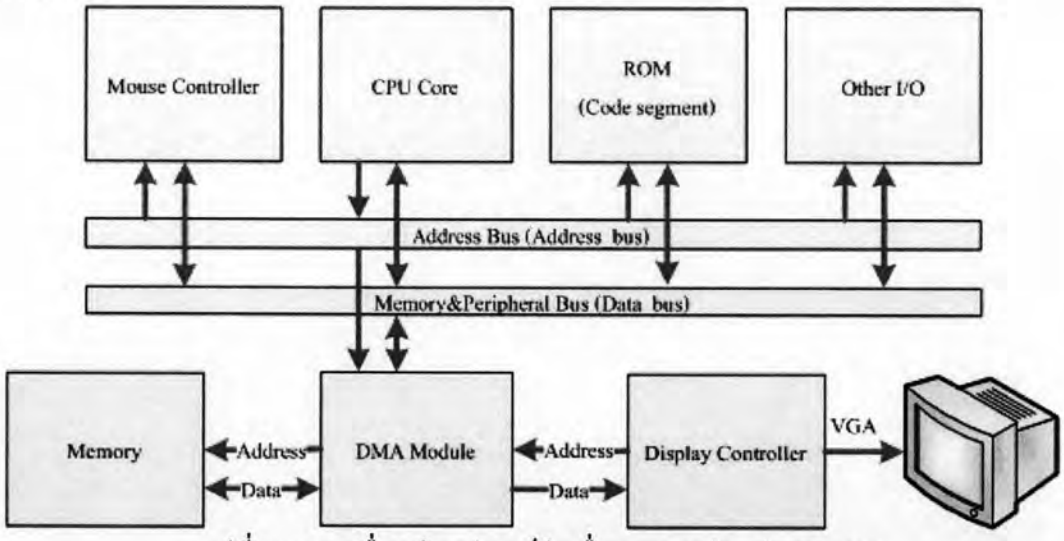
บทที่ 5
การพัฒนาโปรแกรม

ในบทนี้ จะกล่าวถึงโครงสร้างการทำงานของหน่วยประมวลผล ที่จำเป็นสำหรับการพัฒนาโปรแกรมซึ่งเขียนด้วยภาษาแอสเซมบลี (Assembly language) ของหน่วยประมวลผลนี้

5.1 อุปกรณ์ต่อเชื่อมกับหน่วยประมวลผล

หน่วยประมวลผลนั้นเปรียบเสมือนสมองของมนุษย์ ซึ่งคงไม่มีคุณค่าอันใดหากมันปราศจากอวัยวะแขนขาที่มาทำตามสิ่งที่เราคิด หรือหูตาเพื่อรับสิ่งต่างๆ ภายนอกเข้ามาสู่ตัวเรา เช่นเดียวกันหน่วยประมวลผลจำเป็นต้องมีอุปกรณ์ต่อเชื่อม (Peripheral device) เพื่อนำเอาข้อมูลจากภายนอกเข้าสู่หน่วยประมวลผล และนำเอาผลจากการทำงานออกมาใช้ประโยชน์

หน่วยประมวลผลของงานวิจัยนี้ใช้วิธีเมมโมรี่แม็พไอ/โอ (Memory-mapped I/O) ในการทำงานร่วมกับอุปกรณ์เชื่อมต่อต่างๆ ดังแสดงในรูปภาพที่ 5.1 ด้วยวิธีการนี้อุปกรณ์ต่อเชื่อมแต่ละตัวจะได้รับเลขที่อยู่ (Address) เพื่อใช้สำหรับการติดต่อระหว่างหน่วยประมวลผลและอุปกรณ์ต่อเชื่อม



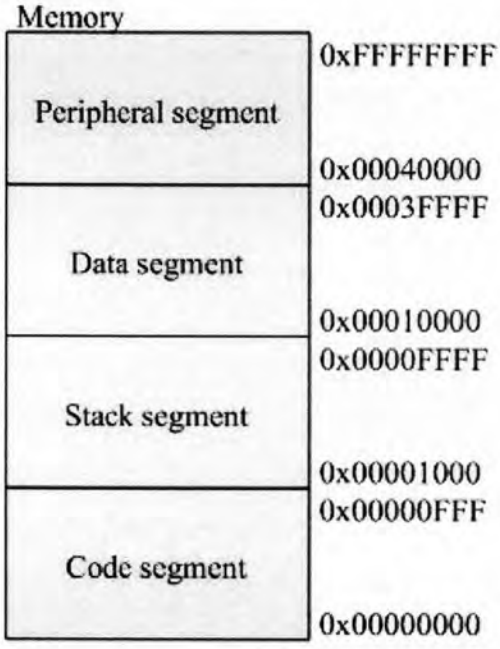
รูปที่ 5.1 การเชื่อมต่ออุปกรณ์ต่อเชื่อมของหน่วยประมวลผล

หน่วยประมวลผลสามารถส่งข้อมูลให้อุปกรณ์ต่อเชื่อมได้ โดยการเขียนข้อมูลลงในหน่วยความจำตำแหน่งที่เป็นเลขที่อยู่ของอุปกรณ์ต่อเชื่อมนั้นๆ ในทำนองเดียวกัน การอ่านข้อมูลจากหน่วยความจำตำแหน่งที่เป็นเลขที่อยู่ของอุปกรณ์ต่อเชื่อม จะเป็นการส่งข้อมูลจากอุปกรณ์ต่อเชื่อมนั้นๆ เข้าสู่หน่วยประมวลผล

