68HC11 Microcontroller Instruction Set

Microcontroller Instruction Set

- Data Handling Group
- Arithmetic and Logic Group
- Branching and Decision Group
- Input/Output group
- Special

Data Handling Group

This group of instructions is used to initialize, alter the contents of a register or memory location, to move data between registers, to move data between a register and memory, or between two memory locations, etc.

Examples:	LOAD
	STORE
	CLEAR

Arithmetic and Logic Group

This group of instructions is used to perform a specific Arithmetic or logic operation over one or more operands.

Examples: ADD
SUBSTRACT
MULTIPLY
OR
AND, etc

Branching and Decision Group

This group of instructions is used to control the execution of the program

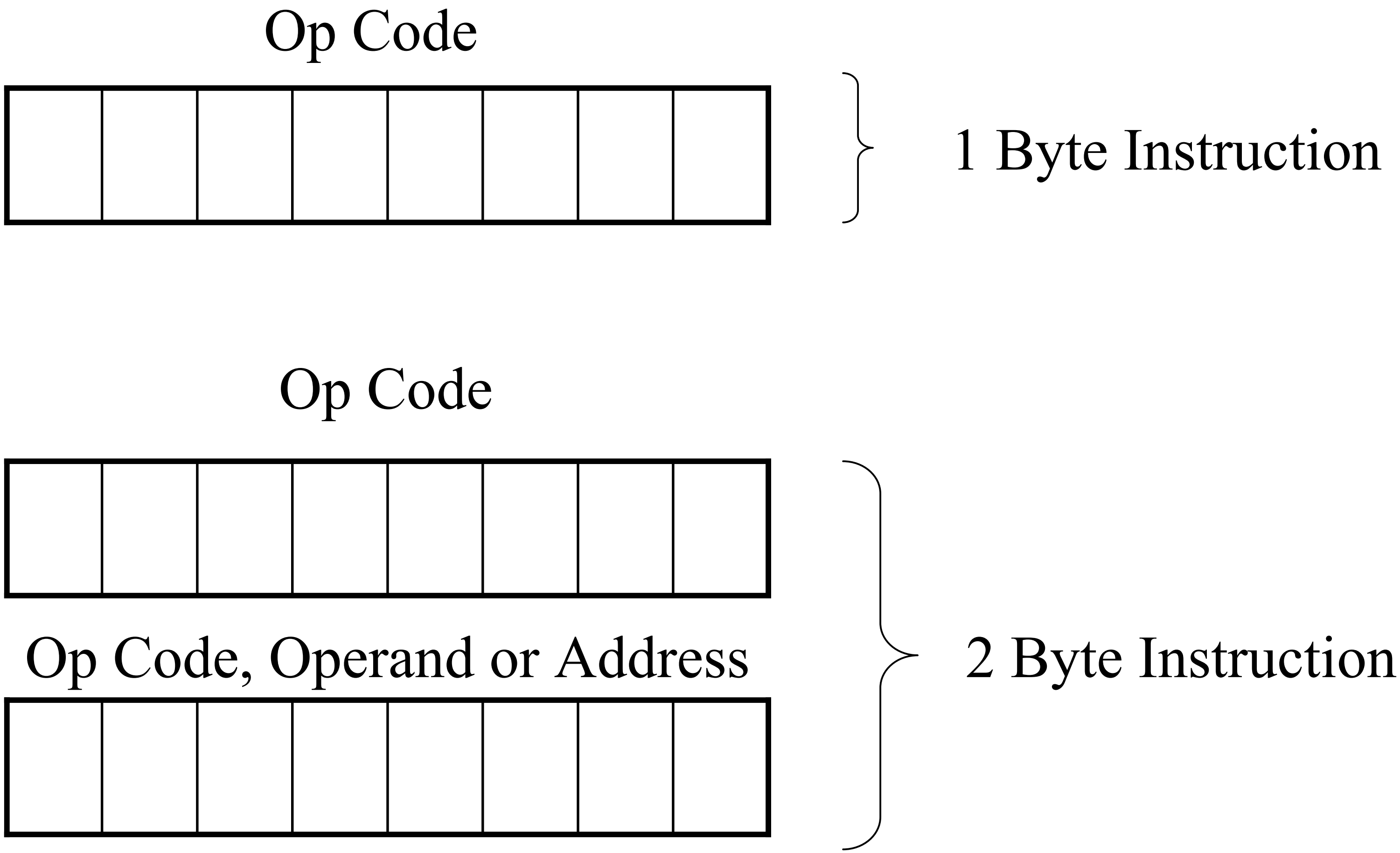
Examples: BRANCH
JUMP
CALL, etc

Special Instruction Group

This class of instructions is very close related to the hardware of the CPU, and may control some internal signals.

