Multiple Instruction Issue and Completion per Clock Cycle Using Tomasulo's Algorithm – A Simple Example

Assumptions:

- The processor has in-order issue but execution may be out-of-order as it is done as soon after issue as operands are available. Instructions commit as they finish execution.
- There is no speculative execution on Branch instructions because out-of-order completion prevents backing out incorrect results. The reason for the out-of-order completion is that there is no buffer between results from the execution and their commitment in the register file. This restriction is just to keep the example simple.
- It is possible for at least two instructions to issue in a cycle.
- The processor has two integer ALU's with one Reservation Station each. ALU operations complete in one cycle.
- We are looking at the operation during a sequence of arithmetic instructions so the only Functional Units shown are the ALU's.

NOTES: Customarily the term "issue" is the transition from ID to the window for dynamically scheduled machines and dispatch is the transition from the window to the FU's. Since this machine has an in-order window and dispatch with no speculation, I am using the term "issue" for the transition to the reservation stations.

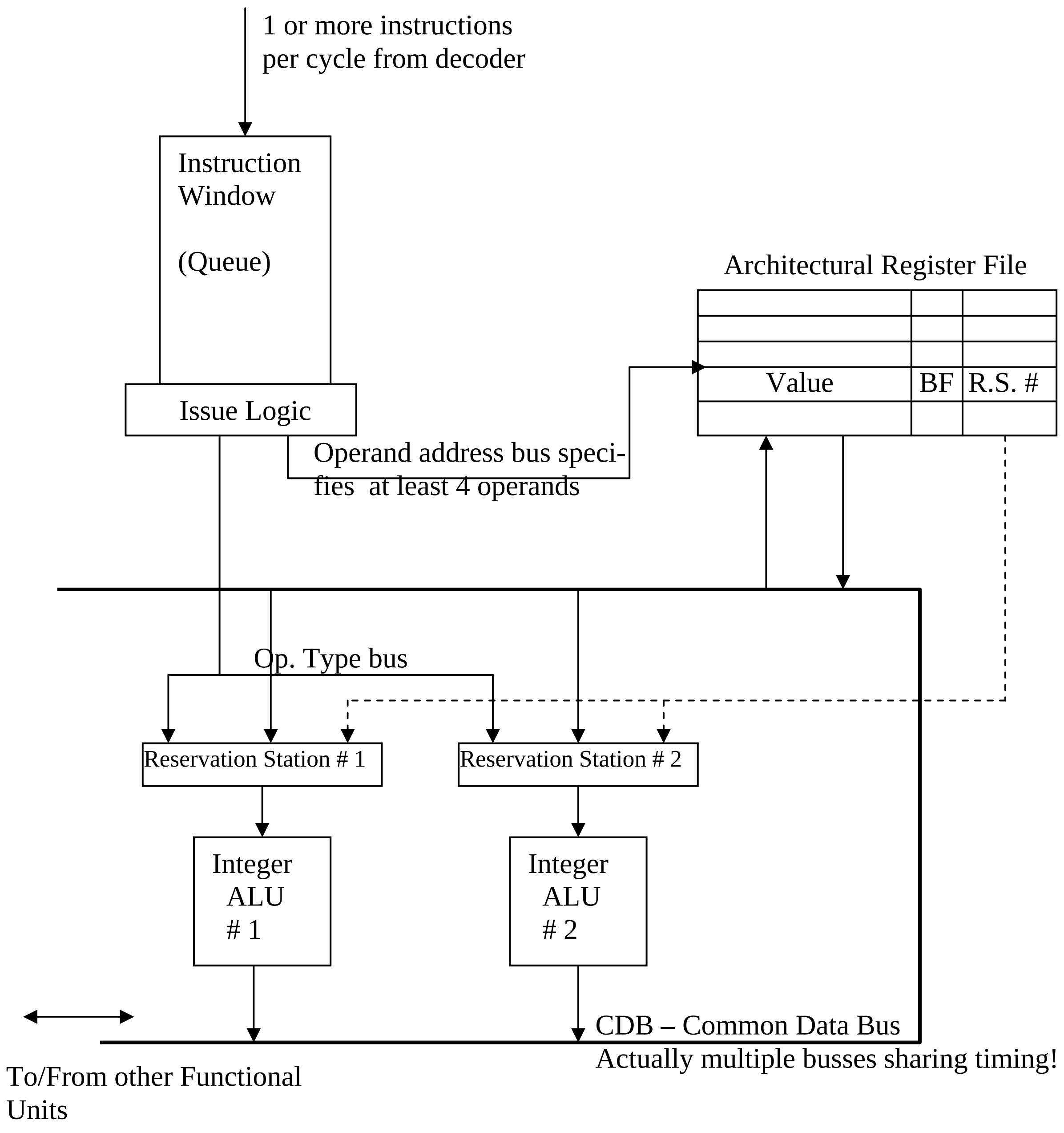
It is possible to have multiple reservation stations in front of a single functional unit. If two instructions are ready for execution on that unit at the same clock cycle, the lowest station number executes.

