CS360 Notes

* A register holds a 32-bit value, SR 2,2 -> subtract register (is 0 as subtracting itself)
* Not uncommon to put address in register, in ASSIST, we normally use 24-bit addresses, ignore first byte
* When your program starts, register 15 = address of the program

D (B) address, B = base register 0-15, D = displacement 0-4095

* The value is the value in register B + displacement
* 24 (7) value in register 7 + 24(base 10) {18 base 16}
* Exception: if B=0, the value is just D

D (X, B) address, X = index register, B = base register, D = displacement

* 28 (2, 9) value = value in register 2 + value in register 9 + displacement

What does a program look like?

* JCL (job control lang) //
* …
* Program code MAIN CSECT
* … …
* … END MAIN
* JCL (couple lines) //, /\*

Comments

* \* in column 1 : the rest of the line is a comment or skip at least one space after any instruction and the rest of the line will be comment
* Do not leave a blank line in the code
* Do not use the tab key in code

Columns

* 1 – labels, 10 – instruction, 16 – arguments, 72 is rarely used, 73-80 not used (sequence numbers on physical punch cards)

From lab training

* Main CSECT (control section)
  + Using main, 15 (using register 15 as our base register)
  + SR 2,2 (subtract register, our first action, set R1 = R1-R2)
  + XDUMP (tool provided by ASSIST, for debugging, pseudo instruction, part of ASSIST program)
  + BR 14 (what to do after program ends)
  + END MAIN (delimiter for program)

Terminology

* Fullword - 4bytes on an address = a multiple of 4 (fullword boundary)
* Halfword – 2bytes on an address = a multiple of 2 (halfword boundary)
* Doubleword – 8bytes on an address = a multiple of 8 (doubleword boundary)

Variables

* numA DS F (not initialized, but ASSIST initializes bytes to ‘F5’ and register to ‘F4F4F4F4F4’)
* (define storage) (fullword)
* numb DC F’137’ (is initialized 137 base 10 as 32 bit signed number)

Commands

* AR R1,R2 (add register, set R1= R1+R2)
* L R,D(X,B) (load, copy a value from a fullword into a register)
* ST R,D(X,B) (store, copy a register’s value into a fullword)
* LR R1,R2 (set R1 = R2)
* LA R,D(X,B) (calculate D(X,B) and store it in R) this gets the address, L gets the value

Whats actually happening

* Assemble(encode) the instruction
* A,R 7,3 🡪 1 A 7 3 (operation code, 2 bytes or 4 hex digits)
* Format RR, operation code is 1 byte and 2 registers

More on XDUMP

* XDUMP by itself prints the values of all 16 register
* XDUMP address,length prints the values of bytes starting at the address

Condition Code

* “special register”, 2 bits long – values 0,1,2,3
* Add or subtract, set condition code, 0 result is 0, 1 - < 0, 2 - > 0, 3 not used
* CR R1,R2 – compare, set condition code, 0 equal, 1 left < right, 2 left > right

Branch on Condition

* BC mask, D(X,B) mask is a 4 bit binary value, values are 0-15 usually written in binary
* If condition code = N, and bit N of the mask is 1, We jump to the address D(X,B)
* BC B’1000’, 120(15) – if condition code = 0, we continue at 120(15)
* If mask is B’0000’, nothing happens. If mask is B’1111’ it is an unconditional branch
* BR 14 is shorthand for BCR B’1111’,14

LABEL DS 0H – allocates 0 halfwords but it will associate LABEL with an address

Loop something to set CC

BC B’….’, ENDLOOP

.

.

BC B’1111’, LOOP

ENDLOOP DS 0H

I/O

* XREAD address,length – Assist extra, reads bytes from “standard input”(a file) and stores them at the address, length can be 1-80 bytes
* Sets the condition code, 0 success(usual), 1 end of file, doesn’t use 2 or 3
* BUFFER DS 80C – sets 80 bytes of character storage
  + XREAD BUFFER, 80 – fills buffer with read data.
  + Or CL80, this is a variable 80 bytes long, C80 is a table of 80-character bytes
* XPRNT address, length – Assist extra, prints bytes take from the address, length can be between 1-133 bytes long
* First byte in the line is the carriage control character, CCC = ‘ ‘ single space, ‘0’ double-spacing, ‘-‘ triples pacing, ‘1’ new page
* Chars are encoded with the EBCDIC sequence, page-37: X’C1’ ‘A’, X’40’ space, X’F1’ 1, X’f9’ 9
  + Line DC C’0’ double spaced
  + DC CL5’HELLO’
  + XPRINT LINE, 6
  + HELLO
* XDECO R, address – convert the 32 bit value in R to a printable base 10 number in 12 bytes at the address
* Reg 7 = X’0000003F’, XDECO 7,B12 🡺 \_\_\_\_\_\_\_\_\_\_63 (10 spaces)
* Minus sign if negative, suppresses leading 0s, numbers range from -2bil to 2 bil, so 10 digits, rounds 11 spaces needed to 12

LOOP XREAD Buffer,80

BC B’0100’,ENDLOOP

.

.

