QUESTION 1: YOU ARE GIVEN THE FOLLOWING MACHINE CODE:

11111000010 000000101 00 00001 01111
10110100 1111111111111111 01111
11010011010 00000 000010 01111 01111
10001011000 01111 000000 00001 00001

• Convert the machine code to Assembly
• Draw the constraint graph for the above Assembly code (include hazards such as RAW, WAW, WAR, and Control Hazards). Assume no delay slots.
• Adjust the code (if possible), to run as Fast as possible on a pipelined processor like that of lab #4.
Please do the constrain graph and put it onto the 2-way Superscalar version of Lab4 CPU (delay slot), both can do ALU and only one can do branch, the other one can do Load/Store

0: LDUR X0, [X1, #1]
1: ADDI X1, X0, #1
2: LDUR X3, [X1, #1]
3: STUR X4, [X2, #1]
4: CBNZ X1 I_LOVE_VLSI
6: ADDI X5, X4, #0
7: BR END
I_LOVE_VLSI:
8: ADDI X5 X4, #1;

END:
For the following two sequence of memory read accesses (byte addressed memory) and cache parameters given, determine the best remaining parameter to find the lowest miss rate. Assume that this memory system only has a L1 cache. Block size must be a power of 2. Access addresses are in hexadecimal.

1. Access: 0x00, 0x08, 0x10, 0x18, 0x20, 0x28, 0x30, 0x38
   Total Cache Size = 32 bytes
   Associativity = 1 (direct mapped)
   Block Size = __________
   Miss Rate based on the above parameter = __________

2. Access: 0x00, 0x40, 0x20, 0x10, 0x02, 0x44, 0x23, 0x15
   Total cache size = 32 bytes
   Associativity = __________
   Block Size = 8 bytes
   Miss Rate based on the above parameter = __________
When you first start your system, you notice there are a lot of cache misses. What is most likely the cause, why do they occur, and can anything be done to prevent this? If not why?
Describe what is the forwarding unit doing for the following instructions in each cycle:

**ADDI X1, X1, #20**

**STUR X1, [X2, #4]**

**LDUR X2, [X1, #8]**

**SUBS X3, X1, X2**

**B.LT END**

**ADD X4, X1, X3**

<table>
<thead>
<tr>
<th>Ifetch</th>
<th>Reg</th>
<th>Exec</th>
<th>Mem</th>
<th>Wr</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ifetch</td>
<td>Reg</td>
<td>Exec</td>
<td>Mem</td>
<td>Wr</td>
</tr>
<tr>
<td>Ifetch</td>
<td>Reg</td>
<td>Exec</td>
<td>Mem</td>
<td>Wr</td>
</tr>
<tr>
<td>Ifetch</td>
<td>Reg</td>
<td>Exec</td>
<td>Mem</td>
<td>Wr</td>
</tr>
<tr>
<td>Ifetch</td>
<td>Reg</td>
<td>Exec</td>
<td>Mem</td>
<td>Wr</td>
</tr>
<tr>
<td>Ifetch</td>
<td>Reg</td>
<td>Exec</td>
<td>Mem</td>
<td>Wr</td>
</tr>
</tbody>
</table>
For a two-way associative cache with 128KB of data and 16-byte blocks, how many total bits are required? (Assume 32-bit addresses).
Homework 8a: Question for the Final

Write the following code into machine language and draw a constraint graph.

//returns the power of two that N is (assuming it’s a power of 2, otherwise the closest power to N)

int powerOfTwo(int N) {
    if (N > 2) { return (powerOfTwo(N/2) +1); }
    else { return 1; }
}
Create a constraint graph for the following program and schedule it for a 4-way VLIW (allowed 1 memory unit and 1 branch unit). Assume all instructions parallel to a branch execute and no delay slots.

1: ADDI X0, X1, #3
2: LDUR X2, [X1, #8]
3: STUR X1, [X0, #2]
4: CMP X2, X1
5: B.EQ  2
6: ADD X2, X1, X31
7: STUR X2, [X2, #0]
8: AND X1, X0, X1

<table>
<thead>
<tr>
<th>ALU1</th>
<th>ALU1</th>
<th>Mem</th>
<th>Branch</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Schedule this code on lab 4 processor

Loop:
LDRR X0, [X1, #0]
ADD X1, X2, X3
SUBS X2, X3, X2
ADDT X0, X1, X0
B.LT END
B Loop

END:

DRAW constraint graph first
The following cache is set up as:

<table>
<thead>
<tr>
<th>Level</th>
<th>Hit time</th>
<th>Hit Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1</td>
<td>1 cycle</td>
<td>80%</td>
</tr>
<tr>
<td>Main Memory</td>
<td>400 cycles</td>
<td>85%</td>
</tr>
<tr>
<td>Disk</td>
<td>25,000 cycles</td>
<td>100%</td>
</tr>
</tbody>
</table>

