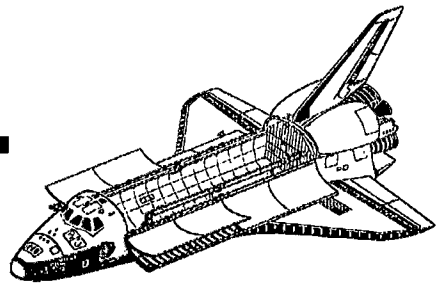


AP 101S Assembly Language Introduction

– Reading and Understanding Assembly Code

OTP275.04

Christopher C. Marchant

PASS SSW Development

February, 2004

USA
United Space Alliance

AP 101S Assembly Language Introduction

- Course Length: 4 hours
- Course Number: OTP275.04
- Prerequisites: None (although a previous introduction to low level computer programming is useful)
- Intended Audience: Shuttle PASS Flight Software Developers, Verifiers, and Requirement Analysts
- Objectives:
  - Provide a brief AP-101S hardware (CPU) introduction
  - Introduce the AP-101S Assembler instruction set
- Content: 2 Training Sessions of 2 hours each

Suggested References

- SPACE SHUTTLE MODEL AP-101S PRINCIPLES OPERATION WITH SHUTTLE INSTRUCTION SET
  - a.k.a. the P.O.O.
  - SFOC-OE0001 (dated 12/18/98)
- AP101 ASSEMBLER USER'S GUIDE
  - USA000516, OI30 (dated 2/27/03)

Assembly Vocabulary - 1

Assembly: The automatic translation of symbolic code into machine code; An assembly language; Assembly code.

Assembler: A program that produces executable machine code from symbolic assembly language.

From: The American Heritage® Dictionary of the English Language, Fourth Edition

Assembly Vocabulary - 2

- Immediate Data: Data which is embedded in the command using it; especially useful for small numbers which can be represented in fewer bits than the number needed to list a separate address pointer
- Main Storage: (or just plain Storage) GPC RAM
- Offset: The difference between two address locations in RAM
- Register: A data storage region of the CPU; registers can be accessed at least twice as fast as any location in RAM
- Load: The act of copying data from storage to a processor register
- Store: The act of copying data from a processor register to storage
- Branch: Basically a GOTO statement; can cause non-linear execution of the instructions within memory; called a "jump" in some systems
- Test (bit): Determine if the bit contains a zero or one
- Set (bit): Makes the contents of a bit equal one
- Floating Point Number: A real number (mathematics); a scalar (HAL/S)
- Fixed Point Number: An integer (or a number made to look like an int)

General Purpose Computer (GPC) Basics – 1

- AP-101S model computers first flew on STS-37 in 1991
- The included CPU uses a modified version of the IBM System 360/370 assembly language instruction set
- Supports 32 bit operations
  - A "Fullword" is 32 bits long
  - A "Halfword" is 16 bits long
  - A "Doubleword" is 64 bits long
- 40 MHz clock speed
  - The P.O.O. has estimated execution times for each instruction
  - Because of the CPU microcode design, all instruction execution times are multiples of 250 nsec. (which is 10 clock ticks)
    - NOTE: The time required for code to execute (in any system) is proportional to the number of clock ticks required to complete each instruction and inversely proportional to the system clock speed

General Purpose Computer (GPC) Basics – 2

- An AP-101S can be thought of as two separate computers condensed into one case
  - CPU, which contains the main processor and RAM
  - Input/Output Processor (IOP), which handles data traffic on:
    - External buses
      - 24 in total (ICC, Flight Critical, DK, MMU, PL, LDB, PCMMU)
      - Each GPC has its own PCMMU bus, but the remaining 23 busses are shared by all GPCs
    - Discretes: crew cabin talkbacks, crew switches, synchronization lines, etc.
- The original AP-101B GPCs filled two cases each
  - The AP-101S upgrade
    - More than doubled the main memory
    - Tripled the CPU speed (measured in MIPS)
    - Cut the GPC size roughly in half (in both volume and weight)
    - Cut the power consumption (in Watts) by over 10%

System Timers – 1

- With many CPUs, the "program counter" is the name of the register that holds the address of the next instruction to be executed
  - The GPCs DO NOT follow this convention
    - Each PSW contains an "Instruction Counter" which identifies the address of the next instruction to be executed
      - PSW stands for "Program Status Word" and will be discussed later
    - The GPC "Program Counters" are actually timers
- PC1 and PC2 can be used to time events within a single minor cycle (normally 40 milliseconds in length), and to time the minor cycle itself

System Timers – 2

- There are 2 Program Counters in each GPC
  - They are called
    - Program Counter 1 (PC1)
    - Program Counter 2 (PC2)
  - They are each 32 bits wide (one Fullword)
    - High order Halfword resides in the PSA
      - The PSA (or Preferential Storage Area) is in RAM, addresses 0x000 through 0x13F
      - The PSA is reserved for hardware use; software or its data can be loaded into the PSA
    - Low order Halfword resides in CPU Hardware
  - Decrementing interval timers
    - Once per Microsecond
    - Generate an interrupt upon roll over of the complete set of 32 bits from 0000 0000 to FFFF FFFF

GPC Main Storage (RAM) – 1

- RAM addressing is on Halfword boundaries
- RAM is divided into 16 sectors
  - Sectors are numbered from 0x0 to 0xF (0-15)
  - 0x8000 (32,768) Halfwords per sector
  - 1 Megabyte of RAM in total
  - 19 bits are required for full addressing of RAM
    - IOP can only handle 18 bit addresses
      - Meaning the IOP can only directly access sectors 0 – 7
        - To load the G3 archive during the G901 to G101 OPS transition, the IOP copies the G3 software into the lower half of memory, and then the CPU copies that information into the upper half of memory
    - CPU instructions can only handle 16 address bits
      - The remaining address information is stored for the CPU in the BSR/DSR/DSE

GPC Main Storage (RAM) – 2

- Store Protect (SP) bits – three allocated per memory address, but hidden from the programmer
  - SP Bits are physically separated from the storage area
  - Set or cleared by the software loader as it fills the GPC RAM, based on whether that address is in a protected or unprotected load block
    - Load blocks are sections of memory allocated by the Software Architect
    - Protected blocks contain CPU code file(s), while unprotected blocks usually contain data (variables and constants)
  - Protected blocks can have constants defined within code modules
    - These data words are also protected even though they are data
  - Related error messages:
    - An attempt by the CPU to store data into a protected location results in a "Store Protect Error" which is a type of "Program Check"
    - An attempt by the CPU to execute non-protected data results in an "Instruction Monitor"

