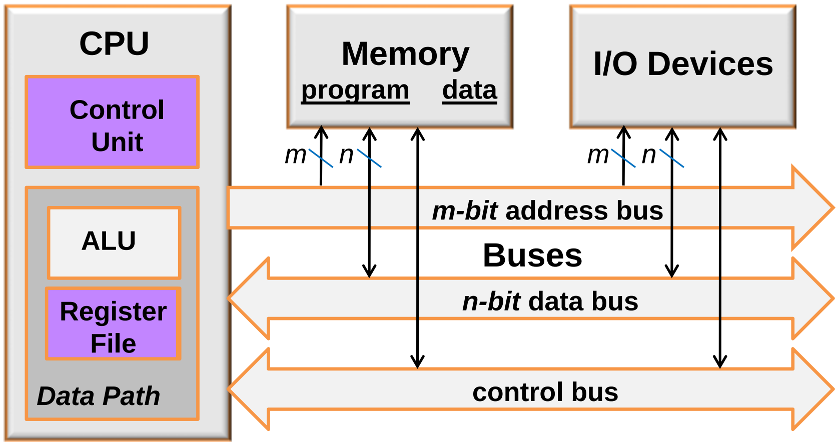
HC12/S12 Programmer's Model

- *Programmer's Model* = model of μC useful to view hardware during execution of software instructions

- Recall: General Microcontroller/Computer Architecture
  - note: Control Unit & Register File



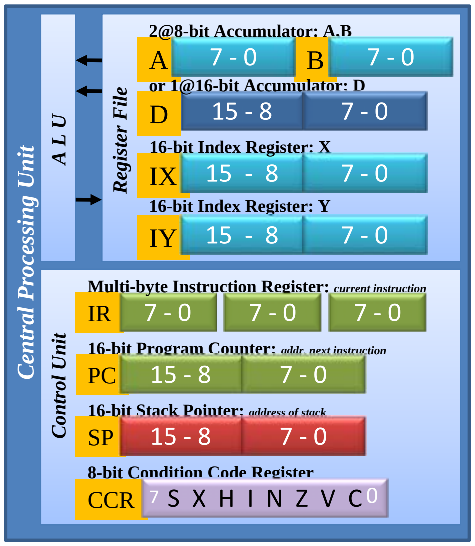
HC12/S12 Programmer's Model

- *Programmer's Model* = model of μC useful to view hardware during execution of software instructions
  - focus on Register File & Control Unit of a specific controller
    - will be different for each different brand/model of controller

Register File	A	7 6 5 4 3 2 1 0	2@8-bit Accumulator: A,B
	B	7 6 5 4 3 2 1 0	
	D	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0	or 1@16-bit Accumulator: D
	IX	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0	16-bit Index Register: X
	IY	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0	16-bit Index Register: Y
Control Unit	IR	7 . . . 0 7 . . . 0 7 . . . 0 7 . . . 0	Multi-byte Instruction Register *current instruction*
	PC	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0	16-bit Program Counter *address of next instruction*
	SP	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0	16-bit Stack Pointer *address of stack*
	CCR	7 S X H I N Z V C 0	8-bit Condition Code Register

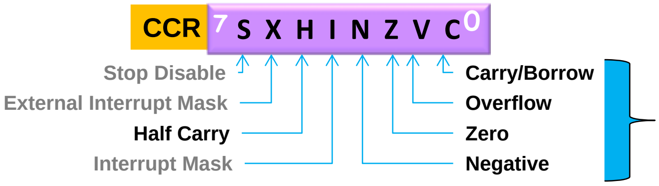
HC12/S12 Programmer's Model

- Register File
  - store data in/out of memory or ALU
  - Accumulators: 2 @ 8-bit OR 1 @ 16-bit
    - general purpose storage
  - Index Registers: 2 @ 16-bit
    - general purpose or in indexed addressing
- Control Unit
  - Instruction Register (IR)
    - holds current instruction, multi-byte
  - Program Counter (PC)
    - holds address of next instruction, 16-bit
  - Stack Pointer (SP)
    - holds address of stack (special block of memory), 16-bit
  - Condition Code Register (CCR)
    - holds "flag" values generated by last instruction executed, 8-bit
    - known as Status Register in other controllers



Condition Code Register

- CCR = register 8 of individually functioning bits
  - AKA: Flags Register, Status Register



BLACK = set by ALU
GRAY = other; covered later

- ALU Status Flags: each condition tested after each instr. exec.
  - C: Carry Flag
    - = 1 if *carry* (addition) or *borrow* (subtraction) occurs in instr. exec.
  - V: Overflow Flag

$$2C\_Overflow = \overline{A_{n-1}} \cdot \overline{B_{n-1}} \cdot S_{n-1} + A_{n-1} \cdot B_{n-1} \cdot \overline{S_{n-1}}$$

    - = 1 if 2's complement overflow occurs in instr. exec.
  - Z: Zero Flag: =1 if ALU result = 0
  - N: Negative Flag: =1 if MSU of ALU result is 1 (i.e., negative in S2C)
  - H: Half-Carry Flag: =1 if carry from lower nibble (nibble = ½ byte =4 bits)

CCR Examples

- Instructions & resulting CCR flag status
  - 8-bit addition (hexadecimal)
    - B7 + 4A
    - 07 + F9
    - B6 + 9A

- *assign each problem to group of students to evaluate as TPS exercise*
- *write answers (C=, V=, etc) on board before next slide*

CCR Examples

- Instructions & resulting CCR flag status
  - 8-bit addition
    - B7 + 4A
    - 07 + F9
    - B6 + 9A

A)

half-carry

1111 11

1011 0111

+ 0100 1010

1 | 00000001

carry-out

C= 1 V= 0 Z= 0 N= 0 H= 1

B)

1111 111

0000 0111

+ 1111 1001

1 | 0000 0000

C= 1 V= 0 Z= 1 N= 0 H= 1

c)

111 11

1011 0110

+ 1001 1010

1 | 01010000

C= 1 V= 1 Z= 0 N= 0 H= 1

Negative + Negative = Positive ? ⇒ 2's Comp. overflow!

V (2C overflow) is ALWAYS checked, even if value is not in S2C

CCR Examples & Carry Flag Rules

- Instructions & resulting CCR flag status
  - 8-bit subtraction (hexadecimal)
    - 00 – 17

$$\begin{array}{r}\$00\\ -\$17\end{array} \rightarrow \begin{array}{r}0000\ 0000\\ -\ 0001\ 0111\end{array} \rightarrow \begin{array}{r}0000\ 0000\\ +\ 1110\ 1001\\ \hline \underline{0}|1110\ 1001\end{array} \begin{array}{l} = -0001\ 0111\\ = -\$17\end{array}$$

means Hex

Carry_out = 0, but "borrow" must occur to evaluate $00 - $17

C = 1 V = 0 Z = 0 N = 1 H = 0

- Carry Flag (C) Rules
  - * if A + B (addition), then C = carry_out (Cout)
  - * if A – B (subtraction), then C = $\overline{\text{Cout}}$ (not_Cout, inverse)
    - if A > B, no borrow is needed → C = 0
    - if A < B, borrow needed → C = 1

CCR Example: Carry Flag & Subtraction

- Evaluate: -4 – (-3)
  - operation is A – B (subt.) → C = $\overline{\text{Cout}}$
  - A < B → borrow will be needed → C should be 1

$$\begin{array}{r}-4\\ \underline{-(-3)}\end{array} \rightarrow \begin{array}{r}-4\\ \underline{+3}\end{array} \rightarrow \begin{array}{r}4^*\\ \underline{+3}\end{array} \rightarrow \begin{array}{r}(0100)^*\\ \underline{+\ 0011}\end{array} \rightarrow \begin{array}{r}1100\\ \underline{0011}\\ 0|1111\end{array} \begin{array}{l} = -(111)^*\\ = -1\end{array} \qquad \checkmark \begin{array}{r}-4\\ \underline{-(-3)}\\ -1\end{array}$$

