# Dynamic Instruction scheduling

**Presented By**

**Dr. Banchhanidhi Dash**

**School of Computer Engineering**
**KIIT University**

- **Static Scheduling(by compiler)**

  Compiler scheduling technique (loop unrolling, software pipelining)

  » separate dependent instructions

  » minimize the number of hazard and stalls

**Dynamic Scheduling( by Hardware )**

Dynamic Scheduling is a technique in which the hardware rearranges the instruction execution to reduce the stalls.

Dynamic scheduling, as its name implies, is a method in which the hardware determines which instructions to execute, as opposed to a statically scheduled machine, in which the compiler determines the order of execution

# The Key idea of Dynamic Scheduling

- Key Idea: Allow instructions behind stall to proceed. => Instructions executing in parallel. There are multiple execution units, so use them.

```
DIVD   F0, F2, F4
ADDD   F10, F0, F8
SUBD   F8, F8, F14
```

Even though ADDD stalls, the SUBD has no dependencies and can run.

- Enables out-of-order execution => out-of-order completion

***Dynamic pipeline scheduling overcomes the limitations of in-order pipelined execution by allowing out-of-order instruction execution.***

# Dynamic Scheduling With A Scoreboard

- The scoreboard is a centralized hardware mechanism
  - In order to execute an instruction as soon as its operands are available and no hazard conditions that prevent it.
- It dynamically constructs the dependency graph by hardware for a window of instructions as they are issued in program order.
- A scoreboard is a "data structure" that provides the information necessary for all pieces of the processor to work together.

# The Key idea of Scoreboards

**Scoreboard : a bookkeeping technique**

- Out-of-order execution divides ID stage:
- **Issue**: decode instructions, check for structural hazards
- **Read operands**: wait until no data hazards, then read operands
- Instructions execute whenever not dependent on previous instructions and no hazards.
- Example: CDC 6600

# CDC 6600 Scoreboard

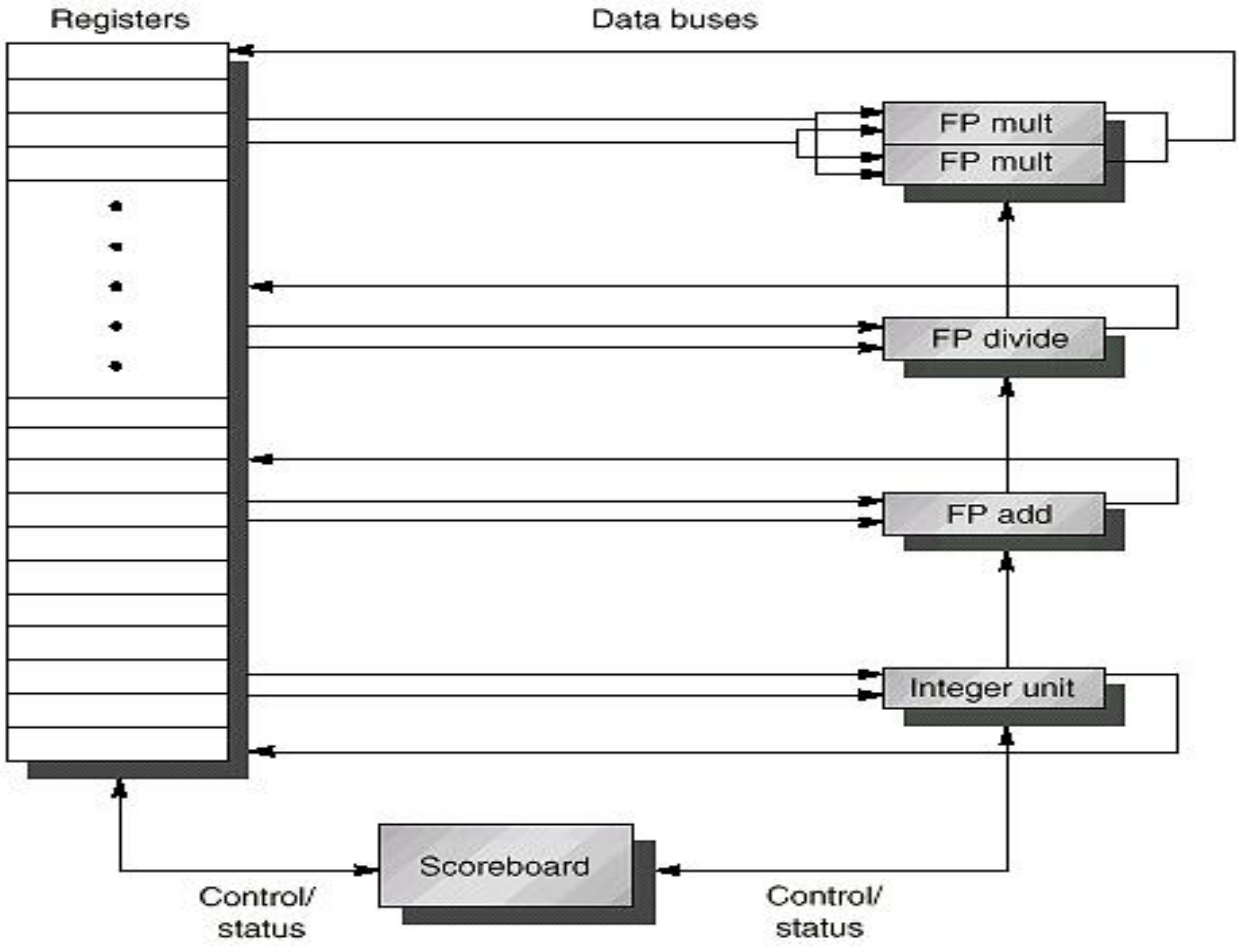
- Speedup 1.7 from compiler; 2.5 by hand
  BUT slow memory (no cache) limits benefit
- First implement for dynamic scheduling
  (though limited)
- Limitations of 6600 scoreboard as for dynamic scheduling
  - Stall on name dependence (WAR and WAW), which is not really necessary
  - Instruction parallelism is limited by # of function units
  - No forwarding hardware

**First used in CDC6600 in 1963. Our example has been modified to fit for MIPS.**

**CDC had 4 FP units, 5 memory reference units, 7 integer units.**

**MIPS scorboard : has 2 FP multiply, 1 FP adder, 1 FP divider, 1 integer.**

# Typical Scoreboard Structure for MIPS




2 FP multiply, 1 FP adder, 1 FP divider, 1 integer

# Scoreboard Implications

- **Out-of-order completion** →WAR, WAW
- **Solutions for WAR:**
  - Stall writeback until regs have been read
  - Read regs only during Read Operands stage
- **Solution for WAW:**
  - Detect hazard and stall issue of new instruction until other instruction completes
- No register renaming!
- Need to have multiple instructions in execution phase
  - multiple execution units or pipelined execution units
- Scoreboard keeps track of dependencies between instructions that have already issued.
- Scoreboard replaces ID, EX, WB with 4 stages

# Using A Scoreboard: 4 stages

1. Issue —decode instructions & check for structural & WAW hazards (ID1)

   If a functional unit for the instruction is free (*no structural hazards*) and no other active instruction has the same destination register (*no WAW*), the scoreboard issues the instruction to the functional unit and updates its internal data structure.

   If a structural or WAW hazard exists, then the instruction issue stalls, and no further instructions will issue until these hazards are cleared.

Always done in program order

2. Read operands —wait until no data hazards, then read operands (ID2)

   A source operand is available if no earlier issued active instruction is going to write it, or if the register containing the operand is being written by a currently active functional unit (*no RAW*).

   When the source operands are available, the scoreboard tells the functional unit to proceed to read the operands from the registers and begin execution. The scoreboard resolves RAW hazards dynamically in this step, and instructions may be sent into execution out of order.

Can be done out of program order

# Using A Scoreboard: 4 stages

3. Execution —operate on operands (EX)

   The functional unit begins execution upon receiving operands. When the result is ready, it notifies the scoreboard that it has completed execution.

4. Write result —finish execution (WB)

   Once the scoreboard is aware of the fact 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.

   Example:

```
DIVD   F0,F2,F4
ADDD   F10,F0,F8
SUBD   F8,F8,F14
```

   Scoreboard would stall SUBD until ADDD reads operands

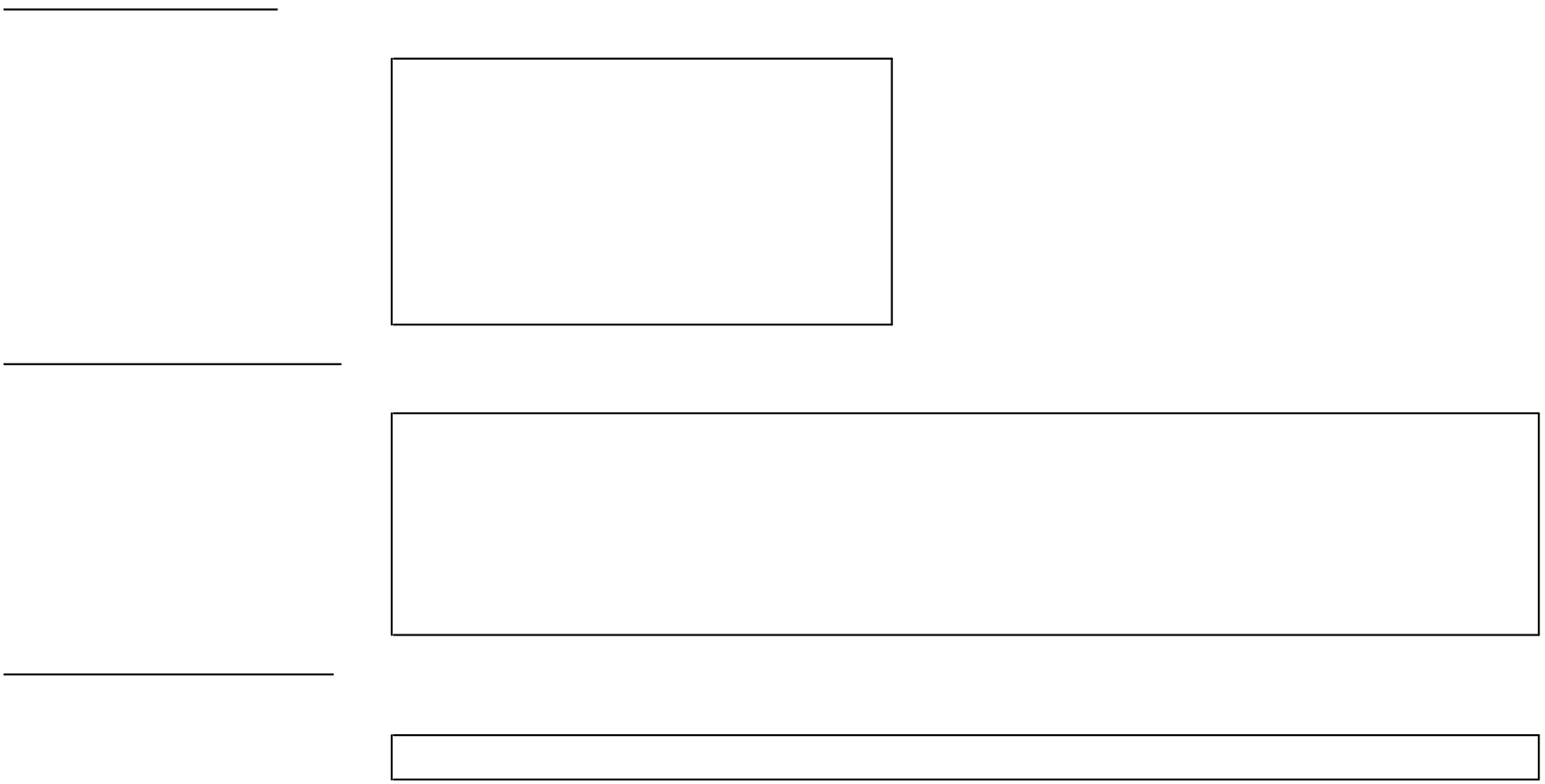
# Using A Scoreboard: 3 parts

1. Instruction status—which of 4 steps the instruction is in

2. Functional unit status—Indicates the state of the functional unit (FU). 9 fields for each functional unit
   - Busy—Indicates whether the unit is busy or not
   - Op—Operation to perform in the unit (e.g., + or –)
   - Fi—Destination register
   - Fj, Fk—Source-register numbers
   - Qj, Qk—Functional units producing source registers Fj, Fk
   - Rj, Rk—Flags indicating when Fj, Fk are ready.

