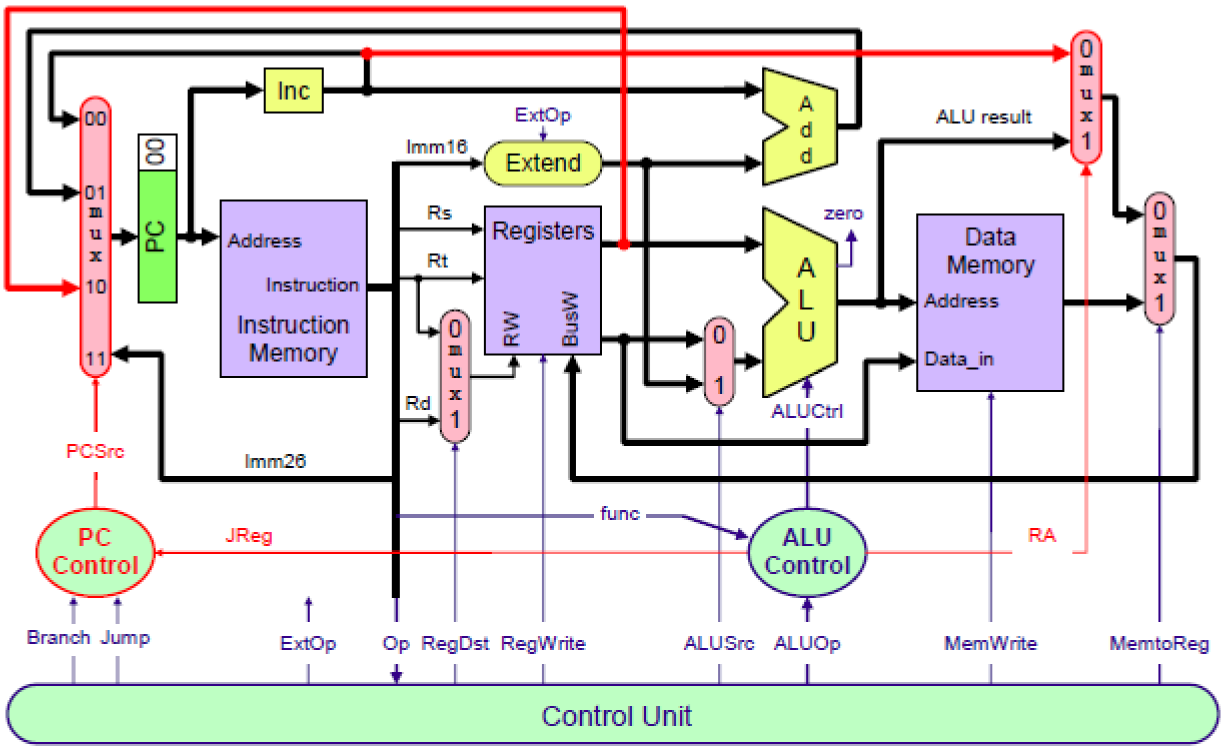
Q1. Single-Cycle MIPS Processor

We wish to add the instruction **jalr** (jump and link register) to the single-cycle datapath. The jump and link register instruction is described below:

```
    jalr rd, rs # rd = pc + 4 , pc = rs
```

op^6 = 0	rs^5	0	rd^5	0	$Funct^6$ = 0x9

a) Add any necessary datapath and control signals and draw the result datapath. You should only add wires, gates, muxes to the datapath; do not modify the main functional units (the memory, register file, and ALU) themselves. Try to keep your diagram neat!



The necessary changes to the datapath and control:

For the datapath, we need a bigger 4-input multiplexer at the input of the PC. The first input is used to increment the PC. The second input is used for taken branches, where the branch target is PC-relative. The third input is used to jump register, where the input to the PC comes from a general-purpose register, and the fourth input is used for jump instructions.

For the implementation of the JALR instruction: to jump to register 'Rs', we need to add a path from the output of register Rs (first ALU input) back to the PC multiplexer input. PC control unit needs to be updated by adding an input control signal JReg (Jump Register) to select PC according to the value of register Rs. JReg is generated by the ALU control unit, since JALR is a R-type instruction and JReg depends on the function field only. When JReg is equal to '1', PCSrc (PC control unit output control signal) will be '10' to select the value of register Rs as input to PC.

