北京大学
PEKING UNIVERSITY

# 智能硬件体系结构

## 第六讲：乱序执行微架构设计

主讲：陶耀宇、李萌

2024年秋季

# 注意事项

北京大学 PEKING UNIVERSITY

- **课程作业情况**
  - 第2次作业将在**10月31日**截止
  - 第1次作业答案已放出
  - 第1次编程实验已放出
    - 具体详细信息请参考：https://aiarchpku.github.io/2024Fall//project/
    - **10月22号~11月25号**
  - 本周五下午2-3点的2教308安排一次习题课，讲解作业1的答案
  - 下周会安排一次Lab答疑，请同学们先熟悉和理解项目代码结构
    - 自行搭建简单Module模块级测试验证环境，把verilog代码跑起来

# 注意事项

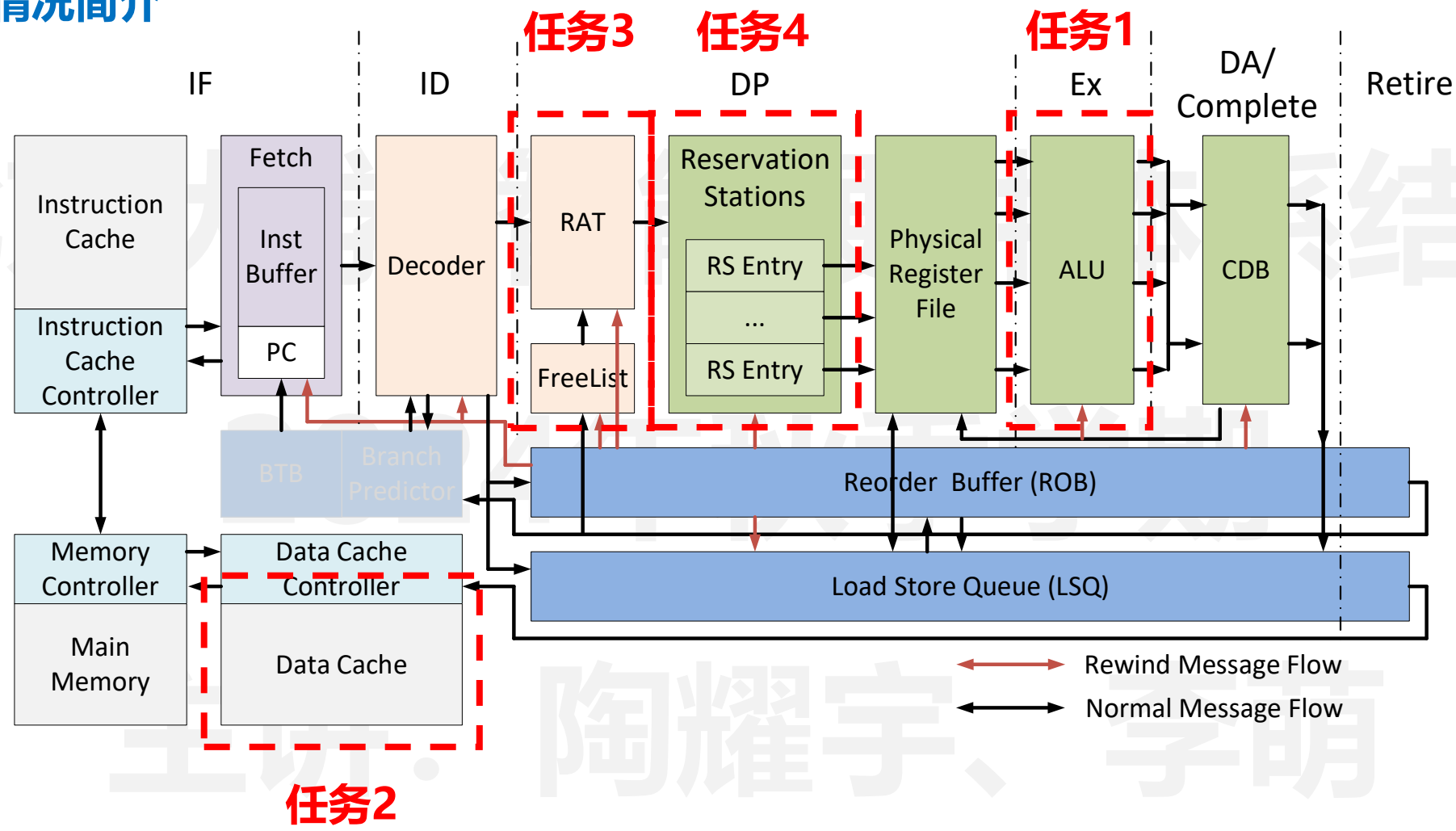
北京大学 PEKING UNIVERSITY

## • Lab 1情况简介

- 项目采用CLAB平台：https://clab.pku.edu.cn/
- CLAB平台的具体使用方式请参考： CLAB使用手册
- 实验采用Linux环境进行开发
  - 所需软件环境已为各位同学安装好，无需自己配置环境
  - Linux运行Lab的说明请参考：Linux使用参考信息
- 如有任何问题，请联系授课老师或助教！

# 注意事项

- Lab 1情况简介




**基于开源MIPS RISC-Style指令集的Superscalar Out-of-Order Processor**

# 注意事项

北京大学 PEKING UNIVERSITY

- Lab 1实验任务简介

**本项目提供项目代码以及测试系统，同学们需要完成的模块包括以下4个：**

**任务1：ALU**

**任务2：DCACHEMEM**

**任务3：RAT/RATTABLE**

**任务4：SUPER_RS**



**请同学们将这4个模块中缺失的代码部分填充完整，缺失代码的起始部分标注如下：**

```
// ============ Start ====================
// ==========Descriptions of Functions=========
  /*
  待填充的代码模块
  */
// =============End====================
```

# 注意事项

北京大学 PEKING UNIVERSITY

- **Lab 1实验任务简介**

1. 在verilog/alu.v源文件中，填补简单ALU模块的功能设计，完成基本算术、逻辑、位移和分支计算功能，模块输入输出解释请参考课程网站Project部分的模块功能介绍，ALUOp需要完成的功能在sys_defs.vh中，可自行构建testbench_alu.v模块进行测试验证，**alu.v中需要填补3个代码块，每一个代码块大约10~20行左右；**
2. 在verilog/dcachemem.v源文件中，填补2-Way Cache的功能设计，完成双路读出、写入功能，可自行构建testbench_dcachemem.v模块进行测试验证，**dcachemem.v中需要填补2个代码块，每一个代码块大约20行左右；**
3. 在**verilog/ratable.v**源文件中，填补完成简单RATTABLE模块的功能设计，支持Architectural Register与Physical Register的映射与回收功能，**需要填补的1个代码块，大约10行左右；**在**verilog/rat.v**源文件中，填补freelist相关逻辑实现对空闲Physical Register的状态检测与更新；可参考testbench/testbench_rat.v构建测试模块进行验证，**需要填补的1个代码块，大约30行左右；**
4. 在**verilog/superrs.v**完成SUPER_RS模块的功能实际，将模块RS（rs.v中）连入SUPER_RS模块以支持超标量=2的指令发射功能，可参考testbench_RS.v构建测试模块进行验证，**需要填补的1个代码块，大约30行左右；**

# 注意事项

北京大学 PEKING UNIVERSITY

- **Lab 1项目文件结构**

| | |
|---|---|
| **---Makefile** | 包含所有make command |
| **---Make.*** | 包含子模块的make command，可新建子模块验证路径单独验证各个子模块<br>示例：make all　　可自动运行program.mem内的程序<br>make clean　　删除所有编译产生文件 |
| **---debug_out** | 仿真产生的.out文件，提供简单的可视化观察模块和流水线的状态 |
| **---vs-asm** | MIPS RISC指令的Assembler编译器，可以将test_progs路径下的指令汇编代码翻译成HEX格式，存入 program.mem文件供testbench文件夹下的tb读取<br>示例：vs-asm test_progs/fib.s > program.mem |
| **---pipeline_gold** | 最简单的顺序5级流水线设计，产生用作验证的参考write_back.out和memory.out，与本项目乱序执行超标量结果进行对比 |
| **---program.mem** | 汇编指令代码的HEX格式文件，有tb读取作为icache的输入 |
| **---run_tests.sh** | 运行该脚本可一次性验证test_progs内的所有汇编指令代码运行是否正确，并比较乱序超标量与顺序5级流水线的CPI |
| **---sys_defs.vh** | 定义架构设计的顶层配置参数，无需改动 |
| **---test_progs** | 包含所有用于测试验证的MIPS RISC-Style汇编代码 |
| **---testbench** | 包含项目所需的所有验证tb文件和可视化支持文件 |
| **---verilog** | 包含项目所需的所有架构设计Verilog源文件，具体解释详见模块说明 |

# 目 录

CONTENTS

# Superscalar：超标量的概念

- Instruction-level parallelism

**Instruction parallelism**

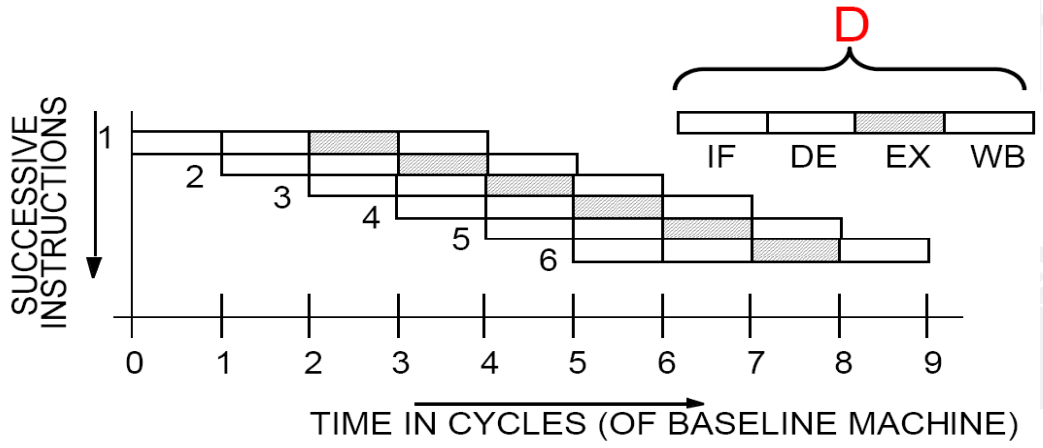
Number of instructions being worked on

Scalar Pipeline (baseline)

Instruction Parallelism = D

Operation Latency = 1

Peak IPC = 1




**Peak IPC**

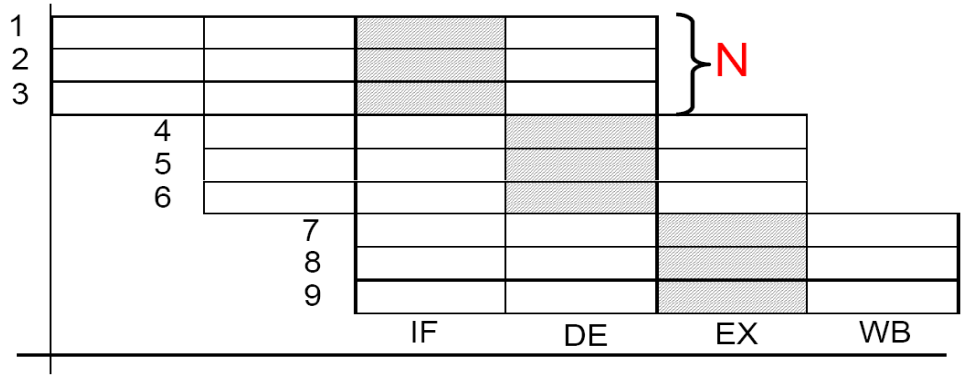
The maximum sustainable number of instructions that can be executed per clock.

Superscalar (Pipelined) Execution

IP = *DxN*

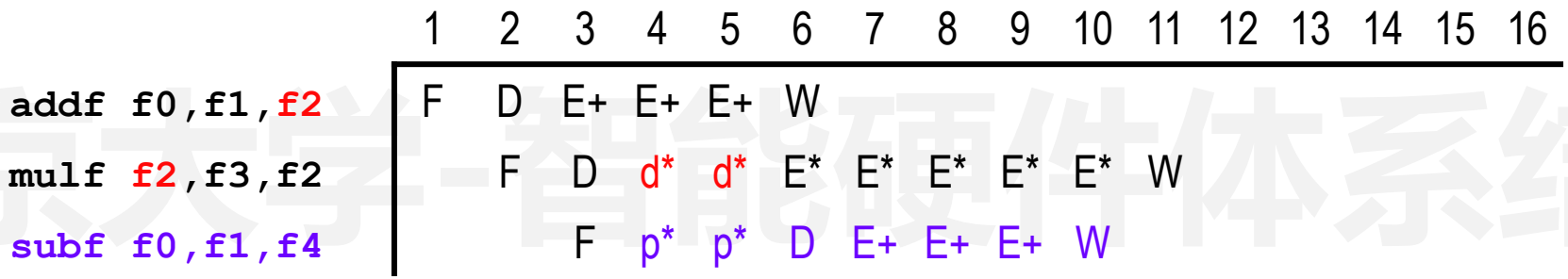
OL = *1 baseline cycles*

Peak IPC = *N per baseline cycle*



# Out-Of-Order：乱序执行的概念

- **Missed Speedup in In-Order Pipelines**




What’s happening in cycle 4?

