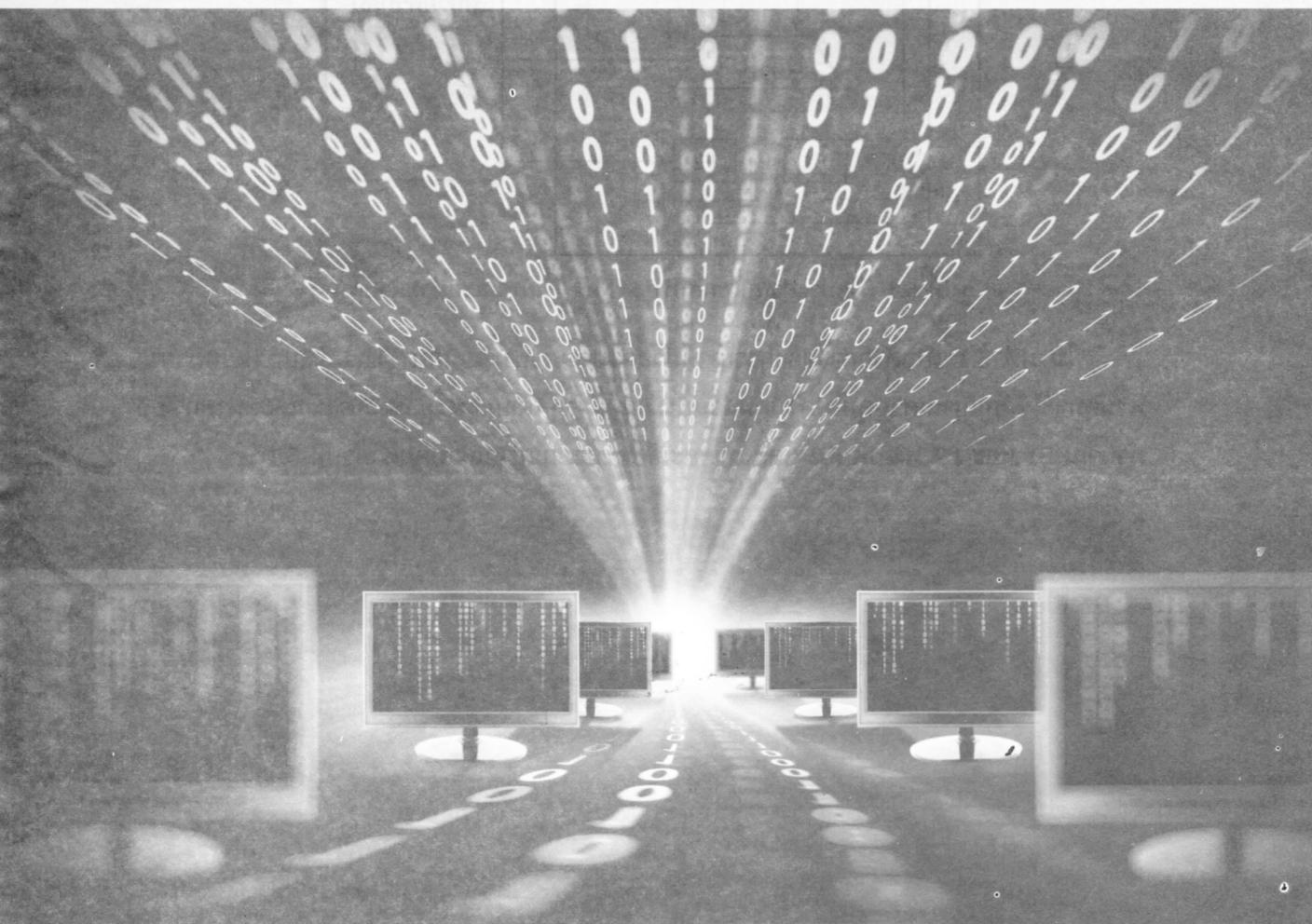
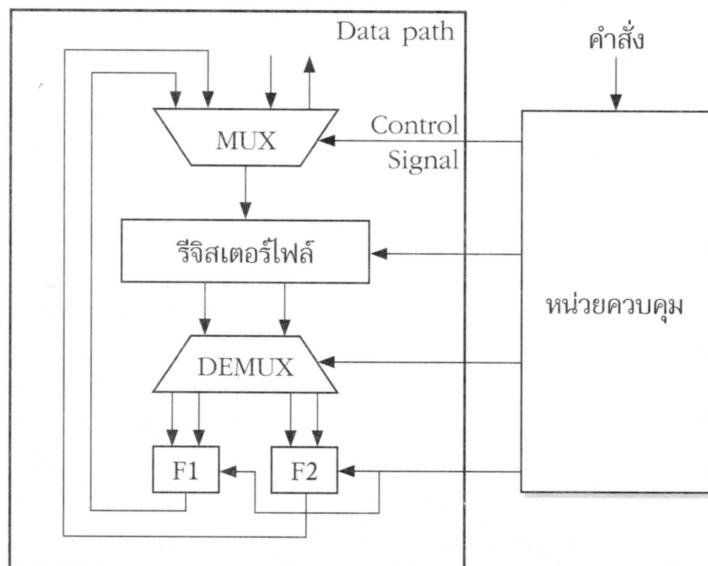


5

การออกแบบหน่วยควบคุม (Control Unit Design)

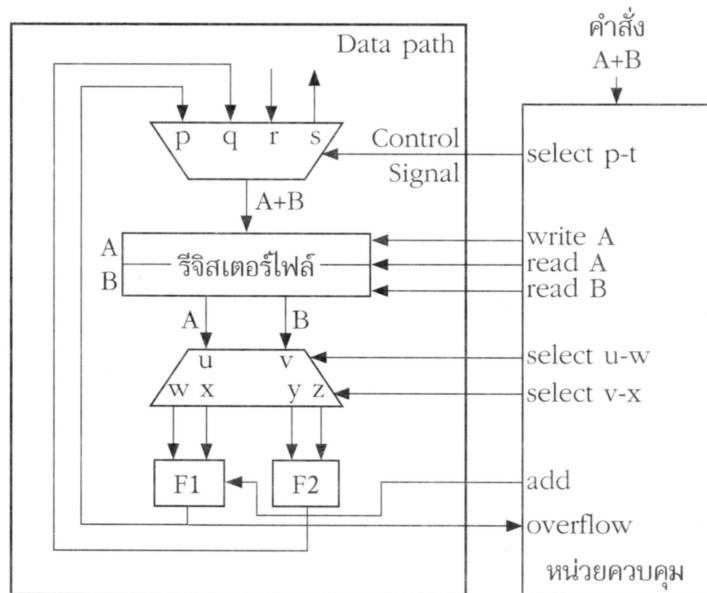


หน่วยควบคุม (Control Unit) มีหน้าที่สร้างสัญญาณควบคุมสำหรับ Data Path ใน Data Path จะมีจุดต่างๆ ที่รอสัญญาณควบคุมที่เหมาะสมในการควบคุมหรือเลือกการทำงานแต่ละหน่วย เช่น การเลือกฟังก์ชันการทำงานของ ALU หรือสำหรับ MUX ใน การส่งข้อมูลไปยังหน่วยต่างๆ ของระบบ เช่น ไปเก็บยังรีจิสเตอร์ หรือไปเป็นโอดีโอเพนเดนต์ของ ALU เป็นต้น ในบทนี้จะกล่าวถึงเทคนิคที่ใช้ในการออกแบบหน่วยควบคุมและตัวอย่างการออกแบบใน MIPS



รูปที่ 5.1 ตัวอย่างส่วนประกอบของ Data Path และหน่วยควบคุม

ในรูปที่ 5.1 เป็นตัวอย่างของการเอาหน่วยควบคุมมาประกอบกับ Data Path โดยหน่วยควบคุมจะให้สัญญาณควบคุมไปควบคุมส่วนประกอบต่างๆ ได้แก่ MUX, รีจิสเตอร์ และหน่วยการคำนวณ F1 และ F2 ในรูปที่ 5.2 แสดงรายละเอียดของสัญญาณควบคุมตามรูปที่ 5.1



รูปที่ 5.2 รายละเอียดสัญญาณควบคุมของรูปที่ 5.1

ตัวอย่างเช่น ถ้าทำงานคำสั่ง ADD A, B ตามสัญญาณควบคุมในรูปที่ 5.2 หน่วย F1 จะถูกเลือกให้ทำการบวก (สัญญาณ Add) ถ้าการบวกเลขมี Overflow เกิดขึ้น จะให้ค่า Overflow = 1 ไปยังหน่วยควบคุม ส่วนสัญญาณ Select v-x ทำ Map อินพุต v ไปที่เอาต์พุตของ Demux ในขา x ผลลัพธ์ของการบวก A + B ที่ได้จาก F1 จะถูกส่งไปยัง MUX และสัญญาณ Select เลือก p-t เป็นเอาต์พุตของ MUX เพื่อให้ A + B ไปเก็บในรีจิสเตอร์ไฟล์ RF ชื่องควบคุม โดยสัญญาณ WriteA

นอกจากนี้ ถ้าเป็นคำสั่งที่ใช้การเอ็คซ์คิวต์หล่ายaise ก็ เช่น การบวกเลข Double อาจจะใช้ Data Path นี้ได โดยในลำดับแรกจะทำการบวกเลข 8 บิตล่าง และในไบต์ถัดไปทำการบวกเลข 8 บิตบนกับตัวที่ได้จาก 8 บิตล่าง คำสั่งประเภทหล่ายaise เกินนี้ต้องใช้สัญญาณควบคุมเข้าควบคุมเข่นกัน

สำหรับประโยชน์เงื่อนไขก็จะเป็นต้องใช้สัญญาณควบคุม ในโปรแกรมที่ใช้ตัวแปรที่ใช้ในเงื่อนไขนั้น ตัวแปรเงื่อนไขจะใช้เลือกลำดับการทำงาน กล่าวคือ ในเวลาถัดไปการทำงานคำสั่งที่ตำแหน่งเดือนกับค่าของตัวแปรเงื่อนไข เชน ในโค๊ดต่อไปนี้ ซึ่งใช้เงื่อนไขจาก Cond1 และ Cond2 เราจึงนำมาสร้างเป็นตารางค่าความจริงดังรูปต่อไป

```

Loop    : if Cond1 = true then ADD A,B
          else SUB A,B
          If Cond2 = true then goto Output
          else goto loop
Output : ...
  
```

| ตำแหน่งปัจจุบัน | Condition Select | ตำแหน่งถัดไป | | Function Select | |
|-----------------|------------------|--------------|----------|-----------------|----------|
| | | C = True | C ≠ True | C = True | C ≠ True |
| Adr1 | Cond1 | Adr2 | Adr2 | Add | Sub |
| Adr2 | Cond2 | Adr3 | Adr1 | | |
| Adr3 | | | | | |

ในตาราง Adr1 หมายถึง ตำแหน่งของประโยค if Cond1 = true และ Adr2 คือ ตำแหน่งของประโยค then และ else คือ การ ADD A, B หรือ SUB A, B ตารางข้างบนสังเกตว่า จากตำแหน่งปัจจุบันในคอลัมน์ ตำแหน่งปัจจุบันคือ แล้ว Addr Adr1 จะไปทำงานที่ Adr2 ตลอดเวลา แต่ Cond1 จะทำคำสั่ง Add หรือ Sub ขึ้นอยู่กับว่า Cond1 = true หรือไม่ ถ้าเป็นจริง จะส่งสัญญาณไปเลือกฟังก์ชัน Add ของ ALU ให้ทำการบวก ถ้าเป็นเท็จ จะส่งสัญญาณไปเลือกฟังก์ชัน Sub เพื่อให้ ALU ทำการลบ สำหรับกรณีตำแหน่ง Addr = Adr2 จะไปทำงานที่ตำแหน่ง Adr3 หรือไปยังตำแหน่ง Adr1 ขึ้นอยู่กับ Cond2 ว่าเป็นจริงหรือไม่ เช่นกัน

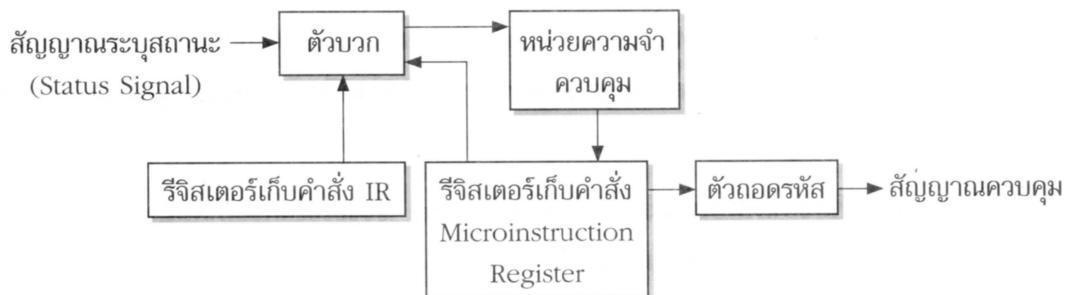
ตารางดังกล่าวเป็นตารางบอกพฤติกรรมของหน่วยควบคุมในการอิมเพลเม้นต์พฤติกรรมของตารางนี้ในฮาร์ดแวร์ ทำได้ 2 แบบ คือ แบบฮาร์ดไวร์ (Hardwire) และแบบไมโครโปรแกรม (Micro Program)

