บทที่ 4
การออกแบบเครื่องเสมือน

บทนี้กล่าวถึงการออกแบบเครื่องเสมือนแบบแสตก ที่มีวงจรแปลคำสั่งทำหน้าที่แปลคำสั่งบนหน่วยประมวลผลแบบเรจิสเตอร์ C1 เครื่องเสมือนแบบแสตกถูกออกแบบให้ทำงานกับโปรแกรมที่อยู่ในรูปแบบชุดคำสั่งแบบแสตก SM1 ซึ่งรายละเอียดของหน่วยประมวลผล C1 และชุดคำสั่งแบบแสตก SM1 กล่าวไว้แล้วในบทที่ 3 และเพื่อความสะดวกขอเรียกหน่วยประมวลผลและชุดคำสั่งของหน่วยประมวลผล C1 นี้ว่า "หน่วยประมวลผล" และ "ชุดคำสั่งเดิม" (Native code) ตามลำดับ และเรียกชุดคำสั่งแบบแสตก SM1 ว่า "ชุดคำสั่งรหัสไบต์" (Bytecode)

การจัดเรียงเนื้อหาในบทนี้เริ่มจากการออกแบบและการทำงานโดยรวมของเครื่องเสมือน หลังจากนั้นในหัวข้อที่ 4.2 กล่าวถึงการปรับปรุงหน่วยประมวลผลให้สามารถรองรับการทำงานของวงจรแปลคำสั่ง สุดท้ายในหัวข้อที่ 4.3 กล่าวถึงการออกแบบวงจรแปลคำสั่ง

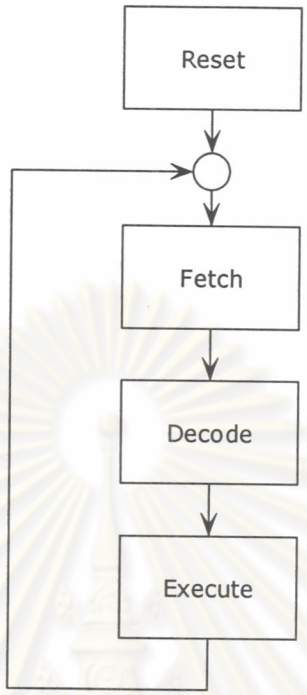
4.1 การออกแบบเครื่องเสมือนแบบแสตก

เครื่องเสมือนแบบแสตก (Virtual stack machine) คือเครื่องที่ทำงานเหมือนกับหน่วยประมวลผลแบบแสตก (Stack-based CPU) กล่าวคือมีสถาปัตยกรรมชุดคำสั่งเป็นชุดคำสั่งแบบแสตก ซึ่งใช้โครงสร้างข้อมูลแบบแสตกในการประมวลผล แต่เครื่องเสมือนไม่ได้เป็นการสร้างหน่วยประมวลผลแบบแสตกโดยตรง แต่เป็นการทำให้หน่วยประมวลผลแบบเรจิสเตอร์สามารถทำงานกับชุดคำสั่งแบบแสตกได้

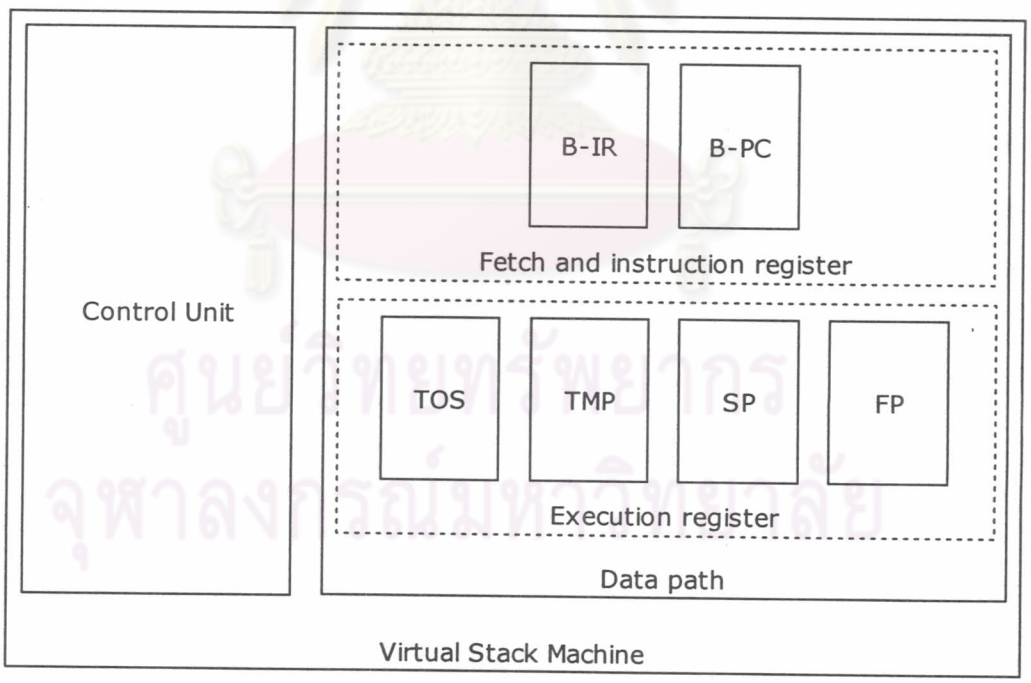
ในการอธิบายการออกแบบเครื่องเสมือนแบบแสตก จะอธิบายเปรียบเทียบกับการออกแบบหน่วยประมวลผลทั่วไป กล่าวคือหน่วยประมวลผลทั่วไปประกอบไปด้วยส่วนประกอบหลัก 2 ส่วนได้แก่หน่วยควบคุม (Control unit) และส่วนทางเดินข้อมูล (Data path) โดยที่หน่วยควบคุมทำหน้าที่ในการควบคุมการทำงานของหน่วยประมวลผลให้เป็นไปตามขั้นตอนดังแผนภาพสถานะในรูปที่ 4.1 เริ่มจากการอ่านคำสั่ง ถอดรหัสคำสั่ง และควบคุมให้ส่วนทางเดินข้อมูล (Data path) ซึ่งเป็นส่วนที่ทำหน้าที่กระทำการตามคำสั่งนั้นๆ

การออกแบบเครื่องเสมือนจึงแบ่งส่วนประกอบออกเป็นส่วนที่ทำหน้าที่ควบคุมและส่วนที่ทำหน้าที่เป็นทางเดินข้อมูลในการกระทำการ โดยส่วนที่ทำหน้าที่ควบคุมมีหน้าที่ควบคุมให้เครื่องเสมือนทำงานตามขั้นตอนดังรูปที่ 4.1 เช่นเดียวกับหน่วยควบคุมทั่วไป แต่ในส่วนทางเดินข้อมูลต้องออกแบบให้รองรับการทำงานกับชุดคำสั่งรหัสไบต์ ซึ่งจำเป็นต้องมีเรจิสเตอร์ที่ใช้ในการทำงาน

ด้วยกัน 2 ประเภทได้แก่เรจิสเตอร์ที่ใช้ในการอ่านและเก็บคำสั่ง และเรจิสเตอร์ที่ใช้ในการประมวลผลคำสั่งดังรูปที่ 4.2



รูปที่ 4.1 แผนภาพสถานะแสดงขั้นตอนการทำงานอย่างคร่าวๆ ของเครื่องเสมือน



รูปที่ 4.2 สถาปัตยกรรมเครื่องเสมือนแบบแสตก

เรจิสเตอร์ B-PC และ B-IR (Bytecode Program Register และ Bytecode Instruction Register) ซึ่งทำหน้าที่เก็บตำแหน่งคำสั่งรหัสไบต์และเก็บคำสั่งรหัสไบต์ที่อ่านมาตามลำดับ

ส่วนเรจิสเตอร์ที่ใช้ในการประมวลผลคำสั่งมีด้วยกัน 4 ตัวได้แก่เรจิสเตอร์ TOS, TMP, SP และ FP โดยหน้าที่ของเรจิสเตอร์เหล่านี้กล่าวไว้ในหัวข้อที่ 3.2

ปัญหาในการออกแบบเครื่องเสมือนแบบแสตกนั้นคือการสร้างเครื่องเสมือนแบบที่มีทำงานตามขั้นตอนดังรูปที่ 4.1 และมีโครงสร้างดังรูปที่ 4.2 จากหน่วยประมวลผล C1 เนื่องจากหน่วยประมวลผลไม่สามารถประมวลผลคำสั่งรหัสไบต์ได้โดยตรง ทำให้ต้องมีวงจรพิเศษในการแปลคำสั่งรหัสไบต์ให้อยู่ในรูปแบบที่หน่วยประมวลผลสามารถประมวลผลได้เสียก่อน ซึ่งก่อนอื่นจะขออธิบายการทำงานตามชุดคำสั่งรหัสไบต์โดยใช้ชุดคำสั่งเดิมเสียก่อน แล้วจึงกล่าวถึงแนวคิดที่ใช้ในการออกแบบเครื่องเสมือน

4.1.1 การทำงานตามชุดคำสั่งรหัสไบต์ด้วยชุดคำสั่งเดิม

โดยปกติแล้วไม่ว่าจะเป็นหน่วยประมวลผลชนิดใดต่างก็ต้องมีการทำงานพื้นฐานคือการอ่านคำสั่งและการเข้าถึงข้อมูลในหน่วยความจำ ต่างกันที่หลักการในการประมวลผล ดังนั้นการทำให้หน่วยประมวลผลแบบเรจิสเตอร์มีการทำงานแบบแสตกก็ย่อมทำได้ เนื่องจากการทำงานพื้นฐานที่จำเป็นสำหรับหน่วยประมวลผลแบบแสตกคือการเข้าถึงข้อมูลในแสตก ซึ่งหน่วยประมวลผลแบบเรจิสเตอร์สามารถทำได้ผ่านคำสั่งในกลุ่มเคลื่อนย้ายข้อมูลระหว่างเรจิสเตอร์และหน่วยความจำ เช่นคำสั่งโหลดและสโตร (Load and store instruction)

ในที่นี้ขอยกตัวอย่างการทำงานตามชุดคำสั่งรหัสไบต์ด้วยชุดคำสั่งเดิมโดยเรจิสเตอร์ TOS, TMP, SP และ FP ถูกกำหนดให้ใช้เรจิสเตอร์ภายในของหน่วยประมวลผลตามที่กล่าวไว้ในหัวข้อที่ 4.1.3

คำสั่ง ADD ของคำสั่งรหัสไบต์จะนำข้อมูลสองตัวบนสุดบนแสตกมาบวกกันแล้วเก็บผลลัพธ์ลงในแสตก สามารถอธิบายได้ด้วยชุดคำสั่งเดิมของหน่วยประมวลผล C1 ดังตารางที่ 4.1 ซึ่งจะขอเรียกว่า*ลำดับคำสั่งเดิม* (Native sequence) โดยในตารางที่ 4.1 มีการใช้แสตกแคชซึ่งข้อมูลบนสุดในแสตกถูกเก็บอยู่ในเรจิสเตอร์ TOS จึงเหลือการอ่านข้อมูลจากหน่วยความจำมาเพียงครั้งเดียว และผลลัพธ์จากการบวกไม่ต้องเก็บลงหน่วยความจำ แต่เก็บลงในเรจิสเตอร์ TOS

ตารางที่ 4.1 การอธิบายการทำงานของคำสั่งรหัสไบต์ ADD ด้วยชุดคำสั่งเดิม

ADD (Bytecode) description	RTL Description	Native Sequence
1. Pop the 2 top of stack	TMP ← MEM[SP+1];	LDW R1,1(R2)
2. Add them together	TMP ← TMP + TOS;	ADD R1,R0
3. Push the result back to stack	TOS ← TMP; SP ← SP+1;	MOV R0,R1 ADDI R2,1

คำสั่งรหัสไบต์บางคำสั่งมีตัวถูกดำเนินการ ดังนั้นการอธิบายการทำงานด้วยลำดับคำสั่งเดิมจำเป็นต้องส่งตัวถูกดำเนินการไปยังคำสั่งเดิมคำสั่งใดคำสั่งหนึ่งในลำดับในการประมวลผล เช่นคำสั่งรหัสไบต์ GET มีตัวถูกดำเนินการขนาด 8 บิตที่มีชื่อว่า LOCAL คำสั่งนี้จะนำค่าตัวแปรเฉพาะที่ที่อ้างอิงโดยตัวชี้กรอบ (FP) ตัวที่ระบุด้วยตัวถูกดำเนินการ LOCAL (FP+LOCAL) ไปเก็บไว้บนสุดของแสตก ซึ่งสามารถอธิบายการทำงานด้วยคำสั่งเดิมดังตารางที่ 4.2

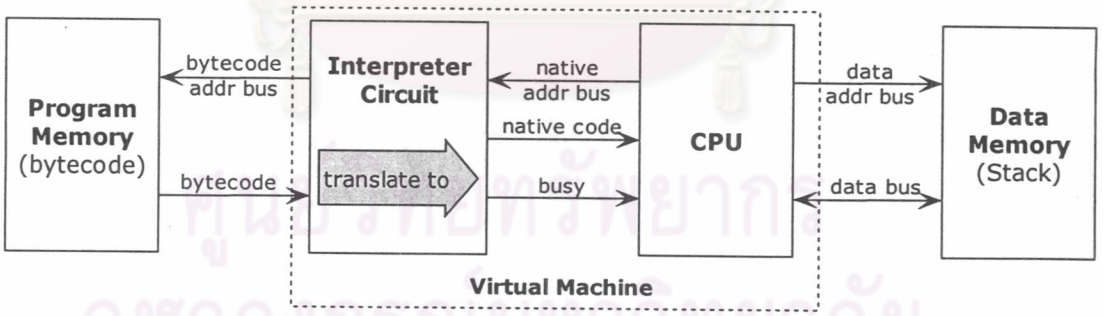
ตารางที่ 4.2 การอธิบายการทำงานของคำสั่งรหัสไบต์ GET ด้วยชุดคำสั่งเดิม

GET (Bytecode) description	RTL Description	Native Sequence
1. Back up TOS to stack	MEM[SP] ← TOS;	STW R0,R2(0)
2. Push the local variable	TOS ← MEM[FP+LOCAL];	LDW R0,LOCAL(R3)
3. Decrement SP	SP ← SP-1;	SUBI R2,1

คำสั่งรหัสไบต์ทั้งหมดสามารถอธิบายในรูปแบบของลำดับคำสั่งเดิมได้ทุกคำสั่ง โดยการทำงานตามคำสั่งรหัสไบต์ด้วยลำดับชุดคำสั่งเดิมทั้งหมดแสดงไว้ในภาคผนวก ง ตารางที่ ง.1

4.1.2 แนวคิดการทำงานของเครื่องเสมือน

จากปัญหาที่หน่วยประมวลผลไม่สามารถประมวลผลคำสั่งรหัสไบต์ได้โดยตรง จึงจำเป็นต้องมีส่วนที่ทำหน้าที่แปลคำสั่งรหัสไบต์ให้อยู่ในรูปแบบของลำดับคำสั่งเดิมก่อน อีกทั้งส่วนดังกล่าวยังต้องทำหน้าที่ควบคุมการทำงานของเครื่องเสมือนให้เป็นไปตามขั้นตอนดังรูปที่ 4.1 ซึ่งส่วนที่ทำหน้าที่ดังกล่าวนั้นคือวงจรแปลคำสั่ง (Interpreter circuit) ความสัมพันธ์ระหว่างวงจรแปลคำสั่งและหน่วยประมวลผลเป็นไปดังรูปที่ 4.3



รูปที่ 4.3 วงจรแปลคำสั่งและหน่วยประมวลผลในเครื่องเสมือน

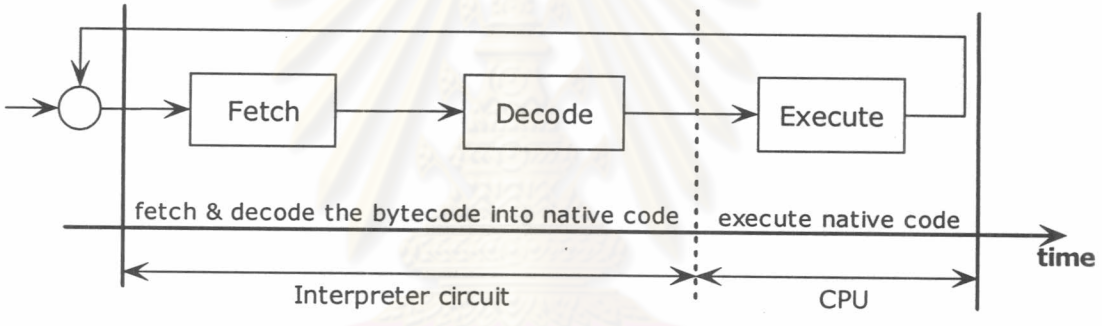
ดังที่กล่าวมาข้างต้นว่าการออกแบบเครื่องเสมือนแบบแสตกจะพิจารณาเปรียบเทียบกับการออกแบบหน่วยประมวลผลทั่วไปที่แบ่งส่วนประกอบออกเป็นหน่วยควบคุมและส่วนทางเดินข้อมูลดังอธิบายในหัวข้อที่ 4.1 ซึ่งการออกแบบเครื่องเสมือนจะกำหนดให้วงจรแปลคำสั่งสวมบทบาทเป็นหน่วยควบคุม และหน่วยประมวลผล C1 ทำหน้าที่เป็นส่วนทางเดินข้อมูล ดังนั้นวงจรแปลคำสั่งจึงหน้าที่อ่านคำสั่งรหัสไบต์มาแล้วถอดรหัสคำสั่งให้อยู่ในรูปแบบลำดับคำสั่งเดิม หลังจากนั้นจึงสลับการทำงานให้หน่วยประมวลผลเข้ามาอ่านคำสั่งจากลำดับคำสั่งเดิมเพื่อนำไป

ประมวลผลทีละคำสั่งจนครบทุกคำสั่งในลำดับ ก็จะเสร็จสิ้นการประมวลผลคำสั่งรหัสไบต์คำสั่งหนึ่ง โดยวงจรแปลคำสั่งใช้สัญญาณ busy ในการควบคุมการสลับการทำงานระหว่างวงจรแปลคำสั่งและหน่วยประมวลผล

การแบ่งหน้าที่การทำงานเช่นนี้มีประเด็นที่ต้องพิจารณา 2 ประการได้แก่

1. จังหวะการทำงาน และกลไกควบคุมการสลับการทำงานระหว่างวงจรแปลคำสั่งและหน่วยประมวลผล
2. กลไกในการอ่านคำสั่งรหัสไบต์ของวงจรแปลคำสั่ง และการเข้าถึงคำสั่งเดิมของหน่วยประมวลผล รวมไปถึงการส่งตัวถูกดำเนินการจากคำสั่งรหัสไบต์ไปยังคำสั่งเดิม

จังหวะการทำงานของเครื่องเสมือนแบบแสตกถูกแบ่งออกเป็นช่วงเวลาสำหรับวงจรแปลคำสั่งและหน่วยประมวลผลอย่างชัดเจน โดยสามารถเปรียบเทียบกับขั้นตอนการทำงานในรูปที่ 4.1 ได้ดังรูปที่ 4.4



รูปที่ 4.4 จังหวะการทำงานของระบบเครื่องเสมือน

วงจรแปลคำสั่งทำหน้าที่ควบคุมการสลับการทำงานของส่วนทั้งสองให้เป็นไปตามรูปที่ 4.4 จากขั้นตอนการทำงานของหน่วยประมวลผลที่อธิบายในหัวข้อที่ 3.3.2 รูปที่ 3.15 เมื่อหน่วยประมวลผลส่งสัญญาณร้องขอการอ่านคำสั่งไปยังหน่วยความจำ หน่วยประมวลผลจะรับคำสั่งเข้ามาเมื่อสัญญาณ busy = 0 ซึ่งแสดงว่าหน่วยความจำพร้อมส่งคำสั่ง จึงนำเอาสัญญาณ busy นี้มาใช้ควบคุมการสลับการทำงาน โดยเปรียบเทียบให้วงจรแปลคำสั่งทำหน้าที่เป็นหน่วยความจำโปรแกรมให้กับหน่วยประมวลผลและส่งสัญญาณ busy แทนหน่วยความจำ ในระหว่างที่วงจรแปลคำสั่งอ่านและถอดรหัสคำสั่งอยู่นั้น วงจรแปลคำสั่งจะให้สัญญาณ busy = 1 จนเมื่อถอดรหัสคำสั่งเสร็จสิ้นจึงให้สัญญาณ busy = 0 เพื่อให้หน่วยประมวลผลเข้ามาอ่านคำสั่งในลำดับคำสั่งเดิมภายในวงจรแปลคำสั่งได้

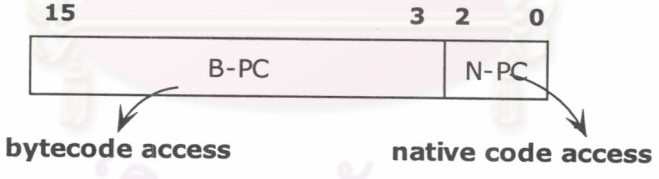
ประเด็นที่ต้องพิจารณาต่อมาคือกลไกการอ่านคำสั่งรหัสไบต์และการเข้าถึงคำสั่งเดิมของหน่วยประมวลผล จากรูปที่ 4.2 การอ่านคำสั่งของวงจรแปลคำสั่งจำเป็นต้องมีเรจิสเตอร์ 2 ตัว

ได้แก่เรจิสเตอร์ B-PC สำหรับอ้างอิงถึงตำแหน่งคำสั่งรหัสไบต์ปัจจุบัน และเรจิสเตอร์ B-IR สำหรับเก็บคำสั่งที่อ่านมาได้เพื่อนำมาประมวลผล

การออกแบบวงจรแปลคำสั่ง เรจิสเตอร์ B-IR เป็นเรจิสเตอร์พิเศษภายในวงจรแปลคำสั่ง เนื่องจากวงจรแปลคำสั่งจำเป็นต้องใช้คำสั่งรหัสไบต์ในการถอดรหัส แต่สำหรับเรจิสเตอร์ B-PC นั้นถ้าให้เป็นเรจิสเตอร์ภายในเช่นเดียวกับ B-IR แล้วจะทำให้เกิดปัญหากับการทำงานกับคำสั่งรหัสไบต์ CALL เนื่องจากการเรียกโปรแกรมย่อยต้องนำค่า B-PC ไปเก็บในแสตกเพื่อใช้ในเวลากลับจากโปรแกรมย่อย แต่เนื่องจากวงจรแปลคำสั่งไม่สามารถเข้าถึงหน่วยความจำข้อมูล (พิจารณาจากรูปที่ 4.3) ทำให้ไม่สามารถนำค่า B-PC ไปเก็บในแสตกได้

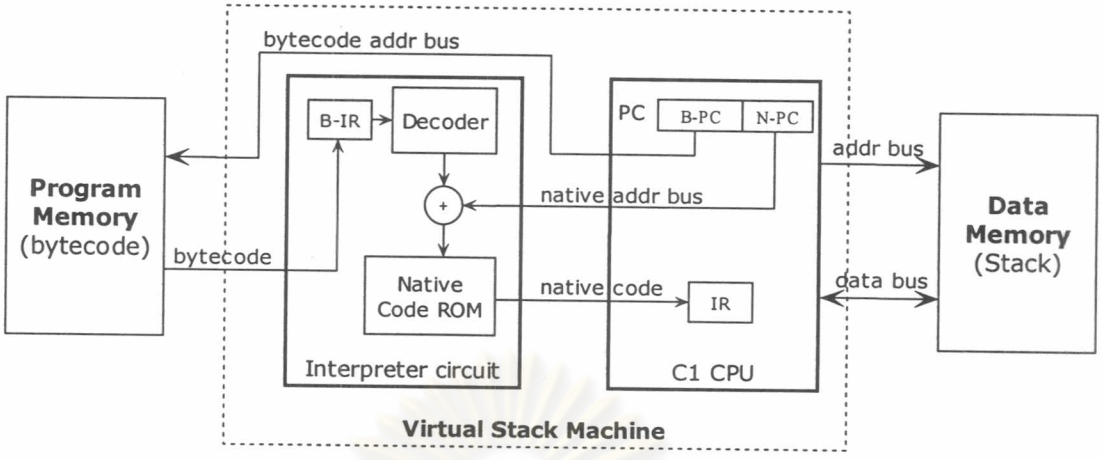
นอกจากนั้นในช่วงกระทำการที่หน่วยประมวลผลต้องเข้าไปอ่านคำสั่งเดิมในลำดับคำสั่งเดิมที่ได้จากการถอดรหัสของวงจรแปลคำสั่ง หน่วยประมวลผลจำเป็นต้องมีการอ้างอิงตำแหน่งว่าต้องการคำสั่งใดในลำดับผ่านเรจิสเตอร์ PC ด้วย

ดังนั้นในงานวิจัยนี้จึงได้ออกแบบให้เรจิสเตอร์ PC ที่อยู่ภายในหน่วยประมวลผลสามารถอ้างถึงได้ทั้งคำสั่งรหัสไบต์ในหน่วยความจำโปรแกรมและคำสั่งเดิมจากวงจรแปลคำสั่งโดยการออกแบบให้เรจิสเตอร์ PC แบ่งออกเป็น 2 ส่วนได้แก่ส่วนที่ใช้เป็น B-PC และส่วนที่ใช้ในการอ้างถึงคำสั่งในลำดับคำสั่งเดิม ซึ่งจะขอเรียกส่วนนี้ว่า N-PC ดังรูปที่ 4.5 โดยเรจิสเตอร์ B-PC ใช้บิตที่ 3 ถึงบิตที่ 15 และ N-PC จะใช้บิตที่ 0 ถึงบิตที่ 2 ของเรจิสเตอร์ PC ตามลำดับ โดยขนาดของ N-PC ได้มาจากการพิจารณาจำนวนคำสั่งในลำดับคำสั่งเดิมที่มากที่สุดซึ่งมีขนาดไม่เกิน 8 คำสั่ง



รูปที่ 4.5 โครงสร้างของเรจิสเตอร์ PC ที่ใช้ในเครื่องเสมือนแบบแสตก

เพื่อให้เรจิสเตอร์ PC สามารถอ้างอิงคำสั่งรหัสไบต์ได้ ต้องปรับปรุงให้วงจรแปลคำสั่งสามารถควบคุมการเพิ่มลดค่าใน B-PC ได้โดยตรง โดยที่หน่วยประมวลผลยังคงสามารถจัดการกับค่าใน PC ได้ตามปกติ โดยการสลับจังหวะการทำงานกันระหว่างวงจรแปลคำสั่งและหน่วยประมวลผลดังรูปที่ 4.4 เป็นกลไกที่สำคัญในการควบคุมค่าใน PC ทุกครั้งที่วงจรแปลคำสั่งอ่านคำสั่งรหัสไบต์แต่ละไบต์จะต้องมีการเพิ่มค่า B-PC เพื่อชี้คำสั่งในไบต์ถัดไป และเมื่อถึงช่วงที่หน่วยประมวลผลทำงานกับคำสั่งเดิมที่ผ่านการแปลคำสั่งมาแล้ว หน่วยประมวลผลจำเป็นต้องใช้ N-PC ในการอ้างถึงตำแหน่งในลำดับคำสั่งเดิม ทุกครั้งที่มีการอ่านคำสั่งเดิม ค่าใน N-PC จะมีค่าเพิ่มขึ้นเพื่ออ้างถึงคำสั่งเดิมในลำดับถัดไป



รูปที่ 4.6 โครงสร้างในการอ่านคำสั่งรหัสไบต์และกลไกในการเข้าถึงลำดับคำสั่งเดิมของเครื่องเสมือนแบบแสตก

ลำดับคำสั่งเดิมทั้งหมดที่อธิบายการทำงานของคำสั่งรหัสไบต์แต่ละคำสั่งถูกจัดเก็บในหน่วยความจำภายในวงจรแปลคำสั่งที่เรียกว่า*รอมคำสั่งเดิม* (Native Code ROM) และเมื่อวงจรแปลคำสั่งอ่านคำสั่งรหัสไบต์แล้วจะถอดรหัสคำสั่งออกมาเป็นตำแหน่งแรกสุดของคำสั่งในลำดับคำสั่งเดิมในรอมคำสั่งเดิม เมื่อถึงช่วงที่หน่วยประมวลผลทำงานก็จะอ่านคำสั่งเดิม โดยอ่านคำสั่งในตำแหน่งที่ N-PC จากตำแหน่งแรกสุดของลำดับที่วงจรแปลคำสั่งถอดรหัสไว้ หรือกล่าวได้ว่าตำแหน่งที่อ่านเป็นผลจากการบวกกันระหว่างตำแหน่งแรกที่ได้จากการถอดรหัสกับค่า N-PC ดังแสดงในรูปที่ 4.6 นอกจากนั้นสิ่งที่ได้จากการถอดรหัสคำสั่งอีกคือตำแหน่งของคำสั่งเดิมในลำดับที่ต้องถูกแทรกตัวถูกดำเนินการ ซึ่งรายละเอียดของการแทรกตัวดำเนินการกล่าวไว้ในหัวข้อที่ 4.3.2

