บทที่ 6

การศึกษาและทดสอบสมรรถนะเมื่อแปรขนาดคิวคำสั่ง

การพัฒนาระบบคอมพิวเตอร์ให้มีสมรรถนะการทำงานที่ดีขึ้น กระทำได้โดยอาศัยเทคโนโลยีที่เกิดขึ้นใหม่ เช่น เปลี่ยนไปใช้อุปกรณ์ที่ทำงานได้เร็วขึ้นแทน หรือปรับเพิ่มรูปแบบองค์กรภายในหน่วยประมวลผล เช่น กำหนดรีจิสเตอร์ใช้งานให้มีจำนวนมากขึ้น หรือเพิ่มวงจรทำงาน วิธีการที่นิยมใช้กันมากที่สุดวิธีหนึ่งคือ การประมวลผลคำสั่งแบบสายท่อ (Pipeline) เพราะเป็นวิธีการที่ทำให้ระบบสามารถประมวลผลคำสั่งในโปรแกรมได้รวดเร็วขึ้น โดยการกำหนดให้แต่ละคำสั่งแบ่งขั้นตอนปฏิบัติการเป็นหลายๆขั้นตอน แล้วให้หน่วยประมวลผลทำงานในขั้นตอนต่างๆนั้นไปพร้อมๆกัน

ภายในตัวแบบจำลองการทำงานของหน่วยประมวลผลต้นแบบ มีการประมวลผลคำสั่งในลักษณะนี้เกิดขึ้นโดยแบ่งเป็น 2 ขั้นตอนคือ การดึงคำสั่งและการปฏิบัติการคำสั่ง ซึ่งได้อธิบายไว้แล้วในบทที่ 4 วิธีการนี้ทำให้ระบบใช้งานบัสข้อมูลในการส่งผ่านข้อมูลได้อย่างเต็มที่ โดยในขณะที่บัสข้อมูลยังว่างอยู่ จะทำการดึงคำสั่งถัดไปจากหน่วยความจำเข้าไปเก็บในที่พักข้อมูล (buffer) หรือคิวคำสั่ง เมื่อหน่วยปฏิบัติการต้องการคำสั่งเพื่อทำงานต่อไป จึงไม่ต้องเสียเวลาในการรอให้หน่วยเชื่อมต่อบัสดึงคำสั่งมาจากหน่วยความจำ ทำให้ระบบสามารถปฏิบัติการคำสั่งได้เร็วขึ้น ดังนั้นแนวทางหนึ่งในการศึกษาผลกระทบต่อสมรรถนะการทำงานและการพัฒนาหน่วยประมวลผลให้มีประสิทธิภาพดีขึ้นคือ การปรับเปลี่ยนการจัดองค์กรภายในตัวแบบจำลอง โดยการพิจารณาขนาดคิวคำสั่งให้เหมาะสมต่อการทำงานภายในระบบ นั่นคือกำหนดจำนวนไบต์เป็นพารามิเตอร์หนึ่งที่ใช้ในการศึกษาและทดสอบสมรรถนะการทำงานของหน่วยประมวลผลต้นแบบนี้

สำหรับวิธีการที่ใช้เปรียบเทียบประสิทธิภาพการทำงานคือ การวัดเวลาที่ใช้ในการประมวลผลคำสั่งในโปรแกรมโดยตัวแบบจำลอง แล้วนำข้อมูลเวลาที่ตัวแบบจำลองใช้ในการปฏิบัติการกรณีที่ไม่มีคิวคำสั่งเปรียบเทียบกับกรณีที่มีคิวคำสั่งขนาดต่างๆ และนำผลลัพธ์ซึ่งเป็นค่าสมรรถนะสัมพัทธ์ (relative performance) ในแต่ละกรณีมาศึกษาเปรียบเทียบรวมทั้งได้นำข้อมูลการใช้งานคิวคำสั่งโดยเฉลี่ยมาพิจารณาด้วย

