The following notation applies to this summary:

A	Accumulator
X, Y	Index Registers
M	Memory
P	Processor Status Register
S	Stack Pointer
✓	Change
–	No Change
+	Add
∧	Logical AND
-	Subtract
∀	Logical Exclusive Or
↑	Transfer from Stack
↓	Transfer to Stack
→	Transfer to
←	Transfer to
V	Logical OR
PC	Program Counter
PCH	Program Counter High
PCL	Program Counter Low
OPER	Operand
#	Immediate Addressing Mode

Note: Shown in parentheses at the top of each table a reference number (Ref: XX) which directs the user to the particular Section in the R6500 Microcomputer Family Programming Manual in which the instruction is defined and discussed.

ADC

Add memory to accumulator with carry

ADC

Operation: A + M + C → A, C

N Z C I D V
✓ ✓ ✓ – – ✓

(Ref: 2.2.1)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Immediate	ADC # Oper	69	2	2
Zero Page	ADC Oper	65	2	3
Zero Page, X	ADC Oper, X	75	2	4
Absolute	ADC Oper	6D	3	4
Absolute, X	ADC Oper, X	7D	3	4*
Absolute, Y	ADC Oper, Y	79	3	4*
(Indirect, X)	ADC (Oper, X)	61	2	6
(Indirect), Y	ADC (Oper), Y	71	2	5*

* Add 1 if page boundary is crossed.

AND

"AND" memory with accumulator

AND

Logical AND to the accumulator

Operation: A ∧ M → A

N Z C I D V
✓ ✓ – – – –

(Ref: 2.2.4.1)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Immediate	AND # Oper	29	2	2
Zero Page	AND Oper	25	2	3
Zero Page, X	AND Oper, X	35	2	4
Absolute	AND Oper	2D	3	4
Absolute, X	AND Oper, X	3D	3	4*
Absolute, Y	AND Oper, Y	39	3	4*
(Indirect, X)	AND (Oper, X)	21	2	6
(Indirect), Y	AND (Oper), Y	31	2	5*

* Add 1 if page boundary is crossed.

ASL

ASL *Shift Left One Bit (Memory or Accumulator)*

ASL

Operation: C ← [7|6|5|4|3|2|1|Ø] ←Ø

N Z C I D V
✓ ✓ ✓ – – –

(Ref: 10.2)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Accumulator	ASL A	ØA	1	2
Zero Page	ASL Oper	Ø6	2	5
Zero Page, X	ASL Oper, X	16	2	6
Absolute	ASL Oper	ØE	3	6
Absolute, X	ASL Oper, X	1E	3	7

BCC

BCC *Branch on Carry Clear*

BCC

Operation: Branch on C = Ø

N Z C I D V
– – – – – –

(Ref: 4.1.2.3)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Relative	BCC Oper	9Ø	2	2*

* Add 1 if branch occurs to same page.
* Add 2 if branch occurs to different page.

BCS

BCS *Branch on carry set*

BCS

Operation: Branch on C = 1

N Z C I D V
– – – – – –

(Ref: 4.1.2.4)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Relative	BCS Oper	BØ	2	2*

* Add 1 if branch occurs to same page.
* Add 2 if branch occurs to next page.

BEQ

BEQ *Branch on result zero*

BEQ

Operation: Branch on Z = 1

N Z C I D V
– – – – – –

(Ref: 4.1.2.5)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Relative	BEQ Oper	FØ	2	2*

* Add 1 if branch occurs to same page.
* Add 2 if branch occurs to next page.

BIT

BIT *Test bits in memory with accumulator* **BIT**

Operation: A ∧ M, M_7 → N, M_6 → V

Bit 6 and 7 are transferred to the status register. N Z C I D V

If the result of A∧M is zero then Z = 1, otherwise M_7 ✓ – – – M_6

Z = Ø

(Ref: 4.2.2.1)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Zero Page	BIT Oper	24	2	3
Absolute	BIT Oper	2C	3	4

BMI

BMI *Branch on result minus* **BMI**

Operation: Branch on N = 1 N Z C I D V

– – – – – –

(Ref: 4.1.2.1)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Relative	BMI Oper	3Ø	2	2*

* Add 1 if branch occurs to same page.

* Add 2 if branch occurs to different page.

BNE

BNE *Branch on result not zero* **BNE**

Operation: Branch on Z = 0 N Z C I D V

– – – – – –

(Ref: 4.1.2.6)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Relative	BNE Oper	DØ	2	2*

* Add 1 if branch occurs to same page.

* Add 2 if branch occurs to different page.

BPL

BPL *Branch on result plus* **BPL**

Operation: Branch on N = Ø N Z C I D V

– – – – – –

(Ref: 4.1.2.2)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Relative	BPL Oper	1Ø	2	2*

* Add 1 if branch occurs to same page.

* Add 2 if branch occurs to different page.

BRK

BRK *Force Break*

BRK

Operation: Forced Interrupt PC + 2 ↓ P ↓

N Z C I D V
- - - 1 - -

(Ref: 9.11)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	BRK	ØØ	1	7

1. A BRK command cannot be masked by setting I.

BVC

BVC *Branch on overflow clear*

BVC

Operation: Branch on V = 0

N Z C I D V
- - - - - -

(Ref: 4.1.2.8)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Relative	BVC Oper	5Ø	2	2*

* Add 1 if branch occurs to same page.
* Add 2 if branch occurs to different page.

BVS

BVS *Branch on overflow set*

BVS

Operation: Branch on V = 1

N Z C I D V
- - - - - -

(Ref: 4.1.2.7)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Relative	BVS Oper	7Ø	2	2*

* Add 1 if branch occurs to same page.
* Add 2 if branch occurs to different page.

