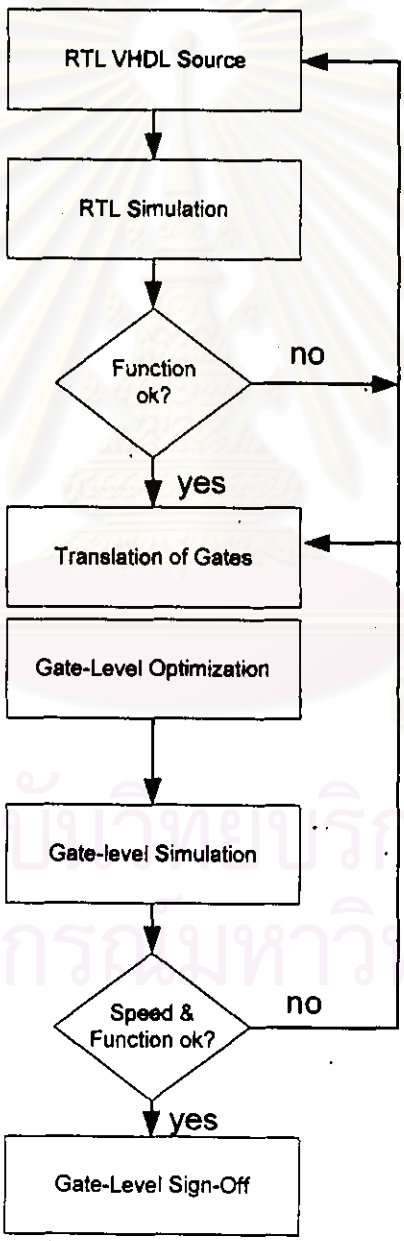
บทที่ 2

หลักการออกแบบระบบดิจิตอล

2.1 ขั้นตอนการออกแบบ



รูปที่ 2. 1 วิธีการออกแบบในระดับสูง

จากรูปที่ 2.1 สามารถอธิบายรายละเอียดในแต่ละขั้นตอนของการออกแบบในระดับสูงด้วยภาษาที่ใช้อธิบายการทำงานของฮาร์ดแวร์ได้ดังนี้

2.1.1 ขั้นตอนการเขียนโค้ดด้วยภาษา VHDLแบบอาร์ทีแอล

จากการออกแบบที่ได้มาจะทำให้เราสามารถเขียนการทำงานของระบบได้โดยการมองให้เป็นลักษณะของแต่ละชิ้นส่วน(component) โดยสามารถเขียนแทนการทำงานได้ด้วยการใช้เอนติตี้(Entity), การทำงานของแต่ละกระบวนการ(process) หรือการใช้แพคเกท(package) ซึ่งเป็นวากยสัมพันธ์ของภาษา VHDL ซึ่งมีรูปแบบของการเขียนและลำดับการเรียกใช้ดังนี้

1. Package ใช้สำหรับเก็บข้อมูลในลักษณะของโปรแกรมสำเร็จรูป
2. Library ใช้เก็บวงจรที่ผ่านการแปลโปรแกรมจากการออกแบบที่ผ่านมา
3. Entity ส่วนของการบอกชื่อในแต่ละชิ้นส่วนและขาสัญญาณที่ติดต่อกับภายนอก
4. Architecture โครงการสร้างการทำงานของวงจร
5. Process เป็นการอธิบายการทำงานในแต่ละชิ้นส่วนย่อย ซึ่งในแต่ละส่วนสามารถเชื่อมโยงการทำงานกันได้

ในงานวิจัยนี้ใช้โปรแกรม Code Wright ช่วนในการเขียนโปรแกรมภาษา VHDL เนื่องจากซอฟแวร์มีความสามารถในการตรวจสอบรูปแบบการใช้งานของภาษาVHDLได้ นอกจากนี้ยังมีความสามารถในการเขียนโปรแกรมภาษาอื่นๆเช่น แอสแซมบี้(Assembly) หรือภาษาซี (C)

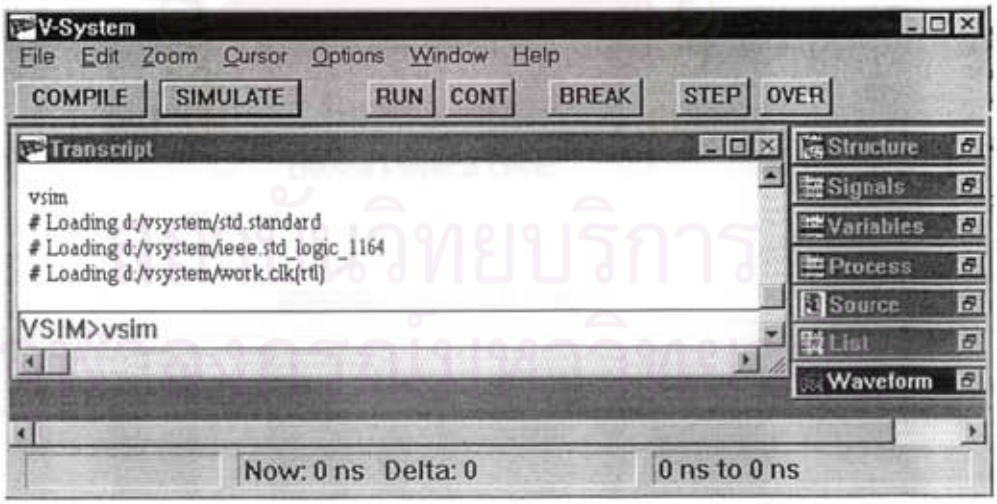
2.1.2 ขั้นตอนในการจำลองการทำงาน

เมื่อทำการแปลโปรแกรม(Compile) ด้วยโปรแกรมที่ชื่อว่า Vsystems แล้วก็จะเริ่มการตรวจสอบการทำงานของระบบโดยใช้การจำลองการทำงานบนโปรแกรม Vsystems Simulation ตัวเดียวกัน โดยมีขั้นตอนการทำงานดังนี้

1. เมื่อได้ตัวโปรแกรมที่ได้จากการเขียนเรียบร้อยแล้วจะทำการแปลโปรแกรม(Compile) โดยการใช้คำสั่ง Vcom หรือการใช้เมนูจากหน้าจอของโปรแกรม

รูปที่ 2. 2 การใช้งาน Vsystems ในการแปลโปแกรม

2. เมื่อโปรแกรมที่อธิบายการทำงานของระบบไม่มีปัญหาหรือไม่มีข้อผิดพลาดในเรื่องของรูปแบบก็จะทำการจำลองการทำงาน ซึ่งถ้าจะให้มีการทำงานสามารถทำได้โดยการเรียกใช้ Vsim ซึ่งจะเกิดหน้าจอเพื่อให้เลือกเอาตัวเอนติตี้ที่ต้องการจำลองการทำงาน มาดังนี้