6.1 การศึกษาค่าสมรรถนะสัมพัทธ์เมื่อแปรขนาดคิวคำสั่ง

การปฏิบัติการคำสั่งภายในหน่วยประมวลผลจะเกิดขึ้น หลังจากที่หน่วยเชื่อมต่อบัสดึงคำสั่งจากโปรแกรมในหน่วยความจำแล้วส่งมายังหน่วยปฏิบัติการได้ครบถ้วน ในกรณีที่คำสั่งใช้ 1 - 2 ไบต์ หน่วยปฏิบัติการสามารถรับคำสั่งได้ครบถ้วนใน 1 รอบการทำงานของหน่วยความจำ ซึ่งได้กำหนดใช้เวลา 4 รอบสัญญาณนาฬิกา จากนั้นจึงเริ่มปฏิบัติการตามคำสั่งที่อ่านเข้ามาด้วยช่วงระยะเวลาหนึ่ง (สมมติให้ใช้เวลาปฏิบัติการคำสั่งนี้เท่ากับ 8 รอบสัญญาณนาฬิกา) ซึ่งขณะนี้เองที่หน่วยเชื่อมต่อบัสจะว่างจากการทำงานกับหน่วยความจำ ดังนั้นระบบสามารถใช้ประโยชน์จากหน่วยเชื่อมต่อบัสในช่วงนี้ได้ ดังที่ได้กล่าวไว้ในบทที่ 5 แล้วว่าคำสั่งหนึ่งๆประกอบด้วยจำนวนไบต์ 1 - 6 ไบต์ ถ้าคำสั่งถัดไปมีความยาว 6 ไบต์ หน่วยปฏิบัติการต้องรอให้หน่วยเชื่อมต่อบัสดึงคำสั่งมาให้จนกว่าจะครบ ซึ่งต้องใช้อย่างน้อย 3 รอบการทำงานของหน่วยความจำ นั่นคือใช้เวลา 12 รอบสัญญาณนาฬิกา จะเห็นว่า เวลาที่หน่วยประมวลผลใช้ในการดึงคำสั่งตั้งแต่คำสั่งแรกจนถึงคำสั่งถัดไปได้ครบสมบูรณ์ ต้องใช้เวลารวม 24 รอบสัญญาณนาฬิกา