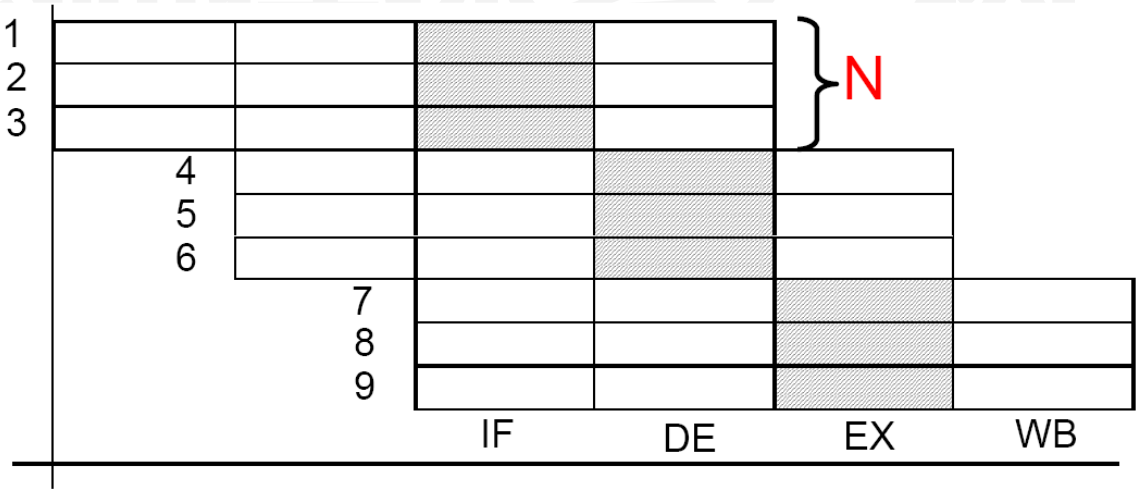
- **`mulf`** stalls due to **RAW hazard**
  - OK, this is a fundamental problem
- **`subf`** stalls due to **pipeline hazard**
  - Why? **`subf`** can’t proceed into D because **`mulf`** is there
  - That is the only reason, and it isn’t a fundamental one

Why can’t **`subf`** go into D in cycle 4 and E+ in cycle 5?

# Out-Of-Order：乱序执行的概念

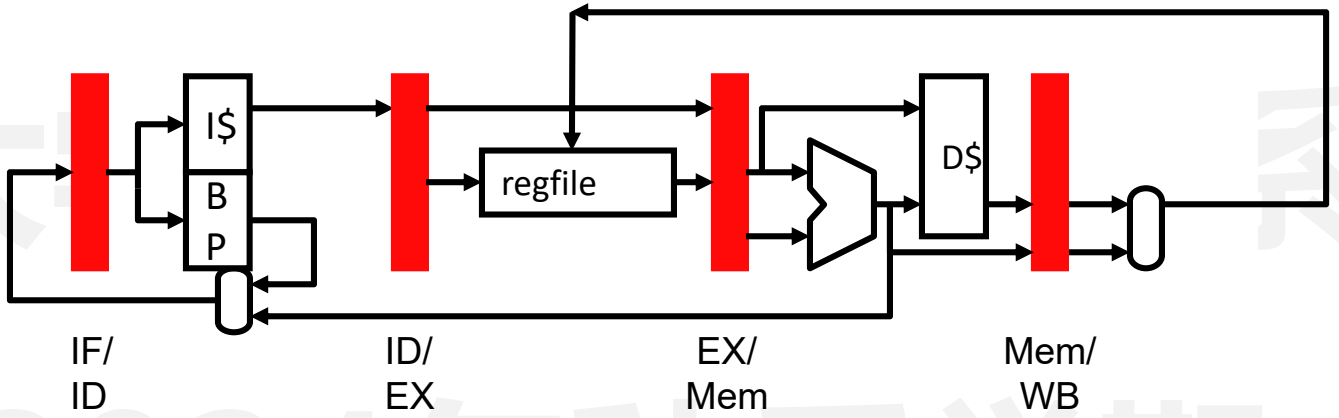
- Instruction-level parallelism

- **CPI of in-order pipelines degrades sharply if the machine parallelism is increased beyond a certain point.**
  - *when NxM approaches average distance between dependent instructions*
- Forwarding is no longer effective
  - *Pipeline may never be full due to frequent dependency stalls!*



# Out-Of-Order：乱序执行的概念

- The Problem With In-Order Pipelines




- **In-order pipeline**
  - **Structural hazard: 1 insn register (latch) per stage**
    - 1 instruction per stage per cycle (unless pipeline is replicated)
    - **Younger instr. can't "pass" older instr. without "clobbering" it**
- **Out-of-order pipeline**
  - Implement "passing" functionality by removing **structural hazard**

# Out-Of-Order：乱序执行的概念

北京大学 PEKING UNIVERSITY

- 乱序执行完全在硬件实现

- **Dynamic scheduling**
  - Totally in the hardware
  - Also called "out-of-order execution" (OoO)
- Fetch many instructions into instruction window
  - Use branch prediction to speculate past (multiple) branches
  - Flush pipeline on branch misprediction
- **Rename to avoid false dependencies (WAW and WAR)**
- **Execute instructions as soon as possible**
  - Register dependencies are known
  - Handling memory dependencies more tricky (much more later)
- **Commit instructions in order**
  - Any strange happens before commit, just flush the pipeline
- Current machines: **100+ instruction scheduling window**

**Out-of-order execution**

Execute instructions in non-sequential order…

**+Reduce RAW stalls**

**+Increase pipeline and functional unit (FU) utilization**

Original motivation was to increase FP unit utilization

**+Expose more opportunities for parallel issue (ILP)**

Not in-order → can be in parallel

…but make it appear like sequential execution

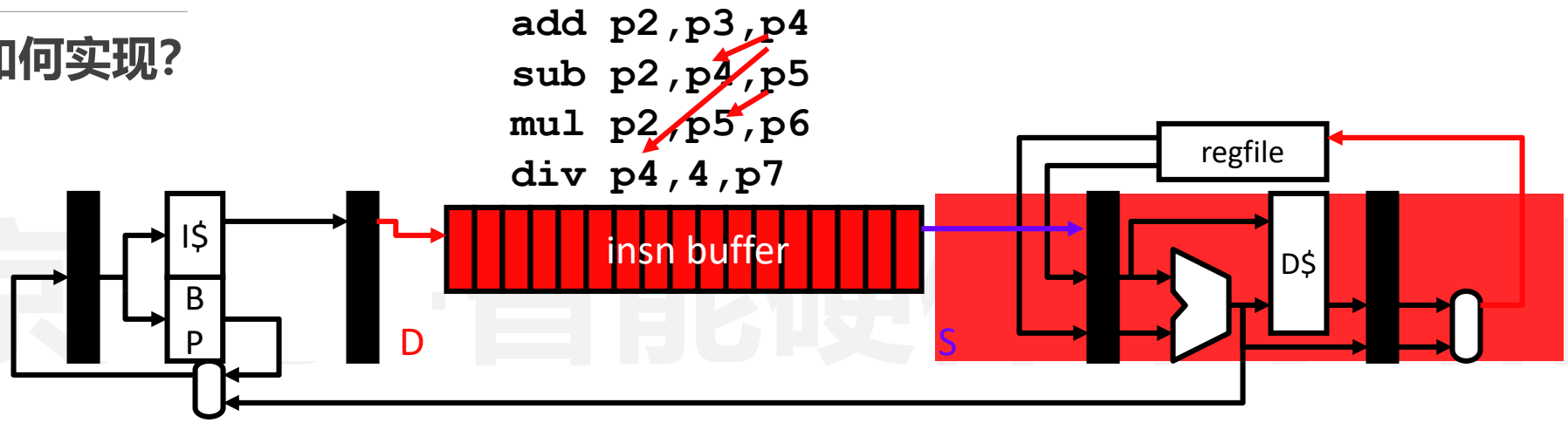
Important

–But difficult

Next few lectures

# Out-Of-Order：乱序执行的概念

- 乱序执行如何实现?




Ready Table

| P2 | P3 | P4 | P5 | P6 | P7 |
|---|---|---|---|---|---|
| Yes | Yes | | | | |
| Yes | Yes | Yes | | | |
| Yes | Yes | Yes | Yes | | Yes |
| Yes | Yes | Yes | Yes | Yes | Yes |

t

```
add p2,p3,p4
sub p2,p4,p5   and  div p4,4,p7
mul p2,p5,p6
```

- Instructions fetch/decoded/renamed into *Instruction Buffer*
  - Also called "instruction window" or "instruction scheduler"
- Instructions (conceptually) check ready bits every cycle
  - Execute immediately when ready

# 数据依赖与数据冲突

- 数据依赖存在于原始任务逻辑，与硬件体系结构如何设计无关

- **A dependency（依赖） exists *independent* of the hardware.**
  - So if Inst #1’s result is needed for Inst #1000 there is a dependency
  - **It is only a *hazard（冲突）* if the hardware has to deal with it.**
    - So in our pipelined machine we only worried if there wasn’t a “buffer” of two instructions between the dependent instructions.

# 数据依赖

- True/False Data Dependencies

- **True data dependency**
  - RAW – Read after Write

    R1 = R2 + R3

    R4 = R1 + R5

- True dependencies prevent reordering
  - **(Mostly) unavoidable**

- **False or Name dependencies**
  - WAW – Write after Write

    R1 = R2 + R3

    ↓

    R1 = R4 + R5
  - WAR – Write after Read

    R2 = R1 + R3

    ↓

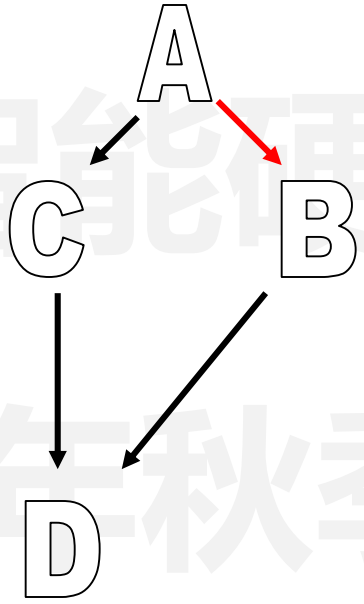
    R1 = R4 + R5

- False dependencies prevent reordering
  - **Can they be eliminated? (Yes, with renaming!)**

# 数据依赖图

- True/False Data Dependencies

```
R1=MEM[R2+0]    // A
R2=R2+4         // B
R3=R1+R4        // C
MEM[R2+0]=R3    // D
```

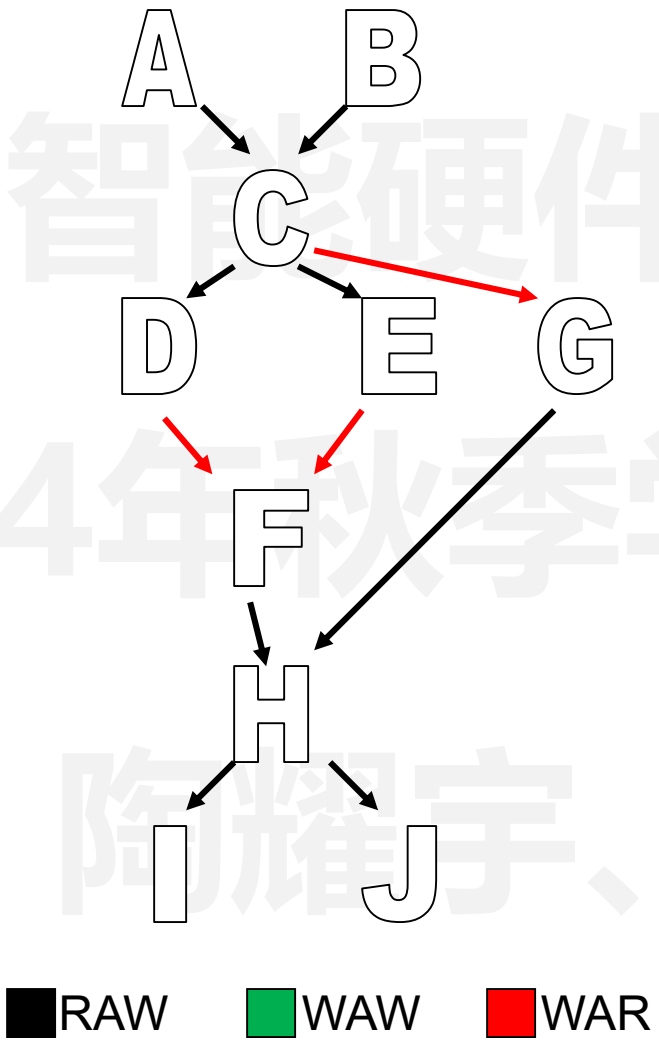



RAW WAW WAR

# 数据依赖图

- True/False Data Dependencies

```
R1=MEM[R3+4]     // A
R2=MEM[R3+8]     // B
R1=R1*R2         // C
MEM[R3+4]=R1     // D
MEM[R3+8]=R1     // E
R1=MEM[R3+12]    // F
R2=MEM[R3+16]    // G
R1=R1*R2         // H
MEM[R3+12]=R1    // I
MEM[R3+16]=R1    // J
```



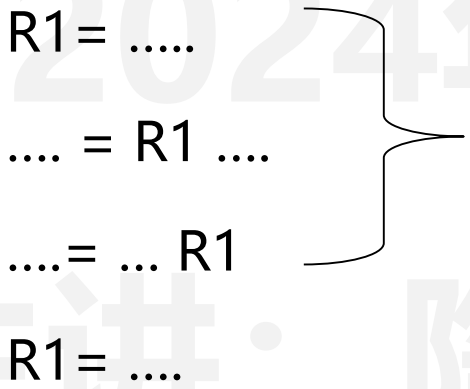

- Well, logically there is no reason for F-J to be dependent on A-E. So…..
  - ABFG
  - CH
  - DEIJ
  - Should be possible.
- But that would cause either C or H to have the wrong reg inputs
- How do we fix this?
  - Remember, the dependency is really on the *name* of the register
  - **So… change the register names!**

