Software Pipeline

Software Pipeline Algorithm

1. Determine the data dependencies
2. Identify the Processing and Management blocks of the loop body
    1. **Note :** When including labels, always exclude the instruction following the label!
3. Indicate perfromance losses due to the dependencies
    1. **Note:** Usually this is done on a MIPS32 and not on SS2, but read carefully!
4. Split stages on the performance losses
    1. **Note**: Don't split up a stage that is on a conditional block! (i.e. instructions after a Branch instruction)
    2. If there are multiple **writes** on one register on two different stages, **Rename** the registers and pay attention to the dependency path. This applies as well to registers being read on different stages without a direct dependency of the preceding stage.
    3. If after renaming a register there is a breakage on the transfer chain (i.e. stages in between writes to a regsiter), **Add a transfer instruction per register, PER STAGE** (`Or Rnew, Rold, R0`)
    4. **Note:** Avoid splitting a stage that needs additional transfer instructions, resulting in an overall performance loss.
5. Implement the software pipeline by **inversing** the order of the stages (S0, S1, S2 → S2, S1, S0)

    1. **Note:** Update the Store and Load instructions indexes accordingly: $\#\,Stage * -(step\ size)$
    2. Be careful **not** to update the index of an instruction that is not updated at all in the management block

    Before pipelining:
    S0: `_For: Lw r8, 0(r4)`
    S1: `Slti r8, r8, 5`
    S2: `Sw r8, 0(r4)`
    `Sw r9, 0(r5)`
    `Addiu r4, r4, 4 ; Management`

    After pipelining:
    S2 ($S(i-2)$): `_For: Sw r8, -8(r4)`
    `Sw r9, 0(r5)` Note: The index isn't updated here since R5 isn't being modified
    S1 ($S(i-1)$): `Slti r8, r8, 5`
    S0 ($S(i-0)$): `Lw r8, 0(r4)`
    `Addiu r4, r4, 4 ; Management`
6. Reorder instructions to **optimize** the code
    1. **Note**: It's better to break dependencies than to replace delayed slots
    2. **Note**: Remember not to move conditional instructions outside the conditional block

Pipeline Rules

The software pipeline has to follow the rules of a hardware pipeline. There are three rules to follow:

1. Stages need to be time-balanced.
2. Data can only pass from one stage to another using only one register.
3. Every component belongs to one and only one stage (only one stage can **write** to the register bank)

Notes

- I would write [Prologue] and [Epilogue] before and after the body of the loop just to be safe.
- We don't necessarily need to split in EVERY performance loss. Sometimes when splitting, we need to introduce additional transfer instructions which negate the performance gain of splitting a stage.
    - e.g. we split a stage between a dependency that costs 1 cycle, but we need to add two transfer instructions (2 cycle cost). In this case it is not worth it to split the stage.
- In a similar manner, when we have the option to choose where we should split a stage, it is better to maintain **symmetry** to follow the best as we can the pipeline rules. (R1. Stages need to be *time-balanced*.)