CEDT Final Project 2568 Digital Logic 2110252: Mini CPU

ให้นิสิตสร้าง CPU โดยใช้โปรแกรม Digital โดยช่วงแรกรอรับคำสั่งเพื่อโหลดโปรแกรมเข้ามาเก็บใน RAM memory ขนาด 256 x 14 bits (pRAM - Program RAM) แล้วทำตามคำสั่งโดยคำสั่งแรกจะอยู่ที่ address 0x00 ทำไปจนเจอคำสั่งให้หยุด จากนั้นรอจนกว่าจะได้รับสัญญาณให้ส่งคำตอบ จึงแสดงผลลัพธ์ที่เก็บไว้ใน RAM memory ขนาด 256 x 8 bits (rRAM - Result RAM) ในตำแหน่ง 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 ก่อนจะให้รอสัญญาณเพื่อเริ่มทำงาน
- reset ขนาด 1 bit เมื่อได้ 1 ให้ reset การทำงานของวงจร และ เคลียร์ค่าของ 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 หรือตอนเริ่มต้นครั้งแรกที่พร้อมรับคำสั่ง reset / progLoad / start
- output ขนาด 8 bits เป็นค่า output ที่ได้จากการทำงาน โดยแสดงผลลัพธ์ที่อยู่ใน rRAM เป็นเลขฐาน 16 โดยจะแสดงค่าหลังจากได้รับสัญญาณ 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
- ไม่จำเป็นต้องทำได้ครบทุกคำสั่ง 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 และคำอธิบายเป็นไปดังตารางด้านล่าง

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

Opcode	คำสั่ง	ความหมาย
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 Operand	ข้ามไปทำคำสั่งใน address ตามที่ระบุใน Operand แบบไม่มีเงื่อนไข
010011	Jump to address Operand if eq	ข้ามไปทำคำสั่งใน address ตามที่ระบุใน Operand ถ้า equal flag เป็น 1
010100	Jump to address Operand if gr	ข้ามไปทำคำสั่งใน address ตามที่ระบุใน Operand ถ้า greater flag เป็น 1
010101	Jump to address Operand if le	ข้ามไปทำคำสั่งใน address ตามที่ระบุใน Operand ถ้า lesser flag เป็น 1
010110	Jump to address Operand if eq or gr	ข้ามไปทำคำสั่งใน address ตามที่ระบุใน Operand ถ้า equal flag หรือ greater flag เป็น 1
010111	Jump to address Operand if eq or le	ข้ามไปทำคำสั่งใน address ตามที่ระบุใน Operand ถ้า equal flag หรือ lesser flag เป็น 1
100000	accA ← accA + accB	นำค่าจาก accA มาบวกกับ accB แล้วไปเก็บที่ accA (ไม่ต้องสนใจกรณีผลบวกเกินขอบเขต) คำนวณแบบ 2’s complement
100001	accA ← accA - accB	นำค่าจาก accA มาลบกับ accB แล้วไปเก็บที่ accA (ไม่ต้องสนใจกรณีผลลบเกินขอบเขต) คำนวณแบบ 2’s complement
100010	accA ← accA * accB	นำค่าจาก accA[3..0] มาคูณกับ accB[3..0] แล้วไปเก็บที่ accA (คิด 4 bits ดังนั้นจะมองเป็นเลขบวกอย่างเดียว)
100011	accA ← accA / accB	นำค่าจาก accA คิดแบบ binary ไม่ดู signed bit มาหารกับ accB แล้วไปเก็บที่ accA
100100	accA ← accA % accB	นำค่าจาก accA คิดแบบ binary ไม่ดู signed bit มา mod กับ accB แล้วไปเก็บที่ accA
100101	accA ← accA ^ accB	นำค่าจาก accA[2..0] มายกกำลัง accB[2..0] แล้วไปเก็บที่ accA (ไม่ต้องสนใจกรณีผลลัพธ์เกินขอบเขต)
101000	accA ← NOT(accA)	กลับบิตของ accA แล้วเก็บไว้ที่ accA
101001	accA ← accA AND accB	นำค่าจาก accA มา bitwise AND กับ accB แล้วไปเก็บที่ accA

