DRAFT: for approval

Identification

Programmed Store A Conditional

V.A.Vyssotsky, 1/26/66

Purpose

The proposed new instruction stac (store A conditional) will not be implemented on initial delivery of the first 645s. To cover this gap, the operation code 007 (mme4) will be reserved to serve the same function, and a fault-handling routine will be programmed, as described below to simulate the stac instruction.

Discussion

The instruction stac Y functions as follows. If $C(Y)\neq 0$, set the zero indicator OFF. If $C(Y)=0$, set the zero indicator ON, and then replace C(Y) by C(A). The instruction uses a read-alter-rewrite cycle, and is not privileged. The chief task in simulating the instruction is simulation of a read-alter-rewrite cycle; no interrupts or process faults may occur during the simulation, except as described below. To avoid undesired interrupts, interrupts must be inhibited, so the simulation routine must run in master mode. To avoid undesired faults, partial interpretation of an appending cycle must be done.

In order to prevent conflict between two or more CPUs simultaneously simulating a stac instruction with the same operand address, the simulation routine employs an interlock to delay all but one of the competng CPUs at the beginning of the simulation procedure, and to sequence the CPUs through the procedure one at a time.

Specification

The data and tables required by the procedure include:

a) A single word, cpulock, shared by all invocations of the procedure.

b) A 'transparent' segment, coreaddress, consisting of a segment descriptor and page table such that each absolute address in core corresponds to the word in segment coreaddress whose word number is the absolute address.

Items a) and b), and the procedure described below, must be resident in core at all times

A procedure-like description of the stac simulator is as follows.

1. Save the contents of bases, registers, control unit and descriptor base register in the concealed stack. Enter master mode, and inhibit interrupts.

2. Set the lock cpulock, and delay until it is set, using the sequence

```
          aos cpulock
          tnz *-1
```

3. Clear associative memory.
4. if the saved control unit mode is absolute or temporary absolute, go to step 19.
5. Retrieve the effective segment number, esegno, and the effective word number, ewordno, from the saved control unit status.
6. Retrieve the bounds field and the block size bit from the saved descriptor base register contents.
7. Perform the bounds test for the descriptor segment, using esegno. If a bounds violation is found, go to step 30. Else,
8. Retrieve the paged/unpaged bit from the saved descriptor base register contents. If it is unpaged, go to step 10. Else,
9. Retrieve the appropriate page table entry, accessing it through segment coreaddress. If the class is directed fault, or if the content of the class field is invalid, go to step 30. Else,
10. Compute the address of the segment descriptor, and retrieve the descriptor, accessing it through segment coreaddress.

11. Check the class of the data segment. If the class is directed fault, or if the content of the class field is invalid, go to step 30. Else,

12. Perform the bounds test for the data segment, using ewordno. If a bounds violation is found, go to step 30. Else,

13) If the data segment is unpaged, manufacture a dummy page table entry descriptor field of class master procedure, slave access permitted, write permitted, and then go to step 15. Else,

14. Retrieve the appropriate page table entry, and check the class. If the class is directed fault, or if the content of the class field is invalid, go to step 30. Else,

15. If the mode of the stac being simulated was master, go to step 19. Else,

16. If the logical nd of the descriptor master access bit, the descriptor write permit bit, the page table entry master access bit and the page table entry write permit bit is zero, go to step 40. Else,

17. If esegno equals the segment number in the saved procedure base register, go to step 19. Else,

18. If the effective class of the operand (see G.A.Oliver memo of 7/9/65 for definition of effective class) is master procedure or execute-only procedure, go to step 40. Else,

19. Fetch the operand, C(Y). If C(Y)=/0, set the zero indicator in the saved indicator register OFF, and go to step 21. Else,

20. Set the zero indicator in the saved indicator register ON, and replace C(Y) by the contents of the saved A register.

21. Reset the lock cpulock, using the sequence

```
lda -1,dl
sta cpulock
```

22. Restore bases, registers and control unit. This step has completed the simulation of stac.

there are no steps numbered 23-29.

30. Load esegno into base bb, and ewordno into base bp.

31. Reset the lock cpulock, using the sequence

```
lda -1,dl
sta cpulock
```

32. do the instruction

```
lda bp!0
```

A fault will result. When (if) control is restored to this point, go to step 2.

there are no steps numbered 33-39.

40. Change the operation code of the saved current instruction to sta. Reset the lock cpulock, by the same sequence as shown above. Restore bases, register and control unit, thus restoring control to the user program. The user program will immediately fault.

There are no step numbers above 40.