

# Computer Architecture

## Final Assessment

Team 3

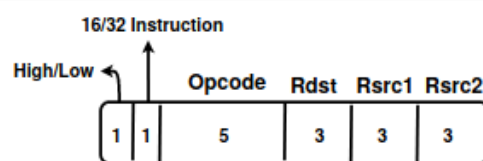
Name:	Sec.	B.N
Ahmed Mahmoud Abd El-Monaem	1	6
Ayman Adel Aly	1	13
Mohamed Ahmed Ibrahim	2	9
Yosry Mohammed Yosry	2	35

# Schematic Diagram:

\*\* The Full Design can be found [Here](#)

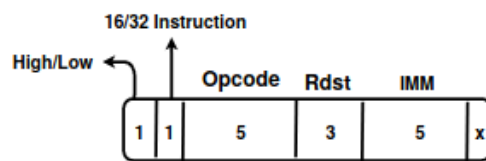
## Instructions Format:

### Group A:



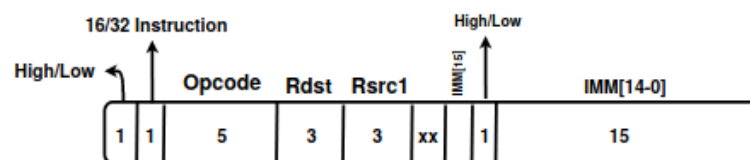
NOP, NOT, INC, DEC, OUT, IN  
SWAP, ADD, SUB, AND, OR, PUSH  
POP, JZ, JMP, CALL, RET, REI

### Group B:



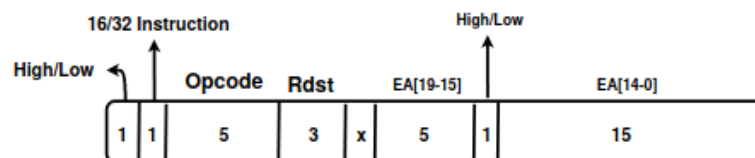
SHL, SHR

### Group C:



LDM, IADD

### Group D:



LDD, STD

\*\* Control Unit detailed design (each instruction and the control signals they generate) can be found [Here](#)

# Data and Control Hazards Analysis:

## One Operand Test Case:

### Data Hazards:

NOT R1	IN R2	NOT R2	DEC R2
NOP	NOT R2	INC R1	OUT R1
INC R1		DEC R2	OUT R2

```
NOT R1
NOP
INC R1
IN R1
IN R2
NOT R2
INC R1
DEC R2
OUT R1
OUT R2
```

### Control Hazards:

- No Control Hazards in one operand test case

## 1. No Forwarding Units or Hazard Detection Units

In this case we have to guarantee that the first instruction causing the hazards (The first in the Pipeline) enter the write back stage before the other one leave the decode stage. So there must be at least two instructions between the instructions causing the Hazards so the code must be like the following figure:

- Before adding NOP, the simulation is done at **14 Clock Cycles** but with some errors as shown in the image below:
  - R1 incremented from 0 not FFFF so the value was 1
  - R2 inverted from 0 not 10 so the value was FFFF

```
NOT R1
NOP
NOP #Stall
INC R1
IN R1
IN R2
NOP #Stall
NOP #Stall
NOT R2
INC R1
NOP #Stall
DEC R2
OUT R1
NOP #Stall
OUT R2
```

(0)	00000000	00000000													
(1)	00000006	00000000	FFFFFFF			00000001	00000005								
(2)	0000000F	00000000						00000010	FFFFFFF						

- After adding **5 NOP**, the simulation is done at **19 Clock Cycles** but with complete and right functionality.

(0)	00000000	00000000													
(1)	00000011	00000000	FFFFFFF						00000000						
(2)	FFFFFFEE	00000000								00000010	FFFFFFF				

## 2. Forwarding Unit

After adding the forwarding unit the code can work properly in all cases without adding any NOP as the operands can be forwarded from the memory stage or the write back stage to the execute stage.

Simulation is done at **14 Clock Cycles** with full functionality so we save **5 Clock Cycles** after adding the Forwarding units.

(0)	00000000	00000000													
(1)	00000006	00000000	FFFFFFF			00000000	00000005								
(2)	FFFFFFEE	00000000						00000010	FFFFFFF						

## 3. Hazard Detection Unit

It will save nothing as there is already no stalls after adding the forwarding unit.

