# Technical Notes

## polyFORTH on the NCR/32

*Michael L. McBride*
*VLSI Processor Products*
*NCR Microelectronics*
*1635 Aeroplaza Drive*
*Colorado Springs, Co. 80916*

### *Introduction*

A very high performance polyFORTH II system implemented on the NCR/32 chip set was presented at the 1984 Rochester Forth Applications Conference. This technical note will clarify some aspects of the implementation.

### *Overview of the NCR/32 Chip Set*

The NCR/32 chip set consists of a full 32-bit microprocessor and a complete family of support chips. The chips in the family are the NCR/32-000 Central Processor Chip (CPC), the NCR/32-010 Address Translation Chip (ATC), the NCR/32-020 Extended Arithmetic Chip (EAC), the NCR/32-500 System Interface Controller (SIC), the NCR/32-580 System Interface Transmitter (SIT), and the NCR/32-590 System Interface Receiver. These chips communicate over the Processor Memory Bus, a full 32-bit multiplexed bi-directional bus. To implement a system based on the NCR/32, an Instruction Storage Unit (ISU) and Memory Storage Unit (MSU) are also required.

#### CPC

The Central Processor Chip is a full 32-bit microprocessor. This chip is externally microprogrammable and has 16 32-bit general registers, a 32-bit ALU, a 32-bit external data path and 22 16-bit special purpose internal registers. It also has a separate 16-bit wide data path to the external microprogram store. It can address up to 16 megabytes of real memory and 4 gigabytes of virtual memory. It can address up to 64k of 16-bit wide microinstructions. The CPC will be discussed in additional detail later.

#### ATC

The Address Translation Chip performs several functions. It translates virtual memory addresses to real memory addresses; it handles Error Checking and Correcting for the memory, including generating ECC bits when data is written to memory and checking and correcting data being read from memory; it handles memory refresh and scrubbing (a technique for reading memory and correcting errors during the refresh cycles). It also provides a Time of Day clock and time interval monitoring.

### EAC

The Extended Arithmetic Chip is a performance booster for the chip set. It does multiplication and division on single and double precision fixed point data; adds, subtracts, multiplies and divides on single and double precision IBM 370 format hex floating point data; adds, subtracts, multiplies and divides on decimal floating point data; and performs data conversions between these formats.

### SIC, SIT and SIR

The System Interface Controller, Transmitter, and Receiver chips form a high speed serial communications subsystem. It can transmit and receive data at up to 24 megabits per second.

### System Clocks

The NCR/32 uses a two phase, non-overlapping clock. Each phase is 75 ns long, giving the system a cycle time of 150 ns. The basic clock frequency is 13.3 megahertz.

### PM Bus

The Processor Memory Bus is a 32-bit, multiplexed, bi-directional bus that is used for inter-chip communication.

### ISU

The Instruction Storage Unit contains the user alterable external microcode for the CPC. It consists of up to 64k by 16 bits wide of fast (45ns access) static RAM. This is accessed from the CPC over the ISU bus. The ISU bus is a 16 bit wide, multiplexed, bi-directional bus that is completely separate from the PM Bus.

### MSU

The Memory Storage Subsystem (MSU) consists of up to 16 megabytes of real memory. It is organized in a 32 bit wide word with 7 additional bits for storing the ECC codes. Using 64k memory chips, the minimum memory size is 256 kilobytes. Using 256k chips, the minimum size is 1 megabyte.

### NCR/32 Performance

The NCR/32 chip set has a basic cycle time of 150 nanoseconds. About 95 percent of the CPC microinstructions execute in one cycle. The rest of the instructions execute in 2 cycles.