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

# Scoreboard Example



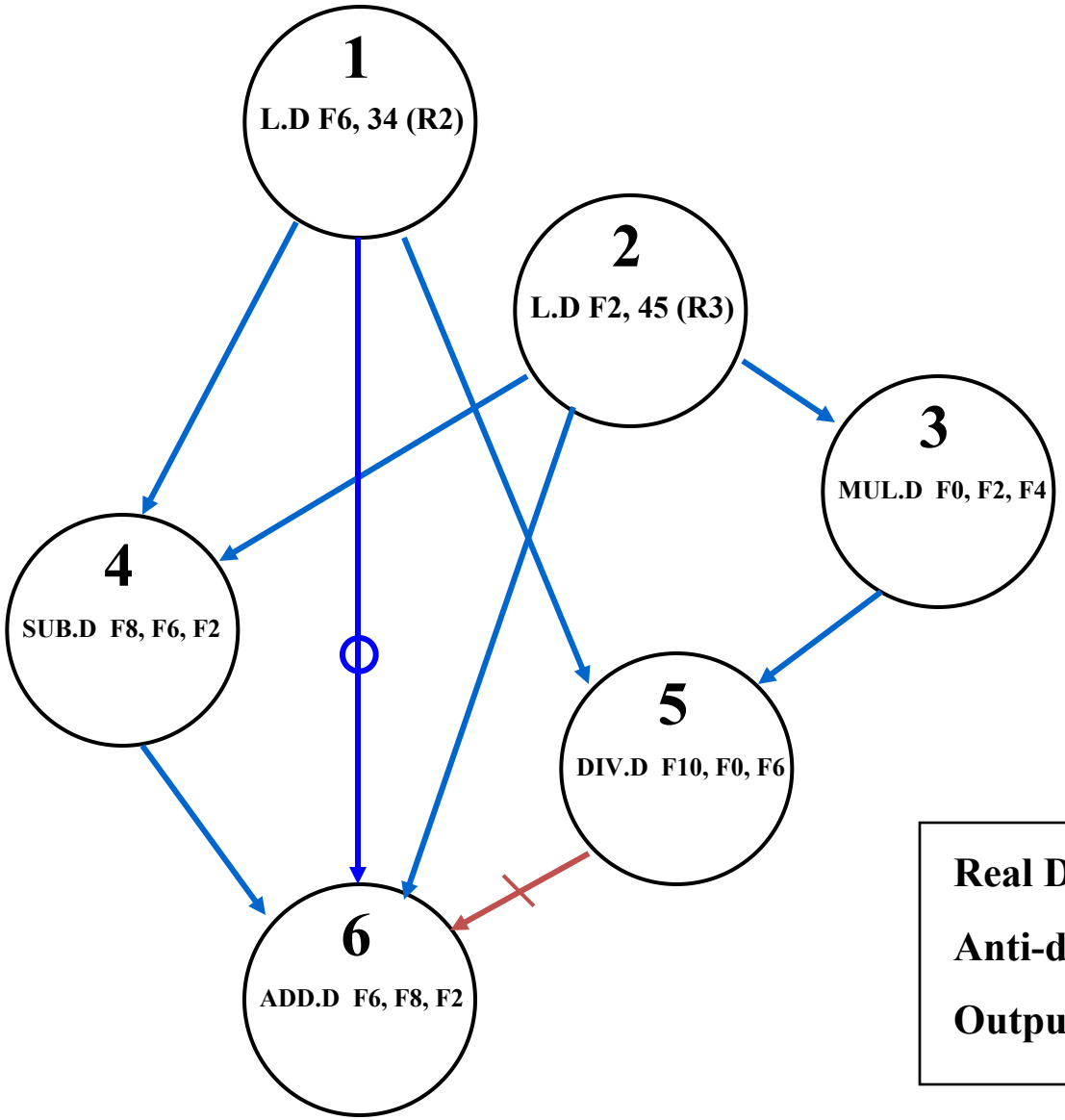
# A Scoreboard Example

The following code is run on the MIPS with a scoreboard given earlier with:

| Functional Unit (FU) | # of FUs | EX cycles |
|---|---|---|
| Integer | 1 | 1 |
| Floating Point Multiply | 2 | 10 |
| Floating Point add | 1 | 2 |
| Floating point Divide | 1 | 40 |

1. L.D F6, 34(R2)
2. L.D F2, 45(R3)
3. MUL.D F0, F2, F4
4. SUB.D F8, F6, F2
5. DIV.D F10, F0, F6
6. ADD.D F6, F8, F2

# Dependency Graph For Example Code




**Example Code**

| | | |
|---|---|---|
| 1 | L.D | F6, 34(R2) |
| 2 | L.D | F2, 45(R3) |
| 3 | MUL.D | F0, F2, F4 |
| 4 | SUB.D | F8, F6, F2 |
| 5 | DIV.D | F10, F0, F6 |
| 6 | ADD.D | F6, F8, F2 |

**Date Dependence:**
(1, 4) (1, 5) (2, 3) (2, 4)
(2, 6) (3, 5) (4, 6)

**Output Dependence:**
(1, 6)

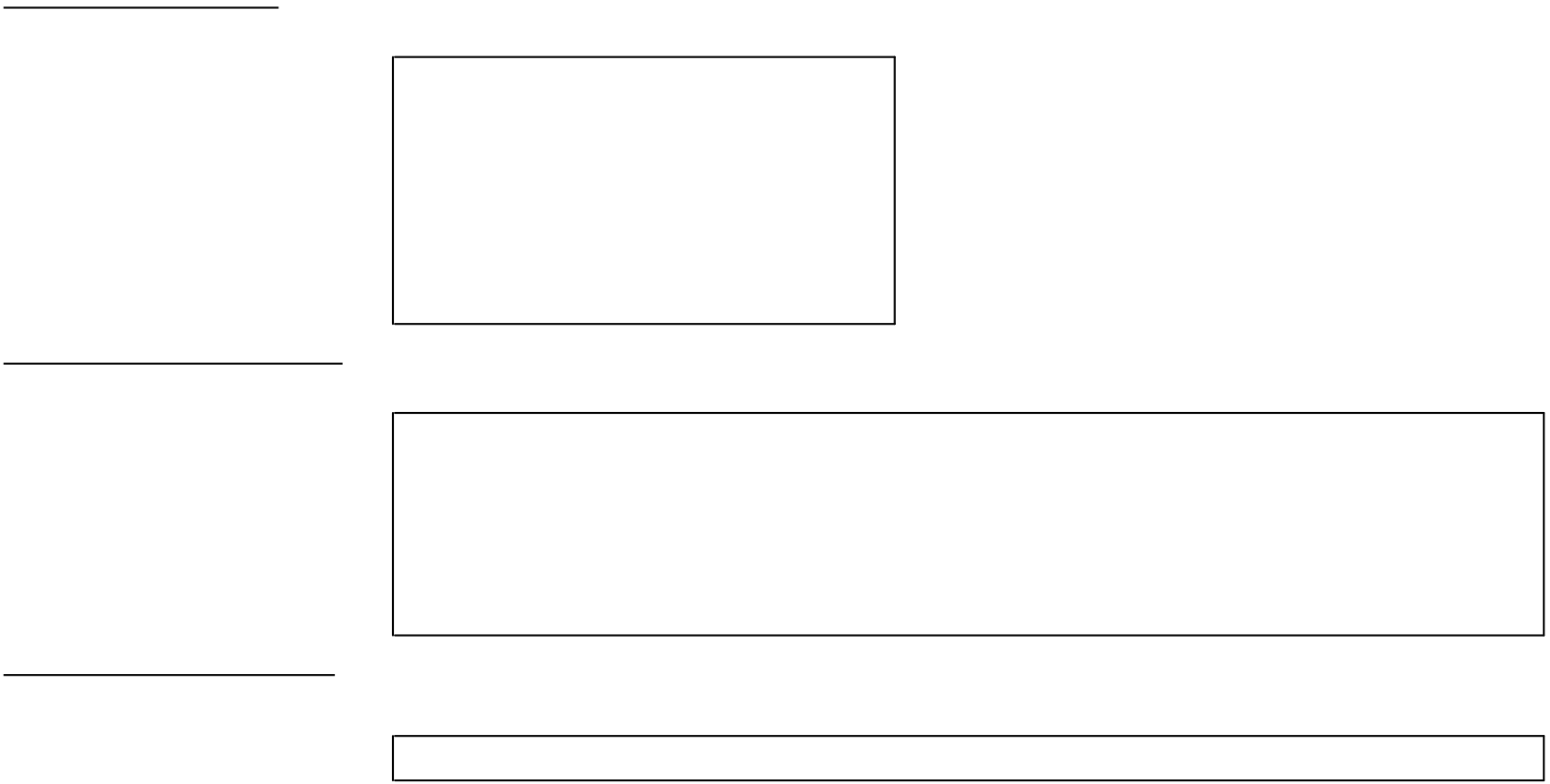
**Anti-dependence:**
(5, 6)

