DIGITAL RESEARCH

SID ™

Productivity Tool

Command Summary

COPYRIGHT

DISCLAIMER

TRADEMARKS

The SID Productivity Tool Command Summary was prepared using the Digital Research TEX Text Formatter and printed in the United States of America.

****
* First Edition: 1978 *
****

SID COMMAND SUMMARY

STARTUP

(1) SID
(2) SID x.y
(3) SID X.HEX
(4) SID X.UTL
(5) SID x.y u.v

Form (1) starts SID without a test program, (2) loads the test program x.y (y is normally COM), (3) loads X.HEX in Intel "hex" format, (4) loads and executes utility x, (5) loads x.y with the symbol table u.v (normally x.SYM). Example:

SID SORT.COM SORT.SYM

RESPONSE

(1) #
(2) SYMBOLS
(3) NEXT PC END
nnnn pppp eeee

Form (1) indicates SID is ready to accept commands, (2) indicates machine code loaded, commencing symbol table load, (3) shows successful machine code and/or symbol load where nnnn, pppp, and eeee are hexadecimal values giving the next unfilled machine code location, the initial program counter, and the last free memory location, respectively.

LETTER COMMANDS

A	Assemble	M	Move
C	Call	P	Pass Point
D	D isplay	R	Read
F	Fill Memory	S	Set Memory
G	Go	T	Trace
H	Hex	U	Untrace
I	Input Line	X	Examine
L	List Mnemonics		

COMMAND LINE

SID reads commands from the svstem console following the # prompt. Each command line is based upon the command letter and optional symbolic expressions. All CP/M Iine editing is available on 64 character lines terminated by carriage returns. A space serves as a comma delimiter. SID terminates whenever control-C is typed.

LITERAL NUMBERS

SID uses the hexadecimal number base, consisting of the decimal digits 0-9 along with the hex digits A-F. Numbers exceeding four digits are truncated to the right. Examples are:

30 3F 3f FF3E F3

DECIMAL NUMBERS

Decimal numbers are preceded by a # and consist of decimal digits 0-9. Numbers exceeding 65535 are truncated to the rightmost 16 bits. Examples are:

#48 #9999 #65535 #0

CHARACTERS

SID accepts graphic ASCII characters within paired string apostrophes ('). Strings of length greater than two are truncated to the right. The rightmost character of a two character string becomes the least significant byte. A one character string has a high order 00 byte, zero length strings are disallowed, and a pair of apostrophes within a string reduces to a single apostrophe. Lower case letters are not translated in strings. Examples are:

'a' 'A' 'xy' '#I'''

SYMBOL REFERENCES

SID symbolic expressions may involve symbol references when a symbol table is present:

(1) .s
(2) @s
(3) =S

Form (1) denotes the address of symbol s, (2) denotes the 16-bit value at .s, (3) denotes the 8-bit value at .s, where s is a sequence of characters matching a symbol table element.

QUALIFIED SYMBOLS

SID searches for a symbol match starting at the first symbol loaded until the first symbol matches. When duplicate symbols exist, a qualified reference of the form:

s1 /s2/ . . . /sn

matches symbols from left to right as the search proceeds sequentially through the symbol table. An example is:

ALPHA/GAMMA/I

SYMBOLIC EXPRESSIONS

Expressions consist of a left to right sequence of literal numbers, decimal numbers, character strings, and symbol references, separated by plus ("+") and minus ("-") operators. Values are added or subtracted, accordingly, with no overflow checks, to produce the final 16-bit result. a leading minus, as in -x, is computed as O-x. A leading plus, as in +X , is computed as x'+x, where xl is the value of the last expression typed. A sequence of n ^'s produces Nth stacked value in the program under test (see the G command). Blanks are not allowed within expressions. Examples are given with individual commands.

