STACK:

Some of the benchmark programs require calling functions or subroutines. Call and return instructions are utilized with the help of a stack. The stack is a separated section of the data RAM. When a function call is executed, the address held in the program counter is stored here. This is essential, this address is used later when the program counter is returning from a subroutine. Sometimes the programmer wants to call functions inside functions. In our case, the Fibonacci benchmark enters many recursions before the first return occurs. To keep track which address belongs to which function call, the stack has a pointer always pointing to the HEAD of the stack. We have decided to use a separate 16-bit wide stack register, which will contain the pointer. This register cannot be accessed by any instruction other than call and return. Whenever a call is initiated the program counter is stored to the location of the HEAD. The ALU simultaneously increases the pointer by 1. Essentially the return addresses will be stored in the same exact order their corresponding function calls were executing. When returning, the pointer is pre decremented, so it can access the last return address. The implementation above will be referred to as the instruction stack. In addition to this, we have also included a data stack, which works in a similar manner. This can be used to store data from one of the CPU's registers while using another as data stack register. When using the data stack, it is important to never overwrite the register used as the pointer. Otherwise, the address and the saved data is lost. This could result in a memory leak.

PIPELINING:

Pipelining is a significant boost in the throughput of the CPU. It overlaps fetching of the upcoming instruction with the previous one’s last cycle. Most of our instructions only take 2 clock cycles (fetching and executing). Implying that pipelining would almost double the speed of the processor. This is desirable, though raises many challenges. First and foremost, the finite-state machine must be designed accordingly. The 3 states are FETCH (00), EXEC1 (01) and EXEC2 (10). During the operation, the system should never return to FETCH. EXTRA is an input coming from the decoder. It is high for the few instructions that require an additional cycle to execute. These will later be referred to as 3 cycle instructions (AIM, SIM, LDI, LOAD, POP, RTN, MUL, MLS). Using Karnaugh maps, the Boolean expressions can be derived for each bit of N. The state transition table was constructed as follows:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Q [1] | Q [0] | EXTRA | N [1] | N [0] |
| 0 | 0 | X | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | X | 0 | 1 |

(In the table Q [1:0] denotes the previous state, while N [1:0] represent the next state.)

The next crucial element of pipelining is the incrementation of the program counter. In the last cycle of an instruction, the register should already contain the address of the consequent one. During normal operation, the state machine remains in EXEC1. This immediately suggests that during 2 cycle instructions the incrementation should happen with the upcoming clock edge in EXEC1. To prevent the very first instruction from executing twice the control line should always be high in FETCH as well. AIM, SIM and LDI use data in the next address immediately after them. In their case incrementation needs to happen in both EXEC1 and EXEC2. Apart from these, during other 3 cycle instructions (LOAD, RTN, POP, MUL, MLS), the control line for updating the program counter must only be high in EXEC2. Inevitably, certain commands, such as jumps, load the program counter with a specific value. Loading and incrementing cannot happen at once. However, this suggests the PC would lag a cycle behind, causing the instruction to execute twice. Loading a pre incremented value (using the ALU) into the program counter resolves the problem. A multiplexer will make sure to feed the correct address into RAM. Based on the above, Boolean expression of the count enable and the pc\_sload control lines are the following:

