การทำรหัสลับแบบ AES บนหน่วยประมวลผลหลายแกนเพื่อเพิ่มประสิทธิภาพ

นายศุภชัย ทองสุข

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมซอฟต์แวร์ ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2556

AN IMPLEMENTATION OF AES ALGORITHM ON MULTICORE PROCESSORS TO IMPROVE EFFICIENCY

Mr. Supachai Thongsuk

A Thesis Submitted in Partial Fulfillment of the Requirements
for the Degree of Master of Science Program in Software Engineering
Department of Computer Engineering
Faculty of Engineering
Chulalongkorn University
Academic Year 2013

หัวข้อวิทยานิพนธ์	การทำรหัสลับแบบ AES บนหน่วยประมวลผลหลายแกนเพื่อเพิ่มประสิทธิภาพ
โดย	นายศุภชัย ทองสุข
สาขาวิชา	วิศวกรรมซอฟต์แวร์
อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก	ศาสตราจารย์ ดร. ประภาส จงสถิตย์วัฒนา

คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย อนุมัติให้นับวิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญามหาบัณฑิต

.....คณบดีคณะวิศวกรรมศาสตร์

(ศาสตราจารย์ ดร. บัณฑิต เอื้ออาภรณ์)

คณะกรรมการสอบวิทยานิพนธ์

.....ประธานกรรมการ

(อาจารย์ ดร. ยรรยง เต็งอำนวย)

.....อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก

(ศาสตราจารย์ ดร. ประภาส จงสถิตย์วัฒนา)

.....กรรมการภายนอกมหาวิทยาลัย

(รองศาสตราจารย์ ดร. วรเศรษฐ สุวรรณิก)

ศุภชัย ทองสุข : การทำรหัสลับแบบ AES บนหน่วยประมวลผลหลายแกนเพื่อเพิ่มประสิทธิภาพ. (AN IMPLEMENTATION OF AES ALGORITHM ON MULTICORE PROCESSORS TO IMPROVE EFFICIENCY) อ.ที่ปรึกษาวิทยานิพนธ์หลัก: ศ. ดร. ประภาส จงสถิตย์วัฒนา, 68 หน้า.

ขั้นตอนวิธีการเข้ารหัส AES (Advanced Encryption Standard) คือ การเข้ารหัสแบบกลุ่ม ซึ่งถูกเผยแพร่โดยสถาบันมาตรฐานและเทคโนโลยีแห่งสหัฐอเมริกา (NIST) ในปี ค.ศ. 2001 การเข้ารหัสได้ถูกใช้ในระบบรักษาความปลอดภัยต่างๆ และปัจจุบันถูกใช้ทั่วโลก การเข้ารหัสแบบ AES นั้นส่วนใหญ่จะทำโดยใช้หน่วยประมวลผลแบบแกนเดียว เพื่อที่จะประมวลผลข้อมูลขนาดใหญ่ งานวิจัยนี้จึงนำเสนอขั้นตอนวิธีการเข้ารหัส AES บนหน่วยประมวลผลหลายแกน โดยวิธีการขนานกันในข้อมูลขนาดใหญ่ ประมวลผลพร้อมกันเพื่อเพิ่มความเร็ว

ภาควิชา วิศวกรรมคอมพิวเตอร์ ลายมือชื่อนิสิต .....

สาขาวิชา วิศวกรรมซอฟต์แวร์ ลายมือชื่อ อ.ที่ปรึกษาวิทยานิพนธ์หลัก .....

ปีการศึกษา 2556

5570998721 : MAJOR SOFTWARE ENGINEERING
KEYWORDS: CRYPTOGRAPHY / AES / MULTICORE PROCESSOR

SUPACHAI THONGSUK: AN IMPLEMENTATION OF AES ALGORITHM ON MULTICORE PROCESSORS TO IMPROVE EFFICIENCY. ADVISOR: PROF. PRABHAS CHONGSTITVATTANA, Ph.D., 68 pp.

AES (Advanced Encryption Standard) algorithm is a block encryption algorithm, established by the U.S. National Institute of Standards and Technology (NIST) in 2001. It has been adopted by many data security systems and now used worldwide. Most of AES implementations are for single-core processors. To achieve high performance for large data, this work proposed an AES algorithm for multi-core processors. Using parallelism inherent in large data, all cores are working concurrently to speed up the task.

Department: Computer Engineering Student's Signature .....
Field of Study: Software Engineering Advisor's Signature .....
Academic Year: 2013

กิตติกรรมประกาศ

วิทยานิพนธ์นี้สำเร็จลงด้วยความกรุณาเป็นอย่างสูงของศาสตราจารย์ ดร.ประภาส จงสถิตย์วัฒนา อาจารย์ที่ปรึกษาวิทยานิพนธ์ ซึ่งได้ให้โอกาส ความรู้และคำแนะนำในการทำวิทยานิพนธ์ตลอดจนความเมตตาและความอดทนในการตรวจผลงานของข้าพเจ้า ได้แก่ โครงร่างวิทยานิพนธ์ ผลงานวิจัยภาษาไทยและภาษาอังกฤษ และวิทยานิพนธ์ ทำให้ผลงานทุกชิ้นสำเร็จลุล่วงเป็นอย่างดี

ขอขอบพระคุณ อาจารย์ ดร.ยรรยง เต็งอำนวย ประธานกรรมการสอบวิทยานิพนธ์ และรองศาสตราจารย์ ดร.วรเศรษฐ สุวรรณิก กรรมการสอบวิทยานิพนธ์ที่กรุณาให้คำแนะนำและชี้แนะแนวทางที่เป็นประโยชน์ต่อการทำวิทยานิพนธ์ในครั้งนี้

ขอขอบพระคุณอาจารย์ทุกท่านที่ให้ความรู้และคำแนะนำที่เป็นประโยชน์ รวมถึงความเมตตาและความเอาใจใส่มาโดยตลอด

ขอขอบพระคุณครอบครัว ได้แก่ บิดามารดา ที่ให้กำลังใจในทุกเรื่อง และสอนให้มองแต่ในแง่ดีและมองเห็นโอกาสและความเป็นไปได้ จนทำให้ข้าพเจ้ามีผลงานที่สำเร็จได้

ขอขอบคุณเพื่อนร่วมชั้นเรียนวิศวกรรมซอฟต์แวร์ และเพื่อนสาขาวิชาวิศวกรรมซอฟต์แวร์ภาคนอกเวลาราชการทุกคนที่ให้ความช่วยเหลือ แจ้งข่าวสารของมหาวิทยาลัยและการประชุมวิชาการที่เป็นประโยชน์ รวมถึงมิตรภาพและกำลังใจที่มีให้เสมอ

สารบัญ

หน้า

หน้า

สารบัญตาราง

หน้า

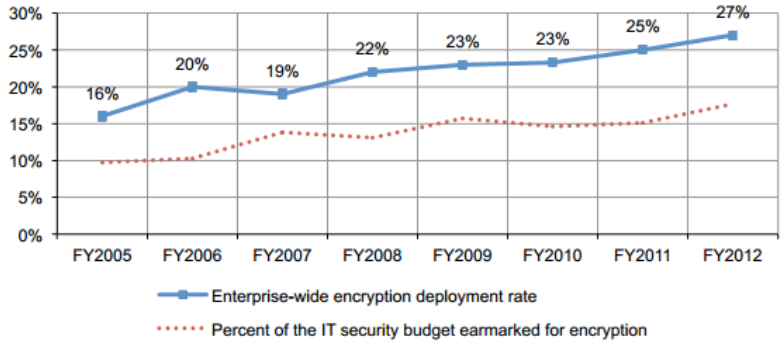
สารบัญภาพ

หน้า

บทที่ 1
บทนำ

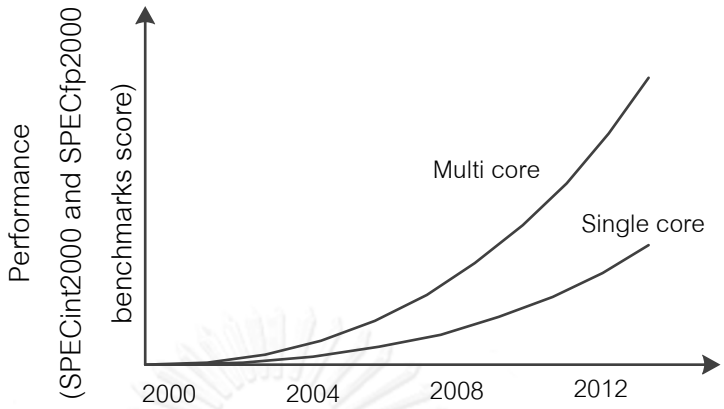
1.1 ความเป็นมาและความสำคัญของปัญหา

AES, Advanced Encryption Standard [1] เป็นวิทยาการเข้ารหัสลับที่ถูกใช้อย่างแพร่หลายโดยออกแบบโดยชาวเบลเยี่ยมชื่อ Joan Daemen และ Vincent Rijmen ซึ่งถูกกำหนดโดยสถาบันมาตรฐานและเทคโนโลยีแห่งสหรัฐอเมริกาหรือ National Institute of Standards and Technology ในปี ค.ศ. 2001 ให้เป็นวิธีเข้ารหัสข้อมูลมาตรฐานของประเทศสหรัฐอเมริกา โดยใช้การคำนวณที่ซับซ้อนตามลำดับขั้นตอน ดังนั้นจึงต้องใช้เวลามากในการประมวลผล โดยจะเพิ่มตามขนาดของข้อมูลส่งผลให้เป็นปัจจัยหลักในการเข้ารหัสและถอดรหัส ความสำคัญของการรักษาความลับของข้อมูลของบุคคล หรือ องค์กร เป็นสิ่งที่ต้องพิจารณาในการแข่งขันทางความคิดและการระบุตัวตนบนในยุคเทคโนโลยี บริษัททางด้านเทคโนโลยีต่างๆ มีแนวโน้มในการใช้งานและลงทุนในเรื่องของการเข้ารหัสลับแสดงดังภาพที่ 1.1 ซึ่งแสดงสัดส่วนการใช้งานการเข้ารหัสลับบนหน่วยงานเทคโนโลยีในแต่ละปี ซึ่งหากเราสามารถเข้ารหัสข้อมูลต่างๆได้อย่างรวดเร็ว ย่อมเป็นปัจจัยที่ส่งผลต่อประสิทธิภาพและความเร็วในการประมวลผลข้อมูลในการแข่งขันของบริษัทเหล่านั้นที่จะนำผลิตภัณฑ์ใหม่ของตนเองไปแข่งขันกับบริษัทอื่น



ภาพที่ 1.1 แนวโน้มการใช้งานของเทคโนโลยีการเข้ารหัส [2]

ในปัจจุบันระบบประมวลผลการเข้ารหัสและถอดรหัสส่วนมากถูกออกแบบให้ทำงานบนหน่วยประมวลผลแบบแกนเดียว เนื่องจากง่ายต่อพัฒนาและบำรุงรักษา แต่จะเกิดความล่าช้าเมื่อจำเป็นต้องเข้ารหัสและถอดรหัสของมูลจำนวนมาก ซึ่งการใช้หน่วยประมวลผลที่สูงกว่าก็ย่อมจะส่งผลให้สามารถทำงานได้มีประสิทธิภาพสูงขึ้น โดยหน่วยประมวลผลที่ถูกพัฒนาในปัจจุบันนั้นจะทำการเพิ่มแกนหน่วยประมวลผลให้มีมากขึ้นจากแบบแกนเดียว (single core processor) เป็นหน่วยประมวลผลแบบหลายแกน (multi core processor) โดยสามารถเพิ่มประสิทธิภาพให้กับการประมวลผลการทำงานที่ดีขึ้นกว่าเดิม ดังภาพที่ 1.2 ประสิทธิภาพการทำงานของหน่วยประมวลผลแบบหลายแกนที่เพิ่มขึ้นเมื่อเทียบกับการทำงานบนหน่วยประมวลผลแกนเดียวในช่วงเวลาที่ผ่านมาและมีแนวโน้มที่เพิ่มขึ้น



ภาพที่ 1.2 ประสิทธิภาพการทำงานของหน่วยประมวลผลหลายแกนและแกนเดียว [2]

ในงานวิจัยนี้จะนำเสนอขั้นตอนวิธีทำการเข้ารหัสและถอดรหัสลับโดยใช้โครงสร้างหน่วยประมวลผลหลายแกน ซึ่งเป็นหน่วยประมวลผลกลางที่ออกแบบมาให้มีการทำงานหลายอย่างในเวลาเดียว ให้เราสามารถประสิทธิภาพในการเข้ารหัสลับบนหน่วยประมวลผล AES ซึ่งเป็นหน่วยประมวลผลจำลองที่ออกแบบมาให้รองรับการพัฒนาชุดคำสั่งและประมวลผลแบบหลายแกน

1.2 วัตถุประสงค์ของการวิจัย

งานวิจัยนี้นำเสนอวิธีการเพิ่มประสิทธิภาพในกระบวนการเข้ารหัสลับแบบ AES ด้วยกุญแจแบบ 128 บิต บนหน่วยประมวลผลหลายแกน

1.3 ขอบเขตของการวิจัย

1.3.1 พัฒนาการเข้ารหัสลับแบบ AES ที่มีกุญแจขนาด 128 บิต และพัฒนาบนหน่วยประมวลผล S2 เป็นหน่วยประมวลผลชนิดแบบหลายแกน

1.3.2 ทดสอบการทำงานของรหัสที่พัฒนาขึ้นด้วยชุดทดสอบบนหน่วยประมวลหลายแกนและแกนเดียว

1.3.3 วิเคราะห์ประสิทธิภาพการทำงานของการเข้ารหัสบนหน่วยประมวลผลแบบหลายแกน เทียบกับหน่วยประมวลผลแกนเดียว

1.4 ขั้นตอนการวิจัย

1.4.1 ศึกษาและทำความเข้าใจวิธีการเข้ารหัสแบบ AES บนหน่วยประมวลผลแบบแกนเดียวและหลายแกน

1.4.2 ศึกษาการทำงานหน่วยประมวลผลแบบหลายแกน S2 และการใช้งาน

1.4.3 ออกแบบการเข้ารหัส AES บนหน่วยประมวลผลแบบหลายแกนและแบบแกนเดียวเพื่อเพิ่มประสิทธิภาพการทำงาน

1.4.4 พัฒนาโปรแกรมการเข้ารหัสด้วยภาษา RZ บนหน่วยประมวลผลแกนเดียว และหลายแกน ด้วยวิธีการที่ออกแบบ

1.4.5 ทดสอบ เปรียบเทียบและประเมินผล โดยทำการนำเข้าข้อมูลชุดเดียวกันด้วย 3 วิธีการ จากนั้นวัดผลปริมาณงาน

1.4.6 ทำสรุปผลการวิจัยและข้อเสนอแนะ

1.5 ประโยชน์ที่คาดว่าจะได้รับ

1.5.1 ได้วิธีการเข้ารหัสลับ AES ที่สามารถเพิ่ม Throughput บนหน่วยประมวลผลหลายแกนและแกนเดียว

1.5.2 ได้ศึกษาวิธีการส่งผ่านข้อมูลของหน่วยประมวลที่มีการเข้ารหัสแบบ AES

1.6 ผลงานตีพิมพ์

ส่วนหนึ่งของวิทยานิพนธ์นี้ได้ตีพิมพ์และนำเสนอในการประชุมวิชาการต่อไปนี้

1.6.1 บทความเรื่อง An implementation of AES algorithm on multicore processor for high throughput โดยผู้แต่งคือ Supachai Thongsuk และ Prabhas Chongstitvatana ในการประชุมวิชาการ 6th ECTI-CARD 2014, Chiangmai, Thailand (May 21-23, 2014)

บทที่ 2
ทฤษฎีและงานวิจัยที่เกี่ยวข้อง

2.1 แนวคิดและทฤษฎี

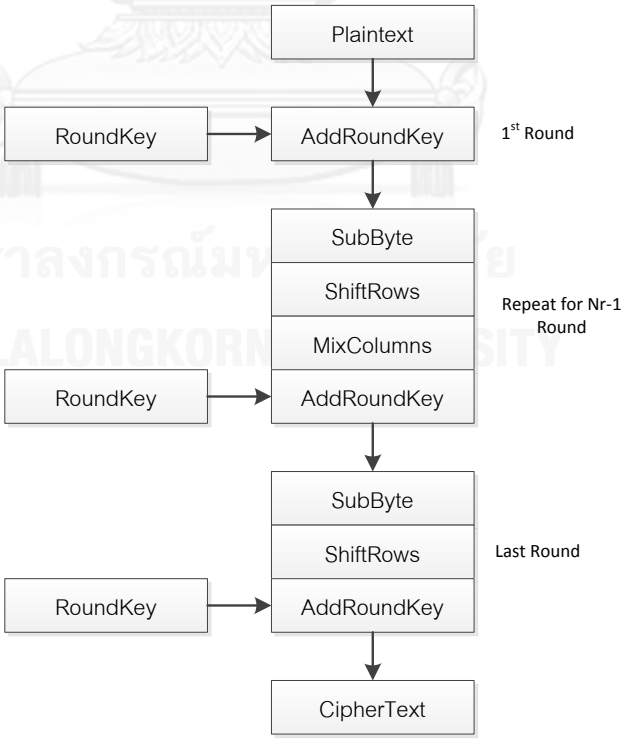
ทฤษฎีที่เกี่ยวข้องกับงานวิจัยมีดังนี้

2.1.1 Advance Encryption Standard (AES)

การเข้ารหัสลับแบบ AES ได้ถูกเผยแพร่โดย National Institute of Standards and Technology ในปี ค.ศ. 2001 เป็นการรหัสแบบสมมาตร ถูกคิดค้นโดยชาวเบลเยี่ยม ชื่อ Joan Daemen และ Vincent Rijmen มีการทำงานด้วยขนาดข้อมูล 128 บิต มีการใช้กุญแจ 128, 192 และ 256 บิต แบ่งการทำงานเป็นสองส่วน คือ การขยายกุญแจ (Key Expansion) และ การเข้ารหัส/ถอดรหัส

2.1.1.1 การทำงานของ AES

อัลกอริทึมของ AES จะเป็นการทำงานซ้ำแสดงดังภาพที่ 2.1 โดยเริ่มต้นด้วย AddRoundKey ตามด้วยการทำงาน SubByte, ShiftRows , MixColums และ AddRoundKey ตามลำดับโดยมีการทำซ้ำจำนวน Nr ครั้งขึ้นกับขนาดของกุญแจ สำหรับรอบสุดท้ายนั้นจะทำงานเพียง 3 ส่วนคือ SubByte,ShiftRows และ AddRoundKey ซึ่งการดำเนินงานเป็นรูปแบบไบต์ โดยกุญแจแต่ละรอบการทำงานได้มาจากการขยายกุญแจในขั้นตอนแรก



ภาพที่ 2.1 การขั้นตอนวิธีการเข้ารหัสแบบ AES

2.1.1.2 การขยายกุญแจ (Key Expansion)

วิธีการเข้ารหัสด้วย AES นั้นจะต้องใช้ cipher key ในแต่รอบการทำงาน โดยจะมีการเตรียมกุญแจย่อยของอัลกอริทึม AES-128 โดยกระบวนการขยายกุญแจ ซึ่งประกอบไปด้วย 2 กระบวนการคือ กระกวนการ Rotword จะเป็นการเลื่อนไบต์ในแต่ละ word เป็นวงกลมไปทางซ้าย 1 ไบต์ และการ Subword ที่จะทำการแทนที่ไบต์ของข้อมูลจากตาราง S-box ตัวเดียวกับที่ใช้ในกระบวนการ Subbyte จากนั้นจะทำการนำผลลัพธ์ที่ได้มา XOR กับ Array ค่าคงที่ เรียกว่าการทำ Rcon[i] เมื่อ i คือลำดับของ word

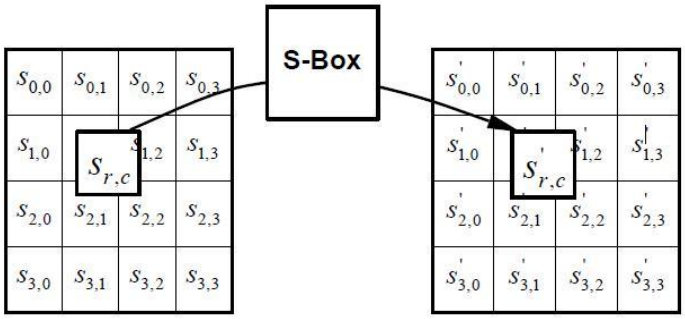
2.1.1.3 ขั้นตอนการเข้ารหัสและถอดรหัสลับ

Block สำหรับขั้นตอนการเข้ารหัสของข้อมูลรับเข้าและส่งออกจะมีค่าเท่ากับ 128 บิต ซึ่งแทนด้วย *Nb* = 4 ด้วยขนาดของกุญแจ (Cipher key) K คือ 128, 192 หรือ 256 บิต แทนด้วย Nk = 4, 6 หรือ 8 ตามลำดับ สำหรับรอบของการเข้ารหัสและถอดรหัสนั้นขึ้นกับขนาดของกุญแจ โดยจำนวนของรอบแทนด้วย Nr ซึ่ง Nr = 10 เมื่อ Nk = 4, Nr = 12 เมื่อ Nk = 6 และ Nr = 14 เมื่อ Nk = 8 ตามตารางที่ 2.1

ตารางที่ 2.1 จำนวนรอบการทำงานตามขนาดของกุญแจ

	ความยาวกุญแจ (Nk words)	ขนาด Block (Nb words)	จำนวนรอบ (Nr)
AES-128	4	4	10
AES-192	6	4	12
AES-256	8	4	14

1) Subbyte เป็นกระบวนการทำงานแบบ non-linear โดยมีการแทนข้อมูลแต่ละไบต์ ดังภาพที่ 2.2 จากตารางแทนค่า (S-box) ขนาด 256 ไบต์ ดังตารางที่ 2.2 มีการกำหนดพิกัดแกน XY แทนค่าด้วยเลขฐาน 16

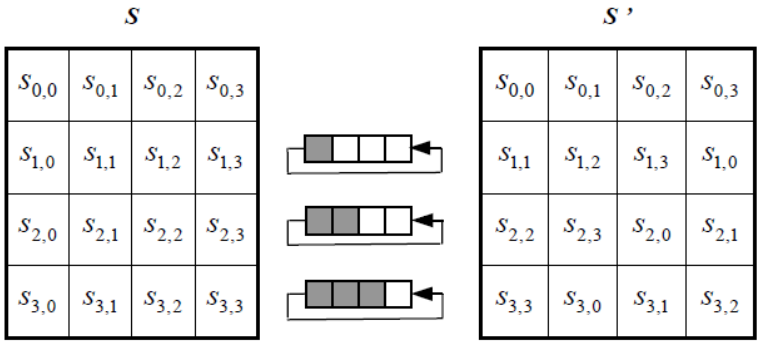


ภาพที่ 2.2 การแทนค่าแต่ละไบต์ข้อมูลด้วยตารางแทนค่า (S-box)

ตารางที่ 2.2 S-Box : ตารางแทนค่าสำหรับข้อมูลไบต์ XY รูปแบบเลขฐานสิบหก

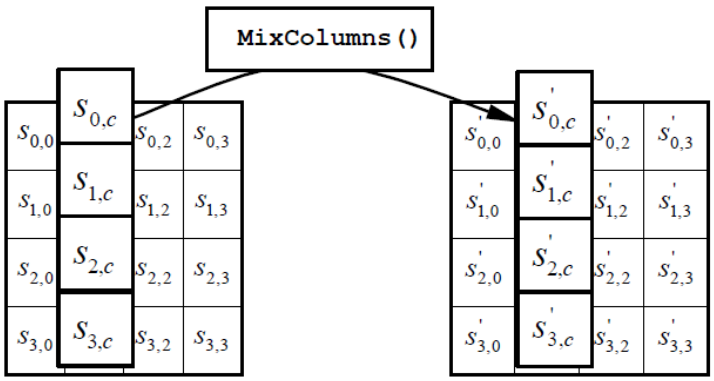
	1	2	3	4	5	6	7	8	9	a	b	c	e	e	f	g
1	52	9	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
2	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	d	e9	cb
3	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
4	8	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
5	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
6	6c	70	48	50	fd	e	b9	da	5e	15	46	57	a7	8d	9d	84
7	90	d8	ab	0	8c	bc	d3	0a	f7	e4	58	5	b8	b3	45	6
8	d0	2c	1e	8f	ca	3f	0f	2	c1	af	bd	3	1	13	8a	6b
9	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
a	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
b	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	b	1b
c	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
d	1f	dd	a8	33	88	7	c7	31	b1	12	10	59	27	80	ec	5f
e	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
f	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61
g	17	2b	4	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d