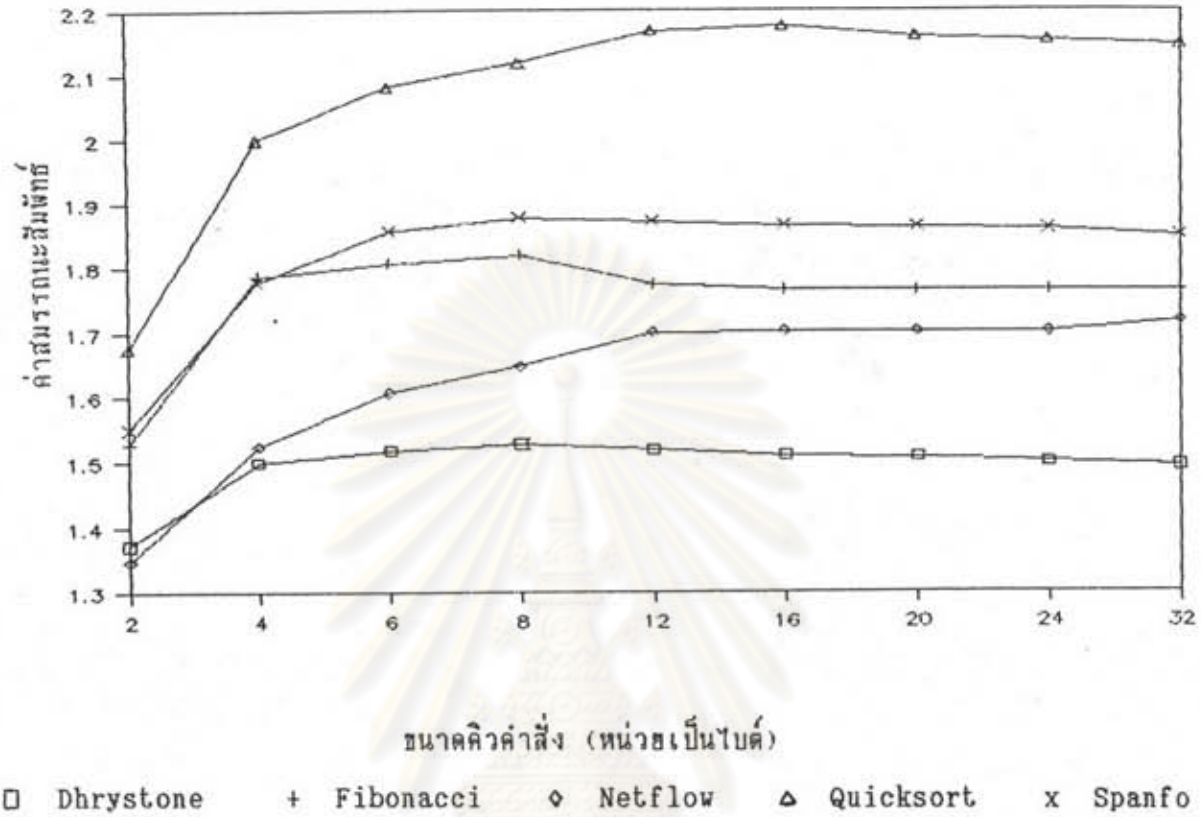
จากลักษณะการประมวลผลคำสั่งข้างต้นเป็นแบบลำดับขั้นที่ต่อเนื่องกัน คือต้องปฏิบัติการคำสั่งขณะนั้นให้เสร็จสิ้นก่อนแล้วจึงเริ่มดำเนินการรอบคำสั่งเครื่องต่อไปได้ จะพบว่าในบางเวลาหน่วยเชื่อมต่อบัสหยุดรออยู่โดยเปล่าประโยชน์ ดังนั้นถ้าต้องการเพิ่มสมรรถนะของหน่วยประมวลผลก็อาจกระทำได้โดยจัดให้ใช้ประโยชน์หน่วยเชื่อมต่อบัสอย่างเต็มที่ นั่นคือกำหนดให้มีคิวคำสั่งเพื่อให้หน่วยเชื่อมต่อบัสดึงคำสั่งถัดไปจากหน่วยความจำมาเก็บไว้ล่วงหน้าตามเงื่อนไขที่ได้กล่าวในบทที่ 4 แล้วนั้น จะช่วยให้หน่วยประมวลผลต้นแบบปฏิบัติการได้เร็วขึ้น เพราะในขณะที่หน่วยปฏิบัติการกำลังปฏิบัติการคำสั่ง หน่วยเชื่อมต่อบัสที่ว่างอยู่ก็ยังคงทำงานกับหน่วยความจำ โดยที่เวลาที่ใช้ปฏิบัติการคำสั่ง 8 รอบสัญญาณนาฬิกานั้น หน่วยเชื่อมต่อบัสสามารถดึงคำสั่งเข้ามาเก็บได้ถึง 4 ไบต์ และเมื่อหน่วยปฏิบัติการต้องการคำสั่งถัดไป จะรอให้หน่วยเชื่อมต่อบัสดึงคำสั่งเข้ามาอีก 2 ไบต์เท่านั้น ซึ่งเสียเวลาเพียง 4 รอบสัญญาณนาฬิกาก็ได้คำสั่งครบ 6 ไบต์ ดังนั้น เวลาทั้งหมดที่หน่วยประมวลผลต้องใช้จึงเท่ากับ 16 รอบสัญญาณนาฬิกา

ในการศึกษาและทดสอบเพื่อดูว่าควรจะต้องกำหนดขนาดคิวคำสั่งให้มีจำนวนไบต์เท่าใดจึงเหมาะสมต่อการทำงานของหน่วยประมวลผลต้นแบบ กระทำโดยการวัดเปรียบเทียบสมรรถนะของหน่วยประมวลผลต้นแบบ เมื่อแปรขนาดของคิวคำสั่งตั้งแต่ 2 - 32 ไบต์ โดยนำข้อมูลเวลาที่ใช้ในการปฏิบัติการโปรแกรมกรณีไม่มีคิวคำสั่ง ซึ่งกำหนดแทนด้วย t_o เปรียบเทียบกับเวลาที่ใช้ในการปฏิบัติการโปรแกรมกรณีที่มีคิวคำสั่งขนาดต่างๆกัน ซึ่งกำหนดแทนด้วย t_q โดยที่ q คือขนาดคิวคำสั่งมีหน่วยเป็นไบต์ นั่นคือค่าสมรรถนะสัมพัทธ์ เท่ากับ t_o / t_q และ

ตัวเลขผลลัพธ์จากการทำงานของตัวแบบจำลองได้แสดงไว้ในตารางที่ 6.1 แล้วและจากข้อมูลในตารางนี้ได้นำมาแสดงผลในรูปกราฟดังรูปที่ 6.1 เพื่อให้ผู้สนใจมองภาพของผลการศึกษาเปรียบเทียบนี้ได้ชัดเจนยิ่งขึ้น

