รายงาน
โครงการสิ่งประดิษฐ์

เรื่อง

โปรแกรมจำลองการทำงานของ ไมโครโพรเซสเซอร์ 6800 ระดับคำสั่ง SIM68

โดย

ฐิต ศิริบูรณ์

กันยายน 2542

จุฬาลงกรณ์มหาวิทยาลัย
โครงการสิ่งประดิษฐ์

รายงาน

โปรแกรมจำลองการทำงานของ
ไมโครโพรเซสเซอร์ 6800 ระดับคำสั่ง
SIM68

โดย

อ. ดร. ฐิต ศิริบูรณ์

มิถุนายน 2542

บทคัดย่อ

ชื่อโครงการ โปรแกรมจำลองการทำงานของไมโครโพรเซสเซอร์ 6800ระดับคำสั่ง (SIM68)

ชื่อผู้ดำเนินการ ดร. ฐิต ศิริบูรณ์

เดือนและปีที่ทำวิจัยเสร็จ มิถุนายน 2542

โปรแกรม SIM68 เป็นโปรแกรมที่ใช้จำลองการทำงานของไมโครโพรเซสเซอร์ รุ่น 6800 ของบริษัทมอโตโรรา ในระดับคำสั่ง ผู้ใช้สามารถนำแฟ้มเลขฐาน 16 ที่เก็บชุดคำสั่งภาษาเครื่องของ 6800 มาทดสอบการทำงาน ก่อนไปใช้ในระบบจริง

ด้วยโปรแกรม SIM68 ผู้ใช้สามารถเห็นผลของแต่ละคำสั่งที่มีต่อรีจิสเตอร์และหน่วยความจำ ผู้ใช้จะเข้าใจการทำงานของคำสั่งของ 6800 และการเขียนโปรแกรมด้วยภาษาเอสแซมปลี

Abstract

Project Title 6800 Microprocessor Instructions Level Simulator Program (SIM68)

Name of the Investigator Dr. Thit Siriboon

Year June 1999

SIM68 is a program that simulate Motorola 6800 microprocessor at instructions level. A user can load a hex file that is a machine code program of 6800 and test run the program before using it in a real system.

With SIM68, a user can observe effects of each instruction on registers and memory. The user will gain more understanding on how 6800 instructions work and how to write an assembly language program.

สารบัญ

รายการตารางประกอบ

1. คำนำ

โปรแกรมจำลองการทำงานระดับคำสั่งของไมโครโพรเซสเซอร์จะ "รับ" และปฏิบัติตามชุดคำสั่งภาษาเครื่องของไมโครโพรเซสเซอร์ที่ต้องการ โดยแสดงผลที่เกิดขึ้นกับ register ต่างๆของไมโครโพรเซสเซอร์ รวมถึงส่วนความจำ (memory) ที่ไมโครโพรเซสเซอร์นั้นสามารถเข้าถึงได้โดยตรง (direct access) และยอมให้ผู้ใช้เปลี่ยนแปลงแก้ไขค่าเหล่านี้ได้ เพื่อตรวจสอบและแก้ไขการทำงานของชุดคำสั่งที่กำลังพัฒนาอยู่

2. แบบจำลอง (model) ของ Motorola 6800

2.1. Registers ของ 6800

6800 มี register ที่ผู้ใช้สามารถเปลี่ยนแปลงได้อยู่ 5 register ดังใน ตารางที่ 1.

Accumulator	เป็น register ที่ใช้ในการคำนวณทางคณิตศาสตร์และตรรกะในคำสั่งส่วนใหญ่ค่าที่ใช้ในการทำงานส่วนหนึ่งจะต้องมาจาก accumulator ใน 6800 มี accumulator 2 อัน คือ A และ B แต่ละอันมีขนาด 8 บิท accumulator ทั้ง 2 อันนี้เหมือนกันทุกประการ คำสั่งที่ใช้กับ A จะใช้ได้กับ B และ ในทางกลับกันก็เป็นจริง
Index register หรือ X register	เป็น register ที่มีขนาด 18 บิท register นี้ส่วนใหญ่จะใช้ใน indexed addressing mode คือค่าที่อยู่ใน register จะถูกใช้เป็น address เพื่อนำข้อมูลมาใช้ในคำสั่ง
Stack Pointer register หรือ SP register	เป็น register ที่มีขนาด 18 บิท register นี้เก็บ address ของหน่วยความจำที่ทำหน้าที่เป็น stack ของระบบ 6800 จะไม่กำหนดส่วนของหน่วยความจำที่ทำหน้าที่เป็น stack ตายตัว สามารถเปลี่ยนแปลงได้ตามค่าที่ SP "ชี้" ไป ซึ่งทำได้ด้วยคำสั่ง (instruction) ของ 6800 หน้าที่หลักของ stack คือ ใช้เก็บค่าของ address เมื่อมีการเรียก subroutine หรือ มี interrupt เกิดขึ้น
Conditional Code register หรือ CC register	เป็น register ที่มีขนาด 8 บิท ค่า 2 บิทแรกที่มีนัยสำคัญสูงสุด (Most significant bit) จะเป็น 1 เสมอ ส่วนบิทที่เหลือจะเป็น flag หรือ status bit ที่เปลี่ยนแปลงตามสภาพของผลการคำนวณดังนี้ (เรียงจากซ้ายไปขวา)
H	Half-carry flag จะเป็นที่เก็บตัวทศ (carry) หรือตัวยืม (borrow) จากบิทที่ 3 ไปยังบิทที่ 4

I	Interrupt masked bit ใช้ในการกำหนดว่าไมโครโพรเซสเซอร์จะตอบสนองต่อสัญญาณ interrupt หรือ ไม่ ถ้าเป็น 0 ไมโครโพรเซสเซอร์จะตอบสนอง interrupt (IRQ) แต่ถ้าเป็น 1 ไมโครโพรเซสเซอร์จะไม่สนใจสัญญาณ IRQ
N	Negative flag แสดงว่าผลของการคำนวณเป็นบวกหรือลบ ถ้าเป็น 1 แสดงผลของการคำนวณว่าเป็นลบ แต่ถ้าเป็น 0 แสดงว่าผลของการคำนวณเป็นบวก
Z	Zero flag แสดงว่าผลของการคำนวณเป็น 0 หรือไม่ ถ้า Z เป็น 1 แสดงว่าผลของการคำนวณเป็น 0 แต่ถ้า Z เป็น 0 แสดงว่าผลของการคำนวณไม่เป็น 0
V	Overflow flag แสดงว่าการคำนวณมี 2's compliment overflow หรือไม่ โดยที่ V จะเป็น 1 ถ้ามี overflow และ V จะเป็น 0 ถ้าไม่มี overflow
C	Carry flag เป็นที่เก็บตัวทศ หรือ ตัวยืมจากบิทที่ 7 นอกจากนี้ยังใช้เป็นที่เก็บบิทที่"เกิน" ออกมาในการ Rotate และ Shift

ตารางที่ 1. Register ของ 6800

2.1.1. Program counter register หรือ PC register

PC register เป็น register ที่ผู้ใช้*ไม่สามารถ*เปลี่ยนค่าได้โดยตรง ตัวไมโครโพรเซสเซอร์เป็นผู้เปลี่ยนค่าเองอยู่ตลอดเวลา PC register มีขนาด 16 บิท ใช้เก็บ address ของคำสั่งที่จะถูก execute ในลำดับต่อไป ค่าของ PC register จะถูกเปลี่ยนทุกครั้งที่ 6800 ดึง (fetch) คำสั่ง (instruction) จากหน่วยความจำ หรือ ทำคำสั่ง JUMP หรือ BRANCH

2.1.2. การทำงานของ Stack และ SP (Stack Pointer) Register

ใน 6800 stack จะเป็นส่วนของหน่วยความจำที่ถูก "ชี้" โดย SP register คือ ค่าที่เก็บใน SP register จะเป็น address ของตำแหน่งในหน่วยความจำที่เป็นจุดบน (top) ของ stack ข้อมูลจะถูกเก็บลงบน stack ด้วยคำสั่ง PSH[1] และ ถูกดึงออกมาด้วยคำสั่ง PUL[2] นอกจากนี้ stack ยังใช้ในการทำงานของ subroutine และ interrupt

---

[1]ใน 6800 จะเก็บข้อมูลลง Stack ครั้งละ 1 byte และ ค่าของ SP register จะลดลง 1

[2]ใน 6800 จะดึงข้อมูลออกมาจาก Stack ครั้งละ 1 byte และ ค่าของ SP register จะเพิ่มขึ้น 1

- เวลา 6800 ทำคำสั่ง JSR (jump to subroutine) หรือ BSR (branch to subroutine) address ของตำแหน่งที่กลับจะต้องมาในโปรแกรมหลักจะถูกเก็บลงบน stack และไปทำงานยังโปรแกรมย่อย (subroutine นั้น) จนกระทั่งทำคำสั่ง RTS (return form subroutine) ค่านี้จะถูกดึงออกมาจาก stack
- เวลา 6800 ทำ interrupt ค่า address ของตำแหน่งที่กลับจะต้องมาในโปรแกรมหลัก สภาวะของ 6800 (ค่าของ accumulator A accumulator B X register และ CC register) จะถูกเก็บลงบน stack อ่านค่า address ของตำแหน่งจุดเริ่มต้นของส่วนทำงาน interrupt (interrupt service routine) จาก interrupt vector และไปทำงานที่ตำแหน่งนั้น จนกระทั่งทำคำสั่ง RTI (return form interrupt) ค่าเหล่านี้จะถูกดึงออกมาจาก stack

2.2. รูปแบบของคำสั่งแอสเซมบลีของ 6800

แต่ละคำสั่ง (Instruction) จะประกอบด้วย

Mnemonic	Accumulator	Operand

Mnemonic กำหนดการทำงานว่าเป็นการทำงานอะไร

Accumulator กำหนดว่าจะใช้กับ accumulator A หรือ accumulator B

Operand กำหนดข้อมูลที่ใช้

ตัวอย่าง

Mnemonic	Accumulator	Operand
LDA	A	1234H

หมายความว่า นำค่า (load) จาก address 1234H มาเก็บที่ accumulator A

2.3. *Mnemonic ของไมโครโพรเซสเซอร์ 6800*

ไมโครโพรเซสเซอร์ 6800 ประกอบด้วย Mnemonic ทั้งหมด 72 คำสั่งดังใน ตารางที่ 2.

ABA	BGE	BPL	CLV	INC	NEG	SBA	SWI
ADC	BGT	BRA	CMP	INS	NOP	SBC	TAB
ADD	BHI	BSR	COM	INX	ORA	SEC	TAP
AND	BIT	BVC	CPX	JMP	PSH	SEI	TBA
ASL	BLE	BVS	DAA	JSR	PUL	SEV	TPA
ASR	BLS	CBA	DEC	LDA	ROL	STA	TST
BCC	BLT	CLC	DES	LDS	ROR	STS	TSX
BCS	BMI	CLI	DEX	LDX	RTI	STX	TXS
BEQ	BNE	CLR	EOR	LSR	RTS	SUB	WAI

ตารางที่ 2. แสดง Mnemonic ทั้งหมดของ 6800

2.4. *ผลของคำสั่งต่อ Condition Code register (CC register)*

เมื่อไมโครโพรเซสเซอร์ 6800 ทำ (execute) คำสั่ง (instruction) นอกจาก register ที่ใช้ในคำสั่งนั้นจะถูกเปลี่ยนแปลงแล้ว บิทใน condition code register จะมีการเปลี่ยนแปลงตามไปด้วย เพื่อแสดงสภาวะของผลลัพธ์ที่เกิดขึ้นตามที่อธิบายใน ตารางที่ 1.

ตารางข้างล่างเป็นการสรุปผลของแต่ละคำสั่งที่มีต่อ CC register

	H	I	N	Z	V	C		H	I	N	Z	V	C		H	I	N	Z	V	C		H	I	N	Z	V	C
ABA	*		*	*	*	*	BPL							INC			*	*	*		SBA			*	*	*	*
ADC	*		*	*	*	*	BRA							INS							SBC						1
ADD	*		*	*	*	*	BSR							INX				*			SEC						
AND			*	*		0	BVC							JMP							SEI		1				
ASL			*	*		*	BVS							JSR							SEV					1	
ASR			*	*		*	CBA			*	*	*	*	LDA			*	*	0		STA			*	*	0	
BCC							CLC						0	LDS			*	*	0		STS				*	0	
BCS							CLI		0					LDX			*	*	0		STX				*	0	
BEQ							CLR			0	1	0	0	LSR			0	*		*	SUB			*	*	*	*
BGE							CLV					0		NEG			*	*	*	*	SWI						
BGT							CMP			*	*	*	*	NOP							TAB			*	*	0	
BHI							COM			*	*	0	1	ORA			*	*	0		TAP						
BIT			*	*	0		CPX			*	*	*		PSH							TBA			*	*	0	
BLE							DAA			*	*	*	*	PUL							TPA						
BLS							DEC			*	*	*		ROL			*	*		*	TST			*	*	0	0
BLT							DES							ROR			*	*		*	TSX						
BMI							DEX				*			RTI							TXS						
BNE							EOR			*	*	0		RTS							WAI						

ตารางที่ 3. แสดงผลของแต่ละคำสั่งที่มีต่อ Condition Code register

เครื่องหมาย * แสดงว่าการทำงานของคำสั่งนั้นมีผลต่อ flag นั้น

0,1 แสดงว่าการทำงานของคำสั่งนั้น clear หรือ set flag นั้นเสมอ

ช่องว่าง แสดงว่าการทำงานของคำสั่งนั้นไม่มีผลต่อ flag นั้น

2.5. Addressing mode ของ 6800

Addressing mode คือแบบต่างๆที่ ไมโครโพรเซสเซอร์ จะนำ operand ไปใช้กำหนดข้อมูลในคำสั่ง ไมโครโพรเซสเซอร์ 6800 มี addressing mode ดังในตารางที่ 4.

Direct addressing mode	ใช้ operand เป็น address แต่แทนที่จะต้องใช้ 2 byte (16 bit) ในการกำหนด address จะใช้เพียง 1 byte เพราะจะถือว่าเป็นการอ้างถึง address ใน 128 location แรกเท่านั้น คือจาก 0000_{16} จนถึง $00FF_{16}$ซึ่งจะเปรียบได้กับการใช้ค่าของ operand เป็น lower byte ของ address โดยที่มี higher byte เป็น 00_{16} ตัวอย่าง LDAA FFH หมายความว่า นำค่าจาก location $00FF_{16}$ มาเก็บใน accumulator A
Extended addressing mode	ใช้ operand เป็น address แต่ที่ต่างกับแบบ direct ก็คือจะต้องใช้ 2 byte ซึ่งมีผลทำให้สามารถอ้างถึง address ได้ทุก address หรือ ทั้ง 64K คือจาก 0000_{16} จนถึง $FFFF_{16}$ ตัวอย่าง LDAA FFFFH หมายความว่า นำค่าจาก location $FFFF_{16}$ มาเก็บใน accumulator A
Immediate addressing mode	ใช้ operand เป็นข้อมูลเลยไม่จำเป็นจะต้องไปนำมาจาก memory อีก โดยทั่วไปจะใช้เครื่องหมาย # เป็นการแสดง addressing mode นี้ ตัวอย่าง LDAA #FFH หมายความว่า นำค่า FF_{16} (127_{10}) มาเก็บใน accumulator A
Indexed addressing mode หรือ Indirect addressing mode	ใช้ operand ไปบวกกับค่าที่อยู่ใน X register แล้วใช้ผลที่ได้เป็น address เพื่อไปนำข้อมูลจาก memory รูปแบบที่ใช้จะเป็น operand,X ตัวอย่าง LDAA 3,X ถ้าใน X register มีค่า 1234_{16} จะหมายความว่า นำค่า location 1237_{16} ($3+1234_{16}=1237_{16}$) มาเก็บใน accumulator A

Implied addressing mode หรือ Inherent addressing mode	ในบางคำสั่งจะไม่มี operand เช่น คำสั่ง RTS (return form subroutine) จะเรียกได้ว่าคำสั่งเหล่านี้อยู่ Implied addressing mode
Relative addressing mode	จะเป็น addressing mode ของ conditional branch ตัว operand จะมีขนาด 1 byte โดยจะเป็นการกำหนดว่าจะ branch จากตำแหน่งปัจจุบันไปอีกเท่าใด

ตารางที่ 4. แสดง Addressing Mode ของ 6800

2.6. พื้นที่หน่วยความจำ (Memory Space) ของ 6800

พื้นที่หน่วยความจำที่ 6800 สามารถอ้างถึงได้โดยตรง มีขนาด 64 K byte (65535 byte) โดยจะเป็นหน่วยความจำภายนอกทั้งหมด มี address ตั้งแต่ 0000H ถึง FFFFH แต่ละ address จะมีขนาด 1 byte ไมโครโพรเซสเซอร์ 6800 ใช้พื้นที่หน่วยความจำนี้ในการเก็บ program code, interrupt vector, data และ stack (Princeton architecture)

2.7. Interrupt และ Interrupt Vector ของ 6800

ไมโครโพรเซสเซอร์ 6800 มี interrupt ทั้งหมด 4 อัน เป็น hardware interrupt 3 อัน คือ Reset NMI IRQ และเป็น software interrupt 1 อัน การกำหนดตำแหน่งเริ่มต้นของ interrupt service routine ของ interrupt เหล่านี้ทำได้โดยใช้ interrupt vector คือ ตำแหน่งเฉพาะในหน่วยความจำที่จะเก็บค่า address ของตำแหน่งเริ่มต้นของ interrupt service routine ของ interrupt นั้น เนื่องจาก address ของ ไมโครโพรเซสเซอร์ 6800 มีขนาด 2 byte ดังนั้นแต่ละ interrupt service routine ต้องใช้ 2 address ในหน่วยความจำ

ชื่อ interrupt	เกิดขึ้นเมื่อ	ตำแหน่ง ของ Interrupt vector
IRQ (Hardware Interrupt)	ขา input IRQ ได้รับสัญญาณ Negative edge (การเปลี่ยนแปลงจาก 1 เป็น 0) และ Interrupt bit (I) ใน CC เป็น 0	FFF8 และ FFF9
SWI (Software Interrupt)	6800 execute ถึงคำสั่ง SWI	FFFA และ FFFB
NMI (Hardware Interrupt)	ขา input NMI ได้รับสัญญาณ Negative edge	FFFC และ FFFD
Reset (Hardware Interrupt)	ขา input Reset ได้รับสัญญาณ Negative edge	FFFE และ FFFF

ตารางที่ 5. Interrupt Vectors

2.8. รหัสภาษาเครื่องของ ไมโครโพรเซสเซอร์ 6800

รหัสภาษาเครื่องของแต่ละ instruction จะประกอบด้วย ส่วน opcode และ ส่วน operand

ส่วนของ opcode จะมีขนาด 1 byte และรหัสจะขึ้นกับ mnemonic accumulator และ addressing mode ที่ใช้ ดังแสดงในตารางที่ 6.

Opcode	Mnemonic	Accumulator ที่ใช้	Addressing Mode	Opcode	Mnemonic	Accumulator ที่ใช้	Addressing Mode
1B	ABA		Implied	31	INS		Implied
89	ADC	A	Immediate	8	INX		Implied
99	ADC	A	Direct	6E	JMP		Indexed
A9	ADC	A	Indexed	7E	JMP		Extended
B9	ADC	A	Extended	AD	JSR		Indexed
C9	ADC	B	Immediate	BD	JSR		Extended
D9	ADC	B	Direct	86	LDA	A	Immediate
E9	ADC	B	Indexed	96	LDA	A	Direct
F9	ADC	B	Extended	A6	LDA	A	Indexed
8B	ADD	A	Immediate	B6	LDA	A	Extended
9B	ADD	A	Direct	C6	LDA	B	Immediate
AB	ADD	A	Indexed	D6	LDA	B	Direct
BB	ADD	A	Extended	E6	LDA	B	Indexed
CB	ADD	B	Immediate	F6	LDA	B	Extended
DB	ADD	B	Direct	8E	LDS		Immediate
EB	ADD	B	Indexed	9E	LDS		Direct
FB	ADD	B	Extended	AE	LDS		Indexed
84	AND	A	Immediate	BE	LDS		Extended
94	AND	A	Direct	CE	LDX		Immediate
A4	AND	A	Indexed	DE	LDX		Direct
B4	AND	A	Extended	EE	LDX		Indexed
C4	AND	B	Immediate	FE	LDX		Extended
D4	AND	B	Direct	44	LSR	A	Implied
E4	AND	B	Indexed	54	LSR	B	Implied
F4	AND	B	Extended	64	LSR		Indexed
48	ASL	A	Implied	74	LSR		Extended
58	ASL	B	Implied	40	NEG	A	Implied
68	ASL		Indexed	50	NEG	B	Implied
78	ASL		Extended	60	NEG		Indexed
47	ASR	A	Implied	70	NEG		Extended
57	ASR	B	Implied	1	NOP		Implied
67	ASR		Indexed	8A	ORA	A	Immediate
77	ASR		Extended	9A	ORA	A	Direct
24	BCC		Relative	AA	ORA	A	Indexed
25	BCS		Relative	CA	ORA	B	Immediate
27	BEQ		Relative	DA	ORA	B	Direct
2C	BGE		Relative	EA	ORA	B	Indexed
2E	BGT		Relative	FA	ORA	B	Extended
22	BHI		Relative	36	PSH	A	Implied

85	BIT	A	Immediate	37	PSH	B	Implied
95	BIT	A	Direct	32	PUL	A	Implied
A5	BIT	A	Indexed	33	PUL	B	Implied
B5	BIT	A	Extended	49	ROL	A	Implied
C5	BIT	B	Immediate	59	ROL	B	Implied
D5	BIT	B	Direct	69	ROL		Indexed
E5	BIT	B	Indexed	79	ROL		Extended
F5	BIT	B	Extended	46	ROR	A	Implied
2F	BLE		Relative	56	ROR	B	Implied
23	BLS		Relative	66	ROR		Indexed
2D	BLT		Relative	76	ROR		Extended
2B	BMI		Relative	3B	RTI		Implied
26	BNE		Relative	39	RTS		Implied
2A	BPL		Relative	10	SBA	A	Implied
20	BRA		Relative	82	SBC	A	Immediate
8D	BSR		Relative	92	SBC	A	Direct
28	BVC		Relative	A2	SBC	A	Indexed
29	BVS		Relative	B2	SBC	A	Extended
11	CBA		Implied	C2	SBC	B	Immediate
0C	CLC		Implied	D2	SBC	B	Direct
0E	CLI		Implied	E2	SBC	B	Extended
4F	CLR	A	Implied	F2	SBC	B	Implied
5F	CLR	B	Implied	0D	SEC		Implied
6F	CLR		Indexed	0F	SEI		Implied
7F	CLR		Extended	0B	SEV		Implied
0A	CLV		Implied	97	STA	A	Direct
81	CMP	A	Immediate	A7	STA	A	Indexed
91	CMP	A	Direct	B7	STA	A	Extended
A1	CMP	A	Indexed	D7	STA	B	Direct
B1	CMP	A	Extended	E7	STA	B	Indexed
C1	CMP	B	Immediate	F7	STA	B	Extended
D1	CMP	B	Direct	9F	STS		Direct
E1	CMP	B	Indexed	AF	STS		Indexed
F1	CMP	B	Extended	BF	STS		Extended
43	COM	A	Implied	DF	STX		Direct
53	COM	B	Implied	EF	STX		Indexed
63	COM		Indexed	FF	STX		Extended
73	COM		Indexed	80	SUB	A	Immediate
8C	CPX		Immediate	90	SUB	A	Direct
9C	CPX		Direct	A0	SUB	A	Indexed
AC	CPX		Indexed	B0	SUB	A	Extended
BC	CPX		Extended	C0	SUB	B	Immediate
19	DAA		Implied	D0	SUB	B	Direct
4A	DEC	A	Implied	E0	SUB	B	Indexed
5A	DEC	B	Implied	F0	SUB	B	Extended
6A	DEC		Indexed	3F	SWI		Implied
7A	DEC		Extended	16	TAB		Implied
34	DES		Implied	6	TAP		Implied
9	DEX		Implied	17	TBA		Implied
88	EOR	A	Immediate	7	TPA		Implied
98	EOR	A	Direct	4D	TST	A	Implied
A8	EOR	A	Indexed	5D	TST	B	Implied
B8	EOR	A	Extended	6D	TST		Indexed

C8	EOR	B	Immediate	7D	TST		Extended
D8	EOR	B	Direct	30	TSX		Implied
E8	EOR	B	Indexed	35	TXS		Implied
F8	EOR	B	Extended	3E	WAI		Implied
4C	INC	A	Implied				
5C	INC	B	Implied				
6C	INC		Indexed				
7C	INC		Extended				

ตารางที่ 6. แสดง Opcode ของ 6800

ขนาดของส่วน operand จะขึ้นอยู่กับ addressing mode ดังในตารางที่ 7.

Addressing Mode	ขนาดของ Operand ในภาษาเครื่อง
Direct	1 byte
Extended	2 byte
Immediate	1 byte หรือ 2 byte
Implied	0
Relative	1 byte

ตารางที่ 7. แสดงขนาดของส่วน Operand

3. การใช้งาน SIM68

ในการเขียนชุดคำสั่งให้กับ microprocessor ผู้เขียนเริ่มจาก source code (ทั่วไปจะใช้ภาษาแอสเซมบลี) ชุดคำสั่งถูกเปลี่ยนเป็น "ภาษาเครื่อง" โดยโปรแกรม assembler เพื่อนำไปใส่ในอุปกรณ์ความจำ (เช่น EPROM)

การนำ "ภาษาเครื่อง" นี้ใส่ลงในอุปกรณ์ความจำจะใช้เครื่องมือพิเศษ (EPROM programmer) :ซึ่งจะแปลง "ภาษาเครื่อง" ให้เป็นสัญญาณไฟฟ้าเพื่อ "บันทึก" ลงในอุปกรณ์ความจำ

"ภาษาเครื่อง" ที่ถูกสร้างโดย assembler จะมีลักษณะเป็น Text file ที่เก็บข้อมูล(โปรแกรม) เป็น block ซึ่งนอกจากตัวข้อมูลที่จะบันทึกแล้วจะต้องระบุตำแหน่ง (address) ที่จะนำไปบันทึก ลักษณะการเก็บข้อมูลและ address ใน file มีหลายแบบ เช่น Slash (/) format, S format เป็นต้น เนื่องจากไมโครโพรเซสเซอร์ 6800 เป็นของบริษัท Motorola ดังนั้น assembler ของ 6800 ส่วนใหญ่จะสร้าง file ในแบบ Motorola S format

4. Motorola S format file

ลักษณะของแต่ละ block ใน Motorola S format จะเป็น

SXNNAAAAVV.....VVSS โดยที่

S จะเป็นอักษรตัวแรกของแต่ละ block

X จะเป็น 1 หรือ 9 หรือ 0 ถ้าเป็น 1 แสดงว่าเป็น block ของ data ถ้าเป็น 9 หรือ 0 แสดงว่าเป็น ending block (block ปิดท้าย)ของ file

NN คือ เลขฐาน 16 จำนวน 2 หลัก ที่บอกจำนวน byte ทั้งหมดของ block

AAAA คือ เลขฐาน 16 จำนวน 4 หลัก ที่กำหนด address เริ่มต้นที่จะ load ค่าของ block ลงใน memory

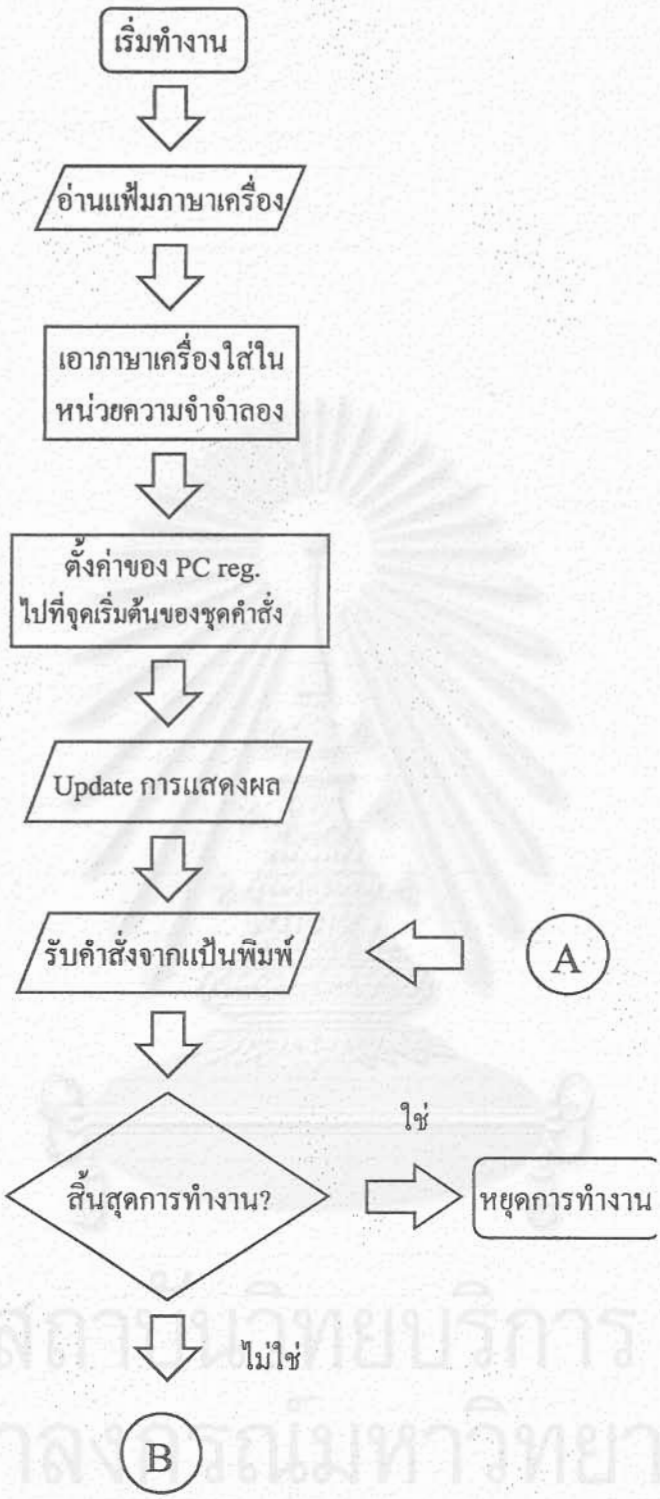
VV คือ ค่าที่จะถูก load เข้าสู่ memory (อาจเป็นโปรแกรม หรือ data)

