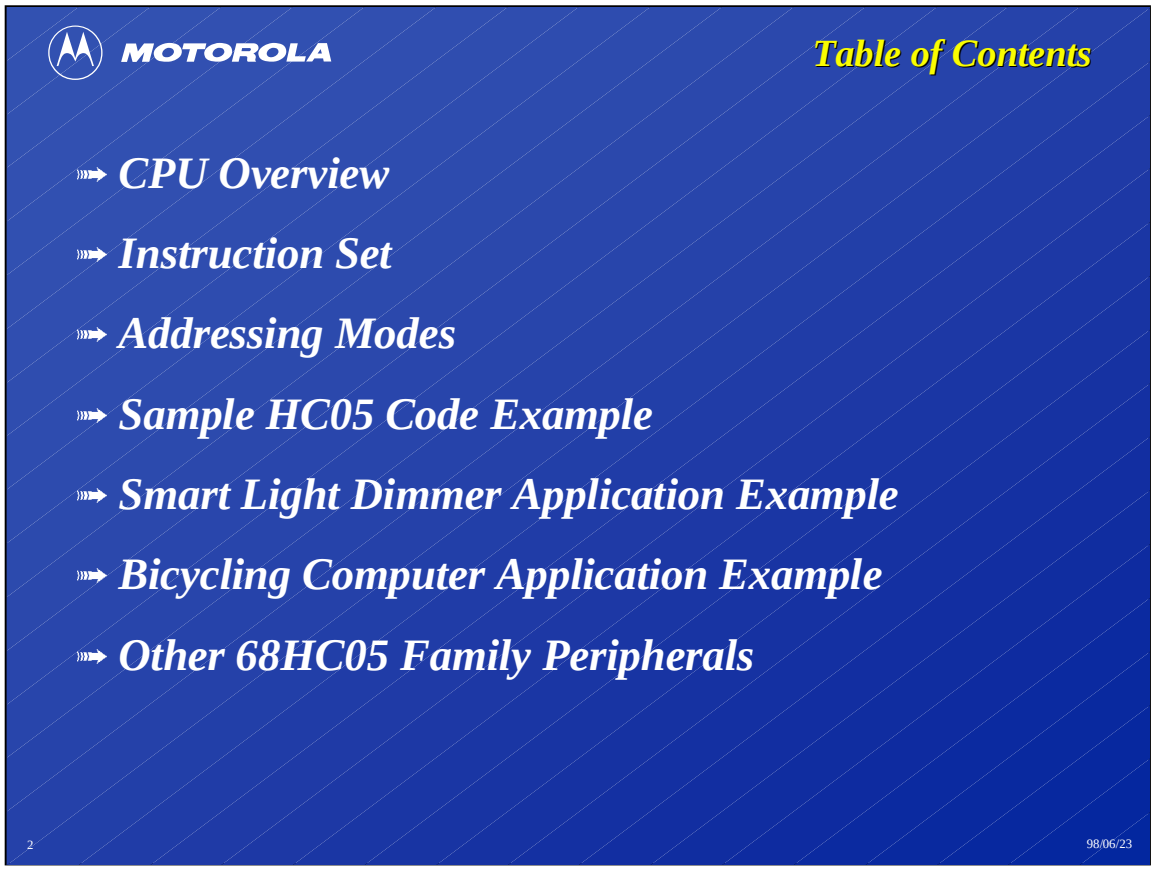
An Introduction to Motorola's 68HC05 Family of 8-Bit Microcontrollers

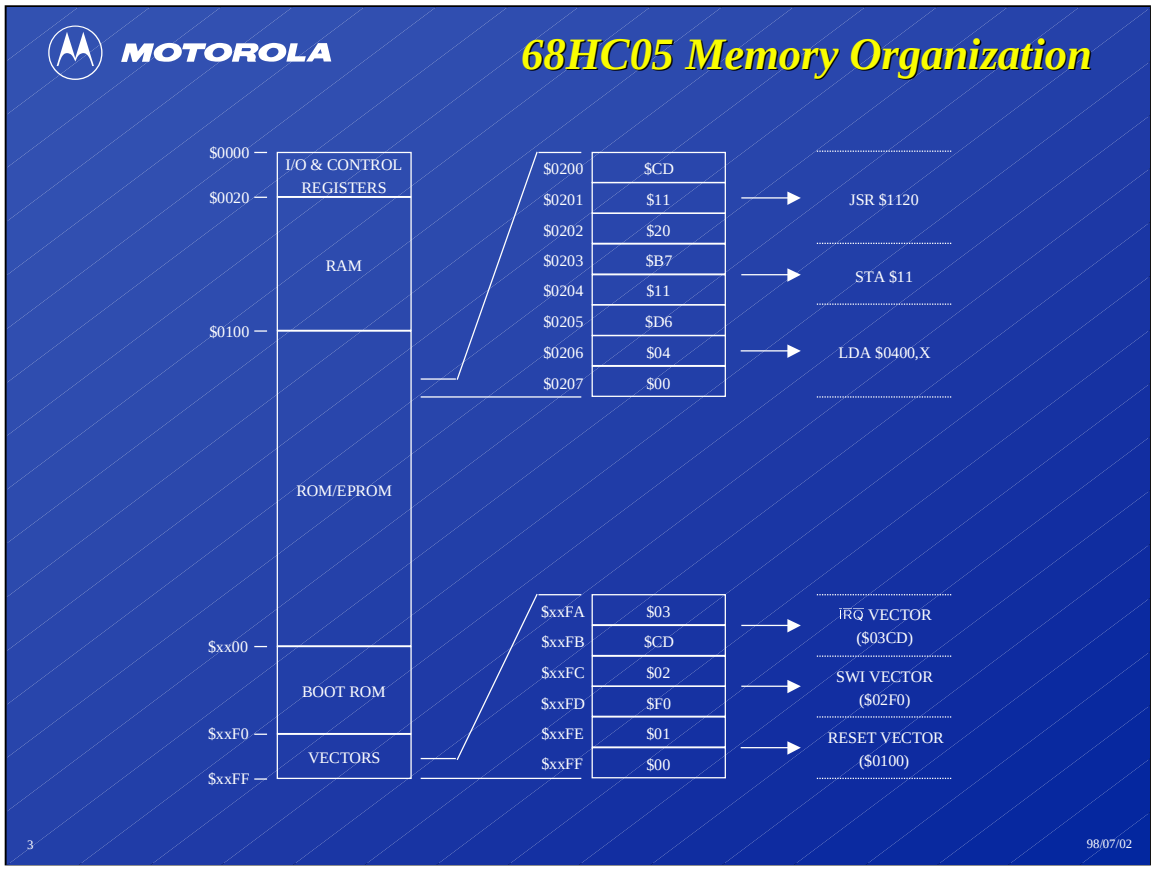
This presentation is a self paced tutorial of the 68HC05 family of 8-bit microcontrollers.



The Tutorial starts with an architectural overview of the 68HC05 central processor unit (CPU). It covers memory organization, the CPU programmer's model, stack pointer operation, and the 68HC05 instruction set and its addressing modes. Once learned, this knowledge is applicable to all 68HC05 devices, because they all use the same CPU.

In the second part of this tutorial, two sample applications illustrate the use of some common 68HC05 peripherals. One of these is a smart light dimmer in which the very low cost MC68HC705KJ1 provides features not available on conventional electro-mechanical dimmers. The other is a cycling computer that uses the MC68HC705P6A to monitor rider heart rate, temperature, humidity, speed, and distance traveled.

Other common 68HC05 peripherals are covered in the third and final section of this tutorial. These provide some of the communication, timing, and display features of embedded control applications not illustrated in the previous examples.

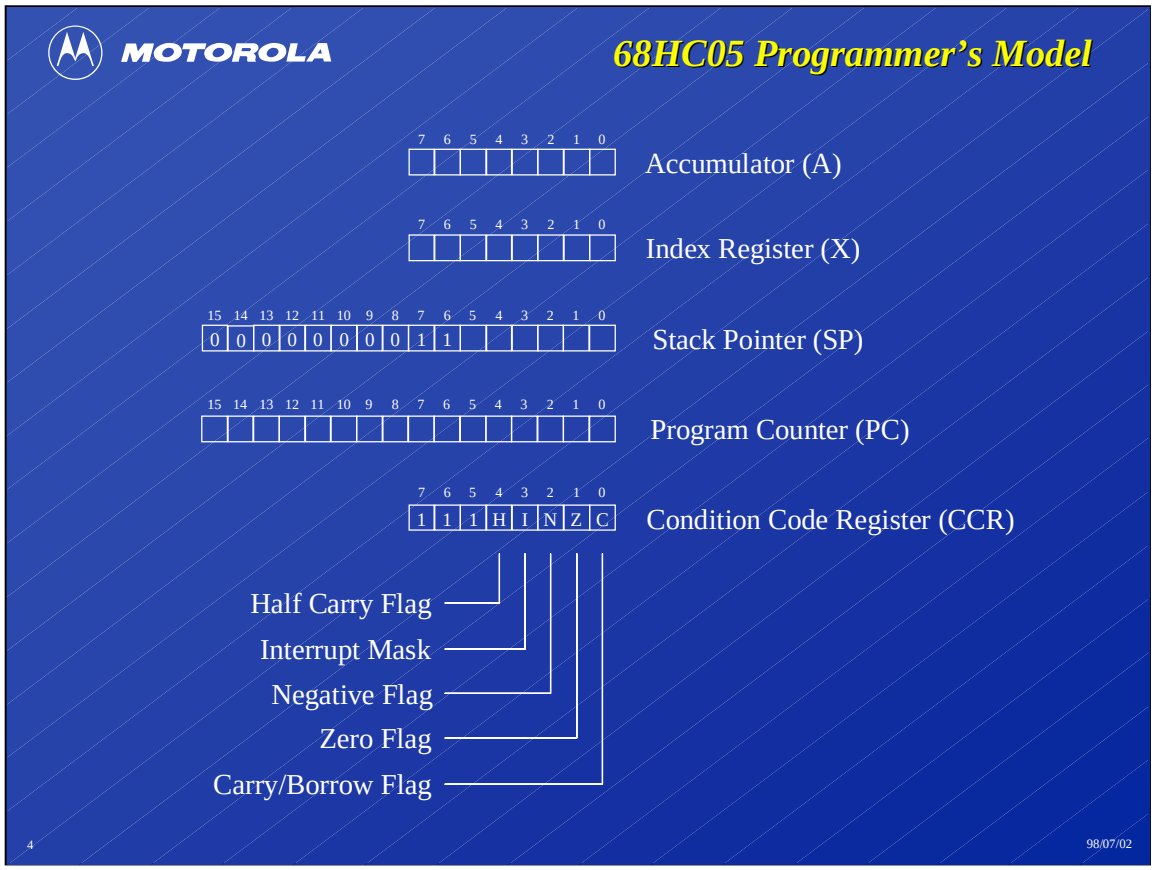


The 68HC05 is a Von Neumann computer. All storage and input/output resources are memory mapped into a single linear address space.

The memory map is organized as bytes, begins at $0000, and ends at $xxFF, where, depending on factors like ROM or EPROM array size, ‘xx’ is $03, $07, $0F, $1F, $3F, or $7F.

The address space on a 68HC05 device is usually sized just large enough to contain the integrated ROM or EPROM, RAM, and control registers. The MC68HC705KJ1, for example, has a 2-Kbyte (‘xx’ equals $07) memory map containing 1240 bytes of EPROM, 64 bytes of RAM, and 14 bytes of other processor accessible resources.

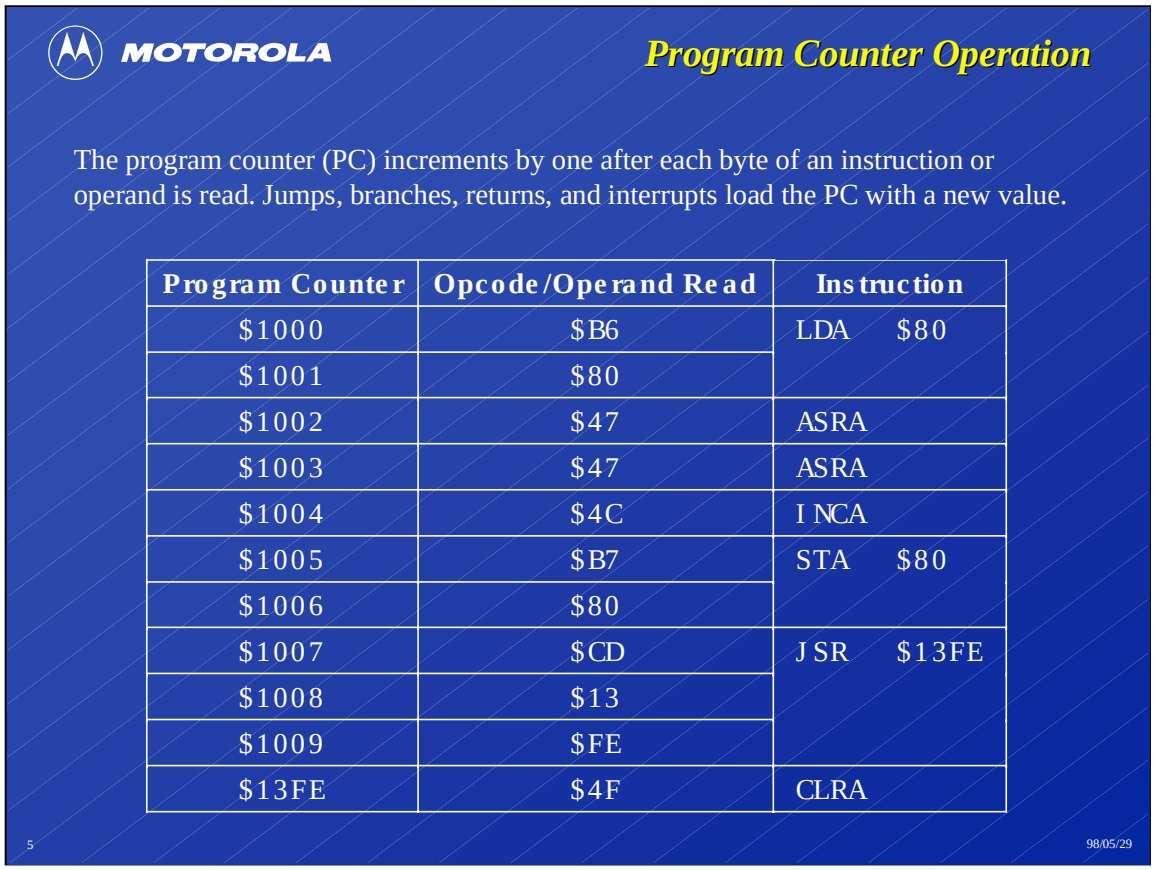
The 68HC05 is also a Big Endian machine. A 16-bit piece of data, or word, is stored in memory with its high byte at address N and its low byte at address N + 1. This ordering applies whether the word is part of an assembled instruction (such as an index register offset), a return address residing on the stack, or the address of a service routine contained in an interrupt vector. See the examples above.



The 68HC05 central processor unit (CPU) consists of an accumulator (A), an index register (X), a stack pointer (SP), a program counter (PC), and a condition code register (CCR).

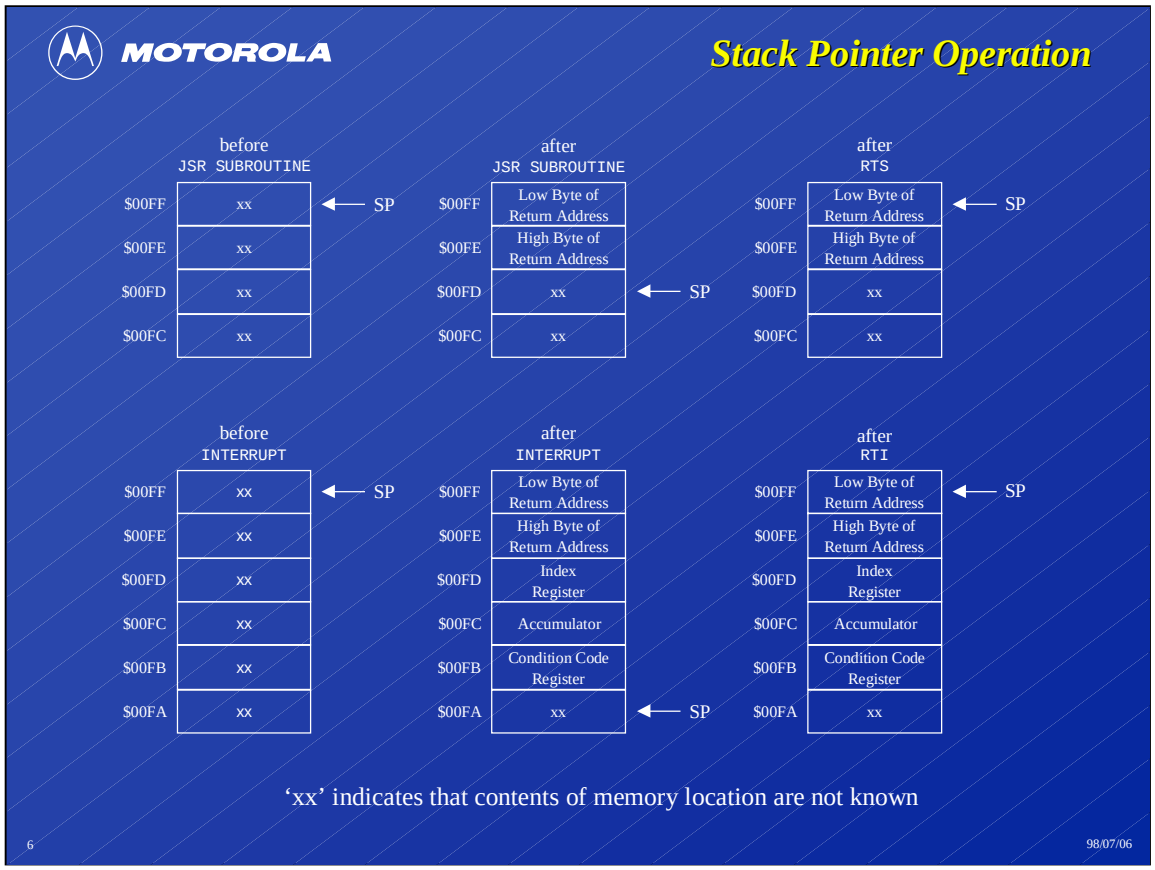
Data can be read from memory into the accumulator and the index register. Likewise, data can be written into memory from the accumulator and the index register. The accumulator, however, is the only register upon which arithmetic and combinatorial logic operations can be performed. Only the index register can provide user-generated effective addresses for operands read into or written from the accumulator. Both the accumulator and the index register support bit-wise shift and rotate operations.

Most accumulator, index register, and memory operations affect status flags in the condition code register. The carry bit is set when an arithmetic carry or borrow has taken place. Shift and rotate operations also move bits through the carry bit. The zero flag is set when all bits of an operand or result are zero. Likewise, the negative flag is set when the MSB of an operand or result is one. The I bit masks interrupts and is set during interrupt processing. Software can also set and clear the I bit. To facilitate BCD arithmetic, the half carry flag is set when a carry from bit three to bit four of an operand occurs as the result of an ADC or ADD instruction.



The program counter (PC) points to the address of the current instruction. It advances one byte at a time as instructions and operands are fetched during the course of normal program execution.

Jumps, branches, returns, and interrupts can change program flow and, thus, the normal procession of the PC. The address of a jump or branch is specified by the instruction's addressing mode (i.e. a relative offset for a branch). Return from interrupt (RTI) and from subroutine (RTS) instructions load the PC with a return address stored on the stack. Interrupts load the PC with a value specified by the vector associated with the interrupt source.



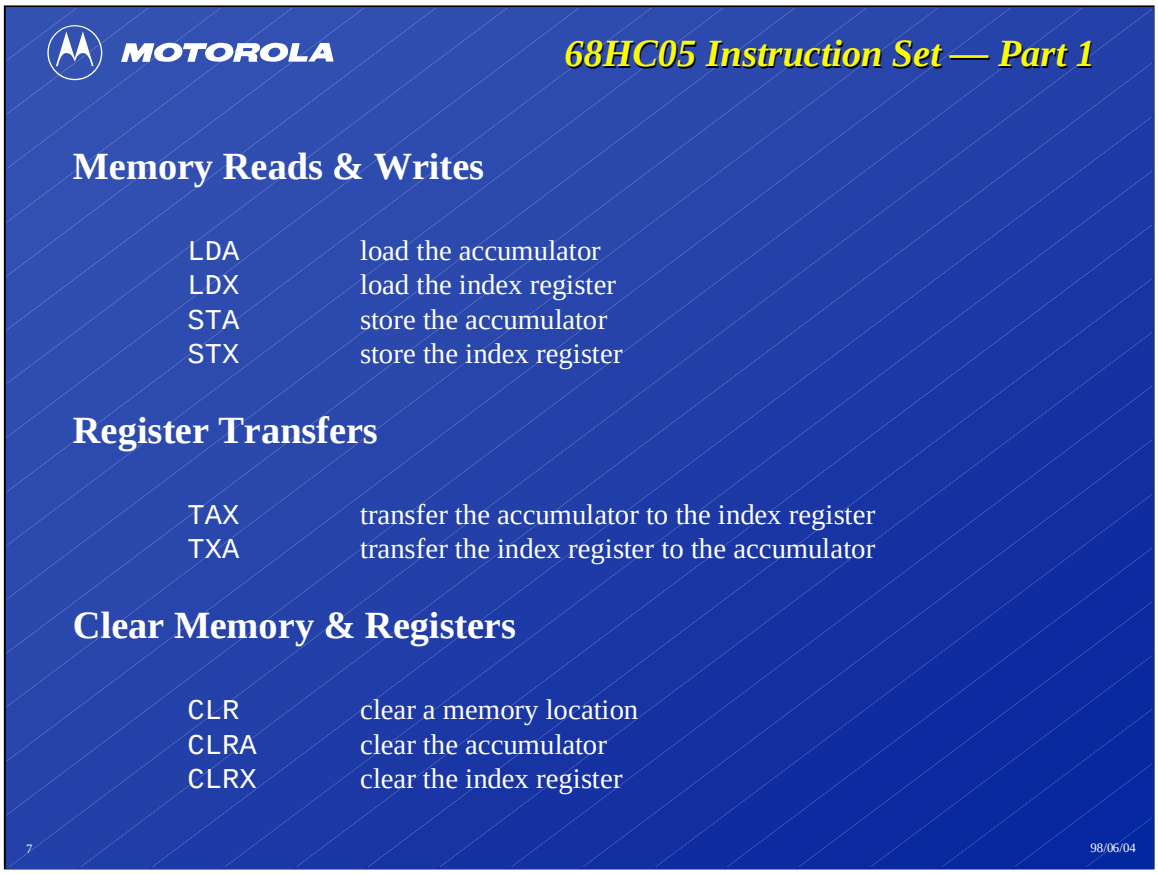
Processor state is saved on the stack when changes in program flow occur. The 68HC05 has a 64-byte hardware-controlled stack. Reset initializes the stack pointer to $00FF; the RSP instruction does the same. No other instructions allow direct user manipulation of the stack.

