# Homework Assignment - 4

*Submitted By*
*Fahd Humayun*
*168000889 (fh186)*

**Question – 1:**

| Instruction | *RegDst* | *ALUSrc* | *MemtoReg* | *RegWrite* | *MemRead* | *MemWrite* | *Branch* | *ALUOp* **1: 0** | *Jump* |
|---|---|---|---|---|---|---|---|---|---|
| *andi* $t0, $t1,12 | *0* | *1* | *0* | *1* | *0* | *0* | *0* | *1 1* | *0* |
| *lw* $t0, 12($12) | *0* | *1* | *1* | *1* | *1* | *0* | *0* | *0 0* | *0* |
| *sw* $t4, 12($12) | *X* | *1* | *X* | *0* | *0* | *1* | *0* | *0 0* | *0* |

The MIPS single cycle implementation diagram and control signals need to be modified to deal with immediate instructions such as $andi, ori$ etc. The ALU Control needs to generate 0000 as an input to ALU to perform $AND$ operation, and as $ALUOp$ 00 is used for *lw* and *sw* that generates 0010 ALU Control input to perform $ADD$ operation, 01 is used for *branch* that generates 0110 to perform $SUB$ operation, and 10 is used for the R-type instructions that depends on the $funct$ field of the R-type to determine what type of arithmetic operation to perform, therefore, assume the modification yields 11 as $ALUOp$ signal for $andi$ instruction that would generate 0000 as ALU Control input to the ALU.

