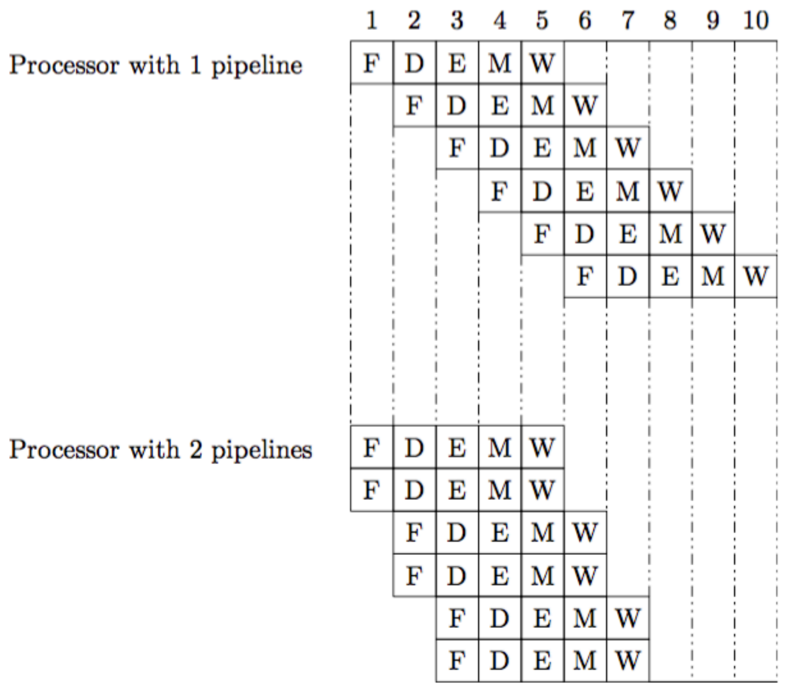
Lecture 6

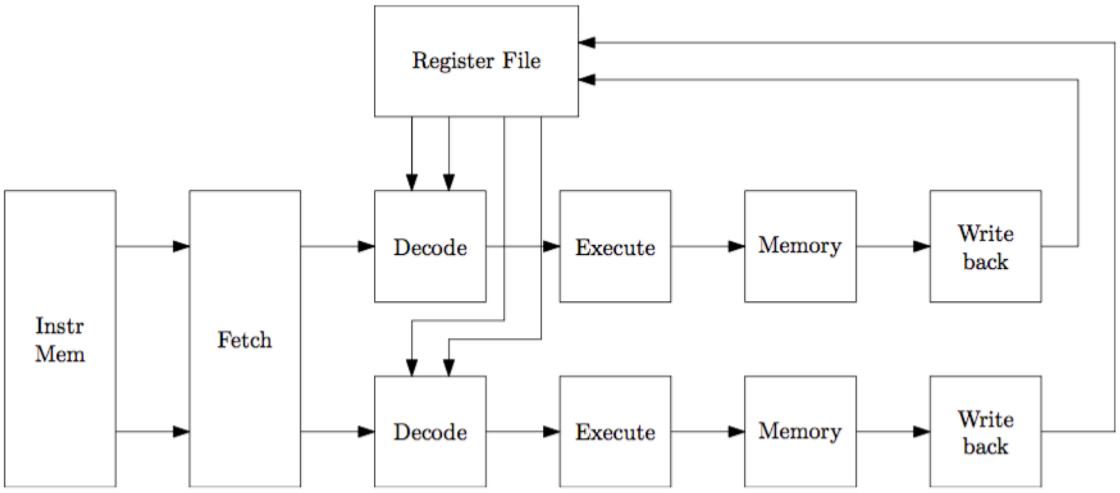
Instruction Level Parallelism

6.1 Superscalar

Superscalar [Wil96] extends the pipelining technique by duplicating the pipeline in a processor. This allows the processor to execute multiple instructions at a time.



Two-way Superscalar Pipeline



Example 6.1 Show how the following program is executed in a processor using *two-way superscalar pipeline* and *data forwarding*.

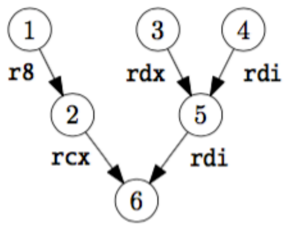
```
irmov  $10,%rax
irmov  $6,%rcx
irmov  $7,%rdx
add    %r8,%rcx
add    %rax,%rdx
add    %rcx,%rdx
mrmov  4(%rdx),%r9
```

6.2 Out-of-order Execution

Only *independent instructions* can be executed in parallel. However, adjacent instructions are usually not independent, and superscalar processors still fetched instructions according to their order in the program. This makes the utilization of the second pipeline become low.

By analyzing the data flow, the instructions can be executed out of their order in the program. We can use a *data flow graph* to show the dependencies by instructions in a program. This graph can also be used to schedule the instruction execution.