GPC Main Storage (RAM) – 3

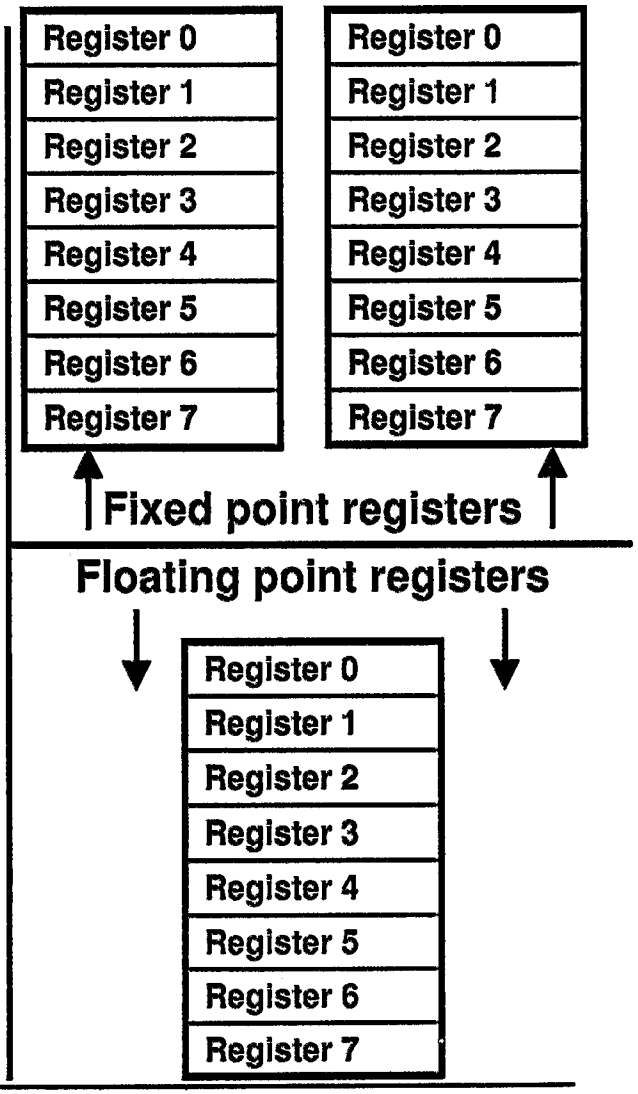
- Store Protect (SP) bits – continued
  - IOP code (for both the BCE and the MSC) executes successfully even though the SP bits are not set (at least for PASS)
    - BCE (Bus Control Elements)
      - One for each external bus, plus one for test purposes
      - BCE programs for bypassable transactions must be un-protected to allow the efficient bypass of the problem element (because FCOS has self-modifying code)
    - MSC (Master Sequence Controller)
      - I/O Manager which controls and monitors the 24 BCEs with attached busses, and interfaces with the CPU

GPC Main Storage (RAM) – 4.

- Six EDAC (Error Detection And Correction) bits per address
  - Used in a scrub of GPC memory (runs every 1.67 seconds)
  - If a single bit within a Halfword of memory is flipped
    - Likely due to a radiation hit
    - The memory scrub detects and corrects that error
      - Seen in Telemetry as a single jump in the Soft Error Count
    - Memory reads prior to a scrub succeed since the EDAC hardware will correct the memory values sent to the CPU
      - The memory location is not corrected until its next scrub
  - If two bits within a Halfword are flipped
    - Highly unlikely; but could be caused by radiation
    - GPC hardware detects, but cannot correct, the error
    - Registers as a newly detected Soft Error every 1.67 seconds
      - Seen in Telemetry as a recurring increase in the Soft Error Count – The count can max out eventually
    - An attempt to use such a memory location results in a machine check and a GPC halt

CPU Register Basics – 1

- Two sets of General Purpose, Fixed Point registers (set 1 used by FCOS, set 0 otherwise)
  - Eight registers per set
  - 32 bits per register
    - 16 bit numbers are *usually* stored in the high order bits (0-15, numbered from left to right), but this varies by instruction
  - Each General Purpose register has an additional 4 bit DSE register associated with it, used only for addressing operations
- One set of 8 Floating Point registers
  - 32 bits per register
  - Used in pairs for double precision calculations
    - Consecutively numbered registers must be used
- Registers in all sets are numbered from 0 to 7



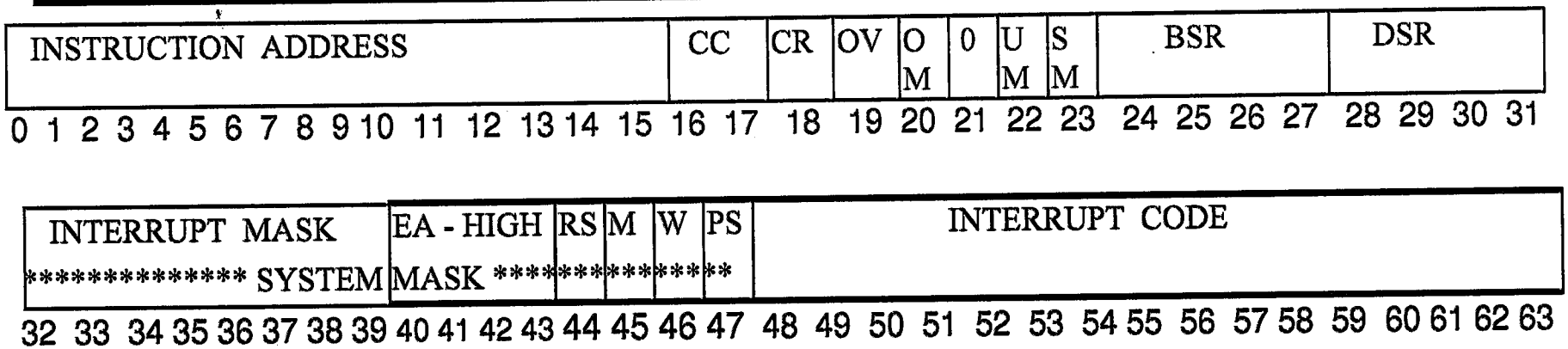
CPU Register Basics – 2.

- Each PASS process is given access to all Floating Point Registers and one set of Fixed Point Registers
  - Whenever FCOS needs to change which application process is active (the process with CPU access), the "context" of the current application is saved, and the context of the new process is loaded
    - The context includes the contents of the following when when the process was interrupted:
      - All registers used by applications (8 GPR and 8 FPR)
      - DSEs 0-3
      - The PSW (Program Status Word) relating to the application
    - The PCT Chain (Process Control Table Chain) holds this data
      - See the PASS Flight Software AP-101 Dump Analysis for more information
  - Whenever a process is resumed, its saved context is reloaded
  - When an application process is interrupted by FCOS, the PSW of the application is saved, but register values do not need to be stored
    - FCOS and applications don't share general purpose registers

CPU Program Status Word (PSW) – 1

- Tracks the basic information required for program execution & control
  - Address of next instruction to be executed
  - Status of the system in relation to the program executing
    - Condition Code
      - Changed by *some* instructions (compare registers, test bit, etc.) to provide feedback on their execution
      - Most often needs to be read after compare instructions and test instructions
    - System / Interrupt Mask
- The PSW is updated for every instruction execution
- The PSW is swapped out to process hardware interrupts
  - The current PSW is stored in the Preferred Storage Area (PSA) of RAM, in the "Interrupt Old PSW"
    - PSA begins at address 0x00000
  - A new PSW, defined for the current Interrupt, is loaded

CPU Program Status Word (PSW) – 2



- Next Instruction Address
- CC - Condition Code
- CR - Carry Status Bit Indicator
- OV - Overflow Status Bit Indicator
- OM - Fixed Point Arithmetic Overflow Mask*
- UM - Floating Point Exponent Underflow Mask*
- SM - Significance Mask*
- BSR - Branch Sector Register
- DSR - Data Sector Register
- Interrupt Mask (PC1, PC2, Instruction Monitor, Ext Interrupts 0 - 4)*
- EA-HIGH - SVC High Order EA Bits
- RS - Register Set Selection
  - (By PASS convention: 1 = FCOS, 0 = other)
- M - Machine Check Mask*
- W - Wait State Bit (1 = WAIT)
- PS (1= Problem, 0=Supervisor)
- Interrupt Code for Program Check, Machine Check, and Ext Interrupts, OR 16 Bit EA for SVC

* 0 = Interrupt Inhibited / 1 = Interrupt Allowed

CPU Program Status Word (PSW) – 3

PSW Bits and Their Uses

- **0 through 15 and 24 through 27: Address of the next instruction to be executed.**
- **16, 17: Condition code describing the results of certain arithmetic, logical and I/O instructions (test bit, compare integers, etc.). These are not error codes.**
- **18: Carry status bit indicator**
  - An integer operation resulting in a value longer than 32 bits.
- **19: Overflow status bit indicator**
  - The final characteristic in a floating point operation exceeds 127 (max value of 7 bits) with a non-zero fraction.
- **20: Fixed Point Arithmetic Overflow Mask. Interrupt inhibited for PASS.**
  - An integer operation resulting in a value with absolute value longer than 31 bits.
- **21: Reserved for future use. Interrupt inhibited for PASS.**
- **22: Floating Point Exponent Underflow Mask. Interrupt inhibited for PASS.**
  - A floating point operation produces a final characteristic less than zero and a non-zero fraction. With the interrupt inhibited, the result is made a true zero.
- **23: Significance Mask. Interrupt inhibited for PASS.**
  - A floating point addition or subtraction resulting in a zero fraction.

