**CSI 402 – Systems Programming – Spring 2013**

**Midterm Examination – March 12, 2013**

**Note:** This examination has **five** questions for a total of 100 points. **Answer all questions. Write all the answers on your blue book.**

**Question I** (10 points total)

Provide *short* answers to the following questions.

(a) In Sun SPARC architecture, each instruction is only 32 bits long. However, memory addresses are 64 bits long. How can an address be specified in an instruction? (2 points)

(b) What is meant by the phrase "relocatable program"? (2 points)

(c) In which pass of a conventional 2-pass assembler are multiply-defined symbols detected? Why? (3 points)

(d) The file `main.c` of a C program contains the statement

```
#include  "struct_def.h"
```

However, in the `makefile`, the dependency line for `main.o` does not contain `struct_def.h`. Indicate what can go wrong in this situation. (3 points)

**Question II** (35 points total)

(a) Consider the following 3-byte instruction for SIC/XE:

```
LDA   #-16
```

The 6-bit opcode for `LDA` is `000000`. Show the assembled form of the above instruction. *Your answer must be in hexadecimal form.* Show work. (11 points)

(b) The assembled form of the 3-byte SIC/XE instruction

```
J   NEXT
```

is `F02009` (hex). The LC-value of the above instruction is 300 (decimal). Find the address of the symbol `NEXT`. *Your answer must be in decimal.* Show work. (11 points)

(c) Consider the following six symbols: `AIM`, `BAT`, `CARE`, `DATA`, `EVAL` and `FINAL`. Show a binary search tree of height *exactly* 4 for this set of symbols. (6 points)

(d) An assembler uses move-ahead-2 heuristic to organize its symbol table. At some point, the list of symbols is as follows:

```
F   E   D   C   B   A
```

Show the list that results at the end of the access sequence `A C D E`. (6 points)

**Question III** (20 points total – 10 points for each part)

(a) A C program has been split into two source files called `main.c` and `funct.c`. The contents of these two files are shown below.

**File:** `main.c`

```
#include <stdio.h>
int p = -31;  char q = 'x';
void mystery(void);
int main(void) {
  printf("%d   %c\n", p, q); mystery();
  printf("%d   %c\n", p, q);
  return 0;
}
```

**File:** `funct.c`

```
extern int p;  extern char q;
void mystery(void) {
  int q;
  p = -11; q = 'y';  return;
}
```

There are no syntax errors in either of the above C files. The executable version (`a.out`) of the program is created using the following Unix command:

```
gcc  main.c  funct.c
```

Indicate the output produced when `a.out` is executed. **No explanation is needed.**

(b) Consider the following module written in SIC/XE assembly language.

```
MAIN     START    0
         +LDA     P1
         STA      BUF
         +SUB     P2
         STA      BUF+6
         +STA     P2
         RSUB
   BUF   RESW     300
   P1    WORD     5
   P2    WORD     10
         END      MAIN
```

Show the symbol table created by a 2-pass assembler for the above module. The LC-values shown in the symbol table must be in *decimal*.

**Question IV** (15 points)

This question deals with 4-byte instructions of SIC/XE. We say that a 4-byte instruction of SIC/XE is **special** if satisfies *all* of the following three conditions:

(a) The `e`-bit specified in the instruction is `1`,

(b) the `n`-bit specified in the instruction is `1` and

(c) the unsigned integer value stored in the least significant 20 bits of the instruction is at least 300 (decimal).

Write a C function `check_special` with the following header:

```
int   check_special (int  instr)
```

Assume that the `int` data type is 4-bytes long. The parameter `instr` to the above function is an integer that represents a 4-byte instruction for SIC/XE. The function must return the value 1 if the 4-byte instruction given by the parameter `instr` is special; otherwise, the function must return the value 0.

You need to show only the C code for the above function. You may use magic numbers and there is no need to include comments in your code.

**Question V** (20 points)

Assume that the following constant and type definitions are available.

```
#define   TITLE_MAX   100
#define   AUTHOR_MAX   30
#define   TOPIC_MAX    15

struct   book_record {
  char   title[TITLE_MAX];  char  author[AUTHOR_MAX];
  char   topic[TOPIC_MAX];  int   qty_in_stock;
  float  price;
};
```

Write a C function `count` with the following header:

```
int  count (char *infile,  char *my_topic, int q)
```

Here, the parameter `infile` gives the name of a *binary* (unformatted) input file which contains zero or more records of type `struct book_record`. We say that a record is **important** if satisfies *both* of the following conditions:

(a) the string stored in `topic` field of the record is identical to that given by the parameter `my_topic` and

(b) the integer value stored in the `qty_in_stock` field of the record is less than that given by the parameter `q`.

Your function must go through the records of the input file. It should compute and return the number of important records in the file.

Your answer needs to contain only the C code for the above function. No error checks are necessary. You may use magic numbers and there is no need to include comments in your code.