5.2 การจัดสรรพื้นที่หน่วยความจำ

เนื่องจากหน่วยประมวลผลนี้มีหน่วยความจำโปรแกรม (Program memory) และหน่วยความจำข้อมูล (Data memory) ร่วมกัน สิ่งที่ผู้พัฒนาโปรแกรมต้องสนใจคือ การแบ่งพื้นที่ในหน่วยความจำ หน่วยประมวลผลของงานวิจัยนี้ แบ่งหน่วยความจำออกเป็น 4 ส่วนได้แก่ ส่วนโปรแกรม (Code segment) ส่วนแสตก (Stack segment) ส่วนข้อมูล (Data segment) และส่วนอุปกรณ์ต่อเชื่อม (Peripheral segment)

รูปที่ 5.2 แสดงตัวอย่างการจัดสรรพื้นที่หน่วยความจำที่ทำในงานวิจัยนี้ โดยให้หน่วยความจำตั้งแต่ตำแหน่งที่ 0 (0x00000000) ถึง 4095 (0x00000FFF) เป็นส่วนโปรแกรม ส่วนอื่นๆ ได้รับการจัดสรรพื้นที่ดังแสดงในรูปที่ 5.2



รูปที่ 5.2 ตัวอย่างการจัดสรรพื้นที่หน่วยความจำที่ใช้ในงานวิจัยนี้

5.2.1 หน่วยความจำส่วนโปรแกรม

หน่วยความจำส่วนโปรแกรม (Code segment) มีขนาด 16 กิโลไบต์ (kByte) ได้รับการกำหนดเลขที่อยู่ตั้งแต่ 0 ถึง 4095 (0x00000FFF) ดังแสดงในรูปที่ 5.2 หน่วยความจำในส่วนนี้เป็นหน่วยความจำอ่านอย่างเดียวหรือรอม (ROM) คือ หน่วยประมวลผลไม่สามารถเขียนค่าใดๆ ลงในหน่วยความจำส่วนนี้ได้

5.2.2 หน่วยความจำส่วนแสตก

หน่วยความจำส่วนแสตก (Stack segment) มีขนาด 240 กิโลไบต์ (kByte)ได้รับการกำหนดเลขที่อยู่ตั้งแต่ 4096 (0x00000FFF) ถึง 65535 (0x0000FFFF) ดังแสดงในรูปที่ 5.2 หน่วยความจำในส่วนนี้ใช้ในการเก็บแอคทิเวชันเรคคอร์ด (Activation record)

5.2.3 หน่วยความจำส่วนข้อมูล

หน่วยความจำส่วนข้อมูล (Data segment) มีขนาด 784 กิโลไบต์ (kByte)ได้รับการกำหนดเลขที่อยู่ตั้งแต่ 65536 (0x00010000) ถึง 262143 (0x0003FFFF) ดังแสดงในรูปที่ 5.2 หน่วยความจำในส่วนนี้ใช้ในการเก็บข้อมูลที่เป็นตัวแปรส่วนกลาง (Global variable) และข้อมูลแถวลำดับ (Array)

5.2.4 หน่วยความจำส่วนอุปกรณ์ต่อเชื่อม

หน่วยความจำส่วนอุปกรณ์ต่อเชื่อม (Peripheral segment) ได้รับการกำหนดเลขที่อยู่ตั้งแต่ 262144 (0x00040000) เป็นต้นไป เลขที่อยู่ที่กำหนดให้อุปกรณ์ต่อเชื่อมจะอยู่ในช่วงนี้

อนึ่ง การจัดสรรพื้นที่หน่วยความจำที่ได้กล่าวมาข้างต้น เป็นการยกตัวอย่างโดยนำเอาการจัดสรรพื้นที่หน่วยความจำที่ทำในงานวิจัยมานำเสนอ ในการใช้งานจริงนั้นขนาดของหน่วยความจำส่วนต่างๆ สามารถปรับให้เหมาะสมกับการใช้งานได้ เช่น หากการใช้งานมีการทำงานกับข้อมูลขนาดใหญ่ ต้องการพื้นที่สำหรับเก็บข้อมูล ผู้ใช้สามารถขยายขนาดของหน่วยความจำส่วนข้อมูลออกไปได้อีก อาจทำโดยการกำหนดให้หน่วยความจำส่วนข้อมูลมีเลขที่อยู่ขยายออกไปถึง 524287 (0x0007FFF) ซึ่งจะทำให้หน่วยความจำส่วนข้อมูลมีขนาดเพิ่มเป็น 1808 กิโลไบต์ (kByte) สำหรับเลขที่อยู่ของอุปกรณ์ต่อเชื่อมให้เริ่มตั้งแต่ 524288 (0xx00080000) เป็นต้นไป เพื่อไม่ให้ซ้ำกับเลขที่อยู่ของส่วนข้อมูล

อย่างไรก็ตามหน่วยความจำส่วนโปรแกรม (Code segment) ถูกเก็บในรอมสำหรับเก็บโปรแกรม ซึ่งมีขนาด 16 กิโลไบต์ การเพิ่มขนาดของหน่วยความจำส่วนโปรแกรม จำเป็นต้องเพิ่มขนาดของรอมนี้ด้วย