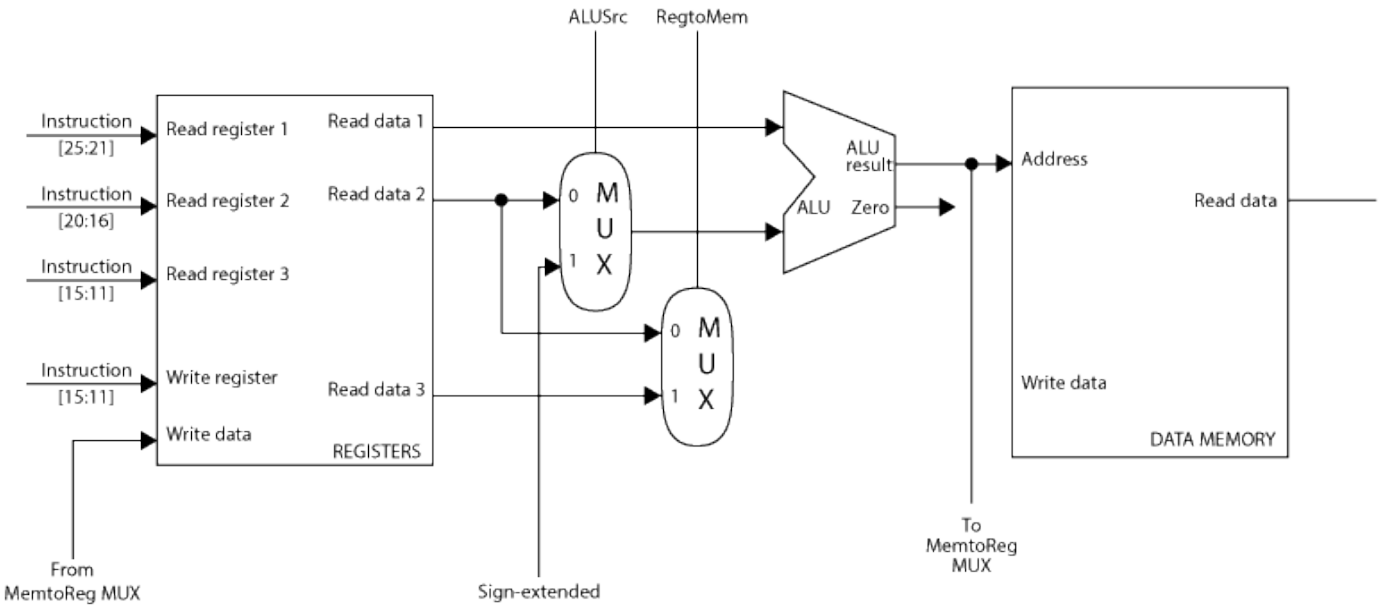
**Question – 2:**

In the instruction *swi* $t1, $t2($t4) registers $t2 and $t4 needs to be added to generate the address and $t1 has the data that needs to be stored in the memory, so, the REGISTERS component of the $datapath$ of single cycle MIPS needs to have $Read\ register$ 3 as an additional input and $Read\ data$ 3 as an additional output, where the $Read\ register$ 3 will take in as an input $instruction$ [15 – 11] bits (treating the instruction as R-type where instruction bits 15 – 11 will represent $rd$ ($t1 in this case)) $Read\ register$ 1 $and$ 2 will be used for the registers $t2, $and$ $t4 as an operands to the ALU to be added and to compute the address of the memory. To the existing $datapath$ (un-pipelined) the input to MEMORY for $Write\ data$ comes from the REGISTERS output i.e. $Read\ data$ 2, so, a multiplexor can be added that takes as input $Read\ data$ 2 and $Read\ data$ 3 and then used as an input to the MEMORY's $Write\ data$, where, this multiplexor can be controlled with an additional control signal/line from the CONTROL unit named as $RegtoMem$. If $RegtoMem$ is 1 then it will write the data from $Read\ data$ 3 and if it is 0 then it will use the $Read\ data$ 2 to write in the

memory. The table below shows all of the control signals that the CONTROL unit needs to generate to execute the instruction given and figure below shows the portion/parts of *datapath* where the changes are/have been needed.

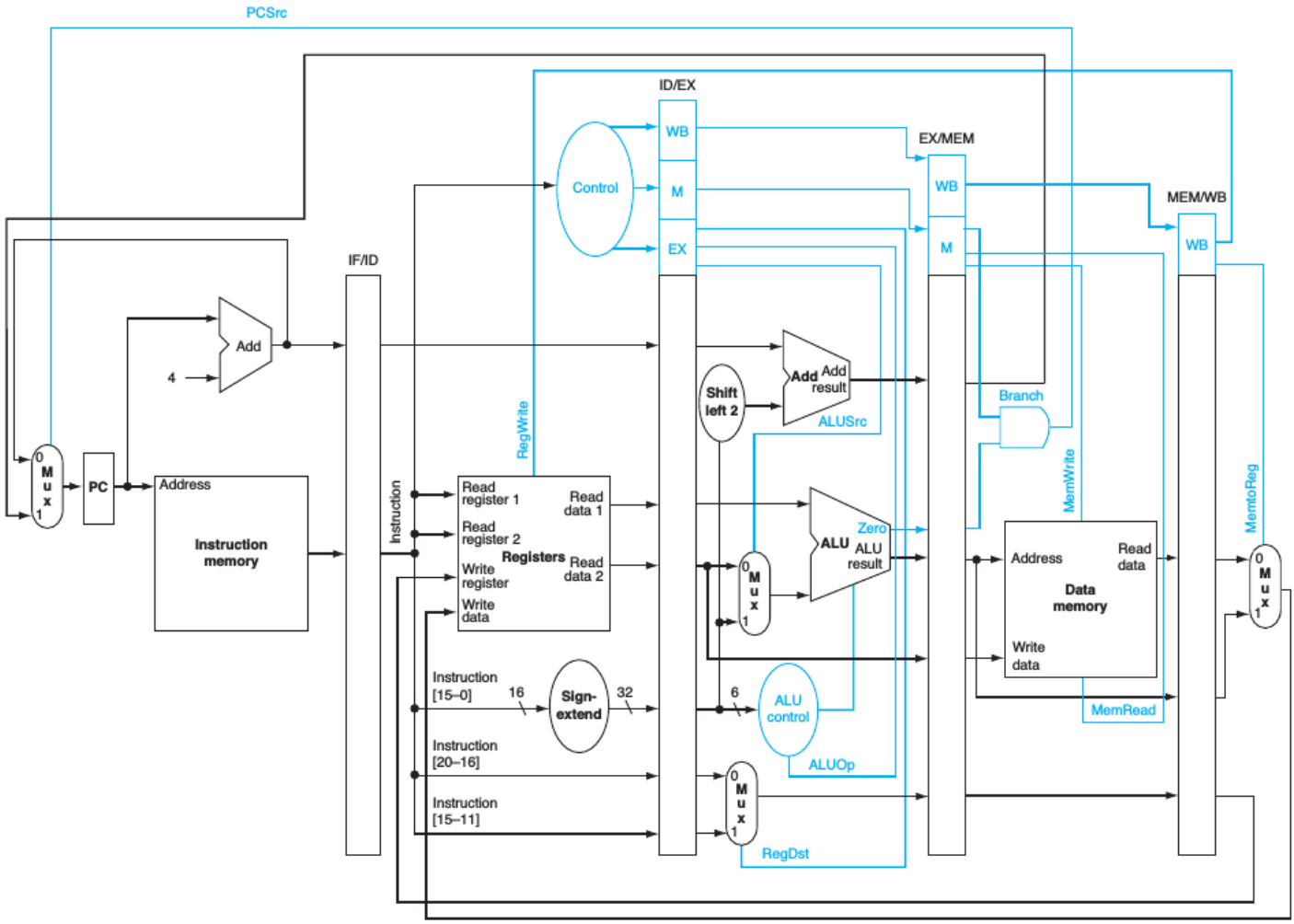
*PS: for a store instruction the ALU needs to perform an addition operation so the funct fields of this instruction (as this instruction can be treated as R-type) can be don't care as the CONTROL unit will generate ALUOp 00 to generate ALU Control input 0010 for addition (assuming the opcode of this instruction will generate the control signals as required)*

