Dynamic Scheduling

Or

yDanicm ceShuldngi

Dynamic Scheduling (or out-of-order execution)

- CDC 6600 scoreboard
  - Instruction storage added to each functional execution unit
  - Instructions issue to FU when no structural hazards, begin execution when dependences satisfied. Thus, instructions issued to different FUs can execute out of order.
  - "scoreboard" tracks RAW, WAR, WAW hazards, tells each instruction when to proceed.
  - No forwarding
  - No register renaming
- Tomasulo (IBM 360/91) [reservation stations]
- Instruction Queue (MIPS R10000, Alpha 21264, …)

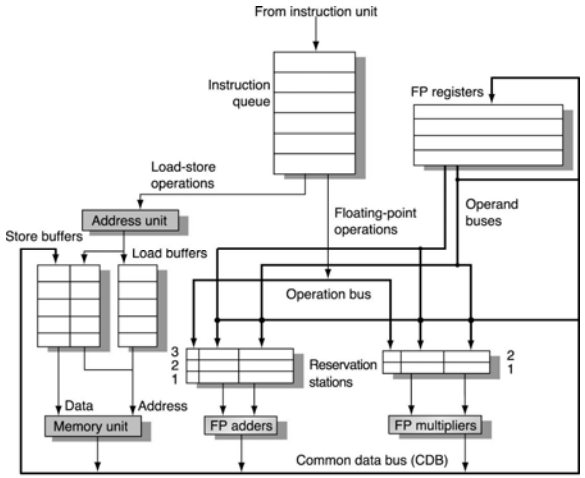
Tomasulo Algorithm

- For IBM 360/91 about 3 years after CDC 6600
- Goal: High Performance without special compilers
- Differences between IBM 360 & CDC 6600 ISA
  - IBM has only 2 register specifiers/instr vs. 3 in CDC 6600
  - IBM has 4 FP registers vs. 8 in CDC 6600
  - Implications?

Differences between Tomasulo Algorithm & Scoreboard

- Control & buffers distributed with Function Units vs. centralized in scoreboard; called "reservation stations"
  => instrs schedule themselves
- Registers in instructions replaced by pointers to reservation station buffer
  scoreboard => registers primary operand storage
  Tomasulo => reservation stations as operand storage
- HW renaming of registers to avoid WAR, WAW hazards
  Scoreboard => both source registers read together
  Tomasulo => each register read as soon as available.
- Common Data Bus broadcasts results to all FUs
  RS's (FU's), registers, etc. responsible for collecting own data off CDB
- Load and Store Queues treated as FUs as well

Tomasulo Organization



Reservation Station Components

RS1 | Op | Q_j | Q_k | V_j | V_k | R_j | R_k | Busy

Op—Operation to perform in the unit (e.g., + or –)

Qj, Qk—Reservation stations producing source registers

Vj, Vk—Value of Source operands

Rj, Rk—Flags indicating when Vj, Vk are ready

Busy—Indicates reservation station is busy

Register result status—Indicates which functional unit will write each register, if one exists. Blank when no pending instructions that will write that register.

Three Stages of Tomasulo Algorithm

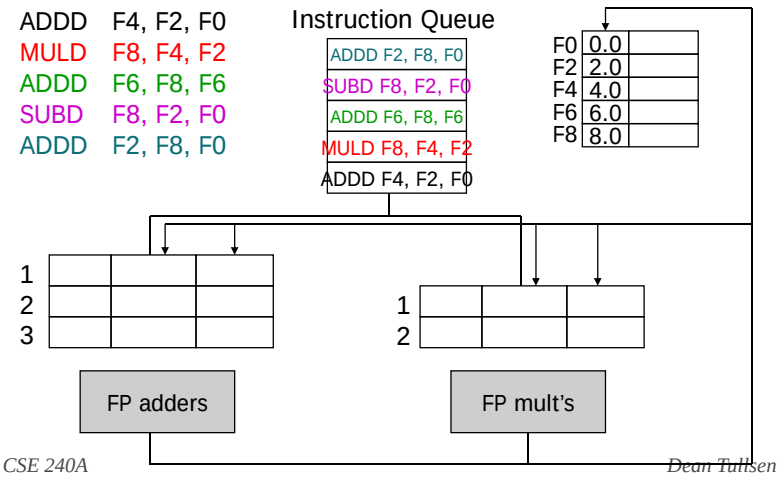
1. Issue—get instruction from FP Inst Queue
   If reservation station free, the IQ issues instr & sends operands (renames registers).
2. Execution—operate on operands (EX)
   When both operands ready then execute;
   if not ready, watch CDB for result
3. Write result—finish execution (WB)
   Write on Common Data Bus to all waiting units;
   mark reservation station available.

