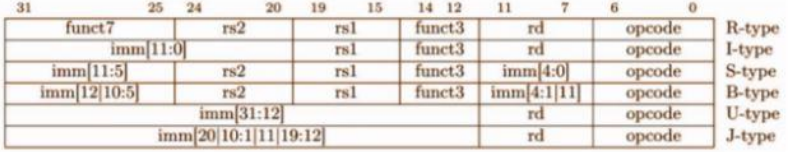
ECE M116C / CS 151B: Week 2 Section

Justin Feng

RV32I Base Instruction Set (MIT 6.004 subset)

imm[31:12]				rd	0110111	LUI
imm[20|10:1|11|19:12]				rd	1101111	JAL
imm[11:0]		rs1	000	rd	1100111	JALR
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	BEQ
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	BNE
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011	BLT
imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011	BGE
imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011	BLTU
imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011	BGEU
imm[11:0]		rs1	010	rd	0000011	LW
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	SW
imm[11:0]		rs1	000	rd	0010011	ADDI
imm[11:0]		rs1	010	rd	0010011	SLTI
imm[11:0]		rs1	011	rd	0010011	SLTIU
imm[11:0]		rs1	100	rd	0010011	XORI
imm[11:0]		rs1	110	rd	0010011	ORI
imm[11:0]		rs1	111	rd	0010011	ANDI
0000000	shamt	rs1	001	rd	0010011	SLLI
0000000	shamt	rs1	101	rd	0010011	SRLI
0100000	shamt	rs1	101	rd	0010011	SRAI
0000000	rs2	rs1	000	rd	0110011	ADD
0100000	rs2	rs1	000	rd	0110011	SUB
0000000	rs2	rs1	001	rd	0110011	SLL
0000000	rs2	rs1	010	rd	0110011	SLT
0000000	rs2	rs1	011	rd	0110011	SLTU
0000000	rs2	rs1	100	rd	0110011	XOR
0000000	rs2	rs1	101	rd	0110011	SRL
0100000	rs2	rs1	101	rd	0110011	SRA
0000000	rs2	rs1	110	rd	0110011	OR
0000000	rs2	rs1	111	rd	0110011	AND



Ex) Machine Code for JAL

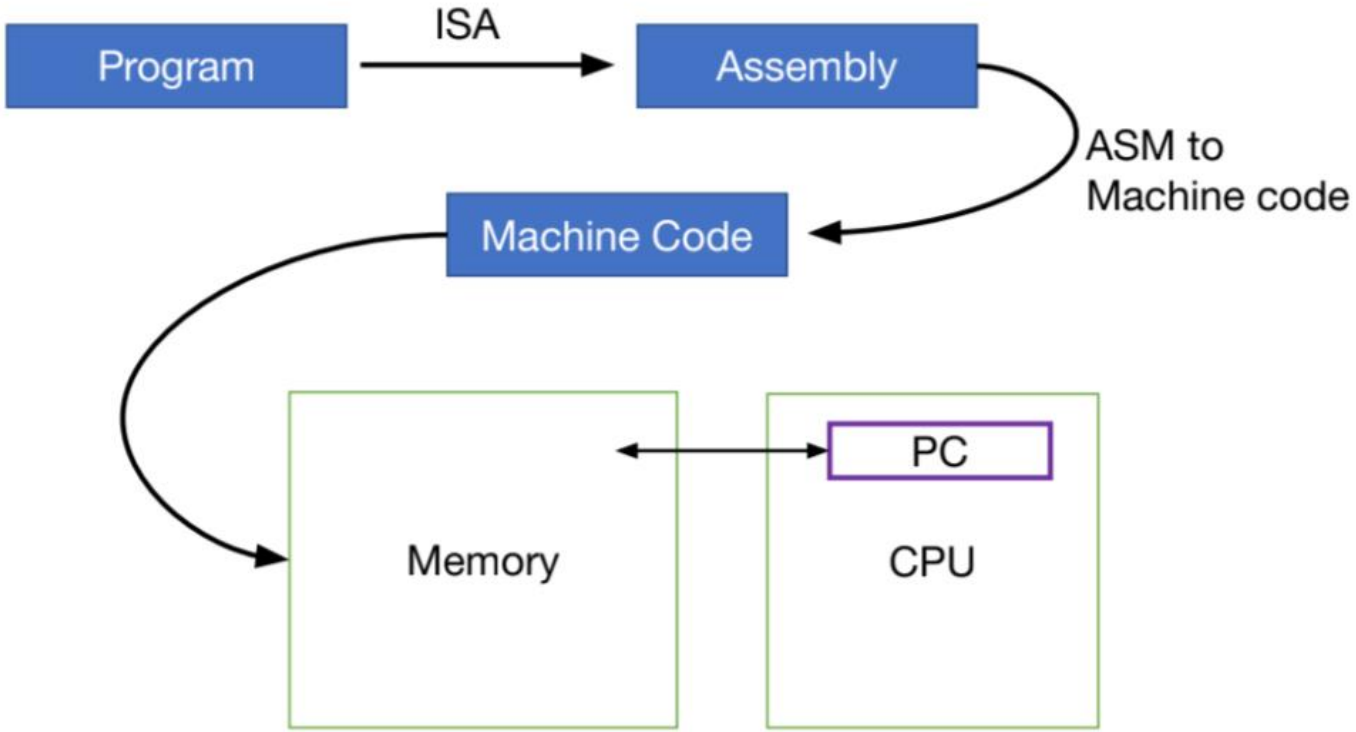
-Write the machine code for: jal ra, -896

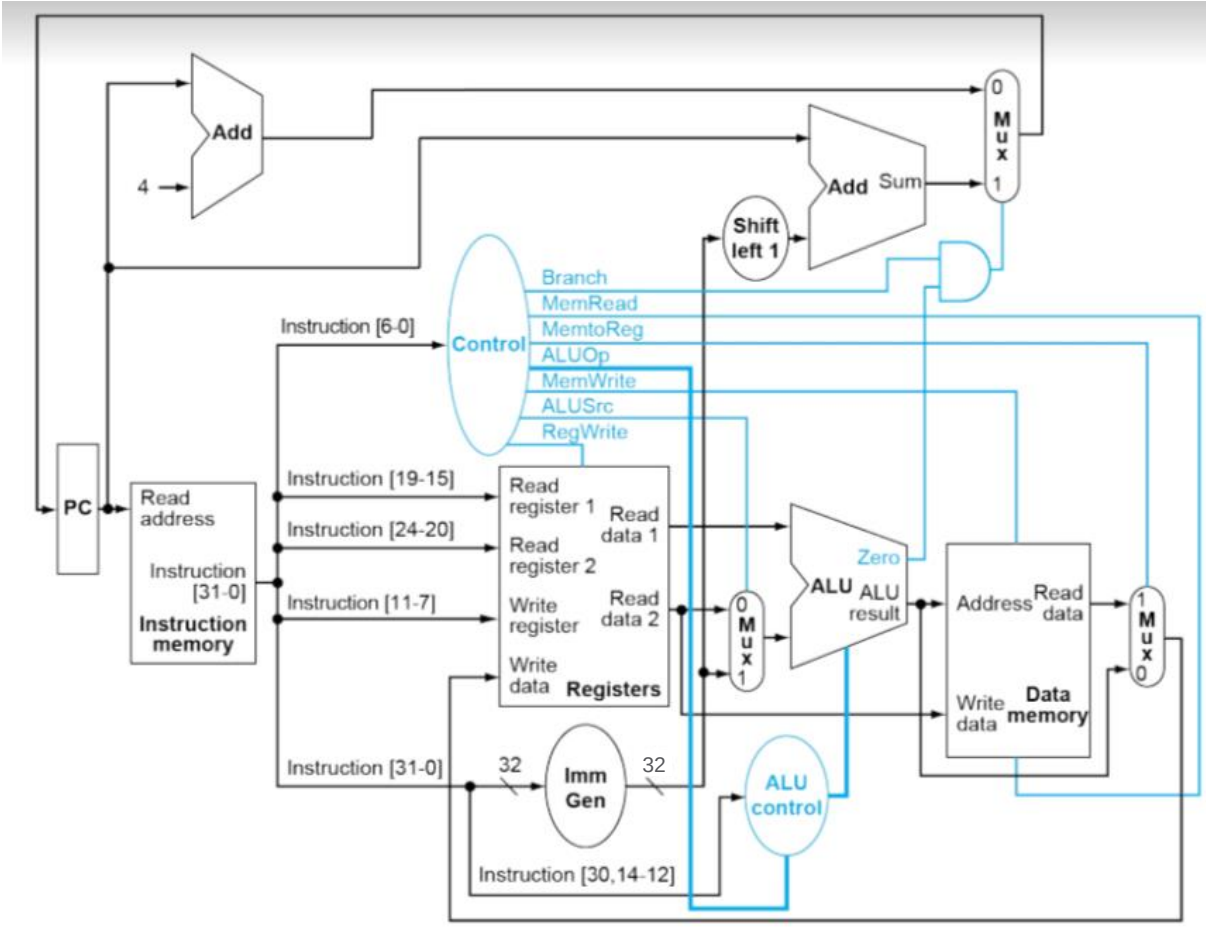
Ex) Machine Code for JAL

-Write the machine code for: jal ra, -896

Ans:
--86 to binary (two's complement):
11111111110010000000
20^{th} bit -> 1
10:1 -> 1001000000
11^{th} bit -> 1
19:12 -> 11111111
11001000000111111111 00001 1101111

Von Neuman Model





Datapath +
Controller

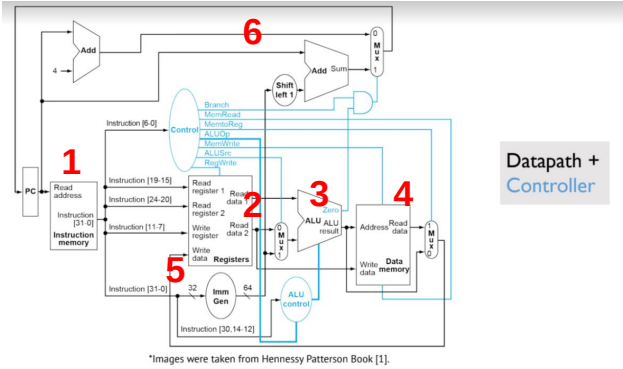
*Images were taken from Hennessy Patterson Book [1].

