CMPE120 Final Project

Khai Nguyen
Camellia Bazargan
Jessabelle Ramos

Professor Jahan Ghofraniha
CMPE120 Section 01

Department of Computer Engineering
San Jose State University
May 17, 2020

Table of Contents

		page
I.	Introduction	2
II.	Problem Statement	4
III.	Design	4 - 17
	A. Overall Description	4
	B. Pipeline Diagram	5
	1. Instruction Fetch	6
	2. Instruction Decode	7
	3. Execute	8
	4. Memory Access	9
	5. Write-Back	10
	C. Pipeline Registers	11
	D. Design Steps	17
IV.	Project Breakdown	18
V.	Conclusion	18
VI.	References	19
VII.	Appendix	19

I. Introduction

Pipelining is an implementation technique in which multiple instructions are overlapped during execution. By overlapping the instructions during execution, the stages can operate concurrently, with the objective of achieving a low CPI with a high clock frequency. The speed-up from pipelining is approximately the number of pipe stages under ideal conditions and with a large number of instructions [2].

We adopt five stages of pipelining from MIPS to our project:

- Instruction Fetch
- Instruction Decode
- Execute
- Memory Access
- Write-Back

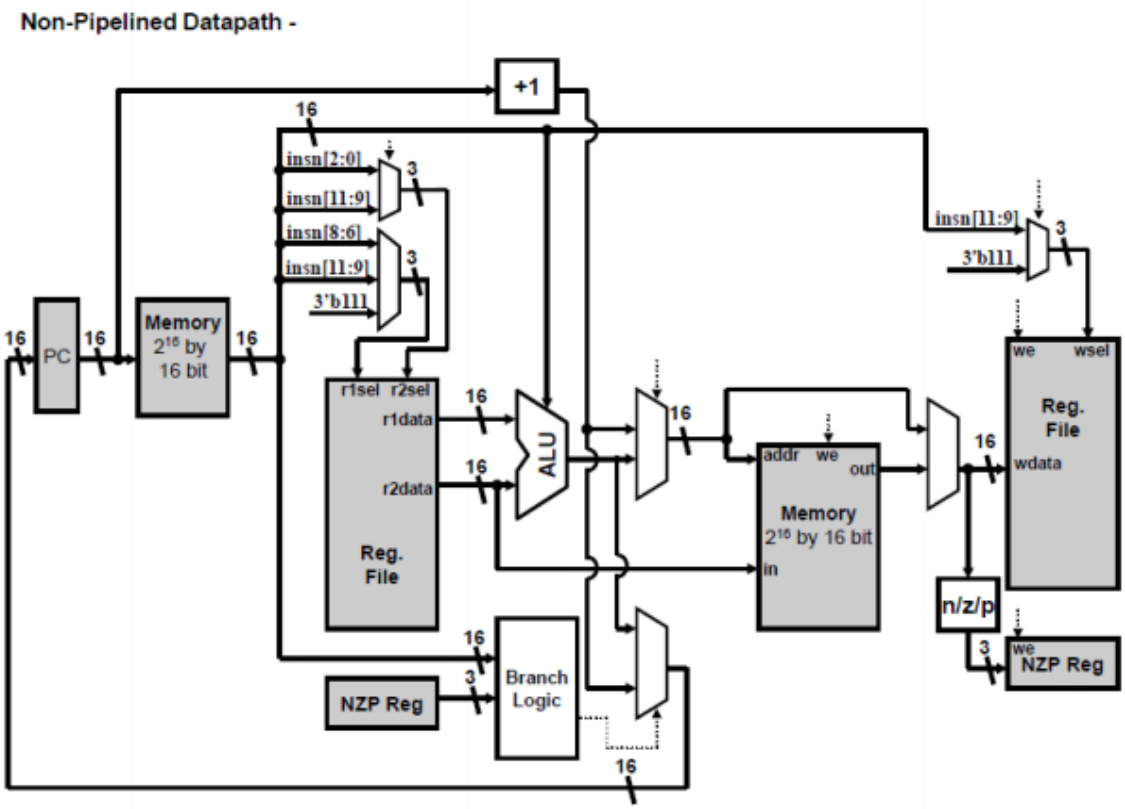


Figure 1.1 Single-Cycle CPU / Datapath [1]

