รายการอ้างอิง

[1] Akihiro Takamura, Masashi Kuwako, Masashi Imai, Taro Fujii, Motokazu Ozawa, Izumi Fukasaku, Yoichiro Ueno, Takashi Nanya. TITAC-2: A 32-bit Asynchronous Microprocessor based on Scalable-Delay-Insensitive Model. Proc. ICCD'97, pp288-294. October. 1997.

[2] Bob Zeidman. Verilog Designer's Library. Prentice-Hall PTR. 1999.

[3] David A. Patterson, John L. Hennessy. Computer Organization & Design: The Hardware/Software Interface. Morgan Kaufmann Publishers. 1994.

[4] John F. Wakerly. Digital Design Principles and Practices. 2nd Edition. Prentice-Hall. 1994.

[5] Open Verilog International. Verilog Hardware Description Language Reference Manual (LRM) Version 1.0. November. 1991.

[6] Scott Hauck. Asynchronous Design Mothodologies: An Overview. IEEE Vol.83, No.1, pp69-93. January. 1995.

[7] Sunggu Lee. Design of Computers and Other Complex Digital Devices. Prentice-Hall. 2000.

[8] Takashi Nanya. Asynchronous VLSI System Design. Research Center for Advanced Science and Technology, University of Tokyo. 1998.

[9] Takashi Nanya, Akihiro Takamura, Masashi Kuwako, Masashi Imai, Motokazu Ozawa, Metehan Ozcan, Rafael Morizawa, Hiroshi Nakamura. Scalable-Delay-Insensitive Design: A high-performance approach to dependable asynchronous systems. Research Center for Advanced Science and Technology, University of Tokyo. 1998.

[10] Takashi Nanya, Akihiro Takamura, Masashi Kuwako, Masashi Imai, Taro Fujii, Motokazu Ozawa, Metehan Ozcan, Izumi Fukasaku, Yoichiro Ueno, Fuyuki Okamoto, Hiroki Fujimoto, Osamu Fujita, Masakazu Yamashina, Masao Fukuma. TITAC-2: A 32-bit Scalable-Delay-Insensitive Microprocessor. Research Center for Advanced Science and Technology, University of Tokyo. Department of Computer Science, Tokyo Institute of Technology. Microelectronics Research Laboratories, NEC Corporation. 1998.

[11] Takashi Nanya, Yoichiro Ueno, Hiroto Kagotani, Masashi Kuwako, Akihiro Takamura. TITAC: Design of a Quasi-Delay-Insensitive Microprocessor. IEEE Design & Test of Computers, pp50-63. 1994.

[12] Thomas C. Bartee. Computer Architecture and Logic Design. McGraw-Hill, Inc. 1985.

[13] V. Carl Hamacher, Zvonko G. Vranesic, Safwat G. Zaky. Computer Organization. Third Edition. McGraw-Hill. 1990.

[14] William Stallings. Computer Organization and Architecture: Principles of Structure and Function. Macmillan Publishing Company. 1987.

[15] รัชดา นุตจรัส. การออกแบบวงจรตอบรับที่ไร้อุปกรณ์ชนิดซีสำหรับวงจรเชิงผสมแบบอสมวารที่ไม่ไวต่อความหน่วงชนิดปรับมาตราส่วนได้. วิทยานิพนธ์ปริญญามหาบัณฑิต ภาควิชาวิศวกรรมคอมพิวเตอร์ บัณฑิตวิทยาลัย จุฬาลงกรณ์มหาวิทยาลัย. 2542.

ภาคผนวก

ภาคผนวก ก

Instruction Set

Memory reference instructions			**Branch instructions**			**Miscellaneous instructions**	
ADD	mem	Add	JMP	adr	Jump	NOP	No operation
SUB	mem	Subtract	JSR	adr	Jump to subroutine	HLT	Halt
AND	mem	And	JCS	adr	Jump if carry set	INP	Input
OR	mem	Or	JCC	adr	Jump if carry clear	OUT	Output
XOR	mem	Exclusive or	JEQ	adr	Jump if equal	STC	Set carry
CMP	mem	Compare	JNE	adr	Jump if not equal	CLC	Clear carry
LDA	mem	Load accumulator				ROR	Rotate right
STA	mem	Store accumulator	**Addressing modes**			ROL	Rotate left
TST	mem	Test	adr	n,pc	Program counter-	LSR	Logical shift right
LDS	mem	Load stack pointer			relative	LSL	Logical shift left
STS	mem	Store stack pointer		(n)	Indirect	INA	Increment accumulator
ADS	mem	Add stack pointer		n	Direct	DCA	Decrement accumulator
SBS	mem	Subtract stack pointer				INS	Increment stack pointer
						DCS	Decrement stack pointer
Addressing modes						RTS	Return from subroutine
mem	#n	Immediate				PUL	Pull accumulator
	n,sp	Stack pointer-relative				PUS	Push accumulator
	(n)	Indirect					
	n	Direct					

ภาคผนวก ข

สรุปการทำงานของชุดคำสั่ง

คำสั่ง	การทำงาน
ADD	A + B
SUB	A – B
AND	A AND B
OR	A OR B
XOR	A XOR B
CMP	A - B แต่ไม่ส่งผลลัพธ์ใดๆ จะมีผลต่อ Flag เท่านั้น สามารถใช้ตรวจสอบได้ว่า A = B ถ้า ZF=1
LDA	นำค่าจาก RAM มาเก็บไว้ในรีจิสเตอร์ Acc
STA	นำค่าจากรีจิสเตอร์ Acc ไปเก็บไว้ใน RAM
LDS	นำค่าจาก RAM มาเก็บไว้ในรีจิสเตอร์ SP
STS	นำค่าจากรีจิสเตอร์ SP ไปเก็บไว้ใน RAM
ADS	A + B แต่ใช้รีจิสเตอร์ SP แทนรีจิสเตอร์ Acc
SBS	A - B แต่ใช้รีจิสเตอร์ SP แทนรีจิสเตอร์ Acc
JMP	Jump ไปทำงานต่อยังตำแหน่งที่กำหนด
JSR	Push ค่าในรีจิสเตอร์ PC ลง Stack แล้ว Jump ไปทำงานที่โปรแกรมย่อย
JCS	Jump ไปทำงานยังตำแหน่งที่กำหนด ถ้า CF = 1
JCC	Jump ไปทำงานยังตำแหน่งที่กำหนด ถ้า CF = 0
JEQ	Jump ไปทำงานยังตำแหน่งที่กำหนด ถ้า ZF = 1
JNE	Jump ไปทำงานยังตำแหน่งที่กำหนด ถ้า ZF = 0
NOP	ไม่มีการดำเนินการใดๆ
HLT	คำสั่งหยุดการทำงาน
INP	รับค่าอินพุตจากพอร์ตเข้ามาเก็บไว้ในรีจิสเตอร์ Acc
OUT	ส่งค่าเอาต์พุตจากรีจิสเตอร์ Acc ไปยังพอร์ต
STC	CF จะถูกเซ็ตเป็น 1 จากส่วนควบคุม
CLC	CF จะถูกเคลียร์เป็น 0 จากส่วนควบคุม
ROR	หมุนบิตไปทางขวา แล้ว CF จะเก็บบิตนัยสำคัญต่ำสุดของผลลัพธ์ไว้
ROL	หมุนบิตไปทางซ้าย แล้ว CF จะเก็บบิตนัยสำคัญต่ำสุดของผลลัพธ์ไว้
LSR	เลื่อนบิตไปทางขวา แล้ว CF จะเก็บบิตนัยสำคัญต่ำสุดของผลลัพธ์ไว้
LSL	เลื่อนบิตไปทางซ้าย แล้ว CF จะเก็บบิตนัยสำคัญต่ำสุดของผลลัพธ์ไว้

