Designing for pipeline picoProcessor

The design of the Datapath:

This pipeline processor consist from five stages and each stage has pipeline registers.

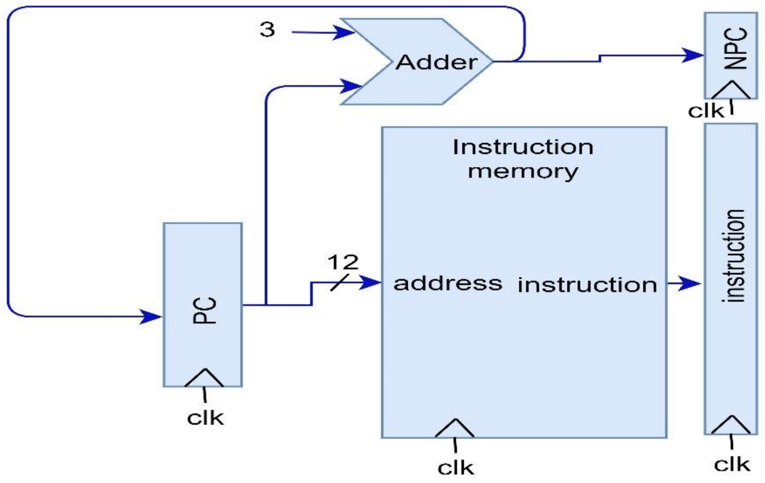
The stages:

1. IF: Instruction Fetch from instruction memory
2. ID: Instruction Decode, register read, and J/Br address
3. EX: Execute operation or calculate load/store address
4. MEM: Memory access for load and store
5. WB: Write Back result to register

The design of the stages:

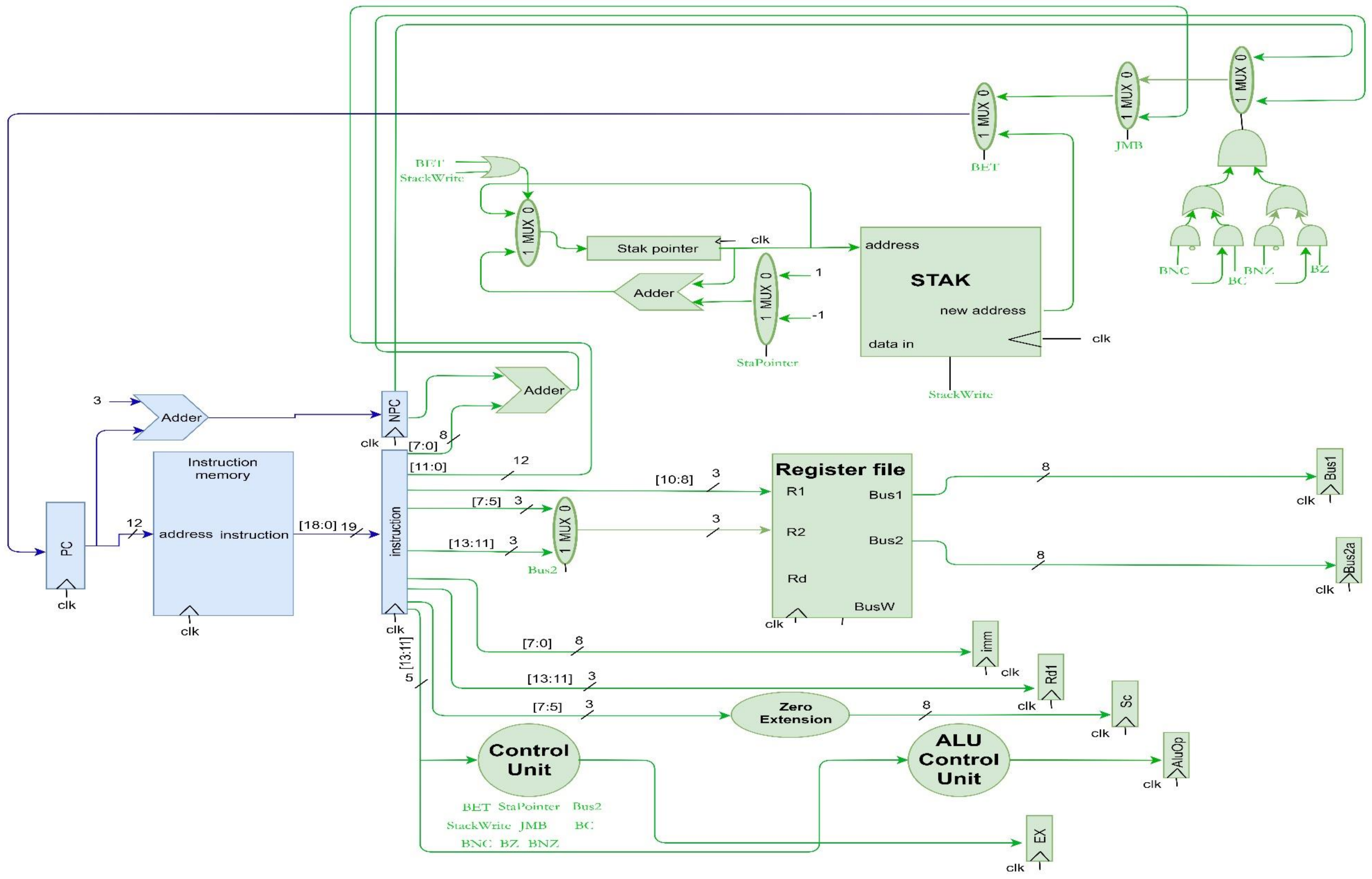
1. The design of IF stage:

In this stage the instruction fetched from the instruction memory and store it in the pipeline register (instruction) and calculate the address of the next instruction and store it in the pipeline register (NPC)



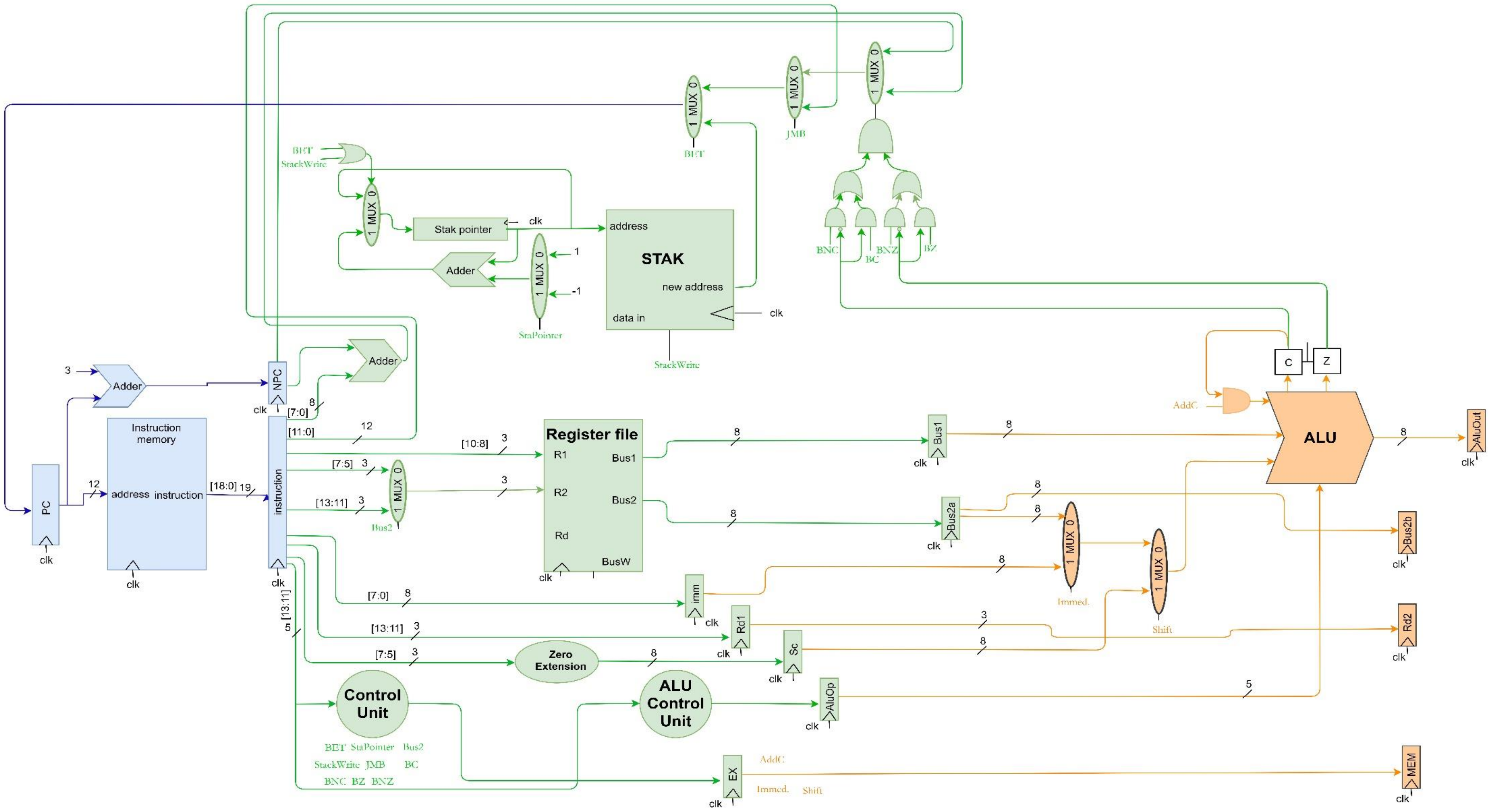
2. The design of ID stage:

In this stage: 1. the instruction decoded and the control signal generated and store the control signal foe next stages in pipeline register (EX). 2. The address of the next instruction determined. 3. The register file accessed for reading the data and store data in pipeline registers(Bus1,Bus2).



