Assembly Language Programming for the VCC Plus CoCo 3

A receiver reports three letters. Follow them through the processor, the stack and RAM. Then make the receiver fail on purpose and prove where it took the other branch.



The VCC Plus Hitachi processor badge.

Brian K. Holman

Book 9 · VCC Plus 3.7.1

© Brian K. Holman 2026

Contents

Read the packet . . . 3
Give the receiver a damaged signal . . . 6
Give a subroutine a contract . . . 7
Borrow a different byte at the same address . . . 9
Draw on the page you cannot see . . . 12
Move the packet with the HD6309 . . . 15
Read the return ticket an interrupt leaves . . . 17
Deliver the receiver . . . 19
Instruction directory for the 6809 and 6309 . . . 22
Registers, addressing and LWASM . . . 69
Keep a workbench notebook . . . 85
Publication and artwork notices . . . 87

Read the packet

Chapter 1 of 8

A receiver reports **ORB**. Did it print those three letters from the message it received, or were they just the programmer's favorite three letters? Catch the receiver after it has made a copy and before it decides what to print. At that instant you can ask the machine for evidence.

Set a stop on a decision

Start with a fresh **Disk BASIC** setup. Save any guest work before loading a program. Save dispatch.asm, choose **Develop → Assembly Development...**, then **File → Open Source...** and open your copy. Choose **MC6809** as the assembler target. This program also runs on an HD6309. If stepping is new to you, try the one-byte experiment first.

Choose **Assemble**, open **Build details → Listing**, and look for three labels: **packet**, **received** and **inspect**. A label names the current assembly address. It can mark data just as easily as an instruction. Select the CMPA instruction on the **inspect** line, choose **Set Stop**, then **Run**. Confirm the load. Leave the machine paused for the next section.

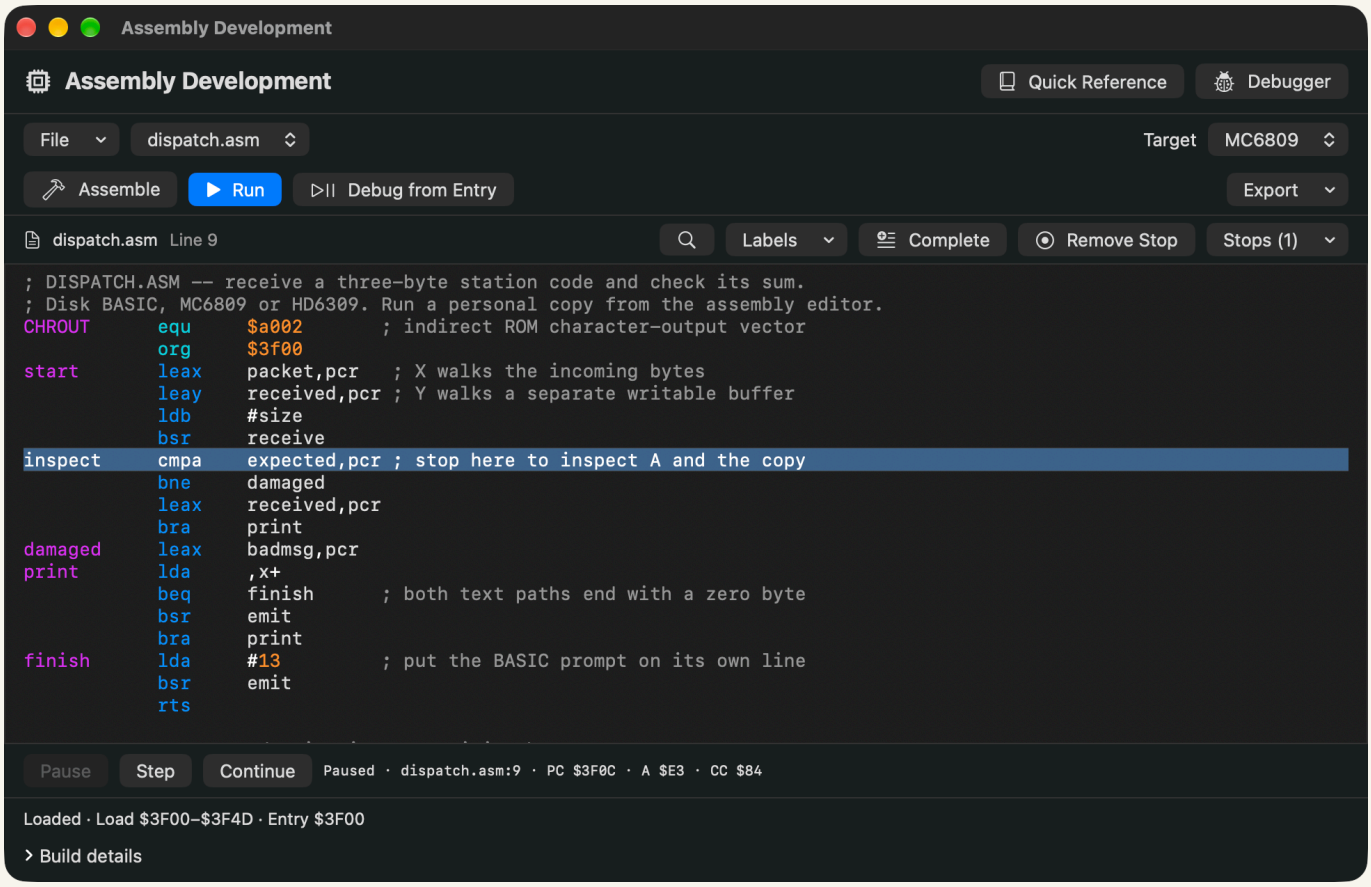


Figure 1: At the stop in dispatch.asm, the highlighted CMPA is next. The status bar already shows A=$E3.

Translate the listing into addresses

```
CHROUT      equ     $a002       ; indirect ROM character-output vector
            org     $3f00
```

```
start       leax    packet,pcr   ; X walks the incoming bytes
            leay    received,pcr ; Y walks a separate writable buffer
            ldb     #size
            bsr     receive
inspect     cmpa    expected,pcr ; stop here to inspect A and the copy
            bne     damaged
```

The **ORG** places the first instruction at $3F00. The **END** at the bottom supplies the entry label. X will point at the input and Y at the destination. **LDB #size** loads the count itself. Read **BSR receive** as "call receive and save where to come back." Execution is now back from that call. The instruction at the highlighted PC has yet to run.

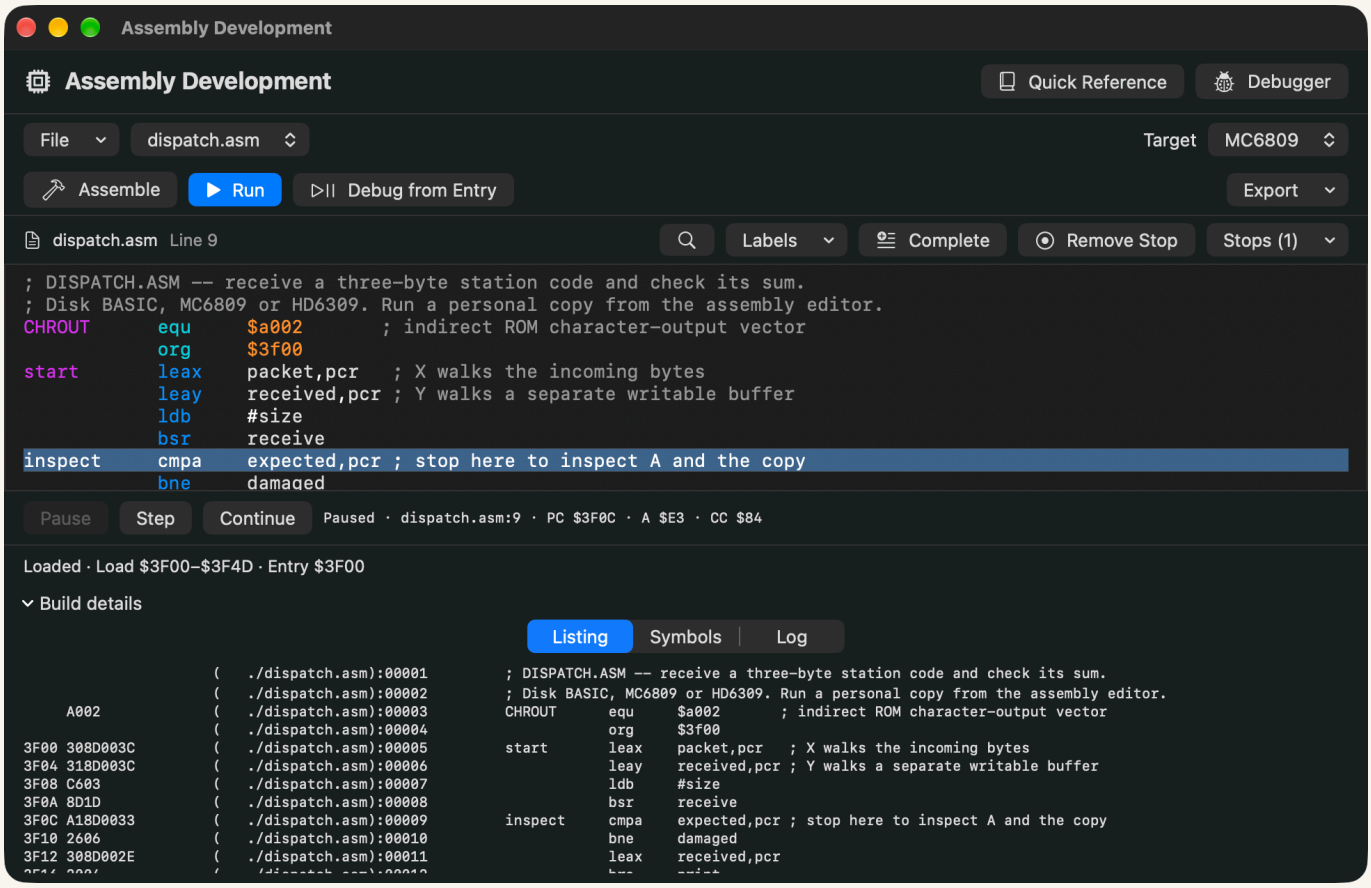


Figure 2: The assembly listing for this build shows the bytes of both LEA instructions, the call at $3F0A, and the comparison at $3F0C.

The first instruction occupies four bytes, **30 8D 00 3C**. LWASM used a signed 16-bit displacement in this build. The processor takes its position *after* the operand ($3F04) and adds $003C. That gives X the address $3F40. LEAX computes an address; it can do its job even before anyone looks at the character there. LEAY uses the same displacement but a different starting PC. Work it out from $3F08 before looking up **received**. Read your own listing after an edit: changing instruction length can move both labels.

Find the copy in RAM

In the workspace choose **Debugger**. Under **Inspector → Memory**, select **Mapped CPU**, enter **3F40** in **Address**, and choose **Go**. A row starts at the address printed on its left; the column adds the last hexadecimal digit. For this build, read across the row like this:

Addresses	Contents at the stop	What placed them here
$3F40–$3F42	4F 52 42	The assembler put the three character bytes of **packet** in the binary.
$3F43	E3	The author supplied the expected sum as data.
$3F44–$3F47	4F 52 42 00	The receive routine wrote the first three bytes. The fourth was initialized to zero and left alone.
$3F48 onward	43 48 45 43 4B 00	The other message, **CHECK**, followed by a terminator.

Open the registers. X=$3F43 and Y=$3F47 both point *one past* the last byte transferred. B has counted down to zero. In A, $E3 is a value calculated from the input. Compare it with the independent byte at $3F43. You have two pieces of evidence even though the guest has yet to print anything.

Return to the workspace and choose **Continue**. The guest prints ORB and returns to BASIC. Type **PRINT 2+3**, then Return. Seeing 5 tells you that BASIC is interpreting fresh input. Next you'll change exactly one input byte and catch the same decision.

Give the receiver a damaged signal

Chapter 2 of 8

A checksum buys you a second opinion about a message. In this receiver the sender and receiver agree that a healthy three-byte message adds up to $E3. You can change the message at the sending end while leaving that agreement in place.

Predict the next PC

Start with your portable receiver. At the bottom, **FCC "ORB"** lays down three character codes, with no length byte or terminating zero supplied by FCC. **size EQU *-packet **is the distance from the packet label to the current assembly position.** FCB $E3** supplies an expected value. Changing that value along with the message would spoil the experiment.

In the intact packet the partial sums are $4F, $A1 and $E3. Save a second copy and change only the quoted **ORB** to **ORD**. The code for D ($44) is two more than B ($42). Predict A after the third addition. Assemble this copy, set a stop on **inspect**, and Run. The destination now reads **4F 52 44 00**. The expected byte still reads **E3**.

With PC at inspect	Intact	Changed
A before CMPA	$E3	$E5
Z **after** stepping CMPA	1	0
PC after stepping BNE	$3F12, success path	$3F18, damaged path
Guest output after Continue	ORB	CHECK

The flag you see on arrival at **inspect** belongs to the earlier instructions. In particular, the last DECB made B zero on **both** runs. Step CMPA before interpreting Z as the result of the comparison. CMPA updates the condition codes as if it subtracted the expected byte; A keeps the received sum. BNE branches when Z is clear.

Ask what the sum can tell you

An eight-bit accumulator keeps the low eight bits of the sum. Carry out of the top bit is lost to the total unless a program explicitly uses it. Can this receiver distinguish the letters O, R, B in one order from the same letters in another? Work the sum on paper. Addition gives the same answer in both orders. A checksum like this can catch the edit you made. It cannot prove the message's order or authorship.

Make two correct comparisons with the same flags

Open **Develop → Assembly Quick Reference…** and choose the example **Compare signed and unsigned**. Open its independent example in the workspace. Use **Debug from Entry** on a disposable guest. Step LDA and CMPA. A holds $FF; the operand was $01. The subtraction used for the flags gives $FE. Z=0, N=1, V=0 and C=0. The comparison leaves A at $FF.

Read the next two branches before stepping them. **BLT** tests whether N differs from V after the subtraction. They differ here, so BLT goes to **signed**. The branch there, **BHI**, tests unsigned "higher": carry clear and zero clear. It goes to **both**. Branches use the flags from CMPA; taking the first branch preserved those flags for the second.

How can both be true? Read the same bits as a signed eight-bit number and $FF is −1, less than +1. Read them as an unsigned count and $FF is 255, greater than 1. The author chooses which question to ask by choosing the branch. The example holds at **both** for inspection. It deliberately loops there; use a fresh BASIC session for the next exercise. The printable receiver instead exits with RTS.

Give a subroutine a contract

Chapter 3 of 8

Imagine handing the receiver to someone who will call it from a second program. “Copies a message” leaves important questions open. Which register holds the length? Does it copy a zero? Who provides memory? Can the caller still find the next instruction after borrowing the stack?

Trace one trip around the loop

```
; Input: X source, Y destination, B positive byte count.
; Output: A = sum modulo 256. X and Y advance; B ends at zero.
receive     clra
next        adda    ,x          ; add this byte, leave X at it for the copy
            pshs    a           ; keep the running sum while moving the byte
            lda     ,x+
            sta     ,y+
            puls    a
            decb
            bne     next
            rts
```

On entry, X points at readable bytes, Y points at a separate writable destination, and B is a **positive** byte count. The caller owns both regions. It reserves one extra initialized zero for the later print routine. **receive** does its first copy before it tests B. Supplying zero here would make the first decrement wrap to $FF and the loop perform 256 transfers. That would run far past this destination. A general-purpose copy routine accepting an empty message would need an entry test. Keep the supplied three-byte count for this lab.

The first ADDA reads from X without moving X. PSHS saves the running sum because LDA needs to borrow A to carry a character. The plus in **,X+** takes the byte at the old address and then advances X by one. The store does the same for Y. PULS restores the saved sum. DECB sets the flag used by BNE. When the last iteration finishes, both pointers have advanced by the original count, B is zero, and A contains the checksum. CC has changed. Code that needs the original pointers or flags must save them or arrange to reconstruct them.

Find both kinds of saved byte

Start the portable receiver again with a stop on **BSR receive**. In the Inspector write down S. Follow the steps below in the assembly window; read memory through **Mapped CPU**. For a refresher, read the return address one byte at a time.

Stop before the instruction	Where to look	What was observed in Disk BASIC
BSR at $3F0A	S	$7F35
CLRA at the start of receive	New S and the byte above it	S=$7F33; **3F 0C** is the address of inspect.
LDA ,X+ immediately after the first PSHS A	Top three bytes starting at S	S=$7F32; **4F 3F 0C**. The one-byte sum sits below the two-byte return address.
DECB after PULS A	S and A	S=$7F33, A=$4F. The return address is again on top.
CMPA at inspect after the routine returns	S and PC	S=$7F35, PC=$3F0C. The caller can keep using its stack.

Use your own S if BASIC started with a different stack position. Set a second stop at **inspect** once you've followed the first pass; Continue lets the other passes finish. The saved sum and the saved PC use the same storage mechanism. The instruction that later consumes those bytes determines their meaning.

Choose how to address an operand

Form	What the CPU uses	Example to find
Immediate	A value embedded in the instruction	LDB #size puts 3 into B.
Direct	An eight-bit operand supplies the low byte; DP supplies the high byte	Inspect DP before using a direct-page location. A direct-page offset of $21 would name $0021 only when DP=$00.
Extended	A full 16-bit logical address	The pointer used for output is at $A002. This address names a ROM vector in this BASIC environment.
Indexed	A register supplies a base; an offset or update changes how it is used	ADDA ,X leaves X alone. LDA ,X+ advances X.
PC relative	A signed displacement is added to the PC after the operand	LEAX packet,PCR finds this program's data even if code and data move together.
Indirect	Memory holds the address to use next	The brackets in JSR [CHROUT] tell the CPU to read the two-byte destination address through the vector.

Cross the ROM boundary

```
emit        pshs    x           ; caller still needs its position after ROM I/O
            jsr     [CHROUT]
            puls    x,pc
```

Look at mapped addresses $A002 and $A003. Our Disk BASIC machine contained **A2 82**, so this call entered $A282. The source names the **vector** rather than baking that observed destination into the program. This service expects a character code in A. The caller still needs X to walk the text after the service returns; **emit** saves X and restores it. PULS X,PC also consumes the return address placed by the caller's BSR. A routine that branches into **emit** without that return address would give PULS the wrong stack layout.

ROM entry points are part of the environment. The supplied examples assume the indicated BASIC ROM is mapped and its services initialized. A program running in another OS needs its own service conventions. The linked receiver gives the copy routine a second caller without changing its contract.

Follow the same registers across an operating-system boundary in Beacon's OS-9 call contract.

Borrow a different byte at the same address

Chapter 4 of 8

A 6809 pointer fits in two bytes. It can name 65,536 locations, yet a 512 KB CoCo 3 has eight times that much RAM. The GIME, the chip that manages memory and video in the CoCo 3, chooses which physical memory answers each CPU address. You can catch it answering the same question two different ways.

Translate one address by hand

Open a fresh **Disk BASIC** setup with **512 KB**. In **Debug → Show Debugger**, choose **Inspector → MMU**. The eight rows divide the CPU's address space into windows of $2000 bytes (8 KB). Read the active task. Task 0 and task 1 are two sets of bank selections. CPU reads follow the active set when the MMU is enabled. **RAM backing** gives a physical RAM location. **Effective read** tells you whether the processor actually reads that RAM or a ROM overlay.

Find the row for CPU addresses $4000–$5FFF. On the practice machine the MMU is enabled, task 0 is active, and that row names bank $3A. The distance from $4000 to $4123 is $0123. Multiply the bank number by the bank size, then add the offset:

Selection	Start of physical bank	CPU $4123 reads at
Bank $3A	$3A × $2000 = $74000	$74123
Bank $01	$01 × $2000 = $02000	$02123

In **Memory**, switch between **Mapped CPU** and **Physical RAM**. Look up $4123 in the first and $74123 in the second. Record the values. Equal bytes are consistent with the map; two zeros alone would be poor evidence that the addresses are connected. Give the second bank a distinctive byte and the connection becomes visible.

Borrow block 2

Save bankwalk.asm and open that copy in Assembly Development. Use the fresh setup above. Before running, check that block 2 has effective RAM reads, the initial bank is $3A and the code starts at $3F00. If the mapping differs, stay with read-only inspection until you have the practice configuration. This example deliberately changes a machine register. Finish its restoration before running anything else.

```
start       pshs    cc,a            ; preserve incoming flags and A
            orcc    #$50            ; hold IRQ and FIRQ until restored
            lda     bank2
            anda    #$3f            ; 512 KB supplies six bank bits
            sta     oldbank,pcr
            lda     #1
            sta     bank2           ; window now sees physical $02123
            lda     window
            sta     oldbyte,pcr     ; keep the byte even on a fresh machine
            lda     #$a5
            sta     window
inspect     lda     oldbyte,pcr     ; pause BEFORE this instruction
            sta     window          ; put the borrowed byte back
            lda     oldbank,pcr
            sta     bank2           ; restore the original translation
```

The program saves its incoming flags and masks IRQ and FIRQ. It saves the six bank bits the 512 KB machine implements. Then it writes $01 to $FFA2, the task 0 register for block 2. From that instruction onward, an ordinary read or write at $4123 goes to the borrowed bank. The program remembers the byte already there, writes $A5, and leaves an instruction labeled **inspect** for you. The code and saved bytes live in block 1. The running BASIC stack is in block 3. Neither is moved by that register write.

Catch the mapping in the act

1. Choose **Debug from Entry**. Read the addresses of **inspect** and **restored** in **Build details**. In the supplied build they are $3F1E and $3F2C.
2. Open the machine **Debugger**. Add those two hexadecimal addresses in **Breakpoints**, then use the debugger's **Continue**. These are CPU address stops; assembly source debugging checks the memory mapping attached to its build. The address stops let you observe a deliberate mapping change.
3. At **inspect**, read the MMU row again. Task 0, block 2 now selects $01. In **Mapped CPU**, read $4123. In **Physical RAM**, read $02123 and $74123. The first two reads give $A5. The original bank still contains its old byte. Read **oldbank** and **oldbyte** from block 1 if you want to see how cleanup will work.
4. Continue to **restored**. Block 2 names $3A again, $4123 reads the original value, and $02123 holds its original byte. Remove both debugger breakpoints and Continue to BASIC. Enter `PRINT 2+3` to check the return. On the fresh practice machine both saved bytes from the two physical locations were zero. Restoration still matters when the borrowed location starts at zero.