CLC

CLC *Clear carry flag*

CLC

Operation: Ø → C

N Z C I D V
- - Ø - - -

(Ref: 3.0.2)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	CLC	18	1	2

CLD

CLD *Clear decimal mode*

Operation: Ø → D

N Z C I D V
– – – – Ø –

(Ref: 3.3.2)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	CLD	D8	1	2

CLI

CLI *Clear interrupt disable bit*

Operation: Ø → I

N Z C I D V
– – – Ø – –

(Ref: 3.2.2)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	CLI	58	1	2

CLV

CLV *Clear overflow flag*

Operation: Ø → V

N Z C I D V
– – – – – Ø

(Ref: 3.6.1)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	CLV	B8	1	2

CMP

CMP *Compare memory and accumulator*

Operation: A - M

N Z C I D V
✓ ✓ ✓ – – –

(Ref: 4.2.1)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Immediate	CMP #Oper	C9	2	2
Zero Page	CMP Oper	C5	2	3
Zero Page, X	CMP Oper, X	D5	2	4
Absolute	CMP Oper	CD	3	4
Absolute, X	CMP Oper, X	DD	3	4*
Absolute, Y	CMP Oper, Y	D9	3	4*
(Indirect, X)	CMP (Oper, X)	C1	2	6
(Indirect), Y	CMP (Oper), Y	D1	2	5*

* Add 1 if page boundary is crossed.

CPX

CPX *Compare Memory and Index X*

CPX

Operation: X - M

N Z C I D V
✓ ✓ ✓ – – –

(Ref: 7.8)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Immediate	CPX #Oper	EØ	2	2
Zero Page	CPX Oper	E4	2	3
Absolute	CPX Oper	EC	3	4

CPY

CPY *Compare memory and index Y*

CPY

Operation: Y - M

N Z C I D V
✓ ✓ ✓ – – –

(Ref: 7.9)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Immediate	CPY #Oper	CØ	2	2
Zero Page	CPY Oper	C4	2	3
Absolute	CPY Oper	CC	3	4

DEC

DEC *Decrement memory by one*

DEC

Operation: M - 1 → M

N Z C I D V
✓ ✓ – – – –

(Ref: 10.8)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Zero Page	DEC Oper	C6	2	5
Zero Page, X	DEC Oper, X	D6	2	6
Absolute	DEC Oper	CE	3	6
Absolute, X	DEC Oper, X	DE	3	7

DEX

DEX *Decrement index X by one*

DEX

Operation: X - 1 → X

N Z C I D V
✓ ✓ – – – –

(Ref: 7.6)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	DEX	CA	1	2

DEY

DEY *Decrement index Y by one*

DEY

Operation: Y - 1 → Y

N Z C I D V
✓ ✓ – – – –

(Ref: 7.7)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	DEY	88	1	2

EOR

EOR *"Exclusive-Or" memory with accumulator*

EOR

Operation: A ∀ M → A

N Z C I D V
✓ ✓ – – – –

(Ref: 2.2.4.3)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Immediate	EOR #Oper	49	2	2
Zero Page	EOR Oper	45	2	3
Zero Page, X	EOR Oper, X	55	2	4
Absolute	EOR Oper	4D	3	4
Absolute, X	EOR Oper, X	5D	3	4*
Absolute, Y	EOR Oper, Y	59	3	4*
(Indirect, X)	EOR (Oper, X)	41	2	6
(Indirect),Y	EOR (Oper), Y	51	2	5*

* Add 1 if page boundary is crossed.

INC

INC *Increment memory by one*

INC

Operation: M + 1 → M

N Z C I D V
✓ ✓ – – – –

(Ref: 10.7)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Zero Page	INC Oper	E6	2	5
Zero Page, X	INC Oper, X	F6	2	6
Absolute	INC Oper	EE	3	6
Absolute, X	INC Oper, X	FE	3	7

INX

INX *Increment Index X by one*

INX

Operation: X + 1 → X

N Z C I D V
✓ ✓ – – – –

(Ref: 7.4)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	INX	E8	1	2

INY

INY *Increment Index Y by one*

Operation: Y + 1 → Y

N Z C I D V
✓ ✓ – – – –

(Ref: 7.5)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	INY	C8	1	2

JMP

JMP *Jump to new location*

Operation: (PC + 1) → PCL
(PC + 2) → PCH

(Ref: 4.0.2)
(Ref: 9.8.1)

N Z C I D V
– – – – – –

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Absolute	JMP Oper	4C	3	3
Indirect	JMP (Oper)	6C	3	5

JSR

JSR *Jump to new location saving return address*

Operation: PC + 2 ↓, (PC + 1) → PCL
(PC + 2) → PCH

(Ref: 8.1)

N Z C I D V
– – – – – –

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Absolute	JSR Oper	20	3	6

LDA

LDA *Load accumulator with memory*

Operation: M → A

(Ref: 2.1.1)

N Z C I D V
✓ ✓ – – – –

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Immediate	LDA # Oper	A9	2	2
Zero Page	LDA Oper	A5	2	3
Zero Page, X	LDA Oper, X	B5	2	4
Absolute	LDA Oper	AD	3	4
Absolute, X	LDA Oper, X	BD	3	4*
Absolute, Y	LDA Oper, Y	B9	3	4*
(Indirect, X)	LDA (Oper, X)	A1	2	6
(Indirect), Y	LDA (Oper), Y	B1	2	5*

* Add 1 if page boundary is crossed.

LDX

LDX *Load index X with memory*

LDX

Operation: M → X

N Z C I D V
✓ ✓ – – – –

(Ref: 7.0)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Immediate	LDX # Oper	A2	2	2
Zero Page	LDX Oper	A6	2	3
Zero Page, Y	LDX Oper, Y	B6	2	4
Absolute	LDX Oper	AE	3	4
Absolute, Y	LDX Oper, Y	BE	3	4*

* Add 1 when page boundary is crossed.

LDY

LDY *Load index Y with memory*

LDY

Operation: M → Y

N Z C I D V
✓ ✓ – – – –

(Ref: 7.1)

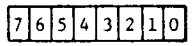
Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Immediate	LDY #Oper	AØ	2	2
Zero Page	LDY Oper	A4	2	3
Zero Page, X	LDY Oper, X	B4	2	4
Absolute	LDY Oper	AC	3	4
Absolute, X	LDY Oper, X	BC	3	4*

* Add 1 when page boundary is crossed.