2) ShiftRows ข้อมูลไบต์ 3 แถวล่าง จะถูกเลื่อนไปทางซ้าย ในขณะที่แถวบนสุดจะไม่ถูกเลื่อน แถวที่ 2 ถูกเลื่อนไปทางซ้าย 1 ตำแหน่ง แถวที่ 3 ถูกเลื่อนไป 2 ตำแหน่ง และแถวที่ 4 จะถูกเลื่อนไป 3 ตำแหน่งดังภาพที่ 2.3



ภาพที่ 2.3 การเลื่อนของไบต์ข้อมูล แถวที่ 1 ถึงแถวที่ 4

3) MixColumns เป็นกระบวนการนำแนวหลักของตารางข้อมูล ไปผสมกับค่าคงที่ในตารางที่ 2.3 โดยภาพที่ 2.4 แสดงลักษณะการผสมกันของข้อมูล

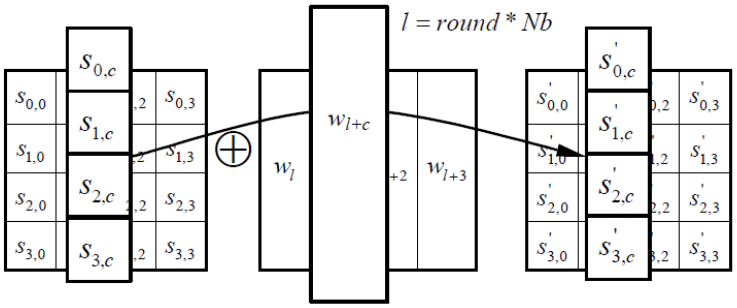


ภาพที่ 2.4 การคูณหลักข้อมูลกับตารางค่าคงที่

ตารางที่ 2.3 ตาราง Matrix ค่าคงที่สำหรับการ Mix Column

$$\begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix}$$

4) AddRoundKey เป็นกระบวนการนำชุดข้อมูล XOR กับ RoundKey ที่ได้สร้างมาก่อนหน้านี้ ดังภาพที่ 2.5



ภาพที่ 2.5 การ XOR แต่ละ Colum ของข้อมูลกับกุญแจ

2.1.2 Cipher block modes operation

วิธีการเข้ารหัสลับแบบกลุ่มสามารถแบ่งลักษณะการทำงานออกเป็น 5 แบบ ดังต่อไปนี้

2.1.2.1 Electronic codebook (ECB)

การทำงานในแบบ Electronic codebook (ECB) เป็นการแบ่งข้อมูลออกเป็นกลุ่ม ตามอัลกอริทึมที่เลือกใช้ เช่น 128 บิตสำหรับ AES หรือ 64 บิต

สำหรับ DES แล้วเข้ารหัสด้วยกุญแจที่กำหนดแสดงในภาพที่ 2.6 โดยวิธีการนี้ให้ความปลอดภัยในระดับที่ต่ำเนื่องจากใช้กุญแจเดิมในการเข้ารหัสแต่ละชุดข้อมูล

2.1.2.2 Cipher block chaining (CBC)

การทำงานในแบบ Cipher Block Chaining (CBC) แสดงในภาพที่ 2.7 โดยข้อมูลที่จะป้อนเข้าสู่อัลกอริทึมการเข้ารหัสนั้น จะเกิดจากผลลัพธ์ของการทำ XOR ระหว่าง Plaintext Block ปัจจุบันกับ Ciphertext Block จากการทำงานก่อนหน้า โดยใช้คีย์เดียวกัน ในทุกการเข้ารหัส ซึ่งจะเป็นผลให้ข้อมูลถูกเปลี่ยนก่อนที่จะเข้าสู่กระบวนการเข้ารหัส โดยการเปลี่ยนนี้จะเกิดขึ้นในลักษณะของลูกโซ่

2.1.2.3 Cipher feedback (CFB)

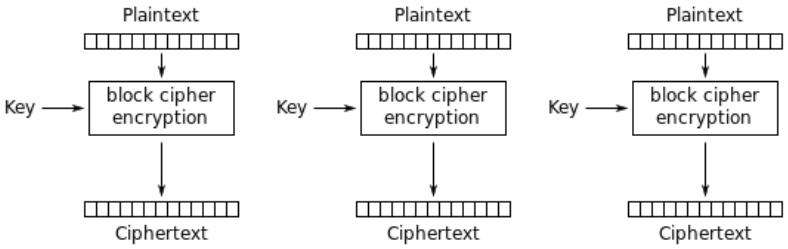
การทำงานในแบบ Cipher Feedback (CFB) แสดงในภาพที่ 2.8 มีการทำงานคล้ายกับ CBC โดยค่า Ciphertext Block ก่อนนำไปประมวลผลการเข้ารหัสแล้วนำมา XOR กับ Plaingtext Block ปัจจุบัน

2.1.2.4 Output feedback (OFB)

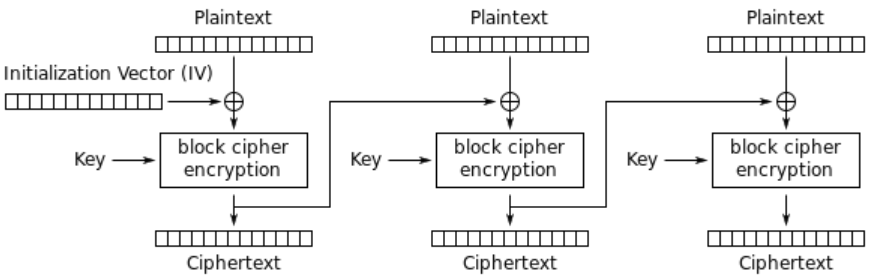
การทำงานในแบบ Output feedback (OFB) แสดงในภาพที่ 2.9 มีการคล้ายกับการทำงาน CFB แต่ข้อมูลที่ส่งต่อไปยังการเข้ารหัสชุดถัดไปคือรหัสที่วิธีการเข้ารหัสแล้วเพื่อให้มีการทำเข้ารหัสซ้ำ

2.1.2.5 Counter (CTR)

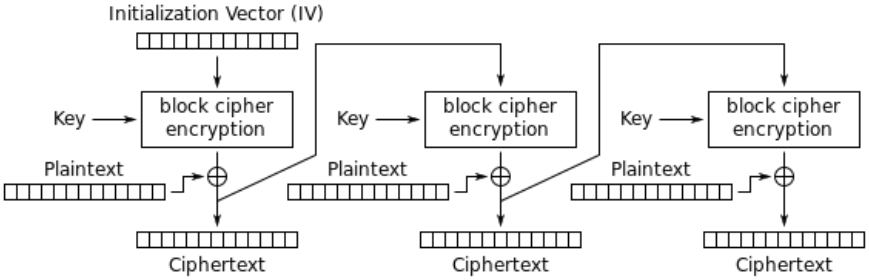
การทำงานในแบบ Counter (CTR) แสดงในภาพที่ 2.10 โดยอาศัยการติดตัวเลขที่เปิดเผย หรือเรียกว่า nonce ไปกับข้อมูล ตัวเลขนี้จะเปลี่ยนไปเรื่อยๆ ในแต่ละกลุ่มข้อมูล เมื่อนำค่า nonce มารวมกับกุญแจลับเพื่อเข้ารหัสแล้ว ข้อมูลจะเปลี่ยนไปเรื่อยๆ ตาม nonce



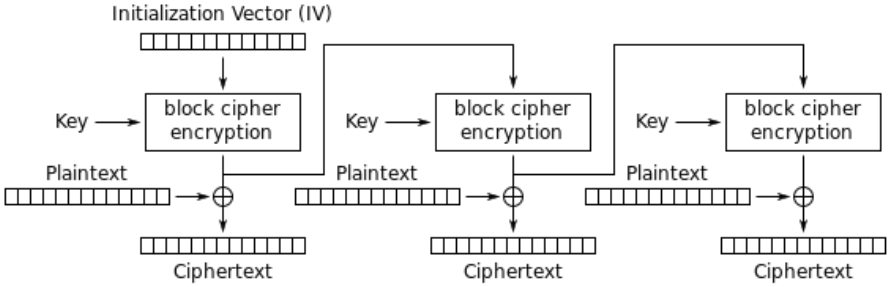
ภาพที่ 2.6 การทำงานของ Electronic codebook (ECB)



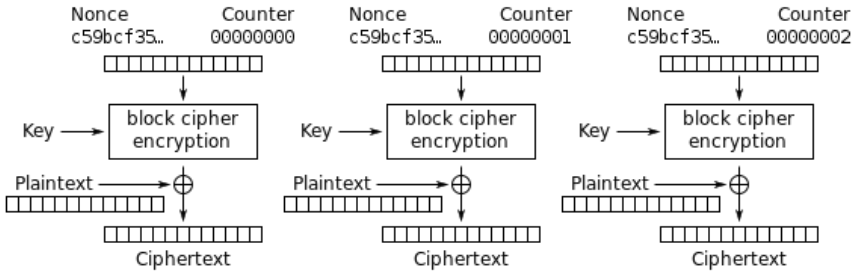
ภาพที่ 2.7 การทำงานของ Cipher Block Chaining (CBC)



ภาพที่ 2.8 การทำงานของ Cipher Feedback (CFB)



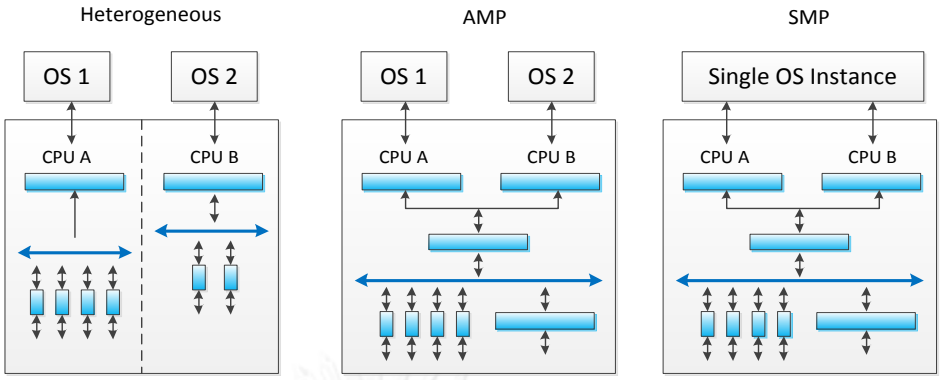
ภาพที่ 2.9 การทำงานของ Output feedback (OFB)



ภาพที่ 2.10 การทำงานของ Counter (CTR)

2.1.3 หน่วยประมวลผลแบบหลายแกน

หน่วยประมวลผลแบบหลายแกน [3] มีการทำงานโดยประมวลมากกว่า 1 หน่วยการทำงานซึ่งมีการบรรจุลงบนหน่วยประมวลผลกลาง โดยที่เป้าหมายของการออกแบบเพื่อให้ระบบสามารถทำงานได้มากขึ้นพร้อมกันและทำงานให้ได้ประสิทธิภาพที่ดีขึ้น โดยสามารถแบ่งโครงสร้างสถาปัตยกรรม ดังภาพที่ 2.11



ภาพที่ 2.11 สถาปัตยกรรมของหน่วยประมวลผลแบบหลายแกน

หน่วยประมวลหลายแกนแบบต่างกัน (Heterogeneous) เป็นหน่วยประมวลผลที่มีการใช้อย่างแพร่หลายในอุปกรณ์ฝังตัว (Embedded device) โดยเป็นหน่วยประมวลที่แต่ละแกนมีลักษณะไม่เหมือนกัน มีการแบ่งการทำงานของหน่วยประมวลผลออกเป็นตัวหลักและตัวรอง โดยที่หน่วยประมวลผลหลักจะทำงานส่วนใหญ่ ในขณะที่หน่วยประมวลผลตัวรองทำงานเฉพาะด้านตามลักษณะงาน ตัวอย่างเช่น หน่วยประมวลผลหลักทำการประมวลผลส่วนตัวรองทำการเข้ารหัสภาพหรือเสียงเพียงอย่างเดียว

หน่วยประมวลผลหลายแกนแบบไม่สมมาตร (Asymmetric Multiprocessing: AMP) เป็นหน่วยประมวลที่มีสองแกนหรือมากกว่าชนิดเดียวกันบนหน่วยประมวลผลกลาง ซึ่งแต่ละแกนสามารถทำงานบนระบบปฏิบัติการที่ต่างกันหรือหน่วยประมวลผลที่เหมือนกัน ตัวอย่างเช่น หน่วยประมวลผลแรกทำงานระบบปฏิบัติการสำหรับแสดงผล (GUI) ในขณะที่หน่วยประมวลผลที่ 2 ทำงานระบบปฏิบัติการสำหรับประมวลผลแบบเวลาจริง สำหรับการพัฒนาโปรแกรมบนหน่วยประมวลผลแบบนี้ง่ายกว่าเพราะโปรแกรมสามารถทำงานแต่ละแกนของหน่วยประมวลได้อย่างอิสระ แต่อย่างไรก็ตามถ้ามีการประมวลผลที่แตกต่างกันในแต่ละแกน จะส่งผลให้ยากต่อการควบคุมสมดุลของการทำงานแต่ละหน่วย

หน่วยประมวลผลหลายแกนแบบสมมาตร (Symmetric Multiprocessing: SMP) เป็นหน่วยประมวลผลที่ทำงานภายใต้ระบบปฏิบัติการเดียว โดยที่ระบบปฏิบัติการจะทำการจัดสรรส่วนการทำงานให้แต่ละหน่วยประมวลผล

2.1.4 หน่วยประมวลผล S2

หน่วยประมวลผลแบบ S2 [4] : 32 บิต Processor Version 3 เป็นโปรแกรมจำลองการทำงานของหน่วยประมวลผลแบบหลายแกน ซึ่งถูกพัฒนาขึ้นเพื่อใช้ในการศึกษาตั้งแต่ปี ค.ศ. 2001 โดยที่ใช้ภาษา RZ และภาษา Assembly ในการพัฒนา เป็นหน่วย

ประมวลผลขนาด 32 บิต และ 32 registers มีรูปแบบของคำสั่งแบบ Three-address instruction formats แสดงในภาพที่ 2.12

op r1 r2 r3 หมายถึง R[r1] = R[r2] op R[r3]

ภาพที่ 2.12 รูปแบบคำสั่ง Three-address instruction

ชุดคำสั่งถูกแบบออกตามการใช้งานเป็น 4 ประเภทคือ 1. กลุ่มของการคำนวณ 2. กลุ่มการเปรียบเทียบ 3.กลุ่มการควบคุมการทำงาน และ 4. กลุ่มจัดการข้อมูล ซึ่งแสดงตัวอย่างในภาพที่ 2.13

arithmetic: add sub mul div mod
logic: and or xor eq ne lt le gt ge shl shr
Control: jmp jt jf jal ret
data: ld st push pop

ภาพที่ 2.13 ประเภทของคำสั่งตามการใช้งาน

2.1.5 ภาษา RZ

RZ [4] เป็นภาษาขนาดเล็กที่มีความคล้ายกับภาษา C ซึ่งถูกใช้ในการเรียนการสอนเกี่ยวกับโครงการสร้างการเขียนโปรแกรมและโครงการของคอมพิวเตอร์ ภาษา RZ สามารถสรุปได้ดังนี้

1) รองรับการจำนวนเต็มที่เป็นชนิด integer เท่านั้น
2) ตัวแปรแบบ Global จะต้องถูกประกาศก่อนใช้งาน ในขณะที่ตัวแปรชนิด Local จะถูกประกาศอัตโนมัติ ซึ่งสามารถเรียกใช้งานได้ทันที
3) ตัวแปรแบบ Global สามารถประกาศเป็นชนิด Array ได้ ซึ่งรองรับเฉพาะแบบ 1 มิติ
4) มีการจองคำเฉพาะสำหรับภาษาดังนี้ if, else , while , return , print.
5) รองรับการทำงาน +, -, *, /, ==, !=, <, <=, >, >=, !, &&, ||, * (dereference), & (address).

ตัวอย่างภาษา RZ แสดงในภาพที่ 2.14 แสดงลักษณะการเขียนโปรแกรมบวกค่าในตัวแปรชนิด Array ซึ่งมีความคล้ายกับโครงสร้างการเขียนโปรแกรมในภาษา C โดยมีการประกาศตัวแปรชนิด global แบบ Array ในโปรแกรมหลักจะมีการกำหนดค่าให้กับตัวแปร ax[ ] จากนั้นทำการเรียกฟังก์ชันการบวกเลข sum() แล้วแสดงผลลัพธ์ด้วยคำสั่ง print()

```
// sum array
ax[10]
sum()
  i = 0
  s = 0
  while( ax[i] != 0 )
    s = s + ax[i]
    i = i + 1
  return s
main()
  ax[0] = 11
  ax[1] = 22
  ax[2] = 33
  ax[3] = 44
  ax[4] = 0
  print(sum())
```

ภาพที่ 2.14 ตัวอย่างการเขียนโปรแกรมบวกเลขในตัวแปร Array ด้วยภาษา RZ

2.2 เอกสารและงานวิจัยที่เกี่ยวข้อง

จากการศึกษาและค้นคว้างานวิจัยพบว่า งานวิจัยส่วนใหญ่ออกแบบการทำงานโดยเพิ่มประสิทธิภาพของการเข้ารหัสแบบ AES โดย Hardware ในรูปแบบ FPGAs และการใช้หน่วยประมวลผลแบบ Graphic (GPGPU) สำหรับการใช้พัฒนาที่เป็นหน่วยประมวลผลแบบหลายแกนที่เป็นหน่วยประมวลผลกลาง (CPU) นั้นมีน้อยกว่ามาก

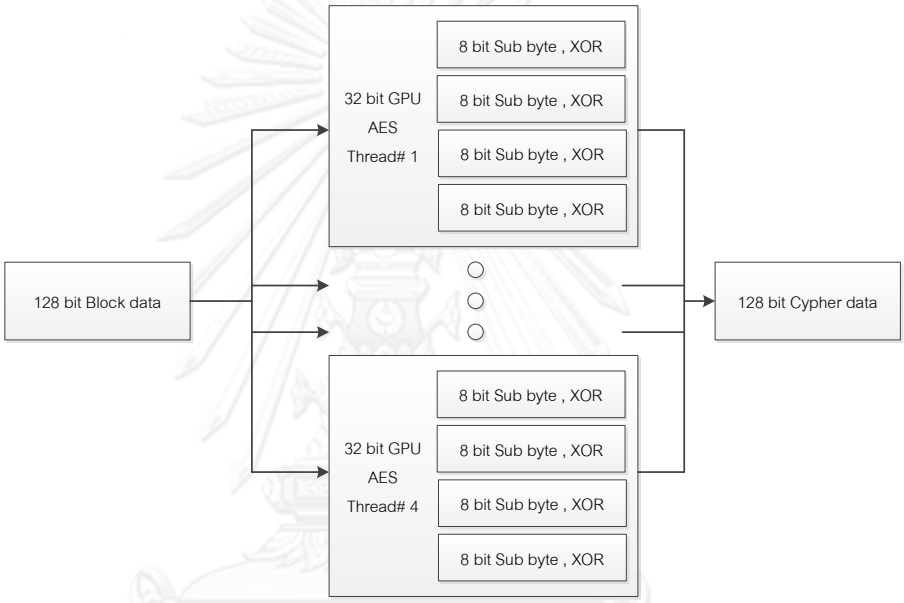
Angelo Barnes [5] นำเสนอการเพิ่ม Throughput ของการเข้ารหัส AES บนหน่วยประมวลผลแบบหลายแกน ด้วยวิธีการแบ่งข้อมูลออกเป็นก้อนตามจำนวนของหน่วยประมวลผล โดยพัฒนาด้วยภาษา C ในการโปรแกรมแบบขนานด้วยฟังก์ชัน fork และ pthread ซึ่งจากผลการวิจัยของ Angelo Barnes พบว่าการพัฒนาด้วยฟังก์ชัน fork และ pthread สามารถเพิ่ม throughput ตามจำนวนของหน่วยประมวลผล สำหรับ Throughput ที่ได้นั้นคือ 6637 Mb/s บนหน่วยประมวลผลแบบ 32 แกน 32 threads ด้วยการพัฒนาแบบ pthread

Svetlin A. Manavski [6] ได้ใช้หน่วยประมวลผลแบบ Graphic (GPGPU) ชนิด NVIDIA GeForce 8800 GTX ในการประมวลผล AES ด้วย CUDA platform ซึ่งเป็นหน่วยประมวลผลแบบ 32 บิต โดยแต่ละคอลัมน์แสดงการประมวลผลดังสมการ (1)

$$e_j = T_0[a_{0,j}] \oplus T_1[a_{1,j+1}] \oplus T_2[a_{2,j+2}] \oplus T_3[a_{3,j+3}] \oplus k_j \quad \text{..... (1)}$$

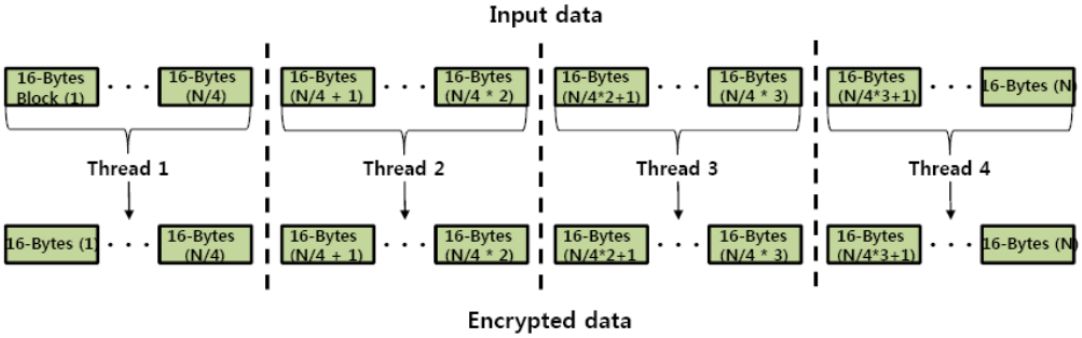
จากสมการ (1) แทนการ Subbyte ซึ่งจะทำการประมวล subbyte ได้ 4 ไบต์ และ AddRoundKey ได้ 4 ไบต์ พร้อมกันในแต่ละรอบจากขนาดหน่วยประมวลผล 32 บิต ดังนั้นจึงสามารถประมวลผลได้ 1 คอลัมน์จาก 4 คอลัมน์ของข้อมูลเข้า ปกติแล้วการประมวลผล AES ในแต่ละรอบจะใช้การ Subbyte ทั้งหมด 16 ครั้ง และ AddRoundKey จำนวน 16 ครั้ง เมื่ออ้างอิงจาก

สมการ (1) สามารถประมวลผลได้ทีละ 4 ครั้งต่อหนึ่งการทำงาน ดังนั้นจึงใช้ทั้งหมด 4 รอบการทำงาน เมื่อ GPU ทำงานจึงมีการคำนวณข้อมูลออกในแต่ละรอบด้วย 4 Threads แต่ละ Thread ประมวลผลคำสั่งดังสมการ (1) และแสดงการไหลของข้อมูลดังรูปภาพที่ 2.15 ซึ่งเป็นข้อดีของการประมวลผลแบบ GPU ในขณะที่ CPU จะประมวลผลได้ทีละ 1 ชุดคำสั่ง ดังนั้นในแต่ละรอบจึงต้องมีคำสั่ง Subbyte และ AddRoundKey จำนวน 16 คำสั่ง ในขณะที่ GPU มีเพียงแค่ 4 คำสั่ง เท่านั้น จากวิธีการดังกล่าวสามารถทำให้ Throughput ของการเข้ารหัสและถอดรหัส 8.28 กิกะบิตต่อวินาที ที่ข้อมูล 8MB ด้วย AES-128

