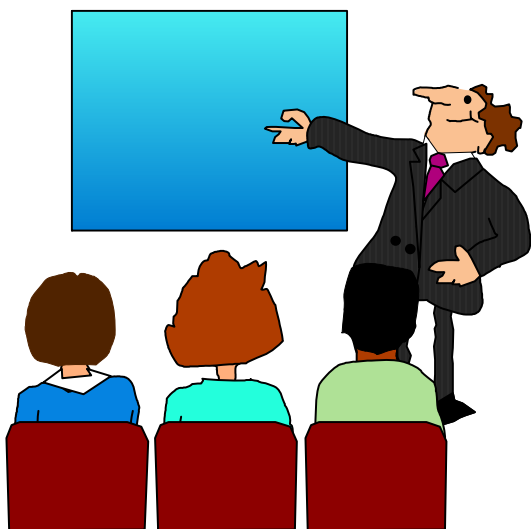


# Assembler Language "Boot Camp" Part 3 - Assembly and Execution; Branching

SHARE 118 in Atlanta  
Session 10346  
March 13, 2012



# Introduction

---

## ■ Who are we?

- John Ehrman, IBM Software Group
- Dan Greiner, IBM Systems & Technology Group

# Introduction

---

- Who are you?
  - An applications programmer who needs to write something in mainframe assembler?
  - An applications programmer who wants to understand z/Architecture so as to better understand how HLL programs work?
  - A manager who needs to have a general understanding of assembler?
  
- Our goal is to provide for professionals an introduction to the z/Architecture assembler language

# Introduction

---

- These sessions are based on notes from a course in assembler language at Northern Illinois University
- The notes are in turn based on the textbook, Assembler Language with ASSIST and ASSIST/I by Ross A Overbeek and W E Singletary, Fourth Edition, published by Macmillan

# Introduction

---

- The original ASSIST (Assembler System for Student Instruction and Systems Teaching) was written by John Mashey at Penn State University
- ASSIST/I, the PC version of ASSIST, was written by Bob Baker, Terry Disz and John McCharen at Northern Illinois University

# Introduction

---

- Both ASSIST and ASSIST/I are in the public domain, and are compatible with the System/370 architecture of about 1975 (fine for beginners)
- Everything we discuss here works the same in z/Architecture
- Both ASSIST and ASSIST/I are available at <http://www.kcats.org/assist>

# Introduction

---

- ASSIST-V is also available now, at <http://www.kcats.org/assist-v>
- Other materials described in these sessions can be found at the same site, at <http://www.kcats.org/share>
- Please keep in mind that ASSIST, ASSIST/I, and ASSIST-V are not supported by Penn State, NIU, NESI, or any of us

# Introduction

---

- Other references used in the course at NIU:
  - Principles of Operation (PoO)
  - System/370 Reference Summary
  - High Level Assembler Language Reference
- Access to PoO and HLASM Ref is normally online at the IBM publications web site
- Students use the S/370 "green card" booklet all the time, including during examinations (SA22-7209)



# Our Agenda for the Week

---

- Assembler Boot Camp (ABC) Part 1: Numbers and Basic Arithmetic (Monday - 9:30 a.m.)
- ABC Part 2: Instructions and Addressing (Monday - 1:30 p.m.)
- ABC Part 3: Assembly and Execution; Branching (Tuesday - 9:30 a.m.)
- ABC Lab 1: Hands-On Assembler Lab Using ASSIST/I (Tuesday - 6:00 p.m.)

# Our Agenda for the Week

---

- ABC Part 4: Program Structures; Arithmetic (Wednesday - 9:30 a.m.)
- ABC Lab 2: Hands-On Assembler Lab Using ASSIST/I (Wednesday - 6:00 p.m.)
- ABC Part 5: Decimal and Logical Instructions (Thursday - 9:30 a.m.)

# Agenda for this Session

---

- Assembly of a Complete Program
- Execution of a Complete Program
- Implicit Addresses and USING
- The Condition Code and Branching
- X-Instructions and ASSIST

# Assembly of a Complete Program

In Which We Take Baby Steps and are Amazed at How a Program Works!



# A Complete Program

---

- Yesterday, we introduced a few instructions and used them to create a complete, if short, program
- Today, we will analyze the object code generated by the assembly of the program, then look at what happens when ASSIST/I executes the program
- "Object code - nothing else matters"

# First Demo Program, Source List

---

- \* This program adds two numbers that are taken
- \* from the 5th and 6th words of the program.
- \* The sum is stored in the 7th word.

```
ADD2      CSECT
          L      1,16(,15)      Load 1st no. into R1
          L      2,20(,15)      Load 2nd no. into R2
          AR     1,2             Get sum in R1
          ST     1,24(,15)      Store sum
          BCR   B'1111',14      Return to caller
          DC    F'4'            Fullword initially 4
          DC    F'6'            Fullword initially 6
          DS    F               Rsrvd only, no init
          END    ADD2
```

# First Demo Program, Assembled

---

LOC	OBJECT	CODE	SOURCE	STATEMENT
000000			ADD2	CSECT
000000	5810	F010		L 1,16(,15)
000004	5820	F014		L 2,20(,15)
000008	1A12			AR 1,2
00000A	5010	F018		ST 1,24(,15)
00000E	07FE			BCR B'1111',14
000010	00000004			DC F'4'
000014	00000006			DC F'6'
000018				DS F
				END ADD2

# First Demo Program, Assembled

---

LOC	OBJECT CODE	SOURCE STATEMENT
000000		ADD2 CSECT
000000	5810 F010	L 1,16(,15)
000004	5820 F014	L 2,20(,15)
000008	1A12	AR 1,2
00000A	5010 F018	ST 1,24(,15)
00000E	07FE	BCR B'1111',14
000010	00000004	DC F'4'
000014	00000006	DC F'6'
000018		DS F
		END ADD2



