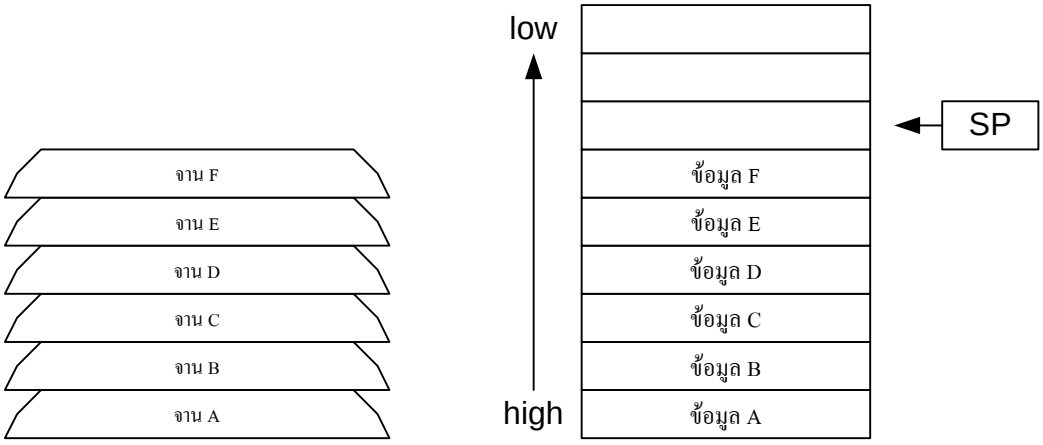
## บทที่ 5 Subroutine

ในการเขียนโปรแกรม ผู้เขียนมักจะไม่ได้เขียนโปรแกรมขนาดใหญ่เพียงโปรแกรมเดียว แต่จะเขียนฟังก์ชั่นย่อยๆ เก็บไว้ เผื่อที่จะมีการเรียกใช้งานซ้ำหลายๆ ครั้งภายในโปรแกรมหลัก หรือเก็บสะสมเอาไว้ใช้งานในโปรแกรมต่อๆ ไปที่จะเขียนขึ้นในอนาคต

**Stack**

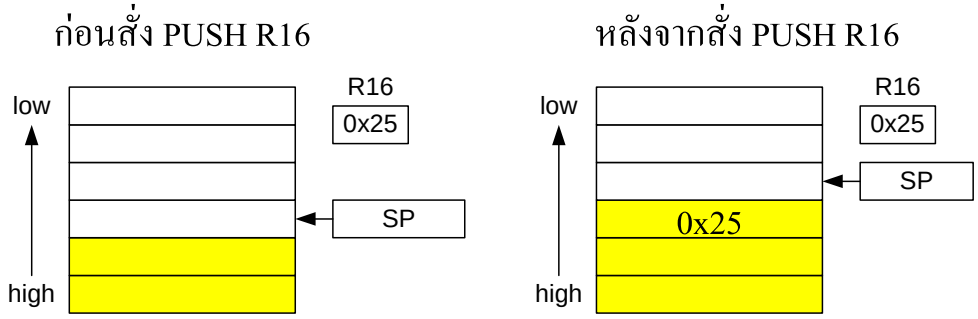
คือโครงสร้างการเก็บข้อมูลอย่างหนึ่งในระบบคอมพิวเตอร์ ซึ่งมีการทำงานแบบ LIFO (Last-in-First-Out) ซึ่งหากจะกล่าวให้เข้าใจง่ายก็เปรียบได้กับการวางจานลงบนที่วาง ซึ่งจานที่วางลงไปก่อนจะถูกจานที่วางตามหลังทับอยู่ ทำให้การที่เราจะเอาจานใบที่อยู่ข้างใต้ขึ้นมาได้นั้น เราต้องเอาจานใบที่ซ้อนทับอยู่บนมันออกไปให้หมดก่อน ดังรูปที่ S1(ซ้ายมือ)