$C = \overline{Cout}$

C = 1 V = 0 Z = 0 N = 1 H = 0

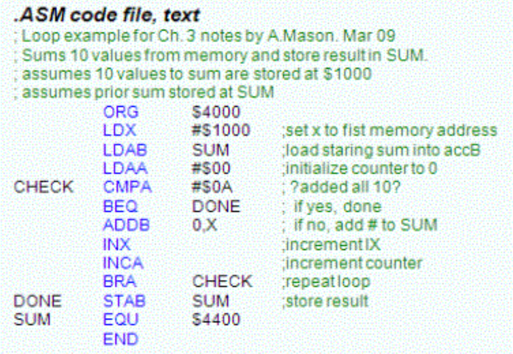
Note: in S2C form, the more negative a number is, the "smaller" it is
EX: -4 = 1100 is smaller than -3 (= 1101)

Thus, you can compare A < or > B in normal/decimal or S2C notation

HC12/S12 Assembly Instructions

- Assembly (ASM) language: programming in the smallest unit of machine (μC, μP) action

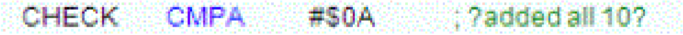
- Example ASM program code



.ASM code file, text

```
; Loop example for Ch. 3 notes by A.Mason. Mar 09
; Sums 10 values from memory and store result in SUM.
; assumes 10 values to sum are stored at $1000
; assumes prior sum stored at SUM
          ORG       $4000
          LDX       #$1000      ;set x to fist memory address
          LDAB      SUM         ;load staring sum into accB
          LDAA      #$00        ;initialize counter to 0
CHECK     CMPA      #$0A        ; ?added all 10?
          BEQ       DONE        ;  if yes, done
          ADDB      0,X         ;  if no, add # to SUM
          INX                   ;increment IX
          INCA                  ;increment counter
          BRA       CHECK       ;repeat loop
DONE      STAB      SUM         ;store result
SUM       EQU       $4400
          END
```

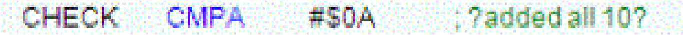
- ASM Instruction Format
  - LABEL INSTR_MNEMONIC OPERAND(S) COMMENT
  - example:



```
CHECK    CMPA     #$0A      ; ?added all 10?
```

HC12/S12 Assembly Instructions

- ASM Instruction Format
  - LABEL INSTR_MNEMONIC OPERAND(S) COMMENT
  - example:



```
CHECK    CMPA     #$0A      ; ?added all 10?
```

- LABEL: identifies location of a line of code
  - used by Assembler (compiler); not part of actual instructions
  - generally used for looping (jump, branch)
  - CHECK is branch location of later instruction (BRA CHECK)
- MNEMONIC: text code for each ASM instruction
  - translated by Assembler to a specific instr. "op-code"
    - op-code = hex machine code for each ASM instruction
  - CMPA is instruction to compare value in accA to operand value
- OPERAND: data/address used by ASM instruction
  - function of "addressing mode" (discussed later)
  - some instr. don't have operands
  - #$0A is the value accA will be compared to

HC12/S12 Assembler Notation

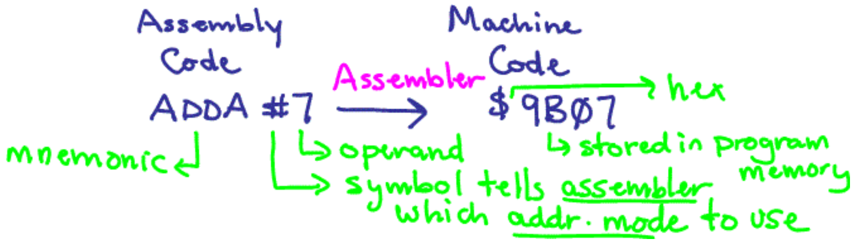
- $ = HEX value
  - EX: $12 = 18$_{10}$
- % = BIN value
  - EX: %10 = 2$_{10}$
- none = DEC value
  - EX: 12 = 12$_{10}$
- # denotes a data value (rather than an address)
  - EX: ADDA #$A5, adds value $A5 to accA
    ADDA $A5, adds value at address $A5 to accA
  - illustration to clarify will come soon!
    - first need to make things even more complicated ☺

ASM Instructions & Program Assembly

- Example ASM program code

TOP ADDA #7 ;add accA + 7
label instr. operand comment

- Program Assembly
  - instr._mnemonics converted to HEX op-codes



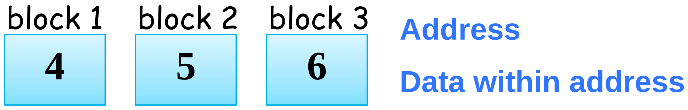
- So… HEX values can represent
  - instruction op-code bytes
  - instruction data or operand bytes
  - data/instruction address bytes
  - and you need to be able to tell which witch is which!

Clear As Mud!

- Relationship between Software (instruction) and Hardware
  - Program Elements:
    - SOFTWARE: location of instr., action to perform, data to act on
    - HARDWARE: address, CPU control bits, data bits (or address of data)

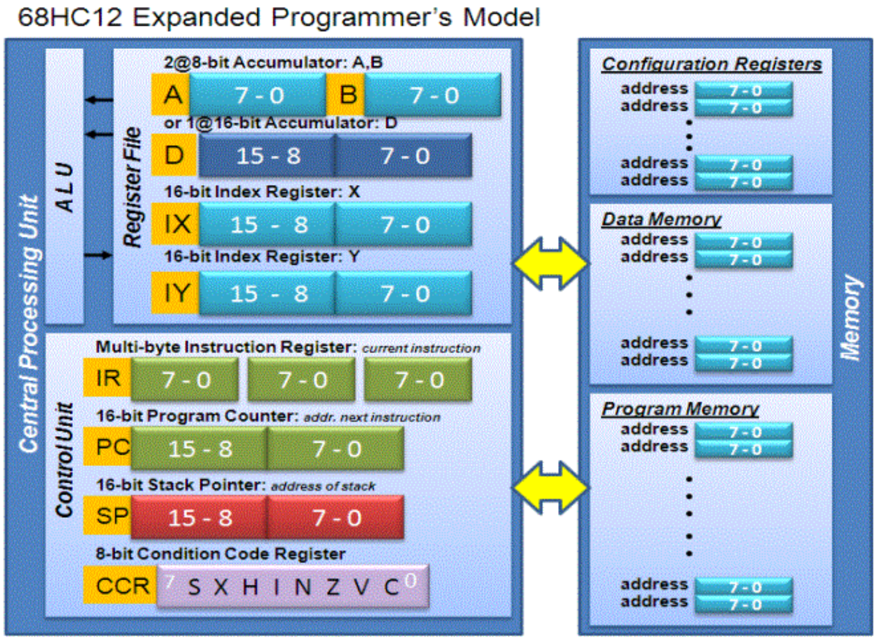
- Relationship between Program and Data memory
  - all instructions are converted to hex and stored in memory
    - Program Memory = block of memory addresses allocated to program bytes
  - most instruction act on data and produce results stored in memory
    - Data Memory = block of memory addresses allocated to data bytes

- Relationship between Address and Data values
  - evaluate:
    - #1 + #2 = #3
    - #1 + 2 = #6
      - no "#" implies an address; value in block 2 is #5, so #1+#5=#6



68HC12 Expanded Programmer's Model

- CPU (ALU, Register File, Control Unit) and Memory
  - note separate memory allocated for Program and Data



Microcontroller Program Development

from concept to action…

A. Write program to complete task
   - check syntax; test functionality (*Simulator*)

B. Assemble program (*Assembler*)
   - ASM code → Machine code (op-codes and operands)

C. Upload program to program memory

D. Run program on *Microcontroller*
   - set PC to start of program memory

   1. <u>fetch</u> instruction to IR from program memory
   2. <u>decode</u> instruction: set ALU to perform instruction
   3. <u>execute</u> instruction: load/store to data memory & register file

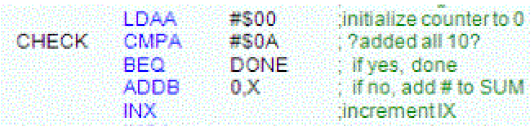
   - advance PC to next instruction in program memory
   - repeat step 1 until commanded to stop

