## Last Time

## In our previous lab we proved that the small lab computers running RDz and the enterprise-class computers were von Neumann machines because we were able to write a COBOL program, run compile & link programs that translate the human-readable source code to object code and then create a load module. Then we were able to execute the program independent of the hardware. That is we didn’t need a source code machine, then a compile machine, then a linkage editor machine then a separate COBOL executable machine all whose hardware was specifically hard-wired to perform these individual specialized tasks. We also learned that the COBOL code we wrote was able to run because of these fundamental principles of the Von Neumann architecture:

## Memory can hold both computer code (programs) and data

## Each memory location has a unique numeric address

## Memory is addressed linearly i.e. addresses are numbered in sequential order

## Memory is addressed without regard for the contents stored at that address

## Our RDz source code editor, the COBOL compiler, the linkage editor and our COBOL executable program were temporarily stored in and executed from memory as needed which made the computer systems we used general purpose in nature.

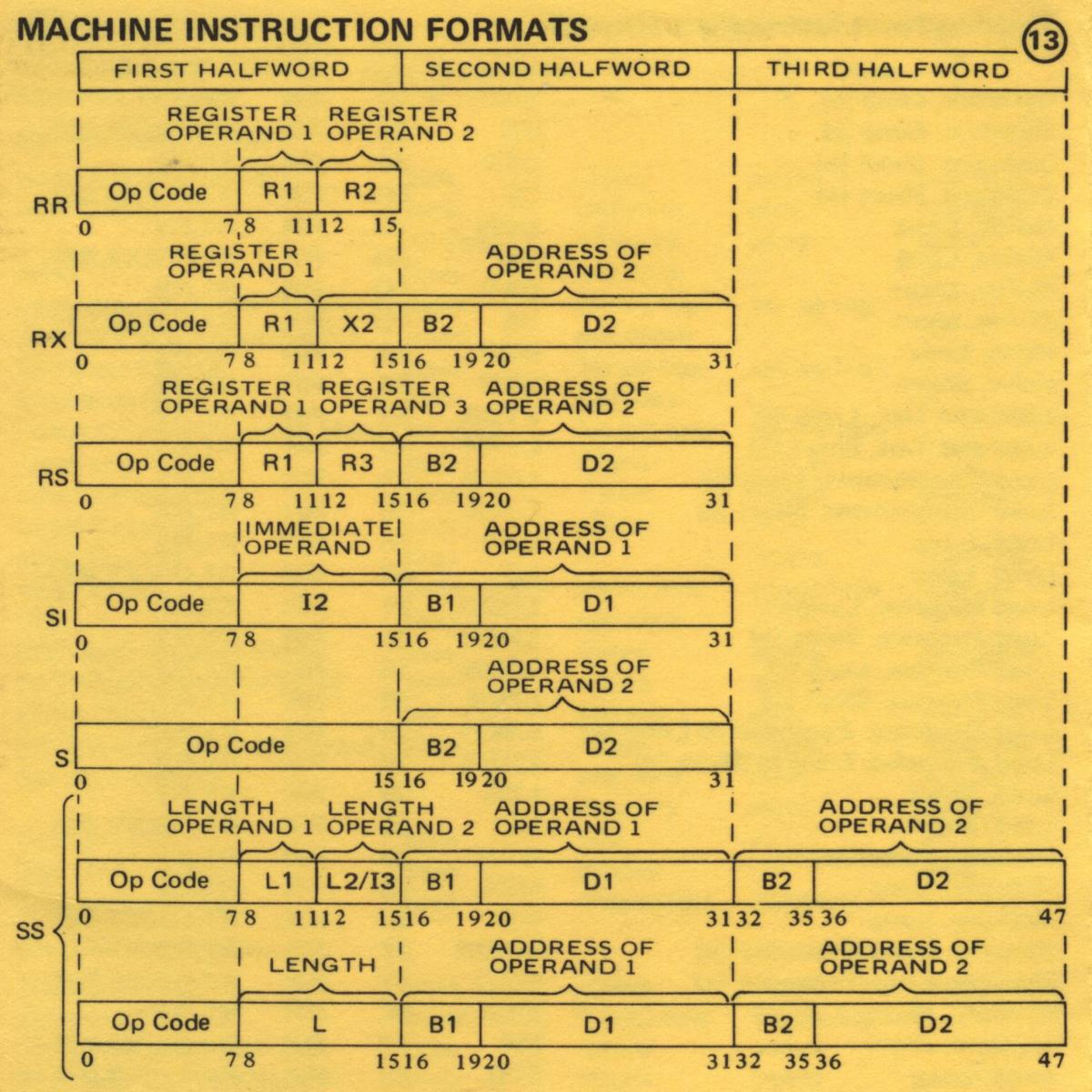
Also, last time, we introduced the concept of **“re-locatable”** programs.Programs must be re-locatable in order to take advantage of the large memory capacities of enterprise computer systems. Our enterprise-class system has 3 terabytes of main memory. The term re-locatable means that the program can run pretty-much anywhere in memory, except in address spaces reserved by the operating system.