When a byte is 'pushed' onto the stack, it is written to the location pointed to by the stack pointer, and then the stack pointer is decremented by one. When a byte is 'pulled' from the stack, the stack pointer is incremented by one, and the location pointed to by the stack pointer is then read.

When the stack pointer is at $00C0, the next push will store a byte at $00C0 and roll the stack pointer to $00FF. This is stack pointer overflow. Subsequent pushes will overwrite information stored on the stack from $00FF on down. Similarly, underflow occurs when the stack pointer is at $00FF and the next pull rolls the stack pointer to $00C0 and reads the byte there.

Subroutines called with the BSR and JSR instructions save a two byte return address on the stack. Interrupts save a two byte return address, the index register, the accumulator, and the condition code register.

The examples above show how subroutine calls and interrupts and their respective return instructions affect the stack.



The LDA and LDX instructions read or “load” data from memory. The memory location read can be specified using one of several addressing modes discussed later.

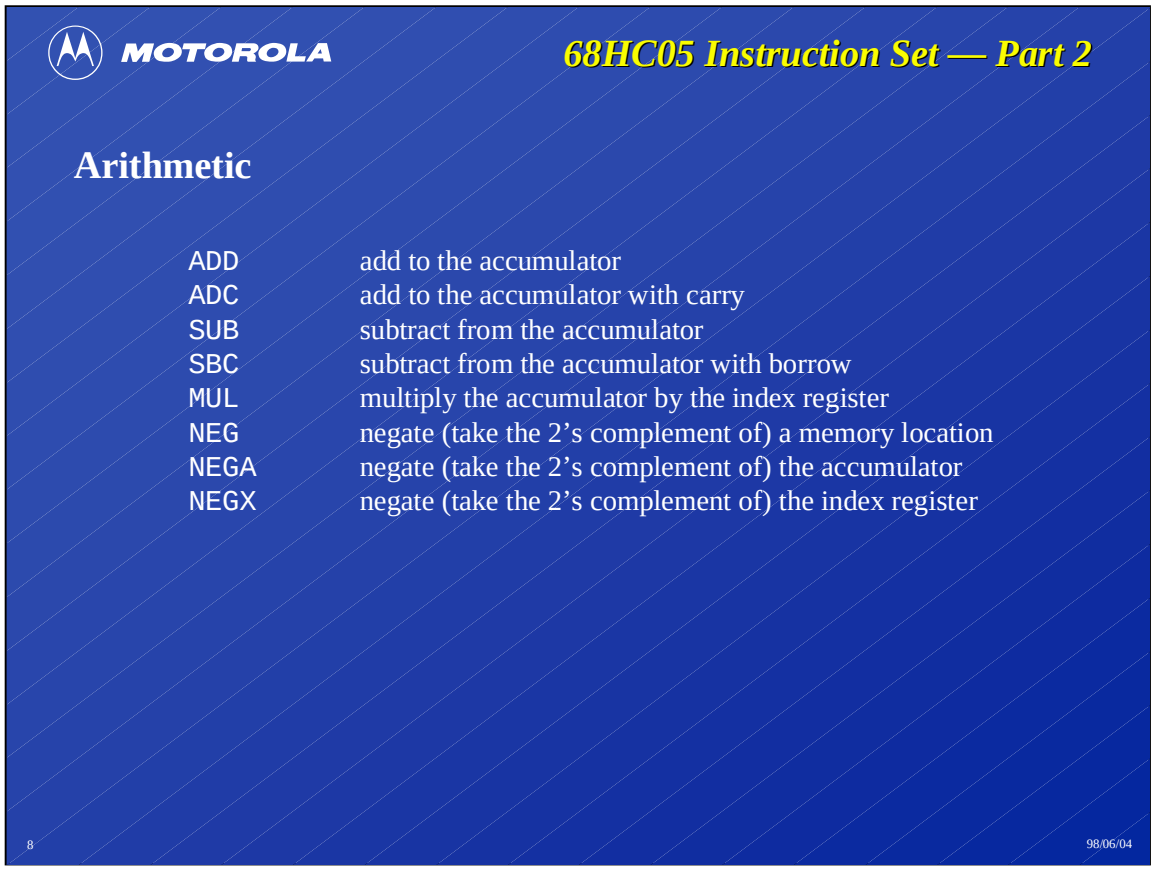
The STA and STX instructions write or “store” data to memory. As with LDA and LDX, several addressing modes are available to specify the desired memory location.

The “clear” instructions are memory efficient ways to zero a memory location or register.

For example, CLR $01 (read as “clear direct address $01”) requires 2 bytes of storage and executes in 5 clock cycles.

On the other hand, LDA #$00 (read as “load accumulator with immediate value $00”) and STA $01 (read as “store accumulator to direct address $01”) require 4 bytes of storage and execute in 6 cycles.

Likewise, CLRA and CLRX each require only 1 byte of storage, versus 2 each for LDA #$00 (read as “load accumulator with immediate value $00”) and LDX #$00 (read as “load index register with immediate value $00”).



The ADD and SUB instructions, respectively, add or subtract a byte to or from the accumulator. ADC and SBC do the same but with a carry or borrow if the condition code register carry bit (C) is set.

The MUL instruction performs an unsigned multiply of the index register (X) and the accumulator (A). The result is stored with the upper byte in X and the lower byte in A.

NEG, NEGA, and NEGX take the two's complement of a memory location, the accumulator, or the index register. The two's complement of a number is zero minus that number.

Decrement & Increment Memory & Registers

INC	increment a memory location by one
INCA	increment the accumulator by one
INCX	increment the index register by one
DEC	decrement a memory location by one
DECA	decrement the accumulator by one
DECX	decrement the index register by one

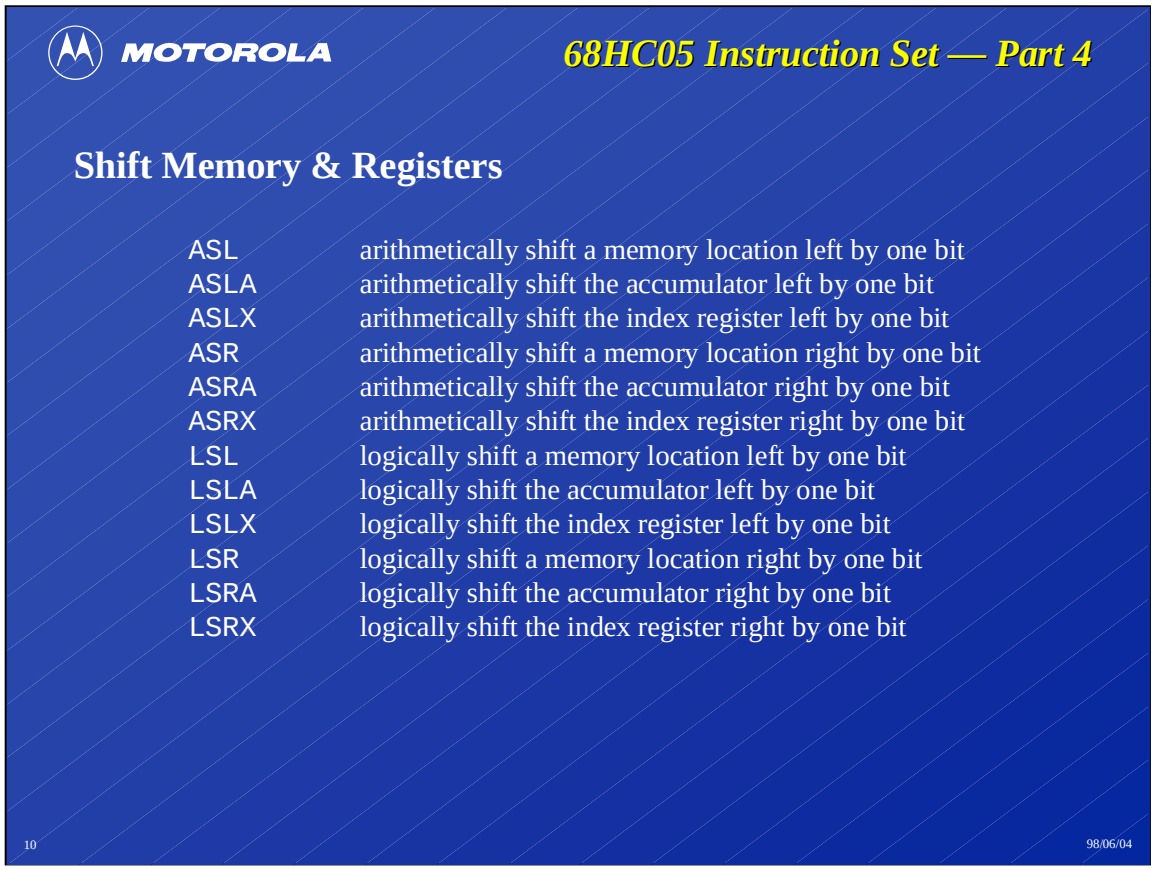
Boolean Logic

AND	logical AND of the accumulator and an operand
ORA	inclusive OR of the accumulator and an operand
EOR	exclusive OR of the accumulator and an operand
COM	take the one’s complement of (invert) a memory location
COMA	take the one’s complement of (invert) the accumulator
COMX	take the one’s complement of (invert) the index register

In addition to the ADD/ADC and SUB/SBC instructions, the 68HC05 has dedicated increment and decrement instructions to add one to or subtract one from a memory location, the accumulator, or the index register.

The AND, ORA, and EOR instructions, respectively, take the logical AND, inclusive OR, and exclusive OR of the accumulator and an operand and store the result in the accumulator.

COM, COMA, and COMX take the one’s complement of a memory location, the accumulator, or the index register. The one’s complement of a number is simply the binary inversion of its bits.



These instructions allow left and right shifts of memory locations, the accumulator, and the index register.

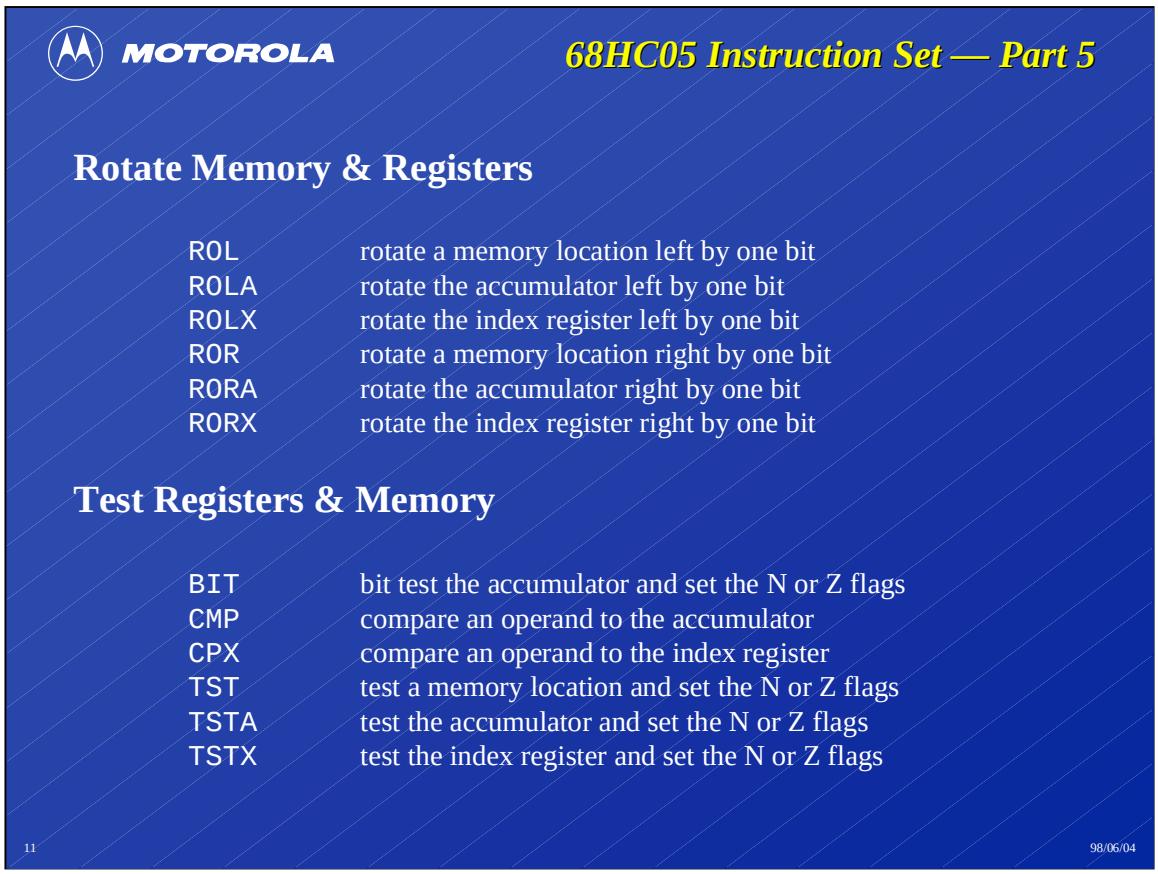
Arithmetic shift left and logical shift left are the same operation, and the ASL, ASLA, and ASLX opcodes are the same as those for LSL, LSLA, and LSLX. In either case, the operand is shifted one bit to the left, with the MSB moving into the condition code register carry (C) bit and a zero moving into the LSB.

Assemblers for the 68HC05 will recognize both the arithmetic shift left and logical shift left instructions and assemble them to the same opcodes.

Arithmetic shift right and logical shift right are two different operations. In both cases, though, the operand is shifted one bit to the right, with the LSB moving into the carry bit.

When an operand is arithmetically shifted right, the MSB remains unchanged. This preserves the sign of the operand. The arithmetic right shift of $80 (-128 in decimal), for example, is $C0 (-64 in decimal).

The MSB of an operand logically shifted right is always set to zero.



Rotate instructions operate in a fashion similar to shift instructions. A rotated operand is first shifted in the direction indicated by the instruction. The empty bit created by the shift (LSB for left and MSB for right) takes the state of the condition code register carry (C) bit, and the carry bit takes the state of the bit shifted out of the operand (MSB for left and LSB for right).

The BIT instruction sets the condition code register negative (N) or (Z) flags based on the logical AND of the accumulator and an operand. The result of this logical AND is discarded.

The compare instructions allow subsequent branch operations to determine if an argument is equal to, not equal to, greater than, greater than or equal to, less than, or less than or equal to the value in the designated register.

A register and an operand are “compared” by non-destructively subtracting the operand from the register and setting the condition code register C, N, or Z bits.

The test instructions set the negative and zero flags by non-destructively subtracting zero from a memory location (TST), the accumulator (TSTA), or the index register (TSTX).

Branches on Condition Code Register Bits

BCC	branch if carry clear (C = 0)
BCS	branch if carry set (C = 1)
BEQ	branch if equal (Z = 0)
BNE	branch if not equal (Z = 1)
BHCC	branch if half carry clear (H = 0)
BHCS	branch if half carry set (H = 1)
BHI	branch if higher (C or Z = 0)
BHS	branch if higher or same (C = 0)
BLS	branch if lower or same (C or Z = 1)
BLO	branch if lower (C = 1)
BMI	branch if minus (N = 1)
BPL	branch if plus (N = 0)
BMC	branch if interrupts are not masked (I = 0)
BMS	branch if interrupts are masked (I = 1)

This group of branch instructions allow changes in program flow based on the states of various condition code register bits. Branch instructions use the relative addressing mode and can move backward 128 bytes or forward 127 bytes in memory from the address of the next instruction.

Notice that the BCS (branch if carry set) and BLO (branch if lower) instructions test the same condition code register bit. These instructions are the same and have the same opcode. The same is also true of BCC (branch if carry clear) and BHS (branch if higher or same).

Assemblers for the 68HC05 will recognize BCS, BLO, BCC, and BHS and assemble them to the appropriate opcodes.

Other Branches

BIH	branch if $\overline{\text{IRQ}}$ pin is high
BIL	branch if $\overline{\text{IRQ}}$ pin is low
BRA	branch always
BRN	branch never
BSR	branch to subroutine and save return address on stack

Single Bit Operations

BCLR	clear the designated memory bit
BSET	set the designated memory bit
BRCLR	branch if the designated memory bit is clear
BRSET	branch if the designated memory bit is set

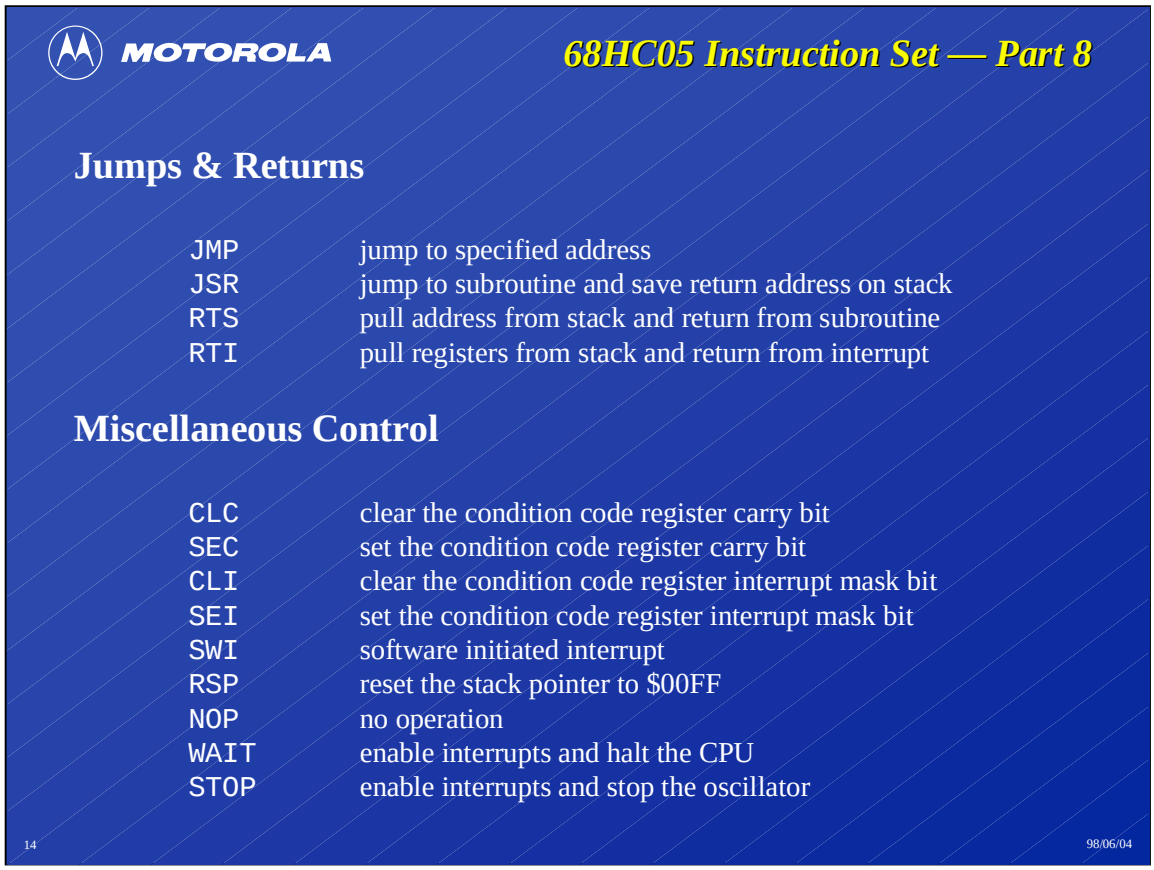
These “other” branch instructions do not examine bits in the condition code register to change program flow. BIH and BIL, in particular, test the state of the actual $\overline{\text{IRQ}}$ pin, not the condition code register interrupt mask (I) bit.

BRN is useful as a three clock cycle no operation instruction. The actual NOP instruction executes in two clock cycles and has a different opcode.

The single bit operations allow setting and clearing of and branching on the set or clear states of single bits in a byte operand. These instructions use the direct addressing mode only and can operand on any bit in the first 256 locations of memory (i.e. internal RAM and peripheral control registers).