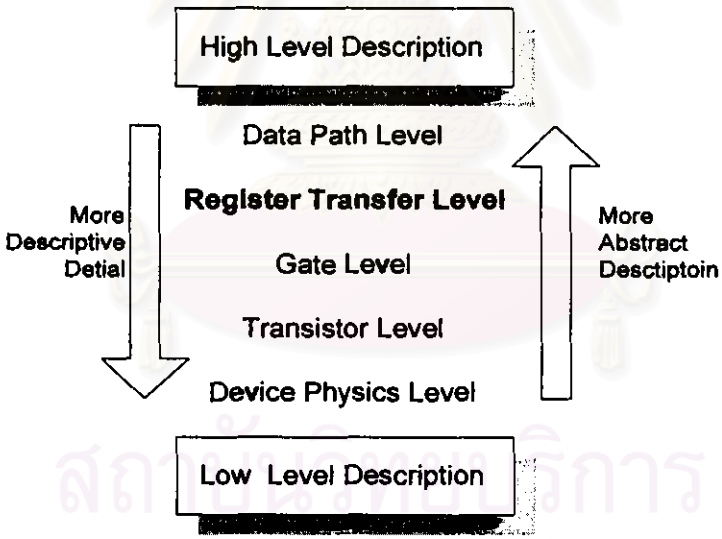
รูปที่ 2. 3 ผลที่ได้จากการใช้คำสั่ง Vsim

3. จากนั้นทำการป้อนค่าสัญญาณต่างๆที่ต้องการแล้วดูผลลัพธ์ที่ได้ว่าตรงกับความต้องการและการทำงานของระบบจริงหรือไม่ โดยการป้อนค่าจะเขียนคำสั่ง VHDL ขึ้นมาเพื่อทำงานแทนการป้อนค่า ถ้าหากมีข้อผิดพลาดต้องทำการเปลี่ยนแปลงโค้ด ใหม่แล้วเริ่มการแปลโปรแกรม ใหม่อีกครั้งจนกว่าจะได้ค่าของสัญญาณตามที่ต้องการ

4. ทำการทดสอบการทำงานของทุกคำสั่งที่มีการใช้งานในอาร์ม 7 โดยการเขียนโปรแกรมทดสอบให้ครอบคลุมทุกคำสั่งที่มีการใช้งานเพื่อตรวจสอบให้ได้มากกรณีที่เป็นไปได้ของคำสั่งทั้งหมด ในที่นี้ทางกลุ่มได้ทดสอบการทำงานของทุกคำสั่งโดยจะแสดงผลลัพธ์ของสัญญาณที่เกิดขึ้นในแต่ละคำสั่งในบทต่อไป

2.2 ข้อกำหนดรายละเอียดและการอธิบายการทำงาน (Specification and Description)

ความสามารถในการกำหนดรายละเอียดและอธิบายการทำงานของฮาร์ดแวร์เป็นหลักการสำคัญของกระบวนการออกแบบวงจรดิจิตอลให้มีการทำงานถูกต้อง โดยในกระบวนการออกแบบมีการแบ่งระดับของการอธิบายการทำงานของวงจรไว้หลายระดับ ซึ่งระดับและภาษาที่ใช้ขึ้นอยู่กับความซับซ้อนของระบบ



รูปที่ 2. 4 ระดับของการอธิบายการทำงานของวงจร

จากรูปที่ 2.4 แสดงการแบ่งระดับของการอธิบายการทำงานในแต่ละขั้นตอนการออกแบบวงจร พบว่าถ้าเป็นการอธิบายการทำงานในระดับล่างการอธิบายการทำงานในแต่ละชิ้นส่วน (Component) จะต้องมีรายละเอียดเพิ่มมากขึ้น วิธีการและภาษาที่ใช้ในการออกแบบแต่ละระดับจะมีความแตกต่างกัน เช่น ในระดับประตูสัญญาณ (Gate-Level) จะใช้วิธีการของตรรกะแบบบูล (Boolean Logic) อธิบายการทำงาน ส่วนในระดับของทรานซิสเตอร์ใช้ทฤษฎีของการสลับ (Switching)

วิธีการออกแบบสามารถแบ่งออกได้เป็น การออกแบบจากบนลงล่าง (Top-down), การออกแบบจากล่างขึ้นบน (Bottom-up) และแบบผสม ดังที่แสดงไว้ในรูปที่ 2.5 การออกแบบจากบนลงล่างคือการออกแบบจากคุณลักษณะที่กำหนดและสร้างวงจรจากระดับของหลักนามธรรม (Abstract) สูงลงไปสู่หลักนามธรรมต่ำ ดังตัวอย่างในรูปที่ 2.6

2.3 การสร้างรายละเอียดในการอธิบายการทำงาน (Generating Detailed Descriptions)

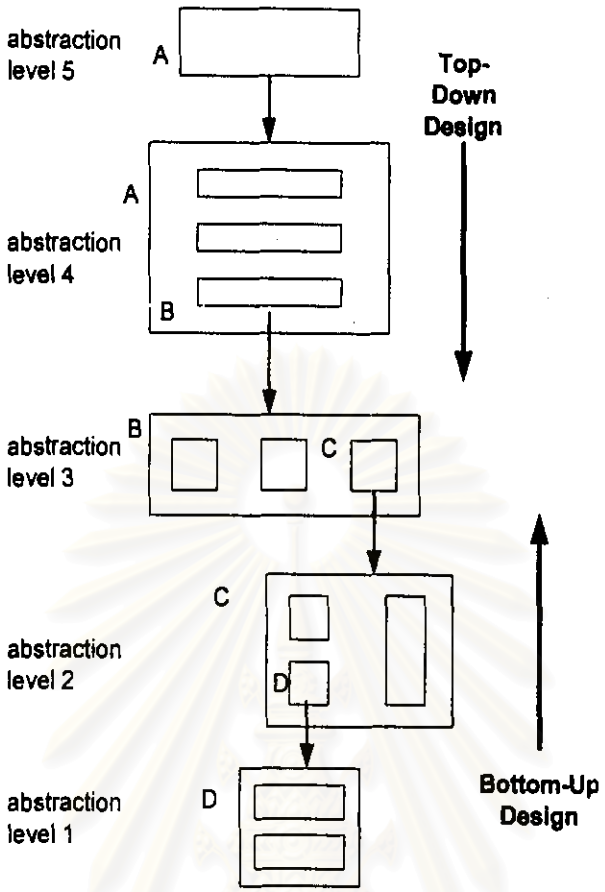
ในรูปที่ 2.6 เป็นการแสดงถึงลำดับขั้นตอนการออกแบบอย่างคราวๆซึ่งเป็นการออกแบบจากบนลงล่าง โดยที่นำมาใช้ในการออกแบบไมโครโพรเซสเซอร์ในการทำงานวิจัยครั้งนี้ ซึ่งสามารถแบ่งออกเป็น 3 ขั้นตอนดังนี้