สำหรับแบบฮาร์ดไวร์ พฤติกรรมนี้จะถูกสร้างออกมากเป็นฮาร์ดแวร์ โดยอาจจะบันทึกลงในรอม และไม่สามารถแก้ไขเปลี่ยนแปลงพฤติกรรมได้อีก แต่หน่วยควบคุมจะทำงานได้เร็ว เพราะฮาร์ดแวร์จะทำงานเร็วกว่า การสร้างฮาร์ดแวร์สำหรับพฤติกรรมดังกล่าวจะใช้วิธีการสร้างเครื่องยนต์สถานะจำกัด (Finite State Machine) โดยสร้างตารางสถานะ (State Table) ดังแสดงในตารางข้างล่าง

| สถานะปัจจุบัน (Current State) | อินพุตปัจจุบัน (Current Input) | สถานะถัดไป | ເອົາຕົມປັບປຸງ | |
|----------------------------------|-----------------------------------|------------|---------------|-------|
| | | | Function | |
| Adr1 | Cond1 = 1 | Adr2 | Add | |
| Adr1 | Cond1 = 0 | Adr2 | Sub | |
| Adr2 | Cond2 = 1 | Adr3 | | |
| Adr2 | Cond2 = 0 | Adr1 | | |

ช่องสถานะปัจจุบันจะแสดงสถานะปัจจุบันซึ่งก็คือ ตำแหน่งในโปรแกรม และช่องสถานะถัดไปจะหมายถึง สถานะใหม่ที่จะไปทำงาน ส่วนสัญญาณควบคุมที่จะເອົາຕົມແດດໃນຄອລັມນີ້ Function ภายใต้ເອົາຕົມປັບປຸງ โดยขึ้นอยู่กับเงื่อนไขอินพุตว่าเป็น Cond1 และ Cond2 เป็นค่าใด (ในช่วงสถานะอินพุต) จากตารางดังกล่าว เราสามารถนำไปสร้างเป็นเครื่องยนต์สถานะและวงจรลำดับ (Sequential Circuit) ได้

สำหรับแบบไมโครโปรแกรมจะใช้หน่วยความจำควบคุม (Control Memory) เพื่อเก็บโปรแกรมของพฤติกรรมนี้ และเขียนไมโครโปรแกรมเพื่อจำลองพฤติกรรมนี้ ลักษณะโครงสร้างภายในของหน่วยควบคุมแบบใช้ไมโครโปรแกรมดังรูปที่ 5.3



รูปที่ 5.3 โครงสร้างหน่วยควบคุมแบบใช้ไมโครโปรแกรม

IR เป็นรีจิสเตอร์ที่เก็บคำสั่งที่อ่านมาแต่ละคำสั่งในไมโครโปรแกรมที่เก็บในหน่วยความจำควบคุม (Control Memory) และค่าใน Micro-Instruction Register บางฟิลด์จะแสดงว่า ณ เวลานี้จะต้องส่งสัญญาณใดบ้าง ส่วนตัวบวกจะใช้ช่วยการคำนวณตำแหน่งถัดไปในการทำงาน

ลักษณะการกำหนดพฤติกรรมในหน่วยความจำ เช่นนี้ ทำให้สามารถเปลี่ยนแปลงไมโครโปรแกรมได้ ดังนั้น จึงมีความยืดหยุ่นมากกว่าการออกแบบโดยใช้รีชาร์ดไวร์ ในส่วนต่อไปจะอธิบายรายละเอียดของแต่ละวิธี

■ 5.1 การควบคุมแบบฮาร์ดไวร์ (Hardwire Control)

วิธีไมโครโปรแกรมและไฮาร์ดไวร์มีข้อดีและข้อเสียแตกต่างกันทั้งในด้านราคาและความเร็ว วิธีการไฮาร์ดไวร์จะใช้ต้นทุนสูงในการสร้างไฮาร์ดแวร์ แต่จะทำงานได้เร็วกว่า ส่วนวิธีของไมโครโปรแกรม มีต้นทุนต่ำกว่า แต่ทำงานช้ากว่า

ในการสร้างหน่วยควบคุมโดยวิธีไฮาร์ดไวร์ เราจะใช้รีชาร์ดไวร์เดียวกับการสร้างวงจรลำดับ โดยการสร้างตารางขึ้นมาก่อน และสร้างไฮาร์ดแวร์จากตารางสถานะนั้นๆ วิธีการสร้างตารางสถานะมี 2 วิธี คือ

1. วิธีตั้งเดิม (Classical Approach) เป็นวิธีการสร้างวงจรลำดับ โดยจำนวนฟลิปฟลอปที่ใช้ขึ้นอยู่กับจำนวนสถานะที่มีอยู่ ถ้าสถานะที่เป็นไปได้ทั้งหมดมี N จำนวนฟลิปฟลอป คือ $\log_2 N$
2. วิธีวันshot (One Hot Method) วิธีนี้ใช้ฟลิปฟลอป 1 ตัวต่อ 1 สถานะ ดังจะได้กล่าวต่อไป

ในทั้ง 2 วิธีจะใช้ตารางสถานะเพื่อเก็บลักษณะการเปลี่ยนสถานะ S_i โดยแต่ละແรมของตารางจะแสดงว่าสำหรับแต่ละอินพุตและสถานะปัจจุบันค่าหนึ่งๆ จะให้ออตพุตเป็นอะไรและเปลี่ยนสถานะไปอย่างไร ในการออกแบบตารางสถานะมี 2 แบบ คือ

1. เครื่องยนต์แบบมอร์ (Moore Machine) ซึ่งจะใช้ในกรณีที่ผลลัพธ์ของการทำงานขึ้นอยู่กับสถานะปัจจุบันอย่างเดียว
2. เครื่องยนต์แบบเมียลี (Mealy Machine) จะถือว่าผลลัพธ์ของการทำงานจะขึ้นอยู่กับสถานะปัจจุบันและอินพุตใหม่ที่เข้ามา

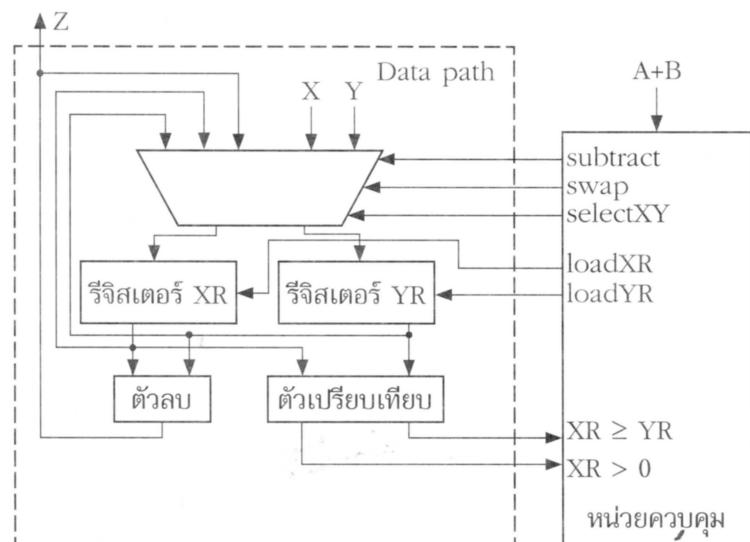
ต่อไปนี้เป็นตัวอย่างการออกแบบหน่วยควบคุมของໂພຣເຊເໜ່ອຣ໌ GCD ซึ่งเป็นໂພຣເຊເໜ່ອຣ໌ เพื่อใช้คำนวนหาตัวหารร่วมมาก โดยใช้อัลกอริົມแบบยูคลิด (Euclid)

```

GCD (x,y : in, z : out)
{
    XR := x;
    YR := y;
    while (XR > 0) {
        if XR ≤ YR then
        {
            temp := YR;
            YR := XR;
            XR := temp;
        }
        XR := XR - YR;
    }
    z := YR;
}

```

อัลกอริธึมนี้ใช้การลบเป็นหลัก โดยกำหนดให้ตัวตั้ง XR มากกว่าตัวลบ YR เสมอ และจะวนจนกว่าผลต่างจะเป็น 0 ลักษณะของ Data Path สำหรับการออกแบบอัลกอริธึมแสดงดังรูปที่ 5.4 ในที่นี้ใช้หน่วยคำนวณ 2 ตัวคือ ตัวลบ (Subtractor) ใช้สำหรับการลบและตัวเปรียบเทียบ (Comparator) ใช้สำหรับการเปรียบเทียบ $XR > 0$ และ $XR \leq YR$ และใช้รีจิสเตอร์ 2 ตัว คือ XR และ YR



รูปที่ 5.4 Data Path ของโปรเซสเซอร์ GCD

MUX จะเป็นตัวเลือกว่าจะส่งค่าใดไปให้รีจิสเตอร์ XR และ YR ตัว MUX สามารถ route ข้อมูลจากภายนอก x, y ไปที่รีจิสเตอร์ XR และ YR ได้โดยกำหนดสัญญาณ select x-y หรือสามารถนำค่าจากรีจิสเตอร์ XR และ YR ไปให้ที่ YR และ XR ตามลำดับ ซึ่งเป็นการสลับค่าในรีจิสเตอร์ทั้งสอง ทำได้โดยกำหนดสัญญาณ swap หรือไปเอาค่าจากผลลัพธ์ระหว่าง XR และ YR ไปเก็บในรีจิสเตอร์ XR กำหนดโดยสัญญาณ subtract ส่วนรีจิสเตอร์ XR และ YR ใช้สัญญาณ loadXR และ loadYR ตามลำดับ เมื่อต้องการให้ค่าใหม่กับรีจิสเตอร์ดังกล่าว ตัวเบรียบเทียบในรูปนี้ต้องการโอลอแวนด์ XR และ YR และให้ผลลัพธ์การเบรียบเทียบได้ 2 แบบ คือ ถ้า $XR \geq YR$ เป็นจริงให้ค่า 1 และถ้า $XR \geq 0$ เป็นจริงให้ค่า 1 สัญญาณทั้งสองนี้เป็นสัญญาณอินพุตของหน่วยควบคุมเพื่อใช้ในการตัดสินใจทำงานต่อไป โดยทำการเลือกว่าจะส่งสัญญาณเอกสารพุดได้บ้าง ออกไป เช่น ถ้า $XR \geq YR$ เป็น 0 หมายความว่า $XR < YR$ ดังนั้น จะต้องทำการสลับที่ XR และ YR ดังเช่นในอัลกอริธึม จึงต้องให้สัญญาณ swap เป็น 1 เพื่อส่งค่า XR ไปที่ YR และส่งค่า YR ไปที่ XR ตามลำดับ และต้องให้สัญญาณ load XR และ load YR เป็น 1 เพื่อทำการอ่านค่าใหม่ เข้าไปในรีจิสเตอร์ทั้งสอง เป็นต้น

จากอัลกอริธึมของ GCD สามารถแบ่งสถานะการทำงานออกเป็น 4 สถานะ คือ

S_0 สถานะเริ่มต้น (Start)

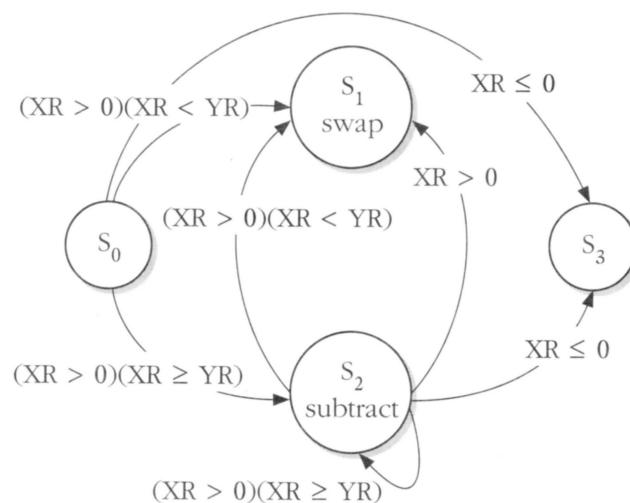
S_1 สถานะการสลับที่ (Swap)