นอกจากนั้นการจัดการกับเรจิสเตอร์ PC ยังต้องคำนึงถึงการกระโดด (Jump) โดยการประมวลผลคำสั่งรหัสไบต์มีการกระโดด 2 แบบคือการกระโดดที่เกิดจากคำสั่งรหัสไบต์ (จากคำสั่ง JMP, JT, JF และ CALL) ที่ทำกับเรจิสเตอร์ B-PC และการกระโดดข้ามลำดับคำสั่งเดิมในกรณีเกิดการตัดสินใจภายในลำดับคำสั่งเดิม (ลำดับคำสั่งเดิมที่อธิบายการทำงานของคำสั่งรหัสไบต์ LT และ EQ) ที่ทำกับ N-PC การเลือกส่วนที่เกิดการเปลี่ยนแปลงภายในเรจิสเตอร์ PC นี้ถูกควบคุมโดยวงจรแปลคำสั่งเช่นกัน รายละเอียดแสดงไว้ในหัวข้อที่ 4.2

4.1.3 การจัดสรรทรัพยากรในการประมวลผลคำสั่งรหัสไบต์

การประมวลผลคำสั่งรหัสไบต์มีเรจิสเตอร์ 4 ตัว โดยใช้เรจิสเตอร์ 2 ตัวสำหรับจัดการแสตกในหน่วยความจำข้อมูลได้แก่เรจิสเตอร์ SP (Stack pointer) สำหรับอ้างถึงข้อมูลบนสุดของแสตก และเรจิสเตอร์ FP (Frame pointer) สำหรับอ้างถึงแอคทิเวชันเรคคอร์ด (Activation

record) ของโปรแกรมย่อยปัจจุบัน นอกจากนั้นเครื่องเสมือนนี้ทำงานโดยใช้แสตกแคชชิ่ง มีเรจิสเตอร์ TOS สำหรับเก็บข้อมูลตัวบนสุดในแสตกไว้ในเรจิสเตอร์แทนการเก็บลงหน่วยความจำ และสุดท้ายการประมวลผลคำสั่งรหัสไบต์ต้องมีเรจิสเตอร์ TMP สำหรับใช้เก็บข้อมูลชั่วคราวในระหว่างการประมวลผล

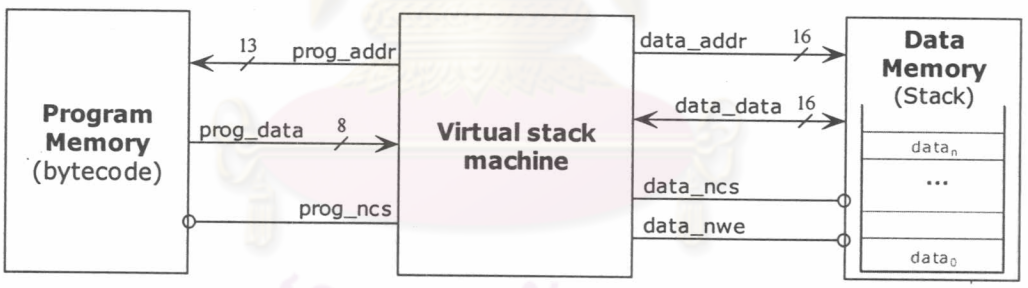
เรจิสเตอร์ทั้ง 4 ตัวของเครื่องเสมือนแบบแสตกนี้ใช้เรจิสเตอร์ทั้ง 4 ตัวของหน่วยประมวลผลดังตารางที่ 4.3

ตารางที่ 4.3 ความสัมพันธ์ระหว่างเรจิสเตอร์ของหน่วยประมวลผลกับเครื่องเสมือน

Stack machine register	C1 register
TOS	R0
TMP	R1
SP	R2
FP	R3

4.1.4 การออกแบบเครื่องเสมือน

แผนภาพบล็อกของระบบฝังตัวที่ใช้เครื่องเสมือนเป็นตัวควบคุมแสดงได้ดังรูปที่ 4.7 โดยเครื่องเสมือนอ่านคำสั่งรหัสไบต์จากหน่วยความจำโปรแกรม และประมวลผลคำสั่งนั้นโดยใช้โครงสร้างข้อมูลแสตกเป็นหลัก ซึ่งใช้หน่วยความจำข้อมูลทำหน้าที่เป็นแสตก สัญญาณต่างๆ ของเครื่องเสมือนแบบแสตกมีหน้าที่การทำงานดังตารางที่ 4.4



รูปที่ 4.7 แผนภาพบล็อกของระบบฝังตัวที่ใช้เครื่องเสมือนแบบแสตก

ตารางที่ 4.4 หน้าที่การทำงานของสัญญาณต่างๆ ของเครื่องเสมือนแบบแสตก

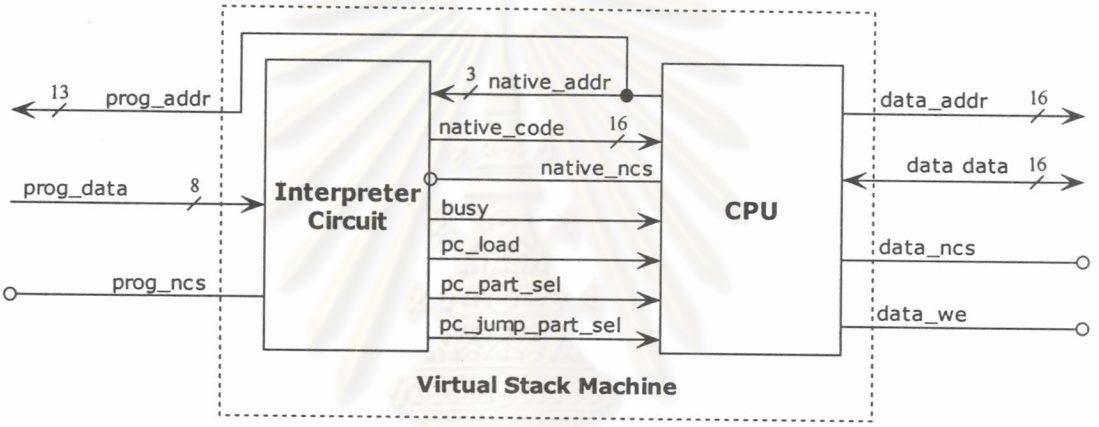
Port Name	Direction	Size	Description
prog_addr	Output	13	Program address bus
prog_data	Input	8	Program bytecode bus
prog_ncs	Output	1	Program memory chip enable (*active low*)
data_addr	Inout	16	Data address bus
data_data	Output	16	Data bus
data_ncs	Output	1	Data memory chip enable (*active low*)
data_nwe	Output	1	Data write enable (active low)

สัญญาณต่างๆ ของเครื่องเสมือนแบบแสตกมีไว้เพื่อเข้าถึงคำสั่งและข้อมูลในหน่วยความจำโปรแกรมและหน่วยความจำข้อมูล เมื่อเครื่องเสมือนต้องการอ่านคำสั่งให้สัญญาณ

prog_ncs = 0 เพื่อให้หน่วยความจำโปรแกรมทำงาน จากนั้นหน่วยความจำโปรแกรมจะส่งคำสั่งในตำแหน่งที่ระบุโดยบัส prog_addr กลับมาบนบัส prog_data ให้กับเครื่องเสมือน

ส่วนการเข้าถึงแสตกและข้อมูลอื่นในหน่วยความจำข้อมูลนั้น จะร้องขอผ่านสัญญาณ data_ncs และ data_nwe โดยสัญญาณ data_ncs จะเป็นสัญญาณกระตุ้นการทำงานของหน่วยความจำ และสัญญาณ data_nwe เป็นสัญญาณระบุการอ่าน/เขียนข้อมูล ถ้าสัญญาณ data_nwe = 0 จะเป็นการเขียนข้อมูล ส่วนถ้า data_nwe = 1 จะเป็นการอ่านข้อมูล การเข้าถึงข้อมูลจะเข้าถึงข้อมูลในตำแหน่งที่ถูกระบุในบัส data_addr โดยมี data_data เป็นบัสข้อมูล

ส่วนประกอบภายในของเครื่องเสมือนเป็นไปตามแผนภาพบล็อกในรูปที่ 4.8 โดยแสดงรายละเอียดของสัญญาณต่างๆ ไว้ในตารางที่ 4.5



รูปที่ 4.8 ส่วนประกอบของเครื่องเสมือนแบบแสตก

ตารางที่ 4.5 สัญญาณระหว่างวงจรแปลคำสั่งและหน่วยประมวลผล

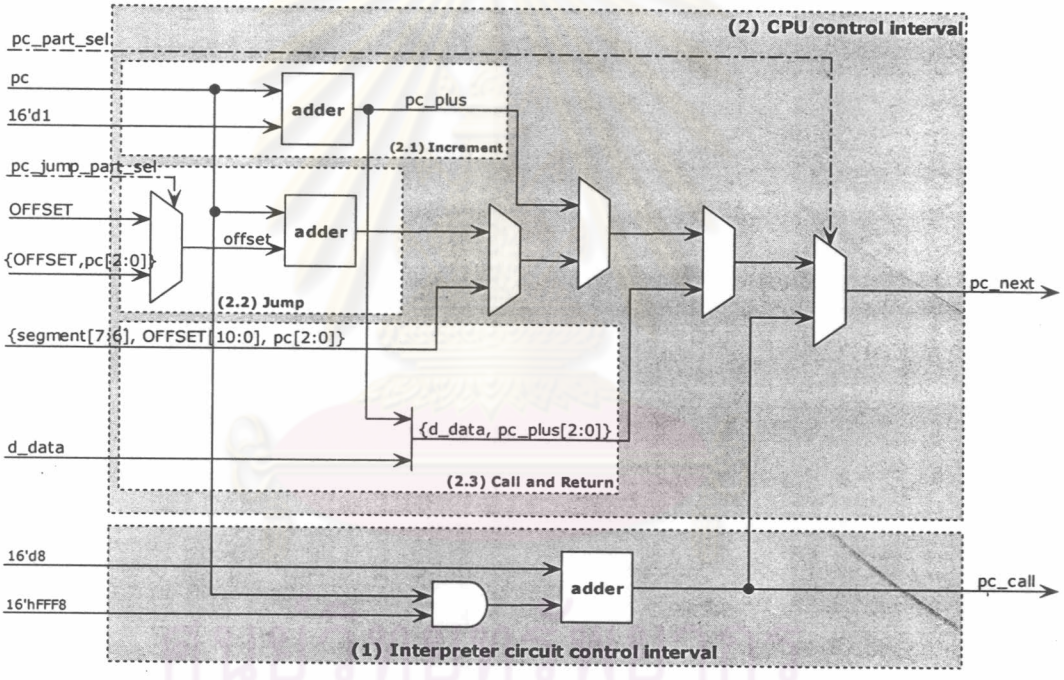
Port name	Size	Description
native_addr	3	Native address bus
native_code	16	Native code bus
native_ncs	1	Native code request (*active low*)
busy	1	Interpreter circuit busy status
pc_load	1	PC increment control signal
pc_part_sel	1	Select PC portion to increment
pc_jump_part_sel	1	Select PC portion to jump

