**Study Guide**

**CSCI 360 Programming**

**In**

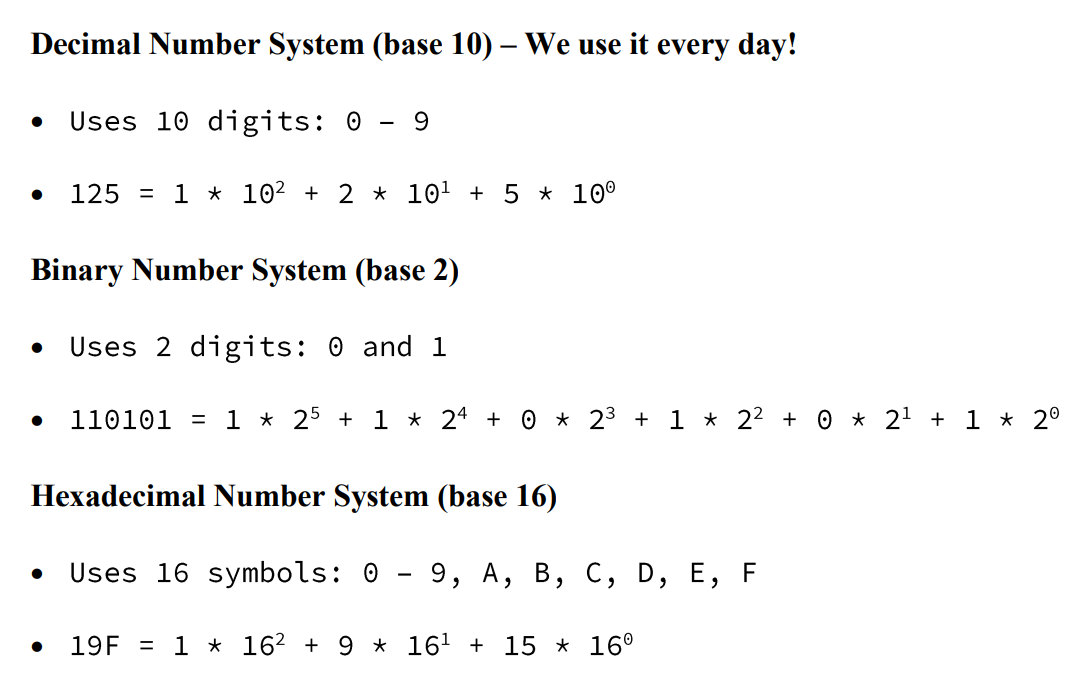
**Assembler using ASSIST**

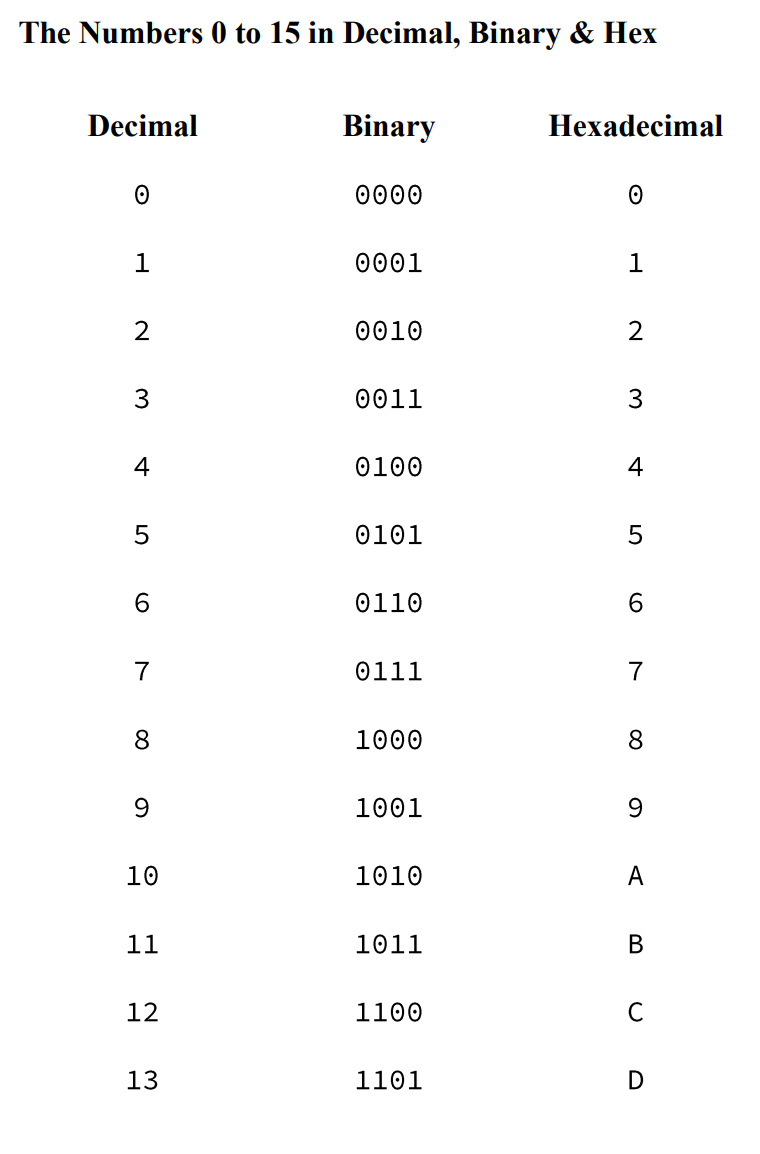
**Binary and Hexadecimal Number**

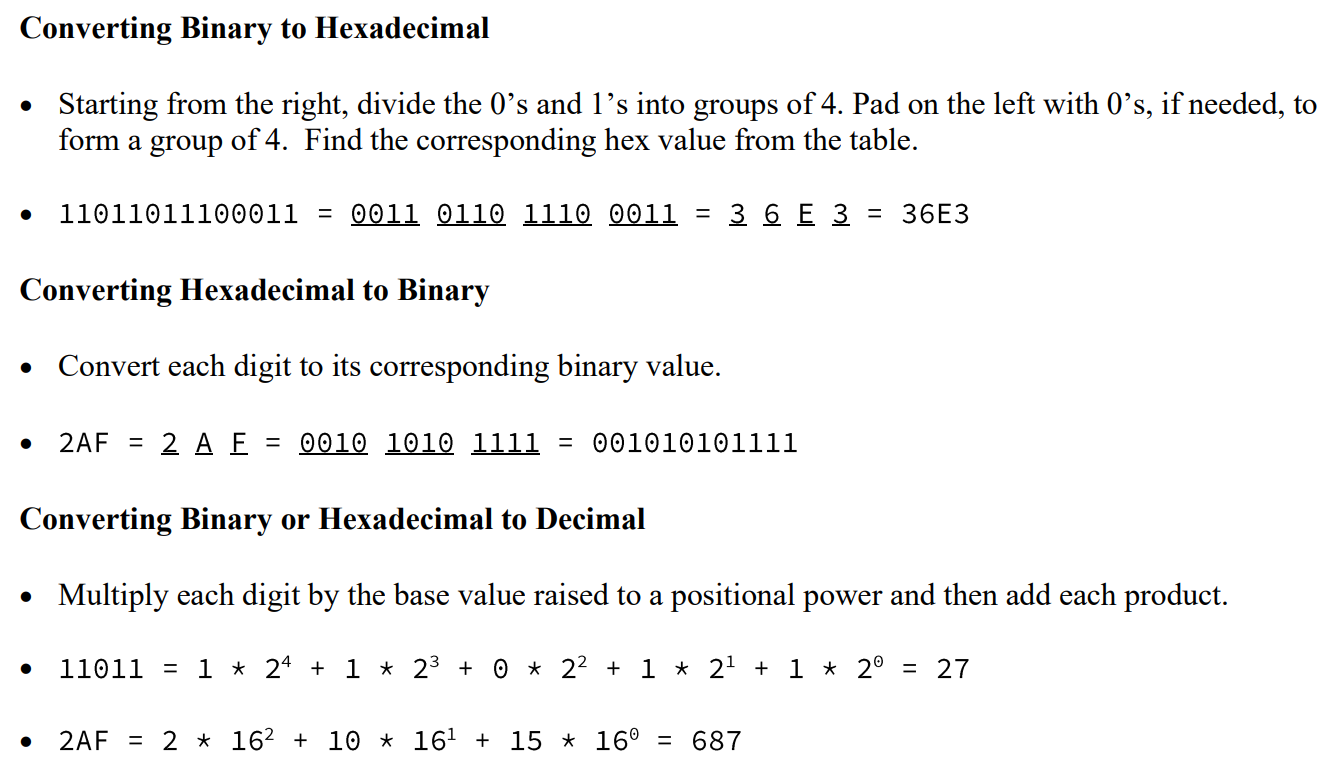
Decimal Number System (base 10)

Binary Number System (base 2)

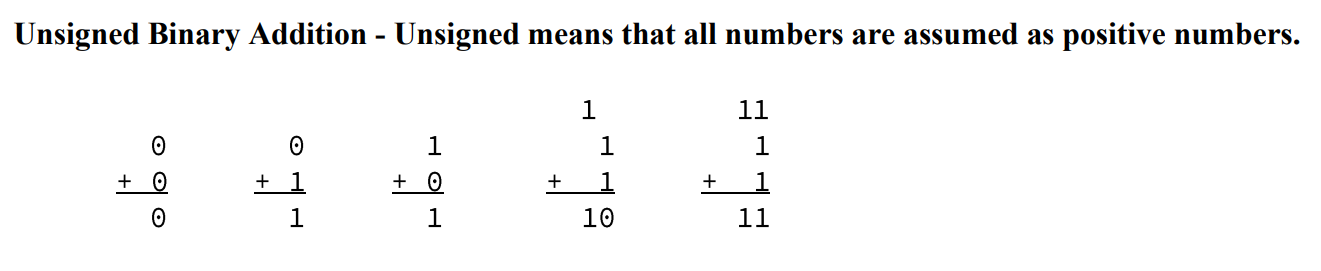
Hexadecimal Number System (base 16)



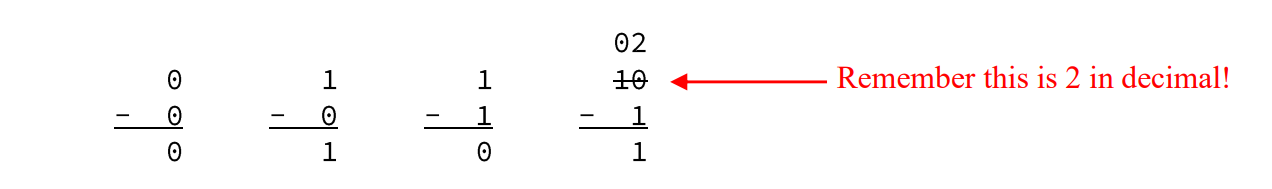




**Binary and Hexadecimal Arithmetic**



**Unsigned Binary Subtraction**



**Storage**

The main storage of a computer’s memory is made up of bits

Number in storage are limited in the number of bits allowed to represent a number.

