Printing Packed Data

The unpack command, UNPK, has an unfortunate side effect.

Two commands to convert binary to printable EBCDIC.

CVD Converts the binary to packed decimal.

UNPK Almost converts to EBCDIC.

Consider the decimal number 42, represented in binary in register R4.

CVD R4,PACKOUT produces the packed decimal 042C.

When this is unpacked it should become F0 F4 F2

Unpack just swaps the sign half byte: F0 F4 C2.

This prints as 04B, because 0xC2 is the EBCDIC code for the letter ‘B’.

We have to correct the zone part of the last byte. See pages 167 & 175.

Printing Packed Data (Part 2)

Here is the code that works.

```
NUMOUT   CVD R4,PACKOUT  CONVERT THE NUMBER TO PACKED
         UNPK THESUM,PACKOUT PRODUCE FORMATTED NUMBER
         MVZ THESUM+7(1),=X'F0'  MOVE 1 BYTE
*                                TO ADDRESS THESUM+7
         BR 8  RETURN ADDRESS IN REGISTER 8
PACKOUT  DS PL8  HOLDS THE PACKED OUTPUT
```

THESUM has eight characters stored as eight bytes. The addresses are:

SUM	SUM +1	SUM +2	SUM +3	SUM +4	SUM +5	SUM +6	SUM +7
					Hundreds	Tens	Units

Again, the expression **`THESUM+7`** is an **address**, not a value.

If THESUM holds C‘01234567’, then THESUM+7 holds C ‘7’.

A Problem with the Above Routine

Consider the decimal number –42, stored in a register
in binary two's–complement form.

CVD produces 042D

UNPK produces F0 F4 D2

The above MVZ will convert this to F0 F4 F2, a positive number.

There are some easy fixes that are guaranteed to produce the correct representation for a negative number.

Most of the fixes using CVD and UNPK depend on placing the minus sign to the right of the digits. So that the negative integer –1234 would be printed as "**1234–**".

My Version of NUMOUT (Number Out)

This routine avoids packed decimal numbers.

We are given a binary number (negative or non–negative) in register R4.

1. Is the number negative?
   If so, set the sign to '–' and take the absolute value.
   Otherwise, leave the sign as either '+' or ' '.

We now have a non–negative number. Assume it is not zero.

2. Divide the number by 10, get a quotient and a remainder.
   The remainder will become the character output.

3. The remainder is a positive number in the range [0, 9].
   Add =X'F0' to produce the EBCDIC code.

4. Place this digit code in the proper output slot.

Is the quotient equal to 0? If so, quit.

If it is not zero, place the quotient in the dividend and return to 2.

NUMOUT: Example

Consider the positive integer 9413. We want to print this out.

Do repeated division by 10 and watch the remainders.

9413 divided by 10: Quotient = 941 Remainder = 3. Generate digit “3”.

941 divided by 10: Quotient = 94 Remainder = 1. Generate digit “1”.

94 divided by 10: Quotient = 9 Remainder = 4. Generate digit “4”.

9 divided by 10: Quotient = 0 Remainder = 9. Generate digit “9”.

Quotient is zero, so the process stops.

As they are generated, the digits are placed right to left, so that the result will print as the string 9413.

NUMOUT: Specifications

The code processes a 32–bit two's–complement integer, stored as a fullword in register R5 and prints it out as a sequence of EBCDIC characters.

The specification calls for printing out at most 10 digits, each as an EBCDIC character. The sign will be placed in the normal spot, just before the number.

For no particular reason, positive numbers will be prefixed with a "+".
I just thought I would do something different.

This will use repeated division, using the even–odd register pair (R4, R5), which contains a 64–bit dividend.

As a part of our processing we shall insure that the dividend is a 32–bit positive number. In that case, the "high order" 32 bits of the number are all 0.

For that reason, we initialize the "high order" register, R4, to 0 and initialize the "low order" register, R5, to the absolute value of the integer to be output.

The EBCDIC characters output will be placed in a 12–byte area associated with the label **`CHARSOUT`**, at byte addresses **`CHARSOUT`** through **`CHARSOUT+11`**.

Two New Instructions: LCR and STC

The code below will use two instructions that have not yet been discussed.

LCR (Load Complement Register)

Example `LCR R1,R2`

This loads register R1 with the negative (two's–complement) of the value in register R2. This is also used in my routine NUMIN.

STC (Store Character)

Example `STC R8,CHARSOUT(R3) PLACE THE DIGIT`

This transfers the EBCDIC character, with code in the low order 8 bits of the source register, to the target address. None of the bits in the register are changed.

The idea behind NUMOUT is to compute the numerical value of a digit in a source register, convert it to an EBCDIC code, and move it to the print line.