สัญญาณ native_addr, native_code และ native_ncs เป็นสัญญาณที่ใช้ในการเข้าถึงคำสั่งเดิมในวงจรแปลคำสั่ง โดย native_addr เป็นบัสที่ต่อมาจากเรจิสเตอร์ PC ส่วนที่เป็น N-PC หน่วยประมวลผลจะส่งสัญญาณ native_ncs = 0 ไปยังวงจรแปลคำสั่งเพื่อร้องขอคำสั่งเดิมในลำดับคำสั่งเดิม เมื่อวงจรแปลคำสั่งพร้อมก็จะส่งคำสั่งเดิมกลับผ่านบัส native_code โดยจะส่งคำสั่งเดิมในตำแหน่งที่ระบุด้วยบัส native_addr ที่อ้างถึงลำดับของคำสั่งเดิม

สัญญาณ busy เป็นสัญญาณที่วงจรแปลคำสั่งใช้ในการควบคุมการสลับการทำงานระหว่างวงจรแปลคำสั่งและหน่วยประมวลผล ส่วนสัญญาณที่เหลือจะเป็นสัญญาณเกี่ยวกับการควบคุมค่าเรจิสเตอร์ PC ภายในหน่วยประมวลผล ซึ่งกล่าวไว้ในหัวข้อที่ 4.2

4.2 การปรับปรุงหน่วยประมวลผล

การปรับปรุงหน่วยประมวลผลให้รองรับการทำงานกับวงจรแปลคำสั่งคือการแก้ไขวงจรถอดรหัสค่า PC (PC Decoder, ***c.2.6***) ซึ่งเป็นวงจรเชิงผสมที่ทำหน้าที่ในการประเมินค่าถัดไปที่จะถูกจัดเก็บในเรจิสเตอร์ PC โดยมีสัญญาณที่ใช้ในการควบคุมเพิ่มขึ้นสองสัญญาณได้แก่สัญญาณ pc_part_sel และ pc_jump_part_sel ที่ส่งมาจากหน่วยควบคุมของวงจรแปลคำสั่งสัญญาณ pc_part_sel เป็นสัญญาณเลือกช่วงในการควบคุมระหว่างช่วงที่วงจรแปลคำสั่งควบคุม และช่วงที่หน่วยประมวลผลควบคุม



รูปที่ 4.9 วงจรถอดรหัสค่าเรจิสเตอร์ PC ที่ปรับปรุงแล้ว

เรจิสเตอร์ PC ถูกวงจรแปลคำสั่งควบคุมในเวลาที่เกิดการอ่านคำสั่งรหัสไบต์ โดยทุกๆ ครั้งที่มีการอ่านคำสั่งรหัสไบต์ 1 ไบต์ ค่าในเรจิสเตอร์ PC ในส่วนที่อ้าง B-PC จะมีค่าเพิ่มขึ้น 1 และทุกครั้งที่มีการอ่านคำสั่งรหัสไบต์จะต้องตั้งค่า N-PC ให้เท่ากับ 0 เพื่อเป็นการเริ่มต้นการอ่านลำดับคำสั่งเดิมใหม่ตั้งแต่ต้น (หรืออาจกล่าวได้ว่าทำให้ค่า PC เพิ่มขึ้น 8 นั่นเอง) ดังรูปที่ 4.9 ในส่วนของวงจรถอดรหัสค่า PC (อ้างในรูปส่วนที่ 1) การเพิ่มของ PC ในช่วงเวลาที่วงจรแปลคำสั่งควบคุมอยู่ โดยค่า PC จะเท่ากับ

```
PC ← ( PC & 16'hFFF8 ) + 16'h0008;
```

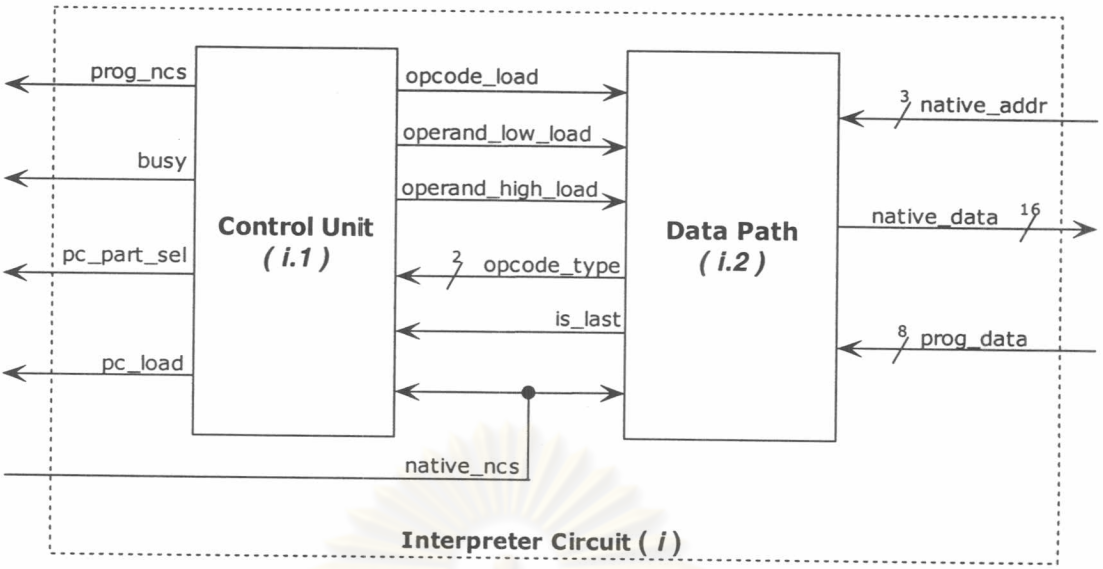
ส่วนช่วงเวลาที่หน่วยประมวลผลควบคุมเรจิสเตอร์ PC ใช้วงจรดังรูปที่ 4.9 (ส่วนที่ 2) โดยจะมีการควบคุมค่า PC ดังนี้

1. ให้ค่า N-PC เพิ่มขึ้น 1 เพื่อใช้ในการอ้างถึงคำสั่งในลำดับคำสั่งเดิมคำสั่งถัดไป (รูปที่ 4.9-2.1)
2. เกิดการกระโดด (รูปที่ 4.9-2.2) ได้ 2 แบบเลือกจากสัญญาณ pc_jump_part_sel ที่มาจากหน่วยควบคุมของวงจรแปลคำสั่ง
   2.1 การกระโดดเพื่อข้ามลำดับการทำงานในลำดับคำสั่งเดิม
   2.2 การกระโดดที่เป็นผลมาจากคำสั่งรหัสไบต์คำสั่ง JMP, JT และ JF
3. เกิดการเรียกโปรแกรมย่อย และเรียกกลับจากโปรแกรมย่อยของคำสั่งรหัสไบต์ CALL, RET และ RETV ซึ่งจะกระทำกับ B-PC ในส่วนที่เป็นการอ้างถึงหน่วยความจำโปรแกรมเท่านั้น (รูปที่ 4.9-2.3)

4.3 การออกแบบวงจรแปลคำสั่ง

วงจรแปลคำสั่งทำหน้าที่ในการอ่านคำสั่งรหัสไบต์จากหน่วยความจำโปรแกรม แล้วถอดรหัสคำสั่งเหล่านั้นให้อยู่ในรูปแบบของลำดับคำสั่งเดิม และควบคุมให้หน่วยประมวลผลเข้ามาอ่านคำสั่งจากลำดับคำสั่งเดิมที่ถอดรหัสได้

โครงสร้างของวงจรแปลคำสั่ง (Interpreter circuit, ***i***) ประกอบไปด้วยหน่วยควบคุม (Control unit, ***i.1***) และส่วนเส้นทางเดินของข้อมูล (Data path, ***i.2***) ดังแผนภาพในรูปที่ 4.10 หน่วยควบคุมทำหน้าที่ในการควบคุมการทำงานของวงจรแปลคำสั่งทั้งหมด ตั้งแต่การอ่านคำสั่งรหัสไบต์ ควบคุมการไหลของข้อมูลในส่วนทางเดินข้อมูลในการถอดรหัสคำสั่ง รวมไปถึงการควบคุมจังหวะการทำงานระหว่างวงจรแปลคำสั่งและประมวลผลกลาง ส่วนเส้นทางเดินข้อมูลเป็นวงจรที่ใช้ในการถอดรหัสคำสั่ง



