Operating Systems

Further Questions, Easter 2002

tim.harris@cl.cam.ac.uk

This question sheet is intended to accompany the first section of the Easter 2002 Part 1A course on *Operating Systems*, covering slides 1–26, which have not usually been the direct subject of Tripos questions.

These questions each concern a computer system, based on the traditional Von Neumann architecture, with the following properties:

- There is a single processor with 16 registers, each of which can hold a 32-bit integer value in *twos' complement* format. *Little endian* format is used in memory.
- There is 256MB of *byte addressed* memory, meaning that address 0 denotes the first byte of storage, address 1 is the next byte, address 2 the next after that etc.
- Programs for the computer are written in an assembly language supporting the following statements:
  - `ld #x -> ri`
    load the 16-bit integer constant *x* into register *ri*
  - `ld [a] -> ri`
    load the 32-bit value from memory starting at address *a* into register *ri*
  - `st ri -> [a]`
    store a 32-bit value from register *ri* into memory starting at address *a*
  - `op ra, rb -> rc`
    perform a binary arithmetic operation, such as addition or multiplication, combining the values in registers *ra* and *rb*, storing the result into register *rc* and setting the *processor flags* as appropriate
  - `br a`
    transfer execution to the instruction at address *a*, with different variants of this instruction performing conditional branches

For example, the following program would form the arithmetic sum of a series of 32-bit integer values starting at address `0x1000` and continuing until the total equals or exceeds 42. The result is placed in register `r1`. Each instruction occupies 4 bytes in memory and is shown, below, against that instruction's address in memory. Execution begins at address 0.

```
0x0000 : ld #0 -> r1
0x0004 : ld #4 -> r2
0x0008 : ld #0x1000 -> r3
0x000c : ld [r3] -> r4
0x0010 : add r4, r1 -> r1
0x0014 : add r3, r2 -> r3
0x0018 : ld #42 -> r5
0x001c : sub r1, r5 -> r5
0x0020 : blt 0x000c
```

1. Show how unsigned numbers can be represented in an 8-bit byte. What is the largest number that can be represented? What is the smallest? Are there any numbers which can be represented in more than one way?

2. Show how a *twos' complement* format can be used to represent signed numbers in an 8-bit byte, giving the binary encoding for -128, for 0 and for +127.

3. Show how a 32-bit signed integer value can be represented using 4 bytes of storage in memory in *little endian* order. Illustrate your answer with the data that would be held for the number 0, the number 256 and the number -65536.

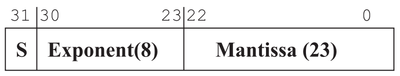
4. Suppose that, using a 32-bit signed representation, the number 10 is stored at address `0x1000`, 12 is stored at address `0x1004` and 22 at address `0x1008`. Illustrate how the *fetch-execute* cycle would proceed if execution starts at address 0.

5. Design your own format for representing these example processor instructions as 32-bit values. What restrictions does this format impose on the addresses `a` to which a branch instruction `br a` may transfer execution? Suggest one way of avoiding these restrictions.

6. The ASCII encoding for characters maps each letter in the English alphabet, each digit and various other punctuation symbols to 8-bit codes. For instance, the letter 'A' is encoded 65 and 'B' is encoded 66.

   Using "PROCESSOR" as an example, describe and illustrate two ways in which text strings can be represented. What are the advantages and disadvantages of each approach?

7. Floating point numbers in the IEEE 754 format are represented using a *sign bit*, *exponent* and a *mantissa*:

   The exponent is held with a bias of 127 (so 0 represents 2^{-127}, 127 represents 2^0 and 255 represents 2^{128}). Show how the number $1\frac{3}{4}$ could be held *(i)* if the mantissa is an ordinary unsigned integer and *(ii)* if the mantissa is held in a *normalized* form.

   Why might the normalized form be preferred?

8. A tree data type is defined as follows in ML:

```
datatype tree = node of tree * tree
              | leaf of int;
```

   Show how trees could be represented in a system in which compound data structures begin with a 32-bit *tag* identifying the constructor, followed by a series of 32-bit *values* containing the data held in each component. Illustrate your answer by considering:

```
node(node(leaf(1),leaf(2)), node(leaf(3),leaf(4)))
```

9. The n^{th} Fibonacci number is defined recursively by $f(n) = f(n-1) + f(n-2)$ where $n > 0$, $f(0) = 0$ and $f(1) = 1$. Write an assembly language program that will compute the first 16 Fibonacci numbers and store them at addresses 0x1000, 0x1004, etc.