รูปที่ S1 โครงสร้างการจัดเก็บข้อมูลแบบ Stack

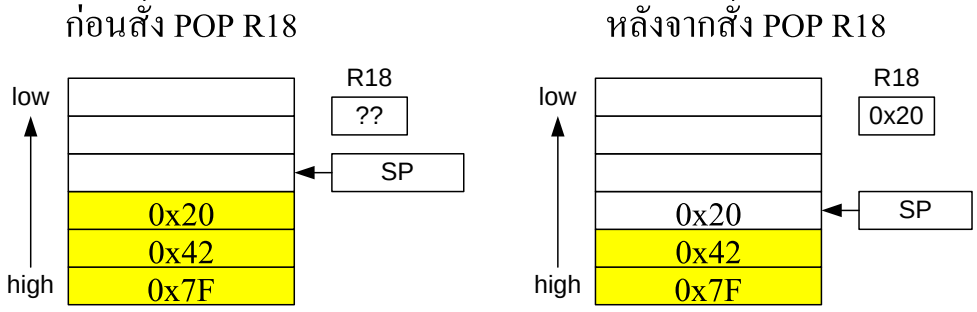
ซีพียูส่วนใหญ่จะสนับสนุนโครงสร้างการจัดการข้อมูลแบบสแตกในระดับฮาร์ดแวร์ กล่าวคือจะมีคำสั่ง Push สำหรับใช้ในการจัดเก็บข้อมูลลงสแตก และมีคำสั่ง pop สำหรับนำข้อมูลจากสแตกออกมาใช้งาน โดยมีรีจิสเตอร์ SP เป็นตัวบอกว่าตำแหน่งหน่วยความจำที่พร้อมจะใช้เป็นสแตกสำหรับเก็บข้อมูลชิ้นบนสุดนั้นอยู่ ณ ตำแหน่งใด ซีพียู AVR จะใช้หน่วยความจำข้อมูล SRAM ภายในสำหรับสร้าง Stack

ค่าของรีจิสเตอร์ SP ของซีพียู AVR แต่ละตัวจะมีค่าเริ่มต้นเท่ากับตำแหน่งสูงสุดที่หน่วยความจำ SRAM ของซีพียูแต่ละเบอร์มีให้ ยกตัวอย่างเช่นซีพียูรุ่น ATMEGA328P มีหน่วยความจำ SRAM ภายในอยู่จำนวน 2 กิโลไบต์ แสดงเมื่อซีพียูตัวนี้ถูกรีเซ็ทแล้วค่าในรีจิสเตอร์ SP จะมีค่าเท่ากับ 0x8FF เป็นต้น และสแตกของ AVR จะเริ่มจากตำแหน่ง Address มากก่อน แล้วเมื่อใส่ข้อมูลลงไปในสแตก ตัว SP จะลดค่าลงไปยังตำแหน่ง Address ค่าน้อยกว่า ดังรูปที่ S1 (รูปขวามือ)




รูปที่ S2 การทำงานของคำสั่ง PUSH ในซีพียู AVR

คำสั่ง PUSH จะทำการนำค่าจากรีจิสเตอร์ขนาด 8 บิตไปเก็บไว้ใน Stack โดยจะนำค่าไปใส่ไว้ในหน่วยความจำตำแหน่งที่รีจิสเตอร์รีจิสเตอร์ SP ชี้อยู่ก่อน เมื่อเขียนค่าเสร็จซีพียูจะทำการเลื่อนค่าใน Stack pointer ให้มีค่าต่ำลงโดยอัตโนมัติ 1 ค่า เพื่อชี้ตำแหน่งที่พร้อมจะเก็บสแตกชิ้นถัดไป ดังรูปที่ S2




รูปที่ S3 การทำงานของคำสั่ง POP ของซีพียู AVR

คำสั่ง POP จะทำการเพิ่มค่าของ SP ขึ้น 1 ค่าก่อน จากนั้นจึงนำค่าจากหน่วยความจำตำแหน่งที่ SP ชี้อยู่ ออกมาใส่ในรีจิสเตอร์ที่กำหนด ดังรูปที่ S3

