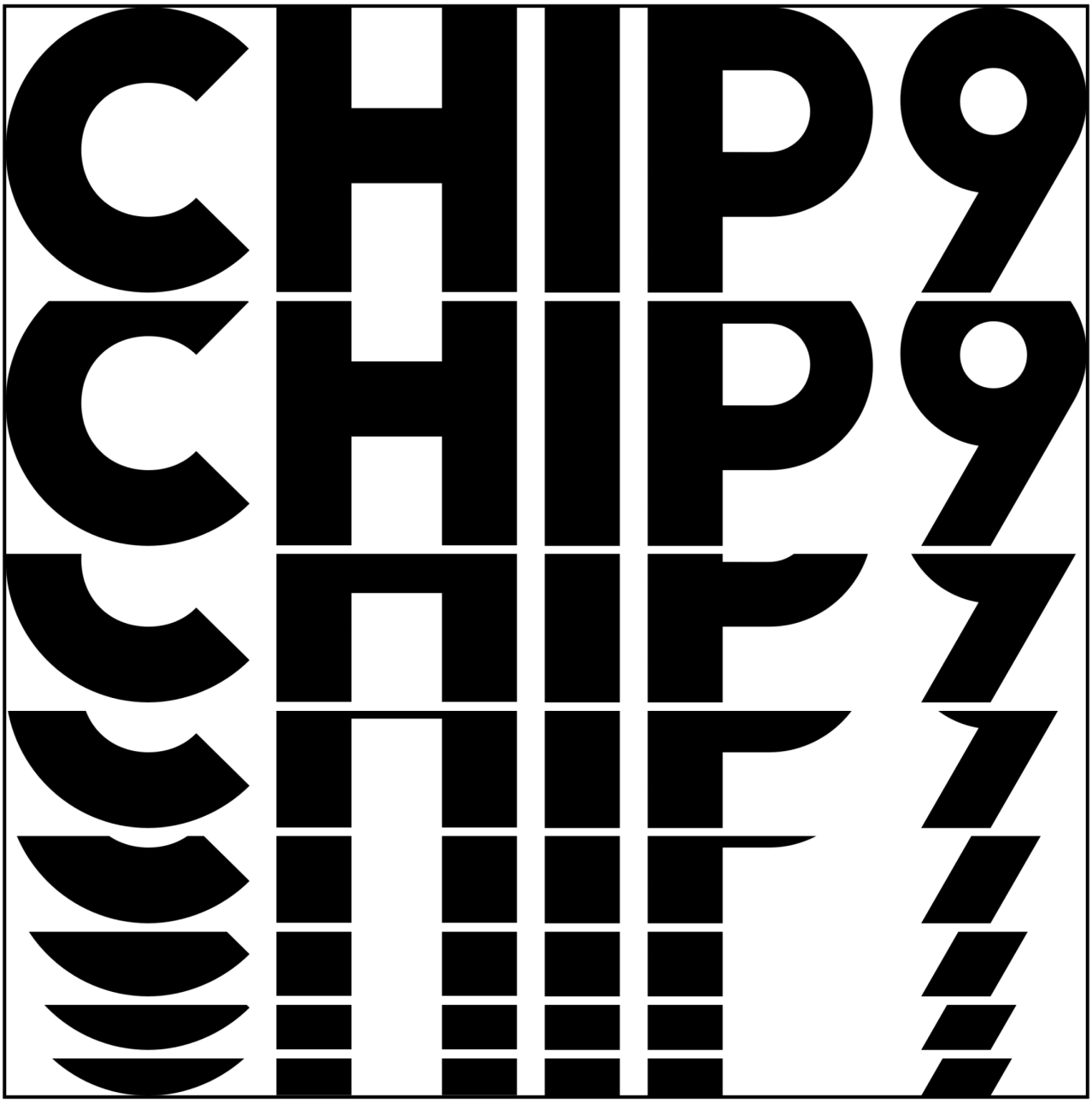
IPB

Reference Manual
IPB Chip 9 Data Processing System



Rev. A

This edition, A24-9901-5, obsoletes A24-9901-4 and all earlier editions. Significant changes have been made throughout the manual, and this edition should be reviewed in its entirety.
This edition also obsoletes these publications:

N24-0001 No. 1 Oct. 23, 1976
N24-0002 No. 1 Oct. 30, 1976
N24-0003 No. 1 Oct. 31, 1976
N24-0009 No. 1 Dec. 18, 1976
N24-0010 No. 1 Dec. 18, 1976
N24-0018 No. 1 Feb. 28, 1977

Preface

This manual is a reference text for the IPB CHIP9 Data Processing System. It provides a detailed explanation of operating codes and the function of the system's components. The reader should have a knowledge of the CHIP9 system and programming techniques.