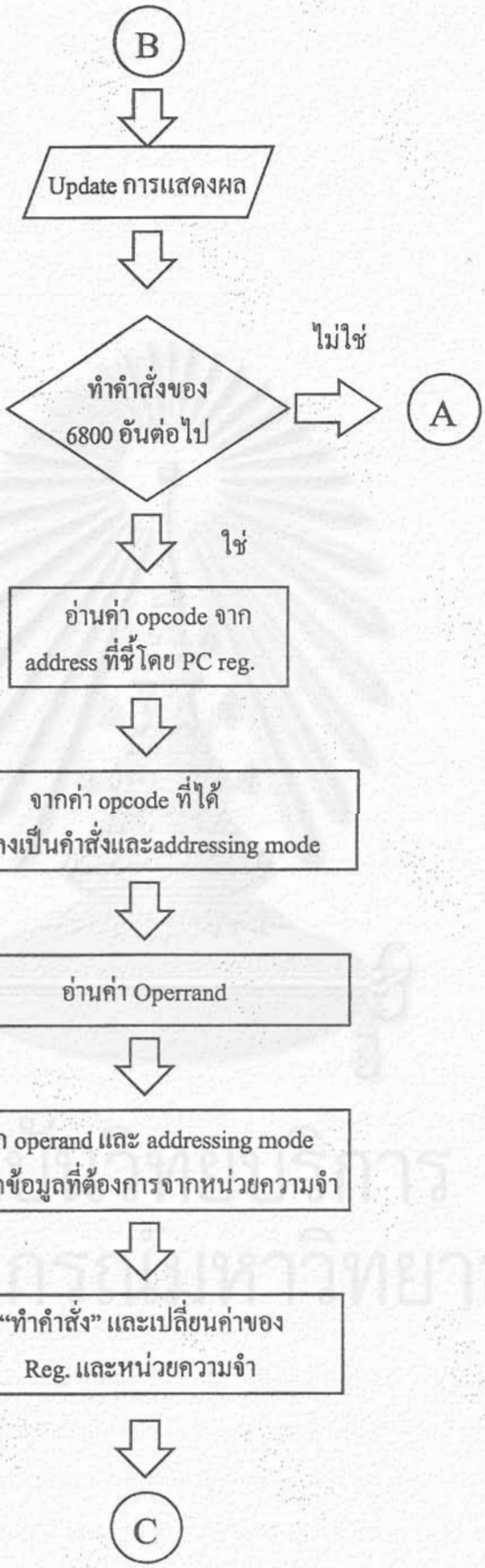
SS คือ check sum ซึ่งเป็น byte สุดท้ายของผลรวมของแต่ละ byte ของ address (AAAA) และ ค่าที่จะถูก load เข้าสู่ memory (VV)

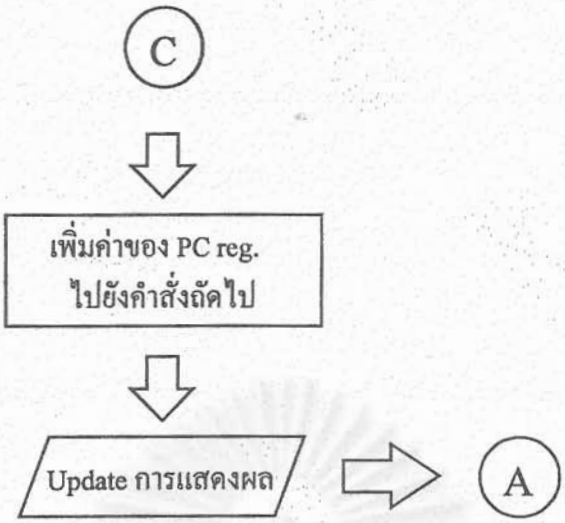
SIM68 จะอ่านแฟ้มข้อมูลที่อยู่ในรูปแบบของ Motorola S format และนำชุดคำสั่งมา "ใส่"ใน address ตามที่กำหนดใน S format เช่นเดียวกับการบันทึกข้อมูลลงในอุปกรณ์ความจำ

5. วิธีดำเนินการ

โปรแกรม sim68 ถูกเขียนขึ้นโดยใช้ภาษาปาสกาล และ ใช้ Borland Turbo Pascal 7.0 เป็นเครื่องมือในการพัฒนา sim68 ทำงานใน DOS และ ในแบบ text mode flow chart ของการทำงานของ Sim68 แสดงในรูปที่ 1




B
Update การแสดงผล
ทำคำสั่งของ
6800 อันต่อไป
ไม่ใช่
A
ใช่
อ่านค่า opcode จาก
address ที่ชี้โดย PC reg.
จากค่า opcode ที่ได้
แปลงเป็นคำสั่งและaddressing mode
อ่านค่า Operrand
จาก operand และ addressing mode
เรียกข้อมูลที่ต้องการจากหน่วยความจำ
"ทำคำสั่ง" และเปลี่ยนค่าของ
Reg. และหน่วยความจำ
C



รูปที่ 1. Flow Chart ของการทำงานของ Sim68

6. การทำงานของโปรแกรม SIM68

6.1. การจำลองหน่วยความจำ

ในการใช้โปรแกรม SIM68 จะเป็นการให้ผู้ใช้นำโปรแกรมภาษาเครื่องของไมโครโพรเซสเซอร์ 6800 ที่ผู้ใช้เขียนขึ้นมาทดลอง run จึงต้องจำลอง "หน่วยความจำ" เพื่อใช้เป็นที่เก็บโปรแกรมภาษาเครื่องและข้อมูลต่างที่จำเป็น ผู้พัฒนาเลือกโครงสร้างข้อมูลแบบตัวแปรชุด (array) เพื่อใช้จำลองหน่วยความจำของไมโครโพรเซสเซอร์ 6800 ทั้งนี้เพื่อสะดวกในการทำคำสั่งกลุ่ม JUMP และ BRANCH

เนื่องจากไมโครโพรเซสเซอร์ 6800 สามารถอ้างถึงหน่วยความจำได้ 64 K byte คือจาก address 0000_{16} ถึง $FFFF_{16}$ และแต่ละ address มีขนาด 1 byte ใน Turbo Pascal มี data type แบบ Byte อยู่แล้วจึงสามารถใช้ array of byte ได้ แต่ใน Turbo Pascal ไม่สามารถจะประกาศ array[0000..FFFF] of byte ได้เพราะ

1. ตัวเลขที่ใช้ในการกำหนดดัชนี(index) จะต้องเป็น integer ซึ่งถ้าจะกำหนดให้ได้ array ที่มี 65536 ช่อง (cell) จะต้องใช้ array[-32768..32767] of byte
2. array[-32768..32767] of byte ไม่สามารถประกาศได้เช่นกันเพราะจะมีโครงสร้างข้อมูลที่มีขนาดใหญ่เกินไป จึงต้องใช้เป็นการประกาศ array[-32767..32767] of byte แทนและมีตัวแปรพิเศษอีก 1 byte เพื่อใช้จำลองหน่วยความจำของ address -32768 (8000_{16})
3. ถึงแม้ว่าจะสามารถประกาศ user define data type ที่เป็น array[-32767..32767] of byte ได้ แต่ไม่สามารถจะประกาศตัวแปรสำหรับ user define data type นี้ได้เนื่องจากใน DOS จำเป็นต้องใช้ data segment ทั้ง segment สำหรับตัวแปรขนาดใหญ่นี้ ใน Turbo Pascal จะต้องไปใช้ส่วนของหน่วยความจำที่ Turbo Pascal เรียกว่า heap ซึ่งอ้างถึงได้โดยการใช้ Pointer และทำ dynamic allocation ตอน run โดยใช้คำสั่ง New

สรุป การจำลองหน่วยความจำ

ใน Module AllConst (AllConst.PAS) ได้ประกาศ user define data type และตัวแปลดังนี้

```
Type
   UpMemoryPtr       = ^UpMemoryType;
   LowMemoryPtr      = Byte;
   UpMemoryType      = array [-32767..32767] of Byte;
Var
   MemoryUp          : UpMemoryPtr;
   MemoryLow         : Byte;
```

และใน ส่วน Main program (SIM68.PAS) ได้ "สร้าง" หน่วยความจำจำลองโดยคำสั่ง

```
New(MemoryUp);
```

การใช้หน่วยความจำมี 2 ลักษณะคือ การอ่านค่าจากหน่วยความจำ และการเขียนค่าลงในหน่วยความจำ ใน SIM68 การอ่านค่าจากหน่วยความจำจะใช้ Procedure GetMemory และ การบันทึก(เขียนค่า)ลงในหน่วยความจำจะใช้ Procedure StoreMemory ซึ่งอยู่ใน SIM68.PAS

```
Procedure GetMemory(Var Address, Num:integer );
begin
  If Address<>MinNum then
     Num:=MemoryUp^[Address]
  Else
     Num:=MemoryLow;
  Inc(Address);
end;


Procedure StoreMemory(Address, Num1 : integer);
var
  Num    : Byte;
begin
  Num:=Lo(Num1);
  If DoRecord then
    With Hist[NextHist] do
    begin
      AddressH:=Address;
      GetMemory(AddressH,Content);
      Dec(AddressH);
      ChangeMem:=True;
    end;
  If Address<>MinNum then
     MemoryUp^[Address]:=Num
  else
     MemoryLow:=Num;

  If (Address>=TopAddress) and (Address<=LastAddress) then
  begin
    ProgMFlag:=True;
  end;
end;
```

Procedure GetMemory มีการทำงานที่ไม่ซับซ้อนคือเพียงแต่ตรวจว่าจะเอาข้อมูลจากที่ใด คือจาก array (ถ้า address อยู่ระหว่าง -32767..32767) หรือจาก ตัวแปร MemoryLow (ถ้า address เป็น 8000_{16}) และเมื่ออ่านข้อมูล

จาก "หน่วยความจำ" แล้ว address จะถูกเพิ่มขึ้นอีก 1 ทั้งนี้เพราะ การอ่านข้อมูลจากหน่วยความจำ ส่วนใหญ่จะเกิดตอน instruction fetch (การอ่านคำสั่งภาษาเครื่อง) ซึ่ง address ที่อ่าน มาจาก PC register และจะถูก increment เพื่อไว้อ่าน instruction ต่อไป

Procedure StoreMemory มีการทำงานที่ในส่วนบันทึกข้อมูลลงหน่วยความจำโดยหลักการเดียวกันกับ Procedure GetMemory แต่มีการทำงานเพิ่มเติมคือ

- โปรแกรม SIM68 สามารถให้ผู้ใช้ทำงานถอยหลัง (Backstep) ได้คือ "ยกเลิก" ผลของการทำคำสั่งต่างได้ ดังนั้นถ้าคำสั่งใดมีการเปลี่ยนแปลงค่าของหน่วยความจำ เมื่อยกเลิกการทำงานของคำสั่งนั้น ค่าที่อยู่ในหน่วยความจำเดิมก็จะกลับมา จึงต้องมีการเก็บค่าที่อยู่ในหน่วยความจำก่อนที่จะถูกบันทึกทำไว้ใน array ชื่อ Hist
- ในขณะที่ SIM68 กำลัง run ภาษา assembly[3] อยู่ จะแสดง Mnemonic ของส่วนของ assembly ที่จอภาพ ดังนั้น ถ้าในขณะที่กำลัง run และการทำคำสั่งของ assembly ทำให้ค่าในหน่วยความจำตำแหน่งที่อยู่ในส่วนที่ถูกแสดงเป็น Mnemonic เปลี่ยนแปลง Mnemonic ก็ต้องเปลี่ยนตามไปด้วย จึงต้องมีการปรับปรุง (update) รายการ Mnemonic ใหม่ ซึ่งทำโดยตรวจว่า address ที่ถูกเปลี่ยนค่าอยู่ในช่วงที่แสดง Mnemonic หรือไม่ (ระหว่าง TopAddress กับ LastAddress) และถ้าใช่ต้องทำรายการ Mnemonic ใหม่โดยการตั้ง Flag ProgMFlag

6.2. การจำลอง register

หน่วยความจำเป็นส่วนที่อยู่นอกไมโครโพรเซสเซอร์ 6800 ส่วนที่อยู่ภายในไมโครโพรเซสเซอร์ 6800 คือ register การทำงานของไมโครโปรเซสเซอร์จะเป็นการอ่านข้อมูลจากภายนอกเข้ามาภายใน ประมวลผล และนำผลที่ได้เก็บยังหน่วยความจำที่อยู่ภายนอก

register ที่ผู้ใช้สามารถใช้ได้ และมีผลต่อการทำงานของ assembly โดยตรงได้แก่ Accumulator A, Accumulator B, Index (X) register, Stack pointer (SP) register, Condition code (CC) register และ Program counter (PC)

register เหล่านี้แบ่งเป็น 2 ขนาดคือ 8 บิท (Accumulator A, Accumulator B และ CC register) และ 16 บิท (X register, SP register และ PC)

เนื่องจาก แต่ละ register จะเก็บค่าเพียง 1 ค่า แต่มีการประมูลผลได้หลายลักษณะ เช่น บวก ลบ การ AND การ OR เป็นต้น และ ในการแสดงผลจะแสดงทั้งแบบ binary แบบ hexadecimal และ แบบ character อีกทั้งผู้ใช้สามารถป้อนข้อมูลได้ทั้งแบบ hexadecimal และ แบบ character จึงใช้โครงสร้างข้อมูลแบบ record ในการจำลอง register ในโปรแกรม SIM68 โดยจะมีส่วนที่เป็น integer เพื่อสะดวกในการประมวลผล มีส่วนที่แทนเลข

---

[3] เพื่อไม่ให้สับสนถ้าใช้คำว่า โปรแกรมจะหมายถึง SIM68 ถ้าใช้ว่าภาษา assembly จะหมายถึงโปรแกรมภาษาเครื่องของ 6800

binary แต่ละบิท(LowerByteB) มีส่วนที่ใช้เก็บตัวแทนเลข hexadecimal (LowerByteH) และ มีส่วนที่เก็บค่าใน แบบเลขฐาน 10 (DecValue) เพื่อใช้ในการแสดงผลหรือรับค่าจากผู้ใช้ ดังการประกาศ type ข้างล่าง

```
BitType       = 0..1;
ByteType      = 0..255;
BinFormat8    = Packed array [0..7] of BitType;
HexFormat2    = Packed array [1..2] of Char;
RegisterType = Record
                 DecValue  : Integer;
                 LowerByteB: BinFormat8;
                 LowerByteH: HexFormat2;
                 Case NumByte : NumByteType of
                   1 : ();
                   2 : ( HighByteB : BinFormat8;
                         HighByteH : HexFormat2);
               end;
```

สำหรับ register ที่มีขนาด 16 บิต ได้แก่ X, SP และ PC เพิ่มส่วนที่เก็บ high order byte คือ HighByteB และ HighByteB

สำหรับ CC Register มีการลักษณะที่ต่างไปคือ แยกเป็น Bit มีการประกาศดังนี้

```
BitofCCType     = (H, I, N, Z, V, C);
CCType          = array[H..C] of bittype;
```

และ ในกรณีของ Accumulator ซึ่งมี 2 อันคือ A และ B ประกาศเป็น Array เพื่อสะดวกต่อการใช้คำสั่ง เนื่องจาก คำสั่งเกือบทั้งหมดสามารถใช้ได้กับ A และ B

```
AccNameType     = (A, B, None);
AccumulatorType = array[A..B] of RegisterType;
```

การประกาศตัวแปรสำหรับ Register ต่างๆมี ดังนี้

```
CC              : CCType;
Accum           : AccumulatorType;
X, SP, PC       : RegisterType;
```

การประกาศ type และ ตัวแปรอยู่ใน Unit Allconst ใน file AllConst.pas

ในการทำงานของ SIM68 จะมีการเปลี่ยนแปลงระหว่าง decimal , binary และ hexadecimal procedure และ function ที่ใช้ในการเปลี่ยนแปลงค่านี้จะอยู่ใน include file ชื่อ Change.inc

```
Procedure ChangeFromBinToHex
Procedure ChangeDecToBin
Procedure ChangeFromBinToDec
Function GetDecFromHex
Procedure ChangeDecToHex
Procedure ChangeDecTo2Hex
Procedure ChangeFromDec
```

7. การแสดงผล

การแสดงผลทั้งหมดนี้ทำใน text mode ของ DOS นอกจากใช้คำสั่ง write และ writeln ของ Pascal แล้ว ยัง ใช้การ "เขียนค่า" ลงในหน่วยความจำของการแสดงผล (video buffer) โดยตรง

ใน text mode ของ DOS แต่ละบรรทัดมี 80 ตัวอักษร (Character) และ มีบรรทัดทั้งหมด 25 บรรทัด รวมเป็น 2,000 ตัวอักษร ตัวอักษรแต่ละตัวใช้หน่วยความจำ 2 bytes คือ bytes แรกจะเป็นรหัส ASCII ของตัวอักษร และ byte ที่ 2 จะเป็น attribute ซึ่งกำหนดลักษณะของตัวอักษรเช่น สีของตัวอักษร สีพื้น ฯลฯ ดังนั้นสำหรับตัวอักษร 2,000 ตัวอักษรต้องใช้หน่วยความจำ 4,000 bytes

SIM68 ได้กำหนด data type ScreenType ซึ่งเป็น array ของ record ที่มี 2 bytes เพื่อใช้กับการจัดการ video buffer ไว้ใน SIM68.PAS ดังนี้

```
ScreenType          = Array [0..1999] of
                        record
                          Ch   : Char;
                          Atch : Byte;
                        end;
```

และกำหนดตัวแปร Screen เป็น pointer ไปยัง ข้อมูลชนิด ScreenType

ตำแหน่ง (address) เริ่มต้นของ video buffer ถูกกำหนดโดย DOS ซึ่งจะขึ้นอยู่กับ video mode โดยอาจแบ่งได้ดังนี้

- จอสี (CGA, EGA, VGA) เริ่มที่ตำแหน่ง B8000H
- จอ Monochrome (HercMono, EGAMono, VGAMono) เริ่มที่ตำแหน่ง B0000H

video mode ถูกตรวจใน Procedure Initialize โดยถ้าเป็น Monochrome จะให้ Screen ชี้ไปที่ address B0000H แต่ถ้าเป็น Color จะให้ Screen ชี้ไปที่ address B8000H

การแสดงผลของตัวอักษรแบ่งเป็น 2 ลักษณะคือแบบปรกติและแบบถูกเลือก ซึ่งทำโดยการกำหนดค่าให้กับ attribute byte ของตัวอักษรนั้น ซึ่งค่าสำหรับลักษณะทั้ง 2 เก็บในตัวแปร TextNorm และ TextInv ตามลำดับ และเป็นค่าที่เหมาะสมกับ video mode ซึ่งถูกเลือกหลังจากการตรวจ video mode ใน Procedure Initialize การกำหนดค่า attribute นี้อาจทำโดยการใส่ค่าลงใน field Atch ของ Screen หรือ ตัวแปร TextAttr ของ Turbo Pascal

การแสดงผลทั้งหมดทำโดยใช้ Procedure Display ในแฟ้ม INC2.INC โดยแบ่งได้เป็น 3 ส่วนคือ

- การแสดง Mnemonic ของชุดคำสั่งภาษาเครื่องของ 6800
- การแสดงค่าที่เก็บใน register ต่างของ 6800 และ ค่าในหน่วยความจำจำลอง
- การแสดงผลออกที่จอภาพจำลอง

7.1. การแสดง Mnemonic ของชุดคำสั่งภาษาเครื่องของ 6800

SIM68 ใช้ data type ShowType ในการเก็บการแสดง Mnemonic ที่จอภาพ

```
ShowType        = record
                    Stg    : String[20];
                    Addr   : Integer;
                    BKPoint: Char;
                  end;
```

โดย Stg เก็บ Mnemonic , สัญลักษณ์ addressing mode (# ถ้าเป็น immediate mode) และ Operand ที่จะแสดงออกมาที่จอภาพ addr เก็บค่า address ของ instruction นั้น และ BKPoint เก็บสัญลักษณ์แทน breakpoint (blank ถ้าไม่มี breakpoint ถ้าเป็น breakpoint แบบถาวร และ >> ถ้าเป็นแบบ breakpoint แบบไม่ถาวร) ส่วนชุดคำสั่งที่ถูกเปลี่ยนเป็นรูปแบบที่ใช้แสดง (ShowType) ถูกเก็บใน array Show ซึ่งเป็น array ที่มีดรรชนีจาก 2 ถึง MaxInstL ที่ดรรชนีเริ่มจาก 2 เพราะในการแสดงที่จอ เริ่มจากบรรทัดที่ 2 ของจอภาพ การเริ่มดรรชนีที่ 2 ทำให้การเขียนโปรแกรมสะดวกขึ้น MaxInstL เป็นค่าคงที่ที่ใช้กำหนดบรรทัดสุดท้ายที่จะแสดง Mnemonic ของชุดคำสั่ง (กำหนดไว้ที่บรรทัดที่ 15)

Procedure MakeInstList ในแฟ้ม INC2.INC สร้างค่าให้กับ array Show จะไปอ่านค่า Opcode จากหน่วยความจำจำลอง โดยเริ่มจาก address ที่กำหนดโดยตัวแปร TopAddress แล้วใช้ค่า Opcode เป็นดรรชนีไปนำข้อมูลจากค่าคงที่แบบ array ของ InstructionType ชื่อ AllIns ตามการประกาศดังนี้

```
AddModeType    = (IMM, DIR, EXT, IND, INH, REL, XXX);
AccNameType    = (A, B, None);
NemunicType    = String[3];
InstructionType= record
                    Nem       : NemunicType;
                    Acc       : AccNameType;
                    AddMode   : AddModeType;
                    ExeTime   : Integer;
                 end;
```

AddModeType เป็น enumerated type ที่กำหนด "ค่า" ตัวแทนของ addressing mode ต่างๆของ 6800 และในกรณีที่เป็น Opcode ที่ 6800 ไม่รู้จักจะใช้ XXX เป็นการแสดงว่าไม่ใช้ addressing mode ของ 6800 AccNameType เป็น enumerated type ที่กำหนด "ค่า" ตัวแทนของชื่อ Accumulator ต่างๆของ 6800 และในกรณีที่เป็น Opcode ที่ไม่ใช้กับ Accumulator จะใช้ None แทน NemunicType เป็น string ขนาด 3 ตัวอักษรใช้เก็บ Mnemonic ของ Opcode ส่วน opcode ของ 6800 มีขนาด 1 byte ดังนั้น instruction ทั้งหมดในทุก addressing mode ที่เป็นได้มีไม่เกิน 256 แบบ ดรรชนีของ AllIns จึงเริ่มจาก 0 ถึง 255

หลังจากที่ได้ข้อมูล Mnemonic และ addressing mode ของ opcode นั้น SIM68 จะคำนวณขนาดของ operand ซึ่งอาจจะมี 0 ถึง 2 bytes และนำ byte เหล่านี้มาจากหน่วยความจำจำลอง แล้วเปลี่ยนเป็นรูปแบบการแสดงผลเก็บลงในแต่ละ cell ของ array Show แล้วไปนำ opcode ต่อไปจากหน่วยความจำจำลองมาอีก เช่นนี้ไปเรื่อยๆจนครบทุก cell ของ array Show (ตามที่กำหนดไว้ทั้งหมด 14 cells)

ในแต่ละครั้งที่อ่านค่า opcode procedure MakeInstList จะคำนวณ address ของ instruction นั้น และไปค้นว่ามี breakpoint ของ instruction นั้นอยู่หรือไม่ (address ของ breakpoint เก็บอยู่ใน array BPUse) ซึ่งการค้นหา breakpoint ทำโดย function SearchBP

SIM68 ใช้ Procedure ShowInst ในการแสดงผลส่วน Mnemonic ของชุดคำสั่งโดยจะตรวจว่าค่าใน PC ยังอยู่ใน array Show หรือไม่ ถ้ายังอยู่จะเปลี่ยนตำแหน่งของตัวอักษรที่ถูกเลือก (เปลี่ยนแถบแสง) แต่ถ้าไม่อยู่จะสร้าง array Show ใหม่โดยเปลี่ยนค่า TopAddress ให้เป็นค่าของ PC แล้วเรียก Procedure MakeInstList

การแสดงชุดคำสั่งที่อยู่ใน array Show ทำโดย procedure DoShowInstList ในการแสดงแต่ละบรรทัดของชุดคำสั่ง proceduew DoShowInstList จะเปรียบเทียบ address ของบรรทัดนั้นว่าตรงกับค่าใน register PC หรือไม่ ถ้าตรงกับค่าของ PC จะแสดงบรรทัดนั้นในแบบถูกเลือก (มีแถบแสง) ถ้าไม่ตรงกับค่าใน PC จะแสดงบรรทัดนั้นในแบบปรกติ

7.2. การแสดงค่าที่เก็บใน register ต่างของ 6800 และ ค่าในหน่วยความจำ

การแสดงค่าที่เก็บใน register แบ่งเป็น 2 กลุ่มคือ

กลุ่มแรก ได้แก่ Accumulator A, Accumulator B, register X และ CC การแสดงผลในกลุ่มนี้จะแสดงในแบบของ bits, Hexadecimal format และ Character (ยกเว้น CC) การแสดงผลในแบบ bits และ Hexadecimal format ทำโดย write ค่าจากส่วนที่เก็บค่าทั้งสองแบบของตัวแปรของ register นั้นออกมา ส่วนการแสดงผลในแบบ Character จะเป็นการแสดงตัวอักษรที่มีรหัส ASCII ตรงกับค่าของ register (ในกรณีของ register X จะแสดง 2 ตัวอักษร) SIM68 ใช้ Function OutDecValue (เป็น local function ที่ประกาศใน procedure Display) ในการเปลี่ยนจากค่าตัวเลขให้เป็นตัวอักษร ในกรณีที่เป็นตัวอักษรที่ไม่สามารถแสดงได้อันได้แก่รหัส ASCII ที่ 0, 7, 8, 9, 10, 13, 127 และ 255 จะใช้อักษร ▒ (ASCII ที่ 176) แทน ผู้ใช้ SIM68 สามารถเปลี่ยนค่าที่เก็บใน register กลุ่มนี้ ได้โดยตรงจากการพิมพ์ค่าที่ต้องการ (ทั้งในแบบ bit, Hexadecimal format และ character) ลงที่ตำแหน่งการแสดงผล

กลุ่มที่สอง ได้แก่ register SP และ register PC ทั้งสอง register มีขนาด 16 bit ค่าที่อยู่ใน register เหล่านี้เป็นค่าที่ใช้อ้างถึง address ของหน่วยความจำ ในการแสดงผลจะแสดงค่าของ register และแสดงค่าจากหน่วยความจำที่อ้างถึงโดย register ในแบบ Hexadecimal format (ใช้เลขฐาน16 สองหลัก ต่อ 1 address) และ แบบ character (ใช้ตัวอักษร 1 ตัว ต่อ 1 address) ในกรณีของ register SP จะแสดง 16 addresses และผู้ใช้สามารถแก้ไขค่าของ register และค่าของหน่วยความจำที่แสดงได้ แต่ในกรณีของ register PC จะแสดง 8 addresses และผู้ใช้ไม่สามารถแก้ไขค่าของ register หรือค่าของหน่วยความจำที่แสดงได้

การแสดงค่าจากหน่วยความจำจำลอง

SIM68 แบ่งการแสดงค่าจากหน่วยความจำจำลองออกเป็นส่วนๆ ทั้งหมด 3 ส่วน โดยแต่ละส่วนจะแสดงค่าของ address "เริ่มต้น" ในแบบ Hexadecimal Format (ใช้เลขฐาน16 สี่หลัก) และค่าจากหน่วยความจำที่ต่อเนื่องกันจาก address "เริ่มต้น" อีก 16 bytes แต่ละ byte แสดงค่าในแบบ Hexadecimal format (ใช้เลขฐาน16 สองหลัก ต่อ 1 address) และแบบ character (ใช้ตัวอักษร 1 ตัว ต่อ 1 address) ค่าของแต่ละ address ที่แสดงจะเปลี่ยนไปตามการทำงานของชุดคำสั่งที่กำลัง run อยู่

ผู้ใช้กำหนด address เริ่มต้นของแต่ละส่วนได้โดยพิมพ์ค่าใหม่ลงใน address เริ่มต้นที่แสดงในแต่ละส่วน และแก้ไขค่าของแต่ละ address ได้โดยการพิมพ์ค่าใหม่ลงในส่วนแสดงผลของ address นั้น ทั้งการแสดงแบบ Hexadecimal format และแบบ character

SIM68 เก็บค่า address เริ่มต้นของแต่ละส่วนใน array WinMemory ซึ่งเป็นชนิด WinMemoryNumType และมีการประกาศดังนี้

```
WinMemoryType     = Record
                      Hex1, Hex2   : HexFormat2;
                      Num          : Integer;
                    end;
WinMemoryNumType = Array[1..3] of WinMemoryType;
```

7.3. การแสดงผลออกที่จอภาพจำลอง

จอภาพจำลองอยู่มุมล่างซ้ายของจอภาพ การแสดงผลที่จอภาพจำลองเกิดจากการทำงานของชุดคำสั่งของ 6800 โดยเรียกใช้ Subroutine ที่ address F803H (OutCh)ซึ่งจะแสดงตัวอักษรของรหัส ASCII ที่อยู่ใน accumulator A หรือ เกิดจากการ echo ของแป้นพิมพ์ที่อ่านเข้าไปด้วยการเรียกใช้ Subroutine ที่ address F800H (InCh)

SIM68 มีตัวแปร UpdateSmallWin ทำหน้าที่เป็น flag เพื่อตรวจสอบว่าจะแสดงผลที่จอภาพจำลองหรือไม่ ค่าของรหัส ASCII ที่ใช้เพื่อแสดงที่จอภาพจำลองถูกเก็บในตัวแปร UpdateValue

SIM68 ใช้คำสั่ง Window ของ Turbo Pascal เพื่อกำหนดขอบเขตของการแสดงผลให้อยู่ในบริเวณของจอภาพจำลอง และใช้คำสั่ง write ในการแสดงตัวอักษรในจอภาพจำลอง เนื่องจากใช้คำสั่ง Windows การทำ word wrap และการเลื่อนบรรทัดจะถูกจัดการโดย Turbo Pascal ซึ่งลดการเขียน code ลง และเพื่อให้การแสดงตัวอักษรในครั้งต่อไปต่อจากตัวอักษรในครั้งปัจจุบัน SIM68 จะเก็บตำแหน่งของ cursor หลังจากการ write ไว้ในตัวแปร CWinX และ CWinY

การทำงานของ subroutine OutCh และ subroutine InCh สร้างขึ้นใน Procedure DoSpecial ที่อยู่ในแฟ้ม SIM68.PAS ซึ่ง Procedure DoSpecial ทำหน้าที่กำหนดค่า ตัวแปร UpdateSmallWinและตัวแปร UpdateValue

8. การรับคำสั่งและข้อมูลจากผู้ใช้

โปรแกรม SIM68 รับคำสั่งและข้อมูลจากผู้ใช้ผ่านทางแป้นพิมพ์ โดย Procedure GetCommand ในแฟ้ม SIM68.PAS จะถูกเรียกหลังจากที่ผู้ใช้กดแป้นพิมพ์ ซึ่งถูกตรวจพบโดย Function KeyPressed

คำสั่งของ SIM68 มีสองแบบ คือใช้ Function Key (F1-F10) พร้อมกับแป้น Shift หรือ แป้น Alt และ แป้น Alt พร้อมกับตัวอักษร ดังแสดงในตารางสรุปคำสั่งของ SIM68

	ไม่กด Shift	กด Shift
F1	RUN (ทำงานต่อเนื่อง)	*ไม่ได้ใช้*
F2	ทำงานที่ละคำสั่ง (Single Step)	ทำงานถอยหลังที่ละคำสั่ง (Back Single Step)
F3	เลื่อนตัวชี้ Breakpoint ขึ้น	ตั้ง Breakpoint แบบถาวร (Sticky)
F4	เลื่อนตัวชี้ Breakpoint ลง	ตั้ง Breakpoint แบบครั้งเดียว (Non-Sticky)
F5	ใช้/ยกเลิก การตรวจสอบค่า (Watch)	เปิดหน้าต่างกำหนดค่าและ register ที่จะตรวจสอบ
F6	*ไม่ได้ใช้*	*ไม่ได้ใช้*
F7	*ไม่ได้ใช้*	*ไม่ได้ใช้*
F8	*ไม่ได้ใช้*	ส่งสัญญาณ IRQ
F9	*ไม่ได้ใช้*	ส่งสัญญาณ NMI
F10	เปลี่ยนแปลงความเร็วในการ RUN	ส่งสัญญาณ Reset

