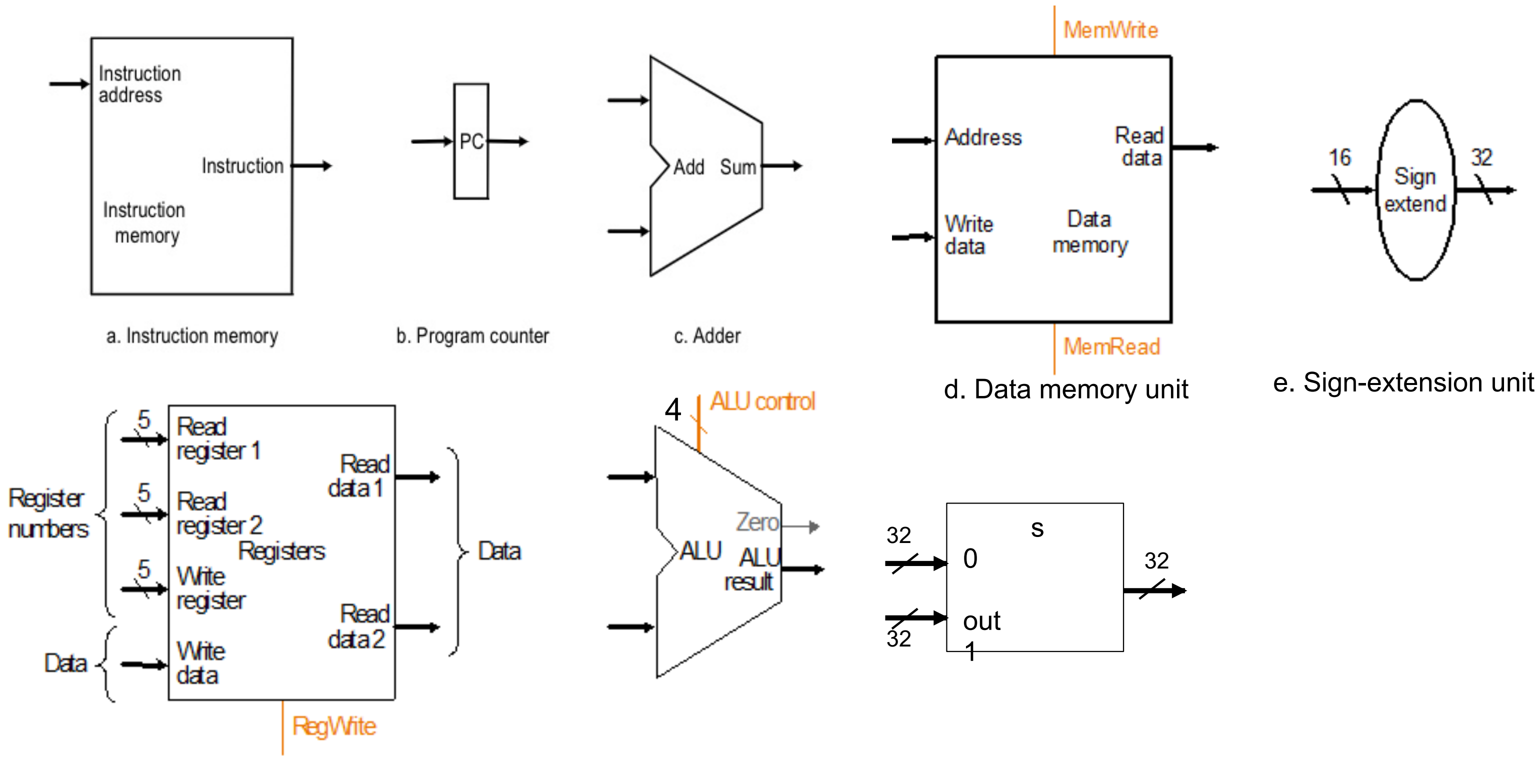
TASK 1: Implementing a datapath to execute Arithmetic (R-type) operations

Building blocks

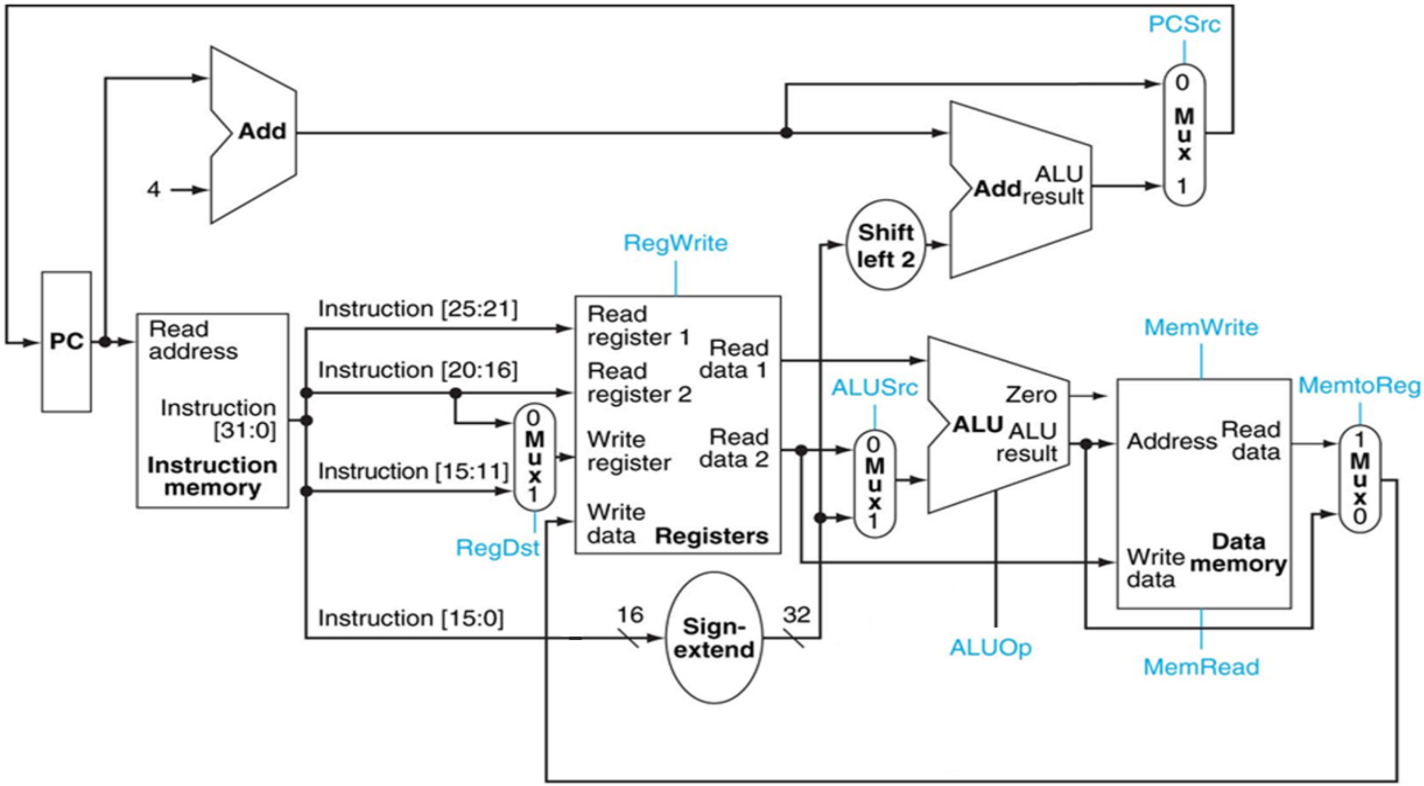


a. Instruction memory
b. Program counter
c. Adder
d. Data memory unit
e. Sign-extension unit
f. Register File
g. ALU
h. 2-to-1 Mux

