แบบฝึกหัด 1:

เติมตารางต่อไปนี้ให้สมบูรณ์ โดยพิจารณาว่าในแต่ละคำสั่งสัญญาณตัวใดควรจะเป็น 0 หรือ 1 หรือ DC (Don't Care)

สำหรับสัญญาณ ALUOp ที่ส่งเข้าไปที่ ALU control ก่อนที่จะได้สัญญาณ output ไปควบคุม ALU อีกต่อหนึ่ง ให้รวบรัดตอบเป็นคำอธิบายฟังก์ชั่น ALU ที่ถูกเลือกในคอลัมน์สุดท้ายเลย

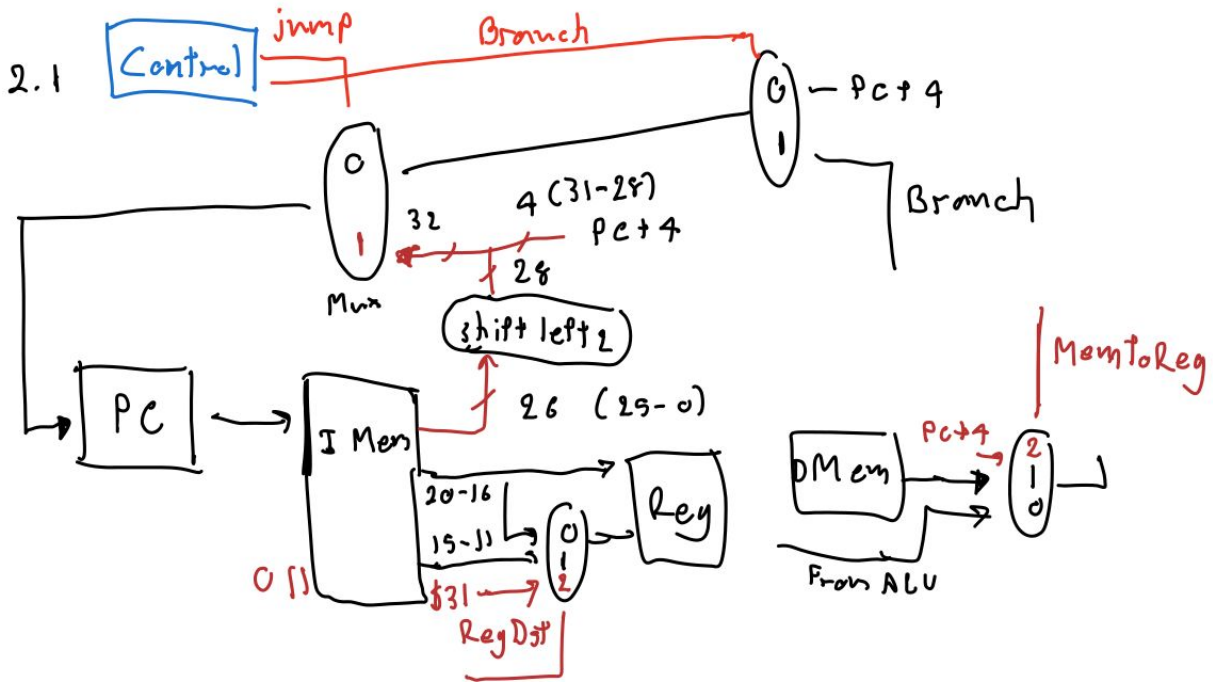
สัญญาณ zero ที่ออกจาก ALU เกิดจากการทำปฏิบัติการลบแล้วดูว่าผลลัพธ์เป็นศูนย์หรือไม่ นั่นคือเป็นการเปรียบเทียบว่าค่าสองค่าเท่ากันหรือไม่ เพื่อตัดสินใจว่าจะ branch ไปที่ target หรือไม่ branch

Instruction	RegDst	ALUSrc	Memto Reg	RegWrite	MemRead	Mem Write	Branch	ALUOP
addu	1	0	0	1	0	0	0	add
addiu	0	1	0	1	0	0	0	add
xori	1	1	0	1	0	0	0	sub
slt	1	0	0	1	0	0	0	slt
subu	1	0	0	1	0	0	0	sub
lw	0	0	1	1	1	0	0	add
sw	dc	1	dc	0	0	1	0	add
beq	dc	0	dc	0	0	0	1	sub
bne	dc	0	dc	0	0	0	1	sub

แบบฝึกหัด 2:

แสดงส่วน datapath และ control ที่ต้องเพิ่มเติมในภาพในแบบฝึกหัดที่ 1 เพื่อรองรับคำสั่ง j และ jal ที่เป็นคำสั่งประเภท J-type สำหรับคำสั่ง jal ต้องระวังให้มากเพราะจะมีการอัพเดททั้งค่า PC และค่า register $ra ด้วย

อธิบายสิ่งที่เพิ่มเติมเข้าไปและโต้แย้งว่าคำสั่ง j และ jal จะประมวลผลได้ถูกต้องถ้ามีส่วนที่เพิ่มเข้าไปนี้



Instruction	RegDst	ALUSrc	Memto Reg	Reg Write	Mem Read	Mem Write	Branch	Jump	ALUOP
j	dc	dc	dc	0	0	0	0	1	-
jal	2	0	2	1	0	0	0	1	-

คำสั่ง j จะใช้บิตที่ 0-26 บอก address จากนั้นจะทำการ jump ไปตำแหน่งนั้นๆ ซึ่งส่วนที่เพิ่มไปเพียงพอต่อการทำคำสั่ง j โดยไม่เกิดข้อผิดพลาด
คำสั่ง jal จะต้องทำการ store pc+4 ลงใน $ra(31) จากนั้นจึงค่อย jump ไปยังตำแหน่งตามบิตที่ใส่ไป เหมือนคำสั่ง j ดังนั้นสิ่งที่เพิ่มเติมต่อจาก j จึงต้องเพิ่มทางเลือกสำหรับเลือก address $ra และ store pc+4 ลง register