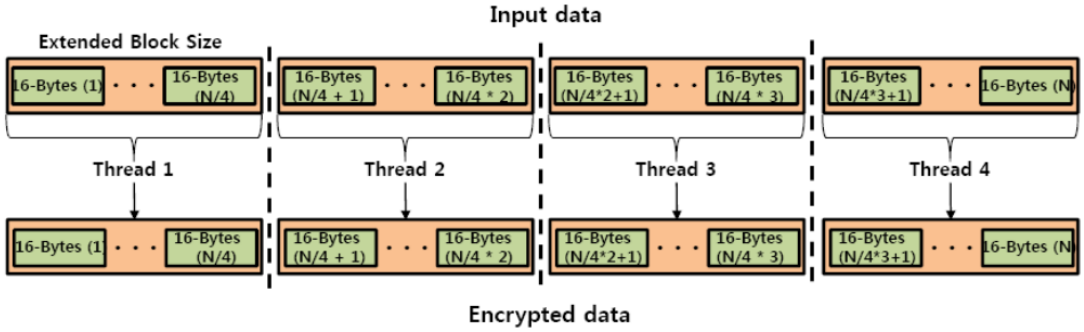


ภาพที่ 2.15 ลักษณะการประมวลผล AES ด้วย CUDA platform ที่ 128 บิต input

Nhat-Phuong Tran [7] เสนอการประมวลวิธีการเข้ารหัสแบบ AES โดยการเพิ่มขนาดของ Block ซึ่งปกตินั้นจะทำการประมวลผลด้วย Block ขนาด 16 ไบต์ เขาได้ทำการเพิ่มขนาด Block ให้มากขึ้นเพื่อที่จะลด Overhead แสดงในภาพที่ 2.16 และภาพที่ 2.17 ในการส่งผ่านข้อมูลรวมถึงการเข้าถึงข้อมูลใน memory ตัวอย่างเช่น ข้อมูลขนาด 1024 ไบต์ ด้วย Block ขนาด 1024 ไบต์ ประมวลผลจำนวนทั้งหมด 64 ครั้งโดยการเพิ่มขนาด Array เป็น 1024 ซึ่งสามารถเพิ่มความเร็วของการเข้ารหัสด้วย CPU 1.25 ~ 1.28 เท่า และด้วย GPU 6.03 ~ 8.53 เท่า โดยมีแนวโน้มที่เพิ่มมากขึ้นเมื่อมีการเพิ่มขนาดของ Block ของข้อมูล



ภาพที่ 2.16 การเข้ารหัสด้วยขนาด Block 16 ไบต์ 4 Thread



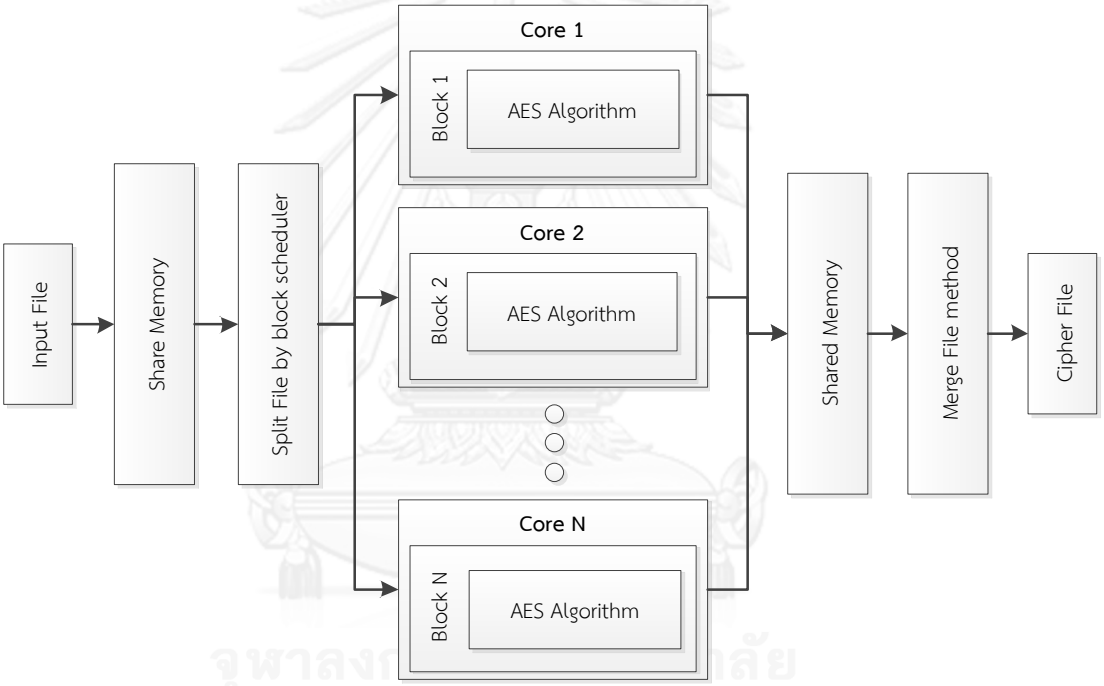
ภาพที่ 2.17 การเข้ารหัสด้วยการเพิ่มขนาดของ Block 4 Thread

Huang, Chang, Lin and Tai [8] นำเสนอการเข้ารหัสแบบ 32 บิต AES บน Xilinx FPGA Chip (Spartan-3 XC3S200) ซึ่งสามารถให้ Throughput 647 เมกะบิตต่อวินาที และ ยังพัฒนา 128 บิต AES โดย 32 บิต AES จำนวน 4 ชุดทำการประมวลผลแบบขนาน ในปี 2008 Gielata, Russek และ Wiatr ได้ทำ AES-128 บิต บน FPGA [8] ทำการเข้ารหัสลับ Pentium II 450MHz ได้ Throughput 77 เมกะบิตต่อวินาที and 74 เมกะบิตต่อวินาที

บทที่ 3
แนวคิดและวิธีดำเนินการวิจัย

3.1 แนวความคิด

หน่วยประมวลผลมีการทำงานในการเข้ารหัสและถอดรหัสจากข้อมูลนำเข้า โดยข้อมูลจะถูกเก็บไว้ในส่วนของหน่วยความจำร่วม จากนั้นจะถูกแบ่งส่วนของข้อมูลออกเป็นกลุ่มด้วยชุดคำสั่งจัดการข้อมูลขนาดที่เท่ากัน ซึ่งหน่วยประมวลผลแต่ละแกนจะทำการนำข้อมูลที่ถูกแบ่งเข้าประมวลผลในแต่ละแกน จากนั้นข้อมูลที่ถูกเข้ารหัสจะเก็บไว้ในส่วนหน่วยความจำร่วมแล้วทำการรวมข้อมูลเข้าด้วยกัน ตามภาพที่ 3.1 แสดงแนวคิดของงานวิจัยนี้ โดยมีสมมติฐานเบื้องต้นคือหน่วยประมวลผลแต่ละแกนจะทำการประมวลผลเป็นอิสระต่อกัน ไม่ถูกขัดจังหวะด้วยงานอื่น

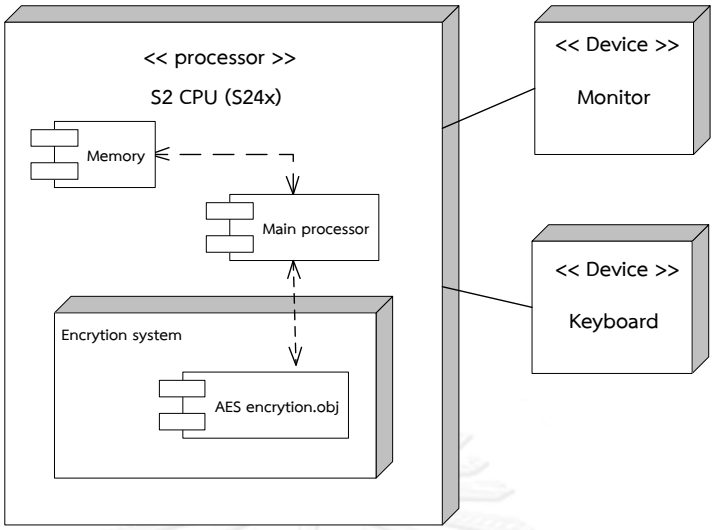


ภาพที่ 3.1 แนวคิดของงานวิจัย

3.2 การออกแบบ

3.1.1 การออกแบบสถาปัตยกรรมระบบ

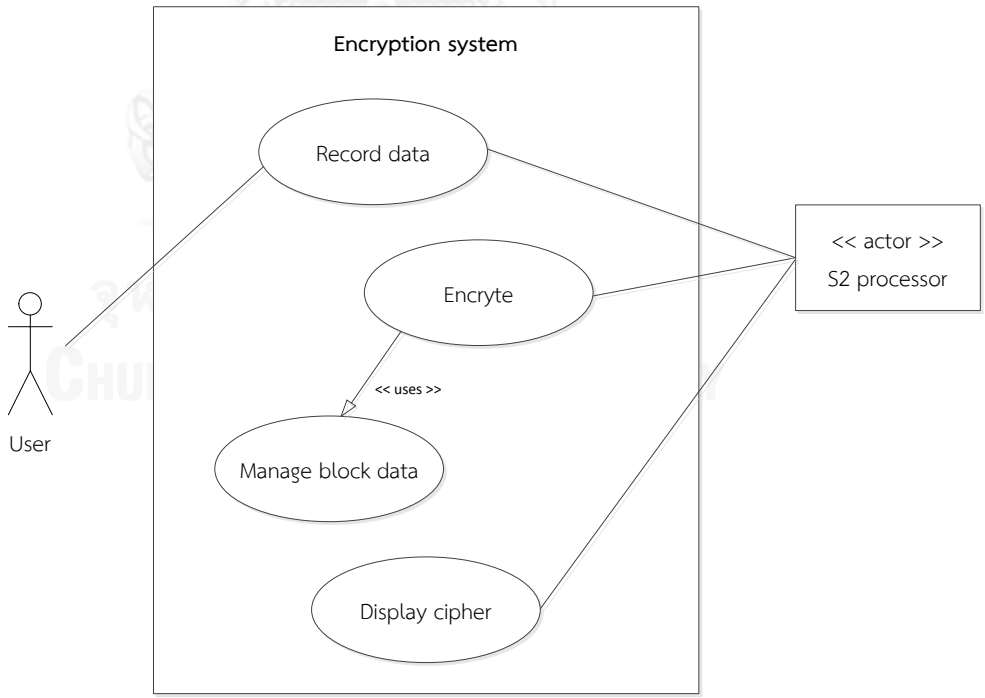
ในงานวิจัยนี้ การเข้ารหัสถูกประมวลผลด้วยหน่วยประมวลผลหลายแกนจำลอง S2 ซึ่งทำหน้าที่รับข้อมูลจากผู้ใช้งานเพื่อประมวลผลแล้วทำการแสดงผลเมื่อทำการประมวลผลการเข้ารหัสเสร็จ จากภาพที่ 3.2 แสดงสถาปัตยกรรมของระบบ โดยส่วนหน่วยประมวลผล S2 มีส่วนแกนการทำงานหลักซึ่งเชื่อมต่อกับหน่วยความจำและชุดรหัสคำสั่งสำหรับเข้ารหัสแบบ AES ที่พัฒนาขึ้น ผู้ใช้งานทำการป้อนข้อมูลสำหรับการเข้ารหัสด้วยอุปกรณ์ในรูปแบบตัวอักษร แล้วแสดงผลออกทางหน้าจอเมื่อหน่วยประมวลผลทำงานเสร็จ



ภาพที่ 3.2 โครงสร้างสถาปัตยกรรมของระบบ

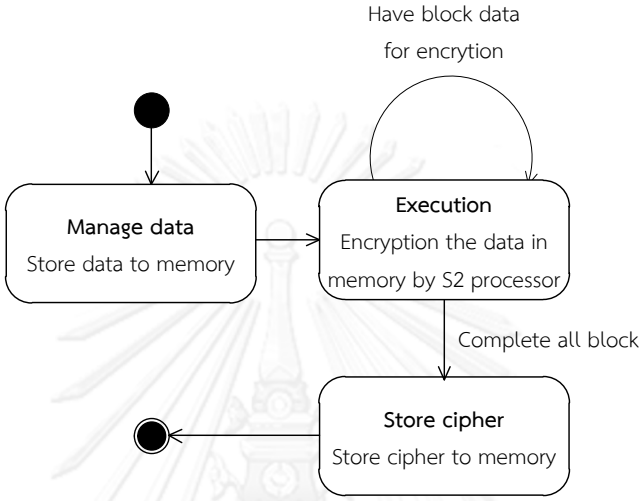
3.1.2 การออกแบบหน้าที่การทำงานของระบบ

ระบบการประมวลผลการเข้ารหัสประกอบด้วยส่วนการบันทึกข้อมูลที่กระทำโดยผู้ใช้งาน มีหน่วยประมวลผล S2 เป็นส่วนในการเข้าถึงข้อมูลจากผู้ใช้งาน และทำการประมวลผลการเข้ารหัส และแสดงผลข้อมูลที่ถูกเข้ารหัสแล้วตามภาพที่ 3.3



ภาพที่ 3.3 แผนภาพยูสเคสของระบบการเข้ารหัส

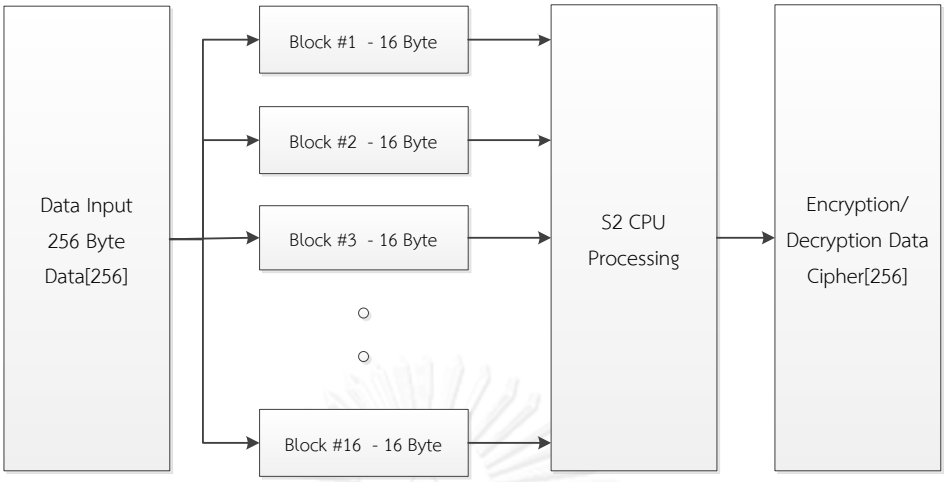
การทำงานตามภาพที่ 3.4 แสดง State diagram ของข้อมูลโดยที่เริ่มจากการเข้าสู่ส่วนการจัดการข้อมูลคือจัดเก็บลงหน่วยความจำและแบ่งออกเป็นกลุ่ม จากนั้นเข้าสู่สถานะการเข้ารหัสที่ประมวลผลโดยหน่วยประมวลผล S2 ด้วยวิธีการที่ออกแบบ ซึ่งหากประมวลผลข้อมูลครบทุกกลุ่มข้อมูล จึงจะไปยังส่วนการบันทึกข้อมูลเข้ารหัสที่หน่วยความจำและแสดงผลไปยังผู้ใช้งาน



ภาพที่ 3.4 แผนภาพ state diagram ของข้อมูลในการเข้ารหัส

3.1.3 การจัดการข้อมูล

เนื่องจากข้อมูลนำเข้าที่ใช้ในการเข้ารหัสมีขนาดเกินกว่าข้อกำหนดของขั้นตอนวิธีการเข้ารหัสแบบ AES ซึ่งมีการทำงานเพียง 16 ไบต์ ซึ่งข้อมูลนำเข้าที่มีมากกว่าจำเป็นต้องมีการจัดการโดยการแบ่งข้อมูลขนาดใหญ่ออกเป็นบล็อกย่อยๆ ข้อมูลที่จะถูกเข้ารหัสหรือถอดรหัสจะถูกแบ่งออกเป็นบล็อกด้วยขนาด 16 ไบต์ต่อบล็อกที่เท่ากัน แล้วถูกกระจายเก็บในหน่วยความจำของแต่หน่วยประมวลผล ดังภาพที่ 3.5 แสดงการแบ่งข้อมูลขนาด 256 ไบต์ เป็น block ละ 16 ไบต์ ซึ่งจะได้ทั้งหมด 16 บล็อกข้อมูล แต่ละแกนประมวลผลจะมีหน้าที่รับผิดชอบในการประมวลผลการเข้ารหัสข้อมูลหมายเลขบล็อกที่กำหนดดังตารางที่ 3.1 ในกรณีที่ข้อมูลมีขนาดน้อยกว่า 16 ไบต์ ซึ่งไม่เต็มบล็อก โปรแกรมจะแทนที่ช่องข้อมูลที่ว่างด้วยข้อมูล 0 ให้เต็มบล็อกข้อมูลนั้น



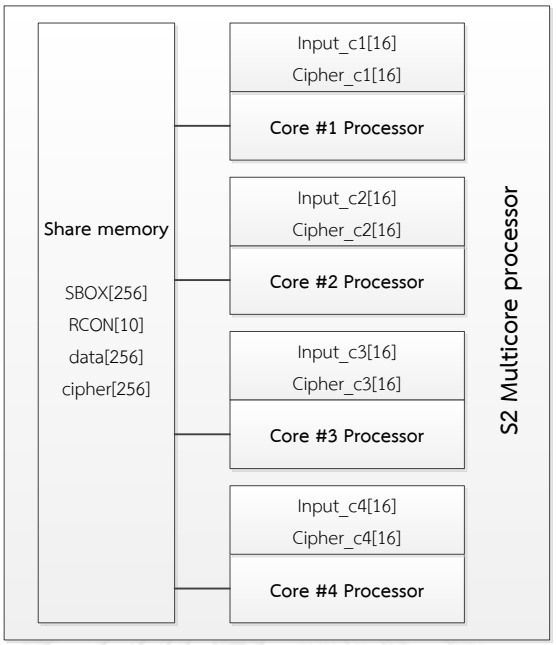
ภาพที่ 3.5 การแบ่งข้อมูลก่อนการประมวลผล

ตารางที่ 3.1 บล็อกข้อมูลสำหรับหน่วยประมวลผลแต่ละแกน

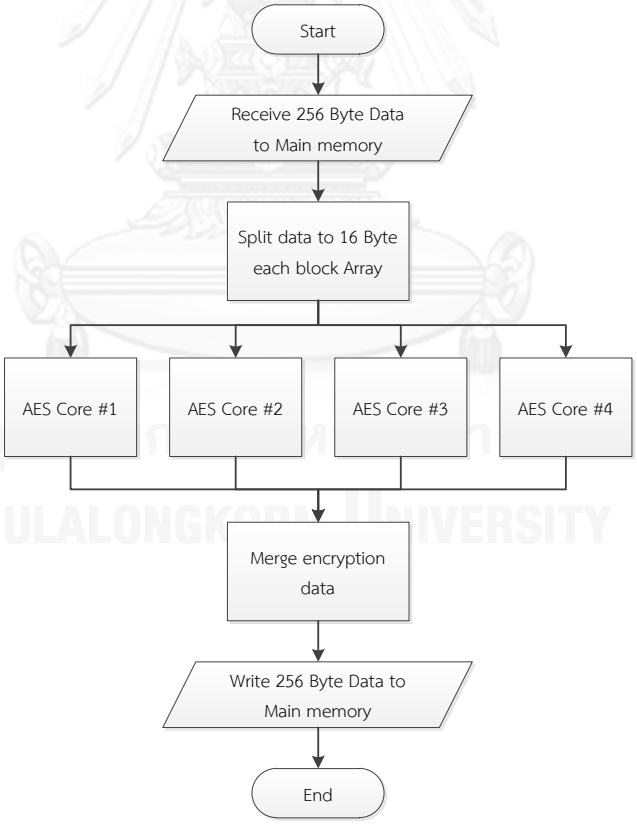
Core 1	Core 2	Core 3	Core 4
Block #1	Block #2	Block #3	Block #4
Block #5	Block #6	Block #7	Block #8
Block #9	Block #10	Block #11	Block #12
Block #13	Block #14	Block #15	Block #16

3.1.4 ออกแบบการประมวลผลแบบแบ่งส่วนเท่ากัน

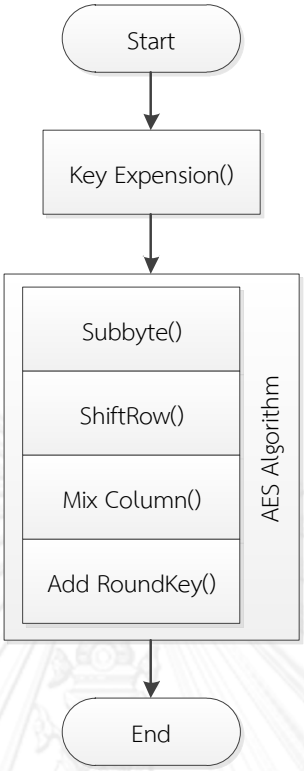
หน่วยประมวลผลแต่ละแกนจะมีการทำงานการเข้ารหัสที่เป็นอิสระต่อกันและไม่ถูกควบคุมโดยหน่วยประมวลใด โดยแต่ละแกนจะมีการประมวลผลข้อมูลที่ได้จากแบ่งชุดข้อมูลโดยมีส่วนการออกแบบของพื้นที่ในการจัดเก็บข้อมูลก่อนและหลังการเข้ารหัสดังภาพที่ 3.6 กำหนดให้ data[256] คือข้อมูลสำหรับเข้ารหัส และ cipher[256] คือพื้นที่เก็บข้อมูลที่ถูกเข้ารหัสแล้ว โดยหน่วยประมวลผลแต่ละแกนต้องเข้าถึงข้อมูลในพื้นที่เหล่านี้เพื่ออ่านและเขียนบนหน่วยความจำหลัก ซึ่งพื้นที่ของหน่วยความจำหลักจะไม่สามารถเข้าใช้งานได้พร้อมกันจากหน่วยประมวลผลแต่ละแกน ซึ่งหากมีการเข้าใช้งานพร้อมกันนั้นได้จำลองให้เกิดการหน่วงเวลาหรือการขัดแย้งกันของแกนประมวลผลด้วยเวลา 2 สัญญาณนาฬิกา โดยมีผังงานการทำงานเข้ารหัสลับของหน่วยประมวลผลแสดงดังภาพที่ 3.7 และแต่ละแกนมีลำดับฟังก์ชันการทำงานดังภาพที่ 3.8



ภาพที่ 3.6 การจองหน่วยความจำในการเข้ารหัสบนหน่วยประมวลผล S2 ขนาด 4 แกน



ภาพที่ 3.7 การไหลของข้อมูลบนหน่วยประมวลในการเข้ารหัส



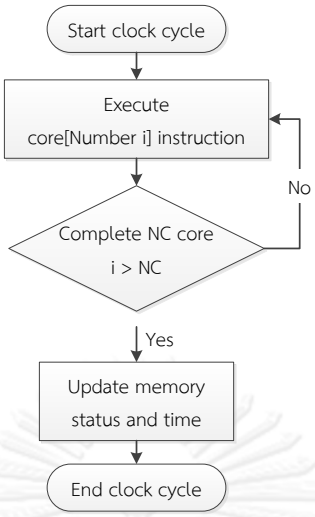
ภาพที่ 3.8 ลำดับการทำงานการเข้ารหัส AES ภายในบนหน่วยประมวลผลแต่ละแกน

3.1.5 ลักษณะประมวลผลภายในของหน่วยประมวลผลหลายแกน S2

ภาพที่ 3.9 แสดงลักษณะชุดคำสั่งการทำงานหน่วยประมวลผลจำลอง S2 ซึ่งมีหลักการประมวลต่อ 1 คาบเวลา T ที่ 1 คำสั่ง โดยอาศัยการวนซ้ำเพื่อประมวลผล ซึ่ง NC คือจำนวนแกนการทำงานที่ต้องประมวลผลในแต่ละคาบเวลาแสดง flow การทำงานตาม ภาพที่ 3.10

```
while( flag ){
      flag = 0;
      for(i = 0; i < NC; i++){
            if(runflag[i]){
               run(i);
            }
      }
      updatememstate();
      T++;
}
```

