Mysteries of MMR1, or, Always Sweat the Small Stuff

Bob Supnik, 23-Mar-2023 (revised 15-Apr-2023)

About seven years ago, a SimH user wrote to me and complained that the PDP11 simulator was incorrectly setting the MMR1 register. On a J11, he wrote, MMR1 did not track PC changes. Further, MMR1 did not track general register changes in floating point instructions at all, and changes to the general registers were undone on an abort. I couldn't see that it made any of this difference, but I dutifully rewrote the PDP11 simulator's specifier flows not to update MMR1 on PC changes. I also rewrote the floating-point routines not to update MMR1 or the general registers in case of an abort – a much bigger PITA, to be honest. And there matters rested.

Then, this year, *another* user wrote to complain that in 11/70 mode, the simulator was incorrectly setting the MMR1 register. On an 11/70, he wrote, MMR1 *did* track PC changes. Further, MMR1 did track general register changes in floating point instructions. This mattered because 2.11 BSD – the last and most ambitious Berkeley Unix on a PDP11 – expected MMR1 to be set on floating point memory management aborts. It had a J11-specific path to account for the J11's "incorrect" behavior, but in 11/70 mode, the simulator was acting like a J11, and 2.11 BSD fault recovery was failing.

MMR1 was part of the PDP11's memory management mechanism, introduced in 1972 with the 11/40 and 11/45. In the 11/45, it recorded changes to the general registers caused by autoincrement and autodecrement operations:

```
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|    amount    |register|    amount    |register|
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

The amount was 2's complement and was 1, 2, 4, 8, -1, -2, -4, or -8. A second register, MMR2, recorded the PC at the start of the instruction. In the 11/40, which implemented only a subset of the 11/45's memory management features, MMR1 always read as zero. Subsequent systems followed either the 11/45's "full" memory management model or the 11/40's "subset" memory management model.

Subset memory management – 11/40, F11, 11/34(a), 11/60
Full memory management – 11/45, 11/70, 11/44, J11

However, as the issue with 2.11 BSD illustrated, this simple classification did not cover all the variations of MMR1 behavior. The PDP11 Handbooks were no help; no MMR1-related issues appeared in the various "tables of incompatibilities." Ultimately, three questions had to be answered:

1. Did subset memory management systems mechanize MMR1 at all, as a read-zero register?
2. Did full memory management systems record autoincrement/autodecrement changes to PC in MMR1?
3. Did full memory management systems record autoincrement/autodecrement changes for floating point instructions?

Mystery #1 – MMR1 in Subset Memory Management Systems

The 11/40, 11/44, and F11 documentation said explicitly that MMR1 was implemented as a read-zero register. The remaining two systems were tested as follows:

- 11/34(a). A real 11/34a was tested. MMR1 exists and reads as zero.
- 11/60. The 11/60 memory management diagnostic tests that MMR1 exists and returns a zero when read.

Thus, all subset memory management systems implement MMR1 as a read-zero register.

Mystery #2 – MMR1 and PC Autoincrement/Autodecrement

The 11/70 (and its older brother, the 11/45) record PC changes. This is explicitly tested in the 11/70 diagnostics.

The 11/44 records PC changes. This is mentioned in the 11/44 technical documentation.

The J11 does not record the "common" PC changes: #literal and @#address. In these cases, the next word in the instruction stream is fetched via the prefetch mechanism, and MMR1 is not invoked. However, the J11 *does* record PC changes for the "uncommon" cases:

-(PC)
@-PC
(PC)+ for a writeable destination operand

In all of these cases, the normal flow of instruction prefetching is disturbed, and the microcode must perform the operations as though the PC were just another register.

In practice, no operating system can depend on MMR1 recording PC changes. The operating system restores the PC from MMR2. It actually has to filter out any PC changes recorded in MMR1 and discard them.

Simulation of precise behavior was implemented in SimH 3-12.3.

Mystery #3 – MMR1 and Floating Point Instruction Autoincrement/Autodecrement

The 11/70 (and its older brother, the 11/45) record autoincrement/autodecrement changes in floating point instructions in MMR1. 2.11 BSD depends on this.

The 11/44 records autoincrement/autodecrement changes in floating point instructions in MMR1. 2.11 BSD depends on this.

The J11 does not record autoincrement/autodecrement changes in floating point instructions in MMR1. Changes to the general registers are backed out on a memory management abort. 2.11 BSD special cases the J11.

The difference between the 11/70 and J11 stems from the constant generation logic that drives MMR1. The 11/70 can generate 1, 2, 4, or 8. The J11 can only generate 1 or 2.

Simulation of precise behavior is required for the correct operation of 2.11 BSD. This was corrected in SimH 3.12-3.

Conclusion

Once again, small details that seem irrelevant to a simulator writer prove to matter a great deal to an operating system that wants to take advantage of them. J11 didn't implement MMR1 tracking in floating point because neither of DEC's "big" operating systems – RSX11M+ or RSTS/E – cared. 2.11 BSD did and had to special case the J11. SimH's original implementation of floating point – in which MMR1 tracked floating-point operations – was right for the 11/70 but wrong for the J11. The next implementation was right for the J11 but wrong for the 11/70. Now, it's right for both. The moral is that in simulation, you have to sweat the small stuff. It actually matters.