**CSCI 360 D(B) and D(X,B) Addressing**

**Another Form of Storage:**

Along with main storage, there are also 16 General Purpose Registers (GPRs).

* Accessed by the computer faster than main storage
* Hold 32 bits
* Numbered from 0 to 15

**Addressing Main Storage:**

In our fake ASSIST world, we pretend that main memory is only 16 MB when, in reality, it is much much bigger. Each and every byte of 16 MB can be referenced by its address in 24-bits (3 bytes). This is called 24-bit addressing.

When ESA, or Enterprise Systems Architecture was developed by IBM, available main memory went up to a whopping 2 GB. Each and every byte of 2 GB can be referenced by its address in 31 bits (almost 4 bytes). This is called 31-bit addressing.

Now, with z/OS and its 64-bit addressing capability, we could theoretically address each and every byte up to 16 EB (1 exabyte = 1,000,000 terabytes!). But, we are really only allowed to reference up to 64 GB.

The giant storage capabilities for main memory allow main memory to be logically partitioned thus allowing the look of many computers on one piece of hardware. This works really well for a corporation that might have their one machine logically partitioned into three partitions. F, for example, a development partition in which the programmers and developers work, a quality assurance testing partition where their software is tested by the users, and, finally, a production partition where the real stuff happens. You will learn more about this in CSCI 465/565 – Enterprise Application Environments.

Every byte of storage has an **absolute address**, ranging from 000000 to FFFFFF. The absolute address is the address of the first byte of an instruction or data item that is referred to.

A slight problem: a programmer does not know where his/her program will be stored in main storage.

To solve the problem: use **base-displacement, relative** or **explicit addressing**. The idea behind this is that the position of two items is fixed. If the absolute address of one is known, then the address of the other statement can be figured out by calculating the displacement between the two items.

Relative address = base address + displacement

The base address is placed in a GPR when a program is executed. The GPR is called a **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. GPR between 0 - 15

4(5) read "4 off of register 5"

36(7)

**How it works:** Convert the displacement to hex and then add it to the LAST three bytes of the address in the register. 1st byte is ignored.

Assume register 7 holds 010234A0, then:

36(7) = 010234A0

+ 24

0234C4

**Special Case:** If the register number is 0, the absolute address 00000000 is used in the calculation.

If register 0 holds 2A763598, then 36(0) results in 000024.

1. **D(X,B)**
   * **D** is the displacement. Decimal number between 0 - 4095
   * **X** is the index register number. GPR between 0 - 15
   * **B** is the base register number. GPR between 0 - 15

4(5,7)

8(9) -- 9 is the index register

2(,3) -- 3 is the base register

Register number defaults to 0 if left off.

Assume register 4 holds 0500029B and register 7 holds 000027A4, then:

3(4,7) = 000027A4

0500029B

+ 3

002A42