NEWT Microprocessor

Technical Manual

Every effort has been made to ensure the accuracy of the information contained herein. If you find errors or inconsistencies please bring them to our attention. In all cases, however, the Verilog HDL source code for the NEWT design defines “proper operation”.

Notice:

“HP-41C”, “HP-41CV”, “HP-41CX” and “HP” are registered trademarks of Hewlett-Packard, Inc. All uses of these terms in this document are to be construed as adjectives, whether or not the noun “calculator”, “CPU” or “device” are actually present.

Acknowledgements:

This project never could have succeeded without Warren Furlow’s excellent Web site hp41.org. And even more important was his SDK41R6 software suite, for code development, and his V41 program for code debugging. Numerous people have answered my dumb questions on the Web site hpmuseum.org, and the book “Inside the HP-41” by Jean-Daniel Dodin was invaluable for getting a foothold on understanding the HP-41 operating system and register usage. Gene Wright was kind enough to be my voice at the HHC 2010 conference.

Index

Revision History

Date	Changes	Pages
01/01/2010	Complete review & update	
01/14/2010	Update module image list to avoid copyright issues	163-169
01/20/2010	clarify y-functions options	157-169
07/02/2010	Correct CLRST instruction description	69
07/19/2010	Change Turbo behavior for a number of instructions: ?LLD, DADD=C, DATA=C, REGn=C	24, 76, 77, 103
07/21/2010	Correct C=REGn instruction description. Update Turbo mode description. Update Keyboard Scanner description Update Memory Organization description	63 123-126 127-128 167-172
09/13/2010	“final” review and update	157-181
09/17/2010	Clarify serial function boundary cases	170-173
10/18/2010	Clarify reg_addr operation with C=REGN and REGN=C Update Light Sleep signal states Update appendices	63, 103 135 157-186
11/10/2010	Added instruction table. Miscellaneous clean-ups.	various
11/12/2010	Added MMU registers for pages 4, 6, and 7.	various
11/19/2010	Added ASTROUI image	167, 187
11/29/2010	Added DISASM-4C image Added 1-page option for SandMath image Changed ENBANKx operation description	Appendices Appendices 81
01/14/2011	Deleted Appendices to separate NEWT from 41CL	
04/04/2011	Updated power-up timing description.	148
04/06/2011	Updated Rate and Control registers to write-only	152, 153
04/12/2011	Updated Tx Status register.	152
07/15/2011	Updated peripheral registers	151, 154
08/22/2011	Updated Page mapping information Updated WCMD operation for special MMU enable/disable.	10, 12-13, 121
03/07/2012	Added description of bank registers for pages 6 and 7.	10-14
01/20/2013	Typos (p14), added Appendix	
02/08/2013	MMU Write command cannot be used for pages 0-3	12
09/22/2014	CHKKB, RSTKB instruction timing	153-154
06/27/2015	Typos (register names)	
06/29/2015	Typos about MMU enable bit	11
06/30/2015	Tx Buffer Empty status bit polarity	153

Introduction

The NEWT (Nut, Expanded, With Turbo) CPU is an upgraded version of the Hewlett-Packard Nut microprocessor, which was employed in a number of HP calculators, including the HP-41 series. Only publicly available documentation was used to create this design so there may be minor differences where the public documentation is misleading or lacking. The instruction set and register architecture are identical between the two designs.

This document should always be used as the final word on the operation of the NEWT microprocessor, but it is useful to refer to the Hewlett-Packard documentation if the description given here is too cryptic. The Nut architecture is over twenty years old, so we assume that it is already at least somewhat familiar to the reader.

As one would expect for a microprocessor targeted for handheld calculators, the NEWT architecture is highly optimized for BCD floating point math. The architecture includes three primary 56-bit (14 digit) registers, two auxiliary 56-bit registers, and one 8-bit general-purpose register. There are two 4-bit pointer registers that are used to select fields within the 56-bit registers. A Carry bit communicates carry information from one operation to the next, and there is a 14-bit general-purpose status register.

A dedicated keyboard scanner continuously scans a keyboard of up to seven rows and six columns. There is also an 8-bit serial status output and a 14-bit serial flag input. The Program Counter (PC) is sixteen bits wide and the instructions are ten bits wide. The program memory is logically separate from the data memory.

In the original Nut design all information was transferred serially to or from the CPU. One of the expanded features of the NEWT design is a traditional 16-bit parallel memory interface. This allows the use of standard word-wide memories to hold both program and data information. It also allows program information to be held in RAM memory, a feature that was not present in the original NUT architecture. This interface contains two Chip Selects, one for a FLASH memory and one for a RAM memory.

As mentioned previously, the PC is sixteen bits wide, providing 64k words of instruction memory. This instruction memory is divided into sixteen pages of 4k words each. In addition, many of these 4k pages can have up to four banks, only one of which is active at a time. In the NEWT design a Memory Management Unit (MMU) can be used to map each

page and bank into a physical address that is in either the Flash memory or the RAM memory.

One unique feature of the original Nut architecture was the ability to transfer control from the CPU to an intelligent peripheral that executes in-line instructions while the CPU is idled. This feature is still available in the NEWT design, even when operating in the Turbo mode.

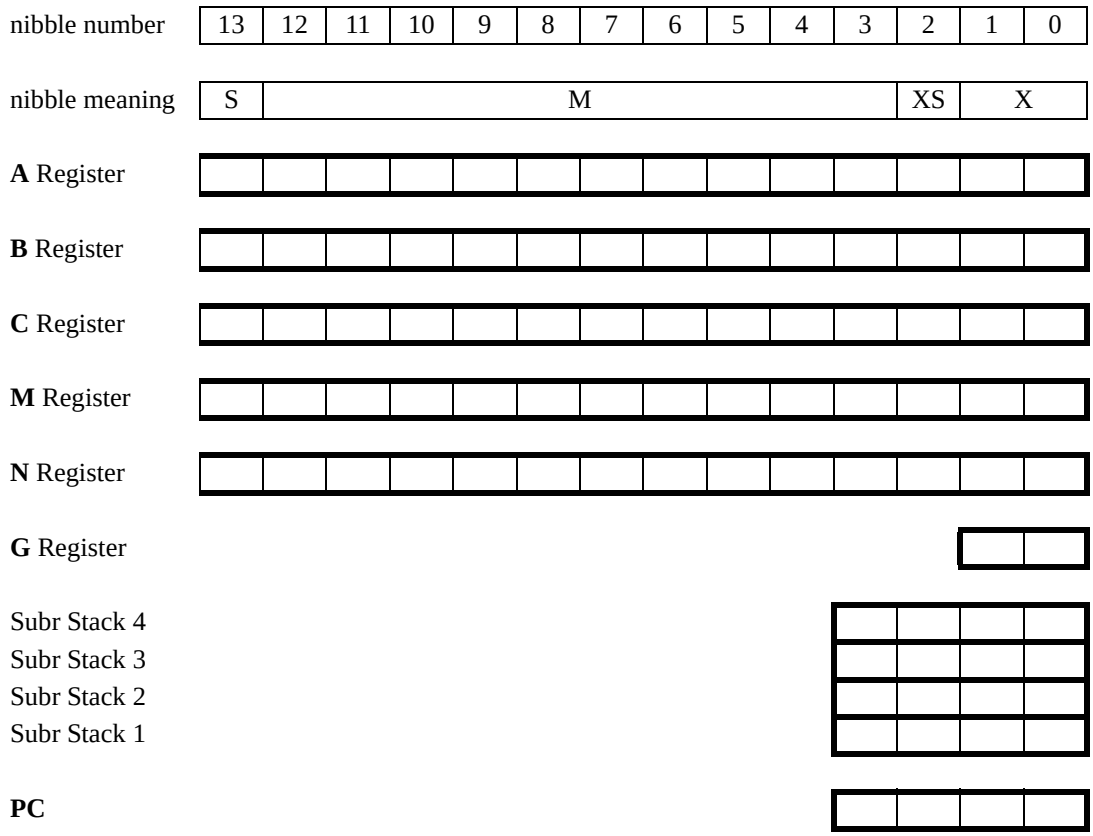
The Turbo mode speeds up the clock to the CPU by a factor of up to fifty, under program control. The serial Nut-compatible signals continue to be generated but the CPU executes via the new parallel memory bus for both instructions and data. Instructions that will affect the serial signals or that must be seen by an external peripheral or memory automatically slow down the CPU to normal speed to execute, preserving system compatibility. In addition, since the instruction is only ten bits wide, two of the unused bits in the 16-bit parallel memory devices are used to modify Turbo mode execution for things such as timing loops.

Throughout this document Nut-compatible pin signals will be denoted in bold (for example, **ph1**). It is important to remember, however, that the actual NEWT design must be surrounded by a virtual pad ring constructed out of discrete circuitry. This is necessary to maintain compatibility with the existing Nut hardware, which uses a 6V supply, and the NEWT hardware, which of necessity will use a lower voltage. This voltage translation is explained in detail in the External Interface chapter.

The NEWT design is partitioned into two sections. The first section contains the majority of the CPU and is designed to be powered down while not in use. The second section, which is much smaller, is continuously powered and controls the power-up and power-down for the CPU. If the NEWT design is implemented in programmable logic, the CPU will be implemented in an FPGA, while the second section will be implemented using a CPLD. If the NEWT design is implemented in a stand-alone chip both sections can be on the same chip, on separate power planes. The details of the power-up and power-down operation is explained in the Power Control chapter.

Programming Model

The NEWT register architecture contains five 56-bit (14 digit) general-purpose registers, called A, B, C, M and N. C is the primary register, and most operations involve this register as either one of the sources or as the destination. These 56-bit registers generally hold BCD floating point numbers, with a two digit exponent, one digit for the exponent sign, a ten-digit mantissa, and one digit for the mantissa sign.



Normally both the exponent and mantissa hold 10's-complement BCD numbers. However, the CPU can also be operated in Hexadecimal mode, where the full binary encoding is allowed in the registers, for binary data or alphanumeric data. In the Hexadecimal mode all arithmetic operations operate in binary fashion, with binary carries from bit-to-bit and

digit-to-digit. In the Decimal mode, BCD digits are handled, with all arithmetic results converted automatically to BCD, with BCD carries across digits as necessary.

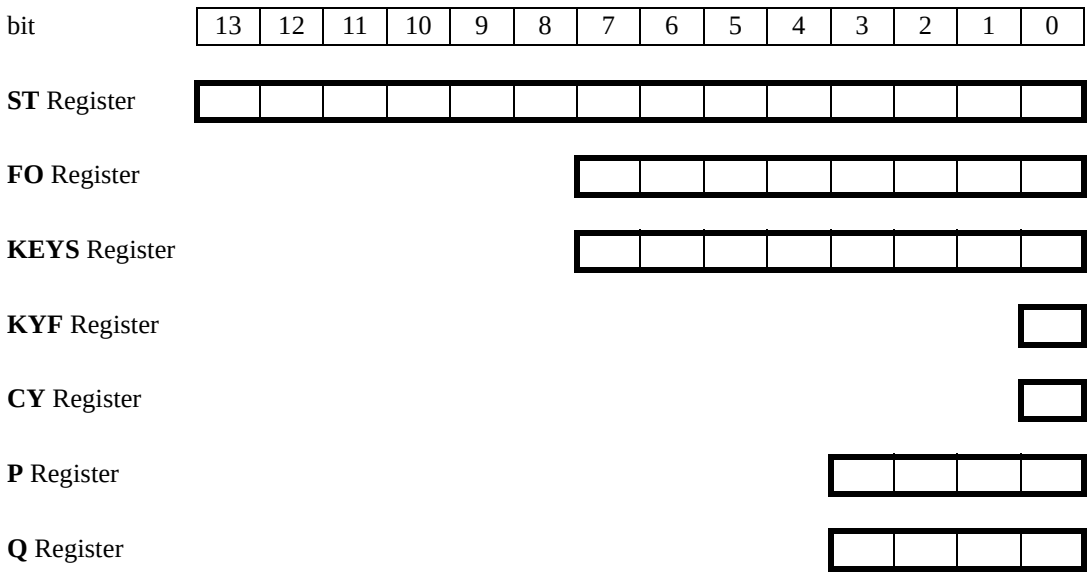
There is also an 8-bit G register that can be loaded, stored and exchanged with an two adjacent digits of the C register.

The Program Counter is 16 bits wide, and a four-level subroutine stack is included. This subroutine stack does not overflow to any external memory, so no more than four levels of nested subroutines can be handled without additional program overhead.

The architecture includes a number of other special-purpose registers. The Status (ST) register is fourteen bits wide and is usually accessed as individual bits. These bits can be used for program status. The Flag Output (FO) register is automatically shifted out serially during each instruction time. The individual bits can be latched externally or merely used to create a repeating waveform.

The Keyboard Flag (KYF) register is set automatically by the keyboard scanner when a keypress is detected. The scan code of the key being pressed is then loaded into the Keyboard (KEYS) register. The keyboard scanner handles multiple key presses and the KYF register operation aids in implementing rollover for the keys. The keyboard scanner operation is covered in more detail in the Keyboard Scanner chapter.

The Carry (CY) flag is a single bit that communicates carry information from one instruction to the next. This flag can also be tested for jumps and subroutine calls. Note that the state of the CY flag does not persist. If it is not explicitly set by an instruction, then it will always be cleared by an instruction. At the end of a peripheral operation, the selected peripheral can communicate information back to the CPU via the CY flag.

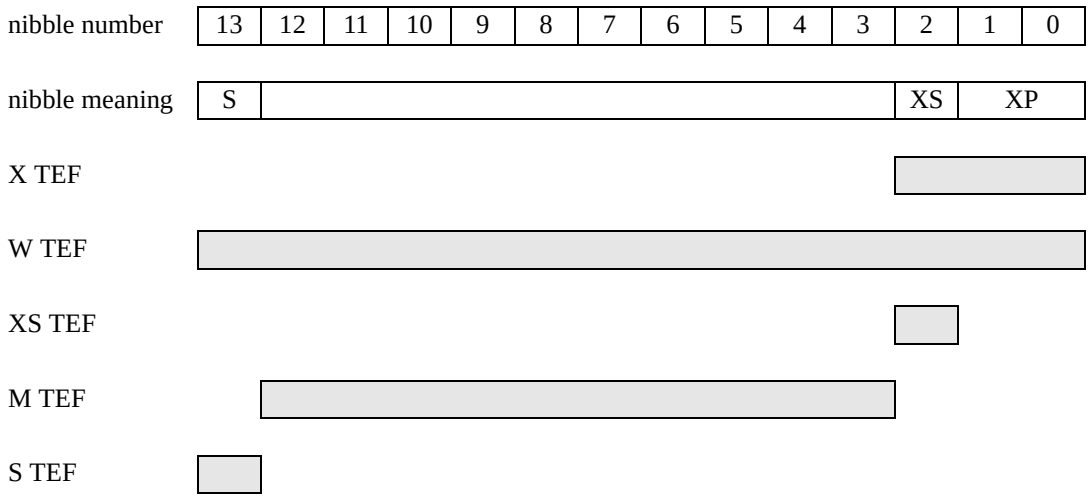


The P and Q registers are four-bit registers used to point to specific digits or ranges of digits in the general-purpose registers. When pointing to a specific digit only one of the registers is active at a time, but both are used when selecting a range of digits.

The CPU does not automatically operate on the entire floating-point number. Rather, software must handle individual fields explicitly. Hence the need for the Time Enable Field, or TEF. Most operations on the general-purpose registers include a TEF, to specify exactly which digits will be affected by the operation. All digits not selected by the TEF are unchanged. Eight different TEF values are encoded into those instructions that use this function according to the following table.

Encoding	Mnemonic	Meaning
000	PT	On Pointer
001	X	Exponent & Sign
010	WPT	Word Through Pointer
011	W	Whole Word
100	PQ	Pointer P through Pointer Q
101	XS	Exponent Sign
110	M	Mantissa Only
111	S	Mantissa Sign

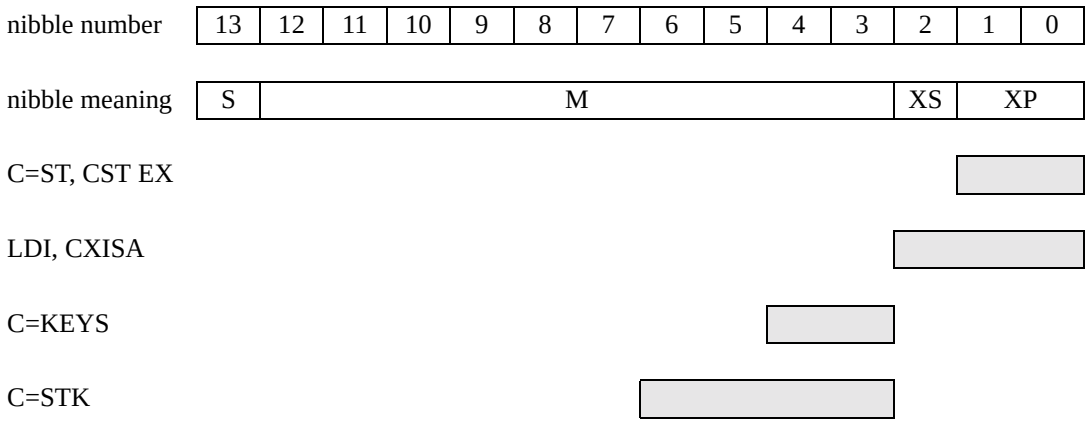
Most of these TEF values are straightforward, selecting one or more of the floating point fields, as shown in the figure below:



The three remaining TEF values use the pointers. The On Pointer (PT) value selects only the digit pointed at by the current pointer. Word Through Pointer (WPT) selects all digits from digit 0 through the digit pointed at by the current pointer.

Pointer P through Pointer Q (PQ) operates as would be expected when P points to a digit lower than the digit pointed at by Q, selecting digit P through digit Q, inclusive. If P and Q are equal only that digit is selected. If P is greater than Q the value of Q is ignored and digits starting with the digit selected by P through the end of the word are selected.

In addition to these Time Enable Fields, there are a few other instructions that use specific digit times. These are shown in the figure below:



Memory Organization

The memory organization of the NEWT microprocessor must be viewed from two different perspectives. The first view is the memory organization of the original Nut microprocessor. This is the logical memory organization and consists of separate program and data memory spaces. The second view is the native memory organization of the NEWT microprocessor itself. This is the physical memory organization and consists of a unified memory space divided in half by the two available chip selects. The logical memory usually maps into the physical memory.

The logical memory is what is normally visible to the programmer. It consists of a 64K by 10-bit program memory space and a 4K by 56-bit “register” memory space. Program memory is accessed via the bidirectional serial **isa_bus** signal, which carries both the instruction address and the instruction data. Register memory is accessed via the bidirectional serial **data_bus** signal for the data, which communicates with the C register. Register addresses are handled differently, with the register address being latched in a dedicated location external to the processor. The program memory is normally read-only and the register memory is read/write.

The 4K by 56-bit register memory is used for data, alarms or user programs. However, from the point of view of the microprocessor, register memory is just read/write data memory. In the NEWT microprocessor, all register memory is mapped automatically into a particular region of physical memory. This mapping is completely transparent to the user. Note that even though the NEWT microprocessor properly handles 4096 registers, existing 41C Operating System software only provides for accessing 1024 registers.

In the original 41C system, register addresses were stored in the register memory chips. The NEWT design stores the register address in a reserved physical memory location. The contents of this memory location, called reg_addr, are retrieved from physical memory each time a register is accessed.

