Exam in EDAF15 Algorithm Implementation

June 2, 2015, 14-19

Enda hjälpmedel är en ASCII-tabell som delas ut!

You may answer in English, på svenska, auf Deutsch, или по-русски.

30 out of 60p are needed to pass the exam.

1. (10p) Consider a RISC processor with the five pipeline stages:
   - Fetch
   - Decode
   - Execute
   - Memory access
   - Write back

   (a) (2p) What is meant by a **pipeline stall**?

   (b) (4p) In the following program:

   ```
   add  r1,r2,r3 // r1 = r2+r3
   sub  r4,r1,r2
   ```

   there is a dependence from the add to the sub instruction. Does this lead to a pipeline stall? Why or why not? If there is a pipeline stall, could an optimizing compiler have avoided it, and in that case how? Explain what to do with the code (if anything) but not the algorithm for doing it.

   (c) (4p) In the following program:

   ```
   load r1,r2,r3 // r1 = MEMORY[r2+r3]
   sub  r4,r1,r2
   add  r5,r2,r3
   ```

   there also is a dependence, from the load to the sub instruction. Does this lead to a pipeline stall? Why or why not? If there is a pipeline stall, could an optimizing compiler have avoided it, and in that case how? Explain what to do with the code (if anything) but not the algorithm for doing it.

2. (3p) In the RISC processor in question one (and in most other processors as well) there is both an instruction cache and a data cache. Why is that a good idea? What would happen if the processor only has one cache?

3. (3p) What does **cache associativity** mean?

4. (4p) What does **cache block size** mean? What happens if the size is too small or too large? What is a "reasonable" size?

5. (20p)

(a) (17p) Implement the four functions declared below for a circular double linked list for which an empty list is represented by a null pointer (i.e. `NULL`).

- (3p) `new_list`
- (4p) `free_list`
- (6p) `reverse`
- (4p) `insert_last`

(b) (3p) For each of `reverse` and `insert_last` you should draw two figures with an example illustrating what you intend your code to achieve as follows.

- (1p) For `reverse` your two figures should show what a list can look like before a loop iteration (assuming you use a loop) and after a loop iteration.
- (2p) For `insert_last` your two figures should show what a list can look like before and after you have inserted a new node, both for an initially empty list, and for a non-empty list.

There is no particular required format for the figures!

```
typedef struct list_t list_t;

struct list_t {
        list_t*         succ; // pointer to next node in the list
        list_t*         pred; // pointer to previous node in the list
        void*           data; // pointer to data
};

// Allocate a new list node with the data.
list_t* new_list(void* data);

// Deallocate all list nodes but not the data in the lists.
void free_list(list_t* list);

// Reverse the list and set *list to point to the new start node.
void reverse(list_t** list);

// Insert new data last in the list.
void insert_last(list_t** list, void* data);
```

6. (10p) Write a function `int count(uint64_t a)` which counts the number of bits with value one in a variable of type `uint64_t` (which normally is the same as `unsigned long long`). Your function should be as fast as possible and assume your computer has a lot of RAM memory such as 64 GB.

   For instance, `count(5) = 2`

7. (10p) How would you rewrite the following loop to increase performance on the assumption that the input is a huge text file with English text, (i.e. no non-ASCII characters such as Å) and:

   - (5p) The assumption that the code in X, Y, Z is very small such as one instruction.
   - (5p) The assumption that the code in X, Y, Z is quite large.

   The function `getchar` reads the next character of input and `EOF` is a macro (usually -1) which stands for end of file.

```
int     c;

while ((c = getchar()) != EOF)
        if (c == '\n')
                X;
        else if (c == ' ')
                Y;
        else
                Z;
```