A ASSEMBLE

(1) As
(2) A
(3) -A

Form(l) begins in-line assembly at location s, where each successive address is displayed until a null line or "." is entered by the operator. Form (2) is equivalent to (1) with assumed starting address derived from last assembled, listed, or traced address. Form (3) removes the assembler/disassembler module, discards existing symbol information, and disables subsequent A or L commands. In this case, machine hex code is displayed in subsequent traces. Examples:

```
A100
A# 100
A.CRLF+5
A@GAMMA+@X-=I
A+30
```

CALL

(1) Cs
(2) Cs,b
(3) Cs,b,d

Form (1) performs a direct call from SID to location s in memory, without disturbing the CPU state of the program under test, and is most often used with SID Utilities. In this case, registers BC=0000, DE=0000. Form (2) calls s with data BC=b, DE=0000, while form (3) also fills DE=d. Examples:

```
C100
C#4096
C.DISPLAY
C@JMPVEC+=X
C.CRLF,#34
C.CRLF,@X,+=X
```

D DISPLAY MEMORY

(1) Ds
(2) Ds,f
(3) D
(4) D,f
(5) DWs
(6) DWs,f
(7) DW
(8) DW,f

Form (1) types memory contents in 8-bit format starting at location s for 1/2 screen with graphic ASCIIto the right of each line, (2) is similar, but ends at location f. Form (3) continues the display from the last displayed location, or the value of the HL register pair following CPU state display, for 1/2 screen, (4) is similar, but terminates at location f. Forms (5) through (8) are equivalent to (1) through (4), but display in word format (16-bits). Examples:

```
DF3F
D# 100,# 200
D.gamma,.DELTA+#30
D,.GAMMA
DW@ALPHA,+#100
```

F FILL MEMORY

Fs,f gd

Fills memory with 8-bit data d starting at location s, continuing through location f. Examples:

```
F100,3FF,ff
f.gamma,+#100,#23
F@ALPHA,+=I,=X
```

G GO TO PROGRAM

(1) G
(2) Gp
(3) G,a
(4) Gp,a
(5) G,a,b
(6) Gp,a,b
(7) -G . . .

Form (1) starts the program under test from the current PC without breakpoints. Execution is in real time. Form (2) is equivalent, but sets PC=p before execution, (3) starts from the current PC with a breakpoint at location a, (4) is similar to (3) but sets the PC to p. Form (5) is equivalent to (3) but sets breakpoints at a and b, while (6) presets the PC to p before execution. Upon encountering a breakpoint (or an externally provided RST 7), the break address is printed in the form:

```
*nn.nn
```

and the optional breakpoints are cleared. Forms given by (7) parallel (1) through (6), except "pass points" are not traced until the corresponding pass count becomes zero (see P command). The symbol "^" in an expression produces the topmost stacked value, which is used to set a break following a subroutine call. Given that a breakpoint has occurred at a subroutine, the command

```
G,^
```

continues execution with a return breakpoint set. Examples:

```
G100
G100,103
G.CRLF,.PRINT,#1024
G@JMPVEC+=I,.ENDC,.ERRC
G,.errsub
G,.ERRSUB,+30
-G100,+10,+10
```

H HEX VALUES

(1) Ha,b
(2) Ha
(3) H

Form (1) produces the hexadecimal sum (a+b) and difference (a-b) of operands. Form (2) performs number conversion by typing the value of a in the format:

hhhb #ddddd 'c' ssss

where hbhh is als hex value, dddd is the decimal value, c is the ASCH value, if it exists, and ssss is the symbolic value, if it exists. Form (3) prints the hex values for each symbol table element (abort with rubout). Examples:

```
H100,200
H#1000,#965
H.GAMMA+=I,@ALPHA-#10
H#53
H@X+=Y-5
```

I INPUT LINE

I c1 c2 . . . cn

Initializes default low memory areas for the R command or the program under test, as if the characters c1 through cn had been read and setup at the console command processor level. Default FCB's are initialized, and the default buffer is set to the initial input line.