Tomasulo Example

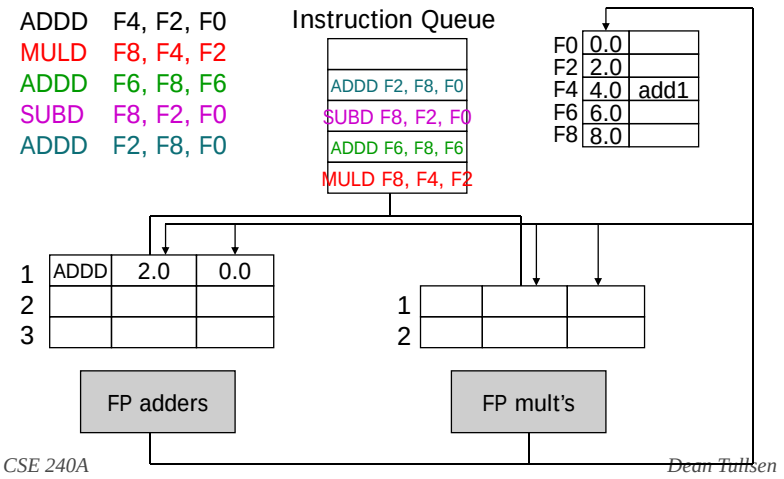
```
ADDD   F4, F2, F0
MULD   F8, F4, F2
ADDD   F6, F8, F6
SUBD   F8, F2, F0
ADDD   F2, F8, F0
```