Below is the Datapath that we will use. We will start with understanding how we can configure the Control signals based on any given instruction

Instruction[31:0]

31:26	25:21	20:16	15:11	10:6	5:0
op	rs	rt	rd	shamt	funct

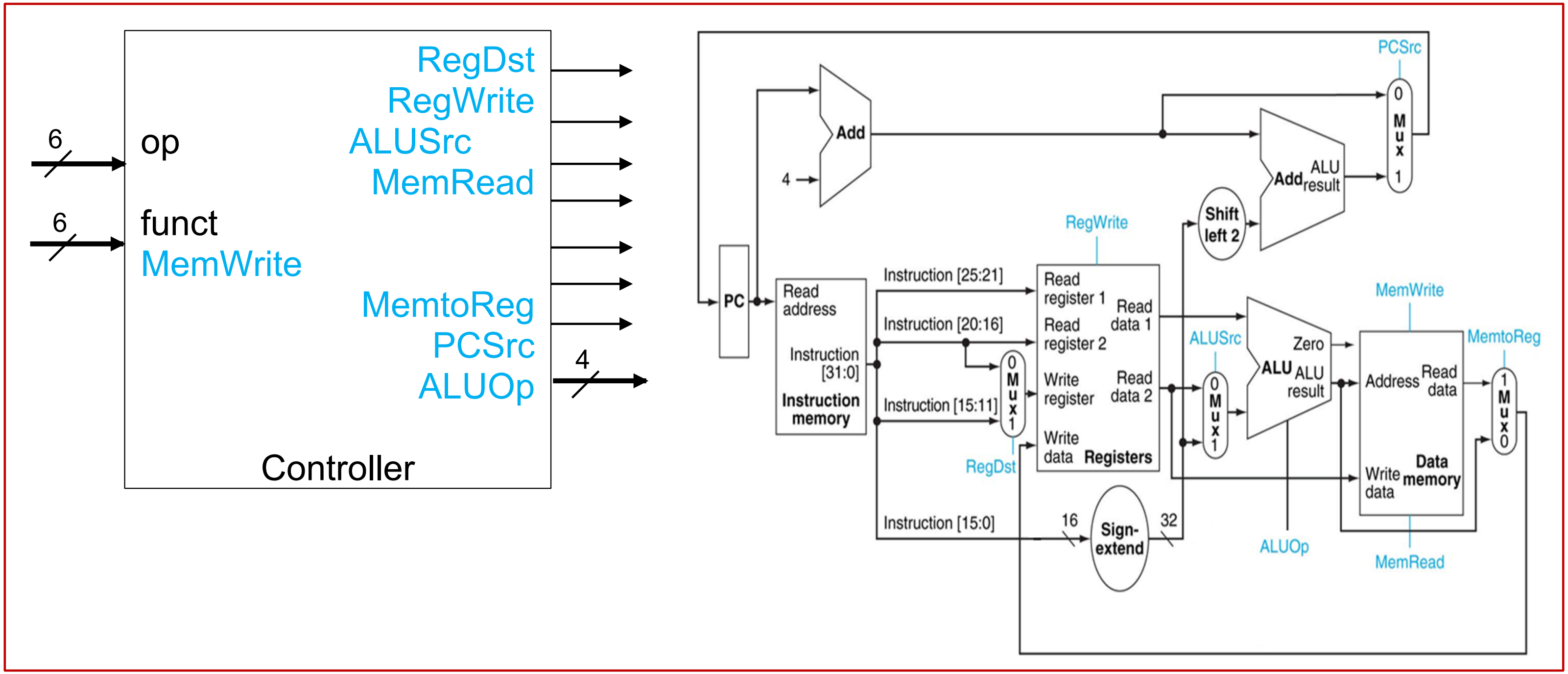


Note: The control signals need to be properly set in order to correctly execute any given instruction. We will change state of the datapath through a controller (not shown on this slide)

Single-cycle processor

Instruction format

31:26	25:21	20:16	15:11	10:6	5:0
op	rs	rt	rd	shamt	funct



Instruction Representation (Addition example)

31:26	25:21	20:16	15:11	10:6	5:0
op	rs	rt	rd	shamt	funct

- A = X + Y
  - X and Y are sources, A is the destination
- In hardware, values of X and Y will be stored in a pair of registers
  - just like a variable in "C programming"
- We need to have a way of telling the datapath that
  - it needs to do "addition" and
  - sources X and Y are in stored a specific register pair and the result will be stored in a specific register.
- Instruction is formed as a 32 bit package
  - √ rs will indicate which register X is stored,
  - √ rt will indicate which register Y is stored,
  - √ rd will indicate where the result of the summation of rs and rt register contents will be stored.
- rs, rt, rd: indicate the register id, there are 32 registers in the register file, therefore 5 bits needed
  - √ Sometimes we will need to shift the contents of a register, since each register is 32bits wide we can do at most 32 bit shift. The "shamt" field is designated for indicating this shift amount.
  - √ "op" and "func" fields (each 6bits) together will be used to determine the nature of operation

Example Operation: Addition

	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	Total of 32 bits
Instruc.	31:26	25:21	20:16	15:11	10:6	5:0	
	op	**rs**	**rt**	**rd**	**shmt**	**funct**	**Expression**
add	000000	used	used	used	00000	100000	Reg[rd] = Reg[rs]+Reg[rt]

- Assume that you have a register file named "Reg" with 32 entries each 32 bits long and you are given the following 32-bits as the instruction:

  000000 10000 01001 00111 00000 100000
- Based on our specification, if the "**op**" == **000000** and "**func**" == **100000** then this will be designated as an **addition** operation.
- It is reading two registers from the register file "Reg"
  - **rs** = **16 (10000)**, **rt=9 (01001)**
    - Read from register file as: Reg[16] and Reg[9]
  - The controller will change the state of the datapath and configure it to execute the addition operation over the values read form the register file
  - The result of the addition will be stored in register **7 (rd).**
    - Reg[7] = Reg[16] + Reg[9]
  - addition operation does not use the "shmt" field. Later we will see instructions that utilize this field in the instruction.

Configuring the ALU to execute a specific arithmetic operation

ALU Table		
Op	ALUOp	Description
ADDITION	0000	ALUResult = A + B
SUBRACTION	0001	ALUResult = A - B
MULTIPLICATION	0010	ALUResult = A * B
AND	0011	ALUResult = A and B
OR	0100	ALUResult = A or B
SET LESS THAN	0101	ALUResult =(A < B)? 1:0
SET EQUAL	0110	ALUResult =(A==B) ? 1:0
SET NOT EQUAL	0111	ALUResult =(A!=B) ? 1:0
LEFT SHIFT	1000	ALUResult = A << B
RIGHT SHIFT	1001	ALUResult = A >> B
ROTATE RIGHT	1010	ALUResult = A ROTR B
COUNT ONES	1011	ALUResult = A CLO
COUNT ZEROS	1100	ALUResult = A CLZ

NOTES:-

MULTIPLICATION : 32-bit signed multiplication results with 64-bit output.
ALUResult will be set to lower 32 bits of the product value.