Would it be more beneficial to reduce the overall access time of the system by adding an L2 cache with a hit time of 25 cycles and hit rate of 81%, or to increase the hit rate of main memory to 90%?
For the code below, would a 1-bit predictor or 2-bit predictor be more accurate at predicting the "if (i%3==0)" branch? Determine the branch prediction accuracy of both.

```c
int count = 0;
while (count < 100) {
    if (count % 2 == 0) {
        for (int i = 0; i < 10; i++) {
            if (i % 3 == 0) {
                count++;
            }
        }
    }
    count++;
}
```
Translate the following assembly code to a C function; you may create your own variables whenever appropriate, you may assume the function has no input parameters.

```
ADDI $x0, $x21, #0
ADD $x1, $x21, #800
ADD $x3, $x21, #4

LOOP:
    LDAURB $x2, [$x0, #0]
    ADD $x2, $x2, $x3
    STAURB $x2, [$x0, #0]
    ADDI $x0, $x0, #1
    CMP $x0, $x1
    B.LS LOOP
```
Calculate the CPI on a pipelined processor that wastes 1 cycle after every branch and 1 cycle after every load. Note that during this time no new instructions can be issued. The program has 20% branches and 25% loads.

(Or)

If an instruction requires a clock period of 100ns, how long would that instruction take to be executed on a 5 stage pipelined processor?
LLPex Plus ( char * src )
    char * end = src + 1000;
    while ( src < end )
        if ( *(src) == 'a' )
            *(src) = 'A';

            src ++ ;

Create a 4x unrolled version of the code with compiler register renaming.
Draw the constraint graph for the following code.

```
LDUR  x0 , [x1,#0]
ADD   x1 , x0 , x1
SUBI  x3 , x2 , #3
STUR  x1 , [x3,#0]
LDUR  x1 , [x4,#0]
ADD   x2 , x0 , x0
ADD   x3 , x0 , x1
STUR  x2 , [x1,#0]
```
We are writing to the following addresses:

<table>
<thead>
<tr>
<th>address</th>
<th>hit time (cycles)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>24</td>
<td>14</td>
</tr>
<tr>
<td>16</td>
<td>4</td>
</tr>
<tr>
<td>32</td>
<td>100</td>
</tr>
</tbody>
</table>

Assuming blocks are at least 8 blocks long and there is no buffer, what is the block size of L1 cache, L2 cache? Is there an L3? (Yes/No/can't tell)
Design a five-stage pipelined CPU that can do LSL and LSR. Determine the control signal in each stage.
Write a program in assembly for the pipelined CPU discussed in class that adds the first five prime numbers, and stores the result in XO. The CPU has branch and load delay slots, forwarding logic, and accelerated branches.
The C function given below takes an array of integers, where each integer appears twice except for one, and returns the integer that appears once.

```c
// numSize = length of nums array
int singleNumber(int* nums, int numSize) {
    int number = 0;
    for (int i = 0; i < numSize; i++) {
        number = number ^ nums[i];
    }
    return number;
}
```

A. Create a 3x unrolled version of the code (call it singleNumber3), with the compiler register renaming. Assume the length of the array is a multiple of 3.

B. Draw the constraint graph for singleNumber3 and for each edge, indicate the cause of the constraint.
For the following C code find the percent correct for each branch prediction (excluding white stalls).

a. use a 1-bit predictor
b. use a 2-bit predictor

Your final answer should have 6 percentages.

```c
void main() {
    int i, num;
    while(1) {
        num = foo();
        if (num == NULL) return;
        for (i=0; i<9; i++)
            if (num % 2 == 0) num = 0;

        return;
}

int foo() { return rand(); }
```
If you are given a cache that has 2 way set associative, 4 byte blocks, total size 32 bytes, with LRU replacement policy.

Draw the cache diagram and calculate the miss rate and write the corresponding miss type.

<table>
<thead>
<tr>
<th>Byte Addr</th>
<th>Block Addr</th>
<th>Hit/Miss</th>
<th>Miss Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>52</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>52</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Miss Rate: _____________
DM, 4 byte blocks, 4 indexes

<table>
<thead>
<tr>
<th>Cl</th>
<th>4 byte blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

2-way set, 4 byte blocks, 2 indexes

<table>
<thead>
<tr>
<th>Cl</th>
<th>4 byte blocks</th>
<th>Cl</th>
<th>4 byte blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Derive an access pattern (in 6 accesses) where the Direct Mapped cache will have less misses than the 2-way Set Associative cache.
What will be the final value in mem[0] and mem[8] if the processor (a) has a forwarding unit, or (b) does not have a forwarding unit?

(processor is the Lab#4 pipelined CPU)

```
ADD   x0, x31, x31
ADDI  x0, x31, #60
ADDI  x1, x31, #480
ADDI  x0, x0, #400
ADDI  x0, x0, #9
SUB   x1, x1, x0
SUBI  x1, x1, #480
STUR  x0 [x31, #0]
STUR  x1, [x31, #8]
```