

Issue Stage

MO801

Overview

- In-order
 - The oldest nonissued instructions are issued when their operands and required resources are available
- Out-of-order
 - Used by most of the latest processors
 - Instructions are issued as their operands and required resources are available
 - Can be based on
 - Reservation stations
 - Distributed issue queues
 - Unified issue queues

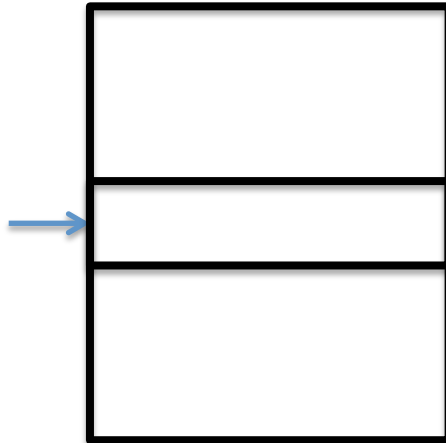
In-order

- Instructions are issued as soon as their operands and required resources are available
- The issue logic usually contains a simple scoreboard with two tables
 - One for data dependencies
 - One for hardware constraints

Issue Logic

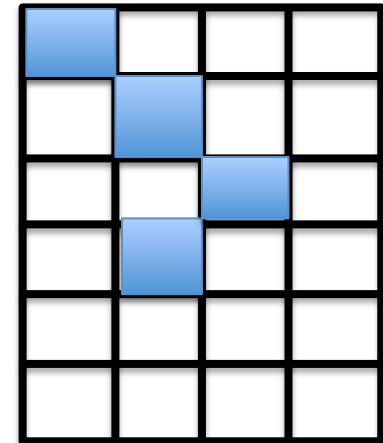
Data Dependence Table

Source Register



- Nonavailable
- Register File
- Bypass network

Resource Table

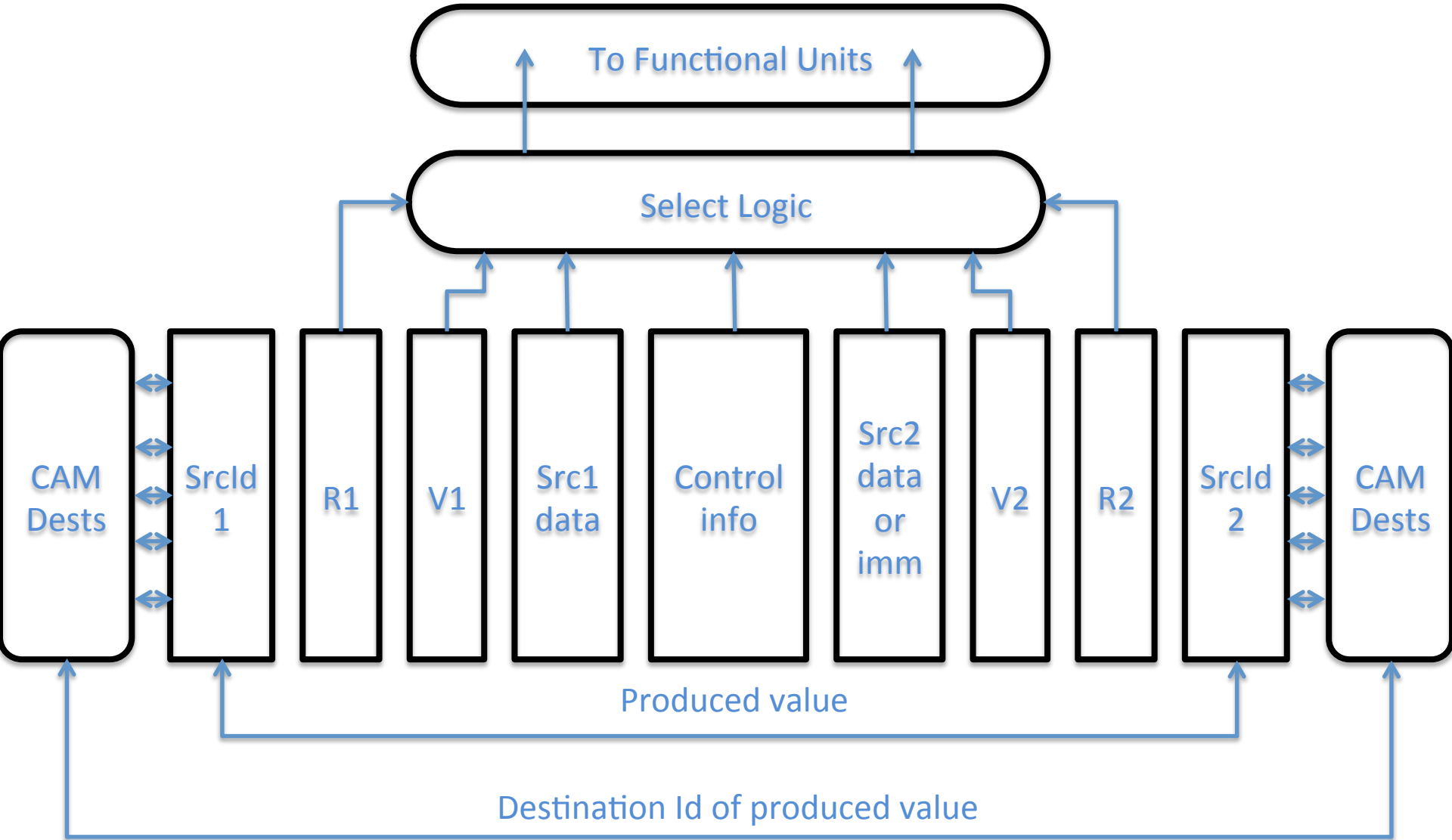


Out-of-Order

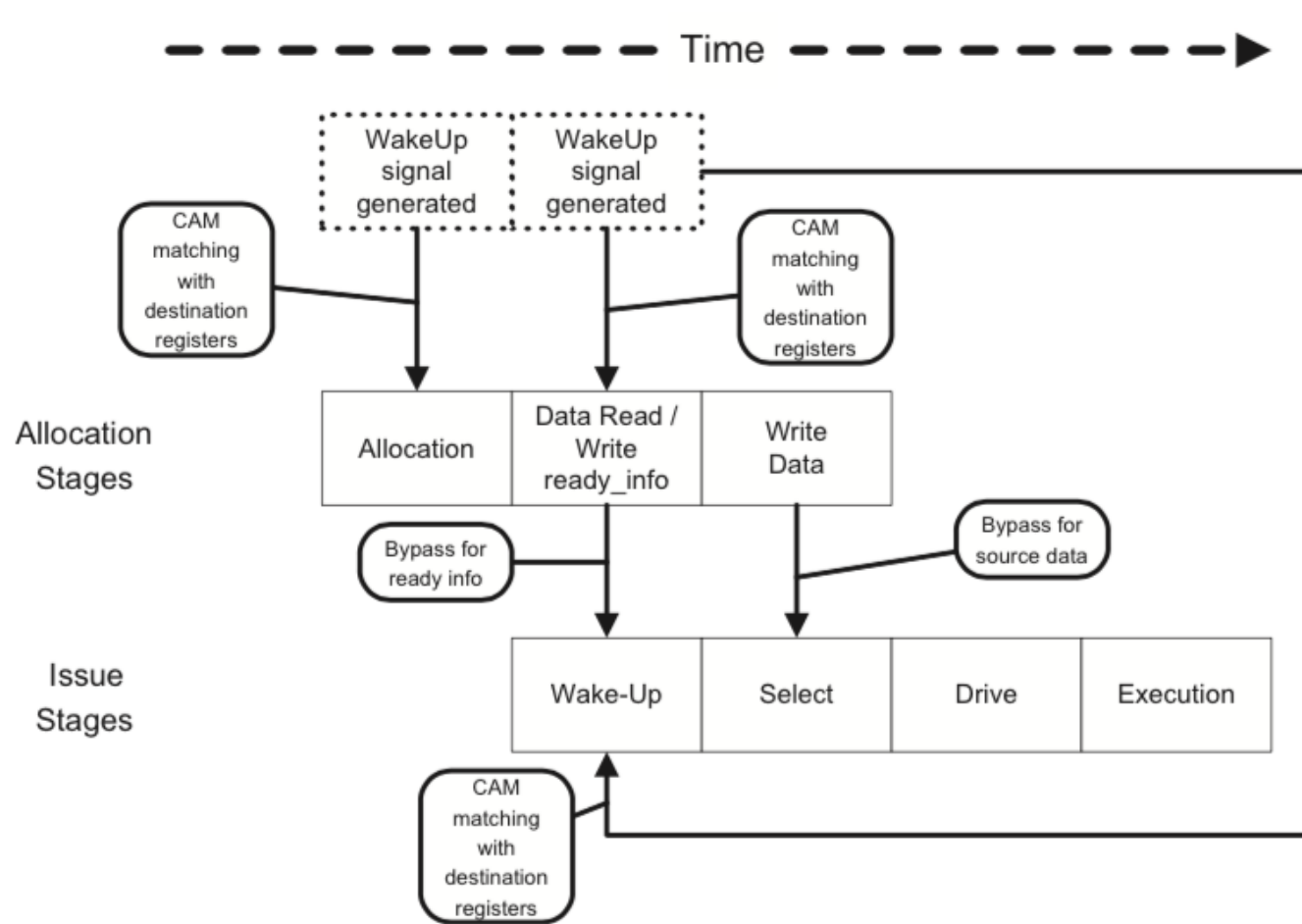
- Represents a critical aspect of the processor
 - Limits the number of instructions executed simultaneously
- Three key scenarios
 - **Unified issue queues**
 - Distributed issue queues
 - Reservation stations

Issue Process

(operands are read before issue)



