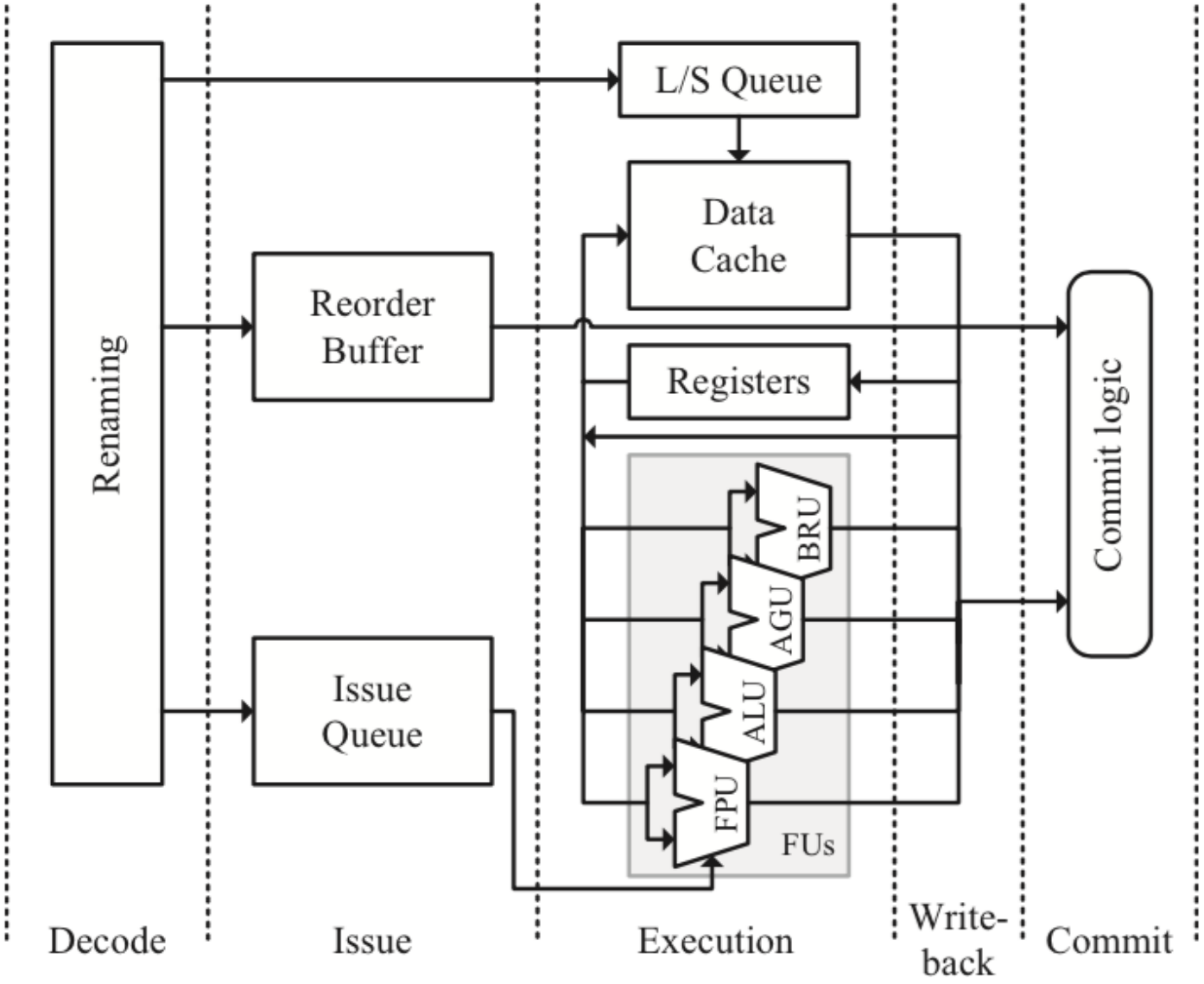
Execute

MO801

Execution Units



Execution Units

- Floating-Point Units → FPU
- Arighmetic and Logical Units → ALU
- Address Generation Units → AGU
- Branch Unit → BRU

Arithmetic and Logical Unit

- Performs arithmetic and logical operations
  - Addition, subtraction
  - AND, OR, NOT, XOR, NAND, NOR, XNOR
  - Shift, rotation, byte-swap
- Condition code or flags
  - For x86: sign, parity, adjust, zero, overflow and carry
  - Usually generated in parallel of each computation

