Introduction to Computer Organization – Fall 2024

Homework 1

Due: September 23rd @ 11:55 pm on Gradescope

Name: ______Qiulin Fan__________ Uniqname: _______rynnefan_______

1. Submit a pdf of your typed or handwritten homework on Gradescope.

2. Your answers should be neat, clearly marked, and concise. Typed work is recommended, but not required unless otherwise stated. Show all your work where requested, and state any special or non-obvious assumptions you make.

3. You may discuss your solution methods with other students, but the solutions you submit must be your own.

4. **Late Homework Policy:** Submissions turned in by 1:00 am the next day will be accepted but with a 5% penalty. Assignments turned in between 1:00 am and 11:55 pm will get a 30% penalty, and any submissions made after this time will not be accepted.

5. When submitting your answers to Gradescope you need to indicate what page(s) each problem is on to receive credit. The grader may choose not to grade the homework if answer locations are not indicated.

6. After each question (or in some cases question part), we've indicated which lecture number we expect to cover the relevant material. So "**(L7)**" indicates that we expect to cover the material in lecture 7.

7. **The last question is a group question**.
    - You may do it in a homework group of up to two students including yourself (yes, you can do them by yourself if you wish).
    - If you work in a group of two for these questions, list the name of the student you worked with in your assignment. Further, we suggest that you not split these problems up but rather work on the problem as a group.
    - Turn these group questions in as part of your individual submission.
    - **For these questions (and these questions ONLY) you are allowed to copy/paste solutions from the other student in your homework group.**
    - It is an honor code violation if a student is listed as contributing who did not actually participate in working on that problem.

Problem 1 (20 points): Unsigned binary numbers and shifting **(L2)**

1. Convert the following (unsigned) binary numbers to decimal. **[4]**
   a. 0b10110

      ________22__________

   b. 0b001001

      ________9____________

2. Convert the following decimal numbers to 8-bit binary. **[6]**
   a. 6

      0b 0000 0110_____

   b. 20

      0b 0001 0100___

   c. 110

      0b 0110 1110____

3. Answer the following (EDIT: assume all binary numbers are unsigned):
   a. What is the decimal representation of the binary number 0b110? **[2]** _6____

   b. What is the decimal representation of the binary number 0b1100? **[2]** _12___

   c. In general, what happens to the value of a binary number when you add a zero to the end of it (least significant)? **[3]**

      Its value is doubled.

   d. When you add 4 zeros to the end? **[3]**

      Its value is multiplied by 2^4 = 16

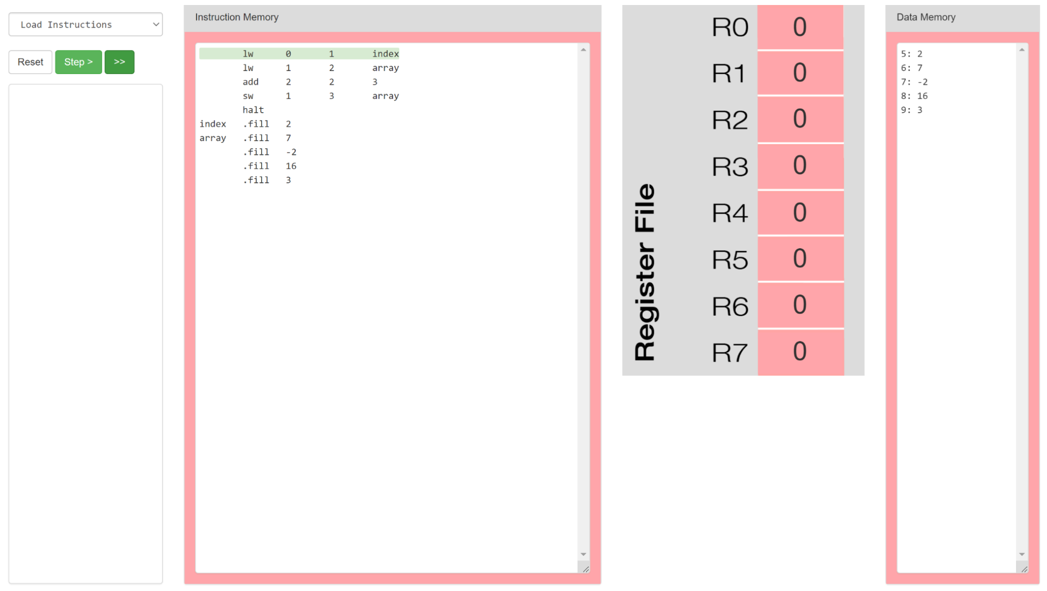
Problem 2 (20 points): Reading LC2K Code **(L3)**