เช่นเดียวกัน หน่วยความจำส่วนแสตก (Stack segment) และหน่วยความจำส่วนข้อมูล (Data segment) นั้น ถูกเก็บในหน่วยความจำเข้าถึงโดยสุ่มหรือแรม (RAM) การเพิ่มขนาดของหน่วยความจำทั้งสองส่วนนี้ จำเป็นต้องเพิ่มขนาดของแรมที่ใช้ด้วย ซึ่งในงานวิจัยนี้แรมที่ใช้มีขนาด 1 เมกะไบต์ (MByte) ซึ่งเท่ากับ 1024 กิโลไบต์ เพียงพอสำหรับเก็บหน่วยความจำทั้งสองส่วนได้พอดี

5.3 การเข้าถึงตัวแปรส่วนกลาง

ตัวแปรส่วนกลาง (Global variable) ถูกเก็บอยู่ในหน่วยความจำส่วนข้อมูล นอกจากนี้ยังมีตัวแปรส่วนกลางบางส่วนถูกเก็บอยู่ในหน่วยความจำส่วนโปรแกรมด้วย การเข้าถึงตัวแปรส่วนกลางก็คือการอ่านหรือเขียนหน่วยความจำตำแหน่งที่เก็บตัวแปรส่วนกลางอยู่นั้นเอง สามารถทำได้โดยคำสั่ง LD, LDL, ST และ STL โดยคำสั่ง LD และ LDL ทำหน้าที่อ่านค่าหน่วยความจำ ส่วนคำสั่ง ST และ STL ทำหน้าที่เขียนหน่วยความจำ

ตัวอย่างต่อไปนี้แสดงโปรแกรมที่ทำการเพิ่มค่าของตัวแปรส่วนกลาง X ซึ่งมีเลขที่อยู่เป็น 80000 ขึ้น 1 ค่า

```
1:  LDL 80000
2:  ADDI 1
3:  STL 80000
```

คำสั่ง LDL 80000 เป็นการอ่านค่าของตัวแปรส่วนกลาง X มาเก็บไว้ใน AC เพื่อทำการเพิ่มค่าด้วยคำสั่ง ADDI 1 ค่าของตัวแปร X ที่ได้รับการเพิ่มค่าแล้วจะถูกใส่กลับลงไปที่เลขที่อยู่ของตัวแปร X เช่นเดิมด้วยคำสั่ง STL 80000

5.4 การเข้าถึงข้อมูลแถวลำดับ

การเข้าถึงข้อมูลแถวลำดับ (Array) มีขั้นตอนซับซ้อนกว่าการเข้าถึงตัวแปรส่วนกลาง เนื่องจากตัวแปรส่วนกลางนั้น มีการกำหนดเลขที่อยู่อย่างชัดเจน แต่สำหรับข้อมูลแถวลำดับนั้นจะต้องทำการคำนวณหาเลขที่อยู่ของข้อมูลที่ต้องการก่อน จึงจะนำเลขที่อยู่นั้นไปใช้ในการเข้าถึงหน่วยความจำได้

การคำนวณหาเลขที่อยู่ของข้อมูลในแถวลำดับนั้น ทำได้โดยการนำเลขที่อยู่ของหัวของข้อมูลแถวลำดับ บวกกับตำแหน่งของข้อมูลตัวที่ต้องการเข้าถึงตัวนั้น เช่น สมมติให้ข้อมูลแถวลำดับ A มีเลขที่อยู่เป็น 300000 เลขที่อยู่ของข้อมูลตำแหน่งที่ 3 ของข้อมูลแถวลำดับ A จะเท่ากับ 300003 เป็นต้น

สำหรับการอ่านข้อมูลในข้อมูลแถวลำดับ ใช้คำสั่ง LDX ในการอ่านค่า คำสั่ง LDX นี้มีการคำนวณหาเลขที่อยู่ของข้อมูลให้อยู่แล้ว การใช้งานคำสั่ง LDX เพียงแค่นำค่าเลขที่อยู่ของหัวของข้อมูลแถวลำดับไปใส่ไว้ในรีจิสเตอร์ AC จากนั้นใช้คำสั่ง LDX โดยให้ตัวถูกดำเนินการ (Operand) ระบุตัวแปรท้องถิ่น ซึ่งค่าของตัวแปรเฉพาะที่นั้นจะถูกใช้ในการระบุตัวข้อมูลในแถวลำดับที่ต้องการอ่าน เมื่อการทำงานคำสั่ง LDX เสร็จสิ้น ค่าของข้อมูลที่ต้องการในข้อมูลแถวลำดับถูกอ่านมาเก็บไว้ในรีจิสเตอร์ AC ดังตัวอย่างต่อไปนี้

ในตัวอย่างต่อไปนี้ กำหนดให้ค่าของตัวแปรเฉพาะที่ตัวที่ 1 ในขณะนั้นมีค่าเท่ากับ 3

```
1:  LITL 300000
2:  LDX 1
```

