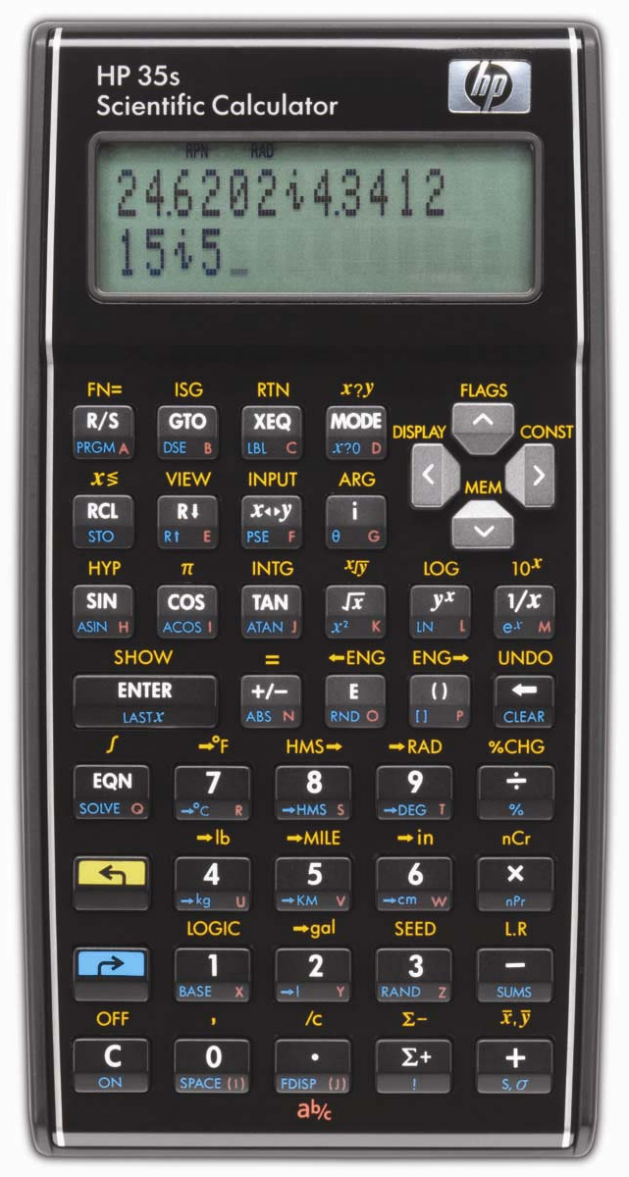
hp calculators



HP 35s Indirect register data packing program

The HP 35s and indirect registers

Saving program memory space

Saving indirect storage space

The program listing

Usage instructions

Entering program lines

Line by line analysis of the program

Saving keystrokes

Usage examples

The HP 35s and indirect registers

The HP35s contains registers or variables that can be referenced directly or indirectly. Variables A through Z can be directly addressed, as in a [↗] [STO] [A] instruction. Indirect addressing uses two of these direct variables as indices that hold the location or address where an operation is to be performed. The two variables that are used this way are [I] and [J]. The indirect registers begin at address 0 and can go up to 800, if the user allocates that many. That is 801 additional storage registers compared to the earlier HP 33s calculator.

It is also possible to address the direct variables and the statistics variables indirectly using addresses of -1 through -32. Address -1 would refer to the direct variable [A], address -26 would refer to the direct variable [Z], and -27 trough -32 would refer to the statistical summation registers. This is shown in a table on page 14-22 of the HP 35s user's guide.

The way indirect addressing works is to store the number corresponding to the register you wish to use in either [I] or [J]. Then you perform a [↗] [STO] [(i)] or [↗] [STO] [(j)] (or any other allowed operation). For example, if you wish to recall a value stored in direct register [A], you can either press [RCL] [A] or store -1 into [I] by [↗] [STO] [(i)] and then perform a [RCL] [(i)]. Both will recall the value stored in A.

At first glance, that may not appear to be worth doing, since it takes more key presses to use the indirect method. However, where it becomes very useful is when you need to work with a lot of numbers, often within a program, or when you may not be able to know in advance where the number you wish to use is stored.

Saving program memory space

For example, suppose you have 20 numbers you wish to sum. A direct program might be written having these numbers stored in [A] through [T]. The program to sum them (which would take 41 lines of code and at least 120 bytes of memory), might look like:

```
A001  LBL A      A005  RCL C      …
A002  RCL A      A006  +          A039  RCL T
A003  RCL B      A007  RCL D      A040  +
A004  +          A008  +          A041  RTN
```

On the other hand, a program using the indirect registers might have the numbers stored in indirect locations 1 through 20. The program to sum the 20 values might look like this one. This program only takes 10 lines of code and only 32 bytes of code. Sure, developing the second program might take a little more time than the first, but it comes at a great reduction of program memory space used.

```
A001  LBL A      A005  DSE I      A009  GTO A006
A002  20         A006  RCL (I)    A010  RTN
A003  STO I      A007  +
A004  RCL (I)    A008  DSE I
```

Saving indirect storage space

To allocate a portion of the HP 35s memory to hold indirect registers, store a non-zero value into the highest register needed. If you need 100 registers to hold numbers and if locations 0 to 99 will work for your need, storing a non-zero value into indirect storage location 100 will allocate HP 35s calculator memory to create the block of indirect registers 0 through 100. Warning: If you store a zero into memory location 100, the HP 35s will dynamically reclaim all zero indirect

storage registers starting with 100 and working down. This can cause quite a shock when you're not expecting it in a program or calculation.

