# Superscalar Design

## Register Data Flow

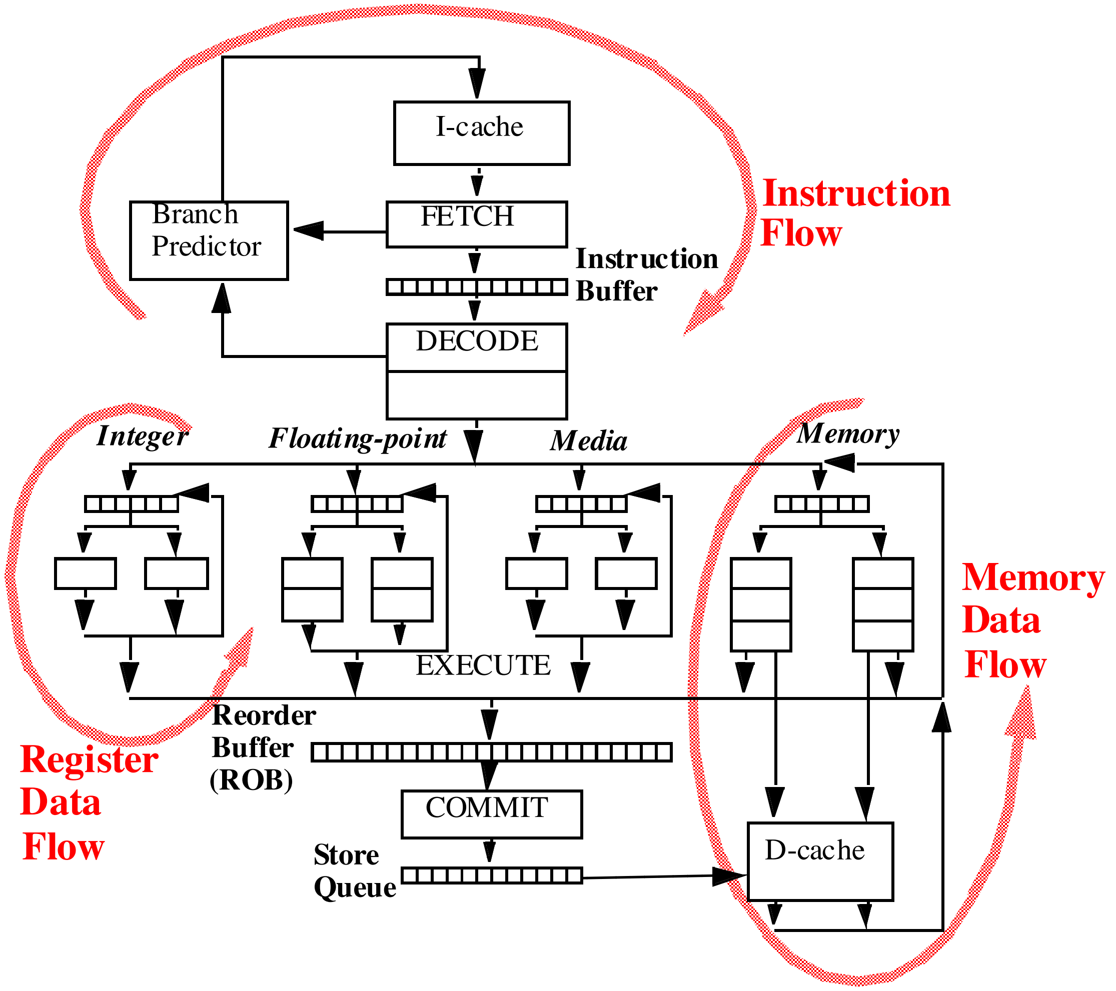
Virendra Singh
Associate Professor
Computer Architecture and Dependable Systems Lab
Department of Electrical Engineering
Indian Institute of Technology Bombay
http://www.ee.iitb.ac.in/~viren/
E-mail: viren@ee.iitb.ac.in

*EE-739: Processor Design*

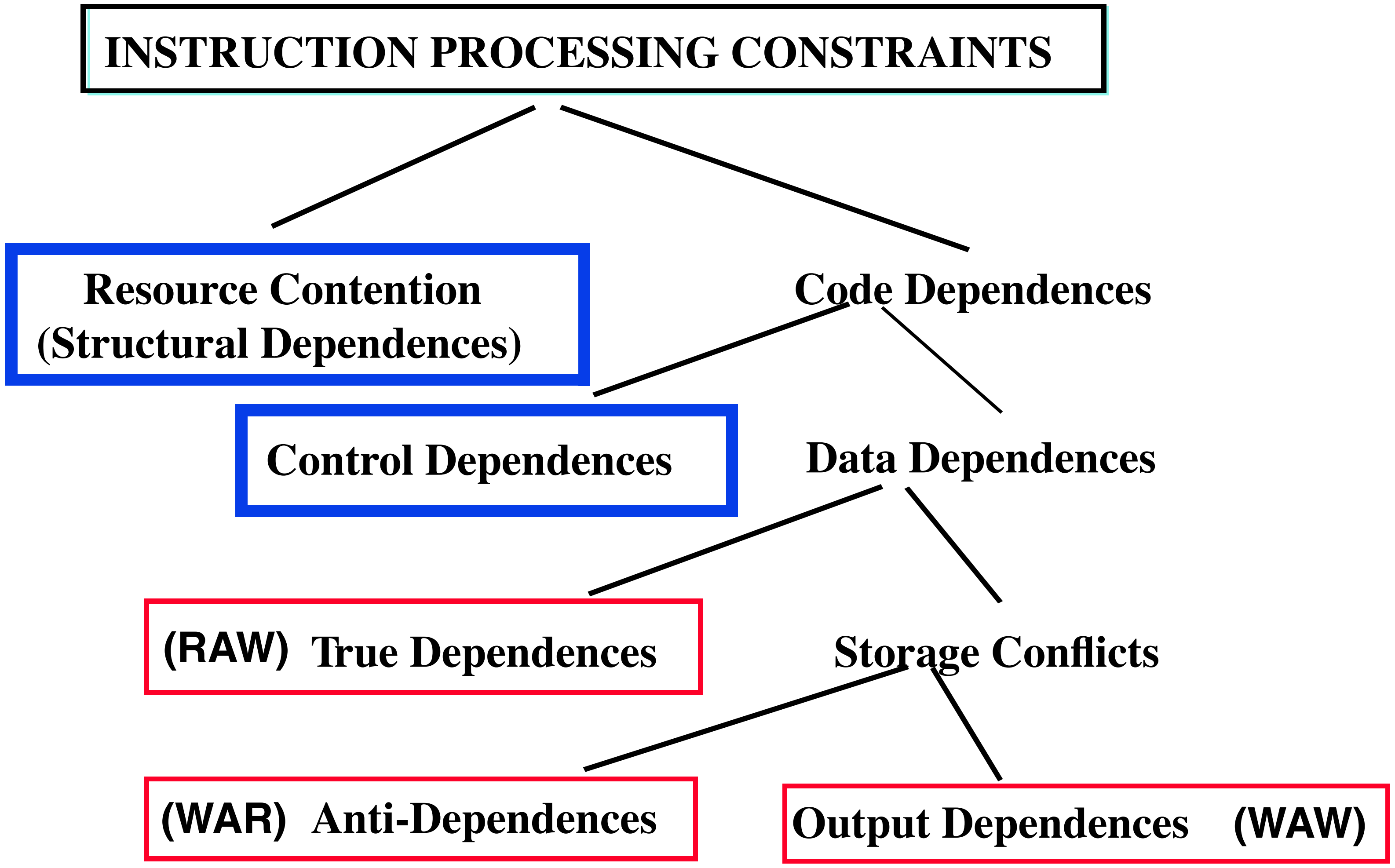
Lecture 6 (29 Jan 2015)

CADSL

# Impediments to High IPC



# The Big Picture



# Register Data Flow

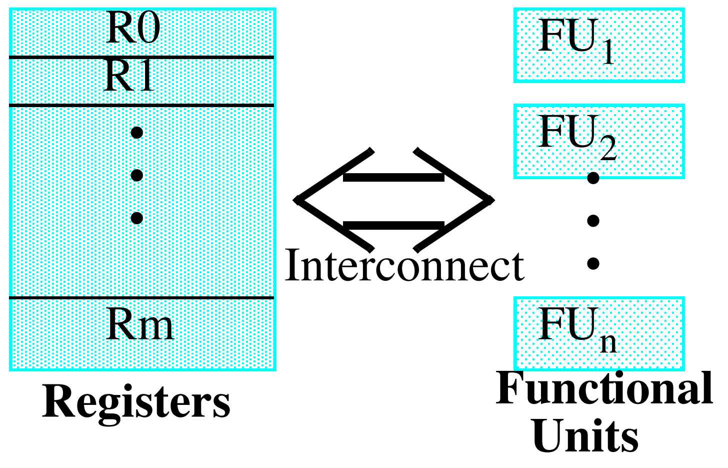
Each ALU Instruction:

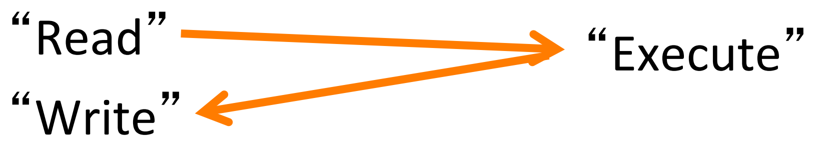
$$R_i \leftarrow F_n \; (R_j, R_k)$$

$$\underbrace{R_i}_{\text{Dest. Reg.}} \quad \underbrace{F_n}_{\text{Funct. Unit}} \quad \underbrace{(R_j, R_k)}_{\text{Source Registers}}$$

"Register Transfer"