* 1 bit 🡪 binary 0 and 1
* 1 byte 🡪 8 bites 🡪 2 hex digits
* 1 halfword 🡪 2 bytes 🡪 16 bits 🡪 4 hex digits
* 1 fullword 🡪 4 bytes 🡪 32 bits 🡪8 hex digits
* 1 doubleword 🡪 8 bytes 🡪 64 bits 🡪16 hex digits

The first digit is called **sign bit**. It represent if it’s a positive or negative value. To determine if it’s positive or negative take a look at the first digit and if its 0 through 7, the hex number is **positive**. If the sign bit starts from 8 through F the hex number will be **negative**.

The largest positive hexadecimal value that can be stored in one fullword (4bytes)

7FFFFFFF or 2^31 -1

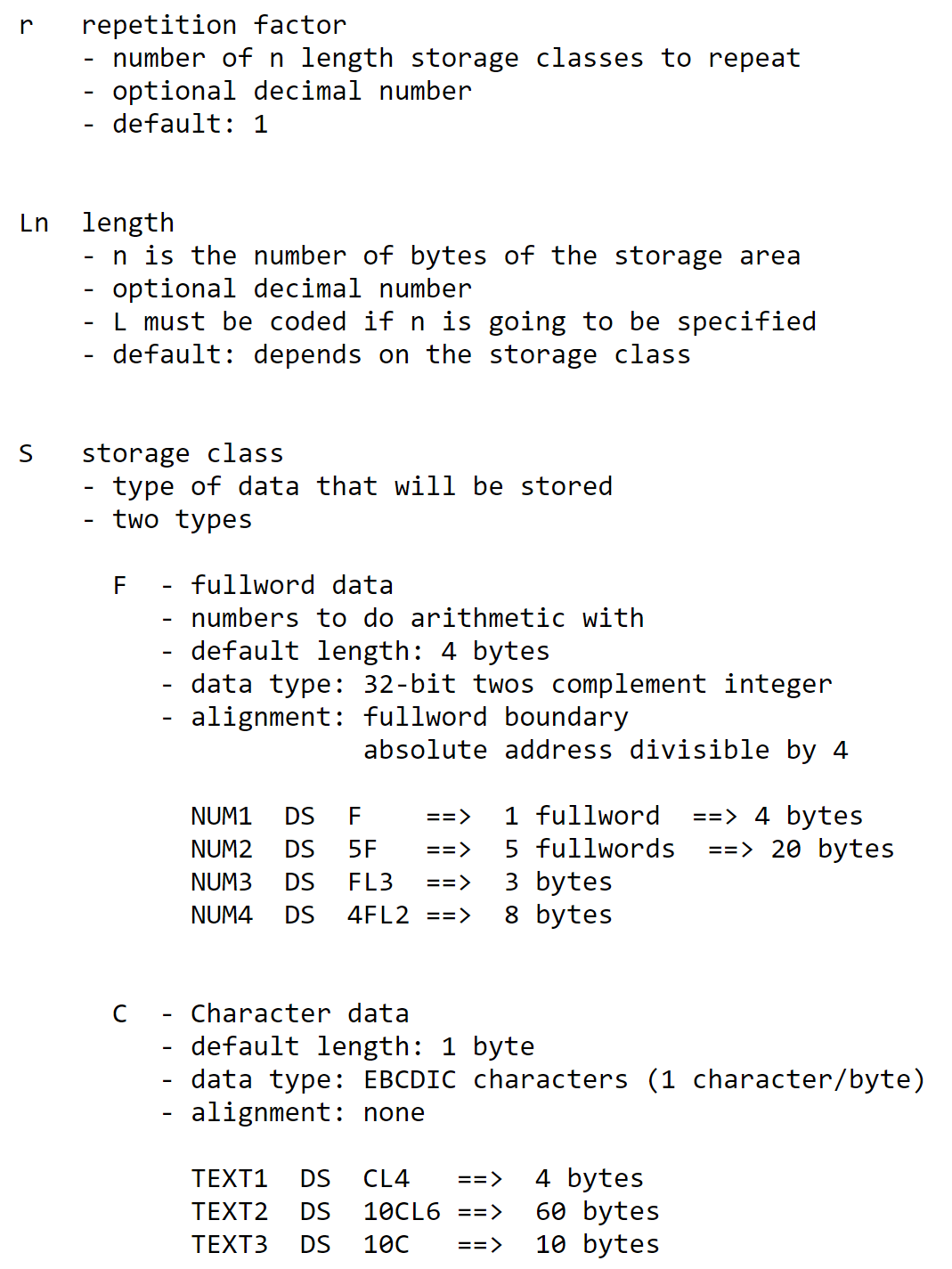
The most negative value that can be stored in one full word (4bytes)

80000000 or -2^31

Reference <https://learn-us-east-1-prod-fleet01-xythos.s3.amazonaws.com/5b06b62e5266a/18382385?response-cache-control=private%2C%20max-age%3D21600&response-content-disposition=inline%3B%20filename%2A%3DUTF-8%27%27360%25202.%2520Addressing%2520Storage%2520-%2520Notes.pdf&response-content-type=application%2Fpdf&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Date=20200708T150000Z&X-Amz-SignedHeaders=host&X-Amz-Expires=21600&X-Amz-Credential=AKIAZH6WM4PLTYPZRQMY%2F20200708%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Signature=edac70fdd5ccd578c11b5134a66c25181840f693ce1e88ae990ab39bcb5cf077>

**II. Memory Organization**

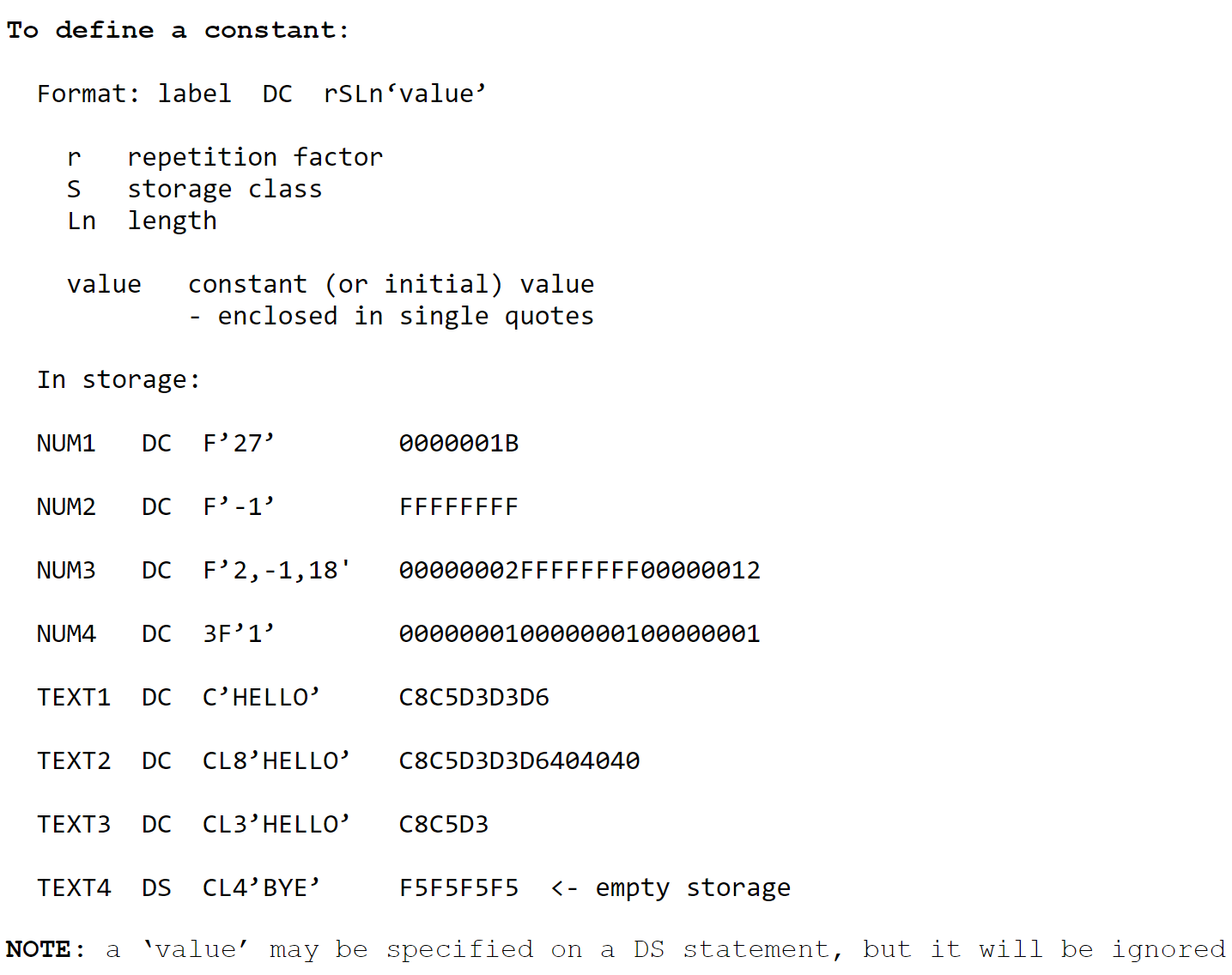
To define empty storage:

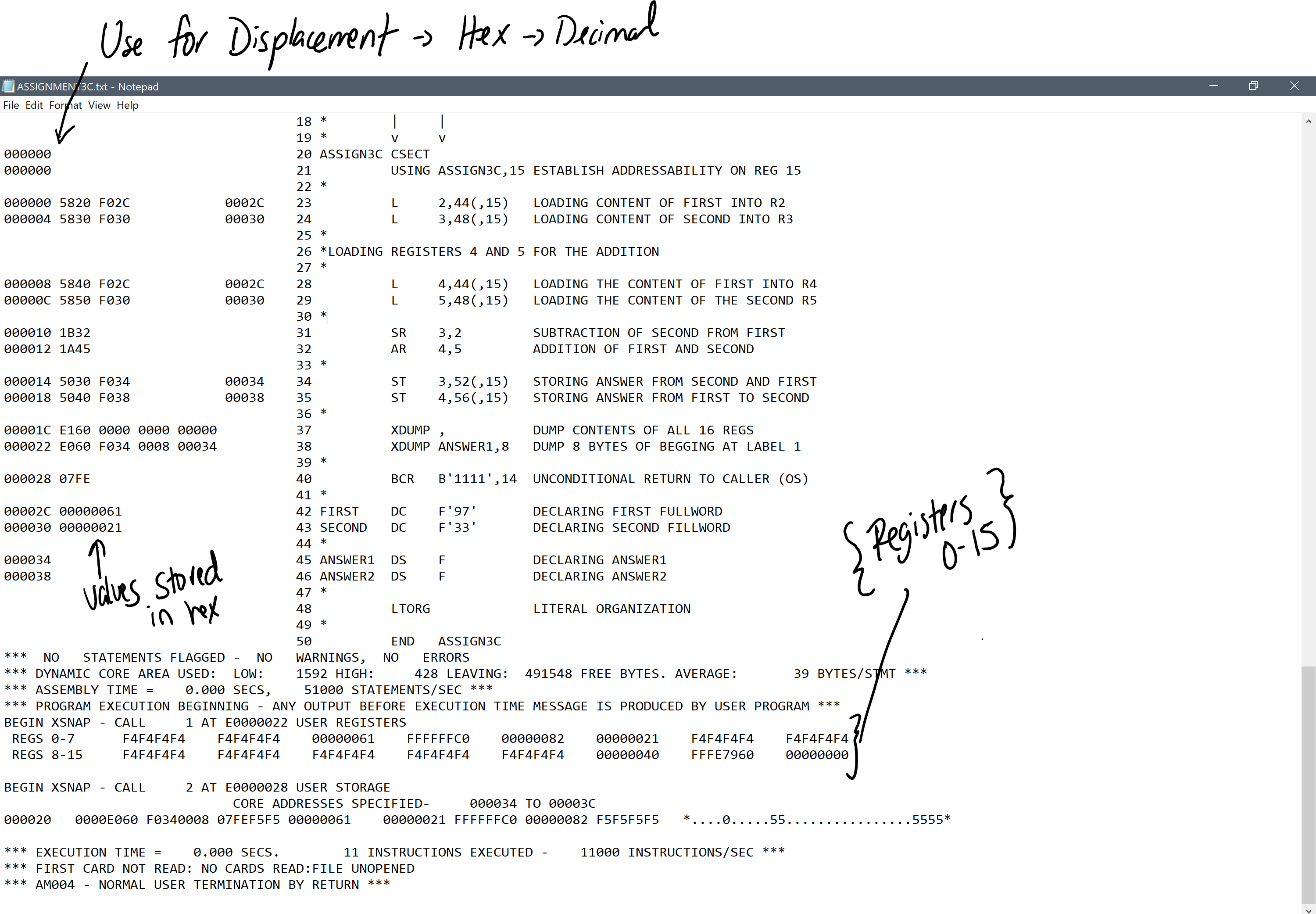
Format: label DS rSLn



DS – defines storage! Aka where its going to go, think about it as an empty variable

