Five stages pipeline processor

**Performance**All instructions are executed in 1 cycle except for:LDM: equivalent to 2 instructions (LDM, stall)  
CALL : equivalent to 5 instructions (PUSH flags, PUSH pc, JMP, stall, stall)  
RET: equivalent to 4 instructions (POP PC, POP Flags, stall, stall)  
RTI: equivalent to 4 instructions (POP PC, POP Flags, stall, stall)  
INT: equivalent to 8 instructions (3 stalls, PUSH PC, PUSH Flags, Jump address 0 , stall, stall)

**Handling Hazards**  
Data hazards : Handled by Full forwarding detected at execute stage  
Load use case : Handled by stalling for one cycle detected at decode stage  
POP Rdst - JMP Rdst : Handled by stalling for one cycle detected in decode stage  
POP Rdst - CALL Rdst : Handled by stalling for one cycle detected in decode stage  
POP Rdst - STD Rsrc, Rdst : Handled by stalling for one cycle detected in decode stage  
POP Rdst - ALU Rsrc, Rdst : Handled by stalling for one cycle detected in decode stage  
POP Rdst - OUT Rdst : Handled by stalling for one cycle detected in decode stage  
INT - CALL : Handled by enabling the write to PC and the processor will treat the call instruction which constructs of 5 instructions as one unit  
INT - JMP : Handled by enabling the write to PC and the processor will treat the jump(unconditional or conditional “with true result”) instruction which constructs of 3 instructions as one unit  
INT - RET : Handled by enabling the write to PC and the processor will treat the ret instruction which constructs of 4 instructions as one unit  
INT - RTI : Handled by enabling the write to PC and the processor will treat the rti instruction which constructs of 4 instructions as one unit

Assembler

1. Go to assembler directory
2. Open command prompt
3. write python assemblerGui.py
4. now you can write you Assembly code
5. press CTRL + s after writing your code

Design Changes:

1. The jump calculation moved to execute stage instead of decode stage to get the correct value of the flag register
2. Send the pc to be pushed to memory in case of call and int
3. Dealing with interrupt:
   1. Stalling 3 cycles at the beginning (to wait for the pipe to be finished)
   2. Push flag register
   3. Push pc
   4. Unconditional jump (performed at the execution stage)
   5. Nop
   6. Nop
4. Push and pop the correct value in the memory stage
5. Add more control signals for the call and return

Assumptions:

* assume that if the interrupt came with positive edge the instruction to be fetched from the memory will be fetched and finishes its executed
* assume that to perform nested interrupt you must wait till the interrupt service routine of the first one to start to be able to raise another interrupt

Current Design:A picture containing diagram

Description automatically generated