แป้นพิมพ์	การทำงาน
Alt+B	เปิดหน้าต่างสำหรับกำหนดค่า Breakpoint
Alt+L	Load ชุดคำสั่งใหม่
Alt+M	เปิดหน้าต่างสำหรับแก้ไขค่าใน Memory
Alt+Q	สิ้นสุดการทำงานของ SIM68
Alt+R	Reset การทำงานของ SIM68
Alt+F10	แสดงผลในแบบ Monochrome

ตารางที่ 8. สรุปคำสั่งของ SIM68

เมื่อผู้ใช้ป้อนคำสั่งเหล่านี้ Procedure GetCommand จะเรียกใช้ Procedure SpecialKeys (ในแฟ้ม Special.Inc) ซึ่งทำหน้าที่ตรวจสอบคำสั่งและทำตามคำสั่งที่ได้รับ

หมายเหตุ ผู้ใช้สามารถใช้แป้นลูกศร (ซ้าย ขวา ขึ้น ลง) ในการเลื่อนตำแหน่งป้อนข้อมูล การรับแป้นพิมพ์นี้ทำโดย Procedure SpecialKeys เช่นเดียวกัน

ในการป้อนข้อมูลผู้ใช้อาจป้อนข้อมูลในแบบ Binary (0,1) หรือแบบ Hexadecimal (0-F) หรือแบบ Character (ASCII Character) ทั้งนี้ขึ้นอยู่กับ "ตำแหน่ง" ที่ผู้ใช้ป้อนข้อมูล โดยจะมี array ชื่อ ScreenPos ซึ่งเก็บค่า "หน้าที่" ของแต่ละตำแหน่งบนจอภาพ ซึ่งมีการประกาศเป็น User Define Type (Enumerated Type) ดังนี้

7\SH

```
ScreenMType      = (SkX,SkY, Home,
                    HB, IB, NB, ZB, VB, CB,
                    ABin, BBin, XBin,
                    CCHex, AHex,
                    BHex,
                    XHex,
                    PCHex, PCMem,
                    SPHex, SPMem,
                    AddHex1, MemHex1,
                    AddHex2, MemHex2,
                    AddHex3, MemHex3,
                    AChar, BChar, XChar, SPChar,
                    MemChar1,MemChar2,MemChar3,
                    IRQB, NMIB, RESB);
```

ค่าของ 6FUHHQ07\SH	ความหมาย
SkX (SkipX)	ให้เลื่อนไปทางขวาอีกหนึ่งตำแหน่ง
SkY (SkipY)	ให้เลื่อนไปลงอีกหนึ่งบรรทัด
Home	ให้เลื่อนไปที่ตำแหน่งเริ่มต้น
HB, IB, NB, ZB, VB, CB	ตำแหน่งของ Bit ของ H,I,N,Z,V,C ตามลำดับ ผู้ใช้สามารถป้อนข้อมูล 0 หรือ 1 ได้
ABin, BBin, Xbin	ตำแหน่งของค่า Binary ของ Acc. A , Acc. B และ X register ตามลำดับ ผู้ใช้สามารถป้อนข้อมูล 0 หรือ 1 ได้
CCHex, AHex, BHex, Xhex	ตำแหน่งของค่า Hexadecimal ของ CC register, Acc. A , Acc. B และ X register ตามลำดับ ผู้ใช้สามารถป้อนข้อมูล 0-F ได้
PCHex	ตำแหน่งของค่า Hexadecimal ของ PC register (ค่าใน PC register) ผู้ใช้สามารถป้อนข้อมูล 0-F ได้
PCMem	ตำแหน่งของค่า ของหน่วยความจำจำลองที่ PC register "ชี้" ผู้ใช้สามารถป้อนข้อมูล 0-F ได้

SPHex	ตำแหน่งของค่า Hexadecimal ของ SP register (ค่าใน PC register) ผู้ใช้สามารถป้อนข้อมูล 0-F ได้
SPMem	ตำแหน่งของค่า ของหน่วยความจำจำลองที่ SP register "ชี้" ผู้ใช้สามารถป้อนข้อมูล 0-F ได้
AddHex1, AddHex2, AddHex3	ตำแหน่งของค่า Hexadecimal ที่กำหนด address ที่แสดงในแต่ละส่วนของการแสดงหน่วยความจำจำลอง ผู้ใช้สามารถป้อนข้อมูล 0-F ได้
MemHex1, MemHex2, MemHex3	ตำแหน่งของค่า Hexadecimal ของค่าในหน่วยความจำจำลอง ผู้ใช้สามารถป้อนข้อมูล 0-F ได้
Achar, Bchar, XChar	ตำแหน่งของ Character ที่ตรงกับค่าใน Acc. A, Acc. B และ X register ตามลำดับ ผู้ใช้สามารถป้อนข้อมูลเป็น ASCII Character ได้
SPChar	ตำแหน่งของ Character ที่ตรงกับค่าในหน่วยความจำจำลองที่ถูก "ชี้" โดย SP register ผู้ใช้สามารถป้อนข้อมูลเป็น ASCII Character ได้
MemChar1, MemChar2, MemChar3	ตำแหน่งของ Character ที่ตรงกับค่าในหน่วยความจำจำลองที่แสดงในแต่ละส่วน ผู้ใช้สามารถป้อนข้อมูลเป็น ASCII Character ได้
IRQB, NMIB, RESB	ตำแหน่งของ Bit ที่ใช้ในการทำการขัดจังหวะ (Interrupt) แบบ IRQ NMI และ Reset ตามลำดับ ผู้ใช้สามารถป้อนข้อมูล 0 หรือ 1 ได้

9. ภาคผนวก (Source Program)

Source Program ของ SIM68 จะแบ่งออกเป็น Main Program และ Module ต่างๆ ดังนี้

SIM68.Pas	Main Program ของ SIM68 และ มี Include files คือ Change.inc, ExcIns.inc, GetAddr.inc, Inc2.inc, Load.inc, SetScree.inc และ Special.ic
Cursor.Pas	เป็น module ที่จัดการเกี่ยวกับ cursor ของ DOS
NewWin.pas	เป็น module ที่จัดการเกี่ยวกับ การสร้าง "หน้าต่าง" ของ DOS (ใช้ในการแสดง popup menu)
Scancode.pas	เป็น module ที่เก็บระหัส "scan code" ของแป้นพิมพ์
AllConst.pas	เป็น module ที่เก็บการประกาศตัวแปรและชนิดข้อมูลที่ใช้

9.1. File SIM68.PAS

```
Program Sim68;
Uses Crt, Dos, Graph, AllConst, NewWin, ScanCode, Cursor;
Const
     ScreenMaxX      = 74;
     ScreenMinX      = 33;
     ScreenMinY      = 3;
     ScreenMaxY      = 25;
     ChangeMemoryWin= 1;
     ErrorWin        = 2;
     BreakPointWin   = 3;
     SplashScreen    = 4;
     MaxHist         = 100;

     VersionNo       = '1.15';

{  ===================================
   version 1.12
      correct NEG inst
        -set V when result is  10000000
        -set C when result is  00000000
      Add Dummy Subroutine for Input with NoECHO
      Add free memory code to MyExit
    version 1.13
      Fix NMI Problem
    version 1.14
      Add SplashScreen (Alt+T)
    version 1.15
      Clean up some code

=========================================}
Type
   ScreenMType       = (SkX,SkY, Home,

                        HB, IB, NB, ZB, VB, CB,
                        ABin, BBin, XBin,
                        CCHex, AHex,
                        BHex,
                        XHex,
                        PCHex, PCMem,
```

```
                    SPHex, SPMem,
                    AddHex1, MemHex1,
                    AddHex2, MemHex2,
                    AddHex3, MemHex3,
                    AChar, BChar, XChar, SPChar,
                    MemChar1,MemChar2,MemChar3,

                    IRQB, NMIB, RESB
                    );
 ScreenType         = Array [0..1999] of
                         record
                           Ch   : Char;
                           Atch : Byte;
                         end;

 ScreenPtr          = ^ScreenType;
 HistRecType        = Record
                      AccumH           : AccumulatorType;
                      CCH              : CCType;
                      XH,SPH,PCH       : RegisterType;
                      Next,past        : Integer;
                      empty            : Boolean;
                      AddressH,Content: Integer;
                      ChangeMem        : Boolean;
                    end;
 RegValueType       = (WPC, WSP, WX, WA, WB, WCC);
 WatchValueType     = Record
                      Case Name : RegValueType of
                        WA,WB,WX,WSP,WPC : (DecValue : Integer;
                                            On       : Boolean );
                        WCC              : (Value    : CCType;
                                            BitOn    : array [H..C] of
Boolean);
                      end;

Var
  ScreenPos         : Array [ScreenMinX..ScreenMaxX,
                             ScreenMinY..ScreenMaxY] of ScreenMType;
  Hist              : Array [1..MaxHist] of HistRecType;
  NextHist          : Integer;
  CScrX, CScrY      : Byte;
  OScrX, OScrY      : Byte;
  DoRun, Exit       : Boolean;
  Watch             : Boolean;
  IRQF,NMIF,RESF    : Boolean;
  NMILV             : Byte;
  ProgMFlag         : Boolean;
  NewProg           : Boolean;
  ForgetSmallScreen : Boolean;
  FName             : String[40];
  Cycles            : LongInt;
  CScreen           : Char absolute $B800:$0000;
  MScreen           : Char absolute $B000:$0000;


  DoRecord          : Boolean;
  BPOn              : Boolean;

  Screen            : ScreenPtr;

  WatchValue        : Array [WPC..WCC] of WatchValueType;

  ShiftByte         : Byte Absolute $0000:$0417;

{-------------------------------------------------}
Procedure Beep ( Num  : Byte);
```

```
begin
  Case Num of
    1: sound(1760);
    2: sound(1975);
    3: sound(2092);
    4: sound(2349);
    5: sound(2793);
  end;
  Delay(75);
  NoSound;
end;
{-------------------------------------------------}
{$I setscreen.inc}
{-------------------------------------------------}
Procedure GetCommand; Forward;
Procedure DisPlay;    Forward;
Procedure MakeInstList; Forward;
Procedure DoShowInstList; Forward;
Procedure MakeNewAddressLink; Forward;
Procedure ShowInst; Forward;
Procedure ClearMemory; Forward;
Procedure ReSetCPU; Forward;
Procedure ResetVariable; Forward;
Procedure ShowSplashScreen; Forward;
{-------------------------------------------------}
{$I Change.Inc}
{-------------------------------------------------}
Procedure GetMemory(Var Address, Num:integer );
begin
 If Address<>MinNum then
    Num:=MemoryUp^[Address]
 Else
    Num:=MemoryLow;
 Inc(Address);

end;
{-------------------------------------------------}
Procedure StoreMemory(Address, Num1 : integer);
var
 Num   : Byte;
begin
 Num:=Lo(Num1);
 If DoRecord then
   With Hist[NextHist] do
   begin
     AddressH:=Address;
     GetMemory(AddressH,Content);
     Dec(AddressH);
     ChangeMem:=True;
   end;

 If Address<>MinNum then
    MemoryUp^[Address]:=Num
 else
    MemoryLow:=Num;


 If (Address>=TopAddress) and (Address<=LastAddress) then
 begin
   ProgMFlag:=True;
 end;
end;
{-------------------------------------------------}
Procedure CalAddress(AddressMode : AddModeType; Var Address : integer);
Var
 Operand      : OperandType;
```

```
begin
  Case AddressMode of
    DIR     :  begin
                 GetMemory(PC.DecValue,Address);
               end;
    EXT     :  begin
                 GetMemory(PC.DecValue, Operand[1]);
                 GetMemory(PC.DecValue, Operand[2]);
                 Address:=Operand[1]*256+Operand[2];
               end;
    IND     :  begin
                 GetMemory(PC.DecValue,Address);
                 Address:=X.DecValue+Address
               end;
  end;
  ChangeFromDec(PC);
end;
{-------------------------------------------------}
Function FindData (AddressMode: AddModeType):Integer;
Var
  Address,Num : Integer;
begin
  Case AddressMode of
    IMM,REL      :  begin
                 GetMemory(PC.DecValue,Num);
                 ChangeFromDec(PC);
               end;
    DIR,EXT,IND :
               begin
                 CalAddress(AddressMode, Address);
                 GetMemory(Address,Num);
               end;
  end;
  FindData:=Num;
end;
{-------------------------------------------------}
Function NotBit( B1        : BitType) : BitType;
begin
  If (B1=0) then NotBit:=1 else NotBit:=0;
end;
{-------------------------------------------------}
Procedure Push (Value      : Integer);
Var
  Address : Integer;
begin
  With SP do
  begin
    Address:=DecValue;
    StoreMemory(Address,Value);
    Dec(DecValue);{DecValue:=DecValue-1;}
  end;
  ChangeFromDec(SP);
end;
{-------------------------------------------------}
Procedure Pull(var Value      : Integer);
Var
  Address : Integer;
begin
  With SP do
  begin
    Inc(DecValue);{DecValue:=DecValue+1;}
    Address:=DecValue;
    GetMemory(Address,Value);
  end;
  ChangeFromDec(SP);
end;
```

```
{-------------------------------------------------}
Function FindBp ( Var II    : Integer;
                  Address   : Integer) : Boolean;
Var
  Found : Boolean;
begin
  Found:=False;
  II:=0;
  While (Not Found) and (II<MaxBP) do
  begin
    II:=II+1;
    Found:=BPUse[II].Address=Address;
  end;
  If Not Found then
  begin
    II:=0;
    While (Not Found) and (II<MaxBP) do
    begin
      II:=II+1;
      Found:=BPUse[II].Bp=NoBp;
    end;
  end;
  FindBp:=Found;

end;
{-------------------------------------------------}
Procedure PreInterrupt;
Var
  ANum          : RegisterType;
  Num           : Integer;
begin
   ChangeFromDec(PC);
   With PC do
   begin
     Num:=Lo(DecValue);
     Push(Num);
     Num:=Hi(DecValue);
     Push(Num);
   end;
   With X do
   begin
     Num:=Lo(DecValue);
     Push(Num);
     Num:=Hi(DecValue);
     Push(Num);
   end;
   Push(Accum[A].DecValue);
   Push(Accum[B].DecValue);
   With ANum do
   begin
     NumByte:=1;
     LowerByteB[7]:=1;
     LowerByteB[6]:=1;
     LowerByteB[5]:=CC[H];
     LowerByteB[4]:=CC[I];
     LowerByteB[3]:=CC[N];
     LowerByteB[2]:=CC[Z];
     LowerByteB[1]:=CC[V];
     LowerByteB[0]:=CC[C];
     ChangeFromBinToDec(LowerByteB,DecValue);
     Push(DecValue);
   end;

end;
{-------------------------------------------------}
Procedure DoInterrupt( IntName : InterNameType);
```

```
Var
  Address, Num : integer;
begin
   If IntName<>RES then
   begin
     PreInterrupt;
     If IntName=IRQ then
        CC[I]:=1;
   end;
   Address:=InterruptAddr[IntName];
   GetMemory(Address,PC.DecValue);
   GetMemory(Address,Num);
   PC.DecValue:=PC.DecValue*256+Num;
   ChangeFromDec(PC);
end;
{-------------------------------------------------}
Procedure SkipCur(Ch : Char);
begin
  While ScreenPos[CScrX,CScrY] in [SkX, SkY, Home] do
  begin
    Case ScreenPos[CScrX,CScrY] of
      SkX: If Ch in [#77,#80] then
              Inc(CScrX)
           else
              Dec(CScrX);
      SkY: begin
             If Ch in [#77,#80] then
                Inc(CScrY)
             else
                Dec(CScrY);
             CScrX:=ScreenMinX;
           end;
      Home: begin
              CScrX:=ScreenMinX+2;
              CScrY:=ScreenMinY;
            end;
    end;
    If CScrX<ScreenMinX then
    begin

      If CScrY <= ScreenMinY then
         CScrX:=ScreenMinX+2
      else
         CScrX:=ScreenMinX;
      Dec(CScrY);
    end;
    If CScrY < ScreenMinY then CScrY:=ScreenMinY;


  end;
end;
{-------------------------------------------------}
{$I GetAddr.Inc}
{-------------------------------------------------}
Procedure DoBlink;
Var
  Temp : Byte;
begin
   Screen^[80*(OScrY-1)+OScrX-1].Atch:=TextNorm;
   With Screen^[80*(CScrY-1)+CScrX-1] do
     Atch:=TextInv or 128;

end;
{-------------------------------------------------}
{$I Load.INC}
{-------------------------------------------------}
```

```
Procedure TakeCareBP;
{ This procedure }
Var
  Num : Integer;
begin
  If BreakPoint<>NoBP then
  begin
    If BreakPoint=NonStrickBP then
    begin
      For Num:=1 to MaxBp do
      begin
        If BPUse[Num].Address=PC.DecValue then
          BPUse[Num].BP:=NoBp;
      end;
      Show[ShowYPos].BkPoint:=' ';
      Screen^[ShowYPos*80-80].Ch:=' ';
    end;
    BreakPoint:=NoBP;
  end;
end;
{-------------------------------------------------}
{$I  Special.inc}
{-------------------------------------------------}
Procedure DoSpecial(Address : Integer);
Var
  Num, II : Integer;
  Ch      : Char;
begin
   Case Address of
    InCh:begin
           ForgetSmallScreen:=False;
           Repeat
             Ch:=ReadKey;
             If Ch=#0 then
             begin
               SpecialKeys;
             end;
           Until (Ch<>#0) or ForgetSmallScreen;
           If Not ForgetSmallScreen   then
           begin
             Accum[A].DecValue:=Ord(ch);
             UpDateValue:=Ord(ch);
             UpDateSmallWin:=True;
             ChangeFromDec(Accum[A]);
           end;
         end;
    InNoEcho:begin
           ForgetSmallScreen:=False;
           Repeat
             Ch:=ReadKey;
             If Ch=#0 then
             begin
               SpecialKeys;
             end;
           Until (Ch<>#0) or ForgetSmallScreen;
           If Not ForgetSmallScreen   then
           begin
             Accum[A].DecValue:=Ord(ch);
             UpDateSmallWin:=False;
             ChangeFromDec(Accum[A]);
           end;
         end;
    OutCh: begin
             UpDateValue:=Accum[A].DecValue;
             UpDateSmallWin:=True;
           end;
```

```
    RandJsr: begin
               Num:=Random(256);
               Accum[A].DecValue:=Num;
               ChangeFromDec(Accum[A]);
             end;
    end;
end;
{-------------------------------------------------}
function SpecialAddress (Address : Integer):Boolean;
begin
  SpecialAddress:=(Address =OutCh) or (Address=InCh) or (Address=RandJsr)
                  or (Address=InNoEcho);
end;
{-------------------------------------------------}
{$I ExcIns.Inc}
{-------------------------------------------------}
{$I Inc2.Inc}
{-------------------------------------------------}
Procedure RunProgram ; Forward;
{-------------------------------------------------}
Procedure GetCommand;
Var
  Ch   : Char;
  Acc : AccNameType;
  Num, II, Num2,
  BaseAddress,
  Address     : Integer;
  Hex1, Hex2 : HexFormat2;
  Bin         : BinFormat8;
  Fin         : Boolean;
begin
  Ch:=ReadKey;
  If Not (Ch in [#10,#13]) then
  begin
     Case Ch of
       #0      :
                SpecialKeys;
       Else
         begin
           Case ScreenPos[CScrX,CScrY] of

             HB..XBin

                    : begin
                        If Ch in ['0','1'] then
                        begin
                          If Ch='0' then Num:=0 else Num:=1;
                          Case ScreenPos[CScrX,CScrY] of
                            HB: CC[H]:=Num;
                            IB: CC[I]:=Num;
                            NB: CC[N]:=Num;
                            ZB: CC[Z]:=Num;
                            VB: CC[V]:=Num;
                            CB: CC[C]:=Num;
                            ABin: begin
                                    With Accum[A] do
                                    begin
                                      LowerByteB[7-CScrX+ScreenMinX]:=Num;

ChangeFromBintoHex(LowerByteB,LowerByteH);
                                      DecValue:=GetDecFromHex(LowerByteH);
                                    end;
                                  end;
                            BBin: begin
                                    With Accum[B] do
                                    begin
```

```
                                        LowerByteB[7-CScrX+ScreenMinX]:=Num;

ChangeFromBintoHex(LowerByteB,LowerByteH);
                                        DecValue:=GetDecFromHex(LowerByteH);
                                      end;
                                    end;
                                XBin: begin
                                        Num2:=7-CScrX+ScreenMinX;
                                        With X do
                                        begin
                                          If Num2<0 then
                                          begin
                                            LowerByteB[Num2+8]:=Num;

ChangeFromBinToHex(LowerByteB,LowerByteH);
                                          end
                                          else
                                          begin
                                            HighByteB[Num2]:=Num;

ChangeFromBinToHex(HighByteB,HighByteH);
                                          end;
                                          Num:=GetDecFromHex(LowerByteH);
                                          Num2:=GetDecFromHex(HighByteH);
                                          DecValue:=Num2*256+Num;
                                        end;
                                    end;
                                end;
                                DisPlay;
                                OScrX:=CScrX;
                                OScrY:=CScrY;
                                If CScrX<ScreenMaxX then
                                begin
                                  Inc(CScrX);
                                  SkipCur(#77);
                                  DoBlink;
                                end;
                              end
                              else
                                Beep(4);
                            end;
                    CCHex..MemHex3:
                            begin
                              If Ch in ['0'..'9','a'..'f','A'..'F'] then
                              begin
                                If Ch in ['a'..'f'] then
                                   Ch:=Chr(Ord(ch)-32);
                                Hex1[1]:='0';
                                Hex1[2]:=Ch;
                                Num:=GetDecFromHex(Hex1);
                                ChangeDecToBin(Num,Bin);
                                Case ScreenPos[CScrX,CScrY] of
                                  CCHex:begin
                                          If CScrX=(ScreenMinX+9) then
                                          begin
                                            CC[I]:=Bin[0];
                                            CC[H]:=Bin[1];
                                          end
                                          else
                                          begin
                                            CC[C]:=Bin[0];
                                            CC[V]:=Bin[1];
                                            CC[Z]:=Bin[2];
                                            CC[N]:=Bin[3];
                                          end;
                                        end;{CChex}
```

```
AHex,BHex:
      begin
        If ScreenPos[CScrX,CScrY]=AHex then
           Acc:=A
        else
           Acc:=B;
        With Accum[Acc] do
          If CScrX=(ScreenMinX+9) then
          begin
            LowerByteB[4]:=Bin[0];
            LowerByteB[5]:=Bin[1];
            LowerByteB[6]:=Bin[2];
            LowerByteB[7]:=Bin[3];
            LowerByteH[1]:=Ch;
            DecValue:=GetDecFromHex(LowerByteH);
          end
          else
          begin
            LowerByteB[0]:=Bin[0];
            LowerByteB[1]:=Bin[1];
            LowerByteB[2]:=Bin[2];
            LowerByteB[3]:=Bin[3];
            LowerByteH[2]:=Ch;
            DecValue:=GetDecFromHex(LowerByteH);
          end;

      end; {Ahex,Bhex}
XHex: begin
        With X do
        begin
          If CScrX=(ScreenMinX+17) then
          begin
            HighByteB[4]:=Bin[0];
            HighByteB[5]:=Bin[1];
            HighByteB[6]:=Bin[2];
            HighByteB[7]:=Bin[3];
            HighByteH[1]:=Ch;
          end
          else  If CScrX=(ScreenMinX+18) then
          begin
            HighByteB[0]:=Bin[0];
            HighByteB[1]:=Bin[1];
            HighByteB[2]:=Bin[2];
            HighByteB[3]:=Bin[3];
            HighByteH[2]:=Ch;
          end
          else  If CScrX=(ScreenMinX+19) then
          begin
            LowerByteB[4]:=Bin[0];
            LowerByteB[5]:=Bin[1];
            LowerByteB[6]:=Bin[2];
            LowerByteB[7]:=Bin[3];
            LowerByteH[1]:=Ch;
          end
          else
          begin
            LowerByteB[0]:=Bin[0];
            LowerByteB[1]:=Bin[1];
            LowerByteB[2]:=Bin[2];
            LowerByteB[3]:=Bin[3];
            LowerByteH[2]:=Ch;
          end;
          DecValue:=GetDecFromHex
(HighByteH)*256+GetDecFromHex(LowerByteH);
        end;
      end; {XHex}
```

```
                SPHex:begin
                        With SP do
                        begin
                          If CScrX=ScreenMinX then
                          begin
                            HighByteB[4]:=Bin[0];
                            HighByteB[5]:=Bin[1];
                            HighByteB[6]:=Bin[2];
                            HighByteB[7]:=Bin[3];
                            HighByteH[1]:=Ch;
                          end
                          else  If CScrX=(ScreenMinX+1) then
                          begin
                            HighByteB[0]:=Bin[0];
                            HighByteB[1]:=Bin[1];
                            HighByteB[2]:=Bin[2];
                            HighByteB[3]:=Bin[3];
                            HighByteH[2]:=Ch;
                          end
                          else  If CScrX=(ScreenMinX+2) then
                          begin
                            LowerByteB[4]:=Bin[0];
                            LowerByteB[5]:=Bin[1];
                            LowerByteB[6]:=Bin[2];
                            LowerByteB[7]:=Bin[3];
                            LowerByteH[1]:=Ch;
                          end
                          else
                          begin
                            LowerByteB[0]:=Bin[0];
                            LowerByteB[1]:=Bin[1];
                            LowerByteB[2]:=Bin[2];
                            LowerByteB[3]:=Bin[3];
                            LowerByteH[2]:=Ch;
                          end;
                          DecValue:=GetDecFromHex
(HighByteH)*256+GetDecFromHex(LowerByteH);
                        end;
                      end;{SPHex}
                AddHex1,AddHex2,AddHex3:
                      begin
                        Case ScreenPos[CScrX,CScrY] of
                          AddHex1: II:=1;
                          AddHex2: II:=2;
                          AddHex3: II:=3;
                        end;
                        With WinMemory[II] do
                        begin
                          If CScrX=ScreenMinX then
                          begin
                            Hex1[1]:=Ch;
                          end
                          else  If CScrX=(ScreenMinX+1) then
                          begin
                            Hex1[2]:=Ch;
                          end
                          else  If CScrX=(ScreenMinX+2) then
                          begin
                            Hex2[1]:=Ch;
                          end
                          else
                          begin
                            Hex2[2]:=Ch;
                          end;

Num:=GetDecFromHex(Hex1)*256+GetDecFromHex(Hex2);
```

```
                                end;
                              end;{AddHex1..AddHex2}
                          SPMem,MemHex1,MemHex2,MemHex3:
                              begin
                                Case ScreenPos[CScrX,CScrY] of
                                  SPMem  : BaseAddress:=SP.DecValue;
                                  MemHex1: BaseAddress:=WinMemory
[1].Num;
                                  MemHex2: BaseAddress:=WinMemory
[2].Num;
                                  MemHex3: BaseAddress:=WinMemory
[3].Num;
                                end;
                                Num:=CScrX-ScreenMinX-9;
                                If CScrY in [9,12,15,18] then
                                   II:=1
                                else
                                   II:=0;
                                Address:=BaseAddress+II*8+(Num Div 3);
                                GetMemory(Address,Num2);
                                Dec(Address);
                                ChangeDecToHex(Hex1,Num2);
                                II:=Num mod 3;
                                Hex1[II+1]:=Ch;
                                Num2:=GetDecFromHex(Hex1);
                                DoRecord:=False;
                                StoreMemory(Address,Num2);
                                DoRecord:=True;
                              end; {SPMem,MemHex1..MemHex3}
                          end;{Case inside}
                          DisPlay;
                          OScrX:=CScrX;
                          OScrY:=CScrY;
                          If CScrX<ScreenMaxX then
                          begin
                            Inc(CScrX);
                            SkipCur(#77);
                            DoBlink;
                          end;
                        end
                        else
                          Beep(3);
                      end; {CCHex..MemHex3}
    IRQB,NMIB,RESB :
                      begin
                        If Ch in ['0','1'] then
                        Case ScreenPos[CScrX,CScrY] of
                            IRQB   :
                                    begin
                                      Sound(880);
                                      Delay(200);
                                      NoSound;
                                      IRQF:= (CH='0') and (CC[I]=0);
                                      Screen^[368].Ch:=Ch;
                                    end;{^F8 IRQ}
                            NMIB   :
                                    begin
                                      Sound(987);
                                      Delay(200);
                                      NoSound;
                                      If Not (((NMILV=1) and (Ch='1')) or
((NMILV=0) and (Ch='0'))) then
                                         If NMILV=1 then
                                         begin
                                           NMILV:=0;
                                           NMIF:=True;
```

```
                                    end
                                    else
                                    begin
                                      NMILV:=1;
                                      NMIF:=False;
                                    end;
                                  Screen^[374].Ch:=Ch;

                                end; {^F9 NMI}

                        RESB    :
                                  begin
                                    Sound(880);
                                    Delay(200);
                                    Sound(987);
                                    Delay(200);
                                    Sound(1046);
                                    Delay(200);
                                    NoSound;
                                    RESF:=Ch='0';
                                    Screen^[380].Ch:=Ch;
                                  end; {^F10 RES}

                      end{Case inside}
                      else
                        BeeP(3);
                    end;
    AChar..MemChar3:
                    begin
                      Case ScreenPos[CScrX,CScrY] of
                        AChar, BChar :
                              begin
                                If ScreenPos[CScrX,CScrY] = AChar then
                                   Acc:=A
                                else
                                   Acc:=B;
                                Accum[Acc].DecValue:=Ord(Ch);
                                ChangeFromDec(Accum[Acc]);
                              end;
                        XChar:  begin
                                  ChangeDecToHex(Hex1,Ord(Ch));
                                  If ScreenPos[CScrX-1,CScrY]=XChar then
                                     X.LowerByteH:=Hex1
                                  else
                                     X.HighByteH:=Hex1;
                                  With X do
                                  begin
                                    Num:=GetDecFromHex(HighByteH);
                                    Num2:=GetDecFromHex(LowerByteH);
                                    DecValue:=Num*256+Num2;
                                    ChangeDecToBin(Num,HighByteB);
                                    ChangeDecToBin(Num2,LowerByteB);
                                  end;
                                end;

                        SPChar,MemChar1,MemChar2,MemChar3:
                              begin
                                Case ScreenPos[CScrX,CScrY] of
                                  SPChar  : BaseAddress:=SP.DecValue;
                                  MemChar1: BaseAddress:=WinMemory
[1].Num;
                                  MemChar2: BaseAddress:=WinMemory
[2].Num;
                                  MemChar3: BaseAddress:=WinMemory
[3].Num;
                                end;
```

```
                                    Num:=CScrX-ScreenMinX-33;
                                    If CScrY in [9,12,15,18] then
                                       II:=1
                                    else
                                       II:=0;
                                    Address:=BaseAddress+II*8+Num ;
                                    DoRecord:=False;
                                    StoreMemory(Address,Ord(Ch));
                                    DoRecord:=True;
                                  end; {SPMem,MemHex1..MemHex3}


                          end;
                          DisPlay;
                          OScrX:=CScrX;
                          OScrY:=CScrY;
                          If CScrX<ScreenMaxX then
                          begin
                            Inc(CScrX);
                            SkipCur(#77);
                            DoBlink;
                          end;
                        end;{AChar..MemChar3}


          end;
        end;
      end;
    end;
    If ProgMFlag then
    begin
      MakeInstList;
      DoShowInstList;
      DisPlay;
      ProgMFlag:=False;
    end;
  end;
  {-------------------------------------------------}
  Procedure SaveHist;
  begin
    With Hist[NextHist] do
    begin
      AccumH:=Accum;
      CCH:=CC;
      SPH:=SP;
      XH:=X;
      PCH:=PC;
      Empty:=False;
      ChangeMem:=False;
    end;
  end;
  {-------------------------------------------------}
  Procedure RunProgram;
  Var
    Code : Integer;
    Inst : InstructionType    ;

  begin
    begin
      While BreakPoint<>NoBp do
        If KeyPressed then GetCommand;
      If Not Wait then
      begin
        SaveHist;
        GetMemory(PC.DecValue, Code);
        ExecuteIns(Code, Inst );
```

```
      Cycles:=Cycles+Inst.ExeTime;
      NextHist:=Hist[NextHist].Next;
    end;

    If RESF then
    begin
      RESF:=False;
      If (Speed=Fast) and (RuningMode=Running) then Delay(500);
      DoInterrupt(RES);
      GotoXy(61,5); write('1');
      Wait:=False;
    end;

    If NMIF then
    begin
      DoInterrupt(NMI);
      NMIF:=False;
      Wait:=False;
    end;
    If IRQF and (CC[I]=0) then
    begin
       DoInterrupt(IRQ);
       Wait:=False;
    end;
    If RuningMode=SingleStep then DoRun:=False;
  end;

end;
{-------------------------------------------------}
Procedure TakeCareWatch;
Var
  Ch      : Char;
  CCBit   : BitOfCCType;
begin
  If WatchValue[WPC].On and (WatchValue[WPC].DecValue=PC.DecValue) then
  begin
    Beep(1);
    Beep(2);
    Beep(4);
    Beep(1);
    ShowWin(ErrorWin);
    Writeln(' PC=$',PC.HighByteH,PC.LowerByteH);
    Write(' Hit any key to continue '); Ch:=ReadKey;
    CloseLastWin;
    TextAttr:=TextNorm;
    If RuningMode<>SingleStep then
    begin
      RuningMode:=SingleStep;
      GotoXy(65,3); Write('Step');
    end;
    DoRun:=False;
  end
  else If WatchValue[WSP].On and (WatchValue[WSP].DecValue=SP.DecValue) then
  begin
    Beep(1);
    Beep(2);
    Beep(4);
    Beep(1);
    ShowWin(ErrorWin);
    Writeln(' SP=$',SP.HighByteH,SP.LowerByteH);
    Write(' Hit any key to continue '); Ch:=ReadKey;
    CloseLastWin;
    TextAttr:=TextNorm;
    If RuningMode<>SingleStep then
    begin
      RuningMode:=SingleStep;
```

```
    GotoXy(65,3); Write('Step');
   end;
   DoRun:=False;
 end
 else If WatchValue[WX].On and (WatchValue[WX].DecValue=X.DecValue) then
 begin
   Beep(1);
   Beep(2);
   Beep(4);
   Beep(1);
   ShowWin(ErrorWin);
   Writeln(' X=$',X.HighByteH,X.LowerByteH);
   Write(' Hit any key to continue '); Ch:=ReadKey;
   CloseLastWin;
   TextAttr:=TextNorm;
   If RuningMode<>SingleStep then
   begin
     RuningMode:=SingleStep;
     GotoXy(65,3); Write('Step');
   end;
   DoRun:=False;
 end
 else If WatchValue[WA].On and (WatchValue[WA].DecValue=Accum[A].DecValue)
then
 begin
   Beep(1);
   Beep(2);
   Beep(4);
   Beep(1);
   ShowWin(ErrorWin);
   Writeln(' Acc. A=$',Accum[A].LowerByteH);
   Write(' Hit any key to continue '); Ch:=ReadKey;
   CloseLastWin;
   TextAttr:=TextNorm;
   If RuningMode<>SingleStep then
   begin
     RuningMode:=SingleStep;
     GotoXy(65,3); Write('Step');
   end;
   DoRun:=False;
 end
 else If WatchValue[WB].On and (WatchValue[WB].DecValue=Accum[B].DecValue)
then
 begin
   Beep(1);
   Beep(2);
   Beep(4);
   Beep(1);
   ShowWin(ErrorWin);
   Writeln(' Acc. B=$',Accum[B].LowerByteH);
   Write(' Hit any key to continue '); Ch:=ReadKey;
   CloseLastWin;
   TextAttr:=TextNorm;
   If RuningMode<>SingleStep then
   begin
     RuningMode:=SingleStep;
     GotoXy(65,3); Write('Step');
   end;
   DoRun:=False;
 end
 Else
 begin
   CCBit:=H;
   While ((Not WatchValue[WCC].BitOn[CCBit])
         or (WatchValue[WCC].Value[CCBit]<>CC[CCBit]))
         and (CCBit<C) do
```

```
        CCBit:=Succ(CCBit);
    If  WatchValue[WCC].BitOn[CCBit]
        and (WatchValue[WCC].Value[CCBit]=CC[CCBit]) then
    begin
    Beep(1);
    Beep(2);
    Beep(4);
    Beep(1);
    ShowWin(ErrorWin);
    Write(' CC Bit ');
    Case CCBit of
     H: Write('H');
     I: Write('I');
     N: Write('N');
     Z: Write('Z');
     V: Write('V');
     C: Write('C');
    end;
    writeln('=',CC[CCBit]);
    Write(' Hit any key to continue '); Ch:=ReadKey;
    CloseLastWin;
    TextAttr:=TextNorm;
    If RuningMode<>SingleStep then
    begin
      RuningMode:=SingleStep;
      GotoXy(65,3); Write('Step');
    end;
    DoRun:=False;
    end;
  end;
  GotoXy(80,25);
end;
Var
  OldExit  : Pointer;
{$F+}
Procedure MyExit;
{$F-}
begin
  ExitProc:=OldExit;
  ClrScr;
  {Dispose(MemoryUp)}; {free memory}
  Set_Display(OldALMode);
  Set_Page(OLDBH);
  Set_Cursor(OldCH,OldCL);
  TextMode(OldMode);
end;
{-------------------------------------------------}
begin {Main program}
  OldMode:=LastMode;
  Get_DisPlay(OldALMode,OldBH);
  Get_Cursor(OldCh,OldCL,OLDBH);
  OldExit:=ExitProc;
  ExitProc:=@MyExit;
  Randomize;
  New(MemoryUp);
  MemoryLow:=0;


  SetScreenPos;
  Initialize;
  MakeInstList;
  DoShowInstList;
  DisPlay;
  LoadProgramS;
```

```
  Display;
  ShowSplashScreen;
  CheckBreak:=True;
  Set_Cursor(32,0);
  Repeat
    If ProgMFlag then
    begin
       MakeInstList;
       DoShowInstList;
       DisPlay;
       ProgMFlag:=False;
    end
    else
       DisPlay;
    If Watch then TakeCareWatch;
    Repeat
      If KeyPressed then
        GetCommand;
    Until DoRun ;
    If RuningMode=Running then
       Case Speed of
         Fast:;
         Mid: Delays(0,19);
         Slow: Delays(0,49);
       end;
    If Not Exit then RunProgram;
  Until Exit;
  Dispose(MemoryUP);
  Set_Display(OldALMode);
  Set_Page(OLDBH);
  Set_Cursor(OldCH,OldCL);
  TextMode(OldMode);
  ClrScr;
end.
```