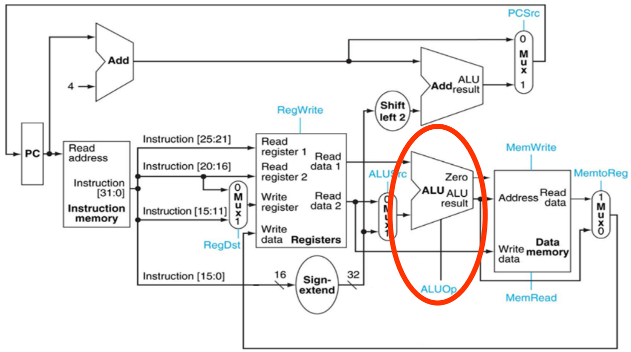
SET LESS THAN : ALUResult is '32'h000000001' if A < B.

LEFT SHIFT: The contents of the 32-bit "A" input are shifted left, inserting zeros into the emptied bits by the amount specified in B.

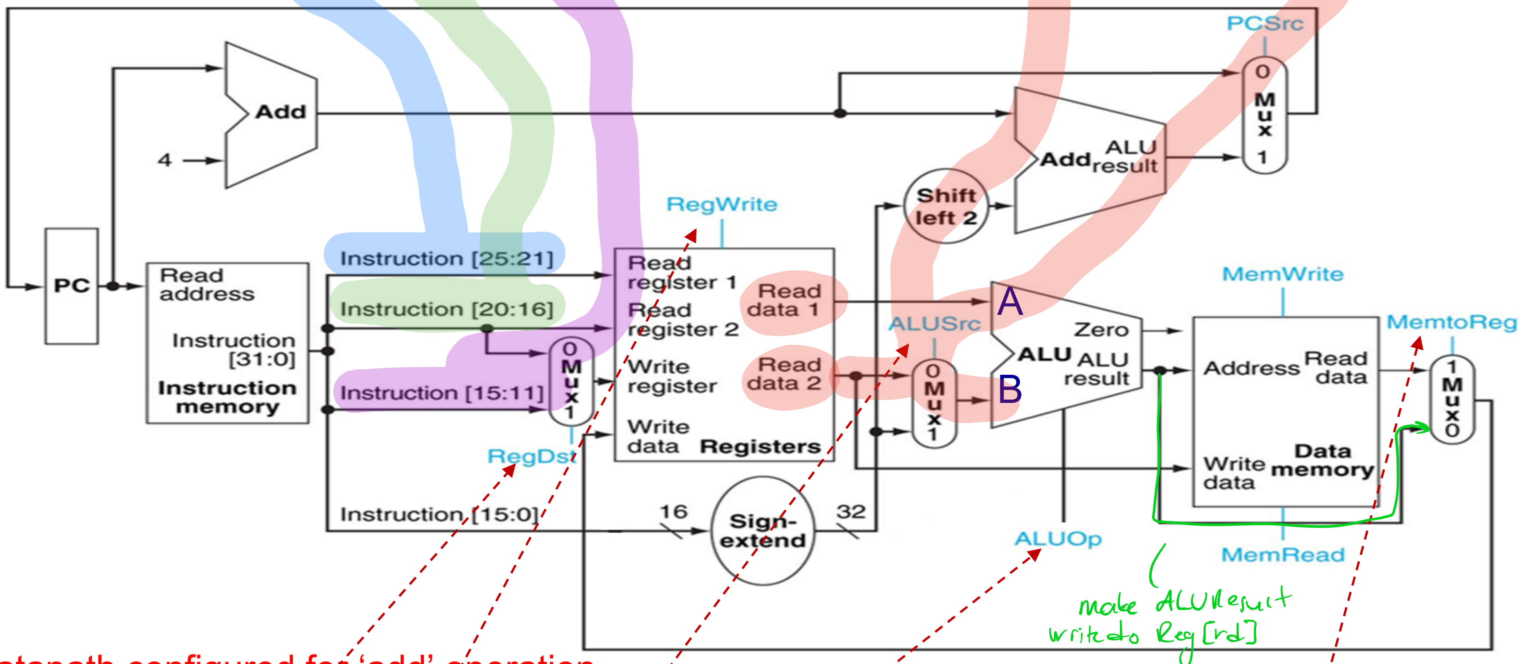
ROTR: logical right-rotate of a word by a fixed number of bits.
The contents of the 32-bit "A" input are rotated right.
The bit-rotate amount is specified by "B".

CLO: Count the number of leading ones in a word.
Bits 31..0 of the input "A" are scanned from most significant to least significant bit.

CLZ: Count the number of leading zeros in a word.
Bits 31..0 of the input "A" are scanned from most significant to least significant bit.



	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	Total of 32 bits
Instruc.	31:26	25:21	20:16	15:11	10:6	5:0	
	op	**rs**	**rt**	**rd**	**shmt**	**funct**	**Expression**
add	000000	used	used	used	00000	100000	Reg[rd] = Reg[rs]+Reg[rt]



Datapath configured for 'add' operation

Type	RegDst	RegWrite	ALUSrc	ALUOp	MemRead	MemWrite	MemtoReg	PCSrc
add	1	1	0	0000 (ALU Table See slide 7)	0	0	0	0

not writing to data memory

Instruction List for Task-1 (R-Type of Operations)

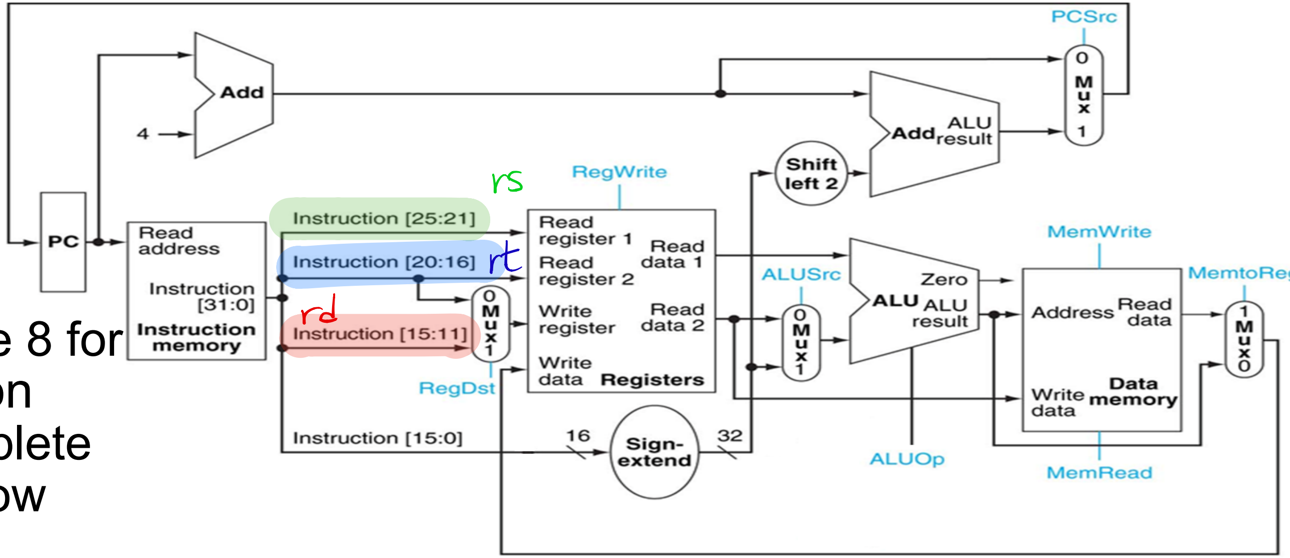
	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	Total of 32 bits
Inst.	op(code)	rs	rt	rd	shmt	funct	Expression
add	000000	used	used	used	x	100000	Reg[rd] = Reg[rs]+Reg[rt]
sub	000000	used	used	used	00000	100010	Reg[rd] = Reg[rs]-Reg[rt]
and	000000	used	used	used	x	100100	Reg[rd] = Reg[rs] AND Reg[rt] (bitwise and)
or	000000	used	used	used	x	100101	Reg[rd] = Reg[rs] OR Reg[rt] (bitwise or)
slt	000000	used	used	used	00000	101010	if (Reg[rs] < Reg[rt]) Reg[rd] = 1 else Reg[rd]=0
sll	000000	x	used	used	used	000000	Reg[rd] = Reg[rt] < < shamt (left shift)
srl	000000	x	used	used	used	000010	Reg[rd] = Reg[rt] >> shamt (logical right shift)
clo	011100	used	not used	used	x	100001	Reg[rd] = count leading ones in Reg[rs]
clz	011100	used	not used	used	x	100000	Reg[rd] = count leading zeros in Reg[rs]
mul	011100	used	used	used	x	000010	Reg[rd] = Reg[rs] X Reg[rt]
rotrv	000000	used	used	used	x	000110	Reg[rd] = Reg[rt] right rotated by Reg[rs]