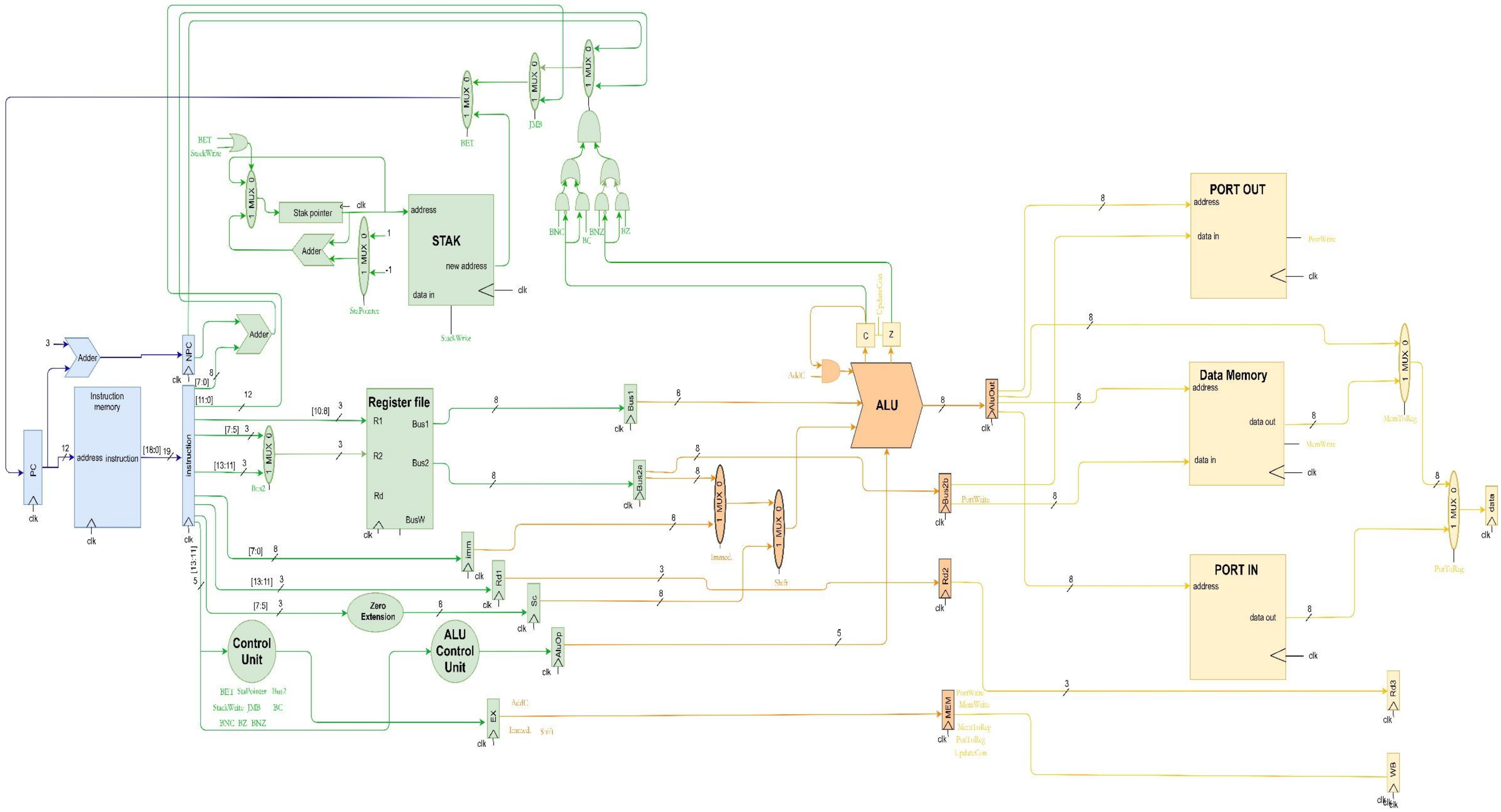
3. The design of the EX stage:

In this stage all the operation that need ALU executed and the result store in the pipeline register (AluOut)



