FORTH Direct Execution Processors in the Hopkins Ultraviolet Telescope

Ben Ballard

The Johns Hopkins University
Applied Physics Laboratory

Abstract

Engineers at the Johns Hopkins University Applied Physics Laboratory have designed and built two computers which will control and monitor the Hopkins Ultraviolet Telescope, a 1986 Space Shuttle experiment. These computers implement a microcoded FORTH nucleus in a word-addressed AMD 2900 series bit-slice architecture. All programs for them are written in FORTH, which takes the place of assembly language in this architecture.

Background

The Hopkins Ultraviolet Telescope (HUT) is a Space Shuttle experiment designed to obtain the far ultraviolet spectra of astronomical objects ranging from Halley's comet to stars, galaxies and quasars as faint as the 17th magnitude. Accurate real-time computation of the wavelength of each photon sensed by the HUT detector requires greater processing power than is available in any space-qualified single chip microprocessor being produced commercially. In addition, the complex and changeable control and monitoring requirements of HUT dictate the use of a powerful and flexible computer system which can be programmed in a high level language.

Since HUT is a single-purpose embedded computing application, a significant performance advantage may be gained by sacrificing the generality of traditional processors with a processor which executes a single language very efficiently. With this philosophy in mind, the HUT designers chose to implement a computer architecture which executes FORTH exclusively. FORTH was chosen for the following reasons:

—simplicity of implementation
—extensibility and flexibility
—experience from previous projects.

This article describes the FORTH-based computer system which resulted from the development effort to meet HUT's data processing requirements. The first of two prototype processors executed its first FORTH program in April, 1982, and the final flight versions of the hardware and software will be complete in May, 1984.

System Architecture

Figure 1 shows the interfaces which connect the HUT flight computer system to the

This research was supported by NASA under contract NAF5-27000 through the Johns Hopkins University.

