

CEDT Final Project 2568 Digital Logic 2110252: Mini CPU

ปล่อย Testcase 20 เรียบร้อยครับ

Template สามารถเข้าถึงได้จาก [TEMPLATE Project 2568.dia](#)

Template ทุก Testcase

https://drive.google.com/file/d/1-lbcwmmF0I7HENbjkT9V_YOa5zmdLla8/view?usp=sharing

Testcase full (Excel)

https://docs.google.com/spreadsheets/d/1MF2TL-rYU_UMsCkPACtszQV-eJuxBxd-/edit?usp=sharing&ouid=115188683733264074142&rtpof=true&sd=true

คะแนนการรันจะนับถึงเที่ยงคืนวันนี้นะครับ

ตอนนี้ทุก test case มีการทำผ่านครบทั้ง 29 cases และวันนี้ครับ แต่ยังไม่มีกลุ่มไหนทำถูกต้องทั้งหมด
แจ้งตอน 21:00

มีทำได้เต็ม 3 กลุ่มแล้วนะครับ ข้อมูลตอน 07:00

เปิด grader ด้วยเดิมแล้วนะครับ

มี Test case ของ 2 คำสั่งพิเศษนี้ ให้ใน template เดิมด้วยนะครับ

<https://drive.google.com/file/d/1phE6zkWXZH67jJy4E8BivXvaieF7YzfN/view?usp=sharing>

กลุ่มไหนส่งแล้วผ่านทั้งหมดเรียบร้อย ให้รายงานในหน้าสุดท้ายของไฟล์นี้ด้วยนะครับ

ให้จองลำดับการนำเสนอ (เข้าด้วยอีเมลนิสิต)

https://docs.google.com/spreadsheets/d/1VMYgTfqbvO5k-T_jqUrLrMDFM8EkkyuT6hb0qVfP3TI/edit?usp=sharing

- นิสิตต้องส่งรายชื่อสมาชิกผ่าน MCV และถึงจะขึ้นชื่อกลุ่มให้เลือก (อันนี้ไม่ได้อัตโนมัตินะครับ ต้องดึงรายชื่อออกรายชื่อที่ ดังนั้นให้ใส่ข้อมูลให้ครบถ้วนภายใน 09:15 เดียวผมจะดึงข้อมูลให้ช่วงเวลา

นั้น) (<https://www.courseville.com/?q=courseville/worksheet/74041/1793002>)

ให้นิสิตสร้าง CPU โดยใช้โปรแกรม Digital โดยช่วงแรกอรับคำสั่งเพื่อโหลดโปรแกรมเข้ามาเก็บใน RAM memory ขนาด 256 x 14 bits (pRAM - Program RAM) และทำการคำสั่งโดยคำสั่งแรกจะอยู่ที่ address 0x00 ทำไปจนเจอคำสั่งให้หยุด จากนั้นร่องกว่าจะได้รับสัญญาณให้ส่งคำตอบ จึงแสดงผลลัพธ์ที่เก็บไว้ใน RAM memory ขนาด 256 x 8 bits (rRAM - Result RAM) ในตัวແண່ງ address 0x00 - 0x0F ออกทาง output และ 7-segment จำนวน 2 ตัว