**Subroutine**

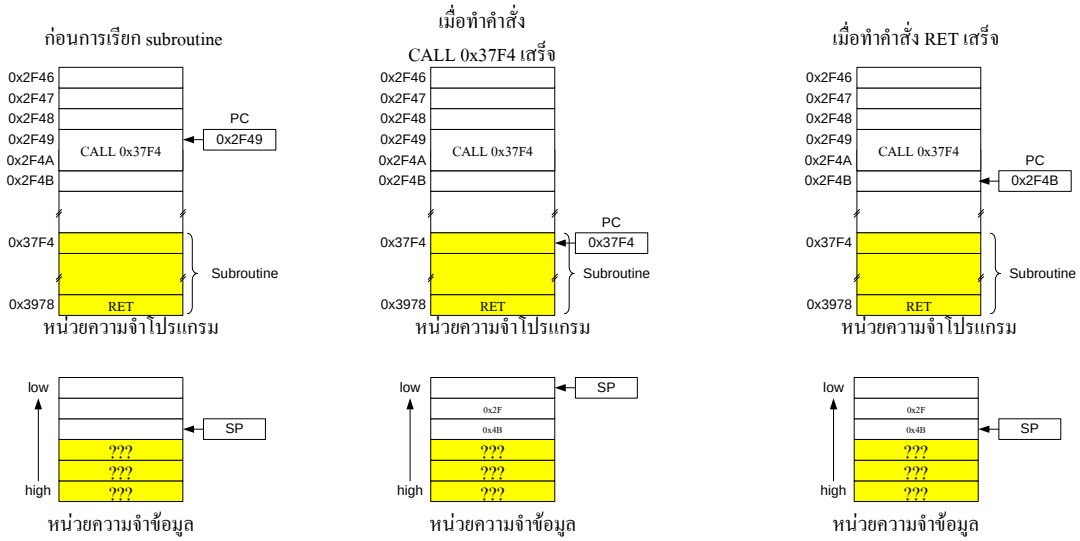
ในการเขียนโปรแกรมภาษาระดับสูง เช่นภาษาซี เรามักจะไม่นิยมเขียนโปรแกรมขนาดใหญ่เพียงชิ้นเดียว แต่นิยมสร้างฟังก์ชั่นย่อยเก็บไว้ การเขียนภาษาแอสเซมบลี้ก็เช่นเดียวกัน ในภาษาแอสเซมบลี้ส่วนใหญ่ จะนิยมเรียกฟังก์ชั่นย่อยว่า Subroutine [1]

---

[1] ซีพียูบางตระกูลจะเรียกฟังก์ชั่นย่อยในภาษาแอสเซมบลีว่า Procedure เช่นสถาปัตยกรรม MIPS เป็นต้น

สถาปัตยกรรม AVR มีคำสั่งในการเรียกใช้ subroutine คือคำสั่ง CALL โดยมันจะทำการเก็บค่า Return address เอาไว้ในสแตกก่อนที่ซีพียูจะกระโดดไปทำคำสั่งใน Subroutine มีข้อสังเกตคือคำสั่ง CALL จะมีขนาด 32 บิต ซึ่งกินเนื้อที่หน่วยความจำโปรแกรมจำนวน 2 ตำแหน่ง[2]

เป็นหน้าที่ของโปรแกรมเมอร์ที่จะต้องใส่คำสั่ง RET เอาไว้ที่ท้าย Subroutine เพื่อบอกซีพียูว่าถึงจุดสิ้นสุดของ subroutine แล้ว ซีพียูเมื่อเจอคำสั่งนี้ก็จะทำการโหลดค่า return address ออกจากสแตกออกมาใส่ใน program counter เพื่อที่จะให้ซีพียูทำคำสั่งหลังการ CALL ต่อไป ดังแสดงในรูปที่ S4




รูปที่ S4 ค่าสถานะของซีพียู (a) ก่อนเรียก Subroutine (b) ระหว่างการเรียก Subroutine (c) หลังการเรียก Subroutine เสร็จ

