# 08.Dynamic Scheduling Scoreboard

Scheduling separates dependent instructions:

- Static – performed by the compiler
- Dynamic – performed by the hardware
  At compiler level we can extract the code and work on these information but some information aren't known at compile time and we have to operate dynamically with a cost in comsumption.
  Advantages of dynamic scheduling
- Handles dependences unknown at compile time
- Simplifies the compiler, more efficient
- Optimization is done at run time, technically fast as it is done in HW
- It allows compiled code to run efficiently on a different pipeline
  Disadvantages of dynamic scheduling
- Cannot eliminate true data dependences. An operation cannot start if the data aren't there
- significant increase in hardware complexity and power consumption with problems in power constraint systems

| Technique | Reduces |
|---|---|
| Dynamic scheduling | Data hazard stalls |
| Dynamic branch pred. | Control stalls |
| Multiple issue | $CPI_{ideal}$ |
| Speculation | Data and control stalls |

These are all HW techniques and so come with a cost and so we need to understand if we need them.
When is it Safe to Issue an Instruction?
Suppose a data structure keeps track of all the instructions in all the functional units
The following checks need to be made before the Issue stage can dispatch an instruction

- Is the required function unit available?
- Is the input data available? → RAW?
- Is it safe to write the destination? → WAR? WAW?

- Is there a structural conflict at the WB stage?
  If we have different functional units that needs to be executed they can potentially runned in parallel. We need to check that the input data are available.

---

## A Data Structure for Correct Issues
## Keeps track of the status of Functional Units

| Name | Busy | Op | Dest | Src1 | Src2 |
|---|---|---|---|---|---|
| Int | | | | | |
| Mem | | | | | |
| Add1 | | | | | |
| Add2 | | | | | |
| Add3 | | | | | |
| Mult1 | | | | | |
| Mult2 | | | | | |
| Div | | | | | |

The instruction i at the Issue stage consults this table

| | |
|---|---|
| FU available? | check the busy column |
| RAW? | search the dest column for i's sources |
| WAR? | search the source columns for i's destination |
| WAW? | search the dest column for i's destination |

An entry is added to the table if no hazard is detected;
An entry is removed from the table after Write-Back