Example: WAIT
NO OPERATION
INTERRUPT ENABLE, etc

Instruction Formats



Op Code

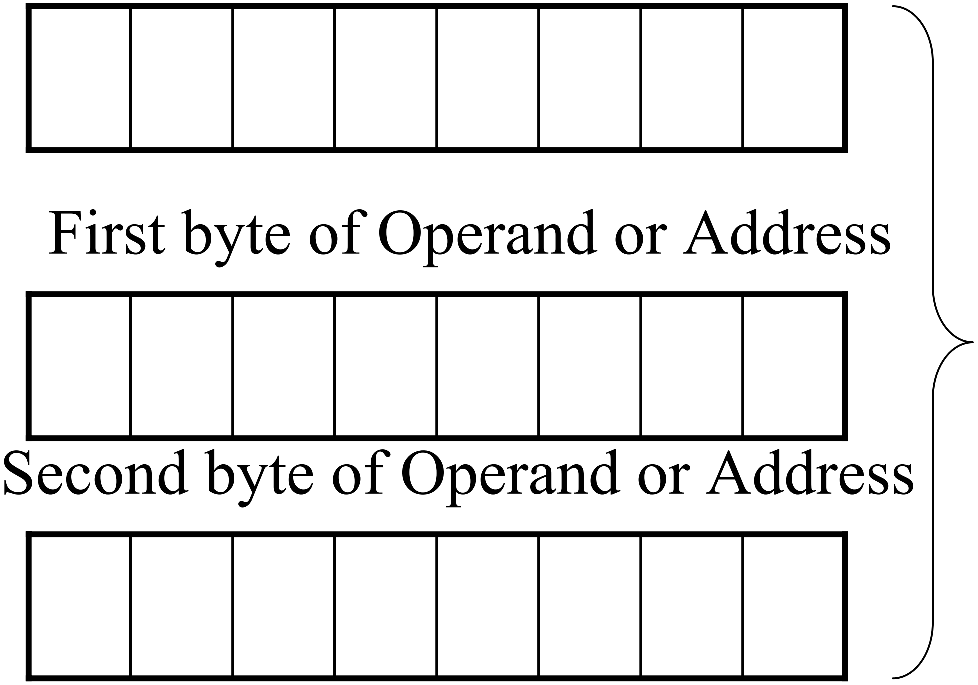
1 Byte Instruction

Op Code

Op Code, Operand or Address

2 Byte Instruction

Instruction Formats



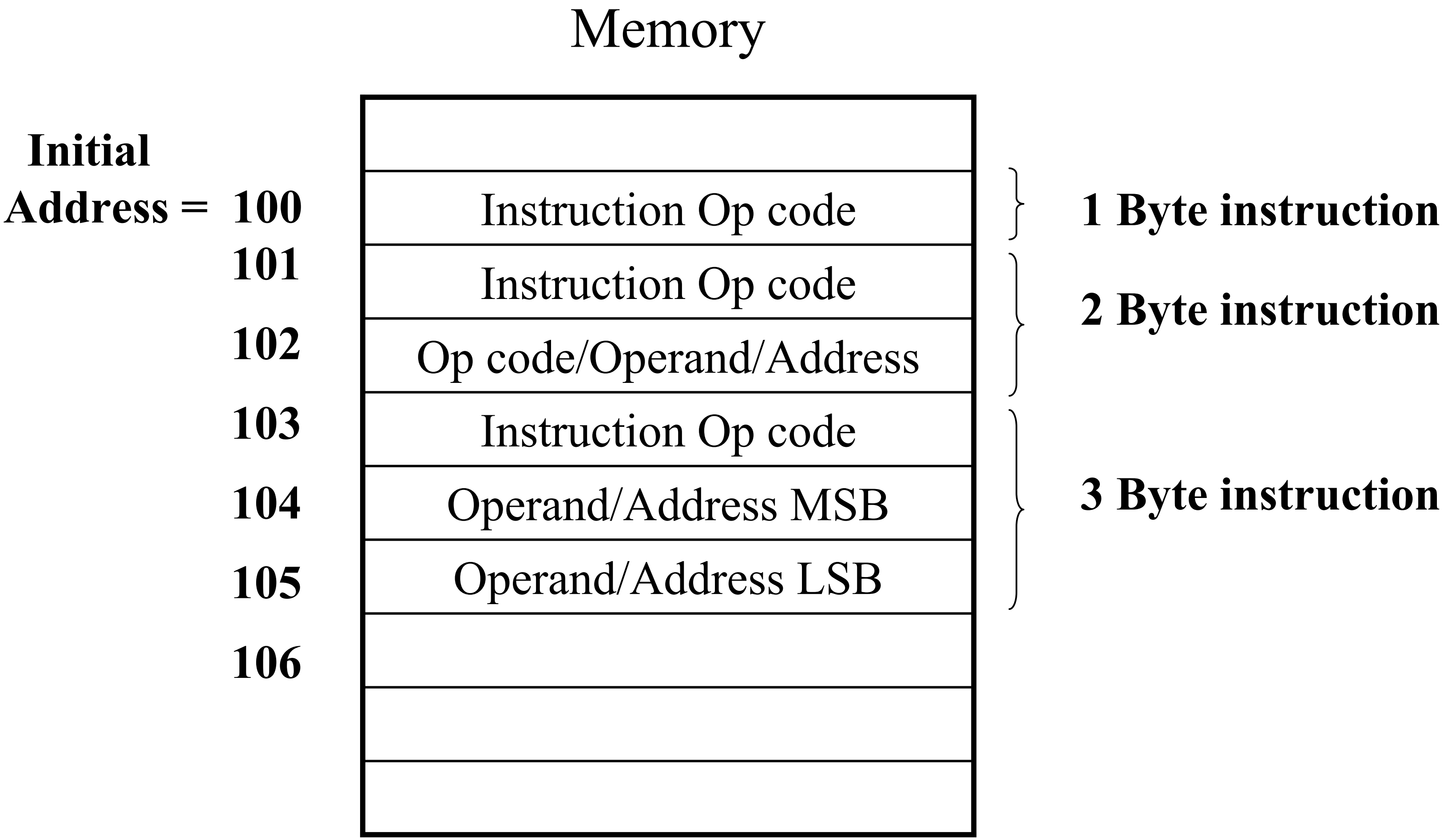
Op Code

First byte of Operand or Address

Second byte of Operand or Address

Three Byte
Instruction

Storing of Instructions in Memory



Memory

Initial Address =	Memory	
100	Instruction Op code	1 Byte instruction
101	Instruction Op code	2 Byte instruction
102	Op code/Operand/Address	
103	Instruction Op code	3 Byte instruction
104	Operand/Address MSB	
105	Operand/Address LSB	
106		

Addressing Modes

Whenever an instruction operates on a data or operand, we need to specify the location were the data resides.
An addressing mode refers to the way we specify the address or location where the data is stored

- Inherent
- Immediate
- Direct
- Indexed

Inherent Addressing

The instruction itself implies the location of the operand. Inherent addressing always deals with data or operands stored in registers not in memory.

Examples:

Address	Code	Mnemonic	Instruction Actions
0100	4F	CLRA	; Clear Acc A
0101	5C	INCB	; Increment Acc B
0102	18 09	DEY	; Decrement register IY
0104	1B	ABA	;Add Acc A to Acc B
0105	18 8F	XGDY	; Swap contents of Acc D with IY

Immediate Addressing

The instruction itself contains the data or operand that is needed by The instruction.

Immediate addressing always deals with data or operands stored in registers not in memory.

