

Assignment – 8

COL216 – Computer Architecture

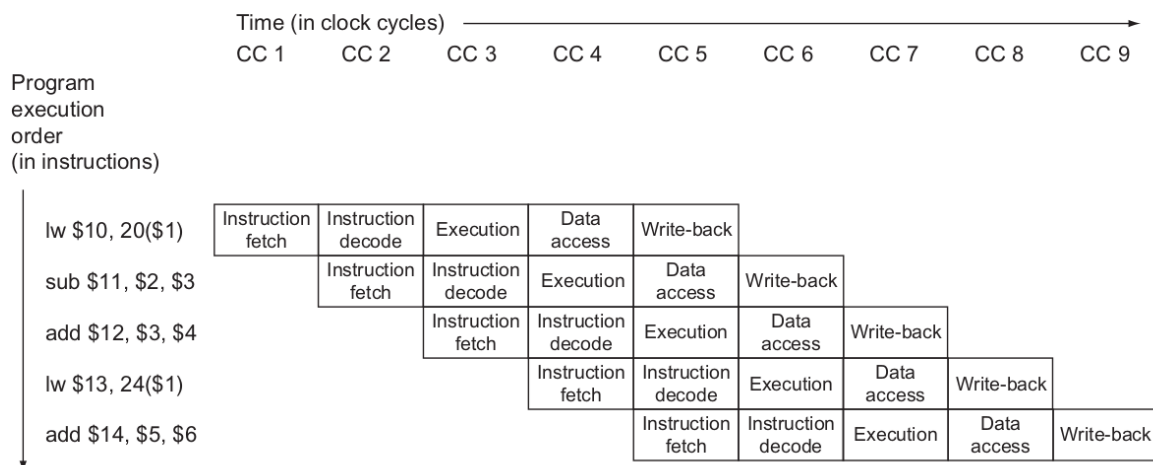
Simulating a Pipelined Processor

This assignment is an extension of the previous assignment. Pipeline has been introduced in this assignment.

There are total five stages of a pipelined processor :

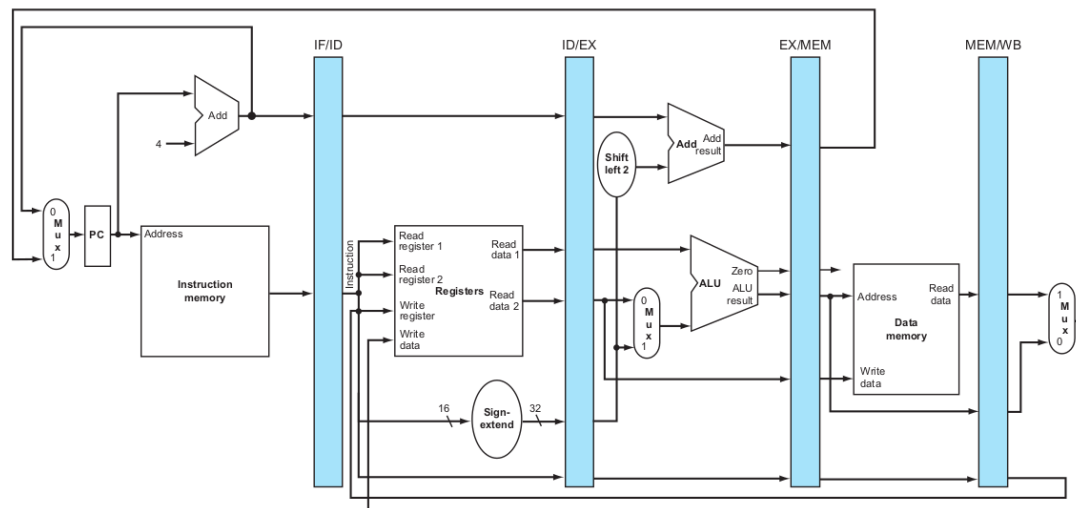
1. IF(Instruction fetch) : This is the first stage of executing an instruction. In this, stage, the processor reads the respective instruction from I-Memory and increase the program counter to 1. after fetching the instrucion, it procced the data to the next stage in next cycle.
2. ID(Instruction Decode)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

add \$14, \$5, \$6	lw \$13, 24(\$1)	add \$12, \$3, \$4	sub \$11, \$2, \$3	lw \$10, 20(\$1)
Instruction fetch	Instruction decode	Execution	Memory	Write-back



The Hazards detection and removing them is explained below :