| ***RegtoMem*** | ***RegDst*** | ***ALUSrc*** | ***MemtoReg*** | ***RegWrite*** | ***MemRead*** | ***MemWrite*** | ***Branch*** | ***ALUOp*** | ***Jump*** |
|---|---|---|---|---|---|---|---|---|---|
| *1* | *X* | *0* | *X* | *0* | *0* | *1* | *0* | *0 0* | *0* |



## Question – 3:

| Instructions | Clock Cycles | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** |
| *add* \$1, \$3, \$5 | *IF* | *ID* | *EX* | *MEM* | *WB* | | | | | | |
| *and* \$10, \$8, \$3 | | *IF* | *ID* | *EX* | *MEM* | *WB* | | | | | |
| *lw* \$4, 16(\$3) | | | *IF* | *ID* | *EX* | *MEM* | *WB* | | | | |
| *sub* \$11, \$2, \$7 | | | | *IF* | *ID* | *EX* | *MEM* | *WB* | | | |
| *sw* \$2, 100(\$6) | | | | | *IF* | *ID* | *EX* | *MEM* | *WB* | | |



## Question – 4 (a):

| Instructions | Clock Cycles | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** |
| *add R3, R2, R7* | *IF* | *ID* | *EX* | *MEM* | *WB* | | | | | | |
| *lw R4,20(R3)* | | *IF* | *ID* | *EX* | *MEM* | *WB* | | | | | |
| *and R6, R5, R4* | | | *IF* | *ID* | *EX* | *MEM* | *WB* | | | | |
| *sw R3, −40(R5)* | | | | *IF* | *ID* | *EX* | *MEM* | *WB* | | | |
| *add R4, R6, R3* | | | | | *IF* | *ID* | *EX* | *MEM* | *WB* | | |
| *sub R5, R6, R4* | | | | | | *IF* | *ID* | *EX* | *MEM* | *WB* | |

The shaded stages show the data availability and the dependency of data.

1. R7 is ready in WB stage of *add R3, R2, R7* instruction i.e. clock cycle 5 and it is needed in ID stage of *lw R4,20(R3)* instruction i.e. clock cycle 3.