Multiply takes 10 clocks, add/sub take 4

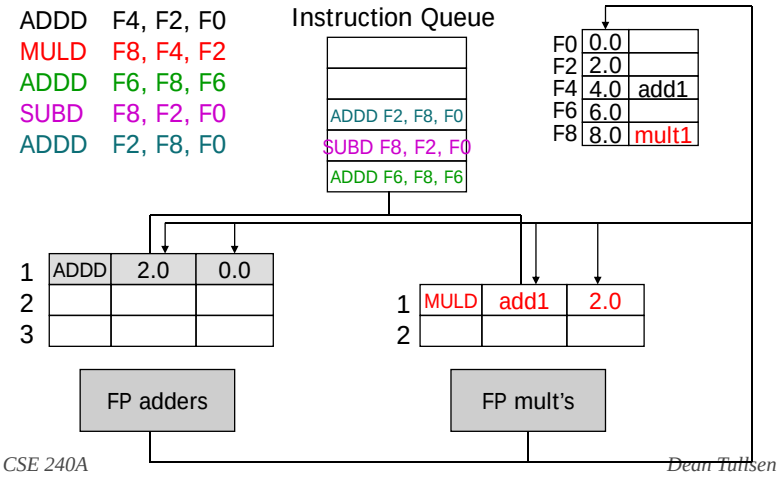
Tomasulo – cycle 0



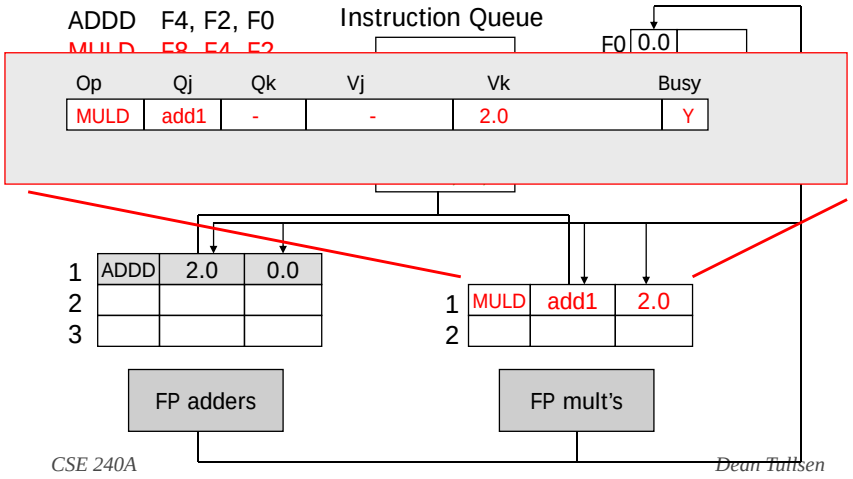
Tomasulo – cycle 1



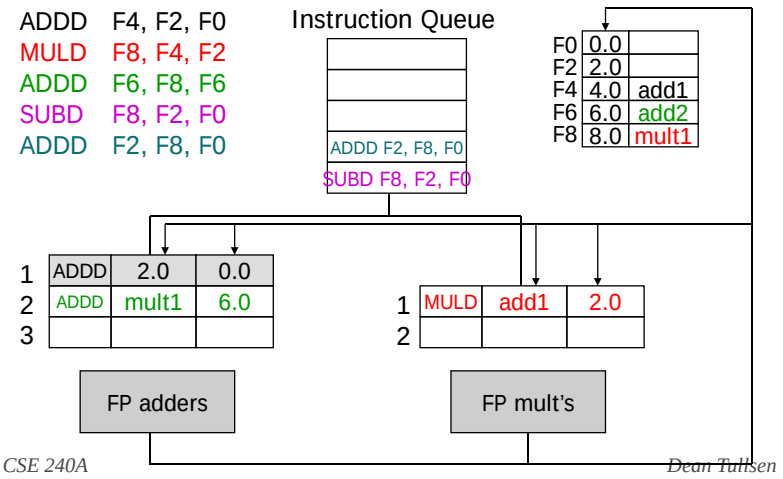
Tomasulo – cycle 2



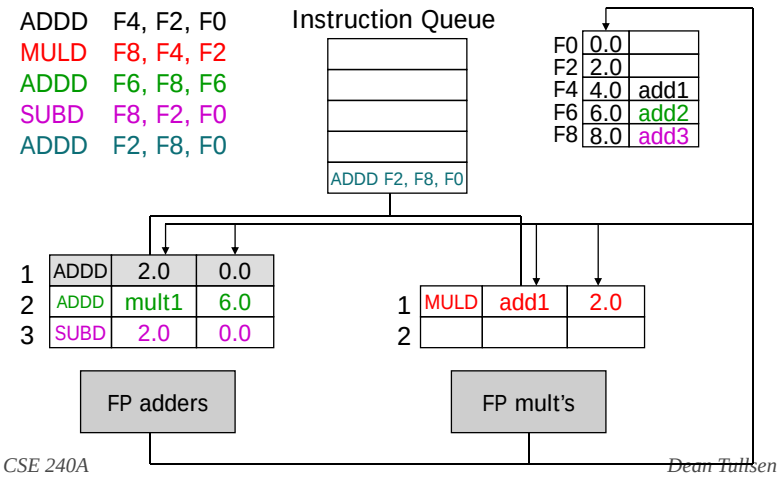
Tomasulo – cycle 2



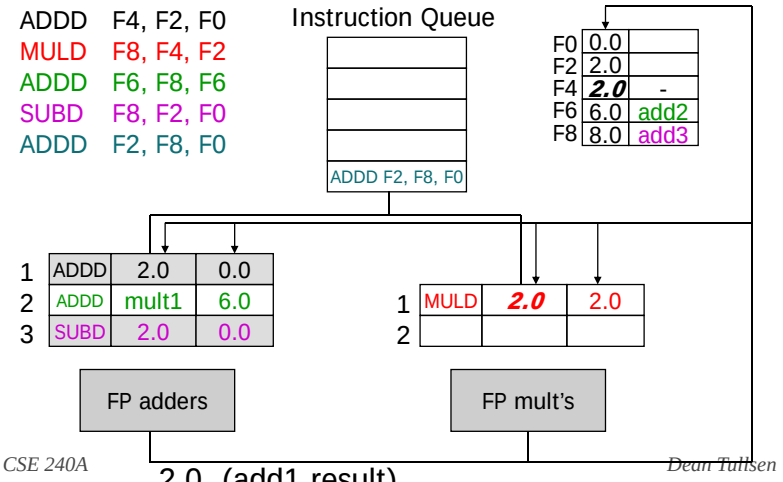
Tomasulo – cycle 3



Tomasulo – cycle 4

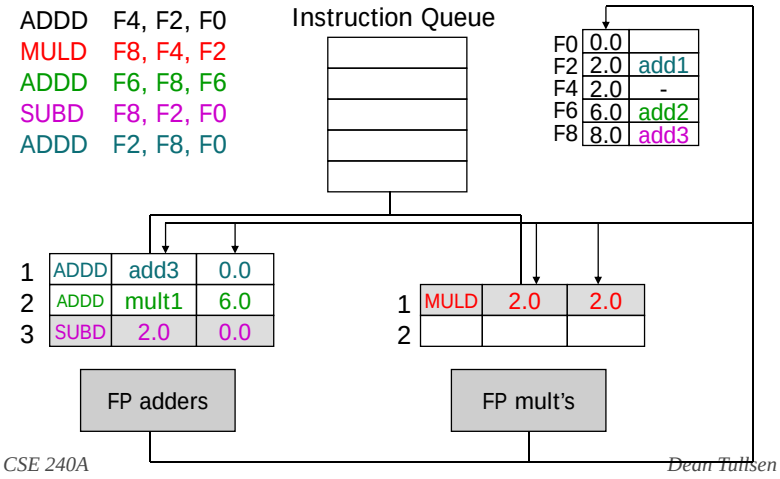


Tomasulo – cycle 5

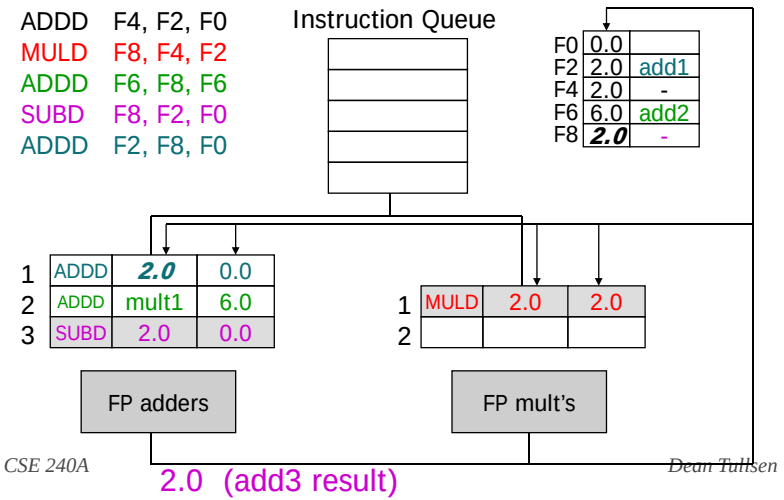


2.0 (add1 result)