The 64K by 10-bit program memory is divided into sixteen 4k “pages” using the upper four bits of the address. Some of these pages support up to four “banks”. A number of pages are dedicated to particular software functions in the HP-41C series of calculators. This is shown in the table below, along with the bank information. Note that the 8K “Ports” allow the use of external plug-in modules for software, and the Card Reader peripheral always uses Page E in Port 4.

Logical Address MSB	Logical Page	Banks	HP-41C function
0000-0010	0-2	one	Operating System
0011	3	one	Extended Functions
0100	4	one	Service Module
0101	5	four	Time Module and Extended Functions
0110	6	four	Printer
0111	7	four	HP-IL
1000-1001	8-9	four	Port 1
1010-1011	A-B	four	Port 2
1100-1101	C-D	four	Port 3
1110-1111	E-F	four	Port 4

The physical memory in the NEWT microprocessor is 16M by 16 bits. This physical memory is divided into two 8M regions, with a separate chip select for each region. The lower half of physical memory uses **MEMCS0B** and is normally connected to a Flash memory device. The upper half uses **MEMCS1B** and is normally connected to a RAM memory device.

The register memory is automatically mapped to the 16K word section of physical memory starting at address 0x800000. This places it at the bottom of RAM memory. The register address is stored in physical memory location 0x804000. The 56-bit register data is automatically packed and unpacked into this 16K word region so that it appears to be the normal register memory. Certain areas of program memory are also automatically mapped to physical memory as shown in the table below.

Logical Page	Bank	Physical Address [23:12]
0		normally 0x000
1		normally 0x001
2		normally 0x002
3		normally 0x003
4		from MMU (if enabled)
5	all four	normally bank 1: 0x006 normally banks 2, 3 & 4: 0x005
6		from MMU (if enabled)
7		0x007 if MMU disabled; otherwise from MMU

8	all four	from MMU (if enabled)
9	all four	from MMU (if enabled)
A	all four	from MMU (if enabled)
B	all four	from MMU (if enabled)
C	all four	from MMU (if enabled)
D	all four	from MMU (if enabled)
E	all four	from MMU (if enabled)
F	all four	from MMU (if enabled)

Logical pages 0-3 and 5 are normally reserved for the Operating System. Enabling the Special MMU operation turns on the translation circuitry for those pages, but only if the the regular MMU operation is also enabled.

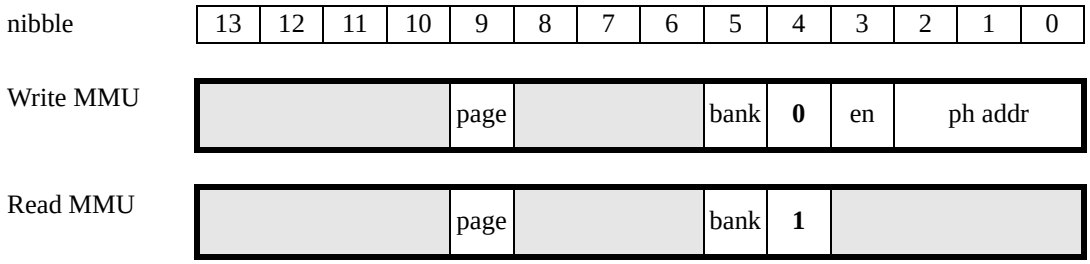
When program memory is mapped to physical memory the ten bits of actual program data are right-justified in the sixteen bits of physical memory. Bits 15-14 and 11-10 are unused and bits 13 and 12 are used by the Turbo control logic to control execution speed on an instruction-by-instruction basis. This operation is explained in the Turbo Mode chapter.

The NEWT-specific WCMD instruction is used to write to either the physical program memory or to the MMU registers, using the contents of the C register for the actual command, address and data information. The command in digit 4 determines the destination of the write data. These commands are described below. When a bank selection is required, it is encoded into digit 5 according to the following table:

Nibble 5	Memory bank selected
xx00	bank 1
xx01	bank 3
xx10	bank 2
xx11	bank 4

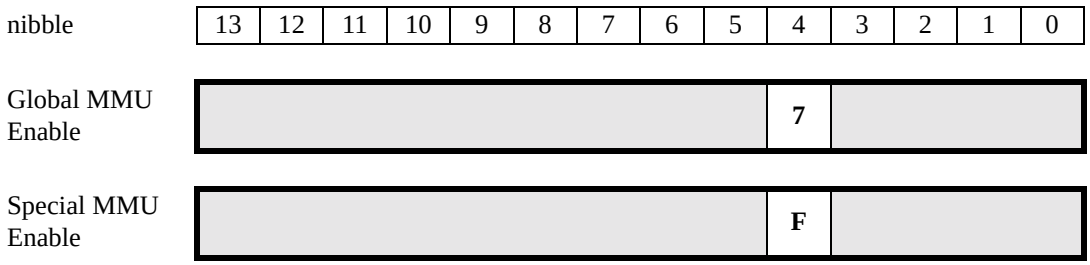
A command value of 0000 identifies a write to an MMU register to enable or disable address translation for a particular page and bank. A command value of 0001 identifies a read of an MMU register. The specific MMU register is selected by the combination of the page value in digit 9 and the bank select value in digit 5. A one in the most-significant bit of digit 3 enables address translation, while a zero in this bit disables address translation. The twelve bits of the physical address that will be substituted for the page value in the selected bank is contained in digits 2-0. Read data is latched by the command and must be accessed using the on-chip I/O Port. Programming the MMU requires some care. The

MMU entry for a particular page/bank should never be updated while executing from that page/bank. Doing so will switch the instruction fetch to another region of physical memory, which may lead to unexpected results.



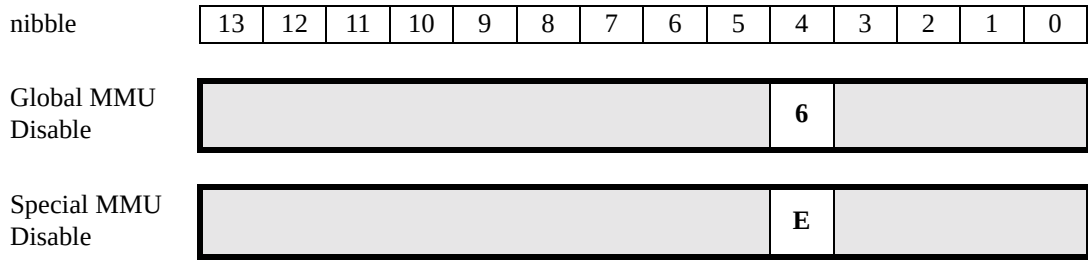
Writes to MMU registers for pages 0-3 are ignored, and these MMU registers must be accessed using the Write Physical Address command. Disabled pages (and/or banks) are not fetched from physical memory, but are fetched from external memory via the **isa_bus**. Disabled pages are always fetched at normal speed. Note that on first power-up the entire MMU is disabled because the MMU registers have not been initialized. Once the MMU has been completely initialized, a WCMD instruction with a command value of 0111 globally enables MMU translation.

A WCMD instruction with a command value of 1111 enables the special MMU translation circuitry, which maps logical pages 0-3 and 5. Note that the special MMU enable information is volatile. That is, the enabled state is lost when the NEWT is powered down.



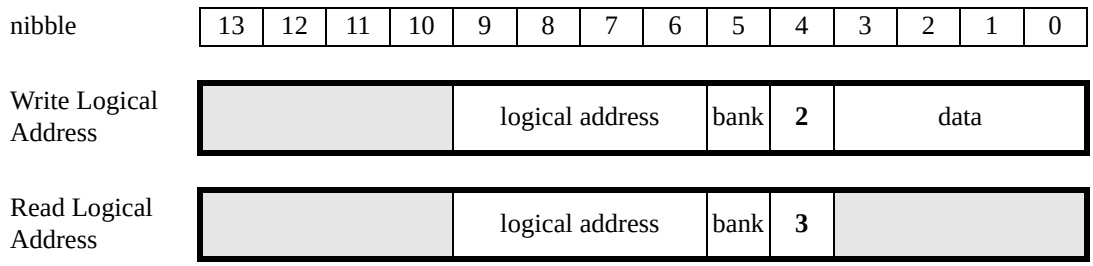
It is also possible to globally disable MMU translation. A WCMD instruction with a command value of 0110 globally disables MMU translation. Both of these commands are buffered, and do not take effect until the Program Counter returns to one of the Operating System pages (0, 1 or 2). This function is necessary to protect the software that enables or disables the MMU from having “the rug yanked out from underneath it” when the MMU programming changes.

A WCMD instruction with a command value of 1110 disables the special MMU translation circuitry, returning pages 0-3 and 5 to their normal locations in Flash memory.



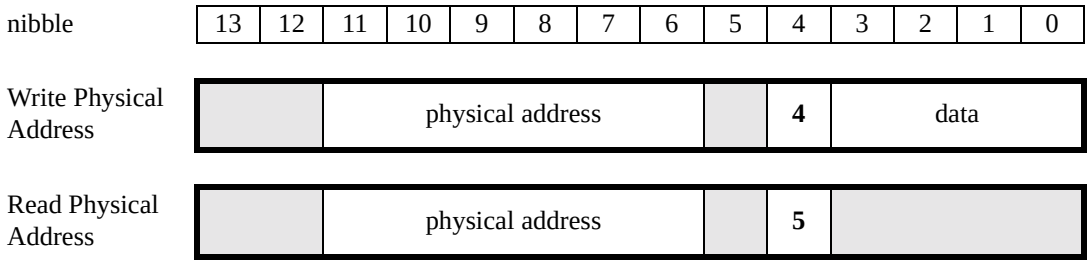
A command value of 0010 identifies a write to a logical address, while a command value of 0011 identifies a read from a logical address. The logical address is specified in digits 9-6 and the bank select value in digit 5. This logical address is automatically translated to a physical address and for a write all 16 bits of data in digits 3-0 are written to the external word-wide memory.

For a read the memory data is latched for and can be read via the on-chip I/O Port. Normally the write command will be used to write to the external RAM, but it can also be used to write to the external Flash memory.



A command value of 0100 identifies a write directly to a physical address, while a command of 0101 identifies a read from a physical address. The physical address specified in digits 11-6. For a write all 16 bits of data in digits 3-0 are written to the external word-wide memory.

For a read the memory data is latched for and can be read via the on-chip I/O Port. Normally the write command will be used to write to the external RAM, but it can also be used to write to the external Flash memory.



Most of the remaining command values are used to control the turbo mode. This is detailed in the Turbo Mode chapter.

The NEWT design automatically latches the bank change information by monitoring the CPU instruction execution. Even though the MMU mapping occurs as a function of the current 4K page and selected bank, there are actually only seven bank select registers. This is consistent with the original HP-41C implementation of the memory bank function. The seven sets of bank select registers are for Page 5, Page 6, Page 7, Port 1 (Pages 8 and 9), Port 2 (Pages A and B), Port 3 (Pages C and D) and Port 4 (Pages E and F). Even though a pair of 4K pages may share a bank select register, each individual 4K page and bank select combination is individually mapped into the physical memory by the MMU. If a particular 4K page does not require separate mapping for each bank, the MMU translation for each bank should still be programmed, in this case pointing to the same 4K page in physical memory.

Note that the MMU registers are not stored in the NEWT microprocessor itself. Rather, they are stored in the 4K page of physical memory starting at address 0x804000. This allows the NEWT microprocessor to be powered down while retaining all data stored in the external RAM memory.

Both the register memory area of physical memory (at physical addresses 0x800000-0x803FFF) and the MMU area of physical memory (at physical addresses 0x804000-0x804FFF) are managed automatically by the NEWT microprocessor itself. Care should be exercised accessing these areas via a user programs.

Instruction Set

This chapter presents the NEWT instruction set, including the assembly language syntax, operands, Carry flag settings, binary encoding, and execution time. The entire instruction set, including those peripheral instructions that affect the CPU, is presented in alphabetical order. Many of the restrictions present in the original NUT instruction set have been eliminated and these differences will be listed in notes at the end of individual instruction descriptions.

All unused binary encodings execute as NOP as far as the CPU is concerned. However, a number of these encodings are interpreted by logic external to the CPU. In some cases this logic is part of the NEWT design, and in other cases the logic is external to the NEWT design. Unused opcodes always execute at normal bus speed for compatibility.

The assembly language syntax is shown here identical to that used by the original HP assembler. At least two other sets of assembly language syntax exist, but no attempt will be made here to support them.

The operation of each instruction is specified in a format similar to Verilog HDL for minimum ambiguity, but no descriptive text or examples are included. In these descriptions bits and bit fields are enclosed in square brackets [and]. In addition, digits (nibbles) and digit fields ar enclosed in triangular brackets < and >. In those cases where more than one operation takes place concurrently, such as in register exchanges, the individual operations that occur simultaneously are enclosed by a Verilog fork-join pair.

The effect of the instruction on the Carry flag is listed separately. In addition, the effect of the two operating modes, Decimal/Hexadecimal and Turbo, are also listed.

Fields in the instruction are listed using shortcuts for common fields. These shortcuts should be self-explanatory in most cases, but will be detailed here for completeness.

The most common field in the instruction specifies a Time Enable Field (shown as TEF), employing the following shortcuts:

Encoding	Mnemonic	Meaning
000	PT	On Pointer
001	X	Exponent & Sign
010	WPT	Word Through Pointer
011	W	Whole Word
100	PQ	Pointer P through Pointer Q
101	XS	Exponent Sign
110	M	Mantissa Only
111	S	Mantissa Sign

The second common field (shown as dddd) selects one of the fourteen digits or digit times during execution. Thus this field has only fourteen valid values, leaving two unused or illegal values. Often these two illegal values will be used to encode an entirely different instruction. Hence the apparent overlap of instruction encodings. The valid encodings are shown in the table below.

Encoding	Mnemonic	Digit
0000	3	3 (Mantissa digit 0)
0001	4	4 (Mantissa digit 1)
0010	5	5 (Mantissa digit 2)
0011	10	10 (Mantissa digit 7)
0100	8	8 (mantissa digit 5)
0101	6	6 (Mantissa digit 3)
0110	11	11 (Mantissa digit 8)
0111		illegal encoding
1000	2	2 (Exponent Sign digit)
1001	9	9 (Mantissa digit 6)
1010	7	7 (Mantissa digit 4)
1011	13	13 (Mantissa sign digit)
1100	1	1 (Exponent digit 1)
1101	12	12 (Mantissa digit 9)
1110	0	0 (Exponent digit 0)
1111		illegal encoding

The third common field (shown as nnnn) selects one of sixteen values, registers (in the NUT sense) or peripherals, using standard binary encoding.

?A#0

Test A Not Equal To Zero

?A#0 operand: Time Enable Field

Operation: CY <= (A<time_enable_field> != 0)

Flag: Set if A is not zero at any time during the Time Enable Field; cleared otherwise.

Dec/Hex: Independent
Turbo: Independent

	Assembly Syntax	Encoding	Machine Cycles
	?A#0 TEF	1101_0TEF_10	1

?A#C

Test A Not Equal To C

?A#C operand: Time Enable Field

Operation: CY <= (A<time_enable_field> != C<time_enable_field>)

Flag: Set if A is not equal to C at any time during the Time Enable Field; cleared otherwise.

Dec/Hex: Independent
Turbo: Independent

	Assembly Syntax	Encoding	Machine Cycles
	?A#C TEF	1101_1TEF_10	1

?A<B

Test A Less Than B

?A<B operand: Time Enable Field

Operation: CY <= (A<time_enable_field> < B<time_enable_field>)

Flag: Set if A is less than B, for the bits during the Time Enable Field; cleared otherwise.

Dec/Hex: Independent
Turbo: Independent

	Assembly Syntax	Encoding	Machine Cycles
	?A<B TEF	1100_1TEF_10	1

?A<C

Test A Less Than C

?A<C operand: Time Enable Field

Operation: CY <= (A<time_enable_field> < C<time_enable_field>)

Flag: Set if A is less than C, for the bits during the Time Enable Field; cleared otherwise

Dec/Hex: Independent
Turbo: Independent

	Assembly Syntax	Encoding	Machine Cycles
	?A<C TEF	1100_0TEF_10	1

?B#0

Test B Equal To Zero

?B#0 operand: Time Enable Field

Operation: CY <= (B<time_enable_field> != 0)

Flag: Set if B is not zero at any time during the Time Enable Field; cleared otherwise.

Dec/Hex: Independent
Turbo: Independent

	Assembly Syntax	Encoding	Machine Cycles
	?B#0 TEF	1011_0TEF_10	1

?C#0

Test C Equal To Zero

?C=0 operand: Time Enable Field

Operation: CY <= (C<time_enable_field> != 0)

Flag: Set if C is not zero at any time during the Time Enable Field; cleared otherwise.

Dec/Hex: Independent
Turbo: Independent

	Assembly Syntax	Encoding	Machine Cycles
	?C#0 TEF	1011_1TEF_10	1

?Fd=1

Test Flag Input Equal to One

?Fd=1 operand: Digit Number

Operation: CY <= FI<digit>

Flag: Set/Cleared to match the selected Flag Input

Dec/Hex: Independent
Turbo Automatically fetched and executed at normal bus speed

	Assembly Syntax	Encoding	Machine Cycles
	?Fd=1	dddd_1011_00	1

Note: The flag input, **fi_bus**, is sampled near the middle of the appropriate digit time (on the falling edge of the second **ph1** clock) during the execution phase of this instruction. The following flags are currently used in an HP-41 system:

flag number	device	Mnemonic	Used for
0	82143A 82242A	?PBSY	Printer Busy
1	82104A	?CRDR	Card Reader
2	82153A	?WNDB	Wand Byte Available
5	82242A	?EDAV	Emitter Diode Available
6	82160A	?IFCR	Interface Clear Received
7	82160A	?SRQR	Service Request Received
8	82160A	?FRAV	Frame Available
9	82160A	?FRNS	Frame Received Not As Sent
10	82160A	?ORAV	Output Register Available
12	82182A	?ALM	Alarm
13	all	?SER	Service Request

?LLD

Low Level Detect

?LLD operand: none

Operation: CY <= low_battery_status

Flag: Set if the **lld** input is Low, signaling a Low Battery; cleared otherwise.

Dec/Hex: Independent
Turbo: Independent

	Assembly Syntax	Encoding	Machine Cycles
	?LLD	0101_1000_00	1

?P=Q

Test P Equal To Q

?P=Q operand: none

Operation: CY <= (P == Q)

Flag: Set if P equals Q; cleared otherwise.

Dec/Hex: Independent
Turbo: Independent

	Assembly Syntax	Encoding	Machine Cycles
	?P=Q	0100_1000_00	1

?PFLGn=1

Test Peripheral Flag N

?PFLGn=1 operand: none

Operation: CY <= Peripheral Flag n

Flag: Set/Cleared as a result of the test

Dec/Hex: Independent
Turbo: Automatically executed at normal bus speed.

	Assembly Syntax	Encoding	Machine Cycles
	?PFLGn=1	nnnn_0000_11	1

Note: This instruction is only available while a peripheral is in control of the bus, and automatically returns control of the bus to the CPU. Peripherals may have up to sixteen flags to communicate information to the CPU via this instruction. The active peripheral drives the state of the selected flag on the **isa_bus** during the first clock cycle of the following bus cycle. Thus the next instruction, which will be executed by the CPU, should be a conditional branch. This next instruction will always be executed at normal bus speed to allow the active peripheral time to respond.