CPU Program Status Word (PSW) – 4

PSW Bits and Their Uses

- 24 through 27: Branch Sector Register - replaces the high-order bit of a branch address when that bit is a 1. Otherwise, an implied sector register of 0000 replaces the high-order bit.
- 28 through 31: Data Sector Register - replaces the high-order bit of a data address when that bit is a 1. See "Expanded Addressing" for details when bit 0 is a zero.
- 32 through 47: System Mask – Set to 0xA00C for FCOS and 0xFC05 otherwise.
  - 32: PC1 Interrupt Mask. Enabled for PASS.
  - 33: PC2 Interrupt Mask. Enabled for PASS when FCOS is not running.
  - 34: Instruction Monitor Mask. Enabled for PASS
  - 35 through 39: The remaining five interrupt masks include I/O end conditions, other application dependent interrupts, and timer overflow conditions. Mask bits 35 through 37 are enabled for PASS when FCOS is not running. The last two mask bits (38 and 39) are unused.
  - 40 through 43: EA-High - For an SVC interrupt, the 4-bit extension to make a 19-bit effective address. Set to 0000 for PASS.

CPU Program Status Word (PSW) – 5.

PSW Bits and Their Uses

- 44: Register Select Field – Determines which of the two sets of general registers is in current use. When this bit is a zero (for non-FCOS activity), register set 0 is used; when this bit is one (for FCOS), register set 1 is used.
- 45: Machine Check Mask - When this bit is a zero, machine check interrupts detected by the CPU are inhibited. For PASS, mask is set to one.
- 46: Wait State - When this bit is a zero, the CPU is in the processing (run) state. When this bit is a one, the CPU is in the Wait state. Bit only visible when set to zero (run).
- 47: Problem/Supervisor - When this bit is a zero, the CPU is in the supervisor state and privileged instructions can be executed. When this bit is a one, the CPU is in the problem state and attempts to execute privileged instructions are inhibited and an interrupt is generated. FCOS runs in the Supervisor state, while all other software runs in the Problem state.

- 48 through 63 are reserved for the interrupt code that uses this PSW.

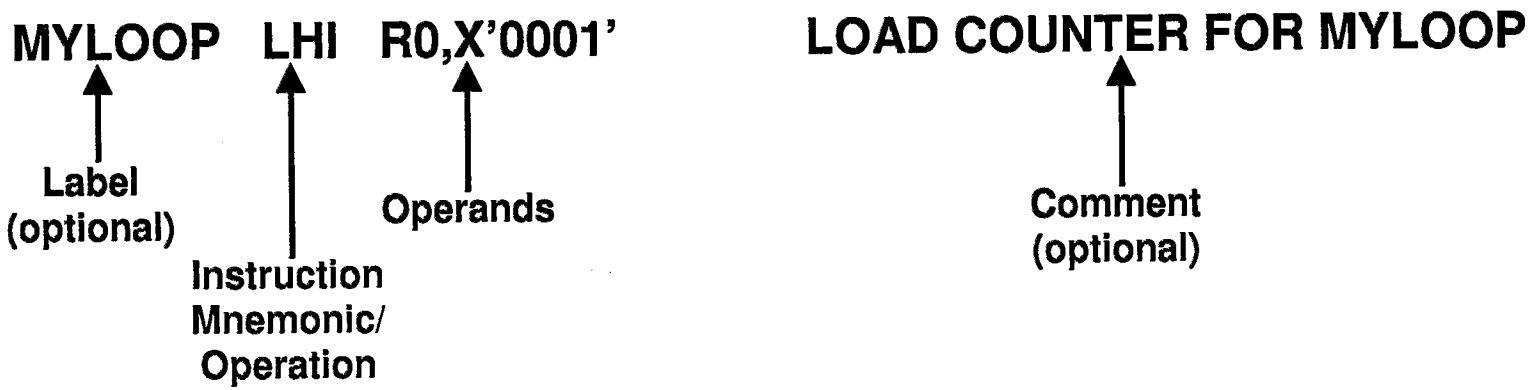
NOTE: FCOS does not include the UI and SC portions of SSW.

Interrupt Protection

- 2 HAL/S macros are available which disable interrupts
  - INTRPT_PROT_NS(HAL_STMTS);
    - Where HAL_STMTS is one or more lines of HAL code
    - No CS/RS syncing is performed by this SVC
  - INTRPT_PROT_DO(LABEL); ... INTRPT_PROT_END(LABEL);
    - The HAL statements to be protected are placed between these two lines
    - RS syncing is performed when the DO statement is executed (but not when the END statement is executed)
- These statements alter the contents of the PSW for the currently executing process
  - Most interrupts are masked off (prevented from being processed) while the interrupt protection is being provided
    - Instruction Monitor and PC1 timer interrupts are still allowed
      - PC1 timer interrupts are significant to PASS only during initialization and not during normal operations

Instruction Layout

- Instructions have four parts:



Note: While LHI (Load Halfword Immediate) is a legitimate Assembler instruction, there is not a corresponding CPU instruction... The Assembler will replace the LHI instruction with a LA (Load Address) instruction when producing the actual machine code.

Also Note: The operands used to identify registers can be aliased differently in each code module. For examples in this class, R0 would be General Purpose Register 0 and FP3 would be Floating Point Register 3.

SAMPLE CODE – Compiler Generated ASSEMBLER LISTING

Address	Instruction	Resolved Address	Label	Command	Operands
0000088			ST#301	EQU	*
00088	B27D 0001	001F		SB	31(R1),1
000008A			ST#302	EQU	*
000008A			LBL#57	EQU	*
0008A	B39D 0400	0027		TB	39(R1),1024
0008C	DC24	0096		BCF	4,*+10
000008D			LBL#60	EQU	*
0008D	B37D 0001	001F		TB	31(R1),1
0008F	DB18	0096		BCF	3,*+7
0000090			LBL#63	EQU	*
00090	B39D 0100	0027		TB	39(R1),256
00092	DB0C	0096		BCF	3,*+4
0000093			LBL#66	EQU	*
0000093			LBL#65	EQU	*
0000093			ST#303	EQU	*
0000093			ST#304	EQU	*
00093	B29D 0100	0027		SB	39(R1),256
0000095			ST#305	EQU	*
0000095			ST#306	EQU	*
00095	DF08	0098		BCF	7,*+3
0000096			LBL#64	EQU	*
0000096			ST#307	EQU	*
00096	B69D FEFF	0027		NIST	39(R1),-257

Declaring Variables / Reserving Memory Space – 1

- Two ways of defining data
  - Define Storage Location (DS) allocates an area of memory for a certain type of data field without defining the contents of the location
    - Define Storage with 0 items can be used to add in label names (useful for branches and testing/dumps) or as delimiters
  - Define Constant (DC) allocates an area of memory, defines the data type and defines the initial contents
    - Note that variables allocated using Define Constant (DC) can be altered - the data defined is just an initial value

```
Table1   DS  10H   (Allocates 10 Halfwords of memory )
Data2    DS  F     (Allocates 1 Fullword of memory)
NOOP     DS  0H    (Allocates NO memory)

Table1   DC  10H'0'  (Allocates 10 Halfwords, each initialized to zero)
Data2    DC  F'8'    (Allocates 1 Fullword of memory, initialized to 8)
```

