NYU Tandon School of Engineering

Fall 2021, ECE 6913

Homework Assignment 2 Solutions

- *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    # i+j
addi x28, x28, 31    # i+j+31
slli x28, x28, 2     # 4*(i+j+31)
add x28, x28, x27    # x28 has &A[i+j+31]
slli x6, x6, 2       # g = 4*g
add x6, x6, x31      # x6 has &C[g]
lw x28, 0(x28)       # read contents of &A[i+j+31] from MEM into
                     # x28
sw x28, 0(x6)        # Write contents of A[i+j+31] into C[g]
```

(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      #x8 has &A[B[9]]
lw x8, 0(x8)         #x8 has A[B[9]]
sub x5, x6, x8
```

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

```
lw x30, 16(x30)      #B[4]
lw x31, 32(x31)      #C[8]
add x24, x30, x31
slli x24, x24, 2
add x4, x27, x24     # &A[C[8] + B[4]]
lw x4, 0(x4)         # A[C[8] + B[4]]
sub x5, x6, x4
```

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

```
add x26, x28, x28    # 2i
addi x26, x26, 1     #2i+1
slli x26, x26,2
add x26, x26, x30    # &B[2i+1]
slli x25, x28, 2
add x25, x25, x27    # &A[i]
lw x24, 0(x26)       # B[2i+1]
sw x24, 0(x25)       #A[i]=B[2i+1]
```

```
addi x26, x26, -4    #B[2i]
slli x23, x28, 2
add x23, x23, x31    #C[i]
```

```
lw x24, 0(x26)
sw x24, 0(x23)      #C[i]=B[2i]
```

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

```
addi x26, x28, -1   # i-1
slli x26, x26, 2
add x26, x26, x30   # &B[i-1]
addi x25, x28, 1    # i+1
slli x25, x25, 2
add x25, x25, x31   # &C[i+1]
slli x24, x28, 2
add x24, x24, x27   # &A[i]
lw x23, 0(x26)      # B[i-1]
slli x23, x23, 2    # 4B[i-1]
lw x22, 0(x25)      # C[i+1]
slli x22, x22, 2    #4C[i+1]
add x23, x23, x22   # 4B+4C
sw x23, 0(x24)      # A
```

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

```
lw x30, 48(x30)     # B[12]
lw x31, 16(x31)     # C[4]
add x24, x30, x31
slli x24, x24, 2
add x4, x27, x24    # &A
lw x4, 0(x4)        # A
sub x5, x6, x4      # 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

x7 = 0x000000000000AAAA

addi x7, x7, -128

0x0000101010101010

1x1111111111110000000

0x00001010101000101010

srai x7, x7, 2

x7 = 0x00000010101010001010

and x7, x7, x6

x6=0x1234567812345678

x7=0x0000000000002A8A

x7 = 10 0000 1000

after AND operation:
```

x7 = 0x0208$_{16}$

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

```
slli x7, x6, 4

x7 = 0x234567812345678016
```

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$_{16}$

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

x7 = 0x0545$_{16}$

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	0x23, 0x3, -	28	3		256
beq x5, x6 ELSE	SB	0x63, 0x0, -	5	6	-	16
add x3, x0, 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;
```

```
A (in register x5) is assigned C[0] (&C[0] in x11) that is left
shifted by 16 bits
lw x5, 0(x11) // x5 is assigned content of C[0]
slli x5, x5, 16 // x5 is shifted left 16 bits
```

(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    // Create bit mask for bits 12 to 7
slli x7, x7, 7       // Shift the masked bits
and x28, x3, x7      // Apply the mask to x3
slli x7, x7, 16      // Shift mask to cover bits 28 to 23
xori x7, x7, -1      // This is a NOT operation
and x4, x4, x7       // "Zero out" positions 28 to 23 of x4
slli x28, x28, 16    // Move selection from x3 into
                     // positions 28 to 23
or x4, x4, x28       // Load bits 28 to 23 from 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]