9.2. File SETSCREEN.INC

```
{-------------------------------------------------}
Procedure SetScreenPos;
Var
  IX, IY :integer;
begin
  For Ix:=ScreenMinX to ScreenMaxX-1 do
    For IY:=ScreenMinY to ScreenMaxY do
        ScreenPos[Ix,IY]:=SkX;
  For IY:=ScreenMinY to ScreenMaxY do
        ScreenPos[ScreenMaxX,IY]:=SkY;
  For Ix:=ScreenMinX to ScreenMaxX-1 do
    ScreenPos[IX,7]:=SkY;
  For Ix:=ScreenMinX to 42 do
    ScreenPos[IX,8]:=Skx;
  For Ix:=ScreenMinX to 42 do
    ScreenPos[IX,9]:=Skx;
  For Ix:=ScreenMinX to ScreenMaxX-1 do
    ScreenPos[IX,10]:=SkY;
  For Ix:=ScreenMinX to ScreenMaxX-1 do
    ScreenPos[IX,13]:=SkY;
  For Ix:=ScreenMinX to ScreenMaxX-1 do
    ScreenPos[IX,16]:=SkY;
  For Ix:=ScreenMinX to ScreenMaxX do
    ScreenPos[IX,19]:=Home;

{=======For CC ===========}
  ScreenPos[42,3]:=CCHex;
  ScreenPos[43,3]:=CCHex;
  ScreenPos[35,3]:=HB;
  ScreenPos[36,3]:=IB;
  ScreenPos[37,3]:=NB;
  ScreenPos[38,3]:=ZB;
  ScreenPos[39,3]:=VB;
  ScreenPos[40,3]:=CB;
  ScreenPos[48,3]:=HB;
  ScreenPos[51,3]:=IB;
  ScreenPos[54,3]:=NB;
  ScreenPos[57,3]:=ZB;
  ScreenPos[60,3]:=VB;
  ScreenPos[63,3]:=CB;
  For Ix:= 64 to ScreenMaxX-1 do
    ScreenPos[IX,3]:=SkY;
{========For A============}
  For Ix:=33 to 40 do
    ScreenPos[IX,4]:=ABin;
  ScreenPos[42,4]:=AHex;
  ScreenPos[43,4]:=AHex;
  ScreenPos[45,4]:=AChar;
  For Ix:= 46 to ScreenMaxX-1 do
    ScreenPos[IX,4]:=SkY;

{========For B============}
  For Ix:=33 to 40 do
    ScreenPos[IX,5]:=BBin;
  ScreenPos[42,5]:=BHex;
  ScreenPos[43,5]:=BHex;
  ScreenPos[45,5]:=BChar;
  For Ix:= 62 to ScreenMaxX-1 do
    ScreenPos[IX,5]:=SkY;
  ScreenPos[49,5]:=IRQB;
  ScreenPos[55,5]:=NMIB;
```

```
  ScreenPos[61,5]:=RESB;

{=======For X============}
  For Ix:=33 to 48 do
    ScreenPos[IX,6]:=XBin;
  ScreenPos[50,6]:=XHex;
  ScreenPos[51,6]:=XHex;
  ScreenPos[52,6]:=XHex;
  ScreenPos[53,6]:=XHex;

  ScreenPos[55,6]:=XChar;
  ScreenPos[56,6]:=XChar;
  For Ix:= 57 to ScreenMaxX-1 do
    ScreenPos[IX,6]:=SkY;
{======For SP=========}
  IY:=8;
  ScreenPos[33,IY]:=SPHex;
  ScreenPos[34,IY]:=SPHex;
  ScreenPos[35,IY]:=SPHex;
  ScreenPos[36,IY]:=SPHex;
  For Ix:=42 to 64 do
    ScreenPos[Ix,IY]:=SPMem;
  Ix:=44;
  While IX<=63 do
  begin
    ScreenPos[Ix,IY]:=SkX;
    IX:=IX+3;
  end;
  For IX:=66 to 73 do
    ScreenPos[Ix,IY]:=SPChar;
  Inc(IY);
  For Ix:=42 to 64 do
    ScreenPos[Ix,IY]:=SPMem;
  Ix:=44;
  While IX<=63 do
  begin
    ScreenPos[Ix,IY]:=SkX;
    IX:=IX+3;
  end;
  For IX:=66 to 73 do
    ScreenPos[Ix,IY]:=SPChar;
{======Window1===========}
  IY:=11;
  For Ix:=ScreenMinX to 42 do
    ScreenPos[IX,IY]:=Skx;

  ScreenPos[33,IY]:=AddHex1;
  ScreenPos[34,IY]:=AddHex1;
  ScreenPos[35,IY]:=AddHex1;
  ScreenPos[36,IY]:=AddHex1;
  For Ix:=42 to 64 do
    ScreenPos[Ix,IY]:=MemHex1;
  Ix:=44;
  While IX<=63 do
  begin
    ScreenPos[Ix,IY]:=SkX;
    IX:=IX+3;
  end;
  For IX:=66 to 73 do
    ScreenPos[Ix,IY]:=MemChar1;
  Inc(IY);
  For Ix:=ScreenMinX to 42 do
    ScreenPos[IX,IY]:=Skx;
  For Ix:=42 to 64 do
    ScreenPos[Ix,IY]:=MemHex1;
  Ix:=44;
```

```
  While IX<=63 do
  begin
    ScreenPos[Ix,IY]:=SkX;
    IX:=IX+3;
  end;
  For IX:=66 to 73 do
    ScreenPos[Ix,IY]:=MemChar1;
{======Window2============}
  IY:=14;
  For Ix:=ScreenMinX to 42 do
    ScreenPos[IX,IY]:=Skx;
  ScreenPos[33,IY]:=AddHex2;
  ScreenPos[34,IY]:=AddHex2;
  ScreenPos[35,IY]:=AddHex2;
  ScreenPos[36,IY]:=AddHex2;
  For Ix:=42 to 64 do
    ScreenPos[Ix,IY]:=MemHex2;
  Ix:=44;
  While IX<=63 do
  begin
    ScreenPos[Ix,IY]:=SkX;
    IX:=IX+3;
  end;
  For IX:=66 to 73 do
    ScreenPos[Ix,IY]:=MemChar2;
  Inc(IY);
  For Ix:=ScreenMinX to 42 do
    ScreenPos[IX,IY]:=Skx;
  For Ix:=42 to 64 do
    ScreenPos[Ix,IY]:=MemHex2;
  Ix:=44;
  While IX<=63 do
  begin
    ScreenPos[Ix,IY]:=SkX;
    IX:=IX+3;
  end;
  For IX:=66 to 73 do
    ScreenPos[Ix,IY]:=MemChar2;
{======Window3============}
  IY:=17;
  For Ix:=ScreenMinX to 42 do
    ScreenPos[IX,IY]:=Skx;
  ScreenPos[33,IY]:=AddHex3;
  ScreenPos[34,IY]:=AddHex3;
  ScreenPos[35,IY]:=AddHex3;
  ScreenPos[36,IY]:=AddHex3;
  For Ix:=42 to 64 do
    ScreenPos[Ix,IY]:=MemHex3;
  Ix:=44;
  While IX<=63 do
  begin
    ScreenPos[Ix,IY]:=SkX;
    IX:=IX+3;
  end;
  For IX:=66 to 73 do
    ScreenPos[Ix,IY]:=MemChar3;
  Inc(IY);
  For Ix:=ScreenMinX to 42 do
    ScreenPos[IX,IY]:=Skx;
  For Ix:=42 to 64 do
    ScreenPos[Ix,IY]:=MemHex3;
  Ix:=44;
  While IX<=63 do
  begin
    ScreenPos[Ix,IY]:=SkX;
```

```
    IX:=IX+3;
  end;
  For IX:=66 to 73 do
    ScreenPos[Ix,IY]:=MemChar3;

end;
```

9.3. File CHANGE.INC

```
{-------------------------------------------------}
(*Procedure Delays (sec, sec100 : word);
Var
 Hr, HrN, Mn, MnN, Se, Se100, SeS, Se100S : Word;
begin
  GetTime(Hr,Mn,Se,Se100);
  SeS:=Se+sec;
  If Se100>(99-sec100) then
  begin
    Se100S:=Sec100-(99-Se100);
    Sec:=Sec+1;
  end
  else
    Se100S:=Se100+Sec100;
  If Se>(59-Sec) then
  begin
    SeS:=Sec-(59-Se);
    Mn:=Mn+1;
  end
  else
    SeS:=Se+Sec;
  If Mn>59 then begin Mn:=0; Hr:=Hr+1; end;
    GetTime(HrN,MnN,Se,Se100);
  While    (Speed<>Fast) and
           (Not ((HrN>Hr) or
               ((HrN=Hr) and (MnN>Mn)) or
               ((HrN=Hr) and (MnN=Mn) and (Se>=SeS) and (Se100>=Se100S)))) do
  begin
    GetTime(HrN,MnN,Se,Se100);
    If KeyPressed then GetCommand;
  end;
end;*)
{-------------------------------------------------}
Procedure Delays (sec, sec100 : word);
begin
  Sec:=Sec*100+Sec100*10;
  Sec100:=1;
  Repeat
    Delay(10);
    If KeyPressed then GetCommand;
    Inc(Sec100);
  Until (Sec100>=Sec) or Exit;
end;
{-------------------------------------------------}
Procedure ChangeFromBinToHex ( Bin : BinFormat8;
                               Var   Hex : HexFormat2);
Var
  II, Num          : Integer;
begin
  Num:=Bin[7]*8+Bin[6]*4+Bin[5]*2+Bin[4]*1;
  For II:=1 to 2 do
  begin
    If Num<10 then
       Hex[II]:=Chr(48+Num)
    else
       Hex[II]:=Chr(55+Num);
    Num:=Bin[3]*8+Bin[2]*4+Bin[1]*2+Bin[0]*1;
  end;
end;
{-------------------------------------------------}
```

```
{$F+}
Procedure ChangeDecToBin( Num     : Integer;
                      Var  Bin     : BinFormat8);
begin

ASM
        MOV AX,WORD PTR [NUM]
        LES DI,DWORD PTR [BIN]
        MOV CX,8
@MOREBIT:
        XOR BL,BL
        SHR AL,1
        RCL BL,1
        MOV [ES:DI],BL
        INC DI
        LOOP @MOREBIT

end;

end;
{$F-}
{-------------------------------------------------}
{$F+}
Procedure ChangeFromBinToDec( Bin : BinForMat8;
              Var  Num : Integer       );
begin
{ Num:=Bin[7]*128+Bin[6]*64+Bin[5]*32+Bin[4]*16+Bin[3]*8+Bin[2]*4+Bin[1]+Bin
[1]
      +Bin[0];}

ASM
  PUSH DS
  LEA  BX, {BIN}
  MOV  AX,SS
  MOV  DS,AX
  XOR  AX,AX
  MOV  CX,8
@TOP:
  MOV  DX,[BX]
  SHR  DL,1
  RCR  AL,1
  INC  BX
  LOOP @TOP
  XOR  AH,AH
  POP  DS
  LES  DI,DWORD PTR [NUM]
  MOV  [ES:DI],AX

END;
end;
{$F-}
{-------------------------------------------------}
Function GetDecFromHex(Hex  : HexFormat2) : Integer;
Var
  Sum, I        :  Integer;
begin
  Sum:=0;
  For I:=1 to 2 do
  begin
    If Hex[I] in ['0'..'9'] then
       Sum:=Sum+Ord(Hex[I])-48
    else
       Sum:=Sum+Ord(Hex[I])-55;
    If I=1 then Sum:=Sum*16;
  end;
  GetDecFromHex:=Sum;
```

```
end;
{-------------------------------------------------}

{$F+}
Procedure ChangeDecToHex( Var Hex : HexFormat2;
                              Num : Integer     );
begin
ASM
        MOV  AX,WORD PTR [Num]
        LES  DI,DWORD PTR [Hex]
        push ax
        and  AL,0F0H
        MOV  CX,4
        SHR  AL,CL
        CMP  AL,9
        JG   @DOAF
        ADD  AL,30H
        JMP  @NEXT
      @DOAF:
        ADD  AL,'A'-10
      @NEXT:
        MOV  BX,AX
        POP  AX
        and  AL,0FH
        CMP  AL,9
        JG   @DOAF2
        ADD  AL,30H
        JMP  @NEXT2
 @DOAF2:
        ADD  AL,'A'-10
 @NEXT2:
        MOV  [ES:DI],BL
        MOV  [ES:DI+1],AL
End; {Asm}
end;
{$F-}
{-------------------------------------------------}
Procedure ChangeDecTo2Hex(Var Hex1, Hex2 : HexFormat2;
                              Decimal    : Integer     );
{ Change From Decimal Value to 2 Bytes Hex. Format }
Var
  Num      : Byte;
  II       : Integer;
begin
  Num:=Hi(Decimal);
  II:=Num;
  ChangeDecToHex(Hex1,II);
  II:=Lo(Decimal);
  ChangeDecToHex(Hex2,II);
end;
{-------------------------------------------------}
Procedure ChangeFromDec (Var Reg  : RegisterType);
{ Change From Decimal Value in a register  to Binary and Hex. Format in the
same }
{ register
}
Var
  Num      : Byte;
  II       : Integer;
begin
  With Reg do
  begin
    IF NumByte=2 then
    begin
      Num:=Hi(DecValue);
      II:=Num;
```

```
      ChangeDecToHex(HigHByteH,II);
      ChangeDecToBin(Num,HighByteB);
      Num:=Lo(DecValue);
    end
    else
    begin
      Num:=Lo(DecValue);
      DecValue:=Num;
    end;
    II:=Num;
    ChangeDecToHex(LowerByteH,II);
    ChangeDecToBin(Num,LowerByteB);
  end;
end;
{-----------------------------------------------------}
Function LowerNibble (Num : integer):integer;
begin
   LowerNibble:=Lo(Num) and $0F;
end;
{-----------------------------------------------------}
Procedure AdjustZandN(AResult : Integer);
begin
  If AResult=0 then
   begin
    CC[Z]:=1;
    CC[N]:=0;
   end
  else
   begin
    CC[Z]:=0;
    if AResult>PosNum then CC[N]:=1 else CC[N]:=0;
   end;
end;
{-----------------------------------------------------}
Procedure DoAdd (Var Result, AResult : Integer; Num1, Num2 : Integer);
Var
  Low1, Low2       : Integer;
begin
  Result:=Num1+Num2;
  If Result>255 then
  begin
    AResult:=Result-256;
    CC[C]:=1;
  end
  else
  begin
    AResult := Result;
    CC[C]:=0;
  end;
{Adjust Z and N}
  AdjustZAndN(AResult);
{Check OverFlow}
  If ((Num1>PosNum) and (Num2>PosNum) and (CC[N]=0))
   or ((Num1<=PosNum) and (Num2<=PosNum) and (CC[N]=1))  then CC[V]:=1
  else  CC[V]:=0;
{Check Half Carry}
  Low1:=LowerNibble(Num1);
  Low2:=LowerNibble(Num2);
  If (Low1+Low2)>15 then CC[H]:=1 else CC[H]:=0;
end;
{-----------------------------------------------------}
```

9.4. File GETADDR.INC

```
Procedure GetAddress (Var Num       : Integer;
                      Var Hex1, Hex2: HexFormat2;
                          Length    : Integer  );

Var
  Code, II, III : Integer;
  AHex : array [1..4] of Char;
  TempHex : array[1..4] of Char;
  Fin, HexCh: Boolean;
  Ch        : Char;
begin
      Fin:=False;
      TempHex[1]:='0';
      TempHex[2]:='0';
      TempHex[3]:='0';
      TempHex[4]:='0';
      Code:=1;
      While Not Fin do
      begin
        HexCh:=False;
        Repeat
          CH:=ReadKey;
          Case Ch of
            '0'..'9', 'a'..'f', 'A'..'F' :
                begin
                  If Ch in ['a'..'f'] then
                     Ch:=Chr(Ord(ch)-32);
                  If Code<=Length then
                  begin
                    Write(Ch);
                    AHex[Code]:=Ch;
                    HexCh:=True;
                    Code:=Code+1;
                  end
                  else
                  begin
                    Write(#7);
                  end;
                end;
              #10,#13:
                begin
                  HexCh:=True;
                  Fin:=True;
                  Case Code of
                    1:begin
                        Num:=0;
                        Hex1[1]:=#13;
                      end;
                    2..5: begin
                            III:=4;
                            For II:=Code-1 downto 1 do
                            begin
                              TempHex[III]:=AHex[II];
                              III:=III-1;
                            end;
                            Hex1[1]:=TempHex[1];
                            Hex1[2]:=TempHex[2];
                            Hex2[1]:=TempHex[3];
                            Hex2[2]:=TempHex[4];
                            Num:=GetDecFromHex(Hex1)*256+GetDecFromHex(Hex2);
```

```
                        end;
                  end;
              end;
          #8     :
              begin
                If Code>1 then
                begin
                  Code:=Code-1;
                  Write(#8,' ',#8);
                end
                else
                 write(#7);
              end;
          #27    : begin
                     Hex1[1]:=#27;
                     Fin:=True;
                     HexCh:=True;
                   end;
          else  write(#7);
        end;{Case}
      Until HexCh;
    end;
end;
{-----------------------------------------}
Procedure GetNum     (Var Num       : Integer;
                      Var ChC       : Char;
                          Length    : Integer  );

Var
  Code, II, III : Integer;
  AHex : array [1..4] of Char;
  TempHex : array[1..4] of Char;
  Fin, DecCh: Boolean;
  Ch        : Char;
begin
     Fin:=False;
     TempHex[1]:='0';
     TempHex[2]:='0';
     TempHex[3]:='0';
     TempHex[4]:='0';
     ChC:='0';
     Code:=1;
     While Not Fin do
     begin
       DecCh:=False;
       Repeat
         CH:=ReadKey;
         Case Ch of
           '0'..'9' :
              begin
                If Code<=Length then
                begin
                  Write(Ch);
                  AHex[Code]:=Ch;
                  DecCh:=True;
                  Code:=Code+1;
                end
                else
                begin
                  Write(#7);
                end;
              end;
           #10,#13:
              begin
                DecCh:=True;
                Fin:=True;
```

```
                Case Code of
                  1:begin
                      Num:=0;
                      ChC:=#13;
                    end;
                  2..5: begin
                          III:=4;
                          For II:=Code-1 downto 1 do
                          begin
                            TempHex[III]:=AHex[II];
                            III:=III-1;
                          end;
                          Num:=0;
                          For II:=1 to 4 do
                          begin
                            Num:=Num*10+Ord(TempHex[II])-48;
                          end;
                        end;
                end;
              end;
          #8    :
              begin
                If Code>1 then
                begin
                  Code:=Code-1;
                  Write(#8,' ',#8);
                end
                else
                 write(#7);
              end;
          #27   : begin
                    Chc:=#27;
                    Fin:=True;
                    DecCh:=True;
                  end;
          else  write(#7);
        end;{Case}
      Until DecCh;
    end;
end;
```

9.5. File LOAD.INC

```
Procedure LoadProgramS;
Var
  LProg           : Text;
  TempText, TT2 : Integer;
  Ch              : Char;
  Hex             : HexFormat2;
  Num, Address,
  Code,II         : Integer;
  Fin, Start, WindowOPen      : Boolean;
begin
  TT2:=TextAttr;
  DoRecord:=False;
  If Newprog then
  begin
  If ParamCount<1 then
  begin
{     PTWOpen(ErrorWin);}
    ShowWin(ErrorWin);

    WindowOpen:=True;
  end
  else
  begin
    FName:=ParamStr(1);
    WindowOpen:=False;
  end;
  Repeat
    If WindowOpen then
    begin
      GotoXy(1,1); ClrEol;
      GotoXy(1,1);
      Write('Enter Load File: ');
      Readln(FName);
    end;
{$I-}
    Assign(LProg,FName);
    Reset(LProg);
    Close(LProg);
    Fin:=(IOResult=0) and (FName<>'');
    If Not Fin then
    begin
      If Not WindowOpen then
      begin
{        PTWOpen(ErrorWin);}
        ShowWin(ErrorWin);
        WindowOpen:=True;
      end;
      TempText:=TextAttr;
      GotoXy(3,3);
      TextAttr:=TextInv;
      Beep(5);
      Write(' FILE NOT FOUND '); Delay(2000);
      TextAttr:=TempText;
      GotoXy(3,3); ClrEol;
    end
    else
    begin
      Assign(LProg,FName);
      Reset(LProg);
      Read(LProg,Hex[1],Hex[2]);
```

```
    If (Hex[1]='S') and (Hex[2]='1') then
    begin
      NewProg:=False;
      If WindowOpen then CloseWin(ErrorWin);
    end
    else
    begin
      Close(LProg);
      If Not WindowOpen then
      begin
        ShowWin(ErrorWin);
        WindowOpen:=True;
      end;
      TempText:=TextAttr;
      GotoXy(3,3);  TextAttr:=TextInv;
      Beep(5);
      Write(' THIS IS NOT A LOADING FILE ');  Delay(2000);
      TextAttr:=TempText;
      GotoXy(3,3); ClrEol;
      Fin:=False;
    end;
  end;
 Until Fin
 end
{$I+}
else
 Assign(LProg,FName);

 TextAttr:=TT2;
 GotoXY(XShowF, YShowF);
 Write('Running Program ',FName);

 Reset(LProg);
 Fin:=False;
 Start:=True;
 Repeat
   Read(LProg,Hex[1],Hex[2]);
   If Hex[2]='1' then
   begin
     Read(LProg,Hex[1],Hex[2]);
     Num:=GetDecFromHex(Hex)-3;
     Read(LProg,Hex[1],Hex[2]);
     Address:=GetDecFromHex(Hex);
     Read(LProg,Hex[1],Hex[2]);
     Address:=Address*256+GetDecFromHex(Hex);
     If Start then
     begin
       TopAddress:=Address;
       StartAddress:=Address;
     end;
     Start:=False;
     For II:=1 to Num do
     begin
       Read(LProg,Hex[1],Hex[2]);
       Code:=GetDecFromHex(Hex);
       StoreMemory(Address,Code);
       Address:=Address+1;
     end;
     Readln(LProg);
   end
   else
     Fin:=True;
 Until Fin;
 Close(LProg);
 PC.DecValue:=StartAddress;
 ShowPCAddr:=StartAddress;
```

```
  ChangeFromDec(PC);
  MakeInstList;
  MakeNewAddressLink;
  DoShowInstList;
  II:=1;
  Fin:=False;

  DoRecord:=True;

end;
```