Single Cycle CPU

Steps in a CPU

1) Fetch Instruction
2) Decode (register)
3) Execution (ALU)
4) Memory
5) Writeback
6) Update PC

-In single cycle cpu, all these steps finish in one cycle!
-Note: order of steps does not indicate in which order they finish



*Images were taken from Hennessy Patterson Book [1].

Datapath

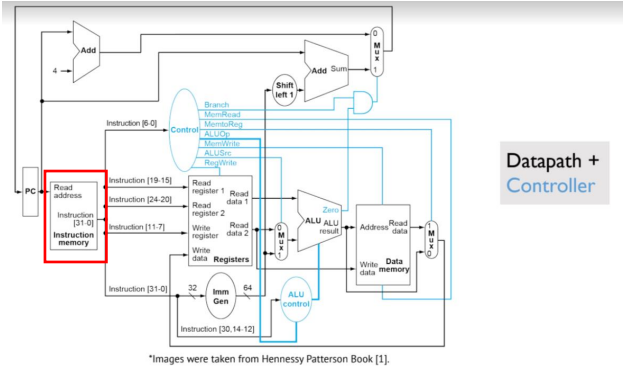
Instruction Memory

-Stores each of our 32 bit instructions

-PC indexes into instruction memory

-PC + 4: the next instruction

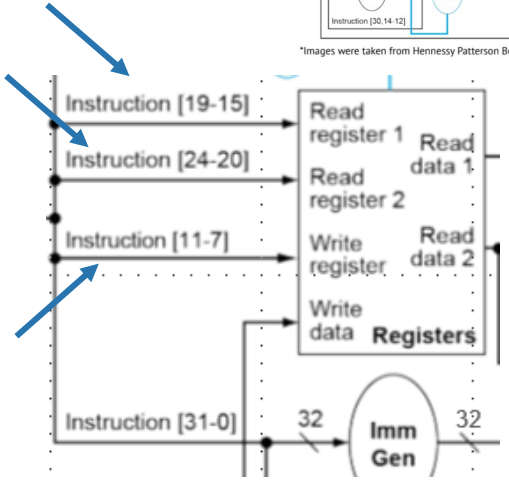
-PC + Offset: the instruction to access after a jump/branch

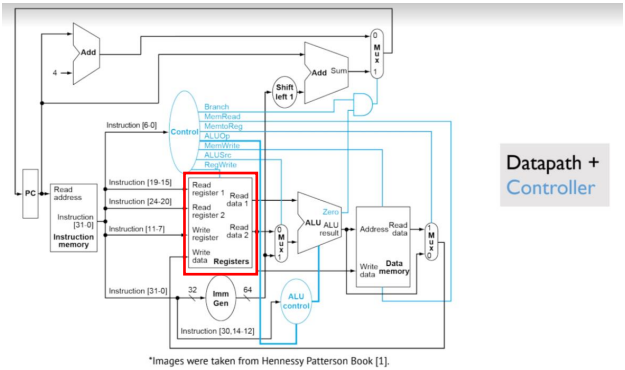


*Images were taken from Hennessy Patterson Book [1].

Register File

-Stores our 32 registers of size 32 bits each for easy access/write.
-Different instruction bits go to different ports
-RS1 address, RS2 address, Rd address
-Address ports used to access values of specific registers
-Rd address specifies which register we should write to after our operation

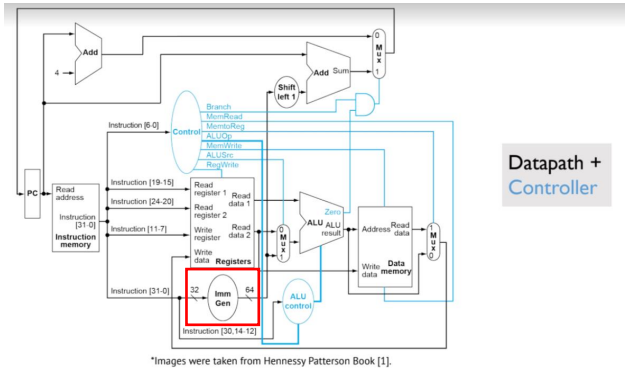




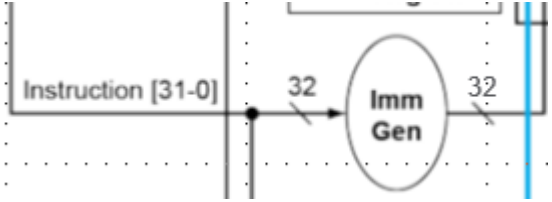
*Images were taken from Hennessy Patterson Book [1].

Imm Gen

-Given a 32 bit instruction, need to construct a 32 bit immediate
-Sign extension
-Use knowledge of full instruction to assign bits to bit positions

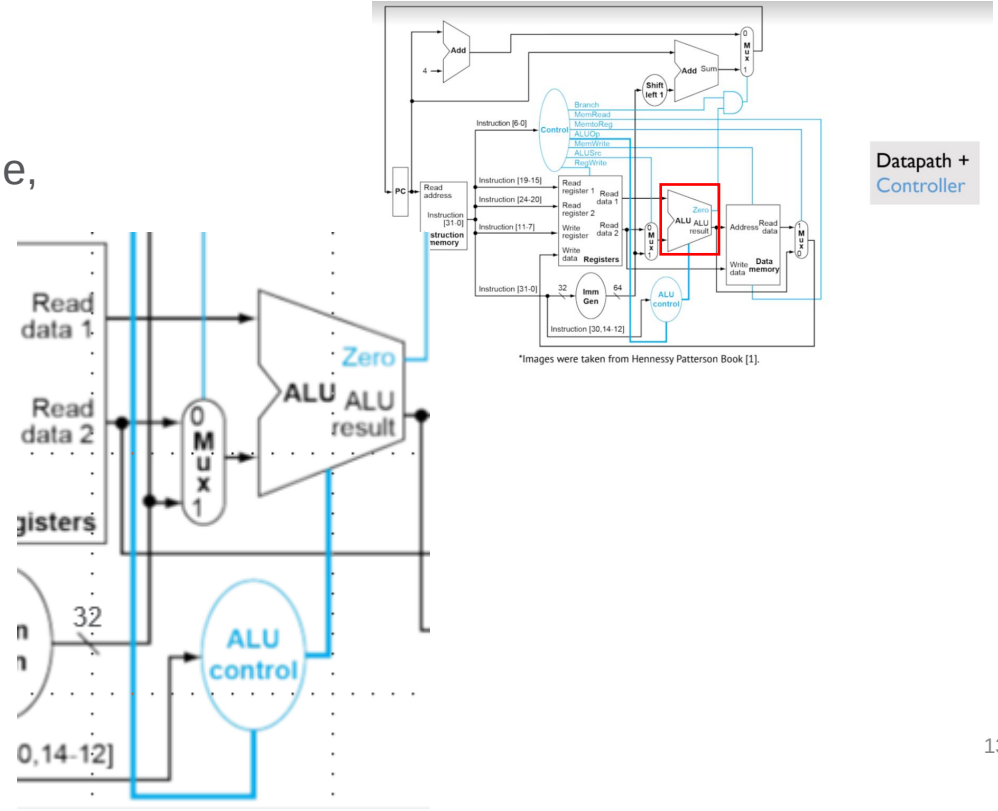


*Images were taken from Hennessy Patterson Book [1].



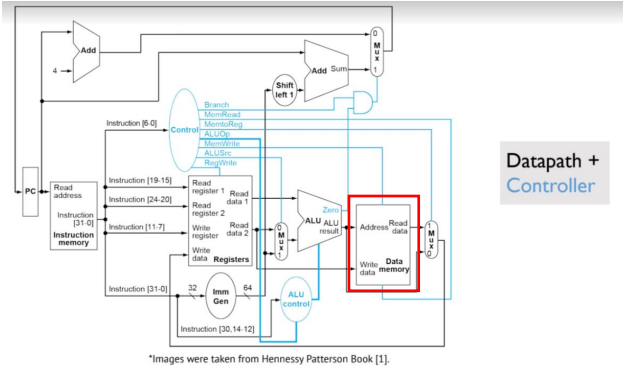
Arithmetic Logic Unit

-Does basic math operations
-Addition, subtraction, logical, etc
-Zero flag: if a branch condition is true, set the zero flag to true such that we know to branch
-ALU Control: used to select which operation to do



Data Memory

-Stores data in longer term storage
-Byte addressable
-Accessed via loads, with result stored in register file
-Modified via stores, given an address (computed by ALU) and the data (from register file)



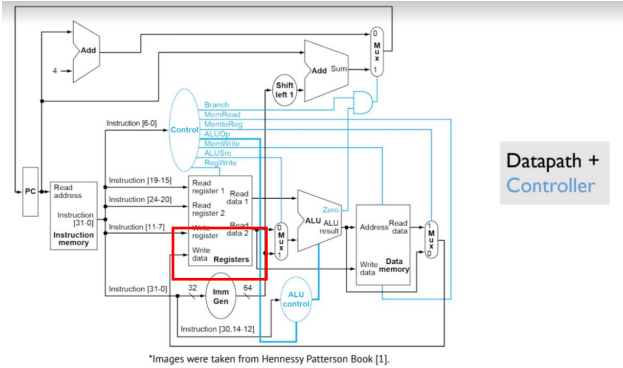
*Images were taken from Hennessy Patterson Book [1].