ขนาดคิวคำสั่ง (ไบต์)	ค่าสมรรถนะสัมพัทธ์				
	Dhrystone	Fibonacci	Netflow	Quicksort	Spanfo
2	1.373656	1.528842	1.349090	1.675120	1.551340
4	1.499956	1.785261	1.524662	1.997634	1.777107
6	1.518541	1.806812	1.607239	2.080444	1.856569
8	1.528364	1.819400	1.648329	2.118061	1.878003
12	1.518924	1.773754	1.700605	2.167768	1.871289
16	1.509651	1.766220	1.701932	2.174819	1.864820
20	1.508471	1.765843	1.702106	2.159451	1.863329
24	1.500581	1.765736	1.701652	2.153156	1.859992
32	1.494709	1.765504	1.718958	2.146237	1.850493

ตารางที่ 6.1 แสดงค่าสมรรถนะสัมพัทธ์ของแต่ละโปรแกรมเมื่อแปรขนาดคิวคำสั่ง



รูปที่ 6.1 แสดงค่าสมรรถนะสัมพัทธ์ของแต่ละโปรแกรมเมื่อแปรขนาดคิวคำสั่ง

จากข้อมูลในตารางที่ 6.1 และกราฟรูปที่ 6.1 ที่ได้นี้ แสดงให้เห็นว่าการกำหนดมีคิวคำสั่งในระบบให้ผลดีต่อการทำงานของระบบในทุกกรณีศึกษา เพราะการมีคิวคำสั่งทำให้หน่วยเชื่อมต่อบัสดึงคำสั่งมาเก็บไว้ได้ และเมื่อหน่วยปฏิบัติการต้องการคำสั่งก็สามารถดึงมาจากคิวคำสั่งนั้นเอง ทำให้หน่วยปฏิบัติการไม่เสียเวลารอให้หน่วยเชื่อมต่อบัสดึงคำสั่งในทุกครั้งที่ต้องการ ดังนั้นในการพิจารณาเพิ่มจำนวนไบต์ในคิวคำสั่งเพื่อให้สามารถเก็บไบต์คำสั่งไว้ได้มากขึ้น น่าจะเป็นผลให้การทำงานของระบบมีประสิทธิภาพดีขึ้น ในแง่ของการช่วยลดเวลาที่หน่วยปฏิบัติการต้องรอคำสั่งจากหน่วยความจำลงอีก แต่จากกราฟได้แสดงให้เห็นว่า การเพิ่มขนาดคิวคำสั่งมากขึ้นไม่ได้ให้ค่าสมรรถนะสัมพัทธ์สูงขึ้นตามด้วยเสมอไป นั่นคือ ในขณะที่เพิ่มจำนวนไบต์ในคิวคำสั่ง ค่าสมรรถนะสัมพัทธ์ก็มีค่าสูงขึ้น แต่เมื่อเพิ่มจำนวนไบต์จนถึงขนาดหนึ่งแล้ว พบว่าค่าสมรรถนะสัมพัทธ์เริ่มมีค่าลดลง ทั้งนี้เนื่องจากภายในโปรแกรมมีคำสั่งโยกย้ายการควบคุมการทำงานซึ่งมีผลกระทบโดยตรงต่อคิวคำสั่งคือ ทำให้หน่วยเชื่อมต่อบัสต้องเริ่มต้นดึงคำสั่ง ณ ตำแหน่งใหม่ที่เปลี่ยนไปเข้ามาเก็บในคิวคำสั่ง นั่นคือคำสั่งเดิมที่มีอยู่ในคิวคำสั่งจะถูกแทนที่ไปโดยทันที ซึ่ง

ในช่วงเวลานี้ หน่วยปฏิบัติการจำเป็นต้องรอให้หน่วยเชื่อมต่อบัสดึงคำสั่งจากหน่วยความจำเข้ามาในคิวคำสั่ง แล้วจึงเริ่มปฏิบัติการคำสั่งต่อไปได้ ดังนั้นการเกิดเหตุการณ์เช่นนี้บ่อยครั้ง จะเป็นผลให้สมรรถนะของระบบลดลง

