Code used to the first Four problems (labeled appropriately below).

```
#define __STDC_FORMAT_MACROS
#include <assert.h>
#include <inttypes.h>
#include <math.h>
#include <stdbool.h>
#include <stddef.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <strings.h>

#define CUR_TEST 14

inline static int64_t max(int64_t x, int64_t y) {
  return x > y ? x : y;
}

void simulate(FILE* inputFile, FILE* outputFile)
{
  // See the documentation to understand what these variables mean.
  int32_t microOpCount;
  uint64_t instructionAddress;
  int32_t sourceRegister1;
  int32_t sourceRegister2;
  int32_t destinationRegister;
  char conditionRegister;
  char TNnotBranch;
  char loadStore;
  int64_t immediate;
  uint64_t addressForMemoryOp;
  uint64_t fallthroughPC;
  uint64_t targetAddressTakenBranch;
  char macroOperation[12];
  char microOperation[23];

  int64_t totalMicroops = 0;
  int64_t totalMacroops = 0;

  // Vars for added code
  int64_t insn_total_len = 0;
  int64_t max_branch_offset = 0;
  int64_t total_branch = 0;
  int64_t gone_over = 0;
  int64_t total_extra_ops = 0;
  int64_t load = 0;
  int64_t store = 0;
  int64_t uncond = 0;
  int64_t cond = 0;
  int64_t other = 0;

  fprintf(outputFile, "Processing trace...\n");

  while (true) {
    int result = fscanf(inputFile,
                        "%" SCNi32
                        "%" SCNx64
                        "%" SCNi32
                        "%" SCNi32
                        "%" SCNi32
                        " %c"
                       " %c"
                        " %c"
                        "%" SCNi64
                        "%" SCNx64
                        "%" SCNx64
                        "%" SCNx64
                        "%11s"
                        "%22s",
                        &microOpCount,
                        &instructionAddress,
                        &sourceRegister1,
                        &sourceRegister2,
                        &destinationRegister,
                        &conditionRegister,
                        &TNnotBranch,
                        &loadStore,
                        &immediate,
                        &addressForMemoryOp,
                        &fallthroughPC,
                        &targetAddressTakenBranch,
                        macroOperation,
                        microOperation);

:
    if (result == EOF) {
      break;
    }

    if (result != 14) {
      fprintf(stderr, "Error parsing trace at line %" PRIi64 "\n", totalMicroops);
      abort();
    }

    if (targetAddressTakenBranch != 0) {
      total_branch++;
```

QUESTION 2

```
      if((int)(2 + floor(log10(abs(instructionAddress - targetAddressTakenBranch))/log10(2))) >
CUR_TEST){
        gone_over++;
      }
      max_branch_offset =
        max(abs(instructionAddress - targetAddressTakenBranch), max_branch_offset);
```

QUESTION 3

```
      if((int)(2 + floor(log10(abs(instructionAddress - targetAddressTakenBranch))/log10(2))) > 8){
        total_extra_ops++;
      }
```

QUESTION 4

```
      if (conditionRegister == '-') uncond++;
      else if (conditionRegister == 'R') cond++;
    } else if (loadStore == 'L') load++;
    else if (loadStore == 'S') store++;
    else other++;

    // For each micro-op
    totalMicroops++;
    // For each macro-op:
    if (microOpCount == 1) {
      totalMacroops++;
```

QUESTION 1

```
      insn_total_len += fallthroughPC - instructionAddress;
    }
  }

  fprintf(outputFile, "Average instruction length: %f\n", (insn_total_len + 0.0) / totalMacroops); // Q1
  fprintf(outputFile, "Max branch offset: %" PRIi64 "\n", max_branch_offset); // Q2
  fprintf(outputFile, "Max branch offset bits: %d\n", (int)(1 +
floor(log10(max_branch_offset)/log10(2)))); // Q2 part 1
  fprintf(outputFile, "Cur test: %d, percent over: %f\n %d\n", CUR_TEST, (gone_over + 0.0) /
total_branch); // Q2
  fprintf(outputFile, "Added ops for 8 bit branch: %f\n", (total_extra_ops + 0.0) / totalMicroops); // Q3
  fprintf(outputFile, "Counts: Load - %" PRIi64 " Store - %" PRIi64
                    " UnCond - %" PRIi64 " Cond - %" PRIi64
                    " Other - %" PRIi64 "\n",
                    load, store, uncond, cond, other);

  fprintf(outputFile, "Processed %" PRIi64 " trace records.\n", totalMicroops);

  fprintf(outputFile, "Micro-ops: %" PRIi64 "\n", totalMicroops);
  fprintf(outputFile, "Macro-ops: %" PRIi64 "\n", totalMacroops);

}
```

QUESTION 18

CPI = (seconds * cycles/ second) / total instructions

CPI = (7.94 * 3.4 x 10 ^ 10) / (200,000 * 458,752)

QUESTION 19, 20 Used similar approach