Opcode	คำสั่ง	ความหมาย
101010	accA ← accA OR accB	นำค่าจาก accA มา bitwise OR กับ accB แล้วไปเก็บที่ accA
101011	accA ← accA XOR accB	นำค่าจาก accA มา bitwise XOR กับ accB แล้วไปเก็บที่ accA
101100	accA ← accA << accB	นำค่าจาก accA มา logical shift left ตามค่าของ accB[2..0] แล้วไปเก็บที่ accA โดย shift left ไม่เกิน 7 bits
101101	accA ← accA <<< accB	นำค่าจาก accA มา rotate logical shift left ตามค่าของ accB[2..0] แล้วไปเก็บที่ accA โดย shift left ไม่เกิน 7 bits
101110	accA ← accA >> accB	นำค่าจาก accA มา logical shift right ตามค่าของ accB[2..0] แล้วไปเก็บที่ accA โดย shift right ไม่เกิน 7 bits
101111	accA ← accA >>> accB	นำค่าจาก accA มา rotate logical shift right ตามค่าของ accB[2..0] แล้วไปเก็บที่ 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 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] ← 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] ←	คำนวณหา Factorial(n) โดย

Opcode	คำสั่ง	ความหมาย
	FAC(accA[2:0])	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
110101	rRAM[0x09] ← max(pRAM[accA:accB])	คำนวณหาค่าที่มากที่สุด ของค่าใน pRAM[7:0] (คิดเฉพาะ 8 บิตด้านขวาเท่านั้น) ตำแหน่งที่ระบุโดย accA ถึง accB เช่น ถ้า accA เป็น 10 และ accB เป็น 20 จะหมายถึงหาค่าที่มากที่สุดจาก pRAM[0x0A : 0x14] นำคำตอบ ที่ได้ไปเก็บที่ rRAM[0x09] ให้ใช้การเปรียบเทียบแบบ 2's complement ในกรณีนี้โจทย์จะตั้งให้ accA <= accB เสมอ
111111	STOP	หยุดการทำงาน ไม่ต้องทำคำสั่งถัดไป แล้วรอสัญญาณ result

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

การทำงานในทีม

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

การให้คะแนน (draft)

- เอกสารด้านเทคนิคไม่เกิน 10 หน้า (10 คะแนน)
  - แนวคิดการออกแบบ CPU
  - ASM Chart หรือ FSM Chart
  - การออกแบบและพัฒนาส่วน Data Path
  - การออกแบบและพัฒนาส่วน Control Unit

- ส่วนการนำเสนอ (10 คะแนน)
  - สุ่มตัวแทนที่จะมานำเสนอ แสดงว่าทุกคนควรรู้เรื่องและอธิบายได้
  - ให้จัดทำสไลด์ประกอบการนำเสนอ
  - ระยะเวลานำเสนอ 5-10 นาที

- การทำงานตาม test case (80 คะแนน)
    - ในแต่ละ โปรแกรมทดสอบจะมีความแตกต่างกันทั้งเรื่องจำนวนคำสั่งที่ใช้ บรรทัดของคำสั่ง
    - สำหรับโปรแกรมทดสอบเดียวกัน ก็จะมีหลาย test case เพื่อทดสอบส่งสัญญาณ input เช่น M, N, reset, start, result ในเวลาที่ไม่เหมือนกันเพื่อดูการตอบสนองของวงจรที่สร้างขึ้น

- ทุกๆ สมาชิกที่เกินกว่า 4 คน จะถูกหักคนละ 5 คะแนน (-5 คะแนน)

