Lab 9. E Register, Memory, Program Counter, I Register

E Register

Since single instruction may perform up to two memory accesses and our main memory can only serve a single request per cycle, we construct the processor in a way that it performs *fetch* cycles and *execute* cycles in an alternating fashion. In other words, it takes our processor two clock cycles to perform one instruction. This is done by introducing REGISTER E (See Figure 1).

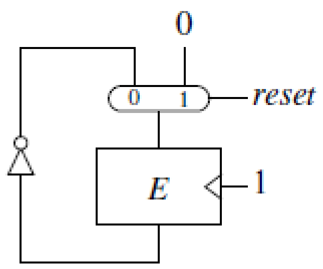

Figure 1

Whenever register E is 0, the processor fetches instruction from main memory.

Whenever register E is 1, the processor executes the fetched instruction.

Memory, PC, I Register

The main memory (Figure 2) is a single port memory that is used to store instructions to be executed and data.

The unit called **program counter (PC)** determines the address of instruction that needs to be executed. PC is incremented once in every two clock cycles. PC is incremented when E is 1.

The address of the memory location where data needs to be read or written is determined by **data_addr_in**, and the value that should be stored is specified by **data_in**. Since we have byte addressable memory, the last two bits of the addresses should be 0s. That is why on figure 2, addresses are represented as 30 bit long busses. The last two bits are 0s, and we use the first 30 bits to determine memory location of a word.

The value of Register E determines whether we access instruction or data. Whenever E is 0 (Fetch cycle) PC decides Read Address. Whenever E is 1 (Execute cycle), data_addr_in decides Read Address.

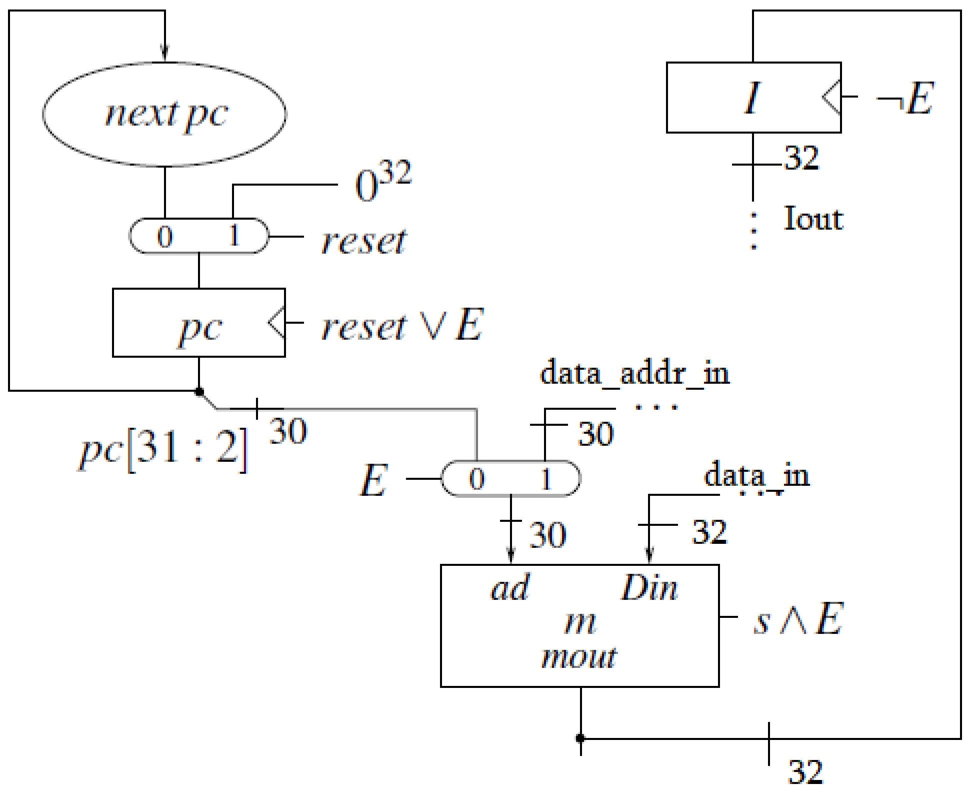

Figure 2

During the fetch cycle (E=0), fetched instruction is stored in instruction register I.

Lab Tasks:

You are asked to write a single module that implements Figure 1 and Figure 2.

Inputs:

Clock – 1 bit signal.

Reset – 1 bit signal. If reset is 1, PC and E becomes 0.

S - 1 bit signal. S is 1, if we have store instruction

Next PC – 32 bit long input. Specifies the address of next instruction. We will implement module that calculates Next PC in the future. At this point, Make it 32 bit long input signal

data_addr_in – 32 bit long input. Address to memory location where data should be stored or read data_in – 32 bit long input. Data that needs to be stored.

Outputs:

PC_out – 32 bit long signal. Specifie the address of instruction that is being executed.

Iout – 32 bit long signal. Instruction that is being executed

Mout – 32 bit long signal. In execution cycle, Mout is Data out for load instruction. In fetch cycle, Mout is instruction that will be executed in the following execution cycle.

E – 1 bit long signal. Specifying whether we have fetch or execution cycle.

1) **Register E:** Your module should implement logic for computing Register E.
2) **Program Counter**: Implement logic for Program Counter. Do not implement logic for Next PC. That comes later.
3) **Memory**: Implement memory as an array of 32 bit long registers. We suggest to have 256 array elements. You can use $readmemb command to initialize memory.
4) **Instruction Register I**: Your module should implement logic for I register.
5) **Simulation & Verification**: Write testbench, generate waveforms and test your module.

We have seen several elegant modules of the previous assignments. If you would like to keep up ELEGANT work, our advice is to review Chapter 8 in Pr. Wolfgang's book (One that you used last semester). Especially, Pay close attention to sections 8.3.0 – 8.3.4.

It PLEASES the heart to see elegant modules. However, our hearts ache when we see reports written carelessly. Writing understandable reports is important for your future careers. That is why you are asked to USE THE TEMPLATE.

Good Luck