9.6. File SPECIAL.INC

```
Procedure SpecialKeys;
Var
  Ch      : Char;
  Fin, Found,
  ClearBPMark     : Boolean;
  Temp, II, TempInv,
  CPos, Start, Last,
  Temp2, LineCount,
  Temp3           : Integer;
  Bk              : BPType;
  Hex1, Hex2      : HexFormat2;
  Num, Num2       : Integer;
  WReg            : RegValueType;
  CCBit           : BitofCCType;
  T               : AddressLinkPointerType;
begin
TempINV:=TextInV;
Ch:=ReadKey;
Fin:=True;
ClearBPMark:=True;
Repeat
  Case Ch of
    AltKF10 : begin
                For II:=0 to 1999 do
                begin
                  With Screen^[II] do
                  begin
                    If AtCh=TextNorm then
                       AtCh:=15
                    else
                       Atch:=112;
                  end;
                end;
                TextNorm:=15;
                Tempinv:=112;
                TextInv:=112;
              end;
    KUP:      begin
                If CScrY>ScreenMinY then
                begin
                  OScrX:=CScrX;
                  OScrY:=CScrY;
                  Dec(CScrY); SkipCur(Ch);
                  DoBlink;
                end;
              end;{UP}
    KLeft:
         begin
            OScrX:=CScrX;
            OScrY:=CScrY;
          If CScrX>ScreenMinX then
          begin
            Dec(CScrX);
            SkipCur(ch);
            DoBlink;
          end
          else If CScrY>ScreenMinY then
               begin
                 Dec(CScrY);
                 If CScrY=ScreenMinY then
```

```
                CScrX:=ScreenMinX+2
              else
                CScrX:=ScreenMinX;
                DoBlink;
            end;

    end;{Left}
KRight:
    begin
        OScrX:=CScrX;
        OScrY:=CScrY;
      If CScrX<ScreenMaxX then
      begin
        Inc(CScrX);
        SkipCur(ch);
        DoBlink;
      end;
    end;{Right}
KDown:
     begin
      OScrX:=CScrX;
      OScrY:=CScrY;
      If CScrY<ScreenMaxY then
      begin
        Inc(CScrY);
        SkipCur(ch);
        DoBlink;
      end;
    end;{Down}
KF10:
    begin
      Case Speed of
        Fast: begin
                Speed:=Mid;
                GotoXy(71,3); Write('Med ');
              end;
        Mid : begin
                Speed:=Slow;
                GotoXy(71,3); Write('Slow');
              end;
        Slow : begin
                 Speed:=Fast;
                 GotoXy(71,3); Write('Fast');
               end;
       end;          {F10}
     end;
 ShfKF8  :
          begin
            Sound(880);
            Delay(200);
            NoSound;
            IRQF:= Not IRQF;
            GotoXy(49,5);
            If IRQF then write('0') else Write('1');
          end;{^F8 IRQ}
 ShfKF9  :
          begin
            Sound(987);
            Delay(200);
            NoSound;
            GotoXy(55,5);
            If NMILV=1 then
            begin
              NMILV:=0;
              NMIF:=True;
              Write('0');
```

```
          end
          else
          begin
             NMILV:=1;
             NMIF:=False;
             Write('1');
           end;
         end; {^F9 NMI}

ShfKF10 :
         begin
           Sound(880);
           Delay(200);
           Sound(987);
           Delay(200);
           Sound(1046);
           Delay(200);
           NoSound;
           GotoXy(61,5); write('0');
           RESF:=True;
         end; {^F10 RES}
KF2     : begin
           If BPOn then
           begin
             Temp:=BreakMarkY*80-80;
             With Screen^[Temp] do
             begin
               If Ch in ['>','฿'] then Ch:=' ';
               Atch:=Atch and  127;
               BPOn:=False;
             end;
           end;
            If RuningMode<>SingleStep then
            begin
              RuningMode:=SingleStep;
              GotoXy(65,3); Write('Step');
            end;
            TakeCareBP;
            DoRun:=True;
         end; {F2 SingleStep}
ShfKF2  : begin
           RuningMode:=SingleStep;
           If BPOn then
           begin
             Temp:=BreakMarkY*80-80;
             With Screen^[Temp] do
             begin
               If Ch='>' then Ch:=' ';
               Atch:=Atch and  127;
               BPOn:=False;
             end;
           end;
           DoRun:=False;
           NextHist:=Hist[NextHist].Past;
           If Hist[NextHist].empty then
              Beep(4)
           else
           begin
             With Hist[NextHist] do
             begin
               Empty:=True;
               Accum:=AccumH;
               CC:=CCH;
               X:=XH;
               PC:=PCH;
               SP:=SPH;
```

```
                If ChangeMem then
                begin
                  DoRecord:=False;
                  StoreMemory(AddressH,Content);
                  DoRecord:=True;
                  ChangeMem:=False;
                end;
              end;
              DisPlay;
            end;
          end;{^F2 backStep}
KF1     : begin
            If BPOn then
            begin
              Temp:=BreakMarkY*80-80;
              With Screen^[Temp] do
              begin
                If Ch='>' then Ch:=' ';
                Atch:=Atch and  127;
                BPOn:=False;
              end;
            end;
            If RuningMode<>Running then
             begin
               RuningMode:=Running;
               GotoXy(65,3); Write('Run ');
             end;
            TakeCareBP;
            DoRun:=True;
          end; {F1 running}
AltG    : TakeCareBP;
AltT    : ShowSplashScreen;
KF5     : begin
            Watch:= Not Watch;
            GotoXy(77,3);
            If Watch then
               write('ON ')
            else
               write('OFF');

          end; {F5 toggle watch}
ShfKF5 : begin
             ShowWin(BreakPointWin);
             Window(28,15,76,19);
             GotoXy(1,1);
             Write('ฉ' );
             For II:=29 to 75 do
               write('ฤ');
             write('ฉ');
             Window(30,16,74,19);
             Writeln(' Use ',#24,#25,' to move the highlight.');
             Writeln(' Hit <CR> to select.');
             Writeln(' Hit ESC to close the window.');
             Window(30,2,74,14);
             Write(' Registers '); GotoXy(14,1);
             Writeln(' Watch Value');
             Writeln('   PC ');
             Writeln('   SP ');
             Writeln('   X  ');
             Writeln('   A  ');
             Writeln('   B  ');
             Writeln('   CC : each bit can be set seperately');
             Writeln('   H  ');
             Writeln('   I  ');
             Writeln('   N  ');
             Writeln('   Z  ');
```

```
Writeln('   V  ');
Write('   C  ');
CPos:=2;
Temp:=TextAttr;
TextInv:=TextNorm;
Repeat
  II:=2;
  For WReg:=WPC  to WX do
  begin
    GotoXy(17,II);
    ChangeDecto2Hex(Hex1,Hex2,WatchValue[Wreg].DecValue);
    Write('$',Hex1,Hex2);
    GotoXy(29,II);
    If WatchValue[Wreg].On then
      write('ON ')
    else
      Write('OFF');
    InC(II);
  end;
  For WReg:=WA to WB do
  begin
    GotoXy(17,II);
    ChangeDectoHex(Hex1,WatchValue[Wreg].DecValue);
    Write('$',Hex1);
    GotoXy(29,II);
    If WatchValue[Wreg].On then
       write('ON ')
    else
       Write('OFF');
    InC(II);
  end;
  InC(II);
  With WatchValue[WCC] do
  begin
    For CCBit:=H to C do
    begin
      GotoXy(18,II);
      Write(Value[CCBit]);
      GotoXy(29,II);
      If BitOn[CCBit] then
        write('ON ')
      else
        Write('OFF');
      InC(II);
    end;
  end;
  Window(30,2,74,14);
  Temp2:=(Hi(WindMin)+CPos-1)*80;
  Num:=Temp2+Lo(WindMin);
  Num2:=Num+32;
  For Temp2:=Num to Num2 do
     Screen^[Temp2].AtcH:=TextInv;
  GotoXY(33,13);
  Ch:=Readkey;
  Case Ch of
    #0 : begin
           Ch:=ReadKey;
           Case Ch of
             KUP  : begin
                      Temp2:=(Hi(WindMin)+CPos-1)*80;
                      Num:=Temp2+Lo(WindMin);
                      Num2:=Num+32;
                      For Temp2:=Num to Num2 do
                         Screen^[Temp2].AtcH:=Temp;
                      Case CPos of
                        3,4,5,6,9,10,11,12,13 :
```

```
                                    begin
                                      Dec(CPOS);
                                      Dec(Num,80);
                                      Dec(Num2,80);
                                    end;
                          2       : begin
                                      CPos:=13;
                                      Inc(Num,880);
                                      Inc(Num2,880);
                                    end;
                          8       : begin
                                      CPos:=6;
                                      Dec(Num,160);
                                      Dec(Num2,160);
                                    end;
                        end;
                        For Temp2:=Num to Num2 do
                            Screen^[Temp2].AtcH:=TextINV;

                        end;
                KDown: begin
                         Temp2:=(Hi(WindMin)+CPos-1)*80;
                         Num:=Temp2+Lo(WindMin);
                         Num2:=Num+32;
                         For Temp2:=Num to Num2 do
                            Screen^[Temp2].AtcH:=Temp;
                         Case CPos of
                           2,3,4,5,8,9,10,11,12 :
                                       begin
                                        Inc(CPOS);
                                        Inc(Num,80);
                                        Inc(Num2,80);
                                       end;
                           13        : begin
                                        CPos:=2;
                                        Dec(Num,880);
                                        Dec(Num2,880);
                                       end;
                           6         : begin
                                        CPos:=8;
                                        Inc(Num,160);
                                        Inc(Num2,160);
                                       end;
                        end;
                        For Temp2:=Num to Num2 do
                           Screen^[Temp2].AtcH:=TextINV;

                        end;
              Else
                Beep(3);
            end;{#0}
            GotoXY(33,13);
      end;   {#0}
#27:;
#10,#13: begin
           Window(30,16,74,19);
           GotoXY(1,1);   ClrEol;
           GotoXy(1,2);   ClrEol;
           GotoXy(1,2);
           Writeln(' Hit ESC to close the window.');
           GotoXy(1,3);   ClrEol;
           GotoXy(1,1);
           Write(' Enter the watch value for ');
           If CPos in [2,3,4,5,6] then
           begin
             Case CPOS of
```

```
                    2: Begin
                        Write(' PC .');
                        GotoXy(1,4);
                        Write(' PC ');
                        WReg:=WPC;
                       end;
                    3: Begin
                        Write(' SP .');
                        GotoXy(1,4);
                        Write(' SP ');
                        WReg:=WSP;
                       end;
                    4: Begin
                        Write(' X  .');
                        GotoXy(1,4);
                        Write(' X  ');
                        WReg:=WX;
                       end;
                    5: Begin
                        Write(' A  .');
                        GotoXy(1,4);
                        Write(' A  ');
                        WReg:=WA;
                       end;
                    6: Begin
                        Write(' B  .');
                        GotoXy(1,4);
                        Write(' B  ');
                        WReg:=WB;
                       end;
                  end;
                  GotoXy(18,4);Write('$');
                  If WReg in [WPC,WSP,WX] then
                     GetAddress(Num, Hex1,Hex2,4)
                  else
                     GetAddress(Num, Hex1,Hex2,2);
                  Case Hex1[1] of
                    #13: begin
                           If WReg in [WPC,WSP,WX] then
                           begin

ChangeDecto2Hex(Hex1,Hex2,WatchValue[Wreg].DecValue);
                             Write(Hex1,Hex2);
                           end
                           else
                           begin
                             ChangeDectoHex(Hex1,WatchValue
[WReg].DecValue);

                             Write(Hex1);
                           end;
                         end;
                    #27: Ch:=#27;
                    else
                         begin
                           WatchValue[WReg].DecValue:=Num;
                         end;
                  end;
                  If Ch<>#27 then
                  begin
                    GotoXy(1,1); ClrEol;
                    Write(' Type O for On;  type F for
Off.');

                    GotoXy(29,4);
                    Repeat
                      Ch:=ReadKey;
```

```
                If Not (Ch in [#13,#27,'o','O','f','F']
) then
                  begin   beep(4); beep(1);  end;
                Until Ch in [#13,#27,'o','O','f','F'];
                If Not (Ch in[#27,#13]) then
                  WatchValue[Wreg].On:=Ch in ['o','O'];
              end;

            end
            else
            begin
              WReg:=WCC;
              Case CPOS of
                8: Begin
                     Write(' CC bit H .');
                     GotoXy(1,4);
                     Write(' CC bit H  ');
                     CCBit:=H;
                   end;
                9: Begin
                     Write(' CC bit I .');
                     GotoXy(1,4);
                     Write(' CC Bit I ');
                     CCBit:=I;
                   end;
               10: Begin
                     Write(' CC Bit N .');
                     GotoXy(1,4);
                     Write(' CC Bit N ');
                     CCBit:=N;
                   end;
               11: Begin
                     Write(' CC bit Z .');
                     GotoXy(1,4);
                     Write(' CC bit Z ');
                     CCBit:=Z;
                   end;
               12: Begin
                     Write(' CC bit V .');
                     GotoXy(1,4);
                     Write(' CC bit V ');
                     CCBit:=V;
                   end;
               13: Begin
                     Write(' CC bit C .');
                     GotoXy(1,4);
                     Write(' CC bit C ');
                     CCBit:=C;
                   end;
              end;
              GotoXy(18,4);
              Repeat
                Ch:=ReadKey;
                If Not (Ch in [#13,#27,'0','1']) then
                begin   beep(4); beep(1);  end;
              Until Ch in [#13,#27,'0','1'];
              Case Ch of
                '0','1': begin
                           WatchValue[WCC].Value
[CCBit]:=Ord(Ch)-48;
                           Write(ch);
                         end;
                #27:;
                #13:  Write(WatchValue[WCC].Value
[CCBit]:1);
              end;
```

```
                                If Ch<>#27 then
                                begin
                                  GotoXy(1,1); ClrEol;
                                  Write(' Type O for On;  type F for
Off.');
                                  GotoXy(29,4);
                                  Repeat
                                    Ch:=ReadKey;
                                    If Not (Ch in [#13,#27,'o','O','f','F']
) then
                                      begin   beep(4); beep(1);  end;
                                  Until Ch in [#13,#27,'o','O','f','F'];
                                  If Not (Ch in[#27,#13]) then
                                     WatchValue[WCC].BitOn[CCBit]:=Ch in
['o','O'];
                                end;
                              end;
                              GotoXY(1,1);   ClrEol;
                              GotoXy(1,2);   ClrEol;
                              GotoXy(1,3);   ClrEol;
                              GotoXy(1,4);   ClrEol;
                              GotoXy(1,1);
                              Writeln(' Use ',#24,#25,' to move the
highlight.');
                              Writeln(' Hit <CR> to select.');
                              Writeln(' Hit ESC to close the window.');
                              Window(30,2,74,14);
                              GotoXy(33,13);
                            end;
                      Else
                          Beep(3);
                    end;{Case}
                  Until Ch=#27;
                  CloseWin(BreakPointWin);

                  TextAttr:=TextNorm;

              end;{^F5 set watch values}
    KF3     : begin
                ClearBPMark:=False;
                If BreakMarkY<=2 then
                begin
                   Beep(1);
                   BreakMarkY:=2;
                   If Not BPOn then
                   begin
                    With Screen^[81] do
                    begin
                      Ch:='>';
                      Atch:=AtcH or 128;
                    end;
                    BPOn:=True;
                   end;
                end
                else
                begin
                  Temp:=BreakMarkY*80-79;
                  If BPOn then
                  With Screen^[Temp] do
                  begin
                    Ch:=' ';
                    Atch:=Atch and  127;
                  end;
                  Temp:=Temp-80;
                  With Screen^[Temp] do
                  begin
```

```
              Ch:='>';
              Atch:=AtcH or 128;
            end;
            BpOn:=True;
            Dec(BreakMarkY);
          end;
        end;{F3 BP Up}
KF4     : begin
          ClearBPMark:=False;
          If BreakMarkY>=MaxInstL then
          begin
            Beep(1);
            BreakMarkY:=MaxInstL;
            If Not BPOn then
            begin
              With Screen^[(MaxInstL-1)*80+1] do
              begin
                Ch:='>';
                Atch:=AtcH or 128;
              end;
              BPOn:=True;
            end;
          end
          else
          begin
            Temp:=BreakMarkY*80-79;
            If BPOn then
            With Screen^[Temp] do
            begin
              Ch:=' ';
              Atch:=Atch and  127;
            end;
            Temp:=Temp+80;
            With Screen^[Temp] do
            begin
              Ch:='>';
              Atch:=AtcH or 128;
            end;
            BpOn:=True;
            Inc(BreakMarkY);
          end;
        end;{F4 Bp Down}
 ShfKF3, ShfKF4 :
        begin
          Temp:=0;
          Repeat
            Inc(Temp);
          Until (BPUse[Temp].BP=NoBp)
                or (BPUse[Temp].Address=Show[BreakMarkY].Addr)
                or (Temp>=MaxBp);
          If (BPUse[Temp].BP<>NoBP)
             and (BPUse[Temp].Address<>Show[BreakMarkY].Addr) then
          begin
            ShowWin(ERRORWin);
            GotoXY(5,2); Write('BREAK POINT FULL');
            Beep(2);
            Beep(3);
            Delay(800);
            {PtwClose;}
            CloseWin(ERRORWin);
            TextAttr:=TextNorm;
          end
          else
          begin
            With BPUse[Temp] do
            begin
```

```
                If Ch=ShfKF3 then
                   BP:=StrickBp
                else
                   BP:=NonStrickBp;
                Address:=Show[BreakMarkY].Addr;
                Show[BreakMarkY].BkPoint:=BPChar[BP];
                Temp:=BreakMarkY*80-80;
                With Screen^[Temp] do
                begin
                  Ch:=BPChar[BP];
                  AtCh:=AtCh and 127;
                end;

              end;
              TextAttr:=TextNorm;
            end;
          end;{^F3 Stricky BP, ^F4 non Stricky BP}

    AltM  : begin
{                MKwin(1,1,35,10,1,black,white);}
                 Set_Cursor(OLDCH,OLDCL);
               ShowWin(ChangeMemoryWin);
               Fin:=False;
               Repeat
                 GotoXy(1,1); Write(' Enter Memory Address : $');
                 GetAddress(Num,Hex1,Hex2,4);
                 GotoXy(1,2);
                 Fin:=Hex1[1] in [#27, #13, #10];
                 While Not (Hex1[1] in [#27, #13, #10]) do
                 begin
                   ChangeDecto2Hex(Hex1,Hex2,Num);
                   Write(' $',Hex1,Hex2,' :  $  ');
                   GetMemory(Num,Num2);
                   Dec(Num);
                   ChangeDecToHex(Hex1,Num2);
                   Write(Hex1,'  $');
                   GetAddress(II,Hex1,Hex2,2);
                   Writeln;
                   If not (Hex1[1] in [#27, #10, #13]) then
                   begin
                      DoRecord:=False;
                      StoreMemory(Num,II);
                      DoRecord:=True;
                      Inc(Num);
                   end
                   else if Hex1[1]=#27 then Fin:=True;
                 end;
                 If Not Fin then ClrScr;
               Until Fin;
{                RmWin;                          }
               CloseLastWin;
               If ProgMFlag then
               begin
                 MakeInstList;
                 DoShowInstList;
                 DisPlay;
                 ProgMFlag:=False;
               end
               else
                 DisPlay;
          Set_Cursor(36,0);
      end; {Alt-M : Memory}

    AltL    : begin
                NewProg:=True;
                LoadProgramS;
```

```
            end;{Alt-L: Load}
    AltR    : begin
                ClearMemory;
                ResetCPU;
                ResetVariable;
                Window(MinWinX,MinWinY,MaxWinX,MaxWinY);
                ClrScr;
                CWinX:=1; CWinY:=1;
                Window(1,1,80,25);
                Release(HeadAddressLink);
{                CurrentAddressLink:=HeadAddressLink^.Next;
                While CurrentAddressLink<>Nil do
                begin
                  T:= CurrentAddressLink;
                  CurrentAddressLink:=CurrentAddressLink^.Next;
                  Dispose(t);
                end;
}
                New(HeadAddressLink);
                Mark(HeadAddressLink);
                CurrentAddressLink:=HeadAddressLink;
                With HeadAddressLink^ do
                begin
                  Next:=Nil;
                  Back:=Nil;
                end;
                MakeInstList;
                LoadProgramS;
                Display;
                ForgetSmallScreen:=True;
              end;
    AltB    : begin  {break Point Alt-B}
                ShowWin(BreakPointWin);
                Window(28,15,76,19);
                GotoXy(1,1);
                Write('ๅ');
                For II:=29 to 75 do
                write('ๆ');
                write('ก');
                Window(30,16,74,19);
                Writeln(' Use ',#24,#25,' to move the highlight.');
                Writeln(' Hit <CR> to select.');
                Writeln(' Hit ESC to close the window.');
                CPos:=1;
                Start:=1;
                Last:=1;
                Temp:=TextAttr;
                TextInv:=TextNorm;
                Window(30,2,74,14);
                Writeln('          Break Points Set Up');
                Repeat
                  Window(30,3,74,14);
                  TextAttr:=Temp;
                  ClrEol;
                  II:=Start;
                  LineCount:=3;
                  While (II<=MaxBP) and (LineCount<=13) do
                  begin
                    With BPUse[II] do
                      If BP<>NoBP then
                      begin
                        ChangeDecto2Hex(Hex1,Hex2,Address);
                        Write(' BreakPoint #',II:2,' at $',Hex1,Hex2);
                        Case BP of
                          StrickBp: Writeln(' Type: Sticky;    ');
                          NonStrickBp: Writeln(' Type: Non-Sticky;');
```

```
                    end;
                  end
                  else
                    Writeln(' BreakPoint #',II:2,' at $xxxx No
Breakpoint;      ');
                   Inc(II);
                   InC(LineCount);
                end;
                Last:=II;
                If Last<MaxBP then
                begin
                  GotoXy(27,12);
                  TextAttr:=TextInv;
                  Write('[MORE]');
                  TextAttr:=Temp;
                end;
                Temp2:=(Hi(WindMin)+CPos-1)*80;
                Num:=Temp2+Lo(WindMin);
                Num2:=Num+42;
                For Temp2:=Num to Num2 do
                    Screen^[Temp2].AtcH:=TextInv;
                GotoXY(33,12);
                Ch:=Readkey;
                Case Ch of
                  #0 : begin
                         Ch:=ReadKey;
                         Case Ch of
                           KUP   : begin
                                     If CPos<>1 then
                                     begin
                                        Temp2:=(Hi(WindMin)+CPos-1)*80;
                                        Num:=Temp2+Lo(WindMin);
                                        Num2:=Num+42;
                                        For Temp2:=Num to Num2 do
                                            Screen^[Temp2].AtcH:=Temp;
                                        Dec(CPOS);
                                        Dec(Num,80);
                                        Dec(Num2,80);
                                     end
                                     else
                                     begin
                                        If Start=1 then
                                        begin
                                          Beep(1);
                                          Beep(3);
                                        end
                                        else
                                        begin
                                          Temp2:=(Hi(WindMin)+CPos-
1)*80;
                                          Num:=Temp2+Lo(WindMin);
                                          Num2:=Num+42;
                                          For Temp2:=Num to Num2 do
Screen^[Temp2].AtcH:=Temp;
                                          GotoXy(1,1);
                                          InsLine;
                                          Dec(Start);
                                          Dec(Last);
                                          With BPUse[Start] do
                                            If BP<>NoBP then
                                            begin
                                              ChangeDecto2Hex
(Hex1,Hex2,Address);
                                              Write(' BreakPoint
#',Start:2,' at $',Hex1,Hex2);
```

```
                                                    Case BP of
                                                        StrickBp: Write('
Type: Sticky;    ');
                                                        NonStrickBp: Write('
Type: Non-Sticky;');
                                                    end;
                                                  end
                                                  else
                                                    Write(' BreakPoint
#',Start:2,' at $xxxx No Breakpoint;      ');
                                                GotoXy(1,12);
                                                TextAttr:=Temp;
                                                ClrEol;
Dec(Last);
                                                GotoXy(27,12);
                                                TextAttr:=TextInv;
                                                Write('[MORE]');
                                                TextAttr:=Temp;
                                               end;
                                             end;
                                             For Temp2:=Num to Num2 do
                                               Screen^[Temp2].AtcH:=TextINV;
                                           end;
                                    KDown: begin
                                             If (CPos<11) and ((CPos+Start-1)
<=MaxBp) then
                                             begin
                                               Temp2:=(Hi(WindMin)+CPos-1)*80;
                                               Num:=Temp2+Lo(WindMin);
                                               Num2:=Num+42;
                                               For Temp2:=Num to Num2 do
                                                   Screen^[Temp2].AtcH:=Temp;
                                                Inc(CPOS);
                                                Inc(Num,80);
                                                Inc(Num2,80);
                                             end
                                             else
                                             begin
                                               If Last>MaxBp then
                                               begin
                                                  Beep(1);
                                                  Beep(3);
                                               end
                                               else
                                               begin
                                                 Temp2:=(Hi(WindMin)+CPos-
1)*80;
                                                 Num:=Temp2+Lo(WindMin);
                                                 Num2:=Num+42;
                                                 For Temp2:=Num to Num2 do
                                                   Screen^[Temp2].AtcH:=Temp;
                                                 GotoXy(40,12);
                                                 Writeln;
                                                 GotoXy(1,11);
                                                 Inc(Start);
                                                  With BPUse[Last] do
                                                    If BP<>NoBP then
                                                    begin
                                                      ChangeDecto2Hex
(Hex1,Hex2,Address);
                                                      Write(' BreakPoint
#',Last:2,' at $',Hex1,Hex2);
                                                      Case BP of
                                                          StrickBp: Write('
Type: Strcky;    ');
```

```
                                                    NonStrickBp: Write('
Type: Non-Sticky;');
                                                      end;
                                                  end
                                                  else
                                                    Write(' BreakPoint
#',Last:2,' at $xxxx No Breakpoint;     ');
                                                  Inc(Last);
                                              end;
                                            end;
                                            For Temp2:=Num to Num2 do
                                              Screen^[Temp2].AtcH:=TextINV;
                                            end;
                                    Else
                                      Beep(3);
                                  end;{#0}
                                  GotoXY(33,13);
                            end;   {#0}
                      #27:;
                      #10,#13: begin
                                  Window(30,16,74,19);
                                  GotoXY(1,1);   ClrEol;
                                  GotoXy(1,2);   ClrEol;
                                  GotoXy(1,2);
                                  Writeln(' Hit ESC to close the window.');
                                  GotoXy(1,3);   ClrEol;
                                  GotoXy(1,1);
                                  II:=Start+CPos-1;
                                  Write(' Enter the Address for breakpoint #
');
                                  Write(II,'.');
                                  GotoXy(12,4); Write('at Address $');
                                  GetAddress(Num,Hex1,Hex2,4);
                                  Case Hex1[1] of
                                    #13 : begin
                                            ChangeDecTo2Hex(Hex1,Hex2,BPUse
[II].Address);
                                            Write(Hex1,Hex2);
                                          end;
                                    #27 : Ch:=#27;
                                    Else
                                          begin
                                            BPUse[II].Address:=Num;
                                          end;
                                  end;
                                  If CH<>#27 then
                                  begin
                                    Num:=II;
                                    GotoXy(1,1); ClrEol;
                                    Write(' C=Clear; S=Sticky; N=Non-Sticky');
                                   Repeat
                                     GotoXy(32,4);
                                     Write('Type: ');
                                     Ch:=ReadKey;
                                     If Ch in [#13,#27, 'c', 'C', 's', 'S',
'n', 'N'] then
                                      Case Ch of
                                        'c', 'C' : BPUse[Num].BP:=NoBp;
                                        's', 'S' : BPUse[Num].BP:=StrickBp;
                                        'n', 'N' : BPUse[Num].BP:=NonStrickBp;
                                        #13,#27:;
                                      end
                                    else
                                      Beep(3);
                                   Until Ch in [#13,#27, 'c', 'C', 's', 'S',
'n', 'N'];
```

```
                                If Ch <>#27 then
                                begin
                                  II:=1;
                                  Window(1,1,80,25);
                                  TextAttr:=TextNorm;
                                  Repeat
                                    Inc(II);
                                  Until (Show[II].Addr=BPUse[Num].Address)
or (II>MaxInstL);
                                  If Show[II].Addr=BPUse[Num].Address then
                                  begin
                                    SHow[II].BkPoint:=BPChar[BPUse
[Num].BP];
                                    GotoXY(1,II); Write(SHow[II].BkPoint);
                                  end;
                                  TextAttr:=Temp;
                                  Window(30,16,74,19);
                                  GotoXY(1,1);   ClrEol;
                                  GotoXy(1,2);   ClrEol;
                                  GotoXy(1,3);   ClrEol;
                                  GotoXy(1,4);   ClrEol;
                                  GotoXy(1,1);
                                  Writeln(' Use ',#24,#25,' to move the
highlight.');
                                  Writeln(' Hit <CR> to select.');
                                  Writeln(' Hit ESC to close the window.');
                                  Window(30,3,74,14);
                                  GotoXy(33,12);

                                end;
                              end;
                            end;
                    Else
                        Beep(3);
                  end;{Case}
                Until Ch=#27;
                CloseLastWin;
                TextAttr:=TextNorm;
                For Num:=1 to MaxBp do
                begin
                  II:=1;
                  Repeat
                    Inc(II);
                  Until (Show[II].Addr=BPUse[Num].Address) or (II>MaxInstL);
                  If Show[II].Addr=BPUse[Num].Address then
                  begin
                    SHow[II].BkPoint:=BPChar[BPUse[Num].BP];
                    GotoXY(1,II); Write(SHow[II].BkPoint); GotoXy(80,25);
                   end;
{--}             end;
                end;
{$IFDEF CLEARBP}
     AltC :     begin { Clear Break Point}
                  II:=0;
                  Repeat
                    Inc(II);
                    Found:=BPUse[II].BP<>NoBP;
                  Until Found or (II>=MaxBP);
                  IF not Found then
                  begin
                    ShowWin(ERRORWin);
                    GotoXY(5,2); Write('NO BREAK POINT ');
                    Beep(3);
                    Beep(2);
                    Delay(800);
                    CloseLastWin;
```

```
                TextAttr:=TextNorm;
              end
              else
              begin
                ShowWin(BreakPointWin);
                Window(29,15,75,19);
                GotoXy(1,1);
                Write('ฺ' );
                For II:=30 to 74 do
                 write('ฺ' );
                write('ฺ' );
                Window(30,16,74,19);
                Writeln(' Type the break point # you want to clear.');
                Writeln(' To clear all break points, hit <CR>.');
                Writeln(' To get out without changing, hit ESC.');
                Window(30,2,74,14);
                For II:=1 to MaxBP do
                begin
                  With BPUse[II] do
                    If BP<>NoBP then
                    begin
                      ChangeDecto2Hex(Hex1,Hex2,Address);
                      Write(' #',II:2,' at $',Hex1,Hex2,';');
                      If (Lo(WindMax)-Lo(WindMin)+1)<(WhereX+14) then
                         Writeln;
                    end;
                end;
                Window(30,16,74,19);
                GotoXy(1,4);
                Write(' Enter BreakPoint #: ');
                GetNum(Num,Ch,2);
                CloseLastWin;
                TextAttr:=TextNorm;
                If (Num=0) or (Ch=#13) then
                begin
                  For II:=1 to MaxBP do
                    BPUse[II].BP:=NoBP;
                  For II:=2 to MaxInstL do
                  begin
                    GoToXy(1,II); write(' ');
                    Show[II].BkPoint:=' ';
                  end;
                end
                else
                begin
                  If Ch<>#27 then
                  begin
                    BPUse[Num].BP:=NoBP;
                    II:=1;
                    Repeat
                      Inc(II)
                    Until (II=MaxInstL) or (Show[II].Addr=BPUse
[Num].Address);
                    If Show[II].Addr=BPUse[Num].Address then
                    begin
                       GotoXy(1,II);
                       Write(' ');
                       Show[II].BkPoint:=' ';
                    end;
                  end;
                end;
              end;
            end;
{$ENDIF}
      AltQ :    begin
                  GotoXy(50,25); Write('Quit y/n?');
```

```
                    Ch:=ReadKey;
                    If Ch in ['Y', 'y'] then
                    begin
                       Exit:=True; Wait:=True; DoRun:=True;
                    end
                    else
                    begin
                      GotoXy(50,25); Write('          ');
                      GotoXy(80,25);
                    end;
                  end;
     else
     begin

       write(#7);                    {Ch:=Readkey; If Ch<>#0 then
Fin:=True;}
     end;
  end;
Until Fin;
TextInv:=TempInv;
If ClearBPMark then
begin
  BPon:=False;
  Temp:=0;
  For II:=2 to MaxInstL do
  begin
    Temp:=Temp+80;
    With Screen^[Temp+1] do
    begin
      Ch:=' ';
      AtCh:= AtCh and 127;
    end;
  end;
end;
GotoXY(80,25);
end;
{-----------------------------------------------------}
```

