Dynamic Execution Core

CS6600: Computer Architecture

Arjun Menon V, Akilesh Kannan
EE18B104, EE18B122

The code developed for this assignment can be found here - link

1 Introduction

Superscalar Out-of-Order (OoO) processors arise as a natural extension of the scalar pipelined processor architecture. Superscalar OoO cores, which enable dynamic execution of instructions in-flight, play a key role in reducing the Cycles per Instruction (CPI) metric of a pipelined processor. These processors can hence assume effective CPI values less than 1 as they process multiple instructions per cycle ($IPC > 1$).

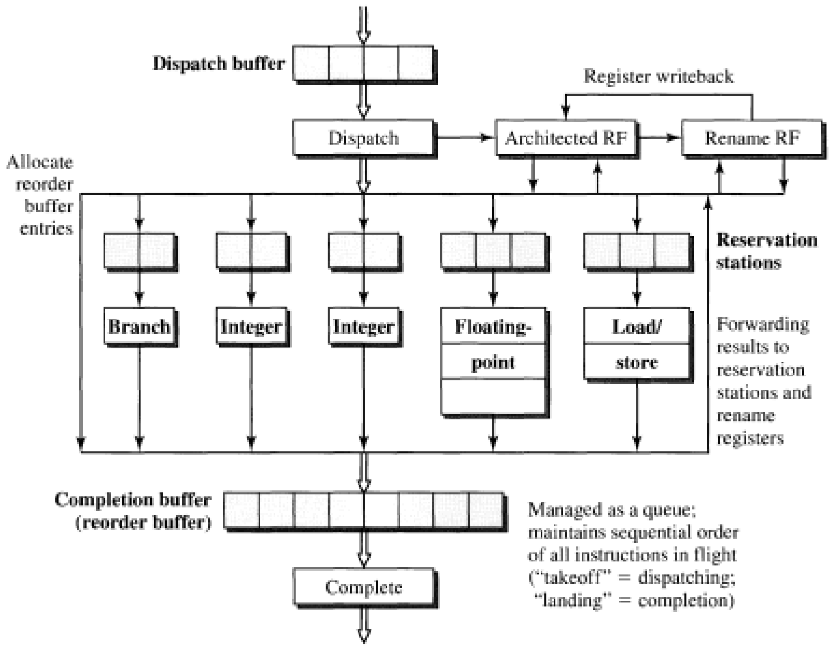
Dispatch buffer
Register writeback
Dispatch
Architected RF
Rename RF
Allocate reorder buffer entries
Reservation stations
Branch
Integer
Integer
Floating-point
Load/ store
Forwarding results to reservation stations and rename registers
Completion buffer (reorder buffer)
Managed as a queue; maintains sequential order of all instructions in flight ("takeoff" = dispatching; "landing" = completion)
Complete

Fig. 1: Dynamic Execution Core in a Superscalar OoO Processor [1]

Executing instructions in an order different from the original program order can lead to unexpected results due to dependencies among instructions (data-, name- and anti-dependencies). To enforce instruction dispatch (into the Functional Units) and instruction completion (to the `architectural register file`) in program order, additional buffers- `dispatch buffer` and `reorder buffer` respectively- are added to the pipeline. Further, to avoid errors due to name- and anti-dependencies (false dependencies), a separate `rename register file (RRF)` is used by the instructions to read from and write to when they are in-flight. When an instruction is completed by the `reorder buffer`, the data in the `RRF` is written back to the `architectural register file (ARF)`. Each Functional Unit includes a `reservation station (RS)`; instructions are issued to the Functional Units depending on the availability of the operands.

2 Dynamic Execution Core Simulator

We implement a python-based Dynamic Execution Core Simulator with specialised Functional Units for Add/Store, Multiply, Divide and Load/Store instructions. An efficient implementation of branch instructions in the design would require a robust Branch Prediction Unit. As our focus here is dynamic execution of instructions, we leave out branch instructions from our design. The instruction set used by the simulator is specified in the problem statement (link).

The functional units for Add/Store, Multiply and Divide are pipelined enabling a maximum throughput of one instruction per cycle regardless of the Functional Unit's latency. Further the reservation stations for these units issue instructions in an Out-of-Order manner depending on the availability of the operands. The Load/Store unit processes instructions in a non-pipelined in-order manner.