S_2 สถานะการลบ (Subtract)

S_3 สถานะสิ้นสุด (End)

กำหนดให้สัญญาณอินพุตใช้ 2 บิตอยู่ในรูป x_1x_2 โดย x_1 หมายถึง สัญญาณ $XR > 0$ และ x_2 หมายถึง สัญญาณ $XR \geq YR$ ส่วนเอกสารพุดคือสัญญาณต่างๆ ที่หน่วยควบคุมให้ ได้แก่ subtract, swap, selectXY, loadXR และ loadYR

ถ้าสัญญาณ x_1 เป็น 0 ซึ่งหมายถึง $XR \geq 0$ เป็น 0 (หรืออีกนัยหนึ่ง $XR \leq 0$) ดังนั้น ลูป ในอัลกอริธึมจะไม่ทำงาน จึงไม่ต้องสนใจว่า x_2 จะเป็นอะไร แต่ถ้า $x_1 = 1$ หรือ $XR \geq 0$ ลูปจะเริ่มการคำนวณ GCD โดยจะสลับที่ก่อนหรือไม่ก็ขึ้นอยู่กับว่า $XR > YR$ หรือไม่ ถ้า $XR > YR$ หรือ $x_2 = 0$ ก็จะให้สัญญาณ Subtract เพื่อทำการลบ แต่ไม่เข่นนั้นก็จะให้สัญญาณ Swap ก่อน เพื่อทำการสลับที่ค่าใน XR และ YR และจึงทำการลบ ซึ่งกำหนดด้วยสัญญาณ Subtract ต่อไป พฤติกรรมนี้สามารถสร้างเป็นไดอะแกรมได้ดังรูปที่ 5.5



รูปที่ 5.5 ไดอะแกรมสถานะและตารางสถานะของโปรแกรมเชอร์ GCD

ถ้าอยู่ใน S_0 คือหมายถึง เริ่มต้นจะต้องการเอาต์พุตสัญญาณ selectXY, loadXR และ loadYR เพื่ออ่านค่า x และ y มาจากภายนอก และเข้าไปเก็บในรีจิสเตอร์ XR และ YR ตามลำดับ ถ้าอยู่ในสถานะ S_1 จะต้องให้สัญญาณ Swap, loadXR และ loadYR เพราะจะต้องการสลับค่าใน XR และ YR ถ้าอยู่ในสถานะ S_2 จะต้องคำนวณการลบระหว่าง XR และ YR จึงต้องส่งสัญญาณ Subtract และหลังจากลบแล้ว ค่า XR จะเก็บค่าผลลบที่ได้ โดยผลลบจะถูก Load ไปยังรีจิสเตอร์ XR ตามประโยค $XR := XR - YR$ จึงต้องให้สัญญาณ loadXR ออกมา ตารางต่อไปนี้แสดงตารางสถานะโดยพฤติกรรมของไดอะแกรมดังกล่าวและแสดงสัญญาณเอาต์พุต

| สถานะ | อินพุต ($XR > 0$) $(XR \geq YR)$ | | | เอาต์พุต | | | | |
|-------|---------------------------------------|-------|-------|----------|------|----------|--------|--------|
| | 0- | 10 | 11 | Subtract | Swap | SelectXY | LoadXR | LoadYR |
| S_0 | S_3 | S_1 | S_2 | 0 | 0 | 1 | 1 | 1 |
| S_1 | S_2 | S_2 | S_2 | 0 | 1 | 0 | 1 | 1 |
| S_2 | S_3 | S_1 | S_2 | 1 | 0 | 0 | 1 | 0 |
| S_3 | S_3 | S_3 | S_3 | 0 | 0 | 0 | 0 | 0 |

จากตารางสถานะดังกล่าว เราจะแสดงการสร้างชาร์ตแวร์โดย 2 วิธี คือ วิธีดั้งเดิม (Classical Method) และวิธีวันshot (One Hot Method)

ขั้นตอนการออกแบบโดยวิธีดั้งเดิมมีดังนี้

- สร้างตารางสถานะที่มีஎ่வுகங்கள் ในการอ่านแต่ละสถานะ อินพุตจะให้สถานะใหม่ และเอาต์พุตเป็นค่าอะไร โดยทำการเข้ารหัสสถานะ อินพุต และเอาต์พุตด้วย
- ใช้พิกัดเก็บสถานะโดยที่มี N สถานะ ใช้ $\log_2 N$ บิต คือ ใช้ฟลิปฟล็อกจำนวน $\log_2 N$ ตัว
- สร้างวงจรคอมไบเนชัน (Combinational Circuit) ที่ให้สัญญาณเอาต์พุตและสถานะถัดไปตามตารางสถานะ

ในตัวอย่างข้างต้น สมมติให้มีการเข้ารหัสสถานะเป็นดังนี้

$$\begin{aligned} S_0 &= 00 \\ S_1 &= 01 \\ S_2 &= 10 \\ S_3 &= 11 \end{aligned}$$

สร้างตารางสถานะได้ดังตารางด้านล่าง

| อินพุต | | สถานะปัจจุบัน | | สถานะถัดไป | | เอาต์พุต | | | | | |
|--------|---------|----------------|----------------|-----------------------------|-----------------------------|----------|------|----------|--------|--------|--|
| XR > 0 | XR ≥ YR | D ₁ | D ₀ | D ₁ ⁺ | D ₀ ⁺ | Subtract | Swap | SelectXY | LoadXR | LoadYR | |
| 0 | d | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | |
| 0 | d | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | |
| 0 | d | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | |
| 0 | d | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | |
| 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | |
| 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | |
| 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | |

(៤៨)

| ອືນພຸດ | | ສານະ ປັຈຸບັນ | | ສານະ ຄັດໄປ | | ເອົາຕົກພຸດ | | | | | |
|--------|---------|-----------------|----------------|-----------------------------|-----------------------------|------------|------|----------|--------|--------|--|
| XR > 0 | XR ≥ YR | D ₁ | D ₀ | D ₁ ⁺ | D ₀ ⁺ | Subtract | Swap | SelectXY | LoadXR | LoadYR | |
| 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | |
| 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | |
| 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | |
| 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | |
| 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | |

จากตารางสามารถสร้างสมการบูลลีนได้

$$D_1^+ = \overline{(XR > 0)} + (XR \geq YR) + D_0$$

$$D_0^+ = D_1 D_0 + \overline{(XR \geq YR)} \overline{D_0} + \overline{(XR > 0)D_0}$$

$$\text{Substract} = D_1 \overline{D_0}$$

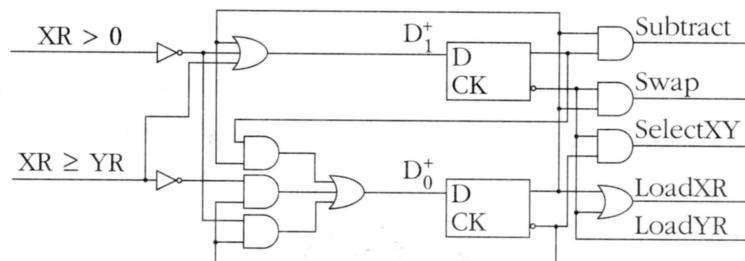
$$\text{Swap} = \overline{D}_1 D_0$$

$$\text{SelectXY} = \overline{D_1} \overline{D_0}$$

$$\text{loadXR} = \overline{D_0} + \overline{D_1}$$

loadYR = $\overline{D_1}$

วงจรแสดงดังรูปที่ 5.6



รูปที่ 5.6 วงจร GCD Processor โดยวิธีตั้งเดิม

ในวิธีดังเดิมจะพยายามลดจำนวนฟลิปฟล็อปเพื่อลดขนาดของหน่วยความจำที่จะให้ในวงจร โดยวิธีนี้จะค่อนข้างยากต่อการหาข้อผิดพลาด เนื่องจากมีการเข้ารหัสสถานะ ทำให้การตรวจสอบความถูกต้องว่าสถานะถัดไปคืออะไร เมื่อพิจารณาสถานะปัจจุบันและอินพุตยก เพราะต้องต้องถอดรหัสออกมาจากทรัพย์สินของอินพุตและสถานะ

วิธีที่ง่ายกว่าในการตรวจสอบก็คือ วิธีการวนรอบ ซึ่งแทนค่าสถานะโดยใช้ตำแหน่งบิตเดียวในการบอกสถานะ ถ้าเรามีสถานะทั้งหมด N สถานะ เราจะต้องใช้ n บิต และตำแหน่งของบิตที่มีค่าเป็น 1 จะสัมพันธ์กับสถานะ เช่น ถ้าเรามี 4 สถานะ อาจกำหนดให้รหัสเป็นดังนี้

$$S_0 = 0001$$

$$S_1 = 0010$$

$$S_2 = 0100$$

$$S_3 = 1000$$

การเข้ารหัสสถานะแบบนี้ง่าย แต่จำนวนบิตจะใช้มาก ทั้งนี้จำนวนบิตขึ้นอยู่กับจำนวนสถานะ สมการบูลีนของค่าสถานะถัดไปจะดูจาก D_i^+ ซึ่งดูจากสถานะอื่นๆ และอินพุตที่ทำให้เกิดสถานะ D_i นั้นคือ

$$D_i^+ = D_1(I_{11} + I_{12} + \dots + I_{1n_1}) + D_2(I_{21} + I_{22} + \dots + I_{2n_2}) + \dots$$

ตัวอย่างข้างต้น เปียนสมการ D_i^+ ได้ว่า

$$D_i^+ = (D_0 + D_2)(XR > 0) \overline{(XR \geq YR)}$$

กล่าวคือ จากตารางสถานะก่อนหน้า ถ้าพิจารณาทุกแrewของสถานะแล้วพบว่าแrewของ S_0 และ S_2 ซึ่งทำให้สถานะถัดไปเป็น S_1 และเมื่ออินพุต $(XR > 0)(XR \geq YR)$ เป็น 10

สำหรับการหาสมการบูลีนของເອາະົາດູ ພິຈາລະນາຈາກຄວບມັນຂອງສັງຄູານເອາະົາດູນໍາໆ ວ່າສາທະໄດບ້າງທີ່ໃຫ້ເອາະົາດູນໍາໆເປັນ 1 ກລວ່າວື້ອ ໃນຄວບມັນນີ້ແລ້ວໄດ້ບ້າງທີ່ເປັນ 1 ເບິ່ງທີ່ຂອງ Subtract ແກັດຈາກສາທະໄດ S_2 ດັ່ງນັ້ນ ໄດ້ສຳຜັກບູລຸລືນໍາຂອງສັງຄູານ Subtract ວ່າ

$$\text{Subtract} = D_2$$

$$\text{ແລະສຳຜັກຂອງສັງຄູານ LoadYR} = D_0 + D_1$$

ດັ່ງນັ້ນ ຈາກຕัวอย่างข้างต้น ຕ້າໃຊ້ວິธີວັນຍອດ ເຮັດວຽກໄດ້ສຳຜັກບູລຸລືນໍາດັ່ງນີ້

$$D_0^+ = 0$$

$$D_1^+ = D_0(XR > 0) \overline{(XR \geq YR)} + D_2(XR > 0) \overline{(XR \geq YR)}$$

$$D_2^+ = D_0(XR > 0)(XR \geq YR) + D_1 + D_2(XR > 0)(XR \geq YR)$$

$$D_3^+ = D_0 \overline{(XR > 0)} + D_2 \overline{(XR > 0)}$$

$$\text{Substract} = D_2$$

$$\text{Swap} = D_1$$

$$\text{SelectXY} = D_2$$

$$\text{loadXR} = D_0 + D_1 + D_2$$

$$\text{loadYR} = D_0 + D_1$$

ขั้นตอนการออกแบบโดยใช้วิธีวันยอตมีดังนี้

1. สร้างตารางสถานะที่มีจำนวน P แถว
2. เข้ารหัสของสถานะสำหรับ P สถานะโดยใช้ฟลิกฟลอกจำนวน P ตัว

$$D_1 D_2 \dots D_i D_{I+1} \dots D_p = 00 \dots 10 \dots 0$$

↑
ตำแหน่งที่ i

สำหรับสถานะที่ I ให้เป็น i มีค่าเป็น 1

3. คำนวนหาสมการของสถานะถัดไป D_i^+ และเอาต์พุต Z_i

$$D_i^+ = \sum_{i=1}^P D_1(I_{j1} + I_{j2} + \dots + I_{jn})$$

$$Z_k = D_{k1} + D_{k2} + \dots + D_{km}$$

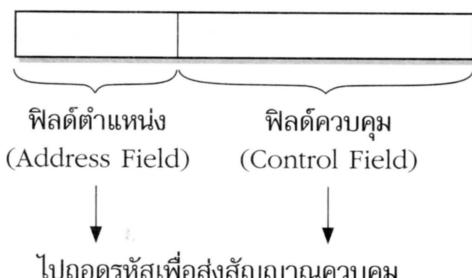
โดย I_{j1}, \dots, I_{jn} เป็นอินพุตต่างๆ ซึ่งจะทำให้เปลี่ยนสถานะจาก S_j ไป S_i จากนั้นนำสมการเหล่านี้ไปใช้ออกแบบวงจรคอมปไบเนชันต่อไป