Instruction Execution Cycle

- 3 steps of instruction execution cycle
  - Fetch: Load instruction byte from program memory into IR
  - Decode: Translate op-code to control signals for ALU and Control Unit
  - Execute: Pass data through ALU to generate desired result

- Example

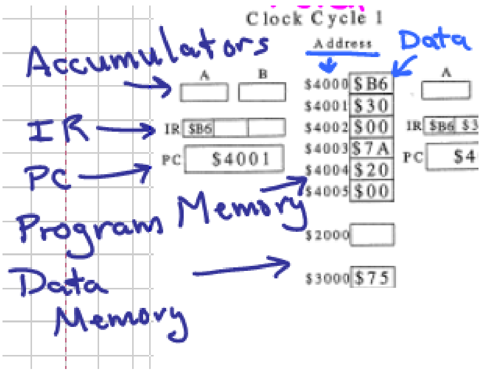
  1. fetch bytes for LDAA #$00
  2. decode: set ALU to load value #$00 into accA
  3. execute: accA contains $00
  4. fetch bytes for CMPA #$0A
  5. decode: set ALU to compare accA with the value $0A
  6. execute: set C/V/N/Z flags (e.g., if accA were $0A, Z→1)
  7. etc…



```
        LDAA    #$00    ;initialize counter to 0
CHECK   CMPA    #$0A    ; ?added all 10?
        BEQ     DONE    ;  if yes, done
        ADDB    0,X     ;  if no, add # to SUM
        INX             ;increment IX
```

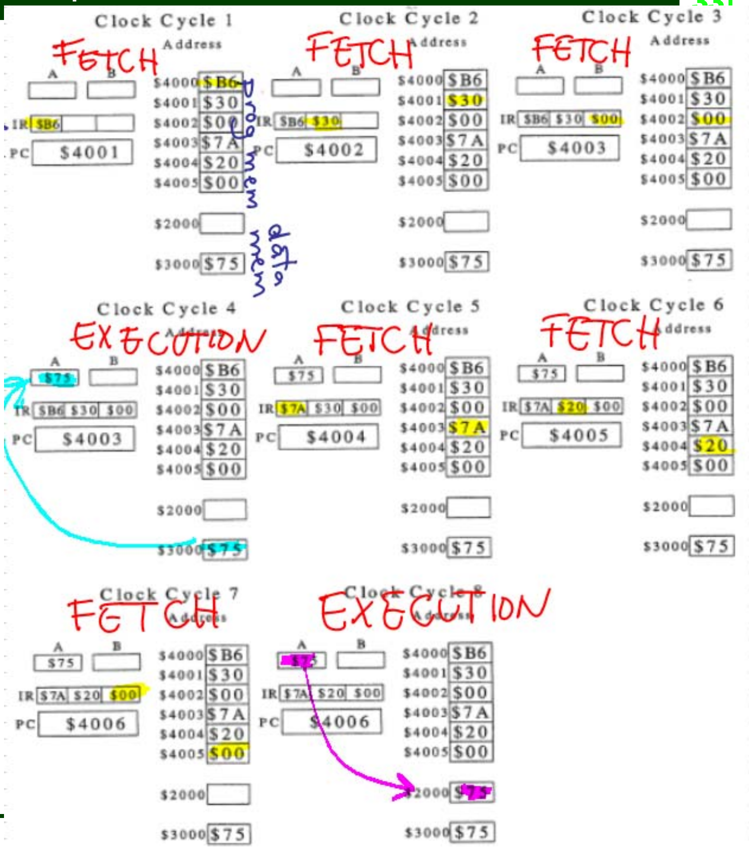
Instruction Cycle Example

- Code Executed:
  - LDAA $3000 (load from $3000)
  - STAA $2000 (store to $2000)
- Explaining the instruction execution chart



Instruction Cycle Example

- Code Executed:
  - LDAA $3000
    - (load from $3000)
  - STAA $2000
    - (store to $2000)



HC12 Instruction Set

- **Instruction Set**: the full set of functional instructions a μC/μP can execute
  - varies with each family of controllers, but generally very similar

- Basic types of instructions (see HO_3)
  - Data Transfer/Manipulation
    - move data to/from memory; shift/rotate; etc.
  - Arithmetic
    - add; subtract; increment; decrement; etc.
  - Logic & Bit Operations
    - Boolean logic; condition flag clears; etc.
  - Data Test
    - compare/test data & set CCR flags (test for conditional branches)
  - Branch
    - jump out of program sequence; if-then-else operations
  - Function Call (Subroutine)
    - start/end subroutines; adjust PC

covered later in class

Reading HO_3

- Information in HO_3 instruction tables
  - mnemonic
  - description of function
  - operation in terms of programmer's model elements
  - CCR flag affects

Data Transfer/Manipulation

Table A. Load instructions put data into CPU memory (e.g. Register File)

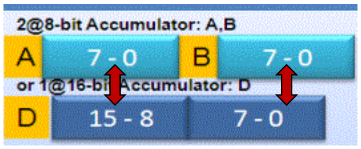
Mnemonic	*Function*	*Operation*	*C*	*V*	*Z*	*N*
LDAA	Load accumulator A	A ← M	--	0	Δ	Δ
LDAB	Load accumulator B	B ← M	--	0	Δ	Δ
LDD	Load accumulator D	A ← M, B ← M+1	--	0	Δ	Δ
LDX	Load index register X	X ← M:M+1	--	0	Δ	Δ
LDY	Load index register Y	Y ← M:M+1	--	0	Δ	Δ
LDS	Load stack pointer (SP)	SP ← M:M+1	--	0	Δ	Δ

Memory address (M) defined by instruction **operand**

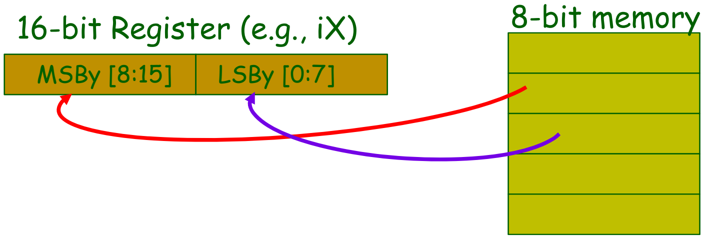
all these instructions **CLEAR** 'V' (make 0) and **CHANGE** 'Z' and 'N' based on ALU result

2-byte Operations

- Accumulators
  - accA is MSBy of accD; accB is LSBy of accD



- Memory operations
  - instructions refer to only one 8-bit memory address
    - where does 2nd byte come from for 16-bit instructions (e.g., LDX)
  - defined/reference memory address (M) → MSBy; M+1 → LSBy


16-bit Register (e.g., iX)

MSBy [8:15] | LSBy [0:7]

8-bit memory

address	e.g., M = $679D
M-1	$679C
M	$679D
M+1	$679E
M+2	$679F
M+3	$67A0

Quick Review: HC12 Instructions

Data Transfer/Manipulation

Table B. Store instructions put data (from CPU) into memory

Mnemonic	*Function*	*Operation*	*C*	*V*	*Z*	*N*
STAA	Store accumulator A	A → M	--	0	Δ	Δ
STAB	Store accumulator B	B → M	--	0	Δ	Δ
STD	Store accumulator D	A → M, B → M+1	--	0	Δ	Δ
STX	Store index register X	X → M:M+1	--	0	Δ	Δ
STY	Store index register Y	Y → M:M+1	--	0	Δ	Δ
STS	Store stack pointer (SP)	SP → M:M+1	--	0	Δ	Δ

You don't have to memorize these. Instruction tables will always be available to you. Even on exams!

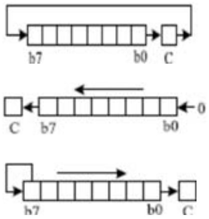
Table C. Move/transfer instructions copy data to a memory/register

Mnemonic	*Function*	*Operation*	*C*	*V*	*Z*	*N*
TAB	Transfer acc. A to acc. B	B ← A	--	0	Δ	Δ
TBA	Transfer acc. B to acc. A	A ← B	--	0	Δ	Δ
TAP	Transfer acc. A to CCR	CCR ← A	Δ	Δ	Δ	Δ
TPA	Transfer CCR to acc.A	A ← CCR	--	--	--	--
MOVB	Mem to Mem: move byte	$(M)_1$ → $(M)_2$	--	--	--	--
MOVW	Mem to Mem: move Word	$(M{:}M+1)_1$→$(M{:}M+1)_2$	--	--	--	--

Shift/rotate instructions

Logic shift: input = 0, output to carry_out
Arithmetic shift: shift right puts copy of MSB into MSB
Rotate: input rolls from output (carry_out included)

which is which?