INSTRUCTION EXECUTION MODEL





**Need Availability of Fn (Structural Dependences)**

**Need Availability of Rj, Rk (True Data Dependences)**

**Need Availability of Ri (Anti-and output Dependences)**

## Causes of (Register) Storage Conflict

**REGISTER RECYCLING**

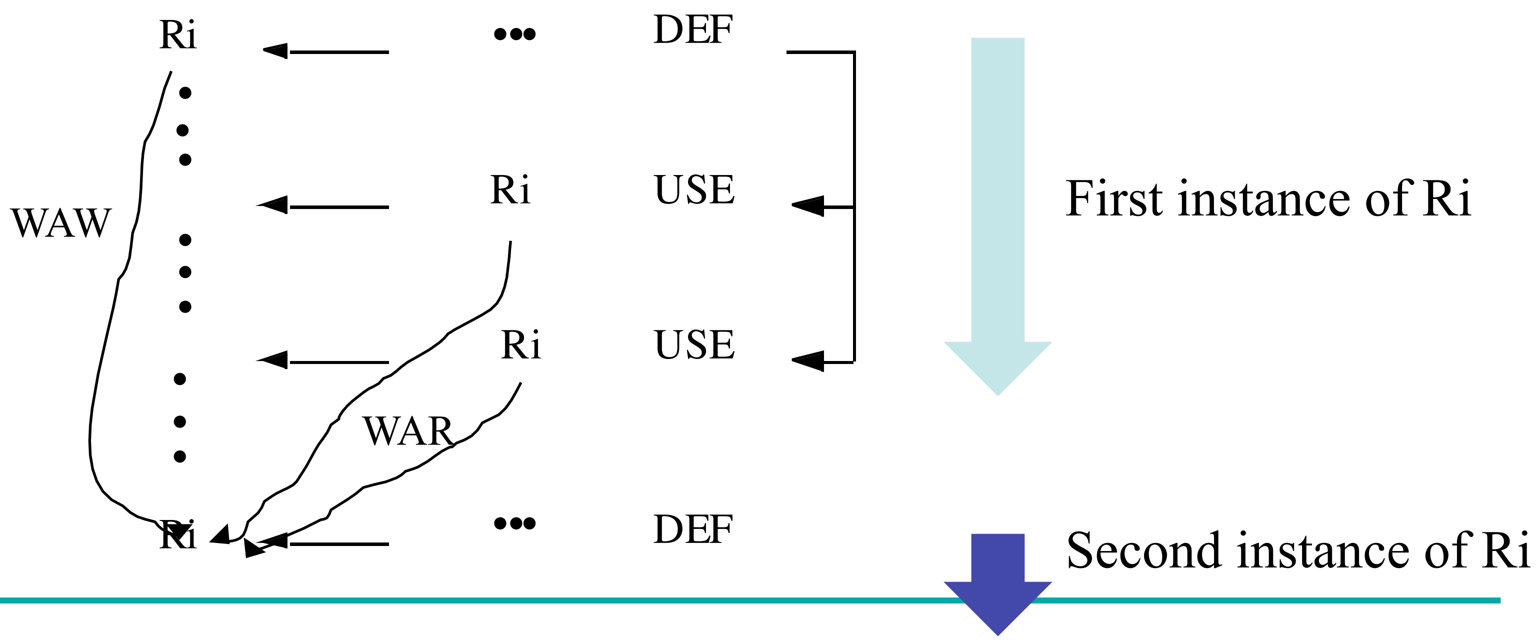
MAXIMIZE USE OF REGISTERS

MULTIPLE ASSIGNMENTS OF VALUES TO REGISTERS

**OUT OF ORDER ISSUING AND COMPLETION**

LOSE IMPLIED PRECEDENCE OF SEQUENTIAL CODE

LOSE 1-1 CORRESPONDENCE BETWEEN VALUES AND REGISTERS



# Contribution to Register Recycling

“Spill code” (if not enough registers)

**COMPILER REGISTER ALLOCATION**

$$\begin{tikzcd}
{} \arrow[d] \\
\boxed{\text{CODE GENERATION}} \arrow[d] \\
\boxed{\text{REG. ALLOCATION}} \arrow[d] \\
\text{INSTRUCTION LOOPS}
\end{tikzcd}$$

Single Assignment, Symbolic Reg.

Map Symbolic Reg. to Physical Reg.
Maximize Reuse of Reg.

**INSTRUCTION LOOPS**

```
9   $34:  mul   $14,  $7,   40
10        addu  $15,  $4,   $14
11        mul   $24,  $9,   4
12        addu  $25,  $15,  $24
13        lw    $11,  0($25)
14        mul   $12,  $9,   40
15        addu  $13,  $5,   $12
16        mul   $14,  $8,   4
17        addu  $15,  $13,  $14
18        lw    $24,  0($15)
19        mul   $25,  $11,  $24
20        addu  $10,  $10,  $25
21        addu  $9,   $9,   1
22        ble   $9,   10,   $34
```

For (k=1;k<= 10; k++)
t += a [i] [k] * b [k] [j] ;

Reuse Same Set of Reg. in
Each Iteration

Overlapped Execution of
Different Iterations

# Resolving Anti-Dependences

•
•
•

(1) R4 ◄– R3 + 1

(2) R3 ◄– R5 + 1

Must Prevent (2) from <u>completing</u> before (1) is <u>dispatched</u>.

**STALL DISPATCHING**

DELAY DISPATCHING OF (2)

REQUIRE RECHECKING AND REACCESSING

**COPY OPERAND**

WAR only

COPY NOT-YET-USED OPERAND TO PREVENT BEING OVERWRITTEN

MUST USE TAG IF ACTUAL OPERAND NOT-YET-AVAILABLE

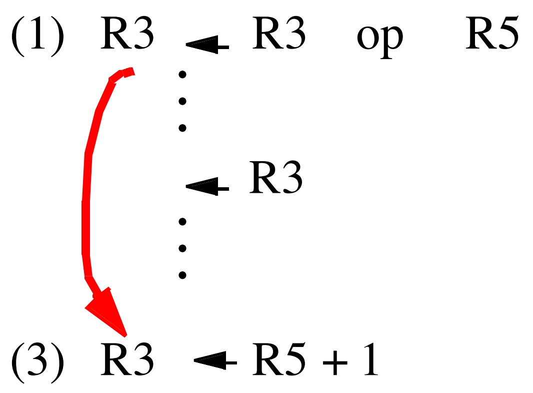
**RENAME REGISTER**

WAR and WAW

HARDWARE ALLOCATION

| | |
|---|---|
| R3 | <= … |
| | <= R3 |
| R3’ | <= … |
| | <= R3’ |

# Resolving Output Dependences



```
(1)  R3  ← R3  op  R5
          :
          ← R3
          :
(3)  R3  ← R5 + 1
```

Must Prevent (3) from completing before (1) completes.

**STALL DISPATCHING/ISSUING**

DENOTE OUTPUT DEPENDENCE

HOLD DISPATCHING UNTIL RESOLUTION OF DEPENDENCE

ALLOW DECODING OF SUBSEQUENT INSTRUCTIONS

**RENAME REGISTER**

HARDWARE ALLOCATION

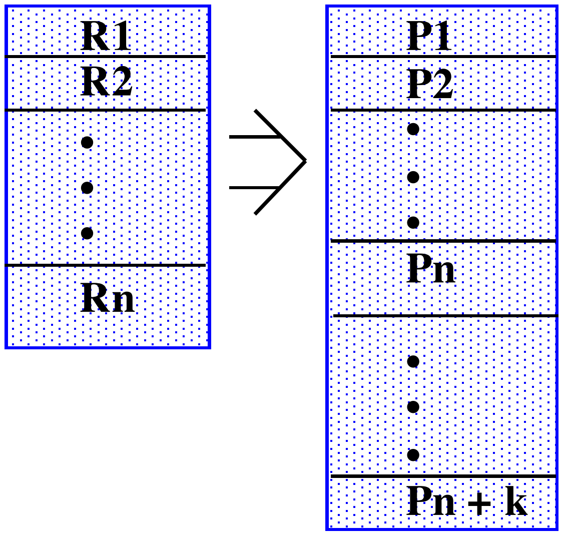
| | |
|---|---|
| R3 | <= … |
| | <= R3 |
| R3’ | <= … |
| | <= R3’ |