Examples:

Address	Code	Mnemonic	Instruction Actions
0100	86 5C	LDAA #$5C	; Load Acc A with 5C
0102	8B 02	ADDA #$02	; Add 02 to the contents of Acc A
0104	CC 12 34	LDD #$1234	; Load register D with 1234

Note that immediate addressing is specified by putting a # before the operand

Data Numbering Definitions

- The symbols that are used to define the numbering system used in the operand field are:

a) No Symbol	Decimal Number
b) $	Hexadecimal Number
c) @	Octal Number
d) %	Binary Number

Examples

- ADDA 25 ;Add 25_d to Acc A
- ADDA \$25 ; Add 25_h (37_d) to Acc A
- ADDA @25 ; Add 25_o (21_d) to Acc A
- ADDA %00011001 ; Add 25_d to Acc A

Direct Addressing

The instruction contains the address of the memory location where the data or operand is stored.

Direct addressing always deals with data or operands stored in memory.

There are two options for direct addressing: short and extended

Examples:

```
Address  Code      Mnemonic         Instruction Actions
 0100    96 5C     LDAA $5C       ; Load acc A with data in location 5C
 0102    8B 02     ADDA $02       ; Add the contents of loc 02 to acc A
 0104    B6 00 05   LDAA $005C    ; Load acc A with data in location 5C
```

Indexed Addressing

❖ The address of the memory location where the data or operand is stored, is specified by using a special register.

❖ There are two registers that can be used to hold the address, and they are commonly known as index registers X and Y

❖ Indexed addressing always deals with data or operands stored in memory.

❖ This is the most complex type of addressing that the micro-controller can use to access a data in memory.

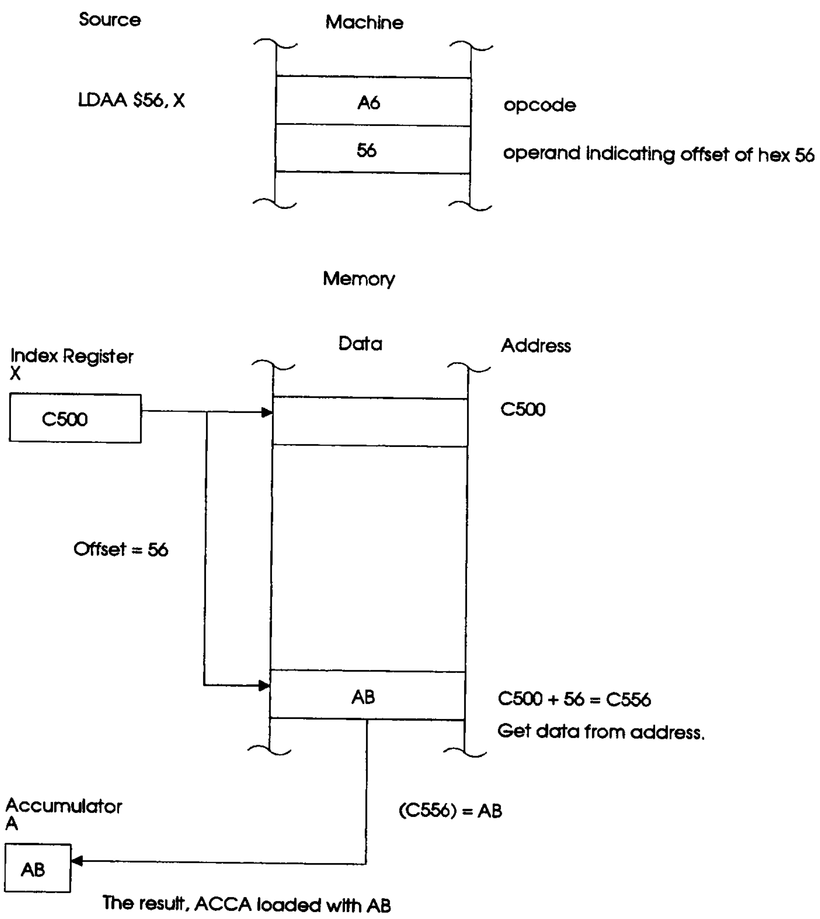
❖ Besides the address pointed by the index register, we can also specify a second number called an “offset”. This number is added to the number stored in the index register to calculate the effective address of the data that is used by the instruction

Indexed Addressing

Examples:

Address	Code	Mnemonic
0100	A6 00	LDAA $0,X
0102	18 E6 56	LDAB $56,X
0105	18 A7 05	STAA $05,Y