Tomasulo – cycle 6

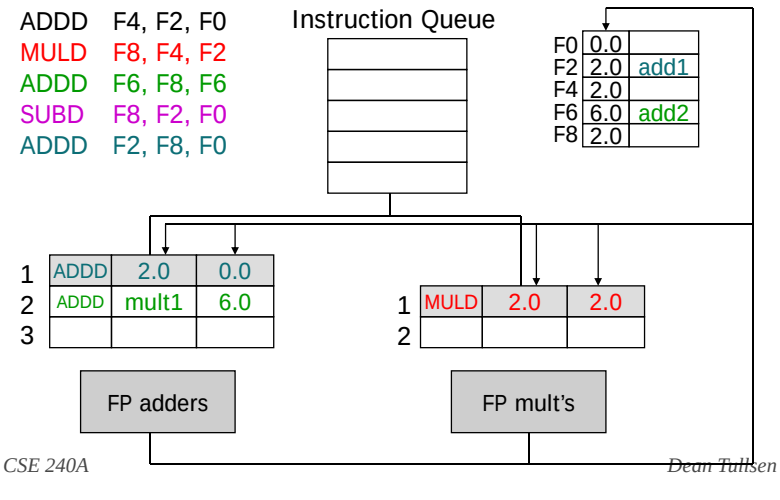


Tomasulo – cycle 8

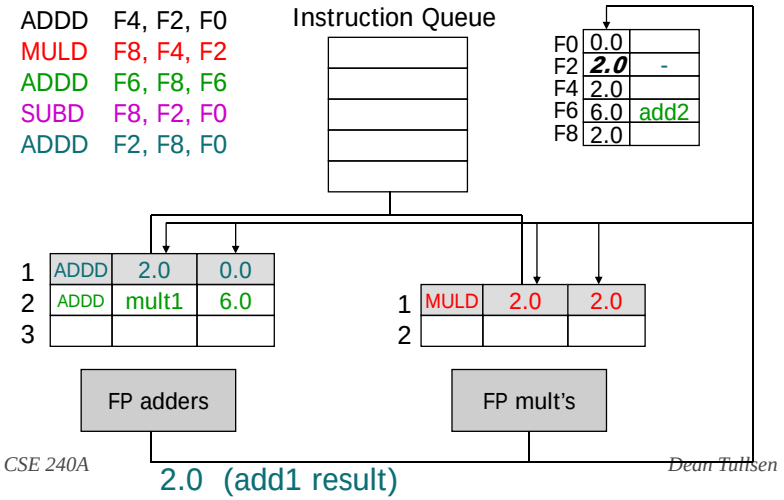


2.0 (add3 result)