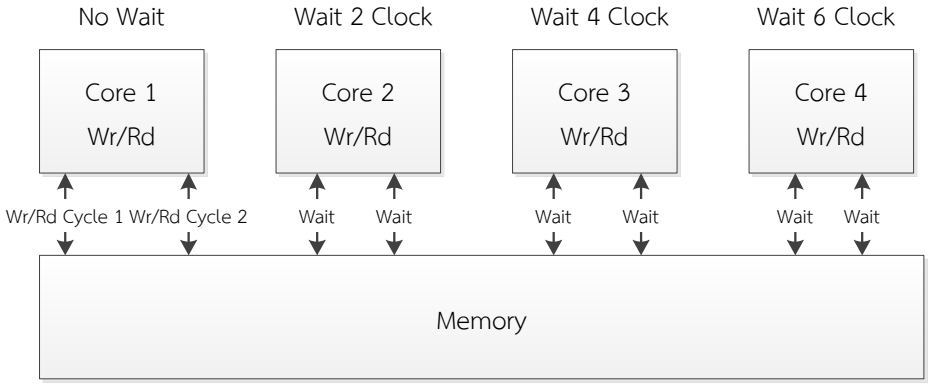
ภาพที่ 3.9 ชุดรหัสคำสั่งเพื่อดึงข้อมูลจากหน่วยความจำหลัก



ภาพที่ 3.10 แสดง Flow chart การทำงานภายในของหน่วยประมวลผลหลายแกน S2

3.1.6 การจำลองความขัดแย้งของหน่วยความจำบนหน่วยประมวลผล S2 (Stall memory)

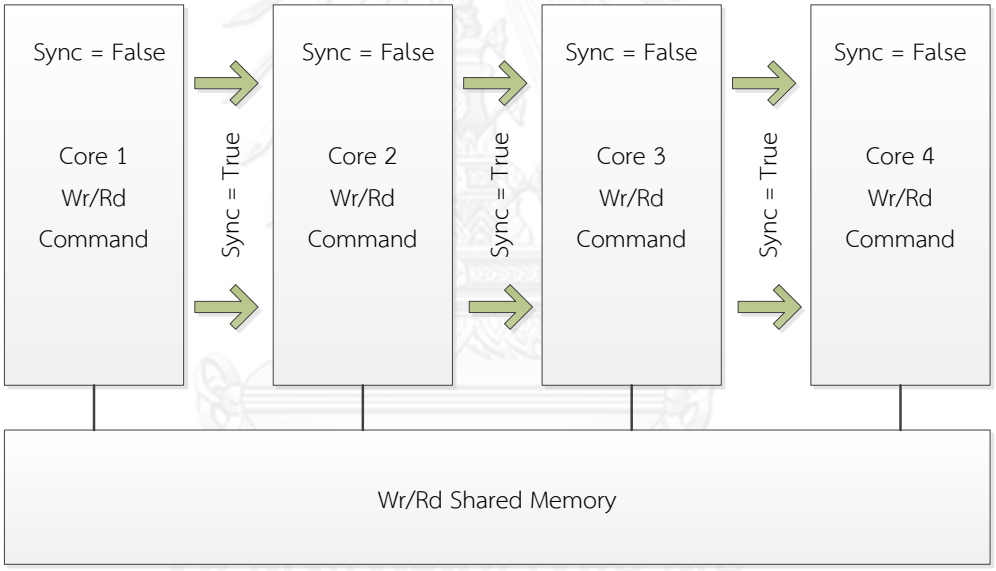
หน่วยความจำบนหน่วยประมวลผล S2 ได้ถูกออกแบบให้มีการพิจารณาในเรื่องการเข้าถึงหน่วยความจำหลัก ซึ่งในทางปฏิบัติแล้วหน่วยความจำจะสามารถเข้าถึงได้เพียงแค่แกนเดียวเท่านั้น ในการอ่านและเขียนข้อมูล โดยหน่วยประมวลผล S2 กำหนดให้ใช้เวลาในการเข้าใช้งานหน่วยความจำ 2 สัญญาณนาฬิกา ซึ่งในระหว่างที่มีการใช้งานหน่วยความจำของแกนใดๆ หน่วยประมวลผลแกนอื่นจะไม่สามารถใช้งานได้จนกว่าหน่วยประมวลผลที่มีเข้าถึงหน่วยความจำประมวลผลคำสั่งนั้นเสร็จแล้วเท่านั้น จากภาพที่ 3.11 แสดงถึงการใช้งานหน่วยความจำของหน่วยประมวลผลขนาด 4 แกน ในกรณีที่มีการเรียกใช้งานหน่วยความจำพร้อมกันทั้ง 4 แกน ซึ่งหน่วยประมวลผลที่ 1 จะสามารถใช้งานได้ทันทีในขณะที่หน่วยประมวลผลที่เหลือต้องรอจนเสร็จสิ้นการทำงานซึ่งแสดงให้เห็นถึงกรณีที่แย่ที่สุดในการใช้งานหน่วยความจำที่ต้องรอถึง 6 สัญญาณนาฬิการในการประมวลผลของแกนที่ 4 ในการเข้าถึงหน่วยความจำ



ภาพที่ 3.11 โครงการใช้งานหน่วยความจำภายในโดยหน่วยประมวลผล

3.1.7 การประมวลผลการเข้ารหัสด้วยการประสานเวลา (Synchronization)

หน่วยประมวลผลแต่และแกนมีการกำหนด synchronization flag เพื่อจัดการการใช้งานของการเข้าถึง share memory โดยที่หน่วยประมวลผลแรกจะทำการเขียนข้อมูลลงในหน่วยความจำรวม ซึ่งทำการตั้ง flag ให้เป็น false ในขณะที่แกนที่สองจะยังไม่สามารถทำการเขียนข้อมูลได้จนกว่าและรอจนกว่า flag ของแกนทีสองจะถูกเปลี่ยนเป็น true ทำโดยแกนแรก ซึ่งจะมีลักษณะเดียวกันในแกนที่สามและแกนที่สี่ ที่ต้องรอ flag ถูกเปลี่ยนแปลงจากแกนก่อนหน้าเช่นเดียวกัน การออกแบบลักษณะการทำงานเช่นนี้จะช่วยให้หน่วยประมวลผลแต่ละแกนลดการขัดแย้งกันในการเข้าถึงข้อมูลในหน่วยความจำ ตามภาพที่ 3.12 โดยการทำงานแต่ละส่วนจะเรียกใช้งานคำสั่ง syscall(17,0) เพื่อระบุว่าแต่ส่วนการทำงานของแกนนั้นเข้าสู่สถานะประสานเวลาแล้วแต่ยังไม่สามารถทำงานในส่วนคำสั่งถัดไปได้จนกว่าทุกแกนทำงานเรียกใช้งาน syscall(17,0) เพื่อระบุว่าแกนการทำงานนั้นเสร็จสิ้นการทำงานในส่วนของตนเองแล้ว



ภาพที่ 3.12 ลักษณะการประสานเวลาเพื่อใช้งานหน่วยความจำ

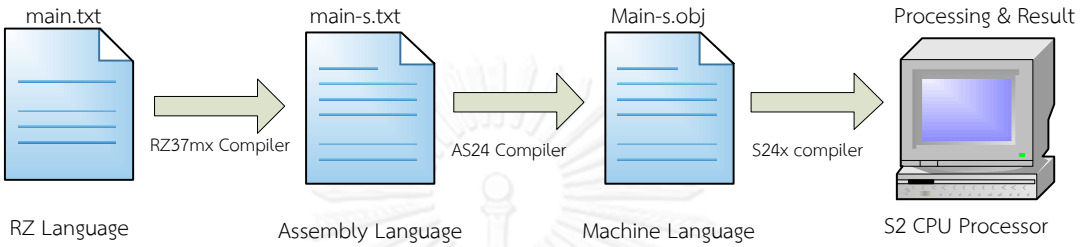
3.3 การพัฒนา

การพัฒนาการเข้ารหัสบนหน่วยประมวลผลหลายแกน ในงานวิจัยนี้ทำการทดลองบนหน่วยประมวล S2 โดยมีการทำงานแต่ละ method ดังนี้

3.3.1 การพัฒนาชุดคำสั่งด้วยภาษา RZ และหน่วยประมวลผล S2

ภาพที่ 3.13 แสดงลักษณะกระบวนการพัฒนาชุดคำสั่งเพื่อใช้กับหน่วยประมวลผล S2 โดยที่เริ่มจากการสร้างชุดคำสั่งด้วยภาษา RZ ซึ่งเป็นภาษาที่ลักษณะคล้ายกับภาษา C ดังภาพที่ 3.14 ตัวอย่างชุดคำสั่งที่เขียนด้วยภาษา RZ ที่มีฟังก์ชันการทำงาน Mod ตัวแปร a ด้วย b แล้วคืนค่ากลับด้วยตัวแปร d เมื่อใช้งาน RZ37mx compiler ให้ผลลัพธ์ออกมาเป็นชุดคำสั่งภาษา Assembly ดังภาพที่ 3.15 จากนั้นทำการ compile อีกครั้งด้วย S24x เพื่อ

แปลงชุดคำสั่งจากภาษา Assembly ให้เป็นภาษาเครื่องสำหรับหน่วยประมวลผล S2 ซึ่งแสดงให้เห็นตามภาพที่ 3.16 โดยภาพที่ 3.17 แสดงการผลลัพธ์ที่ได้จากการประมวลผลการ mod ตัวเลข 24 ด้วย 10 ซึ่งมีชุดคำสั่งในการเรียกฟังก์ชันย่อยการทำงานดังภาพที่ 3.18 แล้วแสดงผลลัพธ์ออกมาคือ 4 บนหน่วยประมวลผลแกนเดียว พร้อมทั้งแสดงจำนวนคำสั่งที่ถูกประมวลผล เวลาในการประมวลผล



ภาพที่ 3.13 การพัฒนาชุดคำสั่งเพื่อใช้กับหน่วยประมวลผล S2

```
modNum(a,b)
   c = a / b
   d = a - ( c * b )
   return d
```

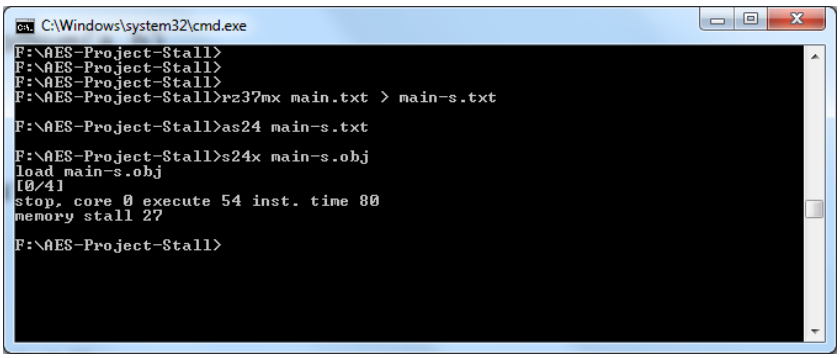
ภาพที่ 3.14 ชุดคำสั่งฟังก์ชันการ mod ตัวเลขด้วยภาษา RZ

```
pop sp r2
pop sp r1
; (= #3 (/ #1 #2 ))
div r5 r1 r2
mov r3 r5
; (= #4 (- #1 (* #3 #2 )))
mul r6 r3 r2
sub r5 r1 r6
mov r4 r5
; (return #4 )
mov retval r4
```

ภาพที่ 3.15 คำสั่งฟังก์ชันการ mod ตัวเลขภาษา Assembly

```
48, 29, 2, 0,
48, 29, 1, 0,
13, 5, 1, 2,
31, 3, 5, 0,
12, 6, 3, 2,
11, 5, 1, 6,
31, 4, 5, 0,
31, 28, 4, 0,
```

ภาพที่ 3.16 คำสั่งฟังก์ชันการ mod ตัวเลขภาษาเครื่องสำหรับหน่วยประมวลผล S2



ภาพที่ 3.17 การทำงานของหน่วยประมวลผล S2 ในการทำงาน mod

```
main()
    result = modNum(24,10)
    print(result)
```

ภาพที่ 3.18 ตัวอย่างการเรียกใช้งาน mod ค่า 24 ด้วย 10

3.3.2 การอ่านข้อมูลจากหน่วยความจำหลัก

ก่อนการประมวลผลการเข้ารหัส AES หน่วยประมวลผลการทำงานแต่ละแกนจะทำการอ่านข้อมูลจากหน่วยความจำหลักแล้วเขียนไปยังหน่วยความจำสำหรับหน่วยประมวลผลแต่ละแกนนั้น โดยกำหนดให้ data[256] เป็นข้อมูลที่จะเข้ารหัสถูกเก็บไว้ในที่อยู่หน่วยความจำหลัก ในขณะที่ input_c1[16] กำหนดให้เป็นตัวแปรที่จัดเก็บข้อมูลที่ถูกอ่านจากหน่วยความจำหลักเข้ามาเขียนไว้ที่หน่วยความจำเฉพาะสำหรับหน่วยประมวลผลของแกนการทำงาน ส่วน a คือตัวแปรเพื่อระบุหมายเลข Block ที่ต้องการเข้าถึง ข้อมูลที่ถูกอ่านมานั้นจะถูกจัดเก็บเพื่อรอสำหรับประมวลผลการเข้ารหัส ดังภาพที่ 3.19 แสดงชุดคำสั่งในหน่วยประมวลผลในแกนที่ 1

```
rdata_c1(a)
    i = 0
    while(i < 16)
        input_c1[i]=data[i+16*a]
        i = i + 1
```

ภาพที่ 3.19 ชุดคำสั่งสำหรับดึงข้อมูลจากหน่วยความจำหลัก

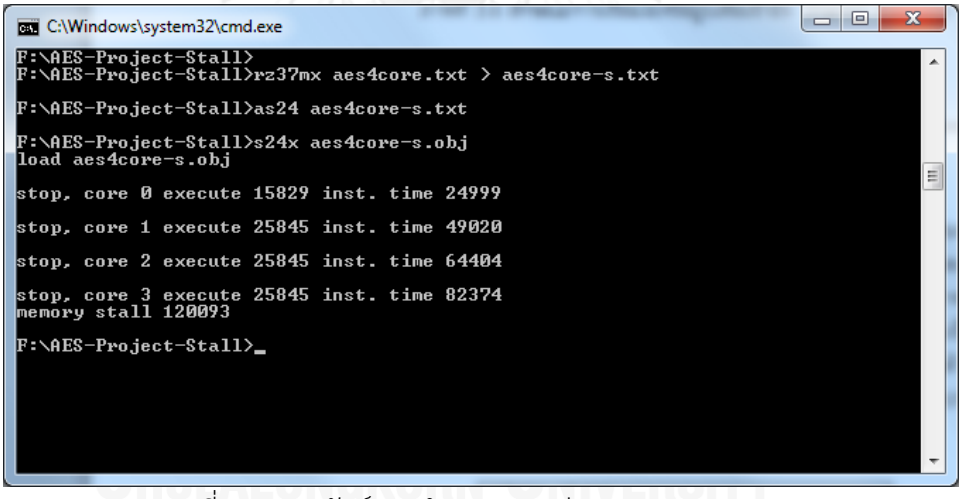
3.3.3 การพัฒนาชุดคำสั่งสำหรับหน่วยประมวล S2

ในการทำงานของชุดคำสั่งของหน่วยประมวลผลหลายแกน S2 นั้น จะมีคำสั่งพิเศษ syscall(16,0) สั่งงานระบบให้มีการเริ่มต้นการทำงานของหน่วยประมวลผลอีก 3 แกนการทำงาน ตามภาพที่ 3.20 ซึ่งฟังชัน Main แต่ละตัวจะถูกเรียกทำงานโดยหน่วยประมวลผลแต่ละแกนตามลำดับเลข เช่น หน่วยประมวลผลแกนที่ 2 ทำงานชุดคำสั่งภายใต้ ฟังก์ชัน Main2( ) หน่วยประมวลผลแกนที่ 3 ทำงานชุดคำสั่งภายใต้ฟังก์ชัน Main3( ) เป็นต้น ซึ่งการใช้พารามิเตอร์ 16 ในคำสั่ง syscall (system call) นั้นหมายถึงการเปิดการทำงาน

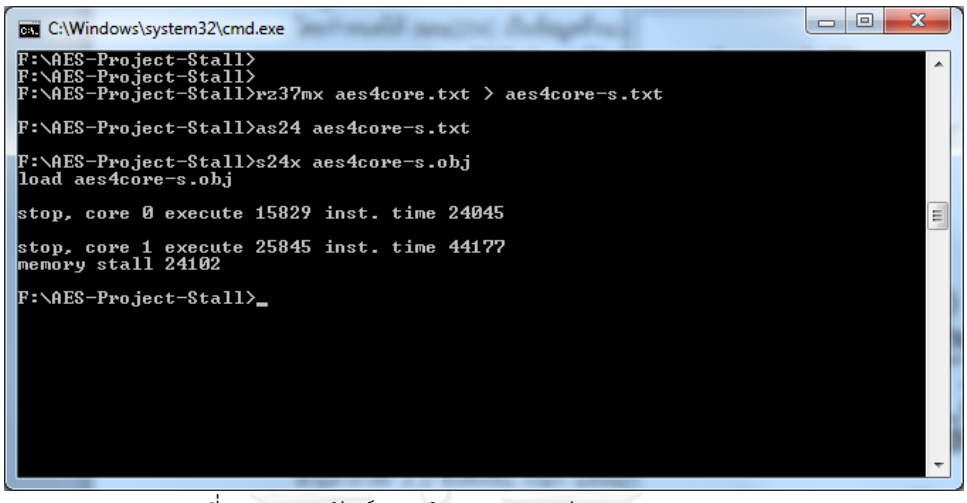
ทั้งหมด 4 แกน ดังภาพที่ 3.21 หากจะการทำงานเพียง หน่วยประมวลผล 2 แกน ต้องเปลี่ยนพารามิเตอร์เป็น syscall(15,0) ดังภาพที่ 3.22 ซึ่งผลลัพธ์การทำงานจะแสดงผลข้อมูลรายละเอียดแยกตามแกนหน่วยประมวลผล โดยจะแสดงคำสั่งที่ถูกทำงาน เวลาที่ใช้ และเวลาที่มีการขัดแย้งกันของหน่วยประมวลผลในการเข้าถึงหน่วยความจำ

```
Main()
      // Start Core #1
      syscall(16,0)
      ......
Main2()
      // Start Core #2
      ....
Main3()
      // Start Core #3
      ....
Main4()
      // Start Core #4
      ....
```

ภาพที่ 3.20 ชุดรหัสคำสั่งในการประมวลผลแต่ละแกน



ภาพที่ 3.21 ผลลัพธ์การทำงานของหน่วยประมวลผล 4 แกน



ภาพที่ 3.22 ผลลัพธ์การทำงานของหน่วยประมวลผล 2 แกน

3.3.4 การพัฒนาชุดรหัสคำสั่ง AES บนหน่วยประมวลผล S2

ชุดคำสั่งการทำงาน AES ออกแบบโดยแบ่งเป็นฟังก์ชันย่อย เพื่อให้สะดวกในการเรียกใช้งานจากชุดคำสั่งหลักในแต่ละแกนการทำงาน โดยชุดคำสั่งถูกแยกออกเป็นฟังก์ชันย่อยสำหรับแกนโดยเฉพาะ เพื่อให้ชุดคำสั่งเป็นอิสระต่อกันในการเรียกใช้งานในแต่ละแกน โดยใช้ชื่อ _cx ต่อท้ายในส่วนฟังก์ชันย่อยและส่วนตัวแปรของแต่ละแกน ซึ่ง cx แทนหมายเลขหน่วยประมวลผลแต่ละแกนแสดงตามภาพที่ 3.23

```
keyExpansion_c1()
     ...
Subbyte_c1()
     ...
ShiftRows_c1()
     ...
MixColumns_c1()
     ...
AddRoundKey_c1(a)
     ...
Main()
     // Start Core #1
     syscall(16,0)
     keyExpansion_c1()
     Subbyte_c1()
     ShiftRows_c1()
     MixColumns_c1()
     AddRoundKey_c1(1)   // Round#1
     ...
Main2()
     // Start Core #2
     keyExpansion_c2()
     ....
```

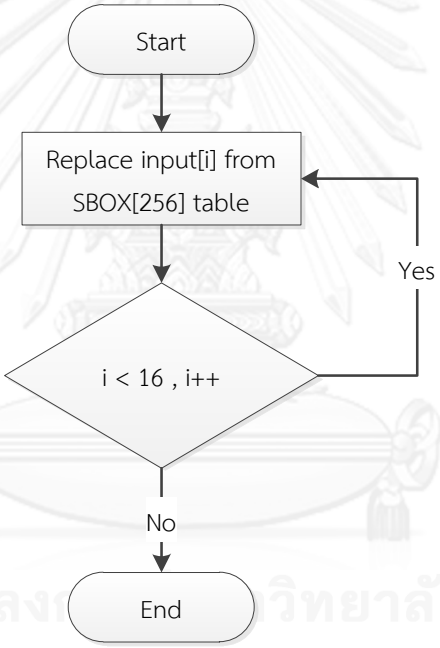
ภาพที่ 3.23 ตัวอย่างรหัสคำสั่งในการประมวลผลแต่ละแกน

3.3.5 Key Expansion

กระบวนการขยายกุญแจ (Key Expansion) ซึ่งประกอบไปด้วย 2 กระบวนการย่อย คือ กระบวนการย่อย Rotword จะทำการเลื่อนไบต์ในแต่ละคำเป็นวงกลมไปทางซ้าย 1 ไบต์ และ กระบวนการย่อย Subword จะทำการแทนที่ไบต์ข้อมูลที่อยู่ในสเตทของกลุ่มกุญแจย่อย โดยอ้างอิงจาก S-box ตัวเดียวกันกับที่ใช้ในกระบวนการย่อย Substitute Byte ผลลัพธ์ที่ได้จากกระบวนการข้างต้นจะนำมา XOR เฉพาะกับตัวแปรแถวลำดับของคำที่เป็นค่าคงที่ เรียกว่า Rcon [i] ซึ่ง i คือลำดับของคำ

3.3.6 SubByte

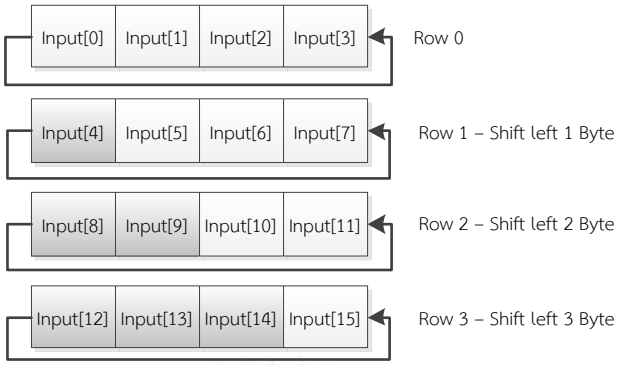
เป็นกระบวนการแทนที่ข้อมูลนำเข้า 16 ไบต์ จากตาราง Sbox 256 ไบต์ โดยอาศัยการสลับที่ด้วยพิกัดแกน X-Y (X คือ ตัวเลขแรก, Y คือ ตัวเลขหลักที่สองของข้อมูลในรูปฐาน 16) ดังภาพที่ 3.24 มีการทำซ้ำข้อมูลทั้งหมด 16 รอบ เพื่อทำการหาข้อมูล



ภาพที่ 3.24 Flow Chart ของการทำงานส่วน Subbyte บนหน่วยประมวลผล

3.3.7 ShiftRows

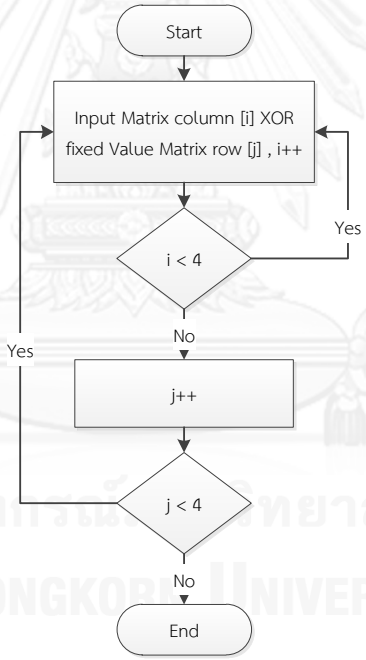
เป็นการเลื่อนข้อมูลในแต่ละแถวตามลำดับแทนที่ด้วยตัวแปร Array input ขนาด 16 ช่อง ซึ่งแต่ละแถวข้อมูลจะถูกเลื่อนด้วยขนาดไบต์ที่เพิ่มขึ้นจากแถวละ 1 ไบต์ ข้อมูลดังภาพที่ 3.25