Declaring Variables / Reserving Memory Space – 2

- The available data types for DC and DS are
  - Character (C)
  - Hexadecimal Integer (X)
  - Binary Integer (B)
  - Fixed Point Decimal (base 10) Fullword (F)
  - Fixed Point Decimal (base 10) Halfword (H)
  - Floating Point, Short – 32 bits (E)
  - Floating Point, Long – 64 bits (D)
  - Fullword Address (A)
  - Halfword Address (Y)
  - Fullword Indirect Address, also called a ZCON (Z)
    - An Indirect Address is called a Pointer in many high-level computer languages

Declaring Variables / Reserving Memory Space – 3

- Equates can be used to declare constants
  - No memory is used to store the equate
  - The Assembler replaces all instances of an equate name with the value associated with it
  - For Example:

```
FIOFFIUA    EQU  10              This Equates a Value of 10 with the Variable Name
            LHI  R4,FIOFFIUA    Stores the value 10 in register 4
            LHI  R4,10          Does the same as the previous instruction
```

Declaring Variables / Reserving Memory Space – 4

- An ENTRY can be placed anywhere in a code module to denote an actual entry point (for a branch instruction), or to denote a variable which can be referenced outside of the module in which it is defined
- The EXTRN command is used to reference variables declared in another code module
  - The presence of the EXTRN for a variable lets the assembler know that the actual Halfword address will be resolved by the linkage editor & placed into the instruction object code
- This is one way to create a global variable

```
Module A  CSECT

          ENTRY AltEntry
AltEntry  DS 0H


          ENTRY Data1
Data1     DC H'45'
```

```
Module B  CSECT
          EXTRN AltEntry, Data1


          LH   R0,Data1


          B    AltEntry
```

This places the contents of Data1 variable in module A into Register 0; the object code after linkedit will be X'98F3yyyy' Where yyyy=the Halfword address of Data1;

Similarly, the address of AltEntry will be placed into the branch instruction at Linkedit time

Declaring Variables / Reserving Memory Space – 5.

- HAL COMPOOLs use EQUATE EXTERNAL commands to enable Assembler modules to reference HAL data
  - Same as creating an ENTRY point in Assembler code
  - Variable name used in the EQUATE EXTERNAL is not necessarily the same as the HAL name
    - Assembler variable names cannot exceed 8 characters in length, whereas HAL/S variable names can be longer

Reading the POO – 1

- Section 12 of the POO lists 8 categories of assembler operations
  - Six types of CPU commands
    - Fixed Point, Branch, Shift, Logical, Floating Point, Special (O/S)
  - Internal Control Operations (ICR)
    - Reading Program Counters (timers), Writing Discretes
  - I/O Operations/Program Controlled Inputs/Outputs (PCI/PCO)
    - Used for CPU Control of and Access to Information in the IOP
    - Configuring I/O Transmitters & Receivers, Setting Internal Control Information for the Master Sequence Controller (MSC) & Specific Bus Control Element (BCE) Processors, Reading Discretes, ...
- Each category corresponds to a different section of the POO
  - Fixed Point: section 4, Branch Operations: section 5, etc.
  - Each instruction has its own page(s) of detailed information
- The Format deals with how the instruction plus its operands look after being changed into machine code

Reading the POO– 2

- Sample Instruction list (from Section 12):

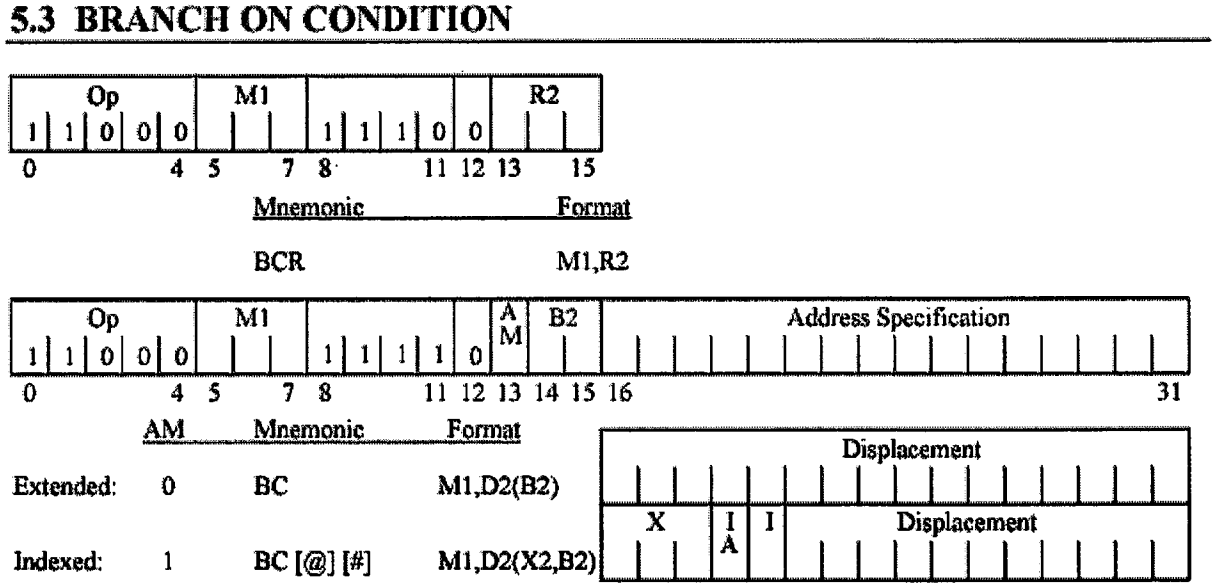
Name	Mnemonics	Format
Branch Operations		
Branch and Link	BALR, BAL	RR, RS
Branch and Index	BIX	RS
Branch on Condition	BCR, BC	RR, RS
Branch on Condition Backward	BCB	SRS
Branch on Condition (Extended)	BCRE	RR
Branch on Condition Forward	BCF	SRS
Branch on Count	BCTR, BCT	RR, RS
Branch on Count Backward	BCTB	SRS
Branch on Overflow and Carry	BVCR, BVC	RR, RS
Branch on Overflow and Carry Forward	BVCF	SRS

Reading the POO– 3

- More on instruction Format:
  - Determines the length of the command
    - 16 bits: RR, SRS
    - 32 bits: RS, RI, SI
  - Determines the source of the two data inputs
    - RR: Register and Register
    - SRS: Short version of Register and Storage
    - RS: Register and Storage
    - RI: Register and Immediate data
    - SI: Storage and Immediate data
  - Determines the method of main storage addressing
    - RR: None
    - SRS: Base Register contents + a displacement
    - RS: Expanded or Indexed (allows more bits in displacement)
    - RI: None
    - SI: Base Register contents + a displacement

Reading the POO– 4

- Example of detailed instruction information:



- Each Instruction description begins with the format(s) of the machine code version(s) of the instruction
  - Branch On Condition can have three different formats, based on the number of operands (2, 3, or 4) included in the instruction

Reading the POO– 5.

DESCRIPTION:

This instruction tests the PSW condition code status bits. Instruction bits 5 through 7 (the M1 field) specify which condition code (bit 16 and 17 of the PSW) is to be tested. Instruction bit 5 tests for a code equal 00, instruction bit 6 tests for a code equal 11, and instruction bit 7 tests for a code equal 01. Whenever the condition code test is successful, the branch is taken. Thus, when more than one bit of the M1 field is a one, the branch is taken for any successful test (e.g., M1 = 111 always branches, M1 = 000 never branches).

The branch address is contained in bits 0 through 15 of general register R2 for the RR format. This 16-bit branch address is expanded to a 19-bit branch address. (See Expanded Addressing.)

RESULTING CONDITION CODE:

The condition code was set following all arithmetic, logical, test and compare instructions, and otherwise remains unchanged unless the program status word is altered. The code is not changed by this instruction.

INDICATORS:

The overflow and carry indicators are not changed by this instruction.