Each indirect register, like each direct register and each stack register, can hold a variety of objects, such as a real number, a complex number, or a 2-D or 3-D vector. Since these take varying amounts of memory to hold them, the HP 35s allocates 37 bytes per register for each location, whether the register needs that many bytes or not (see page 14-24 of the HP 35s user's guide). This means that a group of indirect registers that are only going to hold a real number are only using 1/3 of the possible storage space per register.

To reclaim some of this space that might otherwise be unused, it is possible to pack three real numbers into a 3-D vector and store the group into a single indirect register. This can save a tremendous amount of calculator memory. Storing 100 real numbers using indirect registers normally would use 3700 bytes. Packing them using the program in this learning module will only use 1/3 of that memory, which will then be available for other uses.

This program originally appeared in Datafile, a publication of HPCC. HPCC is a voluntary, independent body run by and for users of handheld and portable computers and calculators. The club has been helping members for more than 20 years to get the most from their Hewlett Packard equipment and to further the exchange of information and ideas. You can find out more about HPCC at their website http://www.hpcc.org/ .

The program listing. Program length is 338 bytes. Checksum C4F6. RDN is R↓ (Roll Down). All flag-related instructions (SF, CF, and FS?) are accessed through ↶ FLAGS. The conditional tests (x=0?, x<0?) are accessed through ↷ x?0. In several areas of this program, stack manipulations occur that look rather odd but manage to preserve the pre-existing stack contents. RPN mode is assumed in the program and throughout these instructions. The program uses one global label, variable register I, and flags 0, 1, 2, and 3.

Y001	LBL Y
Y002	CF 0
Y003	CF 1
Y004	CF 2
Y005	CF 3
Y006	x=0?
Y007	GTO Y062
Y008	x < 0?
Y009	SF 0
Y010	ABS
Y011	RDN
Y012	IDIV(REGT-1,3)
Y013	STO I
Y014	RDN
Y015	LASTx
Y016	ABS
Y017	RDN
Y018	RMDR(REGT-1,3)
Y019	x=0?
Y020	SF 1
Y021	FS? 1
Y022	GTO Y030
Y023	RDN
Y024	REGT-1
Y025	x=0?
Y026	SF 2
Y027	FS? 2
Y028	GTO Y030
Y029	SF 3

Y030	RDN
Y031	FS? 1
Y032	[1, 0, 0]
Y033	FS? 2
Y034	[0, 1, 0]
Y035	FS? 3
Y036	[0, 0, 1]
Y037	RCLx (I)
Y038	FS? 0
Y039	GTO Y058
Y040	+/-
Y041	RDN
Y042	XEQ Y063
Y043	RCL+ (I)
Y044	RDN
Y045	ABS
Y046	CLx
Y047	LASTx
Y048	RDN
Y049	XEQ Y063
Y050	RDN
Y051	REGZ+REGT
Y052	STO(I)
Y053	RDN
Y054	LASTx
Y055	LASTx
Y056	CLX
Y057	+
Y058	CF 0

Y059	CF 1
Y060	CF 2
Y061	CF 3
Y062	RTN
Y063	FS? 1
Y064	REGTx[1, 0, 0]
Y065	FS? 2
Y066	REGTx[0, 1, 0]
Y067	FS? 3
Y068	REGTx[0, 0, 1]
Y069	RTN
Y070	STO I
Y071	[0, 0, 0]
Y072	STO(I)
Y073	DSE I
Y074	GTO Y072
Y075	STO(I)
Y076	CLSTK
Y077	RTN

Usage Instructions:

1) Initialize the indirect registers to be used by providing the number of logical registers desired divided by 3 rounded up to the next highest integer. Then press XEQ Y070. For example, if you want 100 logical registers, give this routine 100 / 3, or 34 as an input. Note that you should probably keep at least 200-300 bytes free on the 35s.

2) To store a number, place the number to be stored in Y and the logical register location in X and press XEQ Y ENTER. Upon completion, the number just stored is in X. The original Z is now in Y and the original T is now in Z and T. LASTx is cleared.

3) To recall a number, place the logical register location to be recalled in X as a negative number and press XEQ Y ENTER. The recalled number is in X. The original contents of Y, Z, and T are undisturbed. LASTx contains the associated identity vector.

Entering program lines. How to enter some of the program lines might not appear obvious at first. Here are the key presses to place them into the program.