### polyFORTH Implementation

The polyFORTH implementation on the NCR/32 is a full 32-bit FORTH implementation. All cells that are 16 bits on a 8-bit or 16-bit FORTH implementation are 32 bits wide on the NCR/32 polyFORTH. Double width data is 64 bits long. An option has also been provided that allows the user to extend the polyFORTH by compiling words into CPC microcode. Since the readers are assumed to be familiar with FORTH, this explanation will not go further. If desired, additional information on this polyFORTH implementation can be obtained from FORTH Inc.[1]

The following sections provide an overvew of programming the NCR/32. Additional information on programming the NCR/32 can be obtained from NCR Microelectronics.[2]

[1] FORTH Inc., 2309 Pacific Coast Highway, Hermosa Beach, Ca. (213) 372-8493.

[2] NCR Microelectronics, 1635 Aeroplaza Drive, Colorado Springs Co. 80916, (800) 525-2252 or (303) 596-5612 (In Colorado or outside the Continental U.S.)

## *CPC Hardware*

This section provides a brief description of the hardware resources in the CPC.

**Instruction Format**

The CPC has a register to register instruction format. Each instruction is 16 (or in a few cases 32) bits wide. These instructions are read and placed in the pipeline 16 bits at a time.

**Pipeline**

The CPC has a three stage pipeline for the external microinstructions. The first stage fetches the instruction from the ISU. The second stage decodes the information and the third stage executes the instruction.

**General Registers**

The CPC has 16 general purpose 32-bit general registers. All 16 registers are word or halfword addressable. The first four registers (register 0 through 3) are also byte addressable.

**Internal Registers**

The CPC has 22 16-bit special purpose registers. They are used for such functions as microinstruction jump registers, state registers, and other specific functions.

**ALU**

The CPC has a full 32-bit ALU. It can do arithmetic, boolean, and comparison operations on word (32-bit) and byte (8-bit) data.

**Field Instructions**

The CPC has implemented hardware to simplify the use of fields of data. A field can be from 1 to 64k bytes long. Special instructions are provided to perform arithmetic, comparison, boolean, and data move operations on these fields.

## *CPC Programming*

This section provides a brief overview of the CPC instructions.

**Data Move Instructions**

The CPC provides instructions for moving words, halfwords, bytes, and digits between registers. It can also move words between registers and memory. It can move halfwords between registers and internal registers and can move words between registers and external registers.

**Arithmetic Instructions**

The CPC provides binary add and subtracts on word, byte, and field data. It provides packed and unpacked decimal arithmetic on byte and field data.

**Compare Instructions**

The CPC provides compare instructions on word, byte, and field data.

**Boolean Instructions**

The CPC provides boolean (and, or, exclusive or) operations on word, byte, digit, and field data. It also provides an invert (one's complement) operation for word and byte data.

**Jumps**

The CPC provides conditional and unconditional jumps. They may be relative or absolute jumps. The offset for relative jumps is contained in the instruction. The address for absolute jumps may be contained in the instruction, in a general register, or in a jump register. One feature supported by the CPC is delayed jumps. When a jump is executed it takes three cycles to flush the pipeline and move the new instruction to the execution stage.

When an immediate jump is executed, the two instructions in the CPC pipeline are not executed. To provide better performance, the CPC has implemented delayed jumps. These jumps allow the two instructions in the pipeline to be executed. This feature can significantly reduce the execution time of a program.

## *polyFORTH Performance*

This section contains a list of several frequently defined FORTH words and their execution time on the NCR/32. All timings include NEXT and are therefore complete, including interpreter overhead. Arithmetic timings are worst case. Timings assume a 150ns cycle time.[3]

| Word(s) | Clocks | Microseconds |
|---|---|---|
| colon entry | 13 | 1.95 |
| DOES> entry | 17 | 2.55 |
| EXIT | 10 | 1.5 |
| IF, WHILE, UNTIL | 12 to 16 | 1.8 to 2.4 |
| ELSE, REPEAT, AGAIN | 10 | 1.5 |
| LOOP | 18 to 20 | 2.7 to 3.0 |
| +LOOP | 22 to 23 | 3.3 to 3.45 |
| CREATE, VARIABLE, etc | 13 | 1.95 |
| LITERAL (explicit or implicit) | 12 | 1.8 |
| USER variable | 17 | 2.55 |
| CONSTANT | 16 | 2.4 |
| 2CONSTANT | 21 | 3.15 |
| >R | 13 | 1.95 |
| R>, I | 12 | 1.8 |
| I' | 13 | 1.95 |
| J | 14 | 2.1 |
| 2>R, 2R>, DO | 17 | 2.55 |
| LEAVE | 12 | 1.8 |
| @ | 10 | 1.5 |
| ! | 15 | 2.25 |
| +! | 18 | 2.7 |
| C@ | 13 | 1.95 |
| C! | 14 to 15 | 2.1 to 2.25 |
| 2@ | 15 | 2.25 |
| 2! | 20 | 3.0 |
| U@[4] | 11 | 1.65 |
| U! | 19 | 2.85 |