BC B’1111’,LOOP

ENDLOOP \_\_\_\_\_\_\_\_\_\_\_

* XDECI R,address – puts a line of data into a register. What it does: starts at the address, skips leading spaces, looks for a digit + or -,
  + If anything else is found – reg 1: address of bad char, R is unchanged, condition code 3.
  + Else (start reading digits, if more than 9 digits found then reg 1 = address of first non digit char after digits, R is unchanged, condition code = 3.
  + Else (convert to a binary value, store in R, register 1 = address of first non dig char after digits, condition code = 0 – zero, 1 – negative, 2 – positive

Adding a list of numbers

* SR 3,3 sum=0
* LOOP XREAD BUFFER,80 read a line
* BC B’0100’, LOOPEND ++ eof, quit
* XDECI 4, BUFFER get a number
* AR 3,4 add to sum
* BC B’1111’, LOOP
* LOOPEND XDECO 3,LSUM
* XPRINT LINE, 18
* BUFFER DS 80C
* LINE DC C’ ‘ single space
* DC CL5’SUM =”
* LSUM DS 12C
* LINEEND DS 0,C
* Length = LINEEND – LINE

Literals

* MAIN CSECT
* .
* .
* BR 14
* LTORG
* Variables
* Different ways to use (L 9,=F’18’)(A 5,=F’1’)(XPRINT =CL6’ Hello’, 6)
* LTORG: list literals here, usually right after BR 14,

Slack Byte

* 0000028 NAME DC CL3’TIM’ (uses only 3 bytes)
* 0000002B is a slack byte
* 000002C TOTAL DS F (fullwords only start on multiples of 4)

Storage

* NAME DC CL6’TIM’ 🡺 TIM\_ \_ \_, Left justified padded with spaces. X’E3C9D4404040’
* NAME DC CL3’MATTHEY🡺 MAT, truncated on the right

More on type RR instructions

* LPR R1, R2 – load positive register, set R1 = abs val of R2, set condition code = 0 zero, 1 neg, 2 pos, 3 overflow (rare, only -2^31 exists while 2^31 does not)
* LCR R1, R2 – set R1=(-1)\*R2, set condition code = 0 zero, 1 neg, 2 pos, 3 overflow
* LNR R1, R2 – set R1=(-1)\*abs val of R2, set condition code = 0 zero, 1 neg, 2 pos
* LTR R1, R2 - Load and test set R1=R2 and set condition code = 0 zero, 1 neg, 2 pos
  + LTR 4,4 useful for checking if = 0 or some other stuff

Abbreviations for branches – page 19

* Extended mnemonic instructions,
* BC B’1000’,PLACE (checks for 0) is the same as BZ PLACE
* BC B’1111’,PLACE (unconditional branch) B PLACE
* BE – BC B’100’, PLACE
* BNE – BC B’0111’, PLACE
* BNL – BC B’1011’ – branch not less,
* BL – BC B’0100’ – branch is less
* BR 14 – BCR B’1111’,14

Multiplying

* 32 bit num \* 32 bit num = 64 bit num, we have a 64 bit number in 2 registers – even and odd
* We identify the pair of registers by naming the even one
* M R1, R2 R1 must be even register, representing an even-odd pair. The product is in R1 and R1+1. We multiply the values in R1 + 1 and R2
* Ex: R5 = x’0000006’, R9 = x’000000005’; M 4,9 🡺 the result will be in registers 4 and 5
  + R4 = x’0000000000’, R5=x’000000001F’

Dividing

* 64 bit num/32 bit num, quotient 32 bits, remainder 32 bits
* DR 4,11 – divide the 64 bit number in registers 4 and 5 by the number in register 11. Remainder in reg 4 and quotient in reg 5
* Ex. Reg 8 = x’00000000’, Reg 9 = x’0000001F’, reg 2 = x’00000007’, 31/7, quotient = 4, remainder = 3. DR 8,2, Result: Reg 8 = x’00000003’, Reg 9 = x’00000004’
* -13/5, Reg 7 = x’FFFFFFF3’, Reg 1 = x’00000005’, Reg 6 = x’FFFFFFFF’, Result - Reg6 = ‘FFFFFFFD’, Reg7 = ‘FFFFFFFE’
* Round up if >.5

ABEND

* Is an abnormal end, that means the program has done something wrong and is terminated

Program status word

* 64 bits, information about what is happening, reference summary p22, BC mode of PSW
* Interruption Code bits 16-31
* Instruction length code bit 32-33, number of halfwords in the current instruction
* Condition code bits 34-35
* Address of next instruction bits 40-63

Equates

* Name EQU value, SIZE EQU 24\*32, search and replace. Basically, runs through first pass of code and replaces, the ‘variable’ with the value
* RO EQU 0, R1 EQU 1, … - register equates

Tables

* TABLE DC 40F’-3’, we process tables with pointers
* EOT DS F end of table
* File: 17,42,-666,13…
* LA 2,TABLE. Points to the top of the table and the address of the first full word of the table
* SR 6,6 counter = 0
* --------------table example w/ counter-----------
* LOOP XREAD BUFFER,80
* BC B’0100’,LOOPEND
* C 6,=F’40’
* BE LOOPEND makes sure you don’t overfill table.
* XDECI 3,BUFFER
* ST 3,0(0,2) places Val in reg 3 in the address of 2, the top of the table
* A 2,=F’4’ advances address in reg 2 to next table cell
* A 6,=F’1’ increment counter
* B LOOP
* LOOPEND ST 2,EOT stores address of end of table
* -----printing a table-----
* LA 2,TABLE 2: table pointer
* LOOP C 2,EOT check are we done
* BE LOOPEND at end, quit
* L 3,0(0,2) get a number
* XDECO 3,LNUM format value
* XPRINT LINE,13
* A 2,=F’4’ advance pointer
* B LOOP repeat
* ENDLOOP DS 0H if not sure what to do, just a place holder