■ 5.2 การควบคุมแบบไมโครโปรแกรม (Microprogrammed Control)

ในการออกแบบโดยวิธีไมโครโปรแกรมนั้น เสื่อนทางการให้สัญญาณควบคุมต่างๆ จะถูกบรรจุอยู่ในหน่วยความจำพิเศษที่เรียกว่าหน่วยความจำควบคุม (Control Memory) ในไมโครโปรแกรมจะมีคำสั่งระดับไมโคร (Microinstruction) ซึ่งเป็นคำสั่งบอกเสื่อนทางการให้สัญญาณในไมโครโปรแกรม

การใช้ไมโครโปรแกรมนั้นมีความยืดหยุ่นมากกว่าแบบฮาร์ดไดร์ เพราะการใช้ไมโครโปรแกรมไม่ได้เป็นฮาร์ดแวร์เฉพาะด้าน แต่ใช้การบรรจุโปรแกรมไว้ในรอม ถ้าต้องการเปลี่ยนหรือเพิ่มสัญญาณควบคุม ก็จะทำได้ง่ายโดยการเปลี่ยนโปรแกรมและบรรจุไปในรอมใหม่ ซึ่งต่างกับกรณีแบบฮาร์ดไดร์ ซึ่งจะต้องสร้างวงจรใหม่แทนที่วงจรเก่า แต่ในแห่งของความเร็วนั้น การใช้ฮาร์ดแวร์ในวิธีฮาร์ดไดร์ยอมรู้ว่าการทำงานของซอฟต์แวร์อย่างไมโครโปรแกรม เพราะตัวไมโครโปรแกรมการควบคุมสำหรับการทำงานตั้งแต่การอ่านคำสั่ง (Fetch) และตีโค้ด (Decode) ไปจนถึงเอ็กซ์คิวต์ เพื่อให้สัญญาณควบคุมออกไปตามที่โปรแกรมໄว้

ในส่วนของหน่วยควบคุมจะต้องใช้คำสั่งระดับไมโคร ซึ่งมีลักษณะคล้ายกับภาษาแอสแซมบลีใน 1 คำสั่งจะประกอบด้วยพิลด์ต่างๆ คือ พิลด์ควบคุม (Control Field) และพิลด์ตำแหน่ง (Address Field) ตัวพิลด์ควบคุมจะใช้กำหนดว่าจะส่งสัญญาณควบคุมใดออกไป และพิลด์ตำแหน่งจะระบุตำแหน่งถัดไปของคำสั่งระดับไมโครที่จะนำมาเอ็กซ์คิวต์



รูปที่ 5.7 พิลด์ตำแหน่งและพิลด์ควบคุม

โดยปกติแล้ว การทำงานของไมโครโปรแกรมจะเหมือนกับการอ็อกซีดิวตี้โปรแกรมปกติ โดยจะอ่านแต่ละคำสั่งระดับไมโครมาทีละคำสั่ง แล้วถอดรหัสไปสู่ภูมิความคุณที่กำหนดในคำสั่งนั้นๆ ส่วนพิล็อกต์ตำแหน่งจะเป็นตัวบอกตำแหน่งถัดไปที่จะอ่านคำสั่งระดับไมโคร

ในการใช้งานจริง รูปแบบของคำสั่งระดับไมโครอาจมีการเพิ่มเติมพิล็อกอื่นๆ เช่น เงื่อนไข หรือโอเพอเรนเดอร์ของคำสั่ง และอื่นๆ ดังนั้นใน 1 คำสั่งระดับไมโคร ขนาดของคำสั่งจะขึ้นอยู่กับ

- จำนวนตัวดำเนินการ (Operation) ที่ให้ทำงานได้พร้อมกัน ถ้าทำงานพร้อมกันได้หลายตัวดำเนินการอาจจะใส่ตัวดำเนินการเหล่านั้นไว้ในคำสั่งระดับไมโครเดียวกัน
- จำนวนพิล็อกควบคุม ซึ่งอาจมีการเข้ารหัสเพื่อประหยัดเนื้อที่
- วิธีการบอกพิล็อกตำแหน่ง ออาจจะบอกว่าแต่ละตำแหน่งมีขนาดจำนวนบิตเท่าใด เมื่อมีหลายคำสั่งระดับไมโครรวมกัน ก็จะกลายเป็นไมโครโปรแกรม ซึ่งขนาดของไมโครโปรแกรมจะขึ้นอยู่กับขนาดของแต่ละคำสั่งระดับไมโครนั้นเอง และขนาดของไมโครโปรแกรม จะบอกถึงขนาดของหน่วยความจำควบคุมหรือรวมที่จะใช้บรรจุไมโครโปรแกรมนั้นด้วย

รูปแบบคำสั่งระดับไมโครทั่วไป ประกอบด้วยพิล็อกดังรูปที่ 5.8

| | | |
|--|-----------------------------------|---------------------------------|
| | | |
| ตัวเลือกเงื่อนไข (Condition Select) | ตำแหน่งกระโดด (Branch Address) | พิล็อกควบคุม (Control Field) |

รูปที่ 5.8 รูปแบบของคำสั่งระดับไมโคร

โดยที่ ตัวเลือกเงื่อนไข (Condition Select) เป็นพิล็อกที่บอกเงื่อนไขที่ใช้ในการพิจารณา การให้สัญญาณเปลี่ยนแฉลเดรสของการทำงาน จะต้องมีการเข้ารหัสที่เหมาะสมตามลักษณะและ จำนวนเงื่อนไขที่ใช้ได้

ตำแหน่งกระโดด (Branch Address) เป็นพิล็อกแยกเดรสใหม่ของคำสั่งระดับไมโครที่ จะไปทำงานเมื่อเงื่อนไขที่กำหนดเป็นจริง

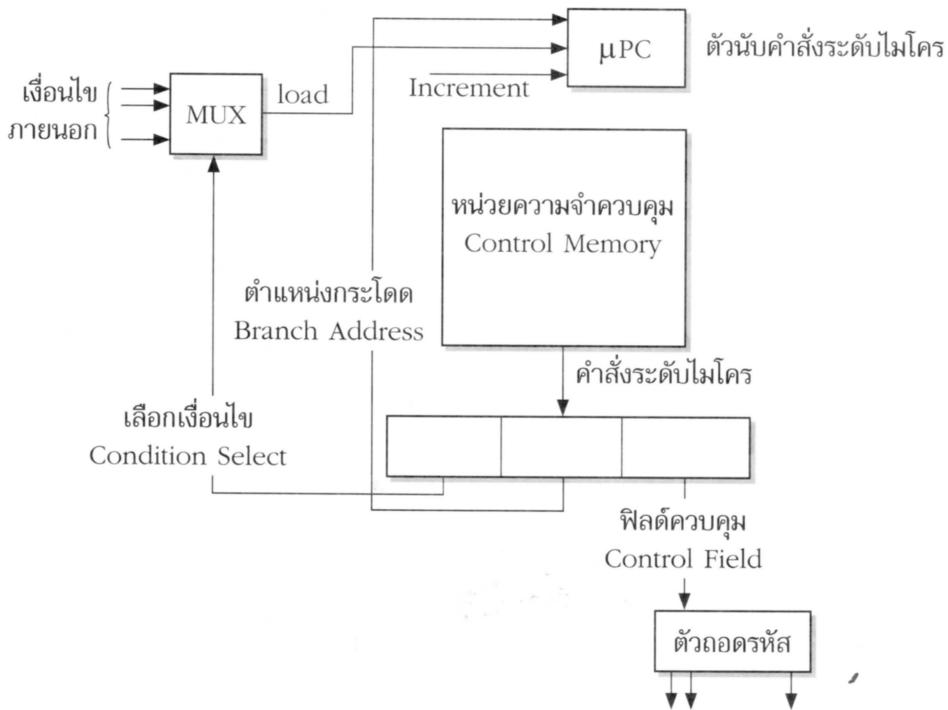
พิล็อกควบคุม (Control Field) ระบุสัญญาณที่ต้องการส่งออกไป

ตัวอย่างเช่น ถ้าโปรแกรมมีตัวแปรเงื่อนไข 2 ตัวคือ V_1 , V_2 จะจะใช้ 2 บิตในการสร้างพิล์ด์ตัวเลือกเงื่อนไขเรียกว่า S_0 , S_1 โดยให้ความหมายของแต่ละบิตเป็นดังนี้

| S_0 | S_1 | ความหมาย |
|-------|-------|-----------------------|
| 0 | 0 | ไม่กระโดด |
| 0 | 1 | กระโดดเมื่อ $V_1 = 1$ |
| 1 | 0 | กระโดดเมื่อ $V_2 = 1$ |
| 1 | 1 | กระโดดไม่มีเงื่อนไข |

โดยทั่วไปแล้ว เราต้องกำหนดรหัสไว้สำหรับในกรณีที่เป็นการเปลี่ยนตำแหน่งแบบไม่มีเงื่อนไข (Unconditional Branch) และเมื่อไม่มีการกระโดด (No Branch) เสมอ ส่วนที่เหลือให้เป็นรหัสที่กำหนดว่าจะกระโดดอย่างไร โดยพิจารณาจากตัวแปรเงื่อนไขที่มี เช่น $V_1 = 1$ หรือ $V_2 = 1$

ลักษณะของหน่วยควบคุมแสดงดังรูปที่ 5.9



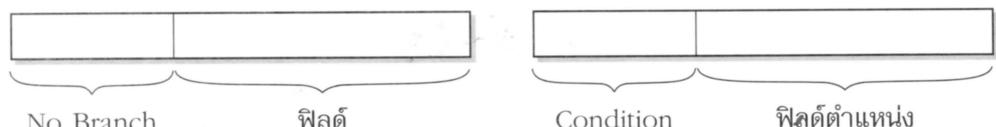
รูปที่ 5.9 โครงสร้างของหน่วยความคุมแบบใช้ไมโครโปรแกรม

ในหน่วยความจำควบคุมจะบรรจุคำสั่งระดับไมโครเอาไว้ โดยแต่ละคำสั่งระดับไมโครประกอบด้วย 3 พิล็อต ดังแสดงในรูปที่ 5.9 โดยเริ่มต้นค่า μPC จะเพิ่มขึ้นทีละหนึ่งเพื่ออ่านคำสั่งถัดไป แต่ถ้าเงื่อนไขที่กำหนดใน Condition Select เป็นจริง ค่า μPC จะถูกเปลี่ยนเป็นตำแหน่งที่กำหนดในพิล็อต Branch Address ส่วนพิล็อตควบคุมจะระบุว่าคำสั่งจะให้สัญญาณควบคุมอะไรบ้าง ณ เวลาใดนั้น นอกจากนี้ ในพิล็อตควบคุมอาจจะมีการเข้ารหัสเพื่อประยัดเดือที่ แต่จะต้องถอดรหัสด้วยเมื่อมีการนำมาใช้

ถ้าพิล็อตควบคุมใช้รูปแบบลักษณะแนวอน (Horizontal) จะไม่มีการเข้ารหัสเลย อาจจะทำให้คำสั่งระดับไมโครหนึ่งฯ ยิ่ง แต่ว่าจะเพิ่มความสามารถในการส่งสัญญาณควบคุมออกไปพร้อมกันได้หลายสัญญาณ แต่ถ้าใช้รูปแบบแนวตั้ง (Vertical) จะใช้การเข้ารหัสเพื่อประยัดพื้นที่แต่ต้องมีการถอดรหัส ซึ่งทำให้ลดความสามารถในการส่งสัญญาณควบคุมออกไปพร้อมๆ กันได้

นอกจากนี้แล้ว การเข้ารหัสพิล็อตควบคุมมีได้หลายรูปแบบ รูปแบบหนึ่งเป็นการเข้ารหัสสัญญาณเขิงหน้าที่ จะเป็นการจำแนกสัญญาณควบคุมออกเป็นกลุ่มๆ ตามหน้าที่ของสัญญาณ เช่น กลุ่มของสัญญาณทำหน้าที่สำหรับการบวก กลุ่มของสัญญาณทำหน้าที่สำหรับการลบ เป็นต้น และก็ทำการเข้ารหัสสัญญาณสำหรับแต่ละกลุ่ม การเข้ารหัสสัญญาณจะต้องพิจารณาว่าถ้าสัญญาณกลุ่มใดไม่มีโอกาสเกิดขึ้นพร้อมกัน จะมีการใช้บิตร่วมกันได้ ทั้งนี้เพื่อประยัดจำนวนบิตรวมทั้งหมด วิธีการที่ใช้ในการเข้ารหัสสัญญาณในพิล็อตควบคุมนี้ก็เป็นปัญหาที่สำคัญในการลดขนาดของคำสั่งระดับไมโคร รวมทั้งลดขนาดของการใช้หน่วยความจำควบคุมด้วย