There is a problem with exception processing when there is no buffering of results being committed to the register file because some register entries may be from instructions past the point at which the exception occurred. For external interrupts this is handled by allowing all current instructions to complete.

Reservation Station Data: Reservation stations are large registers in front of each functional unit. They hold the data for executing an instruction on that unit until all operands are available and the instruction can proceed. They allow the system to avoid data hazards by local storage as soon as an operand is available. That local storage is equivalent to register renaming. If an operand is not available when the instruction is moved into the RS, the BF flag in the register file is set and it means that another functional unit is producing a new value from an earlier instruction. The RS number of that source station is copied from the RS number field of the register file into the RS # field of this reservation station. The value is used by the RS register itself to identify and grab the operand when it appears on the CDB. This is equivalent to data forwarding that we saw in the simple single-issue pipeline.

RS Fields:

Operation Type	Busy Flag	Value of Operand 1	RS # of Opr. 1	Value of Operand 2	RS # of Opr. 2


1 or more instructions per cycle from decoder
Instruction Window
(Queue)
Issue Logic
Architectural Register File
Value
BF
R.S. #
Operand address bus specifies at least 4 operands
Op. Type bus
Reservation Station # 1
Reservation Station # 2
Integer ALU # 1
Integer ALU # 2
CDB – Common Data Bus
Actually multiple busses sharing timing!
To/From other Functional Units

Block Diagram of a Simple Multiple-Issue Processor: Uses Tomasulo's algorithm to avoid data hazards. For simplicity only the integer ALUs are shown.

NOTES: 1. The BF field in the register file is a flag that indicates the value in the register is no longer valid since an instruction is in the process of computing or loading a new one.

2. The RS (Reservation Station) number field is the number of the reservation station that will produce the next value for this register entry. The value fields of the register file and the reservation stations will be updated and the BF flag cleared by logic within the register file when the reservation station of that number puts a datum on the CDB.

3. Only the integer ALUs are shown for simplicity. Even these may be pipelined, further complicating the process of determining when execution completes and instructions commit.

Tomasulo's Algorithm Example with Two Integer Execution Units

The instruction window:

Issue Cycle	Reserv. Station	Commit Cycle	Instruction	Hazard
4	2	5	SUB R1, R6, R7	WAW
4	1	5	ORR R9, R1, R2	RAW
3	1	4	AND R1, R10, R8	WAR
3	2	4	ADD R3, R3, R1	
2	1	3	EOR R4, R8, R3	
1	2	3	SUB R9, R4, R8	RAW
1	1	2	ADD R4, R3, R2	

↑ Execution Order

First cycle:

ADD R4 → RS1, R4 marked BUSY in register file with RS # 1 as its source field; values of R3, R2 sent to RS.

SUB R9 → RS2, R4 marked invalid at RS by putting RS#1 into first operand field, R9 marked BUSY with source RS #2, R8 sent to second operand of RS2.

Old stuff completes

Second cycle:

EOR R10 → RS1, R10 marked BUSY, contents of R8 & R3 copied to RS1

ADD R4 completes, R4 written back to register file and to RS2 as operand for SUB R9 in RS2 does nothing for lack of operand – RAW observed;

Third cycle:

AND R1 → RS1, R1 marked BUSY, R10 and R8 to RS1

ADD R3 → RS2, R3 marked BUSY, R3 and R1 to RS2 WAR avoided by in-order issue

EOR R4 completes, R4 written back and BF cleared

SUB R9 completes, R9 written back and BF cleared

Fourth cycle:

ORR R9 $\rightarrow$ RS1 with operand R2; gets operand R1 from CDB because register file entry marked busy. RAW fixed by grabbing the correct value from CDB

SUB R1 $\rightarrow$ RS2 with operands R6, R7; R1 set BUSY immediately, WAW fixed by grabbing R1 from CDW

ADD R3 completes, R3 not BUSY

AND R1 completes, R1 remains BUSY because SUB R1 operation in RS

Fifth cycle:

ORR R9 and SUB R1 complete and presumably new instructions enter reservation stations.

Assuming two old instructions completed in the first cycle, there are a total of 7 instructions completed in 4 clock cycles with two in the pipeline for the next succeeding cycle. IPC = 1.75.