# 寄存器重命名

北京大学 PEKING UNIVERSITY

- 寄存器Register重命名概念

- The register names are arbitrary
- **The register name only needs to be consistent between writes.**




**The value in R1 is “alive” from when the value is written until the last read of that value. (Or right Before the next write of R1)**

# 寄存器重命名

- 寄存器Register重命名机制
  - **Every time an architecture register is written we assign it to a physical register**
    - Until the architected register is written again, we continue to translate it to the physical register number
    - Leaves **RAW** dependencies intact
  - It is really simple, let’s look at an example:
    - Architecture Regs: `r1,r2,r3`
    - Physical Regs: `p1,p2,p3,p4,p5,p6,p7`
    - Original mapping: `r1→p1, r2→p2, r3→p3, p4–p7` **are “free”**

**Architecture register**
**虚拟的架构寄存器 – 汇编代码中**

**Physical register**
**实际的电路寄存器 – 硬件架构中**

**RAT (Alias T)**

| r1 | r2 | r3 |
|---|---|---|
| p1 | p2 | p3 |
| p4 | p2 | p3 |
| p4 | p2 | p5 |
| p4 | p2 | p6 |

**FreeList**

| |
|---|
| p4,p5,p6,p7 |
| p5,p6,p7 |
| p6,p7 |
| p7 |

**Orig. insns**

```
add r2,r3,r1
sub r2,r1,r3
mul r2,r3,r3
div r1,4,r1
```

**Renamed insns**

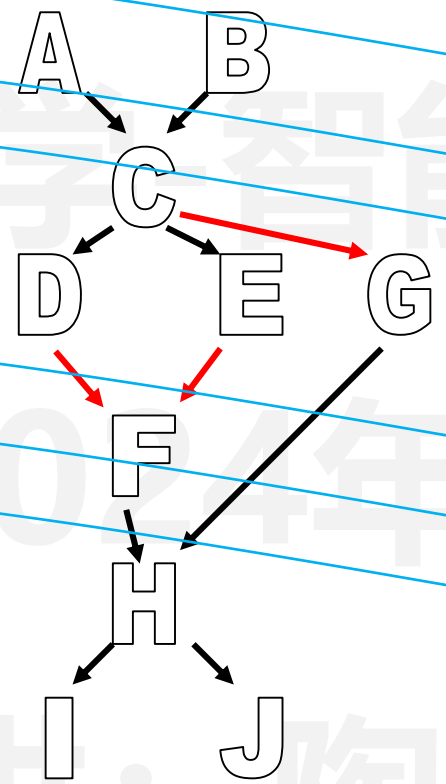
```
add p2,p3,p4
sub p2,p4,p5
mul p2,p5,p6
div p4,4,p7
```

# 寄存器重命名

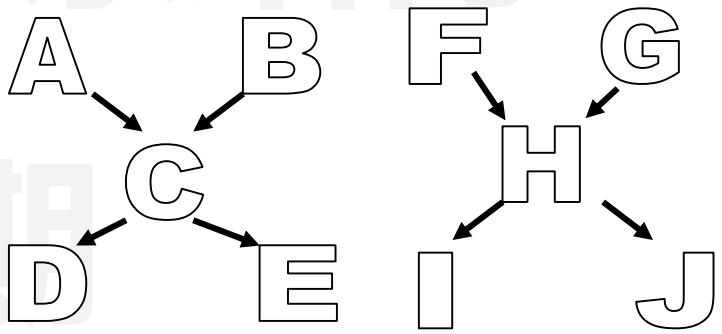
- 寄存器Register重命名的效果

每次写入一个Arch Reg的时候，赋予一个新的Phy Reg

```
R1=MEM[R3+4]     // A
R2=MEM[R3+8]     // B
R1=R1*R2         // C
MEM[R3+4]=R1     // D
MEM[R3+8]=R1     // E
R1=MEM[R3+12]    // F
R2=MEM[R3+16]    // G
R1=R1*R2         // H
MEM[R3+12]=R1    // I
MEM[R3+16]=R1    // J
```




```
P1=MEM[R3+4]     //A
P2=MEM[R3+8]     //B
P3=P1*P2         //C
MEM[R3+4]=P3     //D
MEM[R3+8]=P3     //E
P4=MEM[R3+12]    //F
P5=MEM[R3+16]    //G
P6=P4*P5         //H
MEM[R3+12]=P6    //I
MEM[R3+16]=P6    //J
```




RAW WAW WAR

# 寄存器重命名



- 寄存器Register重命名硬件设计



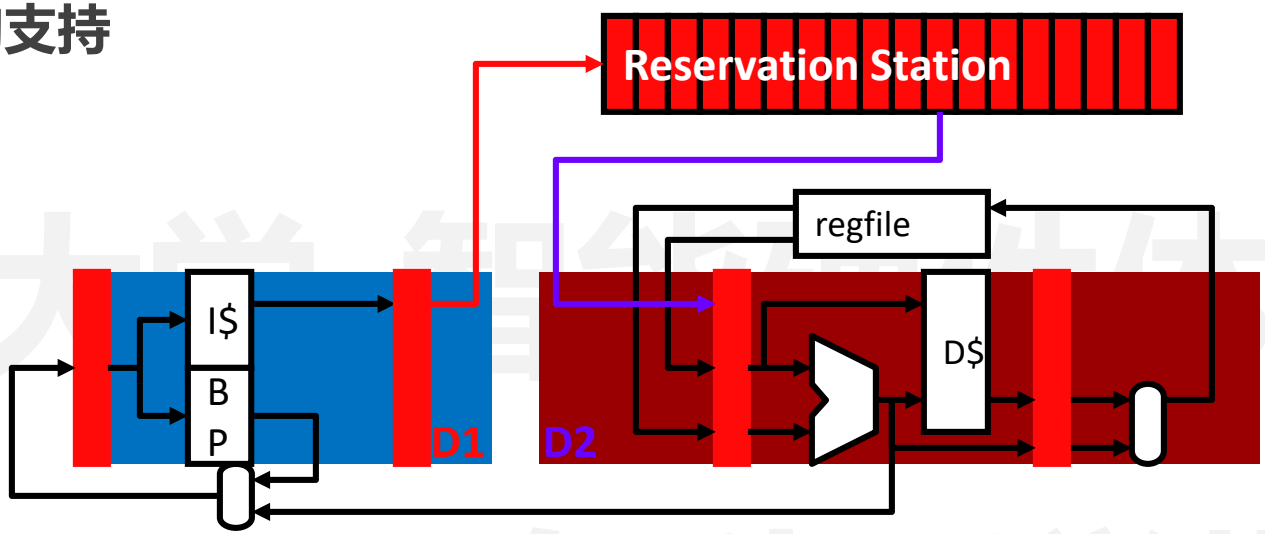
- Really simple table (Reg Alias Table, RAT)
  - **Every time an instruction *which writes a register* is encountered assign it a new physical register number**
- But there is some complexity
  - **When do you free physical registers?**
  - **Next chapter with OoO architecture**

# 目 录

CONTENTS

# 动态发射与乱序执行设计

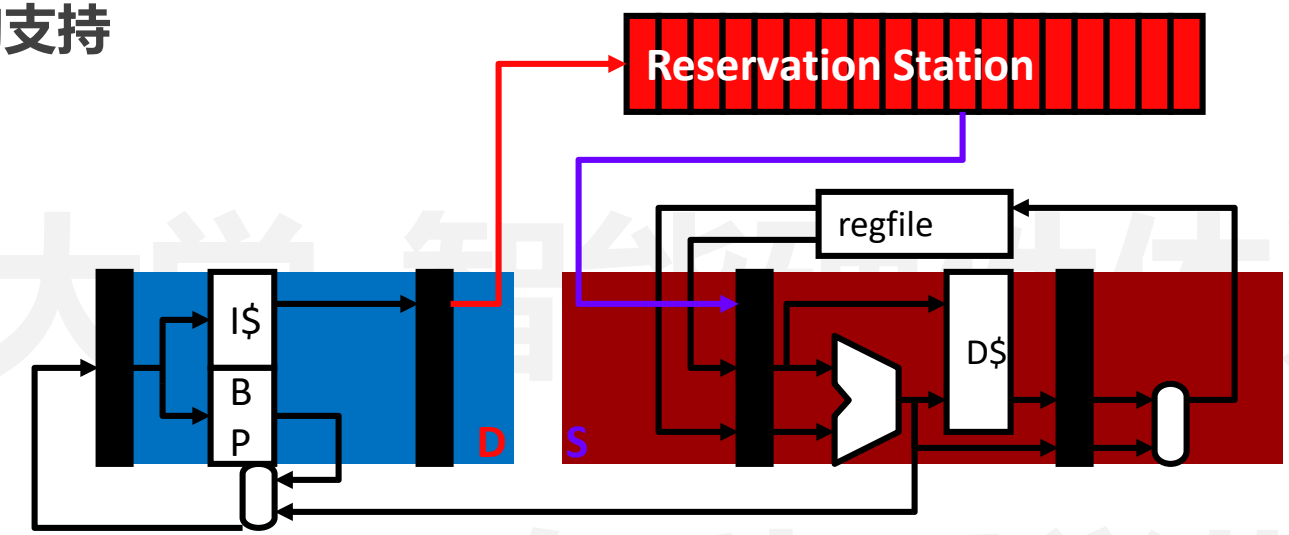
- 乱序执行的架构支持




- **Insn buffer or Reservation station (RS)** (many names for this buffer)
  - Basically: a bunch of latches for holding insns
  - Candidate pool of instructions
- Split ID into two pieces
  - Accumulate decoded insns in buffer **in-order**
  - RS sends insns down rest of pipeline **out-of-order**

# 动态发射与乱序执行设计

- 乱序执行的架构支持




- **Dispatch (D)**: first part of decode
  - Allocate slot in RS insn buffer
    - New kind of structural hazard (insn buffer is full)
  - In order: **stall** back-propagates to younger insns
- **Issue (S)**: second part of decode
  - Send insns from RS insn buffer to execution units
  + Out-of-order: **wait** doesn’ t back-propagate to younger insns

# 动态发射与乱序执行设计

北京大学 PEKING UNIVERSITY

- 指令动态发射算法
  - **Register scheduler**: scheduler driven by register dependences
  - **Two basic register scheduling algorithms**
    - Scoreboard: No register renaming → limited scheduling flexibility
    - Tomasulo: Register renaming → more flexibility, better performance
    - We focus on Tomasulo’ s algorithm in the lecture
    - No test questions on scoreboarding
      - Do note that it is used in certain GPUs.
  - **Issue**
    - If multiple instructions are ready, which one to choose? **Issue policy**
      - Oldest first? Safe
      - Longest latency first? May yield better performance
    - **Select logic**: implements issue policy
      - Most projects use random or priority encoder

# Tomasulo动态指令发射算法

北京大学 PEKING UNIVERSITY

- 指令动态发射算法

- **Tomasulo’s algorithm**
  - **Reservation stations (RS)**: instruction buffer
  - **Common data bus (CDB)**: broadcasts results to RS
  - **Register renaming**: removes WAR/WAW hazards

- First implementation: IBM 360/91 -> Modern x86-x86 Still use!
  - Dynamic scheduling for FP units only

- Our simple example: “Simple Tomasulo”
  - Dynamic scheduling for everything, including load/store
  - 5 RS: 1 ALU, 1 load, 1 store, 2 FP (3-cycle, pipelined)

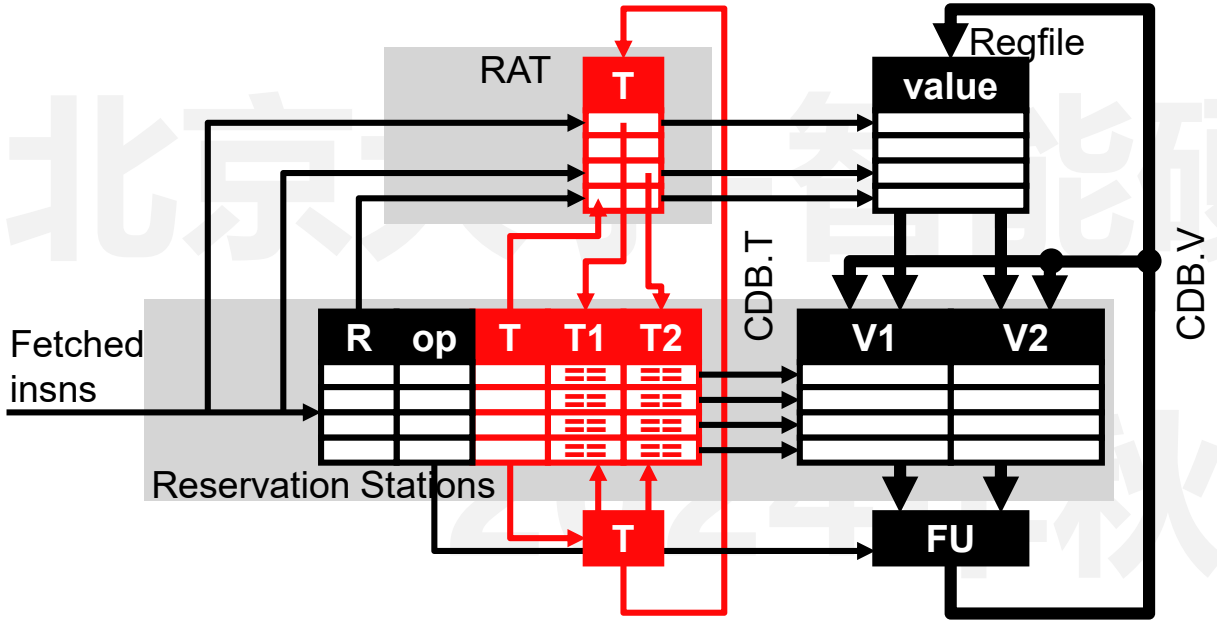
# Tomasulo动态指令发射算法

北京大学 PEKING UNIVERSITY

- **Tomasulo算法的基础结构**
  - Reservation Stations (RS#)
    - **FU**, **busy**, **op**, **R**: destination register name
    - **T**: destination register tag (RS# of this RS)
    - **T1**,**T2**: source register tags (RS# of RS that will produce value)
    - **V1**,**V2**: source register values
  - Rename Table/Map Table/RAT
    - **T**: tag (RS#) that will write this register
  - Common Data Bus (CDB)
    - Broadcasts <RS#, value> of completed insns
  - Tags interpreted as ready-bits++
    - T==0 → Value is ready somewhere
    - T!=0 → Value is not ready, wait until CDB broadcasts T