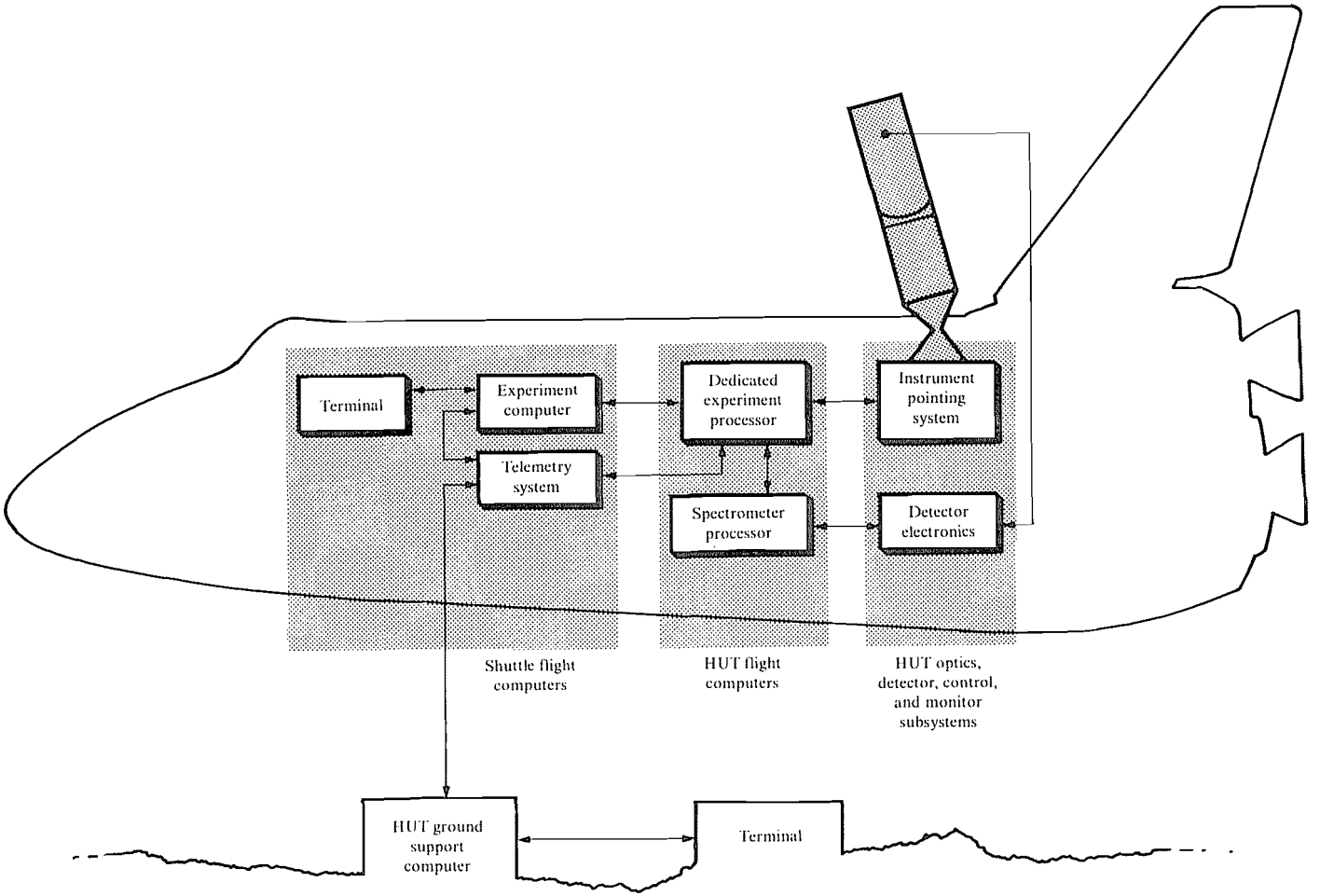


Figure 1. HUT System Interfaces

other components of the telescope and to the Space Shuttle's experiment support equipment. Two independent processors provide the computational power which HUT requires: the Spectrometer Processor (SP), and the Dedicated Experiment Processor (DEP). The SP receives data generated by the telescope's detector, which is a 1024 element self-scanned linear photodiode array (Reticon). As the SP reads the Reticon output, it performs a high-speed centroiding algorithm to compute the wavelength of each ultraviolet photon detected and periodically forwards the accumulated spectral data to the DEP. The DEP receives this data from the SP, handles telescope control and monitor functions, interprets commands and generates displays for HUT's astronaut-operator. In addition, it acquires and analyzes starfield TV pictures to assist in telescope pointing, and transmits the spectral, status and video data to the ground via the Shuttle's telemetry system. Figures 2 and 3 summarize these functions and their performance requirements.

SP Functions	Performance Requirements
Read data from UV detector	1024 6-bit A/D conversions/millisecond
Compute photon wavelengths	Once per photon, up to 9,000 per second
Send UV spectrum to DEP	2067 words every 2 seconds
Interpret 5 command types from DEP	Once per command

Figure 2. HUT Computer Functions: Spectrometer Processor

DEP Functions	Performance Requirements
Receive spectrum from SP	2067 words every 2 seconds
Control 8 motors, 6 power supplies, 14 heaters	Occasional, as commanded
Monitor 60 analog, 56 discrete lines	Once every 2 seconds
Interpret 88 command types	Once per command
Generate one of 4 displays	Once every 2 seconds
Analyze TV picture, compute pointing errors	Once every 20 seconds
Transmit telemetry to ground station	97,656 bits/sec continuous
Control observing sequence	Continuously during observation, average duration 20 minutes

Figure 3. HUT Computer Functions: Dedicated Experiment Processor

Hardware Architecture

The SP and DEP have identical CPU and memory designs, with additional boards to handle unique SP and DEP I/O requirements. Each processor executes approximately 500,000 FORTH primitives per second, and Figure 4 lists some of the hardware features of the SP and DEP. The CPU is a 16-bit microprogrammed machine containing four Am2903 four-bit microprocessor slices, an Am2910 sequencer, an Am2902 carry look ahead unit, an Am2904 status and shift control unit and an Am2925 programmable clock generator from Advanced Micro Devices, Inc. The microprogram consists of 512 64-bit words stored in PROM, and the microinstruction execution time varies from 200 to 400 nanoseconds under microprogram control.

	SP	DEP
Size	3 6″ x 11.5″ boards	8 6″ x 11.5″ boards
Power	~30 watts	~60 watts
CPU	16-bit microprogrammed	16-bit microprogrammed
Clock rate	25 MHz.	25 MHz.
Microcycle rate	2.5–5.0 MHz.	2.5–5.0 MHz.
Instruction rate	500,000 FORTH primitives/sec.	500,000 FORTH primitives/sec.
Microstore PROM	512 64-bit words	512 64-bit words
Hardware signals (analogous to interrupts)	16 available, 4 used	16 available, 10 used
Main memory RAM	20K 16-bit words	48K 16-bit words redundant main memory
		64K 16-bit words video memory
Main memory PROM	4K 16-bit words	4K 16-bit words
I/O access	Memory-mapped	Memory-mapped

Figure 4. HUT Computer Hardware Features

The memory is a 16-bit wide, word addressable design using 16 kilobit NMOS components. The effective speed of the memory is 480 nanoseconds for a write and 400 nanoseconds for a read. Various CPU and memory design features have been incorporated to reduce sensitivity to transient and permanent faults caused by radiation.