Write Back

-Write back value to register file

-Store instruction: don’t need to write back!

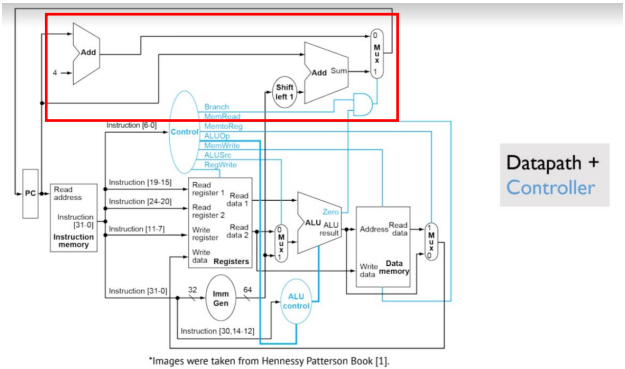
-Grab value from either data memory or ALU, place into register file



*Images were taken from Hennessy Patterson Book [1].

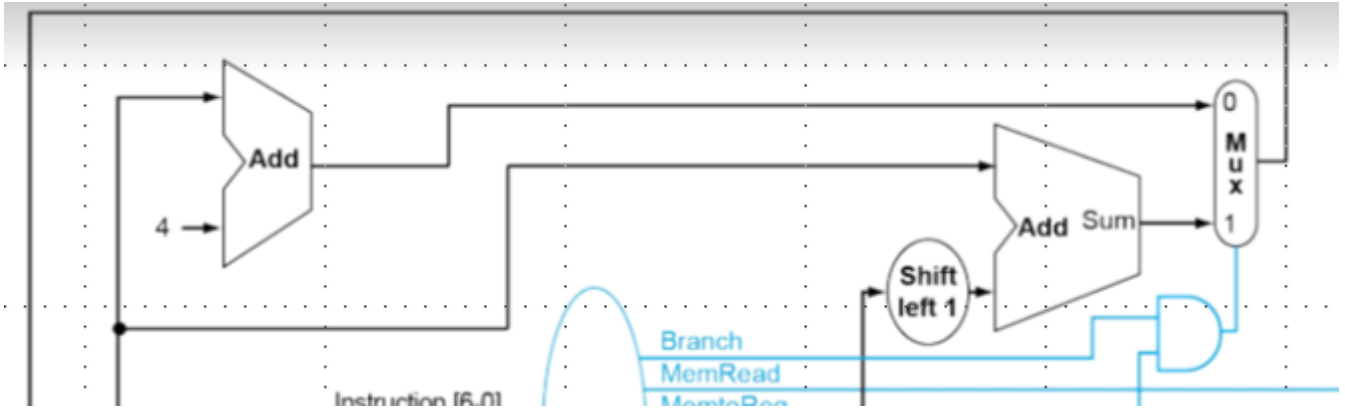
PC Generation

-Compute next instruction to process
-Normal operation: add 4 to existing PC
-Branches/jumps: add offset to PC
-Shift left by 1? Remember, zeroth bit missing in instruction. This also ensures even addresses
-Use mux to select correct PC value



Datapath + Controller

*Images were taken from Hennessy Patterson Book [1].



Controller

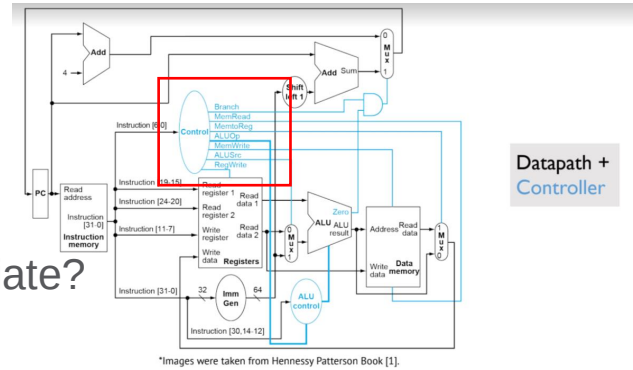
Controller

-Oversight over datapath to do correct operation

-Questions answered:

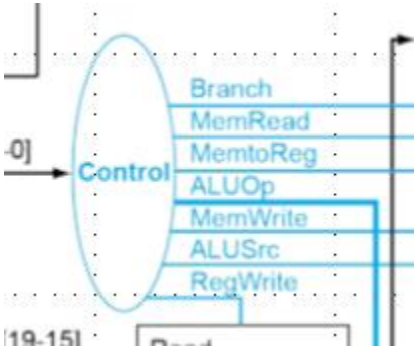
- -Read/write from memory?
- -Place value from memory into register?
- -2nd ALU operand being from register, or immediate?
- -Write into register file?
- -The exact ALU operation?

-Done via just the opcode

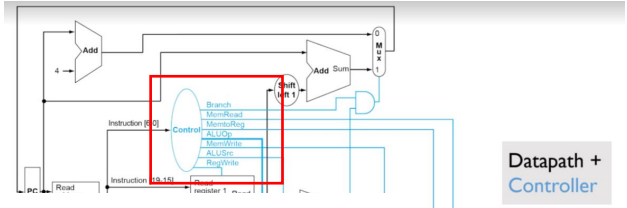


Datapath + Controller

*Images were taken from Hennessy Patterson Book [1].



Controller



Datapath +
Controller

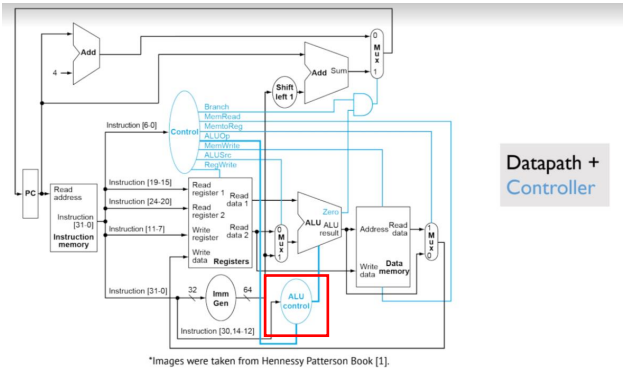
Instruction	Opcode	RegWrite	AluSrc	Branch	MemRe	MemWr	MemtoReg	ALUOp
R-type	0110011	1	0	0	0	0	0	-
I-type	0010011	1	1	0	0	0	0	-
lw	0000011	1	1	0	1	0	1	-
sw	0100011	0	1	0	0	1	0	-
beq	1100011	0	0	1	0	0	0	-

ALU Control

-Given an opcode, return a 2 bit value (what kind of instruction it is (ld, sd, beq, r type)
-Based on what instruction opcode it is (ALUOp) and the func codes, do the correct ALU operation (add, and, etc

Instruction opcode	ALUOp
ld	00
sd	00
beq	01
R-type	10
R-type	10
R-type	10
R-type	10

Funct7 field	Funct3 field	Desired ALU action	ALU control input
XXXXXXX	XXX	add	0010
XXXXXXX	XXX	add	0010
XXXXXXX	XXX	subtract	0110
0000000	000	add	0010
0100000	000	subtract	0110
0000000	111	AND	0000
0000000	110	OR	0001



Datapath + Controller

*Images were taken from Hennessy Patterson Book [1].