Subroutines - internal

* Code runs and then is sent somewhere else to run the sub routine, basically like a function
* BAL R,D(X,B) - Branch and link, used for calling subroutine. First it loads the second half of the PSW into R : the address of next instruction. Second, it goes to the address
  + BAL 10, SUB
  + SUB …… BR 10 to go back
* BALR R1,R2 – branch and link register, same thing but register instead of address
* If you want subroutine to do work on parameters you pass a parameter list PLIST.
* PLIST DC A(W) - PLIST DC A(W,X,Y,Z)
* DC A(X)
* DC A(Y)
* MAIN CSECT
* …
* LA 1, PLIST – loading parameter list into register 1 is the convention
  + LA 1,A(W,X,Y,Z) is also acceptble
* BAL 10,SUB

Saving

* LM R1,R2, D(B) - load multiple,copies fullword values starting at D(B) into registers R1 through R2
* LM 2,9,REGPILE and LM 14,7 REGPILE both work
* STM R1,R2,D(B) - store the values of R1 through R2 in consecutive fullwords starting D(B)
* These are type RS instructions. Encode – OPCODE, R1, R2, B, D
* SUB STM 0,15,SUBSAVE saves registers
* LM \_\_\_\_\_\_\_\_, 0(1) load new thigns into registers
* …. Subroutine main code
* LM 0,15,SUBSAVE sets registers back to what they were before the function
* BR 10 branch back to main code

Small subroutine example subroutine ABS to set a fullword = its absolut value

* X DS F - somewhere in memory
* PLIST DC A(X)
* MAIN CSECT
* ….
* LA 1,PLIST
* BAL 10,ABS
* ….
* ABS LM 0,13,ABSSAVE
* L 2,0(1) loads the address of X into 2
* L 3,0(2) loads the value of X pointed to by the address of 2
* LPR 3,3 makes register 3 positive
* ST 3,0(2) store value of register 3 to the address pointed at by reg 2
* LM 0,15,ABSAVE
* BR 10
* ABSAVE DS 16F

Subroutine to print table

* PLIST DC A(TABLE,EOT)
* The output line is a local variable in the subroutine

Example Subroutine : SWAP

* VARA DS F
* VARB DS F
* PARMS DC A(VARA,VARB)
* LA 1,PARMS
* BAL 11,SWAP
* …
* SWAP STM 0,15,SWAPSAVE
* LM 2,3,0(1) unload params, reg 2 = A(VARA), reg 3 = A(VARB)
* L 4,0(2)
* L 5,0(3)
* ST 4,0(3)
* ST 5,0(2)
* LM 0,15,SWAPSAVE
* BR 11

Counters

* BCT R,D(X,B) - Branch on Count, 1. Decrement R by 1, if R does not now contain 0, go to the address
* L 5,=F’5’
* LOOP XPRINT =CL6’ Hello’,6
* BCT 5,LOOP
* BCTR R1,R2 – Branch on count register, 1. Decrement R1, if R1 does not now contain 0 and R2 != 0 go to the address in R2.
* BCTR 7,0 common way to subtract 1 from register 7

Reading a file

* LOOP1 XREAD BUFFER,80
* BC B’0100’, ENDLOOP1
* LA 1,BUFFER
* LOOP2 XDECI 4,0(1)
* BC B’0001’,ENDLOOP2
* …
* ENDLOOP2
* ENDLOOP1 store the EOT VAL
* BUFFER DS 80C
* DC C’\*’

Printing a set number of items on a line

* Output line is a table of 5 values each is 12 char long

Clearing a Line

* MVC D1(L,B1),D2(B2) - Move character, copies character from 1 place to another.
  + L – length, D2(B2) – address of the source, D1(B1) – address of the destination
* MVC 4(16,3),8(7) - copy 16 bytes from 8(7) into 4(3)
* NAME1 DC CL5’HELEN’
* NAME2 DC CL5
* MVC NAME2(5),NAME1
* MCI D(B)i - Move character immediate, i is an Immediate byte. Copy byte i into the byte at D(B)
* MVI NAME1,C’Y’ - now Name 1 will be ‘YELEN’
* MVC is type SS, Op Code, length, B1,D1,B2,D2 6 bytes encoded, 256 max byte movement
* MVC 4(16,3),8(7) - encoded is D2 0F 3004 7008. 16 -1 = F
* MVI IS TYPE SI instruction, Op Code, I , B, D
* MVI 12(6),X’4B’ - encoded is 92 4B 600C
* CLC D1(L,B1),D2(B2) - compare logical character – compare the character values, L bytes long, at the addresses, left to right, and sets CC: 0-equal,1-left<right, 2-left>right
* ‘HERE’ > ‘HEAR’
* CLI D(B),I -Compare Logical Immediate, compares 2 bytes and sets CC
* We want to refill the line with spaces to blank it
* LINE DC 61C’ ‘
* MVC LINE(61),61C’ ‘ - possible but uses 61 byte literal, waste of space
* MVI LINE,C’ ‘
* MVC LINE+1(60),LINE - destructive overlap
* Can be done with lines longer than 1
* Message DS 20C
* MVC MESSAGE(2),=CL2’AB’
* MVC MESSAGE+2(18),MESSAGE => AB,AB,AB….