## Two Operand Test Case:

### Data Hazards:

IADD R5,..    SUB R6,..    SHL R2,..    SHR R2,..    SWAP R2,..  
 ADD R4,..    AND .....,R6    SHR R2,..    SWAP R2,..    ADD R2,..  
 SUB .....,R4

### Control Hazards:

- No Control Hazards in two operand test case

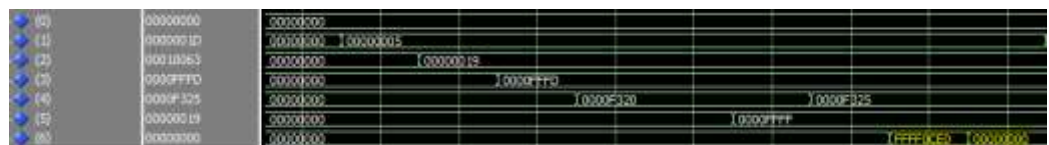
```
in R1
in R2
in R3
in R4
IADD R5,R3,2
ADD R4,R1,R4
SUB R6,R5,R4
AND R6,R7,R6
OR R1,R2,R1
SHL R2,2
SHR R2,3
SWAP R2,R5
ADD R2,R5,R2
```

## 1. No Forwarding Units or Hazard Detection Units

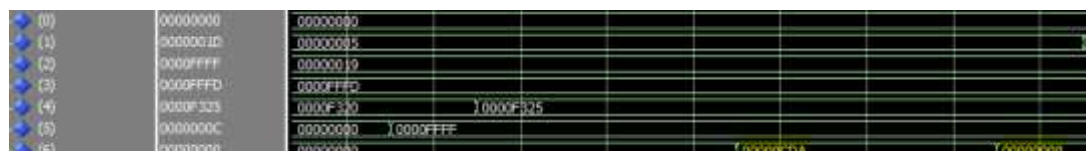
The same as One Operand we have to guarantee that the first instruction causing the hazards enter the write back stage before the other one leave the decode stage so the code must be like the following figure:

```
IN R1
IN R2
IN R3
IN R4
IADD R5,R3,2
ADD R4,R1,R4
NOP #Stall
NOP #Stall
SUB R6,R5,R4
NOP #Stall
NOP #Stall
AND R6,R7,R6
OR R1,R2,R1
SHL R2,2
NOP #Stall
NOP #Stall
SHR R2,3
NOP #Stall
NOP #Stall
SWAP R2,R5
NOP #Stall
NOP #Stall
ADD R2,R5,R2
```

- Before adding NOP, the simulation is done at **18 Clock Cycles** but with some errors as shown in the image below:
  - ADD R6,.. is calculated from the old values of both R4, R5 which is  $0000 - F320 = FFFF0CE0$
  - AND R6,..,R6 is calculated using  $R6 = 0$  so result = 0



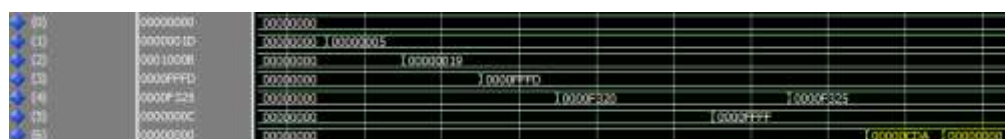
After adding **10 NOP**, the simulation is done at **28 Clock Cycles** but with complete and right functionality.



## 2. Forwarding Unit

The same as One Operand the code can work properly in all cases without adding any NOP because of Operands Forwarding.

Simulation is done at **18 Clock Cycles** with full functionality so we save **10 Clock Cycles** after adding the Forwarding units.



## 3. Hazard Detection Unit:

It will save nothing as there in no stalls.

## Memory Test Case:

### Data Hazards:

LDM R1,..      POP R2  
PUSH R1        STD R2,..