Note: there was some confusion over the naming of the output of the ALU control. We will call it “Operation”

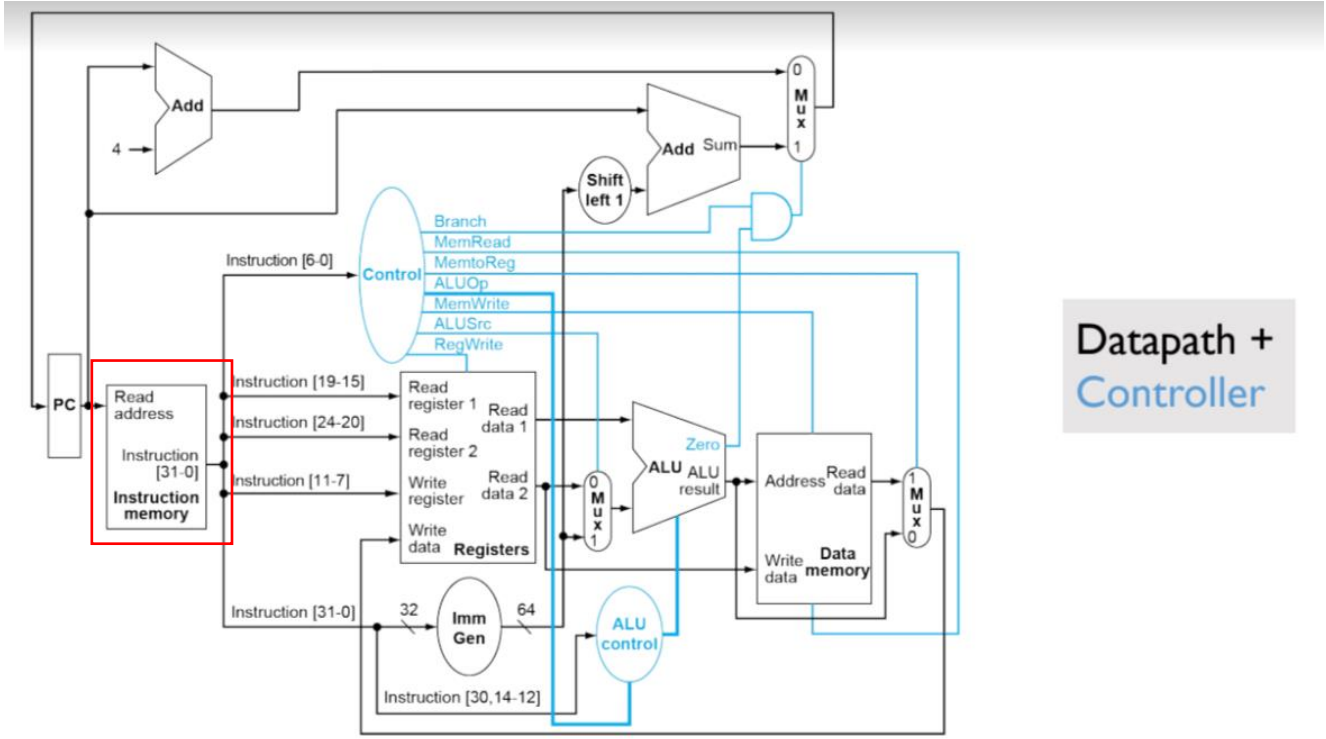
Ex) Add Instruction?

-How does an “add” instruction propagate through the datapath/controller?

add rd, rs1, rs2

Ex) Add Instruction?

-Grab instruction from instruction memory

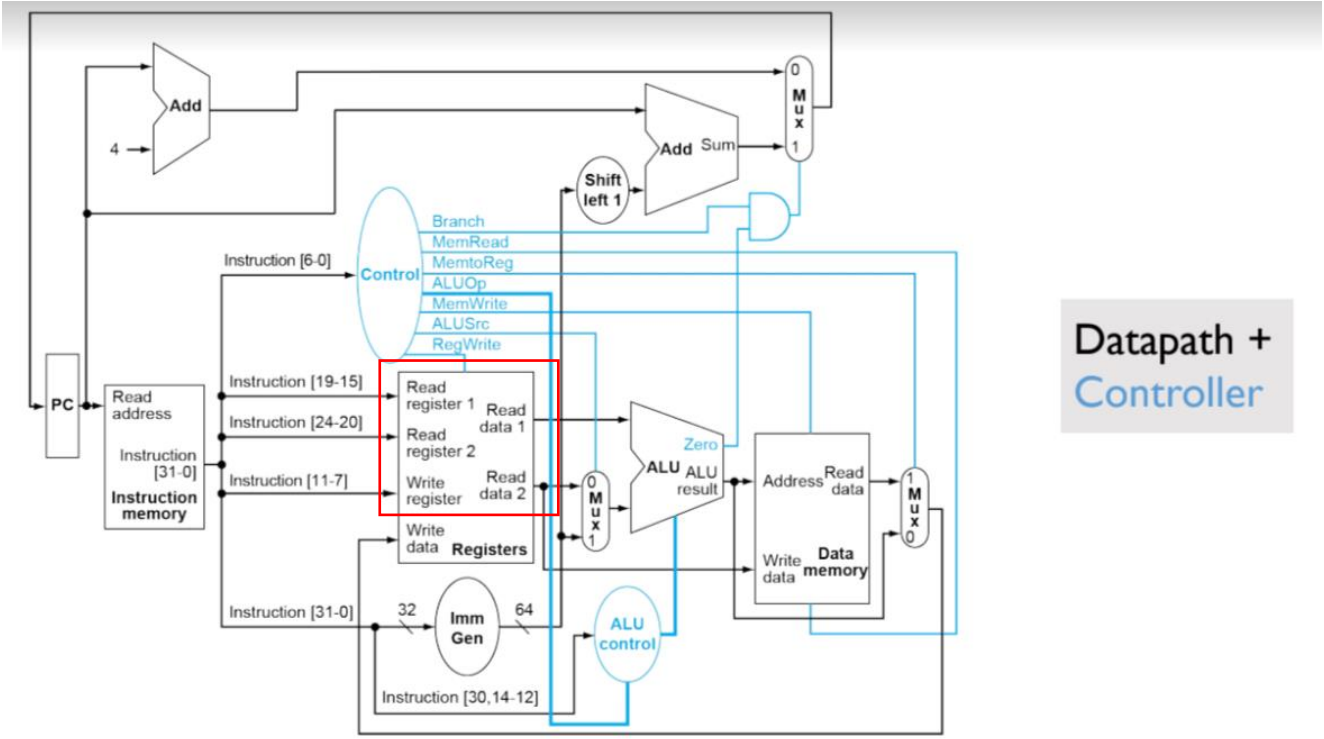


Datapath + Controller

*Images were taken from Hennessy Patterson Book [1].

Ex) Add Instruction?

-Access register values for operands, using register addresses found in instruction (rs1, rs2)

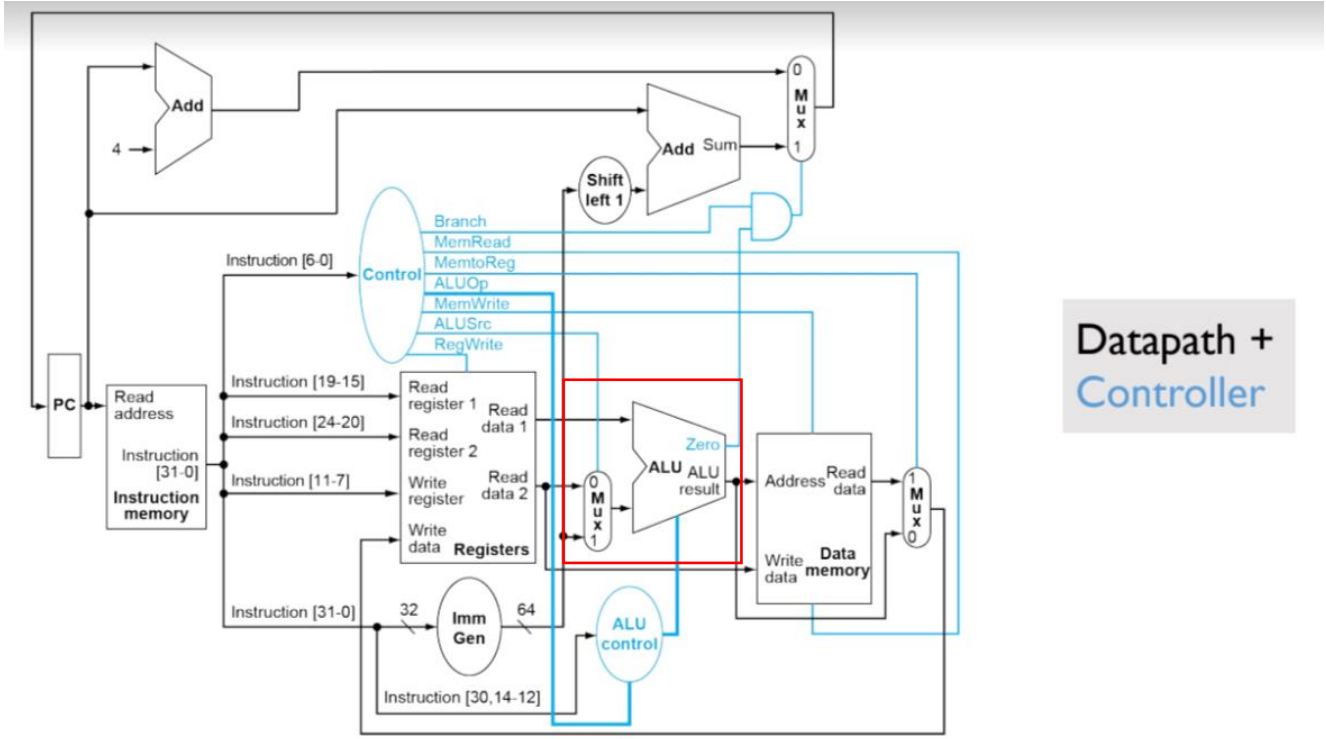


Datapath + Controller

*Images were taken from Hennessy Patterson Book [1].

Ex) Add Instruction?