LSR

LSR *Shift right one bit (memory or accumulator)*

LSR

Operation: Ø →  → C

N Z C I D V
Ø ✓ ✓ – – –

(Ref: 10.1)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Accumulator	LSR A	4A	1	2
Zero Page	LSR Oper	46	2	5
Zero Page, X	LSR Oper, X	56	2	6
Absolute	LSR Oper	4E	3	6
Absolute, X	LSR Oper, X	5E	3	7

NOP

NOP *No operation*

NOP

Operation: No Operation (2 cycles)

N Z C I D V
– – – – – –

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	NOP	EA	1	2

ORA

ORA *"OR" memory with accumulator* ORA

Operation: A ∨ M → A

N Z C I D V
✓ ✓ – – – –

(Ref: 2.2.4.2)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Immediate	ORA #Oper	Ø9	2	2
Zero Page	ORA Oper	Ø5	2	3
Zero Page, X	ORA Oper, X	15	2	4
Absolute	ORA Oper	ØD	3	4
Absolute, X	ORA Oper, X	1D	3	4*
Absolute, Y	ORA Oper, Y	19	3	4*
(Indirect, X)	ORA (Oper, X)	Ø1	2	6
(Indirect), Y	ORA (Oper), Y	11	2	5*

* Add 1 on page crossing

PHA

PHA *Push accumulator on stack* PHA

Operation: A ↓

N Z C I D V
– – – – – –

(Ref: 8.5)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	PHA	48	1	3

PHP

PHP *Push processor status on stack* PHP

Operation: P↓

N Z C I D V
– – – – – –

(Ref: 8.11)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	PHP	Ø8	1	3

PLA

PLA *Pull accumulator from stack* PLA

Operation: A ↑

N Z C I D V
✓ ✓ – – – –

(Ref: 8.6)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	PLA	68	1	4

PLP

PLP *Pull processor status from stack*

PLP

Operation: P ↑

N Z C I D V

From Stack

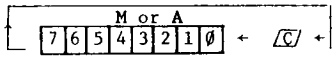
(Ref: 8.12)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	PLP	28	1	4

ROL

ROL *Rotate one bit left (memory or accumulator)*

ROL

Operation: 

N Z C I D V

✓ ✓ ✓ – – –

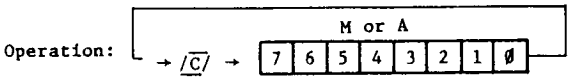
(Ref: 10.3)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Accumulator	ROL A	2A	1	2
Zero Page	ROL Oper	26	2	5
Zero Page, X	ROL Oper, X	36	2	6
Absolute	ROL Oper	2E	3	6
Absolute, X	ROL Oper, X	3E	3	7

ROR

ROR *Rotate one bit right (memory or accumulator)*

ROR

Operation: 

N Z C I D V

✓ ✓ ✓ – – –

(Ref: 10.4)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Accumulator	ROR A	6A	1	2
Zero Page	ROR Oper	66	2	5
Zero Page,X	ROR Oper,X	76	2	6
Absolute	ROR Oper	6E	3	6
Absolute,X	ROR Oper,X	7E	3	7

RTI

RTI *Return from interrupt*

RTI

Operation: P↑ PC↑

N Z C I D V

From Stack

(Ref: 9.6)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	RTI	4Ø	1	6

RTS

RTS *Return from subroutine*

RTS

Operation: PC↑, PC + 1→ PC

N Z C I D V

– – – – – –

(Ref: 8.2)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	RTS	6Ø	1	6

SBC

SBC *Subtract memory from accumulator with borrow*

Operation: A − M − $\overline{C}$ → A

Note: $\overline{C}$ = Borrow (Ref: 2.2.2)

N Z C I D V
✓ ✓ ✓ — — ✓

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Immediate	SBC #Oper	E9	2	2
Zero Page	SBC Oper	E5	2	3
Zero Page, X	SBC Oper, X	F5	2	4
Absolute	SBC Oper	ED	3	4
Absolute, X	SBC Oper, X	FD	3	4*
Absolute, Y	SBC Oper, Y	F9	3	4*
(Indirect, X)	SBC (Oper, X)	E1	2	6
(Indirect), Y	SBC (Oper), Y	F1	2	5*

* Add 1 when page boundary is crossed.

SEC

SEC *Set carry flag*

Operation: 1 → C

(Ref: 3.0.1)

N Z C I D V
— — 1 — — —

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	SEC	38	1	2

SED

SED *Set decimal mode*

Operation: 1 → D

N Z C I D V
— — — — 1 —

(Ref: 3.3.1)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	SED	F8	1	2

SEI

SEI *Set interrupt disable status*

Operation: 1 → I

N Z C I D V
— — — 1 — —

(Ref: 3.2.1)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	SEI	78	1	2

STA

STA *Store accumulator in memory* **STA**

Operation: A → M

N Z C I D V
- - - - - -

(Ref: 2.1.2)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Zero Page	STA Oper	85	2	3
Zero Page, X	STA Oper, X	95	2	4
Absolute	STA Oper	8D	3	4
Absolute, X	STA Oper, X	9D	3	5
Absolute, Y	STA Oper, Y	99	3	5
(Indirect, X)	STA (Oper, X)	81	2	6
(Indirect), Y	STA (Oper), Y	91	2	6

STX

STX *Store index X in memory* **STX**

Operation: X → M

N Z C I D V
- - - - - -

(Ref: 7.2)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Zero Page	STX Oper	86	2	3
Zero Page, Y	STX Oper, Y	96	2	4
Absolute	STX Oper	8E	3	4

STY

STY Store index Y in memory **STY**

Operation: Y → M

N Z C I D V
- - - - - -

(Ref: 7.3)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Zero Page	STY Oper	84	2	3
Zero Page, X	STY Oper, X	94	2	4
Absolute	STY Oper	8C	3	4