The implementation details of rename registers, reservation stations, reorder buffers, functional units and the forwarding unit are described in the following section.

3 Code Structure

- class **regfiles**
  - class *arfEntry*: | data | busy | tag |
  - class *arf*: Architecture register file
  - class *rrfEntry*: | data | busy | valid |
  - class *rrf*: Rename register file
  - **destinationAllocate()**: perform renaming for a given ARF register index.
  - **sourceRead()**: return correct data (if available in ARF or RRF) or the appropriate RRF tag, and the validity of the data.
  - **registerUpdate()**: update the data in the RRF upon finishing of instruction during forwarding and also update the ARF upon committing of instruction.
- class **roBuffer**
  - class *roBufferEntry*: | busy | issued | finished | renameReg |
  - **updateState()**: update the head and tail pointers of the ROB.
  - **insertEntry()**: insert a new instruction in the ROB
  - **updateEntry()**: update an instruction's entry on execution finish.
  - **complete()**: commit the head instruction (if possible) and move the head pointer.

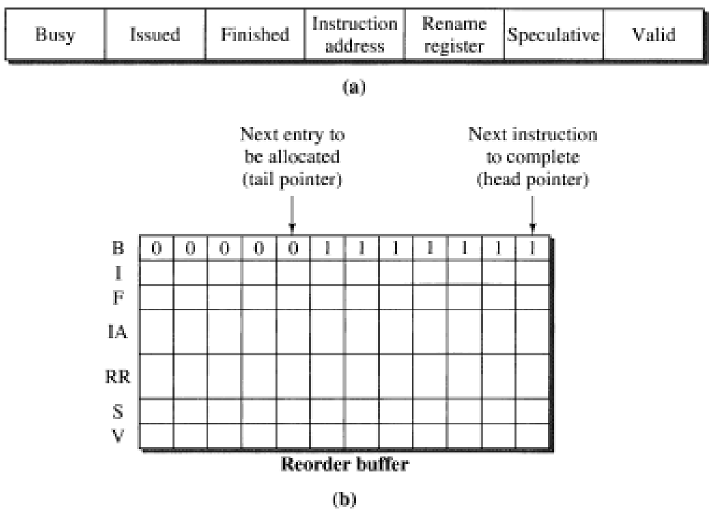

Fig. 2: Reorder Buffer Organisation [1]

- **decode()**: decode a given binary string into appropriate data fields - r1, r2 etc.
- **dispatch()**: dispatch instructions from the dispatch buffer to the appropriate reservation stations. To enforce in-order dispatch, stall if necessary.
- class **reservationStation**
  - class *reservationStationEntry*: | ID | Op1 | valid | Op2 | valid | ready | opcode |
  - **isFull()**: return whether the RS is full or not
  - **addEntry()**: add a new instruction to the reservation station
  - **updateEntries()**: update the data and valid fields of the RS entries during forwarding
  - **putIntoFU()**: push an instruction on the functional unit if it is ready.

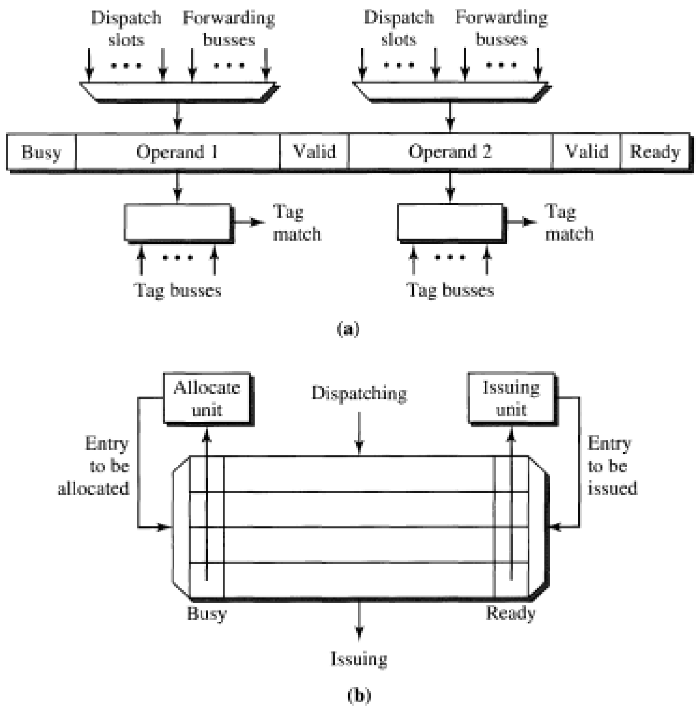

Fig. 3: Reservation Station Organisation [1]