คำสั่ง	การทำงาน
INA	เพิ่มค่าในรีจิสเตอร์ Acc อีก 1
DCA	ลดค่าในรีจิสเตอร์ Acc ลง 1
INS	เพิ่มค่าในรีจิสเตอร์ SP อีก 1
DCS	ลดค่าในรีจิสเตอร์ SP ลง 1
RTS	Pop ค่าจาก Stack ออกมาเก็บไว้ที่รีจิสเตอร์ PC แล้วทำงานต่อไป
PUL	Pull ค่าจาก Stack มาเก็บไว้ที่รีจิสเตอร์ Acc
PUS	Push ค่าจากรีจิสเตอร์ Acc ไปเก็บไว้ที่ Stack

ภาคผนวก ค

Instruction Opcode

Hex Code	Binary Code	Number of Byte	Mnemonic	Addressing Mode
00	0000 0000	1	HLT	-
01	0000 0001	1	NOP	-
02	0000 0010	1	INA	-
04	0000 0100	1	CLC	-
05	0000 0101	1	STC	-
06	0000 0110	1	DCA	-
0A	0000 1010	1	INP	-
0B	0000 1011	1	OUT	-
10	0001 0000	1	RTS	-
11	0001 0001	1	PUL	-
12	0001 0010	1	INS	-
13	0001 0011	1	PUS	-
16	0001 0110	1	DCS	-
20	0010 0000	1	LSR	-
21	0010 0001	1	LSL	-
22	0010 0010	1	ROR	-
23	0010 0011	1	ROL	-
40	0100 0000	2	JMP	Direct
41	0100 0001	2	JSR	Direct
42	0100 0010	2	JNE	Direct
43	0100 0011	2	JEQ	Direct
44	0100 0100	2	JCC	Direct
45	0100 0101	2	JCS	Direct
50	0101 0000	2	JMP	Indirect
51	0101 0001	2	JSR	Indirect
52	0101 0010	2	JNE	Indirect
53	0101 0011	2	JEQ	Indirect
54	0101 0100	2	JCC	Indirect
55	0101 0101	2	JCS	Indirect

Hex Code	Binary Code	Number of Byte	Mnemonic	Addressing Mode
60	0110 0000	2	JMP	PC-Relative
61	0110 0001	2	JSR	PC-Relative
62	0110 0010	2	JNE	PC-Relative
63	0110 0011	2	JEQ	PC-Relative
64	0110 0100	2	JCC	PC-Relative
65	0110 0101	2	JCS	PC-Relative
80	1000 0000	2	AND	Direct
81	1000 0001	2	OR	Direct
82	1000 0010	2	ADD	Direct
83	1000 0011	2	XOR	Direct
86	1000 0110	2	SUB	Direct
8A	1000 1010	2	ADS	Direct
8E	1000 1110	2	SBS	Direct
8F	1000 1111	2	CMP	Direct
90	1001 0000	2	AND	Indirect
91	1001 0001	2	OR	Indirect
92	1001 0010	2	ADD	Indirect
93	1001 0011	2	XOR	Indirect
96	1001 0110	2	SUB	Indirect
9A	1001 1010	2	ADS	Indirect
9E	1001 1110	2	SBS	Indirect
9F	1001 1111	2	CMP	Indirect
A0	1010 0000	2	AND	SP-Relative
A1	1010 0001	2	OR	SP-Relative
A2	1010 0010	2	ADD	SP-Relative
A3	1010 0011	2	XOR	SP-Relative
A6	1010 0110	2	SUB	SP-Relative
AA	1010 1010	2	ADS	SP-Relative
AE	1010 1110	2	SBS	SP-Relative

Hex Code	Binary Code	Number of Byte	Mnemonic	Addressing Mode
AF	1010 1111	2	CMP	SP-Relative
B0	1011 0000	2	AND	Immediate
B1	1011 0001	2	OR	Immediate
B2	1011 0010	2	ADD	Immediate
B3	1011 0011	2	XOR	Immediate
B6	1011 0110	2	SUB	Immediate
BA	1011 1010	2	ADS	Immediate
BE	1011 1110	2	SBS	Immediate
BF	1011 1111	2	CMP	Immediate
88	1000 1000	2	LDA	Direct
89	1000 1001	2	LDS	Direct
C8	1100 1000	2	STA	Direct
C9	1100 1001	2	STS	Direct
98	1001 1000	2	LDA	Indirect
99	1001 1001	2	LDS	Indirect
D8	1101 1000	2	STA	Indirect
D9	1101 1001	2	STS	Indirect
A8	1010 1000	2	LDA	SP-Relative
A9	1010 1001	2	LDS	SP-Relative
E8	1110 1000	2	STA	SP-Relative
E9	1110 1001	2	STS	SP-Relative
B8	1011 1000	2	LDA	Immediate
B9	1011 1001	2	LDS	Immediate
F8	1111 1000	2	STA	Immediate
F9	1111 1001	2	STS	Immediate

*Opcode นอกเหนือจากที่กำหนดนี้ไม่ได้ใช้งาน

ภาคผนวก ง

สายสัญญาณควบคุมและตอบรับ

สายสัญญาณควบคุมมีทั้งสิ้น 42 เส้น ส่งจากส่วนควบคุมไปยังส่วนเส้นทางข้อมูลเพื่อกำหนดการทำงาน

เส้นที่	ชื่อสัญญาณ	หน้าที่การทำงาน
0	LSR	คำสั่งเลื่อนบิตไปทางขวา
1	LSL	คำสั่งเลื่อนบิตไปทางซ้าย
2	ROR	คำสั่งหมุนบิตไปทางขวา
3	ROL	คำสั่งหมุนบิตไปทางซ้าย
4	AND	คำสั่ง AND
5	OR	คำสั่ง OR
6	ADDSUB	คำสั่งให้ทำบวกหรือลบ
7	XOR	คำสั่ง XOR
8	ClearB	สัญญาณที่ทำให้ค่าอินพุตของ ALU เป็น 0 ทางด้านที่รีจิสเตอร์ MI จะเข้ามา
9	Cin	ตัวทดที่เป็นอินพุตของ ALU
10	Cin'	นิเสธสำหรับรหัสรางคู่ของตัวทด
11	SUB	สัญญาณที่กำหนดให้ทำคำสั่งลบ
12	SUB'	นิเสธสำหรับรหัสรางคู่ของคำสั่งลบ
13	CF	Carry Flag
14	CF'	นิเสธสำหรับรหัสรางคู่ของ Carry Flag
15	ram_oe	สัญญาณที่กำหนดให้เป็นการอ่านข้อมูลจาก RAM
16	REQmar_rom	สัญญาณที่กำหนดให้มีการส่งข้อมูลจากรีจิสเตอร์ MAR ไปยัง ROM
17	REQmar_ram	สัญญาณที่กำหนดให้มีการส่งข้อมูลจากรีจิสเตอร์ MAR ไปยัง RAM
18	REQmi_ram	สัญญาณที่กำหนดให้มีการส่งข้อมูลจากรีจิสเตอร์ MI ไปยัง RAM
19	REQmi_mar	สัญญาณที่กำหนดให้มีการส่งข้อมูลจากรีจิสเตอร์ MI ไปยังรีจิสเตอร์ MAR
20	REQsp_mar	สัญญาณที่กำหนดให้มีการส่งข้อมูลจากรีจิสเตอร์ SP ไปยังรีจิสเตอร์ MAR
21	REQpc_mar	สัญญาณที่กำหนดให้มีการส่งข้อมูลจากรีจิสเตอร์ PC ไปยังรีจิสเตอร์ MAR
22	REQalu_mar	สัญญาณที่กำหนดให้มีการส่งข้อมูลจาก ALU ไปยังรีจิสเตอร์ MAR
23	REQsp_mi	สัญญาณที่กำหนดให้มีการส่งข้อมูลจากรีจิสเตอร์ SP ไปยังรีจิสเตอร์ MI
24	REQacc_mi	สัญญาณที่กำหนดให้มีการส่งข้อมูลจากรีจิสเตอร์ Acc ไปยังรีจิสเตอร์ MI
25	REQalu_mi	สัญญาณที่กำหนดให้มีการส่งข้อมูลจาก ALU ไปยังรีจิสเตอร์ MI
26	REQmi_acc	สัญญาณที่กำหนดให้มีการส่งข้อมูลจากรีจิสเตอร์ MI ไปยังรีจิสเตอร์ Acc

