Introduction to Assembler Programming Sessions 17690, 17691

Richard Cebula (riccebu@uk.ibm.com) IBM HLASM

Who am I?

- Richard Cebula – HLASM, IBM Hursley, UK
- riccebu@uk.ibm.com

- Develop and support the following products:
  - HLASM
  - SuperC
  - XREF
  - IDF
  - DISASM
  - Structured Programming Macros

Audience

- This course aims to provide the grounding knowledge for programming in assembler
- The audience should have a basic understanding of computers
- The audience should be new to the world of z Systems Assembler programming
- At the end of this course the attendee should be able to:
  - Understand the basics of assembler programming on z Systems
  - Understand a variety of simple machine instructions
  - Understand how to Assemble, Bind and run simple assembler programs

Introduction to Assembler Programming

- Why program in assembler?
- Computer Organisation – Overview of z/Architecture
- Assemblers, Compilers, Binders – Building programs on z Systems
- Working with the High Level Assembler (HLASM)
  - Using HLASM to assemble your program
  - Syntax
  - Machine vs Assembler instructions
- Programming in Assembler
  - Moving data around
    - Loading, storing and moving data
  - Manipulating Data
    - Logical operations
    - Arithmetic
  - Making Decisions
    - Comparing
    - Branching

Introduction to Assembler Programming

- Programming in Assembler
  - Forming High Level Language constructs
    - If...then...else
    - Looping
  - Addressing Data and some of its subtleties
  - Basics of Calling Conventions
  - Reading Principles of Operation
- Reading the HLASM listing
- The PSW and an introduction to Debugging Assembler Programs

Why program in assembler?

Why program in assembler?

- Assembler programming has been around since the very start of computer languages as an easy way to understand and work directly with machine code
- Assembler programming can produce the most efficient code possible
  - Memory is cheap
  - Chips are fast
  - So what?
- Assembler programming TRUSTS the programmer
  - Humans are smart (?)
  - Compilers are dumb (?)
- Assembler programming requires some skill
  - No more than learning the complex syntax of any high-level language, APIs (that change every few years), latest programming trends and fashions
  - Your favorite language will too become old, bloated and obsolete!

Why program in assembler?

- Misconceptions of assembler programming
  - I need a beard right?
  - It's too hard...
  - Any modern compiler can produce code that's just as efficient now days...
  - I can do that quicker using...
  - But assembler isn't portable...

Why program in assembler?

- Misconceptions of assembler programming
  - I need a beard right?
    - Assembler programmers tend to be old*er* and more experienced and typically wiser
    - Experienced programmers that have used assembler know that they can rely on it for the most complex of programming tasks
  - It's too hard...
    - Learning assembler is just like learning any other language
    - Each instruction to learn is as easy as the next
    - Syntax is consistent
    - No difficult APIs to get to grips with
  - Any modern compiler can produce code that's just as efficient now days...
    - Compilers CAN produce efficient code but that is not to say that they WILL
    - Optimization in compilers is a double-edged sword – compilers make mistakes
  - I can do that quicker using...
    - Good for you, so can I...
  - But assembler isn't portable...
    - Neither is Java, nor C, nor C++... portability depends on your definition of it

Why program in assembler?

- The assembler mindset
  - You are not writing code – you are programming the machine
  - You must be precise
  - Your assembler program is no better than your programming
- Assembler programming provides the programmer with TOTAL freedom
  - What you choose to do with that freedom is your choice and your responsibility
- WYWIWYG – What you write is what you get – even if you *think* you wrote something else...

Why program in assembler? - I thought this was the 21st century...

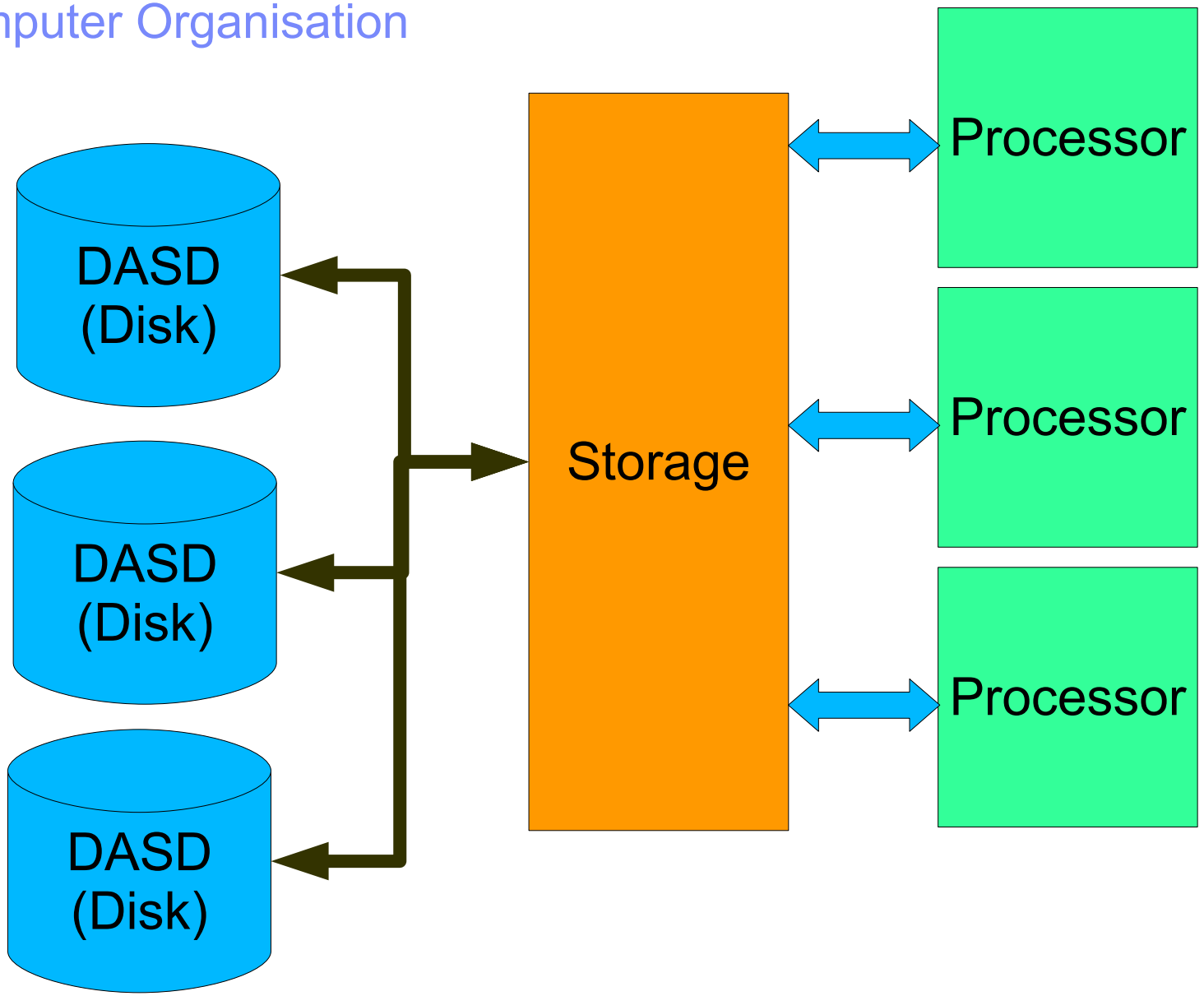
- Well there are a number of good reasons to program in assembler still:
  - Precise machine control – trust me...your compiler doesn't do what you think it does...
  - Precise definition of data – we have more data types than other languages
  - Self-writing code – the world's best macro facility
  - When someone's program goes wrong – finding the cause of the problem is much easier in assembler
  - Assembler is high-tech – no need to wait for compilers to catch up to what your chip can do

Computer Organisation

Computer Organisation

- The main parts of a computer when discussing programming are:
  - Processor – z Systems is a multiprocessor computer
  - Storage – (don't call it RAM on an EC12 and later it's RAIM!!)
  - Disks (more often referred to as DASD in z Systems)
- Programs are stored on disk since disks are non-volatile media, i.e. they do not loose their contents when the computer is not running
- A program is fetched from disk and placed into storage from where it can be executed by the processor
- The processor is the brain of the computer and is responsible for actually executing programs
- Operations inside a computer such as loading programs from disk are performed by a piece of software called an Operating System (OS)
- z Systems have 5 operating systems available z/OS, z/VM, z/VSE, z/TPF and z/Linux

Computer Organisation



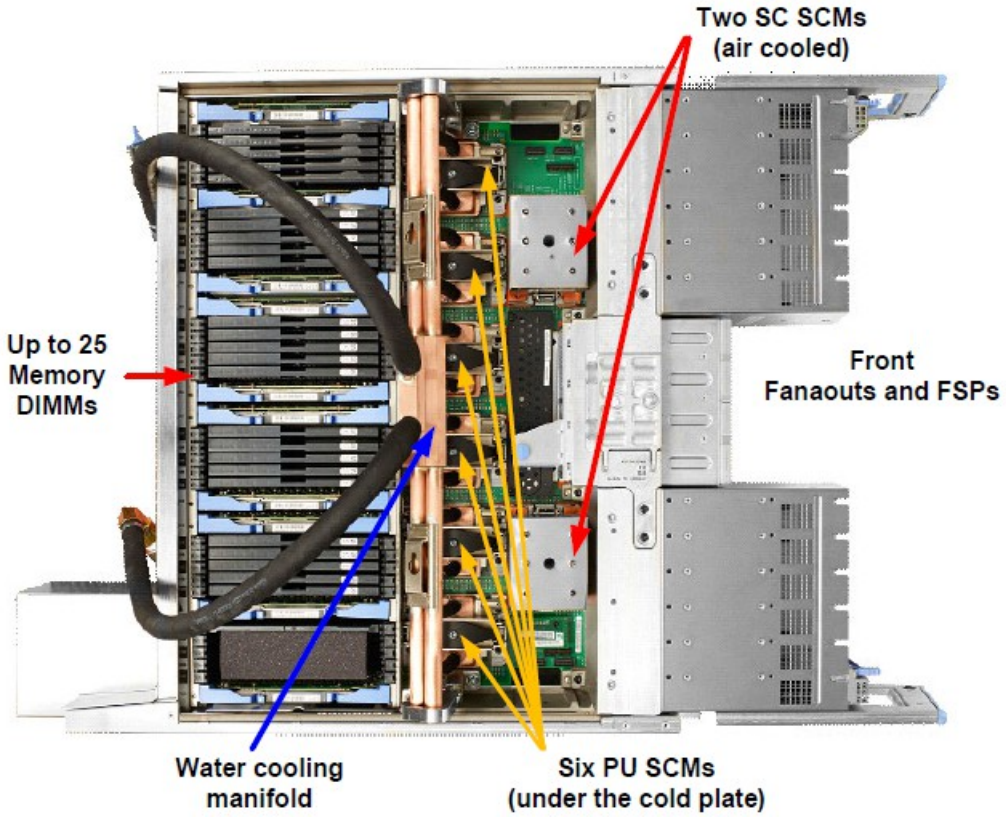
Computer Organisation

- The closer that data gets to the processor, the quicker it is to access the data both in terms of type of storage used and also due to the physical distance to the processor
- In order to improve the speed at which the program is accessed from storage, a special form of storage called *cache* is built into the processor
- Data and instructions are fetched from storage into the cache on the processor and then accessed from there
- Depending on how often the data requires to be accessed, how much data is to be accessed and whether the data needs to be shared between different processors or not, depends on which *cache level* the data is placed into on the processor with level 1 being the smallest and fastest and level 4 being the slowest but largest
- The implementation of cache is dependent on not only the processor architecture, e.g. Intel vs z/Architecture vs ARM etc. but also the model of processor itself, e.g. EC12 cache structure is different to z196

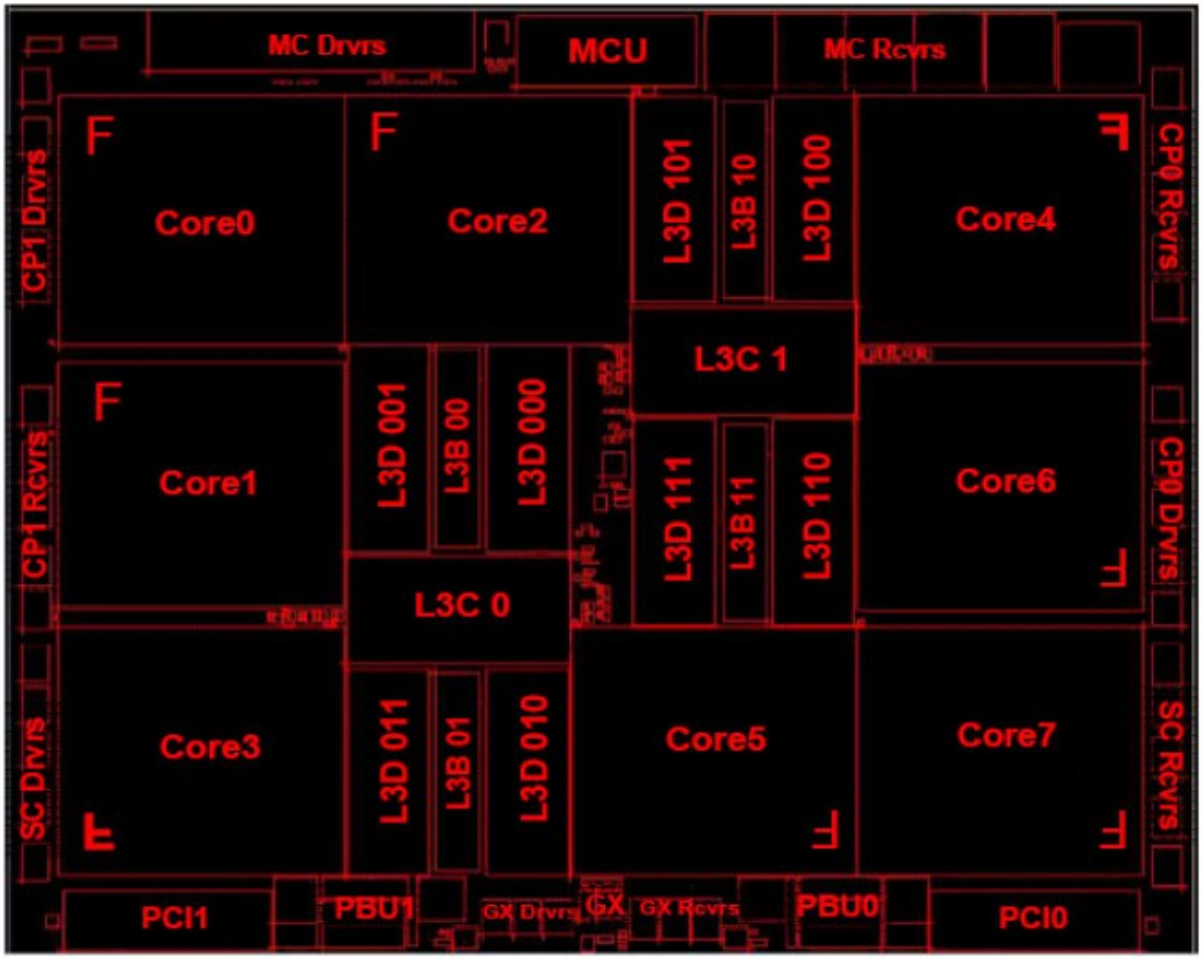
Computer Organisation – z13

- Each processor in z Systems is split into a number of *processor cores* called PUs (processing units)
- Each PU (core) has its own L1 (96KB I, 128KB D) and L2 (2MB I, 2MB D) instruction and data caches
- Each processor chip contains up to 8 processing cores which share a 64MB L3 cache
- PU chips and storage control (SC) chips are packaged as single chip modules (SCMs). 3 PU chips and 1 SC chip form a CPC drawer node. Each SC has a 480MB L4 cache.
- 2 nodes are contained on a CPC drawer. Each system can have up to 4 drawers which contain all the storage for the system.
- All of the highly complex cache structure is transparent to the programmer
  - Experienced programmers will often change their code in order to improve cache performance
- Depending on how the machine is configured, each processor may run a different level of *microcode* which governs how it functions changing the PU into either a CP, zIIP or IFL
- Additionally, z Systems has specialist I/O processors called System Assist Processors (SAP) used to send signals to devices attached to the machine

Computer Organisation – z/Architecture



Computer Organisation – z/Architecture



Computer Organisation – z/Architecture

- In order for data to be manipulated by the processor, it needs to be loaded into a *register*
- A register is the fastest storage available to the processor and is located inside each processor core
- In z/Architecture, each processor core has:
  - 16 64-bit General Purpose Registers (GPRs)
  - 16 32-bit Access Registers (ARs)
  - 16 64-bit Floating Point Registers (FPRs)*
  - 16 64-bit Control Registers (CRs)
  - 32 128-bit Vector Registers (VRs)*
  - 1 Program Status Word (PSW)
- Note that all registers are numbered 0-15 (or 0-31) in machine instructions – the instruction itself determines which type of register is being used
- z/Architecture – the processor architecture used for all z Systems Mainframes
- Processor specifications vary
  - Processor level – the physical (or virtual) chip used
  - Architecture level – the instruction specification of a chip
- * Vector registers 0-15 bits 0-63 are mapped over the FPRs.

Computer Organisation – z/Architecture

- z/Architecture is a big-endian, 64-bit, rich CISC processor architecture
  - Big-endian
    - Data is organised such that the most significant byte of a piece of data is stored in the lowest address of memory
  - 64-bit
    - The size of general purpose registers is 64-bits in length
  - CISC
    - Complex Instruction Set Computer
    - A single instruction comprises a number of micro-instructions which are executed by the processor
    - This scheme allows for a single machine instruction to perform a number of complex tasks
- For historical reasons, the size of data in z/Architecture is measured as:
  - 4 bits = 1 nibble
  - 8 bits = 2 nibbles = 1 byte
  - 16 bits = 2 bytes = a *halfword*
  - 32 bits = 4 bytes = 2 halfwords = a *word*
  - 64 bits = 8 bytes = 2 words = a *doubleword*
  - 128 bits = 16 bytes = 2 doublewords = a *quadword*

Computer Organisation – Understanding Registers

- GPRs – used for arithmetic, logical operations, passing operands to instructions, calling subroutines etc
- ARs – used in “Access Register” mode – provides the ability to access another address space
- FPRs – used for floating point instructions, binary, decimal and hexadecimal floating-point arithmetic
  - Do not confuse decimal floating-point with packed decimal arithmetic – the latter is performed in storage not in registers
- VRs – used for SIMD (Single Instruction Multiple Data) operations including integer, string, floating-point and general operations
- CRs – used for controlling processor operations
- PSW – provides the status of the processor consisting of 2 parts:
  - PSW Flags – these show the state of the processor during instruction execution
  - Instruction address – this is the address of the next instruction to be executed
- GPRs and FPRs are sometimes operated on in pairs by certain instructions
  - GPRs form even-odd pairs, i.e. (0,1), (2,3),...,(14,15)
  - FPRs pair evenly / oddly, i.e. (0,2), (1,3),...,(13,15)

Assemblers, Compilers and Binders
Building programs on z Systems

Assemblers, Compilers and Binders

- Typing in the code for a computer program will not mean that the computer can load and then execute the program
- The written code must be changed into binary form – this is done for the programmer by a compiler or assembler
  - Compilers are used for HLLs and often attempt to optimise the code written by the programmer
  - Assemblers are used for assembly language and do NOT optimise any code written by the programmer – assembler is a WYWIWYG language – What You Write Is What You Get
- Each language will require its own compiler / assembler to process it and change it into binary code
- *Cross compiling* is where a compiler runs on one machine architecture and produces machine code for another architecture

Assemblers, Compilers and Binders

- The machine code produced by the assembler is called *object code*
- It is the job of a *binder* (sometimes called a *linker*) to create a complete program which can then be loaded by the operating system's *loader* into storage and then executed by the processor
- The binder works by ordering a set of objects and resolving any references between them
  - The bound object on z/OS is called a *program object* (for GOFF format) or a *load module* (for OBJ format)
  - Program objects and load modules vary in their capability – it is recommended that new programs use the GOFF format
- Some references are unable to be resolved by the binder at bind time and can only be resolved when the program is loaded into storage by the operating system's loader
  - In order to do this, the binder creates a list of these references and notes their location in the program object / load module
  - Examples of such unresolvable references are various operating system services and shared libraries
- The terms “load module” and “program object” are used interchangeably in this presentation

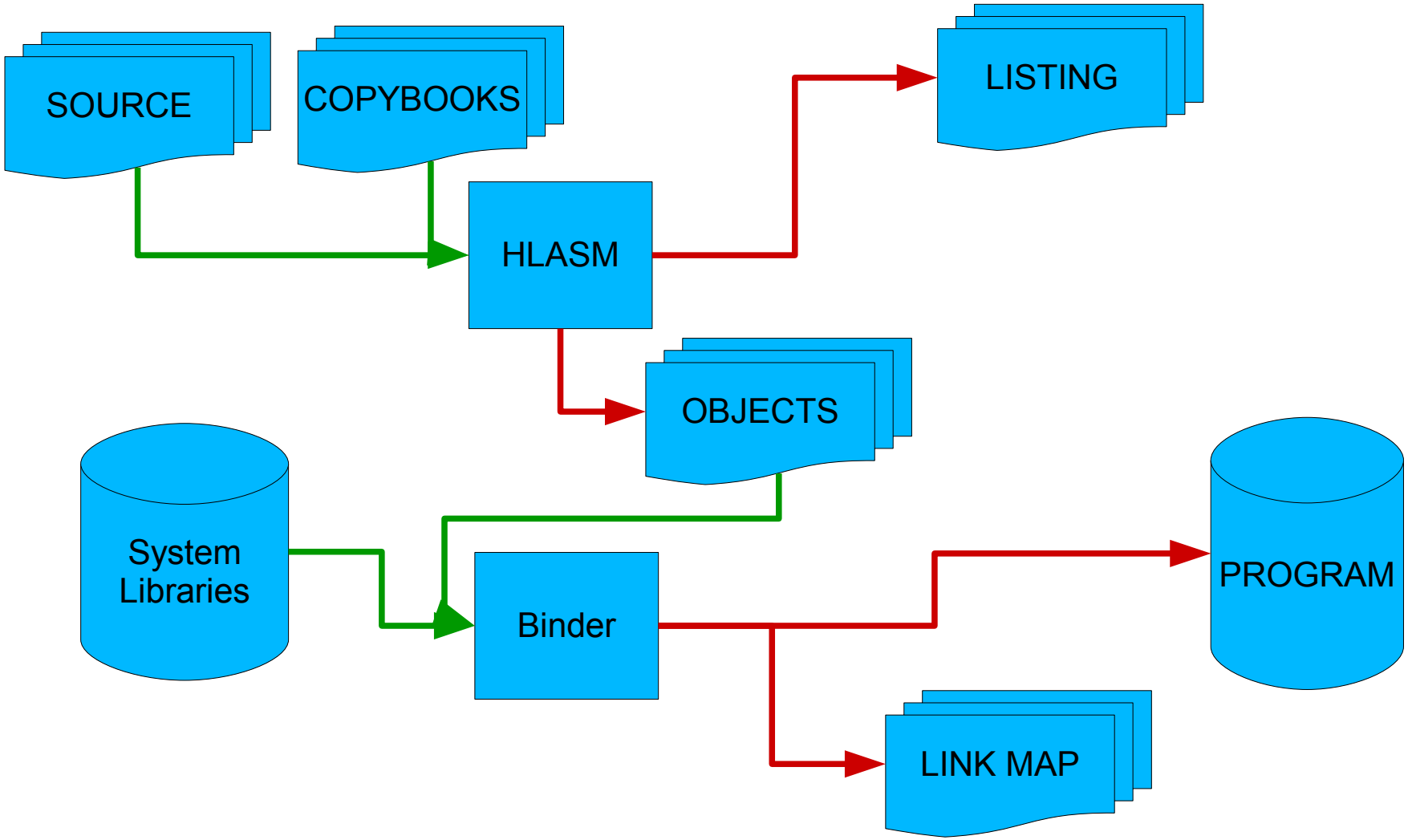
Assemblers, Compilers and Binders – Working with HLASM

- HLASM – IBM's High Level Assembler
- Available on z/OS, z/VM, z/VSE, z/Linux and z/TPF
- *High Level Assembler???* - YES!
  - Provides a wide range of assembler *directives*
    - An assembler *directive* is not a machine instruction
    - It is an instruction to the assembler during assembly of your program
  - A very powerful macro programming facility
  - Structured programming

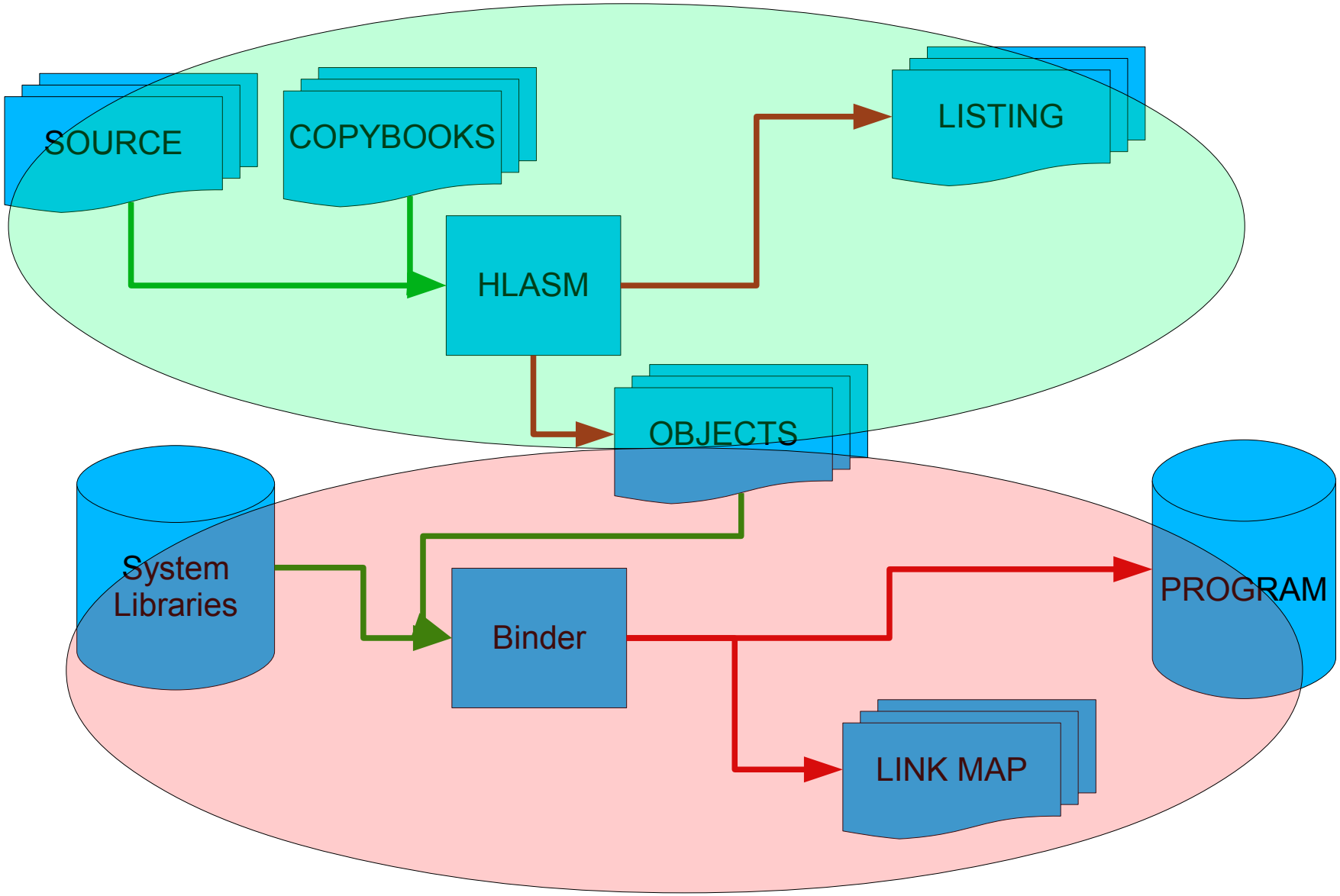
Assemblers, Compilers and Binders – Working with HLASM

- HLASM produces 2 primary outputs
  - OBJECT DECKS – this is the object code that is used as input to binding
  - Listing – this shows any errors, all diagnostics and human readable output from the assemble phase
- The binder produces 2 primary outputs
  - LOAD MODULE – this is the bound object decks forming an executable program
  - A LOAD MAP – this is the Binder equivalent of an assembler listing
- A LOAD MODULE can be loaded into memory by the operating system and run

Assemblers, Compilers and Binders – Working with HLASM



Assemblers, Compilers and Binders – Working with HLASM



Working with the High Level Assembler HLASM

Working with the High Level Assembler HLASM

- These slides explain about using HLASM specifically for z/OS
  - The same principles apply on other platforms although HLASM is started differently, e.g. different command lines on z/VM and z/Linux, VSE JCL for z/VSE

- HLASM is started on z/OS via JCL and is shipped with some JCL PROCs to make using HLASM easier
  - ASMAC – Assembles a program
  - ASMACG – Assembles a program and invokes the loader to bind, load and execute the program (no load module is retained)
  - ASMACL – Assembles and invokes the binder to bind the program producing a load module
  - ASMACLG – Assembles and binds the program then runs the produced load module

