Computation Structures — Tutorial 2

11 October, 2016

1 μ-code for ULG01

1. Give symbolic `Ulg01` microcode for instruction `BRTBL(Ra,Label,Rc)`. This instruction transfers execution to the address

   $$(PC + 4 \times Literal + Ra)\ \&\ 0xFFFFFFFC$$

   The value of the `Literal` field in the instruction is computed by the assembler as in `BEQ(Ra,label,Rc)`, i.e. :

   $$Literal = \frac{OFFSET(label) - OFFSET(CurrentInstruction)}{4} - 1$$

   The address of the instruction immediately following `BRTBL` is saved in register `Rc`.

2. Provide microcode for the `SUBFLAGS(Ra,Rb,Rc)` instruction. It must write in register `Rc` the ALU flags ($\overline{C}, N$ and E in bits 2, 1 and 0 resp.) resulting of `Ra - Rb` substraction. Any other bit of `Rc` shall be zero.

1.1 Suggested exercises

1. Provide microcode for the `JMPGE(Ra,Rb,Rc)` instruction. This behaves as `JMP(Ra,Rc)` when the contents of register `Rb` is above or equal to 0. If not, the instruction has no effect.

2 β-assembly

1. Consider the following C function:

```
int sum(int n) {
    int i = 0;
    int s = 0;
    while (i < n) {
        s += i;
        i++;
    }
    return s;
}
```

   Translate this function in a β-assembly procedure using registers for local variables.

2. The following C function computes the Greatest Common Divisor of two integer numbers:

```c
int gcd(int a, int b) {
    if (a == b)
        return a;

    if (a > b)
        return gcd(a - b, b);

    return gcd(a, b - a);
}
```

   (a) Translate this function in a β-assembly procedure.
   (b) Write β-assembly code that defines two global variables $x = 27$ and $y = 9$ and a *main* function that invokes *gcd* using x and y as arguments.
   (c) How much memory is used on the stack for every call of *gcd* ?
   (d) Give a schema of the stack before *main* branches to *gcd*.
   (e) Give a schema of the stack after the first recursive call to *gcd*.

2.1 Suggested exercises

1. Same exercise than 2.1, but assume that only `R1` and `R2` are available (Tip: use local variables stored in RAM).
2. For a machine that has no `DIV`, `DIVC`, `MUL` nor `MULC` instructions, provide an assembly function `modulo` receiving two positive integers and returning the remainder of the integer division of its first argument by the second one.
3. Write a program that can be run by the BETA emulator BSim calling `modulo`.
4. Write a `modtab` function taking two arguments:
   (a) a DRAM address of an integers (32-bit) array;
   (b) the array length (in items).

   The modtab function will replace each item in the array by the remainder of this item's division by the next item. The last item in the array shall be replaced by 0.

5. Write an `even_sub_odd` function taking two arguments :

   (a) the DRAM address of an array of integers;

   (b) the array length.

   The function computes the difference between

   - the sum of all the items sitting at an *even* position;
   - the sum of all the items sitting at an *odd* position.