Figure 5 is a memory map of the HUT computers' address space. Word addresses 0 through 0FFF hex are available to address memory-mapped I/O devices. 1000 is the initial program execution address at power-up, and words 1000 through 1FFF contain bootstrap PROMs. 2000 through 3FFF are unused, and 4000 through FFFF contain RAM (words 4000 through AFFF are unpopulated on the SP). FFA0 is the address of the initial process descriptor, a four word data structure which supports concurrent programming. FFCF and FFEF are the initial parameter and return stack pointers, respectively, and both stacks grow toward lower memory addresses. After power-up, additional process descriptors and stack

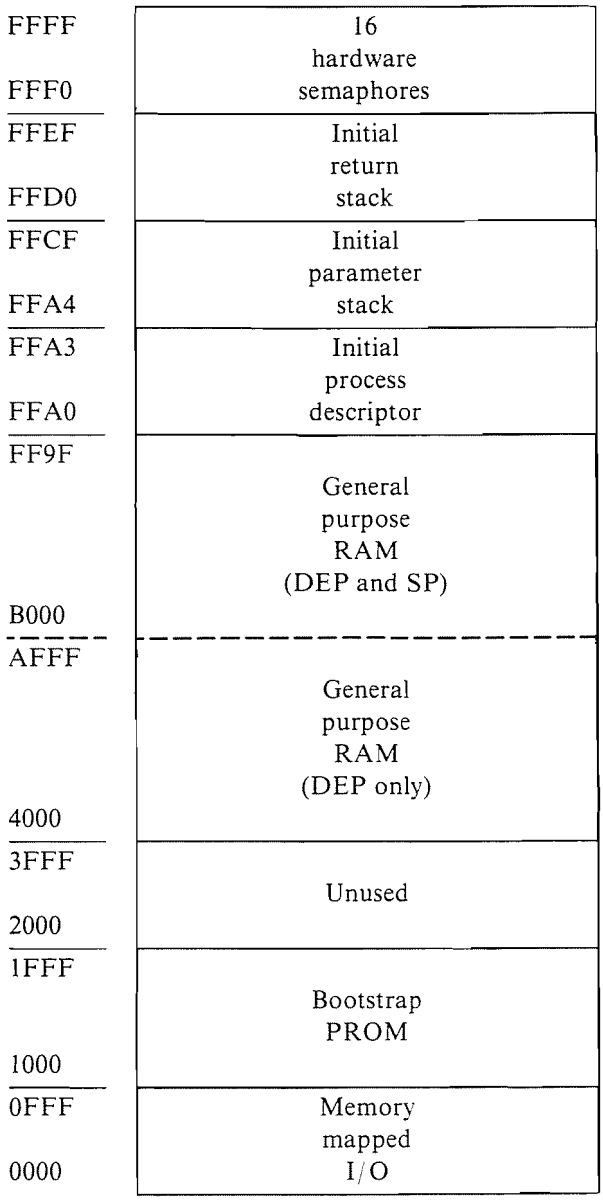


Figure 5. HUT Computers: Memory map

areas may be defined in any area of RAM. Words FFF0 through FFFF are the fixed locations of the 16 available hardware semaphores used to synchronize processing to external events.

Microcode

Slightly over half the microcode space of the SP and DEP is used to implement processor initialization, the macroinstruction fetch-execute loop and 61 FORTH primitives. This microcode is the same in both processors, while the remainder of the microcode space is available for the migration of application software to microcode. This additional space in the SP contains data collection and centroiding routines used in calculating the wavelengths of ultraviolet photons. The DEP uses this space for video processing, character string manipulation and FORTH dictionary search routines. The processor-specific microcode routines are accessible to the programmer as additional FORTH primitives.

Six of the sixteen general purpose registers in the AM2903 are dedicated to maintaining the FORTH machine context, while the rest are available for temporary storage within FORTH primitive routines. The dedicated registers are:

IAR — instruction address register
RSP — return stack pointer
PSP — parameter stack pointer
RUN — pointer to process descriptor of running process
HEAD — pointer to process descriptor of highest priority process
SCHED — indicates whether processor rescheduling is enabled.

The use of these registers is described below.

