รายงาน

เรื่อง 8 bit simple process

จัดทำโดย

นาย ศุภณัฐ โกศัลวิตร

รหัสนักศึกษา 5901012630130

เสนอ

อาจารย์ อรอุมา เทศประสิทธิ์

รายงานนี้เป็นส่วนหนึ่งของรายวิชา Computer Organization รหัสวิชา 010123134

ภาคเรียนที่ 2 ปีการศึกษา 2560 คณะวิศวกรรมศาสตร์

มหาวิทยาลัยเทคโนโลยีพระจอมเกล้าพระนครเหนือ

**คำนำ**

รายงานนี้เป็นส่วนหนึ่งของวิชา Computer Organization ซึ่งจัดทำขึ้นเพื่อศึกษาและออกแบบหน่วยประมวลผลขนาด 8 bit โดยการเขียนภาษาระดับสูง ลงมาถึงภาษาระดับล่าง(MIPS) ซึ่งภายในรายงานนี้ได้มีการออกแบบโครงสร้างลอจิกโดยการเขียนด้วย ภาษา VHDL หากมีข้อผิดพลาดประการใดขออภัยมา ณ ที่นี้ด้วย

ผู้จัดทำ

**วัตถุประสงค์**

เพื่อเป็นการออกแบบและเรียนรู้โครงสร้างภายในของหน่วยประมวลผล ซึ่งจะเริ่มตั้งแต่ภาษาระดับสูงและแปลงไปเป็น Assembly

**คำสั่งที่จากภาษาระดับสูงแปลงเป็นภาษา Assembly(MIPS)**

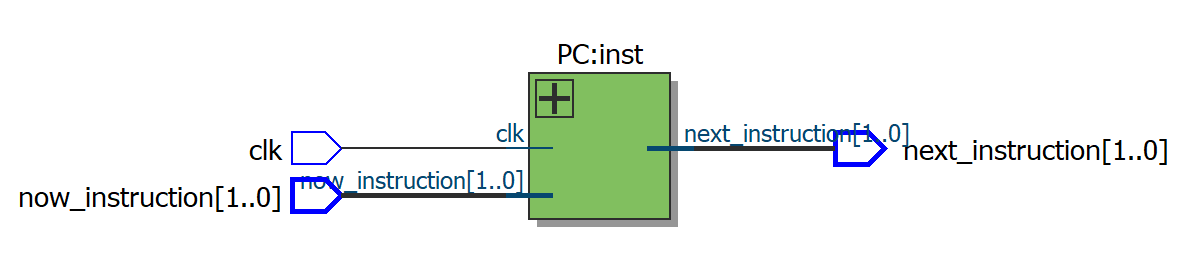
|  |  |
| --- | --- |
| **High Level Languange** | **Assembly(MIPS)** |
| A + B = C | ADD $s3 $s1 $s2 |
| A – C = D | SUB $s4 $s1 $s3 |
| A AND B = C | AND $s3 $s1 $s2 |
| A OR C = D | OR $s4 $s1 $s3 |

**องค์ประกอบมีทั้งหมด 5 ส่วนดังนี้**

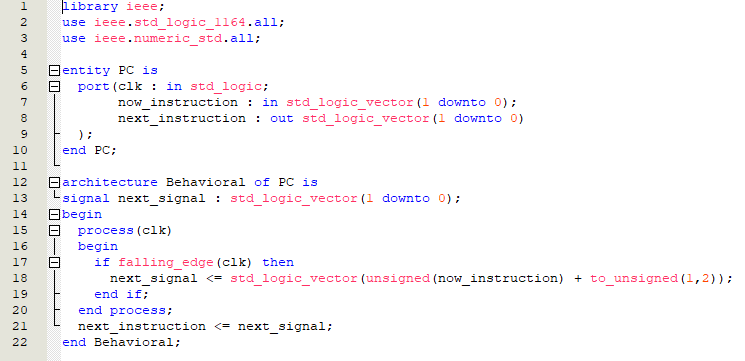
* Program counter (PC)
* Instruction memory
* Control unit
* Register
* Arithmetic and logic unit (ALU)