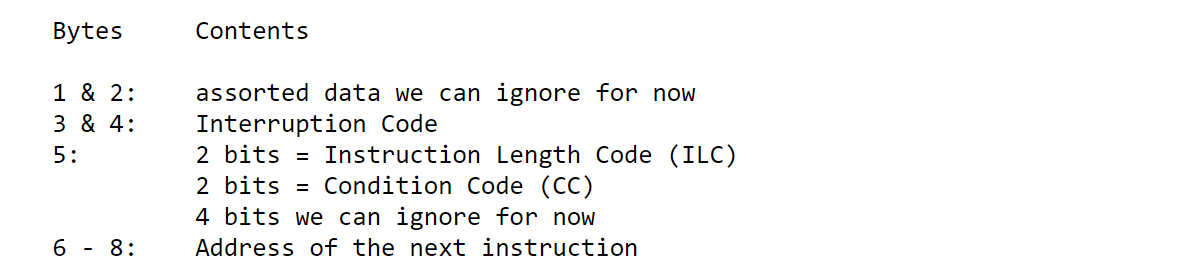
DC – defining a constant! Aka giving it a value already



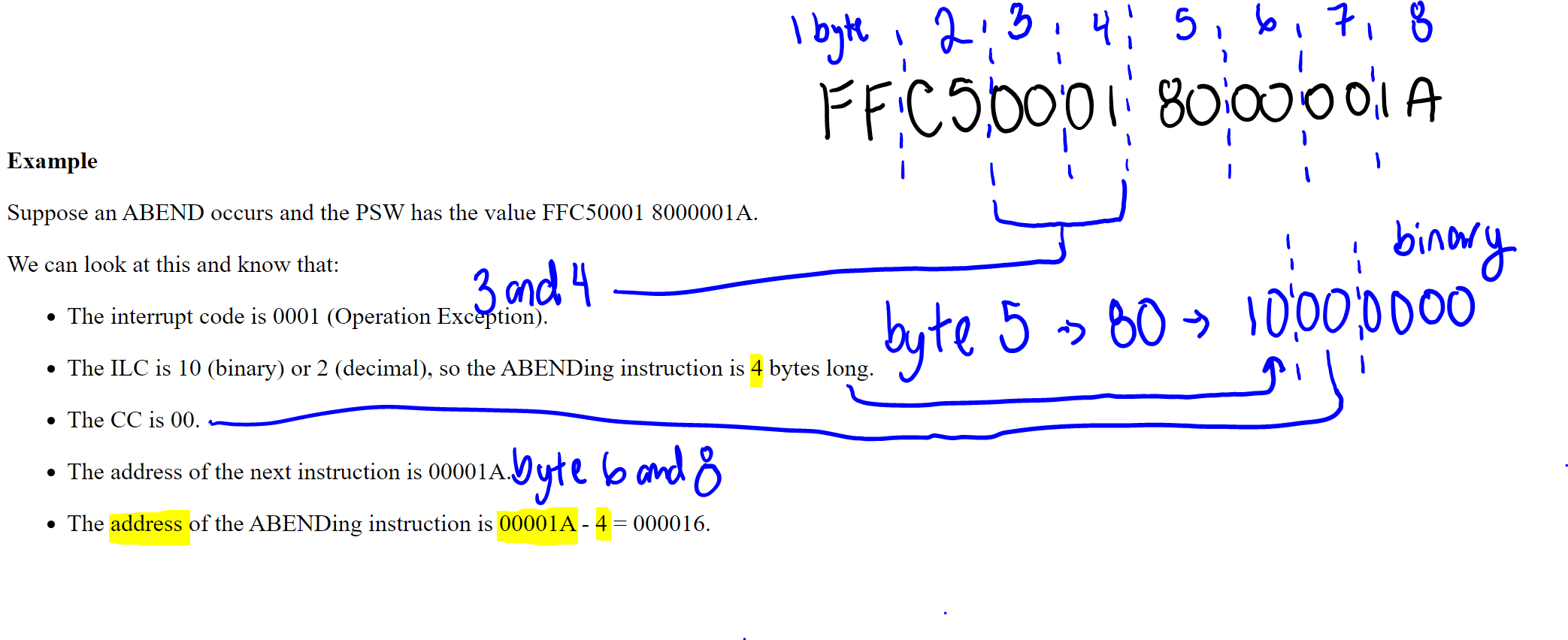
**III. Dumps**

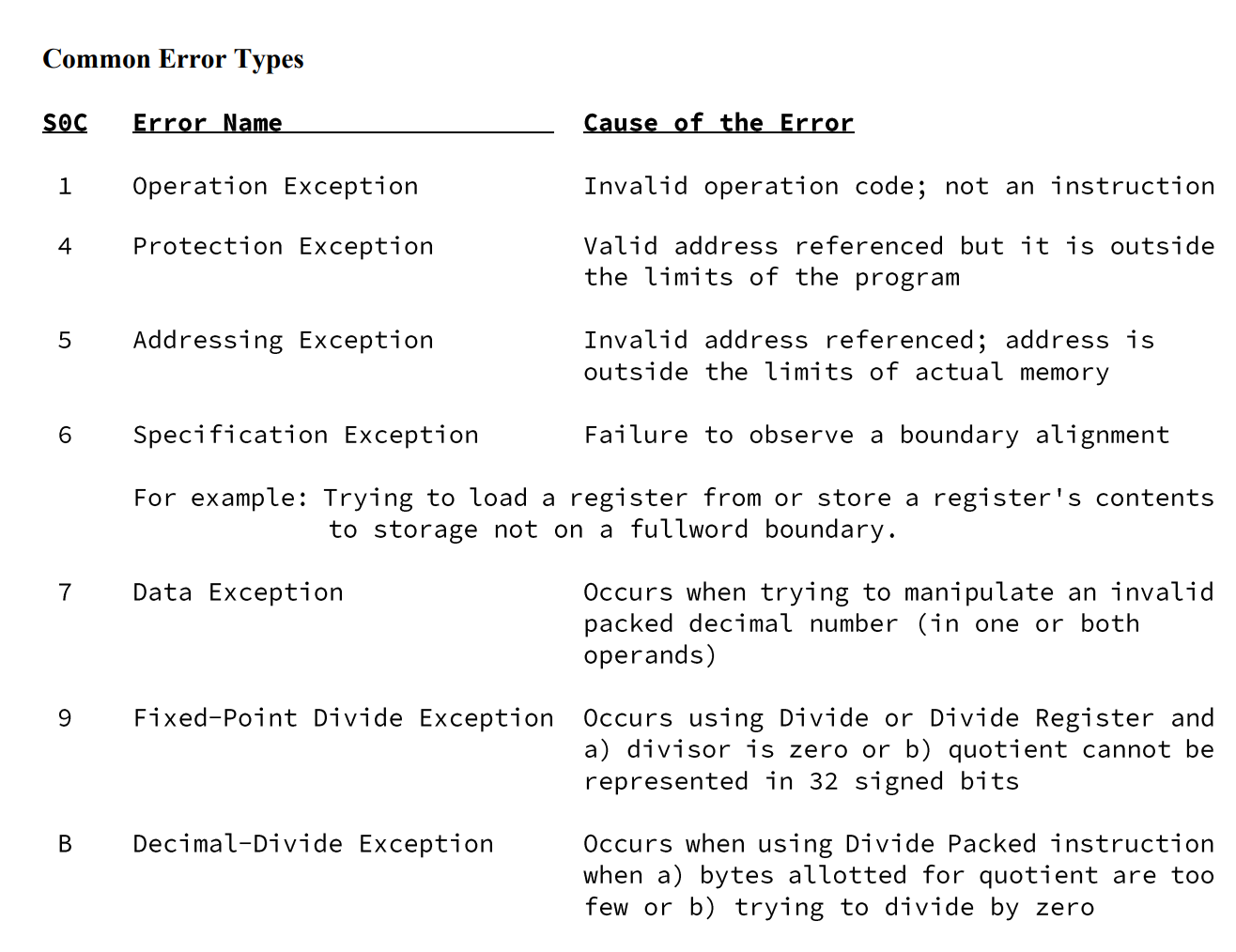
**Program Status Word(PSW)**

The PWS is a collection of data 8 bytes (or 64 bits) long, maintained by the operating system.





**Common Error Types**



**IV. Addressing Storage**

A slight problem, as programmers we do not know where a program is stored, or loaded into main storage. To solve this, programs use base-displacement, relative or explicit addressing.

***Relative address = base address + displacement***

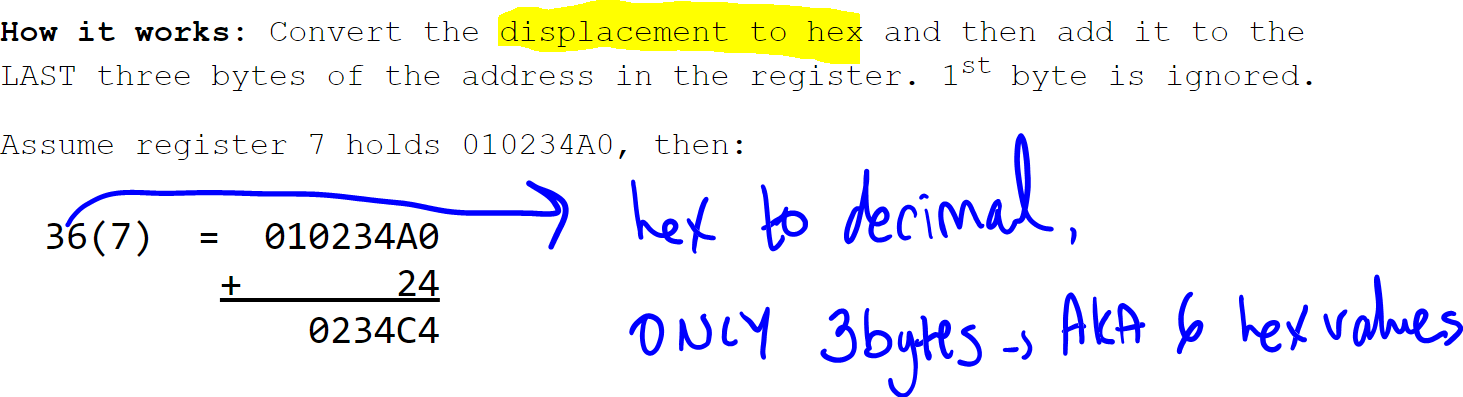
The base address Is place in a general purpose register when its executed. The GPR is called base register

**Two Forms of Relative Addressing**

1. D(B)

* D is the displacement. Decimal number between 0 – 4095
* B is the base register number which is a GPR between 0 – 15

Formats 🡪 D(B), D



1. D(X,B)

* D is the displacement. Decimal number between 0 – 4095
* X is the index register between 0 – 15
* B is the base register number which is 0 – 15

Formats 🡪 D(X,B) , D(X) , D(,B) , D

**DON’T FORGET! Convert the displacement from hex to decimal then add the address, also only 3 bytes format ALWAYS (aka 6 hex values only)**

**Instructions Encoding and Decoding**

Different instructions require different formats of their operands.