Mnemonic	Semantics	Encoding
	Instructions	
NOP	--	0000000---------
BRn IMM9 <LABEL>	N ? PC = PC+1+SEXT(IMM9)	0000100IIIIIIIII
BRnz IMM9 <LABEL>	N|Z ? PC = PC+1+SEXT(IMM9)	0000110IIIIIIIII
BRnp IMM9 <LABEL>	N|P ? PC = PC+1+SEXT(IMM9)	0000101IIIIIIIII
BRz IMM9 <LABEL>	Z ? PC = PC+1+SEXT(IMM9)	0000010IIIIIIIII
BRzp IMM9 <LABEL>	Z|P ? PC = PC+1+SEXT(IMM9)	0000011IIIIIIIII
BRp IMM9 <LABEL>	P ? PC = PC+1+SEXT(IMM9)	0000001IIIIIIIII
BRnzp IMM9 <LABEL>	PC = PC+1+SEXT(IMM9)	0000111IIIIIIIII
ADD Rd, Rs, Rt	Rd = Rs+Rt	0001dddsss000ttt
MUL Rd, Rs, Rt	Rd = Rs*Rt	0001dddsss001ttt
SUB Rd, Rs, Rt	Rd = Rs-Rt	0001dddsss010ttt
DIV Rd, Rs, Rt	Rd = Rs/Rt	0001dddsss011ttt
ADD Rd, Rs, IMM5	Rd = Rs+SEXT(IMM5)	0001dddsss1IIIII
CMP Rs, Rt	NZP = signed-CC(Rs-Rt)	0010sss00 -----ttt
CMPU Rs, Rt	NZP = unsigned-CC(Rs-Rt)	0010sss01 -----ttt
CMPI Rs, IMM7	NZP = signed-CC(Rs-SEXT(IMM7))	0010sss10IIIIIII
CMPIU Rs, UIMM7	NZP = unsigned-CC(Rs-UIMM7)	0010sss11UUUUUUU
JSR IMM11 <LABEL>	R7 = PC+1; PC = (PC&x8000)|(IMM11<<4)	01001IIIIIIIIIII
JSRR Rs	R7 = PC+1; PC = Rs	01000--sss------
AND Rd, Rs, Rt	Rd = Rs&Rt	0101dddsss000ttt
NOT R1, R2	Rd = Rs	0101dddsss001---
OR R1, R2, R3	Rd = Rs|Rt	0101dddsss010ttt
XOR R1, R2, R3	Rd = Rs^Rt	0101dddsss011ttt
AND Rd, Rs, IMM5	Rd = Rs&SEXT(IMM5)	0101dddsss1IIIII
LDR Rd, Rs, IMM6	Rd = dmem[Rs+SEXT(IMM6)]	0110dddsssIIIIII
STR Rd, Rs, IMM6	dmem[Rs+SEXT(IMM6)] = Rd	0111dddsssIIIIII
RTI	PC = R7; PSR[15] = 0	1000----------
CONST Rd, IMM9	Rd = SEXT(IMM9)	1001dddIIIIIIIII
SLL Rd, Rs, UIMM4	Rd = Rs<<UIMM4	1010dddsss00UUUU
SRA Rd, Rs, UIMM4	Rd = Rs>>>UIMM4	1010dddsss01UUUU
SRL Rd, Rs, UIMM4	Rd = Rs>>UIMM4	1010dddsss10UUUU
MOD Rd, Rs, Rt	Rd = Rs%Rt	1010dddsss11-ttt
JMPR Rs	PC = Rs	11000--sss------
JMP IMM11 <LABEL>	PC = PC+1+SEXT(IMM11)	11001iiiiiiiiiii
HICONST Rd, UIMM8	Rd = (Rd&xFF)|(UIMM8<<8)	1101ddd1UUUUUUUU
TRAP UIMM8	R7 = PC+1; PC = (x8000|UIMM8); PSR[15] = 1	1111 ------UUUUUUUU
	Pseudo-instructions	
RET	JMPR R7	
LEA R1, <LABEL>	R1 = address of label	
LC R1, <LABEL>	R1 = constant at label	
	Assembly directives	
.DATA	current memory is data	
.CODE	current memory is code	
.ADDR UIMM16	set current address to UIMM16	
.FALIGN	pad current address to 16-word boundary	
.FILL IMM16	set value at current address to IMM16	IIIIIIIIIIIIIIII
.BLKW UIMM16	reserve UIMM16 words at current address	0000000000000000
.CONST IMM16	associate IMM16 with preceding label	
.UCONST UIMM16	associate UIMM16 with preceding label	

Figure 1.2 ISA for the given processor [1]

II. Problem Statement

Using the given single-cycle datapath in *Figure 1.1* and the instruction set architecture in *Figure 1.2*, we were tasked to design a pipelined version of this processor [1]. To do this, we needed to also consider how to implement bypass logic, stalling logic, branch predictors, flushing, as well as how to prevent or handle pipeline hazards.

III. Design

A. OVERALL DESCRIPTION

Our project required us to use a given single-cycle datapath and instruction set architecture to design a pipelined version of the processor. We added pipeline registers to the datapath, identifying and separating the different stages of pipelining. Next, we created control signals for different multiplexers to accommodate various instruction formats and added a control unit to pass control signals to different stages. We also added a forwarding unit to deal with data hazards.