Quick Review: HC12 Instructions

- Arithmetic
  - Addition: adds Register with Memory (or Register), stores in Register
    - note: A ← A+B, but no B ← A+B
  - Subtraction: subtr. Register with Memory (or Register), stores in Register
  - Decrement: subtr. 1 from value in (M, A, B, SP, X or Y)
  - Increment: add 1 to value in (M, A, B, SP, X or Y)
- Logic
  - AND, OR, XOR, Complement (invert)
  - 2's complement = $00 – A (or B); additive inverse, same as A'+1
- Bit Operations
  - Clear CCR flags: C, I, V
  - Bit Test; AND's A (or B) with Memory: application unknown to me ☺
  - Bit Set/Clear
    - clears (make 0) or sets (make 1) individual bits of a byte
    - can affect 1 or multiple bits
    - very useful for defining individual bits of an I/O port
    - requires a "mask" byte

Masking Concept

- Bit set/clear instructions (BSET, BCLR) [as well as some others we'll learn later] use *mask* bytes.
  - Mask byte defines which bits of a byte will be affected by instr.
    - other bits will not be changed
- Instruction format
  - mnemonic mem_addr mask
    BSET/BCLR addr of data bits to set/clr
  - if mask bit = 1 → change
  - if mask bit = 0 → don't change
- Examples:

*only operates on Memory; not on CPU registers or operands

ASM instruction

EX $4E [10100101]
addr data

BSET $4E %11110000 → $4E [11110101]
set to '1' no change

BCLR $4E %00110011 → $4E [10000100]
set to '0' = clear

because masking is bit-wise operation, mask bytes often specified in binary

BSET/BCLR Examples

- Determine memory value after each instruction

  set/clear all bits that are '1' in the mask operand

  - **BSET $9D %10101010**
    - value in $9D is $AD = %1010 1101
    - bset → $AF
  - **BSET $67B3 %11100111**
    - value in $67B3 is $00
    - bset → $E7

```
data:   1010 1101
mask:   1010 1010
bset->  1010 1111
```

red=set, blue=pass

```
data:   0000 0000
mask:   1110 0111
bset->  1110 0111
```

  - **BCLR $009E %10000001**
    - value in $009E is $E2
    - bclr → $62
  - **BCLR $009C $9C**
    - value in $9C is $9C
    - mask also $9C = %1001 1100
    - bclr → $00

```
data:   1110 0010
mask:   1000 0001
bclr->  0110 0010
```

red=clr, blue=pass

```
data:   1001 1100
mask:   1001 1100
bclr->  0000 0000
```

INITIAL

addr.	value
$009C	$9C
$009D	$AD
$009E	$E2
$67B2	$FF
$67B3	$00

FINAL

addr.	value
$009C	$00
$009D	$AF
$009E	$62
$67B2	$FF
$67B3	$E7

Quick Review: HC12 Instructions

Data Test Instructions

- Compare Data

  Compare: C= Δ, V=Δ, Z=Δ, N=Δ

  - compare 2 values by subtracting them
  - CCR flags will show <, >, or =
    - N will show which was greater

      EX: CBA → Compare A=$5D to B=$37
      → A – B → $5D - $37
      → N = 0 → A > B (subtraction not negative)

    - Z will show if they were equal
- Test Data

  Test: C= 0, V=0, Z=Δ, N=Δ

  - subtracts $00
  - CCR flags show if value in tested memory/register is negative (N) or zero (Z)

Address Modes

- 68HC12 has six (6) addressing modes
  - addressing modes define how data is associated with an instruction
  - Inherent: instruction requires no data form outside the CPU
    - EX: ABA {A ← A+B} or INX {increment iX by 1}
    - inherent instructions have no operands
      - all needed data within CPU registers
  - Immediate: data value is in operand (not in memory)
    - easily identified by having a # in the operand
    - EX: LDAA #$F3 {put $F3 into accA} or ANDA #$2C
  - Extended: data for instruction is in memory
    - 2-byte operand specifies memory address
    - EX: LDAB $20B4 {put value in $20B4 into accB; B ← <$20B4>}
  - Direct: data for instruction is in memory with 1-byte address
    - special case of Extended
      - used for addresses $0000 – $00FF (256 bytes), where MSBy is $00
      - saves program bytes and execution time; eliminates 1 operand value
      - addresses $00–$FF largely used by configuration registers (control I/O functions)
    - EX: ADDA $E3 {A ← A+<$00E3>}

remaining 2 modes are more complicated and will be covered later

<..> notation means the value at this mem. addr.

Address Modes

Identify the address mode for each of the following instructions

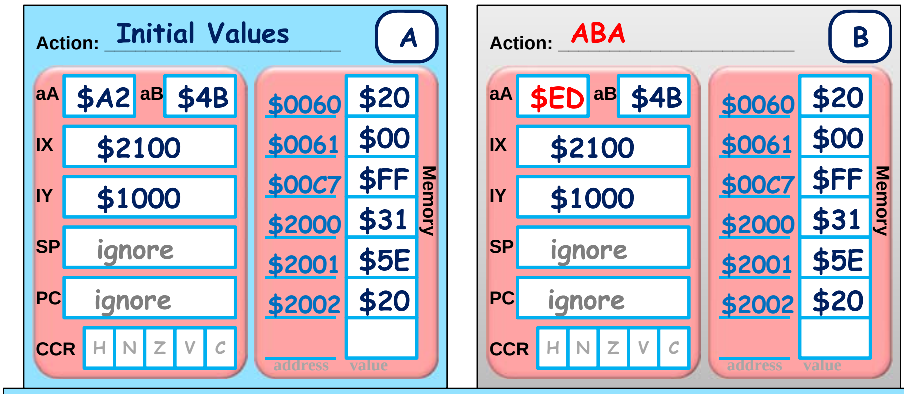
- Inherent, Immediate, Direct, Extended

- LDAB #$4F	Immediate
- STAA $8D2C	Extended
- ABA	Inherent
- LDX #$8D2C	Immediate
- STY $8D2C	Extended
- INCB	Inherent
- ANDA $C6	Direct
- TAB	Inherent
- LSL $F0	Direct
- SUBD $8D2C	Extended
- ADDD #$0750	Immediate

Quiz 1 Topics

- Following topics should be studied for Quiz 1
  - ECE230 review
    - base conversion
    - S2C form and conversions
    - Boolean logic; DeMorgan's rules
    - flip-flop/register operation
  - Microcontroller architecture; structure & name/function of blocks
  - 68HC12 programmers model
  - CCR bits; setting after hexadecimal math
  - 68HC12 instruction format & execution cycle
  - masking concept: BSET/BCLR instructions
  - 68HC12 address modes: the simple ones (INH, IMM, DIR, EXT)

Instruction Register Chart



Action: Initial Values (A)

Register	Value
aA	$A2
aB	$4B
IX	$2100
IY	$1000
SP	ignore
PC	ignore
CCR	H N Z V C

address	value
$0060	$20
$0061	$00
$00C7	$FF
$2000	$31
$2001	$5E
$2002	$20

Action: ABA (B)