- class **{AS/M/D}U**
  - **shiftAndEval()**: advance the various instructions in the unit to their next stages and return the finished computation data and tag for forwarding.

To ensure that the load/store instructions are processed in-order, we implement slightly different functions and classes to take care of their extra fields in the instruction (3 operands, as opposed to 2) and to enforce in-order execution of the reservation station entries.

- class **LSreservationStation**
  - class *LSrsEntry*: | ID | Op1 | valid | Op2 | valid | offset | ready | opcode |
  - **isFull()**: return whether the RS is full or not
  - **addEntry()**: add a new instruction to the reservation station
  - **updateEntries()**: update the data and valid fields of the RS entries during forwarding
  - **putIntoFU()**: push an instruction on the functional unit if it is ready.
- class **{AS/M/D}U**
  - **shiftAndEval()**: advance the various instructions in the unit to their next stages and return the finished computation data and tag for forwarding.

The config file (conf.py) is at the root of the directory. All necessary configuration parameters are highlighted here.

4 Results

Given are the cycle-by-cycle details provided by the simulator for some sample input test cases.

The functional units implemented in this simulator are pipelined and thus can have multiple instructions simultaneously executed, but in different stages of the pipeline. Thus, the reported cycles do not match with what is given as part of the sample test case provided as part of the problem statement.

Test case	Cycles taken
`ins.txt`	7
`ins1.txt`	17
`ins2.txt`	17
`sample.txt`	27

Tab. 1: Cycles taken for different input test cases[1]

Test Case 1: sample.txt

```
0001000100100011
0010010000100011
0011010100010100
0100011001111000
```

```
------------- Cycle 0 start -------------
Decoding Instructions...
Decoded 4 new instructions
Dispatching Instructions...
instruction(inst='ADD', fu='ASU', r1=1, r2=2, r3=3)
instruction(inst='SUB', fu='ASU', r1=4, r2=2, r3=3)
Dispatched 2 instructions to respective RS
Issuing instructions to FUs...
        To ASU: [0, 'ADD', 1, 1]
        Update RoB from ASU :(issued) index =  0
Issued 1 non-NOP instructions
Finishing Instructions...
Finished 0 instructions
Forwarding FU outputs:
        From ASU: tag = None | data = None
        From MU:  tag = None | data = None
        From DU:  tag = None | data = None
        From LSU: tag = None | data = None
Committing ROB head...
Reorder Buffer:         Head:0         Tail:2
------------- Cycle 0 end   -------------
------------- Cycle 1 start -------------
Decoding Instructions...
Decoded 0 new instructions
Dispatching Instructions...
instruction(inst='MUL', fu='MU', r1=5, r2=1, r3=4)
instruction(inst='DIV', fu='DU', r1=6, r2=7, r3=8)
```