B. PIPELINE DIAGRAM

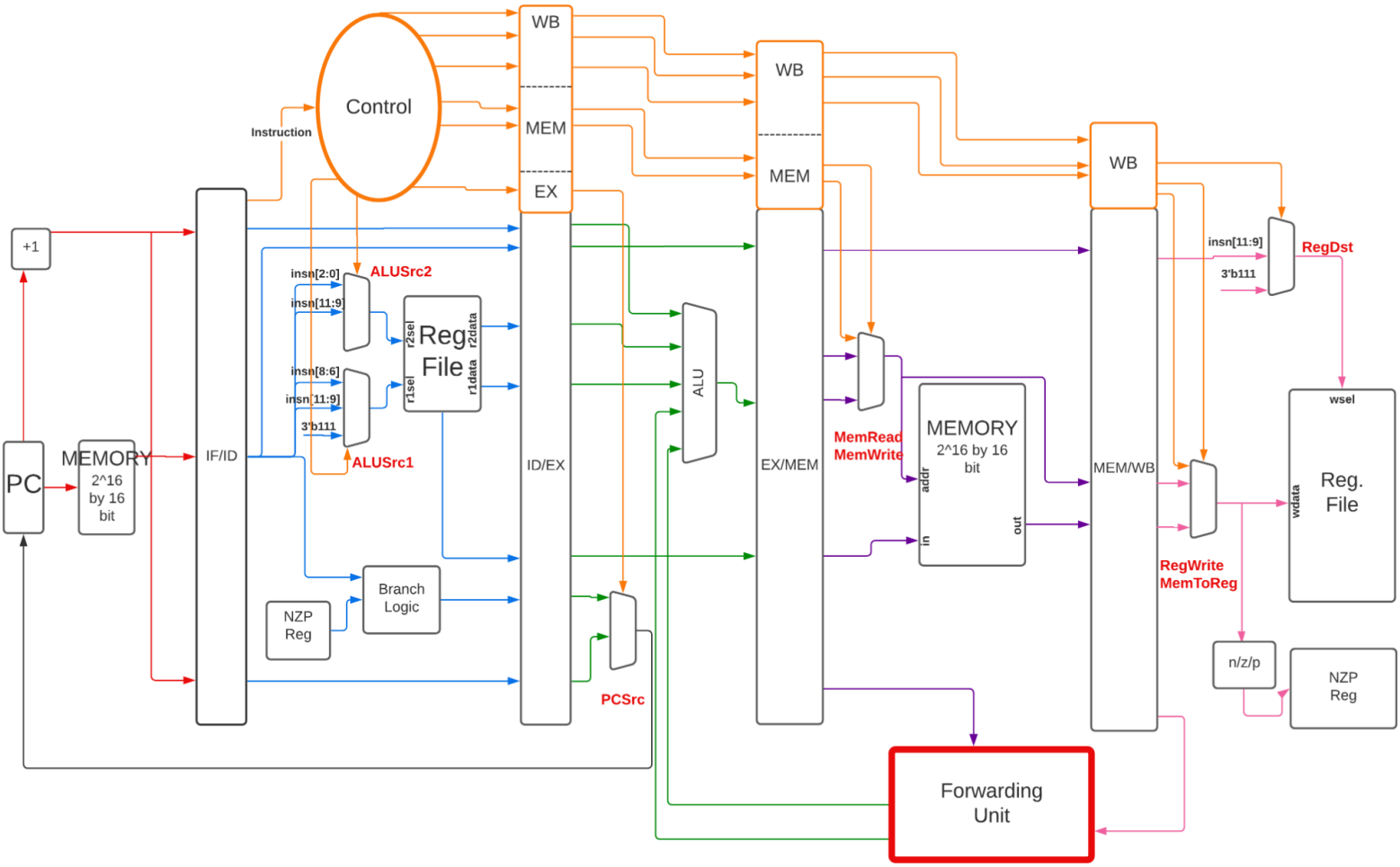
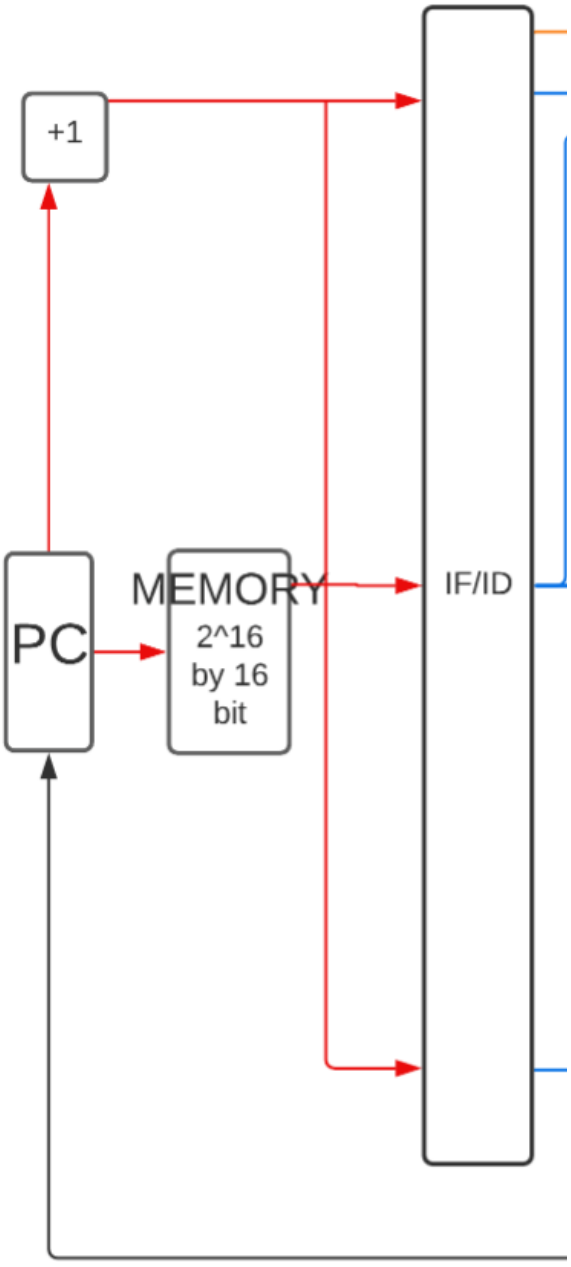