AP101S Addressing – 1

- **AP101S RAM is divided into 16 sectors, each 32,768 Halfwords (0x8000)**

Sector	Address
0	0 - 7FFF
1	8000 - FFFF
2	10000 - 17FFF
3	18000 - 1FFFF
4	20000 - 27FFF
5	28000 - 2FFFF
6	30000 - 37FFF
7	38000 - 3FFFF
8	40000 - 47FFF
9	48000 - 4FFFF
10	50000 - 57FFF
11	58000 - 5FFFF
12	60000 - 67FFF
13	68000 - 6FFFF
14	70000 - 77FFF
15	7F000 - 7FFFF

- **At a given time, data access can be made**
  - **Via normal Halfword addressing to sector 0 & one other sector defined by the PSW DSR**
  - **Via Fullword indirect address pointer (ZCON) addressing to any sector (requires RS instruction format)**
  - **Via use of Data Sector Extension (DSE) register**

NOTE: To Form Real Address from Halfword Address:

(Use Address X'9561' , DSR = 5 as Example)

1. **Convert High Order HEX Digit to Binary (B'1001')**
2. **Replace Bit 0 with DSR (B'0101001')**
3. **Convert Back to HEX (X'29')**
4. **Append to Original Last 3 HEX Digits (X'29561')**

When Using ZCON/DSE or Doing Branch Type Address (Using BSR), Same Conversion Method Applies

AP101S Addressing – 2

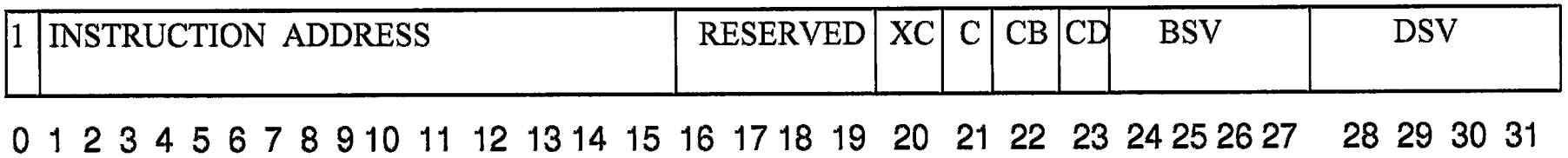
- Data Sector Extension (DSE) Register
  - Each general purpose register has a DSE associated with it
    - Each DSE register is 4 bits in size
  - The DSE is used in a non-branch instruction using a base register which contains a 0 in the 1st bit of the Halfword address
  - The address accessed is in the sector of RAM indicated by the DSE
  - Unique instructions exist for reading/changing values in the DSEs
  - MVH, SCAL, SRET instructions are not part of base register addressing, but they also use the DSE
  - Some instructions change BOTH the base register and the corresponding DSE
    - Check the P.O.O. for more details

AP101S Addressing – 3

- Instruction modifiers:
  - @ : Indirect Addressing
    - The preliminary address in the instruction points to a memory location that contains the actual address to be used in the instruction (Fullword Indirect Address Pointer (ZCON))
  - # : Index Modification
    - Forces the assembler to use an indexed mode of calculation of effective addresses
  - $: Long Format Modifier
    - Forces the assembler to use the long instruction format for the operation
      - If the long format instruction is not forced, the assembler automatically generates the shortest format that it can
      - Instruction type/length determines the formula to be used in calculating the effective address of the operand

AP101S Addressing – 4

- **Fullword Indirect Address Pointer (ZCON)**



XC : Index Control Bit (If 0 Indexing Will Be Done)
C: Control Bit for PSW Modification
CB: Control Bit for Changing PSW's BSR to the BSV
CD: Control Bit for Changing PSW's DSR to the DSV
BSV/DSV: Depending on Type of Instruction & Other Control Bit Settings, Sector to Be Used in Instruction and/or to Replace PSW BSR/DSR

AP101S Addressing – 5

- **Examples:**
  - **ZCON Usage (Non-Branch Instruction)**

```
        LA      R2,AZCON    This Places the Address of AZCON into Register 2
        LHI     R5,2        This Sets Register 5 = 2  (actually X'0002 0000')
        STH@#   R4,0(R5,R2) Preliminary Effective Address = Base Register + Displacement
                                                          (Addr of AZCON + 0)
                            Effective Address = Contents at PEA (i.e., Loc 8023 in Sector 4) +
                                                Index Register 5 Contents (2) (i.e., 20025)
                            Results: Register 4 Contents Stored at Location 20025
AZCON   DC      X'80230004'
```

  - **ZCON Usage (Branch Instruction)**

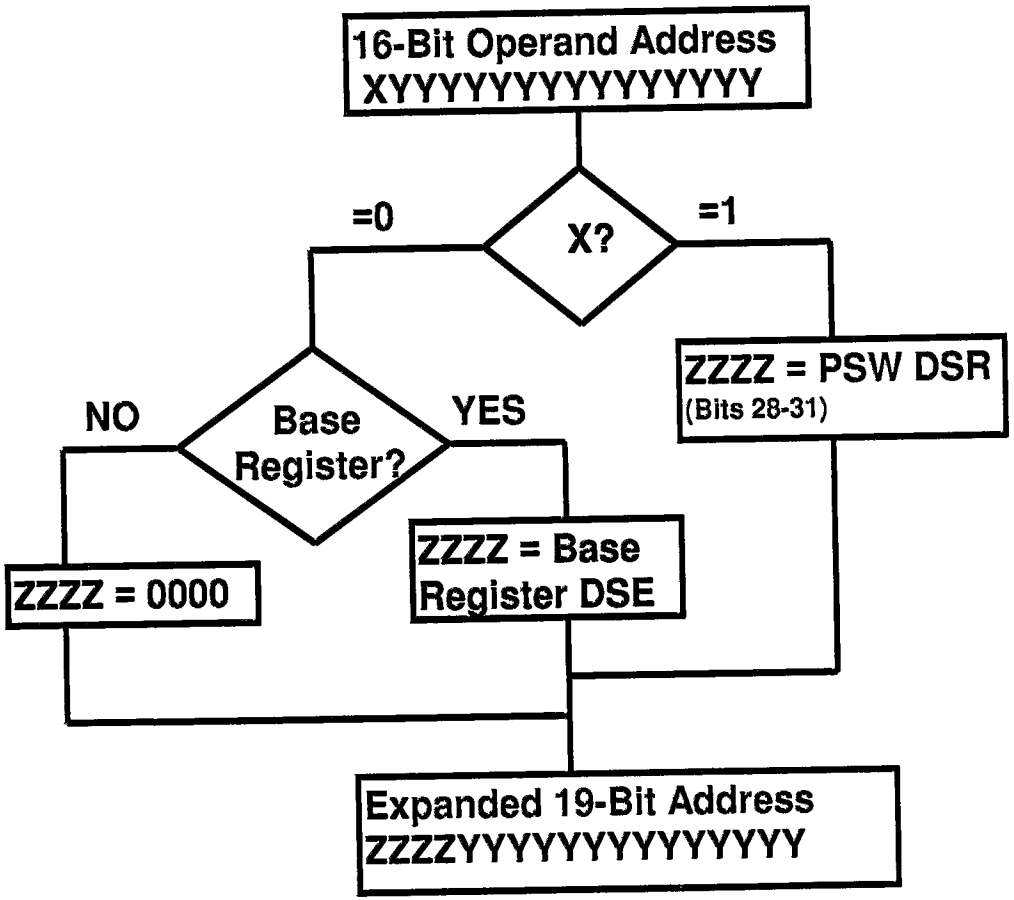
```
        LA      R2,AZCON    This Places the Address of AZCON into Register 2
        BAL@#   R4,0(R5,R2) Preliminary Effective Address = Base Register + Displacement
                                                          (Addr of AZCON + 0)
                            Effective Address = Contents at PEA (i.e., Loc 8023)
                            Results: PSW BSR Will Be Changed to 3, PSW DSR Will Be Changed to 5,
                            Branch to 8023 in Sector  3 (18023), & Return Address Placed Into Register 4
AZCON   DC      X'80230F35'

(Note that Index Is Specified but Not Used - the XC Control Bit in ZCON = 1: NO POSTINDEXING)
```