**รายละเอียดของแต่ละองค์ประกอบ**

**Program Counter (PC)**



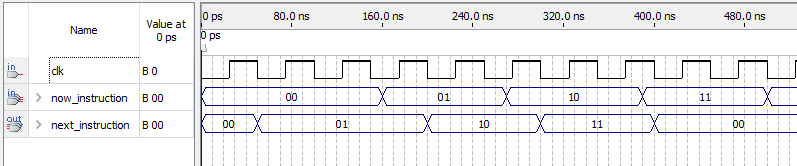
**Code VHDL**



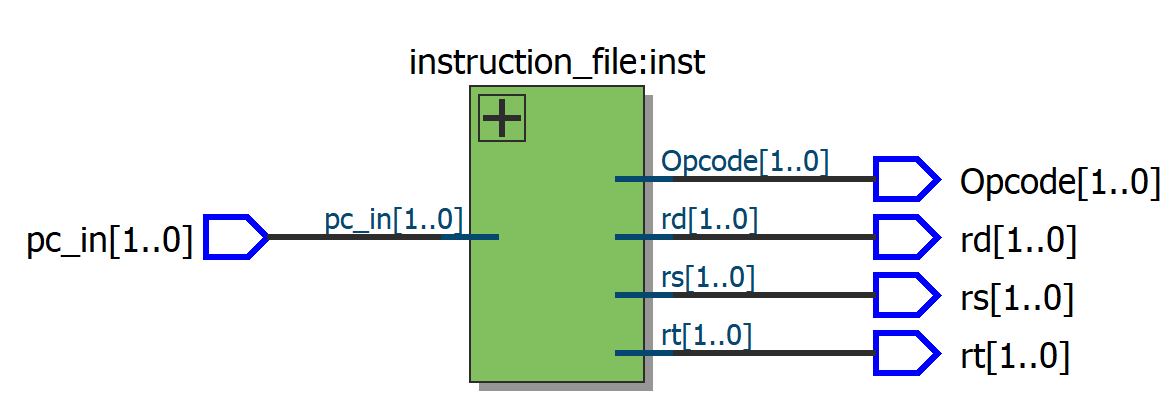
**อธิบายการทำงาน**

รับค่า input เป็น สัญญาณนาฬิกาโดยพิจารณาจากขอบขาลง และ output(next\_instruction) ที่ออกจะมีขนาด 2 bit ส่งไปที่ instruction memory พร้อมทั้งป้อนสัญญาณกลับไปเป็นสัญญาณ input อีกตัว(now\_instrucction)