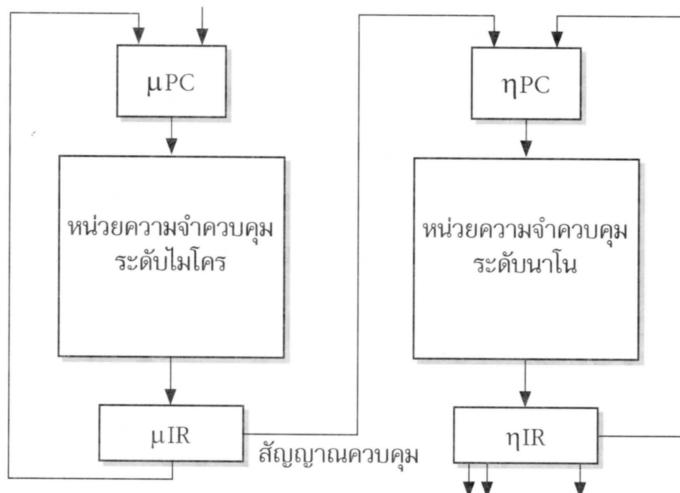
นอกจากนี้ อาจจะสังเกตว่าพิล็อตตำแหน่งจะถูกใช้พร้อมกับพิล็อตควบคุมด้วยหรือไม่ คำสั่งระดับไมโครที่มีการใช้ค่าในพิล็อตตำแหน่งจะเป็นคำสั่งที่เกี่ยวกับการกระโดด ซึ่งเป็นการเปลี่ยนตำแหน่งของคำสั่งที่ทำงานต่อไป และคำสั่งนั้นมักจะไม่มีการให้สัญญาณควบคุมออกมาก ดังนั้น เราอาจจะทำให้คำสั่งระดับไมโครมี 2 รูปแบบ ดังแสดงในรูปที่ 5.10 โดยแบบแรกใช้กรณีไม่มีการกระโดด ให้ระบุแต่ละสัญญาณควบคุม แบบที่สองใช้เมื่อการกระโดด โดยระบุเงื่อนไขการกระโดดในพิล็อต Condition และพิล็อตตำแหน่งระบุตำแหน่งที่จะกระโดดไปทำงาน



รูปที่ 5.10 รูปแบบของคำสั่งระดับไมโครที่มีการใช้พิล็อตตำแหน่ง และไม่มีการใช้พิล็อตตำแหน่ง

การที่จะเลือกใช้รูปแบบใดนั้นขึ้นอยู่กับคำสั่ง ดังนั้น วิธีการนี้สามารถทำให้คำสั่งระดับไมโคร สั้นลงได้โดยเฉลี่ย

การใช้โปรแกรมแบบนาโน (Nanoprogramming) ก็เป็นอีกวิธีในการช่วยประหยัดพื้นที่ของหน่วยความจำควบคุม โดยมีการเพิ่มระดับในการเข้าถึงคำสั่งแต่ละคำสั่งขึ้นมา ดังแสดงในรูปที่ 5.11

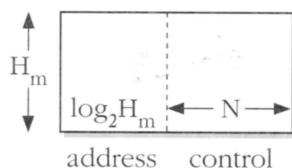


รูปที่ 5.11 โครงสร้างการทำงานของโปรแกรมระดับนาโน

คำสั่งที่บรรจุอยู่ในหน่วยความจำระดับไมโคร จะอ้างไปถึงคำสั่งในหน่วยความจำระดับนาโน คำสั่งในหน่วยความจำนี้เรียกว่าคำสั่งระดับนาโน (Nanoinstruction) ซึ่งจะบอกพิลด์ควบคุม ใน การอ้างถึง 2 ระดับบนนี้จะประหยัดเนื้อที่ของหน่วยความจำควบคุมทั้งหมด ให้ S_1 เป็นพื้นที่ของหน่วยความจำควบคุมที่ใช้ เมื่อใช้แบบ 1 ระดับ ขนาดของ S_1 คือ

$$H_m * (\log_2 H_m + N) \text{ บิต}$$

โดยที่ H_m คือ จำนวนแควรของหน่วยความจำควบคุม และพิลด์ตัวแหน่งมีขนาด $\log_2 H_m$ และพิลด์ควบคุมมีขนาด N บิต ดังแสดงในรูปที่ 5.12



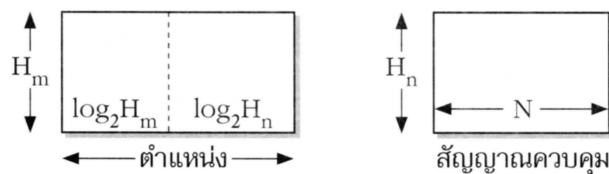
รูปที่ 5.12 ลักษณะหน่วยความจำควบคุมแบบ 1 ระดับ

ถ้าใช้ 2 ระดับ ให้ S_2 เป็นขนาดของหน่วยความจำควบคุมทั้งหมด ขนาดพื้นที่ของ S_2 คือ

$$S_2 = H_m * (\log_2 H_m + \log_2 H_n) + NH_n \quad \text{บิต}$$

เมื่อใช้หน่วยความจำควบคุมระดับนานาในขนาด N และ $\log_2 H_n$ บิต ในการเข้าถึงแต่ละแถว และหน่วยความจำควบคุมระดับนานาแต่ละแถวจะเก็บฟิล์ดควบคุมขนาด N บิต

ในการออกแบบหน่วยความจำ จะใช้ 2 ระดับหรือ 1 ระดับ ขึ้นอยู่กับจำนวนคำสั่งระดับไมโครที่มีอยู่และจำนวนบิตของฟิล์ดควบคุมที่จะใช้ ดังนั้น จะต้องมีการคำนวณว่า $S_2 < S_1$ เมื่อค่าจำนวนฟิล์ดควบคุม N ควรเป็นเท่าไรเมื่อกำหนด H_m และ H_n มาให้ เพื่อคุ้มครองใช้กี่ระดับ



รูปที่ 5.13 ลักษณะหน่วยความจำควบคุมแบบ 2 ระดับ

ตัวอย่างการออกแบบไมโครโปรแกรมและหน่วยความจำ สำหรับการออกแบบโปรเซสเซอร์ GCD เราเขียนไมโครโปรแกรมได้ดังนี้

| ตำแหน่ง | ตัวดำเนินการ | สัญญาณควบคุม |
|---------|-----------------------|---------------------------|
| BEGIN | XR := X, YR := Y | Select XY, loadXR, loadYR |
| AGAIN | If (XR ≤ 0) goto OUT | |
| START | If (XR > YR) goto SUB | |
| SWAP | SWAP (XR, YR) | SWAP, loadXR, loadYR |
| SUB | XR := XR - YR | subtract, loadXR |
| LOOP | goto AGAIN | |
| OUT | Z := YR | เอาต์พุต Z |
| END | | |

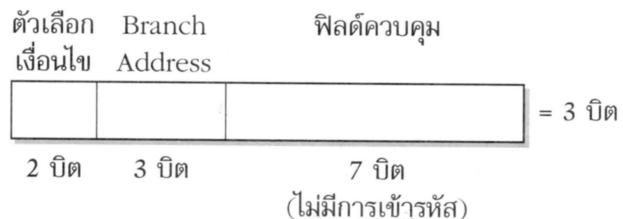
ให้เงื่อนไข Condition Select มีรหัสเป็นดังนี้

| | |
|-------------------------|----|
| ไม่กระโดด | 00 |
| กระโดดเมื่อ $XR \leq 0$ | 01 |
| กระโดดเมื่อ $XR > YR$ | 10 |
| กระโดดไม่มีเงื่อนไข | 11 |

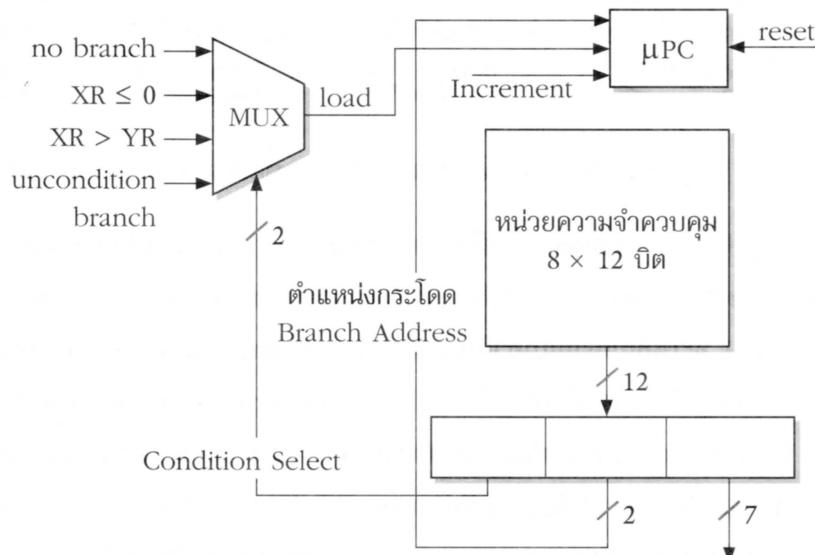
ส่วนฟิลด์ตำแหน่งใช้ขนาด 3 บิต เพราะมี 8 ตำแหน่ง เราเขียนโปรแกรมในรูปแบบตารางเลขฐาน 2 ได้ดังตารางด้านล่าง

| ตำแหน่ง ใน CM | เงื่อนไข | Branch Address | ฟิลด์ควบคุม | | | | | | |
|------------------|----------|-------------------|-------------|----------|--------|--------|------|----------|---------|
| | | | END | SelectXY | LoadXR | LoadYR | Swap | Subtract | OutputZ |
| 000 | 00 | 000 | 0 | 1 | 1 | 1 | 0 | 0 | 0 |
| 001 | 01 | 110 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 010 | 10 | 100 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 011 | 00 | 000 | 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| 100 | 00 | 000 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 101 | 11 | 001 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 110 | 00 | 000 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 111 | 11 | 111 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |

ใน 1 คำสั่งจะต้องมีโครงสร้างจำนวนบิตดังนี้



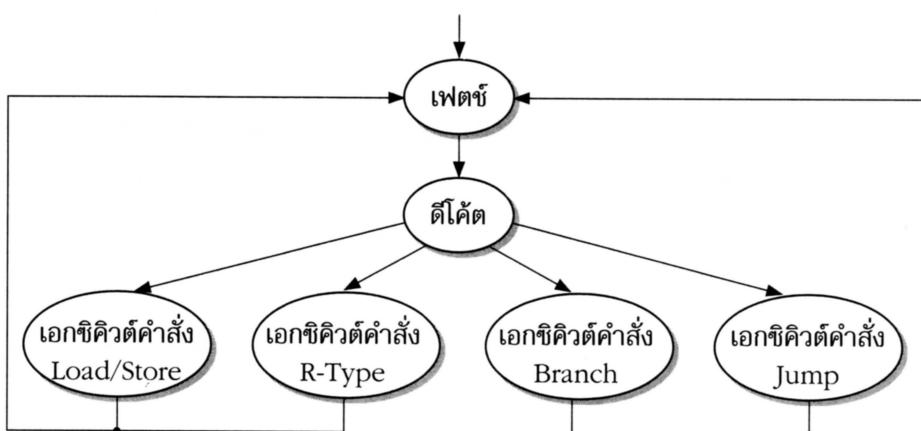
โครงสร้างของหน่วยความจำควบคุมจะแสดงได้ดังรูปที่ 5.14 ตัว MUX มีอินพุต 4 ตัวไว้เลือกตำแหน่งที่จะไปทำงานตามเงื่อนไข ตำแหน่งที่จะไปทำงานจะถูกอ่านเข้าไปในรีจิสเตอร์ PC ของไมโครโปรแกรม สัญญาณ load จะควบคุมการเขียนค่าในรีจิสเตอร์ PC สัญญาณ Increment จะเป็นการสั่งให้เพิ่มค่าในรีจิสเตอร์ PC อัตโนมัติ และอินพุต Branch Address กำหนดตำแหน่งที่กระโดดไปกรณีของเงื่อนไขเป็นจริงหรือเป็นการกระโดดแบบไม่มีเงื่อนไข