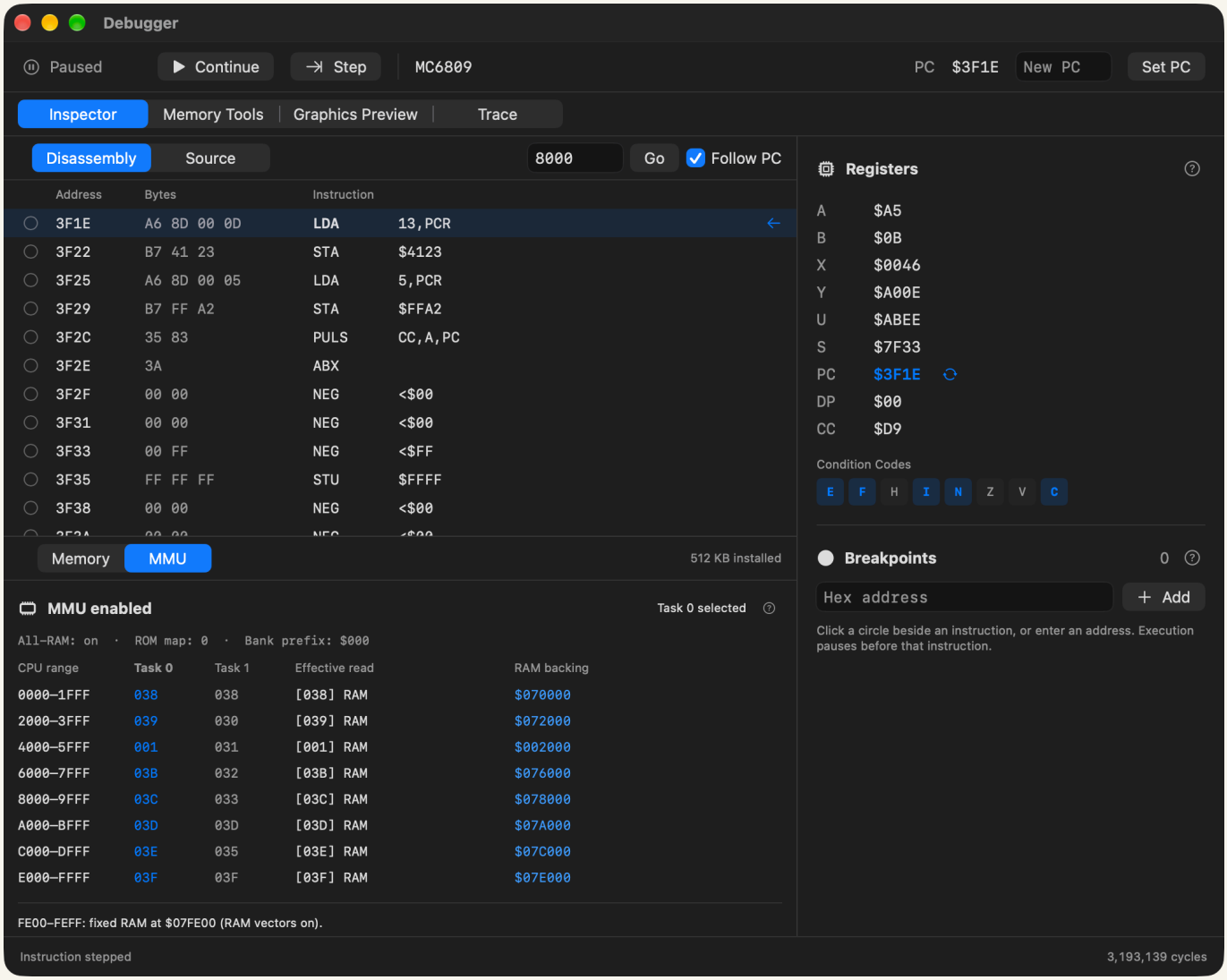


Figure 3: BANKWALK paused at $3F1E. Task 0 maps $4000–$5FFF to physical bank $001, beginning at $002000. Register A still holds the $A5 marker.

Read the exceptions on the map

An address can select a device as well as memory. A write to $FFA2 changes a bank assignment. A write to $FFB6 changes a video palette entry. A call through [$A002] fetches a two-byte destination from the BASIC character-output vector. Those three operations have very different consequences. For the top CPU block, consult the MMU view's ROM, vector and I/O notes before interpreting an address. The debugger can inspect physical RAM behind an overlay; a physical address cannot stand in for a read of a device register. Some device reads acknowledge an event. The debugger marks unavailable values where it cannot inspect safely.

The handbook's debugger guide shows the two memory views. For historical context on the CoCo 3 memory map, see Laurence A. Tepolt, *Assembly Language Programming for the CoCo 3* (TEPCO, 1987), figures 3-4 and 4-1.

Draw on the page you cannot see

Chapter 5 of 8

A display is a reader of memory. Let BASIC allocate two pictures, draw a mark in each, and leave the monitor on the first. The machine debugger can then show you a picture that nobody looking at the CoCo monitor can see. Afterward, use assembly to change a color without rewriting the letters on the screen.

Stop with both pictures in RAM

Choose fresh **Disk BASIC**, 512 KB, and **RGB** in **Settings → Video Signal**. Open PAGES.bas from Book 3 in BASIC Development, set a breakpoint at line **50**, and Run. The source starts in 32-column text, selects PMODE 1 and shows its first page. The filled square lands on the left. It then selects page 3 for drawing, clears it and draws a smaller square on the right. At the breakpoint the second page is complete. BASIC has yet to show it.

In **Debug → Show Debugger → Graphics Preview**, select **Current Video Page**. Next choose **At Address**, **Mapped CPU**, hexadecimal **1A00**, **Go**, **PMODE 1** and **Set 0**. Here are both preview buffers captured at the same stopped instruction:

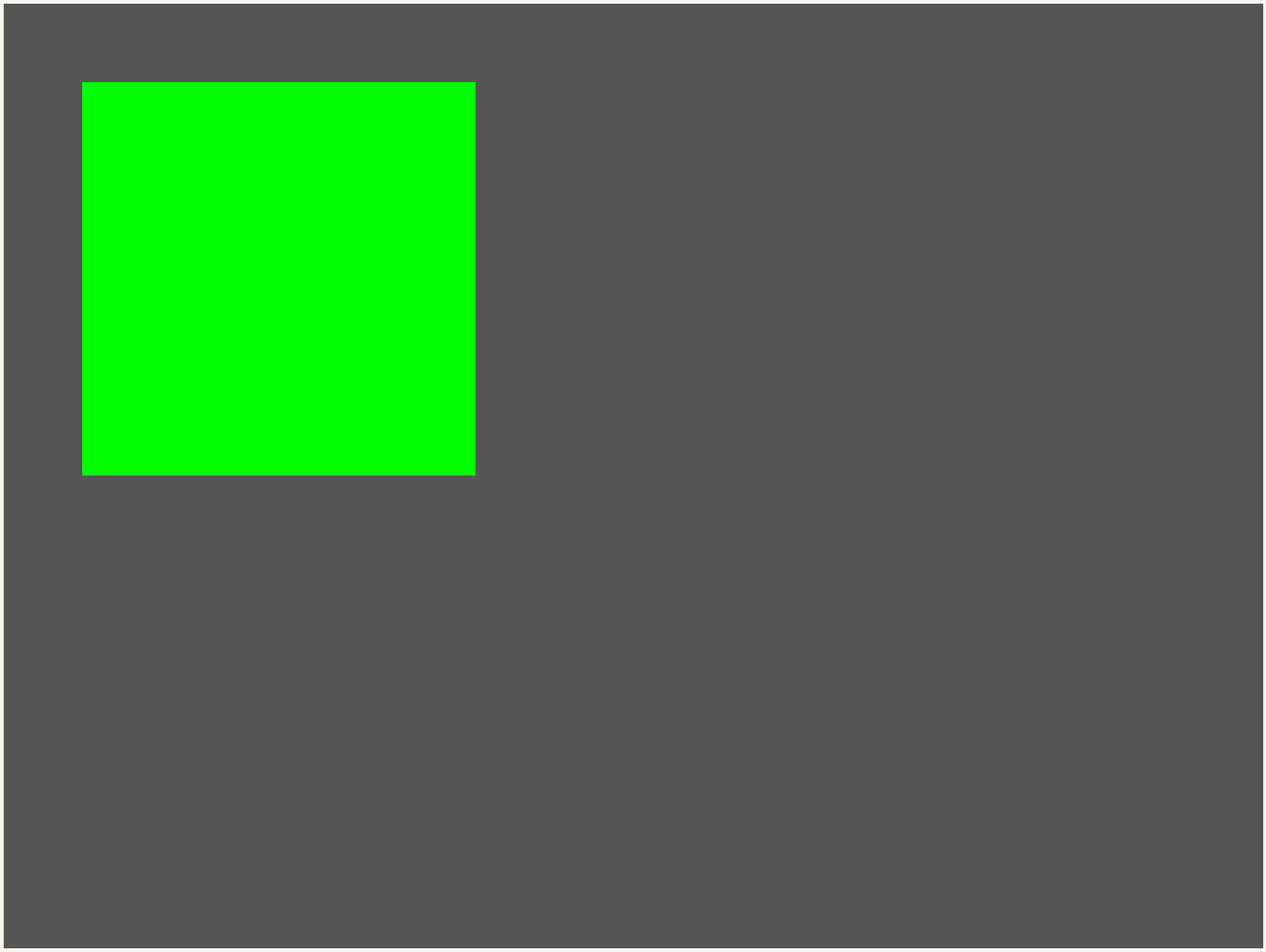


Figure 4: Current Video Page from the paused Disk BASIC guest. Its filled square is on the left.

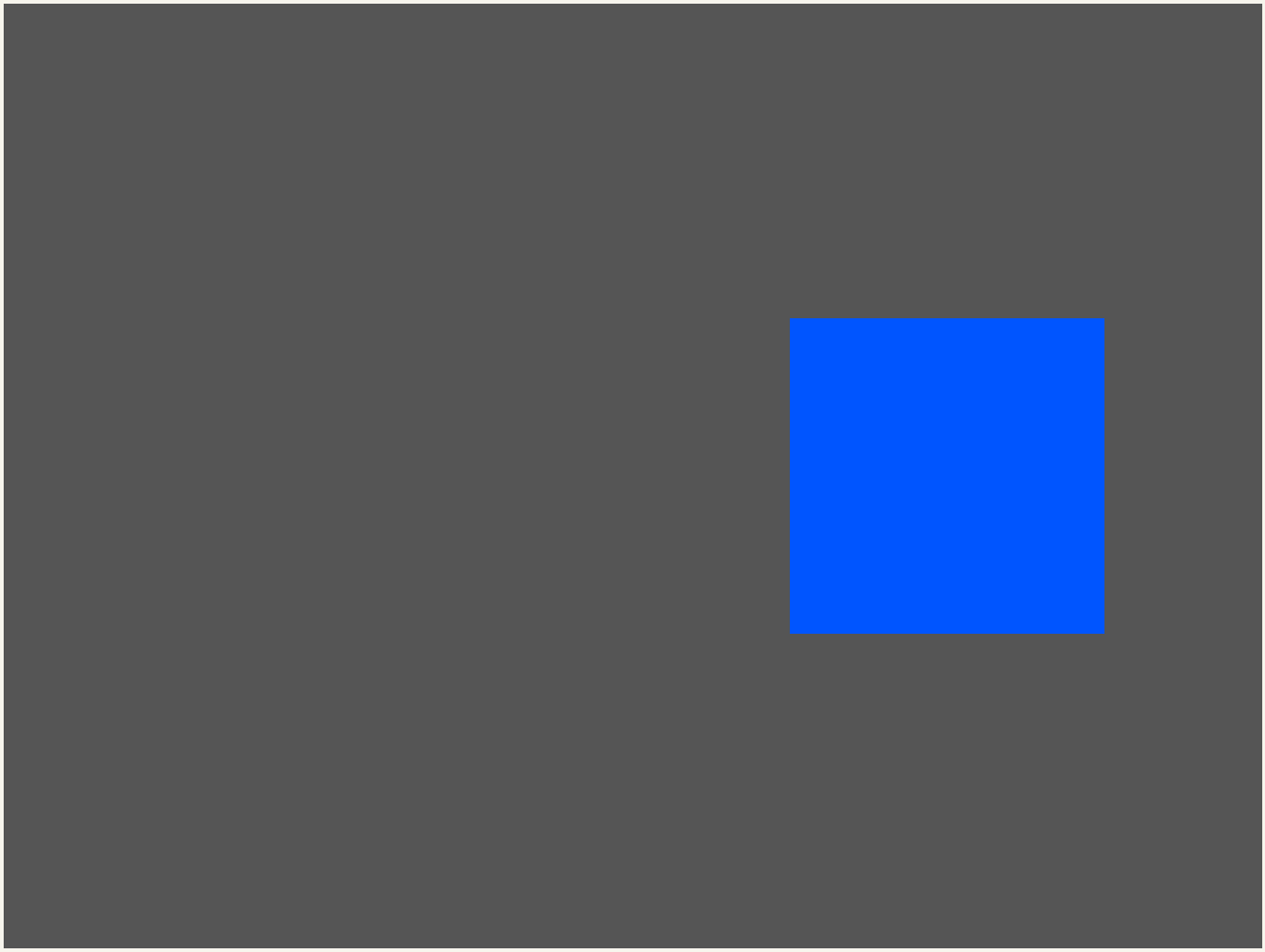


Figure 5: At Address, mapped CPU $1A00, from that same pause. Its filled square is on the right.

PMODE 1 stores 128 × 96 pixels with four two-bit pixels per byte. A row therefore occupies 32 bytes and a picture occupies 3,072 bytes ($0C00). On this setup the first picture corresponds to mapped $0E00 and the second to mapped $1A00. The Current Video Page inspector reported the first one in **physical** RAM at $70E00. Those numbers describe two different address spaces. In BASIC's page numbering this picture occupies two allocation pages; selecting page 3 leaves room for the whole first picture.

Work backward from pixel 81,32

On the hidden page select X **81**, Y **32** in Pixel Inspector. Predict the byte address before using **Show in Memory**. Row 32 begins $0400 bytes after the start ($20 rows × $20 bytes per row). Pixel 81 is in byte 20 ($14) of the row. Add them: $1A00 + $0400 + $14 = **$1E14**. Pixel X=80 occupies the highest pair of bits. X=81 occupies bits 5 and 4, with mask **$30**.

The guest byte is **$AA** (binary 10101010). Masking with $30 leaves $20. Shifting down four places gives the stored pixel code **2**. That code selects a color in the active palette set. BASIC's drawing coordinates and pen numbers sit a level above these packed bits. Book 3's graphics lesson gives the original drawing and a native-window view of the inspector. To prove that you have isolated a single pixel, erase one beacon in hidden RAM and restore it.

Remove the line-50 breakpoint and Continue. Give the CoCo display keyboard focus. Tap and release a letter. The picture you found in RAM is now on the monitor. Another tap brings back the first square, and a third returns

to 40-column text with **BOTH PAGES ARE HOME**. Swapping which page is visible saved redrawing the first square.

Change the paint used by a text cell

Open Earth or Mars in assembly. This time use its CoCo 3 **40-column text** screen. Locate **greet**, set a source stop on `STB BG6`, and Run. Give the CoCo window focus and press **2**. At the stop the source has chosen the Mars string. In the supplied build PC is $3F22, X is $3F8E and B holds $25. Inspect the palette byte at $FFB6 in mapped memory. Its six color bits were $09 on the fresh setup.

Step one instruction. PC advances to $3F25 and the six color bits now read $25. On a repeat with **1**, B holds $0A at the same instruction and the write changes those bits to $0A. This is a change to GIME palette slot 6. The following instructions set BASIC's text attribute so its character-output routine draws the greeting with that background slot. The output routine still receives ordinary character codes. Changing the palette entry can alter how an existing pixel or cell using that slot looks without changing the byte containing its pixel code.

Continue. Read the colored greeting and the BASIC prompt beneath it. The starter clears the text attribute before returning. Its palette write persists; the return restores the text-writing convention, not every palette register. The two guest captures show the rendered choices. For CoCo 3 native high-resolution graphics rather than PMODE or text, Star Chart draws a CoCo 3 HSCREEN with separately defined palette slots.

Move the packet with the HD6309

Chapter 6 of 8

Try the same transfer with a different agreement between the program and the CPU. The portable loop uses B as a counter, A as a sum, and several instructions per byte. The Hitachi processor has a block-transfer instruction and a separate count register. You can see exactly which work it takes over.

Choose both processors deliberately

Save your guest work. In **Machine Settings**, choose the **Hitachi HD6309** processor and apply the restart. Start at the Disk BASIC prompt. In **Assembly Development**, open a copy of dispatch6309.asm and set the assembler's **Target** to HD6309. The Machine setting chooses what executes opcodes. Target chooses what the assembler accepts. As a negative check, choose the MC6809 target and **Assemble**: LDW and TFM are rejected. Return the target to HD6309 before continuing.

```
start       leax    packet,pcr
            leay    received,pcr
            ldw     #size       ; W counts the bytes left to transfer
copy        tfm     x+,y+       ; move from X to Y and increment both
inspect     leax    received,pcr ; stop here to inspect W, X, Y and RAM
next        lda     ,x+
```

W is a sixteen-bit register made from E (high byte) and F (low byte). D is similarly made from A and B. On the 6309, Q is the 32-bit combination D:W. The E **register** and the E **flag in CC** have different jobs. Watch W through E and F if the inspector shows the halves separately.

Cross the transfer in one debugger step

Set a stop on **copy**, choose Run, and look up **packet** and **received** in Symbols. At TFM, X points at the first input byte, Y at the first destination byte, and W is 3. Predict the pointer values after the copy. Step once, then inspect the four bytes at received in **Mapped CPU**.

Value	At copy	At inspect
PC in this build	$3F0C	$3F0F
X	$3F28	$3F2B
Y	$3F2B	$3F2E
W	$0003	$0000
Four destination bytes	00 00 00 00	4F 52 42 00

TFM X+,Y+ copies a byte, advances both pointers and counts down W. In VCC Plus the debugger presents this transfer as a whole instruction. The stop after it is where you compare the buffers. Continue prints ORB and returns. W supplied the count, and the fourth zero came from the original buffer declaration. This version never computes a checksum. Comparing its total run time with the first receiver would mix two different jobs.

Observe native mode, then give BASIC its mode back

The HD6309 can execute the preceding instructions in its startup compatibility mode. Native mode is a *separate* processor setting with effects including interrupt frames and instruction timing. To observe the distinction, use a freshly restarted **HD6309** Disk BASIC guest. Save and open mode.asm. Set a stop on **native** and choose Run. The exercise assumes the reset value MD=$00. It saves CC and masks IRQ and FIRQ during its short mode-changing section. Keep it as supplied and let it finish.

```
start           pshs    cc              ; remember BASIC's interrupt masks
                orcc    #$50            ; mask IRQ and FIRQ for the short experiment
                ldmd    #1              ; select 6309 native mode
native          ldw     #$1234          ; inspect MD, then step and read E and F
                ldmd    #0              ; restore reset mode before calling BASIC
restored        puls    cc              ; restore the caller's condition codes
                rts
```

At **native** the instruction that selects mode 1 has executed. In the Debugger's HD6309 registers inspect MD. Step LDW: E becomes $12 and F becomes $34. Step LDMD #0. At **restored**, MD reads $00 while the data in W still represents $1234. Continue restores the saved condition codes and returns to BASIC. Enter **PRINT 2+3** to check that the guest is listening again.

Changing mode around an unknown interrupt handler would give that code a different environment. In particular, a full native-mode interrupt frame includes E and F. The interrupt experiment starts from a known mode and lets you inspect an actual frame. If you interrupt the mode experiment and lose your place, restart the disposable guest before continuing the book.

Read the return ticket an interrupt leaves

Chapter 7 of 8

A subroutine call saves the address where its caller can resume. An interrupt can arrive while several registers hold half-finished work. How much more does the CPU have to remember? Trigger the event yourself with **SWI** and compare the stack before and after it. You control when the experiment starts, and you can finish it without waiting for an external device.

Follow the vector to a RAM stub

Use a fresh Disk BASIC practice session, with the MC6809 or an HD6309 in its normal emulation mode. Save softirq.asm. Before running it, open the machine debugger and inspect these mapped bytes:

Address	Bytes on the practice machine	What uses them
$FFFA	FE FA	The CPU reads this 16-bit SWI vector and sets PC to $FEFA.
$FEFA	16 02 09	Opcode $16 is LBRA. The displacement is relative to $FEFD, the PC after the instruction. Adding $0209 wraps to $0106.
$0106	00 00 00	Disk BASIC's three-byte RAM stub on this machine. The example saves all three bytes before installing its handler.

If your vector chain differs, leave it alone and use the supplied observations or return to the practice profile. The CPU vector gives an address. The relay at that address is executable code. The stub at the relay destination will become another instruction, an extended JMP to our own handler. These are three distinct things you can identify in memory.

The program saves its caller's CC, D and X, masks IRQ and FIRQ, and installs the temporary JMP. The masks hold off those hardware interrupt classes during the experiment; executing SWI still triggers a software interrupt. Keep the guest in this program through cleanup. If you abandon the experiment before restoration, cold-restart the disposable profile before using it for anything else.

Catch the frame before the handler touches it

Set source breakpoints at **trigger**, **handler** and **returned**, then Run. Read the addresses from the listing after any edit. At **trigger** the supplied 6809 build has PC=$3F26, A=$4F, B=$52 and S=$7F30. The values in A and B spell the first two characters of our station code. Continue to **handler** and look at S **before** executing INC. It now reads $7F24, twelve bytes below its value at SWI.

Switch the machine debugger to **Mapped CPU** memory and look up your handler's S. If you use a different BASIC stack, look at the address in the S register rather than copying $7F24. In increasing address order the test guest showed:

At S + offset	Saved register	Bytes in this run
0	CC	D0
1, 2, 3	A, B, DP	4F 52 00
4–5	X	3F 37
6–7	Y	A0 0E

At S + offset	Saved register	Bytes in this run
8–9	U	AB EE
10–11	PC	3F 27

Multibyte registers have their high byte first. Locate $4F and $52 as a check on your orientation. The final two bytes deserve a closer look. SWI at $3F26 is one byte long; **$3F27** is the instruction immediately after it. The processor stacked a return PC that will let the program carry on. E, bit 7 of the stacked CC, is set. For the 6809 this indicates a full frame.

```
            lda     #$4f            ; known bytes to locate in the stack frame
            ldb     #$52
trigger     swi                     ; CPU stacks state and follows the SWI vector
returned    lda     oldop,pcr        ; breakpoint to compare S after RTI
            sta     hook
            ldx     oldaddr,pcr
            stx     hook+1
restored    puls    cc,d,x,pc        ; restore flags, D and X, return to BASIC

handler     inc     visits,pcr      ; memory survives the return
            rti                     ; reload the stacked context, including PC
```

Let RTI rebuild the context

Continue to **returned**. The handler increments a byte named **visits** and executes RTI. Inspect **visits** using its listing address ($3F3F in the supplied build). It is now 1. Compare registers with the trigger stop: A=$4F, B=$52 and S=$7F30 are back, and PC is the saved $3F27. The memory side effect survives the return. The stack frame has been consumed; old bytes might still be visible below S until some later call reuses that memory.

Set a final source stop at **restored**, then Continue. Read $0106–$0108 again. The original three bytes are back. Continue once more and enter `PRINT 2+3` at the BASIC prompt. An RTS here in place of the handler's RTI would take the top two bytes as an address and leave the rest of the saved context on the stack. You can explain that failure from the table without trying it.