Instruction Actions



Source

Machine

LDAA $56, X

A6

opcode

56

operand indicating offset of hex 56

Memory

Index Register
X

C500

Data

Address

C500

Offset = 56

AB

C500 + 56 = C556

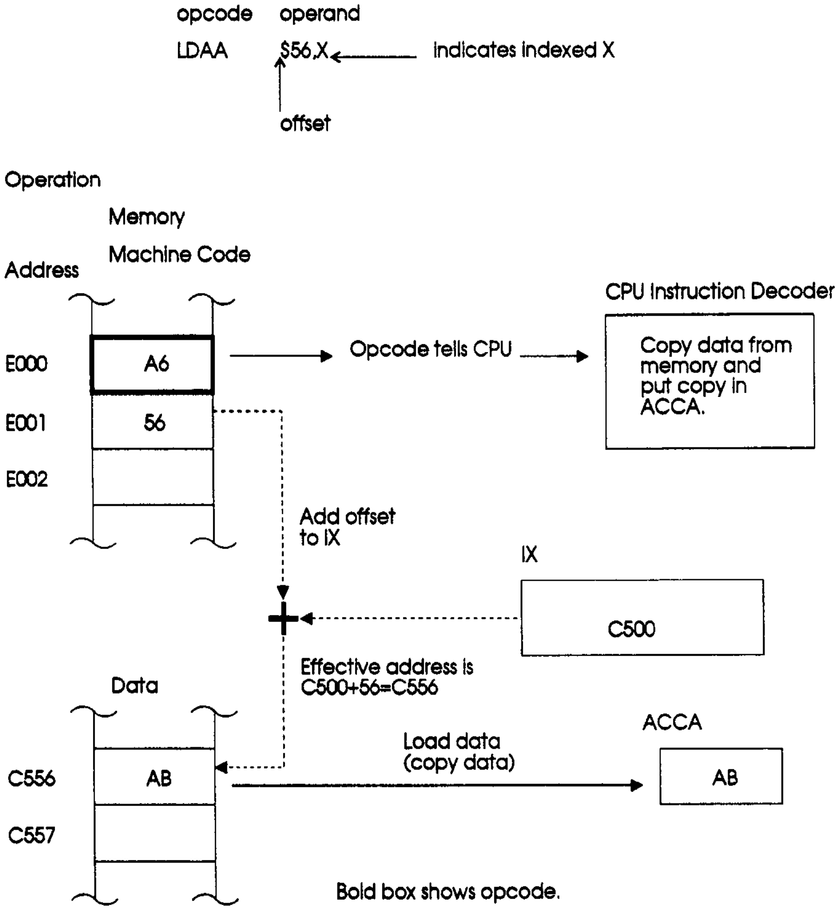
Get data from address.

Accumulator
A

(C556) = AB

AB

The result, ACCA loaded with AB



opcode operand

LDAA $56,X

indicates indexed X

offset

Operation

Memory

Machine Code

Address

CPU Instruction Decoder

E000

A6

Opcode tells CPU

Copy data from memory and put copy in ACCA.

E001

56

E002

Add offset to IX

IX

C500

Effective address is C500+56=C556

Data

ACCA

Load data (copy data)