FORTH Implementation

The version of FORTH implemented on the HUT processors deviates from the FORTH-79 standard in two significant ways. First of all, because the memory is word-addressed, the FORTH words which access byte-oriented data cannot conform to the standard. Instead, the two nonstandard primitives @C and !C address bytes using a base address and a byte offset as input stack parameters. Secondly, since the flight processors do not contain mass storage, the disk access words are not implemented. They are replaced by a set of words which support transfer of ASCII data to and from files on a host system via a serial port. Other small deviations from the standard exist, since language standardization was not one of the original goals of this project.

Figure 6 shows the dictionary entry format used in the SP and DEP. Based on the polyFORTH* format, it consists of a four-word header and an optional variable-length parameter field. The first two header words form the name field and contain the immediate bit, character count and the first three characters of the name. The third word is the dictionary link, which points to the name field of the previous entry in the vocabulary. The last header word is the compilation address, or "opcode" of the FORTH word being defined. The parameter field, if any, contains a string of opcodes and other parameters which make up the word's definition.

For a microcoded primitive, the opcode is the microstore address of the microcode routine which implements it. For non-primitives, the opcode is the main memory address of the first word of the parameter field. As described later, the inner interpreter microcode executes such opcodes directly as machine instructions. Thus every non-primitive FORTH definition actually adds a new opcode to the instruction set of the processor.

*polyForth is a registered trademark of FORTH, Inc., Hermosa Beach, CA.

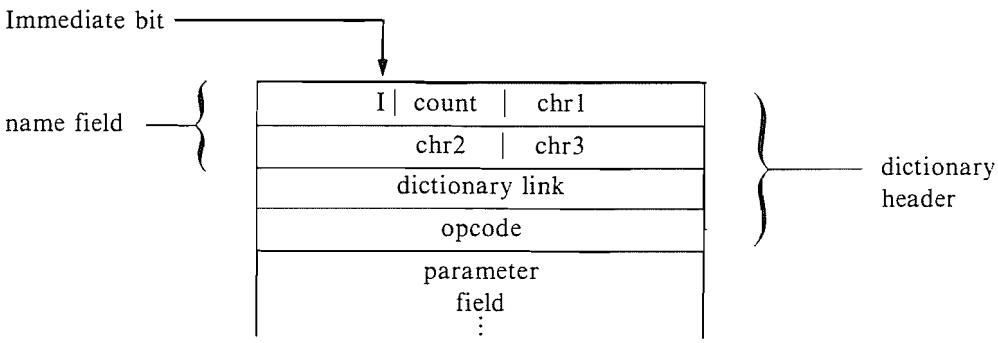


Figure 6. Dictionary Entry Format

As shown in Figure 5, memory addresses 0 through FFF hex are reserved for I/O and therefore are never accessed during instruction fetches. This allows the four most significant bits of an opcode to distinguish primitives from non-primitive FORTH definitions. Opcodes below 1000 hex represent microcoded primitives, while opcodes 1000 and above represent FORTH definitions stored in main memory.

Figure 7 shows four consecutive dictionary entries of various types starting at location 4000 hex. An entry for a primitive, such as DROP , consists only of a header. No parameter field is needed because the definition of the primitive is in the microstore. The parameter field of a colon definition, such as 2DROP or 3DROP , contains the opcodes of the words in its definition, terminated by the opcode of (;) . The parameter field of any other non-primitive contains the opcode of the run-time or DOES> portion of the defining word which created it, followed by any additional parameters needed. For the example ABC in Figure 7, (CONSTANT) is a microcoded primitive which performs the run-time action of the defining word CONSTANT . The structure would work the same way if ABC were created by a defining word written in FORTH. In this case, the first word of ABC's parameter field would contain the address of the beginning of the DOES> portion of the defining word's definition. This address is, in effect, the opcode of the run-time part of the defining word.

The heart of the FORTH machine is the fetch-execute microcode, which implements the inner interpreter as shown in Figure 8. This loop is a direct threaded code version of NEXT which executes only four microinstructions (800 nsec.) in addition to the FORTH instruction fetched from memory.

The inner interpreter microcode first fetches a one-word opcode from the instruction stream and increments the IAR. It then examines the four most significant bits of the opcode to determine whether it is a microcoded primitive or a higher level FORTH word. If these bits are zero, the inner interpreter jumps to the indicated microcode routine, which performs the function of the primitive and then jumps back to the inner interpreter. Otherwise, it simply pushes the IAR onto the return stack and loads the opcode itself into the IAR. Further cycles of the inner interpreter then start executing code sequentially from that address until (;) causes a return to the caller or another non-primitive invokes a further nesting level on the return stack. The return stack push (equivalent to (:) or docolon in other implementations) takes two microinstructions and executes in 480 nsec.