Apply the observation to other interrupts

IRQ and **FIRQ** are signals a device can assert. A real interrupt handler also has to determine which device needs service and acknowledge it according to that device's rules. SWI gives us a repeatable stack experiment without those timing or device interactions.

For the MC6809, IRQ normally saves the full register frame. FIRQ normally saves just CC and PC, with the stacked E bit clear. Code in a fast handler has to preserve other registers it uses. RTI reads the stacked state to determine what to pull. The HD6309 can run in emulation or native mode; full-frame stacking in native mode also includes E and F, and the 6309 provides a mode control affecting FIRQ stacking. Recheck the mode before applying any fixed frame diagram. The twelve bytes shown here were measured on both supported processors in the startup configuration. For full-frame layouts and mode controls see Darren Atkinson, *Motorola 6809 and Hitachi 6309 Programmer's Reference* (2009), parts I–II.

Deliver the receiver

Chapter 8 of 8

Imagine handing your program to someone who has only a CoCo and Disk BASIC. They cannot open your source files and click **Run**. They can load a binary from a floppy. In this chapter you build a single executable from two source files, catch the call where they meet, and give your reader two ways to run the finished disk.

Open the whole project

Save the Dispatch project, expand the ZIP in a working folder, and open **Develop → Assembly Development...**. Choose **File → Open Project Folder...** inside that window and select the folder containing these four files. Start with the **Disk BASIC** profile at its prompt and an **MC6809** assembler target. Use a personal copy of any mounted media.

File	Its job
`main.asm`	Supplies the entry point, packet, expected checksum, destination buffer and screen output.
`receive.asm`	Copies bytes and computes their checksum. The caller supplies X, Y and B.
`vccplus-assembly.json`	Identifies the entry source, additional module and linker script for VCC Plus.
`layout.link`	Places the combined code at hexadecimal 3F00 and selects the symbol `start` as the entry.

The two assembly files each name a section called `code`. Read the first few lines of each. The display module **imports** `receive`, promising to find its definition in another object. The routine **exports** that name, allowing the linker to resolve it. The files can agree on an address and still disagree about what the routine expects. Find the instructions in the caller that satisfy all three inputs in the comment above `receive`.

Read the address that the linker chose

Choose **Assemble**. Expand **Build details** below the source. **Log** shows two assembler invocations followed by the linker. **Listing** describes the assembly of each object. **Symbols** shows where the linker actually placed those sections and symbols. A zero offset at the start of the second object's section can therefore be a perfectly good listing value.

Symbol or instruction	Initial build	What to look for
`start`	$3F00	First byte of the entry module. The linker reports its section length as $0040.
`jsr receive`	$3F0A	Bytes BD 3F 40. The final two bytes contain the absolute address supplied by the linker.
`inspect`	$3F0D	Next instruction after the call. JSR puts this return address on the stack.
`receive`	$3F40	First instruction supplied by the second object. Its section takes $000F bytes.

Select `receive.asm` in the project and place the caret on `receive clra`. Choose **Set Stop**. Set another stop on `inspect cmpa expected,pcr` in `main.asm`. **Run** and confirm the load if prompted. The first stop is in the other module, before CLRA executes. B holds 3. If you examine the two bytes at S in the debugger's mapped memory, they are 3F 0D. The subroutine has a route back into its caller even though the two files were assembled separately.

Continue to `inspect`. A holds $E3 and the four bytes at `received` ($3F36 in this build) are 4F 52 42 00. Continue again. The CoCo prints `ORB` and returns to BASIC. For a deliberate failure, edit the quoted `ORB` in `packet` to `ORD`, leaving `expected` alone. Run the new build. A is now $E5 at the comparison and the CoCo prints `CHECK`. Restore the letter B before you make the disk for your reader.

Move the code and keep the source stop

Try this on a separate copy of the entire project. In `layout.link` change the first line from `section code load 3f00` to `section code load 4100`. Leave the second line, `entry start`, alone. Save, then **Assemble**. Before looking at **Symbols**, predict the second module's new address from the length of the first module.

The source stops are still the CLRA in `receive.asm` and the comparison in `main.asm`. Set them if you opened a fresh copy. On **Run**, the same CLRA now stops at $4140. The stack contains 41 0D. Continue to $410D and check A before finishing. The linker changed the absolute target of JSR to $4140. The packet and its PC-relative load moved together, so their displacement stayed the same. Neither a source stop nor a symbolic operand required you to type an address into the instructions.

The two tested placements are for this short program in the lab's fresh Disk BASIC session. Choosing a new load address also means finding space the guest can spare. A writable address can still contain a BASIC program or another routine you need.

Hand someone a disk

Return to the unmodified $3F00 project. The **Export** menu offers the DECB binary by itself and a floppy image containing that binary. A loose binary is useful for emulator quickload. Make two new images now to test what someone can do at a guest prompt.

1. Choose **Export → Export Disk Image…**. Enter `DISPATCH` as the **Disk BASIC filename**, leave **Install DOS bootloader** clear, choose **Choose Location…**, and save as `Dispatch-plain.dsk`. The eight-letter guest name and the Mac filename serve different purposes.
2. Open the floppy controls. Preserve anything you need from the current FD0 disk and mount your new image in FD0. At the fresh **Disk BASIC** prompt, type `DIR` and press Return. Look for `DISPATCH BIN`. Enter `LOADM"DISPATCH":EXEC`. LOADM puts the records into their assigned addresses and records the entry point. EXEC transfers control there. The greeting for an intact packet is `ORB`.
3. Export again with the same guest name. This time select **Install DOS bootloader** and save the new image as `Dispatch-boot.dsk`. Mount this image in FD0. Type `DOS` on its own. Watch the line underneath it as the loader supplies the full Disk BASIC command. You can still enter `LOADM"DISPATCH":EXEC` yourself on this disk.

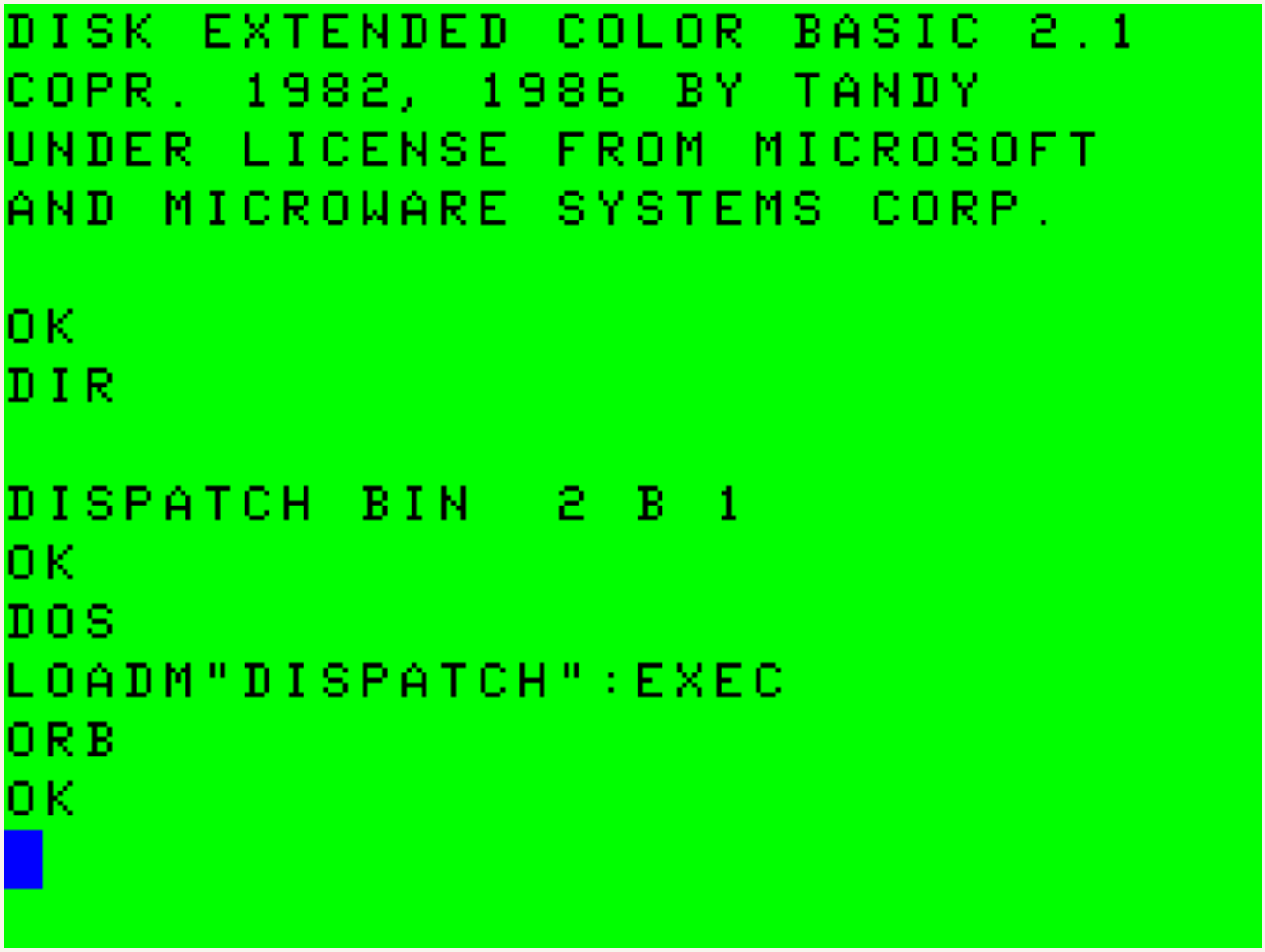


Figure 6: Disk BASIC lists DISPATCH BIN. After DOS, the bootloader supplies LOADM"DISPATCH":EXEC; the receiver prints ORB and BASIC displays OK.

The drive starts the same machine-language program either way. The source stops from the Assembly workspace belong to its direct debugging session. For the disk test, judge success at the CoCo screen. After the final `OK`, enter `PRINT 2+3`. Seeing 5 proves that the program returned with BASIC able to accept and execute another command.

A handoff you can repeat

Give a developer the four project files together. Give someone who wants to run the program the DSK, the required **Disk BASIC** profile, **FD0**, the name `DISPATCH`, and the exact launch command. Identify whether that particular disk has the optional bootloader. Include the expected output and how control returns.

If the project assembles but does not stop where expected, compare its current **Symbols** map with the source file and instruction you selected. If `DIR` lists the wrong contents, check which image is in FD0 before changing any source. If the disk outputs `CHECK`, it has already loaded and started the receiver. Look at the packet and checksum of the project you exported. Those observations separate a build problem, a mounting problem and a working program reporting damaged input.

Instruction directory for the 6809 and 6309

Appendix A

You have stopped on `BNE`. Does it test the accumulator? It reads one bit in CC. Look up BNE, then find the instruction that last wrote Z. In the receiver's counting loop, that writer is DECB. At the packet check it is CMPA. Identical branch instructions answer different questions because they inherit different evidence.

Pick an operation

What do you want the processor to do?	Go to
Get a value, store a value, or calculate where it lives	Move a value or compute an address
Add, subtract, multiply, divide, or extend a signed value	Calculate with integers
Test, clear, set, invert, or combine bits	Test and change bits
Move bits and use carry between operations	Shift and rotate
Compare numbers or select the next instruction	Compare and choose a path
Call a routine, protect registers, or return to a caller	Call, save and return
Interpret an interrupt, a processor mode or a wait	Interrupts and processor control
Copy a sequence of bytes with the HD6309	Transfer a block

Each family begins with an alphabetical list. Alternative spellings lead to the same entry. Search this book from its contents page to find a particular mnemonic. In the Assembly editor, put the caret on an instruction and choose **Look Up in Assembly Quick Reference** for the corresponding native topic.

Read a card from the listing

An instruction card identifies the processor, operands, operation, flags and a printed source. The operand line gives patterns. Pick one pattern and supply your register, constant, expression or label. A blank operand field in the listing is normal for an inherent instruction such as `RTS`. The register and addressing card explains the punctuation.

MC6809 and HD6309 means the hardware operation exists on both. **HD6309 only** means the guest needs the Hitachi processor for that hardware instruction. These entries can execute in its startup emulation mode as well as native mode. Selecting the 6309 assembler target admits the encodings. Selecting the 6309 processor supplies the hardware. The program's mode register controls execution mode and affects interrupt behavior and timing. See HD6309 modes before changing it.

Read **Opcode bytes** from left to right. For an indexed LDA the opcode is `A6`. A postbyte follows to say how to use the index register. For a long conditional branch, `10 26` selects LBNE, and the next two bytes hold the displacement. The directory gives just the opcode for each encoding. The listing in the Assembly workspace shows the complete instruction, operands included.

Flags describe the documented processor. When a flag is undefined, choose another test. If you need an earlier comparison, branch before issuing an instruction that rewrites its evidence. The signed comparison experiment shows how to read N and V together. The DIVQ entry flags differences in VCC Plus's overflow and carry results. For a trace of TFM, debugger Step completes a transfer, whereas Continue can encounter an instruction breakpoint again between bytes.

There are 215 hardware instruction entries and 234 hardware spellings in this edition. An alias can have its own operand convention. For example, BHS is another spelling of BCC; the TFM aliases take bare register names and encode their direction in the mnemonic. For multiple-instruction expansions, data declarations, and link directives, use the LWASM index.

Move a value or compute an address

In the lab, follow the bytes of the receiver.

`ABX` · `EXG` · `LDA` · `LDB` · `LDD` · `LDE` · `LDF` · `LDQ` · `LDS` · `LDU` · `LDW` · `LDX` · `LDY` · `LEAS` · `LEAU` · `LEAX` · `LEAY` · `STA` · `STB` · `STD` · `STE` · `STF` · `STQ` · `STS` · `STU` · `STW` · `STX` · `STY` · `TFR`

ABX

MC6809 and HD6309

```
No operand
```

Opcode bytes. `3A`.
Add the unsigned byte in B to X.
Flags. CC is preserved.
Reading: Atkinson (2009), pp. 4 · Motorola (1981), Appendix A, ABX · Astle, LWTools 4.25, ch. 3.

EXG

MC6809 and HD6309

```
source register,destination register
```

Opcode bytes. `1E`.
Exchange the contents of the two registers.
Flags. Writing CC replaces its flags.
Use equal-width registers on the MC6809. The HD6309 has additional registers and defined mixed-width behavior. Writing PC transfers control.
Reading: Atkinson (2009), pp. 65–66 · Motorola (1981), Appendix A, EXG · Astle, LWTools 4.25, ch. 3.

LDA

MC6809 and HD6309

```
#literal   ·   <direct   ·   offset,X   ·   >address
```

Opcode bytes. direct `96` · indexed `A6` · extended `B6` · immediate `86`.
Copy the operand into A. Use # for a literal or read one byte from the addressed memory.
Flags. N reflects the sign bit. Z marks a zero result. V clears. C is preserved.
LDA #$20 places $20 in A. LDA $0400 reads the byte at $0400. LDA ,X+ reads at X, then advances X by one. The remaining CC bits keep their values. Immediate LDA takes 2 cycles. Extended LDA takes 5 on MC6809 or HD6309 emulation mode and 4 in HD6309 native mode. Indexed timing depends on the postbyte.
Reading: Atkinson (2009), pp. 90 · Motorola (1981), Appendix A, LDA · Astle, LWTools 4.25, ch. 3.

LDB

MC6809 and HD6309

```
#literal   ·   <direct   ·   offset,X   ·   >address
```

Opcode bytes. direct `D6` · indexed `E6` · extended `F6` · immediate `C6`.
Copy the operand into B. Use # for a literal or read one byte from the addressed memory.
Flags. N reflects the sign bit. Z marks a zero result. V clears. C is preserved.
Reading: Atkinson (2009), pp. 90 · Motorola (1981), Appendix A, LDB · Astle, LWTools 4.25, ch. 3.

LDD

MC6809 and HD6309

```
#literal   ·   <direct   ·   offset,X   ·   >address
```

Opcode bytes. direct `DC` · indexed `EC` · extended `FC` · immediate `CC`.
Copy the operand into D. Use # for a literal or read two bytes from the addressed memory.
Flags. N reflects the sign bit. Z marks a zero result. V clears. C is preserved.
Reading: Atkinson (2009), pp. 91 · Motorola (1981), Appendix A, LDD · Astle, LWTools 4.25, ch. 3.

LDE

HD6309 only

```
#literal   ·   <direct   ·   offset,X   ·   >address
```

Opcode bytes. direct `11 96` · indexed `11 A6` · extended `11 B6` · immediate `11 86`.
Copy the operand into E. Use # for a literal or read one byte from the addressed memory.
Flags. N reflects the sign bit. Z marks a zero result. V clears. C is preserved.
Reading: Atkinson (2009), pp. 90 · Astle, LWTools 4.25, ch. 3.

LDF

HD6309 only

```
#literal   ·   <direct   ·   offset,X   ·   >address
```

Opcode bytes. direct `11 D6` · indexed `11 E6` · extended `11 F6` · immediate `11 C6`.
Copy the operand into F. Use # for a literal or read one byte from the addressed memory.
Flags. N reflects the sign bit. Z marks a zero result. V clears. C is preserved.
Reading: Atkinson (2009), pp. 90 · Astle, LWTools 4.25, ch. 3.

LDQ

HD6309 only

```
#literal   ·   <direct   ·   offset,X   ·   >address
```

Opcode bytes. direct `10 DC` · indexed `10 EC` · extended `10 FC` · immediate `CD`.
Copy the operand into Q. Use # for a literal or read four bytes from the addressed memory.
Flags. N reflects the sign bit. Z marks a zero result. V clears. C is preserved.
Reading: Atkinson (2009), pp. 94 · Astle, LWTools 4.25, ch. 3.

LDS

MC6809 and HD6309

```
#literal   ·   <direct   ·   offset,X   ·   >address
```

Opcode bytes. direct `10 DE` · indexed `10 EE` · extended `10 FE` · immediate `10 CE`.

Copy the operand into S. Use # for a literal or read two bytes from the addressed memory.
Flags. N reflects the sign bit. Z marks a zero result. V clears. C is preserved.
Reading: Atkinson (2009), pp. 91 · Motorola (1981), Appendix A, LDS · Astle, LWTools 4.25, ch. 3.

LDU

MC6809 and HD6309

```
#literal   ·   <direct   ·   offset,X   ·   >address
```

Opcode bytes. direct `DE` · indexed `EE` · extended `FE` · immediate `CE`.
Copy the operand into U. Use # for a literal or read two bytes from the addressed memory.
Flags. N reflects the sign bit. Z marks a zero result. V clears. C is preserved.
Reading: Atkinson (2009), pp. 91 · Motorola (1981), Appendix A, LDU · Astle, LWTools 4.25, ch. 3.

LDW

HD6309 only

```
#literal   ·   <direct   ·   offset,X   ·   >address
```

Opcode bytes. direct `10 96` · indexed `10 A6` · extended `10 B6` · immediate `10 86`.
Copy the operand into W. Use # for a literal or read two bytes from the addressed memory.
Flags. N reflects the sign bit. Z marks a zero result. V clears. C is preserved.
LDW #$1234 encodes as 10 86 12 34. E becomes $12 and F becomes $34 in either HD6309 execution mode.
Reading: Atkinson (2009), pp. 91 · Astle, LWTools 4.25, ch. 3.

LDX

MC6809 and HD6309

```
#literal   ·   <direct   ·   offset,X   ·   >address
```

Opcode bytes. direct `9E` · indexed `AE` · extended `BE` · immediate `8E`.
Copy the operand into X. Use # for a literal or read two bytes from the addressed memory.
Flags. N reflects the sign bit. Z marks a zero result. V clears. C is preserved.
Reading: Atkinson (2009), pp. 91 · Motorola (1981), Appendix A, LDX · Astle, LWTools 4.25, ch. 3.

LDY

MC6809 and HD6309

```
#literal   ·   <direct   ·   offset,X   ·   >address
```

Opcode bytes. direct `10 9E` · indexed `10 AE` · extended `10 BE` · immediate `10 8E`.
Copy the operand into Y. Use # for a literal or read two bytes from the addressed memory.
Flags. N reflects the sign bit. Z marks a zero result. V clears. C is preserved.
Reading: Atkinson (2009), pp. 91 · Motorola (1981), Appendix A, LDY · Astle, LWTools 4.25, ch. 3.

LEAS

MC6809 and HD6309

```
offset,index
```

Opcode bytes. 32.

Calculate an indexed address and place it in S. Indirect addressing reads a pointer before producing the address.
Flags. CC is preserved.
Reading: Atkinson (2009), pp. 95 · Motorola (1981), Appendix A, LEAS · Astle, LWTools 4.25, ch. 3.

LEAU

MC6809 and HD6309

```
offset,index
```

Opcode bytes. `33`.
Calculate an indexed address and place it in U. Indirect addressing reads a pointer before producing the address.
Flags. CC is preserved.
Reading: Atkinson (2009), pp. 95 · Motorola (1981), Appendix A, LEAU · Astle, LWTools 4.25, ch. 3.

LEAX

MC6809 and HD6309

```
offset,index
```

Opcode bytes. `30`.
Calculate an indexed address and place it in X. Indirect addressing reads a pointer before producing the address.
Flags. Only Z changes, to indicate a zero address.
Reading: Atkinson (2009), pp. 95 · Motorola (1981), Appendix A, LEAX · Astle, LWTools 4.25, ch. 3.

LEAY

MC6809 and HD6309

```
offset,index
```

Opcode bytes. `31`.
Calculate an indexed address and place it in Y. Indirect addressing reads a pointer before producing the address.
Flags. Only Z changes, to indicate a zero address.
Reading: Atkinson (2009), pp. 95 · Motorola (1981), Appendix A, LEAY · Astle, LWTools 4.25, ch. 3.

STA

MC6809 and HD6309

```
<direct   ·   offset,X   ·   >address
```

Opcode bytes. direct `97` · indexed `A7` · extended `B7`.
Copy A to the destination address. Multi-byte stores write the high byte first.
Flags. N reflects the sign bit. Z marks a zero result. V clears. C is preserved.
Reading: Atkinson (2009), pp. 127 · Motorola (1981), Appendix A, STA · Astle, LWTools 4.25, ch. 3.

STB

MC6809 and HD6309

```
<direct   ·   offset,X   ·   >address
```

Opcode bytes. direct `D7` · indexed `E7` · extended `F7`.
Copy B to the destination address. Multi-byte stores write the high byte first.
Flags. N reflects the sign bit. Z marks a zero result. V clears. C is preserved.
Reading: Atkinson (2009), pp. 127 · Motorola (1981), Appendix A, STB · Astle, LWTools 4.25, ch. 3.

STD

MC6809 and HD6309

```
<direct   ·   offset,X   ·   >address
```

Opcode bytes. direct `DD` · indexed `ED` · extended `FD`.
Copy D to the destination address. Multi-byte stores write the high byte first.
Flags. N reflects the sign bit. Z marks a zero result. V clears. C is preserved.
Reading: Atkinson (2009), pp. 128 · Motorola (1981), Appendix A, STD · Astle, LWTools 4.25, ch. 3.