This manual is divided into these sections:

- system features
- operation codes
- boot process

The sections are independent and need not be used in the order in which they appear.

This manual is intended for the use of programmers and systems personnel who do not know the inner-workings of the IPB CHIP9 Data Processing System. The manual can also be used as a training aid in the instruction of programmers and operators.

Terminology

- Op Code: This is always a single byte that defines the operation to be performed.
- Address: This always consists of a 16-bit number, and represents a byte in memory.

Contents

Section 1

System Features

1.1 Specification

- CPU: 8-bit little-endian (Similar to the i8080 processor)
- Main RAM: 64K Byte
- Screen Resolution: 128x64 1-bit pixels, @60Hz
- Power: DC9V 2.5W

1.2 Processor

The manipulation that data undergoes in order to achieve desired results is called *processing*, and the part of the CHIP9 system that houses these operations is called a *processing unit*. This processor is constructed to accomplish a wide variety of tasks, thanks to the performance of its arithmetic logic unit (ALU).

1.3 Memory

The entire 64K of random-access memory is freely available across its entire addressing space, which ranges from 0x0000 to 0xFFFF included.

1.4 Registers

The IPB CHIP9 processor houses 7 8-bit registers: A, B, C, D, E, H and L. All registers except 'A' can be combined in order to form the 16-bit *register pairs* BC, DE and HL. These register pairs may be used interchangeably with their individual register counterparts.

In addition to the 7 general-purpose registers, the IPB CHIP9 processor also has a 16-bit Stack Pointer (SP), a 16-bit Program Counter (PC) and an 8-bit Flag Register (F), which may not be accessed directly.

1.4.1 Stack Pointer

The Stack Pointer is used to keep track of the top of the "stack". The stack is used for saving variables, saving return addresses, passing arguments to subroutines, and various other uses that might be conceived by the individual programmer. The instructions CALL and PUSH all put information onto the stack. The instructions POP and RET all take information off of the stack.

As information is put onto the stack, the stack grows downward in RAM memory. As a result, the Stack Pointer should always be initialized at the highest location of RAM space that has been allocated for use by the stack. For instance, if a programmer wishes to locate the Stack Pointer at the top of RAM space (0x0000 - 0xFFFF), he would set the Stack Pointer to 0xFFFE using the command LDX SP, 0xFFFE (or 22 FE FF in machine code). The stack has to be always aligned with even memory addresses.

1.4.2 Program Counter

The Program Counter represents the address in RAM of the current instruction that needs to be executed. Once this instruction has been executed, PC will represent the address of the next instruction in memory, if no jump has occurred.

On power up, the IPB CHIP9 Program Counter is initialized to 0x0000 and the instruction found at this location in RAM is executed.

1.4.3 Flag Register

The Flag Register consists of the following bits:

7	6	5	4	3	2	1	0
Z	N	H	C	0	0	0	0

Zero Flag (Z)

This bit is set when the result of a math operation is zero or two values match when using the CMP instruction.

Negative Flag (N)

This bit is set when the result of a math operation is negative (ie has bit 7 set).

Half-Carry Flag (H)

This bit is set if a carry occurred from the lower nibble in the last math operation.

Carry Flag (C)

This bit is set if a carry occurred from the last math operation.

1.5 Logic

The logic function of any kind of data processing system is the ability to execute program steps; but even more, the ability to evaluate conditions and select alternative program steps on the basis of those conditions.

The CHIP9 processor has multiple instructions for branching as well as *function calling* through the use of CALL and RET instructions.

1.6 I/O

The processor has full Serial I/O capabilities, through the use of both Serial IN and Serial OUT instructions. However, most user interaction will be done through the use of its Graphics Display capabilities, and Serial I/O should be used only for debugging purposes only.

Section 2

Operation Codes

Terminology

- R: Any general purpose register.
- RR: Any register pair.
- x: Any 1-bit value.
- xx: Any 8-bit value.
- xxyy: Any 16-bit value.
- (RR): The value in memory at the address pointed by the register pair RR.
- f: Any flag (Z, N, H, C) from the Flag Register.
- cc: Any condition (ie flag f is true or false).

2.1 Memory Operations

2.1.1 Loads

LDI R, xx

This instruction puts the value xx into the register R.

Instruction	Opcode
LDI B, xx	20 xx
LDI C, xx	30 xx
LDI D, xx	40 xx
LDI E, xx	50 xx
LDI H, xx	60 xx
LDI L, xx	70 xx
LDI (HL), xx	80 xx
LDI A, xx	90 xx

LDX RR, xxyy

This instruction puts the value xxyy into the 16-bit register RR.