TAX

TAX *Transfer accumulator to index X* **TAX**

Operation: A → X

N Z C I D V
✓ ✓ - - - -

(Ref: 7.11)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	TAX	AA	1	2

TAY

TAY *Transfer accumulator to index Y* **TAY**

Operation: A → Y

N Z C I D V
✓ ✓ – – – –

(Ref: 7.13)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	TAY	A8	1	2

TYA

TYA *Transfer index Y to accumulator* **TYA**

Operation: Y → A

N Z C I D V
✓ ✓ – – – –

(Ref: 7.14)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	TYA	98	1	2

TSX

TSX *Transfer stack pointer to index X* **TSX**

Operation: S → X

N Z C I D V
✓ ✓ – – – –

(Ref: 8.9)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	TSX	BA	1	2

TXA

TXA *Transfer index X to accumulator* **TXA**

Operation: X → A

N Z C I D V
✓ ✓ – – – –

(Ref: 7.12)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	TXA	8A	1	2

TXS

TXS *Transfer index X to stack pointer* **TXS**

Operation: X → S

N Z C I D V
– – – – – –

(Ref: 8.8)

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	TXS	9A	1	2

APPENDIX C

INSTRUCTION ADDRESSING MODES AND RELATED EXECUTION TIMES

INSTRUCTION ADDRESSING MODES AND RELATED EXECUTION TIMES (in clock cycles)

	Accumulator	Immediate	Zero Page	Zero Page, X	Zero Page, Y	Absolute	Absolute, X	Absolute, Y	Implied	Relative	(Indirect, X)	(Indirect), Y	Absolute Indirect
ADC		2	3	4		4	4*	4*			6	5*	
AND		2	3	4		4	4*	4*			6	5*	
ASL	2		5	6		6	7						
BCC										2**			
BCS										2**			
BEQ										2**			
BIT			3			4							
BMI										2**			
BNE										2**			
BPL										2**			
BRK													
BVC										2**			
BVS										2**			
CLC									2				
CLD									2				
CLI									2				
CLV									2				
CMP		2	3	4		4	4*	4*			6	5*	
CPX		2	3			4							
CPY		2	3			4							
DEC			5	6		6	7						
DEX									2				
DEY									2				
EOR		2	3	4		4	4*	4*			6	5*	
INC			5	6		6	7						
INX									2				
INY									2				
JMP						3							5
JSR										6			
LDA		2	3	4		4	4*	4*			6	5*	
LDX		2	3		4	4		4*					
LDY		2	3	4		4	4*						
LSR	2		5	6		6	7						
NOP									2				
ORA		2	3	4		4	4*	4*			6	5*	
PHA									3				
PHP									3				
PLA									4				
PLP									4				
ROL	2		5	6		6	7						
ROR	2		5	6		6	7						
RTI									6				
RTS									6				
SBC		2	3	4		4	4*	4*			6	5*	
SEC									2				
SED									2				
SEI									2				
STA			3	4		4	5	5			6	6	
STX			3		4	4							
STY			3	4		4							
TAX									2				
TAY									2				
TSX									2				
TXA									2				
TXS									2				
TYA									2				

* Add one cycle if indexing across page boundary
** Add one cycle if branch is taken, Add one additional if branching operation crosses page boundary

APPENDIX D

OPERATION CODE INSTRUCTION LISTING, HEXIDECIMAL SEQUENCE

ØØ - BRK
Ø1 - ORA - (Indirect,X)
Ø2 - Future Expansion
Ø3 - Future Expansion
Ø4 - Future Expansion
Ø5 - ORA - Zero Page
Ø6 - ASL - Zero Page
Ø7 - Future Expansion
Ø8 - PHP
Ø9 - ORA - Immediate
ØA - ASL - Accumulator
ØB - Future Expansion
ØC - Future Expansion
ØD - ORA - Absolute
ØE - ASL - Absolute
ØF - Future Expansion
1Ø - BPL
11 - ORA - (Indirect),Y
12 - Future Expansion
13 - Future Expansion
14 - Future Expansion
15 - ORA - Zero Page,X
16 - ASL - Zero Page,X
17 - Future Expansion
18 - CLC
19 - ORA - Absolute,Y
1A - Future Expansion
1B - Future Expansion
1C - Future Expansion
1D - ORA - Absolute,X
1E - ASL - Absolute,X
1F - Future Expansion

2Ø - JSR
21 - AND - (Indirect,X)
22 - Future Expansion
23 - Future Expansion
24 - BIT - Zero Page
25 - AND - Zero Page
26 - ROL - Zero Page
27 - Future Expansion
28 - PLP
29 - AND - Immediate
2A - ROL - Accumulator
2B - Future Expansion
2C - BIT - Absolute
2D - AND - Absolute
2E - ROL - Absolute
2F - Future Expansion
3Ø - BMI
31 - AND - (Indirect),Y
32 - Future Expansion
33 - Future Expansion
34 - Future Expansion
35 - AND - Zero Page,X
36 - ROL - Zero Page,X
37 - Future Expansion
38 - SEC
39 - AND - Absolute,Y
3A - Future Expansion
3B - Future Expansion
3C - Future Expansion
3D - AND - Absolute,X
3E - ROL - Absolute,X
3F - Future Expansion

4Ø - RTI
41 - EOR - (Indirect,X)
42 - Future Expansion
43 - Future Expansion
44 - Future Expansion
45 - EOR - Zero Page
46 - LSR - Zero Page
47 - Future Expansion
48 - PHA
49 - EOR - Immediate
4A - LSR - Accumulator
4B - Future Expansion
4C - JMP - Absolute
4D - EOR - Absolute
4E - LSR - Absolute
4F - Future Expansion
5Ø - BVC
51 - EOR - (Indirect),Y
52 - Future Expansion
53 - Future Expansion
54 - Future Expansion
55 - EOR - Zero Page,X
56 - LSR - Zero Page,X
57 - Future Expansion
58 - CLI
59 - EOR - Absolute,Y
5A - Future Expansion
5B - Future Expansion
5C - Future Expansion
5D - EOR - Absolute,X
5E - LSR - Absolute,X
5F - Future Expansion

6Ø - RTS
61 - ADC - (Indirect,X)
62 - Future Expansion
63 - Future Expansion
64 - Future Expansion
65 - ADC - Zero Page
66 - ROR - Zero Page
67 - Future Expansion
68 - PLA
69 - ADC - Immediate
6A - ROR - Accumulator
6B - Future Expansion
6C - JMP - Indirect
6D - ADC - Absolute
6E - ROR - Absolute
6F - Future Expansion
7Ø - BVS
71 - ADC - (Indirect),Y
72 - Future Expansion
73 - Future Expansion
74 - Future Expansion
75 - ADC - Zero Page,X
76 - ROR - Zero Page,X
77 - Future Expansion
78 - SEI
79 - ADC - Absolute,Y
7A - Future Expansion
7B - Future Expansion
7C - Future Expansion
7D - ADC - Absolute,X
7E - ROR - Absolute,X
7F - Future Expansion