Pipeline stages w/issue logic



Issue Queue Allocation

- Renaming (allocation) stage places new instructions in the issue queue
- Whenever there are no space available → stall renaming stage
- Avoid processing instructions while queue is full
- Read registers and set available bit

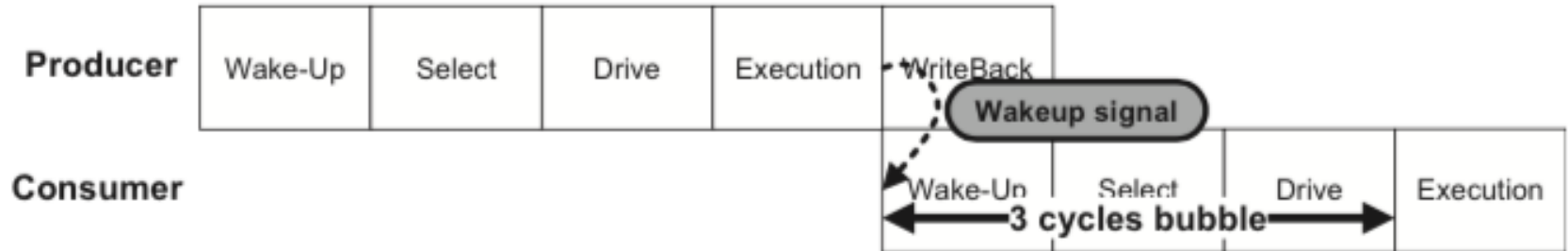
Instruction Wakeup

- Notifies that one operand has been produced
- Identifies the renaming ID and a valid bit
- As soon as the ready bits for both sources are set → instruction becomes ready (woken up)
- When the value is produced but not consumed (instruction not in queue), it should be stored elsewhere and valid bit should be set there too