รูปที่ 4.10 แผนภาพบล็อกส่วนประกอบของวงจรแปลคำสั่ง

4.3.1 รายละเอียดหน่วยควบคุม

หน่วยควบคุม ***(i.1)*** เป็นส่วนที่ทำหน้าที่ควบคุมการทำงานทั้งหมดของวงจรแปลคำสั่ง และการควบคุมจังหวะการทำงานระหว่างวงจรแปลคำสั่งและหน่วยประมวลผล ซึ่งหน่วยควบคุมจะมีสัญญาณต่างๆ ดังตารางที่ 4.6 และแผนภาพสถานะของหน่วยควบคุมแสดงได้ดังรูปที่ 4.11

ตารางที่ 4.6 รายละเอียดสัญญาณของหน่วยควบคุม

Signal name	Direction	Description
is_last	Input	Indicated to accessing of the last native code sequence
opcode_type	Input	Type of bytecode's opcode
native_ncs	Input	Native code request (*active low*)
prog_ncs	Output	Program memory chip enable (*active low*)
busy	Output	Interpreter circuit busy status
pc_load	Output	PC increment control signal
pc_part_sel	Output	Select PC portion to increment
pc_jump_part_sel	Output	Select PC portion to jump
opcode_load	Output	B-IR's opcode load signal
operand_high_load	Output	B-IR's operand high byte load signal
operand_low_load	Output	B-IR's operand low byte load signal

หน่วยควบคุมมีสัญญาณอินพุต 3 สัญญาณได้แก่สัญญาณ is_last, opcode_type และ native_ncs สัญญาณ is_last ทำหน้าที่บ่งชี้ว่าหน่วยประมวลผลได้อ่านคำสั่งเดิมในลำดับสุดท้ายไปประมวลผลแล้ว ส่วน native_ncs เป็นสัญญาณที่หน่วยประมวลผลใช้ร้องขอในการอ่านคำสั่งเดิม โดย is_last และ native_ncs เป็นตัวบ่งชี้ของหน่วยควบคุมในการเริ่มอ่านคำสั่งรหัสไบต์คำสั่งใหม่ ส่วน opcode_type บ่งชี้ถึงประเภทของคำสั่งเพื่อใช้ในการอ่านคำสั่งว่าจะอ่านกี่ไบต์

สัญญาณ prog_ncs และ busy อธิบายแล้วในหัวข้อที่ 4.1.4 ส่วน pc_load, pc_part_sel และ pc_jump_part_sel เป็นสัญญาณที่วงจรแปลคำสั่งใช้ในการควบคุมเรจิสเตอร์ PC ในหน่วยประมวลผลซึ่งกล่าวไปแล้วในหัวข้อที่ 4.2 และสุดท้ายสัญญาณ opcode_load, operand_high_load และ operand_low_load เป็นสัญญาณที่ใช้ในการควบคุม เรจิสเตอร์ B-IR ภายในส่วนทางเดินข้อมูลของวงจรแปลคำสั่ง ซึ่งจะกล่าวถึงต่อไปในหัวข้อที่ 4.3.2

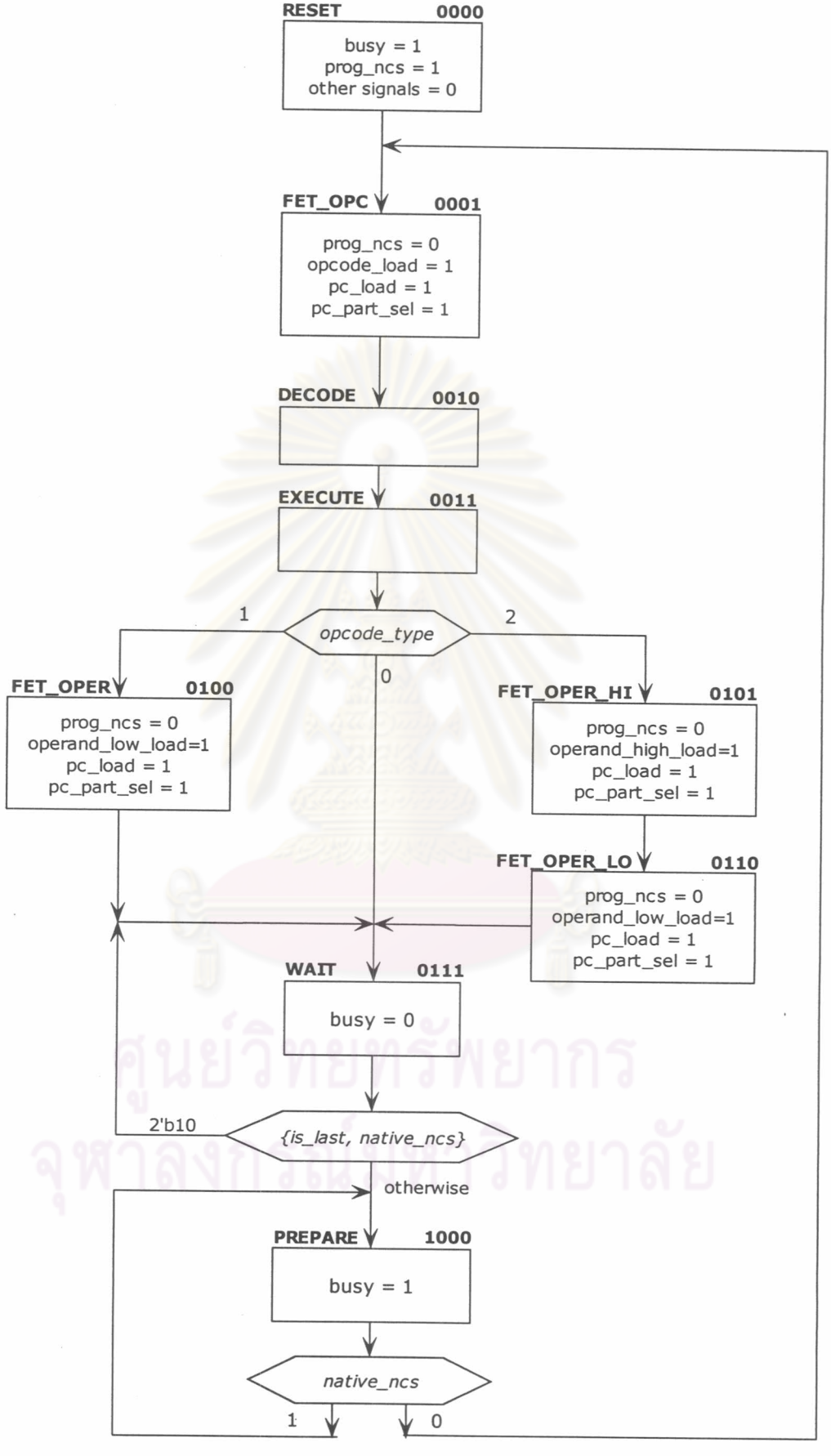
จากรูปที่ 4.11 สถานะการทำงานของหน่วยควบคุมมีด้วยกันทั้งสิ้น 9 สถานะ ซึ่งสัญญาณที่ไม่ได้กล่าวถึงในสถานะนั้นๆ จะถือว่าสัญญาณนั้นๆ ไม่ได้ถูกกระตุ้น (Active low) หรือมีค่าเท่ากับ 0 ในกรณีที่เป็นสัญญาณกระตุ้นสูง (Active high) และมีค่าเป็น 1 ถ้าเป็นสัญญาณกระตุ้นต่ำ (Active low) รายละเอียดของสถานะมีดังนี้

สถานะที่ 0 รีเซต (Reset state, RESET) เป็นสถานะที่เกิดขึ้นหลังจากระบบเริ่มต้นทำงานหรือเกิดการรีเซต มีผลให้หน่วยควบคุมเริ่มต้นทำงานใหม่ และตั้งค่าเริ่มต้นสำหรับทุกๆ สัญญาณควบคุม ในกรณีที่เป็นสัญญาณกระตุ้นสูง (Active high) ค่าเริ่มต้นจะเป็น 0 และกรณีที่เป็นสัญญาณกระตุ้นต่ำ (Active low) ค่าเริ่มต้นจะเท่ากับ 1 แต่จะสังเกตได้ว่าสถานะนี้จะให้สัญญาณ busy = 1 เพื่อไม่ให้หน่วยประมวลผลเข้ามาอ่านคำสั่งเดิม จนกว่าจะอ่านคำสั่งรหัสไบต์ และแปลคำสั่งเสร็จสิ้น

สถานะที่ 1 อ่านรหัสดำเนินการ (Fetch opcode state, FET_OPC) หน่วยควบคุมให้สัญญาณร้องขอคำสั่งรหัสไบต์จากหน่วยความจำโปรแกรม (prog_ncs = 0) สั่งให้ค่า B-PC เพิ่มขึ้น 1 (pc_load = 1) และสั่งให้เรจิสเตอร์ IR เก็บค่าคำสั่งรหัสไบต์ (opcode_load = 1)

สถานะที่ 2 ถอดรหัส (Decode state, DECODE) เมื่อรหัสดำเนินการของคำสั่งรหัสไบต์ถูกจัดเก็บไว้ในเรจิสเตอร์ IR แล้ว ในสถานะนี้จะรอให้ส่วนทางเดินข้อมูลถอดรหัสคำสั่งดังกล่าวว่าเป็นคำสั่งชนิดใด

สถานะที่ 3 กระทำการ (Execute state, EXECUTE) เมื่อส่วนทางเดินข้อมูลถอดรหัสคำสั่งเสร็จแล้วให้สัญญาณ opcode_type ออกมาเพื่อระบุว่าคำสั่งดังกล่าวเป็นคำสั่งชนิดใด โดยสัญญาณ opcode_type เป็นตัวระบุถึงรูปแบบของคำสั่งรหัสไบต์ดังตารางที่ 3.1 ซึ่งในสถานะนี้จะเลือกการทำงานในสถานะต่อไปจาก opcode_type ดังตารางที่ 4.7