Adding Branch Speculation to a Processor with Multiple Instruction Issue and Completion per Clock Cycle Using Tomasulo's Algorithm

The Use of a Reorder Buffer

Assumptions:

- Instructions issue in-order but execution may be out-of-order as it is done as soon after issue as operands are available.
- Speculation or speculative execution is the issue and execution of instructions past a conditional branch before the branch condition and address have been confirmed. To allow speculation, the instructions must commit in-order. A Reorder Buffer (ROB) is added to the figure for our simple example processor to allow this.
- With some modification for how the IF unit branch target address is speculated, the same system works for branches with calculated addresses as is typical of subroutine or function calls.

The Reorder Buffer: The ROB is a small multi-ported SRAM that holds the results of completed computation from the execution units until it is safe to commit them to the architectural register file. As each instruction issues, the dispatch unit assigns an address for an entry in the ROB where the result of the instruction will go if it cannot commit yet. Addresses are assigned sequentially beginning with the first location in the ROB that is not occupied. This address is used on the Common Data Bus (CDB) to route the output of the functional unit on which that instruction executes to the ROB.

The addresses of assignments in the ROB are in program order, that is, the order in which the instructions issue. The results move from the ROB into the register file in address order and hence all instructions complete in program order. If a branch is mispredicted, then all instructions left in the ROB and reservation stations when the branch instruction location in the ROB is the next one ready to commit are marked invalid and are never allowed to change memory locations or architectural register file values.

The ROB has two address pointers maintained by its control logic. The first address points to the oldest instruction result that has not yet committed. Each cycle, if this instruction has its result in the ROB, then it and as many sequential instruction results as are ready are transferred into the register file. This commit address pointer is then updated freeing space for new instructions. At the same time, the Result Valid flags for the ROB locations that have committed are cleared.

The second address points to the first unused entry in the ROB. It supplies the dispatch unit with the necessary information on where in the ROB the results from the next issuing instructions are to be placed. Should this pointer catch up with the commit pointer, the ROB is full and the pipeline must stall at the dispatch unit. The both pointers always increment and the addresses wrap around to the bottom when they reach the top.

The figure below shows the position of the ROB in our example processor. Notice that all of the store data and addresses go through the ROB. This is because the address is calculated in the Load/Store address unit and may be ready before the operand that is to be stored. That operand will be placed in the ROB along with the address and the actual write can only happen when the store instruction is ready to commit and has valid values for both its memory address and the operand to be written. When register file values for an issuing instruction are not available, the source operand field is filled with the ROB address of the required result rather than the functional unit address.

Data in a Reorder Buffer Line:

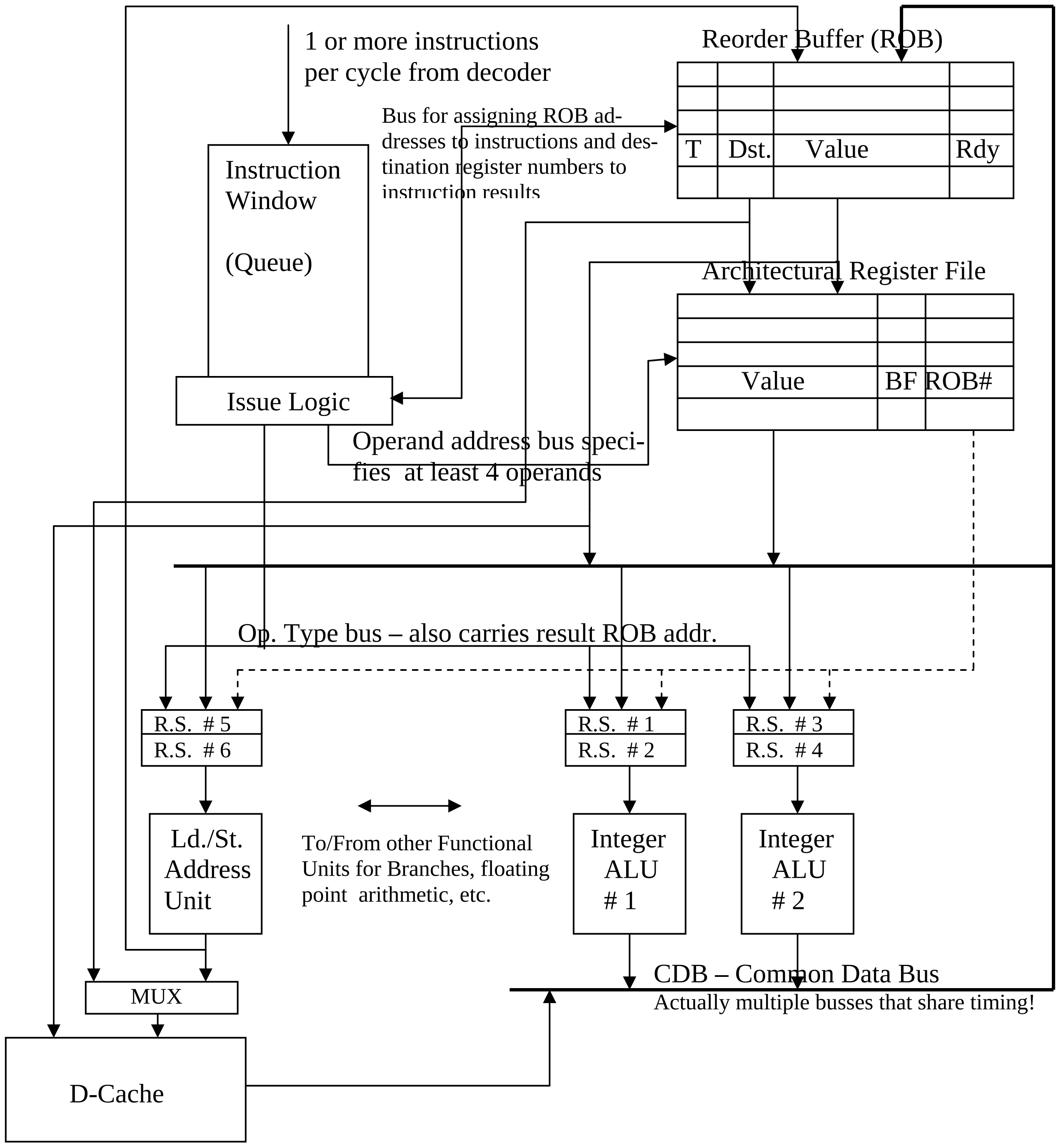
Operation Class	Destination: reg. # or store address	Value of instruction Result	Result Valid