Skipping some lines

* LTR 1,1 LTR 1,1
* BNZ Z \* + 8 same BNZ \*+8
* L 1,=F’4’ L 1,=F’4’
* AHEAD

O

* ORG address - is used as we define variables, will go to that address
* ORG - by itself, go back to the highest previously defined location
* HLINE DC C’1’
* DC 120C’ ‘
* ORG HLINE+10
* DC CL10’FIRST NAME’
* ORG HLINE+30
* DC CL9’LAST NAME’
* ORG
* -
* ORG MAIN+(\*-MAIN+15)/16 \* 16
  + MAIN+(\*-MAIN+15) brings you to the start of variables
  + /16 \* 16 rounds and puts you on a multiple of 16

More on subroutines / external

* Part of the main CSECT, the largest amount of space in CSECT is D(0,15), D is 0-4095 so internal subroutines can take up too much space
* External subroutines use a separate CSECT, has its own base register(not 15)
* Fancy code to call SUB
* By convention register 15 is the address of the CSECT, REG 1 is of the parameter list, register 14 is the return address, register 13 is the area of the local save area
* Each CSECT is individually assembled
* To call an external subroutine:
* L 15,=V(SUB) - v means virtual, will be replaced later with the actual address, this is done in the linking step
* L 1,PLIST
* BALR 14,15
* -
* SUB CSECT
* STM 14,12,0(13) save registers in MAIN’s save area
* LR 12,15 base register
* USING SUB,12
* Code to crosslink save areas
* LA 14,SUBSAVE
* ST 14,8(0,13)
* ST 13, 4(0,14)
* LR 13,14
* …
* L 13,4(0,13) return from sub
* LM 14,12,12(13)
* BR 14
* We will have an 18 full word save area
* MAIN save area, 0 – not used, 1 – backward pointer, 2 – forward pointer = A(sub save area), 3 – R14, 4 – R15, 5 – R0 ….., 17 – R12
* SUB save area, 0 – not used, 1 – backward pointer = A(main save area), 2 – forward pointer, 3 – 18 empty ready to be filled

External Subroutine Example

* IN MAIN
* TABLE DS 40F
* EOT DS F
* RESULT DS F
* PLIST DC A(TABLE,EOT,RESULT)
* MAINSAVE DS 18F
* L 15,=V(COUNT)
* LA 1,PLIST
* BALR 14,15 reg 15 = address of count, reg 1 address of PLIST, reg 14 return address, reg 13 address of save area
* COUNT CSECT
  + LM 14,12,12(13)
  + LR 12,15
  + USING COUNT,12
  + LA 14,CSAVE
  + ST 14,8(0,13)
  + ST 13,4(0,14)
  + LR 13,14
  + LM 2,4,0(1)
  + SR 5,5
* LOOP C 2,0(3) are we done
  + BE LEND if so, stop
  + A 5,=F’1’ increment counter
  + A 2,=F’4’ advance pointer
  + B LOOP
* LOOPEND ST 5,0(0,4) store result
  + L 13,4(0,13)
  + LM 14,12,12(13)
  + BR 14

Options

* A subroutine can provide a return code 0-4095
* In Register 15, modify exit linkage,
  + L 3,4(0,13)
  + L 14,12,(0,13) SKIP 15
  + LM 0,12,20(13)
  + BR 14
* Return a computed value in Register 0, so: skip 0

Table processing (sorting)

* Int T[50], selection sort in C++
* For (I = 0; I < 50; i++) {
  + Small = I;
  + For (J=0; J<50;J++)
    - If(T[J] < T[small]
      * Small = J
    - Swap(T[I],T[small]
* }
* TABLE DS 50F
* EOT DS F ASSUME EOT = address of the first unused fullword
  + LA 4,TABLE
* LOOP1 C 4,EOT AT END OF TABLE
  + BNL END1 stop this loop
  + LR 6,4 small = I
  + LA 5,4(0,4) J = I + 1
* LOOP2 C 5,EOT at tend of table
  + BNL END2
  + L 7,0(5) value of T[J]
  + L 8,0(6) value of T[small]
  + CR 7,8 is T[J] < T[small]
  + BNL SKIP if not skip ahead
  + LR 6,5 small = J
* SKIP LA 5,4(5) J++
  + B LOOP2
* END2 L 9,0(4) value of T[I]
  + ST 9,0(6)
  + ST 8,0(4)
  + LA 4,4(4) I++
  + B LOOP1
* END1 DS 0H

Zone decimal number format

* 1237 - in EBDIC : F1F2F3F7, last byte is the sign digit \_\_ 0-9 ACEF – positive, BD – negative
  + F4F9B6, B makes it negative, = -496
* ZNUM1 DC ZL3’518’ stored as ‘F5 F1 F8’ F8 can be E8, A8 or C8, they are all the same
* ZNUM2 DC ZL5’-247’ stored as ‘\_\_ \_\_ F2 F4 B7’ right justified and padded with zeroes on left
* Inefficient: we are using 10 values out of the 256 for each byte
* Redundant: because we have several ways to store the same number
* We can’t do arithmetic or comparisons directly with zone decimal numbers

Packed decimal numbers

* 12|3F – 2byte – bytes all but the last byte are full of base 10 digits, last byte : 0-9 sign digit A-F
* We can do arithmetic and comparisons with packed decimal numbers
* PNUM1 DC PL2’-79’ : 07 9D right justified and padded with 0 on the left
* BIGP DS PL16 31 digits and a sign digit
* Popular because you can use packed digit numbers for enormous numbers and do operations
* PACK D1(L1,B1),D2(L2,B2) convert zoned decimal to packed decimal
  + D2(B2) = address of a zoned decimal L2 bytes long, D1(B1) address of a packed number L1 bytes long
* ZNUM2 DC ZL4’3918’ = F3 F9 F1 F8, PNUM2 DS PL3
* DC ZL4’248’ = F0 F2 F4 F8
* Zone decimal numbers are right justified
* PACK PNUM2(3),ZNUM2(4) – takes last byte and switches digites, F8 – 8F and throws away other Fs, so F3 F9 F1 F8 -> 03 91 8F
* ZL3’-579’ = F5 F7 D9 🡪 PL2, 57 9D
* UNPK D1(L1,B1),D2(L2,B2), D2(B2) = address of a packed num L2 bytes, D1(B2) = adderss of a zone num L1 bytes long
* PNUM4 DC PL3’-7182’ 🡪 ZNUM DS ZL5, UNPK converts packed to zone
* 07 18 2B 🡪 F0 F7 F1 F8 B2
* What if we have something random we are trying to pack, 3A 77 2B 10 packed 🡪 0A 7B 01
* Pack of unpack can be used to switch the digits of something 1 byte long