Examples

* RR instructions require two registers as operands
* RX instructions require a register as first operands and a D(X,B) type address as second

**Encoding RR Instructions**

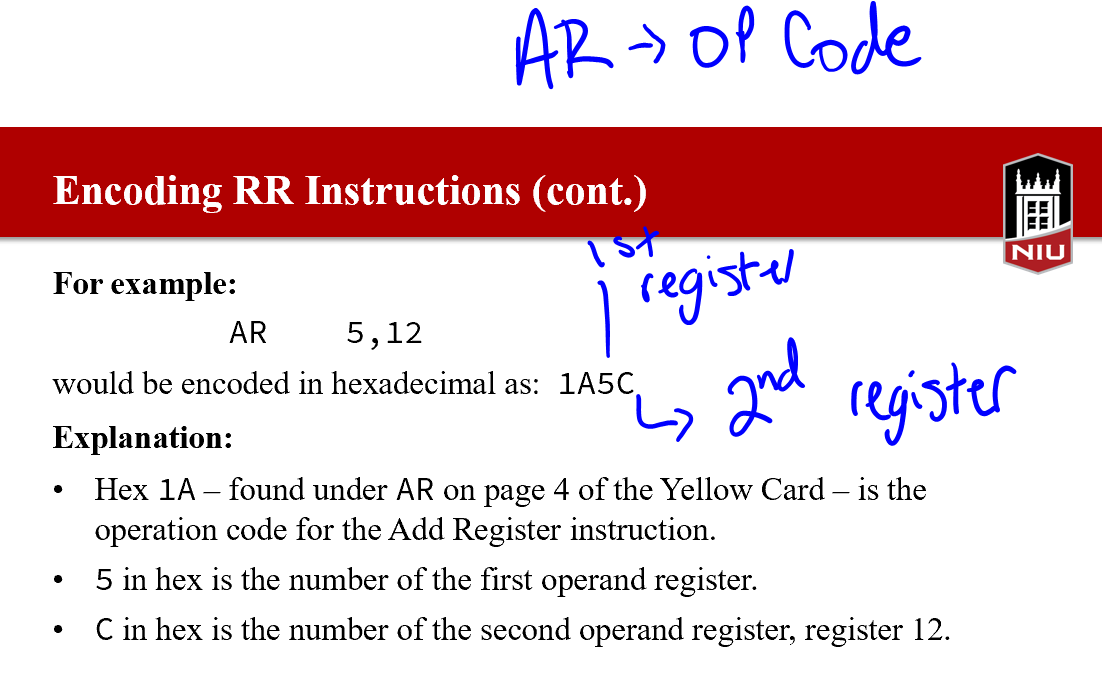
RR instructions are 2 bytes long

RR instructions always have the format:

Bits 0 -7 are the instructions operation code, or “op code” in hex

Bits 8 – 11 represent the first operands register number in hex

Bits 12 – 15 represent the second operands register number in hex



**Encoding RX Instructions**

RX instructions are 4 bytes long

RX instructions always have the format:

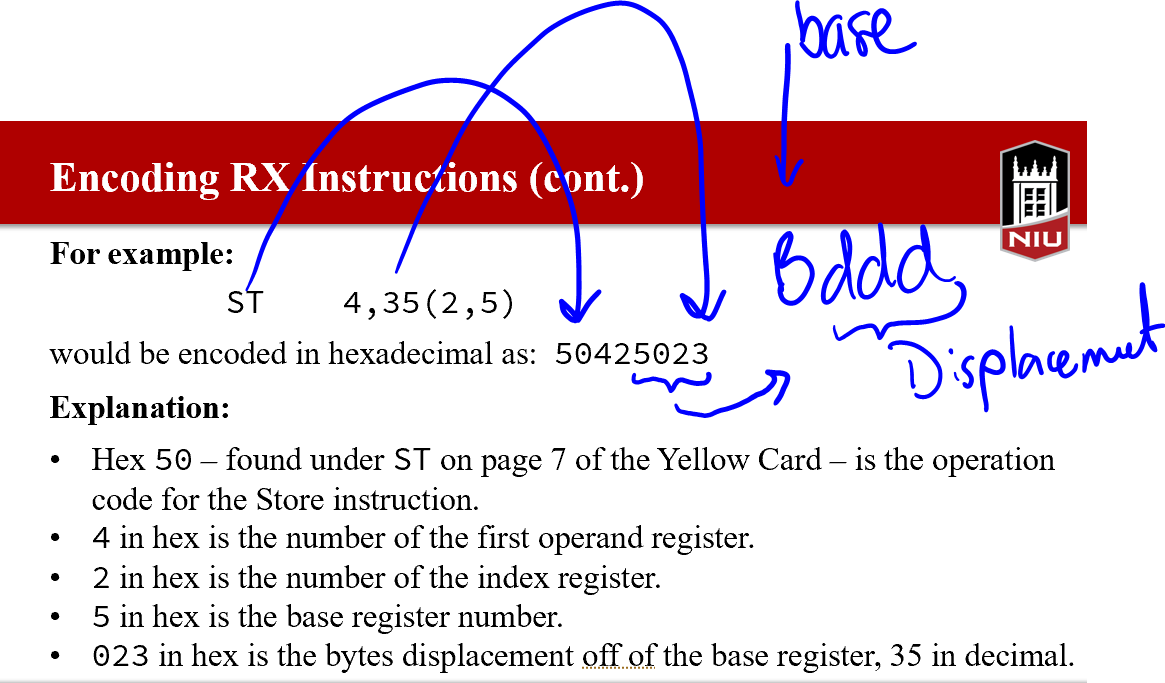
Bits 0 – 7 instructions op code

Bits 8 – 11 first operands register number in hex

Bits 12-15 index register number in hex

Bits 16 – 19 base register number in hex

Bits 20 – 32 displacement in bytes off of base register in hex



**Encoding reference**

<https://learn-us-east-1-prod-fleet01-xythos.s3.amazonaws.com/5b06b62e5266a/18518395?response-cache-control=private%2C%20max-age%3D21600&response-content-disposition=inline%3B%20filename%2A%3DUTF-8%27%27360%25206.%2520Encoding%2520and%2520Decoding%2520Instructions%2520-%2520Notes%25282%2529.pdf&response-content-type=application%2Fpdf&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Date=20200708T180000Z&X-Amz-SignedHeaders=host&X-Amz-Expires=21600&X-Amz-Credential=AKIAZH6WM4PLTYPZRQMY%2F20200708%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Signature=b64ba8a6379495b91da2d628985298922856b23faf16e7b0c37d1b67a48ce47b>

**SS Instructions**

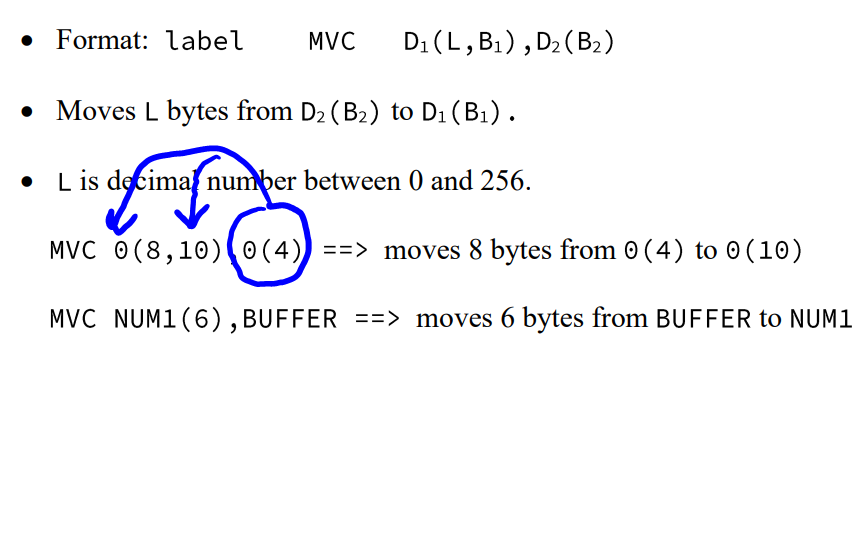
**(Storage to Storage Instructions)**

Move Character (**MVC**) and Control Logical Character (**CLC**) are SS instructions

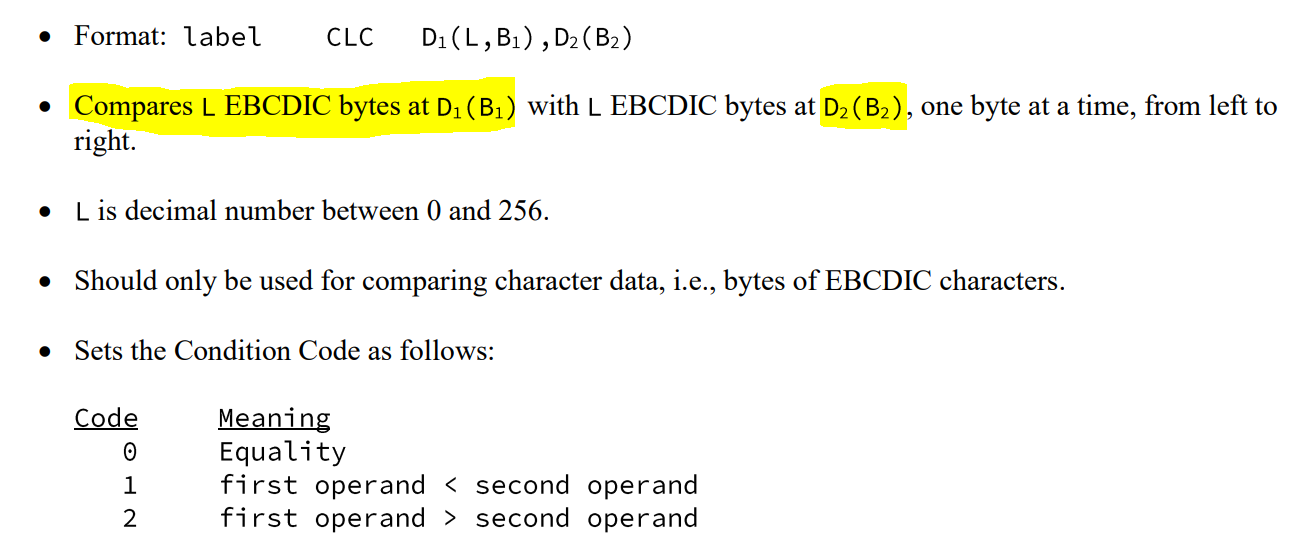
Have a 2 D(B) operands NOT D(B,X), along with a size of 6 byte instructions

**Move Character –**

Rather than move, it copies them to D1(B1)



**Compare Logical Character (CLC)**

As it works its way from left to right, byte-by-byte, logically comparing bytes in the two storage areas, the moment it finds inequality, the comparison stops and the condition code is set.

