Hi there, kiddies! I bet you'll just be thrilled right out of your pants to know that we're going to give you another assignment for a change. This one is just a little-bitty one, but then again, that's what we say every time.

We're going to let you do a little microprogramming, joy of joys. However, this assignment is only a backburner and isn't due until the end of the semester. So any time you want to, you can do it. Now ain't that swell of us (HA!)?

2 WHAT YOU GOTTA DO?

You gotta microprogram, like we said. And what you gotta microprogram is a routine to emulate the BUGS META 4A SRE (Scan Right Equal) instruction. This instruction scans through an arbitrary length string of bytes for a specified character and sets a register to point at it. If the string does not contain the character, the register is set to zero.

You can assume the following things are in micro-registers when your routine is executed:

- **Register E**: this register contains the byte address of the first character in the string. Remember that the META 4 host is halfword addressable, so you'll have to screw around a bit with the address. Each halfword contains two bytes, of course (a left byte and a right byte).

- **Register C**: this micro-register will contain the length of the string, in bytes. The length will not be zero.

- **Register A**: this register will contain the character to be scanned for in bits 8 - 15. Bits 0 - 7 will contain zeroes.

You should scan the string from left to right for a byte equal to the one in register A and set micro-register P to the byte address of the first occurrence of said byte. If there is no occurrence, register P should be set to zero. Do not assume that it contains zero initially.

Wherever in your program you want to stop execution, simply code a micro-instruction with an cp code of "STOP".

Remember that micro-registers 0, 1, and 2 serve special purposes. Also, you should assume that the Memory address/Memory Data registers are 4 and 5, respectively.

The microprogram shouldn't be any longer than 15 or 20
instructions.

3 HOWS ABOUT SOME JCL?

The following JCL should be used to assemble your microprogram. The job must be run in OSBATCHA. On each assembly you will get a deck, which is, of course, the microprogram text. When you are ready to try simulating your routine, you should see Aggie and he will help you. He will be available whenever he has grading hours and just about any other time, too, so what the hell.

```
// job card...
// EXEC PGM=MEPA4ASM
//STEP1 DD DSN=SYS1.P205506.U000,CLRTXT,UNIT=2314,
// VOLUME=SER=FRESS1,DISP=OLD
//SYSUT DD UNIT=SYSDA,SPACE=(7225,15,15)
//SYSPRINT DD SYSOUT=F,DCB=BLKSIZE=133
//SYSPUNCH DD SYSOUT=E,DCB=BLKSIZE=80
//SYSIN DD *

...(your microprogram)
```