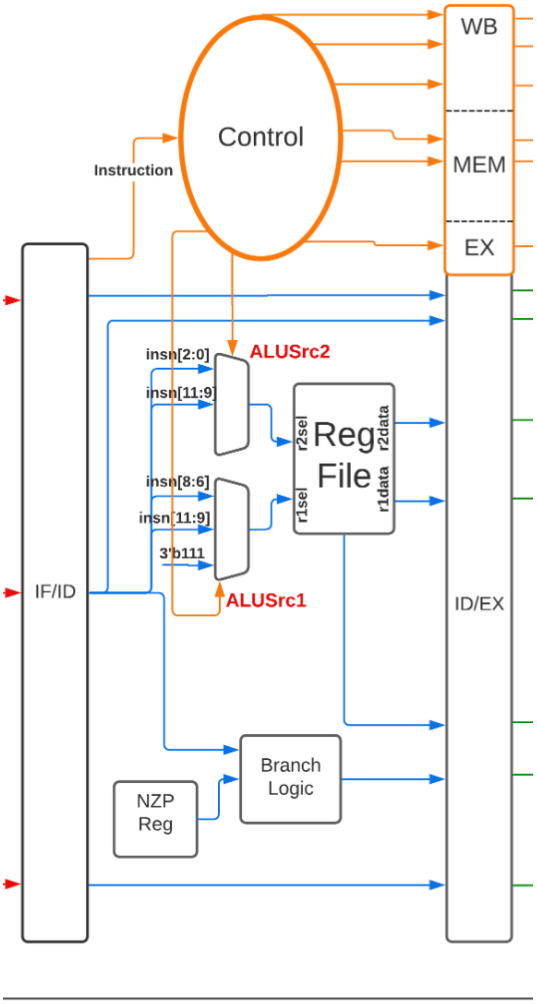
Figure 2.1 Pipeline Diagram

a. IF STAGE



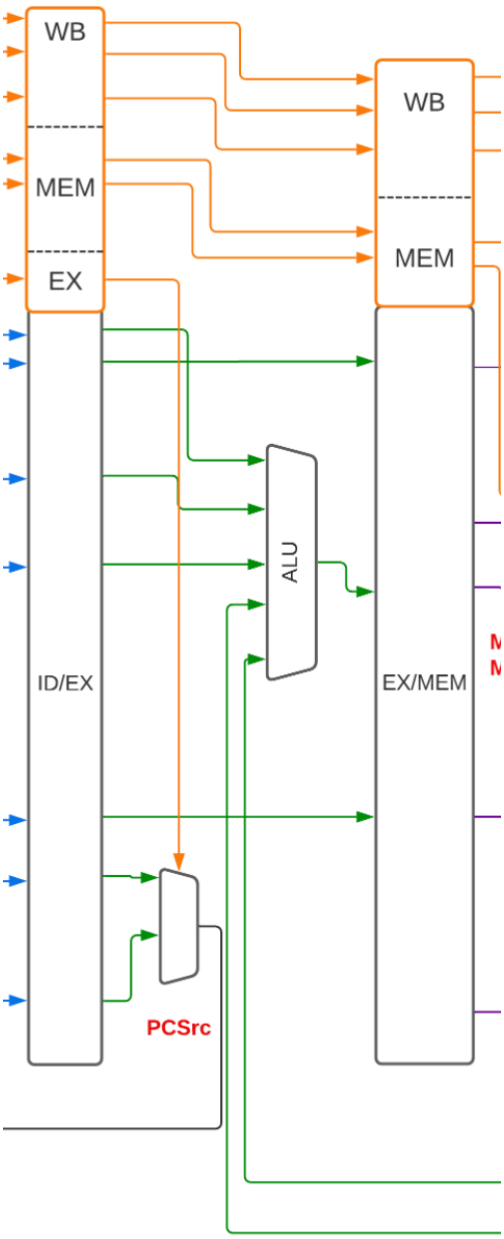
- **Instruction Fetch:**
  Instruction is read from the memory using the address in the PC and then placed in the IF/ID register. The PC address is incremented by 1 and passed down to branch multiplexor

