

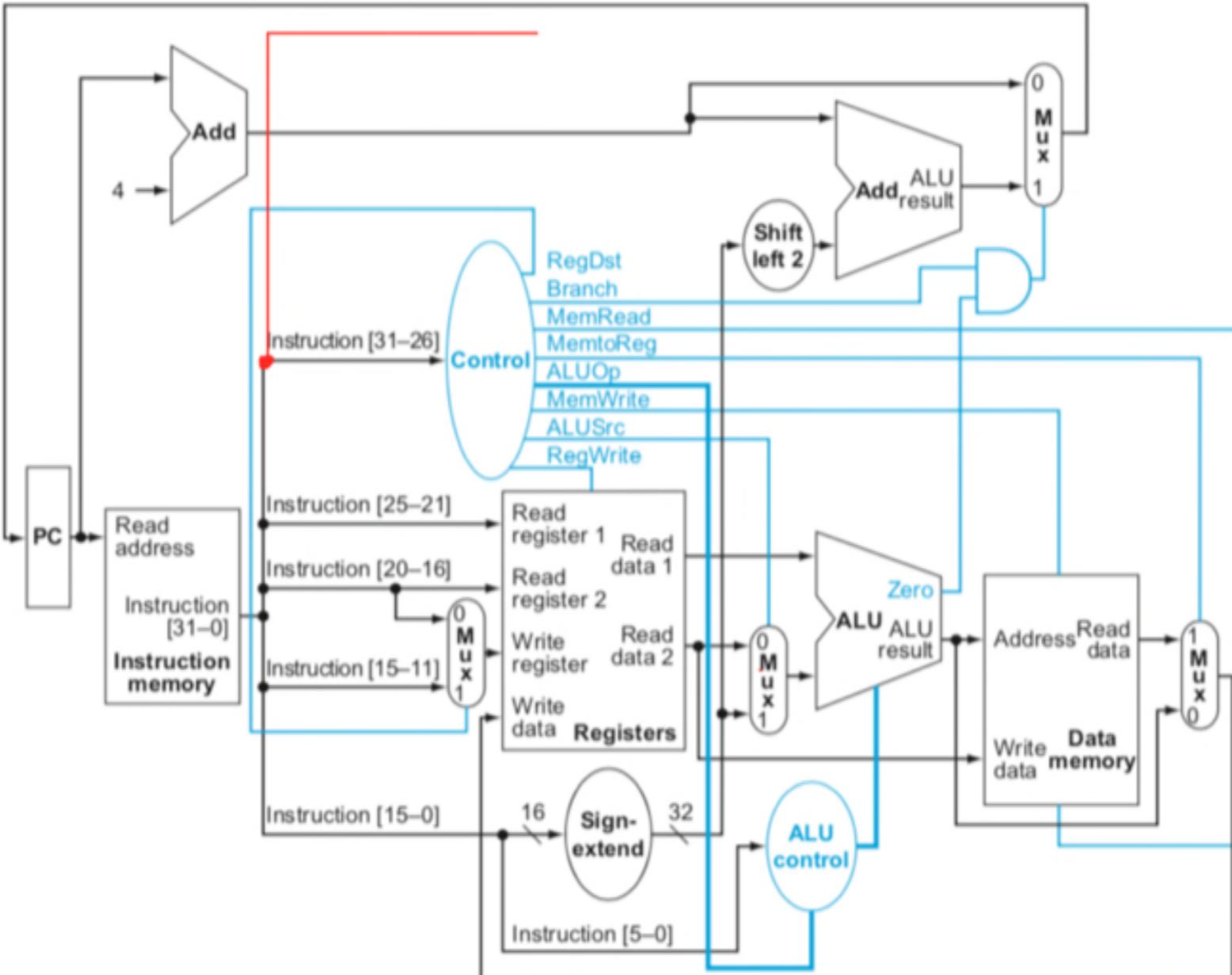
แบบฝึกหัดเกี่ยวกับสัญญาณ control และ  
datapath สำหรับ single-cycle MIPS

## แบบฝึกหัด 1:

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

สำหรับสัญญาณ ALUOp ที่ส่งเข้าไปที่ ALU control ก่อนที่จะได้สัญญาณ output ไปควบคุม ALU อีกต่อหนึ่ง ให้ระบุรดตอบเป็นคำอธิบายพังก์ชัน ALU ที่ถูกเลือกในคลัมมน์ สุดท้ายเลย