Examples:

```
I x.dat
ix.inp y.out
I a:x.inp b:y.out $-p
ITEST.COM
ITEST.HEX TEST.SYM
```

L LIST CODE

(1) Ls
(2) Ls,f
(3) L
(4) -L . . .

Form (1) lists disassembled machine code starting at location s for 1/2 screen, (2) lists mnemonics from location s through f (abort typeouts with rubout). Form (3) lists mnemonics from the last listed, assembled, or traced location for I screen. Form (4) parallels (1) through (3), but labels and symbolic operands are not printed. Labels are printed in the form

ssss:

ahead of the lines to which they correspond. Non-8080 mnemonics are printed as

??= hh

where hh is the hex value at that location. Examples:

```
L100
L#1024,#1034
L.CRLF
L@ICALL,+30
-L.PRBUFF+=I,+TAT
```

M MOVE MEMORY

Ms,h,d

Move data values from start address s through address h to destination address d. Data areas may overlap during the move process. Examples:

```
M100,1FF,300
M.x,.y,.z
M.GAMMA,+FF,.DELTA
M@alpha+=x,+#50,+IOO
```

P PASS COUNTER

(1) Pp
(2) Pp,c
(3) p
(4) -Pp
(5) -P

A "pass point" is a program counter location to monitor during execution of a test program. A pass point has an associated "pass counter" in the range 1-FF (O-#255) which is decremented each time the test program executes the pass point address. When a pass count reaches 1, the pass point becomes a permanent breakpoint and the pass count remains at 1. Unlike a temporary breakpoint (see G), pass points with pass count 1 stop execution following execution of the instruction at the break address. Form (1) sets a pass point at address p with pass count 1, (2) sets pass point p with pass count c, (3) displays active pass points and counts, (4) clears the pass point at p (equivalent to Pp,O), and (5) clears all pass points. Up to 8 pass points can be active at any time. CPU registers are displayed when executing a pass point, with the header

    nn PASS hhhh ssss

showing the pass count nn and address hhhh with optional symbol ssss. Registers are not displayed if -G or -U is in effect until the pass count reaches 1. Execution can be aborted during the pass trace with rubout. Examples:

```
P100,ff
P.BDOS
P@ICALL+30,#20
-P.CRLF
```

R READ CODE/SYMBOLS

(1) R
(2) Rd

The I command sets up code and symbol files for subsequent loading with the R command. Form (1) reads optional code and optional symbols in preparation for program test, (2) is similar. but loads code and/or symbols with the bias value d. The sequence:

```
I X.Y
R
```

Sets up machine code file x.y (y is usually COM), and reads machine code to the transient area. If y is HEX, the file must be in Intel "hex" format. The sequence:

```
I X.Y U.V
R
```

also reads the symbol file u.v (u is usually the same as x, and v is normally SYM). The form:

```
I * u.v
R
```

skips the machine code load, and reads only the symbol file. When a symbol file is specified, the response

```
SYMBOLS
```

shows the start of the symbol file read operation. Thus, a "?" error before the SYMBOL message indicates a machine code read error, while "?" following the SYMBOL message shows a symbol file read error. Examples:

```
I COPY.COM
R
I SORT.HEX SORT.SYM
R
I merge.com merge.sym
R1000
I * test.sym
R-#256
```

S SET MEMORY

(1) Ss
(2) SWs

Form (1) sets memory locations in 8-bit format, (2) sets memory in 16-bit "word" format. In either case, each address is displayed, along with the current content. If a null line is entered, no change is made, and the next address is prompted. If a value is typed, then the data is changed and the next address is prompted. Input terminates with either invalid input, or a single "." from the console. Long ASCII input is entered with form (1) by typing a leading quote (") followed by graphic characters, terminated by a carriage return. The examples show underlined console input:

```
S100
0100  C3 34
0101  24 #254
0102  CF
0103  4B "Ascii
0108  6E =X+5
0109  D4  .
SW.X+#3(T
2300 006D  44F
2302 4F32   @GAMMA
2304  33E2   _
2306  FF11  O+.X+-I-#20
2308  348F   .
```

T TRACE MODE

(1) Tn
(2) T
(3) Tn,c
(4) T,c
(5) -T . . .
(6) TW . . .
(7) -TW . . .