# Tomasulo动态指令发射算法

- Tomasulo算法的基础结构




- Insn fields and status bits
- Tags
- Values

- Reservation Stations (RS#)
  - **FU**, **busy**, **op**, **R**: destination register name
  - **T**: destination register tag (RS# of this RS)
  - **T1**,**T2**: source register tags (RS# of RS that will produce value)
  - **V1**,**V2**: source register values
- Rename Table/Map Table/RAT
  - **T**: tag (RS#) that will write this register
- Common Data Bus (CDB)
  - Broadcasts <RS#, value> of completed insns
- Tags interpreted as ready-bits++
  - T==0 → Value is ready somewhere
  - T!=0 → Value is not ready, wait until CDB broadcasts T

# Tomasulo动态指令发射算法

北京大学 PEKING UNIVERSITY

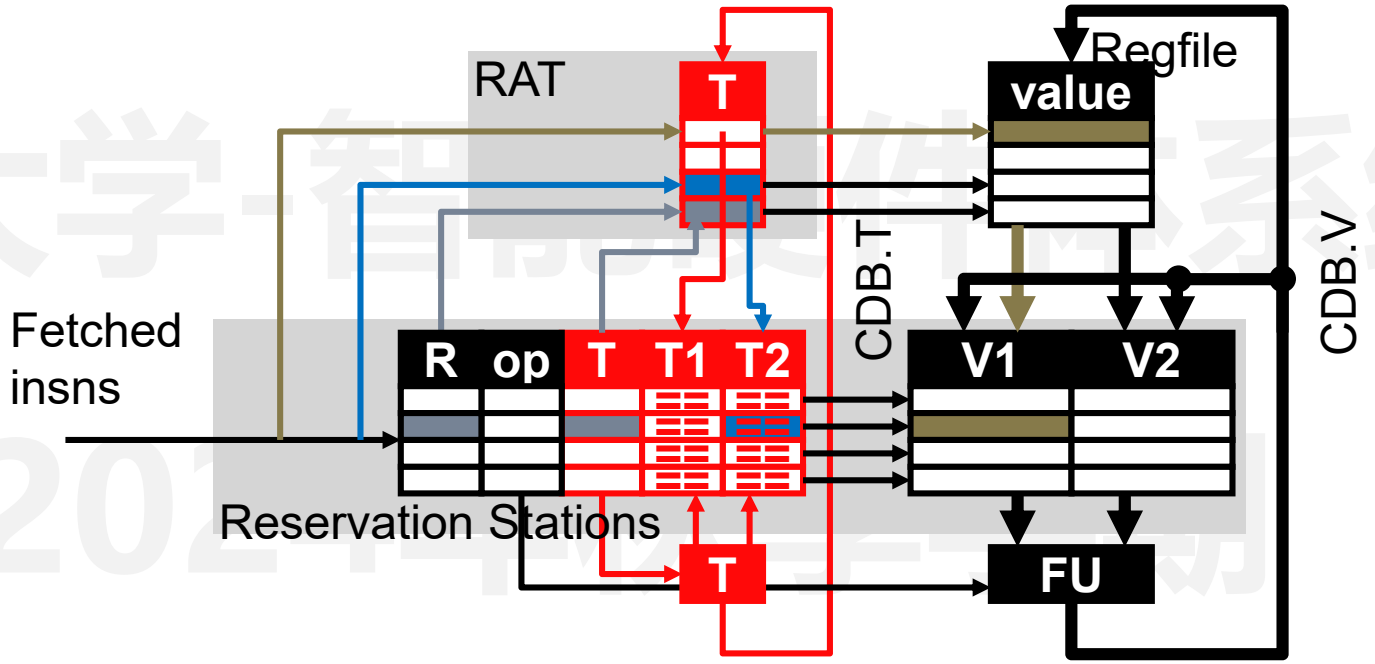
- Tomasulo算法的新增步骤

  - New pipeline structure: F, **D**, **S**, X, **W**
    - **D (dispatch)**
      - **Structural** hazard ? **stall** : allocate RS entry
    - **S (issue)**
      - **RAW** hazard ? **wait** (monitor CDB) : go to execute
    - **W (writeback)**
      - Write register (sometimes…), free RS entry
      - W and RAW-dependent S in same cycle
      - W and structural-dependent D in same cycle

# Tomasulo动态指令发射算法

- Tomasulo算法步骤

## Tomasulo Dispatch (D)

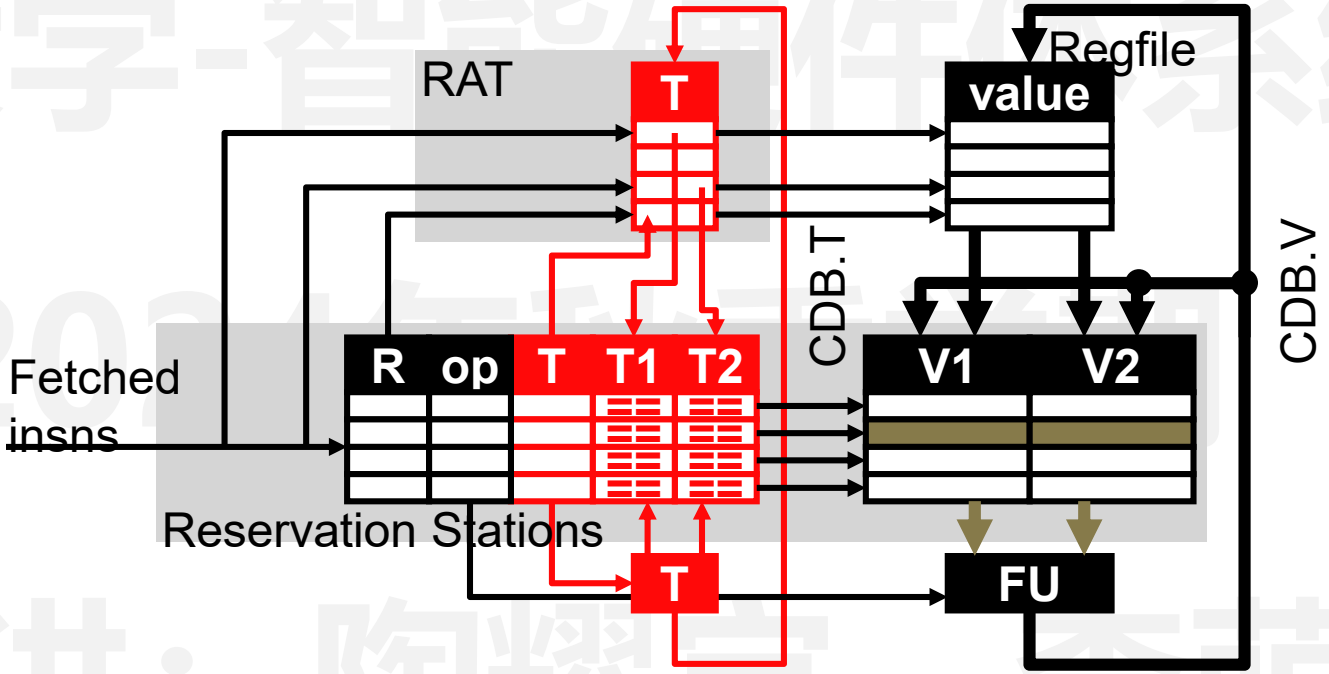



- Stall for structural (RS) hazards
  - Allocate RS entry
  - Input register ready ? read value into RS : read tag into RS
  - Rename output register to RS # (represents a unique value “name”)

# Tomasulo动态指令发射算法

- Tomasulo算法步骤

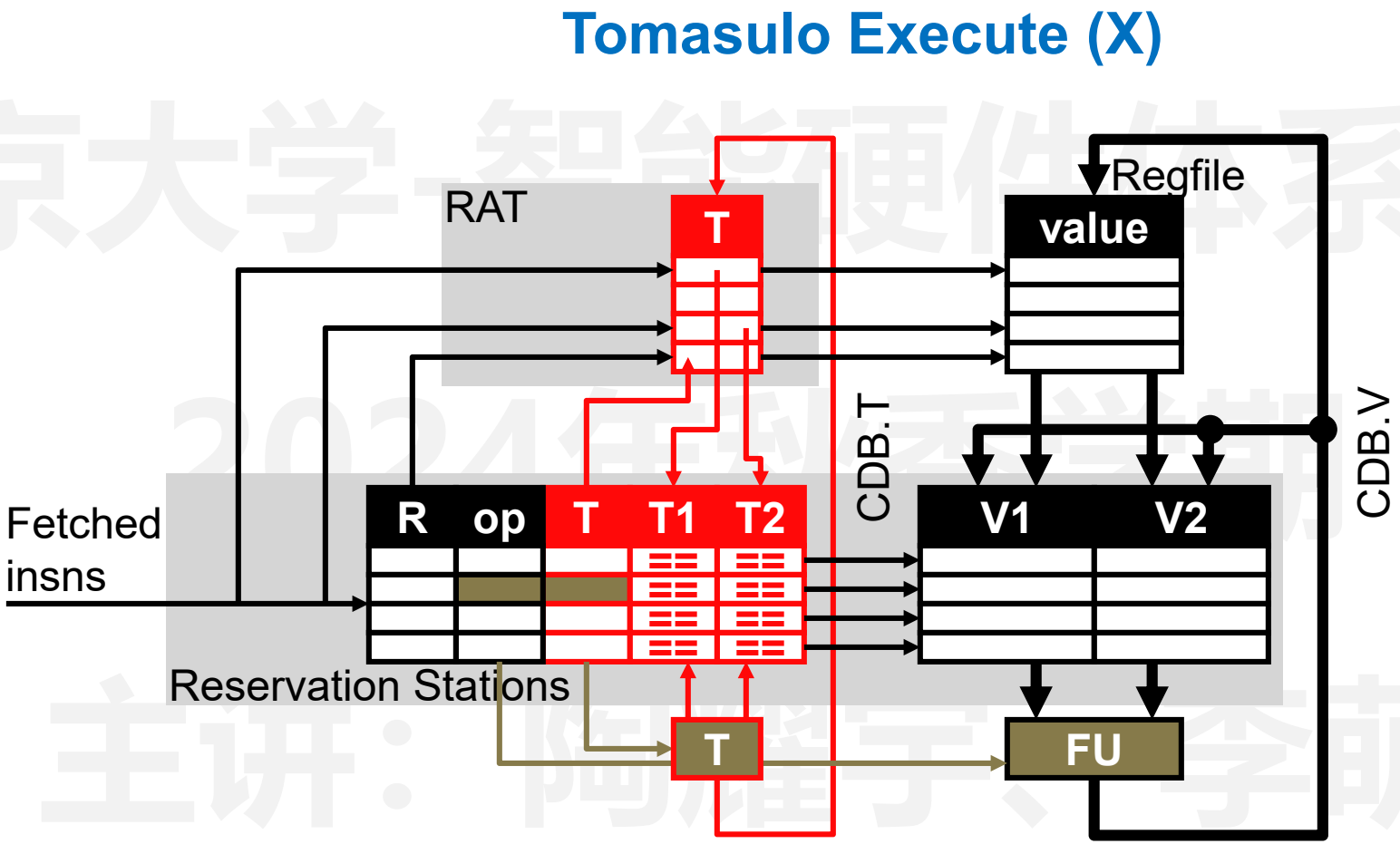
## Tomasulo Issue (S)