วงจรมี input คือ

- M ขนาด 8 bits พารามิเตอร์ที่ 1
 - N ขนาด 8 bits พารามิเตอร์ที่ 2
 - progIn ขนาด 14 bits เป็น data input ที่จะส่งโปรแกรมเข้ามาทีละคำสั่งจนครบ ก่อนจะให้รอ สัญญาณเพื่อเริ่มทำงาน
 - reset ขนาด 1 bit เมื่อได้ 1 ให้ reset การทำงานของวงจร และ เคลี้ยร์ค่าของ rRAM ให้เป็น 0 ทั้งหมด โดยจะมี clock ให้ทำงานส่วนนี้อย่างน้อย 20 clocks
 - progLoad ขนาด 1 bit เมื่อได้ 1 ให้เริ่มอ่านค่าจาก progIn ทีละคำสั่ง ไปเก็บใน pRAM เริ่มตั้งแต่ address 0x00 ไปจนกระทั่ง สัญญาณ progLoad เป็น 0 ซึ่งคำสั่งที่มากที่สุดจะเป็น 256 คำสั่ง
 - start ขนาด 1 bit เมื่อได้ 1 ให้เริ่มทำงานตาม pRAM
 - result ขนาด 1 bit เมื่อได้ 1 ให้เริ่มส่งคำตอบที่อยู่ใน rRAM
 - clk ขนาด 1 bit เป็นสัญญาณ clock สำหรับใช้ในการให้จังหวะ
- ส่วนนี้สำหรับเป็นตัวช่วยในการทำงานของวงจร ในการทดสอบ grader จะไม่ได้ส่งส่วนนี้ให้โดยตรง
- [pRAM ขนาด 256 x 14 bits] สำหรับเก็บโปรแกรม ที่ได้รับเข้ามา สามารถเลือกเป็นแบบใดก็ได้

วงจรมี output คือ

- valid ขนาด 1 bit เป็น 1 เมื่อทำงานตามโปรแกรมเสร็จเรียบร้อยพร้อมส่งคำตอบ
 - done ขนาด 1 bit เป็น 1 เมื่อการทำงานเสร็จสิ้นและส่งผลคำตอบเรียบร้อย หรือตอนเริ่มต้น ครั้งแรกที่พร้อมรับคำสั่ง reset / progLoad / start
 - output ขนาด 8 bits เป็นค่า output ที่ได้จากการทำงาน โดยแสดงผลลัพธ์ที่อยู่ใน rRAM เป็น เลขฐาน 16 โดยจะแสดงค่าห้องจากไดร์บสัญญาณ result
- ส่วนนี้สำหรับเป็นตัวช่วยในการทำงานของวงจร ในการทดสอบ grader จะไม่ได้อ่านส่วนนี้โดยตรง
- [7-segment 2 ตัว] แสดงค่าผลลัพธ์ของ output สำหรับให้ตรวจสอบผลลัพธ์
 - [rRAM ขนาด 256 x 8 bits] สำหรับเก็บคำตอบ สามารถเลือกเป็นแบบใดก็ได้

ข้อมูลเพิ่มเติมสำหรับการพัฒนา CPU

- สามารถใช้อุปกรณ์ อะไรก็ได้เพิ่มเติมที่มีอยู่ใน Digital แบบมาตรฐาน
- จำนวนบรรทัดของคำสั่งจะอยู่ในช่วง 1 - 256 คำสั่ง
- คำสั่งโปรแกรมที่ส่งมาให้จัดเก็บเข้า pRAM โดยเมื่อทำงานให้อ่านคำสั่งของโปรแกรมตามลำดับ โดยให้ทำงานตั้งแต่บรรทัดแรกไปจนเจอคำสั่งที่ให้หยุดการทำงาน
- CPU นี้จะใช้การคำนวณแบบ 8 bits เป็นหลัก
- CPU นี้จะต้องออกแบบให้เป็น Multiple Cycle CPU หรือ Pipeline ไม่อนุญาตให้ทำเป็น Single Cycle CPU
- การ implement ส่วนของ control unit จะทำเป็น sequential logics หรือใช้ microprogram ก็ได้
- ระบบตัวเลขสำหรับคำนวณทางคณิตศาสตร์จะเป็นแบบ 2's complement ยกเว้นจะระบุเป็นอย่างอื่น
- แต่ละคำสั่ง จะออกแบบให้ใช้จำนวน clock เท่ากันหรือไม่เท่ากันก็ได้
- ต้องมีการออกแบบ register เพื่อเก็บค่าสถานะหรือข้อมูลต่างๆเพิ่มเติมเอง ตามความเหมาะสม
- "ไม่จำเป็นต้องทำได้ครบถ้วนคำสั่ง บาง test case อาจจะใช้เพียงไม่กี่คำสั่งก็ได้"
- กำหนดให้ใช้ positive edge clock ในการ Synchronous วงจร