เส้นที่	ชื่อสัญญาณ	หน้าที่การทำงาน
27	REQalu_acc	สัญญาณที่กำหนดให้มีการส่งข้อมูลจาก ALU ไปยังรีจิสเตอร์ Acc
28	REQs_acc	สัญญาณที่กำหนดให้มีการส่งข้อมูลจาก Shifter ไปยังรีจิสเตอร์ Acc
29	REQsp_alu	สัญญาณที่กำหนดให้มีการส่งข้อมูลจากรีจิสเตอร์ SP ไปยัง ALU
30	REQpc_alu	สัญญาณที่กำหนดให้มีการส่งข้อมูลจากรีจิสเตอร์ PC ไปยัง ALU
31	REQacc_alu	สัญญาณที่กำหนดให้มีการส่งข้อมูลจากรีจิสเตอร์ Acc ไปยัง ALU
32	REQmi_alu	สัญญาณที่กำหนดให้มีการส่งข้อมูลจากรีจิสเตอร์ MI ไปยัง ALU
33	REQmi_sp	สัญญาณที่กำหนดให้มีการส่งข้อมูลจากรีจิสเตอร์ MI ไปยังรีจิสเตอร์ SP
34	REQalu_sp	สัญญาณที่กำหนดให้มีการส่งข้อมูลจาก ALU ไปยังรีจิสเตอร์ SP
35	REQmi_pc	สัญญาณที่กำหนดให้มีการส่งข้อมูลจากรีจิสเตอร์ MI ไปยังรีจิสเตอร์ PC
36	REQalu_pc	สัญญาณที่กำหนดให้มีการส่งข้อมูลจาก ALU ไปยังรีจิสเตอร์ PC
37	REQmi_ir	สัญญาณที่กำหนดให้มีการส่งข้อมูลจากรีจิสเตอร์ MI ไปยังรีจิสเตอร์ IR
38	REQacc_s	สัญญาณที่กำหนดให้มีการส่งข้อมูลจากรีจิสเตอร์ Acc ไปยัง Shifter
39	REQalu_flag	สัญญาณที่กำหนดให้มีการส่งข้อมูลจาก ALU ไปยัง Flag
40	REQin_acc	สัญญาณที่กำหนดให้มีการส่งข้อมูลจากรีจิสเตอร์ In ไปยังรีจิสเตอร์ Acc
41	REQacc_out	สัญญาณที่กำหนดให้มีการส่งข้อมูลจากรีจิสเตอร์ Acc ไปยังรีจิสเตอร์ Out

สายสัญญาณตอบรับมีทั้งสิ้น 27 เส้น ส่งจากส่วนเส้นทางข้อมูลกลับไปยังส่วนควบคุมเพื่อสิ้นสุดการทำงาน

เส้นที่	ชื่อสัญญาณ	หน้าที่การทำงาน
0	ACKmi_ir	สัญญาณตอบรับเมื่อรีจิสเตอร์ IR ได้รับข้อมูลจากรีจิสเตอร์ MI แล้ว
1	ACKacc_alu	สัญญาณตอบรับเมื่อ ALU ได้รับข้อมูลจากรีจิสเตอร์ Acc แล้ว
2	ACKpc_alu	สัญญาณตอบรับเมื่อ ALU ได้รับข้อมูลจากรีจิสเตอร์ PC แล้ว
3	ACKsp_alu	สัญญาณตอบรับเมื่อ ALU ได้รับข้อมูลจากรีจิสเตอร์ SP แล้ว
4	ACKalu_acc	สัญญาณตอบรับเมื่อรีจิสเตอร์ Acc ได้รับข้อมูลจาก ALU แล้ว
5	ACKmi_acc	สัญญาณตอบรับเมื่อรีจิสเตอร์ Acc ได้รับข้อมูลจากรีจิสเตอร์ MI แล้ว
6	ACKs_acc	สัญญาณตอบรับเมื่อรีจิสเตอร์ Acc ได้รับข้อมูลจาก Shifter แล้ว
7	ACKalu_pc	สัญญาณตอบรับเมื่อรีจิสเตอร์ PC ได้รับข้อมูลจาก ALU แล้ว
8	ACKmi_pc	สัญญาณตอบรับเมื่อรีจิสเตอร์ PC ได้รับข้อมูลจากรีจิสเตอร์ MI แล้ว
9	ACKalu_sp	สัญญาณตอบรับเมื่อรีจิสเตอร์ SP ได้รับข้อมูลจาก ALU แล้ว
10	ACKmi_sp	สัญญาณตอบรับเมื่อรีจิสเตอร์ SP ได้รับข้อมูลจากรีจิสเตอร์ MI แล้ว
11	ACKalu_mar	สัญญาณตอบรับเมื่อรีจิสเตอร์ MAR ได้รับข้อมูลจาก ALU แล้ว
12	ACKpc_mar	สัญญาณตอบรับเมื่อรีจิสเตอร์ MAR ได้รับข้อมูลจากรีจิสเตอร์ PC แล้ว
13	ACKsp_mar	สัญญาณตอบรับเมื่อรีจิสเตอร์ MAR ได้รับข้อมูลจากรีจิสเตอร์ SP แล้ว
14	ACKmi_mar	สัญญาณตอบรับเมื่อรีจิสเตอร์ MAR ได้รับข้อมูลจากรีจิสเตอร์ MI แล้ว

เส้นที่	ชื่อสัญญาณ	หน้าที่การทำงาน
15	ACKalu_mi	สัญญาณตอบรับเมื่อรีจิสเตอร์ MI ได้รับข้อมูลจาก ALU แล้ว
16	ACKacc_mi	สัญญาณตอบรับเมื่อรีจิสเตอร์ MI ได้รับข้อมูลจากรีจิสเตอร์ Acc แล้ว
17	ACKsp_mi	สัญญาณตอบรับเมื่อรีจิสเตอร์ MI ได้รับข้อมูลจากรีจิสเตอร์ SP แล้ว
18	ACKrom_mi	สัญญาณตอบรับเมื่อรีจิสเตอร์ MI ได้รับข้อมูลจาก ROM แล้ว
19	ACKram_mi	สัญญาณตอบรับเมื่อรีจิสเตอร์ MI ได้รับข้อมูลจาก RAM แล้ว
20	ACKacc_s	สัญญาณตอบรับเมื่อ Shifter ได้รับข้อมูลจากรีจิสเตอร์ Acc แล้ว
21	ACKram_ram	สัญญาณตอบรับเมื่อมีการเขียนข้อมูลลงสู่ RAM แล้ว
22	ACKalu_flag	สัญญาณตอบรับเมื่อ Flag ได้รับข้อมูลจาก ALU แล้ว
23	ACKs_flag	สัญญาณตอบรับเมื่อ Flag ได้รับข้อมูลจาก Shifter แล้ว
24	ACKc_flag	สัญญาณตอบรับเมื่อ Flag ได้รับข้อมูลจากส่วนควบคุมแล้ว
25	ACKin_acc	สัญญาณตอบรับเมื่อรีจิสเตอร์ Acc ได้รับข้อมูลจากรีจิสเตอร์ In แล้ว
26	ACKacc_out	สัญญาณตอบรับเมื่อรีจิสเตอร์ Out ได้รับข้อมูลจากรีจิสเตอร์ Acc แล้ว