### Control Hazards:

- No Control Hazards in memory test case

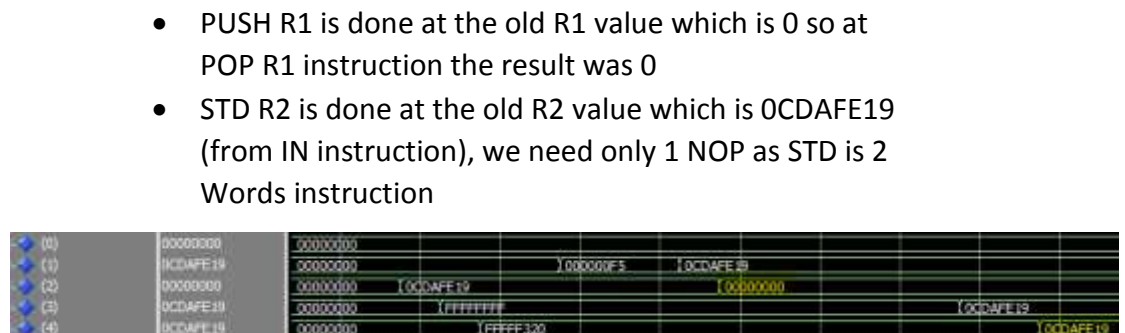
```
IN R2
IN R3
IN R4
LDM R1, F5
PUSH R1
PUSH R2
POP R1
POP R2
STD R2, 200
STD R1, 202
LDD R3, 202
LDD R4, 200
```

## 1. No Forwarding Units or Hazard Detection Units

The same as previous we have to guarantee that the first instruction causing the hazards enter the write back stage before the other one leave the decode stage so the code must be like the following figure:

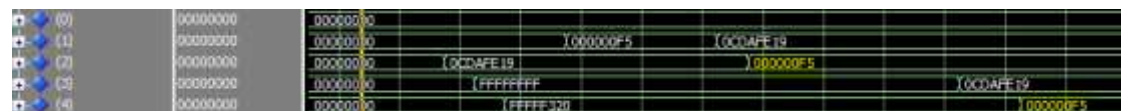
- Before adding NOP, the simulation is done at **21 Clock Cycles** but with some errors as shown in the image below:

```
IN R2
IN R3
IN R4
LDM R1, F5
NOP #Stall
NOP #Stall
PUSH R1
PUSH R2
POP R1
POP R2
NOP #Stall
STD R2, 200
STD R1, 202
LDD R3, 202
LDD R4, 200
```



- PUSH R1 is done at the old R1 value which is 0 so at POP R1 instruction the result was 0
- STD R2 is done at the old R2 value which is 0CDAFE19 (from IN instruction), we need only 1 NOP as STD is 2 Words instruction

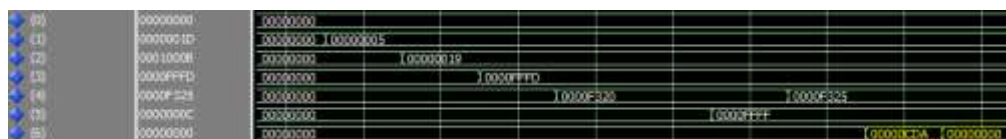
After adding **3 NOP**, the simulation is done at **24 Clock Cycles** but with complete and right functionality.



## 2. Forwarding Unit

The same as One Operand the code can work properly in all cases without adding any NOP because of Operands Forwarding.

Simulation is done at **21 Clock Cycles** with full functionality so we save **3 Clock Cycles** after adding the Forwarding units.



## 3. Hazard Detection Unit:

It will save nothing as there in no stalls.

## Branch Test Case:

### Data Hazards:

NOT R5,..      IN R6      POP R6,..  
INC R5,..      JZ R6,..      CALL R6,..

### Control Hazards:

JZ R2      JZ R3      JZ R6

## 1. No Forwarding , Hazard Detection or Branch Prediction Units:

Regarding Data Hazards, We have two kind of Data Hazards, The first one is the same as previous, The second one is regarding JMP, JZ, CALL instructions as they need to know the register value at fetch stage.

- Before adding NOP, the simulation is done Incorrectly as It jumped to wrong positions in the instruction memory:
  - JZ R6 doesn't jump as INC R5 doesn't set zero flag as expected (First Figure) and even if the zero flag is set, JZ R6 will jump to R6 old value which is FFFFFFFF not 200 from the IN instruction as R6 won't be changed yet, So we got Simulation error (Second Figure)

PC	0000005A	00000053	00000054	00000055	00000056	00000057	00000058	00000059
PC	FFFFFFF	0000...	00000052	00000053	00000054	00000055	00000056	00000057

Regarding Control Hazards, each Taken JZ instruction will cause fetching and executing one Invalid instruction after it **[only one as JZ is done at Decode Stage]**

- Before adding **NOP** after **JZ R2**, we can see that the instruction fetched at **PC = 32 (INC R7)** is executed which shouldn't be happen.