Packed Decimal Operations

* AP D1(L1,B1),D2(L2,B2) L1 and L2 are lengths, the sum will be at D1(B1), add packed
* PNUM5 DC PL2’137’, PNUM6 DC PL3’14892’, AP PNUM6(3),PNUM5(2) = 14892 + 137 = 15029
* AP requires valid data, if we have that data, we will have a data exception – 0007
* If the sum is too large to fit in L1 bytes, decimal overflow exception
* Sets the condition code, 0 - sum = 0, 1 - sum < 0, 2 – sum > 0, 3 – overflow
* SP D1(L1,B1),D2(L2,B2) L1 and L2 are lengths, the sum will be at D1(B1), subtract packed
* ZAP D1(L1,B1),D2(L2,B2) zero and add packed, set first arg to 0, and add second packed. Assignment operation
  + Requires a valid packed number in second argument
* CP D1(L1,B1),D2(L2,B2) compare packed, compares two numbers
* If x’153C’, x’153D’, x’153E’; CLC would get the wrong answer
* Encoding AP D1(L1,B1),D2(L2,B2) – operation code L1 L2 B1 D1 B2 D2
  + Lengths have 1 less than their actual values
  + AP 4(3,7),12(2,9) -> FA 21 7004 900C
* MP D1(L1,B1),D2(L2,B2) Multiply packed, product will be at D1(B1). We need L1 > L2, and L2<=8, the first L2 bytes of first argument must be 0
* PQA DC PL4’23’, PQB DC PL2’5’; MP PQA(4),PQB(2) = 115, fits in PQA
* PQC DC PL4’2000000F’, PQB DC PL2’5’; MP PQA(4),PQB(2) = 10mil,
* PQD DC PL4’2300’, MP PQD(4),PQB(2) doesn’t work, PQD >00 02 30 00 F, not enough 0s infront
* DP D1(L1,B1),D2(L2,B2) divide packed, 1st number ( at D1(b1) ) will contain quotient(L1-L2 bytes) and remainder(L2 bytes)
* Divide by 0 yields a decimal divide exception
* We need L2 < L1 and L2 <= 8, if quotient is too large to fit L1-L2 we have a decimal divide exception
* X’0355A’, x’5A’ divided is 7A 0A

Data type conversions

* CVD R,D(X,B) D(X,B) = address on a double word boundary, convert the binary value on R to a an 8=byte packed decimal value at D(X,B). Convert to Decimal
  + TEMP DS D and CVD 3,TEMP
* CVB R,D(X,B) address on a doubleword boundary, need a valid packed decimal number at D(X,B) 8 bytes long
* If value is too big to fit in reg, we have an exception, fixed point divide exception

Shift and Round

* SRP D(L,B),N,R Shift and round packed, N is a num 0-63 encoded as N(0,0) how far to shift, 0-31 we shift left, 32-63 we shift right by 64-N positions; R is 0-9,
  + SRP VAR(4),2 : 0000071F -> 0007100F
* Right shift – PEF DC PL4’59761’ ;; SRP PEF(4),64-1,5 🡪 00 05 97 6F
  + SRP PEF(4),64-2,5🡪00 00 59 8F

Edit

* PGH PL2’13’ ;; LINE … OUTPGH DS CL4
* ED D1(L,B1),D2(B2) - edit, D2(B2) is the address of packed num, D1(B1) is the address of destination, L is length of destination
* Before we use edit, the destination contains an edit pattern
* MVC OUTPGH(4), =X’\_\_\_\_\_\_\_\_\_\_’ 🡪 ED OUTPGH(4),PGH
  + =X’40 20 20 20’
* Edit patten: 1st byte = fill char, usually x’40’, other bytes x’20’ – digit selector, if significance is on or digit we are looking at is not 0, print the digit else print fill
* PGH = x’013F’ ;; OUTPGH after edit is x’40 40 F1 F3’ ;; significance is turned on after 1(F1)
* X’21’ is the significance starter, like x’20’ but will always turn on significance afterwards
* PGH = x’000F’ ;; pattern: X’40 20 21 20’ ;; OUTPGH after edit is x’40 40 40 F0’ ;;
* PJK = x’00147F’ ;; patter: x’40 20 20 20 21 20’ ;; OUTPJK is x’40 40 40 40 F1 F4 F7’
* PJK = x’54321F’ ;; patter: x’40 20 20 6B 20 21 20’ ;; OUTPJK is x’40 F5 F4 6B F3 F2 F1’
* PJK = x’00321F’ ;; patter: x’40 20 20 6B 20 21 20’ ;; OUTPJK is x’40 40 40 40 F3 F2 F1’
  + Use ebcdic for inserted chars
* Inserted character, if significance is on, print it, else print fill
* COST = x’00 25 93 7F’, COSTOUT DS CL10, x’40 20 20 6b 20 21 20 4B 20 20’,
  + out is x’40 40 40 40 F2 F5 F9 4B F3 F7’
* For negative nums we add a byte
  + If negative, significance is unchanged, if positive, significance is off
* ED will set the condition code if number is 0 negative or positive 0,1,2
* EDMK D1(L,B1),D2(B2) edit and mark, much like ED, will set Register 1 = address of first character such that 1. It was formed by replacing x’20’ or x’21’ by a digit and 2. Significance was off before that
* PAY DC PL3’35695’ 35 69 5F, PAYOUT DS CL7,
* MVC PAYOUT(7),=X’402021204B2020’ 🡪 40 F3 F5 F6 4B F9 F5
* Some instances, significance is off and reg 1 is not set, usually with very small numbers
* How to get around, is preset, reg 1 before all the code, LA 1,(address of the byte at x’21’)