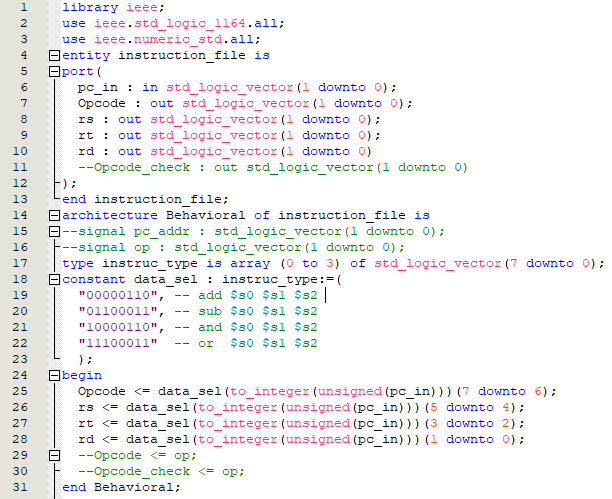
**ผลการจำลอง**



**Instruction memory**

****

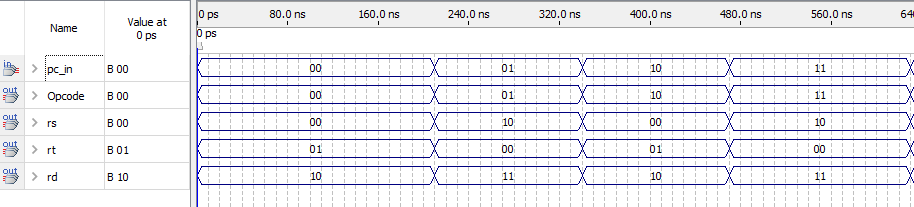
**Code VHDL**



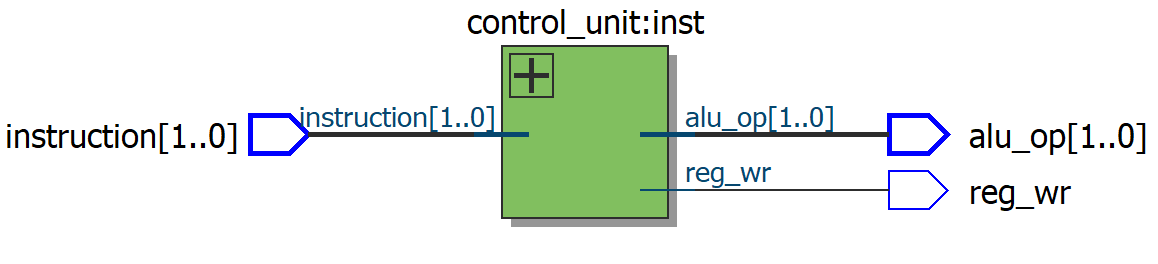
**อธิบายการทำงาน**

รับค่า input(pc\_in) มาจาก PC ซึ่งมีขนาด 2 bit และ output ออกเป็น Opcode , rs ,rt และ rd ซึ่งใน instruction นี้ จะมีการแบ่งคำสั่งออกเป็นอย่างละ 2 bit โดยที่ 2 bit ด้านซ้ายจะเป็น opcode 2 bit ถัดมาจะเป็น rs , rt และ rd ตามลำดับ

