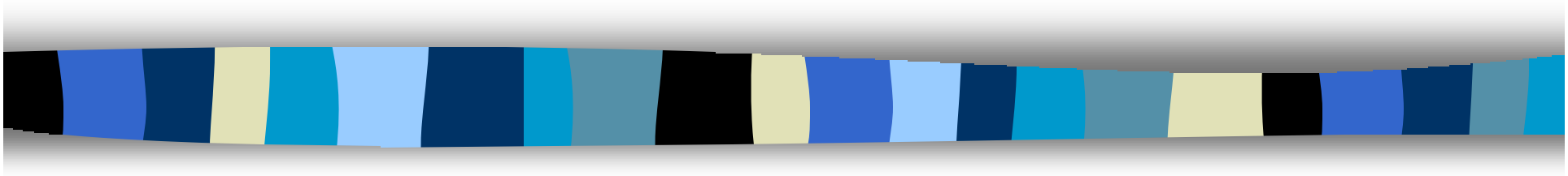
Optimality of Tomasulo's Algorithm



Luna, Dong

Gang, Zhao

Feb 28th, 2002

Our Questions about Tomasulo

- Questions about Tomasulo's Algorithm
  - Is it optimal (**can always produce the wisest instruction execution stream**)?
  - Is there any room for improvement if it is not optimal?
  - Is it a wise trade-off between time and complexity?
- Related work
  - A lot of study on its correctness
  - Some tests on its performance under different implementation details
  - Few theoretical analysis of its performance

Our Approach

- Our approach
  - Build up mathematical models for formal proof or find counterexamples for disproof
- Features of our approach
  - Making comparison with **a reference system – Data Driven System –** which can produce all possible instruction execution orders without violating data dependences.
  - Introducing **time description variables** into the model to record the finish time of each instruction
  - Examining the problem under **a group of ideal assumptions** first, and then drop them one by one.

Our Result

- Unlimited hardware resources, no delay, only calculations – optimal ☺
- With transmission delay – optimal ☺
- With limited instruction window – optimal ☺
- With Load/Store – not optimal ☹ **IMPROVEMENT**
- With limited hardware resources – not optimal ☹ **IMPROVEMENT**
- Issue one instruction per cycle – not optimal ☹

Assumptions for all of the Models

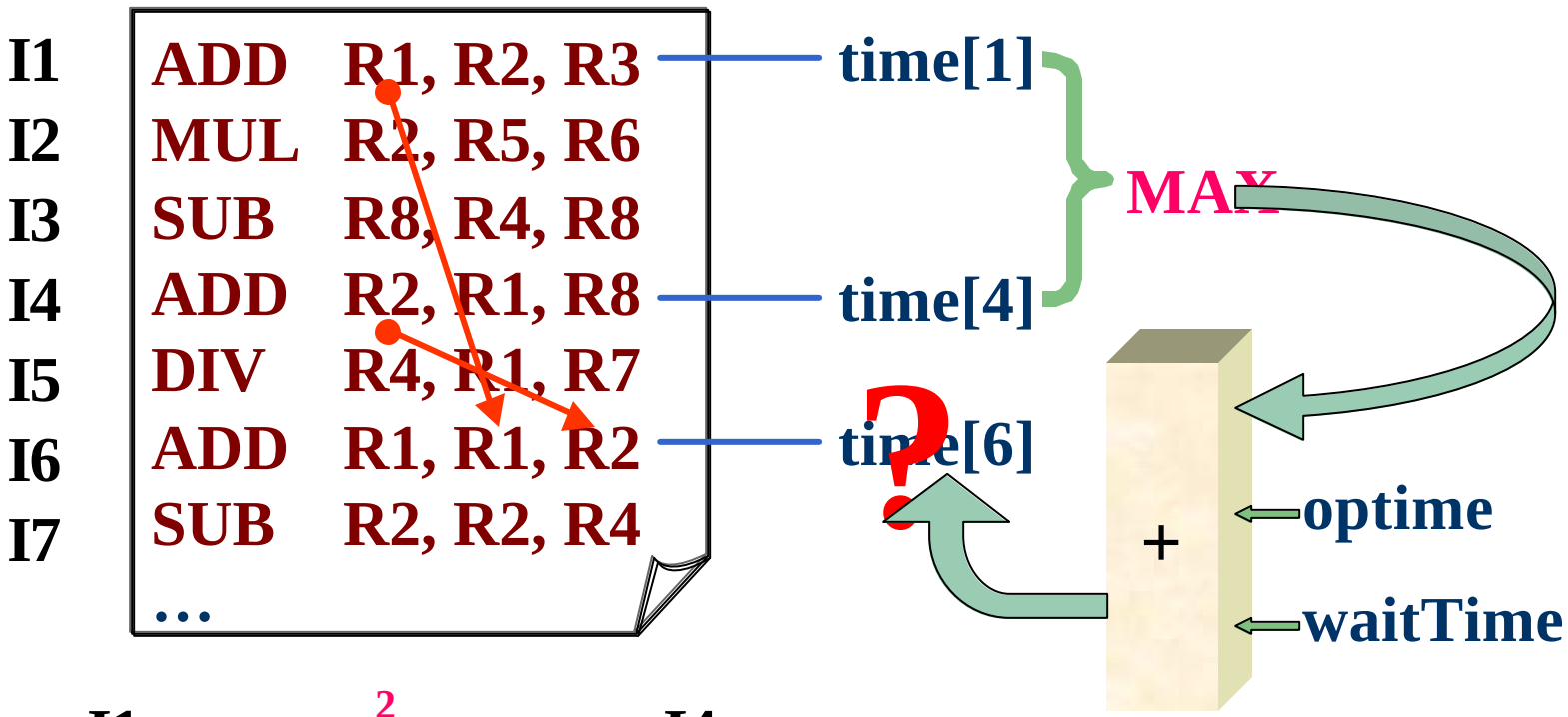
- Register renaming in both systems
  - no WAR, WAW
- No jump, no branch
- Instruction misses fully hidden
- Fixed-point function units are regarded just as another kind of units like floating-point adder and floating-point multiplier