?PT=

Test Pointer Equal To

?PT=	operand: Digit Number
Operation:	CY <= (PT == digit)
Flag:	Set if the pointer is equal to the dddd field in the instruction; cleared otherwise.
Dec/Hex:	Independent
Turbo:	Independent

	Assembly Syntax	Encoding	Machine Cycles
	?PT= d	dddd_0101_00	1

Note: In the original NUT implementation this instruction cannot immediately follow an arithmetic (type 10) instruction. This restriction is not present in the NEWT implementation.

ASL

Shift Left A

ASL	operand: Time Enable Field
Operation:	A<time_enable_field> <= A<time_enable_field> << 1
Flag:	Cleared
Dec/Hex:	Independent (no decimal adjust)
Turbo:	Independent

	Assembly Syntax	Encoding	Machine Cycles
	ASL TEF	1111_1TEF_10	1

Note: Zero is shifted into the least-significant (time-enabled) digit.

ASR

Shift Right A

ASR operand: Time Enable Field

Operation: A<time_enable_field> <= A<time_enable_field> >> 1

Flag: Cleared

Dec/Hex: Independent (no decimal adjust)
Turbo: Independent

	Assembly Syntax	Encoding	Machine Cycles
	ASR TEF	1110_0TEF_10	1

Note: Zero is shifted into the most-significant (time-enabled) digit.

A=0

Clear A

A=0 operand: Time Enable Field

Operation: A<time_enable_field> <= 0

Flag: Cleared

Dec/Hex: Independent
Turbo: Independent

	Assembly Syntax	Encoding	Machine Cycles
	A=0 TEF	0000_0TEF_10	1

A=A+1

Increment A

A=A+1 operand: Time Enable Field

Operation: {CY, A<time_enable_field>} <= A<time_enable_field> + 1

Flag: Set/Cleared as a result of the operation

Dec/Hex: Decimal adjusted in Decimal Mode
Turbo: Independent

	Assembly Syntax	Encoding	Machine Cycles
	A=A+1 TEF	0101_1TEF_10	1

A=A+B

Load A With A + B

A=A+B operand: Time Enable Field

Operation: {CY, A<time_enable_field>} <= A<time_enable_field> + B<time_enable_field>

Flag: Set/Cleared as a result of the operation

Dec/Hex: Decimal adjusted in Decimal Mode
Turbo: Independent

	Assembly Syntax	Encoding	Machine Cycles
	A=A+B TEF	0100_1TEF_10	1

A=A+C

Load A With A + C

A=A+C operand: Time Enable Field

Operation: {CY, A<time_enable_field>} <= A<time_enable_field> + C<time_enable_field>

Flag: Set/Cleared as a result of the operation

Dec/Hex: Decimal adjusted in Decimal Mode
Turbo: Independent

	Assembly Syntax	Encoding	Machine Cycles
	A=A+C TEF	0101_0TEF_10	1

A=A-1

Decrement A

A=A-1 operand: Time Enable Field

Operation: {CY, A<time_enable_field>} <= A<time_enable_field> - 1

Flag: Set/Cleared as a result of the operation

Dec/Hex: Decimal adjusted in Decimal Mode
Turbo: Independent

Assembly Syntax	Encoding	Machine Cycles
A=A-1 TEF	0110_1TEF_10	1

A=A-B

Load A With A - B

A=A-B operand: Time Enable Field

Operation: {CY, A<time_enable_field>} <= A<time_enable_field> - B<time_enable_field>

Flag: Set/Cleared as a result of the operation

Dec/Hex: Decimal adjusted in Decimal Mode
Turbo: Independent

	Assembly Syntax	Encoding	Machine Cycles
	A=A-B TEF	0110_0TEF_10	1

A=A-C

Load A With A - C

A=A-B operand: Time Enable Field

Operation: {CY, A<time_enable_field>} <= A<time_enable_field> - C<time_enable_field>

Flag: Set/Cleared as a result of the operation

Dec/Hex: Decimal adjusted in Decimal Mode
Turbo: Independent

	Assembly Syntax	Encoding	Machine Cycles
	A=A-C TEF	0111_0TEF_10	1

A=C

Load A From C

A=C operand: Time Enable Field

Operation: A<time_enable_field> <= C<time_enable_field>

Flag: Cleared

Dec/Hex: Independent
Turbo: Independent

	Assembly Syntax	Encoding	Machine Cycles
	A=C TEF	0100_0TEF_10	1

ABEX

Exchange A and B

ABEX operand: Time Enable Field

Operation:
```
fork
  A<time_enable_field> <= B<time_enable_field>
  B<time_enable_field> <= A<time_enable_field>
  join
```

Flag: Cleared

Dec/Hex: Independent
Turbo: Independent

	Assembly Syntax	Encoding	Machine Cycles
	ABEX TEF	0001_1TEF_10	1

ACEX

Exchange A and C

ACEX operand: Time Enable Field

Operation:
```
fork
  A<time_enable_field> <= C<time_enable_field>
  C<time_enable_field> <= A<time_enable_field>
  join
```

Flag: Cleared

Dec/Hex: Independent
Turbo: Independent

	Assembly Syntax	Encoding	Machine Cycles
	ACEX TEF	0010_1TEF_10	1

BSR

Shift Right B

BSR	operand: Time Enable Field
Operation:	B<time_enable_field> <= B<time_enable_field> >> 1
Flag:	Cleared
Dec/Hex:	Independent (no decimal adjust)
Turbo:	Independent

	Assembly Syntax	Encoding	Machine Cycles
	BSR TEF	1110_1TEF_10	1

Note: Zero is shifted into the most-significant (time-enabled) digit.

B=0

Clear B

B=0 operand: Time Enable Field

Operation: B<time_enable_field> <= 0

Flag: Cleared

Dec/Hex: Independent
Turbo: Independent

	Assembly Syntax	Encoding	Machine Cycles
	B=0 TEF	0000_1TEF_10	1

B=A

Load B From A

B=A operand: Time Enable Field

Operation: B<time_enable_field> <= A<time_enable_field>

Flag: Cleared

Dec/Hex: Independent
Turbo: Independent

	Assembly Syntax	Encoding	Machine Cycles
	B=A TEF	0010_0TEF_10	1

BCEX

Exchange B and C

BCEX operand: Time Enable Field

Operation:
```
fork
  B<time_enable_field> <= C<time_enable_field>
  C<time_enable_field> <= B<time_enable_field>
  join
```

Flag: Cleared

Dec/Hex: Independent
Turbo: Independent

	Assembly Syntax	Encoding	Machine Cycles
	BCEX TEF	0011_1TEF_10	1

CSR

Shift Right C

CSR operand: Time Enable Field

Operation: C<time_enable_field> <= C<time_enable_field> >> 1

Flag: Cleared

Dec/Hex: Independent
Turbo: Independent

	Assembly Syntax	Encoding	Machine Cycles
	CSR TEF	1111_0TEF_10	1

Note: Zero is shifted into the most-significant (time-enabled) digit.

C=0

Clear C

C=0 operand: Time Enable Field

Operation: C<time_enable_field> <= 0

Flag: Cleared

Dec/Hex: Independent
Turbo: Independent

	Assembly Syntax	Encoding	Machine Cycles
	C=0 TEF	0001_0TEF_10	1

C=A+C

Load C With A + C

C=A+C operand: Time Enable Field

Operation: {CY, C<time_enable_field>} <= A<time_enable_field> + C<time_enable_field>

Flag: Set/Cleared as a result of the operation

Dec/Hex: Decimal adjusted in Decimal Mode
Turbo: Independent

	Assembly Syntax	Encoding	Machine Cycles
	C=A+C TEF	1000_0TEF_10	1

C=A-C

Load C With A - C

C=A-C operand: Time Enable Field

Operation: {CY, C<time_enable_field>} <= A<time_enable_field> - C<time_enable_field>

Flag: Set/Cleared as a result of the operation

Dec/Hex: Decimal adjusted in Decimal Mode
Turbo: Independent

	Assembly Syntax	Encoding	Machine Cycles
	C=A-C TEF	1001_0TEF_10	1

C=B

Load C From B

C=B operand: Time Enable Field

Operation: C<time_enable_field> <= B<time_enable_field>

Flag: Cleared

Dec/Hex: Independent
Turbo: Independent

	Assembly Syntax	Encoding	Machine Cycles
	C=B TEF	0011_0TEF_10	1

C=-C

Negate C

C=-C	operand: Time Enable Field
Operation:	{CY, C<time_enable_field>} <= 0 - C<time_enable_field>
Flag:	Set/Cleared as a result of the operation
Dec/Hex:	10’s complement in Decimal Mode, 16’s complement in Hexadecimal Mode
Turbo:	Independent

	Assembly Syntax	Encoding	Machine Cycles
	C=-C TEF	1010_0TEF_10	1

Note: This is the arithmetic complement, or the value subtracted from zero.

C=-C-1

Complement C

C=-C-1 operand: Time Enable Field

Operation: {CY, C<time_enable_field>} <= 0 - C<time_enable_field> - 1

Flag: Set/Cleared as a result of the operation

Dec/Hex: 9’s complement in Decimal Mode, 15’s complement in Hexadecimal Mode
Turbo: Independent

	Assembly Syntax	Encoding	Machine Cycles
	C=-C-1 TEF	1010_1TEF_10	1

Note: This is the logical complement, which is the same as subtracting the value from negative one.

C=C+1

Increment C

C=C+1 operand: Time Enable Field

Operation: {CY, C<time_enable_field>} <= C<time_enable_field> + 1

Flag: Set/Cleared as a result of the operation

Dec/Hex: Decimal adjusted in Decimal Mode
Turbo: Independent

	Assembly Syntax	Encoding	Machine Cycles
	C=C+1 TEF	1000_1TEF_10	1

C=C+C

Load C With C + C

C=C+C operand: Time Enable Field

Operation: {CY, C<time_enable_field>} <= C<time_enable_field> + C<time_enable_field>

Flag: Set/Cleared as a result of the operation

Dec/Hex: Decimal adjusted in Decimal Mode
Turbo: Independent

	Assembly Syntax	Encoding	Machine Cycles
	C=C+C TEF	0111_1TEF_10	1

C=C-1

Decrement C

C=C-1 operand: Time Enable Field

Operation: {CY, C<time_enable_field>} <= C<time_enable_field> - 1

Flag: Set/Cleared as a result of the operation

Dec/Hex: Decimal adjusted in Decimal Mode
Turbo: Independent

	Assembly Syntax	Encoding	Machine Cycles
	C=C-1 TEF	1001_1TEF_10	1

C=C&A

Load C With C AND A

C=C&A operand: none

Operation: C <= C & A

Flag: Cleared

Dec/Hex: Independent
Turbo: Independent

	Assembly Syntax	Encoding	Machine Cycles
	C=C&A	1110_1100_00	1

Note: In the original NUT implementation this instruction cannot immediately follow an arithmetic (type 10) instruction. This restriction is not present in the NEWT implementation.

C=CORA

Load C With C OR A

C=CORA operand: none

Operation: C <= C | A

Flag: Cleared

Dec/Hex: Independent
Turbo: Independent

	Assembly Syntax	Encoding	Machine Cycles
	C=CORA	1101_1100_00	1

Note: In the original NUT implementation this instruction cannot immediately follow an arithmetic (type 10) instruction. This restriction is not present in the NEWT implementation.

C=DATA

Load C From Register (Indirect)

C=DATA	operand: none
Operation:	C <= REG[reg_addr]
Flag:	Cleared
Dec/Hex:	Independent
Turbo:	Independent for valid register address; fetched and executed at bus speed for an unimplemented register or a peripheral register.

	Assembly Syntax	Encoding	Machine Cycles
	C=DATA	0000_1110_00	1

Note: The register address must have been previously loaded into reg_addr by a DADD=C instruction. The data is actually loaded from the parallel memory bus using the RAM chip select.

As discussed in the Memory Organization chapter, there are unimplemented areas in the register map in 41C mode. Accessing an unimplemented register causes the **data_bus** to remain floating, allowing an external device to drive the **data_bus**.

C=DATAP

Load C From Peripheral

C=DATAP	operand: peripheral register number
Operation:	C <= peripheral register n
Flag:	Cleared
Dec/Hex:	Independent
Turbo:	Automatically executed at bus speed

	Assembly Syntax	Encoding	Machine Cycles
	C=DATAP n	nnnn_1110_1x	1

Note: This instruction is only available while a peripheral is in control of the bus. Peripherals may have up to sixteen registers to communicate information to the CPU via this instruction. The active peripheral drives the contents of the selected register on the **data_bus** during the following bus cycle.

The LSB of this instruction determines whether or not control is immediately returned to the CPU. If the LSB is zero the peripheral remains in control, while if the LSB is one control is returned to the CPU. In either case the bus cycle following the fetch of this instruction is always executed at normal bus speed to allow the peripheral time to transfer the data to the CPU.

C=G

Load C From G

C=G	operand: none
Operation:	C<ptr+:ptr> <= G
Flag:	Cleared
Dec/Hex:	Independent
Turbo:	Independent

	Assembly Syntax	Encoding	Machine Cycles
	C=G	0010_0110_00	1

Note: There are several boundary conditions that can occur when the pointer is pointing at the most significant nibble. These are detailed below:

1. If the active pointer was not changed to point at the most significant nibble immediately prior to this instruction, then:

```
C<13> <= G<1>
C<0>  <= G<0>
```

2. If the active pointer was changed to point at the most significant nibble (using PT=13, INC PT or DEC PT) immediately prior to this instruction, then:

```
fork
  C<13>  <= G<0>
  G      <= {G<0>, G<1>}
  join
```

3. If the active pointer was changed from pointing at the most significant nibble (using DEC PT only) immediately prior to this instruction, then:

```
fork
  C<13:12> <= {G<0>, G<1>}
  C<0>     <= G<0>
  G        <= {G<0>, G<1>}
  join
```

4. If the active pointer was changed from pointing at the most significant nibble (using PT=d only) immediately prior to this instruction, then:

```
fork
  C<ptr+:ptr> <= {G<0>, G<1>}
  C<0>        <= G<0>
  G           <= {G<0>, G<1>}
  join
```

C=KEYS

Load C From KEYS

C=KEYS operand: none

Operation: C<4:3> <= KEYS

Flag: Cleared

Dec/Hex: Independent
Turbo: Automatically executed at bus speed

	Assembly Syntax	Encoding	Machine Cycles
	C=KEYS	1000_1000_00	1

C=M

Load C From M

C=M operand: none

Operation: C <= M

Flag: Cleared

Dec/Hex: Independent
Turbo: Independent

	Assembly Syntax	Encoding	Machine Cycles
	C=M	0110_0110_00	1

C=N

Load C From N

C=N operand: none

Operation: C <= N

Flag: Cleared

Dec/Hex: Independent
Turbo: Independent

	Assembly Syntax	Encoding	Machine Cycles
	C=N	0010_1100_00	1

C=REGN

Load C From Register

C=REGN	operand: register number
Operation:	reg_addr <= {reg_addr[11:4], nnnn} C <= REG[reg_addr]
Flag:	Cleared
Dec/Hex:	Independent
Turbo:	Independent for valid register address; fetched and executed at bus speed for an unimplemented register or a peripheral register.

	Assembly Syntax	Encoding	Machine Cycles
	C=REGN n	nnnn_1110_00	1

Note: Bits 11-4 of the register address must have been previously loaded into reg_addr by a DADD=C instruction. The data is actually loaded from the parallel memory bus using the RAM chip select.

As discussed in the Memory Organization chapter, there are unimplemented areas in the register map in 41C mode. Accessing an unimplemented register causes the **data_bus** to remain floating, allowing an external device to drive the **data_bus**.

Only fifteen encodings are valid for nnnn. The all zeros case is the C=DATA instruction, with indirect register addressing.

C=ST

Load C From ST

C=ST operand: none

Operation: C<1:0> <= ST[7:0]

Flag: Cleared

Dec/Hex: Independent
Turbo: Independent

	Assembly Syntax	Encoding	Machine Cycles
	C=ST	1110_0110_00	1

C=STK

Load C From STK

C=STK operand: none

Operation:
C<6:3> <= STK0
STK0 <= STK1
STK1 <= STK2
STK2 <= STK3
STK3 <= 0000

Flag: Cleared

Dec/Hex: Independent
Turbo: Independent

	Assembly Syntax	Encoding	Machine Cycles
	C=STK	0110_1100_00	1

CGEX

Exchange C and G

CGEX operand: none

Operation:
```
fork
   C<ptr+:ptr> <= G
   G <= C<ptr+,ptr>
   join
```

Flag: Cleared

Dec/Hex: Independent
Turbo: Independent

	Assembly Syntax	Encoding	Machine Cycles
	CGEX	0011_0110_00	1

Note: There are several boundary conditions that can occur when the pointer is pointing at the most significant nibble. These are detailed below:

1. If the active pointer was not changed to point at the most significant nibble immediately prior to this instruction, then:

```
fork
   C<13> <= G<1>
   C<0>  <= G<0>
   G     <= {C<13>, C<0>}
   join
```

2. If the active pointer was changed to point at the most significant nibble (using PT=13, INC PT or DEC PT) immediately prior to this instruction, then:

```
fork
   C<13>  <= G<0>
   G      <= {C<13>, G<1>}
   join
```

3. If the active pointer was changed from pointing at the most significant nibble (using DEC PT only) immediately prior to this instruction, then:

```
fork
   C<13:12> <= {C<0>, G<1>}
   C<0>     <= G<0>
   G        <= C<13:12>
   join
```

4. If the active pointer was changed from pointing at the most significant nibble (using PT=d only) immediately prior to this instruction, then:

```
fork
   C<ptr+:ptr> <= {C<0>, G<1>}
   C<0>        <= G<0>
   G           <= C<ptr+:ptr>
   join
```

CHKKB

Check Keyboard

CHKKB operand: none

Operation: CY <= KYF

Flag: Set/Cleared according to the state of the keyboard flag

Dec/Hex: Independent
Turbo: Automatically executed at bus speed, along with the next two instructions.

Assembly Syntax	Encoding	Machine Cycles
CHKKB	1111_0011_00	1

Note: To guarantee proper operation with the keyboard scanner state machine, this instruction, and the next two instructions, are automatically executed at normal speed.

CLRST

Clear ST

CLRST operand: none

Operation: ST[7:0] <= 0

Flag: Cleared

Dec/Hex: Independent
Turbo: Independent

	Assembly Syntax	Encoding	Machine Cycles
	CLRST	1111_0001_00	1

CLRABC

Clear A, B and C

CLRABC operand: none

Operation: A <= 0
B <= 0
C <= 0

Flag: Cleared

Dec/Hex: Independent
Turbo: Independent

	Assembly Syntax	Encoding	Machine Cycles
	CLRABC	0110_1000_00	1

CLRDATA

Clear Registers

CLRDATA	operand: none
Operation:	No operation
Flag:	Cleared
Dec/Hex:	Independent
Turbo:	Automatically fetched and executed at bus speed

	Assembly Syntax	Encoding	Machine Cycles
	CLRDATA	1010_1100_00	1

Note: This instruction is a NOP for the processor. The original data storage chips used in the HP-41 series cleared all 16 registers on the selected data storage chip as a result of this instruction. Because the NEWT design uses standard parallel RAM devices this function is not supported by the NEWT microprocessor.