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

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 : 3 ID : ADD \$t7 \$t6 \$t2 EX : STALL_datahaz_1st 0 0 0 MEM : ADD \$t6 \$t2 \$t4 WB : NOOP 0 0 0
6. IF : 3 ID : ADD \$t7 \$t6 \$t2 EX : STALL_datahaz_2nd 0 0 0 MEM : STALL_datahaz_1st 0 0 0 WB : ADD \$t6 \$t2 \$t4
7. IF : 4 ID : SLL \$t7 \$t7 3 EX : ADD \$t7 \$t6 \$t2 MEM : STALL_datahaz_2nd 0 0 0 WB : STALL_datahaz_1st 0 0 0
8. IF : 4 ID : SLL \$t7 \$t7 3 EX : STALL_datahaz_1st 0 0 0 MEM : ADD \$t7 \$t6 \$t2 WB : STALL_datahaz_2nd 0 0 0
9. IF : 4 ID : SLL \$t7 \$t7 3 EX : STALL_datahaz_2nd 0 0 0 MEM : STALL_datahaz_1st 0 0 0 WB : ADD \$t7 \$t6 \$t2
10. IF : 5 ID : SRL \$t8 \$t7 2 EX : SLL \$t7 \$t7 3 MEM : STALL_datahaz_2nd 0 0 0 WB : STALL_datahaz_1st 0 0 0
11. IF : 5 ID : SRL \$t8 \$t7 2 EX : STALL_datahaz_1st 0 0 0 MEM : SLL \$t7 \$t7 3 WB : STALL_datahaz_2nd 0 0 0
12. IF : 5 ID : SRL \$t8 \$t7 2 EX : STALL_datahaz_2nd 0 0 0 MEM : STALL_datahaz_1st 0 0 0 WB : SLL \$t7 \$t7 3
13. IF : 6 ID : SW \$t8 12(\$ra) EX : SRL \$t8 \$t7 2 MEM : STALL_datahaz_2nd 0 0 0 WB : STALL_datahaz_1st 0 0 0
14. IF : 6 ID : SW \$t8 12(\$ra) EX : STALL_datahaz_1st 0 0 0 MEM : SRL \$t8 \$t7 2 WB : STALL_datahaz_2nd 0 0 0
15. IF : 6 ID : SW \$t8 12(\$ra) EX : STALL_datahaz_2nd 0 0 0 MEM : STALL_datahaz_1st 0 0 0 WB : SRL \$t8 \$t7 2
16. IF : 7 ID : SLL \$t8 \$t8 2 EX : SW \$t8 12(\$ra) MEM : STALL_datahaz_2nd 0 0 0 WB : STALL_datahaz_1st 0 0 0
17. IF : 8 ID : LW \$s1 12(\$ra) EX : SLL \$t8 \$t8 2 MEM : SW \$t8 12(\$ra) WB : STALL_datahaz_2nd 0 0 0
18. IF : 9 ID : ADD \$s2 \$s2 \$t1 EX : LW \$s1 12(\$ra) MEM : SLL \$t8 \$t8 2 WB : SW \$t8 12(\$ra)
19. IF : 10 ID : SUB \$s3 \$s2 \$s1 EX : ADD \$s2 \$s2 \$t1 MEM : LW \$s1 12(\$ra) WB : SLL \$t8 \$t8 2
20. IF : 10 ID : SUB \$s3 \$s2 \$s1 EX : STALL_datahaz_1st 0 0 0 MEM : ADD \$s2 \$s2 \$t1 WB : LW \$s1 12(\$ra)
21. IF : 10 ID : SUB \$s3 \$s2 \$s1 EX : STALL_datahaz_2nd 0 0 0 MEM : STALL_datahaz_1st 0 0 0 WB : ADD \$s2 \$s2 \$t1
22. IF : 11 ID : BEQ \$s2 \$t1 Add EX : SUB \$s3 \$s2 \$s1 MEM : STALL_datahaz_2nd 0 0 0 WB : STALL_datahaz_1st 0 0 0

23. IF : 7 ID : BranchStall 0 0 0 EX : BEQ \$s2 \$t1 Add MEM : SUB \$s3 \$s2 \$s1 WB :
 STALL_datahaz_2nd 0 0 0
 24. IF : 8 ID : BranchStall 0 0 0 EX : BranchStall 0 0 0 MEM : BEQ \$s2 \$t1 Add WB : SUB
 \$s3 \$s2 \$s1
 25. IF : 9 ID : ADD \$s2 \$s2 \$t1 EX : BranchStall 0 0 0 MEM : BranchStall 0 0 0 WB : BEQ \$s2
 \$t1 Add
 26. IF : 10 ID : SUB \$s3 \$s2 \$s1 EX : ADD \$s2 \$s2 \$t1 MEM : BranchStall 0 0 0 WB :
 BranchStall 0 0 0
 27. IF : 10 ID : SUB \$s3 \$s2 \$s1 EX : STALL_datahaz_1st 0 0 0 MEM : ADD \$s2 \$s2 \$t1
 WB : BranchStall 0 0 0
 28. IF : 10 ID : SUB \$s3 \$s2 \$s1 EX : STALL_datahaz_2nd 0 0 0 MEM : STALL_datahaz_1st
 0 0 0 WB : ADD \$s2 \$s2 \$t1
 29. IF : 11 ID : BEQ \$s2 \$t1 Add EX : SUB \$s3 \$s2 \$s1 MEM : STALL_datahaz_2nd 0 0 0 WB
 : STALL_datahaz_1st 0 0 0
 30. IF : 12 ID : HALT 0 0 0 0 EX : BEQ \$s2 \$t1 Add MEM : SUB \$s3 \$s2 \$s1 WB :
 STALL_datahaz_2nd 0 0 0
 31. IF : 13 ID : EX : HALT 0 0 0 0 MEM : BEQ \$s2 \$t1 Add WB : SUB \$s3 \$s2 \$s1
 32. IF : 14 ID : EX : HALT 0 0 0 0 MEM : HALT 0 0 0 0 WB : BEQ \$s2 \$t1 Add

The program takes total 31 cycles.

EXAMPLE 2 :

Given values :

\$v1 = 4;
 \$a0 = 7;
 \$a1 = 1;
 \$a3 = 3;
 \$t1 = 5;
 \$t2 = 3;
 \$t4 = 2;
 \$t5 = 2;
 \$t6 = 13;

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

The program takes total 11 cycles to complete.

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 581 cycles to complete.

Assignment By
Nikita Bhamu 2018CS50413
Manoj Kumar 2018CS50411