-Select mux input 0 (read data 2)
-Add values in rs1 and rs2 together in ALU

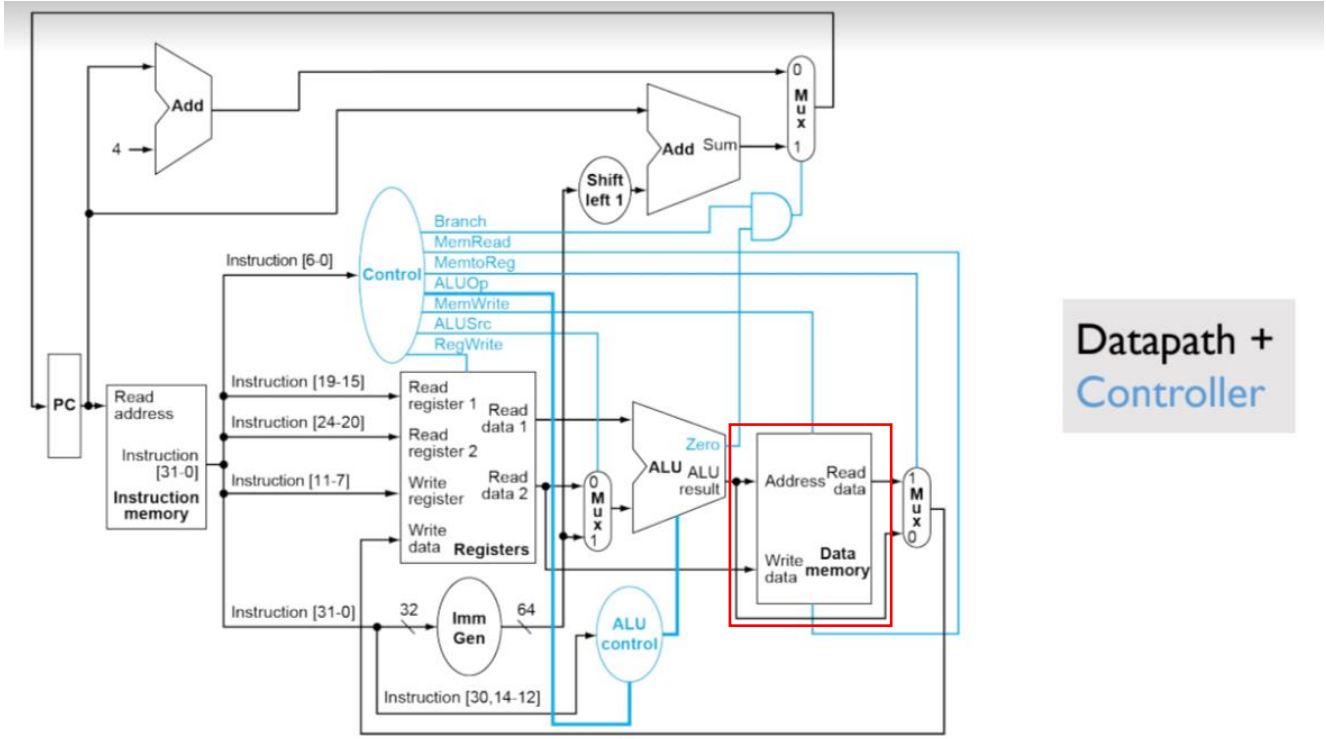


Datapath + Controller

*Images were taken from Hennessy Patterson Book [1].

Ex) Add Instruction?

-Mem?
Nothing!

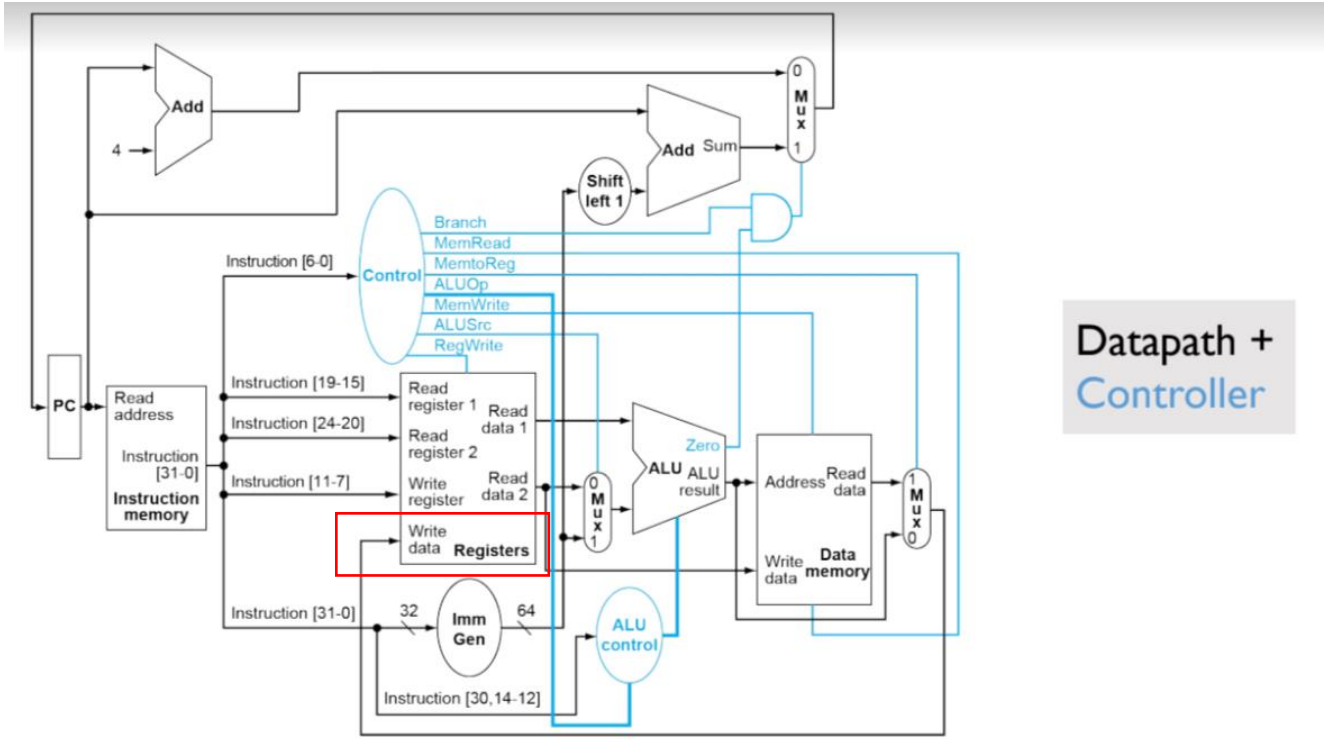


Datapath +
Controller

*Images were taken from Hennessy Patterson Book [1].

Ex) Add Instruction?

-Write back to memory the sum, using address rd

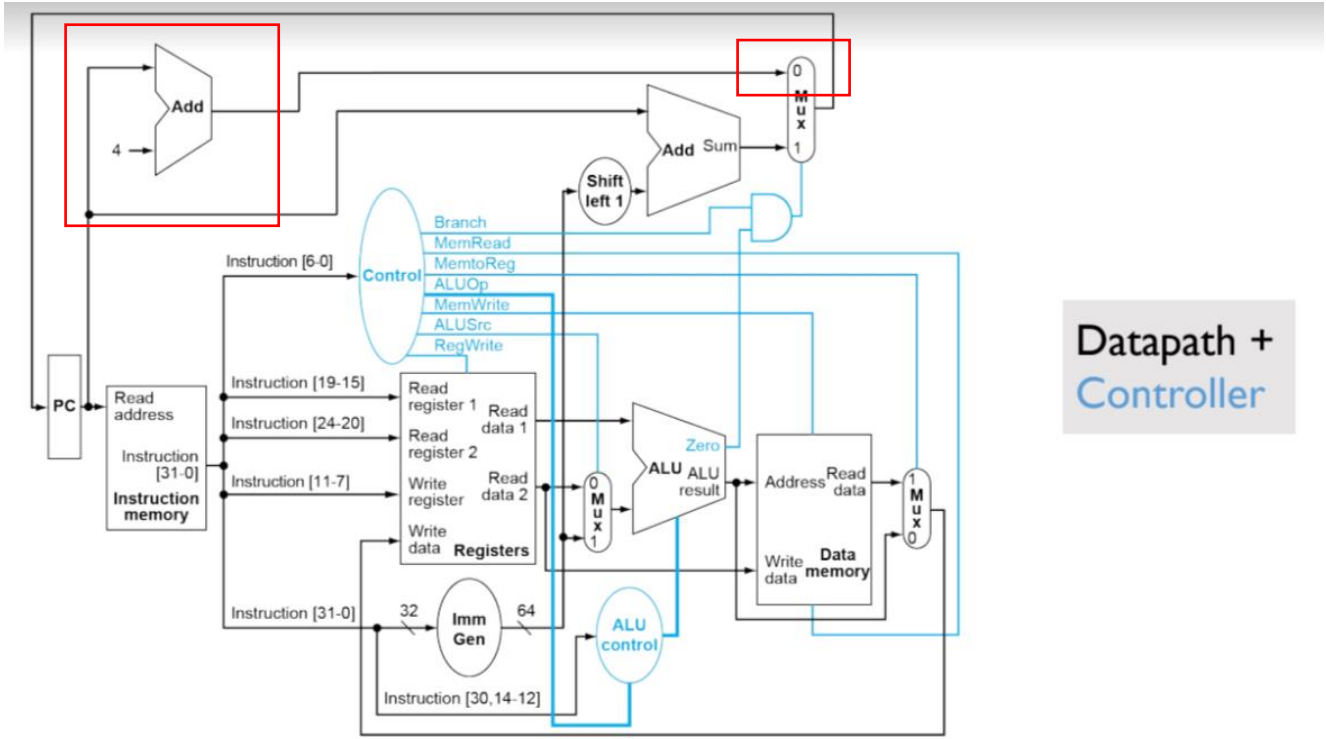


Datapath + Controller

*Images were taken from Hennessy Patterson Book [1].