CMEX

Exchange C and M

CMEX operand: none

Operation:
```
fork
  C <= M
  M <= C
  join
```

Flag: Cleared

Dec/Hex: Independent
Turbo: Independent

	Assembly Syntax	Encoding	Machine Cycles
	CMEX	0111_0110_00	1

CNEX

Exchange C and N

CNEX operand: none

Operation:
```
fork
  C <= N
  N <= C
  join
```

Flag: Cleared

Dec/Hex: Independent
Turbo: Independent

	Assembly Syntax	Encoding	Machine Cycles
	CNEX	0011_1100_00	1

CSTEX

Exchange C and ST

CSTEX operand: none

Operation:
```
fork
  C<1:0> <= ST[7:0]
  ST[7:0] <= C<1:0>
  join
```

Flag: Cleared

Dec/Hex: Independent
Turbo: Independent

	Assembly Syntax	Encoding	Machine Cycles
	CSTEX	1111_0110_00	1

Note: In the original NUT implementation this instruction cannot immediately follow an arithmetic (type 10) instruction. This restriction is not present in the NEWT implementation.

CXISA

Exchange C and ISA

CXISA		operand: none
Operation:	mem_addr <= C<6:3> C<2:0> <= ISA[mem_addr]	
Flag:	Cleared	
Dec/Hex:	Independent	
Turbo:	Independent	

	Assembly Syntax	Encoding	Machine Cycles
	CXISA	1100_1100_00	2

Note: During the second machine cycle of this instruction the contents of C<6:3> are used as a program memory address on **isa_bus**. The contents of this program memory location are then loaded into C<2:0>, right-justified, with the two most significant bits set to 0.

DADD=C

Load Register address from C

DADD=C operand: none

Operation: reg_addr <= C<2:0>

Flag: Cleared

Dec/Hex: Independent
Turbo: Independent

	Assembly Syntax	Encoding	Machine Cycles
	DADD=C	1001_1100_00	1

Note: The register address is loaded into reg_addr (at memory location 0x804000) by the DADD=C instruction.

DATA=C

Load Register from C

DATA=C	operand: none
Operation:	REG[reg_addr] <= C
Flag:	Cleared
Dec/Hex:	Independent
Turbo:	Independent for valid register address; fetched and executed at bus speed for an unimplemented register or a peripheral register.

	Assembly Syntax	Encoding	Machine Cycles
	DATA=C	1011_1100_00	1

Note: The register address must have been previously loaded into reg_addr by a DADD=C instruction. The data is actually written to the parallel memory bus using the RAM chip select.

DECPT

Decrement Pointer

DECPT operand: none

Operation: ptr <= ptr-

Flag: Cleared

Dec/Hex: Independent
Turbo: Independent

	Assembly Syntax	Encoding	Machine Cycles
	DECPT	1111_0101_00	1

Note: This is not a binary or decimal decrement. The pointer decrements to the next lower digit position.

current ptr	next ptr	current digit -> next digit
0000	1000	3 (Mantissa digit 0) -> 2
0001	0000	4 (Mantissa digit 1) -> 3
0010	0001	5 (Mantissa digit 2) -> 4
0011	1001	10 (Mantissa digit 7) -> 9
0100	1010	8 (mantissa digit 5) -> 7
0101	0010	6 (Mantissa digit 3) -> 5
0110	0011	11 (Mantissa digit 8) -> 10
1000	1100	2 (Exponent Sign digit) -> 1
1001	0100	9 (Mantissa digit 6) -> 8
1010	0101	7 (Mantissa digit 4) -> 6
1011	1101	13 (Mantissa sign digit) -> 12
1100	1110	1 (Exponent digit 1) -> 0
1101	0110	12 (Mantissa digit 9) -> 11
1110	1011	0 (Exponent digit 0) -> 13

DISOFF

Display off

DISOFF operand: none

Operation: No operation

Flag: Cleared

Dec/Hex: Independent
Turbo: Automatically fetched and executed at bus speed

Assembly Syntax	Encoding	Machine Cycles
DISOFF	1011_1000_00	1

Note: This instruction is a NOP for the processor, but is interpreted by the (off-chip) LCD display controller, which turns off the display.

DISTOG

Display Toggle

DISTOG operand: none

Operation: No operation

Flag: Cleared

Dec/Hex: Independent
Turbo: Automatically fetched and executed at bus speed

	Assembly Syntax	Encoding	Machine Cycles
	DISTOG	1100_1000_00	1

Note: This instruction is a NOP for the processor, but is interpreted by the (off-chip) LCD display controller, which toggles the state of the display.

ENROMx

Enable ROM bank x

ENROMx	operand: none
Operation:	No operation
Flag:	Cleared
Dec/Hex:	Independent
Turbo:	Automatically fetched and executed at bus speed

	Assembly Syntax	Encoding	Machine Cycles
	ENROM1	0100_0000_00	1
	ENROM2	0110_0000_00	1
	ENROM3	0101_0000_00	1
	ENROM4	0111_0000_00	1

Note: This instruction is a NOP for the processor, but is interpreted by either the on-chip memory controller (if the current page is in system memory) or an external ROM module (if the current page is in an external ROM module).

For the on-chip memory controller, the actual bank select changes at the end of the current machine cycle. This means that the instruction following the ENROMx and all subsequent instructions are fetched from the new bank. This operation in the original NUT is not specified, but the usual code to execute a bank change is duplicated in all of the banks that are physically present. This makes the operation independent of the actual timing. The HP documentation for the 12K ROM chip specifies that the bank changes at the end of the current machine cycle.

F=SB

Load Flag Out from Status Byte

F=SB operand: none

Operation: FO<7:0> <= ST[7:0]

Flag: Cleared

Dec/Hex: Independent
Turbo: Automatically executed a bus speed

	Assembly Syntax	Encoding	Machine Cycles
	F=SB	1001_0110_00	1

Note: The **fo_bus** output timing is not independent of the Turbo mode, so the timing is identical to normal operation only during the execution of this instruction. Thus a timing loop that times the duration of the **fo_bus** output should be tagged to execute at normal bus speed.

FEXSB

Exchange Flag Out with Status Byte

FEXSB operand: none

Operation:
```
fork
  FO<7:0> <= ST[7:0]
  ST[7:0] <= FO<7:0>
  join
```

Flag: Cleared

Dec/Hex: Independent
Turbo: Automatically executed at bus speed

	Assembly Syntax	Encoding	Machine Cycles
	FEXSB	1011_0110_00	1

Note: The **fo_bus** output timing is not independent of the Turbo mode, so the timing is identical to normal operation only during the execution of this instruction. Thus a timing loop that times the duration of the **fo_bus** output should be tagged to execute at normal bus speed.

G=C

Load G From C

G=C operand: none

Operation: G <= C<ptr+:ptr>

Flag: Cleared

Dec/Hex: Independent
Turbo: Independent

	Assembly Syntax	Encoding	Machine Cycles
	G=C	0001_0110_00	1

Note: There are several boundary conditions that can occur when the pointer is pointing at the most significant nibble. These are detailed below:

1. If the active pointer was not changed to point at the most significant nibble immediately prior to this instruction, then:

    G <= {C<13>, C<0>}

2. If the active pointer was changed to point at the most significant nibble (using PT=13, INC PT or DEC PT) immediately prior to this instruction, then:

    G <= {C<13>, G<1>}

3. If the active pointer was changed from pointing at the most significant nibble (using DEC PT only) immediately prior to this instruction, then:

    G <= C<13:12> which is normal operation

GOC

Branch (Relative) on Carry

GOC	operand: relative offset
Operation:	if CY (PC <= PC + rel_offset)
Flag:	Cleared
Dec/Hex:	Independent
Turbo:	Independent

	Assembly Syntax	Encoding	Machine Cycles
	GOC rel_offset	aaaa_aaa1_11	1

Note: The relative offset is a 7-bit 2's complement number that is sign-extended to 16 bits and added to the PC of this instruction if the CY flag is true. This allows a jump in the range of -64 to +63 from the address of this instruction.

GONC

Branch (Relative) on No Carry

GONC	operand: relative offset
Operation:	if !CY (PC <= PC + rel_offset)
Flag:	Cleared
Dec/Hex:	Independent
Turbo:	Independent

	Assembly Syntax	Encoding	Machine Cycles
	GONC rel_offset	aaaa_aaa0_11	1

Note: The relative offset is a 7-bit 2's complement number that is sign-extended to 16 bits and added to the PC of this instruction if the CY flag is false. This allows a jump in the range of -64 to +63 from the address of this instruction.

GOKEYS

Branch to Keys

GOKEYS operand: none

Operation: PC <= {PC[15:8], KEYS}

Flag: Cleared

Dec/Hex: Independent
Turbo: Automatically executed at bus speed

	Assembly Syntax	Encoding	Machine Cycles
	GOKEYS	1000_1100_00	1

GOTOC

Branch using C register

GOTOC operand: none

Operation: PC <= C<6:3>

Flag: Cleared

Dec/Hex: Independent
Turbo: Independent

	Assembly Syntax	Encoding	Machine Cycles
	GOTOC	0111_1000_00	1

GOLC

Branch (Long) on Carry

GOLC	operand: jump address
Operation:	if CY (PC <= jump_addr)
Flag:	Cleared
Dec/Hex:	Independent
Turbo:	Independent

	Assembly Syntax	Encoding	Machine Cycles
	GOLC jump_addr	aaaa_aaaa_01 aaaa_aaaa_11	2

Note: The first word of the instruction contains bits 7-0 of the jump address, and the second word of the instruction contains bits 15-8 of the jump address.

The **sync** signal is suppressed during the fetch of the second word of the instruction to prevent external devices from incorrectly interpreting the contents of the **isa_bus** during the second machine cycle as an instruction.

GOLNC

Branch (Long) on No Carry

GOLNC	operand: jump address
Operation:	if !CY (PC <= jump_addr)
Flag:	Cleared
Dec/Hex:	Independent
Turbo:	Independent

	Assembly Syntax	Encoding	Machine Cycles
	GOLCNC jump_addr	aaaa_aaaa_01 aaaa_aaaa_10	2

Note: The first word of the instruction contains bits 7-0 of the jump address, and the second word of the instruction contains bits 15-8 of the jump address.

The **sync** signal is suppressed during the fetch of the second word of the instruction to prevent external devices from incorrectly interpreting the contents of the **isa_bus** during the second machine cycle as an instruction.

GSUBC

Branch (to Subroutine) on Carry

GSUBC operand: jump address

Operation:
```
if CY begin
   STK3 <= STK2
   STK2 <= STK1
   STK1 <= STK0
   STK0 <= PC
   PC <= jump_addr
   end
```

Flag: Cleared

Dec/Hex: Independent
Turbo: Independent

Assembly Syntax	Encoding	Machine Cycles
GSUBC jump_addr	aaaa_aaaa_01 aaaa_aaaa_01	2

Note: The first word of the instruction contains bits 7-0 of the jump address, and the second word of the instruction contains bits 15-8 of the jump address.

The **sync** signal is suppressed during the fetch of the second word of the instruction to prevent external devices from incorrectly interpreting the contents of the **isa_bus** during the second machine cycle as an instruction.

The PC pushed onto the return stack is the PC of the instruction following the second word of this instruction.

If the instruction at jump_addr is NOP, it is automatically executed as RET to protect against executing from a non-existent ROM.

GSUBNC

Branch (to Subroutine) on No Carry

GSUBNC operand: jump address

Operation:
```
if !CY begin
   STK3 <= STK2
   STK2 <= STK1
   STK1 <= STK0
   STK0 <= PC
   PC <= jump_addr
   end
```

Flag: Cleared

Dec/Hex: Independent
Turbo: Independent

	Assembly Syntax	Encoding	Machine Cycles
	GSUB jump_addr	aaaa_aaaa_01 aaaa_aaaa_00	2

Note: The first word of the instruction contains bits 7-0 of the jump address, and the second word of the instruction contains bits 15-8 of the jump address.

The **sync** signal is suppressed during the fetch of the second word of the instruction to prevent external devices from incorrectly interpreting the contents of the **isa_bus** during the second machine cycle as an instruction.

The PC pushed onto the return stack is the PC of the instruction following the second word of this instruction.

If the instruction at jump_addr is NOP, it is automatically executed as RET to protect against executing from a non-existent ROM.

INCPT

Increment Pointer

INCPT operand: none

Operation: ptr <= ptr+

Flag: Cleared

Dec/Hex: Independent
Turbo: Independent

	Assembly Syntax	Encoding	Machine Cycles
	INCPT	1111_0111_00	1

Note: This is not a binary or decimal increment. The pointer increments to the next higher digit position.

current ptr	next ptr	current digit -> next digit
0000	0001	3 (Mantissa digit 0) -> 4
0001	0010	4 (Mantissa digit 1) -> 5
0010	0101	5 (Mantissa digit 2) -> 6
0011	0110	10 (Mantissa digit 7) -> 11
0100	1001	8 (mantissa digit 5) -> 9
0101	1010	6 (Mantissa digit 3) -> 7
0110	1101	11 (Mantissa digit 8) -> 12
1000	0000	2 (Exponent Sign digit) -> 3
1001	0011	9 (Mantissa digit 6) -> 10
1010	0100	7 (Mantissa digit 4) -> 8
1011	1110	13 (Mantissa sign digit) -> 0
1100	1000	1 (Exponent digit 1) -> 2
1101	1011	12 (Mantissa digit 9) -> 13
1110	1100	0 (Exponent digit 0) -> 1

LC

Load Constant

LC operand: immediate nibble

Operation: C<ptr> <= nnnn
ptr <= ptr-

Flag: Cleared

Dec/Hex: Independent
Turbo: Independent

	Assembly Syntax	Encoding	Machine Cycles
	LC n	nnnn_0100_00	1

Note: The pointer decrements to the next lower digit position.

current ptr	next ptr	current digit -> next digit
0000	1000	3 (Mantissa digit 0) -> 2
0001	0000	4 (Mantissa digit 1) -> 3
0010	0001	5 (Mantissa digit 2) -> 4
0011	1001	10 (Mantissa digit 7) -> 9
0100	1010	8 (mantissa digit 5) -> 7
0101	0010	6 (Mantissa digit 3) -> 5
0110	0011	11 (Mantissa digit 8) -> 10
1000	1100	2 (Exponent Sign digit) -> 1
1001	0100	9 (Mantissa digit 6) -> 8
1010	0101	7 (Mantissa digit 4) -> 6
1011	1101	13 (Mantissa sign digit) -> 12
1100	1110	1 (Exponent digit 1) -> 0
1101	0110	12 (Mantissa digit 9) -> 11
1110	1011	0 (Exponent digit 0) -> 13

LDI

Load Immediate

LDI	operand: immediate 10-bit value
Operation:	C<2:0> <= {2'b00, const}
Flag:	Cleared
Dec/Hex:	Independent
Turbo:	Independent

	Assembly Syntax	Encoding	Machine Cycles
	LDI const	0100_1100_00 const	2

Note: When executed at bus speed the **sync** signal is suppressed during the fetch of the second word of the instruction to prevent external devices from incorrectly interpreting the contents of the **isa_bus** during the second machine cycle as an instruction.

M=C

Load M from C

M=C operand: none

Operation: M <= C

Flag: Cleared

Dec/Hex: Independent
Turbo: Independent

	Assembly Syntax	Encoding	Machine Cycles
	M=C	0101_0110_00	1

N=C

Load N from C

N=C operand: none

Operation: N <= C

Flag: Cleared

Dec/Hex: Independent
Turbo: Independent

	Assembly Syntax	Encoding	Machine Cycles
	N=C	0001_1100_00	1

NOP

No Operation

NOP operand: none

Operation: None

Flag: Cleared

Dec/Hex: Independent
Turbo: Independent

	Assembly Syntax	Encoding	Machine Cycles
	NOP	0000_0000_00	1

PFAD=C

Load Peripheral Address from C

PFAD=C operand: none

Operation: No operation

Flag: Cleared

Dec/Hex: Independent
Turbo: Automatically fetched and executed at bus speed

Assembly Syntax	Encoding	Machine Cycles
PFAD=C	1111_1100_00	1

Note: This instruction is a NOP for the processor, but is interpreted by peripheral devices to perform the chip select function. The peripheral chip select remains active until another PFAD=C instruction selects a different peripheral. Peripheral devices decode the least-significant byte on the **data_bus** according to the following table:

c<1:0>	Peripheral Device
0xF0	NEWT On-chip Port
0xFB	Timer
0xFC	Card Reader
0xFD	LCD Display Driver
0xFE	Wand

POWOFF

Power Down

POWOFF operand: none

Operation: Power Down

Flag: Cleared

Dec/Hex: Independent
Turbo: Automatically fetched and executed at bus speed

	Assembly Syntax	Encoding	Machine Cycles
	POWOFF	0001_1000_00 0000_0000_00	2

Note: This instruction is unique in that the **sync** signal is active during the fetch of the second word of the instruction. This does not cause problem with other devices on the bus because the system is now in the process of powering down.

PT=

Load Pointer immediate

PT= operand: digit

Operation: ptr <= digit

Flag: Cleared

Dec/Hex: Independent
Turbo: Independent

Assembly Syntax	Encoding	Machine Cycles
PT= d	dddd_0111_00	1

RCR

Rotate C right by digits

RCR operand: digit

Operation:
```
for (i=0; i<d; i++) begin
   C <= {C<0>, C<13:1>}
   end
```

Flag: Cleared

Dec/Hex: Independent
Turbo: Independent

	Assembly Syntax	Encoding	Machine Cycles
	RCR d	dddd_1111_00	1

REGN=C

Load Register from C

REGN=C	operand: register number
Operation:	reg_addr <= {reg_addr[11:4], nnnn} REG[reg_addr] <= C
Flag:	Cleared
Dec/Hex:	Independent
Turbo:	Independent for valid register address; fetched and executed at bus speed for an unimplemented register or a peripheral register.

	Assembly Syntax	Encoding	Machine Cycles
	REGN=C n	nnnn_1010_00	1

Note: Bits 11-4 of the register address must have been previously loaded into reg_addr by a DADD=C instruction. The data is actually written to the parallel memory bus using the RAM chip select.

The value nnnn replaces the least significant four bits of the current value in the latched register address reg_addr.

RST KB

Reset Keyboard

RST KB operand: none

Operation: KYF <= 0

Flag: Cleared

Dec/Hex: Independent
Turbo: Automatically executed at bus speed, along with the next two instructions.

	Assembly Syntax	Encoding	Machine Cycles
	RSTKB	1111_0010_00	1

Note: The keyboard flag is only cleared if the key has been released before this instruction is executed. If the key is still down while this instruction is executed the flag will remain set.

To guarantee proper operation with the keyboard scanner state machine, this instruction, and next the two instructions, are automatically executed at normal speed.