ภาคผนวก จ

สายสัญญาณใน Control Storage

สายสัญญาณใน Control Storage มีทั้งสิ้น 43 เส้น ส่งกลับไปให้ AGL 11 เส้น คือ เส้นที่ 0-7, 38, 39, และ 40

เส้นที่ (บิตที่)	ความหมาย
0-7	ตำแหน่งถัดไปใน Control Storage
8-9	00 คือ คำสั่ง LSR 01 คือ คำสั่ง LSL 10 คือ คำสั่ง ROR 11 คือ คำสั่ง ROL
10-11	00 คือ คำสั่ง AND 01 คือ คำสั่ง OR 10 คือ คำสั่ง ADD/SUB 11 คือ คำสั่ง XOR
12	ClearB คือ การทำให้อินพุตของ ALU ค่าหนึ่งเป็น 0
13	Cin สามารถเซ็ตเป็น 0 หรือ 1 ได้
14	สัญญาณบอกให้ทำคำสั่งลบ จะส่งไปพร้อมรหัส 10 ในบิตที่ 10-11
15-16	00 คือ เซ็ตให้ (CF,CF') เป็น (0,0) 01 คือ เซ็ตให้ (CF,CF') เป็น (0,1) 10 คือ เซ็ตให้ (CF,CF') เป็น (1,0) 11 ไม่ได้ใช้งาน
17	สัญญาณบอกให้ RAM ส่งค่าเอาท์พุตออกมา ใช้เมื่อมีการอ่าน RAM (ram_oe)
18	REQmar_rom
19	REQmar_ram
20	REQmi_ram
21-23	000 คือ ไม่มีการส่งข้อมูลอะไร 001 คือ REQmi_mar 010 คือ REQsp_mar 011 คือ REQpc_mar 100 คือ REQalu_mar

เส้นที่ (บิตที่)	ความหมาย
24-25	00 ไม่มีการส่งข้อมูลอะไร 01 คือ REQsp_mi 10 คือ REQacc_mi 11 คือ REQalu_mi
26-27	00 ไม่มีการส่งข้อมูลอะไร 01 คือ REQmi_acc 10 คือ REQalu_acc 11 คือ REQs_acc (Shifter → Acc)
28-29	00 ไม่มีการส่งข้อมูลอะไร 01 คือ REQsp_alu 10 คือ REQpc_alu 11 คือ REQacc_alu
30	REQmi_alu
31	REQmi_sp
32	REQalu_sp
33	REQmi_pc
34	REQalu_pc
35	REQmi_ir
36	REQacc_s (Acc → Shifter)
37	REQalu_flag
38	REQin_acc
39	REQacc_out
40	AMB คือ สัญญาณที่บอกว่าเป็นคำสั่งที่มีการทำงานแยกตาม Addressing Mode
41	Redecode คือ สัญญาณที่ให้ Decode คำสั่งอีกรอบหนึ่งเพื่อให้ชี้ไปยังตำแหน่งใหม่ใน Control Storage
42	DOE (Data Output Enable) คือ สัญญาณที่อนุญาตให้ผลลัพธ์ของตัว Instruction Decoder ออกมาได้