**หลักกว้างๆ ในการเขียน Subroutine**

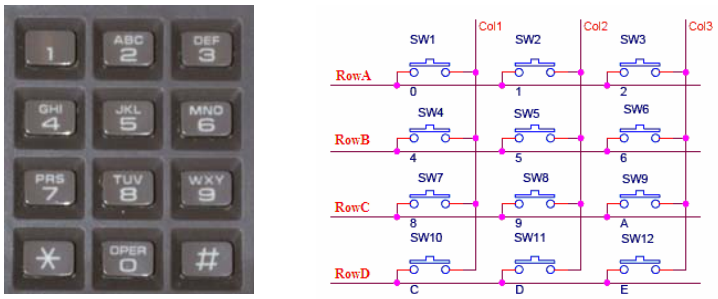
1. กำหนด spec ว่าตัว subroutine จะรับพารามิเตอร์เข้ามาทางรีจิสเตอร์ใดบ้าง
2. กำหนด spec ว่าตัว subroutine จะ return ค่าที่คำนวณได้กลับทางรีจิสเตอร์ใด
3. หากตัว subroutine จะใช้รีจิสเตอร์ตัวใดเป็นตัวแปรชั่วคราวไว้เก็บ local variable ภายในตัว subroutine เองแล้ว ผู้เขียนจะต้อง push ค่าในรีจิสเตอร์ดังกล่าวไว้ใน stack ก่อนการใช้งาน
4. ก่อนออกจากตัว subroutine ผู้เขียนจะต้อง pop ค่ารีจิสเตอร์ที่ใส่ไว้ใน stack กลับสู่ที่เดิม ท้าย subroutine จะต้องจบด้วยคำสั่ง RET

---

[2] หน่วยความจำโปรแกรมของซีพียูส่วนใหญ่มักจะมีให้ใช้ตำแหน่งละ 1 ไบต์ แต่ของซีพียู AVR กลับกำหนดให้ 1 ตำแหน่งหน่วยความจำมีขนาด 2 ไบต์

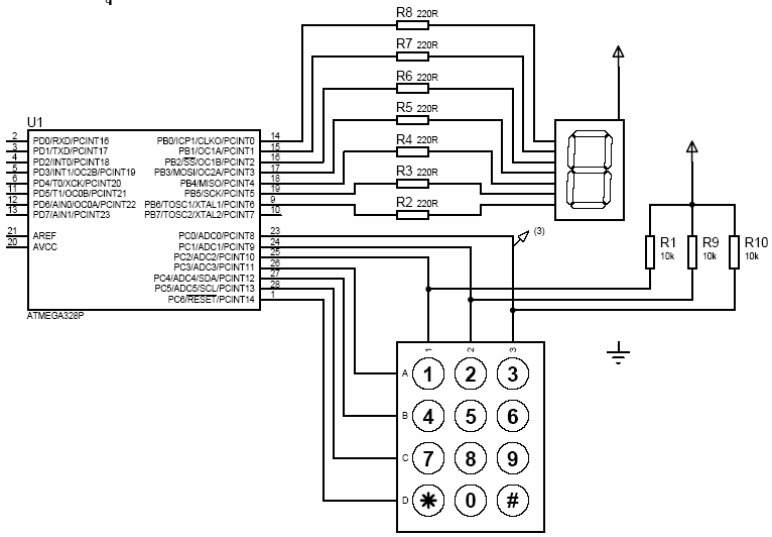
# การสแกน Keypad

การต่อคีย์แพดขนาด 12 ปุ่มกับตัวไมโครคอนโทรลเลอร์ หากใช้การต่อสวิทช์ 1 ตัวกับพอร์ตของไมโครคอนโทรลเลอร์ 1 พอร์ตจะทำให้สูญเสียจำนวนพอร์ตไปกับการใช้เป็นอินพุตเป็นจำนวนมาก ดังนั้นคีย์แพดส่วนใหญ่จะใช้วิธีการต่อแบบ Matrix ดังรูปที่ K1