Working with the High Level Assembler HLASM

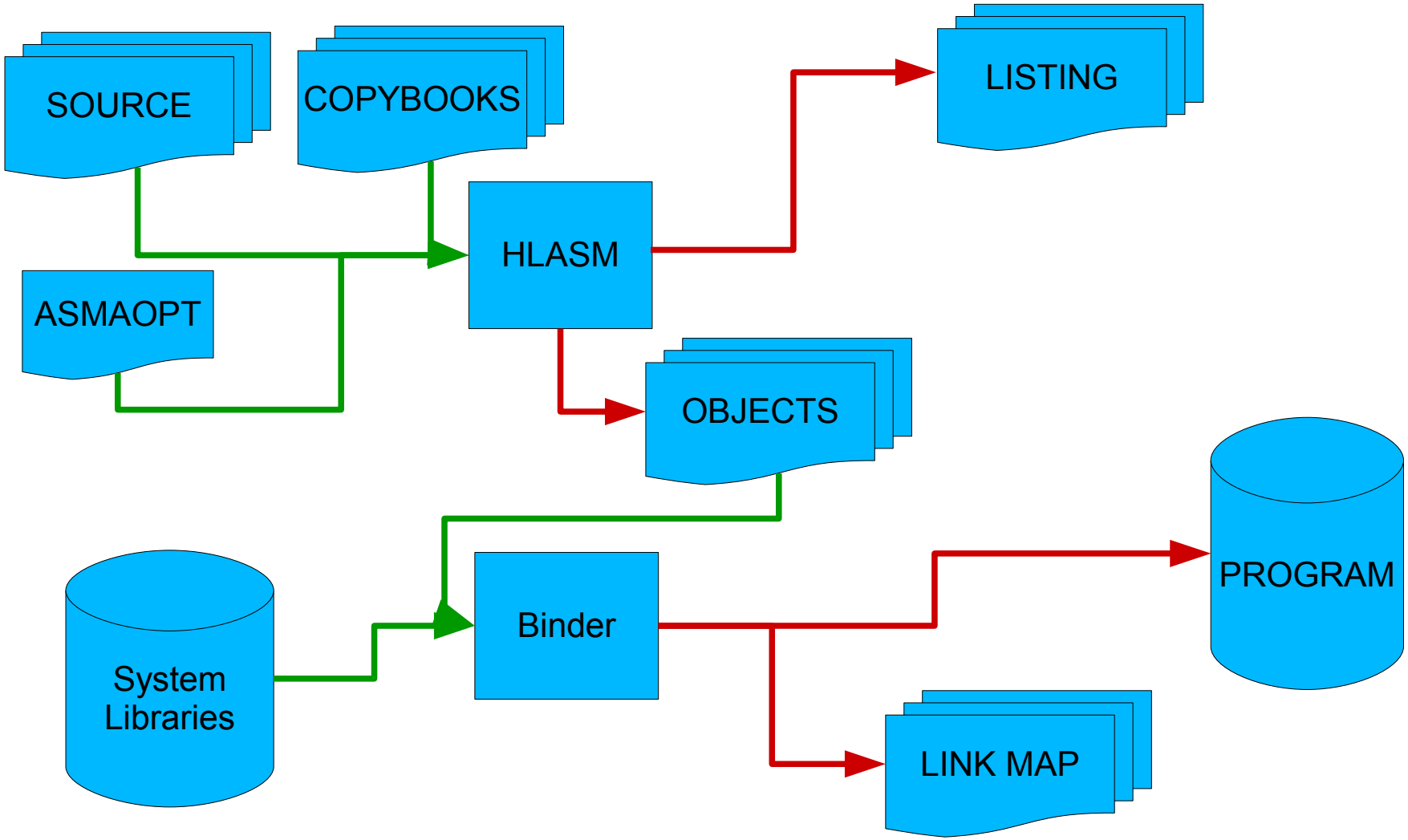
- The following JCL invokes ASMACL to assemble and bind the program into a load module:

```
// SET OPTLIB=&SYSUID..AOPT
// SET SRCLIB=&SYSUID..SOURCE
// SET LSTING=&SYSUID..LISTINGS
// SET LODLIB=&SYSUID..LOAD
//*
// SET PRGNM=MYPROG
//*
//ASMMSAMP EXEC ASMACL,PARM.C=(OBJ,ADATA)
//C.ASMAOPT  DD DSN=&OPTLIB,DISP=SHR
//C.SYSIN    DD DSN=&SRCLIB(&PRGNM),DISP=SHR
//C.SYSPRINT DD DSN=&LSTING(&PRGNM),DISP=OLD
//L.SYSLMOD  DD DSN=&LODLIB(&PRGNM),DISP=SHR
```

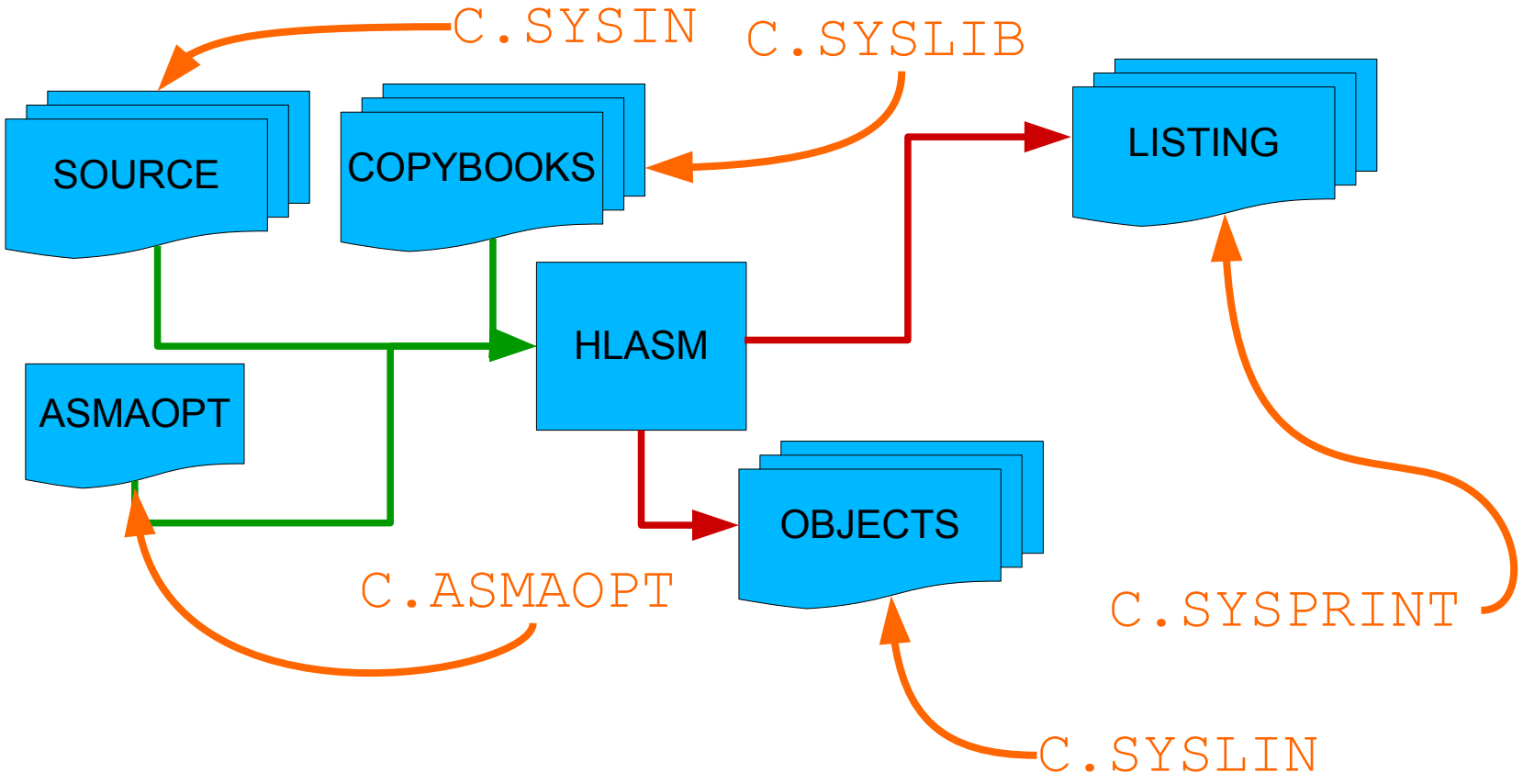
Working with the High Level Assembler HLASM

- The ASMACL JCL PROC has two steps:
  - C – Assemble the program
  - L – Bind the program
  - Using the notation *step*.*ddname* allows multiple DDNAMEs to be specified for the JCL steps
- DDNAMEs used by the Assemble step (C)
  - SYSIN → specifies the program's source
  - SYSLIN → specifies copybooks / macro libraries
  - SYSPRINT → specifies the destination of the program's *listing*
  - ASMAOPT → specifies a data set which contains assembly options*
  - PARM → specifies some options for HLASM*
- DDNAMEs used by the Bind step (L)
  - SYSLMOD → specifies the destination of the bound load module
- * Options may be specified in up to 6 different locations when assembling a program – allowing for a hierarchy of options to be specified.

Working with HLASM – Assembling and Binding a program



Working with HLASM – Assembling and Binding a program



Working with HLASM – Examining the listing

- The program listing produced by HLASM is an invaluable source of information for the programmer
- It lists in detail everything that HLASM has done when assembling a program including:
  - All the options that were used when the program was assembled
  - Any external references that the program produces or relies upon in the External Symbol Dictionary (ESD)
  - Each line of source code and the machine code that was produced
  - Any symbols in the code that require relocation in the Relocation Dictionary (RLD)
  - A summary of all lines in error in the program
  - A list of all data sets used
- The assembler uses the following return codes whenever it issues a message:
  - 0 – Success / Information
  - 2 – Notice – A condition which may need correcting – program appears to be correct
  - 4 – Warning – Program may not work as expected
  - 8 – Error – Error in program
  - 12 – Severe error – Unlikely that the program assembles as expected or runs
  - 16 – Critical – Unlikely that the program runs
  - 20 – Unrecoverable – Unable to continue

Working with HLASM – A look at syntax

```
File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
VIEW                                                    Columns 00001 00080
Command ===>                                             Scroll ===> CSR
****** ********************************* Top of Data ***********************************
000001 ************************************************************************
000002 * SIMPLE DUMMY EXIT FOR HLASM
000003 ************************************************************************
000004 *
000005 * MAIN PROGRAM STARTS HERE
000006 *
000007 LARLLOAD CSECT
000008 LARLLOAD AMODE 31
000009 LARLLOAD RMODE 24
000010 * USUAL PROGRAM SETUP
000011          STM   14,12,12(13)
000012          BALR  12,0          GET THE CURRENT ADDRESS
000013          USING *,12          USE 12 AS THE BASE REGISTER
000014          L     1,=F'12'
000015 LMRET    LM    14,12,12(13)
000016          XR    15,15
000017          BR    14
000018 * ********************************************************************
000019 * END OF PROGRAM
000020 * ********************************************************************
000021          END
****** ********************************* Bottom of Data ********************************
```

Working with HLASM – A look at syntax

```
File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
VIEW                                                    Columns 00001 00080
Command ===>                                             Scroll ===> CSR
****** ********************************* Top of Data ***********************************
000001 ************************************************************************
000002 * SIMPLE DUMMY EXIT FOR HLASM
000003 ************************************************************************
000004 *
000005 * MAIN PROGRAM STARTS HERE
000006 *
000007 LARLLOAD CSECT
000008 LARLLOAD AMODE 31
000009 LARLLOAD RMODE 24
000010 * USUAL PROGRAM SETUP
000011          STM   14,12,12(13)
000012          BALR  12,0          GET THE CURRENT ADDRESS
000013          USING *,12          USE 12 AS THE BASE REGISTER
000014          L     1,=F'12'
000015 LMRET    LM    14,12,12(13)
000016          XR    15,15
000017          BR    14
000018 * ************************************************************************
000019 * END OF PROGRAM
000020 * ************************************************************************
000021          END
****** ********************************* Bottom of Data ********************************
```

Comments start with a * in column 1 or appear after free-form instruction operands until column 72

Working with HLASM – A look at syntax

```
File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
VIEW                                                    Columns 00001 00080
Command ===>                                             Scroll ===> CSR
****** ********************************* Top of Data ***********************************
000001 ************************************************************************
000002 * SIMPLE DUMMY EXIT FOR HLASM
000003 ************************************************************************
000004 *
000005 * MAIN PROGRAM STARTS HERE
000006 *
000007 LARLLOAD CSECT
000008 LARLLOAD AMODE 31
000009 LARLLOAD RMODE 24
000010 * USUAL PROGRAM SETUP
000011          STM   14,12,12(13)
000012          BALR  12,0          GET THE CURRENT ADDRESS
000013          USING *,12          USE 12 AS THE BASE REGISTER
000014          L     1,=F'12'
000015 LMRET    LM    14,12,12(13)
000016          XR    15,15
000017          BR    14
000018 * ************************************************************************
000019 * END OF PROGRAM
000020 * ************************************************************************
000021          END
****** ********************************* Bottom of Data ********************************
```

Labels start in column 1

Working with HLASM – A look at syntax

```
File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
VIEW                                                    Columns 00001 00080
Command ===>                                             Scroll ===> CSR
****** ********************************* Top of Data ***********************************
000001 ************************************************************************
000002 * SIMPLE DUMMY EXIT FOR HLASM
000003 ************************************************************************
000004 *
000005 * MAIN PROGRAM STARTS HERE
000006 *
000007 LARLLOAD CSECT
000008 LARLLOAD AMODE 31
000009 LARLLOAD RMODE 24
000010 * USUAL PROGRAM SETUP
000011          STM   14,12,12(13)
000012          BALR  12,0          GET THE CURRENT ADDRESS
000013          USING *,12          USE 12 AS THE BASE REGISTER
000014          L     1,=F'12'
000015 LMRET    LM    14,12,12(13)
000016          XR    15,15
000017          BR    14
000018 * ************************************************************************
000019 * END OF PROGRAM
000020 * ************************************************************************
000021          END
****** ********************************* Bottom of Data ********************************
```

Instructions start after column 1 or a label

Working with HLASM – A look at syntax

```
File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
VIEW                                                    Columns 00001 00080
Command ===>                                             Scroll ===> CSR
****** ********************************* Top of Data ***********************************
000001 ***************************************************************************
000002 * SIMPLE DUMMY EXIT FOR HLASM
000003 ***************************************************************************
000004 *
000005 * MAIN PROGRAM STARTS HERE
000006 *
000007 LARLLOAD CSECT
000008 LARLLOAD AMODE 31
000009 LARLLOAD RMODE 24
000010 * USUAL PROGRAM SETUP
000011          STM   14,12,12(13)
000012          BALR  12,0          GET THE CURRENT ADDRESS
000013          USING *,12          USE 12 AS THE BASE REGISTER
000014          L     1,=F'12'
000015 LMRET    LM    14,12,12(13)
000016          XR    15,15
000017          BR    14
000018 * ***************************************************************************
000019 * END OF PROGRAM
000020 * ***************************************************************************
000021          END
****** ********************************* Bottom of Data ********************************
```

Operands start after a space after instructions and are delimited by commas and brackets

Working with HLASM – CSECTs and DSECTs

- CSECT → CONTROL SECTION (HLASM directive)
  - A CSECT contains machine instructions to be run on the machine
- DSECT → DUMMY SECTION (HLASM directive)
  - Used to define the structure of data
- Both CSECT and DSECT are terminated with the end statement

```
MYPROG   CSECT                  START OF CODE
     ...awesome assembler program goes here...
MYSTRUCT DSECT                  START OF DATA STRUCTURE
     ...awesome data structure goes here...
         END                    END OF PROGRAM
```

Working with HLASM – Defining Data

- Data is defined via the DC and DS HLASM directives
- DC – Define Constant
  - Defines data and initialises it to a given value
- DS – Define Storage
  - Defines storage for data but does not give it a value
- e.g.

```
NUMBER1   DC    F'12'           DEFINE A FULLWORD WITH VALUE 12
NUMBER2   DC    H'3'            DEFINE A HALFWORD WITH VALUE 3
TOTAL     DS    H               DEFINE A HALFWORD
MYSTR     DC    C'HELLO WORLD'  DEFINE A SERIES OF CHARACTERS
MYHEX     DC    X'FFFF'         DEFINE A SERIES OF HEX
                                CHARACTERS
```

Working with HLASM – Literals

- A literal is an inline definition of data used in an instruction but the space taken for the literal is in the nearest literal pool
- A literal pool collects all previous literals and reserves the space for them
- By default, HLASM produces an implicitly declared literal pool at the end of your CSECT
- To cause HLASM to produce a literal pool, use the LTORG directive

```
          L     1,=F'1'          LOAD REGISTER 1 WITH FULLWORD OF 1
          X     1,=H'2'          XOR REGISTER 1 WITH HALFWORD OF 2
        ...more awesome assembler code here...
            LTORG ,              THE LITERAL POOL IS CREATED
```

Working with HLASM – Machine VS Assembler Instructions

- There are two distinct types of instruction accepted by HLASM – Machine instructions and Assembler instructions
- Machine instructions are changed into machine code by the assembler – these are the instructions that will be used when your program is executed
- Assembler instructions are instructions which affect the behavior of the assembler itself as it is assembling a program
- Some assembler instructions such as DC will cause the resultant program to contain certain pieces of data although the majority of assembler instructions do not
- The assembler also includes a macro language – macros are code which are used to generate other code programmatically and therefore can shorten development time

Introduction to Assembler Programming
Moving Data

Moving Data

- Moving data is the most common operation that ANY computer program performs
- Before any computations can be performed, data must be moved to the correct places
  - Data is moved to the processor (from disk, memory, networks, devices etc)
  - Data is manipulated by the processor
  - The result is stored somewhere (back to disk, memory, networks, devices etc)
- Data is LOADed into the processor's registers via LOAD instructions
- Data is STOREd to memory via STORE instructions

Moving Data – Loading from Register to Register

- The LOAD REGISTER (LR) instruction is used to load the value stored in one register to another

```
LR 1,2    LOAD REGISTER 2 INTO REGISTER 1 (32-BITS)
```

- The instruction copies 32-bits from a register to another
- The instruction has a 64-bit variant LOAD GRANDE REGISTER (LGR)

```
LGR 1,2    LOAD REGISTER 2 INTO REGISTER 1 (64-BITS)
```

- The instruction has a 16-bit variant LOAD HALFWORD REGISTER

```
LHR 1,2    LOAD REGISTER 2 INTO REGISTER 1 (16-BITS)
```

Moving Data – Loading from Memory to Register

- The LOAD (L) instruction is used to load the value stored in memory to a register

```
L 1,MY_NUM     LOAD REGISTER 1 WITH THE VALUE MY_NUM (32-BITS)
```

- The instruction copies 32-bits from memory to a register
- The instruction has a 64-bit variant LOAD GRANDE (LG)

```
LG 1,MY_NUM    LOAD REGISTER 1 WITH THE VALUE MY_NUM (64-BITS)
```

- The instruction has a 16-bit variant LOAD HALFWORD

```
LH 1,MY_NUM    LOAD MY_NUM INTO REGISTER 1 (16-BITS)
```

Moving Data – Storing from a Register to Memory

- The STORE (ST) instruction is used to store the value in a register to memory

```
ST 1,MY_NUM     STORE REGISTER 1 TO MY_NUM (32-BITS)
```

- The instruction copies 32-bits from a register to memory
- The instruction has a 64-bit variant STORE GRANDE (STG)

```
STG 1,MY_NUM    STORE REGISTER 1 TO MY_NUM (64-BITS)
```

- The instruction has a 16-bit variant STORE HALFWORD

```
STH 1,MY_NUM    STORE REGISTER 1 TO MY_NUM (16-BITS)
```

Moving Data – Moving data without registers

- The MOVE (MVC) instruction can be used to move data in memory without the need for a register

```
MVC MY_OUTPUT,MY_INPUT          MOVE MY_INPUT TO MY_OUTPUT
```

- The MVC instruction can move up to 256B from one area of memory to another
- The MVCL instruction can move up to 16M (but uses different parameters)
- The MVCLE instruction can move up to 2G (or up to 16EB in 64-bit addressing)
- In all cases, the move instruction moves 1 byte at a time (left to right in memory)

Why no one writes assembler like this...

- Many instructions may appear very similar but may have very different (and unintended) consequences, e.g.:

```
LR 1,2      Load register 1 with the value of register 2
```

- Unless you know what you're doing, don't do this:

```
L  1,2      Load register 1 with the value at memory offset 2
```

- The very simple example above shows how confusing assembler programming can be as there is no distinction between writing the value of a number and the value of a register in an instruction and instead the distinction is made by which instruction was written by the programmer.
- Many programmers often name their registers r0-r15 to make it clearer to understand which values are registers and which values are numbers in an instruction. To do this, the best solution is to start a program with the statement `ASMDREG` which will include all the register names for your program to use.
  - From now on, this material will refer to GPRs registers as r0-r15
- Using r0-r15 also means that type checking can be performed by HLASM and they will also appear in the cross-reference section of the listing making it easier to find that rogue register...

Introduction to Assembler Programming
Logical Operations

Logical Instructions – EXCLUSIVE OR (X, XG, XR, XGR, XC)

- The EXCLUSIVE OR instructions perform the EXCLUSIVE OR *bit-wise* operation

```
X    r1,MY_NUM     XOR REGISTER 1 WITH MY_NUM (32-BITS)
XG   r1,MY_NUM     XOR REGISTER 1 WITH MY_NUM (64-BITS)
XR   r1,r2         XOR REGISTER 1 WITH REGISTER 2 (32-BITS)
XGR  r1,r2         XOR REGISTER 1 WITH REGISTER 2 (64-BITS)
XC   NUM1,NUM2     XOR NUM1 WITH NUM2 (UP TO 256-BYTES)
```

- Notice a pattern with the instruction mnemonics?
  - Rules of thumb:
    - G → 64bits (DOUBLEWORD)
    - H → 16bits (HALFWORD)
    - R → register
    - C → character (byte / memory)
    - L → logical (i.e. unsigned)

Logical Instructions – AND (N*x*), OR (O*x*)

- The AND instructions perform the AND *bit-wise* operation

```
N   r1,MY_NUM    AND REGISTER 1 WITH MY_NUM (32-BITS)
NG  r1,MY_NUM    AND REGISTER 1 WITH MY_NUM (64-BITS)
NR  r1,r2        AND REGISTER 1 WITH REGISTER 2 (32-BITS)
NGR r1,r2        AND REGISTER 1 WITH REGISTER 2 (64-BITS)
NC  NUM1,NUM2    AND NUM1 WITH NUM2 (UP TO 256-BYTES)
```

- The OR instructions perform the OR *bit-wise* operation

```
O   r1,MY_NUM    OR REGISTER 1 WITH MY_NUM (32-BITS)
OG  r1,MY_NUM    OR REGISTER 1 WITH MY_NUM (64-BITS)
OR  r1,r2        OR REGISTER 1 WITH REGISTER 2 (32-BITS)
OGR r1,r2        OR REGISTER 1 WITH REGISTER 2 (64-BITS)
OC  NUM1,NUM2    OR NUM1 WITH NUM2 (UP TO 256-BYTES)
```

A word on instruction choice

- In 5 basic operations (loading, storing, AND, OR, XOR) we have already seen over 25 instructions!
- How do I decide which instruction to use?
  - The instruction should be chosen for:
    - Its purpose, e.g. don't use a STORE instruction to LOAD a register – it won't work!
    - Its data, e.g. 32-bits, 16-bits, 64-bits, bytes?
- Many instructions can perform *similar* operations, e.g.

```
XR    r15,r15     XOR REGISTER 15 WITH REGISTER 15
L     r15,=F'0'   LOAD REGISTER 15 WITH 0
LA    r15,0       LOAD REGISTER 15 WITH ADDRESS 0
```

- Different instructions NEVER do the same thing even if you think they do
  - The result does not justify the means

Introduction to Assembler Programming
Arithmetic

Arithmetic

- Arithmetic is performed in a wide variety ways on z/Architecture
  - Fixed point arithmetic (including logical) ← performed in GPRs
  - Packed Decimal arithmetic ← performed in memory
  - Binary and Hexadecimal Floating point arithmetic ← performed in FPRs
- Fixed point arithmetic
  - Normal arithmetic, e.g. adding the contents of 2 numbers together
  - Fixed point arithmetic is signed with numbers being stored in 2's complement form
  - Logical fixed point arithmetic is unsigned, i.e. both numbers are positive
- Pack Decimal arithmetic
  - Performed in memory
  - Numbers are in packed decimal format

Arithmetic – Fixed point arithmetic operations

- ADD instructions

```
AR    r1,r2       ADD REGISTER 2 TO REGISTER 1 (32-BIT SIGNED)
ALR   r1,r2       ADD REGISTER 2 TO REGISTER 1 (32-BIT LOGICAL)
A     r1,MY_NUM   ADD MY_NUM TO REGISTER 1 (32-BIT SIGNED)
AL    r1,MY_NUM   ADD MY_NUM TO REGISTER 1 (32-BIT LOGICAL)
AFI   r1,37       ADD 37 TO REGISTER 1 (IMMEDIATE 32-BIT SIGNED)
```

- Note that for immediate instructions, the operand is included in the instruction rather than needing to be obtained from memory
- At the end of the addition, the CC is updated (as specified in POPs)
  - CC → 0 → Result is 0; no overflow
  - CC → 1 → Result less than 0; no overflow
  - CC → 2 → Result greater than 0; no overflow
  - CC → 3 → Overflow occurred

Arithmetic – Fixed point arithmetic operations

- SUBTRACT instructions

```
SR    r1,r2      SUBTRACT REGISTER 2 FROM REGISTER 1 (SIGNED)
SLR   r1,r2      SUBTRACT REGISTER 2 FROM REGISTER 1 (LOGICAL)
S     r1,MY_NUM  SUBTRACT MY_NUM FROM REGISTER 1 (SIGNED)
SL    r1,MY_NUM  SUBTRACT MY_NUM FROM REGISTER 1 (LOGICAL)
AFI   r1,-37     ADD -37 TO REGISTER 1 (IMMEDIATE 32-BIT SIGNED)
```

- At the end of the subtraction, the CC is updated (as specified in POPs)
  - CC → 0 → Result is 0; no overflow
  - CC → 1 → Result less than 0; no overflow
  - CC → 2 → Result greater than 0; no overflow
  - CC → 3 → Overflow occurred

Arithmetic – Fixed point arithmetic operations

- MULTIPLY instructions

```
MR    r2,r7          MULTIPLY REGISTER 2 BY REGISTER 7
M     r2,MY_NUM      MULTIPLY REGISTER 2 BY MY_NUM
```

- The first operand is an even-odd pair – the result of the MULTIPLY is stored in:
  - The even register (of the pair) – top 32-bits of result
  - The odd register (of the pair) – bottom 32-bits of the result
- At the end of the multiplication, the CC is UNCHANGED

Arithmetic – Fixed point arithmetic operations

- DIVIDE instructions

```
DR    r2,r7          DIVIDE REGISTER 2 BY REGISTER 7
D     r2,MY_NUM      DIVIDE REGISTER 2 BY MY_NUM
```

- The first operand is an even-odd pair
  - The even register (of the pair) – top 32-bits of dividend
  - The odd register (of the pair) – bottom 32-bits of the dividend
- The result is stored in the first operand:
  - The quotient is stored in the odd register of the pair
  - The remainder in the even register of the pair
- At the end of the division, the CC is UNCHANGED

Introduction to Assembler Programming
Branching

Branching

- Branching allows control flow in the program to move nonsequentially
- Branches are performed via the BRANCH instructions
- Most branch instructions are *conditional* – i.e. they will pass control to the *branch target* if a condition is met otherwise control will continue sequentially
- The condition on which the branch will take place is called the CONDITION CODE (CC)
  - The CC is 2-bits stored in the PSW; thus the value is 0-3
  - Each instruction may (or may not) set the CC
- A branch instruction provides a *branch mask*
  - The *branch mask* instructs the processor that the branch will be taken if any of the bits in the CC match those in the branch mask
- Fortunately, HLASM provides extended-mnemonics which provide branch masks for most branch instructions

Branching – Using HLASM's extended-mnemonics

- B – Branch (unconditionally)
- BE – Branch on condition Equal
- BL – Branch on condition Lower than
- BH – Branch on condition Higher than
- BNL – Branch Not Low
- BNH – Branch Not High
- BZ – Branch on Zero
- BNZ – Branch Not Zero
- There are also other extended-mnemonics which HLASM provides

Branching – How does a branch mask work

- B – Branch (unconditionally)
  - This is translated to the BRANCH ON CONDITION (BC) instruction with a mask of 15

Condition Code	0	1	2	3
Mask value	8	4	2	1

- So, 15 → b'1111' → 8+4+2+1
- Thus the branch is taken if CC 0, 1, 2 or 3 is met, i.e. ALWAYS

Branching – How does a branch mask work

- BE – Branch on Equal
  - This is translated to the BRANCH ON CONDITION (BC) instruction with a mask of 8

Condition Code	0	1	2	3
Mask value	8	4	2	1

- So, 8 → b'1000' → 8
- Thus the branch is taken if CC 0 is met

Branching – Using a branch to form an *if* statement

```
      LT    r1,MY_NUM   LOAD MY_NUM INTO REGISTER 1 AND SET CC
      BNZ   NONZERO     BRANCH TO 'NONZERO' IF REGISTER 1 IS NOT ZERO
          ...code where register 1 is zero goes here...
      B    COMMONCODE       REJOIN COMMON CODE

NONZERO DS 0H
          ...code where register 1 is non-zero goes here...

COMMONCODE DS 0H
```

Branching – Using a branch to form an *if* statement

```
//Example C-like equivalent
if(register_1==0){
    //Code for register_1 being 0 goes here
}
else{
    //Code for register_1 being non-zero goes here
}

//Common code goes here
```

Introduction to Assembler Programming
Looping

Looping

- A simple loop is formed by using a counter, a comparison and a branch, e.g.

```
         LA    r2,0            INITIALISE COUNTER REGISTER TO 0
MYLOOP   AHI   r2,1            INCREMENT COUNTER
         WTO   'HELLO'         SAY HELLO
         CHI   r2,10           IS THE COUNTER 10?
         BL    MYLOOP          IF IT'S LESS THAN 10, GO TO MYLOOP
```

- That's simple – but there's a better way – use BRANCH ON COUNT (BCT)

```
         LA    r2,10            INITIALISE COUNTER REGISTER TO 10
MYLOOP   WTO   'HELLO'
         BCT   r2,MYLOOP        SUBTRACTS, COMPARES & BRANCHES
```

- There are other instructions similar to BCT that subtract/add values and then branch depending on the result, e.g. BCTR, BXH etc...