**Real Data Dependence (RAW)**

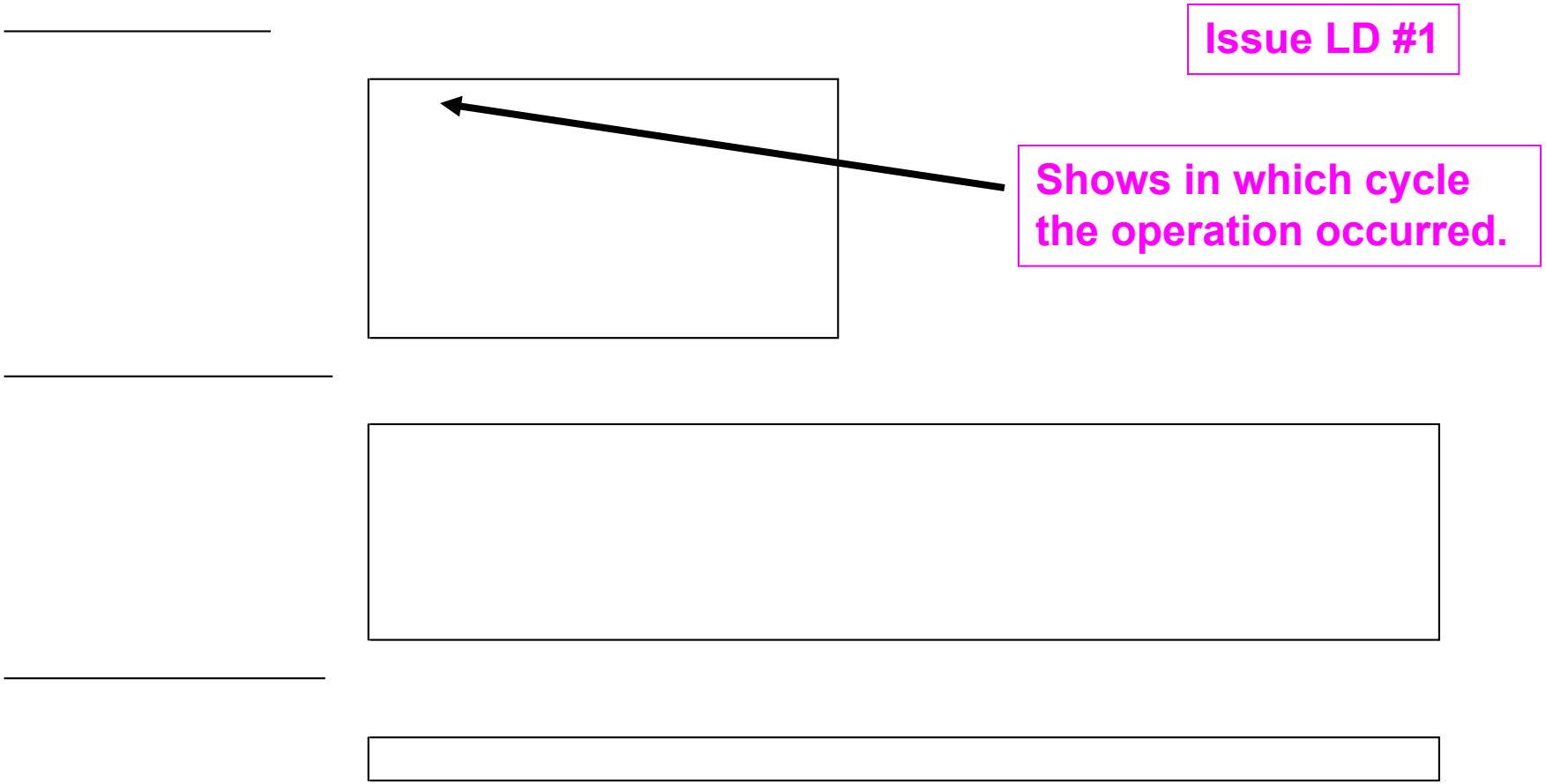
**Anti-dependence (WAR)**

**Output Dependence (WAW)**

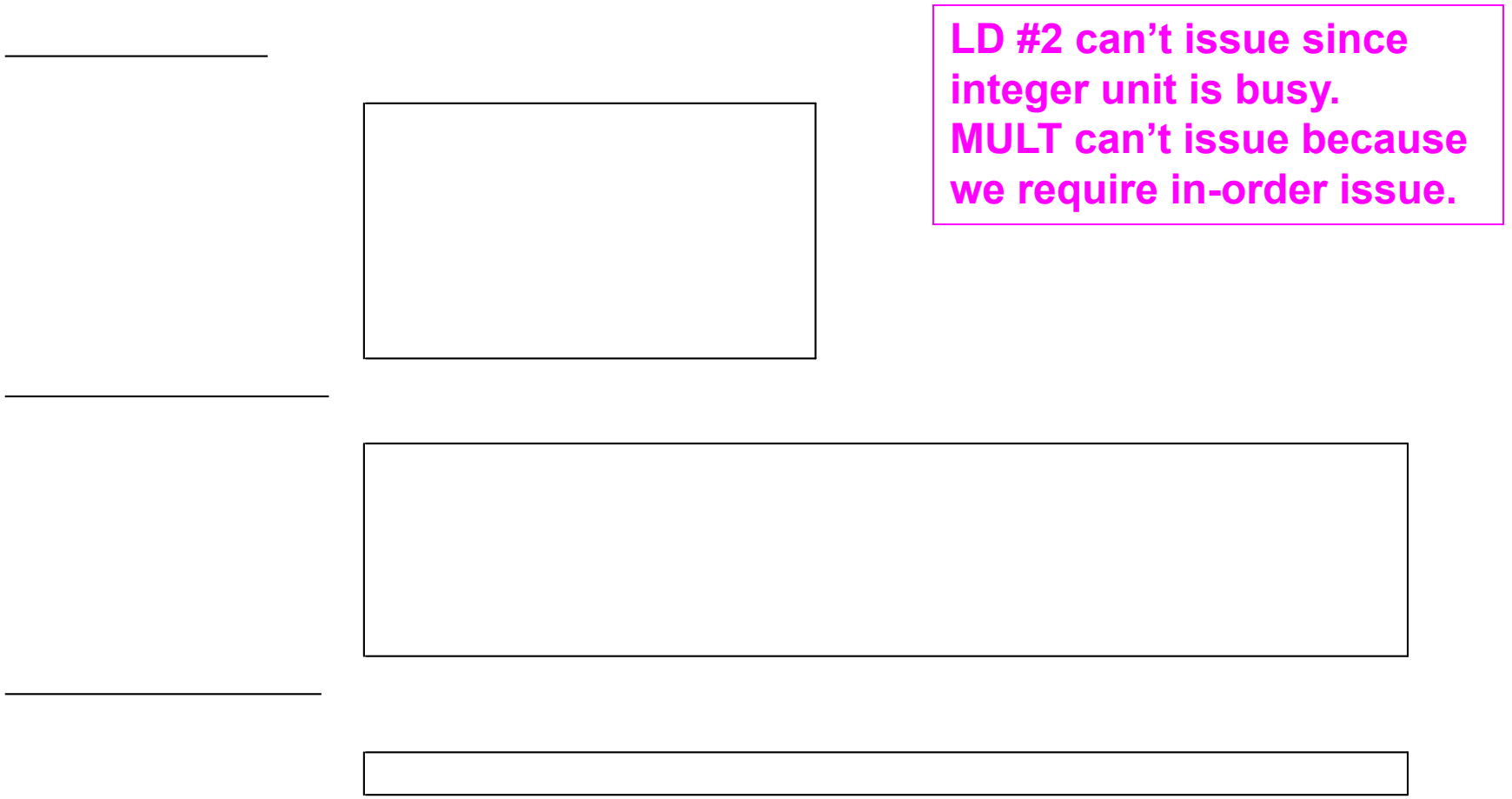
# Scoreboard Example



# Scoreboard Example Cycle 1

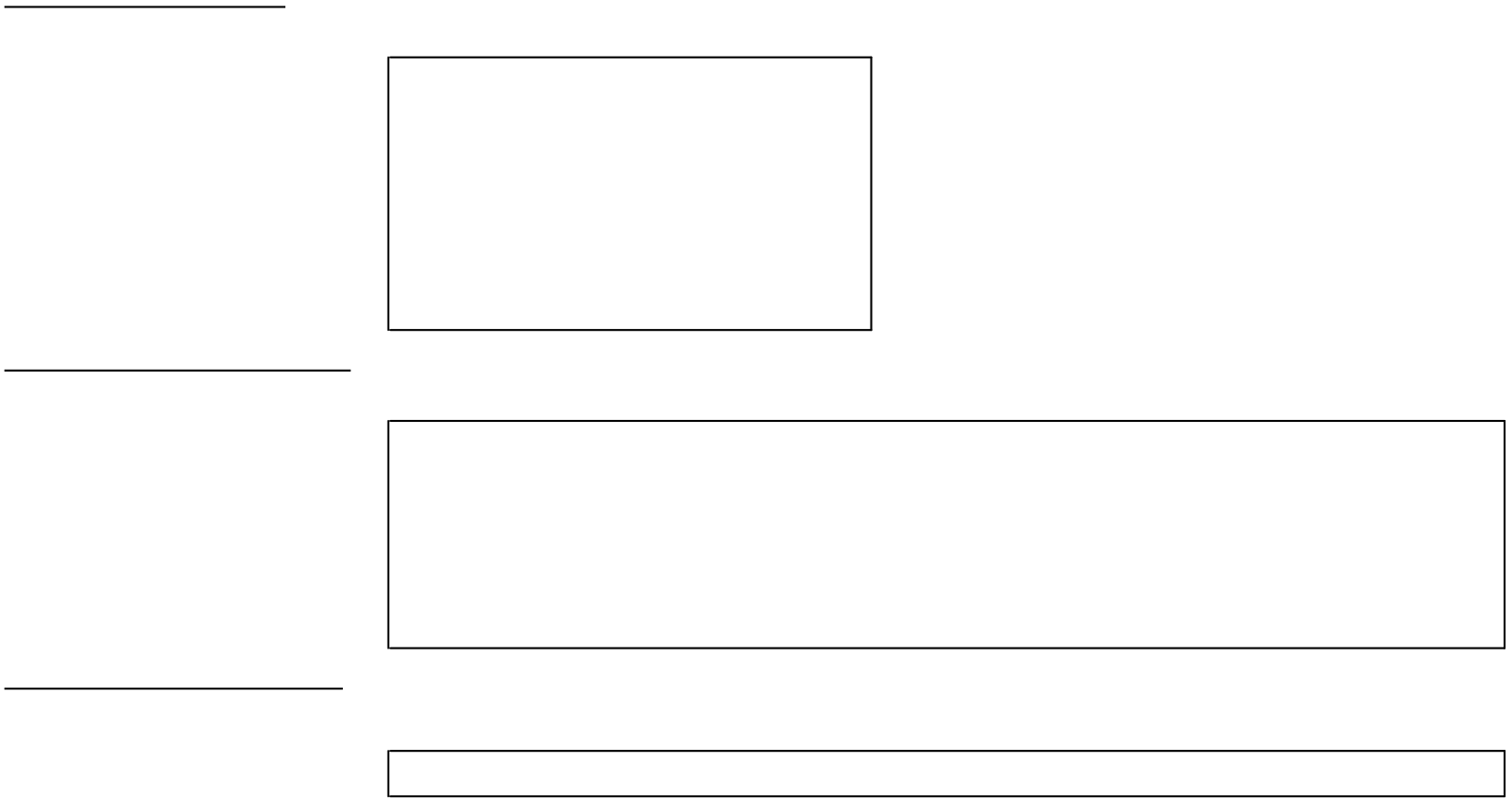


# Scoreboard Example Cycle 2

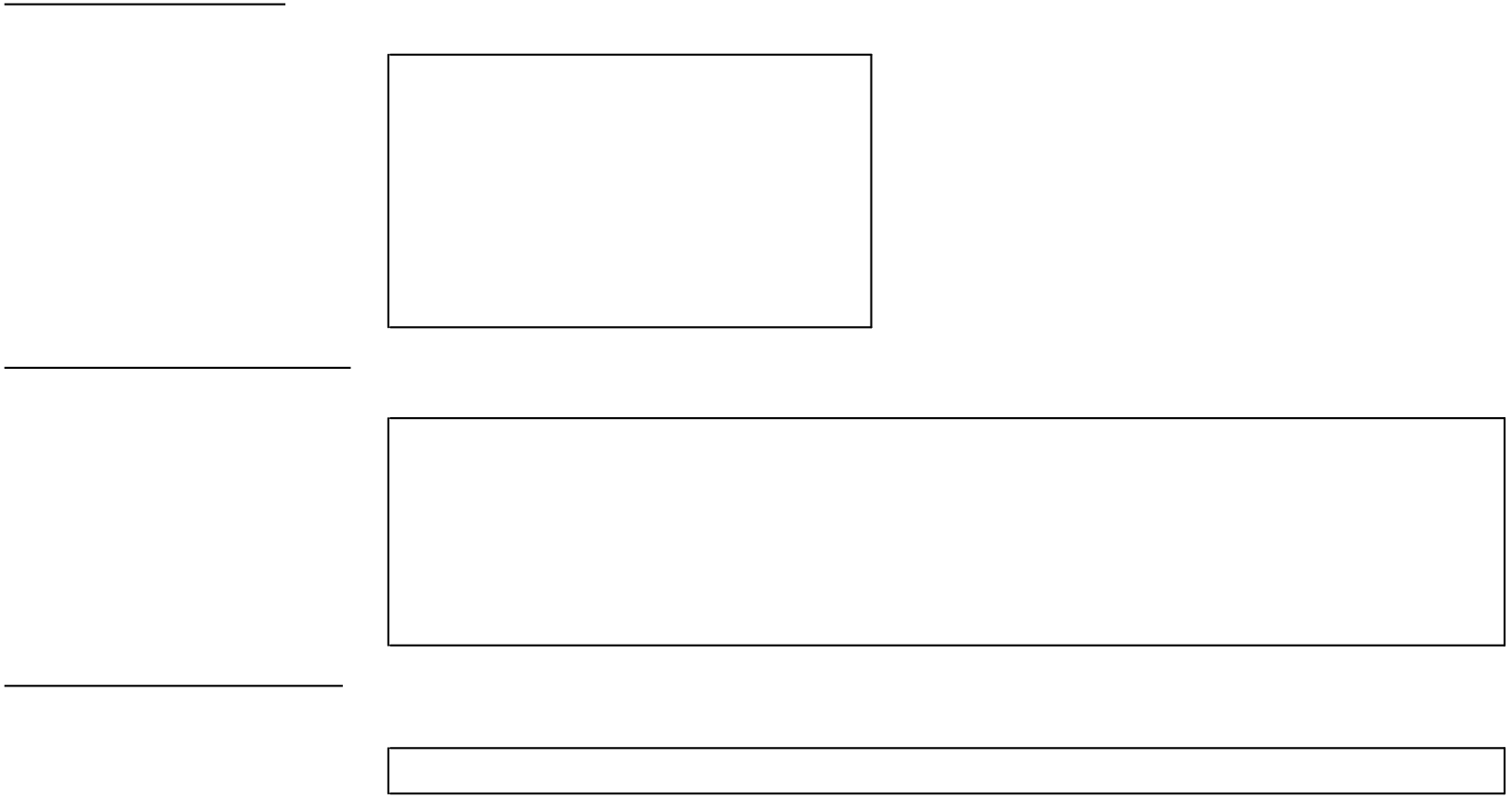


**LD #2 can't issue since integer unit is busy. MULT can't issue because we require in-order issue.**

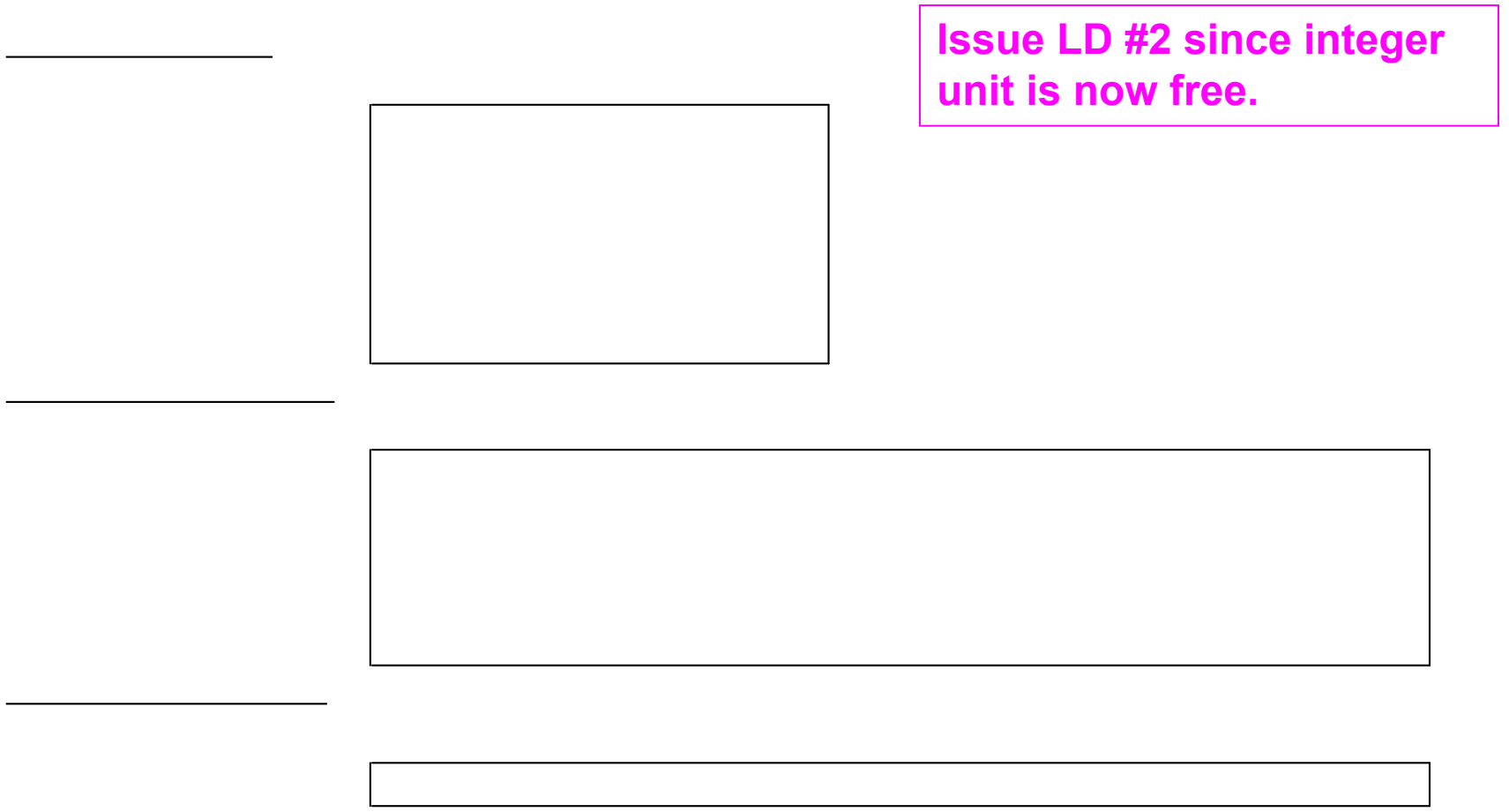