b. ID STAGE



- **Instruction Decode:**
  - If it's not a branch instruction:
    - Depending on the type of instruction, the IF/ID pipeline register supplies register numbers to read the two registers or the immediate value instead of the second register, or nzp value to the branch logic. It will also pass the destination register address to the multiplexor if needed.
  - If it's conditional branching:
    - The nzp register will compare the passed n,z, p values and determine whether to jump to another instruction or not, the result is passed to the ID/EX register

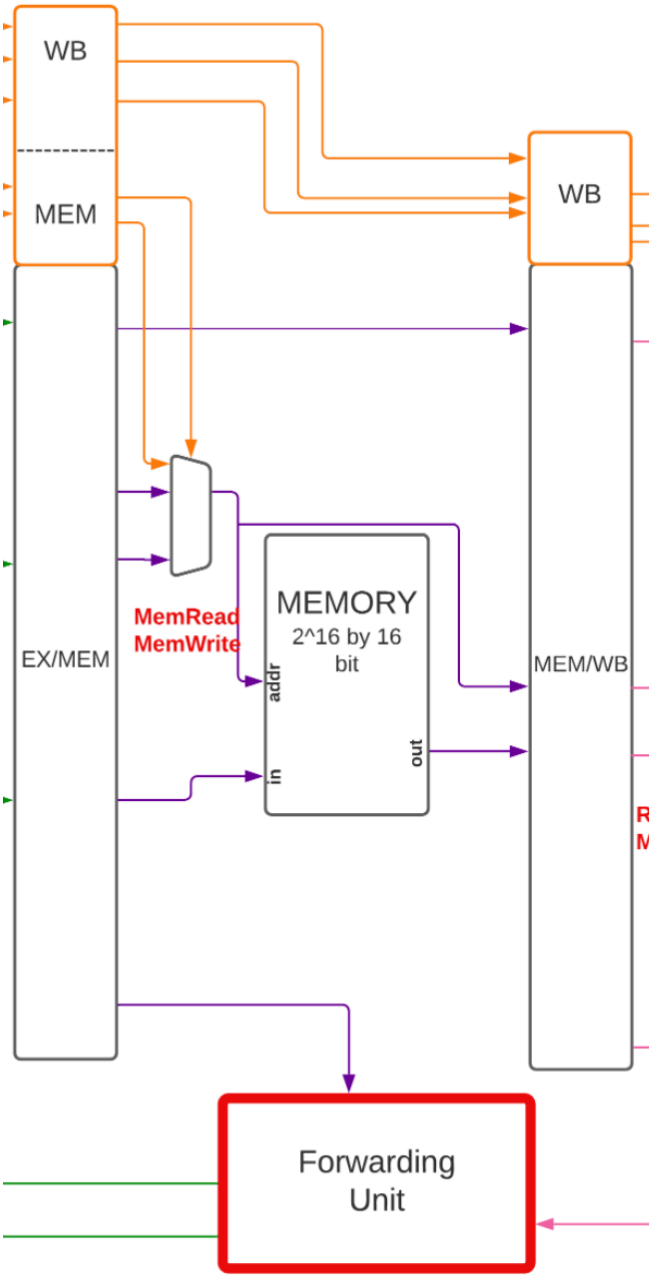
c. EX STAGE



- **Execute Calculation/Branching:**

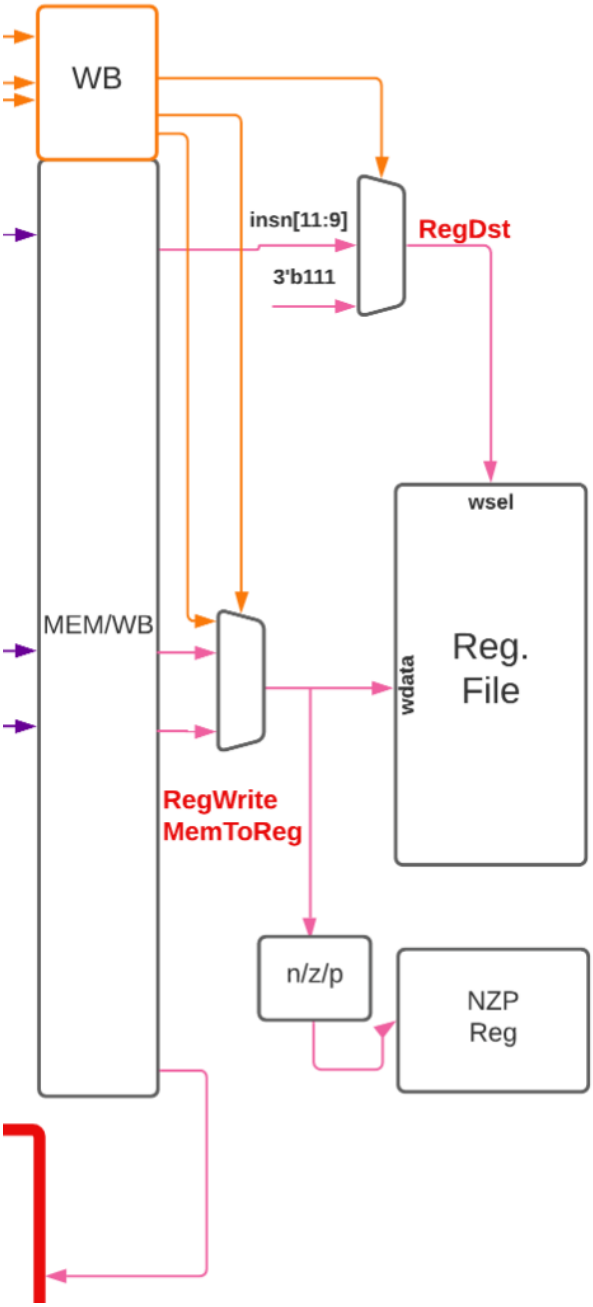
Depending on the type of instruction, the ALU will perform an operation and pass the result to the EX/MEM register. If there's a data hazard that