9.7. File EXCINS.INC

```
{-----------------------------------------------------}
Procedure ExecuteIns(Code : Integer;
                     Var Inst : InstructionType    );
Var
  Result, AResult, Num,
  II, Address           : Integer;
  TC, TH, BitCon, TV    : BitType;
  Operand               : OperandType;
  ANum                  : RegisterType;
  N1, N2, N3            : ShortInt;
  B1, B2, B3            : Byte;
begin
  Inst:= AllIns[Code];
  Case Code of
    27: begin
{          With Inst do
          begin
            Nem := 'ABA';
            Acc := None;
            AddMode :=INH;
          end;
}
          DoAdd(Result,AResult, Accum[A].DecValue, Accum[B].DecValue);
          Accum[A].DecValue:=AResult;
          ChangeFromDec(Accum[A]);
        end; {27 ABA}
    137, 153, 185, 169, 201, 217, 249, 233 :
        begin
          With Inst do
          begin
             Nem := 'ADC';
            If Code in [137, 153, 185, 169] then Acc := A else Acc:=B;
            Case Code of
              137, 201: AddMode :=IMM;
              153, 217: AddMode :=DIR;
              185, 249: AddMode :=EXT;
              169, 233: AddMode :=IND;
            end;

            Num:=FindData(AddMode);
            TH:=0;
            TC:=0;

            If CC[C]>0 then
            begin
              If Accum[Acc].DecValue=255 then
              begin
                Accum[Acc].DecValue:=0;
                CC[C]:=1;
                TC:=1;
                TH:=1;
              end
              else
              begin
                If LowerNibble(Accum[Acc].DecValue)=15 then TH:=1;
                Accum[Acc].DecValue:=Accum[Acc].DecValue+1;
              end;
            end;  {End If CC[C]>0}
            DoAdd(Result, AResult, Accum[Acc].DecValue, Num);
            Accum[Acc].DecValue:=AResult;
```

```
          ChangeFromDec(Accum[Acc]);
          If TC=1 then CC[C]:=1;
          If TH=1 then CC[H]:=1;

        end;

      end; {end ADC}
    139, 155, 187, 171, 203, 219, 251, 235 :
      begin
        With Inst do
        begin
{          Nem := 'ADD';
          If Code in [139, 155, 187, 171] then Acc := A else Acc:=B;
          Case Code of
            139, 203: AddMode :=IMM;
            155, 219: AddMode :=DIR;
            187, 251: AddMode :=EXT;
            171, 235: AddMode :=IND;
          end;
}
          Num:=FindData(AddMode);
          TH:=0;
          TC:=0;
          DoAdd(Result, AResult, Accum[Acc].DecValue, Num);
          Accum[Acc].DecValue:=AResult;
          ChangeFromDec(Accum[Acc]);

        end;
      end; {end ADD}
    132, 148, 180, 164, 196, 212, 244, 228 :
      begin
        CC[V]:=0;
        With Inst do
        begin
{          Nem:='AND';
          If Code in [132,148,180,164] then Acc := A else Acc := B;
          Case Code of
            132, 196 : AddMode:=IMM;
            148, 212 : AddMode:=DIR;
            180, 244 : AddMode:=EXT;
            164, 228 : AddMode:=IND;
          end;
}
          Num:=FindData(AddMode);
          With Accum[Acc] do
            begin
              DecValue:=DecValue And Num;
              ChangeFromDec(Accum[Acc]);
              AdjustZAndN(DecValue);
            end;

        end;
      end; {End 'AND'}
    133, 149, 181, 165, 197, 213, 245, 229 :
      begin
        CC[V]:=0;
        With Inst do
        begin
{          Nem:='BIT';
          If Code in [133,149,181,165] then Acc := A else Acc := B;
          Case Code of
            133, 197 : AddMode:=IMM;
            149, 213 : AddMode:=DIR;
            181, 245 : AddMode:=EXT;
            165, 229 : AddMode:=IND;
          end;
```

```
}

          Num:=FindData(AddMode);
          Result:=Accum[Acc].DecValue And Num;
          If Result=0 then CC[Z]:=1 else CC[Z]:=1;
          If Result>PosNum then CC[N]:=1 else CC[N]:=0;

        end;
      end; {End 'BIT'}
   72, 88, 120, 104, 73, 89, 121, 105 :
      begin
       With Inst do
        begin
{          If Code in [72, 88, 120, 104] then Nem:='ASL' else Nem:='ROL';}
          Case AddMode of
           INH :
            begin
              ANum:=Accum[Acc];
            end;
           EXT, IND :
            begin
              CalAddress(AddMode,Address);
              ANum.NumByte:=1;
              GetMemory(Address,ANum.DecValue);
              Address:=Address-1;
              ChangeFromDec(ANum);
            end;
          end;{Case}
          With Anum do
          begin
            TC:=CC[C];
            CC[C]:=LowerByteB[7];
            CC[V]:=LowerByteB[7] xor LowerByteB[6];
            For II:=7 downto 0 do
                LowerByteB[II]:=LowerByteB[II-1];
            If Nem[1]='A' then LowerByteB[0]:=0 else LowerByteB[0]:=TC ;
            DecValue:=Lo(DecValue shl 1)+LowerByteB[0];
            ChangeFromBintoHex(LowerByteB,LowerByteH);
            AdjustZAndN(DecValue);
          end;
          If AddMode in [EXT, IND ] then
            StoreMemory(Address, ANum.DecValue)
          else
            Accum[Acc]:=ANum;
        end; {With Inst}
      end;  {End 'ASL', 'ROL'}

   71, 87, 119, 103, 68, 84, 116, 100, 70, 86, 118, 102 :
      begin
        With  Inst do
        begin
          TC:=CC[C];
{          Case Code of
            71, 87, 119, 103: Nem:='ASR';
            68, 84, 116, 100: Nem:='LSR';
            70, 86, 118, 102: Nem:='ROR';
          end;
}
          If AddMode=INH then
            begin
              ANum:=Accum[Acc];
            end
          else
            begin
              CalAddress(AddMode,Address);
              GetMemory(Address,ANum.DecValue);
```

```
            ANum.NumByte:=1;
            Address:=Address-1;
            ChangeFromDec(ANum);
          end;
        With Anum do
        begin
          CC[C]:=LowerByteB[0];
          For II:=0 to 6 do
            LowerByteB[II]:=LowerByteB[II+1];
          Case Nem[1] of
            'A':;
            'L': LowerByteB[7]:=0;
            'R': LowerByteB[7]:=TC;
          end;
          DecValue:=Lo(DecValue shr 1)+LowerByteB[7]*128;
          ChangeFromBintoHex(LowerByteB,LowerByteH);
          AdjustZAndN(DecValue);
          Case Nem[1] of
            'A','R': CC[V]:=CC[N] Xor CC[C];
            'L'    : CC[V]:=CC[C];
          end;

        end;
        If AddMode <> INH  then
        begin
          StoreMemory(Address, ANum.DecValue)
        end
        else
        begin
          Accum[Acc]:=ANum;
        end;
      end; {With Inst}
    end;  {End 'ASR', 'LSR', 'ROR'}

  36, 37, 44, 46, 39, 47, 34, 45, 35, 43, 38, 42, 32, 40, 41 :
    begin
      With Inst do
      begin
        Case Code of
        36: begin
{                Nem:='BCC';}
              BitCon:=NotBit(CC[C]);
            end;
        37: begin
{                Nem:='BCS';  }
              BitCon:=CC[C];
            end;
        44: begin
 {               Nem:='BGE';}
              BitCon:=NotBit((CC[N] xor CC[V]));
            end;
        46: begin
{               Nem:='BGT';  }
              BitCon:=NotBit(CC[Z]) and  NotBit(CC[N] xor CC[V]);
            end;
        39: begin
 {               Nem:='BEQ';}
              BitCon:=CC[Z];
            end;
        47: begin
{               Nem:='BLE';  }
              BitCon:=CC[Z] or (CC[N] xor CC[V]);
            end;
        34: begin
 {               Nem:='BHI';}
              BitCon:=NotBit(CC[Z] or CC[C]);
```

```
            end;
          45: begin
{                Nem:='BLT';  }
               BitCon:=CC[N] xor CC[V];
            end;
          35: begin
{                Nem:='BLS';}
               BitCon:=(CC[C] or CC[Z]);
            end;
          43: begin
{                Nem:='BMI';  }
               BitCon:=CC[N];
            end;
          38: begin
{                Nem:='BNE';}
               BitCon:=NotBit(CC[Z]);
            end;
          42: begin
{                Nem:='BPL';  }
               BitCon:=NotBit(CC[N]);
            end;
          32: begin
{                Nem:='BRA';}
               BitCon:=1;
            end;
          40: begin
{                Nem:='BVC';  }
               BitCon:=NotBit(CC[V]);
            end;
          41: begin
 {               Nem:='BVS';}
               BitCon:=CC[V];
            end;
        end;
        BitCon:=BitCon and $01;
        With PC do
        begin
          GetMemory(DecValue,Num);
          If BitCon=1 then
          begin
            If Num<128 then
             DecValue:=DecValue+Num
            else
            begin
             Num:=256-Num;
             DecValue:=DecValue-Num;
            end;
          end;{If CC[C]=0]}
        end;{With PC}
        ChangeFromDec(PC);
      end;
    end;  {End 'BCC'}
  141 :
    begin
{      With Inst do
      begin
        Nem:='BSR';
        AddMode:=REL;
        Acc:=None;
      end;
}
        With PC do
        begin
          GetMemory(DecValue,Num);
          ChangeFromDec(PC);
          Result:=Lo(PC.DecValue);
```

```
            Push(Result);
            Result:=Hi(PC.DecValue);
            Push(Result);
            If Num<128 then
               DecValue:=DecValue+Num
            else
            begin
               Num:=256-Num;
               DecValue:=DecValue-Num;
             end;
          end;{With PC}
          ChangeFromDec(PC);

      end; {BSR}
   17 :
      begin
        With Inst do
        begin
{          Nem:='CBA';
          Acc:=None;
          AddMode:=INH;

          TH:=CC[H];
          DoAdd(Result,AResult,Accum[A].DecValue,256-Accum[B].DecValue);
          CC[H]:=TH;
          AdjustZandN(AResult);}
          B1:=Lo(Accum[B].DecValue);
          N1:=B1;
          B2:=Lo(Accum[A].DecValue);
          N2:=B2;
          N3:=N2-N1;
          B3:=N3;
          B1:=B1 and $80;  {M}
          B2:=B2 and $80;  {X}
          B3:=B3 and $80;  {R}
          If (((B2 and (Not B1) and (Not B3))
              or ((Not B2) and B1 and B3))
             and $80 ) = 0 then CC[V]:=0 else CC[V]:=1;
          If ((   ((Not B2) and B1)
               or (B1 and B3)
               or (B3 and (Not B2)))
               and $80) = 0 then CC[C]:=0 else CC[C]:=1;
          If N3=0 then CC[Z]:=1 else CC[Z]:=0;
          If N3<0 then CC[N]:=1 else CC[N]:=0;


        end;
      end;{CBA}
    12: begin
          CC[C]:=0;
        end;{CLC}
    13: begin
{          Nem:='SEC';}
          CC[C]:=1;
        end;{CLC}
    14: begin
{          Nem:='CLI';}
          CC[I]:=0;
        end;{CLI}
    15: begin
{          Nem:='SEI';}
          CC[I]:=1;
        end;{CLI}
    10: begin
{          Nem:='CLV';}
          CC[V]:=0;
```

```
      end;{CLI}
    11: begin
{           Nem:='SEV';}
          CC[V]:=1;
      end;{CLI}
    79, 95, 127, 111 :
      begin
        With Inst do
        begin
{         Nem:='CLR';}
        CC[N]:=0;
        CC[Z]:=1;
        CC[V]:=0;
        CC[C]:=0;
        Case AddMode of
          INH:
              begin
                With Accum[Acc] do
                begin
                  DecValue:=0;
                  LowerByteH[1]:='0';
                  LowerByteH[2]:='0';
                  For II:=0 to 7 do LowerByteB[II]:=0;
                end;
              end;
          EXT, IND:
              begin
                CalAddress(AddMode, Address);
                StoreMemory(Address,0);
              end;
        end;
       end;
      end;{CLR}
    10: begin
          Nem:='CLV';}
        CC[V]:=0;
      end;{CLV}
   129, 145, 177, 161, 193, 209, 241, 225 :
     begin
       CC[V]:=0;
       With Inst do
       begin
          Nem:='CMP';}
        B1:=Lo(FindData(AddMode));
        N1:=B1;
         ChangeFromDec(ANum);}
        B2:=Lo(Accum[Acc].DecValue);
        N2:=B2;
        N3:=N2-N1;
        B3:=N3;
        B1:=B1 and $80;  {M}
        B2:=B2 and $80;  {X}
        B3:=B3 and $80;  {R}
        If (((B2 and (Not B1) and (Not B3))
            or ((Not B2) and B1 and B3))
           and $80 ) = 0 then CC[V]:=0 else CC[V]:=1;
        If ((   ((Not B2) and B1)
             or (B1 and B3)
             or (B3 and (Not B2)))
             and $80) = 0 then CC[C]:=0 else CC[C]:=1;
        If N3=0 then CC[Z]:=1 else CC[Z]:=0;
        If N3<0 then CC[N]:=1 else CC[N]:=0;


         With Accum[Acc] do
          begin
```

```
            If ANum.DecValue > DecValue then CC[C]:=1 else CC[C]:=0;
            If ANum.DecValue>=128 then
               Num:=DecValue-(256-ANum.DecValue)
            else
            begin
               Num:=DecValue-ANum.DecValue;
               If Num<0 then Num:=256-Num;
            end;
            AdjustZAndN(Num);
          end;}

      end;
    end; {End 'CMP'}
  140, 156, 188, 172 :
    begin
      With Inst do
      begin
{        Nem:='CPX';}
        If AddMode<>IMM then
        begin
          CalAddress(AddMode,Address);
          GetMemory(Address,Result);
          GetMemory(Address,Num);
        end
        else
        begin
          GetMemory(PC.DecValue,Result);
          GetMemory(PC.DecValue,Num);
          ChangeFromDec(PC);
        end;
      end;
      { Compare XH with Result and XL with Num }
      With X do
      begin
        AResult:=GetDecFromHex(HighByteH);
        II:=GetDecFromHex(LowerByteH);
      end;
      If (Aresult=Result) and (II=Num) then
      begin
        CC[Z]:=1;
        CC[N]:=0;
        CC[V]:=0;
      end
      else
      begin
        CC[Z]:=0;
        {-----Address variable are using here to store data only ----}
        If Result>PosNum then Result:=Result-256;
        If AResult>PosNum then AResult:=AResult-256;
        Address:=AResult-Result;
        If (Address>PosNum) or (Address<-128) then
          CC[V]:=1
        else
          CC[V]:=0;
        IF Address>255 then Address:=Address-256
        else If Address< 0 then
               If Address>=-128 then Address:=Address+256
               else  Address:=Address+384;
        If Address>PosNum then CC[N]:=1 else CC[N]:=0;
      end;
    end;
  136, 152, 184, 168, 200, 216, 248, 232 :
    begin
      CC[V]:=0;
      With Inst do
      begin
```

```
{            Nem:='EOR';}
          Num:=FindData(AddMode);
          With Accum[Acc] do
            begin
              DecValue:= DecValue Xor Num;
              ChangeFromDec(Accum[Acc]);
              AdjustZAndN(DecValue);
            end;

        end;
      end; {End 'EOR'}
  138, 154, 186, 170, 202, 218, 250, 234 :
      begin
        CC[V]:=0;
        With Inst do
        begin
{           Nem:='ORA';}
          Num:=FindData(AddMode);
          With Accum[Acc] do
            begin
              DecValue:= DecValue Or Num;
              ChangeFromDec(Accum[Acc]);
              AdjustZAndN(DecValue);
            end;

        end;
      end; {End 'EOR'}
  76, 92, 124, 108, 74, 90, 122, 106 :
      begin  { DEC INC}
        TC:=CC[C];
        With Inst do
        begin
{          if Code in [74, 90, 122, 106] then Nem:='DEC' else Nem:='INC';}
          If AddMode=INH then
          begin
           AddMode:=INH;
           With Accum[Acc] do
           begin
             if Nem='INC' then
             begin
                Inc(DecValue);
                If DecValue=128 then CC[V]:=1 else
                begin
                  CC[V]:=0;
                  If DecValue=256 then
                    DecValue:=0;
                end;
             end
             else
             begin
                If DecValue=128 then CC[V]:=1 else CC[V]:=0;
                Dec(DecValue);
                If DecValue>0 then DecValue:=DecValue+256;
             end;
             AdjustZandN(DecValue);
           end;

           ChangeFromDec(Accum[Acc]);
           CC[N]:=Accum[Acc].LowerByteB[7];
          end
          else
          begin
            ANum.NumByte:=1;
            CalAddress(AddMode, Address);
            GetMemory(Address, ANum.DecValue);
            Address:=Address-1;
```

```
          With ANum do
          begin
            If Nem='INC' then
            begin
              Inc(DecValue);
              If DecValue=128 then
                 CC[V]:=1
              else
              begin
                CC[V]:=0;
                If DecValue=256 then DecValue:=0;
              end;
            end
            else
            begin
              If DecValue=128 then CC[V]:=1 else CC[V]:=0;
              Dec(DecValue);
              If DecValue>0 then DecValue:=DecValue+256;
            end;
            If DecValue=256 then DecValue:=0;
            AdjustZandN(DecValue);
           StoreMemory(Address, DecValue);
           ChangeFromDec(ANum);
           CC[N]:=ANum.LowerByteB[7];
          end;
        end;
      end;
      CC[C]:=TC;
    end; {End 'INC','DEC'}
126, 110 :
    begin
      With Inst do
      begin
{          Nem:='JMP';}
        CalAddress(AddMode,Address);
        PC.Decvalue:=Address;
{         ChangeFromDec(PC);}
      end;
    end;                    {'JMP'}
189, 173 :
    begin
{          Nem:='JSR';}
      CalAddress(Inst.AddMode,Address);
      If Not SpecialAddress(Address) then
      begin
        With PC do
        begin
          Result:=Lo(PC.DecValue);
          Push(Result);
          Result:=Hi(PC.DecValue);
          Push(Result);
          DecValue:=Address;
{          ChangeFromDec(PC);}
        end;{With PC}
      end
      Else
        DoSpecial(Address);
    end; {JSR}
 57 :begin
{          Nem:='RTS';}
      With PC do
      begin
        Pull(DecValue);
        Pull(Num);
        DecValue:=DecValue*256+Num;
      end;
```

```
       ChangeFromDec(PC);}
    end;                    {RTS}
59 :begin
         Nem:='RTI';}
      With ANum do
      begin
          NumByte:=1;
          Pull(DecValue);
          ChangeFromDec(ANum);
          CC[H]:=LowerByteB[5];
          CC[I]:=LowerByteB[4];
          CC[N]:=LowerByteB[3];
          CC[Z]:=LowerByteB[2];
          CC[V]:=LowerByteB[1];
          CC[C]:=LowerByteB[0];
       end;
      Pull(Accum[B].DecValue);
      ChangeFromDec(Accum[B]);
      Pull(Accum[A].DecValue);
      ChangeFromDec(Accum[A]);
      With X do
      begin
        Pull(DecValue);
        Pull(Num);
        DecValue:=DecValue*256+Num;
      end;
      ChangeFromDec(X);
      With PC do
      begin
        Pull(DecValue);
        Pull(Num);
        DecValue:=DecValue*256+Num;
      end;
{      ChangeFromDec(PC);  }
    end; {RTI}
63 :begin
{       Nem:='SWI';}
      DoInterrupt(SWI);
    end;  {SWI}
62 :begin
{       Nem:='WAI';}
      PreInterrupt;
      Wait:=True;
      Dec(PC.DecValue);
    end;  {WAI}
25 :begin
{       Nem:='DAA';}
      With Accum[A] do
        If CC[C]=0 then
        begin
          If ((LowerByteH[1] in ['0'..'8']) and
              (LowerByteH[2] in ['A'..'F']) and (CC[H]=0))
                          or
             ((LowerByteH[1] in ['0'..'9']) and
              (LowerByteH[2] in ['0'..'3']) and (CC[H]=1)) then
             DecValue:=DecValue+6
          else
          begin
            If ((LowerByteH[1] in ['A'..'F']) and
                (LowerByteH[2] in ['0'..'9']) and (CC[H]=0)) then
              begin
                DecValue:=DecValue+96;
                CC[C]:=1;
              end
              Else
               If  ((LowerByteH[1] in ['9'..'F']) and
```

```
                    (LowerByteH[2] in ['A'..'F']) and (CC[H]=0))
                              OR
                    ((LowerByteH[1] in ['A'..'F']) and
                     (LowerByteH[2] in ['0'..'3']) and (CC[H]=1)) then
                    begin
                      DecValue:=DecValue+102;
                      CC[C]:=1;
                    end;
              end;
          end
          Else
          begin
               If ((LowerByteH[1] in ['0'..'2']) and
                  (LowerByteH[2] in ['0'..'9']) and (CC[H]=0)) then
                 begin
                   DecValue:=DecValue+96;
                   CC[C]:=1;
                 end
                 Else
                   If  ((LowerByteH[1] in ['0'..'2']) and
                        (LowerByteH[2] in ['A'..'F']) and (CC[H]=0))
                              OR
                        ((LowerByteH[1] in ['0'..'3']) and
                         (LowerByteH[2] in ['0'..'3']) and (CC[H]=1)) then
                        begin
                          DecValue:=DecValue+102;
                          CC[C]:=1;
                        end;
          end;
          ChangeFromDec(Accum[A]);
          AdjustZandN(Accum[A].DecValue);
     end;{DAA}
134, 150, 182, 166, 198, 214, 246, 230 :
     begin
{          Nem:='LDA';}
          TC:=CC[C];
          TH:=CC[H];
          Accum[Inst.Acc].DecValue:=FindData(Inst.AddMode);
          ChangeFromDec(Accum[Inst.Acc]);
          AdjustZandN(Accum[Inst.Acc].DecValue);
          CC[C]:=TC;
          CC[H]:=TH;
          CC[V]:=0;
     end;                    {'LDA'}
 54, 55 :
     begin
{          Nem:='PSH';}
        Push(Accum[Inst.Acc].DecValue);
     end; { 'PSH'}
50, 51 :
     begin
{          Nem:='PUL';}
        Pull(Accum[Inst.Acc].DecValue);
        ChangeFromDec(Accum[Inst.Acc]);
     end; { 'PUL'}
1: begin
{          Nem:='NOP';}
   end; {'NOP'}
151, 183, 167, 215, 247, 231:
   begin
     With Inst do
     begin
{          Nem:='STA';}
        TC:=CC[C];
        TH:=CC[H];
        CalAddress(AddMode,Address);
```

```
        StoreMemory(Address,Accum[Acc].DecValue);
        AdjustZandN(Accum[Acc].DecValue);
        CC[C]:=TC;
        CC[H]:=TH;
        CC[V]:=0;
      end;
    end; {'STA'}
 22, 23 :
      begin
        With Inst do
        begin
          If Nem='TAB' then
          begin
            Anum:=Accum[A];
            Accum[B]:=Accum[A]
          end
          else
          begin
{             Nem:='TBA';}
            Anum:=Accum[B];
            Accum[A]:=Accum[B]
          end;
        end;
        CC[V]:=0;
        AdjustZandN(ANum.DecValue);
      end;          {'TAB','TBA'}
  6:  begin
        With Inst do
        begin
{         Nem:='TAP';   }
          With Accum[A] do
          begin
            CC[H]:=LowerByteB[5];
            CC[I]:=LowerByteB[4];
            CC[N]:=LowerByteB[3];
            CC[Z]:=LowerByteB[2];
            CC[V]:=LowerByteB[1];
            CC[C]:=LowerByteB[0];
          end;
        end;
      end;                      {'TAP'}
  7:  begin
        With Inst do
        begin
{         Nem:='TPA';}
          With Accum[A] do
          begin
            LowerByteB[7]:=1;
            LowerByteB[6]:=1;
            LowerByteB[5]:=CC[H];
            LowerByteB[4]:=CC[I];
            LowerByteB[3]:=CC[N];
            LowerByteB[2]:=CC[Z];
            LowerByteB[1]:=CC[V];
            LowerByteB[0]:=CC[C];
            ChangeFromBinToDec(LowerByteB,DecValue);
            ChangeFromBinToHex(LowerByteB,LowerByteH);
          end;
        end;
      end;                      {'TPA'}
 48:  begin
{       Nem:='TSX';}
        X.DecValue:=SP.DecValue+1;
        ChangeFromDec(X);
      end;                  {'TSX'}
 53:  begin
```

```
{               Nem:='TXS';}
              SP.DecValue:=X.DecValue-1;
              ChangeFromDec(SP);
          end;                    {'TXS'}
    77, 93, 125, 109:
          begin
            With Inst do
            begin
{              Nem:='TST';}
               If AddMode=INH then
                     AdjustZandN(Accum[Acc].DecValue)
               else
                    begin
                      Result:=FindData(AddMode);
                      AdjustZandN(Result);
                    end;
            end;
            CC[V]:=0;
            CC[C]:=0;
          end; {'TST'}
    128, 144, 176, 160, 192, 208, 240, 224,  {SUB}
    130, 146, 178, 162, 194, 210, 242, 226,  {SBC}
    16 {SBA} :
          begin
            With Inst do
            begin
                  {                 If Code in [128, 144, 176, 160, 192, 208,
240, 224] then
                                    Nem:='SUB'
                                    else
                                    Nem:='SBC';
                                    If Code=16 then Nem:='SBA';
                  }
              If AddMode<>INH then
              begin
                 Num:=FindData(AddMode);
                 AResult:=0;
              end
              else
              begin
                 Num:=Accum[B].DecValue;
                 AResult:=CC[C];
              end;
            end;
            B1:=Lo(Num);
            N1:=B1;
            With Accum[Inst.Acc] do
            begin
              B2:=Lo(DecValue);
              N2:=B2;
              If Inst.Nem='SBC' then
              begin
                 N3:=N2-N1-CC[C];
              end
              else
              begin
                 N3:=N2-N1;
              end;
              B3:=N3;
              DecValue:=B3;
            end;
            B1:=B1 and $80;
            B2:=B2 and $80;
            B3:=B3 and $80;
            If (((B2 and (Not B1) and (Not B3))
                or ((Not B2) and B1 and B3))
```

```
            and $80 ) = 0 then CC[V]:=0 else CC[V]:=1;
          If ((   ((Not B2) and B1)
               or (B1 and B3)
               or (B3 and (Not B2)))
               and $80) = 0 then CC[C]:=0 else CC[C]:=1;
          If N3=0 then CC[Z]:=1 else CC[Z]:=0;
          If N3<0 then CC[N]:=1 else CC[N]:=0;




(*          If Num>POSNum then
            Num:=Num-256;

          With Accum[Inst.Acc] do
          begin
            TC:=CC[C];
            CC[C]:=0;
            If DecValue>POSNum then Result:=Result-256;
            If ((Result>=0) and (Num>=0)) or ((Result<0) and (Result<0))
then
              TV:=1;
            If Abs(Num+AResult)>Abs(Result) then CC[C]:=1;
            If Inst.Nem='SBC' then
              DecValue:=DecValue-Num-TC
            else
              DecValue:=DecValue-Num;
            If DecValue<0 then
               DecValue:=DecValue+256
            else
               If DecValue>256 then
               begin
                 DecValue:=DecValue-256;
                 CC[C]:=1;
                 If (Num>=0) and (TV=1) then CC[V]:=1;
               end
               else
               begin
                 If (Num<0) and (TV=1) then CC[V]:=1;
               end;
          end;                              *)



          ChangeFromDec(Accum[Inst.Acc]);
{          AdjustZandN(Accum[Inst.Acc].DecValue);                    }
        end;      {SUB,SBA,SBC}
   64, 80, 112, 96 :
        begin
          With Inst do
          begin
{           Nem:='NEG';    }
            If AddMode=INH then
            begin
                  With Accum[Acc] do
                  begin
                    CC[V]:=0;
                    CC[C]:=0;
                    If DecValue>PosNum then
                    begin
                      If DecValue=128 then CC[V]:=1;
                      DecValue:=-(DecValue-256);
                    end
```

```
                    else
                    begin
                      If DecValue<>0 then
                       DecValue:=256-DecValue
                      else
                      begin
                        DecValue:=0; CC[C]:=1;
                      end;
                    end;
                    If DecValue<0 then DecValue:=256-DecValue;
                    ChangeFromDec(Accum[Acc]);
                    AdjustZandN(DecValue);
                  end;
                end
              Else
                begin
                  Acc:=None;
                  If Code=112 then AddMode:=EXT else AddMode:=IND;
                  CalAddress(AddMode,Address);
                  GetMemory(Address,Num);
                  Address:=Address-1;
                  CC[V]:=0;
                  CC[C]:=0;
                  If Num>PosNum then
                  begin
                    If Num=128 then CC[V]:=1;
                    Num:=-(Num-256);
                  end
                  else
                  begin
                    If Num<>0 then
                      Num:=256-Num
                    else
                    begin
                      Num:=0; CC[C]:=1;
                    end;
                  end;
                  If Num<0 then Num:=256-Num
                  else If Num>256 then Num:=Num-256;
                  StoreMemory(Address,Num);
                  AdjustZandN(Num);

                end;
          end;
        end;{NEG}
 67, 83, 115, 99 :
        begin
          With Inst do
          begin
{           Nem:='COM';}
            If AddMode=INH then
              begin
                With Accum[Acc] do
                begin
                  DecValue:=255-DecValue;
                  ChangeFromDec(Accum[Acc]);
                  AdjustZandN(DecValue);
                end;
              end
            else
              begin
                CalAddress(AddMode,Address);
                GetMemory(Address,Num);
                Address:=Address-1;
                Num:=255-Num;
                If Num>256 then Num:=Num-256;
```

```
                StoreMemory(Address,Num);
                AdjustZandN(Num);

              end;
        end;
        CC[C]:=1;
        CC[V]:=0;
      end;{COM}


159, 191, 175, 223, 255, 239:
      begin
        With Inst do
        begin
          if Nem='STS' then
          begin
            Anum:=SP;
          end
          else
          begin
            ANum:=X;
          end;
          CalAddress(AddMode,Address);
          With ANum do
          begin
            Result:=GetDecFromHex(HighByteH);
            StoreMemory(Address,Result);
            Address:=Address+1;
            Result:=GetDecFromHex(LowerByteH);
            StoreMemory(Address,Result);
            CC[N]:=HighByteB[7];
            If DecValue=0 then CC[Z]:=1 else CC[Z]:=0;
          end;
          CC[V]:=0;
        end;
      end; {'STS','STX'}

142, 158, 190, 174, 206, 222, 254, 238:
      begin
        With Inst do
        begin
          If AddMode<>IMM then
          begin
            CalAddress(AddMode,Address);
            With ANum do
            begin
              NumByte:=2;
              GetMemory(Address,DecValue);
              GetMemory(Address,Result);
              DecValue:=DecValue*256+Result;
            end;
          end
          else
          begin
            With ANum do
            begin
              NumByte:=2;
              GetMemory(PC.DecValue,DecValue);
              GetMemory(PC.DecValue,Result);
              ChangeFromDec(PC);
              DecValue:=DecValue*256+Result;
            end;
          end;
          ChangeFromDec(ANum);
          With ANum do
          begin
```

```
              CC[N]:=HighByteB[7];
              If DecValue=0 then CC[Z]:=1 else CC[Z]:=0;
            end;
            if Nem='LDS' then
            begin
              SP:=ANum;
            end
            else
            begin
              X:=ANum;
            end;

            CC[V]:=0;
          end;
        end; {'LDS','LDX'}
    52, 49:
        begin
{             Nem:='DES';}
          With SP do
            If Inst.Nem='DES' then
               DecValue:=DecValue-1
            else
               DecValue:=DecValue+1;
          ChangeFromDec(SP);
        end; {'DES' 'INS'}
    9,8 : begin
          With X do
          begin
            If Inst.Nem='DEX' then
              DecValue:=DecValue-1
            else
              DecValue:=DecValue+1;
          end;
          ChangeFromDec(X);
          If X.DecValue=0 then CC[Z]:=1 else CC[Z]:=0;

        end; {'DEX','INX'}

    else  begin
          GotoXy(erX,ErY); Write(' UnKnown Code');GotoXY(1,1);
        end ;{else CASE}
  end; {End Case INS}
  ChangeFromDec(PC);
end;
```