RTN

Return from subroutine

RTN operand: none

Operation:
PC <= STK0
STK0 <= STK1
STK1 <= STK2
STK2 <= STK3
STK3 <= 0000

Flag: Cleared

Dec/Hex: Independent
Turbo: Independent

	Assembly Syntax	Encoding	Machine Cycles
	RTN	1111_1000_00	1

RTNC

Return from subroutine on Carry

RTNC operand: none

Operation:
```
if (CY) begin
   PC <= STK0
   STK0 <= STK1
   STK1 <= STK2
   STK2 <= STK3
   STK3 <= 0000
   end
```

Flag: Cleared

Dec/Hex: Independent
Turbo: Independent

	Assembly Syntax	Encoding	Machine Cycles
	RTNC	1101_1000_00	1

RTNNC

Return from subroutine on No Carry

RTNNC operand: none

Operation:
```
if (!CY) begin
   PC <= STK0
   STK0 <= STK1
   STK1 <= STK2
   STK2 <= STK3
   STK3 <= 0000
   end
```

Flag: Cleared

Dec/Hex: Independent
Turbo: Independent

	Assembly Syntax	Encoding	Machine Cycles
	RTNNC	1110_1000_00	1

SB=F

Load Status Byte from Flag Out

SB=F operand: none

Operation: ST[7:0] <= FO[7:0]

Flag: Cleared

Dec/Hex: Independent
Turbo: Automatically executed a bus speed

	Assembly Syntax	Encoding	Machine Cycles
	SB=F	1010_0110_00	1

SELP

Select Pointer P

SELP operand: none

Operation: ptr = P

Flag: Cleared

Dec/Hex: Independent
Turbo: Independent

	Assembly Syntax	Encoding	Machine Cycles
	SELP	0010_1000_00	1

SELQ

Select Pointer Q

SELQ operand: none

Operation: ptr = Q

Flag: Cleared

Dec/Hex: Independent
Turbo: Independent

	Assembly Syntax	Encoding	Machine Cycles
	SELQ	0011_1000_00	1

SELPF

Select Peripheral

SELPF	operand: peripheral number
Operation:	transfer control to peripheral n
Flag:	Cleared
Dec/Hex:	Independent
Turbo:	Automatically fetched and executed at bus speed

	Assembly Syntax	Encoding	Machine Cycles
	SELPF n	nnnn_1001_00	2 or more

Note: This instruction transfers control from the CPU to an intelligent peripheral. The CPU continues to fetch instructions, incrementing the PC with each fetch, but in general ignores the instructions fetched from the **isa_bus**. Control is returned to the CPU when the instruction fetched has the LSB set to one. All of these fetches are executed at normal bus speed, including the first fetch after control is returned to the CPU.

Two instructions are available to transfer information from the peripheral back to the CPU: First, the ?PFLGn=1 instruction transfers the contents of one of sixteen flags internal to the peripheral back to the CY flag during the first clock cycle of the following instruction (which is executed by the CPU). Second, the C=DATAPn instruction sets the **data_bus** as an input and the contents of the **data_bus** during the execution of this instruction is loaded into the C register. With these two instructions, either status information or data may be communicated from the peripheral to the CPU.

SETDEC

Set Decimal Mode

SETDEC operand: none

Operation: hex_mode = 0

Flag: Cleared

Dec/Hex: Independent
Turbo: Independent

	Assembly Syntax	Encoding	Machine Cycles
	SETDEC	1010_1000_00	1

SETHEX

Set Hexadecimal Mode

SETHEX operand: none

Operation: hex_mode = 1

Flag: Cleared

Dec/Hex: Independent
Turbo: Independent

	Assembly Syntax	Encoding	Machine Cycles
	SETHEX	1001_1000_00	1

SPOPND

Pop Stack

SPOPND operand: none

Operation:
STK0 <= STK1
STK1 <= STK2
STK2 <= STK3
STK3 <= 0000

Flag: Cleared

Dec/Hex: Independent
Turbo: Independent

	Assembly Syntax	Encoding	Machine Cycles
	SPOPND	0000_1000_00	1

ST=0

Clear Status bit

ST=0 operand: Digit Number

Operation: ST[digit] <= 0

Flag: Cleared

Dec/Hex: Independent
Turbo: Independent

	Assembly Syntax	Encoding	Machine Cycles
	ST=0 d	dddd_0001_00	1

Note: In the original NUT implementation this instruction cannot immediately follow an arithmetic (type 10) instruction. This restriction is not present in the NEWT implementation.

ST=1

Set Status bit

ST=1 operand: Digit Number

Operation: ST[digit] <= 1

Flag: Cleared

Dec/Hex: Independent
Turbo: Independent

	Assembly Syntax	Encoding	Machine Cycles
	ST=1 d	dddd_0010_00	1

Note: In the original NUT implementation this instruction cannot immediately follow an arithmetic (type 10) instruction. This restriction is not present in the NEWT implementation.

ST=1?

Test Status Equal To One

ST=1? operand: Digit Number

Operation: CY <= Status<digit>

Flag: Set/Cleared as a the result of the test

Dec/Hex: Independent
Turbo: Independent

	Assembly Syntax	Encoding	Machine Cycles
	ST=1? d	dddd_0011_00	1

Note: In the original NUT implementation this instruction cannot immediately follow an arithmetic (type 10) instruction. This restriction is not present in the NEWT implementation.

ST=C

Load Status from C

ST=C operand: none

Operation: ST[7:0] <= C<1:0>

Flag: Cleared

Dec/Hex: Independent
Turbo: Independent

	Assembly Syntax	Encoding	Machine Cycles
	ST=C	1101_0110_00	1

STK=C

Push C

STK=C operand: none

Operation: STK3 <= STK2
STK2 <= STK1
STK1 <= STK0
STK0 <= C<6:3>

Flag: Cleared

Dec/Hex: Independent
Turbo: Independent

	Assembly Syntax	Encoding	Machine Cycles
	STK=C	0101_1100_00	1

WCMD

Write Command

WCMD operand: none

Operation: no operation

Flag: Cleared

Dec/Hex: Independent
Turbo: Automatically fetched and executed at bus speed

	Assembly Syntax	Encoding	Machine Cycles
	WCMD	0111_1111_00	1

Note: This instruction is a NOP as far as the processor is concerned, but is interpreted by the Memory Management Unit or Turbo Control Unit. Writes automatically transfer the contents of the C register to the destination. Reads latch the relevant data, which can then be accessed via the on-chip I/O Port. The contents of digit 4 are interpreted as a command, and some of the remaining digit contents are used for data. The write commands are:

Command	Meaning
0	Write MMU (per bank)
2	Write Logical Address
4	Write Physical Address
6	Global MMU Disable
7	Global MMU Enable
8	Disable Turbo Mode
9	Enable 2X Turbo Mode
A	Enable 5X Turbo Mode
B	Enable 10X Turbo Mode
C	Enable 20X Turbo Mode
D	Enable 50X Turbo Mode

E	Special MMU Disable
F	Special MMU Enable

The read commands are:

Command	Meaning
1	Read MMU (per bank)
3	Read from Logical Address
5	Read from Physical Address

The table below shows the format of the data used by these commands. Refer to the Memory Organization, Turbo Mode or I/O Port chapter for more details.

nibble	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Write MMU					page				bank	**0**	en	ph addr		
Read MMU					page				bank	**1**				
Write Logical Address				logical address					bank	**2**	write data			
Read Logical Address				logical address					bank	**3**				
Write Physical Address		physical address								**4**	write data			
Read Physical Address		physical address								**5**				
Global MMU Disable										**6**				
Global MMU Enable										**7**				
Disable Turbo Mode										**8**				
Enable 2x Turbo Mode										**9**				
Enable 5x Turbo Mode										**A**				
Enable 10x Turbo Mode										**B**				
Enable 20x Turbo Mode										**C**				
Enable 50x Turbo Mode										**D**				
Special MMU Disable										**E**				
Special MMU Enable										**F**				

WROM

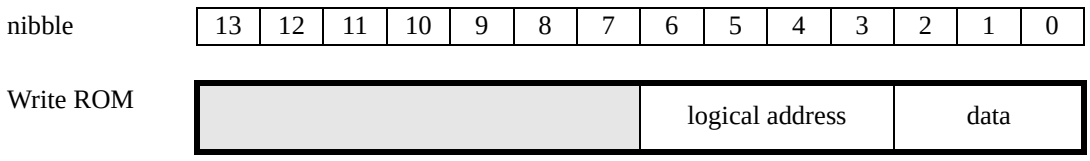
Write ROM

WROM	operand: none
Operation:	no operation
Flag:	Cleared
Dec/Hex:	Independent
Turbo:	Automatically fetched and executed at bus speed

	Assembly Syntax	Encoding	Machine Cycles
	WROM	0001_0000_00	1

Note: This instruction is a NOP as far as the processor is concerned, but is used in legacy software to write to the program address space, similar to the WCMD Write to Logical Address. The contents of the C register are used as follows: digits 6-3 are the logical address and digits 2-0 are the data. The write is performed to the currently active bank in the logical address page. This is different from the WCMD case, where the bank must be explicitly specified.

Only twelve bits of data are available to be written to the memory. The other four bits are always zero. This means that there is no way to control the Turbo mode tag bits in memory when using this write.



The table below shows all possible opcodes, organized sequentially, with the instruction mnemonic and the effect of Turbo mode.

Instruction	Opcode 1	Opcode 2	1x	always seen on bus	Notes
NOP	0000_0000_00				
WROM	0001_0000_00		yes	yes	
	0010_0000_00		yes	yes	reserved
	0011_0000_00		yes	yes	reserved
ENROM1	0100_0000_00		yes	yes	
ENROM3	0101_0000_00		yes	yes	
ENROM2	0110_0000_00		yes	yes	
ENROM4	0111_0000_00		yes	yes	
	1000_0000_00		yes	yes	reserved
	1001_0000_00		yes	yes	reserved
	1010_0000_00		yes	yes	reserved
	1011_0000_00		yes	yes	reserved
	1100_0000_00		yes	yes	reserved
	1101_0000_00		yes	yes	reserved
	1110_0000_00		yes	yes	reserved
	1111_0000_00		yes	yes	reserved
SD=0	dddd_0001_00				
	0111_0001_00		yes	yes	reserved
CLRST	1111_0001_00				
SD=1	dddd_0010_00				
	0111_0010_00		yes	yes	reserved
RSTKB	1111_0010_00		yes		
?SD=1	dddd_0011_00				
	0111_0011_00		yes	yes	reserved
CHKKB	1111_0011_00		yes		
LC	nnnn_0100_00				
?PT=D	dddd_0101_00				
	0111_0101_00		yes	yes	reserved
DECPT	1111_0101_00				
	0000_0110_00		yes	yes	reserved
G=C	0001_0110_00				
C=G	0010_0110_00				
CGEX	0011_0110_00				
	0100_0110_00		yes	yes	reserved

M=C	0101_0110_00				
C=M	0110_0110_00				
CMEX	0111_0110_00				
	1000_0110_00		yes	yes	reserved
F=SB	1001_0110_00		yes		
SB=F	1010_0110_00		yes		
FEXSB	1011_0110_00		yes		
	1100_0110_00		yes	yes	reserved
ST=C	1101_0110_00				
C=ST	1110_0110_00				
CSTEX	1111_0110_00				
PT=D	dddd_0111_00				
	0111_0111_00		yes	yes	reserved
INCPT	1111_0111_00				
SPOPND	0000_1000_00				
POWOFF	0001_1000_00	0000_0000_00	yes	yes	
SELP	0010_1000_00				
SELQ	0011_1000_00				
?P=Q	0100_1000_00				
?LLD	0101_1000_00				
CLRABC	0110_1000_00				
GOTOC	0111_1000_00				
C=KEYS	1000_1000_00		yes		
SETHEX	1001_1000_00				
SETDEC	1010_1000_00				
DISOFF	1011_1000_00		yes	yes	
DISTOG	1100_1000_00		yes	yes	
RTNC	1101_1000_00				
RTNNC	1110_1000_00				
RTN	1111_1000_00				
SELPF	nnnn_1001_00		yes	yes	
REGN=C	nnnn_1010_00				
?Fd=1	dddd_1011_00		yes		
	0111_1011_00		yes	yes	reserved
	1111_1011_00		yes	yes	reserved
	0000_1100_00		yes	yes	reserved
N=C	0001_1100_00				
C=N	0010_1100_00				
CNEX	0011_1100_00				

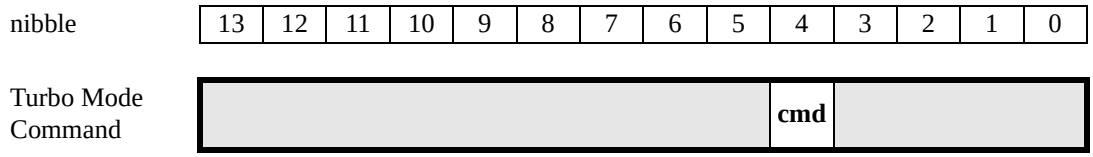
LDI	0100_1100_00	constant			
STK=C	0101_1100_00				
C=STK	0110_1100_00				
	0111_1100_00		yes	yes	reserved
GOKEYS	1000_1100_00		yes		
DADD=C	1001_1100_00				
CLRDATA	1010_1100_00		yes	yes	
DATA=C	1011_1100_00				
CXISA	1100_1100_00				
C=C|A	1101_1100_00				
C=C&A	1110_1100_00				
	1111_1100_00		yes	yes	reserved
	nnnn_1101_00		yes	yes	reserved
C=DATA	0000_1110_00		if peripheral	if peripheral	
C=REGN	nnnn_1110_00		if peripheral	if peripheral	
RCRD	dddd_1111_00				
WCMD	0111_1111_00		yes	yes	
	1111_1111_00		yes	yes	reserved
A=0	0000_0ttt_10				
B=0	0000_1ttt_10				
C=0	0001_0ttt_10				
AB EX	0001_1ttt_10				
B=A	0010_0ttt_10				
AC EX	0010_1ttt_10				
C=B	0011_0ttt_10				
BC EX	0011_1ttt_10				
A=C	0100_0ttt_10				
A=A+B	0100_1ttt_10				
A=A+C	0101_0ttt_10				
A=A+1	0101_1ttt_10				
A=A-B	0110_0ttt_10				
A=A-1	0110_1ttt_10				
A=A-C	0111_0ttt_10				
C=C+C	0111_1ttt_10				
C=A+C	1000_0ttt_10				
C=C+1	1000_1ttt_10				
C=A-C	1001_0ttt_10				
C=C-1	1001_1ttt_10				
C=-C	1010_0ttt_10				

C=-C-1	1010_1ttt_10				
?B#0	1011_0ttt_10				
?C#0	1011_1ttt_10				
?A<C	1100_0ttt_10				
?A<B	1100_1ttt_10				
?A#0	1101_0ttt_10				
?A#C	1101_1ttt_10				
ASR	1110_0ttt_10				
BSR	1110_1ttt_10				
CSR	1111_0ttt_10				
ASL	1111_1ttt_10				
GONC	aaaa_aaa0_11				
GOC	aaaa_aaa1_11				
GOLNC	aaaa_aaaa_01	aaaa_aaaa_10			
GOLC	aaaa_aaaa_01	aaaa_aaaa_11			
GSUBNC	aaaa_aaaa_01	aaaa_aaaa_00			
GSUBC	aaaa_aaaa_01	aaaa_aaaa_01			

Turbo Mode

The input clock frequency for the NEWT microprocessor is 18MHz. When operating at normal speed, this clock is divided by fifty, resulting in a “normal” operating frequency of 360KHz. The Turbo mode provides the option of running the processor at a frequency as high as the input clock frequency.

The WCMD is used to control the Turbo mode. The actual command is contained in nibble 4 and the remaining nibbles are all ignored. The state of the Turbo mode is not preserved during power-down, so the NEWT microprocessor always starts up with the clock divided by fifty, for backwards compatibility with the original Nut design.



The options available for the Turbo mode are shown below. Turbo mode operates by “swallowing” clock pulses in the interface between the 18MHz clock input and the processor core. Even though clock pulses are swallowed, the external bus of the NEWT design always operates at the normal 360KHz speed.

Command	Turbo Mode	Clock divider
8	Disable Turbo Mode	50
9	2X Turbo Mode	25
A	5X Turbo Mode	10
B	10X Turbo Mode	5
C	20X Turbo Mode	2.5
D	50X Turbo Mode	1

The NEWT microprocessor can only operate in Turbo mode while instructions are being fetched from the physical memory. Independent of this fetch and execution speed, the external serial signals are driven to simulate a Nut processor fetching continuous NOP instructions from memory. This guarantees compatibility with other devices connected to the serial bus. However, certain instructions must be “seen” by other devices on this serial bus. The NEWT microprocessor automatically recognizes these instructions and halts the CPU while the external serial bus is driven with the relevant instruction and the following instruction. This allows external devices to “see” these instructions and act accordingly. The relevant instructions are listed below. Note that some instructions are not refetched on the serial bus. These instructions do not need to be seen by external devices but must run at normal speed for some other reason, like proper timing for the **fo_bus** signal.

Opcode	Instruction	Re-fetched?
0001_0000_00	WROM	yes
001x_0000_00	undefined	yes
01xx_0000_00	ENROMx	yes
1xxx_0000_00	undefined	yes
0111_0001_00	undefined	yes
0111_0010_00	undefined	yes
1111_0010_00	RSTKB	no
0111_0011_00	undefined	yes
1111_0011_00	CHKKB	no
0111_0101_00	undefined	yes
xx00_0110_00	undefined	yes
1001_0110_00	F=SB	no
1010_0110_00	SB=F	no
1011_0110_00	FEXSB	no
0111_0111_00	undefined	yes
0001_1000_00	POWOFF	yes
1000_1000_00	C=KEYS	no
1011_1000_00	DISOFF	yes
1100_1000_00	DISTOG	yes
xxxx_1001_00	SELPFn	yes
xxxx_1011_00	?Fn=1	yes
x111_1011_00	undefined	yes
0000_1100_00	undefined	yes
1000_1100_00	GOKEYS	no
1010_1100_00	CLRDATA	yes
1111_1100_00	PFAD=C	yes
xxxx_1101_00	undefined	yes
0000_1110_00	C=DATA	yes, if peripheral register
xxxx_1110_00	C=REGN	yes, if peripheral register
0111_1111_00	WCMD	yes
1111_1111_00	undefined	yes

The register write instructions listed below will be re-fetched and executed at bus speed when writing to peripheral registers. If the register being written is a valid HP41 memory register these instructions execute at full speed.

Opcode	Instruction	Re-fetched?
xxxx_1010_00	REGN=C	yes
1011_1100_00	DATA=C	yes

There are other circumstances where it might not be convenient to change the Turbo mode but it might still be necessary to briefly operate at regular speed. One example of this would be timing loops. The NEWT design provides for this type of operation by using two of the unused bits in the 16-bit physical memory. If either of these two bits are set the speed is automatically throttled back to normal speed until an instruction is fetched with both of the bits cleared. One bit forces the logic to fetch the marked instruction at normal speed. This also means that the instruction will be refetched on the serial bus (and the previous instruction to execute at normal speed). The other bit forces the next instruction to be fetched at normal speed (and the current instruction to execute at normal speed).

Physical Data 13	Physical Data 12	This instruction	Next instruction
0	0	Turbo if enabled	Turbo if enabled
0	1	execute in normal	fetch in normal
1	x	refetch, then execute in normal	fetch in normal

Bit 13 will normally only be set for the first instruction in a sequence of instructions that need to execute at normal speed, and all other instructions in the sequence will have bit 12 set. Another alternative would be to set bit 12 in the instruction immediately prior to the first instruction that needs to execute at normal speed. The settings of bits 13 and 12 for those instructions that are automatically executed at normal speed are ignored.

Keyboard Scanner

The NEWT microprocessor contains a hardware keyboard scanner that accommodates seven column lines and nine row lines. This keyboard scanner operates continuously while the processor is running and always operates at the normal 1x frequency. The column lines, called **kc0** through **kc6**, are driven Low one after the other, starting with **kc0**, starting with the first seven nibble times. Refer to the Timing chapter for detailed timing diagrams. During the column scanning time the inactive column lines are undriven to limit current drain in case more than one key on the same row are pressed simultaneously.