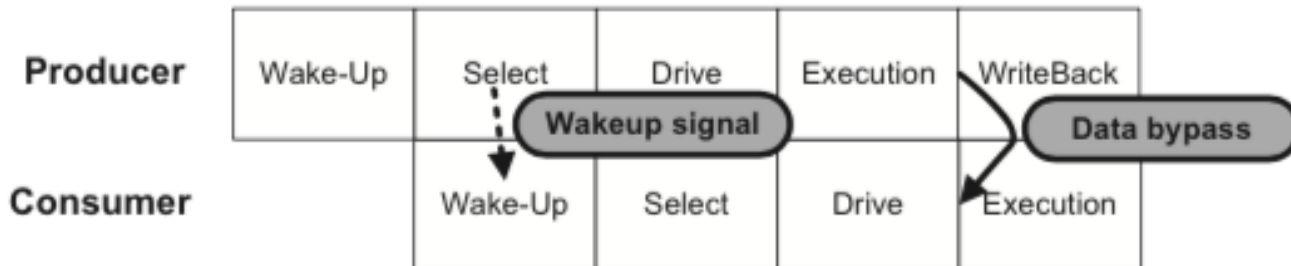
Early wakeup

----- Time ----->

1 WakeUp signal received when value becomes available



2 WakeUp signal received 3 cycles before becomes available



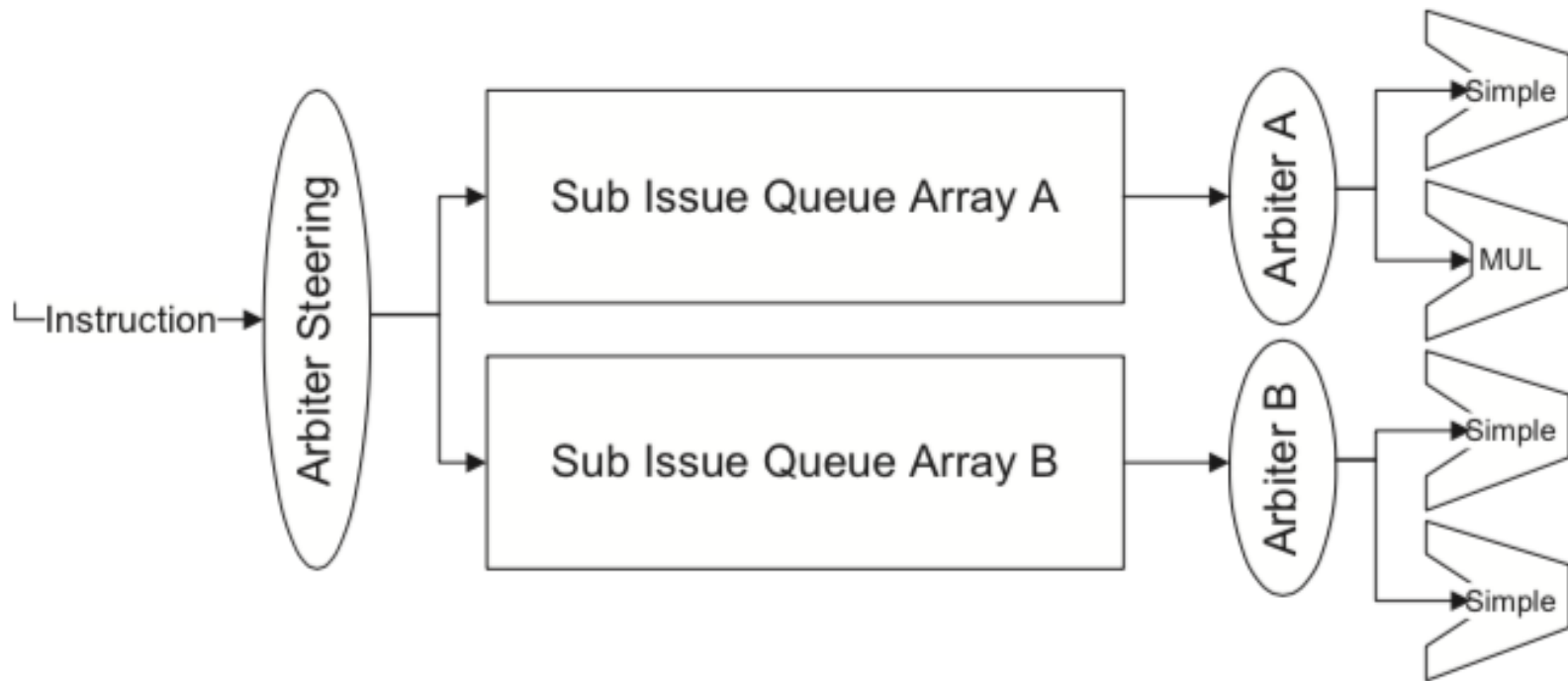
Early wakeup

- Can be issued when we know how long one instruction will take
- Issue 3 cycles before instruction finish, as in previous slide
- For load instructions, it is not possible to pre-calculate the latency
 - Wait until the end of load to wakeup consumer
 - Speculatively wakeup load consumer

Instruction Selection

- Selects the next instruction to be executed
 - Requires all source registers to be available
 - Requires enough hardware resources
- Usually split into arbiters or schedulers
 - Instead of issuing 4 instructions, use two arbiters to issue 2 instructions each
 - Instructions are assigned to different arbiters
 - Each arbiter is responsible to a subset of the instructions and functional unit

