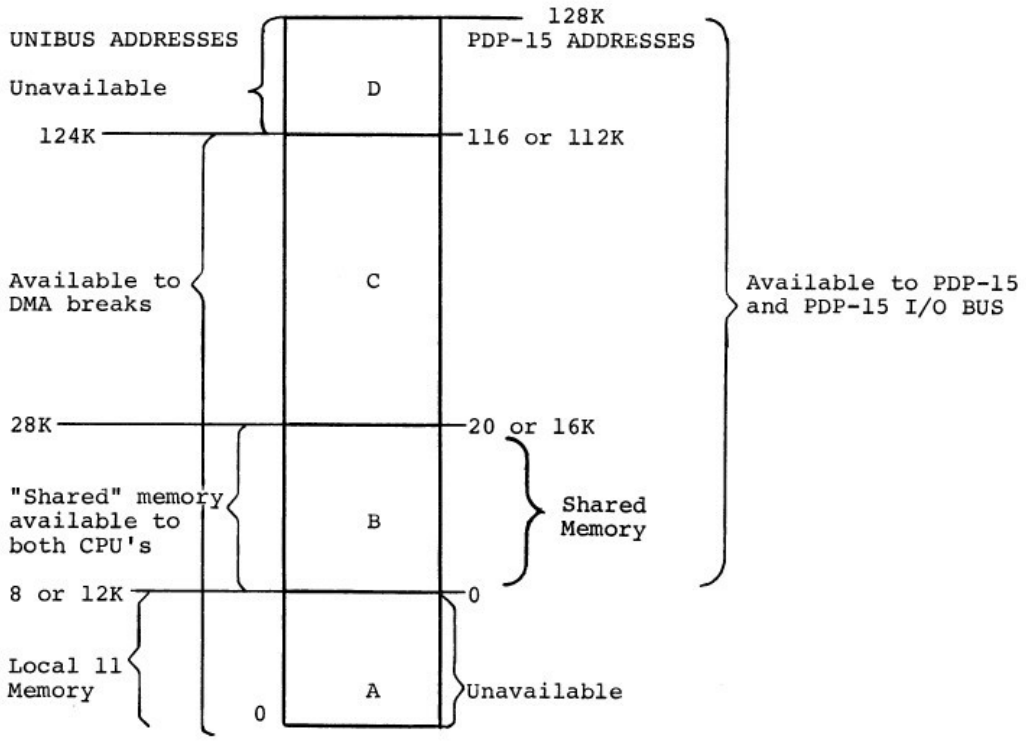
The Design of the Simulated PDP-15/76

The PDP-15/76 was a loosely coupled multiprocessor system. It consisted of a PDP-15 system and an auxiliary PDP-11/05, which was known as the UC15 and acted as an I/O coprocessor. The purpose of the UC15 was to allow the PDP-15 to use inexpensive PDP-11 peripherals, such as the RK11/RK05 disk subsystem and the LP11 line printers, which were produced in much higher volume than native PDP-15 peripherals. The system was introduced in 1972. The UC15 remained an option on PDP-11 systems until they were discontinued in 1977.

The PDP-15 and the PDP-11 communicated via two mechanisms:

1. Shared memory. The PDP-15 had a multiport memory that connected to both the PDP-15 central processor and the PDP-11 Unibus. The Unibus, nominally a 16-bit bus, used its two parity lines as extra data lines to transmit 18-bit data.
2. The systems were cross-connected via parallel links: the DR15 on the PDP-15 side, and a pair of DR11-Cs on the PDP-11 side.

The memory map was rather complicated:



"Local" PDP-11 Memory = A
"Shared" Memory = B
PDP-11 CPU Address Space = A + B
UNIBUS DMA Address Space = A + B + C
PDP-15 Address Space = B + C + D

The PDP-11 has a small amount of private memory, typically 4KW or 8KW. PDP-11 memory addresses above private memory mapped to PDP-15 memory, starting at PDP-15 location 0. PDP-11 memory addresses above 28KW were only accessible to Unibus DMA devices. Thus, the PDP-11 had private memory that the PDP-15 could not access, but unless the PDP-15 implemented its maximum memory of 128KW, all of PDP-15 memory was accessible to the PDP-11 Unibus.