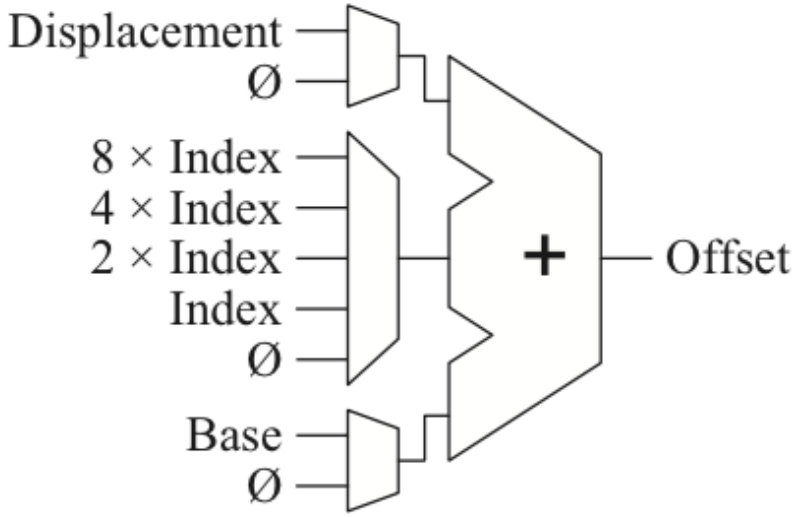
Integer Multiplication and Division

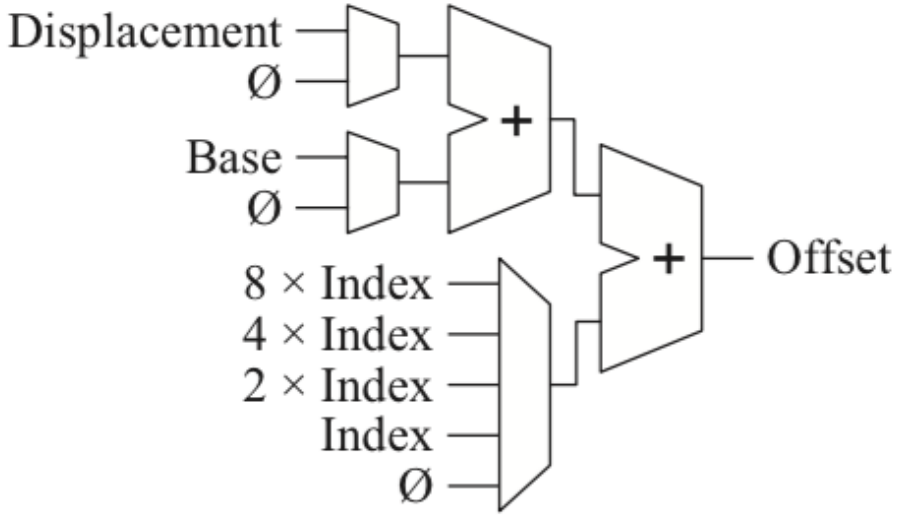
- Sometimes integrated to the conventional ALU
- For simplicity, can be implemented by
  - Converting integer to floating-point
  - Performing the floating-point operation
  - Converting the result back
  - This saves power and area at the cost of latency
- Different latencies for each operation and implementation

Address Generation Unit

- Converts the operands to a memory address
- Flat memory model
  - There is one continuous address space
  - Linear address space
- Segmented memory model
  - Several independent address spaces
  - Segment base address + offset
- The result is called effective address

x86 AGU

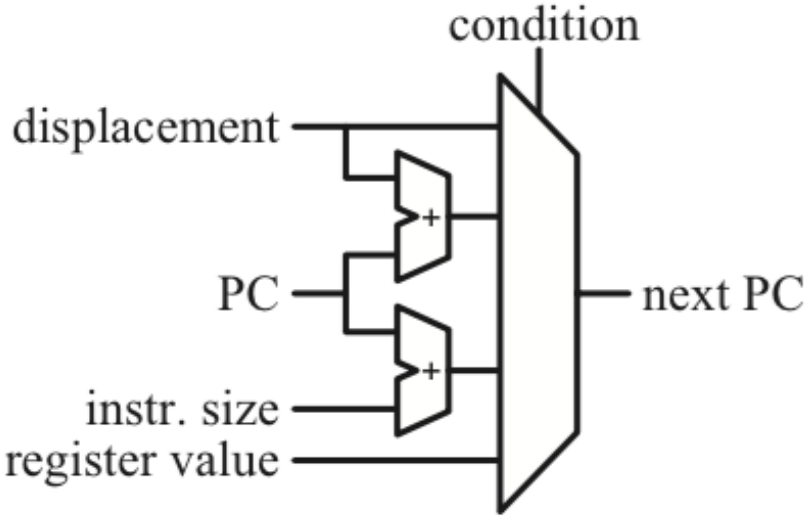




Wake-up	Select	Data read	Drive	Calculate offset / read segment info	Calculate linear address / check segment limit

Branch Unit

- Direct absolute
  - The instruction defines the next PC value explicitly
- Direct PC-relative
  - Adds an offset to create the next PC
- Indirect
  - Selects from an integer register