```
One way to reverse each bit is to apply the xor function
between the register x6 and the number -1 in the immediate
```

12-bit field using the xor immediate instruction: xori.

The number -1 in 12 bits is:

1 equals: 0000 0000 0000 0001

-1 using 2s complement obtained by reversing each bit in

1 and adding 1 to it:

reverse each bit:

1111 1111 1111 1110

add 1

1111 1111 1111 1111

applying the xor function between any binary string and

a string of 1's reverses the binary string because

b xor 1 yields a 0 when b is a 1; yields a 1 when b is 0

Thus:

The xor immediate instruction

XORI RegD, Reg1, Immed-12

XORI x5, x6 -1

will invert the bits in register x6 and write them into

register x5

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?)

- RISC V supports compressed instructions – that are 16 bits long, so instruction addresses point to half words and always end with a '0'
- Offsets for instructions are thus 'shifted' left by one bit with a '0' padded to the right
- This enables a 21-bit equivalent offset as the range (in bytes) or a 19-bit equivalent as the range of (word-aligned) addresses with the last bit always a '0'
- Thus, the jumping range is a 2s complement range of this 21-bit field where the MSB is the sign bit in the 2s

complement representation and the LSB is always 0. The leading 20 bits are identified from the 20 bit immediate field of the J format instruction.

- So, the 2s complement range is $-2^{N-1} \rightarrow +2^{N-1}-1$ if we had 21 bits, but since the LSB is always 0, the range is reduced to -2N-1 → +2N-1 – 2
- The **maximum positive number** has a leading bit (bit 20) of '0' followed by 19 '1's with bit 21 padded at the right end as a '0' or 0 1111 1111 1111 1111 1110 which is **'0FFFFE'**
- The **largest negative number** has a leading bit of '1' followed by 20 '0's (including the padded '0' at the right end) or 1 0000 0000 0000 0000 0000 which (in hex) is **'10 0000'**

<u>to get the upper boundary of the range:</u>

0110 0000 0000 0000 0000 0000 0000 0000 +

0000 0000 000**0 1111 1111 1111 1111 1110**

0110 0000 0000 **1111 1111 1111 1111 1110**

----

or 6**00FFFFE** in hex

<u>to get the lower boundary of the range:</u>

(of a **jal** instruction with PC content at **60000000**)

0110 0000 0000 0000 0000 0000 0000 0000 +

1111 1111 1111 0000 0000 0000 0000 0000

0**101 1111 1111** 0000 0000 0000 0000 0000

or **5FF00000** in hex

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?)

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

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;

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**22334455667788.

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

`Data in 0×10000007 is 0x88`

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

`Data in 0×10000007 is 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          // loads the 5 hex numbers 11223 into
                          the upper 20 bits of x10

                              // note that each hex number
                          corresponds to 4 bits, so loading the upper
                          20 bits with lui can only permit exactly 5
                          hex numbers. Hence the choice of '11223'
                          into the upper 20 bits

addi x10, x10, 0x344      // we can extend the above string of
                          11223 by adding the immediate value of 3 hex
                          numbers of '344' in the lower 12 bits to the
                          above string of '11223' to get '11223344'
                          into x10

slli x10, x10, 32         // moves the 8 hex numbers '11223344'
                          to the upper half of the 64 bit double word

lui x5, 0x55667           // loads the MSBs of the lower half of
                          the desired double word with the 5 hex
                          numbers '55667'

addi x5, x5, 0x788        // just as we did previously, these 5
                          hex numbers are added to 3 hex numbers '788'
                          to produce the lower half string: '55667788'

add x10, x10, x5          // this added to the upper half string
                          yields the full desired string of
                          '1122334455667788' loaded into the double
                          word
```

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?

There is an overflow if $128 + x6 > 2^{63} - 1$ In other words, if $x6 > 2^{63} - 129$

There is also an overflow if $128 + x6 < -2^{63}$ that is, if $x6 < -2^{63} - 128$ (which is impossible given the range of x6)

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

There is an overflow if $128 - x6 > 2^{63} - 1$

In other words, if $x6 < -2^{63} + 129$ There is also an overflow if $128 - x6 < -2^{63}$ In other words,

if $x6 > 2^{63} + 128$ (which is impossible given the range of x6).

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

There is an overflow if $x6 - 128 > 2^{63} - 1$ In other words, if $x6 < 2^{63} + 127$ (which is impossible given the range of x6) There is also an overflow if $x6 - 128 < -2^{63}$ In other words, if $x6 < -2^{63} + 128$