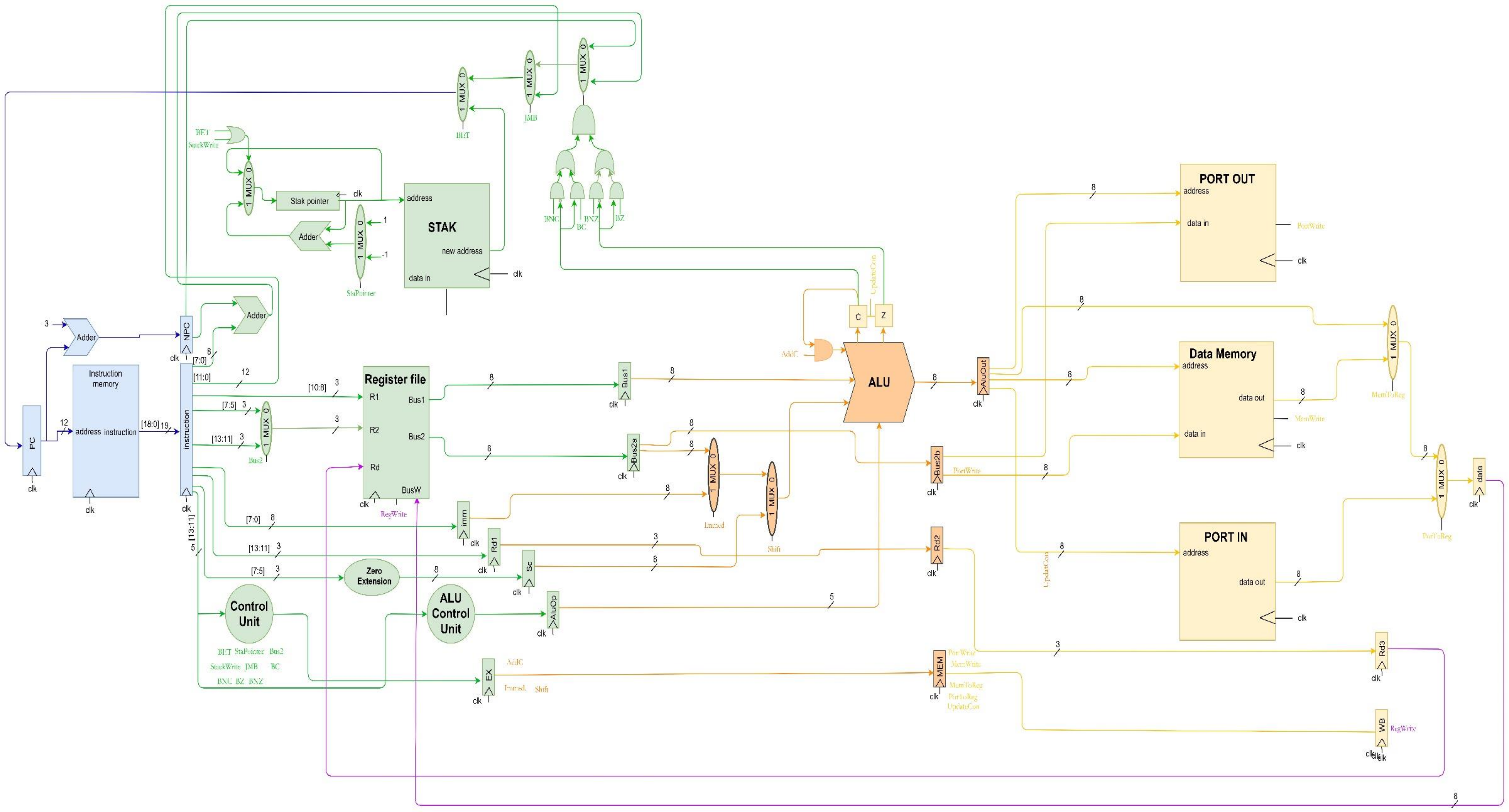
4. The design of MEM stage:

In this stage the data memory, Port in and port out accessed for reading or writing data and store it in the pipeline registers (data).



5. The design of the WB stage:

In this stage the data written in the file register in the destination register.



The pipeline registers:

The stage	The register	The function
IF	Instruction	To store the instruction to the ID stage
	NPC	To store the address of the next instruction in the instruction memory
ID	Bus1	To store the first operand of the ALU
	Bus2a	To store the second operand of ALU or the data that will store in data memory or PORT OUT
	Imm	To store the immediate value to use it in EX stage
	Sc	To store the Sc value to use it in EX stage
	Rd1	To store the address of the destination register to use it in WB stage
	AluOp	To store the control signals for the ALU
	EX	To store the control signal for next stages
EX	AluOut	To store the result of ALU to use it in MEM stage
	Bus2b	To store the data that will store in data memory or PORT OUT
	Rd2	To store the address of the destination register to use it in WB stage
	MEM	To store the control signal for next stages
MEM	data	To store the data that will be written in WB stage
	Rd3	To store the address of the destination register to use it in WB stage
	WB	To store the control signal for next stages

The design of the ALU Control Unit :

Input		Output	
ADD	00000	ADD	0000
ADDC	00001	ADD	0000
SUB	00010	SUB	0001
SUBC	00011	SUB	0001
AND	00100	AND	0010
OR	00101	OR	0011
XOR	00110	XOR	0100
MSK	00111	MSK	0101
ADDI	01000	ADD	0000
ADDCI	01001	ADD	0000
SUBI	01010	SUB	0001
SUBCI	01011	SUB	0001
ANDI	01100	AND	0010
ORI	01101	OR	0011
XORI	01110	XOR	0100
MSKI	01111	MSK	0101
SHL	11000	SHL	0110
SHR	11001	SHR	0111
ROL	11010	ROL	1000
ROR	11011	ROR	1001
LDM	10000	ADD	0000
STM	10001	ADD	0000
INP	10010	ADD	0000
OUT	10011	ADD	0000
BZ	10100	X	X
BNZ	10101	X	X
BC	10110	X	X
BNC	10111	X	X
JMP	11100	X	X
JSB	11101	X	X
RET	11110	X	X

The design of the Control Unit :