Whenever a keypress is detected, by one of the row inputs being sampled Low during a column time, the corresponding keycode is latched into the keyboard buffer and the keyboard flag is set. The keyboard flag is tested by a dedicated instruction and the keyboard buffer can either be loaded into the C register or loaded into the least significant bits of the Program Counter, creating a virtual jump table. Once the keycode has been accessed the keyboard flag can be reset, again with a dedicated instruction.

To assist in providing a 2-key rollover, the keyboard scanner has three distinct states. In State 1 the keyboard flag is reset and the keyboard is enabled and waiting to detect a keypress. This is the reset and idle state of the keyboard scanner. State 2 is entered when a keypress is detected, and in this state the keyboard flag is set and the keyboard register holds the keycode. The keyboard scanner will remain in this state until a RST KB instruction is executed, the key that first caused the transition to State 2 is no longer down, and there are not two or more keys down on the same column. All three of these conditions must be met before the keyboard scanner transitions to State 3. In State 3 the keyboard flag is reset, but the keyboard register retains the value latched in State 2 and the keyboard scanner continues to scan the keyboard. However, scanning is effectively disabled because no key presses can be registered in this state. Instead, a CHK KB instruction is required to cause the scanner to transition back to State 1, where the scanning again is enabled.

The column table below shows the keyboard buffer encoding for the column lines. In the case of simultaneous keypress on different columns the key that is first sampled active by a column line is the one that will be latched into the keyboard buffer.

Column line	KEYS[7:4] or C<4>
kc0	0001
kc1	0011
kc2	0111
kc3	1000
kc4	1100
kc5	1110
kc6	1111

The row table below shows the encoding for the row lines. In the case of simultaneous keypress on the same column the keyboard scanner logic automatically prioritizes them highest-to-lowest in the row table.

Row line	KEYS[3:0] or C<3>
por	1000
kr7	0111
kr6	0110
kr5	0101
kr4	0100
kr3	0011
kr2	0010
kr1	0001
kr0	0000

Circuitry in the always-powered section of the NEWT microprocessor monitors the keyboard while the CPU itself is powered down. While the NEWT CPU is powered down (deep sleep mode) the column line **kc0** is driven Low, while the remaining column selects are floating. The row inputs are monitored to potentially wake up the CPU. This operation is covered in detail in the Power Control chapter.

While the NEWT CPU is powered but not running (light sleep mode) all of the column lines, **kc0-kc6**, are driven Low, which allows any keypress to initiate operation.

External Interface

The NEWT design must be surrounded by voltage translation circuitry to translate between the NEWT operating voltage and the original Nut operating voltage of 6V. There can be either one, two, or three NEWT signals corresponding to one Nut-compatible signal, depending on the required functionality for the Nut-compatible signal.

Not all of the Nut-compatible signals have to operate at 6V. Only those signals present on the HP-41 I/O ports or connected to the display subsystem need to maintain voltage compatibility. The lower operating voltage on the other signals reduces system power consumption and simplifies the interface circuitry.

The remainder of this chapter will describe both the Nut-compatible signals and the new NEWT signals.

clk Main Processor Clock

Input, 3.3V. The main processor clock is the output of an external 18MHz oscillator. This signal should be held Low (and the oscillator disabled) during power-down.

data_bus Register Read/Write Bus

Bi-directional, 6V. The register read/write bus normally outputs the contents of the C register during each machine cycle and would input register data to the C register during a register read. Since in the NEWT design register data is stored in physical memory the register data is output on this bus during a register read. While operating in Turbo mode this signal is always Low. This signal will be high-impedance during power-down, which means that an external resistive pull-down is required.

dpwo Display Power Off

Input, 6V. The display power-off signal is generated by the display driver to perform the auto-shutoff function. Normally High, this signal will go Low after several minutes of inactivity to power-down the NEWT processor.

fi_bus Flag Input Bus

Bidirectional, 6V, active-Low. The flag input bus is sampled once during each digit time (only during the ?Fn=1 instruction) of an external bus cycle to communicate external conditions to the CPU. This signal is precharged High during the last clock cycle of every instruction.

fo_bus Flag Output Bus

Output, 6V. The flag output bus reflects the state of the FO register, during the first eight digit times on the external bus.

isa_bus Instruction/Address Bus

Bi-directional, 6V. The instruction/address bus is main system bus for the NEWT processor, carrying both the instruction address pointer and the instruction itself. Timing for the instruction/address bus is determined by the **sync** signal, which marks the instruction time on this bus.

The instruction/address bus is high-impedance (but should be pulled Low with an external resistor) during power-down (deep sleep) and light sleep. Forcing the **isa_bus** signal High during power-down will initiate the power-up sequence. Forcing the **isa_bus** signal High during light sleep will wake up the CPU.

The instruction/address bus is also used to transfer one of sixteen peripheral flag inputs to the CY flag when an intelligent peripheral returns control of the bus to the CPU. This transfer occurs during the first clock cycle of the instruction following the instruction which returns control to the CPU.

kc0-kc6 Keyboard Scanner Column Outputs

Outputs, 3.3V, active-Low. The keyboard scanner column outputs are activated sequentially at the start of each machine cycle on the external bus. During power-down (deep sleep) **kc0** is Low and all other column outputs are floating (but will be pulled High by external pull-up resistors). During light sleep all of the column outputs, **kc0**-**kc6**, are driven Low. The column outputs are precharged High during the **ph2** time at the start of every digit time so that high-value pull-up resistors can be used to conserve power.

kr0-kr7, por Keyboard Scanner Row Inputs

Inputs, 3.3V, active-Low. The keyboard scanner row inputs are sampled once during each column scan digit time. During power-down a Low on the **por** input will initiate the power-up sequence. All of the row inputs need external pull-ups.

lld Low Battery Level Detect

Input, 3.3V, active-Low. The low battery level detect signal is used to indicate a low battery voltage. This signal does nothing by itself, but must be sampled by a dedicated instruction.

MEMADDR[22:0] Physical Memory Address Bus

Outputs, 3.3V. The physical memory address bus carries the address for an access of physical memory.

MEMCS0B Physical Memory Chip Select 0

Output, 3.3V, active Low. The physical memory chip select 0 is active when accessing the lower half of the physical memory space, which is normally populated with Flash memory. This signal will only be active during a memory access, and is High at all other times. During power-down this signal is not driven, which is consistent with the Flash memory being unpowered.

MEMCS1B Physical Memory Chip Select 1

Output, 3.3V, active Low. The physical memory chip select 1 is active when accessing the upper half of the physical memory space, which is normally populated with RAM memory. This signal will only be active during a memory access, and is High at all other times, including power-down.

MEMDATA[15:0] Physical Memory Data Bus

Bi-directional, 3.3V. The physical memory data bus carries the data to and from physical memory. When no memory read or write is in progress (which includes light sleep) all data lines are driven with the contents of the lower 16-bits of the C register.

MEMRDB Physical Memory Read Strobe

Output, 3.3V. The physical memory read strobe enables data from either Flash or RAM. This signal will only be active during a memory access, and is High at all other times. During power-down this signal is not driven, which is acceptable given that the Flash is powered down and the RAM chip select is inactive.

MEMWRB Physical Memory Write Strobe

Output, 3.3V. The physical memory write strobe transfers data into either Flash or RAM. This signal will only be active during a memory access, and is High at all other times. During power-down this signal is not driven, which is acceptable given that the Flash is powered down and the RAM chip select is inactive.

MODE41 41C Memory Mapping Enable

Input, 3.3V. Tying the 41C memory mapping enable High forces the NEWT operation to mimic that of the 41C operation with regards to “holes” in the register map. Refer to the Memory Organization chapter for the details.

PDATA[7:0] Parallel Data

Outputs, 3.3V. This byte-wide output port carries various internal signals useful for debugging:

PDATA[7:5] reports the current Turbo state.

PDATA[4] is High while the serial port is operating in the clocked serial mode.

PDATA[3] is High while the peripheral address register contains 0xF0, selecting the internal peripheral port.

PDATA[2] is High while an external peripheral is in control of the bus.

PDATA[1] is High during the time that the address is on the **isa_bus** signal.

PDATA[0] is High during the last clock cycle of an instruction. This is the precharge signal for the **fi_bus** signal.

PPLS Parallel Data Pulse

Output, 3.3V. This signal toggles every time that the Parallel Control Register is written. This register is at Peripheral address 0xF0 with register address 0x8.

ph1 Clock Phase 1

Output, 6V. The clock phase 1 is used to sample Nut-compatible inputs. Note that both the **data_bus** and **isa_bus** signals are precharged Low while **ph1** is High.

ph2 Clock Phase 2

Output, 6V. The clock phase 2 signal rising edge defines bit times on the external bus. Most Nut-compatible outputs change state on the rising edge of **ph2**.

pwo Power On

Output, 6V. The power on signal indicates that the processor is powered up and running. The edges on the power on signal occur at a specific time during the instruction cycle to synchronize other system components. The power on signal is de-asserted by a dedicated instruction.

The original Nut design allowed the **pwo** signal to be asserted external to the CPU to force a power-down. This feature is supported by the NEWT microprocessor, but we are not aware of any peripheral devices that take advantage of this feature.

pwr_up Power-Up Indicator

Input, 3.3V. The power-up indicator signal is a NEWT-specific signal that goes active shortly after the system has been connected to the battery. This function is required to guarantee that the MMU is disabled when first starting up a NEWT system.

SDIO_CEB SDIO Port Chip Enable

Output, 3.3V. This is the active-Low chip enable for the SDIO port. It is controlled by a bit in the UART/SDIO Control Register. This register is at Peripheral address 0xF0 with register address 0xB.

SDIO_DI SDIO Port Data Input

Input, 3.3V. This is the data input for the SDIO port. Data is transferred from the SDIO port to the processor via the UART/SDIO Data Register. This register is at Peripheral address 0xF0 with register address 0xB.

SDIO_DO SDIO Port Data Output

Output, 3.3V. This is the data output for the SDIO port. Data is transferred from the processor to the SDIO port via the UART/SDIO Data Register. This register is at Peripheral address 0xF0 with register address 0xB.

SDIO_SCK SDIO Port Serial Clock

Output, 3.3V. This is the serial clock output for the SDIO port. The serial clock is automatically generated for an SDIO data transfer, pulsing eight times to transfer a byte of data in each direction.

SDIO_WPB SDIO Port Write Protect

Output, 3.3V. This is the active-Low write protect signal for the SDIO port. It is controlled by a bit in the UART/SDIO Control Register. This register is at Peripheral address 0xF0 with register address 0xB.

ser_rx UART Port Data Input

Input, 3.3V. This is the data input for the UART port. Data is transferred from the UART port to the processor via the UART/SDIO Data Register. This register is at Peripheral address 0xF0 with register address 0xB.

ser_tx UART Port Data Output

Output, 3.3V. This is the data output for the UART port. Data is transferred from the processor to the UART port via the UART/SDIO Data Register. This register is at Peripheral address 0xF0 with register address 0xB.

sync Synchronization

Output, 6V. The synchronization signal is High during the instruction time on the **isa_bus** signal. This is what allows intelligent peripherals to monitor and respond to the instruction

stream. The synchronization signal is unique because during power-down it must reflect the state of the **dpwo** input, to communicate this information to any external devices connected to the HP-41 bus.

vci Vcc Request

Output, 3.3V. The vcc request signal is activated when any of the power-up conditions have been detected, and is used to enable the master oscillator and the voltage regulators that supply the powered-down section of the design. In the original Nut design, the **vci** signal was actually a current signal that turned on the dedicated power-supply chip.

vco Vcc Acknowledge

Output, 3.3V. The vcc acknowledge signal is activated when the voltage regulators for the powered-down section of the design are stable and the oscillator is running. The signal is not usually used in a NEWT system, but is effectively the active-Low reset signal for the NEWT processor.

The table below provides a summary of all of the NEWT signals.

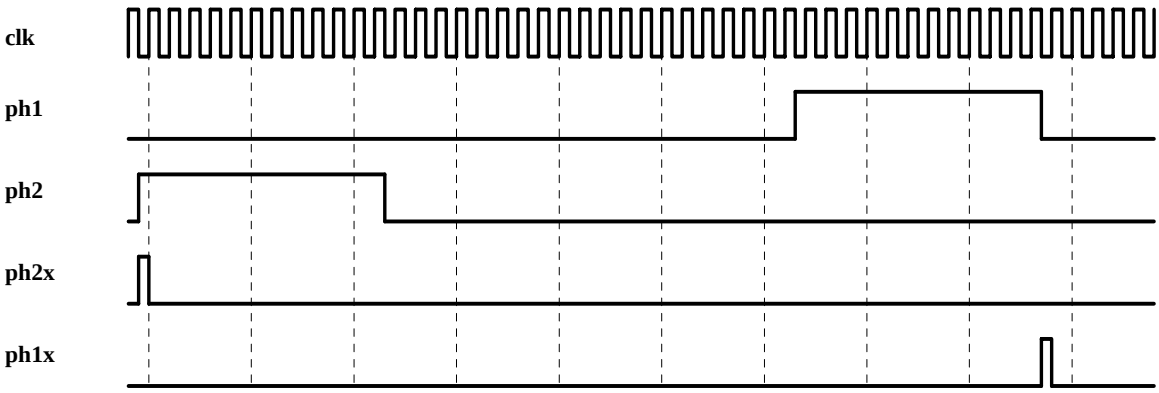
Signal Name(s)	Voltage	Direction	Related Signals	Light Sleep	Deep Sleep
clk	3.3V	input		Running	Low
data_bus	6V	bidi	DATA_BUS_IN DATA_BUS_HI DATA_BUS_EN	Low (ext pulldown)	Low (ext pulldown)
dpwo	6V	input	DPWO_IN	High	Low
fi_bus	6V	input	FI_BUS_IN	Low (ext_pulldown)	Low (ext pulldown)
fo_bus	6V	output	FO_BUS_HI	Low	Undriven
isa_bus	6V	bidi	ISA_BUS_IN ISA_BUS_HI ISA_BUS_EN	Low (ext pulldown)	Low (ext pulldown)
kc0-kc6	3.3V	outputs	KC0_HI - KC6_HI	all Low	Low (only kc0)
kr0_kr7, por	3.3V	inputs	KR0_HI - KR7_HI, POR	High (ext pullup)	High (ext pullup)
lld	3.3V	input	LLD_IN	High	Low
MEMADDR	3.3V	outputs		all Low	Undriven
MEMCS0B	3.3V	output		High	Undriven
MEMCS1B	3.3V	output	mem_cs1	High	High
MEMDATA	3.3V	bidi		all Low	Undriven
MEMRDB	3.3V	output		High	Undriven
MEMWRB	3.3V	output		High	Undriven

MODE41	3.3V	input		High	Low
PDATA	3.3V	output		Low	Undriven
PPLS	3.3V	output		Low	Undriven
ph1	6V	output	PH1_HI	Low	Undriven
ph2	6V	output	PH2_HI	Low	Undriven
pwo	6V	output	PWO_HI	Low	Low
pwr_up	3.3V	input		High (ext pullup)	High (ext pullup)
SDIO_CEB	3.3V	output		High	Undriven
SDIO_DI	3.3V	input			
SDIO_DO	3.3V	output		High	Undriven
SDIO_SCK	3.3V	output		High	Undriven
SDIO_WPB	3.3V	output		High	Undriven
ser_rx	3.3V	input			
ser_tx	3.3V	output		High	Undriven
sync	6V	output	SYNC_HI SYNC_DRV	dpwo state	Low
vci	3.3V	output		High	Low
vco	3.3V	output	VCO_IN	Low	Low

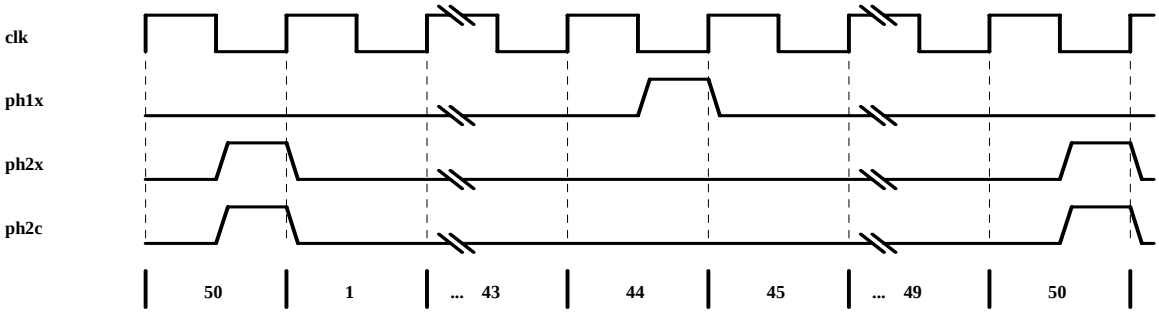
Timing

As mentioned previously, the input clock frequency for the NEWT microprocessor is 18MHz. This input clock is divided by fifty to create the native 360KHz frequency of the original Nut design. This 360KHz is the 1x frequency for the NEWT design, and is the only frequency seen on the external bus.

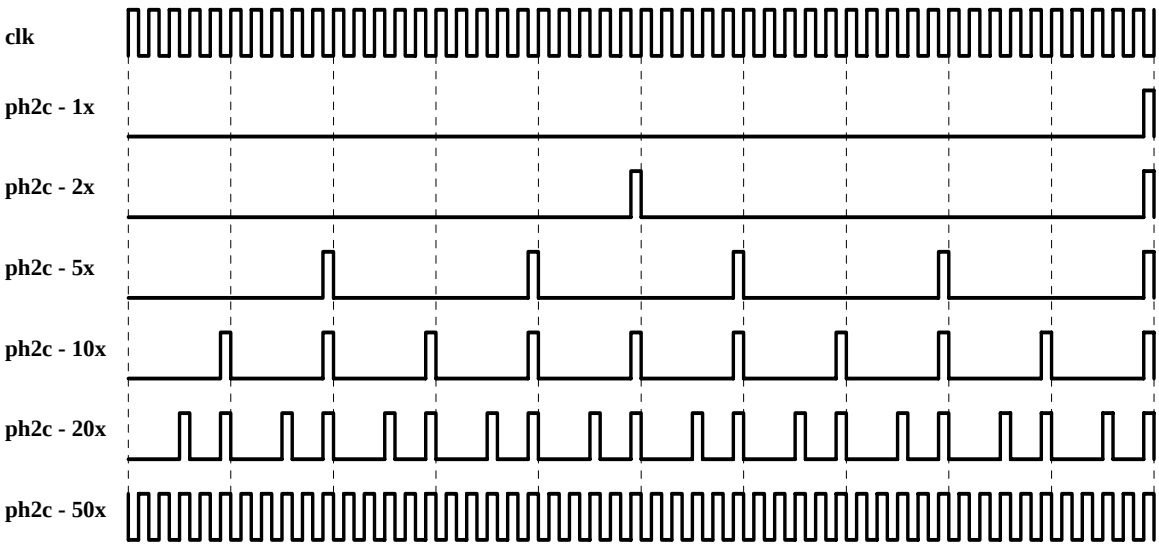
The figure below shows the relationship between the input clock, the external clock signals, and the actual internal clock pulses used in the design. These internal clocks are timed to allow for matching the timing of the Nut design. The dotted lines refer to states in the internal divider.