Ex) Add Instruction?

-PC gets updated!
-No branch



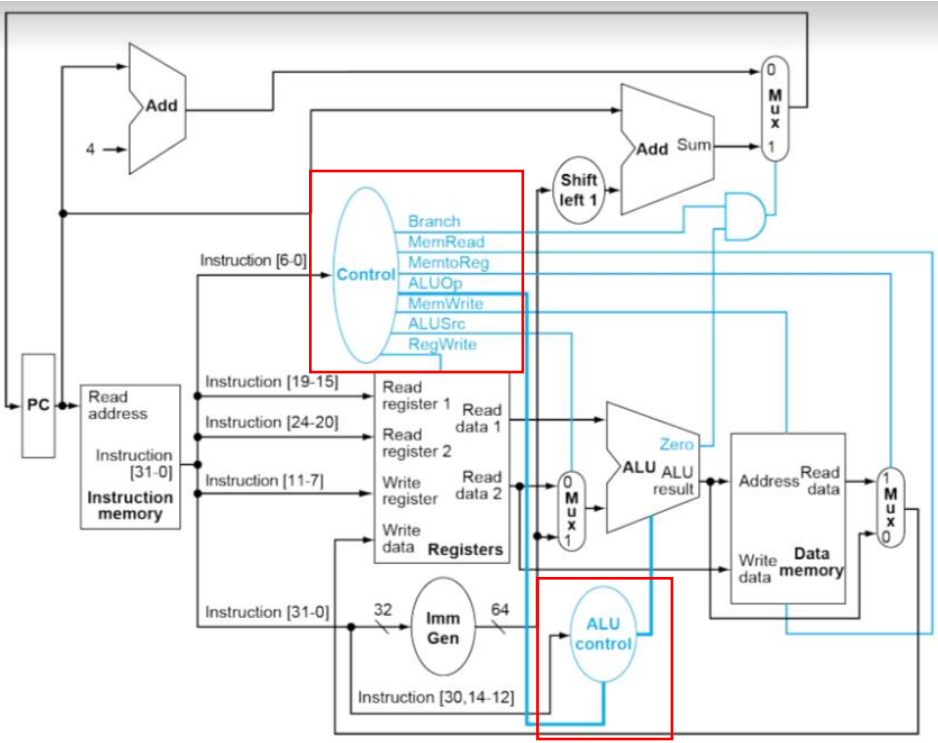
Datapath + Controller

*Images were taken from Hennessy Patterson Book [1].

Ex) Add Instruction?

-Controller:
-Branch: 0
-MemRead: 0
-MemtoReg: 0
-ALUOp: 10
-MemWrite: 0
-ALUSrc: 0
-RegWrite: 1

-ALU control: 0010



Datapath + Controller

*Images were taken from Hennessy Patterson Book [1].

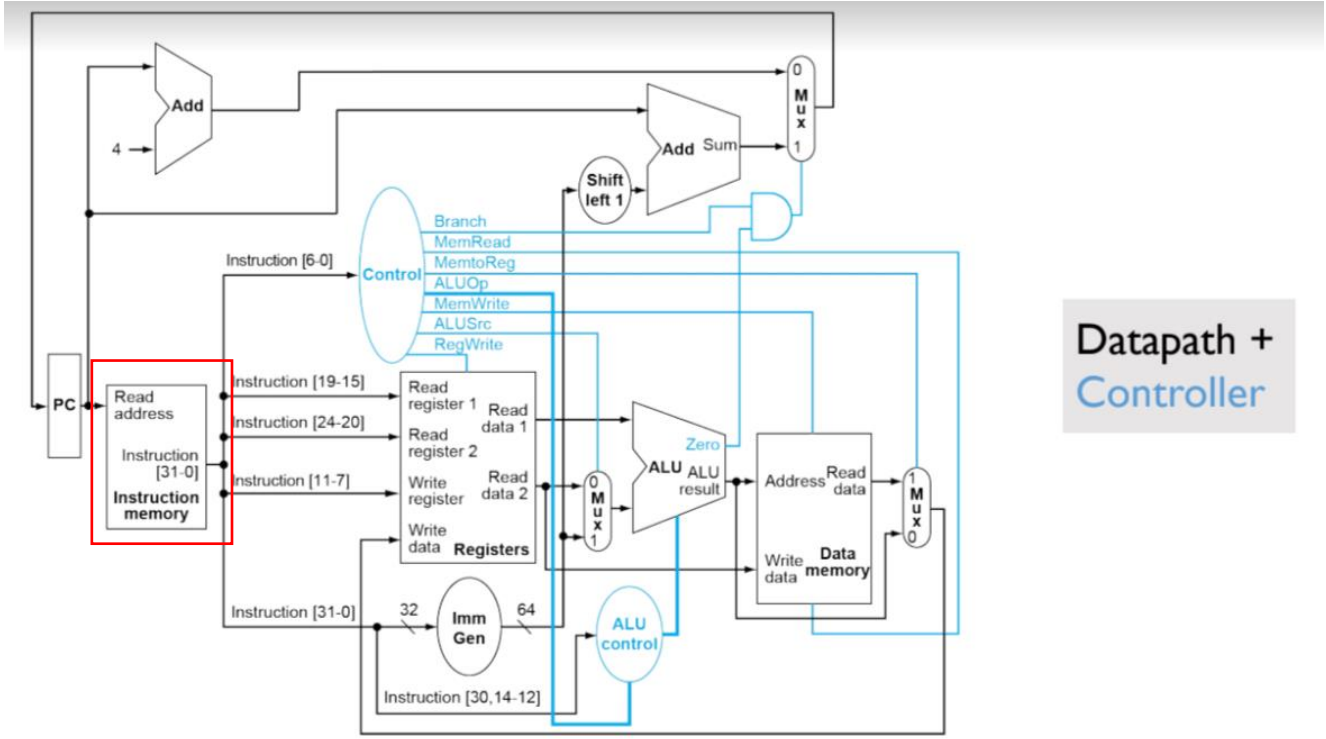
Ex) Store Instruction?

-How does a “store” instruction propagate through the datapath/controller?

sw rs2, imm(rs1)

Ex) Store Instruction?

-Grab instruction from instruction memory

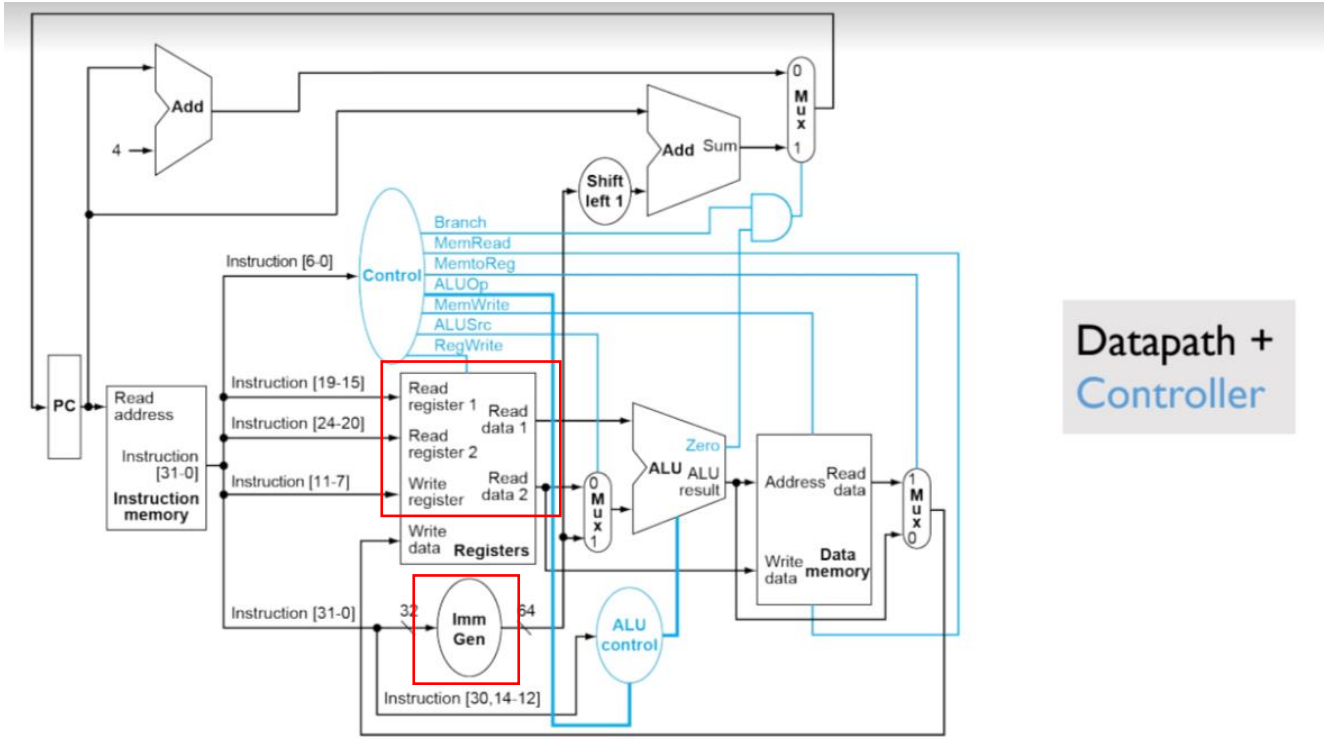


Datapath + Controller

*Images were taken from Hennessy Patterson Book [1].