# Scoreboard Example Cycle 3



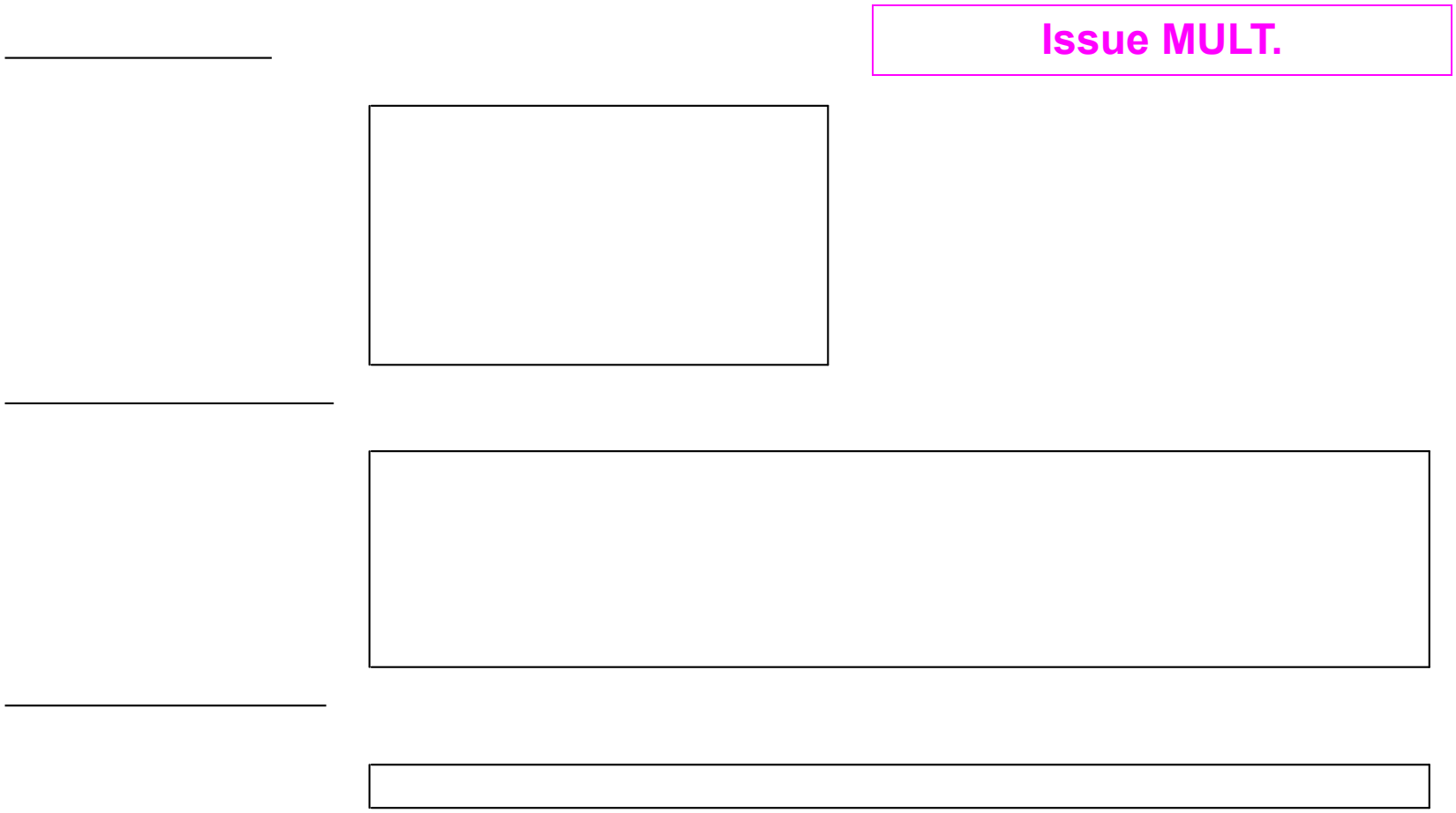
# Scoreboard Example Cycle 4



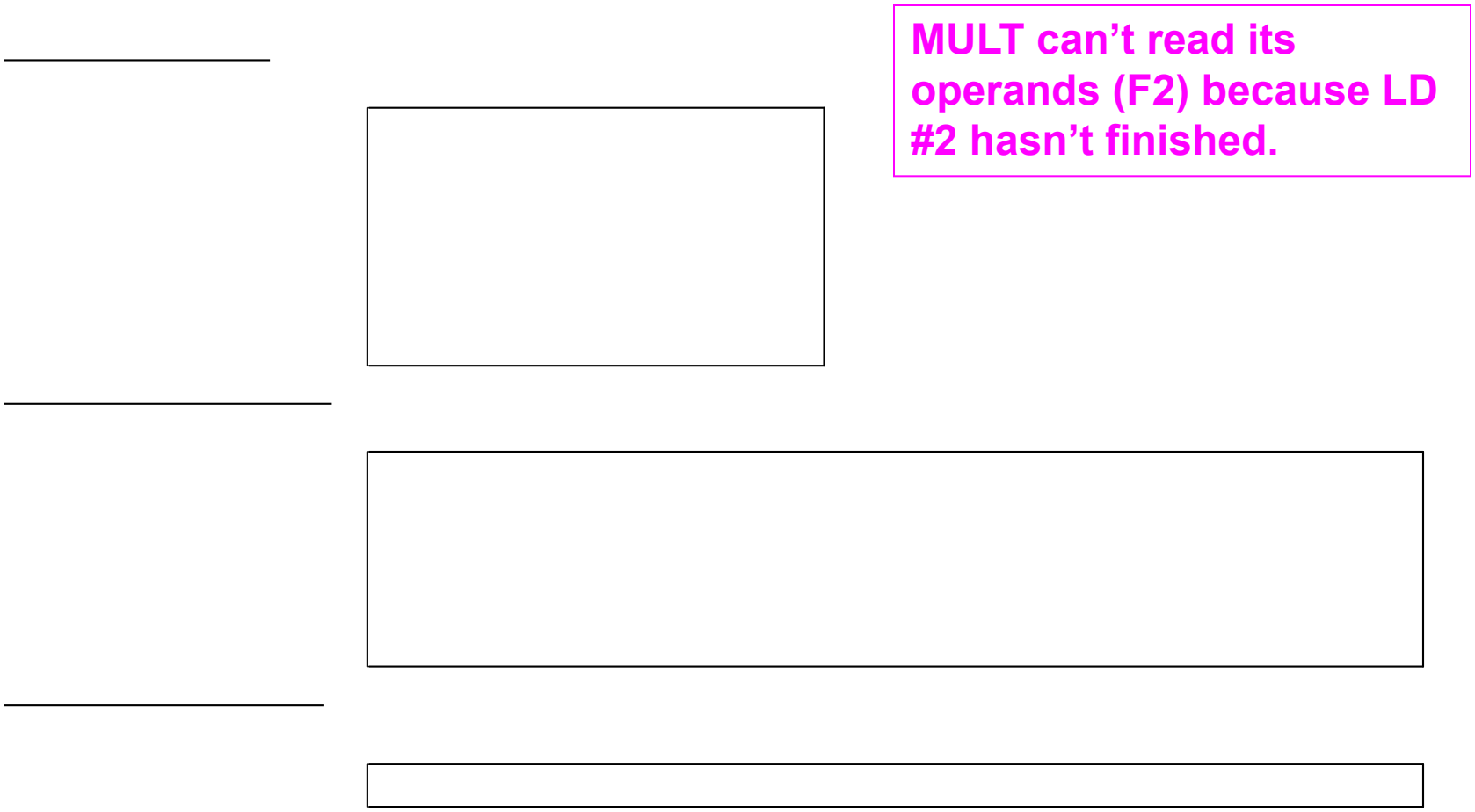
# Scoreboard Example Cycle 5



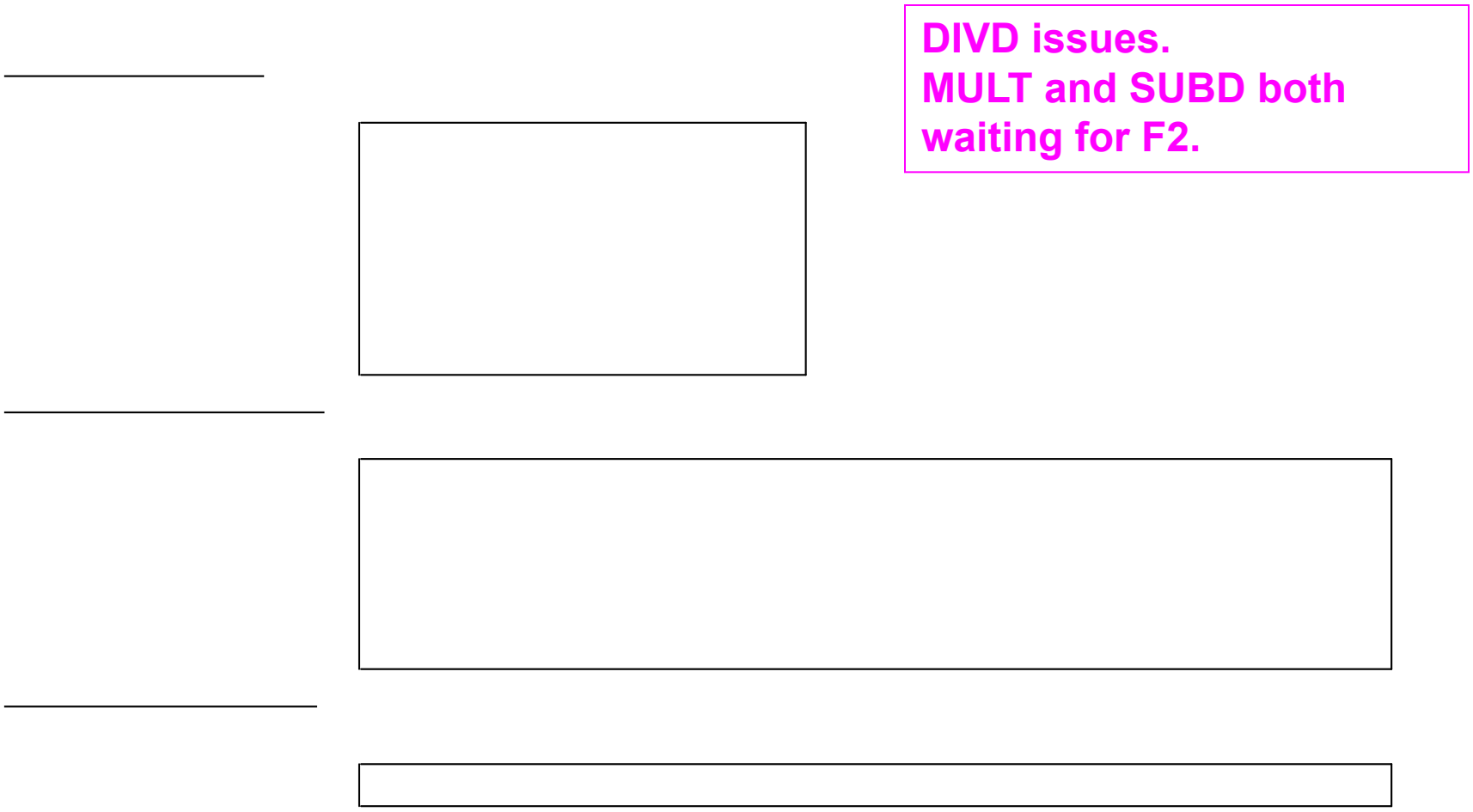
# Scoreboard Example Cycle 6



# Scoreboard Example Cycle 7

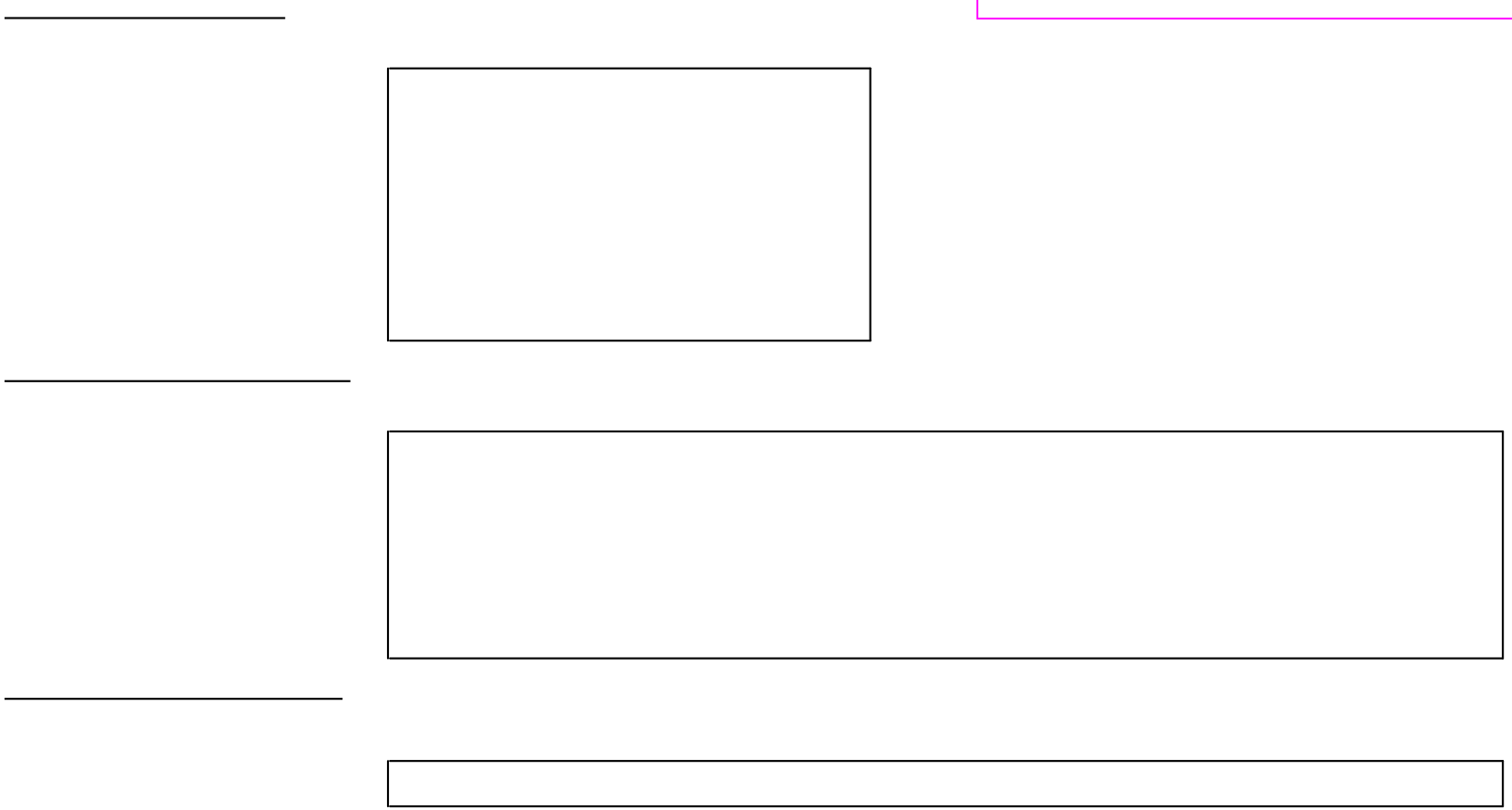


# Scoreboard Example Cycle 8a

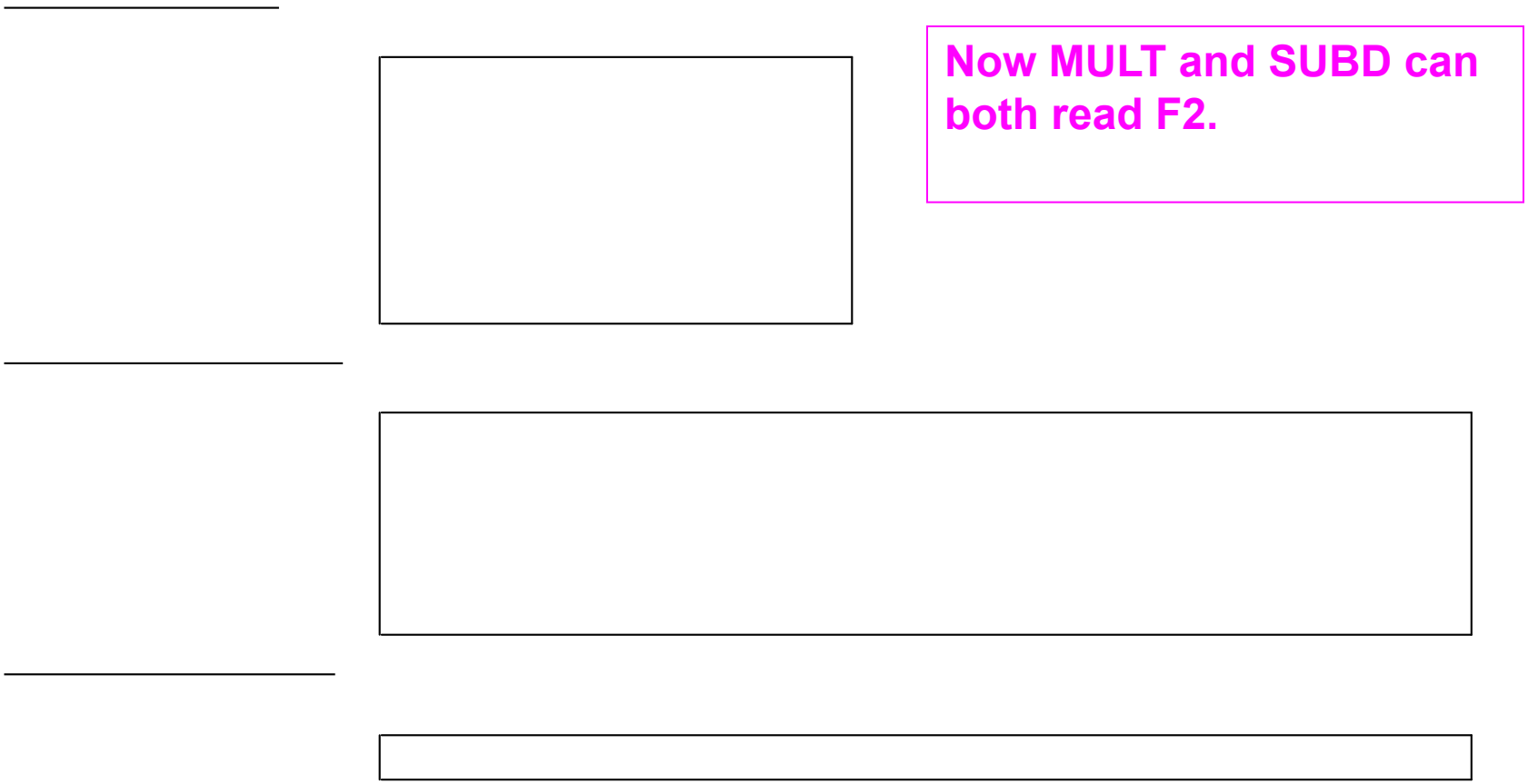