9.8. File INC2.INC

```
Procedure ShowSplashScreen;
begin
  ShowWin(SplashScreen);
  GotoXy(3,2);Write('Sim68: 6800 Simulator');
  GotoXy(3,3);Write('  By Thit Siriboon');
  GotoXy(3,4);Write('     Chulalongkorn University');
  GotoXy(3,5);Write('     Bangkok, Thailand');
  Delay(2000);
  CloseWin(SplashScreen);
end;
{-------------------------------------------------}
Procedure ResetCPU;
begin
  PC.DecValue:=0;
  ChangeFromDec(PC);
  SP.DecValue:=1000;
  ChangeFromDec(SP);
  X.DecValue:=0;
  ChangeFromDec(X);
  Accum[A].DecValue:=0;
  ChangeFromDec(Accum[A]);
  Accum[B].DecValue:=0;
  ChangeFromDec(Accum[B]);
  CC[H]:=0;
  CC[I]:=0;
  CC[N]:=0;
  CC[Z]:=0;
  CC[V]:=0;
  CC[C]:=0;
end;
{-------------------------------------------------}
Procedure ClearMemory;
var
  II, III : integer;
 { PReg: PIARegName;  }
begin
  For II:=MinNum+1 to MaxNum do
  begin
    MemoryUp^[II]:=0;
  end;
  MemoryLow:=0;

end;
{-------------------------------------------------}
Procedure ReSetVariable;
Var
  II  : Integer;
begin
  Wait:=False;
  ProgMFlag:=False;
  BPOn:=False;
  CyCles:=0;
  CScrX:=ScreenMinX+2;
  CScrY:=ScreenMinY;
  BreakMarkY:=2;
  DoRun:=False;
  Exit:=False;
  For II:=1 to MaxHist do
    With Hist[II] do
    begin
```

```
      Next:=II+1;
      Past:=II-1;
      Empty:=True;
      ChangeMem:=False;
    end;
  Hist[1].Past:=MaxHist;
  Hist[MaxHist].Next:=1;
  NextHist:=1;
  DoRecord:=True;
  BreakPoint:=NoBp;
  UpdateSmallWin:=False;
  For II:=1 to MaxBP do
    With BPUse[II] do
    begin
      Address:=0;
      BP:=NoBP;
    end;
  IRQF:=False;
  NMIF:=False;
  NMILV:=1;
  RESF:=False;
  CWinX:=1; CWinY:=1;
  LastAddress:=0;TopAddress:=0;
end;
{-------------------------------------------------}
Procedure Initialize;
Var
  II, III : integer;
  CCBit : BitofCCType;
  AName : AccNameType;
  VName : RegValueType;
begin
  ResetVariable;
  ShiftByte := ShiftByte and $0F;
  NewProg:=True;
  For VName:=WPC to WB do
    With WatchValue[VName] do
    begin
         Name:=VName;
         DecValue:=0;
         On:=False;
    end;
  With WatchValue[WCC] do
  begin
    Name:=WCC;
    For CCBit:=H to C do
    begin
      Value[CCBit]:=0;
      BitOn[CCBit]:=False;
    end;
  end;
  DetectGraph(II,III);
  If (II=HercMono) or (II= EGAMono) or (II=-2) then
  begin
    TextNorm:=MonoNorm;
    TextInv:=MonoInv;
    Screen:=@MScreen;
  end
  else
  begin
    If (II=CGA) or (II=MCGA) then
    begin
      TextNorm:=ColorNorm;
      TextInv:=ColorInv;
    end
    else
```

```
  begin
    TextNorm:=EGANorm;
    TextInv:=EGAInv;
  end;
  PTOOLWIN_Screen_Type :='C' ;
  Screen:=@CScreen;
end;
TextAttr:=TextNorm;
Assign(CON,'CON');
Rewrite(CON);
ClearMemory;
Wait:=False;
For II:=1 to 3 do
 With WinMemory[II] do
 begin
   Num:=0;
   Hex1:='00';
   Hex2:='00';
 end;
For AName:=A to B do
begin
  With Accum[AName] do
  begin
    NumByte:=1;
    DecValue:=0;
    ChangeFromDec(Accum[AName]);
  end;
end;
For CCBit:= H to C do
begin
  CC[CCbit]:=0;
end;
PC.NumByte:=2;
PC.DecValue:=0;
ChangeFromDec(PC);
X.NumByte:=2;
X.DecValue:=0;
ChangeFromDec(X);
SP.NumByte:=2;
SP.DecValue:=1000;
ChangeFromDec(SP);
ClrScr;
{=================================================================}
{   Set up DisPlay                                                }
GotoXy(3,1); Write('ADDR'); GotoXy(10,1); Write('OPERATIONS');
GotoXy(29,1);
Write('6800       Simulator/Debugger     Version ',VersionNo);
GotoXy(29,2);
Write('CPU Registers');
GotoXy(48,2);
Write ('H  I  N  Z  V  C');
GotoXy(29,3);
Write('CC: 11');
GotoXy(29,4); Write('A : ');
GotoXy(29,5); Write('B : ');
GotoXy(29,6); Write('X : ');
GotoXy(29,7); Write('PC: ');
GotoXy(29,8); Write('SP: ');
GotoXy(48,4); Write('IRQ   NMI   RES');
GotoXy(49,5); Write('1');
GotoXy(55,5); Write('1');
GotoXy(61,5); Write('1');
GotoXy(68,4); Write('Cycles');
GotoXy(65,5); Write(Cycles:10);
RuningMode:=SingleStep;
GotoXy(65,2); Write('Mode');
```

```
  GotoXy(65,3); Write('Step');
  Speed:=Mid;
  GotoXy(70,2); Write('Speed');
  GotoXy(71,3); Write('Mid');
  Watch:=False;
  GotoXy(76,2); Write('Watch');
  GotoXy(77,3); Write('OFF');
  GotoXY(32,10); Write('Memory');
  GotoXy(50,10); Write('Contents ');
  GotoXY(32,13); Write('Memory');
  GotoXy(50,13); Write('Contents ');
  GotoXY(32,16); Write('Memory');
  GotoXy(50,16); Write('Contents ');
  {=================================================================}
  Set_CurSor(22,23);
  GotoXy(MinWinX-1,MinWinY-1); Write('ษ');
  CWinY:=MinWinY-1;
  For CWinX:=MinWinX+1 to MaxWinX+1 do
  begin
    Write('ส');
  end;
  Write('ป');          CWinX:=MaxWinX+1;
  For CWinY:=MinWinY to MaxWinY do
  begin
    GotoXY(CWinX,CWinY); Write('บ');
    GotoXY(MinWinX-1,CWinY); Write('บ');
  end;
  For CWinX:=MinWinX-1 to MaxWinX do
  begin
    GotoXy(CWinX,MaxWinY+1); Write('ส');
  end;
  GotoXY(MinWinX-1,MaxWinY+1); write('ศ');
  GotoXY(MaxWinX+1,MaxWinY+1); write('ผ');
  CWinX:=1; CWinY:=1;

  SetWindow(ChangeMemoryWin,2,2,34,9,white,Black,True);
  SetWindow(ErrorWin,30,19,40,5,white,Black,True);
  SETWindow(BreakPointWin,29,2,46,19,white,Black,True);
  SetWindow(SplashScreen,20,10,33,5,Magenta,Yellow,True);
  New(HeadAddressLink);
  Mark(HeadAddressLink);
  CurrentAddressLink:=HeadAddressLink;
  With HeadAddressLink^ do
  begin
    Next:=Nil;
    Back:=Nil;
  end;
end;

{-------------------------------------------------}
Function SearchBp ( Address  : Integer) : BPType;
Var
  II : integer;
  Found : Boolean;
begin
  Found:=False;
  II:=0;
  While (Not Found) and (II<MaxBP) do
  begin
    II:=II+1;
    Found:=BPUse[II].Address=Address;
  end;
  If Not Found then
    SearchBp:=NoBp
  else
    SearchBP:=BPUse[II].BP;
```

```
end;
{-------------------------------------------------}
Procedure MakeInstList;
Var
  Num, Code, CYInst, Address, Temp  : integer;
  OP1           : HexFormat2;

begin
  Address:=TopAddress;
  CYInst:=2;
  While CYinsT<=MaxInstL do
  begin
     GotoXY(1,CYInst);
     LastAddress:=Address;
     Temp:=Address;
     Show[CYInst].BkPoint:=BPChar[SearchBP(Address)];
     If Address>0 then
     begin
       ChangeDecToHex(OP1, (Address Div 256));;
       Show[CYInst].Stg:=OP1;
       ChangeDecToHex(OP1, (Address mod 256));
       Show[CYInst].Stg:=Show[CYInst].Stg+OP1;
     end
     else
     begin
       Address:=Address+1+MaxNum;
       ChangeDecToHex(OP1, (Address Div 256)+128);;
       Show[CYInst].Stg:=OP1;
       ChangeDecToHex(OP1, (Address mod 256));
       Show[CYInst].Stg:=Show[CYInst].Stg+OP1;
     end;
     Show[CYInst].Addr:=LastAddress;
     Address:=Temp;
     GetMemory(Address,Code);
     With AllIns[Code] do
      begin
        Show[CYInst].Stg:=Show[CYInst].Stg+' '+Nem;
        Case Acc of
          A : Show[CYInst].Stg:=Show[CYInst].Stg+' A  ';
          B : Show[CYInst].Stg:=Show[CYInst].Stg+' B  ';
          None : Show[CYInst].Stg:=Show[CYInst].Stg+'    ';
        end;
        Case AddMode of
          INH: Show[CYInst].Stg:=Show[CYInst].Stg+'          ';
          XXX: Show[CYInst].Stg:=Show[CYInst].Stg+' Memory    ';
          IMM: begin
                 GetMemory(Address,Num);
                 ChangeDecToHex(OP1,Num);
                 Show[CYInst].Stg:=Show[CYInst].Stg+' #$'+OP1;
                 If Nem[3] in ['X' , 'S' ]then
                 begin
                   GetMemory(Address,Num);
                   ChangeDecToHex(OP1,Num);
                   Show[CYInst].Stg:=Show[CYInst].Stg+OP1+'  ';
                 end
                 else
                   Show[CYInst].Stg:=Show[CYInst].Stg+'    ';
               end;
          DIR: begin
                 GetMemory(Address,Num);
                 ChangeDecToHex(OP1,Num);
                 Show[CYInst].Stg:=Show[CYInst].Stg+' $'+OP1+'       ';
               end;
          IND: begin
                 GetMemory(Address,Num);
                 ChangeDecToHex(OP1,Num);
```

```
                Show[CYInst].Stg:=Show[CYInst].Stg+' $'+OP1+',X     ';
              end;
         EXT: begin
                GetMemory(Address,Num);
                ChangeDecToHex(OP1,Num);
                Show[CYInst].Stg:=Show[CYInst].Stg+' $'+OP1;
                GetMemory(Address,Num);
                ChangeDecToHex(OP1,Num);
                Show[CYInst].Stg:=Show[CYInst].Stg+OP1+'    ';
              end;
         REL: begin
                GetMemory(Address,Num);
                If Num>PosNum then
                   Num:=Address+(Num-256)
                else
                   Num:=Address+Num;
                Show[CYInst].Stg:=Show[CYInst].Stg+' $';
                If Num>0 then
                begin
                  ChangeDecToHex(OP1, (Num Div 256));;
                  Show[CYInst].Stg:=Show[CYInst].Stg+OP1;
                  ChangeDecToHex(OP1, (Num mod 256));
                  Show[CYInst].Stg:=Show[CYInst].Stg+OP1+'    ';
                end
                else
                begin
                  Num:=Num+1+MaxNum;
                  ChangeDecToHex(OP1, (Num Div 256)+128);;
                  Show[CYInst].Stg:=Show[CYInst].Stg+OP1;
                  ChangeDecToHex(OP1, (Num mod 256));
                  Show[CYInst].Stg:=Show[CYInst].Stg+OP1+'    ';
                end;

              end;

       end;
     end;
     CYInst:=CYInst+1;

  end;

end;
{-------------------------------------------------}
Procedure DoShowInstList;
Var
  II      : Integer;
begin
 For II:=2 to MaxInstL do
 begin
   GotoXy(1,II); Write(Show[II].BKPoint,' ');
   If Show[II].Addr=PC.DecValue then
   begin
      TextAttr:=TextInv;
      ShowYPos:=II;
      Case Show[II].BKPoint of
       ' ': BreakPoint:=NoBp;
       #16: BreakPoint:=StrickBp;
       #175: BreakPoint:=NonStrickBp;
      end;
   end;
   Write(Show[II].Stg); TextAttr:=TextNorm;
 end;
end;
{-------------------------------------------------}
Function InAddressLink(Address : Integer): Boolean;
Var
```

```
  Temp      : AddressLinkPointerType;
  Found     : Boolean;
begin
  Temp:=HeadAddressLink^.Next;
  Found:=False;
  While (Temp<>Nil) and Not Found do
  begin
    Found:=(Address>=Temp^.ATopAddress) and (Address<=Temp^.ALastAddress);
    Temp:=Temp^.Next;
  end;
  If Found then
  begin
    Temp:=Temp^.Back;
    TopAddress:=Temp^.ATopAddress
  end;
  InAddressLink:=Found;
end;
{-----------------------------------------------------}
Procedure MakeNewAddressLink;
begin
  New(CurrentAddressLink^.Next);
  With CurrentAddressLink^.Next^ do
  begin
    Back:=CurrentAddressLink;
    Next:=Nil;
    ATopAddress:=TopAddress;
    ALastAddress:=LastAddress;
  end;
  CurrentAddressLink:=CurrentAddressLink^.Next;
end;
{-----------------------------------------------------}
Function InList (Address  : Integer) : Boolean;
Var
  Found   : Boolean;
  II      : Integer;
begin
  II:=2;
  Found:=False;
  While (II<MaxInstL) and Not Found do
  begin
    Found:=Show[II].Addr=Address;
    II:=II+1;
  end;
  InList:=Found;
end;
{-----------------------------------------------------}
Procedure ShowInst;
Var
  Num, Code, CYInst, Address, Temp  : integer;
  Fin,XFin           : Boolean;
  II : Byte;

begin
  Fin:=False;
  If ((PC.DecValue>=TopAddress) and (PC.DecValue <=LastAddress)) or
     ((PC.DecValue<=TopAddress) and (PC.DecValue >=LastAddress))then
  begin
       Fin:=True;
       II:=1;
       Repeat
         Inc(II);
       Until (II>=MaxInstL) or (Show[II].addr=PC.DecValue);
       XFin:=Show[II].addr=PC.DecValue;
       If Not XFin Then LastAddress:=PC.DecValue;
  end
  else     If ((PC.DecValue>=0) and (LastAddress>=0)
```

```
            and (PC.DecValue>TopAddress)
            and (PC.DecValue<(LastAddress+CodeViewRange*3)))
          or((PC.DecValue<0) and (LastAddress<0)
             and (PC.DecValue<TopAddress)
             and (PC.DecValue>(TopAddress-CodeViewRange*3)))
          then
          begin
           MakeNewAddressLink;
           TopAddress:=LastAddress;
           MakeInstList;
           If Not InList(PC.DecValue) then
           begin
             MakeNewAddressLink;
             TopAddress:=PC.DecValue;
             MakeInstList;
           end

          end
          else
          begin
           If InAddressLink(PC.DecValue) then
           begin
             MakeInstList;
             If Not InList(PC.DecValue) then
             begin
               MakeNewAddressLink;
               TopAddress:=PC.DecValue;
               MakeInstList;
             end;
           end
           else
           begin
             MakeNewAddressLink;
             TopAddress:=PC.DecValue;
             MakeInstList;
           end;
          end;

 IF not Fin then
    DoShowInstList
 else
 begin
   Code:=1;
   Repeat
     Inc(Code);
   Until (Show[Code].Addr=PC.DecValue) or (Code>MaxInstL);
   If Code<>ShowYPos then
   begin
     Num:=(ShowYPos-1)*80+2;
     Temp:=Num+19;
     For Address:=Num to temp do
         Screen^[Address].AtCh:=TextNorm;
     Num:=(Code-1)*80+2;
     Temp:=Num+19;
     For Address:=Num to temp do
         Screen^[Address].AtCh:=TextINV;
     ShowYPos:=Code;
     Case Show[Code].BKPoint of
       ' ': BreakPoint:=NoBp;
       #16: BreakPoint:=StrickBp;
       #175: BreakPoint:=NonStrickBp;
      end;
   end;
 end;
end;
{-------------------------------------------------}
```

```
Procedure DisPlay;
Const
  MemLS                          = 8;
Var
  ASCI                           : Packed Array[1..MemLs] of Char;
  II, Weight, Address,
  Num, III, IV, WinM             : integer;
  Hex                            : HexFormat2;
  CCBit                          : BitofCCType;
  ANum                           : RegisterType;

Function OutDecValue(Num : integer): Char;
begin
  If Num in [0,7..10,13,127,255] then Num:=176;
  OutDecValue:=Chr(Num);
end;
begin
  TextAttR:=TextNorm;
  GotoXy(35,3);
  II:=192;
  Weight:=32;
{====== OutPut (CC) ============================}
  For CCBit:= H to C do
  begin
    Write(CC[CCbit]);
    II:=CC[CCbit]*Weight+II;
    Weight:=Weight shr 1;
  end;
  ChangeDecToHex(Hex,II);
  Write(':',Hex);
  GotoXy(48,3);
  For CCBit:= H to V do
  begin
    Write(CC[CCbit],'  ');
  end;
  Write(CC[C]);

{====== OutPut (Accumulator) ===================}
  GotoXy(33,4);
  With Accum[A] do
  begin
    For II:=7 downto 0 do
      Write(LowerByteB[II]);
    Write(':',LowerByteH,':',OutDecValue(DecValue));
  end;
  GotoXy(33,5);
  With Accum[B] do
  begin
    For II:=7 downto 0 do
      Write(LowerByteB[II]);
    Write(':',LowerByteH,':',OutDecValue(DecValue)) ;
  end;

{====== OutPut (X) =============================}
  GotoXy(33,6);
  With X do
  begin
    For II:=7 Downto 0 do
      Write(HighByteB[II]);
    For II:=7 Downto 0 do
      Write(LowerByteB[II]);
    Write(':',HighByteH,LowerByteH,':',OutDecValue(GetDecFromHex(HighByteH)),
    OutDecValue(GetDecFromHex(LowerByteH)));
  end;

{====== Show Instructions List ================================}
```

```
  ANum.NumByte:=1;
  ShowInst;
  TextAttR:=TextNorm;

{====== OutPut Interrupt Bits ==============================}
  GotoXy(49,5);
  If IRQF then write('0') else Write('1');
  GotoXy(55,5);
  write(NMILV:1);
  GotoXy(61,5);
  If RESF then write('0') else Write('1');
  GotoXy(65,5); Write('           ');
  GotoXy(65,5); Write(Cycles:10);
{====== OutPut (PC) ============================}
  GotoXy(33,7);
  With PC do
  begin
    Write(HighByteH,LowerByteH);
    Address:=GetDecFromHex(HighByteH)*256+GetDecFromHex(LowerByteH);
  end;
  GotoXy(41,7); Write('');
  Address:=PC.DecValue;
  For II:=1 to 8 do
  begin
    With ANum do
    begin
      GetMemory(Address,DecValue);
      ChangeDectoHex(LowerByteH,DecValue);
      Write(LowerByteH,' ');
    end;
  end;
{=================================================}
{====== OutPut (SP) ============================}
  GotoXy(33,8);
  With SP do
  begin
    Write(HighByteH,LowerByteH);
    Address:=GetDecFromHex(HighByteH)*256+GetDecFromHex(LowerByteH);
  end;
  GotoXy(41,8); Write('');
  For II:=1 to MemLs do
  begin
    With ANum do
    begin
      GetMemory(Address,DecValue);
      ChangeDectoHex(LowerByteH,DecValue);
      ASCI[II]:=OutDecValue(DecValue);
      Write(LowerByteH,' ');
    end;
  end;
  GotoXy(42+3*MemLs,8); write(ASCI);
  GotoXy(41,9); Write('');
  For II:=1 to MemLs do
  begin
    With ANum do
    begin
      GetMemory(Address,DecValue);
      ChangeDectoHex(LowerByteH,DecValue);
      ASCI[II]:=OutDecValue(DecValue);
      Write(LowerByteH,' ');
    end;
  end;
  GotoXy(42+3*MemLs,9); write(ASCI);

{=================================================}
  WinM:=10;
```

```
  For IV:=1 to 3 do
  begin
    Address:=WinMemory[IV].Num;
    GotoXy(33,WinM+1); Write(WinMemory[IV].Hex1,WinMemory[IV].Hex2);
    GotoXy(41,WinM+1); Write('ฎ');
    For III:=1 to 2 do
    begin
      GotoXy(42,WinM+III);
      For II:=1 to MemLs do
      begin
        GetMemory(Address,Num);
        ChangeDecToHex(Hex,Num);
        ASCI[II]:=OutDecValue(Num);
        Write(Hex,' ');
      end;
      GotoXy(42+3*MemLs,WinM+III); write(ASCI);
    end;
    WinM:=WinM+3;
  end;

{%%%%%%%%%%%%%%}
{=====Up date Small Screen====}
  If UpDateSmallWin then
  begin
     Num:=UpDateValue;
     Window(MinWinX,MinWinY,MaxWinX,MaxWinY);
     GotoXy(CWinX,CWinY); Write(Chr(Num));
     CWinX:=WhereX; CWinY:=WhereY;
     Window(1,1,80,25);
     UpDateSmallWin:=False;
  end;
  DoBlink;
  GotoXy(80,25);
end;
{-------------------------------------------------}
```