รูปที่ 4.11 แผนภาพสถานะของหน่วยควบคุม

ตารางที่ 4.7 การเลือกสถานะถัดไปของสถานกระทำการ

ชนิด	ความหมาย	สถานะถัดไป
00	คำสั่งถูกอ่านมาครบแล้ว	สถานะที่ 7 สถานะรอ
01	ต้องอ่านตัวถูกดำเนินการอีก 1 ไบต์	สถานะที่ 4 อ่านตัวถูกดำเนินการ
10	ต้องอ่านตัวถูกดำเนินการอีก 2 ไบต์	สถานะที่ 5 อ่านตัวถูกดำเนินการไบต์สูง

สถานะที่ 4 อ่านตัวถูกดำเนินการขนาด 1 ไบต์ (Fetch operand one byte state, FETCH_OPER)อ่านตัวถูกดำเนินการขนาด 1 ไบต์จากหน่วยความจำโปรแกรม เมื่ออ่านเสร็จแล้วต่อไปจะไปอยู่ในสถานะรอ

สถานะที่ 5 อ่านตัวถูกดำเนินการไบต์สูง (Fetch high byte operand state, FET_OPER_HI) อ่านตัวถูกดำเนินการไบต์สูง (Most significant byte) จากหน่วยความจำโปรแกรม เมื่ออ่านเสร็จแล้วสั่งให้ B-PC เพิ่มขึ้น 1 เพื่อใช้ในการอ่านตัวถูกดำเนินการไบต์ต่ำต่อไป

สถานะที่ 6 อ่านตัวถูกดำเนินการไบต์ต่ำ (Fetch low byte operand state, FET_OPER_LO) อ่านตัวถูกดำเนินการไบต์ต่ำ (Least significant byte) จากหน่วยความจำโปรแกรม เมื่ออ่านเสร็จแล้วสั่งให้ค่า B-PC เพิ่มขึ้น 1

สถานะที่ 7 สถานะรอ (Wait state, WAIT) วงจรแปลคำสั่งถอดรหัสคำสั่งรหัสไบต์เสร็จสิ้นและสลับการทำงานไปยังหน่วยประมวลผล โดยให้สัญญาณ busy = 0 เพื่ออนุญาตให้หน่วยประมวลผลทำงาน จนกว่าหน่วยประมวลผลจะอ่านคำสั่งสุดท้ายของลำดับคำสั่งเดิม จึงจะเปลี่ยนสถานะไปยังสถานะเตรียมพร้อม

สถานะที่ 8 สถานะเตรียมพร้อม (Prepare state, PREPARE) เมื่อหน่วยประมวลผลอ่านคำสั่งสุดท้ายของลำดับคำสั่งเดิมไปแล้ว หน่วยควบคุมจะมาอยู่ในสถานะเตรียมพร้อมเพื่อรอการอ่านคำสั่งรหัสไบต์คำสั่งต่อไป วงจรแปลคำสั่งจะให้สัญญาณ busy = 1 เพื่อป้องกันไม่ให้หน่วยประมวลผลเข้ามาอ่านคำสั่ง สถานะการทำงานจะเปลี่ยนไปเป็นสถานะอ่านรหัสดำเนินการก็ต่อเมื่อหน่วยประมวลผลมีการร้องขอคำสั่งเดิม (native_ncs = 0) เข้ามา

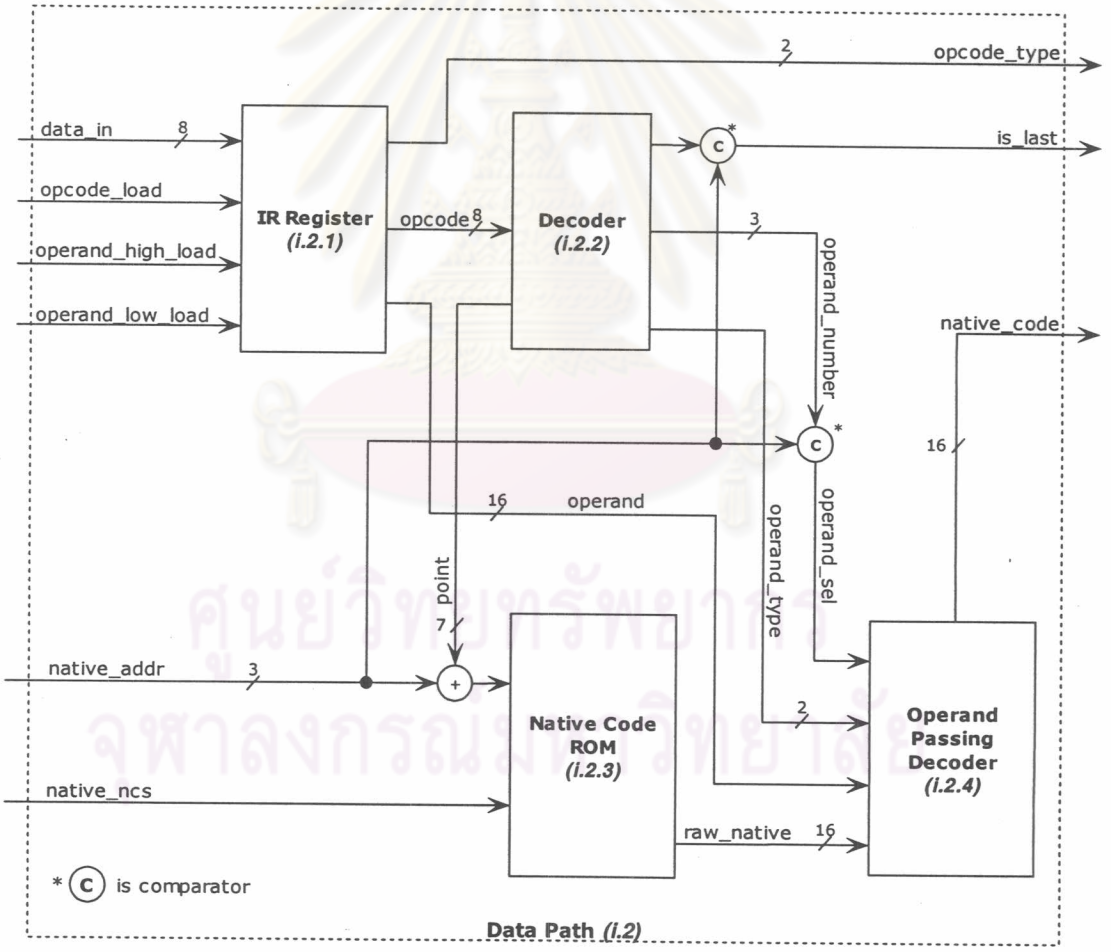
4.3.2 ส่วนทางเดินข้อมูล

ส่วนทางเดินข้อมูล (Data Path, ***i.2***) ถูกสั่งการโดยหน่วยควบคุม วงจรดังกล่าวจะประกอบไปด้วยเรจิสเตอร์ IR (IR Register, ***i.2.1***) ตัวถอดรหัสคำสั่งรหัสไบต์ (Decoder, ***i.2.2***) รอมเก็บคำสั่งเดิม (Native Code ROM, ***i.2.3***) และวงจรส่งตัวถูกดำเนินการ (Operand Passing Decoder, ***i.2.4***) ดังรูปที่ 4.12

เรจิสเตอร์ IR ทำหน้าที่เก็บคำสั่งรหัสไบต์ปัจจุบัน และคำสั่งดังกล่าวถูกถอดรหัสด้วยตัวถอดรหัสคำสั่งรหัสไบต์ได้ผลลัพธ์ออกมาเป็นสัญญาณ 4 สัญญาณได้แก่

1. ตำแหน่งเริ่มต้นของลำดับคำสั่งเดิมที่อธิบายคำสั่งรหัสไบต์ในรอมคำสั่งเดิม (point)
2. ตำแหน่งคำสั่งสุดท้ายในลำดับคำสั่งเดิม (native_number)
3. ตำแหน่งคำสั่งที่จำเป็นต้องส่งตัวถูกดำเนินการไปยังคำสั่งเดิม (operand_number)
4. ชนิดของตัวถูกดำเนินการ (operand_type)

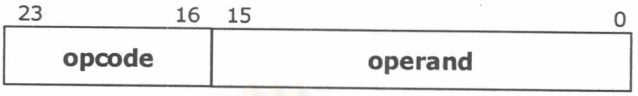
เมื่อหน่วยประมวลผลส่งสัญญาณร้องขอคำสั่งเดิม ตำแหน่งของคำสั่งเดิมจะมาจากผลรวมของตำแหน่งเริ่มต้นของลำดับที่ได้จากตัวถอดรหัสและสัญญาณ native_addr จากหน่วยประมวลผล คำสั่งเดิมที่ถูกเข้าถึงต้องผ่านตัวจัดการการส่งตัวถูกดำเนินการเพื่อส่งตัวถูกดำเนินการจากคำสั่งรหัสไบต์ไปยังการทำงานในลำดับคำสั่งเดิมบางคำสั่ง