Also, we need to store PC+4 in register Rd. To accomplish this, we need another multiplexer to select between the incremented PC, the ALU result and data memory out, to be placed on BusW. Also, we need to add a path from the output of the incremented PC to the input of this new multiplexer. A control signal 'RA' (Return Address) is needed to select between the incremented PC and the ALU result. The MemtoReg multiplexer selects between the output of the 'RA' multiplexer and the Data Memory output to place on BusW.

b) Show the values of the control signals to control the execution of the **jalr** instruction. If you need add a new control signal, please add it along with its value to the table below. Use the following table for ALUCtrl.

ALU function	4-bit ALU Control
AND	0001
OR	0010
XOR	0011
ADD	0100
SUB	0101
SLT	0110

The main control signals for the JALR instruction are the same for other R-type instructions, such as ADD and SUB. The ALU Control signals for the JALR instruction require JReg = 1, RA = 0 and ALUCtrl is a don't care. These control signals are shown in the table below:

RegDst	RegWrite	ExtOp	ALUSrc	MemRead	MemWrite	MemtoReg	ALUCtrl	J	Beq	Bne	RA	JReg
Rd = 1	1	X	X	0	0	0	XXXX	0	0	0	0	1

Q2. Processor Performance

Suppose we add the multiply and divide instructions. The operation times are as follows:

Instruction memory access time = 190 ps, Data memory access time = 190 ps,
Register file read access time = 150 ps, Register file write access = 150 ps
ALU delay for basic instructions = 190 ps, ALU delay for multiply or divide = 550 ps
Ignore the other delays in the multiplexers, control unit, sign-extension, etc.

Assume the following instruction mix: 30% ALU, 15% multiply & divide, 20% load, 10% store, 15% branch, and 10% jump.

a) What is the total delay for each instruction class and the clock cycle for the single-cycle CPU design?

Instruction Class	Instruction Memory	Register Read	ALU	Data Memory	Register Write	Total Delay
Basic ALU	190 ps	150 ps	190 ps		150 ps	680 ps
Mul & Div	190 ps	150 ps	550 ps		150 ps	1040 ps
Load	190 ps	150 ps	190 ps	190 ps	150 ps	870 ps
Store	190 ps	150 ps	190 ps	190 ps		720 ps
Branch	190 ps	150 ps	190 ps			530 ps
Jump	190 ps	150 ps				340 ps

Clock cycle = max delay = 1040 ps.

b) Assume we fix the clock cycle to 200 ps for a multi-cycle CPU, what is the CPI for each instruction class and the speedup over a fixed-length clock cycle?

Solution:
CPI for Basic ALU = 4 cycles
CPI for Multiply & Divide = 6 cycles
CPI for Load = 5 cycles
CPI for Store = 4 cycles
CPI for Branch = 3 cycles
CPI for Jump = 2 cycles

Average CPI = 0.3 * 4 + 0.15 * 6 + 0.2* 5 + 0.1 * 4 + 0.15 * 3 + 0.1 * 2 = 4.15

Speedup of multi-cycle over single-cycle = (1040 * 1) / (200 * 4.15) = 1.253

Q3. (10 pts) Consider the following MIPS code sequence:

```
a: add $t0, $s0, $s1
b: sub $t1, $s2, $t0
c: xor $t0, $s0, $s1
d: or  $t2, $t1, $t0
```

a) (5 pts) Identify all the RAW dependencies between pairs of instructions.

Instruction b is dependent on instruction a ($t0)
Instruction d is dependent on instruction b ($t1)
Instruction d is dependent on instruction c ($t0)

b) (3 pts) Identify all the WAR dependencies between pairs of instructions
Instruction c is dependent on instruction b ($t0)

c) (2 pts) Identify all the WAW dependencies between pairs of instructions
Instruction c is dependent on instruction a ($t0)