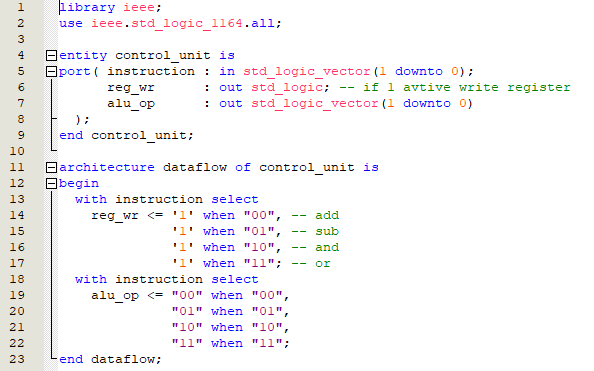
**ผลการจำลอง**



**Control unit**



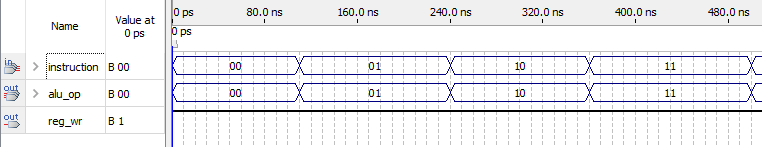
**Code VHDL**



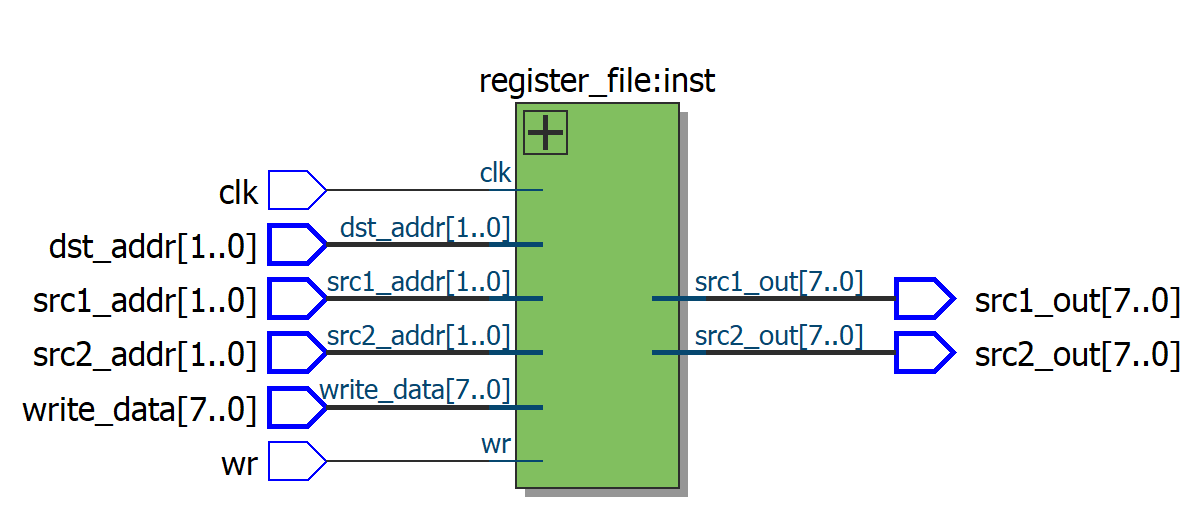
**อธิบายการทำงาน**

รับค่า input(instruction) มาเป็นขนาด 2 bit และส่ง output ออกไปได้แก่ reg\_wr และ alu\_op ซึ่ง reg\_wr จะมีสถานะลอจิกคือ 0 และ 1 แต่ในที่นี้จะมีสถานะลอจิกเป็น 1 ทั้งหมดซึ่งจะส่งสถานะลอจิก 1 ไปที่ register file เพื่อเป็นการกำหนดให้ active การ write data ลงไปใน register ส่วน alu\_op จะทำหน้าที่ส่ง address ขนาด 2 bit ไปให้ Arithmetic and logical unit (ALU) เพื่อเป็นการกำหนดว่าจะทำคำสั่งใดใน ALU