# Scoreboard Example Cycle 8b

**LD #2 writes F2.**



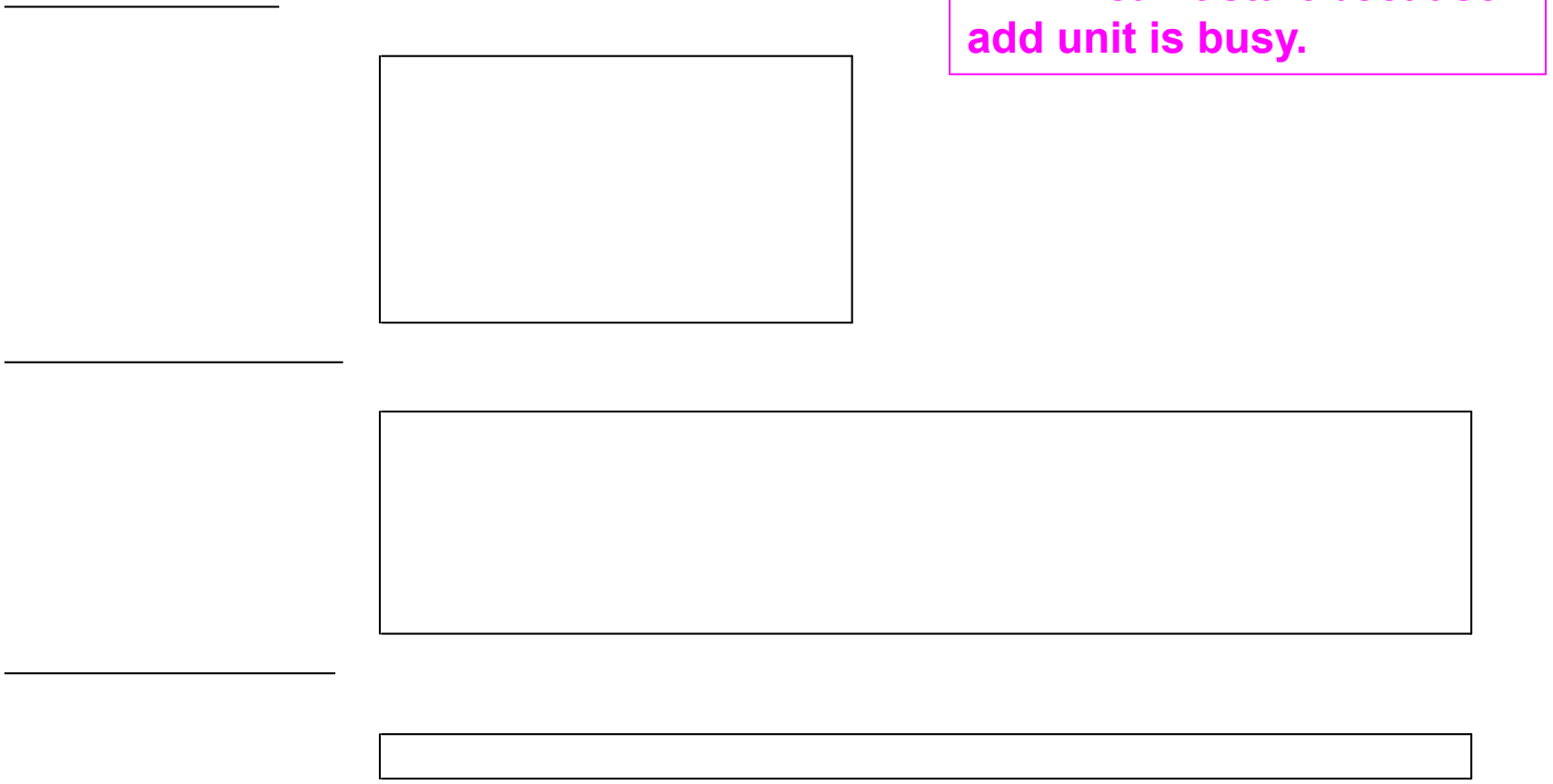
# Scoreboard Example Cycle 9



**Now MULT and SUBD can both read F2.**

# Scoreboard Example Cycle 11

**ADDD can't start because add unit is busy.**

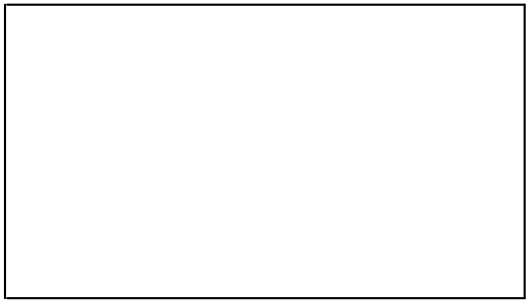


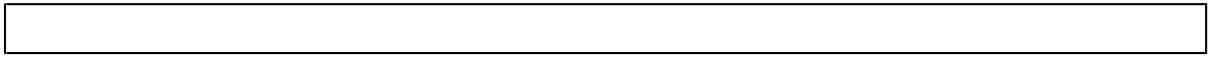
# Scoreboard Example Cycle 12

**SUBD finishes.**
**DIVD waiting for F0.**

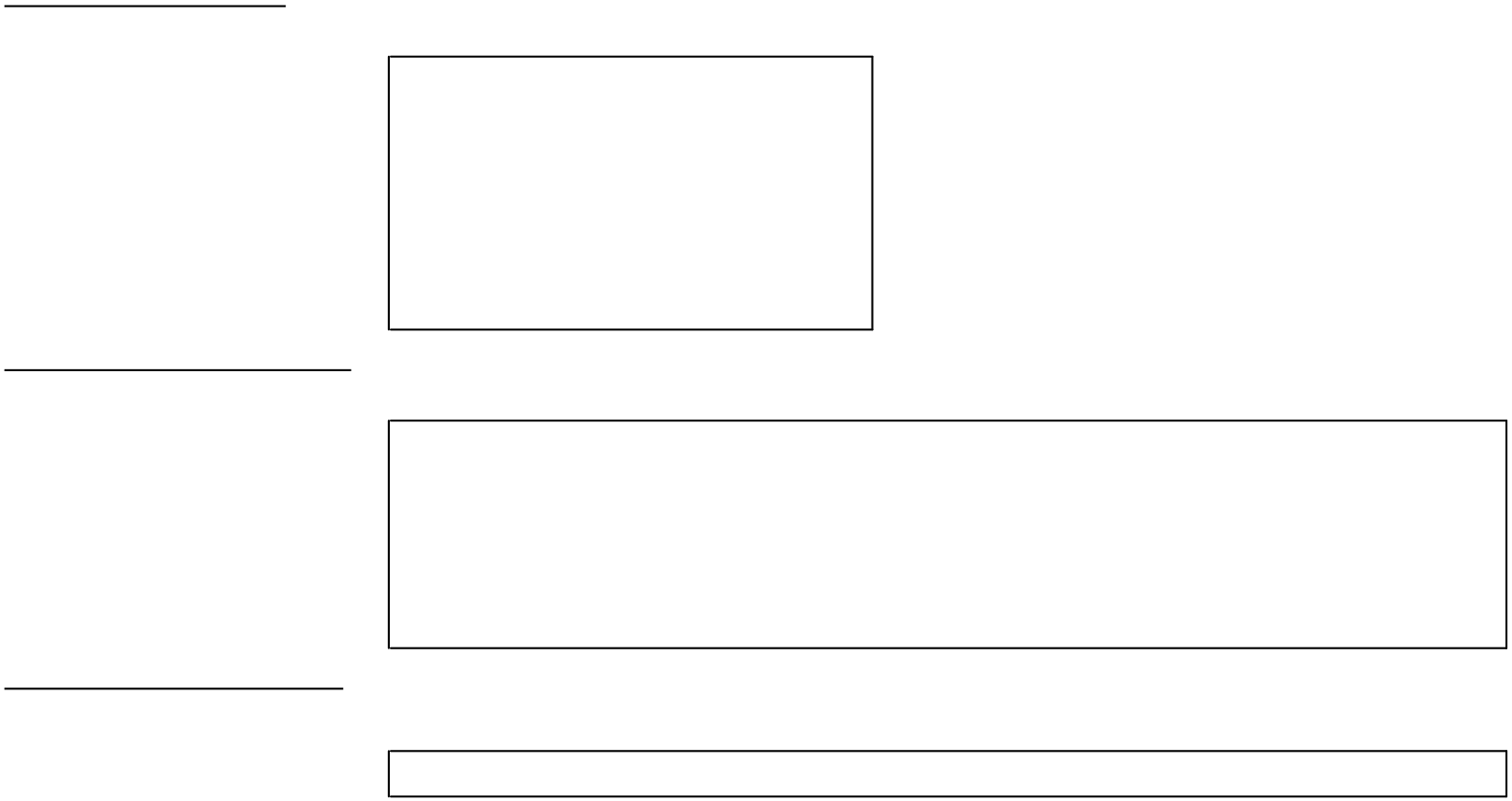
# Scoreboard Example Cycle 13

**ADDD issues.**

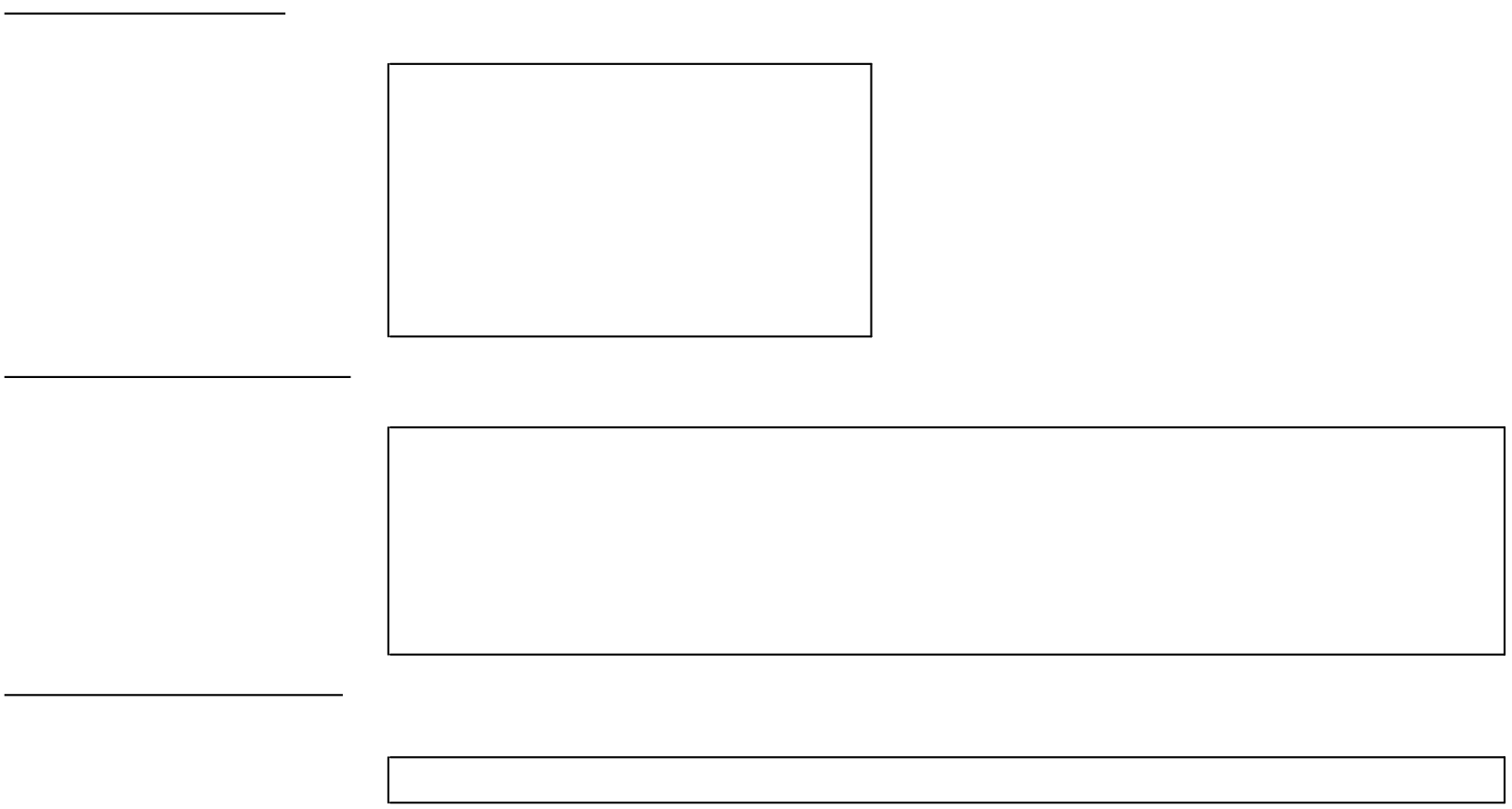