The PDP-11 ran a small program called PIREX. PIREX was a variable priority, fixed number of tasks, real-time executive. It implemented the PDP-11 side of a master-slave protocol that allowed the PDP-15 to use IO devices on the PDP-11. To start an IO operation, the PDP-15 created a Task Control Block (TCB) and wrote its address to the PDP-11 via the DR15 parallel interface. This caused an interrupt to PIREX, which queued the TCB to the appropriate device task. The device task initiated a PDP-11 IO operation and transferred the data to the PDP-15, either via DMA or by writing into shared memory. PIREX signaled completion by causing an interrupt on the PDP-15. The mechanism was both interlocked, with appropriate signals to prevent overrun, and multi-threaded – up to four IO requests could be outstanding. For more details on PIREX and the interprocessor protocol, see *PDP-15/76: Unichannel 15 Overview*.

Alternatives for Simulation

Prior to implementation of the simulated 15/76, two different approaches had been tried for multiprocessors:

1. Loose coupling of two simulator instances via network links (HP Access 2000).
2. Instantiation of multiple processors with a single simulator instance (SiCortex SC1).

Neither was entirely satisfactory. The network model suffered from long latencies in simulating interprocessor transfers, which were nearly instantaneous on real hardware. In addition, it offered no easy way to simulate shared memory. The multiple processor model required rewriting CPUs to be subordinate to a new master controller. In addition, it could not take advantage of the growing number of cores in most PCs and mobile devices.

For heterogeneous multiprocessors, the problems involved in combining systems into a single instance were acute. SimH implements a simple Processor-Memory-Switch (PMS) model. It has no good way to represent multiple instruction dispatch routines, multiple IO busses, and multiple memories. Further, the cut-and-paste construction of most simulators leads to constant global name clashes when two processors are put together. The single-instance model would have required substantial rewrites of both the PDP-15 and PDP-11 simulators, with the likely introduction of new, subtle bugs.

Instead, this design uses two simulator instances linked by a new SimH mechanism, shared memory, which was designed and coded by Mark Pizzolato. This mechanism is used to implement not only the actual shared memory of the PDP-15 but also the state shared between

the DR15 and DR11Cs. As a result, only minimal modifications were needed to the existing PDP-15 and PDP-11 simulators.

Design Details

Shared Memory

The MX15 multiport memory is simulated with a named shared section. On the PDP-15 side, the user first enables the DR15, which is off by default. This causes the simulator to create (or join) a maximum sized (512KB = 128KW of 18b) shared memory section. The existing private PDP-15 memory is deallocated. This works because the PDP-15 simulator accesses memory through a pointer *M rather than as a fixed array.

On the PDP-11 side, the UC15 is a compile-time variant of a standard PDP-11. It includes the two DR11-C parallel interfaces used for interprocessor communication. The reset routine of the DR11's creates or joins the shared memory section. The UC15 compile-time switch also causes the centralized memory read/write and IO read/write routines to be replaced with routines that understand the private/shared structure of memory as seen from the PDP-11.

Access to shared memory is neither interlocked nor synchronized in any way. This works fine on Intel processors, which guarantee both write ordering and (eventual) cache consistency, but it may not work on processors, such as Alpha or ARM, with weaker guarantees.

Interprocessor Communication

Interprocessor communication is also simulated with a named shared section. A common definitions file is used by both the PDP-15 and the UC15 to guarantee agreement on locations within the shared section. Each control variable has a value and a value-changed flag. Each variable is owned by one side or the other; there is never write contention.