mul: General purpose register (GPR) rs is multiplied by the 32-bit value in GPR rt, treating both operands as signed values, to produce a 64-bit result. The least significant 32 bits of the product are written to GPR rd

sll: The contents of the 32-bit word of GPR rt are shifted left, inserting zeros into the emptied bits; the word result is placed in GPR rd. The bit-shift amount is specified by shmt

srl: The contents of the 32-bit word of GPR rt are shifted right, inserting zeros into the emptied bits; the word result is placed in GPR rd. The bit-shift amount is specified by shmt.

X: don't care



Exercise-1

Refer to Slide 8 for the instruction specs., Complete the table below

Type	Opcode	funct	ALUSrc	RegDst	RegWrite	ALUOp	MemRead	MemWrite	MemtoReg	PCSrc
add	000000	100000	0	1	1	0000	0	0	0	0
sub	000000	100010	0	1	1	0001	0	0	0	0
and	000000	100100	0	1	1	0011	0	0	0	0
or	000000	100101	0	1	1	0100	0	0	0	0
slt	000000	101010	0	1	1	0101	0	0	0	0
sll	000000	000000	See slide 14 and 15							
srl	000000	000010	See slide 14 and 15							
clo	011100	100001	0	1	1	1011	0	0	0	0
clz	011100	100000	0	1	1	1100	0	0	0	0
mul	011100	000010	0	1	1	0010	0	0	0	0

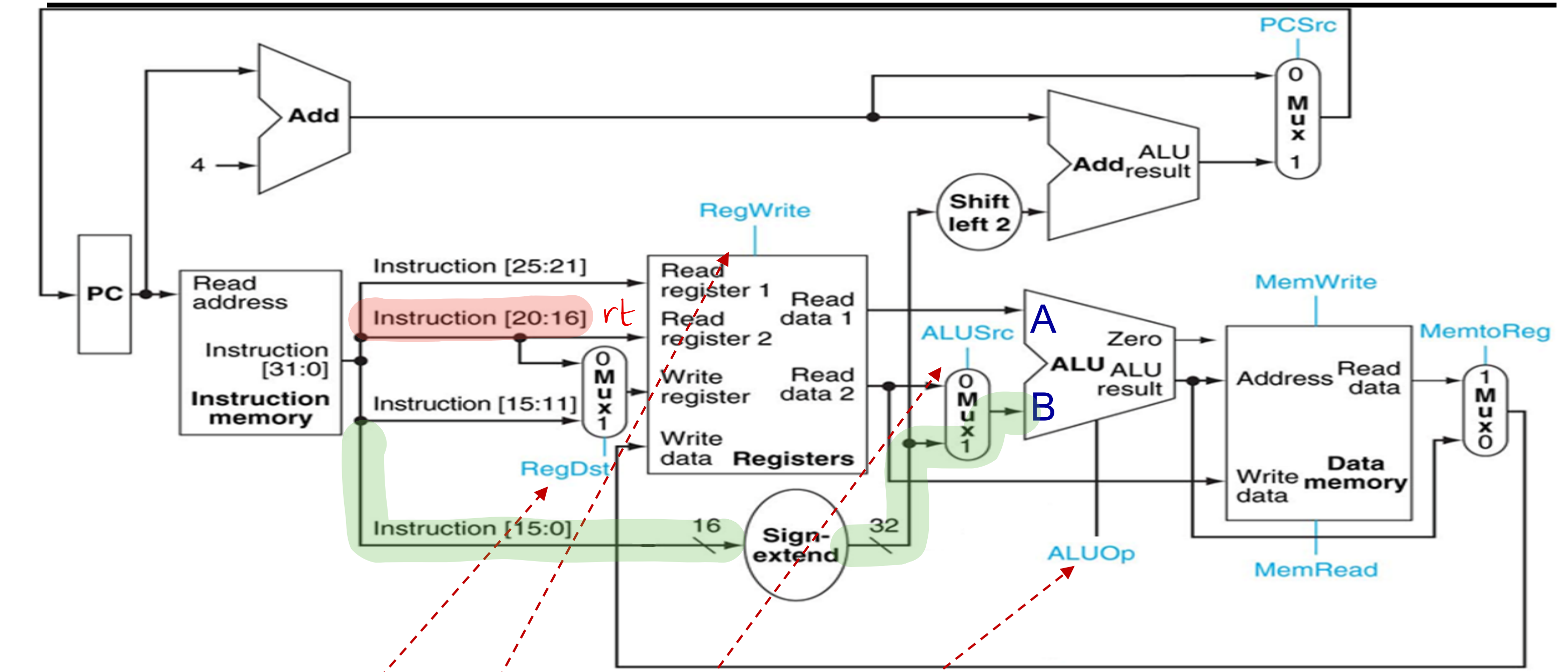
Arithmetic with Immediate Values

31:26	25:21	20:16	15:0
op	rs	rt	Immediate/address

	Instruction Fields				Expression
	6 bits	5 bits	5 bits	16 bits	
Inst.	opcode	rs	rt	offset	
addi	001000	used	used	16 bits immediate value	Reg[rt] = Reg[rs] + immediate
ori	001101	used	used	16 bits immediate value	Reg[rt] = Reg[rs] OR immediate