Dummy sections for complicated tables

* ENTRY DSECT
* DNAME DS CL20
* DTELE DS ZL10
* DAGE DS ZL3
* ….
* LA 2,TABLE
* USING ENTRY,2 attach the DSECT to register 2
* MVC OUTNAME(20),DNAME
* .
* DROP 2 detach DSECT from the register

Translate

* TR D1(L,B1),D2(B2) translate, first address of a character string L bytes long, second address is the address of the translate table. Goes through a string like ‘hello’ considers each char as a number 0- 255 and replace character N with the byte at TRTABLE+N
* ‘A’ = X’C1’ = 193, replaces ‘A’ with the byte at TRTABLE+193
* Translate table is a 256 byte table
* TRT D1(L,B1),D2(B2) translate and test, now address 2 is of a search table. For each byte N in the string: if (STABLE+N) is not 0, stop and R1 = address of char found, R2, last byte = the char from the table, condition code = 2 if we found the character at the end of the string, 1 if we found it earlier, 0 if we didn’t
* STABLE DC 256x’00’
  + ORG STABLE+C’A’ uppercase search table
  + DC 9X’FF’
  + ORG STABLE+C’J’
  + DC 9X’FF’
  + ORG STABLE+C’S’
  + DC 8X’FF
* STABLE DC 256X’00’
  + ORG STABLE+C’ ‘
  + DC X’01’
  + ORG
* NSPTABLE DC 256X’01’ search for anything not 0
  + ORG NSPTABLE+C’ ‘
  + DC X’00’
  + ORG
* STR DC CL10’ABC\_ \_ 123 \_ \_’
* TRT STR(10), SPTABLE == Reg 1 = address of STR+3, Reg2, last byte = X’01’, CC = 1

Execute

* EX R,D(X,B) address is of an instruction. 1. Makes a copy of the instruction at D(X,B) 2. Modify the copy: 2nd byte of copy =2nd byte or 4th of R 3. Execute the modified copy
* Reg 5, last byte contains X’23’
* MYCR CR 0,0 -> Object code = 1900
* EX 5,MYCR takes the 2nd byte ‘00’
* 0 0
* OR 2 3
* Bit 1 or bit 2 = 1 if at least one of two bits is a 1, =0 otherwise
* 0000 0000
* 0010 0011
* --------------
* 0010 0011
* So EX changes MYCR to 1923, CR 2,3 and executes it
* If we try to use EX recursively, we have an execute exception
* Put MYCR after BR 14 among the variables
* \_\_\_\_123\_\_\_\_\_\_
* TRT BUFFER(80),DTABLE with a DTABLE for digits
* LR 4,1
* TRT 0(80,1),NDTABLE with a NDTable that will skip all digits
* SR 1,4 now Reg 1 = length of the number
* (in variables) - MYPACK PACK PNUM(5),0(0,4) - F2 40 (address)
* BCTR 1,0
* EX 1,MYPACK makes PACK PNUM(5),0(3,4) 3 is the 3 byte length of the number

Some TR examples : find a word

* Buffer: \_\_\_\_JERRY\_\_\_\_ ;; FNAME DC 12C’ ‘
* NBTABLE – non blank table, BTABLE – Search table for a blank
* TRT BUFFER(80),NBTABLE
* LR 4,1
* TRT 0(80,1),BTABLE
* SR 1,4 ->reg 1 length of name
* BCTR 1,0 correct the length
* EX 1,MYMVC
* In Memory
* MYMVC MVC FNAME(0),0(4)

Some TRT Examples: find a number

* BUFFER \_\_\_\_\_ 4719\_\_\_\_\_\_3658\_\_\_\_
* TRT BUFFER(80),NBTABLE
* LR 4,1
* TRT 0(80,1),BTABLE
* LR 5,1
* SR 5,4
* BCTR 5,0
* EX 5,MYPACK1
* TRT 0(80,1),NBTABLE
* LR 4,1
* TRT 0(80,1),BTABLE
* SR 1,4
* BTCR 1,0
* EX 1,MYPACK2
* MYPACK1 PACK PNUM1(3),0(4)
* MRPACK2 PACK PNUM2(3),

Bitwise Instructions

* Bit 1 OR Bit 2 = 1 if at least one of them is a 1 or 0 otherwise
* Bit 1 AND Bit 2 = 0 if at least one of them is 0, or 1 otherwise
* Bit 1 XOR Bit2 = 1 if they are not equal, 0 if they are equal (either or)
* 0110 1110 0110 1110
* OR 1101 1011 AND 1101 1011
* --------------- ---------------
* 1111 1111 0100 1010
* NR R1,R2 an AND operations on 32 pairs of bits, new Val in R1
* OR R1,R2 an OR operation
* XR R1,R2 an XOR operation
* N R,D(X,B) AND operation, must be on a fullword
* NI D(B),I AND operation on the byte at D(b) and the immediate byte
* BYTE DC B’11001010’
* OI BYTE,B’0001 0000’ - result in BYTE will be B’1101 1010’ only 1 bit changed
* NI BYTE,B’0111 1111’ - result in BYTE will be B’0100 1010’ turned 1 bit off
* NC D1(L,B1),D2(B2) - AND operation on 2 L-byte values at the two addresses
* XR 3,3 sets a register to 0
* XR 1,2 XOR swapping, this will swap two things
* XR 2,1
* XR 1,2
* MESSAGE KEY
* MESSAGE = MESSAGE XOR KEY XOR encryption
* ….. send it to someone who knows the key
* MESSAGE = MESSAGE XOR KEY unencrypts