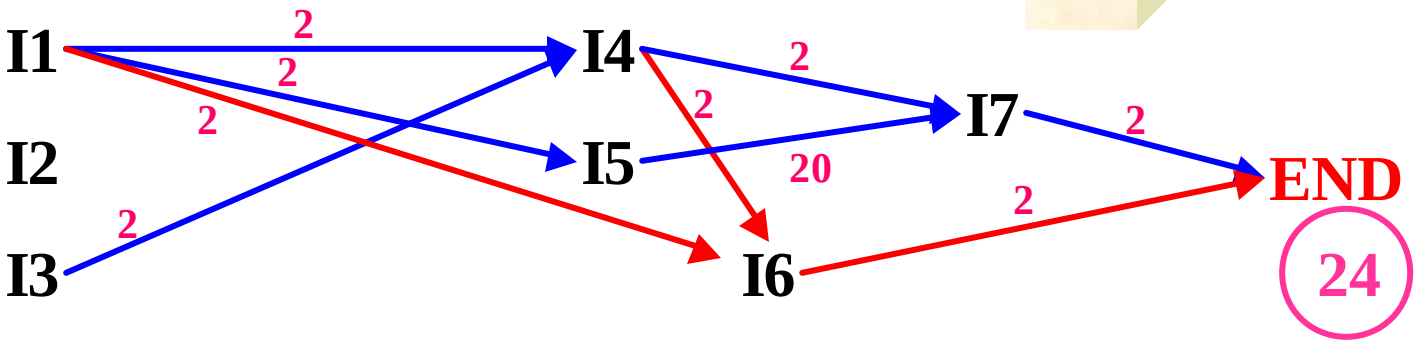
Our Result

- **Unlimited hardware resources, no delay – optimal ☺**
- With transmission delay – optimal ☺
- With limited instruction window – optimal ☺
- With Load/Store – not optimal ☹
- With limited hardware resources – not optimal ☹
- Issue one instruction per cycle – not optimal ☹