8Ø - Future Expansion
81 - STA - (Indirect,X)
82 - Future Expansion
83 - Future Expansion
84 - STY - Zero Page
85 - STA - Zero Page
86 - STX - Zero Page
87 - Future Expansion
88 - DEY
89 - Future Expansion
8A - TXA
8B - Future Expansion
8C - STY - Absolute
8D - STA - Absolute
8E - STX - Absolute
8F - Future Expansion
9Ø - BCC
91 - STA - (Indirect),Y
92 - Future Expansion
93 - Future Expansion
94 - STY - Zero Page,X
95 - STA - Zero Page,X
96 - STX - Zero Page,Y
97 - Future Expansion
98 - TYA
99 - STA - Absolute,Y
9A - TXS
9B - Future Expansion
9C - Future Expansion
9D - STA - Absolute,X
9E - Future Expansion
9F - Future Expansion

AØ - LDY - Immediate
A1 - LDA - (Indirect,X)
A2 - LDX - Immediate
A3 - Future Expansion
A4 - LDY - Zero Page
A5 - LDA - Zero Page
A6 - LDX - Zero Page
A7 - Future Expansion
A8 - TAY
A9 - LDA - Immediate
AA - TAX
AB - Future Expansion
AC - LDY - Absolute
AD - LDA - Absolute
AE - LDX - Absolute
AF - Future Expansion
BØ - BCS
B1 - LDA - (Indirect),Y
B2 - Future Expansion
B3 - Future Expansion
B4 - LDY - Zero Page,X
B5 - LDA - Zero Page,X
B6 - LDX - Zero Page,Y
B7 - Future Expansion
B8 - CLV
B9 - LDA - Absolute,Y
BA - TSX
BB - Future Expansion
BC - LDY - Absolute,X
BD - LDA - Absolute,X
BE - LDX - Absolute,Y
BF - Future Expansion

CØ - CPY - Immediate
C1 - CMP - (Indirect,X)
C2 - Future Expansion
C3 - Future Expansion
C4 - CPY - Zero Page
C5 - CMP - Zero Page
C6 - DEC - Zero Page
C7 - Future Expansion
C8 - INY
C9 - CMP - Immediate
CA - DEX
CB - Future Expansion
CC - CPY - Absolute
CD - CMP - Absolute
CE - DEC - Absolute
CF - Future Expansion
DØ - BNE
D1 - CMP - (Indirect),Y
D2 - Future Expansion
D3 - Future Expansion
D4 - Future Expansion
D5 - CMP - Zero Page,X
D6 - DEC - Zero Page,X
D7 - Future Expansion
D8 - CLD
D9 - CMP - Absolute,Y
DA - Future Expansion
DB - Future Expansion
DC - Future Expansion
DD - CMP - Absolute,X
DE - DEC - Absolute,X
DF - Future Expansion

EØ - CPX - Immediate
E1 - SBC - (Indirect,X)
E2 - Future Expansion
E3 - Future Expansion
E4 - CPX - Zero Page
E5 - SBC - Zero Page
E6 - INC - Zero Page
E7 - Future Expansion
E8 - INX
E9 - SBC - Immediate
EA - NOP
EB - Future Expansion
EC - CPX - Absolute
ED - SBC - Absolute
EE - INC - Absolute
EF - Future Expansion
FØ - BEQ
F1 - SBC - (Indirect),Y
F2 - Future Expansion
F3 - Future Expansion
F4 - Future Expansion
F5 - SBC - Zero Page,X
F6 - INC - Zero Page,X
F7 - Future Expansion
F8 - SED
F9 - SBC - Absolute,Y
FA - Future Expansion
FB - Future Expansion
FC - Future Expansion
FD - SBC - Absolute,X
FE - INC - Absolute,X
FF - Future Expansion

APPENDIX E

SUMMARY OF ADDRESSING MODES

Appendix E is intended to serve the user by serving as a reference for the R6500 addressing modes. Each mode of address is shown with a symbolic illustration of the bus status at each cycle during the instruction fetch and execution. The example number as found in the text is provided for reference purposes.

E.1 IMPLIED ADDRESSING

Example 5.3: Illustration of Implied Addressing

Clock Cycle	Address Bus	Program Counter	Data Bus	Comments
1	PC	PC + 1	OP CODE	Fetch OP CODE
2	PC + 1	PC + 1	New OP CODE	Ignore New OP CODE; Decode Old OP CODE
3	PC + 1	PC + 2	New OP CODE	Fetch New OP CODE; Execute Old OP CODE

E.2 IMMEDIATE ADDRESSING

Example 5.4: Illustration of Immediate Addressing

Clock Cycle	Address Bus	Program Counter	Data Bus	Comments
1	PC	PC + 1	OP CODE	Fetch OP CODE
2	PC + 1	PC + 2	Data	Fetch Data, Decode OP CODE
3	PC + 2	PC + 3	New OP CODE	Fetch New OP CODE, Execute Old OP CODE

E.3 ABSOLUTE ADDRESSING

Example 5.5: Illustration of Absolute Addressing

Clock Cycle	Address Bus	Program Counter	Data Bus	Comments
1	PC	PC + 1	OP CODE	Fetch OP CODE
2	PC + 1	PC + 2	ADL	Fetch ADL, Decode OP CODE
3	PC + 2	PC + 3	ADH	Fetch ADH, Retail ADL
4	ADH, ADL	PC + 3	Data	Fetch Data
5	PC + 3	PC + 4	New OP CODE	Fetch New OP CODE, Execute Old OP CODE

E.4 ZERO PAGE ADDRESSING

Example 5.6: Illustration of Zero Page Addressing

Clock Cycle	Address Bus	Program Counter	Data Bus	Comments
1	PC	PC + 1	OP CODE	Fetch OP CODE
2	PC + 1	PC + 2	ADL	Fetch ADL, Decode OP CODE
3	00, ADL	PC + 2	Data	Fetch Data
4	PC + 2	PC + 3	New OP CODE	Fetch New OP CODE, Execute Old OP CODE

