The Idea:

**CP** is a SS₂ instruction which is used to compare packed decimal fields. This instruction sets the condition code to “equal” (condition code = 0), “low” (condition code = 1) or “high” (condition code = 2) based on a comparison of the two fields as decimal numbers, and indicates how the first operand compares to the second operand. The fields may be of equal or different sizes. The only restrictions on the field lengths is that they must be a maximum of 16 bytes in length (the typical restriction for SS₂ fields). Consider the fields and instructions below.

```
APK      DC     P’123’ = X’123C’
BPK      DC     PL3’100’ = X’00100C
CHX      DC     X’123F’
... 
CP      APK,BPK     C.C.= HIGH
CP      APK,CHX     C.C.= EQUAL
CLC    APK,CHX     C.C.= LOW
```

In the first **CP**, the fields are compared arithmetically and it is found that +123 is greater than +100. In the second compare, the condition code is set to “equal” since +123 is equal to +123 (x’F’ is a valid plus sign). The third compare is a logical compare rather than an arithmetic compare. Since x’3C’ is lower than x’3F’ in the EBCDIC collating sequence, the condition code is set to “low”.

After setting the condition code with a **CP**, the condition code can be tested with a branch instruction. The typical branch instructions you might use are BE or BNE, BL or BNL, and BH or BNH.

**Examples**

Some unrelated CP’s:

```
QPK      DC     P’12345’ = X’12345C’
RPK      DC     P’-32’    = X’032D’
```
SZONED DC Z’11’ = X’F1C1’

... Results:
CP QPK,=P’20’ C.C. = HIGH
CP RPK,=P’20’ C.C. = LOW
CP SZONED,=P’999’ ABEND - SZONED IS NOT PACKED
CP QPK,QPK C.C. = EQUAL
CP QPK,RPK C.C. = HIGH
CP RPK,QPK C.C. = LOW
CP QPK,=X’324A’ C.C. = HIGH - LITERAL IS VALID PACKED

DATA
CP QPK+2(1),RPK BAD IDEA, BUT IT DOES WORK C.C. = HIGH,

Trying It Out in VisibleZ:

1) Load the program cp.obj from the \Codes directory and single step through each instruction. Why does CP set the condition code to high (2)?

2) Load the program cp1.obj from the \Codes directory and single step through each instruction. Why does CP set the condition code to equal/zero (0) when the fields have different signs and different lengths?

3) Load the program cp2.obj from the \Codes directory and single step through each instruction. Does the condition code represent the relationship of operand 1 compared to operand 2, or the other way round?

4) Load the program cp3.obj from the \Codes directory and single step through each instruction. Why does execution fall through the first BCR and take the branch on the second BCR?