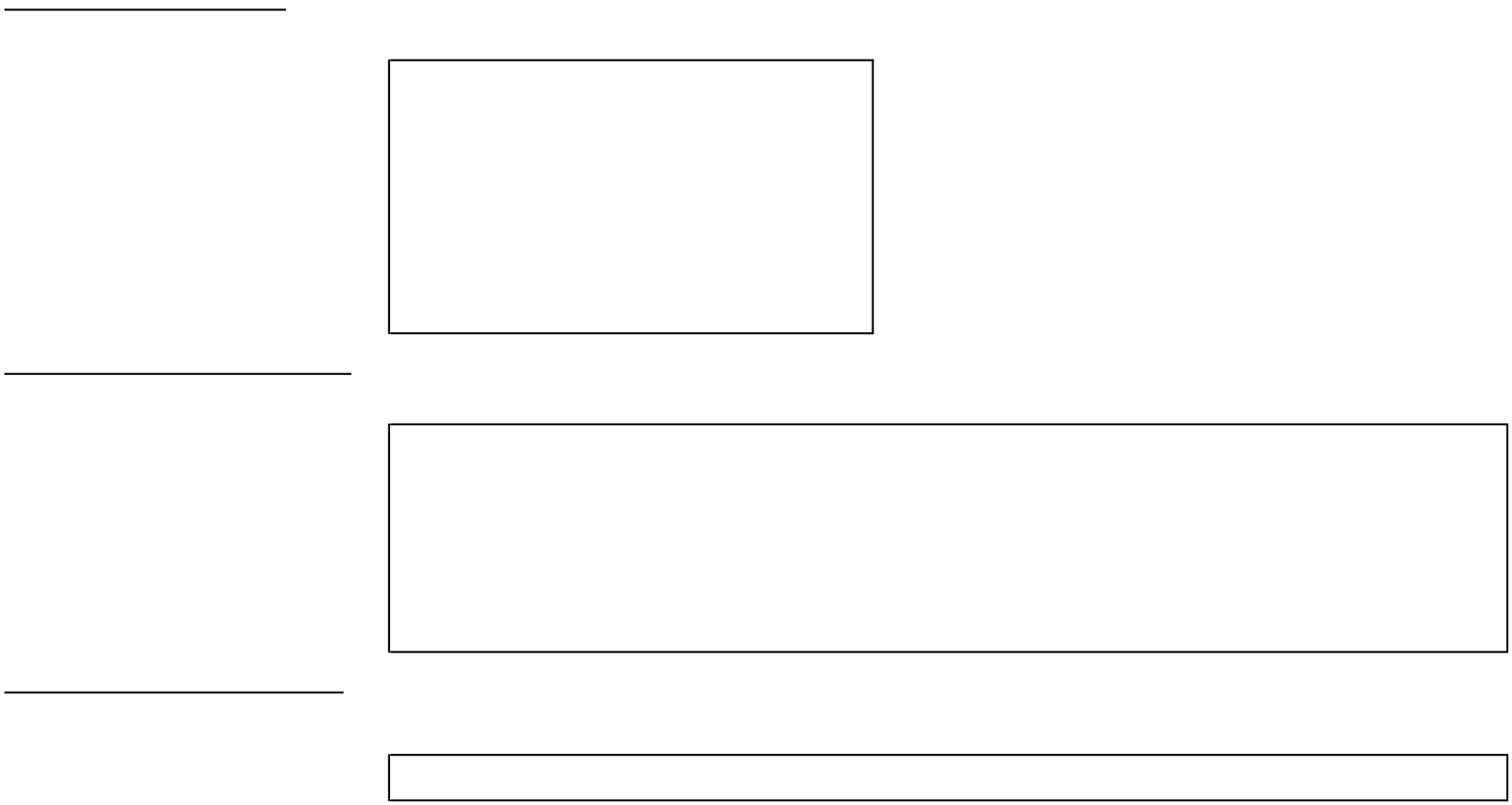
# Scoreboard Example Cycle 14



# Scoreboard Example Cycle 15

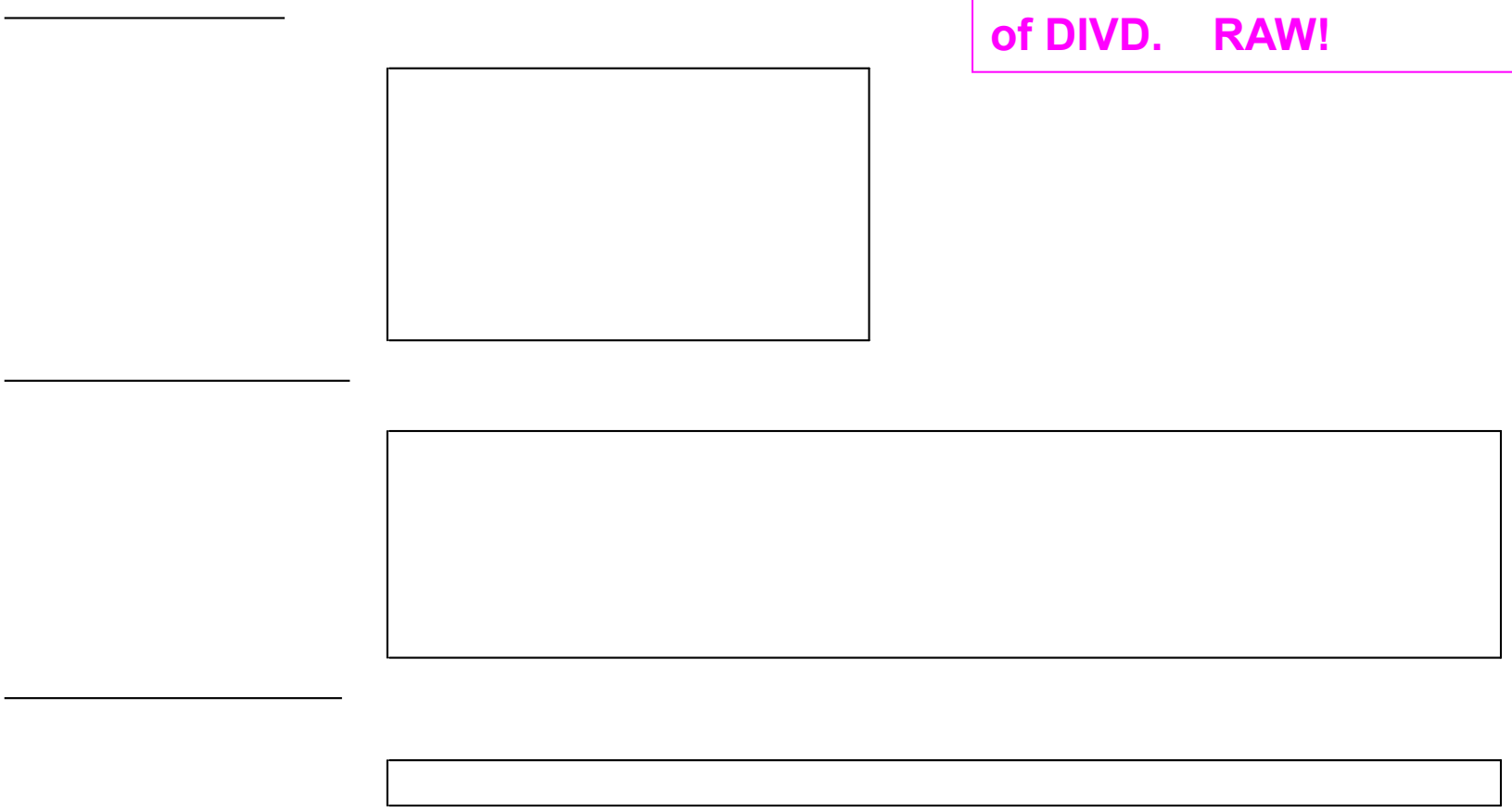


# Scoreboard Example Cycle 16



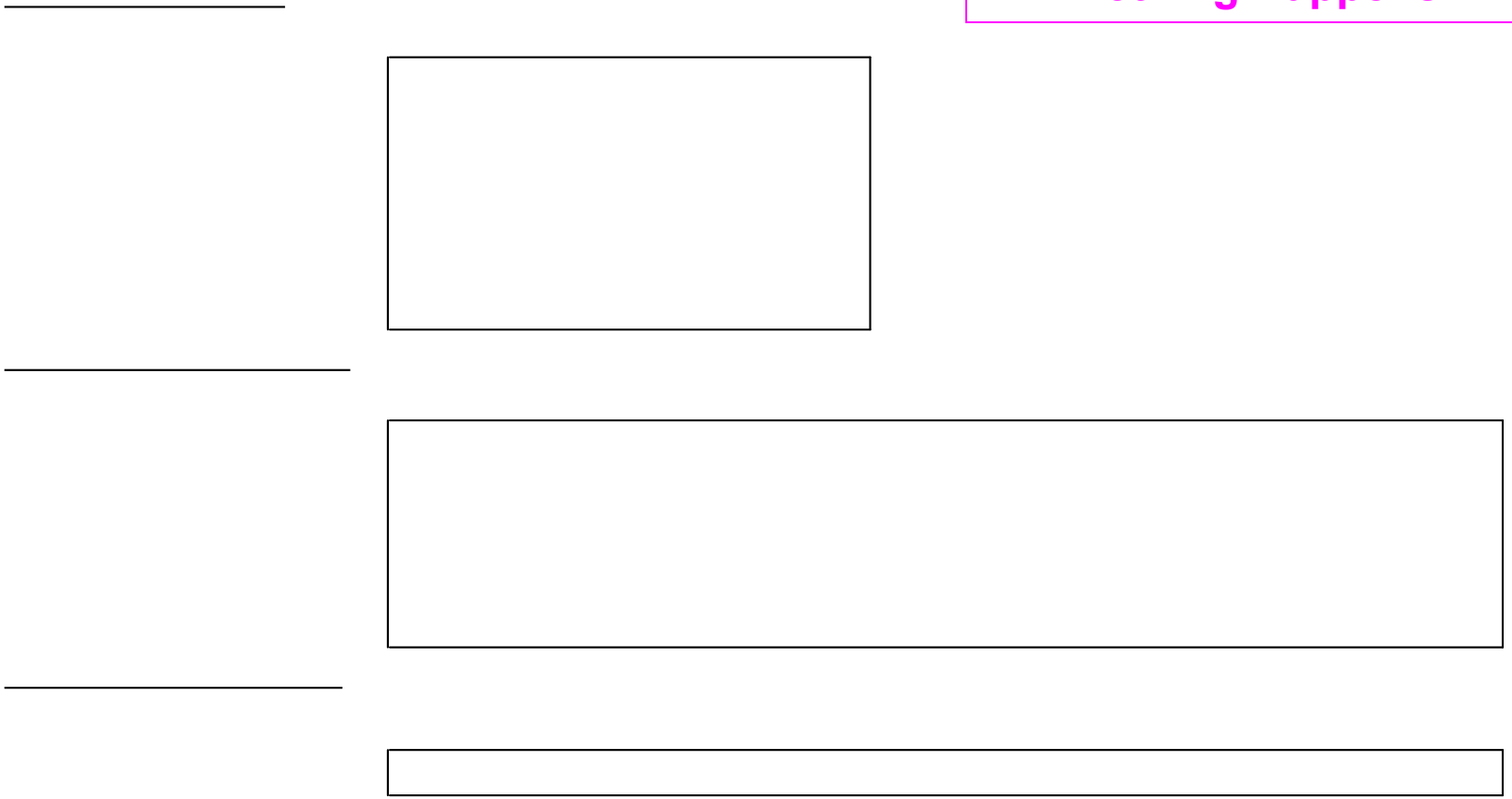
# Scoreboard Example Cycle 17

**ADDD can't write because of DIVD. RAW!**



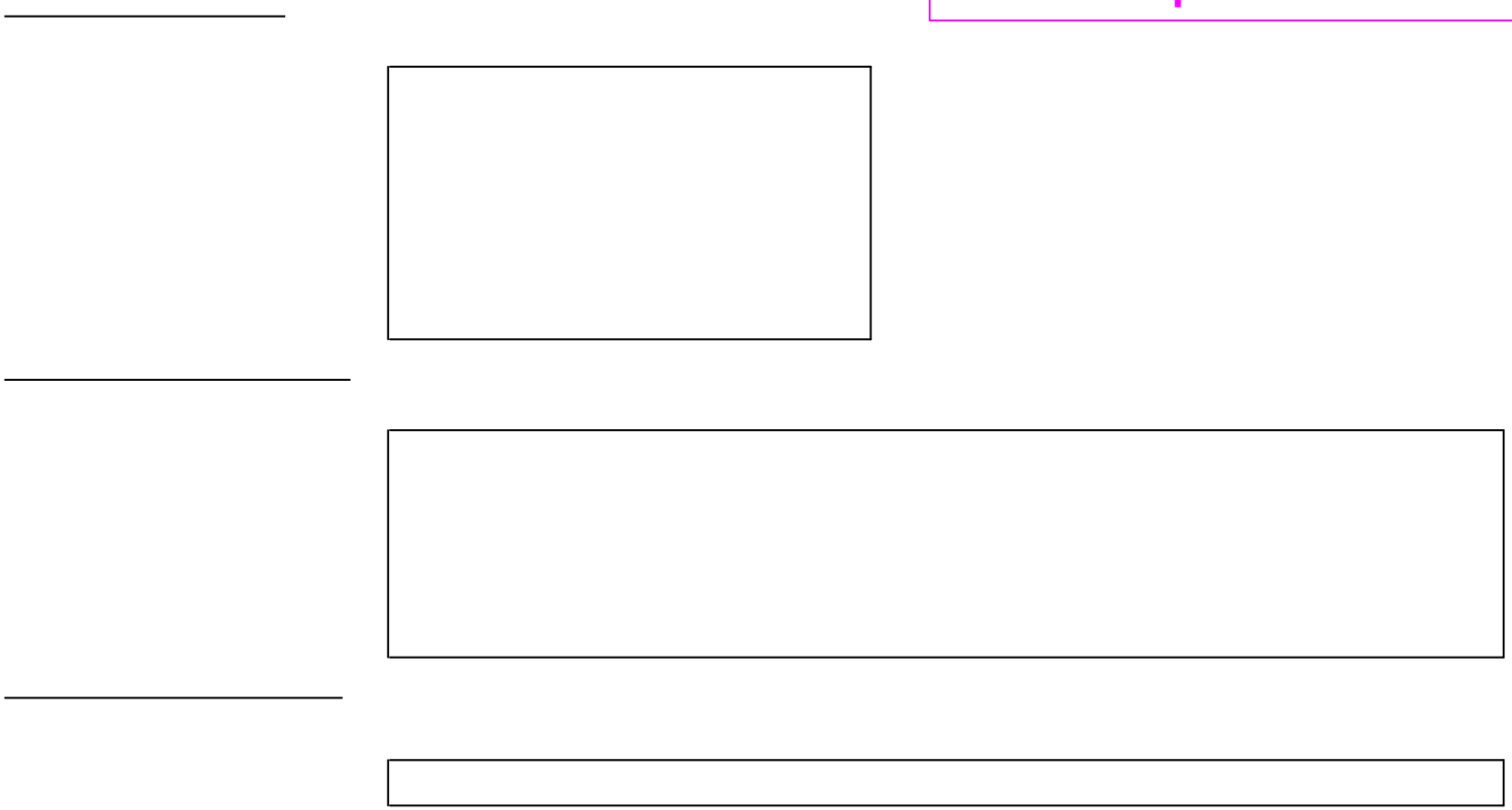
# Scoreboard Example Cycle 18

**Nothing Happens!!**



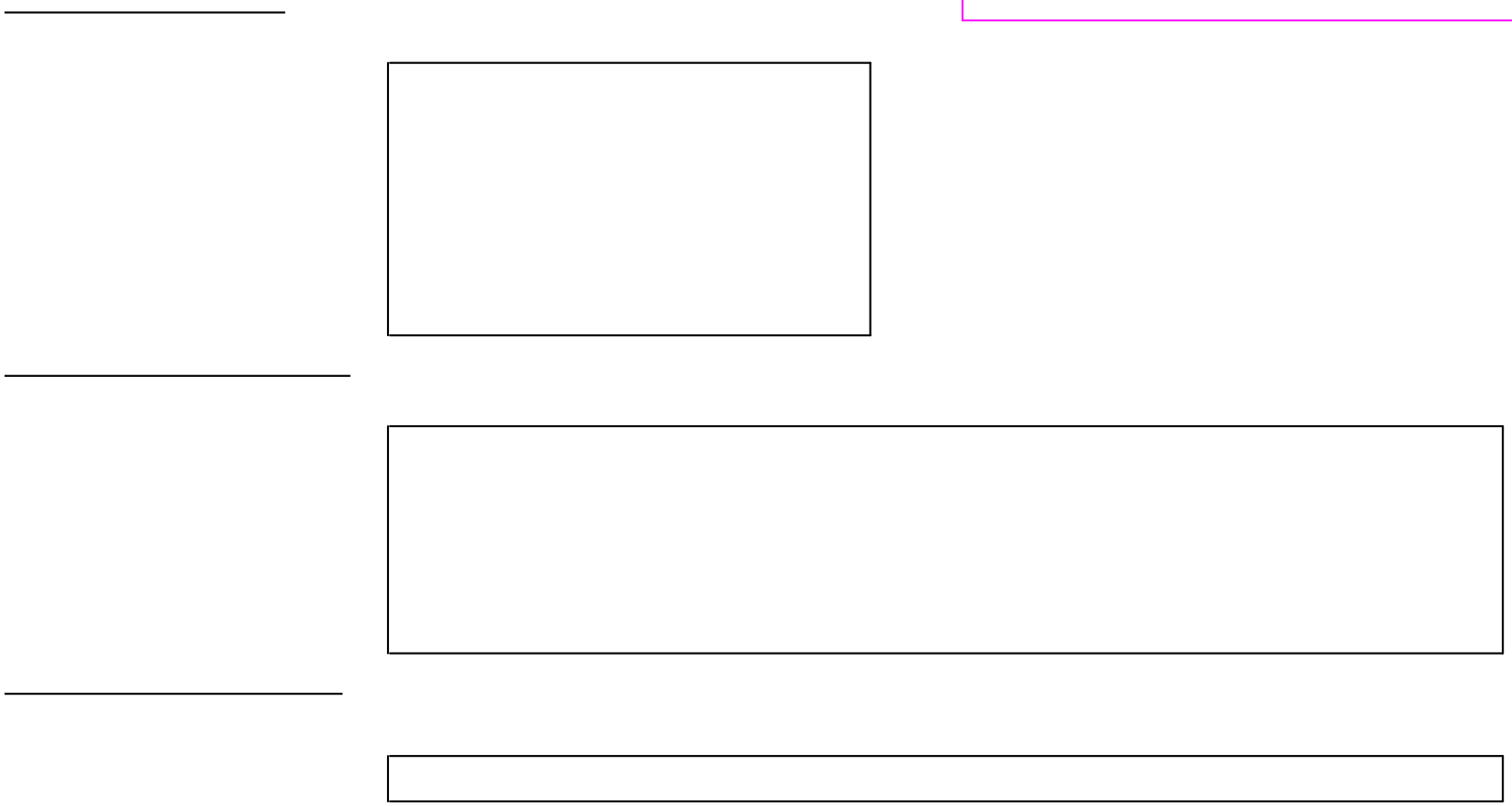