2.3.1 การแบ่งส่วนการออกแบบ (Design Partitioning) : คือการแบ่งระบบหรือชิ้นส่วนที่ต้องการออกแบบให้มีขนาดเล็กเพื่อให้ง่ายต่อการอธิบายการทำงานและตรวจสอบผล เนื่องจากยิ่งมีขนาดใหญ่และซับซ้อนมากจะทำให้การตรวจสอบเป็นไปได้ยาก เนื่องจากมีหลายๆโมดูลซ้อนทับการทำงานกันมากขึ้น

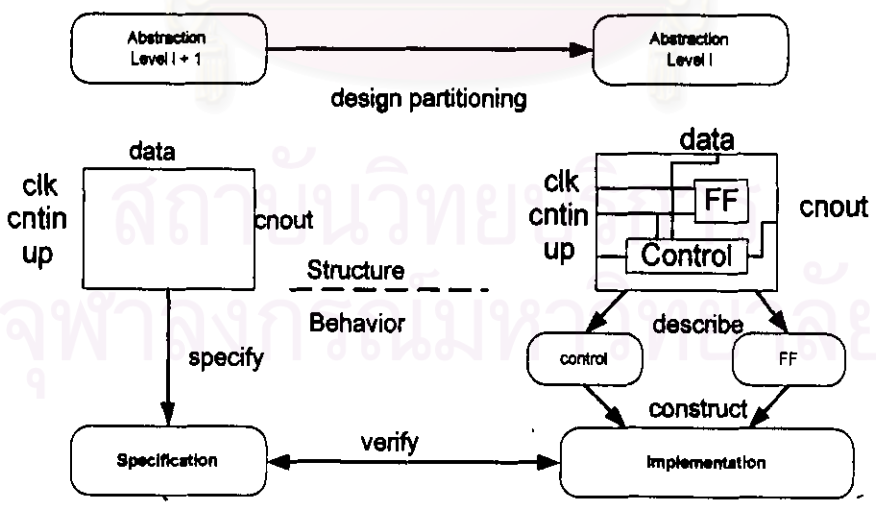
2.3.2 ข้อกำหนดคุณลักษณะและการอธิบายการทำงาน : ในการอธิบายการทำงานของวงจรดิจิตอลสามารถทำได้ 2 แบบคือการอธิบายถึงพฤติกรรมของวงจรนั้น และการอธิบายโครงสร้างของวงจร โดยในแต่ละแบบจะมีข้อดีข้อเสียที่ต่างกันไป ดังนี้ การอธิบายพฤติกรรมของวงจรจะทำให้สามารถเข้าใจการทำงานของวงจรนั้นได้ง่ายกว่าการเขียนแบบโครงสร้างของวงจร แต่ในขณะเดียวกันการอธิบายพฤติกรรมของวงจรในบางรูปแบบไม่สามารถทำการสังเคราะห์หรือสร้างวงจรได้จริงด้วยความสามารถของซอฟแวร์ที่มีใช้อยู่ในปัจจุบัน เช่น คำสั่ง wait until ไม่สามารถสังเคราะห์เป็นวงจรที่ใช้งานได้จริงเป็นต้น

2.3.3 การทวนสอบความถูกต้อง (Verification) : เป็นกระบวนการเพื่อแสดงให้เห็นว่าวงจรที่ได้จากการออกแบบได้อธิบายการทำงานนั้นถูกต้องตรงกับคุณลักษณะที่ต้อง (Requirement)

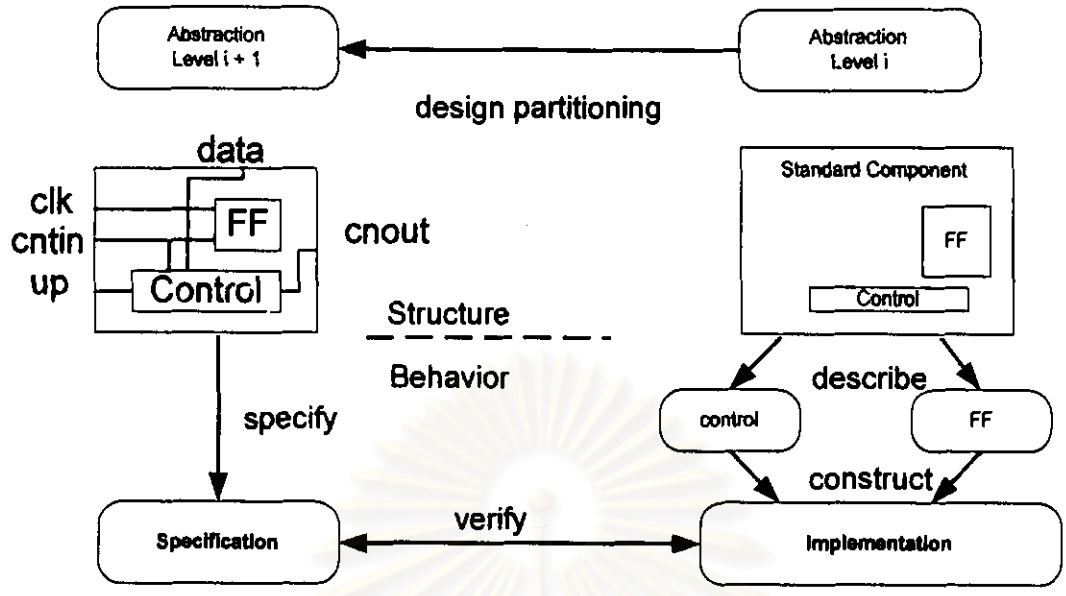
ในรูปที่ 2.7 เป็นตัวอย่างของการออกแบบแบบล่างขึ้นบนจะเพิ่มความเป็นหลักนามธรรมมากขึ้นมาจากรายละเอียดของวงจร การออกแบบวงจรจะเริ่มจากข้อกำหนดคุณลักษณะซึ่งเป็นข้อกำหนดและความต้องการของระบบซึ่งอาจทำโดยวิธีของการใช้ออกแบบรูปวงจรแผนภาพเค้าร่าง (Schematics), แผนผังของเวลาในการทำงาน (Timing Diagram) และภาษาที่ใช้อธิบายการทำงานของวงจรโดยข้อกำหนด คุณลักษณะประกอบด้วย 3 อย่างหลักๆดังนี้