[3] All timings and coding examples are courtesy of and copyright by FORTH Inc, and/or Athena Programming

[4] The U@ and U! words are used to fetch and store data in ISU.

| Word(s) | Clocks | Microseconds |
|---|---|---|
| H@[5] | 12 | 1.8 |
| H! | 18 | 2.7 |
| H* | 84 | 10.6 |
| * | 178 | 26.7 |
| U* | 182 | 27.3 |
| M* | 187 | 28.05 |
| H/ | 102 | 15.3 |
| H/MOD | 104 | 15.6 |
| /MOD | 213 | 31.95 |
| MOD | 211 | 31.65 |
| / | 228 | 34.2 |
| U/MOD | 217 | 32.55 |
| M/ | 230 | 34.5 |
| */MOD | 384 | 57.6 |
| */ | 402 | 60.3 |
| DUP | 9 | 1.35 |
| DROP | 10 | 1.5 |
| SWAP, OVER | 12 | 1.8 |
| ROT | 17 | 2.55 |
| ?DUP | 11 | 1.65 |
| 'S | 10 | 1.5 |
| 2DUP | 14 | 2.1 |
| 2DROP | 10 | 1.5 |
| 2OVER | 18 | 2.7 |
| 2SWAP | 22 | 3.3 |
| +, −, NEGATE | 11 | 1.65 |
| 1+, 1−, 2+, 2−, 2*, 2/ | 8 | 1.2 |
| 4*, 4/ | 9 | 1.35 |
| D+ | 19 | 2.85 |
| DNEGATE | 15 | 2.25 |
| M+ | 17 | 2.55 |
| AND, OR, XOR, 0=, 0<, NOT | 11 | 1.65 |
| <, =, >, U<, | 14 | 2.1 |
| WITHIN | 18 | 2.7 |
| ABS | 9 to 11 | 1.35 to 1.65 |
| MIN, MAX | 13 | 1.95 |
| RELEASE | 14 to 15 | 2.10 to 2.25 |

[5] The H@ and H! words are used to fetch and store 16 bit words in MSU.

| Word(s) | Clocks | Microseconds |
|---|---|---|
| HERE | 17 | 2.55 |
| PAD | 19 | 2.85 |
| HOLD | 23 | 3.45 |
| DIGIT | 13 | 1.95 |
| UPDATE | 17 | 2.55 |
| COMPILE | 21 | 3.15 |
| | | |
| MOVE | 11+5n/4 | 1.65+.19n |
| CMOVE | 16+17n/4 | 2.4+.64n |
| FILL, ERASE | 14+21n/4 | 2.1+.79n |
| −TEXT (n chars before no match) | 21+7n/4 | 3.15+.26n |

Why do you need such a powerful FORTH machine? The answer to this question may seem self evident to anyone who has ever run out of computing power. The main reason is that it opens up new fields for FORTH applications. Also, it can extend the range of existing applications. One case in point is an application where this FORTH is used to drive 200 interactive terminals. The average response time when all terminals were active is under 100 milliseconds.

Why does this FORTH run so fast? Aside from the fact that the basic machine has a very fast cycle time, it is specially designed to emulate different computers.[6] Since the core set of FORTH words can be viewed as an instruction set for a "FORTH computer", the CPC is an ideal vehicle for implementing FORTH.