ภาคผนวก ฉ

Microprogrammed Control Storage (Verilog Source Code)

```
/*
Module          : Control Storage
Author          : Anon Tamtrakarn
Code Type       : Behavioral and RTL
Needed Modules : -
*/

`define DEL 1
`define CS_WIDTH 43
`define CS_DEPTH 256
`define ADDR_SZ 8

module control_storage(Dout, address, clk);

input [`ADDR_SZ-1:0] address;
input clk;
output [`CS_WIDTH-1:0] Dout;
reg [`CS_WIDTH-1:0] mem[`CS_DEPTH-1:0], Dout;

/*
00-07 Next Address
08-09 00 - LSR
      01 - LSL
      10 - ROR
      11 - ROL
10-11 00 - AND (Decoder will enable ALU/non-ALU function in IR)
      01 - OR
      10 - ADD/SUB
      11 - XOR
12    ClearB
13    Cin (Decoder will let out (Cin,Cin') due to ALU function in IR)
14    S (Decoder will let out (S,S') due to ALU function in IR)
15-16 00 - (CF,CF') = (0,0)
      01 - (CF,CF') = (0,1)
      10 - (CF,CF') = (1,0)
      11 - No Action / NA
17    ram_oe
18    REQmar_rom
19    REQmar_ram
20    REQmi_ram
21-23 000 No Transfer
      001 REQmi_mar
      010 REQsp_mar
      011 REQpc_mar
      100 REQalu_mar
24-25 00 No Transfer
      01 REQsp_mi
      10 REQacc_mi
      11 REQalu_mi
26-27 00 No Transfer
      01 REQmi_acc
      10 REQalu_acc
      11 REQs_acc
28-29 00 No Transfer
      01 REQsp_alu
      10 REQpc_alu
      11 REQacc_alu
30    REQmi_alu
31    REQmi_sp
```

```
32    REQalu_sp
33    REQmi_pc
34    REQalu_pc
35    REQmi_ir
36    REQacc_s
37    REQalu_flag
38    REQin_acc
39    REQacc_out
40    Addressing-Mode Branching
      (D - Direct, I - Indirect, R - SP/PC Relative, M - Immediate)
41    Redecode
42    doe (Decoder Output Enable)

4_44_33_3333_3333_2222_22222_2111_11111_1100_00000000
2_10_98_7654_3210_9876_54321_0987_65432_1098_76543210
*/

initial begin
// [Fetch]
// REQpc_mar
// ACKpc_mar
mem[0]=`CS_WIDTH'b0_00_00_0000_0000_0000_00011_0000_00000_0000_00000001;
// REQpc_alu, ClearB, Cin=1, ADD
// ACKpc_alu
mem[1]=`CS_WIDTH'b0_00_00_0000_0000_1000_00000_0000_00011_1000_00000010;
// REQmar_rom
// ACKrom_mi
mem[2]=`CS_WIDTH'b0_00_00_0000_0000_0000_00000_0010_00000_0000_00000011;
// REQalu_pc
// ACKalu_pc
mem[3]=`CS_WIDTH'b0_00_00_0001_0000_0000_00000_0000_00000_0000_10100010;

// <PC=PC+1> REQpc_alu, ClearB, Cin=1, ADD
// ACKpc_alu
mem[4]=`CS_WIDTH'b0_00_00_0000_0000_1000_00000_0000_00011_1000_00000101;
// REQalu_pc
// ACKalu_pc
mem[5]=`CS_WIDTH'b0_00_00_0001_0000_0000_00000_0000_00000_0000_00000000;

// [STC]
// (CF,CF') = (1,0)
// ACKc_flag
mem[6]=`CS_WIDTH'b0_00_00_0000_0000_0000_00000_0000_10000_0000_00000000;

// [CLC]
// (CF,CF') = (0,1)
// ACKc_flag
mem[7]=`CS_WIDTH'b0_00_00_0000_0000_0000_00000_0000_01000_0000_00000000;

// [LSR,LSL,ROR,ROL]
// REQacc_s, LSR/LSL/ROR/ROL
// ACKacc_s
mem[8]=`CS_WIDTH'b0_00_00_0100_0000_0000_00000_0000_00000_0000_00001100;
mem[9]=`CS_WIDTH'b0_00_00_0100_0000_0000_00000_0000_00000_0001_00001100;
mem[10]=`CS_WIDTH'b0_00_00_0100_0000_0000_00000_0000_00000_0010_00001100;
mem[11]=`CS_WIDTH'b0_00_00_0100_0000_0000_00000_0000_00000_0011_00001100;
// REQs_acc
// ACKs_acc & ACKs_flag
mem[12]=`CS_WIDTH'b0_00_00_0000_0000_0011_00000_0000_00000_0000_00000000;

// [INA]
// REQacc_alu, ClearB, Cin=1, ADD
// ACKacc_alu
mem[13]=`CS_WIDTH'b0_00_00_0000_0000_1100_00000_0000_00011_1000_00001111;
// [DCA]
// REQacc_alu, ClearB, Cin=0, S=1, ADD
// ACKacc_alu
mem[14]=`CS_WIDTH'b0_00_00_0000_0000_1100_00000_0000_00101_1000_00001111;
```

```
// REQalu_acc
// ACKalu_acc
mem[15]=`CS_WIDTH'b0_00_00_0000_0000_0010_00000_0000_00000_0000_11100000;

// [INS]
// REQsp_alu, ClearB, Cin=1, ADD
// ACKsp_alu
mem[16]=`CS_WIDTH'b0_00_00_0000_0000_0100_00000_0000_00011_1000_00010010;
// [DCS]
// REQsp_alu, ClearB, Cin=0, S=1, ADD
// ACKsp_alu
mem[17]=`CS_WIDTH'b0_00_00_0000_0000_0100_00000_0000_00101_1000_00010010;
// REQalu_sp
// ACKalu_sp
mem[18]=`CS_WIDTH'b0_00_00_0000_0100_0000_00000_0000_00000_0000_00000000;

// [PUS]
// REQsp_mar
// ACKsp_mar
mem[19]=`CS_WIDTH'b0_00_00_0000_0000_0000_00010_0000_00000_0000_11100101;
// REQmar_ram, REQmi_ram
// ACKram_ram
mem[20]=`CS_WIDTH'b0_00_00_0000_0000_0000_00000_1100_00000_0000_11100111;

// [PUL]
// REQsp_alu, ClearB, Cin=0, S=1, SUB
// ACKsp_alu
mem[21]=`CS_WIDTH'b0_00_00_0000_0000_0100_00000_0000_00101_1000_00010110;
// REQalu_sp
// ACKalu_sp
mem[22]=`CS_WIDTH'b0_00_00_0000_0100_0000_00000_0000_00000_0000_11101000;
// REQmar_ram, ram_oe
// ACKram_mi
mem[23]=`CS_WIDTH'b0_00_00_0000_0000_0000_00000_0101_00000_0000_00011000;
// REQmi_acc
// ACKmi_acc
mem[24]=`CS_WIDTH'b0_00_00_0000_0000_0001_00000_0000_00000_0000_00000000;

// [RTS]
// REQsp_alu, ClearB, Cin=0, S=1, ADD
// ACKsp_alu
mem[25]=`CS_WIDTH'b0_00_00_0000_0000_0100_00000_0000_00101_1000_00011010;
// REQalu_sp
// ACKalu_sp
mem[26]=`CS_WIDTH'b0_00_00_0000_0100_0000_00000_0000_00000_0000_11101001;
// REQmar_ram, ram_oe
// ACKram_mi
mem[27]=`CS_WIDTH'b0_00_00_0000_0000_0000_00000_0101_00000_0000_00011100;
// REQmi_pc
// ACKmi_pc
mem[28]=`CS_WIDTH'b0_00_00_0000_1000_0000_00000_0000_00000_0000_00000000;

// <from JMP at 64> REQmar_rom
// AMB, ACKrom_mi -> (D30, I94, R158)
mem[29]=`CS_WIDTH'b0_01_00_0000_0000_0000_00000_0010_00000_0000_00011110;

// <JMP - D> REQmi_pc
// ACKmi_pc
mem[30]=`CS_WIDTH'b0_00_00_0000_1000_0000_00000_0000_00000_0000_00000000;

// <from STA/STS at 212/213> REQmar_ram, REQmi_ram
// ACKram_ram
mem[31]=`CS_WIDTH'b0_00_00_0000_0000_0000_00000_1100_00000_0000_00000100;

// <from * at 35> REQmi_mar
// ACKmi_mar
mem[32]=`CS_WIDTH'b0_00_00_0000_0000_0000_00001_0000_00000_0000_10000001;
```

```
// <from STA/STS> REQmi_mar, redecode, doe
// ACKmi_mar -> (212/213)
mem[33]=`CS_WIDTH'b1_10_00_0000_0000_0000_00001_0000_00000_0000_00000000;

// [HLT]
mem[34]=`CS_WIDTH'b1_00_00_0000_0000_0000_00000_0000_00000_0000_00100010;

// <from * - D at 128> REQmar_rom
// ACKrom_mi
mem[35]=`CS_WIDTH'b0_00_00_0000_0000_0000_00000_0010_00000_0000_00100000;

// <from ** - D at 212> REQmar_rom
// ACKrom_mi
mem[36]=`CS_WIDTH'b0_00_00_0000_0000_0000_00000_0010_00000_0000_00100001;

mem[37]=0;
mem[38]=0;
mem[39]=0;
mem[40]=0;
mem[41]=0;
mem[42]=0;
mem[43]=0;
mem[44]=0;
mem[45]=0;
mem[46]=0;
mem[47]=0;
mem[48]=0;
mem[49]=0;
mem[50]=0;
mem[51]=0;
mem[52]=0;
mem[53]=0;
mem[54]=0;
mem[55]=0;
mem[56]=0;
mem[57]=0;
mem[58]=0;
mem[59]=0;
mem[60]=0;
mem[61]=0;
mem[62]=0;
mem[63]=0;

// [JMP]
// REQpc_mar
// ACKpc_mar
mem[64]=`CS_WIDTH'b0_00_00_0000_0000_0000_00011_0000_00000_0000_00011101;

// [JSR]
// REQpc_alu, ClearB, Cin=1, ADD
// ACKpc_alu
mem[65]=`CS_WIDTH'b0_00_00_0000_0000_1000_00000_0000_00011_1000_01000010;
// REQsp_mar
// ACKsp_mar
mem[66]=`CS_WIDTH'b0_00_00_0000_0000_0000_00010_0000_00000_0000_01000011;
// REQalu_mi
// ACKalu_mi
mem[67]=`CS_WIDTH'b0_00_00_0000_0000_0000_11000_0000_00000_0000_01000100;
// REQmar_ram, REQmi_ram
// ACKram_ram
mem[68]=`CS_WIDTH'b0_00_00_0000_0000_0000_00000_1100_00000_0000_01000101;
// REQsp_alu, ClearB, Cin=1, ADD
// ACKsp_alu
mem[69]=`CS_WIDTH'b0_00_00_0000_0000_0100_00000_0000_00011_1000_01000110;
// REQalu_sp
// ACKalu_sp
mem[70]=`CS_WIDTH'b0_00_00_0000_0100_0000_00000_0000_00000_0000_01000000;
```

```
mem[71]=0;
mem[72]=0;
mem[73]=0;
mem[74]=0;
mem[75]=0;
mem[76]=0;
mem[77]=0;
mem[78]=0;
mem[79]=0;
mem[80]=0;
mem[81]=0;
mem[82]=0;
mem[83]=0;