Tomasulo – cycle 9

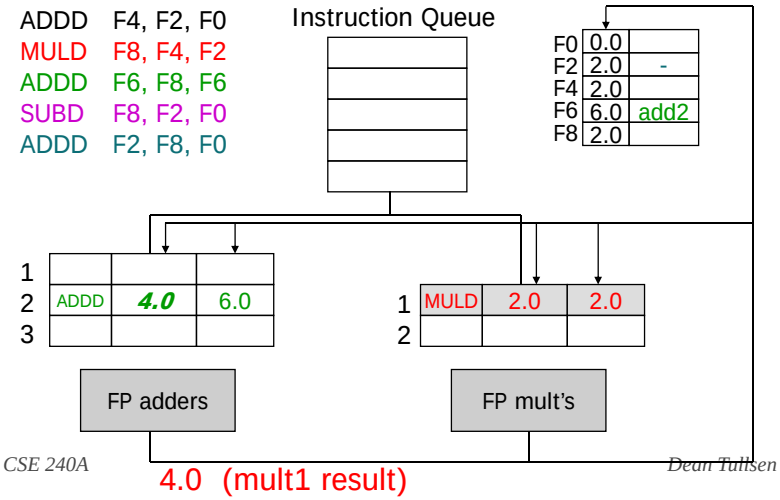


Tomasulo – cycle 12



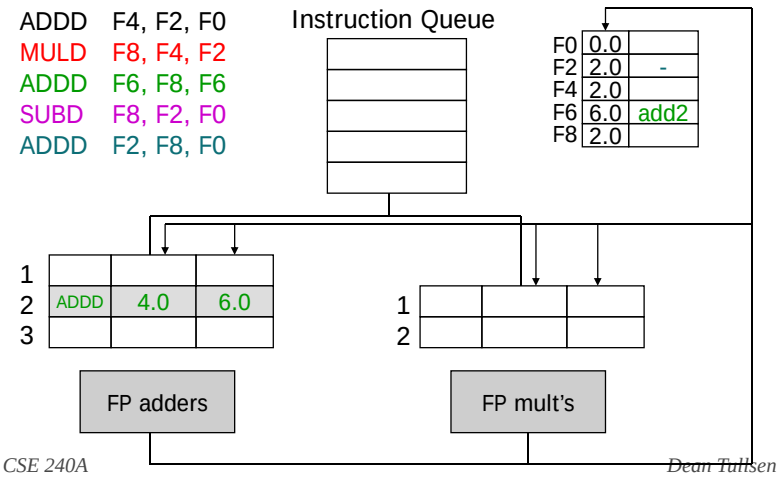
2.0 (add1 result)

Tomasulo – cycle 15

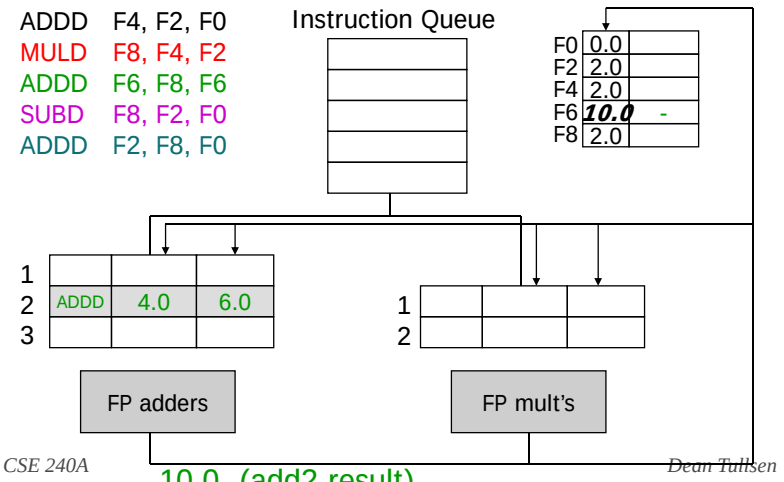


4.0 (mult1 result)

Tomasulo – cycle 16



Tomasulo – cycle 19



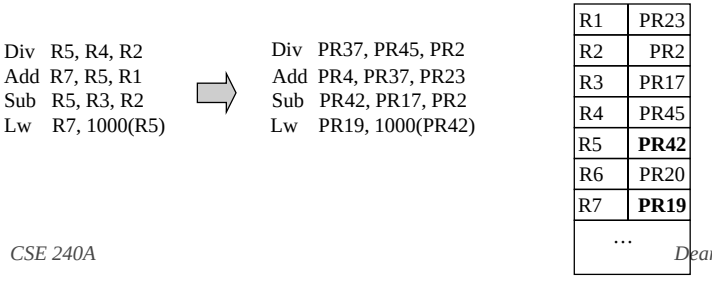
10.0 (add2 result)

Tomasulo Summary

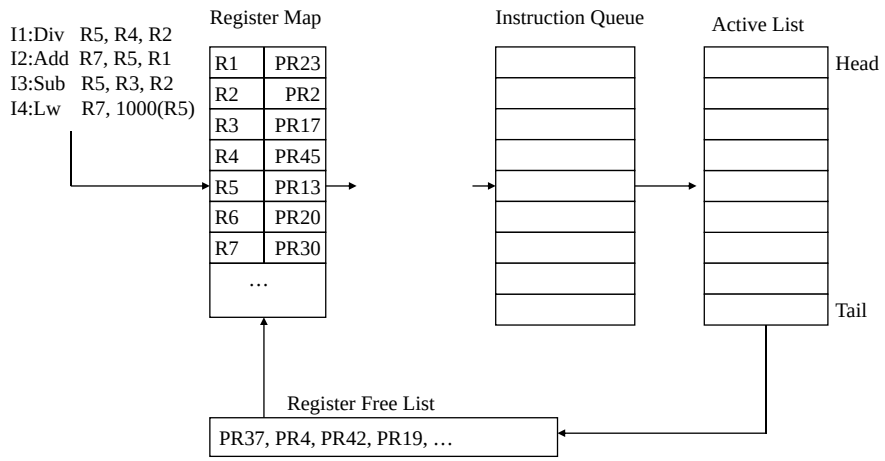
- Prevents Register as bottleneck
- Avoids WAR, WAW hazards of Scoreboard
- Allows loop unrolling in HW
- Not limited to basic blocks (provided branch prediction)
- Lasting Contributions
  - Dynamic scheduling
  - Register renaming (in what way does the register name *change*?)
  - Load/store disambiguation

Modern Architectures

- Alpha 21264+, MIPS R10K+, Pentium 4 use an ***instruction queue***.
- They use *explicit register renaming*. Registers are not read until instruction dispatches (begins execution). Register renaming ensures no conflicts.