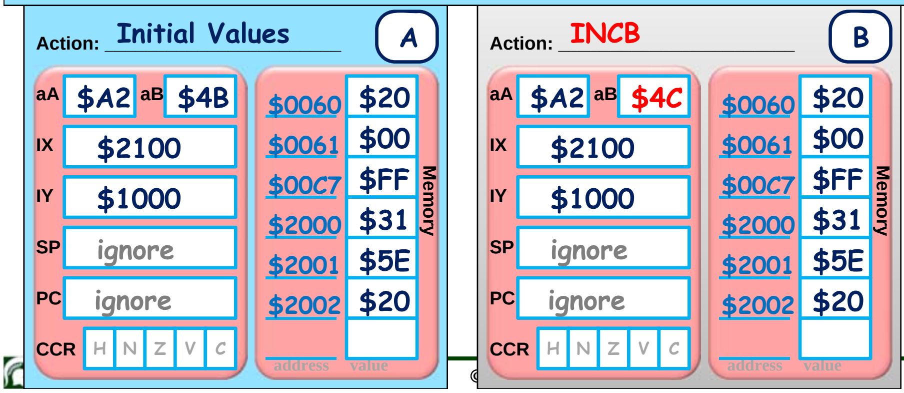
Register	Value
aA	$ED
aB	$4B
IX	$2100
IY	$1000
SP	ignore
PC	ignore
CCR	H N Z V C

address	value
$0060	$20
$0061	$00
$00C7	$FF
$2000	$31
$2001	$5E
$2002	$20

Inherent

Add B to A

A2
+4B
=ED



Action: Initial Values (A)

Register	Value
aA	$A2
aB	$4B
IX	$2100
IY	$1000
SP	ignore
PC	ignore
CCR	H N Z V C

address	value
$0060	$20
$0061	$00
$00C7	$FF
$2000	$31
$2001	$5E
$2002	$20

Action: INCB (B)

Register	Value
aA	$A2
aB	$4C
IX	$2100
IY	$1000
SP	ignore
PC	ignore
CCR	H N Z V C

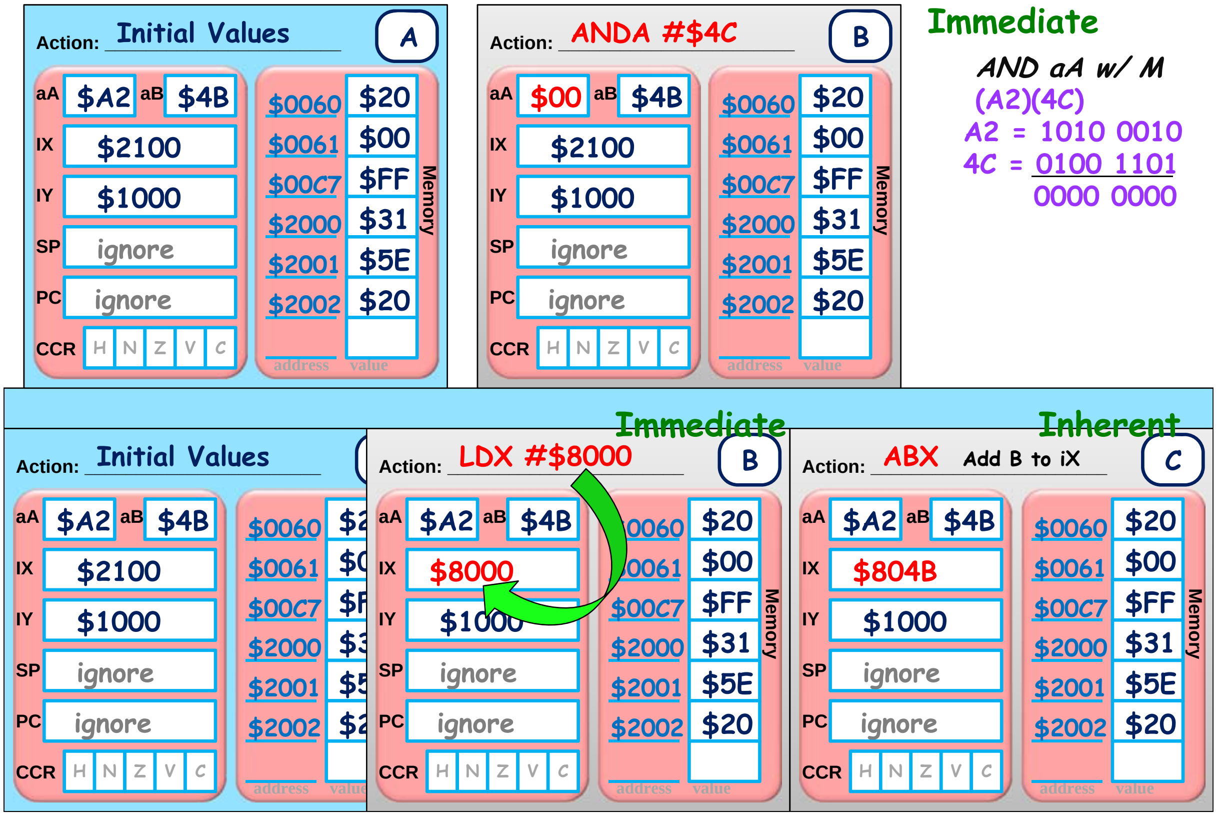
address	value
$0060	$20
$0061	$00
$00C7	$FF
$2000	$31
$2001	$5E
$2002	$20

Inherent

Increment B
(B ← B+1)

Instruction Register Chart

ECE 331



A Action: Initial Values

Reg	Value	Address	Value
aA / aB	$A2 / $4B	$0060	$20
IX	$2100	$0061	$00
IY	$1000	$00C7	$FF
SP	ignore	$2000	$31
PC	ignore	$2001	$5E
CCR	H N Z V C	$2002	$20

B Action: ANDA #$4C

Reg	Value	Address	Value
aA / aB	$00 / $4B	$0060	$20
IX	$2100	$0061	$00
IY	$1000	$00C7	$FF
SP	ignore	$2000	$31
PC	ignore	$2001	$5E
CCR	H N Z V C	$2002	$20

Immediate

AND aA w/ M

(A2)(4C)

A2 = 1010 0010

4C = 0100 1101

0000 0000

Action: Initial Values

Reg	Value	Address	Value
aA / aB	$A2 / $4B	$0060	$2
IX	$2100	$0061	$0
IY	$1000	$00C7	$F
SP	ignore	$2000	$3
PC	ignore	$2001	$5
CCR	H N Z V C	$2002	$2

Immediate

B Action: LDX #$8000

Reg	Value	Address	Value
aA / aB	$A2 / $4B	0060	$20
IX	$8000	0061	$00
IY	$1000	$00C7	$FF
SP	ignore	$2000	$31
PC	ignore	$2001	$5E
CCR	H N Z V C	$2002	$20

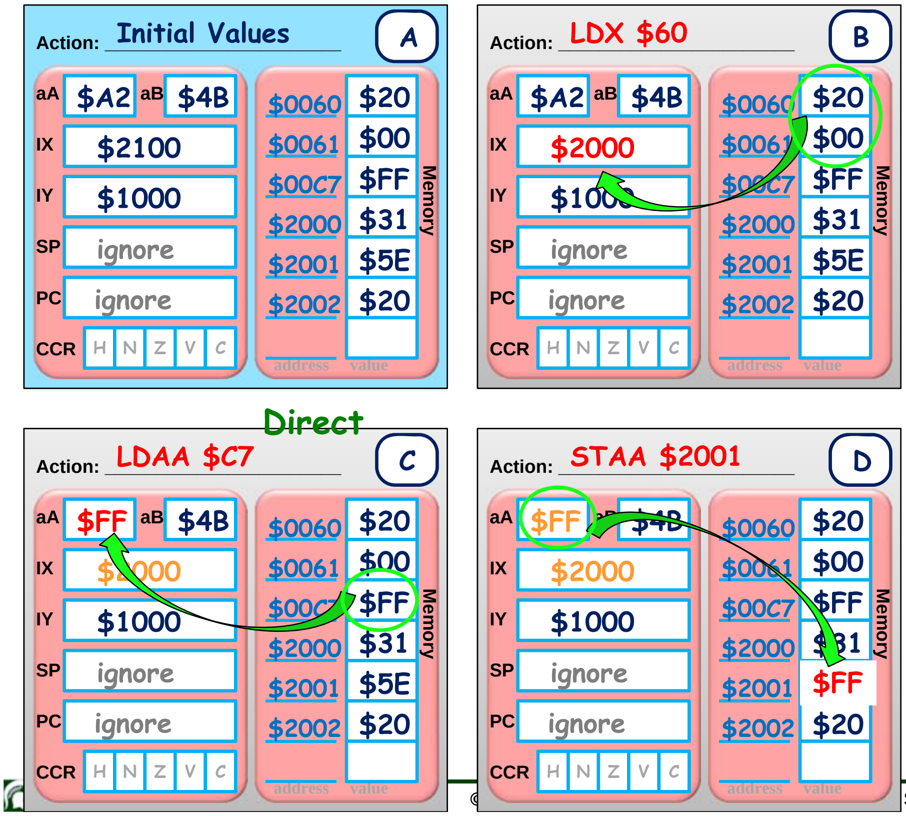
Inherent

C Action: ABX Add B to iX

Reg	Value	Address	Value
aA / aB	$A2 / $4B	$0060	$20
IX	$804B	$0061	$00
IY	$1000	$00C7	$FF
SP	ignore	$2000	$31
PC	ignore	$2001	$5E
CCR	H N Z V C	$2002	$20