รูปที่ 2. 5 แสดงการออกแบบด้วยวิธีการจากบนลงล่างและจากล่างขึ้นบน



รูปที่ 2. 6 ตัวอย่างการออกแบบแบบบนลงล่าง



รูปที่ 2. 7 ตัวอย่างการออกแบบแบบล่างขึ้นบน

- สื่อหรือตัวแทนที่แสดงความต้องการของการออกแบบ
- สิ่งจำเป็นในการตรวจสอบ
- การเชื่อมต่อเข้ากับเครื่องมือในการออกแบบ

แผนภาพทางเรขาคณิต (Geometrical Diagram) และทฤษฎีว่าด้วยกราฟ (Graph Theory) ใช้ในการกำหนดลักษณะทางโครงสร้างของวงจร ส่วนการอธิบายด้วยตรรกะแบบบูล และภาษาของการเขียนโปรแกรมจะใช้ในการกำหนดพฤติกรรมและเวลาของการทำงานของวงจร ปัจจุบันภาษาที่ใช้ในการออกแบบวงจรหรือที่เรียกว่า Hardware Description Language (HDL) เป็นที่นิยมกันมาก เนื่องจากสามารถใช้ได้ง่าย และสะดวก จึงทำให้มีความรวดเร็วในการพัฒนางานทางด้านการออกแบบ ตัวอย่างเช่นภาษา ISPS, Verilog, ELLA และ VHDL

มีการนำเอาภาษาตรรกะแบบบูลลีนใช้ในการอธิบายการทำงานของวงจรซึ่งเป็นภาษาที่มีแบบแผน (Formal Language) แน่นอนดังนั้นจึงสามารถพิสูจน์การทำงานเพื่อทวนสอบความ ถูกต้องของวงจรได้ด้วยวิธีการทางคณิตศาสตร์ โดยที่ต้องมีรูปแบบของภาษาที่ใช้กำหนดคุณลักษณะดังนี้

- สามารถแสดงโครงสร้างของระบบได้ โดยเฉพาะในส่วนของการทำแบ่งส่วนการออกแบบ (Design Partition)
- สามารถแสดงพฤติกรรมการทำงานทั้งฟังก์ชันการทำงานและเวลาที่ใช้
- สามารถอธิบายการทำงานของวงจรตั้งแต่คุณลักษณะไปจนถึงระดับของประตูสัญญาณได้ด้วยภาษาเพียงภาษาเดียว

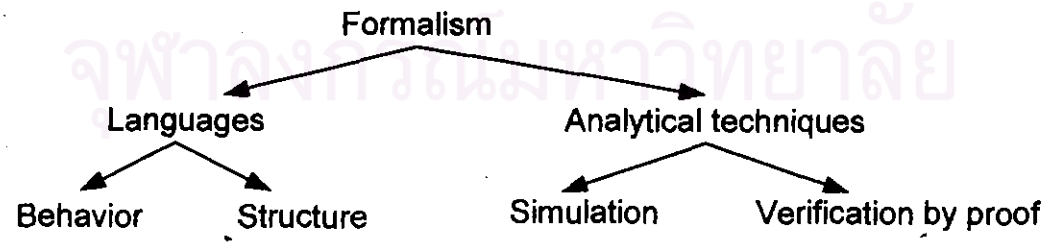
- รองรับการทำงานและเทคนิคการทวนสอบ ภาษาที่เป็นแบบแผนซึ่งในปัจจุบันภาษาที่ใช้ในการออกแบบจริงจะไม่เป็นภาษาที่เป็นแบบแผน ซึ่งก็รวมทั้ง VHDL ด้วย

กระบวนการของการตรวจสอบข้อผิดพลาดในวงจรที่ออกแบบต้องทำก่อนจะนำไปสู่กระบวนการผลิต เนื่องจากค่าใช้จ่ายในกระบวนการนั้นสูงมาก ซึ่งการหาข้อผิดพลาดต้องเริ่มตั้งแต่การออกแบบในระดับของซอฟแวร์ ซึ่งสามารถทำการแก้ไขข้อผิดพลาดได้ การตรวจสอบความถูกต้องในการทำงานของวงจรที่ออกแบบด้วย HDL จะทำได้โดยการจำลองการทำงาน โดยใช้โปรแกรมจำลองการทำงาน ตัวจำลองโดยทั่วๆไปจะมีรูปแบบเหมือนๆกันโดยใช้โมดูลของวงจรให้ทำงานตามเงื่อนไขต่างๆที่สร้างขึ้น ในทำนองเดียวกันนี้เทคนิคของการพิสูจน์ความถูกต้องเราจะเรียกว่าการทวนสอบอย่างมีแบบแผนซึ่งแบบนี้จำเป็นต้องมีโมดูลที่อ้างอิงในการเปรียบเทียบ

การอธิบายโครงสร้างการทำงาน (Structural Description) เป็นการแสดงความสัมพันธ์การเชื่อมต่อระหว่างชิ้นส่วนต่างๆ ที่ประกอบรวมกันเป็นโครงสร้างที่ซับซ้อน

การอธิบายพฤติกรรมการทำงาน (Behavior Description) เป็นการอธิบายถึงพฤติกรรมของวงจรหรือแต่ละชิ้นส่วนตามคุณลักษณะที่ต้องการ

ภาษาที่ใช้ในการอธิบายการทำงาน (Description Language) ซึ่งจะถูกใช้แสดงการทำงานของวงจรทั้งแบบโครงสร้างและพฤติกรรมการทำงาน โดยจะเรียกว่า HDL และสามารถนำไปวิเคราะห์ได้ด้วยการจำลองการทำงาน นอกจากนี้ยังมีการใช้ตรรกะแบบบูล และภาษา High-order Logic (HOL) อธิบายการทำงานของวงจรได้อย่างมีแบบแผนและรองรับเทคนิคของการทวนสอบโดยใช้วิธีการอย่างมีแบบแผน ซึ่งวงจรที่ออกแบบไม่จำเป็นต้องใช้การจำลองการทำงานในการทวนสอบความถูกต้องของวงจร จากรูปที่ 2.8 อธิบายรูปแบบในการกำหนดคุณลักษณะของฮาร์ดแวร์และการทวนสอบ