# Register Renaming

**Register Renaming Resolves:**

**Anti-Dependences**
**Output Dependences**

**Architected Registers** **Physical Registers**



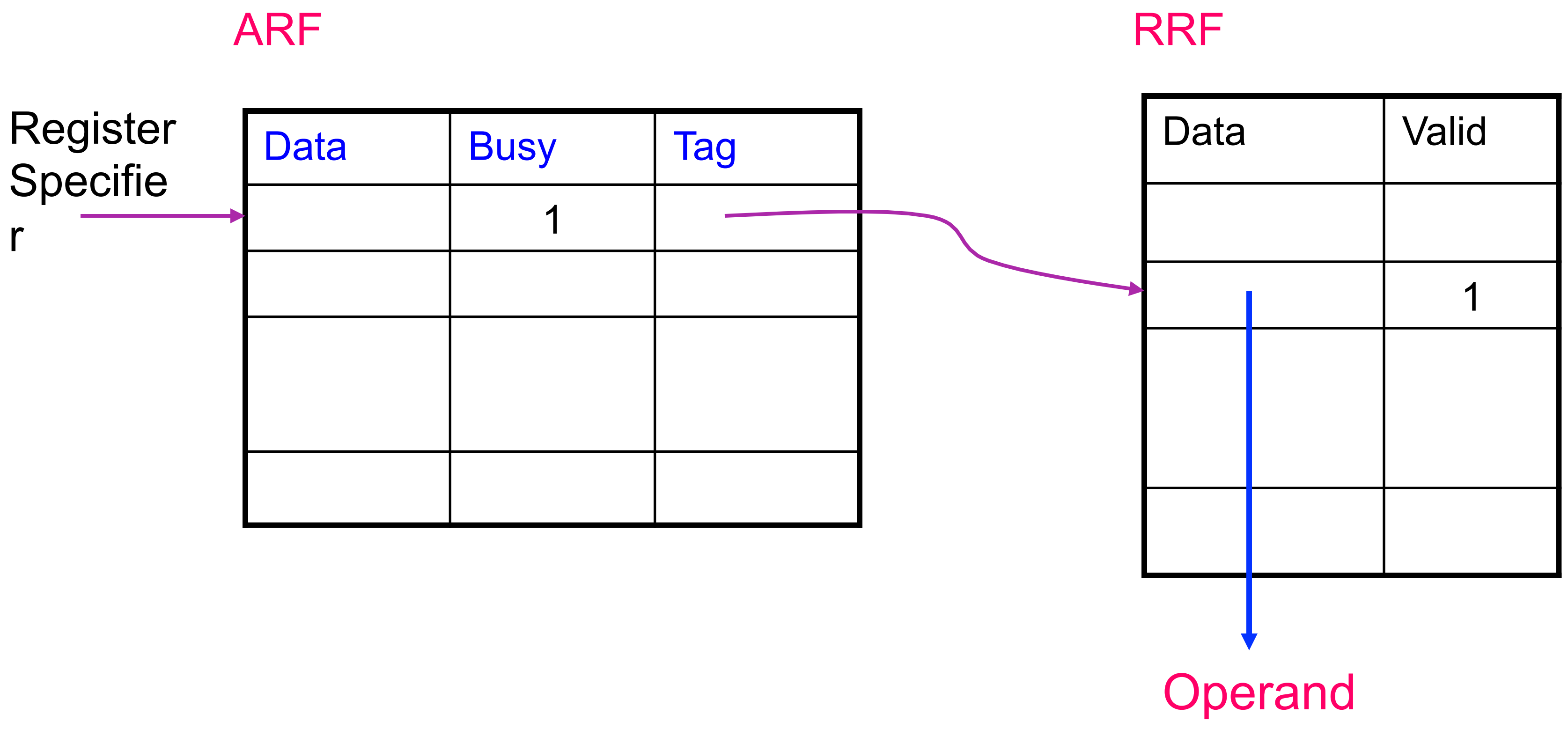
**Design of Redundant Registers**

**Number:**
- **One**
- **Multiple**

**Allocation:**
- **Fixed for Each Register**
- **Pooled for all Regsiters**

**Location:**
- **Attached to Register File (Centralized)**
- **Attached to functional units (Distributed)**

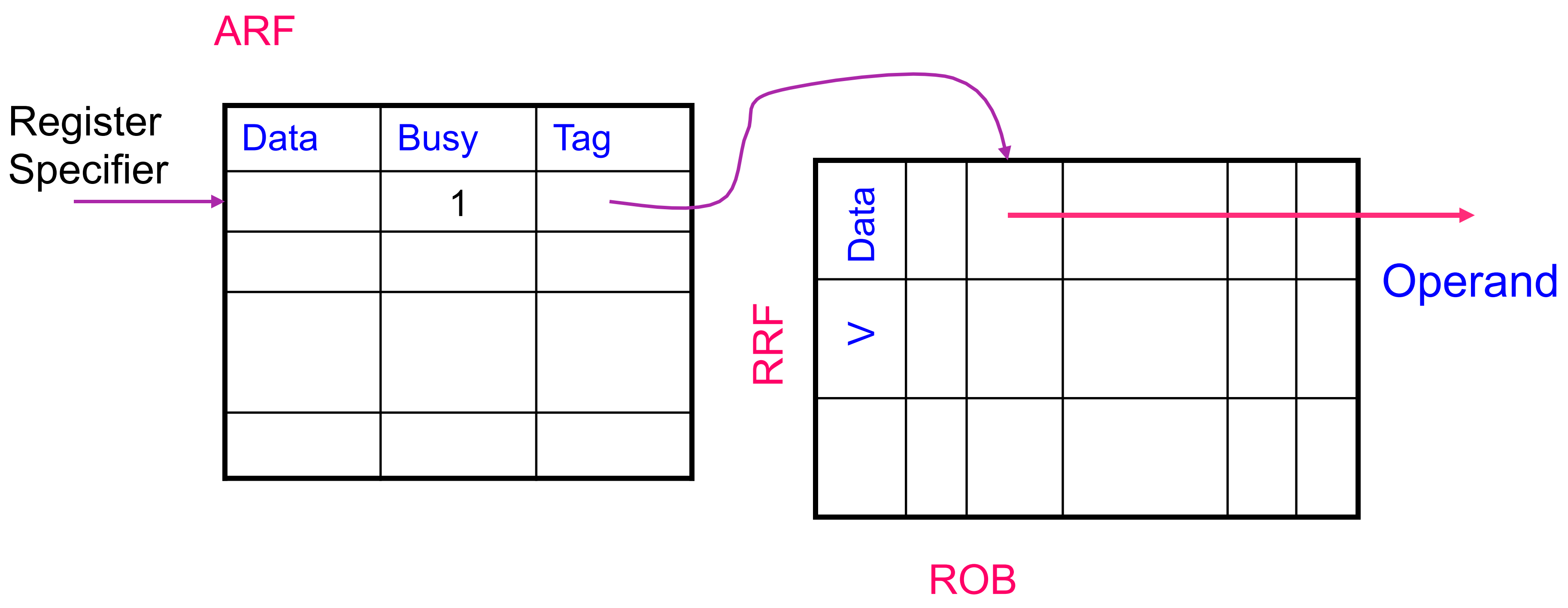
# Register Renaming



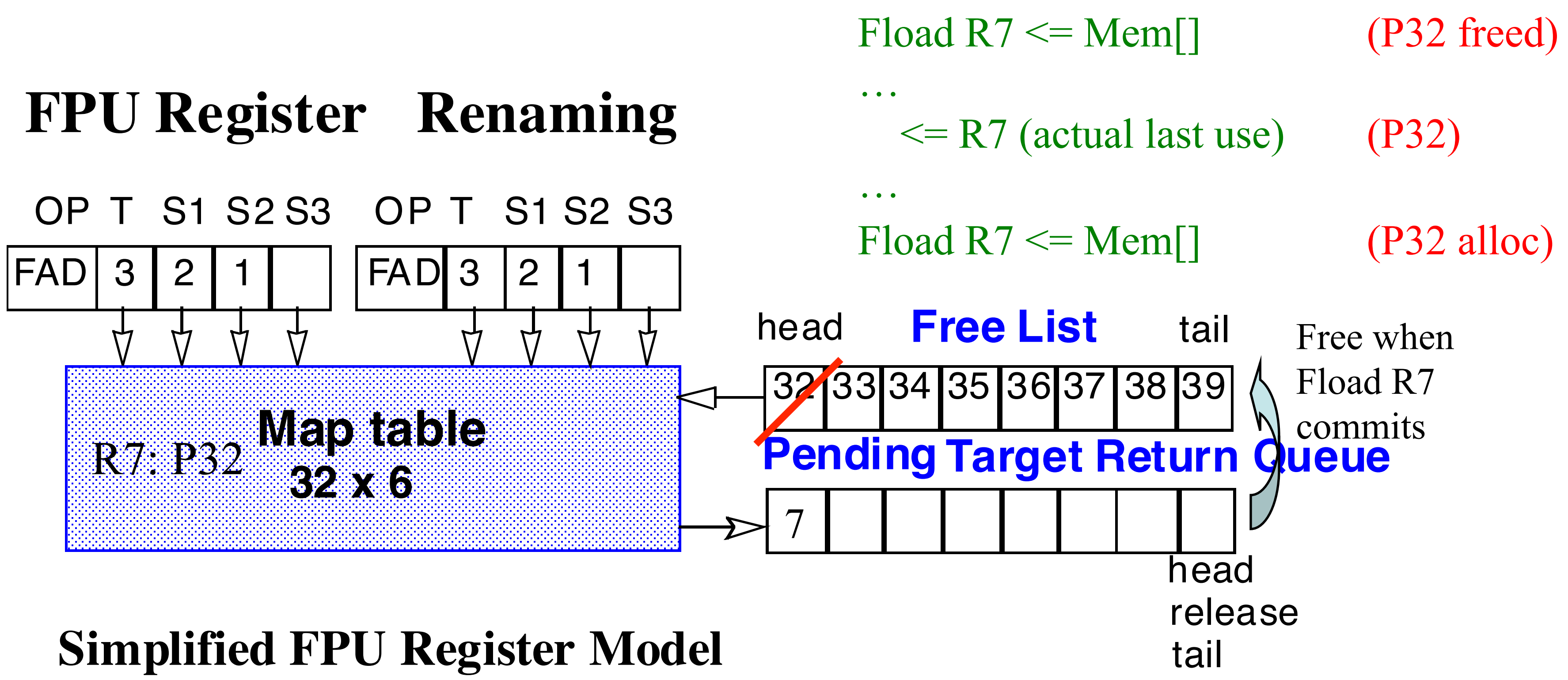
# Register Data Flow Techniques

- Register Reuse and False dependencies
- Register Renaming
- True data dependencies and data flow limits
- Tomasulo’s algorithm

# Register Renaming



# Register Renaming in the RIOS-I FPU



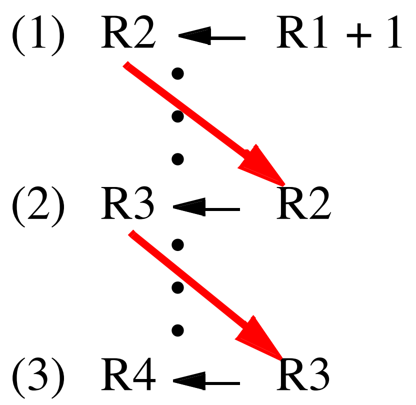
**Simplified FPU Register Model**

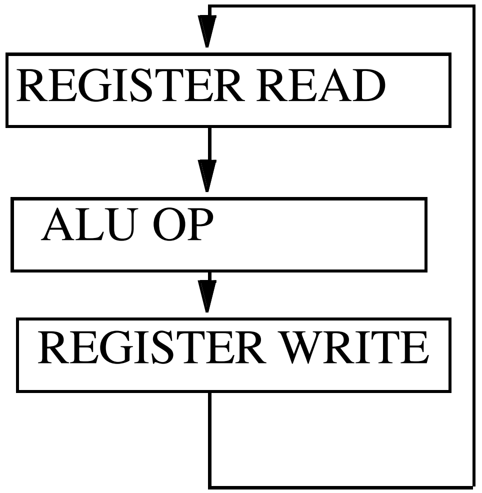
Incoming FPU instructions pass through a renaming table prior to decode

The 32 architectural registers are remapped to 40 physical registers

- Physical register names are used within the FPU
- Complex control logic maintains active register mapping