Ex) Store Instruction?

-Access register values for operands, using register addresses found in instruction (rs1, rs2)
-Generate immediate

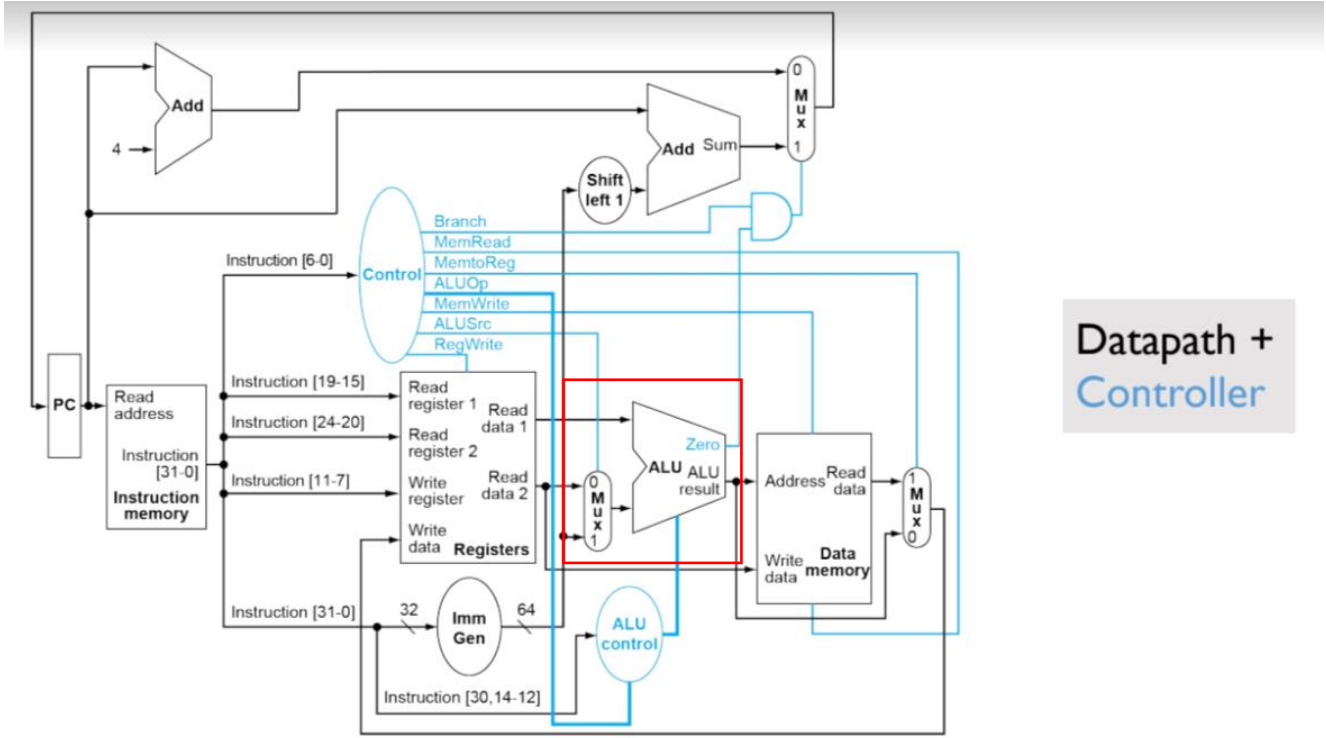


Datapath + Controller

*Images were taken from Hennessy Patterson Book [1].

Ex) Store Instruction?

-Select mux input 1 (immediate)
-Add values in rs1, and immediate together

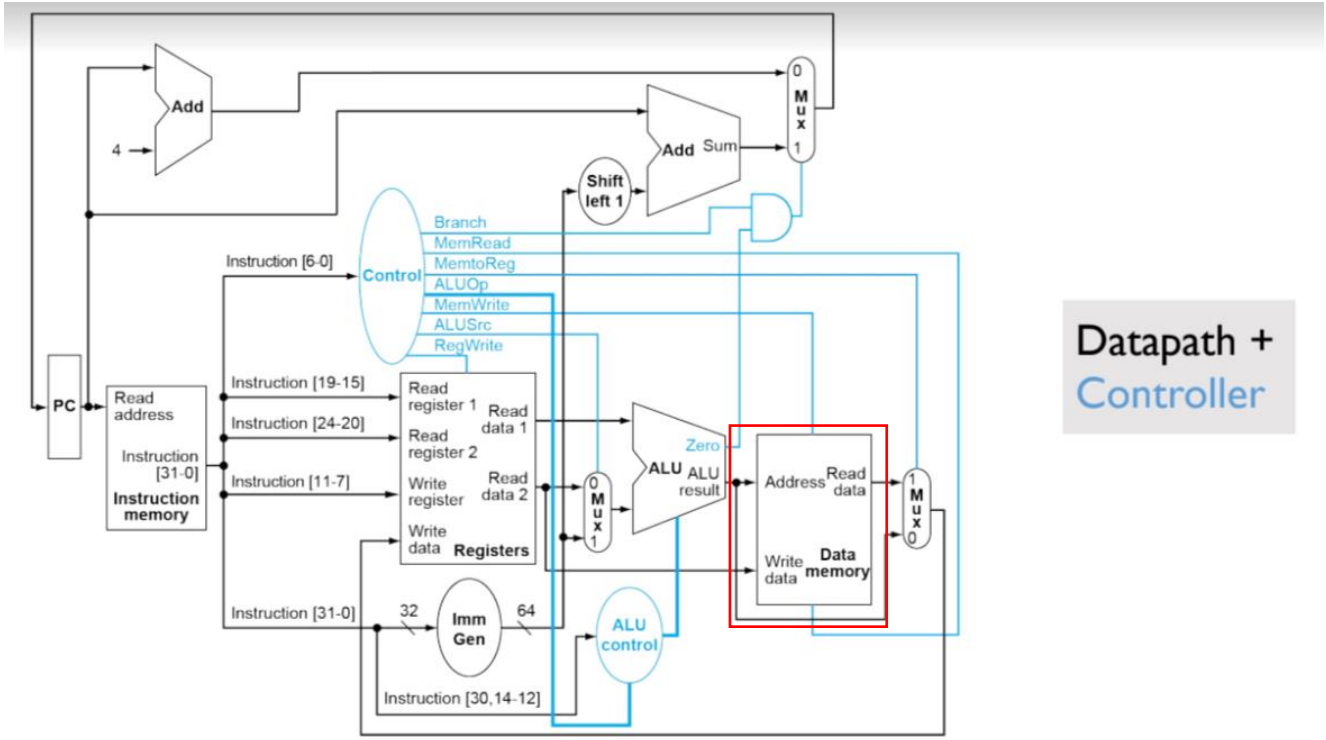


Datapath + Controller

*Images were taken from Hennessy Patterson Book [1].

Ex) Store Instruction?

-Using result of ALU, store value in rs2 into data memory (address of: value in rs1, + imm)

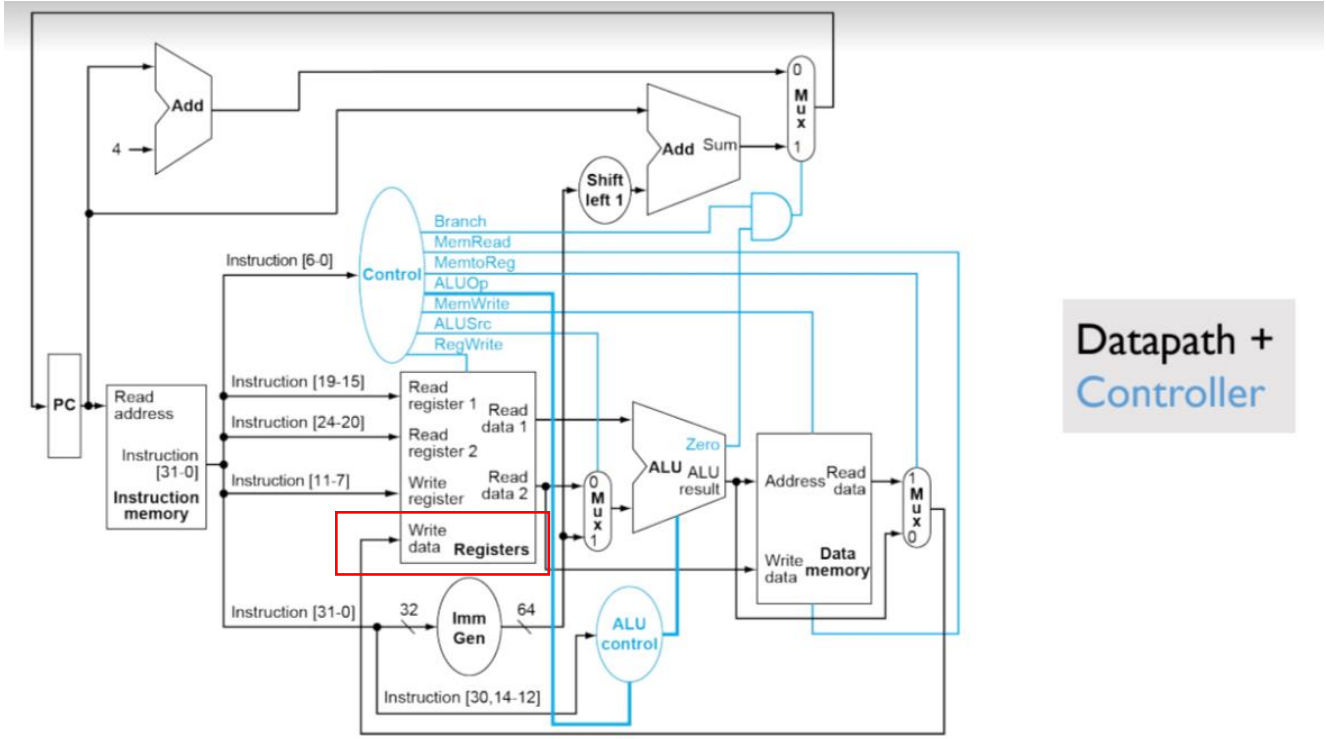


Datapath + Controller

*Images were taken from Hennessy Patterson Book [1].