The BCLR and BSET instructions each have eight opcodes, one for each bit in a byte. BCLR and BSET require two bytes of storage and execute in five clock cycles which makes them the most memory and time efficient way to clear or set a bit. The same operations using the LDA, AND/ORA, and STA instructions require six bytes and nine clock cycles.

BRCLR and BRSET are similarly efficient, needing only three bytes and five cycles for an operation that would otherwise require six bytes and eight cycles if using a sequence of LDA, AND/ORA, and BEQ/BNE instructions.



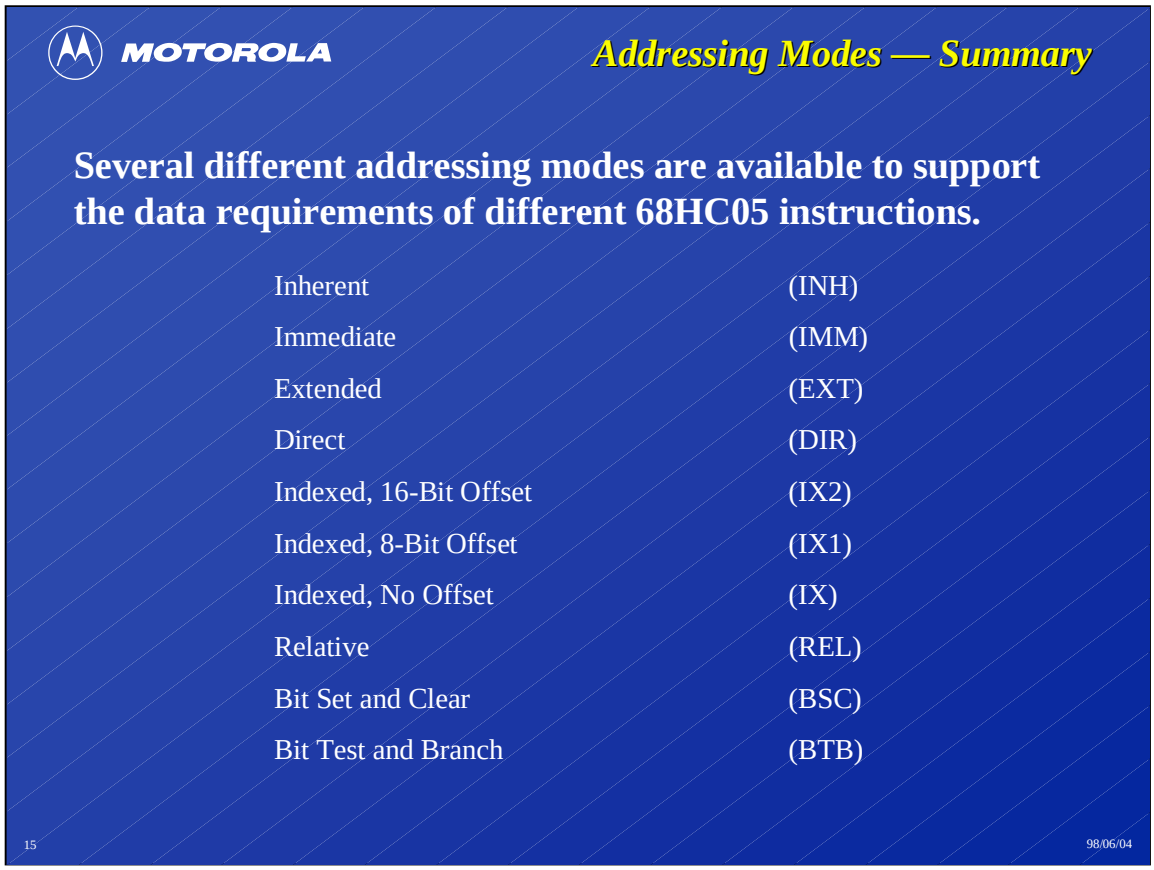
JMP and JSR are analogous to BRA and BSR and allow changes in program flow to be made with 16-bit addresses or index register offsets.

RTS is used to return from a BSR or JSR subroutine call. It pulls only a return address from the stack. RTI is used to return from an interrupt service routine and pulls the condition code register, accumulator, and index register, as well as a return address, from the stack.

The SWI instruction allows an interrupt to be taken under software control, regardless of the state of the condition code register interrupt mask (I) bit. SWI stacks a return address and all of the CPU registers, and jumps to an address specified by its own interrupt vector.

WAIT and STOP allow a 68HC05 device two enter one of two low power modes. WAIT clears the interrupt mask bit (I = 0) and halts the CPU. This allows interrupts from on-chip peripherals or external interrupt pins (e.g. $\overline{\text{IRQ}}$) to re-start execution.

STOP also clears the I bit, but it halts the entire MCU by stopping the clock oscillator. This is the lowest power mode available on a 68HC05 and only interrupts from dedicated external interrupt pins, like $\overline{\text{IRQ}}$, can re-start execution.

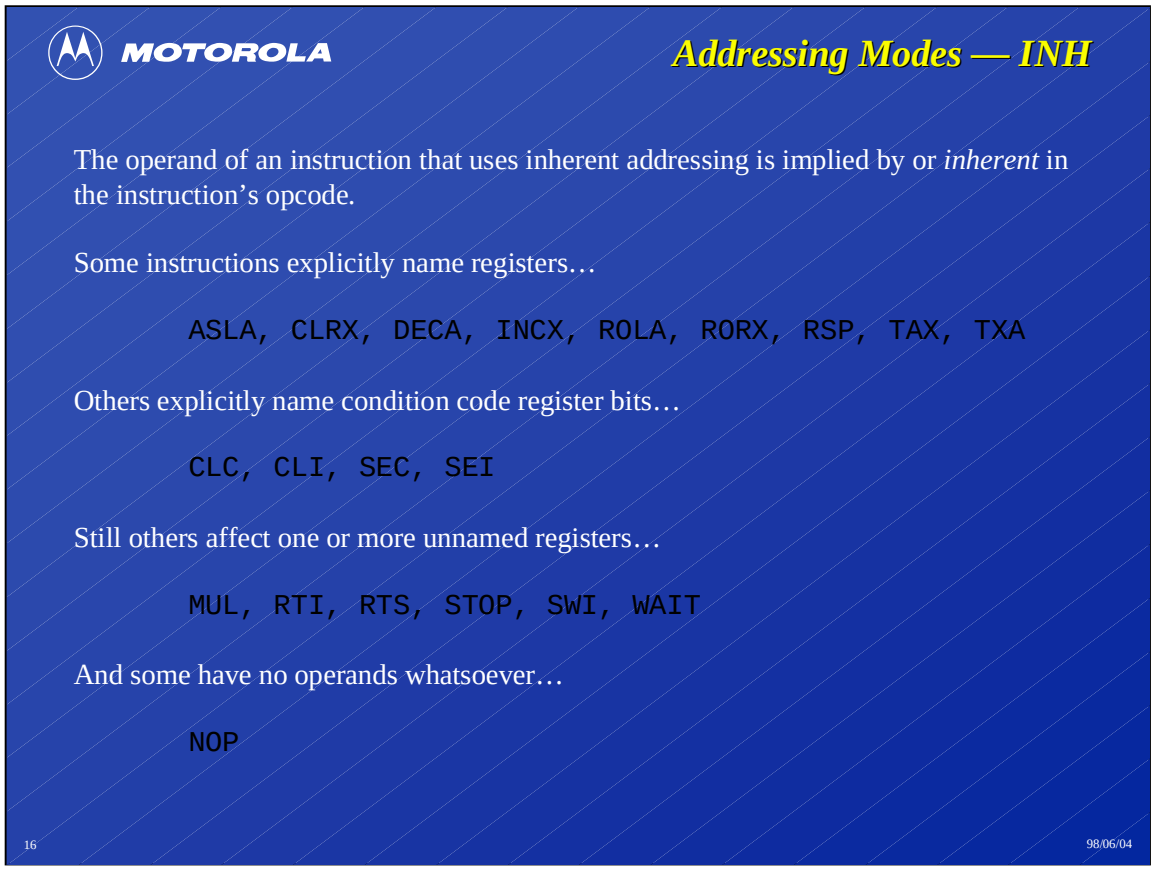


The simple availability of powerful instructions does not alone comprise a good microcontroller architecture. Flexible addressing modes are also needed so that these instructions can efficiently access the different types of data that may be distributed in memory.

In addition to its 65 basic instructions, the 68HC05 has eight addressing modes that determine the source and/or destination of the data upon which these instructions operate.

For practical reasons, no single 68HC05 instruction can use all eight addressing modes. Branches, for example, are relative operations, so it would make no sense for them to use any addressing mode other than relative.

On the other hand, those instructions that must be capable of operating on any memory location and the accumulator or the index register should have the widest selection of addressing modes. Thus, these instructions, like ADD, CMP, and LDA, can use all modes except inherent and relative.

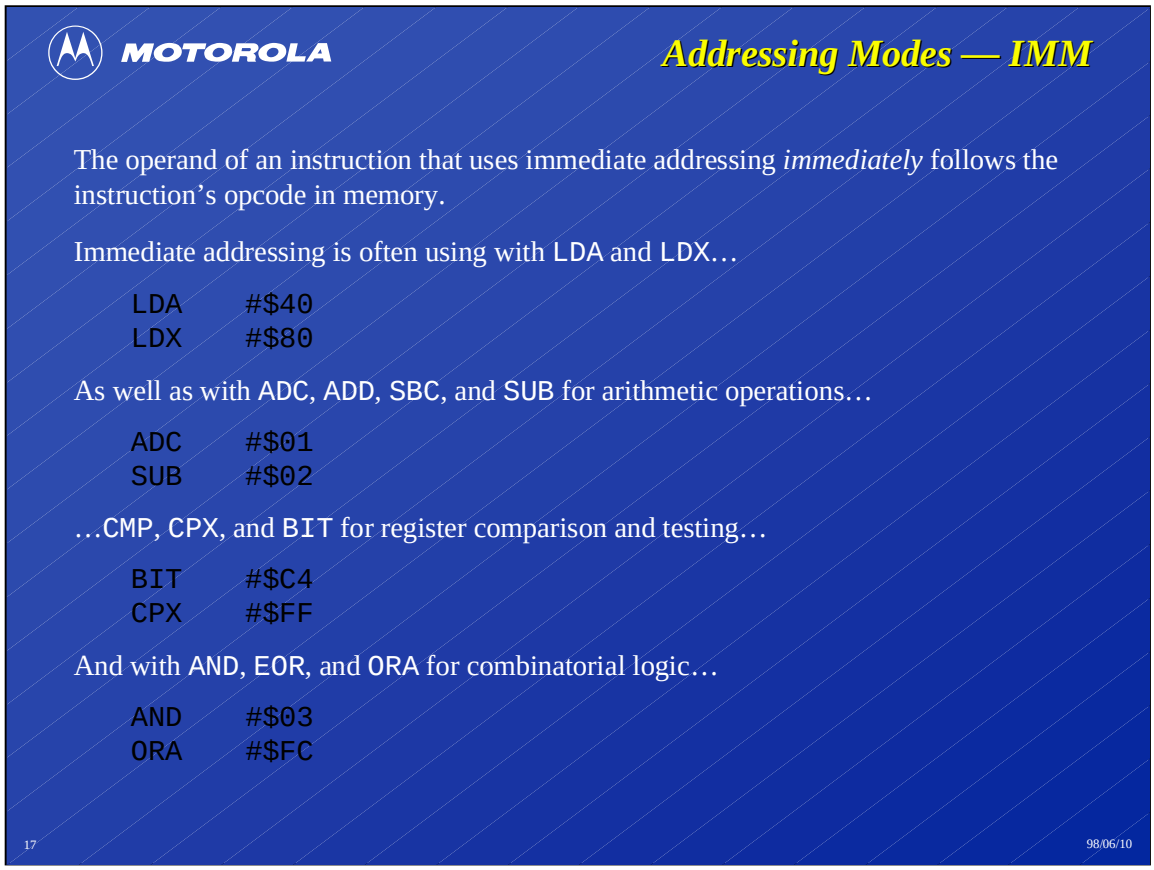


Instructions that use inherent addressing need only one byte of program storage. These are the simplest 68HC05 instructions because they have fixed operands.

Any instruction that operates directly on a named register without an attendant read or write cycle uses inherent addressing. ASLX, CLRA, DECX, INCA, RSP, and TXA are obvious examples. The same also applies to the CLC, CLI, SEC, and SEI instructions that directly manipulate condition code register bits.

Inherent mode instructions that do not explicitly name registers typically affect multiple registers or a single condition code register bit. MUL, for example, uses the accumulator and the index register. SWI and RTI operate on all of the CPU registers, while STOP and WAIT only clear the condition code register interrupt mask (I) bit.

Instructions that use inherent addressing can have no other addressing modes, only analogous instructions that use other addressing modes. For example, LSLA, LSLX, and LSL all perform a logical shift left. LSLA operates only on the accumulator, LSLX only on the index register, and LSL only on a memory location.

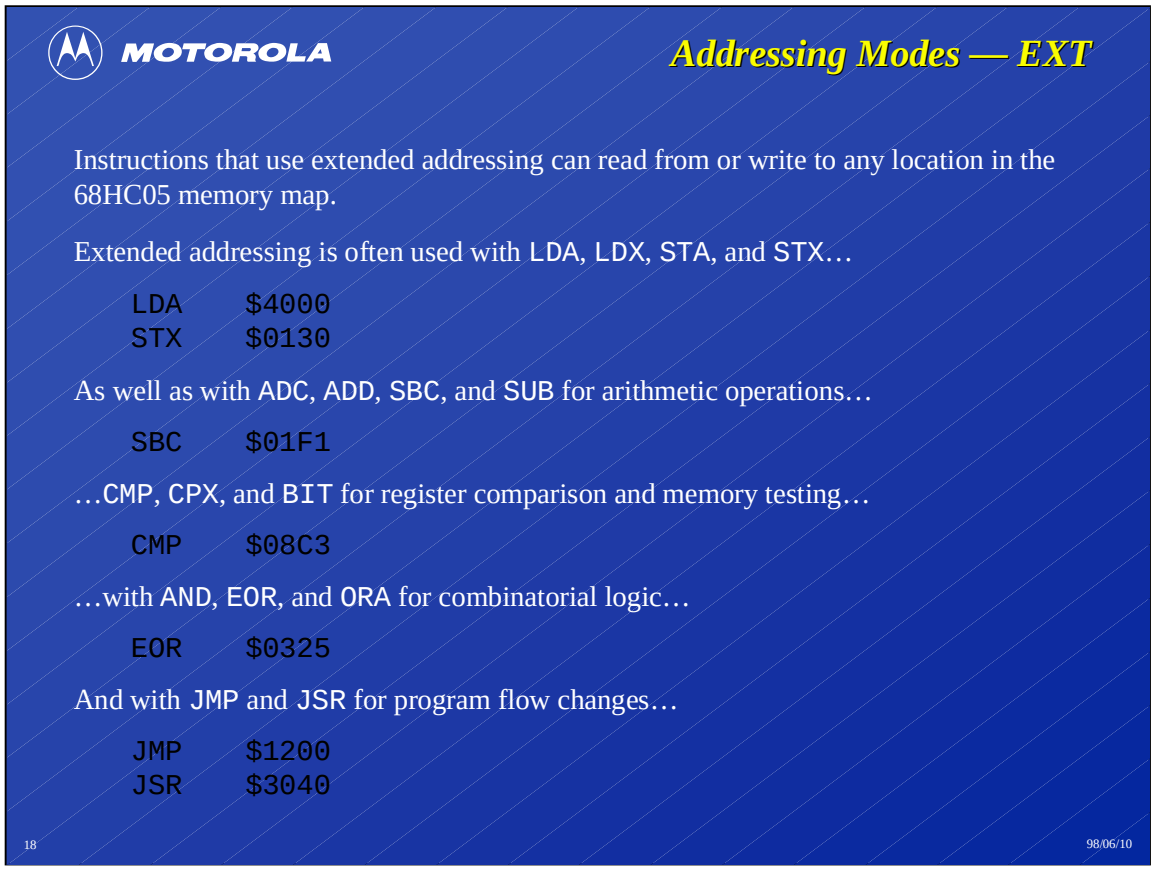


Immediate addressing is used to specify values that remain constant during program execution. On the 68HC05, the address from which an immediate value is read is always one more than the address of the instruction reading that value.

Assembling LDA #$40 at address $1000, for example, will place $A6 (the opcode for LDA when immediate addressing is used) in memory location $1000 and $40 (the operand) in memory location $1001.

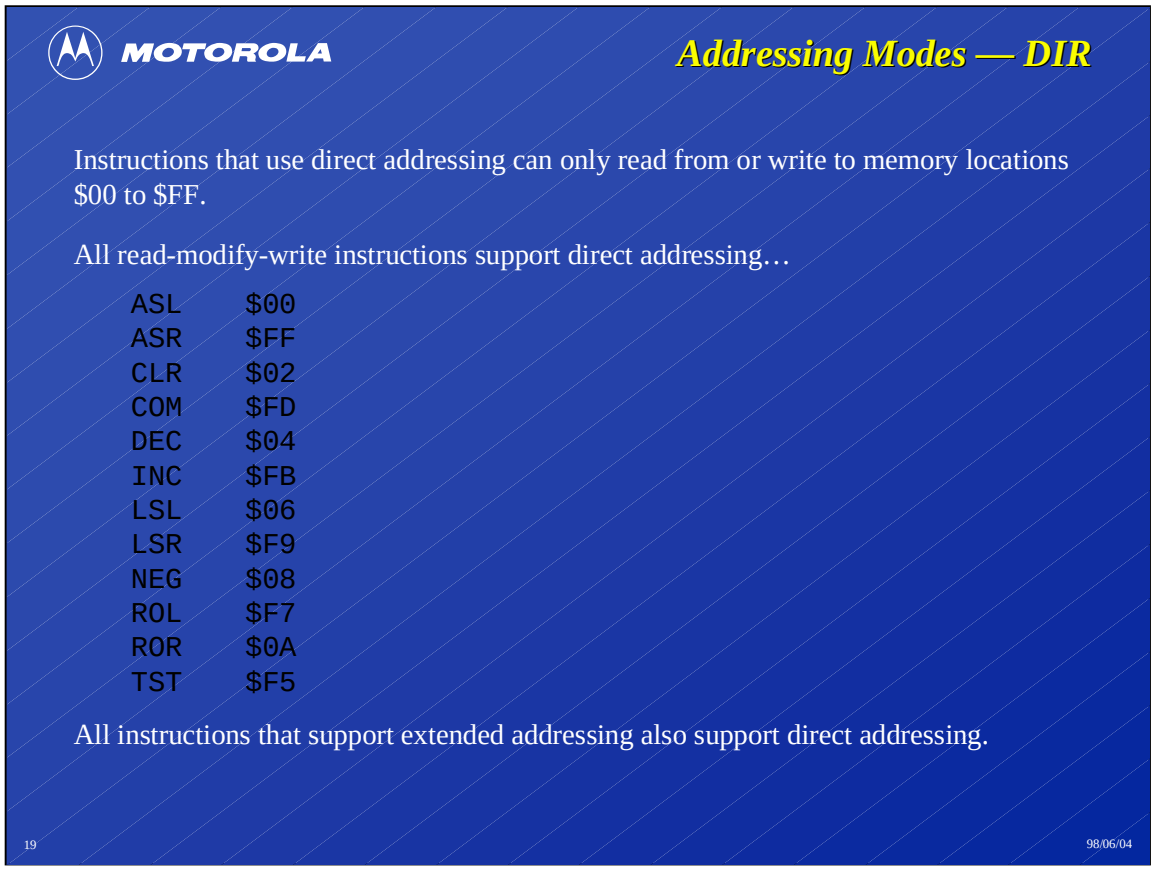
The pound sign (#) is used to designate an operand as an immediate value in Motorola assembly language syntax. It can also be used with assembler-defined symbols. For example, if ASCTONUM is equated to $30, the instruction SUB #ASCTONUM will subtract $30 (decimal 48) from the accumulator.

Pound sign omission is a common assembly language programming error. LDA $40 will not load the accumulator with immediate value $40 (decimal 64). The absent pound sign designates $40 as a direct address, so LDA $40 will load the accumulator with the contents of memory location $0040. For similar reasons, adding a pound sign where it is not required is also a common error.



Extended addressing allows any location in the 68HC05 memory map to be read from or written to using a single instruction. The address is always specified using a 16-bit value.

All instructions that use extended mode addressing require three bytes of program storage, one for the opcode and two for the operand.