# Resolving True Data Dependences



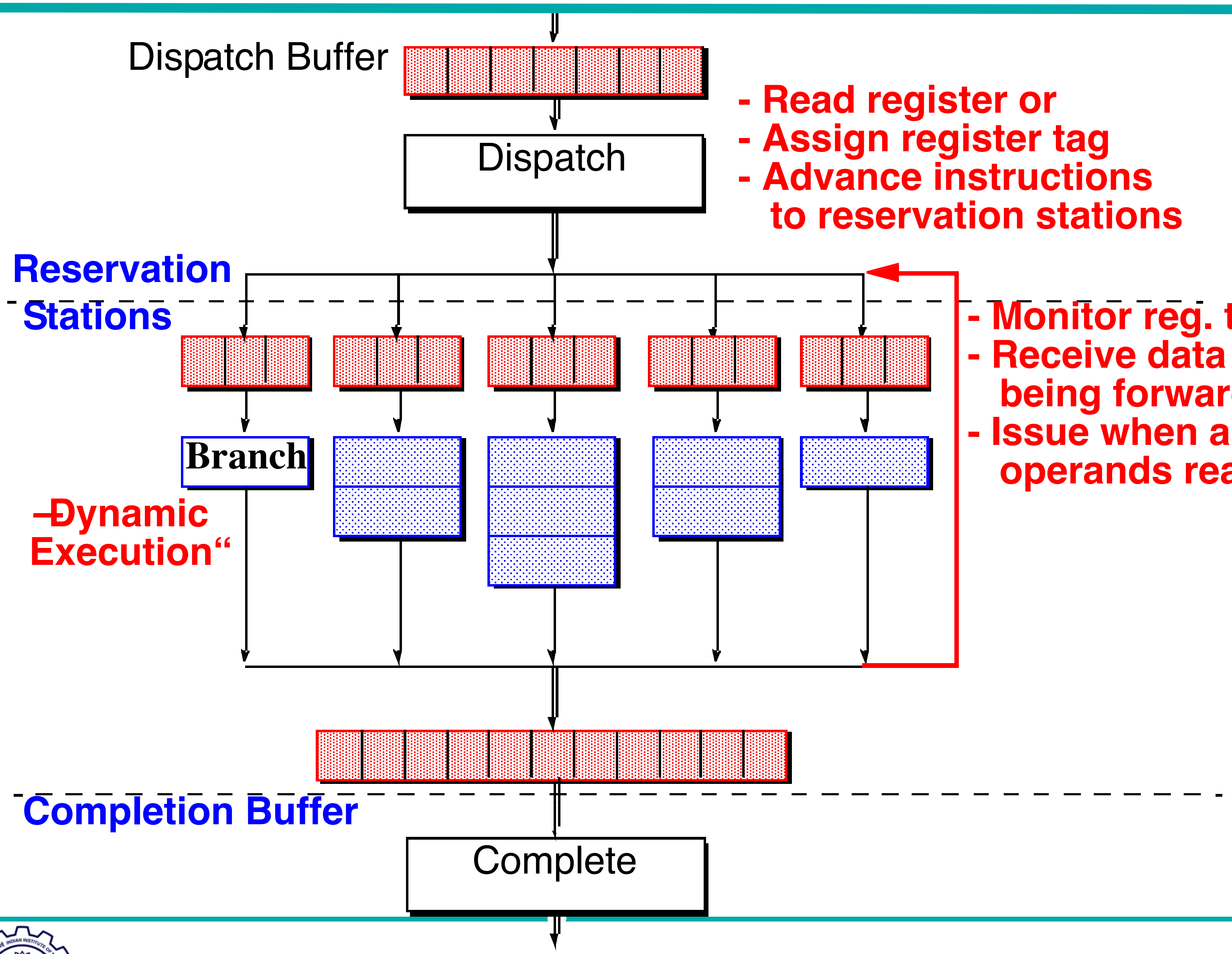


**STALL DISPATCHING**

**ADVANCE INSTRUCTIONS**

1) Read register(s), get "IOU" if not ready
2) Advance to reservation station
3) Wait for "IOU" to show up
4) Execute

# Embedded "Data Flow" Engine

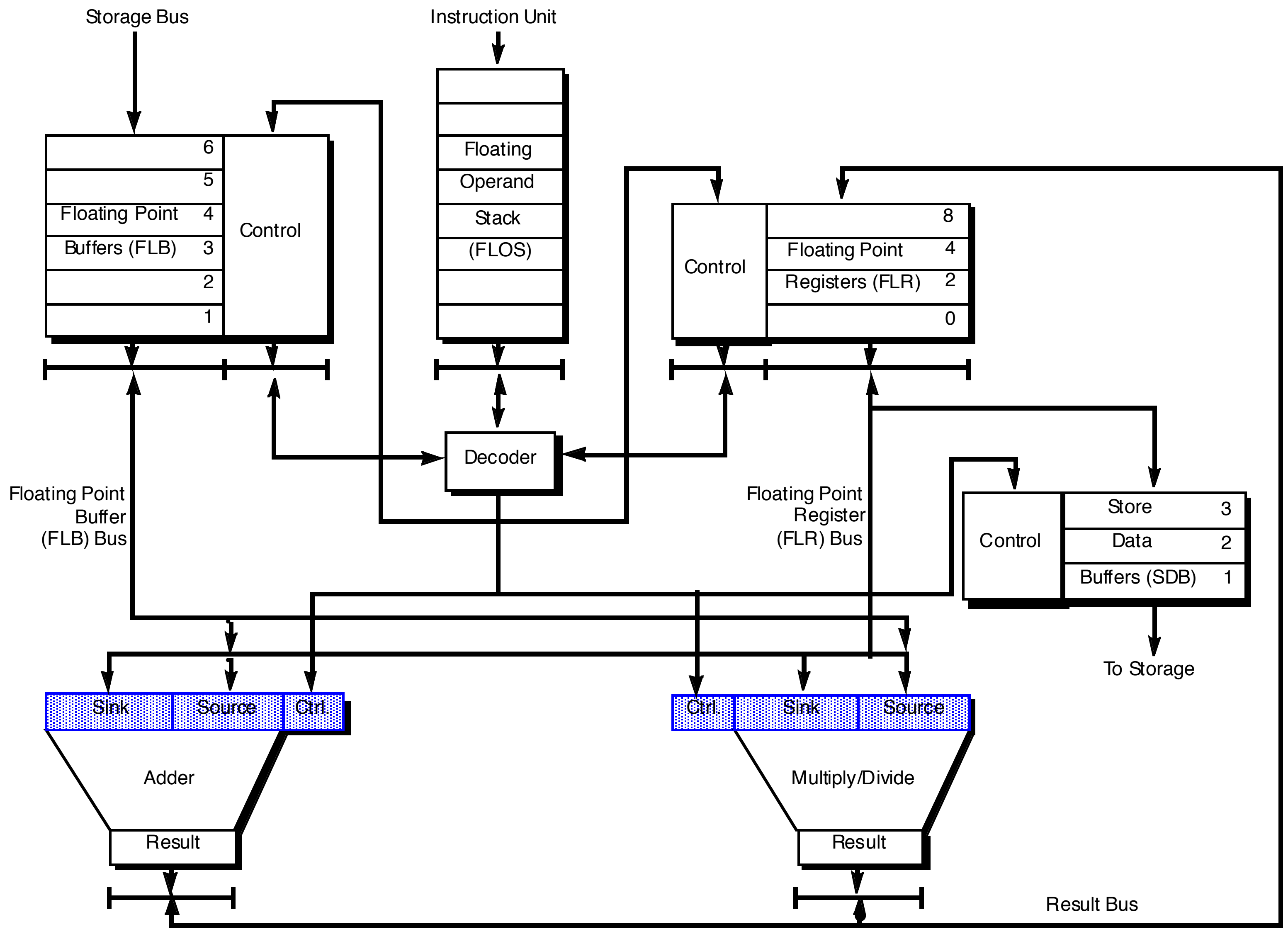


- Read register or
- Assign register tag
- Advance instructions to reservation stations

- Monitor reg. tag
- Receive data being forwarded
- Issue when all operands ready

"Dynamic Execution"

# Tomasulo's Algorithm [Tomasulo, 1967]



# IBM 360/91 FPU

- **Multiple functional units (FU's)**
  - Floating-point add
  - Floating-point multiply/divide
- **Three register files (pseudo reg-reg machine in floating-point unit)**
  - (4) floating-point registers (FLR)
  - (6) floating-point buffers (FLB)
  - (3) store data buffers (SDB)
- **Out of order instruction execution**:
  - After decode the instruction unit passes all floating point instructions (in order) to the floating-point operation stack (FLOS) [actually a queue, not a stack]
  - In the floating point unit, instructions are then further decoded and issued from the FLOS to the two FU's
- **Variable operation latencies**:
  - Floating-point add: 2 cycles
  - Floating-point multiply: 3 cycles
  - Floating-point divide: 12 cycles
- Goal: **achieve concurrent execution of multiple floating-point instructions, in addition to achieving one instruction per cycle in instruction pipeline**

# Dependence Mechanisms

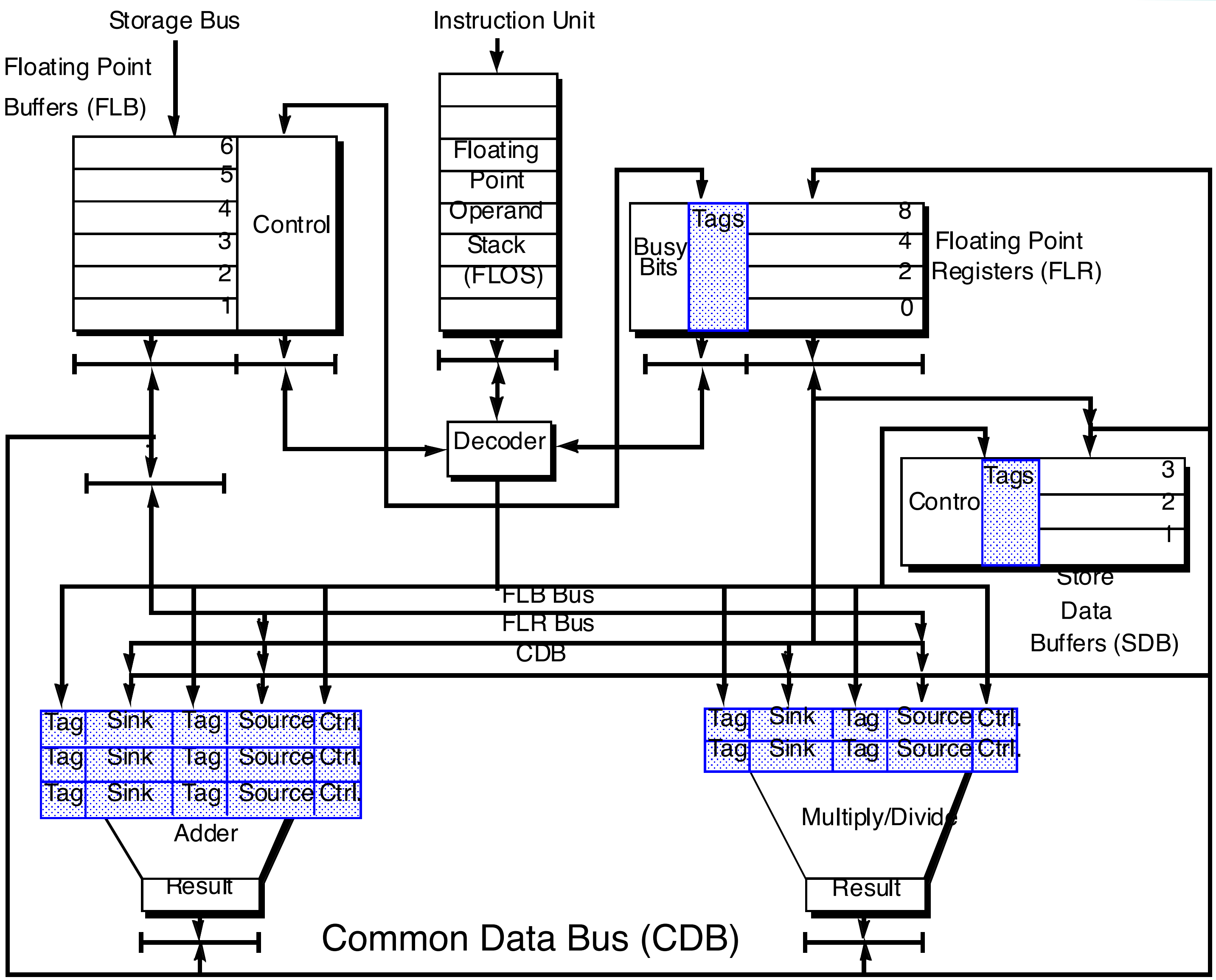
**Two Address IBM 360 Instruction Format:**