E.5 RELATIVE ADDRESSING (BRANCH POSITIVE, NO CROSSING OF PAGE BOUNDARIES)

Example 5.8: Illustration of Relative Addressing -- Branch Positive Take; No Crossing of Page Boundaries

Cycle	Address Bus	Data Bus	External Operation	Internal Operation
1	0100	OP CODE	Fetch OP CODE	Finish Previous Operation, Increment Program Counter to 101
2	0101	+50	Fetch Offset	Interpret Instruction, Increment Program Counter to 102
3	0102	Next OP CODE	Fetch Next OP CODE	Check Flags, Add Relative to PCL, Increment Program Counter to 103
4	0152	Next OP CODE	Fetch Next OP CODE	Transfer Results to PCL, Increment Program Counter to 153

E.6 ABSOLUTE INDEXED ADDRESSING (WITH PAGE CROSSING)

Step 5 is deleted and the data in step 4 are valid when no page crossing occurs.

Example 6.7: Absolute Indexed; With Page Crossing

Cycle	Address Bus	Data Bus	External Operation	Internal Operation
1	0100	OP CODE	Fetch OP CODE	Finish Previous Operation Increment PC to 101
2	0101	BAL	Fetch BAL	Interpret Instruction Increment PC to 102
3	0102	BAH	Fetch BAH	Add BAL + Index Increment PC to 103
4	BAH,BAL +X	Data (Ignore)	Fetch Data (Data is ignored)	Add BAH + Carry
5	BAH+1, BAL+X	Data	Fetch Data	
6	0103	Next OP CODE	Fetch Next OP CODE	Finish Operation

E.7 ZERO PAGE INDEXED ADDRESSING

Example 6.8: Illustration of Zero Page Indexing

Cycle	Address Bus	Data Bus	External Operation	Internal Operation
1	0100	OP CODE	Fetch OP CODE	Finish Previous Operation
2	0101	BAL	Fetch Base Address Low (BAL)	Interpret Instruction
3	00,BAL	Data (Discarded	Fetch Discarded Data	Add: BAL + X
4	00,BAL +X	Data	Fetch Data	
5	0102	Next OP CODE	Fetch Next OP CODE	Finish Operation

E.8 INDEXED INDIRECT ADDRESSING

Example 6.10: Illustration of Indexed Indirect Addressing

Cycle	Address Bus	Data Bus	External Operation	Internal Operation
1	0100	OP CODE	Fetch OP CODE	Finish Previous Operation
2	0101	BAL	Fetch BAL	Interpret Instruction
3	00,BAL	DATA (Discarded)	Fetch Discarded DATA	Add BAL + X
4	00,BAL + X	ADL	Fetch ADL	Add 1 to BAL + X
5	00,BAL + X + 1		Fetch ADH	Hold ADL
6	ADH,ADL	DATA	Fetch DATA	
7	0102	Next OP	Fetch Next OP CODE	Finish Operation

E.9 INDIRECT INDEXED ADDRESSING (WITH PAGE CROSSING)

Step 6 is deleted and the data in step 5 are valid when no page crossing occurs.

Example 6.12: Indirect Indexed Addressing (With Page Crossing)

Cycle	Address Bus	Data Bus	External Operation	Internal Operation
1	0100	OP CODE	Load OP CODE	Finish Previous Operation
2	0101	IAL	Fetch IAL	Interpret Instruction
3	00,IAL	BAL	Fetch BAL	Add 1 to IAL
4	00,IAL + 1	BAH	Fetch BAH	Add BAL to Y
5	BAH,BAL + Y	DATA (Discarded)	Fetch DATA (Discarded)	Add 1 to BAH
6	BAH + 1 BAL + Y	DATA	Fetch Data	
7	0102	Next OP CODE	Fetch Next OP CODE	Finish This Operation

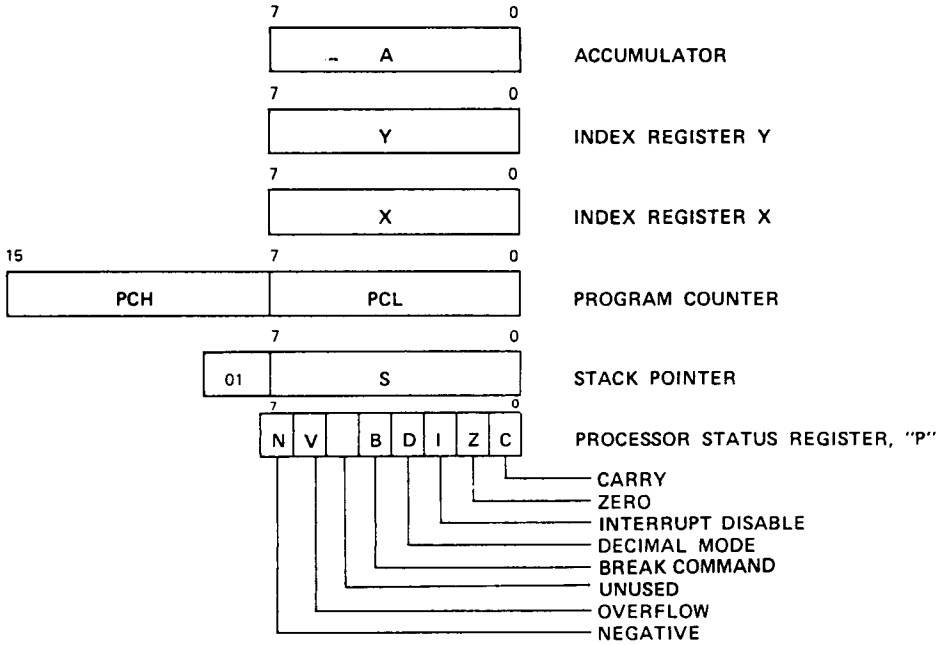
APPENDIX F

R6500

PROGRAMMING

MODEL

PROGRAMMING MODEL R6500



APPENDIX G

DISCUSSION -- INDIRECT ADDRESSING

The R6500 System's family of microprocessors have a special form of addressing known as "Indirect". The basic operation of Indirect addressing is described in the main body of this manual. It is the intent of Appendix H to acquaint the user with some of the uses and applications of Indirect addressing.