# First Demo Program, Assembled

---

LOC	OBJECT CODE	SOURCE STATEMENT
000000		ADD2 CSECT
000000	5810 F010	L 1,16(,15)
000004	5820 F014	L 2,20(,15)
000008	1A12	AR 1,2
00000A	5010 F018	ST 1,24(,15)
00000E	07FE	BCR B'1111',14
000010	00000004	DC F'4'
000014	00000006	DC F'6'
000018		DS F
		END ADD2

# First Demo Program, Assembled

---

LOC	OBJECT	CODE	SOURCE	STATEMENT
000000			ADD2	CSECT
000000	5810	F010		L 1,16(,15)
000004	5820	F014		L 2,20(,15)
000008	1A12			AR 1,2
00000A	5010	F018		ST 1,24(,15)
00000E	07FE			BCR B'1111',14
000010	00000004			DC F'4'
000014	00000006			DC F'6'
000018				DS F
				END ADD2

# First Demo Program, Assembled

---

LOC	OBJECT CODE	SOURCE STATEMENT
000000		ADD2 CSECT
000000	5810 F010	L 1,16(,15)
000004	5820 F014	L 2,20(,15)
000008	1A12	AR 1,2
00000A	5010 F018	ST 1,24(,15)
00000E	07FE	BCR B'1111',14
000010	00000004	DC F'4'
000014	00000006	DC F'6'
000018		DS F
		END ADD2

# First Demo Program, Assembled

---

LOC	OBJECT	CODE	SOURCE	STATEMENT
000000			ADD2	CSECT
000000	5810	F010		L 1,16(,15)
000004	5820	F014		L 2,20(,15)
000008	1A12			AR 1,2
00000A	5010	F018		ST 1,24(,15)
00000E	07FE			BCR B'1111',14
000010	00000004			DC F'4'
000014	00000006			DC F'6'
000018				DS F
				END ADD2

# First Demo Program, Assembled

---

LOC	OBJECT CODE	SOURCE STATEMENT
000000		ADD2 CSECT
000000	5810 F010	L 1,16(,15)
000004	5820 F014	L 2,20(,15)
000008	1A12	AR 1,2
00000A	5010 F018	ST 1,24(,15)
00000E	07FE	BCR B'1111',14
000010	00000004	DC F'4'
000014	00000006	DC F'6'
000018		DS F
		END ADD2

# First Demo Program, Assembled

---

LOC	OBJECT CODE	SOURCE STATEMENT
000000		ADD2 CSECT
000000	5810 F010	L 1,16(,15)
000004	5820 F014	L 2,20(,15)
000008	1A12	AR 1,2
00000A	5010 F018	ST 1,24(,15)
00000E	07FE	BCR B'1111',14
000010	00000004	DC F'4'
000014	00000006	DC F'6'
000018		DS F
		END ADD2

# First Demo Program, Assembled

---

LOC	OBJECT CODE	SOURCE STATEMENT
000000		ADD2 CSECT
000000	5810 F010	L 1,16(,15)
000004	5820 F014	L 2,20(,15)
000008	1A12	AR 1,2
00000A	5010 F018	ST 1,24(,15)
00000E	07FE	BCR B'1111',14
000010	00000004	DC F'4'
000014	00000006	DC F'6'
000018	<input type="text"/>	<input type="text"/> DS <input type="text"/> F
		END ADD2

# First Demo Program, Assembled

---

LOC	OBJECT	CODE	SOURCE	STATEMENT
000000			ADD2	CSECT
000000	5810	F010		L 1,16(,15)
000004	5820	F014		L 2,20(,15)
000008	1A12			AR 1,2
00000A	5010	F018		ST 1,24(,15)
00000E	07FE			BCR B'1111',14
000010	00000004			DC F'4'
000014	00000006			DC F'6'
000018				DS F
				END


```
graph TD; ADD2[ADD2] --- CSECT[CSECT]; CSECT --- L1[L 1,16(,15)]; L1 --- L2[L 2,20(,15)]; L2 --- AR[AR 1,2]; AR --- ST[ST 1,24(,15)]; ST --- BCR[BCR B'1111',14]; BCR --- DC1[DC F'4']; DC1 --- DC2[DC F'6']; DC2 --- DS[DS F]; DS --- END[END]; END --- ADD2[ADD2];
```



# First Demo Program, Assembled

---

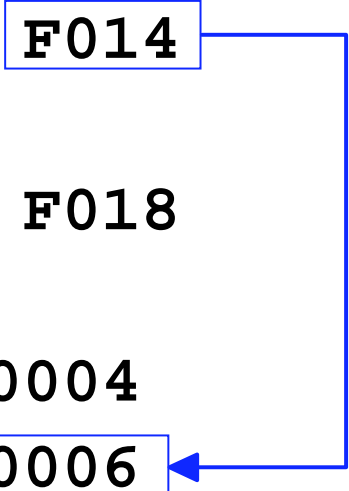
LOC	OBJECT	CODE	SOURCE	STATEMENT
000000			ADD2	CSECT
000000	5810	F010		L 1,16(,15)
000004	5820	F014		L 2,20(,15)
000008	1A12			AR 1,2
00000A	5010	F018		ST 1,24(,15)
00000E	07FE			BCR B'1111',14
000010		00000004		DC F'4'
000014		00000006		DC F'6'
000018				DS F
				END ADD2



# First Demo Program, Assembled

---

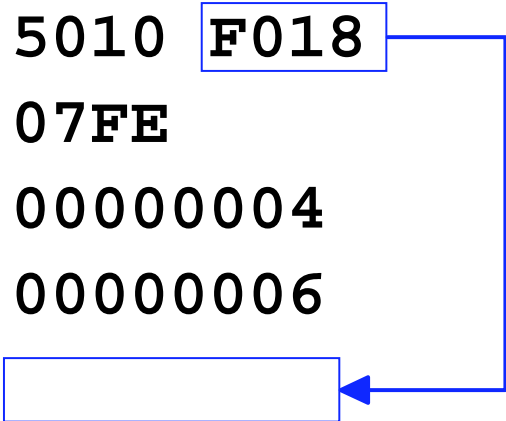
LOC	OBJECT	CODE	SOURCE	STATEMENT
000000			ADD2	CSECT
000000	5810	F010		L 1,16(,15)
000004	5820	F014		L 2,20(,15)
000008	1A12			AR 1,2
00000A	5010	F018		ST 1,24(,15)
00000E	07FE			BCR B'1111',14
000010	00000004			DC F'4'
000014	00000006			DC F'6'
000018				DS F
				END ADD2