Selection Logic

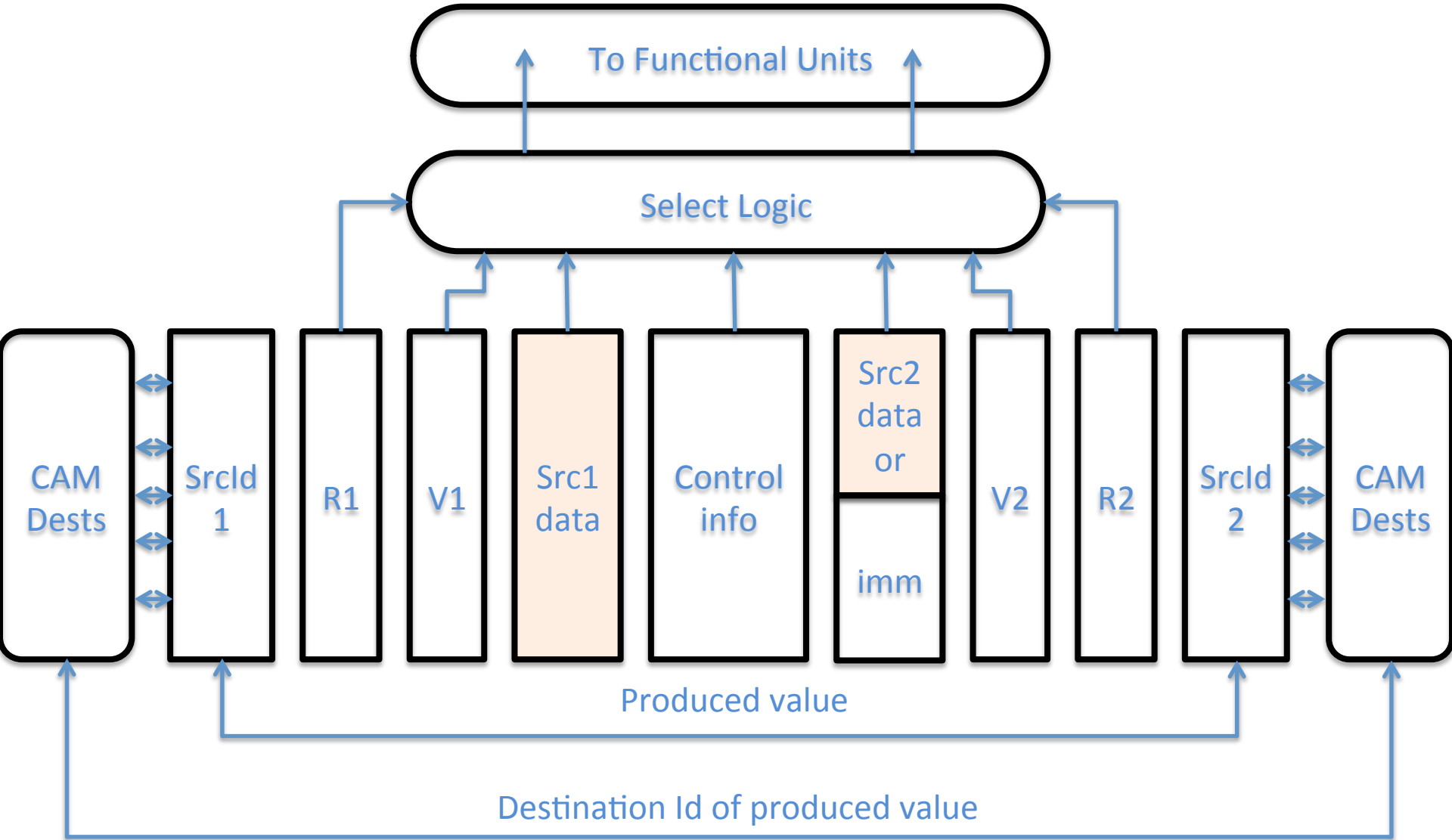


Entry reclamation

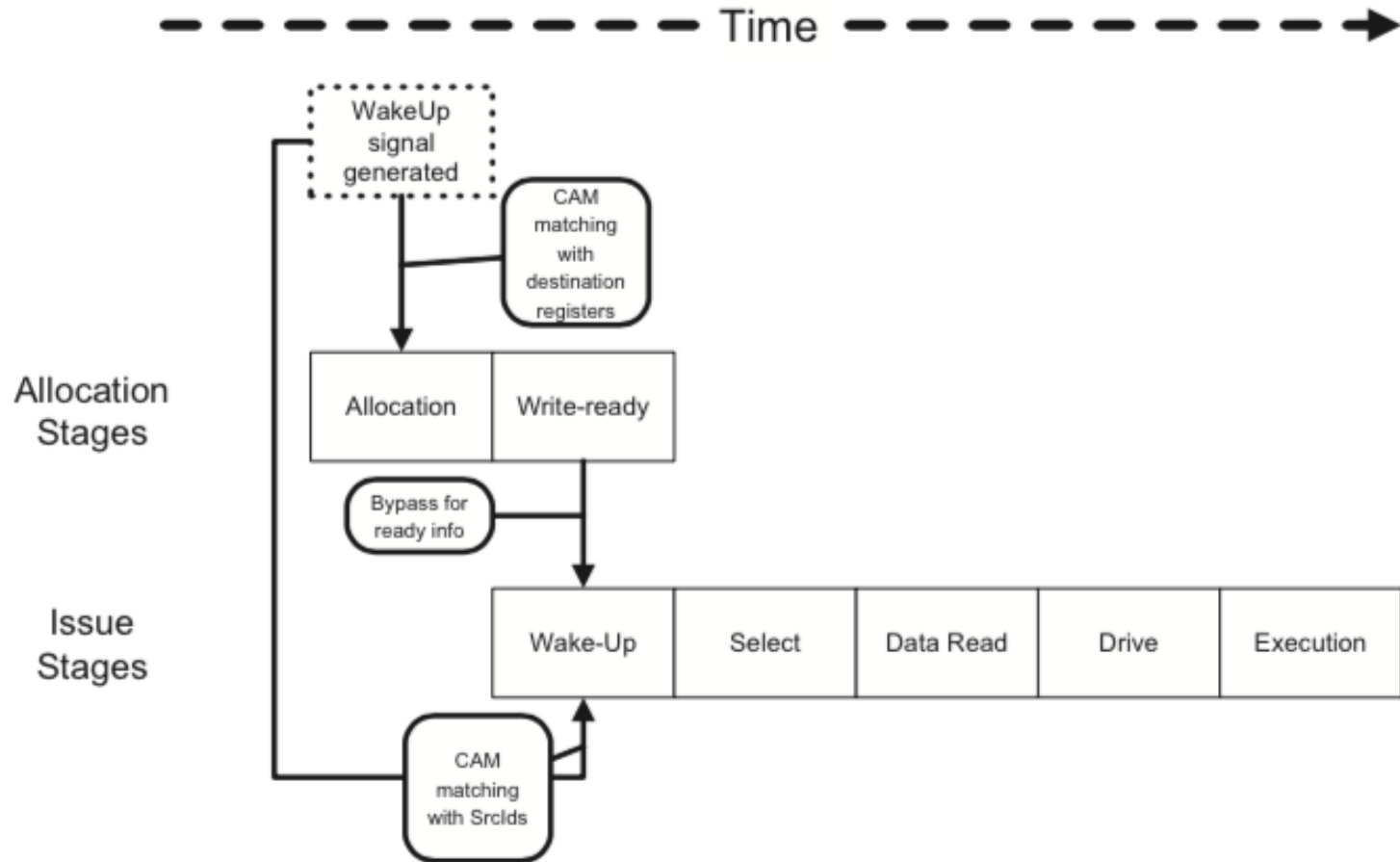
- Frees the instruction queue after instruction issue
- Can be delayed if the processor speculatively wakeup instructions