ตัวอย่างอ่านข้อมูลตำแหน่งที่ 3 ของข้อมูลแถวลำดับ A ที่มีเลขที่อยู่เป็น 300000 คำสั่งแรก LITL 300000 เป็นการใส่ค่าเลขที่อยู่ของข้อมูลแถวลำดับ A ซึ่งเท่ากับ 300000 ลงในรีจิสเตอร์ AC ก่อนที่จะทำการอ่านข้อมูลตำแหน่งที่ 3 ด้วยคำสั่ง LDX 1 ซึ่งค่าของตัวแปรเฉพาะที่ตัวที่ 1 ขณะนั้นมีค่าเท่ากับ 3 จะถูกบวกเข้ากับค่าในรีจิสเตอร์ AC ก่อนใช้ค่านั้นระบุตำแหน่งของหน่วยความจำในการอ่านค่า

ในการเขียนข้อมูลลงในข้อมูลแถวลำดับนั้น ใช้คำสั่ง STV มีขั้นตอนซับซ้อนกว่าการอ่าน เนื่องจากคำสั่ง STV ไม่มีการคำนวณหาเลขที่อยู่ของข้อมูลที่ต้องการอ่านในข้อมูลแถวลำดับ การคำนวณหาเลขที่อยู่ของข้อมูล ต้องกระทำในโปรแกรมให้เสร็จแล้วใส่ค่าเลขที่อยู่ดังกล่าวไว้ในรีจิสเตอร์ AC จากนั้นจึงใช้คำสั่ง STV โดยให้ตัวถูกดำเนินการระบุตัวแปรเฉพาะที่ที่ต้องการเขียนลงในข้อมูลแถวลำดับ ดังตัวอย่างต่อไปนี้

```
1:  LITL 300000
2:  ADDI 3
3:  STV 1
```

ตัวอย่างเขียนข้อมูลตำแหน่งที่ 3 ของข้อมูลแถวลำดับ A ที่มีเลขที่อยู่เป็น 300000 โดยค่าที่ต้องการเขียนถูกเก็บอยู่ในตัวแปรเฉพาะที่ตัวที่ 1 คำสั่งแรก LITL 300000 เป็นการใส่ค่าเลขที่อยู่ของข้อมูลแถวลำดับ A ซึ่งเท่ากับ 300000 ลงในรีจิสเตอร์ AC ก่อน จากนั้นคำสั่ง ADDI 3 ทำการคำนวณหาเลขที่อยู่ของข้อมูลตำแหน่งที่ 3 ของข้อมูลแถวลำดับ A เมื่อได้เลขที่อยู่ของข้อมูลที่ต้องการเขียนแล้ว ใช้คำสั่ง STV ระบุตัวแปรเฉพาะที่ที่เก็บค่าที่ต้องการเขียน ในที่นี้ค่าถูกเก็บในตัวแปรเฉพาะที่ตัวที่ 1 จึงใช้คำสั่ง STV 1

5.5 การเข้าถึงตัวแปรเฉพาะที่

ตัวแปรเฉพาะที่ (Local variable) ถูกเก็บอยู่ในแอคทิเวชันเรคคอร์ด (Activation record) เลขที่อยู่ของตัวแปรเฉพาะที่จึงขึ้นอยู่กับเลขที่อยู่ของแอคทิเวชันเรคคอร์ด ซึ่งเลขที่อยู่ของแอคทิเว-

ชันเรคคอร์ดนี้ ถูกเก็บอยู่ในรีจิสเตอร์ FP เลขที่อยู่ของตัวแปรเฉพาะที่จึงต้องคำนวณหาจากค่าของรีจิสเตอร์ FP อีกทีหนึ่ง

การเข้าถึงตัวแปรเฉพาะที่สามารถทำได้ด้วยคำสั่ง GET และ PUT โดยให้ตัวถูกดำเนินการ (Operand) ระบุตัวแปรเฉพาะที่ที่ต้องการเข้าถึง คำสั่ง GET และ PUT จะคำนวณหาค่าเลขที่อยู่ที่ตัวแปรเฉพาะที่ถูกเก็บอยู่ เพื่อใช้ในการเข้าถึงตัวแปรเฉพาะที่นั้นๆ

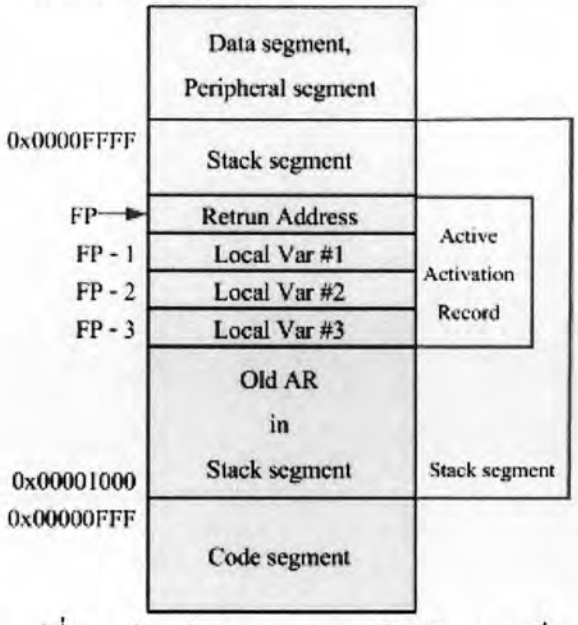
ตัวอย่างต่อไปนี้แสดงโปรแกรมที่ทำการนำค่าของตัวแปรเฉพาะที่ M ไปใส่ไว้ในตัวแปรเฉพาะที่ N โดยตัวแปรเฉพาะที่ M และ N เป็นตัวแปรเฉพาะที่ตัวที่ 1 และ 2 ตามลำดับ

```
1:  GET 1
2:  PUT 2
```