NOTES:
1. The operation class field shows whether the instruction assigned to this location is a branch instruction, an arithmetic or logical instruction, a store instruction, or a load instruction. (For a branch instruction, the value field may not be used.)
2. The destination field holds the target register number for arithmetic and load instructions. It holds the destination address calculated in the load/store address unit for store operations. Store operations cannot be written to memory until the instruction itself reaches the stage in the ROB operation that it is free to commit.
3. The value field holds the instruction result for arithmetic and load operations. For store operations it holds the value to be written to memory.
4. The Valid Result field is a flag bit that signals when the other fields are all correct. Normally set when the value field is written.

RS Fields:

Operation Type	Busy Flag	Value of Operand 1	ROB Address of Operand 1	Value of Operand 2	ROB Address of Operand 2	ROB Address for result

NOTES:
1. There is only one change to the first six fields that were used in our processor with no speculative execution. The change is from reservation station number to ROB addresses for the addresses of operands that are not loaded from the register file because the register file entry is “Busy.” (“Busy” means invalid because a new value is being calculated in another reservation station/functional unit.)
2. The ROB result address field is an addition to place the result into program order when the calculation is finished. This is the ROB address assigned to the instruction itself at the time it issued.
3. The RS for the load/store address unit may have an additional reservation station field to hold the immediate offset for the address calculation from the appropriate field in the instruction itself.


1 or more instructions per cycle from decoder
Reorder Buffer (ROB)
Bus for assigning ROB addresses to instructions and destination register numbers to instruction results
T
Dst.
Value
Rdy
Instruction Window
(Queue)
Architectural Register File
Value
BF
ROB#
Issue Logic
Operand address bus specifies at least 4 operands
Op. Type bus – also carries result ROB addr.
R.S. # 5
R.S. # 6
R.S. # 1
R.S. # 2
R.S. # 3
R.S. # 4
Ld./St. Address Unit
To/From other Functional Units for Branches, floating point arithmetic, etc.
Integer ALU # 1
Integer ALU # 2
CDB – Common Data Bus
Actually multiple busses that share timing!
MUX
D-Cache

Simplified Block Diagram of a Multiple Issue – Multiple Commit Processor with Branch Prediction and Speculative Execution: Based on combining Tomasulo's algorithm with a reorder buffer. Instructions issue in program order, execute whenever they have operands (generally out of order) and commit in order.

NOTES:

1. I put two reservation stations per functional unit to illustrate that one can do this with only the penalty of complexity in the reservation stations themselves. Some

arbitration to assure that instructions cannot get stuck here is the only necessary change to the basic system.

2. Note that I have left out the functional units for Branch address calculation and condition testing. That belongs in parallel with the ALUs but also has direct control lines up to the instruction fetch stage and the ROB and register file control blocks.
3. Frequently the execution functional units themselves are pipelined even for fixed point arithmetic operations.
4. Floating point units can also go in the same spot as the fixed point ALUs but can also be treated as a coprocessor unit as in ARM's superscalar processors. Coprocessors themselves have structures similar to this figure.