9.9. File AllConst.Pas

```
Unit AllConst;
Interface
Const
   PosNum           = 127;
   MaxNum           = 32767;
   MinNum           = -32768;
   OutCh            = -2045;
   InCh             = -2048;
   InNoEcho         = -2047;
   RandJsr          = -2038;
   CodeViewRange    = 15;
   ErX              = 29;
   ErY              = 22;
   MaxWinX          = 27;
   MaxWinY          = 24;
   MinWinX          = 2;
   MinWinY          = 17;
   MaxInstL         = 15;
   XRead            = 29;
   YRead            = 24;
   XShowF           = 5;
   YShowF           = 25;
   SingleStep       = 1;
   Running          = 2;
   ColorInv         = 112;
   ColorNorm        = 15;
   EGAInv           = 78 {112};
   EGANorm          = 30 {15};
   MonoInv          = 120;
   MonoNorm         = 10;
   RealTimeX        = 55;
   RealTimeY        = 24;
   MaxBP            = 25;
   ComX             = 35;
   ComY             = 22;
   MaxWin           = 12;

Type
    InterNameType   = (RES,NMI,SWI,IRQ);
    BPType          = (NoBP, StrickBP, NonStrickBP);
    SpeedType       = (Fast,Mid,Slow);
    BPUseType       = Array [1..MaxBP] of
                        Record
                            Address : Integer;
                            BP      : BPType;
                        end;
    AddModeType     = (IMM, DIR, EXT, IND, INH, REL, XXX);
    AccNameType     = (A, B, None);
    BitofCCType     = (H, I, N, Z, V, C);
    BitType         = 0..1;
    ByteType        = 0..255;
    BinFormat8      = Packed array [0..7] of BitType;
    HexFormat2      = Packed array [1..2] of Char;
    NumByteType     = 1..2;
    CCType          = array[H..C] of bittype;
    NemunicType     = String[3];
    ShowType        = record
                        Stg   : String[20];
                        Addr  : Integer;
                        BKPoint: Char;
```

```
                    end;
    ShoWInstType    = Array[2..MaxInstL] of ShowType;
    InstructionType= record
                       Nem        : NemunicType;
                       Acc        : AccNameType;
                       AddMode    : AddModeType;
                       ExeTime    : Integer;
                      end;
    OperandType     = array[1..2] of Integer;
    RegisterType    = Record
                       DecValue  : Integer;
                       LowerByteB: BinFormat8;
                       LowerByteH: HexFormat2;
                       Case NumByte : NumByteType of
                            1 : ();
                            2 : ( HighByteB : BinFormat8;
                                  HighByteH : HexFormat2);
                      end;
    AccumulatorType = array[A..B] of RegisterType;
    UpMemoryPtr     = ^UpMemoryType;
    LowMemoryPtr    = Byte;
    UpMemoryType    = array [-32767..32767] of Byte;
{    LowMemoryType   = Byte;}
    WinMemoryType   = Record
                        Hex1, Hex2  : HexFormat2;
                        Num         : Integer;
                      end;
    WinMemoryNumType = Array[1..3] of WinMemoryType;
    AddressLinkPointerType= ^AddressLinkType;
    AddressLinkType  = Record
                         ATopAddress, ALastAddress: Integer;
                         Back, Next               : AddressLinkPointerType;
                       end;
    AddressValueList = array [1..24] of integer;
{=======================================================}
Const
    AllIns          : array [0..255] of InstructionType
                    = (
                        ( Nem: 'XXX'; Acc: None ;AddMode: XXX; ExeTime: 0 ),
                        ( Nem: 'NOP'; Acc: None ;AddMode: INH; ExeTime: 2 ),
                        ( Nem: 'XXX'; Acc: None ;AddMode: XXX; ExeTime: 0 ),
                        ( Nem: 'XXX'; Acc: None ;AddMode: XXX; ExeTime: 0 ),
                        ( Nem: 'XXX'; Acc: None ;AddMode: XXX; ExeTime: 0 ),
                        ( Nem: 'XXX'; Acc: None ;AddMode: XXX; ExeTime: 0 ),
                        ( Nem: 'TAP'; Acc: None ;AddMode: INH; ExeTime: 2 ),
                        ( Nem: 'TPA'; Acc: None ;AddMode: INH; ExeTime: 2 ),
                        ( Nem: 'INX'; Acc: None ;AddMode: INH; ExeTime: 4 ),
                        ( Nem: 'DEX'; Acc: None ;AddMode: INH; ExeTime: 4 ),
                        ( Nem: 'CLV'; Acc: None ;AddMode: INH; ExeTime: 2 ),
                        ( Nem: 'SEV'; Acc: None ;AddMode: INH; ExeTime: 2 ),
                        ( Nem: 'CLC'; Acc: None ;AddMode: INH; ExeTime: 2 ),
                        ( Nem: 'SEC'; Acc: None ;AddMode: INH; ExeTime: 2 ),
                        ( Nem: 'CLI'; Acc: None ;AddMode: INH; ExeTime: 2 ),
                        ( Nem: 'SEI'; Acc: None ;AddMode: INH; ExeTime: 2 ),
                        ( Nem: 'SBA'; Acc: A    ;AddMode: INH; ExeTime: 2 ),
                        ( Nem: 'CBA'; Acc: None ;AddMode: INH; ExeTime: 2 ),
                        ( Nem: 'XXX'; Acc: None ;AddMode: XXX; ExeTime: 0 ),
                        ( Nem: 'XXX'; Acc: None ;AddMode: XXX; ExeTime: 0 ),
                        ( Nem: 'XXX'; Acc: None ;AddMode: XXX; ExeTime: 0 ),
                        ( Nem: 'XXX'; Acc: None ;AddMode: XXX; ExeTime: 0 ),
                        ( Nem: 'TAB'; Acc: None ;AddMode: INH; ExeTime: 2 ),
                        ( Nem: 'TBA'; Acc: None ;AddMode: INH; ExeTime: 2 ),
                        ( Nem: 'XXX'; Acc: None ;AddMode: XXX; ExeTime: 0 ),
                        ( Nem: 'DAA'; Acc: None ;AddMode: INH; ExeTime: 2 ),
                        ( Nem: 'XXX'; Acc: None ;AddMode: XXX; ExeTime: 0 ),
                        ( Nem: 'ABA'; Acc: None ;AddMode: INH; ExeTime: 2 ),
```

```
( Nem: 'XXX'; Acc: None ;AddMode: XXX; ExeTime: 0 ),
( Nem: 'XXX'; Acc: None ;AddMode: XXX; ExeTime: 0 ),
( Nem: 'XXX'; Acc: None ;AddMode: XXX; ExeTime: 0 ),
( Nem: 'XXX'; Acc: None ;AddMode: XXX; ExeTime: 0 ),
( Nem: 'BRA'; Acc: None ;AddMode: REL; ExeTime: 4 ),
( Nem: 'XXX'; Acc: None ;AddMode: XXX; ExeTime: 0 ),
( Nem: 'BHI'; Acc: None ;AddMode: REL; ExeTime: 4 ),
( Nem: 'BLS'; Acc: None ;AddMode: REL; ExeTime: 4 ),
( Nem: 'BCC'; Acc: None ;AddMode: REL; ExeTime: 4 ),
( Nem: 'BCS'; Acc: None ;AddMode: REL; ExeTime: 4 ),
( Nem: 'BNE'; Acc: None ;AddMode: REL; ExeTime: 4 ),
( Nem: 'BEQ'; Acc: None ;AddMode: REL; ExeTime: 4 ),
( Nem: 'BVC'; Acc: None ;AddMode: REL; ExeTime: 4 ),
( Nem: 'BVS'; Acc: None ;AddMode: REL; ExeTime: 4 ),
( Nem: 'BPL'; Acc: None ;AddMode: REL; ExeTime: 4 ),
( Nem: 'BMI'; Acc: None ;AddMode: REL; ExeTime: 4 ),
( Nem: 'BGE'; Acc: None ;AddMode: REL; ExeTime: 4 ),
( Nem: 'BLT'; Acc: None ;AddMode: REL; ExeTime: 4 ),
( Nem: 'BGT'; Acc: None ;AddMode: REL; ExeTime: 4 ),
( Nem: 'BLE'; Acc: None ;AddMode: REL; ExeTime: 4 ),
( Nem: 'TSX'; Acc: None ;AddMode: INH; ExeTime: 4 ),
( Nem: 'INS'; Acc: None ;AddMode: INH; ExeTime: 4 ),
( Nem: 'PUL'; Acc: A    ;AddMode: INH; ExeTime: 4 ),
( Nem: 'PUL'; Acc: B    ;AddMode: INH; ExeTime: 4 ),
( Nem: 'DES'; Acc: None ;AddMode: INH; ExeTime: 4 ),
( Nem: 'TXS'; Acc: None ;AddMode: INH; ExeTime: 4 ),
( Nem: 'PSH'; Acc: A    ;AddMode: INH; ExeTime: 4 ),
( Nem: 'PSH'; Acc: B    ;AddMode: INH; ExeTime: 4 ),
( Nem: 'XXX'; Acc: None ;AddMode: XXX; ExeTime: 0 ),
( Nem: 'RTS'; Acc: None ;AddMode: INH; ExeTime: 5 ),
( Nem: 'XXX'; Acc: None ;AddMode: XXX; ExeTime: 0 ),
( Nem: 'RTI'; Acc: None ;AddMode: INH; ExeTime: 10 ),
( Nem: 'XXX'; Acc: None ;AddMode: XXX; ExeTime: 0 ),
( Nem: 'XXX'; Acc: None ;AddMode: XXX; ExeTime: 0 ),
( Nem: 'WAI'; Acc: None ;AddMode: INH; ExeTime: 9 ),
( Nem: 'SWI'; Acc: None ;AddMode: INH; ExeTime: 12 ),
( Nem: 'NEG'; Acc: A    ;AddMode: INH; ExeTime: 2 ),
( Nem: 'XXX'; Acc: None ;AddMode: XXX; ExeTime: 0 ),
( Nem: 'XXX'; Acc: None ;AddMode: XXX; ExeTime: 0 ),
( Nem: 'COM'; Acc: A    ;AddMode: INH; ExeTime: 2 ),
( Nem: 'LSR'; Acc: A    ;AddMode: INH; ExeTime: 2 ),
( Nem: 'XXX'; Acc: None ;AddMode: XXX; ExeTime: 0 ),
( Nem: 'ROR'; Acc: A    ;AddMode: INH; ExeTime: 2 ),
( Nem: 'ASR'; Acc: A    ;AddMode: INH; ExeTime: 2 ),
( Nem: 'ASL'; Acc: A    ;AddMode: INH; ExeTime: 2 ),
( Nem: 'ROL'; Acc: A    ;AddMode: INH; ExeTime: 2 ),
( Nem: 'DEC'; Acc: A    ;AddMode: INH; ExeTime: 2 ),
( Nem: 'XXX'; Acc: None ;AddMode: XXX; ExeTime: 0 ),
( Nem: 'INC'; Acc: A    ;AddMode: INH; ExeTime: 2 ),
( Nem: 'TST'; Acc: A    ;AddMode: INH; ExeTime: 2 ),
( Nem: 'XXX'; Acc: None ;AddMode: XXX; ExeTime: 0 ),
( Nem: 'CLR'; Acc: A    ;AddMode: INH; ExeTime: 2 ),
( Nem: 'NEG'; Acc: B    ;AddMode: INH; ExeTime: 2 ),
( Nem: 'XXX'; Acc: None ;AddMode: XXX; ExeTime: 0 ),
( Nem: 'XXX'; Acc: None ;AddMode: XXX; ExeTime: 0 ),
( Nem: 'COM'; Acc: B    ;AddMode: INH; ExeTime: 2 ),
( Nem: 'LSR'; Acc: B    ;AddMode: INH; ExeTime: 2 ),
( Nem: 'XXX'; Acc: None ;AddMode: XXX; ExeTime: 0 ),
( Nem: 'ROR'; Acc: B    ;AddMode: INH; ExeTime: 2 ),
( Nem: 'ASR'; Acc: B    ;AddMode: INH; ExeTime: 2 ),
( Nem: 'ASL'; Acc: B    ;AddMode: INH; ExeTime: 2 ),
( Nem: 'ROL'; Acc: B    ;AddMode: INH; ExeTime: 2 ),
( Nem: 'DEC'; Acc: B    ;AddMode: INH; ExeTime: 2 ),
( Nem: 'XXX'; Acc: None ;AddMode: XXX; ExeTime: 0 ),
( Nem: 'INC'; Acc: B    ;AddMode: INH; ExeTime: 2 ),
( Nem: 'TST'; Acc: B    ;AddMode: INH; ExeTime: 2 ),
```

```
( Nem: 'XXX'; Acc: None ;AddMode: XXX; ExeTime: 0 ),
( Nem: 'CLR'; Acc: B    ;AddMode: INH; ExeTime: 2 ),
( Nem: 'NEG'; Acc: None ;AddMode: IND; ExeTime: 7 ),
( Nem: 'XXX'; Acc: None ;AddMode: XXX; ExeTime: 0 ),
( Nem: 'XXX'; Acc: None ;AddMode: XXX; ExeTime: 0 ),
( Nem: 'COM'; Acc: None ;AddMode: IND; ExeTime: 7 ),
( Nem: 'LSR'; Acc: None ;AddMode: IND; ExeTime: 7 ),
( Nem: 'XXX'; Acc: None ;AddMode: XXX; ExeTime: 0 ),
( Nem: 'ROR'; Acc: None ;AddMode: IND; ExeTime: 7 ),
( Nem: 'ASR'; Acc: None ;AddMode: IND; ExeTime: 7 ),
( Nem: 'ASL'; Acc: None ;AddMode: IND; ExeTime: 7 ),
( Nem: 'ROL'; Acc: None ;AddMode: IND; ExeTime: 7 ),
( Nem: 'DEC'; Acc: None ;AddMode: IND; ExeTime: 7 ),
( Nem: 'XXX'; Acc: None ;AddMode: XXX; ExeTime: 0 ),
( Nem: 'INC'; Acc: None ;AddMode: IND; ExeTime: 7 ),
( Nem: 'TST'; Acc: None ;AddMode: IND; ExeTime: 7 ),
( Nem: 'JMP'; Acc: None ;AddMode: IND; ExeTime: 4 ),
( Nem: 'CLR'; Acc: None ;AddMode: IND; ExeTime: 7 ),
( Nem: 'NEG'; Acc: None ;AddMode: EXT; ExeTime: 6 ),
( Nem: 'XXX'; Acc: None ;AddMode: XXX; ExeTime: 0 ),
( Nem: 'XXX'; Acc: None ;AddMode: XXX; ExeTime: 0 ),
( Nem: 'COM'; Acc: None ;AddMode: EXT; ExeTime: 6 ),
( Nem: 'LSR'; Acc: None ;AddMode: EXT; ExeTime: 6 ),
( Nem: 'XXX'; Acc: None ;AddMode: XXX; ExeTime: 0 ),
( Nem: 'ROR'; Acc: None ;AddMode: EXT; ExeTime: 6 ),
( Nem: 'ASR'; Acc: None ;AddMode: EXT; ExeTime: 6 ),
( Nem: 'ASL'; Acc: None ;AddMode: EXT; ExeTime: 6 ),
( Nem: 'ROL'; Acc: None ;AddMode: EXT; ExeTime: 6 ),
( Nem: 'DEC'; Acc: None ;AddMode: EXT; ExeTime: 6 ),
( Nem: 'XXX'; Acc: None ;AddMode: XXX; ExeTime: 0 ),
( Nem: 'INC'; Acc: None ;AddMode: EXT; ExeTime: 6 ),
( Nem: 'TST'; Acc: None ;AddMode: EXT; ExeTime: 6 ),
( Nem: 'JMP'; Acc: None ;AddMode: EXT; ExeTime: 3 ),
( Nem: 'CLR'; Acc: None ;AddMode: EXT; ExeTime: 6 ),
( Nem: 'SUB'; Acc: A    ;AddMode: IMM; ExeTime: 2 ),
( Nem: 'CMP'; Acc: A    ;AddMode: IMM; ExeTime: 2 ),
( Nem: 'SBC'; Acc: A    ;AddMode: IMM; ExeTime: 2 ),
( Nem: 'XXX'; Acc: None ;AddMode: XXX; ExeTime: 0 ),
( Nem: 'AND'; Acc: A    ;AddMode: IMM; ExeTime: 2 ),
( Nem: 'BIT'; Acc: A    ;AddMode: IMM; ExeTime: 2 ),
( Nem: 'LDA'; Acc: A    ;AddMode: IMM; ExeTime: 2 ),
( Nem: 'XXX'; Acc: None ;AddMode: XXX; ExeTime: 0 ),
( Nem: 'EOR'; Acc: A    ;AddMode: IMM; ExeTime: 2 ),
( Nem: 'ADC'; Acc: A    ;AddMode: IMM; ExeTime: 2 ),
( Nem: 'ORA'; Acc: A    ;AddMode: IMM; ExeTime: 2 ),
( Nem: 'ADD'; Acc: A    ;AddMode: IMM; ExeTime: 2 ),
( Nem: 'CPX'; Acc: None ;AddMode: IMM; ExeTime: 3 ),
( Nem: 'BSR'; Acc: None ;AddMode: REL; ExeTime: 8 ),
( Nem: 'LDS'; Acc: None ;AddMode: IMM; ExeTime: 3 ),
( Nem: 'XXX'; Acc: None ;AddMode: XXX; ExeTime: 0 ),
( Nem: 'SUB'; Acc: A    ;AddMode: DIR; ExeTime: 3 ),
( Nem: 'CMP'; Acc: A    ;AddMode: DIR; ExeTime: 3 ),
( Nem: 'SBC'; Acc: A    ;AddMode: DIR; ExeTime: 3 ),
( Nem: 'XXX'; Acc: None ;AddMode: XXX; ExeTime: 0 ),
( Nem: 'AND'; Acc: A    ;AddMode: DIR; ExeTime: 3 ),
( Nem: 'BIT'; Acc: A    ;AddMode: DIR; ExeTime: 3 ),
( Nem: 'LDA'; Acc: A    ;AddMode: DIR; ExeTime: 3 ),
( Nem: 'STA'; Acc: A    ;AddMode: DIR; ExeTime: 4 ),
( Nem: 'EOR'; Acc: A    ;AddMode: DIR; ExeTime: 3 ),
( Nem: 'ADC'; Acc: A    ;AddMode: DIR; ExeTime: 3 ),
( Nem: 'ORA'; Acc: A    ;AddMode: DIR; ExeTime: 3 ),
( Nem: 'ADD'; Acc: A    ;AddMode: DIR; ExeTime: 3 ),
( Nem: 'CPX'; Acc: None ;AddMode: DIR; ExeTime: 4 ),
( Nem: 'XXX'; Acc: None ;AddMode: XXX; ExeTime: 0 ),
( Nem: 'LDS'; Acc: None ;AddMode: DIR; ExeTime: 4 ),
( Nem: 'STS'; Acc: None ;AddMode: DIR; ExeTime: 5 ),
```

```
( Nem: 'SUB'; Acc: A    ;AddMode: IND; ExeTime: 5 ),
( Nem: 'CMP'; Acc: A    ;AddMode: IND; ExeTime: 5 ),
( Nem: 'SBC'; Acc: A    ;AddMode: IND; ExeTime: 5 ),
( Nem: 'XXX'; Acc: None ;AddMode: XXX; ExeTime: 0 ),
( Nem: 'AND'; Acc: A    ;AddMode: IND; ExeTime: 5 ),
( Nem: 'BIT'; Acc: A    ;AddMode: IND; ExeTime: 5 ),
( Nem: 'LDA'; Acc: A    ;AddMode: IND; ExeTime: 5 ),
( Nem: 'STA'; Acc: A    ;AddMode: IND; ExeTime: 6 ),
( Nem: 'EOR'; Acc: A    ;AddMode: IND; ExeTime: 5 ),
( Nem: 'ADC'; Acc: A    ;AddMode: IND; ExeTime: 5 ),
( Nem: 'ORA'; Acc: A    ;AddMode: IND; ExeTime: 5 ),
( Nem: 'ADD'; Acc: A    ;AddMode: IND; ExeTime: 5 ),
( Nem: 'CPX'; Acc: None ;AddMode: IND; ExeTime: 6 ),
( Nem: 'JSR'; Acc: None ;AddMode: IND; ExeTime: 8 ),
( Nem: 'LDS'; Acc: None ;AddMode: IND; ExeTime: 6 ),
( Nem: 'STS'; Acc: None ;AddMode: IND; ExeTime: 7 ),
( Nem: 'SUB'; Acc: A    ;AddMode: EXT; ExeTime: 4 ),
( Nem: 'CMP'; Acc: A    ;AddMode: EXT; ExeTime: 4 ),
( Nem: 'SBC'; Acc: A    ;AddMode: EXT; ExeTime: 4 ),
( Nem: 'XXX'; Acc: None ;AddMode: XXX; ExeTime: 0 ),
( Nem: 'AND'; Acc: A    ;AddMode: EXT; ExeTime: 4 ),
( Nem: 'BIT'; Acc: A    ;AddMode: EXT; ExeTime: 4 ),
( Nem: 'LDA'; Acc: A    ;AddMode: EXT; ExeTime: 4 ),
( Nem: 'STA'; Acc: A    ;AddMode: EXT; ExeTime: 5 ),
( Nem: 'EOR'; Acc: A    ;AddMode: EXT; ExeTime: 4 ),
( Nem: 'ADC'; Acc: A    ;AddMode: EXT; ExeTime: 4 ),
( Nem: 'ORA'; Acc: A    ;AddMode: EXT; ExeTime: 4 ),
( Nem: 'ADD'; Acc: A    ;AddMode: EXT; ExeTime: 4 ),
( Nem: 'CPX'; Acc: None ;AddMode: EXT; ExeTime: 5 ),
( Nem: 'JSR'; Acc: None ;AddMode: EXT; ExeTime: 9 ),
( Nem: 'LDS'; Acc: None ;AddMode: EXT; ExeTime: 5 ),
( Nem: 'STS'; Acc: None ;AddMode: EXT; ExeTime: 6 ),
( Nem: 'SUB'; Acc: B    ;AddMode: IMM; ExeTime: 2 ),
( Nem: 'CMP'; Acc: B    ;AddMode: IMM; ExeTime: 2 ),
( Nem: 'SBC'; Acc: B    ;AddMode: IMM; ExeTime: 2 ),
( Nem: 'XXX'; Acc: None ;AddMode: XXX; ExeTime: 0 ),
( Nem: 'AND'; Acc: B    ;AddMode: IMM; ExeTime: 2 ),
( Nem: 'BIT'; Acc: B    ;AddMode: IMM; ExeTime: 2 ),
( Nem: 'LDA'; Acc: B    ;AddMode: IMM; ExeTime: 2 ),
( Nem: 'XXX'; Acc: None ;AddMode: XXX; ExeTime: 0 ),
( Nem: 'EOR'; Acc: B    ;AddMode: IMM; ExeTime: 2 ),
( Nem: 'ADC'; Acc: B    ;AddMode: IMM; ExeTime: 2 ),
( Nem: 'ORA'; Acc: B    ;AddMode: IMM; ExeTime: 2 ),
( Nem: 'ADD'; Acc: B    ;AddMode: IMM; ExeTime: 2 ),
( Nem: 'XXX'; Acc: None ;AddMode: XXX; ExeTime: 0 ),
( Nem: 'XXX'; Acc: None ;AddMode: XXX; ExeTime: 0 ),
( Nem: 'LDX'; Acc: None ;AddMode: IMM; ExeTime: 3 ),
( Nem: 'XXX'; Acc: None ;AddMode: XXX; ExeTime: 0 ),
( Nem: 'SUB'; Acc: B    ;AddMode: DIR; ExeTime: 3 ),
( Nem: 'CMP'; Acc: B    ;AddMode: DIR; ExeTime: 3 ),
( Nem: 'SBC'; Acc: B    ;AddMode: DIR; ExeTime: 3 ),
( Nem: 'XXX'; Acc: None ;AddMode: XXX; ExeTime: 0 ),
( Nem: 'AND'; Acc: B    ;AddMode: DIR; ExeTime: 3 ),
( Nem: 'BIT'; Acc: B    ;AddMode: DIR; ExeTime: 3 ),
( Nem: 'LDA'; Acc: B    ;AddMode: DIR; ExeTime: 3 ),
( Nem: 'STA'; Acc: B    ;AddMode: DIR; ExeTime: 4 ),
( Nem: 'EOR'; Acc: B    ;AddMode: DIR; ExeTime: 3 ),
( Nem: 'ADC'; Acc: B    ;AddMode: DIR; ExeTime: 3 ),
( Nem: 'ORA'; Acc: B    ;AddMode: DIR; ExeTime: 3 ),
( Nem: 'ADD'; Acc: B    ;AddMode: DIR; ExeTime: 3 ),
( Nem: 'XXX'; Acc: None ;AddMode: XXX; ExeTime: 0 ),
( Nem: 'XXX'; Acc: None ;AddMode: XXX; ExeTime: 0 ),
( Nem: 'LDX'; Acc: None ;AddMode: DIR; ExeTime: 4 ),
( Nem: 'STX'; Acc: None ;AddMode: DIR; ExeTime: 5 ),
( Nem: 'SUB'; Acc: B    ;AddMode: IND; ExeTime: 5 ),
( Nem: 'CMP'; Acc: B    ;AddMode: IND; ExeTime: 5 ),
```

```
                    ( Nem: 'SBC'; Acc: B    ;AddMode: IND; ExeTime: 5 ),
                    ( Nem: 'XXX'; Acc: None ;AddMode: XXX; ExeTime: 0 ),
                    ( Nem: 'AND'; Acc: B    ;AddMode: IND; ExeTime: 5 ),
                    ( Nem: 'BIT'; Acc: B    ;AddMode: IND; ExeTime: 5 ),
                    ( Nem: 'LDA'; Acc: B    ;AddMode: IND; ExeTime: 5 ),
                    ( Nem: 'STA'; Acc: B    ;AddMode: IND; ExeTime: 6 ),
                    ( Nem: 'EOR'; Acc: B    ;AddMode: IND; ExeTime: 5 ),
                    ( Nem: 'ADC'; Acc: B    ;AddMode: IND; ExeTime: 5 ),
                    ( Nem: 'ORA'; Acc: B    ;AddMode: IND; ExeTime: 5 ),
                    ( Nem: 'ADD'; Acc: B    ;AddMode: IND; ExeTime: 5 ),
                    ( Nem: 'XXX'; Acc: None ;AddMode: XXX; ExeTime: 0 ),
                    ( Nem: 'XXX'; Acc: None ;AddMode: XXX; ExeTime: 0 ),
                    ( Nem: 'LDX'; Acc: None ;AddMode: IND; ExeTime: 6 ),
                    ( Nem: 'STX'; Acc: None ;AddMode: IND; ExeTime: 7 ),
                    ( Nem: 'SUB'; Acc: B    ;AddMode: EXT; ExeTime: 4 ),
                    ( Nem: 'CMP'; Acc: B    ;AddMode: EXT; ExeTime: 4 ),
                    ( Nem: 'SBC'; Acc: B    ;AddMode: EXT; ExeTime: 4 ),
                    ( Nem: 'XXX'; Acc: None ;AddMode: XXX; ExeTime: 0 ),
                    ( Nem: 'AND'; Acc: B    ;AddMode: EXT; ExeTime: 4 ),
                    ( Nem: 'BIT'; Acc: B    ;AddMode: EXT; ExeTime: 4 ),
                    ( Nem: 'LDA'; Acc: B    ;AddMode: EXT; ExeTime: 4 ),
                    ( Nem: 'STA'; Acc: B    ;AddMode: EXT; ExeTime: 5 ),
                    ( Nem: 'EOR'; Acc: B    ;AddMode: EXT; ExeTime: 4 ),
                    ( Nem: 'ADC'; Acc: B    ;AddMode: EXT; ExeTime: 4 ),
                    ( Nem: 'ORA'; Acc: B    ;AddMode: EXT; ExeTime: 4 ),
                    ( Nem: 'ADD'; Acc: B    ;AddMode: EXT; ExeTime: 4 ),
                    ( Nem: 'XXX'; Acc: None ;AddMode: XXX; ExeTime: 0 ),
                    ( Nem: 'XXX'; Acc: None ;AddMode: XXX; ExeTime: 0 ),
                    ( Nem: 'LDX'; Acc: None ;AddMode: EXT; ExeTime: 5 ),
                    ( Nem: 'STX'; Acc: None ;AddMode: EXT; ExeTime: 6 )
                );
{=====================================================================}
Const
   InterruptAddr    : Array [RES..IRQ] of Integer = (-2,-4,-6,-8);
   BPChar           : Array [NoBp..NonStrickBp] of Char =  (' ',#16,#175);


Var
   II, III, Save    : integer;
   CC               : CCType;
   Accum            : AccumulatorType;
   X, SP, PC        : RegisterType;
   MemoryUp         : UpMemoryPtr;
   MemoryLow        : LowMemoryPtr;
   Code, TheY       : Integer;
   Inst             : InstructionType ;
   Wait             : Boolean;
   UpdateSmallWin   : Boolean;
   UpDateValue      : Integer;
   CON              : Text;
   BPUse            : BPUseType;
   CWinX, CWinY,
   BreakMarkY,
   OldBreakMarkY,
   ShowYPos         : Integer;
   TopAddress,
   LastAddress,
   StartAddress,
   ShowPCAddr,
   RuningMode,
   TextInv,
   TextNorm         : Integer;
   Show             : ShowInstType;
   WinMemory        : WinMemoryNumType;
   BreakPoint       : BPType;
   Speed            : SpeedType;
```

```
   HeadAddressLink,
   CurrentAddressLink : AddressLinkPointerType;
   PTOOLWIN_Screen_Type : Char ;
   OLDCH,OLDCL : Byte;
   OldMode : Word;
   OldAlMode, OldBH : Byte;
   ListOfAddr  : AddressValueList;
Implementation
end.
```

9.10. File CurSor.Pas

```
Unit  Cursor;
Interface
Procedure Set_Cursor(Top, Bottom: Byte);
Procedure Get_Cursor(Var Top, Bottom : Byte; BPage : Byte);
Procedure Get_DisPlay(Var Dmode, BPage : Byte);
Procedure Set_DisPlay(DMode : Byte);
Procedure Set_Page(BPage : Byte);
Implementation
Uses Dos;
Procedure Set_Cursor;
Var
  RecPack : Registers;
begin
  With RecPack do
  begin
    AH:=1;
    CH:=Top;
    CL:=Bottom;
  end;
  Intr($10, RecPack);
end;
Procedure Set_Page;
Var
  RecPack : Registers;
begin
  With RecPack do
  begin
    AH:=5;
    AL:=BPage;
  end;
  Intr($10, RecPack);
end;
Procedure Set_DisPlay;
Var
  RecPack : Registers;
begin
  With RecPack do
  begin
    AH:=0;
    AL:=DMode;
  end;
  Intr($10, RecPack);
end;
Procedure Get_Cursor;
Var
  RecPack : Registers;
begin
  RecPack.AH:=3;
  RecPack.BH:=BPage;
  Intr($10, RecPack);
  With RecPack do
  begin
    Top:=CH;
    Bottom:=CL;
  end;
end;
Procedure Get_Display;
Var
  RecPack : Registers;
begin
```

```
  RecPack.AH:=15;
  Intr($10, RecPack);
  DMode:=RecPack.AL;
  BPage:=RecPack.BH;
end;
end.
```

9.11. File NewWin.Pas

```
Unit NewWin;
Interface
Uses Dos, Crt;
Type
   WSizeType     = Record
                     X,Y,H,W      : byte;
                     PSize        : word;
                     OldAttr      : word;
                     HaveFrame    : Boolean;
                     BackColor, ForeColor : Integer;
                     WSave         : Pointer;
                   end;
   Windows_type = array[1..50] of WSizeType;
Procedure SetWindow (WinName  : Byte; tX,tY,tW,tH : byte;
                     tBackColor, tForeColor : Integer;
                     DoFrame  : Boolean);
Procedure ShowWin (WinName : Byte);
Procedure CloseWin(WinName : Byte);
Procedure CloseLastWin;
Implementation
Type
   SingleType        = Record
                         Ch   : Char;
                         Atch : Byte;
                       end;
   ScreenType        = Array [0..1999] of SingleType;

   ScreenPtr         = ^ScreenType;
   StorageType       = Array[1..2000] of SingleType;
Var
   Windows        : Windows_Type;
   ST             : StorageType;
   Screen         : ScreenPtr;
   QueWin         : Array [1..50] of Byte;
   LastOne        : Byte;
   CScreen        : Char absolute $B800:$0000;
   MScreen        : Char absolute $B000:$0000;
   UPL,TOP,UPR,
   SIDEL,SIDER,
   LowL,BOT,LowR : Char;

  Procedure SetWindow;
  begin
    With Windows[WinName] do
    begin
      x:=tx;
      y:=ty;
      H:=th;
      W:=tW;
      BackColor:=tBackColor;
      ForeColor:=tForeColor;
      HaveFrame:=DoFrame;
      If DoFrame then
        PSize:=(H+2)*2*(W+2)
      else
        PSize:=H*2*W;
    end;
  end;
  Procedure SaveBack( var WSize : WSizeType    );
  Var
```

```
  C,Cy, CX,  I,II,SM, OFX   : word;

begin
  With WSize do
  begin
    Cy:=H;
    CX:=W;
    SM:=(Y-1)*80;
    OFX:=X;
  end;
  If WSize.HaveFrame then
  begin
    Inc(CX,2);
    Inc(CY,2);
    Dec(SM,80);
    Dec(OFX);
  end;
  GetMem(WSize.WSave,WSize.PSize);
  C:=1;
  For I:=0 to CY do
  begin
    For II:=0 to CX do
    begin
      ST[C]:=Screen^[SM+OFX+II-1];
      Inc(C);
    end;
    Inc(SM,80);
  end;
  Move(ST,WSize.WSave^,WSize.PSize);
end;
Procedure RestoreBack( var WSize: WSizeType);
Var
  C,Cy, CX,  I,II,SM, OFX   : word;
  P : Pointer;

begin
  With WSize do
  begin
    Cy:=H;
    CX:=W;
    SM:=(Y-1)*80;
    OFX:=X;
  end;
  If WSize.HaveFrame then
  begin
    Inc(CX,2);
    Inc(CY,2);
    Dec(SM,80);
    Dec(OFX);
  end;
  With WSize do
  begin
    Move(WSave^,ST,PSize);
    P:=WSave;
    FreeMem(P,PSize);
  end;
  C:=1;
  For I:=0 to CY do
  begin
    For II:=0 to CX do
    begin
      Screen^[SM+OFX+II-1]:=ST[C];
      Inc(C);
    end;
    Inc(SM,80);
  end;
```

```
  end;
  Procedure DrawFrame(WSize : WSizeType);
  Var
    SX,SY,EX,EY, I : Word;
  begin
   With WSize do
   begin
     If HaveFrame then
     begin
          SX:=X-1;
          SY:=Y-1;
          EX:=X+W+1;
          EY:=Y+H+1;
{          Crt.Textbackground(BackColor);
          Crt.TextColor(ForeColor);}
          GotoXy(SX,SY);
          Write(UPL);
          For I:=SX+1 to EX-1 do
            Write(TOP);
          Write(UPR);
          For I:=SY+1 to Ey-1 do
          begin
            GotoXY(SX,I); Write(SIDEL);
            GotoXy(EX,I); Write(SIDER);
          end;
          GotoXy(SX,EY);
          Write(LOWL);
          For I:=SX+1 to EX-1 do
            Write(BOT);

  {        Write(LOWR);}
         I:=Pred(Ey)*80+Pred(Ex);
         With Screen^[I] do
         begin
           Ch:=LoWR;
           Atch:=Screen^[Pred(I)].Atch;
         end;
       end;
     end;
  end;
Procedure ShowWin;
Var
  EX,EY    : Byte;
begin
  SaveBack(Windows[WinName]);
  With Windows[WinName] do
  begin
    OldAttr:=TextAttr;
    EX:=X+W;
    EY:=Y+H;
    Crt.Textbackground(BackColor);
    Crt.TextColor(ForeColor);
    DrawFrame(Windows[WinName]);
    Window(X,Y,EX,EY);
    ClrScr;
  end;
  QueWin[LastOne]:=WinName;
  Inc(LastOne);
end;
Procedure CloseWin;
begin
  RestoreBack(Windows[WinName]);
  Window(1,1,80,25);
  TextAttr:=Windows[WinName].OldAttr;
  Dec(LastOne);
end;
```

```
Procedure CloseLastWin;
begin
  Dec(LastOne);
  RestoreBack(Windows[QueWin[LastOne]]);
  Window(1,1,80,25);
  TextAttr:=Windows[QueWin[LastOne]].OldAttr;
end;
begin
  Screen:=@CScreen;
  If LastMode=Mono then Screen:=@MScreen;
  UPL:='ษ';
  TOP:='อ';
  UPR:='ป';
  SIDEL:='บ';
  SIDER:='บ';
  LowL:='ศ';
  BOT:='อ';
  LowR:='ผ';
  LastOne:=1;
end.
```

9.12. File Scancode.Pas

```
Unit ScanCode;
Interface
Const
  Nul          = #3;
  ShfTab       = #15;
  AltQ         = #16;
  AltW         = #17;
  AltE         = #18;
  AltR         = #19;
  AltT         = #20;
  AltY         = #21;
  AltU         = #22;
  AltI         = #23;
  AltO         = #24;
  AltP         = #25;
  AltA         = #30;
  AltS         = #31;
  AltD         = #32;
  AltF         = #33;
  AltG         = #34;
  AltH         = #35;
  AltJ         = #36;
  AltK         = #37;
  AltL         = #38;
  AltZ         = #44;
  AltX         = #45;
  AltC         = #46;
  AltV         = #47;
  AltB         = #48;
  AltN         = #49;
  AltM         = #50;
  KF1          = #59;
  KF2          = #60;
  KF3          = #61;
  KF4          = #62;
  KF5          = #63;
  KF6          = #64;
  KF7          = #65;
  KF8          = #66;
  KF9          = #67;
  KF10         = #68;
  KHOME        = #71;
  KUP          = #72;
  KPGUP        = #73;
  KLEFT        = #75;
  KRIGHT       = #77;
  KEND         = #79;
  KDOWN        = #80;
  KPGDN        = #81;
  KINS         = #82;
  KDEL         = #83;
  SHFKF1       = #84;
  SHFKF2       = #85;
  SHFKF3       = #86;
  SHFKF4       = #87;
  SHFKF5       = #88;
  SHFKF6       = #89;
  SHFKF7       = #90;
  SHFKF8       = #91;
  SHFKF9       = #92;
```

```
  SHFKF10       = #93;
  CtrKF1        = #94;
  CtrKF2        = #95;
  CtrKF3        = #96;
  CtrKF4        = #97;
  CtrKF5        = 98;
  CtrKF6        = #99;
  CtrKF7        = #100;
  CtrKF8        = #101;
  CtrKF9        = #102;
  CtrKF10       = #103;
  AltKF1        = #104;
  AltKF2        = #105;
  AltKF3        = #106;
  AltKF4        = #107;
  AltKF5        = #108;
  AltKF6        = #109;
  AltKF7        = #110;
  AltKF8        = #111;
  AltKF9        = #112;
  AltKF10       = #113;
  CtrlPrtSc     = #114;
  CtrlLeft      = #115;
  CtrlRight     = #116;
  CtrlEnd       = #117;
  CtrlPgDn      = #118;
  CtrlHome      = #119;

implementation
end.
```