คำสั่ง GET 1 ทำการอ่านค่าของตัวแปรเฉพาะที่ M มาเก็บไว้ในรีจิสเตอร์ AC จากนั้นคำสั่ง PUT 2 ทำการเขียนค่าของรีจิสเตอร์ AC ลงในตัวแปรเฉพาะที่ N

5.6 แอคทิเวชันเรคคอร์ด

แอคทิเวชันเรคคอร์ด (Activation record) เป็นโครงสร้างการทำงานของโปรแกรมย่อย แอคทิเวชันเรคคอร์ดนี้ทำหน้าที่เก็บตำแหน่งเลขที่อยู่กลับ (Return address) และตัวแปรเฉพาะที่ (Local variable) ของโปรแกรมย่อย แอคทิเวชันเรคคอร์ดนี้ถูกเก็บในหน่วยความจำส่วนแสตก (Stack segment) แอคทิเวชันเรคคอร์ดของโปรแกรมย่อยที่กำลังทำงานอยู่ (Active activation record) ถูกชี้โดยรีจิสเตอร์ FP ดังแสดงในรูปที่ 5.3



รูปที่ 5.3 ส่วนประกอบของแอคทิเวชันเรคคอร์ด

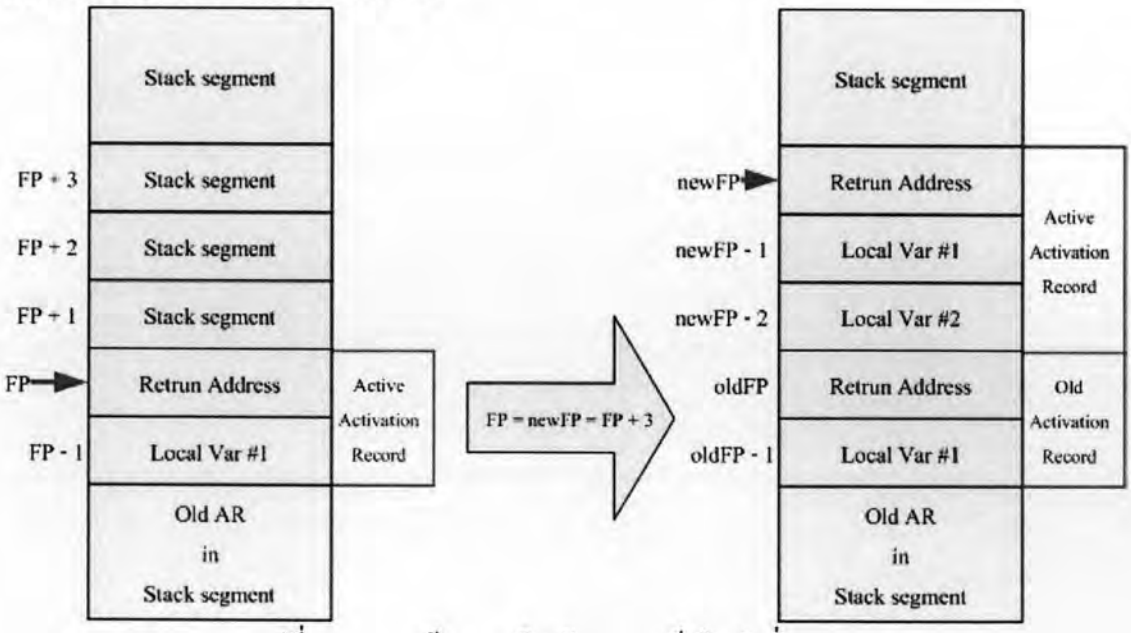
ในการทำงานของหน่วยประมวลผล เมื่อมีการเรียกโปรแกรมย่อยด้วยคำสั่ง CALL หน่วยประมวลผลจะทำการสร้างแอคทิเวชันเรคคอร์ดสำหรับโปรแกรมย่อยที่ถูกเรียก โดยแอคทิเวชันเรคคอร์ดใหม่นี้ จะถูกสร้างในตำแหน่งเลขที่อยู่ที่สูงกว่าตำแหน่งเลขที่อยู่ของแอคทิเวชันเรคคอร์ดเดิม เมื่อการทำงานในโปรแกรมย่อยสิ้นสุด มีการกลับสู่โปรแกรมหลักด้วยคำสั่ง RET หน่วยประมวลผลจะทำลายแอคทิเวชันเรคคอร์ดของโปรแกรมย่อยที่ทำการคืนค่า

ขนาดของแอคทิเวชันเรคคอร์ดขึ้นอยู่กับจำนวนตัวแปรเฉพาะที่ (Local variable) ของโปรแกรมย่อยนั้นๆ ขนาดของแอคทิเวชันเรคคอร์ดเท่ากับ N + 1 โดย N เป็นจำนวนตัวแปรเฉพาะที่ของโปรแกรมย่อย ในที่นี้ขนาดของแอคทิเวชันเรคคอร์ดวัดจากจำนวนตำแหน่งในหน่วยความจำ ที่ใช้ในการเก็บแอคทิเวชันเรคคอร์ด