Introduction to Assembler Programming
Addressing Data

Addressing Data

- There are 2 ways to access data for manipulation
  - Base-Displacement (and index) addressing
  - Relative addressing
- Relative addressing is a new form of addressing which calculates the data's relative position from the current PSW (in half-word increments)

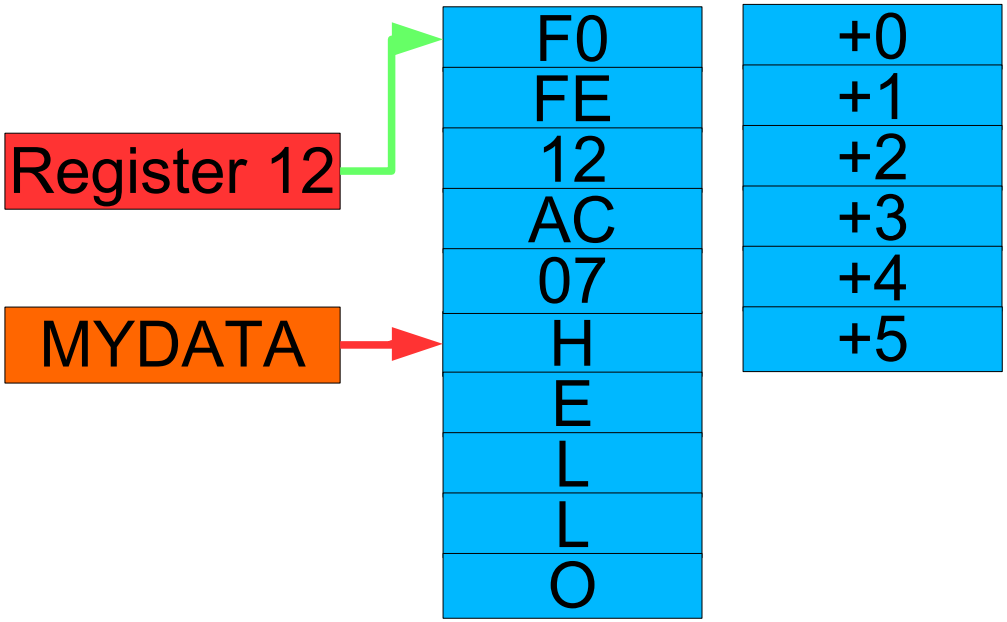
```
          LRL   r1,MY_NUM    LOAD RELATIVE REGISTER 1 WITH MY_NUM
        ...more awesome assembler code here...
MY_NUM DC   F'23'
```

Addressing Data - Base-Displacement-Index

- Base-Displacement(-index) addressing involves using a register as a pointer to memory – this is called the BASE register
  - Base (and index) registers, are GPRs and the term *base* and *index* are used purely for referring to the different usage in a particular set of instructions
- A displacement is usually between 0 and 4095 bytes allowing a single base register to address 4K of memory
- An index register is an additional register whose value is added to the base and displacement to address more memory
- Incrementing an index register allows the assembler programmer to cycle through an array whilst maintaining the same base-displacement
- Note that register 0 cannot be used as a base or index register
  - Register 0 used in this way means that the *value* 0 will be used as a base / index and NOT the contents of register 0
- Base, displacement and indexes are optionally specified on an instruction
  - Implicit default value for each is 0

Addressing Data - Base-Displacement-Index

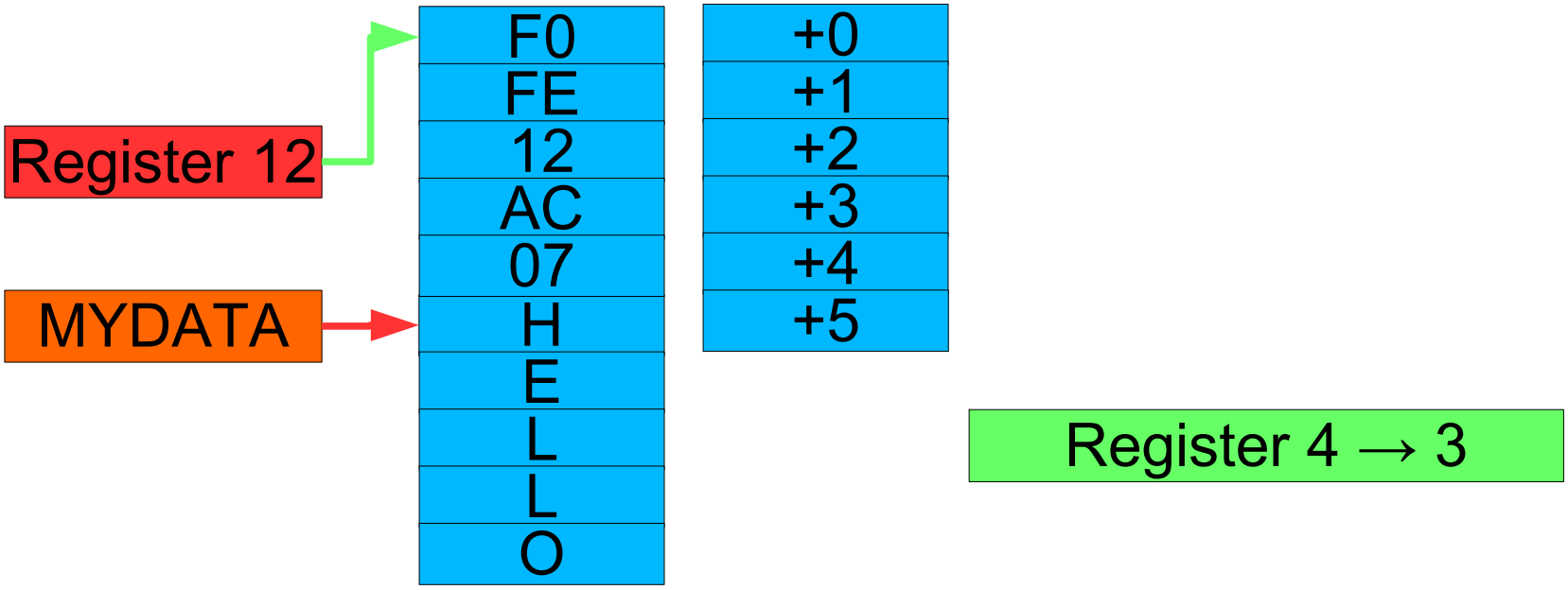
- Address = BASE(register) + INDEX(register) + DISPLACEMENT



Register 4 → 3

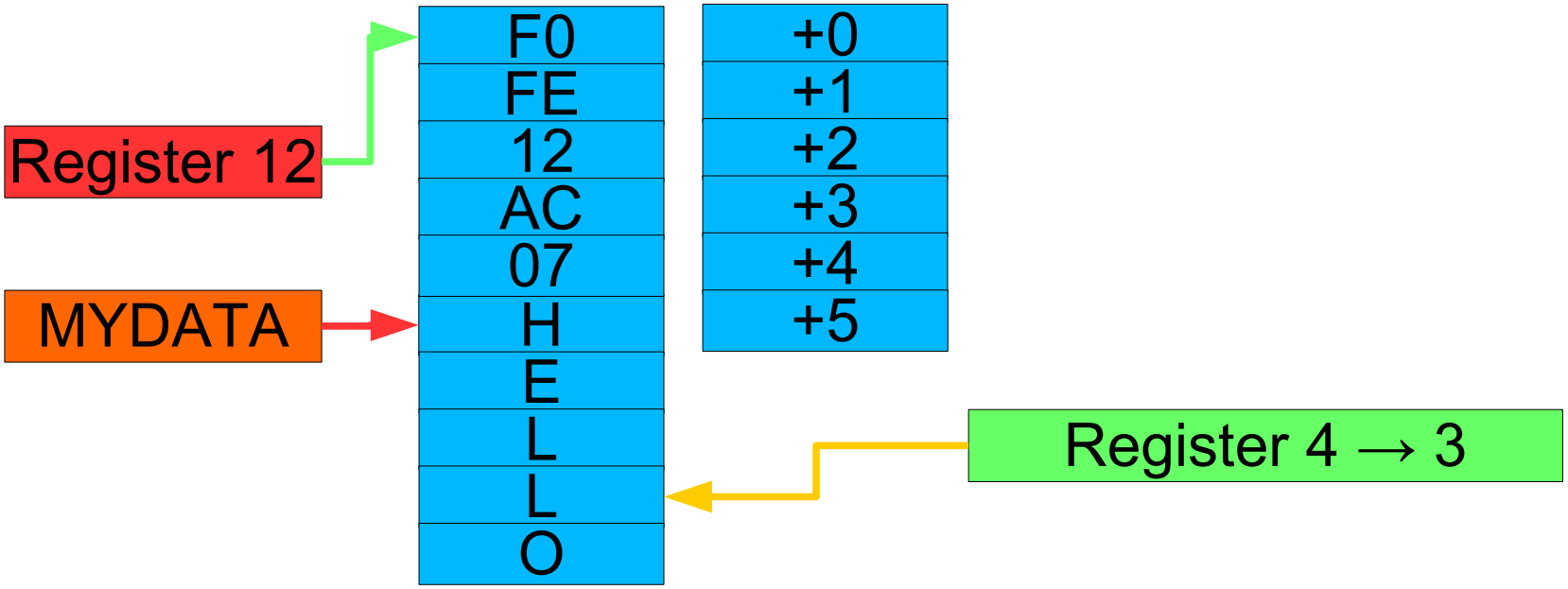
Addressing Data - Base-Displacement-Index

- Address of MYDATA = 5(r0,r12) → *displacement 5 + index (register) 0 + base (register) 12*



Addressing Data - Base-Displacement-Index

- Address of 'L' in 'HELLO' = 5(r4,r12) → *displacement 5 + index (register) 4 + base (register) 12*



Addressing Data – Loading addresses

- To load an address into a register, use the LOAD ADDRESS (LA) instruction

```
LA    r1,DATA        LOAD ADDRESS OF DATA INTO REGISTER 1
```

- The LA instruction can be used to set a register to between 0 and 4095 by specifying a base and index register of 0 – these are automatically implicitly specified, e.g.

```
LA    r1,12          base=0, index=0, displacement=12
```

- To store a 'L' in 'HELLO' in the previous example:

```
...some setup for REGISTER 12...
LA    r4,3              LOAD ADDRESS 3* INTO REGISTER 4
IC    r3,=C'L'          LOAD CHARACTER 'L' INTO REGISTER 3
STC   r3,MYDATA(r4)     base=12, index=4, displacement=5
```

Addressing Data – Base VS Index registers

- Both base and index registers can be dropped from an instruction and can *almost* be used interchangeably. Consider the following:

```
LA    r4,MYDATA       LOAD ADDRESS OF MYDATA INTO REGISTER 4
IC    r3,=C'H'        LOAD CHARACTER 'H' INTO REGISTER 3
STC   r3,0(,r4)       STORE H AT 0 DISP + (0 INDEX) + BASE 4
```

- Contrast with:

```
LA    r4,MYDATA       LOAD ADDRESS OF MYDATA INTO REGISTER 4
IC    r3,=C'H'        LOAD CHARACTER 'H' INTO REGISTER 3
STC   r3,0(r4)        STORE H AT 0 DISP + 4 INDEX + (BASE 0)
```

- Both store the character H at the same address, i.e. at a displacement 0 from the start of MYDATA.
- However, if the program is running in Access Register Mode, then using a *base* register will cause its corresponding access register to be involved in the address calculation too!

Addressing Data – When a base register is not a base register...

- Certain instructions, e.g. various shift instructions, specify a base-displacement operand but do not address storage
- Instead, the calculated address forms a value for the operand which is used by the instruction
- The advantage of instructions such as this is that by varying the contents of a register, the same instruction in a program can be used repeatedly but it will modify varying amounts of data

Addressing Data – the HLASM USING instruction

- HLASM can be used to generate base-displacement values for your program automatically.
- In order to do this, the USING instruction is used to specify a base register for program / data addressability.
- HLASM will then calculate any displacements necessary in order to address parts of the program or data, e.g.:

```
BALR    r12,0       LOAD ADDRESS OF NEXT INSTRUCTION INTO R12
USING   *,r12       USE R12 AS A BASE REGISTER FROM HERE
```

- If a register that is being used as a base register by HLASM needs to be used for another purpose, it is good practice to use the DROP instruction to cancel the previous USING instruction for that register, e.g.:

```
DROP    r12         DROP THE USE OF R12 AS A BASE
```

- Note that HLASM has different forms for the USING instruction – the HLASM Language Reference should be consulted for more information.

Addressing Data – using multiple base registers

- For most instructions, the displacement field is only 12-bits in length and therefore a maximum displacement value of 4095 can be achieved.
- If a section of code or data is bigger than 4096-bytes in size, then the programmer must use another base register to address the data further into the code. This can be done be issuing another USING instruction and assigning another register as a base register.
  - It is important that the other base register's contents point at the correct place in the code which it will be addressing
- A common use of multiple base registers is to assign one set of base registers for a piece of code and another for the data in the program.
- The HLASM PUSH instruction can be used to save the current state of the USINGs for a program and the POP instruction can be used to restore the previous USING state.

Addressing Data – using multiple base registers

```
* Call the subroutine MYSUB
             LA    r15,MYSUB       Prepare to call subroutine
             BALR  r14,r15         Call it
             LTR   r15,r15         Check return code



* ************************************************************************
* START OF MY SUBROUTINE MYSUB
* This subroutine starts with the contents of register 15 pointing to the
* start of the code.  The mainline code of the program uses register 12 as
* its base register.
* ************************************************************************
MYSUB        DC    0H
* Subroutine code goes here...
             L     r15,RETURN_CODE     Set return code
             BR    r14                 Branch back to caller
RETURN_CODE  DC    F'1'
```

Addressing Data – using multiple base registers

```
000000 05C0                          11          BALR  r12,0
                  R:C  00002         12          USING *,r12
000002 41F0 C008            0000A    13          la    r15,mysub
000006 05EF                          14          balr  r14,r15
000008 12FF                          15          ltr   r15,r15
                                     16 * ****************************
                                     17 * MYSUB
                                     18 * ****************************
00000A                               19 mysub    dc    0h
00000A 58F0 C00E            00010    20          l     r15,return_code
00000E 07FE                          21          br    r14
000010 00000001                      22 return_code dc f'1'
```

- Register 12 is being used as a base register for both the main program and the subroutine.

Addressing Data – using multiple base registers

```
* Call the subroutine MYSUB
             LA    r15,MYSUB       Prepare to call subroutine
             BALR  r14,r15         Call it
             LTR   r15,r15         Check return code



* ********************************************************************
* START OF MY SUBROUTINE MYSUB
* This subroutine starts with the contents of register 15 pointing to the
* start of the code.  The mainline code of the program uses register 12 as
* its base register.
* ********************************************************************
MYSUB        DC    0H
* Subroutine code goes here...
             PUSH  USING
             USING *,r15                 Set new base register*
             L     r15,RETURN_CODE       Set return code
             BR    r14                   Branch back to caller
             POP   USING
RETURN_CODE  DC    F'1'
```

Addressing Data – using multiple base registers

```
000000 05C0                          11          BALR  r12,0
                    R:C  00002       12          USING *,r12
000002 41F0 C008             0000A   13          la    r15,mysub
000006 05EF                          14          balr  r14,r15
000008 12FF                          15          ltr   r15,r15
                                     16          DROP  r12
                                     17 * ********************************
                                     18 * MYSUB
                                     19 * ********************************
00000A                               20 mysub    dc    0h
                                     21          push  using
                    R:F  0000A       22          using *,r15
00000A 58F0 F006             00010   23          l     r15,return_code
00000E 07FE                          24          br    r14
                                     25          pop   using
000010 00000001                      26 return_code dc f'1'
```

- Register 12 is being used as a base register for the main program
- The current state of the USINGs is saved via the PUSH USING statement
- Register 15 is established as the base register for the subroutine

Addressing Data – Addressing beyond 4096 bytes

- There are a number of methods for addressing code and data beyond 4096 bytes – we have already see how we could use more than one base register to do this.
- The LONG DISPLACEMENT family of instructions can address up to 512KB (both positive and negative from the base register) of data using 20-bits. This is only available if the long-displacement facility is installed.
- Using relative instructions, an immediate field is encoded into the instruction which specifies the number of half-words that the target of the instruction is from the instruction itself.
  - Relative instructions can be used to eliminate the need for base registers altogether – so long as your code has been designed for it

Introduction to Assembler Programming
Calling conventions

Calling Conventions

- A calling convention is a convention used between programs and subroutines to call each other
- The calling convention is not enforced, but if it is disregarded undesirable and unpredictable results may occur
- In general, when programming in assembler, the *caller* will provide a *save area* and the *called* program or routine will save all GPRs into that save area.
- The subroutine will then execute its code
- To return control to the caller, the subroutine will typically:
  - Set a return code in a register
  - Prepare the register on which it should branch back on
  - Restore all other registers
  - Branch back

Calling Conventions – Typical register usage on z/OS

- Although free to do as they please, most assembler programs on z/OS use the following register convention during initialisation
  - Register 1 → parameter list pointer
  - Register 13 → pointer to register save area provided by caller
  - Register 14 → return address
  - Register 15 → address of subroutine
- Once the registers are saved, the called subroutine will:
  - Update register 13 to point to a new save area (so that it can call other programs / routines)
  - Establish register 12 as a base register for the program
- Upon termination, the called subroutine will:
  - Set a return code in register 15
  - Restore register 13 to the value it was previously
  - Restore registers 14,0,1,...,12 from the save area pointed to by register 13
  - Branch back on register 14

Calling a subroutine in code – Going in...

- The caller calls the subroutine

```
LA    r1,PARAMS        POINT TO PARAMETERS
LA    r15,SUB1         LOAD ADDRESS OF SUBROUTINE
BALR  r14,r15          BRANCH AND LINK
LTR   r15,r15          CHECKS RETURN CODE 0?
...caller code continues here...
```

- The subroutine saves the caller's registers and establishes a base register

```
STM   r14,r12,12(r13)   STORE REGISTERS
LR    r12,r15           GET ENTRY ADDRESS
...subroutine code continues here...
```

Calling a subroutine in code – Getting out...

- The subroutine restores the caller's registers, sets the return code and branches back

```
LM    r14,r12,12(r13)   RESTORE REGISTERS
XR    r15,r15           SET RETURN CODE 0
BR    r14               BRANCH BACK TO CALLER
```

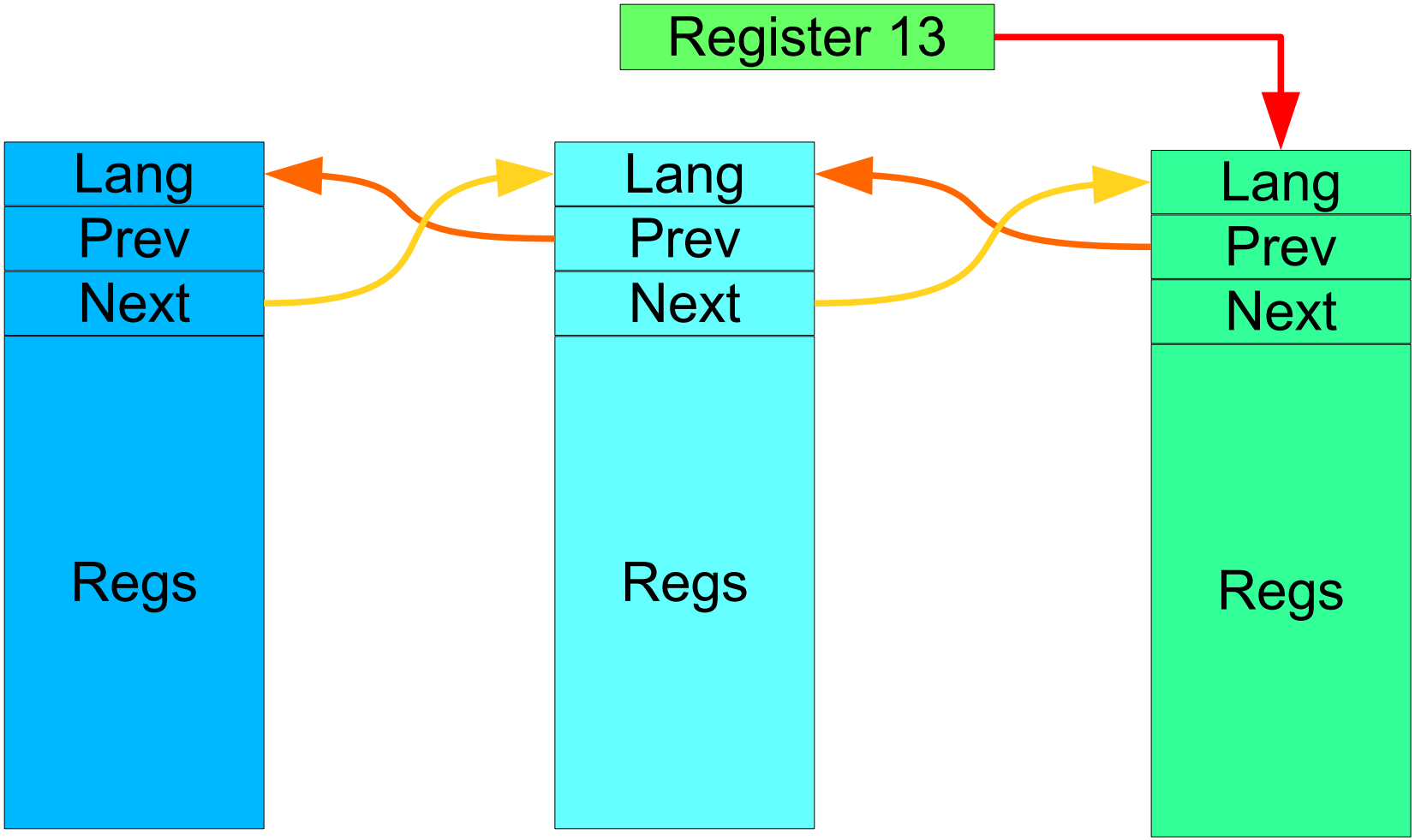
- Due to this calling convention, during epilog and prologue of a program or subroutine or when calling or having control returned from a program or subroutine, avoid using registers 0, 1, 12, 13, 14, 15
- z/OS services, typically will use registers 0, 1, 14, 15
- Not sure which registers are used by a service?
  - The manuals explain in detail

What exactly is a “save area”?

- As with most things in software, the answer is, “it depends”
  - On z Systems, there are different types of save area which are used differently depending on the calling convention in use.
  - Your program must use the correct type of save area
- The standard z/OS linkage save area has the following format:

Offset in save area	Purpose
0	Used by language products
4	Address of previous save area
8	Address of next save area
12	Register 14
16	Register 15
20	Registers 0-12

Chaining Save areas



Chaining save areas – Going in...

- The caller calls the subroutine

```
LA     r1,PARAMS        POINT TO PARAMETERS
LA     r15,SUB1         LOAD ADDRESS OF SUBROUTINE
BALR   r14,r15          BRANCH AND LINK
LTR    r15,r15          CHECKS RETURN CODE 0?
...caller code continues here...
```

- The subroutine saves the caller's registers and establishes a base register

```
STM    r14,r12,12(r13)   STORE REGISTERS
GETMAIN RU,LV=72         Get storage for save area
ST     r13,4(,r1)        Chain previous save area to new
ST     r1,8(,r13)        Chain new to previous
LR     r13,r1            Set r13 to new save area
LR     r12,r15           GET ENTRY ADDRESS
...subroutine code continues here...
```

Chaining save areas – Getting out...

- The caller calls the subroutine

```
LA    r1,PARAMS        POINT TO PARAMETERS
LA    r15,SUB1         LOAD ADDRESS OF SUBROUTINE
BALR  r14,r15          BRANCH AND LINK
LTR   r15,r15          CHECKS RETURN CODE 0?
...caller code continues here...
```

- The subroutine restores frees its save area, restores the caller's registers, sets a return code and branches back

```
LR    r1,r13             Address of save area to free
LA    r0,72              Length of save area
L     r13,4(,r13)        Point at previous save area
FREEMAIN R,LV=(0),A=(1)  Free save area
LM    r14,r12,12(13)     Restore registers
XR    r15,r15            Set return code
BR    r14                Branch back to caller
```

Introduction to Assembler Programming
How to read Principles of Operation

Reading POPs

- Principles of Operation (better known as POPs) is the z/Architecture manual
- It explains everything from system organisation and memory, to instructions and number formats
- It provides a useful set of appendices some of which provide good detailed examples of instruction use, including programming techniques
- The vast majority of POPs is instruction descriptions
  - Hint – Appendix A contains examples of instructions

Reading POPs – Understanding Instruction Descriptions

- Each instruction is described in exact detail including:
  - The instruction's syntax
  - Machine code
  - Operation
  - Condition code settings
  - Programming Exceptions
- There are 2 forms of syntax provided for each instruction
  - The syntax for the assembler, i.e. what is written in your assembler program
  - The machine code for the instruction, i.e. the binary code run on the processor
- The instruction's machine code is grouped together with other instructions which share a similar machine code layout called an *instruction format*

Reading POPs – Instruction Formats

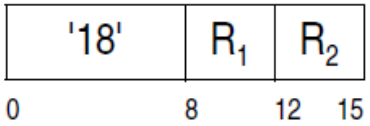
- The instruction format used, is generally related to
  - The assembler syntax used to code the instruction
  - The operation that the instruction performs
- Instructions that we've used have had the following formats:
  - RR – Register-Register – this form usually manipulates registers, e.g. LR, MR, DR
  - RX – Register, Index, base displacement – usually moving data between memory and registers, e.g. L, LA, ST, A, X, S, D, M
  - SS – Storage-Storage – acts on data in memory, e.g. MVC

Reading POPs – Instruction Formats – RR – LR instruction

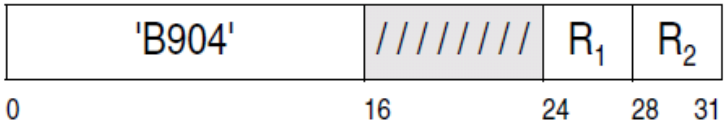
LOAD

Register-and-register formats:

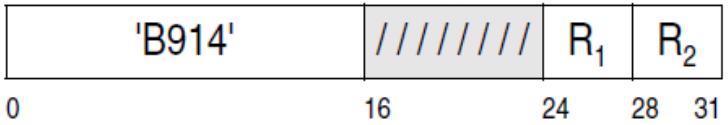
LR R_1,R_2 [RR]



LGR R_1,R_2 [RRE]



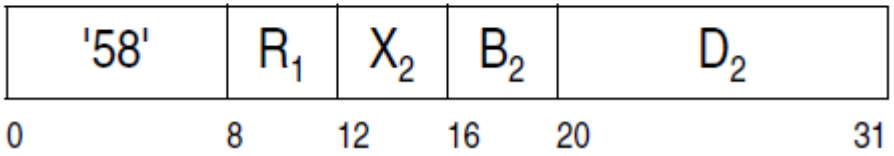
LGFR R_1,R_2 [RRE]



Reading POPs – Instruction Formats – RX – L instruction

Register-and-storage formats:

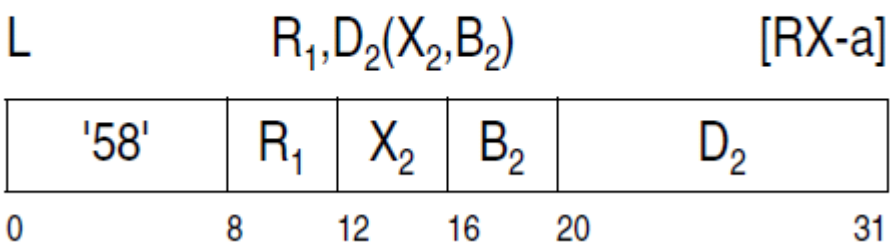
L $R_1,D_2(X_2,B_2)$ [RX-a]



Looking at Instruction Formats

- Why do I need to know an instruction's format?
  - You don't...but it might come in useful when debugging...
- Consider having a dump and the failing instruction was 5810 0004
- Examining the “Principles of Operation” z/Architecture manual tells me:
  - 58 = LOAD Instruction
  - Format = RX-a

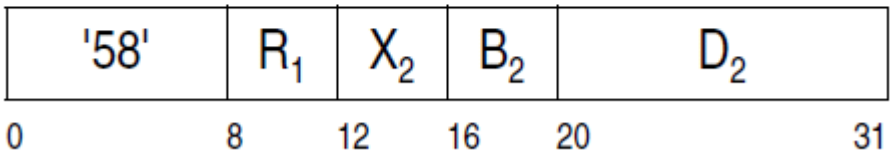
Register-and-storage formats:



Looking at Instruction Formats

Register-and-storage formats:

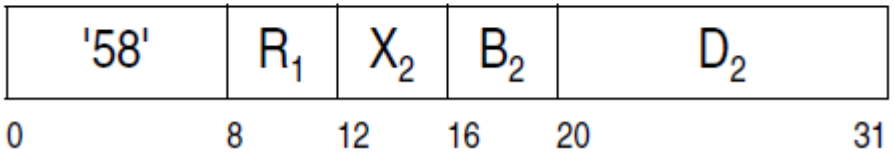
L $R_1,D_2(X_2,B_2)$ [RX-a]