Extra Credit

- กลุ่ม (จำนวนไม่เกิน 5 คน) ที่สามารถส่งวงจรผ่าน grader ได้ถูกต้องทุก test case จำนวน 5 กลุ่มแรก (ในกลุ่มต้องทำเอง ห้ามทุจริตเอาของกลุ่มอื่นมาดัดแปลงแล้วส่ง) จะได้รางวัลพิเศษ (ต้องแสดงรหัสและชื่อสมาชิกในไฟล์ .dig และแสดงหน้าที่ของแต่ละคน โดยยึดรายชื่อตามไฟล์ครั้งแรกที่ส่งผ่านครบทุก test case
    - ถือว่าสอบผ่านได้ S ทั้งกลุ่ม โดยไม่พิจารณาคะแนนแล็ปและคะแนนสอบ
    - คนที่เป็นหลักในการออกแบบและพัฒนาจำนวนไม่เกิน 2 คนต่อกลุ่ม จะได้ S* คือใบ Certificate ยอดเยี่ยมประจำวิชา

ตัวอย่างการทำงาน

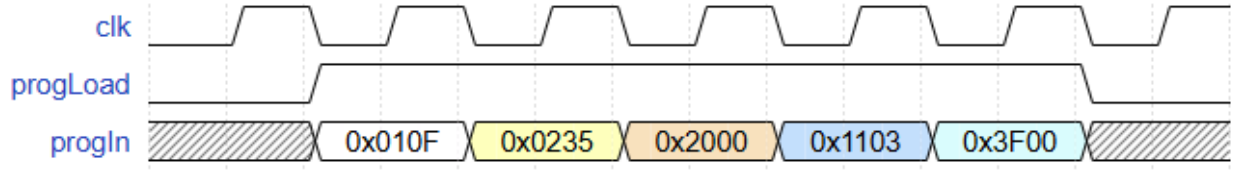
กำหนดให้ข้อมูลโปรแกรมทดสอบเป็นดังนี้

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

เริ่มต้นจะมีสัญญาณ reset เพื่อให้วงจรทำการตั้งต้นค่าของระบบ โดยจะมี clock อย่างน้อยจำนวน 20 clock ก่อนจะมีสัญญาณ progLoad

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

ในกรณีนี้จะส่งมาทั้งหมด 5 คำสั่ง ให้จัดเก็บที่ pRAM address 0x00 - 0x04

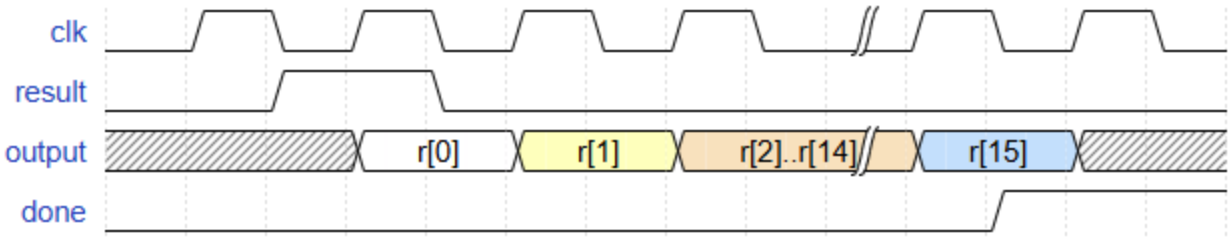


รอสัญญาณ start เป็น 1 (จะเป็น 1 สั้นๆ) จะให้เริ่มทำคำสั่งตามลำดับใน pRAM
เริ่มจากให้ accA ← 15
accB ← 53
accA ← 15 + 53 = 68
rRAM_adr[3] ← 68
เมื่อทำงานเสร็จ ให้กำหนดสัญญาณ valid = 1

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

rRAM[0x00] → 0
rRAM[0x01] → 0
rRAM[0x02] → 0
rRAM[0x03] → 68 → 0x44
…
rRAM[0x0F] → 0

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


clk
result
output
r[0]
r[1]
r[2]..r[14]
r[15]
done

Template

Template สามารถเข้าถึงได้จาก [TEMPLATE Project 2568.dig]