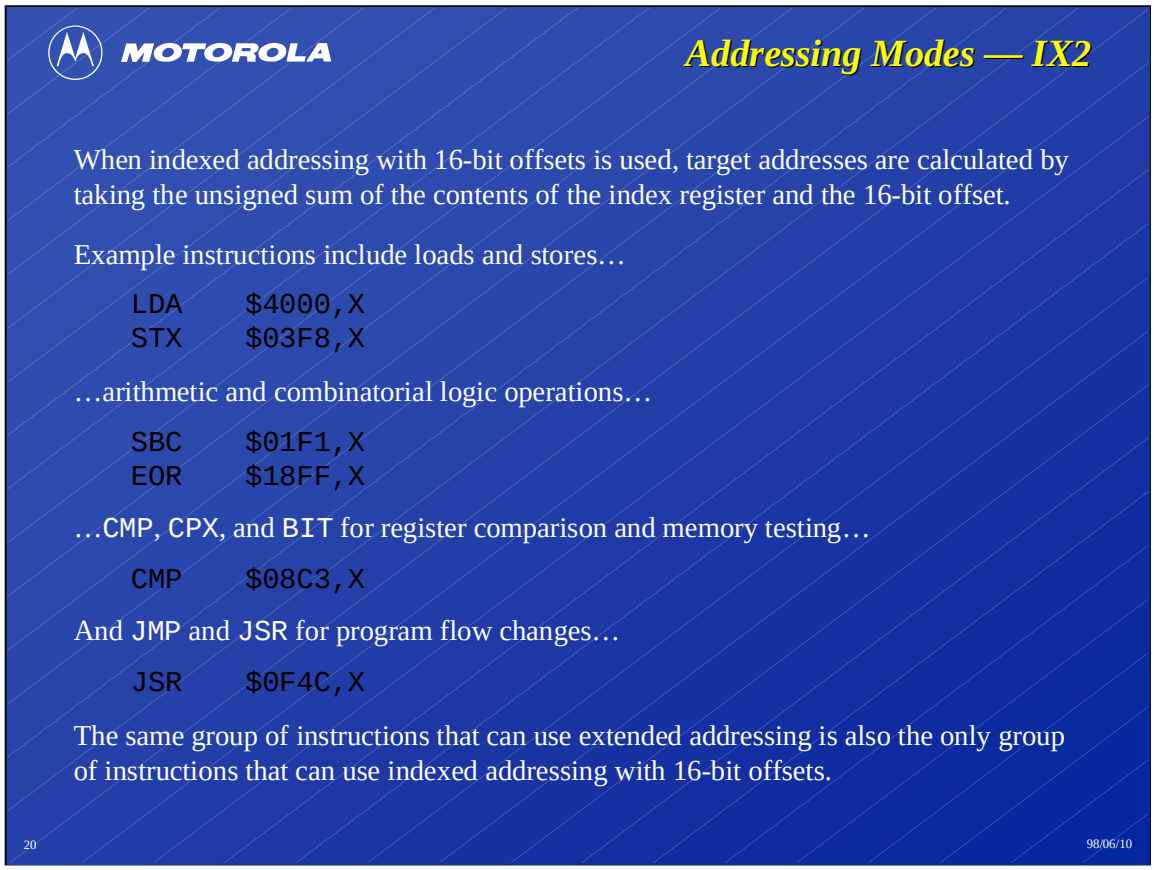


Direct addressing is like a simplified form of extended addressing. It allows any location between $0000 and $00FF to be read from or written to by specifying only the lower eight address bits. Direct mode instructions execute in one less cycle and require one less byte of storage than their extended mode counterparts.

Some instructions, like test, shift, rotate, complement, negate, decrement, and increment have direct, but not extended, mode forms. This may seem to be an omission from the 68HC05, but careful consideration reveals that this is not the case.

Every instruction that has a direct, but not extended, mode form performs a read-modify-write operation. The targets for these instructions must be on-chip RAM or peripheral control registers. Most RAM on 68HC05 devices is located below address $00FF. Similarly, on-chip peripheral registers are mapped starting at address $0000.

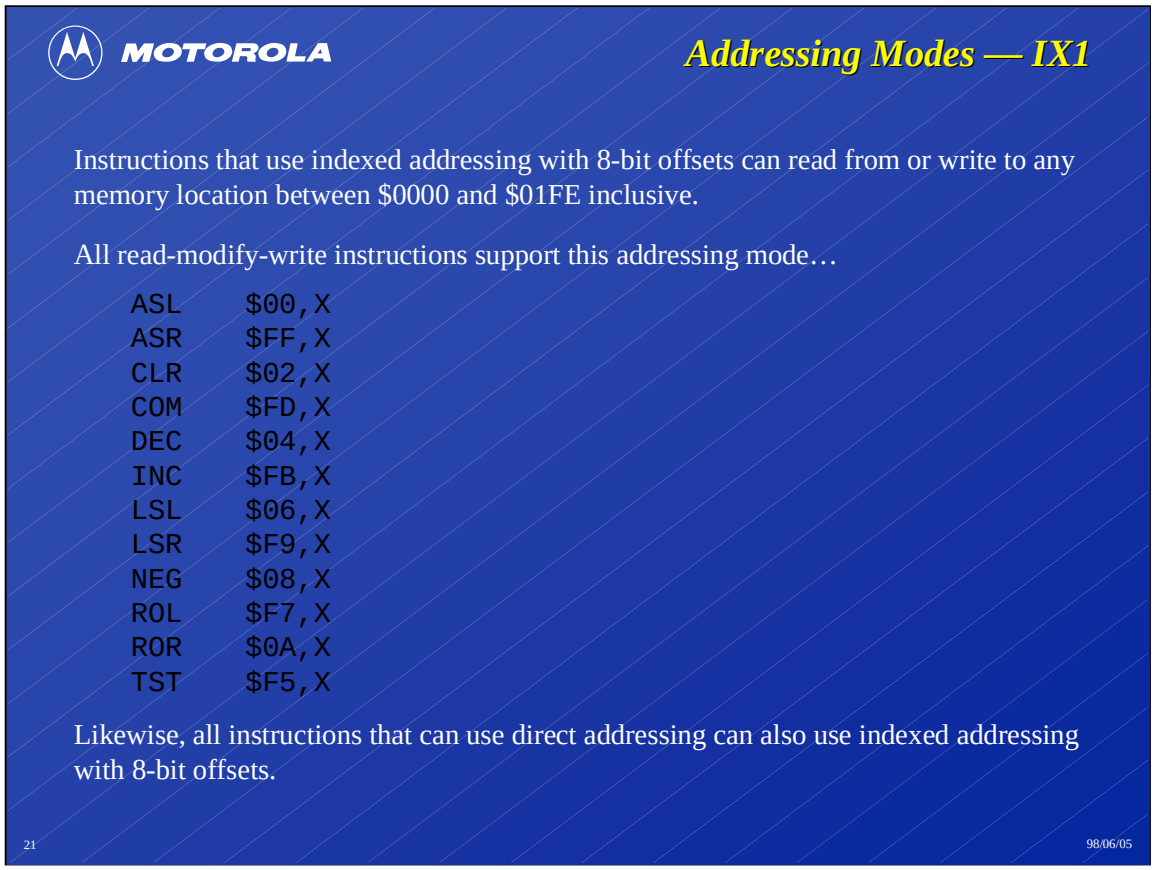
Because the vast majority of read-modify-write operations will target addresses between $0000 and $00FF, the lack of extended addressing (as well as indexed addressing with 16-bit offsets) does not impair device functionality.



Instructions that use indexed addressing with 16-bit offsets can read from or write to any location in the 68HC05 memory map. These instructions always require three bytes of storage.

Target addresses are formed by taking the unsigned sum of the 16-bit offset and the contents of the index register and always range from $0000 to $FFFF. Thus, if X = $41, the instruction STA $FFE0,X will write to address $0021.

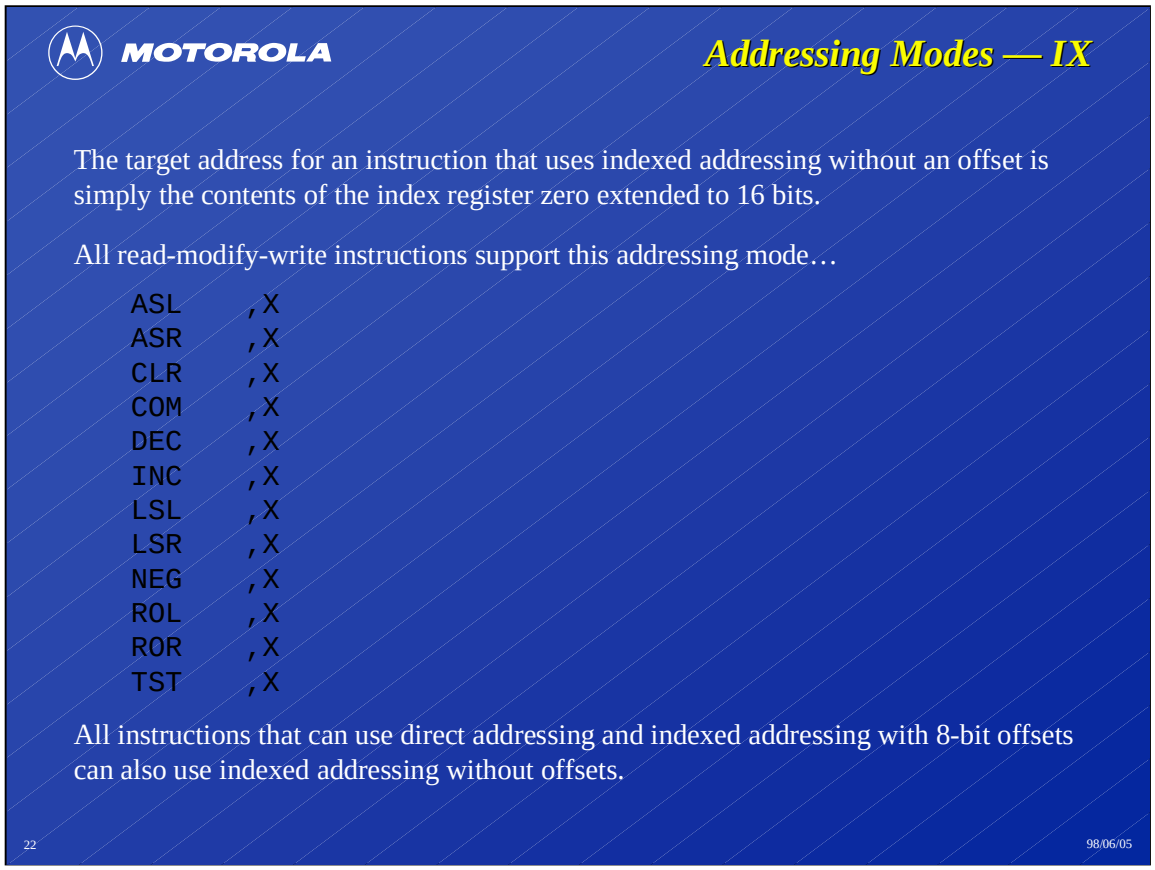
Indexed addressing with 16-bit offsets is especially useful for accessing tabular or string data stored in on-chip ROM which, on most 68HC05 devices, resides primarily above $0100.



Instructions that use indexed addressing with an 8-bit offset have access to the first 511 locations in memory. These instructions execute in one less cycle and require one less byte of storage than their counterparts that use 16-bit offsets.

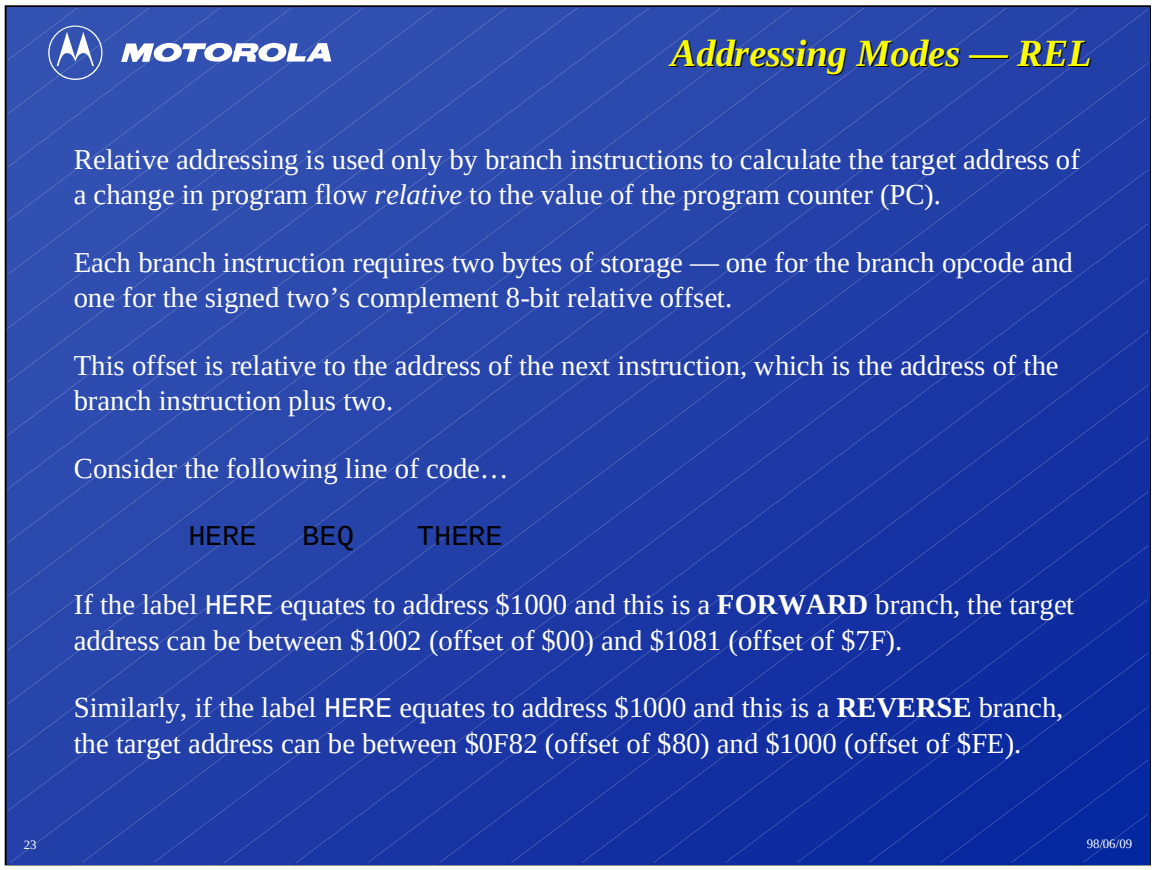
Target addresses are formed by taking the unsigned sum of the 8-bit offset and the contents of the index register and can range from $0000 (offset = $00 and X = $00) to $01FE (offset = $FF and X = $FF).

Most 68HC05 devices have 512 bytes of RAM or less, which makes indexed addressing with 8-bit offsets especially useful for operations on tabular or string data maintained in on-chip RAM.



Instructions that use indexed addressing without offsets can only access memory locations $00 to $FF. Target addresses for these instructions are formed by zero extending the contents of the 8-bit index register to 16 bits.

This is the fastest indexed addressing mode, with instructions needing one less cycle to execute than if an 8-bit offset if used. Looked at differently, these instructions execute as quickly as their direct mode counterparts but require only one byte of storage.

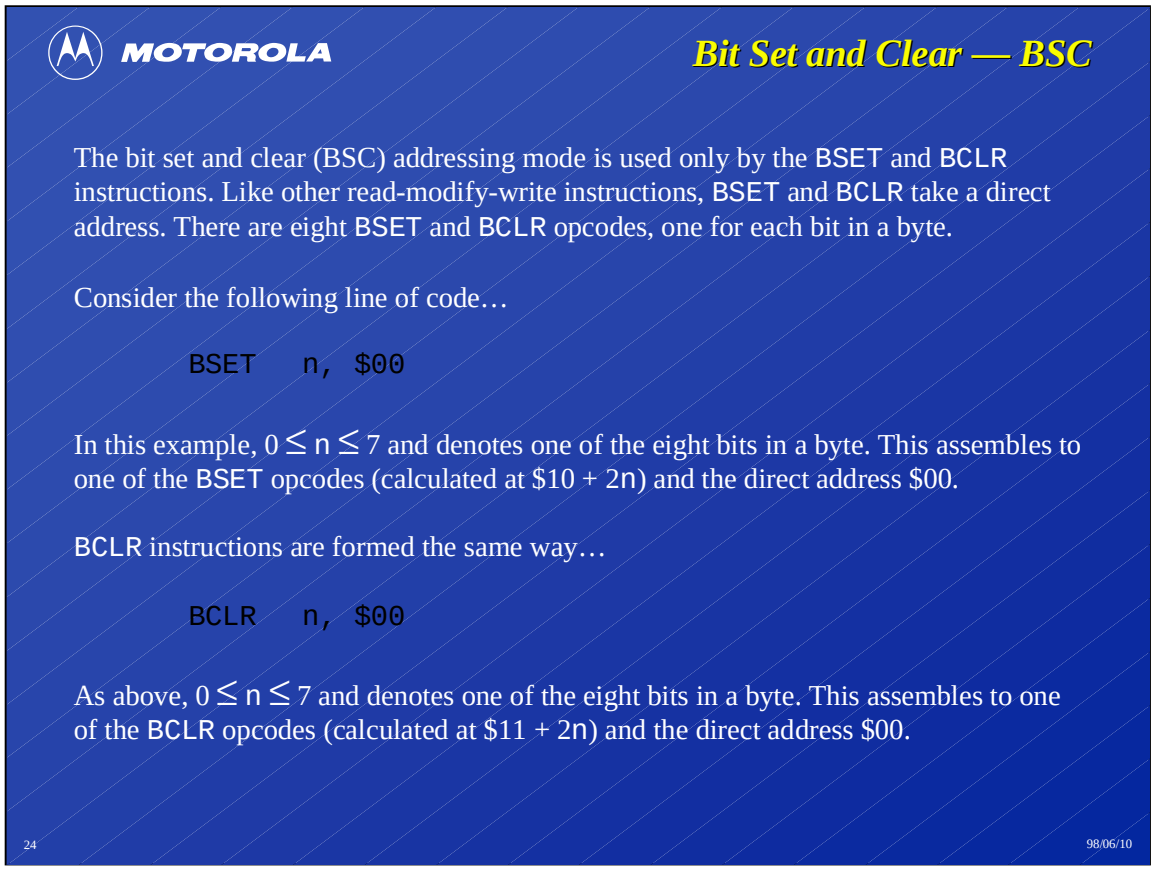


Only branch instructions use relative addressing. A branch instruction executes in three clock cycles and consists of two bytes. The first byte is the opcode for the particular branch condition, and the second byte is a signed two's complement offset from the address of the next instruction.

The target address of a branch instruction is calculated as…

New Program Counter = Address of Branch Opcode + 2 + Signed Offset

Because the offset for a branch is from the address of the next instruction, the smallest practical negative offset is -2 ($FE). An offset of zero causes no change in program flow, essentially making the branch instruction a three cycle NOP.



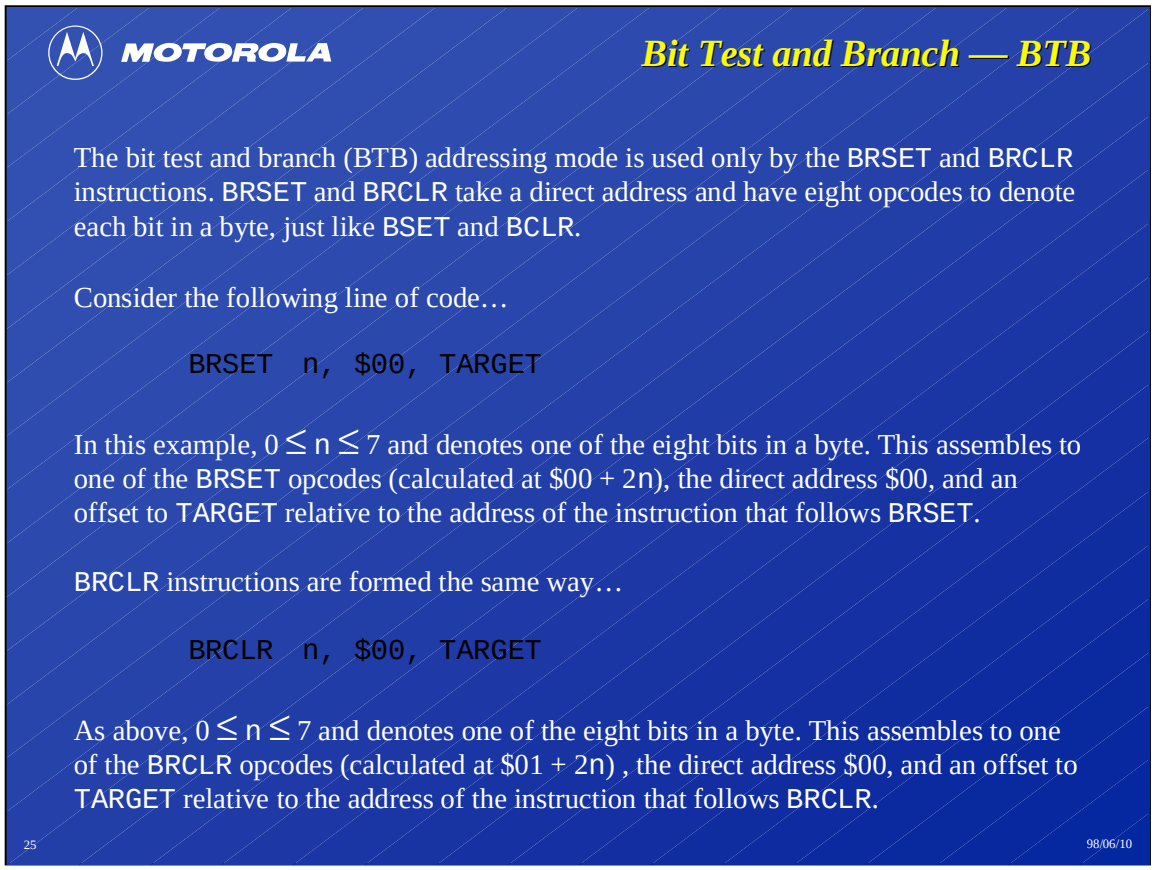
Bit set and bit clear instructions have a unique addressing mode that is similar to direct mode. The source code format for these instructions is:

`BSET/BCLR bit_number, direct_address`

Although each instruction has two operands, an assembled BSET or BCLR instruction consists only of an opcode and a direct address. Each bit that can be modified has its own BSET and BCLR opcodes.

