Assignment – 9
COL216 – Computer Architecture
Overcoming Data Hazards in a Pipelined Processor

This assignment is an extension of the previous assignment. We overcome the data hazards of the previous assignment by forwarding and bypassing.

There are total five stages of a pipelined processor :

1. IF(Instruction fatch) : This is the first stage of executing an instruction. In this, stage, the processor reads the respective instruction from I-Memory and increase the program counter to 1. after fatching the instrucion, it procced the data to the next stage in next cycle.

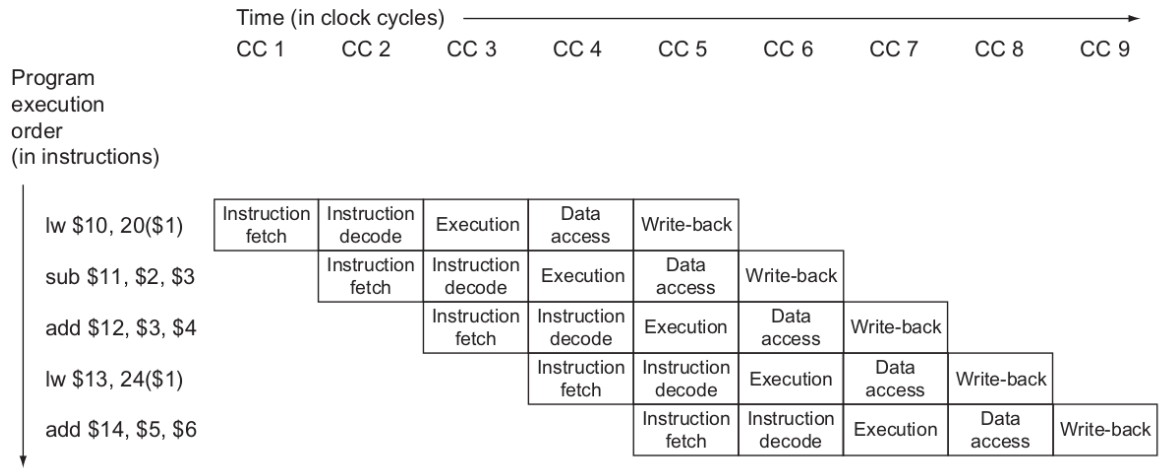
2. ID(Instruction Decode) : the second stage of pipelined processor. It decodes the data, we get from IF stage. It indetifies the type of the instruction, the registers, destination register, offset(if any), and branch target or label(for beq, bne, blez, bgtz).

3. EX(Instruction Execution) : this stage is also called ALU stage because the main work of this stage is to compute. In case of R-type instrucions, it computes the final calculated result that is supposed to be stored in the destination register. We read the values of registers and calculate thier resultant in this stage.