mem[84]=0;
mem[85]=0;
mem[86]=0;
mem[87]=0;
mem[88]=0;
mem[89]=0;
mem[90]=0;
mem[91]=0;
mem[92]=0;
mem[93]=0;

// <JMP - I> REQmi_mar
// ACKmi_mar
mem[94]=`CS_WIDTH'b0_00_00_0000_0000_0000_00001_0000_00000_0000_01011111;
// REQmar_ram, ram_oe
// ACKram_mi
mem[95]=`CS_WIDTH'b0_00_00_0000_0000_0000_00000_0101_00000_0000_00011110;

// <from * - I at 99> REQmi_mar
// ACKmi_mar
mem[96]=`CS_WIDTH'b0_00_00_0000_0000_0000_00001_0000_00000_0000_10000010;

// <from ** - I at 100> REQmi_mar
// ACKmi_mar
mem[97]=`CS_WIDTH'b0_00_00_0000_0000_0000_00001_0000_00000_0000_01100010;

// <from ** - I at 97> REQmar_ram, ram_oe
// ACKram_mi
mem[98]=`CS_WIDTH'b0_00_00_0000_0000_0000_00000_0101_00000_0000_00100001;

// <from * - I at 128> REQmar_rom
// ACKrom_mi
mem[99]=`CS_WIDTH'b0_00_00_0000_0000_0000_00000_0010_00000_0000_01100000;

// <from ** - I at 212> REQmar_rom
// ACKrom_mi
mem[100]=`CS_WIDTH'b0_00_00_0000_0000_0000_00000_0010_00000_0000_01100001;

mem[101]=0;
mem[102]=0;
mem[103]=0;
mem[104]=0;
mem[105]=0;
mem[106]=0;
mem[107]=0;
mem[108]=0;
mem[109]=0;
mem[110]=0;
mem[111]=0;
mem[112]=0;
mem[113]=0;
mem[114]=0;
mem[115]=0;
mem[116]=0;
mem[117]=0;
```

```
mem[118]=0;
mem[119]=0;
mem[120]=0;
mem[121]=0;
mem[122]=0;
mem[123]=0;
mem[124]=0;
mem[125]=0;
mem[126]=0;
mem[127]=0;

// [AND,OR,ADD,XOR,SUB,LDA,LDS,ADS,SBS,CMP]
// REQpc_mar, AMB
// ACKpc_mar -> * (D35, I99, R163, M227)
mem[128]=`CS_WIDTH'b0_01_00_0000_0000_0000_00011_0000_00000_0000_00100011;

// <from *> REQmar_ram, ram_oe, redecode, doe
// ACKram_mi -> (192, ...)
mem[129]=`CS_WIDTH'b1_10_00_0000_0000_0000_00000_0101_00000_0000_00000000;

// <from * - I at 96> REQmar_ram, ram_oe
// ACKram_mi
mem[130]=`CS_WIDTH'b0_00_00_0000_0000_0000_00000_0101_00000_0000_00100000;

// <from * - R at 160> REQalu_mar
// ACKalu_mar
mem[131]=`CS_WIDTH'b0_00_00_0000_0000_0000_00100_0000_00000_0000_10000001;

mem[132]=0;
mem[133]=0;
mem[134]=0;
mem[135]=0;
mem[136]=0;
mem[137]=0;
mem[138]=0;
mem[139]=0;
mem[140]=0;
mem[141]=0;
mem[142]=0;
mem[143]=0;
mem[144]=0;
mem[145]=0;
mem[146]=0;
mem[147]=0;
mem[148]=0;
mem[149]=0;
mem[150]=0;
mem[151]=0;
mem[152]=0;
mem[153]=0;
mem[154]=0;
mem[155]=0;
mem[156]=0;
mem[157]=0;

// <JMP - R> REQpc_alu, REQmi_alu, ADD
// ACKpc_alu
mem[158]=`CS_WIDTH'b0_00_00_0000_0001_1000_00000_0000_00000_1000_10011111;
// REQalu_pc
// ACKalu_pc
mem[159]=`CS_WIDTH'b0_00_00_0001_0000_0000_00000_0000_00000_0000_00000000;

// <from * - R at 163> REQsp_alu, REQmi_alu, ADD
// ACKsp_alu
mem[160]=`CS_WIDTH'b0_00_00_0000_0001_0100_00000_0000_00000_1000_10000011;

// <from ** - R at 164> REQsp_alu, REQmi_alu, ADD
// ACKsp_alu
```

```
mem[161]=`CS_WIDTH'b0_00_00_0000_0001_0100_00000_0000_00000_1000_10100101;

// <from [Fetch] at 3> REQmi_ir, doe
// ACKmi_ir
mem[162]=`CS_WIDTH'b1_00_00_0010_0000_0000_00000_0000_00000_0000_00000000;

// <from * - R at 128> REQmar_rom
// ACKrom_mi
mem[163]=`CS_WIDTH'b0_00_00_0000_0000_0000_00000_0010_00000_0000_10100000;

// <from ** - R at 212> REQmar_rom
// ACKrom_mi
mem[164]=`CS_WIDTH'b0_00_00_0000_0000_0000_00000_0010_00000_0000_10100001;

// <from ** - R at 161> REQalu_mar, redecode, doe
// ACKalu_mar -> (212/213)
mem[165]=`CS_WIDTH'b1_10_00_0000_0000_0000_00100_0000_00000_0000_00000000;

mem[166]=0;
mem[167]=0;
mem[168]=0;
mem[169]=0;
mem[170]=0;
mem[171]=0;
mem[172]=0;
mem[173]=0;
mem[174]=0;
mem[175]=0;
mem[176]=0;
mem[177]=0;
mem[178]=0;
mem[179]=0;
mem[180]=0;
mem[181]=0;
mem[182]=0;
mem[183]=0;
mem[184]=0;
mem[185]=0;
mem[186]=0;
mem[187]=0;
mem[188]=0;
mem[189]=0;
mem[190]=0;
mem[191]=0;

// [AND]
// REQacc_alu, REQmi_alu, AND
// ACKacc_alu
mem[192]=`CS_WIDTH'b0_00_00_0000_0001_1100_00000_0000_00000_0000_11010000;

// [OR]
// REQacc_alu, REQmi_alu, OR
// ACKacc_alu
mem[193]=`CS_WIDTH'b0_00_00_0000_0001_1100_00000_0000_00000_0100_11010000;

// [ADD]
// REQacc_alu, REQmi_alu, ADD
// ACKacc_alu
mem[194]=`CS_WIDTH'b0_00_00_0000_0001_1100_00000_0000_00000_1000_11010000;

// [XOR]
// REQacc_alu, REQmi_alu, XOR
// ACKacc_alu
mem[195]=`CS_WIDTH'b0_00_00_0000_0001_1100_00000_0000_00000_1100_11010000;

// [INP]
// REQin_acc
// ACKin_acc
```

```
mem[196]=`CS_WIDTH'b0_00_01_0000_0000_0000_00000_0000_00000_0000_00000000;

// [OUT]
// REQacc_out
// ACKacc_out
mem[197]=`CS_WIDTH'b0_00_10_0000_0000_0000_00000_0000_00000_0000_00000000;

// [SUB]
// REQacc_alu, REQmi_alu, SUB, Cin=1, S=1
// ACKacc_alu
mem[198]=`CS_WIDTH'b0_00_00_0000_0001_1100_00000_0000_00110_1000_11010000;

mem[199]=0;

// [LDA]
// REQmi_acc
// ACKmi_acc
mem[200]=`CS_WIDTH'b0_00_00_0000_0000_0001_00000_0000_00000_0000_00000100;

// [LDS]
// REQmi_sp
// ACKmi_sp
mem[201]=`CS_WIDTH'b0_00_00_0000_0010_0000_00000_0000_00000_0000_00000100;

// [ADS]
// REQsp_alu, REQmi_alu, ADD
// ACKsp_alu
mem[202]=`CS_WIDTH'b0_00_00_0000_0001_0100_00000_0000_00000_1000_11010001;

mem[203]=0;
mem[204]=0;
mem[205]=0;

// [SBS]
// REQsp_alu, REQmi_alu, SUB, Cin=1, S=1
// ACKsp_alu
mem[206]=`CS_WIDTH'b0_00_00_0000_0001_0100_00000_0000_00110_1000_11010001;

// [CMP]
// REQacc_alu, REQmi_alu, SUB, Cin=1, S=1
// ACKacc_alu
mem[207]=`CS_WIDTH'b0_00_00_0000_0001_1100_00000_0000_00110_1000_11100001;

// <from 192/193/194/195/198> REQalu_acc
// ACKalu_acc
mem[208]=`CS_WIDTH'b0_00_00_0000_0000_0010_00000_0000_00000_0000_11100001;

// <from ADS at 202 and SBS at 206> REQalu_sp
// ACKalu_sp
mem[209]=`CS_WIDTH'b0_00_00_0000_0100_0000_00000_0000_00000_0000_00000100;

mem[210]=0;

// [STA,STS]
// REQpc_mar, AMB
// ACKpc_mar -> ** (D36, I100, R164, M-)
mem[211]=`CS_WIDTH'b0_01_00_0000_0000_0000_00011_0000_00000_0000_00100100;

// [STA]
// REQacc_mi
// ACKacc_mi
mem[212]=`CS_WIDTH'b0_00_00_0000_0000_0000_10000_0000_00000_0000_00011111;

// [STS]
// REQsp_mi
// ACKsp_mi
mem[213]=`CS_WIDTH'b0_00_00_0000_0000_0000_01000_0000_00000_0000_00011111;
```

```
mem[214]=0;
mem[215]=0;
mem[216]=0;
mem[217]=0;
mem[218]=0;
mem[219]=0;
mem[220]=0;
mem[221]=0;
mem[222]=0;
mem[223]=0;

// REQalu_flag
// ACKalu_flag
mem[224]=`CS_WIDTH'b0_00_00_1000_0000_0000_00000_0000_00000_0000_00000000;

// REQalu_flag
// ACKalu_flag
mem[225]=`CS_WIDTH'b0_00_00_1000_0000_0000_00000_0000_00000_0000_00000100;

mem[226]=0;

// <from * - M at 128> REQmar_rom, redecode, doe
// ACKrom_mi -> * (192, ...)
mem[227]=`CS_WIDTH'b1_10_00_0000_0000_0000_00000_0010_00000_0000_00000000;

mem[228]=0;

// REQacc_mi
// ACKacc_mi
mem[229]=`CS_WIDTH'b0_00_00_0000_0000_0000_10000_0000_00000_0000_11100110;
// REQsp_alu, ClearB, Cin=1, ADD
// ACKsp_alu
mem[230]=`CS_WIDTH'b0_00_00_0000_0000_0100_00000_0000_00011_1000_00010100;

// REQalu_sp
// ACKalu_sp
mem[231]=`CS_WIDTH'b0_00_00_0000_0100_0000_00000_0000_00000_0000_00000000;

// REQalu_mar
// ACKalu_mar
mem[232]=`CS_WIDTH'b0_00_00_0000_0000_0000_00100_0000_00000_0000_00011000;

// REQalu_mar
// ACKalu_mar
mem[233]=`CS_WIDTH'b0_00_00_0000_0000_0000_00100_0000_00000_0000_00011011;

mem[234]=0;
mem[235]=0;
mem[236]=0;
mem[237]=0;
mem[238]=0;
mem[239]=0;
mem[240]=0;
mem[241]=0;
mem[242]=0;
mem[243]=0;
mem[244]=0;
mem[245]=0;
mem[246]=0;
mem[247]=0;
mem[248]=0;
mem[249]=0;
mem[250]=0;
mem[251]=0;
mem[252]=0;
mem[253]=0;
mem[254]=0;
mem[255]=0;
```

```
end

always @(posedge clk) begin
Dout <= #`DEL mem[address];
end

endmodule
```