[1] Test cases and corresponding output log files are present in the GitHub repository here - link

```
Dispatched 2 instructions to respective RS
Issuing instructions to FUs...
        To ASU: [1, 'SUB', 1, 1]
        Update RoB from ASU :(issued) index =  1
        To DU: [3, 'DIV', 1, 1]
        Update RoB from DU :(issued) index =  3
Issued 2 non-NOP instructions
Finishing Instructions...
        Update RoB from ASU: (finished) index =  0
Finished 1 instructions
Forwarding FU outputs:
        From ASU: tag = 0 | data = 2
        From MU:  tag = None | data = None
        From DU:  tag = None | data = None
        From LSU: tag = None | data = None
Committing ROB head...
        Completed head instruction
Reorder Buffer:         Head:1         Tail:4
------------- Cycle 1 end   -------------
------------- Cycle 2 start -------------
Decoding Instructions...
Decoded 0 new instructions
Dispatching Instructions...
Dispatched 0 instructions to respective RS
Issuing instructions to FUs...
Issued 0 non-NOP instructions
Finishing Instructions...
        Update RoB from ASU: (finished) index =  1
Finished 1 instructions
Forwarding FU outputs:
        From ASU: tag = 1 | data = 0
        From MU:  tag = None | data = None
        From DU:  tag = None | data = None
        From LSU: tag = None | data = None
Committing ROB head...
        Completed head instruction
Reorder Buffer:         Head:2         Tail:4
------------- Cycle 2 end   -------------
------------- Cycle 3 start -------------
Decoding Instructions...
Decoded 0 new instructions
Dispatching Instructions...
Dispatched 0 instructions to respective RS
Issuing instructions to FUs...
        To MU: [2, 'MUL', 2, 0]
        Update RoB from MU :(issued) index =  2
Issued 1 non-NOP instructions
Finishing Instructions...
Finished 0 instructions
Forwarding FU outputs:
        From ASU: tag = None | data = None
        From MU:  tag = None | data = None
        From DU:  tag = None | data = None
        From LSU: tag = None | data = None
Committing ROB head...
Reorder Buffer:         Head:2         Tail:4
```

```
------------- Cycle 3 end   -------------
------------- Cycle 4 start -------------
Decoding Instructions...
Decoded 0 new instructions
Dispatching Instructions...
Dispatched 0 instructions to respective RS
Issuing instructions to FUs...
Issued 0 non-NOP instructions
Finishing Instructions...
Finished 0 instructions
Forwarding FU outputs:
        From ASU: tag = None | data = None
        From MU:  tag = None | data = None
        From DU:  tag = None | data = None
        From LSU: tag = None | data = None
Committing ROB head...
Reorder Buffer:         Head:2         Tail:4
------------- Cycle 4 end   -------------
------------- Cycle 5 start -------------
Decoding Instructions...
Decoded 0 new instructions
Dispatching Instructions...
Dispatched 0 instructions to respective RS
Issuing instructions to FUs...
Issued 0 non-NOP instructions
Finishing Instructions...
        Update RoB from MU: (finished) index =  2
        Update RoB from DU: (finished) index =  3
Finished 2 instructions
Forwarding FU outputs:
        From ASU: tag = None | data = None
        From MU:  tag = 2 | data = 0
        From DU:  tag = 3 | data = 1
        From LSU: tag = None | data = None
Committing ROB head...
        Completed head instruction
Reorder Buffer:         Head:3         Tail:4
------------- Cycle 5 end   -------------
------------- Cycle 6 start -------------
Decoding Instructions...
Decoded 0 new instructions
Dispatching Instructions...
Dispatched 0 instructions to respective RS
Issuing instructions to FUs...
Issued 0 non-NOP instructions
Finishing Instructions...
Finished 0 instructions
Forwarding FU outputs:
        From ASU: tag = None | data = None
        From MU:  tag = None | data = None
        From DU:  tag = None | data = None
        From LSU: tag = None | data = None
Committing ROB head...
        Completed head instruction
Reorder Buffer:         Head:4         Tail:4
------------- Cycle 6 end   -------------
```

```
------------- Cycle 7 start -------------
Decoding Instructions...
Decoded 0 new instructions
Dispatching Instructions...
Dispatched 0 instructions to respective RS
Issuing instructions to FUs...
Issued 0 non-NOP instructions
Finishing Instructions...
Finished 0 instructions
Forwarding FU outputs:
        From ASU: tag = None | data = None
        From MU:  tag = None | data = None
        From DU:  tag = None | data = None
        From LSU: tag = None | data = None
Committing ROB head...
        No instruction left to complete
Reorder Buffer:         Head:4         Tail:4
------------- Cycle 7 end   -------------
Number of Cycles elapsed:  7
```

References

[1] Shen, J. P., and Lipasti, M. H. *Modern processor design: fundamentals of superscalar processors*, reissued ed. Waveland Press, Long Grove, Illinois, 2013.