can be solved by forwarding, the forwarding unit will forward results from EX/MEM or MEM/WB register back to ALU inputs for calculation. If it's a branch instruction, the multiplexor will determine the result and passed the next instruction address back to the PC

d. MEM STAGE



- **Memory Access:**
  Depending on the type of instruction, only the load and store need to access memory. Using the address from the EX/MEM pipeline, the load instruction reads the data memory and loads the data into the MEM/WB pipeline register. If it's a store instruction, the data will be read from the pipeline registers and placed into the memory.

e. WB STAGE



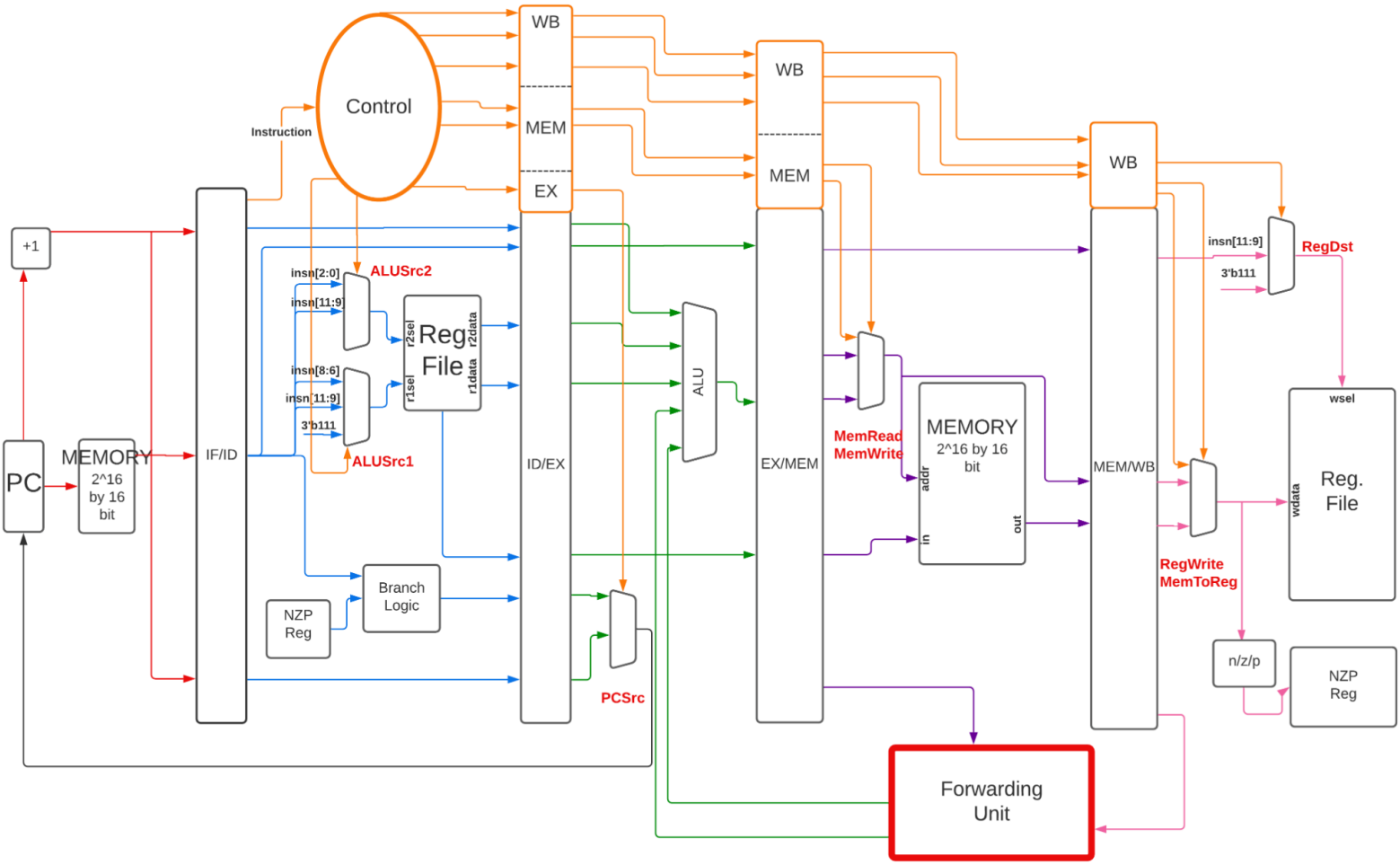
- **Write-Back:**
  Depending on the type of instruction, it will write back to the register. It will either write back to the destination register or register R7.