## CPU and Memory

In this lab we are going to introduce new computer architecture concepts. Some of the new concepts include the instruction set architecture and the fetch/execution cycle. As you recall from the previous lab, compilers produce object code by translating human readable source code into machine readable code. Before a source code instruction can be executed a number things have to take place. This is where language compilers, like the COBOL compiler come in. Compilers create object code that take advantage of the instruction set architecture. In doing so, the compiler will create code that uses different instruction formats. These instructions can then be executed by the CPU execution units. But the execution unit needs CPU registers to do some its work and needs main memory as a staging area for your program’s instructions. Once translated, each of your program’s instructions require these six steps:

* **Program Load -** your program’s load module or some piece of it must be retrieved from secondary storage then loaded into the computer’s main memory (or RAM)
* **Fetch -** each instruction must be retrieved from memory by the CPU’s Fetch Unit via the fetch portion of the Fetch/Execution cycle and placed in the **Instruction Register**
* **Increment for next address -** **Instruction Address Register** is incremented by the current instruction’s length to calculate the address of the next instruction**.** In the System z this register is called a **Program Status Word** or PSW
* **Decode -** the instruction in the Instruction Register is decoded to determine which circuits to activate
* **Execution -** the decoded instruction is executed by the execution unit to activate the appropriate circuits
* **Write back-** the system clock has a set time period that waits for the bits to settle in i.e. not all bits are turned on or off at the exact same instant

Unlike the Little Man Computer model, the System z has a rich set of complex instructions. Computers that have hundreds of instructions that have complex formats are referred as Complex Instruction Set Computing (CISC) architecture. As you can see from the diagram below, there are seven different instruction formats where there are three different instruction lengths two (2), four (4) to six bytes (6). The COBOL compiler is responsible for making sure your source code is translated correctly to these instruction formats.



**Learning Objectives**

After this lab exercise the student will be able to:

* Describe the concept, operation and interactions of CPU registers
* Describe the operation of memory, and the roles of the memory address register and memory data register
* Describe the Fetch-Execute Instruction cycle
* Explain the purposes for which buses are used within a computer system
* Explain the characteristics of instruction sets and instruction formats

Use RDz and ISPF to create, compile and execute a COBOL program

**Lab Exercise Setup**

Using RDz, sign on to the enterprise system using your mainframe account credentials and verify that you have all of the data sets you need to create, compile and run COBOL programs.

1. We are going to write our next COBOL program that will demonstrate the complexity of the instruction set. We will be adding some new instructions to the Lab 3 you created last week.

As a general rule in programming very few programs are written from scratch. They are built from reusable components called objects. It is easy build new COBOL programs from reusable components. In this lab we will start with last week’s **LAB3**. Find it in your SUSnnnn.GET239.COBOL library. Right click on LAB3 and open up with the COBOL editor and will something like this. No problem if yours looks a little different.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* PURPOSE: This program gives examples:

\* 1. Structure of a COBOL program

\* 2. named memory locations

\* 3. Sequential control

\* 4. Some COBOL reserved words

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ID DIVISION.

PROGRAM-ID. LAB3.

AUTHOR. Dave D.

DATE-WRITTEN. September 2016.

DATE-COMPILED. CURRENT-DATE.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ENVIRONMENT DIVISION.

\* INPUT-OUTPUT SECTION. <to be used later>

\* FILE-CONTROL. <to be used later>

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

DATA DIVISION.

\* FILE SECTION. <to be used later>

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

WORKING-STORAGE SECTION.

\*\*\*\* NAMED MEMORY LOCATIONS \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

01 WS-NAMED-MEMORY-LOCATIONS.

05 MAILBOX-1 PIC X(50) VALUE

'This is an example of PIC X character string'.

