**NYU Tandon School of Engineering**

**Fall 2022, ECE 6913**

**Homework Assignment 3**

---

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

*Course Assistants*

Varadraj Kakodkar (vns2008), Kartikay Kaushik (kk4332), Siddhanth Iyer (si2152), Swarnashri Chandrashekar (sc8781), Karan Sheth (kk4332), Haotian Zheng (hz2687), Haoren Zhang (kk4332), Varun Kumar (vs2411)

**Homework Assignment 3** [released Monday September 26th 2022] [due Wednesday October 5th by 11:59PM]

You *are allowed* to discuss HW assignments with anyone. You are *not allowed* to share your solutions with other colleagues in the class. Please feel free to reach out to the Course Assistants or 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 Brightspace portal to upload your completed HW.*

---

- *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]`

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

(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`

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

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

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

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

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

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

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

Solutions:

(a) lw x5, 40(x27)

(b) sw x5, 68(x27)

(c) add x7, x5, x6

(d)

add x28, x28, x29 // i + j

addi x28, x28, 31 // i + j + 31, the offset

slli x28, x28, 2 // calculate the total address offset of (i + j + 31)

add x28, x28, x27 // get the address of A[i + j + 31]

lw x28, 0(x28) // load the content at address of A[i + j + 31] to x28

slli x6, x6, 2 // calculate the total address offset of g

add x6, x6, x31 // get the address of C[g]

sw x28, 0(x6)

(e)

(i)

lw x7, 36(x30) // get B[9]

slli x7, x7, 2 // get the offset

add x7, x7, x27 // get the address of A[B[9]]

lw x7, 0(x7) // load content at A[B[9]]

sub x5, x6, x7

(ii)

lw x7, 32(x31) // get C[8]

lw x28, 16(x30) // get B[4]

add x7, x7, x28 // get C[8] + B[4]

slli x7, x7, 2 // get the total offset

```
add x7, x7, x27 // get the address of A[C[8] + B[4]]
lw x7, 0(x7) // load content at A[C[8] + B[4]]
sub x5, x6, x7
```

(iii)

```
add x5, x28, x28 // get 2 * i
addi x5, x5, 1 // get 2 * i + 1
slli x5, x5, 2 // get the offset of 2 * i + 1
add x5, x5, x30 // get the address of B[2 * i + 1]
lw x7, 0(x5) // load B[2 * i + 1] to x7
slli x6, x28, 2 //  get the offset of i
add x6, x6, x27 // get the address of A[i]
sw x7, 0(x6)

addi x5, x5, -4// get the address of B[2 * i]
lw x7, 0(x5) // load B[2 * i] to x7
slli x6, x28, 2 //  get the offset of i
add x6, x6, x31 // get the address of C[i]
sw x7, 0(x6)
```

(iv)

```
addi x5, x28, -1 // get i - 1
slli x5, x5, 2 // get the offset of i - 1
add x5, x5, x30 // get the address of B[i - 1]
lw x6, 0(x5) // load B[i - 1] to x6
slli x6, x6, 2 // get 4 * B[i - 1]

addi x5, x28, 1 // get i + 1
```

```
slli x5, x5, 2 // get the offset of i + 1
add x5, x5, x31 // get the address of C[i + 1]
lw x7, 0(x5) // load C[i + 1] to x7
slli x7, x7, 2 // get 4 * C[i + 1]
add x6, x6, x7 // get 4 * B[i - 1] + 4 * C[i + 1]

slli x5, x28, 2 //get the offset of i
add x5, x5, x27 // get the address of  A[i]
sw x6, 0(x5) // store 4 * B[i - 1] + 4 * C[i + 1] to A[i]
```

(v)

```
lw x7, 16(x31) // get C[4]
lw x28, 48(x30) // get B[12]
add x7, x7, x28 // get C[4] + B[12]
slli x7, x7, 2 // get the total offset
add x7, x7, x27 // get the address of A[C[4] + B[12]]
lw x7, 0(x7) // load content at A[C[4] + B[12]]
sub x5, x6, x7
```

**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
x7 = 0x0000000000000AAAA
addi x7, x7, -128
x7 = 0x000000000000AA2A
```

```
srai x7, x7, 2
x7 = 0x0000000000002A8A
and x7, x7, x6
x7 = 0x0000000000000208
```

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

```
slli x7, x6, 4
x7 = 0x2345678123456780
```

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

```
srli x7, x5, 3
x7 = 0x0000000015555555
andi x7, x7, 0xFEF
x7 = 0x0000000000000545
```

**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
Format: R
op: 011 0011, 0x33
rs1: 6
rs2: 7
rd: 5
imm: not applicable
func3: 000
func7: 0000 000
32 bits code:
0000 0000 0111 0011 0000 0010 1011 0011
```

8 hex char:

0x007302B3

addi x8, x5, 512

Format: I

op: 001 0011, 0x13

rs1: 5

rs2: not applicable

rd: 8

imm: 512, 0x200

func3: 000

func7: not applicable

32 bits code:

0010 0000 0000 0010 1000 0100 0001 0011

8 hex char:

0x20028413

ld x3, 128(x27)

Format: I

op: 000 0011, 0x03

rs1: 27

rs2: not applicable

rd: 3

imm: 128

func3: 011

func7: not applicable

32 bits code:

0000 1000 0000 1101 1011 0001 1000 0011

8 hex char:

0x080DB183

```
sd x3, 256(x28)
```

Format: S

op: 010 0011, 0x23

rs1: 28

rs2: 3

rd: not applicable

imm: 256

func3: 011

func7: not applicable

32 bits code:

0001 0000 0011 1110 0011 0000 0010 0011

8 hex char:

0x103E3023

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

Format: SB

op: 110 0011, 0x63

rs1: 5

rs2: 6

rd: not applicable

imm: 16

func3: 000

func7: not applicable

32 bits code:

0000 0010 0110 0010 1000 0000 0110 0011

8 hex char:

0x02628063

**add** x3, x0, x0

Format: R

op: 011 0011, 0x33

rs1: 0

rs2: 0

rd: 3

imm: not applicable

func3: 000

func7: 0000 000

32 bits code:

0000 0000 0000 0000 0000 0001 1011 0011

8 hex char:

0x000001B3

auipc x3, FFEFA

Format: U

op: 001 0111, 0x33

rs1: not applicable

rs2: not applicable

rd: 3

imm: 0xFFEFA

func3: not applicable

func7: not applicable

32 bits code:

0000 0000 0000 0000 1111 0001 1001 0111

8 hex char:

0x0000F2B3

```
jal x3 ELSE
Format: UJ
op: 110 1111, 0x6F
rs1: not applicable
rs2: not applicable
rd: 3
imm: 16
func3: not applicable
func7: not applicable
32 bits code:
0000 0010 0000 0000 0000 0001 1110 1111
8 hex char:
0x020001EF
```

**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;
lw x5, 0(x11)
slli x5, x5, 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.)

addi x7, x0, 0x3F // set a 6-bits length

slli x7, x7, 7 // leftshift 7 bits to corresponding position

and x8, x7, x3 // extract the bits 12 to 7 of x3, and then store it to x8

slli x8, x8, 16 // align this part to the 28th bit to 23th bit

slli x7, x7, 16

xor x7, x7, -1 // align this mask and set the corresponding number to 0

and x4, x4, x7 // set the 28th bit to 23th bit of x4 to 0

or x4, x4, x8 // set the extracted part from x3 to x4

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

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

```
xor x5, x6, -1
```

*[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?)

According to the instruction manual, the jal instruction stores the 1 - 20$^{th}$ bit of the immediate number. The least significant bit of the immediate number is always 0. Therefore, the maximum immediate number is

0 1111 1111 1111 1111 1110 = 0x0FFFFE = 1048574

The least immediate number is

1 0000 0000 0000 0000 0000 = 0x100000= -2^20 = -1048576

Thus, the possible values for the PC after the jump instruction is

[0x60000000 - 0x10000, 0x60000000 + 0x0FFFFE] = [5FF0 0000, 600F FFFE]

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?)

According to the instruction manual, the jal instruction stores the 1 - 12$^{th}$ bit of the immediate number. The least significant bit of the immediate number is always 0. Therefore, the maximum immediate number is

0 1111 1111 1110 = 0x0FFE = 4094

The least immediate number is

1 0000 0000 0000 = 0x1000= -2^12 = -4096

Thus, the possible values for the PC after the jump instruction is

[0x60000000 - 0x1000, 0x60000000 + 0x0FFE] = [5FFF F000, 6000 0FFE]

**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?

```
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;
for (i = 10; i > 0; i--)
        acc += 2;
```

**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?

4 * N + 1

Because after the last iteration, the first line "beq" need to be executed again. Then this piece of code will finally exit.

**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.

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

```
acc = 0;
for (i = 10; i >= 0; i--)
        acc += 2;
```

**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;
```

addi x7, x0, 0 // i = 0, and only initialize once

```
LOOPI:
        bge x7, x5, ENDI // if i >= a, break
        addi x30, x10, 0 // x30 = The address of D[0]
        addi x29, x0, 0 // j = 0, initialize every iteration of i
LOOPJ:
        bge x29, x6, ENDJ // if j >= b, break
        add x31, x7, x29 // let x31 = i + j
        sw x31, 0(x30) // store D[4 * j] = x31/ (i + j)
        addi x30, x30, 32 // x30 = The address of D[4*(j+1)]
        addi x29, x29, 1 // j++
        jal x0, LOOPJ
ENDJ:
        addi x7, x7, 1 // i++;
        jal x0, LOOPI
ENDI:
```

**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?

Totally 12 instructions.

If a = 10 and b = 1, totally 12 * 10 + 1 (initialization of i) + 1 (final judgement of i) = 122

**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?

Since the machine is big-endian, the most significant byte of "0×**11**22334455667**88**" will be stored at the least address. Thus the address 0x10000007 will store byte "0x88".

**b.** What value is stored in **0×10000007** on a littleendian machine?

Since the machine is little-endian, the most significant byte of "0×**11**22334455667**88**" will be stored at the biggest address. Thus the address 0x10000007 will store byte "0x11".

**Problem 9:**

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

lui x10, 0x12345

addi x10, x10, 0x678

slli x10, x10, 32

lui x5, 0x12345

addi x5, x5, 0x678

add x10, x10, x5

**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?

x30 will store the sum of x5 and x6. Since there is an overflow, x6 should be a positive number.

Therefore, the minimum value of x6 is:

2^31 - 128 = 2147483519

The range of x6 is [2147483519, 2147483647].

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

x30 will store the restul of x5 subtract x6. Since there is an overflow and x5 = 128 > 0, x6 should be negative.

So the minimum value of x6 satisfies:

x5 - x6 = 2^31, x6 = -2147483520

Thus, the range for x6 is [-2147483648, -2147483520]

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

x30 will store the restul of x6 subtract x5. Since there is an overflow and x5 = 128 > 0, x6 should be negative.

So the maximum value of x6 satisfies:

x6 - x5 = 2^31 - 1 = 2147483647 = 0x7FFFFFFF

x6 = -2147483521

Thus, the range for x6 is [-2147483648, -2147483521]