STE

HD6309 only

```
<direct   ·   offset,X   ·   >address
```

Opcode bytes. direct `11 97` · indexed `11 A7` · extended `11 B7`.
Copy E to the destination address. Multi-byte stores write the high byte first.
Flags. N reflects the sign bit. Z marks a zero result. V clears. C is preserved.
Reading: Atkinson (2009), pp. 127 · Astle, LWTools 4.25, ch. 3.

STF

HD6309 only

```
<direct   ·   offset,X   ·   >address
```

Opcode bytes. direct `11 D7` · indexed `11 E7` · extended `11 F7`.
Copy F to the destination address. Multi-byte stores write the high byte first.
Flags. N reflects the sign bit. Z marks a zero result. V clears. C is preserved.
Reading: Atkinson (2009), pp. 127 · Astle, LWTools 4.25, ch. 3.

STQ

HD6309 only

```
<direct   ·   offset,X   ·   >address
```

Opcode bytes. direct `10 DD` · indexed `10 ED` · extended `10 FD`.
Copy Q to the destination address. Multi-byte stores write the high byte first.
Flags. N reflects the sign bit. Z marks a zero result. V clears. C is preserved.
Reading: Atkinson (2009), pp. 130 · Astle, LWTools 4.25, ch. 3.

STS

MC6809 and HD6309

```
<direct   ·   offset,X   ·   >address
```

Opcode bytes. direct `10 DF` · indexed `10 EF` · extended `10 FF`.
Copy S to the destination address. Multi-byte stores write the high byte first.
Flags. N reflects the sign bit. Z marks a zero result. V clears. C is preserved.
Reading: Atkinson (2009), pp. 128 · Motorola (1981), Appendix A, STS · Astle, LWTools 4.25, ch. 3.

STU

MC6809 and HD6309

```
<direct   ·   offset,X   ·   >address
```

Opcode bytes. direct `DF` · indexed `EF` · extended `FF`.
Copy U to the destination address. Multi-byte stores write the high byte first.
Flags. N reflects the sign bit. Z marks a zero result. V clears. C is preserved.
Reading: Atkinson (2009), pp. 128 · Motorola (1981), Appendix A, STU · Astle, LWTools 4.25, ch. 3.

STW

HD6309 only

```
<direct   ·   offset,X   ·   >address
```

Opcode bytes. direct `10 97` · indexed `10 A7` · extended `10 B7`.
Copy W to the destination address. Multi-byte stores write the high byte first.
Flags. N reflects the sign bit. Z marks a zero result. V clears. C is preserved.
Reading: Atkinson (2009), pp. 128 · Astle, LWTools 4.25, ch. 3.

STX

MC6809 and HD6309

```
<direct   ·   offset,X   ·   >address
```

Opcode bytes. direct `9F` · indexed `AF` · extended `BF`.
Copy X to the destination address. Multi-byte stores write the high byte first.
Flags. N reflects the sign bit. Z marks a zero result. V clears. C is preserved.
Reading: Atkinson (2009), pp. 128 · Motorola (1981), Appendix A, STX · Astle, LWTools 4.25, ch. 3.

STY

MC6809 and HD6309

```
<direct   ·   offset,X   ·   >address
```

Opcode bytes. direct `10 9F` · indexed `10 AF` · extended `10 BF`.
Copy Y to the destination address. Multi-byte stores write the high byte first.
Flags. N reflects the sign bit. Z marks a zero result. V clears. C is preserved.
Reading: Atkinson (2009), pp. 128 · Motorola (1981), Appendix A, STY · Astle, LWTools 4.25, ch. 3.

TFR

MC6809 and HD6309

```
source register,destination register
```

Opcode bytes. `1F`.
Copy the source register to the destination.
Flags. Writing CC replaces its flags.
Use equal-width registers on the MC6809. The HD6309 has additional registers and defined mixed-width behavior. Writing PC transfers control.
Reading: Atkinson (2009), pp. 137–138 · Motorola (1981), Appendix A, TFR · Astle, LWTools 4.25, ch. 3.

Calculate with integers

In the lab, observe the checksum and its flags.
`ADCA` · `ADCB` · `ADCD` · `ADCR` · `ADDA` · `ADDB` · `ADDD` · `ADDE` · `ADDF` · `ADDR` · `ADDW` · `DAA` · `DEC` · `DECA` · `DECB` · `DECD` · `DECE` · `DECF` · `DECW` · `DIVD` · `DIVQ` · `INC` · `INCA` · `INCB` · `INCD` · `INCE` · `INCF` · `INCW` · `MUL` · `MULD` · `NEG` · `NEGA` · `NEGB` · `NEGD` · `SBCA` · `SBCB` · `SBCD` · `SBCR` · `SEX` · `SEXW` · `SUBA` · `SUBB` · `SUBD` · `SUBE` · `SUBF` · `SUBR` · `SUBW`

ADCA

MC6809 and HD6309

```
#literal   ·   <direct   ·   offset,X   ·   >address
```

Opcode bytes. direct `99` · indexed `A9` · extended `B9` · immediate `89`.
Add the operand and incoming carry to A. Keep the low 8 bits in A.
Flags. N reflects the sign bit. Z marks a zero result. V marks signed overflow. C marks unsigned carry. H records carry from bit 3.
Reading: Atkinson (2009), pp. 5 · Motorola (1981), Appendix A, ADCA · Astle, LWTools 4.25, ch. 3.

ADCB

MC6809 and HD6309

```
#literal   ·   <direct   ·   offset,X   ·   >address
```

Opcode bytes. direct `D9` · indexed `E9` · extended `F9` · immediate `C9`.
Add the operand and incoming carry to B. Keep the low 8 bits in B.
Flags. N reflects the sign bit. Z marks a zero result. V marks signed overflow. C marks unsigned carry. H records carry from bit 3.
Reading: Atkinson (2009), pp. 5 · Motorola (1981), Appendix A, ADCB · Astle, LWTools 4.25, ch. 3.

ADCD

HD6309 only

```
#literal   ·   <direct   ·   offset,X   ·   >address
```

Opcode bytes. direct `10 99` · indexed `10 A9` · extended `10 B9` · immediate `10 89`.
Add the operand and incoming carry to D. Keep the low 16 bits in D.
Flags. N reflects the sign bit. Z marks a zero result. V marks signed overflow. C marks unsigned carry. H is preserved.
Reading: Atkinson (2009), pp. 6 · Astle, LWTools 4.25, ch. 3.

ADCR

HD6309 only

```
source register,destination register
```

Opcode bytes. `10 31`.
Add source and carry to destination.
Flags. N reflects the sign bit. Z marks a zero result. V marks signed overflow. C marks unsigned carry.
The destination selects the width. A word source supplies its low byte to a byte destination. For a word destination A/B supply D, E/F supply W, CC is zero-extended and DP supplies the high byte. PC is undefined in this family. See Atkinson, p. 143.
Reading: Atkinson (2009), pp. 7 · Astle, LWTools 4.25, ch. 3.

ADDA

MC6809 and HD6309

```
#literal   ·   <direct   ·   offset,X   ·   >address
```

Opcode bytes. direct `9B` · indexed `AB` · extended `BB` · immediate `8B`.
Add the operand to A. Keep the low 8 bits in A.
Flags. N reflects the sign bit. Z marks a zero result. V marks signed overflow. C marks unsigned carry. H records carry from bit 3.
Reading: Atkinson (2009), pp. 8 · Motorola (1981), Appendix A, ADDA · Astle, LWTools 4.25, ch. 3.

ADDB

MC6809 and HD6309

```
#literal   ·   <direct   ·   offset,X   ·   >address
```

Opcode bytes. direct `DB` · indexed `EB` · extended `FB` · immediate `CB`.
Add the operand to B. Keep the low 8 bits in B.
Flags. N reflects the sign bit. Z marks a zero result. V marks signed overflow. C marks unsigned carry. H records carry from bit 3.
Reading: Atkinson (2009), pp. 8 · Motorola (1981), Appendix A, ADDB · Astle, LWTools 4.25, ch. 3.

ADDD

MC6809 and HD6309

```
#literal   ·   <direct   ·   offset,X   ·   >address
```

Opcode bytes. direct `D3` · indexed `E3` · extended `F3` · immediate `C3`.
Add the operand to D. Keep the low 16 bits in D.
Flags. N reflects the sign bit. Z marks a zero result. V marks signed overflow. C marks unsigned carry. H is preserved.
Reading: Atkinson (2009), pp. 9 · Motorola (1981), Appendix A, ADDD · Astle, LWTools 4.25, ch. 3.

ADDE

HD6309 only

```
#literal   ·   <direct   ·   offset,X   ·   >address
```

Opcode bytes. direct `11 9B` · indexed `11 AB` · extended `11 BB` · immediate `11 8B`.
Add the operand to E. Keep the low 8 bits in E.
Flags. N reflects the sign bit. Z marks a zero result. V marks signed overflow. C marks unsigned carry. H records carry from bit 3.
Reading: Atkinson (2009), pp. 8 · Astle, LWTools 4.25, ch. 3.

ADDF

HD6309 only

```
#literal   ·   <direct   ·   offset,X   ·   >address
```

Opcode bytes. direct `11 DB` · indexed `11 EB` · extended `11 FB` · immediate `11 CB`.
Add the operand to F. Keep the low 8 bits in F.
Flags. N reflects the sign bit. Z marks a zero result. V marks signed overflow. C marks unsigned carry. H records carry from bit 3.
Reading: Atkinson (2009), pp. 8 · Astle, LWTools 4.25, ch. 3.

ADDR

HD6309 only

```
source register,destination register
```

Opcode bytes. `10 30`.
Add source to destination.
Flags. N reflects the sign bit. Z marks a zero result. V marks signed overflow. C marks unsigned carry.
The destination selects the width. A word source supplies its low byte to a byte destination. For a word destination A/B supply D, E/F supply W, CC is zero-extended and DP supplies the high byte. PC is undefined in this family. See Atkinson, p. 143.
Reading: Atkinson (2009), pp. 10 · Astle, LWTools 4.25, ch. 3.

ADDW

HD6309 only

```
#literal   ·   <direct   ·   offset,X   ·   >address
```

Opcode bytes. direct `10 9B` · indexed `10 AB` · extended `10 BB` · immediate `10 8B`.
Add the operand to W. Keep the low 16 bits in W.
Flags. N reflects the sign bit. Z marks a zero result. V marks signed overflow. C marks unsigned carry. H is preserved.
Reading: Atkinson (2009), pp. 9 · Astle, LWTools 4.25, ch. 3.

DAA

MC6809 and HD6309

```
No operand
```

Opcode bytes. `19`.
Adjust A after adding packed BCD bytes, using H and C from the addition.
Flags. N reflects the sign bit. Z marks a zero result. C reports decimal carry. V is undefined.
Reading: Atkinson (2009), pp. 56 · Motorola (1981), Appendix A, DAA · Astle, LWTools 4.25, ch. 3.

DEC

MC6809 and HD6309

```
<direct   ·   offset,X   ·   >address
```

Opcode bytes. direct `0A` · indexed `6A` · extended `7A`.
Operate on a memory byte. Subtract one.
Flags. N reflects the sign bit. Z marks a zero result. V sets on crossing from minimum negative to maximum positive. C is preserved.
Reading: Atkinson (2009), pp. 57–58 · Motorola (1981), Appendix A, DEC · Astle, LWTools 4.25, ch. 3.

DECA

MC6809 and HD6309

```
No operand
```

Opcode bytes. `4A`.
Operate on register A. Subtract one.
Flags. N reflects the sign bit. Z marks a zero result. V sets on crossing from minimum negative to maximum positive. C is preserved.
Reading: Atkinson (2009), pp. 57–58 · Motorola (1981), Appendix A, DECA · Astle, LWTools 4.25, ch. 3.

DECB

MC6809 and HD6309

```
No operand
```

Opcode bytes. `5A`.
Operate on register B. Subtract one.
Flags. N reflects the sign bit. Z marks a zero result. V sets on crossing from minimum negative to maximum positive. C is preserved.
Reading: Atkinson (2009), pp. 57–58 · Motorola (1981), Appendix A, DECB · Astle, LWTools 4.25, ch. 3.

DECD

HD6309 only

```
No operand
```

Opcode bytes. `10 4A`.
Operate on register D. Subtract one.
Flags. N reflects the sign bit. Z marks a zero result. V sets on crossing from minimum negative to maximum positive. C is preserved.
Reading: Atkinson (2009), pp. 57–58 · Astle, LWTools 4.25, ch. 3.

DECE

HD6309 only

```
No operand
```

Opcode bytes. `11 4A`.
Operate on register E. Subtract one.

Flags. N reflects the sign bit. Z marks a zero result. V sets on crossing from minimum negative to maximum positive. C is preserved.
Reading: Atkinson (2009), pp. 57–58 · Astle, LWTools 4.25, ch. 3.

DECF

HD6309 only

```
No operand
```

Opcode bytes. `11 5A`.
Operate on register F. Subtract one.
Flags. N reflects the sign bit. Z marks a zero result. V sets on crossing from minimum negative to maximum positive. C is preserved.
Reading: Atkinson (2009), pp. 57–58 · Astle, LWTools 4.25, ch. 3.

DECW

HD6309 only

```
No operand
```

Opcode bytes. `10 5A`.
Operate on register W. Subtract one.
Flags. N reflects the sign bit. Z marks a zero result. V sets on crossing from minimum negative to maximum positive. C is preserved.
Reading: Atkinson (2009), pp. 57–58 · Astle, LWTools 4.25, ch. 3.

DIVD

HD6309 only

```
#literal   ·   <direct   ·   offset,X   ·   >address
```

Opcode bytes. direct `11 9D` · indexed `11 AD` · extended `11 BD` · immediate `11 8D`.
Divide signed D by a signed byte. The quotient goes in B and the remainder in A. A zero divisor traps.
Flags. For an in-range quotient, N and Z describe it, V clears, and C receives its low bit. Signed overflow sets V. Range overflow preserves the dividend and clears N, Z and C.
Reading: Atkinson (2009), pp. 59 · Astle, LWTools 4.25, ch. 3.

DIVQ

HD6309 only

```
#literal   ·   <direct   ·   offset,X   ·   >address
```

Opcode bytes. direct `11 9E` · indexed `11 AE` · extended `11 BE` · immediate `11 8E`.
Divide signed Q by a signed word. The quotient goes in W and the remainder in D. A zero divisor traps.
Flags. For an in-range quotient, N and Z describe it, V clears, and C receives its low bit. Signed overflow sets V. Range overflow preserves the dividend and clears N, Z and C.
VCC Plus's retained core has differences in DIVQ overflow and carry handling. Treat these flags as processor documentation.
Reading: Atkinson (2009), pp. 60 · Astle, LWTools 4.25, ch. 3.

INC

MC6809 and HD6309

```
<direct   ·   offset,X   ·   >address
```

Opcode bytes. direct `0C` · indexed `6C` · extended `7C`.
Operate on a memory byte. Add one.
Flags. N reflects the sign bit. Z marks a zero result. V sets on crossing from maximum positive to minimum negative. C is preserved.
Reading: Atkinson (2009), pp. 67–68 · Motorola (1981), Appendix A, INC · Astle, LWTools 4.25, ch. 3.

INCA

MC6809 and HD6309

```
No operand
```

Opcode bytes. `4C`.
Operate on register A. Add one.
Flags. N reflects the sign bit. Z marks a zero result. V sets on crossing from maximum positive to minimum negative. C is preserved.
Reading: Atkinson (2009), pp. 67–68 · Motorola (1981), Appendix A, INCA · Astle, LWTools 4.25, ch. 3.

INCB

MC6809 and HD6309

```
No operand
```

Opcode bytes. `5C`.
Operate on register B. Add one.
Flags. N reflects the sign bit. Z marks a zero result. V sets on crossing from maximum positive to minimum negative. C is preserved.
Reading: Atkinson (2009), pp. 67–68 · Motorola (1981), Appendix A, INCB · Astle, LWTools 4.25, ch. 3.

INCD

HD6309 only

```
No operand
```

Opcode bytes. `10 4C`.
Operate on register D. Add one.
Flags. N reflects the sign bit. Z marks a zero result. V sets on crossing from maximum positive to minimum negative. C is preserved.
Reading: Atkinson (2009), pp. 67–68 · Astle, LWTools 4.25, ch. 3.

INCE

HD6309 only

```
No operand
```

Opcode bytes. `11 4C`.
Operate on register E. Add one.

Flags. N reflects the sign bit. Z marks a zero result. V sets on crossing from maximum positive to minimum negative. C is preserved.
Reading: Atkinson (2009), pp. 67–68 · Astle, LWTools 4.25, ch. 3.

INCF

HD6309 only

```
No operand
```

Opcode bytes. `11 5C`.
Operate on register F. Add one.
Flags. N reflects the sign bit. Z marks a zero result. V sets on crossing from maximum positive to minimum negative. C is preserved.
Reading: Atkinson (2009), pp. 67–68 · Astle, LWTools 4.25, ch. 3.

INCW

HD6309 only

```
No operand
```

Opcode bytes. `10 5C`.
Operate on register W. Add one.
Flags. N reflects the sign bit. Z marks a zero result. V sets on crossing from maximum positive to minimum negative. C is preserved.
Reading: Atkinson (2009), pp. 67–68 · Astle, LWTools 4.25, ch. 3.

MUL

MC6809 and HD6309

```
No operand
```

Opcode bytes. `3D`.
Multiply unsigned A by B and place the 16-bit product in D.
Flags. Z marks a zero product. C receives product bit 7.
Reading: Atkinson (2009), pp. 100 · Motorola (1981), Appendix A, MUL · Astle, LWTools 4.25, ch. 3.

MULD

HD6309 only

```
#literal   ·   <direct   ·   offset,X   ·   >address
```

Opcode bytes. direct `11 9F` · indexed `11 AF` · extended `11 BF` · immediate `11 8F`.
Multiply signed D by a signed 16-bit operand and store the 32-bit product in Q.
Flags. N reflects the sign bit. Z marks a zero result. V and C clear.
Reading: Atkinson (2009), pp. 101 · Astle, LWTools 4.25, ch. 3.

NEG

MC6809 and HD6309

```
<direct   ·   offset,X   ·   >address
```

Opcode bytes. direct `00` · indexed `60` · extended `70`.
Operate on a memory byte. Subtract the value from zero.
Flags. N reflects the sign bit. Z marks a zero result. C sets for a nonzero input. V marks negation of the most negative value.
Reading: Atkinson (2009), pp. 102–103 · Motorola (1981), Appendix A, NEG · Astle, LWTools 4.25, ch. 3.

NEGA

MC6809 and HD6309

```
No operand
```

Opcode bytes. `40`.
Operate on register A. Subtract the value from zero.
Flags. N reflects the sign bit. Z marks a zero result. C sets for a nonzero input. V marks negation of the most negative value.
Reading: Atkinson (2009), pp. 102–103 · Motorola (1981), Appendix A, NEGA · Astle, LWTools 4.25, ch. 3.

NEGB

MC6809 and HD6309

```
No operand
```

Opcode bytes. `50`.
Operate on register B. Subtract the value from zero.
Flags. N reflects the sign bit. Z marks a zero result. C sets for a nonzero input. V marks negation of the most negative value.
Reading: Atkinson (2009), pp. 102–103 · Motorola (1981), Appendix A, NEGB · Astle, LWTools 4.25, ch. 3.

NEGD

HD6309 only

```
No operand
```

Opcode bytes. `10 40`.
Operate on register D. Subtract the value from zero.
Flags. N reflects the sign bit. Z marks a zero result. C sets for a nonzero input. V marks negation of the most negative value.
Reading: Atkinson (2009), pp. 102–103 · Astle, LWTools 4.25, ch. 3.

SBCA

MC6809 and HD6309

```
#literal   ·   <direct   ·   offset,X   ·   >address
```

Opcode bytes. direct `92` · indexed `A2` · extended `B2` · immediate `82`.
Subtract the operand and incoming borrow from A. Write the difference to A.
Flags. N reflects the sign bit. Z marks a zero result. V marks signed overflow. C marks a borrow. H is undefined.
Reading: Atkinson (2009), pp. 122 · Motorola (1981), Appendix A, SBCA · Astle, LWTools 4.25, ch. 3.

SBCB

MC6809 and HD6309

```
#literal   ·   <direct   ·   offset,X   ·   >address
```

Opcode bytes. direct `D2` · indexed `E2` · extended `F2` · immediate `C2`.
Subtract the operand and incoming borrow from B. Write the difference to B.
Flags. N reflects the sign bit. Z marks a zero result. V marks signed overflow. C marks a borrow. H is undefined.
Reading: Atkinson (2009), pp. 122 · Motorola (1981), Appendix A, SBCB · Astle, LWTools 4.25, ch. 3.

SBCD

HD6309 only

```
#literal   ·   <direct   ·   offset,X   ·   >address
```

Opcode bytes. direct `10 92` · indexed `10 A2` · extended `10 B2` · immediate `10 82`.
Subtract the operand and incoming borrow from D. Write the difference to D.
Flags. N reflects the sign bit. Z marks a zero result. V marks signed overflow. C marks a borrow.
Reading: Atkinson (2009), pp. 123 · Astle, LWTools 4.25, ch. 3.

SBCR

HD6309 only

```
source register,destination register
```

Opcode bytes. `10 33`.
Subtract source and borrow from destination.
Flags. N reflects the sign bit. Z marks a zero result. V marks signed overflow. C marks a borrow.
The destination selects the width. A word source supplies its low byte to a byte destination. For a word destination A/B supply D, E/F supply W, CC is zero-extended and DP supplies the high byte. PC is undefined in this family. See Atkinson, p. 143.
Reading: Atkinson (2009), pp. 124 · Astle, LWTools 4.25, ch. 3.

SEX

MC6809 and HD6309

```
No operand
```

Opcode bytes. `1D`.
Copy B's sign bit across A to extend B into D.
Flags. N reflects the sign bit. Z marks a zero result. V and C are preserved.
Reading: Atkinson (2009), pp. 125 · Motorola (1981), Appendix A, SEX · Astle, LWTools 4.25, ch. 3.

SEXW

HD6309 only

```
No operand
```

Opcode bytes. `14`.
Copy W's sign bit across D to extend W into Q.
Flags. N reflects the sign bit. Z marks a zero result. V and C are preserved.
Reading: Atkinson (2009), pp. 126 · Astle, LWTools 4.25, ch. 3.

SUBA

MC6809 and HD6309

```
#literal   ·   <direct   ·   offset,X   ·   >address
```

Opcode bytes. direct `90` · indexed `A0` · extended `B0` · immediate `80`.
Subtract the operand from A. Write the difference to A.
Flags. N reflects the sign bit. Z marks a zero result. V marks signed overflow. C marks a borrow. H is undefined.
Reading: Atkinson (2009), pp. 131 · Motorola (1981), Appendix A, SUBA · Astle, LWTools 4.25, ch. 3.