05 MAILBOX-2 PIC X(50).

\*\*\*\* EXAMPLE OF REPURPOSING THE SAME MEMORY ADDRESSES \*\*\*\*\*\*\*\*\*\*\*

05 MAILBOX-3 PIC 9(5) VALUE 15.

05 MAILBOX-4 REDEFINES MAILBOX-3 PIC XX.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

PROCEDURE DIVISION.

\*\*\*\* HERE IS WHERE YOUR EXECUTABLE INSTRUCTIONS GO \*\*\*\*\*\*\*\*\*\*\*\*\*\*

000-DISPLAY-MESSAGES.

MOVE 'GET239 Enterprise Technologies' TO MAILBOX-1.

DISPLAY MAILBOX-1

DISPLAY 'THIS IS GET239 LAB 3'

DISPLAY MAILBOX-3.

ADD 45 to MAILBOX-3.

MOVE MAILBOX-3 TO MAILBOX-4.

DISPLAY MAILBOX-4.

display 'Notice the COBOL commands are case-insensitive'

Display ' ... but the data is case-sensitive'.

100-END-PROGRAM.

STOP RUN.

Now create a new member called LAB4, then copy and paste the contents of your working LAB3 program into the new LAB4 member, then save it. Now you have a working program in which to make your changes.

1. In the WORKING-STORAGE SECTION below we will be renaming our memory locations that were previously named MAILBOX-1 through MAILBOX-4. Also, we will be establishing new constraints on these storage areas by changing the PIC clause.

WORKING-STORAGE SECTION.

\*\*\*\* NAMED MEMORY LOCATIONS \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

01 WS-NAMED-MEMORY-LOCATIONS.

05 MAILBOX-1 PIC X(50) VALUE

'This is an example of PIC X character string'.

05 MAILBOX-2 PIC X(50).

\*\*\*\* EXAMPLE OF REPURPOSING THE SAME MEMORY ADDRESSES \*\*\*\*\*\*\*\*\*\*\*

05 MAILBOX-3 PIC 9(5) VALUE 15.

05 MAILBOX-4 REDEFINES MAILBOX-3 PIC XX.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

In your new Lab 4 program change WORKING-STORAGE to look like this:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

WORKING-STORAGE SECTION.

01 WS-NAMED-MEMORY-LOCATIONS.

\*\*\*\* STUDENT NAME and STUDENT FIRST NAME share the same memory

\*\*\*\* address. The same is true for STUDENT BIRTH DATE and

\*\*\*\* STUDENT BIRTH MM

**The data names with higher numbers are subsets of the lower numbers. For example, the entire STUDENT NAME can be referenced by itself or each part of the STUDENT NAME can be referenced individually**

\*\*\*\* Can you explain why this is true?

05 STUDENT-NAME.

10 STUDENT-FIRST-NAME PIC X(10).

10 STUDENT-MIDDLE-INIT PIC X.

10 STUDENT-LAST-NAME PIC X(20).

05 STUDENT-BIRTH-DATE.

10 STUDENT-BIRTH-MM PIC 99.

10 PIC x VALUE '/'.

10 STUDENT-BIRTH-DD PIC 99.

10 PIC X VALUE '/'.

10 STUDENT-BIRTH-YY PIC 9(4).

\*\*\*\* The V in the PIC clause allows you to keep track of the

\*\*\*\* decimal point alignment

05 STUDENT-GPA PIC 9V999 VALUE 0.

05 STUDENT-GPA2 PIC 9V999 VALUE 0.

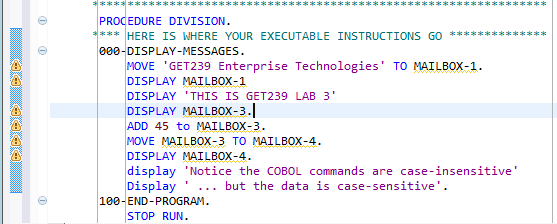
\*\*\*\* The Z is an edit mask character to suppressing leading 0's

05 STUDENT-GPA-Z PIC Z.999.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Here you can see that STUDENT-NAME references a memory location consisting of 31 characters (or bytes) and contains the student first name (first 10 characters), middle initial (1 character) and last name (20 characters.) STUDENT-BIRTH-DATE works the same way. Here you can reference each part of the date (month, day, and year) separately. Lastly you can define a storage location as numeric and redefine it with an edit mask so that it can be displayed with special characters like dollar signs, decimal points or with zero suppression.