รูปที่ 2. 8 แบบแผนของระบบในการกำหนดคุณลักษณะของวงจรดิจิตอล

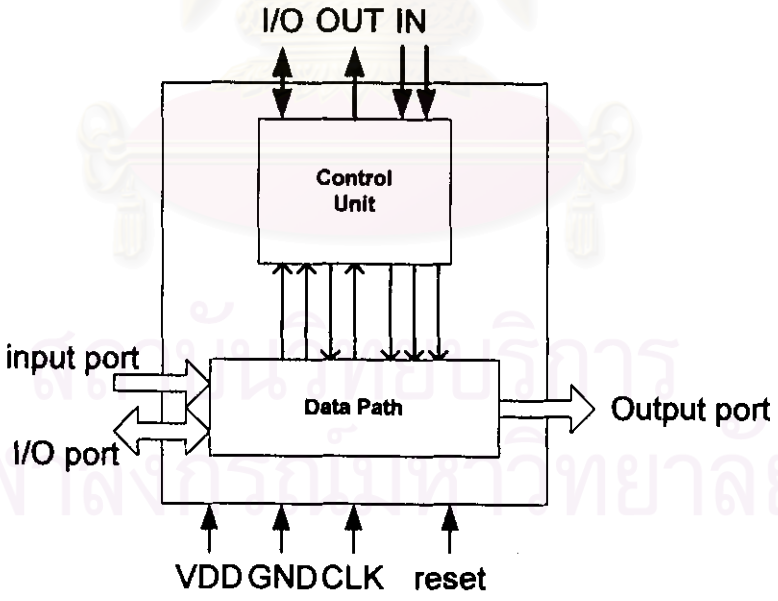
ข้อจำกัดของการใช้การทวนสอบแบบมีแบบแผน (Formal Verification) มีดังนี้

- เราไม่สามารถพิสูจน์ได้ว่าข้อกำหนดคุณลักษณะที่ต้องการถูกต้องกับความต้องการจริง
- เครื่องมือหรือภาษาที่ใช้ในการออกแบบในโลกความเป็นจริงหรือในเชิงพาณิชย์ ไม่เป็นภาษาที่เป็นแบบแผน
- มีข้อกำจัดในการใช้วิธีการอย่างมีแบบแผนเมื่อต้องการตรวจสอบเวลาในการทำงาน
- โครงสร้างที่ทำหน้าในการควบคุม (Control Unit) จะแสดงการทำงานด้วยภาษาที่เป็นแบบแผนได้ยาก การทวนสอบจะซับซ้อนยุ่งยากมากขึ้น

2.4 การอธิบายโครงสร้างและพฤติกรรมการทำงานของฮาร์ดแวร์

ปัจจุบันการออกแบบวงจรได้พัฒนาให้มีการใช้ภาษาเป็นสิ่งอธิบายการทำงานและสร้างวงจรขึ้นใช้จริง เราเรียกภาษานั้นว่า Hardware Description Language (HDL) ในงานวิจัยนี้ใช้ภาษา VHDL ซึ่งมีโครงสร้างของภาษาบนพื้นฐานของภาษาปาสกาล (Pascal)

โครงสร้างภายในไมโครโพรเซสเซอร์หรือระบบทั่วไปจะประกอบด้วย 2 ส่วนคือดาต้าพาท และส่วนของการควบคุมดังรูปที่ 2.9



รูปที่ 2. 9 โครงสร้างภายในไมโครโพรเซสเซอร์ทั่วไป

ในแต่ละชิ้นส่วนสามารถแยกออกแบบในแต่ละชิ้นส่วนย่อยได้ และสามารถสร้างเป็นคลังโปรแกรม (Library) เก็บไว้ใช้งานในวงจรหรือการออกแบบอื่นๆได้

2.5 ไปป์ไลน์ (Pipeline)

ไปป์ไลน์เป็นหนึ่งในรูปแบบของการทำงานแบบขนานที่นิยมใช้กันมาก คำสั่งการทำงานจะถูกแบ่งออกเป็นคำสั่งย่อยๆได้ k คำสั่ง ดังนั้นระบบจะมีสเตตในการทำงานทั้งหมด k สเตต สามารถแบ่งการทำงานของไปป์ไลน์ออกได้เป็น 2 ชนิด

- ไปป์ไลน์คำสั่ง (Instruction Pipeline) เป็นการนำคำสั่งเข้ามาและทำงานในไปป์ไลน์
-ไปป์ไลน์ทางคณิตศาสตร์ (Arithmetic Pipeline) เป็นการนำการทำงานของทางคณิตศาสตร์มาใช้ในสเตตของไปป์ไลน์

ไปป์ไลน์คำสั่งจะถูกนำไปใช้ในการออกแบบไมโครโพรเซสเซอร์รุ่นใหม่ แต่ไปป์ไลน์ทางคณิตศาสตร์ จะถูกใช้เฉพาะในส่วนของการออกแบบหน่วยคำนวณและตรรกะ (ALU) ซึ่งในที่นี้จะกล่าวถึงเฉพาะส่วนของไปป์ไลน์คำสั่ง โดยให้ 1 ไปป์ไลน์มีการทำงาน 4 สเตต (k=4) ดังนี้

1. เฟส (F-Fetch) คำสั่งถูกนำเข้าสู่หน่วยประมวลผล
2. ถอดรหัส (D-Decode) คำสั่งถูกถอดรหัสเพื่อให้มีการเข้าถึงค่าในรีจิสเตอร์
3. ประมวลผล (E-Execute) คำสั่งถูกประมวลผลทำงาน
4. เก็บค่า (W-write back) ผลลัพธ์ที่ได้จากการทำงานถูกเขียนกับลงยังปลายทางที่ต้องการ

ตัวอย่างของไปป์ไลน์ 4 สเตตโดยมีคำสั่งย่อยเป็น I , I+1, I+2, I+3 มีการทำงานอย่างต่อเนื่องทำให้ไปป์ไลน์ไม่ว่างถูกใช้งานตลอดเวลา มีความเป็นไปได้ที่การทำงานในสเตตของการถอดรหัสจะใช้มากกว่า 1 คาบเวลา ถ้าคำสั่งมีขนาดใหญ่กว่า 1 คำแล้วขนาดของบัสข้อมูลมีแค่ 1 คำ ดังนั้นสเตตของการเฟสจะสามารถใช้งานมากกว่า 1 คาบเวลา

2.5.1 ปัญหาในไปป์ไลน์ (Pipeline Hazard)

คือการที่ไปป์ไลน์โดนขัดขังหวะการทำงานทำให้ไม่สามารถทำงานได้อย่างต่อเนื่อง ซึ่งสามารถแบ่งออกได้เป็น 3 ชนิดของการเกิดปัญหาในไปป์ไลน์

- ปัญหาที่เกิดจากโครงสร้าง(Structure Hazard) เป็นปัญหาที่เกิดจากการใช้ทรัพยากรที่ทำให้การทำงานของไปป์ไลน์ไม่สามารถดำเนินต่อไปได้ตามปกติ ระบบมีการใช้ทรัพยากรในแต่ละ สเตตดังตารางดังต่อไปนี้