แต่ละคำสั่งจะมีโครงสร้างคำสั่งดังนี้

| | | | | | | | | | | | | | |
|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|---|---|---|---|---|---|---|---|---|---|

| | |
|---------------|----------------|
| Opcode 6 bits | Operand 8 bits |
|---------------|----------------|

โครงสร้างของ instruction ขนาด 14 bits แบ่งเป็น opcode 6 bits และ operand 8 bits

Opcode และค่าอธิบายเป็นไปดังตารางด้านล่าง

| Opcode | คำสั่ง | ความหมาย |
|--------|-----------------------------|---|
| 000000 | NOPE | ไม่ต้องทำงานอะไร ข้ามไปทำงานคำสั่งถัดไป |
| 000001 | accA ← Operand | นำค่า 8 bits ของ operand ไปเก็บไว้ใน accA |
| 000010 | accB ← Operand | นำค่า 8 bits ของ operand ไปเก็บไว้ใน accB |
| 000011 | accA ← accB | นำค่าจาก accB ไปเก็บไว้ใน accA |
| 000100 | accB ← accA | นำค่าจาก accA ไปเก็บไว้ใน accB |
| 000101 | regC ← accA | นำค่าจาก accA ไปเก็บไว้ใน regC |
| 000110 | accA ← regC | นำค่าจาก regC ไปเก็บไว้ใน accA |
| 000111 | regD ← accA | นำค่าจาก accA ไปเก็บไว้ใน regD |
| 001000 | accA ← regD | นำค่าจาก regD ไปเก็บไว้ใน accA |
| 001001 | regC ← M | นำค่าจาก input M ไปเก็บไว้ใน regC |
| 001010 | regC ← N | นำค่าจาก input N ไปเก็บไว้ใน regC |
| 001011 | regD ← M | นำค่าจาก input M ไปเก็บไว้ใน regD |
| 001100 | regD ← N | นำค่าจาก input N ไปเก็บไว้ใน regD |
| 001101 | regC ← M regD ← N | นำค่าจาก input M ไปเก็บไว้ใน regC นำค่าจาก input N ไปเก็บไว้ใน regD |
| 001110 | accA ← regC accB ← regD | นำค่าจาก regC ไปเก็บไว้ใน accA นำค่าจาก regD ไปเก็บไว้ใน accB |
| 001111 | accA ← pRAM_adr[Operand] | นำค่าจาก pRAM ใน address ที่ระบุโดย operand ไปเก็บไว้ใน accA โดยจัดเก็บเฉพาะ 8 bits ด้านขวาเท่านั้น |
| 010000 | accA ← rRAM_adr[Operand] | นำค่าจาก rRAM ใน address ที่ระบุโดย operand ไปเก็บไว้ใน accA |
| 010001 | rRAM_adr[Operand] ← accA | นำค่าจาก accA ไปเก็บไว้ใน rRAM ใน address ที่ระบุโดย operand |
| 010010 | Jump to address | ข้ามไปทำคำสั่งใน address ตามที่ระบุใน Operand แบบไม่มี |