(7)	00000000	00000000	FFFFFFF				00000000
/cpu/Input_Port	00000200	00000200					
/cpu/Output_Port	00000000	00000000					
/cpu/Flags	0110	0000	0001	0000	0101	0100	0110
PC	00000103	00...	00000031	00000032	00000050	00000051	00000100

- After adding NOP in the appropriate positions as show in code figure on the left, Simulation is done at **40 Clock Cycles** with full functionality.

```
.ORG 10
IN R1
IN R2
IN R3
IN R4
IN R6
IN R7
Push R4
JMP R1
INC R7
.ORG 30
AND R5,R1,R5
JZ R2
NOP #Stall
INC R7
.ORG 50
JZ R3
NOP #Stall
NOT R5
NOP #Stall
NOP #Stall
INC R5
in R6
NOP #Stall
NOP #Stall
NOP #Stall
JZ R6
NOP #Stall
INC R1
POP R6
NOP #Stall
NOP #Stall
NOP #Stall
Call R6
INC R6
NOP
NOP
.ORG 300
Add R6,R3,R6
Add R1,R1,R2
ret
INC R7
.ORG 500
NOP
NOP
```

```
.ORG 10
IN R1
IN R2
IN R3
IN R4
IN R6
IN R7
Push R4
JMP R1
INC R7
.ORG 30
AND R5,R1,R5
JZ R2
INC R7
.ORG 500
NOP

.ORG 50
JZ R3
NOT R5
INC R5
in R6
JZ R6
INC R1
.ORG 200
POP R6
Call R6
INC R6
NOP
NOP
.ORG 300
Add R6,R3,R6
Add R1,R1,R2
ret
INC R7
```

## 2. Forwarding Unit

Adding the Forwarding Units will save some Clock Cycles:

- In the 1<sup>st</sup> Data Hazard on R5, the operands will be forwarded from the write back stage to the execute stage so the 2 NOP won't be required any more.
- One Cycle can be saved on the 2<sup>nd</sup> Data hazard which is on R6 as now R6 can be forwarded from the Memory Stage so we'll use 2 NOP instead of 3.

(6)	00000200	FFFFFFF					00000200
(7)	FFFFFFF	FFFFFFF					
/cpu/Input_Port	00000200	00000200					
/cpu/Output_Port	00000000	00000000					
/cpu/Flags	0100	0010	0101				0100
PC	00000202	00000054	00000055	00000056	00000057	00000058	00000200

- In the 3<sup>rd</sup> Data hazard which is also on R6, We can't save any Clock Cycles as POP R6 is a Memory instruction so we have to wait it till it reaches the Write Back stage.
- Control Hazards NOP will be as is.

Simulation is done at **37 Clock Cycles**

## 3. Hazard Detection Unit:

After adding Hazard Detection Unit:

- All Data Hazards NOP can be removed as now the stalls will be done automatically from the hardware.

(6)	00000401	FFFFFFF					00000200
(7)	FFFFFFF	FFFFFFF					
/cpu/Input_Port	00000200	00000200					
/cpu/Output_Port	00000000	00000000					
/cpu/Flags	0000	0010	0101				0100
PC	00000206	00000054	00000055		00000056		00000200

- Control Hazards NOP will be as is.

Simulation is done at **37 Clock Cycles**

## 4. Branch Prediction Flushing:

Now the flushing can be done using the hardware so there is no need for the NOP after each JZ instruction.

- Instruction fetched at **PC = 32 (INC R7)** isn't executed without adding **NOP** after **JZ R2**.

(7)	FFFFFFF	00000000	FFFFFFF				
/cpu/Input_Port	00000200	00000200					
/cpu/Output_Port	00000000	00000000					
/cpu/Flags	0000	0000	0001	0000			0010
PC	00000303	00000031	00000032	00000050	00000051	00000052	00000053

- One Cycle is saved using the Dynamic Branch Prediction which JZ R3 as it's not taken and the initial state is weakly not taken.

Simulation is done at **36 Clock Cycles**.