Input		output																	
		ID stage									EX stage				MEM stage				WB stage
		StackWrite	StaPointer	JMB	BET	BZ	BNZ	BC	BNC	Bus2	AddC	Immed	Shift	UpdateCon	MemWrite	PortWrite	PorToReg	MemToReg	RegWrite
ADD	00000	0	X	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1
ADDC	00001	0	X	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	1
SUB	00010	0	X	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1
SUBC	00011	0	X	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	1
AND	00100	0	X	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1
OR	00101	0	X	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1
XOR	00110	0	X	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1
MSK	00111	0	X	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1
ADDI	01000	0	X	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	1
ADDCI	01001	0	X	0	0	0	0	0	0	0	1	1	0	1	0	0	0	0	1
SUBI	01010	0	X	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	1
SUBCI	01011	0	X	0	0	0	0	0	0	0	1	1	0	1	0	0	0	0	1
ANDI	01100	0	X	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	1
ORI	01101	0	X	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	1
XORI	01110	0	X	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	1
MSKI	01111	0	X	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	1
SHL	11000	0	X	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1
SHR	11001	0	X	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1
ROL	11010	0	X	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1
ROR	11011	0	X	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1
LDM	10000	0	X	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	1
STM	10001	0	X	0	0	0	0	0	0	1	0	1	0	0	1	0	X	X	0
INP	10010	0	X	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	1
OUT	10011	0	X	0	0	0	0	0	0	1	0	1	0	0	0	1	X	X	0
BZ	10100	0	X	0	0	1	0	0	0	X	X	X	X	0	0	0	X	X	0
BNZ	10101	0	X	0	0	0	1	0	0	X	X	X	X	0	0	0	X	X	0
BC	10110	0	X	0	0	0	0	1	0	X	X	X	X	0	0	0	X	X	0
BNC	10111	0	X	0	0	0	0	0	1	X	X	X	X	0	0	0	X	X	0
JMP	11100	0	X	1	0	0	0	0	0	X	X	X	X	0	0	0	X	X	0
JSB	11101	1	0	1	0	0	0	0	0	X	X	X	X	0	0	0	X	X	0
RET	11110	0	1	0	1	0	0	0	0	X	X	X	X	0	0	0	X	X	0

The types of the hazard and its solution:

1. Data hazard:

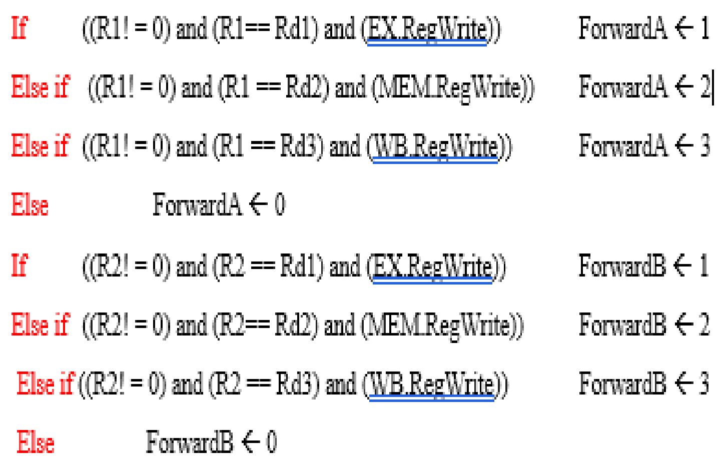
The data hazard occurs when the dependency between the instructions happened and there are three type of the dependency:

- The dependcy on the result of ALU:

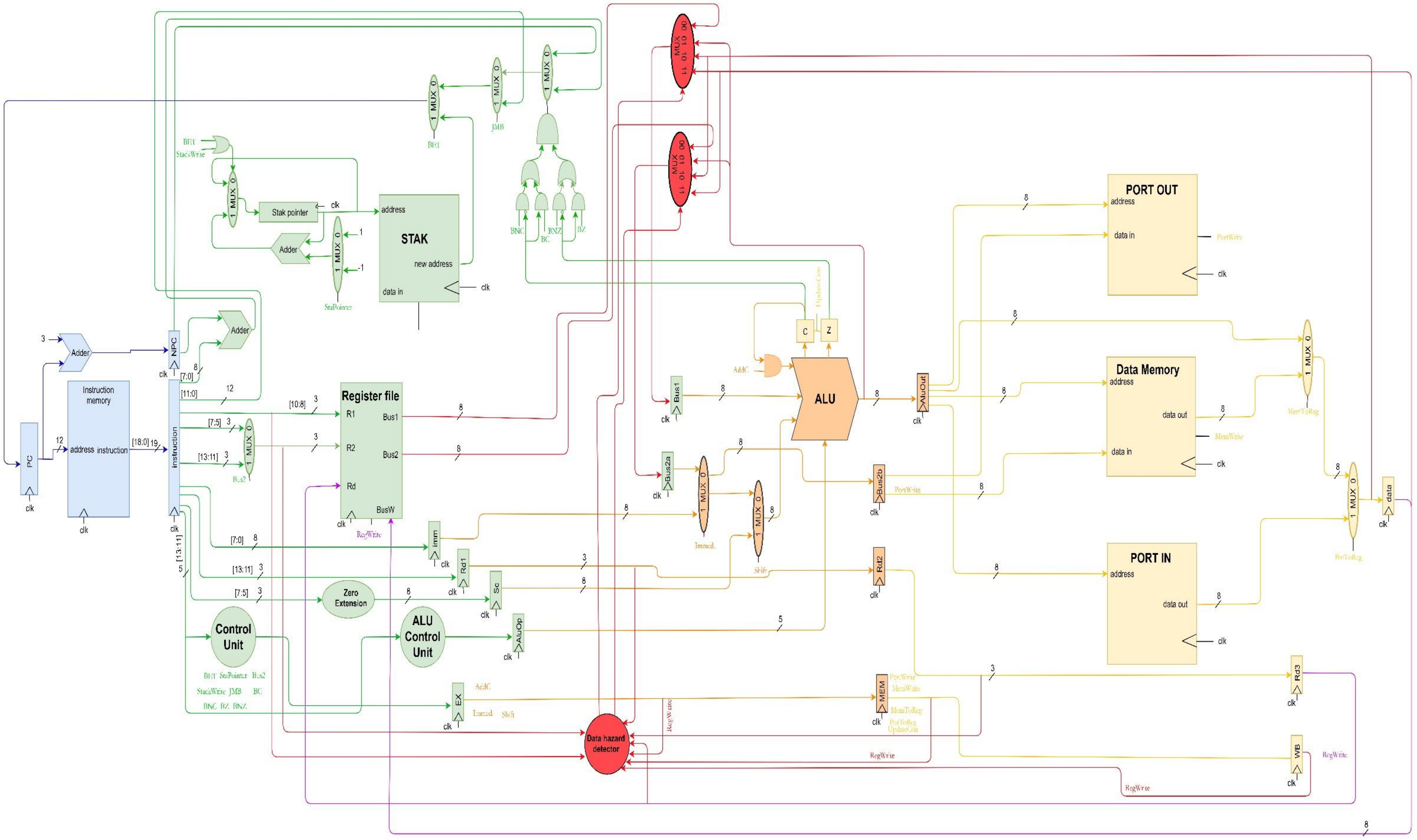
To solve this hazard we have two solution :

- Software solution : add two NOP instruction to Stall the processor two cycles until the data write in the file register (we stall just two cycle because we use write before read method) but it is not good solution because this delay will affect the performance of the processor.
- Hardwar solution: Add a detector and two 4-to-1multiplexors in our design to detect the dependcy then forward the data between stages and the design of it as following:

Signal	Explanation
ForwardA = 0	First ALU operand comes from register file = Value of (R1)
ForwardA = 1	Forward result of previous instruction to A (from ALU stage)
ForwardA = 2	Forward result of 2nd previous instruction to A (from MEM stage)
ForwardA = 3	Forward result of 3rd previous instruction to A (from WB stage)
ForwardB = 0	Second ALU operand comes from register file = Value of (R2)
ForwardB = 1	Forward result of previous instruction to B (from ALU stage)
ForwardB = 2	Forward result of 2nd previous instruction to B (from MEM stage)
ForwardB = 3	Forward result of 3rd previous instruction to B (from WB stage)



```
If       ((R1! = 0) and (R1== Rd1) and (EX.RegWrite))        ForwardA ← 1
Else if  ((R1! = 0) and (R1 == Rd2) and (MEM.RegWrite))      ForwardA ← 2
Else if  ((R1! = 0) and (R1 == Rd3) and (WB.RegWrite))       ForwardA ← 3
Else              ForwardA ← 0
If       ((R2! = 0) and (R2 == Rd1) and (EX.RegWrite))       ForwardB ← 1
Else if  ((R2! = 0) and (R2== Rd2) and (MEM.RegWrite))       ForwardB ← 2
Else if ((R2! = 0) and (R2 == Rd3) and (WB.RegWrite))        ForwardB ← 3
Else              ForwardB ← 0
```



PC
clk
12
Instruction memory
address
instruction
[18:0]
19
Adder
3
NPC
instruction
[7:0]
8
[11:0]
12
[10:8]
3
[7:5]
3
[13:11]
3
MUX
1
0
Bus2
Register file
R1
R2
Rd
Bus1
Bus2
BusW
RegWrite
8
imm
Rd1
Sc
AluOp
[7:0]
8
[13:11]
3
[7:5]
3
5
Zero Extension
Control Unit
ALU Control Unit
BH1
StaPointer
Bus2
StackWrite
JMB
BC
BNC
BZ
BNZ
EX
AddC
Immed.
Shift
Stak pointer
STAK
address
new address
data in
-1
StaPointer
BH1
StackWrite
JMB
BNC
BC
BNZ
BZ
00
01
10
11
Bus1
Bus2a
Immed.
Shift
AddC
C
Z
UpdateCon
ALU
AluOut
Bus2b
PortWrite
Rd2
MEM
PortWrite
MemWrite
MemToReg
PorToReg
UpdateCon
Data hazard detector
RegWrite
PORT OUT
address
data in
PortWrite
Data Memory
address
data out
MemWrite
data in
PORT IN
address
data out
MemToReg
PorToReg
data
Rd3
WB
RegWrite