Ex) Store Instruction?

-Write back?
-Nope!

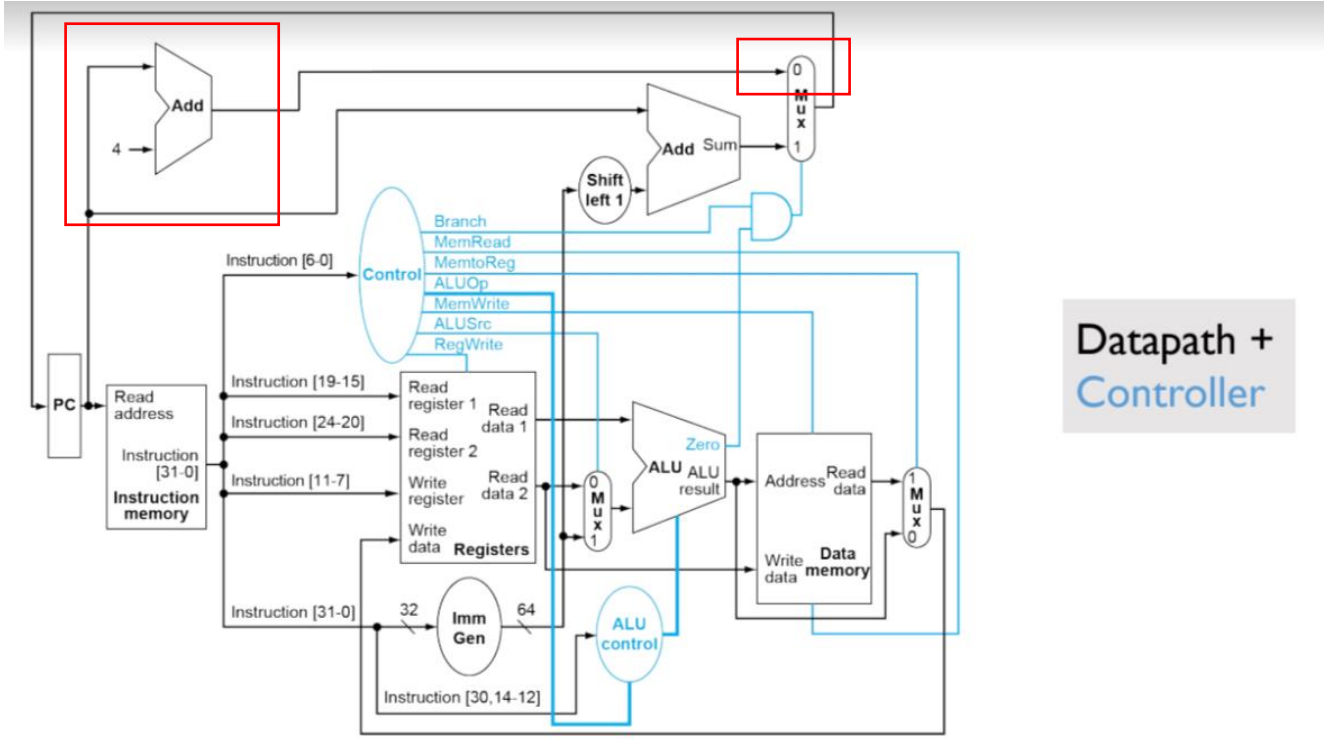


Datapath + Controller

*Images were taken from Hennessy Patterson Book [1].

Ex) Store Instruction?

-PC gets updated!
-No branch



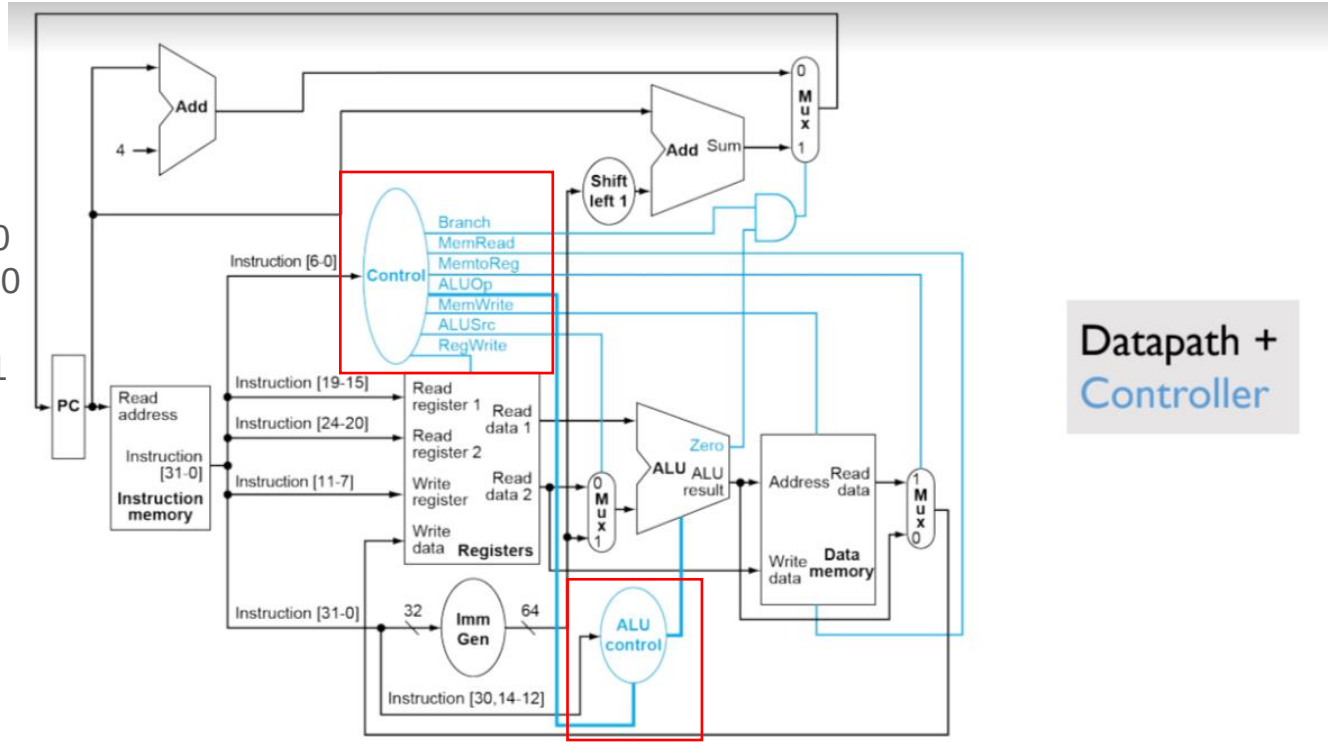
Datapath + Controller

*Images were taken from Hennessy Patterson Book [1].

Ex) Store Instruction?

-Controller:
-Branch: 0
-MemRead: 0
-MemtoReg: 0
-ALUOp: 00
-MemWrite: 1
-ALUSrc: 1
-RegWrite: 0

-ALU control:
0010



Datapath + Controller

*Images were taken from Hennessy Patterson Book [1].

Ex) New Instruction?

-If one wanted to implement a “swap” instruction, where two register values switch locations, and one needs to use existing instruction types (such as r-type, i-type, etc), what would need to be changed with the existing hardware?

Ex) New Instruction?

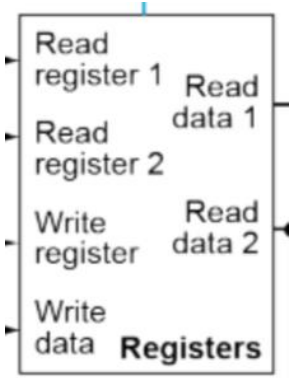
-If one wanted to implement a “swap” instruction, where two register values switch locations, and one needs to use existing instruction types (such as r-type, i-type, etc), what would need to be changed with the existing hardware?

-Ans:

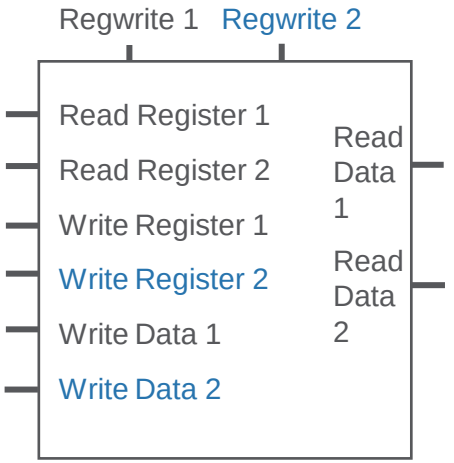
-Additional write address

-Additional write port

-Additional reg write control port



Normal



New