The Indirect address is really an address which would have been coded in-line as in the case of Absolute, except for the fact that the address is not known at the time the user writes the program. As has been indicated several times in the body of this manual, it is significantly more efficient with the organization of the R6500 to assign addresses and implement them if the addressing structure is known; however, this is not always possible to accomplish. For instance, in order to minimize the coding of a subroutine or general-purpose set of coding, it is often desirable to work with a range of addressing that is impossible to cover in a normal index, or, in the case of subroutine, where it is necessary for the addresses to be variable depending on which part of the whole program called the address.

It is probably this discussion which best amplifies the need for calculated addresses. It should be fairly obvious to the user that a general-purpose subroutine cannot contain the address of the operations. Therefore, instead of having the instruction LDA followed by the value that the programmer wants to load, in a subroutine it may be desirable to perform a Load A from a calculated or specified address.

The use of the Indirect Addressing Mode is to give the user a location in Page Zero in which can be put the calculated address. Then the subroutine instruction can call this calculated address, using the form Load A from an address pointed to by the next byte in program sequence. The word "indirect" technically comes from the fact that instead of taking the address which is immediately following the instruction, the next value in program sequence is a pointer to the address.

The Indirect pointer will be referred to from now on as IAL, because it is a Zero Page address and, therefore, is a low-order byte. The indirect instructions are written in the form "Load A" followed by IAL.

IAL points to an address which had been previously stored into Page Zero. This gives the user the flexibility of addressing anywhere in memory with a calculated address. However, the real value of Indirect is not simply in having Indirect, but in having the ability to have Indirect modified. This is the reason for which indirect indexed instruction is implemented rather than straight indirect. An example of the indirect indexed in subroutining is covered in Section 6.5, but it should be noted that the indirect indexed instruction should be employed whenever the user does not know the exact address at time of compilation. Although there may be other interesting and esoteric uses of the indirect index instruction, this is the most common one.

The second form of indirect is very powerful for certain types of applications. Chapter 11 shows the use of tables which have pointers, and the advantage of running down one table of pointers until a match is found and then using the same index to address a second table to perform an operation. This is the classical stack processor type of architecture but it requires a special discipline at the time a program is originally defined. Both the indirects require a concept of memory management that is not obvious to the novice programmer.

The concept of indexed indirect is that memory has to be viewed as a series of tables, in which access to one set of tables is accomplished by indexing through a list of pointers. One set of tables might be searched to perform some type of testing or operation; then the same index is employed to process another set of pointers. This concept is only applicable to operations in which a variety of inputs are being serviced. A classical application is when several remote devices are being managed by the same control program. An example might be having three teletypes tied on to a device; each teletype is being manually controlled and can be under control of the user program. In this type of message-handling environment, the control program for the teletypes does nothing more than collect strings of data from the input device and then perform operations on the string upon seeing a control signal, usually a carriage

return in this case of the teletype. Because any one of the teletypes can be causing any one of the series of operations, this program does not lend itself well to the concept of absolute addressing. In fact, most of the subroutines which deal with the individual processing should be written address-independent. This normally allows the addition of more devices without paying any penalty in terms of programming. Therefore, this is a subroutine or nonabsolute type of operation in which the indirect indexed would not apply, because each of the various operations use a function of position. In other words, one can assign a series of tables that point at the teletype itself, another set that points at an outgoing message stream, and another set that points to a series of tables which keep the status of the device. Each of these pointers is considered to be an individual address at the beginning of a string. Each string has a variable length. The teletype strings may consist of a three-character message followed by a carriage return, or a 40-character message followed by a carriage return. In the R6500, this system is implemented by developing a series of indirect pointers. Each teletype has an indirect pointer. Its I/O port has another indirect pointer that points at the put-away string, another one that points at the teletype message output string, another one that points at its status table. If all of the teletypes work in this manner, it can be seen that the coding to put data into the input message table is the same for all the teletypes and is totally independent of the teletype in which data is being stored.

The index register X serves as a control for the tables so that if all tables were sequentially organized, X would point at the proper value for each operation. A sample operation might be: read teletype three, transfer the data to teletype three input register, update teletype three counter, check to see that teletype three is still active, and decide whether or not to return to signal teletype three back. The coding to perform each of these operations would be exactly the same as coding for teletype two, if the tables were organized in such a way that X was an index register for the pointers.

This is the type of string manipulation application for which indexed indirect was designed, and only when a program can be organized for this technique is the indirect used to its maximum potential. The advantages for organizing for this type of approach when the problem requires string manipulation is significant: the comprehensive I/O program is roughly one-half the memory and one-fourth the execution time of several other microprocessors which do not have this indexed indirect feature.

APPENDIX H

REVIEW OF BINARY

AND

BINARY-CODED DECIMAL

ARITHMETIC

The number 1789 is assumed by most people to mean one thousand, seven hundred and eighty-nine,or $1 \times 10^3 + 7 \times 10^2 + 8 \times 10^1 + 9 \times 10^0$. However, until the number base is defined, it might mean

$$1 \times 16^3 + 7 \times 16^2 + 8 \times 16^1 + 9 \times 16^0$$

which is hexadecimal and the form that is used in the microprocessor.

In order to distinguish between numbers on different bases, mathematicians usually write 1789_{10} or just 1789 for base 10, or decimal, and 1789_{16} for base 16 for hexadecimal. Because very few computers or I/O devices allow subscripting, all hexadecimal numbers are preceded by a "$" notation; thus, "1789" indictes base 10 and "$1789" indicates base 16.

Why hexadecimal? This is a convenient way of representing 8 bits in 2 digits.

The R6500 is a byte-oriented microprocessor which means most operations have 8-bit operations. There are 2 ways to look at 8 bits. The first is as 8 individual bits in which 00001000 means that bit 3 (bit 7 to 0 representation) is on and all other bits are off, or as an 8-bit binary number in which case the value is

$0 \times 2^7 + 0 \times 2^6 + 0 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 0 \times 2^0 = 8$

or $08.