Model I – Utopia Model

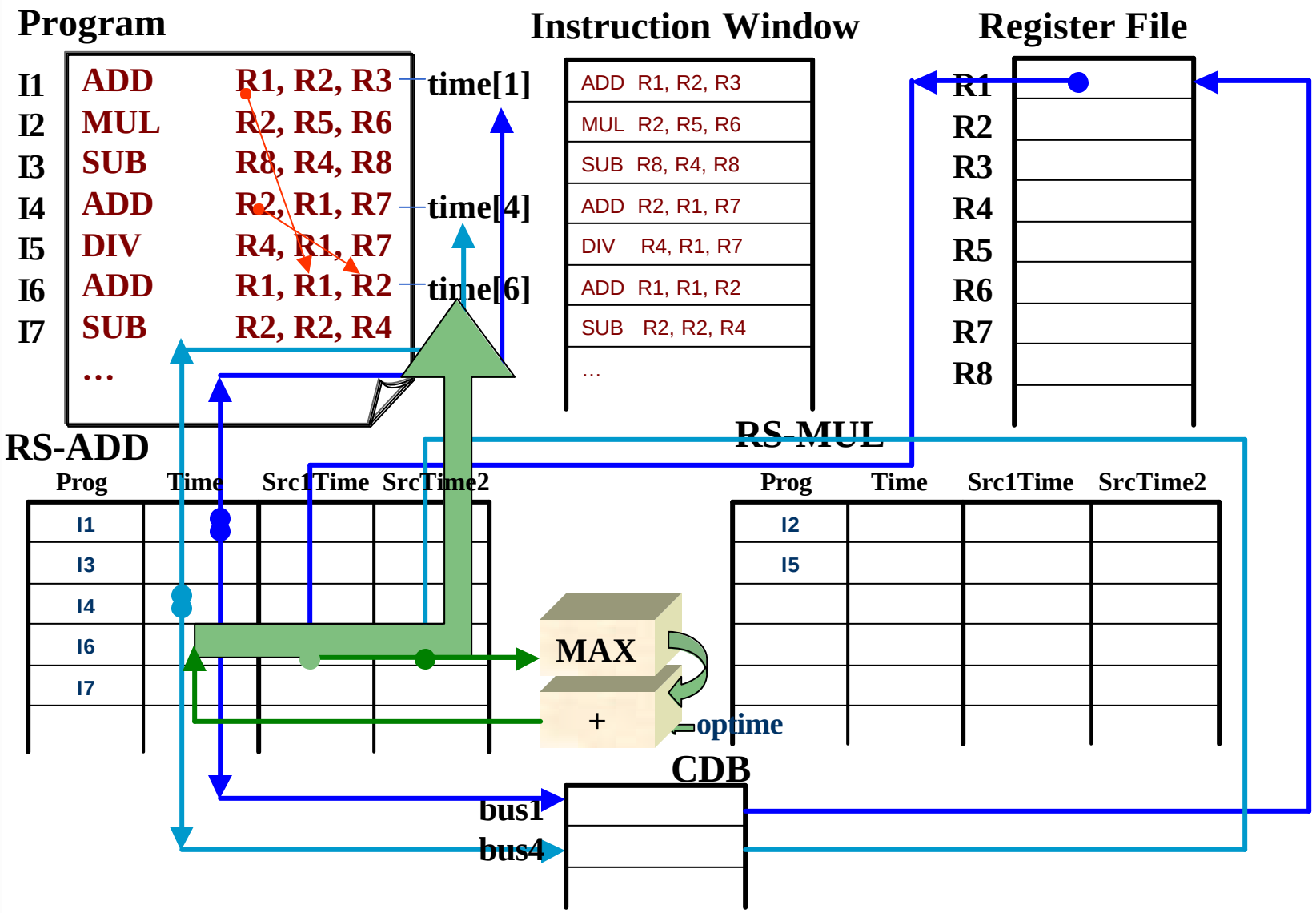
DATA DRIVEN SYSTEM





Model I – Utopia Model

TOMASULO SYSTEM



Model I – Utopia Model

PROOF

$$Time_d[n] = \max(time_d[sr1], time_d[sr2]) + opTime + waitTime$$

$$>= \max(time_d[sr1], time_d[sr2]) + opTime$$

IV IV IV II

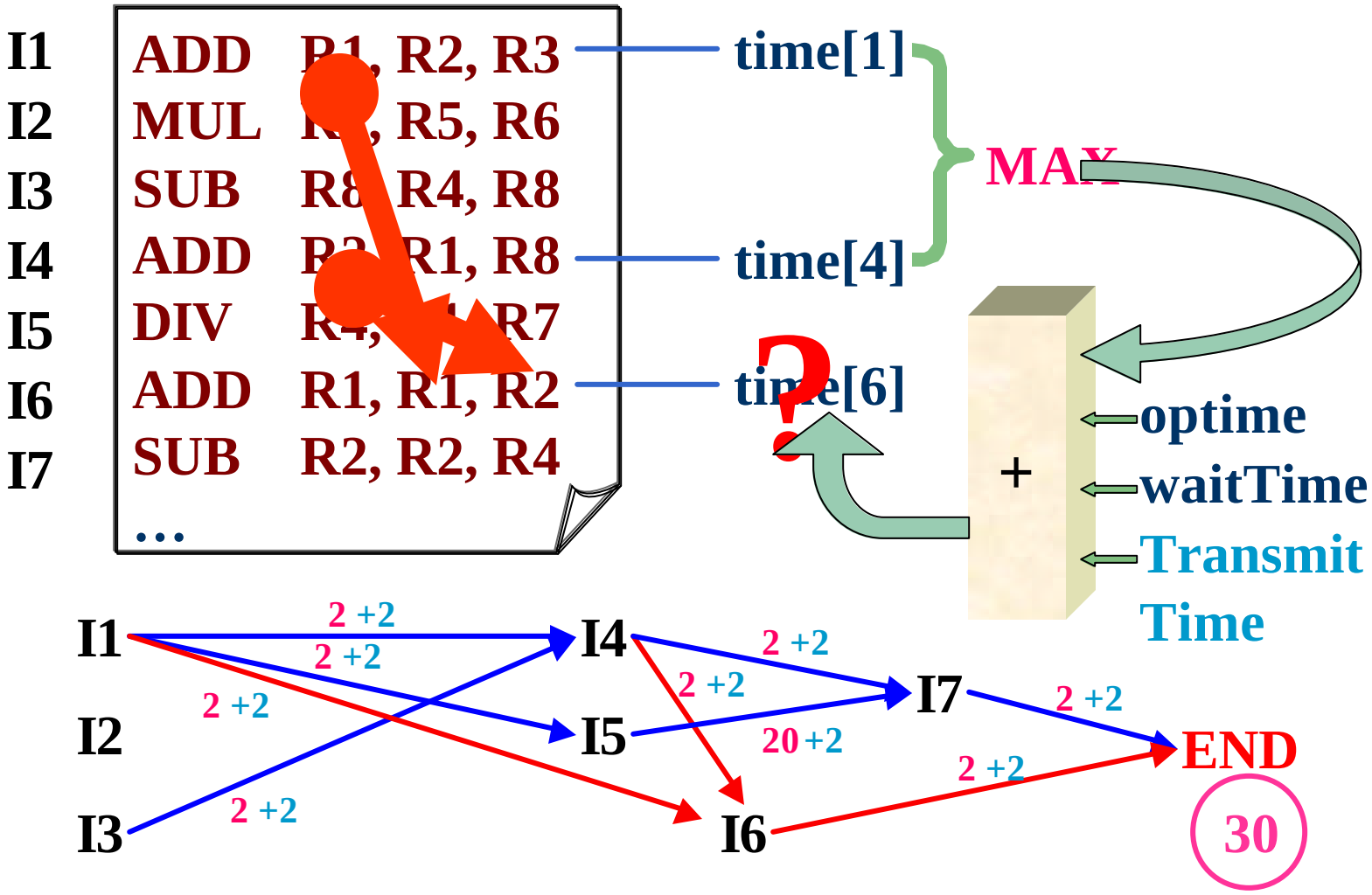
$$Time_t[n] = \max(time_t[sr1], time_t[sr2]) + opTime$$

Our Result

- Unlimited hardware resources, no delay – optimal ☺
- **With transmission delay – optimal ☺**
- With limited instruction window – optimal ☺
- With Load/Store – not optimal ☹
- With limited hardware resources – not optimal ☹
- Issue one instruction per cycle – not optimal ☹