Binary Shifts

* Reg 3 = X’1234ABCD’ Shift the bits 4 to the right 🡺 X’01234ABC’
* SRL R3,4 Shift right logical, second Val, length can be 0-63
* SLL R3,8 shift left logical 🡺 X’34ABCD00’
* SLA R,N Shift left arithmetic, multiply by 2^N, the signed bit stays in place, the other 31 bits move left, zeroes appear at the right. Sets CC 0 if 0, 1 neg, 2 pos, 3 overflow
* Reg 7 = B’010…0’ ;; SLA 7,1 🡺 B’00…0’ overflow if a bit not matching the sign bit is lost
* SRA R,N shift right arithmetic, the sign bit doesn’t move, the other 31 bits move right, copies of the sign bit appear at the left, sets CC 0, 1, 2 not 3 because you can’t overflow
* For 64 bit integer in 2 reg, even-odd, SLDL shift left double logical, SRDL,SLDA,SRDA
* IC R,D(X,B) copy the byte at D(X,B) into 4th byte of R
* STC R,D(X,B) copy the 4th byte of R into the byte at D(X,B)
* ICM R, mask 4 bits long, D(B) insert character under mask copy consecutive bytes from D(B) into some bytes of R corresponding to 1s in the mask
* ICM 4,B’1100’,8(3) reg 4 = x’A3F1784E’ at 8(3) we have x’0F1A’ finally result is x’0F1A784E’
* ICM 4,B’0101’,8(3) == x’A30F781A’
* STCM R,mask,D(B) copy some bytes from R corresponding to 1s in the mask, into the address
* Reg 3 = x’F1E2D3C4’ PLACE DC X4’21EF34AA’ STCM 3,B’1011’,PLACE 🡺 x’F1D3C4AA’
* CLM R,mask,D(B) Compare logical under mask, compare some bytes of R to consecutive bytes at D(B)
* Reg 11 = x’00FF54EA’ 4(6) we have x’2A390BF1’ CLM 11,B’1001’,4(6)
  + Compares ’00 EA’ < ‘2A 39’ cc = 1
  + CLM 11,B’0110’,4(6) 🡺 ‘FF54’ > ‘2A39’ cc = 2
* TM D(B),8bit mask or immediate byte test under mask, look at some bits in the byte at D(B), corresponding to 1s in the mask and sets condition code,
  + 0 if mask is all 0 or all selected bits are 0, 3 if all selected bits are 1, 1 otherwise
* BYTE DC B’01101110’ is the 6th bit a 1 or a 0, TM BYTE,B’00000100’, CC set to 3

Halfword Instructions

* H2 DC H’5’ stored as a signed 16bit integer
* LH R,D(X,B) Load Half, convert the 1 bit value at D(X,B) to a 32 bit value in R
* STH R,D(X,B) copy the last 2 bytes in R into the half word at the address
* AH R,D(X,B) add two numbers, put the sum in R, overflow is possible, set the condition code
* SH R,D(X,B) similar, just subtracts
* CH R,D(X,B) compare 2 numbers and set the condition code
* MH R,D(X,B) multiply the 32 bit num in R and the 16bit value in the halfword and product in R
  + Only uses 1 register, useful in mult by small num, MH 7,=H’3’
* No divide half

Storing data in small amounts of space

* Ex. Month 1-12 storable in 4 bits, Day 1-31 5 bits, bowling scores 0-300 9 bits, year – 1900 storable in 7 bits
* Date YYYYMMDD store in 2 bytes or 16 bits. Use the PACK instruction to put Day in an 8 byte Packed decimal field on a doubleword boundary, then use CVB. Use pack on month, and CVB so 1 reg with 10001, and another with 1011. Use SLL to put day in position and add

Assign8

* Store ID as 4 bytes
* xor encryption with the id as the key
* convert day of week to 1 digit 0 to 6 and store in 3 bits
* store the hour (0-23) in 5 bits
* store the minute (0-59) in 6 bits
  + Ex. Monday 10:30
  + pack the hour, cvb, register contains hour: R7. Have day of week in a register: R8
  + R7 0….0,01010 SLL 8,5
  + R8 0….0,001
  + 0,00101010
* store the room number in 10 bits

Floating point numbers

* 4 byte floating point format – sign bit, 7 bits -exponent(-64 to 63) stored as value 0-127, 24 bits is the fraction
* 24 bits: 2^24 = 16 million, 7 sig figs
* Long format 8 bytes, similar but fraction is 56 bits
* FPA DS E short
* FPB DS D long