**R1 <-- R1 op R2**

**<u>Major dependence mechanisms:</u>**

- **Structural (FU) dependence = > virtual FU' s**
  - Reservation stations
- **True dependence = > pseudo operands + result forwarding**
  - Register tags
  - Reservation stations
  - Common data bus (CDB)
- **Anti-dependence = > operand copying**
  - Reservation stations
- **Output dependence = > register renaming + result forwarding**
  - Register tags
  - Reservation stations
  - Common data bus (CDB)

# IBM 360/91 FPU

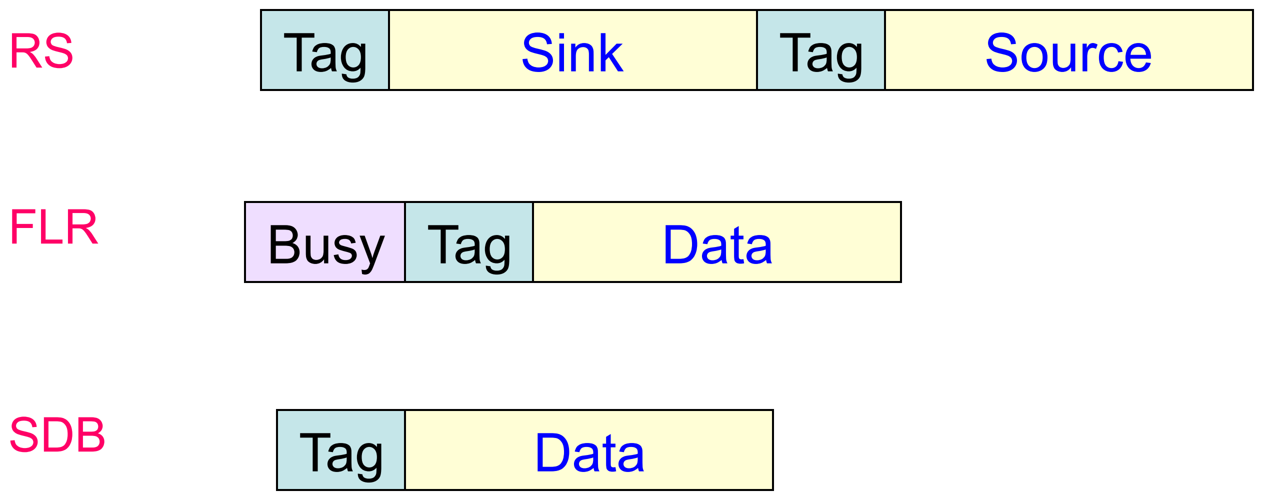

Storage Bus
Instruction Unit
Floating Point Buffers (FLB)
6
5
4
3
2
1
Control
Floating Point Operand Stack (FLOS)
Busy Bits
Tags
8
4
2
0
Floating Point Registers (FLR)
Decoder
Control
Tags
3
2
1
Store Data Buffers (SDB)
FLB Bus
FLR Bus
CDB
Tag Sink Tag Source Ctrl.
Tag Sink Tag Source Ctrl.
Tag Sink Tag Source Ctrl.
Adder
Result
Tag Sink Tag Source Ctrl.
Tag Sink Tag Source Ctrl.
Multiply/Divide
Result
Common Data Bus (CDB)

# Reservation Stations

- Used to collect operands or pseudo operands (tags).
- Associate more than one set of buffering registers (control, source, sink) with each FU, = > virtual FU's.
- Add unit: three reservation stations
- Multiply/divide unit: two reservation stations

| Tag | Sink | Tag | Source |
|---|---|---|---|
| 0 implies valid data | Source value | 0 implies valid data | Source value |

# Tomasulo’s Algorithm



# Tomasulo's Algorithm

w: R4 ← R0 + R8

x: R2 ← R0 * R4

y: R4 ← R4 + R8

z: R8 ← R4 * R2

# Tomasulo's Algorithm

w: R4 ← R0 + R8

x: R2 ← R0 * R4

y: R4 ← R4 + R8

z: R8 ← R4 * R2

Cycle 1: Dispatched instructions: w, x (in order)

RS (Adder)

| | | Tag | Sink | Tag | Source |
|---|---|---|---|---|---|
| w | 1 | 0 | 6.0 | 0 | 7.8 |
| | 2 | | | | |
| | 3 | | | | |

RS (Mult/Div)

| | | Tag | Sink | Tag | Source |
|---|---|---|---|---|---|
| x | 4 | 0 | 6.0 | 1 | ----- |
| | 5 | | | | |

FLR

| | Busy | Tag | Data |
|---|---|---|---|
| 0 | | | 6.0 |
| 2 | yes | 4 | 3.5 |
| 4 | yes | 1 | 10.0 |
| 8 | | | 7.8 |

# Tomasulo's Algorithm

w: R4 ← R0 + R8

x: R2 ← R0 * R4

y: R4 ← R4 + R8

z: R8 ← R4 * R2

Cycle 2: Dispatched instructions: y, z (in order)

RS

| | | Tag | Sink | Tag | Source |
|---|---|---|---|---|---|
| w | 1 | 0 | 6.0 | 0 | 7.8 |
| y | 2 | 1 | --- | 0 | 7.8 |
| | 3 | | | | |

Adder

RS

| | | Tag | Sink | Tag | Source |
|---|---|---|---|---|---|
| x | 4 | 0 | 6.0 | 1 | ----- |
| z | 5 | 2 | --- | 4 | --- |

Mult/Div

FLR

| | Busy | Tag | Data |
|---|---|---|---|
| 0 | | | 6.0 |
| 2 | yes | 4 | 3.5 |
| 4 | yes | 2 | 10.0 |
| 8 | yes | 5 | 7.8 |

# Tomasulo's Algorithm

w: R4 ← R0 + R8

x: R2 ← R0 * R4

y: R4 ← R4 + R8

z: R8 ← R4 * R2

Cycle 3: Dispatched instructions:

RS

| | | Tag | Sink | Tag | Source |
|---|---|---|---|---|---|
| | | | | | |
| y | 2 | 0 | 13.8 | 0 | 7.8 |
| | 3 | | | | |

Adder

RS

| | | Tag | Sink | Tag | Source |
|---|---|---|---|---|---|
| x | 4 | 0 | 6.0 | 0 | 13.8 |
| z | 5 | 2 | --- | 4 | --- |

Mult/Div

FLR

| | Busy | Tag | Data |
|---|---|---|---|
| 0 | | | 6.0 |
| 2 | yes | 4 | 3.5 |
| 4 | yes | 2 | 10.0 |
| 8 | yes | 5 | 7.8 |

# Tomasulo's Algorithm

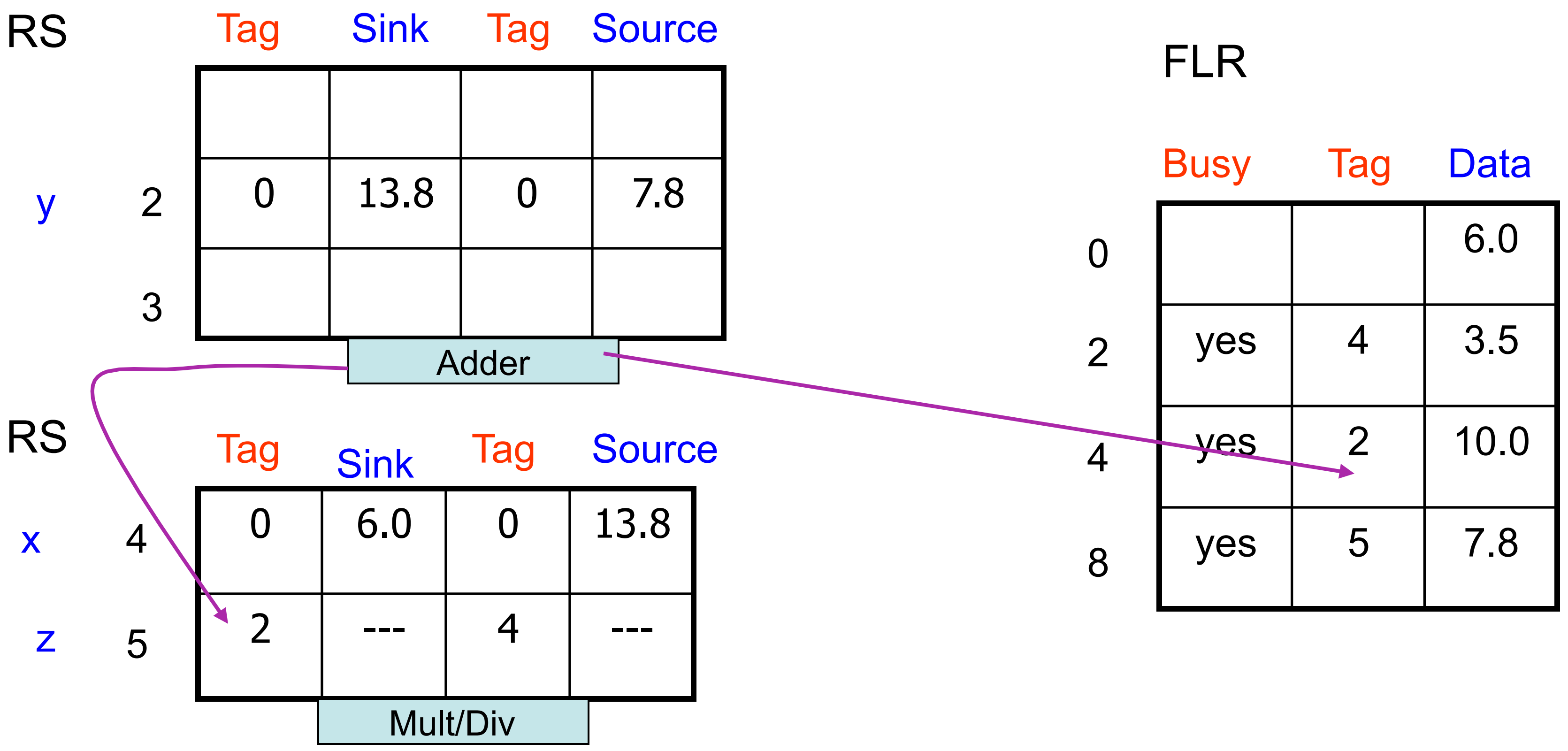
w: R4 ← R0 + R8

x: R2 ← R0 * R4

y: R4 ← R4 + R8

z: R8 ← R4 * R2

Cycle 4: Dispatched instructions:



RS (Adder)

| | | Tag | Sink | Tag | Source |
|---|---|---|---|---|---|
| | | | | | |
| y | 2 | 0 | 13.8 | 0 | 7.8 |
| | 3 | | | | |

RS (Mult/Div)

| | | Tag | Sink | Tag | Source |
|---|---|---|---|---|---|
| x | 4 | 0 | 6.0 | 0 | 13.8 |
| z | 5 | 2 | --- | 4 | --- |

FLR

| | Busy | Tag | Data |
|---|---|---|---|
| 0 | | | 6.0 |
| 2 | yes | 4 | 3.5 |
| 4 | yes | 2 | 10.0 |
| 8 | yes | 5 | 7.8 |

# Tomasulo’s Algorithm

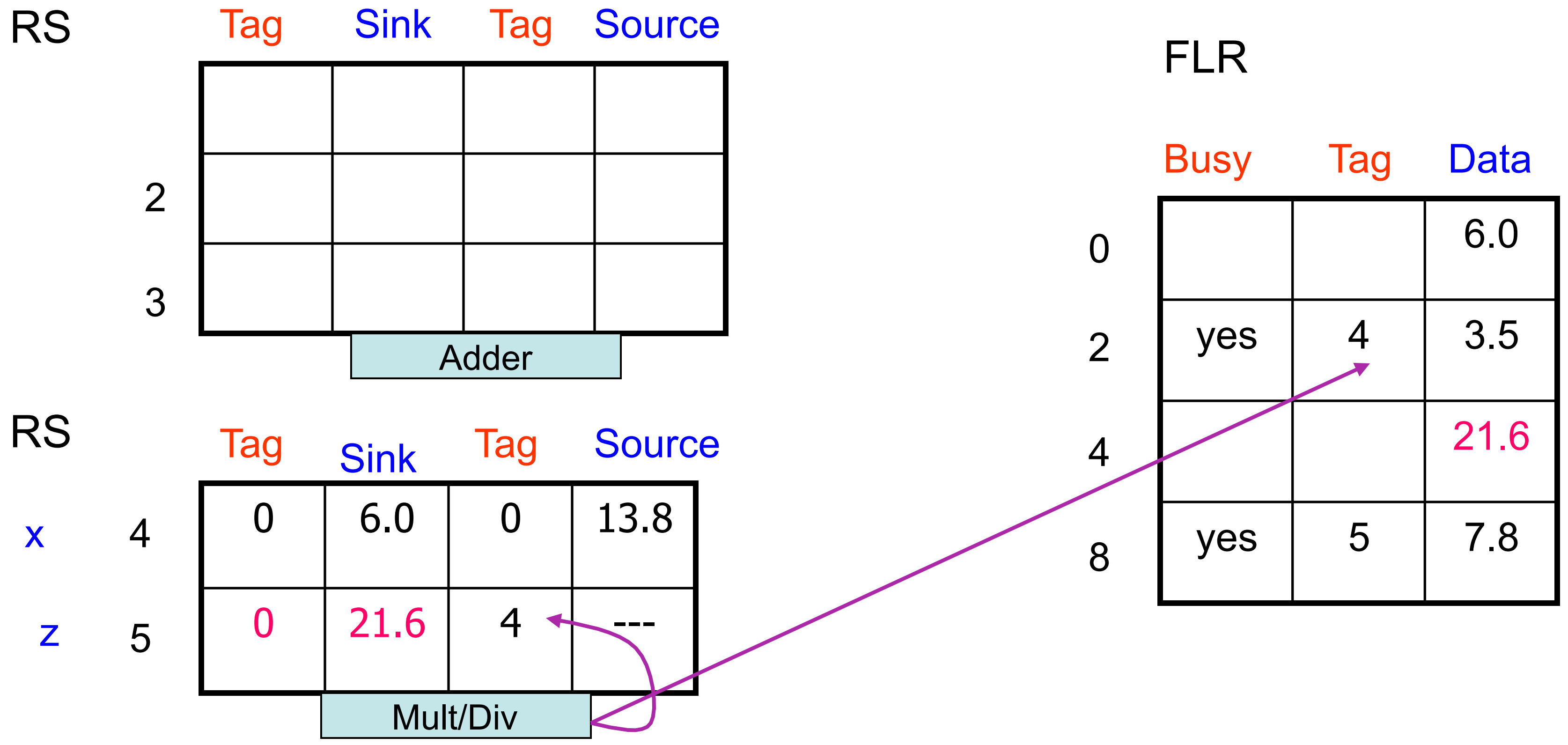
w: R4 ← R0 + R8

x: R2 ← R0 * R4

y: R4 ← R4 + R8

z: R8 ← R4 * R2

Cycle 5: Dispatched instructions:



RS

| | | Tag | Sink | Tag | Source |
|---|---|---|---|---|---|
| | | | | | |
| | 2 | | | | |
| | 3 | | | | |

Adder

RS

| | | Tag | Sink | Tag | Source |
|---|---|---|---|---|---|
| x | 4 | 0 | 6.0 | 0 | 13.8 |
| z | 5 | 0 | 21.6 | 4 | --- |

Mult/Div

FLR

| | Busy | Tag | Data |
|---|---|---|---|
| 0 | | | 6.0 |
| 2 | yes | 4 | 3.5 |
| 4 | | | 21.6 |
| 8 | yes | 5 | 7.8 |

# Tomasulo's Algorithm

w: R4 ← R0 + R8

x: R2 ← R0 * R4

y: R4 ← R4 + R8

z: R8 ← R4 * R2

Cycle 6: Dispatched instructions:

RS

| | Tag | Sink | Tag | Source |
|---|---|---|---|---|
| | | | | |
| 2 | | | | |
| 3 | | | | |

Adder

RS

| | | Tag | Sink | Tag | Source |
|---|---|---|---|---|---|
| | 4 | | | | |
| z | 5 | 0 | 21.6 | 0 | 82.8 |

Mult/Div

FLR

| | Busy | Tag | Data |
|---|---|---|---|
| 0 | | | 6.0 |
| 2 | | | 82.8 |
| 4 | | | 21.6 |
| 8 | yes | 5 | 7.8 |

# Tomasulo' s Algorithm

w: R4 ← R0 + R8

x: R2 ← R0 * R4

y: R4 ← R4 + R8

z: R8 ← R4 * R2

Cycle 7: Dispatched instructions:

RS

| | | Tag | Sink | Tag | Source |
|---|---|---|---|---|---|
| | | | | | |
| | 2 | | | | |
| | 3 | | | | |

Adder

RS

| | | Tag | Sink | Tag | Source |
|---|---|---|---|---|---|
| | 4 | | | | |
| z | 5 | 0 | 21.6 | 0 | 82.8 |

Mult/Div

FLR

| | Busy | Tag | Data |
|---|---|---|---|
| 0 | | | 6.0 |
| 2 | | | 82.8 |
| 4 | | | 21.6 |
| 8 | yes | 5 | 7.8 |