Issue Process

(operands are read after issue)



Pipeline



Read Port Reduction

- Reads after issue may require more read ports
 - Machine width vs issue width
- Some processors (Alpha) split the register file and the number of ports
- Most of the source data are read from the bypass network instead of from the register file
- Active port reduction
 - Synchronizes arbiters to use less read ports
- Reactive port reduction
 - Cancel instructions if the number of ports are bigger than the available
- Both require a fair policy to perform cancellation

Exercise

- Count the dynamic distance among data through instructions in a program. Create a histogram of the distances

```
add r1, r2, r3
```

```
sub r4, r5, r6
```

```
add r7, r1, r4
```

- Distances
 - r1 → 2
 - r4 → 1

Distributed Issue Queue

- Processors distribute functional units in execution clusters
- Each cluster implements its own issue queue
- Pentium 4 has two execution clusters
 - Memory operations
 - Nonmemory operations

Reservation Stations

- Buffers per functional unit
- Store instructions with their inputs
- Receive instructions right after renaming
- Instructions broadcast their produced values to all reservation stations
- Whenever an instruction has all its sources, it can be executed

Memory operations

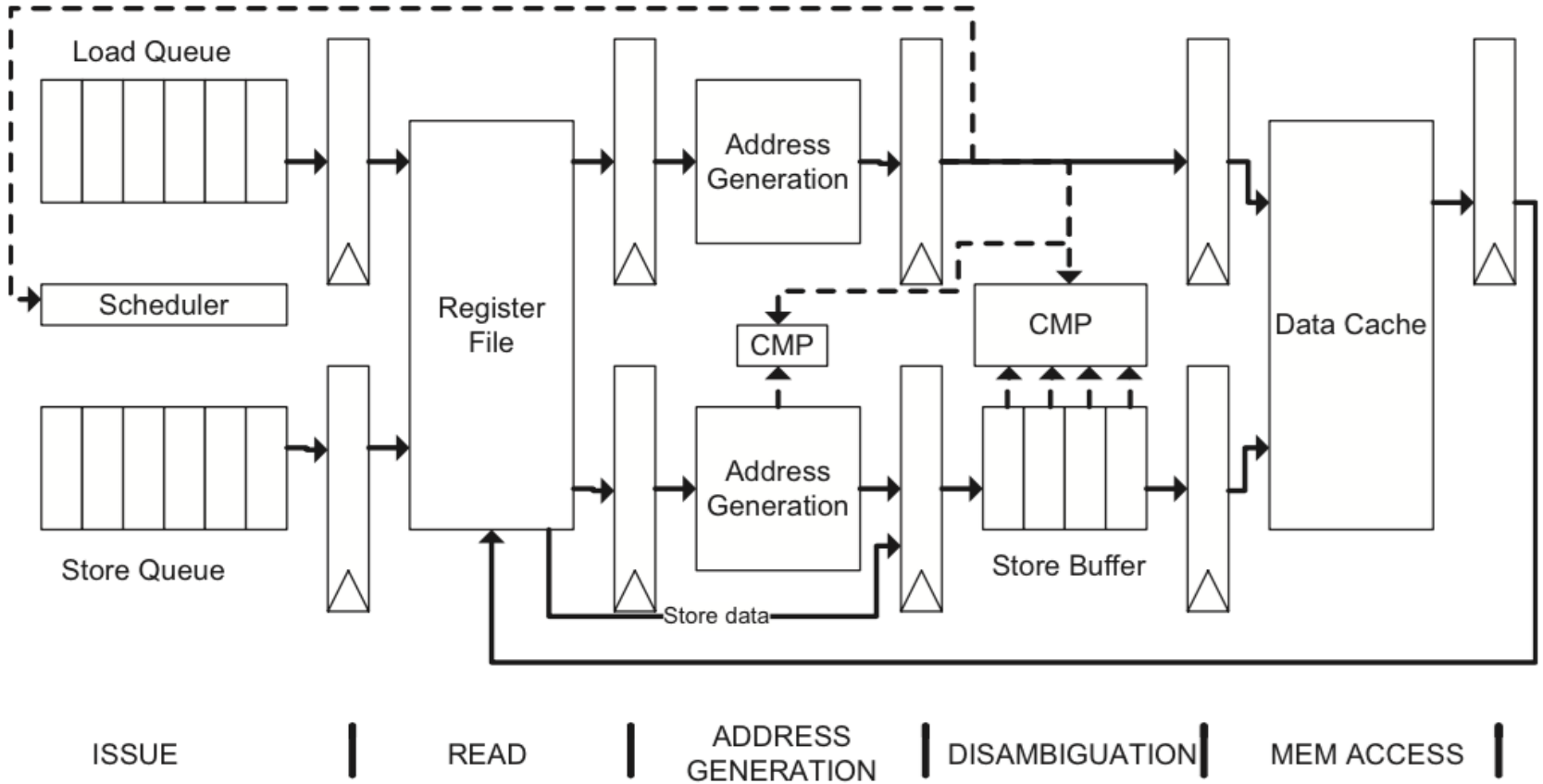
- Dependencies through memory are not solved by renaming
- Memory disambiguation → handles memory dependencies
- Nonspeculative disambiguation
 - Waits to be sure there is no memory dependency with previous operation
- Speculative disambiguation
 - Tries to predict whether memory operation have dependence
- 30% of instructions are memory operations