C556

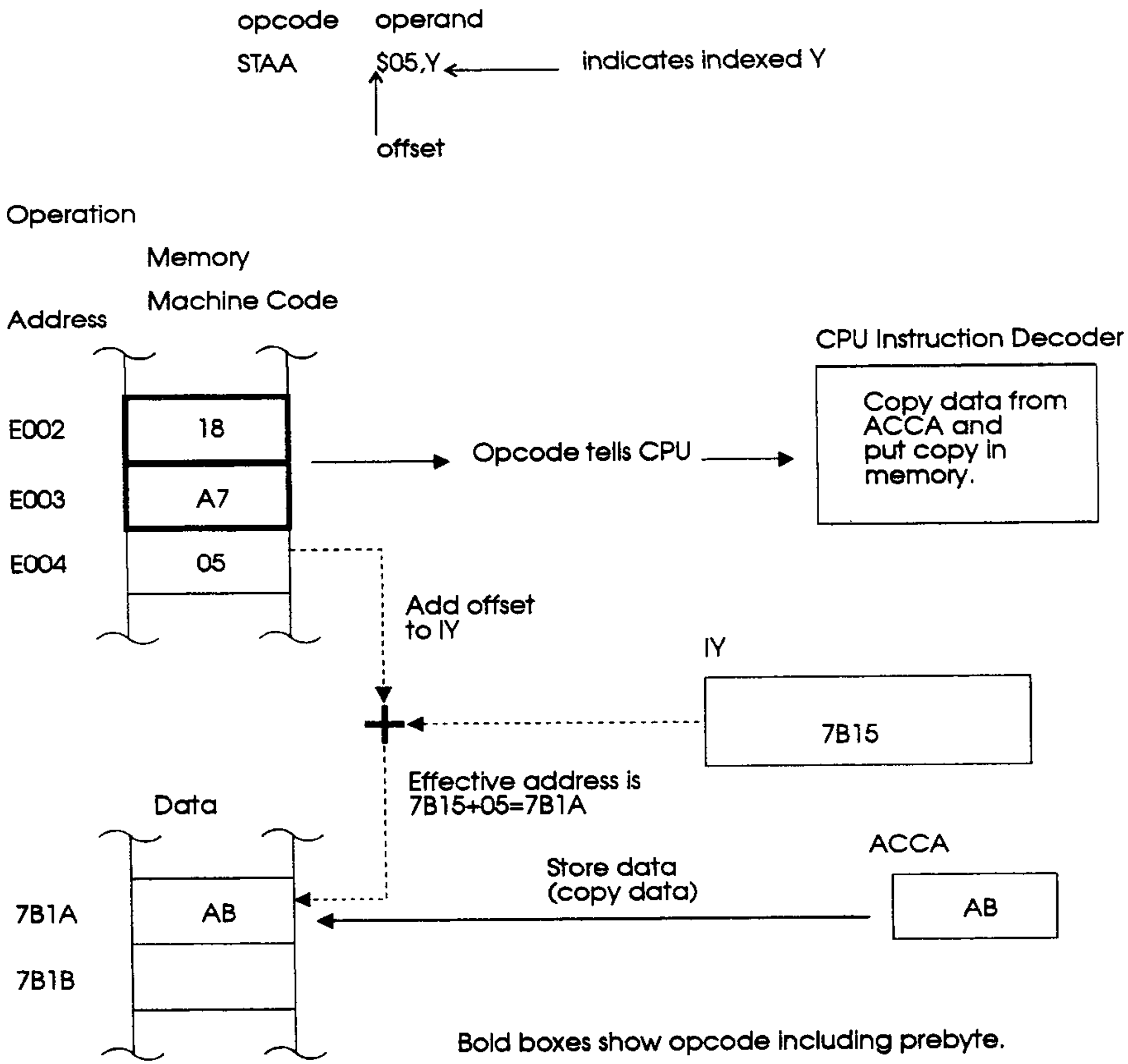
AB

AB

C557

Bold box shows opcode.