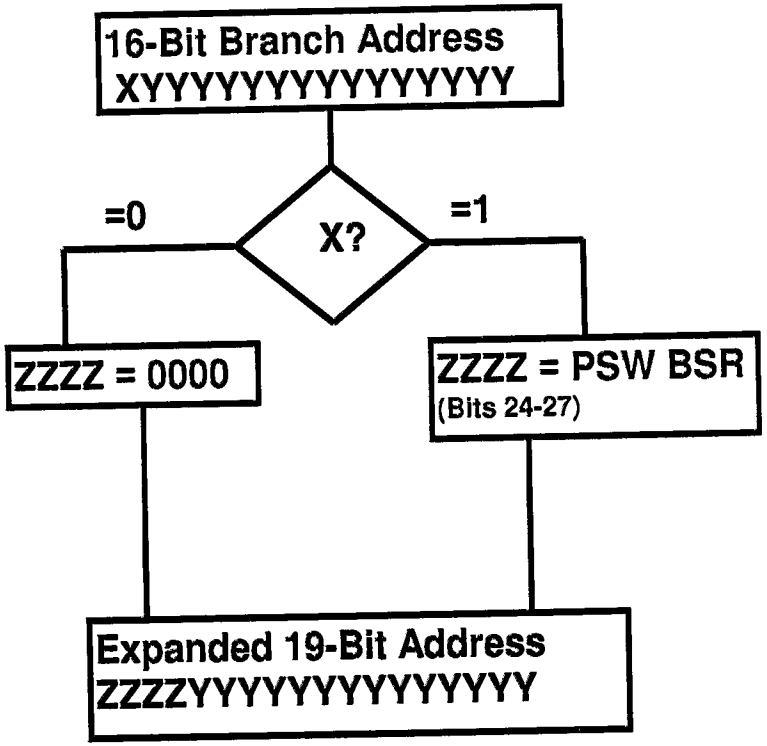
AP101S Addressing – 6.

From the POO:

Data Reference Instructions



Branch Instructions



Base Registers and the USING Statement – 1

- Affects where data is loaded from or stored into memory
  - The final address used is the sum of the contents of the base register and the value provided in the actual instruction
- All instructions located between a USING command and either a DROP command or another USING command make use of the base register
- The Base Register (General Purpose Register 0, 1, 2, or 3) must be set up prior to the data reference, by loading an address into the register
  - Restrictions on Register 3
    - Register 3 can only be used as a base register with the SRS instruction format
    - If Register 3 is coded as the base register for non SRS instructions, no base register is used
  - Registers 0, 1, and 2 do not have this restriction

Base Registers and the USING Statement – 2

- **USING statements are processed sequentially - NOT in execution order**

```
Example:
     LA       R3,Loc3          Load address of Loc3 in R3
     USING    *,R3             R3 to be used as a base register
     B        A1               Branch to instruction labeled A1
     DROP     R3               Stop using R3 as a base register
     MIH      R2,7             A Multiply instruction
     USING    Loc2,R2          R2 to be used as a base register
A1   DS       0H               Just a place holder; not an instruction
     LH       R5,Loc1          Load an address into R5
     DROP     R2               Stop using R2 as a base register
```

The value loaded into register 5 by the LH command is at address (Loc1 + Loc2)

- **The compiler must be able to determine address data at compile time, thus the most recent using statement in the code applies, even if it looks like that instruction was branched around during execution.**

Compares, Condition Codes, & Branching-1

- Various Assembler Instructions modify the value of the two PSW Condition Code (CC) bits based on the results of the operation
  - Add, Subtract, Test Bits, Compare, etc.
- These instructions can be followed by a Branch, or other instruction, which reads the CC bits and takes an action based on their value
  - The M1 bits in the instruction state how the CC bits are interpreted
  - The M1 is listed in decimal in the assembly code
  - For Example:
    - CR R0,R1 Compare Values in Registers 0 and 1
    - BCF 2,6 Branch on Condition Forward, M1 = 2, Displacement = 6

M1	Branch when		M1	Branch when
0	Never		4	R0 = R1
1	R0 > R1		5	R0 ≥ R1
2	R0 < R1		6	R0 ≤ R1
3	R0 ≠ R1		7	Always

Compares, Condition Codes, & Branching-2

	M1 FIELD (TEST)		
	(5)	(6)	(7)
ARITHMETIC & TALLY			
ZERO	1	0	0
NEGATIVE	0	1	0
POSTIVE (>0)	0	0	1
LOGICAL			
ZERO	1	0	0
NOT ZERO	0	1	0
TEST			
ZERO	1	0	0
MIXED	0	1	0
ALL ONES	0	0	1
COMPARE			
EQUAL	1	0	0
D1 < D2	0	1	0
D1 > D2	0	0	1

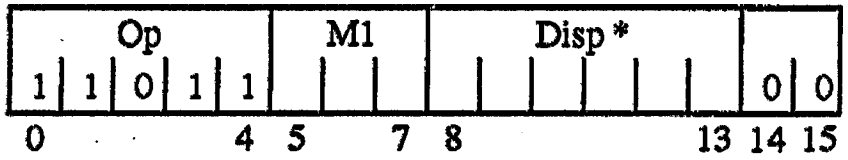
In the POO, the M1 values are often listed in binary

M1 FIELD = B'111' (7 decimal) means ALWAYS BRANCH.

(Check P.O.O. for other M1 Field use, e.g. ISPB Instruction.)

INSTRUCTION BITS 5 THROUGH 7 (THE M1 FIELD) SPECIFY WHICH CONDITION CODE (BITS 16 AND 17 OF THE PSW) IS TO BE TESTED. INSTRUCTION BIT 5 TESTS FOR A CODE EQUAL TO 00, INSTRUCTION BIT 6 TESTS FOR A CODE EQUAL TO 11, AND INSTRUCTION BIT 7 TESTS FOR A CODE EQUAL TO 01.

5.6 BRANCH ON CONDITION FORWARD



* Displacements of the form 111XXX are not valid.

Mnemonic	Format
BCF	M1,D2

DESCRIPTION:

This instruction tests the PSW condition code status bits. Instruction bits 5 through 7 (the M1 field) specify which condition code (bits 16 and 17 of the PSW) is to be tested. Instruction bit 5 tests for a code equal 00, instruction bit 6 tests for a code equal 11, and instruction bit 7 tests for a code equal 01. Whenever the condition code test is successful, the branch is taken by adding the Disp to the updated IC. Thus, when more than one bit of the M1 field is a one, the branch is taken for any successful test (e.g., M1 = 111 always branches).

NOTE: PSW ADDR is pointing to the next location after the BCF when Disp. is calculated.

Compares, Condition Codes, & Branching-4

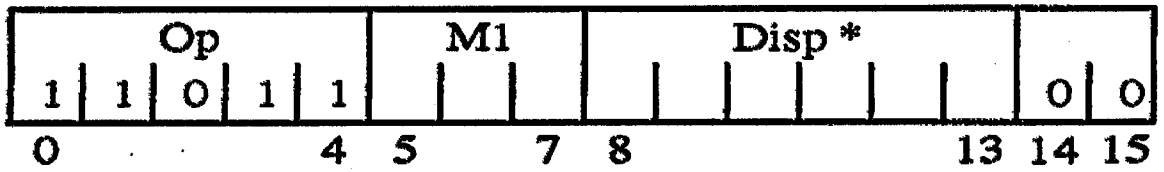
- What will be the machine code generated for the following assembler command?