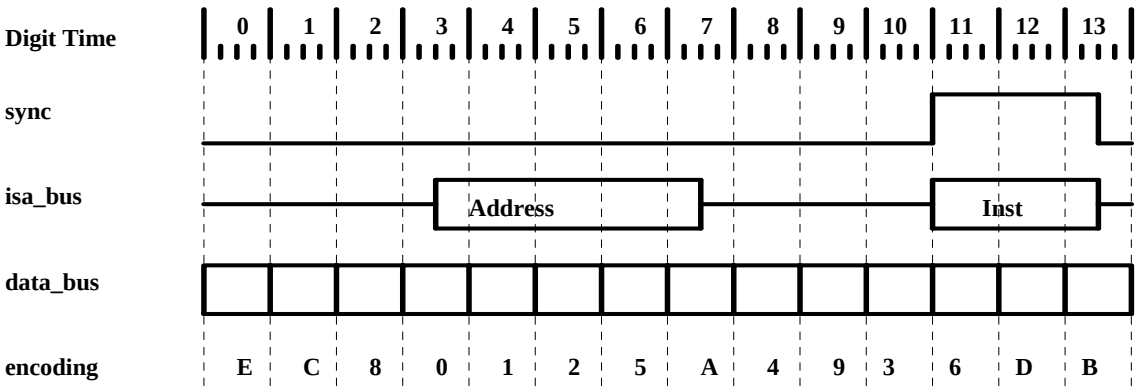
The ph1x and ph2x signals are used by the external bus portion of the design. A separate clock, called ph2c, is used by the logic of the CPU. When running in 1x mode ph2x and ph2c have the same timing. The details of this timing is shown in the figure below.



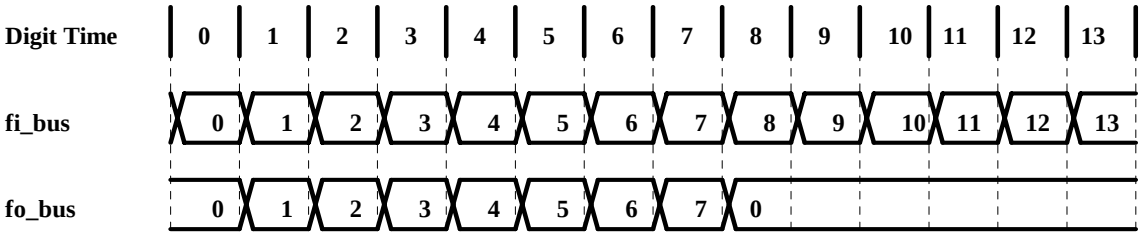
When running in one of the turbo modes, the external bus and ph2x are unaffected. However, the ph2c signal frequency is increased as shown in the figure below. Note that the ph2c signal is symmetric in all cases except for the 20x case.



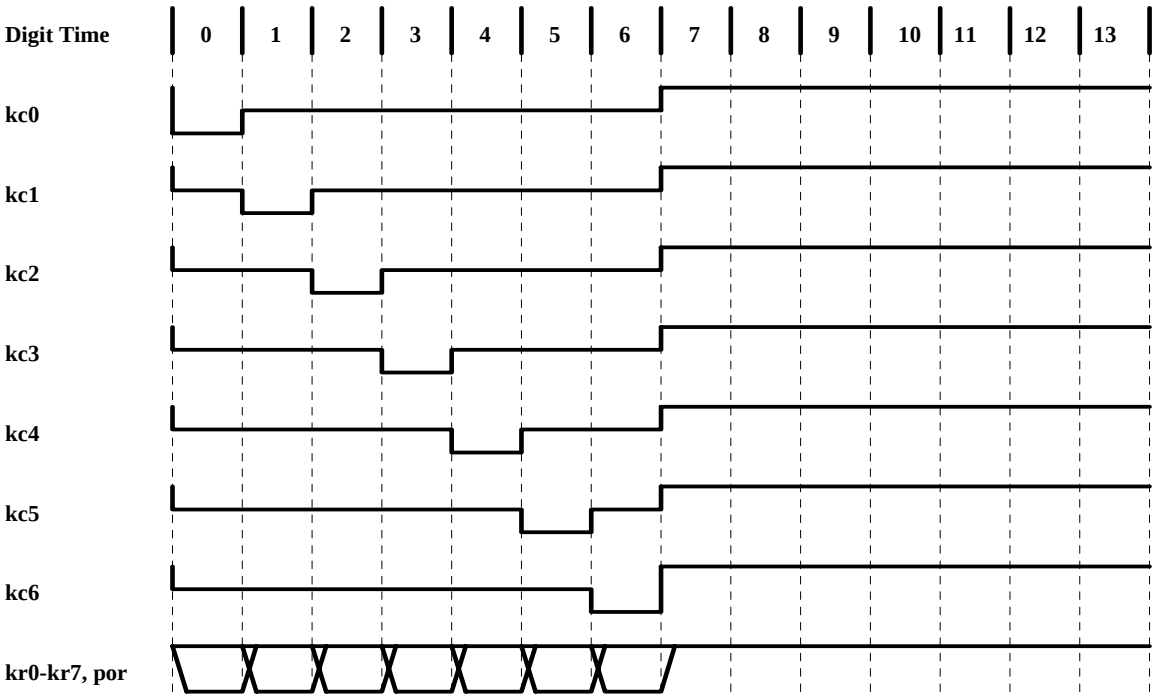
Both the external bus and the NEWT CPU operate with fixed machine cycles that are fifty-six clock (ph2x or ph2c) cycles long. These fifty-six clock cycles correspond to the fifty-six bits (or fourteen digits) of the Nut data word. The figure below shows a NEWT machine cycle.



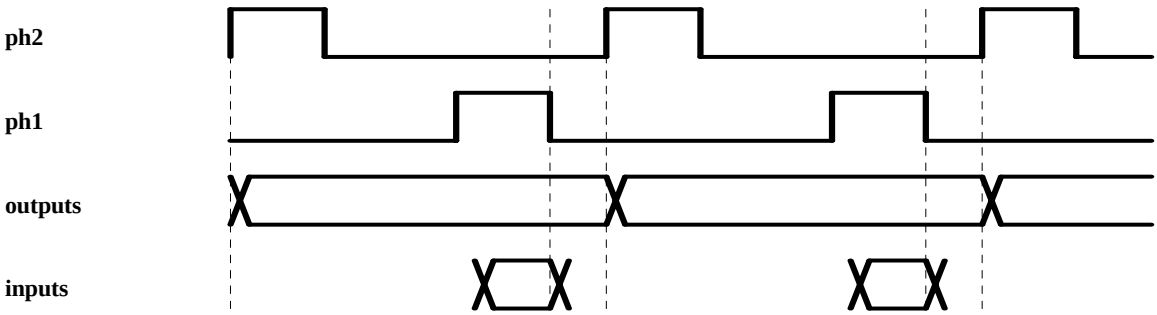
Both the flag input signal and the flag output signal always operate at 1x speed for compatibility. The timing for these two signals is shown in the figure below. Note that the state of bit 0 of the flag output register persists from the beginning of digit 8 time through the end of the machine cycle and for the first digit of the subsequent machine cycle. The state of the **fi_bus** signal is actually sampled by the second **ph1** clock in each digit time.



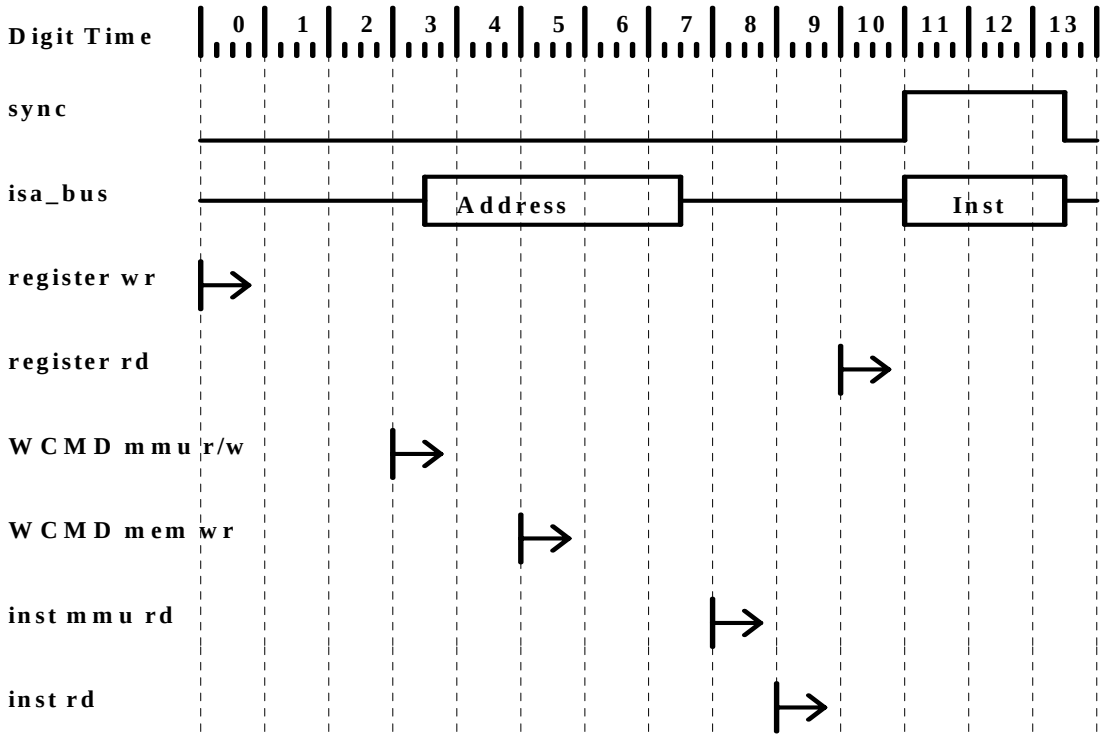
The keyboard scanner always operates at 1x speed. The timing for the column drive signals is shown in the figure below. The row inputs are sampled by the last **ph1** clock in the digit times where a column drive signal is active. During each of the first seven digit times one column drive signal is driven Low while the rest are High-impedance, to prevent excessive current consumption if two keys on the same column are down at the same time. All of the column outputs are precharged High during the first **ph2** of each digit time.



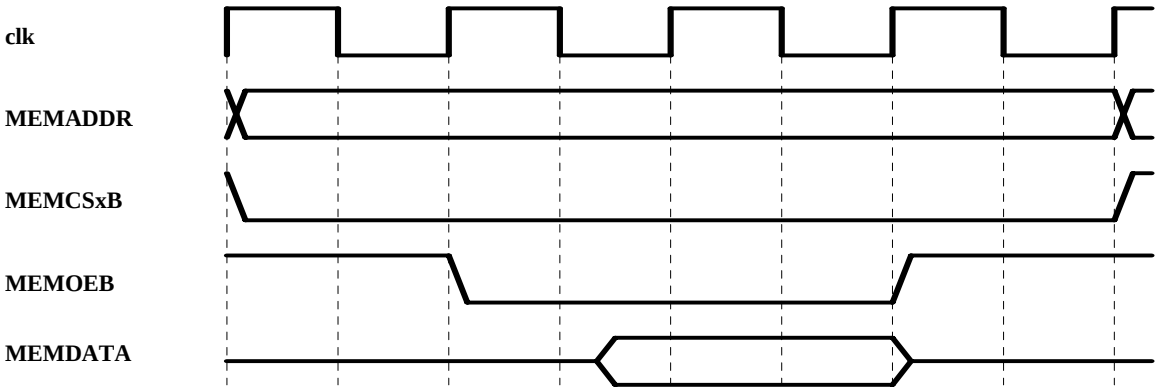
All Nut-compatible NEWT outputs change state relative to the rising edge of the **ph2** signal and all Nut-compatible inputs are sampled by the falling edge of the **ph1** signal. The original Nut documentation is vague on these timing details, but this is what can be inferred from the available information. This timing is shown in the figure below.



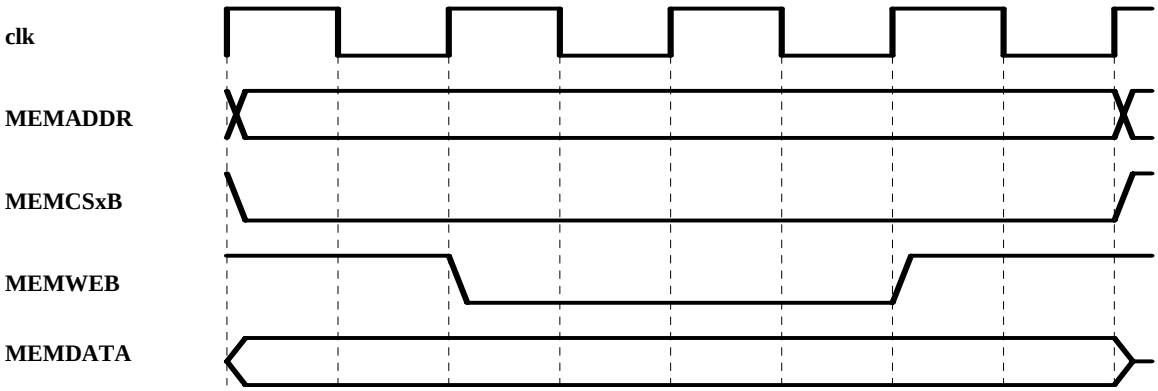
The NEWT CPU starts memory accesses of the physical memory at specific times during machine cycles, independent of the Turbo mode. Register accesses consist of four successive accesses. Note that register instructions are always executed in 1x mode, so all four physical memory accesses complete well before the end of the first bit time.



Memory accesses are always four clock cycles (of the input clock) in length. When not performing a memory access, the memory control signals are all inactive, the memory address is driven with all zeros, and the memory data bus is also driven with all zeros. The timing for a memory read is shown in the figure below.



The timing for a memory write is shown in the figure below.

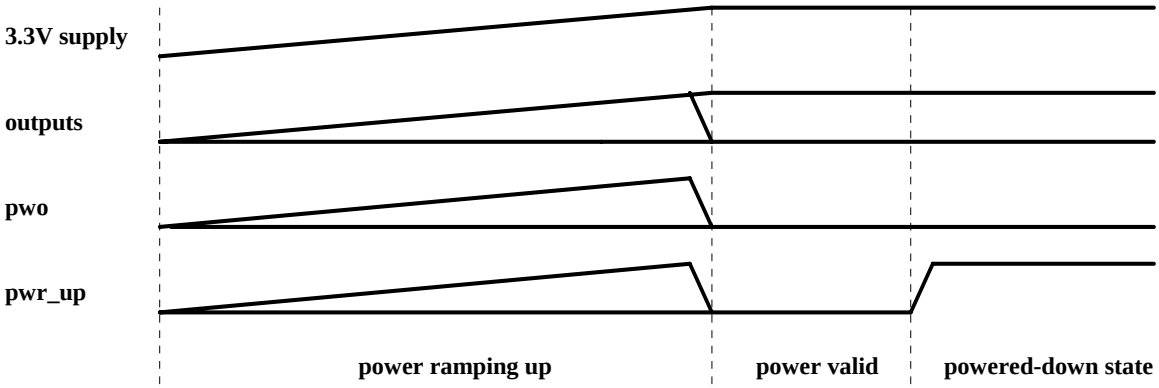


Power Control

The NEWT processor is partitioned into two sections, one section that is always powered, and one section that is powered down when the system is not in use. In addition to the obvious powered and powered-down (deep sleep) states, there is also a third state, called light sleep. In this state the entire processor is powered and the oscillator is running, but the CPU is held in reset. These three states can be identified by the states of the **pwo** and **dpwo** signals according to the table below. The **dpwo** signal is an input to the NEWT processor that comes from the display controller, and remains active for approximately 10 minutes before going Low and forcing the powered-down state. Note that the final state is transient, as the display controller drives **dpwo** High in response to **pwo** going High.

dpwo	**pwo**	State	Operation
Low	Low	Deep Sleep	System powered down
Low	High	illegal	powering up (transient state)
High	Low	Light Sleep	System in standby
High	High	Running	System running

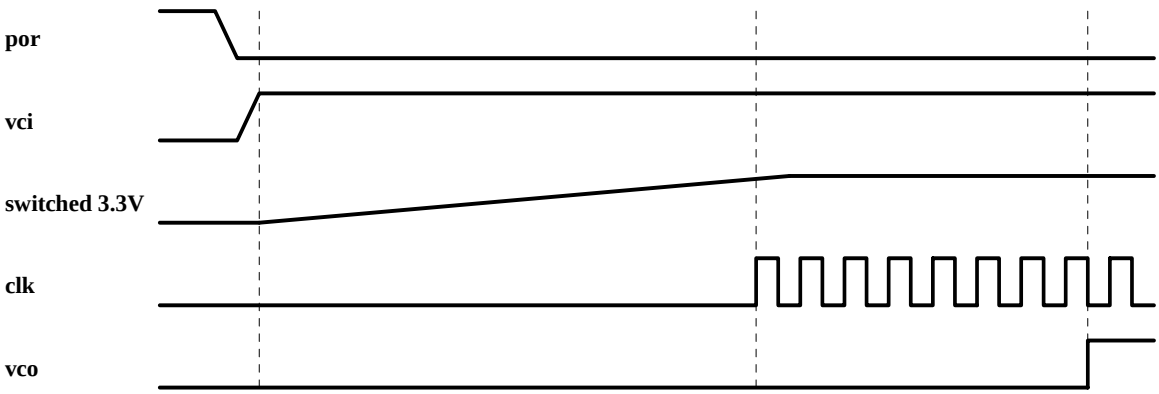
When a NEWT system is first powered the **pwr_up** signal guarantees that the system will be in the powered-down state by initializing the necessary flip-flops in the powered section. This is the only use of the **pwr_up** signal, and an example of the sequence is shown below. No time scale is implied. The **pwr_up** signal must only remain Low long enough to guarantee the state of flip-flops in the powered section.



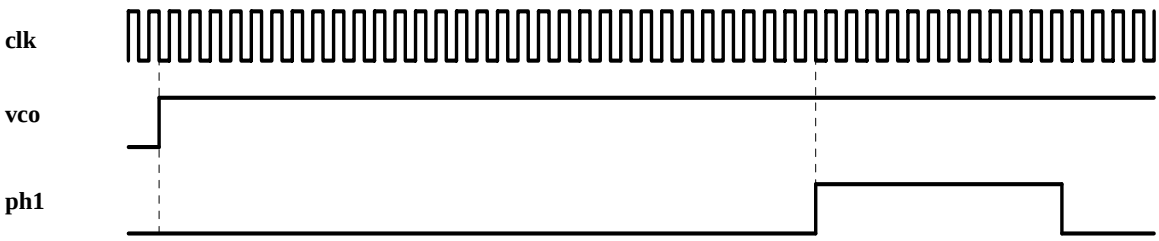
In the powered-down state the logic in the powered section watches for one of two conditions to initiate the power-up sequence. The first condition is a Low on the **por** input. This corresponds to a user pressing the ON key on the calculator. This condition is possible because the keyboard column output **kc0** is forced Low during power-down, while all of the keyboard row inputs have passive pull-ups.

The second condition is a High on the **isa_bus** signal. This condition is used by external peripherals to start CPU operation. This condition is possible because there is a high-impedance pull-down on the **isa_bus** signal that normally keeps it Low.