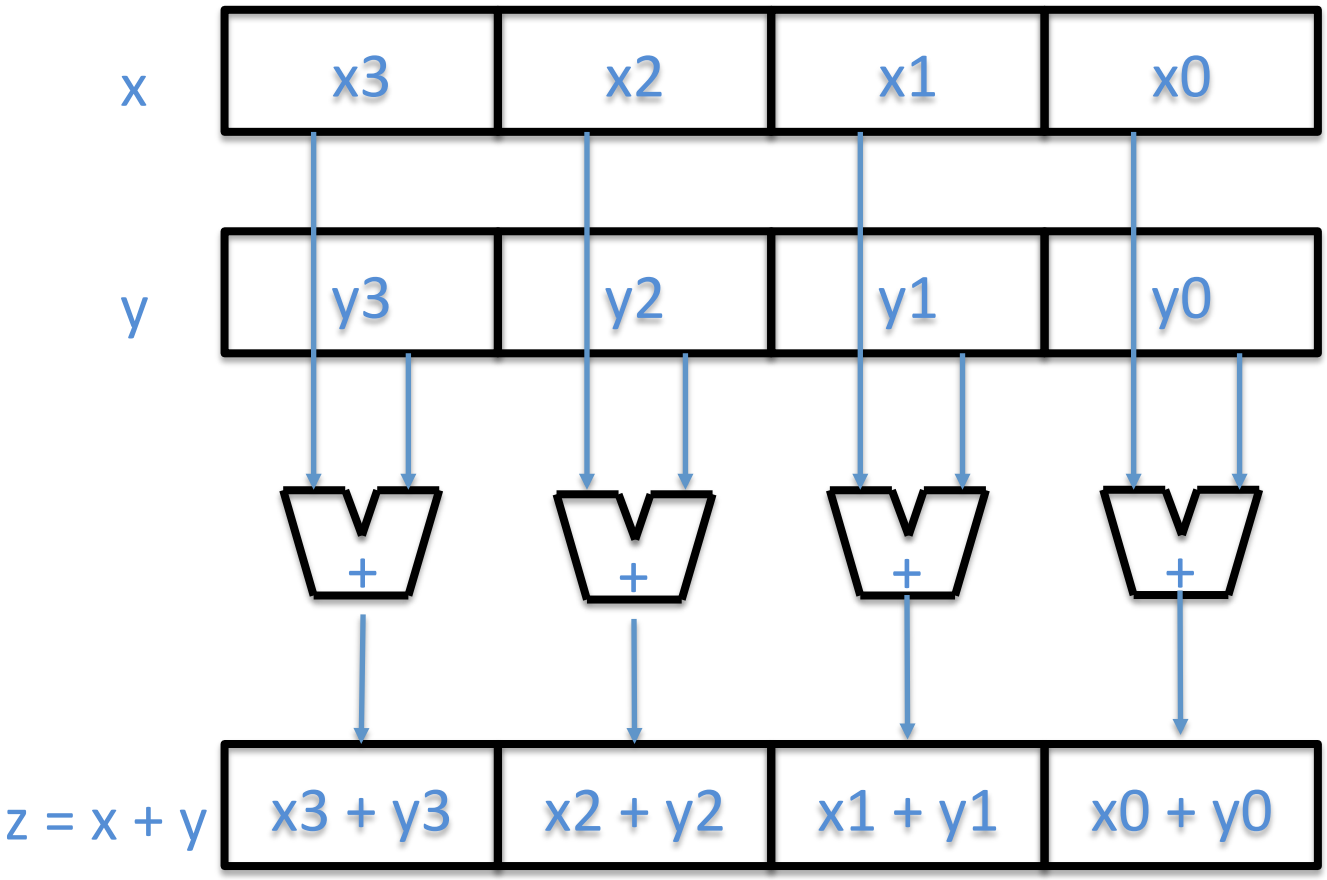
Floating-Point Unit

- Performs arithmetic operations on floating-point numbers
- It is a very complex (big) unit
- Uses a separate register file
  - Usually have instructions to move from one register file to the other
- IEEE 754 specifies 4 formats
  - Single precision (32 bits)
  - Double precision (64 bits)
  - Single-extended precision (≥ 43 bits)
  - Double-extended precision (≥ 79 bits)

SIMD

- Single Instruction Multiple Data
- Operates on SIMD registers
- In the past, SIMD machines = vector machines
  - Vectors of hundreds of elements from/to memory
- Recent SIMD machines operate on short vectors
- SIMD extensions
  - x86 MMX, SSE, AVX
  - Power AltiVec

SIMD Unit



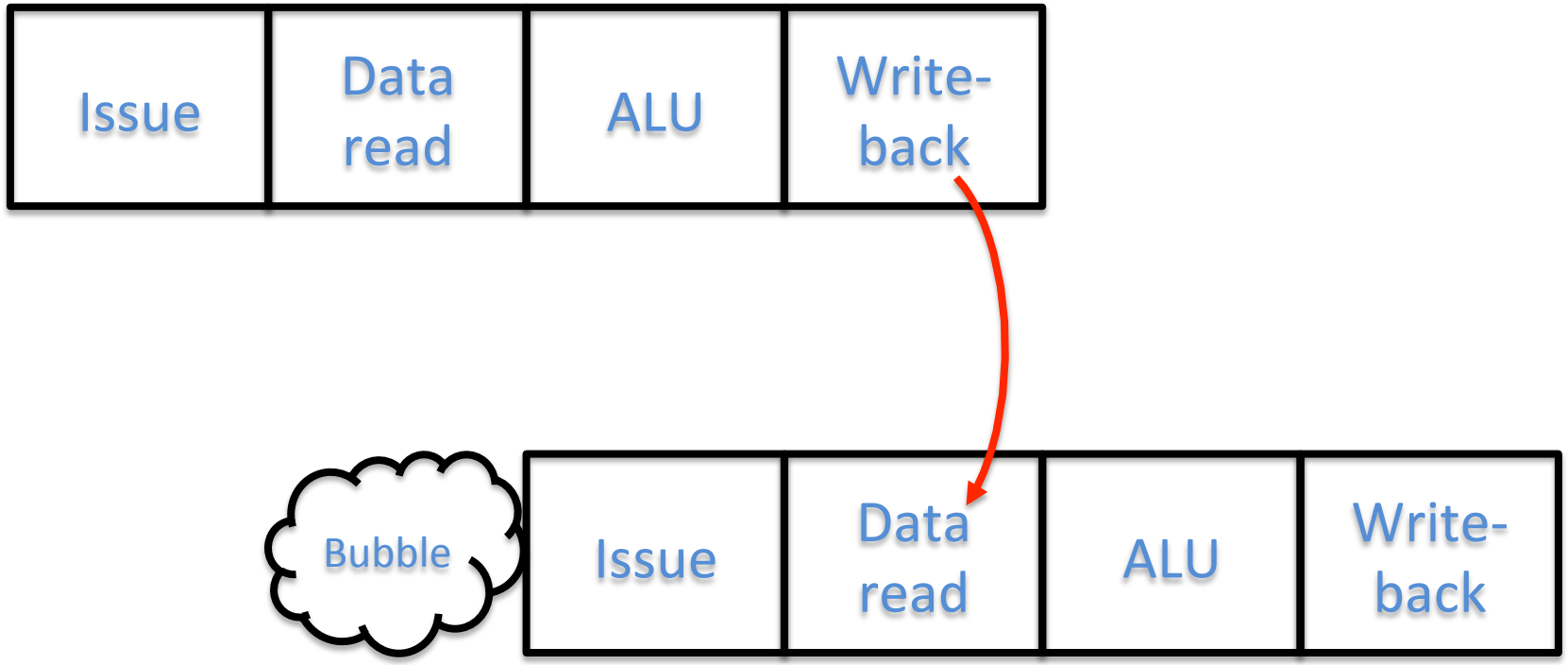
Operand sizes for 128-bit registers

- 16 bytes
- 8 words (16 bits)
- 4 doublewords (32 bits)
- 2 quadword (64 bits)
- 4 single-precision FP (32 bits)
- 2 double-precision FP (64 bits)