Testcase

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

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	Program1	Test1	ทดสอบการโหลดโปรแกรมและการส่งค่าทั่วไป													testcase format
2	M	N	progIN	reset	progLoad	start	result	clk	valid	done	output		Meaning	Assembly	Machine code	M N progIN reset progLoad start result clk valid done output
45	0	0	271	0	1	0	0	0	0	0	x			accA ← Operand	000001 0000 1111	0 0 271 0 1 0 0 0 0 0 x
46	0	0	271	0	1	0	0	1	0	0	x		โหลดคำสั่งที่ 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	x			accB ← Operand	000010 0011 0101	0 0 565 0 1 0 0 0 0 0 x
48	0	0	565	0	1	0	0	1	0	0	x		โหลดคำสั่งที่ 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	x			accA ← accA + accB	100000 0000 0000	0 0 8192 0 1 0 0 0 0 0 x
50	0	0	8192	0	1	0	0	1	0	0	x		โหลดคำสั่งที่ 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	x			RAM_add[Operand] ← accA	010001 0000 0011	0 0 4355 0 1 0 0 0 0 0 x
52	0	0	4355	0	1	0	0	1	0	0	x		โหลดคำสั่งที่ 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	x			STOP	111111 0000 0000	0 0 16128 0 1 0 0 0 0 0 x
54	0	0	16128	0	1	0	0	1	0	0	x		โหลดคำสั่งที่ 5 เข้า pRAM[0x04]			0 0 16128 0 1 0 0 1 0 0 x
55	0	0	0	0	0	1	0	0	0	0	x					0 0 0 0 0 1 0 0 0 0 x
56	0	0	0	0	0	1	0	1	0	0	x		รับคำสั่ง start ให้เริ่มทำงาน			0 0 0 0 0 1 0 1 0 0 x
57	0	0	0	0	0	0	0	0	x	0	x					0 0 0 0 0 0 0 0 x 0 x
58	0	0	0	0	0	0	0	1	x	0	x		ทำงานตามโปรแกรม			0 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 ออกมาแล้ว
- 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 ใหม่โดยไม่ได้ โหลดโปรแกรมใหม่
- 10_P07T01 ทดสอบคำสั่งพิเศษ LCM
- 11_P07T02 ทดสอบคำสั่งพิเศษ LCM โดยให้ค่าคำตอบจะเกิน 8 บิต และเก็บตัวตันไว้ที่ address 0x0F ซึ่งจะไว้เก็บคำตอบด้วย
- 12_P08T01 ทดสอบการ + - * / % ^
- 13_P08T02 ทดสอบการ + - * / % ^ โดยการสั่งแก้บางส่วนของโปรแกรมแล้วทำงานเลย
- 14_P09T01 ทดสอบ bitwise operation NOT AND OR XOR <<
- 15_P09T02 ทดสอบ bitwise operation NOT AND OR XOR << แล้วก็ reset ระหว่างทำงาน

- 16_P01T03 ทดสอบการโหลดโปรแกรมและการส่งค่าทั่วไป แล้วก็ทดลอง request result ซ้ำ ทดลองใช้ regC regD
- 17_P02T02 Jumpทุกรูปแบบ 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 + - * / % ^ โดยการสั่งแก้บางส่วนของโปรแกรมแล้วทำงานเลย
- 25_P09T03 ทดสอบ bitwise operation NOT AND OR XOR << <<< แล้วก็ reset ระหว่างทำงาน
- 26_P10T01 ทดสอบคำสั่งพิเศษ factorial
- 27_P10T02 ทดสอบคำสั่งพิเศษ factorial
- 28_P11T01 ทดสอบคำสั่งพิเศษ max
- 29_P11T02 ทดสอบคำสั่งพิเศษ max