ภาพที่ 3.25 การเลื่อนข้อมูลในแต่ละแถว

3.3.8 MixColumn

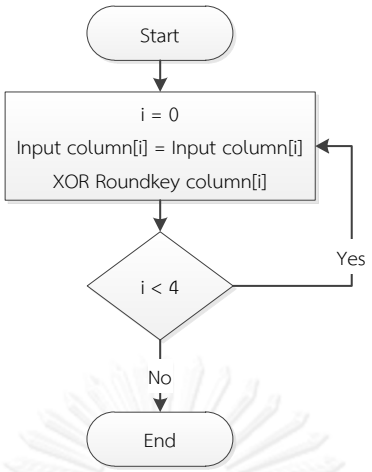
นำข้อมูลแต่ละคอลัมน์ของตัวแปร Column[ ] นำมา XOR กับเมทริกซ์ค่าคงที่ Row[ ] ที่กำหนดไว้ ซึ่งแสดงรอบการทำงานในภาพที่ 3.26 และนำคำตอบที่ได้ใส่เข้าไปแทนที่ในตำแหน่งตัวแปรเดิม



ภาพที่ 3.26 Flow Chart ของการทำงานส่วน MixColumn บนหน่วยประมวลผล

3.3.9 AddRoundKey

จะทำการ XOR เฉพาะระหว่างกลุ่มข้อมูลกับกุญแจย่อย และนำคำตอบที่ได้ใส่เข้าไปแทนที่ในตำแหน่งเดิม ซึ่งกุญแจรหัสจะถูกนำมาใช้ในขั้นตอน AddRoundKey เท่านั้น ด้วยเหตุนี้การเข้ารหัสจึงเริ่มและจบด้วยขั้นตอน AddRoundKey ส่วนขั้นตอนอื่น ๆ สามารถทำงานย้อนกลับได้โดยไม่ต้องใช้กุญแจ ดังแสดงในภาพ 3.27



ภาพที่ 3.27 Flow Chart ของการทำงานส่วน AddRoundKey บนหน่วยประมวลผล

3.3.10 การพัฒนาชุดคำสั่ง AES บนหน่วยประมวลผลแกนเดียว

ลักษณะการพัฒนาชุดคำสั่ง AES บนหน่วยประมวลผลแบบแกนเดียวจะมีการเรียกส่วนการทำงานย่อยภายในฟังก์ชันหลักของหน่วยประมวลผลแกนที่ 1 เพียงแกนเดียวตามภาพที่ 3.28 ซึ่งจะทำการประมวลผลครั้งละ 1 Block ข้อมูลขนาด 16 ไบต์ เมื่อเสร็จสิ้น จึงมีการอ่านข้อมลูลใน Block ถัดไปถัดไปด้วยแกนประมวลผลเดิมจนกว่าจะหมดข้อมูลสำหรับการประมวลผล

```
Main()

     keyExpansion()
     AddFirstRoundKey()
     Subbyte()
     ShiftRows()
     MixColumns()
     AddRoundKey(1)  // Round#1
     ...
     ...
     Subbyte()
     ShiftRows()
     AddRoundKey(10)  // Round#10
```

ภาพที่ 3.28 ชุดคำสั่งในการประมวลผลแต่ละแกนเดียว

3.3.11 การพัฒนาชุดคำสั่ง AES บนหน่วยประมวลผลหลายแกน

ชุดรหัสคำสั่งในการพัฒนา AES บนหน่วยประมวลผล S2 จะมีการเรียกใช้ฟังก์ชันการทำงานย่อยแยกตามแกนการทำงานตาม ภาพที่ 3.29 แสดงชุดคำสั่งที่มีการทำงานอิสระแยกตามแกนทำงานโดยมีลักษณะชุดคำสั่งที่เหมือนกันโดยแตกต่างกันเพียงข้อมูลนำเข้าเพื่อประมวลผลการเข้ารหัสเท่านั้น

```
Main()

     keyExpansion_c1()
     AddFirstRoundKey_c1()
     Subbyte_c1()
     ShiftRows_c1()
     MixColumns_c1()
     AddRoundKey_c1(1)   // Round#1
     .
     .
     Subbyte_c1()
     ShiftRows_c1()
     AddRoundKey_c1(10)  // Round#10
```

```
Main2()

     keyExpansion_c2()
     AddFirstRoundKey_c2()
     Subbyte_c2()
     ShiftRows_c2()
     MixColumns_c2()
     AddRoundKey_c2(1)   // Round#1
     .
     .
     Subbyte_c2()
     ShiftRows_c2()
     AddRoundKey_c2(10)   // Round#10
```

```
Main3()

     keyExpansion_c3()
     AddFirstRoundKey_c3()
     Subbyte_c3()
     ShiftRows_c3()
     MixColumns_c3()
     AddRoundKey_c3(1)   // Round#1
     .
     .
     Subbyte_c3()
     ShiftRows_c3()
     AddRoundKey_c3(10)  // Round#10
```

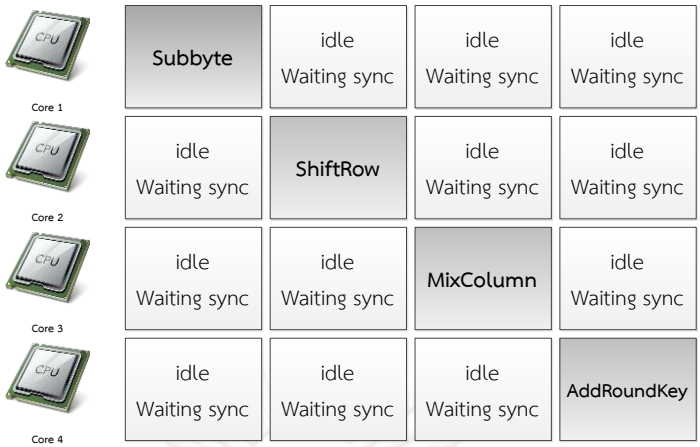
```
Main4()

     keyExpansion_c4()
     AddFirstRoundKey_c4()
     Subbyte_c4()
     ShiftRows_c4()
     MixColumns_c4()
     AddRoundKey_c4(1)   // Round#1
     .
     .
     Subbyte_c4()
     ShiftRows_c4()
     AddRoundKey_c4(10)   // Round#10
```

ภาพที่ 3.29 ตัวอย่างชุดรหัสคำสั่ง AES สำหรับหน่วยประมวลผลแบบ 4 แกน

3.3.12 การทำ AES ด้วยวิธีการประสานเวลา (synchronization)

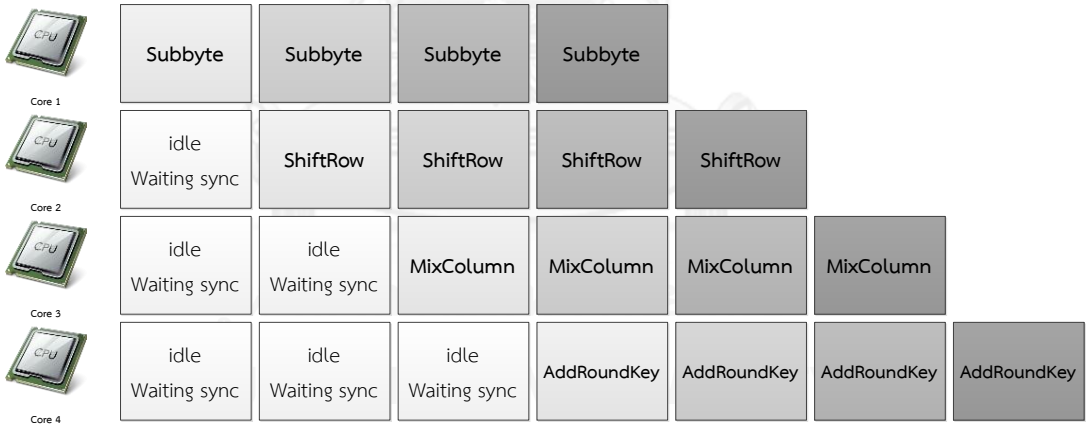
การพัฒนาการเข้ารหัสด้วยตัวประสานเวลาของหน่วยประมวลผลหลายแกนมีลักษณะโครงสร้างตามภาพที่ 3.30 ซึ่งแสดงการทำงานของการประมวลผลการเข้ารหัส AES ของส่วนการทำงานย่อยทั้ง 4 ส่วนคือ Subbyte ShiftRow MixColumn และ AddRoundKey บนหน่วยประมวลผลที่แตกต่างกันโดยอาศัยหลักการประสานเวลากันของแกนการประมวลผลที่ 4 ซึ่งข้อมูลชุดเดียวจะถูกส่งต่อไปยังหน่วยประมวลแกนถัดไปเพื่อทำงาน มีคำสั่ง syscall(17,0) เพื่อระบุการประสานกันของแกนประมวลผลทั้ง 4 แกน การทำงานเช่นนี้ในช่วงเวลาการทำงานหนึ่งจะมีเพียงแกนการทำงานเดียวที่มีการใช้งานหน่วยความจำ เมื่อประมวลผลข้อมูลกลุ่มแรกเรียบร้อยจะทำการวนประมวลผลกลุ่มข้อมูลถัดไปจนครบจำนวนกลุ่มของข้อมูลนำเข้า



ภาพที่ 3.30 ลักษณะโครงสร้างการทำงาน AES บนหน่วยประมวลผลแบบ synchronization

3.3.13 การทำ AES ด้วยวิธีการประสานเวลา (synchronization) ร่วมกับ pipeline

หน่วยประมวลผลแต่ละแกนทำงานโดยมีการใช้งานแบบประสานเวลากัน ซึ่งมีการประมวลผลฟังก์ชันการทำงานเดิมตลอดเวลาตามภาพที่ 3.31 โดยข้อมูลจะถูกจะส่งไปเพื่อประมวลผลการทำงานถัดฟังก์ชันการทำงานอื่นที่หน่วยประมวลผลถัดไป ซึ่งหลังจากเสร็จการทำงานแต่ละช่วงเวลาจะมีการเรียกใช้งาน syscall(17,0) เพื่อกำหนดการประสานเวลาของแต่ละแกนเพื่อให้ข้อมูลถูกส่งต่อไปยังแกนถัดไปและประมวลผลได้ถูกต้อง

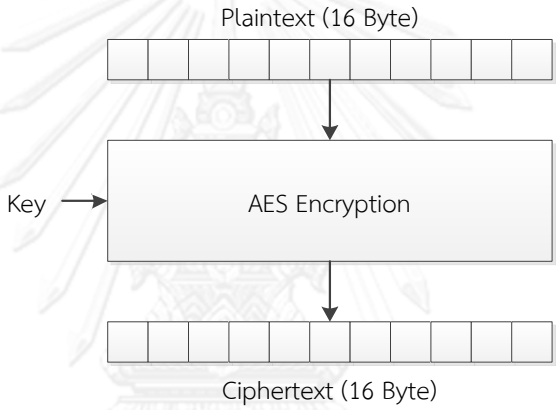


ภาพที่ 3.31 โครงสร้าง AES แบบ Pipeline บนหน่วยประมวลผลด้วย synchronization

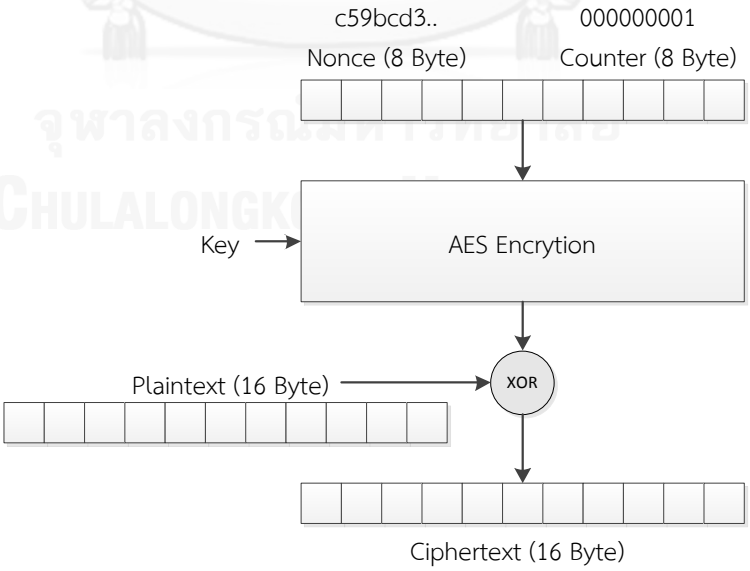
3.3.14 การพัฒนาการเข้ารหัสแบบ AES ด้วยโหมดตัวนับ (CTR)

กระบวนการเข้ารหัสลับแบบเป็นกลุ่มนั้นมีปัญหาที่สำคัญ คือการใช้กุญแจเดิมหลายรอบ ซึ่งจะทำให้ได้ข้อมูลแบบเดิมออกมาเรื่อยๆ ถึงแม้ว่าจะมีการคิดค้นวิธีการเข้ารหัสใหม่ที่มีความสามารถในการวิเคราะห์ข้อมูลที่แท้จริงคืออะไรแล้วก็ตาม ซึ่งแฮกเกอร์ก็จะเห็นข้อมูลซ้ำแบบเดิมก็สามารถเปิดเผยข้อมูลบางอย่างได้ ซึ่งกระบวนการเข้ารหัสเป็นกลุ่มที่พื้นฐานที่สุดนั้นคือ Electronic Code Book (ECB) ซึ่งมีการใช้งานกุญแจเดิมในการเข้ารหัสในทุกๆ ข้อมูลนำเข้า ซึ่งปัจจัยนี้เป็นเหตุให้โหมดการเข้ารหัสนี้ไม่เป็นที่นิยมนัก จึงได้มีการพัฒนา

AES ด้วยโหมด Counter (CTR) ซึ่งอาศัยการติดตัวเลขที่เปิดเผย หรือเรียกว่า nonce กับข้อมูล ซึ่งตัวเลขนี้จะเปลี่ยนไปเรื่อยๆ ในแต่ละกลุ่มข้อมูล การพัฒนาชุดรหัสคำสั่งของ AES บนหน่วยประมวลผล S2 นี้จะเพิ่มในส่วนของ nonce โดยกำหนดเป็นเลขสุ่มขนาด 8 ไบต์ และตัวนับขนาด 8 ไบต์ เริ่มจาก 0 เพิ่มขึ้นทีละ 1 ตามจำนวนบล็อกข้อมูลนำเข้า แล้วทำการเข้ารหัสด้วยชุดรหัสคำสั่งเดิมในโหมด ECB แล้วทำการ XOR ด้วยชุดข้อมูลนำเข้าหลังจากเข้ารหัสเสร็จสิ้นแสดงดังภาพที่ 3.33 ซึ่งการพัฒนาเพื่อเปรียบเทียบการทำงานบนหน่วยประมวลผลแบบ pipeline ซึ่งวิธีการนี้จะส่งผลให้มีความซับซ้อนมากยิ่งขึ้นโดยได้ทำการเปรียบเทียบการทำงานระหว่างการประมวลผลระหว่างโหมดการเข้ารหัสด้วย ECB ตามภาพที่ 3.32 และ CTR ตามภาพที่ 3.33



ภาพที่ 3.32 การพัฒนา AES ด้วยโหมด ECB

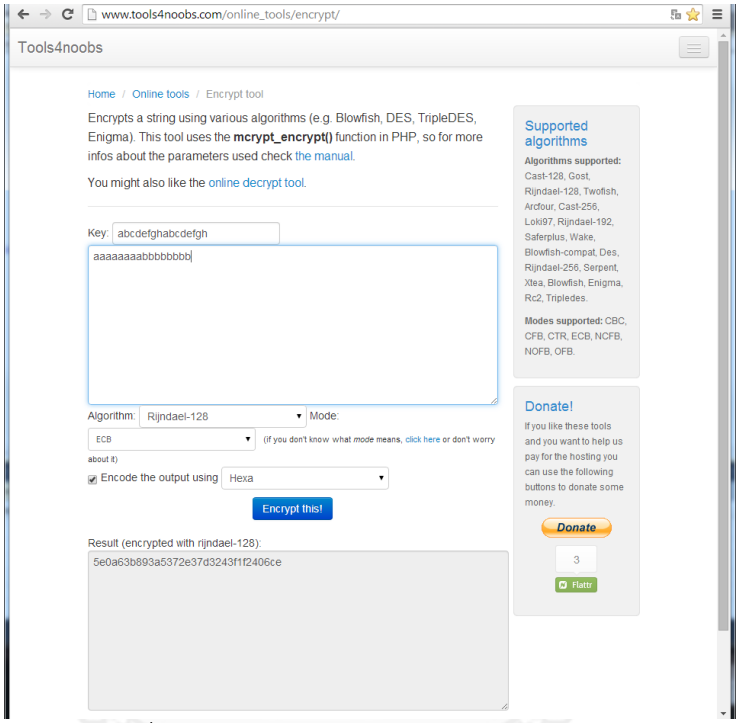


ภาพที่ 3.33 การพัฒนา AES ด้วยโหมด CTR

3.3.15 การตรวจสอบความถูกต้องของข้อมูล

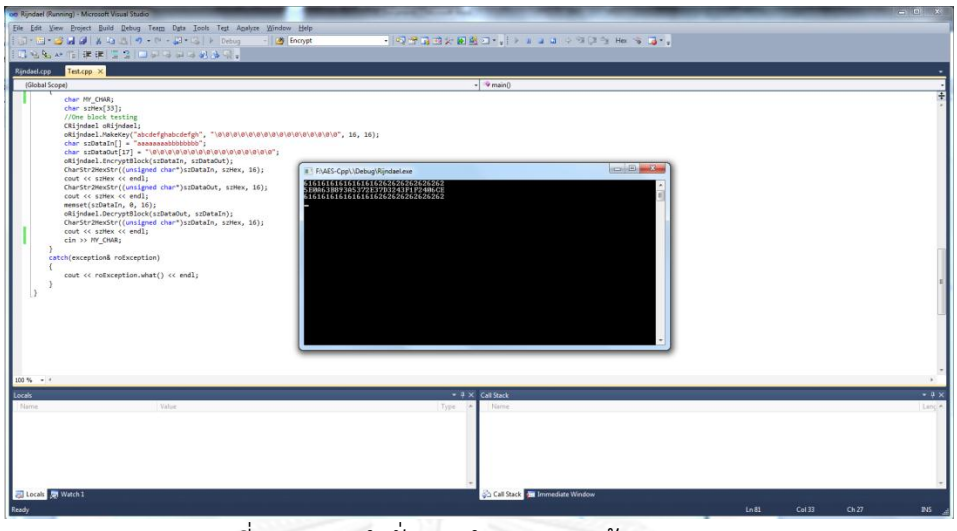
การตรวจสอบความถูกต้องของการทำงานโดยดูจากผลลัพธ์ที่ได้จากการประมวลผลแต่ละแบบด้วย 3 แหล่งอ้างอิงที่น่าเชื่อถือ

เวป tools4noobs เป็นผู้ให้บริการรวบรวมชุดคำสั่งให้ใช้บริการบนอินเทอร์เน็ต รวมถึงชุดคำสั่งการเข้ารหัสแบบ AES และแบบอื่นๆ ฟรี โดยที่การใช้งานใส่ชุดข้อมูลและกุญแจในการเข้ารหัส แล้วเลือกลักษณะการทำงานเข้ารหัสที่ต้องการ ตามภาพที่ 3.34



ภาพที่ 3.34 เวป http://www.tools4noobs.com

ชุดคำสั่งที่พัฒนาด้วยภาษา C++ โดย George Anescu มีการพัฒนาตามหลักการมาตรฐาน AES ซึ่งเผยแพร่ในเวป codeproject ที่รวบรวมชุดคำสั่งพัฒนาขึ้นเพื่อใช้ในการศึกษาและพัฒนา การทำงานทดสอบบนโปรแกรม Visual Studio 2012 โดยกำหนดข้อมูลชุดตัวแปร szDataIn[] และกุญแจในพารามิเตอร์ Makekey ตามภาพที่ 3.35



ภาพที่ 3.35 ชุดคำสั่งการทำงาน AES ด้วยภาษา C++

ตัวอย่างจากหนังสือ Cryptographer and Network Security โดย William Stallings ซึ่งในบทที่ 5 ได้กล่าวถึงกระบวนการเข้ารหัสและตัวอย่างชุดคำสั่งการเข้ารหัสแบบ AES ขนาด 128 บิต พร้อมทั้งแสดงผลแต่ละขั้นตอนการทำงานในแต่ละรอบ

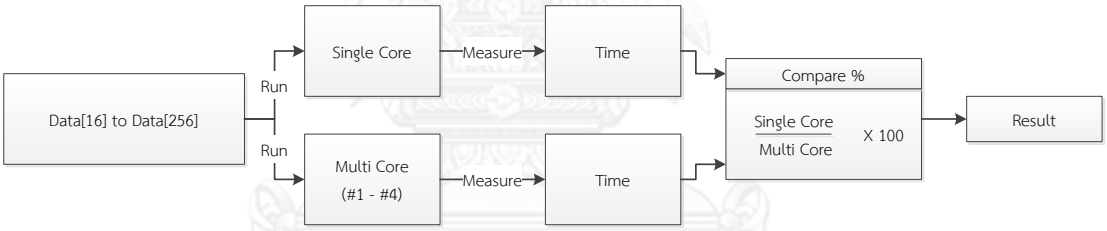
บทที่ 4
การทดลองและผลการทดลอง

4.1 เครื่องมือที่ใช้

การทดลองนี้ได้ทำการทดลองบนโปรแกรมจำลองหน่วยประมวล S2 ซึ่งเป็นหน่วยประมวลผลการทำงานแบบแกนเดียวและหลายแกน โดยโปรแกรมถูกเขียนด้วยภาษา RZ บนเครื่องคอมพิวเตอร์ส่วนบุคคลที่มีหน่วยประมวลผลกลาง AMD FX-6300 6 core CPU 3.5GHz หน่วยความจำ 8GB ระบบปฏิบัติการ Windows 7 64 บิต

4.2 วิธีการทดลอง

ทำการทดสอบการเข้ารหัสด้วยข้อมูลสุ่มขนาด 16 ไบต์ ไปจนถึงข้อมูลสูงสุดที่ 256 ไบต์ โดยมีการแบ่งข้อมูลออกเป็นกลุ่มละ 16 ไบต์ ให้มีการทำงานบนหน่วยประมวลผลแกนเดียวและการทำงานบนหน่วยประมวลผลหลายแกน ซึ่งด้วยขนาดข้อมูลจะทำให้สามารถแบ่งงานออกเป็น 16 กลุ่ม โดยที่หน่วยประมวลผลแต่ละแกนจะได้รับงานในอัตราส่วนที่เท่ากัน ความถูกต้องของข้อมูลจะอ้างอิงจากงานของบุคคลอื่นที่พัฒนาการเข้ารหัสแบบ AES ดังภาพที่ 4.1 แล้วทำการบันทึกค่าเวลาเพื่อเปรียบเปอร์เซ็นต์ประสิทธิภาพที่เพิ่มขึ้นเมื่อมีการเพิ่มแกนประมวลผล



ภาพที่ 4.1 วิธีการทดสอบการทำงาน

4.3 ผลการทดลอง

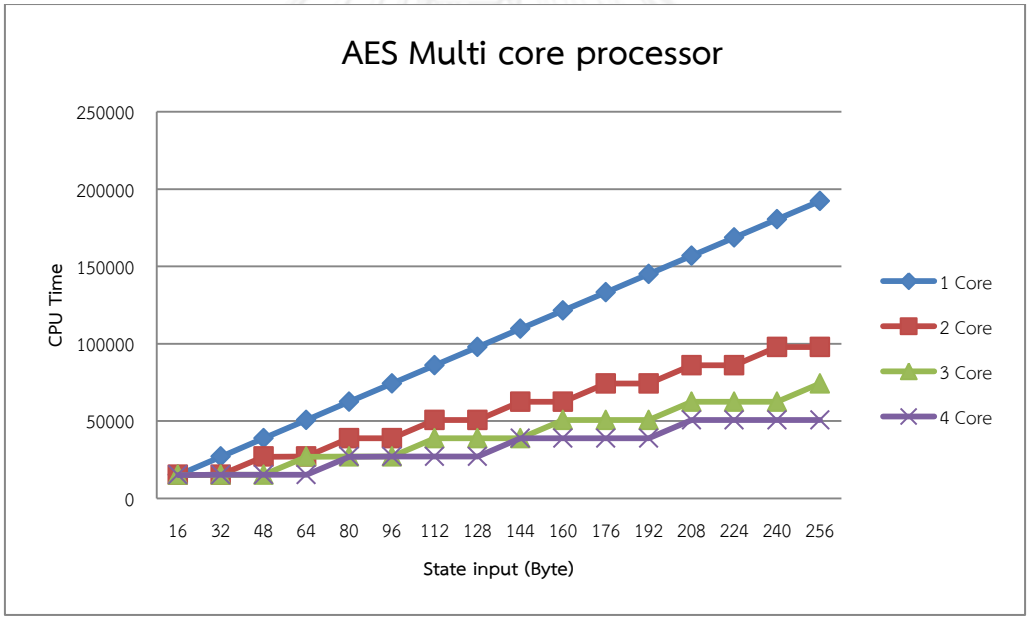
4.3.1 ผลการทำงานของหน่วยประมวลแบบแกนเดียวและหลายแกน