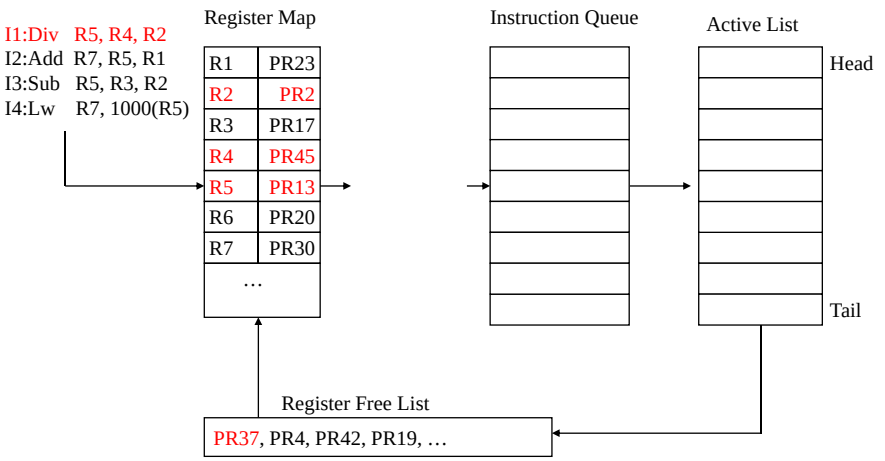


MIPS R10000, some detail

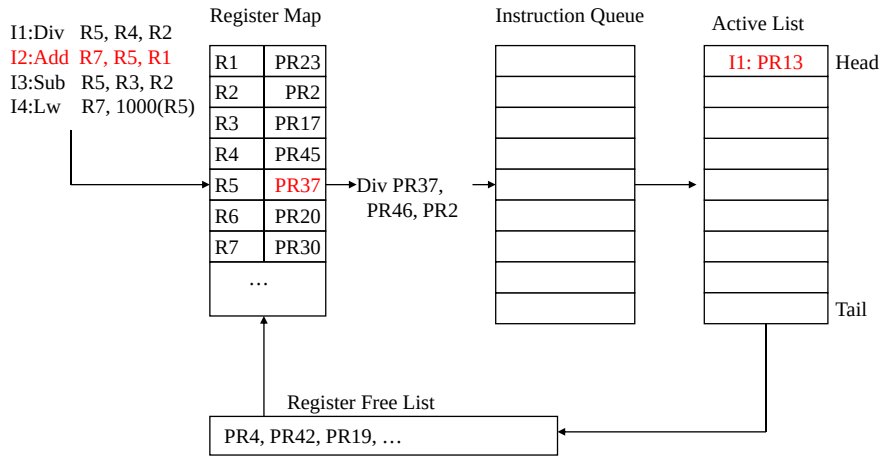


Active list – maintains original instruction order, determines when a physical register can be freed.