- Wait for RAW hazards
  - Read register values from RS

# Tomasulo动态指令发射算法

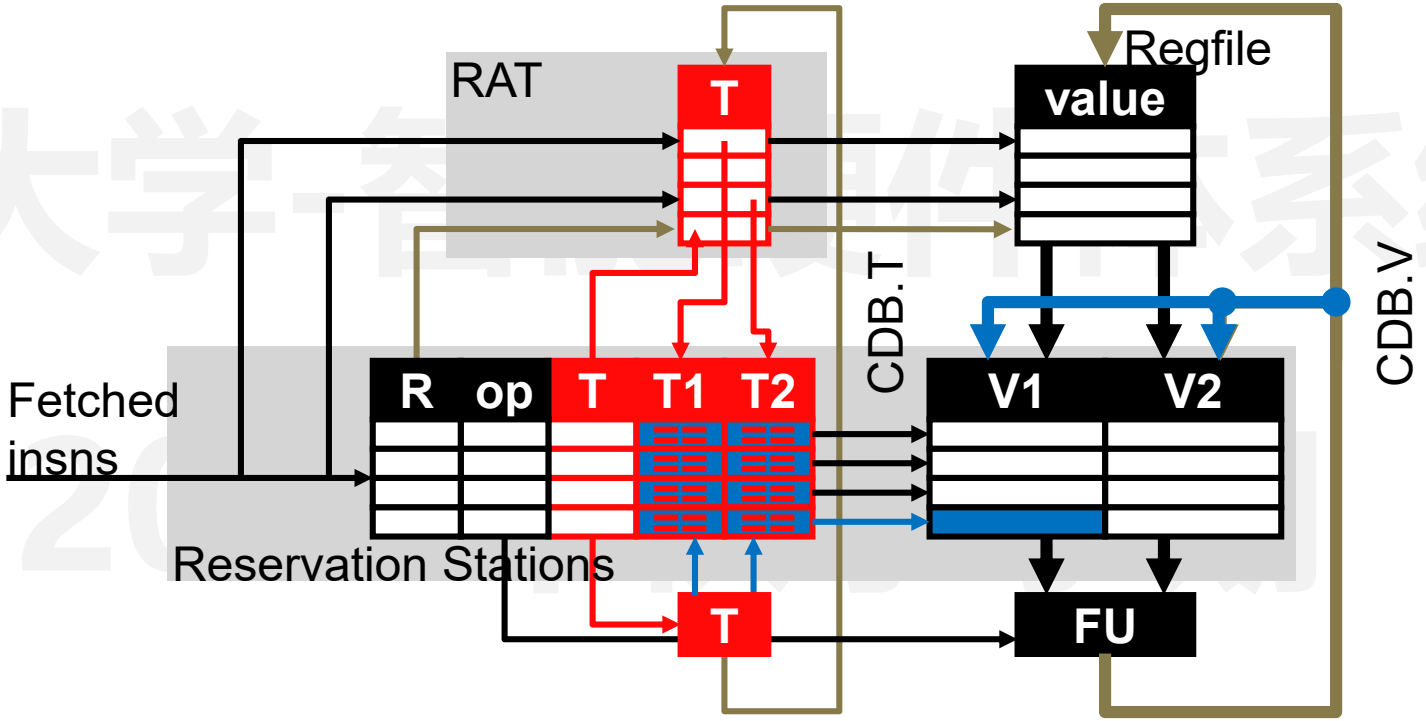
- Tomasulo算法步骤



# Tomasulo动态指令发射算法

- Tomasulo算法步骤

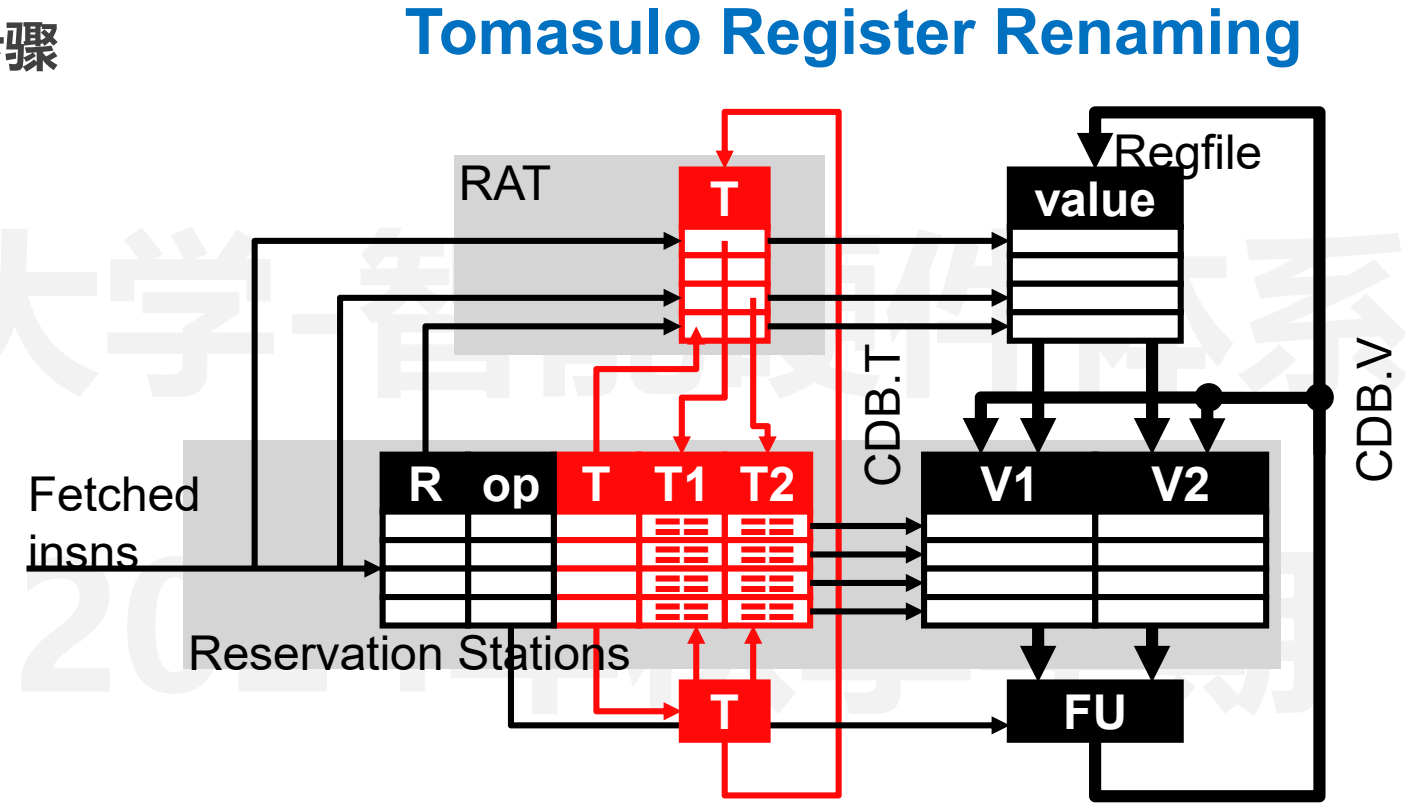
## Tomasulo Writeback (W)




- Wait for structural (CDB) hazards
  - If RAT rename still matches ? Clear mapping, write result to regfile
  - CDB broadcast to RS: tag match ? clear tag, copy value
  - Free RS entry

# Tomasulo动态指令发射算法

- Tomasulo算法步骤




- What in Tomasulo implements **register renaming**?
  - **Value copies in RS (V1, V2)**
  - Insn stores correct input values in its own RS entry

  + Future insns can overwrite master copy in regfile, doesn’ t matter

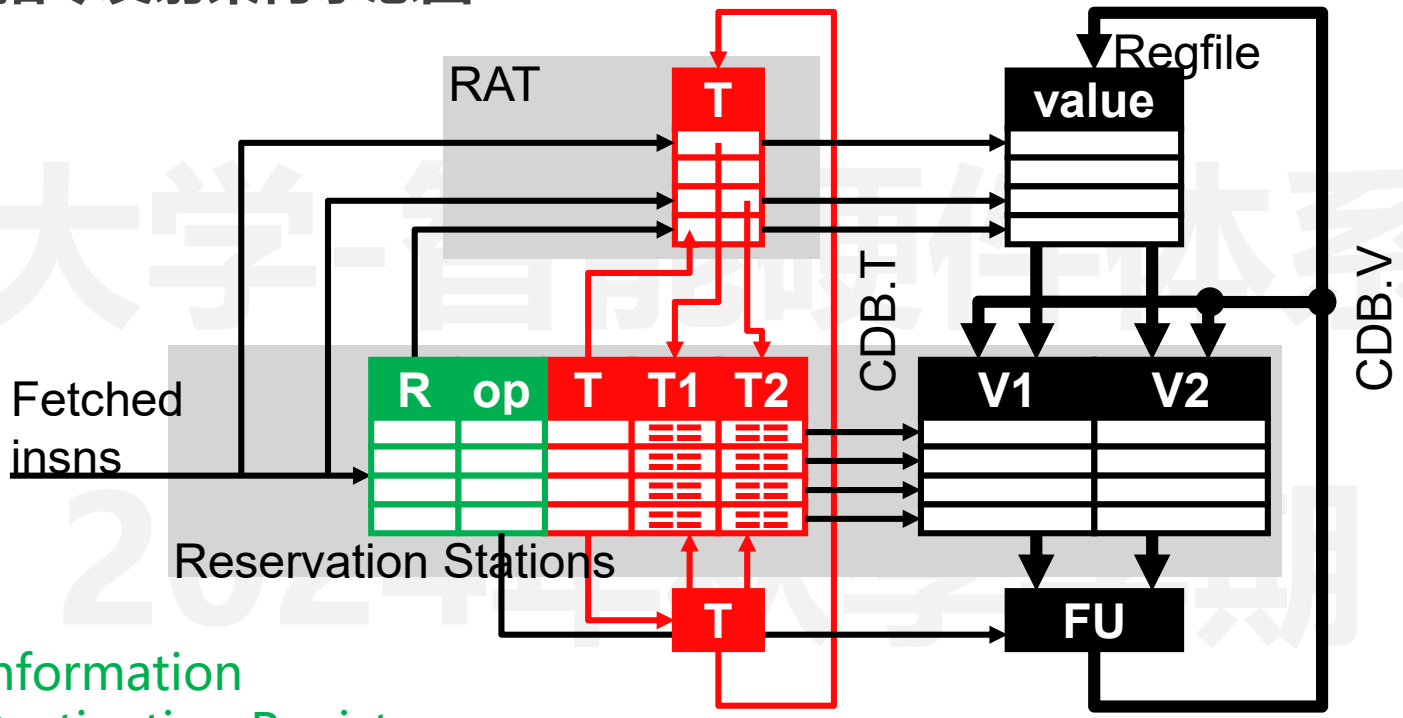
# Tomasulo动态指令发射算法

北京大学 PEKING UNIVERSITY

- **Value-based / Copy-based Register Renaming**
  - Tomasulo-style register renaming
    - Called "**value-based**" or "**copy-based**"
    - **Names**: architectural registers
    - **Storage locations: register file and reservation stations**
      - Values can and do exist in both
      - **Register file holds master (i.e., most recent) values**
      + **RS copies eliminate WAR hazards**
    - Storage locations referred to internally by RS# tags
      - **Register table translates names to tags**
      - Tag == 0 value is in register file
      - Tag != 0 value is not ready and is being computed by RS#
    - CDB broadcasts values with tags attached
      - So insns know what value they are looking at

# Tomasulo动态指令发射算法

- Tomasulo动态指令发射架构示意图




- RS:
  - Status information
    - R: Destination Register
    - op: Operand (add, etc.)
  - Tags
    - T1, T2: source operand tags
  - Values
    - V1, V2: source operand values

- Map table (also RAT: Register Alias Table)
  - Maps registers to tags
- Regfile (also ARF: Architected Register File)
  - Holds value of register if no value in RS

# Tomasulo动态指令发射算法

- Tomasulo动态指令发射实例

| Insn Status | | | | |
|---|---|---|---|---|
| Insn | D | S | X | W |
| ldf X(r1),f1 | | | | |
| mulf f0,f1,f2 | | | | |
| stf f2,Z(r1) | | | | |
| addi r1,4,r1 | | | | |
| ldf X(r1),f1 | | | | |
| mulf f0,f1,f2 | | | | |
| stf f2,Z(r1) | | | | |

| Map Table | |
|---|---|
| Reg | T |
| f0 | |
| f1 | |
| f2 | |
| r1 | |

| CDB | |
|---|---|
| T | V |
| | |

| Reservation Stations | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| T | FU | busy | op | R | T1 | T2 | V1 | V2 |
| 1 | ALU | no | | | | | | |
| 2 | LD | no | | | | | | |
| 3 | ST | no | | | | | | |
| 4 | FP1 | no | | | | | | |
| 5 | FP2 | no | | | | | | |

# Tomasulo动态指令发射算法

- Tomasulo动态指令发射实例

**Tomasulo:**

**Cycle 1**

| Insn Status | | | | |
|---|---|---|---|---|
| Insn | D | S | X | W |
| ldf X(r1),f1 | c1 | | | |
| mulf f0,f1,f2 | | | | |
| stf f2,Z(r1) | | | | |
| addi r1,4,r1 | | | | |
| ldf X(r1),f1 | | | | |
| mulf f0,f1,f2 | | | | |
| stf f2,Z(r1) | | | | |

| Map Table | |
|---|---|
| Reg | T |
| f0 | |
| f1 | RS#2 |
| f2 | |
| r1 | |

| CDB | |
|---|---|
| T | V |
| | |

| Reservation Stations | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| T | FU | busy | op | R | T1 | T2 | V1 | V2 |
| 1 | ALU | no | | | | | | |
| 2 | LD | yes | ldf | f1 | - | - | - | [r1] |
| 3 | ST | no | | | | | | |
| 4 | FP1 | no | | | | | | |
| 5 | FP2 | no | | | | | | |

allocate

# Tomasulo动态指令发射算法

- Tomasulo动态指令发射实例

**Tomasulo:**

**Cycle 2**

| Insn Status | | | | |
|---|---|---|---|---|
| Insn | D | S | X | W |
| ldf X(r1),f1 | c1 | c2 | | |
| mulf f0,f1,f2 | c2 | | | |
| stf f2,Z(r1) | | | | |
| addi r1,4,r1 | | | | |
| ldf X(r1),f1 | | | | |
| mulf f0,f1,f2 | | | | |
| stf f2,Z(r1) | | | | |

| Map Table | |
|---|---|
| Reg | T |
| f0 | |
| f1 | RS#2 |
| f2 | RS#4 |
| r1 | |

| CDB | |
|---|---|
| T | V |
| | |

| Reservation Stations | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| T | FU | busy | op | R | T1 | T2 | V1 | V2 |
| 1 | ALU | no | | | | | | |
| 2 | LD | yes | ldf | f1 | - | - | - | [r1] |
| 3 | ST | no | | | | | | |
| 4 | FP1 | yes | mulf | f2 | - | RS#2 | [f0] | - |
| 5 | FP2 | no | | | | | | |

allocate

# Tomasulo动态指令发射算法

- Tomasulo动态指令发射实例

**Tomasulo:**

**Cycle 3**

| Insn Status | | | | |
|---|---|---|---|---|
| Insn | D | S | X | W |
| ldf X(r1),f1 | c1 | c2 | c3 | |
| mulf f0,f1,f2 | c2 | | | |
| stf f2,Z(r1) | c3 | | | |
| addi r1,4,r1 | | | | |
| ldf X(r1),f1 | | | | |
| mulf f0,f1,f2 | | | | |
| stf f2,Z(r1) | | | | |

| Map Table | |
|---|---|
| Reg | T |
| f0 | |
| f1 | RS#2 |
| f2 | RS#4 |
| r1 | |

| CDB | |
|---|---|
| T | V |
| | |

| Reservation Stations | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| T | FU | busy | op | R | T1 | T2 | V1 | V2 |
| 1 | ALU | no | | | | | | |
| 2 | LD | yes | ldf | f1 | - | - | - | [r1] |
| 3 | ST | yes | stf | - | RS#4 | - | - | [r1] |
| 4 | FP1 | yes | mulf | f2 | - | RS#2 | [f0] | - |
| 5 | FP2 | no | | | | | | |

allocate

# Tomasulo动态指令发射算法

- Tomasulo动态指令发射实例

**Tomasulo:**

**Cycle 4**

Insn Status

| Insn | D | S | X | W |
|---|---|---|---|---|
| ldf X(r1),f1 | c1 | c2 | c3 | c4 |
| mulf f0,f1,f2 | c2 | c4 | | |
| stf f2,Z(r1) | c3 | | | |
| addi r1,4,r1 | c4 | | | |
| ldf X(r1),f1 | | | | |
| mulf f0,f1,f2 | | | | |
| stf f2,Z(r1) | | | | |