Implementation

- To reduce hardware size, processor can have lanes
  - One lane: executes all operations at once
  - Two lanes: split operations into two cycles
- Different operations can have different lane configurations

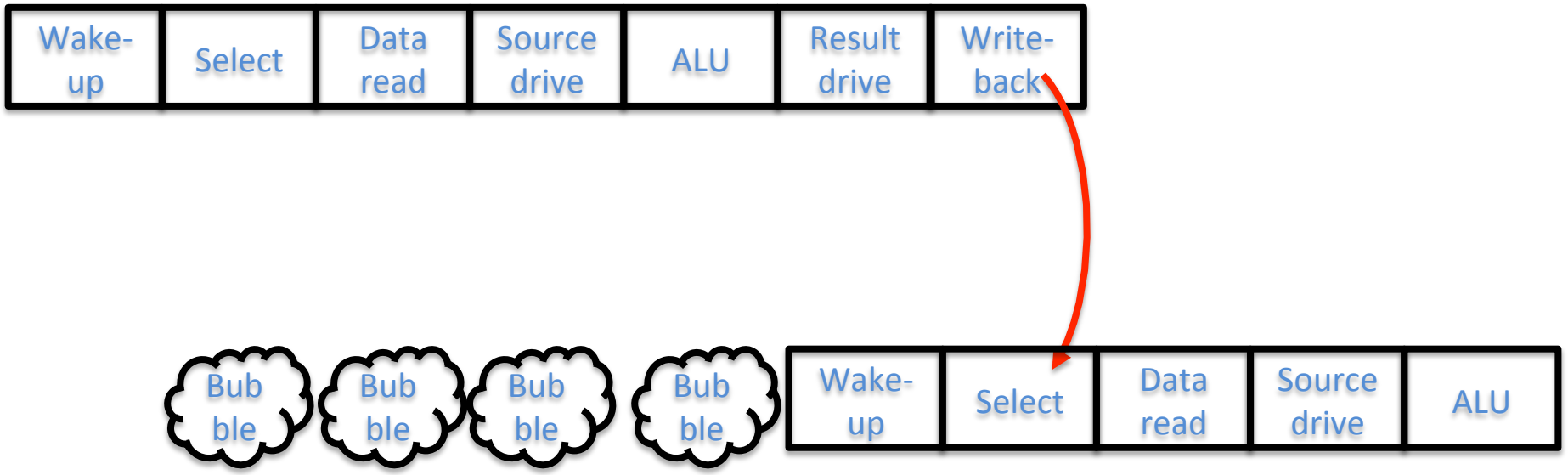
Result Bypassing



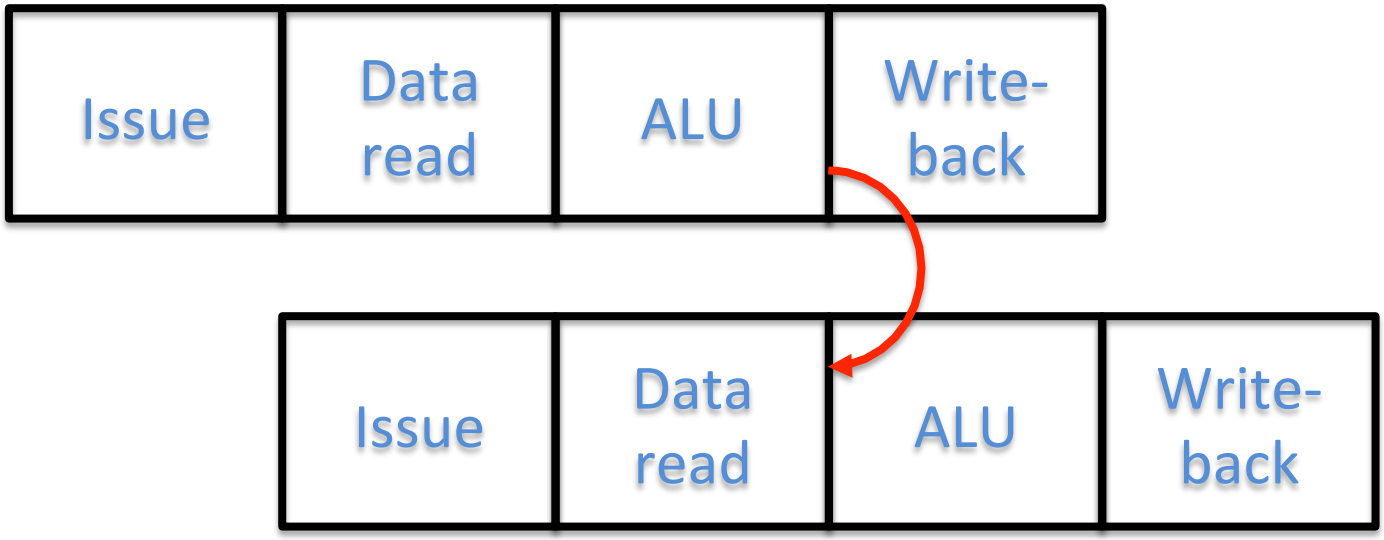
Bypassing

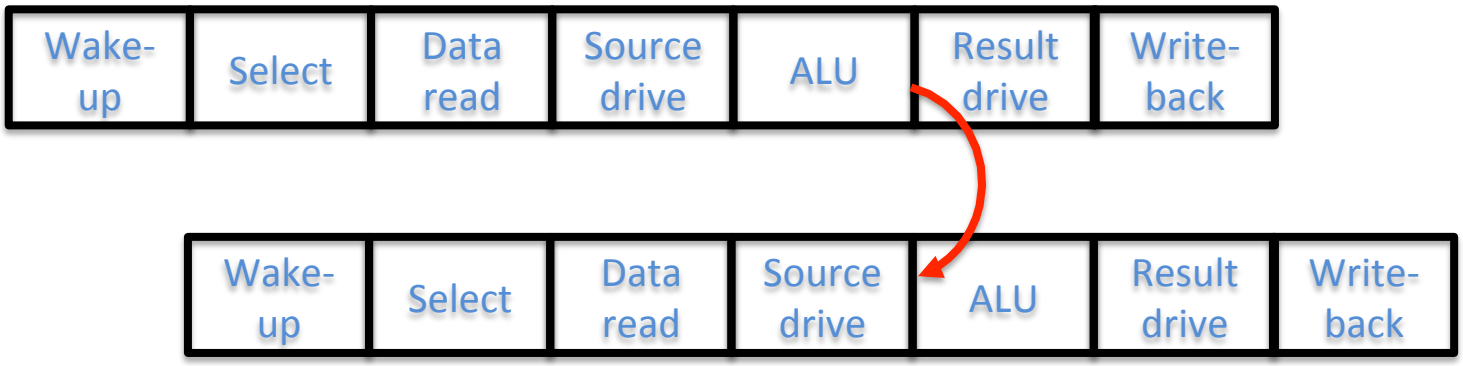
- Results from a computation can be used speculatively after the write-back stage
- Write-back stage and Data read can share the same cycle in several processors
- The bigger the pipeline (higher frequency), the higher is the cost of waiting instructions
