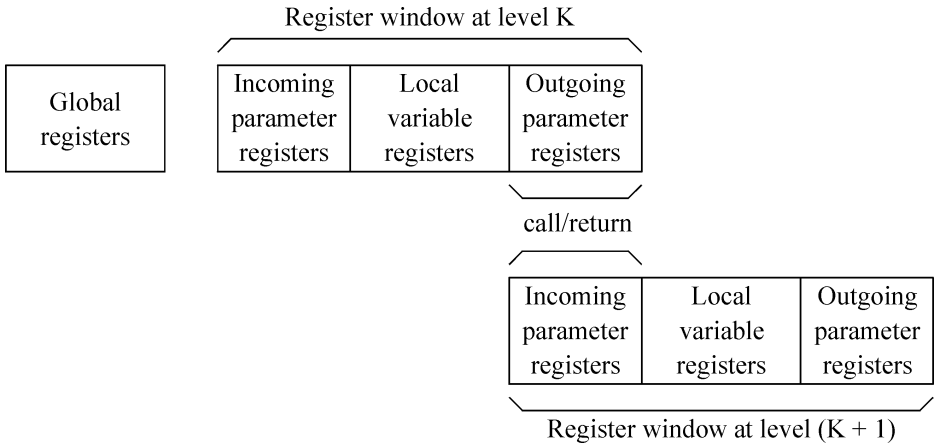
1. a) Why are complex instructions of CISC (Complex Instr. Set Computer) machines difficult to pipeline?

b) Why are RISC machines usually Load & Store machines (i.e., only Load and Store instructions access memory)?

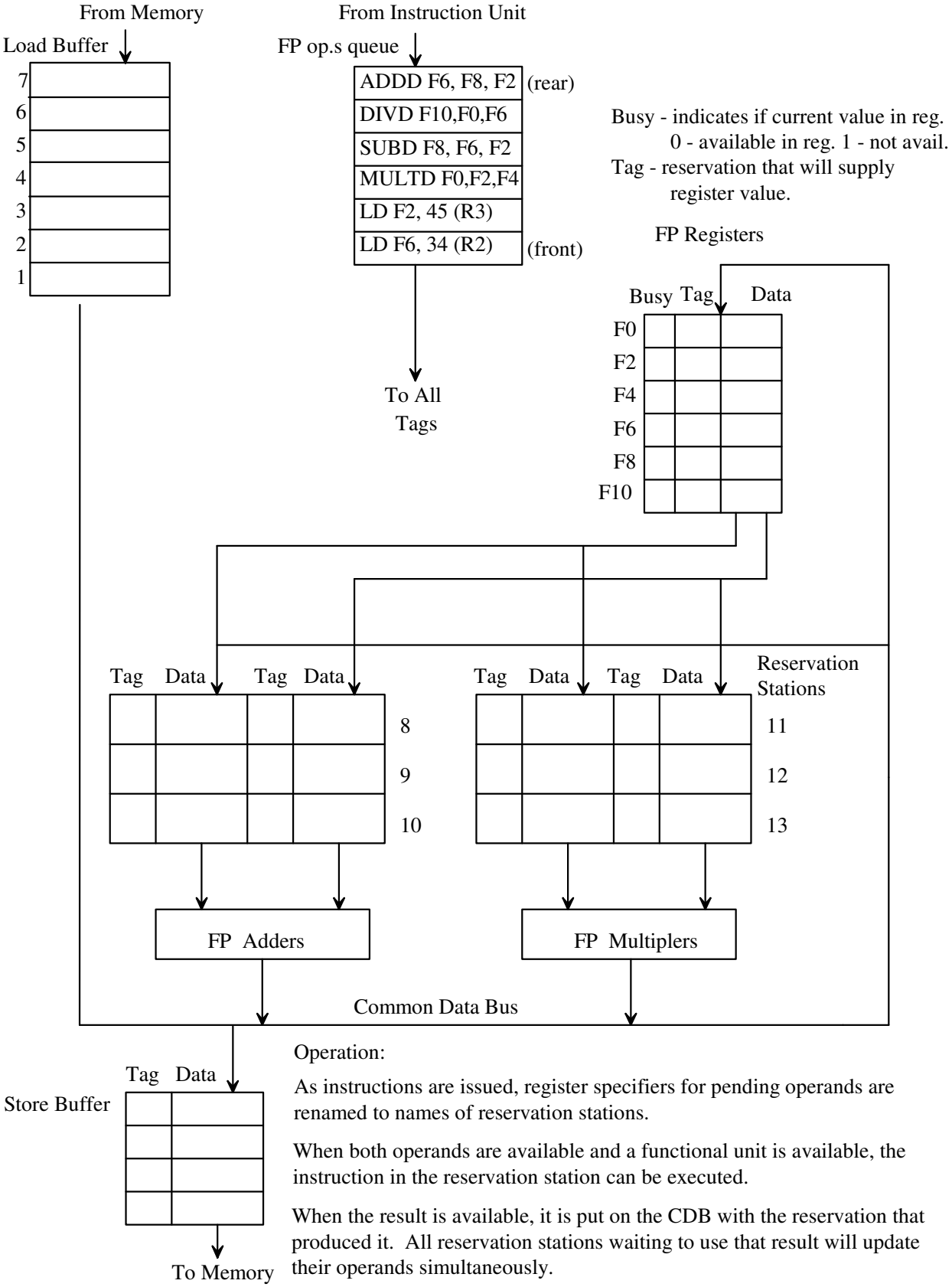
c) One characteristic of RISC machines is to have a large register file. Itanium -- large number of registers (128 general-purpose and 128 fl. pt. registers) with overlapping register windows