เมื่อได้พิจารณาลักษณะกราฟรูปที่ 6.1 จะพบว่าค่าสมรรถนะสัมพัทธ์ของกลุ่มโปรแกรม Dhrystone Fibonacci และ Spanfo ลดลงอย่างสังเกตได้ เมื่อขนาดคิวคำสั่ง มากกว่า 8 ไบต์ ในขณะที่ Quicksort และ Netflow จะให้ค่าสมรรถนะสัมพัทธ์ลดลงเมื่อจำนวนไบต์ของคิวคำสั่ง มากกว่า 16 ไบต์ และ 20 ไบต์ตามลำดับ ซึ่งการที่จำนวนไบต์ของคิวคำสั่งที่ให้ผลดีต่อการทำงานของโปรแกรม Dhrystone Fibonacci และ Spanfo มีค่าน้อยกว่าโปรแกรม Quicksort และ Netflow อธิบายได้ว่าเนื่องจากตัวเลขจำนวนไบต์ก่อนมีการกระโดดข้ามคำสั่งของโปรแกรมทั้ง 3 มีค่าน้อยกว่าตัวเลขของโปรแกรม Quicksort และ Netflow ประมาณ 1.6 เท่า แสดงว่ามีความจำเป็นในการใช้เนื้อที่เก็บคำสั่งล่วงหน้าไว้ในคิวคำสั่งน้อยกว่า ในขณะที่การทำงานของโปรแกรม Quicksort และ Netflow จำเป็นต้องใช้คิวคำสั่งที่มีจำนวนไบต์มากขึ้นเพื่อเพิ่มประสิทธิภาพการทำงานของระบบ เพราะจะช่วยให้หน่วยปฏิบัติการสามารถทำงานได้โดยไม่ต้องรอหน่วยเชื่อมต่อบัส

6.2 การศึกษาการใช้งานคิวคำสั่งเมื่อแปรขนาดคิวคำสั่ง

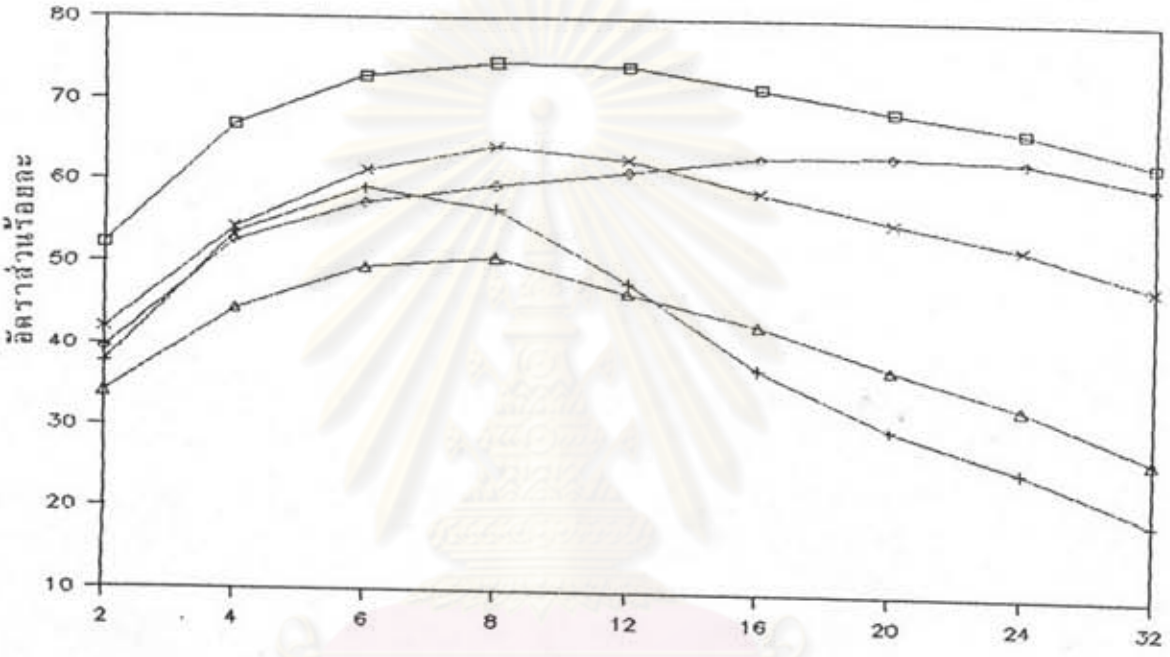
ในการใช้งานคิวคำสั่งภายในระบบ เราอาจพิจารณาได้จากจำนวนไบต์ที่มีอยู่ในคิวคำสั่งตลอดเวลาที่ปฏิบัติการโปรแกรม ซึ่งจากค่าเฉลี่ยที่ได้ในตารางที่ 6.2 ทำให้ทราบว่าในบางเวลามีคำสั่งอยู่ในคิวคำสั่งน้อยกว่าจำนวนไบต์ที่ระบบกำหนด ซึ่งมีสาเหตุจากกฎเกณฑ์การดึงคำสั่งเข้าในคิวคำสั่ง ดังที่ได้กล่าวในบทที่ 4 และผลจากการทำงานของคำสั่งเปลี่ยนการควบคุมในโปรแกรมที่ทำให้ระบบต้องเริ่มต้นกำหนดข้อมูลในคิวคำสั่งใหม่ เมื่อพิจารณาค่าข้อมูลในแต่ละโปรแกรมแล้ว พบว่า โปรแกรม Dhrystone ให้ค่าเฉลี่ยนี้เป็นอันดับสูงสุดในทุกขนาดคิวคำสั่ง รองลงมาคือ Spanfo Netflow Fibonacci และ Quicksort ตามลำดับ เมื่อจำนวนไบต์อยู่ในช่วง 2 - 12 ไบต์ แต่เมื่อจำนวนไบต์เพิ่มขึ้นเป็น 16 - 32 ไบต์กลับพบว่าอันดับค่าเฉลี่ยที่รองลงมาตามลำดับเป็นดังนี้ Netflow Spanfo Quicksort และ Fibonacci สำหรับ Netflow นั้น เมื่อกำหนดขนาดคิวคำสั่งเท่ากับ 4 และ 6 ไบต์ ปรากฏว่า Fibonacci ให้ค่าเฉลี่ยสูงกว่า Netflow