รูปที่ 5.14 การออกแบบโปรแกรม GCD โดยใช้ไมโครโปรแกรมออกแบบ

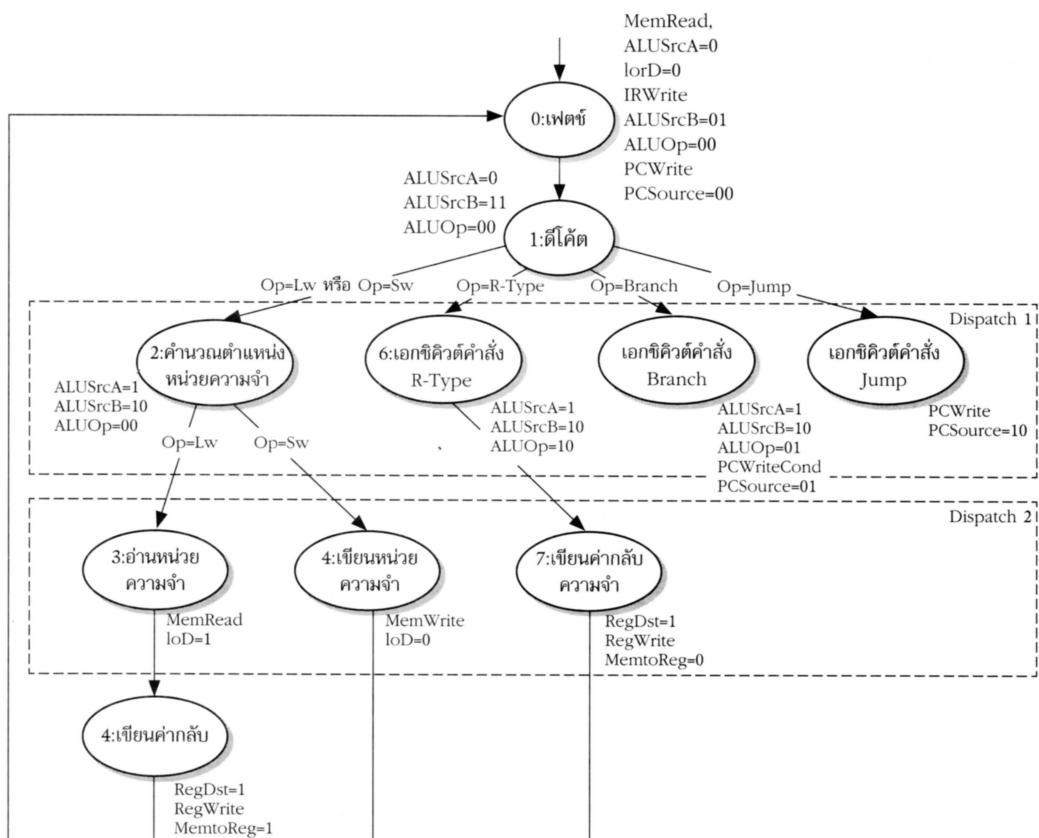
5.3 การออกแบบสัญญาณควบคุมของ MIPS

การอิมเพลเม้นต์สัญญาณควบคุมของ MIPS ซึ่งจะใช้เครื่องยนต์สถานะจำกัดคล้ายกับที่กล่าวถึงไปแล้วข้างต้น ในที่นี้จะแบ่งสถานะตามขั้นตอนต่างๆ และจะทำงานตามประเภทของคำสั่งดังแสดงในรูปที่ 5.15 เนื่องจากในการแบ่งขั้นตอนการทำงานใน MIPS ในบทที่แล้วจะเห็นว่าทุกคำสั่งทำงานเหมือนกันในขั้นที่ 1-2 เมื่อถึงขั้นการเล็กซิคิวต์จะทำงานแยกตามประเภทของคำสั่ง ในแต่ละขั้นจะมองเป็นการทำงานใน 1 สถานะของเครื่องยนต์สถานะจำกัด



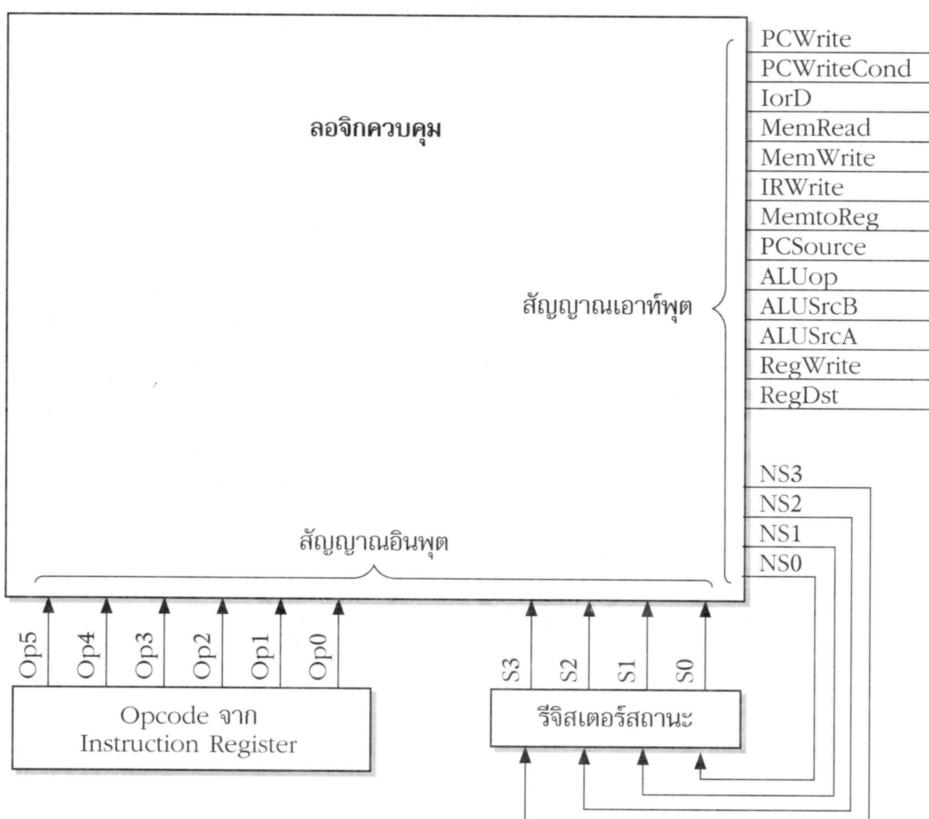
รูปที่ 5.15 เครื่องยนต์สถานะจำจัดแสดงขั้นตอนการทำงานของ MIPS

ในรูปที่ 5.16 แสดงการให้สัญญาณควบคุมในรายละเอียดสำหรับแต่ละประเภทของคำสั่ง หลังจากถอดรหัสคำสั่ง โดยแบ่งเป็น 2 ขั้นตอนย่อยๆ เรียกว่า Dispatch 1 และ Dispatch 2 โดย Dispatch 1 จะให้สัญญาณเกี่ยวกับคำสั่ง ALU, Branch, Jump และการคำนวนตำแหน่ง อีกนัยหนึ่ง ขั้นตอนนี้จะเกี่ยวกับการใช้ ALU และ Dispatch 2 เป็นขั้นตอนเกี่ยวกับสัญญาณสำหรับคำสั่ง Load/Store เพื่ออ่าน-เขียนหน่วยความจำ และจากนั้นจะไปสถานะเกี่ยวกับการ Write Back สำหรับคำสั่ง R-Type เครื่องยนต์นี้เป็นแบบมอร์ (Moore Machine) ซึ่งเป็นตัวให้ค่าเอาต์พุต ตามสถานะที่อยู่ และสัญญาณความหมายของสัญญาณได้อธิบายไว้ในบทที่ 4 แล้ว สังเกตว่า คำสั่งแบบ Branch และ Jump ทำงานเสร็จลิ้นใน 3 สถานะ คำสั่งแบบ R-Type ทำงานเสร็จลิ้นภายใน 4 สถานะ และคำสั่งแบบ Load/Store ทำงานเสร็จลิ้นใน 5 สถานะ

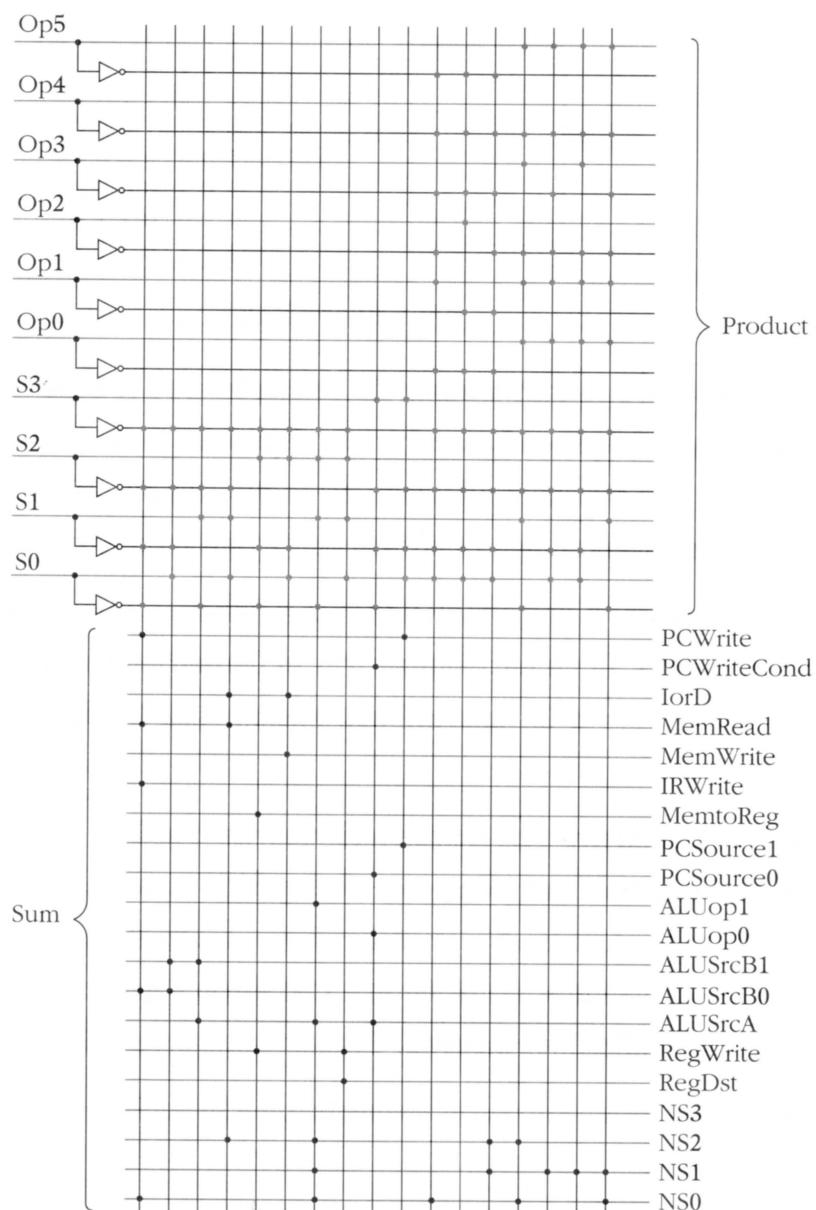


รูปที่ 5.16 เครื่องยนต์สถานะจำกัดและการให้ค่าสัญญาณควบคุม

รูปที่ 5.17 แสดงไดอะแกรมของตัวควบคุมสัญญาณของ MIPS โดยจะรับอินพุตจากรีจิสเตอร์สถานะและฟิลด์ Opcode จาก Instruction Register และผลตรหัสสัญญาณเอาต์พุตในส่วนของลوجิกควบคุมภายในอาจจะใช้ PLA (Programmable Logic Array) ดังแสดงในรูปที่ 5.18 ซึ่งเป็นการอิมเพลเม้นต์ด้วย Sum of Product โดยจุดด้านบนในแต่ละคอลัมน์จะหมายถึง Product ของแต่ละอินพุตตามจุด และด้านล่างจะหมายถึง Sum เช่น $PCWrite = \overline{S_3} \cdot \overline{S_1} \cdot \overline{S_1} \cdot \overline{S_0} + S_3 \cdot \overline{S_2} \cdot \overline{S_1} \cdot S_0$