# First Demo Program, Assembled

---

LOC	OBJECT	CODE	SOURCE	STATEMENT
000000			ADD2	CSECT
000000	5810	F010		L 1,16(,15)
000004	5820	F014		L 2,20(,15)
000008	1A12			AR 1,2
00000A	5010	F018		ST 1,24(,15)
00000E	07FE			BCR B'1111',14
000010	00000004			DC F'4'
000014	00000006			DC F'6'
000018				DS F
				END ADD2



# Execution of a Complete Program

In Which We See  
the World Go By  
in Single Steps



# ADD2 Program Before Execution

---

PSW AT BREAK    FFC50000 0F000000

R0-7 :    F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4  
R8-15:    F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4 00000020 00000068 00000000

000000	5810F010	5820F014	1A125010	F01807FE	*..0.....&.....*
000010	00000004	00000006	F5F5F5F5	F5F5F5F5	*.....55555555*
000020	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000030	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000040	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000050	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000060	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000070	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000080	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000090	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
0000A0	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
0000B0	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
0000C0	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
0000D0	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
0000E0	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
0000F0	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*

==> B(rkpt.), D(ump), G(o), M(emory), P(SW), Q(uit), R(eg.), S(tep), T(race)  
:

# ADD Program Before Execution

Here is our program loaded into memory

```
FFC50000 0F000000
```

```
F4 F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4
F4 F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4 00000020 00000068 00000000
```

000000	5810F010	5820F014	1A125010	F01807FE	*..0.....&.....*
000010	00000004	00000006	F5F5F5F5	F5F5F5F5	*.....55555555*
000020	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000030	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000040	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000050	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000060	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000070	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000080	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000090	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
0000A0	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
0000B0	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
0000C0	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
0000D0	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
0000E0	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
0000F0	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*

```
==> B(rkpt.), D(ump), G(o), M(emory), P(SW), Q(uit), R(eg.), S(tep), T(race)
```

```
:
```

# ADD2 Program Before Execution

Address of the first instruction

PSW AT BREAK FFC50000 0F000000

R0-7 : F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4  
 R8-15: F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4 00000020 00000068 00000000

000000	5810F010	5820F014	1A125010	F01807FE	*..0.....&.....*
000010	00000004	00000006	F5F5F5F5	F5F5F5F5	*.....55555555*
000020	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000030	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000040	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000050	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000060	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000070	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000080	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000090	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
0000A0	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
0000B0	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
0000C0	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
0000D0	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
0000E0	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
0000F0	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*

==> B(rkpt.), D(ump), G(o), M(emory), P(SW), Q(uit), R(eg.), S(tep), T(race)  
 :

# ADDRESS of the next instruction

Address of the next instruction

```

PSW AT BREAK   FFC50000 0F000000

R0-7  :  F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4
R8-15:  F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4 00000020 00000068 00000000

000000    5810F010 ← 5820F014 1A125010   F01807FE           *..0.....&.....*
000010    00000004 00000006  F5F5F5F5   F5F5F5F5           *.....55555555*
000020    F5F5F5F5  F5F5F5F5  F5F5F5F5   F5F5F5F5           *5555555555555555*
000030    F5F5F5F5  F5F5F5F5  F5F5F5F5   F5F5F5F5           *5555555555555555*
000040    F5F5F5F5  F5F5F5F5  F5F5F5F5   F5F5F5F5           *5555555555555555*
000050    F5F5F5F5  F5F5F5F5  F5F5F5F5   F5F5F5F5           *5555555555555555*
000060    F5F5F5F5  F5F5F5F5  F5F5F5F5   F5F5F5F5           *5555555555555555*
000070    F5F5F5F5  F5F5F5F5  F5F5F5F5   F5F5F5F5           *5555555555555555*
000080    F5F5F5F5  F5F5F5F5  F5F5F5F5   F5F5F5F5           *5555555555555555*
000090    F5F5F5F5  F5F5F5F5  F5F5F5F5   F5F5F5F5           *5555555555555555*
0000A0    F5F5F5F5  F5F5F5F5  F5F5F5F5   F5F5F5F5           *5555555555555555*
0000B0    F5F5F5F5  F5F5F5F5  F5F5F5F5   F5F5F5F5           *5555555555555555*
0000C0    F5F5F5F5  F5F5F5F5  F5F5F5F5   F5F5F5F5           *5555555555555555*
0000D0    F5F5F5F5  F5F5F5F5  F5F5F5F5   F5F5F5F5           *5555555555555555*
0000E0    F5F5F5F5  F5F5F5F5  F5F5F5F5   F5F5F5F5           *5555555555555555*
0000F0    F5F5F5F5  F5F5F5F5  F5F5F5F5   F5F5F5F5           *5555555555555555*
  
```

==> B(rkpt.), D(ump), G(o), M(emory), P(SW), Q(uit), R(eg.), S(tep), T(race)  
:



# ADD2 Program After 1st Instruction

---

PSW AT BREAK    FFC50000 8F000004

R0-7 :    F4F4F4F4 00000004 F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4  
R8-15:    F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4 00000020 00000068 00000000

000000	5810F010	5820F014	1A125010	F01807FE	*..0.....&.....*
000010	00000004	00000006	F5F5F5F5	F5F5F5F5	*.....55555555*
000020	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000030	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000040	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000050	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000060	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000070	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000080	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000090	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
0000A0	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
0000B0	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
0000C0	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
0000D0	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
0000E0	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
0000F0	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*