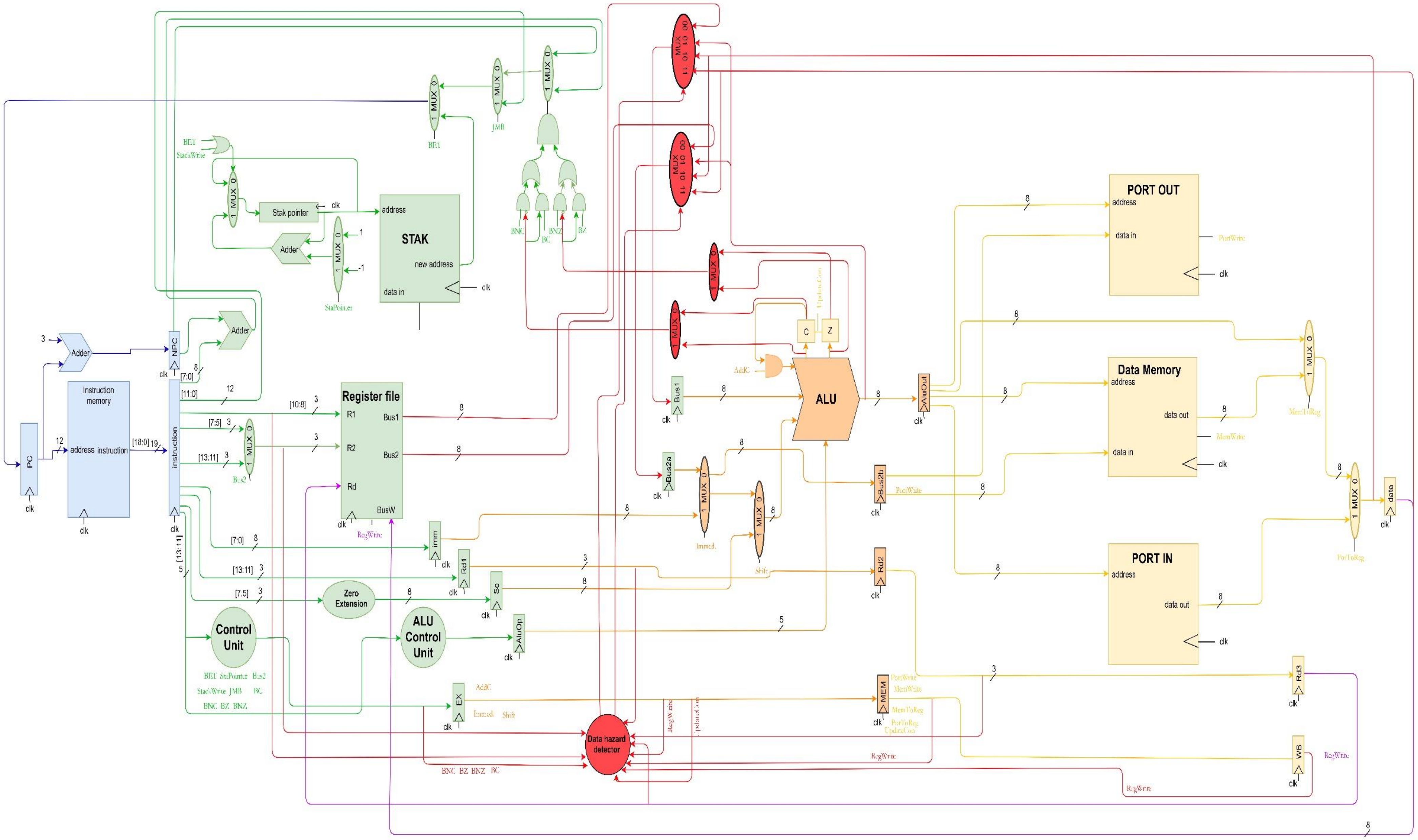
- The dependcy on Z or C:

The hazard occur in one case if the instruction that determine the value of Z or C in EX stage and the branch instruction in ID stage because C and Z updating in MEM stage. To solve this hazard we have two solution :

- Software solution: to add NOP instruction to delay branch instruction one cycle .
- Hardwar solution : to add a hazard detector and two multiplexors to our processor to detect this case and forward the value of C or Z in EX stage.