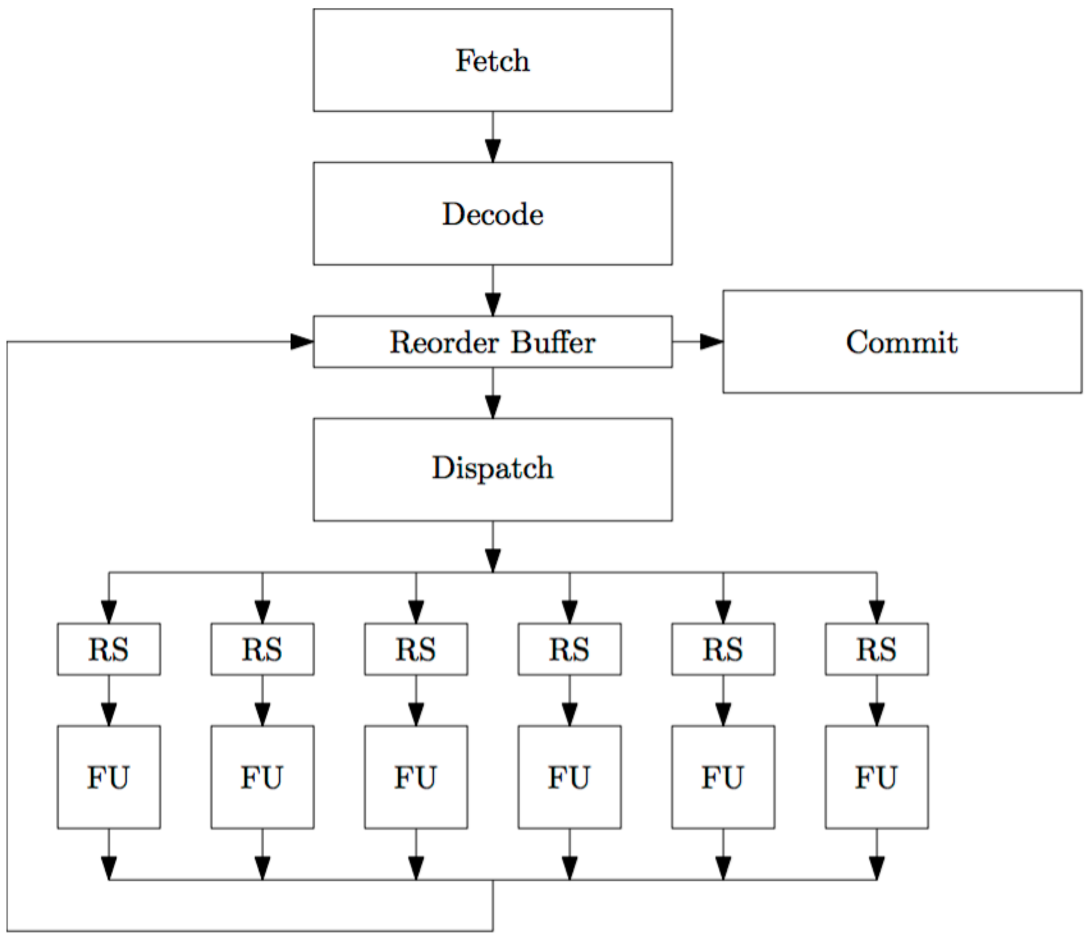
```
(i1) mov 4(%rax), %r8
(i2) add %r8, %rcx
(i3) add $1, %rdx
(i4) mov $8, %rdi
(i5) add %rdx, %rdi
(i6) imul %rcx, %rdi
```



Exercise 6.1 Draw a data flow graph for the following program.

```
irmov  $10,%rax
irmov  $6,%rcx
irmov  $7,%rdx
add    %r8,%rcx
add    %rax,%rdx
add    %rcx,%rdx
mrmov  4(%rdx),%r9
```

To allow *out-of-order execution*, the datapath has been modified as shown below.



Here are how the out-of-order execution works:

1. Instructions are *fetched* and *decoded* according to their orders in the program. The decoded instructions are stored in the *reorder buffer* which is a FIFO list.

2. The *dispatch* unit gets the next instruction from the queue, and sends it to the *reservation stations (RS)* of the *functional units (FU)* according to the instruction type. The instruction state is set to *issue*.

3. An instruction placed in the *reservation station* may wait for its operands. When all the operands are available, the instruction will be executed in the corresponding *functional unit*. The instruction state is changed to *execute*. By doing this, RAW hazards can be avoided.

4. When the result is available, it is written to the *reorder buffer*. The state becomes *write result*.

5. The results in the *reorder buffer* are then committed according to its order in the program. The state finally becomes *commit*.

6. The *fetch* unit uses the *branch prediction* to generate a sequence of instructions. When a conditional jump instruction reaches the head of the *reorder buffer* but the prediction is incorrect, the *reorder buffer* is then flushed and the execution is performed again with the correct branch.

Example 6.2 A processor using the out-of-order execution has three functional units: *add*, *load*, and *multiply*. Each unit is attached with a reservation station with two entries. Show how to execute the following program.

Reorder Buffer

Entry	Instruction	State	Destination	Value
1	`mrmov 4(%rax), %rcx`			
2	`mrmov 8(%rbp), %rdx`			
3	`imul %rcx, %r8`			
4	`add $8, %rsi`			
5	`imul %r8, %rsi`			
6	`add $10, %rdx`			

Reservation Stations

Name	Instruction	Operand1	Operand2	Src1	Src2	Dest
Load1						
Load2						
Add1						
Add2						
Mult1						
Mult2						

References

[Wil96] B. Wilkinson. *Computer Architecture: Design and Performance (2nd edition)*. Prentice Hall, 1996.