==> B(rkpt.), D(ump), G(o), M(emory), P(SW), Q(uit), R(eg.), S(tep), T(race)  
:

Contents of  
word 5  
LOADed to  
R1

# Program After 1st Instruction

C50000 8F000004

R0-7 : F4F4F4F4 00000004 F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4  
R8-15: F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4 00000020 00000068 00000000

000000	5810F010	5820F014	1A125010	F01807FE	*..0.....&.....*
000010	00000004	00000006	F5F5F5F5	F5F5F5F5	*.....55555555*
000020	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000030	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000040	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000050	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000060	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000070	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000080	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000090	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
0000A0	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
0000B0	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
0000C0	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
0000D0	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
0000E0	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
0000F0	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*

==> B(rkpt.), D(ump), G(o), M(emory), P(SW), Q(uit), R(eg.), S(tep), T(race)

:

# ADDRESS of the next instruction

```
PSW AT BREAK   FFC50000 8F000004

R0-7 :  F4F4F4F4 00000004 F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4
R8-15:  F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4 00000020 00000068 00000000

000000    5810F010    5820F014    1A125010    F01807FE    *..0.....&.....*
000010    00000004    00000006    F5F5F5F5    F5F5F5F5    *.....55555555*
000020    F5F5F5F5    F5F5F5F5    F5F5F5F5    F5F5F5F5    *5555555555555555*
000030    F5F5F5F5    F5F5F5F5    F5F5F5F5    F5F5F5F5    *5555555555555555*
000040    F5F5F5F5    F5F5F5F5    F5F5F5F5    F5F5F5F5    *5555555555555555*
000050    F5F5F5F5    F5F5F5F5    F5F5F5F5    F5F5F5F5    *5555555555555555*
000060    F5F5F5F5    F5F5F5F5    F5F5F5F5    F5F5F5F5    *5555555555555555*
000070    F5F5F5F5    F5F5F5F5    F5F5F5F5    F5F5F5F5    *5555555555555555*
000080    F5F5F5F5    F5F5F5F5    F5F5F5F5    F5F5F5F5    *5555555555555555*
000090    F5F5F5F5    F5F5F5F5    F5F5F5F5    F5F5F5F5    *5555555555555555*
0000A0    F5F5F5F5    F5F5F5F5    F5F5F5F5    F5F5F5F5    *5555555555555555*
0000B0    F5F5F5F5    F5F5F5F5    F5F5F5F5    F5F5F5F5    *5555555555555555*
0000C0    F5F5F5F5    F5F5F5F5    F5F5F5F5    F5F5F5F5    *5555555555555555*
0000D0    F5F5F5F5    F5F5F5F5    F5F5F5F5    F5F5F5F5    *5555555555555555*
0000E0    F5F5F5F5    F5F5F5F5    F5F5F5F5    F5F5F5F5    *5555555555555555*
0000F0    F5F5F5F5    F5F5F5F5    F5F5F5F5    F5F5F5F5    *5555555555555555*
```

==> B(rkpt.), D(ump), G(o), M(emory), P(SW), Q(uit), R(eg.), S(tep), T(race)

:

# ADD2 Program After 2nd Instruction

---

PSW AT BREAK    FFC50000 8F000008

R0-7 :    F4F4F4F4 00000004 00000006 F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4  
R8-15:    F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4 00000020 00000068 00000000

000000	5810F010	5820F014	1A125010	F01807FE	*..0.....&.....*
000010	00000004	00000006	F5F5F5F5	F5F5F5F5	*.....55555555*
000020	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000030	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000040	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000050	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000060	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000070	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000080	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000090	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
0000A0	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
0000B0	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
0000C0	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
0000D0	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
0000E0	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
0000F0	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*

==> B(rkpt.), D(ump), G(o), M(emory), P(SW), Q(uit), R(eg.), S(tep), T(race)  
:

# ADD

Contents of  
word 6  
LOADed to  
R2

# am After 2nd Instruction

PSW AT BRK 0000008

R0-7 : F4F4F4F4 00000004 00000006 ← F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4  
 R8-15: F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4 00000020 00000068 00000000

000000	5810F010	5820F014	1A125010	F01807FE	*..0.....&.....*
000010	00000004	00000006	F5F5F5F5	F5F5F5F5	*.....55555555*
000020	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000030	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000040	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000050	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000060	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000070	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000080	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000090	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
0000A0	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
0000B0	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
0000C0	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
0000D0	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
0000E0	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
0000F0	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*

==> B(rkpt.), D(ump), G(o), M(emory), P(SW), Q(uit), R(eg.), S(tep), T(race)  
:

# ADD

Address of the next instruction

# Program After 2nd Instruction

PSW AT BREAK FFC50000 8F000008

R0-7 : F4F4F4F4 00000004 00000006 F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4  
R8-15: F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4 00000020 00000068 00000000

000000	5810F010	5820F014	1A125110	F01807FE	*..0.....&.....*
000010	00000004	00000006	F5F5F5F5	F5F5F5F5	*.....55555555*
000020	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000030	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000040	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000050	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000060	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000070	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000080	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000090	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
0000A0	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
0000B0	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
0000C0	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
0000D0	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
0000E0	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
0000F0	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*

==> B(rkpt.), D(ump), G(o), M(emory), P(SW), Q(uit), R(eg.), S(tep), T(race)

:

# ADD2 Program After 3rd Instruction

---

PSW AT BREAK    FFC50000 8F00000A

R0-7 :    F4F4F4F4 0000000A 00000006 F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4  
R8-15:    F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4 00000020 00000068 00000000

000000	5810F010	5820F014	1A125010	F01807FE	*..0.....&.....*
000010	00000004	00000006	F5F5F5F5	F5F5F5F5	*.....55555555*
000020	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000030	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000040	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000050	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000060	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000070	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000080	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000090	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
0000A0	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
0000B0	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
0000C0	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
0000D0	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
0000E0	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
0000F0	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*

==> B(rkpt.), D(ump), G(o), M(emory), P(SW), Q(uit), R(eg.), S(tep), T(race)  
:

Contents of  
R2 added to  
contents of  
R1

# Program After 3rd Instruction

50000 8F00000A

R0-7 : F4F4F4F4 0000000A 00000006 F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4  
R8-15: F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4 00000020 00000068 00000000

000000	5810F010	5820F014	1A125010	F01807FE	*..0.....&.....*
000010	00000004	00000006	F5F5F5F5	F5F5F5F5	*.....55555555*
000020	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000030	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000040	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000050	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000060	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000070	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000080	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000090	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
0000A0	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
0000B0	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
0000C0	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
0000D0	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
0000E0	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
0000F0	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*

==> B(rkpt.), D(ump), G(o), M(emory), P(SW), Q(uit), R(eg.), S(tep), T(race)

:



# ADI **am** After 3rd Instruction

Address of the next instruction

PSW AT BREAK FFC50000 8F00000A

R0-7 : F4F4F4F4 0000000A 00000006 F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4  
R8-15: F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4 00000020 00000068 00000000

000000	5810F010	5820F014	1A125010	F01807FE	*..0.....&.....*
000010	00000004	00000006	F5F5F5F5	F5F5F5F5	*.....55555555*
000020	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000030	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000040	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000050	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000060	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000070	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000080	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000090	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
0000A0	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
0000B0	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
0000C0	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
0000D0	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
0000E0	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
0000F0	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*

==> B(rkpt.), D(ump), G(o), M(emory), P(SW), Q(uit), R(eg.), S(tep), T(race)  
:

# ADD2 Program After 4th Instruction

---

PSW AT BREAK    FFC50000 8F00000E

R0-7 :    F4F4F4F4 0000000A 00000006 F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4  
R8-15:    F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4 00000020 00000068 00000000

000000	5810F010	5820F014	1A125010	F01807FE	*..0.....&.....*
000010	00000004	00000006	0000000A	F5F5F5F5	*.....5555*
000020	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000030	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000040	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000050	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000060	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000070	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000080	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000090	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
0000A0	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
0000B0	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
0000C0	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
0000D0	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
0000E0	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
0000F0	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*

==> B(rkpt.), D(ump), G(o), M(emory), P(SW), Q(uit), R(eg.), S(tep), T(race)

:

Contents of  
R1 STOREd  
to word 7

# Program After 4th Instruction

250000 8F00000E

R0-7 : F4F4F4F4 0000000A 00000006 F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4  
R8-15: F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4 00000020 00000068 00000000

000000	5810F010	5820F014	1A125010	F01807FE	*..0.....&.....*
000010	00000004	00000006	0000000A	F5F5F5F5	*.....5555*
000020	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000030	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000040	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000050	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000060	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000070	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000080	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
000090	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
0000A0	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
0000B0	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
0000C0	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
0000D0	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
0000E0	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*
0000F0	F5F5F5F5	F5F5F5F5	F5F5F5F5	F5F5F5F5	*5555555555555555*

==> B(rkpt.), D(ump), G(o), M(emory), P(SW), Q(uit), R(eg.), S(tep), T(race)

:

# ADP Program After 4th Instruction

Address of the next instruction

```
PSW AT BREAK   FFC50000 8F00000E

R0-7 :  F4F4F4F4 0000000A 00000006 F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4
R8-15:  F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4 00000020 00000068 00000000

000000    5810F010    5820F014    1A125010    F01807FE    *..0.....&.....*
000010    00000004    00000006    0000000A    F5F5F5F5    *.....5555*
000020    F5F5F5F5    F5F5F5F5    F5F5F5F5    F5F5F5F5    *5555555555555555*
000030    F5F5F5F5    F5F5F5F5    F5F5F5F5    F5F5F5F5    *5555555555555555*
000040    F5F5F5F5    F5F5F5F5    F5F5F5F5    F5F5F5F5    *5555555555555555*
000050    F5F5F5F5    F5F5F5F5    F5F5F5F5    F5F5F5F5    *5555555555555555*
000060    F5F5F5F5    F5F5F5F5    F5F5F5F5    F5F5F5F5    *5555555555555555*
000070    F5F5F5F5    F5F5F5F5    F5F5F5F5    F5F5F5F5    *5555555555555555*
000080    F5F5F5F5    F5F5F5F5    F5F5F5F5    F5F5F5F5    *5555555555555555*
000090    F5F5F5F5    F5F5F5F5    F5F5F5F5    F5F5F5F5    *5555555555555555*
0000A0    F5F5F5F5    F5F5F5F5    F5F5F5F5    F5F5F5F5    *5555555555555555*
0000B0    F5F5F5F5    F5F5F5F5    F5F5F5F5    F5F5F5F5    *5555555555555555*
0000C0    F5F5F5F5    F5F5F5F5    F5F5F5F5    F5F5F5F5    *5555555555555555*
0000D0    F5F5F5F5    F5F5F5F5    F5F5F5F5    F5F5F5F5    *5555555555555555*
0000E0    F5F5F5F5    F5F5F5F5    F5F5F5F5    F5F5F5F5    *5555555555555555*
0000F0    F5F5F5F5    F5F5F5F5    F5F5F5F5    F5F5F5F5    *5555555555555555*
```

==> B(rkpt.), D(ump), G(o), M(emory), P(SW), Q(uit), R(eg.), S(tep), T(race)  
:

# Implicit Addresses and USING

In Which We Make  
Assembler Programming  
"Easy"



# A Slight Change

---

- What if we want to make a change to our original program? Maybe use register 12 instead of R15 for the base register
- That's no problem - just insert one LR instruction in front of the first LOAD, copying R15 to R12
- Then change all instructions which use R15 as the base register to use R12!