| Opcode | คำสั่ง | ความหมาย |
|--------|-------------------------------------|--|
| | Operand | เงื่อนไข |
| 010011 | Jump to address Operand if eq | ข้ามไปท่าคำสั่งใน address ตามที่ระบุใน Operand ถ้า equal flag เป็น 1 |
| 010100 | Jump to address Operand if gr | ข้ามไปท่าคำสั่งใน address ตามที่ระบุใน Operand ถ้า greater flag เป็น 1 |
| 010101 | Jump to address Operand if le | ข้ามไปท่าคำสั่งใน address ตามที่ระบุใน Operand ถ้า lesser flag เป็น 1 |
| 010110 | Jump to address Operand if eq or gr | ข้ามไปท่าคำสั่งใน address ตามที่ระบุใน Operand ถ้า equal flag หรือ greater flag เป็น 1 |
| 010111 | Jump to address Operand if eq or le | ข้ามไปท่าคำสั่งใน address ตามที่ระบุใน Operand ถ้า equal flag หรือ lesser flag เป็น 1 |
| 100000 | $accA \leftarrow accA + accB$ | นำค่าจาก accA มาบวกกับ accB และไปเก็บที่ accA (ไม่ต้องสนใจกรณีผลบวกเกินขอบเขต) คำนวณแบบ 2's complement |
| 100001 | $accA \leftarrow accA - accB$ | นำค่าจาก accA มาลบกับ accB และไปเก็บที่ accA (ไม่ต้องสนใจกรณีผลลบเกินขอบเขต) คำนวณแบบ 2's complement |
| 100010 | $accA \leftarrow accA * accB$ | นำค่าจาก accA[3..0] มาคูณกับ accB[3..0] และไปเก็บที่ accA (คิด 4 bits ดังนั้นจะมองเป็นเลขบวกอย่างเดียว) |
| 100011 | $accA \leftarrow accA / accB$ | นำค่าจาก accA คิดแบบ binary ไม่ดู signed bit มาหารกับ accB และไปเก็บที่ accA |
| 100100 | $accA \leftarrow accA \% accB$ | นำค่าจาก accA คิดแบบ binary ไม่ดู signed bit มา mod กับ accB และไปเก็บที่ accA |
| 100101 | $accA \leftarrow accA ^ accB$ | นำค่าจาก accA[2..0] ยกกำลัง accB[2..0] และไปเก็บที่ accA (ไม่ต้องสนใจกรณีผลลัพธ์เกินขอบเขต) |
| 101000 | $accA \leftarrow NOT(accA)$ | กลับบิตของ accA และเก็บไว้ที่ accA |
| 101001 | $accA \leftarrow accA AND accB$ | นำค่าจาก accA มา bitwise AND กับ accB และไปเก็บที่ accA |
| 101010 | $accA \leftarrow accA OR accB$ | นำค่าจาก accA มา bitwise OR กับ accB และไปเก็บที่ accA |
| 101011 | $accA \leftarrow accA XOR accB$ | นำค่าจาก accA มา bitwise XOR กับ accB และไปเก็บที่ accA |
| 101100 | $accA \leftarrow accA << accB$ | นำค่าจาก accA มา logical shift left ตามค่าของ accB[2..0] และไปเก็บที่ accA โดย shift left ไม่เกิน 7 bits |

| Opcode | คำสั่ง | ความหมาย |
|--------|--|---|
| 101101 | $accA \leftarrow accA \lll accB$ | นำค่าจาก accA มา rotate logical shift left ตามค่าของ accB[2..0] และไปเก็บที่ accA โดย shift left ไม่เกิน 7 bits |
| 101110 | $accA \leftarrow accA \ggg accB$ | นำค่าจาก accA มา logical shift right ตามค่าของ accB[2..0] และไปเก็บที่ accA โดย shift right ไม่เกิน 7 bits |
| 101111 | $accA \leftarrow accA \ggg accB$ | นำค่าจาก accA มา rotate logical shift right ตามค่าของ accB[2..0] และไปเก็บที่ accA โดย shift right ไม่เกิน 7 bits |
| 110000 | accA CMP accB | เทียบค่า accA กับ accB คำนวณแบบ 2's complement - ถ้า $accA == accB$ ค่า equal flag จะเป็น 1 - ถ้า $accA > accB$ ค่า greater flag จะเป็น 1 - ถ้า $accA < accB$ ค่า lesser flag จะเป็น 1 |
| 110001 | isPrime(accA) | ตรวจสอบว่า accA เป็นจำนวนเฉพาะหรือไม่ ถ้า - accA เป็นจำนวนเฉพาะ ค่า equal flag จะเป็น 1 - กรณีนี้ไม่กระทบกับ greater flag และ lesses flag ให้คิด accA และ accB แบบเลขฐาน 2 ปกติ ไม่มีเลขลบ |
| 110010 | accB accA | ทดสอบว่า accA ที่เป็นตัวตั้งหารด้วย accB ลงตัวหรือไม่ - ถ้าลงตัวให้ equal flag เป็น 1 - กรณีนี้ไม่กระทบกับ greater flag และ lesses flag ให้คิด accA และ accB แบบเลขฐาน 2 ปกติ ไม่มีเลขลบ เช่น $accA = 20$ และ $accB = 5$ จะได้ว่า $5 20$ คือ 20 หารด้วย 5 ลงตัว จะทำให้ equal flag เป็น 1 |
| 110011 | $rRAM[0x0E:0x0F] \leftarrow LCM(rRAM_adr[Operand[7:4]], rRAM_adr[Operand[3:0]])$ | คำนวณหาค่าคูณร่วมน้อย LCM(m,n) โดย m คือค่าใน rRAM ตำแหน่งตาม operand[7:4] n คือค่าใน rRAM ตำแหน่งตาม operand[3:0] ให้คิดตัวเลขแบบ binary เป็นจำนวนเต็มบวกเท่านั้น นำผลลัพธ์ที่ได้เก็บไว้ที่ rRAM[0x0E] และ rRAM[0x0F] โดยค่า most significant บิต result[15:8] เก็บที่ rRAM[0x0E] โดยค่า least significant บิต result[7:0] เก็บที่ rRAM[0x0F] https://en.wikipedia.org/wiki/Least_common_multiple |
| 110100 | $rRAM[0x0A:0x0B] \leftarrow FAC(accA[2:0])$ | คำนวณหา Factorial(n) โดย n คือค่าใน accA เฉพาะบิต [2:0] ดังนั้นค่าของ n จะอยู่ระหว่าง 0 - 7 เท่านั้น นำผลลัพธ์ที่ได้เก็บไว้ที่ rRAM[0x0A] และ rRAM[0x0B] โดยค่า most significant บิต result[15:8] เก็บที่ rRAM[0x0A] โดยค่า least significant บิต result[7:0] เก็บที่ rRAM[0x0B] ในคำสั่งนี้ให้ใช้ การวนลูปผ่าน ASM ไม่อนุญาตให้ใช้ ROM https://en.wikipedia.org/wiki/Factorial |