The inner interpreter also services hardware generated signals after every instruction cycle as indicated in Figure 8. This processing is analogous to interrupt servicing in other

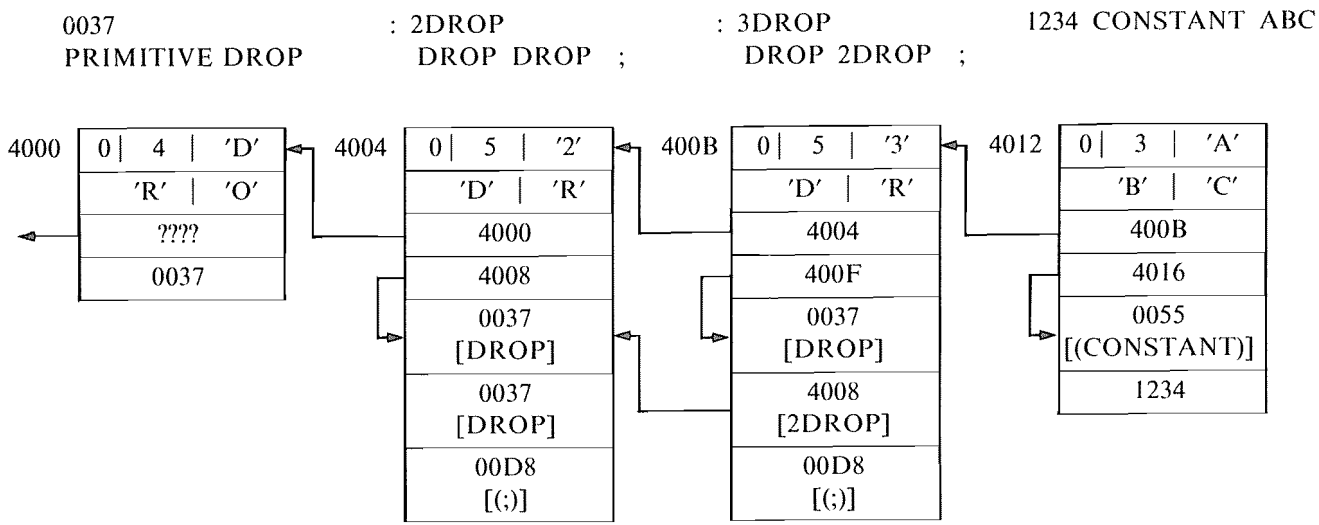
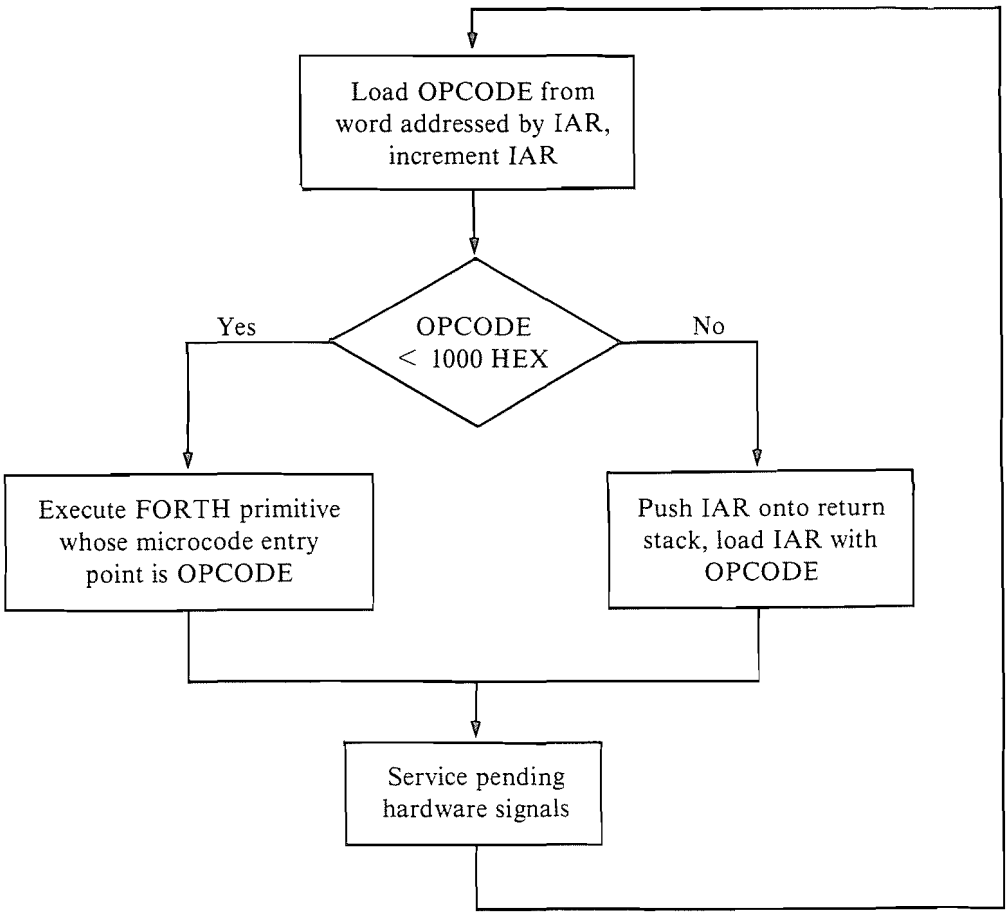