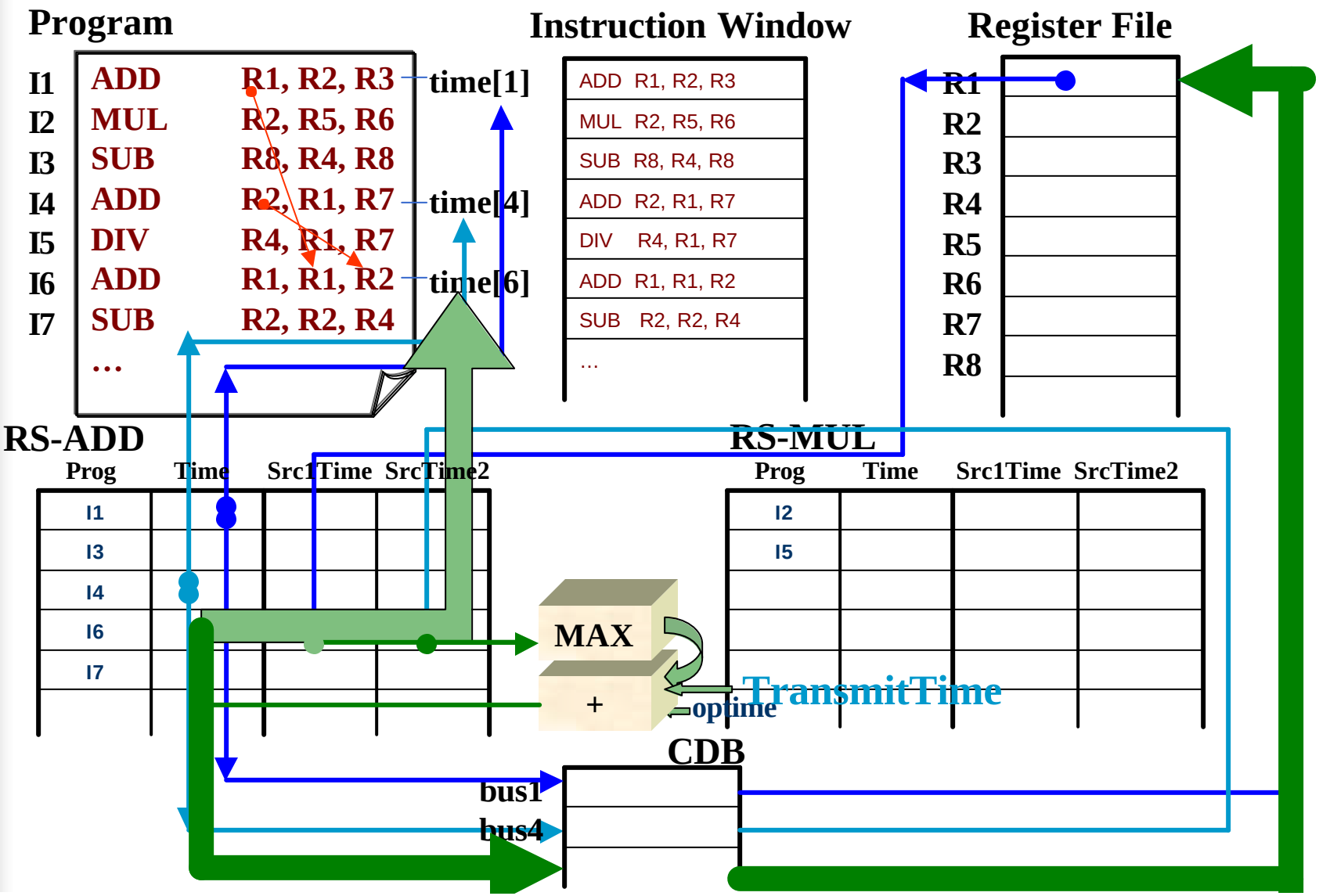
Model II – Transmission Delay

DATA DRIVEN SYSTEM



Model II – Transmission Delay

TOMASULO SYSTEM

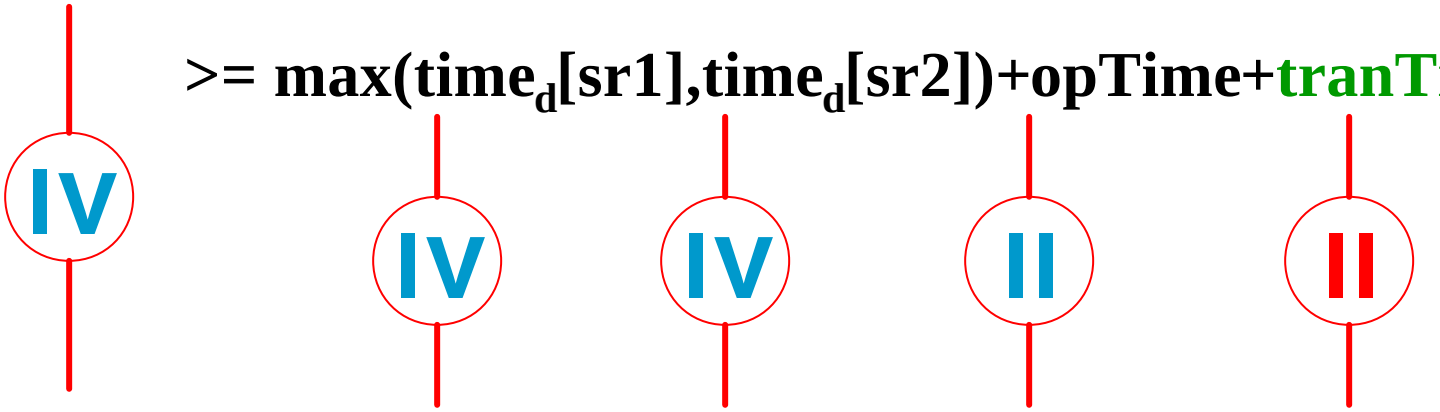


Model II – Transmission Delay

PROOF

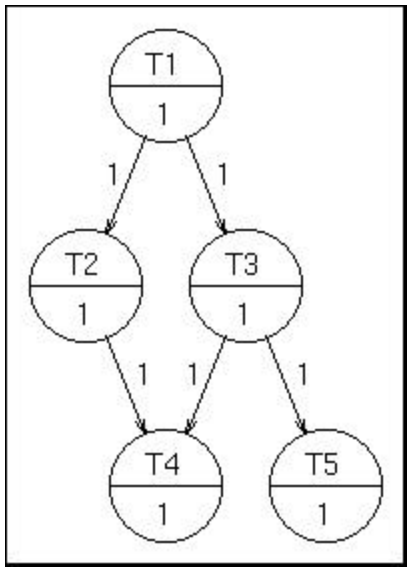
$Time_d[n] = max(time_d[sr1], time_d[sr2]) + opTime + waitTime + tranTime$

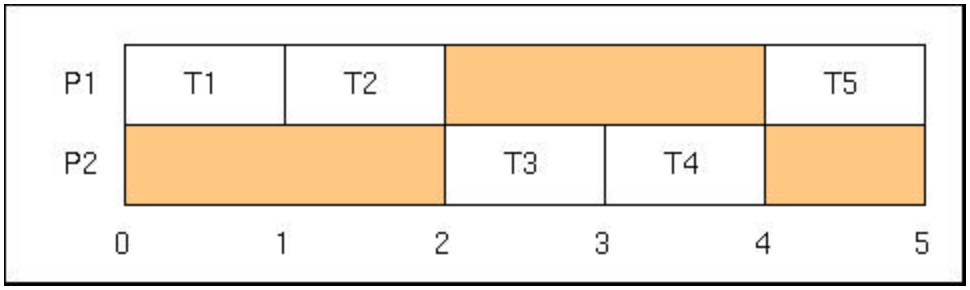
$>= max(time_d[sr1], time_d[sr2]) + opTime + tranTime$