Map Table

| Reg | T |
|---|---|
| f0 | |
| f1 | RS#2 |
| f2 | RS#4 |
| r1 | RS#1 |

CDB

| T | V |
|---|---|
| RS#2 | [f1] |

ldf finished (W)
clear f1 RegStatus
CDB broadcast

Reservation Stations

| T | FU | busy | op | R | T1 | T2 | V1 | V2 | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | ALU | yes | addi | r1 | - | - | [r1] | - | allocate |
| 2 | LD | no | | | | | | | free |
| 3 | ST | yes | stf | - | RS#4 | - | - | [r1] | |
| 4 | FP1 | yes | mulf | f2 | - | RS#2 | [f0] | CDB.V | RS#2 ready → grab CDB value |
| 5 | FP2 | no | | | | | | | |

# Tomasulo动态指令发射算法

- Tomasulo动态指令发射实例

**Tomasulo:**

**Cycle 5**

| Insn Status | | | | |
|---|---|---|---|---|
| Insn | D | S | X | W |
| ldf X(r1),f1 | c1 | c2 | c3 | c4 |
| mulf f0,f1,f2 | c2 | c4 | c5 | |
| stf f2,Z(r1) | c3 | | | |
| addi r1,4,r1 | c4 | c5 | | |
| ldf X(r1),f1 | c5 | | | |
| mulf f0,f1,f2 | | | | |
| stf f2,Z(r1) | | | | |

| Map Table | |
|---|---|
| Reg | T |
| f0 | |
| f1 | RS#2 |
| f2 | RS#4 |
| r1 | RS#1 |

| CDB | |
|---|---|
| T | V |
| | |

| Reservation Stations | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| T | FU | busy | op | R | T1 | T2 | V1 | V2 |
| 1 | ALU | yes | addi | r1 | - | - | [r1] | - |
| 2 | LD | yes | ldf | f1 | - | RS#1 | - | - |
| 3 | ST | yes | stf | - | RS#4 | - | - | [r1] |
| 4 | FP1 | yes | mulf | f2 | - | - | [f0] | [f1] |
| 5 | FP2 | no | | | | | | |

allocate

# Tomasulo动态指令发射算法

- Tomasulo动态指令发射实例

假设 multf 需要3个cycle完成

**Tomasulo:**

**Cycle 6**

| Insn Status | | | | |
|---|---|---|---|---|
| Insn | D | S | X | W |
| ldf X(r1),f1 | c1 | c2 | c3 | c4 |
| mulf f0,f1,f2 | c2 | c4 | c5+ | |
| stf f2,Z(r1) | c3 | | | |
| addi r1,4,r1 | c4 | c5 | c6 | |
| ldf X(r1),f1 | c5 | | | |
| mulf f0,f1,f2 | c6 | | | |
| stf f2,Z(r1) | | | | |

| Map Table | |
|---|---|
| Reg | T |
| f0 | |
| f1 | |
| f2 | RS#4RS#5 |
| r1 | RS#1 |

| CDB | |
|---|---|
| T | V |
| | |

no D stall on WAW: scoreboard would overwrite f2 RegStatus
anyone who needs old f2 tag has it

| Reservation Stations | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| T | FU | busy | op | R | T1 | T2 | V1 | V2 |
| 1 | ALU | yes | addi | r1 | - | - | [r1] | - |
| 2 | LD | yes | ldf | f1 | - | RS#1 | - | - |
| 3 | ST | yes | stf | - | RS#4 | - | - | [r1] |
| 4 | FP1 | yes | mulf | f2 | - | - | [f0] | [f1] |
| 5 | FP2 | yes | mulf | f2 | - | RS#2 | [f0] | - |

allocate

# Tomasulo动态指令发射算法

- Tomasulo动态指令发射实例

假设 multf 需要3个cycle完成

| Insn Status | | | | |
|---|---|---|---|---|
| Insn | D | S | X | W |
| ldf X(r1),f1 | c1 | c2 | c3 | c4 |
| mulf f0,f1,f2 | c2 | c4 | c5+ | |
| stf f2,Z(r1) | c3 | | | |
| addi r1,4,r1 | c4 | c5 | c6 | c7 |
| ldf X(r1),f1 | c5 | c7 | | |
| mulf f0,f1,f2 | c6 | | | |
| stf f2,Z(r1) | | | | |

| Map Table | |
|---|---|
| Reg | T |
| f0 | |
| f1 | RS#2 |
| f2 | RS#5 |
| r1 | RS#1 |

| CDB | |
|---|---|
| T | V |
| RS#1 | [r1] |

**Tomasulo:**

**Cycle 7**

| Reservation Stations | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| T | FU | busy | op | R | T1 | T2 | V1 | V2 |
| 1 | ALU | no | | | | | | |
| 2 | LD | yes | ldf | f1 | - | RS#1 | - | CDB.V |
| 3 | ST | yes | stf | - | RS#4 | - | - | [r1] |
| 4 | FP1 | yes | mulf | f2 | - | - | [f0] | [f1] |
| 5 | FP2 | yes | mulf | f2 | - | RS#2 | [f0] | - |

no W wait on WAR: scoreboard would anyone who needs old r1 has RS copy

D stall on store RS: structural

addi finished (W)
clear r1 RegStatus
CDB broadcast

RS#1 ready → grab CDB value

# Tomasulo动态指令发射算法

- Tomasulo动态指令发射实例

假设 multf 需要3个cycle完成

**Tomasulo:**

**Cycle 8**

| Insn Status | | | | |
|---|---|---|---|---|
| Insn | D | S | X | W |
| ldf X(r1),f1 | c1 | c2 | c3 | c4 |
| mulf f0,f1,f2 | c2 | c4 | c5+ | c8 |
| stf f2,Z(r1) | c3 | c8 | | |
| addi r1,4,r1 | c4 | c5 | c6 | c7 |
| ldf X(r1),f1 | c5 | c7 | c8 | |
| mulf f0,f1,f2 | c6 | | | |
| stf f2,Z(r1) | | | | |

| Map Table | |
|---|---|
| Reg | T |
| f0 | |
| f1 | RS#2 |
| f2 | RS#5 |
| r1 | |

| CDB | |
|---|---|
| T | V |
| RS#4 | [f2] |

mulf finished (W)
don't clear f2 RegStatus
already overwritten by 2nd mulf (RS#5)
CDB broadcast

| Reservation Stations | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| T | FU | busy | op | R | T1 | T2 | V1 | V2 |
| 1 | ALU | no | | | | | | |
| 2 | LD | yes | ldf | f1 | - | - | - | [r1] |
| 3 | ST | yes | stf | - | RS#4 | - | CDB.V | [r1] |
| 4 | FP1 | no | | | | | | |
| 5 | FP2 | yes | mulf | f2 | - | RS#2 | [f0] | - |

RS#4 ready → grab CDB value

# Tomasulo动态指令发射算法

- Tomasulo动态指令发射实例

**假设 multf 需要3个cycle完成**

**Tomasulo:**

**Cycle 9**

| Insn Status | | | | |
|---|---|---|---|---|
| Insn | D | S | X | W |
| ldf X(r1),f1 | c1 | c2 | c3 | c4 |
| mulf f0,f1,f2 | c2 | c4 | c5+ | c8 |
| stf f2,Z(r1) | c3 | c8 | c9 | |
| addi r1,4,r1 | c4 | c5 | c6 | c7 |
| ldf X(r1),f1 | c5 | c7 | c8 | c9 |
| mulf f0,f1,f2 | c6 | c9 | | |
| stf f2,Z(r1) | | | | |

| Map Table | |
|---|---|
| Reg | T |
| f0 | |
| f1 | RS#2 |
| f2 | RS#5 |
| r1 | |

| CDB | |
|---|---|
| T | V |
| RS#2 | [f1] |

**2nd ldf finished (W)**
**clear f1 RegStatus**
**CDB broadcast**

| Reservation Stations | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| T | FU | busy | op | R | T1 | T2 | V1 | V2 |
| 1 | ALU | no | | | | | | |
| 2 | LD | no | | | | | | |
| 3 | ST | yes | stf | - | - | - | [f2] | [r1] |
| 4 | FP1 | no | | | | | | |
| 5 | FP2 | yes | mulf | f2 | - | RS#2 | [f0] | CDB.V |

**RS#2 ready → grab CDB value**

# Tomasulo动态指令发射算法

- Tomasulo动态指令发射实例

假设 multf 需要3个cycle完成

**Tomasulo:**

**Cycle 10**

| Insn Status | | | | |
|---|---|---|---|---|
| Insn | D | S | X | W |
| ldf X(r1),f1 | c1 | c2 | c3 | c4 |
| mulf f0,f1,f2 | c2 | c4 | c5+ | c8 |
| stf f2,Z(r1) | c3 | c8 | c9 | c10 |
| addi r1,4,r1 | c4 | c5 | c6 | c7 |
| ldf X(r1),f1 | c5 | c7 | c8 | c9 |
| mulf f0,f1,f2 | c6 | c9 | c10 | |
| stf f2,Z(r1) | c10 | | | |

| Map Table | |
|---|---|
| Reg | T |
| f0 | |
| f1 | |
| f2 | RS#5 |
| r1 | |

| CDB | |
|---|---|
| T | V |
| | |

stf finished (W)
no output register → no CDB broadcast

| Reservation Stations | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| T | FU | busy | op | R | T1 | T2 | V1 | V2 |
| 1 | ALU | no | | | | | | |
| 2 | LD | no | | | | | | |
| 3 | ST | yes | stf | - | RS#5 | - | - | [r1] |
| 4 | FP1 | no | | | | | | |
| 5 | FP2 | yes | mulf | f2 | - | - | [f0] | [f1] |

free → allocate

# 超标量+动态指令发射

北京大学 PEKING UNIVERSITY

- **Tomasulo动态指令发射实例**

  - Dynamic scheduling and multiple issue are orthogonal
    - E.g., Pentium4: dynamically scheduled 5-way superscalar
    - Two dimensions
      - **N**: superscalar width (number of parallel operations)
      - **W**: window size (number of reservation stations)

  - What do we need for an **N**-by-**W** Tomasulo?
    - RS: **N** tag/value w-ports (D), **N** value r-ports (S), **2N** tag CAMs (W)
    - Select logic: **W**→**N** priority encoder (S)
    - MT: **2N** r-ports (D), **N** w-ports (D)
    - RF: **2N** r-ports (D), **N** w-ports (W)
    - CDB: **N** (W)
    - Which are the expensive pieces?