ตารางที่ 2. 1 การใช้ทรัพยากรในแต่ละสเตตในไปป์ไลน์

สเตตไปป์ไลน์	ทรัพยากรที่จำเป็นต้องใช้งาน
F	PC, MAR, Address / Data bus
D	Decoder, Internal Bus
E	ALU, MAR, Address and Data Bus
W	Internal Bus

*MAR = Memory Address Register

PC = Program counter (ตัวนับระบุตำแหน่งคำสั่ง)

ALU = Arithmetic & Logic Unit (หน่วยคำนวณและตรรกะ)

จากการศึกษาพบว่า เอ็มเออาร์, บัสข้อมูล และบัสเลขที่อยู่ (Address bus) ถูกใช้ในสเตตของการประมวลผล คำสั่งย่อยที่ I+1 ในขณะที่สเตตประมวลผลก็ต้องการใช้ทรัพยากรเดียวกัน คือมีการนำเอาข้อมูลจากหน่วยความจำในระหว่างการทำงาน หรือในการถอดรหัสกับการเขียนผลลัพธ์อาจจะมีการใช้บัสภายใน เหล่านี้คือสาเหตุของการเกิดปัญหาในไปป์ไลน์ในลักษณะของโครงสร้าง ซึ่งสามารถทำการแก้ไขได้โดยการสร้างตำแหน่งของทรัพยากรขึ้นมาใหม่เช่น ถ้ามี หน่วยความจำแคช (cache memory) 2 ตัวโดยตัวหนึ่งใช้สำหรับเก็บคำสั่งโปรแกรม อีกตัวใช้สำหรับเก็บข้อมูล ซึ่งจะแยกการเข้าถึงข้อมูลในบัสและเอ็มเออาร์ออกจากกัน หน่วยประมวลผลจะสามารถเข้าถึงข้อมูลในแคชที่เก็บคำสั่งในการทำงานของสเตตการนำข้อมูล ในขณะที่สเตตประมวลผลจะเข้าถึงข้อมูลส่วนในแคชข้อมูล ทำให้มีการทำงานอย่างต่อเนื่อง ในการแก้ปัญหาของโครงสร้างไปป์ไลน์ในการใช้บัสภายใน สามารถทำได้โดยการใช้บัสภายในหลายชุดกับรีจิสเตอร์หลายๆพอร์ท นอกจากนี้อาจเกิดภัยโครงสร้างได้ถ้ามีการใช้หน่วยคำนวณและตรรกะ ไปคำนวณการเพิ่มขึ้นของค่า พีซี หลังจากที่มีการไปนำเอาคำสั่งโดยสามารถแก้ไขได้โดยการสร้าง ลอจิกพิเศษเพื่อใช้ในการเพิ่มค่า พีซี ทำให้ไม่มีการประมวลผลในหน่วยคำนวณและตรรกะ

- ปัญหาที่เกิดจากข้อมูล (Data Hazard)

  สามารถอธิบายได้ด้วยตัวอย่างดังต่อไปนี้

```
add r3,r2,r1      ; r2 + r3 -> r1
sub r4,r1,r5      ; r4 - r1  -> r5
```

จากคำสั่งข้างต้นสามารถเขียนเป็นการทำงานของไปป์ไลน์ได้ดังนี้

ตารางที่ 2. 2 การทำงานของไปป์ไลน์ในการทำคำสั่ง add และ sub

คาบเวลา	1	2	3	4	5
Add	F	D	E (ค่าใหม่ใน r1 ประมวลผล)	W	
Sub		F	D (ค่าใน r1 ถูกอ่านไปใช้)	E	W

จากลำดับการทำงานข้างต้นพบว่าค่าใหม่ของรีจิสเตอร์ที่ 1 จะถูกเก็บค่าในคาบที่ 4 แต่ในขณะที่ในคาบที่ 3 มีการอ่านค่าในรีจิสเตอร์ไปใช้งานในคำสั่งลบ ดังนั้นค่าที่ได้จะเป็นค่าที่ยังมีการเปลี่ยนแปลงหลังจากการทำคำสั่งบวก ลักษณะการผิดพลาดของข้อมูลแบบนี้เรียกว่า ปัญหาในไปป์ไลน์ที่เกิดจากข้อมูล

วิธีการแก้ปัญหาแบบง่ายคือการใช้วิธีการ Stalling Pipeline เพื่อหยุดการทำงานและรอจนกระทั่งค่าในรีจิสเตอร์ที่ 1 ถูกเก็บค่าใหม่เรียบร้อยแล้วจึงทำการถอดรหัสเอาค่าไปใช้ในคำสั่งถัดไป ทำให้ต้องเสียเวลาในการทำงานไป 2 คาบเวลาดังตารางที่ 2.3 โดยให้ ST แทนการทำ Stalling Pipeline

ตารางที่ 2. 3 แสดงการใช้ Stalling Pipeline เพื่อแก้ปัญหาภัยจากข้อมูล

คาบเวลา	1	2	3	4	5	6	7
Add	F	D	E	W			
Sub		F	ST	ST	D	E	W

มีวิธีการแก้ปัญหาของไปปน์ไลน์ที่เกิดจากข้อมูล ที่ทำให้การทำงานของไปป์ไลน์ยังคงสามารถทำงานได้คงเดิมและไม่เสียคาบเวลาโดยใช้วิธีการของ Forwarding โดยผลลัพธ์ที่ได้จากการทำงานในหน่วยคำนวณและตรรกะจะมี 2 ส่วนคือ ข้อมูลถูกนำไปเก็บในรีจิสเตอร์ที่ต้องการ และส่วนของข้อมูลถูกนำกลับเข้าอุปกรณ์รวมสัญญาณ (Multiplexor) แล้วส่งไปเป็นอินพุทของหน่วยคำนวณและตรรกะและถูกใช้งานได้เมื่อคำสั่งถัดไปต้องการใช้งาน แต่การทำ Forwarding ไม่สามารถแก้ไขปัญหาของไปปน์ไลน์ที่เกิดจากข้อมูลได้ทุกชนิด ซึ่งมีโอกาสเป็นไปได้ว่าการทำงานในคำสั่งก่อนหน้าไม่สามารถเสร็จภายในคาบเดียว ดังนั้นคำสั่งถัดไปจะไม่สามารถได้ค่าที่ถูกต้องใช้งาน เช่นในการเข้าถึงหน่วยความจำในการเก็บข้อมูลไม่สามารถทำให้เสร็จภายในคาบเวลาเดียว ดังนั้นถ้ามีการใช้งานค่า ณ. ตำแหน่งมาถูกเก็บในคำสั่งถัดไป จะทำให้ค่าที่ได้ผิดพลาด แม้แต่การบรรจุ (Load) ข้อมูลจากหน่วยความจำแคชก็ต้องใช้เวลาในการคำนวณตำแหน่งที่เก็บข้อมูล โดยการเปลี่ยนจากเลขที่อยู่เสมือน (Virtual Memory) ไปเป็นเลขที่อยู่จริงบนหน่วยความจำ เมื่อมีการตรวจพบปัญหาในไปป์ไลน์ที่เกิดจากข้อมูลและมีการใช้ Stalling Pipeline ในระบบเราเรียกว่าไปป์ไลน์เกิด interlock ปัญหาในไปป์ไลน์ที่เกิดจากข้อมูลสามารถแบ่งออกได้เป็น 3 ชนิดโดยกำหนดให้คำสั่ง i เกิดก่อนคำสั่ง j ดังนี้