ตัวอย่างแอคทิเวชันเรคคอร์ดในรูปที่ 5.3 ใช้พื้นที่ในหน่วยความจำจำนวน 4 ตำแหน่ง โดยใช้ในการเก็บเลขที่อยู่สำหรับใช้ในตอนคืนค่าจากโปรแกรมย่อย 1 ตำแหน่ง และเก็บตัวแปรเฉพาะที่จำนวน 3 ตัวใช้หน่วยความจำ 3 ตำแหน่ง

ขนาดของแอคทิเวชันเรคคอร์ดนี้มีความสำคัญในการพัฒนาโปรแกรม เนื่องจากในคำสั่ง CALL และ RET ต้องระบุขนาดของแอคทิเวชันเรคคอร์ดที่ต้องสร้างหรือทำลายด้วย หากขนาดของแอคทิเวชันเรคคอร์ดที่ระบุในคำสั่งไม่ถูกต้อง จะทำให้เกิดความผิดพลาดในการทำงานของหน่วยประมวลผลได้

การสร้างแอคทิเวชันเรคคอร์ดตอนกระทำคำสั่ง CALL ทำโดยการเพิ่มค่าของรีจิสเตอร์ FP ไปเท่ากับขนาดของแอคทิเวชันเรคคอร์ดที่ระบุในคำสั่ง ซึ่งทำให้รีจิสเตอร์ FP ชี้ไปที่แอคทิเวชันเรคคอร์ดตัวใหม่ที่มีขนาดเท่ากับขนาดที่ระบุในคำสั่ง ในรูปที่ 5.4 เป็นตัวอย่างของการสร้างแอคทิเวชันเรคคอร์ดที่มีขนาดเท่ากับ 3



รูปที่ 5.4 การสร้างแอคทิเวชันเรคคอร์ดในคำสั่ง CALL

5.7 กลไกในการเรียกโปรแกรมย่อย

กลไกในการเรียกโปรแกรมย่อย ประกอบด้วย 2 ขั้นตอน โดยต้องทำการส่งผ่านพารามิเตอร์ (Parameter) จากโปรแกรมหลักไปยังโปรแกรมย่อยก่อน แล้วจากนั้นจึงทำการจองพื้นที่สำหรับเก็บตัวแปรเฉพาะที่ (Local variable) ของโปรแกรมย่อย

5.7.1 การส่งผ่านพารามิเตอร์

การส่งผ่านพารามิเตอร์ (Parameter) จากโปรแกรมหลักไปโปรแกรมย่อย ทำได้โดยการเขียนค่าลงในแอคทิเวชันเรคคอร์ดของโปรแกรมย่อย การเขียนค่าลงในแอคทิเวชันเรคคอร์ดของโปรแกรมย่อย สามารถทำได้โดยใช้คำสั่ง PASS ซึ่งจะทำการเขียนค่าลงในหน่วยความจำตำแหน่งที่ค่าของรีจิสเตอร์ FP บวกกับค่าของตัวถูกดำเนินการ (Operand) ที่ระบุในคำสั่ง สำหรับโปรแกรมย่อยที่มีการรับพารามิเตอร์จากโปรแกรมหลัก การส่งผ่านพารามิเตอร์นี้จะต้องกระทำให้เสร็จก่อนการเรียกโปรแกรมย่อยทุกครั้ง

สำหรับการส่งพารามิเตอร์ไปยังตัวแปรตัวที่ N ของโปรแกรมย่อยที่มีขนาดแอคทิเวชันเรคคอร์ดเท่ากับ M นั้น สามารถทำได้โดยใช้คำสั่ง PASS M-N

ยกตัวอย่างประกอบรูปที่ 5.4 หากต้องการส่งค่าพารามิเตอร์ ซึ่งเป็นตัวแปรเฉพาะที่ (Local variable) ตัวที่ 2 ของโปรแกรมย่อยซึ่งแอคทิเวชันเรคคอร์ดมีขนาดเท่ากับ 3 สามารถทำได้โดยใช้คำสั่ง PASS 1 ซึ่งจะทำการเขียนค่าลงในหน่วยความจำตำแหน่งที่ FP + 1 เมื่อทำการเรียกโปรแกรมย่อยตำแหน่งดังกล่าวจะเป็นตำแหน่งที่เก็บตัวแปรเฉพาะที่ตัวที่ 2 ของโปรแกรมย่อย

ดังนั้นสิ่งที่ผู้พัฒนาโปรแกรมต้องทราบในการส่งผ่านพารามิเตอร์จึงประกอบด้วย (1) ตัวแปรที่เป็นพารามิเตอร์ว่าเป็นตัวแปรตัวไหนของโปรแกรมย่อย (2) ขนาดของแอคทิเวชันเรคคอร์ดของโปรแกรมย่อย เพื่อให้ใช้งานคำสั่ง PASS ได้อย่างถูกต้อง