| Opcode | คำสั่ง | ความหมาย |
|--------|---|--|
| 110101 | $rRAM[0x09] \leftarrow \max(pRAM[accA:accB])$ | <p>ค่านวนหาค่าที่มากที่สุด ของค่าใน pRAM[7:0] (คิดเฉพาะ 8 มิต ด้านขวาเท่านั้น) ต่อหนึ่งที่ระบุโดย accA ถึง accB เช่น ถ้า accA เป็น 10 และ accB เป็น 20 จะหมายถึงหาค่าที่มากที่สุดจาก pRAM[0x0A : 0x14] นำค่าตอบ ที่ได้ไปเก็บที่ rRAM[0x09]</p> <p>ให้ใช้การเปรียบเทียบแบบ 2's complement</p> <p>ในการนี้โจทย์จะตั้งให้ $accA \leq accB$ เชื่อ</p> |
| 111111 | STOP | หยุดการทำงาน ไม่ต้องทำคำสั่งถัดไป และรอสัญญาณ result |

accA, accB, regC, regD คือ Accumulator A, Accumulator B, Register C, Register D ตามลำดับ โดยมีขนาด 8 bits

การทำงานในทีม

- ให้ทำงานเป็นกลุ่ม กลุ่มละ 3-4 คน โดยนิสิตสามารถจับกลุ่มกันเองได้
 - กลุ่ม 2 คนก็ได้ พอนุ่ม แต่ถ้ามีเพื่อนไม่มีกลุ่ม ก็อาจจะขอให้เพื่อนร่วมกลุ่มด้วย
- นิสิตสามารถปรึกษากันระหว่างกลุ่มได้ แต่ห้ามคัดลอก
- สามารถใช้ generative AI ในการช่วยวิเคราะห์และพัฒนาวางแผนจาร์ได้

การให้คะแนน (draft)