BCF 2,6 Branch Conditional Forward

- Answer:

DA14

From the POO:



Mnemonic	Format
BCF	M1,D2

Compares, Condition Codes, & Branching-5.

- The M1 is the first operand supplied (which is 2)
- The second operand supplied is the offset distance to branch
- The intent of the instruction "BCF 2,6" is to branch, as needed, to an address which is equal to
  - The address of "BCF 2,6" + 6 Halfwords
- When the command "BCF 2,6" executes, the PSW changes its next instruction address to point to the instruction after "BCF 2,6"
  - The displacement in the machine code needs to be equal to the requested displacement minus the size of the instruction needed to accomplish the branch
  - BCF is a Halfword in size
  - So, the final Disp. is 6 – 1 = 5

Common Instruction Gotchas – 1

- Some instructions assume Fullword boundaries
  - A Fullword related command doubles the address index (to convert the count of Fullwords to Halfwords)
- Some instructions assume Doubleword (64 bit) boundaries
  - A Doubleword related command multiplies the address index by four (to convert the count of Doublewords to Halfwords)
  - So an address of 0X0123 + an index of 2 would be
    - 0x0125 for a HW related instruction
    - 0x0127 for a FW related instruction
    - 0x012B for a DW related instruction

Common Instruction Gotchas – 2

Shifting Logical vs. Shifting Arithmetical

- Logical Shift (SLL, SLDL, SRL, SRDL)
  - Bits that are shifted beyond the length of the register are lost; zeros fill in vacancies that are introduced at the other end
- Arithmetic Shift Right (SRA, SRD)
  - Bits that are shifted beyond the length of the register are lost; fills empty spaces with a value equal to the original sign bit
- Shift and Rotate (SRR, SRDR)
  - No data is lost; bits rotated outside the right end of the register are moved into the other (left) end

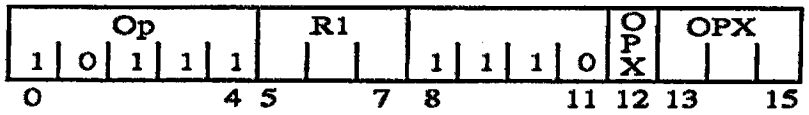
- *The only available Shift Left instruction is the Logical version*

Common Instruction Gotchas – 3

- Multiply (M)
  - If multiplying integers, and the MIH instruction is not used:
    - Result must be shifted *right arithmetic* one position for proper scaling
- Divide (D)
  - If dividing integers:
    - Result must be shifted *left logical* one position for proper scaling
- Remember, with binary numbers:
  - Shifting a number to the right by one is the same as dividing by two
  - Shifting a number to the left by one is the same as multiplying by two

Common Instruction Gotchas – 4

4.17 LOAD FIXED IMMEDIATE



Mnemonic	Format
LFXI	R1,Value

DESCRIPTION:

A fixed point literal value is loaded into the general register specified by R1.

The immediate values are -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 or 13. The immediate is loaded into bits 0 through 15 of general register R1. Bits 16 through 31 of general register R1 are set to zero.

OPX (Bits 12, 13, 14, & 15) (hex)	Immediate Value --> R1 (hex)
0	FFFE0000
1	FFFF0000
2	00000000
3	00010000
4	00020000
5	00030000
6	00040000
7	00050000
8	00060000
9	00070000
A	00080000
B	00090000
C	000A0000
D	000B0000
E	000C0000
F	000D0000

The value that is in the assembly command.

The value that appears in the machine code instruction (in bits 12-15).

The value that is loaded into the register.

Common Instruction Gotchas – 5

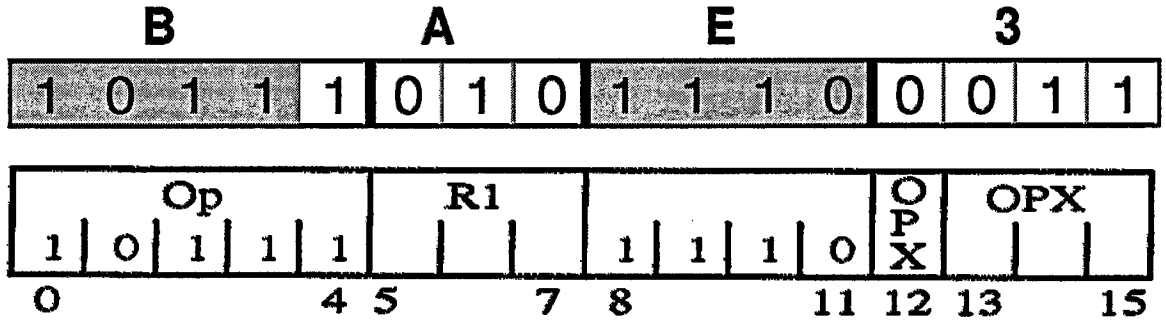
Example:

LFXI R2, 1

(Load into register 2, the value "1")

Assembles to:

BAE3



CONSIDER USING LHI -
It uses an extra 16 bits of RAM, but is worth that cost to avoid the confusion of LFXI.

Common Instruction Gotchas – 6.

- What does the following assembly instruction do:

SVC 40,(R1)

  - It executes an SVC (supervisor call), but it might not be SVC #40
    - An SVC is a request, made by an application, for the operating system (FCOS) to perform a specific service
  - This instruction tells the CPU to look at the contents of main memory, 40 locations past the value stored in register R1
    - R1 normally contains a pointer for the starting point of the #D info for the current process
    - The memory location being specified contains the ID of the SVC to be executed
  - The list of valid SVCs is located in Appendix F of the FCOS User's Guide

FCOS Macros – 1

- Allows Developers to produce assembly code that looks more structured
  - The Assembler converts the macros into regular assembly code
  - Cannot be used for patching FSW
- Contains a variety of useful command structures
  - IF THEN ELSE
  - DO WHILE and DO UNTIL loops
  - CASE statements
  - PROCedure calls

FCOS Macros – 2.

MACRO Mnemonics (abbreviations)

When Used	Mnemonic	Meaning	Complement
after compares	GT, H	greater than, high	LE, NH
	LT, L	less than, low	GE, NL
	EQ	equal	NE
after arithmetic	P	plus (positive)	NP
	M	minus (negative)	NM
	Z	zero	NZ
After logic	Z	zero	NZ
After test	O	ones	NO
	M	mixed	NM
	Z	zeros	NZ

NOTE: An "O" means one and not zero!

Zero is neither positive nor negative!

Exercises

Is this an atomic operation? – 1

```
x = x * y;
```

- Atomic: (adj.) Cannot be split into unique parts; indivisible; undividable

Is this an atomic operation? – 2.

Answer: No!

- A computer must perform a series of step to complete the task:

  x = x * y;

  1. Load the value of x into a register (R4, perhaps)
  2. Load the value of y into a different register (R6, perhaps)
  3. Multiply the contents of R4 by the value in R6
  4. Store the new value in R4 back into the memory address for x

- Remember, what may look like a single operation in a high level language is often composed of multiple steps
  - The Move Halfword (MVH) assembler instruction is composite in nature and can be interrupted when only partially complete
    - %COPY often is compiled to a MVH instruction
  - Several other CPU instructions can be interrupted by the IOP performing DMA (see the P.O.O. for details)
  - Only TS & TSB (Test and Set (Bits)) are guaranteed to be atomic

Estimating Execution times – 1

- How long might it take the CPU to calculate:

  x = x * y;

Estimating Execution times – 2

- **It depends...**
  - **The following data is taken from the POO:** Chapter 17: AP-101S Instruction Execution Times

L: Load M: Multiply S: Store E: Floating Point D: Double H: Halfword

All execution times are estimates, in units of microseconds, and assume the RS format

Instruction	Time
L	0.25
LE	1.20
LED	1.50
LH	0.25
S	0.25
SE	2.50
SED	6.50
SH	0.25

Instruction	Inputs	Output	Time
M	Fullword Integers	Doubleword Integer	2.40
M	Fullword Integers	Fullword Integer	2.15
ME	Fullword Floats	Doubleword Float	6.25
ME	Fullword Floats	Fullword Float	5.75
MED	Doubleword Floats	Doubleword Float	19.00
MH	Halfword Integers	Fullword Integer	1.35
MIH	Halfword Integers	Halfword Integer	1.70

Estimating Execution times – 3

- **Best case with Halfword integers:**

```
  LH    0.250   (load x)
  LH    0.250   (load y)
  MIH   1.750   (multiply)*
+ SH    0.250   (store x)
-------------------------
        2.500 microseconds
```

* **NOTE:** MIH takes longer than MH, but does not need to be followed by a shift instruction.