Instruction Register Chart

ECE 331



A Action: Initial Values

Reg	Value	Address	Value
aA / aB	$A2 / $4B	$0060	$20
IX	$2100	$0061	$00
IY	$1000	$00C7	$FF
SP	ignore	$2000	$31
PC	ignore	$2001	$5E
CCR	H N Z V C	$2002	$20

B Action: LDX $60

Reg	Value	Address	Value
aA / aB	$A2 / $4B	$0060	$20
IX	$2000	$0061	$00
IY	$1000	$00C7	$FF
SP	ignore	$2000	$31
PC	ignore	$2001	$5E
CCR	H N Z V C	$2002	$20

Direct

Load iX from M (60)

Direct

C Action: LDAA $C7

Reg	Value	Address	Value
aA / aB	$FF / $4B	$0060	$20
IX	$2000	$0061	$00
IY	$1000	$00C7	$FF
SP	ignore	$2000	$31
PC	ignore	$2001	$5E
CCR	H N Z V C	$2002	$20

D Action: STAA $2001

Reg	Value	Address	Value
aA / aB	$FF / $4B	$0060	$20
IX	$2000	$0061	$00
IY	$1000	$00C7	$FF
SP	ignore	$2000	$31
PC	ignore	$2001	$FF
CCR	H N Z V C	$2002	$20

Extended

Instruction Register Chart: CCR Flags

ECE 331

Action: ABA

B

aA	$ED	aB	$4B
IX	$2100		
IY	$1000		
SP	ignore		
PC	ignore		
CCR	0 1 0 0 0		

address	value
$0060	$20
$0061	$00
$00C7	$FF
$2000	$31
$2001	$5E
$2002	$20

Memory

Action: INCB

B

aA	$A2	aB	$4C
IX	$2100		
IY	$1000		
SP	ignore		
PC	ignore		
CCR	- 0 0 x -		

address	value
$0060	$20
$0061	$00
$00C7	$FF
$2000	$31
$2001	$5E
$2002	$20

Memory

Action: ANDA #$4C

B

aA	$00	aB	$4B
IX	$2100		
IY	$1000		
SP	ignore		
PC	ignore		
CCR	- 0 1 0 -		

address	value
$0060	$20
$0061	$00
$00C7	$FF
$2000	$31
$2001	$5E
$2002	$20

Memory

Action: LDX #$8000

B

aA	$A2	aB	$4B
IX	$8000		
IY	$1000		
SP	ignore		
PC	ignore		
CCR	- 1 0 0 -		

address	value
$0060	$20
$0061	$00
$00C7	$FF
$2000	$31
$2001	$5E
$2002	$20

Memory

Instruction Register Chart: CCR Flags

ECE 331

Action: ABX Add B to iX

C

aA	$A2	aB	$4B
IX	$8048		
IY	$1000		
SP	ignore		
PC	ignore		
CCR	0 1 0 0 0		

address	value
$0060	$20
$0061	$00
$00C7	$FF
$2000	$31
$2001	$5E
$2002	$20

Memory

Action: LDX $60

B

aA	$A2	aB	$4B
IX	$2000		
IY	$1000		
SP	ignore		
PC	ignore		
CCR	- 0 0 0 -		

address	value
$0060	$20
$0061	$00
$00C7	$FF
$2000	$31
$2001	$5E
$2002	$20

Memory

Action: LDAA $C7

C

aA	$FF	aB	$4B
IX	$2000		
IY	$1000		
SP	ignore		
PC	ignore		
CCR	- 1 0 0 -		

address	value
$0060	$20
$0061	$00
$00C7	$FF
$2000	$31
$2001	$5E
$2002	$20

Memory

Action: STAA $2001

D

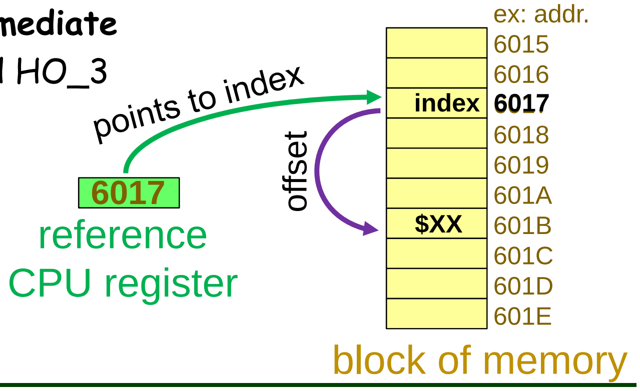
aA	$FF	aB	$4B
IX	$2000		
IY	$1000		
SP	ignore		
PC	ignore		
CCR	- 1 0 0 -		

address	value
$0060	$20
$0061	$00
$00C7	$FF
$2000	$31
$2001	$FF
$2002	$20

Memory

Indexed Address Mode

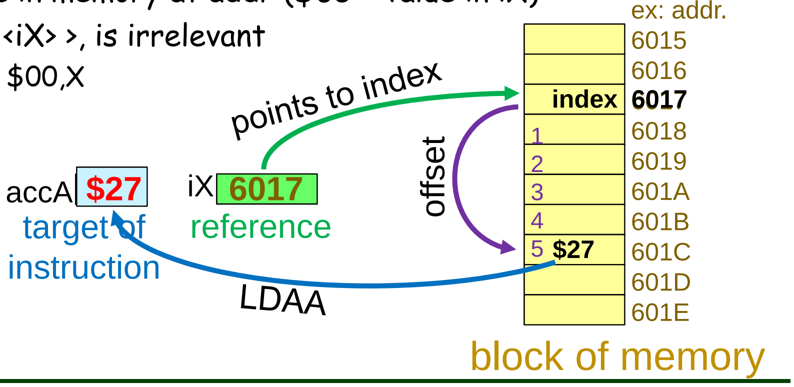
- Indexed: instruction data is in memory at address specified relative to (offset from) a reference address that is stored in a CPU register
  - reference address can be in iX, iY, SP, or PC
  - offsets are signed number → can offset forward or backward
  - useful for accessing a list of data beginning (or ending) at the reference address
- Several varieties of indexed addressing in HC12 ASM
  - we will only study **Indexed-Immediate**
  - others covered in textbook and HO_3



Indexed-Immediate Addressing

- Indexed-Immediate
  - reference address in iX, iY, SP, or PC
  - offset address in operand
    - just like Immediate address mode has data in operand
- Format: MNEMONIC offset,reference
- Example: LDX #$6017

  LDAA $05,X {accA ← <$B5+<iX>>}
    - accA loaded with value in memory at addr ($65 + value in iX)
    - note: value at index, < <iX> >, is irrelevant
      - unless instr. was LDAA $00,X



Indexed Immediate Addressing

- Example
  - LDY #$5000
  - ADDA $4C,Y

A ← A + M,
M = <$4C + <iY> >
M = **$504C**
A ← $21 + **$1A**
A ← **$3B**

INITIAL

accA	$21
accB	$80
iX	$2000
iY	$5000

reference

	addr.
B3	5000
28	5001
52	5002
00	5003
. . .	. . .
50	504A
4B	504B
1A	504C
04	504D
5E	504E

memory block

FINAL

accA	$3B
accB	$80
iX	$2000
iY	$5000

reference

	addr.
B3	5000
28	5001
52	5002
00	5003
. . .	. . .
50	504A
4B	504B
1A	504C
04	504D
5E	504E

