This is an open book and notes exam. There are 7 parts, 8 pages plus one blank sheet. Make sure that you have them all before you begin. The duration is 2 hours.

PART 1 (8 points) The vocabulary in this problem has specific meaning in the topic of operating systems.

1. What is the difference between an “executable program” and a “process”?

2. Why are system calls (such as fork(), exec(), exit(), ...) implemented by a trap (or interrupt or other “supervisor call” instruction) that causes an interrupt or exception rather than a conventional subroutine call (say using a jal instruction) into the kernel?

PART 2 (15 points) Finish what the output of the dlx402 assembler should be: (The opcode for addi is 0x08, the funcode for addu is 0x21 and the opcode for sw is 0x2b.)

; .text 0
;
; addi r2, r0, forlab-2
;
; addu r3, r4, r5
;
;forlab: sw 0x11(r3), r2
;
; .end 0
PART 3 (23 points) Here the contents of the file exam.h which defines a hypothetical file format for examinations (like this one), suggestive of ELF.

typedef unsigned short EX_Half;
typedef unsigned long EX_Offset;
typedef unsigned long EX_Word;

/*
 * Examination file header appears at the beginning of the file
 */
#define EX_NIDENT 16
typedef struct {
    unsigned char ex_ident[EX_NIDENT]; /* examination file identification bytes*/
    EX_Half ex_questcount; /* number of questions*/
    EX_Half ex_points; /* number of points, usually 100*/
    EX_Offset ex_qheadsoff; /* offset to beginning of question header table*/
} EX_Examhdr;

/*
 * Question header table entry, one for each question
 */
typedef struct {
    EX_Half qh_qnum; /* question number*/
    EX_Half qh_points; /* number of points for this question*/
    EX_Word qh_size; /* size in bytes of question body*/
    EX_Offset qh_offset; /* file offset of question body*/
} EX_Qhdr;

Beginning below and on the next page, write a clearly written, complete and correct C program that does the following. Label the code that does each of the steps with its number.

1. Open for reading the file named by the first command line argument. If this cannot be done for any reason, print an informative error message and exit.

2. Read the file header and print every item in it, using an informative format.

3. Print every item in every question header in the question header table, using an informative format.

4. For each question, print a newline, the string Question= followed by the question number from the header in decimal, a newline and then print each character in the question body by calling putchar for each.

#include <stdlib.h> /* declares malloc, calloc, exit*/
#include <stdio.h> /* declares fopen, fread, fseek, printf, putchar, FILE type*/
#include "exam.h"
int main(int argc, char *argv[])
{

PART 4 (18 points) This question is to demonstrate the operation of the hash table data structure specified for the assembler project. To make the question feasible in an examination, the table sizes have been made unreasonably small.

The symbols and their values, in the order they are to be inserted, are:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>first</td>
<td>312</td>
</tr>
<tr>
<td>second</td>
<td>238</td>
</tr>
<tr>
<td>third</td>
<td>0</td>
</tr>
<tr>
<td>fourth</td>
<td>5</td>
</tr>
<tr>
<td>fifth</td>
<td>5</td>
</tr>
<tr>
<td>sixth</td>
<td>0</td>
</tr>
<tr>
<td>seventh</td>
<td>779</td>
</tr>
</tbody>
</table>

The hash table size is 3, so the hash values are in the range 0 to 2. Instead of having you compute a specific hash function value for each symbol, you should make up these values. Do not make them all the same. They should “look random.”

The C declarations for the data structure are:

```c
typedef struct {
    unsigned short name; /* name=0 means entry is unused */
    unsigned short next; /* link, next=0 means last in chain */
    unsigned long value;
} ST_entry;

ST_entry HT[3]; /* the hash table, very small */
ST_entry OV[6]; /* space for overflow entries, OV[0] is wasted */
unsigned int OV_free = 1; /* first available entry in OV table */
char string[60]; /* the string table, string[0] is wasted */
unsigned int string_free = 1; /* first available entry in string table */
```

Show the contents of the data structures on the next page after all seven symbols and their values have been inserted. Write characters in the string table and decimal numerals in all other relevant boxes.
Note:
Layout of an
*ST_entry*

is

- name
- next
- value
Part 5 (8 points) Consider the C language variable x declared by:

```c
union {
    char s[4];
    long unsigned int n;
} x;
```

The implementation uses 8 bit chars and 32 bit long integers. Suppose the storage for x is the bytes whose addresses are 4, 5, 6 and 7. Suppose the assignment statement `x.n = 0x02030405;` were executed.