When one side wants to send a signal or a value to the other side, it writes the value in shared memory and then uses an interlocked memory operation to set the "signal/value written" flag to 1. The other side is polling frequently, looking for changes in the various "signal/value written" flags. When it sees a change, it validates that the change is real with an interlocked operation and then reads the value.

For example, when the PDP-15 wants to set new Task Control Block pointer (TCBP) to the UC15, it does so with IOT 6001 (LIOR). The DR15 simulators picks up the TCBP from the AC, stores it in shared section offset UC15_TCBP with a normal write, and then sets control variable UC15_TCBP_WR with an atomic Compare and Swap operation. The atomic operation guarantees memory consistency at that point.

The UC15 is polling for changes in control variable UC15_TCBP_WR. If it sees a change, it verifies that the change is real and also resets the control variable to 0 with another atomic Compare and Swap operation. It can then pick up the new TCBP value from UC15_TCBP.

Conversely, when the UC15 wants to signal an API interrupt, it will write a value and set the matching control variable. The PDP-15 is polling for changes in the control variable and picks up the changed value in an analagous way.

The master/slave protocol, and the invocation of atomic operations and thus memory barriers by I/O operations, means that memory is consistent at key points in the protocol.

1. PDP-15 writes a task control block (TCB) in shared memory. For writes, it also writes a buffer in shared memory.
2. PDP-15 issues an LIOR IOT. Atomic operation/memory barrier issued from the PDP-15.
3. UC15 acknowledges task control block write. Atomic operation/memory barrier from the UC15. TCBP, TCB, and I/O buffer will be visible to the UC15.
4. When PIREX reads the TCBP register, the UC15 sends a signal to the PDP15. Atomic operation/memory barrier.
5. PDP-15 sees that the TCBP has been read and can now issue another command, up to a limit of four I/O operations in progress.
6. UC15 processes I/O request. Data is transferred, either via a Unibus DMA device or by program transfer, to or from shared memory.
7. At I/O completion, UC15 signals an interrupt to the PDP-15. Atomic operation/memory barrier from the UC15.
8. The PDP-15 recognizes the interupt request. Atomic operation/memory barrier from the PDP-15.
9. The PDP-15 can now process the buffer (on a read) or reuse the buffer (on a write).

The paired memory barriers are overkill; in theory, one barrier should force a system-wide synchronization. Still, it doesn't hurt to be careful.

Problems with the Model

There are a couple of problems with the model. First, the PDP-15 doesn't always use a protocol involving I/O operations to transfer data. The cross-loader, for example, relies on a non-interlocked semaphore. The PDP-15 writes a data word to a shared location and sets another shared location as a flag. The UC15 monitors the flag for a non-zero value, picks up the data word, zeroes the flag, and waits for the next word. Meanwhile, the PDP-15 monitors to see that the flag is zero. At that point, it is free to transfer another word. This works fine on real hardware, and probably works okay on systems with strict write ordering and guaranteed eventual cache consistency, like the x86. However, it probably won't work on systems with weak write ordering and consistency, like Alpha and ARM.

Second, both the PDP-15 and the UC15 must poll fairly frequently in order to provide timely operation. This means neither simulator can idle, and the simulator processes will run at 100%

CPU utilization. The 15/76 simulator can only really be run on a desktop system or a laptop with wall power and good cooling.

Third, the simulator requires two closely coupled processors – either a dual processor SMP system or (these days) two cores in a multi-core processor. If the two simulators try to share a single core, the result will be erratic performance, or worse.

Further Work

At the moment, the shared memory facility is only available on Windows and Linux. It needs to be extended to other supported hosts, notably VMS.

Ideally, the shared memory capability would be augmented by an interprocess interrupt capability. This would allow polling for state changes to be replaced by a signal. This in turn facilitate implementation of idling. When idle, the simulator would wait for a timer to expire or a signal to arrive from the other simulated processor. PIREX on the UC15 has a well-defined idle state, using the PDP-11's WAIT instruction. The idle state on the PDP-15 side is likely to be trickier to detect and also to be operating system dependent.