Merging tables

* Table 1 – ABC GHI LMN Table 2- DEF JKL PQR Table3 – ABC DEF GHI JKL …

Macros

* MACRO
* &LABEL NAME list of 0 or more symbolic parameters
* …
* MEND
* Ex. Set a fullword = sum of 2 other fullwords
* MACRO
* ADD &A,&B,&C
* B \*+8
* DS F
* ST 4,\*-4
* L 4,&A
* A 4,&B
* ST 4,&C
* L 4,\*-20
* MEND
* ADD X,Y,Z
* Can’t use ordinary label, when macro is called twice, it will have a duplicate label error
* PRINT NOGEN will suppress printing the generated lines
* .\* is a macro comment, not reproduced in the listing
* .label label that exists only in the macro code, doesn’t show in program listing put this on special macro instructions
* AIF(logical condition).LABEL logical conditions: 0 false, 1 true, comparison
  + Comparisons: EQ – equals, GT – Greater than, LT - Less than, NE – not equal, GE – GT or EQ, LE – LT or EQ
* AIF(‘&A’ EQ ‘ ‘).ERROR
* MNOTE ‘message’ printed in the assembler listed
* .ERROR MNOTE ‘Missing argument’
* AGO .label unconditional branch
* .\* Ex. Set a fullword = its own absolute value
* .\* &A = label or D(B) address of a fullword
* MACRO
* &LABEL ABS &A
* AIF (‘&A’ EQ ‘‘).ERROR checks for parameters
* &LABEL B \*+8
* DS F
* ST 2,\*-4
* L 2,&A
* LPR 2,2
* ST 2,&A
* L 2,\*-22
* MEXIT
* .ERROR MNOTE ‘Missing Argument’
* MEND
* ANOP means do nothing
* If you have an instruction too long for one line and want to continue on next line
* Col 1 Col 10 Col 16 …………………, Col 72 – X
* To continue a line, end 1st line with a comma, an X in Col 72, resume in column 16 of next line

Unique Labels in Macros

* &SYSNDX – counter initially = 0001; each time a program calls a macro that uses this, it increments. L&SYSNDX \_\_\_\_\_\_\_\_\_\_\_\_
* Macro to find the larger of 2 numbers
* MACRO
* LARGER &LEFT,&RIGHT,&RESULT
* AIF (‘&LEFT’ EQ ‘’).ERROR
* AIF (‘&RIGHT’ EQ ‘’).ERROR
* AIF (‘&RESULT’ EQ ‘’).ERROR
* B M&SYSNDX
* L&SYSNDX DS F
* M&SYSNDX ST 7,L&SYSNDX
* L 7,&LEFT
* C 7,&RIGHT
* BC B’0010’,N&SYSNDX
* L 7,&RIGHT
* ST 7,&RESULT
* B P&SYSNDX
* N&SYSNDX ST 7,&RESULT
* P &SYSNDX L 7,L&SYSNDX
* MEXIT
* .ERROR MNOTE ‘Missing argument’
* MEND
* The ampersand variables will replace with the D(B) address, if &RESULT = 4(10)
  + MVC &RESULT(4),&RIGHT will be MVC 4(10)(4),&RIGHT and thus have an error
* // SYSIN DD \*
* Your macros here
* // DD DSN=….driver program

More on Macros

* NAME List of symbolic parameters
* Often called positional parameters(listed in order, no default values except ‘’)
* There are also keyword parameter – we can have default values
* Ex. ARRIVE &REGS=NO,&BR=,&CSECT=NO,&SAVE=NO
* ADD &SUM,&LIST <- sublist called by ADD TOTAL,(A,B,C,D)
* Access sublist with &LIST(1) -> A, &LIST(2) -> B
* &SYSLIST(n) = the nth positional parameter
* &SYSLIST(N,K) = the kth item in the nth positional parameter (a sublist)
* Ex. MACALL ONE,TWO,(3,(456),,8),,TEN
* &SYSLIST(0) Is “CALL3”, &SYSLIST(1) is “ONE”, &SYSLIST(2) is “TWO”, &SYSLIST(3) is ‘,(3,(456),,8)’
* &SYSLIST(3,2) is ‘(4,5,6)’, &SYSLIST(3,2,1) is ‘4’
* IF3 : Check the value of a fullword and branch in one of 3 ways for zero, positive or negative
* MACRO
* &LABEL IF3 &VALUE,&NEG,&ZERO,&POS
* AIF(…).ERROR
* B M&SYSNDX
* L&SYSNDX DS F
* M&SYSNDX ST 2,L&SYSNDEX
* L 2,&VALUE
* LTR 2,2
* BE Z&SYSNDX
* BL N&SYSNDX
* B P&SYSNDX
* Z&SYSNDX L 2,L&SYSNDX
* B &ZERO
* MEXIT
* N&SYSNDX L 2,L&SYSNDX
* B &NEG
* MEXIT
* P&SYSNDX L 2,L&SYSNDX
* B &POS
* MEXIT
* .ERROR MNOTE ‘MISSING ARGUMENT’
* MEND

Local Vars in Macros

* LCLA &N local arithmetic, initial value 0,
* &N SETA expression change the value
* AIF (&N LT 8).something comparison
* LCLB &F &F is a boolean variable, 0 -false 1-true, initially 0
* &F SETB Logical Expression
* LCLC &S character string
* &S SETC String Expression
* &S SETC ‘Hello’
* &S SETC &ABC
* &S SETC ‘hello’&ABC
* &S SETC &ABC.’hello’
* &S SETC &ABC(2,3) substring
* Global Vars in Macros - GLBA, GLBB, GLBC
* MACRO
* ADD &SUM,&LIST with &LIST being a sublist
* LCLA &N
* &N SETA 1
* N’&P = the number of items in a sublist
* K’&A the number of chars in &A, If &A is a symbolic parameter its value is a char string
* L’&P = the number of bytes defined at that label
* T’&P = one letter indicating the type of data
* &LABEL ADD &TOTAL,&LIST macro using local arithmetic variable
* LCLA &N
* AIF….
* B M&
* DS F
* M& ST 2,L&SYSNDX
* SR 2,2
* &N SETA 1
* .LOOP AIF(&N GT N’&LIST).LPEND
* A 2,&SYSLIST(2,&N)
* &N SETA &N+1
* AGO .LOOP
* .LPEND ANOP
* ST 2,&TOTAL
* L 2,L&SYSNDX
* MEXIT
* .ERROR
* MEND