Q4. (25 pts) Use the following MIPS code fragment:

```
        I1:   ADDI $3, $0, 100          # $3 = 100
        I2:   ADD       $4, $0, $0              # $4 = 0
  Loop:
        I3:   LW        $5, 0($1)               # $5 = MEM[$1]
        I4:   ADD  $4, $4, $5              # $4 = $4 + $5
        I5:   LW        $6, 0($2)               # $6 = MEM[$2]
        I6:   SUB  $4, $4, $6              # $4 = $4 - $6
        I7:   ADDI      $1, $1, 4               # $1 = $1 + 4
        I8:   ADDI      $2, $2, 4               # $2 = $2 + 4
        I9:   ADDI      $3, $3, -1              # $3 = $3 - 1
        I10:  BNE  $3, $0, Loop         # if ($3 != 0) goto Loop
```

a) (10 pts) Show the timing of one loop iteration on the 5-stage MIPS pipeline **without forwarding hardware**. Complete the timing table, showing all the stall cycles. Assume that the register write is in the first half of the clock cycle and the register read is in the second half. Also assume that the branch will stall the pipeline for 1 clock cycle only. Ignore the "startup cost" of the pipeline.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
I1	IF	ID	EX	MEM	WB																		
I2		IF	ID	EX	MEM	WB																	
I3			IF	ID	EX	MEM	WB																
I4				IF	Stall	Stall	ID	EX	MEM	WB													
I5							IF	ID	EX	MEM	WB												
I6								IF	Stall	Stall	ID	EX	MEM	WB									
I7											IF	ID	EX	MEM	WB								
I8												IF	ID	EX	MEM	WB							
I9													IF	ID	EX	MEM	WB						
I10														IF	Stall	Stall	ID	EX	MEM	WB			

I3																	IF	IF	ID	EX	MEM	WB		
I4																			IF	Stall	Stall	ID	EX	

b) According to the timing diagram of part (a), compute the number of clock cycles and the average CPI to execute ALL the iterations of the above loop.

There are 100 iterations

Each iteration requires 15 cycles =

8 cycles to start the 8 instructions in loop body + 7 stall cycles

There are 2 additional cycles to start the first 2 instructions before the loop.

Therefore, total cycles = 100 * 15 + 2 (can be ignored) = 1502 cycles ≈ 1500 cycles

Total instruction executed = 2 + 8 * 100 = 802 instructions (counting first two)

Average CPI = 1502 / 802 = 1.87

If we ignore first two instructions and the time to terminate last iteration then

Average CPI = 1500/800 = 1.88 (almost same answer)

c) Redo part (a) to show the timing of one loop iteration with **full forwarding** hardware. If forwarding happens, please show how the data is forwarded with an arrow.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

I1	IF	ID	EX	MEM	WB												
I2		IF	ID	EX	MEM	WB											
I3			IF	ID	EX	MEM	WB										
I4				IF	ID	EX	MEM	WB									
I5					IF	ID	EX	MEM	WB								
I6						IF	ID	EX	MEM	WB							
I7							IF	ID	EX	MEM	WB						
I8								IF	ID	EX	MEM	WB					
I9									IF	ID	EX	MEM	WB				
I10										IF	ID	EX	MEM	WB			
I3											IF	IF	ID	EX	MEM	WB	
I4													IF	ID	EX	MEM	WB

d) Reorder the instructions of the above loop to fill the load-delay and the branch delay slots, without changing the computation. Write the code of the modified loop.

```
      ADDI $3, $0, 100          # $3 = 100
      ADD $4, $0, $0        # $4 = 0
Loop:
      LW $5, 0($1)              # $5 = MEM[$1]
      LW $6, 0($2)              # Moved earlier to avoid load-delay
      ADDI $3, $3, -1       # Moved earlier
      ADD $4, $4, $5        # $4 = $4 + $5
```

```
ADDI $1, $1, 4        # $1 = $1 + 4
ADDI $2, $2, 4        # $2 = $2 + 4
BNE $3, $0, Loop           # if ($3 != 0) goto Loop
SUB $4, $4, $6        # Fills branch delay slot
```

e) (5 pts) Compute the number of cycles and the average CPI to execute ALL the iteration of the modified loop. What is the speedup factor?

There are 100 iterations

Each iteration requires 8 cycles =

8 cycles to start the 8 instructions in loop body + 0 stall cycles

There are 2 additional cycles to start the first 2 instructions before the loop

+ 4 additional cycles to terminate the ADDI instruction in the last iteration.

Therefore, total cycles = 100 * 8 + 6 (can be ignored) = 806 cycles ≈ 800 cycles

Total instruction executed = 2 + 8 * 100 = 802 instructions (counting first two)

Average CPI = 806 / 802 = 1.00

If we ignore first two instructions and the time to terminate last iteration then

Average CPI = 800/800 = 1.00 (almost same answer)

Speedup Factor = CPIpart-b/CPIpart-d = 1.88/1.00 = 1.88