SUBB

MC6809 and HD6309

```
#literal   ·   <direct   ·   offset,X   ·   >address
```

Opcode bytes. direct `D0` · indexed `E0` · extended `F0` · immediate `C0`.
Subtract the operand from B. Write the difference to B.
Flags. N reflects the sign bit. Z marks a zero result. V marks signed overflow. C marks a borrow. H is undefined.
Reading: Atkinson (2009), pp. 131 · Motorola (1981), Appendix A, SUBB · Astle, LWTools 4.25, ch. 3.

SUBD

MC6809 and HD6309

```
#literal   ·   <direct   ·   offset,X   ·   >address
```

Opcode bytes. direct `93` · indexed `A3` · extended `B3` · immediate `83`.
Subtract the operand from D. Write the difference to D.
Flags. N reflects the sign bit. Z marks a zero result. V marks signed overflow. C marks a borrow.
Reading: Atkinson (2009), pp. 132 · Motorola (1981), Appendix A, SUBD · Astle, LWTools 4.25, ch. 3.

SUBE

HD6309 only

```
#literal   ·   <direct   ·   offset,X   ·   >address
```

Opcode bytes. direct `11 90` · indexed `11 A0` · extended `11 B0` · immediate `11 80`.
Subtract the operand from E. Write the difference to E.
Flags. N reflects the sign bit. Z marks a zero result. V marks signed overflow. C marks a borrow. H is undefined.
Reading: Atkinson (2009), pp. 131 · Astle, LWTools 4.25, ch. 3.

SUBF

HD6309 only

```
#literal   ·   <direct   ·   offset,X   ·   >address
```

Opcode bytes. direct `11 D0` · indexed `11 E0` · extended `11 F0` · immediate `11 C0`.
Subtract the operand from F. Write the difference to F.
Flags. N reflects the sign bit. Z marks a zero result. V marks signed overflow. C marks a borrow. H is undefined.
Reading: Atkinson (2009), pp. 131 · Astle, LWTools 4.25, ch. 3.

SUBR

HD6309 only

```
source register,destination register
```

Opcode bytes. `10 32`.
Subtract source from destination.
Flags. N reflects the sign bit. Z marks a zero result. V marks signed overflow. C marks a borrow.
The destination selects the width. A word source supplies its low byte to a byte destination. For a word destination A/B supply D, E/F supply W, CC is zero-extended and DP supplies the high byte. PC is undefined in this family. See Atkinson, p. 143.
Reading: Atkinson (2009), pp. 133 · Astle, LWTools 4.25, ch. 3.

SUBW

HD6309 only

```
#literal   ·   <direct   ·   offset,X   ·   >address
```

Opcode bytes. direct `10 90` · indexed `10 A0` · extended `10 B0` · immediate `10 80`.
Subtract the operand from W. Write the difference to W.
Flags. N reflects the sign bit. Z marks a zero result. V marks signed overflow. C marks a borrow.
Reading: Atkinson (2009), pp. 132 · Astle, LWTools 4.25, ch. 3.

Test and change bits

In the lab, see which value controls a pixel.
`AIM` · `ANDA` · `ANDB` · `ANDD` · `ANDR` · `BAND` · `BEOR` · `BIAND` · `BIEOR` · `BIOR` · `BITA` · `BITB` · `BITD` · `BOR` · `CLR` · `CLRA` · `CLRB` · `CLRD` · `CLRE` · `CLRF` · `CLRW` · `COM` · `COMA` · `COMB` · `COMD` · `COME` · `COMF` · `COMW` · `EIM` · `EORA` · `EORB` · `EORD` · `EORR` · `LDBT` · `OIM` · `ORA` · `ORB` · `ORD` · `ORR` · `STBT` · `TIM`

AIM

HD6309 only

```
#mask,address   ·   #mask,offset,X
```

Opcode bytes. direct `02` · indexed `62` · extended `72`.
Combine the literal mask and the addressed byte with AND. Write the result to that address.
Flags. N reflects the sign bit. Z marks a zero result. V clears. C is preserved.
Reading: Atkinson (2009), pp. 11 · Astle, LWTools 4.25, ch. 3.

ANDA

MC6809 and HD6309

```
#literal   ·   <direct   ·   offset,X   ·   >address
```

Opcode bytes. direct `94` · indexed `A4` · extended `B4` · immediate `84`.
Keep bits set in both inputs. Combine A with the operand and save the result in A.
Flags. N reflects the sign bit. Z marks a zero result. V clears. C is preserved.
Reading: Atkinson (2009), pp. 12 · Motorola (1981), Appendix A, ANDA · Astle, LWTools 4.25, ch. 3.

ANDB

MC6809 and HD6309

```
#literal   ·   <direct   ·   offset,X   ·   >address
```

Opcode bytes. direct `D4` · indexed `E4` · extended `F4` · immediate `C4`.
Keep bits set in both inputs. Combine B with the operand and save the result in B.
Flags. N reflects the sign bit. Z marks a zero result. V clears. C is preserved.
Reading: Atkinson (2009), pp. 12 · Motorola (1981), Appendix A, ANDB · Astle, LWTools 4.25, ch. 3.

ANDD

HD6309 only

```
#literal   ·   <direct   ·   offset,X   ·   >address
```

Opcode bytes. direct `10 94` · indexed `10 A4` · extended `10 B4` · immediate `10 84`.
Keep bits set in both inputs. Combine D with the operand and save the result in D.
Flags. N reflects the sign bit. Z marks a zero result. V clears. C is preserved.
Reading: Atkinson (2009), pp. 14 · Astle, LWTools 4.25, ch. 3.

ANDR

HD6309 only

```
source register,destination register
```

Opcode bytes. `10 34`.
Apply bitwise AND to the destination using the source.
Flags. N reflects the sign bit. Z marks a zero result. V clears. C is preserved.
The destination selects the width. A word source supplies its low byte to a byte destination. For a word destination A/B supply D, E/F supply W, CC is zero-extended and DP supplies the high byte. PC is undefined in this family. See Atkinson, p. 143.
Reading: Atkinson (2009), pp. 15 · Astle, LWTools 4.25, ch. 3.

BAND

HD6309 only

```
A|B|CC,source bit,destination bit,direct address
```

Opcode bytes. `11 30`.
Combine the selected memory bit with the destination register bit using AND. Only the destination bit changes.
Flags. CC can change when a destination bit in CC is selected.
Reading: Atkinson (2009), pp. 20 · Astle, LWTools 4.25, ch. 3.

BEOR

HD6309 only

```
A|B|CC,source bit,destination bit,direct address
```

Opcode bytes. `11 34`.
Combine the selected memory bit with the destination register bit using exclusive OR. Only the destination bit changes.

Flags. CC can change when a destination bit in CC is selected.
Reading: Atkinson (2009), pp. 23 · Astle, LWTools 4.25, ch. 3.

BIAND

HD6309 only

```
A|B|CC,source bit,destination bit,direct address
```

Opcode bytes. `11 31`.
Combine the selected memory bit with the destination register bit using AND with the inverted source. Only the destination bit changes.
Flags. CC can change when a destination bit in CC is selected.
Reading: Atkinson (2009), pp. 29 · Astle, LWTools 4.25, ch. 3.

BIEOR

HD6309 only

```
A|B|CC,source bit,destination bit,direct address
```

Opcode bytes. `11 35`.
Combine the selected memory bit with the destination register bit using exclusive OR with the inverted source. Only the destination bit changes.
Flags. CC can change when a destination bit in CC is selected.
Reading: Atkinson (2009), pp. 30 · Astle, LWTools 4.25, ch. 3.

BIOR

HD6309 only

```
A|B|CC,source bit,destination bit,direct address
```

Opcode bytes. `11 33`.
Combine the selected memory bit with the destination register bit using OR with the inverted source. Only the destination bit changes.
Flags. CC can change when a destination bit in CC is selected.
Reading: Atkinson (2009), pp. 31 · Astle, LWTools 4.25, ch. 3.

BITA

MC6809 and HD6309

```
#literal  ·  <direct  ·  offset,X  ·  >address
```

Opcode bytes. direct `95` · indexed `A5` · extended `B5` · immediate `85`.
Test the bits of A selected by the operand. Preserve both inputs.
Flags. N reflects the sign bit. Z marks a zero result. V clears. C is preserved.
Reading: Atkinson (2009), pp. 32 · Motorola (1981), Appendix A, BITA · Astle, LWTools 4.25, ch. 3.

BITB

MC6809 and HD6309

```
#literal  ·  <direct  ·  offset,X  ·  >address
```

Opcode bytes. direct `D5` · indexed `E5` · extended `F5` · immediate `C5`.
Test the bits of B selected by the operand. Preserve both inputs.
Flags. N reflects the sign bit. Z marks a zero result. V clears. C is preserved.
Reading: Atkinson (2009), pp. 32 · Motorola (1981), Appendix A, BITB · Astle, LWTools 4.25, ch. 3.

BITD

HD6309 only

```
#literal   ·   <direct   ·   offset,X   ·   >address
```

Opcode bytes. direct `10 95` · indexed `10 A5` · extended `10 B5` · immediate `10 85`.
Test the bits of D selected by the operand. Preserve both inputs.
Flags. N reflects the sign bit. Z marks a zero result. V clears. C is preserved.
Reading: Atkinson (2009), pp. 33 · Astle, LWTools 4.25, ch. 3.

BOR

HD6309 only

```
A|B|CC,source bit,destination bit,direct address
```

Opcode bytes. `11 32`.
Combine the selected memory bit with the destination register bit using OR. Only the destination bit changes.
Flags. CC can change when a destination bit in CC is selected.
Reading: Atkinson (2009), pp. 41 · Astle, LWTools 4.25, ch. 3.

CLR

MC6809 and HD6309

```
<direct   ·   offset,X   ·   >address
```

Opcode bytes. direct `0F` · indexed `6F` · extended `7F`.
Operate on a memory byte. Write zero.
Flags. Z sets. N, V and C clear.
Reading: Atkinson (2009), pp. 48–49 · Motorola (1981), Appendix A, CLR · Astle, LWTools 4.25, ch. 3.

CLRA

MC6809 and HD6309

```
No operand
```

Opcode bytes. `4F`.
Operate on register A. Write zero.
Flags. Z sets. N, V and C clear.
Reading: Atkinson (2009), pp. 48–49 · Motorola (1981), Appendix A, CLRA · Astle, LWTools 4.25, ch. 3.

CLRB

MC6809 and HD6309

```
No operand
```

Opcode bytes. `5F`.

Operate on register B. Write zero.
Flags. Z sets. N, V and C clear.
Reading: Atkinson (2009), pp. 48–49 · Motorola (1981), Appendix A, CLRB · Astle, LWTools 4.25, ch. 3.

CLRD

HD6309 only

```
No operand
```

Opcode bytes. `10 4F`.
Operate on register D. Write zero.
Flags. Z sets. N, V and C clear.
Reading: Atkinson (2009), pp. 48–49 · Astle, LWTools 4.25, ch. 3.

CLRE

HD6309 only

```
No operand
```

Opcode bytes. `11 4F`.
Operate on register E. Write zero.
Flags. Z sets. N, V and C clear.
Reading: Atkinson (2009), pp. 48–49 · Astle, LWTools 4.25, ch. 3.

CLRF

HD6309 only

```
No operand
```

Opcode bytes. `11 5F`.
Operate on register F. Write zero.
Flags. Z sets. N, V and C clear.
Reading: Atkinson (2009), pp. 48–49 · Astle, LWTools 4.25, ch. 3.

CLRW

HD6309 only

```
No operand
```

Opcode bytes. `10 5F`.
Operate on register W. Write zero.
Flags. Z sets. N, V and C clear.
Reading: Atkinson (2009), pp. 48–49 · Astle, LWTools 4.25, ch. 3.

COM

MC6809 and HD6309

```
<direct   ·   offset,X   ·   >address
```

Opcode bytes. direct `03` · indexed `63` · extended `73`.
Operate on a memory byte. Invert every bit.

Flags. N reflects the sign bit. Z marks a zero result. C sets and V clears.
Reading: Atkinson (2009), pp. 53–54 · Motorola (1981), Appendix A, COM · Astle, LWTools 4.25, ch. 3.

COMA

MC6809 and HD6309

```
No operand
```

Opcode bytes. `43`.
Operate on register A. Invert every bit.
Flags. N reflects the sign bit. Z marks a zero result. C sets and V clears.
Reading: Atkinson (2009), pp. 53–54 · Motorola (1981), Appendix A, COMA · Astle, LWTools 4.25, ch. 3.

COMB

MC6809 and HD6309

```
No operand
```

Opcode bytes. `53`.
Operate on register B. Invert every bit.
Flags. N reflects the sign bit. Z marks a zero result. C sets and V clears.
Reading: Atkinson (2009), pp. 53–54 · Motorola (1981), Appendix A, COMB · Astle, LWTools 4.25, ch. 3.

COMD

HD6309 only

```
No operand
```

Opcode bytes. `10 43`.
Operate on register D. Invert every bit.
Flags. N reflects the sign bit. Z marks a zero result. C sets and V clears.
Reading: Atkinson (2009), pp. 53–54 · Astle, LWTools 4.25, ch. 3.

COME

HD6309 only

```
No operand
```

Opcode bytes. `11 43`.
Operate on register E. Invert every bit.
Flags. N reflects the sign bit. Z marks a zero result. C sets and V clears.
Reading: Atkinson (2009), pp. 53–54 · Astle, LWTools 4.25, ch. 3.

COMF

HD6309 only

```
No operand
```

Opcode bytes. `11 53`.
Operate on register F. Invert every bit.
Flags. N reflects the sign bit. Z marks a zero result. C sets and V clears.

Reading: Atkinson (2009), pp. 53–54 · Astle, LWTools 4.25, ch. 3.

COMW

HD6309 only

```
No operand
```

Opcode bytes. `10 53`.
Operate on register W. Invert every bit.
Flags. N reflects the sign bit. Z marks a zero result. C sets and V clears.
Reading: Atkinson (2009), pp. 53–54 · Astle, LWTools 4.25, ch. 3.

EIM

HD6309 only

```
#mask,address   ·   #mask,offset,X
```

Opcode bytes. direct `05` · indexed `65` · extended `75`.
Combine the literal mask and the addressed byte with exclusive OR. Write the result to that address.
Flags. N reflects the sign bit. Z marks a zero result. V clears. C is preserved.
Reading: Atkinson (2009), pp. 61 · Astle, LWTools 4.25, ch. 3.

EORA

MC6809 and HD6309

```
#literal   ·   <direct   ·   offset,X   ·   >address
```

Opcode bytes. direct `98` · indexed `A8` · extended `B8` · immediate `88`.
Set bits where the inputs differ. Combine A with the operand and save the result in A.
Flags. N reflects the sign bit. Z marks a zero result. V clears. C is preserved.
Reading: Atkinson (2009), pp. 62 · Motorola (1981), Appendix A, EORA · Astle, LWTools 4.25, ch. 3.

EORB

MC6809 and HD6309

```
#literal   ·   <direct   ·   offset,X   ·   >address
```

Opcode bytes. direct `D8` · indexed `E8` · extended `F8` · immediate `C8`.
Set bits where the inputs differ. Combine B with the operand and save the result in B.
Flags. N reflects the sign bit. Z marks a zero result. V clears. C is preserved.
Reading: Atkinson (2009), pp. 62 · Motorola (1981), Appendix A, EORB · Astle, LWTools 4.25, ch. 3.

EORD

HD6309 only

```
#literal   ·   <direct   ·   offset,X   ·   >address
```

Opcode bytes. direct `10 98` · indexed `10 A8` · extended `10 B8` · immediate `10 88`.
Set bits where the inputs differ. Combine D with the operand and save the result in D.
Flags. N reflects the sign bit. Z marks a zero result. V clears. C is preserved.
Reading: Atkinson (2009), pp. 63 · Astle, LWTools 4.25, ch. 3.

EORR

HD6309 only

```
source register,destination register
```

Opcode bytes. `10 36`.
Toggle destination bits selected by the source.
Flags. N reflects the sign bit. Z marks a zero result. V clears. C is preserved.
The destination selects the width. A word source supplies its low byte to a byte destination. For a word destination A/B supply D, E/F supply W, CC is zero-extended and DP supplies the high byte. PC is undefined in this family. See Atkinson, p. 143.
Reading: Atkinson (2009), pp. 64 · Astle, LWTools 4.25, ch. 3.

LDBT

HD6309 only

```
A|B|CC,source bit,destination bit,direct address
```

Opcode bytes. `11 36`.
Combine the selected memory bit with the destination register bit using copy. Only the destination bit changes.
Flags. CC can change when a destination bit in CC is selected.
Reading: Atkinson (2009), pp. 92 · Astle, LWTools 4.25, ch. 3.

OIM

HD6309 only

```
#mask,address   ·   #mask,offset,X
```

Opcode bytes. direct `01` · indexed `61` · extended `71`.
Combine the literal mask and the addressed byte with OR. Write the result to that address.
Flags. N reflects the sign bit. Z marks a zero result. V clears. C is preserved.
Reading: Atkinson (2009), pp. 105 · Astle, LWTools 4.25, ch. 3.

ORA

MC6809 and HD6309

```
#literal   ·   <direct   ·   offset,X   ·   >address
```

Opcode bytes. direct `9A` · indexed `AA` · extended `BA` · immediate `8A`.
Set bits present in either input. Combine A with the operand and save the result in A.
Flags. N reflects the sign bit. Z marks a zero result. V clears. C is preserved.
Reading: Atkinson (2009), pp. 106 · Motorola (1981), Appendix A, ORA · Astle, LWTools 4.25, ch. 3.

ORB

MC6809 and HD6309

```
#literal   ·   <direct   ·   offset,X   ·   >address
```

Opcode bytes. direct `DA` · indexed `EA` · extended `FA` · immediate `CA`.
Set bits present in either input. Combine B with the operand and save the result in B.
Flags. N reflects the sign bit. Z marks a zero result. V clears. C is preserved.

Reading: Atkinson (2009), pp. 106 · Motorola (1981), Appendix A, ORB · Astle, LWTools 4.25, ch. 3.

ORD

HD6309 only

```
#literal   ·   <direct   ·   offset,X   ·   >address
```

Opcode bytes. direct `10 9A` · indexed `10 AA` · extended `10 BA` · immediate `10 8A`.
Set bits present in either input. Combine D with the operand and save the result in D.
Flags. N reflects the sign bit. Z marks a zero result. V clears. C is preserved.
Reading: Atkinson (2009), pp. 108 · Astle, LWTools 4.25, ch. 3.

ORR

HD6309 only

```
source register,destination register
```

Opcode bytes. `10 35`.
Apply bitwise OR to the destination using the source.
Flags. N reflects the sign bit. Z marks a zero result. V clears. C is preserved.
The destination selects the width. A word source supplies its low byte to a byte destination. For a word destination A/B supply D, E/F supply W, CC is zero-extended and DP supplies the high byte. PC is undefined in this family. See Atkinson, p. 143.
Reading: Atkinson (2009), pp. 109 · Astle, LWTools 4.25, ch. 3.

STBT

HD6309 only

```
A|B|CC,source bit,destination bit,direct address
```

Opcode bytes. `11 37`.
Copy the selected register bit to the destination bit in direct-page memory.
Flags. CC can change when a destination bit in CC is selected.
Reading: Atkinson (2009), pp. 129 · Astle, LWTools 4.25, ch. 3.

TIM

HD6309 only

```
#mask,address   ·   #mask,offset,X
```

Opcode bytes. direct `0B` · indexed `6B` · extended `7B`.
Combine the literal mask and the addressed byte with AND. Use the result only to test flags.
Flags. N reflects the sign bit. Z marks a zero result. V clears. C is preserved.
Reading: Atkinson (2009), pp. 139 · Astle, LWTools 4.25, ch. 3.

Shift and rotate

In the lab, interpret the high bit and carry.

`ASL` · `ASLA` · `ASLB` · `ASLD` · `ASR` · `ASRA` · `ASRB` · `ASRD` · `LSL` · `LSLA` · `LSLB` · `LSLD` · `LSR` · `LSRA` · `LSRB` · `LSRD` · `LSRW` · `ROL` · `ROLA` · `ROLB` · `ROLD` · `ROLW` · `ROR` · `RORA` · `RORB` · `RORD` · `RORW`

ASL

Other spellings: `LSL`.

MC6809 and HD6309

```
<direct   ·   offset,X   ·   >address
```

Opcode bytes. direct `08` · indexed `68` · extended `78`.

Operate on a memory byte. Shift one place left and fill bit 0 with zero.

Flags. N reflects the sign bit. Z marks a zero result. C receives the old top bit. V is N XOR C. H is undefined.

Reading: Atkinson (2009), pp. 16–17 · Motorola (1981), Appendix A, ASL · Astle, LWTools 4.25, ch. 3.

ASLA

Other spellings: `LSLA`.

MC6809 and HD6309

```
No operand
```

Opcode bytes. `48`.

Operate on register A. Shift one place left and fill bit 0 with zero.

Flags. N reflects the sign bit. Z marks a zero result. C receives the old top bit. V is N XOR C. H is undefined.

Reading: Atkinson (2009), pp. 16–17 · Motorola (1981), Appendix A, ASLA · Astle, LWTools 4.25, ch. 3.

ASLB

Other spellings: `LSLB`.

MC6809 and HD6309

```
No operand
```

Opcode bytes. `58`.

Operate on register B. Shift one place left and fill bit 0 with zero.

Flags. N reflects the sign bit. Z marks a zero result. C receives the old top bit. V is N XOR C. H is undefined.

Reading: Atkinson (2009), pp. 16–17 · Motorola (1981), Appendix A, ASLB · Astle, LWTools 4.25, ch. 3.

ASLD

Other spellings: `LSLD`.

HD6309 only

```
No operand
```

Opcode bytes. `10 48`.

Operate on register D. Shift one place left and fill bit 0 with zero.

Flags. N reflects the sign bit. Z marks a zero result. C receives the old top bit. V is N XOR C.

Reading: Atkinson (2009), pp. 16–17 · Astle, LWTools 4.25, ch. 3.

ASR

MC6809 and HD6309

```
<direct   ·   offset,X   ·   >address
```

Opcode bytes. direct `07` · indexed `67` · extended `77`.

Operate on a memory byte. Shift right, preserving the sign bit.

Flags. N reflects the sign bit. Z marks a zero result. C receives the old bit 0. V is preserved. H is undefined.
Reading: Atkinson (2009), pp. 18–19 · Motorola (1981), Appendix A, ASR · Astle, LWTools 4.25, ch. 3.

ASRA

MC6809 and HD6309

```
No operand
```

Opcode bytes. `47`.
Operate on register A. Shift right, preserving the sign bit.
Flags. N reflects the sign bit. Z marks a zero result. C receives the old bit 0. V is preserved. H is undefined.
Reading: Atkinson (2009), pp. 18–19 · Motorola (1981), Appendix A, ASRA · Astle, LWTools 4.25, ch. 3.

ASRB

MC6809 and HD6309

```
No operand
```

Opcode bytes. `57`.
Operate on register B. Shift right, preserving the sign bit.
Flags. N reflects the sign bit. Z marks a zero result. C receives the old bit 0. V is preserved. H is undefined.
Reading: Atkinson (2009), pp. 18–19 · Motorola (1981), Appendix A, ASRB · Astle, LWTools 4.25, ch. 3.