- Compiler can alleviate some of this problems by interleaving instructions
- In-order pipelines suffer more than out-of-order

Deeply pipelined processor



Improving performance

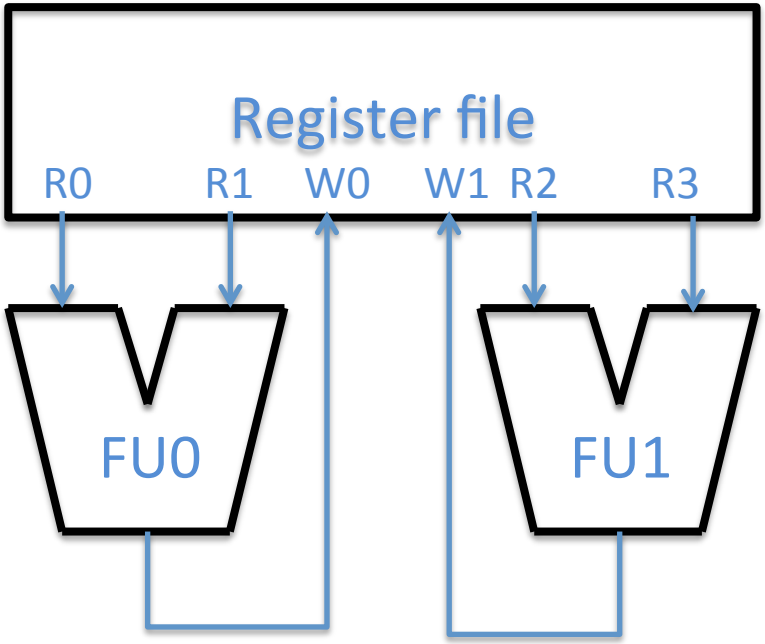


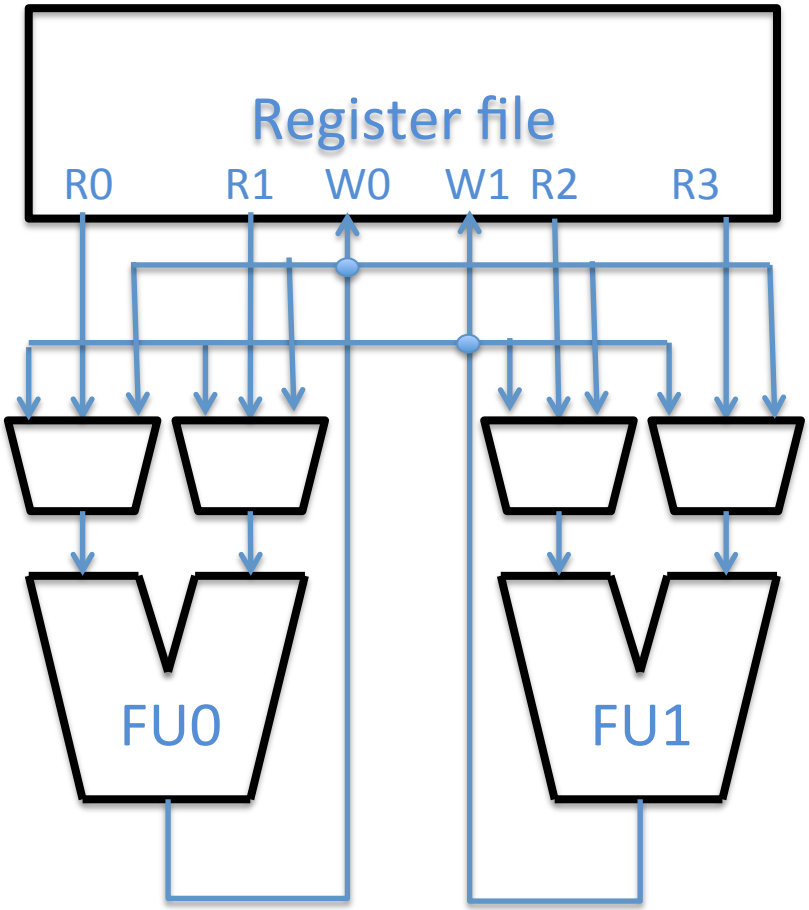


Complexity

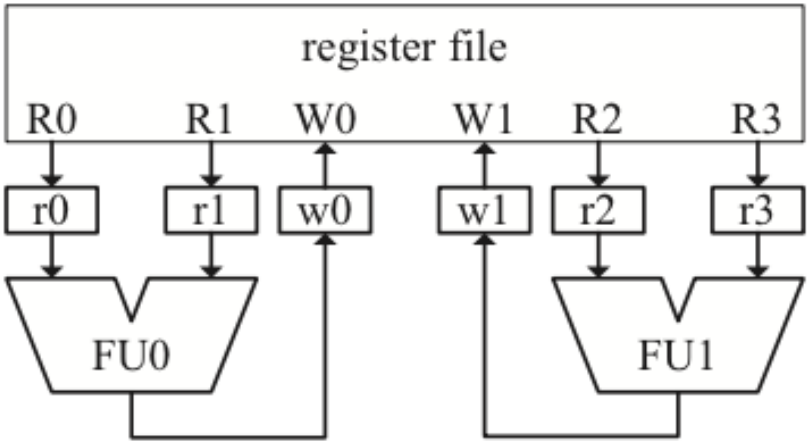
- Bypass lines grow as the number of sources and destinations grow
- Affects area, power critical path and physical layout
- Improves IPC but may reduce cycle time
- Some machines reduces the bypass network as a tradeoff instead of using a full bypass network