The BSET opcodes for bits 0, 1, 2, 3, 4, 5, 6, and 7 are $10, $12, $14, $16, $18, $1A, $1C, and $1E, respectively. Similarly, the BCLR opcodes for bits 0, 1, 2, 3, 4, 5, 6, and 7 are $11, $13, $15, $17, $19, $1B, $1D, and $1F, respectively.

Like other read-modify-write instructions that take a direct address, BSET and BCLR execute in 5 clock cycles.



Like the bit set and bit clear instructions, the branch on bit set and branch on bit clear instructions also have a unique addressing mode. This bit test and branch (BTB) addressing mode is best described as a cross between the bit set and clear (BSC) and relative (REL) addressing modes. The source code format for these instructions is:

`BRSET/BRCLR bit_number, direct_address, offset`

Although each instruction has three operands, an assembled BRSET or BRCLR instruction consists of an opcode, a direct address, and a signed two's complement offset to the target address. Each bit that can be tested has its own BRSET and BRCLR opcodes.

Target addresses for bit test and branch instructions are calculated as…

New Program Counter = Address of BRCLR/BRSET Opcode + 3 + Signed Offset

Like a regular branch, the target address for BRSET and BRCLR is offset from the address of the next instruction. This makes the smallest practical negative offset for branch on bit set/clear -3 ($FD). These instructions execute in five clock cycles.

MOTOROLA

A Sample 68HC05 Program

The sample function that follows finds the cosine of an angle between 0 and 180 degrees inclusive by interpolating the result from a look up table.

The table consists of 46 elements representing the cosine of every fourth degree, again, from 0 to 180 degrees inclusive, scaled by 127.

A simple linear interpolation is performed using these standardized equations:

$$\text{Cosine of Given } \theta = \text{Cosine of Known } \theta - \text{DELTA}$$

$$\text{DELTA} = \frac{\text{Cosine of Known Lower } \theta - \text{Cosine of Known Upper } \theta}{\text{Known Upper } \theta - \text{Known Lower } \theta} \times (\text{Given } \theta - \text{Known Lower } \theta)$$

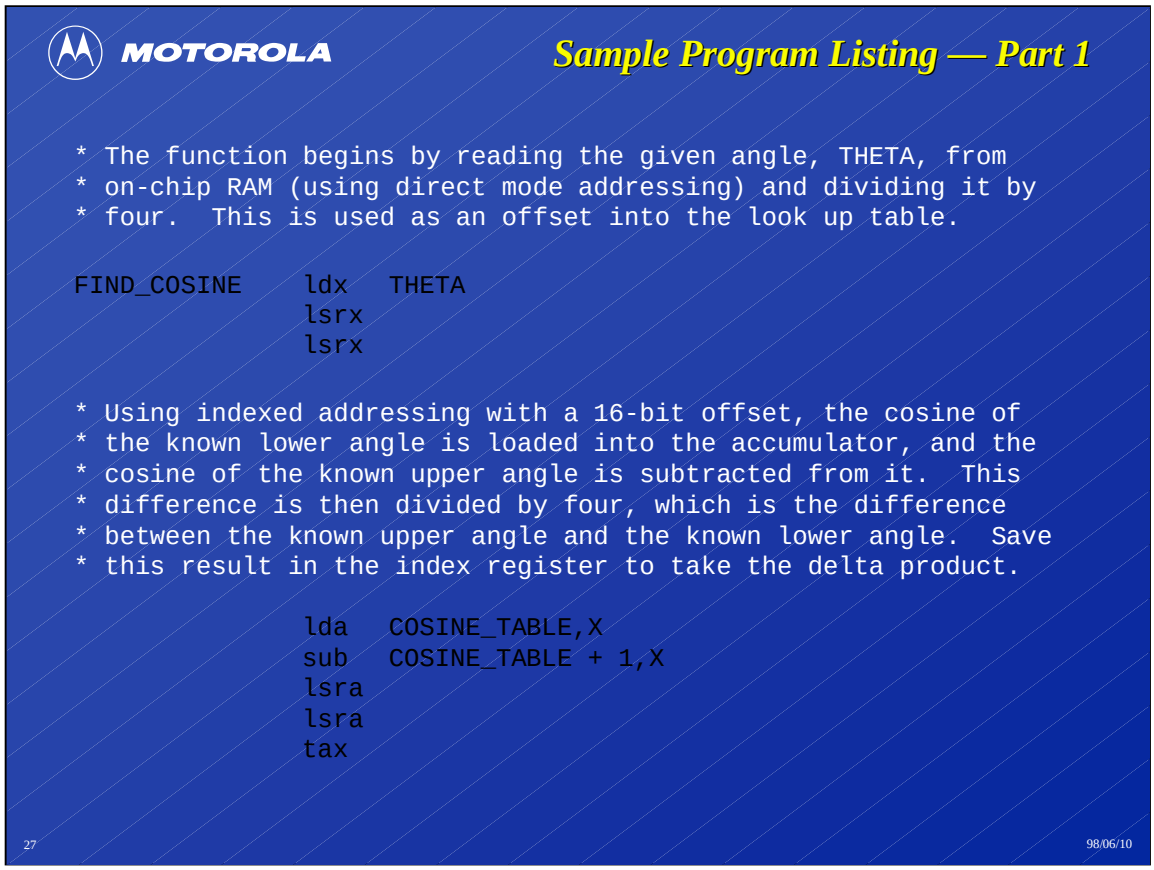
To simplify the interpolation math, the look up table includes the cosine of every fourth degree (0°, 4°, 8°, and so on) rather than, say, every fifth degree (0°, 5°, 10°, and so on).

This makes the difference between the known upper and lower angles four, and allows the division of the difference between the cosines of the known lower and upper angles to be accomplished with two right shift instructions.

Similarly, because all the known angles are multiples of four, the difference between the given angle and the known lower angle is found by logically ANDing the given angle with three.

These simplifications allow the interpolation to be taken with only three mathematical operations:

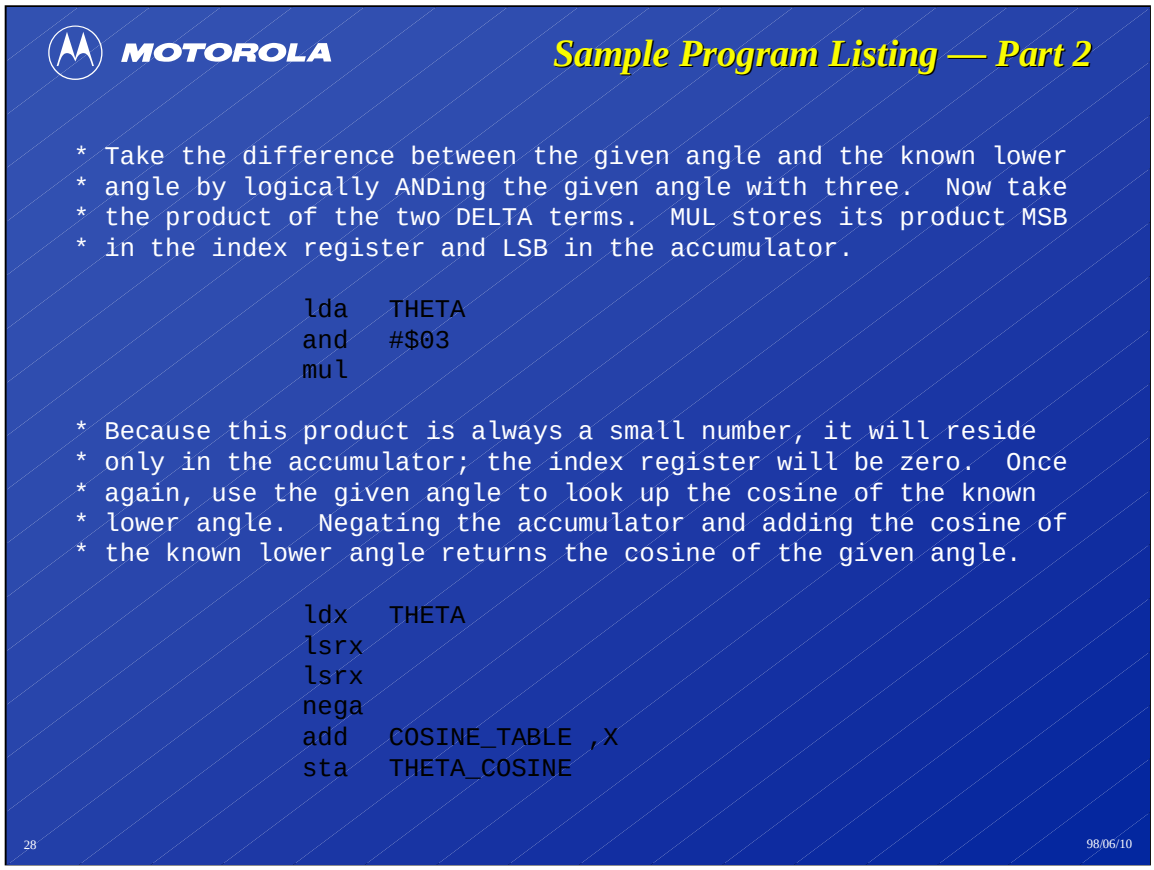
1. the difference between the cosine of the known angle and delta
2. the difference in the numerator of the delta fraction
3. the product of the two delta terms



The FIND_COSINE subroutine starts by reading the given angle, THETA, from on-chip RAM. This would likely be done using direct addressing, because RAM on most 68HC05 devices is located below $00FF. There is usually no explicit need to denote whether an instruction should use direct or extended addressing — most assemblers will make this determination automatically, based on the value to which the symbol THETA is equated.

THETA is loaded into the index register and shifted right two places (effectively dividing it by four) and used with the offset COSINE_TABLE, to read the cosine of the first known angle less than THETA. The value at COSINE_TABLE + 1 subtracted from the accumulator would be the cosine of the first known angle greater than THETA. The offset COSINE_TABLE can be eight or sixteen bits, and as with THETA, most assemblers will determine which addressing mode should be used.

The difference between the cosines of the first known angles less than and greater than THETA is then divided by four (again, using two right shift instructions). Next, it is transferred to the index register where it will be multiplied by the difference between THETA and the first known angle less than THETA.



As noted previously, the following relation is always true because the difference between any two adjacent known angles is always four.

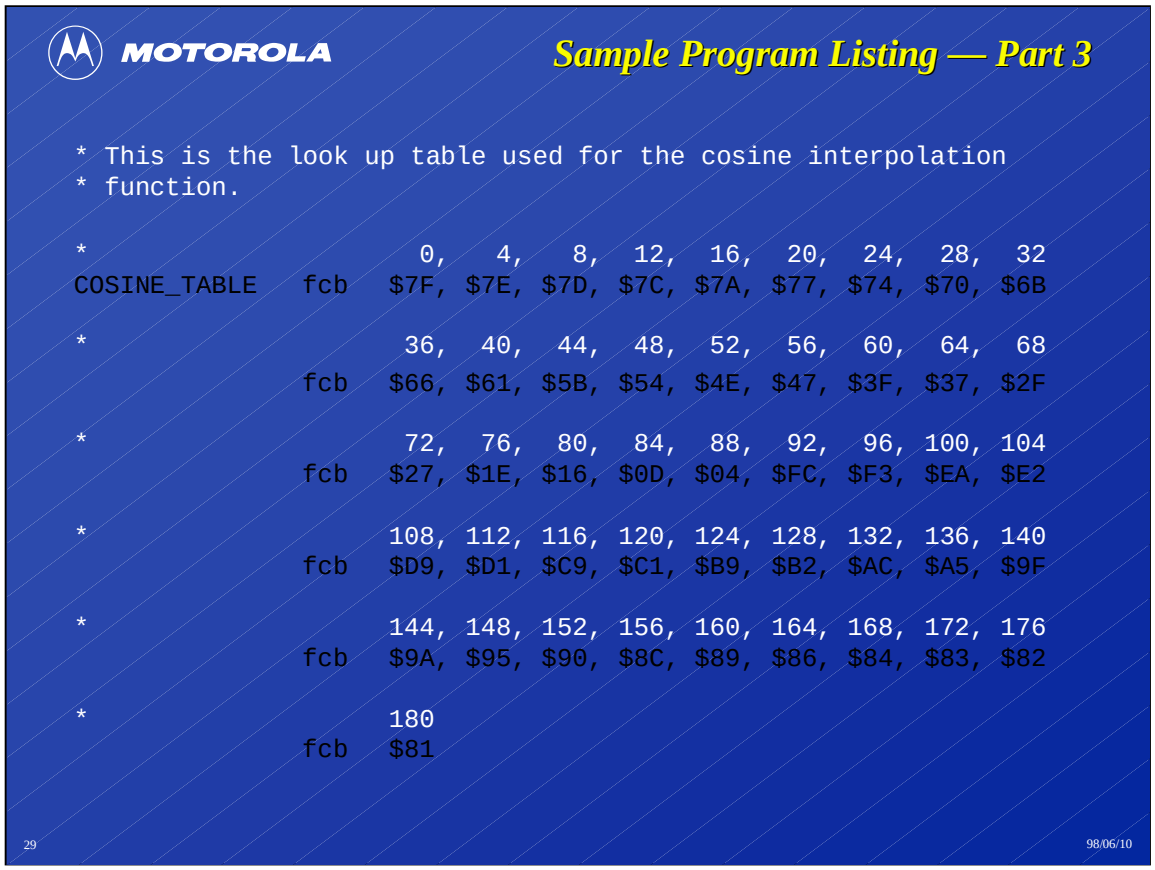
$$0 \leq (\text{THETA} - \text{first known angle less than THETA}) \leq 3$$

Consequently, the difference between THETA and the first known angle less than THETA is simply THETA logically ANDed with three.

With the two terms that comprise DELTA now calculated, DELTA itself can be calculated by taking their product. The unsigned product returned by the MUL instruction is stored with the MSB in the index register and the LSB in the accumulator.

Because the maximum difference between any two cosine terms is nine, and is subsequently divided by four and multiplied by a number no greater than three, the DELTA product will always reside only in the accumulator.

Knowing this, THETA is again read to calculate the offset into the look up table of the first known angle less than THETA. DELTA is negated and added to (effectively subtracting it from) the cosine of the first known angle less than THETA. This is the interpolated cosine of THETA.

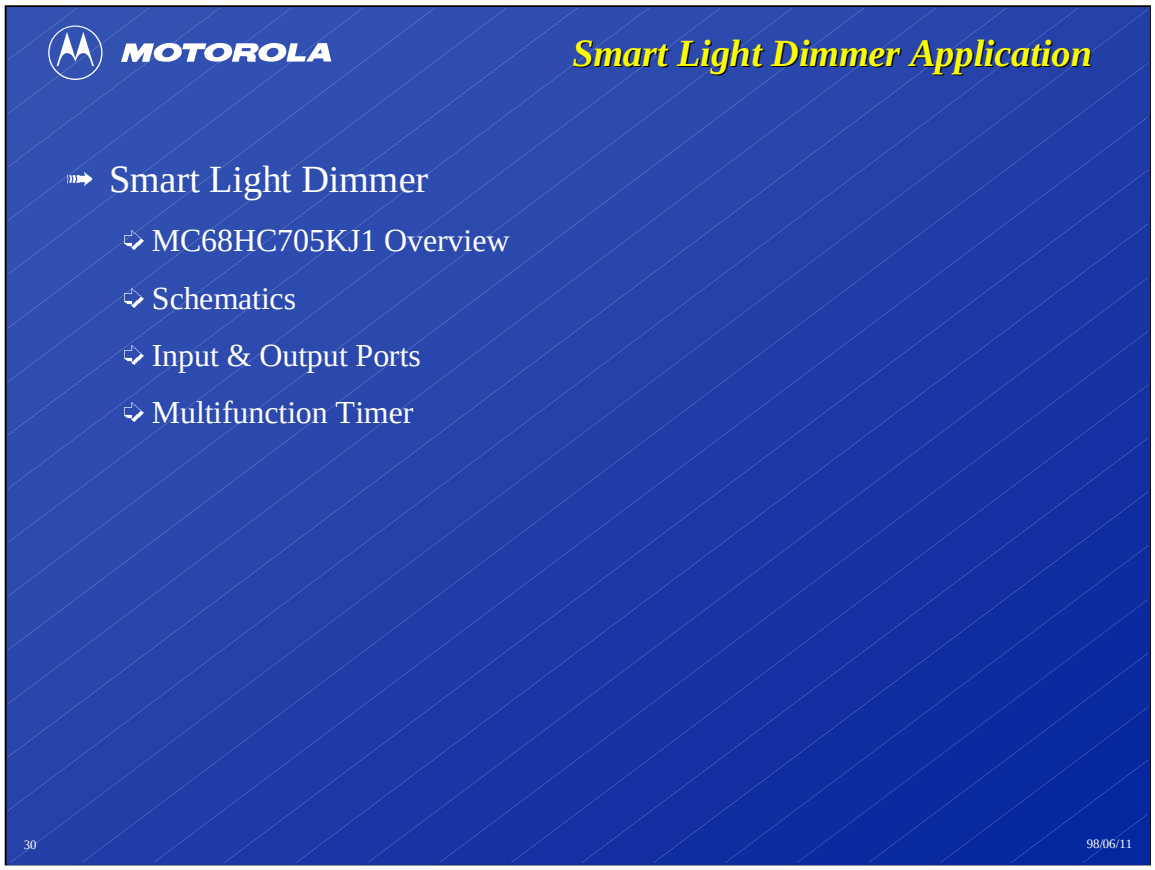


This is the look up table used for the cosine interpolation function just presented. It consists of the 46 values that represent the cosine of every fourth angle from 0 to 180 degrees, inclusive, multiplied by 127.

Together, this table and the interpolation code require 46 bytes of storage. A complete table and associated look up code for every integer degree from 0 to 180 would require 186 bytes of storage.

If the slightly reduced accuracy of this method can be tolerated, a code space savings of 75% is achieved over a fully implemented look up table. A more accurate 91 entry table (a cosine for every second angle from 0 to 180 degrees inclusive) and associated look up code require 116 bytes of storage and still manage to save 70 bytes over the complete alternative.

Other options are available for implementing trigonometric functions. Taylor series approximations exist for sine and cosine but require arithmetic capability well beyond the means of small microcontrollers. Table interpolation, then, offers a good compromise between space, accuracy, and computational requirements.



Every 68HC05 device has a particular mix of integrated peripheral devices. This section shows how the peripherals on two 68HC05 devices are used in different applications.

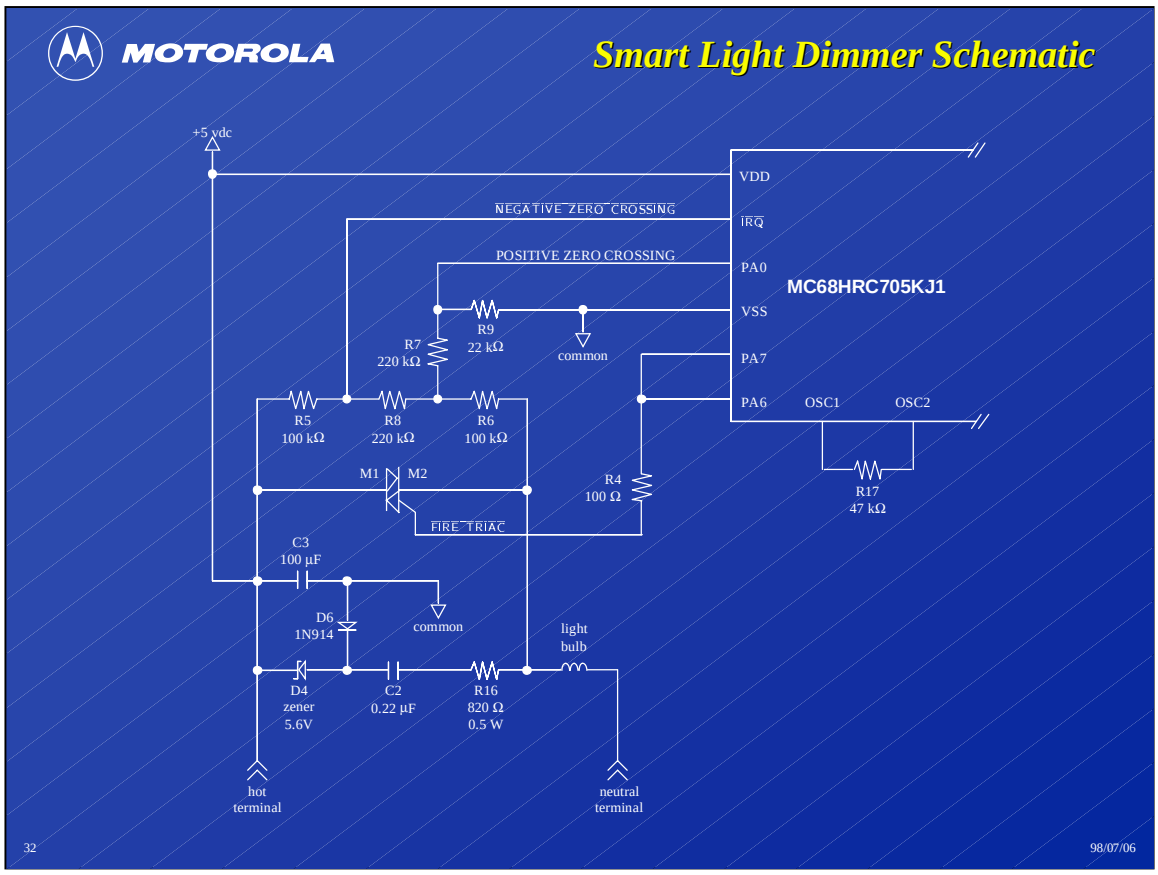
The first example is a smart light dimmer. A typical light dimmer consists of little more than a user-actuated potentiometer that offers varying levels of brightness. By using the MC68HC705KJ1, the smart dimmer can do this and, additionally, can pleasantly fade in and fade out room lighting.