**reference** <http://faculty.cs.niu.edu/~byrnes/csci360/notes/360ss.htm>



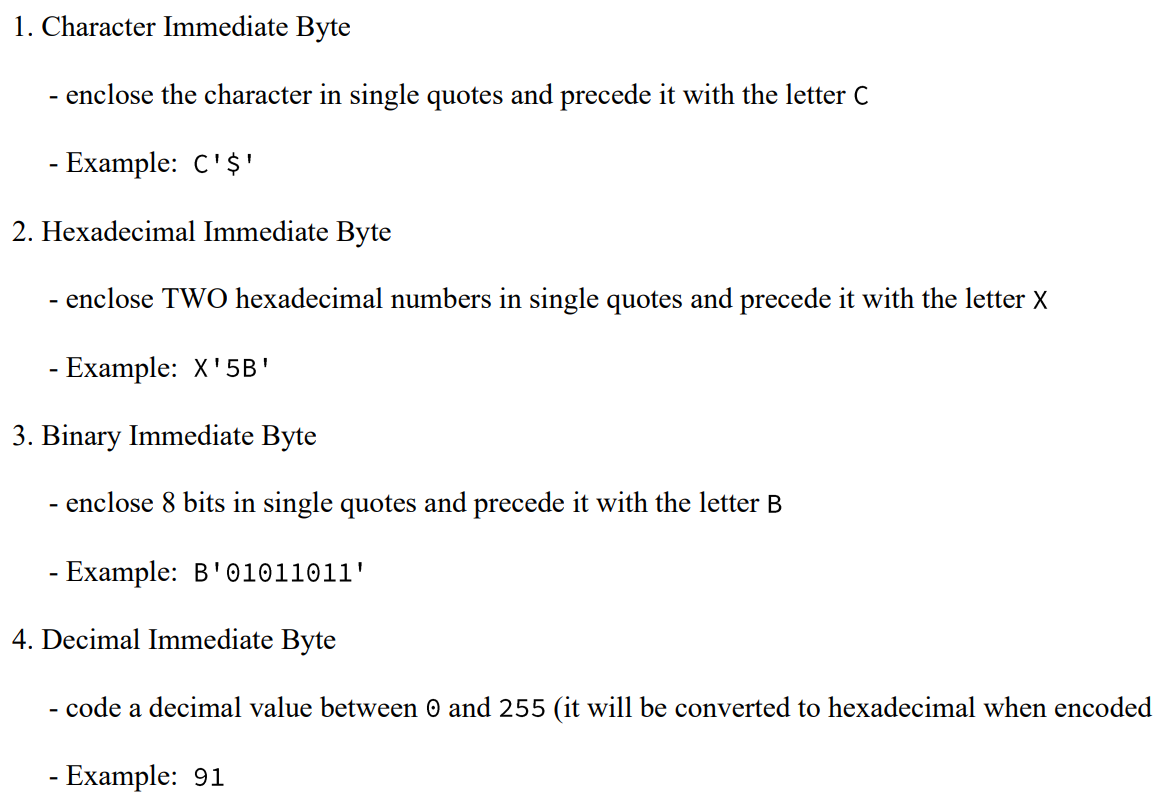
**Immediate Byre Instructions**

**(MVI, CLI)**

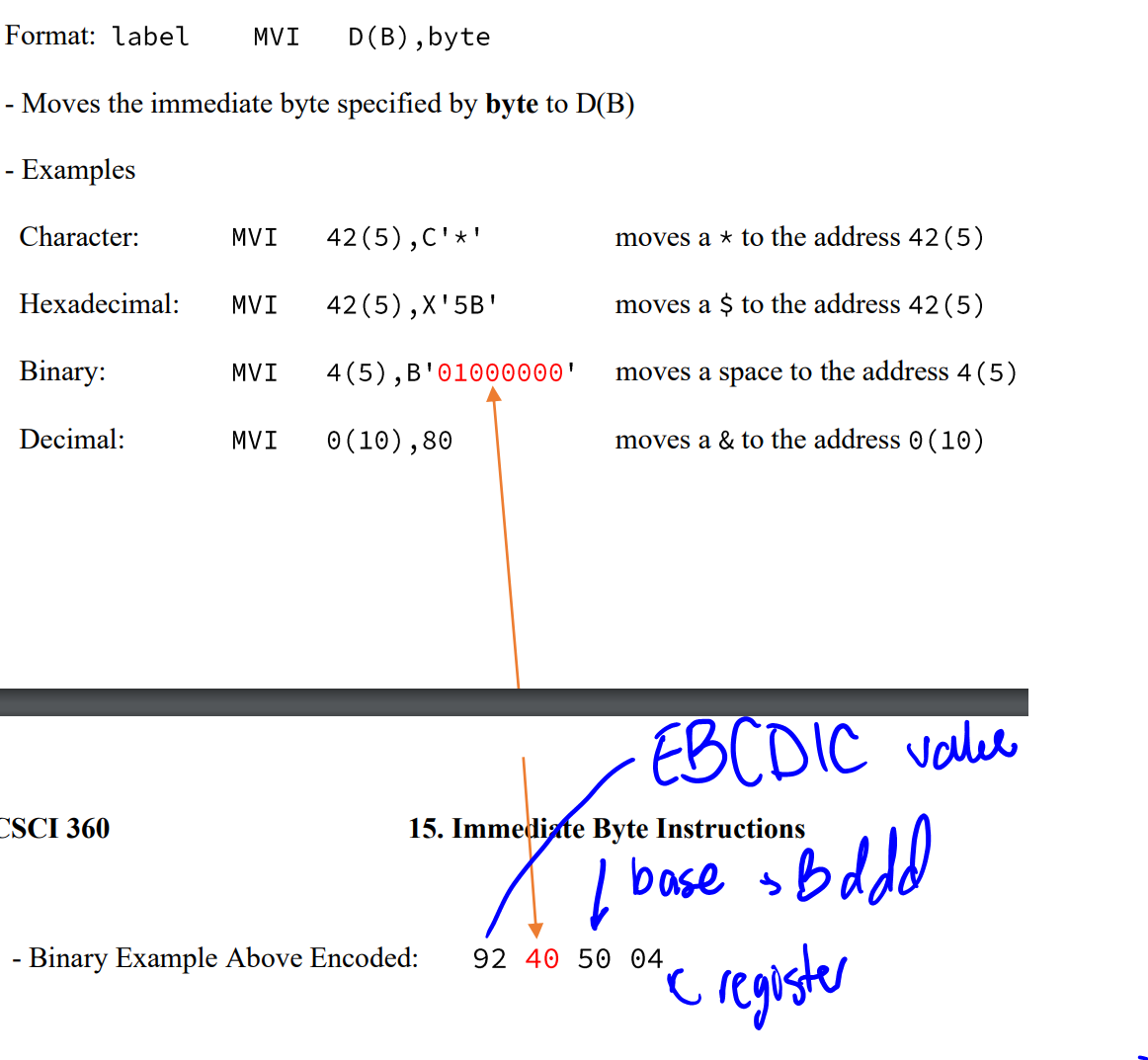
Storage Immediate (SI) Type Instructions Format

Instructions that involve a byte in storage represented by D(B) address and an immediate byte. The immediate byte is a part of the 4 -byte encoded instructions. In fact, it is the second byte of the encoded instructions.

They can be specified in 4 ways

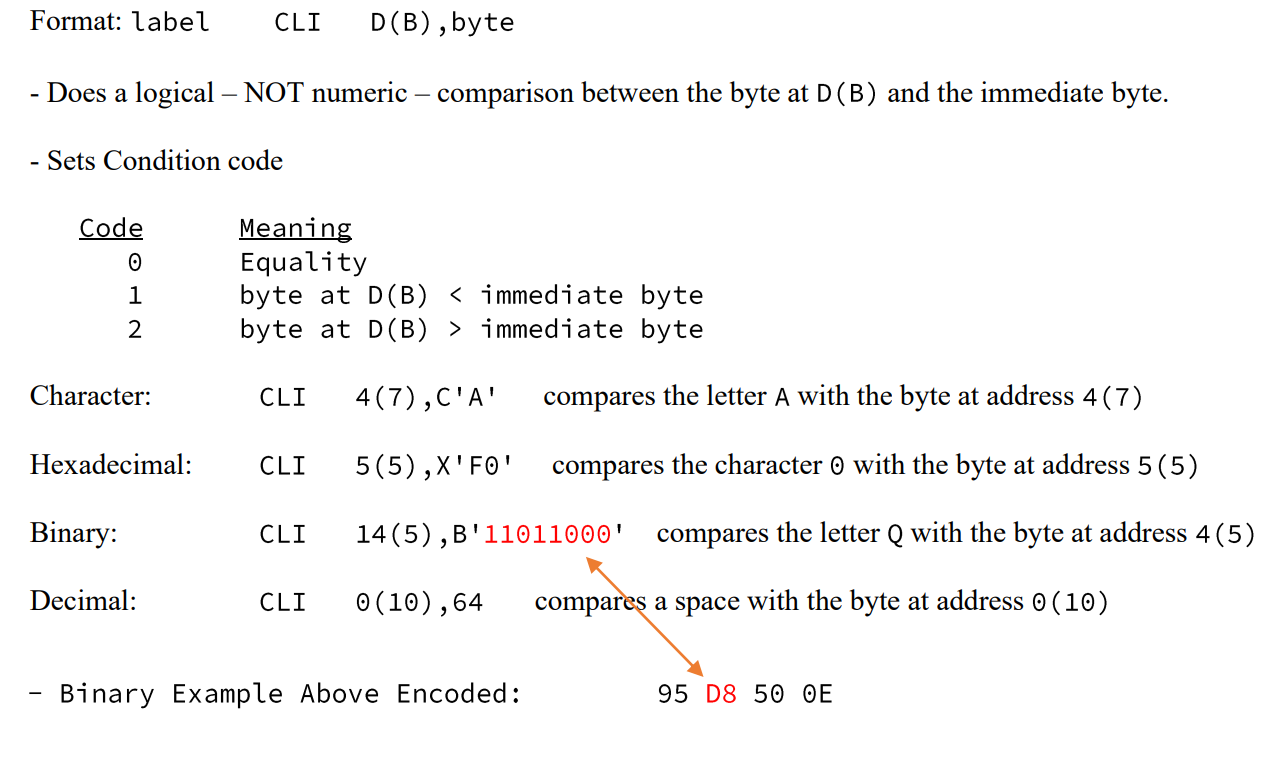


**Move Immediate**Moves the immediate byte specified by byte to D(B)



**Compare Logical Immediate**

Does a logical – NOT numeric – comparing between the byte ad D(B) and the immediate byte.