# A Slight Change

---

- But that moves everything down and our data areas are no longer where they were, and that means we have to re-calculate the displacements

# Updated ADD2 Demo Program

---

- \* This program adds two numbers that are taken
- \* from the 6th and 7th words of the program.
- \* The sum is stored in the 8th word.
- \* (How do we know which words?)

```
ADD2      CSECT
          LR      12,15      Copy addr of 1st inst
          L       1,??(,12)  Load 1st no. into R1
          L       2,??(,12)  Load 2nd no. into R2
          AR      1,2        Get sum in R1
          ST      1,??(,12)  Store sum
          BCR     B'1111',14  Return to caller
          DC      F'4'       Fullword initially 4
          DC      F'6'       Fullword initially 6
          DS      F          Rsrvd only, no init
          END      ADD2
```



# Updated ADD2 Demo Program

---

- \* This program adds two numbers that are taken
- \* from the 6th and 7th words of the program.
- \* The sum is stored in the 8th word.

```
ADD2      CSECT
          LR      12,15      Copy addr of 1st inst
          L       1,20(,12)  Load 1st no. into R1
          L       2,24(,12)  Load 2nd no. into R2
          AR      1,2        Get sum in R1
          ST      1,28(,12)  Store sum
          BCR     B'1111',14 Return to caller
          DC      F'4'       Fullword initially 4
          DC      F'6'       Fullword initially 6
          DS      F          Rsrvd only, no init
          END      ADD2
```

# Updated ADD2 Demo Program

---

- As you are looking at the previous and next slides, you should ask yourself
  - why did displacements increase by 4 and not 2?
  - why did the locations of the three data areas increase by four bytes, when we added only a two-byte instruction at the beginning of the program?
- Specifically, notice that BCR occupies only two bytes of storage

# Updated ADD2 Demo Program Assembled

---

LOC	OBJECT	CODE	SOURCE	STATEMENT
000000			ADD2	CSECT
000000	18CF			LR 12,15
000002	5810	C014		L 1,20(,12)
000006	5820	C018		L 2,24(,12)
00000A	1A12			AR 1,2
00000C	5010	C01C		ST 1,28(,12)
000010	07FE			BCR B'1111',14
000014	00000004			DC F'4'
000018	00000006			DC F'6'
00001C				DS F
				END ADD2

# Updated ADD2 Demo Program Assembled

---

LOC    OBJECT    CODE    SOURCE    STATEMENT

```

ADD2            CSECT
LR             12,15
L              1,20(,12)
L              2,24(,12)
AR             1,2
ST             1,28(,12)
BCR            B'1111',14
DC             F'4'
DC             F'6'
DS             F
END            ADD2
  
```

The next field is fullword aligned, even though this instruction ends two bytes "early"

```

00000C 5000 C01C
000010 07FE
000014 00000004
000018 00000006
00001C
  
```

# There Must Be an Easier Way!

---

- What if our program has hundreds or thousands of instructions? Do we have to calculate displacements for every data area?
- Or, what if we change something? Do we have to re-calculate all displacements?
- Fortunately, we don't, or no one would ever use the assembler

# Use Labels!

---

- Remember: "Object Code - Nothing Else Matters"
- So, as long as the assembler generates the correct object code, we can do "anything we want" by way of writing source instructions

# Use Labels!

---

- We want to use labels - implicit addresses - instead of explicit base and displacement, and let the assembler do the calculations
- We can do this as long as we tell the assembler what base register and base address to use

# The USING Instruction

---

- So, instead of writing a base and displacement form of address, we will simply place a label on the storage area definition, then write that label in any instruction operand which references that data
- The assembler must convert the implicit reference to a valid base and displacement



# The USING Instruction

---

- We tell the assembler which base register it can use and what base address the register has via the USING assembler instruction (or directive)
- The USING instruction is not executable, and has the following format:

**USING baseaddress, baseregister**

# The USING Instruction

---

- This tells the assembler how to choose the correct base register and how to assign displacements
- USING is your "promise" that at execution time the base register will contain the memory address at which the base address has been loaded

# The USING Instruction

---

- Be sure to compare the object code generated by the next version of the program, with the object code generated by the version without USING and labels
- The object code is exactly the same, so the program will execute in exactly the same way
- "Object code - nothing else matters"

# Demo Program with Labels

---

- \* This program adds two numbers that are taken
- \* from WORD1 and WORD2 in the program.
- \* The sum is stored in WORD3.

```
ADD2      CSECT
          LR      12,15          Copy addr of 1st inst
          USING   ADD2,12       Tell assembler
          L       1,WORD1       Load 1st no. into R1
          L       2,WORD2       Load 2nd no. into R2
          AR      1,2           Get sum in R1
          ST      1,WORD3       Store sum
          BCR     B'1111',14    Return to caller
WORD1     DC      F'4'         Fullword initially 4
WORD2     DC      F'6'         Fullword initially 6
WORD3     DS      F           Rsrvd only, no init
          END      ADD2
```

# Demo Program with Labels

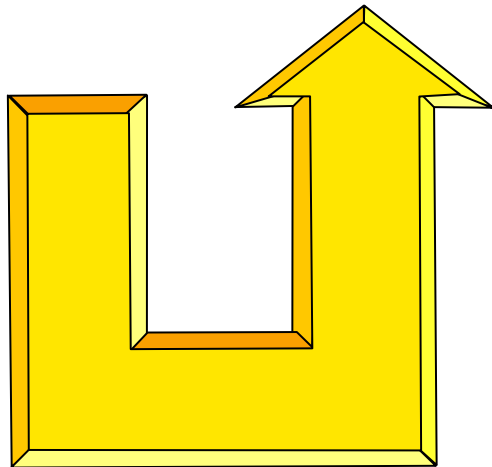
---

LOC	OBJECT	CODE	SOURCE	STATEMENT
000000			ADD2	CSECT
000000	18CF			LR 12,15
000002				USING ADD2,12
000002	5810	C014		L 1,WORD1
000006	5820	C018		L 2,WORD2
00000A	1A12			AR 1,2
00000C	5010	C01C		ST 1,WORD3
000010	07FE			BCR B'1111',14
000014	00000004		WORD1	DC F'4'
000018	00000006		WORD2	DC F'6'
00001C			WORD3	DS F
				END ADD2