- เอกสารด้านเทคนิคไม่เกิน 10 หน้า (10 คะแนน)
 - แนวคิดการออกแบบ CPU
 - ASM Chart หรือ FSM Chart
 - การออกแบบและพัฒนาส่วน Data Path
 - การออกแบบและพัฒนาส่วน Control Unit
- ส่วนการนำเสนอ (10 คะแนน)
 - สุ่มตัวแทนที่จะมานำเสนอ และแสดงว่าทุกคนควรรู้เรื่องและอธิบายได้
 - ให้จัดทำสไลด์ประกอบการนำเสนอ
 - ระยะเวลานำเสนอ 5-10 นาที
- การทำงานตาม test case (80 คะแนน)
 - ในแต่ละ โปรแกรมทดสอบจะมีความแตกต่างกันทั้งเรื่องจำนวนคำสั่งที่ใช้ บรรทัดของคำสั่ง
 - สำหรับโปรแกรมทดสอบเดียวกัน ก็จะมีหลาย test case เพื่อทดสอบสิ่งสัญญาณ input เช่น M, N, reset, start, result ในเวลาที่ไม่เหมือนกันเพื่อตัดการตอบสนองของวงจรที่สร้างขึ้น
- ทุกๆ สมาชิกที่เกินกว่า 4 คน จะถูกหักคะแนนละ 5 คะแนน (-5 คะแนน)

Extra Credit

- กลุ่ม (จำนวนไม่เกิน 5 คน) ที่สามารถส่ง wangjgrader ได้ถูกต้องทุก test case จำนวน 5 กลุ่ม แรก (ในกลุ่มต้องทำเอง ห้ามทูลวิธีเอาของกลุ่มอื่นมาดัดแปลงแล้วส่ง) จะได้รางวัลพิเศษ (ต้องแสดงรหัสและชื่อสมาชิกในไฟล์ .dig และแสดงหน้าที่ของแต่ละคน โดยยึดรายชื่อตามไฟล์ครั้งแรก ที่ส่งผ่านครบทุก test case
 - ถือว่าสอบผ่านได้ S หังกลุ่ม โดยไม่มีพิจารณาค่าคะแนนแล้วปะและคะแนนสอบ
 - คนที่เป็นหลักในการออกแบบและพัฒนาจำนวนไม่เกิน 2 คนต่อกลุ่ม จะได้ S* คือใน Certificate ยอดเยี่ยมประจำวิชา

ตัวอย่างการทำงาน

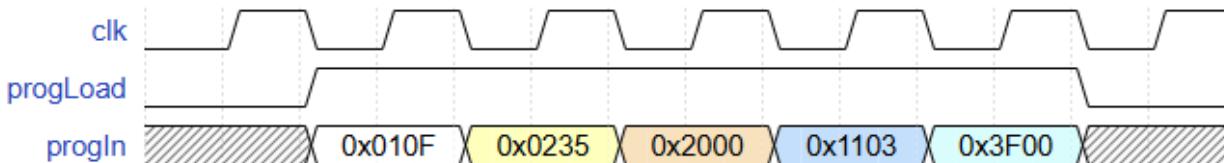
กำหนดให้ข้อมูลโปรแกรมทดสอบเป็นดังนี้

| | |
|----------------------|-------------------------------------|
| 0b 00 0001 0000 1111 | accA \leftarrow Operand |
| 0b 00 0010 0011 0101 | accB \leftarrow Operand |
| 0b 10 0000 0000 0000 | accA \leftarrow accA + accB |
| 0b 01 0001 0000 0011 | rRAM_adr[Operand] \leftarrow accA |
| 0b 11 1111 0000 0000 | STOP |

เริ่มต้นจะมีสัญญาณ reset เพื่อให้วงจรทำการตั้งค่าของระบบ โดยจะมี clock อย่างน้อยจำนวน 20 clock ก่อนจะมีสัญญาณ progLoad

รอสัญญาณ progLoad เป็น 1 จะให้เริ่มรับคำสั่งทีละคำสั่งผ่านทาง progIn ไปเก็บไว้ใน pRAM โดยเริ่มต้นที่ address 0x00 และค่อยๆเพิ่ม address ทีละ 1 จนกว่าสัญญาณ progLoad เป็น 0 ซึ่งหมายถึง program ได้ถูกส่งให้ครบถ้วนแล้ว