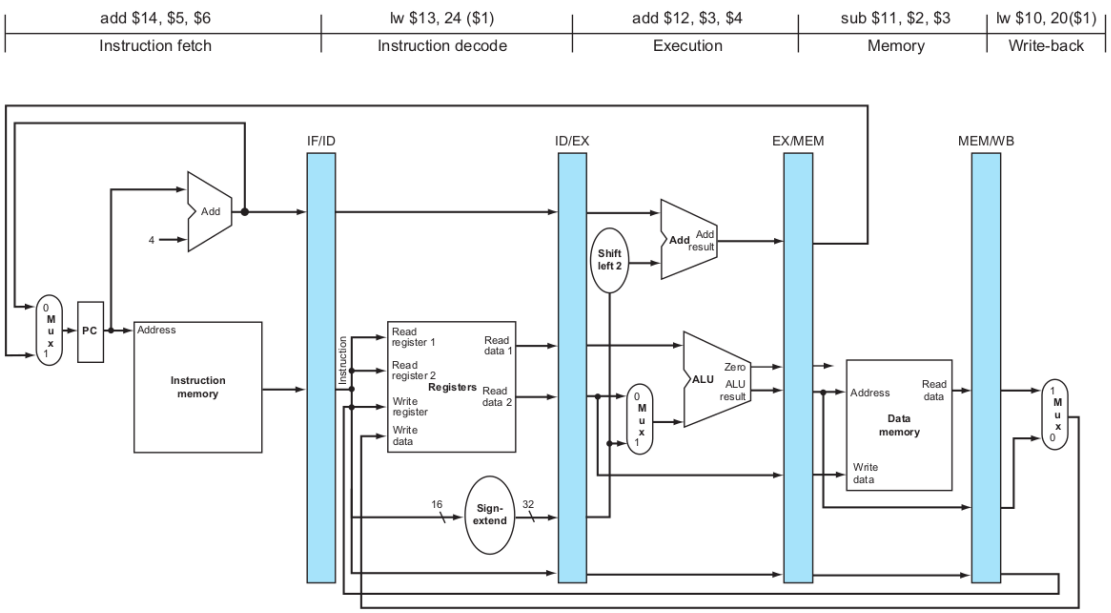
4. MEM(Memory Data Excess/ Read Memory) : the memory excess port of the D-memory gets activated in this stage. This stage is active in LW and SW instructions only.

5. WB(Write Back) : The final stage of pipe is write back, it executes the ALU-Result or the value that is supposed to write in a register. We simply upadate the register with the new updated value.

The traditional multiple-clock-cycle pipelined diagram of five instrucions is shown below :



The single-clock-cycle diagram corresponding to clock cycle 5 of the above pipeline is shown below:



The Hazards detection and removing them is explained below (not optimized):

1. if we find an instruction in EX stage such that data hazard occurs, (i.e. destination register of EX is same as one of input registers of ID stage) then we simply put two stalls “STALL_datahaz_1st” and “STALL_datahaz_2nd”.

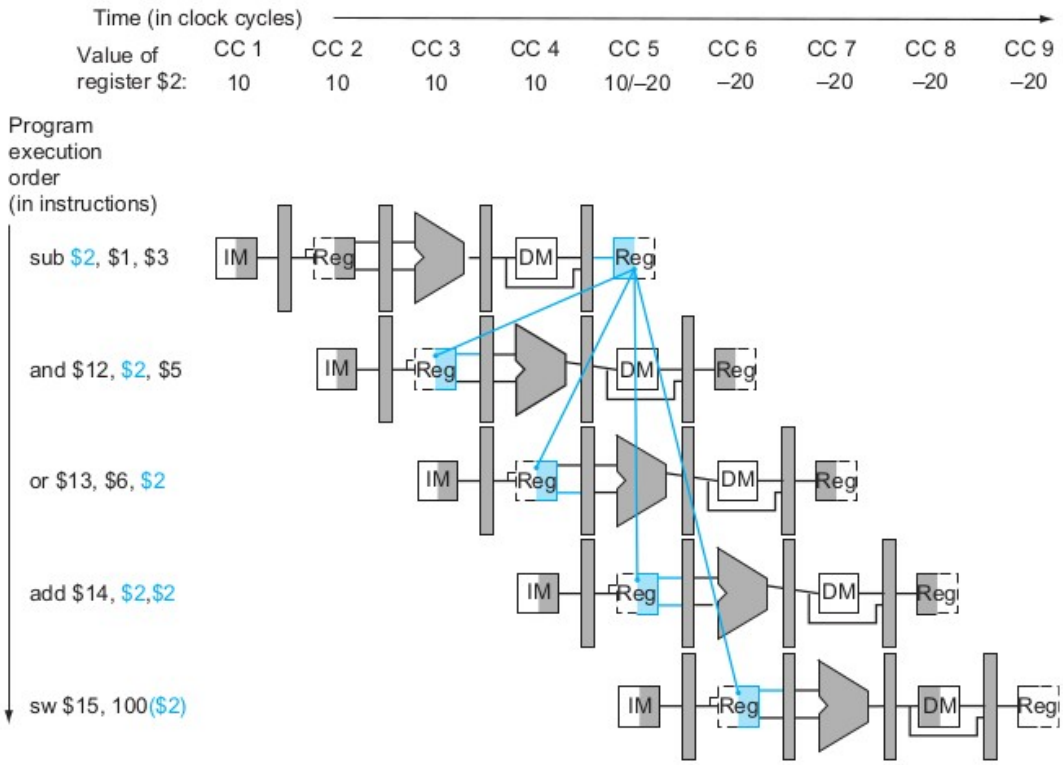
2. if in the EX stage of the current cycle, BEQ, BNE, BGTZ, BLEZ and the program counter changes then we insert two stalls of “BranchHazard”.