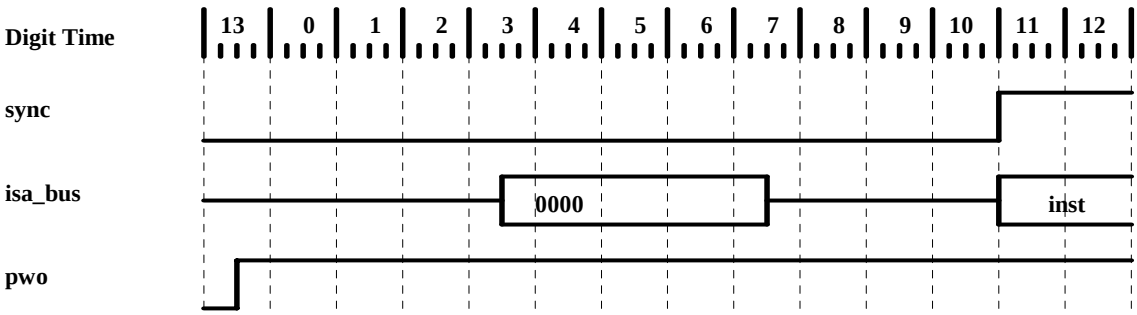
The transition from powered-down to powered is shown below. The **vci** signal is the enable for the main oscillator as well as the power supplies for the powered-down section. Note that in reality it takes the oscillator and power supplies some time to stabilize, so the **vco** signal is not asserted until after 16,384 clock cycles (about 1mS).



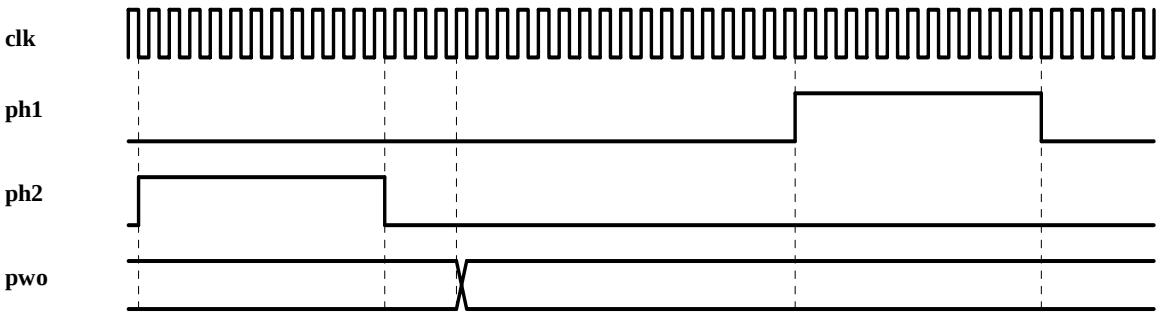
Once the **vco** signal is asserted, the NEWT exits the reset state and begins operation. It takes thirty-two clock cycles before the first **ph1** is output. At this point the NEWT is operating normally, at 1x speed.



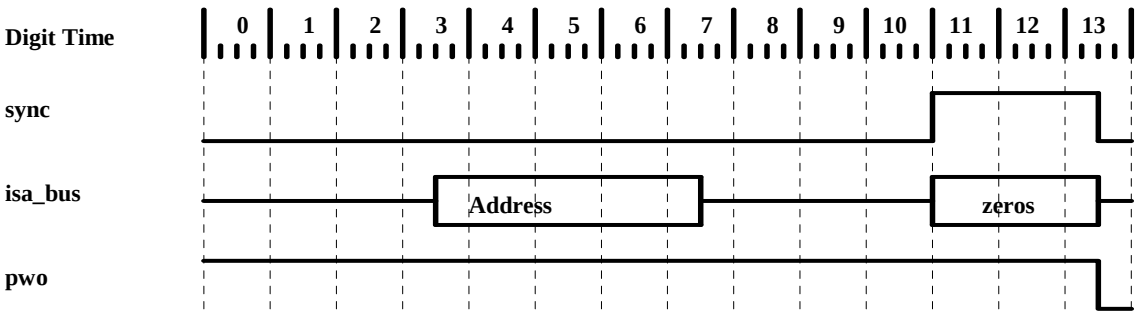
Even though the NEWT processor has begun operation at this point, the remainder of the system requires the **pwo** signal to be asserted to synchronize all system components to the NEWT machine cycle. These other components in the system require that the **pwo** signal change state at a specific time in the machine cycle, shown below.



The **pwo** signal has unique timing, changing state between the **ph2** and **ph1** signal. The specific timing is shown in the figure below.



The power-up state is where all user-visible system operation occurs. Once all operations are complete, a POWOFF instruction is executed to place the system in either the power-down state or the Standby state. The timing for the falling edge of the **pwo** signal is the same as the timing for the rising edge. The **pwo** signal changes state in the machine cycle following the machine cycle that fetches the POWOFF instruction. If entering the power-down state, execute the DISOFF instruction prior to the POWOFF instruction to reduce the **dpwo** signal delay time to zero.



I/O Ports

The NEWT microprocessor contains three on-chip I/O ports that were not present in the original NUT design. These three I/O ports are addressed via the PFAD=C instruction with an address of 0xF0. The following registers are available for control of these I/O ports:

Address	Register function	Read/Write
0xF	UART/SDIO Data	Read/Write
0xE	UART/SDIO Rx Status	Read
0xD	UART/SDIO Tx Status	Read
0xC	UART/SDIO Rate	Write
0xB	UART/SDIO Control	Write
0xA	Master Reset	Write
0x8	Parallel Control	Write
0x6	MMU Status	Read
0x4	Turbo Status	Read
0x2	WCMD Read Data	Read

The parallel control toggles the **ppls** signal with each write to the register.

The UART supports full-duplex 8N1 (the transmitter actually sends two stop bits) asynchronous communication, with one byte of buffering for both the receiver and one byte for the transmitter. The baud rate set by a dedicated divider running off of the 18MHz system clock.

The SDIO interface shares logic with the UART, so these two ports cannot be used at the same time. This interface supports serial flash devices under software control.

The WCMD Read Data register allows software to access any location in the physical address space using one of the three WCMD Read commands (MMU register, logically addressed memory location, or physically addressed memory location). The memory data is latched by the WCMD Read command and is available until rewritten by another WCMD Read command.

The Turbo Status register allows software to determine the current Turbo Mode in effect. The MMU Status register returns the enable/disable status of the MMU.

UART/SDIO Data Register (USDR) (Address = 0xF)		
Bit(s)	**Value**	**Description**
15:8		These bits are always zero.
7:0	Read	Returns the contents of the receive buffer.
	Write	Loads the transmit buffer with a data byte for transmission.

UART/SDIO Rx Status Register (USRSR) (Address = 0xE)		
Bit(s)	**Value**	**Description**
15:5		These bits are always zero.
4	0	The receive buffer was not overrun.
(rd-only)	1	The receive buffer was overrun. This bit is cleared by reading the receive buffer.
3:1		These bits are always zero.
0	0	The receive buffer is empty
(rd-only)	1	There is at least one byte in the receive buffer.

UART/SDIO Tx Status Register (USTSR) (Address = 0xD)		
Bit(s)	**Value**	**Description (Async mode only)**
15:5		These bits are always zero.
4	0	The transmitter is sending a byte.
(rd-only)	1	The transmitter is idle. This bit is set when the last bit has been sent. This is bit 0 of the data in SDIO mode and the Stop bit in Async mode.
3:1		These bits are always zero.
0	0	The transmit buffer is not empty.
(rd-only)	1	The transmit buffer is empty.

UART/SDIO Rate Register (USRR) (Address = 0xC)		
Bit(s)	**Value**	**Description**
15:12		These bits are always zero.
11:0 (wr-only)		The divider that generates the serial clock for the UART/SDIO. If the contents of this register are n, the counter counts modulo n+1.

UART/SDIO Control Register (USCR) (Address = 0xB)		
Bit(s)	**Value**	**Description**
15:9		These bits are always zero.
8 (wr-only)	0	Drive SDIO WP# signal High.
	1	Drive SDIO WP# signal Low.
7:5		These bits are always zero.
4 (wr-only)	0	Drive SDIO CE# signal High.
	1	Drive SDIO CE# signal Low.
3:1		These bits are always zero.
0 (wr-only)	0	UART enabled. Data is sent and received LSB-first.
	1	SDIO enabled. Data is sent and received MSB-first.

Master Reset Register (MRR) (Address = 0xA)		
Bit(s)	**Value**	**Description**
15:1		These bits are always zero.
0	0	No effect.
	1	Reset the UART and SDIO. This command must be issued before attempting to program any of the UART or SDIO Registers, because these peripherals are not affected by the normal Reset.

Parallel Control Register (PCR) (Address = 0x8)		
Bit(s)	**Value**	**Description**
15:0	Write	Toggle **ppls** signal.

MMU Status Register (MSR) (Address = 0x6)		
Bit(s)	**Value**	**Description**
15:1		These bits are always zero.
0 (rd-only)	0	MMU is disabled.
	1	MMU is enabled.

Turbo Status Register (TSR) (Address = 0x4)		
Bit(s)	**Value**	**Description**
15:3		These bits are always zero.
2:0 (rd-only)	000	Normal operating speed.
	001	2X Turbo Mode in effect.
	010	5X Turbo Mode in effect
	011	10X Turbo Mode in effect.
	100	20X Turbo Mode in effect.
	101	50X Turbo Mode in effect.
	11x	These bit combinations are reserved.

WCMD Read Data Register (WRDR) (Address = 0x2)		
Bit(s)	**Value**	**Description**
15:0	Read	Data latched by last WCMD Read.

Pin Assignments

The NEWT design is implemented in two separate packages. The powered-down portion of the design uses a 144-pin Actel A3P125 FPGA. The pin assignments for this portion of the design are shown below:

pin	signal	pin	signal
1	MEMADDR[8]	73	VPUMP
2	MEMADDR[19]	74	no connect
3	MEMADDR[18]	75	TDO
4	MEMADDR[17]	76	TRST
5	MEMADDR[10]	77	VJTAG
6	MEMADDR[9]	78	DPWO_IN
7	MEMADDR[7]	79	LLD_IN
8	MEMADDR[6]	80	PWO_IN
9	1.5V	81	3.3V
10	GND	82	GND
11	3.3V	83	POR_IN
12	MEMADDR[12]	84	PWO_RES
13	MEMADDR[11]	85	MMU_SET
14	MEMADDR[5]	86	MMU_RES
15	MEMADDR[4]	87	MMU_EN
16	MEMADDR[14]	88	SYNC_HI
17	GND*	89	VCO_IN
18	CLK_IN	90	no connect
19	GND*	91	SDIO_WPB
20	MEMADDR[13]	92	SDIO_CEB
21	MEMADDR[3]	93	UART_TX
22	MEMADDR[2]	94	UART_RX
23	MEMADDR[16]	95	no connect
24	MEMADDR[15]	96	no connect
25	MEMADDR[0]	97	no connect
26	MEMADDR[1]	98	3.3V
27	GND	99	GND
28	3.3V	100	1.5V

29	MEMADDR[20]	101	mem_cs1
30	MEMADDR[21]	102	no connect
31	MEMCS0B	103	KR7_IN
32	MEMRDB	104	KR6_IN
33	MEMWRB	105	KR5_IN
34	MEMADDR[22]	106	KR4_IN
35	3.3V	107	3.3V
36	GND	108	GND
37	no connect	109	MEMDATA[12]
38	FO_BUS_HI	110	MEMDATA[4]
39	ISA_BUS_HI	111	MEMDATA[11]
40	ISA_BUS_EN	112	MEMDATA[3]
41	DATA_BUS_HI	113	MEMDATA[13]
42	DATA_BUS_EN	114	MEMDATA[5]
43	PWO_HI	115	no connect
44	PH2_HI	116	KR3_IN
45	1.5V	117	3.3V
46	GND	118	GND
47	3.3V	119	1.5V
48	PPLS	120	KR2_IN
49	PH1_HI	121	KR1_IN
50	PDAT[0]	122	KR0_IN
51	PDAT[1]	123	KC6_HI
52	PDAT[2]	124	KC5_HI
53	PDAT[3]	125	MEMDATA[10]
54	PDAT[4]	126	MEMDATA[2]
55	PDAT[5]	127	MEMDATA[14]
56	PDAT[6]	128	MEMDATA[6]
57	PDAT[7]	129	MEMDATA[9]
58	SDIO_DI	130	MEMDATA[1]
59	SDIO_DO	131	MEMDATA[15]
60	SDIO_SCK	132	MEMDATA[7]
61	MODE41	133	KC4_HI
62	1.5V	134	3.3V
63	GND	135	GND
64	3.3V	136	1.5V
65	FI_BUS_IN	137	KC3_HI
66	ISA_BUS_IN	138	KC2_HI
67	DATA_BUS_IN	139	KC1_HI
68	GND	140	KC0_HI
69	TCK	141	MEMDATA[8]
70	TDI	142	MEMDATA[0]
71	TMS	143	GND
72	3.3V	144	3.3V

The always-powered portion of the design uses a 100-pin Xilinx XC2C64A CPLD. The pin assignments for this portion of the design are shown below:

pin	signal	pin	signal
1	KR4_IN	51	3.3V
2	KR5_IN	52	kc3
3	KR6_IN	53	kr4
4	KR7_IN	54	no connect
5	VJTAG	55	kc2
6	no connect	56	kc4
7	mem_cs1	57	1.8V
8	UART_RX	58	SYNC_DRV
9	UART_TX	59	no connect
10	no connect	60	pwo
11	no connect	61	fi_bus
12	no connect	62	GND
13	VCO	63	no connect
14	SYNC_HI	64	data_bus
15	MMU_EN	65	no connect
16	MMU_RES	66	no connect
17	MMU_SET	67	isa_bus
18	PWO_RES	68	lld
19	POR_IN	69	GND
20	no connect	70	vci
21	GND	71	ser_tx
22	PWO_IN	72	ser_rx
23	LLD_IN	73	no connect
24	DPWO_IN	74	MEMCS1B
25	no connect	75	no connect
26	1.8V	76	KC0_HI
27	clk	77	KC1_HI
28	DATA_BUS_IN	78	KC2_HI
29	ISA_BUS_IN	79	KC3_HI
30	FI_BUS_IN	80	no connect
31	GND	81	KC4_HI
32	dpwo	82	no connect
33	kc6	83	TDO
34	kc5	84	GND
35	kc0	85	no connect
36	kc1	86	no connect
37	kr0	87	no connect
38	3.3V	88	3.3V

39	**kr1**	**89**	**KC5_HI**
40	**kr2**	**90**	**KC6_HI**
41	**kr3**	**91**	**KR0_IN**
42	**por**	**92**	**KR1_IN**
43	**kr7**	**93**	**no connect**
44	**no connect**	**94**	**KR2_IN**
45	**TDI**	**95**	**no connect**
46	**no connect**	**96**	**no connect**
47	**TMS**	**97**	**KR3_IN**
48	**TCK**	**98**	**3.3V**
49	**kr6**	**99**	**pwr_up**
50	**kr5**	**100**	**GND**

Instruction Set

This Appendix presents the NEWT instruction set organized alphabetically.

Instruction	Opcode 1	Opcode 2	1x	always seen on bus
?A#0	1101_0ttt_10			
?A#C	1101_1ttt_10			
?A<B	1100_1ttt_10			
?A<C	1100_0ttt_10			
?B#0	1011_0ttt_10			
?C#0	1011_1ttt_10			
?Fd=1	dddd_1011_00		yes	
?LLD	0101_1000_00			
?P=Q	0100_1000_00			
?PT=d	dddd_0101_00			
?Sd=1	dddd_0011_00			
A=0	0000_0ttt_10			
A=A+1	0101_1ttt_10			
A=A+B	0100_1ttt_10			
A=A+C	0101_0ttt_10			
A=A-1	0110_1ttt_10			
A=A-B	0110_0ttt_10			
A=A-C	0111_0ttt_10			
A=C	0100_0ttt_10			
ABEX	0001_1ttt_10			
ACEX	0010_1ttt_10			
ASL	1111_1ttt_10			
ASR	1110_0ttt_10			
B=0	0000_1ttt_10			
B=A	0010_0ttt_10			
BCEX	0011_1ttt_10			

BSR	1110_1ttt_10			
C=0	0001_0ttt_10			
C=A+C	1000_0ttt_10			
C=A-C	1001_0ttt_10			
C=B	0011_0ttt_10			
C=C+C	0111_1ttt_10			
C=C+1	1000_1ttt_10			
C=C-1	1001_1ttt_10			
C=C|A	1101_1100_00			
C=C&A	1110_1100_00			
C=-C	1010_0ttt_10			
C=-C-1	1010_1ttt_10			
C=DATA	0000_1110_00		if peripheral	if peripheral
C=REGn	nnnn_1110_00		if peripheral	if peripheral
C=G	0010_0110_00			
C=KEYS	1000_1000_00		yes	
C=M	0110_0110_00			
C=N	0010_1100_00			
C=ST	1110_0110_00			
C=STK	0110_1100_00			
CGEX	0011_0110_00			
CHKKB	1111_0011_00		yes	
CLRABC	0110_1000_00			
CLRDATA	1010_1100_00		yes	yes
CLRST	1111_0001_00			
CMEX	0111_0110_00			
CNEX	0011_1100_00			
CSR	1111_0ttt_10			
CSTEX	1111_0110_00			
CXISA	1100_1100_00			
DADD=C	1001_1100_00			
DATA=C	1011_1100_00			
DECPT	1111_0101_00			
DISOFF	1011_1000_00		yes	yes
DISTOG	1100_1000_00		yes	yes
ENROM1	0100_0000_00		yes	yes
ENROM2	0110_0000_00		yes	yes
ENROM3	0101_0000_00		yes	yes
ENROM4	0111_0000_00		yes	yes

F=SB	1001_0110_00		yes	
FEXSB	1011_0110_00		yes	
G=C	0001_0110_00			
GOC	aaaa_aaa1_11			
GOKEYS	1000_1100_00		yes	
GOLC	aaaa_aaaa_01	aaaa_aaaa_11		
GOLNC	aaaa_aaaa_01	aaaa_aaaa_10		
GONC	aaaa_aaa0_11			
GSUBNC	aaaa_aaaa_01	aaaa_aaaa_00		
GSUBC	aaaa_aaaa_01	aaaa_aaaa_01		
GOTOC	0111_1000_00			
INCPT	1111_0111_00			
LC	nnnn_0100_00			
LDI	0100_1100_00	constant		
M=C	0101_0110_00			
N=C	0001_1100_00			
NOP	0000_0000_00			
POWOFF	0001_1000_00	0000_0000_00	yes	yes
PT=d	dddd_0111_00			
RCRd	dddd_1111_00			
REGn=C	nnnn_1010_00			
RSTKB	1111_0010_00		yes	
RTN	1111_1000_00			
RTNC	1101_1000_00			
RTNNC	1110_1000_00			
SB=F	1010_0110_00		yes	
Sd=0	dddd_0001_00			
Sd=1	dddd_0010_00			
SELP	0010_1000_00			
SELPF	nnnn_1001_00		yes	yes
SELQ	0011_1000_00			
SETDEC	1010_1000_00			
SETHEX	1001_1000_00			
SPOPND	0000_1000_00			
ST=C	1101_0110_00			
STK=C	0101_1100_00			
WCMD	0111_1111_00		yes	yes
WROM	0001_0000_00		yes	yes