Figure 7. Sample Dictionary Entries



IAR = instruction address register (dedicated CPU register)
RSP = return stack pointer (dedicated CPU register)
OPCODE = temporary CPU register

Figure 8. Microcoded FORTH Inner Interpreter

processors and is described in detail below. It only requires processing time in excess of the 800 nsec. of the inner interpreter loop if any previously unserviced hardware signals are pending.

Figure 9 lists the microcoded primitives. Words which are not part of FORTH-79 are briefly described in the comments. In addition, the words supporting the concurrency features of the HUT computers are all described in detail below.

Stack manipulation:

DUP
DROP
SWAP
OVER
ROT
R>
>R
R@ , I
J

Arithmetic:

+
−
NEGATE
1+
1−
2*
2/
M* (Signed multiply)
U*
+!
OVF (Return arithmetic overflow flag)
CARRY (Return arithmetic carry flag)

Memory reference:

@
!
(MOVE) (Block move to memory)
FMOVE (Block move to I/O channel)

Constants, variables:

0 (Returns constant 0)
1 (Returns constant 1)
−1 (Returns constant −1)
(LITERAL) (FORTH-79 'literal')
(CONSTANT) (FORTH-79 'constant')
(VARIABLE) (FORTH-79 'variable')

Logical:

AND
OR
XOR
ROTATE

Comparison:

>
<
U> (Unsigned >)
U<
=
0<
0= , NOT

Control flow:

(;)
BRANCH (FORTH-79 'else')
0BRANCH (FORTH-79 'if')
(LOOP) (FORTH-79 'loop')
(/LOOP) (Unsigned loop control)
(+LOOP) (FORTH-79 '+loop')
(DO) (FORTH-79 'do')
EXECUTE

Figure 9. Microcoded FORTH Primitives

Figure 9 continued.

CPU register access:

@SP	(Fetch parameter stack pointer)
!SP	(Store to parameter stack pointer)
@RP	(Fetch return stack pointer)
!RP	(Store to return stack pointer)
@RUN	(Fetch RUN register)
@HEAD	(Fetch HEAD register)
!HEAD	(Store to HEAD register)

SP-specific:

CUMUP	(Collect raw UV detector data)
CENTROID	(Compute photon wavelengths)

Concurrency:

DISABLE	(Inhibit processor rescheduling)
ENABLE	(Enable processor rescheduling)
WAIT	(Suspend process until indicated event occurs)
SIGNAL	(Indicate occurrence of software-generated event)

DEP-specific:

@C	(Fetch byte from string)
!C	(Store byte to string)
(FIND)	(Dictionary search)
@V	(Fetch video word)
!V	(Store video word)
@P	(Fetch video pixel)
!P	(Store video pixel)

Concurrency Features

The SP and DEP microcode extends FORTH to support concurrent programming, allowing the definition of any number of independent processes which compete for the processor on a priority basis. Hardware events (analogous to interrupts) and software initiated interprocess signals are represented by counting semaphores. Processes use the WAIT and SIGNAL primitives, respectively, to wait for and indicate the occurrence of events which particular semaphores represent. Source code for a sample process is shown in Figure 10.

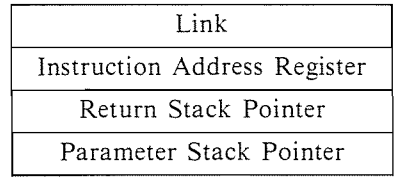
Each process consists of a code body, parameter and return stacks and a process descriptor. The process descriptor is a four word data structure which saves the context of the process when it is not executing. This context consists of the values of the IAR, RSP and PSP at the point of suspension. Loading these saved values back into the processor registers at a later time causes the process to continue from that point.