Form (1) traces n program steps, showing the CPU state at each step, while (2) traces one step. Form (3) is used with SID utilities, and "calls" the utility function c at each trace step. Form (4) is similar to (3), but traces only one step. Form (5) parallels (1) to (4), but disables symbols. Form (6) parallels (1) to (4), but performs "trace without call" showing only local execution. Form (7) is similar to (6) with symbols disabled. Examples:

```
T100
T#30,.COLLECT
-TW=I,3EO3
```

U UNTRACE MODE

(1) U . . .
(2) -U
(3) UW . . .
(4) -UW . . .

U performs the same function as T, except the register state is not displayed. Forms (2) and (4), however, disable intermediate pass point trace (see P). U and T both run fully monitored, with automatic breaks at each instruction. Execution can be aborted with rubout. Examples:

```
Uffff
U#10000,.COLLECT
UW=GAMMA,.COLLECT
```

X EXAMINE CPU STATE

(1) X
(2) Xf
(3) Xr

Form (1) displays the CPU State in the format:

f A=a B=b D=d H=h S=s P=P i s

where f is the "flag state," a is the 8080 accumulator content, b is the 16-bit BC register pair value, d is the DE value, h is the HL value, s is the SP value, p is the PC value, i is the decoded instruction at p, and s is symbolic information. The flag, are represented by dashes when false, and their letters when true:

Carry Zero Minus Even parity
Interdigit carry

Form (2) allows flag state change, where f is one of C,ZM,E, orI. The current state is displayed (either "-" or the letter). Enter the value 1 for true, 0 for false, or null for no change. Form (3) allows register state changes where r is one of A, B, D, H, S, or P. Symbol information is given at s when i references an address, including LDAX and STAX. The form "=mm", is printed for memory referencing instructions (e.g., INR M ADD M), where mm is the memory value before execution. Examples with opera or input underlined:

```
XM
M 0
XB
3E04 3EFF
XP
446E CRLF+10
```

SID UTILITIES

Utilities execute with SID to provide additional debugging facilities. A utility is loaded initially by typing:

SID X.UTL

where x is the utility name. Upon loading, the utility is setup for execution with SID, and responds with:

.INITIAL = iiii
.COLLECT = cccc
.DISPLAY dddd

where iiii, cccc, and dddd are three absolute address entries to the utility for (re)initializing, collecting debug data, and displaying collected information, respectively. The SID symbol table contains these three entry names. A utility is reinitialized by typing:

Ciiii or C.INITIAL

The display information is obtained by typing:

Cdddd or C.DISPLAY

while data collection occurs during monitored execution using the T or U commands, where the second argument gives the collection address. xamples are:

```
Uffff,.collect
U#1000,403
TW1000,.COLLECT
UW@GAMMA,.COLLECT
```

Pass points may be set during data collection to stop the monitoring at the end of program areas under test. The actual initialization, collection, and display functions depend upon the particular SID utility.

THE HIST UTILITY

The HIST utility creates a histogram of program execution between two locations given during initialization. Program addresses are monitored during U or T mode execution, with summary data displayed at any time. Upon startup or reinitialization, HIST prompts with:

```
TYPE HISTOGRAM BOUNDS:
```

Respond with:

```
aaaa,bbbb
```

for a histogram between locations aaaa and bbbb, inclusive. Collect data in U or T mode, then display results. Output is scaled to the maximum collected value, accumulating until reinitialization. An example:

```
SID HIST.UTL
TYPE HISTOGRAM BOUNDS 100,A00
 INITIAL = 30O3
.COLLECT     3E06
.DISPLAY     3E09
#I SORT.COM SORT.SYM
#R
SYMBOLS
#UFF,.COLLECT
(register display and break)
#C.DISPLAY
(histogram@isplay)
U1000,.COLLECT
(display an eventual break)
C.DISPLAY
(update histogram display)
  C.INITIAL
 (histogram bounds reset)
   . . .
```