# 超标量+动态指令发射

- Tomasulo动态指令发射实例

  - Superscalar select logic: W→N priority encoder
    - – Somewhat complicated ($N^2$ logW)
    - Can simplify using different RS designs
  - **Split design**
    - Divide RS into N banks: 1 per FU?
    - Implement N separate W/N→1 encoders
    - \+ Simpler: N * logW/N
    - – Less scheduling flexibility
  - **FIFO design**
    - Can issue only head of each RS bank
    - \+ Simpler: no select logic at all
    - – Less scheduling flexibility (but surprisingly not that bad)

# 目 录

CONTENTS

# 分支处理机制与地址预测

北京大学 PEKING UNIVERSITY

- **Tomasulo动态发射的潜在问题**

  - When can Tomasulo go wrong?
    - Branches
      - What if a branch finishes after younger instructions (after the branch) finish?
    - Exceptions!!
      - No way to figure out relative order of instructions in RS
      - **We need a mechanism to predict branch results**
      - **We need a mechanism to ensure finish in order**

# 分支预测



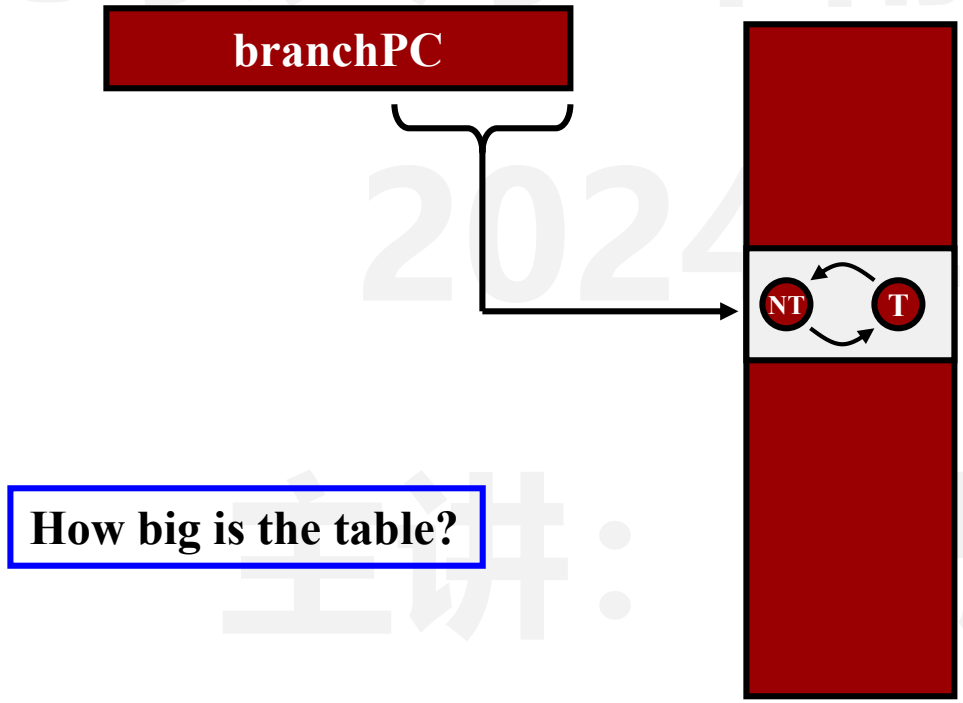
- 包括方向预测、地址预测与恢复机制

- Direction Predictor
  - For conditional branches
    - **Predicts whether the branch will be taken**
  - Examples:
    - Always taken; backwards taken
- Address Predictor
  - Predicts the target address (use if predicted taken)
  - Examples:
    - **BTB;  Return Address Stack; Precomputed Branch**
- Recovery logic

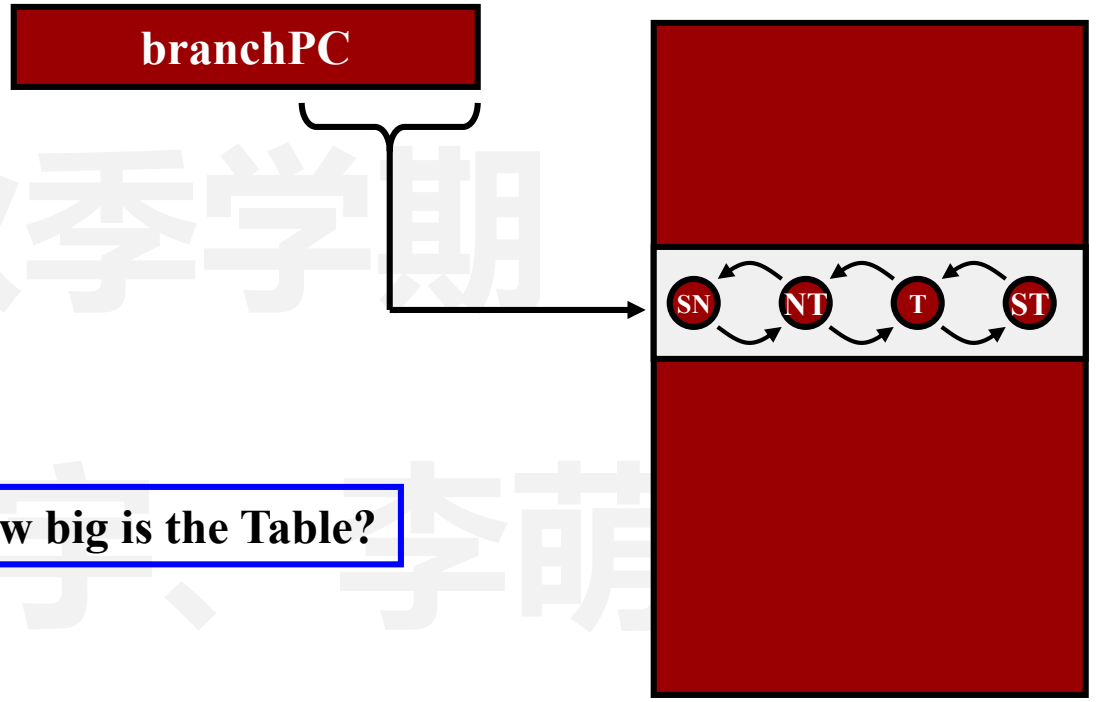
# 分支预测

- 方向预测 – 基于历史的简单状态机FSM

- 1-bit history (direction predictor)
  - Remember the last direction for a branch




- 2-bit history (direction predictor)



# 分支预测



- 方向预测 – 基于历史的简单状态机FSM

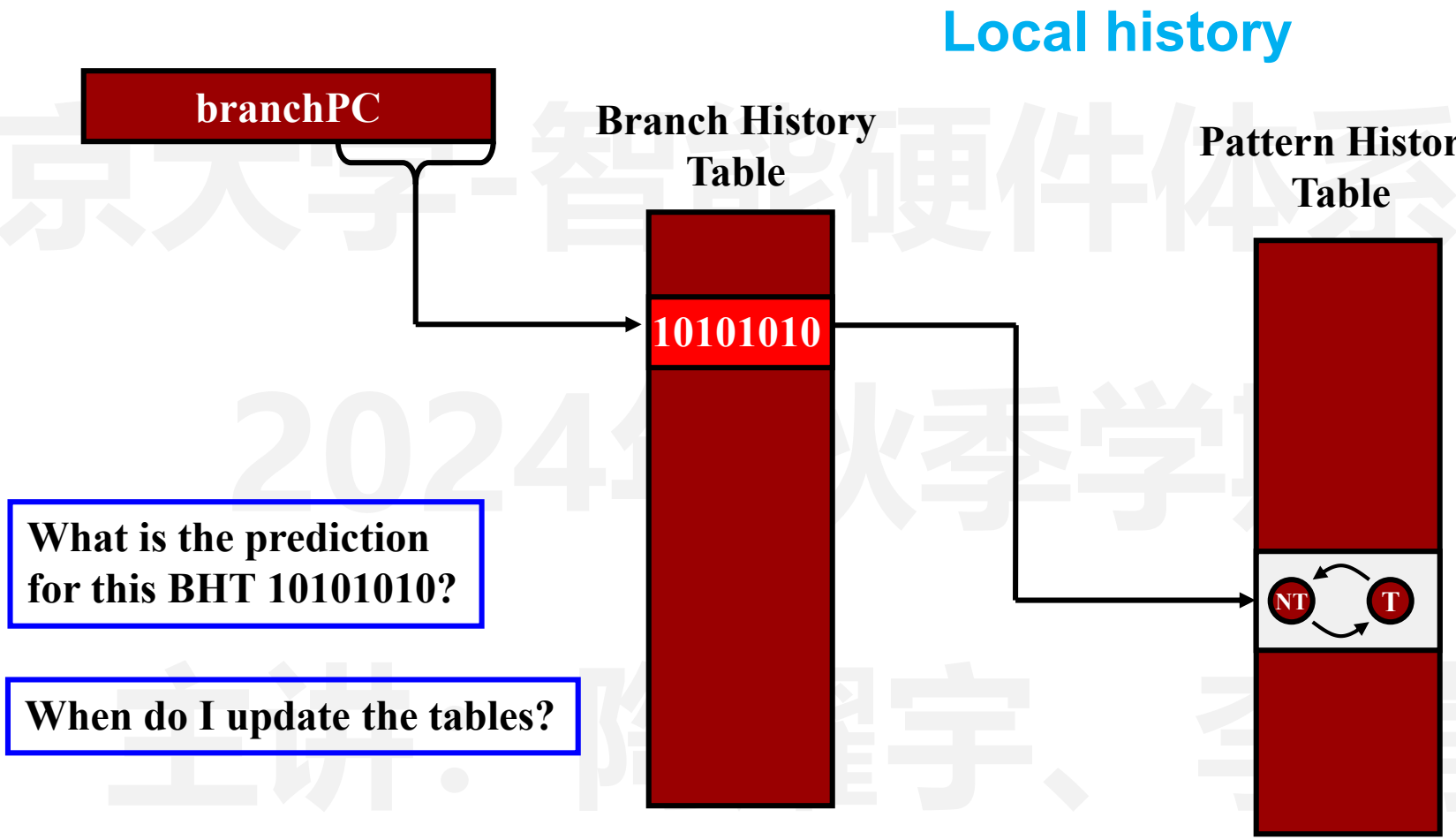
  - **~80 percent of branches are either heavily TAKEN or heavily NOT-TAKEN**
  - For the other 20%, we need to look a patterns of reference to see if they are predictable using a more complex predictor
  - Example: gcc has a branch that flips each time

