consider the following code:

```
lb x6, 0(x7)
sd x6, 8(x7)
```

Assume that the register x7 contains the address **0×10000000** and the data at address is 0×**11**22334455667**88**.

a. What value is stored in `0×10000007` on a bigendian machine?

Address	Value
0x10000000	0x11
0x10000001	0x22
0x10000002	0x33
0x10000003	0x44
0x10000004	0x55
0x10000005	0x66
0x10000006	0x77
0x10000007	0x88

b. What value is stored in **`0x10000007`** on a littleendian machine?

Address	Value
0x10000000	0x88
0x10000001	0x77
0x10000002	0x66
0x10000003	0x55
0x10000004	0x44
0x10000005	0x33
0x10000006	0x22
0x10000007	0x11

Problem 9:

Write the RISC-V assembly code that creates the 64-bit constant $0x1234567812345678_{hex}$ and stores that value to register `x10`.

```
Lower side: 0001 0010 0011 0100 0101 0110 0111 1000
Upper side: 0001 0010 0011 0100 0101 0110 0111 1000
Lui x30, 74565
Addi x30, x30, 1656
Lui x31, 74565
Addi x31, x31, 1656
Slli x31, x31, 0x20
Or x10, x30, x31
```

Problem 10: Assume that **`x5`** holds the value 128_{10}.

a. For the instruction **`add x30, x5, x6,`** what is the range(s) of values for **`x6`** that would result in overflow?

If x5 and x6 are 8-bit registers, x6 >= 128

If x5 and x6 are 16-bit registers, x6 >= 65408

If x5 and x6 are 32-bit registers, x6 >= 2^32

b. For the instruction **`sub x30, x5, x6,`** what is the range(s) of values for **`x6`** that would result in overflow?

Overflow will occur is x6 will be in the range of -2^31 to –(2^31 – 128)

c. For the instruction **`sub x30, x6, x5,`** what is the range(s) of values for **`x6`** that would result in overflow?

Overflow will occur is x6 will be in the range of -2^31 to –(2^31 – 127)