When you change the data names to the ones indicated above you’ll notice that the RDz COBOL editor will give you errors in the PROCEDURE DIVISION because the data names: MAILBOX-1 thru MAILBOX-4 are no longer defined in WORKING-STORAGE. This isn’t a problem because we will be making changes to those instructions next.



1. In the PROCEDURE DIVISION we will be making some changes that will affect the new data names we just created.

First add the displays for the “Splash Page” that identifies the course, lab number, your name. Surround this name data with rows of asterisks. Notice I’m also using a paragraph name that describe “what” the instruction in the paragraph do.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

PROCEDURE DIVISION.

\*\*\*\* The Splash Page identifies your program output to me \*\*\*\*\*\*\*

000-DISPLAY-SPLASH-PAGE.

DISPLAY '\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*'.

DISPLAY '\* GET239 Enterprise Technologies LAB4 \*'.

DISPLAY '\* <your name here> \*'.

DISPLAY '\* <the date> \*'.

DISPLAY '\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*'.

1. Let’s add some new COBOL instructions. In this lab well add four new instructions: **COMPUTE, DIVIDE, MULTIPLY and SUBTRACT**. See if you can build the following program. Compile it then execute it. You have all of the JCL you need to compile and execute it **but be sure to change the JCL to point to your LAB4 in your COBOL and LOADLIB libraries.**

100-FORMAT-STUDENT-NAME.

\*\*\*\* Put your name here - notice character strings are in quotes

MOVE 'David' TO STUDENT-FIRST-NAME.

MOVE 'E' TO STUDENT-MIDDLE-INIT.

MOVE 'Dischiave' TO STUDENT-LAST-NAME.

200-FORMAT-STUDENT-NAME.

\*\*\*\* Put your birth date here - notice your numeric literals do

\*\*\*\* not need to be delineated by quotes

MOVE 02 TO STUDENT-BIRTH-MM.

MOVE 14 TO STUDENT-BIRTH-DD.

MOVE 2016 TO STUDENT-BIRTH-YY.

300-CALCULATE-GPA.

\*\*\*\* Calculate GPA here - use arithmetic instructions \*\*\*\*\*\*\*\*\*\*\*

ADD 1 TO STUDENT-GPA.

DIVIDE STUDENT-GPA BY 2 GIVING STUDENT-GPA.

MULTIPLY STUDENT-GPA BY 7 GIVING STUDENT-GPA.

SUBTRACT 3 FROM STUDENT-GPA.

\*\*\*\* The same calculations can be accomplished by the

\*\*\*\* COMPUTE statement as follows:

COMPUTE STUDENT-GPA2 = ((((STUDENT-GPA2 + 1) / 2) \* 7) - 3).

400-DISPLAY-RESULTS.

\*\*\*\* Move your result to an edit mask field for display purposes

MOVE STUDENT-GPA TO STUDENT-GPA-Z.

\*\*\*\* Display results - notice character strings are in quotes \*\*\*

DISPLAY 'Student Name: ' STUDENT-NAME.

DISPLAY 'Birth Date : ' STUDENT-BIRTH-DATE.

DISPLAY 'GPA : ' STUDENT-GPA-Z.

MOVE STUDENT-GPA2 TO STUDENT-GPA-Z.

DISPLAY 'GPA2 : ' STUDENT-GPA-Z.

DISPLAY '\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*'.

\*\*\*\* End your program \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

999-END-PROGRAM.

STOP RUN.

\*\*\*\* End of Lab 4 Program \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Quick Instruction Summary**

In COBOL you can perform calculations on numeric fields only. That is, you can only perform arithmetic operations on only data that you have described as PIC 9. The maximum number of digits in a numeric field is 18. So your picture clause would look like PIC 9(18) and the largest (non-floating point) integer number you can store in memory is 999,999,999,999,999,999. Historically this hasn’t been a problem for business applications.

**ADD –** use the ADD instruction to create the sum. The ADD has a number of formats. You can add literals, figurative constants, or data fields defined in input records or working storage. You can add a number of values at the same time. Also, the ADD statement can have a GIVING clause.

ADD STUDENT-GRADE TO STUDENT-GPA.

ADD 1 TO STUDENT-GPA.

ADD 1, STUDENT-GRADE TO STUDENT-GPA.

ADD 1, 2, 3 TO STUDENT-GPA.

ADD STUDENT-GRADE TO STUDENT-GPA GIVING STUDENT-GPA2.