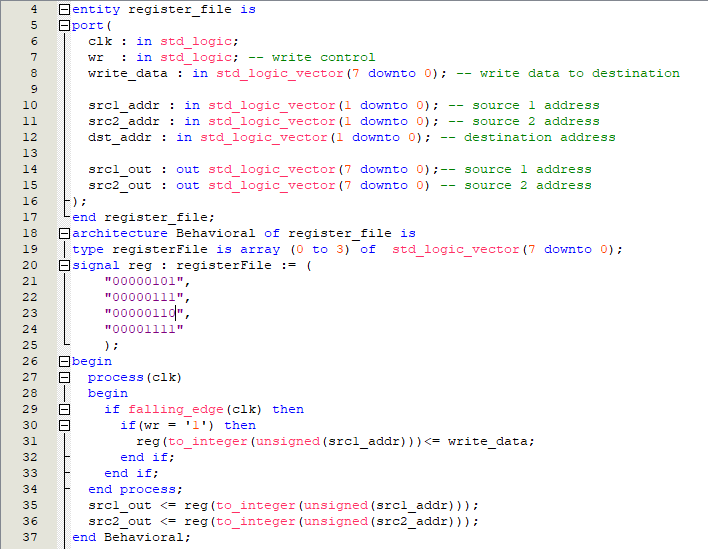
**ผลการจำลอง**



**Register**



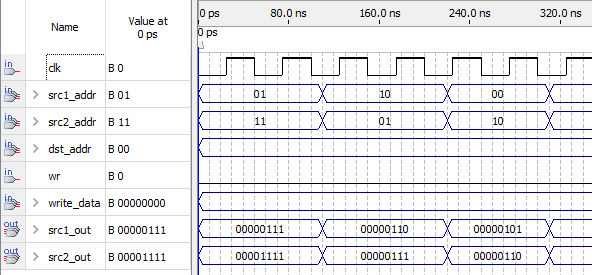
**Code VHDL**



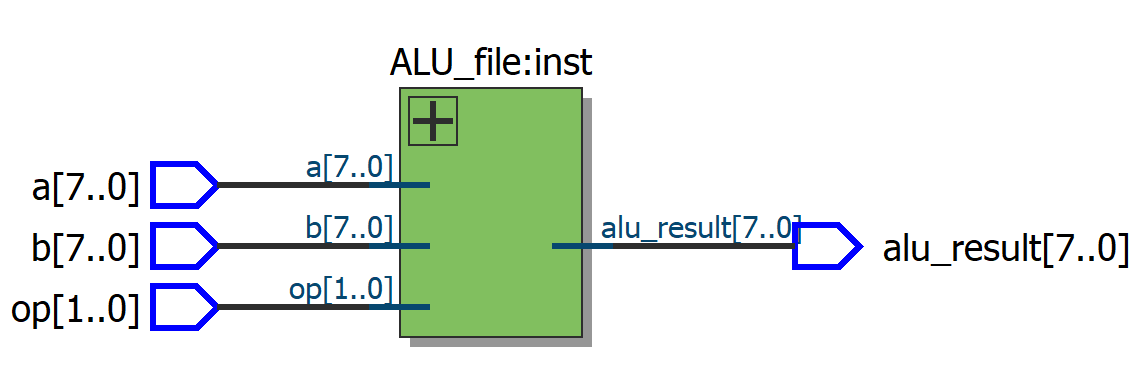
**อธิบายการทำงาน**

รับค่า input มาเป็นสัญญานาฬิกา (clk) โดยพิจารณาที่ขอบขาลงนอกจากนี้ยังรับ input มาเป็น address ของ rs, rt, rd (จาก code จะใช้เป็น src1\_addr, src2\_addr และ dst\_addr ตามลำดับ) และเมื่อ wr มีสถานะลอจิกเป็น 1 จะกำให้ write\_data เขียนไว้ที่ src1\_addr และหลังจากนั้นก็ส่ง output ขนาด 8 bit ออกมาคือ src1\_out และ src2\_out ตามลำดับ ซึ่งน register นี้ได้ทำการกำหนดข้อมูลไว้ภายในแล้ว