For it to be an affordable replacement for conventional dimmers, the smart light dimmer needs a cost-efficient microcontroller. The MC68HC705KJ1 fits the bill by providing features appropriate for this application.

Apart from the EPROM and RAM used for program and variable storage, the smart light dimmer depends on three MC68HC705KJ1 features. In the schematic diagram that follows, a triac controls the AC waveform seen by the light bulb. Because it has 10 mA current sink capability on all of its I/O pins, the MC68HC705KJ1 can drive this triac directly. A less capable device would require triac drive hardware that would increase the smart light dimmer's cost.

The triac automatically turns off each time the AC waveform crosses zero. By using both the active low $\overline{IRQ}$ interrupt and the optional active high interrupt capability of the low order port A pins, an interrupt is generated each time the AC waveform crosses zero.

With the multifunction timer, a delay can be inserted between each zero crossing and subsequent firing of the triac. This delay determines the conduction period of the triac, the length of which is directly proportional to the brightness of the light bulb.



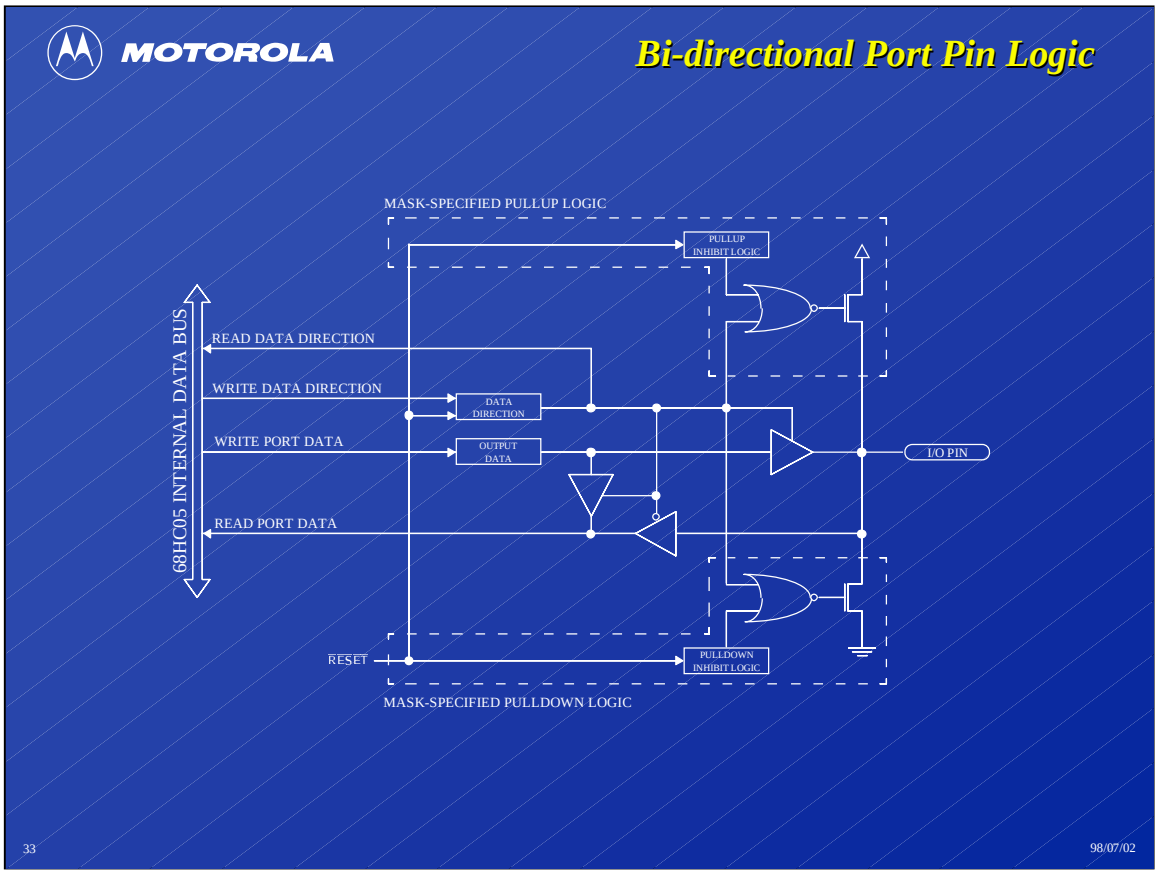
All 68HC05 devices have some number of pins, either dedicated or shared with other peripherals, that can be used as digital inputs and outputs. The partial circuit for the smart light dimmer, shown above, illustrates two useful features available on the I/O pins of the MC68HRC705KJ1.

In order to directly drive LEDs, or in this circuit, the gate control of a triac, many 68HC05 devices, including the MC68HRC705KJ1, have outputs with low side current sink capability of 10 mA or more.

The inputs on some 68HC05 devices can optionally be used as active high interrupt request lines. This is the case with the four low order bits of port A on the MC68HRC705KJ1.

In the example above, negative and positive zero crossings of the AC line voltage are detected by using the $\overline{\text{IRQ}}$ external interrupt pin and the optional port A interrupt capability, respectively.

Fade in and fade out of the light bulb (initiated by touch sensors connected to other inputs not shown in this diagram) are accomplished by using the MC68HRC705KJ1 multifunction timer to vary the delay between each zero crossing and MCU-controlled firing of the triac.

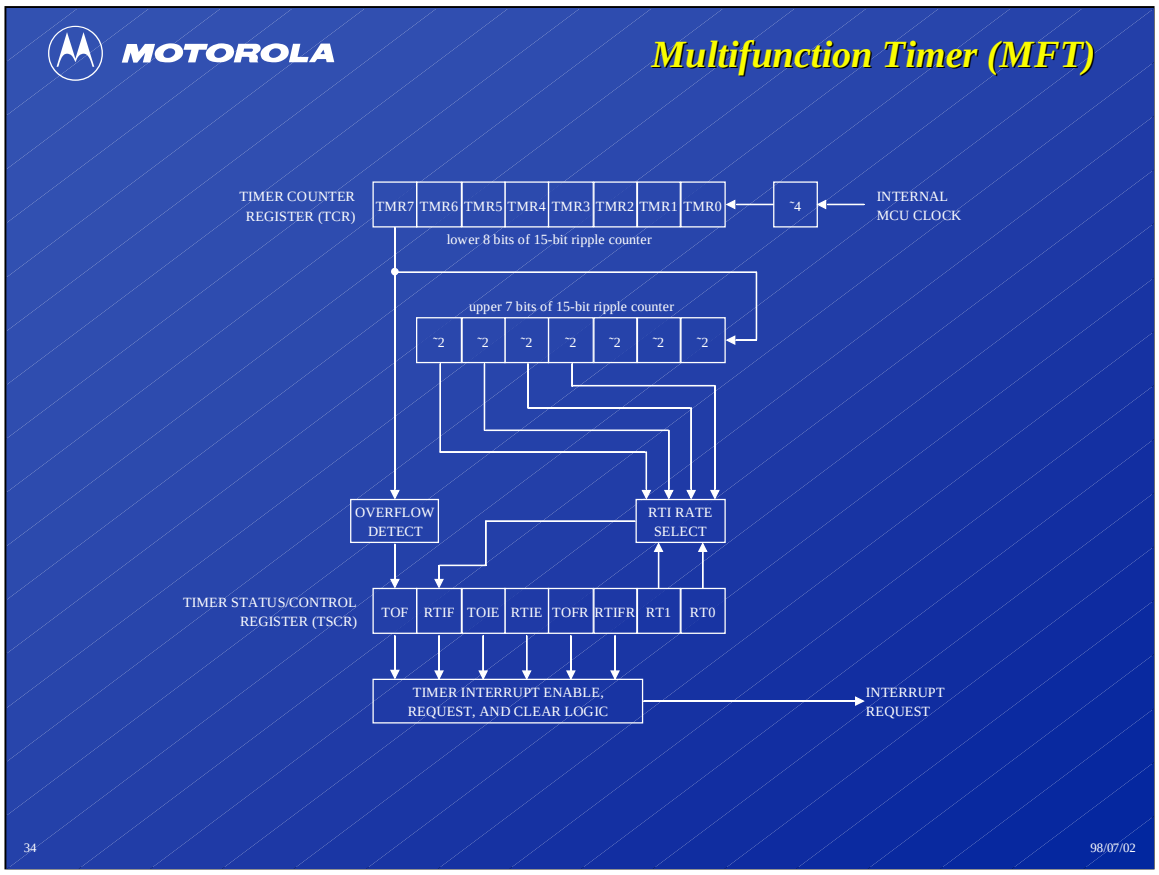


The diagram above shows a simplified version of the logic that comprises a bi-directional port pin. I/O pins, typically in groups of eight, comprise a single port.

All ports have a data register that holds the value written to or read from the port. Bi-directional ports share a single data register that, when read, returns the state of any pins configured as inputs and, when written, drives pins configured as outputs to their specified states. A data direction register specifies whether each pin associated with a bi-directional port is an input or an output. All bi-directional pins are preconfigured as inputs during and immediately after $\overline{\text{RESET}}$ assertion.

Many 68HC05 devices have mask programmable pulldown or pullup devices that keep pins configured as inputs in a known state. Integrated pulldown or pullup devices generally eliminate the need for similar external components and, in turn, help reduce current consumption by preventing CMOS input devices from “floating” at indeterminate levels.

On 68HC05 devices with UV-erasable or one-time programmable EPROM, integrated pulldown or pullup devices are controlled by a mask option register that is programmed in the same fashion as the main EPROM array.



Simple 68HC05 devices have a simple timer, the multifunction timer (MFT). Consisting of a 15-bit ripple counter, the lower eight bits of which can be read from the timer counter register (TCR), the MFT is clocked at one fourth the internal MCU clock frequency.

The MFT is essentially a circuit for generating various periodic interrupts. Bits in the timer status/control register (TSCR) can enable interrupts for overflow of the lower eight bits of the ripple counter (flagged by TOF, enabled by TOIE, and cleared by TOFR) and overflow of the real-time interrupt (flagged by RTIF, enabled by RTIE, and cleared by RTIFR).

One of four real-time interrupts rates can be selected by the RT[1:0] bits in TSCR. Overflows from TCR clock the upper seven bits of the ripple counter which serve as the time base for real-time interrupts.

Interrupts can be generated every 1024 internal clock cycles by the timer overflow interrupt and every 16384, 32768, 65536, or 131,072 internal clock cycles by the real-time interrupt.

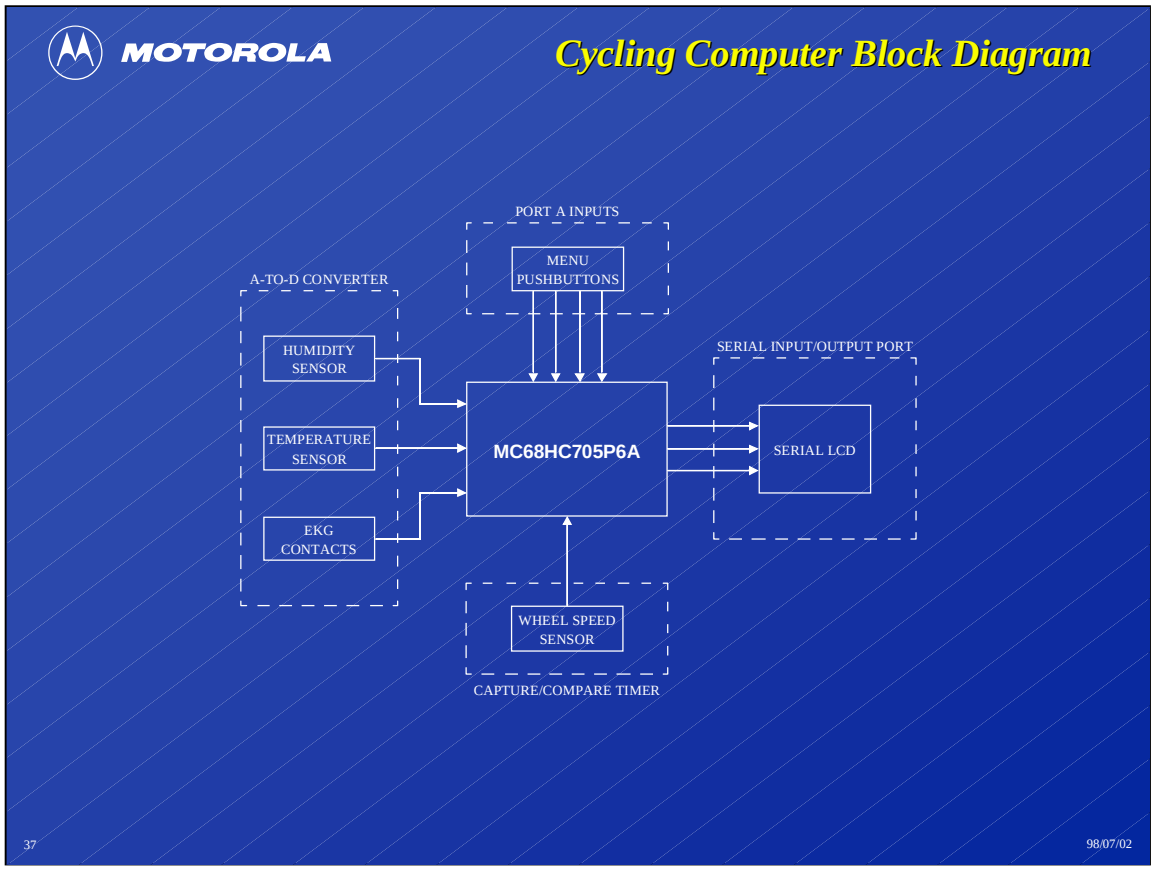
Information that cyclists need to ride smarter is provided by the bicycling computer in the second example. This application uses the MC68HC705P6A to monitor heart rate, air temperature, humidity, speed, and distance traveled. With this data, cyclists can track course performance and avoid over-exertion if it is too hot or too humid.

Based on a Motorola University Design Contest entry, this next example makes extensive use of the features found on the MC68HC705P6A. The peripherals used in this application (analog-to-digital converter, timer, serial input/output port, and I/O pins) are found on many different 68HC05 devices.

With three of the analog-to-digital converter's (ADC) four channels, the cycling computer monitors rider heart rate, air temperature, and ambient humidity. If desired, the remaining ADC channel could be used to monitor the cycling computer's battery voltage.

Bicycle speed and distance traveled are calculated using the 16-bit timer's overflow and input capture functions. The output compare function synchronizes ADC operation.

This information are gathered and shown on a serially interfaced liquid crystal display (LCD) driven by the MC68HC705P6A serial input/output port (SIOP). Pushbuttons connected to the port A inputs (with optional pullup devices and interrupt capability enabled) allow the user to cycle through the different statistics displayed, start and stop an elapsed time counter, and adjust certain operating parameters. The cycling computer must know tire size, for example, to accurately calculate speed and distance traveled

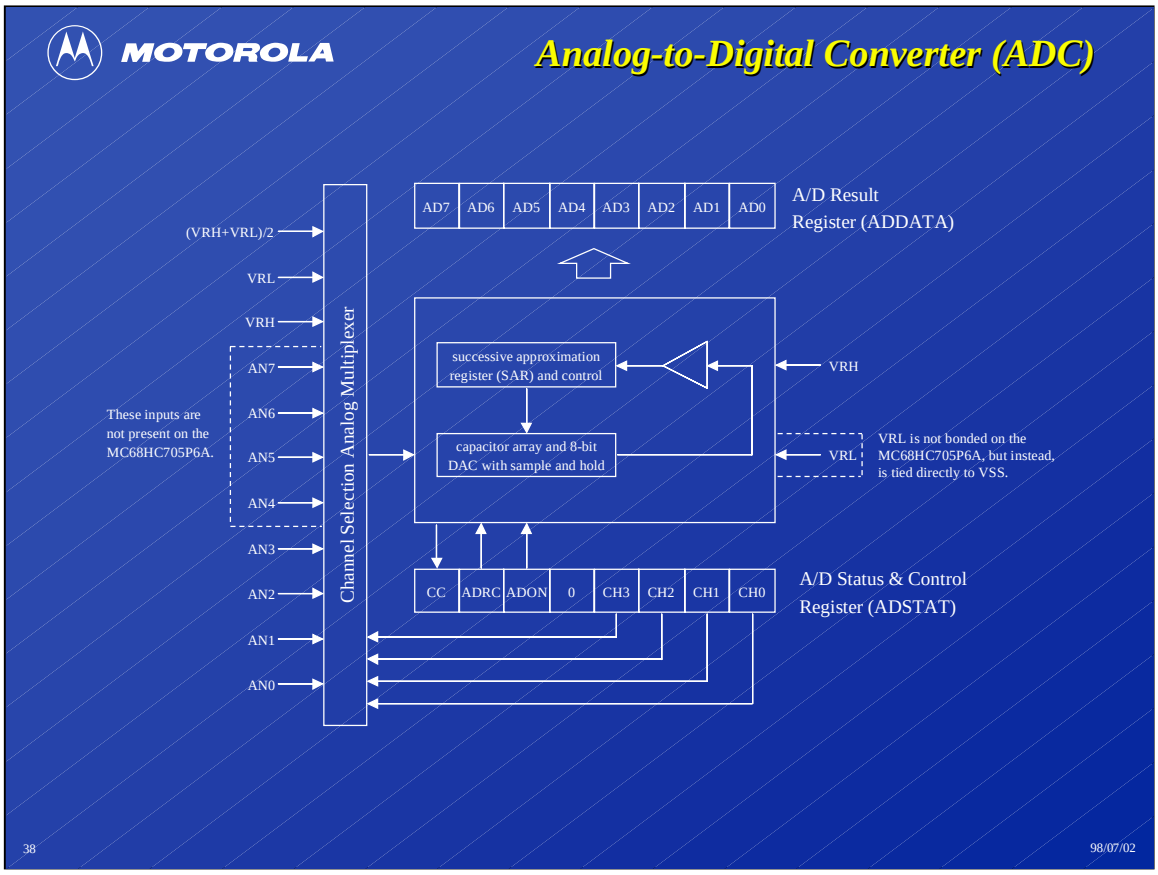


For riders to find the cycling computer useful, it must collect and present its statistics with a minimum of fuss. Ambient temperature and humidity are easily monitored with sensors attached to the cycling computer's housing, but the rider's heart rate presents a special problem.

In its original design, the cycling computer monitored heart rate using either a set of rib cage electrodes or a glove mounted photosensor. Neither of these would be acceptable in a commercial product, because connecting wires between the sensors and the cycling computer could become tangled or impede rider movement.

New foam handlebar grips with integrated foil contacts solve this problem and are readily available, because they are commonly used on stationary exercise bicycles. When gripped by the rider, an EKG signal that can be monitored by the ADC appears across these contacts.

The 16-bit timer input capture function on the MC68HC705P6A provides the information needed to calculate speed and distance traveled. A magnetic switch connected to the input capture pin detects wheel rotations, the number of which is directly proportional to distance traveled. Likewise, speed can be calculated from the number of wheel rotations that occur during a given period of time.



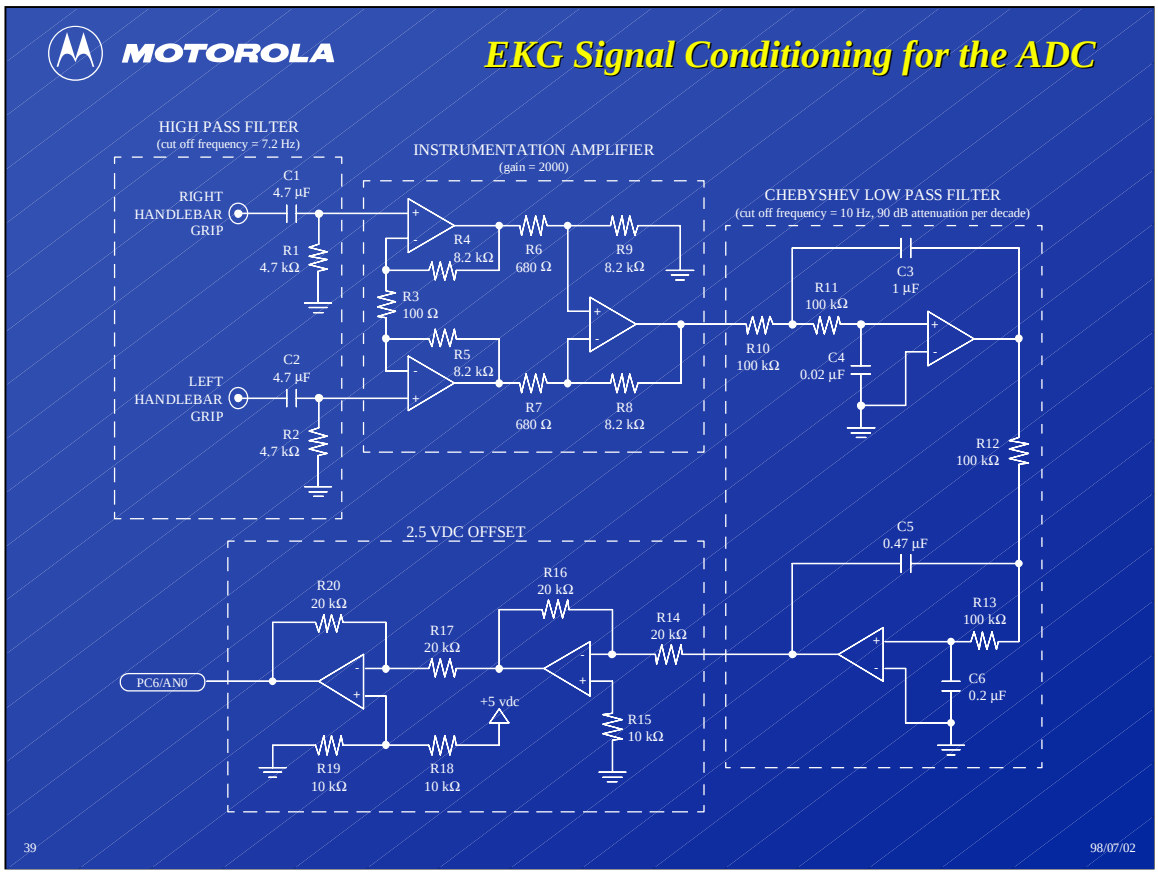
The MC68HC705P6A incorporates a 4-channel version of the 8-bit analog-to-digital converter (ADC) used on several 68HC05 devices.