ในการนี้จะส่งมาทั้งหมด 5 คำสั่ง ให้จัดเก็บที่ pRAM address 0x00 - 0x04



รอสัญญาณ start เป็น 1 (จะเป็น 1 สักๆ) จะให้เริ่มทำการคำสั่งตามลำดับใน pRAM

เริ่มจากให้ accA \leftarrow 15

accB \leftarrow 53

accA \leftarrow 15 + 53 = 68

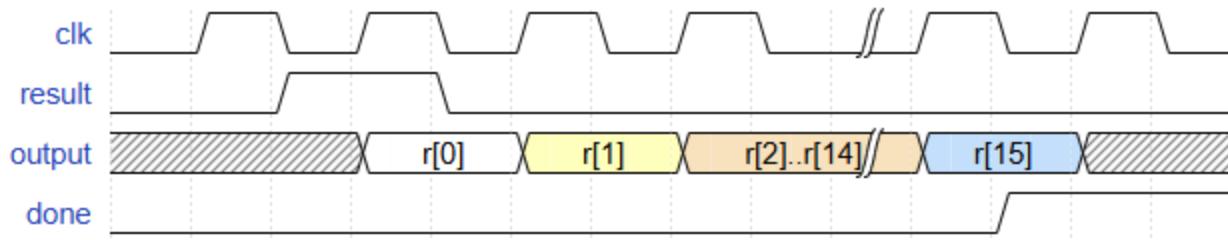
rRAM_adr[3] \leftarrow 68

เมื่อทำงานเสร็จ ให้กำหนดสัญญาณ valid = 1

จากนั้นรอลุนได้สัญญาณ result = 1 จึงเริ่มส่งค่าของ rRAM ตั้งแต่ address 0x00 ไปจนถึง 0x0F ซึ่งจะได้

| | |
|------------|-------------------------------------|
| rRAM[0x00] | \rightarrow 0 |
| rRAM[0x01] | \rightarrow 0 |
| rRAM[0x02] | \rightarrow 0 |
| rRAM[0x03] | \rightarrow 68 \rightarrow 0x44 |
| ... | |
| rRAM[0x0F] | \rightarrow 0 |

เมื่อแสดงผลครบ ให้ส่งสัญญาณ done ออกเป็น 1 เพื่อรอสัญญาณการทำงานใหม่ โดยต้องสามารถรับสัญญาณ reset, progLoad และ start ได้อย่างต่อเนื่อง



Template

Template สามารถเข้าถึงได้จาก [TEMPLATE Project 2568.dig](#)

Testcase

ไฟล์ที่ไว้สร้าง test case สามารถทดลองใช้ได้จาก excel (ให้โหลดลงที่เครื่องมาทดลอง ตัว testcase จะอยู่ที่คอลัมน์ P)

| 1 | Program | Test1 | ทดสอบการโหลดโปรแกรมและภาระการทำงาน | | | | | | | | | | | | Meaning | Assembly | testcase format | | | | | | | | | | | |
|----|---------|-------|------------------------------------|---|--------|-------|----------|-------|--------|-----|-------|------|--------|-------------------------------|------------------|----------|-----------------|--------|-------|----------|-------|--------|-----|-------|------|--------|---|--|
| | | | M | N | progIN | reset | progLoad | start | result | clk | valid | done | output | Machine code | | M | N | progIN | reset | progLoad | start | result | clk | valid | done | output | | |
| 45 | 0 | 0 | 271 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0x | accA += Operand | 00000100001111 | 0 | 0 | 271 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | x | | |
| 46 | 0 | 0 | 271 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0x | ทดสอบค่าที่ 1 เข้า pRAM[0x00] | | 0 | 0 | 271 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | x | | |
| 47 | 0 | 0 | 565 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0x | accA += Operand | 00001000110101 | 0 | 0 | 565 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | x | | |
| 48 | 0 | 0 | 565 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0x | ทดสอบค่าที่ 2 เข้า pRAM[0x01] | | 0 | 0 | 565 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | x | | |
| 49 | 0 | 0 | 8192 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0x | accA += accA + accB | 10000000000000 | 0 | 0 | 8192 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | x | | |
| 50 | 0 | 0 | 8192 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0x | ทดสอบค่าที่ 3 เข้า pRAM[0x02] | | 0 | 0 | 8192 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | x | | |
| 51 | 0 | 0 | 4355 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0x | RAM_add[Operand] = accA | 0100010000001111 | 0 | 0 | 4355 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | x | | |
| 52 | 0 | 0 | 4355 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0x | ทดสอบค่าที่ 4 เข้า pRAM[0x03] | | 0 | 0 | 4355 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | x | | |
| 53 | 0 | 0 | 16128 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0x | STOP | 11111100000000 | 0 | 0 | 16128 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | x | | |
| 54 | 0 | 0 | 16128 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0x | ทดสอบค่าที่ 5 เข้า pRAM[0x04] | | 0 | 0 | 16128 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | x | | |
| 55 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0x | เริ่มส่ง start ให้รันโปรแกรม | | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | x | | |
| 56 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0x | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | x | | | |
| 57 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0x | | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | x | | |
| 58 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0x | ทำงานตามโปรแกรม | | | | 0 | 0 | 0 | 0 | 0 | 0 | 1 | x | 0 | x | |