ASRD

HD6309 only

```
No operand
```

Opcode bytes. `10 47`.
Operate on register D. Shift right, preserving the sign bit.
Flags. N reflects the sign bit. Z marks a zero result. C receives the old bit 0. V is preserved.
Reading: Atkinson (2009), pp. 18–19 · Astle, LWTools 4.25, ch. 3.

LSR

MC6809 and HD6309

```
<direct   ·   offset,X   ·   >address
```

Opcode bytes. direct `04` · indexed `64` · extended `74`.
Operate on a memory byte. Shift right, bringing zero into the top bit.
Flags. N clears. Z marks zero. C receives the old bit 0. V is preserved.
Reading: Atkinson (2009), pp. 98–99 · Motorola (1981), Appendix A, LSR · Astle, LWTools 4.25, ch. 3.

LSRA

MC6809 and HD6309

```
No operand
```

Opcode bytes. `44`.
Operate on register A. Shift right, bringing zero into the top bit.
Flags. N clears. Z marks zero. C receives the old bit 0. V is preserved.

Reading: Atkinson (2009), pp. 98–99 · Motorola (1981), Appendix A, LSRA · Astle, LWTools 4.25, ch. 3.

LSRB

MC6809 and HD6309

```
No operand
```

Opcode bytes. `54`.
Operate on register B. Shift right, bringing zero into the top bit.
Flags. N clears. Z marks zero. C receives the old bit 0. V is preserved.
Reading: Atkinson (2009), pp. 98–99 · Motorola (1981), Appendix A, LSRB · Astle, LWTools 4.25, ch. 3.

LSRD

HD6309 only

```
No operand
```

Opcode bytes. `10 44`.
Operate on register D. Shift right, bringing zero into the top bit.
Flags. N clears. Z marks zero. C receives the old bit 0. V is preserved.
Reading: Atkinson (2009), pp. 98–99 · Astle, LWTools 4.25, ch. 3.

LSRW

HD6309 only

```
No operand
```

Opcode bytes. `10 54`.
Operate on register W. Shift right, bringing zero into the top bit.
Flags. N clears. Z marks zero. C receives the old bit 0. V is preserved.
Reading: Atkinson (2009), pp. 98–99 · Astle, LWTools 4.25, ch. 3.

ROL

MC6809 and HD6309

```
<direct   ·   offset,X   ·   >address
```

Opcode bytes. direct `09` · indexed `69` · extended `79`.
Operate on a memory byte. Shift left, moving the old carry into bit 0.
Flags. N reflects the sign bit. Z marks a zero result. C receives the old top bit. V is N XOR C.
Reading: Atkinson (2009), pp. 116–117 · Motorola (1981), Appendix A, ROL · Astle, LWTools 4.25, ch. 3.

ROLA

MC6809 and HD6309

```
No operand
```

Opcode bytes. `49`.
Operate on register A. Shift left, moving the old carry into bit 0.
Flags. N reflects the sign bit. Z marks a zero result. C receives the old top bit. V is N XOR C.
Reading: Atkinson (2009), pp. 116–117 · Motorola (1981), Appendix A, ROLA · Astle, LWTools 4.25, ch. 3.

ROLB

MC6809 and HD6309

```
No operand
```

Opcode bytes. `59`.
Operate on register B. Shift left, moving the old carry into bit 0.
Flags. N reflects the sign bit. Z marks a zero result. C receives the old top bit. V is N XOR C.
Reading: Atkinson (2009), pp. 116–117 · Motorola (1981), Appendix A, ROLB · Astle, LWTools 4.25, ch. 3.

ROLD

HD6309 only

```
No operand
```

Opcode bytes. `10 49`.
Operate on register D. Shift left, moving the old carry into bit 0.
Flags. N reflects the sign bit. Z marks a zero result. C receives the old top bit. V is N XOR C.
Reading: Atkinson (2009), pp. 116–117 · Astle, LWTools 4.25, ch. 3.

ROLW

HD6309 only

```
No operand
```

Opcode bytes. `10 59`.
Operate on register W. Shift left, moving the old carry into bit 0.
Flags. N reflects the sign bit. Z marks a zero result. C receives the old top bit. V is N XOR C.
Reading: Atkinson (2009), pp. 116–117 · Astle, LWTools 4.25, ch. 3.

ROR

MC6809 and HD6309

```
<direct   ·   offset,X   ·   >address
```

Opcode bytes. direct `06` · indexed `66` · extended `76`.
Operate on a memory byte. Shift right, moving the old carry into the top bit.
Flags. N reflects the sign bit. Z marks a zero result. C receives the old bit 0. V is preserved.
Reading: Atkinson (2009), pp. 118–119 · Motorola (1981), Appendix A, ROR · Astle, LWTools 4.25, ch. 3.

RORA

MC6809 and HD6309

```
No operand
```

Opcode bytes. `46`.
Operate on register A. Shift right, moving the old carry into the top bit.
Flags. N reflects the sign bit. Z marks a zero result. C receives the old bit 0. V is preserved.
Reading: Atkinson (2009), pp. 118–119 · Motorola (1981), Appendix A, RORA · Astle, LWTools 4.25, ch. 3.

RORB

MC6809 and HD6309

```
No operand
```

Opcode bytes. `56`.
Operate on register B. Shift right, moving the old carry into the top bit.
Flags. N reflects the sign bit. Z marks a zero result. C receives the old bit 0. V is preserved.
Reading: Atkinson (2009), pp. 118–119 · Motorola (1981), Appendix A, RORB · Astle, LWTools 4.25, ch. 3.

RORD

HD6309 only

```
No operand
```

Opcode bytes. `10 46`.
Operate on register D. Shift right, moving the old carry into the top bit.
Flags. N reflects the sign bit. Z marks a zero result. C receives the old bit 0. V is preserved.
Reading: Atkinson (2009), pp. 118–119 · Astle, LWTools 4.25, ch. 3.

RORW

HD6309 only

```
No operand
```

Opcode bytes. `10 56`.
Operate on register W. Shift right, moving the old carry into the top bit.
Flags. N reflects the sign bit. Z marks a zero result. C receives the old bit 0. V is preserved.
Reading: Atkinson (2009), pp. 118–119 · Astle, LWTools 4.25, ch. 3.

Compare and choose a path

In the lab, test signed and unsigned decisions.
`BCC` · `BCS` · `BEQ` · `BGE` · `BGT` · `BHI` · `BHS` · `BLE` · `BLO` · `BLS` · `BLT` · `BMI` · `BNE` · `BPL` · `BRA` · `BRN` · `BVC` · `BVS` · `CMPA` · `CMPB` · `CMPD` · `CMPE` · `CMPF` · `CMPR` · `CMPS` · `CMPU` · `CMPW` · `CMPX` · `CMPY` · `JMP` · `LBCC` · `LBCS` · `LBEQ` · `LBGE` · `LBGT` · `LBHI` · `LBHS` · `LBLE` · `LBLO` · `LBLS` · `LBLT` · `LBMI` · `LBNE` · `LBPL` · `LBRA` · `LBRN` · `LBVC` · `LBVS` · `TST` · `TSTA` · `TSTB` · `TSTD` · `TSTE` · `TSTF` · `TSTW`

BCC

Other spellings: `BHS`.
MC6809 and HD6309

```
target label
```

Opcode bytes. `24`.
Continue at the target when C is clear (unsigned higher or equal after a compare). Otherwise execute the next instruction.
Flags. CC is preserved.
The signed 8-bit displacement is relative to the following instruction.
Reading: Atkinson (2009), pp. 21 · Motorola (1981), Appendix A, BCC · Astle, LWTools 4.25, ch. 3.

BCS

Other spellings: `BLO`.

MC6809 and HD6309

```
target label
```

Opcode bytes. 25.
Continue at the target when C is set (unsigned lower). Otherwise execute the next instruction.
Flags. CC is preserved.
The signed 8-bit displacement is relative to the following instruction.
Reading: Atkinson (2009), pp. 22 · Motorola (1981), Appendix A, BCS · Astle, LWTools 4.25, ch. 3.

BEQ

MC6809 and HD6309

```
target label
```

Opcode bytes. 27.
Continue at the target when Z is set (equal). Otherwise execute the next instruction.
Flags. CC is preserved.
The signed 8-bit displacement is relative to the following instruction.
Reading: Atkinson (2009), pp. 24 · Motorola (1981), Appendix A, BEQ · Astle, LWTools 4.25, ch. 3.

BGE

MC6809 and HD6309

```
target label
```

Opcode bytes. 2C.
Continue at the target when N equals V (signed greater or equal). Otherwise execute the next instruction.
Flags. CC is preserved.
The signed 8-bit displacement is relative to the following instruction.
Reading: Atkinson (2009), pp. 25 · Motorola (1981), Appendix A, BGE · Astle, LWTools 4.25, ch. 3.

BGT

MC6809 and HD6309

```
target label
```

Opcode bytes. 2E.
Continue at the target when Z is clear and N equals V (signed greater). Otherwise execute the next instruction.
Flags. CC is preserved.
The signed 8-bit displacement is relative to the following instruction.
Reading: Atkinson (2009), pp. 26 · Motorola (1981), Appendix A, BGT · Astle, LWTools 4.25, ch. 3.

BHI

MC6809 and HD6309

```
target label
```

Opcode bytes. 22.

Continue at the target when C and Z are both clear (unsigned higher). Otherwise execute the next instruction.
Flags. CC is preserved.
The signed 8-bit displacement is relative to the following instruction.
Reading: Atkinson (2009), pp. 27 · Motorola (1981), Appendix A, BHI · Astle, LWTools 4.25, ch. 3.

BLE

MC6809 and HD6309

```
target label
```

Opcode bytes. 2F.
Continue at the target when Z is set or N differs from V (signed less or equal). Otherwise execute the next instruction.
Flags. CC is preserved.
The signed 8-bit displacement is relative to the following instruction.
Reading: Atkinson (2009), pp. 35 · Motorola (1981), Appendix A, BLE · Astle, LWTools 4.25, ch. 3.

BLS

MC6809 and HD6309

```
target label
```

Opcode bytes. 23.
Continue at the target when C or Z is set (unsigned lower or equal). Otherwise execute the next instruction.
Flags. CC is preserved.
The signed 8-bit displacement is relative to the following instruction.
Reading: Atkinson (2009), pp. 37 · Motorola (1981), Appendix A, BLS · Astle, LWTools 4.25, ch. 3.

BLT

MC6809 and HD6309

```
target label
```

Opcode bytes. 2D.
Continue at the target when N differs from V (signed less). Otherwise execute the next instruction.
Flags. CC is preserved.
The signed 8-bit displacement is relative to the following instruction.
Reading: Atkinson (2009), pp. 38 · Motorola (1981), Appendix A, BLT · Astle, LWTools 4.25, ch. 3.

BMI

MC6809 and HD6309

```
target label
```

Opcode bytes. 2B.
Continue at the target when N is set (negative). Otherwise execute the next instruction.
Flags. CC is preserved.
The signed 8-bit displacement is relative to the following instruction.
Reading: Atkinson (2009), pp. 39 · Motorola (1981), Appendix A, BMI · Astle, LWTools 4.25, ch. 3.

BNE

MC6809 and HD6309

```
target label
```

Opcode bytes. 26.
Continue at the target when Z is clear (unequal). Otherwise execute the next instruction.
Flags. CC is preserved.
The signed 8-bit displacement is relative to the following instruction.
Reading: Atkinson (2009), pp. 40 · Motorola (1981), Appendix A, BNE · Astle, LWTools 4.25, ch. 3.

BPL

MC6809 and HD6309

```
target label
```

Opcode bytes. 2A.
Continue at the target when N is clear (nonnegative). Otherwise execute the next instruction.
Flags. CC is preserved.
The signed 8-bit displacement is relative to the following instruction.
Reading: Atkinson (2009), pp. 42 · Motorola (1981), Appendix A, BPL · Astle, LWTools 4.25, ch. 3.

BRA

MC6809 and HD6309

```
target label
```

Opcode bytes. 20.
Continue at the target. A branch to its own label forms a hold loop.
Flags. CC is preserved.
The signed 8-bit displacement is relative to the following instruction.
Reading: Atkinson (2009), pp. 43 · Motorola (1981), Appendix A, BRA · Astle, LWTools 4.25, ch. 3.

BRN

MC6809 and HD6309

```
target label
```

Opcode bytes. 21.
Read the displacement and continue at the next instruction.
Flags. CC is preserved.
The signed 8-bit displacement is relative to the following instruction.
Reading: Atkinson (2009), pp. 44 · Motorola (1981), Appendix A, BRN · Astle, LWTools 4.25, ch. 3.

BVC

MC6809 and HD6309

```
target label
```

Opcode bytes. 28.
Continue at the target when V is clear (no signed overflow). Otherwise execute the next instruction.

Flags. CC is preserved.
The signed 8-bit displacement is relative to the following instruction.
Reading: Atkinson (2009), pp. 46 · Motorola (1981), Appendix A, BVC · Astle, LWTools 4.25, ch. 3.

BVS

MC6809 and HD6309

```
target label
```

Opcode bytes. 29.
Continue at the target when V is set (signed overflow). Otherwise execute the next instruction.
Flags. CC is preserved.
The signed 8-bit displacement is relative to the following instruction.
Reading: Atkinson (2009), pp. 47 · Motorola (1981), Appendix A, BVS · Astle, LWTools 4.25, ch. 3.

CMPA

MC6809 and HD6309

```
#literal   ·   <direct   ·   offset,X   ·   >address
```

Opcode bytes. direct `91` · indexed `A1` · extended `B1` · immediate `81`.
Subtract the operand from A. Keep A intact for a later branch.
Flags. N reflects the sign bit. Z marks a zero result. V marks signed overflow. C marks a borrow. H is undefined.
Reading: Atkinson (2009), pp. 50 · Motorola (1981), Appendix A, CMPA · Astle, LWTools 4.25, ch. 3.

CMPB

MC6809 and HD6309

```
#literal   ·   <direct   ·   offset,X   ·   >address
```

Opcode bytes. direct `D1` · indexed `E1` · extended `F1` · immediate `C1`.
Subtract the operand from B. Keep B intact for a later branch.
Flags. N reflects the sign bit. Z marks a zero result. V marks signed overflow. C marks a borrow. H is undefined.
Reading: Atkinson (2009), pp. 50 · Motorola (1981), Appendix A, CMPB · Astle, LWTools 4.25, ch. 3.

CMPD

MC6809 and HD6309

```
#literal   ·   <direct   ·   offset,X   ·   >address
```

Opcode bytes. direct `10 93` · indexed `10 A3` · extended `10 B3` · immediate `10 83`.
Subtract the operand from D. Keep D intact for a later branch.
Flags. N reflects the sign bit. Z marks a zero result. V marks signed overflow. C marks a borrow.
Reading: Atkinson (2009), pp. 51 · Motorola (1981), Appendix A, CMPD · Astle, LWTools 4.25, ch. 3.

CMPE

HD6309 only

```
#literal   ·   <direct   ·   offset,X   ·   >address
```

Opcode bytes. direct `11 91` · indexed `11 A1` · extended `11 B1` · immediate `11 81`.

Subtract the operand from E. Keep E intact for a later branch.
Flags. N reflects the sign bit. Z marks a zero result. V marks signed overflow. C marks a borrow. H is undefined.
Reading: Atkinson (2009), pp. 50 · Astle, LWTools 4.25, ch. 3.

CMPF

HD6309 only

```
#literal   ·   <direct   ·   offset,X   ·   >address
```

Opcode bytes. direct `11 D1` · indexed `11 E1` · extended `11 F1` · immediate `11 C1`.
Subtract the operand from F. Keep F intact for a later branch.
Flags. N reflects the sign bit. Z marks a zero result. V marks signed overflow. C marks a borrow. H is undefined.
Reading: Atkinson (2009), pp. 50 · Astle, LWTools 4.25, ch. 3.

CMPR

HD6309 only

```
source register,destination register
```

Opcode bytes. `10 37`.
Compare destination minus source, preserving both.
Flags. N reflects the sign bit. Z marks a zero result. V marks signed overflow. C marks a borrow.
The destination selects the width. A word source supplies its low byte to a byte destination. For a word destination A/B supply D, E/F supply W, CC is zero-extended and DP supplies the high byte. PC is undefined in this family. See Atkinson, p. 143.
Reading: Atkinson (2009), pp. 52 · Astle, LWTools 4.25, ch. 3.

CMPS

MC6809 and HD6309

```
#literal   ·   <direct   ·   offset,X   ·   >address
```

Opcode bytes. direct `11 9C` · indexed `11 AC` · extended `11 BC` · immediate `11 8C`.
Subtract the operand from S. Keep S intact for a later branch.
Flags. N reflects the sign bit. Z marks a zero result. V marks signed overflow. C marks a borrow.
Reading: Atkinson (2009), pp. 51 · Motorola (1981), Appendix A, CMPS · Astle, LWTools 4.25, ch. 3.

CMPU

MC6809 and HD6309

```
#literal   ·   <direct   ·   offset,X   ·   >address
```

Opcode bytes. direct `11 93` · indexed `11 A3` · extended `11 B3` · immediate `11 83`.
Subtract the operand from U. Keep U intact for a later branch.
Flags. N reflects the sign bit. Z marks a zero result. V marks signed overflow. C marks a borrow.
Reading: Atkinson (2009), pp. 51 · Motorola (1981), Appendix A, CMPU · Astle, LWTools 4.25, ch. 3.

CMPW

HD6309 only

```
#literal   ·   <direct   ·   offset,X   ·   >address
```

Opcode bytes. direct `10 91` · indexed `10 A1` · extended `10 B1` · immediate `10 81`.
Subtract the operand from W. Keep W intact for a later branch.
Flags. N reflects the sign bit. Z marks a zero result. V marks signed overflow. C marks a borrow.
Reading: Atkinson (2009), pp. 51 · Astle, LWTools 4.25, ch. 3.

CMPX

MC6809 and HD6309

```
#literal   ·   <direct   ·   offset,X   ·   >address
```

Opcode bytes. direct `9C` · indexed `AC` · extended `BC` · immediate `8C`.
Subtract the operand from X. Keep X intact for a later branch.
Flags. N reflects the sign bit. Z marks a zero result. V marks signed overflow. C marks a borrow.
Reading: Atkinson (2009), pp. 51 · Motorola (1981), Appendix A, CMPX · Astle, LWTools 4.25, ch. 3.

CMPY

MC6809 and HD6309

```
#literal   ·   <direct   ·   offset,X   ·   >address
```

Opcode bytes. direct `10 9C` · indexed `10 AC` · extended `10 BC` · immediate `10 8C`.
Subtract the operand from Y. Keep Y intact for a later branch.
Flags. N reflects the sign bit. Z marks a zero result. V marks signed overflow. C marks a borrow.
Reading: Atkinson (2009), pp. 51 · Motorola (1981), Appendix A, CMPY · Astle, LWTools 4.25, ch. 3.

JMP

MC6809 and HD6309

```
<direct   ·   offset,X   ·   >address
```

Opcode bytes. direct `0E` · indexed `6E` · extended `7E`.
Set PC to the effective address.
Flags. CC is preserved.
Reading: Atkinson (2009), pp. 69 · Motorola (1981), Appendix A, JMP · Astle, LWTools 4.25, ch. 3.

LBCC

Other spellings: `LBHS`.

MC6809 and HD6309

```
target label
```

Opcode bytes. `10 24`.
Continue at the target when C is clear (unsigned higher or equal after a compare). Otherwise execute the next instruction.
Flags. CC is preserved.
The signed 16-bit displacement is relative to the following instruction.
Reading: Atkinson (2009), pp. 71 · Motorola (1981), Appendix A, LBCC · Astle, LWTools 4.25, ch. 3.

LBCS

Other spellings: `LBL0`.

MC6809 and HD6309

```
target label
```

Opcode bytes. `10 25`.
Continue at the target when C is set (unsigned lower). Otherwise execute the next instruction.
Flags. CC is preserved.
The signed 16-bit displacement is relative to the following instruction.
Reading: Atkinson (2009), pp. 72 · Motorola (1981), Appendix A, LBCS · Astle, LWTools 4.25, ch. 3.

LBEQ

MC6809 and HD6309

```
target label
```

Opcode bytes. `10 27`.
Continue at the target when Z is set (equal). Otherwise execute the next instruction.
Flags. CC is preserved.
The signed 16-bit displacement is relative to the following instruction.
Reading: Atkinson (2009), pp. 73 · Motorola (1981), Appendix A, LBEQ · Astle, LWTools 4.25, ch. 3.

LBGE

MC6809 and HD6309

```
target label
```

Opcode bytes. `10 2C`.
Continue at the target when N equals V (signed greater or equal). Otherwise execute the next instruction.
Flags. CC is preserved.
The signed 16-bit displacement is relative to the following instruction.
Reading: Atkinson (2009), pp. 74 · Motorola (1981), Appendix A, LBGE · Astle, LWTools 4.25, ch. 3.

LBGT

MC6809 and HD6309

```
target label
```

Opcode bytes. `10 2E`.
Continue at the target when Z is clear and N equals V (signed greater). Otherwise execute the next instruction.
Flags. CC is preserved.
The signed 16-bit displacement is relative to the following instruction.
Reading: Atkinson (2009), pp. 75 · Motorola (1981), Appendix A, LBGT · Astle, LWTools 4.25, ch. 3.

LBHI

MC6809 and HD6309

```
target label
```

Opcode bytes. `10 22`.

Continue at the target when C and Z are both clear (unsigned higher). Otherwise execute the next instruction.
Flags. CC is preserved.
The signed 16-bit displacement is relative to the following instruction.
Reading: Atkinson (2009), pp. 76 · Motorola (1981), Appendix A, LBHI · Astle, LWTools 4.25, ch. 3.

LBLE

MC6809 and HD6309

```
target label
```

Opcode bytes. `10 2F`.
Continue at the target when Z is set or N differs from V (signed less or equal). Otherwise execute the next instruction.
Flags. CC is preserved.
The signed 16-bit displacement is relative to the following instruction.
Reading: Atkinson (2009), pp. 78 · Motorola (1981), Appendix A, LBLE · Astle, LWTools 4.25, ch. 3.

LBLS

MC6809 and HD6309

```
target label
```

Opcode bytes. `10 23`.
Continue at the target when C or Z is set (unsigned lower or equal). Otherwise execute the next instruction.
Flags. CC is preserved.
The signed 16-bit displacement is relative to the following instruction.
Reading: Atkinson (2009), pp. 80 · Motorola (1981), Appendix A, LBLS · Astle, LWTools 4.25, ch. 3.

LBLT

MC6809 and HD6309

```
target label
```

Opcode bytes. `10 2D`.
Continue at the target when N differs from V (signed less). Otherwise execute the next instruction.
Flags. CC is preserved.
The signed 16-bit displacement is relative to the following instruction.
Reading: Atkinson (2009), pp. 81 · Motorola (1981), Appendix A, LBLT · Astle, LWTools 4.25, ch. 3.