Instruction Actions



Pseudo instructions and Directives

a) Control Directives

ORG	;Specify Memory Storage Location
END	;Specify end of program

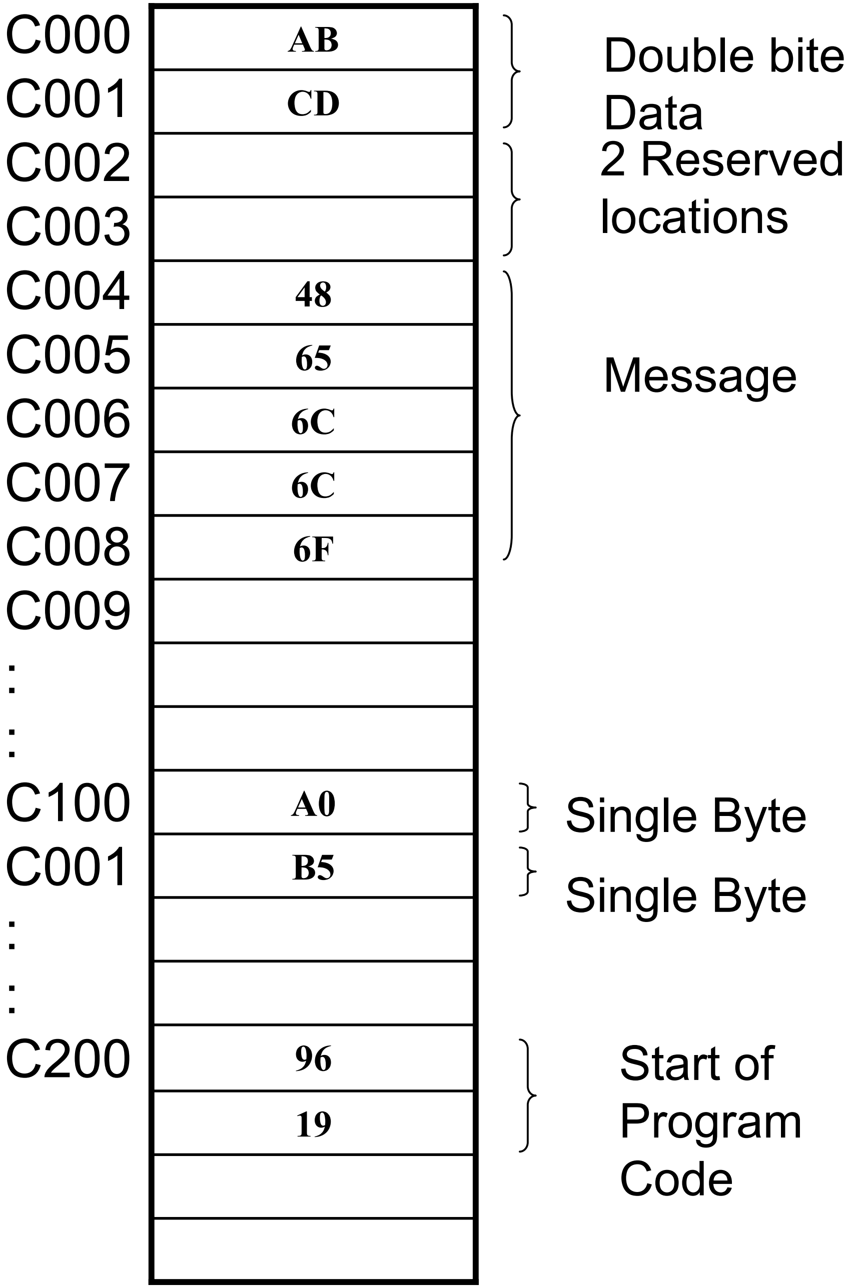
b) Data Directives

FCB	;Specify constant Byte
FDB	;Specify double constant byte
FCC	;Form Constant Character
EQU	;Assign Value to a Label
RMB	;Reserve Memory Block

Pseudo-Instruction Usage Examples

```
         ORG   $C000
NUM1     EQU   25
NUM2     EQU   $25
CONST    FDB    $ABCD
MESSA    FCC ‘HELLO’
         ORG $C100
DATA1    FCB $A0
DATA2    FCB $B5
BUFFER   RMB 2
         ORG $C200
START     LDAA  NUM2
          END
```

Program in Memory



C000	AB	Double bite Data
C001	CD	
C002		2 Reserved locations
C003		
C004	48	Message
C005	65	
C006	6C	
C007	6C	
C008	6F	
C009		
:		
:		
C100	A0	Single Byte
C001	B5	Single Byte
:		
:		
C200	96	Start of Program Code
	19	

Function	Mnemonic	IMM	DIR	EXT	INDX	INDY	INH
Clear memory byte	CLR			X	X	X	
Clear accumulator A	CLRA						X
Clear accumulator B	CLRB						X
Load accumulator A	LDAA	X	X	X	X	X	
Load accumulator B	LDAB	X	X	X	X	X	
Load double accumulator D	LDD	X	X	X	X	X	
Push B onto stack	PSHB						X
Store accumulator A	STAA		X	X	X	X	
Store accumulator B	STAB		X	X	X	X	
Store double accumulator D	STD		X	X	X	X	
Load index register X	LDX	X	X	X	X	X	
Load index register Y	LDY	X	X	X	X	X	
Store index register X	STX	X	X	X	X	X	
Store index register Y	STY	X	X	X	X	X	
Transfer A to B	TAB						X
Transfer A to CCR	TAP						X
Transfer B to A	TBA						X
Transfer CCR to A	TPA						X
Exchange D with X	XGDX						X
Exchange D with Y	XGDY						X