5.7.2 การจองพื้นที่สำหรับตัวแปรเฉพาะที่

เมื่อทำการส่งผ่านพารามิเตอร์ให้โปรแกรมย่อยเสร็จเรียบร้อยแล้ว สิ่งที่ต้องทำต่อไปในการเรียกโปรแกรมย่อยคือ การจองพื้นที่สำหรับตัวแปรเฉพาะที่ (Local variable) ของโปรแกรมย่อย ซึ่งก็คือการสร้างแอคทิเวชันเรคคอร์ดของโปรแกรมย่อยนั้นเอง เนื่องจากการเรียกโปรแกรมย่อยต้องทำการการจองพื้นที่สำหรับตัวแปรเฉพาะที่ทุกครั้ง การจองพื้นที่นี้จึงรวมอยู่ในการทำงานของคำสั่ง CALL

คำสั่ง CALL มีตัวถูกดำเนินการ 2 ตัว ตัวหนึ่งใช้ในการระบุเลขที่อยู่ของโปรแกรมย่อย อีกตัวหนึ่งใช้ในการบอกขนาดของแอคทิเวชันเรคคอร์ดที่ต้องสร้างสำหรับโปรแกรมย่อยที่ถูกเรียก ซึ่งแอคทิเวชันเรคคอร์ดมีขนาดเท่ากับ จำนวนตัวแปรเฉพาะที่ของโปรแกรมย่อยบวกหนึ่ง ดังที่ได้กล่าวไปแล้ว

การใส่ขนาดของแอคทิเวชันเรคคอร์ดน้อยเกินไป จะทำให้พื้นที่ที่จองไว้ไม่เพียงพอสำหรับแอคทิเวชันเรคคอร์ดของโปรแกรมย่อย ในกรณีนี้แอคทิเวชันเรคคอร์ดของโปรแกรมย่อยจะซ้อนทับแอคทิเวชันเรคคอร์ดของโปรแกรมหลัก ทำให้แอคทิเวชันเรคคอร์ดของโปรแกรมหลักเสียหาย และก่อให้เกิดความผิดพลาดขึ้นในการทำงานของหน่วยประมวลผล

การใส่ขนาดของแอคทิเวชันเรคคอร์ดมากเกินไป แม้ไม่ก่อให้เกิดความผิดพลาดในการทำงานของหน่วยประมวลผล แต่ก็ทำให้ประสิทธิภาพของหน่วยประมวลผลลดลง เนื่องจากแอคทิ-เวชันเรคคอร์ดถูกเก็บอยู่ในหน่วยความจำส่วนแสตก แอคทิเวชันเรคคอร์ดที่ใหญ่เกินไปย่อมทำให้หน่วยความจำส่วนแสตกเต็มได้ง่ายขึ้น

5.8 กลไกในการคืนค่าจากโปรแกรมย่อย

สำหรับการคืนค่าจากโปรแกรมย่อยที่มีการคืนค่า สามารถทำได้โดยการนำค่าดังกล่าวใส่ไว้ในรีจิสเตอร์ AC แล้วจึงเรียกคำสั่ง RET ในคำสั่ง RET นี้การทำงานจะกลับสู่โปรแกรมหลัก และรีจิสเตอร์ FP กลับมาชี้ที่แอคทิเวชันเรคคอร์ดของโปรแกรมหลักเช่นเดิม ไม่มีการเปลี่ยนแปลงค่าของรีจิสเตอร์ AC แต่อย่างใด

5.9 การระบุระยะทางของการกระโดด

ในคำสั่งกระโดดทุกคำสั่ง ได้แก่ JMP, JMPL, JT, JTL, JF, JFL, JC, JCL, JNC, JNCL และ IJX มีตัวถูกดำเนินการที่ใช้ในการระบุระยะทางของการกระโดดนั้น ระยะทางของการกระโดดนี้จะถูกนำไปบวกกับค่าของรีจิสเตอร์ PC ขณะนั้น เพื่อใช้เป็นเลขที่อยู่ของคำสั่งที่จะกระทำถัดไป ค่าระยะทางนี้เป็นได้ทั้งจำนวนบวกและจำนวนลบ เพื่อให้ทำการกระโดดไปด้านหน้าและกระโดดไปด้านหลังได้

วิธีในการหาระยะกระโดด สำหรับใช้ในการสร้างตัวสร้างโค้ด (Code generator) สามารถหาได้โดย การนำเลขที่อยู่ของคำสั่งปลายทาง มาลบด้วยค่าของรีจิสเตอร์ PC ขณะที่กำลังกระทำคำสั่งกระโดดนั้น ซึ่งค่าของรีจิสเตอร์ PC นี้เท่ากับเลขที่อยู่ของคำสั่งกระโดดนั้นบวกหนึ่ง

ในกรณีที่คำสั่งกระโดดนั้นเป็นคำสั่งรูปแบบยาวที่ถูกการอัดคำสั่งแบ่งออกเป็น 2 ส่วน แยกใส่ไว้ในหน่วยความจำคนละตำแหน่ง ให้ใช้เลขที่อยู่ของคำสั่งส่วนที่เป็นบิตสูง (Most significant bit) ของคำสั่งดังกล่าวมาใช้ในการคำนวณหาระยะกระโดด

