Forth as a language for Digital Signal Processing

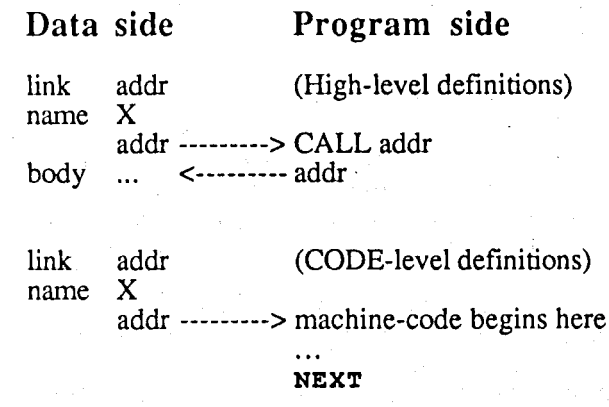
by Martin J. Tracy, FORTH Inc.

The FB320 (Forth board 320), recently developed at FORTH, Inc., supports a digital signal processor (DSP) with the interactive high-level Forth language. The chip used in this first version of the DSP development system is the Texas Instrument TMS32020. It is mounted on a Pacific Microcircuits IBM board, which includes a 50 KHz A/D and D/A converter, up to 128K 16-bit cells of RAM or ROM, and an external bus. The polyFORTH development package runs directly on the TMS32020 chip, using the IBM for terminal and disk support.

Many DSP chips use the *Harvard architecture,* in which the program and data memory spaces are separated. This allows the DSP chip to use *two* data buses for rapid evaluation of polynomials. Typically, the program bus is used for coefficients and the data bus for the numeric terms. The TMS32020 is capable of a 16 x 16 signed multiply and shifted 32-bit accumulation in a total of 200 nsec. This architecture required a Forth implementation which combines ROM-ability with separated headers. Furthermore, to coach maximum speed from the Forth, a direct-threaded coding (DTC) implementation was chosen.

Direct-threaded code

In a DTC Forth, the code field of each definition contains machine code rather than a pointer to machine code. In definitions created by : ("colon") or **CREATE** , this is usually a branch or call to the machine-code responsible for the run-time action of the word. The branch instruction is usually the same size as the call instruction, so the location of the body of the word can be determined by adding a fixed offset to the code field, as it is done in most Forths. **CODE** definitions simply begin with their machine code and so have no body.



There are two related reasons for the increased speed of a DTC Forth over a traditional ITC (indirect-threaded code) Forth. First, there is only one level of indirection from the compilation address to the code field. Second, **NEXT** is shorter, and so can be compiled *in-line* rather than by generating a branch to it. The TMS32020 **NEXT** is only two instructions. In high-level definitions, the code field can be further optimized by tailoring the branch or call instruction to the class of definition. In the FP320 polyFORTH, this means selecting one register for colon and **DOES>** code fields and another for **CREATE** code fields.

The increase in performance is offset by an increase in the code field size, which is one 16-bit *cell* larger than its ITC counterpart. On the other hand, **CODE** definitions are one cell smaller. A DTC nucleus is actually smaller than an ITC nucleus.

The ' ("tick") operator in a DTC Forth points to the code field of a word rather than to its body, since **CODE** definitions have no body. This also allows **EXECUTE** and **@EXECUTE** to operate at maximum speed.

Harvard architecture

In the Harvard architecture, machine code can only run on the program side. Data memory access, on the other hand, is reasonable only on the data side. This split-memory system strongly resembles the CSEG vs DSEG architecture of the Intel 8086 chip. In either case, the link and name fields of a definition should be on the data side, followed by a pointer to the code field on the program side. Furthermore, the code field of a high-level word must be followed by a pointer back to its body on the data side.

Register discipline

The TMS32020 register assignments are as follows:

AR0 is scratch.
AR1 is the Forth **I** register.
AR2 is the Forth **S** register.
AR3 is the Forth **R** register.
AR4 is scratch.