ขนาดคิวคำสั่ง (ไบต์)	ค่าเฉลี่ยจำนวนไบต์ที่รอในคิวคำสั่ง				
	Dhrystone	Fibonacci	Netflow	Quicksort	Spanfo
2	1.0447	0.7536	0.7884	0.6816	0.8383
4	2.6748	2.1398	2.1065	1.7746	2.1688
6	4.3713	3.5412	3.435	2.9686	3.6749
8	5.9631	4.5258	4.7553	4.0528	5.1381
12	8.8979	5.729	7.3433	5.5796	7.5282
16	11.4607	5.976	10.0962	6.8318	9.4118
20	13.7841	6.0297	12.6874	7.4929	11.0574
24	15.9898	6.0772	15.1084	7.9111	12.5676
32	20.1103	6.1608	19.2072	8.6221	15.2899

ตารางที่ 6.2 แสดงการใช้งานคิวคำสั่งโดยพิจารณาด้วยค่าเฉลี่ยจำนวนไบต์ที่รออยู่ในคิวคำสั่ง

การที่ได้ค่าเฉลี่ยออกมาในลักษณะนี้ เราอาจจะวิเคราะห์ได้ว่า กลุ่มโปรแกรมที่มีการทำงานแบบเวียนเกิด ซึ่งได้แก่ Fibonacci และ Quicksort มีการใช้งานคิวคำสั่งด้วยค่าเฉลี่ยที่น้อยกว่ากลุ่มโปรแกรมที่ไม่มีการทำงานในลักษณะดังกล่าว น่าจะเป็นผลมาจากภายในโปรแกรมมีการทำงานแบบเวียนเกิด ซึ่งทำให้มีอัตราส่วนการใช้งานคำสั่งย้ายการควบคุมมากกว่า

โปรแกรมที่ไม่มีการทำงานแบบนี้ โดยเฉลี่ยประมาณ 2.2 เท่า (ดูตารางที่ 5.2) และเมื่อมีการกระโดดข้ามคำสั่งไปทำงานที่ตำแหน่งอื่นบ่อยครั้ง ทำให้เกิดเหตุการณ์ที่ระบบต้องเริ่มต้นกำหนดข้อมูลในคิวคำสั่งใหม่ เป็นผลให้ค่าเฉลี่ยที่แสดงการใช้งานคิวคำสั่งมีค่าลดลงไป



