NYU Tandon School of Engineering

Fall 2023, ECE 6913

Homework Assignment 2

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

Course Assistant Office Hour Schedule

On Zoom: 9:30AM – 11AM Monday, Tuesday, Wednesday & Thursday

On Zoom: 4PM – 6PM Monday (Focus on Project A)

1. Arushi Arora, aa10350@nyu.edu

2. Prashanth Rebala, pr2359@nyu.edu

3. Moin Khan, mk8793@nyu.edu

4. Raj Ghodasara, g4357@nyu.edu

Homework Assignment 2 [released Saturday Sept 30th 2023] [due Sunday Oct 8th 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]`

```
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 x28, x28, x29
addi x28, x28, 31
slli x28, x28, 2
add x28, x28, x27
slli x6, x6, 2
add x6, x6, x31
lw x28, 0(x28)
sw x28, 0(x6)
```

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

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

```
lw x8, 36(x30)
slli x8, x8, 2
add x8, x8, x27
lw x8, 0(x8)
sub x5, x6, x8
```

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

```
lw x30, 16(x30)
lw x31, 32(x31)
add x24, x30, x31
slli x24, x24, 2
add x4, x27, x24
lw x4, 0(x4)
sub x5, x6, x4
```

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

```
add x26, x28, x28
addi x26, x26, 1
slli x26, x26,2
add x26, x26, x30
slli x25, x28, 2
```

```
add x25, x25, x27
lw x24, 0(x26)
sw x24, 0(x25)
addi x26, x26, -4
slli x23, x28, 2
add x23, x23, x31
addi x26, x26, -4
slli x23, x28, 2
add x23, x23, x31
lw x24, 0(x26)
sw x24, 0(x23)
```

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

```
addi x26, x28, -1
slli x26, x26, 2
add x26, x26, x30
addi x25, x28, 1
slli x25, x25, 2
add x25, x25, x31
slli x24, x28, 2
add x24, x24, x27
lw x23, 0(x26)
slli x23, x23, 2
lw x22, 0(x25)
slli x22, x22, 2
add x23, x23, x22
sw x23, 0(x24)
```

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

```
lw x30, 48(x30)
lw x31, 16(x31)
add x24, x30, x31
```

```
slli x24, x24, 2
add x4, x27, x24
lw x4, 0(x4)
sub x5, x6, x4
```

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
```

`0x000000000000AAAA`

```
addi x7, x7, -128
```

`0x000000000000AA4A`

```
srai x7, x7, 2
```

`0x0000000000002A8A`

```
and x7, x7, x6
```

`0x0000000000000208`

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

```
slli x7, x6, 4
```

`0x2345678123456780`

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

```
srli x7, x5, 3
```

`0x0000000015555555`

```
andi x7, x7, 0xFEF
```

`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

addi x8, x5, 512

ld x3, 128(x27)

sd x3, 256(x28)

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

add x3, x0, x0

auipc x3, FFEFA

jal x3 ELSE
```

Instruction	Type	Opcode, func3, func7	rs1	rs2	rd	imm
add x5, x6, x7	R	0x33, 0x0, 0x0	6	7	5	-
addi x8, x5, 512	I	0x13, 0x0, -	5	-	8	512
ld x3, 128 (x27)	I	0x3, 0x3, -	27	-	3	128
sd x3, 256 (x28)	S	0x23m 0x3, -	28	3		256
beq x5, x6 ELSE	SB	0x63, 0x0, -	5	6	-	16
add x3, x10, x0	R	0x33, 0x0, 0x0	0	0	3	-
auipc x3, FFEFA	U	0x17, -, -	-	-	3	FFEFA
jal x3 ELSE	UJ	6F, -, -	-	-	3	16

8 hex characters of above 8 instructions:
1. 0x007302B3
2. 0x20028413
3. 0x080DB183
4. 0x103E3023
5. 0x00628863
6. 0x000001B3
7. 0xFFEFA197
8. 0x010001EF

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
slli x7, x7, 7
and x28, x3, x7
slli x7, x7, 16
xori x7, x7, -1
and x4, x4, x7
slli x28, x28, 16
or x4, x4, x28
```

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

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

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