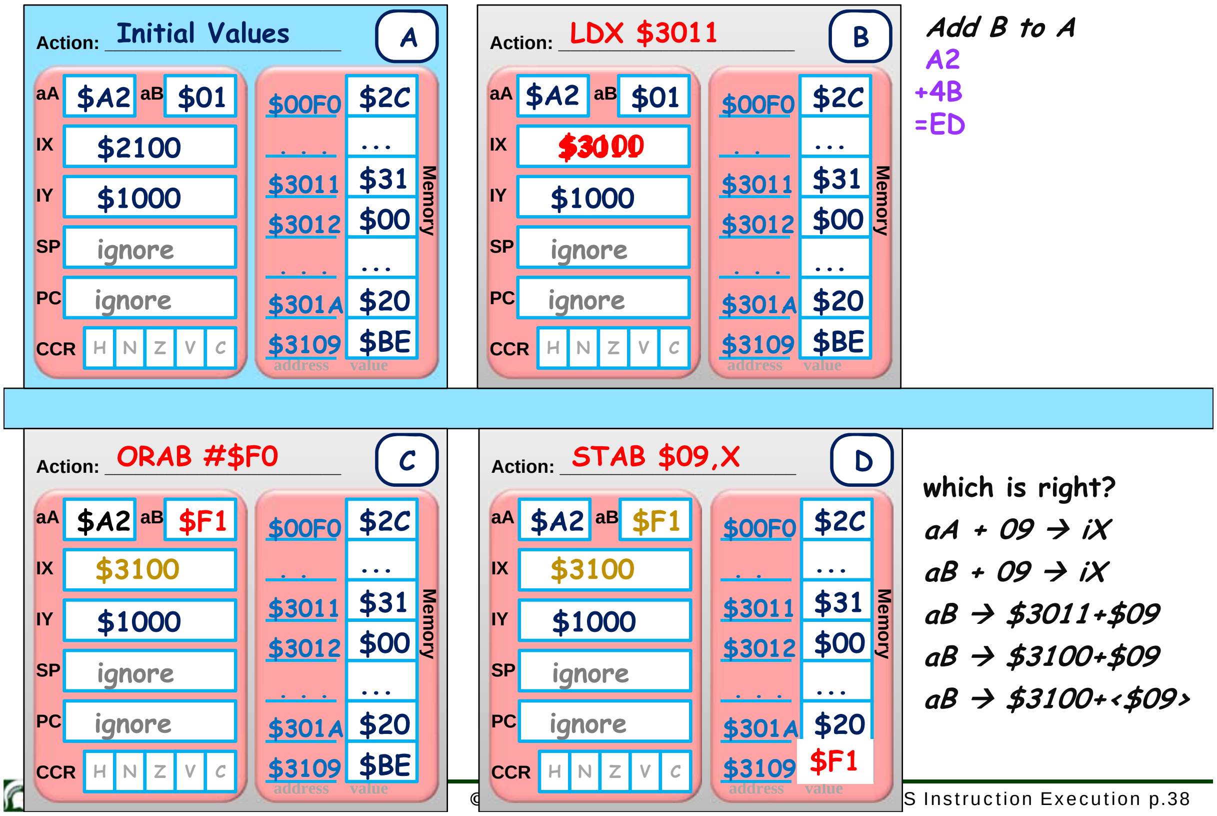
memory block

- Example (same memory as above)
  1) LDY #$5000
  2) ADDA 0,Y
  3) INY
  4) ADDA $0,Y

	1	2	3	4
accA	$21	$D4	$D4	$FC
accB	$80	$80	$80	$80
iX	$2000	$2000	$2000	$2000
iY	$5000	$5000	$5001	$5001

- can add a series of numbers like this
- reload Y to move to another set

Instruction Register Chart



Action: Initial Values (A)

aA	$A2	aB	$01
IX	$2100		
IY	$1000		
SP	ignore		
PC	ignore		
CCR	H N Z V C		

address	value
$00F0	$2C
. . .	. . .
$3011	$31
$3012	$00
. . .	. . .
$301A	$20
$3109	$BE

Memory

Action: LDX $3011 (B)

aA	$A2	aB	$01
IX	$3100		
IY	$1000		
SP	ignore		
PC	ignore		
CCR	H N Z V C		

address	value
$00F0	$2C
. . .	. . .
$3011	$31
$3012	$00
. . .	. . .
$301A	$20
$3109	$BE

Memory

Add B to A
A2
+4B
=ED

Action: ORAB #$F0 (C)

aA	$A2	aB	$F1
IX	$3100		
IY	$1000		
SP	ignore		
PC	ignore		
CCR	H N Z V C		

address	value
$00F0	$2C
. . .	. . .
$3011	$31
$3012	$00
. . .	. . .
$301A	$20
$3109	$BE

Memory

Action: STAB $09,X (D)

aA	$A2	aB	$F1
IX	$3100		
IY	$1000		
SP	ignore		
PC	ignore		
CCR	H N Z V C		

address	value
$00F0	$2C
. . .	. . .
$3011	$31
$3012	$00
. . .	. . .
$301A	$20
$3109	$F1

Memory

which is right?
aA + 09 → iX
aB + 09 → iX
aB → $3011+$09
aB → $3100+$09
aB → $3100+<$09>

HC12 Compiler Directives

- Directive: command to compiler; makes coding easier
- HC12 ASM Directives (see HO_3)

EQU equates symbol with numeric valve

-use to define memory location or constant

-assembler replaces label with correct # value

EX: LIST EQU $5B

defines variable 'LIST' = $5B

ORG origin: set memory addr. of instructions/data that follow

-all programs must specify their ORG

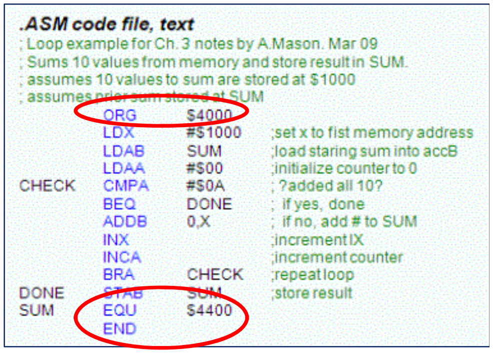
EX: TOP ORG $6000

sets program origin at $6000

END set end of program

-any ASM instructions following END are ignored

-can have directives after END



.ASM code file, text

```
; Loop example for Ch. 3 notes by A.Mason. Mar 09
; Sums 10 values from memory and store result in SUM.
; assumes 10 values to sum are stored at $1000
; assumes prior sum stored at SUM
        ORG     $4000
        LDX     #$1000    ;set x to fist memory address
        LDAB    SUM       ;load staring sum into accB
        LDAA    #$00      ;initialize counter to 0
CHECK   CMPA    #$0A      ; ?added all 10?
        BEQ     DONE      ;  if yes, done
        ADDB    0,X       ;  if no, add # to SUM
        INX               ;increment IX
        INCA              ;increment counter
        BRA     CHECK     ;repeat loop
DONE    STAB    SUM       ;store result
SUM     EQU     $4400
        END
```

HC12 Compiler Directives

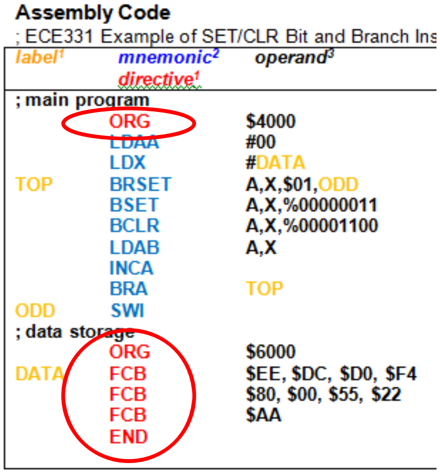
- Directive: command to compiler
- HC12 ASM Directives (see HO_3)