5810 0004

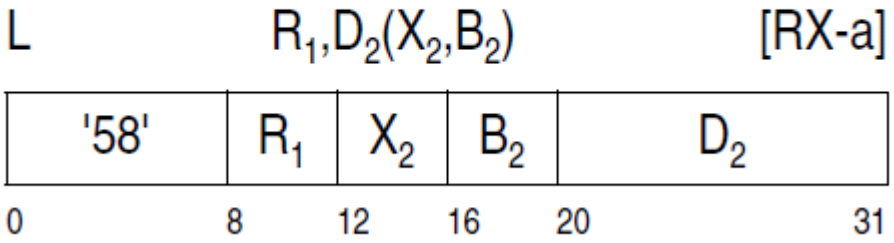
Register-and-storage formats:

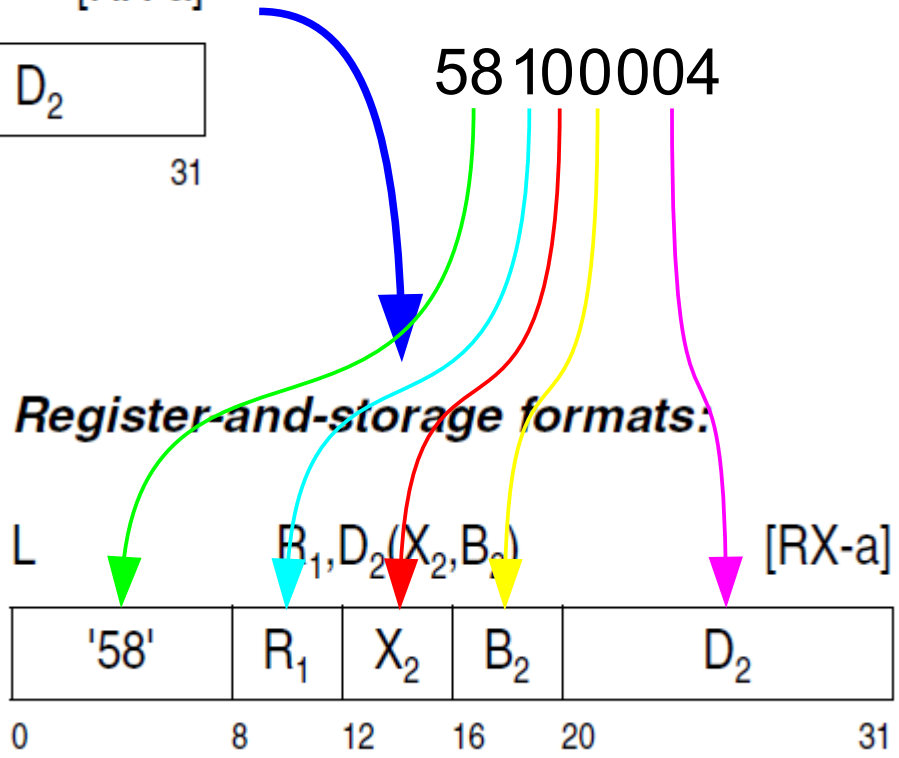
L $R_1,D_2(X_2,B_2)$ [RX-a]



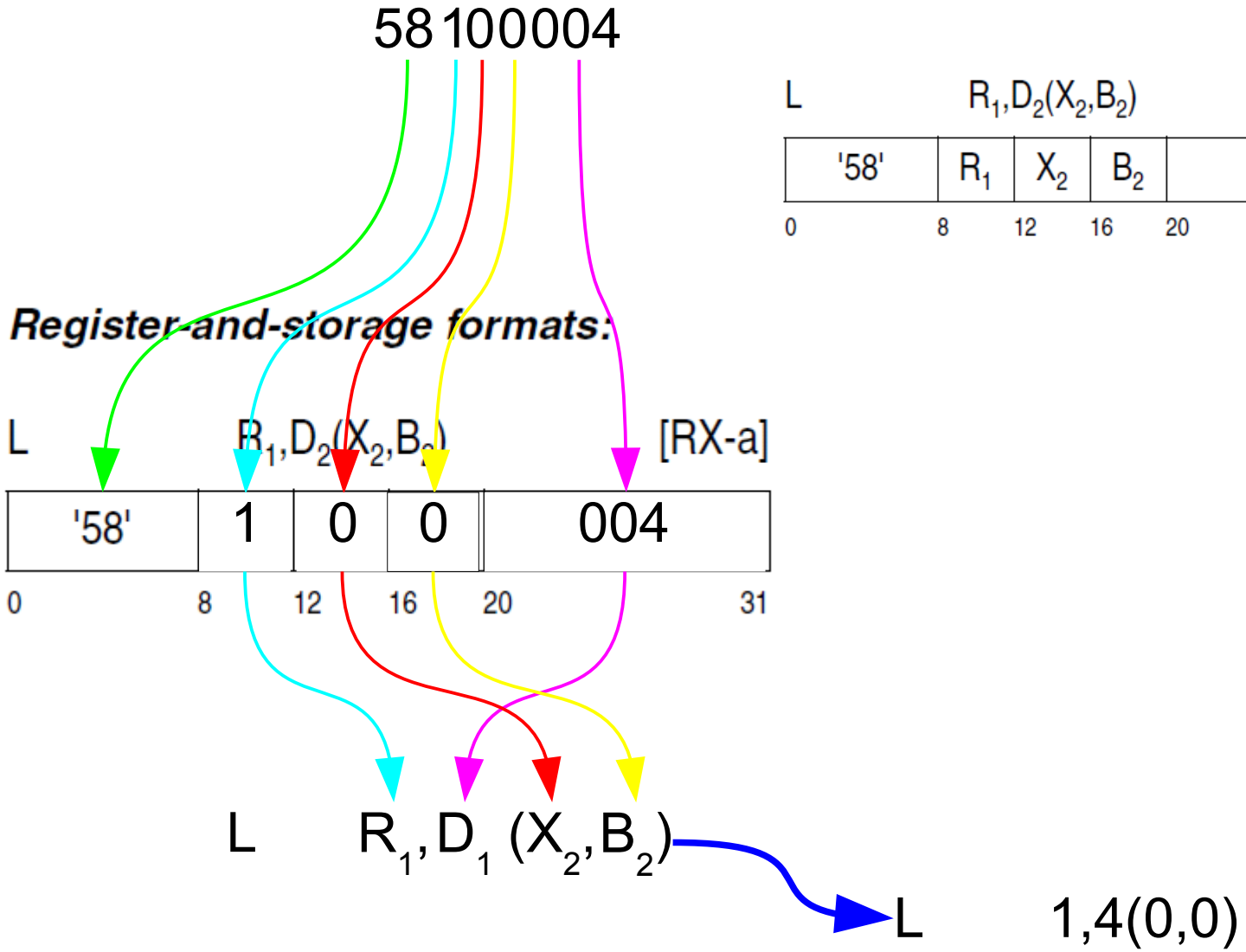
Looking at Instruction Formats

Register-and-storage formats:





Looking at Instruction Formats



Reading HLASM Listings

Reading HLASM Listings

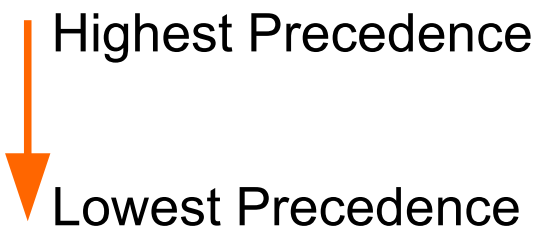
- One of the outputs produced by HLASM is the “listing” - it explains in detail all that transformations that have happened to change your source code into the produced object code.
- In certain programs, it is not possible to use a debugger and it is in these circumstances where relying on a dump and the assembler listing proves invaluable.
  - Examining a dump and a listing can also be much quicker to solve the problem than trying to work through the program with a debugger
- Unlike a lot of compilers, the HLASM listing contains much more information than just error messages that were produced when attempting to assemble your source code:

Reading HLASM Listings

- The HLASM listing is divided into:
  - High Level Assembler Option Summary
  - External Symbol Dictionary
  - Source Statements
  - Relocation Dictionary
  - Ordinary Symbol and Literal Cross Reference
  - Unreferenced Symbols Defined in CSECTs
  - Macro and Copy Code Source Summary
  - Macro and Copy Code Cross Reference
  - DSECT Cross Reference
  - Using Map
  - General Purpose Register Cross Reference
  - Diagnostic Cross Reference and Assembler Summary

Specifying options

- HLASM options can be specified in:
  - *PROCESS OVERRIDE
  - ASMAOPT
  - Invocation Parms
  - *PROCESS
  - Installation Defaults



- The location in which an option is specified since the location of each option specifies its precedence over other options.
  - Also some options cannot be specified in certain places, e.g. *PROCESS OVERRIDE VS *PROCESS

Specifying options

Our example specifies:

In source:

```
*process override(adata,mxref(full))
*process align
*process nodbcs
*process mxref(full),nolibmac
*process flag(0)
*process nofold,language(ue)
*process nora2
*process nodbcs
*process xref(full)
```

In JCL procedure - parms:

```
OPTS1='NOOBJECT,language(en),size(4meg)',
OPTS2='xref(short,unrefs)',
OPTS3='nomxref,norxref,adata,noadata'
//C    EXEC PGM=ASMA90,
//     PARM='&OPTS1,&OPTS2,&OPTS3'
```

In ASAMOPT DD:

```
//ASMAOPT  DD *
* My ASMAOPTS Overides
sysparm(thisisatestsysparm)     pass this to assembler
goff                            create GOFF object code
/*
```

HLASM Option Summary

```
                                High Level Assembler Option Summary                (PTF R160   )   Page    1
                                                                                   HLASM R6.0  2013/11/01 13.45
  Overriding ASMAOPT Parameters -
    >* My ASMAOPTS Overides
    >sysparm(thisisatestsysparm)     pass this to assembler
    >goff                            create GOFF object code
  Overriding Parameters-  NOOBJECT,language(en),size(4meg),xref(short,unrefs),nomxref,norxref,adata,noadata
  Process Statements-     override(adata,mxref(full))
                          align
                          nodbcs
                          mxref(full),nolibmac
                          flag(0)
                          nofold,language(ue)
                          nora2
                          nodbcs
                          xref(full)

** ASMA434N GOFF/XOBJECT option specified, option LIST(133) will be used
** ASMA400W Error in invocation parameter - size(4meg)
** ASMA423N Option adata, in a *PROCESS OVERRIDE statement conflicts with invocation or default option.
Option is not permitted in a *PROCESS statement and has been ignored.
** ASMA422N Option language(ue) is not valid in a *PROCESS statement.
** ASMA437N Attempt to override invocation parameter in a *PROCESS statement. Suboption full of xref
option ignored.
```

HLASM Option Summary

```
  Options for this Assembly
 3 Overriding Parms NOADATA
        5 *Process    ALIGN
                   NOASA
                   NOBATCH
                     CODEPAGE(047C)
                   NOCOMPAT
        5 *Process  NODBCS
                   NODECK
                     DXREF
                     ESD
                   NOEXIT
        5 *Process    FLAG(0,ALIGN,NOCONT,EXLITW,NOIMPLEN,NOPAGE0,PUSH,RECORD,NOSUBSTR,USING0)
        5 *Process  NOFOLD
        2  ASMAOPT    GOFF(NOADATA)
                   NOINFO
 3 Overriding Parms    LANGUAGE(EN)
        5 *Process  NOLIBMAC
                     LINECOUNT(60)
                     LIST(133)
                     MACHINE(,NOLIST)
1 *Process Override    MXREF(FULL)
 3 Overriding Parms NOOBJECT
                     OPTABLE(UNI,NOLIST)
                   NOPCONTROL
                   NOPESTOP
                   NOPROFILE
        5 *Process  NORA2
                   NORENT
                     RLD
 3 Overriding Parms NORXREF
                     SECTALGN(8)
                     SIZE(MAX)
                     TYPECHECK(MAGNITUDE,REGISTER)
                     USING(NOLIMIT,MAP,NOWARN)
                   NOWORKFILE
 3 Overriding Parms    XREF(SHORT,UNREFS)
```

External Symbol Dictionary (ESD)

```
                          External Symbol Dictionary

Symbol   Type   Id     Address  Length   Owner Id Flags Alias-of
LISTINGB  SD 00000001
B_IDRL    ED 00000002                    00000001
B_PRV     ED 00000003                    00000001
B_TEXT    ED 00000004 00000000 00000084  00000001   08
LISTINGB  LD 00000005 00000000           00000004   08
EXTERNAL_FUNCTION
          ER 00000006                    00000001
FUNCY     ER 00000007                    00000001
listme    ER 00000008                    00000001       LISTINGZ
COMMON_DATA
          SD 00000009
B_IDRL    ED 0000000A                    00000009
B_PRV     ED 0000000B                    00000009
B_TEXT    ED 0000000C 00000000 00000018  00000009   00
COMMON_DATA
          CM 0000000D 00000000           0000000C   00
```

This section of the listing contains the External Symbol Dictionary information passed to the Binder

External Symbol Dictionary

- Each entry in the ESD has a particular type:
- SD – Section Definition
  - The symbol appeared in the name field of a START, CSECT or RSECT instruction
- LD – Label Definition
  - The symbol appeared as the operand of an ENTRY statement. When you specify the GOFF assembler option on z/OS or CMS, the assembler generates an entry type of LD for each CSECT and RSECT name.
- ER – External Reference
  - The symbol appeared as the operand of an EXTRN statement or appeared as an operand of a V-type address constant.
- CM – Common control section definition
  - The symbol appeared in the name field of a COM statement

Source Statement

- This section of the listing documents source statements of the module and the resulting object code.
- The TITLE, CEJECT and EJECT assembler instructions can be used to control when the page title is printed

```
                                                  360 *
00000034                                          361 continue_code_again dc 0h'0'    define a label
00000034 9836 2014                 00000014       362          lm   r3,r6,l_regs       reload registers
00000038 58B0 A038                 00000080       363          l    r11,=f'1504'       load constant
0000003C 50B0 3014                 00000014       364          st   r11,comm_country   ... and save in common
00000040 58C0 A028                 00000070       365          l    r12,pDateFormat    load address constant
00000044 07FE                                     366          br   r14                and exit program
                                                  367 *


                                                                                                      Page    4
  Active Usings: linkage_data(X'54'),R2  common_data,R3  static_data(X'3C'),R10
R-Loc    Object Code      Addr1    Addr2    Stmt  Source Statement                HLASM R6.0  2013/11/01 15.23
00000046 0000
00000048                                          369 static_data dc 0d'0'            start of static
00000048 839697A899898788                         370 copyright   dc c'copyright IBM(UK) Ltd 2013'
00000062 0000
00000064 00000000                                 371 listingx    dc v(external_function)     declare external
00000068 E8E8E8E8D4D4C4C4                         372 dateFormat  dc cl(date_len)'YYYYMMDD'   define format
```

Source Statement – Location Counter and Statement number

```
                                                    360 *
00000034                                            361 continue_code_again dc 0h'0'    define a label
00000034 9836 2014                  00000014        362           lm  r3,r6,l_regs      reload registers
00000038 58B0 A038                  00000080        363           l   r11,=f'1504'      load constant
0000003C 50B0 3014                  00000014        364           st  r11,comm_country  ... and save in common
00000040 58C0 A028                  00000070        365           l   r12,pDateFormat   load address constant
00000044 07FE                                       366           br  r14               and exit program
                                                    367 *


                                                                                                      Page    4
  Active Usings: linkage_data(X'54'),R2  common_data,R3  static_data(X'3C'),R10
R-Loc    Object Code      Addr1    Addr2    Stmt  Source Statement                    HLASM R6.0  2013/11/01 15.23
00000046 0000
00000048                                            369 static_data dc 0d'0'            start of static
00000048 839697A899898788                           370 copyright   dc c'copyright IBM(UK) Ltd 2013'
00000062 0000
00000064 00000000                                   371 listingx    dc v(external_function)     declare external
00000068 E8E8E8E8D4D4C4C4                           372 dateFormat  dc cl(date_len)'YYYYMMDD'   define format
```

Source Statement – Location Counter and Statement number

```
  Active Usings: linkage_data(X'54'),R2  common_data,R3  static_data(X'3C'),R10
R-Loc    Object Code      Addr1    Addr2    Stmt  Source Statement               HLASM R6.0  2013/11/01 15.23

00000074 00000000                            374 pZERO       dc a(0)
00000078 00000000                            375 ListData    dc v(FUNCY)
                                             376          extrn listingz
                                             377 listingz alias c'listme'
                                             378 *
00000080                                     379          ltorg  ,
00000080 000005E0                            380               =f'1504'
                                             381 *
                       00000084              382 static_data_end equ *
                                             383 *
                                             384          drop  r2
                                             385          drop  r3
                                             386 *
00000000               00000000 00000018     387 common_data com    ,
00000000 A8A8A8A894948484                    388 comm_date    dc cl(date_len)'yyyy....
00000008 A8A8A8A894948484                    389 comm_user    dc cl(10)'yyyymmdd'
00000012 0000
00000014 00000000                            390 comm_country dc f'0000'
                                             391 *
00000000               00000000 00000054     392 linkage_data dsect ,
00000000 C140D58194854040                    393 l_name  dc cl(10)'A Name'
0000000A A8A8A8A894948484                    394 l_date  dc cl(date_len)'yyyymmdd'
```

Source Statement – Location Counter and Statement number

```
                                19          print on,gen                print statements
                                20          sysstate archlvl=1          set arch level
                                21+*        THE VALUE OF SYSSTATE IS NOW SET TO ASCENV=P AMODE64=NO
                                  +               L=1 OSREL=00000000
                                22          ieabrcx  DEFINE             use relative branching
          00000048              341         larl  r10,static_data
00000048                        342         using (static_data,static_data_end),r10
                                343 *
```

The column following the statement number contains one of these values:

A space ( ) indicates open source

A plus sign (+) indicates that the statement was generated as the result of macro call processing.

An unnumbered statement with a plus sign (+) is the result of open code substitution.

A minus sign (-) indicates that the statement was read by a preceding AREAD instruction.

An equals sign (=) indicates that the statement was included by a COPY instruction.

A greater-than sign (>) indicates that the statement was generated as the result of a preceding AINSERT instruction. If the statement is read by an AREAD instruction, this takes precedence and a minus sign is printed.

Source Statement - Addr1 and Addr2 Fields and USING Statements

```
R-Loc    Object Code      Addr1    Addr2    Stmt  Source Statement
                                               21+*        THE VALUE OF SYSSTATE IS NOW SET TO ASCENV=P..
                                               22          ieabrcx  DEFINE            use relative branching
00000000 C0A0 0000 0024           00000048    341          larl  r10,static_data
                  R:A 00000048                 342          using (static_data,static_data_end),r10
                                      343 *
00000006 17FF                                  344          xr  r15,r15           initialise register
                  R:2 00000000                 345          using (linkage_data,l_end),r2   set using scope
                  R:3 00000000                 346          using common_data,r3            set using scope
00000008 C040 0000 000A           0000001C    347          larl  r4,address_constant       address of....
0000000E D207 3000 200A 00000000  0000000A    348          mvc   comm_date,l_date   copy
00000014 9036 2014                00000014    349          stm   r3,r6,l_regs       save a copy of the reg..
00000018 A7F4 0008                00000028    350          j     continue_code      jump over constant
                                               351 *
0000001C C1C4C4D9C5E2E240                      352 address_constant dc cl12'ADDRESS'   constant value
```

Relocation Dictionary (RLD)

```
                                        Relocation Dictionary
  Pos.Id   Rel.Id   Address  Type  Action
00000004 00000004 00000070   A 4     +
00000004 00000006 00000064   V 4    ST
00000004 00000007 00000078   V 4    ST
```

```
00000048 839697A899898788                    370 copyright   dc c'copyright IBM(UK) Lt
00000062 0000
00000064 00000000                            371 listingx    dc v(external_function)
00000068 E8E8E8E8D4D4C4C4                    372 dateFormat  dc cl(date_len)'YYYYMMDD'
00000070 00000068                            373 pDateFormat dc a(dateFormat)       po
00000074 00000000                            374 pZERO       dc a(0)                po
00000078 00000000                            375 ListData    dc v(FUNCY)            de
                                             376          extrn listingz            de
```

Location counter 70, and ADCON symbol pDateFormat has a value of 00000068 – which is the location counter for symbol dateFormat

- This section of the listing describes the relocation dictionary information passed to the Binder.
- The entries describe the address constants in the assembled program that are affected by relocation.
- This section helps you find relocatable constants in your program.

Ordinary Symbol and Literal Cross Reference

```
                              Ordinary Symbol and Literal Cross Reference
Symbol   Length   Value     Id    R Type Asm  Program   Defn References
address_constant
             12 0000001C 00000004     C  C                352  347
comm_country
              4 00000014 0000000C     F  F                390  364M
comm_date
              8 00000000 0000000C     C  C                388  348M
common_data
              1 00000000 0000000C     J                   387  346U
continue_code
              2 00000028 00000004     H  H                354  350B
continue_code_again
              2 00000034 00000004     H  H                361  357B
date_len      1 00000008 00000004 A   U                     4  372   388   394


r10           1 0000000A 00000004 A   U  GR32              12  341M  342U
r11           1 0000000B 00000004 A   U  GR32              13  363M  364
r12           1 0000000C 00000004 A   U  GR32              14  365M
r14           1 0000000E 00000004 A   U  GR32              16  366B
r15           1 0000000F 00000004 A   U  GR32              17  344M  344
```

Symbol name , length, value, Assembler type, Definition

References – no suffix, **B**ranch, **M**odified, **U**sing, **D**rop and e**X**ecution

This section of the listing concerns symbols and literals that are defined and used in the program.

Unreferenced Symbols Defined in CSECTs

```
                                Unreferenced Symbols Defined in CSECTs
Defn  Symbol
 370  copyright
 375  ListData
 371  listingx
 374  pZERO
   6  r0
   7  r1
  15  r13
```

This section of the listing shows symbols that have been defined in CSECTs but not referenced. This may help you to remove unnecessary data definitions, and reduce the size of your program.

Macro and Copy Code Source Summary

```
                          Macro and Copy Code Source Summary                                  Page    8
 Con Source                          Volume    Members                          HLASM R6.0  2013/11/01 15.23
     PRIMARY INPUT                           B         BAL       BAS       BC        BCT       BE        BH
                                             BL        BM        BNE       BNH       BNL       BNM       BNO
                                             BNP       BNZ       BO        BP        BXH       BXLE      BZ
  L2 SYS1.MACLIB                     37SY01    IEABRC    IEABRCX   SYSSTATE
```

In section 'Diagnostic Cross Reference and Assembler Summary'

```
Data Sets Allocated for this Assembly
 Con DDname   Data Set Name                                  Volume  Member
  A1 ASMAOPT  SMORSA.LISTINGC.JOB63822.D0000101.?
  P1 SYSIN    SMORSA.ASM.ASM                                 37P001  LISTINGC
  L1 SYSLIB   SMORSA.ASM.ASM                                 37P001
  L2          SYS1.MACLIB                                    37SY01
     SYSLIN   SYS13305.T152320.RA000.LISTINGC.OBJ.H01
     SYSPRINT SMORSA.LISTINGC.JOB63822.D0000102.?
```

This section of the listing shows the names of the macro libraries from which the assembler read macros or copy code members, and the names of the macros and copy code members that were read from each library.

Macro and Copy Code Cross Reference

```
                                                Macro and Copy Code Cross Reference
Macro     Con  Called By         Defn  References
B              PRIMARY INPUT      117  355
BAL                               273
BAS                               297
BC             PRIMARY INPUT      177  356
BCT                               201
BE                                121
BH                                129
BL                                125
BM                                133
BNE                               137
....
BXLE                              225
BZ                                173
IEABRC    L2   IEABRCX             -  22C
IEABRCX   L2   PRIMARY INPUT       -  22
SYSSTATE  L2   PRIMARY INPUT       -  20
```

```
              19          print on,gen               print statements
              20          sysstate archlvl=1         set arch level
              21+*        THE VALUE OF SYSSTATE IS NOW SET TO ASCENV=P AMODE64=N
                +               L=1 OSREL=00000000
              22          ieabrcx  DEFINE            use relative branching
             341          larl  r10,static_data
             342          using (static_data,static_data_end),r10
```

This section of the listing shows the names of macros and copy code members and the statements where the macro or copy code member was called.

DSECT Cross Reference

```
                                                Dsect Cross Reference
Dsect     Length      Id       Defn
linkage_data
          00000054  FFFFFFFF     392
```

```
                                                391 *
00000000                  00000000 00000054     392 linkage_data dsect ,                 data passed in to me
00000000 C140D58194854040                       393 l_name  dc cl(10)'A Name'            users name
0000000A A8A8A8A894948484                       394 l_date  dc cl(date_len)'yyyymmdd'    users date
00000012 0000
00000014 0000000000000000                       395 l_regs  dc 4f'0,0,0,0'               return registers values
                          00000054              396 l_end   equ *                        end
                                                397 *
```

This section of the listing shows the names of all internal or external dummy sections defined in the program, and the number of the statement where the definition of the dummy section began.

Using Map

```
                                                    Using Map
 Stmt  -----Location-----  Action  ----------------Using-----------------
         Count      Id             Type          Value    Range     Id
  342  00000006  00000004 USING  ORDINARY    00000048 0000003C 00000004
  345  00000008  00000004 USING  ORDINARY    00000000 00000054 FFFFFFFF
  346  00000008  00000004 USING  ORDINARY    00000000 00001000 0000000C
  384  00000084  00000004 DROP
  385  00000084  00000004 DROP
```

```
Stmt  -----Location-----  Action  ------------..... Reg Max     Last Label and Using Text
        Count      Id             Type          .....     Disp    Stmt
 342  00000006  00000004 USING  ORDINARY    ..... 10 00038    365 (static_data,static_data_end),r10
 345  00000008  00000004 USING  ORDINARY    .....  2 00014    362 (linkage_data,l_end),r2
 346  00000008  00000004 USING  ORDINARY    .....  3 00014    364 common_data,r3
 384  00000084  00000004 DROP               .....  2             r2
 385  00000084  00000004 DROP               .....  3             r3
```

```
        Addr1    Addr2    Stmt  Source Statement
                00000048   341          larl  r10,static_data
  R:A 00000048             342          using (static_data,static_data_end),r10


  Active Usings: linkage_data(X'54'),R2  common_data,R3  static_data(X'3C'),R10
R-Loc    Object Code      Addr1    Addr2    Stmt  Source Statement
00000046 0000
00000048                                     369 static_data dc 0d'0'            start of
00000048 839697A899898788                    370 copyright   dc c'copyright IBM(UK) Ltd 2
00000062 0000
00000064 00000000                            371 listingx    dc v(external_function)
```

This section of the listing shows a summary of the USING, DROP, PUSH USING, and POP USING instructions used in your program.

General Purpose Register Cross Reference

```
                                  General Purpose Register Cross Reference
 Register  References (M=modified, B=branch, U=USING, D=DROP, N=index)
    0(0)    (no references identified)
    1(1)    (no references identified)
    2(2)     345U  384D
    3(3)     346U  349   362M  385D
    4(4)     347M  349   362M
    5(5)     349   362M
    6(6)     349   362M
    7(7)    (no references identified)
    8(8)    (no references identified)
    9(9)    (no references identified)
   10(A)     341M  342U
   11(B)     363M  364
   12(C)     365M
   13(D)    (no references identified)
   14(E)     366B
   15(F)     344M  344
```

Register 10 – no DROP statement?

```
  Loc    Object Code      Addr1    Addr2    Stmt  Source Statement
00000000 C0A0 0000 0024          00000048    341          larl  r10,static_data
                    R:A 00000048             342          using (static_data,static_data_end),r10
```

Register 5?

```
00000014 9036 2014               00000014    349          stm   r3,r6,l_regs

00000034 9836 2014               00000014    362          lm  r3,r6,l_regs
```

This section of the listing shows all references in the program to each of the general registers. Additional flags indicate the type of reference.