เนื่องจากระยะกระโดดที่ระบุในคำสั่งกระโดดนั้น ใช้เลขส่วนเติมเต็มสอง (Two's complement) ในการระบุค่า ทำให้ระยะกระโดดที่คำสั่งรูปแบบสั้นสามารถระบุได้อยู่ในช่วง -128 ถึง 127 หากระยะกระโดดไกลกว่าช่วงนี้ คำสั่งกระโดดดังกล่าวต้องใช้คำสั่งรูปแบบยาวแทน

5.10 ตัวอย่างโค้ดภาษาแอสเซมบลีของหน่วยประมวลผลนี้

ในส่วนสุดท้ายของบทนี้ จะเป็นการแสดงตัวอย่างโค้ดภาษาแอสเซมบลี (Assembly language) ของหน่วยประมวลผลนี้ ซึ่งโค้ดภาษาแอสเซมบลีนี้ จะได้รับการแปลงเป็นรหัสภาษาเครื่อง (Machine code) โดยแอสเซมเบลอร์ (Assembler) ก่อนถูกใส่ลงในหน่วยความจำส่วนโปรแกรม เพื่อให้อยู่ในรูปที่หน่วยประมวลผลสามารถทำงานได้

ในโค้ดภาษาแอสเซมบลีนี้ มีสัญลักษณ์ที่ใช้ในการประกาศตัวแปร หรือใช้บ่งบอกลักษณะการใช้งานของตัวแปรที่ตามหลังมา เพื่อให้แอสเซมเบลอร์นำตัวแปรต่างๆไปใช้งานได้อย่างถูกต้อง ความหมายของสัญลักษณ์เหล่านี้แสดงในตารางที่ 5.1

ตารางที่ 5.1 สัญลักษณ์ที่ใช้ในแอสเซมเบลอร์

สัญลักษณ์	คำอธิบาย
CONSTANT	ประกาศตัวแปรแบบค่าคงที่ ตัวแปรประเภทนี้ไม่สามารถเปลี่ยนค่าได้ ถูกเก็บในหน่วยความจำโปรแกรม โดยตัวแปรแบบค่าคงที่นี้ต้องเป็นตัวแปรส่วนกลาง (Global variable) เท่านั้น และถูกประกาศนอกโปรแกรมย่อย
ARRAY	ประกาศตัวแปรแถวลำดับ (Array) ตัวแปรประเภทนี้ต้องเป็นตัวแปรส่วนกลาง (Global variable) เท่านั้น และถูกประกาศนอกโปรแกรมย่อย
VAR	ประกาศตัวแปร ตัวแปรที่ประกาศนี้จะเป็นตัวแปรเฉพาะที่ (Local variable) หากประกาศในโปรแกรมย่อย หรือเป็นตัวแปรส่วนกลาง (Global variable) หากประกาศนอกโปรแกรมย่อย
FUNCTION	ประกาศเริ่มต้นโปรแกรมย่อย ชื่อโปรแกรมย่อยที่ประกาศตามหลังสัญลักษณ์นี้ต้องระบุพารามิเตอร์ที่รับมาจากโปรแกรมหลักด้วย
ENDFUNCTION	ประกาศสิ้นสุดโปรแกรมย่อย
:	ประกาศปลายทางของการกระโดด ชื่อปลายทางการกระโดดที่ตามหลังสัญลักษณ์ ถูกใช้ในคำสั่งกระโดดต่างๆในโปรแกรม

โปรแกรมตัวอย่างต่อไปนี้ เขียนด้วยโค้ดภาษาซี (C programming language) เป็นตัวอย่างใช้ประกอบในการอธิบายโค้ดภาษาแอสเซมบลีของหน่วยประมวลผลนี้ เพื่อช่วยให้ผู้อ่านสามารถทำความเข้าใจโปรแกรมที่เขียนด้วยโค้ดภาษาแอสเซมบลี

อนึ่ง ในการทดสอบการทำงานของวงจรหน่วยประมวลผลนี้ โปรแกรมที่ใช้ทดสอบถูกเขียนด้วยโค้ดภาษาแอสเซมบลีแบบเดียวกับที่แสดงในตัวอย่างนี้ โค้ดภาษาแอสเซมบลีนี้จะได้รับการแปลงเป็นรหัสภาษาเครื่องโดยแอสเซมเบลอร์ ที่พัฒนาควบคู่มากับวงจรหน่วยประมวลผลของงานวิจัยนี้

```
const int const0 = 0;
int gvar;
int num_array[10];

void main(){
        int i;
        gvar = 10;
        for(i = 0; i < gvar; i++){
                num_array[i] = const0;
        }
}
```

โปรแกรมตัวอย่างข้างต้น เขียนเป็นโค้ดภาษาแอสเซมบลีของหน่วยประมวลผลที่นำเสนอในงานวิจัยนี้ ได้ดังนี้

```
CONSTANT const0 0
VAR gvar
ARRAY num_array 10

FUNCTION main()
        VAR i
        VAR temp_var0
        LIT 10
        ST gvar
        LIT 0
        PUT i
```

```
:main_label0
        LD const0
        PUT temp_var0
        LIT num_array
        ADD i
        STV temp_var0
        LD gvar
        IJX main_label0 i
        RET
ENDFUNCTION
```

temp_var0 เป็นตัวแปรสำหรับเก็บค่าชั่วคราว ถูกเพิ่มเข้าไปในโปรแกรมเพื่อทำหน้าที่เก็บค่าที่จะทำการเขียนด้วยคำสั่ง STV