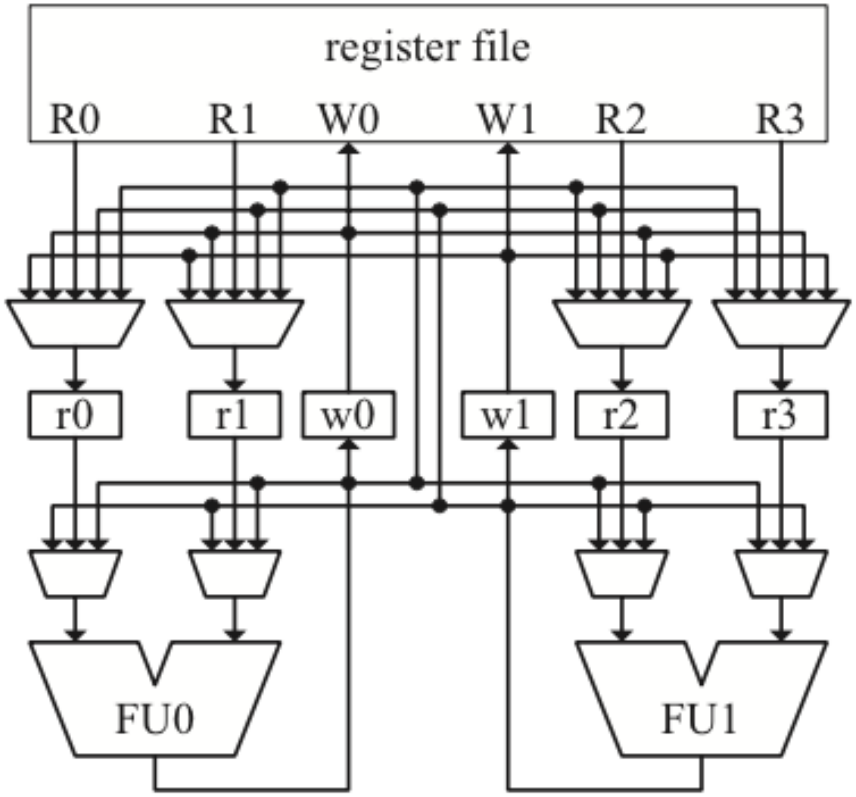
Basic implementation



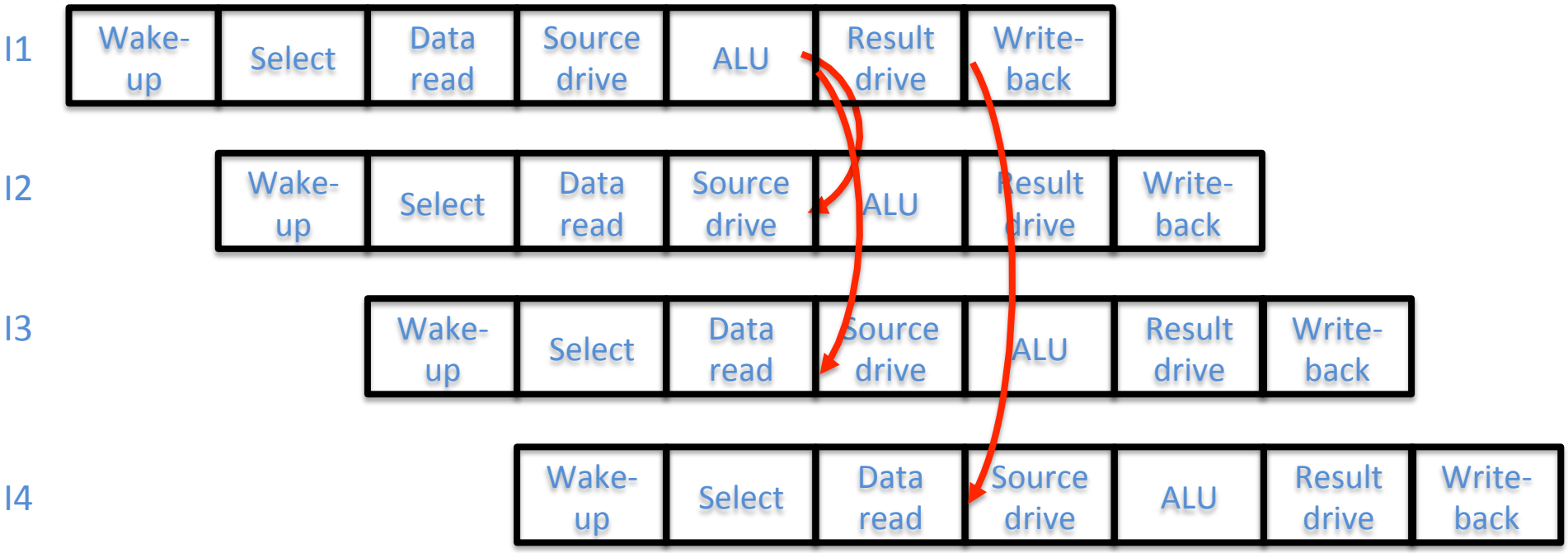


Deeply pipelined execution engine with bypass





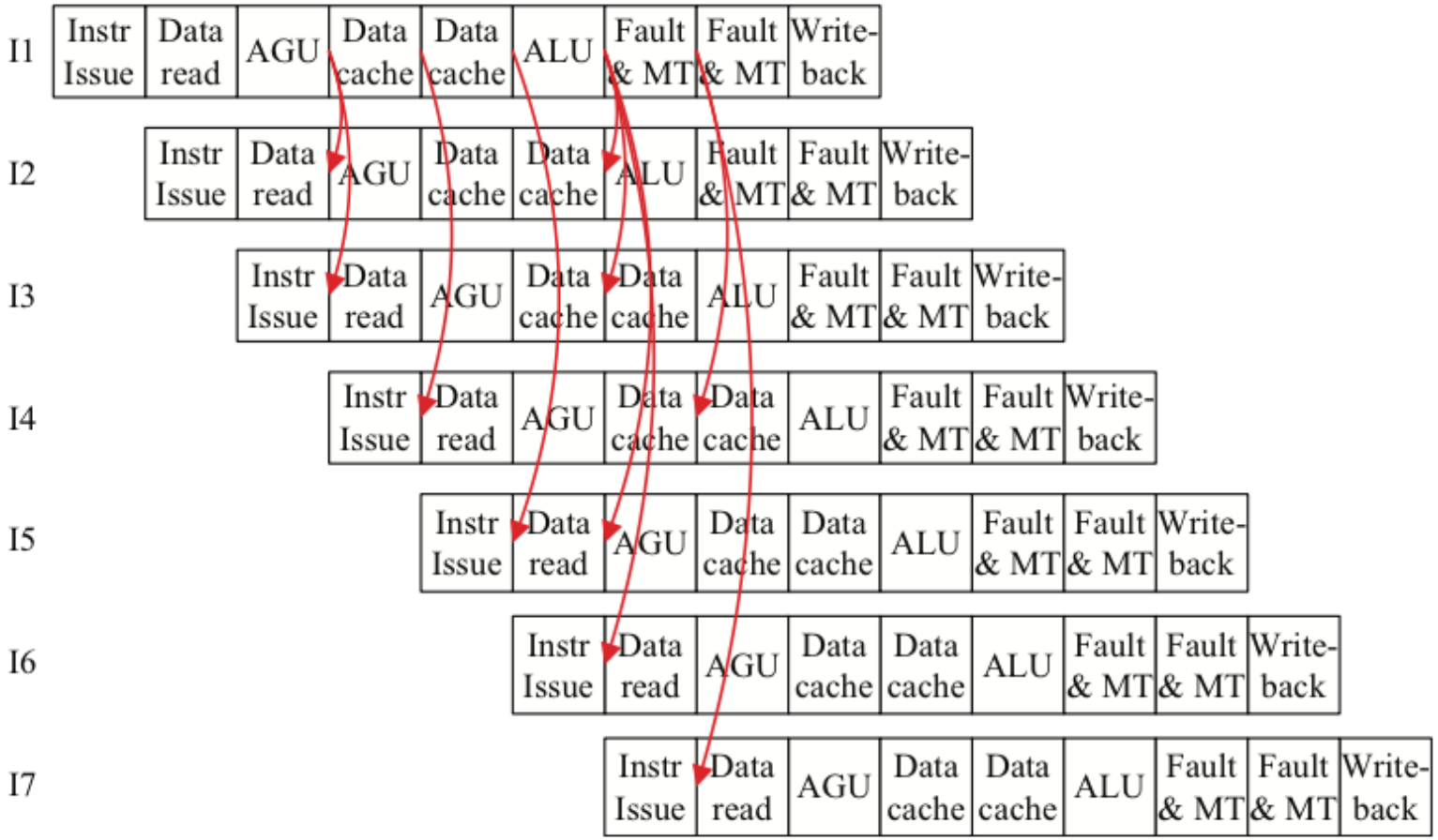
Bypass in the pipeline



In-order Processors

- Due to timing constraints, instructions that finish first may have to wait several cycles for the previous instruction still running
  - Several staging latches are necessary
  - Need forward from those staging latches to all FU
- Complexity can be higher than Out-of-Order pipeline

Possible Intel Atom in-order execution engine



Complexity

- In-order implementation will require lots of staging registers
- Each register may walk through the pipeline
- As an alternative approach, processors create a SRF (staging register file) that contains all staging registers
  - Very similar to ROB for data
- It is very difficult, in modern processors, to bypass results from any FU to any other.
  - The bypass network would grow too large
  - Not all FU share bypass: floating-point, SIMD