Diagnostic Cross Reference and Assembler Summary

```
                                Diagnostic Cross Reference and Assembler Summary

Statements Flagged
    24(P1,24)

      1 Statement  Flagged in this Assembly        8 was Highest Severity Code
HIGH LEVEL ASSEMBLER, 5696-234, RELEASE 6.0, PTF R160
SYSTEM: z/OS 01.13.00              JOBNAME: LISTINGB    STEPNAME: B          PROCSTEP: C
```

```
R-Loc    Object Code      Addr1    Addr2    Stmt  Source Statement
0000000C 50B0 3014               00000014     22          st  r11,comm_country
00000010 58C0 F040               00000040     23          l   r12,pDateFormat
00000014 0000 0000               00000000     24          l   r7,someData
** ASMA044E Undefined symbol - r7
** ASMA029E Incorrect register specification - r7
** ASMA044E Undefined symbol - someData
** ASMA435I Record 24 in SMORSA.ASM.ASM(LISTINGB) on volume: 37P001
00000018 07FE                                     25          br  r14
                                                  26 *
```

```
Data Sets Allocated for this Assembly
 Con DDname   Data Set Name                                  Volume  Member
  A1 ASMAOPT  SMORSA.LISTINGB.JOB63825.D0000101.?
  P1 SYSIN    SMORSA.ASM.ASM                                 37P001  LISTINGB
  L1 SYSLIB   SMORSA.ASM.ASM                                 37P001
  L2          SYS1.MACLIB                                    37SY01
     SYSLIN   SYS13305.T173626.RA000.LISTINGB.OBJ.H01
     SYSPRINT SMORSA.LISTINGB.JOB63825.D0000102.?
```

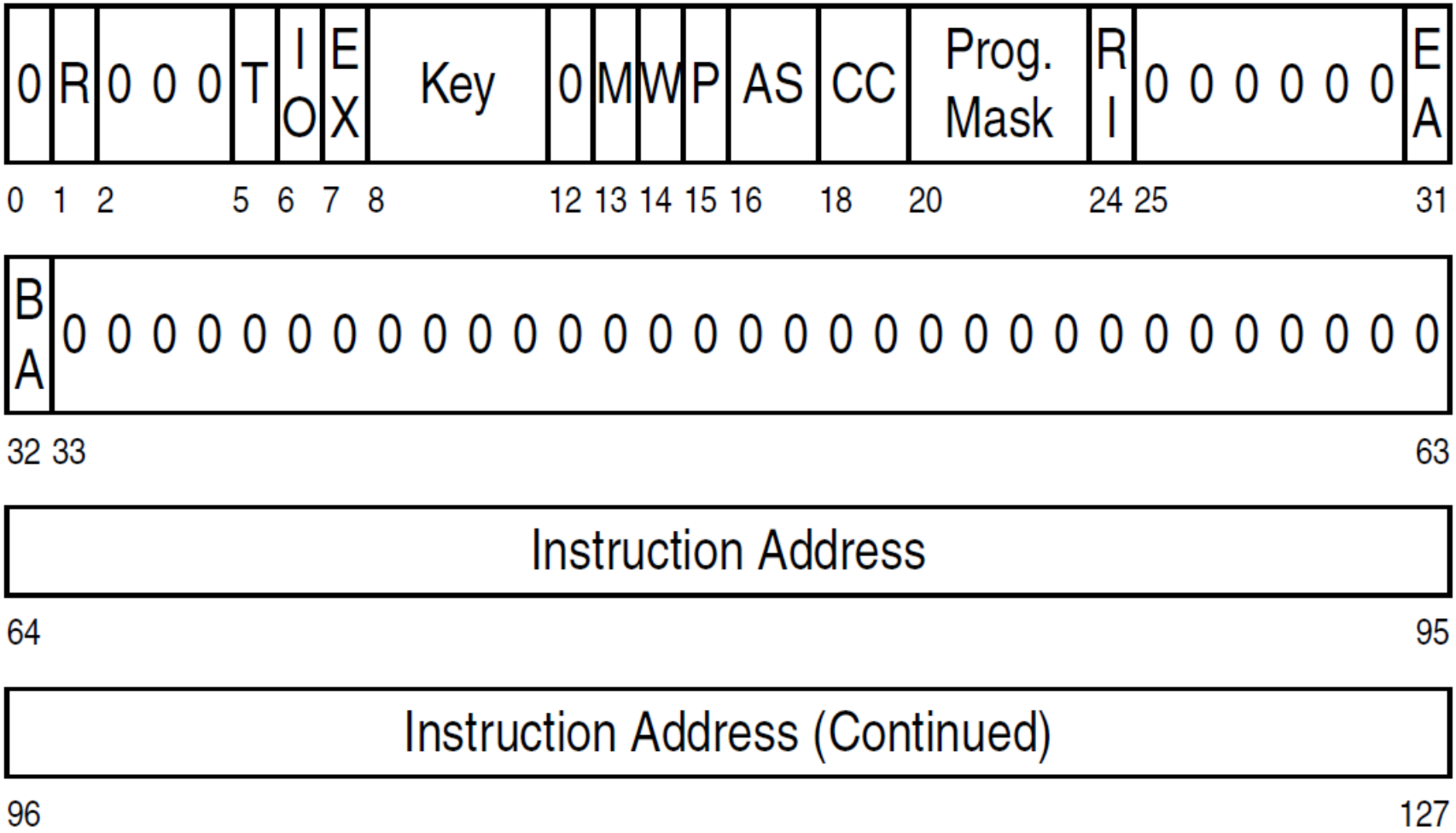
This section of the listing summarises the error diagnostic messages issued during the assembly, and provides statistics about the assembly.

Diagnostic Cross Reference and Assembler Summary

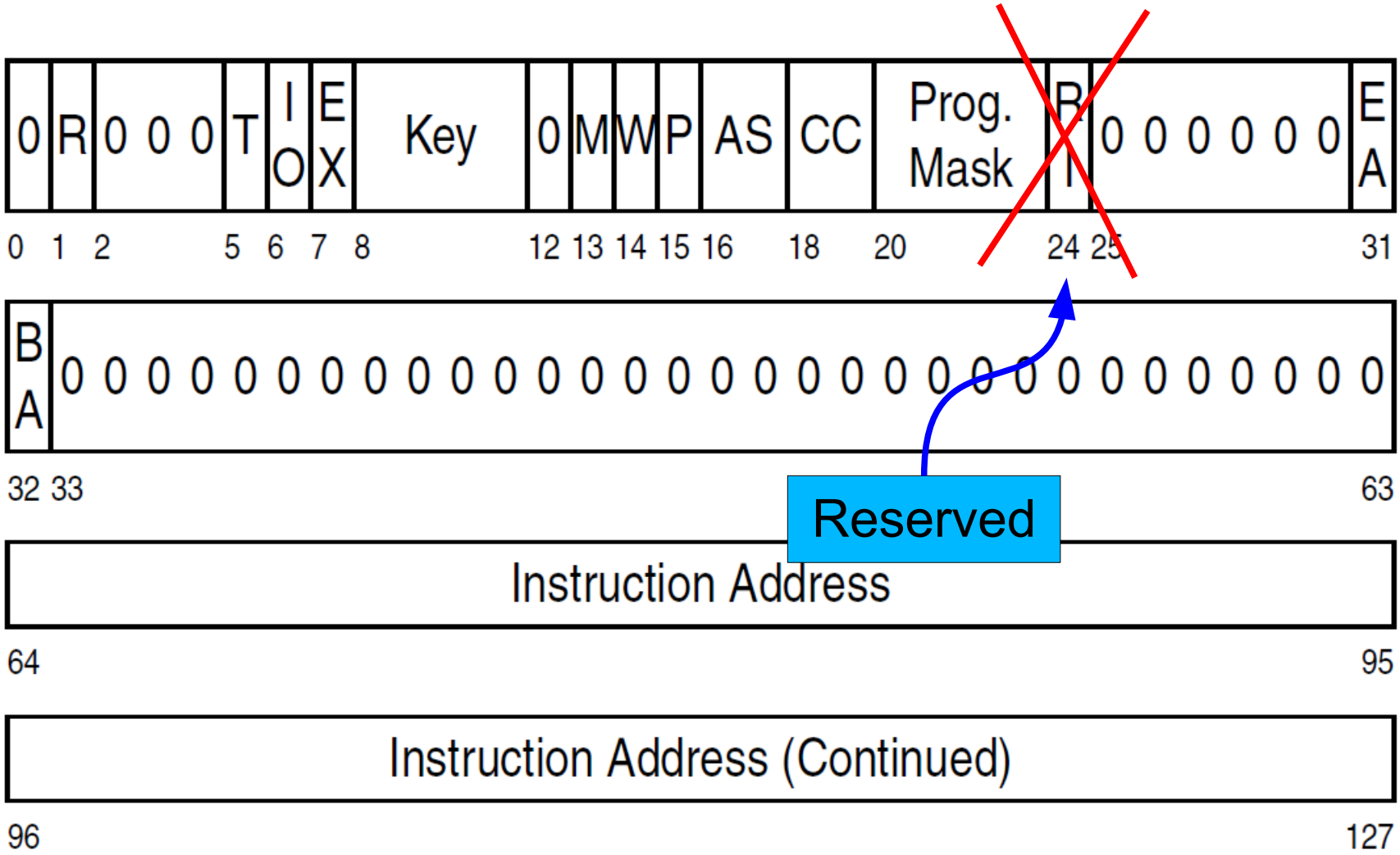
```
                                Diagnostic Cross Reference and Assembler Summary

     No Statements Flagged in this Assembly
HIGH LEVEL ASSEMBLER, 5696-234, RELEASE 6.0, PTF R160
SYSTEM: z/OS 01.13.00              JOBNAME: LISTINGC    STEPNAME: B          PROCSTEP: C
Data Sets Allocated for this Assembly
 Con DDname   Data Set Name                                 Volume  Member
  A1 ASMAOPT  SMORSA.LISTINGC.JOB63822.D0000101.?
  P1 SYSIN    SMORSA.ASM.ASM                                37P001  LISTINGC
  L1 SYSLIB   SMORSA.ASM.ASM                                37P001
  L2          SYS1.MACLIB                                   37SY01
     SYSLIN   SYS13305.T152320.RA000.LISTINGC.OBJ.H01
     SYSPRINT SMORSA.LISTINGC.JOB63822.D0000102.?

   4096K allocated to Buffer Pool       Storage required     200K
     76 Primary Input Records Read         1093 Library Records Read                0 Work File Reads
      2 ASMAOPT Records Read                312 Primary Print Records Written        0 Work File Writes
     23 Object Records Written                0 ADATA Records Written
Assembly Start Time: 15.23.20 Stop Time: 15.23.20 Processor Time: 00.00.00.0044
Return Code 000
```

The PSW and an introduction to debugging assembler programs

Program Status Word (PSW)



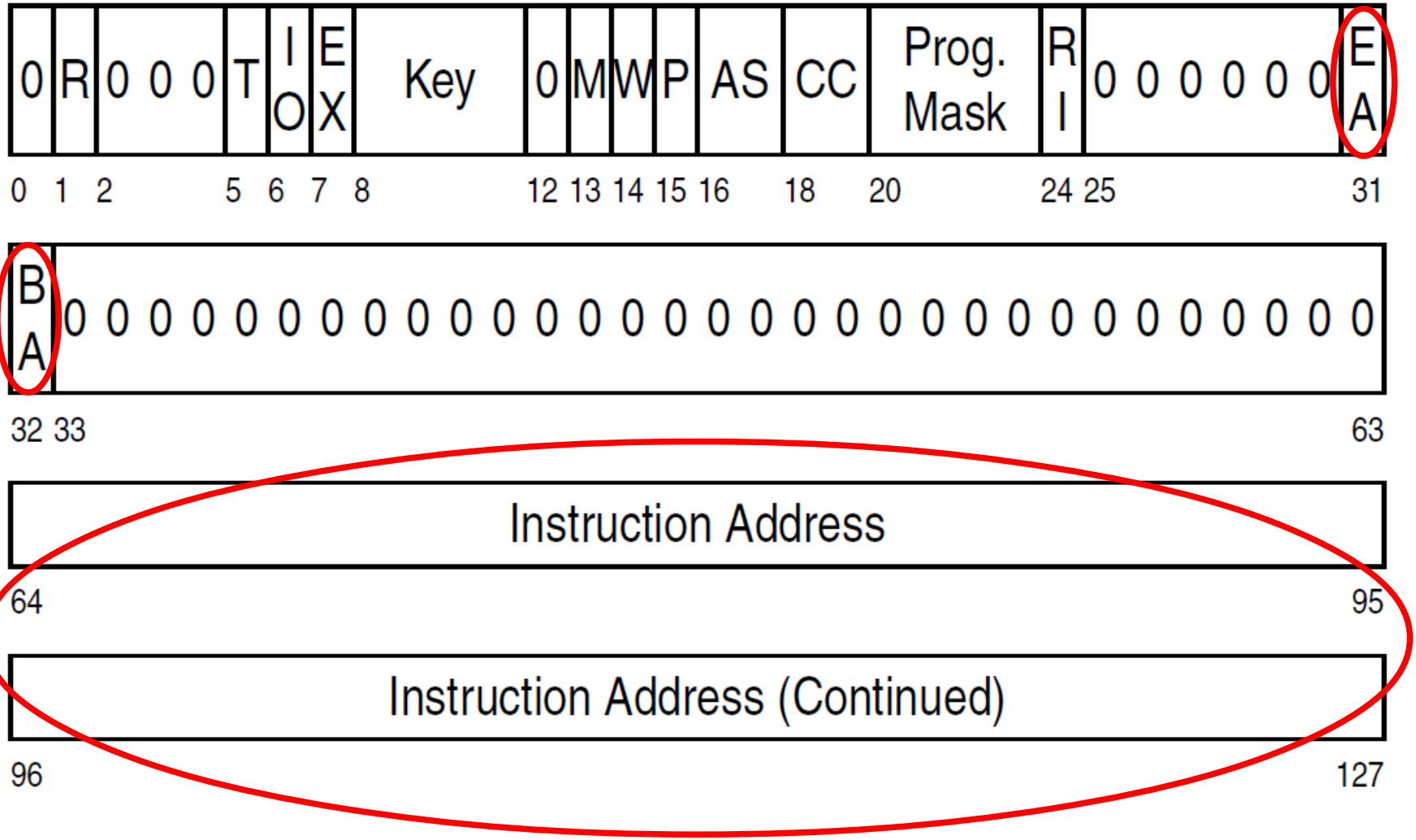
Program Status Word (PSW)



Program Status Word (PSW)

- The Program Status Word (PSW) is a register in the processor which includes control information to determine the state of the CPU.
- The z/Architecture PSW is 128-bits in length
  - Bits 0-32 contain flag bits indicating control information for the CPU
  - Bits 33-63 are 0
  - Bits 64-127 contain the instruction address
- EPSW – Extract PSW
  - Obtain bits 0-63 of the PSW and place them into operands of the instruction
- LPSW(E) – Load PSW (Extended)
  - Replace the entire PSW with the contents of storage
  - This means that the instruction branches – well might do...

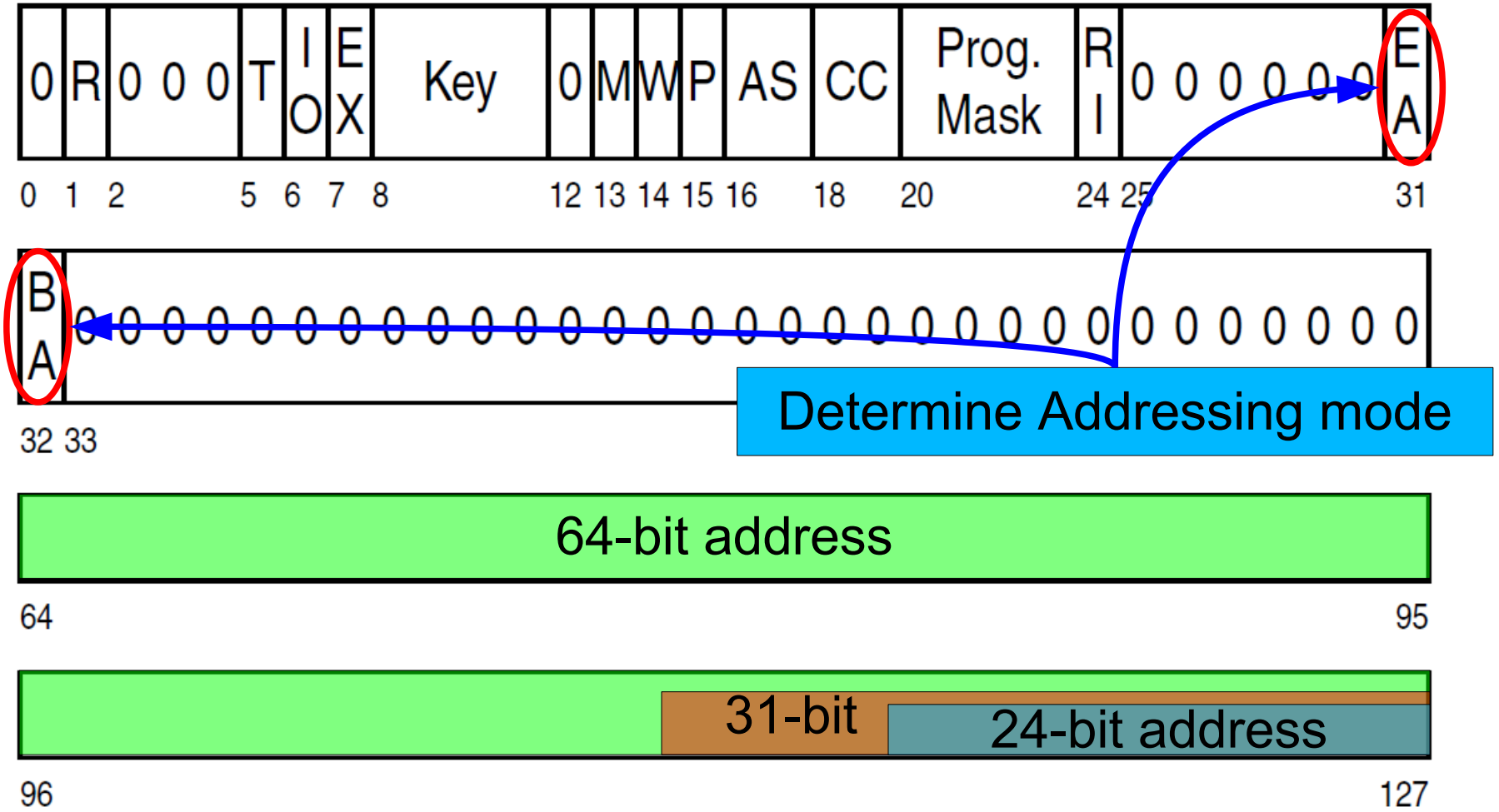
Program Status Word (PSW) – Addresses



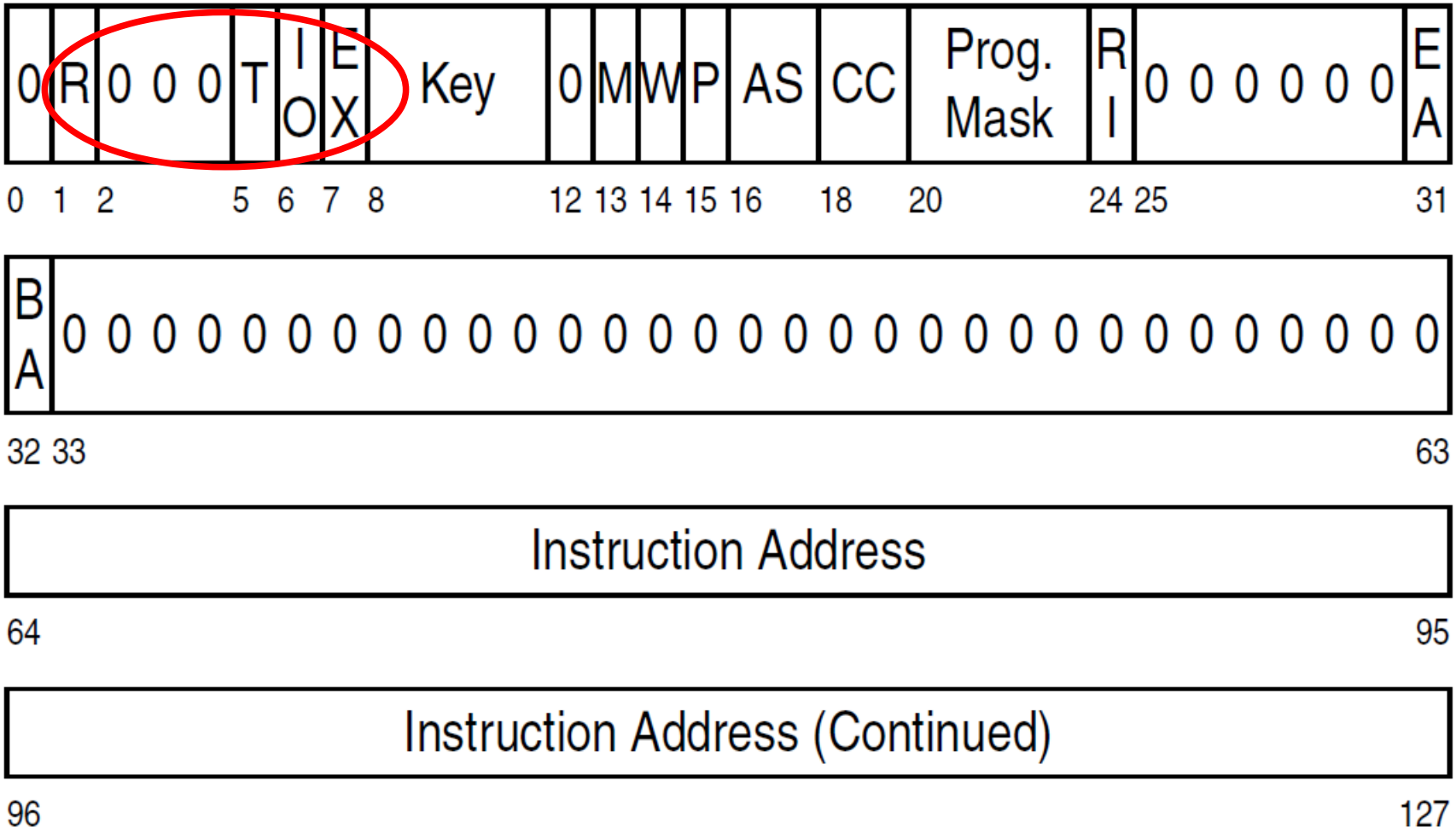
Program Status Word (PSW) – Addresses

- Since the z/Architecture can run in a number of addressing modes, the instruction address is determined by a variable number of bits in the PSW. The current addressing mode is determined by bits 31-32 of the PSW with the following combinations:
  - 00 → 24-bit mode
  - 01 → 31-bit mode
  - 10 → invalid
  - 11 → 64-bit mode
- Bits 64-127 are used to determine the address of the next instruction to be executed
  - However, some instructions may be interrupted and therefore the PSW may point at the same instruction which was being executed so that it is redriven

Program Status Word (PSW) – Instruction Address



Program Status Word (PSW) – Flag bits

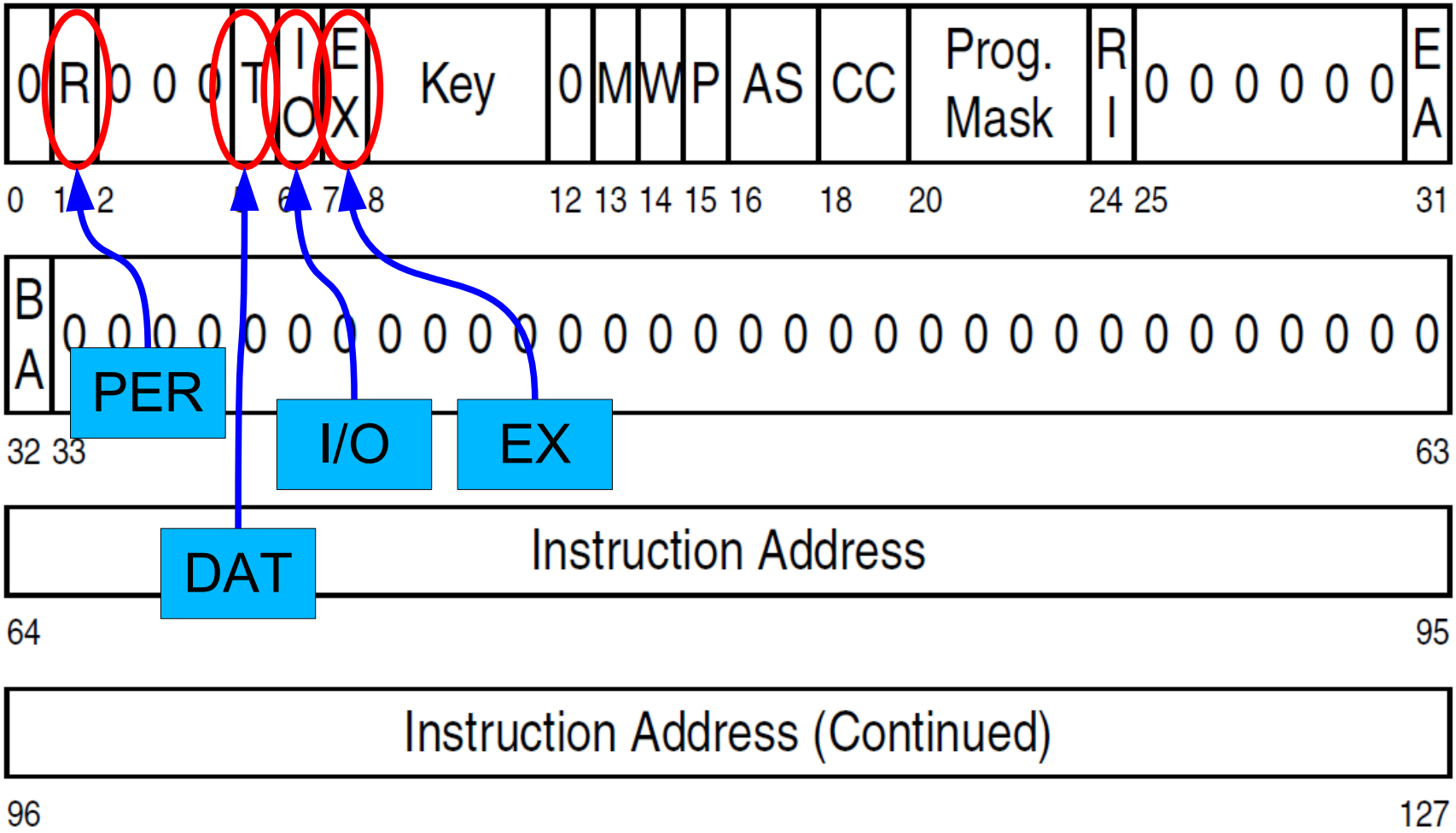


Program Status Word (PSW) – Flag bits

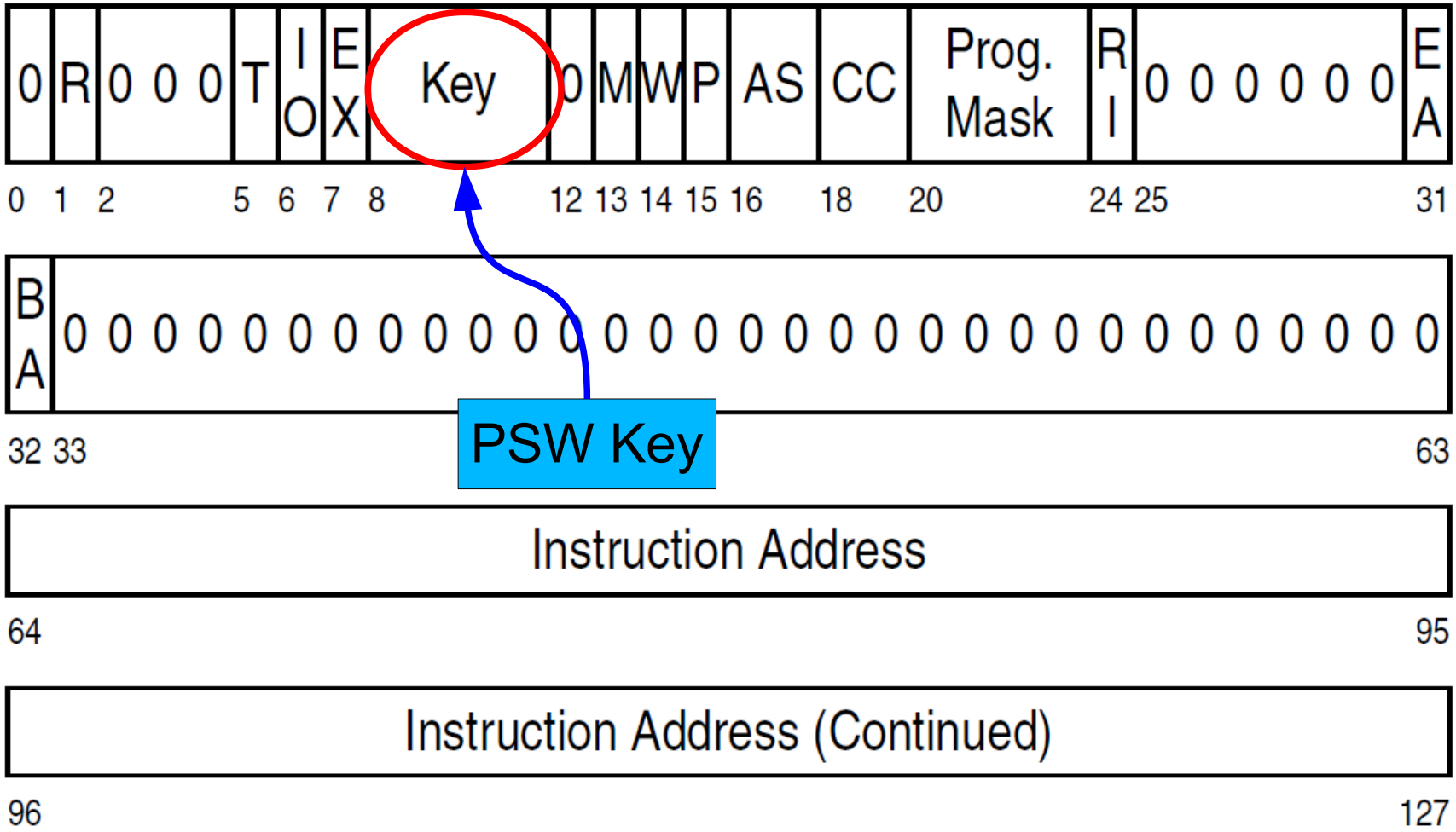
(A bit value of 1 indicates that the CPU is enabled for a function unless stated otherwise)

- Bit 1 – Program Event Recording (PER) Mask
  - Controls whether the CPU is enabled for interrupts associated with PER
- Bit 5 – DAT Mode
  - Controls whether the CPU has Dynamic Address Translation turned on
- Bit 6 – I/O Mask
  - Controls whether the CPU is enabled for interrupts
- Bit 7 – External Mask
  - Controls whether the CPU is enabled for external interrupts

Program Status Word (PSW) – Flag bits



Program Status Word (PSW) – PSW Key



Program Status Word (PSW) – PSW Key

- Bits 8-11 are used to represent the PSW Key (value of 0-15)
- PSW Keys are used to provide a security mechanism over various regions of memory with key 0 being the most secure
- Whenever an instruction attempts to access a storage location that is protected against that type of reference (read/write of storage) and the storage key does not match the access key, a protection exception is recognised.
- Programs running in PSW key 0 have read write access to storage in every storage key
- Programs in keys 1 – 15 have read access to:
  - Storage which matches their PSW key
  - Storage (in any key) that's not fetch protected
  - Storage in key 9 if the hardware feature “subsystem storage protection override” is installed
- Programs in keys 1-15 have write access to:
  - Storage whose key matches their PSW key
  - Storage in key 9 if subsystem storage protection override is installed

Program Status Word (PSW) – PSW Key – Manipulation

- IPK – Insert PSW Key
  - Used to *insert* the PSW Key *into* register 2
  - Used to store a copy of the current PSW Key typically before a switch to another key.
  - Bits 56-59 of register 2 are updated to contain the PSW Key, bits 60-63 are set to 0 and all other bits remain unmodified
- IPK cannot be used when bit 36 of CR0 is set to 0 and in problem state
- SPKA – Set PSW Key from Address
  - Used to set the PSW Key from an *address value*
  - Bits 56-59 of the 2nd operand are inserted into the PSW Key
- SPKA can only be used to set a key to which the current task is allowed to set a key determined by the PSW Key mask in CR3
- Both IPK and SPKA are privileged instructions

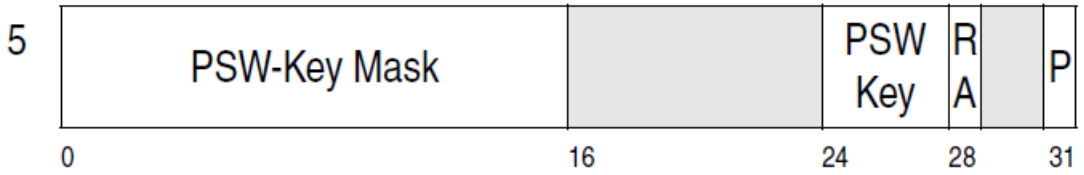
Program Status Word (PSW) – PSW Key – Manipulation

- MVCK – Move with Key
  - Moves an operand with an access key specified as part of the instruction
  - If the program is not enabled to use that access key, then a privileged operation exception is raised
  - Can be a slow instruction
- BSA – Branch and Set Authority
  - Used to branch to another place in code and set the PSW key at the same time
  - Works as a flip-flop branching from “base authority” state to “reduced authority” state

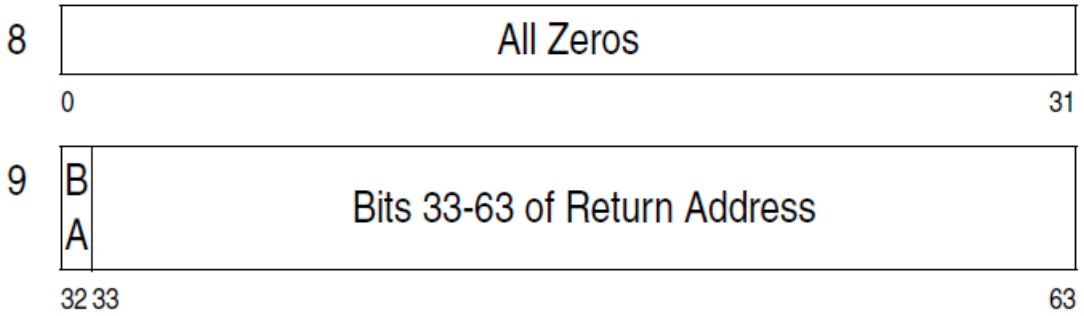
Program Status Word (PSW) – PSW Key – BSA Operation

- BSA – Branch and Set Authority – example scenario
  - A service routine, e.g. a middleware service begins in the *base authority* state
  - The routine issues a BSA to switch to running a user routine
  - The user routine runs in *reduced authority* state
  - When the user routine wants to invoke the middleware service, it issues a BSA which branches back to a fixed location in the middleware and the state is returned to running in base authority state
- The control of the states is determined by the Dispatchable Unit Control Table (DUCT)
  - The BSA instruction uses words 5, 8 and 9 of the DUCT.