4. In case of J, JR, JAL, we simply put a stall “JumpStall”.

Optmization of the Hazards:

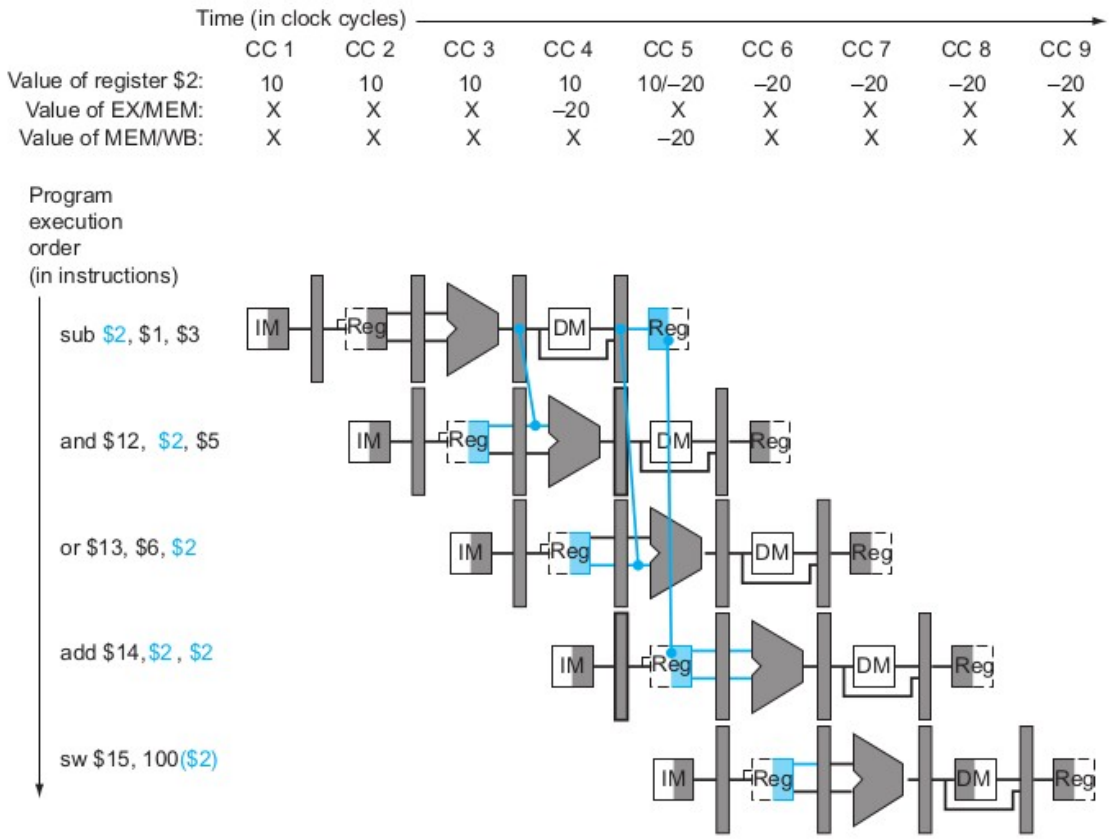
We can remove the data hazards by *forwarding* the values. Data hazards occur when we need the value of a register in a stage but it has not updated yet.