ผลการทดสอบแสดงดังตารางที่ 4.1 ซึ่งแสดงผลลัพธ์เวลาที่ใช้ในการเข้ารหัส AES บนหน่วยประมวลผลแกนเดียวและเมื่อเพิ่มแกนการประมวลผลด้วยข้อมูลขนาด 16 ไบต์ จนถึง 256 ไบต์ ที่เมื่อแกนประมวลผลมากขึ้นจะเวลาที่ใช้ลดลงในลักษณะเชิงเส้นตามกราฟ ซึ่งจากผลการทดลองตามตารางที่ 4.1 และภาพกราฟที่ 4.2 นั้นไม่ได้พิจารณาในส่วนของการขัดแย้งกันในการใช้งานหน่วยความจำซึ่งแสดงให้เห็นว่ามีประสิทธิภาพถึง 379 เปอร์เซ็นต์ แต่เมื่อทำการทดลองโดยการเพิ่มในส่วนของการเข้าถึงหน่วยความจำและการขัดแย้งกันของข้อมูลโดยที่หน่วยประมวลผลจะช้าลง 2 รอบนาฬิกา จากผลลัพธ์การทดลองตามตารางที่ 4.2 และภาพกราฟที่ 4.3 จะพบว่าเมื่อเพิ่มจำนวนแกนการทำงานแต่ความเร็วไม่เพิ่มขึ้น เนื่องมาจากความขัดแย้งกันของการเข้าถึงหน่วยความจำ ตารางที่ 4.3 จะพบว่าเมื่อเพิ่มแกนการทำงานมากขึ้นความขัดแย้งจะเพิ่มขึ้นเช่นกัน อย่างไรก็ตามหากมีการเพิ่มในส่วนของ Cache ระดับ 1 ซึ่งเป็นหน่วยความจำแยกกันที่มีความใกล้ชิดกับหน่วยประมวลผล

ในแต่ละแกนการทำงาน ซึ่งผลน่าจะมีแนวโน้มที่เหมือนกับภาพที่ 4.2 เนื่องมาจากไม่จำเป็นต้องมีการขัดแย้งกันในการเข้าถึงข้อมูลในหน่วยความจำ

ตารางที่ 4.1 เวลาในการเข้ารหัส AES โดยไม่รวมเวลาการขัดแย้งการใช้หน่วยความจำ

ข้อมูล (ไบต์)	เวลาทำงาน AES บนหน่วยประมวลผล S2				เปรียบเทียบผล
	1 Core	2 Core	3 Core	4 Core	
16	15131	-	-	-	100 %
64	50552	26993	26938	15186	332 %
128	97780	50607	38800	26993	362 %
192	145008	74221	50607	38800	373 %
256	192236	97835	74166	50607	379 %



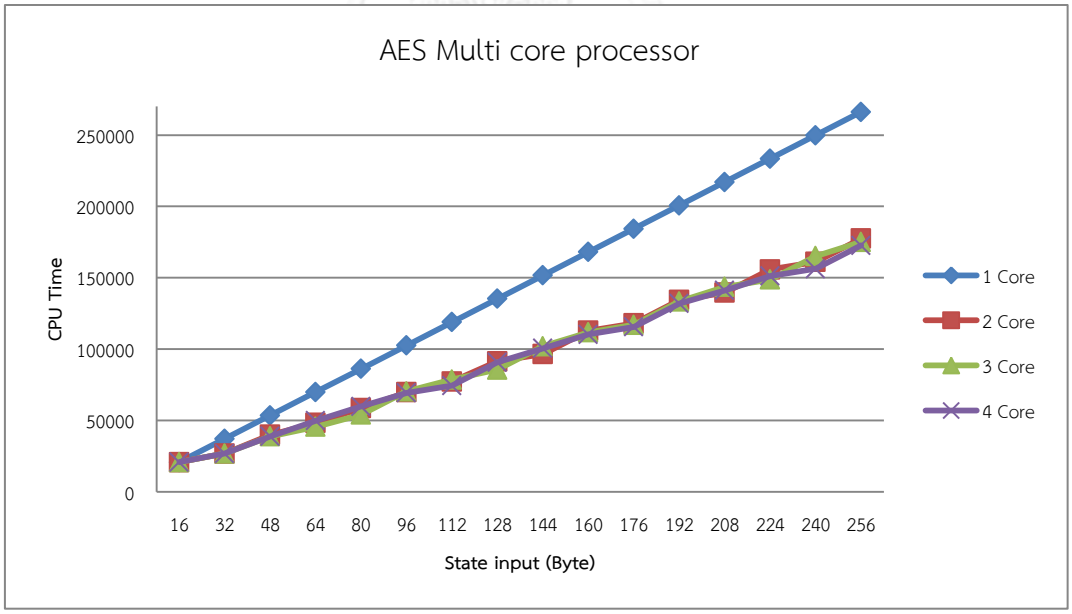
ภาพที่ 4.2 กราฟแสดงผลลัพธ์ CPU Time การทำงานของ AES บนหน่วยประมวลผล S2

ตารางที่ 4.2 เวลาการเข้ารหัส AES ด้วยผลรวมเวลา Stall ของหน่วยความจำ

ข้อมูล (ไบต์)	เวลาทำงาน AES บนหน่วยประมวลผล S2				เปรียบเทียบผล
	1 Core	2 Core	3 Core	4 Core	
16	20833	-	-	-	100%
64	69901	48358	45664	49924	140%
128	135325	91428	85669	90828	149%
192	200749	134393	133459	131910	152%
256	266173	177448	175088	172687	154%

ตารางที่ 4.3 ผลลัพธ์ Stall time ในการเข้ารหัส (ผลรวม Stall ของทุกแกนการทำงาน)

ข้อมูล (ไบต์)	1 Core	2 Core	3 Core	4 Core	เปรียบเทียบผล
32	10346	19456	-	-	188%
64	19444	35475	62383	81733	420%
128	37640	67695	119257	150909	401%
256	74000	131995	222702	289148	391%



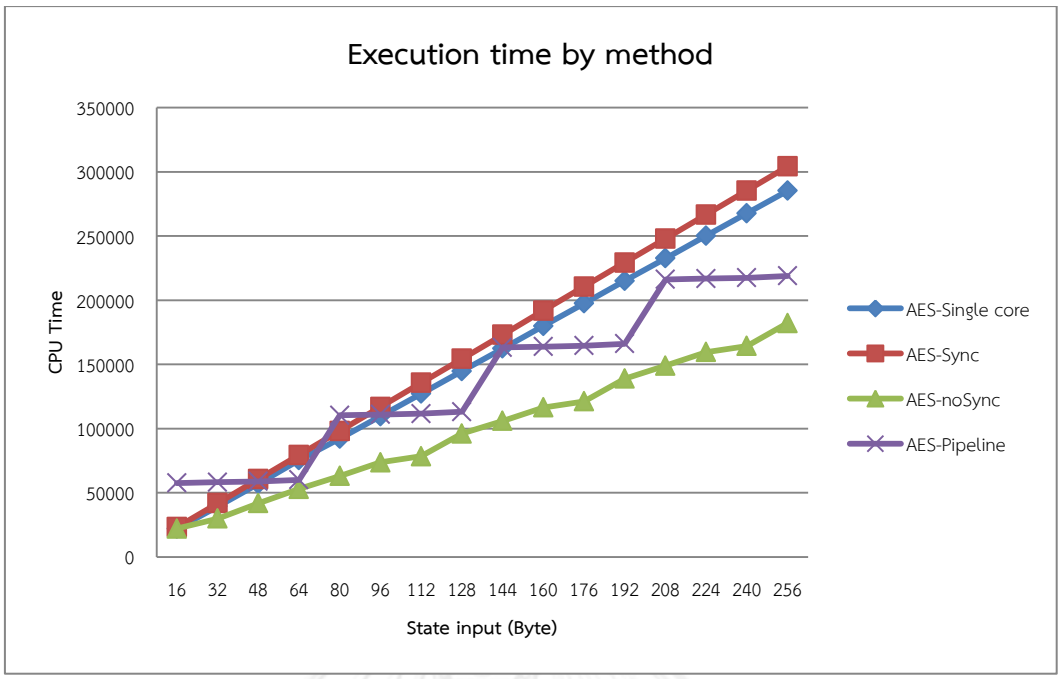
ภาพที่ 4.3 กราฟแสดงผลลัพธ์ CPU Time ด้วยหน่วยประมวลผล S2 รวมกับ Stall memory

4.3.2 ผลการทำงานของหน่วยประมวลผลด้วยวิธีการประสานเวลาการทำงาน

ผลการทดสอบแสดงดังตารางที่ 4.4 และภาพที่ 4.5 แสดงผลลัพธ์เวลาที่ใช้ในการเข้ารหัส AES บนหน่วยประมวลผลหลายแกนแยกตามลักษณะวิธีการประมวลผลเทียบกับการทำงานบนหน่วยประมวลผลแกนเดียว ซึ่งผลลัพธ์เมื่อพิจาณาที่ขนาดสูงสุด 256 ไบต์ แสดงให้เห็นว่าความเร็วในการประมวลผลแบบไม่มีการประสานเวลา (No synchronization) ให้เวลาที่ดีที่สุดถึง 156 เปอร์เซ็น รองลงมาคือแบบ Pipeline ที่ 130 เปอร์เซ็น และแบบประสานเวลาการทำงาน (synchronization) ลดลงที่ 94 เปอร์เซ็น ถึงแม้ว่าการทำงานแบบไม่มีการประสานเวลาให้ผลลัพธ์ความเร็วในการประมวลผลที่ดีสุดแต่เวลาที่เสียในการเข้าใช้งานหน่วยความจำก็มีเพิ่มมากที่สุดเช่นกันถึง 378 เปอร์เซ็น ในขณะที่การทำงานแบบ Pipeline เสียเวลาเพิ่มขึ้นเพียง 266 เปอร์เซ็น เมื่อเทียบกับการทำงานแบบแกนเดียวแสดงตามตารางที่ 4.5 และภาพที่ 4.6 การทำงานแบบ Pipeline มีค่าแนวโน้มเวลาสูงกว่าการทำงานแบบไม่มีการประสานเวลาเนื่องเวลาจากการประมวลผล AES มีการคำนวณทางคณิตศาสตร์ที่อยู่มากในแต่ละส่วนการทำงาน ซึ่งในช่วงการรอการประสานเวลาการทำงานของแกนอื่นๆ หน่วยประมวลผลบางแกนอยู่ในสถานะไม่ทำงานไม่ว่าจะคำสั่งใดๆ เพื่อรอสัญญาณการประสานเวลา ในขณะที่หน่วยประมวลผลแบบไม่มีการประสานเวลาสามารถประมวลผลได้ตลอดเวลา

ตารางที่ 4.4 ผลลัพธ์เวลาในประมวลการเข้ารหัสแยกตามลักษณะการประมวลผล

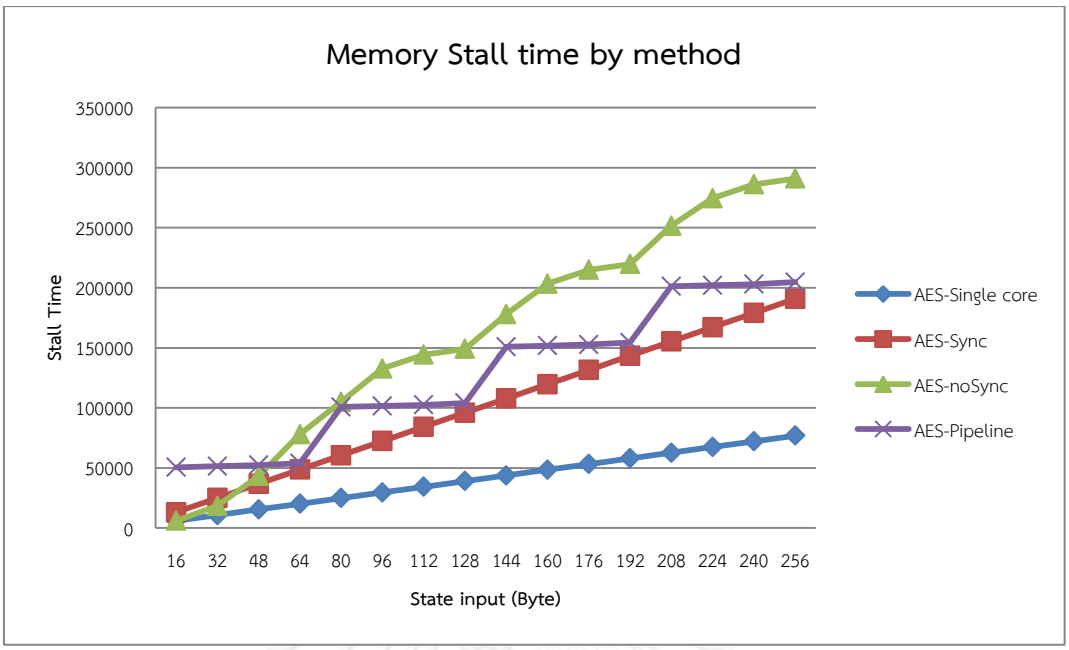
ข้อมูล (ไบต์)	แกนเดียว	หน่วยประมวลผลหลายแกน					
		NoSync	ผลลัพธ์	Sync	ผลลัพธ์	Pipeline	ผลลัพธ์
16	21884	22061	99%	23173	94%	57573	38%
32	39388	29952	131%	41938	93%	58233	67%
64	75424	52929	142%	79414	94%	60134	125%
128	144796	96191	150%	154366	93%	113090	128%
256	285340	182139	156%	304206	94%	218986	130%



ภาพที่ 4.4 เวลาการประมวลผลด้วยวิธีการประมวลผลแบบต่างๆ

ตารางที่ 4.5 ผลลัพธ์ Stall time ในการเข้ารหัสตามลักษณะการประมวลผล

ข้อมูล (ไบต์)	แกนเดียว	หน่วยประมวลผลหลายแกน					
		NoSync	ผลลัพธ์	Sync	ผลลัพธ์	Pipeline	ผลลัพธ์
16	5977	6005	100%	13062	218%	50534	845%
32	10689	18465	172%	24913	233%	51455	481%
64	20145	78166	388%	48623	241%	53910	267%
128	39057	149141	381%	96041	245%	104162	266%
256	76881	291006	378%	190865	248%	204672	266%



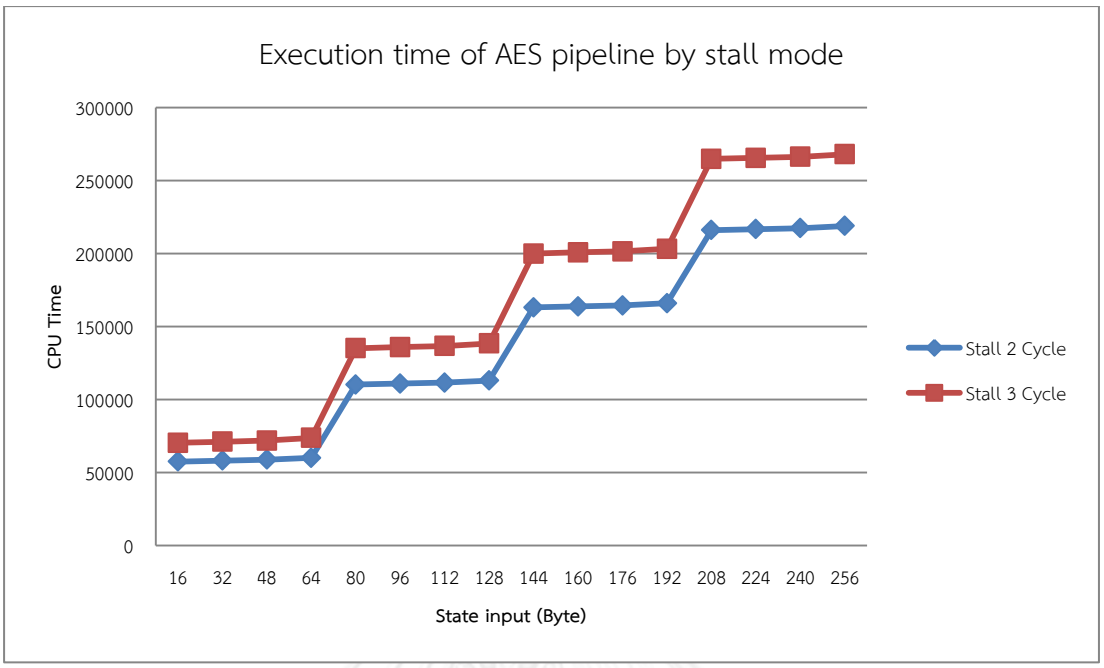
ภาพที่ 4.5 เวลาการเข้าใช้งานหน่วยความจำด้วยวิธีการประมวลแบบต่างๆ

4.3.3 ผลการทำงานด้วยเวลาการเข้าใช้งานหน่วยความจำ (Stall time)

เนื่องมาจากการทำงานเข้ารหัสลับแบบ AES บนหน่วยประมวลผลมีการใช้งานหน่วยความจำเพื่อเข้าถึงข้อมูลหลายคำสั่ง จึงได้พิจารณาเพิ่มเวลาในการเข้าถึงหน่วยความจำของคำสั่งเขียนและอ่านบนหน่วยประมวลผลโดยมีการปรับเปลี่ยนเวลาในการเข้าถึงหน่วยความจำเป็น 3 สัญญาณนาฬิกา ซึ่งจากตารางที่ 4.6 แสดงให้ถึงเวลาที่ใช้เพิ่มขึ้นในการประมวลผลการเข้ารหัสลับเป็น 122.44 เปอร์เซ็นต์หรือเพิ่มขึ้น 1.22 เท่า เมื่อพิจารณาที่ข้อมูลขนาด 256 ไบต์ ด้วยวิธีการประมวลผลแบบ Pipeline และตารางที่ 4.7 แสดงเวลาการใช้งานหน่วยความที่เพิ่มขึ้นเป็น 159.25 เปอร์เซ็น จากภาพที่ 4.6 และ 4.7 แสดงแนวโน้มของเวลาในการประมวลผลและการใช้งานหน่วยความจำเมื่อเพิ่มขนาดของข้อมูลการเข้ารหัส

ตารางที่ 4.6 ผลลัพธ์เวลาในการเข้ารหัส (Execution time)

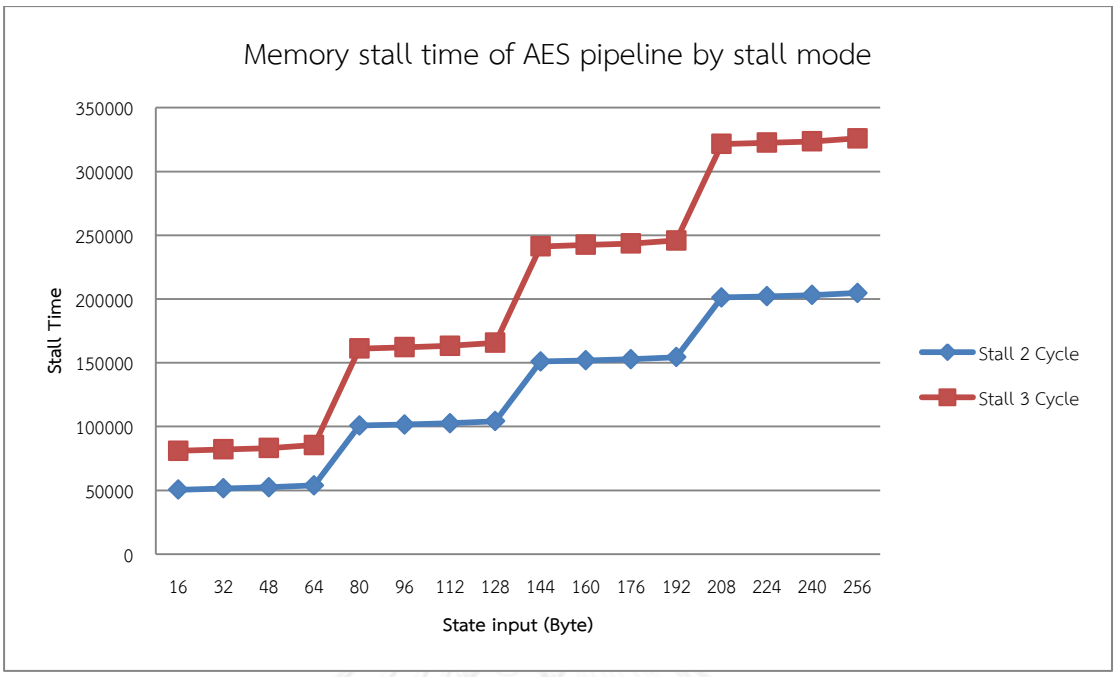
ข้อมูล (ไบต์)	2 Stall	3 Stall	เปรียบเทียบผล
16	57573	70378	122.24%
32	58233	71172	122.21%
64	60134	73752	122.64%
128	113090	138547	122.51%
256	218986	268137	122.44%



ภาพที่ 4.6 เวลาในประมวลผล AES ที่มีการเข้าถึงหน่วยความจำ 2 และ 3 สัญญาณนาฬิกา

ตารางที่ 4.7 ผลลัพธ์เวลาในการเข้าใช้งานหน่วยความ (Stall time)

ข้อมูล (ไบต์)	2 Stall	3 Stall	เปรียบเทียบผล
16	50534	80987	160.26%
32	51455	82066	159.49%
64	53910	85531	158.65%
128	104162	165671	159.05%
256	204672	325951	159.25%



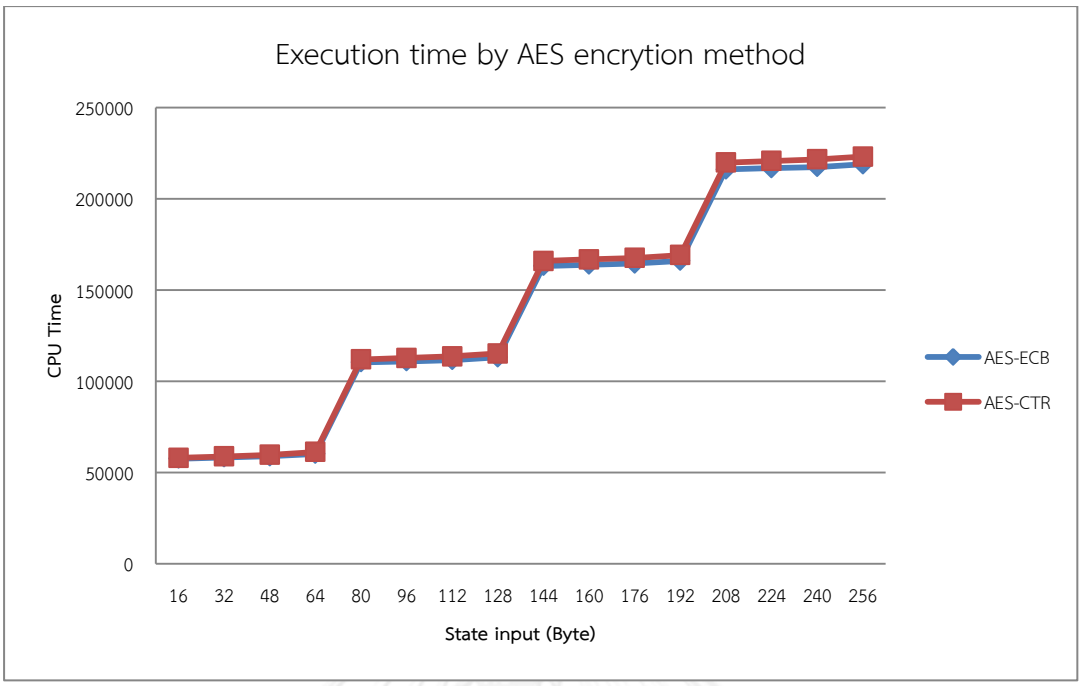
ภาพที่ 4.7 เวลาในการเข้าใช้งานหน่วยความ (Stall time)