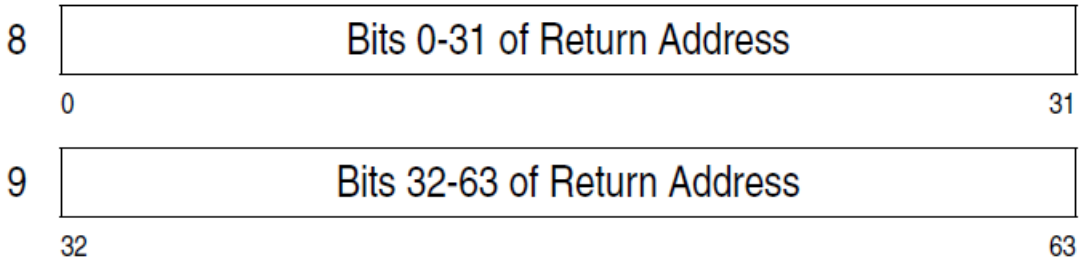
Program Status Word (PSW) – PSW Key – BSA Operation - DUCT



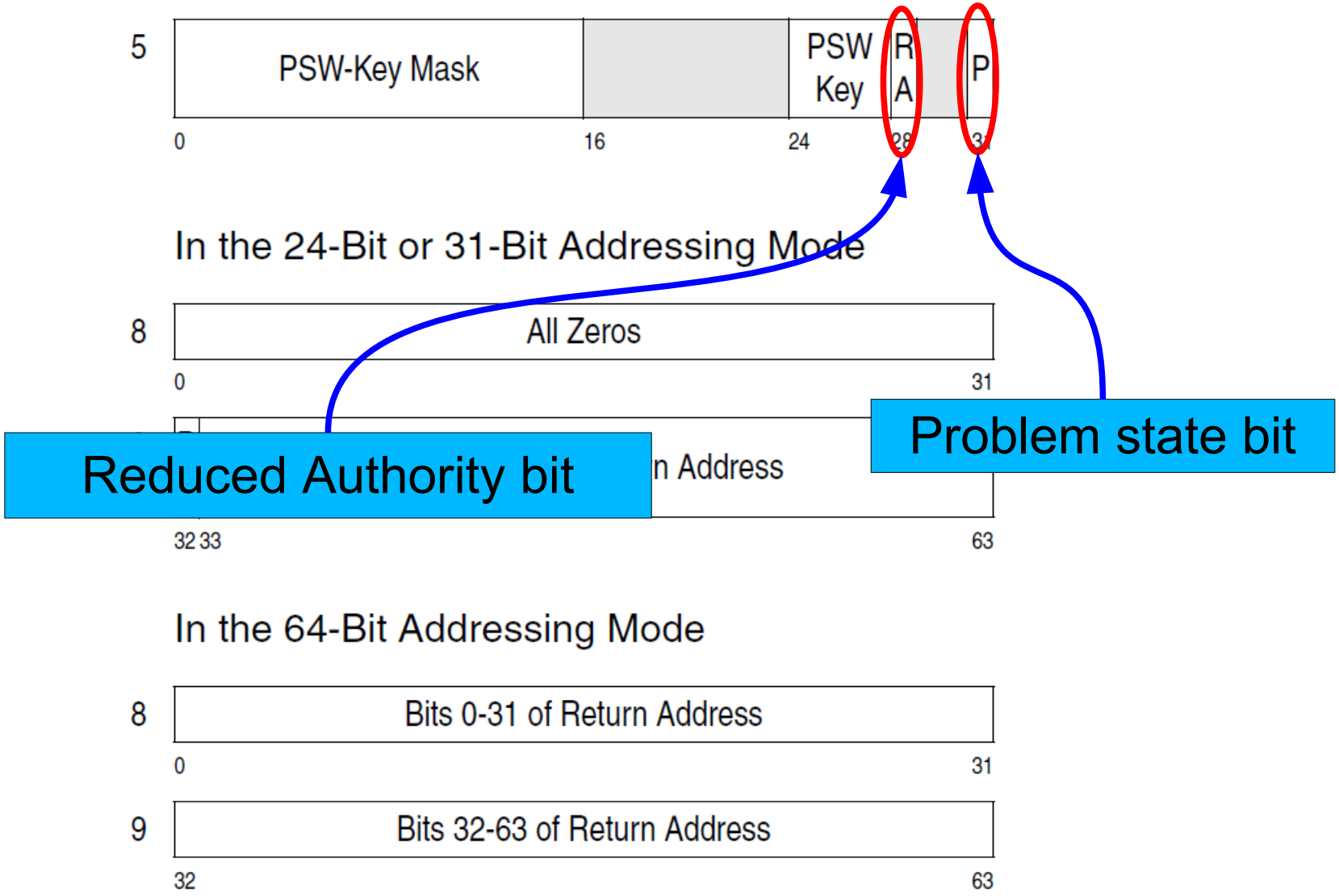
In the 24-Bit or 31-Bit Addressing Mode



In the 64-Bit Addressing Mode



Program Status Word (PSW) – PSW Key – BSA Operation



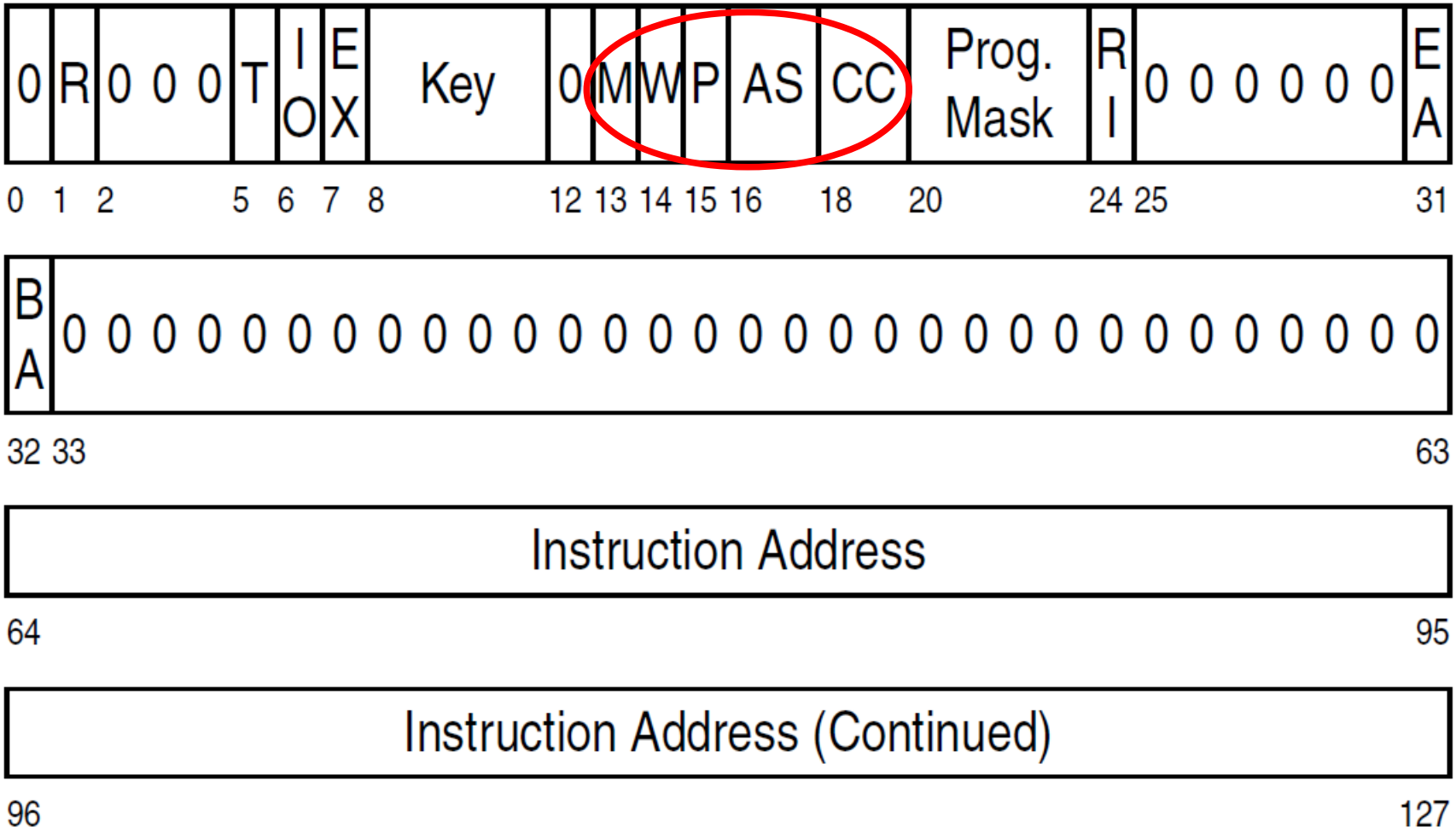
Program Status Word (PSW) – PSW Key – BSA Operation – BA

- When the BSA instruction is used in base authority, the following is stored in the DUCT:
  - The PSW-key Mask (from CR3)
  - The current PSW Key
  - Problem state bit
  - The return address
- BSA then sets the Reduced Authority bit (RA) to 1 and loads:
  - The PSW-key Mask into CR3 from operand 1
  - The PSW Key from operand 1
  - The branch address into the PSW

Program Status Word (PSW) – PSW Key – BSA Operation – RA

- When the BSA instruction is used in reduced authority, the following is restored from the DUCT:
  - The PSW-key Mask (to CR3)
  - The current PSW Key (to the PSW)
  - Problem state bit (to the PSW)
  - The return address (to the PSW – therefore the machine branches...)
- BSA then sets the Reduced Authority bit (RA) to 0

Program Status Word (PSW) – More flag bits



Program Status Word (PSW) – More flag bits

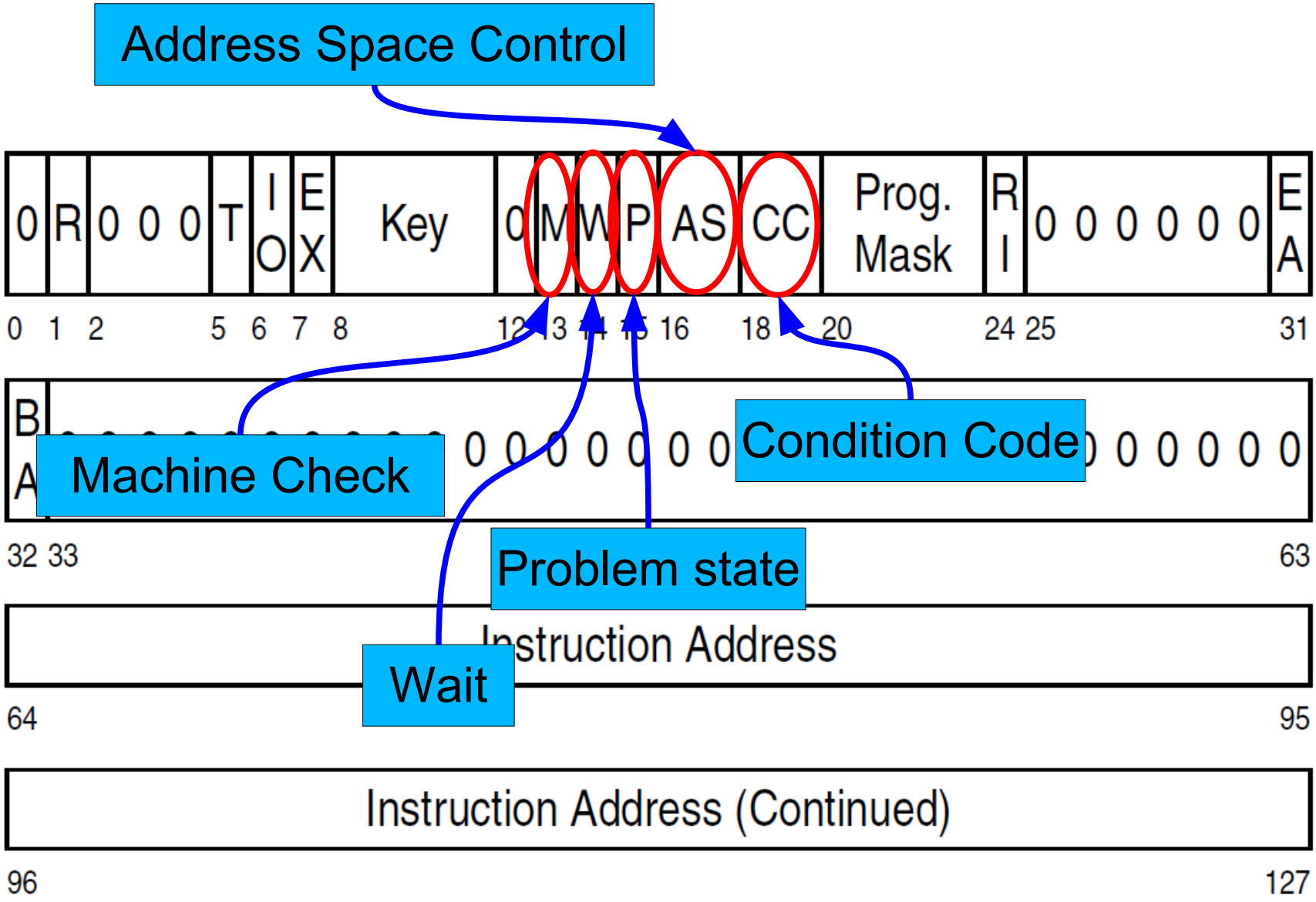
- Bit 13 – Machine Check Mask
  - Controls whether the CPU is enabled for interrupts by machine check conditions
- Bit 14 – Wait State
  - If on, the machine is waiting and no instructions are processed but interrupts may take place.
- Bit 15 – Problem State
  - The machine operates in two states – problem state (used for user code) and supervisor state (used for privileged code)
  - If an attempt is made to execute a privileged instruction in problem state, then a privileged operation exception occurs.
    Some instructions are *semi-privileged* and may or may not be permitted to execute in problem state depending on the outcome of other flags
  - All instructions are valid in supervisor state

Program Status Word (PSW) – More flag bits

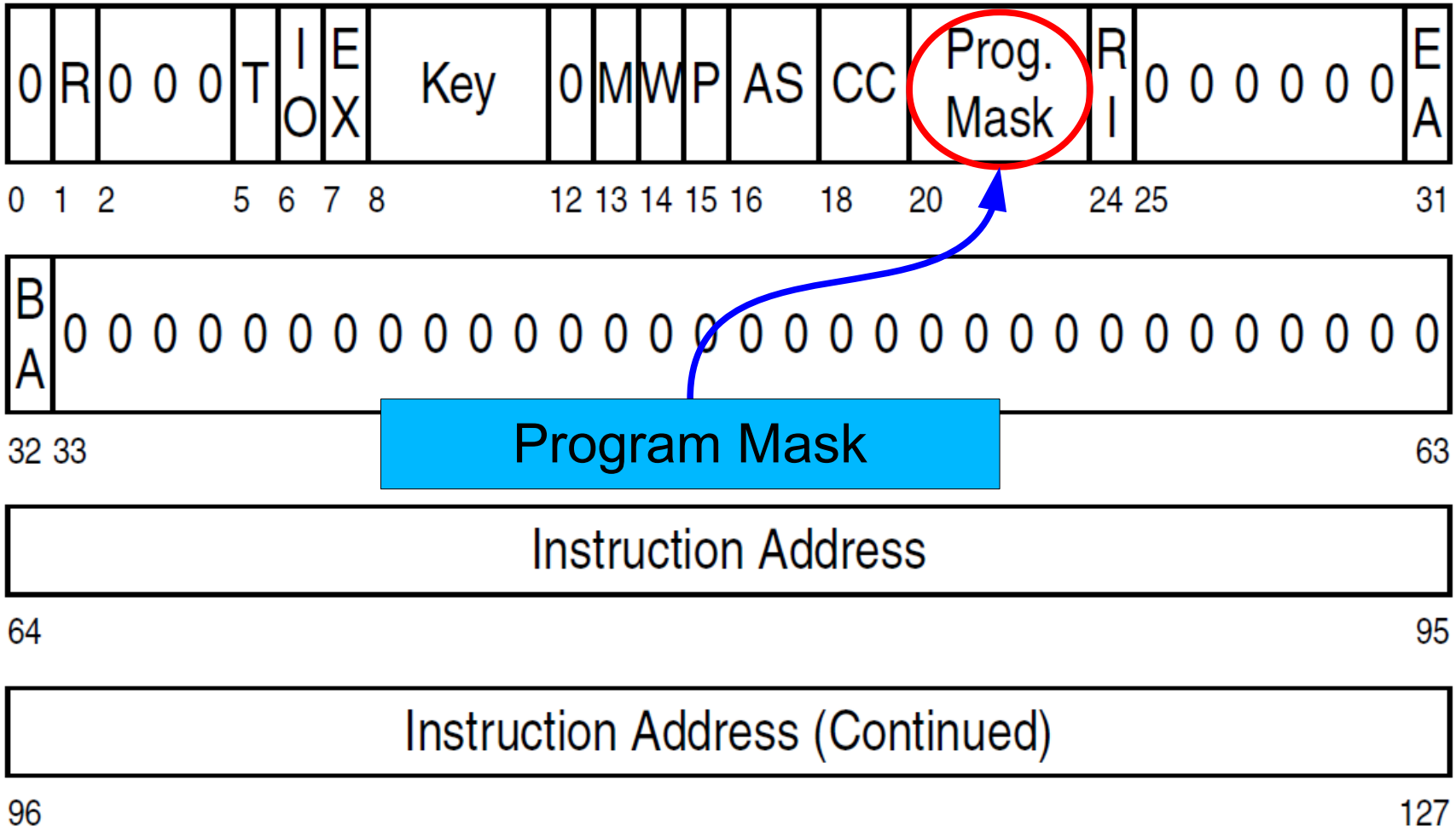
- Bit 16-17 – Address-Space Control
  - Determines how addresses are handled in conjunction with bit 5 (DAT) via the following table:

5	16	17	DAT	Mode	Instruction Addresses	Logical Addresses
0	0	0	Off	Real Mode	Real	Real
0	0	1	Off	Real Mode	Real	Real
0	1	0	Off	Real Mode	Real	Real
0	1	1	Off	Real Mode	Real	Real
1	0	0	On	Primary-space Mode	Primary virtual	Primary virtual
1	0	1	On	Access-register Mode	Primary virtual	AR specified vrt
1	1	0	On	Secondary-space Mode	Primary virtual	Secondary vrt
1	1	1	On	Home-space Mode	Home virtual	Home virtual

Program Status Word (PSW) – More flag bits



Program Status Word (PSW) – PSW Key – Program Mask



Program Status Word (PSW) – Program Mask

- Bits 20-23 – Program Mask
  - Controls a set of program exceptions
  - When the corresponding bit is on, the exception results in an interrupt

Program Mask PSW bit	Program Exception
20	Fixed-point overflow
21	Decimal overflow
22	HFP exponent underflow
23	HFP significance

- The Program Mask can be manipulated by using the instruction SET PROGRAM MASK (SPM)
- The contents of the Program Mask can be examined using the instruction INSERT PROGRAM MASK (IPM)

Program Status Word (PSW) – Using the PSW for debugging

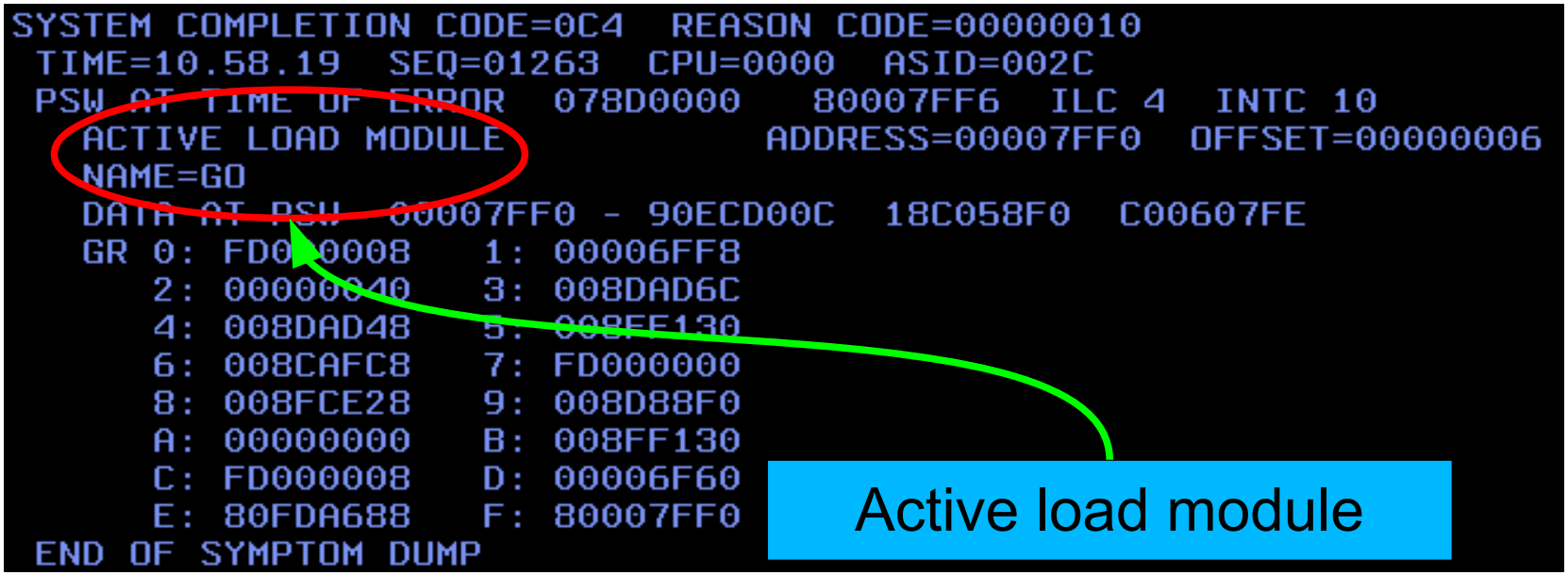
- The PSW stores invaluable information about the general state of the machine during a program's execution
- The most interesting time to examine a PSW is when something goes wrong. Even a summary dump will provide the programmer with:
  - The contents of the PSW
  - The contents of the general purpose registers
  - The next instruction to be executed

Program Status Word (PSW) – Something has gone wrong

```
SYSTEM COMPLETION CODE=0C4  REASON CODE=00000010
 TIME=10.58.19  SEQ=01263  CPU=0000  ASID=002C
 PSW AT TIME OF ERROR  078D0000   80007FF6  ILC 4  INTC 10
   ACTIVE LOAD MODULE           ADDRESS=00007FF0  OFFSET=00000006
   NAME=GO
   DATA AT PSW  00007FF0 - 90ECD00C  18C058F0  C00607FE
   GR 0: FD000008   1: 00006FF8
      2: 00000040   3: 008DAD6C
      4: 008DAD48   5: 008FF130
      6: 008CAFC8   7: FD000000
      8: 008FCE28   9: 008D88F0
      A: 00000000   B: 008FF130
      C: FD000008   D: 00006F60
      E: 80FDA688   F: 80007FF0
 END OF SYMPTOM DUMP
```

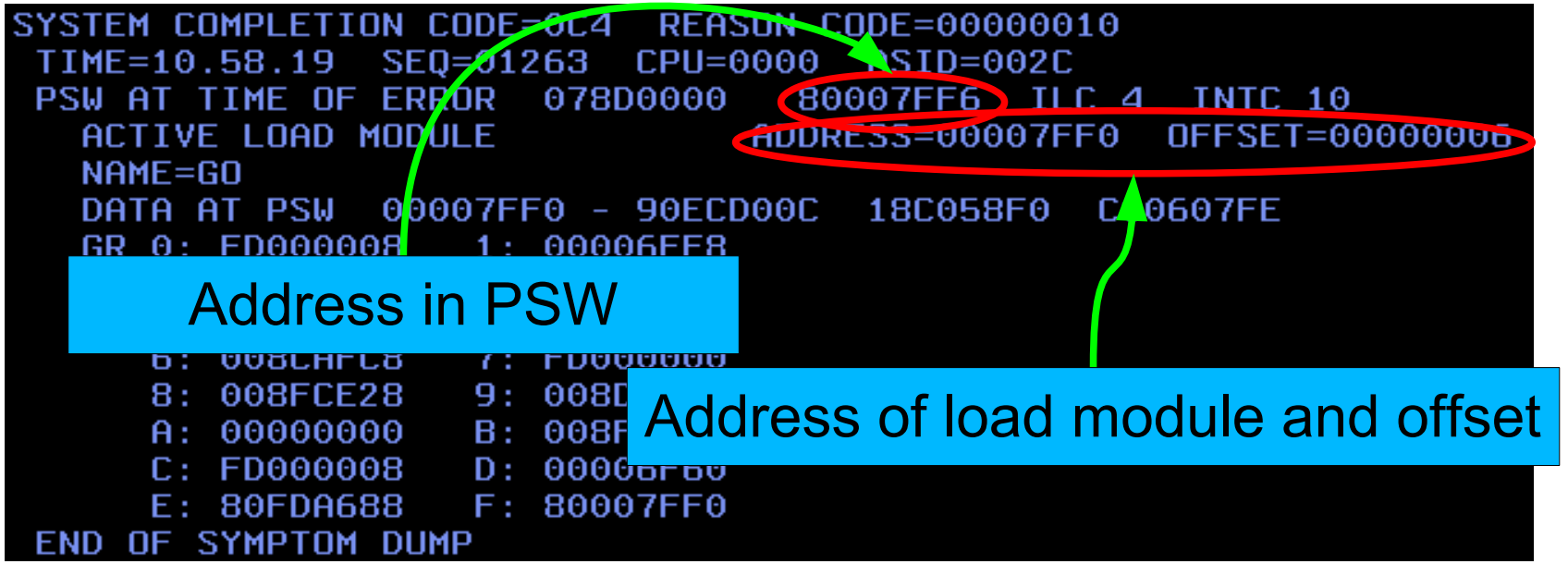
- Running a job has resulted in an 0C4 ABEND occurring. The summary dump in the job may be enough information to work out what has gone wrong.