At 40 clock ticks per microsecond, this task involves at most 100 CPU microcode steps.

- **Worst case with Doubleword floating points:**

```
  LED    1.50   (load x)
  LED    1.50   (load y)
  MED   19.00   (multiply)
+ SED    6.50   (store x)
-------------------------
        28.50 microseconds
```

At 40 clock ticks per microsecond, this task involves at most 1,140 CPU microcode steps.

Estimating Execution times – 4

- An easier method – looking at compiled code

```
1  HAL/S    FC-30.0                    OI301700.APPL.SRC(GCQORB)RVL=CN
0  LOC      CODE          EFFAD LABEL    INSN    OPERANDS          SYMBOLIC OPERAND
00000014                          ST#62    EQU     *
 0000014                          ST#63    EQU     *
 00014 9A51                0014           LH      R2,20(R1)         TIME: 0.25;
 00015 B606 BFFF           0001           NIST    1(R2),-16385      TIME: 3.0
 0000017                          ST#64    EQU     *
 00017 B206 0100           0001           SB      1(R2),256         TIME: 3.0
```

- An estimated execution time (in microseconds) is provided for each assembly instruction in the third copy of any HAL/S code module in a compile output
  - The first copy of the code shows the deltas
  - The second copy of the code is the formatted version
  - The third copy of the code is the assembly version

NIST = aNd Immediate to STorage SB = Set Bits

Estimating Execution times – 5

- Another method of calculating execution time

The HAL/S-FC Compiler System Specification, in section 5, has a table with mathematical HAL/S commands. Many of these are listed with their execution time in microseconds.

Estimating Execution times – 6.

Duty Cycle considerations

- A PASS Major Cycle lasts 960,000 microseconds
  - The CPU duty cycle increases by 1% for every 9,600 microseconds of new code added
  - For a process that runs at 25 HZ, the CPU duty cycle increases by 1% for every 400 microseconds of code added
    - 9,600 / 24 = 400
      - A 25 HZ function executes 24 times per major cycle
  - If the floating point multiply in this example were to be added to a 25 HZ process, it would increase the CPU duty cycle by 0.07125%
- Real time systems like PASS tend to fail (or at least become unstable) well before their duty cycle reaches 100%

For Loop – 1

```
DO FOR TEMPORARY I = 1  TO  CZ2V_NBR_GPCS;
   CZ2V_G3ARCH_MM_AREA$(I;) = 0;
END;
```

For Loop – 1b

- **The FSW compiler produces a formatted version of HAL/S code with statement numbers added for reference.**

```
ST#782        DO FOR TEMPORARY I = 1  TO  CZ2V_NBR_GPCS;

ST#783           CZ2V_G3ARCH_MM_AREA I = 0;

ST#784        END;
```

For Loop – 2

```
00000C5                              ST#782   EQU     *
000C5 EEF3 0001                               LHI     R6,1
000C7 DF18           00CE                     BCF     7,*+7
00000C8                              LBL#94   EQU     *
00000C8                              ST#783   EQU     *
000C8 1FE6                                    LR      R7,R6
000C9 9A05           0001                     LH      R2,1(R1)
000CA A1F6 E027      0027                     ZH      39(R7,R2)
00000CC                              ST#784   EQU     *
000CC B0E6 0001                               AHI     R6,1
00000CE                              LBL#93   EQU     *
000CE B5E6 0005                               CHI     R6,5
000D0 DE26           00C8                     BCB     6,*-8
00000D1                              LBL#95   EQU     *
```

For Loop – 3

EQU: Equate - Used to mark a memory location. Only appears in code listings; does not use any computer memory.

LHI: Load Halfword Immediate - Copy 16 bits from the current instruction into a CPU register.

BCF: Branch Conditional Forward - Check the Condition Code bits and increase the value in the next instruction counter if needed.

LR: Load Register - Copy a value from one register to another.

LH: Load Halfword - Load a value from memory to a register.

ZH: Zero Halfword - Store a zero in a memory location.

AHI: Add Halfword Immediate - Add a value included in the instruction to the contents of a register.

CHI: Compare Halfword Immediate - Compare register contents to a value included in the instruction.

BCB: Branch Conditional Backwards - Check the Condition Code bits and reduce the value of the next instruction counter if needed.

For Loop – 4

```
000C5 EEF3 0001          LHI    R6,1
000C7 DF18       00CE    BCF    7,*+7
000C8 1FE6               LR     R7,R6
000C9 9A05       0001    LH     R2,1(R1)
000CA A1F6 E027  0027    ZH     39(R7,R2)
000CC B0E6 0001          AHI    R6,1
000CE B5E6 0005          CHI    R6,5
000D0 DE26       00C8    BCB    6,*-8
```

(1) Initialize "temporary I" to 1

(2) (Branch)

For Loop – 5

```
000C5 EEF3 0001          LHI    R6,1
000C7 DF18       00CE    BCF    7,*+7
000C8 1FE6               LR     R7,R6
000C9 9A05       0001    LH     R2,1(R1)
000CA A1F6 E027  0027    ZH     39(R7,R2)
000CC B0E6 0001          AHI    R6,1
000CE B5E6 0005          CHI    R6,5
000D0 DE26       00C8    BCB    6,*-8
```

(3) Compare I to 5

(4) If I < 5, Branch

Note: The machine code version of this branch has a displacement of 9. When branching backwards, the machine code value is one larger than the listed assembler version.

For Loop – 6

```
000C5 EEF3 0001          LHI    R6,1          (5) Perform contents
000C7 DF18       00CE    BCF    7,*+7             of loop

000C8 1FE6               LR     R7,R6  <-- Load array index
000C9 9A05       0001    LH     R2,1(R1) <-- Load data CSECT
                                               address
000CA A1F6 E027  0027    ZH     39(R7,R2) <-- Store zero in the
                                               variable contents

000CC B0E6 0001          AHI    R6,1
000CE B5E6 0005          CHI    R6,5
000D0 DE26       00C8    BCB    6,*-8
```

For Loop – 7

```
000C5 EEF3 0001          LHI    R6,1
000C7 DF18       00CE    BCF    7,*+7
000C8 1FE6               LR     R7,R6
000C9 9A05       0001    LH     R2,1(R1)
000CA A1F6 E027  0027    ZH     39(R7,R2)
000CC B0E6 0001          AHI    R6,1   <-- (6) Increment I
000CE B5E6 0005          CHI    R6,5   <-- (7) Check if I <= 5
000D0 DE26       00C8    BCB    6,*-8      (8) If so, Branch
```

For Loop – 8.

```
000C5 EEF3 0001          LHI    R6,1
000C7 DF18       00CE    BCF    7,*+7
000C8 1FE6               LR     R7,R6
000C9 9A05       0001    LH     R2,1(R1)
000CA A1F6 E027  0027    ZH     39(R7,R2)
000CC B0E6 0001          AHI    R6,1
000CE B5E6 0005          CHI    R6,5
000D0 DE26       00C8    BCB    6,*-8
```

(9) Repeat loop contents a total of 5 times

(10) Increment I to 6

(11) Exit loop by continuing to the next instruction and not branching

Execution continues with the next instruction...