ภาคผนวก ช

ชุดโปรแกรมทดสอบ

```
; Test Loading & Storing
LDA #1          ; [0,1] Acc = 1
STA 0           ; [2,3] Mem[0] = 1
STA (0)         ; [4,5] Mem[1] = 1
STS 2           ; [6,7] Mem[2] = 224
STS -10,sp      ; [8,9] Mem[214] = 224
STA -11,sp      ; [10,11] Mem[213] = 1
HLT             ; [12] End program
```

โปรแกรมที่ 1 ทดสอบคำสั่งประเภท Load และ Store

```
; Test Shifter
LDA #10         ; [0,1] Acc = 10
STA 0           ; [2,3] Mem[0] = 10
LSR             ; [4] Acc = 5, CF = 1
STA 1           ; [5,6] Mem[1] = 5
LSL             ; [7] Acc = 10, CF = 0
STA 2           ; [8,9] Mem[2] = 10
LDA #1          ; [10,11] Acc = 1
STA 3           ; [12,13] Mem[3] = 1
ROR             ; [14] Acc = -128, CF = 0
STA 4           ; [15,16] Mem[4] = -128
ROL             ; [17] Acc = 1, CF = 1
STA 5           ; [18,19] Mem[5] = 1
HLT             ; [20,21] End program
```

โปรแกรมที่ 2 ทดสอบการทำงานของ Shifter

```
; Test Incrementing & Decrementing
LDA #10     ; [0,1] Acc = 10
STA 0       ; [2,3] Mem[0] = 10
INA         ; [4] Acc = 11
STA 1       ; [5,6] Mem[1] = 11
DCA         ; [7] Acc = 10
DCA         ; [8] Acc = 9
STA 2       ; [9,10] Mem[2] = 9
INS         ; [11] SP = 225
STS 3       ; [12,13] Mem[3] = 225
DCS         ; [14] SP = 224
STS 4       ; [15,16] Mem[4] = 224
HLT         ; [17] End program
```

โปรแกรมที่ 3 ทดสอบการเพิ่มค่าและการลดค่า

```
; Test 1-byte instructions
STS 0       ; [0,1] Mem[0] = 224
LDA 0       ; [2,3] Acc = 224
STC         ; [4] CF = 1
CLC         ; [5] CF = 0
STC         ; [6] CF = 1
NOP         ; [7]
DCA         ; [8] Acc = 223
PUS         ; [9] Mem[224] = 223, SP = 225
STS 1       ; [10,11] Mem[1] = 225
PUS         ; [12] Mem[225] = 223, SP = 226
STS 2       ; [13,14] Mem[2] = 226
PUL         ; [15] Acc = 223, SP = 225
STS 3       ; [16,17] Mem[3] = 225
INP         ; [18] Acc = 100 (Port[In] = 100)
STA 4       ; [19] Mem[4] = 100
INA         ; [20] Acc = 101
STA 5       ; [21] Mem[5] = 101
OUT         ; [22] Port[Out] = 101
HLT         ; [23] End program
```