# The Condition Code and Branching

In Which We Learn How  
to Go Back Where We  
Came From, or Maybe  
Go Somewhere Else



# The Condition Code

---

- The ADD (A, AR) and SUBTRACT (S, SR) instructions have an additional characteristic not yet mentioned - they set the 2-bit condition code in the following way

<u>CC</u>	<u>Meaning</u>
0	Result is 0
1	Result is $< 0$
2	Result is $> 0$
3	Overflow occurred

# The Condition Code

---

- Another pair of instructions, COMPARE (C, CR) also set the condition code, but its values are interpreted in a slightly different way

<u>CC</u>	<u>Meaning</u>
0	Contents equal
1	1st operand val < 2nd operand val
2	1st operand val > 2nd operand val
3	---- (not set)



# The Condition Code

---

- The condition code is actually two bits of the PSW (bits 34 & 35 in ASSIST/I, bits 18 & 19 in z/Architecture)
- In order to test for all possible combinations of the four CC values, we can use a four-bit mask

Mask	B	'	x	x	x	x	'	(recall B means binary)
CC			0	1	2	3		(CC value tested)

# BRANCH ON CONDITION

---

- So a bit mask of  $B'1010'$  will test for condition codes 0 and 2
  - The condition code can be tested in only one way - by using the conditional branch instructions BC and BCR
- (There are other conditional branch instructions, but they are beyond the scope of this Boot Camp)

# BRANCH ON CONDITION

---

- The two forms of BRANCH ON CONDITION are

[RX] label BC B 'mask' , D<sub>2</sub> ( X<sub>2</sub> , B<sub>2</sub> )

[RR] label BCR B 'mask' , R<sub>2</sub>

- The encoded form of each instruction is

[RX]  $h_{OP} h_{OP} h_{M1} h_{X2} h_{B2} h_{D2} h_{D2} h_{D2}$

[RR]  $h_{OP} h_{OP} h_{M1} h_{R2}$

- Note that this has an M<sub>1</sub> subscript instead of the R<sub>1</sub> seen in previous instructions

# BRANCH ON CONDITION

---

- If the mask part of the BC or BCR has a 1 in a position corresponding to the current setting of the CC, the next instruction to be executed will be the one whose address is given by the second operand
- Otherwise, the next instruction will be the one whose address is already in the PSW, the one in memory immediately after the branch instruction

# BRANCH ON CONDITION

---

- Notice that `BCR B'1111',14` is an unconditional branch to the address in R14 and is the instruction used to end execution of a program
- The mask `B'1111'` matches all four possible CC values

# A Program Which Tests the Condition Code by Branching

---

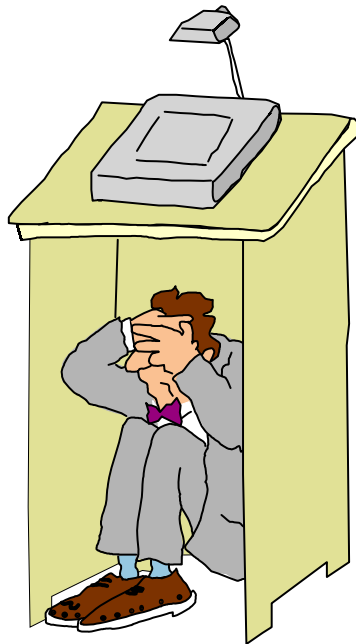
```
MAX          CSECT
             USING  MAX,15
             L      1,W1      Get First number
             L      2,W2      Get second number
             CR     1,2       Compare
             BC     B'0010',ONEHIGH Branch if W1 high
             ST     2,W3       Else store second number
             BCR    B'1111',14 Return to caller
ONEHIGH     ST     1,W3       Store first number
             BCR    B'1111',14 Return to caller
W1          DC     F'321'     First number
W2          DC     F'123'     Second number
W3          DS     F          Max of first and second
             END    MAX
```

# A Program Which Tests the Condition Code by Branching

LOC	OBJECT	CODE	SOURCE	STATEMENT
000000			MAX	CSECT
000000				USING MAX,15
000000	5810	F01C		L 1,W1
000004	5820	F020		L 2,W2
000008	1912			CR 1,2
00000A	4720	F014		BC B'0010',ONEHIGH
00000E	5020	F024		ST 2,W3
000012	07FE			BCR B'1111',14
000014	5010	F024	ONEHIGH	ST 1,W3
000018	07FE			BCR B'1111',14
00001C	00000141		W1	DC F'321'
000020	0000007B		W2	DC F'123'
000024			W3	DS F
				END MAX

# X-Instructions and ASSIST/

In Which We Get Data  
Into and Out of an  
Assembler Program





# Character Data

---

- At this point, we've seen only numeric data, represented as binary fullwords, and created in assemblies as data type F
- But binary data is much too difficult to type, so a different, external, format is used
- In order to work with external data, we will have to know how to represent characters in single bytes of memory

# Character Data

---

- There are three standards in current use for representing character data: EBCDIC, ASCII, and Unicode
- We will stick to the EBCDIC representation in these sessions, since
  - It's the only representation known to ASSIST/I
  - It's the representation most common in z/Architecture

# Some EBCDIC Representations

A	C1
B	C2
C	C3
D	C4
E	C5
F	C6
G	C7
H	C8
I	C9
J	D1
K	D2
L	D3
M	D4

N	D5
O	D6
P	D7
Q	D8
R	D9
S	E2
T	E3
U	E4
V	E5
W	E6
X	E7
Y	E8
Z	E9

0	F0
1	F1
2	F2
3	F3
4	F4
5	F5
6	F6
7	F7
8	F8
9	F9

blank	40
-------	----

.	4B
<	4C
(	4D
+	4E
&	50
!	5A
\$	5B
*	5C
)	5D
-	60
>	6E
@	7C
'	7D

# Character Data

---

- We can create character data in our programs by using data type C in a DC instruction

```
MESSAGE DC C'1HELLO WORLD!'
```