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

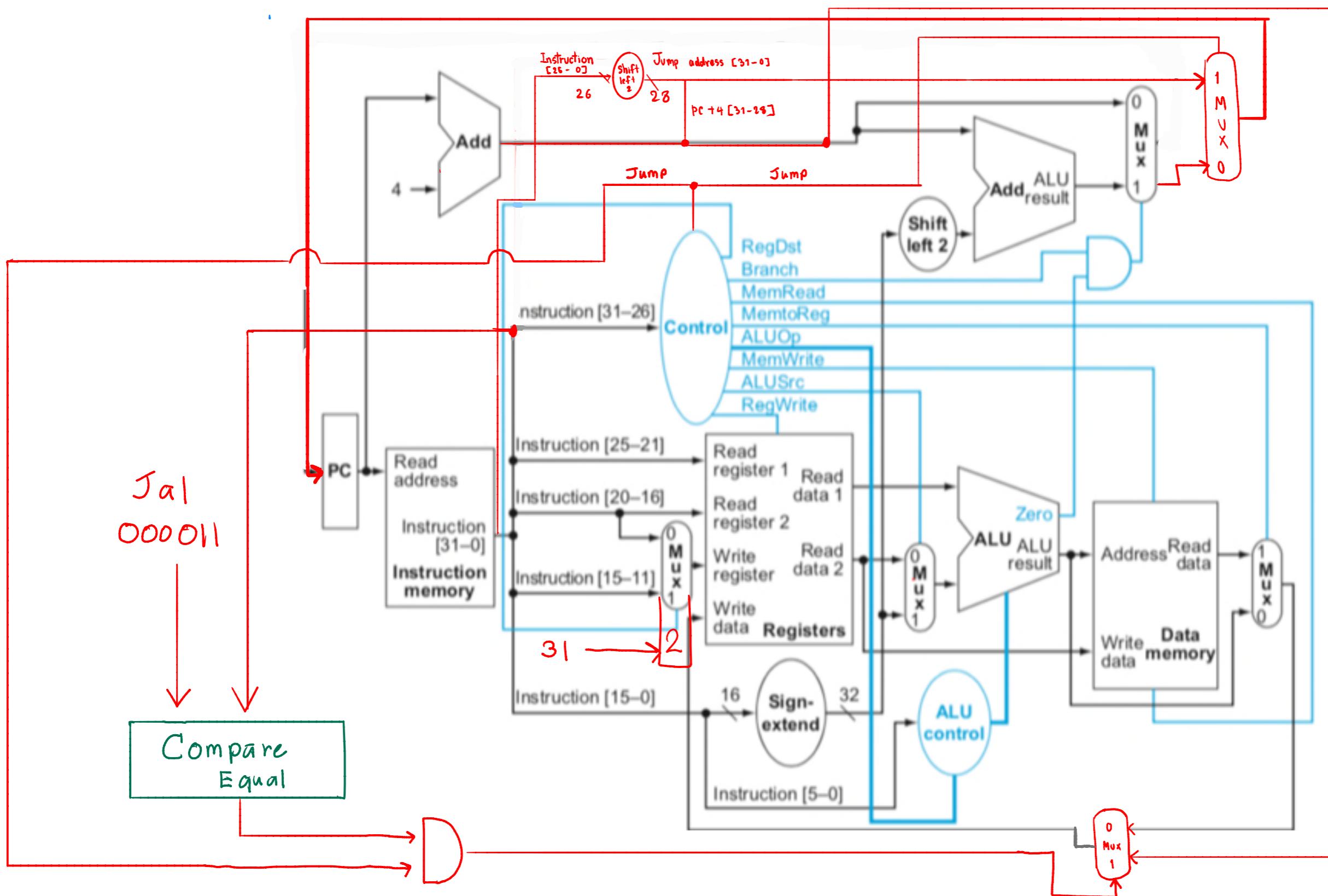


| Instruction | RegDst | ALUSrc | MemtoReg | RegWrite | MemRead | MemWrite | Branch | ปฏิบัติการที่ ALU กระทำการ  |
|-------------|--------|--------|----------|----------|---------|----------|--------|---|
| addu        | 1      | 0      | 0        | 1        | 0       | 0        | 0      | นำ register 2 ตัวอักษรกับ register 1 แบบ Unsigned                 |
| addiu       | 0      | 1      | 0        | 1        | 0       | 0        | 0      | นำ register 2 ตัวอักษร immediate ไปบวกกับ register 1 แบบ Unsigned |
| xori        | 0      | 1      | 0        | 1        | 0       | 0        | 0      | นำ register 1 บวก immediate แล้ว xor ให้ได้ผลลัพธ์                |
| slt         | 1      | 0      | 0        | 1        | 0       | 0        | 0      | บวกขึ้นไปลบค่าแบบน้อยกว่า   |
| subu        | 1      | 0      | 0        | 1        | 0       | 0        | 0      | นำ register 2 ตัวอักษรกับ register 1 แบบ Unsigned                 |
| lw          | 0      | 1      | 1        | 1        | 1       | 0        | 0      | load ค่าลง register สำหรับ  |
| sw          | 0      | 1      | 0        | 1        | 0       | 1        | 0      | store ค่าลงใน register  |
| beq         | 0      | 1      | 0        | 1        | 0       | 0        | 1      | branch ถ้า register 2 ต่างกัน                                     |
| bne         | 0      | 1      | 0        | 1        | 0       | 0        | 1      | branch ถ้า register 2 ไม่เท่ากัน                                  |

## แบบฝึกหัด 2:

แสดงส่วน datapath และ control ที่ต้องเพิ่มเติมในภาพในแบบฝึกหัดที่ 1 เพื่อรับคำสั่ง j และ jal ที่เป็นคำสั่งประเภท J-type สำหรับคำสั่ง jal ต้องระวังให้มาก เพราะจะมีการอัพเดททั้งค่า PC และค่า register \$ra ด้วย