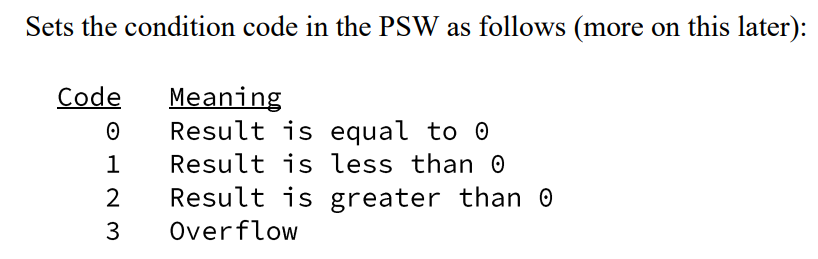


**Reference** <https://learn-us-east-1-prod-fleet01-xythos.s3.amazonaws.com/5b06b62e5266a/18530145?response-cache-control=private%2C%20max-age%3D21600&response-content-disposition=inline%3B%20filename%2A%3DUTF-8%27%27360%252015.%2520Immediate%2520Byte%2520Instructions%2520-%2520Notes%25281%2529.pdf&response-content-type=application%2Fpdf&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Date=20200708T210000Z&X-Amz-SignedHeaders=host&X-Amz-Expires=21600&X-Amz-Credential=AKIAZH6WM4PLTYPZRQMY%2F20200708%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Signature=1910d7df041b9c5c87141623546178f9998215eefa58f87a42e589b40b18bea9>

**VI. Important Instructions**

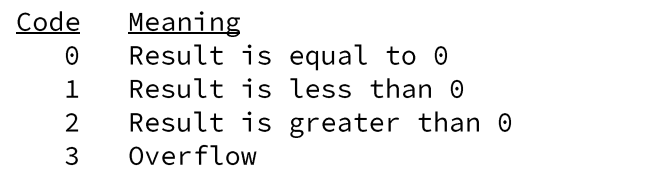
**Add**

Format: label A R,D(X,B)

Take the 4 byte from the absolute address represented by D(X,B) and adds it to the contents of R The result is store.

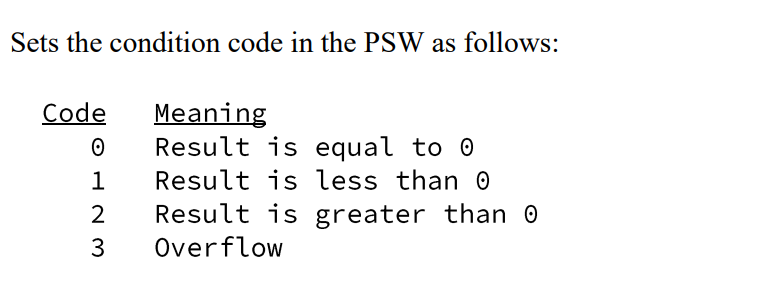
**Add Register**

Format: label AR R1,R2

Adds the contents of R2 to R1. Results is place in R1

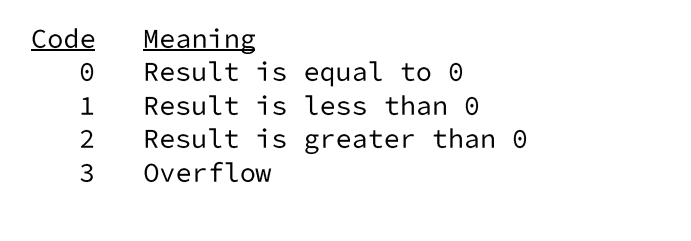
**Subtract**

Format: label S R,D(X,B)

Takes the 4 bytes from the absolute address represent by D(X,B) and subtracts it from the content of R.

**Subtract Register**

Format: label SR R1,R2

Subtracts the contents of R2 from R1

**Load**

Format: label L R,D(X,B)

Copies the 4 bytes at the absolute address represented by D(X,B) into R, the previous contents of R are overwritten.

**Load Address**

Format: label LA R,D(X,B)

Loads the 24 bits address – NOT THE CONTENT! Of the four bytes at the absolute address represent by D(X,B), replace that 3 byte address in the last three bytes of the register and zeroes out the first byte of the registers ALWAYS! No RR equivalent.

**Load Register**

Format: label LR R1,R2

Copies the contents or R2 into R1, the previous content of R1 gets overwritten.

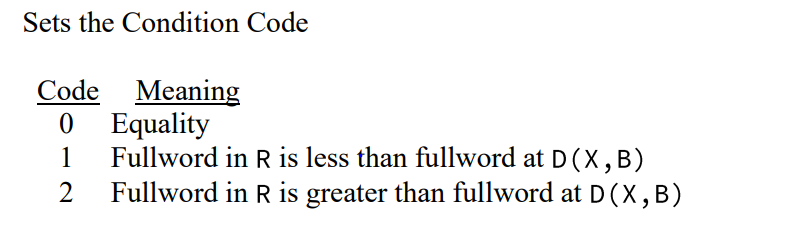
**Store**

Format: label ST R,D(X,B)

Stores the content of R at the absolute address represented by D(X,B)

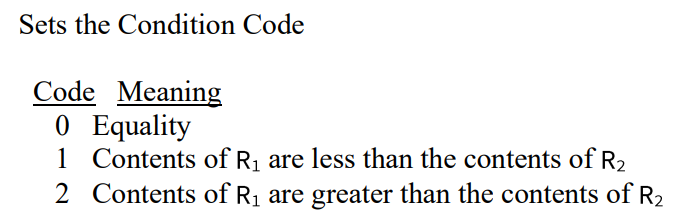
**Compare**

RX format: label C R,D(X,B)

Compares the fullword in R with fullword at D(X,B)

**Compare Register**

RR Format: label CR R1,R2

Compare the value in R1 to the value in R2

**Branching**

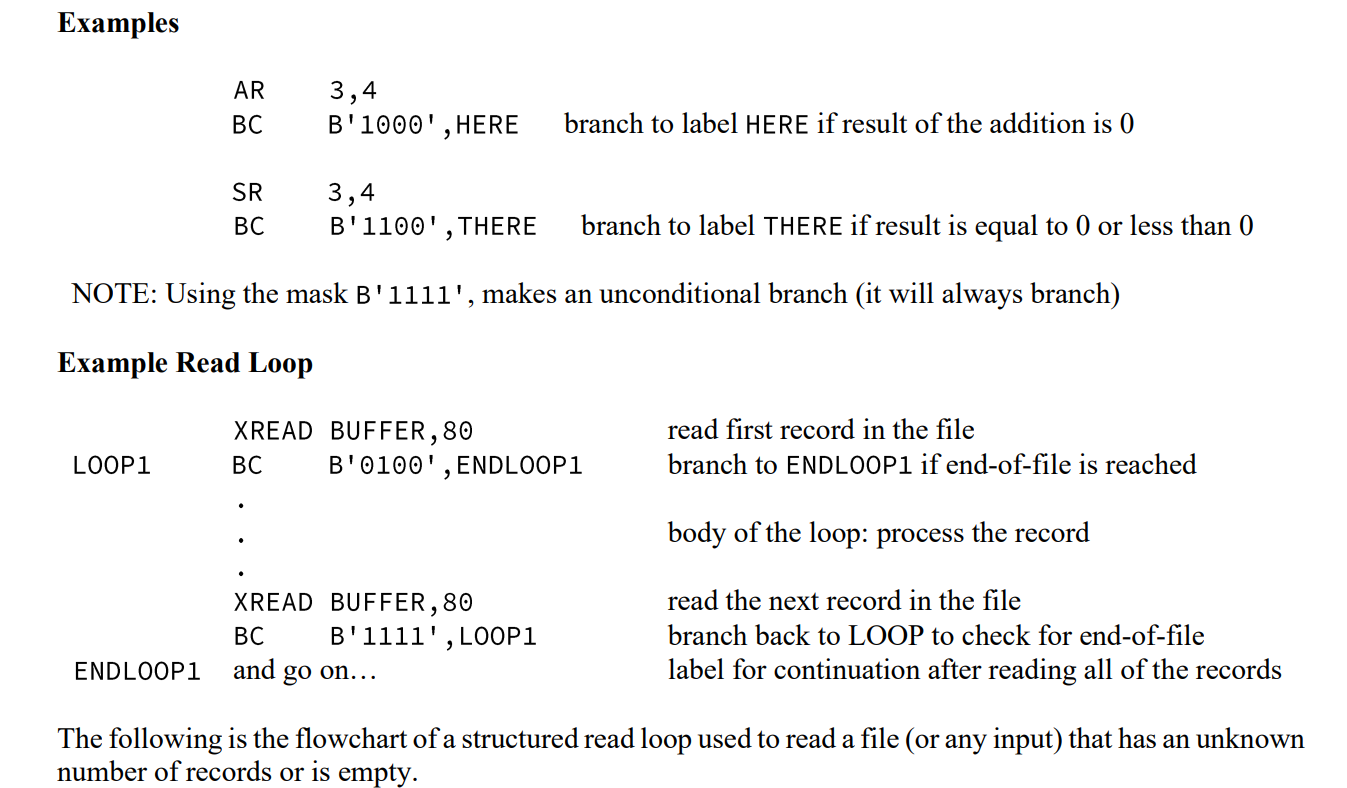
Used to alter the flow of program execution depending on the condition code set by a previous instruction.

**Branching on Condition**

RX format: label BC B’mask’,D(X,B)

* Mask is a 4 bit binary mask indicating which condition code(s) to branch on
* D(X,B) is the address to branch to

The mask is interpreted positionally

B indicates that what follows inside the single quotes, or ticks, is in binary. If you think about it, its like a loop in C++ If, then.

**Branch on Condition Register**

RR format: label BCR B’mask’,R

* Mask is a bit binary mask that indicating which condition code(s) to branch on
* R is the register with the address of the branch to.

**Multiplication**

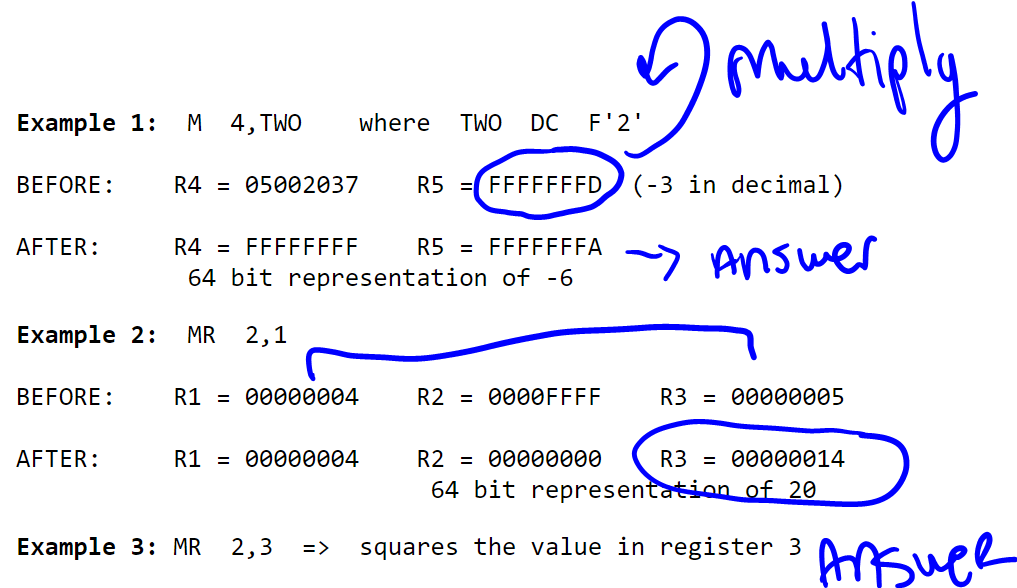
Multiply two 32-bit number to obtain a 64-bit result that is spread between an even-odd pair or registers

RX format: label M R,D(X,B)

* R is an even number register represented an even-odd pair of registers. The number is multiply must be in the odd numbered register.
* D(X,B) is the address of a fullword number to multiply by

RR format: label MR R1,R2

* R1is an even number register represented an even-odd pair of registers. The number is multiply must be in the odd numbered register.
* R2 is the number of a register containing the number to multiply by



**Division**

Divide a 64-bit number that is stored between an even-odd pair of registers by a 32-bit number. The results in 32-bit quotient stored in the odd register and a 32 remainder stored in the even register. Remember that the remainder will ALWAYS have the same sign as the number being divided.