Program Status Word (PSW) – Something has gone wrong



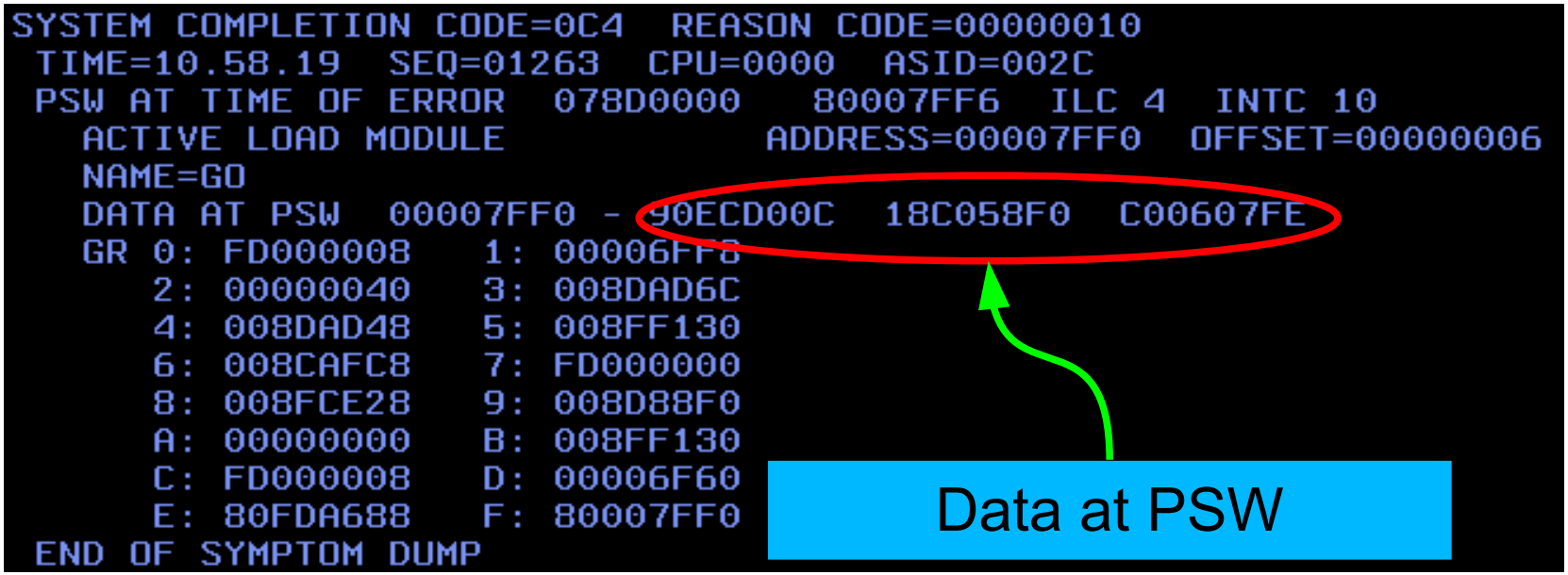
- First, look at the active load module
  - In this example, the load module name is GO since the LKEDG JCL procedure was used. From this we already know that the error occurred in our load module and not in either the assembler, linkage editor nor other part of z/OS

Program Status Word (PSW) – Something has gone wrong



- Next, double check the information in the PSW against the other information in the summary dump
  - The PSW shows that the next instruction address to be executed is X'7FF6'
  - This agrees with the data in the dump showing the address of the load module (X'7FF0') and the offset into the load module (X'0006')

Program Status Word (PSW) – Something has gone wrong



- The data at the PSW shows the instructions which were, are being, and will be executed

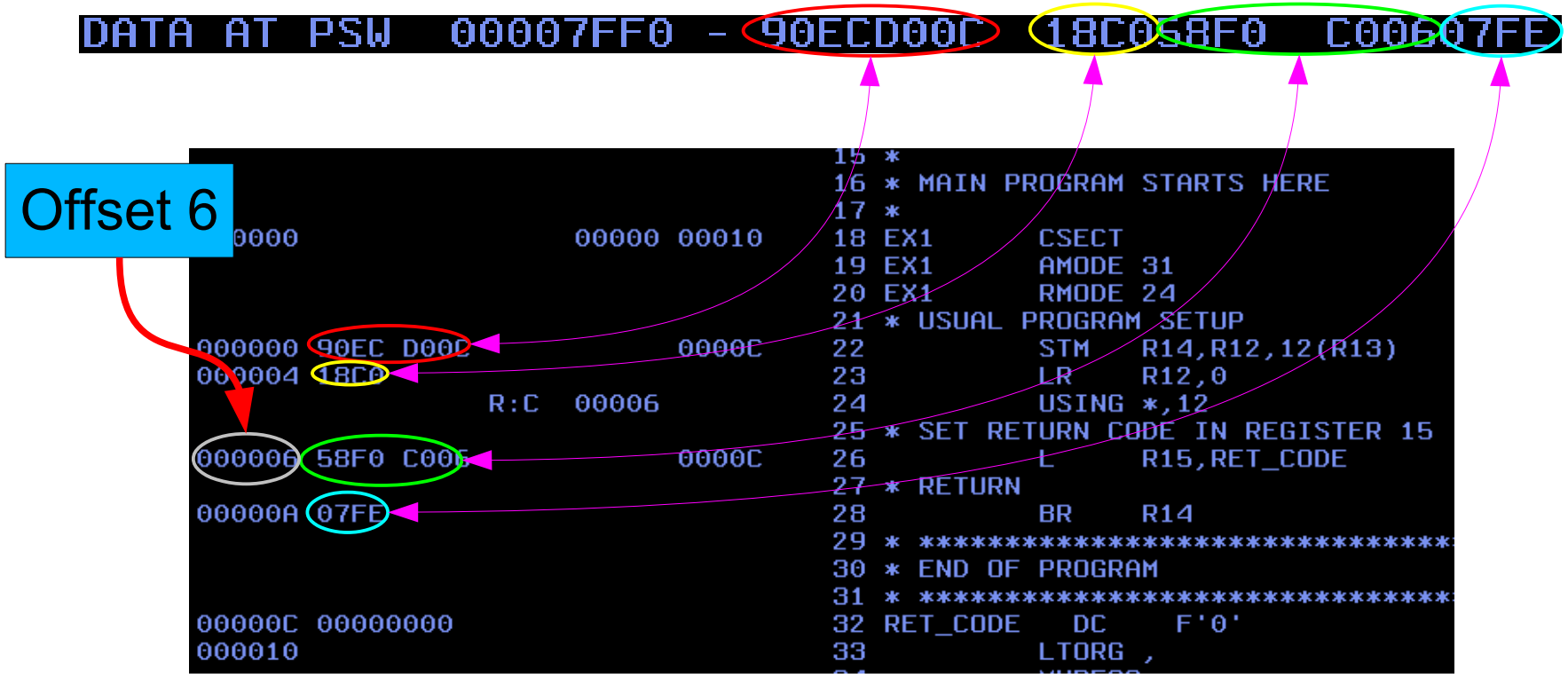
Program Status Word (PSW) – Something has gone wrong

```
DATA AT PSW  00007FF0 - 90ECD00C  18C058F0  C00607FE
```

```
                                                  15 *
                                                  16 * MAIN PROGRAM STARTS HERE
                                                  17 *
000000                  00000 00010               18 EX1      CSECT
                                                  19 EX1      AMODE 31
                                                  20 EX1      RMODE 24
                                                  21 * USUAL PROGRAM SETUP
000000 90EC D00C                0000C             22          STM   R14,R12,12(R13)
000004 18C0                                       23          LR    R12,0
                    R:C  00006                    24          USING *,12
                                                  25 * SET RETURN CODE IN REGISTER 15
000006 58F0 C006                0000C             26          L     R15,RET_CODE
                                                  27 * RETURN
00000A 07FE                                       28          BR    R14
                                                  29 * ********************************
                                                  30 * END OF PROGRAM
                                                  31 * ********************************
00000C 00000000                                   32 RET_CODE   DC    F'0'
000010                                            33          LTORG ,
```

- Examining the program listing at offset 6 shows where the error occurred. Using the data at the PSW and looking at the machine code generated by HLASM in the listing confirms this and that so far our diagnosis of the problem is correct

Program Status Word (PSW) – Something has gone wrong



- Examining the program listing at offset 6 shows where the error occurred. Using the data at the PSW and looking at the machine code generated by HLASM in the listing confirms this and that so far our diagnosis of the problem is correct
- We now know the instruction which caused the error was:
  58F0 C006 → L R15,RET_CODE

Program Status Word (PSW) – Something has gone wrong

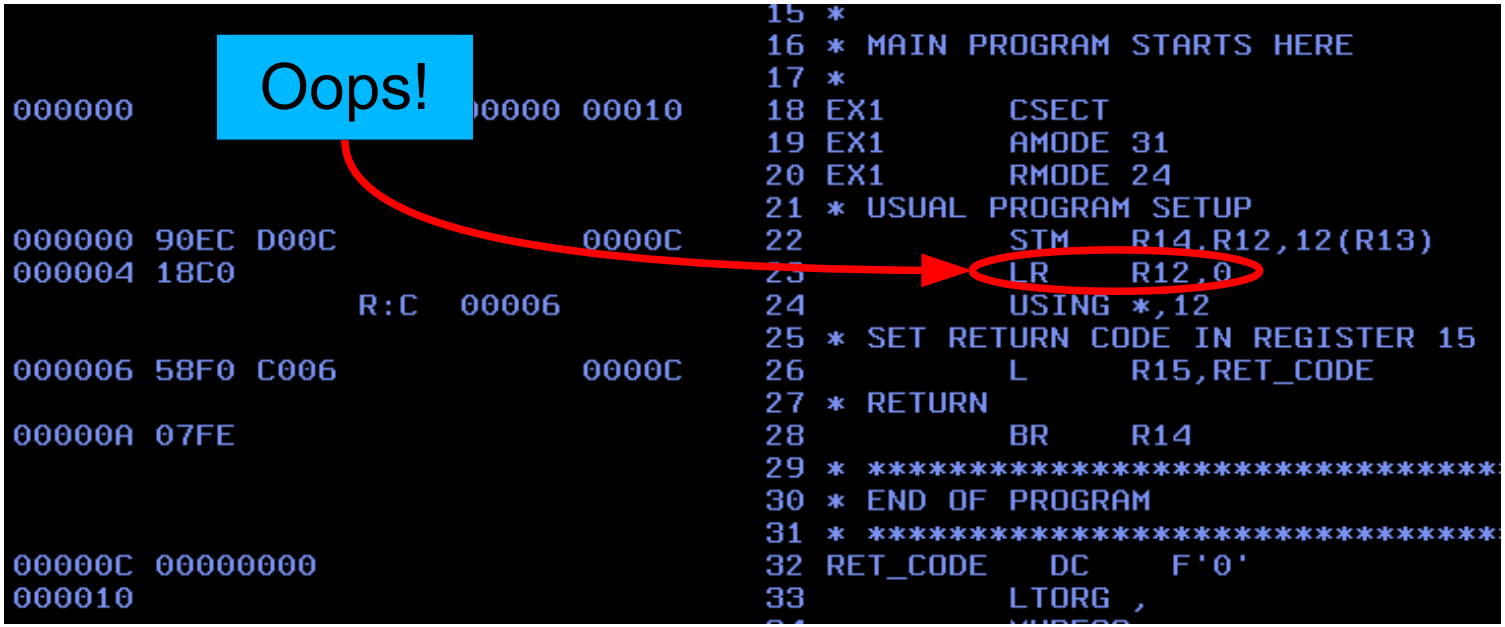
- At this stage of our debugging we know:
  - The load module name that caused the error
  - The offset into the load module at which the error occurred
- We have also double-checked that what was printed in the summary dump is confirmed by the data at the PSW
- Examining the instruction at fault, we determine the following:
  - 58F0 C006 → L R15,RET_CODE
  - 58 – OPCODE = LOAD
  - F – Register 15, the register to be loaded
  - 0 – Index register (unused since it has a value of 0)
  - C – Base register is register 12
  - 006 – Displacement from the base register from which the data will be loaded
- So, the instruction is attempting to load register 15 with the contents of memory at an offset of 6 from register 12.

Program Status Word (PSW) – Something has gone wrong

```
SYSTEM COMPLETION CODE=0C4  REASON CODE=00000010
 TIME=10.58.19  SEQ=01263  CPU=0000  ASID=002C
 PSW AT TIME OF ERROR  078D0000   80007FF6  ILC 4  INTC 10
   ACTIVE LOAD MODULE           ADDRESS=00007FF0  OFFSET=00000006
   NAME=GO
   DATA AT PSW  00007FF0 - 90ECD00C  18C058F0  C00607FE
   GR 0: FD000008   1: 00006FF8
      2: 00000040   3: 008DAD6C
      4: 008DAD48   5: 008FF130
      6: 008CAFC8   7: FD000000
      8: 008FCE28   9: 008D88F0
      A: 00000000   B: 008FF130
      C: FD000008   D: 00006F60
      E: 80FDA688   F: 80007FF0
 END OF SYMPTOM DUMP
```

- The summary dump also shows the contents of the general purpose registers
- The value in register 12 is X'FD000008'
- The instruction at fault is attempting to load a value from address X'FD00000E' – which is unaddressable by our program and therefore the cause of the error
- Note that the value of register 12 is the same as the value of register 0...

Program Status Word (PSW) – Something has gone wrong



- Looking back through the program, we can see that register 12 was loaded with the value of register 0 during program startup
- It looks as if the programmer made a typo and instead of using LR 12,0 should have used BALR 12,0 in order to load the address of the next instruction into register 12. This would make sense since they are using register 12 to establish addressability to the program's data
- Correcting this mistake fixes the program

Summary

- Introductory topics
  - Computer organisation and z/Architecture
  - Building programs on z Systems
  - Working with HLASM
- Programming in Assembler
  - Loading, storing and moving data
  - Manipulating data – logic and arithmetic
  - Making decisions
  - Branching and looping
  - Reading Principles of Operation
- Reading the HLASM Listing
- The PSW and an introduction to debugging assembler programs

Where can I get help?

- z/OS V2R1 Elements and Features
  http://www.ibm.com/systems/z/os/zos/bkserv/v2r1pdf/#IEA
- HLASM Programmer's Guide (SC26-4941-06)
  http://publibz.boulder.ibm.com/epubs/pdf/asmp1021.pdf
- HLASM Language Reference (SC26-4940-06)
  http://publibz.boulder.ibm.com/epubs/pdf/asmr1021.pdf
- z/Architecture Principles of Operation
  http://www.ibm.com/support/docview.wss?uid=isg2b9de5f05a9d57819852571c500428f9a

Putting it all together

Putting it all together

- The following slides show a small demo program which determines whether or not an employee is eligible for a pay increase
- The slides are ordered as:
  - JCL to assemble, bind and run the program called SALARY
  - Assembler source code for the SALARY program
  - Job output from the program

Putting it all together

JCL to assemble, bind and run SALARY program

```
//XXXXXXX  JOB NOTIFY=&SYSUID
//S1       EXEC PGM=ASMA90
// SET PRGNM=SALARY
//* ************************************************************
// SET     SRCLIB=&SYSUID..STAGE1.ASM0.ANSWERS
//SYSPRINT DD SYSOUT=*
//SYSLIN   DD DSN=&&TEMP,DISP=(,PASS),SPACE=(CYL,1)
//SYSIN    DD DSN=&SRCLIB(&PRGNM),DISP=SHR
//SYSLIB   DD DSN=SYS1.MACLIB,DISP=SHR
//         DD DSN=PP.HLASM.ZOS201.SASMMAC1,DISP=SHR
//         DD DSN=PP.HLASM.ZOS201.SASMMAC2,DISP=SHR
//S2       EXEC LKEDG,COND=(8,LE),
//             PARM.LKED='XREF,LIST,NCAL,MAP'
//SYSLIN   DD DISP=OLD,DSN=*.S1.SYSLIN
//SYSPRINT DD SYSOUT=*
//GO.SYSUDUMP DD SYSOUT=*
//* ************************************************************
//* ************************************************************
//* ************************************************************
```

Putting it all together

Assembler source code for SALARY program (1)

```
*********************************************************************
* PUTTING IT ALL TOGETHER PROGRAM
*********************************************************************
*
* The purpose of this small demo program is to demonstrate some small
* parts of assembler programming.
* The demo pretends that it has been passed an employee record via
* register 1.
* It will copy this record to some working storage and then proceed
* to determine whether or not the employee is eligible for a pay
* increase by comparing the employee's annual salary to the target
* salary.
*
* The employee's annual salary is calculated as:
*    12 x (MONTHLY_PAY-BENEFITS) + BONUS
*
*
         ASMDREG ,
SALARY   CSECT
SALARY   AMODE 31
SALARY   RMODE 24
* USUAL PROGRAM SETUP
         STM   14,12,12(13)
         BALR  12,0
         USING *,12
* Point register 1 at the first employee to process...
         la    r1,employee_id_1
```

Putting it all together

Assembler source code for SALARY program (2)

```
*
* The data for the employee record is passed in from register 1.
* This small program will determine whether or not the employee
* is eligable for a pay rise or not.
*
         xc    employee(employee_rec_len),employee         Clear WS
         mvc   employee(employee_rec_len),0(r1)   Copy record to WS
*
* Output the name of the employee that is being processed
*
         xc    wto_text,wto_text           Clear text buffer
         mvc   wto_text(l'process_text),process_text
         mvc   wto_text+l'process_text(l'employee_name),employee_name
* Calculate the amount of text to output
         lhi   r5,l'process_text+l'employee_name
         sth   r5,wto_buf                  Store length in buffer
         la    r5,wto_buf                  Load address of bufffer
         WTO   TEXT=(5)                    Output text
*
* Calculate employee's yearly pay as 12*(MONTHLY_PAY-BENEFITS)+BONUS
* We will use register 3 as a work register
*
         l     r3,employee_monthly_pay
         s     r3,employee_benefits        MONTHLY-BENEFITS
         m     r2,=f'12'                   Multiply by 12
         ah    r3,employee_bonus           Add yearly bonus
         c     r3,target_salary            Compare total with target
         bl    deserves_increase
```

Putting it all together

Assembler source code for SALARY program (3)

```
         WTO   'Employee has matched or exceeded target salary'
         b     resume_code
deserves_increase    dc     0h
         WTO   'Employee deserves a pay increase'
*
* Return to the caller of the program
*
resume_code          dc     0h
         LM    14,12,12(13)
*
         XR    15,15
         BR    14
* ****************************************************************
* END OF PROGRAM - DATA FOLLOWS
* ****************************************************************
*
WTO_BUF              DC H'0'
WTO_TEXT             DS CL256
PROCESS_TEXT         DC C'Processing employee '
* SALARY SCHEME DATA
TARGET_SALARY        DC F'24000'            TARGET SALARY FOR COMPANY
* EMPLOYEE RECORD STRUCTURE
EMPLOYEE             DC 0F
EMPLOYEE_NAME        DS CL40                   EMPLOYEE'S NAME
EMPLOYEE_MONTHLY_PAY DS F                      VALUE OF MONTHLY PAY
EMPLOYEE_BONUS       DS H                      YEARLY BONUS AMOUNT
EMPLOYEE_BENEFITS    DS F                      MONTHLY BENEFITS
EMPLOYEE_REC_LEN     EQU *-EMPLOYEE            SIZE OF EMPLOYEE RECORD
```

Putting it all together

Assembler source code for SALARY program (4)

```
* EMPLOYEE EXAMPLE DATA
EMPLOYEE_ID_1         DC    0F
                      DC CL40'BOB SMITH'
                      DC F'2000'
                      DC H'1000'
                      DC F'50'
          LTORG ,
          END
```

Putting it all together

Job output for assembling, binding and running SALARY program (1)

```
1                J E S 2  J O B  L O G  --  S Y S T E M  M V 3 3  --  N O D E  W I N M V S 3 3
0
 21.20.35 JOB47759 ---- TUESDAY,   29 APR 2014 ----
 21.20.35 JOB47759  IRR010I  USERID XXXXXXX  IS ASSIGNED TO THIS JOB.
 21.20.35 JOB47759  IEF677I WARNING MESSAGE(S) FOR JOB XXXXXXX  ISSUED
 21.20.35 JOB47759  ICH70001I XXXXXXX  LAST ACCESS AT 21:00:58 ON TUESDAY, APRIL 29, 2014
 21.20.35 JOB47759  $HASP373 XXXXXXX  STARTED - INIT 1    - CLASS A        - SYS MV33
 21.20.35 JOB47759  IEF403I XXXXXXX - STARTED
 21.20.35 JOB47759  -                                        --TIMINGS (MINS.)--            ----PAGING COUNTS---
 21.20.35 JOB47759  -JOBNAME  STEPNAME PROCSTEP    RC   EXCP    CPU    SRB  CLOCK    SERV  PG    PAGE    SWAP     VIO SWAPS STEPNO
 21.20.35 JOB47759  -XXXXXXX           S1          00     95    .00    .00    .00     615   0       0       0       0     0      1
 21.20.35 JOB47759  -XXXXXXX  S2       LKED        00     28    .00    .00    .00     191   0       0       0       0     0      2
 21.20.35 JOB47759  +Processing employee BOB SMITH
 21.20.35 JOB47759  +Employee has matched or exceeded target salary
 21.20.35 JOB47759  -XXXXXXX  S2       GO          00      4    .00    .00    .00      51   0       0       0       0     0      3
 21.20.35 JOB47759  IEF404I XXXXXXX - ENDED
 21.20.35 JOB47759  -XXXXXXX  ENDED.  NAME-                     TOTAL CPU TIME=   .00  TOTAL ELAPSED TIME=   .00
 21.20.35 JOB47759  $HASP395 XXXXXXX  ENDED
0------ JES2 JOB STATISTICS ------
-  29 APR 2014 JOB EXECUTION DATE
-           24 CARDS READ
-          669 SYSOUT PRINT RECORDS
-            0 SYSOUT PUNCH RECORDS
-           37 SYSOUT SPOOL KBYTES
-         0.00 MINUTES EXECUTION TIME
         1 //XXXXXXX  JOB NOTIFY=&SYSUID                                    JOB47759
           IEFC653I SUBSTITUTION JCL - NOTIFY=XXXXXXX
         2 //S1       EXEC PGM=ASMA90
           //* ***************************************************************
           //* CHANGE THE FOLLOWING LINE TO REFLECT THE PROGRAM NAME
           //* ***************************************************************
```

Putting it all together

Job output for assembling, binding and running SALARY program (2)

```
          3 // SET PRGNM=SALARY
            //* ****************************************************************
            //* ****************************************************************
            //* ****************************************************************
          4 // SET     SRCLIB=&SYSUID..STAGE1.ASM0.ANSWERS
            IEFC653I SUBSTITUTION JCL - SRCLIB=XXXXXXX.STAGE1.ASM0.ANSWERS
          5 //SYSPRINT DD SYSOUT=*
          6 //SYSLIN   DD DSN=&&TEMP,DISP=(,PASS),SPACE=(CYL,1)
          7 //SYSIN    DD DSN=&SRCLIB(&PRGNM),DISP=SHR
            IEFC653I SUBSTITUTION JCL - DSN=XXXXXXX.STAGE1.ASM0.ANSWERS(SALARY),DISP=SHR
          8 //SYSLIB   DD DSN=SYS1.MACLIB,DISP=SHR
          9 //         DD DSN=PP.HLASM.ZOS201.SASMMAC1,DISP=SHR
         10 //         DD DSN=PP.HLASM.ZOS201.SASMMAC2,DISP=SHR
         11 //S2       EXEC LKEDG,COND=(8,LE),
            //             PARM.LKED='XREF,LIST,NCAL,MAP'
         12 XXLKED   EXEC PGM=HEWLH096,PARM='SIZE=(384K,96K),XREF,LIST,NCAL',       00033302
            XX         REGION=512K                                                  00066600
         13 //SYSPRINT DD SYSOUT=*
            X/SYSPRINT DD  SYSOUT=A                                                 00100000
         14 //SYSLIN   DD DISP=OLD,DSN=*.S1.SYSLIN
            X/SYSLIN   DD  DDNAME=SYSIN                                             00150000
         15 XXSYSLMOD  DD  DSN=&&GOSET(GO),SPACE=(1024,(50,20,1)),                 *00200000
            XX         UNIT=SYSDA,DISP=(MOD,PASS)                                   00250000
         16 XXSYSUT1   DD  UNIT=(SYSDA,SEP=(SYSLMOD,SYSLIN)),                     *00300000
            XX         SPACE=(1024,(200,20))                                        00400000
         17 XXGO       EXEC PGM=*.LKED.SYSLMOD,COND=(4,LT,LKED)                    00450000
         18 //GO.SYSUDUMP DD SYSOUT=*
            //* ****************************************************************
            //* ****************************************************************
            //* ****************************************************************
STMT NO. MESSAGE
```

Putting it all together

Job output for assembling, binding and running SALARY program (3)

```
       11 IEFC001I PROCEDURE LKEDG WAS EXPANDED USING SYSTEM LIBRARY SYS1.PROCLIB
       14 IEF648I INVALID DISP FIELD- PASS SUBSTITUTED
ICH70001I XXXXXXX  LAST ACCESS AT 21:00:58 ON TUESDAY, APRIL 29, 2014
IEF236I ALLOC. FOR XXXXXXX S1
IEF237I JES2 ALLOCATED TO SYSPRINT
IGD101I SMS ALLOCATED TO DDNAME (SYSLIN  )
        DSN (SYS14119.T212035.RA000.XXXXXXX.TEMP.H01     )
        STORCLAS (STANDARD) MGMTCLAS (        ) DATACLAS (        )
        VOL SER NOS= VIO
IGD103I SMS ALLOCATED TO DDNAME SYSIN
IEF237I ADA1 ALLOCATED TO SYSLIB
IEF237I ADA1 ALLOCATED TO
IEF237I ADA1 ALLOCATED TO
IEF142I XXXXXXX S1 - STEP WAS EXECUTED - COND CODE 0000
IEF285I   XXXXXXX.XXXXXXX.JOB47759.D0000101.?          SYSOUT
IGD106I SYS14119.T212035.RA000.XXXXXXX.TEMP.H01      PASSED,    DDNAME=SYSLIN
IGD104I XXXXXXX.STAGE1.ASM0.ANSWERS                  RETAINED,  DDNAME=SYSIN
IEF285I   SYS1.MACLIB                                  KEPT
IEF285I   VOL SER NOS= 33SY02.
IEF285I   PP.HLASM.ZOS201.SASMMAC1                     KEPT
IEF285I   VOL SER NOS= 33SY02.
IEF285I   PP.HLASM.ZOS201.SASMMAC2                     KEPT
IEF285I   VOL SER NOS= 33SY02.
IEF373I STEP/S1      /START 2014119.2120
IEF032I STEP/S1      /STOP  2014119.2120
        CPU:     0 HR  00 MIN  00.01 SEC    SRB:     0 HR  00 MIN  00.00 SEC
        VIRT:   220K  SYS:   264K  EXT:    65536K  SYS:    10404K
        ATB- REAL:                  36K  SLOTS:                    0K
             VIRT- ALLOC:       6M SHRD:       0M
IEF236I ALLOC. FOR XXXXXXX LKED S2
IEF237I JES2 ALLOCATED TO SYSPRINT
```

Putting it all together

Job output for assembling, binding and running SALARY program (4)

```
IGD103I SMS ALLOCATED TO DDNAME SYSLIN
IGD101I SMS ALLOCATED TO DDNAME (SYSLMOD )
        DSN (SYS14119.T212035.RA000.XXXXXXX.GOSET.H01    )
        STORCLAS (STANDARD) MGMTCLAS (        ) DATACLAS (        )
        VOL SER NOS= VIO
IGD101I SMS ALLOCATED TO DDNAME (SYSUT1  )
        DSN (SYS14119.T212035.RA000.XXXXXXX.R0111176     )
        STORCLAS (STANDARD) MGMTCLAS (        ) DATACLAS (        )
        VOL SER NOS= VIO
IEF142I XXXXXXX LKED S2 - STEP WAS EXECUTED - COND CODE 0000
IEF285I   XXXXXXX.XXXXXXX.JOB47759.D0000102.?          SYSOUT
IGD106I SYS14119.T212035.RA000.XXXXXXX.TEMP.H01      PASSED,    DDNAME=SYSLIN
IGD106I SYS14119.T212035.RA000.XXXXXXX.GOSET.H01     PASSED,    DDNAME=SYSLMOD
IGD105I SYS14119.T212035.RA000.XXXXXXX.R0111176      DELETED,   DDNAME=SYSUT1
IEF373I STEP/LKED    /START 2014119.2120
IEF032I STEP/LKED    /STOP  2014119.2120
        CPU:     0 HR  00 MIN  00.00 SEC    SRB:     0 HR  00 MIN  00.00 SEC
        VIRT:   100K  SYS:   268K  EXT:     1772K  SYS:    10376K
        ATB- REAL:                    0K  SLOTS:                    0K
             VIRT- ALLOC:       0M SHRD:       0M
IEF236I ALLOC. FOR XXXXXXX GO S2
IGD103I SMS ALLOCATED TO DDNAME PGM=*.DD
IEF237I JES2 ALLOCATED TO SYSUDUMP
Processing employee BOB SMITH
Employee has matched or exceeded target salary
IEF142I XXXXXXX GO S2 - STEP WAS EXECUTED - COND CODE 0000
IGD104I SYS14119.T212035.RA000.XXXXXXX.GOSET.H01     RETAINED,  DDNAME=PGM=*.DD
IEF285I   XXXXXXX.XXXXXXX.JOB47759.D0000103.?          SYSOUT
IEF373I STEP/GO      /START 2014119.2120
IEF032I STEP/GO      /STOP  2014119.2120
        CPU:     0 HR  00 MIN  00.00 SEC    SRB:     0 HR  00 MIN  00.00 SEC
```