โปรแกรมที่ 4 ทดสอบคำสั่งที่ไม่มีโอเปอแรนด์

```
; Test Addition & Subtraction
LDA #5          ; [0,1] Acc = 5
STA 0           ; [2,3] Mem[0] = 5
ADD 0           ; [4,5] Acc = 5 + 5 = 10
STA 1           ; [6,7] Mem[1] = 10
ADD #10         ; [8,9] Acc = 10 + 10 = 20
STA 2           ; [10,11] Mem[2] = 20
SUB 2           ; [12,13] Acc = 20 - 20 = 0
STA 3           ; [14,15] Mem[3] = 0
ADS #6          ; [16,17] SP = 230
STS 4           ; [18,19] Mem[4] = 230
SBS #6          ; [20,21] SP = 224
STS 214         ; [22,23] Mem[214] = 224
LDA 4           ; [24,25] Acc = 230
SUB -10,sp      ; [26,27] Acc = 230 - 224 = 6
STA -11,sp      ; [28,29] Mem[213] = 6
HLT             ; [30] End program
```

โปรแกรมที่ 5 ทดสอบการบวกลบ

```
; Test Logic operation
LDA #3          ; [0,1] Acc = 3
STA 0           ; [2,3] Mem[0] = 3
INA             ; [4,5] Acc = 4
STA 20          ; [6,7] Mem[20] = 4
OR 0            ; [8,9] Acc = 4 OR 3 = 7
STA (0)         ; [10,11] Mem[3] = 7
LDA 0           ; [12,13] Acc = 3
AND (0)         ; [14,15] Acc = 3 AND 7 = 3
STA 21          ; [16,17] Mem[21] = 3
XOR (0)         ; [18,19] Acc = 3 XOR 7 = 4
STA (3)         ; [20,21] Mem[7] = 4
XOR 7           ; [22,23] Acc = 4 XOR 4 = 0
STA 22          ; [24,25] Mem[22] = 0, ZF = 1
HLT             ; [26] End program
```

โปรแกรมที่ 6 ทดสอบการทำงานทางตรรกะ

```
; Test Jump operation
CLC         ; [0] CF = 0
JMP 6       ; [1,2]
JMP 8       ; [3,4]
STC         ; [5] CF = 1
JMP -4,pc   ; [6,7]
LDA #10     ; [8,9] Acc = 10
STA 0       ; [10,11] Mem[0] = 10
INA         ; [12] Acc = 11
CMP 0       ; [13,14] if(11!=10) [+3] else [+7]
JEQ 3,pc    ; [15,16]
JNE 7,pc    ; [17,18]
LDA #2      ; [19,20] Acc = 2
STA 3       ; [21,22] Mem[3] = 2
JMP 16,pc   ; [23,24]
LDA #1      ; [25,26] Acc = 1
STA 1       ; [27,28] Mem[1] = 1
LDA #20     ; [29,30] Acc = 20
STA 2       ; [31,32] Mem[2] = 20
LDA #13     ; [33,34] Acc = 13
PUS         ; [35] Mem[224] = 13
LDA 0       ; [36,37] Acc = 10
JMP (224)   ; [38,39]
HLT         ; [40] End program
```

โปรแกรมที่ 7 ทดสอบคำสั่งประเภท Jump

```
; Test Subroutine
LDA #10     ; [0,1] Acc = 10
STA 0       ; [2,3] Mem[0] = 10
DCA         ; [4] Acc = 9
DCA         ; [5] Acc = 8
ADD #10     ; [6,7] Acc = 8 + 10 = 18
STA (0)     ; [8,9] Mem[10] = 18
JSR 6,pc    ; [10,11]
INA         ; [12]
INA         ; [13]
STA 1       ; [14,15] Mem[1] = 10
HLT         ; [16] End program

; Subroutine
SUB #10     ; [17,18] Acc = 18 - 10 = 8
STS 2       ; [19,20] Mem[2] = 225
STA 3       ; [21,22] Mem[3] = 8
RTS         ; [23]
```

โปรแกรมที่ 8 ทดสอบการทำงานกับโปรแกรมย่อย

```
; R = SUM(N) ; Mem[0] = R
             ; Mem[1] = N
             ; Mem[223]  = i

LDA #9       ; [0,1] Acc = 9
STA 1        ; [2,3] N = Acc
LDA #0       ; [4,5] Acc = 0
STA 0        ; [6,7] R = Acc
LDA #1       ; [8,9] Acc = 1
STA 223      ; [10,11] i = Acc
CMP 1        ; [12,13] if(i==N) [+14] else [12]
JEQ 14,pc    ; [14,15]
LDA 0        ; [16,17] Acc = R
ADD 223      ; [18,19] Acc = R + i
STA 0        ; [20,21] R = Acc
LDA 223      ; [22,23] Acc = i
INA          ; [24] Acc = i + 1
STA 223      ; [25,26] i = Acc
JMP 12       ; [27,28]
LDA 0        ; [29,30] Acc = R
ADD 1        ; [31,32] Acc = R + N
STA 0        ; [33,34] R = Acc
HLT          ; [35] End program
```

โปรแกรมที่ 9 โปรแกรมหาค่าผลรวมของ

1+2+3+...+N

```
; R = X^Y       ; Mem[0] = R
                ; Mem[1] = X
                ; Mem[2] = Y
                ; Mem[223]  = temp
                ; Mem[222]  = i
                ; Mem[221]  = j

LDA #5          ; [0,1] Acc = 5
STA 1           ; [2,3] X = Acc
STA 0           ; [4,5] R = Acc
LDA #3          ; [6,7] Acc = 3
STA 2           ; [8,9] Y = Acc
LDA #1          ; [10,11] Acc = 1
STA 1           ; [12,13] i = Acc
LDA 0           ; [14,15] Acc = R
STA 223         ; [16,17] temp = R
LDA #1          ; [18,19] Acc = 1
STA 221         ; [20,21] j = Acc
LDA 222         ; [22,23] Acc = i
CMP 2           ; [24,25] if(i==Y) [+27] else [–39]
JEQ 27,pc       ; [26,27]
LDA 221         ; [28,29] Acc = j
CMP 1           ; [30,31] if(j==X) [+14] else [-16]
JEQ 14,pc       ; [32,33]
LDA 0           ; [34,35] Acc = R
ADD 223         ; [36,37] Acc = R + temp
STA 0           ; [38,39] R = Acc
LDA 221         ; [40,41] Acc = j
INA             ; [42] Acc = j + 1
STA 221         ; [43,44] j = Acc
JMP -16,pc      ; [45,46]
LDA 222         ; [47,48] Acc = i
INA             ; [49] Acc = i + 1
STA 222         ; [50,51] i = Acc
JMP -39,pc      ; [52,53]
HLT             ; [54] End program
```

โปรแกรมที่ 10 โปรแกรมหาค่า X ยกกำลัง Y

ประวัติผู้เขียนวิทยานิพนธ์

นายอนล ธรรมตระการ เกิดเมื่อวันที่ 7 มีนาคม พ.ศ. 2518 ที่จังหวัดกรุงเทพมหานคร สำเร็จการศึกษาปริญญาตรีวิศวกรรมศาสตรบัณฑิต จากภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ มหาวิทยาลัยเกษตรศาสตร์ ในปีการศึกษา 2540 และเข้าศึกษาต่อในหลักสูตรวิทยาศาสตรมหาบัณฑิต สาขาวิชาวิทยาศาสตร์คอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย ในปีการศึกษา 2541