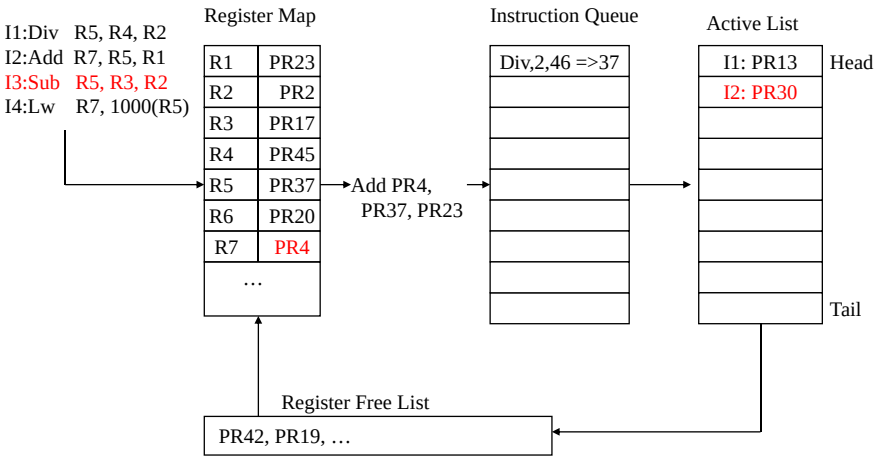
MIPS R10000, some detail



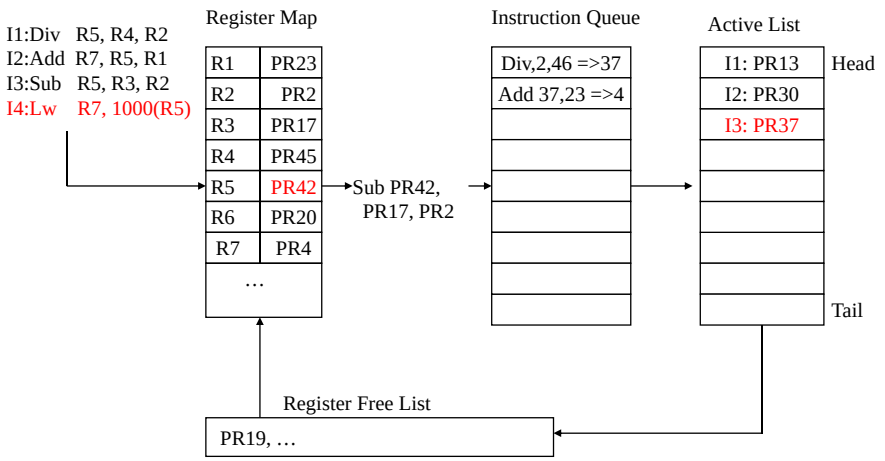
MIPS R10000, some detail



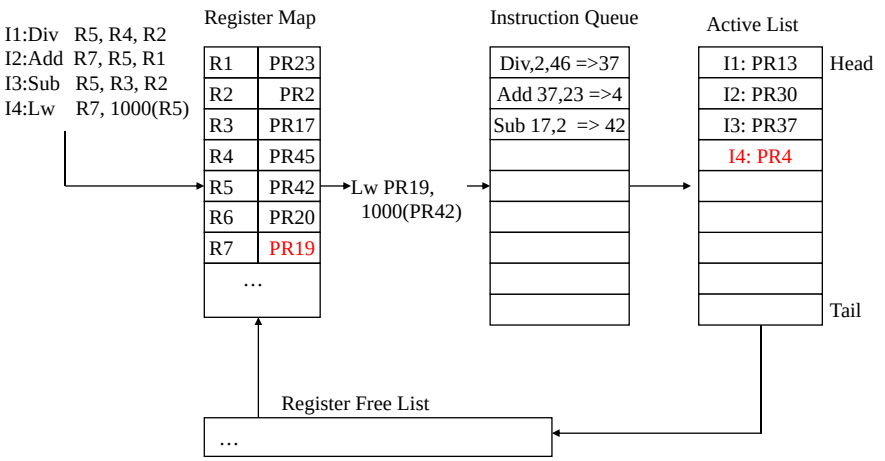
MIPS R10000, some detail



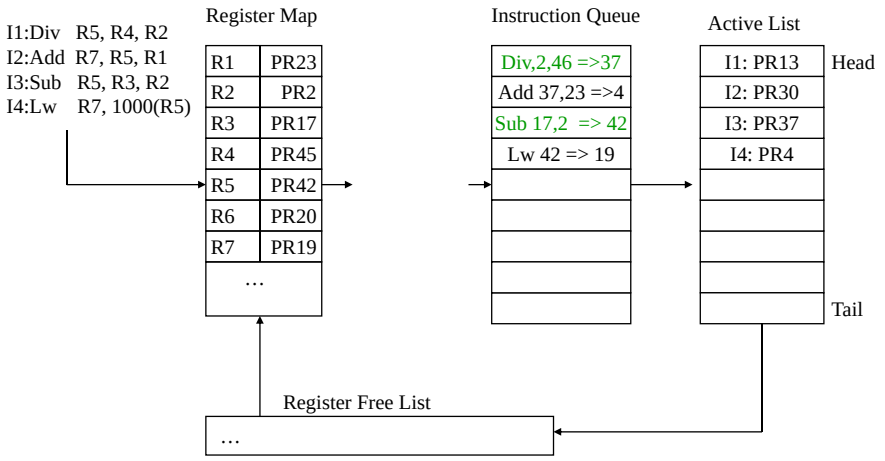
MIPS R10000, some detail



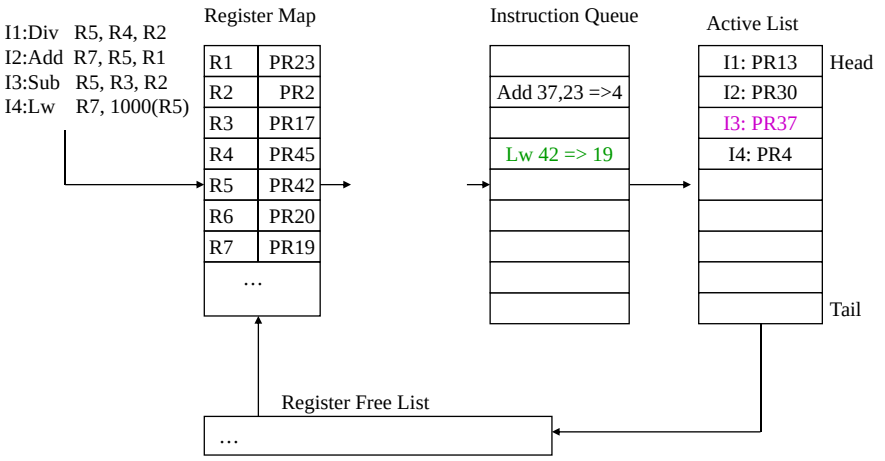
MIPS R10000, some detail



MIPS R10000, some detail

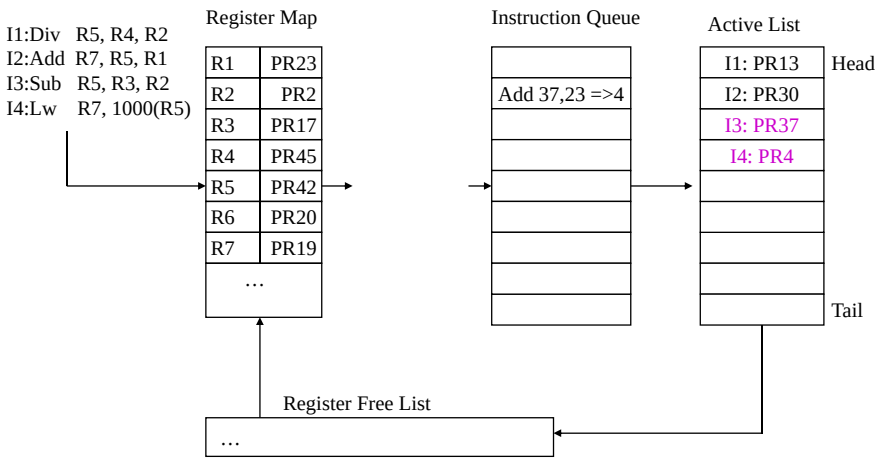


MIPS R10000, some detail



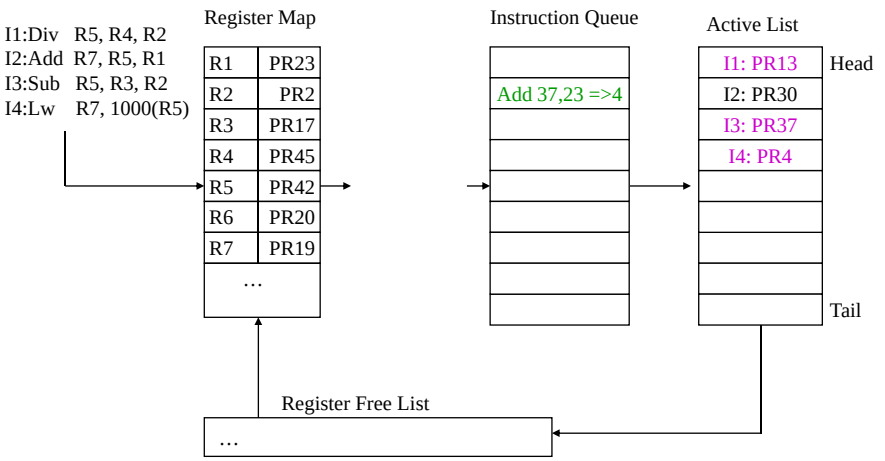
I3, producing register 42, completes, broadcasts a completion signal to IQ

MIPS R10000, some detail