## Branch Prediction Test Case:

### Data Hazards:

```

LDM R3,    ADD R4,..,.. INC R0    INC R4    INC R0
LDM R4,    OUT R4      JMP R3    OUT R4    ADD ..,R0
JMP R3                      SUB ..,R0,..
1st and 3rd Data Hazards are repeated twice.
    
```

### Control Hazards:

```

JZ R1      JZ R3
    
```

```

.ORG 10      .ORG 50
LDM R2,0A    LDM R0,0
LDM R0,0     LDM R2,8
LDM R1,50    LDM R3,60
LDM R3,20    LDM R4,3
LDM R4,2     JMP R3
JMP R3       .ORG 60
.ORG 20      ADD R4,R4,R4
SUB R5,R0,R2 OUT R4
JZ R1        INC R0
ADD R4,R4,R4 AND R5,R0,R2
OUT R4       JZ R3
INC R0       INC R4
JMP R3       OUT R4
    
```

## 1. No Forwarding , Hazard Detection or Branch Prediction Units:

```

.ORG 10
LDM R2,0A
LDM R0,0
LDM R1,50
LDM R3,20
LDM R4,2
NOP #Stall
JMP R3
.ORG 20
SUB R5,R0,R2
JZ R1
NOP #Stall
ADD R4,R4,R4
NOP #Stall
NOP #Stall
OUT R4
INC R0
NOP #Stall
JMP R3
    
```

- Regarding Data Hazards, they are exactly the same as previous test cases, so we need to add multiple NOP as shown in the code in the left figure
- Regarding Control Hazards, each Taken JZ instruction will cause fetching and executing one Invalid instruction after it **[only one as JZ is done at Decode Stage]**
  - Here we have Two Loops, In the 1<sup>st</sup> Loop JZ will be not taken multiple times then it'll be taken at the last iteration so without adding NOP after JZ, R4 will be updated one more time which isn't required. Although **R5 = 0 (Termination Condition), R4 is Updated**

(4)	00000800	00000800	00001000
(5)	FFFFFFFF	FFFFFFFF	00000000

```

.ORG 50
LDM R0,0
LDM R2,8
LDM R3,60
LDM R4,3
NOP #Stall
JMP R3
.ORG 60
ADD R4,R4,R4
NOP #Stall
NOP #Stall
OUT R4
INC R0
NOP #Stall
NOP #Stall
AND R5,R0,R2
JZ R3
NOP #Stall
INC R4
NOP #Stall
NOP #Stall
OUT R4
    
```

- In the 2<sup>nd</sup> Loop JZ will be taken multiple times then it'll be not taken at the last iteration so without adding NOP R4 will be incremented multiple times Instead of being incremented one time after the last iteration. Although **R1 = 1 (Not Termination Condition), R4 is incremented before it's multiplied by 2.**

(0)	00000001	00000000	00000001
(1)	00000050	00000050	
(2)	00000008	00000008	
(3)	00000060	00000060	
(4)	0000000C	00000006	00000007 0000000C

- After adding NOP in the appropriate positions as show in code figure on the left, Simulation is done at **157 Clock Cycles** with full functionality.
- R4 in the 1<sup>st</sup> Loop wasn't updated one more time.

(4)	00000800	00000800	
(5)	00000000	FFFFFFFF	00000000



- R4 in the 2<sup>nd</sup> Loop wasn't incremented at each iteration and incremented only once at last iteration.

(0)	00000002	00000000	00000001			
(1)	00000050	00000050				
(2)	00000008	00000008				
(3)	00000060	00000060				
(4)	0000000D	00000006				0000000C

## 2. Forwarding Unit

Adding the Forwarding Units will eliminate all Data Hazards and only the NOP after each JZ will remain.

- Simulation is done at **147 Clock Cycles**

## 3. Hazard Detection Unit:

It will save nothing as there in no stalls.

## 4. Branch Prediction Flushing:

Now the flushing can be done using the hardware so there is no need for the NOP after each JZ instruction.

- In the 1<sup>st</sup> loop, the Dynamic Branch Prediction will save Clock Cycle at each iteration except the last one as it's always not taken and at the last iteration when it'll be taken, it'll be predicted as not taken.
- In the 2<sup>nd</sup> loop the Dynamic Branch Prediction will save Clock cycle for each Iteration except the 1<sup>st</sup> and last Iteration, the 1<sup>st</sup> Iteration will be predicated as not taken although it's taken and the last iteration will be predicted as taken although it's not taken.

Simulation is done at **131 Clock Cycles**.