THE TRACE UTILITY

The TRACE utility provides a dynamic backtrace of up to 256 instructions which ended at the current break address. Instruction address collection occurs only in U or T mode. Pass points can be active, however, during the data collection, and will halt execution when the pass count becomes 1. Initialization clears the accumulated instructions, collection records the instruction address in a wraparound buffer, and display prints the backtrace in decoded mnemonic form with symbol references and labels when they occur. If "-A" is in effect, only instruction addresses are given. In this case ,TRACE is loaded by typing:

```
SID
#-A
#I TRACE.UTL
#R
ADDRESSES ONLY
 . . .
```

An example of normal operation:

```
SID TRACE.UTL
READY FOR SYMBOLIC BACKTRACE
#1 MERGE.COM MERGE.SYM
#R
#UFFF,.COLLECT
(register display, wait, break)
#C.DISPLAY
(symbolic backtrace appears)
```

IMPLEMENTATION NOTES

The SID program operates in about 6K bytes, and self-relocates directly below the BDOS (overlaying the CCP area). The SID symbol table fills downward from the base of SID. As the table fills, the BDOS jump address is altered to reflect the reduced free space. Programs which "size" memory using the BDOS jump address should not be started until all symbols are loaded.

The "-A" command increases the free space by about 1 1/2 K bytes. Any existing symbol information must be reloaded after issuing the command.

Programs will trace up to the BDOS where tracing is discontinued until control returns to the calling program. ROM subroutine tracing is discontinued when ROM is entered through a call, jump, or PCHL, and resumed upon return to the calling program in RAM.

Use rubout to abort programs running fully monitored in T or U mode, and an externally provided restart (RST 7) when running unmonitored with G.

8080 MNEMONICS

The 8080 mnemonics which follow (reproduced with permission from Intel(c) Corporation), can be entered directly in assembly mode (see A), and are produced by SID in list mode (see L). Data fields can consist of symbolic expressions. Given that "A100" has been typed, and that the symbols X, Y, and Z exist, the following is valid input:

```
MOV    A,B
MVI    A,FF
mviI   b,#255
MVI    M,'x'
LXI    H,'ab'
JMP    100
CALL   .X
JZ     @Y
Ixi    h,@X+=Z
JMP    .X/Y+5
```

Notable differences between MACand the SID "A" command are that no pseudo operations are allowed, operands are SID symbolic expressions*, labels cannot be inserted, and register references must be names, not numbers.

*In particular, note that

LXI H,'ab'

fills H with 'a' and L with 'b' due to the nature of SID expressions, which is counter to the MAC convention.

C3	JMP	CD	CALL	C9	RET
C2	JNZ	C4	CNZ	C0	RNZ
CA	JZ	CC	CZ	C8	RZ
D2	JNC	D4	CNC	D0	RNC
DA	JC	DC	CC	D8	RC
E2	JPO	E4	CPO	E0	RPO
EA	JPE	EC	CPE	E8	RPE
F2	JP	F4	CP	F0	RP
FA	JM	FC	CM	F8	RM
E9	PCHL				

06	MVI B,	C6	ADI	01	LXI B,
OE	MVI C,	CE	ACI	11	LXI D,
16	MVI D,	D6	SUI	21	LXI H,
1E	MVI E,	DE	SB I	31	LXI SP,
26	MVI H,	E6	ANI		
2E	MVI L,	EE	XRI		
36	MVI M,	F6	ORI		
3E	MVI A,	FE	CPI	09	DAD B
				19	DAD D
				29	DAD H
				39	DAD SP