Similarly,

2. R4 is ready in the instruction *lw R4,20(R3)* at clock cycle 6 while it is needed in the instruction *and R6, R5, R4* at clock cycle 4.
3. R6 is ready in the instruction *and R6, R5, R4* at clock cycle 7 while it is needed in the instruction *add R4, R6, R3* at clock cycle 6.
4. R4 is ready in the instruction *add R4, R6, R3* at clock cycle 9 while it is needed in the instruction *sub R5, R6, R4* at clock cycle 7.

## Question – 4 (b):

| Instruction | Clock Cycles | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** | **16** | **17** |
| *add R3, R2, R7* | *IF* | *ID* | *EX* | *MEM* | *WB* | | | | | | | | | | | | |
| | | *Stall* | *Stall* | *Stall* | *Stall* | *Stall* | | | | | | | | | | | |
| | | | *Stall* | *Stall* | *Stall* | *Stall* | *Stall* | | | | | | | | | | |
| *lw R4,20(R3)* | | | | *IF* | *ID* | *EX* | *MEM* | *WB* | | | | | | | | | |
| | | | | | *Stall* | *Stall* | *Stall* | *Stall* | *Stall* | | | | | | | | |
| | | | | | | *Stall* | *Stall* | *Stall* | *Stall* | *Stall* | | | | | | | |
| *and R6, R5, R4* | | | | | | | *IF* | *ID* | *EX* | *MEM* | *WB* | | | | | | |
| *sw R3, −40(R5)* | | | | | | | | *IF* | *ID* | *EX* | *MEM* | *WB* | | | | | |
| | | | | | | | | | *Stall* | *Stall* | *Stall* | *Stall* | *Stall* | | | | |
| *add R4, R6, R3* | | | | | | | | | | *IF* | *ID* | *EX* | *MEM* | *WB* | | | |
| | | | | | | | | | | | *Stall* | *Stall* | *Stall* | *Stall* | *Stall* | | |
| | | | | | | | | | | | | *Stall* | *Stall* | *Stall* | *Stall* | *Stall* | |
| *sub R5, R6, R4* | | | | | | | | | | | | | *IF* | *ID* | *EX* | *MEM* | *WB* |

- 17 clock cycles needed to run the given code.

## Question – 4 (c):

| Instructions | Clock Cycles | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** |
| *add R3, R2, R7* | *IF* | *ID* | *EX* | *MEM* | *WB* | | | | | | |
| *lw R4,20(R3)* | | *IF* | *ID* | *EX* | *MEM* | *WB* | | | | | |
| | | | *Stall* | *Stall* | *Stall* | *Stall* | *Stall* | | | | |
| *and R6, R5, R4* | | | | *IF* | *ID* | *EX* | *MEM* | *WB* | | | |
| *sw R3, −40(R5)* | | | | | *IF* | *ID* | *EX* | *MEM* | *WB* | | |
| *add R4, R6, R3* | | | | | | *IF* | *ID* | *EX* | *MEM* | *WB* | |
| *sub R5, R6, R4* | | | | | | | *IF* | *ID* | *EX* | *MEM* | *WB* |

- 11 clock cycles needed to run the given code.

## Question – 4 (d):

| Instructions | Clock Cycles | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| *add R3, R2, R7* | *IF* | *ID* | *EX* | *MEM 1* | *MEM 2* | *WB* | | | | | |
| *lw R4,20(R3)* | | *IF* | *ID* | *EX* | *MEM 1* | *MEM 2* | *WB* | | | | |
| *and R6, R5, R4* | | | *IF* | *ID* | *EX* | *MEM 1* | *MEM 2* | *WB* | | | |
| *sw R3, −40(R5)* | | | | *IF* | *ID* | *EX* | *MEM 1* | *MEM 2* | *WB* | | |
| *add R4, R6, R3* | | | | | *IF* | *ID* | *EX* | *MEM 1* | *MEM 2* | *WB* | |
| *sub R5, R6, R4* | | | | | | *IF* | *ID* | *EX* | *MEM 1* | *MEM 2* | *WB* |

1. R3 available in clock cycle (CC) 6 in instruction 1 and needed in CC3 in instruction 2.
2. R4 available in CC7 in instruction 2 and needed in CC4 in instruction 3.
3. R3 available in CC6 in instruction 1 and needed in CC5 in instruction 4.
4. R6 available in CC8 in instruction 3 and needed in CC6 in instruction 5.
5. R4 available in CC10 in instruction 5 and needed in CC7 in instruction 6.