รูปที่ 6.2 แสดงอัตราส่วน (ร้อยละ) การใช้งานคิวคำสั่งเมื่อแปรขนาดคิวคำสั่ง

รูปที่ 6.2 แสดงให้เห็นการใช้ประโยชน์คิวคำสั่งที่มีขนาดต่างๆกัน ในรูปอัตราส่วนร้อยละ โดยนำค่าจำนวนไบต์โดยเฉลี่ยที่รอในคิวคำสั่งหารด้วยจำนวนไบต์ในคิวคำสั่งตามที่กำหนดใช้ในระบบ จากรูปพบว่าทุกโปรแกรมให้อัตราส่วนการใช้งานคิวคำสั่งสูงขึ้นเมื่อจำนวนไบต์ในคิวคำสั่งเพิ่มขึ้น แต่หลังจากที่เพิ่มจำนวนไบต์จนถึงขนาดหนึ่งซึ่งให้อัตราส่วนการใช้งานคิวคำสั่งสูงสุดแล้ว หลังจากจุดนี้เมื่อเพิ่มจำนวนไบต์มากขึ้น อัตราส่วนการใช้งานคิวคำสั่งกลับลดลง ทั้งนี้เป็นผลจากการทำงานของคำสั่งย้ายการควบคุมในโปรแกรม โดยที่เมื่อมีการกระโดดข้ามคำสั่ง

หน่วยประมวลผลจะต้องเริ่มต้นบรรจุคำสั่งในคิวคำสั่งใหม่ เป็นผลให้ค่าเฉลี่ยจำนวนไบต์ที่รอในคิวคำสั่งลดลง และการเกิดเหตุการณ์เช่นนี้บ่อยครั้งจะเป็นตัวจำกัดไม่ให้ใช้ประโยชน์จากการมีคิวคำสั่งได้เต็มที่

และจากลักษณะกราฟที่ได้นี้แสดงให้เห็นว่าในการปฏิบัติการโปรแกรม Fibonacci โดยกำหนดคิวคำสั่งขนาด 6 ไบต์ จะได้อัตราส่วนการใช้งานคิวคำสั่งสูงสุดคือร้อยละ 59.02 ในขณะที่กลุ่มโปรแกรม Dhrystone Quicksort และ Spanfo ใช้งานคิวคำสั่งขนาด 8 ไบต์ ด้วยอัตราส่วนที่สูงสุดโดยเฉลี่ยร้อยละ 63.14 และสำหรับโปรแกรม Netflow มีการใช้งานคิวคำสั่งขนาด 20 ไบต์ด้วยอัตราส่วนสูงสุดคือ ร้อยละ 63.43 ซึ่งการที่ได้อัตราส่วนการใช้งานคิวคำสั่งสูงสุด ณ ขนาดคิวคำสั่งที่แตกต่างกันนี้ อาจเป็นผลจากจำนวนไบต์คำสั่งที่เข้าปฏิบัติการก่อนมีการกระโดดข้ามคำสั่ง นั่นคือบอกถึงความจำเป็นที่แต่ละโปรแกรมต้องใช้เนื้อที่สำหรับเก็บคำสั่งล่วงหน้า โดยพบว่าจำนวนไบต์คำสั่งเฉลี่ยที่เข้าปฏิบัติการก่อนมีการกระโดดข้ามคำสั่งภายในโปรแกรม Fibonacci เท่ากับ 6 ไบต์ สำหรับ Dhrystone Quicksort และ Spanfo มีค่าใกล้เคียงกัน ซึ่งโดยเฉลี่ยเท่ากับ 17 ไบต์และ Netflow มีค่าจำนวนไบต์คำสั่งดังกล่าวสูงสุดคือเท่ากับ 26 ไบต์ แต่การที่จำนวนไบต์ที่ให้อัตราส่วนการใช้งานคิวคำสั่งสูงสุดมีค่าต่ำกว่าหรือเท่ากับ จำนวนไบต์คำสั่งเฉลี่ยที่เข้าปฏิบัติการก่อนมีการกระโดดข้ามคำสั่งภายในโปรแกรม อาจเป็นผลเกี่ยวเนื่องมาจากการทำงานของหน่วยประมวลผลเมื่อมีการย้ายการทำงานในโปรแกรม ดังที่ได้อธิบายในข้างต้น

ขนาดคิวคำสั่ง	2	4	6	8	12	16	20	24	32
อัตราส่วน	41.06	54.32	59.97	61.08	58.46	54.72	51.05	48.04	43.37

ตารางที่ 6.3 แสดงอัตราส่วน (ร้อยละ) การใช้งานคิวคำสั่งโดยเฉลี่ย

จากตารางที่ 6.3 ซึ่งแสดงอัตราส่วนการใช้งานคิวคำสั่งโดยเฉลี่ยของทุกโปรแกรม พบว่าการเพิ่มจำนวนไบต์ในคิวคำสั่งจาก 2 ไปจนถึง 8 ไบต์ จะให้อัตราส่วนโดยเฉลี่ยที่มีค่าสูงขึ้น และหลังจากที่ได้อัตราส่วนโดยเฉลี่ยสูงสุดที่ขนาดคิวคำสั่งเท่ากับ 8 ไบต์แล้ว การเพิ่มจำนวนไบต์ในคิวคำสั่งก็ไม่ทำให้ค่าดังกล่าวสูงขึ้นแต่อย่างใด จึงกล่าวโดยสรุปในขณะนี้ได้ว่าจำนวนไบต์สูงสุดของคิวคำสั่งที่เหมาะสมกับการทำงานในทุกโปรแกรม คือ 8 ไบต์