**SUBTRACT** – use subtract to reduce a field value. The SUBTRACT shares the same formats as the ADD instruction, except you use FROM instead of TO. See examples:

SUBTRACT STUDENT-GRADE FROM STUDENT-GPA.

SUBTRACT 1 FROM STUDENT-GPA.

SUBTRACT 1, STUDENT-GRADE FROM STUDENT-GPA.

SUBTRACT 1, 2, 3 FROM STUDENT-GPA.

SUBTRACT STUDENT-GRADE FROM STUDENT-GPA GIVING STUDENT-GPA2.

**MULTIPLY** – use to multiply values. You can also round your result by using the ROUNDED verb. See example:

MULTIPLY STUDENT-GPA BY 7 GIVING STUDENT-GPA.

MULTIPLY STUDENT-GPA BY 7 GIVING STUDENT-GPA ROUNDED.

MULTIPLY STUDENT-GPA BY NUMBER-OF-COURSES GIVING STUDENT-COURSES.

**DIVIDE** – use to divide values. You can also round your result by using the ROUNDED verb. See examples:

DIVIDE STUDENT-GPA BY 2 GIVING STUDENT-GPA.

DIVIDE STUDENT-GPA BY 2 GIVING STUDENT-GPA ROUNDED.

DIVIDE STUDENT-GPA BY NUMBER-COURSES GIVING STUDENT-GPA.

**COMPUTE** – this is a shorthand method for accomplishing more complex calculations. I recommend using parenthesis liberally to keep the order of precedence so that you get the desired result. The expressions in the parenthesis (inner to outer get resolved first.) You can also use the ROUNDED verb. See examples:

COMPUTE STUDENT-GPA2 = ((((STUDENT-GPA2 + 1) / 2) \* 7) - 3).

COMPUTE STUDENT-GPA2 ROUNDED = ((((STUDENT-GPA2 + 1) / 2) \* 7) - 3).

**After you execute the LAB4 you should get results like this:**

J E S 2 J O B L O G -- S Y S T E M S Y S 1 -- N O D E N 1

21.57.03 JOB06327 ---- WEDNESDAY, 19 SEP 2012 ----

21.57.03 JOB06327 IRR010I USERID SUS0000 IS ASSIGNED TO THIS JOB.

21.57.03 JOB06327 ICH70001I SUS0000 LAST ACCESS AT 21:55:50 ON WEDNESDAY, SEPTEMBER 19, 2012

21.57.03 JOB06327 $HASP373 SUS0000T STARTED - INIT 1 - CLASS A - SYS SYS1

21.57.03 JOB06327 $HASP395 SUS0000T ENDED

------ JES2 JOB STATISTICS ------

19 SEP 2012 JOB EXECUTION DATE

15 CARDS READ

54 SYSOUT PRINT RECORDS

0 SYSOUT PUNCH RECORDS

3 SYSOUT SPOOL KBYTES

0.00 MINUTES EXECUTION TIME

1 //SUS0000T JOB (000000),'Dave D', JOB06327

// NOTIFY=&SYSUID,

// CLASS=A,

// MSGCLASS=X,

// REGION=4096K

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//\*STEP TO EXECUTE COBOL PROGRAM \*/

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

IEFC653I SUBSTITUTION JCL - (000000),'Dave D',NOTIFY=SUS0000,CLASS=A,MSGCLASS=X,REGION=4096K

2 //STEP01 EXEC PGM=LAB4

3 //STEPLIB DD DSN=&SYSUID..GET239.LOADLIB,

// DISP=SHR

IEFC653I SUBSTITUTION JCL - DSN=SUS0000.GET239.LOADLIB,DISP=SHR

4 //SYSPRINT DD SYSOUT=\*

5 //SYSOUT DD SYSOUT=\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

6 //

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Here is a sample of your LAB4 output. Notice that the leading zeroes for the GPA have been suppressed.**

**Your results should look similar to this.**

\* GET239 Enterprise Technologies LAB4 \*

\* <your name here> \*

\* <the date> \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Student Name: David EDischiave

Birth Date : 10/31/2011

GPA : .500

GPA2 : .500

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Type your answers and attach your screen prints.**

1. **Once a program has been made machine readable i.e. compiled and linked, list and explain the six (6) steps that take place in order to execute a program instruction? (6)**

a. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

b. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

c. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

d. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

e. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

f. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. **Describe the purpose of the four COBOL Divisions (4)**

a.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

b.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

c. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

d. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. **Write a COBOL program named LAB4A to do the following. (10 points)**

**Be sure your program is well-documented according to our programming standards**

1. Display splash page that has the following items:

* GET239 Enterprise Technologies LAB4A
* Your name (First & Last)
* Today’s Date
* Surround the splash page with asterisks

1. Define the following memory locations in the WORKING-STORAGE SECTION:

EMPLOYEE-NAME

EMP-FIRST-NAME, alphanumeric, length 20

EMP-LAST-NAME, alphanumeric, length 30

HIRED-DATE.