- The object code generated by this DC is

```
F1C8C5D3D3D640E6D6D9D3C45A
```

# Character Data

---

- The 1 in front of the message is a "printer carriage control" character which will cause the message to be printed at the top of a new page
- Other printer carriage control characters are
  - ' ' (blank) - Single space
  - ' 0 ' - Double space
  - ' - ' - Triple space

# The X-Instructions of ASSIST/I

---

- For assembler programmers, moving data into and out of a program is a very complex affair, involving numerous operating system I/O "macros"
- For users of ASSIST and ASSIST/I, however, the process is extremely simple and requires only the use of a few non-standard instructions

# The X-Instructions of ASSIST/I

---

- In ASSIST and ASSIST/I, these instructions assemble just like "real" instructions, but they are not part of the standard (or any other) instruction set
  - X-instructions are available as "macros" for the High Level Assembler
- The first is XREAD, which reads a record from the input file and places as many bytes as requested (up to 80) starting at the memory address provided; this area is called a "buffer"

# The XREAD Instruction

---

■ **label XREAD  $D_1(X_1, B_1), L_2$**

$D_1(X_1, B_1)$  is the address of the first byte of the input buffer

$L_2$  is the number of characters (1 to 80) to be transferred from the input record

■ The input record is expected to contain character data



# The XREAD Instruction

---

- The XREAD instruction sets the condition code
  - 0 - Read was successful, data placed in memory
  - 1 - end-of-file encountered, no data transferred
  - 2 - [not set]
  - 3 - [not set]

# The XDECI Instruction

---

- Numbers are input as characters in the EBCDIC representation and must be converted to binary before we can manipulate them (e.g., ADD or SUBTRACT)
- Input conversion is performed by XDECI

# The XDECI Instruction

---

- The XDECI instruction converts an EBCDIC numeric value in memory to a binary numeric value and places it in a register

■ **label      XDECI      R<sub>1</sub>, D<sub>2</sub>(X<sub>2</sub>, B<sub>2</sub>)**

R<sub>1</sub> is the register to hold the result

D<sub>2</sub>(X<sub>2</sub>, B<sub>2</sub>) is the memory location at which the search for numeric characters begins

# The XDECI Instruction Logic - Part 1

---

1. Start at  $D_2(X_2, B_2)$ , scan for the first non-blank
2. If the first non-blank is anything but + or - or a decimal digit, set the condition code to 3 and quit
3. Otherwise, 1 to 9 digits are scanned and the resulting number is converted to binary and placed in register  $R_1$

# The XDECI Instruction Logic - Part 2

---

4. Register 1 is set to the address of the first non-digit (so register operand  $R_1$  should not be 1!)
5. If ten or more digits are found, register 1 is set to the address of the first subsequent non-digit, the condition code is set to 3, and register  $R_1$  is unchanged

# The XDECI Instruction Logic - Part 3

---

- XDECI sets the condition code
  - 0 - The number converted was 0
  - 1 - The number converted was  $< 0$
  - 2 - The number converted was  $> 0$
  - 3 - Non-numeric scanned, or too many digits
- To avoid scanning past the end of the buffer, include a "scan stopper":

<b>INPUT</b>	<b>DS</b>	<b>CL80</b>	Input Buffer
	<b>DC</b>	<b>C' * '</b>	Non-digit stopper

# The XDECO Instruction

---

- The XDECO instruction converts a binary numeric value in a register to an EBCDIC numeric value in memory (action opposite that of XDECI)
- Positive numbers appear with no sign, but negative numbers appear with a leading minus sign

# The XDECO Instruction

---

■ **label      XDECO       $R_1, D_2(X_2, B_2)$**

$R_1$  is the register containing the binary number to convert

$D_2(X_2, B_2)$  is the beginning memory location, usually within a print buffer, to store the EBCDIC number, right-justified, occupying 12 bytes



# The XPRNT Instruction

---

- The XPRNT instruction will print one line of output of maximum length 133 characters, including the carriage control (CC)

■ **label XPRNT  $D_1(X_1, B_1), L_2$**

$D_1(X_1, B_1)$  is the address of the first byte of the output buffer (the CC character)

$L_2$  is the number of characters (including carriage control) to be transferred from memory to the print line

# Example With X-Instructions (1 of 2)

## [demob.asm]

---

\* THIS PROGRAM READS DATA CARDS EACH HAVING TWO  
\* NUMBERS. THE SUM OF THE NUMBERS IS PRINTED.

\*

```
SUMUP      CSECT  
           USING SUMUP,15
```

\*

```
           XPRNT  HEADING,28      PRINT PAGE HDR  
           XREAD  CARD,80        READ 1ST CARD
```

\*

```
CHECKEOF  BC      B'0100',EXIT   BR ON EOF
```

\*

```
           XDECI  2,CARD          ASSUME BOTH NUMS  
           XDECI  3,0(,1)        ARE VALID
```

\*

```
           AR      2,3           CALCULATE THE SUM
```

# Example With X-Instructions (2 of 2)

## [demob.asm]

---

```

XDECO 2,OUTPUT          PRINTABLE FORM
*                          INTO PRINT LINE
XPRNT CRG,13            PRINT THE SUM
*                          AFTER SINGLE SPACE
XREAD CARD,80          TRY ANOTHER READ
BC B'1111',CHECKEOF    GO CHECK FOR EOF
*
EXIT BCR B'1111',14    TERMINATE PROGRAM
*
CARD DS CL80           INPUT BUFFER
*
CRG DC C' '            SINGLE SPACE CC
OUTPUT DS CL12         SUM GOES HERE
HEADING DC C'1THIS IS THE OUTPUT OF SUMUP'
END SUMUP
```

# What's Next?

---

- Tomorrow, we we will construct program loops
- Also, we will see one of the simplest and most challenging machine instructions, the LOAD ADDRESS instruction (when you are comfortable with addresses, you will have a good understanding of assembler)
- We will also do more arithmetic (multiplication and division) if we have time