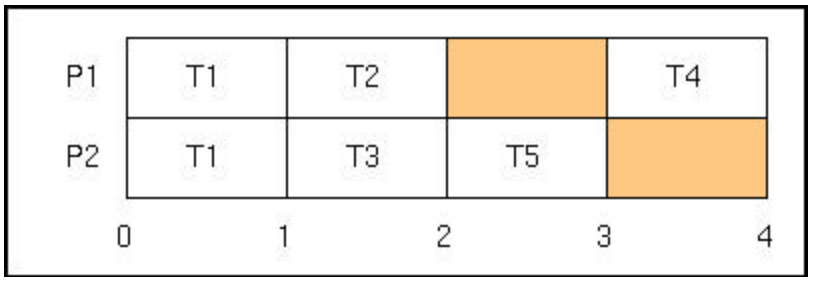


$Time_t[n] = max(time_t[sr1], time_t[sr2]) + opTime + tranTime$

Model II – Transmission Delay



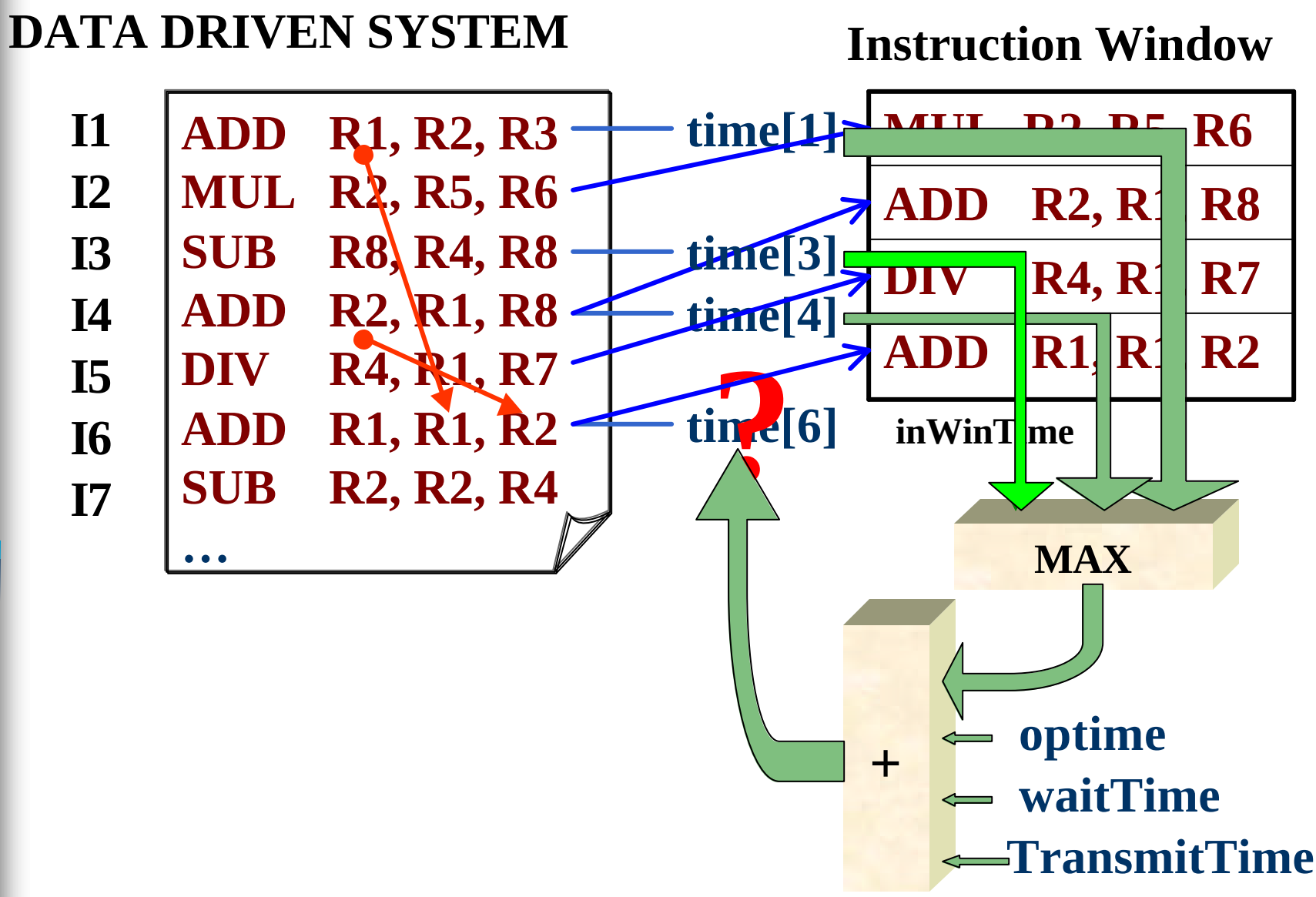




Our Result

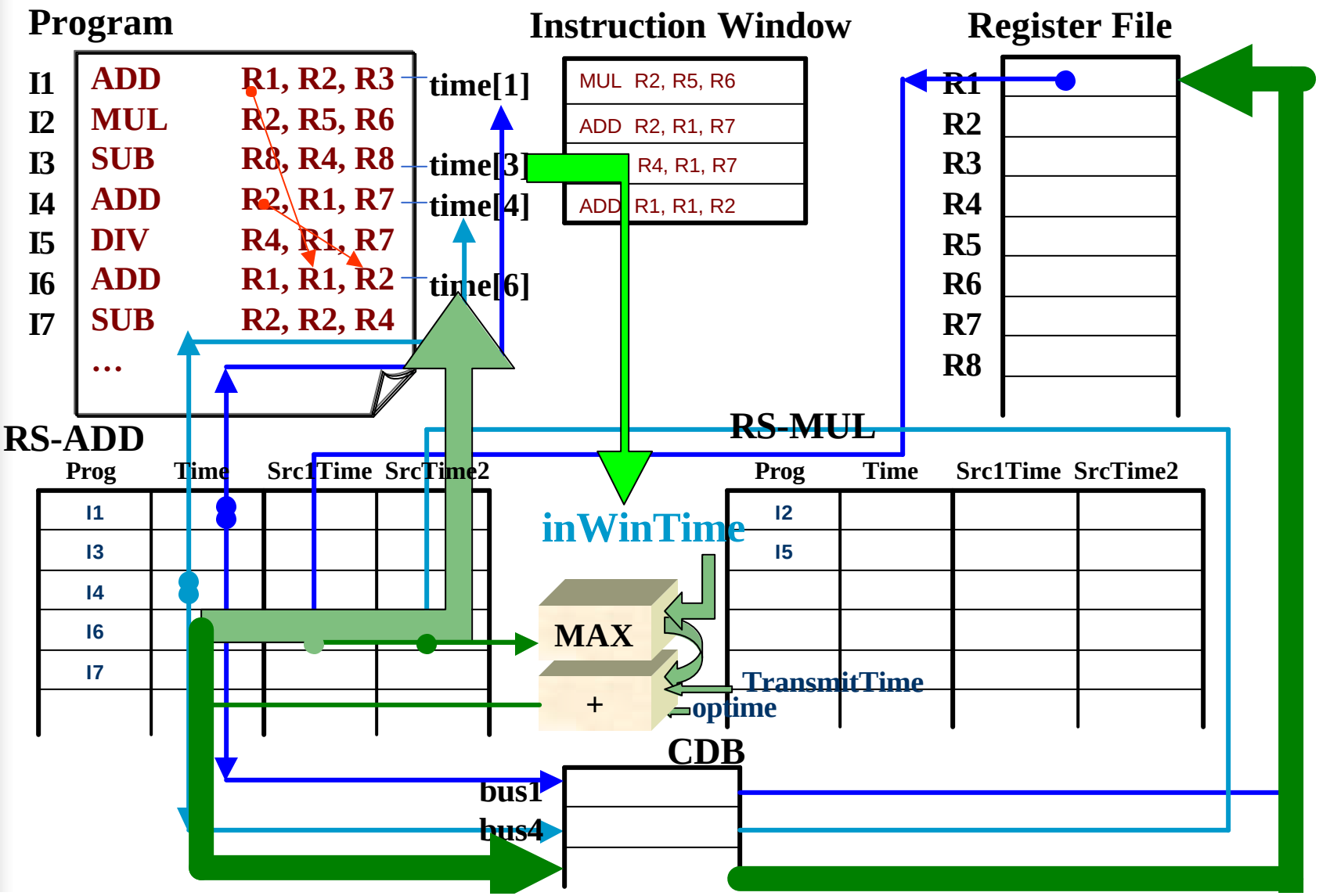
- Unlimited hardware resources, no delay – optimal ☺
- With transmission delay – optimal ☺
- **With limited instruction window – optimal ☺**
- With Load/Store – not optimal ☹
- With limited hardware resources – not optimal ☹
- Issue one instruction per cycle – not optimal ☹

Model III – Limited Instru Win



Model III – Limited Instru Win

TOMASULO SYSTEM



Model III – Limited Instru Win

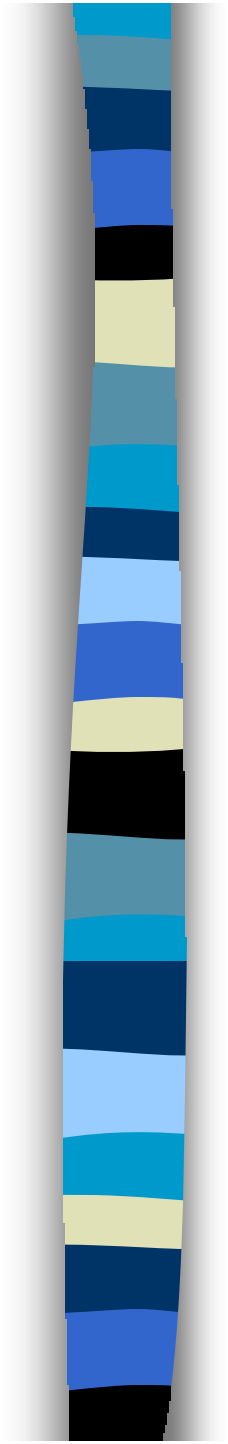
PROOF

$Time_d$ = max($time_d$[sr1],$time_d$[sr2],inWinTime)+opTime+waitTime+tranTime

>= max($time_d$[sr1],$time_d$[sr2],inWinTime)+opTime+tranTime

IV IV IV IV II II

$Time_t$ = max($time_t$[sr1],$time_t$[sr2],inWinTime)+opTime+tranTime



Our Result

- Unlimited hardware resources, no delay – optimal ☺
- With transmission delay – optimal ☺
- With limited instruction window – optimal ☺
- **With Load/Store – not optimal ☹**
- With limited hardware resources – not optimal ☹
- Issue one instruction per cycle – not optimal ☹