Because of the way the CPC instructions are partitioned, a good programmer can make use of nearly every machine cycle. For example, to fetch a word from memory requires two CPC instructions (a fetch and a receive) that each take one machine cycle. It takes three cycles for the fetched data to be valid on the bus. This third cycle (between the fetch and receive instructions) can be used for any single cycle instruction that does not access the PM Bus. The delayed jump is another instruction that enhances CPC performance. It takes three cycles for a jump instruction to take effect. (This is due to the fact that any instruction must be stepped through the pipeline, which takes three cycles.) An immediate jump bypasses the execution of the other two instructions currently in the pipeline. A delayed jump allows these two instructions to be executed. We will see how this can be used when we look at the NEXT word.

Finally, since the CPC's external microcode is user alterable, the FORTH can be optimized for particular applications. Functions that are run frequently, or are very time sensitive, may be microcoded. This allows the most efficient use of the processing time available to the application.

[6] For example, it has support features to allow emulation of an IBM System/370. For a description of this see [MCB84b].

## *Typical FORTH Words*

This section contains the definition of the FORTH words NEXT, DUP, EXIT, and colon. Only the NEXT will be discussed.

NEXT

Definition

```
LABEL next      I I F)+ 0 RCV   0 Q0 3 2 2 1'S IF: 0 JOR
      0 W F)+ 1 RCV  1 JOR D
LABEL semi      uHERE 2+ 1 Q0 128 JRZ
      4 # R SUB   R ST  4 # S SUB   S ST  UN NEXT' D
         1 I F)+ W T MOV
```

or, to organize it more logically;

```
1    I I F)+ 0 RCV
2    0 Q0 3 2 2 1'S IF: 0 JOR
3    0 W F)+  1 RCV
4    1 JOR D   uHERE 2+  1 Q0 128 JRZ
5    4 # R SUB   R ST  4 # S SUB
6    S ST  UN NEXT' D  1 I F)+ W T MOV
```

Line 1 fetches the next cell from memory and increments the address to point at the next cell in memory. Line 2 determines if this cell references a code definition. If it references a code definition, it jumps directly to the microcode. Line 3 fetches the first word of the code field and sets up the parameter field address. Line 4 determines if the code field refers to a microcode routine. Note that the uHERE corresponds to the HERE word for microcode accesses. If it does, control is transferred to that routine. Line 5 is executed for DOES> definitions. Line 6 continues processing the DOES>.

There are 4 entry points to this routine. This is done to allow microcode routines to fully utilize processor cycles. For example, a routine can initiate a delayed jump to NEXT and execute the first two instructions in NEXT after the delayed jump. In this instance, the entry point to NEXT would be the third instruction.

Executing a NEXT to code takes 7 clocks, or 1.05 microseconds. Executing a NEXT to ;code takes 11 clocks, or 1.65 microseconds. Again, for more information on this, contact FORTH, Inc.

DUP

```
CODE DUP ( n – n n ) I I F)+ 4 # S SUB 0 RCV S ST next
4+ BR D 0 Q0 3 2 1'S IF: 0 JOR
```

EXIT

```
CODE EXIT R R F)+ UN NEXT' D I RCV I I F)+
```

colon

```
LABEL colon W W F)+ 4 # R SUB 0 RCV R ST W I MOV 0 Q0 3
2 1'S IF: 0 JOR 0 W F)+ semi BR D 1 RCV 1 JOR D
```

## *Conclusion*

The microprogrammability of the NCR/32 allows a very compact, high performance FORTH implementation. Because the microcode is user alterable, the system can be easily optimized for different applications, making it an ideal microprocessor for a wide variety of FORTH computers.

Manuscript received August 1984.

## *References*

[MCB84a] McBride, Michael L., "Implementing Forth on the NCR/32", *1984 Rochester Forth Applications Conference*. Institute for Applied Forth Research, Inc., Rochester, NY. 1984.

[MCB84b] McBride, Michael L., "Microprogrammable chipset emulates mainframe processing", *Electronic Design*, Aug. 9, 1984, page 229.