4 EXTRA GOODIES

If you want to do some extra stuff, you can try emulating a S/360 or a S/370, or better yet, you can design your own computer and maybe IBM will give you a contract and pay you infinite bucks. Then again, maybe they won't.
* THIS IS A MICROPROGRAM FOR THE DIGITAL SCIENTIFIC META4 COMPUTER
* AUTHOR--C.J. MATHIAS, AM101
*
* THE PURPOSE OF THIS PROGRAM IS TO EMUlate THE SRE (SCAN RIGHT EQUAL)
* INSTRUCTION CURRENTLY IMPLEMENTED ON THE META4 AT BROWN. THE PROGRAM,
* WHEN PASSED CONTROL, ASSUMES THE FOLLOWING:
*
* R8--CONTAINS THE BYTE ADDRESS OF THE
* START OF THE STRING TO BE SCANNED.
* IT MUST BE NOTED THAT THE META4 IS
* HALFWORD ADDRESSABLE ONLY.
*
* R9--CONTAINS THE LENGTH OF THE STRING,
* IN BYTES.
*
* RAO--CONTAINS THE CHARACTER TO BE
* SCANNED FOR IN BYTES 0-15 (THE
* RIGHT BYTE). THE LEFT BYTE
* IS ASSUMED TO BE ALL ZEROS.
*
* THE MICROPROGRAM SCANS THE STRING FOR THE PRESENCE OF THE
* CHARACTER IS R8. IF THE CHARACTER IS FOUND IN THE SPECIFIED
* STRING, THE BYTE ADDRESS OF THE FIRST OCCURRENCE IS LOADED
* INTO R8(15). IF THE BYTE IS NOT FOUND, R15 IS SET TO ZERO.
*
* REGISTER ALLOCATION AND ECU'S:
*
27 ZR ECU  0$  R0 ALWAYS CONTAINS ZERO
28 SR ECU  1$  BIT 2 OF R1 IS THE SHIFT BIT
29 LK ECU  2$  R2 IS THE LINK REGISTER
30 GF ECU  3$  R3 IS USED FOR GENERAL CALCULATIONS
31 MA ECU  4$  R4 IS THE MEMORY ADDRESS REGISTER
32 MD ECU  5$  R5 IS THE MEMORY DATA REGISTER
33 FF ECU  6$  R8 CONTAINS THE ADDRESS OF THE FIRST BYTE
35 LN ECU  9$  R9 CONTAINS THE LENGTH OF THE STRING
36 CH ECU AF  R10 CONTAINS THE CHARACTER TO BE SCANNED
*   FOR, AS NOTED ABOVE.
38 AN ECU  F$  R15 IS THE ANSWER REGISTER
39 *
40 THE ROUTINE BEGINS BY PUTTING THE BYTE TO BE TESTED INTO
41 BOTH HALVES OF R10. THE ANSWER REGISTER IS THEN LOADED
42 WITH THE BYTE ADDRESS IN R8, WHICH IS THE START ADDRESS
43 FOR THE STRING BEING SCANNED. THE BYTE ADDRESS IS THEN
44 CONVERTED TO HALFWORDS BY SHIFTING IT RIGHT ONE PLACE
45 AND SAVING THE SHIFT BIT. THIS BIT CAN THEN BE USED TO
46 DETERMINE IF THE ORIGINAL ADDRESS WAS ON A HALFWORD
47 BOUNDARY TO START WITH. IF NOT, A BRANCH IS MADE TO THE
48 PART OF THE ROUTINE WHICH MAKES THE RIGHT BYTE OF THE
49 HALFWORD IN QUESTION.
50 *
51 ADD CH GE ZR  L8 MOVE CH BYTE TO GP AND SHIFT
52 ADD GF CH CF  EUT CH BYTE INTO LEFT BYTE OF CH REG
53 ADD FE AN ZB  ICAL ANSWER REG WITH FIRST BYTE ADDRESS
54 ADD FB MA ZR  R1, SC, MR SHIFT CONVERTS BYTES TO HALFWORDS
55 *  SAVE SHIFT BIT FOR ALIGNMENT TEST
* NEXT, THE LINK IS LOADED WITH THE ADDRESS OF THE TCF OF THE
* LOCPF. AN EXCLUSIVE-CF IS USED TO TEST THE HALFWORD READ
* AGAINST THE CH REGISTER. A CONDITIONAL BRANCH IS MADE TO
* "RITE" IF THE SHIFT BIT INDICATES THAT ONLY THE RIGHT BYTE
* OF THE HALFWORD SHOULD BE TESTED, AS THE LEFT BYTE IS NOT PART
* OF THE ORIGINAL STRING.

    ADDI ZR, IK          NEXT LOAD LINK WITH ADDR OF TCF CF LOCPF
    XCR MD, CF, CH      PZ XCR AFTER PAUSE FCF DATA IN MD
    BNEZ SR             RITE 2$ TEST BIT 2 OF CF R1 FOR SHIFT

    "NEXT" IS THE TCF OF THE LOCPF WHICH TESTS HALFWORDS. BOTH
    IT AND "RITE" MAKE CONDITIONAL BRANCHES IF THEY FIND THE
    CHARACTER CF IF THEY DO NOT AND THE COUNTER RUNS OUT.