Memory disambiguation

Name	Speculative	Description
Total ordering	X	All memory accesses are processed in order
Partial ordering	X	All stores are processed in order, but loads execute out of order as long as all previous stores have computed their address
Load ordering Store ordering	X	Execution between loads and stores is out of order, but all loads execute in order among them, and all stores execute in order among them
Store ordering	✓	Stores execute in order, but loads execute completely out of order

AMD K6

Load ordering and store ordering

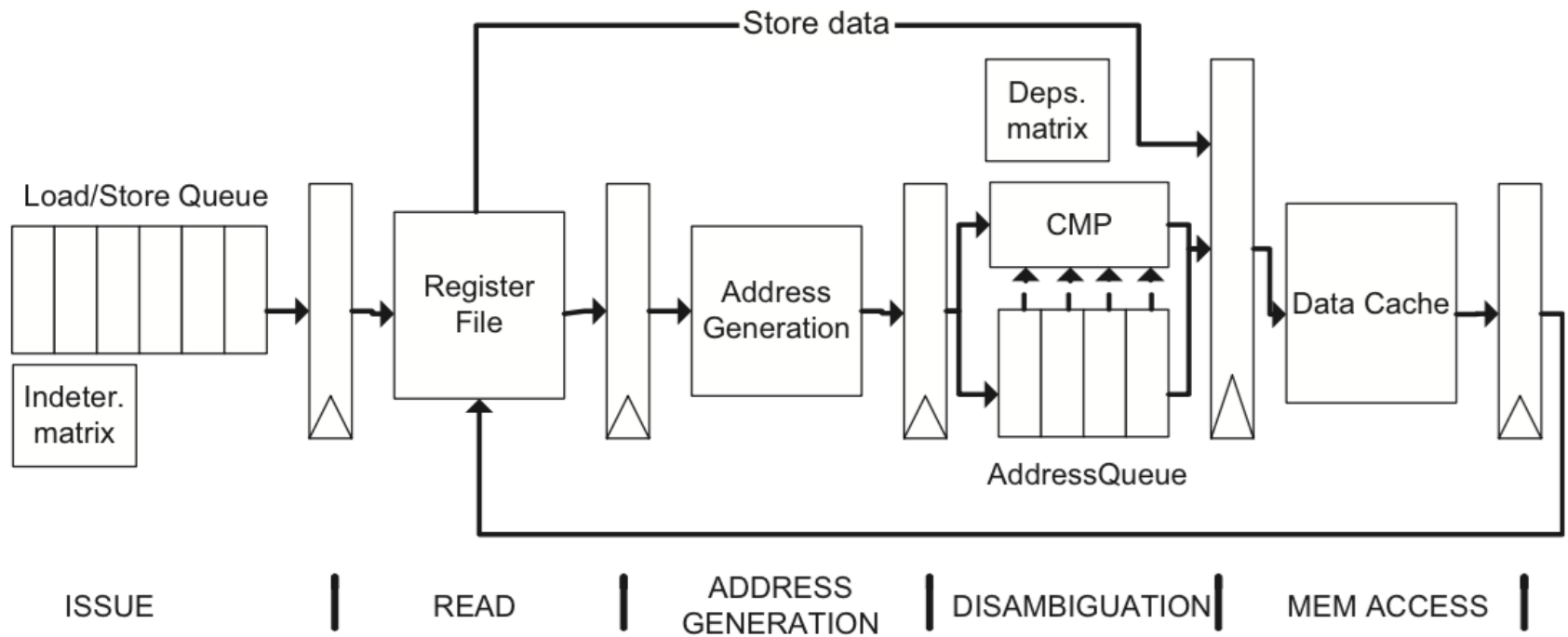


AMD K6

- Load queue
 - Keeps loads in program order. Loads stay in queue until they are the oldest on queue and their operands are ready
- Address generation
 - Calculates address of memory operations
- Store queue
 - Keeps the store operations in program order. Stores stay in queue until they are the oldest on queue and their operands are ready
- Store buffer
 - Keeps the store operations until they are the oldest in-flight instruction in the processor