Here is the pipelined dependences in a five-instruction sequence using simplified datapaths to show the dependences: All the dependent actions are shown in color, and “CC 1” at the top of the fi gure means clock cycle 1. Th e fi rst instruction writes into $2, and all the following instructions read $2. Th is register is written in clock cycle 5, so the proper value is unavailable before clock cycle 5. (A read of a register during a clock cycle returns the value written at the end of the fi rst half of the cycle, when such a write occurs.) The colored lines from the top datapath to the lower ones show the dependences. Th ose that must go backward in time are pipeline data hazards.



To prevent stalls because of such dependences, we use forward bypassing.
The dependences between the pipeline registers move forward in time, so it is possible to supply the inputs to the ALU need by the ‘AND’ instruction and ‘OR’ instruction by forwarding the results found in the pipeline registers in the above example. Th e values in the pipeline registers show that the desired value is available before it is written into the register fi le. We assume that the register fi le forwards values that are read and written during the same clock cycle, so the add does not stall, but the values come from the register fi le instead of a pipeline register. Register file forwarding” — that is, the read gets the value of the write in that clock cycle—is why clock cycle 5 shows register $2 having the value 10 at the beginning and −20 at the end of the clock cycle. As in the rest of this section, we handle all forwarding except for the value to be stored by a store instruction.

Forwarding is shown in the diagram below :



EXAMPLE 1 :

Initial values :
$t1 = 1;
$t2 = 2;
$t3 = 3;
$t4 = 4;
$ra = 50;

Instructions :

ADD $t6 $t2 $t4
ADD $t7 $t6 $t2
SLL $t7 $t7 3
SRL $t8 $t7 2
SW $t8 12($ra)
SLL $t8 $t8 2
LW $s1 12($ra)

Add:

ADD $s2 $s2 $t1
SUB $s3 $s2 $s1
BEQ $s2 $t1 Add
HALT 0 0 0 0

Final Register Values :

reg: $zero : 0 $at : 0 $v0 : 0 $v1 : 0 $a0 : 0 $a1 : 0 $a2 : 0 $a3 : 0 $t0 : 0 $t1 : 1 $t2 : 2 $t3 : 3
$t4 : 4 $t5 : 5 $t6 : 6 $t7 : 64 $s0 : 0 $s1 : 16 $s2 : 2 $s3 : -14 $s4 : 0
$s5 : 0 $s6 : 0 $s7 : 0 $t8 : 64 $t9 : 0 $k0 : 0 $k1 : 0 $gp : 0 $sp : 0 $fb : 0 $ra : 50

Here is the data of every clock cycle that shows the instruction in each stage :
(Initially all the stages are NOOP)