# Tomasulo’s Algorithm

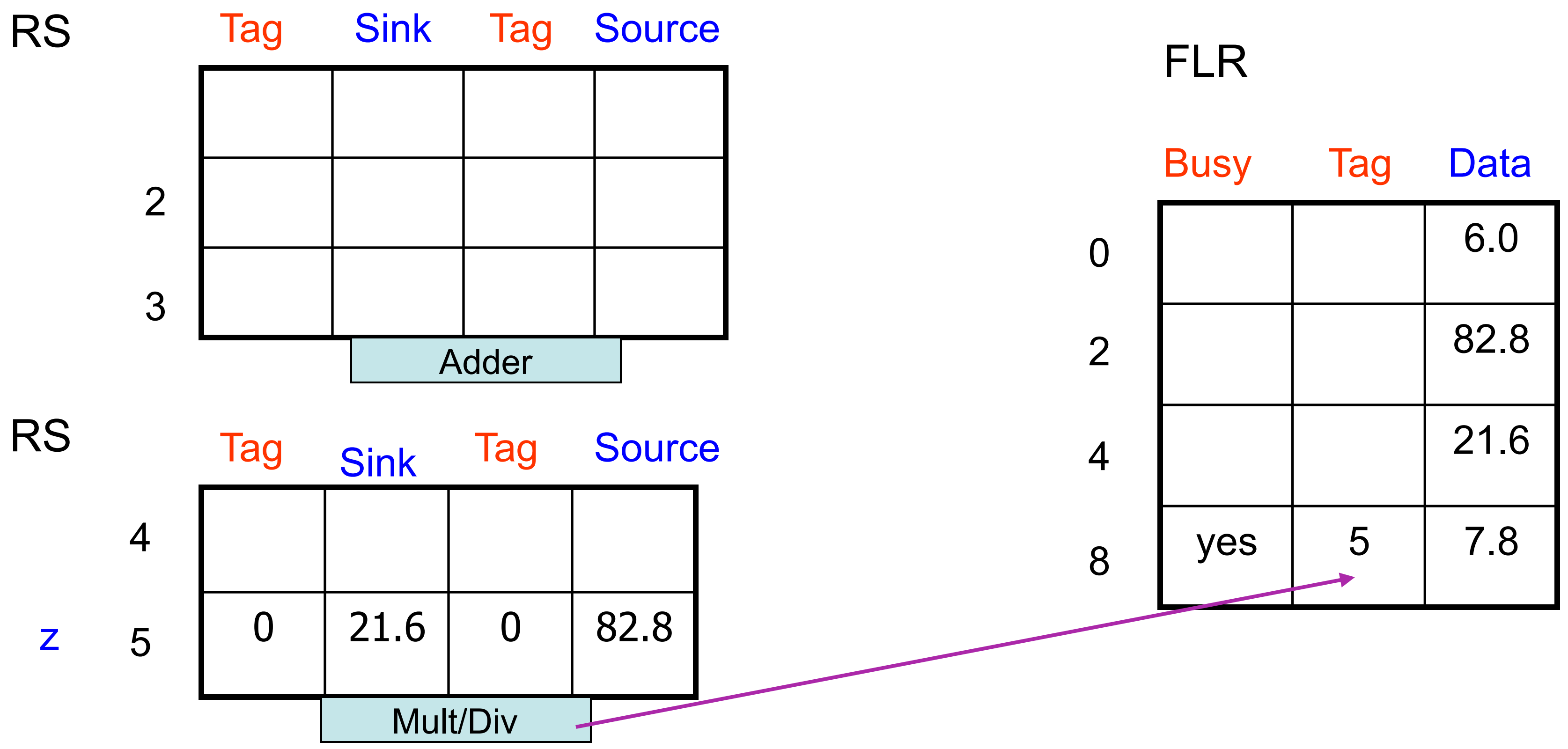
w: R4 ← R0 + R8

x: R2 ← R0 * R4

y: R4 ← R4 + R8

z: R8 ← R4 * R2

Cycle 8: Dispatched instructions:



RS

| | Tag | Sink | Tag | Source |
|---|---|---|---|---|
| | | | | |
| 2 | | | | |
| 3 | | | | |

Adder

RS

| | | Tag | Sink | Tag | Source |
|---|---|---|---|---|---|
| | 4 | | | | |
| z | 5 | 0 | 21.6 | 0 | 82.8 |

Mult/Div

FLR

| | Busy | Tag | Data |
|---|---|---|---|
| 0 | | | 6.0 |
| 2 | | | 82.8 |
| 4 | | | 21.6 |
| 8 | yes | 5 | 7.8 |

# Tomasulo's Algorithm

w: R4 ← R0 + R8

x: R2 ← R0 * R4

y: R4 ← R4 + R8

z: R8 ← R4 * R2

Cycle 9: Dispatched instructions:

RS

| | Tag | Sink | Tag | Source |
|---|---|---|---|---|
| | | | | |
| 2 | | | | |
| 3 | | | | |

Adder

RS

| | Tag | Sink | Tag | Source |
|---|---|---|---|---|
| 4 | | | | |
| 5 | | | | |

Mult/Div

FLR

| | Busy | Tag | Data |
|---|---|---|---|
| 0 | | | 6.0 |
| 2 | | | 82.8 |
| 4 | | | 21.6 |
| 8 | | | 1788.4 |

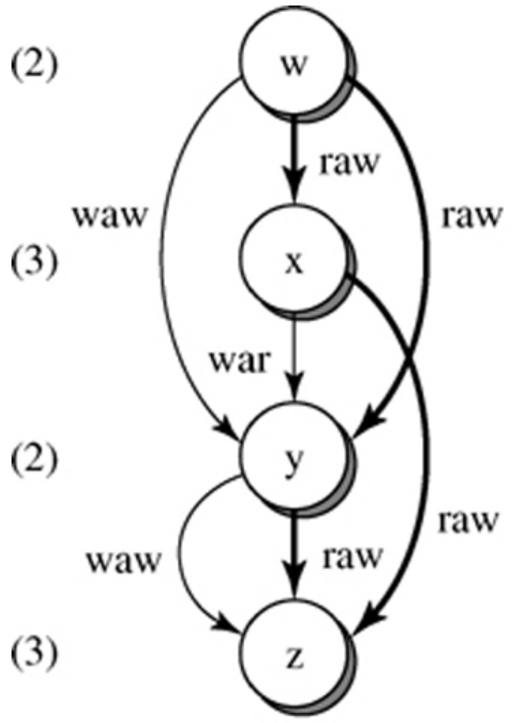
# Data Dependency

w: R4 ← R0 + R8

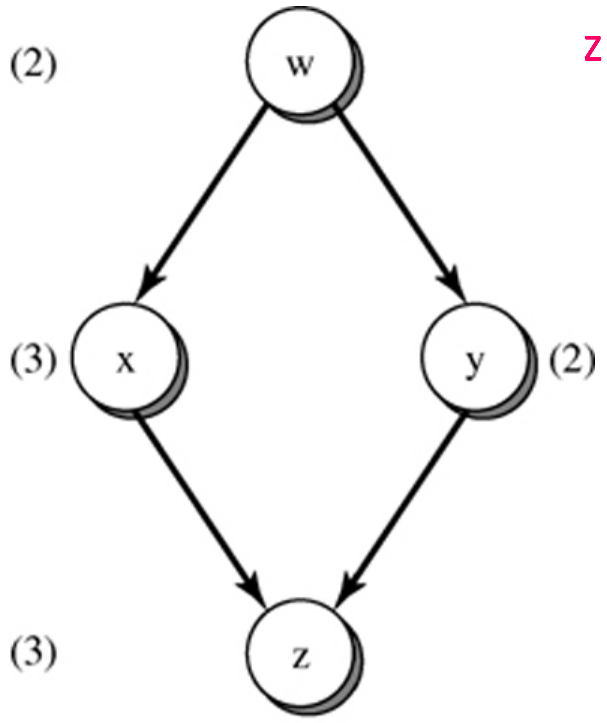
x: R2 ← R0 * R4

y: R4 ← R4 + R8

z: R8 ← R4 * R2

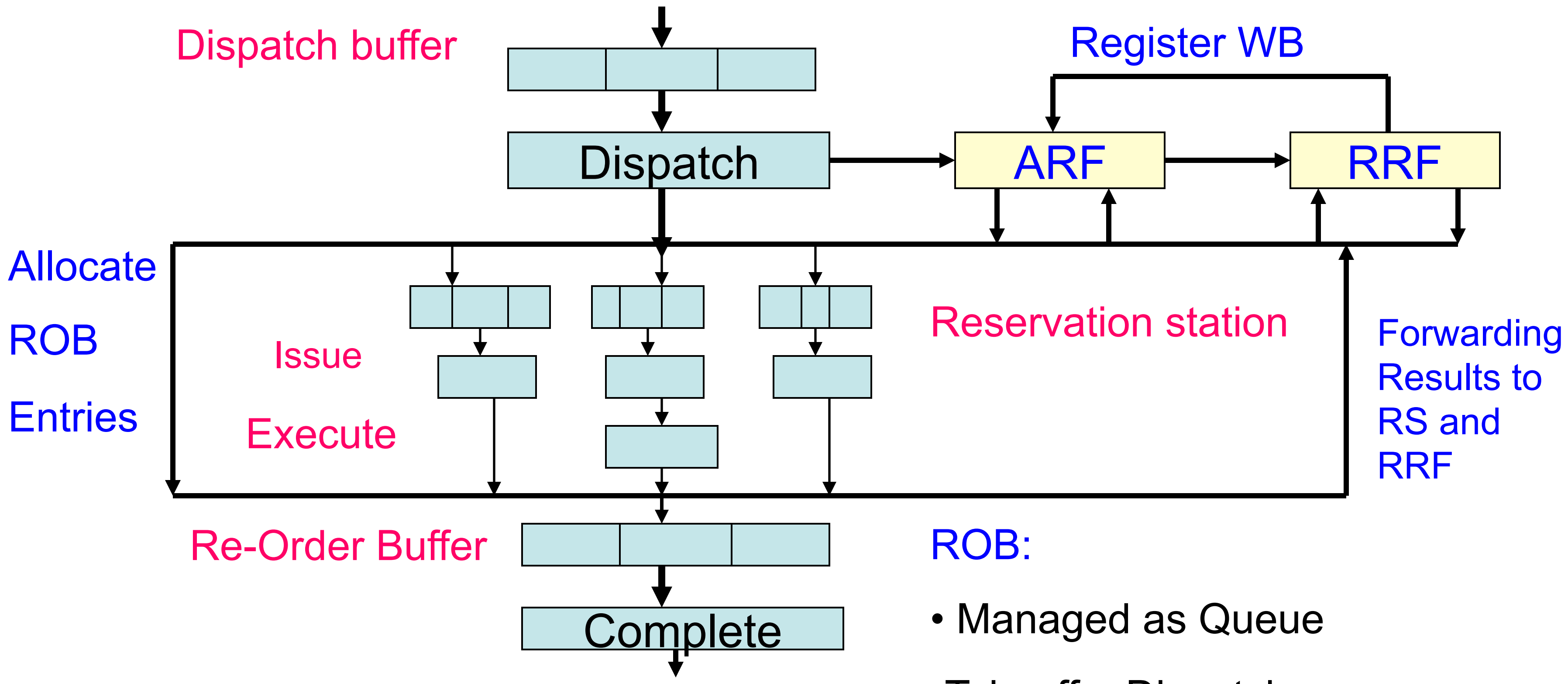


(a)



(b)