- Sometimes values are available at the compilation time, such as the constant values you declare or expressions with constant values in a C program: A=A+2. You don't need to store the constant value in a register. The instruction itself can provide this value to the Datapath. We refer to this class of instructions as "Immediate" type, where value of A will be retrieved from the register file using the "rs" field and the value "2" will be stored in the least significant 16 bits of the instruction. In this case, all we need to do is read the "A" value from the register file and configure ALU to do addition with its second input set to 2.
- You will be supporting "addi" and "ori" operations on your Datapath. Controller will determine that this is a " addi" or "ori" instruction based on its unique opcode (op) value of 8 or 13 respectively.
- We will read the contents of "rs" register (Reg[rs])
- Hardcoded immediate value is a 16-bit number. Our Datapath is a 32-bit architecture. ALU receives two 32-bit inputs. Therefore we need to extend the 16-bit immediate value to 32 bits. While doing this we need to take into account the sign of the value. Offset amount can be positive or negative. Therefore we will first sign extend this immediate value and then feed into the ALU.
- We will write the result of the operation (output of ALU) to our destination register rt. Note that in R-type of instructions, destination register is always indicated by the "rd" field. For the immediate field we no longer have the "rd" field and we will use the "rt" field in the instruction instead
- Now let's identify the values of Datapath control signals for executing addi and ori instructions.
- You will see the role for some of the muxes shown in the datapath with this exercise. (next slide)

	6 bits	5 bits	5 bits	16 bits	Expression
Inst.	opcode	rs	rt	offset	
addi	001000	used	used	16 bits immediate value	Reg[rt] = Reg[rs] + immediate

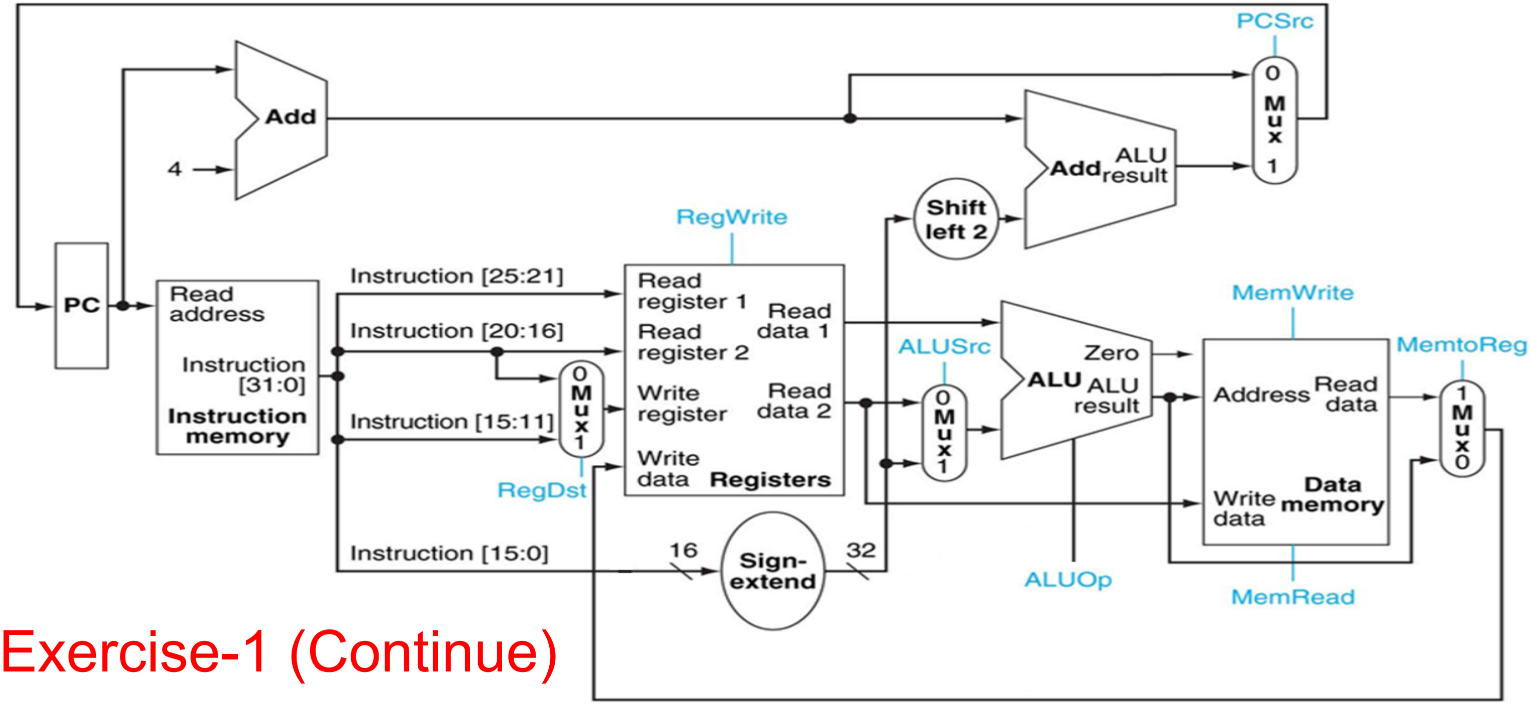


Datapath configured for 'add' operation

Type	RegDst	RegWrite	ALUSrc	ALUOp	MemRead	MemWrite	MemtoReg	PCSrc
addi	0	1	1	0000 (ALU Table See slide 7)	0	0	0	0

0 to write into Reg[rt]

Not writing to data memory



Exercise-1 (Continue)

Type	opcode	ALUSrc	RegDst	RegWrite	ALUOp	MemRead	MemWrite	MemtoReg	PCSrc
addi	001000	1	0	1	0000	0	0	0	0
ori	001101	1	0	1	0100	0	0	0	0

	Instruction Fields				Expression
	6 bits	5 bits	5 bits	16 bits	
Inst.	opcode	rs	rt	offset	
addi	001000	used	used	16 bits immediate value	Reg[rt] = Reg[rs] + immediate
ori	001101	used	used	16 bits immediate value	Reg[rt] = Reg[rs] OR immediate

Datapath completion

31:26	25:21	20:16	15:11	10:6	5:0
op	rs	rt	rd	shamt	funct

Inst.	[31:26]	[25:21]	[20:16]	[15:11]	[10:6]	[5:0]	
	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	Total of 32 bits
	Opcode	rs	rt	rd	shamt	funct	Expression
sll	000000	x	used	used	used	000000	Reg[rd] = Reg[rt] < < shmt (left shift)
srl	000000	x	used	used	used	000010	Reg[rd] = Reg[rt] >> shmt (right shift)

- The given Datapath (Slide 3) is not ready to support the shift left and shift right instructions shown above
- The shamt bits (Instruction[10:6]) should be fed to the B input of ALU. Also the [rt] register content should be fed into the A input of ALU.
- Since the shamt has only 5 bits, zero extension instead of sign extension should be used so that the 5-bit value turns into a 32-bit value before feeding into the ALU.

Hint:

#1) 2x1 muxes should be used to make sure that the input A of ALU and the input B of ALU get the correct input signals when the shifting operation needs to be executed.