รูปที่ K1 สวิทช์คีย์แพดแบบ Matrix

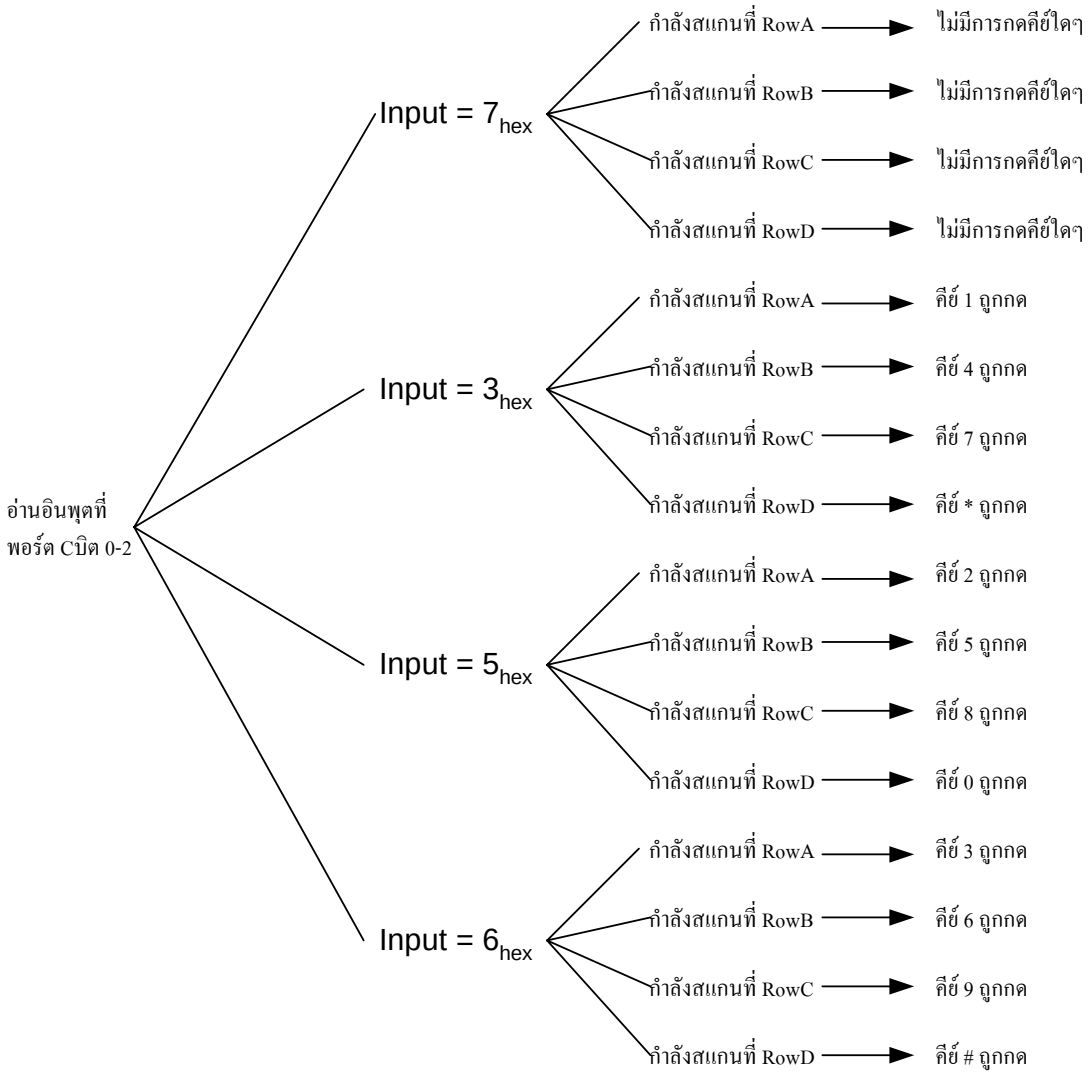
จากรูปจะเห็นว่าคีย์แพดต่อกับพอร์ต C โดยขาสัญญาณด้านแถว (Row) จะต่อกับบิตที่ 3-6 และขาสัญญาณด้านหลัก (Column) ต่อกับบิตที่ 0-2 ในการตรวจสอบว่ามีการกดคีย์อะไรเราจะใช้วิธีการสแกนค่าโดยส่งเอาต์พุตออกทางบิตที่ 3-6 และอ่านค่าสถานะว่ามีการกดคีย์หลักอะไรที่บิตที่ 0-2