This problem will get you some exposure into the correspondence between C code and assembly, specifically the LC2K instruction set architecture.

Copy-paste the following code into the "Instruction Memory" window of the LC2K simulator linked on the website:

```
        lw      0       1       index
        lw      1       2       array
        add     2       2       3
        sw      1       3       array
        halt
index   .fill   2
array   .fill   7
        .fill   -2
        .fill   16
        .fill   3
```

Click "verify" and you should see the following window, showing the initial values of the register file (all zero), and the contents of data memory (initialized to the values specified by .fill directives in the program). (*Note, if you get an error at this step, try copy-pasting the above code into a plain text editor, and then copying it into the instruction window. Make sure to include the leading white space on the first line.)

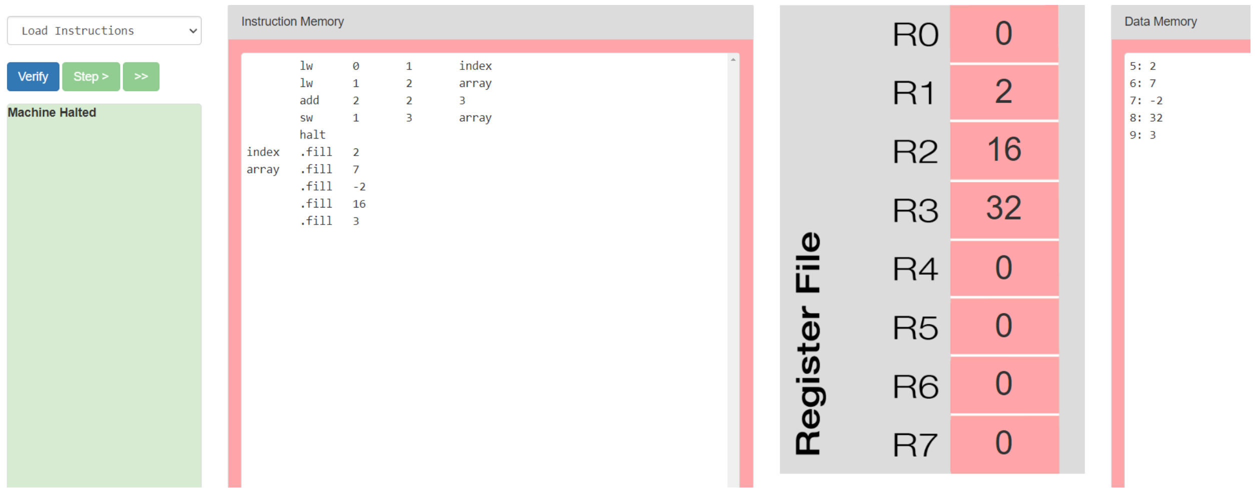


```
        lw      0       1       index
        lw      1       2       array
        add     2       2       3
        sw      1       3       array
        halt
index   .fill   2
array   .fill   7
        .fill   -2
        .fill   16
        .fill   3
```