1. Assume that the implementation byte ordering is “Big Endian,” in other words, MSB first or left-to-right. Complete the box diagram that shows the four bytes used for x, with (1) their addresses, (2) the contents of each byte, and (3) which ones are `x.s[0]`, `x.s[1]`, `x.s[2]` and `x.s[4]`.

```
X.s[?]  X.s[ ]  X.s[ ]  X.s[ ]  X.s[ ]
```

Contents: 
Byte addr: __ __ __ __

2. Repeat the above assuming that the byte ordering is “Little Endian.”

```
X.s[?]  X.s[ ]  X.s[ ]  X.s[ ]  X.s[ ]
```

Contents: 
Byte addr: __ __ __ __
PART 6 (18 points) Here are two assemblies as in Beck’s book. Fill in the object file header (with the program name), the external definition table, the external reference table and the modification records as on pages 131-132 of Beck’s book. (These are parts of the object files produced by the relocatable assembler. Remember that the word size in Beck’s architecture is 3 bytes. Also note that the name of the section like PROGA and PROGB is a symbol which represents the address of the section’s first byte.) Use any format that clearly expresses the logical content and uses strings (not reference numbers) for the symbols.

<table>
<thead>
<tr>
<th>Loc</th>
<th>Source Statement</th>
<th>Obj.code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>PROGA START 0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>EXTDEF SHEEP</td>
<td></td>
</tr>
<tr>
<td></td>
<td>EXTREF CAT</td>
<td></td>
</tr>
<tr>
<td>0000</td>
<td>WORD SHEEP 000003</td>
<td></td>
</tr>
<tr>
<td>0003</td>
<td>SHEEP WORD 151925</td>
<td>151925</td>
</tr>
<tr>
<td>0006</td>
<td>WORD CAT 000000</td>
<td></td>
</tr>
<tr>
<td></td>
<td>END</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Loc</th>
<th>Source Statement</th>
<th>Obj.code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>PROGB START 0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>EXTDEF CAT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>EXTREF SHEEP</td>
<td></td>
</tr>
<tr>
<td>0000</td>
<td>WORD 1B3F27</td>
<td>1B3F27</td>
</tr>
<tr>
<td>0003</td>
<td>CAT WORD GOAT 00006</td>
<td></td>
</tr>
<tr>
<td>0006</td>
<td>GOAT WORD CAT 00003</td>
<td></td>
</tr>
<tr>
<td>0009</td>
<td>WORD SHEEP 000000</td>
<td></td>
</tr>
<tr>
<td>000C</td>
<td>WORD GOAT-CAT FFFFFD</td>
<td></td>
</tr>
<tr>
<td></td>
<td>END</td>
<td></td>
</tr>
</tbody>
</table>

Header

External Definition Table

External Reference Table

Modification Records
PART 7 (10 points, multiple choice) Choose the one of the three questions below that you can answer most accurately. Write a clear answer of 200 words or less. Superficial answers will not count much.

1. The traditional UNIX file system “inode” has both some direct pointers to disk blocks and some indirect pointers to disk blocks. Explain the difference and explain why the file system was designed that way.

2. What model of high-performance computation is replacing traditional supercomputers? What specific developments are making this transition to lower cost supercomputing possible?

3. Beck’s SIC/XE instruction set architecture includes instructions (like LDA) with program counter relative direct addressing mode (as in the dlx402 jal instruction), with a 12 bit displacement field. Suppose an instruction using this mode appears in one section and its effective address is given by a symbol that will be defined in a different section. Invent a scheme whereby such a displacement field will be correctly modified by Beck’s relocating linker and loader so separately assembled sections can be linked. The scheme must be implemented by the assembler and must not require new modification types be added to Beck’s linker and loader. Hints: The assembler always emits the name of the section as a symbol, for example PROGA, PROGB and PROGC. A clear example will be better than a vague description.