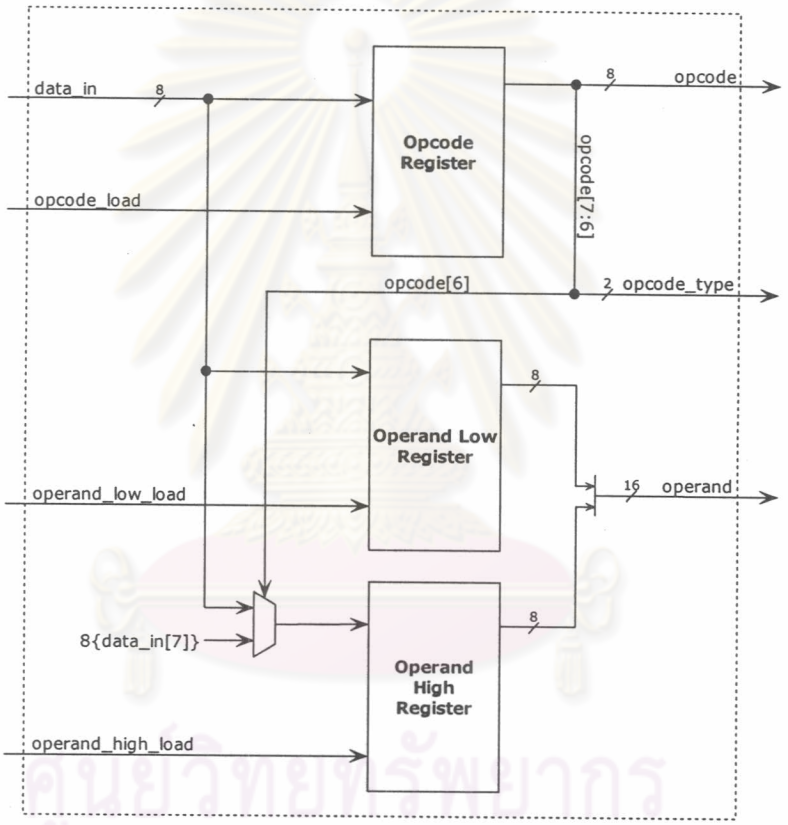
รูปที่ 4.12 แผนภาพบล็อกของส่วนทางเดินข้อมูล

เรจิสเตอร์ IR

เรจิสเตอร์ IR (Instruction Register, ***i.2.1***) เป็นเรจิสเตอร์ขนาด 24 บิตใช้เก็บคำสั่งรหัสไบต์ โครงสร้างของเรจิสเตอร์ IR เป็นไปดังรูปที่ 4.13 บิตที่ 16 ถึง 23 จะเป็นที่เก็บรหัสดำเนินการของคำสั่ง ส่วนในบิตที่ 0 ถึง 15 จะเป็นที่เก็บตัวถูกดำเนินการขนาด 16 บิต แผนภาพบล็อกของวงจรเรจิสเตอร์ IR เป็นไปดังรูปที่ 4.14



รูปที่ 4.13 โครงสร้างของเรจิสเตอร์ IR



รูปที่ 4.14 แผนภาพบล็อกของวงจรเรจิสเตอร์ IR

จากรูปที่ 4.14 เรจิสเตอร์ IR ประกอบไปด้วยเรจิสเตอร์ขนาด 8 บิตจำนวน 3 ตัวซึ่งจะแบ่งออกเป็นเรจิสเตอร์สำหรับเก็บรหัสดำเนินการ 1 ตัว (Opcode Register) และตัวถูกดำเนินการอีก 2 ตัว (Operand Low and Operand High Register) เรจิสเตอร์แต่ละตัวถูกตั้งค่าให้เท่ากับสัญญาณ data_in ที่เข้ามาเมื่อสัญญาณ opcode_load สัญญาณ operand_low_load และสัญญาณ operand_high_load เท่ากับ 1 ตามลำดับ

จะสังเกตได้ว่าถ้ารหัสดำเนินการบิตที่ 6 มีค่าเท่ากับ 1 ซึ่งหมายถึงคำสั่งรหัสไบต์ในขณะนั้นเป็นคำสั่งที่มีรูปแบบตัวถูกดำเนินการขนาด 1 ไบต์ ซึ่งการเก็บตัวถูกดำเนินการขนาด 8 ไบต์นั้นจำเป็นต้องมีการขยายบิตเครื่องหมาย (Sign extension) ซึ่งจากรูปที่ 4.14 ก็มีมัลติเพล็กเซอร์ในการเลือกว่าเป็นข้อมูลตัวถูกดำเนินการ หรือเป็นการขยายเครื่องหมาย

ตัวถอดรหัสไบต์ และรอมเก็บคำสั่งเดิม

จากหัวข้อที่ 4.1.1 คำสั่งรหัสไบต์ทุกคำสั่งสามารถเขียนให้อยู่ในรูปแบบของลำดับคำสั่งเดิมได้ และลำดับคำสั่งเดิมทั้งหมดถูกจัดเก็บลงในหน่วยความจำภายในวงจรแปลคำสั่ง ที่มีชื่อว่ารอมเก็บคำสั่งเดิม (Native Code ROM, ***i.2.3***) สำหรับให้หน่วยประมวลผลเข้ามาอ่านลำดับของคำสั่งที่ทำงานตามความหมายของคำสั่งรหัสไบต์ไปทำงาน

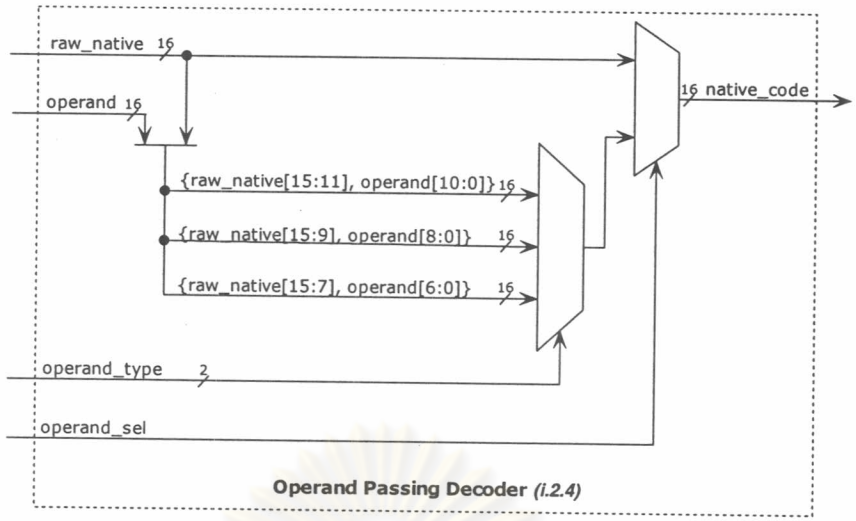
ตัวถอดรหัสคำสั่งรหัสไบต์ (Decoder, ***i.2.2***) มีหน้าที่ในการถอดรหัสคำสั่งเพื่อระบุถึงตำแหน่งแรกของลำดับคำสั่งเดิมในรอมสำหรับเก็บคำสั่งเดิมของคำสั่งรหัสไบต์ ณ ขณะนั้น โดยที่วงจรถอดรหัสจะให้ผลลัพธ์ออกมาเป็นสัญญาณ 4 สัญญาณด้วยกันตามที่ได้กล่าวในหัวข้อ 4.3.2

การส่งตัวถูกดำเนินการ

สำหรับคำสั่งรหัสไบต์ที่มีตัวถูกดำเนินการนั้น เมื่อแปลคำสั่งให้อยู่ในรูปลำดับคำสั่งเดิมแล้วจำเป็นต้องมีการส่งตัวถูกดำเนินการไปยังคำสั่งเดิมเพื่อใช้ในการทำงานด้วย โดนวงจรส่งตัวถูกดำเนินการ (Operand Passing Decoder, ***i.2.4***)

วงจรส่งตัวถูกดำเนินการประกอบไปด้วยมัลติเพล็กเซอร์ 4 ต่อ 1 และมัลติเพล็กเซอร์ 2 ต่อ 1 ขนาด 16 บิต อย่างละ 1 ตัว ดังแผนภาพบล็อกรูปที่ 4.15 โดยมัลติเพล็กเซอร์ 2 ต่อ 1 เป็นตัวเลือกว่าคำสั่งเดิมนั้นเป็นคำสั่งที่ต้องแทรกตัวถูกดำเนินการไปในคำสั่งหรือไม่จากสัญญาณ operand_sel ถ้า operand_sel = 0 แสดงว่าคำสั่งเดิมปัจจุบันเป็นคำสั่งที่ไม่ต้องแทรกตัวถูกดำเนินการลงไป แต่ถ้า operand_sel = 1 แสดงว่าคำสั่งเดิมคำสั่งนี้ต้องถูกแทรกตัวถูกดำเนินการลงไปตามชนิดของตัวถูกดำเนินการในคำสั่งรหัสไบต์และคำสั่งเดิม ซึ่งถูกเลือกมาจากมัลติเพล็กเซอร์ 4 ต่อ 1

มัลติเพล็กเซอร์ 4 ต่อ 1 เป็นตัวเลือกคำสั่งเดิมที่ผ่านการแทรกตัวถูกดำเนินการ 3 แบบ ได้แก่ตัวถูกดำเนินการขนาด 11 บิตที่ใช้ในแอดเดรสซิ่งโหมดแบบสัมพัทธ์ และแบบสัมบูรณ์ (Relative and Absolute addressing mode) ของคำสั่งเดิม ตัวดำเนินการขนาด 9 บิตที่ใช้ในแอดเดรสซิ่งโหมดแบบทันที (Immediate addressing mode) และตัวดำเนินการขนาด 7 บิตที่ใช้ในแอดเดรสซิ่งโหมดแบบดรรชนี (Index addressing mode) โดยการเลือกของมัลติเพล็กเซอร์ 4 ต่อ 1 ใช้สัญญาณ opcode_type เป็นตัวเลือกดังตารางที่ 4.8



รูปที่ 4.15 แผนภาพบล็อกวงจรส่งตัวถูกดำเนินการ

ตารางที่ 4.8 การเลือกค่าผลลัพธ์ของมัลติเพล็กเซอร์ 4 ต่อ 1

operand_type	**คำสั่งเดิมที่ถูกเลือกโดยมัลติเพล็กเซอร์ 4 ต่อ 1**
00	ส่งตัวถูกดำเนินการขนาด 11 บิต ใช้กับคำสั่งกระโดด
01	ส่งตัวถูกดำเนินการขนาด 9 บิตใช้กับคำสั่งที่มีตัวถูกดำเนินการทันที
10	ส่งตัวถูกดำเนินการขนาด 7 บิตใช้กับคำสั่งที่มีตัวถูกดำเนินการแบบดรรชนี
11	ไม่มีการใช้งาน