Load and store

- **Data dependence in load and store**.
  - The principles (We have discussed in our class)
    - A stores is dependent on the previous store
    - All Load are dependent on the previous store
    - A store is dependent on all loads between it and the previous store.
- **Load and store buffer used in Tomasulo.**
  - L/S dependence may block instruction window as Tomasulo's design.
    - For load instructions, no address conflict in store buffer
    - For store instructions, no address conflict in load and store buffer
  - In Tomasulo, the L/S buffer is a little bit conservative. More aggressive strategy may boost throughput in some case.

Load and store (cont.)

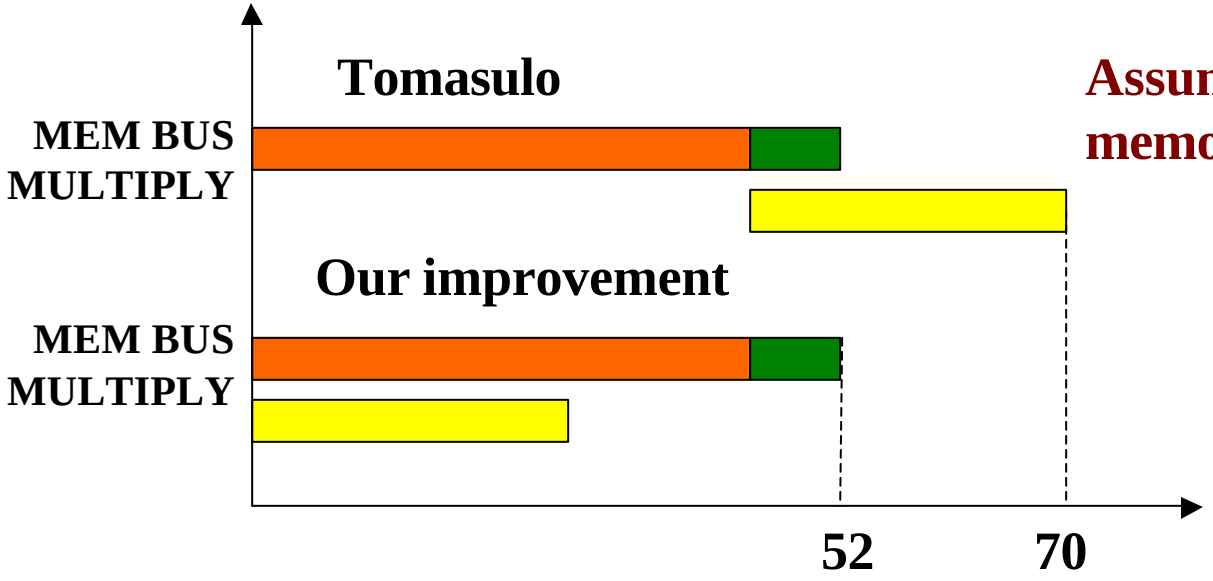
(1) Store R1, (A1); 50 cycles(cache miss)

…

(n) Load R1, (A1); 2 cycles

(n+1) Mult R3, R2, R4; 20 cycles

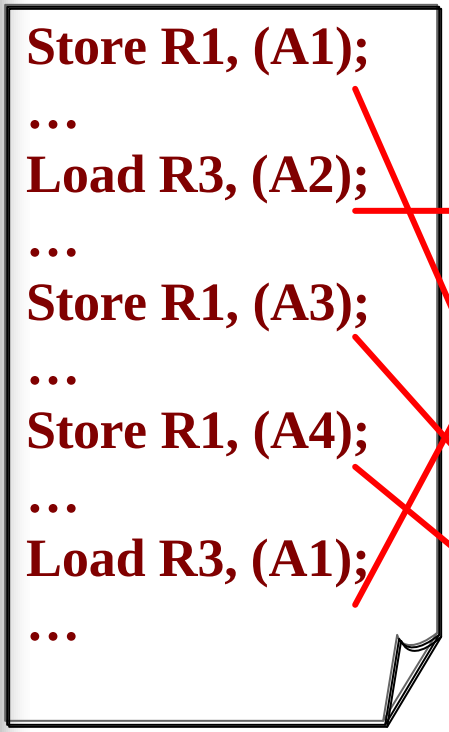
RAW dependence here,



Assume only one memory bus

Load and store (cont.)

- Modifying L/S buffer to RS-like structure.



Load Buffer

ID	Busy	Vj	Qj	Index	Ready
0	Yes	R [A2]		100	Yes
1	Yes		ADD2	103	No
2	No				

Store Buffer

ID	Busy	Vj	Vk	Qj	Qk	Index	Ready
0	Yes	R [A1]	R [R1]			99	Yes
1	Yes			ADD1	MUL1	101	No
2	Yes	R [A4]	R[R3]			102	No
3	No						

Our Result

- Unlimited hardware resources, no delay – optimal ☺
- With transmission delay – optimal ☺
- With limited instruction window – optimal ☺
- With Load/Store – not optimal ☹
- **With limited hardware resources – not optimal ☹**
- Issue one instruction per cycle – not optimal ☹

The last-ditch of the optimality of Tomasulo

- Resource competition forces you to make choice.
  - Finite RS,FU, CDB
  - Scheduling under resource competition is an **NPC** problem. (It's impossible to get the optimal result in polynomial time.)
  - Tomasulo is actually a greedy algorithm.
  - Tomasulo uses simple FIFO strategy to to solve ties in competition.