กำหนดการและแนวทางในการส่ง

วันประกาศโครงงาน
- ให้นิสิตจับกลุ่มและแจ้งรายละเอียดกลุ่ม โดยให้สมาชิกเพียงคนเดียวเป็นตัวแทนในการกรอก (รวมถึงต้องใช้ account นี้ในการ update และส่งข้อมูลตลอด)

- ให้กรอกชื่อทีมและสมาชิกภายใน วันจันทร์ที่ 20 ต.ค. 68 เวลา 14:00
- เริ่มออกแบบและพัฒนาวงจร

ก่อนวันกำหนดส่งระยะหนึ่งภายในวันที่ 27 ต.ค. 68
- แจ้งตัวอย่าง test case ที่ใช้ในการทดสอบเบื้องต้น (ซึ่งนิสิตจะทราบคำสั่งที่มี จำนวนคำสั่ง จำนวน clock มากที่สุดที่ยอมให้ใช้)
- เปิด grader ให้ทดลองส่งบาง test case

ก่อนวันกำหนดส่ง 29 ต.ค. 68 06:00
- ทยอยเพิ่มและแจ้ง Test case (เกือบ) ทั้งหมดที่ใช้ในการทดสอบ
- ให้จองลำดับการนำเสนอ

ก่อนกำหนดส่งเล็กน้อย 29 ต.ค. 68
เวลาประมาณ 19:00
- เพิ่ม Test case ทั้งหมดที่ใช้ในการทดสอบ
- เริ่มนับเวลาส่งอย่างเป็นทางการเพื่อคัดเลือกทีมที่ส่งได้ครบทุก test case ตาม Extra Credit

ช่วงการนำเสนอ
- ให้ Upload เอกสารสำหรับนำเสนอให้เสร็จ
- นำเสนอกลุ่มละประมาณ 5-10 นาที
- โดยคนนำเสนอและตอบคำถามจะถูกสุ่มจากสมาชิกคนใดคนหนึ่ง
- นำเสนอทีละกลุ่ม ตามลำดับการจอง

หลังการนำเสนอ
- มีเวลาให้ทำการพัฒนา CPU เพิ่มเติม
- Grader จะปิดรับการส่งในวันที่ 30 ตุลาคม 2568 ตอนเที่ยงคืน
- ให้ Upload เอกสารต่างๆ เช่นไฟล์สำหรับนำเสนอ รายงาน ไฟล์โปรแกรม และ capture หน้าจอคะแนน grader ผ่านทาง MCV ให้เสร็จก่อนเที่ยงคืน

หลังวันกำหนดส่ง 1-2 พฤศจิกายน 2568
- พักผ่อนหรือเตรียมตัวสำหรับวิชาถัดไป
- ขอให้นิสิตทุกคนโชคดีและสนุกกับการเรียน

มีคำถามตรงไหนเพิ่มเติมสามารถ เพิ่ม comment มาได้เลยนะครับ

1. เวลา reset แล้วค่าใน register กับ flags จะเปลี่ยนไหมครับ
    - ควร reset ให้เรียบร้อยครับ โดยค่า default ควรจะเป็น 0

2. การสุ่มตัวแทนสุ่มจำนวนกี่คนคะ เป็นสลับกันตอบคำถามมั้ยคะ และรูปเล่มมีตัวอย่างให้มั้ยคะ
    - สุ่ม 1 คน ในกลุ่ม ทั้งนำเสนอและตอบคำถาม คนเดียวรวดครับ
    - เล่มไม่มีตัวอย่างครับ

3. Data Input 14 bit ชื่อว่า progIn หรือ progIN นะครับ พอดีว่าในเอกสารกับใน test case ไม่ตรงกันครับ (ผมใช้ progIN แล้วส่งใน grader ผ่านนะครับ น่าจะ progIN แหละ)
    - progIn ครับ ผมแก้ไขใน grader template และ excel แล้ว
4. สามารถใช้ single cycle processor ในการออกแบบได้หรือเปล่าครับ
    - ไม่ได้ครับ