NUMOUT: Part 1

The first part checks the sign of the integer in register R4 and sets the sign character appropriately.

```
NUMOUT   MVC CHARSOUT,ZEROOUT  DEFAULT TO 0
         MVI THESIGN,C‘+’    DEFAULT TO A PLUS SIGN
         C   R5,=F‘0’        COMPARE R5 TO 0
         BE  DONE            VALUE IS 0, NOTHING TO DO
         BH  ISPOS           VALUE IS POSITIVE
         MVI THESIGN,C‘-’    PLACE A MINUS SIGN
         LCR R5,R5           2’S COMPLEMENT R5 TO MAKE POS
ISPOS    SR  R4,R4           CLEAR REGISTER 4
```

Here are some data declarations used with this part of the code.

```
*              123456789012
ZEROOUT   DC C‘           0’   11 SPACES AND A ZERO
CHARSOUT  DS CL12              UP TO 11 DIGITS AND A SIGN
```

Division (Specifically D – Divide Fullword)

This instruction divides a 64–bit dividend, stored in an even–odd register pair, by a fullword, and places the quotient and remainder back into the register pair.

This will use the even–odd register pair (R4, R5). The specifics of the divide instruction are as follows.

	R4	R5
Before division	Dividend (high order 32 bits)	Dividend (low order 32 bits)
After division	Remainder	Quotient

There are specific methods to handle dividends that might be negative.

As we are considering only positive dividends, we ignore these general methods.

Our Example of Division

Start with a binary number in register R5.

We assume that register R4 has been cleared to 0, as this example is limited to a 32–bit positive integer.

This code will later be modified to process the remainder, and store the result as a printable EBCDIC character.

```
DIVIDE   D  R4,=F‘10’   DIVIDE (R4,R5) BY TEN
*
*        THE REMAINDER, IN R4, MUST BE PROCESSED AND STORED
*
         SR R4,R4       CLEAR R4 FOR ANOTHER LOOP
         C  R5,=F‘0’    CHECK THE QUOTIENT
         BH DIVIDE      CONTINUE IF QUOTIENT > 0
```

Placing the Digits

At this point, our register and storage usage is as follows:

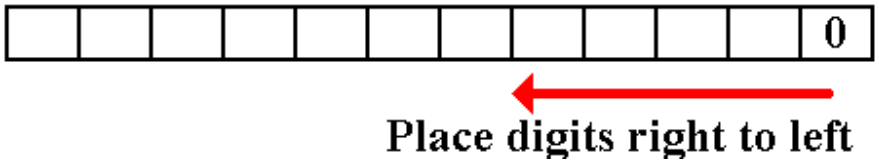
Register R3 will be used as an index register.

Register pair (R4, R5) is being used for the division.

Register pair (R6, R7) is reserved for use by the BXH instruction.

`CHARSOUT  DS CL12` contains the twelve characters that form the print representation of the integer.

The strategy calls for first placing a digit in the units slot (overwriting the '0') and then moving left to place other digits. To allow for a sign, no digit is to be placed in slot 0, at address **`CHARSOUT`**.



The idea will be to place the character into a byte specified by **`CHARSOUT(R3)`**.

The register is initialized at 11 and decremented by 1 using the BXH instruction.

The Digit Placement Code

Here is a sketch of the digit placement code. It must be integrated into the larger DIVIDE loop in order to make sense.

The register pair (R6, R7) is used for the BXH instruction.

R6 holds the increment value

R7 holds the limit value

```
L   R6,=F‘-1’        SET INCREMENET TO -1
SR  R7,R7            CLEAR R7.  LIMIT VALUE IS 0.
L   R3,=F‘11’        SET INDEX TO 11 FOR LAST DIGIT.
A   R4,=X‘F0’        ADD TO GET EBCDIC CODE FOR DIGIT
STC R4,CHARSOUT(R3) PLACE THE CHARACTER
BXH R3,R6,DIVIDE    GO BACK TO TOP OF LOOP
MVC CHARSOUT(R3),THESIGN   PLACE THE SIGN
```

The Complete Divide Loop

Here is the complete code for the divide loop. Note the branch out of the loop.

```
         L   R6,=F‘-1’    SET INCREMENET TO -1
         SR  R7,R7        CLEAR R7.  LIMIT VALUE IS 0.
         L   R3,=F‘11’    SET INDEX TO 11 FOR LAST DIGIT.
*
DIVIDE   D   R4,=F‘10’     DIVIDE (R4,R5) BY TEN
         A   R4,=X‘F0’    ADD TO GET EBCDIC CODE FOR DIGIT
         STC R4,CHARSOUT(R3)    PLACE THE CHARACTER
         SR  R4,R4         CLEAR R4 FOR ANOTHER LOOP
         C   R5,=F‘0’      CHECK THE QUOTIENT
         BNH PUTSIGN       EXIT LOOP IF QUOTIENT <= 0
         BXH R3,R6,DIVIDE    GO BACK TO TOP OF LOOP
*
PUTSIGN  MVC CHARSOUT(R3),THESIGN   PLACE THE SIGN
```