- RAW – read after write : คือการที่คำสั่ง j พยายามอ่านค่าก่อนที่คำสั่ง i ถูกเขียนเก็บ
- WAR – write after read : คือคำสั่ง j พยายามเขียนค่าลงในรีจิสเตอร์ปลายทางก่อนที่คำสั่ง i จะถูกอ่าน
- WAW – write after write : คำสั่ง j พยายามเขียนเป็นถูกคำเนินการ (Operand) ก่อนที่จะถูกเขียนเก็บโดยคำสั่ง I โดยสามารถเกิดขึ้นได้ในไปป์ไลน์ เมื่อมีคาบเวลาการเขียนมากกว่า 1 สเตตดังนั้นจะไม่เกิดขึ้นเลยถ้ามีการเขียน เสร็จภายใน 1 คาบเวลา

- ปัญหาในไปป์ไลน์ที่เกิดจากการควบคุม (Control Hazard)

เมื่อมีการทำคำสั่งกระโดดข้ามการทำงาน (Jump or Branch) ลำดับคำสั่งย่อยในไปป์ไลน์จะถูกยกเลิกทิ้งไป คำสั่งใหม่จะถูกนำเอาเข้ามาทำงาน เพื่อให้มีผลกระทบต่อไปป์ไลน์น้อยที่สุด(จำนวนคำสั่งย่อยที่ถูกยกเลิกไปน้อยที่สุด) จะมีขั้นตอนการทำงานดังนี้

- ตรวจหาคำสั่งกระโดดข้ามให้เร็วที่สุดเท่าที่เป็นไปได้ และถ้าเป็นการกระโดดข้ามแบบไม่มีการตรวจเงื่อนไขจะทำได้ง่ายเนื่องจากสามารถยกเลิกคำสั่งที่อยู่ถัดไปได้ทันทีกายในสเตตของการถอดรหัส สำหรับการกระโดดข้ามการทำงานแบบมีเงื่อนไขจะทำได้ยากมากกว่าเนื่องจากต้องทำการตรวจสอบเงื่อนไขก่อนหลังจากที่ถอดรหัสเรียบร้อยแล้วทำให้ต้องเสียเวลาเพิ่มขึ้น

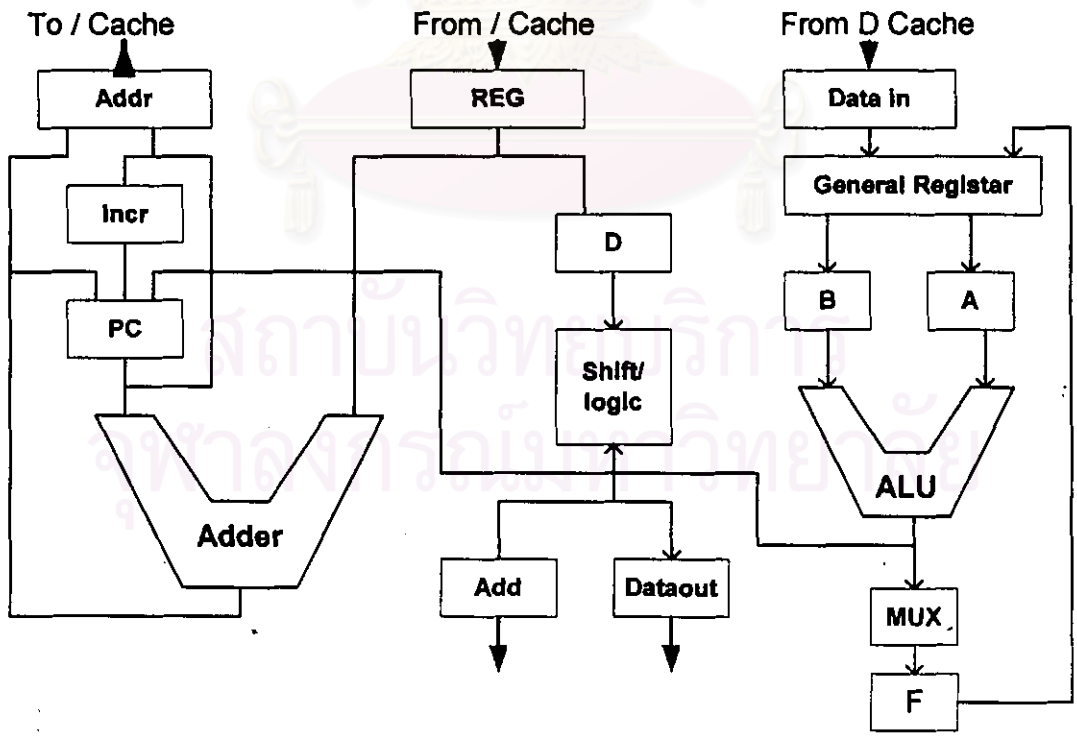
- โหลดค่าตำแหน่งของคำสั่งใหม่ใส่ใน พีซี ให้เร็วที่สุดซึ่งถ้าหากมีการอ้างอิงหน่วยความจำที่ต้องคำนวณค่าทำให้ต้องเพิ่มคาบเวลาในการทำงานพิเศษเข้าไป