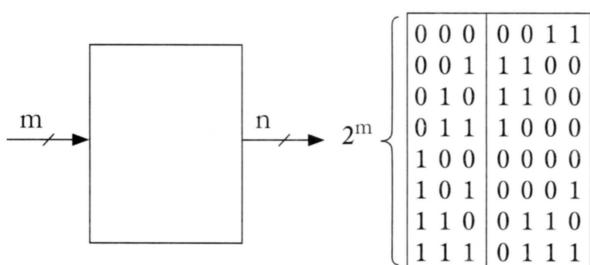


รูปที่ 5.17 ตัวอย่างไดอะแกรมของตัวควบคุมของ MIPS



รูปที่ 5.18 ตัวอย่าง PLA ของตัวควบคุมของ MIPS

นอกจากนี้ยังสามารถใช้รอมในการอิมพลีเม็นต์ได้อีกด้วย ในการใช้รอมหมายถึงการใช้ Table Lookup แบบการใช้ตารางค่าความจริง โดยถ้าแอดเดรสมีขนาด m บิต จะสามารถมีจำนวนแถว (Entry) ได้เท่ากับ 2^m และ ตั้งแสดงในรูปที่ 5.19 ตัวแปร m หมายถึง จำนวนแถวที่จะได้ (2^m) และ n จะหมายถึงจำนวนเอ้าต์พุต



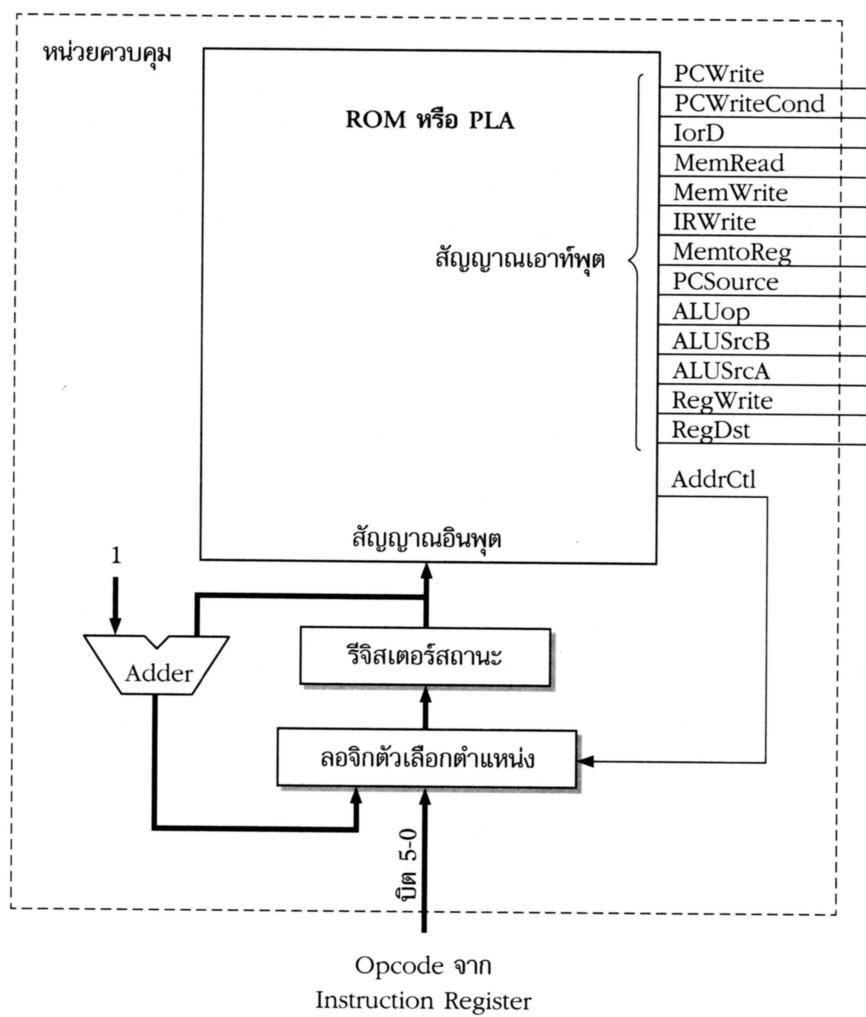
รูปที่ 5.19 การใช้ ROM

ตัวอย่างในรูปที่ 5.18 จำนวนอินพุตเท่ากับ 10 บิต (Opcode ใช้ 6 บิต สถานะใช้ 4 บิต) ดังนั้น จะได้สัญญาณอินพุต 10 เส้น และจำนวนเอาต์พุตเท่ากับ 20 บิต (สัญญาณควบคุมใช้ 16 บิต, สถานะใช้ 4 บิต) ขนาดของ ROM จึงเท่ากับ $2^{10} \times 20 = 20k$ บิต ซึ่งเป็นขนาดที่ใหญ่ ความจริงแล้วมีเอาต์พุตบางตัวซ้ำกันอยู่หลายคราวด้วย ซึ่งสามารถทำการลดขนาดลงໄປได้อีก

ถ้าเราจัดแบ่งเป็น 2 ตาราง โดยใช้บิตสถานะ 4 บิตในการกำหนดเอาต์พุต 16 เส้น จะได้ขนาด ROM เท่ากับ $2^4 \times 16$ บิต และใช้อินพุตจำนวน 10 บิตเพื่อกำหนดสถานะตัดไปจำนวน 4 บิต ทำให้ได้ ROM ขนาด $2^{10} \times 4$ บิต และรวมเป็นขนาดทั้งหมด 4.3k บิต ซึ่งลดໄປมากเมื่อเทียบกับของเดิมที่ใช้ 20k บิต

เปรียบเทียบกับการใช้ PLA ซึ่งสามารถใช้เทอมต่างๆ ร่วมกันได้ และสามารถใช้แต่แผลที่จำเป็น รวมทั้งยังสามารถใช้เทอม Don't Care เพื่อลดรูปได้อีก ขนาดของ PLA จะเท่ากับ ($\text{จำนวนอินพุต} \times \text{จำนวน Product-term}$) + ($\text{จำนวนเอาต์พุต} \times \text{จำนวน Product-term}$) ในตัวอย่างนี้จะเท่ากับ $(10 \times 17) + (20 \times 17)$ เท่ากับ PLA Cell จำนวน = 460 Cell โดยทั่วไปแล้ว ขนาดของ PLA ควรจะใกล้เคียงกับขนาดของ ROM ซึ่ง ROM อาจจะใหญ่กว่าเล็กน้อย

ในอีกรูปแบบของไมโครโปรแกรม ดังแสดงในรูปที่ 5.20 ซึ่งเป็นการนำ PLA หรือ ROM มาใช้กำหนดสัญญาณออก



รูปที่ 5.20 การใช้ไมโครโปรแกรม

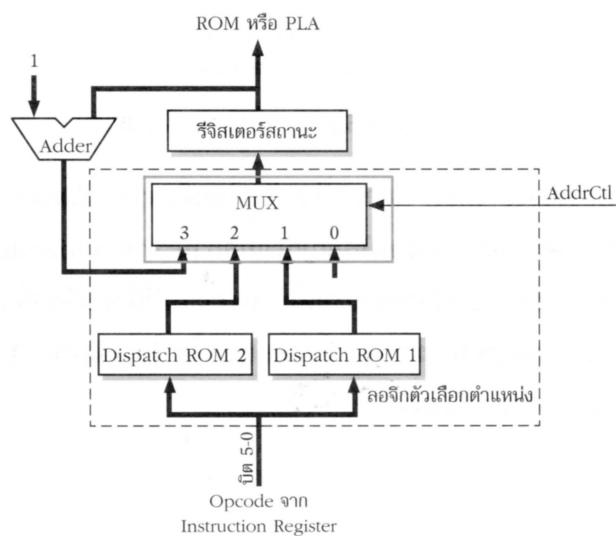
อินพุตของวงจรจะ ได้แก่ สถานะปัจจุบัน โดยดึงมาจากบิตของ Opcode ในคำสั่ง วงจร ลอกจิกเลือกตำแหน่งจะเป็นการถอดรหัสัญญาณสถานะจาก Opcode ที่เข้ามา และค่าสถานะ ปัจจุบันจะถูกบวก 1 เพื่อปรับเป็นสถานะถัดไป เพราะในที่นี้สมมติว่าขั้นตอนถัดไปคือสถานะถัดไป ซึ่งเป็นค่าที่ต่อเนื่อง หน่วยควบคุมอาจจะถูกออกแบบโดยใช้ ROM หรือ PLA โดยใช้อินพุตจากสถานะ ปัจจุบันจากรีจิสเตอร์สถานะเข้ามา

ตารางรวมจะถูกแบ่งออกเป็น 2 ส่วนดังแสดงในรูปที่ 5.21

| Dispatch ROM 1 | | | Dispatch ROM 2 | | |
|----------------|-------------|-------------|----------------|-------------|--|
| ค่า Opcode | ชื่อ Opcode | ค่า (Value) | ค่า Opcode | ชื่อ Opcode | |
| 000000 | R-format | 0110 | 100011 | lw | |
| 000010 | jmp | 1001 | 101011 | sw | |
| 000100 | beq | 1000 | | | |
| 100011 | lw | 0010 | | | |
| 101011 | sw | 0010 | | | |

รูปที่ 5.21 ตาราง ROM 1 และ ROM 2

ใน Dispatch ROM 1 ใช้สำหรับขั้นของการເອັກຫື້ຄວົງ สำหรับคำสั่ง R-Type, Jump, Branch, Lw, Sw โดยเป็นส่วนการคำนวนที่ใช้ ALU และใน Dispatch ROM 2 จะใช้สำหรับขั้นตอนที่ต้องมีการใช้หน่วยความจำข้อมูลสำหรับคำสั่ง Lw/Sw ในรูปที่ 5.22 แสดงการอิมພลีเมนต์โดยใช้ Opcode จะถูกแยกสัญญาณและจะถูกใช้ในการค้นหาจาก Dispatch ROM 1, ROM 2 และการเปลี่ยนเป็นสถานะถัดไปได้จากตัวควบคุม MUX โดยอาจจะได้จากค่าที่อ่านจาก ROM ทั้ง 2 ขา (1, 2 ของ MUX) หรือตำแหน่งถัดไปที่ได้จาก Adder (ขา 3) หรือค่า 0 มาจาก (ขา 4) โดยใช้สัญญาณ AddrCtl เป็นตัวกำหนด



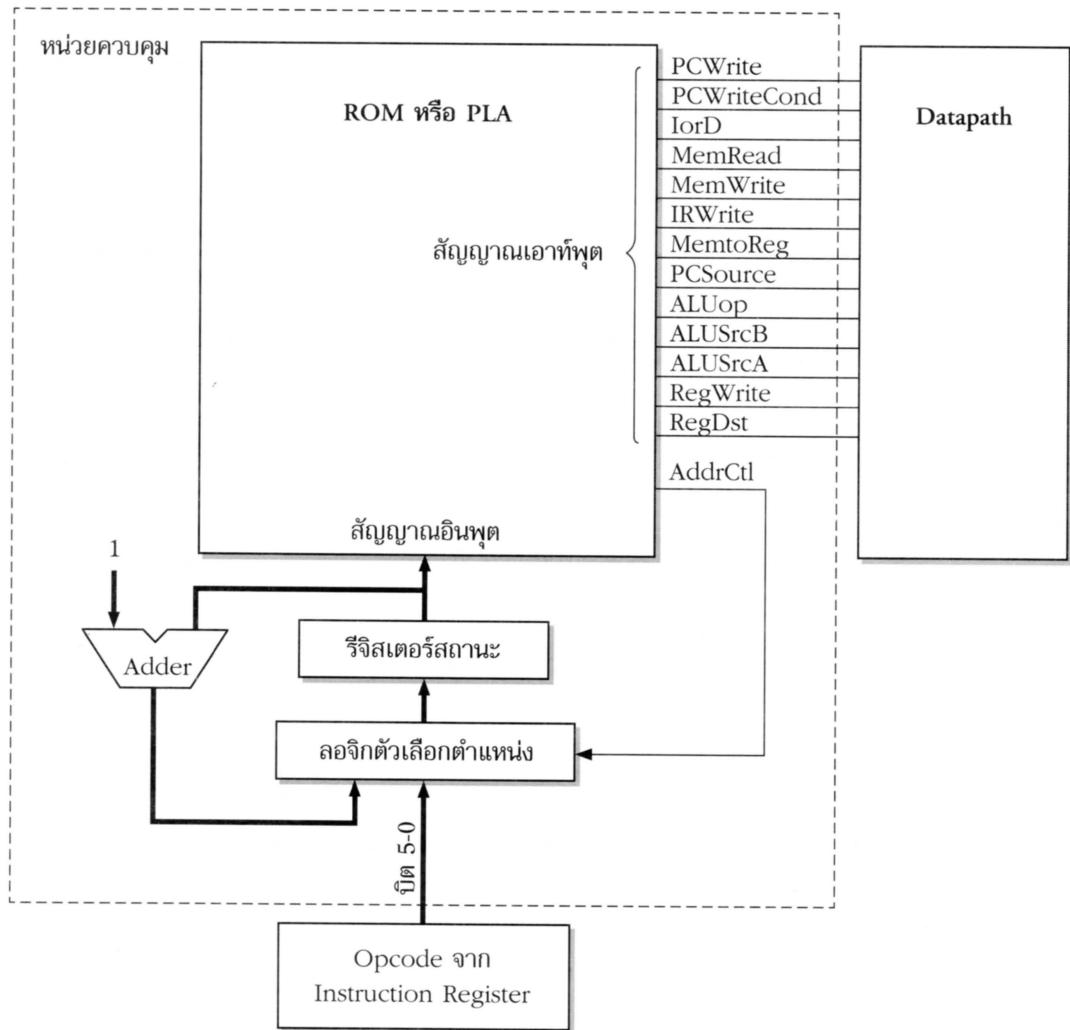
รูปที่ 5.22 การควบคุมของไมโครโปรแกรม

ตารางด้านล่างแสดงความหมายของแต่ละสถานะและค่าของสัญญาณ AddrCtl ของแต่ละสถานะ เพื่อเลือกอินพุตของ MUX โดยแต่ละแกวzerbai ถึงแต่ละสถานะและการใช้สัญญาณ AddrCtl ตามสถานะ