HIRED-MM, numeric, length 2

HIRED-DD, numeric, length 2

HIRED-YY, numeric, length 4

SERVICE-YEARS, numeric, length 2

ANNUAL-SALARY, numeric, length 9 (7 with 2 decimal positions)

BI-WEEKLY-GROSS-PAY, numeric, length 9 (7 with 2 decimal positions)

------------------------------------------------------------------------------------------------------------

1. Move your first name to EMP-FIRST-NAME
2. Move your last name to EMP-LAST-NAME
3. Move 01011996 to DATE-HIRED
4. Calculate number of years of service. Use the current date function to get the **current year** **(research to find the format and syntax of this function. Hint: google COBOL Date function.)**
5. Move 79000.13 to ANNUAL-SALARY
6. Divide ANNUAL-SALARY by 26 pay periods and save the result in BI-WEEKLY-GROSS-PAY. **Round your result.**
7. Display the employee first and last name, current year, year hired, service years, annual salary, bi-weekly-gross-pay **(be sure to suppress leading zeroes on numeric fields. Hint: use a Z in your PIC clause)**
8. Print your LAB4A source code and the output from JES to prove that your program execution works. See next page for an example of the LAB4A execution.

**Your LAB4A program execution should look likes something like this.**

A

J E S 2 J O B L O G -- S Y S T E M S Y S 1 -- N O D E N 1

23.19.46 JOB06334 ---- WEDNESDAY, 19 SEP 2012 ----

23.19.46 JOB06334 IRR010I USERID SUS0000 IS ASSIGNED TO THIS JOB.

23.19.46 JOB06334 ICH70001I SUS0000 LAST ACCESS AT 23:19:33 ON WEDNESDAY, SEPTEMBER 19, 2012

23.19.46 JOB06334 $HASP373 SUS0000T STARTED - INIT 1 - CLASS A - SYS SYS1

23.19.46 JOB06334 $HASP395 SUS0000T ENDED

------ JES2 JOB STATISTICS ------

19 SEP 2012 JOB EXECUTION DATE

15 CARDS READ

59 SYSOUT PRINT RECORDS

0 SYSOUT PUNCH RECORDS

3 SYSOUT SPOOL KBYTES

0.00 MINUTES EXECUTION TIME

1 //SUS0000T JOB (000000),'Dave D', JOB06334

// NOTIFY=&SYSUID,

// CLASS=A,

// MSGCLASS=X,

// REGION=4096K

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//\*STEP TO EXECUTE COBOL PROGRAM \*/

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**Format your output to look like the sample below. Notice that the leading zeroes for the Annual Salary and Bi-Weekly Amount have been suppressed.**

**Keep in mind your actual results will be different than the ones below so you need to verify that your program calculated the years of service and the biweekly amount correctly.**

**Just like in LAB3LMC you need to submit your fully-documented COBOL source code and your program’s output indicated by A, B and C**

IEFC653I SUBSTITUTION JCL - (000000),'Dave D',NOTIFY=SUS0000,CLASS=A,MSGCLASS=X,REGION=4096K

2 //STEP01 EXEC PGM=LAB4A

3 //STEPLIB DD DSN=&SYSUID..GET239.LOADLIB,

// DISP=SHR

IEFC653I SUBSTITUTION JCL - DSN=SUS0000.GET239.LOADLIB,DISP=SHR

4 //SYSPRINT DD SYSOUT=\*

5 //SYSOUT DD SYSOUT=\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

6 //

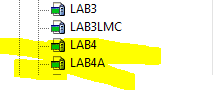
ICH70001I SUS0000 LAST ACCESS AT 23:19:33 ON TUESDAY, February 4, 2014

B

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* GET239 Enterprise Technologies LAB4A \*

\* <your name> \*

\* <the current date> \*

C

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Employee First Name: David

Employee Last Name: Dischiave

Current Year : 2016

Year Hired : 1996

Years of Service : 20

Annual Salary : 79,000.13

Bi-weekly Amount : 3,038.47

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* End of LAB4A \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*