1. IF : 0 ID : NOOP 0 0 0 EX : NOOP 0 0 0 MEM : NOOP 0 0 0 WB : NOOP 0 0 0
2. IF : 1 ID : NOOP 0 0 0 EX : NOOP 0 0 0 MEM : NOOP 0 0 0 WB : NOOP 0 0 0
3. IF : 2 ID : ADD $t6 $t2 $t4 EX : NOOP 0 0 0 MEM : NOOP 0 0 0 WB : NOOP 0 0 0
4. IF : 3 ID : ADD $t7 $t6 $t2 EX : ADD $t6 $t2 $t4 MEM : NOOP 0 0 0 WB : NOOP 0 0 0
5. IF : 4 ID : SLL $t7 $t7 3 EX : ADD $t7 $t6 $t2 MEM : ADD $t6 $t2 $t4 WB : NOOP 0 0 0
6. IF : 5 ID : SRL $t8 $t7 2 EX : SLL $t7 $t7 3 MEM : ADD $t7 $t6 $t2 WB : ADD $t6 $t2 $t4
7. IF : 6 ID : SW $t8 12($ra) EX : SRL $t8 $t7 2 MEM : SLL $t7 $t7 3 WB : ADD $t7 $t6 $t2
8. IF : 7 ID : SLL $t8 $t8 2 EX : SW $t8 12($ra) MEM : SRL $t8 $t7 2 WB : SLL $t7 $t7 3
9. IF : 8 ID : LW $s1 12($ra) EX : SLL $t8 $t8 2 MEM : SW $t8 12($ra) WB : SRL $t8 $t7 2
10. IF : 9 ID : ADD $s2 $s2 $t1 EX : LW $s1 12($ra) MEM : SLL $t8 $t8 2 WB : SW $t8 12($ra)
11. IF : 10 ID : SUB $s3 $s2 $s1 EX : ADD $s2 $s2 $t1 MEM : LW $s1 12($ra) WB : SLL $t8 $t8 2
12. IF : 7 ID : BEQ $s2 $t1 Add EX : SUB $s3 $s2 $s1 MEM : ADD $s2 $s2 $t1 WB : LW $s1 12($ra)
13. IF : 8 ID : STALL3 0 0 0 EX : BEQ $s2 $t1 Add MEM : SUB $s3 $s2 $s1 WB : ADD $s2 $s2 $t1
14. IF : 9 ID : ADD $s2 $s2 $t1 EX : STALL3 0 0 0 MEM : BEQ $s2 $t1 Add WB : SUB $s3 $s2 $s1
15. IF : 10 ID : SUB $s3 $s2 $s1 EX : ADD $s2 $s2 $t1 MEM : STALL3 0 0 0 WB : BEQ $s2 $t1 Add
16. IF : 11 ID : BEQ $s2 $t1 Add EX : SUB $s3 $s2 $s1 MEM : ADD $s2 $s2 $t1 WB : STALL3 0 0 0
17. IF : 12 ID : HALT 0 0 0 0 EX : BEQ $s2 $t1 Add MEM : SUB $s3 $s2 $s1 WB : ADD $s2 $s2 $t1
18. IF : 13 ID : EX : HALT 0 0 0 0 MEM : BEQ $s2 $t1 Add WB : SUB $s3 $s2 $s1
19. IF : 14 ID : EX : HALT 0 0 0 0 MEM : HALT 0 0 0 0 WB : BEQ $s2 $t1 Add

The program takes total 18 cycles, rather the same example in unoptimized pipelining took 31 cycles.

EXAMPLE 2 :

Given values :
$v1 = 4;
$a0 = 7;
$a1 = 1;
$a3 = 3;
$t1 = 5;

$t2 = 3;
$t4 = 2;
$t5 = 2;
$t6 = 13;
memory[28] = 73;

Instructions :

```
first:
        ADD $v1 $a0 $a1
        SUB $t0 $t1 $t2
        BEQ $t4 $t5 third

second:
        SRL $a3 $t5 3
        SW $t6 18($v1)
third:
        SLL $a1 $a3 3
        LW $t7 20($v1)
        HALT 0 0 0
```

Final Register Values :

$zero : 0 $at : 0 $v0 : 0 $v1 : 8 $a0 : 7 $a1 : 24 $a2 : 0 $a3 : 3 $t0 : 2 $t1 : 5 $t2 : 3
$t3 : 0 $t4 : 2 $t5 : 2 $t6 : 13 $t7 : 73 $s0 : 0 $s1 : 0 $s2 : 0 $s3 : 0 $s4 : 0
$s5 : 0 $s6 : 0 $s7 : 0 $t8 : 0 $t9 : 0 $k0 : 0 $k1 : 0 $gp : 0 $sp : 0 $fb : 0 $ra : 0

Here is the data of every clock cycle that shows the instruction in each stage :
(Initially all the stages are NOOP)

