Architecture

1. Consider the following loop nest:

```
float x[1024], out[1024], coeff[3];
for (i=0;i<n;i++) {
  out[i] = 0;
  for (j=0;j<3;j++)
    out[i] = out[i] * x[i] + coeff[j];
}
```

a. Is the inner loop parallel? Why or why not?

b. Unroll the inner loop (completely) and convert the loop nest to MIPS assembly.

c. Assume the resulting code (outer loop and unrolled inner loop) is executed on a single issue, in-order 5-stage processor pipeline. Assume the processor is fully pipelined with data forwarding. Also assume loads and stores have a fixed latency of 2 cycles (in the memory stage), integer instructions a fixed latency of 1 cycle (in the execute stage), and floating point instructions a fixed latency of 3 cycles (in the execute stage). In other words, the execute and memory "stages" have potentially multiple cycles of latency. Estimate the resulting CPI.

d. Schedule the unrolled inner loop to minimize the impact of data dependencies and repeat part c.

2. Consider a memory system that has the following characteristics.

memory	type	line size	miss rate	write policy	% dirty	access time	bandwidth to lower level
L1	split	I: 64 bytes D: 16 bytes	I: 2% D: 7%	D: write-back		0	6.4 GB/s
L2	unified	64 bytes	2%	write back	50%	10 ns	3.2 GB/s
L3	unified	2 KB	1%	write back	60%	50 ns	1.6 GB/s
RAM						400 ns	

Assume the base instruction CPI is 1. For a particular program under test, assume that 15% of all executed instructions are loads and 5% of all executed instructions are stores. Calculate the overall CPI. State any assumptions.

3. Consider the following loop:

```
float img[1024*1024];
for (i=0;i<h;i++)
  for (j=0;j<w;j++) {
    sum=0;
    for (k=0;k<5;k++)
      sum+=img[(i+k)*w+j];
  }
```

a. What is the arithmetic intensity (ops per byte) of this loop?

b. Assume this loop nest is executed on a processor with a peak floating point throughout of 16 Gflops and a peak memory bandwidth of 12 GB/s. Is this loop memory bound or compute bound? Why?

c. Assume the processor supports 4-way SIMD instructions, but only when data is accessed from consecutive memory locations. Would such instructions benefit this loop? If not, is it possible to change to the loop in order to effectively utilize these instructions?

Compilers

1. Suppose we want to build expressions out of **VAR** and **CONST** tokens with the following allowed operators (eight in all):

Operators	**Type**	**Associativity**
^, $	unary postfix	left to right
@, #	binary infix	left to right
*, &	unary prefix	right to left
<	binary infix	none
%, =	binary infix	right to left

   Groups of operators have the same type, precedence, and associativity. Groups are listed in order of decreasing precedence. Expressions can also include parentheses to coerce evaluation order as usual.

   Give the rules section of a yacc (bison) source file to build an LALR parser for these expressions. Use `expr` as the start symbol. Your grammar should reflect the precedence and associativity of the operators as closely as possible. No semantic actions are required. [Your yacc syntax need not be exactly correct as long as it is close.]

2. Provide semantic actions for the middle break loop, which has the following syntax:

$$S \rightarrow \text{loop } S_1 \text{ until } B \; S_2 \text{ endloop ;}$$

   When the loop is entered, S_1 is executed before the termination test B. If B evaluates to false the loop exits. If not, execution proceeds with S_2 and then with S_1 before testing B again.

3. The following fragment of 3-address code was produced by a non-optimizing compiler:

```
 1  start:  x := 1
 2          y := 1
 3          sum := x
 4          sum := y
 5  loop:   if x = n then goto out1
 6          t1 := y
 7          t2 := t1 * t1
 8          t3 := t1 + x
 9          if t3 < n then goto skip
10          t3 := t3 - n
11  skip:   sum := sum + t3
12          y := y + 1
13          goto loop
14  out2:   x := sum
15          sum := x + 1
16          x := x + 1
17  out1:   x := y - 1
18          t1 := x
19          print t1
20          if x = 0 then goto out
21          goto out2
22          goto out1
23  out:    no_op
```

Assume that there are no entry points into the code from outside other than at `start`.

(a) Decompose the code into basic blocks $B_1, B_2, \ldots$, giving a range of line numbers for each.

(b) Draw the control flow graph, and describe any unreachable code.

(c) List which variables are live immediately before line 8 and which variables are live immediately before line 18. Treat the array `a` as a single variable. Assume that `n` and `sum` are the only live variables immediately after line 23.

(d) Describe any simplifying transformations that can be performed on the code (i.e., transformations that preserve the semantics but reduce (i) the complexity of an instruction, (ii) the number of instructions, (iii) the number of branches, or (iv) the number of variables).

Algorithms

1. **Find** a tight asymptotic upper bound on the function $T(n)$ defined by the following recurrence:

$$T(n) = 2T\left(\frac{n}{3} + 8\right) + 5n$$

Prove, using the substitution method, that your answer is correct. You are only required to show the inductive step; you need not show how base cases are handled.

2. Leo decides to send messages using a simple code in which each letter is encoded by its (zero-based) index in the alphabet. The table below shows a few examples.

Letter	**Code**
a	0
b	1
c	2
⋮	⋮
h	7
⋮	⋮
z	25

Unfortunately, Leo forgot to include delimiters between the code numbers, and as a result, his messages are ambiguous. For example, the message '24' might be one letter ('y') or two ('ce'). Likewise, the message '111' might be 'bbb', 'bl', or 'lb'. The message '1920171911418' has 96 different decodings, including the strings 'bjcabhbjbos', 'tcartlebi', and 'turtles'.

Describe, in pseudocode, a $\Theta(n)$ time algorithm that takes as input a string of length n encoded in this way, and outputs the number of distinct decodings of that string. (Your algorithm only needs to output the *number* of decodings; it does not need to generate the decoded strings themselves.) **Explain** why your algorithm works, in enough detail to convince an intelligent but skeptical reader that it is correct.

3. Let T_1 and T_2 denote two binary search trees that each contain the same set of n distinct keys, but whose shapes may be distinct. **Prove** that there exists a sequence of $\Theta(n)$ rotation operations that transforms T_1 into T_2.