The first word of the process descriptor is a link field. The process descriptors of all active processes are chained in a circular singly-linked list in priority order as shown in Figure 11. The CPU's HEAD register points to the descriptor of the highest priority process, while the RUN register points to the descriptor of the running process. Position with respect to HEAD in this process list determines process priority, and the descriptor of the lowest priority process is linked back to the HEAD process descriptor.

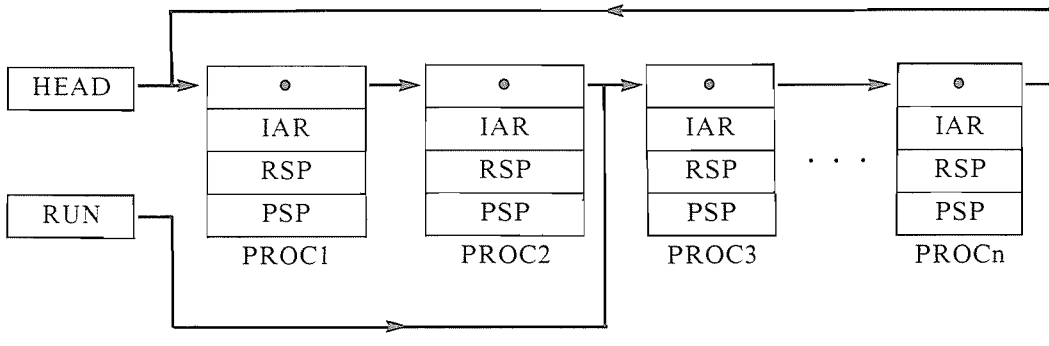
```
FFF6 CONSTANT HARDWARECLOCK        ( CLOCK SEMAPHORE ADDRESS )
VARIABLE TIME                      ( REAL TIME COUNTER       )
VARIABLE 8TICKS                    ( REAL TIME / 8 COUNTER   )

10 20 PROCESS: TIMERPROCESS        ( DEFINE TIMERPROCESS     )
                                   (     RET. STACK: 10 WORDS )
                                   (     PARAM. STACK: 20 WORDS )
        0 HARDWARECLOCK !          ( INIT CLOCK SEMAPHORE    )
        0 TIME !                   ( INIT REAL TIME COUNTER  )
        BEGIN
          8 0
          DO
             HARDWARECLOCK WAIT    ( WAIT FOR CLOCK TICK     )
             1 TIME +!             ( INCREMENT REAL TIME     )
          LOOP
          8TICKS SIGNAL            ( SIGNAL REAL TIME / 8    )
        AGAIN
    ;PROCESS
```

Figure 10. Hardware Clock Service Process



Process Descriptor



Process List

Figure 11. Concurrency Data Structures

A semaphore is a one word VARIABLE anywhere in memory which represents a particular event. SIGNAL increments the semaphore variable, while WAIT decrements it (down to a minimum of zero, at which time the process is suspended). Thus the value of a semaphore is the number of times the event corresponding to the semaphore has occurred without being balanced by a WAIT for the event.

A process uses the WAIT primitive (with the semaphore address as a stack parameter) to suspend itself until an event occurs. Likewise, it uses the SIGNAL primitive to signal the occurrence of an event to another process. Both WAIT and SIGNAL may cause the processor to be reassigned to another process, depending on process priorities and the value of the semaphore involved. Hardware generated events simply cause the SIGNAL operation to be performed for semaphores at fixed locations in upper memory. This processing is performed after each FORTH primitive by the inner interpreter microcode (see Figure 8).

Normally, WAIT , SIGNAL and hardware events cause the processor to be reassigned to the highest priority process not WAITing for an event to occur. The DISABLE and ENABLE primitives may be used to inhibit and resume processor rescheduling for critical program segments. Hardware and software events which occur while rescheduling is DISABLEd are not lost, but are tallied in the corresponding semaphores. Processes WAITing for these events run (priority permitting) as soon as rescheduling is reENABLEd.

Figure 10 is an example of a process called TIMERPROCESS which services a clock wired to signal the hardware semaphore at address FFF6 hex. After performing initialization, it enters an infinite loop which increments the variable TIME each time the hardware clock ticks. In addition, it signals the software semaphore 8TICKS every eighth hardware clock tick. Another process can use 8TICKS (just as TIMERPROCESS uses HARDWARECLOCK) to perform other processing every 8 clock ticks. Notice that the distinction between hardware and software events is determined solely by semaphore addresses and is otherwise transparent to the processes which service them.