For example, if RR is BC and xxyy is 0x1337, register B will have the value 0x13 and C will have the value 0x37.

Instruction	Opcode
LDX BC, xxyy	21 yy xx
LDX DE, xxyy	31 yy xx
LDX HL, xxyy	41 yy xx
LDX SP, xxyy	22 yy xx

2.1.2 Stack pushes

PUSH R

This instruction pushes register R onto the stack.

For example, if SP is 0xFFFE, the 8-bit value of register R will be placed at 0xFFFE, then the stack pointer will decrease by 2 in order to keep the alignment. SP becomes 0xFFFC.

Instruction	Opcode
PUSH B	81
PUSH C	91
PUSH D	A1
PUSH E	B1
PUSH H	C1
PUSH L	D1
PUSH (HL)	C0
PUSH A	D0

PUSH RR

This instruction pushes the register pair RR onto the stack.

For example, if SP is 0xFFFE, and RR is BC, the 8-bit value of register C will be placed at 0xFFFE, the 8-bit value of register B will be placed at 0xFFFF, and SP will decrease by 2, thus becoming 0xFFFC.

Instruction	Opcode
PUSH BC	51
PUSH DE	61
PUSH HL	71

2.1.3 Stack pops

POP R

This instruction pops the register R from the stack.

For example, if SP is 0xFFFC, the stack pointer will first increase by 2, then the register R will have the value pointed currently by the stack pointer, which is the value at 0xFFFE.

Instruction	Opcode
POP B	82
POP C	92
POP D	A2
POP E	B2
POP H	C2
POP L	D2
POP (HL)	C3
POP A	D3

POP RR

This instruction pops the register pair RR from the stack.

For example, if SP is 0xFFFC, and RR is BC, the stack pointer will first increase by 2, then the register C will get the value stored at SP (0xFFFE) and the register B will get the value stored at SP + 1 (0xFFFF)

Instruction	Opcode
POP BC	52
POP DE	62
POP HL	72

2.1.4 Register Movement

MOV R1, R2

This instruction puts the value of R2 into R1.

Instruction	Opcode
MOV B, B	09
MOV B, C	19
MOV B, D	29
MOV B, E	39
MOV B, H	49
MOV B, L	59
MOV B, (HL)	69
MOV B, A	79

Instruction	Opcode
MOV C, B	89
MOV C, C	99
MOV C, D	A9
MOV C, E	B9
MOV C, H	C9
MOV C, L	D9
MOV C, (HL)	E9
MOV C, A	F9

Instruction	Opcode
MOV D, B	0A
MOV D, C	1A
MOV D, D	2A
MOV D, E	3A
MOV D, H	4A
MOV D, L	5A
MOV D, (HL)	6A
MOV D, A	7A

Instruction	Opcode
MOV E, B	8A
MOV E, C	9A
MOV E, D	AA
MOV E, E	BA
MOV E, H	CA
MOV E, L	DA
MOV E, (HL)	EA
MOV E, A	FA

Instruction	Opcode
MOV H, B	0B
MOV H, C	1B
MOV H, D	2B
MOV H, E	3B
MOV H, H	4B
MOV H, L	5B
MOV H, (HL)	6B
MOV H, A	7B

Instruction	Opcode
MOV L, B	8B
MOV L, C	9B
MOV L, D	AB
MOV L, E	BB
MOV L, H	CB
MOV L, L	DB
MOV L, (HL)	EB
MOV L, A	FB

Instruction	Opcode
MOV (HL), B	0C
MOV (HL), C	1C
MOV (HL), D	2C
MOV (HL), E	3C
MOV (HL), H	4C
MOV (HL), L	5C
HCF	6C
MOV (HL), A	7C

Instruction	Opcode
MOV A, B	8C
MOV A, C	9C
MOV A, D	AC
MOV A, E	BC
MOV A, H	CC
MOV A, L	DC
MOV A, (HL)	EC
MOV A, A	FC

MOV RR1, RR2

This instruction puts the value of register pair RR2 into register pair RR1.

Instruction	Opcode
MOV HL, BC	ED
MOV HL, DE	FD

2.2 Arithmetic

2.2.1 Flag Setting

CLRFLAG

This instruction sets all ZNHC flags to 0.

Instruction	Opcode
CLRFLAG	08

SETFLAG f, x

This instruction sets the value of flag f as x.