LBMI

MC6809 and HD6309

```
target label
```

Opcode bytes. `10 2B`.
Continue at the target when N is set (negative). Otherwise execute the next instruction.
Flags. CC is preserved.
The signed 16-bit displacement is relative to the following instruction.
Reading: Atkinson (2009), pp. 82 · Motorola (1981), Appendix A, LBMI · Astle, LWTools 4.25, ch. 3.

LBNE

MC6809 and HD6309

```
target label
```

Opcode bytes. `10 26`.
Continue at the target when Z is clear (unequal). Otherwise execute the next instruction.
Flags. CC is preserved.
The signed 16-bit displacement is relative to the following instruction.
Reading: Atkinson (2009), pp. 83 · Motorola (1981), Appendix A, LBNE · Astle, LWTools 4.25, ch. 3.

LBPL

MC6809 and HD6309

```
target label
```

Opcode bytes. `10 2A`.
Continue at the target when N is clear (nonnegative). Otherwise execute the next instruction.
Flags. CC is preserved.
The signed 16-bit displacement is relative to the following instruction.
Reading: Atkinson (2009), pp. 84 · Motorola (1981), Appendix A, LBPL · Astle, LWTools 4.25, ch. 3.

LBRA

MC6809 and HD6309

```
target label
```

Opcode bytes. `16`.
Continue at the target. A branch to its own label forms a hold loop.
Flags. CC is preserved.
The signed 16-bit displacement is relative to the following instruction.
Reading: Atkinson (2009), pp. 85 · Motorola (1981), Appendix A, LBRA · Astle, LWTools 4.25, ch. 3.

LBRN

MC6809 and HD6309

```
target label
```

Opcode bytes. `10 21`.
Read the displacement and continue at the next instruction.
Flags. CC is preserved.
The signed 16-bit displacement is relative to the following instruction.
Reading: Atkinson (2009), pp. 86 · Motorola (1981), Appendix A, LBRN · Astle, LWTools 4.25, ch. 3.

LBVC

MC6809 and HD6309

```
target label
```

Opcode bytes. `10 28`.
Continue at the target when V is clear (no signed overflow). Otherwise execute the next instruction.

Flags. CC is preserved.
The signed 16-bit displacement is relative to the following instruction.
Reading: Atkinson (2009), pp. 88 · Motorola (1981), Appendix A, LBVC · Astle, LWTools 4.25, ch. 3.

LBVS

MC6809 and HD6309

```
target label
```

Opcode bytes. `10 29`.
Continue at the target when V is set (signed overflow). Otherwise execute the next instruction.
Flags. CC is preserved.
The signed 16-bit displacement is relative to the following instruction.
Reading: Atkinson (2009), pp. 89 · Motorola (1981), Appendix A, LBVS · Astle, LWTools 4.25, ch. 3.

TST

MC6809 and HD6309

```
<direct   ·   offset,X   ·   >address
```

Opcode bytes. direct `0D` · indexed `6D` · extended `7D`.
Operate on a memory byte. Test the value while preserving it.
Flags. N reflects the sign bit. Z marks a zero result. V clears. C is preserved.
Reading: Atkinson (2009), pp. 140–141 · Motorola (1981), Appendix A, TST · Astle, LWTools 4.25, ch. 3.

TSTA

MC6809 and HD6309

```
No operand
```

Opcode bytes. `4D`.
Operate on register A. Test the value while preserving it.
Flags. N reflects the sign bit. Z marks a zero result. V clears. C is preserved.
Reading: Atkinson (2009), pp. 140–141 · Motorola (1981), Appendix A, TSTA · Astle, LWTools 4.25, ch. 3.

TSTB

MC6809 and HD6309

```
No operand
```

Opcode bytes. `5D`.
Operate on register B. Test the value while preserving it.
Flags. N reflects the sign bit. Z marks a zero result. V clears. C is preserved.
Reading: Atkinson (2009), pp. 140–141 · Motorola (1981), Appendix A, TSTB · Astle, LWTools 4.25, ch. 3.

TSTD

HD6309 only

```
No operand
```

Opcode bytes. `10 4D`.

Operate on register D. Test the value while preserving it.
Flags. N reflects the sign bit. Z marks a zero result. V clears. C is preserved.
Reading: Atkinson (2009), pp. 140–141 · Astle, LWTools 4.25, ch. 3.

TSTE

HD6309 only

```
No operand
```

Opcode bytes. `11 4D`.
Operate on register E. Test the value while preserving it.
Flags. N reflects the sign bit. Z marks a zero result. V clears. C is preserved.
Reading: Atkinson (2009), pp. 140–141 · Astle, LWTools 4.25, ch. 3.

TSTF

HD6309 only

```
No operand
```

Opcode bytes. `11 5D`.
Operate on register F. Test the value while preserving it.
Flags. N reflects the sign bit. Z marks a zero result. V clears. C is preserved.
Reading: Atkinson (2009), pp. 140–141 · Astle, LWTools 4.25, ch. 3.

TSTW

HD6309 only

```
No operand
```

Opcode bytes. `10 5D`.
Operate on register W. Test the value while preserving it.
Flags. N reflects the sign bit. Z marks a zero result. V clears. C is preserved.
Reading: Atkinson (2009), pp. 140–141 · Astle, LWTools 4.25, ch. 3.

Call, save and return

In the lab, follow a return address on S.
`BSR` · `JSR` · `LBSR` · `PSHS` · `PSHSW` · `PSHU` · `PSHUW` · `PULS` · `PULSW` · `PULU` · `PULUW` · `RTS`

BSR

MC6809 and HD6309

```
target label
```

Opcode bytes. `8D`.
Push the following instruction's address on S, then continue at the target. RTS returns to the caller.
Flags. CC is preserved.
The signed 8-bit displacement is relative to the following instruction.
Reading: Atkinson (2009), pp. 45 · Motorola (1981), Appendix A, BSR · Astle, LWTools 4.25, ch. 3.

JSR

MC6809 and HD6309

```
<direct   ·   offset,X   ·   >address
```

Opcode bytes. direct `9D` · indexed `AD` · extended `BD`.
Push the return address on S, then set PC to the effective address.
Flags. CC is preserved.
Reading: Atkinson (2009), pp. 70 · Motorola (1981), Appendix A, JSR · Astle, LWTools 4.25, ch. 3.

LBSR

MC6809 and HD6309

```
target label
```

Opcode bytes. `17`.
Push the following instruction's address on S, then continue at the target. RTS returns to the caller.
Flags. CC is preserved.
The signed 16-bit displacement is relative to the following instruction.
Reading: Atkinson (2009), pp. 87 · Motorola (1981), Appendix A, LBSR · Astle, LWTools 4.25, ch. 3.

PSHS

MC6809 and HD6309

```
register[,register…]   ·   #mask
```

Opcode bytes. `34`.
Save selected registers on S. The stack grows toward lower addresses.
Flags. CC is preserved.
Select CC, A, B, DP, X, Y, U or PC. LWASM also accepts D for A and B. Hardware determines the register order.
Reading: Atkinson (2009), pp. 110 · Motorola (1981), Appendix A, PSHS · Astle, LWTools 4.25, ch. 3.

PSHSW

HD6309 only

```
No operand
```

Opcode bytes. `10 38`.
Push both bytes of W onto S, subtracting two from the pointer.
Flags. CC is preserved.
Reading: Atkinson (2009), pp. 111 · Astle, LWTools 4.25, ch. 3.

PSHU

MC6809 and HD6309

```
register[,register…]   ·   #mask
```

Opcode bytes. `36`.
Save selected registers on U. The stack grows toward lower addresses.
Flags. CC is preserved.
Select CC, A, B, DP, X, Y, S or PC. LWASM also accepts D for A and B. Hardware determines the register order.

Reading: Atkinson (2009), pp. 110 · Motorola (1981), Appendix A, PSHU · Astle, LWTools 4.25, ch. 3.

PSHUW

HD6309 only

```
No operand
```

Opcode bytes. `10 3A`.
Push both bytes of W onto U, subtracting two from the pointer.
Flags. CC is preserved.
Reading: Atkinson (2009), pp. 112 · Astle, LWTools 4.25, ch. 3.

PULS

MC6809 and HD6309

```
register[,register…]   ·   #mask
```

Opcode bytes. `35`.
Restore selected registers from S, advancing the stack pointer. Pulling PC transfers control.
Flags. Pulling CC replaces all its flags.
Select CC, A, B, DP, X, Y, U or PC. LWASM also accepts D for A and B. Hardware determines the register order.
Reading: Atkinson (2009), pp. 113 · Motorola (1981), Appendix A, PULS · Astle, LWTools 4.25, ch. 3.

PULSW

HD6309 only

```
No operand
```

Opcode bytes. `10 39`.
Pull two bytes into W from S, advancing the pointer by two.
Flags. CC is preserved.
Reading: Atkinson (2009), pp. 114 · Astle, LWTools 4.25, ch. 3.

PULU

MC6809 and HD6309

```
register[,register…]   ·   #mask
```

Opcode bytes. `37`.
Restore selected registers from U, advancing the stack pointer. Pulling PC transfers control.
Flags. Pulling CC replaces all its flags.
Select CC, A, B, DP, X, Y, S or PC. LWASM also accepts D for A and B. Hardware determines the register order.
Reading: Atkinson (2009), pp. 113 · Motorola (1981), Appendix A, PULU · Astle, LWTools 4.25, ch. 3.

PULUW

HD6309 only

```
No operand
```

Opcode bytes. `10 3B`.
Pull two bytes into W from U, advancing the pointer by two.

Flags. CC is preserved.
Reading: Atkinson (2009), pp. 115 · Astle, LWTools 4.25, ch. 3.

RTS

MC6809 and HD6309

```
No operand
```

Opcode bytes. `39`.
Pull the return address from S into PC and advance S by two.
Flags. CC is preserved.
Reading: Atkinson (2009), pp. 121 · Motorola (1981), Appendix A, RTS · Astle, LWTools 4.25, ch. 3.

Interrupts and processor control

In the lab, examine the saved interrupt frame.
`ANDCC` · `BITMD` · `CWAI` · `LDMD` · `NOP` · `ORCC` · `RTI` · `SWI` · `SWI2` · `SWI3` · `SYNC`

ANDCC

MC6809 and HD6309

```
#mask
```

Opcode bytes. `1C`.
Keep only CC bits selected by the mask.
Flags. Any CC bit can clear.
Reading: Atkinson (2009), pp. 13 · Motorola (1981), Appendix A, ANDCC · Astle, LWTools 4.25, ch. 3.

BITMD

HD6309 only

```
#mask
```

Opcode bytes. `11 3C`.
Test the masked trap status bits of MD. Reading a selected sticky error clears it.
Flags. Z sets if no selected error was pending.
Reading: Atkinson (2009), pp. 34 · Astle, LWTools 4.25, ch. 3.

CWAI

MC6809 and HD6309

```
#mask
```

Opcode bytes. `3C`.
AND CC with the mask, stack the full interrupt context on S, then wait for an eligible interrupt.
Flags. The mask updates CC. E sets for the stacked full context.
Reading: Atkinson (2009), pp. 55 · Motorola (1981), Appendix A, CWAI · Astle, LWTools 4.25, ch. 3.

LDMD

HD6309 only

```
#mask
```

Opcode bytes. `11 3D`.
Write the low two mode-control bits of MD.
Flags. CC is preserved.
Reading: Atkinson (2009), pp. 93 · Astle, LWTools 4.25, ch. 3.

NOP

MC6809 and HD6309

```
No operand
```

Opcode bytes. `12`.
Advance to the following instruction.
Flags. CC is preserved.
Reading: Atkinson (2009), pp. 104 · Motorola (1981), Appendix A, NOP · Astle, LWTools 4.25, ch. 3.

ORCC

MC6809 and HD6309

```
#mask
```

Opcode bytes. `1A`.
Set each CC bit selected by the mask.
Flags. Any CC bit can set.
Reading: Atkinson (2009), pp. 107 · Motorola (1981), Appendix A, ORCC · Astle, LWTools 4.25, ch. 3.

RTI

MC6809 and HD6309

```
No operand
```

Opcode bytes. `3B`.
Pull CC and the return state from S. Stacked E determines whether to restore a full frame.
Flags. Restores saved CC.
Reading: Atkinson (2009), pp. 120 · Motorola (1981), Appendix A, RTI · Astle, LWTools 4.25, ch. 3.

SWI

MC6809 and HD6309

```
No operand
```

Opcode bytes. `3F`.
Save a full interrupt frame on S and fetch the new PC from $FFFA.
Flags. E sets. I and F set after stacking.
Reading: Atkinson (2009), pp. 134 · Motorola (1981), Appendix A, SWI · Astle, LWTools 4.25, ch. 3.

SWI2

MC6809 and HD6309

```
No operand
```

Opcode bytes. `10 3F`.
Save a full interrupt frame on S and fetch the new PC from $FFF4.
Flags. E sets. I and F keep their values.
Reading: Atkinson (2009), pp. 134 · Motorola (1981), Appendix A, SWI · Astle, LWTools 4.25, ch. 3.

SWI3

MC6809 and HD6309

```
No operand
```

Opcode bytes. `11 3F`.
Save a full interrupt frame on S and fetch the new PC from $FFF2.
Flags. E sets. I and F keep their values.
Reading: Atkinson (2009), pp. 134 · Motorola (1981), Appendix A, SWI · Astle, LWTools 4.25, ch. 3.

SYNC

MC6809 and HD6309

```
No operand
```

Opcode bytes. `13`.
Wait for an interrupt request. Even a masked request can release the wait.
Flags. CC is preserved.
Reading: Atkinson (2009), pp. 135 · Motorola (1981), Appendix A, SYNC · Astle, LWTools 4.25, ch. 3.

Transfer a block

In the lab, inspect the destination and W.
`COPY` · `COPY+` · `COPY–` · `EXP` · `EXPAND` · `IMP` · `IMPLODE` · `TFM` · `TFRM` · `TFRP` · `TFRR` · `TFRS`

TFM

Other spellings: `COPY`, `COPY+`, `COPY–`, `TFRP`, `TFRM`, `IMP`, `IMPLODE`, `TFRS`, `EXP`, `EXPAND`, `TFRR`.
HD6309 only

```
r+,r+   ·   r–,r–   ·   r+,r   ·   r,r+
```

Opcode bytes. r+,r+ `11 38` · r-,r- `11 39` · r+,r `11 3A` · r,r+ `11 3B`.
Copy W bytes between addresses held in D, X, Y, U or S. Each suffix advances, retreats or holds its pointer after a byte. W counts down to zero. Without interruption the transfer costs 6 cycles plus 3 per byte.
Flags. CC is preserved.
LWASM's alternative spellings take two bare registers. COPY/COPY+/TFRP advance both, COPY-/TFRM retreat both, IMP/IMPLODE/TFRS advance the source and EXP/EXPAND/TFRR advance the destination. In VCC Plus, debugger Step finishes the block. Continue can hit a breakpoint again between bytes.
Reading: Atkinson (2009), pp. 136 · VCC Plus, Native Debugger, HD6309 Step · Astle, LWTools 4.25, ch. 3.

Registers, addressing and LWASM

Appendix B

Suppose the debugger says X points at the letter O. `LDA ,X+` reads O and advances X. `LEAX packet,PCR` finds the address of the letter and puts that address in X. The result in the register tells you which question you asked. Keep this page beside the receiver when one printed symbol changes the meaning of a line.

Read the register display

Look here	Use it for
A, B, D	Two 8-bit accumulators or one 16-bit value. A is the high byte of D. A change to A or B changes D immediately.
X, Y, U, S	Four 16-bit registers for indexed addressing. U and S can also address stacks. Hardware uses S on a subroutine call or interrupt.
PC, DP	Find the next instruction. Supply the upper address byte for a direct-page operand.
CC	Read the condition bits from high to low as E, F, H, I, N, Z, V, C. On a compare, read C for unsigned borrow and N together with V for signed order.
HD6309 E, F, W, Q	E and F form W, just as A and B form D. Q combines D above W.
HD6309 V, MD	V is another 16-bit register. MD controls the processor's execution mode and FIRQ frame choice and reports trap status.

The names E, F and V each appear twice in the Hitachi debugger. The accumulators E and F hold eight bits each, and the register V holds sixteen. The flags CC.E, CC.F and CC.V each hold one bit. For an interrupt return the saved **CC.E** answers whether an entire state frame was stacked. For a 6309 transfer the **E register** is the high half of the count in W. The interrupt exercise and the block-copy exercise let you see the distinction.

A

MC6809 and HD6309

Eight-bit accumulator. A forms the high byte of D.

Reading: Motorola (1981), ch. 1.

B

MC6809 and HD6309

Eight-bit accumulator. B forms the low byte of D.

Reading: Motorola (1981), ch. 1.

CC.C

MC6809 and HD6309

Carry or borrow. An unsigned addition sets C when the sum exceeds the register. A comparison sets C when the unsigned left operand is lower.

Reading: Motorola (1981), ch. 1.

CC.E

MC6809 and HD6309

Entire state stacked. RTI checks E in the saved condition byte to choose between the short and full interrupt frames.

Reading: Motorola (1981), ch. 1.

CC.F

MC6809 and HD6309

Fast interrupt request mask. Set F to mask FIRQ. A saved CC value can restore the previous mask.

Reading: Motorola (1981), ch. 1.

CC.H

MC6809 and HD6309

Half carry. An eight-bit addition places the carry from bit 3 here. DAA uses H to adjust a packed decimal result.

Reading: Motorola (1981), ch. 1.

CC.I

MC6809 and HD6309

Interrupt request mask. Set I to mask IRQ. Clear I to permit IRQ when the external request is active.

Reading: Motorola (1981), ch. 1.

CC.N

MC6809 and HD6309

Negative. For operations that set N from the result, it copies the most significant bit. A signed comparison also needs V.

Reading: Motorola (1981), ch. 1.

CC.V

MC6809 and HD6309

Signed overflow. For example, adding one to $7F produces $80 in a byte and sets V.

Reading: Motorola (1981), ch. 1.

CC.Z

MC6809 and HD6309

Zero. Many arithmetic and load instructions set Z when the result is zero. After a compare, BEQ uses Z to test equality.

Reading: Motorola (1981), ch. 1.

Condition codes

Other spellings: `CC`.

MC6809 and HD6309

CC holds eight one-bit values. E marks an entire interrupt frame. F masks FIRQ, H is half carry, I masks IRQ. N is negative, Z zero, V signed overflow and C carry or borrow.

Branch choices depend on how the previous instruction defined these bits. For example, Z=1 after CMP means equal; C=1 after CMP means unsigned lower. An undefined bit cannot support a portable decision.

Reading: Motorola (1981), ch. 1.

D

MC6809 and HD6309

Sixteen-bit accumulator built from A (high) and B (low). Writing D immediately changes both halves.

Reading: Motorola (1981), ch. 1.

DP

MC6809 and HD6309

Direct-page register. Its byte supplies the upper address bits of a direct operand. LWASM SETDP describes your assumption during assembly.

Reading: Motorola (1981), ch. 1.

E

HD6309 only

Eight-bit accumulator and high byte of the HD6309 W register. CC.E is a separate flag.

Reading: Burke (1993), register file.

F

HD6309 only

Eight-bit accumulator and low byte of W. CC.F is the fast-interrupt mask.

Reading: Burke (1993), register file.

HD6309 modes

HD6309 only

The HD6309 starts in emulation mode. Many instructions take fewer cycles in native mode. LDMD #1 selects native mode with the ordinary short FIRQ frame. Mode selection is independent of the LWASM target.

MD bit 0 selects native mode. Bit 1 selects full FIRQ stacking. A native full-state interrupt frame includes E and F; the mode on return must agree with the frame. Cycle counts also depend on indexed postbytes, branch outcomes and transfer lengths. In VCC Plus, Step completes a TFM instruction across its byte transfers.

Reading: Atkinson (2009), pp. 2, 120, 136, 144 · Burke (1993), mode register · VCC Plus, Native Debugger, HD6309 Step.

MD

HD6309 only

Mode register. LDMD controls execution mode and FIRQ stacking. BITMD tests the latched illegal-instruction and division-by-zero status.

Reading: Burke (1993), register file.

PC

MC6809 and HD6309

Program counter. Branches, jumps, calls and returns change the address of the next instruction.

Reading: Motorola (1981), ch. 1.

Q

HD6309 only

Thirty-two-bit accumulator. From most to least significant its bytes are A, B, E, F. D is the high word and W the low word.

Reading: Burke (1993), register file.

S

MC6809 and HD6309

Sixteen-bit hardware stack pointer. JSR, interrupts and PSHS use memory below S. Initialize a writable stack before making a standalone call.

Reading: Motorola (1981), ch. 1.

U

MC6809 and HD6309

Sixteen-bit user stack pointer, also available as an index register. PSHU and PULU use this stack.

Reading: Motorola (1981), ch. 1.

V

HD6309 only

Additional 16-bit HD6309 register used in register operations. CC.V is a separate overflow flag.
Reading: Burke (1993), register file.

W

HD6309 only

Sixteen-bit accumulator combining E (high) and F (low). TFM uses W as a byte count.
Reading: Burke (1993), register file.

X

MC6809 and HD6309

Sixteen-bit index register. Use X to address a table or traverse bytes.
Reading: Motorola (1981), ch. 1.

Y

MC6809 and HD6309

Sixteen-bit index register. X and Y can walk independent buffers.
Reading: Motorola (1981), ch. 1.

Decode an operand

Read the punctuation first. In these LWASM patterns, # selects the value itself; < or > requests a shorter or longer address encoding. An indexed operand describes how to form an address. Brackets add a pointer fetch. The instruction determines what happens to the resulting value or address.

Look at the receiver's source	What to check in the debugger
`LEAX packet,PCR`	X receives the assembled address of the packet. PCR lets the assembler encode its distance from the following instruction.
`ADDA ,X` followed by `LDA ,X+`	Both instructions read the same byte. Only the second advances X. Follow the saving and restoring of A around the copy.
`JSR [CHROUT]`	The two bytes at the CHROUT vector supply the subroutine address. At entry the saved return address is on S.
`BNE next`	The displacement is signed, measured from PC after the branch operand. This branch goes backward while the count is nonzero.

A CPU address selects memory *through* the current map. It identifies a location within a physical RAM block only when that block supplies the access. The MMU lesson shows how to check the active task, block and offset before drawing conclusions from a memory dump.

Direct

MC6809 and HD6309

```
LDA <$34
```

Join DP with the 8-bit operand to make the memory address. With DP=$12, LDA <$34 reads $1234.
Reading: Motorola (1981), ch. 2 · Astle, LWTools 4.25, ch. 3.

Extended

MC6809 and HD6309

```
LDA >$0400
```

Use the full 16-bit address in the instruction. LDA >$0400 reads a byte there.
On the CoCo 3, the address in STA >$1A00 selects a byte through the CPU memory map. The MMU chooses the physical 8 KiB block and preserves the low 13 address bits. Use Mapped CPU in the debugger to inspect that store.
Reading: Tepolt (1987), p. 20, fig. 3-4 · Motorola (1981), ch. 2 · Astle, LWTools 4.25, ch. 3.