Without forwarding 22 clock cycles:

| Instructions | Clock Cycles | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 |
| add R3, R2, R7 | IF | ID | EX | MEM1 | MEM2 | WB | | | | | | | | | | | | | | | | |
| | | X | X | X | X | X | X | | | | | | | | | | | | | | | |
| | | | X | X | X | X | X | X | | | | | | | | | | | | | | |
| | | | | X | X | X | X | X | X | | | | | | | | | | | | | |
| lw R4, 20(R3) | | | | | IF | ID | EX | MEM1 | MEM2 | WB | | | | | | | | | | | | |
| | | | | | | X | X | X | X | X | X | | | | | | | | | | | |
| | | | | | | | X | X | X | X | X | X | | | | | | | | | | |
| | | | | | | | | X | X | X | X | X | X | | | | | | | | | |
| and R6, R5, R4 | | | | | | | | | IF | ID | EX | MEM1 | MEM2 | WB | | | | | | | | |
| sw R3, -40(R5) | | | | | | | | | | IF | ID | EX | MEM1 | MEM2 | WB | | | | | | | |
| | | | | | | | | | | | X | X | X | X | X | X | | | | | | |
| | | | | | | | | | | | | X | X | X | X | X | X | | | | | |
| add R4, R6, R3 | | | | | | | | | | | | | IF | ID | EX | MEM1 | MEM2 | WB | | | | |
| | | | | | | | | | | | | | | X | X | X | X | X | X | | | |
| | | | | | | | | | | | | | | | X | X | X | X | X | X | | |
| | | | | | | | | | | | | | | | | X | X | X | X | X | X | |
| sub R5, R6, R4 | | | | | | | | | | | | | | | | | IF | ID | EX | MEM1 | MEM2 | WB |

With forwarding 13 clock cycles:

| Instructions | Clock Cycles | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** |
| *add R3, R2, R7* | *IF* | *ID* | *EX* | *MEM 1* | *MEM 2* | *WB* | | | | | | | |
| *lw R4,20(R3)* | | *IF* | *ID* | *EX* | *MEM 1* | *MEM 2* | *WB* | | | | | | |
| | | | *Stall* | *Stall* | *Stall* | *Stall* | *Stall* | *Stall* | | | | | |
| | | | | *Stall* | *Stall* | *Stall* | *Stall* | *Stall* | *Stall* | | | | |
| *and R6, R5, R4* | | | | | *IF* | *ID* | *EX* | *MEM 1* | *MEM 2* | *WB* | | | |
| *sw R3, −40(R5)* | | | | | | *IF* | *ID* | *EX* | *MEM 1* | *MEM 2* | *WB* | | |
| *add R4, R6, R3* | | | | | | | *IF* | *ID* | *EX* | *MEM 1* | *MEM 2* | *WB* | |
| *sub R5, R6, R4* | | | | | | | | *IF* | *ID* | *EX* | *MEM 1* | *ME M2* | *WB* |

**Question – 5:**

Load instructions immediately followed by an instruction that uses the value just loaded in requires 1 stall/NOP.

Store instruction immediately preceded by an R-type instruction that computes the value that is written to memory does not need any stall.

ALU instruction immediately followed by another ALU instruction that uses the value just computed also does not need any stall.

Ideal CPI = 1

Total CPI = 1 + (30%)(40%)(number of stalls) = 1 + (0.30)(0.40)(1) = 1.12

IPC = 1 / CPI = 1/1.12 = 0.893 instructions/cycle

---