1. IF : 0 ID : NOOP 0 0 0 EX : NOOP 0 0 0 MEM : NOOP 0 0 0 WB : NOOP 0 0 0
2. IF : 1 ID : NOOP 0 0 0 EX : NOOP 0 0 0 MEM : NOOP 0 0 0 WB : NOOP 0 0 0
3. IF : 2 ID : ADD $v1 $a0 $a1 EX : NOOP 0 0 0 MEM : NOOP 0 0 0 WB : NOOP 0 0 0
4. IF : 3 ID : SUB $t0 $t1 $t2 EX : ADD $v1 $a0 $a1 MEM : NOOP 0 0 0 WB : NOOP 0 0 0
5. IF : 5 ID : BEQ $t4 $t5 third EX : SUB $t0 $t1 $t2 MEM : ADD $v1 $a0 $a1 WB : NOOP 0 0 0
6. IF : 6 ID : STALL3 0 0 0 EX : BEQ $t4 $t5 third MEM : SUB $t0 $t1 $t2 WB : ADD $v1 $a0 $a1
7. IF : 7 ID : SLL $a1 $a3 3 EX : STALL3 0 0 0 MEM : BEQ $t4 $t5 third WB : SUB $t0 $t1 $t2
8. IF : 8 ID : LW $t7 20($v1) EX : SLL $a1 $a3 3 MEM : STALL3 0 0 0 WB : BEQ $t4 $t5 third
9. IF : 9 ID : HALT 0 0 0 EX : LW $t7 20($v1) MEM : SLL $a1 $a3 3 WB : STALL3 0 0 0
10. IF : 10 ID : EX : HALT 0 0 0 MEM : LW $t7 20($v1) WB : SLL $a1 $a3 3
11. IF : 11 ID : EX : HALT 0 0 0 MEM : HALT 0 0 0 WB : LW $t7 20($v1)

The program takes total 10 cycles to complete rather the same example in unoptimized pipelining took 11 cycles.

TEST CASE 3

Given Values :

$a0 = 7;

$t1 = 3;
$t3 = 2;
$t4 = 1;
$sp = 90;

Instructions:

```
JAL fibonacci
ADD $a1 $v0 $zero
J exit
fibonacci:
SUB $sp $sp $t1
SW $ra 2($sp)
SW $s0 1($sp)
SW $s1 0($sp)
ADD $s0 $a0 $zero
ADD $v0 $t4 $zero
SUB $t2 $s0 $t3
BLEZ $t2 fibonacciexit
SUB $a0 $s0 $t4
JAL fibonacci
ADD $s1 $v0 $zero
SUB $a0 $s0 $t3
JAL fibonacci
ADD $v0 $s1 $v0
fibonacciexit:
LW $ra 2($sp)
LW $s0 1($sp)
LW $s1 0($sp)
ADD $sp $sp $t1
JR $ra
exit:
HALT
```

Final Register Values :

reg: $zero : 0 $at : 0 $v0 : 13 $v1 : 0 $a0 : 1 $a1 : 13 $a2 : 0 $a3 : 0 $t0 : 0 $t1 : 3
$t2 : -1 $t3 : 2 $t4 : 1 $t5 : 0 $t6 : 0 $t7 : 0 $s0 : 0 $s1 : 0 $s2 : 0 $s3 : 0 $s4 : 0 $s5 : 0
$s6 : 0 $s7 : 0 $t8 : 0 $t9 : 0 $k0 : 0 $k1 : 0 $gp : 0 $sp : 90 $fb : 0 $ra : 1

mem: memory[72] : 1
memory[73] : 3
memory[74] : 16
memory[75] : 1
memory[76] : 3
memory[77] : 16
memory[78] : 1
memory[79] : 3
memory[80] : 16
memory[81] : 3
memory[82] : 5
memory[83] : 16
memory[84] : 8

memory[85] : 7
memory[86] : 16
memory[87] : 0
memory[88] : 0
memory[89] : 1

The program takes total 468 cycles to complete rather the same example in unoptimized pipelining took 581 cycles.

Assignment By
Nikita Bhamu 2018CS50413
Manoj Kumar 2018CS50411