R0	0
R1	0
R2	0
R3	0
R4	0
R5	0
R6	0
R7	0

```
5: 2
6: 7
7: -2
8: 16
9: 3
```

You can now click "Step >" to see the effect of each instruction executing. You should see either one of the register values changing or one of the lines in data memory changing after each instruction



```
        lw      0       1       index
        lw      1       2       array
        add     2       2       3
        sw      1       3       array
        halt
index   .fill   2
array   .fill   7
        .fill   -2
        .fill   16
        .fill   3
```

R0	0
R1	2
R2	16
R3	32
R4	0
R5	0
R6	0
R7	0

```
5: 2
6: 7
7: -2
8: 32
9: 3
```

1. Explain, line-by-line, what this program is doing with each instruction executed (you don't need to explain the .fill directives, those aren't executed instructions. They just initialize the contents of data memory when we load the program). It should be clear from your explanation what each operand listed is doing. **[10]**

   Line 1: load R1 = mem[value[R0] + index=5]. (the value of R0 is always 0, so the address loaded to R1 is mem[5] = 2)
   Line 2: load R2 = mem[value[R1](=2) + array(=6)] = mem[8] = 16
   Line 3: value[R3] = value[R2] + value[R2] = 16 + 16 = 32
   Line 4: save mem[value[R1](=2) + array(=6)] = mem[8] by value[R3] = 32.
   Line 5: halt

2. Write a C program that has the same functionality as the assembly code above. Assume all values are signed integers, and that values initialized in the assembly code using .fill directives are all initialized as global variables with the same labels used in the assembly program. **[10]**

   int reg[8];
   int index = 2;
   int array[4] = {7, -2, 16, 3};

   int main() {
   reg[1] = index;
   reg[2] = array[reg[1]];
   reg[3] = reg[2] *2;

```
    array[reg[1]]  = reg[3];
    return 0;
}
```

(or a shorter version below)

```
int index = 2;
int array[4] = {7, -2, 16, 3};

int main() {
    array[index]  *= 2;
    return 0;
}
```

Problem 3 (20 Points): Moving data between memory and registers (L5)

For the following LEGv8 assembly code, indicate the final state of the registers and memory locations listed once this code has finished. Assume any memory address outside of that displayed in the initial memory state contains 0.

```
MOVZ        X0, #0x1001
LDURH       X2, [X4, #0]
STURB       X3, [X0, #3]
LDURSW      X4, [X0, #-1]
STURH       X4, [X0, #1]
```

Initial Register State		**Initial Memory State**	
Register	Value	Address	Value
X0	0x5	0x00001000	0x4
X1	0x1000	0x00001001	0x12
X2	0x6E	0x00001002	0x33
X3	0x41FF	0x00001003	0x2E
X4	0x1002	0x00001004	0xA1
X5	0x0	0x00001005	0x77

1) Fill in the following table. Express your final answers in hexadecimal. Assume memory is organized in a **little endian** manner. **[10]**

X0 = 0x1001
X2 = mem[X4] = mem[0x1002 ~ 1003] = 0x2E33 (by little endian)
Mem[X0 + 3] (= Mem[0x1004]) = X3 (FF)
    Little endian: sig bit at front, so 0x1004 = FF
X4 = mem[X0 -1] = mem[0x1000] (by 4 bytes),
    So 0x2E331204
mem[X0 + 1] (= mem[0x1002 ~ 1003]) = X4 (by 2 bytes, 1204)
    So 0x1002 = 04, 0x1003 = 12

Final Register State		**Final Memory State**	
Register	Value	Address	Value
X0	0x1001	0x00001000	0x4
X1	0x1000	0x00001001	0x12
X2	0x2E33	0x00001002	0x04
X3	0x41FF	0x00001003	0x12
X4	0x2E331204	0x00001004	0xFF
X5	0x0	0x00001005	0x77

2) Fill in the following table. Express your final answers in hexadecimal. Assume memory is organized in a **big endian** manner. **[10]**