    NEXT ERZ GP          CK 1 TEST LEFT BYTE FOR CHARACTER FOUND
    ADDI IN, IN          FFPS$ INCREMENT LEN IN COUNTER REG
    ERZ IN               DNF W IF COUNTER=-C, THEN BRANCH TO DNF
    ADDI AN, AN          1$ INCREMENT ANSWER REG

    "RITE" TESTS THE RIGHT BYTE OF THE REGISTER IN THE SAME MANNER
    AS DCESS "NEXT".

    RITE ERZ GP          CK R TEST RIGHT BYTE
    ADDI IN, IN          FFPS$ INCREMENT COUNTER
    ERZ IN               DNF W IF COUNTER=-C, BRANCH TO DNF
    ADDI AN, AN          1$ INCREMENT ANSWER REG

    THE FE REGISTER IS INCREMENTED AND STORED BACK INTO THE MA
    REGISTER FCF THE NEXT MEMORY READ. THE SHIFT CONVERTS BYTES
    TO HALFWORDS.

    ADDI FB, FB          2$ INCREMENT FE REGISTER
    ALL FB, MA          ZE IF PUT FE INTO MA, SHIFT FCF CONVERSION,

    LASTLY, AN XCR IS PERFORMED ON THE NEW HALFWORD
    AND A JUMP IS TAKEN TO THE TCF OF THE LOCPF.

    XCR MD, CF, CH      PZ, J EXCLUSIVE-CF AND JUMP TO TCF OF LOCPF

    "DNF" (DID NOT FIND) LOADS THE ANSWER REG WITH A ZERO.

    DNF ADD ZR, AN, Z6       ZERO R15 TO INDICATE DNF CONDITION

    "OK" STORES THE PROGRAM, WITH THE ADDRESS IN BYTES OF THE
    FOUND CHARACTER REMAINING UNTOUCHED IN R15.

    STOP*                 ILLEGAL CFCCODE FOR TERMINATION

    * AND THAT'S ALL THERE IS TO IT----------------->
<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>VALUE</th>
<th>DEFN</th>
<th>REFERENCES</th>
</tr>
</thead>
<tbody>
<tr>
<td>AN</td>
<td>000F</td>
<td>38</td>
<td>53 75 75 83 83 100</td>
</tr>
<tr>
<td>CH</td>
<td>000A</td>
<td>36</td>
<td>51 52 52 65 96</td>
</tr>
<tr>
<td>DNF</td>
<td>0024</td>
<td>100</td>
<td>74 82</td>
</tr>
<tr>
<td>FE</td>
<td>00C8</td>
<td>34</td>
<td>53 54 69 69 90</td>
</tr>
<tr>
<td>GF</td>
<td>0003</td>
<td>31</td>
<td>51 52 65 72 80 96</td>
</tr>
<tr>
<td>IK</td>
<td>0002</td>
<td>30</td>
<td>64</td>
</tr>
<tr>
<td>LN</td>
<td>0009</td>
<td>35</td>
<td>73 73 74 81 81 82</td>
</tr>
<tr>
<td>MA</td>
<td>0004</td>
<td>32</td>
<td>54 50</td>
</tr>
<tr>
<td>ME</td>
<td>0005</td>
<td>33</td>
<td>65 96</td>
</tr>
<tr>
<td>NEXT</td>
<td>000E</td>
<td>72</td>
<td>64</td>
</tr>
<tr>
<td>CK</td>
<td><strong>UNDEFINED</strong></td>
<td>72</td>
<td>80</td>
</tr>
<tr>
<td>HITE</td>
<td>0016</td>
<td>80</td>
<td>66</td>
</tr>
<tr>
<td>SR</td>
<td>0001</td>
<td>29</td>
<td>66</td>
</tr>
<tr>
<td>ZR</td>
<td>0000</td>
<td>28</td>
<td>51 54 64 90 100 100</td>
</tr>
</tbody>
</table>
DIAGNOSTICS

SIMT MESSAGE

72 ERROR AT COLUMN 20: UNEFFINE SYMBOL
80 ERROR AT COLUMN 20: UNEFFINE SYMBOL
105 UNKNOWN COMMAND NAME

3 STATEMENTS FLAGGED IN THIS ASSEMBLY