Computation Structures — Tutorial 13

December 11, 2018

1. **Open book, Q1, 01/2018** - We introduce the instruction `IND_SWAPIF(Rc, Rb, Ra)`. This instruction conditionally swaps the values found in memory at the address Mem[Reg[Ra]], which we will call the effective address (`EA`), with the value contained in the register `Rc`. The condition for the swap to take place is that `Reg[Rb]` is strictly greater than `Mem[EA]`.

```
IND_SWAPIF(Rc, Rb, Ra): PC <- PC + 4
                        EA <- Mem[Reg[Ra]]
                        TMP <- Mem[EA]
                        if TMP < Reg[Rb] then;
                            SWAP(Reg[Rc], Mem[EA])
```

   (a) Implement the microcode of `IND_SWAPIF` for ULg02 in supervisor mode.

   (b) On Ulg02, are the microcodes of `IND_SWAPIF` in supervisor and user mode different? Explain.

   (c) Can this instruction be implemented on ULg03 in user mode ? And in supervisor mode ? Explain.

2. **Open book, Q2, 01/2018** - Consider the `ssort()` function that sorts an array given a pointer to the first element of this array and its size.

```
1  void swap(int* a, int* b) {
2      int tmp = *a;
3      *a = *b;
4      *b = tmp;
5  }
6
7  int divceil(int a, int b) {
8      int div = a / b;
9      if (a % b != 0) {
10         return div + 1;
11     } else {
12         return div;
13     }
14 }
15
16 void ssort(int* array, int size) {
17     if (size <= 1) {
18         return;
19     }
20     if (array[0] > array[size - 1]) {
21         swap(array, array + size - 1);
22     }
23     if (size > 2) {
24         int onethird = size / 3;
25         int twothird = divceil(2 * size, 3);
26         ssort(array, twothird);
27         ssort(array + onethird, size - onethird);
28         ssort(array, twothird);
29     }
30 }
```

   (a) Translate this function to β-assembly code.

(b) Represent the largest content of the stack that occurs if the main program just contains a call to `ssort` on the array `[4, 7, 2, 9]` of size 4.

3. **Closed book, Q3, 01/2018** - Check whether each of the four programs below runs as expected on the performance-oriented β-machine with a 4-stage pipeline (see Figure 1 on the next page). If it does not: explain why, give a possible (hardware or software) solution and its advantages and drawbacks. You **must** provide a **different** solution for each program that would not be executed correctly.

(a)
```
1 ADD(R3, R7, R8)
2 ADD(R1, R2, R3)
3 MULC(R3, 4, R4)
4 LD(R4, 0, R6)
5 ST(R6, 4, R4)
```

(b)
```
1 LD(R5, 12, R5)
2 ADD(R2, R5, R5)
3 MUL(R2, R3, R7)
4 MULC(R5, 4, R6)
```

(c)
```
1 LD(R1, 0, R2)
2 LD(R1, 4, R3)
3 LD(R1, 8, R4)
4 ADDC(R2, 1, R2)
5 ADDC(R3, 1, R3)
```

(d)
```
1 ADDC(R2, 5, R3)
2 ADD(R6, R2, R4)
3 MULC(R5, 4, R17)
4 LD(R17, 0, R6)
```

You may use the diagram on the next page to describe any hardware solution you propose, but do write your name on it !

Name:
Surname:
ULg ID:

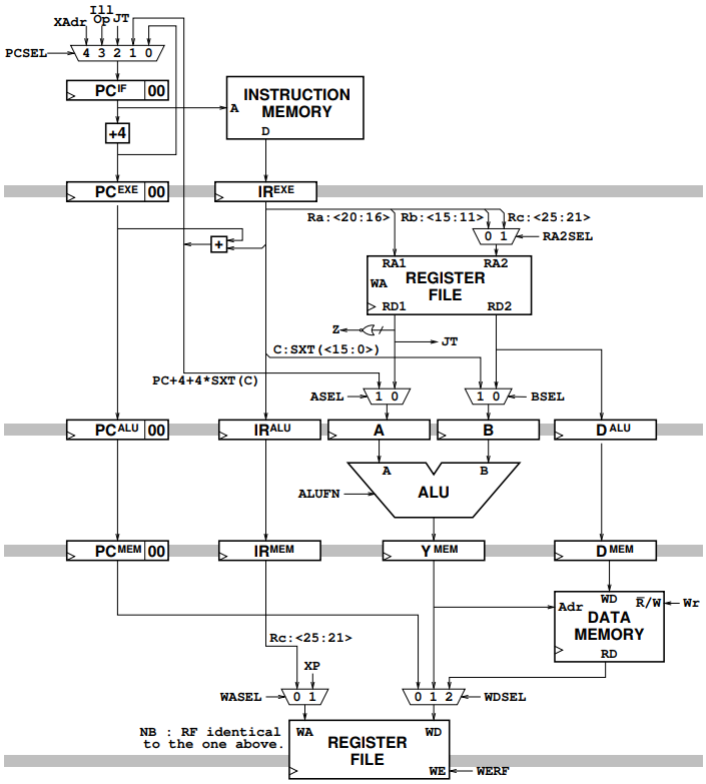

Figure 1: The performance-oriented β-machine with a 4-stage pipeline