[Testcase template 2568.xlsx](#)

Test case นะครับ

- Test case ใน grader ที่ให้ทดสอบจะสอดคล้องกับ test case ในไฟล์ [Testcase template 2568.xlsx](#)
- 01_P01T01 ทดสอบการโหลดโปรแกรมและการส่งค่าทั่วไป
- 02_P01T02 request result เป็นครั้งที่ 2 หลังจากส่ง output ออกมานแล้ว
- 03_P02T01 Test simple jump
- 04_P03T01 pRamLoad_Test
- 05_P03T02 pRamLoad_Test
- 06_P04T01 isPrime_Test
- 07_P05T01 isPrime + CMP
- 08_P06T01 ทดสอบการอ่านค่า M และ N
- 09_P06T02 ทดสอบการอ่านค่า M และ N / แล้วก็ทดสอบการ run ใหม่โดยไม่ได้โหลดโปรแกรมใหม่
- 10_P07T01 ทดสอบคำสั่งพิเศษ LCM
- 11_P07T02 ทดสอบคำสั่งพิเศษ LCM โดยให้ค่าคำตอบจะเกิน 8 บิต และเก็บตัวตนไว้ที่ address 0xF ซึ่งจะໄwake ก็จะได้คำตอบด้วย
- 12_P08T01 ทดสอบการ + - * / % ^
- 13_P08T02 ทดสอบการ + - * / % ^ โดยการสั่งแก้ไขงานส่วนของโปรแกรมแล้วทำงานเลย
- 14_P09T01 ทดสอบ bitwise operation NOT AND OR XOR <<
- 15_P09T02 ทดสอบ bitwise operation NOT AND OR XOR << และ reset ระหว่างทำงาน
- 16_P01T03 ทดสอบการโหลดโปรแกรมและการส่งค่าทั่วไป และก็ทดลอง request result ข้าม
- ทดสอบใช้ regC regD
- 17_P02T02 Jump ทุกรูปแบบ และมีการหยุดระหว่างทาง
- 18_P03T03 pRamLoad_Test extend
- 19_P04T02 isPrime_Test Extend
- 20_P04T03 accB | aacA ทดสอบว่า accA หารด้วย accB ลงตัวหรือไม่

- 21_P05T02 isPrime_Test + flag condition
- 22_P06T03 ทดสอบการอ่านค่า M และ N ใส่ regC regD
- 23_P07T03 ทดสอบคำสั่งพิเศษ LCM
- 24_P08T03 ทดสอบการ + - * / % ^ โดยการสั่งแก็บงานส่วนของโปรแกรมแล้วทำงานเลย
- 25_P09T03 ทดสอบ bitwise operation NOT AND OR XOR << <<< และ reset ระหว่างทำงาน
- 26_P10T01 ทดสอบคำสั่งพิเศษ factorial
- 27_P10T02 ทดสอบคำสั่งพิเศษ factorial
- 28_P11T01 ทดสอบคำสั่งพิเศษ max
- 29_P11T02 ทดสอบคำสั่งพิเศษ max