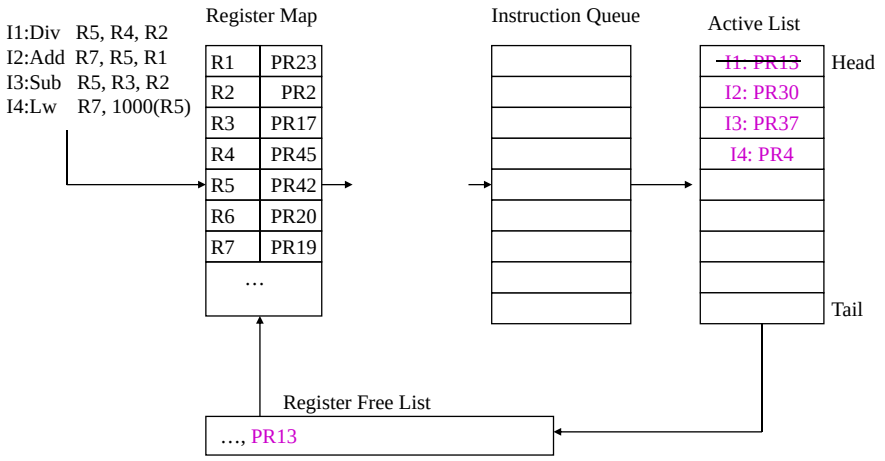
I4, producing register 19, completes, broadcasts a completion signal to IQ

MIPS R10000, some detail



I1, producing register 37, completes, broadcasts a completion signal to IQ

MIPS R10000, some detail



I2, producing register 4, completes, broadcasts a completion signal to IQ
I1 commits.

Dynamic Scheduling Key Points

- Dynamic scheduling is code motion in HW.
- Dynamic scheduling can do things SW scheduling (static scheduling) cannot.
- Register renaming eliminates WAW, WAR dependencies.
- To get cross-iteration parallelism, we need to eliminate WAW, WAR dependencies.