MIPS R10000

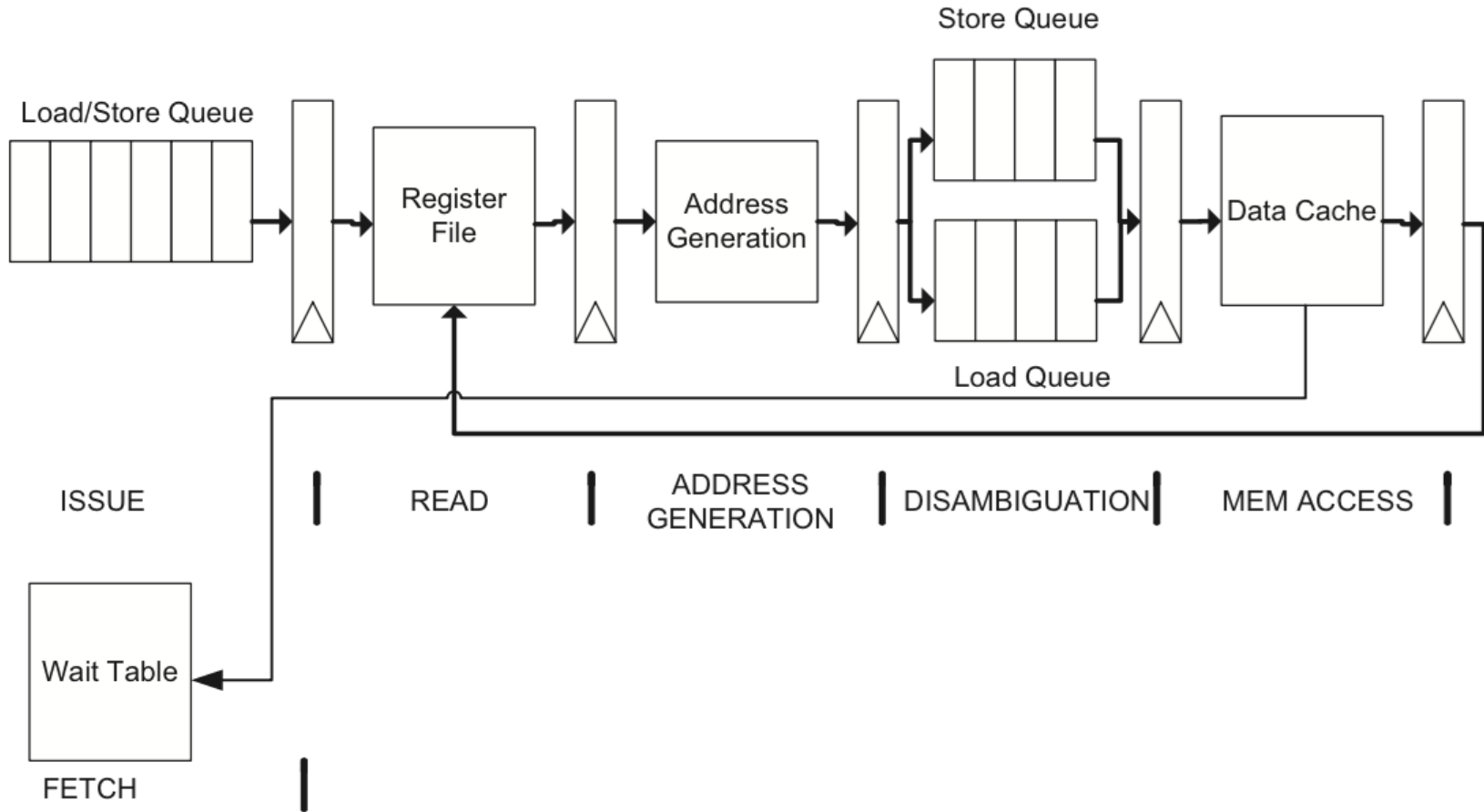
Partial ordering



MIPS R10000

- Load/store queue
 - 16-entry. Instructions wait until operands are ready
- Indetermination matrix
 - Used to mark whether instruction addresses are computed
- Dependency matrix
 - Store dependencies among memory operations
- Address generation
 - Compute memory address
- Address queue
 - Keeps memory addresses of loads and stores that want to access the cache

Speculative Memory Disambiguation



Alpha 21264

- Load/Store Queue
 - Holds memory operation until operands are ready
- Load Queue
 - Stores physical addresses of the loads in program order
- Store Queue
 - Stores the physical addresses of the stores and its data in program order
- Wait Table
 - Keeps track of loads to detect whenever it violates dependencies
 - Also tracks previous loads that caused dependencies so that they are not scheduled before the store it depends on

Speculative wakeup of load consumers

- Loads will take cycles to wakeup the next instruction
- Most of the time, Loads face a cache hit
- We can speculatively wakeup the next instructions and roll-back if there is a cache miss
 - Cancel the next instruction
 - Reissue it again
- Alternatives
 - Keep instruction in issue queue
 - Create additional structures to handle these instructions

Example