The machine stack of the TMS32020 is only four items deep. PolyFORTH uses only one level of this stack. The data and return stacks are created in software using the **AR2** and **AR3** registers. The return stack is a **post-decrement** stack. This means **R** does not point to the top element on the stack but rather to the next available location. In other words, **R** (alias **AR3**) points to a scratch RAM area.

The polyFORTH DSP development system was written to be relocatable, by using only the indirect addressing mode. The page register is free to be used and changed by the application software at any time. The total address space is 64K cells on each "side," ie both data and program memory. For this reason, stacks are one cell wide. Because the accumulator is two cells wide and is used for arithmetic, the **SX** sign-extension bit is always kept on.

The TMS32020 only allows one of the five auxilary registers to be active at a time. In other words, while using the data stack, the return stack is not available. However the active register can be changed as a "free" operation by any indirect memory access instruction. Otherwise, changing the active register takes a full instruction cycle. The proper register discipline for maximum speed is to select the I (AR1) register before NEXT.

Two Examples

Let us consider the case where we are in the midst of a high-level definition, about to call a variable followed by a colon definition:

Data side

```
          ...
I -> X
          ?
          ...
```

Both X and ? are represented by compilation addresses on the program side of their respective code fields. The I (AR1) register is assumed to be active as we procede through NEXT:

Program side

```
*+ S 00 LAC     ( Increment I and load X into the accumulator)
                ( Also change to S register)
BACC            ( Jump to address in accumulator)
```

We make the assumption that we will be using the data stack next by selecting S (AR2) as we go. In this case, we are right. We are now running in the code field of X on the program side:

```
constant *- CALL    ( Call machine code at constant)
                    ( Also decrement S)
address of X data   ( Compiled in-line address)
```

PolyFORTH is ROMable, which means that variables like X have the same action as constants, ie, to push a literal on the stack. We assume that the correct code field action for any word constructed with CREATE or CONSTANT or VARIABLE is to push something on the stack, so we decrement S accordingly. Such words are invariably followed by the compiled literal to be pushed on the stack, in this case a pointer back to the data field of X on the data memory side. Fortunately, this literal is at the return address which the CALL instruction has obligingly pushed on the machine stack. We are now running at constant on the program side:

```
POP             ( Pop the return address into the accumulator)
* I TBLR        ( Read the literal into the data stack)
                ( Also select the I register)
NEXT            ( The NEXT macro)
```

This completes the action of **X** for a total time of about 2 microseconds. **NEXT** takes us next to the code field of **?** , making the **S** register active, as before. In this case we have guessed wrong:

```
colon * R CALL          ( Call machine code at colon)
                        ( Also change to the R register)
address of ? body       ( Compiled in-line address)
```

The **CALL** to **colon** gives us a change to change our minds and select the **R** register instead. The data stack is not decremented, since we will be pushing nothing on it. As before, **CALL** pushes the address of the body of **?** on the machine stack. We are now running at **colon**:

```
I *- SAR        ( Save I on the return stack)
POP             ( Pop the return address into the accumulator)
* TBLR          ( Read the literal into a scratch area)
I * I LAR       ( Put the literal into I and make it active)
NEXT            ( The NEXT macro)
```

PolyFORTH is now running in the **?** definition. The total time from **NEXT** to **colon** and back again is well under 3 microseconds.

The development environment

The polyFORTH DSP development system includes a built-in full macro assembler with local numeric labels. It includes three source editors: string, function key, and full screen editors. There are two fixed-point math packages: a mixed 32-bit fixed-point (Q31,16) and a faster signed 16-bit fixed-point (Q15) package.

The IBM provides access to all normal IBM facilities: terminal, disk, printer, serial interface and so on. The IBM and DSP polyFORTHs are so well matched that they can share source code. Both polyFORTHs run concurrently, sharing information through mailboxes. A spectrum analyser demo is provided in which the DSP polyFORTH collects microphone data and runs a 256 point FFT in real time, while the IBM polyFORTH displays it on the high-resolution graphics screen. The polyFORTH DSP development system is available now from FORTH, Inc.