PROCESS: is a defining word which creates a dictionary header, builds a data structure containing process initialization information, allocates stacks and a process descriptor, and enters compile mode. Its arguments are the sizes of the return and parameter stacks to be allocated. ;PROCESS ends the definition of a process by exiting compile mode. The execute-time action of a word defined by PROCESS: is to return the address of its process initialization block. This is used by the words which initialize process descriptors, activate processes at particular priorities and deactivate running processes.

These concurrency concepts are used extensively in the HUT applications software. The SP software consists of 7 independent processes, while the DEP software is implemented as 26 processes. The ability to divide the required functions easily into independent processes with simple communication and synchronization interfaces has simplified the software development task tremendously. This simplification began in the initial software design phase and led to an understandable modular design. As a result, it has been easy to incorporate significant changes in requirements at all stages in the development cycle. The simple concurrency scheme extends the modularity of FORTH into multitasking in a natural and consistent way.

Software

The HUT software engineers have developed several support tools in addition to the SP and DEP application software which will control the telescope in flight. These tools include a microassembler, a metacompiler, PROM programming support software and interactive diagnostic and test packages.

The microassembler is a general purpose program based on the work of Greg Cholmondeley (reference 1). It supports user-defined microword formats with shared fields,

microinstruction labeling with forward and backward referencing, conditional assembly and sorted symbol table output. This program is used to assemble and document all of the microcode for the breadboard and flight processors.

The metacomplier allows for target compilation of programs which run in environments different from the original compilation environment. It supports target applications stored at any address in either RAM or PROM, and provides tools for adding defining words and other metacompiler directives in a uniform way. The metacompiler is used to generate and maintain the operating system for each processor as well as the portions of the HUT application software which are stored in PROM.

Both the microassembler and the metacompiler use the PROM programming support tools. With this software, the output of either can be sent directly to a PROM programmer or stored and used later to program PROMs. All of the HUT microcode and software PROMs are programmed in this manner.

The diagnostic and test software is designed to verify the operation of the hardware in each processor. It includes memory, processor and I/O tests for both the DEP and SP. In addition, the SP software includes routines which thoroughly test the interface to the telescope detector. The diagnostic package is useful for initial debug of new interfaces, and will later be part of the formal verification and documentation package for the integrated telescope.

Project Status

The engineering prototypes for both the SP and DEP computers have been running since August, 1982. Fabrication and checkout of flight versions, qualified for the Shuttle orbital environment and designed for efficient heat dissipation in a vacuum, will be completed by April of 1984. Development of all microcode and of the SP application software is complete, and the DEP application software will be completed by May 1984. Integration with the rest of the telescope will begin in May, and the complete instrument will be delivered to Kennedy Spaceflight Center in December 1984 to begin the Shuttle mission integration process. The first of three 10-day Shuttle flights is scheduled for March, 1986, to coincide with the encounter with Halley's comet.

The HUT engineers have started to design a single-board implementation of the architecture described above for use in future applications. The board will contain, as a minimum, the processor, 60K bytes of byte-addressable memory, and a serial port. This machine will offer higher speed and use less power than the HUT processors. In addition, it will allow use of memory IC's of various technologies and execute the recently approved FORTH-83 version of the FORTH language.

Acknowledgements

A number of people contributed to development of the HUT computer system. The principal members of the HUT computer development team included Tom Zaremba, computer system manager and architect; Bob Henshaw, lead hardware designer; and Ben Ballard, lead software designer.

The HUT computer team wishes to thank Professor Arthur F. Davidsen of the Johns Hopkins Physics Department and Mr. Glen H. Fountain of the Johns Hopkins Applied Physics Laboratory for their support on this project. Professor Davidsen is the HUT Principal Investigator, and Mr. Fountain is Program Manager for the design, fabrication and integration of the experiment.

Reference

[1] Gregory E. Cholmondeley, A FORTH Based, Micro-sized Micro Assembler, *FORTH Dimensions* Volume 3, Number 4, November/December, 1981.

Manuscript received April 1984.

Ben Ballard received his BSE in electrical engineering and computer science from Princeton University in 1978. Since then he has been an associate engineer in the computer engineering group at the Johns Hopkins University Applied Physics Laboratory. When not at work, he participates in a variety of activities in music, sports and aviation.