อีกวิธีที่จะช่วยลดการหน่วงของเวลาที่เกิดจากปัญหาในไปป์ไลน์ที่เกิดจากการควบคุม คือมีหน่วยความจำแคชเก็บตำแหน่งที่จะกระโดดข้ามไปทำงานในหน่วยประมวลผล ซึ่งจะประกอบด้วยชุดของคำสั่งแรกที่มีความเป็นไปได้ในการทำงานเมื่อเกิดการกระโดดข้ามการทำงาน โดยจะมีการเก็บคำสั่งแรกหลังจากข้ามการทำงานไว้ทำให้ใช้เวลาน้อยลงกว่าการนำเอาคำสั่งเข้าไปป์ไลน์ วิธีการแก้ไขปัญหาในไปป์ไลน์ที่เกิดจากการควบคุมอย่างง่ายและนิยมใช้กันคือการ Stalling Pipeline จนกระทั่งคำสั่งใหม่ที่เกิดจากการนำเอาเข้ามา ส่วนการทำนายการเกิดกระโดดข้ามการทำงาน (Branch Prediction) เป็นอีกหนึ่งวิธีของการแก้ปัญหาของไปป์ไลน์ที่เกิดจากการควบคุมวิธีที่ 1 คือการออกแบบฮาร์ดแวร์โดยมีสมมติฐานว่าไม่มีการเกิดปัญหาในไปป์ไลน์ที่เกิดจากการควบคุมในระบบ ดังนั้นคำสั่งต่อจากการกระโดดข้ามการทำงานจะถูกทำงานตลอดถ้ามีการกระโดดข้ามเกิดขึ้นจริงไปป์ไลน์จะถูกหยุด คำสั่งถัดไปหลังจากการกระโดดข้ามจะเข้ามาแทน อีกวิธีหนึ่งที่การเก็บสถิติการเกิดกระโดดข้ามการทำงาน วิธีการลดเวลาหลังจากเกิดปัญหาในไปป์ไลน์ที่เกิดจากการควบคุมที่ทำกันมากในระบบโปรเซสเซอร์แบบลดทอนคำสั่ง คือการทำ Delay Branch หมายถึงการหน่วงเวลาหลังจากที่มีการใช้งานคำสั่งในการกระโดดข้ามการทำงาน

ในกรณีเช่นนี้คำสั่งถัดไปจะเกิดการทำงานเสมอ ในขณะที่ช่วงเวลาของการทำคำสั่งกระโดดข้ามจะยาวขึ้น ดังนั้นจึงไม่มีคาบเวลาที่เสียไปจำนวนของคาบเวลาในการทำงานจะน้อยที่สุด

2.5 โครงสร้างไมโครโพรเซสเซอร์แบบลดทอนคำสั่ง (RISC)

คอมพิวเตอร์ลดทอนคำสั่งหรือที่เรียกกันว่าริกส์ (RISC) มาจากคำว่า Reduced Instruction-set Computers เป็นการสร้างที่ทำให้โครงสร้างมีความเร็วในการทำงานมากขึ้น และสามารถทำงาน 1 คำสั่งให้เสร็จได้ภายใน 1 คาบสัญญาณนาฬิกาโดยคอมพิวเตอร์แบบลดทอนคำสั่งจะมีความสัมพันธ์กับโครงสร้างของไปป์ไลน์ รวมทั้งคำสั่งจะไม่มีความซับซ้อน แนวความคิดนี้ถูกสร้างเมื่อประมาณกลางยุค 1970 ที่ IBM โดย John Cocke[อ้างตาม Dave Jaggar, 1996] ทุกๆคำสั่งจะมีขนาดที่พอเหมาะในการทำงานในไปป์ไลน์ ซึ่งมีการทำงานทั้งหมด 3 สเตตไปป์ไลน์ โดยเริ่มจากรีจิสเตอร์ทั่วไปที่สามารถถูกแลตช์ได้ภายใน 1 คาบสัญญาณนาฬิกา เข้าสู่ A,B และ D รีจิสเตอร์ ซึ่งรีจิสเตอร์ A, B จะถูกส่งเข้าหน่วยคำนวณและตรรกะ แล้วผลลัพธ์ในรีจิสเตอร์ F ถูกเก็บเข้าสู่รีจิสเตอร์ทั่วไป ส่วนที่ต่อกับแคชข้อมูลแบ่งเป็น 2 ส่วนคือ ส่วนข้อมูลและส่วนของเลขที่อยู่ ซึ่งส่วนของข้อมูลสามารถมาจากหน่วยคำนวณและตรรกะ หรือส่วนของการเลื่อน ทางซ้ายมือมี พีซี ที่เพิ่มค่าขึ้นใน 1 คาบนาฬิกา แล้วส่งค่าไปเปลี่ยนแปลงเลขที่อยู่ของหน่วยความจำ



รูปที่ 2. 10 การทำงานในไปป์ไลน์

การวิเคราะห์ประสิทธิภาพของโปรเซสเซอร์ที่มีไปป์ไลน์เช่นใน โครงสร้างแบบลดทอนคำสั่ง สามารถทำได้โดยการคำนวณค่าเฉลี่ยของ รอบต่อคำสั่ง ที่ทำงาน ซึ่งจุดมุ่งหมายของการทำไปป์ไลน์คือ การให้ค่า รอบต่อคำสั่งมีค่าเป็น 1 ตัวอย่างเช่น เงื่อนไขของการกระโดดข้ามการทำงานเกิดขึ้นทุกๆ 4 คำสั่ง และโปรเซสเซอร์มีความฉลาดพอที่จะคำนวณและคาดการณ์ 80 เปอร์เซ็นต์ซึ่งใช้เวลาทั้งหมด 2 รอบ (1 รอบแรกใช้ในการคำนวณเงื่อนไข และ 1 รอบหลังใช้ในการเฟสคำสั่งใหม่เข้ามา) ดังนั้นคิดเป็น 25 เปอร์เซ็นต์ของคำสั่ง ซึ่งมีค่าเป็น 20 เปอร์เซ็นต์ ของ 2 รอบ ดังนั้นมีค่าเฉลี่ยอยู่ที่ 0.1 รอบต่อคำสั่ง คำสั่งกระโดดข้ามการทำงานมีผลต่อรอบต่อคำสั่งเป็น 1.0 – 1.1 ซึ่งถ้ามีค่าของข้อมูลที่หายไปในหน่วยความจำแคช 2 เปอร์เซ็นต์ของเวลาและเวลาในการหาข้อมูลที่หายไปในแต่ละมีค่า 10 รอบ ดังนั้นจึงต้องเพิ่มค่าอีก 0.2 รอบต่อคำสั่ง จะทำให้ผลของรอบต่อคำสั่งเป็น 1.3

เมื่อเปรียบเทียบโครงสร้างไมโครโพรเซสเซอร์แบบลดทอนคำสั่งกับแบบอื่น พบว่าคาบเวลาในการทำงานจะน้อยกว่า แต่ไมโครโพรเซสเซอร์แบบลดทอนคำสั่งจะมีคำสั่งใช้งานมากและโครงสร้างที่ซับซ้อนกว่า แต่ไมโครโพรเซสเซอร์แบบลดทอนคำสั่งสามารถเกิดปัญหาได้ในกรณีที่มีคำสั่งที่ใช้เวลาในการทำงานมากกว่าใน 1 คาบเวลา ดังนั้นจึงจำเป็นต้องใช้คาบเวลามากกว่าปกติเพื่อทำงานคำสั่งนั้นให้เสร็จ