Computer Architecture
Preparation for the Midterm Exam

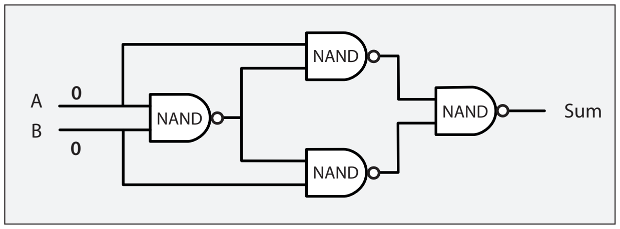
Paul Mellies

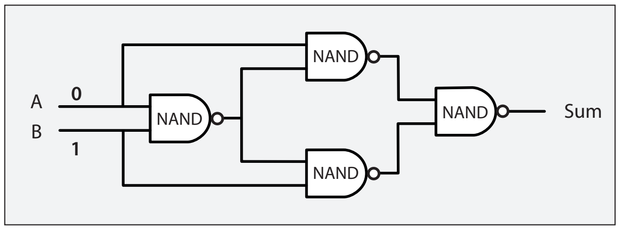
Exercice 1

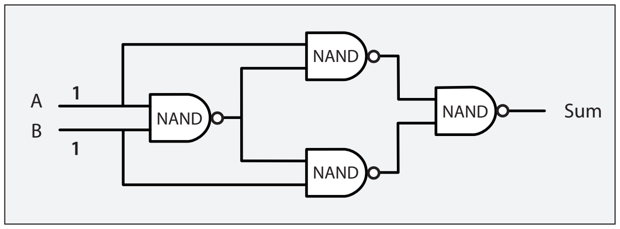
§1a. As explained during the course, the following circuit is the specific part of the half-adder designed to compute the sum of the two inputs A and B. Each of three diagrams below corresponds to one of the four possible inputs

$$A = 0\,,\ B = 0 \qquad A = 0\,,\ B = 1 \qquad A = 1\,,\ B = 1$$

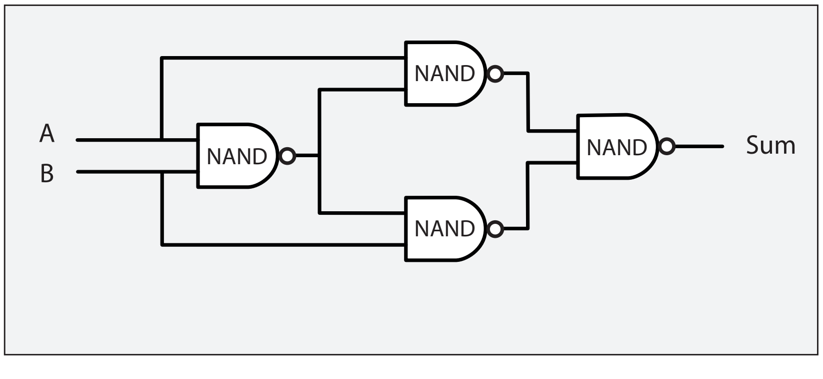
Indicate for each of the three inputs A and B the value of *every wire* in the circuit, and in particular the value of the output Sum:



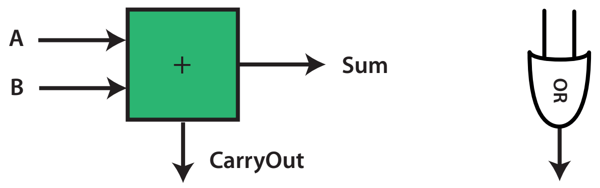




§1b. Complete the circuit below in order to obtain a half-adder:



§1c. Recall that an half-adder and a OR gate are depicted as follows:



Using this notation, explain with a diagram how to construct a full adder using two half-adders together with an OR gate:

Exercise 2.

§2. Ben Bitdiddle and Alyssa P. Hacker are having an argument. Ben says, « All integers greater than zero and exactly divisible by six have exactly two 1's in their binary representation. » Alyssa disagrees. She says, « No, but all such numbers have an even number of 1's in their representation. » Do you agree with Ben or Alyssa or both or neither? Explain.

Exercise 3.

Suppose that you are a collector of old computers and that you bought a machine whose signed and unsigned integer are represented using 8 bits integers.

§3a. What are the maximal and minimal unsigned integers which can be represented as an 8-bit integer of your machine?

§3b. What are the maximal and minimal signed integers (two's complement) which can be represented as an 8-bit integer of the machine?

§3c. Suppose that you use the signed add instruction of the machine in order to add the signed integer 102 and the signed integer 88. Could you describe what happens? What would happen if you used the unsigned add instruction instead? And what would be the result in that case?

Exercise 4.

§4a. Translate the following decimal numbers into binary numbers of length 8 bits:

15 82 97 114

§4b. Translate the same decimal numbers into hexadecimal numbers

15 82 97 114

§4c. Translate the decimal numbers into signed binary numbers (two's complement) of length 8 bits:

-1 -15 -82 -97 -114

§4d. Describe a simple recipe (or algorithm) to translate a positive number represented as a signed binary number into its opposite number, also represented as a signed binary number on 8 bits,

§4e. Indicate whether the same recipe (or algorithm) works in order to translate a negative number (typically -5) into its absolute value (typically 5).

§4f. Explain what one means by « sign extension » from 8 bits to 16 bits.

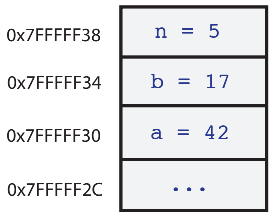
Exercise 5.