| สถานะ | สิ่งที่ต้องทำ | ค่าของ AddrCtl |
|-------|----------------------|----------------|
| 0 | เพิ่มสถานะไป 1 ลำดับ | 3 |
| 1 | ใช้ Dispatch ROM 1 | 1 |
| 2 | ใช้ Dispatch ROM 2 | 2 |
| 3 | เพิ่มสถานะไป 1 ลำดับ | 3 |
| 4 | ไปสถานะ 0 | 0 |
| 5 | ไปสถานะ 0 | 0 |
| 6 | เพิ่มสถานะไป 1 ลำดับ | 3 |
| 7 | เพิ่มสถานะไป 1 ลำดับ | 0 |
| 8 | เพิ่มสถานะไป 1 ลำดับ | 0 |
| 9 | เพิ่มสถานะไป 1 ลำดับ | 0 |

รูปที่ 5.23 การให้สัญญาณ AddrCtl

ภาพรวมของการควบคุมด้วยวิธีนี้แสดงดังรูปที่ 5.24



รูปที่ 5.24 การควบคุมของไมโครโปรแกรมทั้งหมด

ลักษณะพฤติกรรมของไมโครโปรแกรมแสดงได้ดังรูปที่ 5.25 ในแต่ละคลื่นจะเป็นคำอธิบายการให้สัญญาณควบคุมแต่ละตัว หันนีชื่อนี้อยู่กับขั้นตอนต่างๆ เช่น ในช่อง Label ระบุ Label และขั้นตอนต่างๆ ของสัญญาณควบคุม ALU ระบุว่าในชั้นนี้ให้สัญญาณควบคุม ALU อย่างไร ก็ล่าวคือ ให้ ALU ทำตัวดำเนินการอะไร ของໂອເປົວແຮນດໍາບະນະບຸວ່າຂາບ້ານຂອງ ALU ต้องປັບປຸງໃນພຸດຕະໂລກ ของໂອເປົວແຮນດໍາບະນະບຸວ່າຂາບ້ານຂອງ ALU ต้องປັບປຸງໃນພຸດຕະໂລກ ໃດ ช่องสัญญาณควบคุมມີເຈົ້າສເຕືອນບຸວ່າຕ້ອງทำໄວ້ກັບເຈົ້າສເຕືອນຕົວໄດ້ ชອງໜ່າຍຄວາມຈຳຮະບຸວ່າຄຳສັ່ນນີ້ຕ້ອງທ່າຍ່າງໄວ້ກັບໜ່າຍຄວາມຈຳບ້າງ ແລະ ຂອງ Sequencing ບຸວ່າລຳດັບຕ່ອໄປຈະໄປທຳນານທີ່ໂດດສ່ວນໄດ້

ตัวอย่างเช่น ที่ Label Fetch จะต้องทำการเพิ่มค่าตำแหน่งโดยใช้ค่าโอเปอเรนด์ข้าแรกจาก PC และหาที่สองเป็นค่าคงที่ 4 ให้สัญญาณอ่านหน่วยความจำตำแหน่งที่ระบุในรีจิสเตอร์ PC ซึ่งอ่าน PC ใช้ ALU ในกระบวนการ แล้วเขียนไปยังรีจิสเตอร์ PC ใหม่ ระบุโดยสัญญาณในช่อง PCWrite จากนั้นทำการดับต่อไป (Seq.) บรรทัดต่อไปเป็นการติดต่อให้สัญญาณอ่านรีจิสเตอร์ แล้วไปทำขั้น Dispatch 1 ซึ่งเป็นการจัดการคำสั่ง LW/ SW, R-Type, Branch, Jump ใน Label Mem1, Rformat1, BEQ1, JUMP1 ซึ่งเป็นการใช้ ALU กระทำกับโอเปอเรนด์ขานและข้าล่างต่างๆ กัน จากนั้นสำหรับข้า Mem1 ซึ่งเป็น LW/SW จะไปทำ Dispatch 2 ซึ่งจะแยกเป็น LW1 หรือ SW1 และแต่คำสั่ง สำหรับ LW1 จะไปทำการอ่านหน่วยความจำเก็บไว้ใน MDR จากนั้นจะทำการดับต่อไปโดยจะเขียนไปค่าที่อ่านได้จาก MDR ไปยังรีจิสเตอร์ และกลับไปทำ Fetch ใหม่ และสำหรับ SW1 จะไปทำการเขียนไปยังหน่วยความจำ และกลับไป Fetch ใหม่ สำหรับคำสั่ง Rformat1 จะทำการดับต่อไปเพื่อเขียนค่าในรีจิสเตอร์ และกลับไป Fetch ใหม่ ส่วน Branch1 และ Jump1 ก็จะกลับไป Fetch ใหม่เลย

| Label | สัญญาณ ควบคุม ALU | โอเปอ- แรนด์ขาน | โอเปอ- แรนด์ขาน ล่าง | สัญญาณ ควบคุม รีจิสเตอร์ | หน่วย ความจำ | สัญญาณ ควบคุม PCWrite | คำสั่ง ต่อไป |
|----------|-------------------------|--------------------|----------------------------|--------------------------------|-----------------|-----------------------------|-----------------|
| Fetch | Add | PC | 4 | | ReadPC | ALU | Seq |
| | Add | PC | ExtShft | Read | | | Dispatch 1 |
| Mem1 | Add | A | Extend | | | | Dispatch 2 |
| LW2 | | | | | Read ALU | | Seq |
| | | | | Write MDR | | | Fetch |
| SW2 | | | | | | Write ALU | Fetch |
| Rformat1 | Func code | A | B | | | | Seq |
| | | | | Write ALU | | | Fetch |
| BEQ1 | Subt | A | B | | | ALUOut-cond | Fetch |
| JUMP1 | | | | | | Jump address | Fetch |

รูปที่ 5.25 ตารางคำสั่งในไมโครโปรแกรม

■ 5.5 สรุป

การออกแบบหน่วยควบคุมเป็นส่วนสำคัญในการกำหนดสัญญาณ เพื่อไปควบคุมหน่วยอื่นๆ ในเครื่องคอมพิวเตอร์ จากโครงสร้าง Data Path ของบทที่แล้วจะพบว่าต้องการสัญญาณควบคุมมากมาย ดังนั้น ในบทนี้ได้กล่าวถึงการออกแบบในส่วนของหน่วยควบคุมซึ่งเป็นส่วนที่ต้องให้สัญญาณเหล่านี้ตามขั้นตอนต่างๆ ที่เหมาะสม หน่วยควบคุมนี้อยู่ภายในชีพียุ การส่งสัญญาณที่เหมาะสมไปยังหน่วยต่างๆ เพื่อการให้ผลของข้อมูลที่ถูกต้อง ทำให้ชีพียุสามารถทำการคำนวณตามคำสั่งต่างๆ ได้ วิธีการออกแบบตัวควบคุมมี 2 วิธีหลัก ได้แก่ วิธีฮาร์ดไวร์ (Hardwire) และวิธีไมโครโปรแกรม (Microprogrammed) โดยวิธีฮาร์ดไวร์จะสร้างฮาร์ดแวร์เฉพาะอย่างเพื่อทำหน้าที่นี้ และจะไม่มีความยืดหยุ่นมากนัก ทำให้การเพิ่มคำสั่งในชีพียุเป็นไปได้ยาก แต่จะทำงานได้เร็ว ในบทนี้ได้กล่าวถึง 2 วิธีในการออกแบบ คือ วิธีดั้งเดิม (Classical Method) ซึ่งเป็นการออกแบบฮาร์ดแวร์โดยใช้เครื่องยนต์สถานะจำกัดและสร้างวงจรลำดับ โดยสถานะจะมีการเข้ารหัสก่อน ส่วนวิธีวันshot (One Hot Method) เป็นวิธีที่ไม่มีการเข้ารหัสสถานะ ทำให้ง่ายแก่การตรวจสอบ แต่จะทำให้วงจรมีขนาดฮาร์ดแวร์ที่ใหญ่กว่าแบบแรก การใช้ไมโครโปรแกรมเป็นการนำเอาวิธีการให้สัญญาณไปไว้ในหน่วยความจำควบคุมซึ่งเป็นรองพิเศษที่สามารถโปรแกรมได้ การใช้ริชีไมโครโปรแกรมจะต้องมีการเพตช์คำสั่งในROMมาและอีกชีคิวต์เพื่อส่งสัญญาณออกไปอย่างเหมาะสม การออกแบบโดยใช้ริชีไมโครโปรแกรมต้องคำนึงถึงการเข้ารหัสคำสั่ง ว่าแต่ละพิลต์ในคำสั่งมีขนาดเท่าไร เพราะจะมีผลกับขนาดของหน่วยความจำควบคุมที่จะใช้ และมีผลกับการอีกชีคิวต์ไมโครโปรแกรมว่าเมื่อส่งสัญญาณไปจะทำได้เร็วเท่าไร และยังได้แสดงการตัวอย่างการออกแบบของ MIPS อีกด้วย

คำถ้ามก้าวบท

- การออกแบบด้วยวิธีการฮาร์ดໄว์ร์และวิธีการไมโครโปรแกรมต่างกันอย่างไร
- การทำหนดสัญญาณควบคุมต้องพิจารณาอะไรบ้าง
- ไมโครโปรแกรมต่างจากนาโนโปรแกรมอย่างไร
- วิธีการวินชอตสำหรับการออกแบบเครื่องยนต์สถานะจำกัดต่างจากวิธีทั่วไปอย่างไร
- ขนาดของромสำหรับบรรจุไมโครโปรแกรมถูกกำหนดด้วยอะไรบ้าง
- วิธีการ PLA จะดีกว่าการใช้รวมในกรณีใดบ้าง
- พิจารณาโปรแกรมต่อไปนี้ จงออกแบบการให้สัญญาณควบคุมด้วยเครื่องยนต์สถานะจำกัด

```
i=0
while (i > 0)
{
    sum[i]++;
    i++;
}
```

- จากโค้ดในข้อที่ 7 จงออกแบบด้วยวิธีการไมโครโปรแกรม และเปรียบเทียบกับการออกแบบในรูปแบบของวิธีการฮาร์ดໄว์ร์
- จงเปรียบเทียบการออกแบบด้วยวิธีไมโครโปรแกรมและวิธีฮาร์ดໄว์ร์สำหรับตัวอย่างต่อไปนี้ พิจารณาการควบคุมการทำงานแบบ MIPS ถ้ามีการรับปุ่มไปโ sen และทำการเพิ่มสัญญาณ เอกต์พุตควบคุมอีก 2 เส้น เพิ่มสถานะไปอีก 8 สถานะ โดยไปอยู่ใน Dispatch 2 จำนวน 3 สถานะ และที่เหลือเพิ่มใน Dispatch 3 ทั้งหมด อยากรู้ว่า
 - ขนาดของромจะเป็นเท่าไร ขนาดของ PLA จะเป็นเท่าไร

- 9.2 ถ้าใช้รูปแบบใหม่โคโรโปรแกรม จงวาดรูปแสดงโครงสร้างใหม่นี้ และแสดงรูปโครงสร้างการควบคุมของใหม่โคโรโปรแกรมใหม่นี้ด้วย
10. พิจารณาคำสั่ง MIPS รูปแบบที่เพิ่มขึ้นมา เช่น ADD Rd, Rs, Rt โดย Rt จะอยู่ในรูปแบบ 100(R1) อธิบายถึงการเปลี่ยนแปลงนี้กับรูปแบบคำสั่ง Data Path การให้สัญญาณควบคุมรวมไปถึงขั้นตอนต่างๆ ของการทำงาน