Signal	Explanation
ForwardZ = 0	Forward Z after written it in Z flipflop
ForwardZ = 1	Forward Z before written it in Z flipflop
ForwardC = 0	Forward C after written it in C flipflop
ForwardC = 1	Forward C before written it in C flipflop

```
If      (EX.UpdateCon.==1 and (ID. BZ or ID.BNZ))
ForwardZ ← 1
Else if        ForwardZ ← 0
If      (EX.UpdateCon.==1 and (ID. BC or ID.BNC))
ForwardC ← 1
Else if        ForwardC ← 0
```


PC
clk
Instruction memory
address
instruction
12
[18:0]
19
Adder
3
NPC
instruction
[7:0]
8
[11:0]
12
Adder
[10:8]
[7:5]
[13:11]
MUX
Bus2
Register file
R1
R2
Rd
Bus1
Bus2
BusW
RegWrite
imm
Rd1
Sc
AluOp
Zero Extension
Control Unit
ALU Control Unit
BIT StPointer Bus2
StackWrite JMB BC
BNC BZ BNZ
EX
AddC
Immed
Shift
BIT
StackWrite
Stak pointer
STAK
address
new address
data in
1
-1
StPointer
JMB
BIT
BNC
BC
BNZ
BZ
Bus1
Bus2a
Bus2b
ALU
C
Z
UpdateCon
AluOut
Rd2
MEM
PortWrite
MemWrite
MemToReg
PortToReg
UpdateCon
Data hazard detector
BNC BZ BNZ BC
RegWrite
PORT OUT
Data Memory
PORT IN
data out
MemWrite
MemToReg
PortToReg
data
Rd3
WB
5

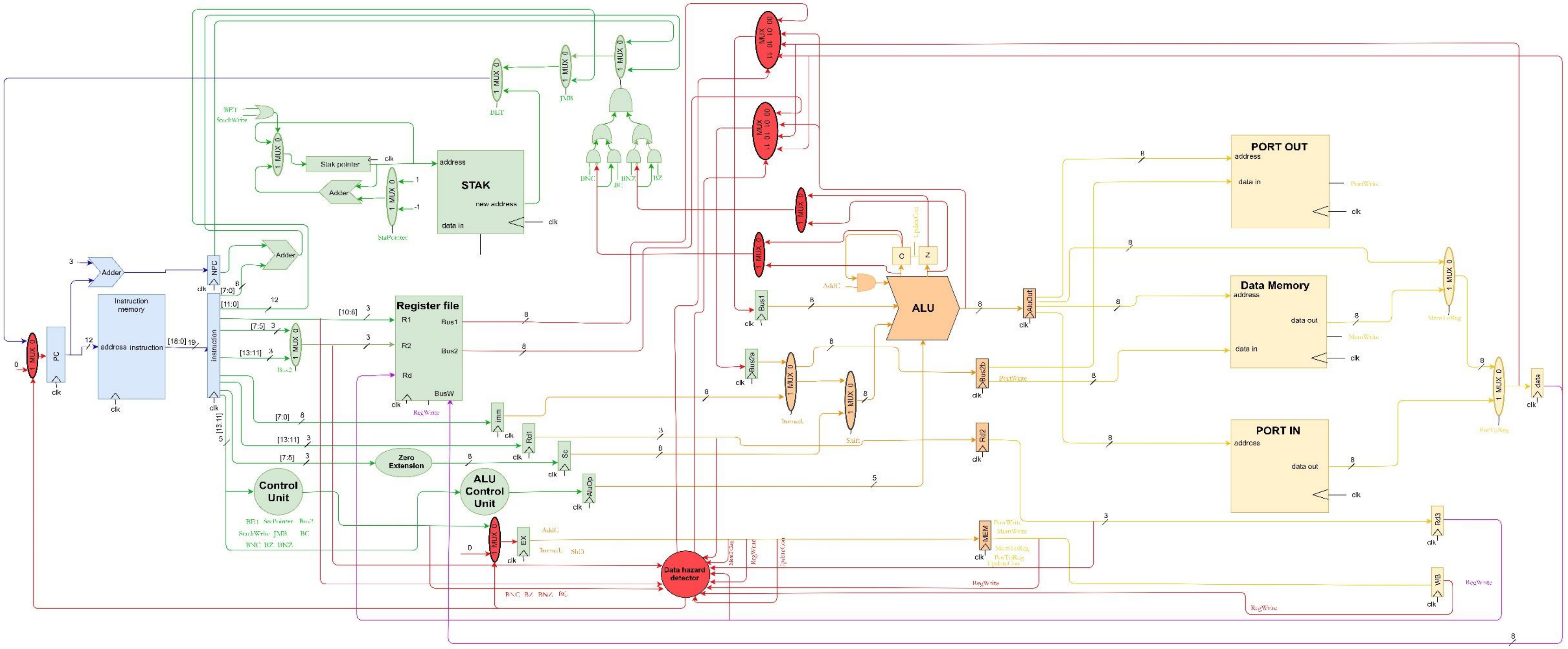
- The dependcy on the data of the memory or PORT IN:

To solve this hazard we have two solution :

- Software solution : same as the first type
- Hardwar solution: same as the first type put in one case we must stall the processor one cycle. This case if the instruction that has the data in EX stage and the instruction that need the data in ID stage:

If ((EX. MemToReg == 1) and (ForwardA==1 or ForwardB==1)) Stall

If ((EX. PorToReg == 1) && (ForwardA==1 or ForwardB==1)) Stall



2. The control hazard:

The control hazard occur in jump and branch instruction (BZ,BNZ,BC,BNCM,JMB,JSB,BST). The target determined in the second stage (ID). This cause one cycle delay because there is one instruction begin execute before the target instruction execute. The solution of this hazard is software solution by add a NOP instruction to delay the processor one cycle.