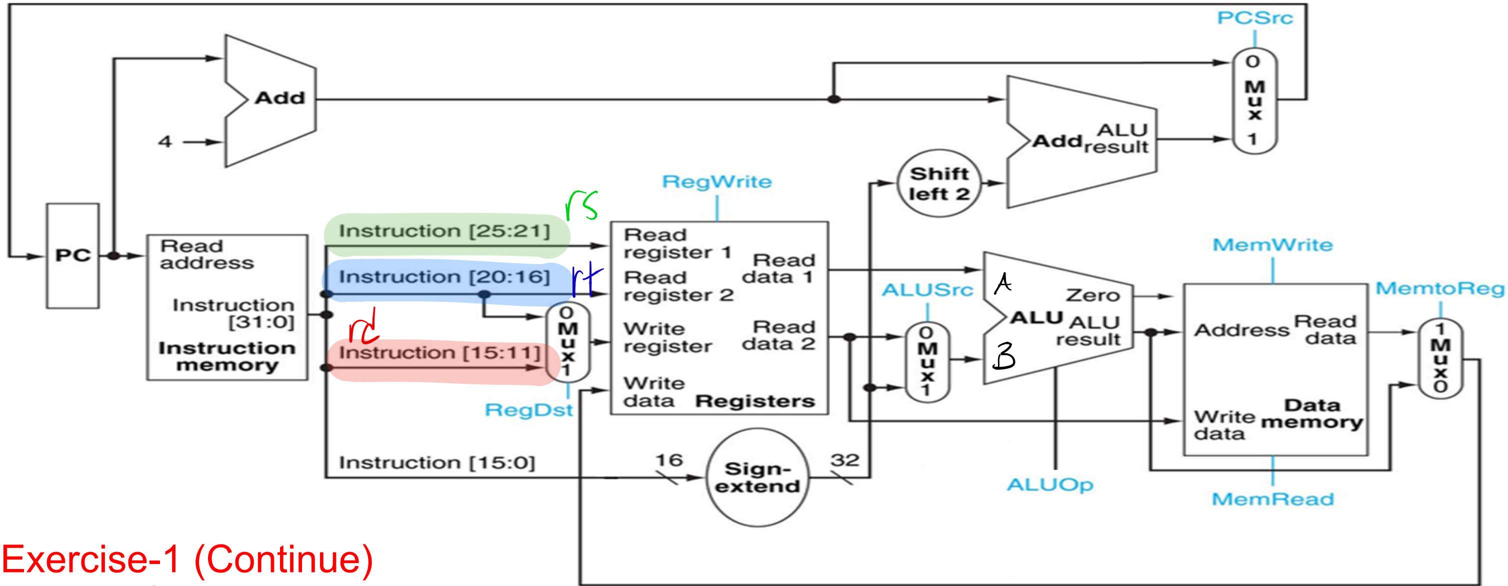
In ALU, the shift operation is A >> B or A << B,

Therefore, modify the Datapath such that

input A of ALU can also come from Reg[rt] (in addition to Reg [rs])

input B of ALU can also come from shamt (Instruction bits [10:6] shown in the table above) (in addition to output of the mux (with ALUSrc as select signal) - as shown in Datapath (Slide 3))

#2) More control signals should be added to the Controller (extra control signals are from the select signals of the muxes that you have to add in Hint #1)



Exercise-1 (Continue)

(1) Modify the Datapath above based on slide 14,

(2) Add the control signals (the select signals used with muxes that you just add) to the table below)

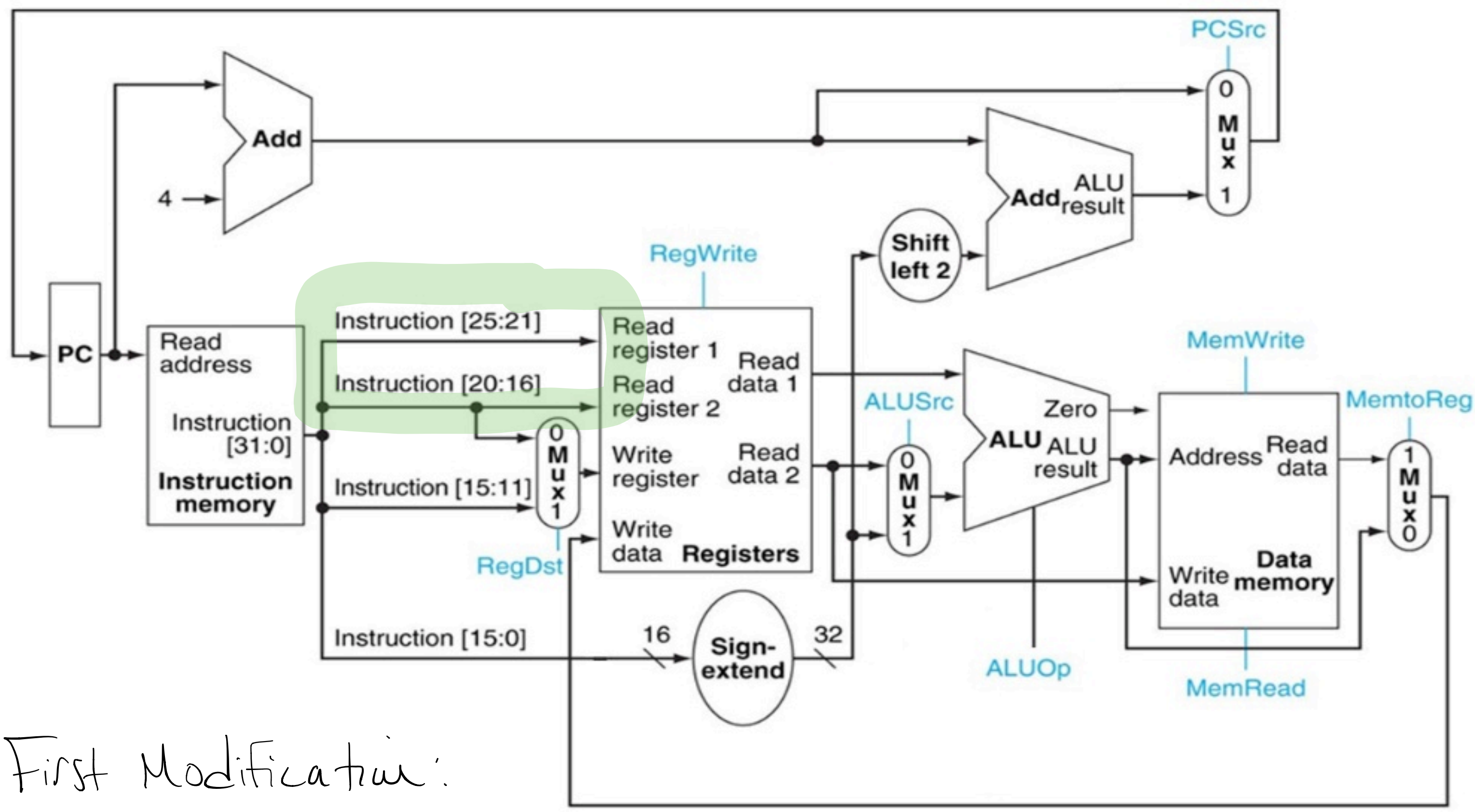
(3) Complete the table below

type	ALUSrc	RegDst	RegWrite	ALUOp	MemRead	MemWrite	MemtoReg	PCSrc	RegA	RegB
sll	X	1	1	1000	0	0	0	0	1	1
srl	X	1	1	1001	0	0	0	0	1	1