อธิบายสิ่งที่เพิ่มเติมเข้าไปและโต้แย้งว่าคำสั่ง j และ jal จะประมวลผลได้ถูกต้องถ้ามีส่วนที่เพิ่มเข้าไปนี้



| Instruction | RegDest | ALUSrc | MemtoReg | RegWrite | MemRead | MemWrite | Branch | Jump | ปฏิบัติการที่ ALU ทำ                               |
|-------------|---------|--------|----------|----------|---------|----------|--------|------|--|
| j           | DC      | DC     | DC       | O        | O       | O        | O      | 1    | โคลี่ปั้ง target โดยไม่บวก offset address          |
| jal         | 10      | DC     | DC       | 1        | O       | O        | O      | 1    | โคลี่ปั้ง target โดยหัก offset address ไว้กับ \$ra |

ଓଡ଼ିଆ :

1. វិន័យនៃ Jump ដែលអាចបង្កើតឡើងមកពី Instruction memory

ឯកជាតិចង្វារ នឹង តាត់ bit នៃ 0 តាំង 25 តើម្ខានវិនាទនេរ immediate

ຈາກນີ້ໃຊ້ shift extend ດັກ 2 bit ແລ້ວໃຫ້ bit ໃນ PC +4 bit

ပေါင်းမြန်မာစာ မျှေးဆုံးလုပ်ခွင့် ပေါင်းမြန်မာစာ မျှေးဆုံးလုပ်ခွင့်

លាសីអ្នកចាប់បន្ទាន់ក្នុងការការពារក្នុងការការពារ

উন্মুক্ত জাতিকর্মসূলি জ = সামাজিক মান এবং ইতিহাসের উপর পুরো বিশ্বের জন্য

2. ໃນລັບນັກ Jal (Jump and link) ຂໍ້າກນິ້າໂປ່ງຂຸລວາກ Instruction

memory ຕົວຢ່າງ bit ກໍ 31 ດີ 26 ຫີ້ສັງເກົ່າທຸລະ b bit ຈຶ່ນຍົກ compare

equal | ກັນ | PC | 40000011) ເນື່ອ check ວ່າ ເມື່ນ Jal ນີ້ໂປ່ງ ໄດ້

ຈຶ່ນຍົກລົດທີ່ ຂາ and ກັນ Control Jump ຈະໄດ້ຕ່າງປະຕົມ ຖໍ່ mux

ດົກການ ຫີ້ສັນ mux ດ້ວຍໂລດ 0 ຈຶ່ນຕ່າມາກ memory ດ້ວຍໂລດ 1 ຈຶ່ນຕ່າມ  
ມາກາ PC+4 ດັກລົດທີ່ໄດ້ຈຳນິ້າໃນ write back ໃນ register ຫີ້ເກົ່າທັງການ register \$ra  
ແລ້ວ ພົບເກົ່າທັງໝົດ ໂດຍມີຕົວເລື່ອກົດ mux ທີ່ໃຫ້ Control unit Reg Dest ຕ້ອງແນ່ນເນື່ອ 2 bit

ຜົນໜີນ ຈະໄດ້ວ່າດີ່ນໍ້າ Jal ຈະສາມາດປ່ານມານ ລວໄດ້ດູກຕົວ ຕາມກຳກຳລ່າຍົງຕົນ