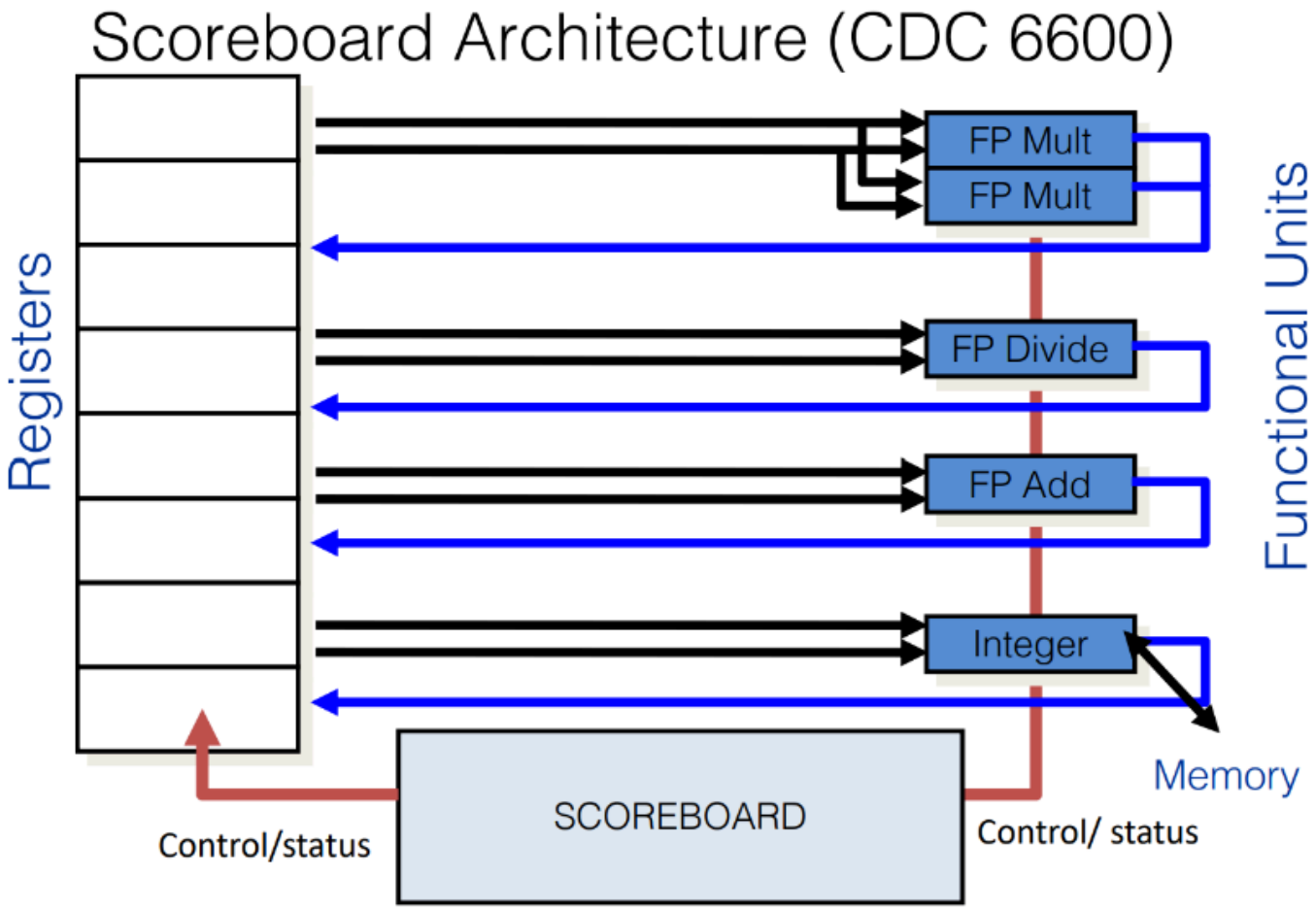
We have the resources, we have a column representing if a unit is available, then we have the destination register and the source register.
When we have an operation if the source register are in the destination register we cannot start(RaW). Also if I have a destination register is the source of an operation in the table the operation has to be stalled. If I pass all the checks I can start the operation and when the operation is ended I can remove it from the table.
The problem around dynamic scheduling I have data dependencies that cannot be resolved with optimisation in the single issue in the pipeline causing stalls(Hazards are problem from the program, Stalls are problem from the hardware).
The idea of dynamic scheduling is that if there is an operation stalling it will wait but if there is an operation after it that can be executed it will be executed. This is done automatically at runtime. Out of order execution and order of operation completion. This could potentially introduce further dependencies but the table will solve automatically these new hazards.

## CDC6600 Scoreboard

Instructions dispatched in-order to functional units provided no structural hazard or WAW

- Stall on structural hazard, no functional units available

- Only one pending write to any register
  Instructions wait for input operands (RAW hazards) before execution
- Can execute out-of-order
  Instructions wait for output register to be read by preceding instructions (WAR)
- Result held in functional unit until register free



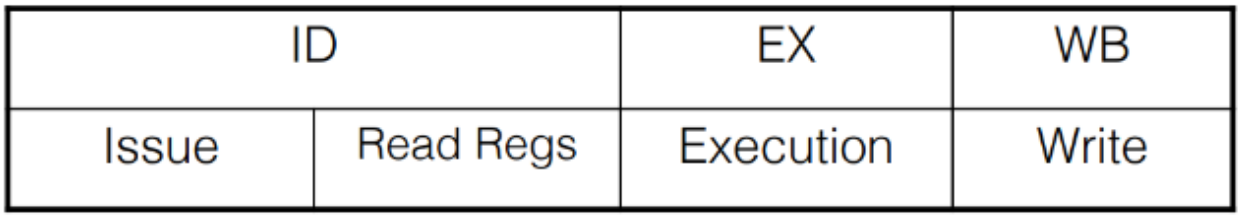

Parallel functional units that operate indipendently and we have a unique table/point where we analyze the hazards.