Line Y012: EQN ↶ INTG 2 R↓ ▷ ▷ ENTER − 1 ▷ 3 ENTER
Line Y018: EQN ↶ INTG 3 R↓ ▷ ▷ ENTER − 1 ▷ 3 ENTER
Line Y024: EQN R↓ ▷ ▷ ENTER − 1 ENTER
Line Y037: RCL × (i)
Line Y043: RCL + (i)
Line Y046: ↷ CLEAR 1
Line Y051: EQN R↓ ▷ ENTER + R↓ ▷ ▷ ENTER ENTER
Line Y056: ↷ CLEAR 1
Line Y064: EQN R↓ ▷ ▷ ENTER × ↷ [] 1 ↶ , 0 ↶ , 0 ENTER
Line Y066: EQN R↓ ▷ ▷ ENTER × ↷ [] 0 ↶ , 1 ↶ , 0 ENTER
Line Y068: EQN R↓ ▷ ▷ ENTER × ↷ [] 0 ↶ , 0 ↶ , 1 ENTER
Line Y076: ↷ CLEAR 5

Line by line analysis of the program. The description below explains what the program is doing in more detail. It may be of interest to see how the program operates.

Lines	What they do
Lines Y002 through Y005:	Reset flags.
Lines Y006 andY007:	Exits the program if an attempt is made to store into logical register zero, which is not supported. Lines Y008 and Y009 set flag zero if the logical register location is input as a negative, indicating a recall register input. Lines Y010 through Y013 store the indirect register number to be used into the I register.
Lines Y014 through Y018:	Determines the position in the 3-D vector where the value to be stored/recalled is found.
Lines Y019 through Y029:	Sets flag 1, 2, or 3, depending on the position within the 3-D vector for the value to be stored/recalled.
Lines Y033 through Y036:	Enters the appropriate identity vector.
Line Y037:	Extracts the proper value from the identity vector
Lines Y038 and Y039:	Exits the program if this is a recall entry.
Line Y040 and Y041:	Changes the sign of the extracted value and place it in stack register T.
Line Y042:	Calls a subroutine that creates an identity vector with the value of 1 in the vector replaced by the extracted value with its sign changed.

Line Y043:	Places in X the vector from the proper indirect register now with a zero in the location being replaced.
Lines Y044 through Y048:	These lines are stack manipulations to preserve the stack and store the previously extracted value into LASTx.
Line Y049:	Calls the subroutine that creates a vector with the proper position holding the value to be stored with the other locations holding a zero.
Line Y050:	Places this vector in T. At this point, register Z of the stack contains the original vector with a zero in the position where the value is to be stored, and register T of the stack contains a vector that has zeroes in the locations not being changed and the value being stored in the proper location within the vector.
Line Y051:	Adds these vectors in T and Z together and places the result in X.
Line Y052:	Stores this new vector back into the proper indirect register.
Lines Y053 through Y057:	Cleans up the stack so that the original value is in X, the original level Z is in Y, and the original level T is in Z and T.
Lines Y058 through Y062:	Cleans up the flags and exit the program.
Lines Y063 through Y069:	These lines are the subroutine called at lines Y042 and Y049.
Lines Y070 through Y077:	These lines are the initialization routine which stores vectors containing zeroes in the proper indirect registers.

Usage Examples. This program can be used in manual run mode or from within a program. The table below shows several examples of how it might be used. Since the program preserves the stack, usage does not require much special consideration, other than LASTx, as noted above. Program usage is the same as run mode usage – the routine is simply called as a subroutine.

Usage Examples	Keystrokes
1) Set aside 50 indirect registers. 50 divided by 3 is 17, rounded up to the next highest integer.	17 [XEQ] [Y] [0] [7] [0]
2) Store the following numbers:	
1.23456789 into logical register 10.	1.23456789 [ENTER] 10 [XEQ] [Y] [ENTER]
55 into logical register 7.	55 [ENTER] 7 [XEQ] [Y] [ENTER]
35.456565 into logical register 34.	35.455565 [ENTER] 34 [XEQ] [Y] [ENTER]
3) Compute the following:	
Logical register 34 divided by logical register 7.	34 [+/-] [XEQ] [Y] [ENTER] 7 [+/-] [XEQ] [Y] [ENTER] [÷]
Multiply logical register 7 by 3. Multiply logical register 34 by 5. Subtract the difference.	7 [+/-] [XEQ] [Y] [ENTER] 3 [×] 34 [+/-] [XEQ] [Y] [ENTER] 5 [×] [−]
Add logical register 10 to the result just computed.	10 [+/-] [XEQ] [Y] [ENTER] [+]

Saving keystrokes. Storing a number manually into an indirect register requires six key presses (↗ STO I ↗ STO (i)) while this program only requires four key presses, not counting the location and value to be stored which would be the same in both instances. In the example below, 15 is stored into indirect register 10. Manually, this requires 10 key presses. Using this program only requires 8 key presses.

Manually	Using this program
10 ↗ STO I 15 ↗ STO (i)	15 ENTER 10 XEQ Y ENTER

Recalling the same number from indirect register 10 takes 7 key presses manually and only 6 key presses using this program.

Manually	Using this program
10 ↗ STO I RCL (i)	10 +/- XEQ Y ENTER