```
XORI RegD, Reg1, Immed-12
XORI x5, x6 -1
```

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?)

The RISC-V `jal` (jump and link) instruction is used for function calls and jumps to a new address while also saving the return address in the link register (`ra`, x1). The `jal` instruction uses a signed immediate offset to calculate the target address.

The offset is added to the PC to compute the new PC value. The offset is sign-extended and left-shifted by one bit to account for the fact that instructions in RISC-V are 4 bytes (32 bits) long. So, the formula to calculate the new PC is:

New_PC = PC + (offset << 1)

Since the offset is signed, it can be both positive and negative. Therefore, the range of addresses that can be reached using the `jal` instruction from a starting PC value of 0x60000000hex depends on the range of signed immediate values that can be represented in the `jal` instruction.

In RISC-V RV32I (32-bit) architecture, the `jal` instruction uses a 20-bit signed immediate, which means the range of addresses that can be reached is from:

PC + (-(2^20) << 1) to PC + ((2^20 - 1) << 1)

Substituting the PC value you provided (0x60000000hex) into the formula, the range of addresses that can be reached is:

0x60000000 - (2^21) to 0x60000000 + (2^21 - 2)

In hexadecimal notation:

0x5FE00000 to 0x601FFFFE

So, the set of possible values for the PC after the `jal` instruction executes ranges from 0x5FE00000 to 0x601FFFFE.

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 RISC-V `beq` (branch if equal) instruction is used to perform a conditional branch based on whether two registers are equal or not. The branch offset is calculated as a signed immediate value, and it is added to the current PC if the condition is true. Otherwise, the program proceeds to the next instruction.

In RISC-V RV32I (32-bit) architecture, the `beq` instruction uses a 12-bit signed immediate for the branch offset. This means the range of addresses that can be reached is determined by the possible values of the signed immediate.

The branch offset is sign-extended and left-shifted by one bit to account for the fact that

instructions in RISC-V are 4 bytes (32 bits) long. So, the formula to calculate the new PC is:

New_PC = PC + (offset << 1)

Since the offset is signed, it can be both positive and negative. Therefore, the range of addresses that can be reached using the `beq` instruction from a starting PC value of 0x60000000hex depends on the range of signed immediate values that can be represented in the `beq` instruction.

For a 12-bit signed immediate, the range of addresses that can be reached is from:

PC + (-(2^11) << 1) to PC + ((2^11 - 1) << 1)

Substituting the PC value you provided (0x60000000hex) into the formula, the range of addresses that can be reached is:

0x60000000 - (2^12) to 0x60000000 + (2^12 - 2)

In hexadecimal notation:

0x5FFFFE00 to 0x600001FE

So, the set of possible values for the PC after the `beq` instruction executes ranges from 0x5FFFFE00 to 0x600001FE.

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

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?

Since 4 instructions are executed in each loop, 4N instructions would be executed within the loop until the branch is taken. After exiting from the loop, one more instruction corresponding to the DONE label is executed. Total number of instructions executed = 4N + 1

a. 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.

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

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;
```

```
LOOPI:
addi x7, x0, 0 // Init i = 0
bge x7, x5, ENDI // While i < a
addi x30, x10, 0 // x30 = &D
addi x29, x0, 0 // Init j = 0

LOOPJ:
bge x29, x6, ENDJ // While j < b
add x31, x7, x29 // x31 = i+j
sd x31, 0(x30) // D[4*j] = x31
addi x30, x30, 32 // x30 = &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?

The code requires 13 RISC-V instructions. When a = 10 and b = 1, this results in 123 instructions being executed.

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?
0x88

b. What value is stored in **0×10000007** on a littleendian machine?
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, 0x11223

addi x10, x10, 0x344

slli x10, x10, 32

lui x5, 0x55667

addi x5, x5, 0x788

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?

$$128 + x6 > 2^{63} - 1 \Rightarrow x6 > 2^{63} - 129$$

$$128 + x6 < -2^{63} \Rightarrow x6 < -2^{63} - 128 \text{ (Impossible)}$$

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

overflow?

$$128 - x6 > 2^{63} - 1 \Rightarrow x6 < -2^{63} + 129$$

$$128 - x6 < -2^{63} \Rightarrow x6 > 2^{63} + 128 \text{ (Impossible)}$$

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

$$x6 - 128 > 2^{63} - 1 \Rightarrow x6 < 2^{63} + 127 \text{ (Impossible)}$$

$$x6 - 128 < -2^{63} \Rightarrow x6 < -2^{63} + 128$$