Counterexamples (Sorry, Tomasulo) : Resource Competition

(1) ADD A1, A2, A3 — 2 cycles

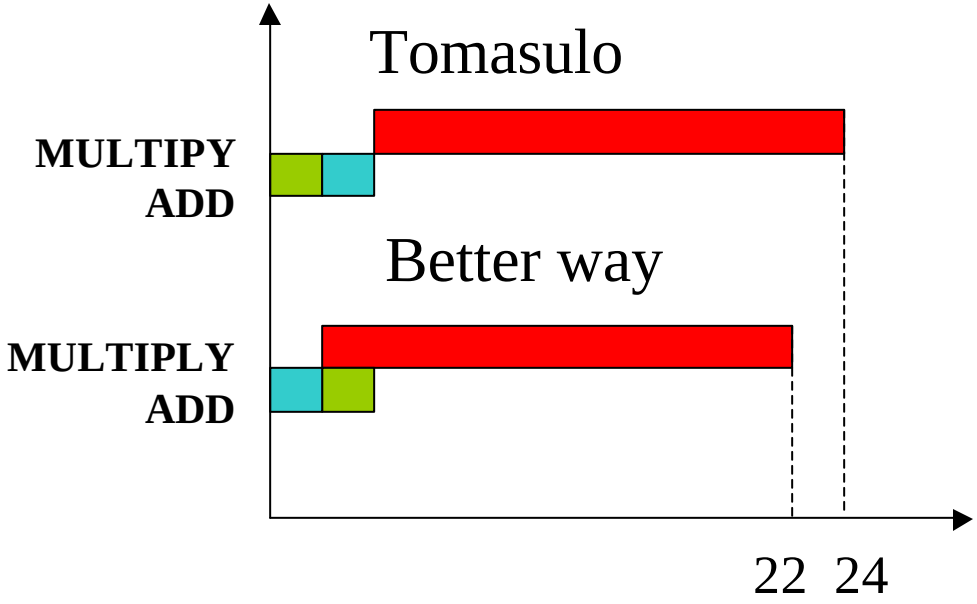
(2) ADD A4, A5, A6 — 2 cycles

(3) MUL A7, A8, A4 — 20 cycles

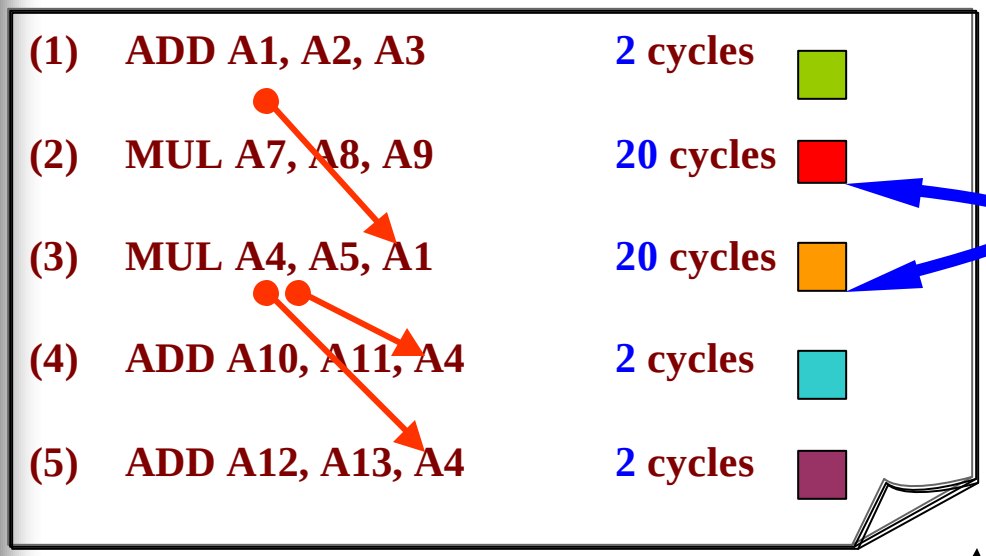
Compete if these two instructions get ready at same time.

Assume we have only one multiply FU and add FU.

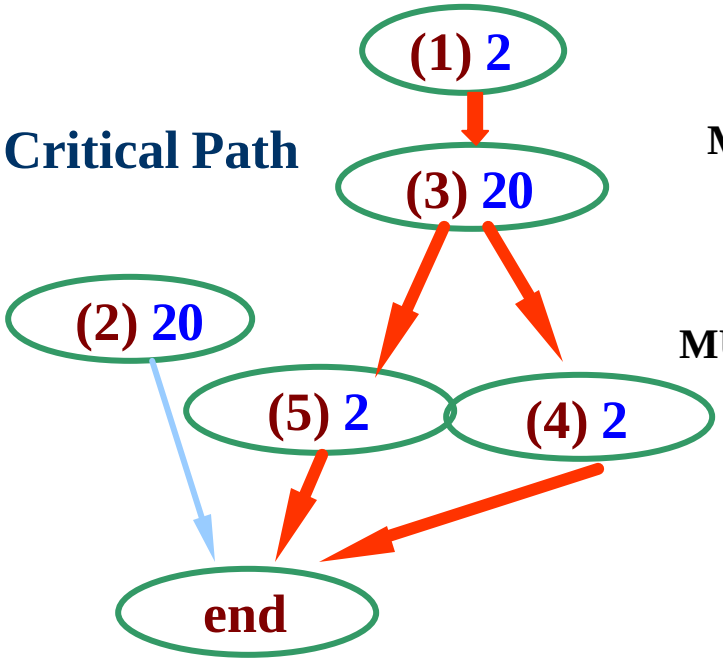
The similar situation for finite RS and CDB

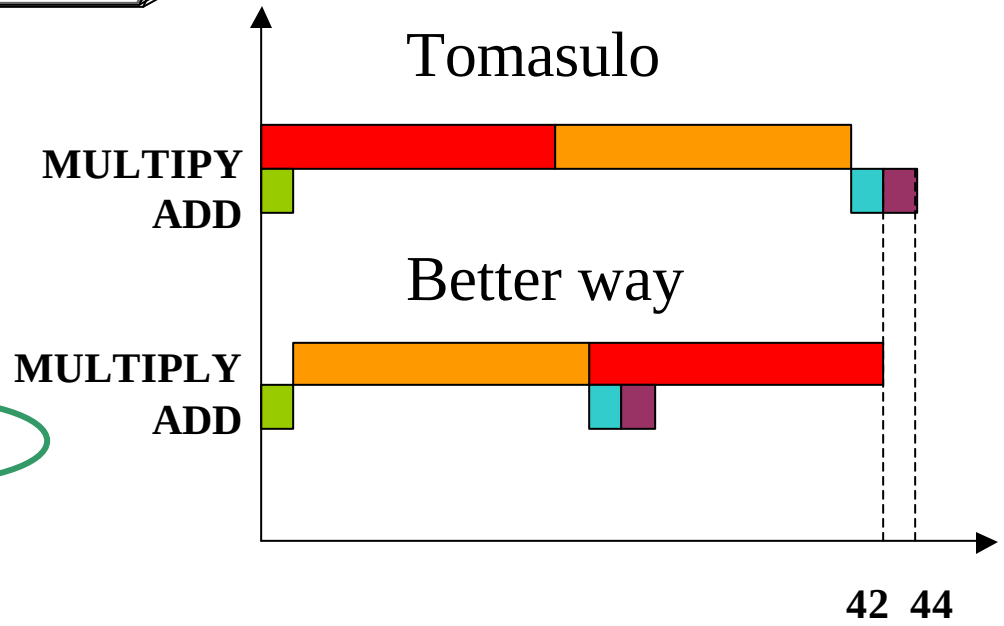


If it can be smarter further more.



There is no competition here. But if we just let FU idle one cycle, we can get better outcome.