# Scoreboard Example Cycle 19

**MULT completes execution.**



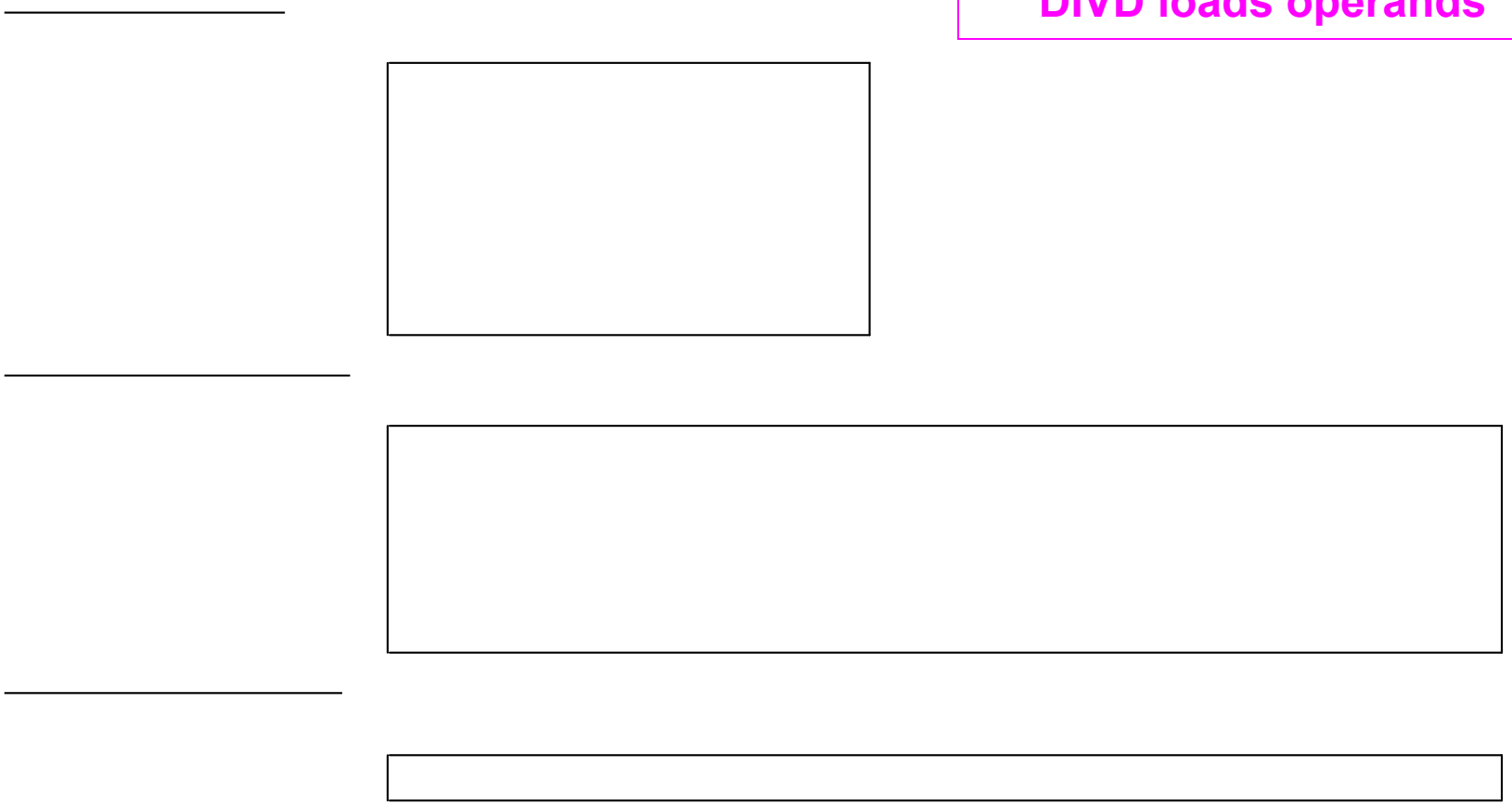
# Scoreboard Example Cycle 20

**MULT writes.**



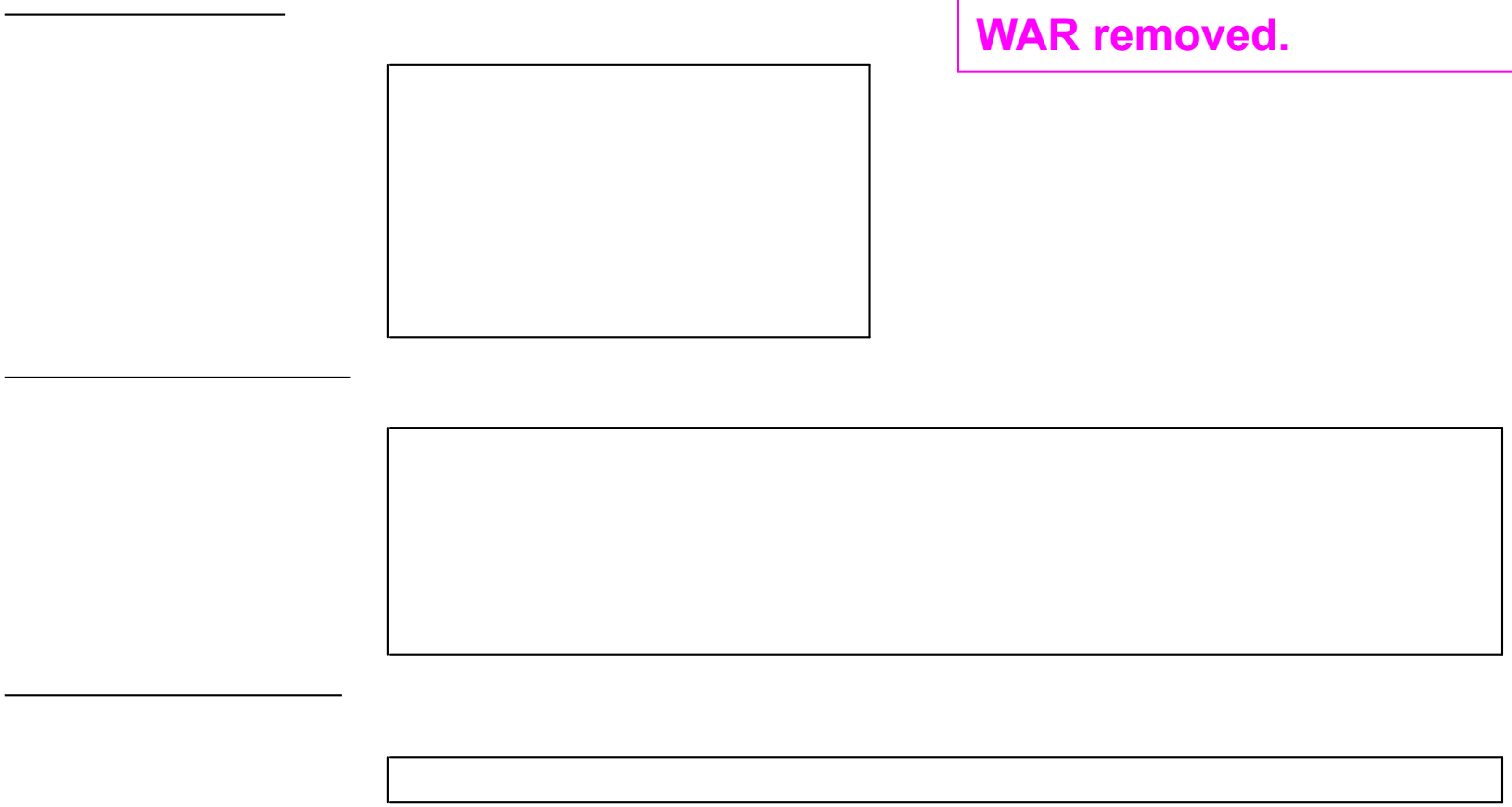
# Scoreboard Example Cycle 21

DIVD loads operands



# Scoreboard Example Cycle 22

**Now ADDD can write since WAR removed.**

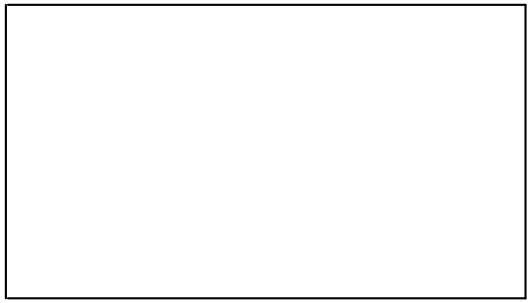


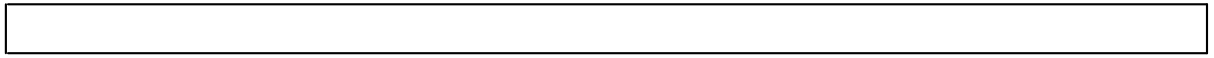
# Scoreboard Example Cycle 61

**DIVD completes execution**

# Scoreboard Example Cycle 62

DONE!!





# Limitations of Scoreboard

- The amount of parallelism available among the instructions (chosen from the same basic block)
- The number of score entries (The size of the scoreboard determines the size of the window)
- The number and types of functional units (Structural hazards increase when dynamic scheduling is used)
- The presence of antidependence and output dependences lead to WAR and WAW stalls.