5. reset ต้องเคลียร์ทุกบิตใน rRam เป็น 0 ทุกช่องเลยหรอครับ หรือแค่ 20 บิตแรกก็พอครับ
    - ควรจะ clear ครับ โดยเฉพาะ 16 bytes แรก

6. สามารถใช้อุปกรณ์ อะไรก็ได้เพิ่มเติมที่มีอยู่ใน Digital แบบมาตรฐาน:
   อันนี้ ROM ใช้ได้ใช่ไหมครับ ผมจำได้ว่าอาจารย์ไม่ให้ใช้ ใน lab/สอบ แต่ว่าครั้งนี้ (project) สามารถใช้ได้ใช่ไหมครับ
    - ใช้ได้ครับ รอบนี้เลือกได้เลย

7. ASM/FSM/Datapath จำเป็นต้องใช้โปรแกรมวาดไหมครับ หรือวาดมือได้
    - วาดมือได้ครับ

8. progLoad จะเป็น 1 ตั้งแต่ตอนที่ clock เป็น 0 ตลอดรึเปล่าครับ หรือว่าสามารถเกิดเหตุการณ์ที่ progLoad เริ่มเป็น 1 ตอนที่ clock เป็น 1 พอดีได้
    - จะเป็น 1 ตอนที่ clock เป็น 0 เพื่อรอไว้ก่อนครับ

9. การส่ง 5 กลุ่มแรกต้องส่งที่ไหนครับผม ต้องส่งอะไรบ้าง และจะเผยแพร่ test case อีกทีกี่โมงครับ
    - ตัวเต็มก็ส่งผ่าน grader ครับ แต่ตอนนี้ยังไม่ให้เห็นนะ
    - จะปล่อย test case เรื่อยๆ ครับ
    - คาดว่าจะปล่อยตัวเต็มในวันพุธเวลา 18:00

10. มีวิธีใส่ค่าลง RAM พร้อมกันหลาย address มั้ยครับ
    - ไม่แน่ใจนะ ถ้าใครทราบมาแชร์หน่อย
11. คะแนนส่วนการทำงานตาม test case (80 คะแนน) นี่ดูจาก grader อย่างเดียวรึป่าวครับ
    - ใช่ครับ แต่จำนวน test case ยังมีเพิ่มเรื่อยๆ เท่าที่เปิดคือเปิด แต่ตัวเต็มก็จะมีมากกว่านี้

12. Test Case ใน Grader ตรงกับใน Template หรือไม่ครับ เนื่องจากเห็นว่า Test Case ที่ให้มามีข้อผิดพลาดบางส่วน จึงอยากทราบว่าใน Grader เป็นแบบเดียวกันหรือไม่ หากแตกต่างจะทราบได้อย่างไรว่าไม่เป็นข้อผิดพลาดที่ Test Case แต่หากเหมือนกันก็ดีครับ
    - เหมือนกันครับ
    - ช่วงนี้ทดสอบ test case ต่างๆ ถ้าเจอตรงไหนไม่ถูกต้อง ก็แจ้งมาได้ครับ

13. ทำไมไม่การประกาศเลยคะว่ามีเทสเคสใหม่ เพื่อนๆรู้ได้ไงว่ามีมาใหม่🥺🥺
    - กำลังเพิ่ม testcase แล้วก็ทดสอบไป เลยยังไม่ได้แจ้งทางการ ขออภัยด้วยนะครับ
    - ที่เพื่อนๆทราบน่าจะเพราะว่า ส่งวงจรผ่าน grader เลยเห็นว่ามีการเพิ่ม test case อยู่

14. หลังจากทำงานเสร็จ (done หลังการทำงานครั้งแรกแล้ว) ต้อง reset register และ flag ไหมครับ
    - ไม่ต้องครับ
    - รอคำสั่ง control line ถัดๆไป

15. จะมี testcase ที่ result หลังจาก ทำงานเสร็จเลยไหมครับ เช่น หลังจาก opcode = 111111 1 cylcle แล้ว result = 1 เลย
    - ไม่มีครับ