C. PIPELINE REGISTERS

What got passed between each stage:

Instruction stage	Value passed
Instruction fetch	PC+1 is passed. The whole instruction is passed since the computer doesn't know what type of instruction yet
Instruction decode	Now the computer knows what type of instruction it is and produces control signals based on the instruction. Depends on the type of instruction, it will send: ● Registers operand and register destination for ALU operation ● NZP value for branching ● Immediate values if the instruction specified ● PC+1 is passed as well ● Control signals for EX, MEM and WB depends on the instruction
Execute	The computer executing the instruction depends on its type. ● If it's an ALU operation, the ALU will perform operation and pass the result ● If it's load or store word, the ALU will calculate the memory address and passed the result ● If it's branching, the branch logic will do the computation and branch within the cycle. Otherwise PC+1 will be passed ● Control signals for MEM and WB depends on the instruction
Memory	Only load and store word need to access memory, which will dictate by the control signal ● If it does, access the memory address calculated by the ALU and passed the result ● Control signal for WB

Aside from the value needed to perform the instructions, we also need to pass control ALU op and control signals along each stage. We get some control signals from MIPS but also add new ones. Every multiplexor in this datapath needs at least one control signal.



Differences/similarities between this ALU and MIPS ALU:

- This ALU supports multiplication as well as division, while MIPS ALU does not.
- This ALU does *not* determine branching.
- We used ALUOp to have multiple levels of control inputs, similar to MIPS
    - 00 is left for ALU to decide
    - 01 is ADD, 10 is SUB, 11 is OR
- ALU Control Input: Since there are 8 actions, we will use 3 bits

Action	ALUOp	Instruction operation	ALU Control Input
ADD	00	ADD	000
MUL	00	MUL	001
SUB	00	SUB	010
DIV	00	DIV	011
AND	00	AND	100
NOT	00	NOT	101
OR	00	OR	110
XOR	00	XOR	111
LDR	01	ADD	000
STB	01	ADD	000
BR	01	ADD	000
CMP	10	SUB	010
JSR	11 or XX	OR	110
HICONST	11	OR	110
TRAP	10	SUB	010

Control Signal:

The difference between this multiplexor and MIPS is that MIPS multiplexores have two inputs while some multiplexors here have three inputs, therefore we will need three values in some cases.

Signal Name	00	01	10
PCSrc	The PC is replaced by the output of the adder that computes the branch target	The PC is replaced by value of PC+1	The PC is computed by nzp reg branch logic

ALUSrc1	The first ALU operand comes from [8:6]	The first ALU operand comes from [11:9]	The first ALU operand is R7
ALUSrc2	The second ALU operand comes from [2:0]	The second ALU operand comes from [11:9]	
RegWrite	None	The register on the Write register input is written with the value on the Write data input	
RegDst	The register destination number for write register comes from [11:9]	The register destination number for write register is R7	
MemRead	None	Data memory contents designated by the address input are put on the Read data output	
MemWrite	None	Data memory contents designated by the address input are replaced by the value on the Write data input	
MemtoReg	The value fed to the register Write data input comes from the ALU	The value fed to the register Write data input comes from the data memory	