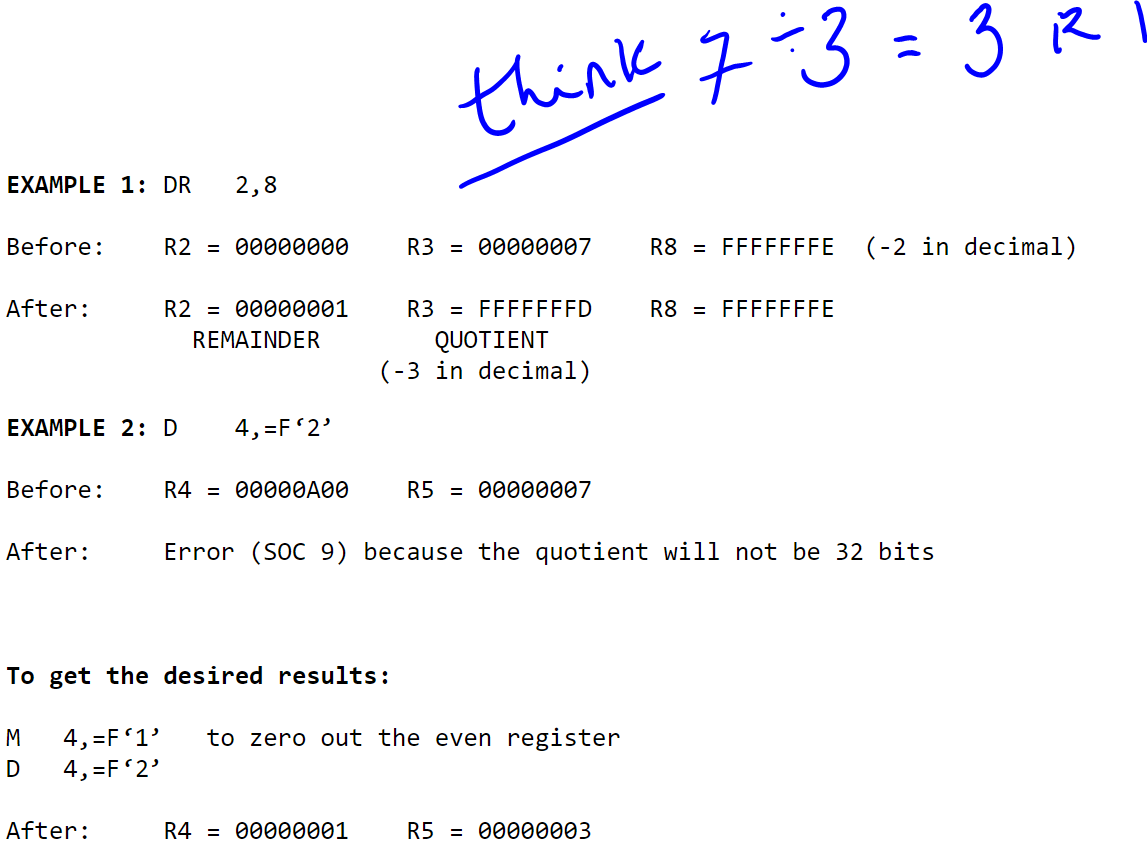


RX format: label D R,D(X,B)

* R is an even numbered register representing an even-odd pair of register. The dividend (aka the number to be divided) must be spread between the two registers.
* D(X,B) is the address of a fullword number to divide by (aka the divisor)

RR format: label DR R1,R2

* R1 is an even numbered register representing an even-odd pair of register. The dividend (aka the number to be divided) must be spread between the two registers.
* R2 is the number of a register containing the numbered to divide by (aka the divisor)



For the top example, divide normally then convert to the negative value using the calculator.

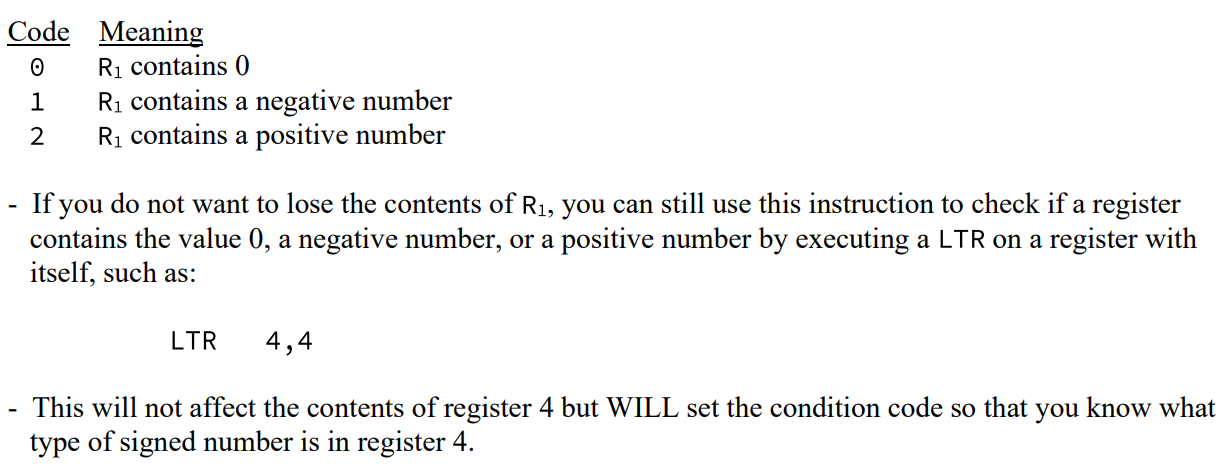
Reference <https://learn-us-east-1-prod-fleet01-xythos.s3.amazonaws.com/5b06b62e5266a/18518405?response-cache-control=private%2C%20max-age%3D21600&response-content-disposition=inline%3B%20filename%2A%3DUTF-8%27%27360%252010%2520Multiplication%2520and%2520Division%2520-%2520Notes.pdf&response-content-type=application%2Fpdf&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Date=20200709T000000Z&X-Amz-SignedHeaders=host&X-Amz-Expires=21600&X-Amz-Credential=AKIAZH6WM4PLTYPZRQMY%2F20200709%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Signature=31c1f71ce266df7798e23bb5c7353ae06488de78491f70efb3d7d7cdce5343d3>

**More Load Instructions**

**Load and Test Registers**

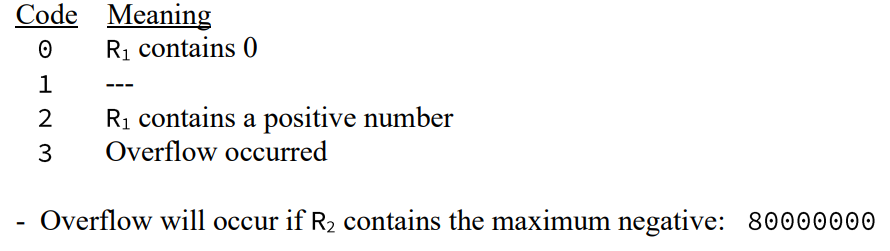
Format: label LTR R1,R2

Loads the contents of R2 into R1 and sets only the condition code. Of course, R1 contents are lost but the register you are testing, R2, with instructions remains unaffected.

Think about it as a check for the registers what it contains, a 0 or pos or negative.

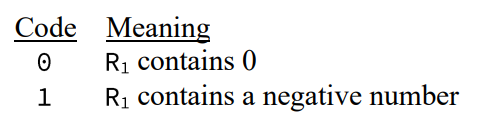
**Load Positive Register**

Format: label LPR R1,R2

Loads the **absolute value** of the contents of R2, into R1, and sets the condition code.

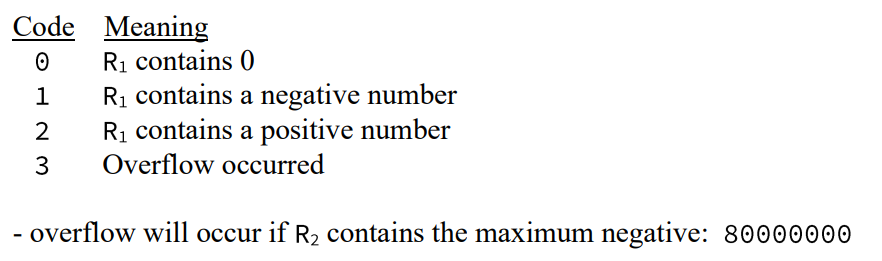
**Load Negative Register**

Format: label LNR R1,R2

Loads the negative of the **absolute value** of the contents of R2, into R1and sets the condition code.

**Load Complement Register**

Format: label LCR R1,R2

Loads the complement (opposite) of the contents or R2, into R1, and sets the condition code.

Reference <https://learn-us-east-1-prod-fleet01-xythos.s3.amazonaws.com/5b06b62e5266a/18518604?response-cache-control=private%2C%20max-age%3D21600&response-content-disposition=inline%3B%20filename%2A%3DUTF-8%27%27360%252011%2520More%2520Load%2520Instructions%2520-%2520Notes.pdf&response-content-type=application%2Fpdf&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Date=20200709T000000Z&X-Amz-SignedHeaders=host&X-Amz-Expires=21600&X-Amz-Credential=AKIAZH6WM4PLTYPZRQMY%2F20200709%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Signature=bcab34aadc7ecd028db4760bb392a64670cb14e8810cd009e77ec49dce82317f>

**Equates and Extended Mnemonics**

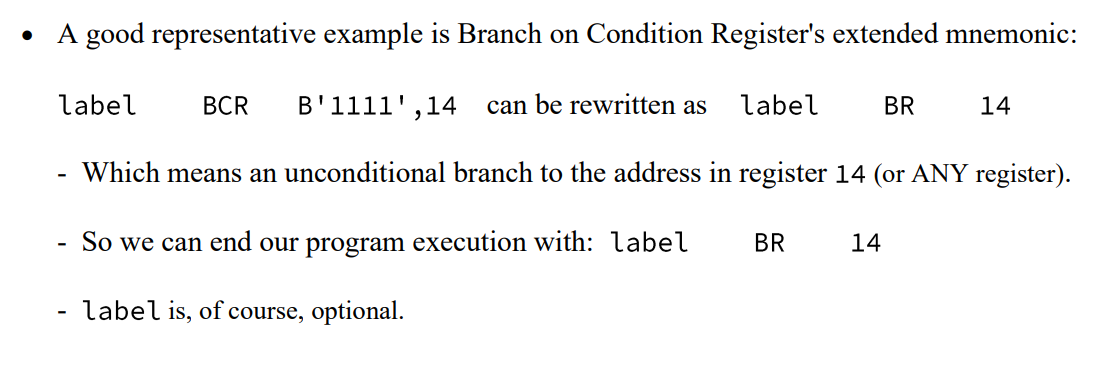
**EQUates Assign a value to a label**

Format: label EQU expression

What this does is, gives a label toe the value of the expression.