***Scoreboard Operation***

Scoreboard centralizes hazard management

- Every instruction goes through the scoreboard. The scoreboard can individue when an operation will read the registers.
- Scoreboard determines when the instruction can read its operands and begin execution. Can decide when a stalling operation can resume
- Monitors changes in hardware and decides when a stalled instruction can execute
- Controls when instructions can write results

New pipeline




***Scoreboard Scheme***

ID stage divided in two parts:

- Issue (decode and check structural hazard). Structurally the resource is not available
- Read Operands (wait until no data hazards). Data need to be available. Then orocede in the execution stage
  In-order issue(every operation enter in the issue stage in order) BUT out-of-order read-operands(needs to be available at the time)
  Scoreboard allows instructions without dependencies to execute

## Four Stages of Scoreboard Control

1. Issue
   Decode instructions & check for structural hazards.
   Instructions issued in program order (for hazard checking)
   If a functional unit for the instruction is free and no other active instruction has the same destination register (WAW), the scoreboard issues the instruction to the functional unit and updates its internal data structure.
   If a structural or a WAW hazard exists, then the instruction issue stalls, and no further instructions will issue until these hazards are cleared.
2. Read Operands
   Wait until no data hazards, then read operands
   A source operand is available if:
   - no earlier issued active instruction will write it or
   - A functional unit is writing its value in a register
     When the source operands are available, the scoreboard tells the functional unit to proceed to read the operands from the registers and begin execution.
     RAW hazards are resolved dynamically in this step, and instructions may be sent into execution out of order. No forwarding of data in this model
3. Execution
   Operate on operands
   The functional unit begins execution upon receiving operands. When the result is ready, it notifies the scoreboard that it has completed
   execution.
   FUs are characterized by:
   - latency (the effective time used to complete one operation)

- Initiation interval (the number of cycles that must elapse between issuing two operations to the same functional unit).

4. Write result
   Finish execution
   Once the scoreboard is aware that the functional unit has completed execution, the scoreboard checks for WAR hazards. If none, it writes results. If WAR, then it stalls the instruction. Assume we can overlap issue and write

## Scoreboard Implications

Solution for WAW:

- Detect hazard and stall issue of new instruction until the other instruction completes
  No register renaming
  Need to have multiple instructions in execution phase and this implies Multiple execution units or pipelined execution units
  Scoreboard keeps track of dependences and state of operations

## Scoreboard structure: three parts

1. Instruction status indicates the status of the operations in the execution plane
2. Functional Unit status
   Indicates the state of the functional unit (FU):
   Busy: Indicates whether the unit is busy or not
   Op: The operation to perform in the unit (+,-, etc.)
   Fi: Destination register
   Fj, Fk: Source register numbers
   Qj, Qk: Functional units producing source registers
   Rj, Rk: Flags indicating when Fj, Fk are ready.
   There are flags that represent if the resource registers are ready and it means that there is another operation that is going to produce one or more of its operands.
3. Register result status
   Indicates which functional unit will write each register.
   Blank if no pending instructions will write that register

### CDC 6600 Scoreboard

Key idea of Scoreboard: Allow instructions behind stall to proceed (Decode Þ Issue Instruction & Read Operands)
Speedup of 2.5 w.r.t. no dynamic scheduling
Speedup 1.7 by reorganizing instructions from compiler
BUT slow memory (no cache) limits benefit
Limitations of 6600 scoreboard:

- No forwarding hardware

- Limited to instructions in basic block (small window)
- Small number of functional units (structural hazards), especially integer/load store units
- Do not issue on structural hazards
- Wait for WAR hazards
- Prevent WAW hazards