Function	Mnemonic	IMM	DIR	EXT	INDX	INDY	INH
Add memory to A	ADDA	X	X	X	X	X	
Add memory to B	ADDB	X	X	X	X	X	
Add accumulators	ABA						X
Add with carry to A	ADCA	X	X	X	X	X	
Add with carry to B	ADCB	X	X	X	X	X	
Add memory to D (16 bit)	ADDD	X	X	X	X	X	
Increment memory byte	INC			X	X	X	
Increment accumulator A	INCA						X
Increment accumulator B	INCB						X
Increment index register X	INX						X
Increment index register Y	INY						X
Decrement memory byte	DEC			X	X	X	
Decrement accumulator A	DECA						X
Decrement accumulator B	DECB						X
Decrement index register X	DEX						X
Decrement index register Y	DEY						X
Add accumulator B to X	ABX						X
Add accumulator B to Y	ABY						X

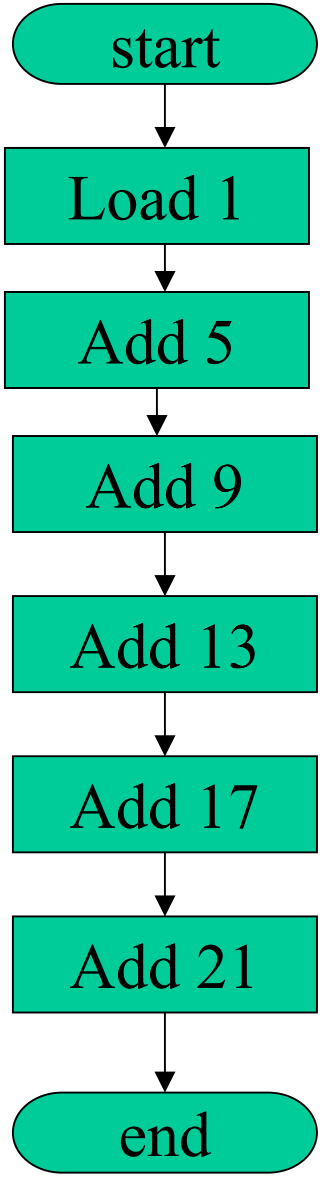
Function	Mnemonic	IMM	DIR	EXT	INDX	INDY	INH
Subtract memory from A	SUBA	X	X	X	X	X	
Subtract memory from B	SUBB	X	X	X	X	X	
Subtract with carry from A	SBCA	X	X	X	X	X	
Subtract with carry from B	SBCB	X	X	X	X	X	
Subtract memory from D (16 bit)	SUBD	X	X	X	X	X	
Compare A to B	CBA						X
Compare A to memory	CMPA	X	X	X	X	X	
Compare B to memory	CMPB	X	X	X	X	X	
Compare D to memory (16 bit)	CPD	X	X	X	X	X	
Twos complement memory byte	NEG			X	X	X	
Twos complement accumulator A	NEGA						X
Twos complement accumulator B	NEGB						X
Test for zero or minus	TST			X	X	X	
Test for zero or minus A	TSTA						X
Test for zero or minus B	TSTB						X

Program Example 1

Write a program in assembly language to add the numbers that are stored in memory at locations C100 – C1005

Address	Memory
C100	1
C101	5
C102	9
C103	13
C104	17
C105	21

Program



```
Start   LOAD C100
        ADD  C101
        ADD  C102
        ADD  C103
        ADD  C104
        ADD  C105
End
```

Program Syntax

Mnemonic	Instruction	Action
LDAA	Load acc A	Moves a new data to Acc
ADDA	Add a data using A	Adds a new data to contents of acc A result is left in acc A

```
Label   Mnemonic                    Action

        ORG    $C200   ; specify starting address of program
Start   LDAA  $C100   ; loads first operant into A
        ADDA  $C101   ; adds second data
        ADDA  $C102   ; adds third data
        ADDA  $C103   ; adds fourth data
        ADDA  $C104   ; adds fifth data
        ADDA  $C105   ; adds sixth data
End
```