§5a. Consider the C program below

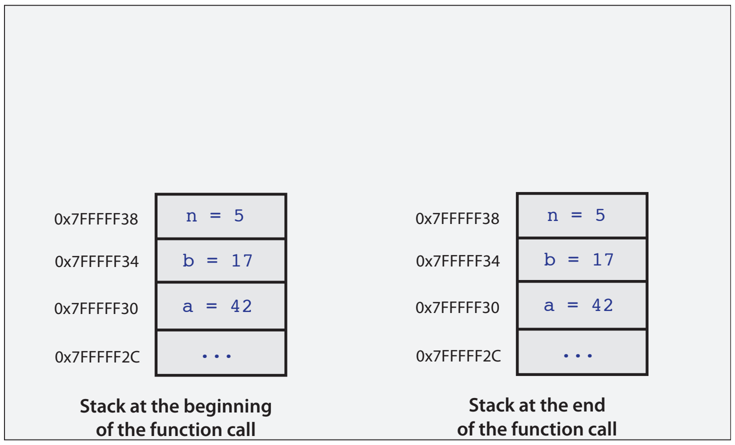
```c
void swap(int x, int y){
    int tmp;
    tmp = y;
    y = x;
    x = tmp;
}

int main()
{
    int a = 42;
    int b = 17;
    int n = 5;

    printf("a=%d b=%d \n", a , b);
    swap(a,b);
    printf("a=%d b=%d \n", a , b);
}
```

and indicate what the `main` program prints as output

§5b. [optional] Suppose that the stack has the following shape when the function `swap` is called by the `main` function:



Explain how many local variables are allocated during the function call, and describe the shape of the stack at the beginning and at the end of the execution of the `swap` function, before it returns.

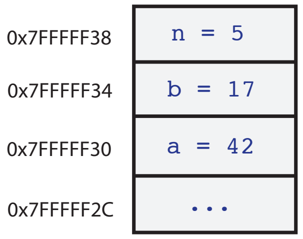


§5c. Consider now the C program below where the `swap` function has been replaced by the `swapbyptr` function:

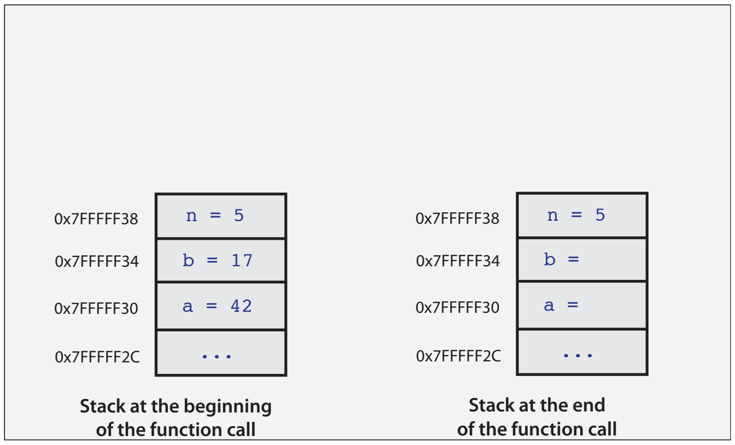
```
void swapbyptr(int *xptr, int *yptr){
    int tmp;
    tmp = *yptr;
    *yptr = *xptr;
    *xptr = tmp;
}

int main()
{
    int a = 42;
    int b = 17;
    int n = 5;

    printf("a=%d b=%d \n", a , b);
    swapbyptr(&a,&b);
    printf("a=%d b=%d \n", a , b);
}
```

Indicate below what the `main` program prints as output in that case

§5d. [optional] Suppose that the stack has the same shape as previously:



when the function `swapbyptr` is called by the `main` function. Explain how many local variables are allocated during the function call, and describe below the shape of the stack at the beginning and at the end of the execution of the `swapbyptr` function:



Exercise 6.

As explained during the course, one declares the following struct type `matrix` in order to implement Java-like matrices in memory.

```
struct matrix {
    int width;
    int height;
    int **content;
};
```

§6a. Can you draw a picture explaining how a matrix of width 3 and height 2 would be represented in memory.

§6b. Suppose that the struct `matrix` is of size 16 bytes. Give a simple formula expressing the size of allocated memory for a matrix with width w and height h.

§6c. Explain why the C function `clone_matrix` below whose purpose is to clone a Java-like matrix and to return a pointer to its clone matrix will not work as expected.

```
struct matrix *clone_matrix (struct matrix *pm){
  struct matrix *pmc;
  int i, j;

  // (1) allocate the structure of the clone
  pmc = malloc (sizeof(struct matrix));
  // (2) copies the content of the fields
  pmc -> width = pm -> width;
  pmc -> height = pm -> height;
  pmc -> content = pm -> content;

return pmc; // return the address of the clone matrix
}
```

§6d. Describe with a drawing the other possible representation of matrices in memory (also discussed during the course) with struct `matrix` defined in that case as

```
struct matrix {
    int width;
    int height;
    int *content;
};
```

Suppose as in §6b that the struct `matrix` is of size 16 bytes. Give a simple formula expressing the size of allocated memory for a matrix with width w and height h in that representation.

§6e. Explain the advantages and disadvantages of representing matrices of integers as Java-like matrices instead of the alternative representation discussed in §6d.

Exercise 7.

§7a. Explain in a few words the difference between an architecture and a microarchitecture.

§7b. Explain in a few words the notion of « memory leak » encountered in our study of dynamic allocation in the programming language C.

§7c. Explain in a few words why the `free` function does not need to take as parameter the size in memory of the deallocated block, but only its pointer.