Instruction	Opcode
SETFLAG Z, 1	18
SETFLAG Z, 0	28
SETFLAG N, 1	38
SETFLAG N, 0	48
SETFLAG H, 1	58
SETFLAG H, 0	68
SETFLAG C, 1	78
SETFLAG C, 0	88

2.2.2 Addition

ADD R

This instruction adds the value of register R with the value of register A, and stores the result back in register R.

This instruction sets all ZNHC flags.

Instruction	Opcode
ADD B	04
ADD C	14
ADD D	24
ADD E	34
ADD H	44
ADD L	54
ADD (HL)	64
ADD A	74

ADDI xx

This instruction adds the value of register A with the value of xx, and stores the result back in register A.

This instruction sets all ZNHC flags.

Instruction	Opcode
ADDI xx	A7 xx

ADDX RR

This instruction adds the value of register pair RR with the value of register A, and stores the result back in register pair RR.

This instruction sets all ZNHC flags.

Instruction	Opcode
ADDX BC	83
ADDX DE	93
ADDX HL	A3

2.2.3 Subtraction

SUB R

This instruction subtracts the value of register R with the value of register A, and stores the result back in register R.

This instruction sets all ZNHC flags.

Instruction	Opcode
SUB B	84
SUB C	94
SUB D	A4
SUB E	B4
SUB H	C4
SUB L	D4
SUB (HL)	E4
SUB A	F4

SUBI xx

This instruction subtracts the value of register A with the value of xx, and stores the result back in register A.

This instruction sets all ZNHC flags.

Instruction	Opcode
SUBI xx	B7 xx

2.2.4 Increment

INC R

This instruction increments the value of register R.
This instruction sets all ZNHC flags.

Instruction	Opcode
INC B	03
INC C	13
INC D	23
INC E	33
INC H	43
INC L	53
INC (HL)	63
INC A	73

INX RR

This instruction increments the value of register pair RR.
This instruction does **NOT** affect any flags.

Instruction	Opcode
INX BC	A8
INX DE	B8
INX HL	C8

2.2.5 Decrement

DEC R

This instruction decrements the value of register R.
This instruction sets all ZNHC flags.

Instruction	Opcode
DEC B	07
DEC C	17
DEC D	27
DEC E	37
DEC H	47
DEC L	57
DEC (HL)	67
DEC A	77

2.3 Logical Operations

2.3.1 AND

AND R

This instruction performs a logical AND between the values of register R and register A, and stores the result back in register R.

This instruction sets the Z and N flags accordingly, and clears the H and C flags.

Instruction	Opcode
AND B	05
AND C	15
AND D	25
AND E	35
AND H	45
AND L	55
AND (HL)	65
AND A	75

ANDI xx

This instruction performs a logical AND between the values of register A and xx, and stores the result back in register A.

This instruction sets the Z and N flags accordingly, and clears the H and C flags.

Instruction	Opcode
ANDI xx	C7 xx

2.3.2 OR

OR R

This instruction performs a logical OR between the values of register R and register A, and stores the result back in register R.

This instruction sets the Z and N flags accordingly, and clears the H and C flags.

Instruction	Opcode
OR B	85
OR C	95
OR D	A5
OR E	B5
OR H	C5
OR L	D5
OR (HL)	E5
OR A	F5

ORI xx

This instruction performs a logical OR between the values of register A and xx, and stores the result back in register A.

This instruction sets the Z and N flags accordingly, and clears the H and C flags.

Instruction	Opcode
ORI xx	D7 xx

2.3.3 XOR

XOR R

This instruction performs a logical XOR between the values of register R and register A, and stores the result back in register R.

This instruction sets the Z and N flags accordingly, and clears the H and C flags.

Instruction	Opcode
XOR B	06
XOR C	16
XOR D	26
XOR E	36
XOR H	46
XOR L	56
XOR (HL)	66
XOR A	76

XORI xx

This instruction performs a logical XOR between the values of register A and xx, and stores the result back in register A.

This instruction sets the Z and N flags accordingly, and clears the H and C flags.

Instruction	Opcode
XORI xx	E7 xx

2.3.4 Comparisons

CMP R

This instruction compares the value of register R with that of register A, and sets the Z flag if they are equal, and the N flag if the value of register R is smaller than register A.

Internally, this instruction uses a subtraction between register R and register A, but the resulting values are ignored.

Instruction	Opcode
CMP B	86
CMP C	96
CMP D	A6
CMP E	B6
CMP H	C6
CMP L	D6
CMP (HL)	E6
CMP A	F6

CMPI xx

This instruction compares the value of register A with xx, and sets the Z flag if they are equal, and the N flag if the value of register A is smaller than xx.

Internally, this instruction uses a subtraction between register A and xx, but the resulting values are ignored.