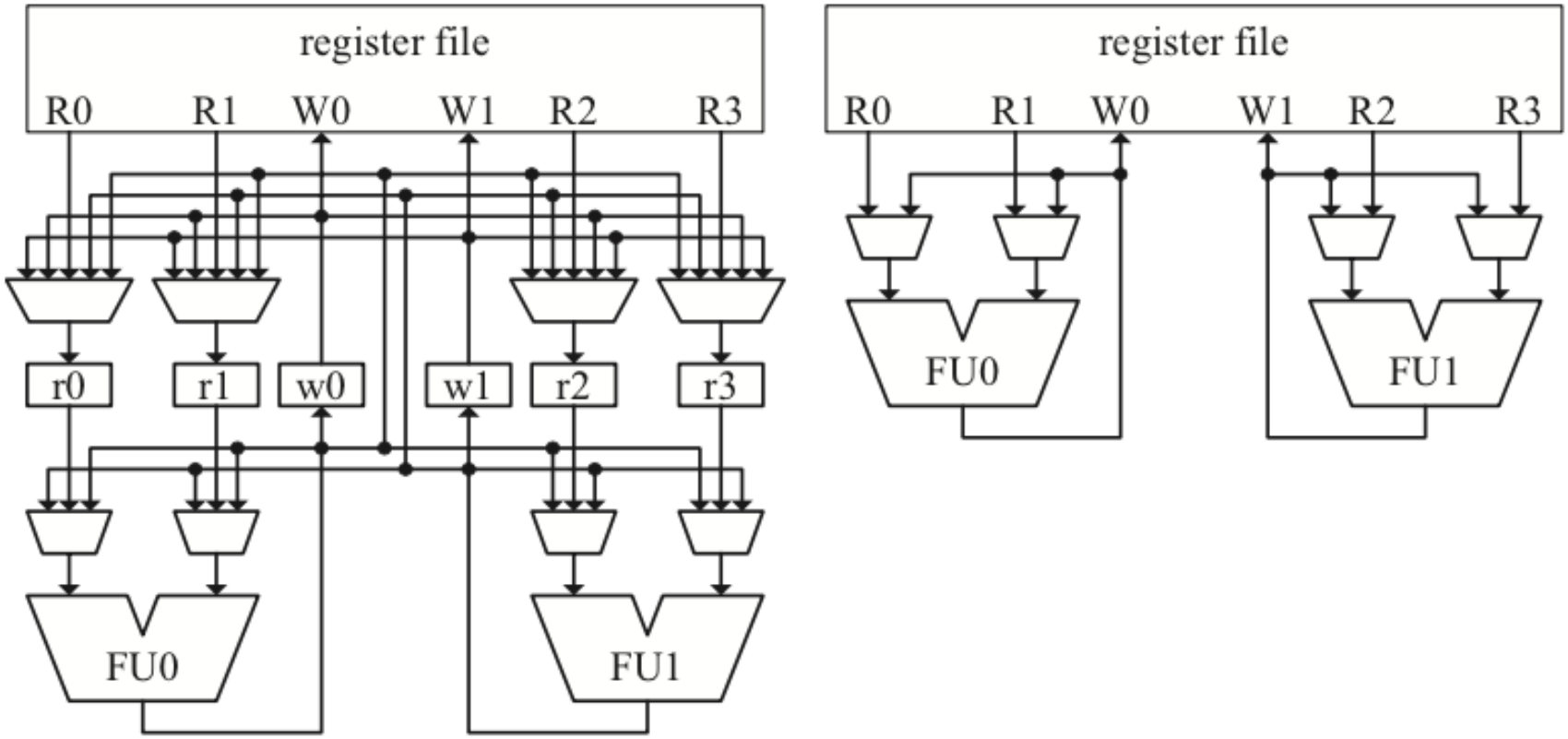
Clustering

- Architectures have evolved and increased in complexity
- Power, temperature, wire length restrict processor growth
- An effective approach to minimize this problem is to partition the hardware
  - Replicate arrays in caches
  - Divide register files, issue queues and bypass network

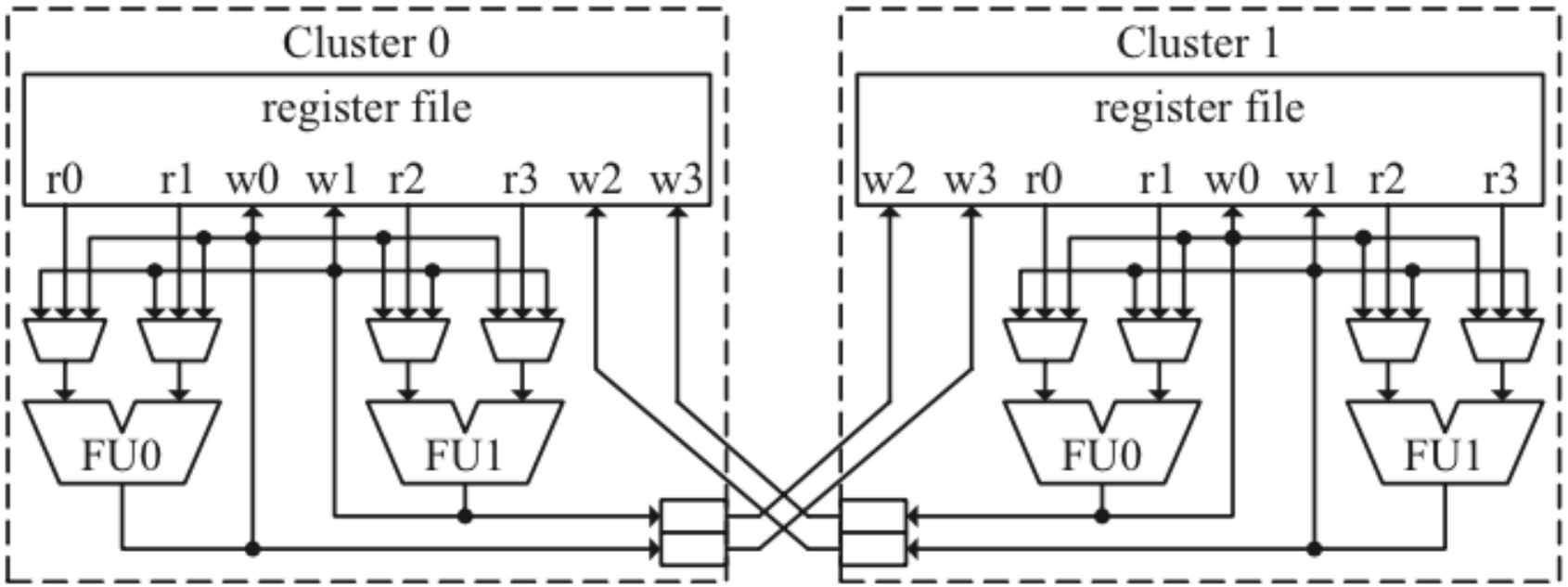
Clustering the Bypass Network

- Restricts the FUs that can receive data from other FUs
- The simpler approach is to allow only FUx to receive data from FUx.
- This approach can reduce one pipeline stage from the execution core
  - In complex processors, bypassing can use one pipeline stage

Clustering the bypass network



Clustering with Replicated Register Files



Clustering with Distributed Issue Queue and Register Files