The ADC is a ratiometric, fully monotonic, successive approximation converter. It has an analog multiplexer that supports up to eight input channels and also allows the high and low references and one half the difference between references to be read for calibration purposes.

Pins used by the ADC may also be used as inputs and sometimes outputs. Total converter accuracy is ±1 or ±1.5 LSB when its channels are input-only or bi-directional pins, respectively. The converter's high reference voltage is supplied by a separate pin. The low reference input, when not bonded to an external pin, is internally tied to VSS.

Each single channel conversion requires 32 internal clock cycles. A separate RC oscillator, enabled under software control, can clock the ADC when MCU operating frequencies are below 1 MHz.

The converter is disabled on power-up and is enabled by setting the ADON bit in the A/D status and control register. Once enabled, conversions continue until the ADON bit is cleared or the MCU enters low-power stop mode.

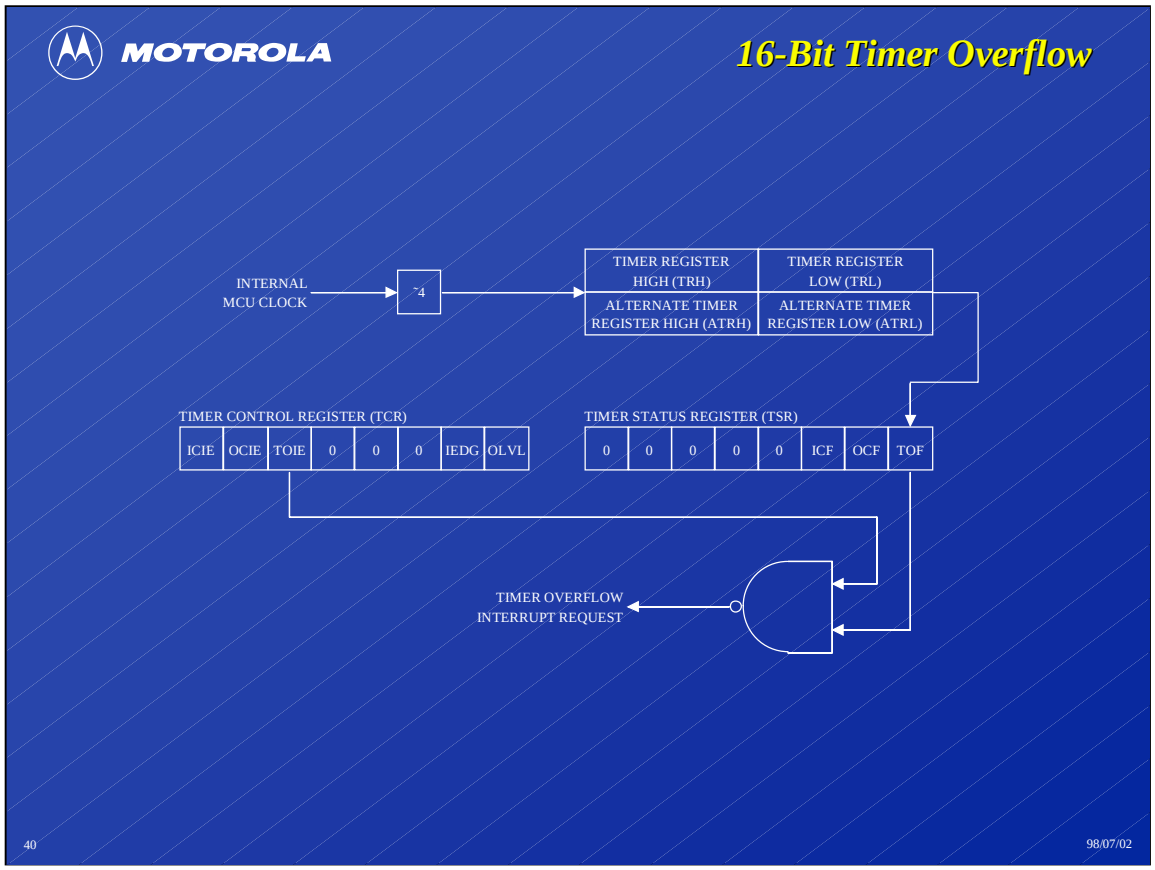


The cycling computer uses the analog-to-digital converter to sample the EKG signal that appears across the handlebar contacts when they are gripped.

Short leads connect the handlebar contacts to an instrumentation amplifier that multiplies the 4 mV peak-to-peak EKG signal by 1980. Input capacitors C1 and C2, along with shunt resistors R1 and R2, precede the instrumentation amplifier and form a high pass filter with a cut off frequency of 7.2 Hz.

After amplification, the signal is passed through a fourth order Chebyshev low pass filter that provides 90 dB of attenuation per decade beyond the cut off frequency of 10 Hz. This, along with the input high pass filter, eliminates extraneous noise beyond the 5 to 10 Hz frequency band of the signal that is digitized by the MC68HC705P6A.

After amplification and filtering, the EKG signal should lie approximately between -2.25 and +2.25 volts. A final stage in the signal conditioning chain provides a DC offset of 2.5 volts, placing the EKG waveform in the 0 to 5 volt range required by the MC68HC705P6A analog-to-digital converter.

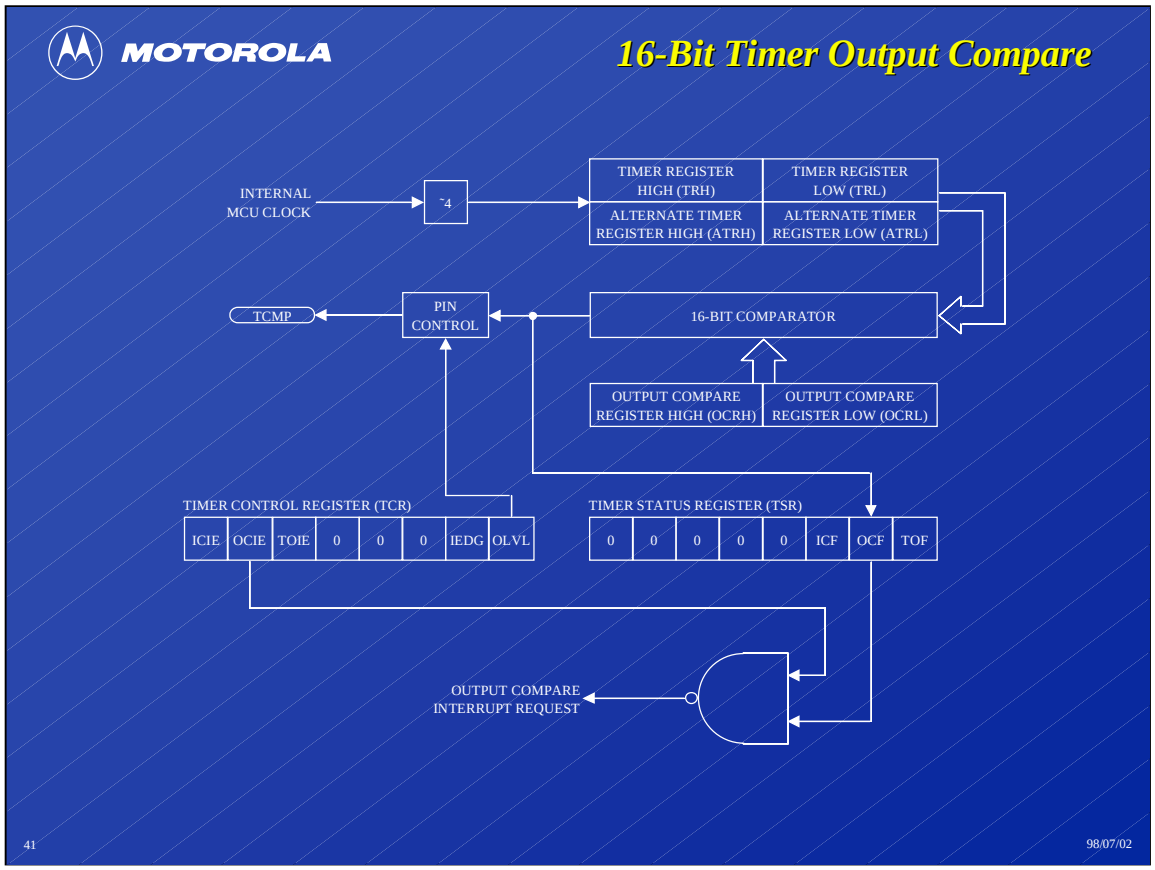


All functions of the 16-bit timer are related to the 16-bit counter which serves as its time base.

The counter can be read from two locations, both of which return the same data. A read of the timer register high (TRH) byte or the alternate timer register high (ATRH) byte returns the high byte of the 16-bit counter and latches the low byte of the 16-bit counter in a buffer until it can be read from TRL or ATRL. Repetitive reads of the high byte will not change the low byte in the buffer until it is read.

The 16-bit counter increments once every four internal clock cycles, and upon reaching $FFFF, it rolls over to $0000. This event sets the timer overflow flag (TOF) in the timer status register (TSR) and generates a timer overflow interrupt request if the timer overflow interrupt enable (TOIE) bit in the timer control register (TCR) is set.

The TOF bit in TSR remains set unless explicitly cleared. A read of TSR, followed by a read of TRL, clears TOF. The same sequence will not clear TOF if ATRL is read in place of TRL. This allows the counter to be read at all times (from ATRH and ATRL) without the possibility of missing a timer overflow interrupt.

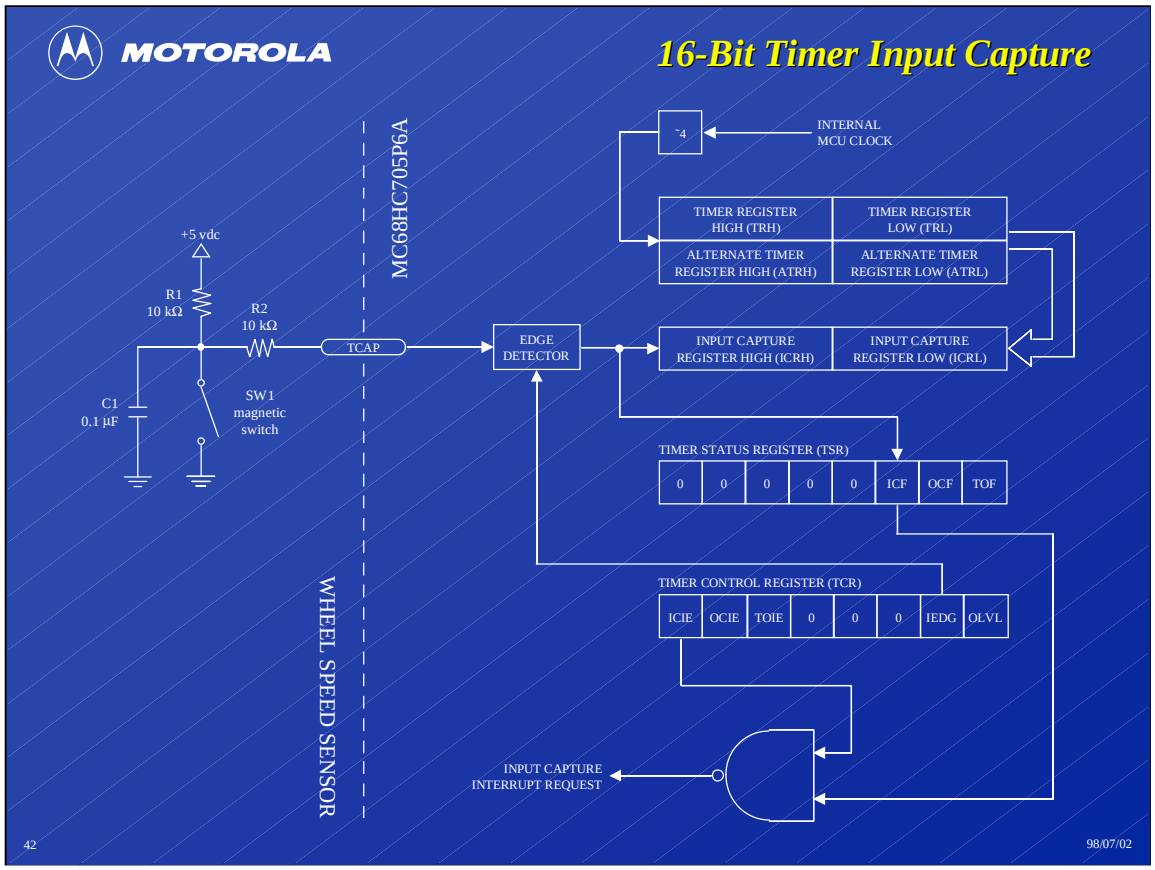


Output compare is essentially a method for generating delays, because it allows future events to be synchronized to the current value of the 16-bit timer counter.

The output compare high (OCRH) and low (OCRL) registers hold the value that the 16-bit timer counter will match at some point in the future. When writing to the output compare registers, first write data to OCRH. This prevents a match from occurring until OCRL is written.

When a match occurs, the TCMP pin will be driven to the level specified by the OLVL bit in the timer control register (TCR), and the output compare flag (OCF) bit in the timer status register (TSR) will be set. An interrupt will also be generated if the output compare interrupt enable (OCIE) bit in TCR is set. A read of TSR, followed by a read of or write to OCRL, clears OCF.

The EKG signal that appears across the handlebar contacts must be sampled at a fixed frequency in order to make accurate pulse calculations. Analog-to-digital conversion of the EKG waveform occurs during the service routine of a 100 Hz interrupt generated by the output compare hardware. By finding two adjacent peaks of similar amplitude on the EKG wave, the time between two heart beats is known in hundredths of a second and can be converted to a pulse rate in beats per minute.



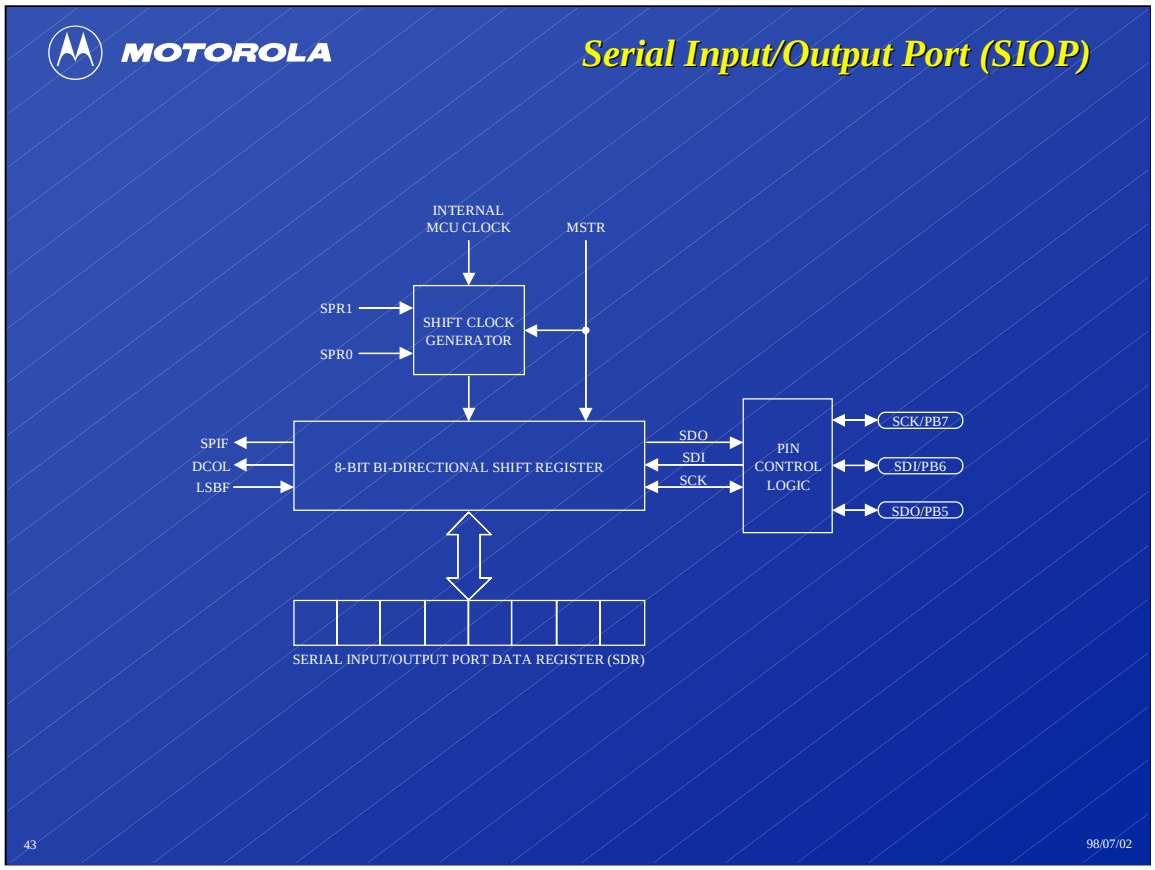
Input capture functions like output compare in reverse. Instead of generating an edge on the TCMP pin at a specific time, input capture saves the time at which a specific edge occurs on the TCAP pin.

The IEDG bit in the timer control register (TCR) specifies whether rising or falling edges are recognized by the input capture hardware. When a capture occurs, the input capture flag (ICF) bit in the timer status register (TSR) is set, and an interrupt will be generated if the input capture interrupt enable (ICIE) bit in TCR is set.

To clear ICF, read the input capture register high (ICRH) byte. This prevents further captures and latches the low byte of the result until ICRL is read.

In the wheel speed sensor shown above, each rotation of the bicycle’s front tire closes the magnetic switch. This generates a falling edge that is detected by the input capture hardware. The number of wheel rotations that occurs during a given period of time is directly proportional to the distance traveled.

When the MC68HC705P6A is clocked with a 4.194 MHz crystal, the 16-bit counter overflows eight times per second. This allows the cycling monitor to accurately update and display the speed and distance traveled every one and ten seconds, respectively.

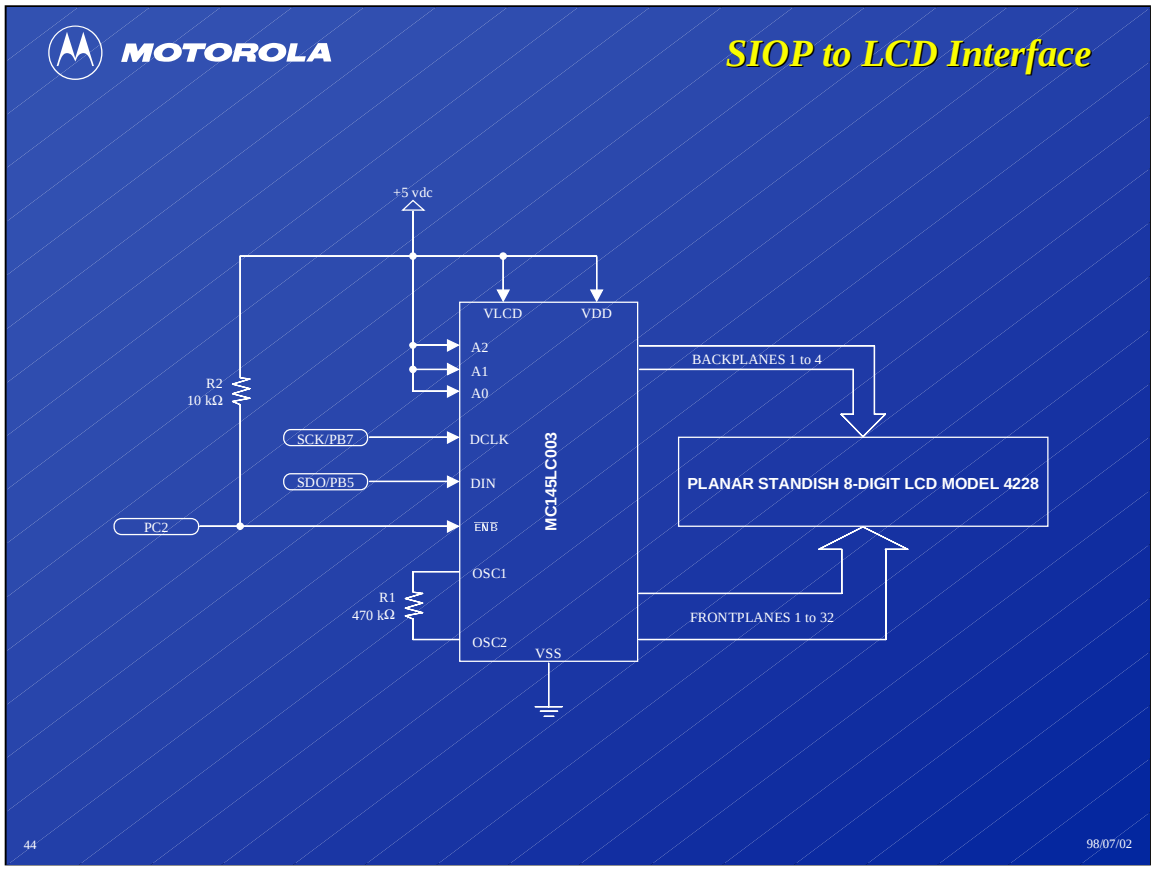


The serial input/output port (SIOP) is a simplified version of the serial peripheral interface (SPI) that appears on numerous Motorola microcontrollers ranging from other 68HC05 devices to members of the highly-integrated 68300 family. The SIOP supports master (processor initiated) and slave (externally initiated) mode transfers.