เมื่อพิจารณาข้อมูลโดยรวมแล้วกล่าวได้ว่า การกำหนดขนาดคิวคำสั่งในช่วงตั้งแต่ 2 - 8 ไบต์ จะให้ค่าสมรรถนะสัมพัทธ์และอัตราส่วนการใช้งานคิวคำสั่งในลักษณะที่เพิ่มขึ้นโดยตลอด โดยที่ขนาดคิวคำสั่งที่ให้ค่าสมรรถนะสัมพัทธ์สูงสุดในช่วงที่พิจารณานี้ และให้อัตราส่วนการใช้งานคิวคำสั่งเฉลี่ยสูงสุดคือ ขนาดคิวคำสั่งที่มีจำนวนไบต์ 8 ไบต์ แต่จากการพิจารณาการเปลี่ยนแปลงของค่าสมรรถนะสัมพัทธ์ในทุกโปรแกรมเมื่อเพิ่มจำนวนไบต์ในคิวคำสั่งครั้งละ 2 ไบต์โดยเริ่มจากคิวคำสั่งขนาด 2 ไบต์ 4 ไบต์ 6 ไบต์ และ 8 ไบต์ พบว่าค่าสมรรถนะสัมพัทธ์เฉลี่ยเท่ากับ 1.50 1.72 1.77 และ 1.80 ตามลำดับ (ดูตารางที่ 6.1) และเมื่อพิจารณาอัตราส่วนการใช้งานคิวคำสั่งในทำนองเดียวกันพบว่าให้อัตราส่วนโดยเฉลี่ยเท่ากับ 41.06 54.32 59.97 และ 61.08 ตามลำดับ (ดูตารางที่ 6.3) จะสังเกตได้ว่าในขณะที่เพิ่มจำนวนไบต์ในคิวคำสั่งกลับมีการเปลี่ยนแปลงของค่าสมรรถนะสัมพัทธ์และอัตราส่วนการใช้งานคิวคำสั่งในลักษณะที่ลดลงโดยที่คิวคำสั่งขนาด 4 ไบต์จะให้ค่าการเปลี่ยนแปลงที่มากกว่าคิวคำสั่งขนาดอื่นๆ

6.3 สรุป

การกำหนดมีคิวคำสั่งช่วยให้การทำงานของระบบมีประสิทธิภาพมากขึ้น เพราะสามารถก่อให้เกิดการประมวลผลคำสั่งแบบสายท่อ ซึ่งทำให้หน่วยประมวลผลทำงานได้อย่างเต็มที่ และการดึงคำสั่งเก็บไว้ล่วงหน้ายังช่วยลดเวลาที่หน่วยประมวลผลต้องรอการทำงานของหน่วยความจำลงได้ ทำให้หน่วยประมวลผลมีสมรรถนะการทำงานที่ดีขึ้น ดังที่ได้เห็นจากผลการศึกษาทดสอบแต่การเพิ่มขนาดคิวคำสั่งมากเกินไป ไม่ช่วยให้สมรรถนะการทำงานของหน่วยประมวลผลดีขึ้นโดยตลอด ทั้งนี้เนื่องจากลักษณะโปรแกรมที่เข้าปฏิบัติการมีคำสั่งย้ายการควบคุม จึงทำให้ใช้ประโยชน์คิวคำสั่งได้ไม่เต็มที่ สำหรับในการศึกษาและทดสอบนี้ ได้พิจารณาเลือกขนาดคิวคำสั่งที่เหมาะสมกับการทำงานภายในระบบ คือคิวคำสั่งที่มีจำนวนไบต์ 6 ไบต์ เพราะให้ค่าสมรรถนะสัมพัทธ์และให้อัตราส่วนการใช้งานคิวคำสั่งเฉลี่ยสูงกว่าขนาดคิวคำสั่ง 4 ไบต์ และมีการเปลี่ยนแปลงของค่าทั้ง 2 มากกว่าขนาดคิวคำสั่ง 8 ไบต์