04	INR B	05	DCR B		
OC	INR C	0D	DCR C		
14	INR D	15	DCR D		
1C	INR E	1D	DCR E		
24	INR H	25	DCR H	0A	LDAX B
2C	INR L	2D	DCR L	1A	LDAX D
34	INR M	35	DCR M	2A	LHLD Adr
3C	INR A	3D	DCR A	3A	LDA Adr

03	INX B	0B	DCX B	02	STAX B
13	INX D	1B	DCX D	12	STAX D
23	INX H	2B	DCX H	22	SHLD Adr
33	INX SP	3B	DCX SP	32	STA Adr

C7	RST 0	07	RLC	58	MOV E,B
CF	RST 1	0F	RRC	59	MOV E,C
D7	RST 2	17	RAL	5A	MOV E,D
DF	RST 3	1F	RAR	5B	MOV E,E
E7	RST 4			5C	MOV E,H
EF	RST 5			5D	MOV E,L
F7	RST 6			5E	MOV E,M
FF	RST 7			5F	MOV E,A
		00	NOP	60	MOV H,B
		76	HLT	61	MOV H,C
		F3	DI	62	MOV H,D
		FB	El	63	MOV H,E
				64	MOV H,H
				65	MOV H,L
				66	MOV H,M
C5	PUSH B			67	MOV H,A
D5	PUSH D	40	MOV B,B	68	MOV L,B
E5	PUSH H	41	MOV B,C	69	MOV L,C
F5	PUSH PSW	42	MOV B,D	6A	MOV L,D
Cl	POP B	43	MOV B,E	6B	MOV L,E
Dl	POP D	44	MOV B,H	6C	MOV L,H
El	POP H	45	MOV B,L	6D	MOV L,L
Fl	POP PSW	46	MOV B,M	6E	MOV L,M
		47	MOV B,A	6F	MOV L,A
E3	XTHL	48	MOV C,B	70	MOV M,B
F9	SPHL	49	MOV C,C	71	MOV M,C
		4A	MOV C,D	72	MOV M,D
		4B	MOV C,E	73	MOV M,E
		4C	MOV C,H	74	MOV M,H
		4D	MOV C, L	75	MOV M,L
EB	XCHG	4E	MOV C,M		--------------
27	DAA	4F	MOV C,A	77	MOV M,A
2F	CMA				
37	STCT	50	MOV D,B	78	MOV A,B
3F	CMCT	51	MOV D,C	79	MOV A,C
		52	MOV D, D	7A	MOV A,D
		53	MOV D,E	7B	MOV A,E
		54	MOV D,H	7C	MOV A, H
		55	MOV D,L	7D	MOV A, L
D3	OUT	56	MOV D,M	7E	MOV A,M
DB	IN	57	MOV D,A	7F	MOV A,A

80 ADD B
81 ADD C
82 ADD D
83 ADD E
84 ADD H
85 ADD L
86 ADD M
87 ADD A

88 ADC B
89 ADC C
8A ADC D
8B ADC E
8C ADC H
8D ADC L
8E ADC M
8F ADC A

90 SUB 8
91 SUB C
92 SUB D
93 SUB E
94 SUB H
95 SUB L
96 SUB M
97 SUB A

98 SBB B
99 SBB C
9A SBB D
9B SBB E
9C SBB H
9D SBB L
9E SBB M
9F SBB A

A0 ANA B
Al ANA C
A2 ANA D
A3 ANA E
A4 ANA H
A5 ANA L
A6 ANA M
A7 ANA A21

A8 XRA B
A9 XRA C
AA XRA D
AB XRA E
AC XRA H
AD XRA L
AE XRA M
AF XRA A

B0 ORA B
Bl ORA C
B2 ORA D
B3 ORA E
B4 ORA H
B5 ORA L
B6 ORA M
B7 ORA A

B8 CMP B
B9 CMP C
BA CMP D
BB CMP E
BC CMP H
BD CMP L
BE CMP M
BF CMP A