X0 = 0x1001
X2 = mem[X4] = mem[0x1002 ~ 1003] = 0x332E (by big endian)
Mem[X0 + 3] (= Mem[0x1004]) = X3 (FF)
    Little endian: sig bit at front, so 0x1004 = FF
X4 = mem[X0 -1] = mem[0x1000] (by 4 bytes),
    So 0x0412332E
mem[X0 + 1] (= mem[0x1002 ~ 1003]) = X4 (by 2 bytes, 332E)
    So 0x1002 = 33, 0x1003 = 2E

Final Register State		**Final Memory State**	
Register	Value	Address	Value
X0	0x1001	0x00001000	0x4
X1	0x1000	0x00001001	0x12

X2	0x332E	0x00001002	0x33
X3	0x41FF	0x00001003	0x2E
X4	0x0412332E	0x00001004	0xFF
X5	0x0	0x00001005	0x77

Problem 4: LEGv8 (20 points)--Place your answers in the appropriate box. (L6)

Variable to register mappings

```
int64_t t - X2        int64_t v - X4        int64_t x - X7
int64_t u - X3        int64_t w - X5        int64_t y - X8
```

1) Convert the following C code to no more than 4 lines of equivalent LEGv8 assembly, assume `int64_t arr[]`'s base address is held in `X6`. Additionally use register `X1` as a temporary register. Recall that unlike LC2K, the LEGv8 ISA is byte addressed. **[10]**

```
w = arr[x];
arr[x+7] = y;
```

```
LSL  X1, X7, #3
LDUR X5, [X6, X1]
ADDI X1, X1, #56
SDUR X8, [X6, X1]
```

2) Convert the following C code to no more than 5 lines of LEGv8 assembly. Please use **CBZ** or **CBNZ** for your conditional instruction. **[10]**

```
while(x != 0){
      x--;
      y+=x;
}
```

```
loop: CBZ   X7, end
      SUBIS X7, X7, #1
      ADD   X8, X8, X7
      CBNZ  X7, loop
end:
```

Problem 5 (20 points, group): Transistor Performance (L1)

You must **type** the answer to these questions. Read about Moore's Law and Dennard Scaling on Wikipedia (or some other source if you wish) then answer the following questions. Each answer should be under 100 words.

1. Define Moore's Law in your own words.

   Moore's Law states that the number of transistors on a microprocessor doubles about every two years, indicating an exponential growth in the density of transistors on integrated circuits over time.
   Moore's Law has broken because of physical constraints, as we cannot make transistors unboundedly small.

2. Define Dennard Scaling in your own words.

   Dennard Scaling states that as transistors get smaller, the power density remains constant, because the voltage and current scale down with the size. This scaling implies performance improves while power consumption stays the same.
   Dennard Scaling has broken since small transistors leak power, causing chips to heat up.

3. Imagine you have a microprocessor that follows Dennard Scaling perfectly. If you make the chip have twice as many transistors of the same size (thus doubling its area), *roughly* what would you expect would happen to the performance and power consumption of the microprocessor? (We are looking for things like "go up", "go down" "stay the same" etc.) Briefly explain your reasoning.

   Performance would go up because doubling the number of transistors usually increases processing power, allowing for more computations to occur simultaneously.
   Power consumption would stay the same because Dennard Scaling (ideally) ensures that the increased number of transistors would not increase the overall power consumption, as voltage and current scale down proportionally to offset the increase in transistor count.

4. Say that the power consumed by a single transistor is constant and you halve the number of transistors on the chip while keeping the area the same (i.e., each transistor becomes bigger). *Roughly* what would you expect would happen to the performance and

power consumption of the microprocessor? (We are looking for things like “go up”, “go down” “stay the same” etc.) Briefly explain your reasoning.

Performance would go down as fewer transistors weaken computational capacity. Power consumption would go down since the power consumed by a single transistor is constant and there are fewer transistors, so the total power goes down about half.