4.3.4 ผลการทำงานด้วยวิธีการประมวลเข้ารหัสด้วยโหมดตัวนับ (CTR)

การทำงานการเข้ารหัส AES ด้วยโหมดการเข้ารหัสแบบ ECB และ CTR แสดงผลลัพธ์เวลาการทำงานดังตารางที่ 4.8 ซึ่งการทำงานในโหมด CTR มีการทำงานที่ช้ากว่าแบบ ECB 1.9 เปอร์เซ็นต์ ที่ขนาดข้อมูล 256 ไบต์ และเสียเวลาในการเข้าถึงหน่วยความจำ 1.8 เปอร์เซ็นต์ ตามตารางที่ 4.9 ซึ่งจากข้อมูลการเปรียบเทียบแสดงให้เห็นว่าการทำงานของ ECB และ CTR มีการทำงานที่ใช้เวลาไม่ต่างกัน ตามภาพที่ 4.8 และ 4.9 แสดงแนวโน้มการทำงานเมื่อมีการเข้ารหัสตั้งแต่ข้อมูลขนาด 16 ไบต์จนถึง 256 ไบต์

ตารางที่ 4.8 ผลลัพธ์เวลา (Execution time) ในการเข้ารหัสด้วยวิธี ECB และ CTR

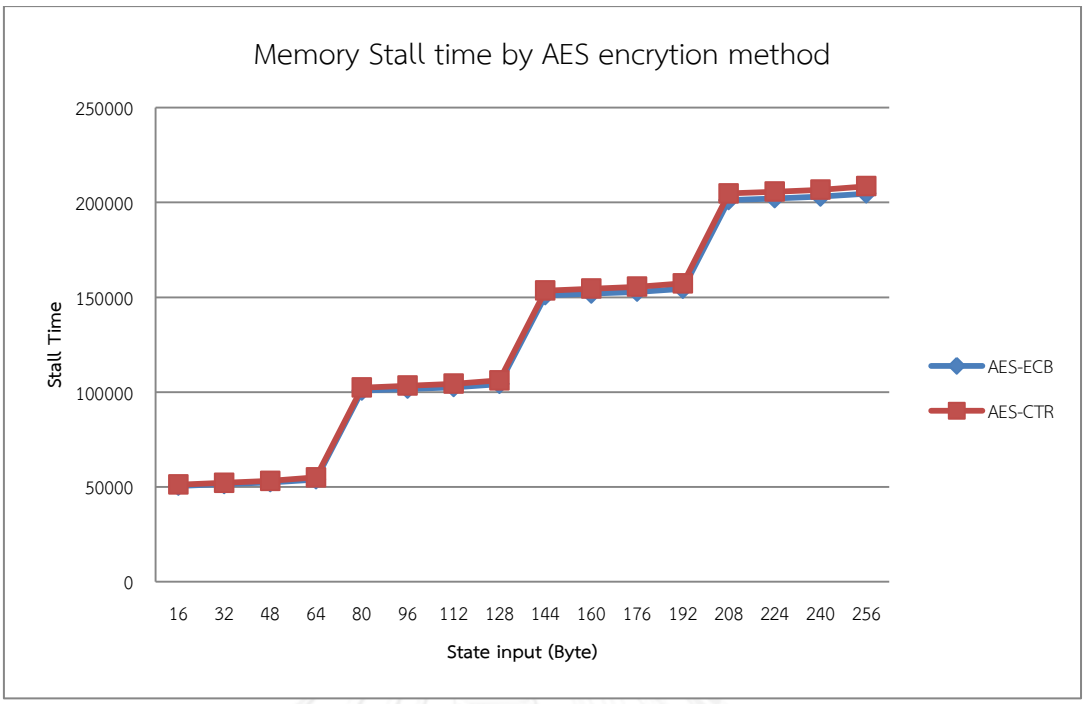
ข้อมูล (ไบต์)	ECB	CTR	เปรียบเทียบผล
16	57573	57949	100.65%
32	58233	58794	100.96%
64	60134	61189	101.75%
128	113090	115175	101.84%
256	218986	223185	101.91%



ภาพที่ 4.8 กราฟผลลัพธ์เวลา (Execution time) ในการเข้ารหัสด้วยวิธี ECB และ CTR

ตารางที่ 4.9 ผลลัพธ์เวลาในการเข้าใช้งานหน่วยความจำ (Stall time) ด้วยวิธี ECB และ CTR

ข้อมูล (ไบต์)	ECB	CTR	เปรียบเทียบผล
16	50534	51136	101.19%
32	51455	52142	101.33%
64	53910	54934	101.89%
128	104162	106129	101.88%
256	204672	208520	101.88%



ภาพที่ 4.9 กราฟเวลาในการเข้าใช้งานหน่วยความจำ (Stall time) ด้วยวิธี ECB และ CTR

บทที่ 5
สรุปผลวิจัยและข้อเสนอแนะ

5.1 สรุปผลงานวิจัย

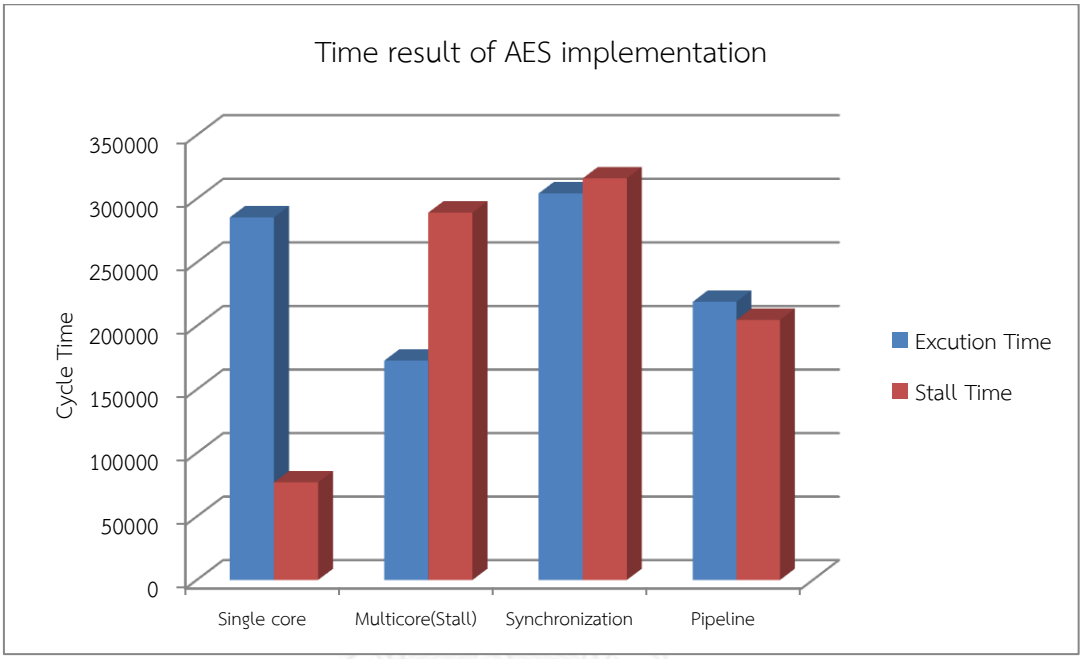
งานวิจัยนี้นำเสนอวิธีการปรับปรุงการเข้ารหัสลับเพื่อเพิ่มประสิทธิภาพการทำงานในส่วนของความเร็วและทรัพยากร บนหน่วยประมวลผลแบบหลายแกน โดยมีแนวความคิดในการกระจายข้อมูลขนาดใหญ่ออกเป็นส่วนย่อย และกระจายงานให้แต่ละแกนประมวลผล ซึ่งอาศัยข้อดีของหน่วยประมวลหลายแกนที่สามารถทำงานพร้อมกัน

การทดลองได้ทำการทดลองบนชุดข้อมูลขนาด 256 ไบต์ โดยมีการเปรียบเทียบการทำงานการเข้ารหัสบนหน่วยประมวลผลแกนเดียวและหน่วยประมวลผลหลายแกน โดยมีการวัดระยะเวลาการทำงานของการเข้ารหัสบนหน่วยประมวลผลแกนเดียว เทียบกับการวิธีการเข้ารหัสแบบแบ่งข้อมูลเท่ากันและวิธีการจัดการข้อมูลด้วยตัวขัดจังหวะบนหน่วยประมวลผลหลายแกน แล้วทำการเปรียบเทียบเวลาการทำงานและทรัพยากรที่ใช้

เมื่อนำผลการเปรียบเทียบเวลาการทำงานการเข้ารหัสบนหน่วยประมวลผลแกนเดียวและหน่วยประมวลผลหลายแกนของชุดข้อมูล ผลคือวิธีการทำงานที่ปรับปรุงมีประสิทธิภาพเพิ่มขึ้น 379 เปอร์เซ็นต์ ซึ่งค่าที่ได้ยังห่างจากค่าในอุดมคติที่ 400 เปอร์เซ็นต์ เพราะยังมีส่วนของการเข้าถึงหน่วยความจำร่วมและกระบวนการจัดการข้อมูล และเพื่อรวมพิจารณาถึงเวลาในการขัดแย้งกันของหน่วยความจำซึ่งกำหนดที่ 2 สัญญาณนาฬิกา พบว่าความเร็วนั้นลดลงเหลือ 154 เปอร์เซ็นต์ โดยจำนวนหน่วยประมวลผลมากขึ้นไม่ได้ช่วยให้มีการทำงานที่เร็วขึ้นตามไปด้วย อันเนื่องมาจากการขัดแย้งกัน (Stall) ในการใช้งานหน่วยความจำเมื่อหน่วยความจำใช้เวลาขนาด 2 สัญญาณนาฬิกา ซึ่งจากวิธีการนี้แสดงให้เห็นว่าการใช้หน่วยประมวลผลหลายแกนขนาด 2 แกน ที่ใช้ในหน่วยประมวลผลการทำงานให้ประสิทธิภาพที่เหมาะสมที่สุดทั้งเรื่องความเร็วและทรัพยากร

เนื่องจากปัญหาการใช้งานหน่วยความจำส่งผลให้การทำงานของการเข้ารหัสของของหน่วยประมวลผลหลายแกนไม่ได้ในทางเชิงอุดมคติ จึงมีการพัฒนาเรื่องการประสานเวลากันของหน่วยประมวลผล ซึ่งผลการทดลองด้วยวิธีการพัฒนาด้วยการประมวลผลแบบประสานเวลาและ Pipeline แสดงให้เห็นเวลาที่ขัดแย้งในการใช้งานหน่วยความจำลดลงในกรณีที่ประมวลผลด้วยหน่วยประมวลผลหลายแกน แต่เวลาที่ใช้ในการประมวลผลการทำงานมากขึ้นเนื่องจากเสียเวลารอในช่วงเวลาในการประสานเวลากันของหน่วยประมวลผล ซึ่งการเข้ารหัสแบบ AES มีการทำงานคำนวณทางคณิตศาสตร์อยู่มากซึ่งเป็นเหตุผลให้การทำงานแบบไม่มีการประสานเวลาการทำงานสามารถประมวลผลได้ทันทีไม่จำเป็นต้องรอการประสานกันของแกนอื่นๆ แสดงเวลาดังภาพที่ 5.1 ถึงแม้แนวโน้มเวลาในการเข้ารหัสด้วยวิธีการดังกล่าวไม่ดีกว่าแบบไม่มีตัวประสานเวลาการทำงานแต่ผลที่ได้ดีกว่าการทำงานแบบหน่วยประมวลผลแกนเดียว ซึ่งหากมีการออกแบบการใช้งานหน่วยความจำที่ดีขึ้นก็จะส่งผลให้เวลาที่ใช้ในการประมวลผลการเข้ารหัสดีขึ้นตาม โดยการทดลองการเพิ่มเวลาเมื่อมีการใช้งานหน่วยความจำ 3 นาฬิกาแสดงให้เห็นเวลาเป็น 159 เปอร์เซ็นต์เมื่อเทียบกับแบบ 2 สัญญาณนาฬิกา ส่วนเรื่องการทดลองเรื่องความปลอดภัยยังแสดงให้เห็นถึงวิธีการเข้ารหัส AES ด้วย

รูปแบบ ECB เทียบกับ CTR เมื่อมีการพัฒนาด้วยหน่วยประมวลผลหลายแกน การทำงานแบบ CTR ใช้เวลาในการประมวลผลและเวลาในการใช้งานหน่วยความจำมากกว่าแบบ ECB เพียงแค่ 2 เปอร์เซ็นต์ แต่เมื่อพิจารณาถึงความปลอดภัยและการใช้งานจริงนั้นรูปแบบการประมวลผลแบบ CTR ดีกว่าแบบ ECB



ภาพที่ 5.1 เวลาในการประมวลผลและเวลาที่ใช้ในหน่วยความจำ

5.2 ข้อเสนอแนะ

เพื่อให้ค่าที่ได้เข้าใกล้ค่าในอุดมคติมากขึ้น การพัฒนาต่อไปควรมีการปรับปรุงวิธีการใช้งานหน่วยความจำและทำการทดลองกับหน่วยประมวลผลที่มีจำนวนแกนมากขึ้นเพื่อแบ่งส่วนข้อมูลในการจัดการข้อมูลและทดลองกับหน่วยประมวลผล GPU มาร่วมด้วยซึ่งในปัจจุบันมีการพัฒนาให้มีการทำงานประมวลที่เร็วขึ้นและยังสามารถประมวลผลการทำงานที่ซับซ้อนได้

รายการอ้างอิง

1. NIST, *Announcing the advanced encryption standard (AES), FIPS 197.* . November 2001.: National Institute of Standards and Technology Technical report
2. Institute, P., *2012 Global Encryption Trends Study.* Ponemon Institute Research Report, 2008.
3. Hasegawa, A. *Renesas' Multi-Core Technology [Online]*. 2013; Available from: http://www.renesas.com/products/mpumcu/multi_core/child/multicore.jsp. .
4. Chongstitvatana, P. *S2: A Hypothetical 32-bit Processor Version 3[Online]*. 2013; Available from: http://www.cp.eng.chula.ac.th/faculty/pjw/project/s2/s23.htm
5. Barnes, A.F., R.; Mettananda, K.; Ragel, R, *Improving the throughput of the AES algorithm with multicore processors*, in *Industrial and Information Systems (ICIIS), 2012 7th IEEE International Conference*. p. 1-6.
6. Manavski, S.A., *CUDA Compatible GPU as an Efficient Hardware Accelerator for AES Cryptography*, in *IEEE International Conference on Signal Processing and Communications, 2007. ICSPC 2007*. p. 65-68.
7. Lee, N.-P.T.M.L.S.H.S.-J., *Parallel Execution of AES-CTR Algorithm Using Extended Block Size*, in *Computational Science and Engineering (CSE), 2011 IEEE 14th International Conference* p. 191-198.
8. Gielata, A.R., P.; Wiatr, K., *AES hardware implementation in FPGA for algorithm acceleration purpose*, in *International Conference on Signals and Electronic Systems, 2008. ICSES '08*. p. 137-140.

ภาคผนวก

ภาคผนวก ก.

ชุดข้อมูลสำหรับการทดสอบ

ตารางกุญแจขนาด 128 บิต

กุญแจขนาด 128 บิต (16 ไบต์ต่อช่อง) ในรูปแบบข้อมูลฐาน 16 สำหรับการทดสอบการเข้ารหัสลับ

ตารางที่ ก.1 กุญแจขนาด 128 บิต สำหรับการทดสอบ

KeyList[16]															
0F	15	71	C9	47	D9	E8	59	0C	B7	AD	D6	AF	7F	67	98

ข้อมูลเข้ารหัส 256 ไบต์

ตารางที่ ก.2 ข้อมูลสำหรับการรหัสลับขนาด 256 ไบต์

Data[256]															
1	01	89	FE	76	23	AB	DC	54	45	CD	BA	32	67	EF	98	10
2	DC	12	87	27	7B	96	41	8A	2E	CB	CB	CC	08	97	46	71
3	6F	A8	7E	E5	DF	8B	D5	19	64	65	66	23	A6	4B	EB	F0
4	ED	6B	E7	CE	20	84	4C	4E	D0	06	C5	D9	C2	42	72	D5
5	38	62	BA	57	90	BB	C4	A5	24	53	92	7B	82	86	31	E4
6	30	80	39	DE	FF	32	C0	42	20	C8	E3	54	7F	B3	F7	E2
7	A6	11	27	55	C9	98	53	5C	9E	79	34	29	AF	FA	55	02
8	F7	0A	BA	58	B3	12	25	5B	7D	92	03	A8	78	26	0F	83
9	2F	E7	A8	C8	6C	C7	60	D7	1D	80	45	6A	29	97	06	B5
10	70	5D	6F	F2	CC	F2	BD	E9	27	3C	A7	66	9C	4C	04	9D
11	A6	FC	C4	01	23	E9	1C	14	2D	E9	9F	50	C7	11	98	10
12	87	84	C1	15	06	11	AE	91	F9	3A	2A	AF	F1	82	6A	5D
13	84	AB	05	4C	F9	FA	F5	6C	71	41	4A	F5	F7	63	28	44
14	04	B4	B4	5F	02	18	52	41	70	E5	77	A8	BC	F0	AF	E6
15	F4	9A	15	FB	B1	D3	66	B2	79	12	3E	B4	5F	3D	F0	1F
16	57	F9	6F	76	E6	8D	61	7A	2F	6E	72	46	74	F0	C4	85

ผลลัพธ์การเข้ารหัส

ตารางที่ ก.3 ข้อมูลที่ได้จากการเข้ารหัสลับขนาด 256 ไบต์

Cipher[256]																
1	FF	08	69	64	0B	53	34	14	84	BF	AB	8F	4A	7C	43	B9
2	BD	1C	47	98	C5	37	FE	C7	A0	BF	B2	F5	C3	06	58	02
3	8D	C4	49	8D	56	80	BE	16	2B	20	CE	F6	F0	B7	27	0E
4	3A	DD	DA	F5	35	DF	A0	3D	67	66	08	AE	73	8B	BC	57
5	2B	2D	A7	83	73	76	17	A6	04	A8	49	20	6B	7B	BD	F6
6	46	E3	29	A2	EE	90	89	74	36	5A	BA	53	36	FC	CE	92
7	95	58	2B	5B	20	48	4F	0E	33	EE	06	77	ED	49	FC	D2
8	6C	EA	DC	3E	D9	85	AF	45	81	A2	74	42	D6	9E	28	35
9	5C	14	D1	E1	2F	70	54	64	C8	02	39	59	95	5D	6B	8A
10	A1	84	1A	04	EF	A4	A6	44	9F	47	2F	9B	73	AD	CD	87
11	B2	C0	33	E5	30	A8	0B	5A	3C	2E	12	21	DE	B3	2F	9B
12	63	2F	3C	53	BC	94	90	30	12	A4	91	94	6D	5F	41	A7
13	5D	02	89	44	8C	C7	BD	6F	02	2D	6F	FB	C3	3E	16	F9
14	87	CD	1F	51	EC	37	E4	92	A8	F7	7D	B5	4A	5A	9C	E1
15	95	24	E1	8C	52	53	B3	FA	0E	33	1A	D8	6B	F7	BE	5A
16	B8	BF	DA	59	3D	1A	2F	3F	C5	8C	42	58	9A	C4	E2	75

ภาคผนวก ข.
ผลลัพธ์ของการทดลอง

ตารางที่ ข.1 ผลลัพธ์ CPU Time ของการเข้ารหัสโดยไม่รวมเวลา Stall

ข้อมูล (ไบต์)	1 Core	2 Core	3 Core	4 Core
16	15131	-	-	-
32	26938	15186	-	-
48	38745	26938	15186	-
64	50552	26993	26938	15186
80	62359	38745	26993	26938
96	74166	38800	26993	26993
112	85973	50552	38745	26993
128	97780	50607	38800	26993
144	109587	62359	38800	38745
160	121394	62414	50552	38800
176	133201	74166	50607	38800
192	145008	74221	50607	38800
208	156815	85973	62359	50552
224	168622	86028	62414	50607
240	180429	97780	62414	50607
256	192236	97835	74166	50607

ตารางที่ ข.2 ผลลัพธ์ CPU Time ของการเข้ารหัสโดยรวมเวลา Stall

ข้อมูล (ไบต์)	1 Core	2 Core	3 Core	4 Core
16	20833	-	-	-
32	37189	26917	-	-
48	53545	39974	38915	-
64	69901	48358	45664	49924
80	86257	58628	54096	59821
96	102613	69866	70452	69338
112	118969	77299	78682	74472
128	135325	91428	85669	90828
144	151681	96513	102025	100614
160	168037	112869	111768	110336
176	184393	118037	117103	115554
192	200749	134393	133459	131910
208	217105	139536	143580	141157
224	233461	155892	148758	151220
240	249817	161092	165114	156331
256	266173	177448	175088	172687

ตารางที่ ข.3 ผลลัพธ์ Stall time ในการเข้ารหัส (ผลรวม Stall ของทุกแกนการทำงาน)

ข้อมูล (ไบต์)	1 Core	2 Core	3 Core	4 Core
32	10346	19456	-	-
64	19444	35475	62383	81733
128	37640	67695	119257	150909
256	74000	131995	222702	289148

ตารางที่ ข.4 ผลลัพธ์ CPU Time ในการเข้ารหัสด้วยวิธีการประสานเวลา (Syncronization)

ข้อมูล (ไบต์)	AES-Sync	AES-Sync with Pipeline
16	23173	57573
32	41938	58233
48	60676	58893
64	79414	60134
80	98152	110356
96	116890	110927
112	135628	111587
128	154366	113090
144	173104	163215
160	191842	163875
176	210580	164535
192	229318	166038
208	248056	216163
224	266730	216823
240	285468	217483
256	304206	218986

ตารางที่ ข.5 ผลลัพธ์ Stall time ในการเข้ารหัสด้วยวิธีการประสานเวลา (Syncronization)

ข้อมูล (ไบต์)	AES-Sync	AES-Sync with Pipeline
16	13062	50534
32	24913	51455
48	36768	52376
64	48623	53910
80	60478	100758
96	72331	101622
112	84186	102543
128	96041	104162
144	107896	150956
160	119751	151877
176	131606	152798
192	143461	154417
208	155316	201211
224	167155	202132
240	179010	203053
256	190865	218986

ตารางที่ ข.6 ผลลัพธ์ CPU Time ในการเข้ารหัส AES รูปแบบ ECB และ CTR

ข้อมูล (ไบต์)	ECB	CTR
16	57573	57949
32	58233	58794
48	58893	59639
64	60134	61189
80	110356	111935
96	110927	112780
112	111587	113625
128	113090	115175
144	163215	165923
160	163875	166768
176	164535	167613
192	166038	169151
208	216163	219945
224	216823	220790
240	217483	221635
256	218986	223185

ตารางที่ ข.7 ผลลัพธ์ Stall time ในการเข้ารหัส AES รูปแบบ ECB และ CTR

ข้อมูล (ไบต์)	ECB	CTR
16	50534	51136
32	51455	52142
48	52376	53148
64	53910	54934
80	100758	102331
96	101622	103337
112	102543	104343
128	104162	106129
144	150956	153489
160	151877	154495
176	152798	155501
192	154417	157287
208	201211	204722
224	202132	205728
240	203053	206734
256	204672	208520

ตารางที่ ข.8 ผลลัพธ์ CPU time ในการเข้ารหัส AES ด้วยการประมวลผลด้วย Pipeline เมื่อหน่วยความจำแบบ 2 และ 3 สัญญาณนาฬิกา

ข้อมูล (ไบต์)	2 Stall	3 Stall
16	57573	70378
32	58233	71172
48	58893	71978
64	60134	73752
80	110356	135173
96	110927	135967
112	111587	136773
128	113090	138547
144	163215	199968
160	163875	200926
176	164535	201568
192	166038	203342
208	216163	264926
224	216823	265557
240	217483	266363
256	218986	268137

ตารางที่ ข.9 ผลลัพธ์ Stall time ในการเข้ารหัส AES ด้วยการประมวลผลด้วย Pipeline เมื่อหน่วยความจำแบบ 2 และ 3 สัญญาณนาฬิกา