รูปที่ K2 การต่อซีพียู AVR กับคีย์แพดและแอลอีดแบบ 7-segment

พอร์ต C บิตที่ 0-2 มีตัวต้านทาน Pull-up ต่ออยู่เพื่อให้สามารถอ่านค่าได้ลอจิกสูงในขณะที่ไม่มีการกดคีย์ใดๆ โดยในที่นี้จะถือว่าคีย์แพดนี้สามารถกดปุ่มได้แค่ครั้งละ 1 ปุ่มเท่านั้น ในการสแกนตัวซีพียูจะต้องส่งค่าลอจิกต่ำออกมาทางขาด้านแถวที่ต้องการสแกน เมื่อมีการกดปุ่มใดๆ บนคีย์แพดจะสามารถอ่านค่าลอจิกเอาต์พุตได้ดังนี้

| ค่าที่อ่านได้จากบิต 2,1,0 ของพอร์ต C | สถานะ |
|---|---|
| 111 ($7_{hex}$) | ไม่มีการกดปุ่มใดๆ |
| 011 ($3_{hex}$) | มีการกดปุ่มที่คอลัมน์ 1 |
| 101 ($5_{hex}$) | มีการกดปุ่มที่คอลัมน์ 2 |
| 110 ($6_{hex}$) | มีการกดปุ่มที่คอลัมน์ 3 |




รูปที่ K3 การตัดสินใจของโปรแกรมว่าคีย์ใดถูกกดทั้ง 16 กรณี

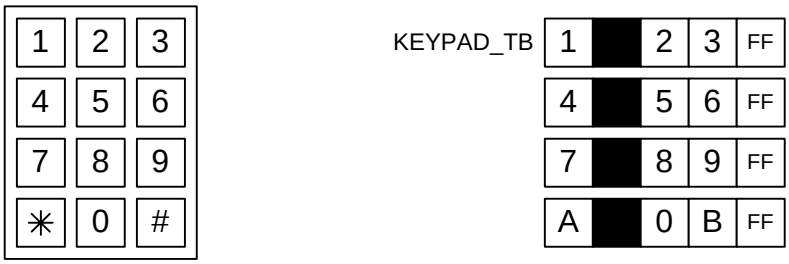
จากรูปที่ K3 จะเห็นว่าการตัดสินใจว่าคีย์ใดถูกกด จะขึ้นอยู่กับสถานะของโปรแกรมว่ากำลังสแกนอยู่ที่ Row ใด และอินพุตที่อ่านได้ที่พอร์ต C บิต 0-2 ในการตัดสินใจบอกว่ากำลังกดคีย์อะไรอีกทีหนึ่ง จะเห็นว่าการตัดสินใจมี 16 กรณีด้วยกัน ซึ่งหากจะเขียนภาษาแอสเซมบลี้ด้วยโครงสร้างเงื่อนไข if-then-else หรือเขียนเป็นแบบโครงสร้าง switch-case จะทำให้การเขียน

โปรแกรมมีความยุ่งยากอย่างมาก วิธีที่ง่ายกว่าคือการสร้างตาราง Look-up table โดยทำการสร้างตารางที่จะเปิดค่าของแต่ละคอลัมน์ ดังรูปที่ K4 ซึ่งจะเห็นว่าค่าในตาราง KEYPAD_TB จะเก็บค่าของการกดปุ่มแต่ละปุ่มเอาไว้ การกดปุ่มในคอลัมน์ 1, 2, 3 จะให้ค่าออกมาเท่ากับ 3, 5, 6 ตามลำดับ และหากไม่มีการกดปุ่มจะได้ค่า 0xFF ออกมา ค่าตำแหน่งแอดเดรสของตาราง KEYPAD_TB ที่จะเปิดสามารถคำนวณได้จาก

ตำแหน่งแอดเดรสเป้าหมาย = KEYPAD_TB + COL + ROW-3

เมื่อ

- KEYPAD_TB คือค่าตำแหน่งแอดเดรสของหน่วยความจำโปรแกรมตำแหน่งเริ่มต้นของตาราง lookup table
- COL คือค่าที่อ่านได้จากพอร์ต C บิต 0-2
- ROW มีค่าเปลี่ยนแปลงตาม Row ที่กำลังสแกน โดยมีค่าดังนี้
  - ROW = 0 เมื่อกำลังสแกนอยู่ ณ Row A
  - Row = 5 เมื่อกำลังสแกนอยู่ ณ Row B
  - ROW = 10 เมื่อกำลังสแกนอยู่ ณ Row C
  - Row = 15 เมื่อกำลังสแกนอยู่ ณ Row D




รูปที่ K4 Lookup table ของการสแกนค่าคีย์แพด