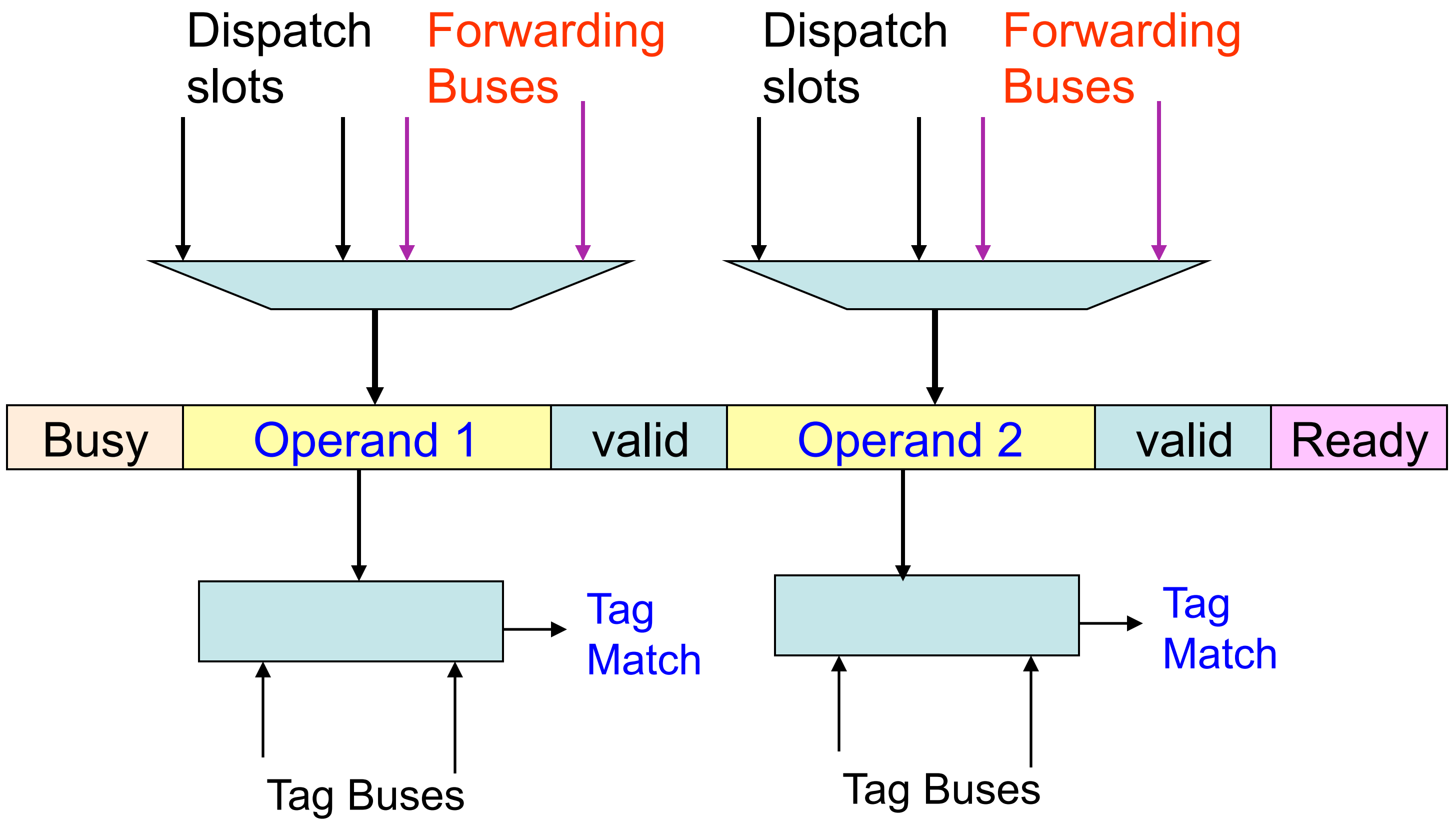
# Dynamic Execution Core



ROB:

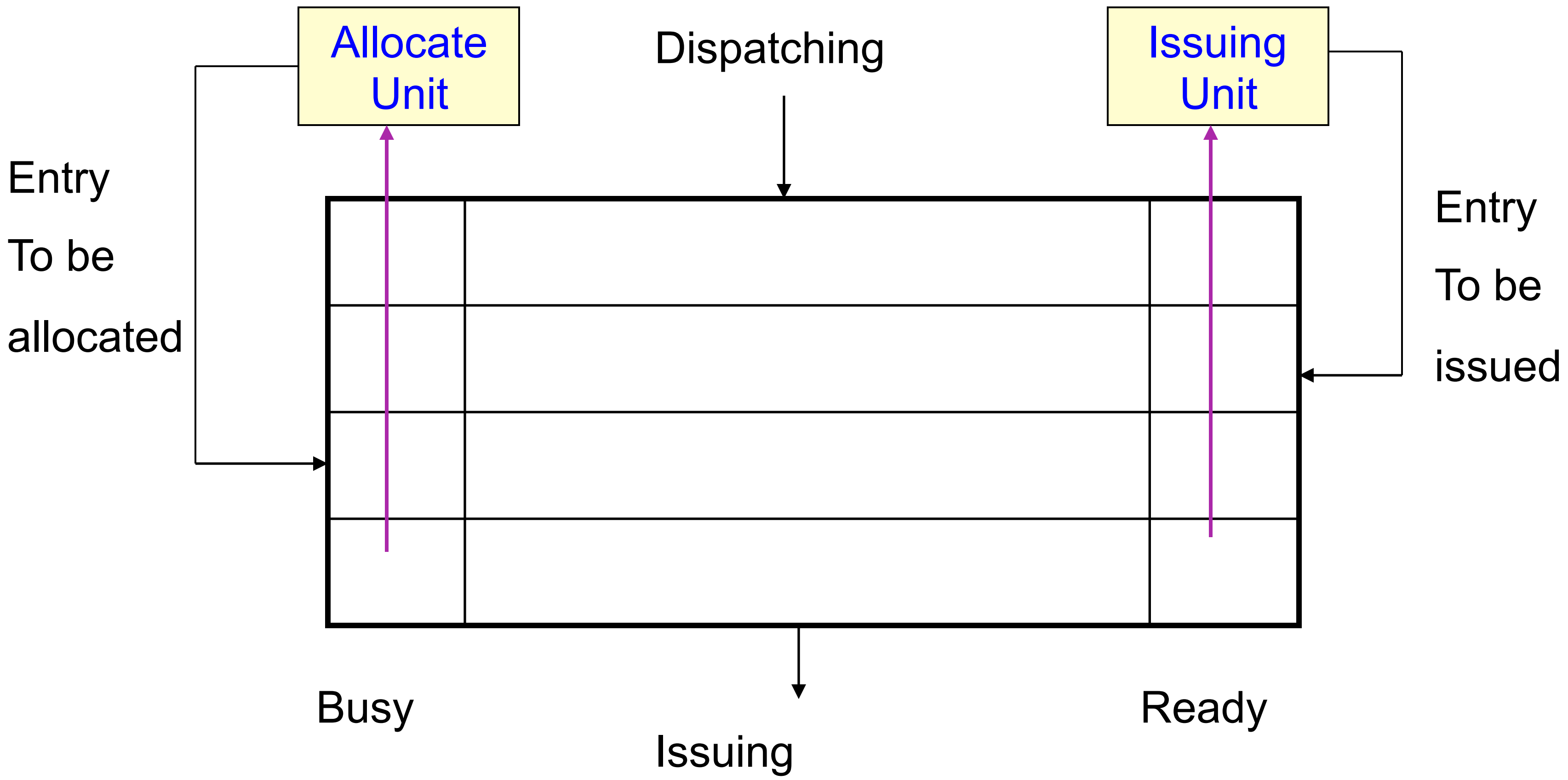
- Managed as Queue
- Takeoff – Dispatch
- Landing Completion

# Reservation Station



RS Entry

# Reservation Station



# Re-Order Buffer (ROB)

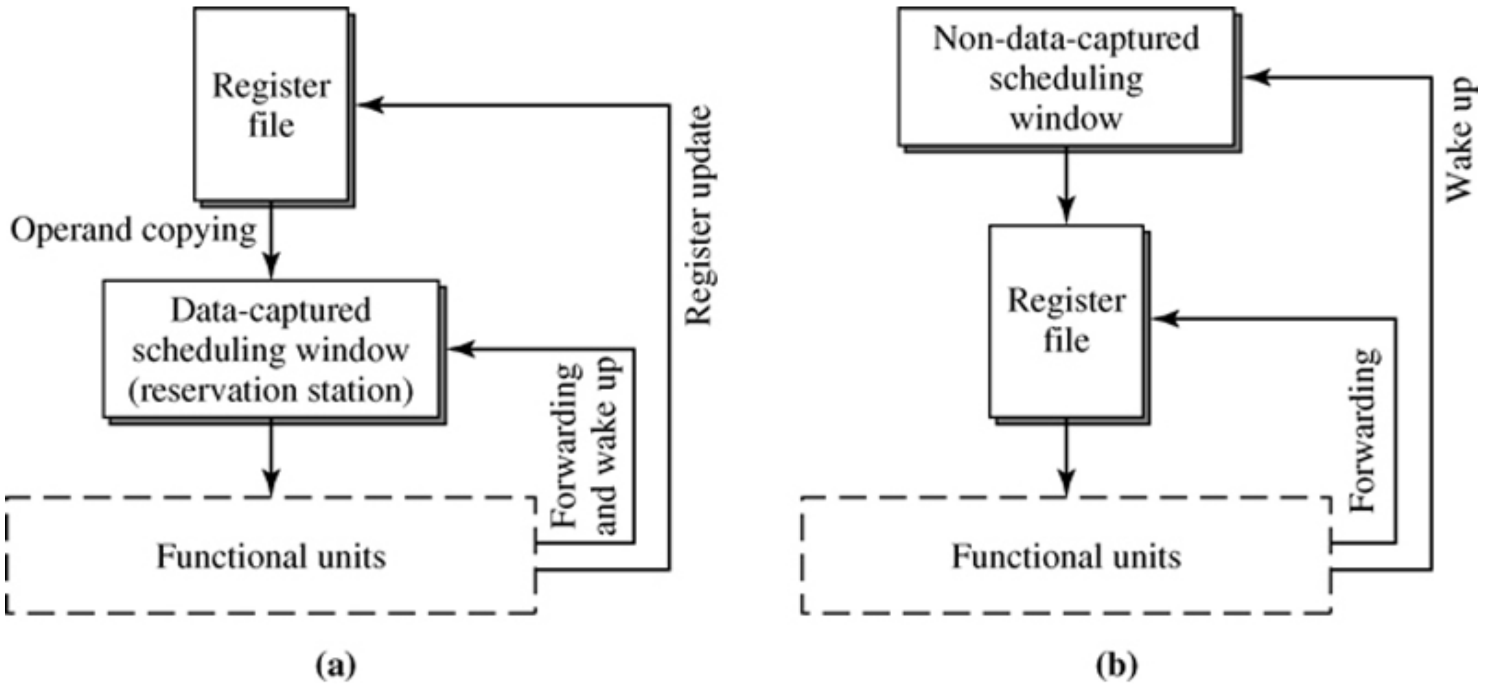
| Busy | Issued | Finished | Inst. Address | Rename Reg | Spec. | valid |
|---|---|---|---|---|---|---|

Next entry to be allocated (Tail pointer)

Next instruction to complete (Head pointer)

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| B | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| I | | | | | | | | | | |
| F | | | | | | | | | | |
| IA | | | | | | | | | | |
| RR | | | | | | | | | | |
| S | | | | | | | | | | |
| V | | | | | | | | | | |

# Dynamic Instruction Scheduler



# Thank You