T(1) NT(0) 10101010101010101010101010101010

**Using History Patterns**

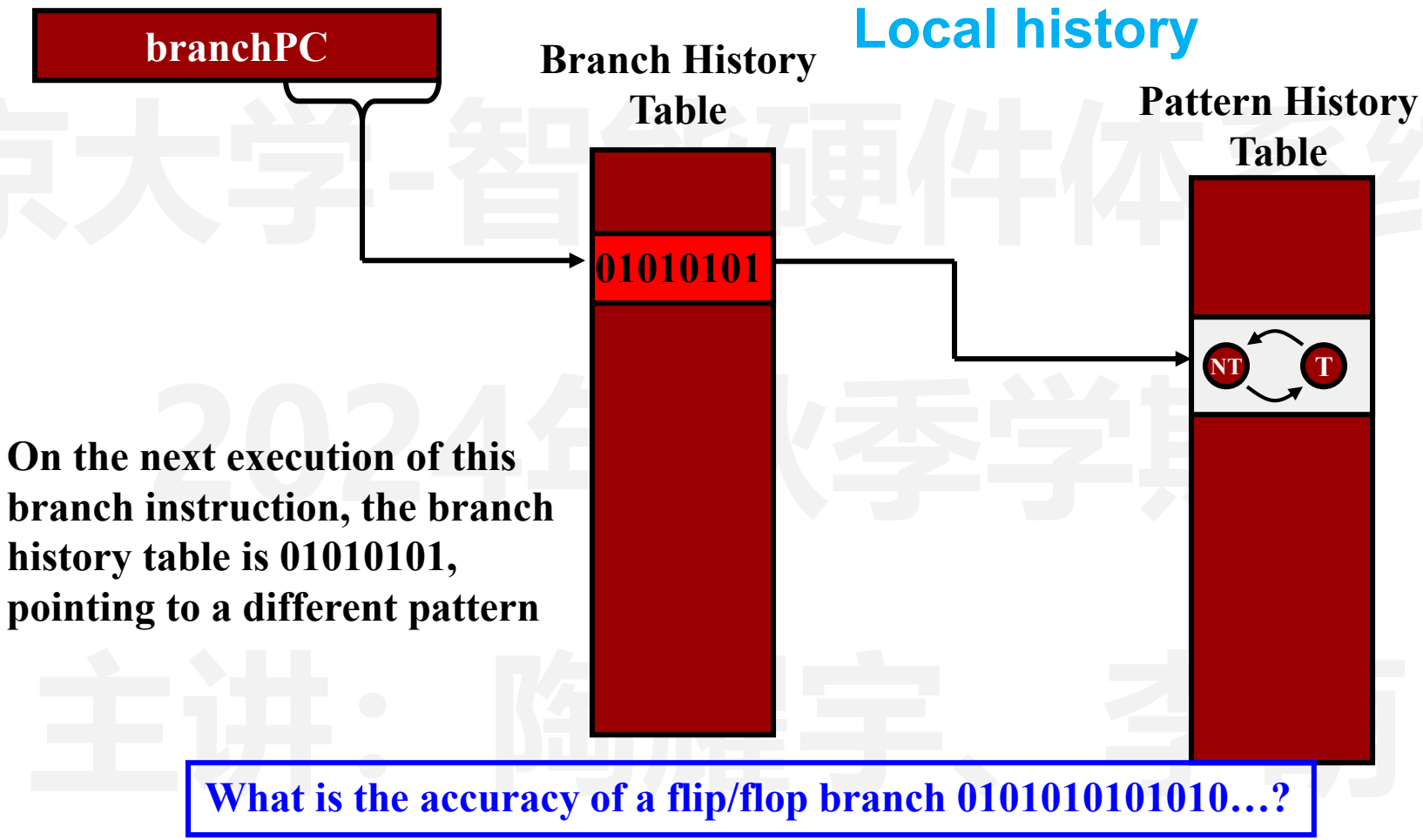
# 分支预测

- 方向预测 – 基于历史的简单状态机FSM



# 分支预测

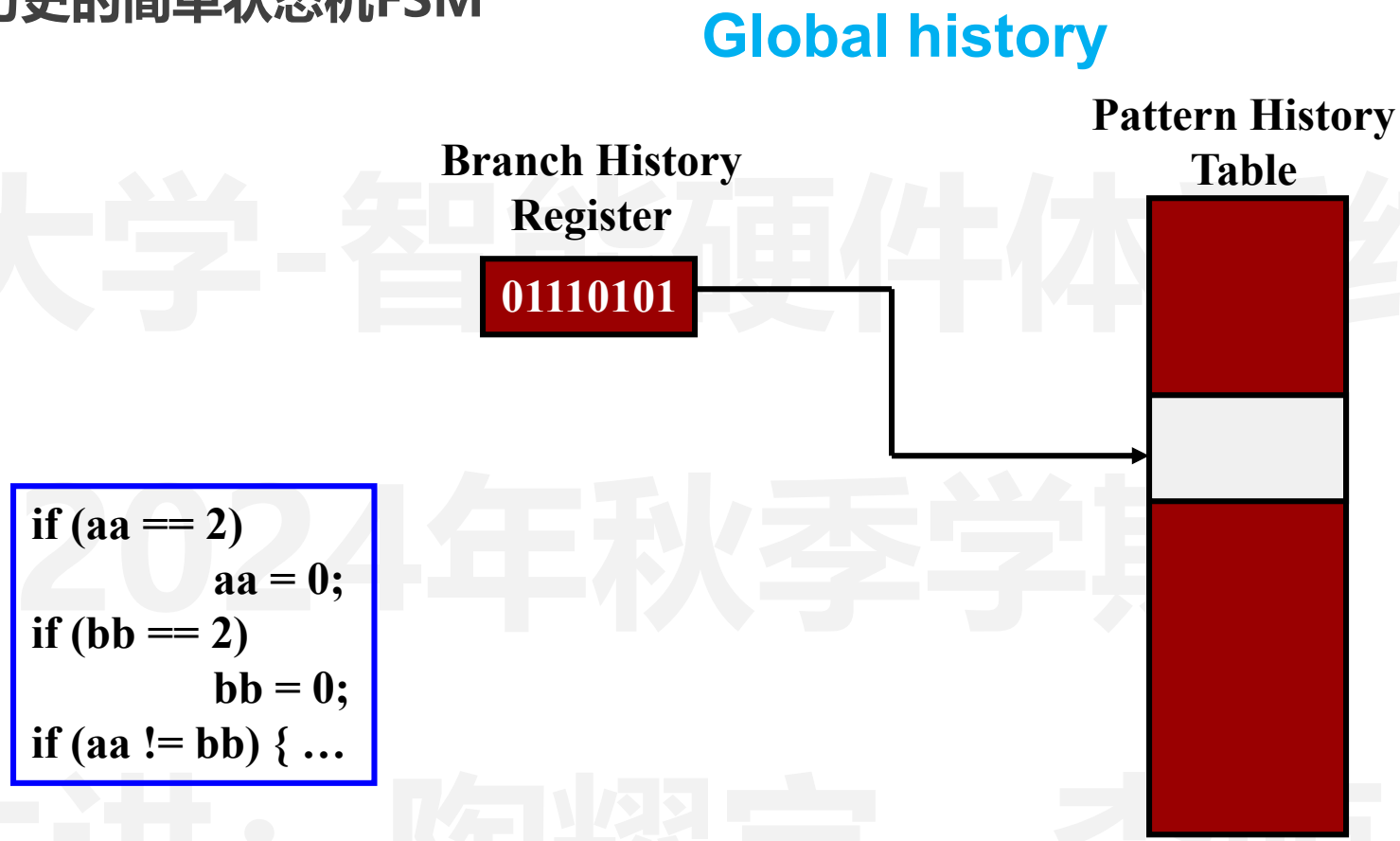
- 方向预测 – 基于历史的简单状态机FSM




**Using History Patterns**

# 分支预测

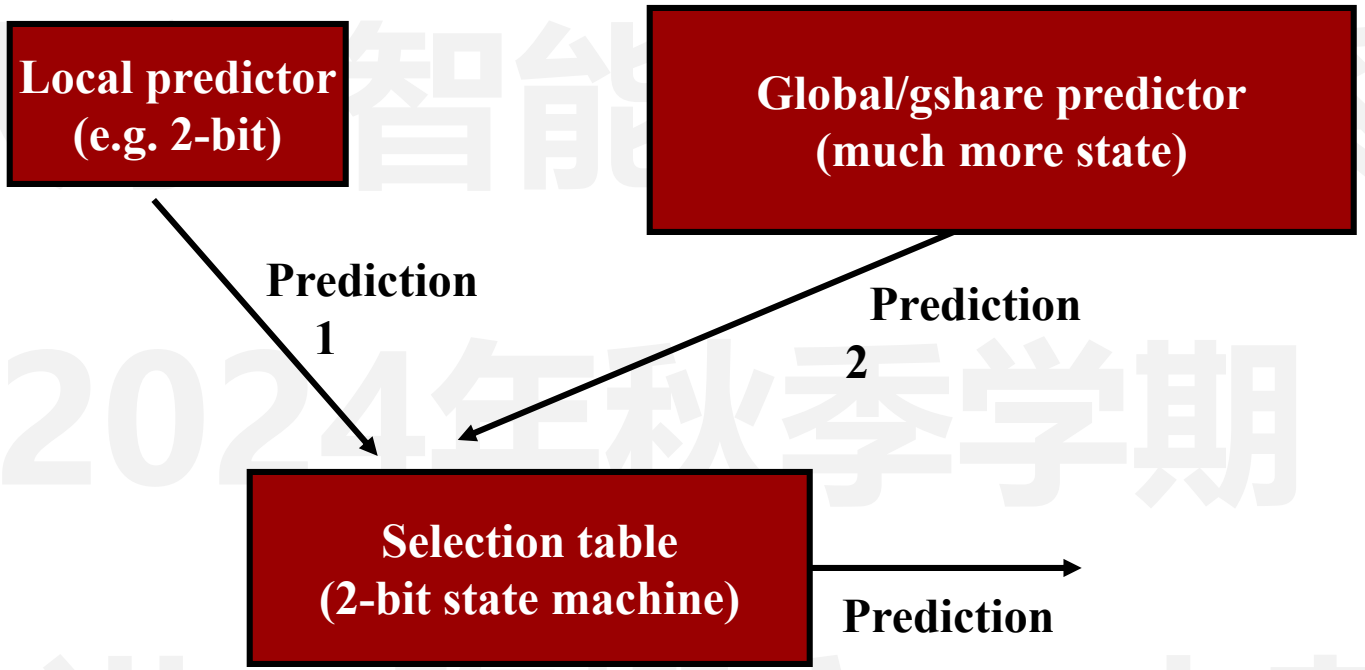
- 方向预测 – 基于历史的简单状态机FSM




**Using History Patterns**

- 方向预测 – 基于历史的简单状态机FSM

## Hybrid predictors




How do you select which predictor to use?
How do you update the various predictor/selector?

## Using History Patterns

# 分支预测

北京大学 PEKING UNIVERSITY

- 地址预测 – Branch Target Buffer

- BTB indexed by current PC
  - If entry is in BTB fetch target address next
- Generally set associative (too slow as FA)
- Often qualified by branch taken predictor

| Branch PC | Target address |
|---|---|
| 0x05360AF0 | 0x05360000 |
| … | … |
| … | … |
| … | … |
| … | … |
| … | … |

# 分支流水线处理机制

- 对于顺序多级流水线比较简单，对乱序执行需要额外的机制

## 顺序多级流水线

- Squash and restart fetch with right address
  - Just have to be sure that nothing has "committed" its state yet.
- In our 5-stage pipe, state is only committed during MEM (for stores) and WB (for registers)

## Tamosulo

- Recovery seems really hard
  - What if instructions after the branch finish before we find that the branch was wrong?
    - This could happen. Imagine

      R1=MEM[R2+0]

      BEQ R1, R3 DONE ← Predicted not taken

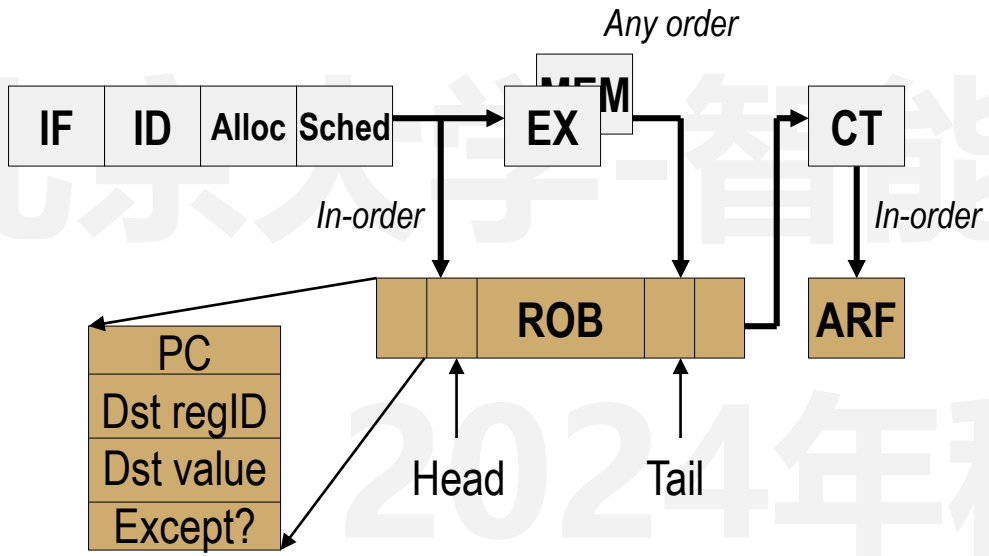
      R4=R5+R6
  - So we have to not speculate on branches or not let anything pass a branch
    - Which is really the same thing.
    - Branches become serializing instructions.
      - Note that can be executing some things before and after the branch once branch resolves.

# MIPS R10K：超标量+动态指令发射

- **Adding a Reorder Buffer, aka ROB**
  - Why need Reorder Buffer
    - ROB is an *in-order* queue where instructions are placed.
    - **Instructions *complete* (retire) in-order**
    - **Instructions still *execute* out-of-order**
    - Still use RS
      - **Instructions are issued to RS and ROB at the same time**
      - Rename is to ROB entry, not RS.
      - When *execute* done instruction leaves RS
    - Only when all instructions in before it in program order are done does the instruction retire.

# MIPS R10K：超标量+动态指令发射

- Adding a Reorder Buffer, aka ROB

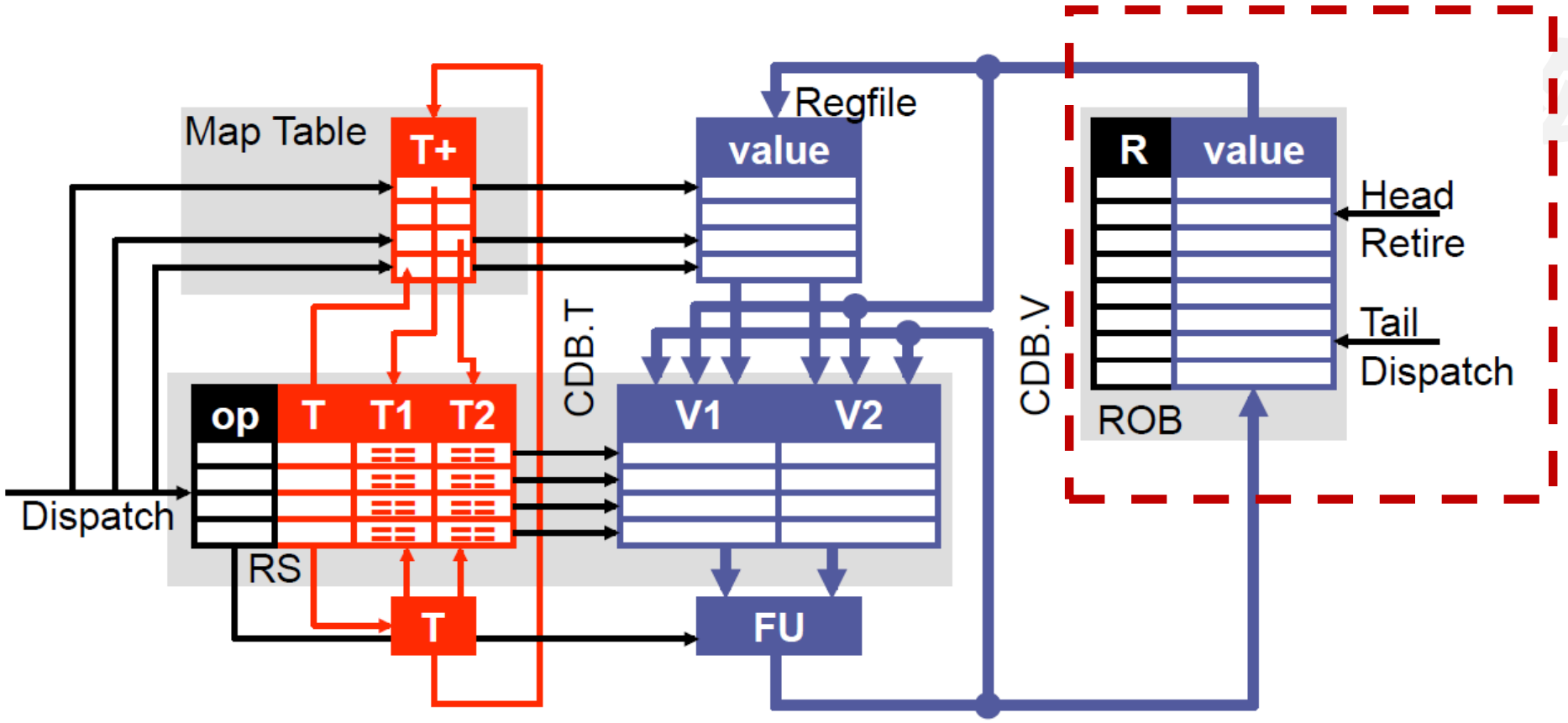



- Reorder Buffer (ROB)
  - Circular queue of spec state
  - May contain multiple definitions of *same* register

- @ **Alloc**
  - Allocate result storage at Tail
- @ **Sched**
  - Get inputs (ROB T-to-H then ARF)
  - Wait until all inputs ready
- @ **WB**
  - Write results/fault to ROB
  - Indicate result is ready
- @ **CT**
  - Wait until inst @ Head is done
  - If fault, initiate handler
  - Else, write results to ARF
  - Deallocate entry from ROB

# MIPS R10K：超标量+动态指令发射

- Adding a Reorder Buffer, aka ROB



# 目 录

CONTENTS

# MIPS R10K：超标量+动态指令发射

- 下一次课

北京大学-智能硬件体系结构

.....

2024年秋季学期

主讲：陶耀宇、李萌