ข้อมูล (ไบต์)	2 Stall	3 Stall
16	50534	80987
32	51455	82066
48	52376	83150
64	53910	85531
80	100758	161127
96	101622	162206
112	102543	163290
128	104162	165671
144	150956	241267
160	151877	242532
176	152798	243430
192	154417	245811
208	201211	321595
224	202132	322486
240	203053	323570
256	204672	325951

ภาคผนวก ค.
ตัวอย่างชุดรหัสข้อมูล

รายละเอียดของชุดรหัสข้อมูลที่ใช้ในการวิจัย อยู่ในภาษา RZ

ตารางที่ ค.1 ตัวอย่างชุดรหัสส่วนการขยายกุญแจ (KeyExpension)

```
keyExpansion()
   // W0 - W1 - W2 - W3
   i = 0
   while(i < 16)
     roundKey[i] = key[i]
     i = i + 1
   // W4
   rwordBuff = key[12]
   rwordArray[0] = key[13]
   rwordArray[1] = key[14]
   rwordArray[2] = key[15]
   rwordArray[3] = rwordBuff
    i = 0
   while(i < 4)
     rwordArray[i] = sbox[ rwordArray[i] ]
     i = i + 1
   roundKey[16] = rwordArray[0] ^ rcon[1]
   roundKey[17] = rwordArray[1] ^ 0
   roundKey[18] = rwordArray[2] ^ 0
   roundKey[19] = rwordArray[3] ^ 0
   roundKey[16] = roundKey[16] ^ roundKey[0]
   roundKey[17] = roundKey[17] ^ roundKey[1]
   roundKey[18] = roundKey[18] ^ roundKey[2]
   roundKey[19] = roundKey[19] ^ roundKey[3]
   // W5
   roundKey[20] = roundKey[16] ^ roundKey[4]
   roundKey[21] = roundKey[17] ^ roundKey[5]
   roundKey[22] = roundKey[18] ^ roundKey[6]
   roundKey[23] = roundKey[19] ^ roundKey[7]
   // W6
   roundKey[24] = roundKey[20] ^ roundKey[8]
   roundKey[25] = roundKey[21] ^ roundKey[9]
   roundKey[26] = roundKey[22] ^ roundKey[10]
   roundKey[27] = roundKey[23] ^ roundKey[11]
   // W7
   roundKey[28] = roundKey[24] ^ roundKey[12]
   roundKey[29] = roundKey[25] ^ roundKey[13]
   roundKey[30] = roundKey[26] ^ roundKey[14]
   roundKey[31] = roundKey[27] ^ roundKey[15]
  // W8 - W9 - W10 - W11
   rwordBuff = roundKey[28]
   rwordArray[0] = roundKey[29]
   rwordArray[1] = roundKey[30]
   rwordArray[2] = roundKey[31]
   rwordArray[3] = rwordBuff
   i = 0
   while(i < 4)
     rwordArray[i] = sbox[ rwordArray[i] ]
     i = i + 1
```

```
    roundKey[32] = rwordArray[0] ^ rcon[2]
    roundKey[33] = rwordArray[1] ^ 0
    roundKey[34] = rwordArray[2] ^ 0
    roundKey[35] = rwordArray[3] ^ 0
    i = 32
    while(i < 36)
      roundKey[i] = roundKey[i] ^ roundKey[i-16]
      i = i + 1
    i = 36
    while(i < 48)
      roundKey[i] = roundKey[i-4] ^ roundKey[i-16]
      i = i + 1
   // W12 - W13 - W14 - W15
    rwordBuff = roundKey[44]
    rwordArray[0] = roundKey[45]
    rwordArray[1] = roundKey[46]
    rwordArray[2] = roundKey[47]
    rwordArray[3] = rwordBuff
    i = 0
    while(i < 4)
      rwordArray[i] = sbox[ rwordArray[i] ]
      i = i + 1
    roundKey[48] = rwordArray[0] ^ rcon[3]
    roundKey[49] = rwordArray[1] ^ 0
    roundKey[50] = rwordArray[2] ^ 0
    roundKey[51] = rwordArray[3] ^ 0
    i = 48
    while(i < 52)
      roundKey[i] = roundKey[i] ^ roundKey[i-16]
      i = i + 1
    i = 52
    while(i < 64)
      roundKey[i] = roundKey[i-4] ^ roundKey[i-16]
      i = i + 1
// W16 - W17 - W18 - W19
    rwordBuff = roundKey[60]
    rwordArray[0] = roundKey[61]
    rwordArray[1] = roundKey[62]
    rwordArray[2] = roundKey[63]
    rwordArray[3] = rwordBuff
    i = 0
    while(i < 4)
      rwordArray[i] = sbox[ rwordArray[i] ]
      i = i + 1
    roundKey[64] = rwordArray[0] ^ rcon[4]
    roundKey[65] = rwordArray[1] ^ 0
    roundKey[66] = rwordArray[2] ^ 0
    roundKey[67] = rwordArray[3] ^ 0
    i = 64
    while(i < 68)
      roundKey[i] = roundKey[i] ^ roundKey[i-16]
      i = i + 1
    i = 68
    while(i < 80)
      roundKey[i] = roundKey[i-4] ^ roundKey[i-16]
      i = i + 1
// W20 - W21 - W22 - W23
    rwordBuff = roundKey[76]
```

```
    rwordArray[0] = roundKey[77]
    rwordArray[1] = roundKey[78]
    rwordArray[2] = roundKey[79]
    rwordArray[3] = rwordBuff
    i = 0
    while(i < 4)
      rwordArray[i] = sbox[ rwordArray[i] ]
      i = i + 1
    roundKey[80] = rwordArray[0] ^ rcon[5]
    roundKey[81] = rwordArray[1] ^ 0
    roundKey[82] = rwordArray[2] ^ 0
    roundKey[83] = rwordArray[3] ^ 0
    i = 80
    while(i < 84)
      roundKey[i] = roundKey[i] ^ roundKey[i-16]
      i = i + 1

    i = 84
    while(i < 96)
      roundKey[i] = roundKey[i-4] ^ roundKey[i-16]
      i = i + 1
// W24 - W25 - W26 - W27
    rwordBuff = roundKey[92]
    rwordArray[0] = roundKey[93]
    rwordArray[1] = roundKey[94]
    rwordArray[2] = roundKey[95]
    rwordArray[3] = rwordBuff
    i = 0
    while(i < 4)
      rwordArray[i] = sbox[ rwordArray[i] ]
      i = i + 1
    roundKey[96] = rwordArray[0] ^ rcon[6]
    roundKey[97] = rwordArray[1] ^ 0
    roundKey[98] = rwordArray[2] ^ 0
    roundKey[99] = rwordArray[3] ^ 0
    i = 96
    while(i < 100)
      roundKey[i] = roundKey[i] ^ roundKey[i-16]
      i = i + 1
    i = 100
    while(i < 112)
      roundKey[i] = roundKey[i-4] ^ roundKey[i-16]
      i = i + 1
// W28 - W29 - W30 - W31
    rwordBuff = roundKey[108]
    rwordArray[0] = roundKey[109]
    rwordArray[1] = roundKey[110]
    rwordArray[2] = roundKey[111]
    rwordArray[3] = rwordBuff
    i = 0
    while(i < 4)
      rwordArray[i] = sbox[ rwordArray[i] ]
      i = i + 1
    roundKey[112] = rwordArray[0] ^ rcon[7]
    roundKey[113] = rwordArray[1] ^ 0
    roundKey[114] = rwordArray[2] ^ 0
    roundKey[115] = rwordArray[3] ^ 0
    i = 112
```

```
    while(i < 116)
      roundKey[i] = roundKey[i] ^ roundKey[i-16]
      i = i + 1
    i = 116
    while(i < 128)
      roundKey[i] = roundKey[i-4] ^ roundKey[i-16]
      i = i + 1
// W32 - W33 - W34 - W35
    rwordBuff = roundKey[124]
    rwordArray[0] = roundKey[125]
    rwordArray[1] = roundKey[126]
    rwordArray[2] = roundKey[127]
    rwordArray[3] = rwordBuff
    i = 0
    while(i < 4)
      rwordArray[i] = sbox[ rwordArray[i] ]
      i = i + 1
    roundKey[128] = rwordArray[0] ^ rcon[8]
    roundKey[129] = rwordArray[1] ^ 0
    roundKey[130] = rwordArray[2] ^ 0
    roundKey[131] = rwordArray[3] ^ 0
    i = 128
    while(i < 132)
      roundKey[i] = roundKey[i] ^ roundKey[i-16]
      i = i + 1
    i = 132
    while(i < 144)
      roundKey[i] = roundKey[i-4] ^ roundKey[i-16]
      i = i + 1
// W36 - W37 - W38 - W39
    rwordBuff = roundKey[140]
    rwordArray[0] = roundKey[141]
    rwordArray[1] = roundKey[142]
    rwordArray[2] = roundKey[143]
    rwordArray[3] = rwordBuff
    i = 0
    while(i < 4)
      rwordArray[i] = sbox[ rwordArray[i] ]
      i = i + 1
    roundKey[144] = rwordArray[0] ^ rcon[9]
    roundKey[145] = rwordArray[1] ^ 0
    roundKey[146] = rwordArray[2] ^ 0
    roundKey[147] = rwordArray[3] ^ 0
    i = 144
    while(i < 148)
      roundKey[i] = roundKey[i] ^ roundKey[i-16]
      i = i + 1
    i = 148
    while(i < 160)
      roundKey[i] = roundKey[i-4] ^ roundKey[i-16]
      i = i + 1
// W40 - W41 - W42 - W43
    rwordBuff = roundKey[156]
    rwordArray[0] = roundKey[157]
    rwordArray[1] = roundKey[158]
    rwordArray[2] = roundKey[159]
    rwordArray[3] = rwordBuff
```

```
   i = 0
   while(i < 4)
     rwordArray[i] = sbox[ rwordArray[i] ]
     i = i + 1
   roundKey[160] = rwordArray[0] ^ rcon[10]
   roundKey[161] = rwordArray[1] ^ 0
   roundKey[162] = rwordArray[2] ^ 0
   roundKey[163] = rwordArray[3] ^ 0
   i = 160
   while(i < 164)
     roundKey[i] = roundKey[i] ^ roundKey[i-16]
     i = i + 1
   i = 164
   while(i < 176)
     roundKey[i] = roundKey[i-4] ^ roundKey[i-16]
     i = i + 1
```

ตารางที่ ค.2 ตัวอย่างชุดรหัสส่วนการแทนที่ข้อมูล (Subbyte)

```
subByte_c1()
   i = 0
   while(i < 16)
     cipher_c1[i] = sbox[ cipher_c1[i] ]
     i = i + 1
```

ตารางที่ ค.3 ตัวอย่างชุดรหัสส่วนการเลื่อนแถว (ShiftRow)

```
ShiftRows_c1()
   i = 0
   // row 0 shift 0
   // row 1 shift 1
   shiftBuff = cipher_c1[4]
   cipher_c1[4] = cipher_c1[5]
   cipher_c1[5] = cipher_c1[6]
   cipher_c1[6] = cipher_c1[7]
   cipher_c1[7] = shiftBuff
   // row 3 shift 2
   shiftBuff = cipher_c1[8]
   cipher_c1[8] = cipher_c1[9]
   cipher_c1[9] = cipher_c1[10]
   cipher_c1[10] = cipher_c1[11]
   cipher_c1[11] = shiftBuff
   shiftBuff = cipher_c1[8]
   cipher_c1[8] = cipher_c1[9]
   cipher_c1[9] = cipher_c1[10]
   cipher_c1[10] = cipher_c1[11]
   cipher_c1[11] = shiftBuff
   // row 3 shift 3
   shiftBuff = cipher_c1[12]
   cipher_c1[12] = cipher_c1[13]
   cipher_c1[13] = cipher_c1[14]
   cipher_c1[14] = cipher_c1[15]
   cipher_c1[15] = shiftBuff
   shiftBuff = cipher_c1[12]
   cipher_c1[12] = cipher_c1[13]
```

```
   cipher_c1[13] = cipher_c1[14]
   cipher_c1[14] = cipher_c1[15]
   cipher_c1[15] = shiftBuff
   shiftBuff = cipher_c1[12]
   cipher_c1[12] = cipher_c1[13]
   cipher_c1[13] = cipher_c1[14]
   cipher_c1[14] = cipher_c1[15]
   cipher_c1[15] = shiftBuff
   cipher_c1[16] = 0
```

ตารางที่ ค.4 ตัวอย่างชุดรหัสส่วนการผสมคอลัมภ์ (MixColumn)

```
MixColumns_c1()
   i = 0
   // r0
   cipherBuff[0] =  ( ( cipher_c1[0] << 1 ) % 256 )
   if ( cipher_c1[0] > 128 )
      cipherBuff[0] = cipherBuff[0] ^ 27
   cipherBuff[0] = cipherBuff[0] ^ ( ( cipher_c1[4] << 1 ) % 256 )
   if ( cipher_c1[4] > 128 )
      cipherBuff[0] = cipherBuff[0] ^ 27
   cipherBuff[0] = cipherBuff[0] ^ cipher_c1[4]
   cipherBuff[0] = cipherBuff[0] ^ cipher_c1[8]  ^ cipher_c1[12]
   // r4
   cipherBuff[4] =  ( ( cipher_c1[4] << 1 ) % 256 )
   if ( cipher_c1[4] > 128 )
      cipherBuff[4] = cipherBuff[4] ^ 27
   cipherBuff[4] = cipherBuff[4] ^ ( ( cipher_c1[8] << 1 ) % 256 )
   if ( cipher_c1[8] > 128 )
      cipherBuff[4] = cipherBuff[4] ^ 27
   cipherBuff[4] = cipherBuff[4] ^ cipher_c1[8]
   cipherBuff[4] = cipherBuff[4] ^ cipher_c1[0]  ^ cipher_c1[12]
   // r8
   cipherBuff[8] =  ( ( cipher_c1[8] << 1 ) % 256 )
   if ( cipher_c1[8] > 128 )
      cipherBuff[8] = cipherBuff[8] ^ 27
   cipherBuff[8] = cipherBuff[8] ^ ( ( cipher_c1[12] << 1 ) % 256 )
   if ( cipher_c1[12] > 128 )
      cipherBuff[8] = cipherBuff[8] ^ 27
   cipherBuff[8] = cipherBuff[8] ^ cipher_c1[12]
   cipherBuff[8] = cipherBuff[8] ^ cipher_c1[0]  ^ cipher_c1[4]
   // r12
   cipherBuff[12] =  ( ( cipher_c1[12] << 1 ) % 256 )
   if ( cipher_c1[12] > 128 )
      cipherBuff[12] = cipherBuff[12] ^ 27
   cipherBuff[12] = cipherBuff[12] ^ ( ( cipher_c1[0] << 1 ) % 256 )
   if ( cipher_c1[0] > 128 )
      cipherBuff[12] = cipherBuff[12] ^ 27
   cipherBuff[12] = cipherBuff[12] ^ cipher_c1[0]
   cipherBuff[12] = cipherBuff[12] ^ cipher_c1[4]  ^ cipher_c1[8]
   // r1
   cipherBuff[1] =  ( ( cipher_c1[1] << 1 ) % 256 )
   if ( cipher_c1[1] > 128 )
      cipherBuff[1] = cipherBuff[1] ^ 27
   cipherBuff[1] = cipherBuff[1] ^ ( ( cipher_c1[5] << 1 ) % 256 )
   if ( cipher_c1[5] > 128 )
      cipherBuff[1] = cipherBuff[1] ^ 27
```

```
cipherBuff[1] = cipherBuff[1] ^ cipher_c1[5]
cipherBuff[1] = cipherBuff[1] ^ cipher_c1[9]  ^ cipher_c1[13]
// r5
cipherBuff[5] =  ( ( cipher_c1[5] << 1 ) % 256 )
if ( cipher_c1[5] > 128 )
   cipherBuff[5] = cipherBuff[5] ^ 27
cipherBuff[5] = cipherBuff[5] ^ ( ( cipher_c1[9] << 1 ) % 256 )
if ( cipher_c1[9] > 128 )
   cipherBuff[5] = cipherBuff[5] ^ 27
cipherBuff[5] = cipherBuff[5] ^ cipher_c1[9]
cipherBuff[5] = cipherBuff[5] ^ cipher_c1[1]  ^ cipher_c1[13]
// r9
cipherBuff[9] =  ( ( cipher_c1[9] << 1 ) % 256 )
if ( cipher_c1[9] > 128 )
   cipherBuff[9] = cipherBuff[9] ^ 27
cipherBuff[9] = cipherBuff[9] ^ ( ( cipher_c1[13] << 1 ) % 256 )
if ( cipher_c1[13] > 128 )
   cipherBuff[9] = cipherBuff[9] ^ 27
cipherBuff[9] = cipherBuff[9] ^ cipher_c1[13]
cipherBuff[9] = cipherBuff[9] ^ cipher_c1[1]  ^ cipher_c1[5]
// r13
cipherBuff[13] =  ( ( cipher_c1[13] << 1 ) % 256 )
if ( cipher_c1[13] > 128 )
   cipherBuff[13] = cipherBuff[13] ^ 27
cipherBuff[13] = cipherBuff[13] ^ ( ( cipher_c1[1] << 1 ) % 256 )
if ( cipher_c1[1] > 128 )
   cipherBuff[13] = cipherBuff[13] ^ 27
cipherBuff[13] = cipherBuff[13] ^ cipher_c1[1]
cipherBuff[13] = cipherBuff[13] ^ cipher_c1[5]  ^ cipher_c1[9]
// r2
cipherBuff[2] =  ( ( cipher_c1[2] << 1 ) % 256 )
if ( cipher_c1[2] > 128 )
   cipherBuff[2] = cipherBuff[2] ^ 27
cipherBuff[2] = cipherBuff[2] ^ ( ( cipher_c1[6] << 1 ) % 256 )
if ( cipher_c1[6] > 128 )
   cipherBuff[2] = cipherBuff[2] ^ 27
cipherBuff[2] = cipherBuff[2] ^ cipher_c1[6]
cipherBuff[2] = cipherBuff[2] ^ cipher_c1[10]  ^ cipher_c1[14]
 // r6
cipherBuff[6] =  ( ( cipher_c1[6] << 1 ) % 256 )
if ( cipher_c1[6] > 128 )
   cipherBuff[6] = cipherBuff[6] ^ 27
cipherBuff[6] = cipherBuff[6] ^ ( ( cipher_c1[10] << 1 ) % 256 )
if ( cipher_c1[10] > 128 )
   cipherBuff[6] = cipherBuff[6] ^ 27
cipherBuff[6] = cipherBuff[6] ^ cipher_c1[10]
cipherBuff[6] = cipherBuff[6] ^ cipher_c1[2]  ^ cipher_c1[14]
// r10
cipherBuff[10] =  ( ( cipher_c1[10] << 1 ) % 256 )
if ( cipher_c1[10] > 128 )
   cipherBuff[10] = cipherBuff[10] ^ 27
cipherBuff[10] = cipherBuff[10] ^ ( ( cipher_c1[14] << 1 ) % 256 )
if ( cipher_c1[14] > 128 )
   cipherBuff[10] = cipherBuff[10] ^ 27
cipherBuff[10] = cipherBuff[10] ^ cipher_c1[14]
cipherBuff[10] = cipherBuff[10] ^ cipher_c1[2]  ^ cipher_c1[6]
// r14
cipherBuff[14] =  ( ( cipher_c1[14] << 1 ) % 256 )
```

```
   if ( cipher_c1[14] > 128 )
      cipherBuff[14] = cipherBuff[14] ^ 27
   cipherBuff[14] = cipherBuff[14] ^ ( ( cipher_c1[2] << 1 ) % 256 )
   if ( cipher_c1[2] > 128 )
      cipherBuff[14] = cipherBuff[14] ^ 27
   cipherBuff[14] = cipherBuff[14] ^ cipher_c1[2]
   cipherBuff[14] = cipherBuff[14] ^ cipher_c1[6]  ^ cipher_c1[10]
   // r3
   cipherBuff[3] =  ( ( cipher_c1[3] << 1 ) % 256 )
   if ( cipher_c1[3] > 128 )
      cipherBuff[3] = cipherBuff[3] ^ 27
   cipherBuff[3] = cipherBuff[3] ^ ( ( cipher_c1[7] << 1 ) % 256 )
   if ( cipher_c1[7] > 128 )
      cipherBuff[3] = cipherBuff[3] ^ 27
   cipherBuff[3] = cipherBuff[3] ^ cipher_c1[7]
   cipherBuff[3] = cipherBuff[3] ^ cipher_c1[11]  ^ cipher_c1[15]
    // r7
   cipherBuff[7] =  ( ( cipher_c1[7] << 1 ) % 256 )
   if ( cipher_c1[7] > 128 )
      cipherBuff[7] = cipherBuff[7] ^ 27
   cipherBuff[7] = cipherBuff[7] ^ ( ( cipher_c1[11] << 1 ) % 256 )
   if ( cipher_c1[11] > 128 )
      cipherBuff[7] = cipherBuff[7] ^ 27
   cipherBuff[7] = cipherBuff[7] ^ cipher_c1[11]
   cipherBuff[7] = cipherBuff[7] ^ cipher_c1[3]  ^ cipher_c1[15]
   // r11
   cipherBuff[11] =  ( ( cipher_c1[11] << 1 ) % 256 )
   if ( cipher_c1[11] > 128 )
      cipherBuff[11] = cipherBuff[11] ^ 27
   cipherBuff[11] = cipherBuff[11] ^ ( ( cipher_c1[15] << 1 ) % 256 )
   if ( cipher_c1[15] > 128 )
      cipherBuff[11] = cipherBuff[11] ^ 27
   cipherBuff[11] = cipherBuff[11] ^ cipher_c1[15]
   cipherBuff[11] = cipherBuff[11] ^ cipher_c1[3]  ^ cipher_c1[7]
   // r15
   cipherBuff[15] =  ( ( cipher_c1[15] << 1 ) % 256 )
   if ( cipher_c1[15] > 128 )
      cipherBuff[15] = cipherBuff[15] ^ 27
   cipherBuff[15] = cipherBuff[15] ^ ( ( cipher_c1[3] << 1 ) % 256 )
   if ( cipher_c1[3] > 128 )
      cipherBuff[15] = cipherBuff[15] ^ 27
   cipherBuff[15] = cipherBuff[15] ^ cipher_c1[3]
   cipherBuff[15] = cipherBuff[15] ^ cipher_c1[7]  ^ cipher_c1[11]
   i = 0
   while(i < 16)
     cipher_c1[i] = cipherBuff[i]
     i = i + 1
```

ตารางที่ ค.5 ตัวอย่างชุดรหัสส่วนการผสมกุญแจ (AddRoundKey)

```
AddRoundKey_c1(a)
   i = 0
   j = a * 16
   while(i < 4)
     cipher_c1[i] = cipher_c1[i] ^ roundKey[j]
     i = i + 1
     j = j + 4
```

```
i = 4
j = ( a * 16 ) + 1
while(i < 8)
  cipher_c1[i] = cipher_c1[i] ^ roundKey[j]
  i = i + 1
  j = j + 4
i = 8
j = ( a * 16 ) + 2
while(i < 12)
  cipher_c1[i] = cipher_c1[i] ^ roundKey[j]
  i = i + 1
  j = j + 4
i = 12
j = ( a * 16 ) + 3
while(i < 16)
  cipher_c1[i] = cipher_c1[i] ^ roundKey[j]
  i = i + 1
  j = j + 4
```

ประวัติผู้เขียนวิทยานิพนธ์

นายศุภชัย ทองสุข เกิดเมื่อวันที่ 4 มีนาคม 2532 ที่อำเภอเมือง จังหวัดอุบลราชธานี สำเร็จการศึกษาปริญญาวิศวกรรมศาสตร์บัณฑิต สาขาวิชาวิศวกรรมคอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ มหาวิทยาลัยขอนแก่น เมื่อปีการศึกษา 2553 และเข้าศึกษาต่อในหลักสูตรวิทยาศาสตร์มหาบัณฑิต สาขาวิศวกรรมซอฟต์แวร์ ภาควิชาวิศวกรรมคอมพิวเตอร์ จุฬาลงกรณ์มหาวิทยาลัย ในปีการศึกษา 2555

ผู้เขียนวิทยานิพนธ์ได้ตีพิมพ์ผลงานวิชาการในรายงานการประชุม ECTI – CARD 2014 ชื่อ “An implementation of AES algorithm on multicore processor for high throughput” ณ โรงแรมเชียงใหม่แกรนด์วิว ประเทศไทย เมื่อเดือนพฤษภาคม พ.ศ. 2557