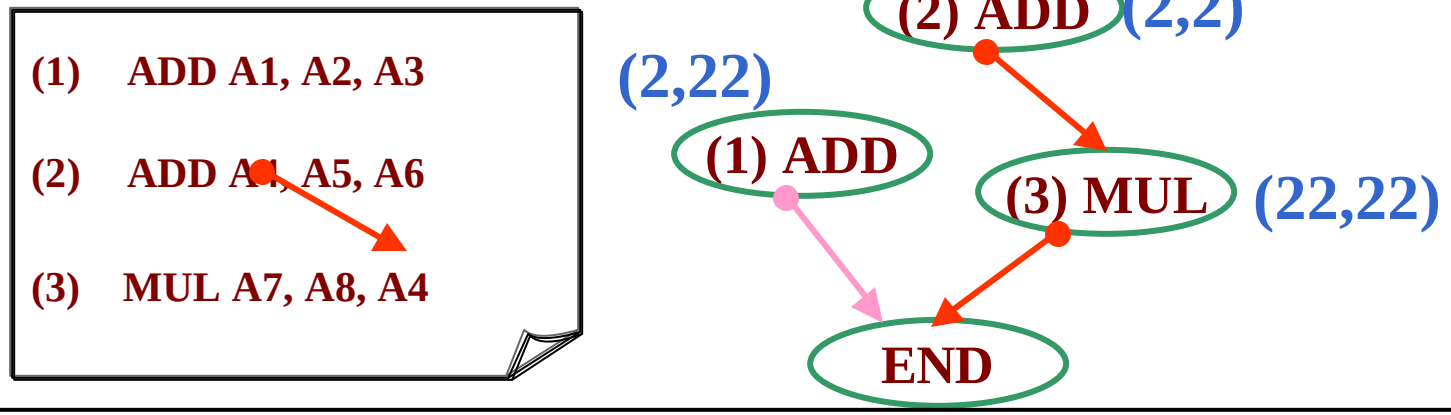


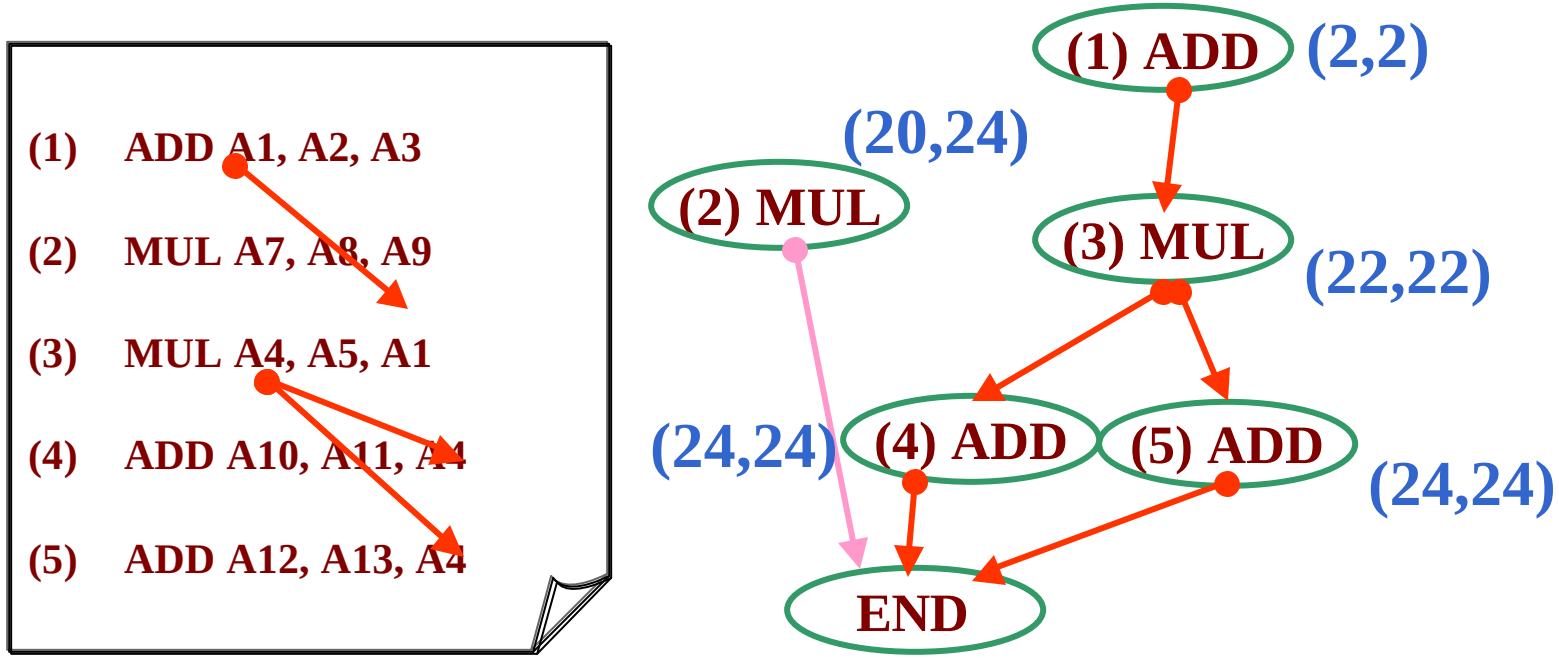


Improvement

- So far, only **“previous information”** is used.
  - Enough to deal with dependence problem.
  - Far less for wise scheduling.
- **“Following information”** is valuable for scheduling previous instruction.
  - Construct the critical path information in RS
  - Instructions in the critical path should get higher priority.
  - Following instruction entered RS should affect previous instructions’ priority in some way.
  - Introducing earliest possible finish time and latest necessary finish time to recode the information of critical path.
- Again, scheduling under resource competition is an **NP-complete** problem. It’s still true for our improvement.

Improvement(cont.)





Improvement(cont.)

- Is this always better than Tomasulo's when facing single FU? (LIKELY)
- Is it optimal in this condition? (HOPEFULLY)
- Is this always better than Tomasulo's when facing multiple FUs? (NOT SURE)
- Is it optimal in this condition? (NEVER)

Our Result

- Unlimited hardware resources, no delay – optimal ☺
- With transmission delay – optimal ☺
- With limited instruction window – optimal ☺
- With Load/Store – not optimal ☹
- With limited hardware resources – not optimal ☹
- **Issue one instruction per cycle – not optimal ☹**

Answers to our previous questions.

- Is it optimal?
  - The essence of the problem underlying is NP-Complete.
- Is there any room for improvement if it is not optimal?
  - Yes, some problem comes from Tomasulo algorithm itself.
    - Conservative Load and store buffer.
    - Greedy dispatch.
    - FIFO strategy to solve ties in competition.
  - Two ways to verify the our improvement.
- Is it a wise trade-off between time and complexity?
  - Parallelism, SMT, ILP
  - Cost-efficient.

IT'S TIME FOR DINNER.