A bi-directional shift register, dividers that derive the shift clock from the internal MCU clock, and three shared pins with associated control logic comprise the SIOP.

Depending on the device, three or four registers control the SIOP. All SIOP implementations have a data register (SDR) from which received data is read and to which data for transmission is written. The SIOP status register (SSR) reports transfer completion (via the SPIF bit) and data collisions (via the DCOL bit) if SDR is read or written before SPIF is set.

The SIOP control register (SCR) enables the SIOP and configures it for master or slave mode. Baud rate is implementation dependent and, on some devices, may be fixed at some fraction of the internal MCU clock. Other devices have one or more rate control bits resident either in SCR or a mask option register (MOR) that permit alternative SIOP baud rates to be selected.



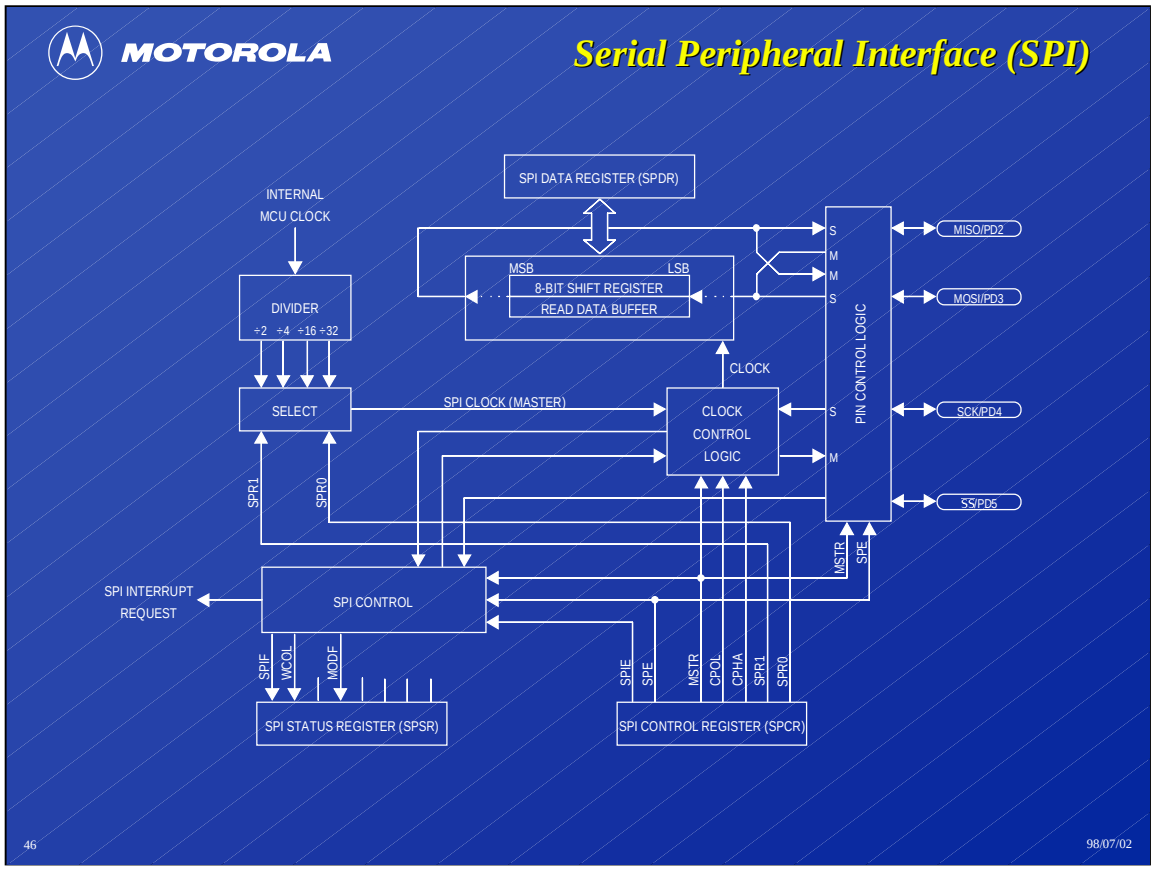
The cycling computer can display the following statistics on its Planar Standish 8-digit LCD: temperature, relative humidity, speed, distance traveled, pulse rate, and stopwatch time.

Motorola's MC14LC5003 is used to simplify the LCD interface. This device multiplexes 32 frontplanes with 4 backplanes to drive up to 128 segments. This is a perfect fit for the Planar Standish LCD which uses 32 frontplanes and 4 backplanes to display eight 15-segment alphanumeric characters.

The decision to use a serially interfaced LCD was prompted by the need to keep the cycling computer code as compact as possible. A microcontroller with built-in LCD drive capability would reduce component count, and possibly cost, but at the expense of requiring more complicated software to manage LCD updates.

On the other hand, a serially interfaced LCD can be managed with simpler, loop-oriented code that treats all updates identically. The resulting message handler is smaller and less complicated than the equivalent code for a direct drive LCD.

No single application can possibly demonstrate every available 68HC05 peripheral. Listed above are several other common 68HC05 peripherals that are covered in the following section for completeness.



The serial peripheral interface (SPI) is the full-duplex, synchronous data transfer mechanism upon which the simpler serial input/output port (SIOP) is based. Differences between the SIOP and the SPI are discussed below.

Master mode transfers at 1/2, 1/4, 1/16, or 1/32 of the internal MCU clock frequency are supported by the SPI. In slave mode, transfers are synchronized by the shift clock from the external master device and can occur at frequencies up to that of the internal MCU clock.

The SPI also supports four different transfer protocols. Each one is defined by a unique combination of the clock phase (CPHA) and clock polarity (CPOL) bits in the SPI control register (SPCR). Unless masked otherwise, the SIOP only supports the single CPOL = CPHA = 1 protocol.

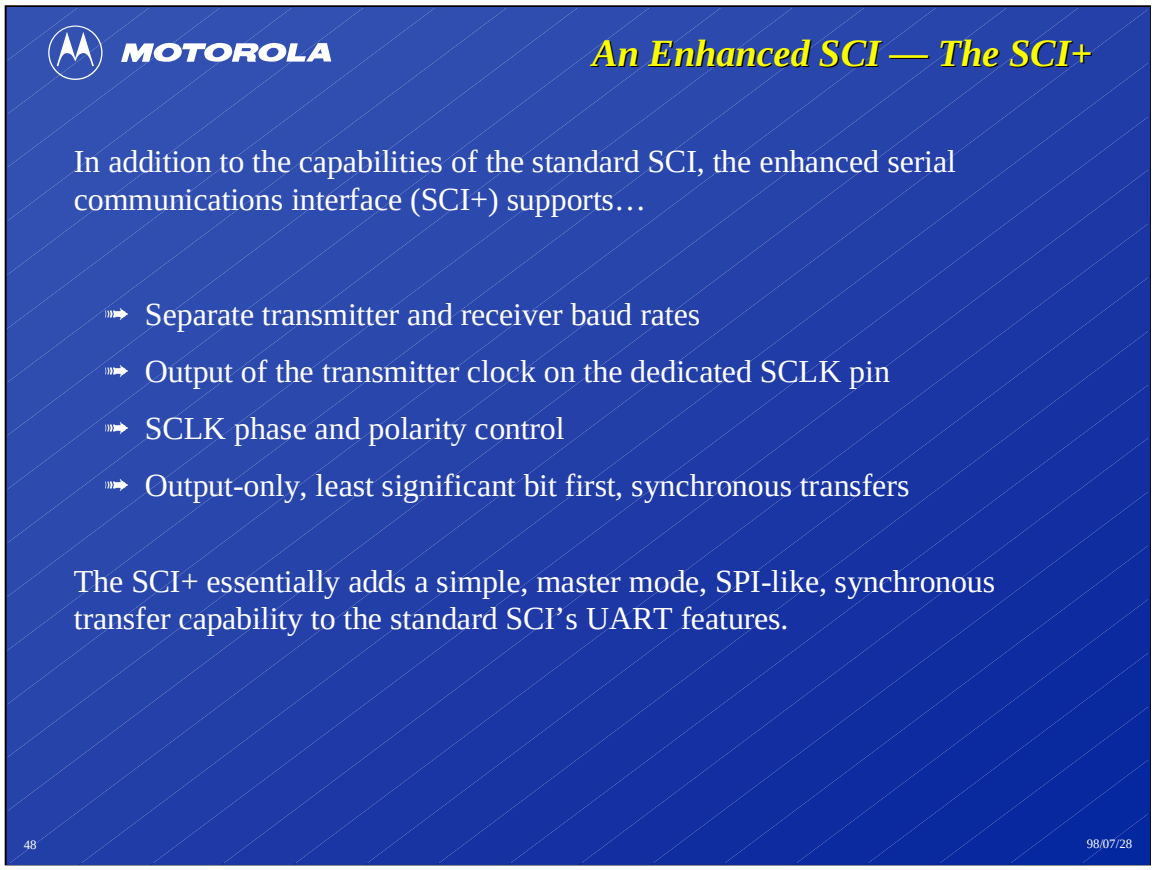
For proper operation in multiple master systems, SPI mode fault logic should be enabled. This is done by making the $\overline{SS}$ (slave select) pin an input when a device becomes the bus master. Normal transfers will take place as long as $\overline{SS}$ remains at logic one. When logic zero appears on $\overline{SS}$, the fault detection hardware on the current master will automatically disable the SPI subsystem, make all of its pins as inputs, and generate an SPI interrupt. This releases the bus for a new master. The SIOP has no such fault detection mechanism.

The serial communications interface (SCI) is the universal asynchronous receiver/transmitter (UART) peripheral on 68HC05 devices.

A single 8-bit register from which received data is read and to which data for transmission is written is shared by the receiver and transmitter sections of the SCI. Separate bits in SCI control register 1 (SCCR1) provide access to the ninth data bit when it is used for parity of address mark purposes.

Two prescaler bits allow the baud rate generator to run at the internal MCU clock frequency divided by 1, 3, 4, or 13. Three additional selection bits permit division of the prescaler output by 1, 2, 4, 8, 16, 32, 64, or 128. At internal MCU clock frequencies of 2 MHz and 4 MHz, the highest standard baud rates available are 9600 and 19200 baud, respectively.

Flags in the SCI status register (SCSR) report when the transmit data register is empty (TDRE), when a transmission is complete (TC), when the receive data register is full (RDRF), when the reciver goes idle (IDLE), and when receiver over-run (OR), framing (FE), and noise (NF) errors occur. Interrupts may be independently enabled for the TDRE, TC, and IDLE conditions. When enabled, the receiver interrupt is triggered by both RDRF and OR.

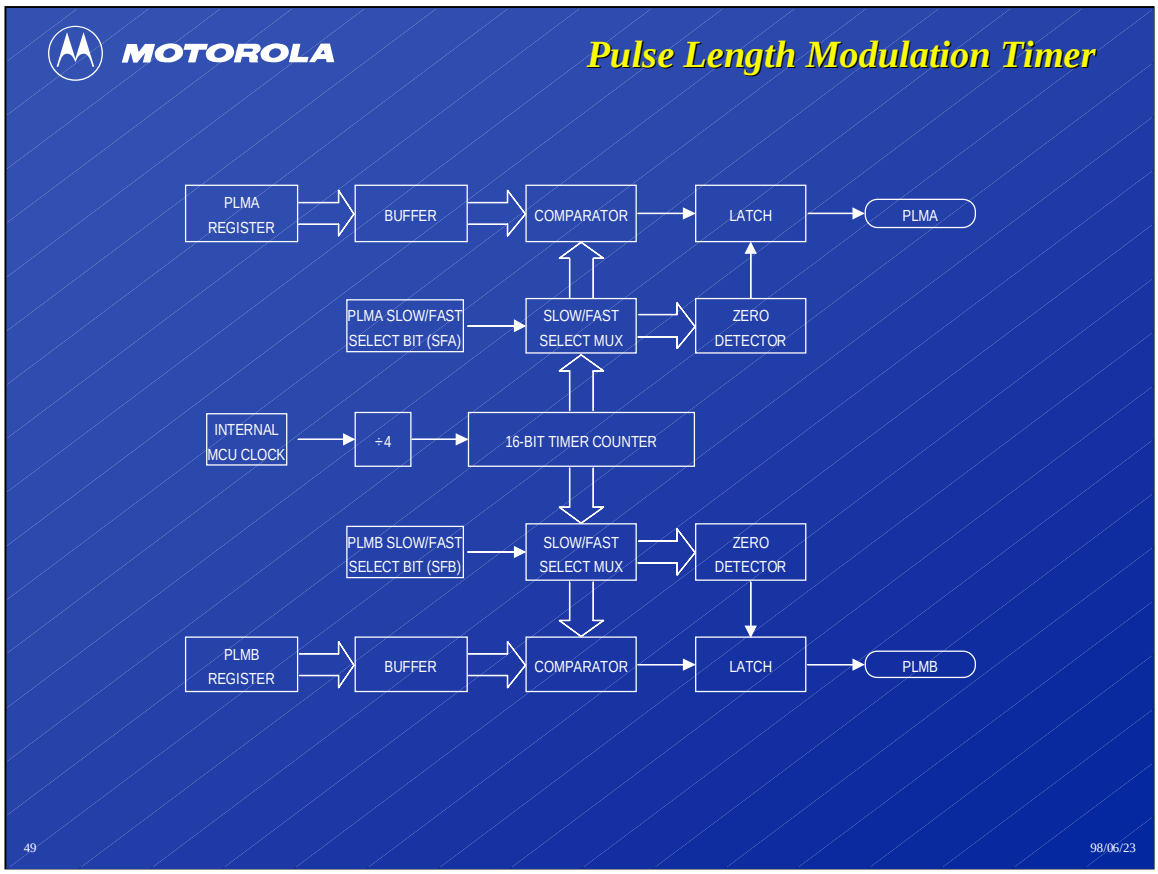


As noted above, the SCI+ can perform simple, output-only, synchronous transmissions in addition to its standard UART functions.

Like the regular SCI, the SCI+ has a prescaler that divides the internal MCU clock by 1, 3, 4, or 13. Three control bits on the standard SCI further divide this prescaler output by 1, 2, 4, 8, 16, 32, 64, or 128 to derive the transmitter and receiver clocks. On the SCI+, two sets of these bits provide the same division factors and allow independent baud rate selection for both the receiver and transmitter.

In addition to its asynchronous transfer capability, the SCI+ can transmit data synchronously by using the transmitter clock signal present on the dedicated SCLK pin. The synchronous transfer mode of the SCI+ is not, however, fully SPI-compliant. While it does have the obligatory CPOL and CPHA bits to control the polarity and phase of the shift clock on SCLK, the SCI+ performs all trannsfers in the same order as a UART — LSB first — which is opposite that used by the SPI.

Nonetheless, data placed in the proper bit order is transmitted in the same fashion on the SCI+ as it is on the SPI. Additionally, because a single data register is used for all transfers, the SCI+ can use 8- or 9-bit data words for synchronous as well as asynchronous transfers.

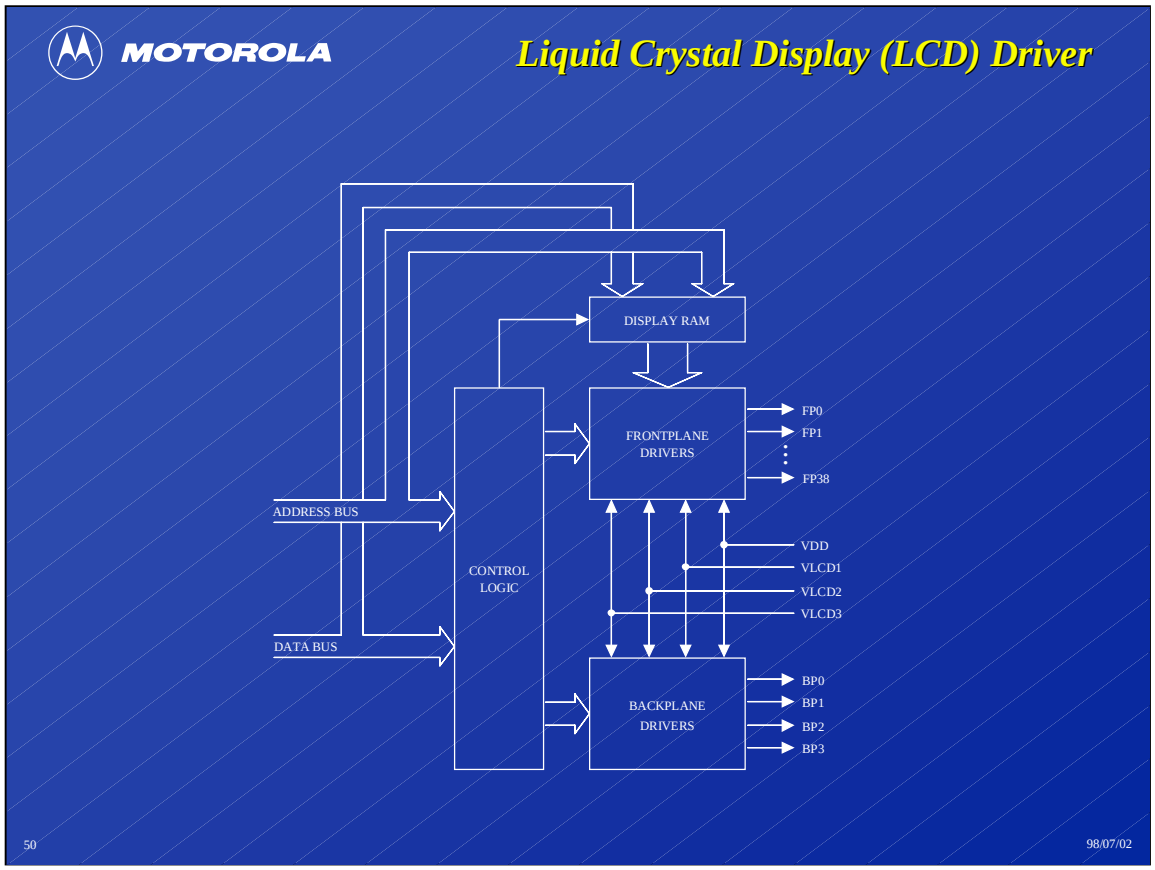


The pulse length modulation (PLM) timer has two channels, each with an 8-bit, buffered duty cycle register, an 8-bit comparator, zero detection circuitry, a slow/fast rate multiplexer, and a latch that drives an associated output pin.

PLM waveforms are output at one of two fixed frequencies and are active high for a user-specified length of time. A rate selection multiplexer allows each channel to choose a fast or slow 8-bit time base consisting, respectively, of bits [7:0] or [11:4] of the counter associated with the 16-bit timer subsystem.

In fast and slow modes, one count of the 8-bit time base is, respectively, 4 and 64 internal MCU clock cycles. The resulting output waveform periods are likewise $4 \times 256 = 1024$ and $64 \times 256 = 16384$ internal MCU clock cycles. Each PLM duty cycle register, just as its associated time base, is 8 bits wide.

Once a time base and duty cycle are selected, a PLM channel operates as follows. Starting at $00, the 8-bit time base increments at the user-specified fast or slow rate. When a non-zero duty cycle is in effect, the channel output pin is driven high and remains there until the duty cycle register value matches that of the 8-bit time base. The comparator detects this match and clears the output control latch, driving the associated PLM pin low. The waveform remains low until the 8-bit time base rolls over from $FF to $00, and the zero detection circuit sets the output control latch, again driving the PLM pin high.



All 68HC05 “L” family members have liquid crystal display (LCD) interfaces of varying degrees of sophistication. A representative implementation is the LCD driver on the popular MC68HC705L16, shown in the block diagram above.

Supporting as many as 39 frontplanes and up to 4 backplanes, the MC68HC705L16 can drive up to 156 segments. Bias voltages for this LCD interface are input directly on the VDD, VLCD1, VLCD2, and VLCD3 pins, typically from taps off a resistive ladder network. Bias levels of 1/1, 1/2, 1/3, and 1/3 are used when driving one, two, three, and four backplanes, respectively. Uncommitted backplane pins may be used as discrete outputs. Similarly, separate enable and multiplexer control bits also allow independent configuration of frontplanes [38:27] as discrete outputs.

The MC68HC705L16 has two other notable features. Like other 68HC05 “L” family devices, it has an alternate low-power oscillator from which LCD waveform timing can be derived. Additionally, when the LCD interface is disabled, the register locations that comprise display RAM are still accessible and can be used to store variables or prepare new messages for display at a later time.