Instruction	Opcode
CMPI A xx	F7 xx

CMPS R

This instruction does a **signed** comparison between register R and register A, meaning that it takes into account negative numbers (which have bit 7 set).

This instruction sets the Z flag if both values are equal, and the N flag if the value of register R is smaller than the value of register A.

Instruction	Opcode
CMP B	0D
CMP C	1D
CMP D	2D
CMP E	3D
CMP H	4D
CMP L	5D
CMP (HL)	6D
CMP A	7D

2.4 I/O

2.4.1 Serial

SIN

This instruction takes the current 8-bit value in the serial input buffer and stores it into register A.

Instruction	Opcode
SIN	E0

SOUT

This instruction prints the character with the ASCII value stored in register A.

Instruction	Opcode
SOUT	E1

2.4.2 Screen

CLRSCR

This instruction clears the screen, setting all pixels black.

Instruction	Opcode
CLRSCR	F0

DRAW

This instruction draws register A on the screen at the signed X coordinate stored in register C and at the signed Y coordinate stored in register B.

The drawing of register A is done by drawing a row of 8 pixels according to the binary representation of register A:

7	6	5	4	3	2	1	0
A7	A6	A5	A4	A3	A2	A1	A0

Where Ai is register A's i-th bit.

For example, if register B has the value 0x00, register C has the value 0x00 and register A has a value of 0xCD (0b11001101), this instruction will draw on the screen:

Y \ X	0	1	2	3	4	5	6	7	8	9	...
0	■	■			■	■		■			...
1											...
2											...
...	...	...	...	...	...	...	...	...	...	...	...

As another example, if register B has the value 0x00, register C has the value 0xFF (or -1) and register A has a value of 0xCD (0b11001101), the instruction will draw on the screen:

Y \ X	0	1	2	3	4	5	6	7	8	9	...
0	■			■	■		■				...
1											...
2											...
...	...	...	...	...	...	...	...	...	...	...	...

Instruction	Opcode
DRAW	F1

2.5 Branching

2.5.1 Jumping

JMP xxyy

This instruction sets the PC to the value xxyy.

Instruction	Opcode
JMP xxyy	0F yy xx

JMPcc xxyy

This instruction sets the PC to the value xxyy only if the condition cc is met.

For example JMPNZ jumps only if the Z flag is **not** set. JMPZ jumps only if the Z flag is set.

Instruction	Opcode
JMPZ xxyy	1F yy xx
JMPNZ xxyy	2F yy xx
JMPN xxyy	3F yy xx
JMPNN xxyy	4F yy xx
JMPH xxyy	5F yy xx
JMPNH xxyy	6F yy xx
JMPC xxyy	7F yy xx
JMPNC xxyy	8F yy xx

2.5.2 Near-Jumping

JMP xx

This instruction first sets PC to the next instruction, then adds the **signed** xx value to the PC.

For example, if PC is 0x1200 and we have a JMP 0x0A instruction at 0x1200, the processor will first read the opcode 0x9F at 0x1200, get the instruction argument 0x0A at 0x01201, then set the PC as 0x1202 for the next instruction. Then the processor will add to PC the value 0x0A, thus the instruction at 0x120C will be executed.

Note: The instruction JMP 0xFE translates to a subtraction of PC by 2 bytes, which in turn will create an infinite loop.

Instruction	Opcode
JMP xx	9F xx

JMPcc xx

This instruction is a conditional near-jump.

Instruction	Opcode
JMPZ xx	AF xx
JMPNZ xx	BF xx
JMPN xx	CF xx
JMPNN xx	DF xx
JMPH xx	EF xx
JMPNH xx	FF xx
JMPC xx	EE xx
JMPNC xx	FE xx

2.5.3 Functions

CALL xxyy

This instruction pushes the PC for the instruction after the CALL onto the stack, and subsequently sets PC as xxyy.

Instruction	Opcode
CALL xxyy	1E yy xx

RET

This instruction pops the PC off the stack.

Instruction	Opcode
RET	0E

2.6 Miscellaneous

2.6.1 No Operation

NOP

This instruction has no effect.

Instruction	Opcode
NOP	00

2.6.2 Halt and Catch Fire

HCF

The machine halts and catches fire.

Instruction	Opcode
HCF	6C

Section 3

Boot Process

The IPB CHIP9 machine first loads its Boot ROM at memory address 0x0000, then the desired ROM to run at the precise memory address 0x0597. The PC starts with the value of 0x0000, but the other registers are left uninitialized, having values affected by the electronic entropy. The Boot ROM is designed to work with any initial register values.