Itanium: first 32 registers for global variables and remaining 96 registers for local variables and parameters.

How does the overlap of Itanium register windows improve program performance?

2. Example using Tomasulo's Algorithm



Operation:

As instructions are issued, register specifiers for pending operands are renamed to names of reservation stations.

When both operands are available and a functional unit is available, the instruction in the reservation station can be executed.

When the result is available, it is put on the CDB with the reservation that produced it. All reservation stations waiting to use that result will update their operands simultaneously.

Tomasulo's Algorithm is an example of ***dynamic scheduling***. In dynamic scheduling the stages of the pipeline are split into three stages to allow for out-of-order execution:

1. *Issue* - decodes instructions and checks for structural hazards. Instructions are issued in-order through a FIFO queue to maintain correct data flow. If there is not a free reservation station of the appropriate type, the instruction queue stalls.
2. *Read operands* - waits until no data hazards, then read operands
3. *Write result* - send the result to the CDB to be grabbed by any waiting register or reservation stations

All instructions pass through the issue stage in order, but instructions stalling on operands can be bypassed by later instructions whose operands are available.

RAW hazards are handled by delaying instructions in reservation stations until all their operands are available.

WAR and WAW hazards are handled by renaming registers in instructions by reservation station numbers.

Load and Store instructions to different memory addresses can be done in any order, but the relative order of a Store and accesses to the same memory location must be maintained. One way to perform ***dynamic disambiguation*** of memory references, is to perform effective address calculations of Loads and Stores in program order in the issue stage.

- Before issuing a Load from the instruction queue, make sure that its effective address does not match the address of any Store instruction in the Store buffers. If there is a match, stall the instruction queue until, the corresponding Store completes. (Alternatively, the Store could forward the value to the corresponding Load)
- Before issuing a Store from the instruction queue, make sure that its effective address does not match the address of any Store or Load instructions in the Store or Load buffers.

==========

3. If ADD and SUB take one cycle and MUL takes 7 cycles to execute, then what would be the first instruction to complete using Tomasula's algorithm on the following program?

```
MUL   R6, R4, R8
ADD   R2, R6, R7
STORE  R2, 8(R6)
ADD   R2, R3, R4
SUB   R4, R5, R2
LOAD R4, 16(R4)
ADD   R1, R2, R3
```

b) How does register renaming help the STORE instruction save the correct R2 value to memory while still allowing later instructions to execute?