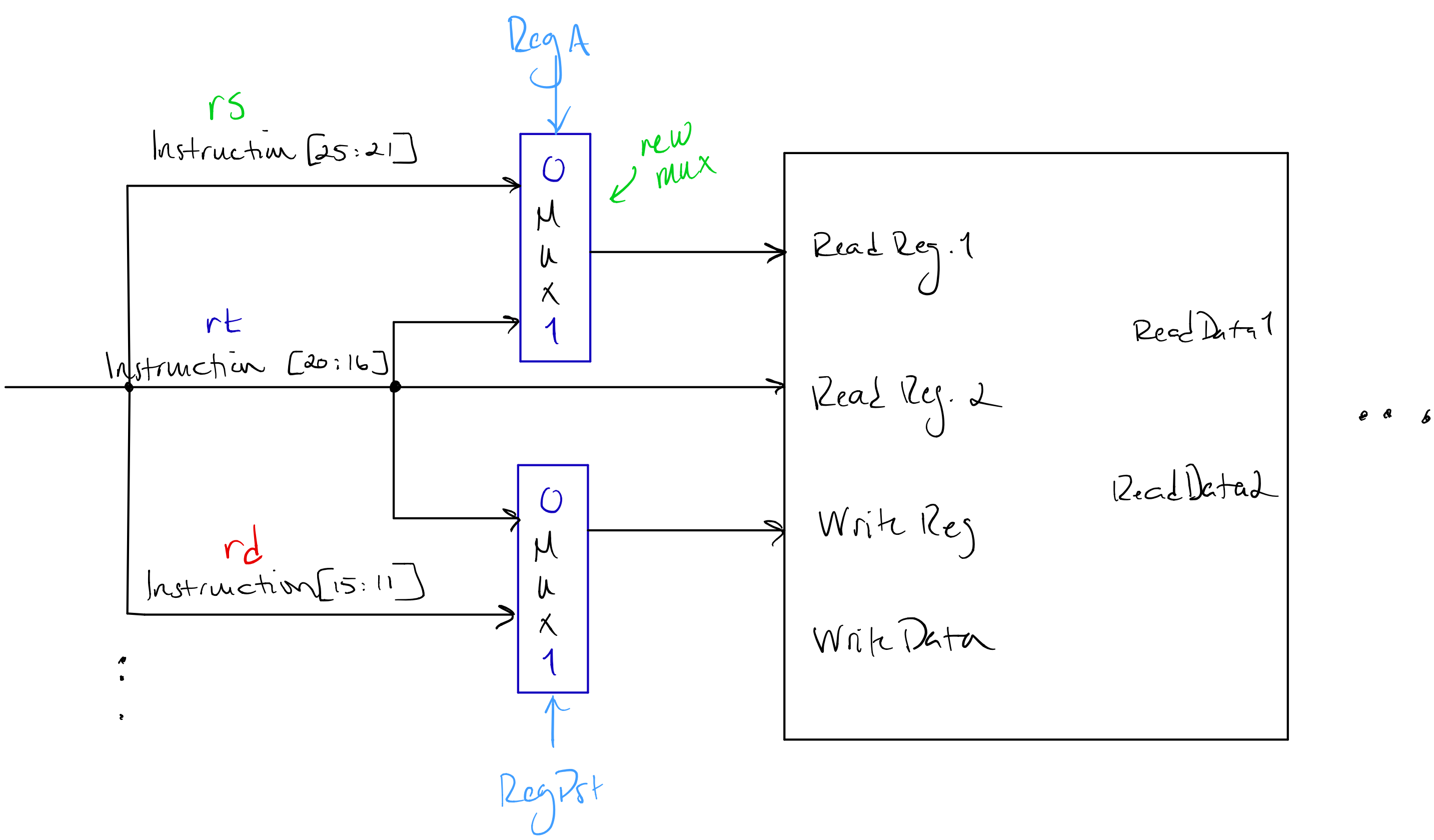
don't cares

Inst.	[31:26]	[25:21]	[20:16]	[15:11]	[10:6]	[5:0]	
	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	Total of 32 bits
	Opcode	rs	rt	rd	shamt	funct	Expression
sll	000000	x	used	used	used	000000	Reg[rd] = Reg[rt] < < shmt (left shift)

Modification below:

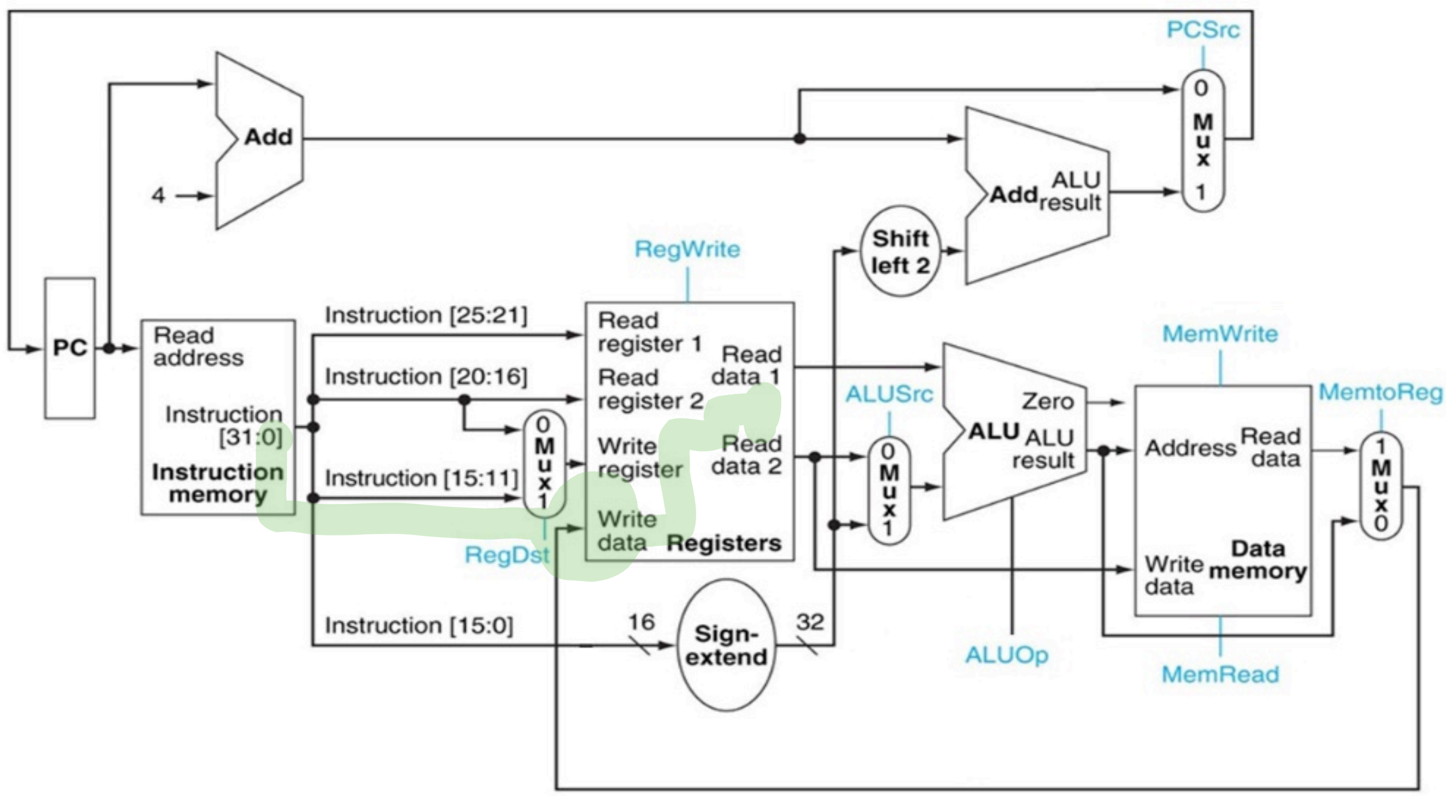


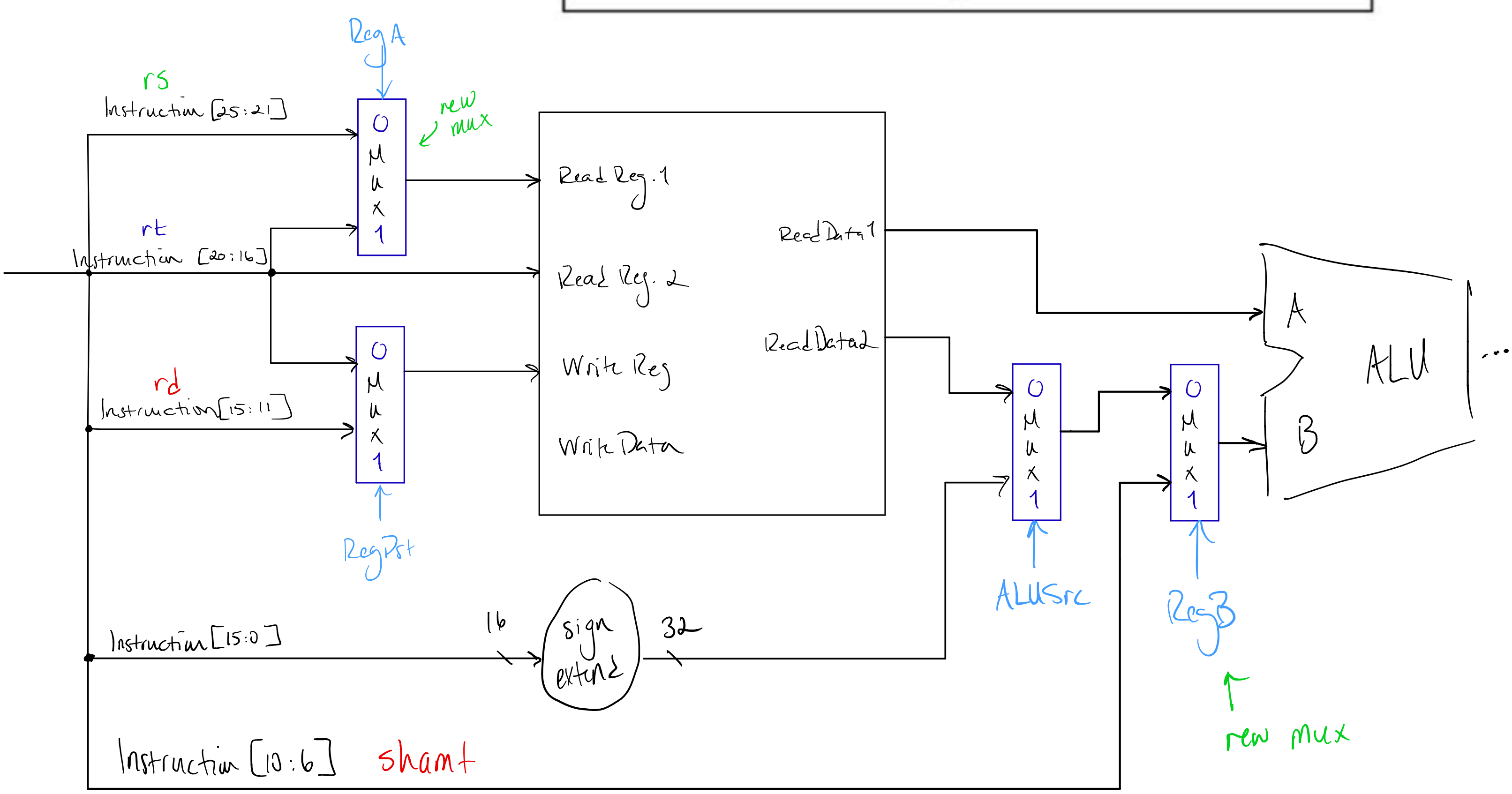
First Modification:



Input A of ALU (Read Data 1) can also come from Reg[rt] in addition to Reg[rs].

Second Modification:





Input B of ALU can also come from shamt (instruction [10:6])
in addition to the output of mux with ALUSrc as select signal.

*note, result from ALUSrc doesn't matter since RegB will be 1 & use shamt as input B *

Task 1: Implementation and Functional Verification for R and I type operations

- Controller: Using all the tables generated in Exercise 1, implement (write Verilog code) a controller for the given Datapath.

  The controller (slide 4) has 2 inputs: 6-bit opcode and 6-bit func. Its outputs are control signals listed in the table of Exercise 1 (Slide 10, 13 and 15).

- Datapath: Using slide 15, write the Verilog code to implement Datapath (structural way – connecting several modules (from Task 0) together as shown on Slide 15 (with your added muxes to support the shift operations)).

  It has two inputs: Clock, Reset and

  1 output: the output from the rightmost 2x1 32-bit mux on slide 15.

  All control signals should be declared as wire.

- Integrate the controller with the Datapath. Call/Instantiate Controller in the Datapath code.
- Run both behavioral and post-synthesis simulations. Write the testbench.

Initializing the Instruction Memory for testing

- In the folder “task1_r-type” in “DatapathComponents” zipped file, the contents of “rtype_inst_memory.txt” is already in the initialization block (initial begin … end) in the instruction memory component.

- The “r_type_with_answers.s” file shows the original source code with the expected results
  – Values in registers 8, 16, 17, 18, and 19 (RegFile[8], [16], [17], [18] and [19]) will be monitored during your post-routing simulation.
  – These registers correspond to t0, s0, s1, s2, and s3 in the commented parts of the “rtype_inst_memory.txt”

- After initializing the instruction memory, synthesize your design and run post-synthesis simulation.
- Bring registers 8, 16, 17, 18, and 19 (RegFile[8], [16], [17], [18] and [19]) from your RegisterFile to your simulation waveform
- Monitor the values in these registers.

Task 1 (R-type and I-type): Expected Output and Grading Scheme

- Task 1 (325 pts):
  - (10 pts) Complete slides 10 and 12
  - (315 pts) Post synthesis simulation for each instruction in Instruction memory (see the table on the right)

- Penalty Conditions
  - 30% penalty if work only in the behavioral simulation
  - maximum of 25% of the total score will be given if your processor does not function at all.

Cycle	Inst.	Value	Points
1	addi	Reg[16]=14	20
2	addi	Reg[17]=15	5
3	addi	Reg[18]=29	5
4	addi	Reg[19]=-15	5
5	add	Reg[8]=44	25
6	and	Reg[8]=12	25
7	mul	Reg[8]=210	25
8	or	Reg[8]=31	25
9	ori	Reg[8]=30	25
10	sub	Reg[8]=-15	25
11	clo	Reg[8]=28	25
12	clz	Reg[8]=27	25
13	slt	Reg[8]=1	20
14	slt	Reg[8]=0	5
15	sll	Reg[8]=60	25
16	srl	Reg[8]=3	25
		Total	315