	ALU Src1	ALU Src2	ALU OP1	ALU OP2	PCSrc	Mem Read	Mem Write	Reg Write	Mem toReg	Reg Dst
ALU	00	01 or 10	0	0	01	00	00	01	00	00
BR	XX	XX	0	1	00	00	00	00	X	X
JSR	10	XX	0	1	00	00	00	01	00	01
LDR	00	XX	0	1	01	01	00	01	01	00
STR	00	XX	0	1	01	00	01	00	X	X
CMP	00	00 or XX	1	0	01	00	00	00	X	X
HICONST	01	XX	1	1	01	00	01	01	00	00
TRAP	XX	10	1	0	01	00	00	01	00	01

For ALUSrc2, some instructions don't need a second ALU source register but rather the immediate values of the instructions therefore the control signal is XX for them. However, we don’t see any sign exten units for this datapath

D. DESIGN STEPS

- Forwarding / Bypassing Logic
    - Similar to MIPS, we implemented a forwarding unit to forward the ALU result instead of stalling the pipeline. [4]
    - Forwarding is used to help with read after write hazard, instead of waiting for the result to be available at the Write Back stage for the next read instruction, we can forward the calculated result from pipeline registers at EX and MEM stage back to the ALU input.
- Stalling Logic
    - Forwarding cannot solve the problem of the use after load case, therefore we need to stall. Similar to MIPS, we implemented stalled by inserting nops and set all control signals to 0. [4]
        - RegWrite, MemRead and MemWrite will all be 0, therefore there would be no register or memory written
- Branch Predictor
    - We will assume branch not taken. Unlike MIPS which determines branch prediction in ALU, this determines branch prediction with nzp registers and can be determined during the ID cycle and branch in the same EX cycle. However, we still need to stall for one cycle if branch not taken is false
- Flushing
    - When an unexpected event happens, instructions in a pipeline are discarded.
    - In order to do so, the original control values are changed to 0, the same way they are changed in stalling. The difference between stalling and flushing is that in flushing the instructions for the IF, EX, and ID stages also change when the branch reaches the MEM stage [5].

IV. Project Breakdown

A. Khai Nguyen:
    a. Overall Description
    b. Pipeline Diagram (IF, ID, and EX)
    c. Pipeline Registers
    d. Design Steps
B. Camellia Bazargan:
    a. Introduction
    b. Problem Statement
    c. Overall Description
    d. Pipeline Diagram (MEM and WB)
    e. Design Steps
    f. Conclusion
    g. References
C. Jessabelle Ramos:
    a. Introduction
    b. Problem Statement
    c. Pipeline Diagrams
    d. Pipeline Registers
    e. References

V. Conclusion

Our project involved using a given single-cycle datapath and instruction set architecture to design a pipelined version of the processor. To do this, we needed to consider how to implement bypass logic, stalling logic, branch predictors, and flushing, as well as how to prevent or handle pipeline hazards. We added pipeline registers to the datapath and identified the stages of pipelining (Instruction Fetch, Instruction Decode, Execute, Memory Access, and Write-Back), as well as a forwarding unit. We also implemented a control unit, which sends control signals to the multiplexores in the datapath.

VI. References

1. Ghofraniha, J. (2021, April 12). *CMPE120FinalProject* [PDF]. San Jose State University: Jahan Ghofraniha.

2. Patterson, D. A., & Hennessy, J. L. (2014). Chapter 4.5 Pipelined Datapath and Control. In 1377749194 1006710366 T. Green (Ed.), *Computer Organization and Design: The Hardware/Software Interface* (5th ed., pp. 286-303). Elsevier.

3. Patterson, D. A., & Hennessy, J. L. (2014). Chapter 4.6 Pipelined Datapath and Control. In 1377749194 1006710366 T. Green (Ed.), *Computer Organization and Design: The Hardware/Software Interface* (5th ed., pp. 286-303). Elsevier.

4. Patterson, D. A., & Hennessy, J. L. (2014). Chapter 4.7 Data Hazards: Forwarding versus Stalling. In 1377749194 1006710366 T. Green (Ed.), *Computer Organization and Design: The Hardware/Software Interface* (5th ed., pp. 303-316). Elsevier.

5. Patterson, D. A., & Hennessy, J. L. (2014). Chapter 4.8 Control Hazards. In 1377749194 1006710366 T. Green (Ed.), *Computer Organization and Design: The Hardware/Software Interface* (5th ed., pp. 316-325). Elsevier.

6. Wang, Z. (2021, May 15). *Lecture 8: Pipelining: Datapath and Control*. Lecture presented in Florida State University.

VII. Appendix

- The LC-3b ISA was a helpful resource that helped us understand the instruction of the processor: LC3b_ISA.pdf (illinois.edu)