HD6309 indexed

HD6309 only

```
LDA E,X   ·   LDA ,W++
```

E, F and W can supply offsets for X, Y, U or S. W also supports ,W, offset,W, ,W++ and ,--W, including their indirect forms.
Reading: Atkinson (2009), p. 148 · Astle, LWTools 4.25, ch. 3.

Immediate

MC6809 and HD6309

```
LDA #$20
```

Use the following operand bytes as the value.
Reading: Motorola (1981), ch. 2 · Astle, LWTools 4.25, ch. 3.

Indexed

MC6809 and HD6309

```
LDA ,X+   ·   LDD 2,S   ·   LDA B,Y
```

Calculate an address from a register and an offset. LDA ,X+ reads at the original X then increments X by one. Use ,X++ for a two-byte advance. Indexed operands can also use A, B or D as signed offsets.
Reading: Motorola (1981), ch. 2 · Astle, LWTools 4.25, ch. 3.

Indirect

MC6809 and HD6309

```
LDA [,X++]   ·   JMP [$4000]
```

Use square brackets to fetch a two-byte pointer at the intermediate address, then access the final address. Use a two-byte increment when walking a pointer table.
Reading: Motorola (1981), ch. 2 · Astle, LWTools 4.25, ch. 3.

Inherent

MC6809 and HD6309

```
INCA
```

The opcode specifies the operation and its operands. INCA, for example, acts on A.
Reading: Motorola (1981), ch. 2 · Astle, LWTools 4.25, ch. 3.

Relative

MC6809 and HD6309

```
BNE again   ·   LDA table,PCR
```

Add a signed displacement to PC after the operand. Branch targets are labels. Indexed PC-relative data references use PCR in LWASM.
Reading: Motorola (1981), ch. 2 · Astle, LWTools 4.25, ch. 3.

Assembler index

When you single-step, the processor fetches instruction bytes. `FCC "ORB"` produces three data bytes; `EQU` records a value the assembler can substitute later; `END start` supplies the execution address for this DECB build. None is a step the processor takes to construct the packet. The linked build shows how a symbol that names a location in one module receives its final CPU address.
Find the directive by its name or its alternative spelling in this book's search. Each entry gives the canonical LWASM syntax. Sections, imports, exports and relocatable object output belong together. Origin and entry directives for an absolute binary belong to a different build. A directive that reserves storage can advance the location counter without depositing a known byte; use an explicit initializer when the exercise depends on that byte's first value.

*PRAGMAPOP

```
*PRAGMAPOP
```

Restore saved pragma settings.
LWASM directive. Its effects take place during assembly.
Reading: Astle, LWTools 4.25, ch. 3.

*PRAGMAPUSH

```
*PRAGMAPUSH
```

Save pragma settings on the assembler's pragma stack.
LWASM directive. Its effects take place during assembly.
Reading: Astle, LWTools 4.25, ch. 3.

ALIGN

```
ALIGN boundary
```

Advance to a boundary divisible by the supplied value.
LWASM directive. Its effects take place during assembly.
Reading: Astle, LWTools 4.25, ch. 3.

DEPHASE

```
DEPHASE
```

Resume the original address calculation after PHASE.
LWASM directive. Its effects take place during assembly.

Reading: Astle, LWTools 4.25, ch. 3.

DTB

```
DTB
```

Emit six host-local date and time bytes.
LWASM directive. Its effects take place during assembly.
Reading: Astle, LWTools 4.25, ch. 3.

DTS

```
DTS
```

Emit a textual build timestamp. Its value changes with the host clock.
LWASM directive. Its effects take place during assembly.
Reading: Astle, LWTools 4.25, ch. 3.

ELSE

```
ELSE
```

Select the alternate branch of a conditional assembly block.
LWASM directive. Its effects take place during assembly.
Reading: Astle, LWTools 4.25, ch. 3.

END

Other spellings: `.END`.

```
END entry
```

Finish the assembly and specify the DECB execution address. END itself emits no machine instruction.
LWASM directive. Its effects take place during assembly.
Reading: Astle, LWTools 4.25, ch. 3.

ENDC

Other spellings: `ENDIF`.

```
ENDC
```

Close the current conditional assembly block.
LWASM directive. Its effects take place during assembly.
Reading: Astle, LWTools 4.25, ch. 3.

ENDM

```
ENDM
```

Finish a macro definition.
LWASM directive. Its effects take place during assembly.
Reading: Astle, LWTools 4.25, ch. 3.

ENDS

Other spellings: `ENDSTRUCT`.

```
ENDS
```

Close a structure definition.
LWASM directive. Its effects take place during assembly.
Reading: Astle, LWTools 4.25, ch. 3.

ENDSECT

Other spellings: `ENDSECTION`.

```
ENDSECT
```

Finish the current object section.
LWASM directive. Its effects take place during assembly.
Reading: Astle, LWTools 4.25, ch. 3.

EQU

Other spellings: =.

```
name EQU expression
```

Assign a fixed value to a symbol.
LWASM directive. Its effects take place during assembly.
Reading: Astle, LWTools 4.25, ch. 3.

ERROR

```
ERROR message
```

Report an assembly error with the supplied text.
LWASM directive. Its effects take place during assembly.
Reading: Astle, LWTools 4.25, ch. 3.

EXPORT

Other spellings: `.GLOBL`.

```
EXPORT name
```

Expose a symbol to other object modules.
LWASM directive. Its effects take place during assembly.
Reading: Astle, LWTools 4.25, ch. 3.

EXTDEP

```
name EXTDEP
```

Mark an external symbol as a link dependency without emitting storage for it.
LWASM directive. Its effects take place during assembly.
Reading: Astle, LWTools 4.25, ch. 3.

EXTERN

Other spellings: `EXTERNAL`, `IMPORT`.

```
name EXTERN
```

Declare a name defined in another object module.
LWASM directive. Its effects take place during assembly.
Reading: Astle, LWTools 4.25, ch. 3.

FCB

Other spellings: `.BYTE`, `.DB`.

```
FCB byte[,byte…]
```

Emit each expression as one byte. The listing shows data at the current location.
LWASM directive. Its effects take place during assembly.
Reading: Astle, LWTools 4.25, ch. 3.

FCC

Other spellings: `.ASCII`, `.STR`.

```
FCC /text/
```

Emit character bytes between matching delimiters.
LWASM directive. Its effects take place during assembly.
Reading: Astle, LWTools 4.25, ch. 3.

FCN

Other spellings: `FCZ`, `.ASCIZ`, `.STRZ`.

```
FCN /text/
```

Emit a string followed by a zero byte.
LWASM directive. Its effects take place during assembly.
Reading: Astle, LWTools 4.25, ch. 3.

FCS

Other spellings: `.ASCIS`, `.STRS`.

```
FCS /text/
```

Emit a string with the high bit set in its last character.
LWASM directive. Its effects take place during assembly.
Reading: Astle, LWTools 4.25, ch. 3.

FDB

Other spellings: `.WORD`, `.DW`.

```
FDB word[,word…]
```

Emit 16-bit words, high byte first.

LWASM directive. Its effects take place during assembly.
Reading: Astle, LWTools 4.25, ch. 3.

FDBS

```
FDBS word[,word…]
```

Emit 16-bit values low byte first, opposite to the processor's normal word order.
LWASM directive. Its effects take place during assembly.
Reading: Astle, LWTools 4.25, ch. 3.

FILL

```
FILL byte,count
```

Emit the given byte repeatedly.
LWASM directive. Its effects take place during assembly.
Reading: Astle, LWTools 4.25, ch. 3.

FQB

Other spellings: `.4BYTE`, `.QUAD`.

```
FQB value
```

Emit 32-bit quantities, high byte first.
LWASM directive. Its effects take place during assembly.
Reading: Astle, LWTools 4.25, ch. 3.

IFDEF

```
IFDEF symbol
```

Test whether a symbol has already been defined.
LWASM directive. Its effects take place during assembly.
Reading: Astle, LWTools 4.25, ch. 3.

IFEQ

```
IFEQ expression
```

Include the following block when the expression equals zero.
LWASM directive. Its effects take place during assembly.
Reading: Astle, LWTools 4.25, ch. 3.

IFGE

```
IFGE expression
```

Include the block when the expression is zero or positive.
LWASM directive. Its effects take place during assembly.
Reading: Astle, LWTools 4.25, ch. 3.

IFGT

```
IFGT expression
```

Include the block when the expression is positive.
LWASM directive. Its effects take place during assembly.
Reading: Astle, LWTools 4.25, ch. 3.

IFLE

```
IFLE expression
```

Include the block when the expression is zero or negative.
LWASM directive. Its effects take place during assembly.
Reading: Astle, LWTools 4.25, ch. 3.

IFLT

```
IFLT expression
```

Include the block when the expression is negative.
LWASM directive. Its effects take place during assembly.
Reading: Astle, LWTools 4.25, ch. 3.

IFNDEF

```
IFNDEF symbol
```

Test whether a symbol is still undefined.
LWASM directive. Its effects take place during assembly.
Reading: Astle, LWTools 4.25, ch. 3.

IFNE

Other spellings: `IF`.

```
IFNE expression
```

Include the block when the expression is nonzero.
LWASM directive. Its effects take place during assembly.
Reading: Astle, LWTools 4.25, ch. 3.

IFP1

```
IFP1
```

Legacy pass-number conditional. LWASM warns that it cannot reproduce pass-one assembly semantics.
LWASM directive. Its effects take place during assembly.
Reading: Astle, LWTools 4.25, ch. 3.

IFP2

```
IFP2
```

Legacy pass-number conditional. LWASM warns that it cannot reproduce pass-two assembly semantics.
LWASM directive. Its effects take place during assembly.
Reading: Astle, LWTools 4.25, ch. 3.

IFPRAGMA

Other spellings: `IFOPT`.

```
IFPRAGMA option
```

Include a block according to the current pragma setting.
LWASM directive. Its effects take place during assembly.
Reading: Astle, LWTools 4.25, ch. 3.

IFSTR

```
IFSTR comparison,arguments
```

Include a block according to the specified string comparison.
LWASM directive. Its effects take place during assembly.
Reading: Astle, LWTools 4.25, ch. 3.

INCLUDE

Other spellings: `INCL`, `LIB`, `USE`.

```
INCLUDE filename
```

Read another source file at this location.
LWASM directive. Its effects take place during assembly.
Reading: Astle, LWTools 4.25, ch. 3.

INCLUDEBIN

```
INCLUDEBIN filename
```

Copy bytes from a binary file into the assembly output.
LWASM directive. Its effects take place during assembly.
Reading: Astle, LWTools 4.25, ch. 3.

INCLUDESTR

```
INCLUDESTR string
```

Treat the contents of the given string as additional assembler source.
LWASM directive. Its effects take place during assembly.
Reading: Astle, LWTools 4.25, ch. 3.

MACRO

Other spellings: `MACR`.

```
name MACRO
```

Begin defining a sequence of source lines for later expansion.
LWASM directive. Its effects take place during assembly.
Reading: Astle, LWTools 4.25, ch. 3.

ORG

```
ORG address
```

Set the address of the following output. In DECB an origin determines where a data block loads.
LWASM directive. Its effects take place during assembly.
Reading: Astle, LWTools 4.25, ch. 3.

PHASE

```
PHASE expression
```

Give following code a different logical address for labels and expressions while retaining its output placement.
LWASM directive. Its effects take place during assembly.
Reading: Astle, LWTools 4.25, ch. 3.

PRAGMA

Other spellings: `*PRAGMA`, `**PRAGMA`, `OPT`.

```
PRAGMA option
```

Change a LWASM dialect option. A target-changing pragma affects subsequent assembly.
LWASM directive. Its effects take place during assembly.
Reading: Astle, LWTools 4.25, ch. 3.

REORG

```
REORG
```

Restore the location associated with the previous ORG.
LWASM directive. Its effects take place during assembly.
Reading: Astle, LWTools 4.25, ch. 3.

RMB

Other spellings: `.BLKB`, `.DS`, `.RS`.

```
RMB count
```

Advance the location counter by a number of bytes. Use ZMB when the load image must contain zeros.
LWASM directive. Its effects take place during assembly.
Reading: Astle, LWTools 4.25, ch. 3.

RMD

Other spellings: `RMW`.

```
RMD count
```

Reserve 16-bit words.
LWASM directive. Its effects take place during assembly.
Reading: Astle, LWTools 4.25, ch. 3.

RMQ

```
RMQ count
```

Reserve 32-bit values.
LWASM directive. Its effects take place during assembly.
Reading: Astle, LWTools 4.25, ch. 3.

SECTION

Other spellings: `SECT`, `.AREA`.

```
SECTION name
```

Begin a section in an object module. The linker determines final placement.
LWASM directive. Its effects take place during assembly.
Reading: Astle, LWTools 4.25, ch. 3.

SET

```
name SET expression
```

Assign a symbol a value that later SET statements may change.
LWASM directive. Its effects take place during assembly.
Reading: Astle, LWTools 4.25, ch. 3.

SETDP

```
SETDP page
```

Tell the assembler the assumed direct page. Guest software must establish the matching DP value.
LWASM directive. Its effects take place during assembly.
Reading: Astle, LWTools 4.25, ch. 3.

SETSTR

```
SETSTR name=string
```

Set an assembler string variable.
LWASM directive. Its effects take place during assembly.
Reading: Astle, LWTools 4.25, ch. 3.

STRUCT

```
name STRUCT
```

Begin a structure layout. Field labels describe offsets in its instances.
LWASM directive. Its effects take place during assembly.
Reading: Astle, LWTools 4.25, ch. 3.

WARNING

Other spellings: `MSG`.

```
WARNING message
```

Report a warning with the supplied text.
LWASM directive. Its effects take place during assembly.
Reading: Astle, LWTools 4.25, ch. 3.

ZMB

Other spellings: `BSZ`, `FZB`.

```
ZMB count
```

Emit the requested number of zero bytes.
LWASM directive. Its effects take place during assembly.
Reading: Astle, LWTools 4.25, ch. 3.

ZMD

```
ZMD count
```

Emit zero-filled words.
LWASM directive. Its effects take place during assembly.
Reading: Astle, LWTools 4.25, ch. 3.

ZMQ

```
ZMQ count
```

Emit zero-filled four-byte quantities.
LWASM directive. Its effects take place during assembly.
Reading: Astle, LWTools 4.25, ch. 3.

Dialect, expansion and undocumented spellings

Check the listing before counting debugger steps. A spelling can denote an actual HD6309 operation in one build and a sequence emitted by an assembler pragma in another. The names grouped below describe separate assembler features. The instruction directory identifies the corresponding hardware operations.

ASM-09 dialect

Spellings in this group: `IFC`, `IFNC`, `EXITM`, `EXIT`.
The ASM-09 compatibility dialect adds string equality and inequality conditionals and an early macro exit.
Reading: Astle, LWTools 4.25, ch. 3.

Assembler conveniences

Spellings in this group: `ASRQ`, `CLRQ`, `COMQ`, `LSLE`, `LSLF`, `LSLQ`, `LSRQ`, `NEGE`, `NEGF`, `NEGW`, `NEGQ`, `TSTQ`.

LWASM can expand convenience operations into several instructions when the corresponding pragma is active. The listing gives the actual expansion. A 6809 emulation of CLRD, for example, executes CLRA and CLRB separately.

6809 expansions include ASLD, ASRD, CLRD, COMD, LSLD, LSRD, NEGD and TSTD. These names also identify real HD6309 operations. Additional 6309 convenience spellings include ASRQ, CLRQ, COMQ, LSLE, LSLF, LSLQ, LSRQ, NEGE, NEGF, NEGW, NEGQ and TSTQ. MC6800 compatibility is a separate optional dialect.

Reading: Astle, LWTools 4.25, ch. 3.

Compatibility directives

Spellings in this group: `NAM`, `PAG`, `PAGE`, `SPC`, `TTL`, `.BANK`, `.MODULE`.

LWASM accepts these listing and tool-compatibility directives without generating code.

Reading: Astle, LWTools 4.25, ch. 3.

MC6800 compatibility

Spellings in this group: `ABA`, `CBA`, `CLC`, `CLF`, `CLI`, `CLIF`, `CLV`, `CPX`, `DES`, `DEX`, `DEY`, `INS`, `INX`, `INY`, `SBA`, `SEC`, `SEF`, `SEI`, `SEIF`, `SEV`, `TAB`, `TAP`, `TBA`, `TPA`, `TSX`, `TXS`, `WAI`.

These optional LWASM spellings expand to 6809 encodings with the 6800compat pragma. Expansion can require several instructions. Inspect the listing when stepping.

Reading: Astle, LWTools 4.25, ch. 3.

OS-9 output directives

Spellings in this group: `OS9`, `MOD`, `EMOD`.

OS9 encodes SWI2 followed by a service byte for OS-9 or object output. MOD and EMOD frame an OS-9 module. Assembly Development runs standalone Disk BASIC block images.

Reading: Astle, LWTools 4.25, ch. 3.

Undocumented and emulator opcodes

Spellings in this group: `HCF`, `RHF`, `RESET`, `HALT`, `BREAK`, `LOG`.

LWASM lists HCF, RHF and RESET for certain 6809 encodings. Behavior outside the documented instruction set can differ by chip. BREAK and LOG are emulator extensions. VCC Plus's Mac decoder also excludes the optional HALT/BREAK encodings.

Reading: Atkinson (2009), p. 153 · Astle, LWTools 4.25, ch. 3 · VCC Plus, Native Debugger, disassembly.

Keep a workbench notebook

Appendix C

Keep a record short enough that you will actually use it. For a surprising byte, write down the guest CPU and mode, the source label, the PC of the *next* instruction, the address space and the byte you expected. If you share only a hexadecimal address, another reader can inspect precisely the wrong thing.

Read the original assembler manual

The complete LWTools 4.25 manual is available offline alongside this book, with its original source, PDF and license notices. Consult it for LWASM syntax and LWLINK scripts; the bundled command-line help also records installed options and VCC Plus extensions. **Ask the Manuals** and project MCP manual tools search these references together with the teaching books. The complete CMOC manual is useful when linking assembly with C.

Save the examples

Question to test	File to save	Setup and destination
Did the message survive?	dispatch.asm	Either CPU, Disk BASIC. Change the last letter in a copy.
Which physical block backs this window?	bankwalk.asm	Use the disposable setup in chapter 4.
Which page supplied the pixels?	PAGES.bas	Open in BASIC Development and use the two stops in chapter 5.
What does W count?	dispatch6309.asm	HD6309 guest and target. Stop before and after TFM.
Which mode did the program leave behind?	mode.asm	Fresh HD6309 Disk BASIC, MD=$00 before starting.
How does a software interrupt return?	softirq.asm	Follow the setup and restore sequence in chapter 7.
Can I rebuild and deliver two modules?	Dispatch project	Expand the ZIP into a writable folder, then open the project folder as described in chapter 8.

In the app, selecting an assembly file or project archive opens a save panel. Work on the copy you saved. A BASIC example link opens its own BASIC workspace. In the browser edition you can download the files. With an unmodified checkout the whole shelf also opens locally from **docs/manuals/index.html**.

Choose the right address book

A 6809 address runs from $0000 to $FFFF. It answers “where in the guest CPU’s current view?” For a RAM location, the enabled MMU, active task and block select the backing page. The physical-video base answers a different question about what the video hardware displays. A ROM overlay can change what the CPU reads in part of that space. Addresses near the top can reach device registers and vectors; a read or write performed by guest code may have a hardware side effect.

- To inspect a label in your assembled listing, begin with **Mapped CPU**. Look up its backing using **MMU** before comparing it to a physical address.
- To ask why a screen didn’t change, inspect **Current Video Page** as well as the chosen address. Record the graphics mode, palette set and whether the address is mapped or physical.

- To call a BASIC service, identify its arguments, registers you need afterward and how you will return. The exercises assume the Disk BASIC environment. CHROUT uses a vector at $A002 and accepts the character in A. The planet program also uses keyboard polling. Read the key-to-color walkthrough for a concrete call site.
- To change a mapping or install an interrupt handler, first save any work you care about. Keep the executing code and its stack reachable, save the state you will modify, and plan a path that restores it. Write down a checkpoint *after* restoration.

Get a known BASIC prompt

If you stepped inside a ROM service, place a stop at the next instruction in your own source and **Continue**. If you stopped just before the final RTS of a direct-run exercise, continue through the return. At the BASIC prompt type **PRINT 2+3**. A fresh answer of 5 distinguishes an interactive return from an old OK still on screen.

An example that ends in a deliberate hold loop keeps executing the loop. Save the source; restart the disposable guest to move on. A program stopped partway through an MMU, mode or vector change still has that intermediate state. Follow the documented cleanup path if it is reachable; otherwise restart the practice setup. Avoid changing the PC to a convenient-looking address while the stack still belongs to an interrupt handler.

Books and tools behind the experiments

- Motorola, Inc. *MC6809–MC6809E 8-Bit Microprocessor Programming Manual*. M6809PM (AD), March 1981, reprinted May 1983. Chapters 1–2 and Appendix A. Registers, addressing, condition codes and the interrupt frame.
- Tepolt, Laurence A. *Assembly Language Programming for the CoCo 3*. TEPCO, 1987. Figures 3-4 and 4-1, pp. 20 and 23. Historical context for the relationship between processor and video addressing in the CoCo 3.
- Atkinson, Darren. *Motorola 6809 and Hitachi 6309 Programmer's Reference*. 2009. Parts I and II. Comparative instruction, addressing and mode descriptions.
- Burke, Chris. *The 6309 Book: Inside the 6309 Microprocessor*. Second edition. Burke & Burke, 1993. Extended registers and native-mode behavior.
- Barden, William, Jr. *TRS-80 Color Computer Assembly Language Programming*. Radio Shack, 1983. Subroutines, addressing and machine-language practice on the Color Computer.
- Astle, William, and LWTools contributors. *LW Tool Chain*. Manual distributed with LWTools 4.25. LWASM and LWLINK chapters. Source syntax, object sections, linking and output formats.

The Color Computer Archive preserves historical documentation that informed this book. The lessons and lab programs here were written for VCC Plus.

Original VCC Plus prose and instructional images © 2026 Brian K. Holman. The Hitachi badge on the cover identifies the processor. See the application's **Licenses & Notices** and the library's artwork credits.

Publication and artwork notices

Original VCC Plus prose and instructional images © 2026 Brian K. Holman. Original component and example notices remain with their respective material.

Except where otherwise identified, original artwork owned by Brian K. Holman is © 2026 Brian K. Holman. All rights reserved.

Permission is granted to use and redistribute this artwork unchanged with VCC Plus, including modified versions of its GPL-licensed software. Other reproduction, modification or redistribution requires prior written permission, except as permitted by applicable law. Third-party materials remain subject to their respective licenses and rights.

The CoCo 3 and CM-8 monitor presentation in VCC Plus's skeuomorphic System view was inspired by SockMaster's "pic of CoCo 3 and CM-8 monitor," published on CoCopedia. VCC Plus's artwork was designed and rendered independently. The remaining hardware illustrations were independently designed using historical photographs and renderings as references.

Read this book online

CMOC license notice · LWTools COPYING notice