For logic analysis purposes each bit is unique, but for arithmetic purposes the 8 bits are treated as a binary number.

Binary Arithmetic Rules:

0 + 0 = 0

0 + 1 = 1

1 + 0 = 1

1 + 1 = 0 with a carry

Carry occurs when the resulting number is too long for the base. In decimal, 8 + 4 = 2 + 10. In hexadecimal, $8 + $4 = $C (see hexadecimal details), so that 8 + 4 has a carry in base 10 but not in base 16.

Using these rules to add 8 + 2 in binary gives the following:

00001000	8	1×2^3
00000010	+2	1×2^1
00001010	10	$1 \times 2^3 + 1 \times 2^1$

Therefore, any number from 0 - 255 may be represented in 8 bits, and binary addition performed using the basic binary add equation, $R_j = (A_j \forall B_j \forall C_{j-1})$, where, as defined previously, $\forall$ is notation for Exclusive-Or.

In most applications, it is also necessary to subtract. Subtract operations require either a different hardware implementation or a new way of representing numbers.

A combination of this is to implement a simple inverter in each bit. This would make

00001100	12
11110011	-12

However, when subtracting 12 from 12, the result should also be 0.

00001100	+12
11110011	-12
11111111	0

However, if a carry is added to the complemented number:

1	Carry
00001100	12
11110011	-12
00000000 =	0

If, instead of representing -12 as the complement of 12, it is represented as the complement plus carry, the following is obtained:

```
11110011 = 12
       1 = Carry
11110100 = -12
00001100   +12
00000000 =  0
```

This representation is called two's complement and represents the way that negative numbers are kept in the microcomputer. Below are examples of negative numbers represented in two's complement form.

-Ø = 00000000
-1 = 11111111
-2 = 11111110
-3 = 11111101
-4 = 11111100
-5 = 11111011
-6 = 11111010
-7 = 11111001
-8 = 11111000
-9 = 11110111

Hexadecimal is the representation of numbers to the base 16. The following table shows the advantages of Hex:

Hexadecimal	Binary	Decimal
0	0000	00
1	0001	01
2	0010	02
3	0C11	03
4	0100	04
5	0101	05
6	0110	06
7	0111	07
8	1000	08
9	1001	09
A	1010	10
B	1011	11
C	1100	12
D	1101	13
E	1110	14
F	1111	15

Because 16 is a multiple of 2, hexadecimal is a convenient shorthand for representation of 4 binary digits or bits. The rules on arithmetic also hold.

Binary	Hex
0100 1111	4F
+ 0110 0010	+ 62
1011 0001	B1

To take advantage of this shorthand, all addresses in this manual are shown in hexadecimal notation. It should be noted that the reader should learn to operate in Hex as soon as possible. Continual translation back to decimal is both time-consuming and error-prone. Working in Hex and binary will quickly force learning of hexadecimal manipulation and the familiarity with working with this convenient representation.

Although many microcomputer applications can successfully be accomplished with binary operations, some applications are best performed in decimal. Although the use of one decimal character per byte would be a legitimate way to solve this problem, this is an inefficient use of the capability of the 8-bit byte.

The microprocessor allows the use of BCD representation. This representation is, in 4-bit form:

0 = 0000
1 = 0001
2 = 0010
3 = 0011
4 = 0100
5 = 0101
6 = 0110
7 = 0111
8 = 1000
9 = 1001

In BCD, the number 79 is represented:

Binary	BCD	Hex
01111001 =	79 =	79

The microprocessor automatically takes this into account and corrects for the fact that

Decimal		BCD		Hex
79	=	01111001	79 =	01111001
+12	=	00010010	12 =	00010010
91	=	10010001	8B =	10001011

The only difference between Hex and BCD representation is that the microprocessor automatically adjusts for the fact that BCD does not allow for Hex values A - F during add and subtract operations.

The offset which follows a branch instruction is in signed two's complement form which means that

$+50 = +80 = 01010000
and $-50 = -80 = 10110000
Proof = 00000000

The sign for this operation is in bit 7 where an 0 equals positive and a 1 equals negative.

This bit is correct for the two's complement representation but also flags the microprocessor whether to carry or borrow from the address high byte.

The following four examples represent the combinations of offsets which might occur (all notations are in hexadecimal):

Example H.4.1: Forward Reference, No Page Crossing

0105	BNE
0106	+55
0107	Next OP CODE

To calculate next instruction if the branch is taken

Offset	+55	01010101
Address Low for Next OP CODE	07	00000111
	5C	01011100

with no carry, giving 015C as the result.

Example H.4.2: Backward Reference, No Page Crossing

015A	BNE
015B	-55
015C	Next OP CODE

To calculate if branch is taken,

Offset	-55 =	AB =	10101011
+ Address Low for Next OP CODE	+5C =	5C =	01011100
	07	07	00000111

The carry is expected because of the negative offset and is ignored, thus giving 0107 as the result.

Example H.4.3: Backward Reference If Page Boundary Crossed

0105	BNE
0106	-55
0107	Next OP CODE

To calculate if branch is taken, first calculate a low byte

Offset	-55 =	AB =	10101011
Address Low for Next OP CODE	07 =	07 =	00000111
	B2 =	B2 =	10110010

There is no carry from a negative offset; therefore, a carry must be made:

	-1 =	-1 = FF =	11111111
+ Address High	=	01 = 01 =	00000001
		00 00	00000000

This gives 00 B2 as a result.

Example H.4.4: Forward Reference Across Page Boundary

00B0	BNE
00B1	+55
00B2	Next OP CODE

To calculate next instruction if branch is taken,

```
Offset          55 = 01010101
Address Low
  for Next
  OP CODE       B2 = 10110010
                07   00000111
```

with carry on positive number.

```
                +1    1 = 00000001
Address High    00 = 00000000
                 1 = 00000001
```

which gives 0107.

CUT ALONG THIS LINE

DOCUMENT REGISTRATION FORM

Please fill out and return this card to automatically receive all updates to your manual.

DOCUMENT NAME:

DOCUMENT NUMBER:

REVISION:

(Name)

(Company)

(Street Address)

(City) (State) (Zip)