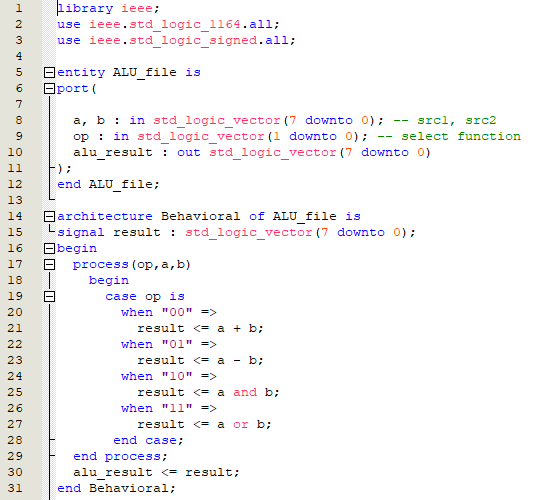
**ผลการจำลอง**



**Arithmetic and logical unit (ALU)**



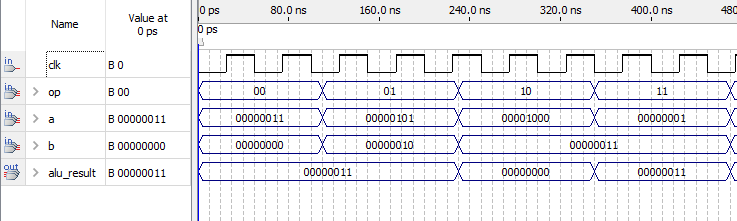
Code VHDL



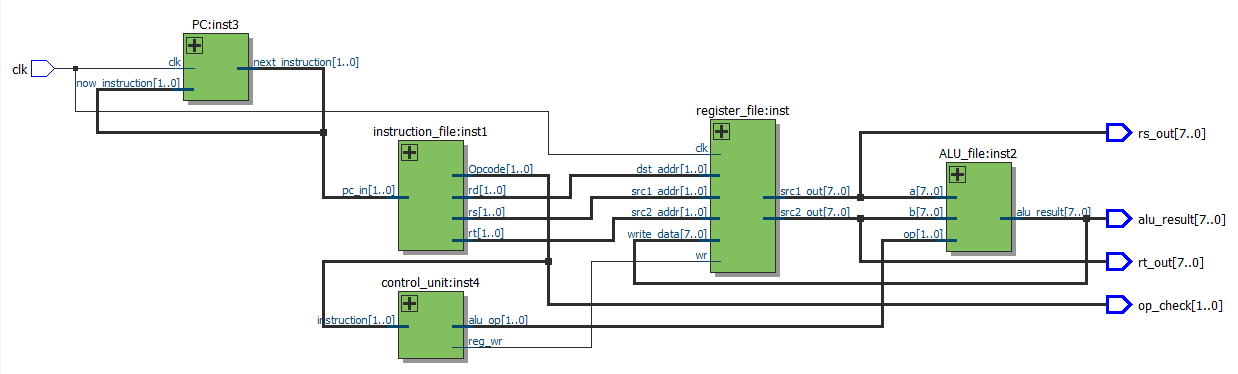
**อธิบายการทำงาน**

รับค่า input เป็นข้อมูลขนาด 8 bit และ op ขนาด 2 bit ซึ่ง op คือที่รับมาจะเป็น address เพื่อระบุว่าจะทำงานคำสั่งใด ถ้าเป็น “00” จะเป็น add “01” จะเป็น sub “10” จะเป็น or และ “11” จะเป็น and เพื่อคำสั่งเสร็จแล้วจะทำการส่ง output ออกไปเป็น ขนาด 8 bit

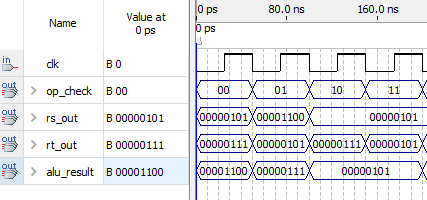
**ผลการจำลอง**

****

**ภาพจำลอง RTL Viewer**



**ผลการจำลองรวมทุกองค์ประกอบ**

****

|  |  |  |  |
| --- | --- | --- | --- |
| 00 | 01 | 10 | 11 |
| ADD | SUB | AND | OR |

เมื่อพิจารณาสัญญาณ clk ตมขอบขาลง เมื่อ op\_check เป็น “01” จะทำการ ADD กันระหว่าง rs\_out กับ rt\_out และได้ผลลัพธ์เป็น alu\_reult หลังจากนั้น เมื่อ op\_check เป็น “01” จะทำการนำผลลัพธ์เมื่อ op\_check มีสถานะเป็น “00” มาใช้ต่อโดยเก็บค่าไว้ที่ rs\_out ซึ่งสถานะ “01” คือการทำ SUB หลังจากนั้นก็จะทำ SUB ระหว่าง rs\_out และ rt\_out เมื่อเสร็จแล้ว op\_check จะไปที่ สถานะ “10” และทำคำสั่ง AND ซึ่ง rs\_out ในที่นี้จะไปนำข้อมูลมาจาก register file และหลังจากนั้นเมื่อทำคำสัง AND เสร็จ op\_check ก็จะมีสถานะเป็น “11” ซึ่งเป็นคำสั่ง OR หลังจากนั้น rc\_out จะเป็นค่าของ alu\_result เมื่อตอนทำคำสั่ง AND และหลังจากที่คำสั่ง OR เสร็จแล้ว op\_check ก็จะกลับไปที่สถานะ “00” และทำงานตามลำดับที่ได้กล่าวไปในข้างต้น