**Extended Branch Mnemonics**

Creating extended branch mnemonics makes it easier for the branch instruction by eliminating the need to explicitly specify a mask. Its like short cuts pretty much!



At the following link, there’s a table were you can see the different branch mnemonics you can use.

<https://learn-us-east-1-prod-fleet01-xythos.s3.amazonaws.com/5b06b62e5266a/18518735?response-cache-control=private%2C%20max-age%3D21600&response-content-disposition=inline%3B%20filename%2A%3DUTF-8%27%27360%252012%2520EQUates%2520and%2520Extended%2520Mnemonics%25281%2529.pdf&response-content-type=application%2Fpdf&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Date=20200709T000000Z&X-Amz-SignedHeaders=host&X-Amz-Expires=21600&X-Amz-Credential=AKIAZH6WM4PLTYPZRQMY%2F20200709%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Signature=484231446a87910e1be69bf452f87c143a20c48c0269f5163ef71f04842d373f>

**BCT and BCTR Instructions**

**Branch on Count**

RX Format: label BCT R,D(X,B)

Decrements the value in R by 1 and branches to D(X,B) if the value in R is not 0. If R contains 0, instructions immediately following BCT is executed.



**Branch on Count Register**

RR Format: label BCTR R1,R2



Decrements the value in R1 by 1 and branches to the address in R2, if the value in R1 is not 0. If R1 contains 0, the instructions following BCTR is executed.



Reference and examples of loops

<https://learn-us-east-1-prod-fleet01-xythos.s3.amazonaws.com/5b06b62e5266a/18526113?response-cache-control=private%2C%20max-age%3D21600&response-content-disposition=inline%3B%20filename%2A%3DUTF-8%27%27360%252013%2520BCT%2520and%2520BCTR%2520Instructions%2520-%2520Notes.pdf&response-content-type=application%2Fpdf&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Date=20200709T150000Z&X-Amz-SignedHeaders=host&X-Amz-Expires=21600&X-Amz-Credential=AKIAZH6WM4PLTYPZRQMY%2F20200709%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Signature=e80bfa8379a36f2319ad6512de91b25c51cc6fc092578d504c89a65c44e1fa7a>

**ASSIST’s X-Type Instruction**

XDUMP

Used to “dump” the GPRs or program storage

Format1: XDUMP , comment

Produces a hexadecimal dump of the GPRs

Format 2: XDUMP D(X,B), length

Produces a hexadecimal dump of storage starting from D(X,B) for length bytes. A label can be substituted for D(X,B), along with an expression that resolves to an integer value can be substituted for length

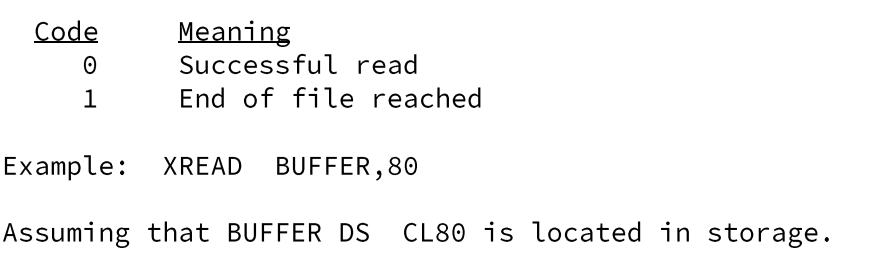
XREAD

Used to read an input record from the file indicated on FT05F001 DD card or from instream data

Format: label XREAD D(X,B),length

Reads length bytes into the input buffer located at D(X,B)

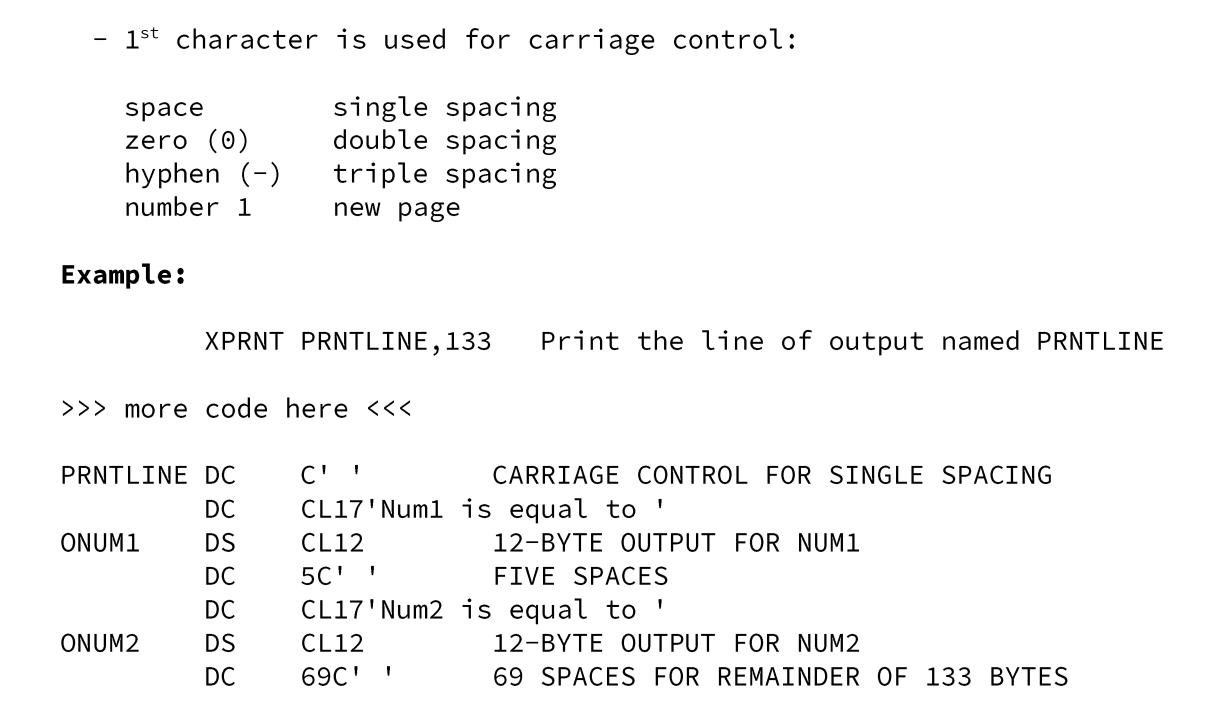
Sets the condition code



**XPRNT**

Used to print a print line of output

Format: label XPRNT D(X,B),length

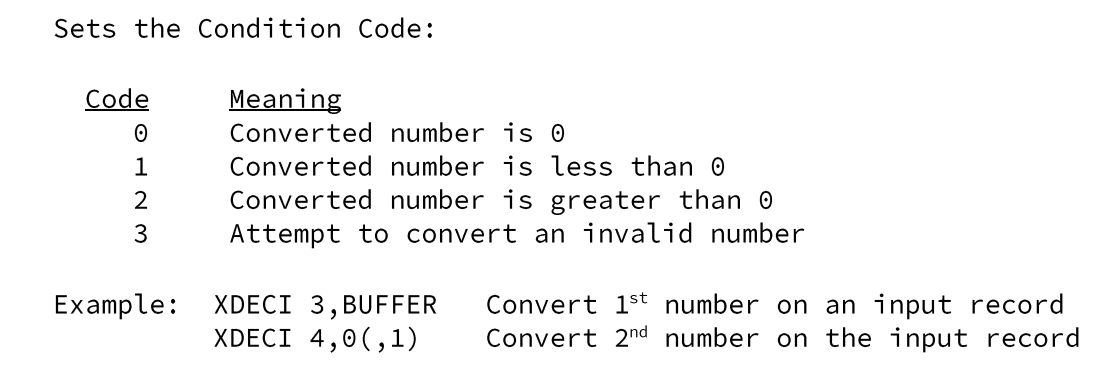
Prints length bytes of the print line defined at D(X,B), the print lines mist be explicitly defined in the program storage (usually 132 characters long)

**XDECI**

Used to convert a number from its character representation to is binary representation so that I t can be used in arithmetic.

Format: label XDECI R,D(X,B)

Converts the number D(X,B) to binary and stores it in R, it stops scanning for digits when a space is reached.

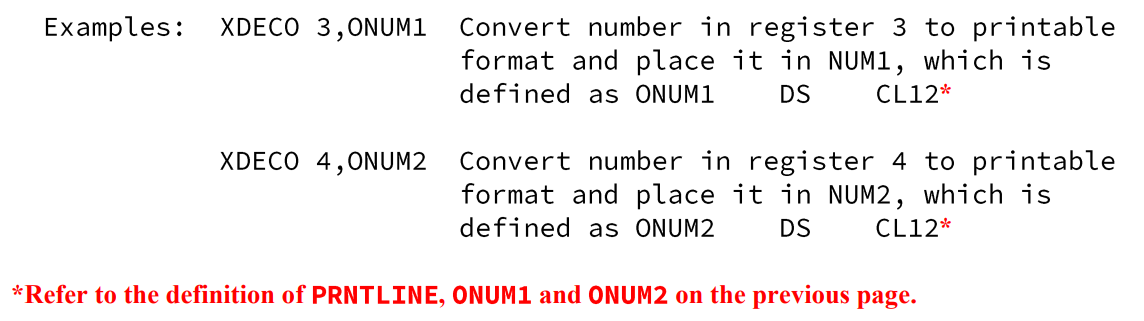


**XDECO**

Used to convert a number from its binary representation to is character representation so it can be printed,

Format: label XDECO R,D(X,B)

Converts the binary number in register R to printable characters and stores it at D(X,B). The number is converted to a 12 byte character representation. If the number is negative, a minus sign is printed to left of 1st digit.



Reference to above

<https://learn-us-east-1-prod-fleet01-xythos.s3.amazonaws.com/5b06b62e5266a/18411006?response-cache-control=private%2C%20max-age%3D21600&response-content-disposition=inline%3B%20filename%2A%3DUTF-8%27%27360%25204.%2520Basic%2520Assembler%2520and%2520ASSIST%2520X-Type%2520Instructions%2520-%2520Notes%25282%2529.pdf&response-content-type=application%2Fpdf&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Date=20200709T000000Z&X-Amz-SignedHeaders=host&X-Amz-Expires=21600&X-Amz-Credential=AKIAZH6WM4PLTYPZRQMY%2F20200709%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Signature=1021f084f96249a14e60b36494bf3eb5647b5fba115a5ae870047180be944820>

**Assembler Literals**

Assembler literals allow you to put a hardcoded value within a program rather than having to use a DC in storage