FCB form constant byte

-reserves block of memory & initiates contents of reserved block

EX: ABC FCB $11, $12, $13

reserves 3 bytes w/ valves $11, $12 & $13 at addr. assigned to label ABC



Assembly Code

```
; ECE331 Example of SET/CLR Bit and Branch Ins
label¹        mnemonic²     operand³
              directive¹
; main program
              ORG           $4000
              LDAA          #00
              LDX           #DATA
TOP           BRSET         A,X,$01,ODD
              BSET          A,X,%00000011
              BCLR          A,X,%00001100
              LDAB          A,X
              INCA
              BRA           TOP
ODD           SWI
; data storage
              ORG           $6000
DATA          FCB           $EE, $DC, $D0, $F4
              FCB           $80, $00, $55, $22
              FCB           $AA
              END
```

FDB form double-byte

-same as FCB but 2 bytes per operand

FCC form constant character

-stores ASCII code for alphanumeric characters enclosed in " " symbols

EX: NAME FCC "MIKE"

stores 4 ASCII bytes for MIKE

RMB reserve block of memory

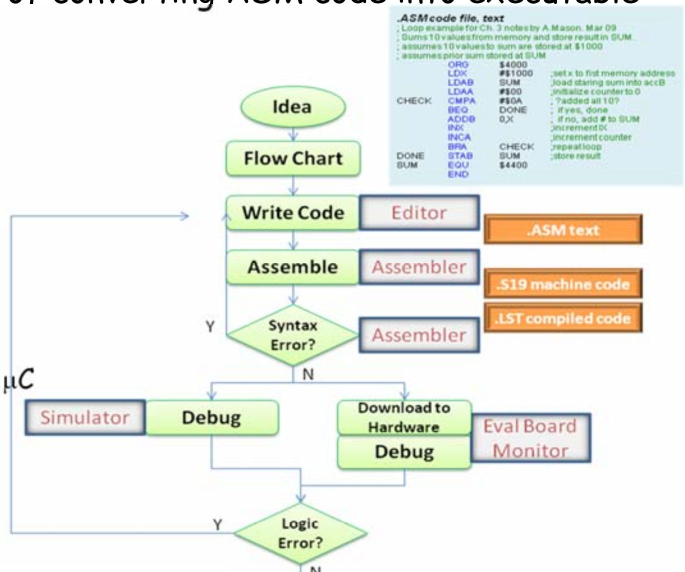
EX: TEMP RMB $10

reserves 16 ($10) bytes starting at addr. assigned to label TEMP

Assembly Process

Assembly Process: The process of converting ASM code into executable machine code.

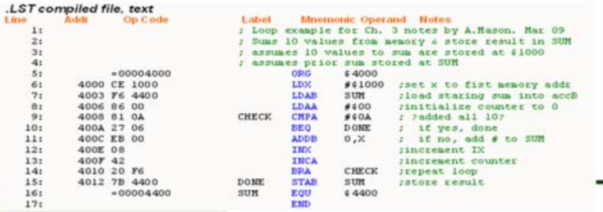
- Input
  - .**ASM** (text file)
- Outputs
  - .**LST**
    - compiled code
    - program addresses & op-codes
  - .**S19** record
    - HEX file that can be uploaded to μC to store program to memory
- Testing paths
  - Simulator
  - Test on hardware



.ASM code file, text

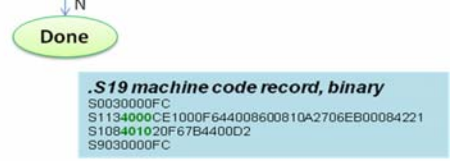
```
; Loop example for Ch. 3 notes by A.Mason. Mar 09
; Sums 10 values from memory and store result in SUM.
; assumes 10 values to sum are stored at $1000
; assumes prior sum stored at SUM
          ORG     $4000
          LDX     #$1000     ;set x to fist memory address
          LDAB    SUM        ;load staring sum into accB
          LDAA    #$00       ;initialize counter to 0
CHECK     CMPA    #$0A       ; ?added all 10?
          BEQ     DONE       ;  if yes, done
          ADDB    0,X        ;  if no, add # to SUM
          INX                ;increment IX
          INCA               ;increment counter
          BRA     CHECK      ;repeat loop
DONE      STAB    SUM        ;store result
SUM       EQU     $4400
          END
```

Idea
Flow Chart
Write Code
Editor
.ASM text
Assemble
Assembler
.S19 machine code
.LST compiled code
Syntax Error?
Y
N
Assembler
Simulator
Debug
Download to Hardware
Debug
Eval Board Monitor
Logic Error?
Y
N



.LST compiled file, text

```
Line   Addr   Op Code         Label    Mnemonic Operand   Notes
  1:                          ; Loop example for Ch. 3 notes by A.Mason. Mar 09
  2:                          ; Sums 10 values from memory & store result in SUM
  3:                          ; assumes 10 values to sum are stored at $1000
  4:                          ; assumes prior sum stored at SUM
  5:             =00004000             ORG     $4000
  6:      4000 CE 1000                 LDX     #$1000   ;set x to fist memory addr
  7:      4003 F6 4400                 LDAB    SUM      ;load staring sum into accB
  8:      4006 86 00                   LDAA    #$00     ;initialize counter to 0
  9:      4008 81 0A          CHECK    CMPA    #$0A     ; ?added all 10?
 10:      400A 27 06                   BEQ     DONE     ;  if yes, done
 11:      400C EB 00                   ADDB    0,X      ;  if no, add # to SUM
 12:      400E 08                      INX              ;increment IX
 13:      400F 42                      INCA             ;increment counter
 14:      4010 20 F6                   BRA     CHECK    ;repeat loop
 15:      4012 7B 4400        DONE     STAB    SUM      ;store result
 16:             =00004400    SUM      EQU     $4400
 17:                                   END
```



.S19 machine code record, binary

```
S0030000FC
S1134000CE1000F644008600810A2706EB00084221
S108401020F67B4400D2
S9030000FC
```

Assembly Process Example

Assembly Code **Assembled Code (.LST file)**

; ECE331 Example of SET/CLR Bit and Branch Instructions

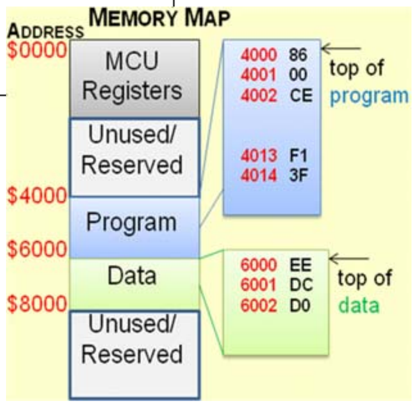
label[1]	*mnemonic*[2] *directive*[1]	*operand*[3]	*prog./data* *mem. addr.*[4]	*machine opcode*[5]
; main program				
	ORG	$4000	4000	
	LDAA	#00	4000	86 00
	LDX	#DATA	4002	CE 6000
TOP	BRSET	A,X,$01,ODD	4005	0E E4 01 0B
	BSET	A,X,%00000011	4009	0C E4 03
	BCLR	A,X,%00001100	400C	0D E4 0C
	LDAB	A,X	400F	E6 E4
	INCA		4011	42
	BRA	TOP	4012	20 F1
ODD	SWI		4014	3F
; data storage				
	ORG	$6000	6000	
DATA	FCB	$EE, $DC, $D0, $F4	6000	EE DC D0 F4
	FCB	$80, $00, $55, $22	6004	80 00 55 22
	FCB	$AA	6008	AA
	END			

Machine Code Upload Record (.S19 file)

```
S0030000FC
S11340008600CE600000EE4010B0CE4030DE40CE624
S1084010E44220F13F31
S10C6000EEDCD0F480005522AA64
S9030000FC
```

Notes:

YELLOW HIGHLIGHTS show memory addresses
GREEN HIGHLIGHTS show data for data memory
BLUE HIGHLIGHTS show ASM program opcodes to be stored in program memory



MEMORY MAP
ADDRESS
$0000
MCU Registers
Unused/ Reserved
$4000
Program
$6000
Data
$8000
Unused/ Reserved
4000 86
4001 00
4002 CE
4013 F1
4014 3F
top of program
6000 EE
6001 DC
6002 D0
top of data