ELEC 464 : MICROCOMPUTER SYSTEM DESIGN
FALL 1995

Assignment 6 - Exceptions

Due October 24, 1995

Question 1

Question 21 from Chapter 6 of the text. Implement your design as a combinational circuit using a PEEL 18CV8 PAL. Include the CUPL `.pld`, `.doc`, `.si` and `.so` files.

The inputs to the device will be FC0, to FC2, D7 to D6, AS*, and the DTACK* and BERR* that normally go to the 68000 CPU. The device's outputs will be a modified DTACK* and BERR* to the CPU.

In the case of an invalid interrupt number the device should prevent DTACK* from being asserted and assert BERR* instead.

The test vectors should include a test sequence emulating a user data read cycle that reads the value 0 and two interrupt acknowledge cycles, one for an interrupt number of 0 and another for an interrupt number of 64.

Question 2

Assume you are part of a team designing a new version of the 68000 CPU that will feature a new 32-bit bit-reversal instruction. The new instruction has an opcode of $A095 and is followed by a 32-bit pointer to the memory location containing the 32-bit value to be bit-reversed (i.e. the whole instruction is three 16-bit words long). The result is stored back in the same memory location. This instruction will not modify any registers or status bits.

You are to write an exception handler that will emulate the new instruction so that the software that makes use of this new instruction can be tested before the new CPU chips are ready. Your emulator should ignore all other opcodes (those that are not $A095).

You may assume that when your emulator runs the supervisor stack will have 32 bytes available for use by your routine.

Follow the submission instructions given for Assignment 1 with the following changes:

- The file name should be ASG5.X68.
- The entry point label should be REVEMU.
- As before, your routine must start at location $400 but of course it must now end with an RTE instruction.

The marking code will set up the supervisor stack frame and invoke your exception handler. If your emulator returns properly the marking code will test to make sure the correct memory location was bit-reversed and that no registers were modified.

Hints: (1) The $d(A_n)$ addressing mode (see hints for question 3) is an effective way to retrieve values at known offsets in a stack frame. (2) Your exception handler will have to retrieve the opcode and the pointer to the operand from program memory.

Question 3

This question asks you to analyze the behaviour of an interrupt-driven 68000 system.

The following instructions are executed immediately after the system is reset:

```
        ORG     $400            ; location
```

```
        MOVE.L  #$500,$020
        MOVE.L  #$500,$07C
        MOVE.L  #$600,$080
        MOVE.L  #$1000,A7
        MOVE.W  #$0704,SR
        MOVE.L  #$2000,A7
LOOP:   BRA     LOOP
```

All three interrupt request lines are tied together and connected to a 1 Hz clock signal which is initially high and then starts generating clock pulses only after the reset code has been executed. This system's hardware generates VPA* in response to any interrupt acknowledge cycle. The system has an 8-bit parallel output port at address $8004.

The following instructions appear in memory at the addresses indicated by the ORG assembler directives:

```
        ORG     $500
        MOVE.W  0(A7),D1
        MOVE.B  D1,$8004
        RTE

        ORG     $600
        MOVE.W  2(A7),D0
        MOVE.B  D0,$8004
        RTE
```

1. Which interrupt vectors are initialized by the start-up code? (Be precise.)
2. Does the start-up code leave the CPU in the supervisor state?
3. What is the value of SSP after the start-up code is executed?
4. What is the value of the interrupt mask after the start-up code is executed?
5. Will the clock signal cause an interrupt? If so, will the interrupt happen only once or once per second? If once per second, will it happen on the rising or falling edge of the clock?
6. Which interrupt handlers (if any) will be executed?
7. What value(s) will be written to the parallel output port?

Hints: (1) Register A7 is also the stack pointer(s) for 68000 CPUs. (2) The addressing mode $d(A_n)$ yields an effective address equal to the contents of register A_n plus the displacement d in bytes. (3) The total length of the start-up code is less than 2^{16} bytes. (4) You may not need to use all of these hints.