Putting it all together

Job output for assembling, binding and running SALARY program (5)

```
         VIRT:     8K  SYS:   252K  EXT:        0K  SYS:    10440K
         ATB- REAL:                    0K  SLOTS:                    0K
              VIRT- ALLOC:       0M SHRD:       0M
 IGD105I SYS14119.T212035.RA000.XXXXXXX.TEMP.H01      DELETED,   DDNAME=SYSLIN
 IGD105I SYS14119.T212035.RA000.XXXXXXX.GOSET.H01     DELETED,   DDNAME=SYSLMOD
 IEF375I  JOB/XXXXXXX /START 2014119.2120
 IEF033I  JOB/XXXXXXX /STOP  2014119.2120
         CPU:     0 HR  00 MIN  00.01 SEC    SRB:     0 HR  00 MIN  00.00 SEC
1                                  High Level Assembler Option Summary                 (PTF UI11676)   Page    1
-                                                                                     HLASM R6.0  2014/04/29 21.20
0  No Overriding ASMAOPT Parameters
   No Overriding Parameters
   No Process Statements


   Options for this Assembly
0  NOADATA
     ALIGN
   NOASA
   NOBATCH
     CODEPAGE(047C)
   NOCOMPAT
   NODBCS
   NODECK
     DXREF
     ESD
   NOEXIT
     FLAG(0,ALIGN,NOCONT,EXLITW,NOIMPLEN,NOPAGE0,PUSH,RECORD,NOSUBSTR,USING0)
   NOFOLD
   NOGOFF
   NOINFO
```

Putting it all together

Job output for assembling, binding and running SALARY program (6)

```
      LANGUAGE(EN)
    NOLIBMAC
      LINECOUNT(60)
      LIST(121)
      MACHINE(,NOLIST)
      MXREF(SOURCE)
      OBJECT
      OPTABLE(UNI,NOLIST)
    NOPCONTROL
    NOPESTOP
    NOPROFILE
    NORA2
    NORENT
      RLD
      RXREF
      SECTALGN(8)
      SIZE(MAX)
    NOSUPRWARN
      SYSPARM()
    NOTERM
    NOTEST
      THREAD
    NOTRANSLATE
      TYPECHECK(MAGNITUDE,REGISTER)
      USING(NOLIMIT,MAP,NOWARN)
    NOWORKFILE
      XREF(SHORT,UNREFS)

   No Overriding DD Names
1                                    External Symbol Dictionary                                Page    2
-Symbol   Type   Id     Address  Length   Owner Id Flags Alias-of                    HLASM R6.0  2014/04/29 21.20
```

Putting it all together

Job output for assembling, binding and running SALARY program (7)

```
0          PC 00000001 00000000 00000000             00
 SALARY    SD 00000002 00000000 000002C4             02
1                                                                                                    Page    3
   Active Usings: None
0  Loc  Object Code    Addr1 Addr2  Stmt   Source Statement                                HLASM R6.0  2014/04/29 21.20
0                                     1 ************************************************************************
                                      2 * PUTTING IT ALL TOGETHER PROGRAM
                                      3 ************************************************************************
                                      4 *
                                      5 * The purpose of this small demo program is to demonstrate some small
                                      6 * parts of assembler programming.
                                      7 * The demo pretends that it has been passed an employee record via
                                      8 * register 1.
                                      9 * It will copy this record to some working storage and then proceed
                                     10 * to determine whether or not the employee is eligible for a pay
                                     11 * increase by comparing the employee's annual salary to the target
                                     12 * salary.
                                     13 *
                                     14 * The employee's annual salary is calculated as:
                                     15 *    12 x (MONTHLY_PAY-BENEFITS) + BONUS
                                     16 *
                                     17 *
                                     18          ASMDREG ,
                                     19+         PUSH  PRINT                                                01-ASMDR
                                    119+         POP   PRINT                                                01-ASMDR
 000000                 00000 002C4  120 SALARY   CSECT
                                     121 SALARY   AMODE 31
                                     122 SALARY   RMODE 24
                                     123 * USUAL PROGRAM SETUP
 000000 90EC D00C             0000C  124          STM   14,12,12(13)
 000004 05C0                         125          BALR  12,0
```

Putting it all together

Job output for assembling, binding and running SALARY program (8)

```
                 R:C  00006         126          USING *,12
                                    127 * Point register 1 at the first employee to process...
000006 4110 C286            0028C   128          la    r1,employee_id_1
                                    129 *
                                    130 * The data for the employee record is passed in from register 1.
                                    131 * This small program will determine whether or not the employee
                                    132 * is eligable for a pay rise or not.
                                    133 *
00000A D733 C252 C252 00258 00258   134          xc    employee(employee_rec_len),employee         Clear WS
000010 D233 C252 1000 00258 00000   135          mvc   employee(employee_rec_len),0(r1)   Copy record to WS
                                    136 *
                                    137 * Output the name of the employee that is being processed
                                    138 *
000016 D7FF C13A C13A 00140 00140   139          xc    wto_text,wto_text          Clear text buffer
00001C D213 C13A C23A 00140 00240   140          mvc   wto_text(l'process_text),process_text
000022 D227 C14E C252 00154 00258   141          mvc   wto_text+l'process_text(l'employee_name),employee_name
                                    142 * Calculate the amount of text to output
000028 A758 003C            0003C   143          lhi   r5,l'process_text+l'employee_name
00002C 4050 C138            0013E   144          sth   r5,wto_buf                 Store length in buffer
000030 4150 C138            0013E   145          la    r5,wto_buf                 Load address of bufffer
                                    146          WTO   TEXT=(5)                   Output text
000034                              148+         CNOP  0,4                                              01-WTO
000034 A715 003A            000A8   149+         BRAS  1,IHB0002A              BRANCH AROUND MESSAGE      @LCC 01-WTO
000038 0008                         150+         DC    AL2(8)                  TEXT LENGTH              @YA17152 01-WTO
00003A 0010                         151+         DC    B'0000000000010000'     MCSFLAGS                        01-WTO
00003C 00000000                     152+         DC    AL4(0)                  MESSAGE TEXT ADDRESS       @L5A 01-WTO
000040 02                           153+         DC    AL1(2)                  VERSION LEVEL              @PJC 01-WTO
000041 00                           154+         DC    B'00000000'             MISCELLANEOUS FLAGS        @L2A 01-WTO
000042 00                           155+         DC    AL1(0)                  REPLY LENGTH               @L2A 01-WTO
1                                                                                                    Page    4
   Active Usings: SALARY+X'6',R12
```

Putting it all together

Job output for assembling, binding and running SALARY program (9)

```
0  Loc  Object Code    Addr1 Addr2  Stmt   Source Statement                                HLASM R6.0  2014/04/29 21.20
0000043 68                           156+         DC    AL1(104)                 LENGTH OF WPX              @L5C 01-WTO
 000044 0080                         157+         DC    B'0000000010000000'      EXTENDED MCS FLAGS         @L2A 01-WTO
 000046 0000                         158+         DC    AL2(0)                   RESERVED                   @L2A 01-WTO
 000048 00000000                     159+         DC    AL4(0)                   REPLY BUFFER ADDRESS       @P7C 01-WTO
 00004C 00000000                     160+         DC    AL4(0)                   REPLY ECB ADDRESS          @P7C 01-WTO
 000050 00000000                     161+         DC    AL4(0)                   CONNECT ID                 @01C 01-WTO
 000054 0000                         162+         DC    B'0000000000000000'      DESCRIPTOR CODES           @L2A 01-WTO
 000056 0000                         163+         DC    AL2(0)                   RESERVED                   @L2A 01-WTO
 000058 0000000000000000             164+         DC    XL16'00000000000000000000000000000000'                 X01-WTO
 000060 0000000000000000                +                                        EXTENDED ROUTING CODES     @L2A
 000068 0000                         165+         DC    B'0000000000000000'      MESSAGE TYPE               @L2A 01-WTO
 00006A 0000                         166+         DC    AL2(0)                   MESSAGE'S PRIORITY         @L2A 01-WTO
 00006C 4040404040404040             167+         DC    CL8'        '            JOB ID                     @L2A 01-WTO
 000074 4040404040404040             168+         DC    CL8'        '            JOB NAME                   @L2A 01-WTO
 00007C 4040404040404040             169+         DC    CL8'        '            RETRIEVAL KEY              @L2A 01-WTO
 000084 00000000                     170+         DC    AL4(0)                   TOKEN FOR DOM              @P1C 01-WTO
 000088 00000000                     171+         DC    AL4(0)                   CONSOLE ID                 @P1C 01-WTO
 00008C 4040404040404040             172+         DC    CL8'        '            SYSTEM NAME                @L2A 01-WTO
 000094 4040404040404040             173+         DC    CL8'        '            CONSOLE NAME               @L3A 01-WTO
 00009C 00000000                     174+         DC    AL4(0)                   REPLY CONSOLE NAME/ID ADDR @L3A 01-WTO
 0000A0 00000000                     175+         DC    AL4(0)                   CART ADDRESS               @L4C 01-WTO
 0000A4 00000000                     176+         DC    AL4(0)                   WSPARM ADDRESS             @L6C 01-WTO
 0000A8                              177+IHB0002A DS    0H                                                  01-WTO
 0000A8 18E1                         178+         LR    14,1                     FIRST BYTE OF PARM LIST    @L2A 01-WTO
 0000AA 1BFF                         179+         SR    15,15                    CLEAR REGISTER 15          @L2A 01-WTO
 0000AC 4AF1 0000           00000    180+         AH    15,0(1,0)                ADD LENGTH OF TEXT + 4     @L2A 01-WTO
 0000B0 1AEF                         181+         AR    14,15                    FIRST BYTE AFTER TEXT      @L2A 01-WTO
 0000B2 5050 1004           00004    182+         ST    5,4(0,1)                 STORE TEXT ADDR INTO PLIST @L5A 01-WTO
 0000B6 0A23                         183+         SVC   35                       ISSUE SVC 35               @L6A 01-WTO
                                     184 *
```

Putting it all together

Job output for assembling, binding and running SALARY program (10)

```
                                      185 * Calculate employee's yearly pay as 12*(MONTHLY_PAY-BENEFITS)+BONUS
                                      186 * We will use register 3 as a work register
                                      187 *
 0000B8 5830 C27A            00280    188          l     r3,employee_monthly_pay
 0000BC 5B30 C282            00288    189          s     r3,employee_benefits       MONTHLY-BENEFITS
 0000C0 5C20 C2BA            002C0    190          m     r2,=f'12'                  Multiply by 12
 0000C4 4A30 C27E            00284    191          ah    r3,employee_bonus          Add yearly bonus
 0000C8 5930 C24E            00254    192          c     r3,target_salary           Compare total with target
 0000CC 4740 C106            0010C    193          bl    deserves_increase
                                      194          WTO   'Employee has matched or exceeded target salary'
 0000D0                               196+         CNOP  0,4                                                  01-WTO
 0000D0 A715 001B            00106    197+         BRAS  1,IHB0004A               BRANCH AROUND MESSAGE      @LCC 01-WTO
 0000D4 0032                          198+         DC    AL2(50)                  TEXT LENGTH              @YA17152 01-WTO
 0000D6 0000                          199+         DC    B'0000000000000000'      MCSFLAGS                          01-WTO
 0000D8 C594979396A88585              200+         DC    C'Employee has matched or exceeded target salary'       X01-WTO
 0000E0 408881A2409481A3                +                                      MESSAGE TEXT               @L6C
 000106                               201+IHB0004A DS    0H                                                         01-WTO
 000106 0A23                          202+         SVC   35                       ISSUE SVC 35               @L6A 01-WTO
 000108 47F0 C130            00136    203          b     resume_code
 00010C                               204 deserves_increase    dc     0h
                                      205          WTO   'Employee deserves a pay increase'
 00010C                               207+         CNOP  0,4                                                  01-WTO
 00010C A715 0014            00134    208+         BRAS  1,IHB0006A               BRANCH AROUND MESSAGE      @LCC 01-WTO
 000110 0024                          209+         DC    AL2(36)                  TEXT LENGTH              @YA17152 01-WTO
 000112 0000                          210+         DC    B'0000000000000000'      MCSFLAGS                          01-WTO
1                                                                                                          Page    5
   Active Usings: SALARY+X'6',R12
0  Loc  Object Code    Addr1 Addr2  Stmt   Source Statement                                HLASM R6.0  2014/04/29 21.20
0000114 C594979396A88585             211+         DC    C'Employee deserves a pay increase'                     X01-WTO
 00011C 408485A28599A585                +                                      MESSAGE TEXT               @L6C
 000134                               212+IHB0006A DS    0H                                                         01-WTO
```

Putting it all together

Job output for assembling, binding and running SALARY program (11)

```
000134 0A23                            213+         SVC   35                      ISSUE SVC 35              @L6A 01-WTO
                                       214 *
                                       215 * Return to the caller of the program
                                       216 *
000136                                 217 resume_code          dc     0h
000136 98EC D00C            0000C      218          LM    14,12,12(13)
                                       219 *
00013A 17FF                            220          XR    15,15
00013C 07FE                            221          BR    14
                                       222 * ------------------------------------------------------------------
                                       223 * END OF PROGRAM - DATA FOLLOWS
                                       224 * ------------------------------------------------------------------
                                       225 *
00013E 0000                            226 WTO_BUF              DC H'0'
000140                                 227 WTO_TEXT             DS CL256
000240 D799968385A2A289                228 PROCESS_TEXT         DC C'Processing employee '
                                       229 * SALARY SCHEME DATA
000254 00005DC0                        230 TARGET_SALARY        DC F'24000'             TARGET SALARY FOR COMPANY
                                       231 * EMPLOYEE RECORD STRUCTURE
000258                                 232 EMPLOYEE             DC 0F
000258                                 233 EMPLOYEE_NAME        DS CL40                    EMPLOYEE'S NAME
000280                                 234 EMPLOYEE_MONTHLY_PAY DS F                       VALUE OF MONTHLY PAY
000284                                 235 EMPLOYEE_BONUS       DS H                       YEARLY BONUS AMOUNT
000288                                 236 EMPLOYEE_BENEFITS    DS F                       MONTHLY BENEFITS
                     00034             237 EMPLOYEE_REC_LEN     EQU *-EMPLOYEE             SIZE OF EMPLOYEE RECORD
                                       238 * EMPLOYEE EXAMPLE DATA
00028C                                 239 EMPLOYEE_ID_1        DC    0F
00028C C2D6C240E2D4C9E3                240                      DC CL40'BOB SMITH'
0002B4 000007D0                        241                      DC F'2000'
0002B8 03E8                            242                      DC H'1000'
0002BA 0000
```

Putting it all together

Job output for assembling, binding and running SALARY program (12)

```
 0002BC 00000032                     243                      DC F'50'
 0002C0                              244          LTORG ,
 0002C0 0000000C                     245                =f'12'
                                     246          END
1                               Ordinary Symbol and Literal Cross Reference                                Page    6
-Symbol   Length   Value     Id    R Type Asm  Program   Defn References                  HLASM R6.0  2014/04/29 21.20
0deserves_increase
              2 0000010C 00000002     H  H               204  193B
 EMPLOYEE     4 00000258 00000002     F  F               232  134M  134   135M  237
 EMPLOYEE_BENEFITS
              4 00000288 00000002     F  F               236  189
 EMPLOYEE_BONUS
              2 00000284 00000002     H  H               235  191
 EMPLOYEE_ID_1
              4 0000028C 00000002     F  F               239  128
 EMPLOYEE_MONTHLY_PAY
              4 00000280 00000002     F  F               234  188
 EMPLOYEE_NAME
             40 00000258 00000002     C  C               233  141   141   143
 EMPLOYEE_REC_LEN
              1 00000034 00000002 A   U                  237  134   135
 IHB0002A     2 000000A8 00000002     H  H               177  149B
 IHB0004A     2 00000106 00000002     H  H               201  197B
 IHB0006A     2 00000134 00000002     H  H               212  208B
 PROCESS_TEXT
            20 00000240 00000002     C  C               228  140   140   141M  143
 resume_code
              2 00000136 00000002     H  H               217  203B
 R1           1 00000001 00000001 A   U                   28  128M  135
 R2           1 00000002 00000001 A   U                   29  190M
 R3           1 00000003 00000001 A   U                   30  188M  189M  191M  192
```

Putting it all together

Job output for assembling, binding and running SALARY program (13)

```
 0002BC 00000032                     243                      DC F'50'
 0002C0                              244          LTORG ,
 0002C0 0000000C                     245                =f'12'
                                     246          END
1                               Ordinary Symbol and Literal Cross Reference                                Page    6
-Symbol   Length   Value     Id    R Type Asm  Program   Defn References                  HLASM R6.0  2014/04/29 21.20
0deserves_increase
              2 0000010C 00000002     H  H               204  193B
 EMPLOYEE     4 00000258 00000002     F  F               232  134M  134   135M  237
 EMPLOYEE_BENEFITS
              4 00000288 00000002     F  F               236  189
 EMPLOYEE_BONUS
              2 00000284 00000002     H  H               235  191
 EMPLOYEE_ID_1
              4 0000028C 00000002     F  F               239  128
 EMPLOYEE_MONTHLY_PAY
              4 00000280 00000002     F  F               234  188
 EMPLOYEE_NAME
             40 00000258 00000002     C  C               233  141   141   143
 EMPLOYEE_REC_LEN
              1 00000034 00000002 A   U                  237  134   135
 IHB0002A     2 000000A8 00000002     H  H               177  149B
 IHB0004A     2 00000106 00000002     H  H               201  197B
 IHB0006A     2 00000134 00000002     H  H               212  208B
 PROCESS_TEXT
            20 00000240 00000002     C  C               228  140   140   141M  143
 resume_code
              2 00000136 00000002     H  H               217  203B
 R1           1 00000001 00000001 A   U                   28  128M  135
 R2           1 00000002 00000001 A   U                   29  190M
 R3           1 00000003 00000001 A   U                   30  188M  189M  191M  192
```

Putting it all together

Job output for assembling, binding and running SALARY program (14)

```
 R5             1 00000005 00000001 A   U                   32  143M  144   145M
 SALARY         1 00000000 00000002     J                  120  121   122
 TARGET_SALARY
                4 00000254 00000002     F  F               230  192
 WTO_BUF        2 0000013E 00000002     H  H               226  144M  145
 WTO_TEXT     256 00000140 00000002     C  C               227  139M  139   140M  141M
 =f'12'         4 000002C0 00000002     F                  245  190
1                                Unreferenced Symbols Defined in CSECTs                                  Page    7
-  Defn  Symbol                                                                        HLASM R6.0  2014/04/29 21.20
0   103  AR0
    104  AR1
    113  AR10
    114  AR11
    115  AR12
    116  AR13
    117  AR14
    118  AR15
    105  AR2
    106  AR3
    107  AR4
    108  AR5
    109  AR6
    110  AR7
    111  AR8
    112  AR9
     84  CR0
     85  CR1
     94  CR10
     95  CR11
     96  CR12
     97  CR13
```

Putting it all together

Job output for assembling, binding and running SALARY program (15)

```
98  CR14
99  CR15
86  CR2
87  CR3
88  CR4
89  CR5
90  CR6
91  CR7
92  CR8
93  CR9
46  FR0
47  FR1
56  FR10
57  FR11
58  FR12
59  FR13
60  FR14
61  FR15
48  FR2
49  FR3
50  FR4
51  FR5
52  FR6
53  FR7
54  FR8
55  FR9
27  R0
37  R10
38  R11
39  R12
40  R13
```

Putting it all together

Job output for assembling, binding and running SALARY program (16)

```
     41  R14
     42  R15
1                              Unreferenced Symbols Defined in CSECTs                              Page    8
-  Defn  Symbol                                                                  HLASM R6.0  2014/04/29 21.20
0    31  R4
     33  R6
     34  R7
     35  R8
     36  R9
     65  VR0
     66  VR1
     75  VR10
     76  VR11
     77  VR12
     78  VR13
     79  VR14
     80  VR15
     67  VR2
     68  VR3
     69  VR4
     70  VR5
     71  VR6
     72  VR7
     73  VR8
     74  VR9
1                                 Macro and Copy Code Source Summary                               Page    9
- Con Source                                 Volume    Members                 HLASM R6.0  2014/04/29 21.20
0  L1 SYS1.MACLIB                            33SY02    SYSSTATE WTO
   L3 PP.HLASM.ZOS201.SASMMAC2               33SY02    ASMDREG
1                                             Using Map                                            Page   10
-                                                                               HLASM R6.0  2014/04/29 21.20
```

Putting it all together

Job output for assembling, binding and running SALARY program (17)

```
     41  R14
     42  R15
1                              Unreferenced Symbols Defined in CSECTs                              Page    8
-  Defn  Symbol                                                                  HLASM R6.0  2014/04/29 21.20
0    31  R4
     33  R6
     34  R7
     35  R8
     36  R9
     65  VR0
     66  VR1
     75  VR10
     76  VR11
     77  VR12
     78  VR13
     79  VR14
     80  VR15
     67  VR2
     68  VR3
     69  VR4
     70  VR5
     71  VR6
     72  VR7
     73  VR8
     74  VR9
1                          Macro and Copy Code Source Summary                                      Page    9
- Con Source                                Volume    Members                 HLASM R6.0  2014/04/29 21.20
0  L1 SYS1.MACLIB                          33SY02    SYSSTATE WTO
   L3 PP.HLASM.ZOS201.SASMMAC2             33SY02    ASMDREG
1                                             Using Map                                            Page   10
-                                                                             HLASM R6.0  2014/04/29 21.20
```

Putting it all together

Job output for assembling, binding and running SALARY program (18)

```
   Stmt  -----Location----- Action ----------------Using----------------- Reg Max     Last Label and Using Text
           Count      Id           Type          Value    Range     Id        Disp    Stmt
0   126  00000006  00000002 USING  ORDINARY    00000006 00001000 00000002  12 002BA    203 *,12
1                                  General Purpose Register Cross Reference                                Page   11
- Register  References (M=modified, B=branch, U=USING, D=DROP, N=index)                      HLASM R6.0  2014/04/29 21.20
0    0(0)      124   218M
     1(1)      124   128M  135   149M  178   180N  182   197M  208M  218M
     2(2)      124   190M  218M
     3(3)      124   188M  189M  190M  191M  192   218M
     4(4)      124   218M
     5(5)      124   143M  144   145M  182   218M
     6(6)      124   218M
     7(7)      124   218M
     8(8)      124   218M
     9(9)      124   218M
    10(A)      124   218M
    11(B)      124   218M
    12(C)      124   125M  126U  218M
    13(D)      124   218
    14(E)      124   178M  181M  218M  221B
    15(F)      124   179M  179   180M  181   218M  220M  220
1                                  Diagnostic Cross Reference and Assembler Summary                        Page   12
-                                                                                          HLASM R6.0  2014/04/29 21.20
0     No Statements Flagged in this Assembly
 HIGH LEVEL ASSEMBLER, 5696-234, RELEASE 6.0, PTF UI11676
0SYSTEM: z/OS 02.01.00             JOBNAME: XXXXXXX     STEPNAME: S1         PROCSTEP: (NOPROC)
0Data Sets Allocated for this Assembly
  Con DDname   Data Set Name                              Volume  Member
   P1 SYSIN    XXXXXXX.STAGE1.ASM0.ANSWERS                33P002  SALARY
   L1 SYSLIB   SYS1.MACLIB                                33SY02
   L2          PP.HLASM.ZOS201.SASMMAC1                   33SY02
```

Putting it all together

Job output for assembling, binding and running SALARY program (19)

```
   L3          PP.HLASM.ZOS201.SASMMAC2                    33SY02
      SYSLIN   SYS14119.T212035.RA000.XXXXXXX.TEMP.H01
      SYSPRINT XXXXXXX.XXXXXXX.JOB47759.D0000101.?

   64840K allocated to Buffer Pool       Storage required     384K
      91 Primary Input Records Read         3707 Library Records Read                0 Work File Reads
       0 ASMAOPT Records Read                384 Primary Print Records Written       0 Work File Writes
      10 Object Records Written                0 ADATA Records Written
0Assembly Start Time: 21.20.35 Stop Time: 21.20.35 Processor Time: 00.00.00.0068
 Return Code 000

1z/OS V2 R1 BINDER     21:20:35 TUESDAY APRIL 29, 2014
 BATCH EMULATOR  JOB(XXXXXXX ) STEP(S2      ) PGM= HEWLH096  PROCEDURE(LKED    )
 IEW2278I B352 INVOCATION PARAMETERS - XREF,LIST,NCAL,MAP
 IEW2650I 5102 MODULE ENTRY NOT PROVIDED.  ENTRY DEFAULTS TO SECTION SALARY.

1                        *** M O D U L E  M A P ***

 ---------------
 CLASS  B_TEXT            LENGTH =      2C4  ATTRIBUTES = CAT,   LOAD, RMODE= 24
                          OFFSET =        0 IN SEGMENT 001     ALIGN = DBLWORD
 ---------------

  SECTION    CLASS                                      ------- SOURCE --------
   OFFSET   OFFSET  NAME               TYPE    LENGTH  DDNAME   SEQ  MEMBER

                0  SALARY             CSECT       2C4  SYSLIN    01  **NULL**
1          ***  DATA SET SUMMARY  ***

 DDNAME    CONCAT   FILE IDENTIFICATION
```

Putting it all together

Job output for assembling, binding and running SALARY program (20)

```
 SYSLIN      01     SYS14119.T212035.RA000.XXXXXXX.TEMP.H01

          *** E N D   O F   M O D U L E   M A P ***

1                                   C R O S S - R E F E R E N C E   T A B L E
                                    _____________________________________

 TEXT CLASS = B_TEXT

 ---------------  R E F E R E N C E  -------------------------  T A R G E T  ----------------------------------------
   CLASS                            ELEMENT       |                                        ELEMENT                 |
   OFFSET  SECT/PART(ABBREV)         OFFSET  TYPE | SYMBOL(ABBREV)   SECTION (ABBREV)          OFFSET CLASS NAME       |
                                                  |                                                                  |
 ***  NO ADDRESS CONSTANTS FOR THIS CLASS   ***
                                    *** E N D   O F   C R O S S   R E F E R E N C E ***


                               *** O P E R A T I O N   S U M M A R Y   R E P O R T ***

1PROCESSING OPTIONS:

    ALIASES             NO
    ALIGN2              NO
    AMODE               UNSPECIFIED
    CALL                NO
    CASE                UPPER
    COMPAT              UNSPECIFIED
    COMPRESS            AUTO
    DCBS                NO
```

Putting it all together

Job output for assembling, binding and running SALARY program (21)

```
DYNAM                NO
EXTATTR              UNSPECIFIED
EXITS:               NONE
FILL                 NONE
GID                  UNSPECIFIED
HOBSET               NO
INFO                 NO
LET                  04
LINECT               060
LIST                 SUMMARY
LISTPRIV             NO
LONGPARM             NO
MAP                  YES
MAXBLK               032760
MODMAP               NO
MSGLEVEL             00
OVLY                 NO
PRINT                YES
RES                  NO
REUSABILITY          UNSPECIFIED
RMODE                UNSPECIFIED
SIGN                 NO
STORENX              NOREPLACE
STRIPCL              NO
STRIPSEC             NO
SYMTRACE
TERM                 NO
TRAP                 ON
UID                  UNSPECIFIED
UPCASE               NO
WKSPACE              000000K,000000K
```

Putting it all together

Job output for assembling, binding and running SALARY program (22)

```
    XCAL                NO
    XREF                YES
    ***END OF OPTIONS***



1SAVE OPERATION SUMMARY:

    MEMBER NAME         GO
    LOAD LIBRARY        SYS14119.T212035.RA000.XXXXXXX.GOSET.H01
    PROGRAM TYPE        LOAD MODULE
    VOLUME SERIAL
    MAX BLOCK           32760
    DISPOSITION         ADDED NEW
    TIME OF SAVE        21.20.35  APR 29, 2014


1SAVE MODULE ATTRIBUTES:

    AC                  000
    AMODE                31
    COMPRESSION         NONE
    DC                  NO
    EDITABLE            YES
    EXCEEDS 16MB        NO
    EXECUTABLE          YES
    LONGPARM            NO
    MIGRATABLE          YES
    OL                  NO
    OVLY                NO
```

Putting it all together

Job output for assembling, binding and running SALARY program (23)

```
    PACK,PRIME          NO,NO
    PAGE ALIGN          NO
    REFR                NO
    RENT                NO
    REUS                NO
    RMODE                24
    SCTR                NO
    SIGN                NO
    SSI
    SYM GENERATED       NO
    TEST                NO
    XPLINK              NO
    MODULE SIZE (HEX)   000002C8


1 ENTRY POINT AND ALIAS SUMMARY:

  NAME:            ENTRY TYPE AMODE C_OFFSET CLASS NAME        STATUS

  SALARY            MAIN_EP      31 00000000 B_TEXT

                        *** E N D   O F   O P E R A T I O N   S U M M A R Y   R E P O R T ***



1z/OS V2 R1 BINDER     21:20:35 TUESDAY APRIL 29, 2014
 BATCH EMULATOR  JOB(XXXXXXX ) STEP(S2      ) PGM= HEWLH096  PROCEDURE(LKED    )
 IEW2008I 0F03 PROCESSING COMPLETED.  RETURN CODE =  0.
```

Putting it all together

Job output for assembling, binding and running SALARY program (24)

```
1----------------------
 MESSAGE SUMMARY REPORT
 ----------------------
  TERMINAL MESSAGES      (SEVERITY = 16)
  NONE

  SEVERE MESSAGES        (SEVERITY = 12)
  NONE

  ERROR MESSAGES         (SEVERITY = 08)
  NONE

  WARNING MESSAGES       (SEVERITY = 04)
  NONE

  INFORMATIONAL MESSAGES (SEVERITY = 00)
  2008  2278  2650


  **** END OF MESSAGE SUMMARY REPORT ****
```