# REGISTER RENAMING

dynamic scheduling, with out of order completions: must preserve exception behaviour. there must not imprecise exceptions (when program sequence order is not maintained).

to allow out of order execution, ID pipe stage in split into 2 stages.

① issue - decode the instruction
check for structural hazards.

② read open - wait unit no data hazards. then read operands.

we must be able to distinguish when an instruction begins and when it completes, and in between we have, instr. execution.

dyn. sched.
↳ allows multiple instr. to be executed at the same time

without this capability a major advantage of dynamic scheduling is lost.

this requires multiple functional units and pipeline functional units, integer unit + floating point unit,


or both.

early days - called "score boarding" out of order execution, CDC 6600.

## TOMASOLO'S ALGO

WAR, WAW. hazards.
handles antidependencies and output dependencies by effectively renaming registers dynamically.

can be extended for handling dynamic speculative execution. (temporary stored in buffer).

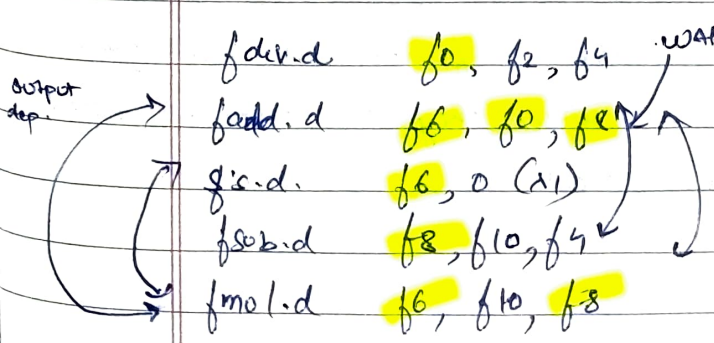
en tomasolo's algo can be extended to handle speculation.

## DYNAMIC SCHEDULING USING TOMASULO'S APPROACH

employed in IBM 360/91 (purely for commercial apps.) (some models for scientific apps.) FP -ops.

floating point unit has this tomasulo's.

the scheme minimize RAW hazards, and introduces register renaming. this register renaming is to avoid WAW and WAR hazards.



output dependencies between fadd.d and fmul.d.

WAR hazard in the use of f8 by fadd.d and fsub.d.

WAW hazard between fadd.d and fmul.d.

there are also 3 true data dependencies:

① between fdiv.d & fadd.d
② between fsub.d & fmul.d.
③ between fadd.d & fs.d.

temporary registers S and T.

| | |
|---|---|
| fdiv.d | f0, f2, f4 |
| fadd.d | S, f0, f8 |
| fsd | S, 0(x1) |
| fsub.d | T, f10, f4 |
| fmul.d | f6, f10, T. |

In this scheme, register renaming is provided by reservation stations, reservation stations other than centralized registers file.

① hazard detection and execution control are distributed.
② results are directly passed to functional units from the reservation stations where they are offered, rather than going through registers.

there are 3 steps in execution. (Tomasulo's algo).

1. issue (reservation station should be free).
2. execute (if both oper. are available, else wait in res. stat.)
3. write result (result will be put on common data bus).

there are 8 fields in the reservation stat.

each reservation station has 7 fields.

Op ① operation to be carried out (perform) on source operands $S_1$ and $S_2$. $Q_j$, $Q_k$.

$Q_j$, $Q_k$ → reservation stations that will produce the $V_j$, $V_k$ corresponding source operand.

→ the value of the source operands

A: → used to hold information for the memory address.

Busy: → indicates that this reservation station and its accompanying functional units are occupied.

the register file has a field $Q_i$

1. fld. f6, 32(x2)
2. fld f2, 44(x3) ← effective address.
3. fmul.d f0, f2, f4.
4. fsub.d f8, f2, f6.
5. fdiv.d f10, f0, f6.
6. fadd.d f6, f8, f2.

## INSTRUCTION STATUS

| Instruction | issue | execute | write result. |
|---|---|---|---|
| ① | ✓ | ✓ | ✓ |
| ② | ✓ | ✓ | |
| ③ | ✓ | | |
| ④ | ✓ | | |
| ⑤ | ✓ | | |
| ⑥ | ✓ | | |

## RESERVATION STATIONS

| Op name | busy | op | $V_j$ | $V_k$ | $q_j$ | $q_k$ |
|---|---|---|---|---|---|---|
| load1 | X | | | | | |
| load2 | ✓ | f2↑ | | | | |
| add1 | ✓ | sub | | mem[32+x2] | | load2 |
| add2 | ✓ | add | | | add1 | load2 |
| add3 | X | | | | | |
| mul1 | ✓ | mul | | regs[x4] mem[32+x2] | | load2 |
| mul2 | ✓ | div | | mem[32+x2] | | mul1 |

## REGISTER STATUS

| field. | F0 | F2 | f4 | f6 | f8 | f10 | f12 ... f30 |
|---|---|---|---|---|---|---|---|
| Q: | mult1 | load2 | | add2 | add1 | mult2 | |